A Visualization of Synthesized Person Trips on a Typical Weekday Throughout The Continental United States

Tharald S. Fongaard & Jacob Perricone

Advisor: Professor Alain Kornhauser

Submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Engineering

Department of Operations Research and Financial Engineering

Princeton University

May 2015

We hereby declare that we are the sole authors of this thesis.

We authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Jacob Perricone	Tharald S. Fongaard

We further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Jacob Perricone	Tharald S. Fongaard

Abstract

Given an enormous static dataset of synthesized travel activity for all individuals in the United States, we developed a robust data framework through which one can easily query, manipulate, and analyze any set of trips within the nation. This data was mapped using tableau software to display the aggregate travel activity for each state. Then each trip was linked to the United states road network, and the shortest path for each trip was found using dijkstra's shortest path algorithm. The result is an organized MySQL database that contains the location of every individual at every point in the day.

By analyzing trip generation in the United States, we will display key statistics in informative graphics. Specifically, given a node network of the entire nation and trip generation files, we will provide a means to determine the number of vehicles, people, and the average vehicle occupancy on each link in the network at each time of the day.

Contents

Abstract iii

List of Tables vii

List of Figures viii

1 Introduction **1**

2 Creating The Pixel Grid **3**

2.1 Pixel Structure and Algorithm Method 3

2.1.1 Geometric Analysis 3

2.1.2 Methodoloy 4

2.2 Uploading Data to the SQL Database 5

2.3 Visualizing the Grid 6

3 Creating The oTrip Files **8**

3.1 Wyrough's Data 8

3.2 The Process 8

3.3 Utility of oTrip Files 10

4 Creating The Network **11**

4.1 Data Source 11

4.2 Cleaning The Data 11

4.3 Assigning Entry Nodes 12

4.4 Weighting The Links 13

5 Creating The Path Files **14**

5.1 Creating The Framework 14

5.1.1 Path Finder 15

5.2 Dependencies 17

5.2.1 LinkObject 17

5.2.2 NodeObject 18

5.2.3 KdTreeST 19

5.3 Multi Threading 19

5.4 PathFinder Output 20

6 Visualizing The Data **21**

6.1 WebGL Globe Application 21

6.1.1 Creating The Data 21

6.1.2 Data Aggregation 22

6.1.3 Discussion and Results 22

6.1.4 Pictures 23

6.2 Tableau Visualization 23

7 Programming Choices **25**

7.1 Databases 26

7.1.1 Global 26

7.1.2 Path 27

7.1.3 Grid 28

7.1.4 oTrip 28

7.1.5 Database Statistics 28

7.2 Programming Languages 34

A Visualizations **35**

A.1 Total Cumulative Trip Miles in Continental United States 36

A.2 Nationwide Statistics 39

A.3 Statistics by State 40

B Code **186**

B.1 Java Code 187

B.1.1 Grid Creation 187

B.1.2 oTrip Generation 195

B.1.3 Path File Generation 197

B.1.4 Data Visualization 203

B.1.5 Database Assist & Data Structures 204

B.2 JavaScript & HTML Code 208

B.2.1 WebGL Globe Animation 208

B.3 Python & Bash Scripts 213

B.3.1 Network Data Cleaning 213

List of Tables

2.1 SQL Table Structure 5

7.1 EntryNodes Table Structure 26

7.2 Path Table Structure 27

A.1 Total Cumulative Trip Miles in Continental United States Midnight to Noon 36

A.2 Total Cumulative Trip Miles in Continental United States Noon to Midnight 37

A.3 Nationwide Statistics 39

List of Figures

2.1 The grid displayed on our OSM interface in TExas 6

2.2 The grid displayed on our OSM interface in Texas zoomed in 7

5.1 PathFinder Framework 15

5.2 KdTreeST graphical representation 19

6.1 WebGL Animation 23

A.1 Total Cumulative Trip Miles in Continental United States 38

A.2 Alabama: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 42

A.3 Alabama: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 43

A.4 Alabama: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 44

A.5 Arizona: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 45

A.6 Arizona: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 46
A.7 Arizona: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 47
A.8 Arkansas: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 48
A.9 Arkansas: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 49
A.10 Arkansas: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 50
A.11 California: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 51
A.12 California: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 52
A.13 California: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 53
A.14 Colorado: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 54
A.15 Colorado: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 55

A.16 Colorado: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 56
A.17 Connecticut: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 57
A.18 Connecticut: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 58
A.19 Connecticut: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 59
A.20 Delaware: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 60
A.21 Delaware: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 61
A.22 Delaware: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 62
A.23 Florida: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 63
A.24 Florida: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 64
A.25 Florida: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 65

A.26 Georgia: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 66
A.27 Georgia: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 67
A.28 Georgia: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 68
A.29 Idaho: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 69
A.30 Idaho: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 70
A.31 Idaho: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 71
A.32 Illinois: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 72
A.33 Illinois: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 73
A.34 Illinois: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 74
A.35 Indiana: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 75

A.36 Indiana: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 76
A.37 Indiana: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 77
A.38 Iowa: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 78
A.39 Iowa: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 79
A.40 Iowa: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 80
A.41 Kansas: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 81
A.42 Kansas: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 82
A.43 Kansas: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 83
A.44 Kentucky: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 84
A.45 Kentucky: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 85
A.46 Kentucky: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 86

A.47 Louisiana: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 87
A.48 Louisiana: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 88
A.49 Louisiana: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 89
A.50 Maine: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 90
A.51 Maine: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 91
A.52 Maine: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 92
A.53 Maryland: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 93
A.54 Maryland: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 94
A.55 Maryland: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 95
A.56 Massachusetts: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 96

A.57 Massachusetts: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 97
A.58 Massachusetts: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 98
A.59 Michigan: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 99
A.60 Michigan: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 100
A.61 Michigan: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 101
A.62 Minnesota: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 102
A.63 Minnesota: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 103
A.64 Minnesota: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 104
A.65 Mississippi: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 105

A.66 Mississippi: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 106
A.67 Mississippi: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 107
A.68 Missouri: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 108
A.69 Missouri: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 109
A.70 Missouri: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 110
A.71 Montana: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 111
A.72 Montana: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 112
A.73 Montana: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 113
A.74 Nebraska: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 114
A.75 Nebraska: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 115

A.76 Nebraska: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 116
A.77 Nevada: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 117
A.78 Nevada: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 118
A.79 Nevada: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 119
A.80 New_Hampshire: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 120
A.81 New_Hampshire: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 121
A.82 New_Hampshire: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 122
A.83 New_Jersey: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 123
A.84 New_Jersey: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 124

A.85 New_Jersey: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 125
A.86 New_Mexico: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 126
A.87 New_Mexico: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 127
A.88 New_Mexico: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 128
A.89 New_York: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 129
A.90 New_York: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 130
A.91 New_York: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 131
A.92 North_Carolina: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 132
A.93 North_Carolina: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 133

A.94 North_Carolina: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 134
A.95 North_Dakota: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 135
A.96 North_Dakota: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 136
A.97 North_Dakota: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 137
A.98 Ohio: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 138
A.99 Ohio: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 139
A.100 Ohio: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 140
A.101 Oklahoma: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 141
A.102 Oklahoma: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 142
A.103 Oklahoma: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 143

A.104 Oregon: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 144
A.105 Oregon: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 145
A.106 Oregon: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 146
A.107 Pennsylvania: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 147
A.108 Pennsylvania: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 148
A.109 Pennsylvania: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 149
A.110 Rhode_Island: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 150
A.111 Rhode_Island: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 151
A.112 Rhode_Island: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 152

A.113 South_Carolina: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 153
A.114 South_Carolina: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 154
A.115 South_Carolina: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 155
A.116 South_Dakota: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 156
A.117 South_Dakota: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 157
A.118 South_Dakota: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 158
A.119 Tennessee: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 159
A.120 Tennessee: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 160
A.121 Tennessee: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 161

A.122 Texas: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 162
A.123 Texas: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 163
A.124 Texas: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 164
A.125 Utah: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 165
A.126 Utah: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 166
A.127 Utah: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 167
A.128 Vermont: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 168
A.129 Vermont: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 169
A.130 Vermont: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 170
A.131 Virginia: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 171

A.132 Virginia: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 172
A.133 Virginia: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 173
A.134 Washington: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 174
A.135 Washington: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 175
A.136 Washington: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 176
A.137 West_Virginia: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 177
A.138 West_Virginia: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 178
A.139 West_Virginia: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 179
A.140 Wisconsin: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 180

A.141 Wisconsin: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 181
A.142 Wisconsin: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 182
A.143 Wyoming: Heat map of Radial Distance oPixel to Entry Node Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode 183
A.144 Wyoming: Trip Distance Distribution, Cumulative Distribution of Trip Miles PDF of Trip Departure Times, Cumulative Trip Departure Times 184
A.145 Wyoming: Heat Map of Trip Generation by Pixel Heat Map of Trips to dPixel 185

Chapter 1

Introduction

This thesis aims to visualize data from Wyrough?s thesis, which used stochastic techniques to generate a set of trip files that model the activity of all Americans throughout the day, thereby allowing the viewer to immediately gather information on trip generation and vehicle flow throughout the day. Although the data generated by our colleague is essential to our project, its form, a set of massive CSV files, which vary in size and total to around 115 gigabytes, is impossible to efficiently query or manipulate. Thus, before any real visualization progress could be made, the data that drives the visualization had to be restructured and stored in a database that could be easily queried and manipulated.

Expecting that the data generated by Wyrough will be subject to many forms of future analysis, ranging from ride-sharing to optimal repositioning of autonomous vehicles, we decided that it would be of great utility to store and reorganize the data into a more accessible form. Restructuring Wyrough?s data and storing it on an online SQL server is not only essential for our project but will allow future students to skip tedious data processing and move immediately into more fascinating problems.

Firstly, the data generated by Wyrough did not group trips into pixelated sets, which is instrumental if one aims to efficiently select trips within a geographic area. Thus a procedure to pixelate the United States needed to be established. This methodology provides a means of grouping trips that originate within a set of geographic coordinates into a distinct collection with a unique pixel id. More information on the exact procedure of pixilation is provided in chapter **??**.

The second step of data reorganization involved creating a set of State tables in a Grid database that stores all pixels that have centroids within a given state. Wyrough's data stores in each row an individual's travel activity. Since individuals enter and leave their origin states throughout the day, the trips in a given row needed

to be broken up according to their origin state. The State tables provide the means to do this, and thus allow us to transform Hill's data, which contain information grouped by the individual, into a set of oTrip tables that house all relevant information about each trip within each state. More information on the procedure used to generate the Grid database is provided in section **??**.

Chapter 2

Creating The Pixel Grid

The first step of the visualization required that the United States be broken up into pixels so that trips could be easily identified and tracked. Our analysis considers the 48 contiguous states, which together span around 3.8 million square miles. Each square pixel spans an area of .25 miles, implying that around 12 million pixels are needed to uniquely map the entire country. The center of the National grid is at 37°N Latitude, -97.50°W Longitude, placing the grid's centroid close to the geographic center of the continental US along the time meridian.

2.1 Pixel Structure and Algorithm Method

The primary motivation for creating a pixelated grid is to establish a simple method to track, store and retrieve trips within the pixel. Thus each pixel object must contain a unique id, an (i,j) coordinate, and the latitude and longitude of its four corners and its centroid.

2.1.1 Geometric Analysis

In order to algorithmically create a pixelated grid, a methodology of mapping latitude and longitude to (i,j) coordinates was established. Given that along a Meridian each degree of latitude traverses approximately 69.174 miles and that along a Parallel each degree of longitude traverses approximately $69.174 \cdot \cos(Latitude)$ one can map any latitude and longitude coordinate to an (i,j) pixel. To determine the set of (lon, lat) coordinates within a pixel, the angular height and width of each pixel as calculated. The Y-height of a pixel is 0.00722814 degrees Latitude and the X-width of a pixel is

0.00944344 degrees Latitude. Each (i,j) Pixel thus includes all points within:

$$YHeight = 0.00722814 \mid CenterLat = 97.5 \mid CenterLon = 37.0 \tag{2.1}$$

$$\frac{CenterLat + YHeight \cdot i}{\cos(CenterLon + YHeight \cdot j)} \leq Longitude < \frac{CenterLat + YHeight \cdot (i+1)}{\cos(CenterLon + YHeight \cdot (j+1)} \tag{2.2}$$

$$CenterLat + YHeight \cdot j \leq Latitude < CenterLat + YHeight \cdot (j+1) \tag{2.3}$$

The inverse of the equation above was used to map (lon, lat) coordinates to pixel coordinates and was imperative in constraining our algorithm to the US.

$$xPixel = i = floor(138.348 \cdot longitude + CenterLat \cdot \cos(latitude)) \tag{2.4}$$

$$yPixel = j = floor(138.348 \cdot latitude - CenterLon) \tag{2.5}$$

Where floor casts the decimal result as an integer, rounded down, and 138.348 is a correcting constant.

2.1.2 Methodoloy

The southwest and northeast corner of the US provided the initial and final (lon, lat) coordinates of our grid. The initial point (southwest corner) has a longitude of -120.5°and a latitude of -63.5°. The final point (northeast corner) has a longitude of -63.5°and a latitude of 48.9°. Using equation 2.4 and 2.5, the initial and final (i,j) coordinates corresponding to the corners of the nation were found and used as starting and breaking conditions during pixel construction. Then, running through a double for loop indexed by i and j, each (i,j) pair was sent to a Pixel class for creation and then added to a hashtable. The Pixel class takes an (i,j) coordinate in its constructor and using the equations in 2.1 determines the longitude and latitude of its four corners and centroid. Furthermore, in order to prevent collisions within the hashtable, each pixel creates a unique id by using its (i,j) coordinates as the input for the Cantor pairing function, which uniquely encodes two natural numbers into a single natural number. Since the Cantor pairing function only holds for positive

numbers, i and j are offset by the absolute value of the initial (i,j) point.

$$f_{ij} = (1/2)\cdot(i+|(InitialX+InitialY)|+j)\cdot(i+|(InitialX+InitialY)|+j+1)+j+|(InitialY)| \quad \text{(Cantor Function)}$$

A sketch of the initialization loop is shown below.

```
for i in range(iStart, iEnd):
   for j in range(jStart, jEnd):
            Pixel = createPixel(i,j)
            Pixels.add(Pixel.key(), Pixel)
```

2.2 Uploading Data to the SQL Database

Every pixel is added to a SQL database (mySQL). The database is located on a rackspace cloud platform and is divided into several sections. The pixels are added to the "Grid" database, which has a table for each state as well as a table to store pixels that fall outside of the borders of the united states. In the program, each state is stored as a polygon object in a kdtree mapped to its centroid. When a new pixel is created, polygons are returned in order of proximity to the state centroids. Each polygon is then queried until the polygon which contains the pixel centroid is found. This polygon is then queried for its state name, which coresponds to the table to which the pixel is uploaded. The pixels are added with a runtime of about 80 pixels per second per thread, which allows us to create about 640 pixels per second while multithreading on two quad core computeres with sufficient heap space. The total creation time for a grid of about 18 million pixels is then about 8hrs. The table structure is shown in the table 7.1. The table is updated with a java database connector (JDBC) package.

Table 2.1: SQL Table Structure

ID	X	Y	C_Long	C_Lat	BL_Long	BL_Lat	BR_Long	BR_Lat	TL_Long	TL_Lat	TR_Long	TR_Lat
20172271942	2694	848	-70,81	43,13	-70,82	43,13	-70,81	43,13	-70,82	43,14	-70,81	43,14
20172271943	2693	849	-70,82	43,14	-70,83	43,14	-70,82	43,14	-70,82	43,14	-70,81	43,14
20172472799	2698	845	-70,78	43,11	-70,79	43,11	-70,78	43,11	-70,78	43,12	-70,78	43,12
20172472800	2697	846	-70,79	43,12	-70,79	43,12	-70,78	43,12	-70,79	43,12	-70,78	43,12
20172472801	2696	847	-70,80	43,13	-70,80	43,12	-70,79	43,12	-70,80	43,13	-70,79	43,13
20172472802	2695	848	-70,80	43,13	-70,81	43,13	-70,80	43,13	-70,81	43,14	-70,80	43,14

Table 7.1 is an extract from Grid.Maine on our rackspace account.

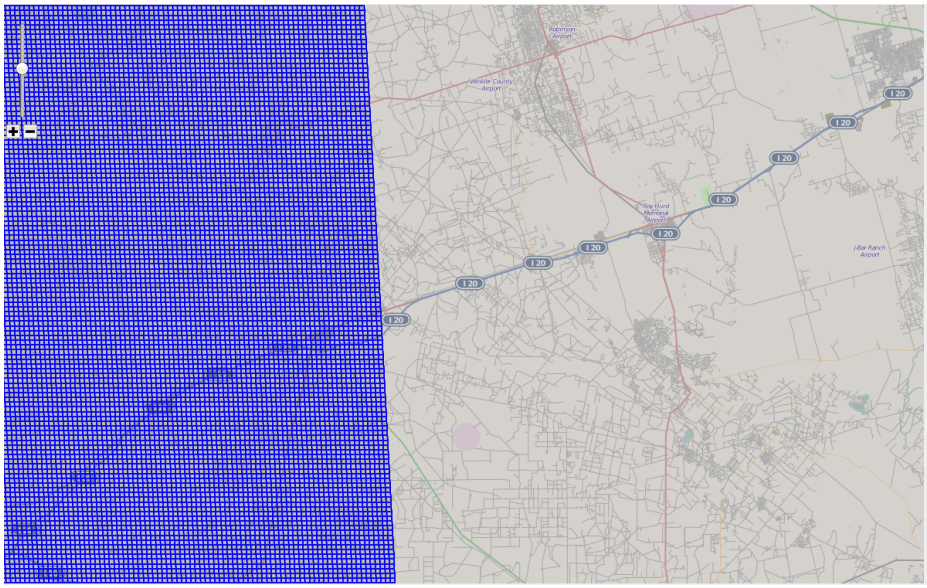

Figure 2.1: The grid displayed on our OSM interface in TExas

2.3 Visualizing the Grid

The grid can be visualized through our Open Street Map Java Interface. In figure 2.1 the grid is downloaded from the server and displayed as a layer on top of the map.

When zoomed, the properties of the grid is displayed. The pixels are increasingly non-square as one moves away from the (0,0) pixel.

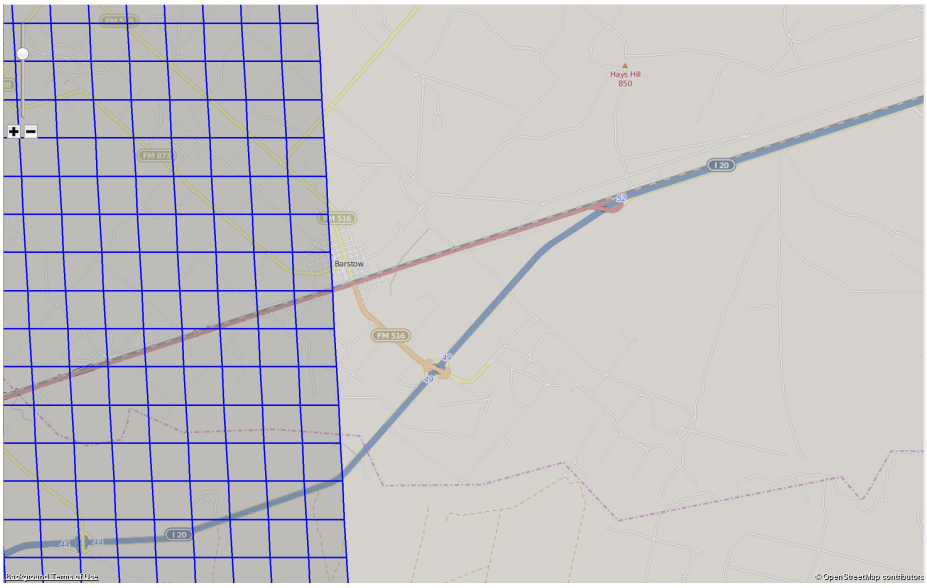

Figure 2.2: The grid displayed on our OSM interface in Texas zoomed in

Chapter 3

Creating The oTrip Files

3.1 Wyrough's Data

Each row in Wyrough?s data models the activity of one individual throughout the day and is identified by a unique Person ID. Each trip represents a node in an individual?s travel network. There are up to seven nodes in each row.

The first node in the network always has type H, indicating that each individual begins his travel for the day from home. The other trip types, O and H, signify trips departing from Other and Work, respectively, where Other denotes all recreational trips. Each node also stores the type of its predecessor and successor as well as important geographical data. Wyrough's model assumes that the last trip of the day always returns home, and thus the last trip of the day is not explicitly stated within the data. In order to account for this when programmatically generating the oTrip tables, all destination information for the last node was taken from the first node in the row. Since all of Wyrough's data is not essential for our analysis, the Block Code, Activity Pattern, County Code and House Hold ID was ignored.

3.2 The Process

A java program that iterates through all lines within a file, extracts the necessary information from each row, and uploads the data into the correct oTrip SQL table was created. A second bash script was created that iterates through each State Directory in Wyrough's dataset and calls the aforementioned java program for each file in the directory. Bear in mind that the data restructured by our program is over 100 gigabytes, thus creating the oTrip tables required a large amount of memory and

incredibly processing speed. So much so that if it were run on a local computer it would take over a month to finish, given the computer does not crash. The enormity of the dataset required that we multithread our program on a rackspace server with more memory and processing power. A further outline of our full-stack programming setup can be viewed in programming choices.

As outlined in section 7.1 the MySQL database contains a set of State tables within the Grid database, each of which contains the ids of all pixels that fall within the state. To increase program simplicity, all of the Grid tables were aggregated using SQL queries into a two-column global Info table that recorded the pixel id and the name of the state to which the pixel belongs.

Given variations in the data, the code was constructed to modify its execution based on the file's input. For example, if an individual does not take seven trips throughout the day, not all columns are populated with meaningful data. To compensate, the program, upon detecting illogical input, stops reading data and progresses to the next trip. Less trivially, some rows contained columns with erroneous latitudes and longitudes that did not fall within the United States. In some cases, the whole row contained absurd coordinates, whereas, for others, only a few nodes had nonsensical information. To compensate for the latter case, the program scanned the row skipping nodes with invalid data until an adequate destination coordinate was found. The origin node and the destination node were then linked. If the origin location was the first node in the row and no subsequent nodes had adequate coordinates, the whole trip was discarded.

A java class, Pixel Generator, was created that takes a file as input. The constructor then calls a private function that iterates through each line of the file, parses the line, and uploads the necessary information to the database. Each line, referred hereafter as an individual's travel network, was split into an array of strings. The person id given by Wyrough's data was stored for future use. If the first node in the network had an erroneous latitude or longitude coordinate, the trip was discarded and the program proceeded to the next line of the file. The code then iterated through each node in the travel network using an index variable to reference the correct position within the travel network array. Additionally, a trip-number variable, TN, was created to keep track of how many trips the individual actually takes. In order to ensure that the program terminates at the end of an individual's trip network, and thus does not reference an erroneous cell, the code checks whether the current node type or the successor's node type is N or NA, breaking out of the loop if the condition is true. In order to query the database, a unique trip id was created for each node

in the row by dividing the trip number by ten and adding the result to the person id variable.

The equations detailed in **??** were used to convert the geographic coordinates (Lat, Long) of the origin and destination nodes to pixelated coordinates (x,y). Using the cantor pairing function, the pixel coordinates were mapped to distinct pixel ids that could be used to query the Global Info table for the correct oTrip table. After fetching the state name from the global table, the trip information was uploaded to the correct oTrip table. If the pixel falls between a state, or is mapped to a location in the ocean, the Trip is put into the NoState table, which harbors all outliers. The table schema is displayed below.

3.3 Utility of oTrip Files

With the data in the oTrip tables, one can access, manipulate, and analyze Wyrough's data across a variety of platforms ranging from web applications to desktop visualization software. The trips, though now categorized by geographic origin, still retain the original data, that is, each trip can still be mapped to a person's travel network. The new structure, however, distances the client from the massive data files and allows for subsets of trips to be quickly selected. With this flexibility, one can distill key statistics, such as average trip distance or the most active pixel, with much greater efficiency. Furthermore, the uniformity of structure within the oTrip tables greatly enhances the developer's ability to create informative and flexible graphics, ranging from heat maps of trip generation to cumulative distributions of trip distances by pixel, super-pixel, or state. Moreover, each of these graphs can be customized by time of day. The most difficult visualization task is to link the oTrip data to a road network to visualize traffic flow. In order accomplish this goal, one needs a method of mapping latitude and longitude (or, better yet, pixels) to a node on the network and an algorithm to calculate the shortest path on the network to the destination pixel.

Chapter 4

Creating The Network

4.1 Data Source

A dataset containing a network of the United States road system was provided by Mike Bodden, the Vice President of Technology at ALK Technologies Inc. The dataset consisted of two files. One file contained a list of node latitude and longitude coordinates separated by spaces. The total number of nodes in the network is 497,712. The second file, the link file, contained six space-delimited fields: Node A number, Node B number, the distance of the link, the link speed limit, the link direction, and the link name. The total number of links in the network is 1,058,557.

4.2 Cleaning The Data

In order to convert the raw data into a usable csv file, two python script were created to clean the files. The node file did not contain decimal points in either the latitude or longitude fields. Thus one script, nodeclean.py, added decimal points, node numbers, and commas to the data file. For the link file, each line that contained *Skip* had to be ignored and all of fields had to be comma delimited. Since not all of the rows contained entries for all the fields and MySql tables cannot be of variable column length, the program, linkclean.py, had to ensure that all empty fields were replaced by comma separated empty strings. A python script later, and the link file was now in a usable form. Uploading the two groomed files to the Global database, the data for the US Travel network was now in place.

4.3 Assigning Entry Nodes

To route all oTrip files onto the network, it was necessary to assign an *entry node* for each pixel. To do this, every node in the network was first put in a 2d-tree, which stores the nodes as key value pairs, where the key is the geographical location of the node and the value is the node id number. The algorithm used for this process is found in appendix B.1.3. Once every node was stored in the 2d-tree, the algorithm runs through each state table in the oTrip database[1] and runs the following SQL query though the JDBC connector:

```
SELECT oTrip.State.oPixel, Grid.State.C_Long, Grid.State.C_Lat
FROM oTrip.State
LEFT JOIN Grid.State ON oTrip.State.oPixel = Grid.State.ID
GROUP BY oPixel
```

This query finds all oPixels being used in the current state for the purposes of reducing the running time of the entry node algorithm. Once the algorithm knows all unique pixels being used, it finds the closest node to the pixel by utilizing the kd-tree method *nearest*. The *nearest* method runs through the kd tree in a red black BST fashion[2], reducing the query area by half for every iteration. The kd tree algorithm used in this model can be found in appendix B.1.5. After the nearest node for a pixel is found, it is stored in the database Global under in table EntryNodes[3], where geographical location of both the pixel and its corresponding entry node is stored. The algorithm can be simplified as shown below.

```
for node n in Nodes:
     put node location, id in kd-Tree
end
for state s in States:
     Find all unique pixels P in s
     for p in P:
               store kd-Tree.nearest(p) in Global.EntryNodes
               store associated data in Global.EntryNodes
     end
end
```

[1]See chapter 7.1 for an overview of the database structure
[2]The red black BST is a famous data structure developed at Princeton University (**?**)
[3]See chapter 7.1 for an overview of the database structure

4.4 Weighting The Links

Because the Dijkstra's algorithm used in chapter 5 is run as a *weight minimization*, every link in the network had to be stored with a weight. The two most regularly used weight parameters used for Dijkstra's are *time* and *distance*. In the model developed in this thesis, *time* is used to weigh the links. To calculate the expected travel time for a link, the speed limit of the link in mph is simply divided by the length of the link un miles, and then multiplied by 3600 to get the expected travel time for a link in seconds.

$$E[Time(link)] = \frac{Distance(link)}{Speed(link)} 3600 \tag{4.1}$$

In the network used in this model, there are eight speed limits, which are classified with the road type parameters 1 through 8. The classifications used for the speed limit Ω given road type θ are shown in equation 4.2. The road types are the following:

1. Interstate
2. Special Interstate
3. Divided Highway
4. Arterial highways
5. Ramp
6. Slow local road
7. Good local road
8. Other

$$\Omega = \begin{cases} 65, & if\ \theta = 1 \\ 65, & if\ \theta = 2 \\ 55, & if\ \theta = 3 \\ 40, & if\ \theta = 4 \\ 5, & if\ \theta = 5 \\ 30, & if\ \theta = 6 \\ 35, & if\ \theta = 7 \\ 20, & if\ \theta = 8 \end{cases} \tag{4.2}$$

Chapter 5

Creating The Path Files

In essence the path database records the location of every individual in the United States throughout the day. In effect, it holds the data to draw any set of trips on the network. From it one can determine the number of miles traveled along each segment of a trip, record the arrival time at each node of a trip, plot an individual's travel activity throughout the day, and even determine the number of trips on a given link in a period of time. The path files contain information on what exact roads in the network are used for each trip, as well as the corresponding entry and arrival times for each link. The potential utility of such a dataset is huge, as it contains information that can be used for road planning, flow simulation, autonomous taxi positioning, traffic control and a number of other applications. Note: the arrival times at each node are calculated using network data and are not dependent on the data generated by Wyrough 3.

5.1 Creating The Framework

The framework used to create the path files is developed as a result of the available data (chapter 3), the underlying network (chapter 4), as well as available hardware and our programming choices (chapter 7). The framework was designed to run efficiently on a 8vCPU, 30GB RAM, 1.2TB SD HD Ubuntu server through a java virtual machine.

Java was chosen as a programming platform as it allows for an efficient integration of all the platforms that contains the necessary data. In order to create path files, it was necessary to build a program that could combine data resources such as MySQL and XML through fast and memory efficient data structures and algorithms.

The top level java program created by the authors, *PathFinder.java* combines all

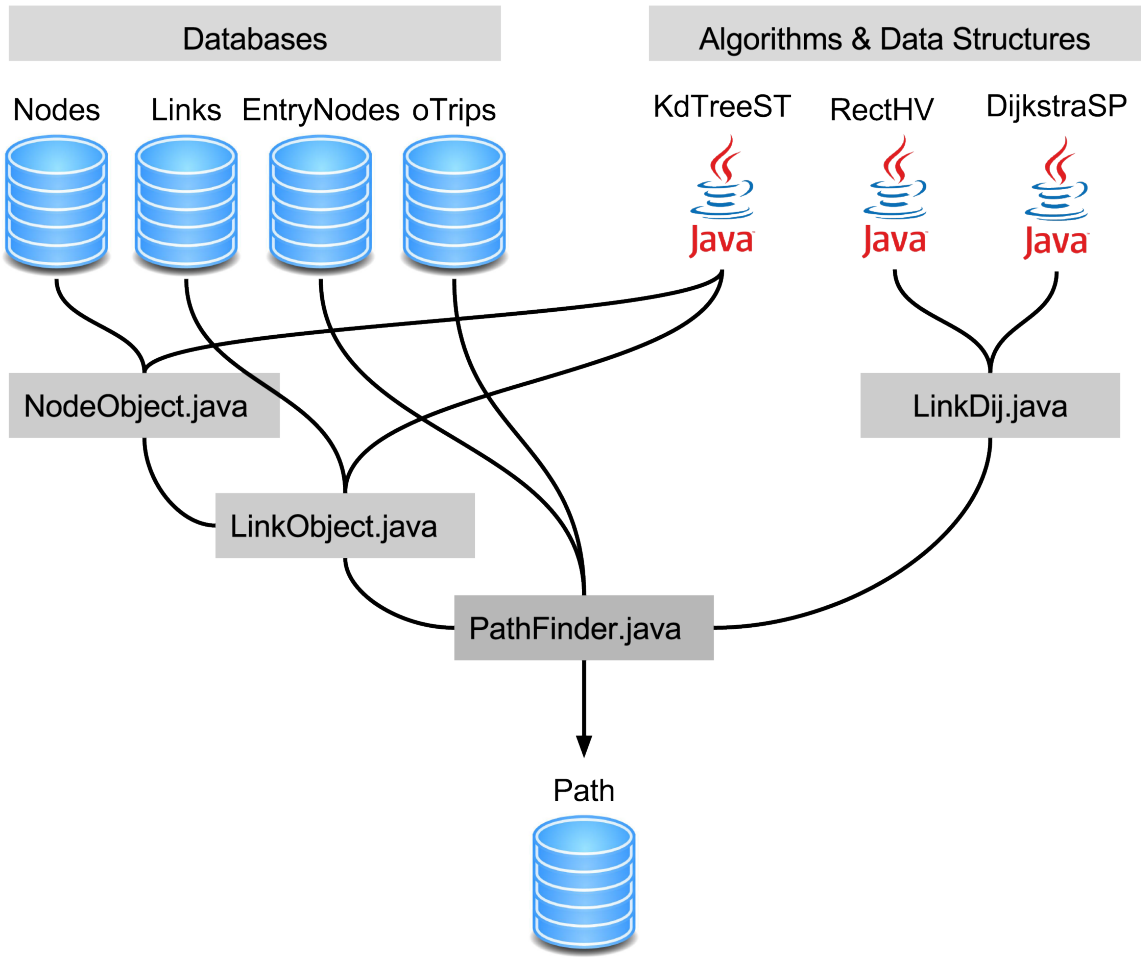

Figure 5.1: PathFinder Framework

of the above resources through a object oriented java interface, open source database drivers, Princeton University java libraries and proprietary data structures implemented by the authors. The top level java file and all its dependencies can be found in appendix B. The framework is shown in figure 5.1.

5.1.1 Path Finder

The PathFinder program was designed to run as a multithreaded process, and therefore takes as input a *from* and *to* node ID.The multithreading process is explained in section 5.3. PathFinder then opens a JDBC connection to the Global database and fetches all the unique pixels that have trip originations as well as their associated entry nodes. This information is fetched from the EntryNodes table[1]. Once Pathfinder

[1]The algorithms used to find the entry nodes are explained in section 4.3

has fetched all the unique oPixels and their entry nodes, the program sorts the pixels by node, then pixel. The fetching, multi-thread division and sorting is done server side through MySQL, and so this process does not consume memory or CPU time allocated to the forked java process. The query sent to to MySQL server is shown below.

```
SELECT Pixel, EntryNode
FROM EntryNodes
WHERE (EntryNode > 'from' AND EntryNode < 'to')
ORDER BY EntryNode, Pixel
```

Once PathFinder has fetched the necessary oPixel and EntryNode data from the database the program creates a LinkObject. The LinkObject implementation is explained in section 5.2.1 and the code can be found in appendix B.1.3. PathFinder then proceeds to create a new Dijkstra object by fetching all the links within a 250x250 mile square surrounding the source node. The distance calculation is made by moving 250 miles along Earth's surface in the SW and NE direction, and creating a square with the corresponding xmin, xmax, ymin and ymax values. The distance calculation formula used in in Pathfinder.java is shown in equation 5.1[2]

$$
\begin{aligned}
\phi_2 &= asin[sin(\phi_1) \cdot cos(\varrho) + cos(\phi_1) \cdot sin(\varrho) \cdot cos\theta \\
\lambda_2 &= \lambda_1 + atan2(sin(\theta) \cdot sin(\varrho), cos(\varrho) - sin(\phi_1) \cdot sin(\phi_2)) \\
\phi &: Latitude \\
\lambda &: Longitude \\
\theta &: Bearing\ clockwise\ from\ north \\
\varrho &: Angular\ distance \frac{d}{R} \\
d &: Distance\ traveled\ =\ 250\ miles \\
R &: Radius\ of\ the\ earth\ =\ 3963\ miles
\end{aligned} \tag{5.1}
$$

With the calculated boundary surrounding the source node, PathFinder retrieves an iterable of DirectedEdges from LinkObject.java, which in turn keeps track of the data structures containing the geographically sorted key, value pairs of edges in the network. PathFinder then passes the source node and its associated iterable of DirectedEdges to LinkDij, which returns to PathFinder a DijkstraSP object. The result

[2]Source: http://www.movable-type.co.uk/scripts/latlong.html

of this process is that the PathFinder only in total fetches as many Dijkstra Objects as there are unique entry nodes, resulting in the most memory efficient and the fastest possible runtime given the size of the underlying dataset. The execution sequence described above was developed as a result of runtime analysis of different implementations PathFinder's order of execution, which showed that the most time and memory consuming part of the execution was related to creating new DijkstraSP objects. The pseudo code related to this execution phase can be simplified as below:

```
connect to oTrip
get all trips T
for (trip t in T)
     if(entry node trip[t] != entry node  trip[t-1])
          toss old DijkstraSP
          fetch new DijkstraSP
     else
         reuse prev DijkstraSP

     get path L to dest node
     for(link l in L)
          store trip segment in Path.State
```

5.2 Dependencies

5.2.1 LinkObject

When initiated, LinkObject connects to Global.Links[3] and stores them in a KdTreeST data structure, with the geographical location of the aNode of the link as the key, and a bag og all links starting at node A as the value. This allows the LinkObject to fetch DirectedEdges within geographical boundaries in O(log n) time, maximizing the efficiency of the searching part of the DijkstraSP initiation process, as well as reducing the memory footprint and runtime of the DijkstraSP object it provides data for. part of the execution was related to creating new DijkstraSP objects.

LinkObject also acts as a gatekeeper for NodeObject.java, and uses the data contained in NodeObject to get information on the nodes associated with its links. If PathFinder queries LinkObject for node information, LinkObject returns data from

[3]See chapter 7.1 for an overview of the database structure

its associated NodeObject. This execution sequence was implemented to reduce the total memory footprint of PathFinder, by using one NodeObject instead of two.

LinkObject contains a series of public methods to allow PathFinder to access its private data structures.The pseudo code related to LinkObjects initiation sequence can be simplified as below:

```
create empty KdTreeST tree
create new NodeObject
connect to Global.Link
get all links L
for (link l in L)
     if(l.aNode already exists in tree)
          fetch bag B linked to key
          put link in B
          tree.put(l.aNode, B)
     else
         create new bag B
         put link in B
         tree.put(l.aNode, B)
```

5.2.2 NodeObject

When initiated, NodeObject connects to Global.Nodes[4] and and stores them in a KdTreeST data structure, with the geographical location of the node as the key, and its integer id as value. NodeObject contains a series of public methods to allow LinkObject to to access its private data structures.The pseudo code related to LinkObjects initiation sequence can be simplified as below:

```
create empty KdTreeST tree
connect to Global.Node
get all nodes N
for (node n in N)
      tree.put(node, node.ID)
```

[4]See chapter 7.1 for an overview of the database structure

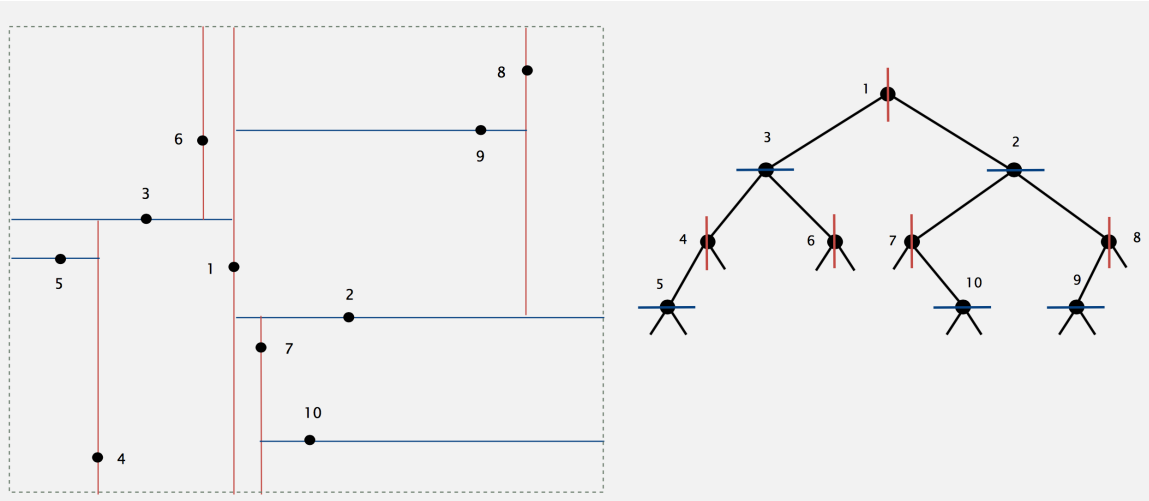

Figure 5.2: KdTreeST graphical representation

5.2.3 KdTreeST

The KdTreeST data structure used in this model is based on the KdTreeST implementation made by the authors in COS 226[5]

The implementation stores data in key value pairs, where the key is a point with x coordinate equal to a longitude and a y coordinate equal to a latitude. To search trough the data structure, the tree compares a query point with the x coordinate of the root node. The tree then alternates between comparing the query node with the y and x coordinates of the root of the subtree returned.

The result is that for each comparison, the tree becomes on average half as big. A graphical representation of the KdTreeST implementation used in this framework is shown in figure 5.2. The KdTree data structure is used in the PathFinder framework to store both links and nodes by their location.

5.3 Multi Threading

As the execution sequence of PathFinder is sorted by entry node, its multithreading is also constrained by node numbers. When PathFinder is executed server side, it is forked by every 24000 entry nodes. The result of this is 20 parallel process running on its own oTrip datasets containing as many trips as the node sequence it was assigned holds. (Each entry node is mapped to multiple oPixels.

[5]COS 226 is a class on Algorithms and Data Structures at Princeton University

The multi threading capabilities of were developed to allow maximum usage of the 8vCPU, 30GB RAM, 1.2TB SD HD Ubuntu server.

5.4 PathFinder Output

Pathfinder writes its output to the MySQL database Path, which is partitioned by state. The format of the output is shown in chapter 7.1.2, 27.

The columns in the Path tables are the following:

- ID
- PersonID
- Segment
- aNode
- bNode
- LinkID
- LinkName
- oPixel
- dPixel
- dTime
- aTime
- oType
- dType
- a_Lat
- a_Long
- b_Lat
- b_Long
- aNode_Time
- bNode_Time

The output deviates very little from trusted shortest path algorithms such as Google maps. In addition, the travel times on each link are correspond to expected travel times both in terms of distance, and roadtype.

Chapter 6

Visualizing The Data

6.1 WebGL Globe Application

WebGL is javascript framework to create interactive 3D maps. Using WebGL, Trip Generation Intensity was drawn as vertical bars on a 3D globe. The web-application is hosted at: http://162.209.99.128/globe/. Make sure to allow the page to load before hovering over each time segment (Chrome is recommended for WebGL applications).

WebGL is built upon a larger javascript framework called three.js. The general recipe for creating a three dimensional globe is to create a series of three dimensional meshes, style them with custom shading, add them to a three.js scene object, assign a camera object to each scene, and use WebGL's rendering platform to take care of the visualization. The camera object essentially controls the user's view of the scene objects, thus giving them their three dimensional feel. Once the object class is created, the client simply instantiates the class and adds points to the globe via a JSON file that details the latitude, longitude, and magnitude of each point. Finally, the client animates the globe, which renders it on the screen. The globe object class, the html file, and the JSON file can be found in the appendix under globe.js and index.html.

6.1.1 Creating The Data

In order to supply the data for the globe, a java program CreateJson.java was created that iterates through each state in the oTrip table, fetches all pixels that have trips within a given time period, sums the number of trips within each pixel, divides the sum by a constant to create the magnitude, and, for each pixel, prints out the coordinates and the magnitude in the proper format.

Given the enormity of the dataset, pixels were aggregated into 50x50 hyper-pixels and the time increment was expanded to three hours. The initial goal was to provide smaller increments of time for the visualization; however, if the time increments or the hyper-pixels were any smaller the web-application experienced functionality loss due to latency caused by the size of the dataset.

6.1.2 Data Aggregation

After querying the database for all trips within a state and ordering by the (X,Y) coordinates, the programs enters a while loop that iterates through all rows of the result set, summing the trip magnitudes for all the pixels within the hyperpixel. An integer counter keeps track of the number of pixels iterated through and resets the magnitude to zero every time a hyperpixel is finished. The counter is initialized to one in order to prevent infinite values for the corner case where the result set is empty. The centroid of the hyper-pixel was used for the latitude and longitude data. The most difficult aspect of the program was catching all of the corner cases and formatting the print statements correctly. This, however, is not of tremendously interesting, and so the reader is left to peruse the code at the appendix.

```
Pseudo-Code:
for(t=0;t<=24;t= t+3)
     for( state: XML.getallStates())
     counter = 1
     resultSet = SQL.query(?Query?)
          while (resultSet.next())
               if (j < 2501)
       magnitude = magnitude + resultSet.get(index)
       *Catch Corner Case 1:
        end of Result Set and end of all states, print
        Catch Corner Case 2:
        End of Result Set Corner case, print

        else
                *Case 1: the last state in the while loop and last value in the resultSet, calculate centroid, print
                *else: Calculate centroid, print,
                 magnitude = 0;
                 *get next row of result set, increment j;
                  magnitude = resultSet.get(index);
                  j = 2;
```

6.1.3 Discussion and Results

The most active region by far is the northeast, and the most active time period appears to be from 6:00-9:00 p.m. One also may notice that California seems rather sparsely populated. The reason for this is that Wyrough's data in fact did not include any files for California. The trips that are within California had their first trip of the day in another state.

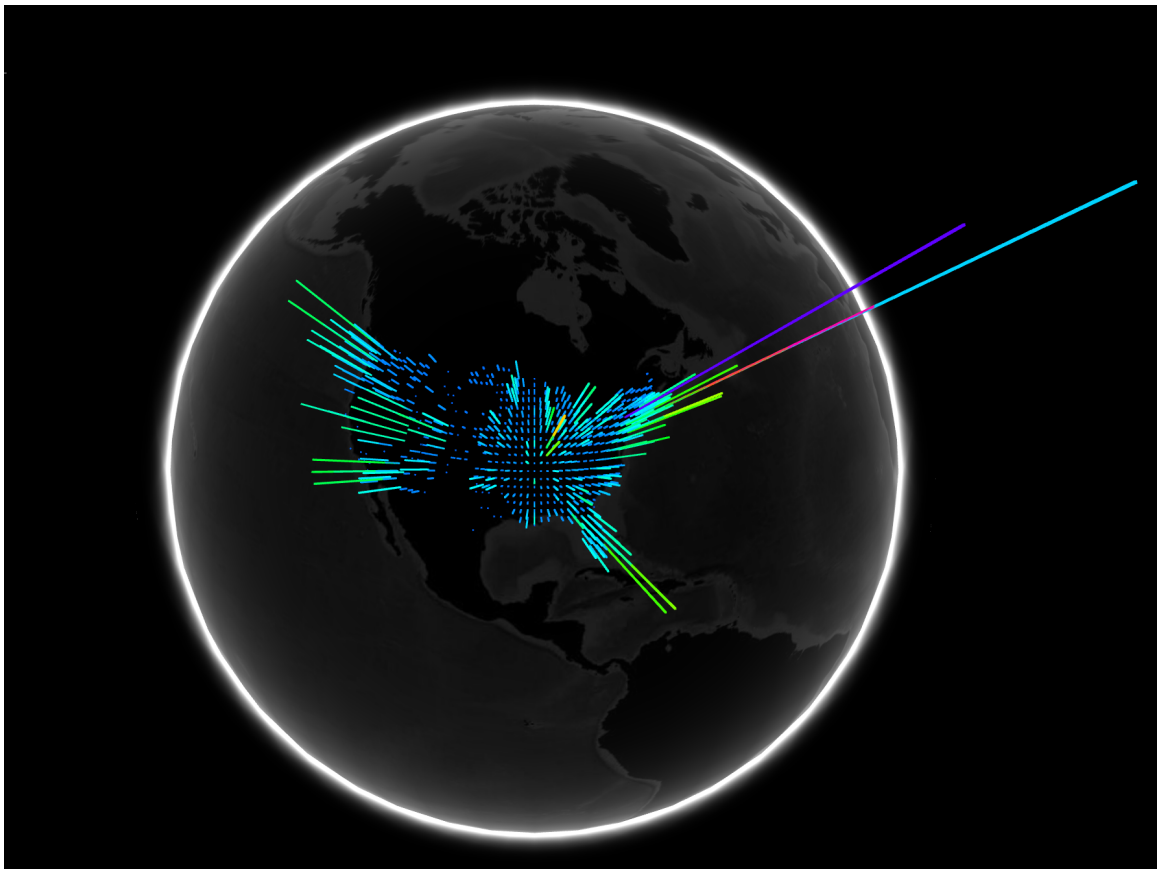

Figure 6.1: WebGL Animation

6.1.4 Pictures

6.2 Tableau Visualization

The initial aim was to create a javascript application similar to what Mr. Fongaard created for New Jersey last spring. However, given the size of the dataset, rendering it on a browser appeared impossible. In order for Mr. Fongaard to draw the trips on a google map interface, the data for each possible combination of origin and destination pixel had to be hashed to a database. Each county's hash combination required 2.2 megabytes of space, with all possible hash combinations for the county requiring around 9 gigabytes of data. Given that the nation contains around 3000 counties and over ten times as many hash combinations, the data would need to be stored on a 27 terabyte server with lightning network speed. Given the processing constraints, it proved more realistic to process the data on Tableau Desktop, a data processing and visualization platform, and then publish the finish work sheets to a Tableau Server

that can be accessed by user ID and password.

Before displaying the visualizations, it is important to note some filtering techniques used in order to eliminate errors in Wyrough's data. A fraction of the trips' departure times and/or arrivals times were negative values or values well above 86,400 (the number of seconds in a day). Given that no times should fall outside this range, the data was filtered to remove these values. Although the error is troubling, the majority of the trips generated by Wyrough fall within a valid time range and thus removing these trips from our visualization does not drastically impact the results.

Chapter 7

Programming Choices

The sheer size of the data our programs have process required that we think strategically and long term when we set up our software stack. In order to lay the groundwork for future work and eliminate the use of hefty csv files, it was essential that the data and the programs we created were stored and ran on a remote server. We chose to use a rackspace Ubuntu Server since it has a linux operating system and allows for easy installations of crucial packages, such as apache tomcat, java, jclouds, MySql, and Apache 2. Apache Tomcat is a collection of apache services, such as the Apache 2 webserver and the Jasper JSP package, that give the server a robust library of software to build upon. Apache 2, for example, allows us to host the WebGL application, and jclouds allows us to run code on the cloud. Furthermore, one is able to take an image of an Ubuntu server and rebuild it on an upgraded version. This proved crucial since it allowed us to upgrade our server when we encountered speed and memory issues throughout our research.

Our initial server had a 1 CPU core, 1GB of ram, a 20GB system disk, and network speed of 200 Megabits per second. Since the data we are working with is well over 200GB, we loaded a storage volume of 300GB to MySql on the server, so that when MySQL queries the server for storage, the server allocates the extra 300GB disk storage for its task. However, when we set out to create the oTrip files, we realized that we severely underestimated the processing intensity of the task. With the basic server setup, it would have take over a month to only create the oTrip files! After running a series of test we observed that our bottleneck occurred between the attached storage volume and the server: the I/O capabilities between the server and the attached volume just did not cut it.

To fix the issue we upgraded our server to have 8 CPU cores, 30GB of ram, 1.2 terabytes of system disk, and a network speed of 1.2 Gigabits per second. With the

upgraded server, we were able to move the MySQL database onto the system disk and thus house all data on the server. After the reconfiguration, the oTrip data was process in a matter of days, rather than weeks. Nevertheless, the enormity of the data proves to be an issue every step of the way, since the result sets of queries are often millions of rows long!

7.1 Databases

7.1.1 Global

EntryNodes

The aim of the EntryNodes table is to provide a convenient mechanism to link a trip to a node on the network. Since each trip is mapped to a .5x.5 miles pixel, the Pixel ID of a trip (given in the oTrip table) provides the perfect primary key to link trips to entry-nodes. One can then easily fetch all pixels corresponding to an entry-node, get all the trips corresponding to those pixels, retrieve the destination pixels of those trips, retrieve the exit nodes from the destination pixels, and run a shortest path algorithm along the network. The latitude and longitude of both the entry-node and the centroid of the pixel are included so that one can analyze the distribution of how far entry-nodes are from the pixel that generated the trip.

Table 7.1: EntryNodes Table Structure

Pixel	EntryNode	Pixel_C_Lat	Pixel_C_Long	Node_Lat	Node_Long
18889363902	25178	34.474	-120.291	34.486	-120.228
18889947034	24028	34.654	-120.534	34.655	-120.537
18890141398	24557	34.604	-120.450	34.629	-120.433
18890918897	24315	34.640	-120.469	34.639	-120.476
18891113273	24389	34.640	-120.460	34.639	-120.458
18891113274	24315	34.647	-120.471	34.639	-120.476
18891307650	24437	34.640	-120.451	34.639	-120.449
18891307655	24218	34.676	-120.505	34.678	-120.497
18891502025	24557	34.618	-120.410	34.629	-120.433
18891502028	24455	34.640	-120.442	34.639	-120.445

7.1.2 Path

Table 7.2: Path Table Structure

ID	PersonID	Segment	aNode	bNode	LinkID	LinkName	oPixel	dPixel	dTime	aTime	oType	dType	a_Lat	a_Long	b_Lat	b_Long	aNode_Time	bNode_Time
16000485278	16000485278	0	-1	49505	14382	Tekoa Farmington Road	19381559415	19395146779	41722	44540	H	O	47.161	-117.040	47.168	-117.070	41722	41722
16000485278	16000485278	1	49505	49526	14382	Tekoa Farmington Road	19381559415	19395146779	41722	44540	H	O	47.168	-117.070	47.204	-117.069	41722	41967
16000485278	16000485278	2	49526	49544	14381	Tekoa Farmington Road	19381559415	19395146779	41722	44540	H	O	47.204	-117.069	47.221	-117.068	41967	42083
16000485278	16000485278	3	49544	49459	14380	Park Street	19381559415	19395146779	41722	44540	H	O	47.221	-117.068	47.221	-117.071	42083	42096
16000485278	16000485278	4	49459	49409	14379	Park Street	19381559415	19395146779	41722	44540	H	O	47.221	-117.071	47.221	-117.073	42096	42106
16000485278	16000485278	5	49409	49408	14370	WA-27	19381559415	19395146779	41722	44540	H	O	47.221	-117.073	47.227	-117.073	42106	42141
16000485278	16000485278	6	49408	49418	14371	WA-274	19381559415	19395146779	41722	44540	H	O	47.227	-117.073	47.227	-117.073	42141	42142
16000485278	16000485278	7	49418	49482	14372	WA-274	19381559415	19395146779	41722	44540	H	O	47.227	-117.073	47.227	-117.070	42142	42153
16000485278	16000485278	8	49482	49793	14373	WA-274	19381559415	19395146779	41722	44540	H	O	47.227	-117.070	47.240	-117.040	42153	42311
16000485278	16000485278	9	49793	50627	14374	ID-60	19381559415	19395146779	41722	44540	H	O	47.240	-117.040	47.229	-116.931	42311	42807
16000485278	16000485278	10	50627	50695	14416	US-95	19381559415	19395146779	41722	44540	H	O	47.229	-116.931	47.284	-116.916	42807	43073
16000485278	16000485278	11	50695	50804	14415	US-95	19381559415	19395146779	41722	44540	H	O	47.284	-116.916	47.318	-116.893	43073	43258
16000485278	16000485278	12	50804	50813	14414	US-95	19381559415	19395146779	41722	44540	H	O	47.318	-116.893	47.335	-116.888	43258	43335
16000485278	16000485278	13	50813	51077	14412	ID-5	19381559415	19395146779	41722	44540	H	O	47.335	-116.888	47.347	-116.724	43335	44126
16000485278	16000485278	14	51077	51402	14413	ID-5	19381559415	19395146779	41722	44540	H	O	47.347	-116.724	47.315	-116.563	44126	45041
16000485278	16000485278	15	51402	51398	14693	ID-3	19381559415	19395146779	41722	44540	H	O	47.315	-116.563	47.321	-116.566	45041	45091

Info

The Info table is primarily a helper table that maps the Primary Key, Pixel ID, to a state name so that one can easily determine the state a pixel is in. This table was useful in the creation of the oTrip tables, for it allowed us to determine which oTrip table to insert each trip into.

Nodes

The Node table is instrumental in creating the Path Database since it maps nodes to geographic coordinates. Its primary key is a Node Number.

Links

The Link table creates the edges of the network on which dijkstra runs, and thus is instrumental in visualizing trip flow. Node A and Node B provide references to the Node table so that one can determine the geographic coordinates of the edges of a Link. In addition, each Link has a distance that is used along with the RoadType to weight edges in the network.

7.1.3 Grid

The Grid database has 51 tables, the 50 states and an additional NoState. Each table holds information about the pixels within it, including the ID, the (X,Y) grid coordinates, and the latitude and longitude of all four of its corners and its centroid. The grid tables were used to create the Info table in the Global Database.

7.1.4 oTrip

The oTrip database has 51 tables, the 50 states and an additional NoState. Each row corresponds to a trip with a primary key, ID, that is a combination of the trip number (Trip) and the PersonID. Various statistics acquired from Wyrough's data, such as the departure and arrival time, are contained within each row.

7.1.5 Database Statistics

Aggregate Statistics		
Database Name	**Rows**	**Size (Gigabytes)**
oTrip (Total)	802,791,150.00	114.30
Global (Total)	24,110,172.00	1.20
Grid (Total)	19,996,641.00	4.30
Path (Partially Completed)	1,460,816,371.00	509.20
Total Size	**2,307,714,334.00**	**629.00**

Global Database		
Table Names	**Rows**	**Size (Megabytes)**
EntryNodes	2,093,296.00	322.96
Info	20,484,110.00	765.46
Links	1,034,662.00	80.35
Nodes	498,104.00	21.54
Global (Total)	**24,110,172.00**	**1,190.31**

Grid Database		
Table Names	**Rows**	**Size (Megabytes)**
Alabama	282,629.00	47.79
Alaska	-	-
Arizona	204,225.00	95.08
Arkansas	236,707.00	48.86
California	781,281.00	136.02
Colorado	331,319.00	88.75
Connecticut	24,711.00	4.73
Delaware	10,756.00	2.64
Florida	306,021.00	70.91
Georgia	261,903.00	50.95
Hawaii	-	-
Idaho	269,160.00	71.94
Illinois	126,502.00	50.95
Indiana	165,772.00	36.24
Iowa	151,554.00	48.86
Kansas	301,820.00	74.06
Kentucky	216,524.00	42.57
Louisiana	242,941.00	52.00
Maine	170,444.00	34.14
Maryland	39,060.00	12.08
Massachusetts	44,942.00	12.08
Michigan	340,057.00	75.10
Minnesota	372,145.00	76.17
Mississippi	278,277.00	45.71

Missouri	298,977.00	60.41
Montana	666,771.00	122.36
Nebraska	417,702.00	68.81
Nevada	497,821.00	90.87
New_Hampshire	37,577.00	9.98
New_Jersey	38,859.00	8.93
New_Mexico	551,799.00	102.40
New_York	150,590.00	49.92
NoState	7,113,569.00	1,596.98
North_Carolina	184,072.00	52.00
North_Dakota	297,852.00	63.55
Ohio	145,350.00	43.60
Oklahoma	360,483.00	62.50
Oregon	443,004.00	89.80
Pennsylvania	143,145.00	43.60
Pixels	-	-
Rhode_Island	6,070.00	1.59
South_Carolina	186,996.00	29.95
South_Dakota	367,545.00	68.81
Tennessee	179,799.00	43.61
Texas	1,296,414.00	201.13
Utah	281,853.00	74.04
Vermont	42,062.00	9.98
Virginia	141,875.00	42.55
Washington	250,107.00	66.73
West_Virginia	82,482.00	28.90
Wisconsin	232,713.00	58.31
Wyoming	422,404.00	84.56
Grid(Total)	**19,996,641.00**	**4,353.51**

Path Database		
TableName	**Rows**	**Size(Megabytes)**
Alabama	-	-
Alaska	-	-
Arizona	265,019,445.00	91,908.73
Arkansas	-	0.02
California	1,019,777.00	398.46
Colorado	215,967,031.00	68,704.80
Connecticut	-	-
Delaware	-	-
Florida	-	-
Georgia	-	-

Hawaii	-	-
Idaho	35,666,970.00	20,285.75
Illinois	-	-
Indiana	-	-
Iowa	7,577,719.00	2,210.40
Kansas	16,269,804.00	6,902.78
Kentucky	-	-
Louisiana	-	-
Maine	-	-
Maryland	-	-
Massachusetts	-	-
Michigan	-	-
Minnesota	9,529,535.00	2,660.24
Mississippi	-	0.02
Missouri	-	0.02
Montana	12,250,933.00	4,173.33
Nebraska	34,126,904.00	11,647.58
Nevada	207,533,962.00	76,004.98
New_Hampshire	-	-
New_Jersey	-	-
New_Mexico	54,557,687.00	31,387.03
New_York	-	-
NoState	-	-
North_Carolina	-	-
North_Dakota	9,187,509.00	3,449.82
Ohio	-	-
Oklahoma	133,342,779.00	37,921.75
Oregon	184,334,977.00	71,336.72
Pennsylvania	-	-
Rhode_Island	-	-
South_Carolina	-	-
South_Dakota	695,765.00	204.39
Tennessee	-	-
Texas	2,058,913.00	443.55
Utah	13,520,688.00	4,204.79
Vermont	-	-
Virginia	-	-
Washington	251,448,904.00	73,030.17
West_Virginia	-	-
Wisconsin	-	-
Wyoming	6,707,069.00	2,337.28
Path(Total)	**1,460,816,371.00**	**509,212.59**

oTrip Database		
Table Name	Rows	Size (Megabytes)
Alabama	14,045,217.00	2,000.68
Alaska	-	0.02
Arizona	20,818,459.00	2,975.86
Arkansas	9,525,233.00	1,351.61
California	86,030.00	14.20
Colorado	16,836,945.00	2,386.56
Connecticut	11,848,444.00	1,691.35
Delaware	3,112,024.00	447.74
Florida	61,125,049.00	8,646.56
Georgia	32,077,939.00	4,560.26
Hawaii	-	0.02
Idaho	5,148,099.00	737.15
Illinois	43,362,024.00	6,189.74
Indiana	21,112,514.00	3,015.70
Iowa	9,837,914.00	1,407.19
Kansas	9,290,284.00	1,328.55
Kentucky	14,627,779.00	2,089.81
Louisiana	14,813,619.00	2,117.07
Maine	4,351,043.00	623.90
Maryland	21,637,637.00	3,089.10
Massachusetts	21,880,781.00	3,121.61
Michigan	32,677,252.00	4,642.05
Minnesota	17,909,713.00	2,536.51
Mississippi	9,635,085.00	1,367.34
Missouri	19,332,453.00	2,737.83
Montana	3,248,241.00	462.42
Nebraska	6,044,073.00	858.78
Nevada	9,094,061.00	1,289.75
New_Hampshire	4,346,827.00	618.66
New_Jersey	29,036,593.00	4,127.20
New_Mexico	6,731,764.00	963.64
New_York	64,470,849.00	9,200.21
NoState	558,642.00	80.41
North_Carolina	31,706,429.00	4,524.61
North_Dakota	2,051,494.00	294.55
Ohio	38,091,689.00	5,436.87
Oklahoma	12,183,914.00	1,739.59
Oregon	11,937,814.00	1,703.94
Pennsylvania	41,575,384.00	5,932.84

Rhode_Island	3,618,484.00	519.05
South_Carolina	15,475,801.00	2,192.57
South_Dakota	29,438.00	5.78
Tennessee	21,092,985.00	2,988.44
Texas	225,884.00	34.16
Utah	9,030,493.00	1,279.26
Vermont	2,054,949.00	292.42
Virginia	26,506,821.00	3,750.76
Washington	22,345,437.00	3,162.51
West_Virginia	5,885,965.00	834.67
Wisconsin	18,497,137.00	2,618.29
Wyoming	1,858,445.00	264.08
oTrip(Total)	**802,791,150.00**	**114,253.87**

7.2 Programming Languages

We chose java as our main programming language due to familiarity, the wealth of data structures the language contains, and the ease with which it connects with MySQL . The Princeton algs4.jar package, which both of us have experience with from COS 226, was heavily used in the implementation of our algorithms. Aside from java, we used Python, Bash, HTML, and Javascript. Python, a robust scripting language, allowed us to quickly and painlessly modify the network files whereas Bash automated the execution of many of our larger processes. Lastly, HTML and Javascript were used to construct the WebGL application.

Appendix A

Visualizations

A.1 Total Cumulative Trip Miles in Continental United States

Table A.1: Total Cumulative Trip Miles in Continental United States Midnight to Noon

Time	12:00-1:00am	1:00-2:00am	2:00-3:00am	3:00-4:00am	4:00-5:00am	5:00-6:00am	6:00-7:00am	7:00-8:00am	8:00-9:00am	9:00-10:00am	10:00-11:00am	11:00-12:00am
Alabama	337 906	1 191 930	2 175 929	3 409 863	5 214 844	8 827 330	16 818 337	28 483 647	37 651 015	42 555 070	47 974 012	55 566 259
Alaska	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
Arizona	817 862	2 793 694	5 405 977	9 128 817	14 516 455	23 796 363	40 477 973	61 941 569	77 859 329	86 348 457	98 027 453	115 495 672
Arkansas	240 843	1 106 185	2 005 953	3 029 280	4 311 531	6 530 342	11 261 249	18 153 964	23 692 513	26 769 205	30 156 496	34 912 036
California	NULL	1 038 346	2 044 864	2 969 152	3 916 810	4 923 435	5 854 349	6 898 914	7 897 021	8 881 344	9 830 004	10 840 390
Colorado	153 416	806 486	1 564 953	2 557 748	4 009 460	7 037 016	13 786 089	24 294 423	33 537 882	38 796 980	44 101 831	51 102 577
Connecticut	84 197	532 890	996 765	1 664 216	2 676 015	4 891 809	10 667 779	20 196 214	28 275 369	32 557 007	37 192 726	43 820 568
Delaware	25 953	151 582	303 021	555 753	941 673	1 613 687	2 960 378	5 093 006	6 950 805	8 038 929	9 234 794	10 895 853
Florida	931 907	4 603 273	8 655 145	13 484 984	19 765 033	31 862 785	60 177 601	105 464 112	144 185 091	165 173 007	190 176 834	227 053 181
Georgia	674 182	2 512 393	4 528 863	6 929 081	9 902 986	15 323 893	28 592 004	50 756 391	69 855 968	80 290 456	90 305 198	103 405 066
Hawaii	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
Idaho	143 758	428 856	741 333	1 151 149	1 743 940	2 918 992	5 282 585	8 539 212	11 088 494	12 445 177	13 874 113	15 867 233
Illinois	685 150	2 549 378	4 604 544	7 123 001	10 383 819	17 284 273	36 470 308	68 976 391	96 252 867	110 747 369	126 761 491	149 194 435
Indiana	267 372	1 214 482	2 283 006	3 654 808	5 433 983	8 680 436	16 573 179	29 746 303	41 096 928	47 277 207	53 521 176	61 925 758
Iowa	169 359	646 275	1 169 052	1 803 220	2 656 887	4 393 755	8 737 321	15 321 928	20 480 215	23 144 361	25 943 286	29 965 599
Kansas	145 956	671 955	1 301 326	2 084 809	3 060 546	4 737 594	8 562 321	14 443 056	19 265 545	21 893 396	24 639 116	28 384 735
Kentucky	277 432	1 025 679	1 841 926	2 842 094	4 101 767	6 373 794	11 789 082	20 737 163	28 412 267	32 542 232	36 679 614	42 329 632
Louisiana	561 303	2 072 858	3 672 716	5 465 062	7 582 887	11 146 310	18 348 870	28 901 788	37 717 622	42 784 032	47 983 203	54 784 961
Maine	81 646	389 510	780 027	1 339 433	2 223 255	3 961 167	7 336 355	11 712 807	15 039 322	16 842 678	19 081 808	22 294 715
Maryland	187 947	1 244 358	2 448 648	3 952 019	5 846 170	9 202 855	17 406 328	31 788 246	44 871 295	52 027 059	58 859 806	67 993 995
MA	174 062	1 276 531	2 543 415	4 106 853	6 131 405	10 062 680	20 153 126	37 351 280	52 267 645	60 464 004	69 400 929	81 799 317
Michigan	534 315	2 291 727	4 297 371	6 955 546	10 704 175	17 689 405	34 014 481	59 589 643	80 847 330	92 667 257	105 653 735	123 273 629
Minnesota	292 770	1 333 786	2 496 255	4 055 634	6 393 922	11 144 260	21 111 940	35 355 821	46 877 934	53 191 122	59 901 205	69 081 257
Mississippi	322 975	952 886	1 684 740	2 586 949	3 853 694	6 353 441	11 842 838	19 610 249	25 593 662	28 718 724	31 922 442	36 265 310
Missouri	441 412	1 523 412	2 744 846	4 174 471	6 007 113	9 460 424	17 943 382	31 658 312	43 134 495	49 448 757	56 172 948	65 385 590
Montana	77 914	349 216	695 253	1 341 997	2 486 994	3 948 039	5 844 382	7 982 407	9 675 794	10 724 261	12 025 752	13 883 917
Nebraska	70 756	293 775	571 416	908 011	1 382 388	2 290 939	4 384 866	7 613 165	10 317 749	11 751 104	13 274 365	15 474 759
Nevada	413 987	1 277 540	2 394 968	3 706 387	5 084 088	6 858 850	9 936 489	14 981 981	19 733 328	22 354 488	25 573 606	30 432 963
N Hampshire	59 440	328 016	636 590	1 026 857	1 636 857	2 988 000	6 072 970	10 487 871	13 940 361	15 839 545	17 907 930	20 745 534
NJ	214 429	1 385 674	2 767 574	4 619 049	7 248 128	12 196 487	23 866 874	43 122 483	60 434 806	70 219 155	79 742 113	92 476 892
NM	164 827	526 559	974 237	1 689 923	3 203 663	6 082 359	10 246 594	14 817 949	18 327 822	20 234 550	22 472 057	25 707 069
NY	849 093	4 253 933	8 015 905	12 386 177	17 765 928	27 698 443	51 039 071	89 730 225	124 646 024	144 417 129	164 669 507	192 225 282
NoState	8 288	47 243	98 220	154 313	242 790	393 998	671 158	1 077 744	1 434 592	1 631 692	1 848 800	2 148 731
N Carolina	740 369	2 476 644	4 390 055	6 699 605	9 748 840	15 504 300	29 628 395	52 177 386	70 883 365	80 946 192	91 650 995	106 508 193
N Dakota	45 259	207 733	387 911	602 630	965 522	1 676 416	2 971 461	4 469 765	5 553 545	6 140 455	6 842 596	7 847 360
Ohio	521 907	2 187 635	4 053 307	6 402 095	9 512 876	15 247 801	29 883 811	55 000 810	77 024 777	89 168 433	101 977 243	119 537 861
Oklahoma	242 723	917 647	1 696 447	2 636 087	3 976 706	6 769 886	12 906 554	21 716 373	28 627 055	32 307 230	36 428 676	42 331 369
Oregon	181 781	725 888	1 361 240	2 232 126	3 433 594	5 749 750	10 782 168	18 475 058	25 021 890	28 684 442	32 805 932	38 635 479
Pennsylvania	724 575	2 713 460	4 940 821	7 811 436	11 752 411	19 271 017	36 959 850	65 891 909	90 777 905	104 369 683	119 480 115	140 895 185
RI	31 391	140 906	309 636	544 370	852 216	1 359 607	2 530 099	4 585 367	6 490 085	7 614 574	8 696 976	10 068 461
S Carolina	287 860	1 126 432	2 042 261	3 171 994	4 686 571	7 682 352	14 750 179	25 964 390	35 431 148	40 652 511	46 430 714	54 465 923
S Dakota	NULL	14 675	27 302	41 558	73 225	104 624	142 152	177 771	212 765	246 277	287 336	327 620
Tennessee	434 614	1 459 612	2 580 241	3 945 846	5 615 964	8 793 818	17 039 496	30 921 186	42 843 076	49 361 922	56 215 674	65 646 669
Texas	398	578 187	1 176 667	1 778 559	2 375 979	2 943 245	3 584 681	4 208 330	4 877 040	5 624 419	6 458 427	7 338 179
Utah	194 381	804 521	1 641 766	2 808 422	4 271 848	6 554 840	10 473 552	15 933 455	20 597 724	23 105 630	25 591 052	29 008 688
Vermont	43 190	240 097	457 633	701 004	1 037 767	1 649 677	2 931 697	4 746 846	6 204 195	7 052 101	7 964 097	9 187 486
Virginia	561 760	2 111 121	3 801 036	5 853 750	8 435 347	12 941 780	23 298 312	40 552 584	55 991 063	64 464 105	72 588 839	83 451 382
Washington	225 069	1 176 631	2 292 958	3 896 034	6 275 722	11 199 903	22 455 928	39 737 678	54 252 420	62 441 572	71 306 871	83 223 513
W Virginia	183 778	677 438	1 270 390	1 899 703	2 644 384	3 951 867	6 812 862	11 006 751	14 507 274	16 490 611	18 581 196	21 437 198
Wisconsin	251 052	1 138 878	2 244 441	3 851 536	6 177 455	10 526 870	19 925 359	33 486 462	44 379 453	50 362 222	56 978 094	66 114 645
Wyoming	67 651	228 025	434 542	874 132	1 668 252	2 687 432	3 923 237	5 075 707	5 877 488	6 333 889	6 892 173	7 742 275
Total	14 148 215	59 745 959	111 553 495	176 607 544	263 933 885	425 288 345	789 230 070	1 358 956 090	1 840 913 359	2 106 142 020	2 392 085 355	2 788 500 403

Table A.2: Total Cumulative Trip Miles in Continental United States Noon to Midnight

Time	12:00-1:00pm	1:00-2:00pm	2:00-3:00pm	3:00-4:00pm	4:00-5:00pm	5:00-6:00pm	6:00-7:00pm	7:00-8:00pm	8:00-9:00pm	9:00-10:00pm	10:00-11:00pm	11:00-12:00pm
Alabama	60 557 779	67 892 716	76 782 252	88 458 769	102 331 882	116 826 919	130 328 223	142 049 941	151 622 746	159 044 212	164 433 281	168 068 887
Alaska	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
Arizona	124 593 754	140 371 049	160 025 305	185 004 821	213 514 129	242 467 680	269 524 875	293 442 447	313 987 991	330 707 758	343 618 557	353 106 120
Arkansas	37 900 630	42 396 021	48 328 309	56 681 918	67 058 523	78 082 703	88 255 261	96 721 430	103 238 935	107 966 038	111 105 414	113 053 563
California	11 327 177	13 142 916	17 813 384	26 762 957	39 259 216	52 667 790	64 323 447	73 057 805	78 857 772	82 373 575	84 322 352	85 153 910
Colorado	55 720 569	62 138 750	69 957 786	80 426 638	93 366 912	107 405 376	120 769 946	132 353 516	141 910 973	149 227 948	154 421 761	157 830 101
Connecticut	48 347 937	54 620 147	61 577 410	70 441 948	81 071 110	92 545 079	103 742 425	113 620 470	121 837 891	128 197 477	132 789 398	135 873 955
Delaware	12 016 725	13 563 730	15 301 295	17 529 265	20 249 768	23 241 498	26 207 748	28 893 859	31 134 509	32 894 974	34 167 812	35 009 788
Florida	250 845 186	284 612 356	322 609 250	371 538 075	430 256 199	493 550 368	553 771 700	606 308 109	649 140 637	681 978 907	705 254 282	720 741 275
Georgia	112 363 637	124 463 531	139 573 106	160 758 579	187 601 398	217 273 979	245 900 877	270 823 672	291 094 959	306 417 476	317 239 600	324 321 932
Hawaii	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
Idaho	17 277 889	19 406 264	22 288 382	26 128 973	30 436 468	34 716 019	38 558 690	41 713 695	44 213 916	46 106 420	47 428 740	48 295 434
Illinois	164 087 471	185 764 719	211 190 530	244 876 186	286 136 908	331 525 244	375 760 146	415 123 249	447 507 304	472 689 672	491 239 151	503 952 428
Indiana	67 730 399	75 500 379	84 856 977	97 539 388	113 120 019	129 832 315	145 515 695	158 947 727	169 727 400	177 877 987	183 583 655	187 276 250
Iowa	32 765 067	36 782 256	41 788 463	48 582 805	56 728 476	65 184 004	72 822 412	79 197 759	84 173 751	87 853 417	90 365 707	91 990 937
Kansas	30 977 485	34 645 151	39 220 437	45 583 023	53 481 897	61 951 203	69 818 358	76 596 668	81 960 012	85 950 872	88 725 789	90 506 038
Kentucky	46 195 518	51 367 526	57 676 213	66 332 621	77 122 456	88 707 879	99 459 197	108 603 995	115 818 761	121 153 795	124 842 873	127 215 786
Louisiana	59 135 168	65 460 571	73 850 515	85 726 359	100 598 174	116 483 933	131 034 323	143 400 677	153 104 220	160 239 160	165 116 501	168 235 496
Maine	24 072 677	26 981 491	30 479 647	34 933 989	40 066 211	45 376 777	50 410 027	54 801 309	58 479 628	61 450 433	63 691 279	65 310 494
Maryland	74 258 464	82 584 165	92 557 900	106 303 180	123 992 535	143 734 089	162 827 448	179 420 675	192 818 574	202 753 722	209 653 274	214 041 222
MA	89 917 358	101 371 892	114 412 939	131 312 859	152 022 414	174 703 391	197 001 658	216 988 098	233 691 963	246 606 972	255 985 799	262 364 728
Michigan	135 029 290	151 760 943	171 354 733	196 944 670	227 695 202	260 528 708	291 964 322	319 673 360	342 713 320	360 678 216	373 764 235	382 611 839
Minnesota	74 940 490	83 592 953	94 449 158	109 041 253	126 710 061	145 589 031	163 437 135	178 988 178	191 688 132	201 455 541	208 505 309	213 280 857
Mississippi	39 177 835	43 298 223	48 399 644	55 203 881	63 398 746	71 907 059	79 756 612	86 378 995	91 682 843	95 709 314	98 576 680	100 463 199
Missouri	71 540 707	79 960 596	89 765 412	102 822 343	118 927 567	136 284 039	152 954 341	167 649 153	179 701 126	188 813 649	195 329 365	199 659 415
Montana	14 942 586	16 739 628	19 128 532	22 329 003	26 264 136	30 517 105	34 488 953	37 879 990	40 559 227	42 584 358	44 012 452	45 004 530
Nebraska	16 989 893	19 076 475	21 594 362	25 032 303	29 310 639	33 874 680	38 164 760	41 754 662	44 626 516	46 728 022	48 203 593	49 139 168
Nevada	33 311 447	37 414 619	42 511 518	49 303 328	57 510 852	66 293 735	74 674 023	82 136 171	88 463 569	93 499 100	97 309 716	100 022 163
NH	22 482 680	25 076 249	28 081 887	31 955 940	36 676 518	41 745 344	46 722 092	51 198 379	54 959 429	57 901 636	60 077 455	61 596 045
NJ	101 539 394	113 033 536	126 436 553	144 909 674	169 268 010	197 347 960	225 784 292	251 385 404	272 255 534	288 054 537	299 118 856	306 154 527
NM	27 544 951	30 595 389	34 670 213	39 994 155	46 146 304	52 383 590	58 130 594	63 024 854	67 021 209	70 198 177	72 528 475	74 166 165
NY	210 170 496	234 620 061	263 784 492	303 153 419	352 224 603	405 227 511	455 567 961	499 255 051	534 548 525	561 254 741	579 944 145	592 178 174
NoState	2 350 528	2 664 089	3 030 733	3 514 035	4 115 626	4 758 327	5 371 521	5 917 140	6 357 501	6 684 422	6 929 812	7 101 522
N Carolina	116 602 107	130 644 695	147 369 736	169 593 338	196 752 484	225 956 013	253 701 063	277 888 907	297 564 343	312 654 722	323 467 655	330 678 906
N Dakota	8 459 750	9 451 508	10 767 521	12 558 384	14 813 810	17 259 015	19 569 766	21 555 069	23 126 470	24 283 006	25 121 692	25 668 758
Ohio	131 638 862	147 730 771	166 025 661	190 066 430	219 485 214	251 529 098	282 265 558	309 340 482	331 591 333	348 738 440	360 914 833	368 963 982
Oklahoma	46 169 990	51 794 699	58 498 737	67 381 048	77 981 247	89 183 560	99 666 600	108 647 082	115 916 104	121 392 914	125 356 567	127 993 232
Oregon	42 191 406	47 302 181	53 380 075	61 308 157	70 748 302	80 728 402	90 126 029	98 264 661	104 921 471	110 017 003	113 668 676	116 119 309
PA	155 124 937	174 392 343	195 594 056	222 666 907	255 552 951	291 206 945	325 744 960	356 313 093	381 644 863	401 295 044	415 495 450	425 107 868
RI	11 020 671	12 201 717	13 478 383	15 207 669	17 439 689	20 010 654	22 575 240	24 914 286	26 859 616	28 363 876	29 427 887	30 113 204
S Carolina	59 678 382	67 079 092	75 620 579	86 666 894	99 955 598	114 262 861	127 991 053	139 999 665	149 971 694	157 681 264	163 240 121	167 027 598
S Dakota	358 726	420 089	556 170	780 883	1 059 767	1 382 202	1 696 557	1 948 177	2 135 755	2 254 500	2 339 631	2 391 464
Tennessee	72 088 336	80 551 228	90 264 771	103 018 636	118 873 615	136 147 874	152 811 842	167 380 348	179 321 936	188 459 304	194 949 645	199 237 135
Texas	7 954 907	9 463 054	12 979 056	19 770 534	29 978 138	42 159 879	54 436 435	64 915 465	72 753 951	78 266 844	81 719 237	83 753 237
Utah	31 387 708	34 837 324	39 426 664	45 750 559	53 534 052	61 671 976	69 266 927	75 684 995	80 739 651	84 589 280	87 317 552	89 084 184
Vermont	9 947 922	11 096 542	12 513 216	14 406 967	16 791 177	19 421 496	21 846 600	24 006 615	25 771 454	27 105 163	28 040 328	28 661 465
Virginia	90 987 198	100 929 649	112 869 756	129 608 659	150 823 257	174 360 707	196 857 596	216 363 607	231 882 422	243 445 804	251 338 146	256 408 911
Washington	90 733 850	101 596 416	114 544 938	131 557 316	152 354 293	174 907 652	196 831 462	216 293 949	232 452 337	244 988 281	254 048 985	260 189 843
W Virginia	23 260 239	25 720 857	28 670 454	32 628 787	37 540 578	42 883 404	47 947 414	52 335 364	55 873 862	58 479 840	60 297 771	61 488 228
Wisconsin	72 116 808	80 621 975	90 553 346	103 500 138	119 162 428	135 913 023	151 895 331	165 922 624	177 511 393	186 448 215	192 928 293	197 286 611
Wyoming	8 227 827	9 079 584	10 319 543	12 047 504	14 009 904	16 025 099	17 818 874	19 316 369	20 522 928	21 435 140	22 093 090	22 541 881
Total	3 048 060 813	3 415 812 114	3 856 930 000	4 447 645 166	5 173 519 462	5 957 743 194	6 706 098 371	7 358 397 197	7 889 460 457	8 294 951 164	8 584 050 877	8 776 441 763

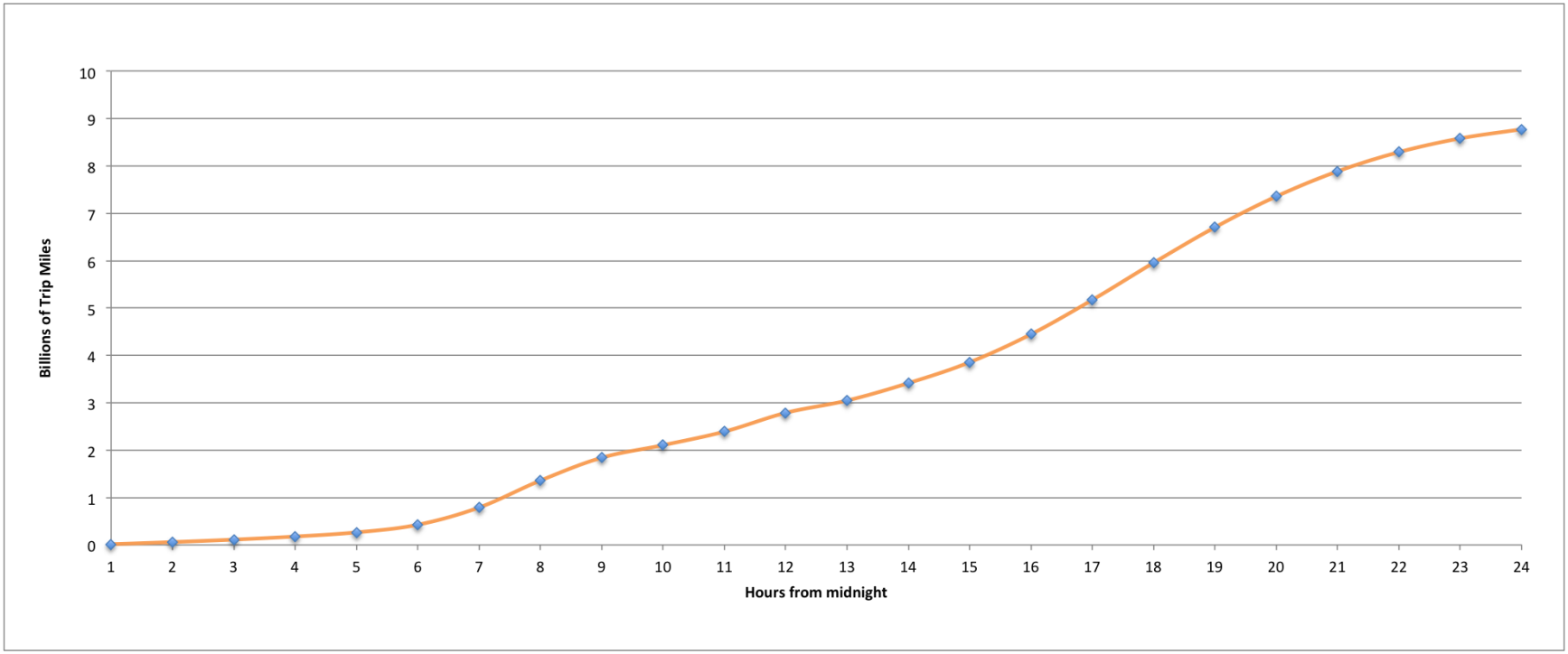

Figure A.1: Total Cumulative Trip Miles in Continental United States

A.2 Nationwide Statistics

Table A.3: Nationwide Statistics

State	Total # Pixels	oPixels used	Utilization rate	Population	# Trips	Avg trips person	# Trips dTime <0	# Trips dTime >86400
Alabama	210974	65582	31.09%	3798007	14047676	3.698696711	12063	513877
Alaska	0	0	N/A	0	0	N/A	0	0
Arizona	457460	30374	6.64%	5506390	20824276	3.781838192	26636	1049834
Arkansas	212123	51398	24.23%	2605597	9524339	3.655338489	8379	324886
California	683061	16208	2.37%	78880	92109	1.167710446	1	2281
Colorado	416267	36705	8.82%	4383303	16836209	3.840986808	19921	573185
Connecticut	22212	15932	71.73%	3201366	11860177	3.704723859	4711	402542
Delaware	10867	5266	48.46%	1042780	3111669	2.984012927	1042	107468
Florida	303957	69804	22.97%	16121250	61123253	3.791471071	81163	2243784
Georgia	241925	86114	35.60%	8590325	32045503	3.730417999	31454	1058790
Hawaii	0	0	N/A	0	0	N/A	0	0
Idaho	332972	23808	7.15%	1392827	5148229	3.696244401	5946	152812
Illinois	232578	92214	39.65%	11823004	43366874	3.668008063	26251	1529364
Indiana	145157	74940	51.63%	5776264	21112251	3.655001053	11167	649511
Iowa	224697	77568	34.52%	2709914	9838074	3.630400817	4234	273490
Kansas	330053	58056	17.59%	2591401	9291111	3.585362127	6689	275843
Kentucky	162252	53998	33.28%	4292171	14628725	3.408234434	9481	447106
Louisiana	220773	43502	19.70%	3928607	14813426	3.770656113	14211	582137
Maine	143692	24352	16.95%	1166898	4346032	3.724431784	2761	200432
Maryland	50856	26306	51.73%	6054723	21634524	3.573164949	9566	664698
MA	47686	23653	49.60%	5952267	21858431	3.67228671	9208	803980
Michigan	386512	96702	25.02%	8625697	32715257	3.792766776	21412	1235700
Minnesota	346501	84986	24.53%	4811955	17907454	3.721450845	11152	657615
Mississippi	195932	52528	26.81%	2635915	9634011	3.654901998	11207	324408
Missouri	279585	88739	31.74%	5417395	19328721	3.567899516	15953	668237
Montana	588155	24736	4.21%	862417	3247648	3.765751371	4273	145369
Nebraska	309585	46028	14.87%	1628107	6042331	3.711261606	3628	159741
Nevada	443633	8430	1.90%	2423658	9092934	3.751739726	8793	344427
NH	37558	15724	41.87%	1268903	4350396	3.428470104	2758	192871
New_Jersey	35740	21950	61.42%	8245286	29049381	3.523150198	10216	879239
New_Mexico	488532	22532	4.61%	1777972	6731198	3.785885267	6707	245958
New_York	217971	92215	42.31%	17306143	64472938	3.725436569	23228	1976145
North_Carolina	217946	92197	42.30%	8433008	31707505	3.759928249	28427	1067101
North_Dakota	282321	29541	10.46%	598462	2051396	3.427779876	1330	69525
Ohio	179976	98618	54.80%	10377446	38093763	3.67082257	21310	1267008
Oklahoma	280617	58728	20.93%	3224750	12183498	3.778121715	10679	412081
Oregon	402595	32167	7.99%	3194878	11938203	3.736669444	8954	378023
PA	184540	94809	51.38%	11277325	41577143	3.686791238	21397	1424139
Rhode_Island	6503	3455	53.13%	1117316	3618272	3.2383605	1643	113293
S. Carolina	132600	51777	39.05%	4230545	15473126	3.657478174	13523	556323
S. Dakota	307637	1702	0.55%	28914	31862	1.101957529	0	1027
Tennessee	169283	73452	43.39%	5746662	21091201	3.670165567	20641	687200
Texas	1089562	29683	2.72%	202619	225414	1.112501789	10	17913
Utah	339918	12441	3.66%	2370325	9029934	3.809576324	8237	274844
Vermont	38481	13523	35.14%	570378	2054706	3.602358436	1335	80571
Virginia	173400	69609	40.14%	7256584	26503849	3.65238644	16674	829568
Washington	287753	42305	14.70%	5876639	22344626	3.802279841	18677	822941
W. Virginia	97055	29942	30.85%	1766877	5882579	3.329365315	3713	203323
Wisconsin	261309	83526	31.96%	4972933	18496323	3.719399196	8589	635682
Wyoming	391171	11947	3.05%	498916	1857630	3.723332184	1971	66142
Total	12621933	2259772	17.90%	217763999	802236187	3.683970678	591321	27592434

A.3 Statistics by State

Section A.2	Alabama	Page 42
Section A.5	Arizona	Page 45
Section A.8	Arkansas	Page 48
Section A.11	California	Page 51
Section A.14	Colorado	Page 54
Section A.17	Connecticut	Page 57
Section A.20	Delaware	Page 60
Section A.23	Florida	Page 63
Section A.26	Georgia	Page 66
Section A.29	Idaho	Page 69
Section A.32	Illinois	Page 72
Section A.35	Indiana	Page 75
Section A.38	Iowa	Page 78
Section A.41	Kansas	Page 81
Section A.44	Kentucky	Page 84
Section A.47	Louisiana	Page 87
Section A.50	Maine	Page 90
Section A.53	Maryland	Page 93
Section A.56	Massachusetts	Page 96
Section A.59	Michigan	Page 99
Section A.62	Minnesota	Page 102
Section A.65	Mississippi	Page 105
Section A.68	Missouri	Page 108
Section A.71	Montana	Page 111
Section A.74	Nebraska	Page 114
Section A.77	Nevada	Page 117
Section A.80	New_Hampshire	Page 120
Section A.83	New_Jersey	Page 123
Section A.86	New_Mexico	Page 126
Section A.89	New_York	Page 129
Section A.92	North_Carolina	Page 132
Section A.95	North_Dakota	Page 135
Section A.98	Ohio	Page 138
Section A.101	Oklahoma	Page 141
Section A.104	Oregon	Page 144
Section A.107	Pennsylvania	Page 147
Section A.110	Rhode_Island	Page 150
Section A.113	South_Carolina	Page 153
Section A.116	South_Dakota	Page 156
Section A.119	Tennessee	Page 159
Section A.122	Texas	Page 162
Section A.125	Utah	Page 165
Section A.128	Vermont	Page 168
Section A.131	Virginia	Page 171
Section A.134	Washington	Page 174
Section A.137	West_Virginia	Page 177
Section A.140	Wisconsin	Page 180
Section A.143	Wyoming	Page 183

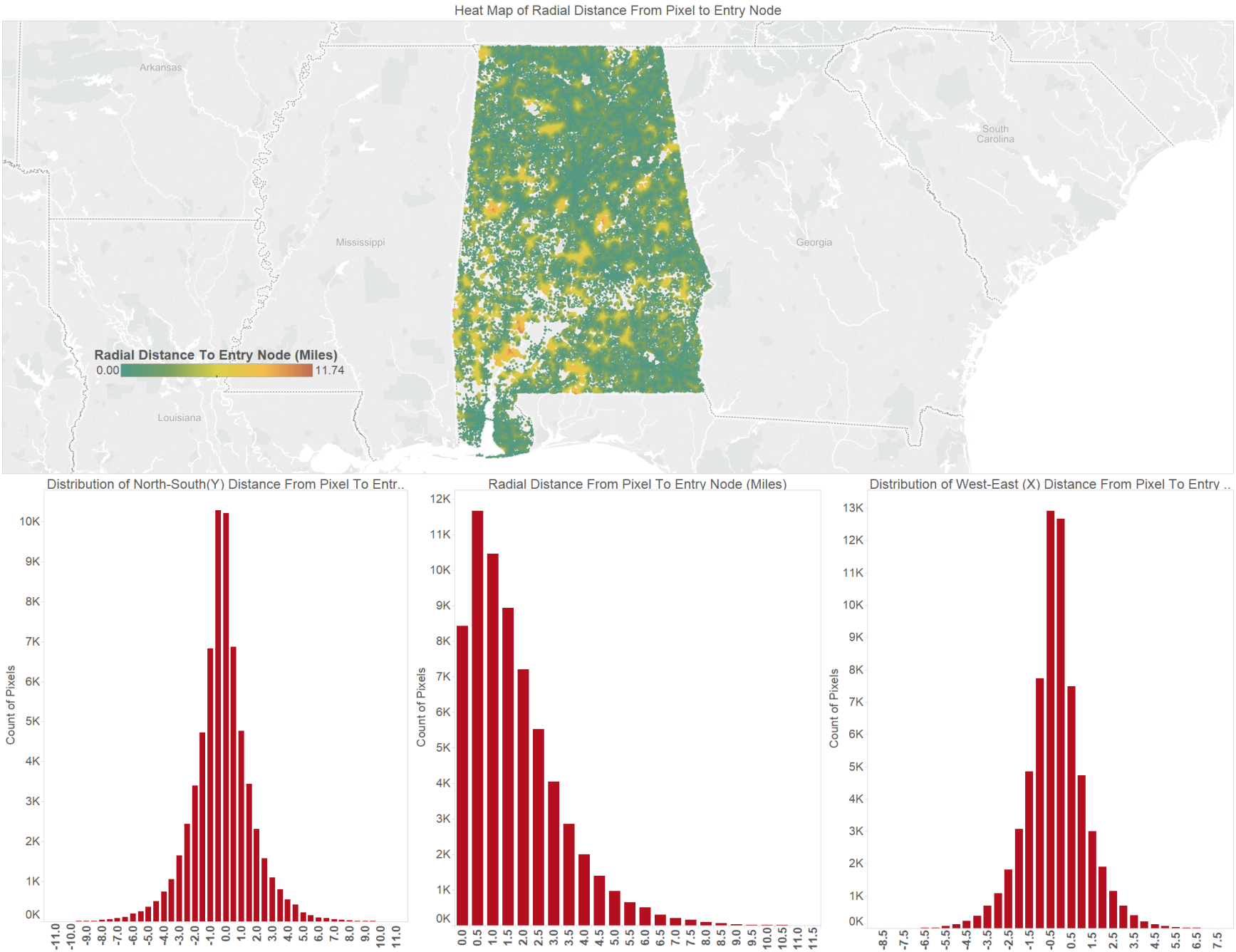

Figure A.2: Alabama: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

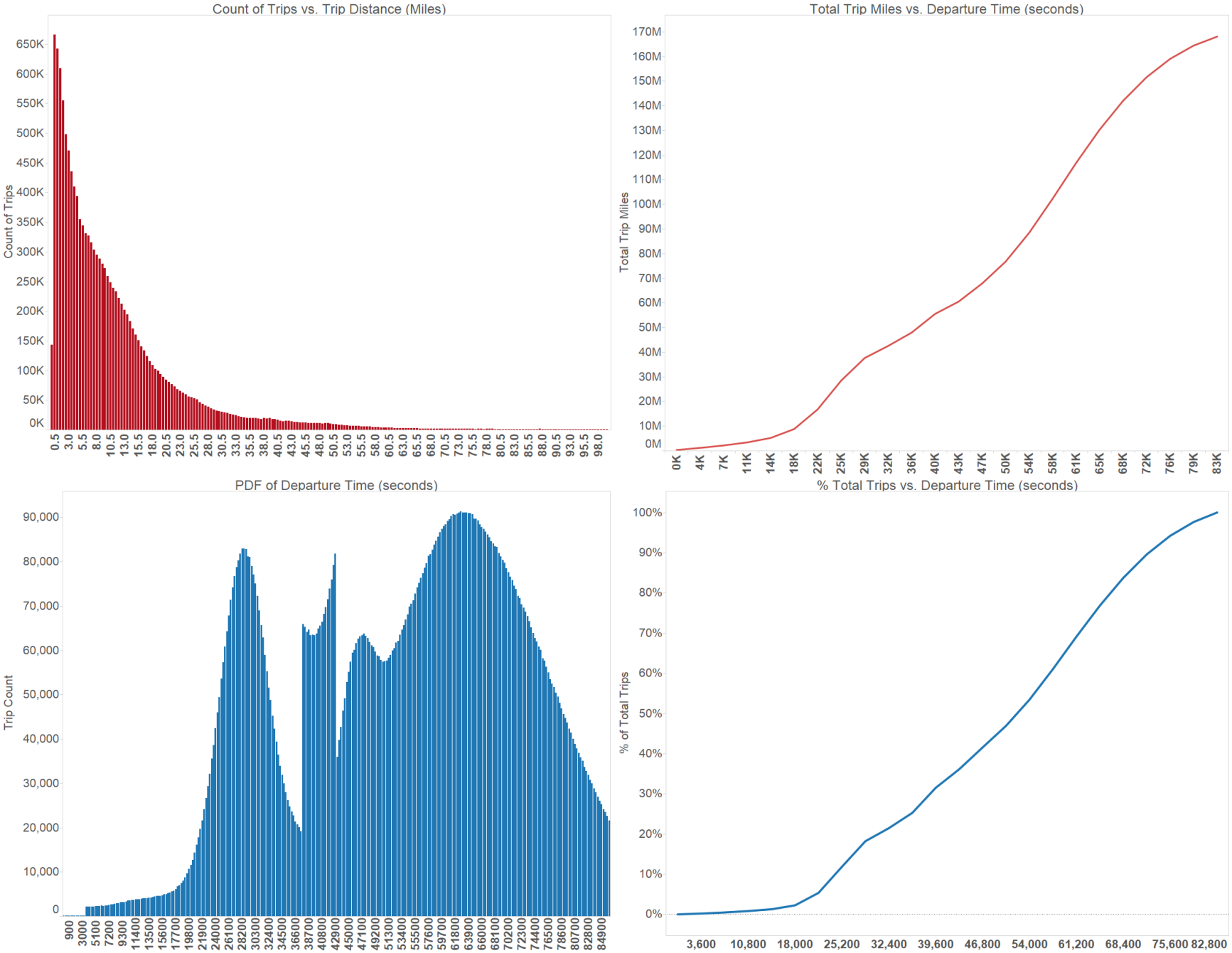

Figure A.3: Alabama: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

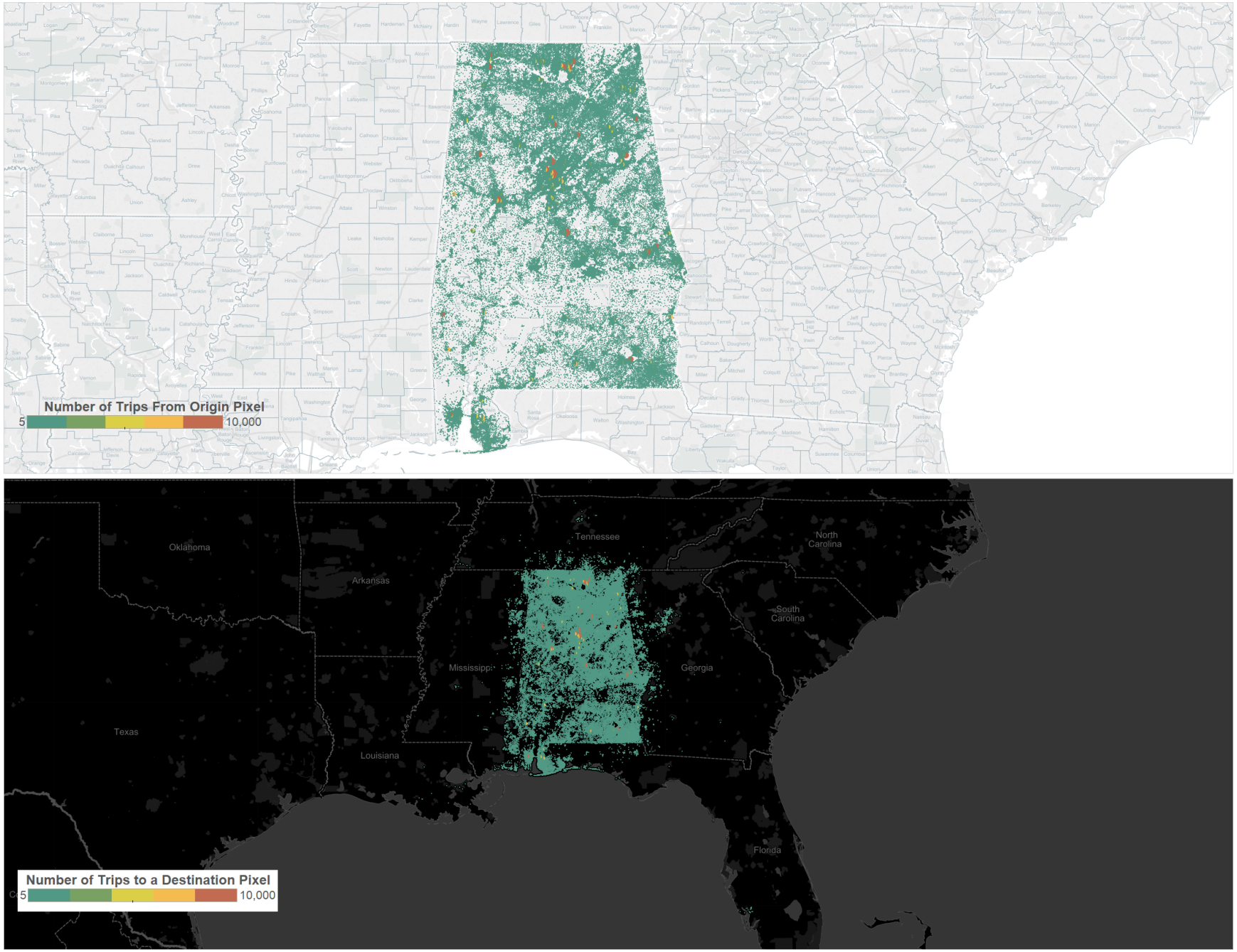

Figure A.4: Alabama: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

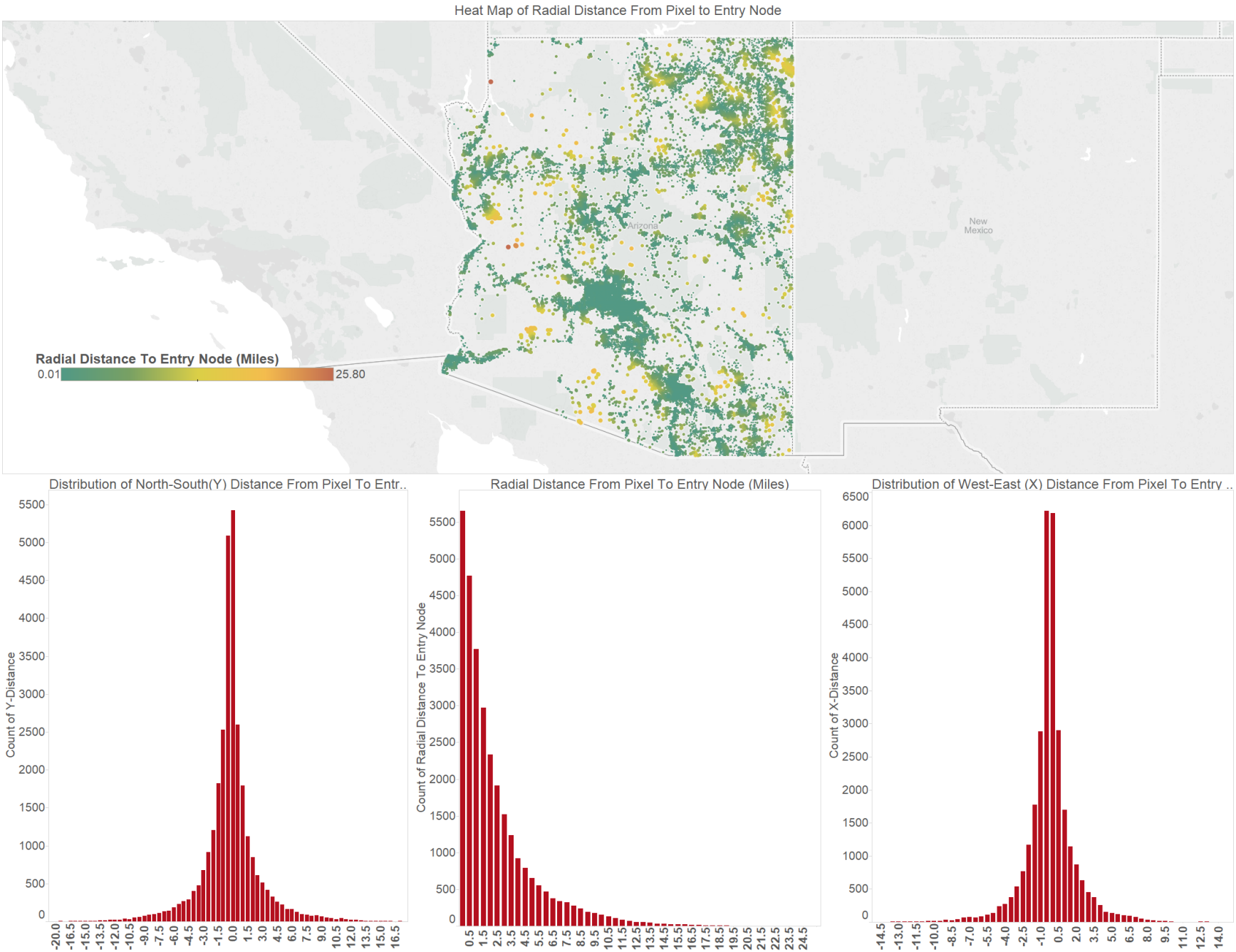

Figure A.5: Arizona: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

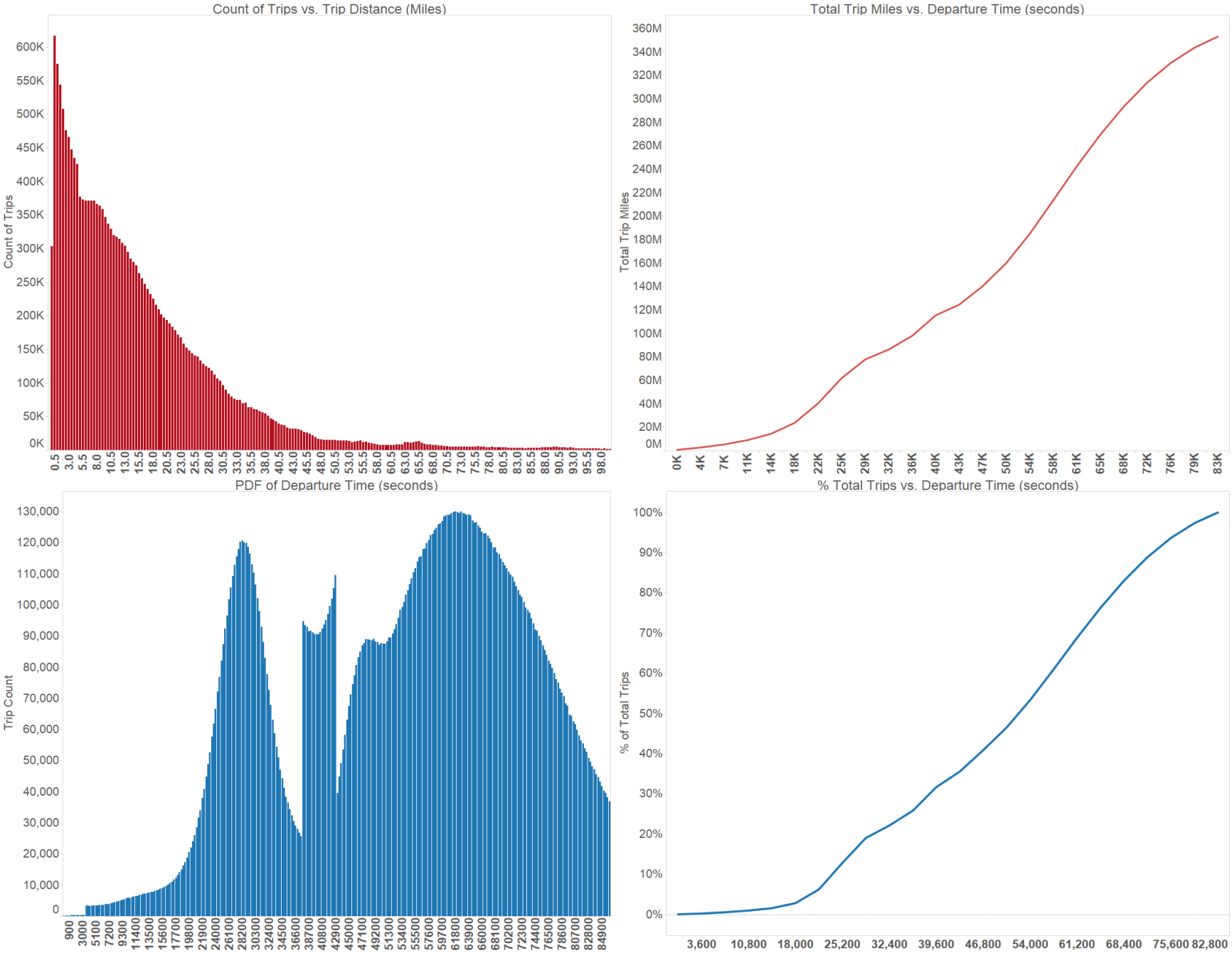

Figure A.6: Arizona: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

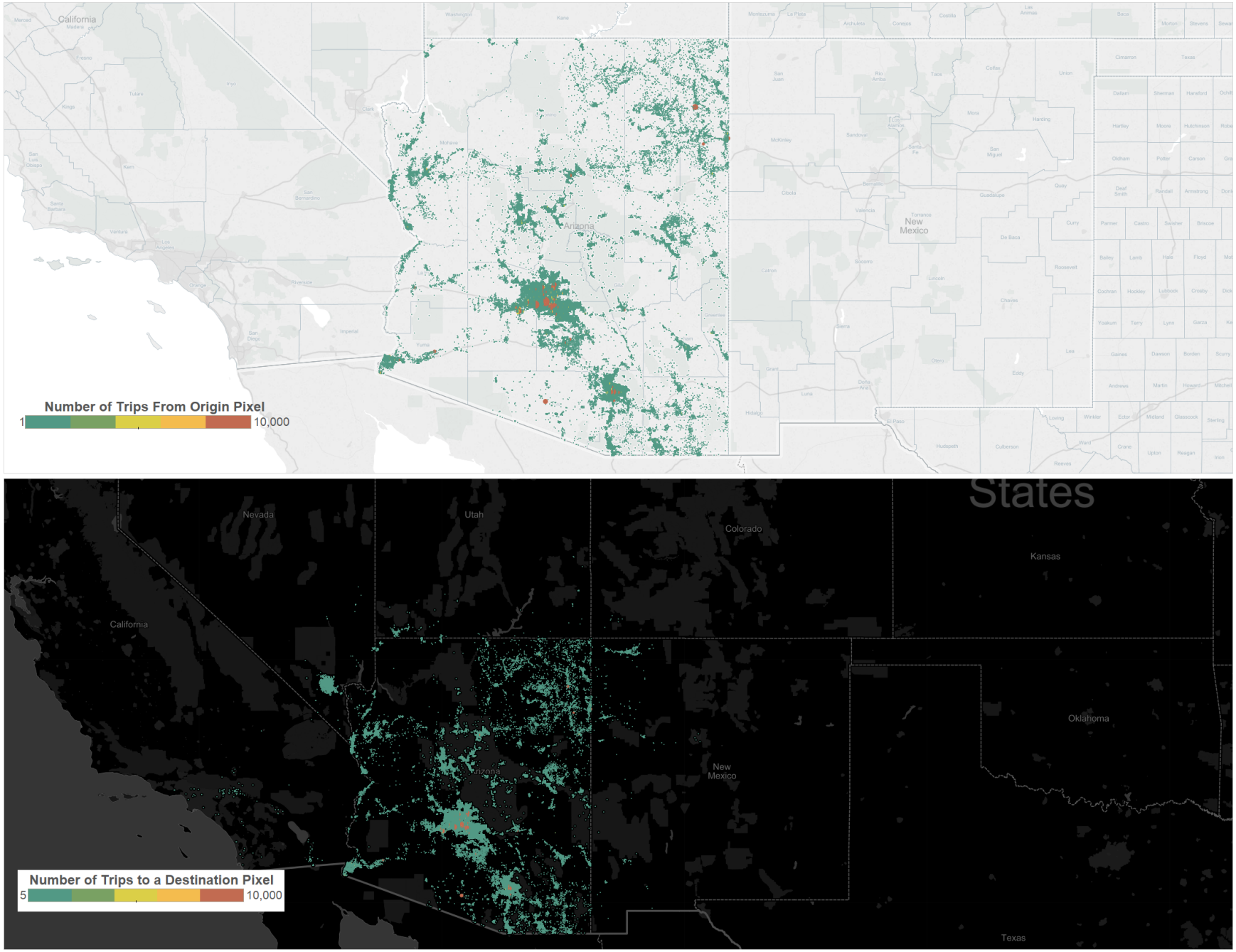

Figure A.7: Arizona: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

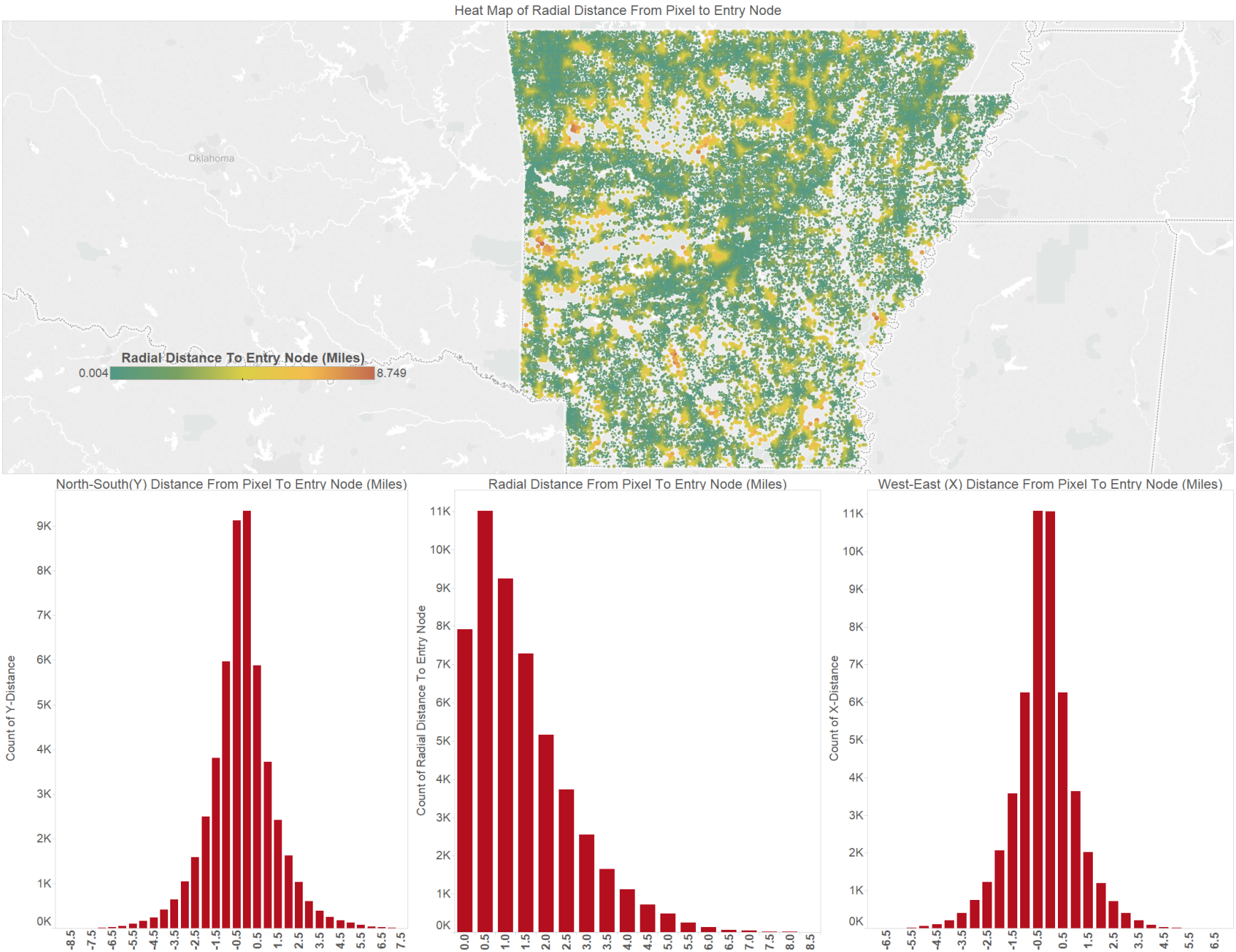

Figure A.8: Arkansas: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

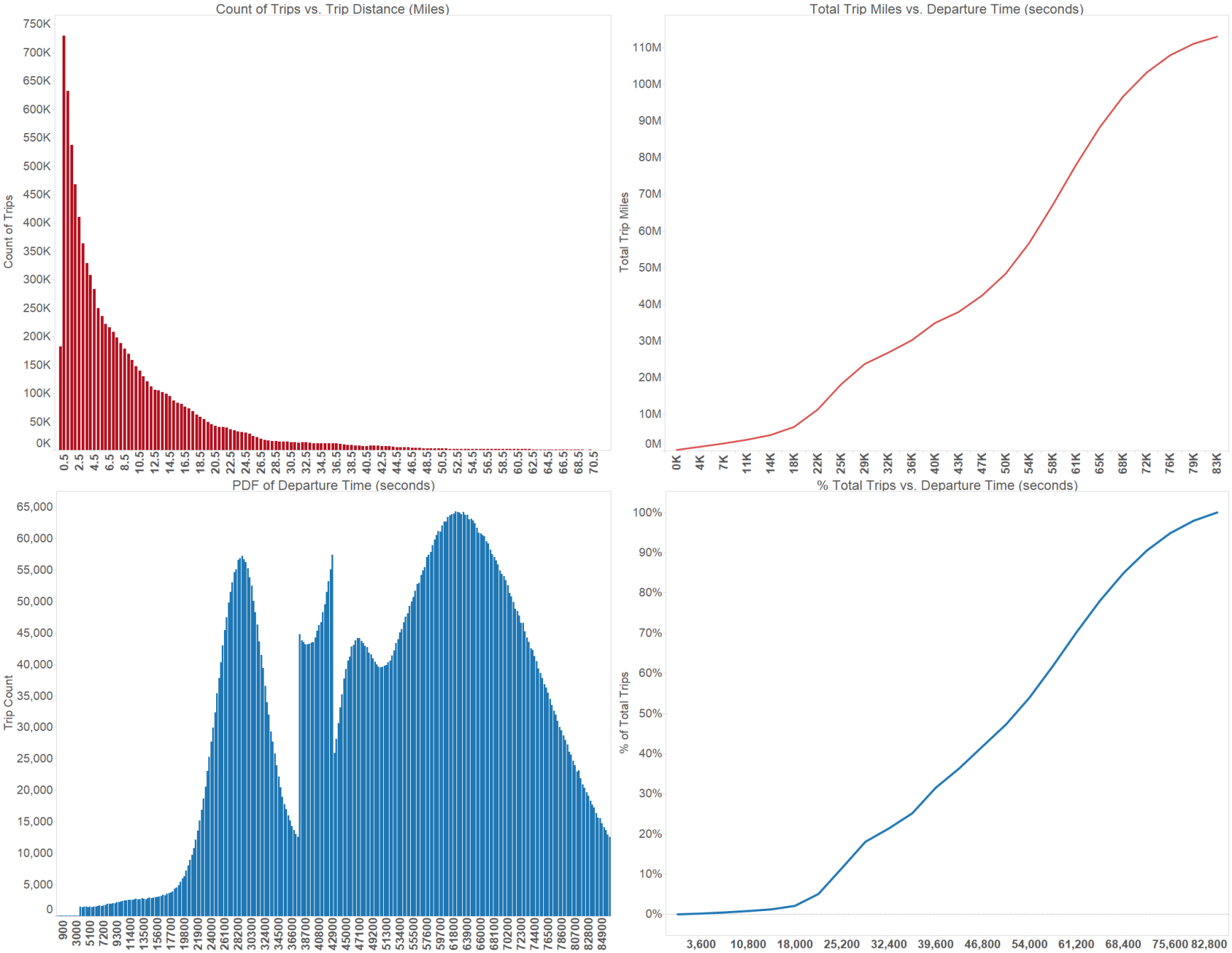

Figure A.9: Arkansas: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

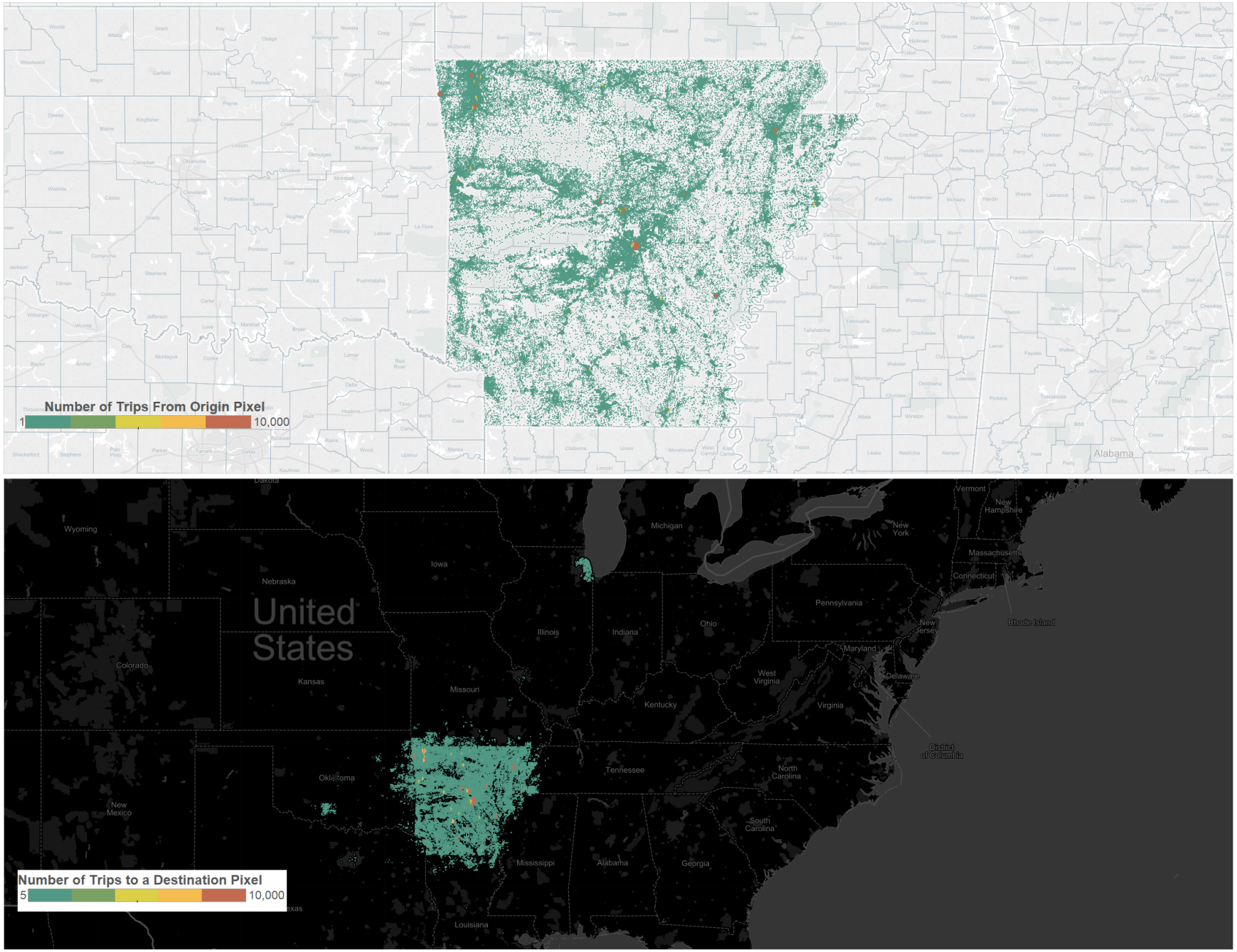

Figure A.10: Arkansas: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

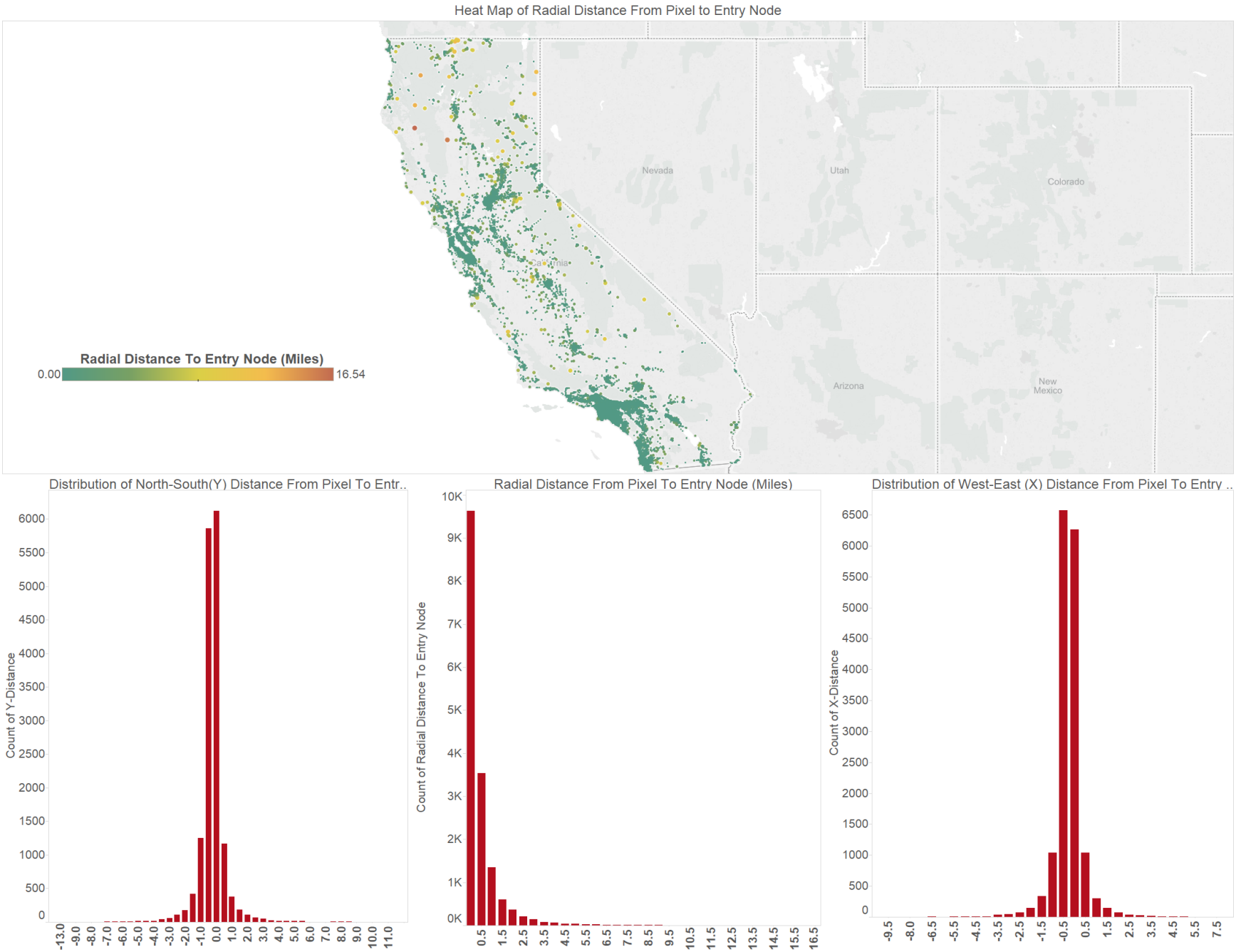

Figure A.11: California: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

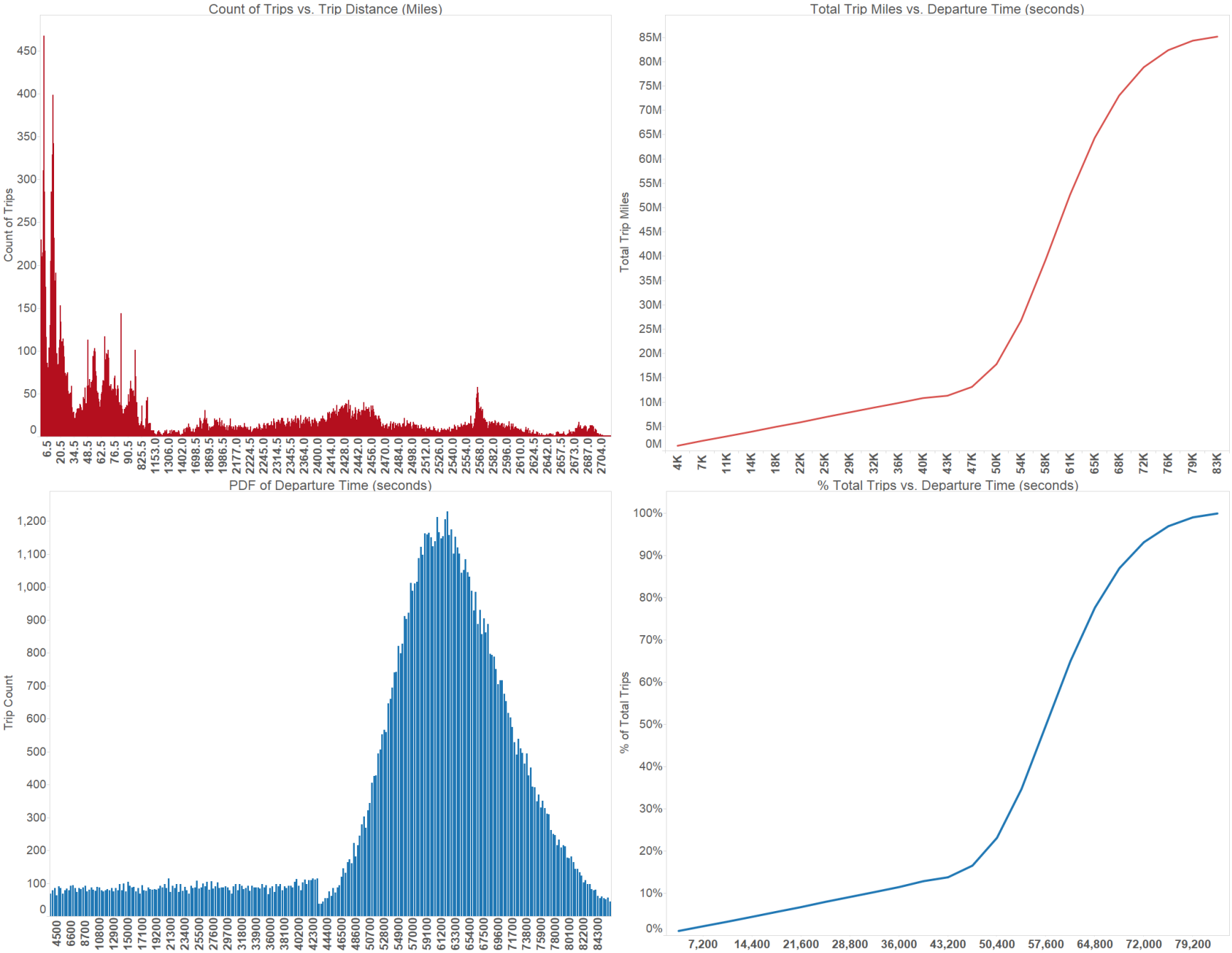

Figure A.12: California: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

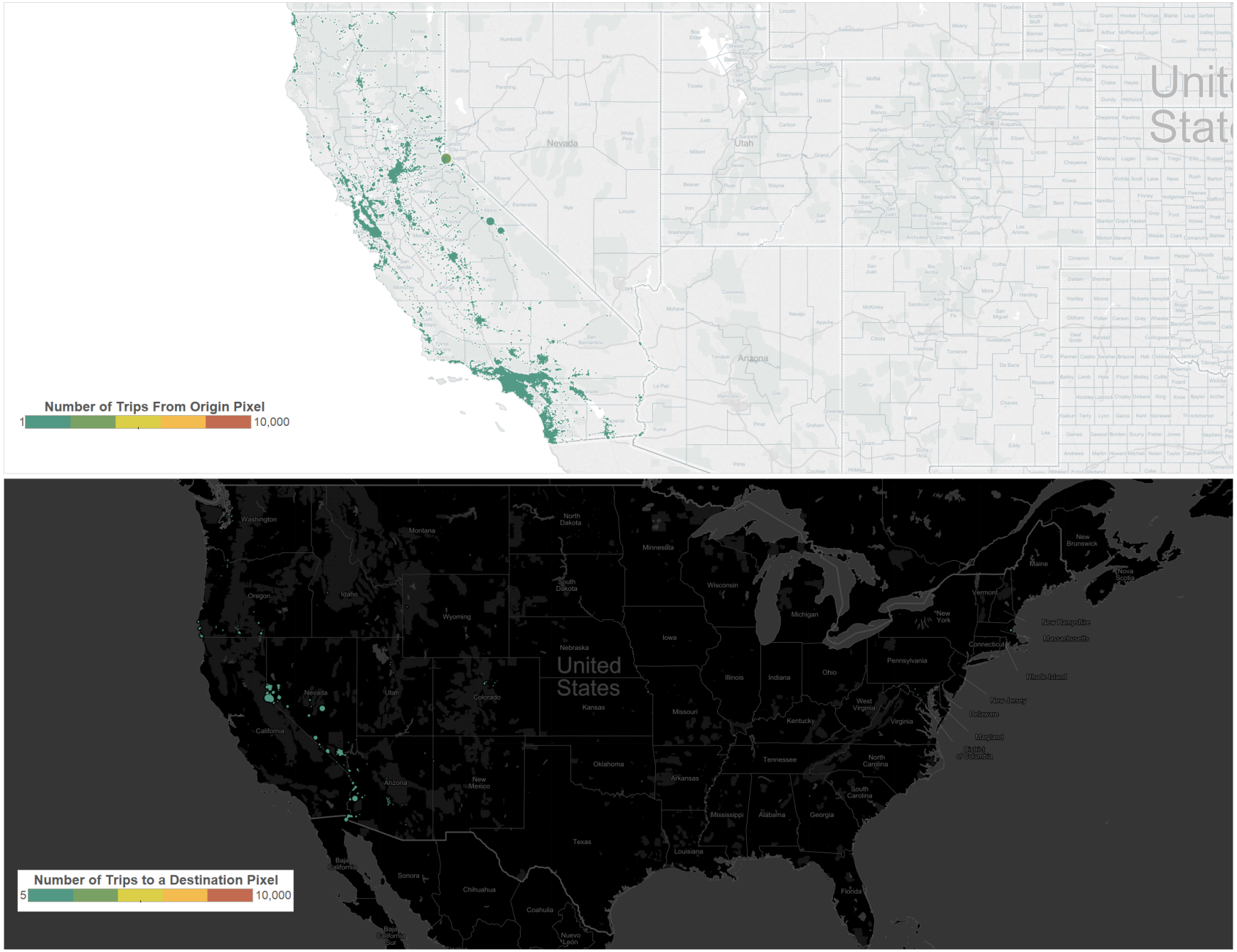

Figure A.13: California: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

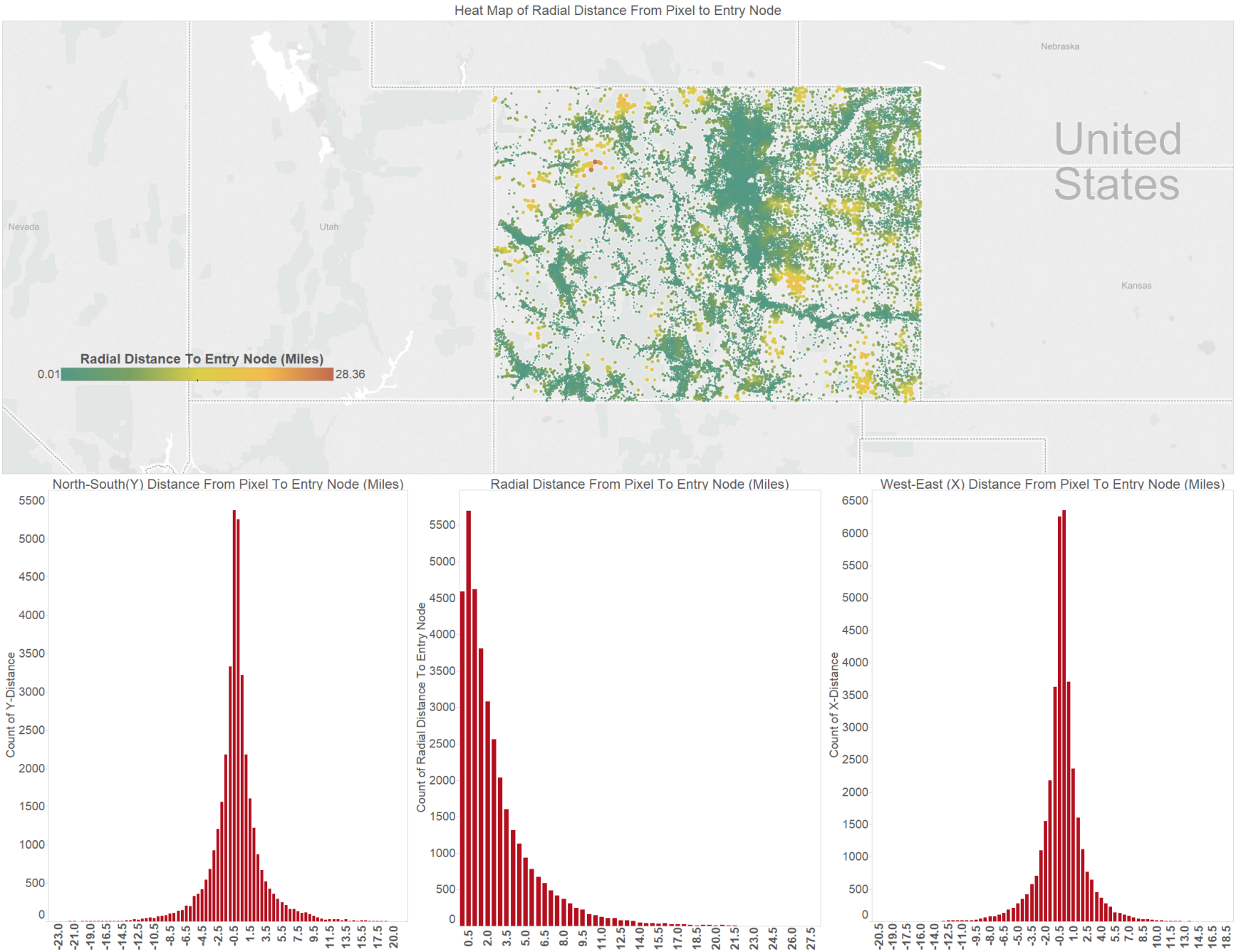

Figure A.14: Colorado: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

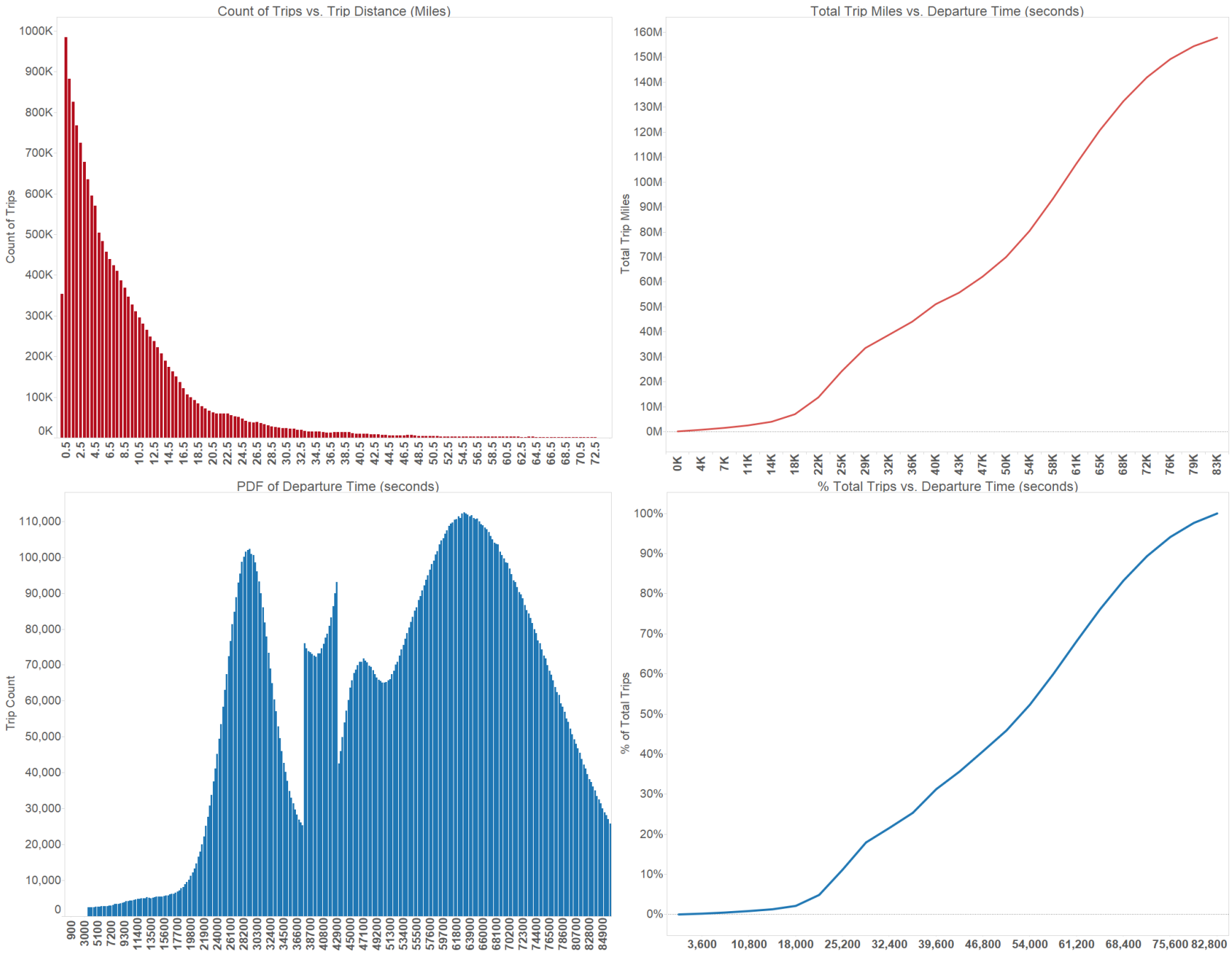

Figure A.15: Colorado: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

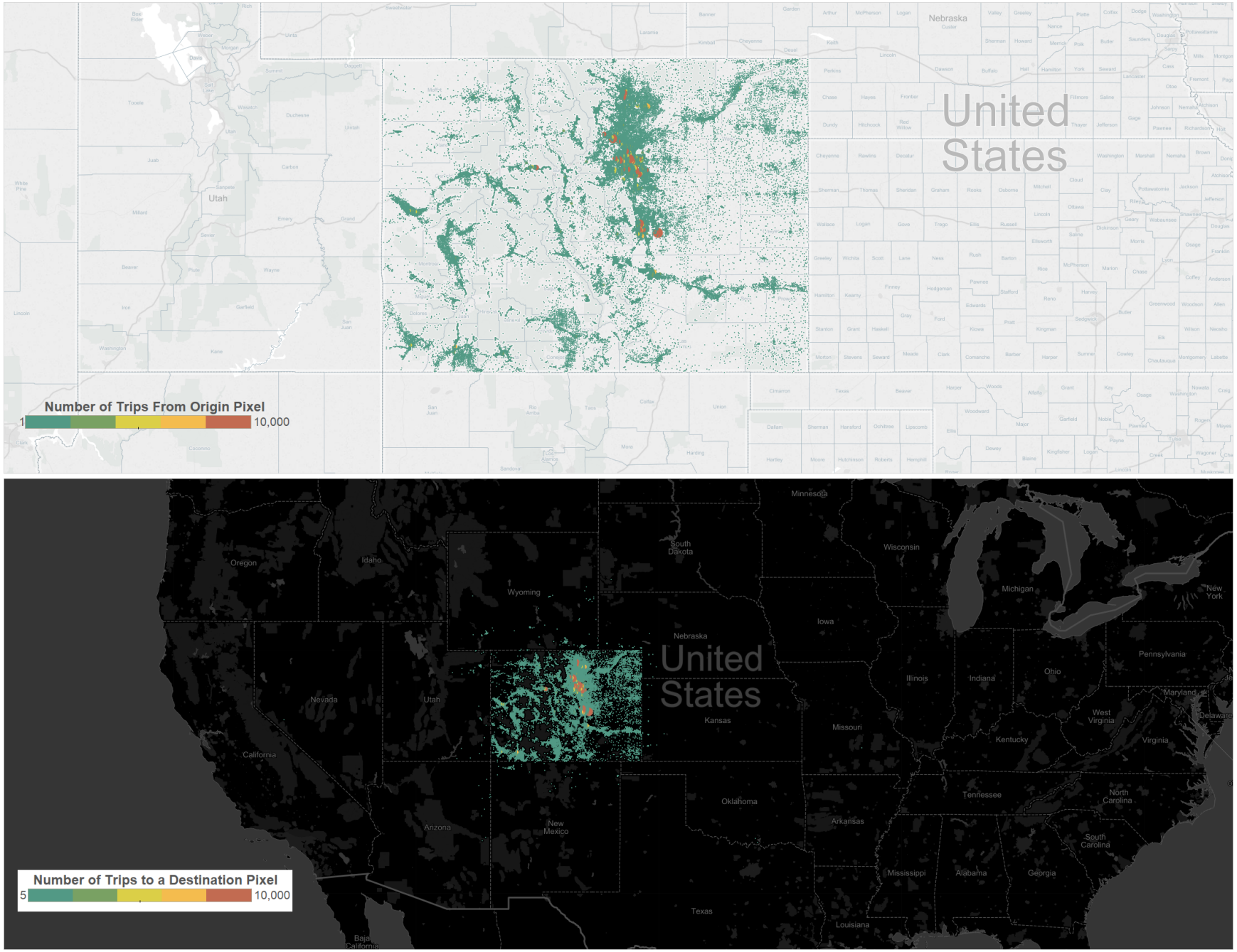

Figure A.16: Colorado: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

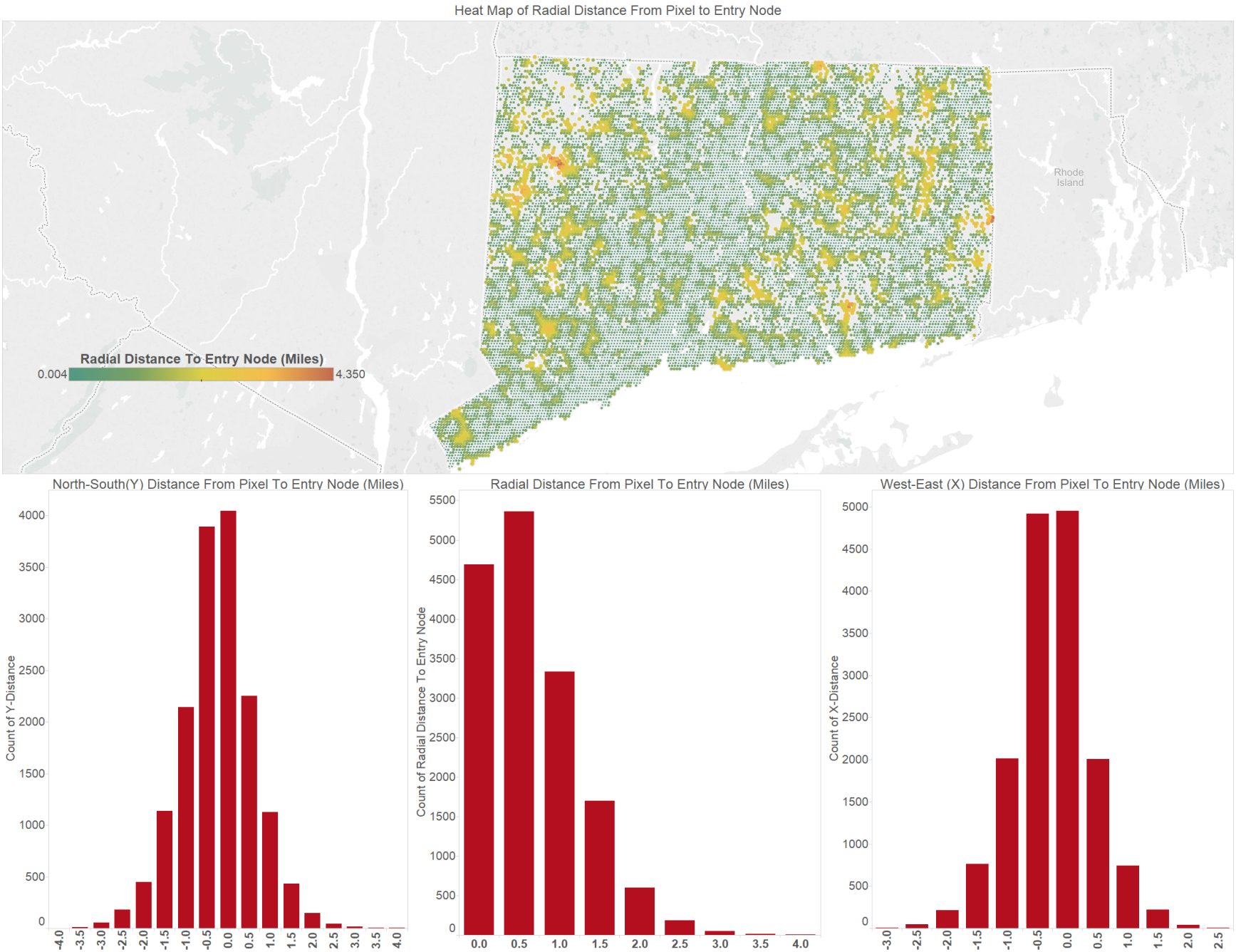

Figure A.17: Connecticut: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

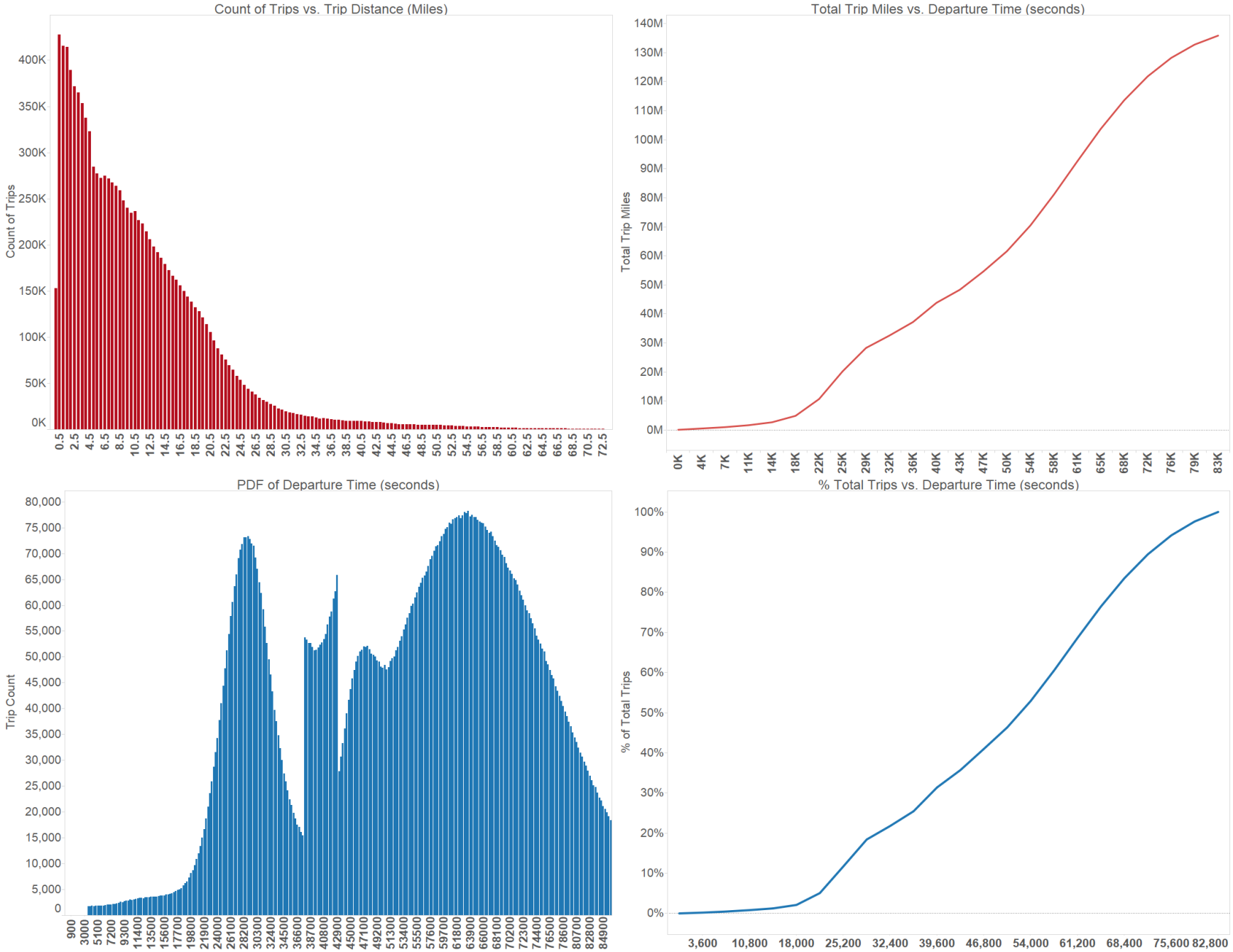

Figure A.18: Connecticut: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

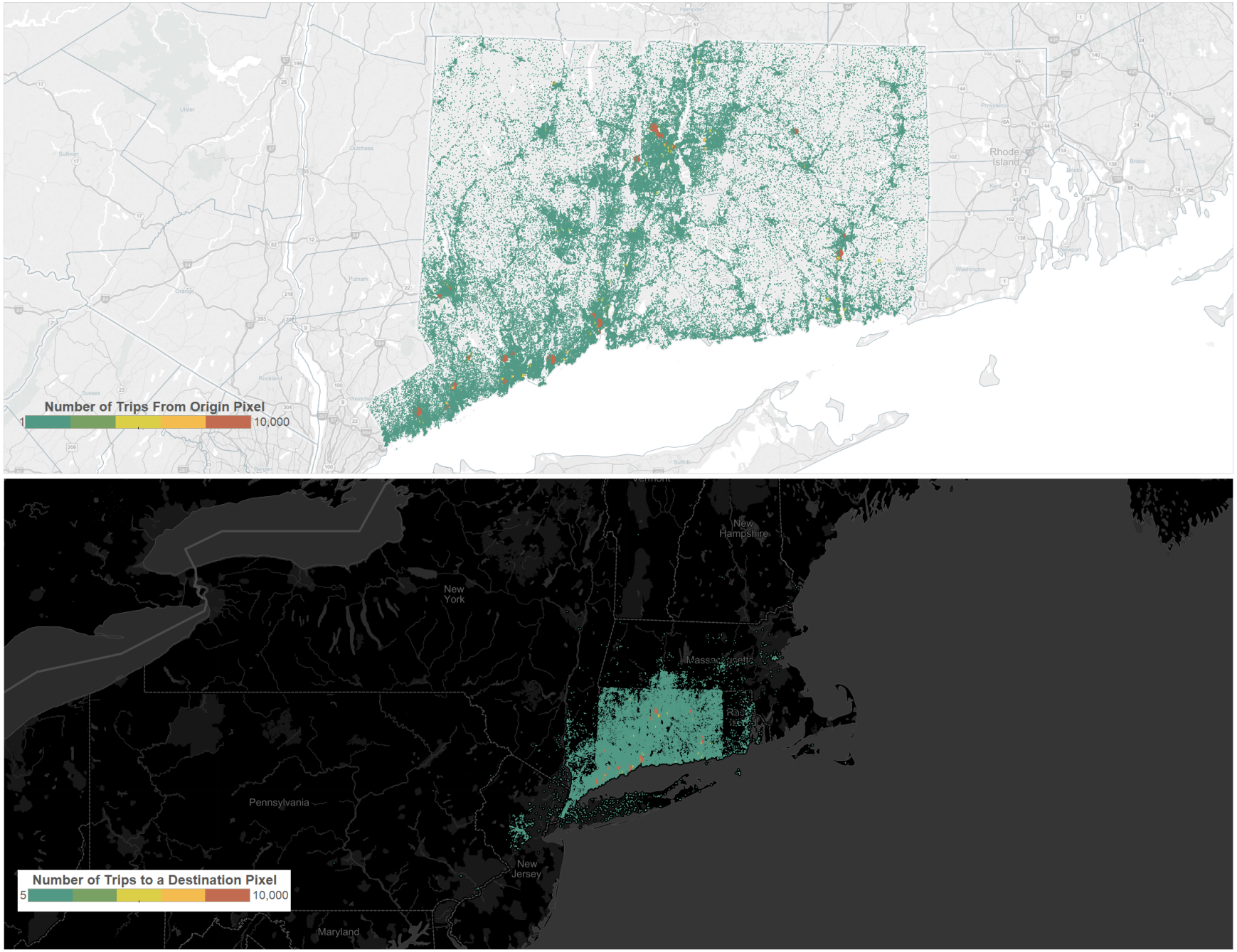

Figure A.19: Connecticut: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

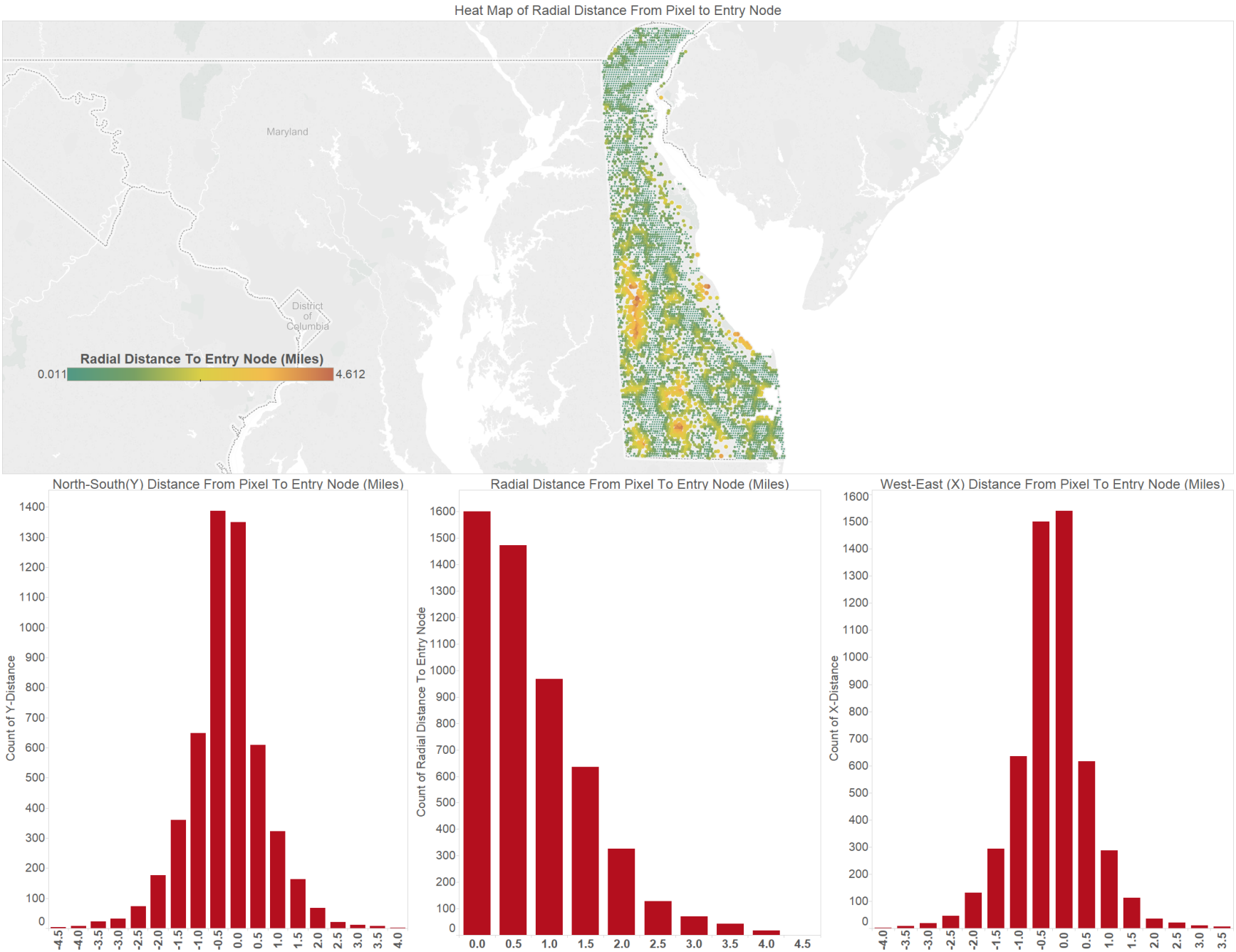

Figure A.20: Delaware: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

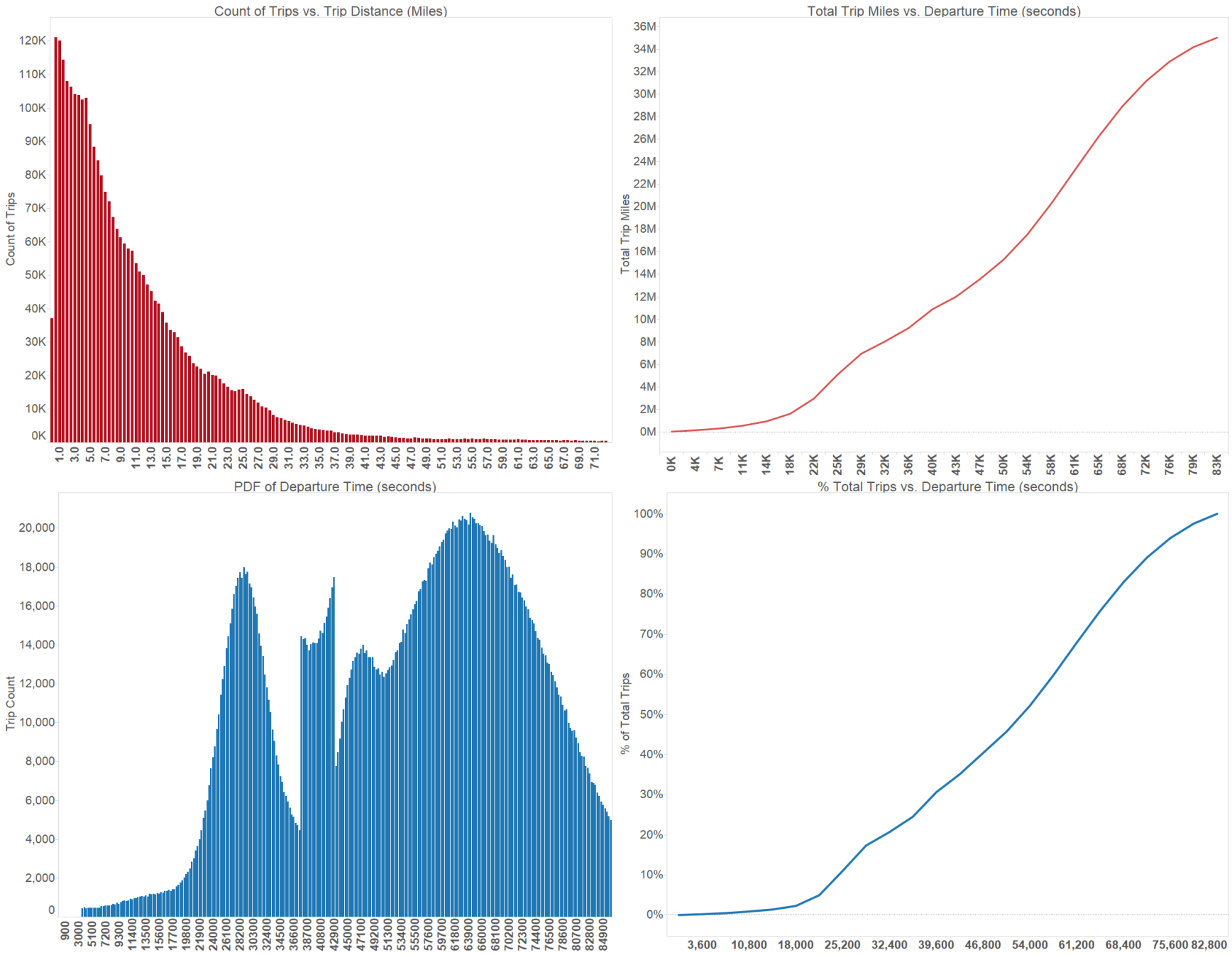

Figure A.21: Delaware: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

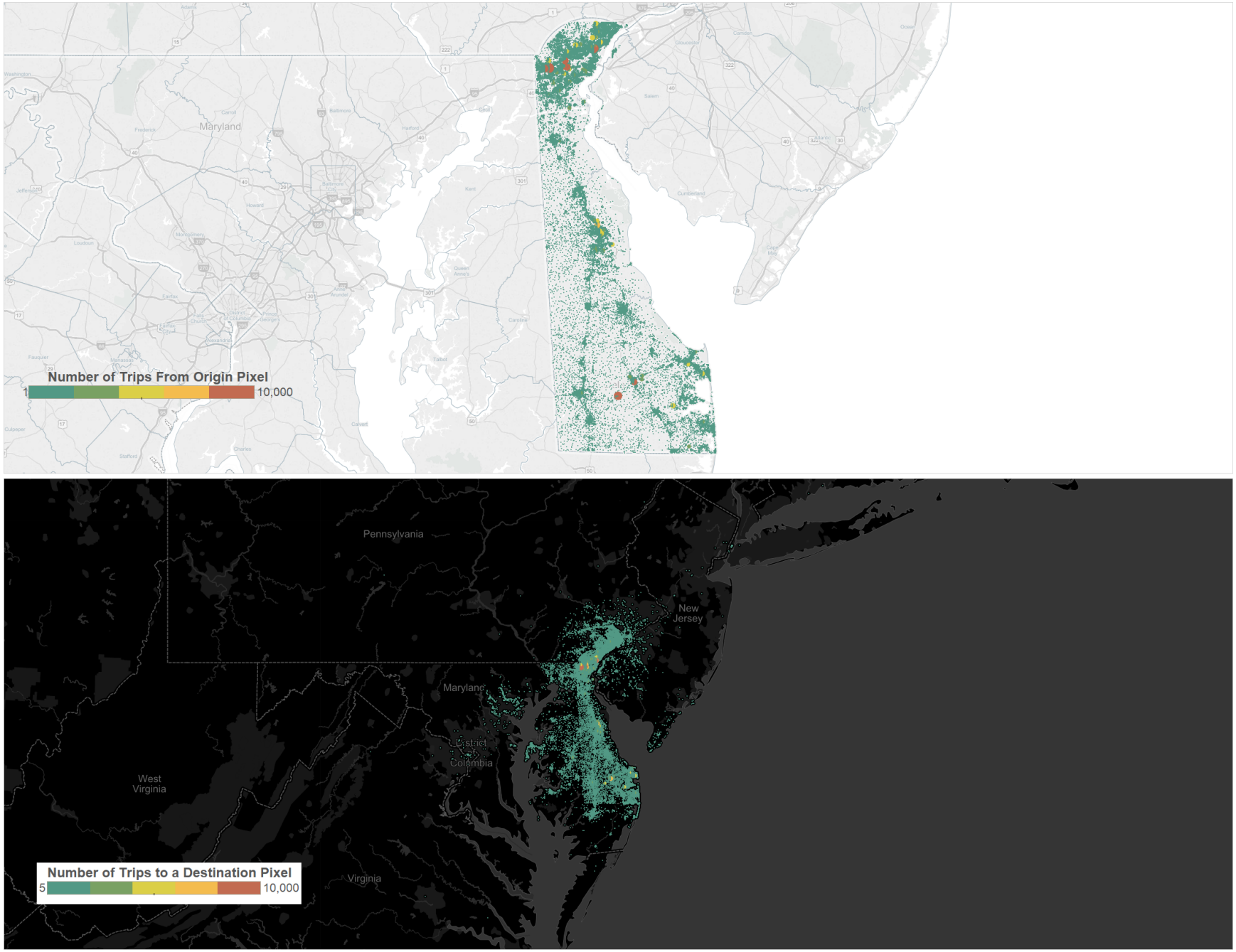

Figure A.22: Delaware: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

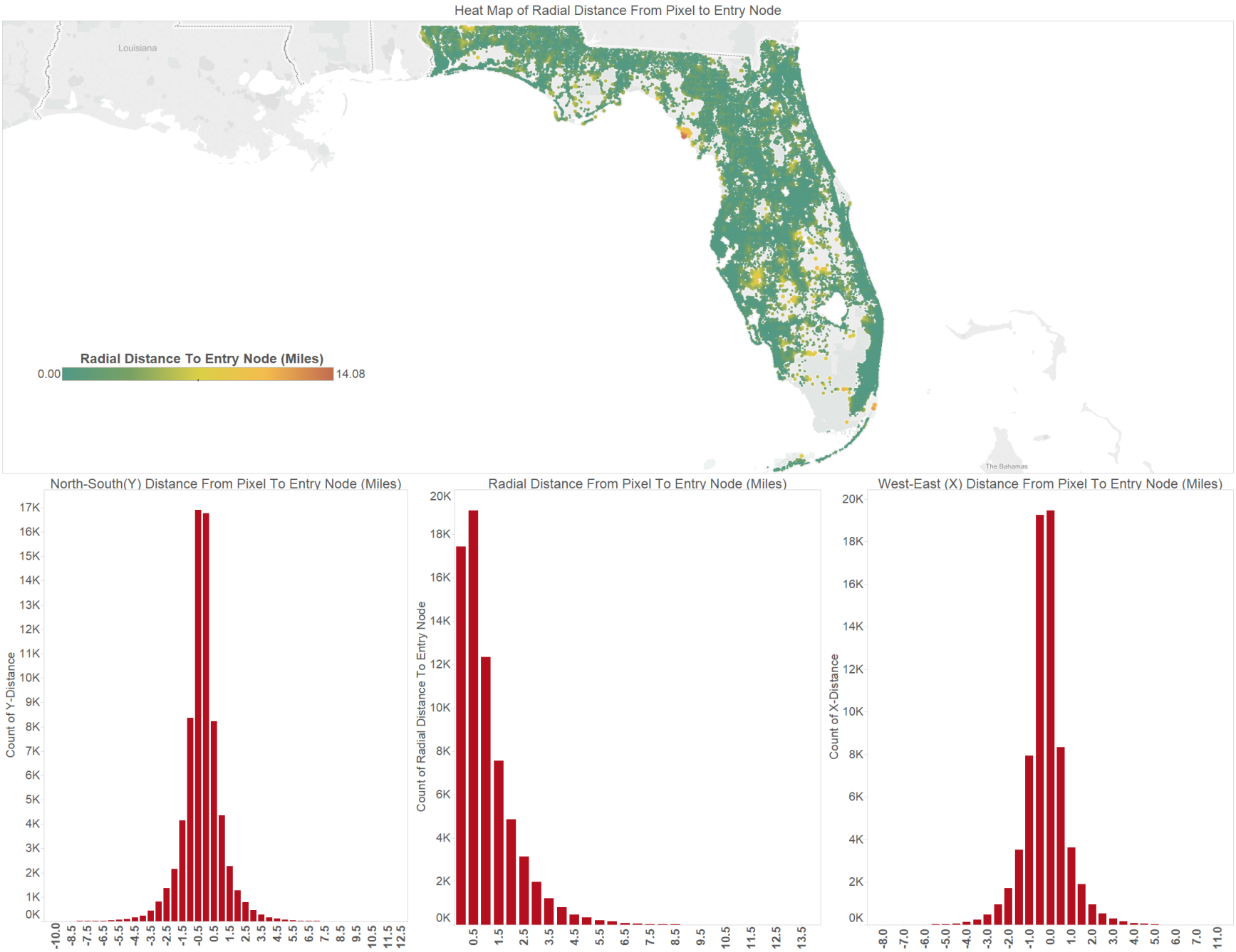

Figure A.23: Florida: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

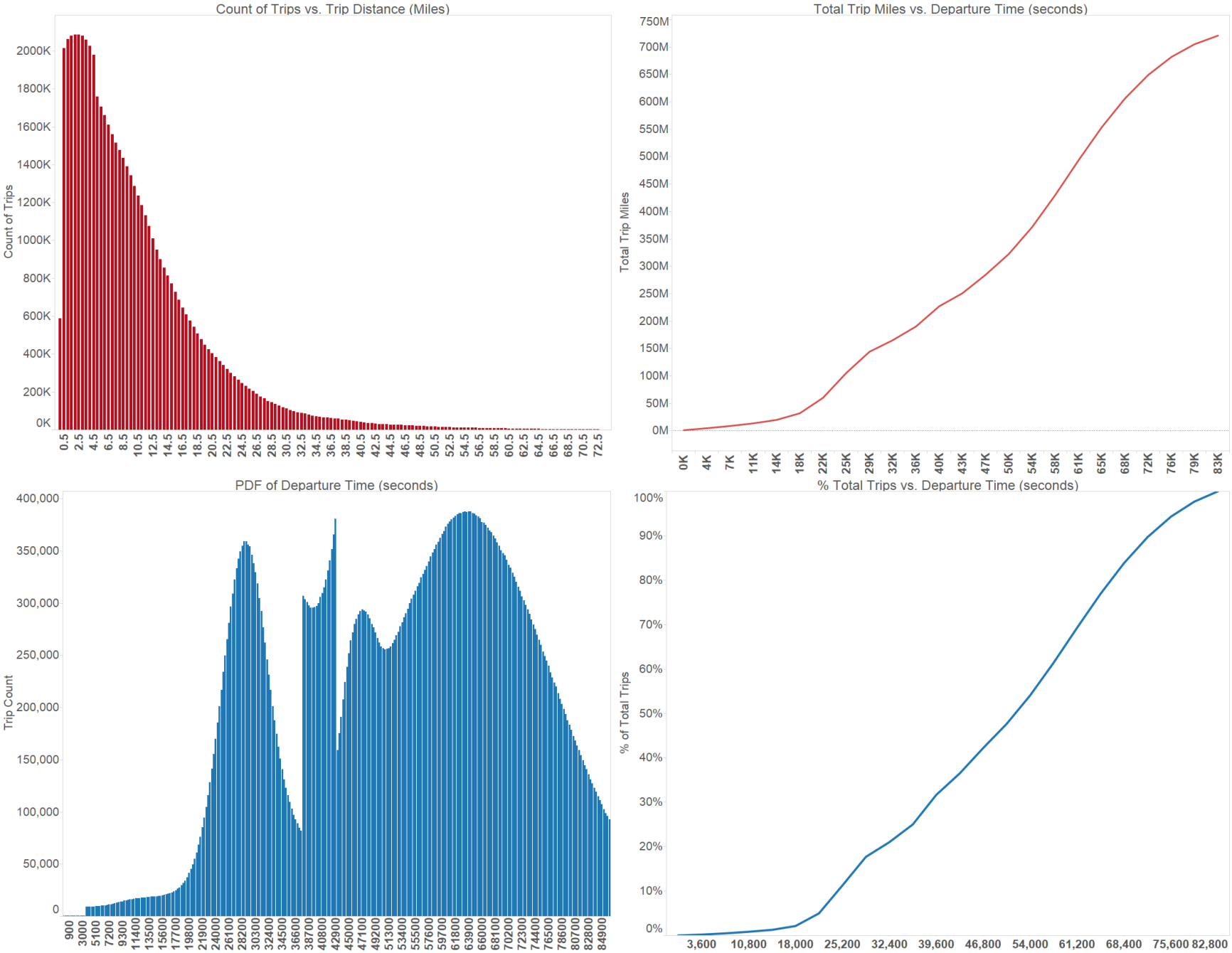

Figure A.24: Florida: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

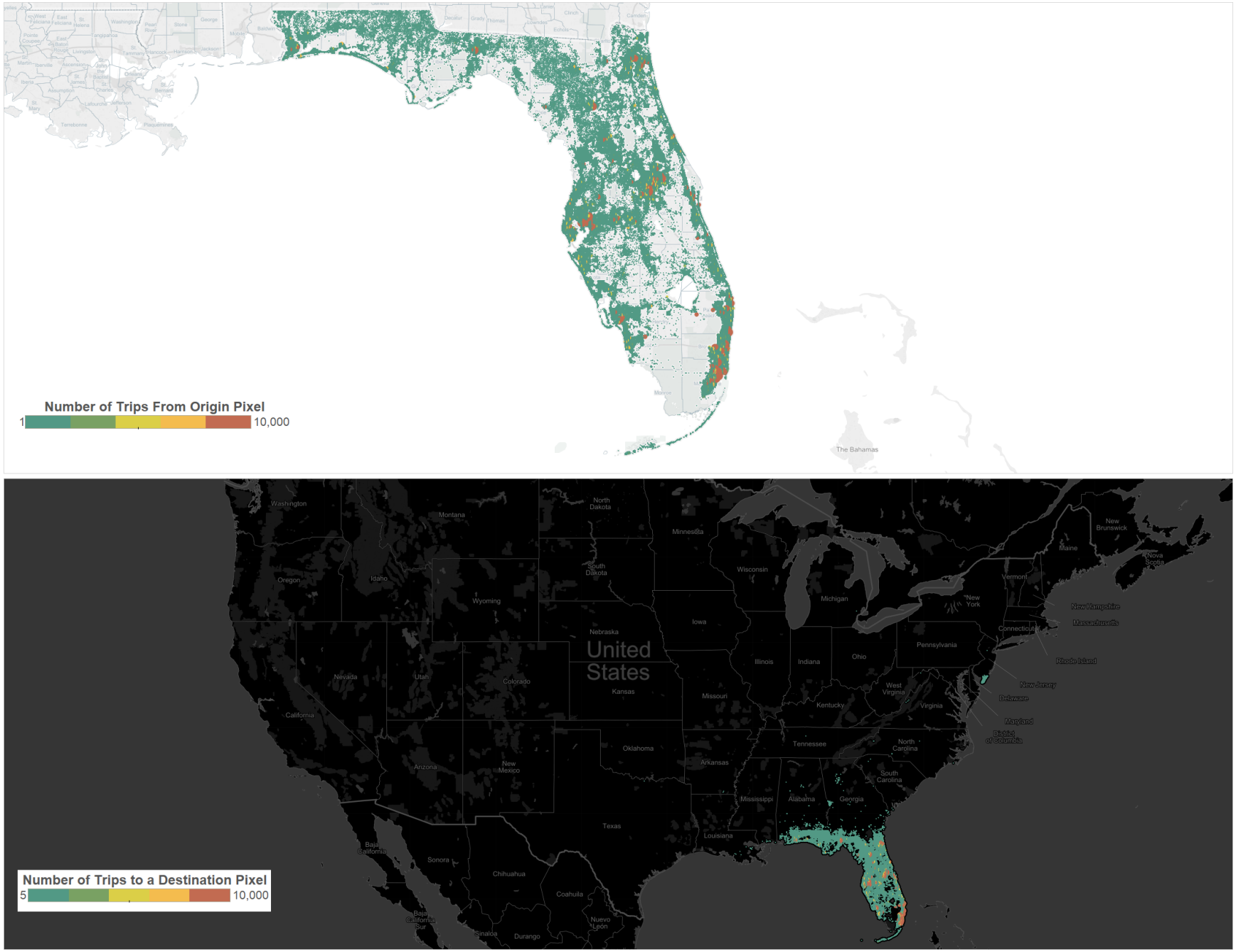

Figure A.25: Florida: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

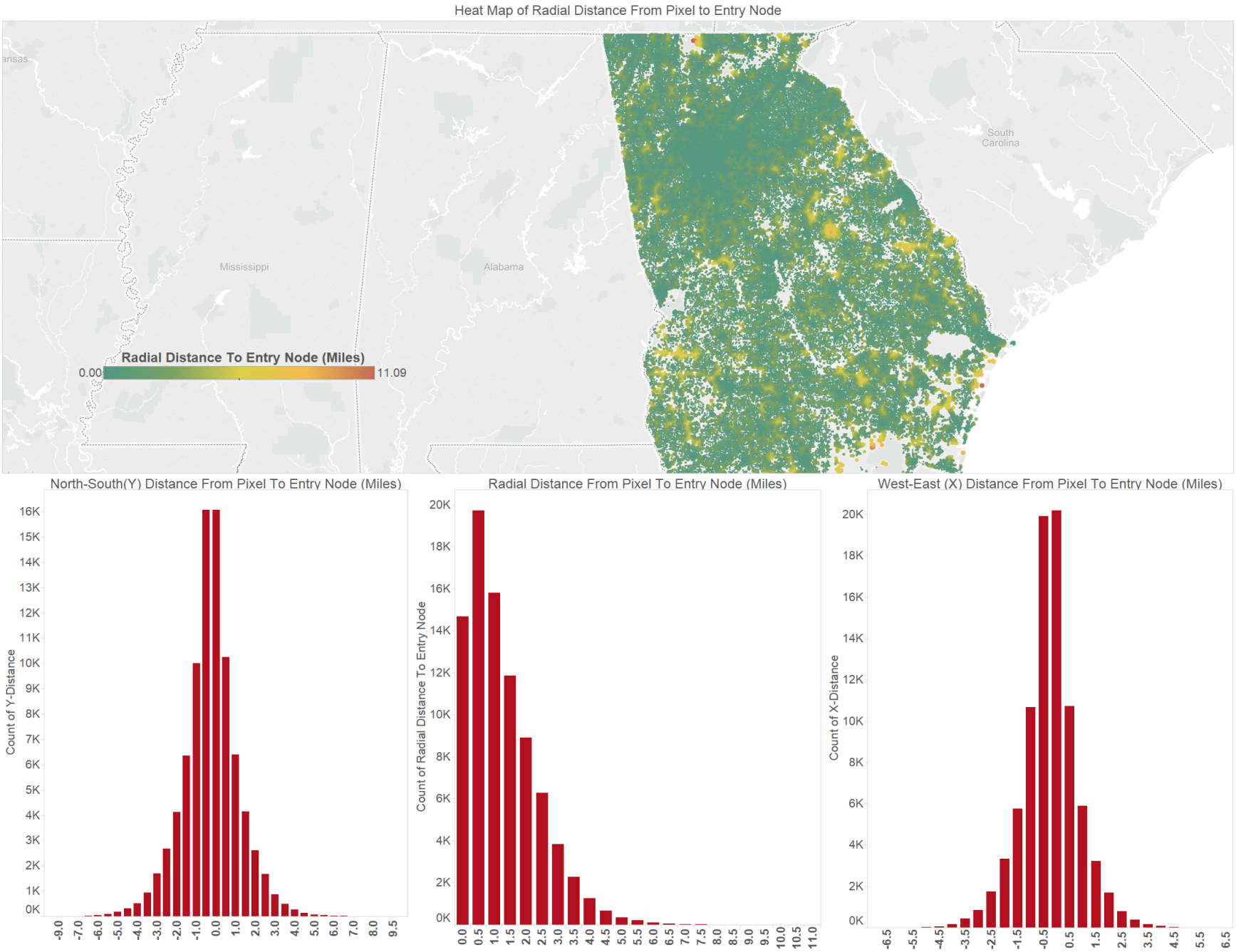

Figure A.26: Georgia: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

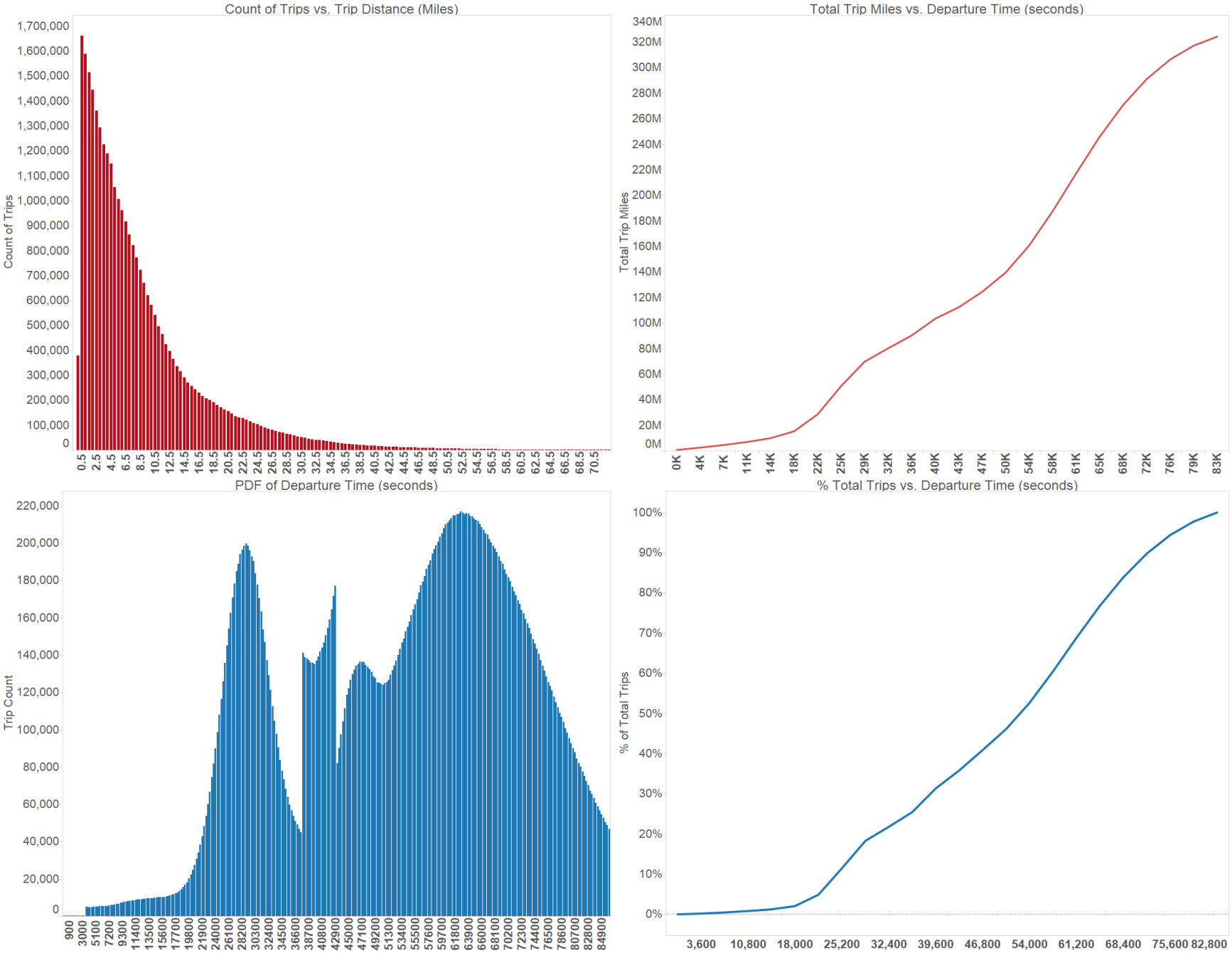

Figure A.27: Georgia: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

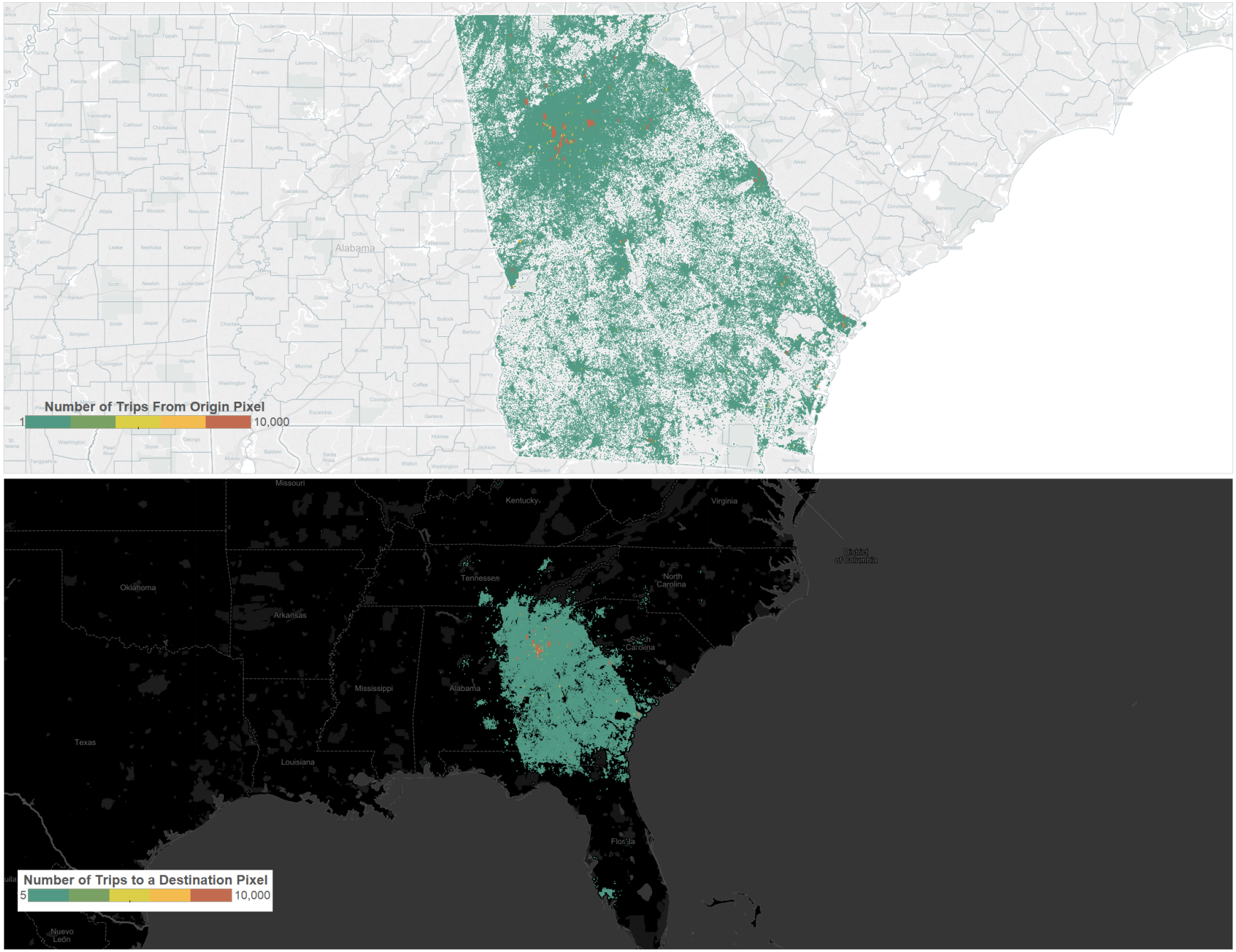

Figure A.28: Georgia: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

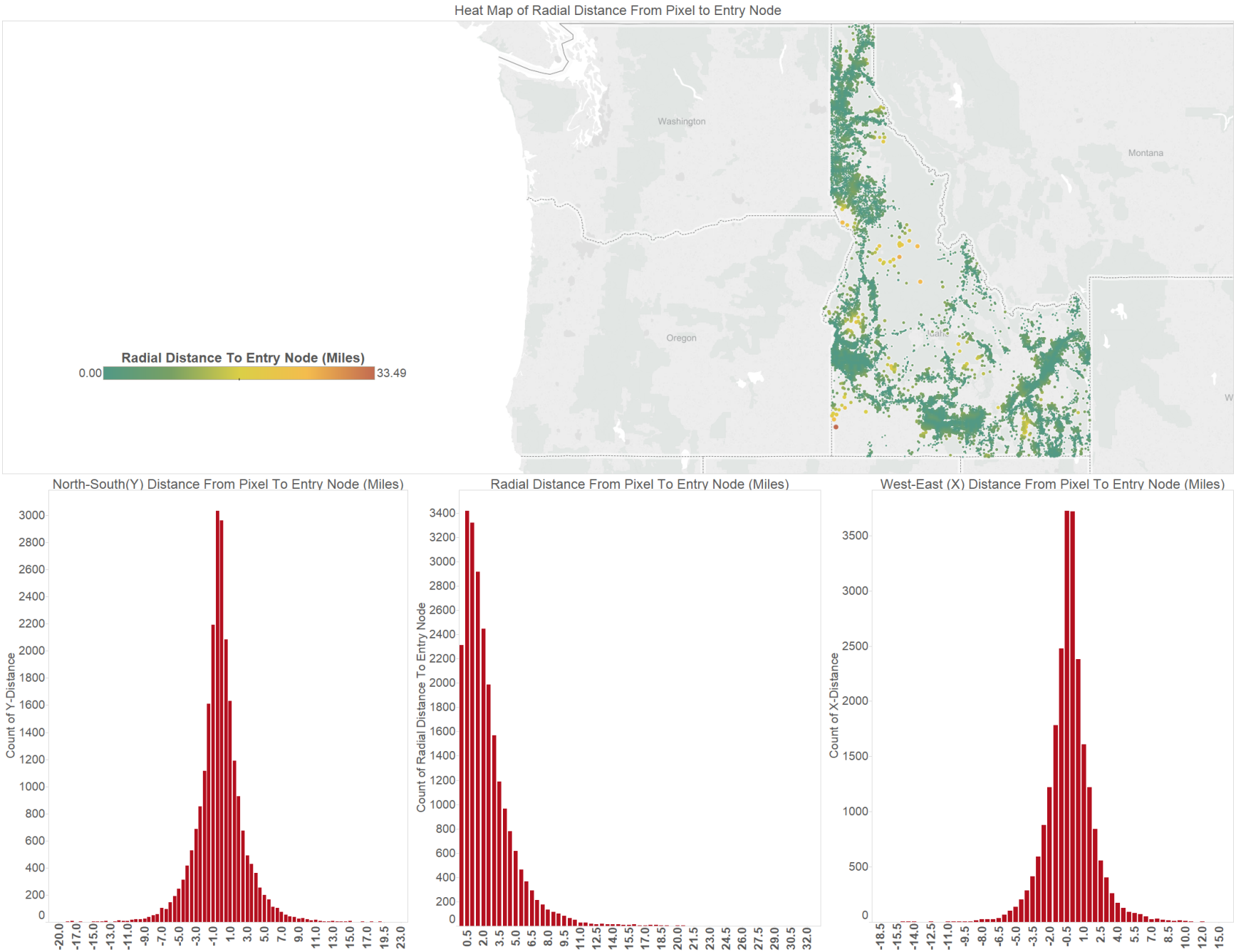

Figure A.29: Idaho: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

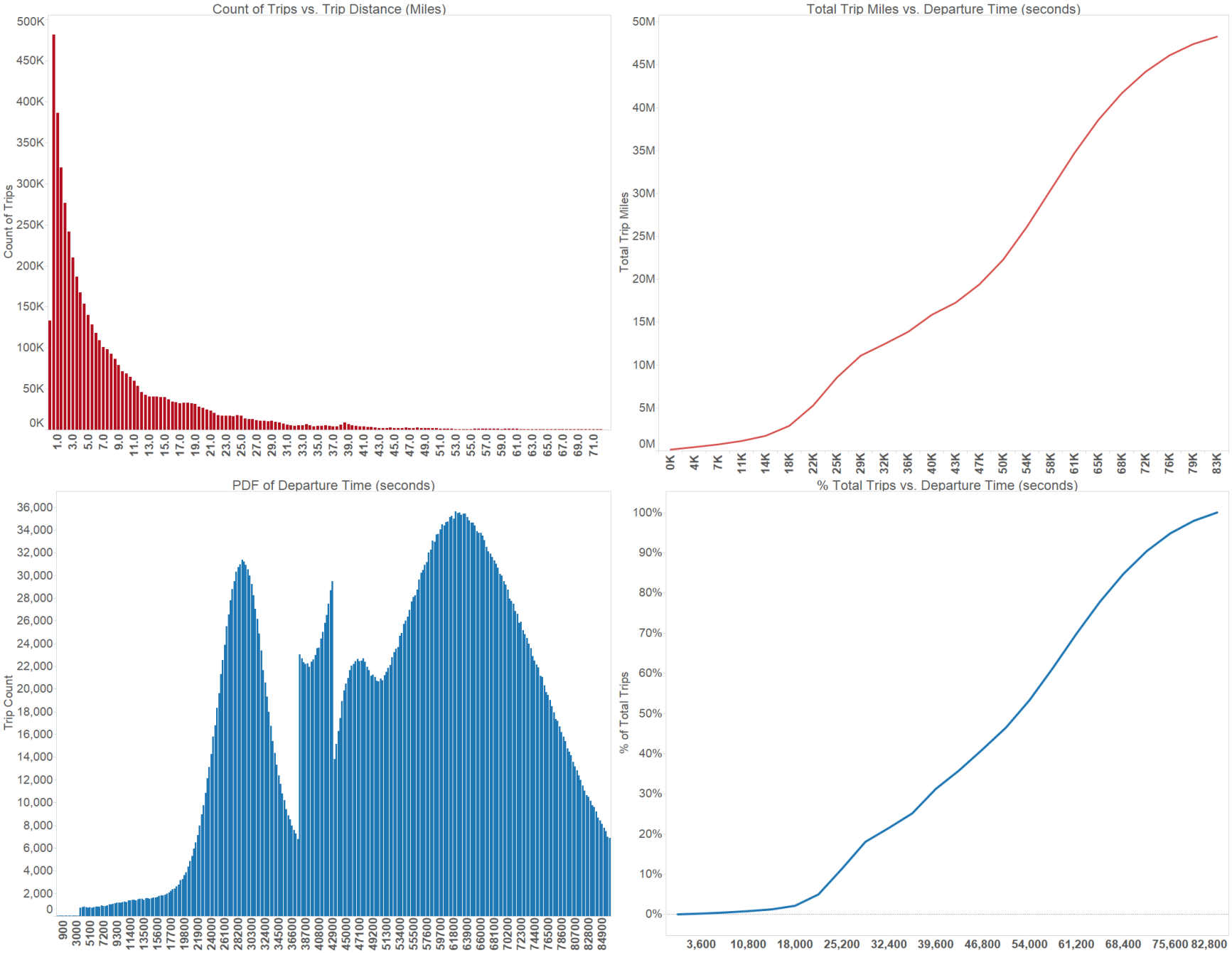

Figure A.30: Idaho: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

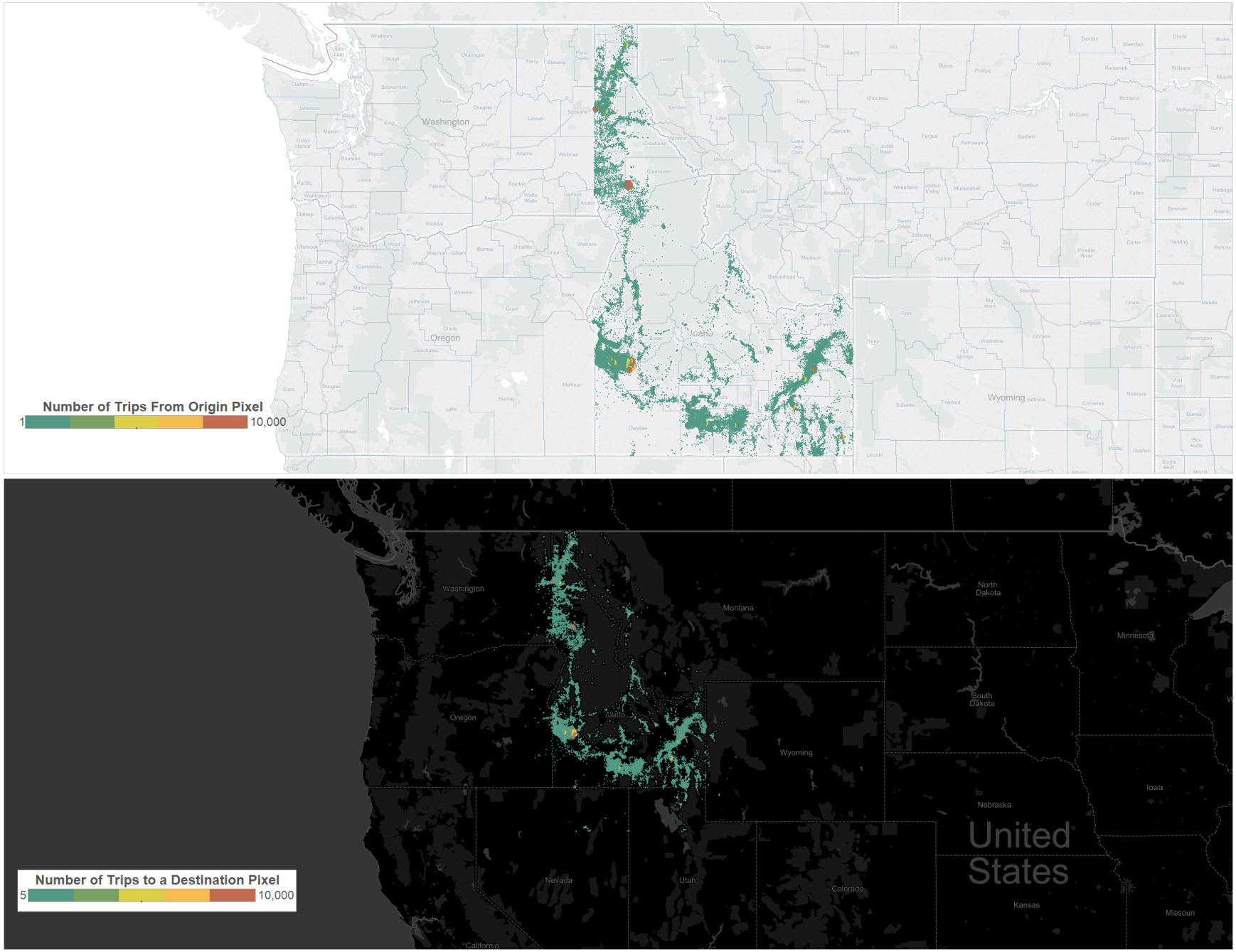

Figure A.31: Idaho: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

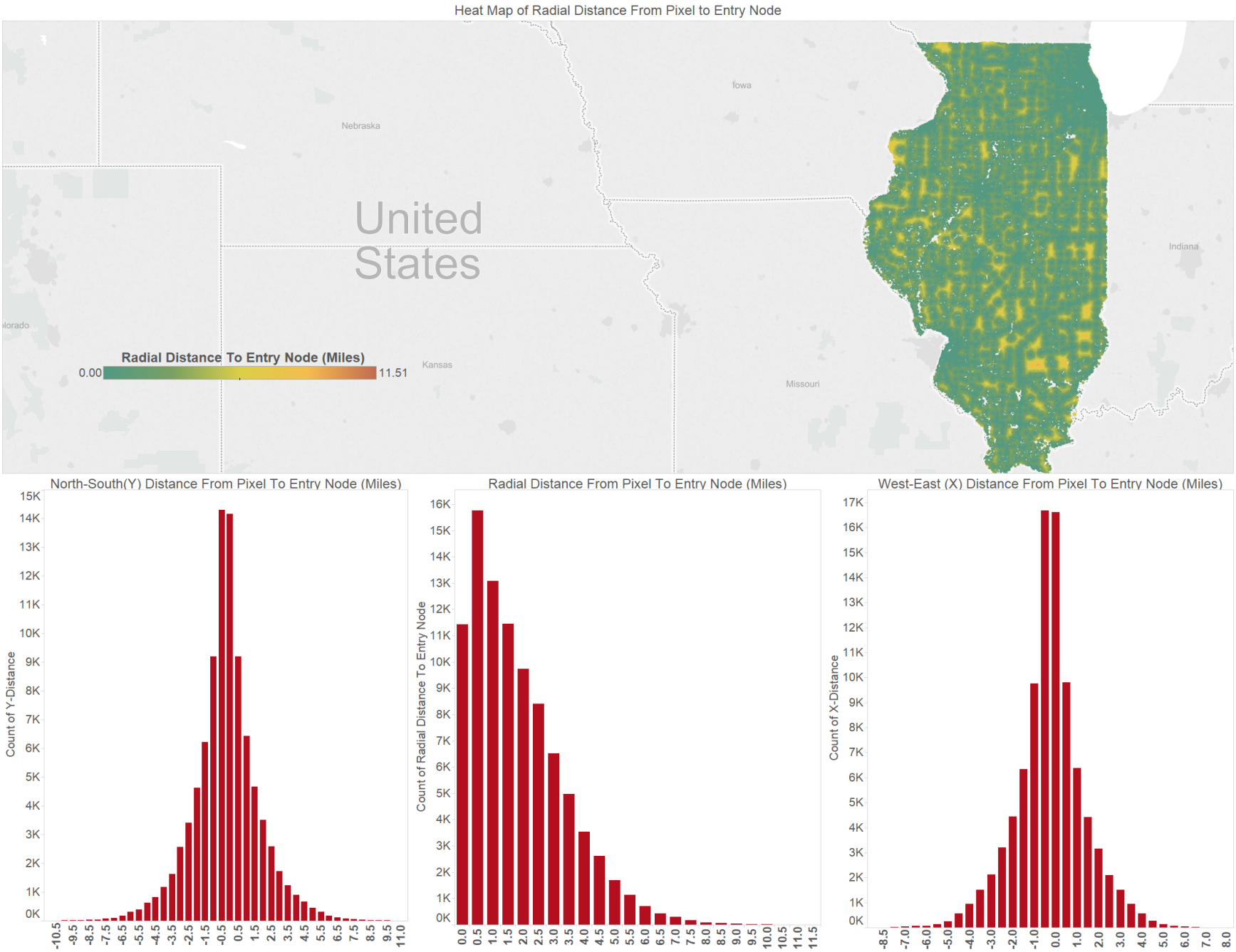

Figure A.32: Illinois: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

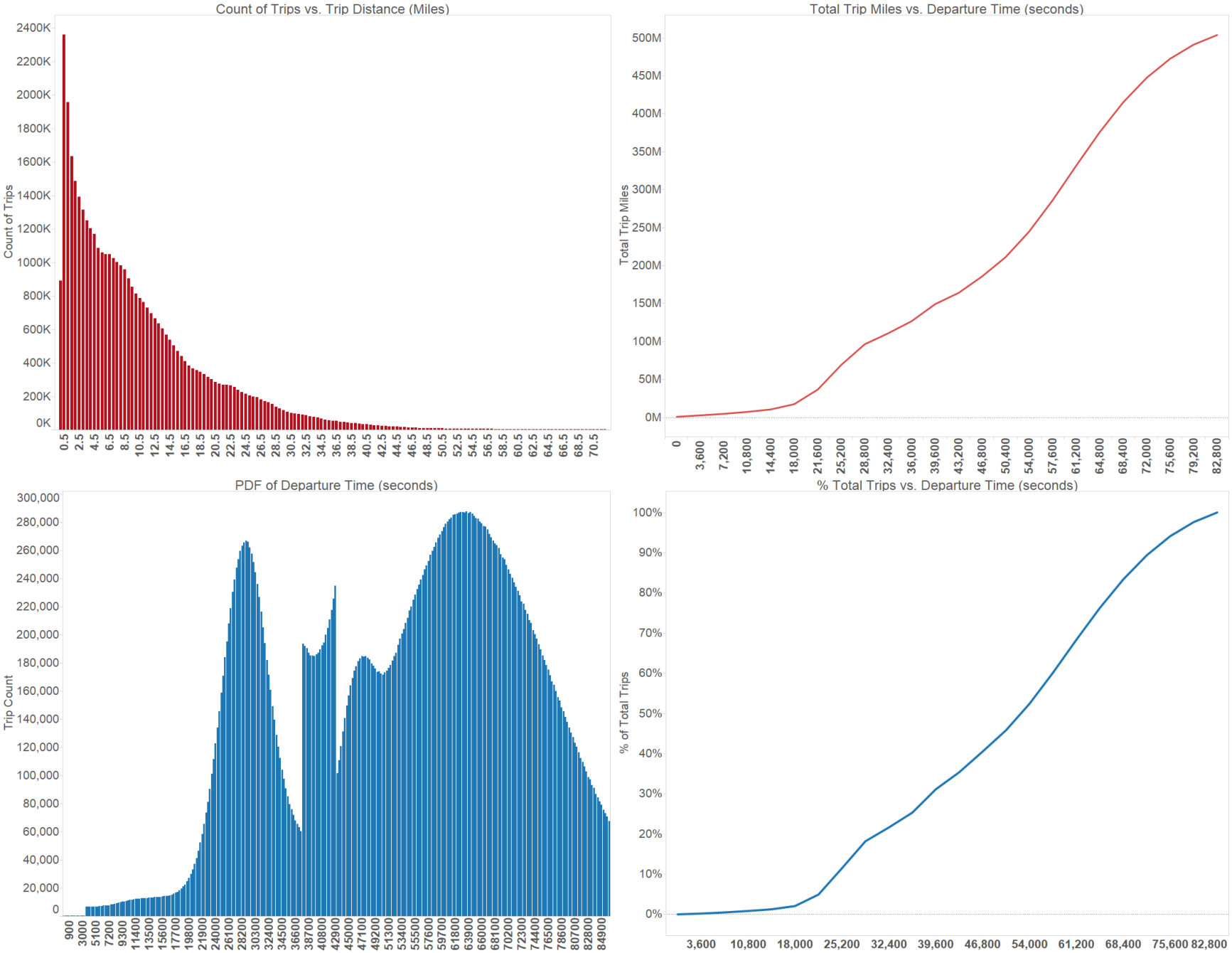

Figure A.33: Illinois: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

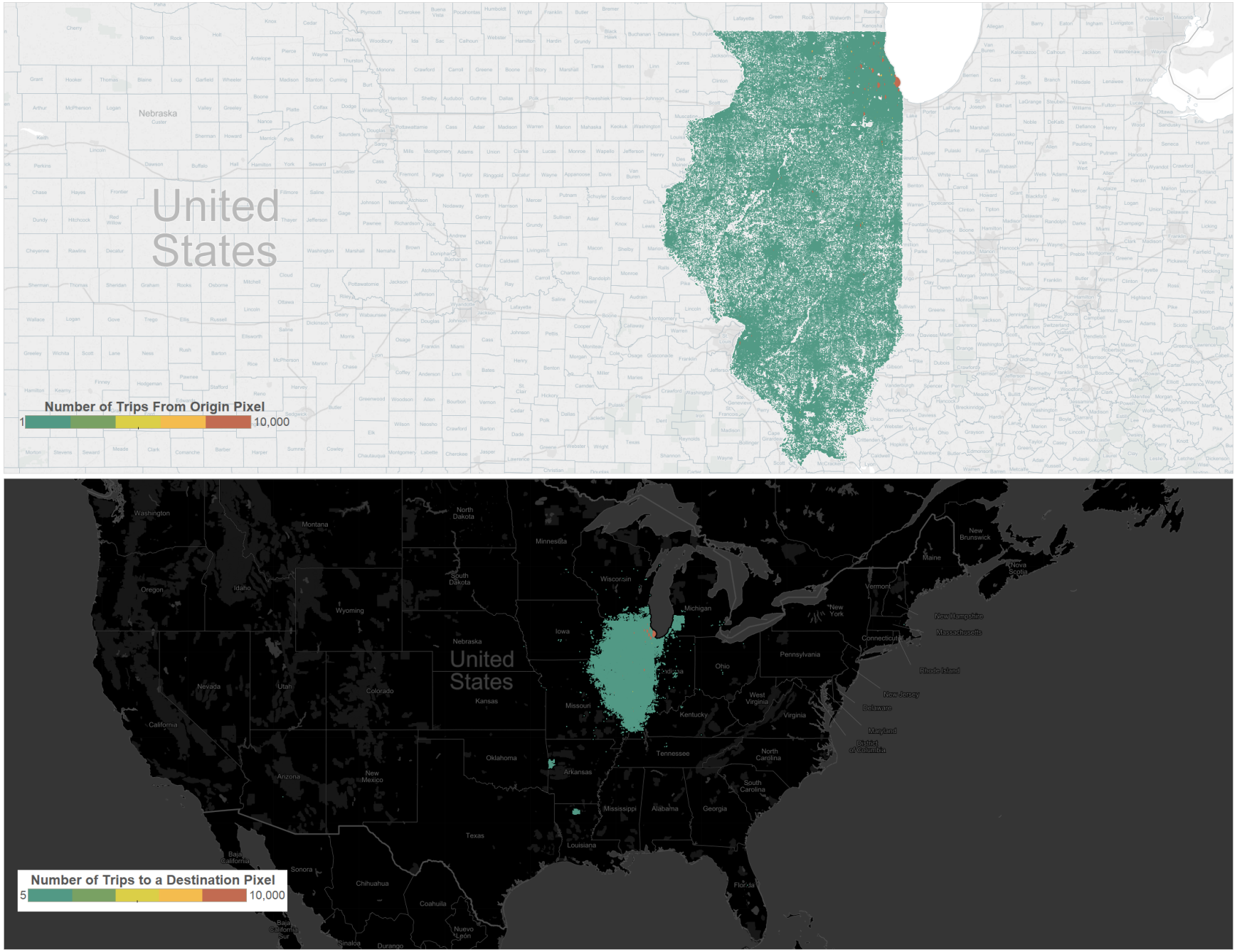

Figure A.34: Illinois: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

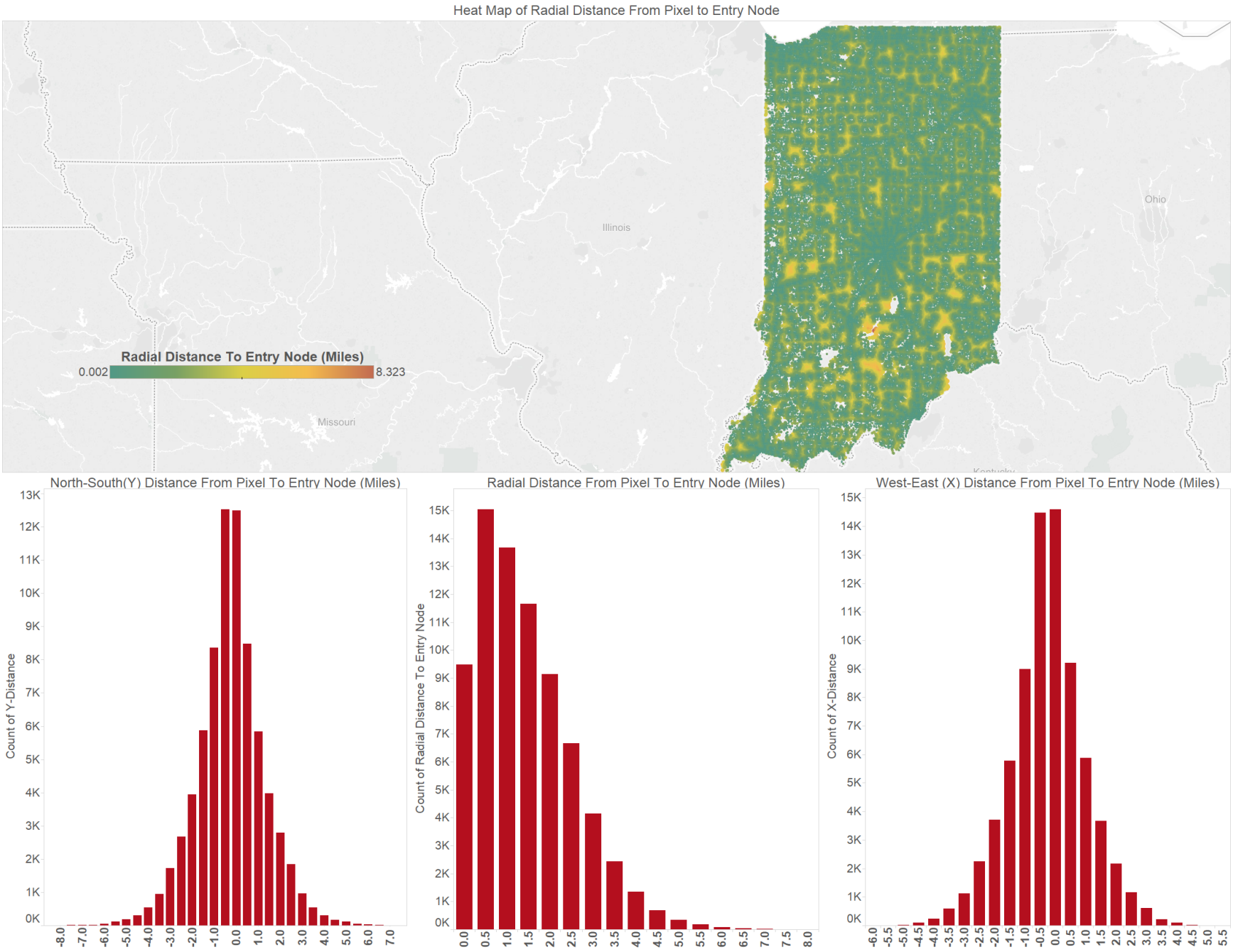

Figure A.35: Indiana: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

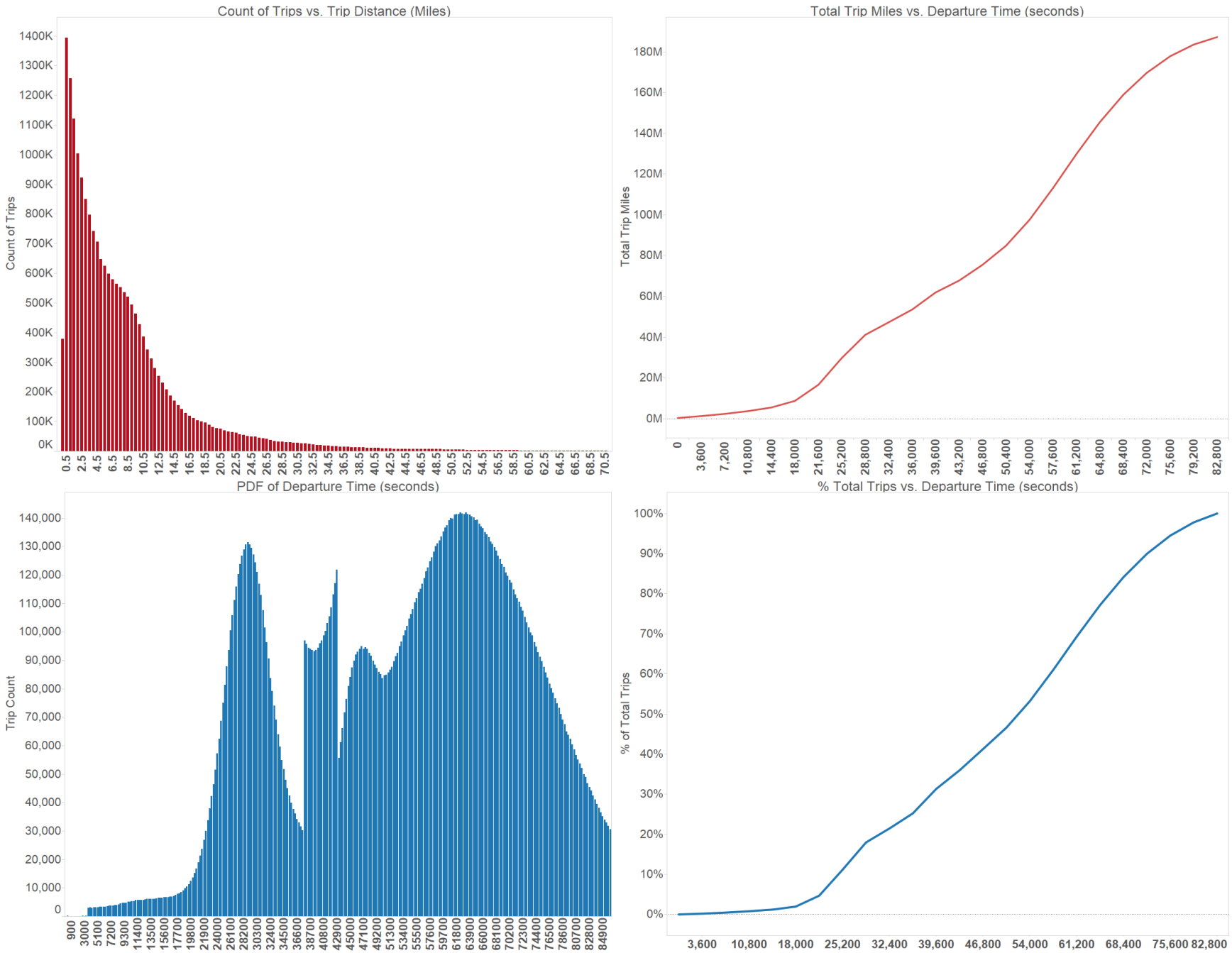

Figure A.36: Indiana: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

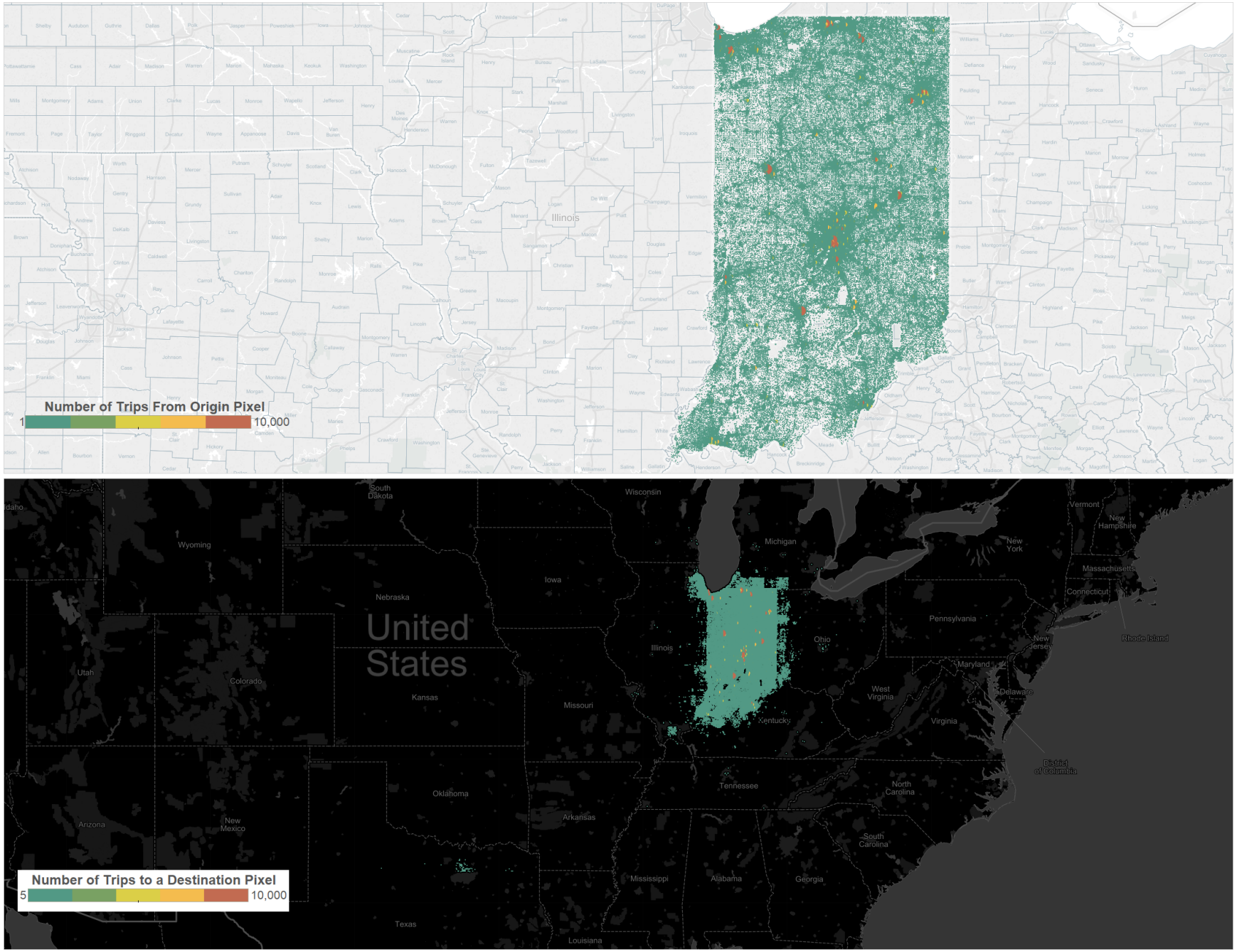

Figure A.37: Indiana: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

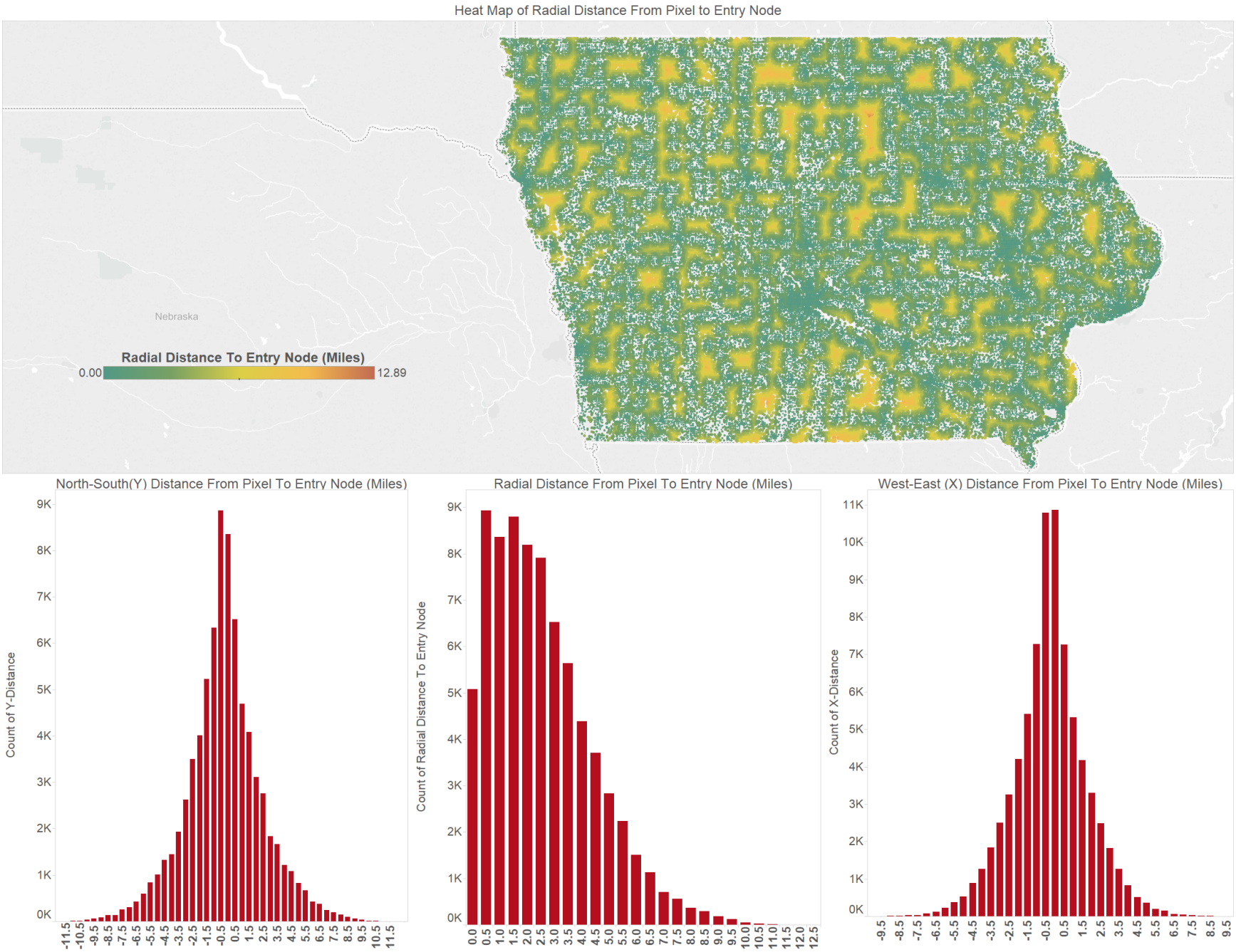

Figure A.38: Iowa: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

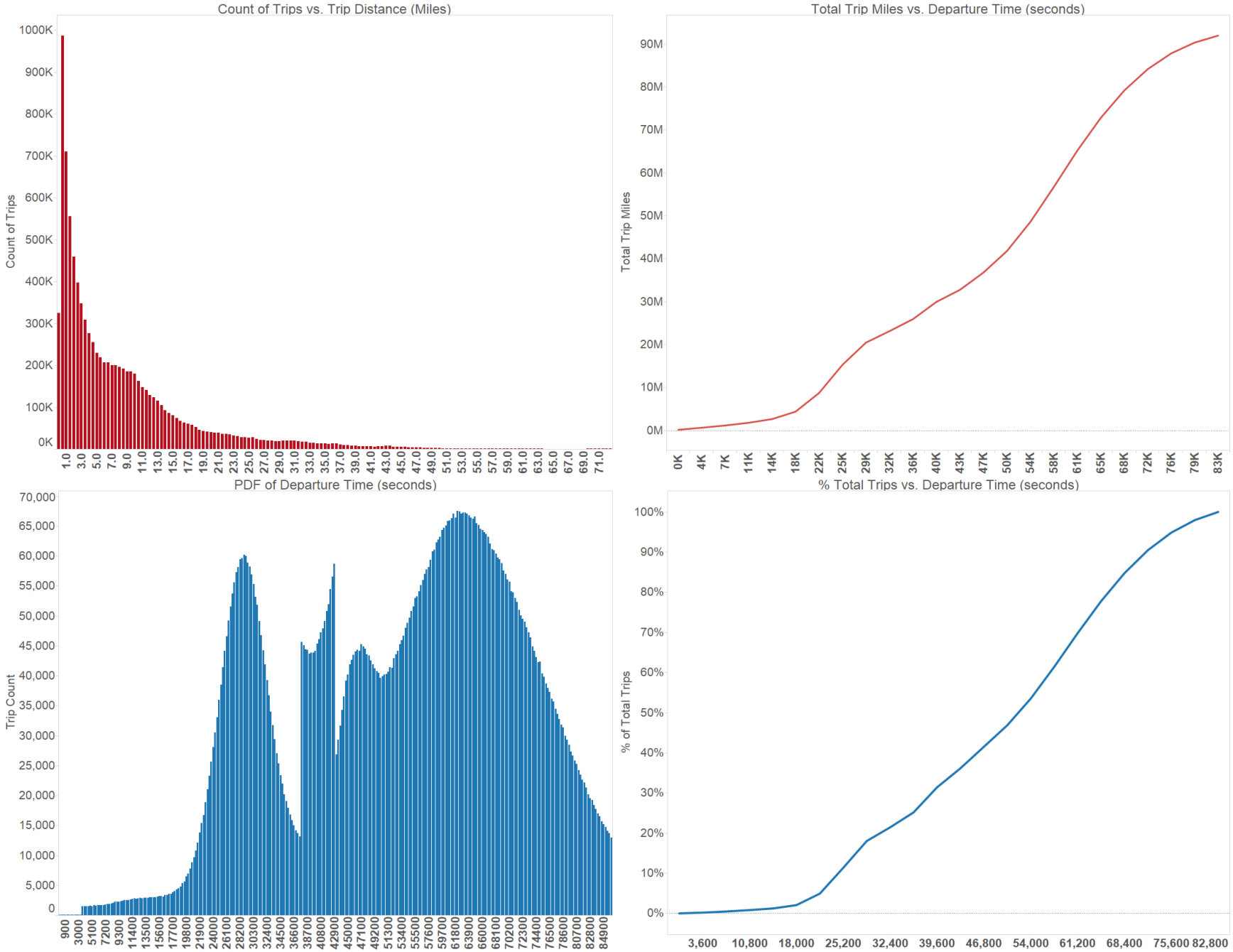

Figure A.39: Iowa: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

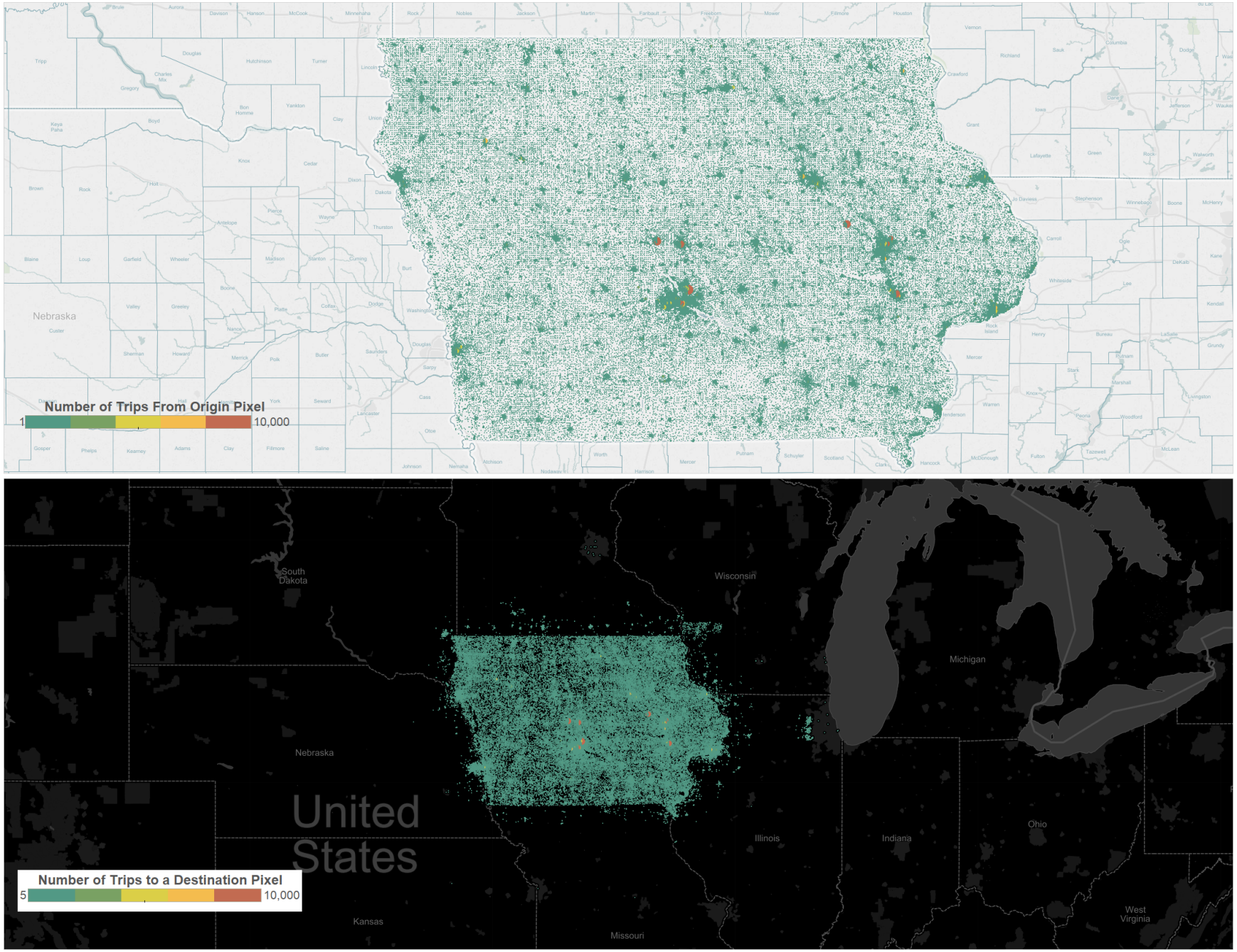

Figure A.40: Iowa: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

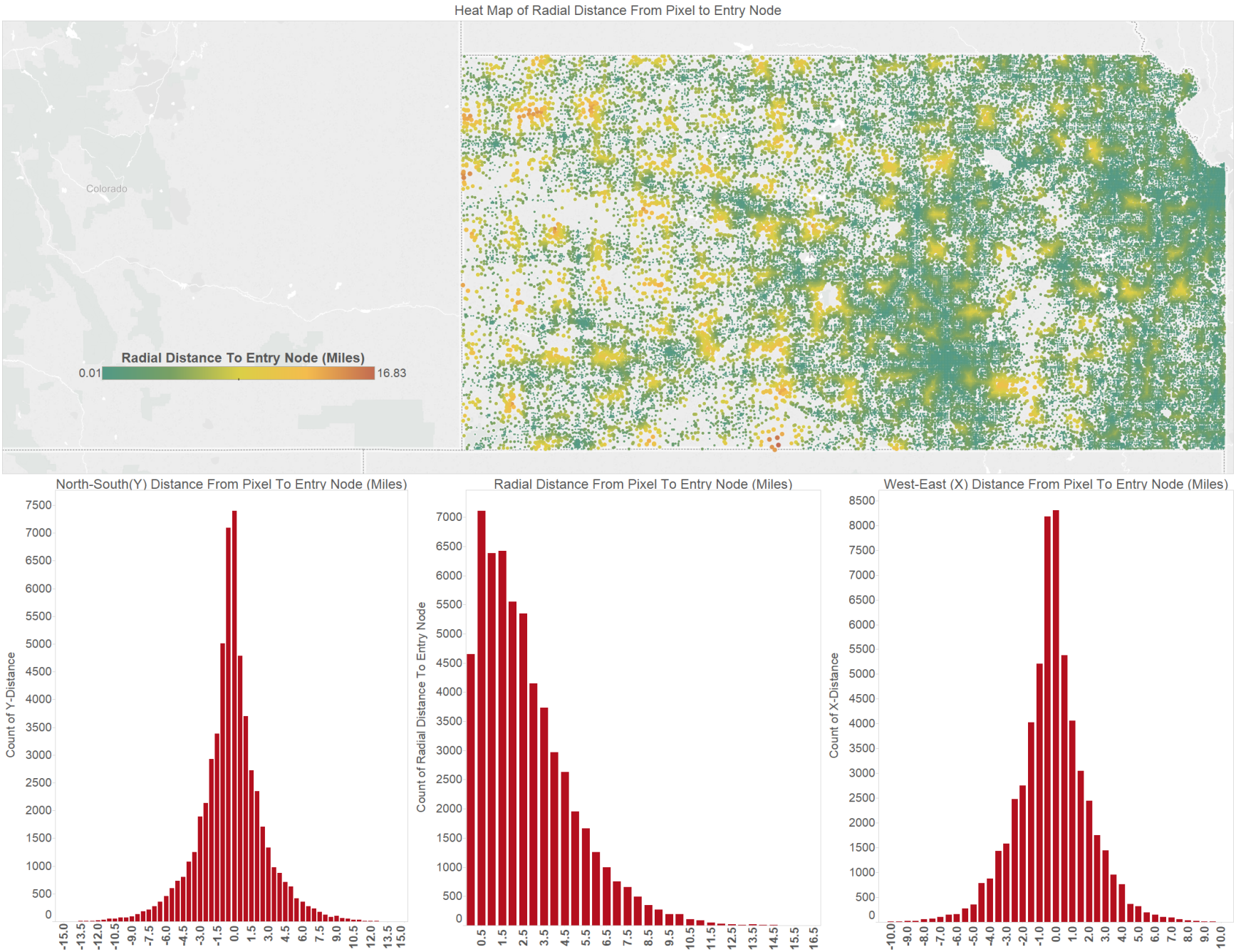

Figure A.41: Kansas: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

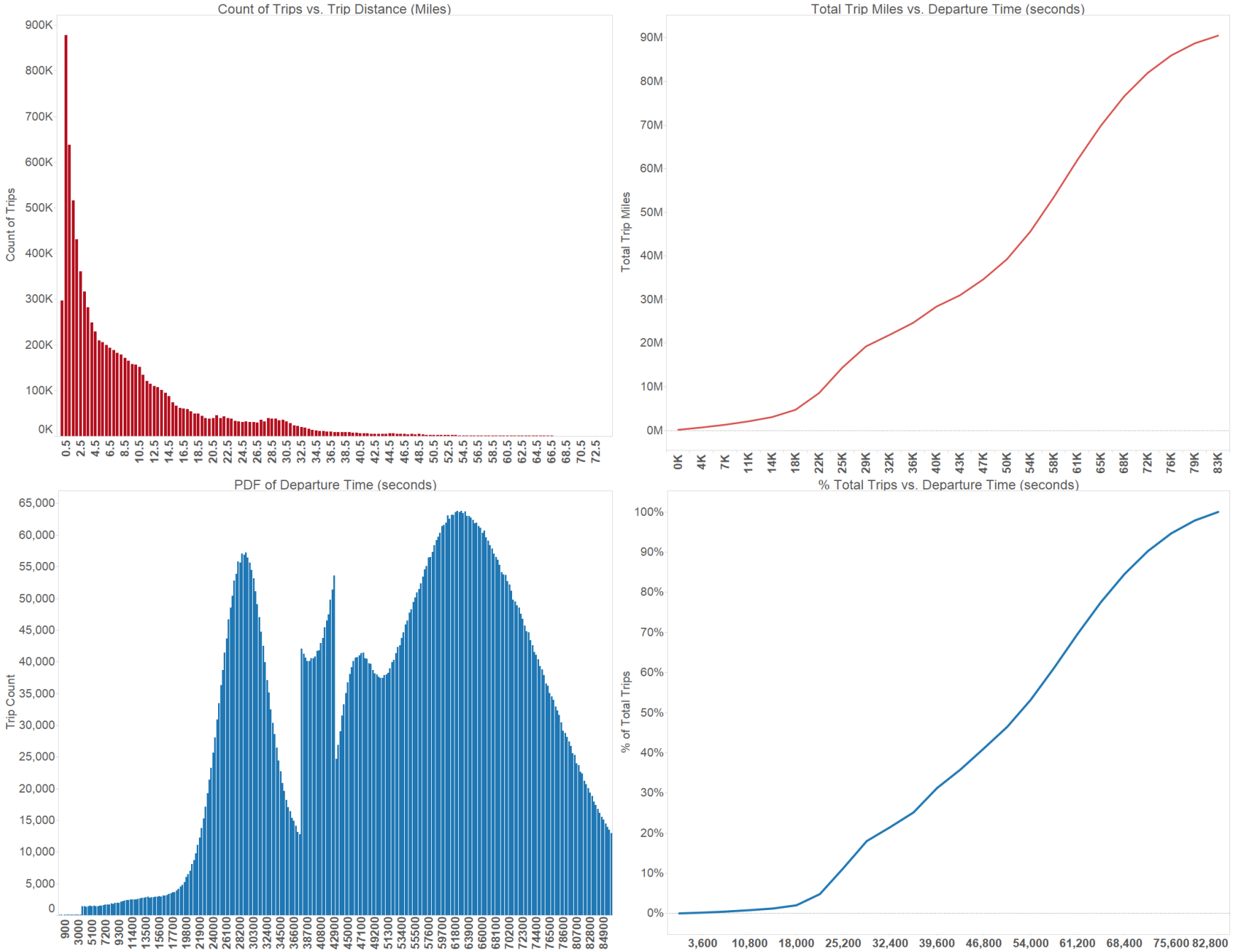

Figure A.42: Kansas: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

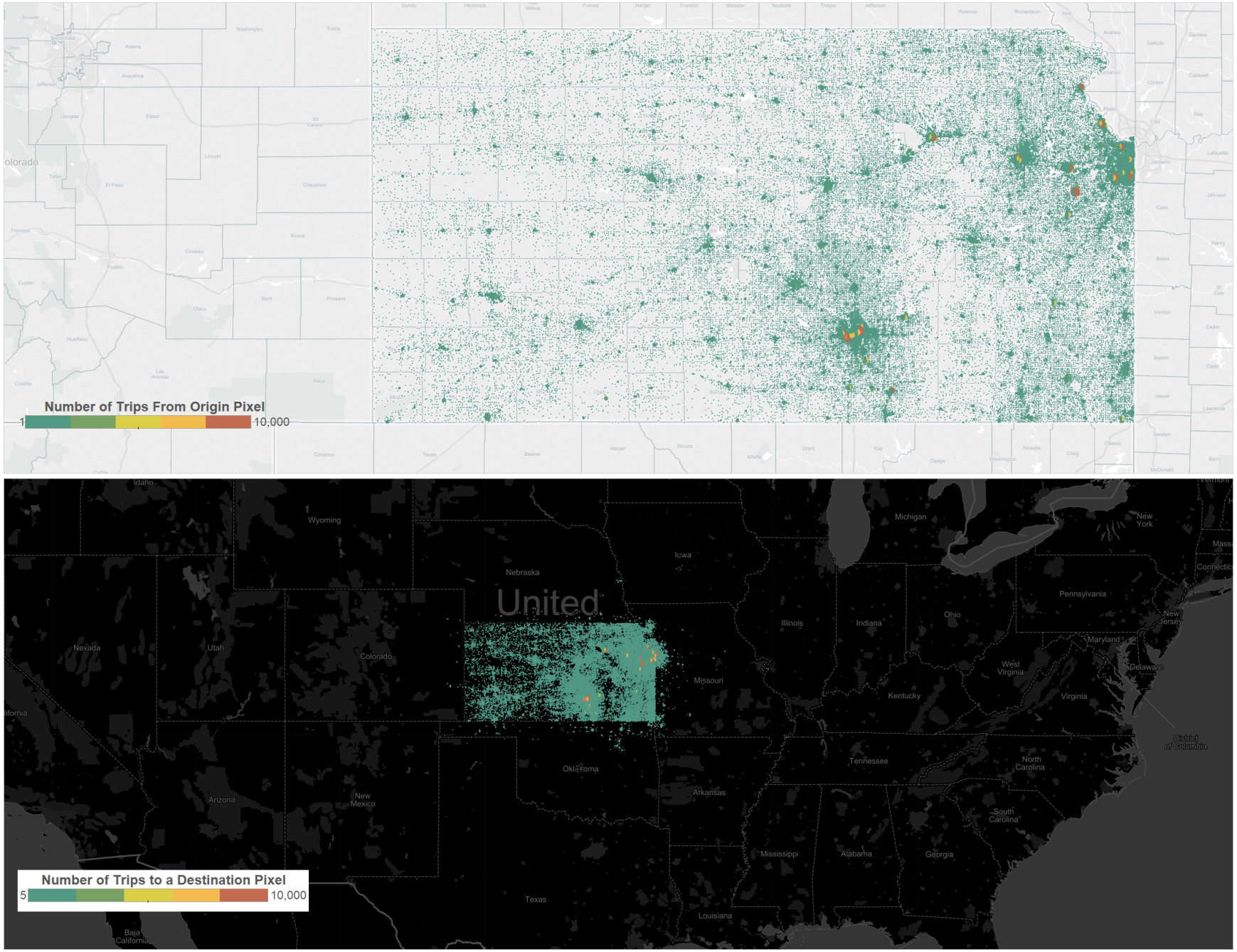

Figure A.43: Kansas: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

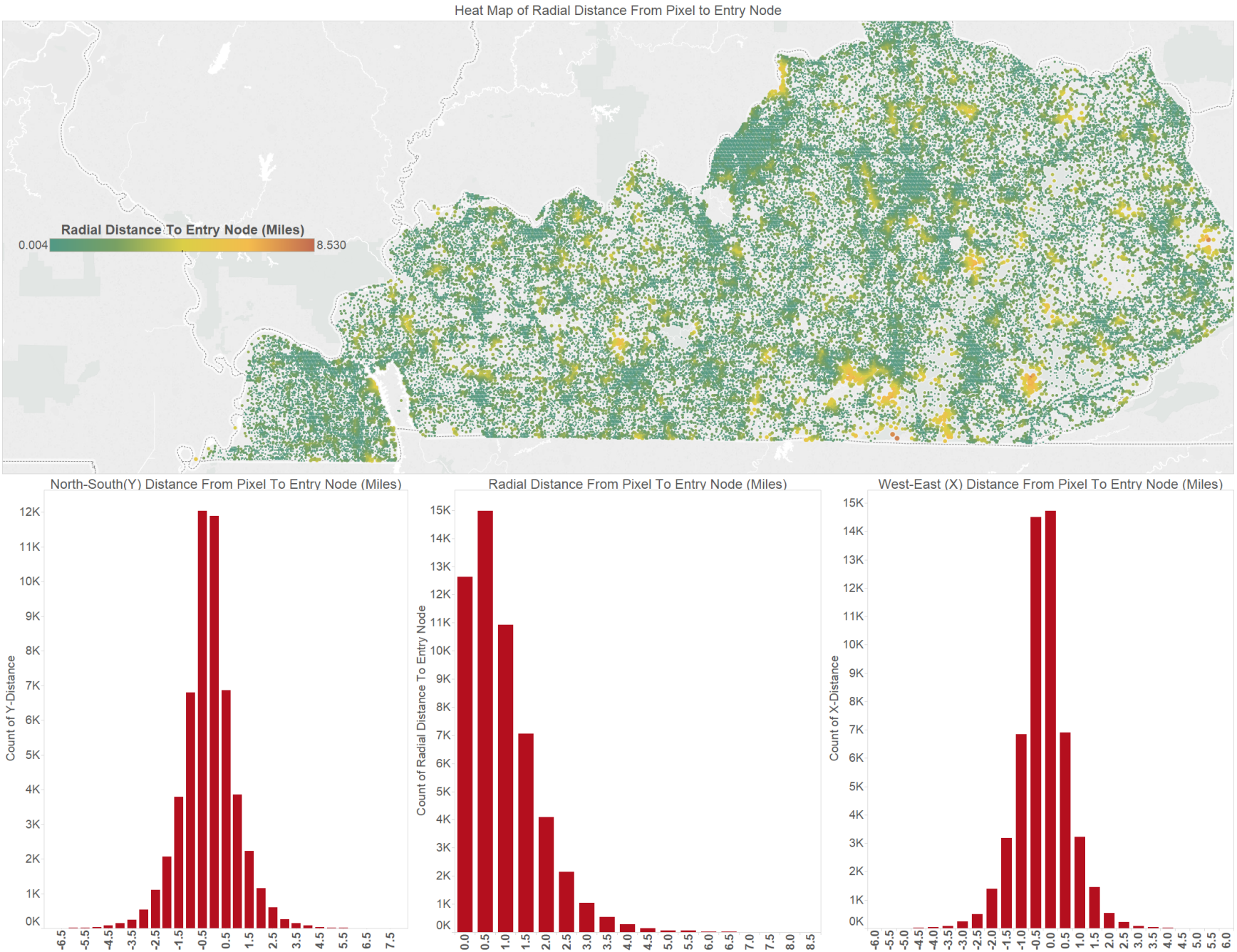

Figure A.44: Kentucky: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

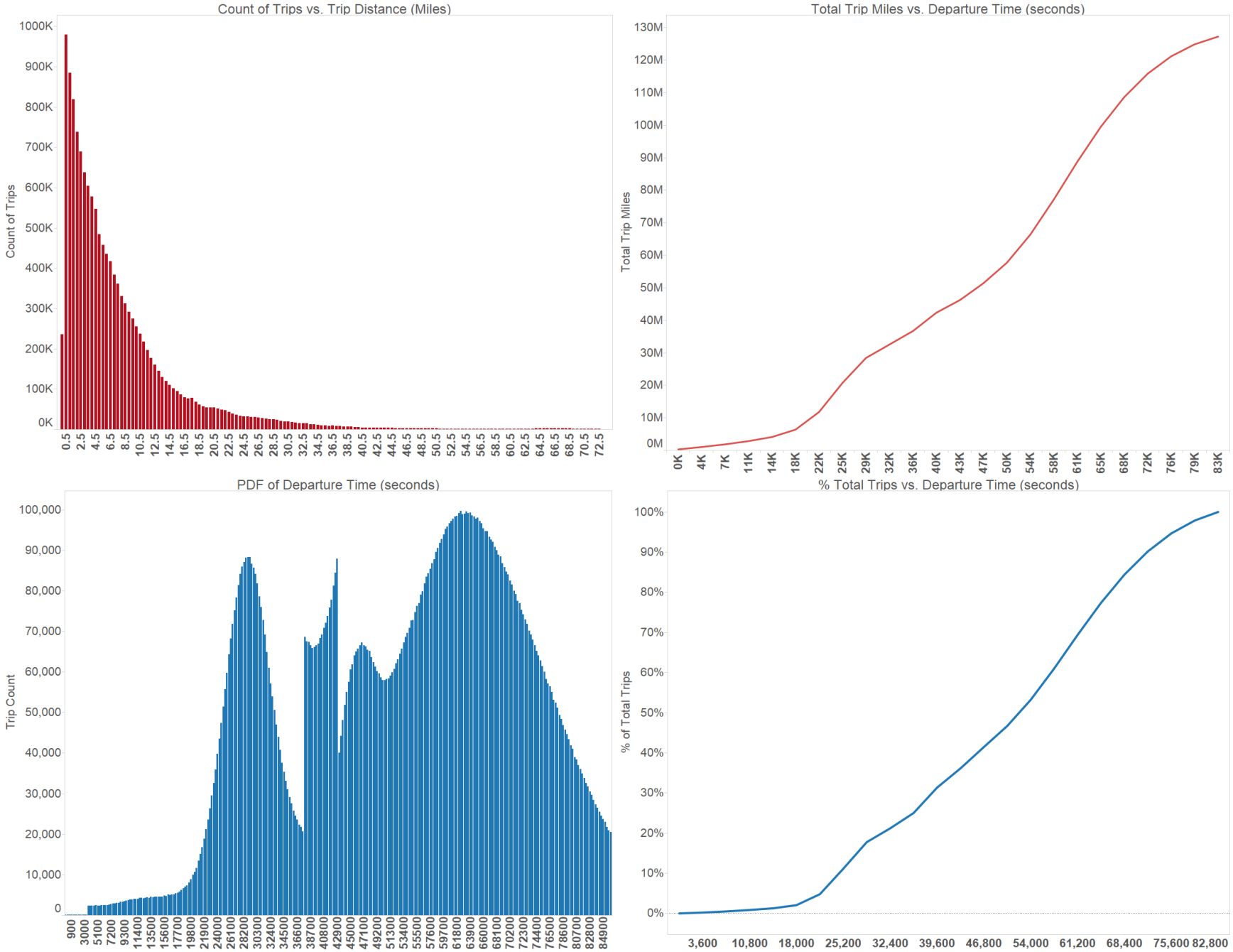

Figure A.45: Kentucky: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

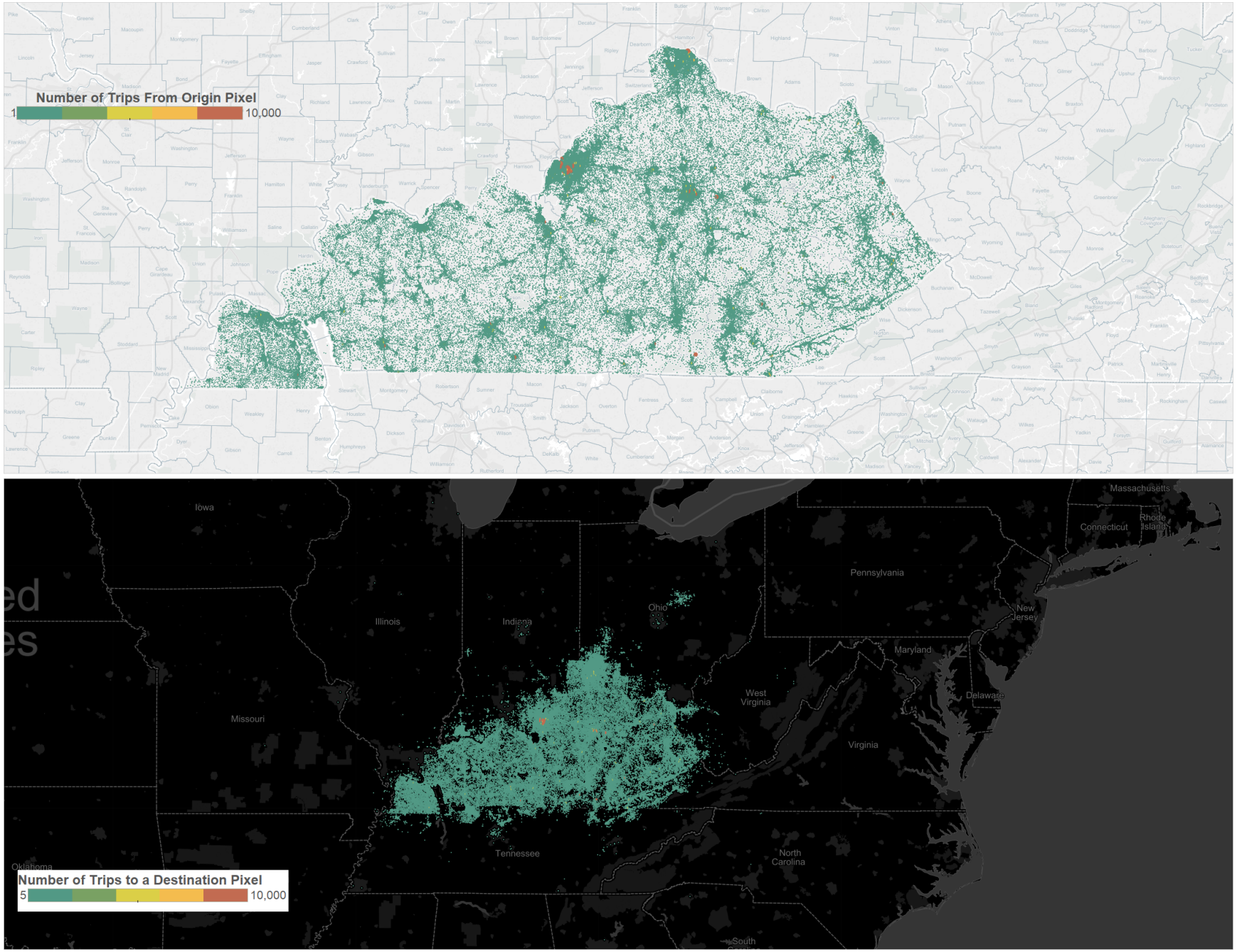

Figure A.46: Kentucky: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

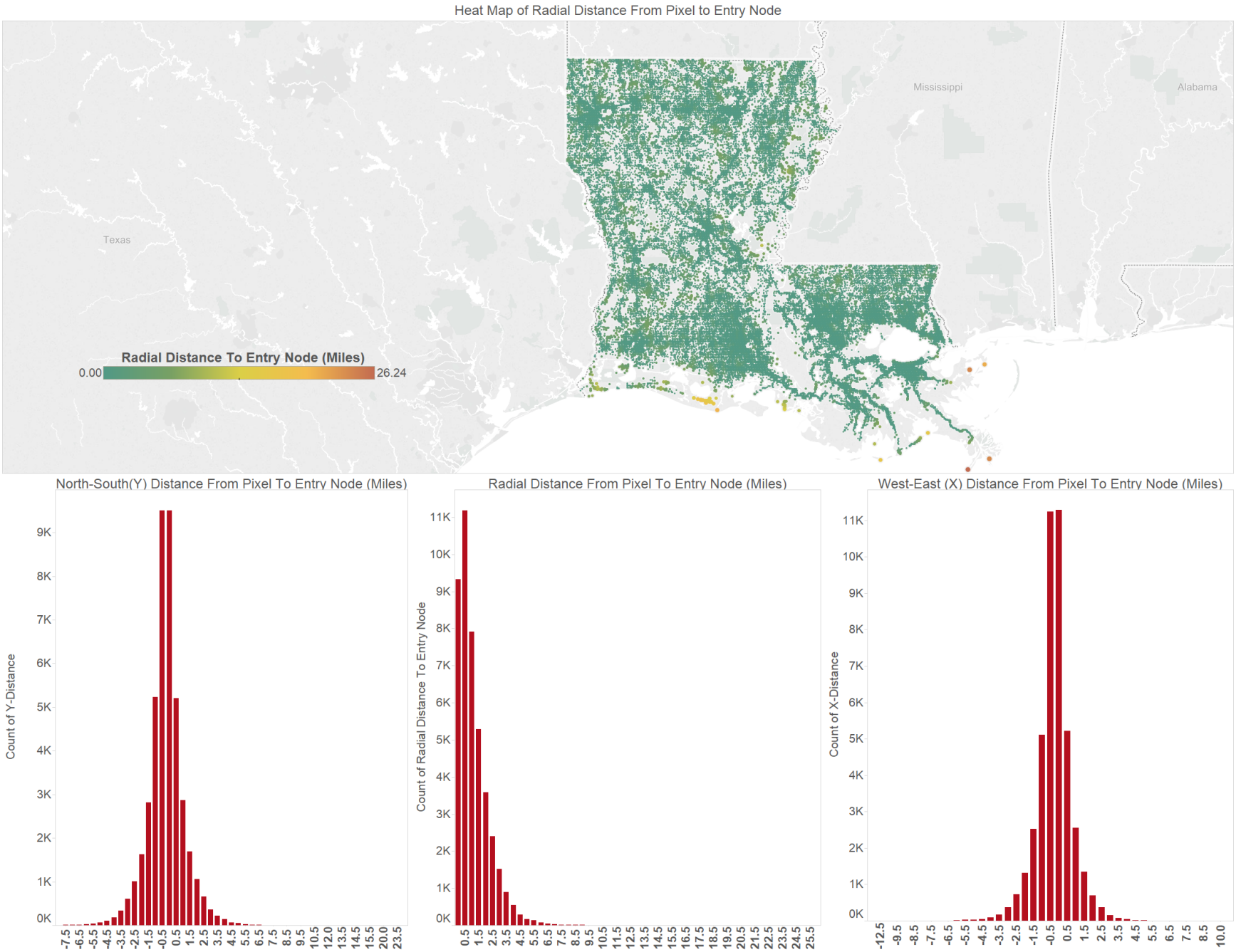

Figure A.47: Louisiana: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

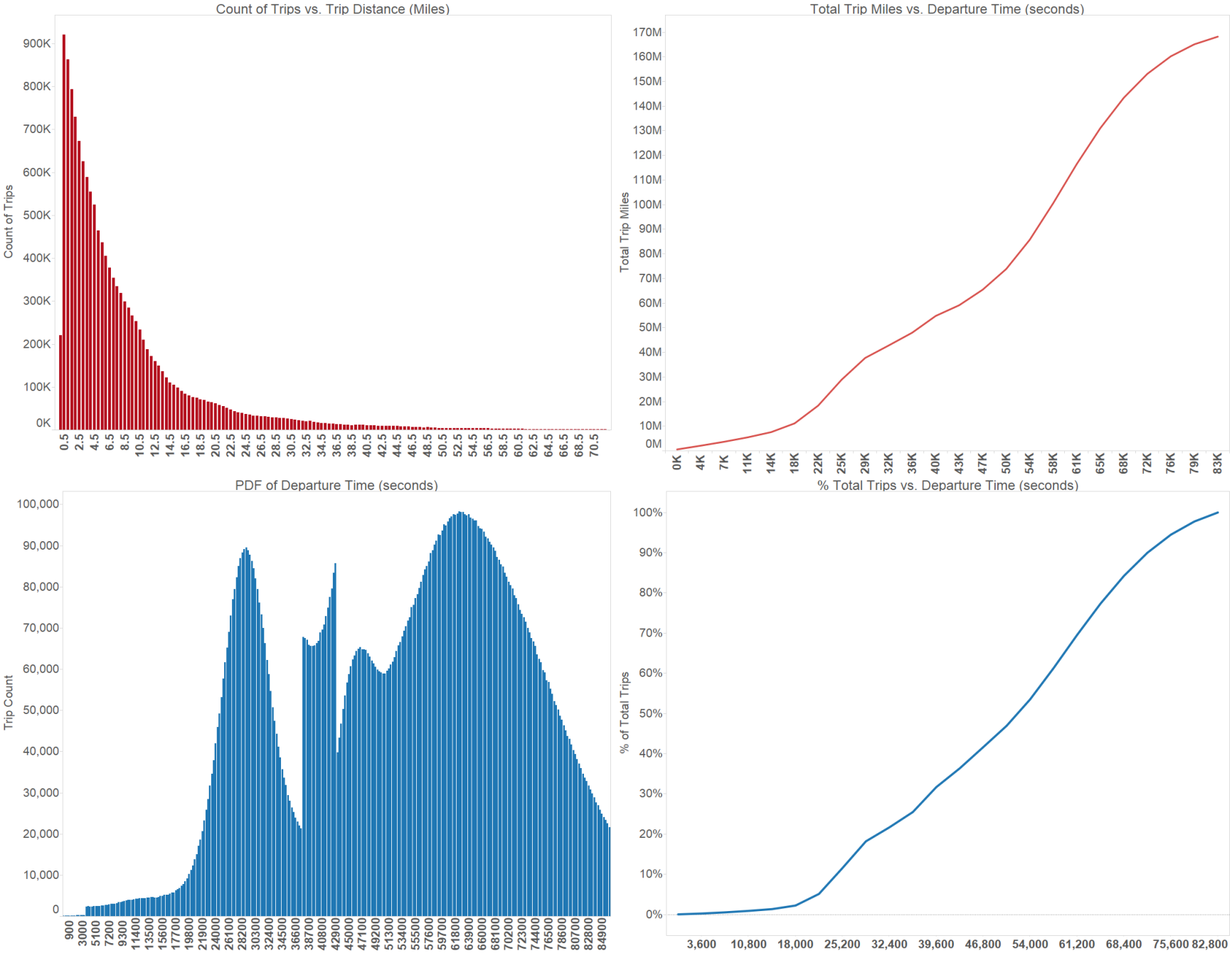

Figure A.48: Louisiana: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

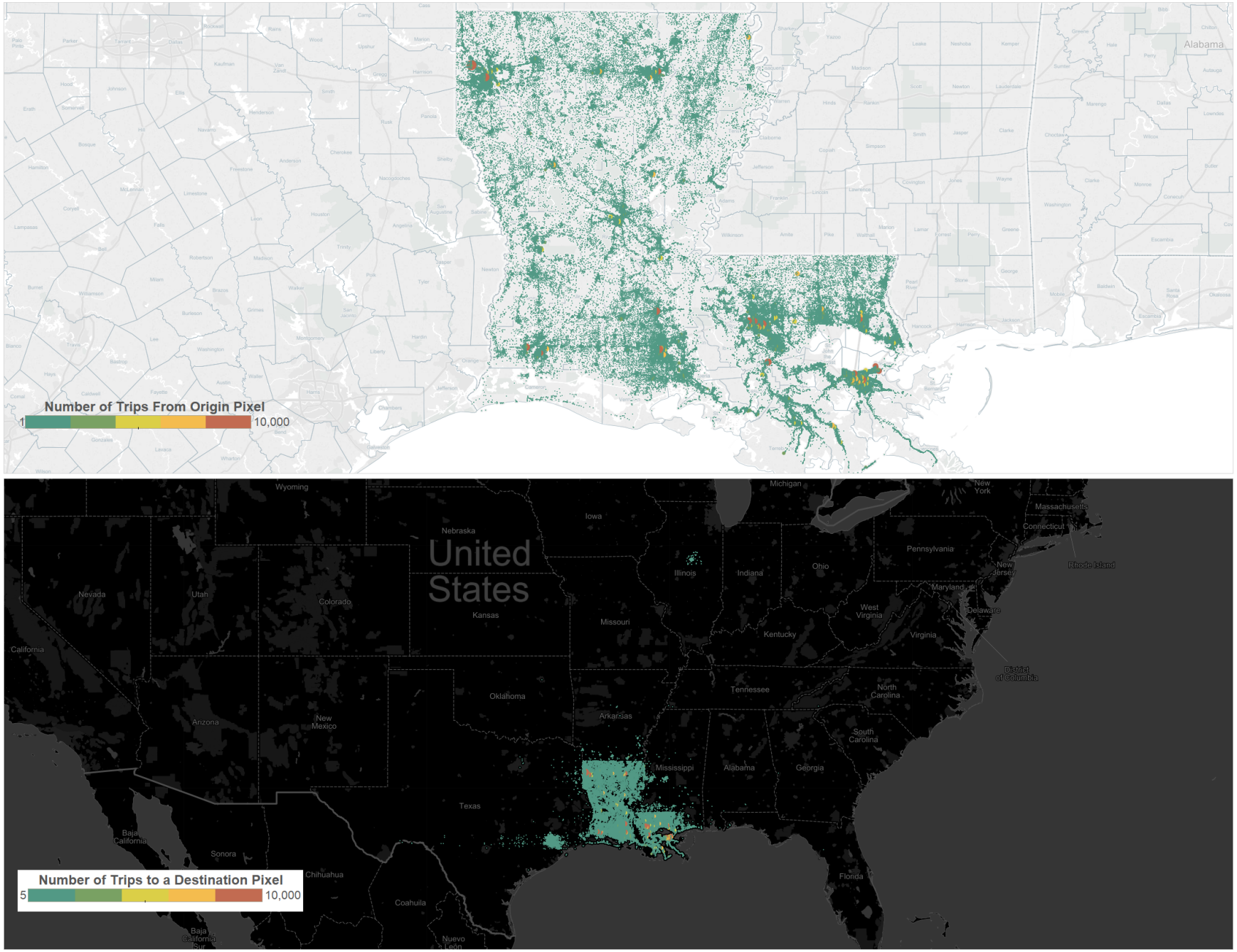

Figure A.49: Louisiana: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

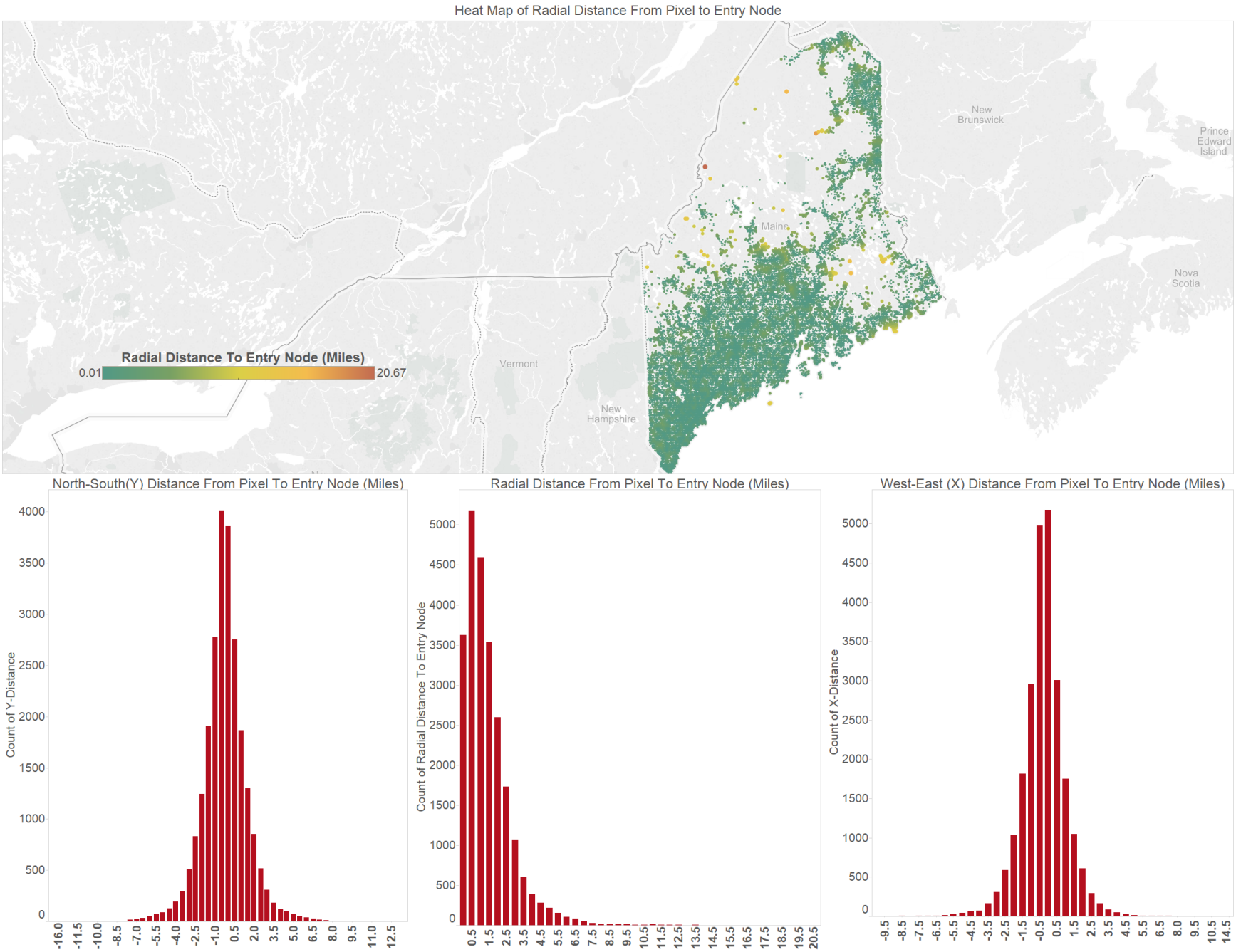

Figure A.50: Maine: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

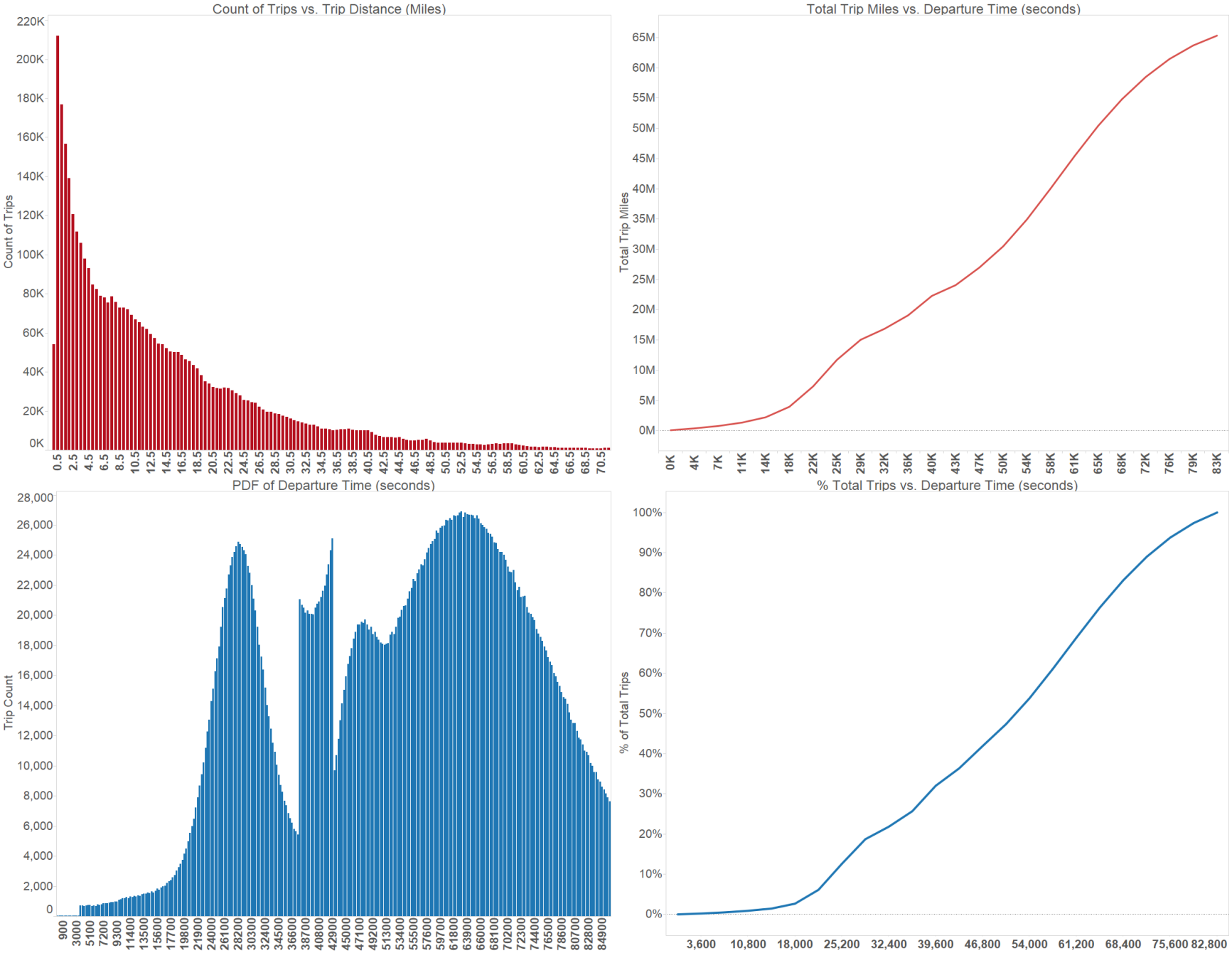

Figure A.51: Maine: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

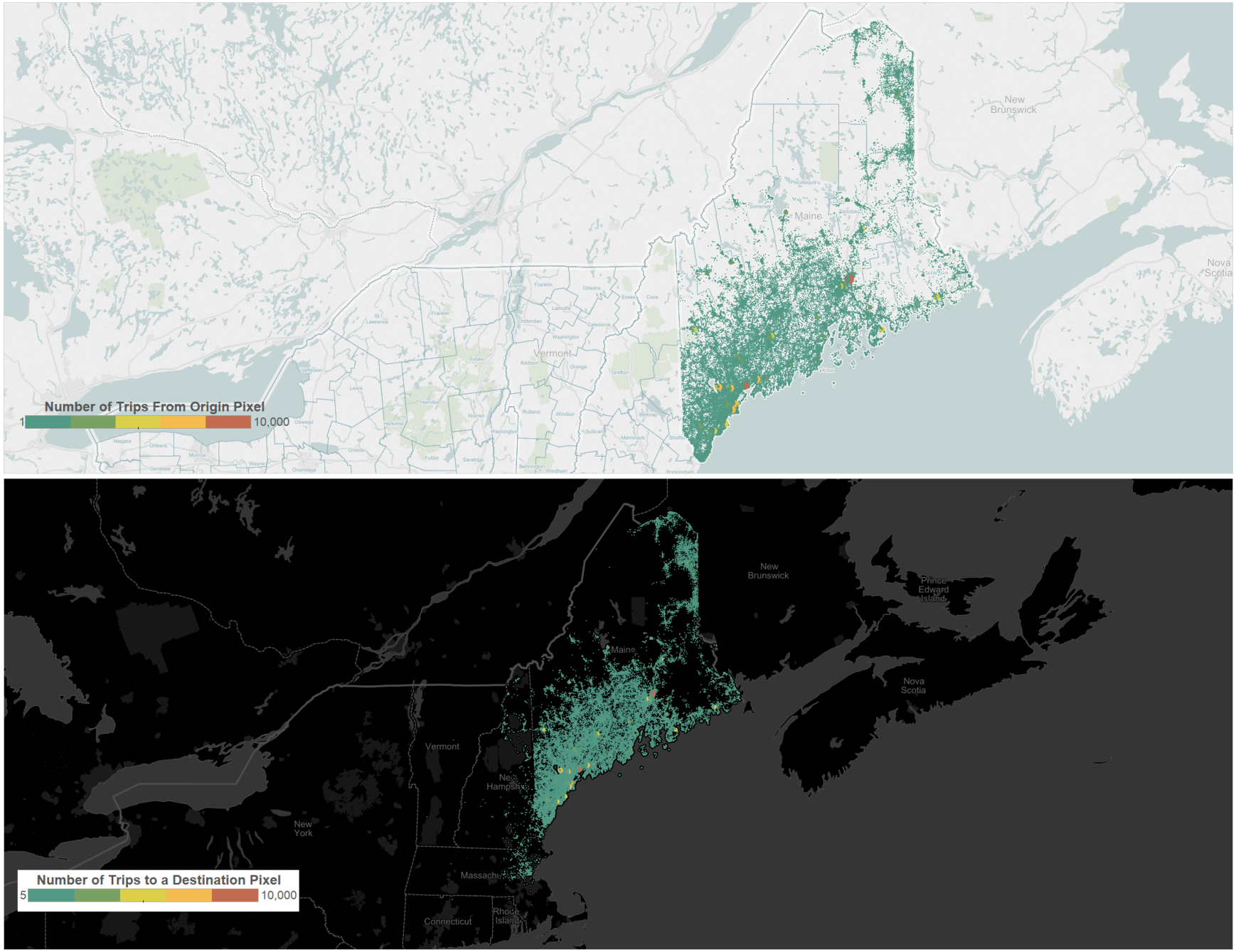

Figure A.52: Maine: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

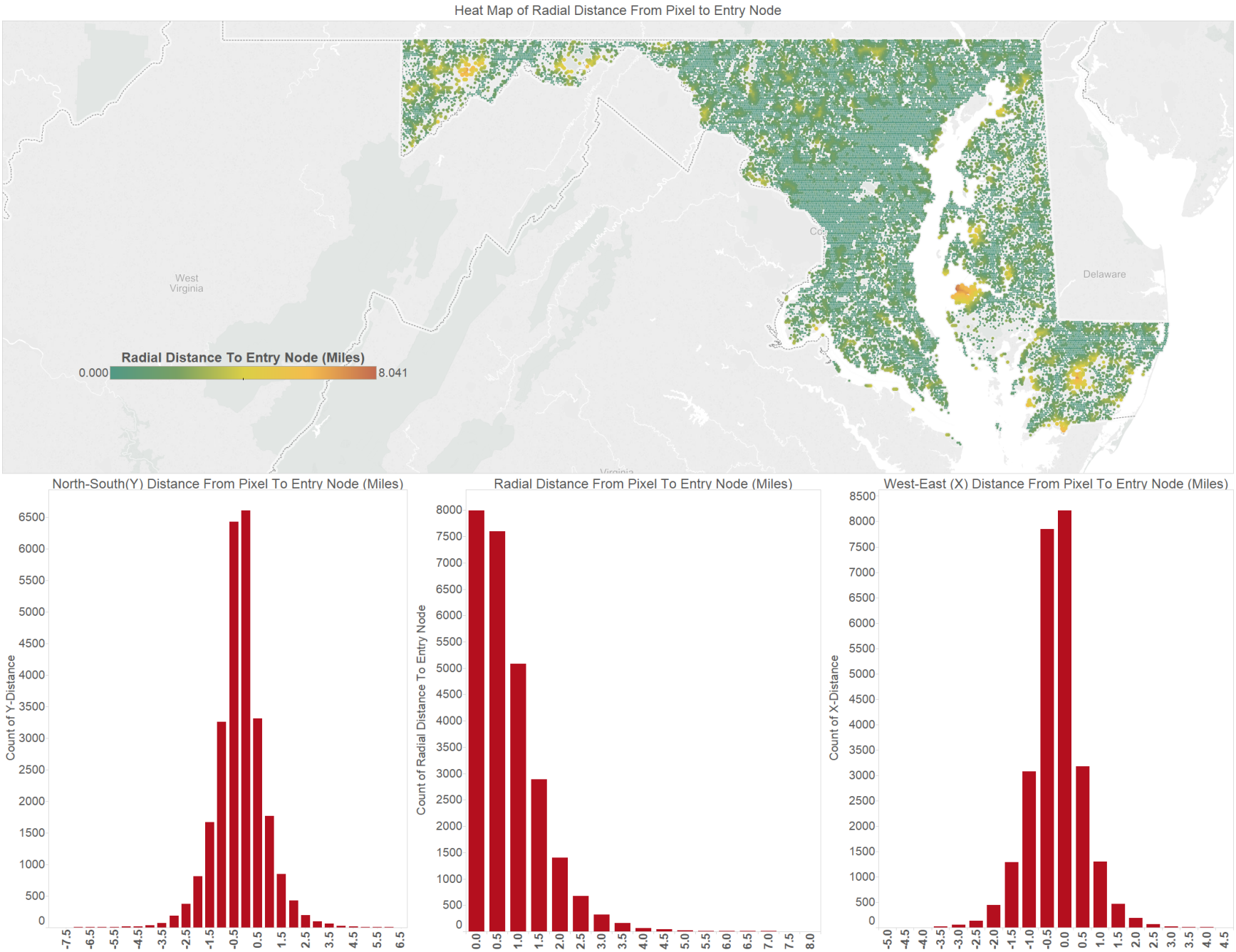

Figure A.53: Maryland: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

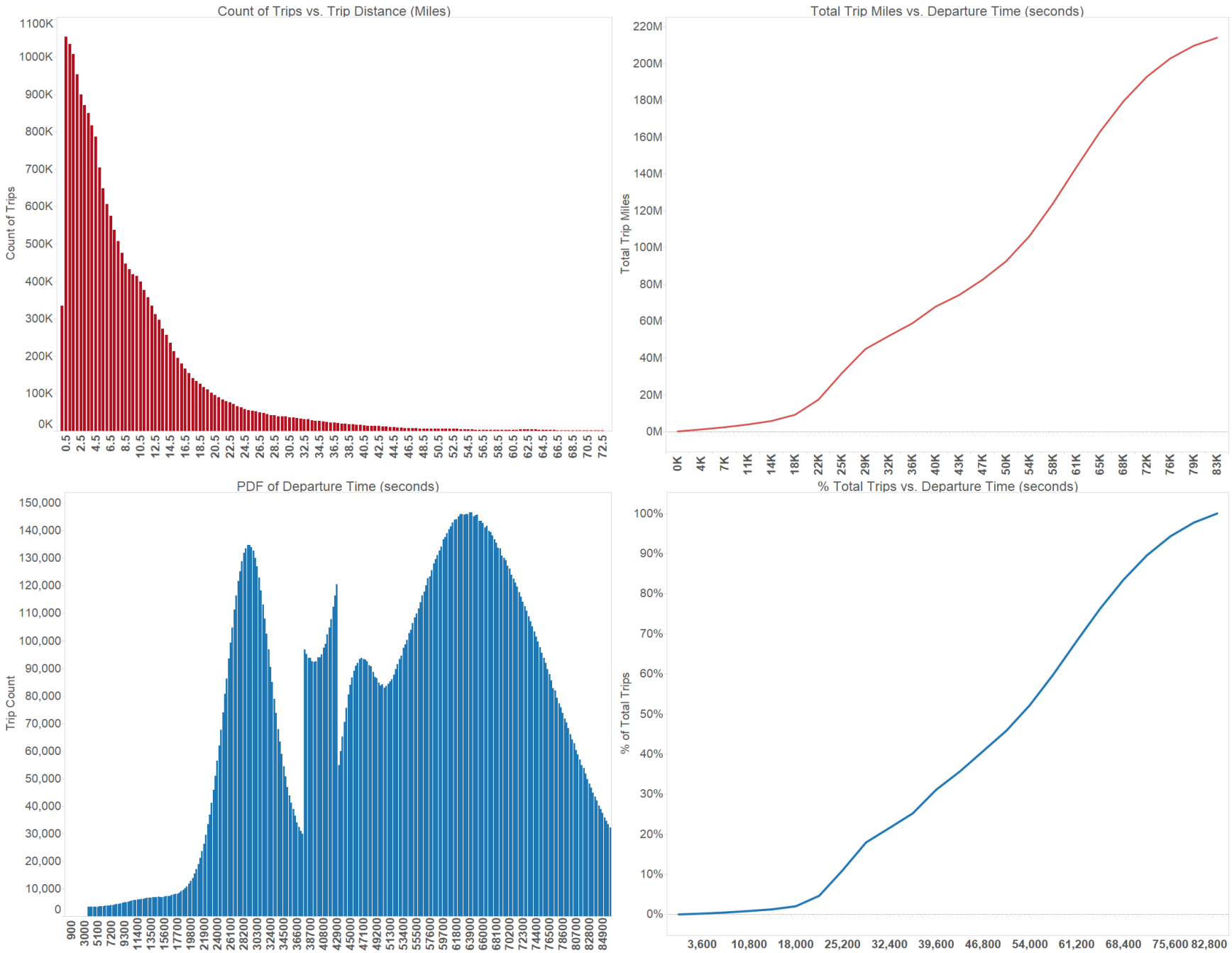

Figure A.54: Maryland: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

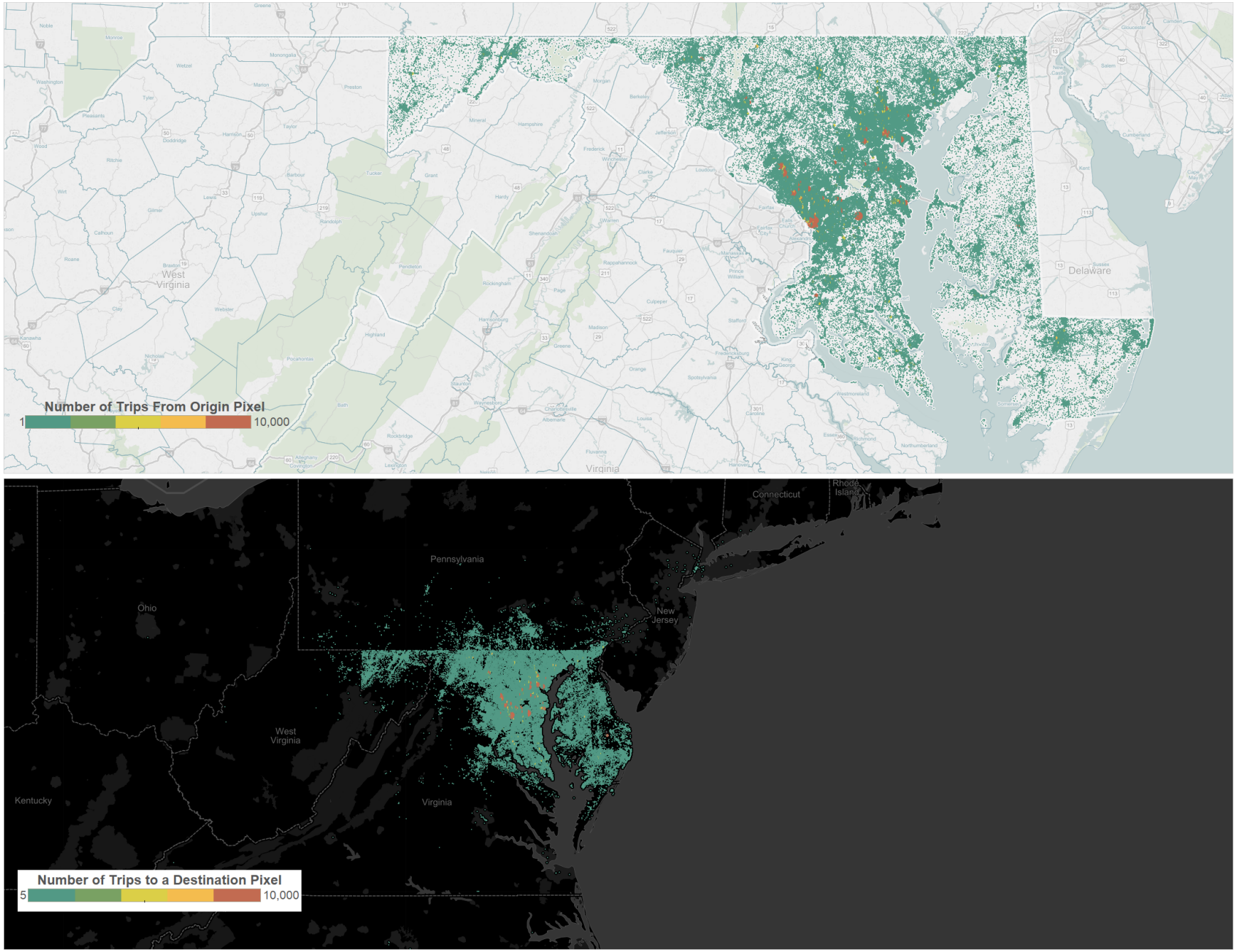

Figure A.55: Maryland: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

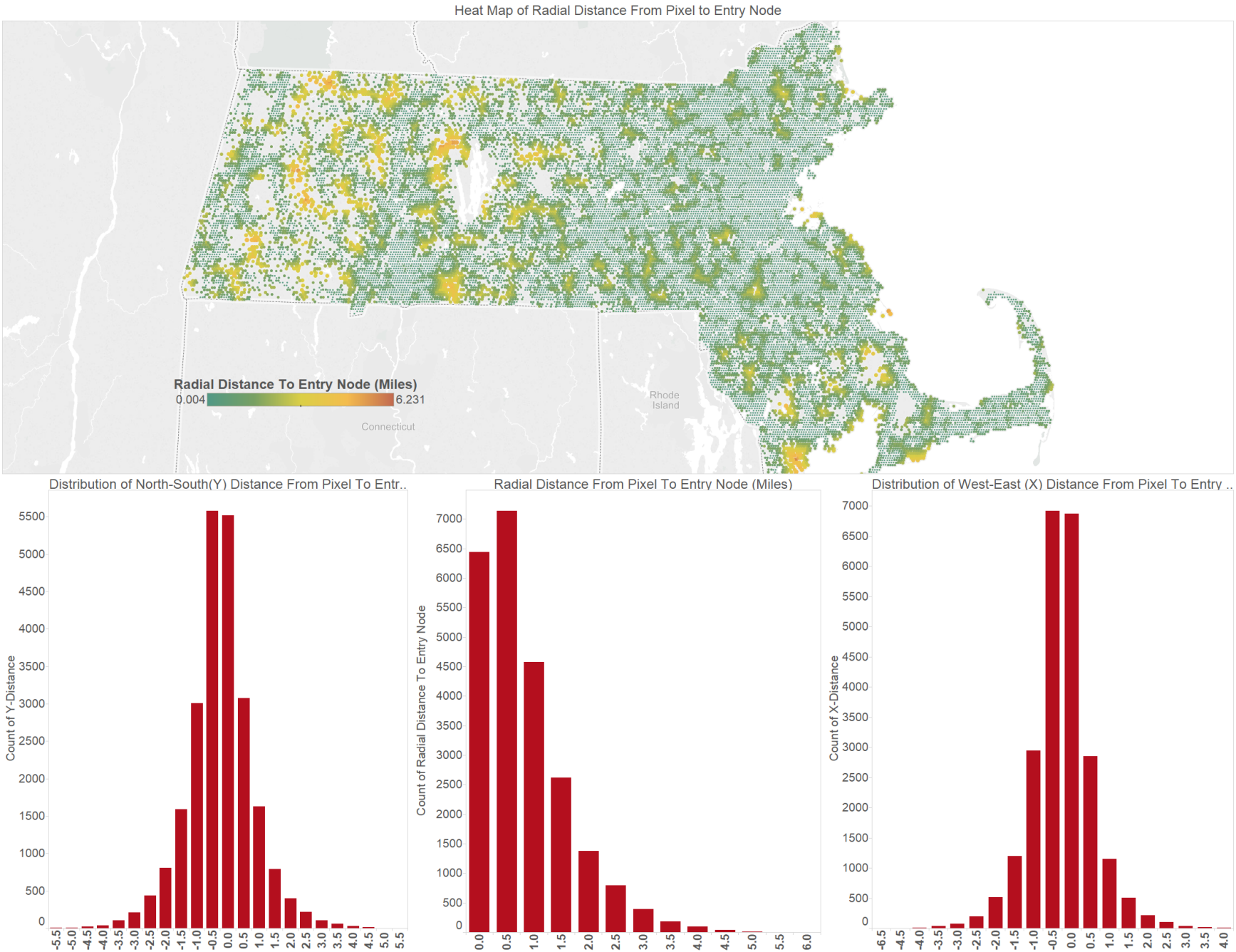

Figure A.56: Massachusetts: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

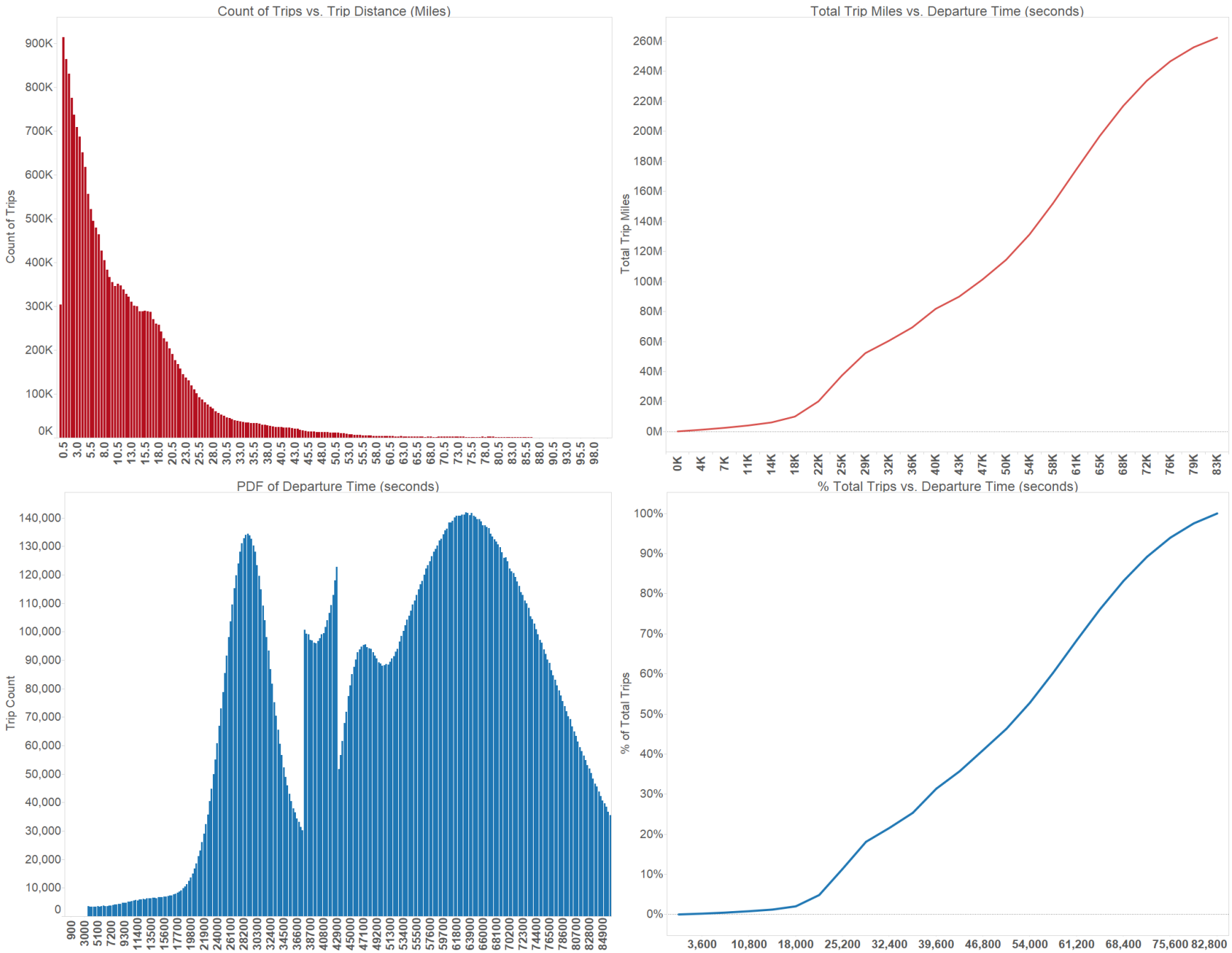

Figure A.57: Massachusetts: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

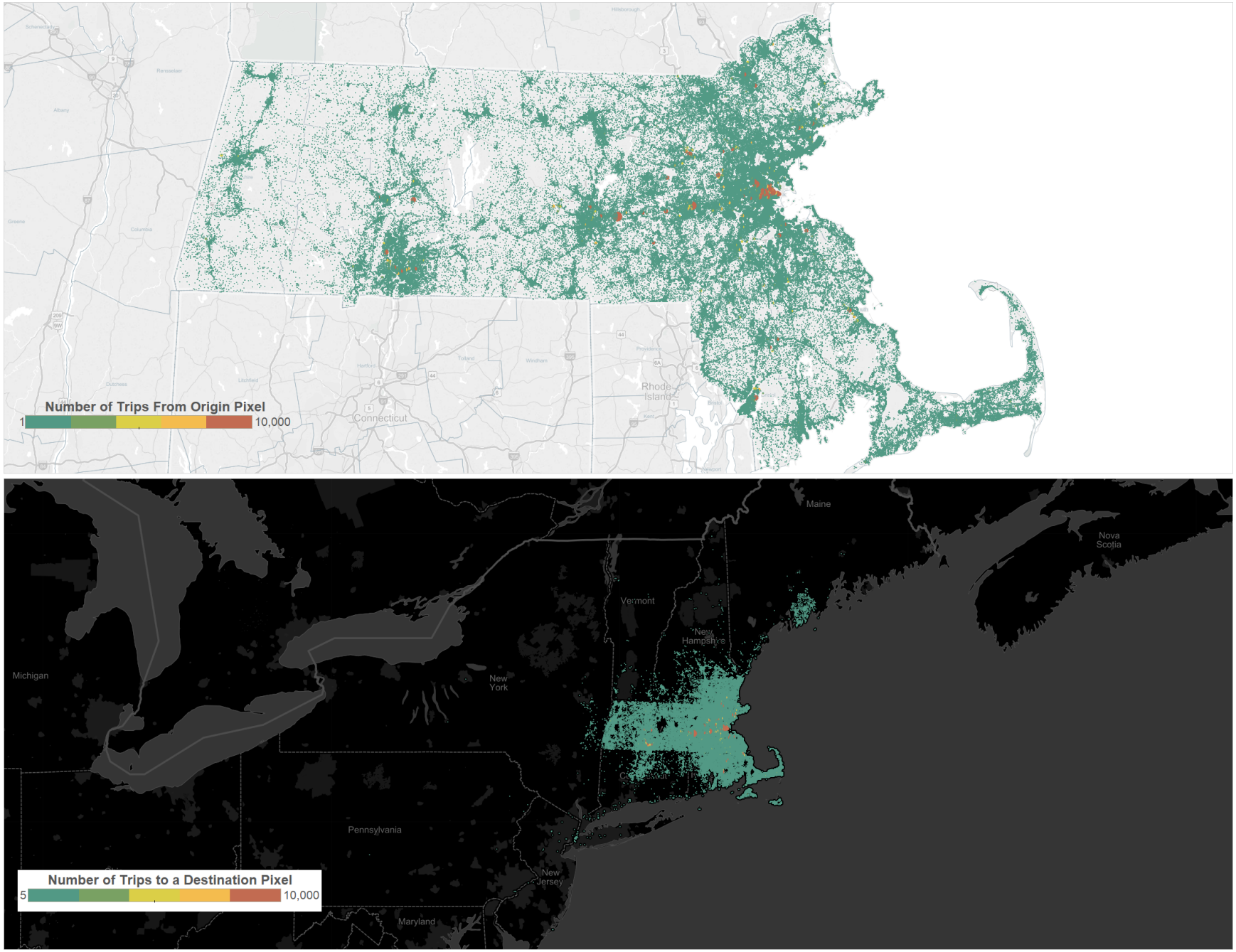

Figure A.58: Massachusetts: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

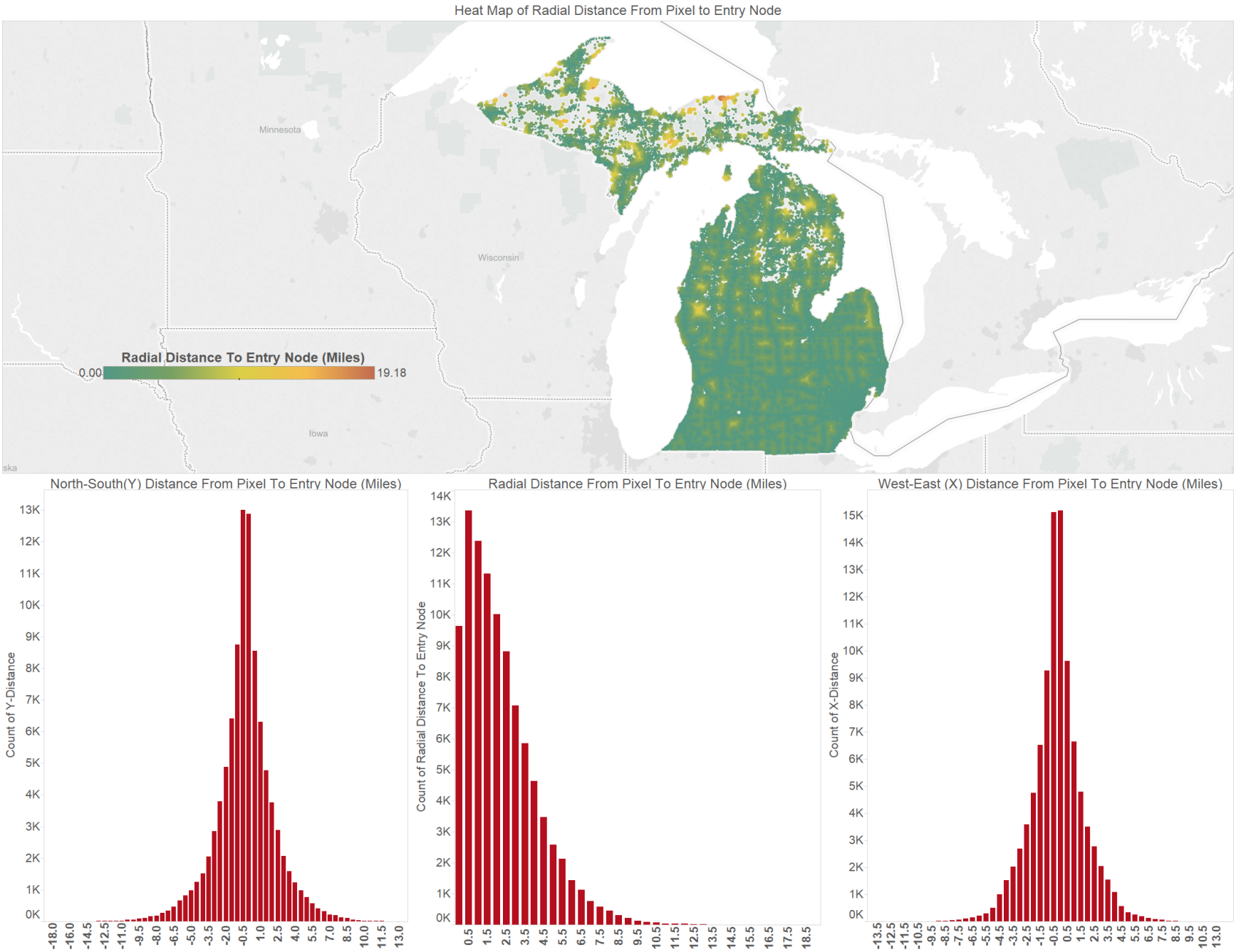

Figure A.59: Michigan: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

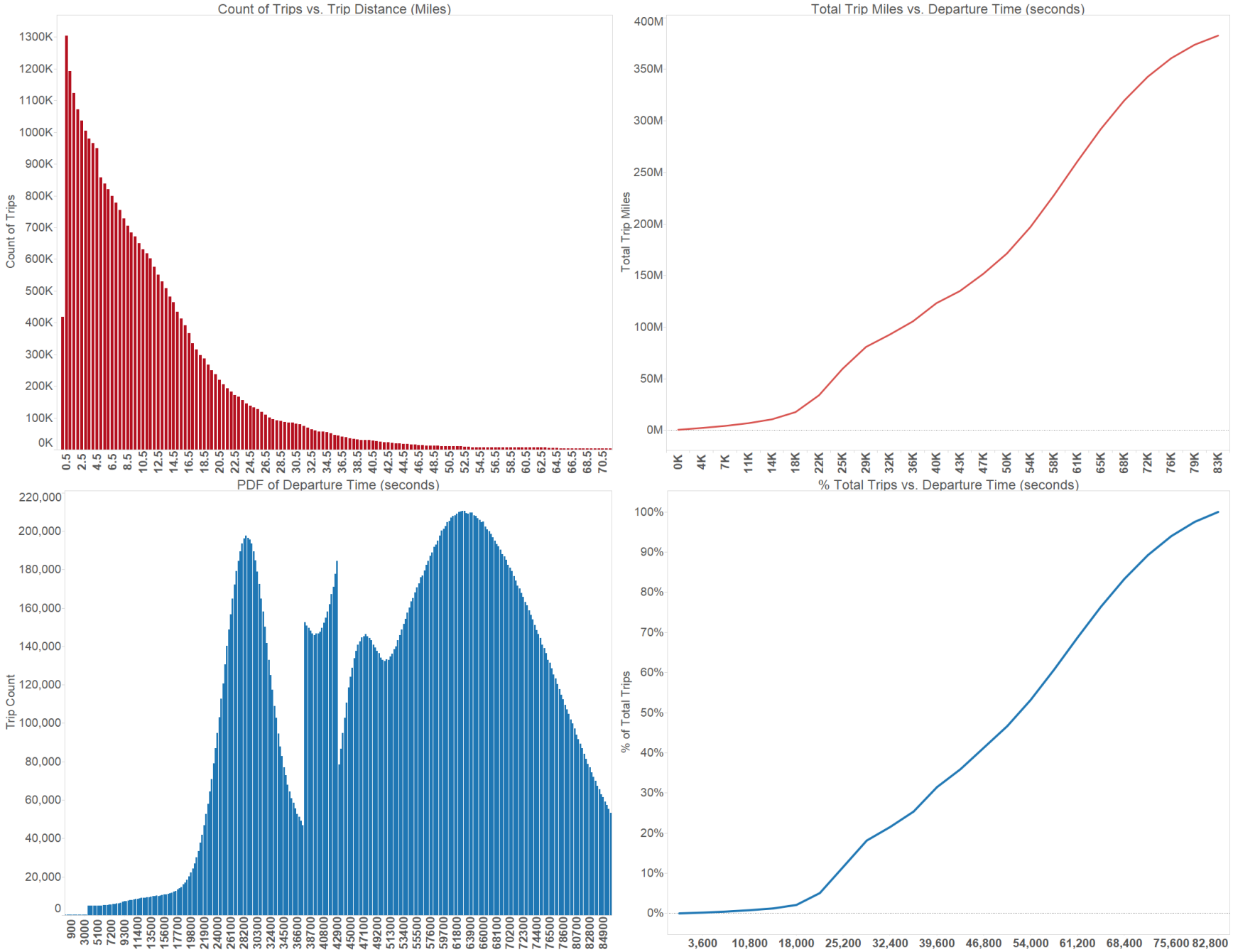

Figure A.60: Michigan: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

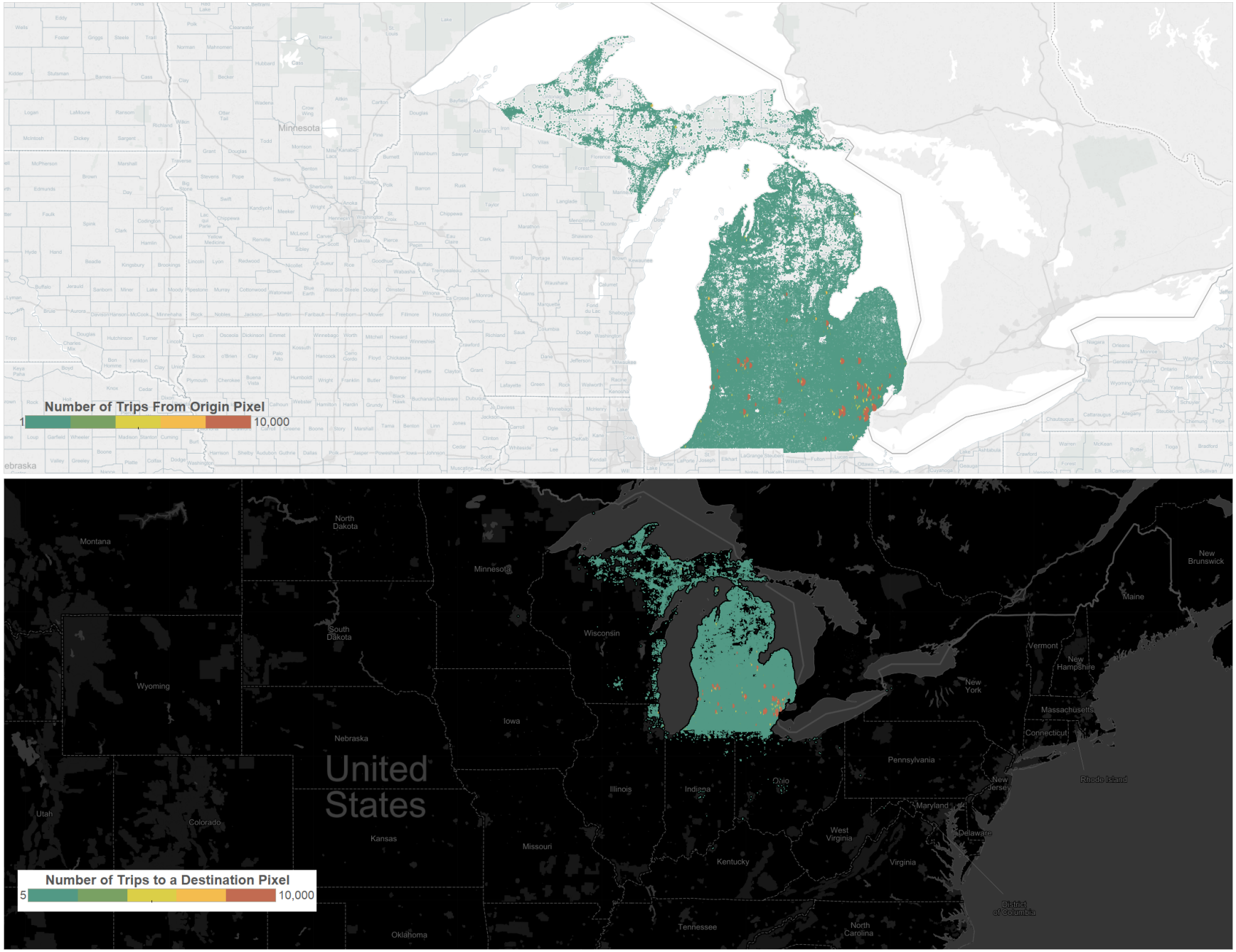

Figure A.61: Michigan: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

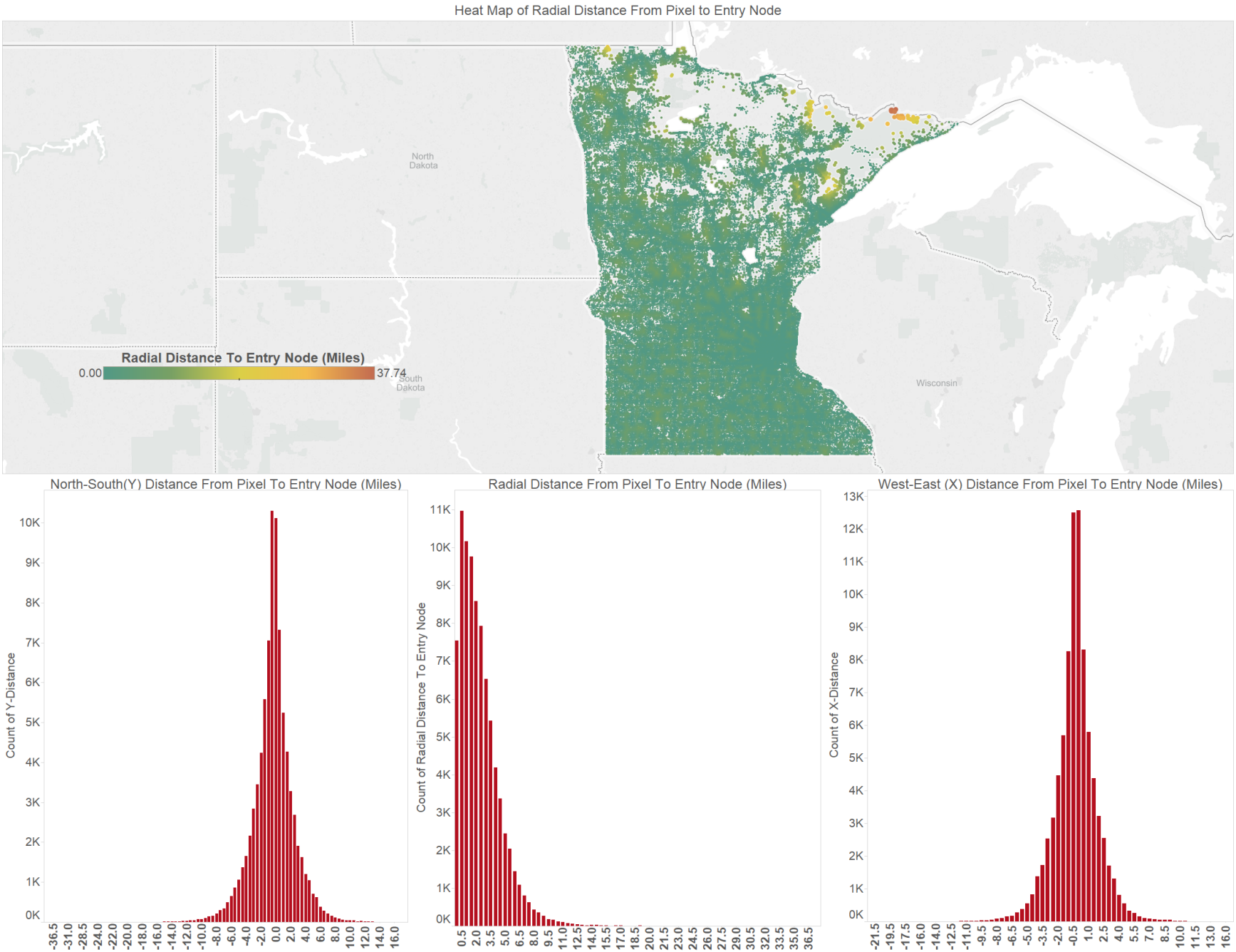

Figure A.62: Minnesota: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

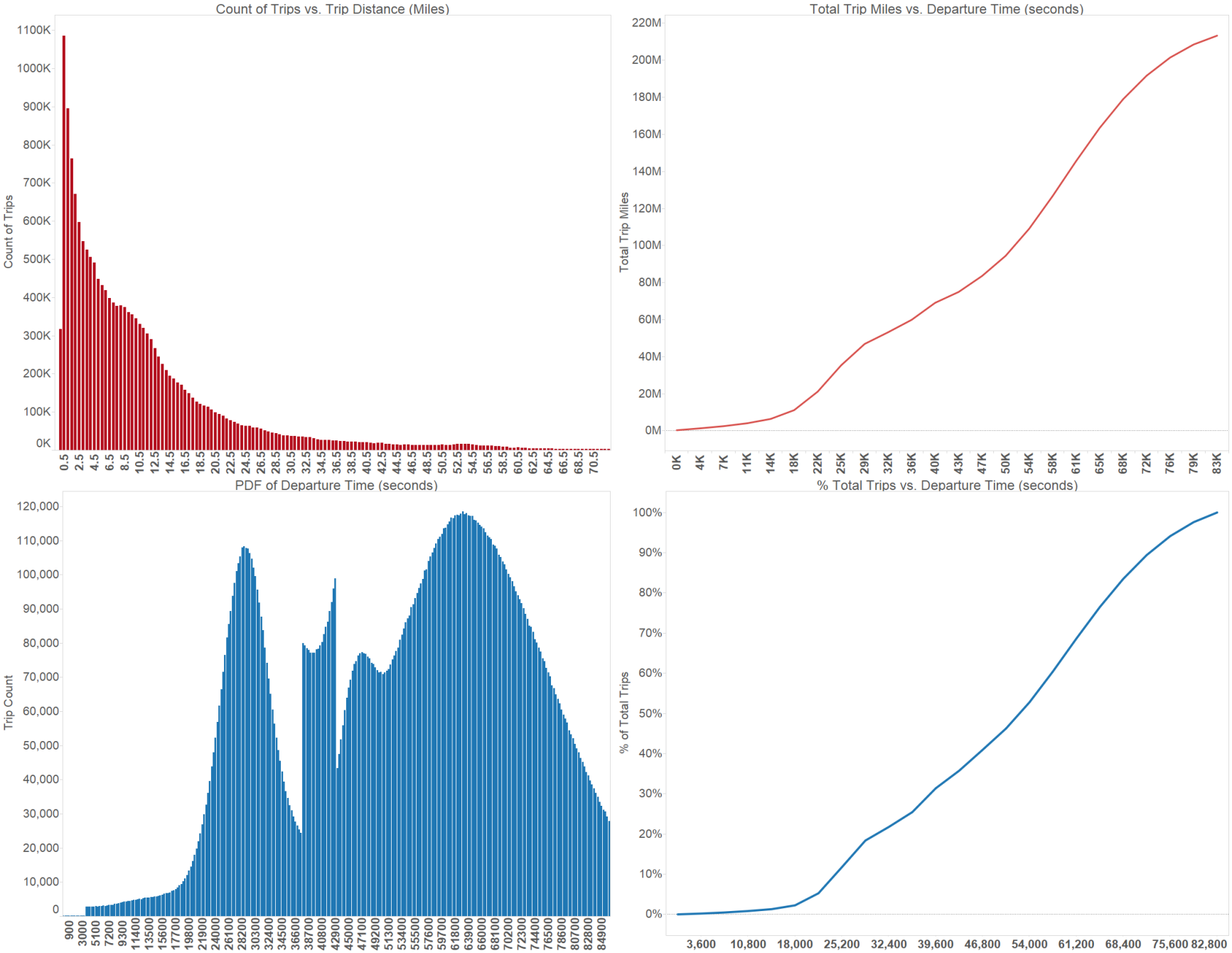

Figure A.63: Minnesota: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

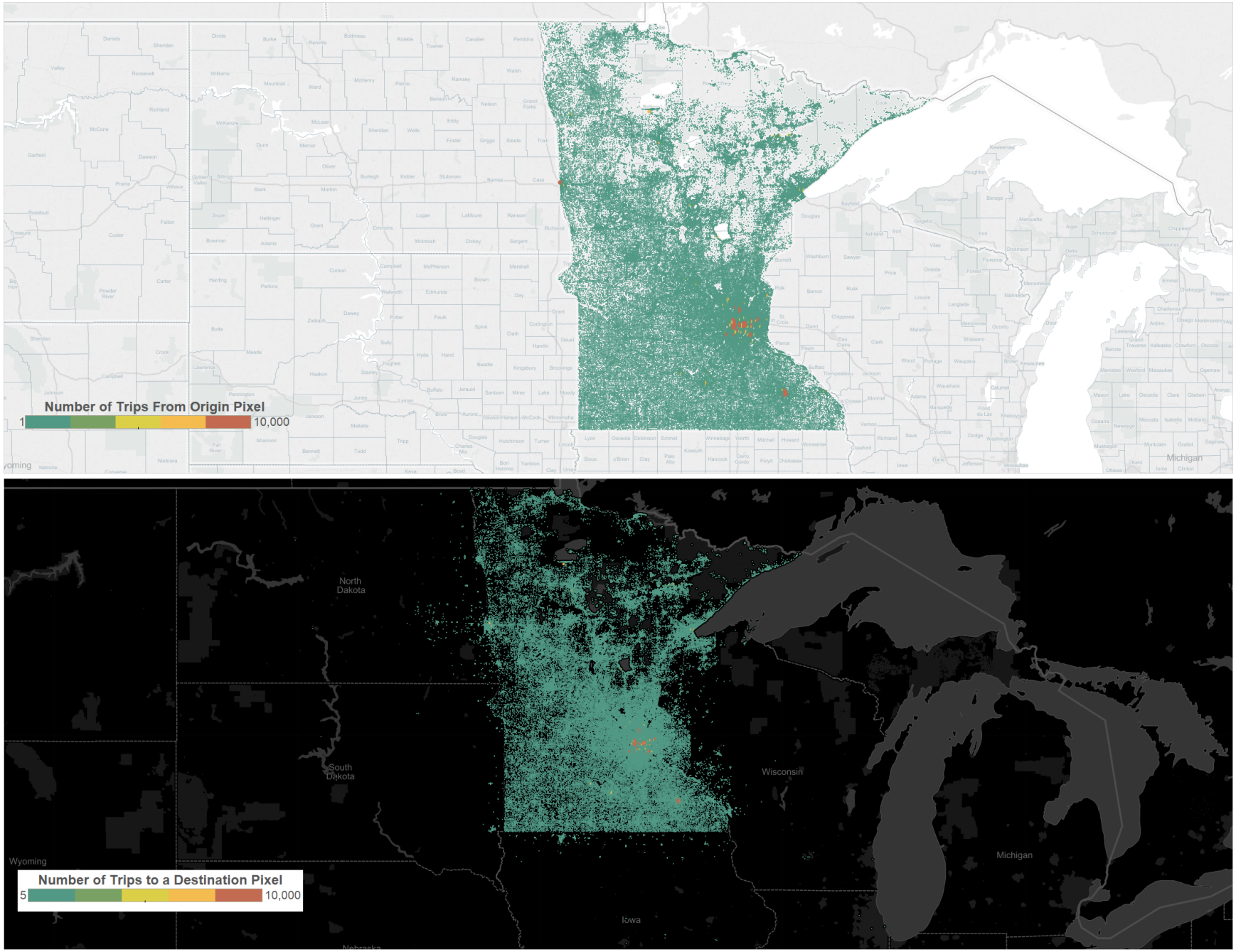

Figure A.64: Minnesota: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

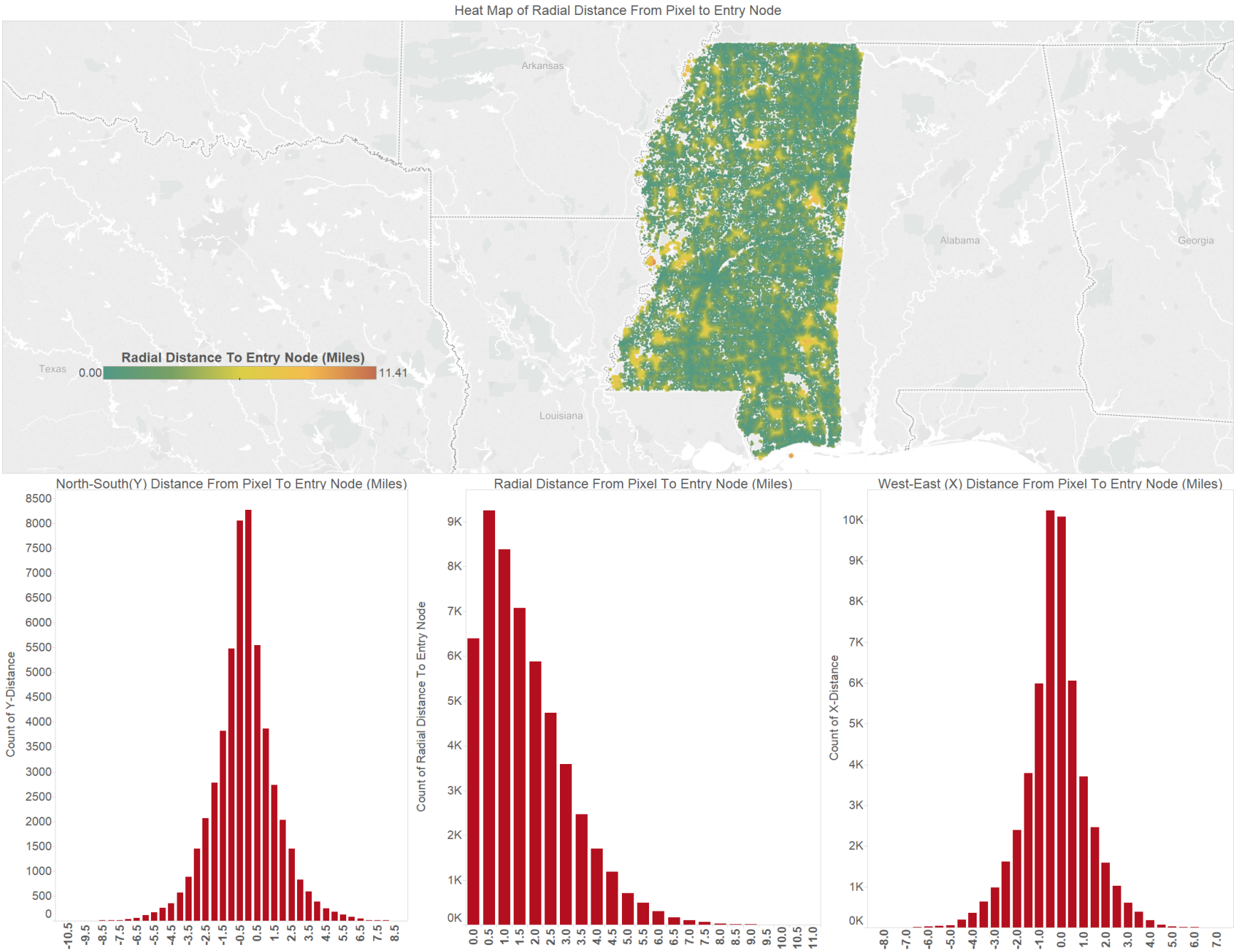

Figure A.65: Mississippi: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

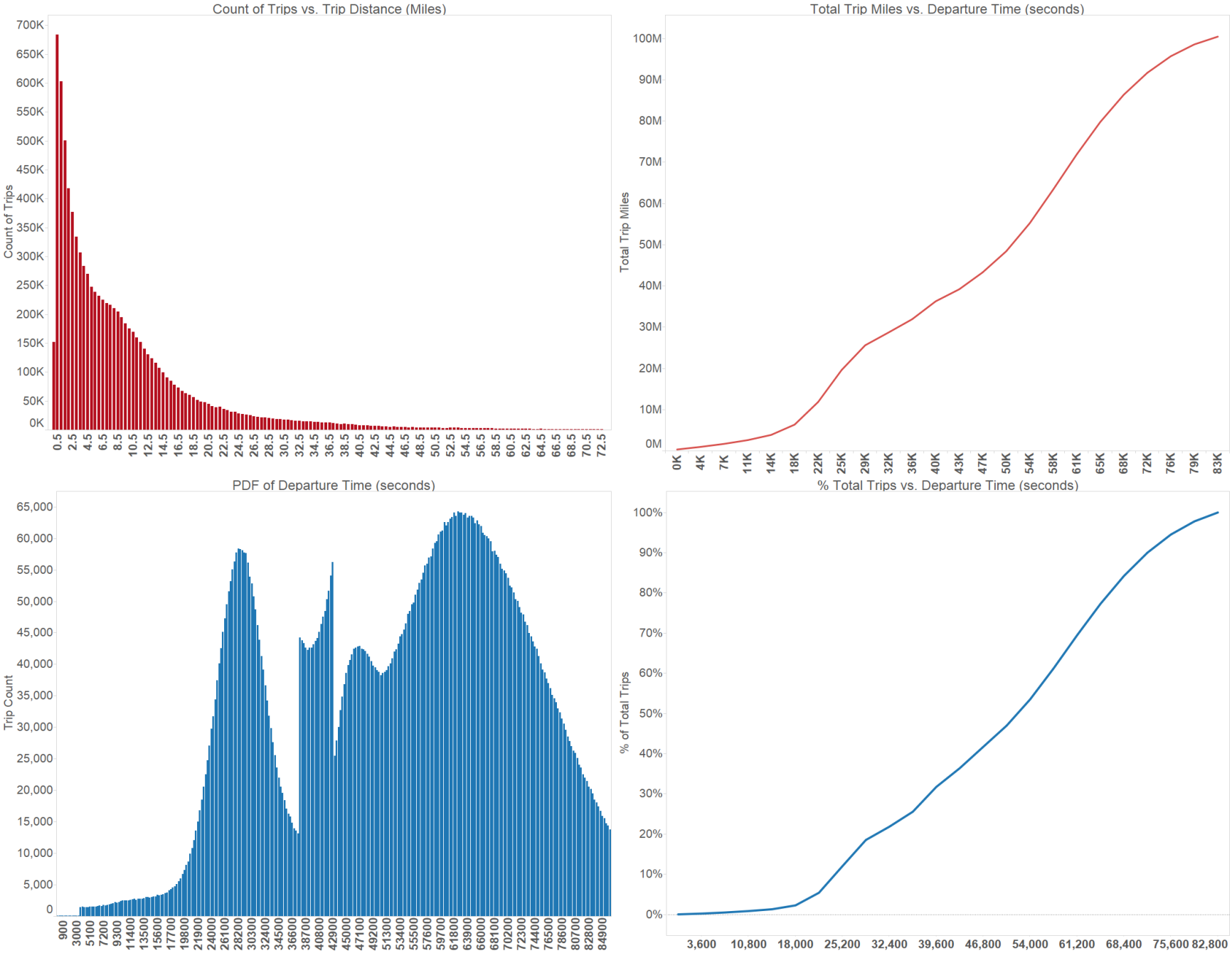

Figure A.66: Mississippi: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

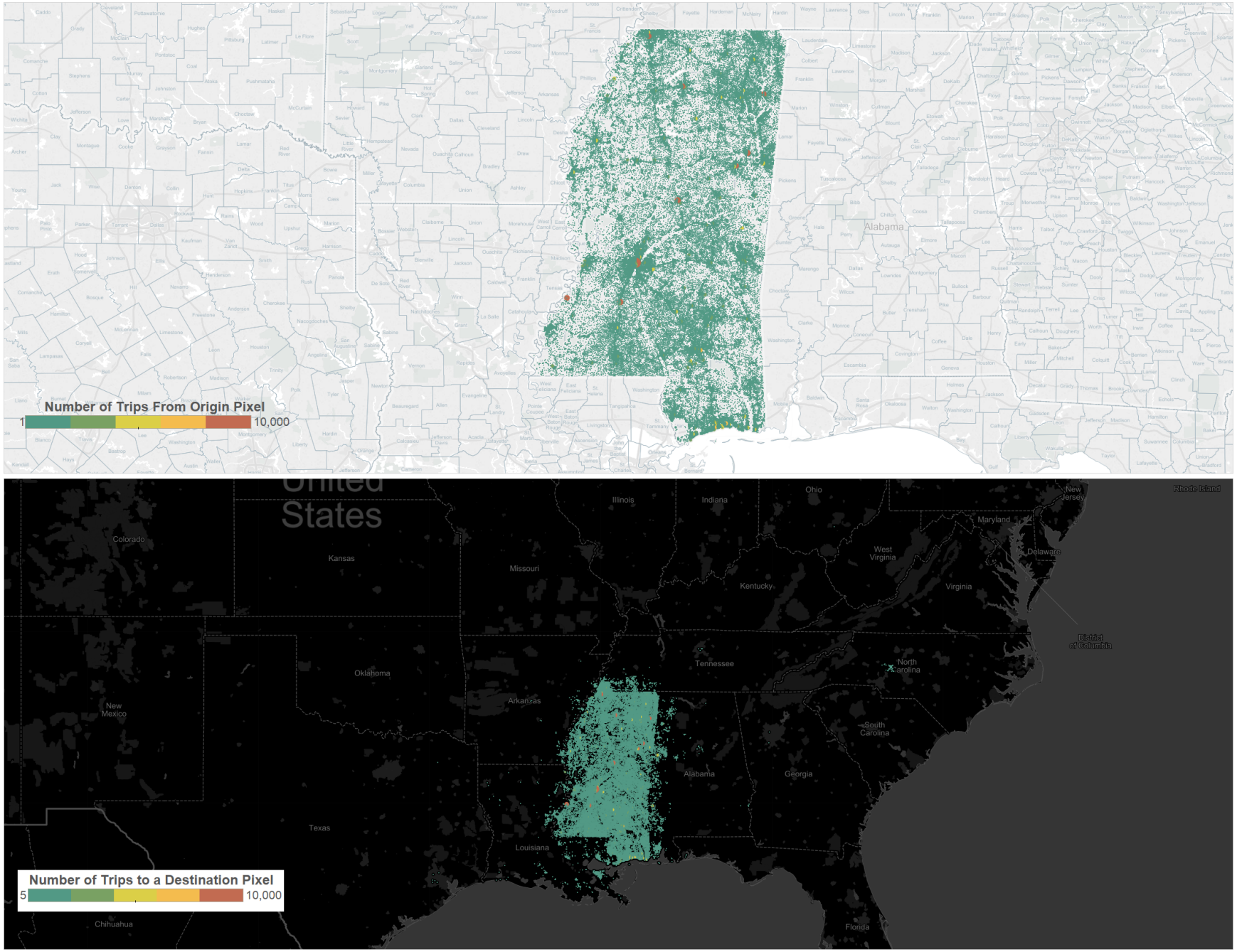

Figure A.67: Mississippi: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

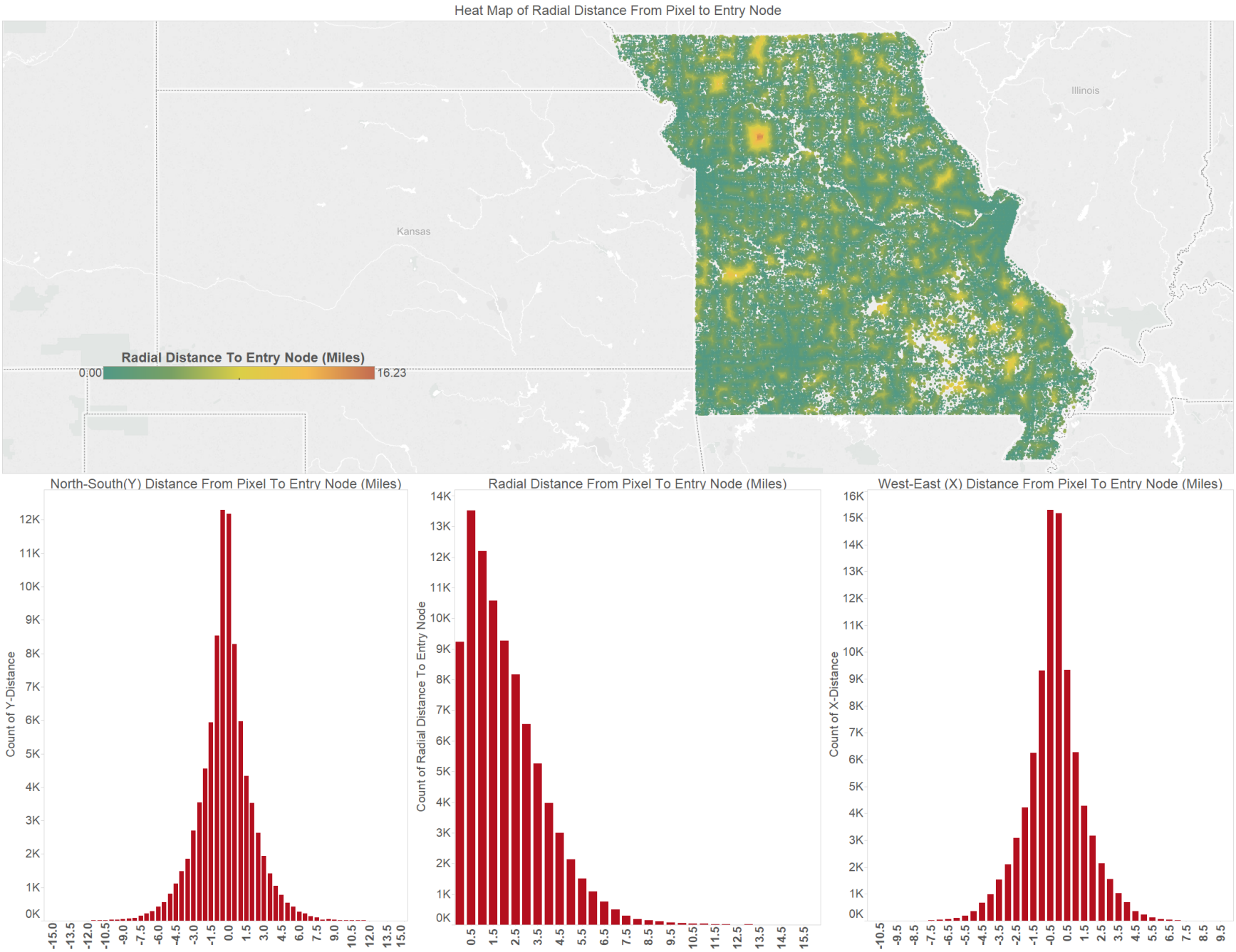

Figure A.68: Missouri: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

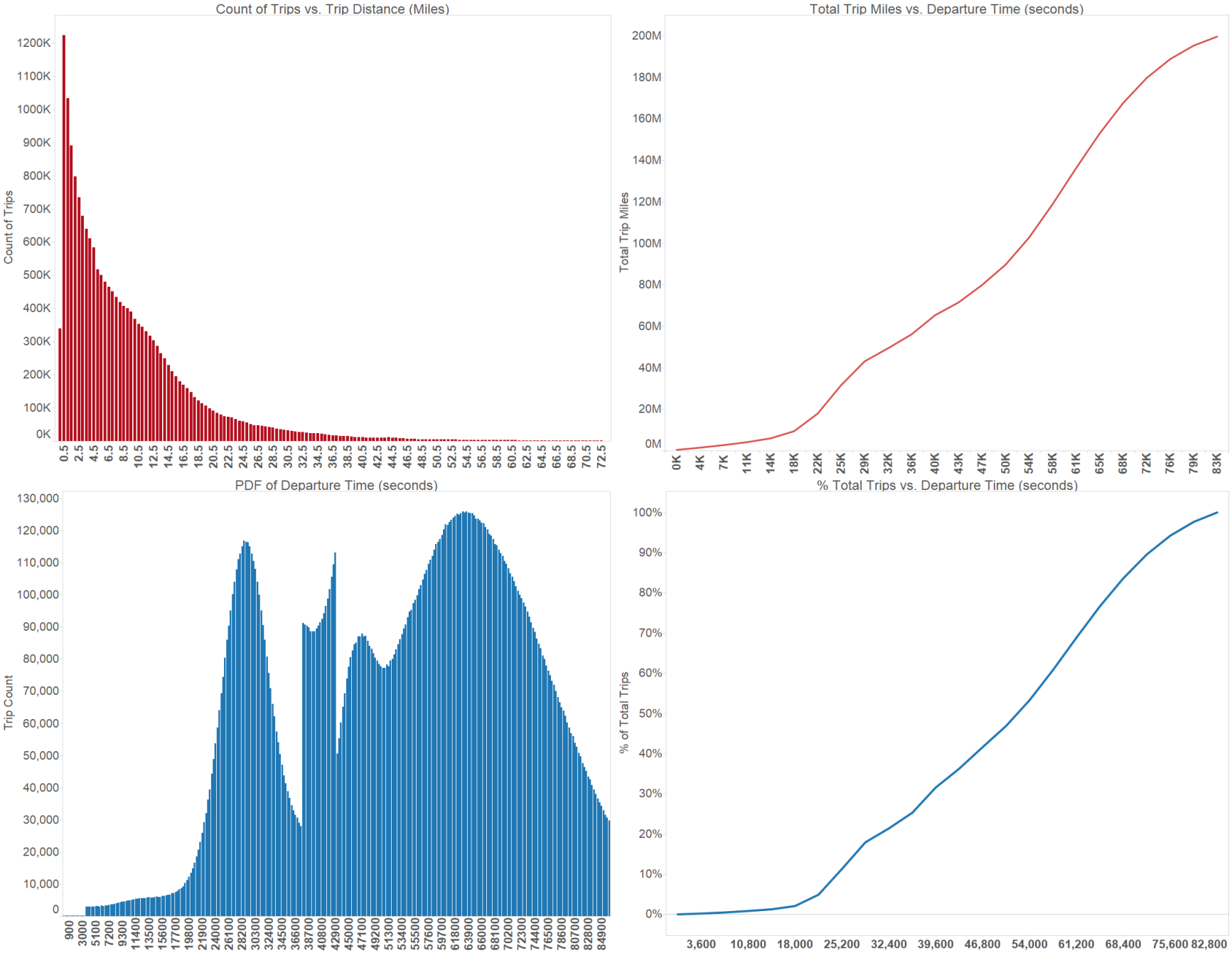

Figure A.69: Missouri: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

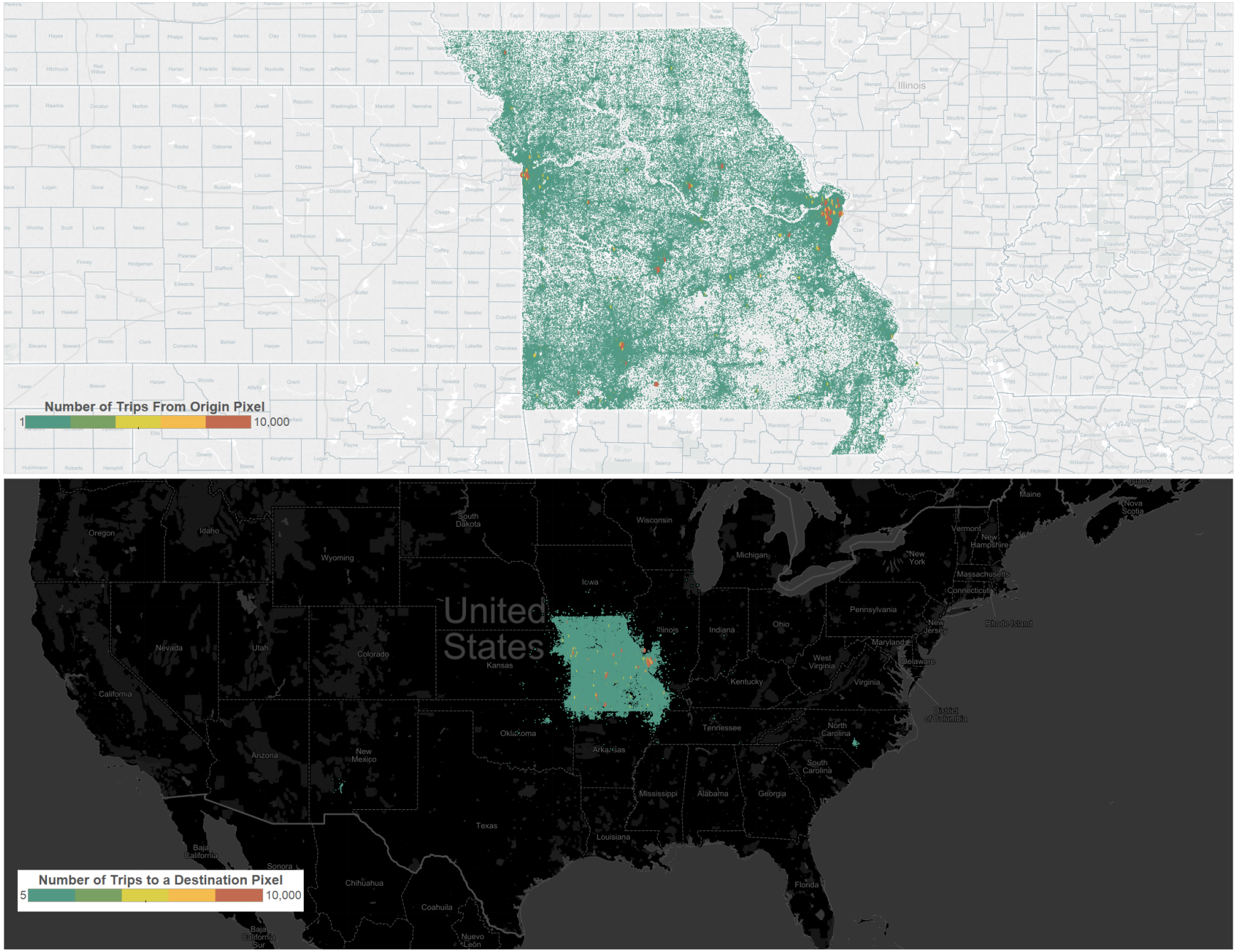

Figure A.70: Missouri: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

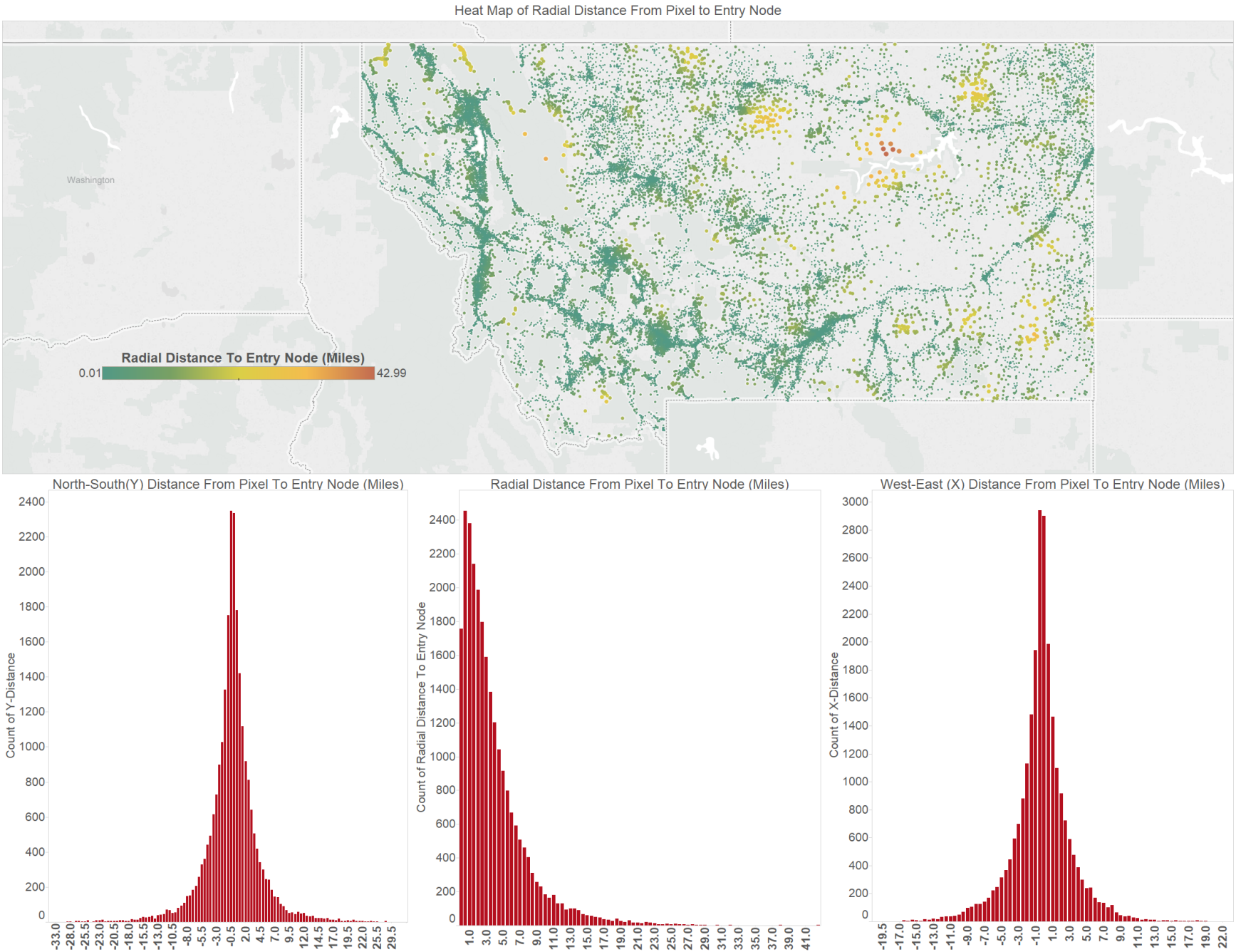

Figure A.71: Montana: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

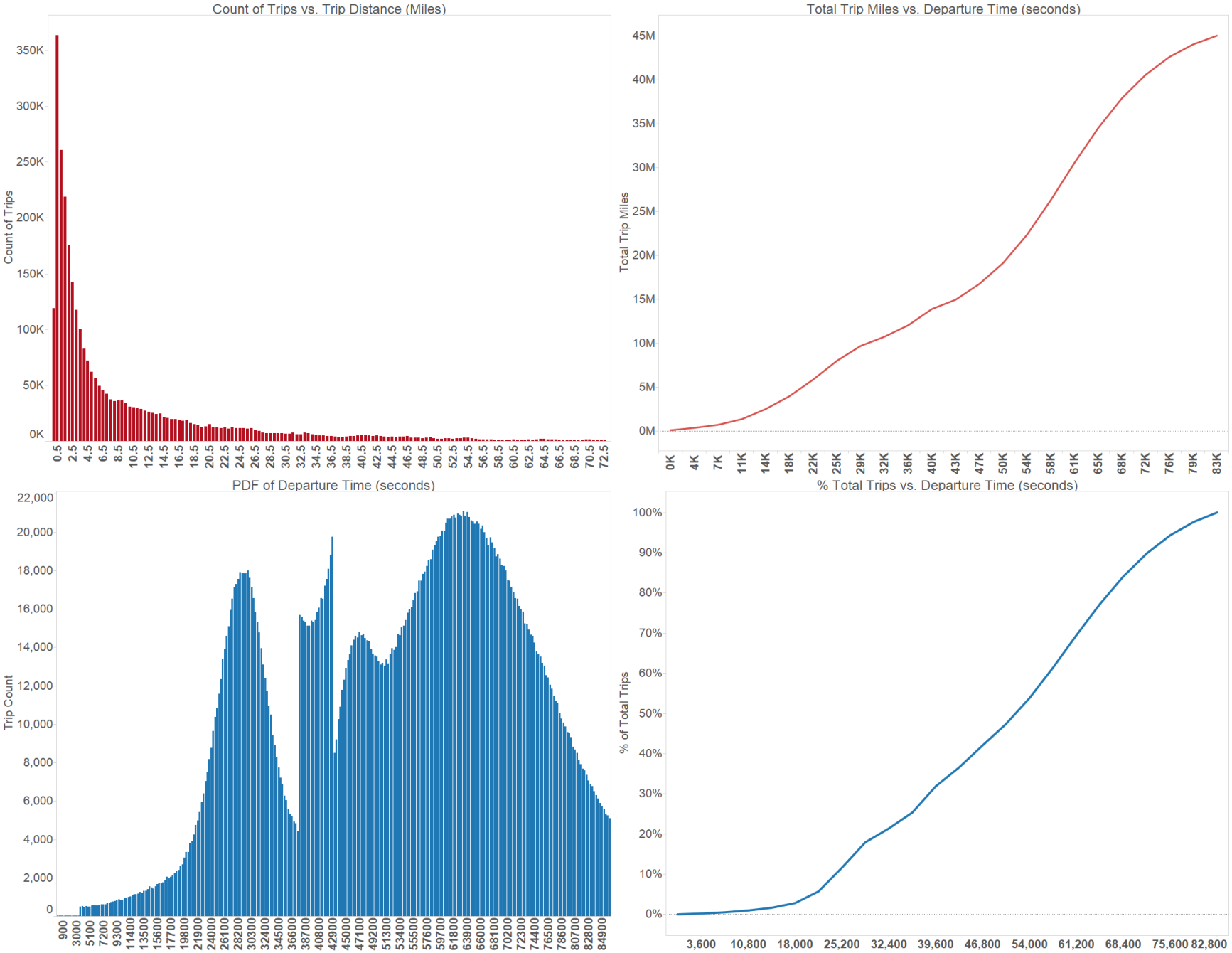

Figure A.72: Montana: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

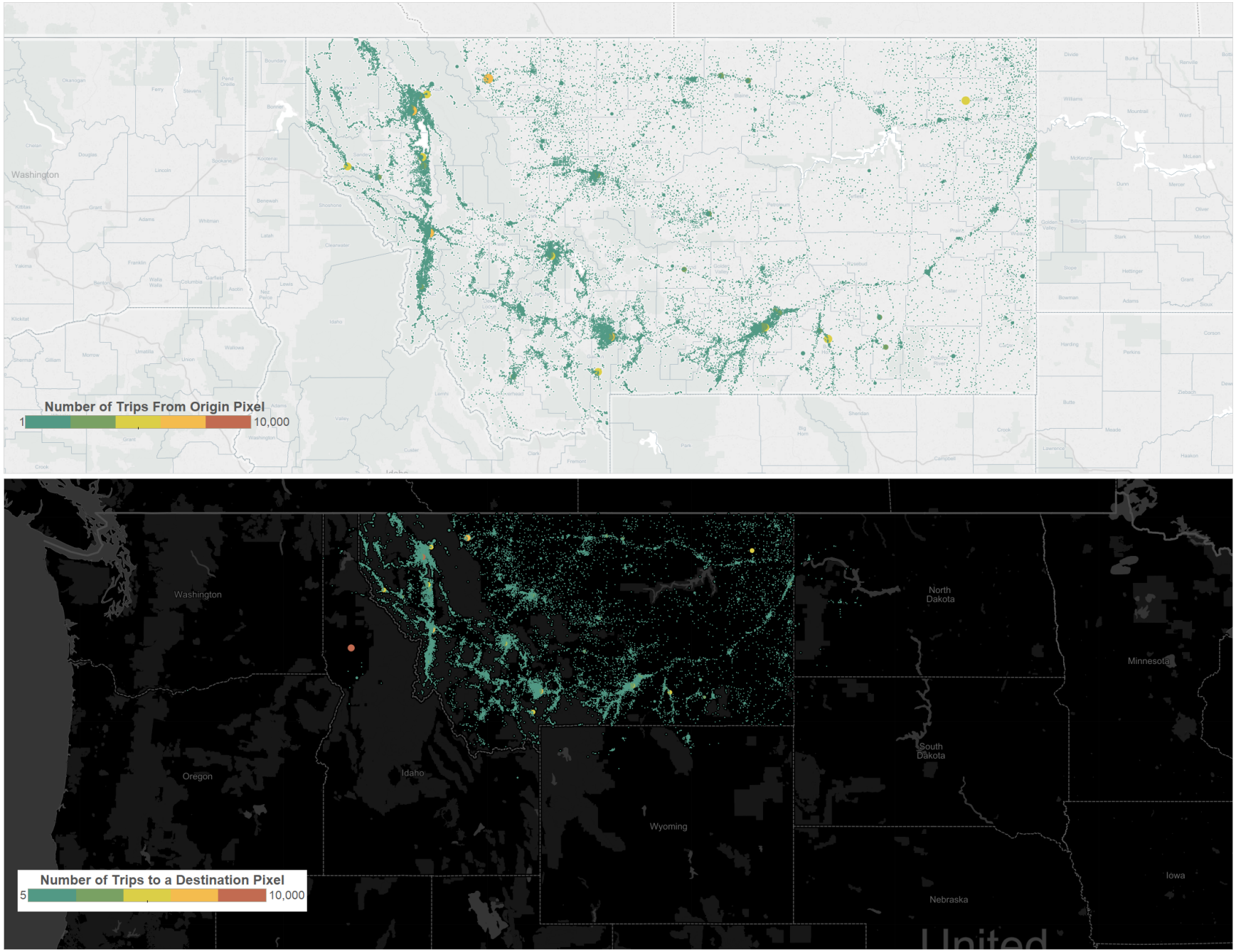

Figure A.73: Montana: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

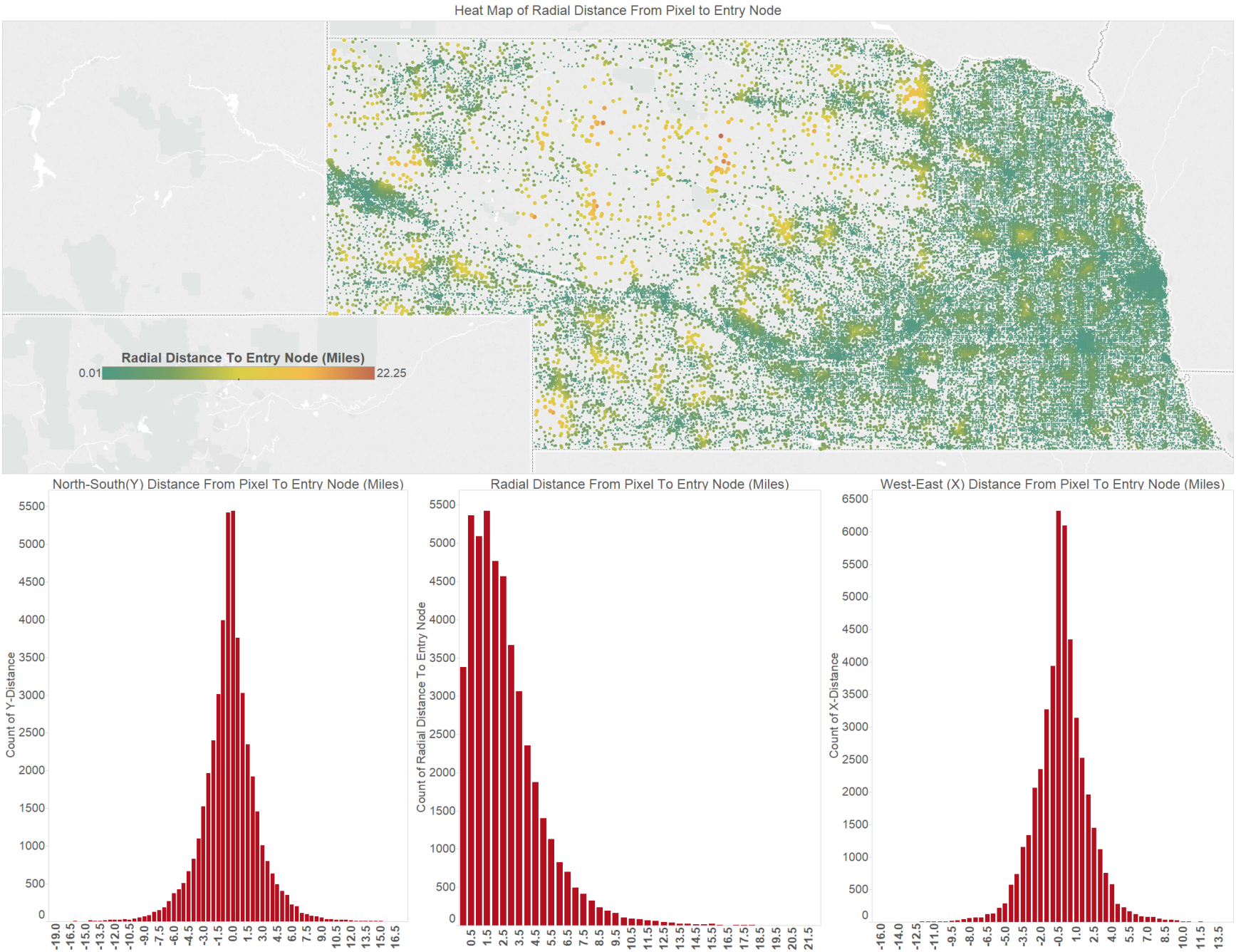

Figure A.74: Nebraska: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

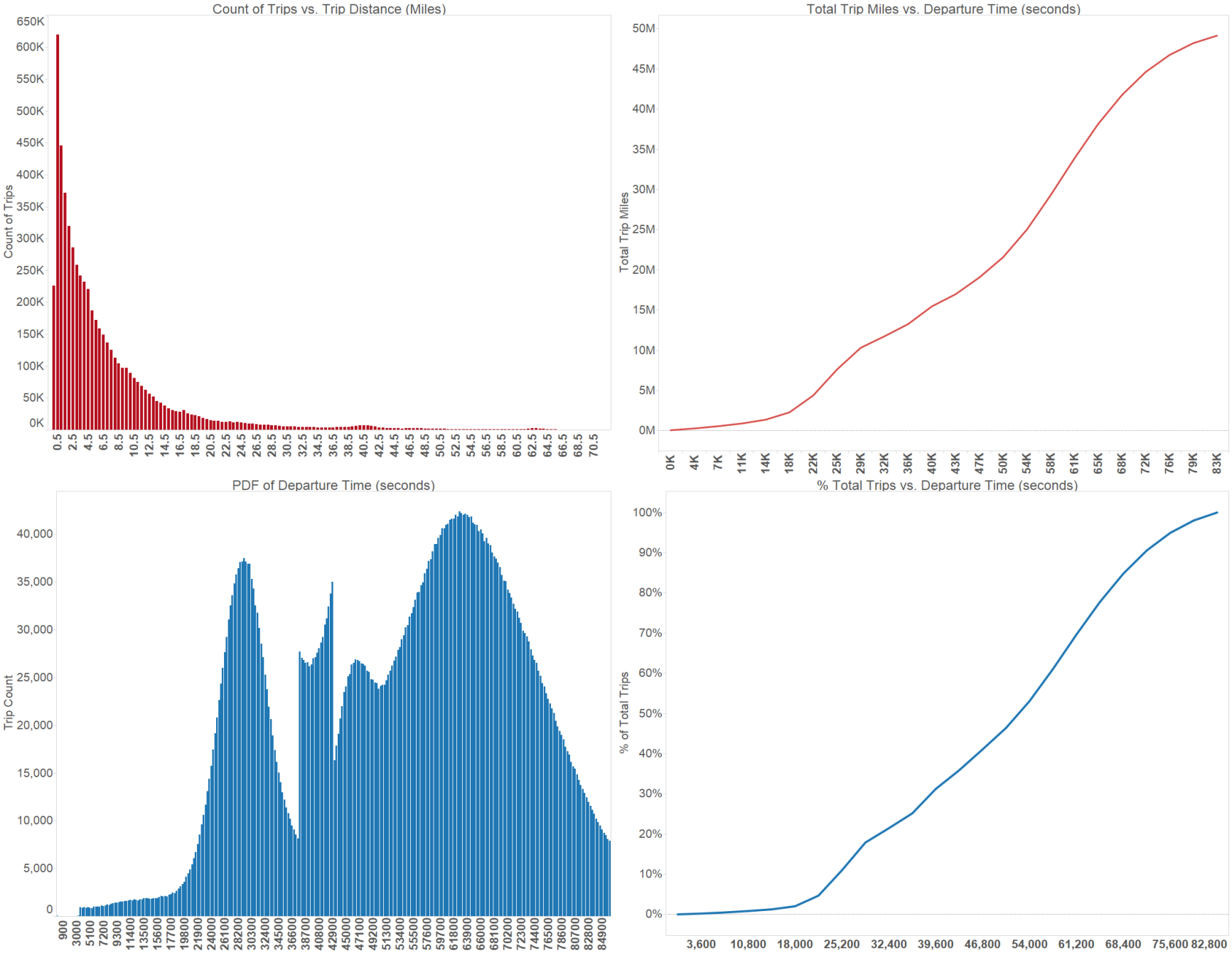

Figure A.75: Nebraska: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

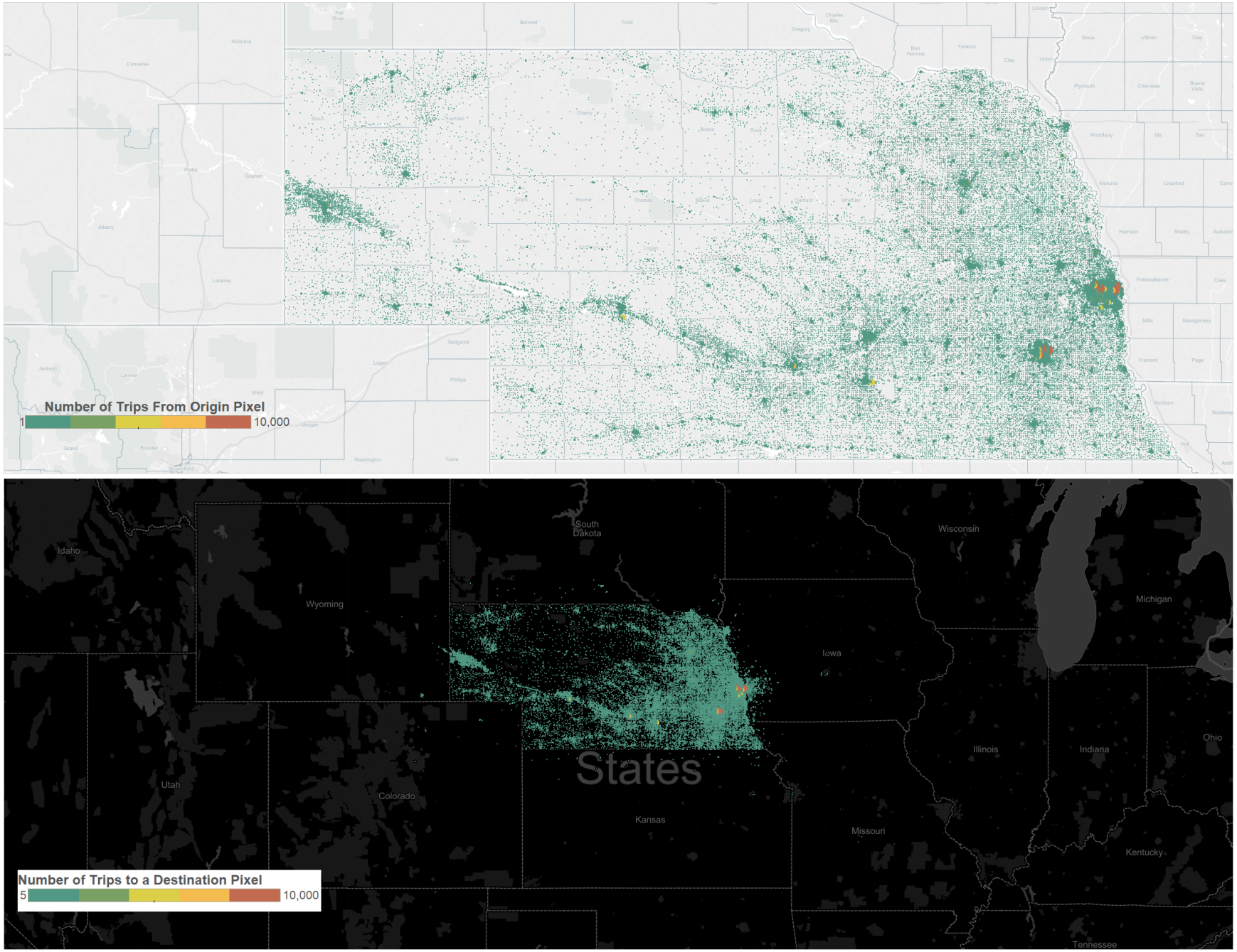

Figure A.76: Nebraska: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

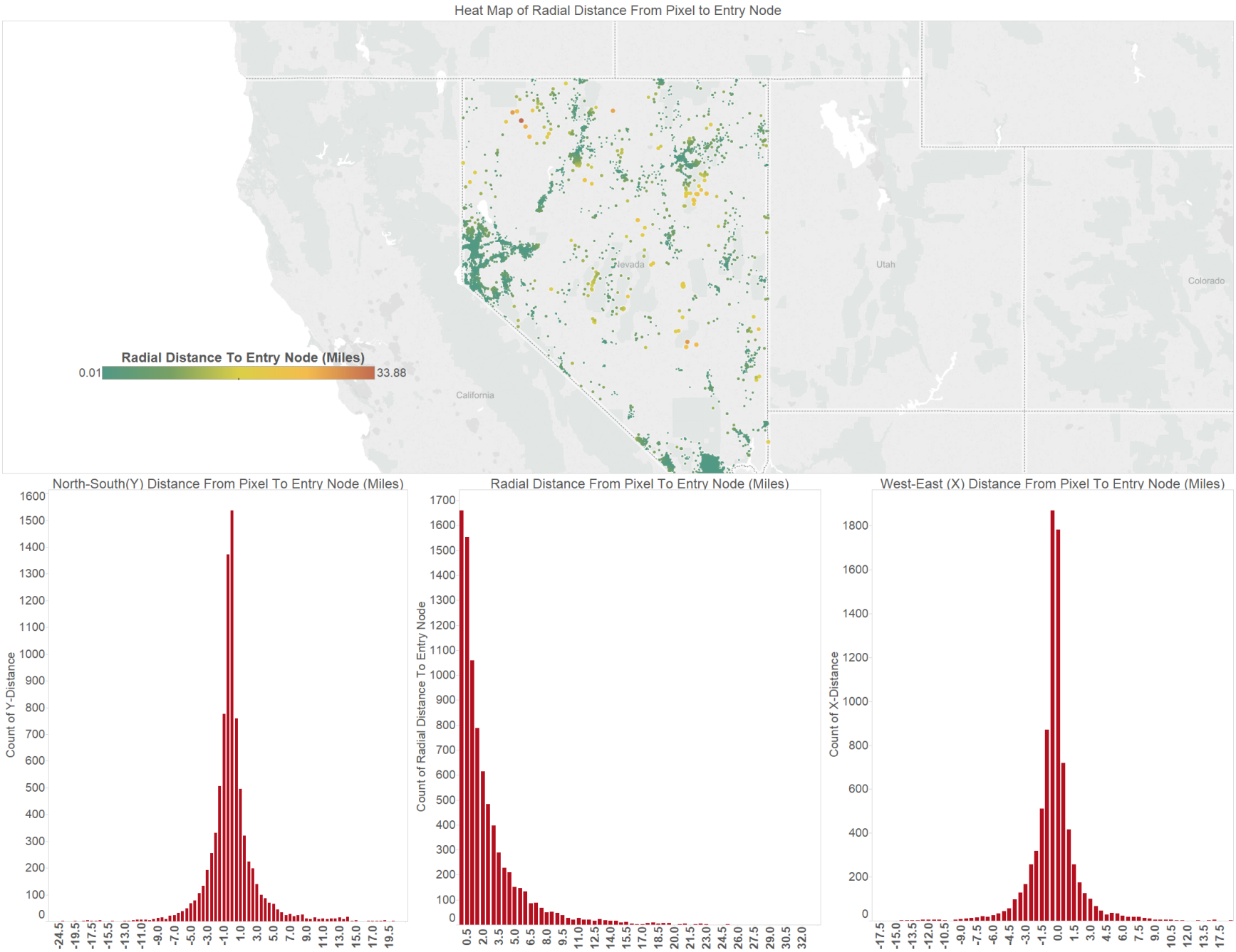

Figure A.77: Nevada: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

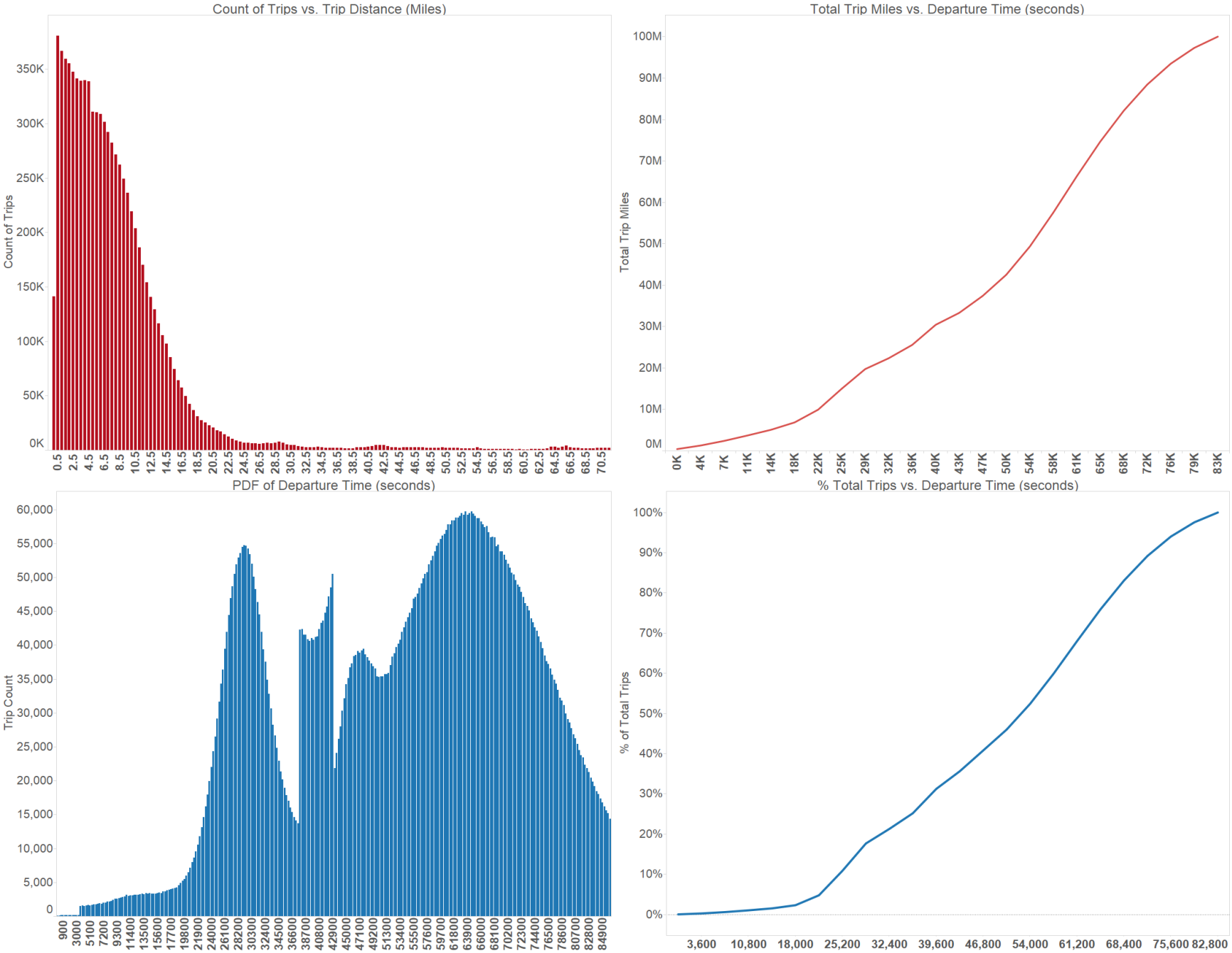

Figure A.78: Nevada: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

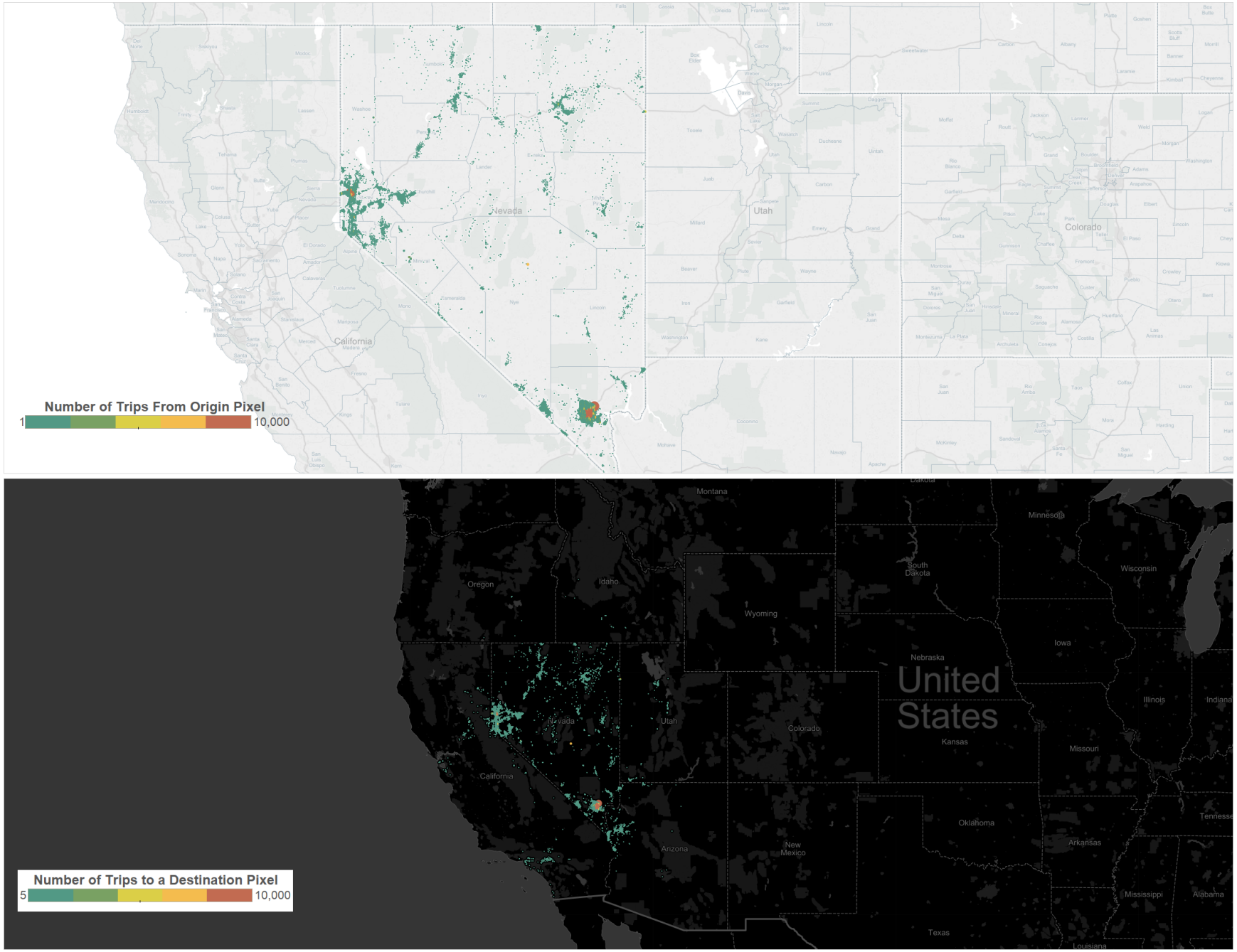

Figure A.79: Nevada: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

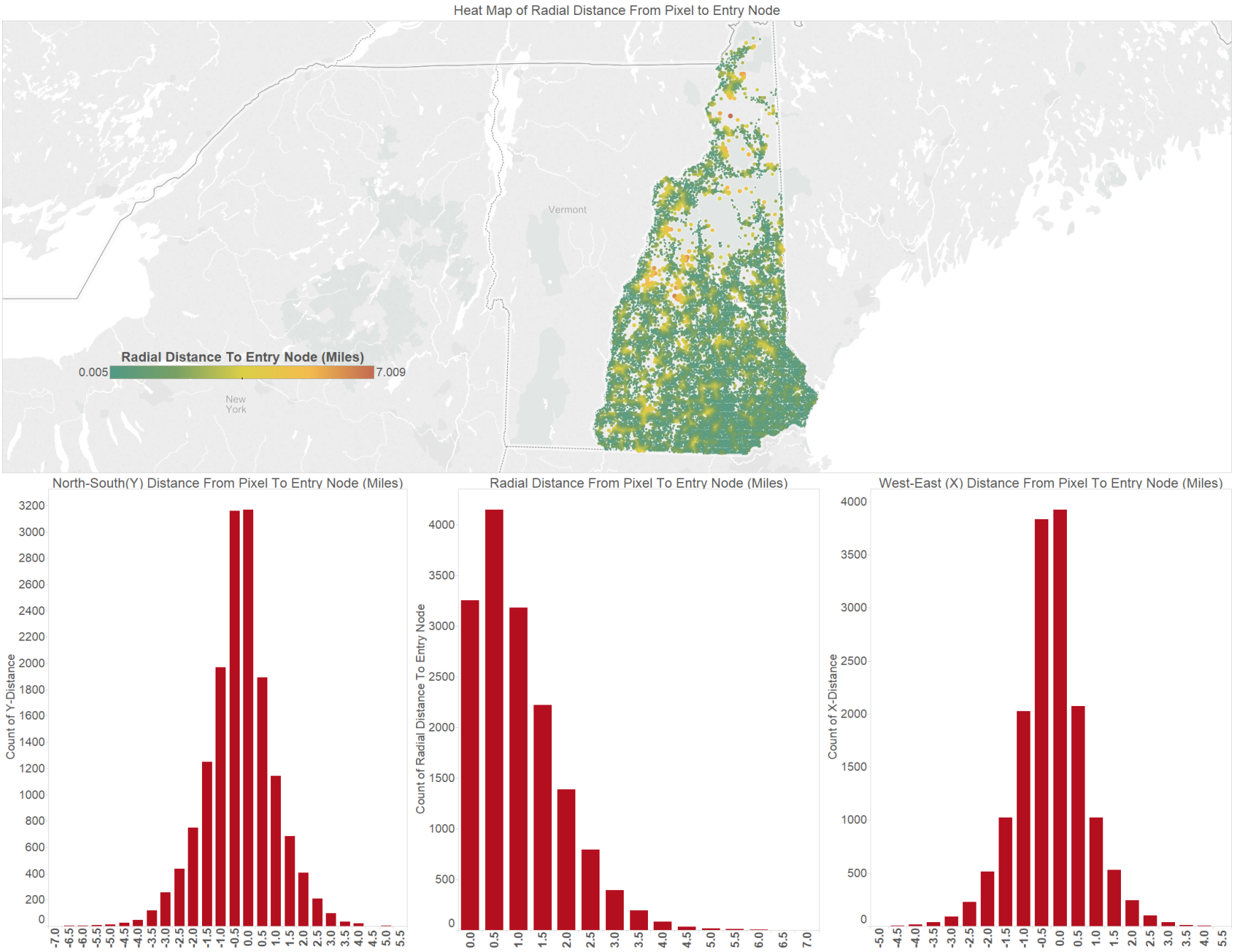

Figure A.80: New_Hampshire: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

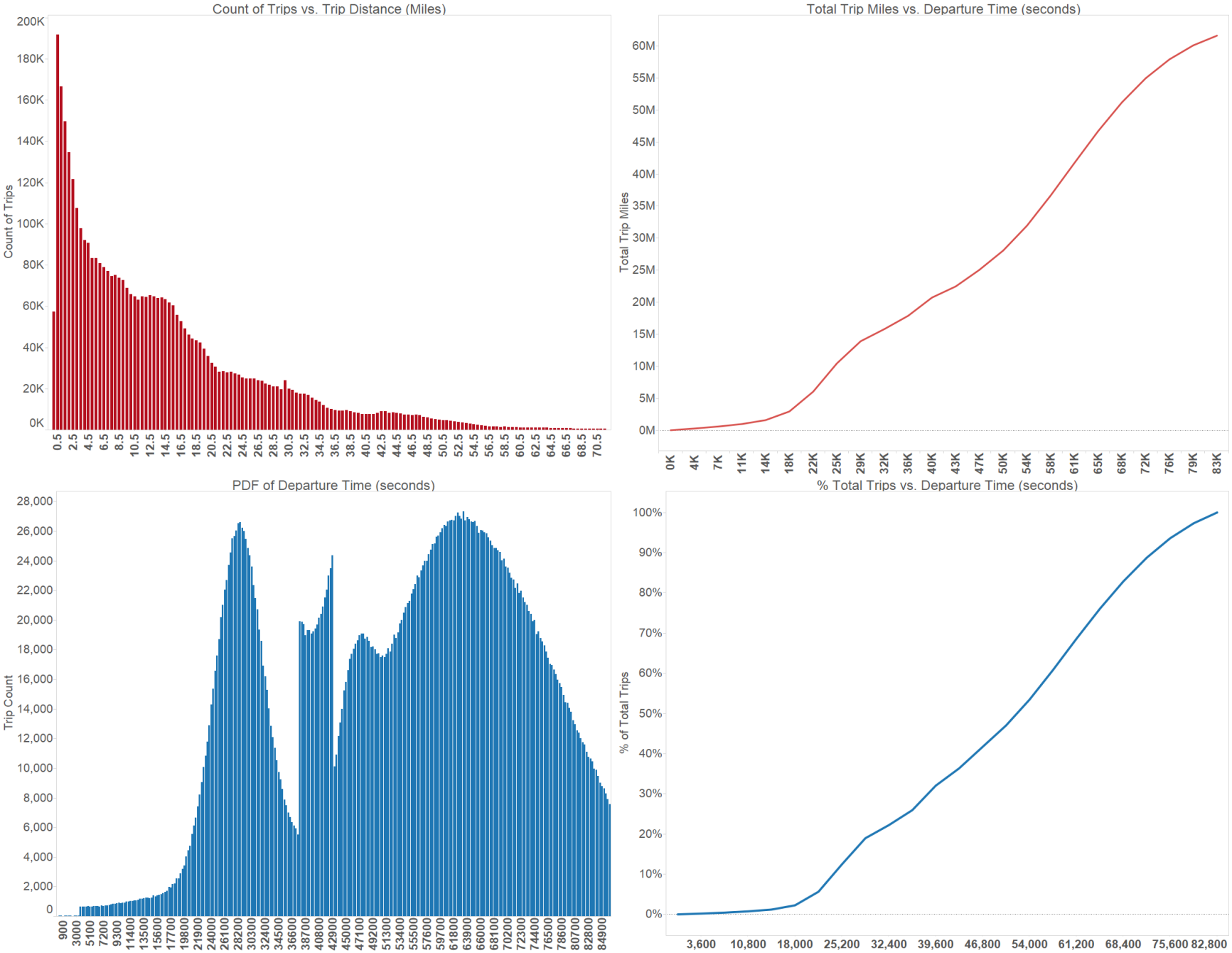

Figure A.81: New_Hampshire: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

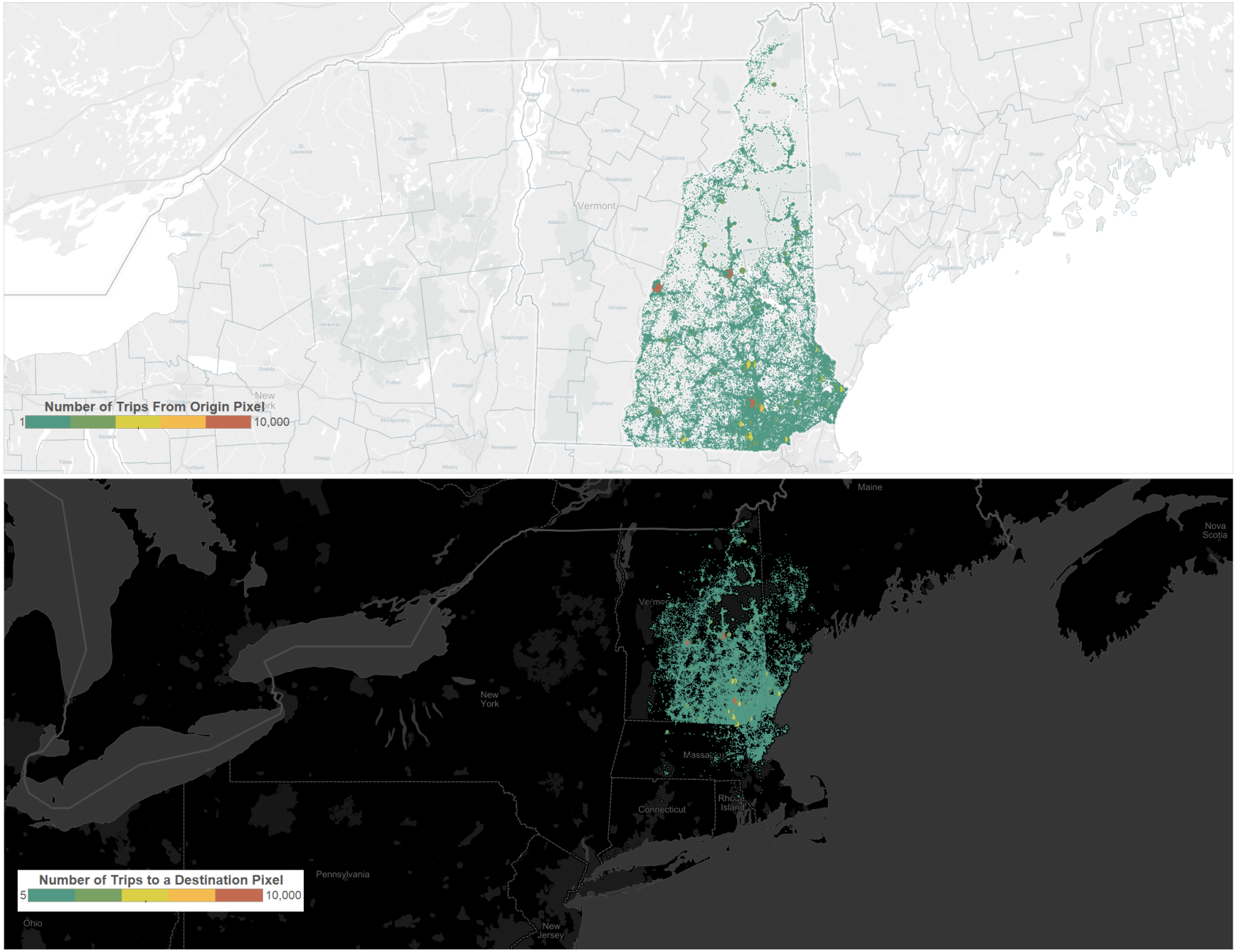

Figure A.82: New_Hampshire: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

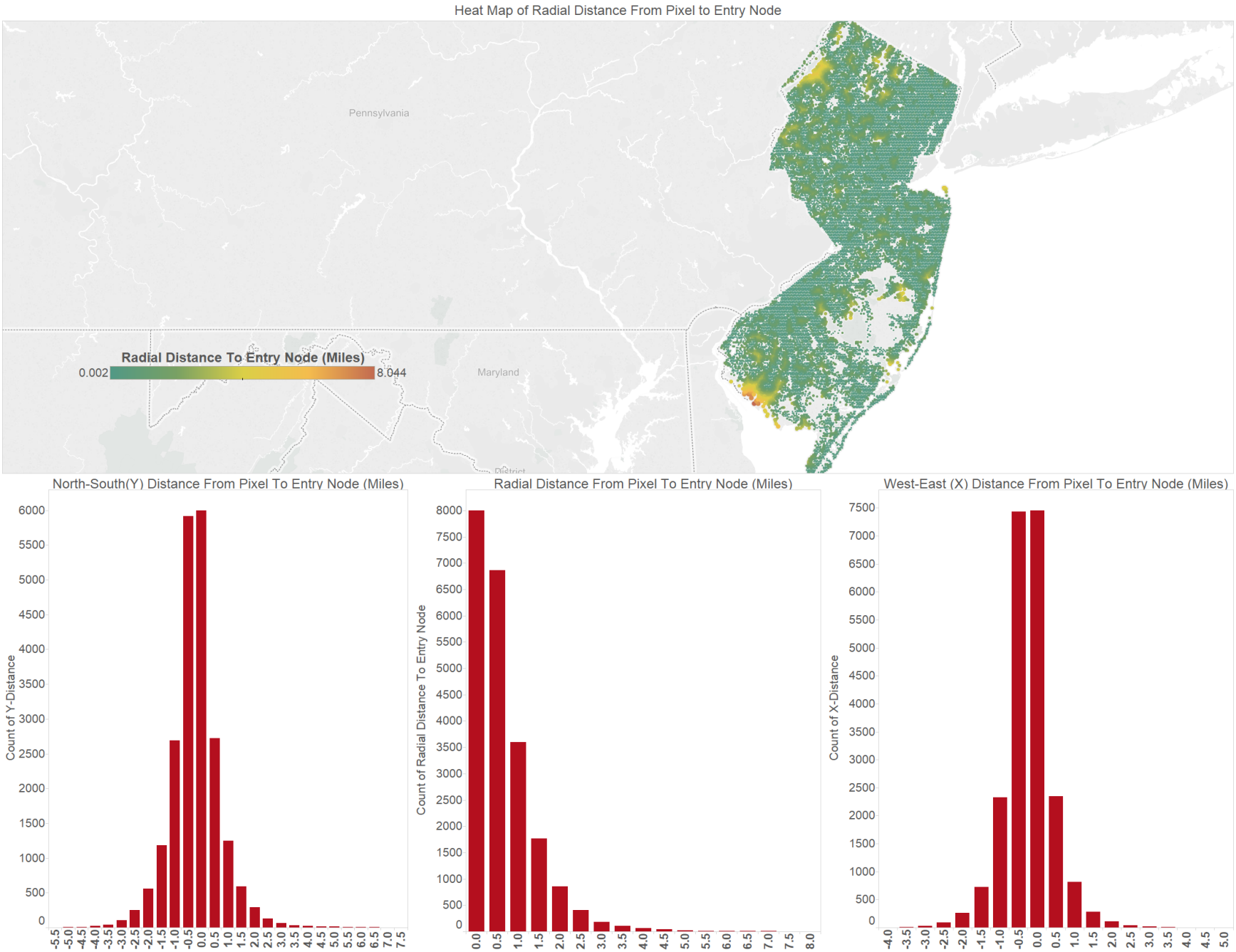

Figure A.83: New_Jersey: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

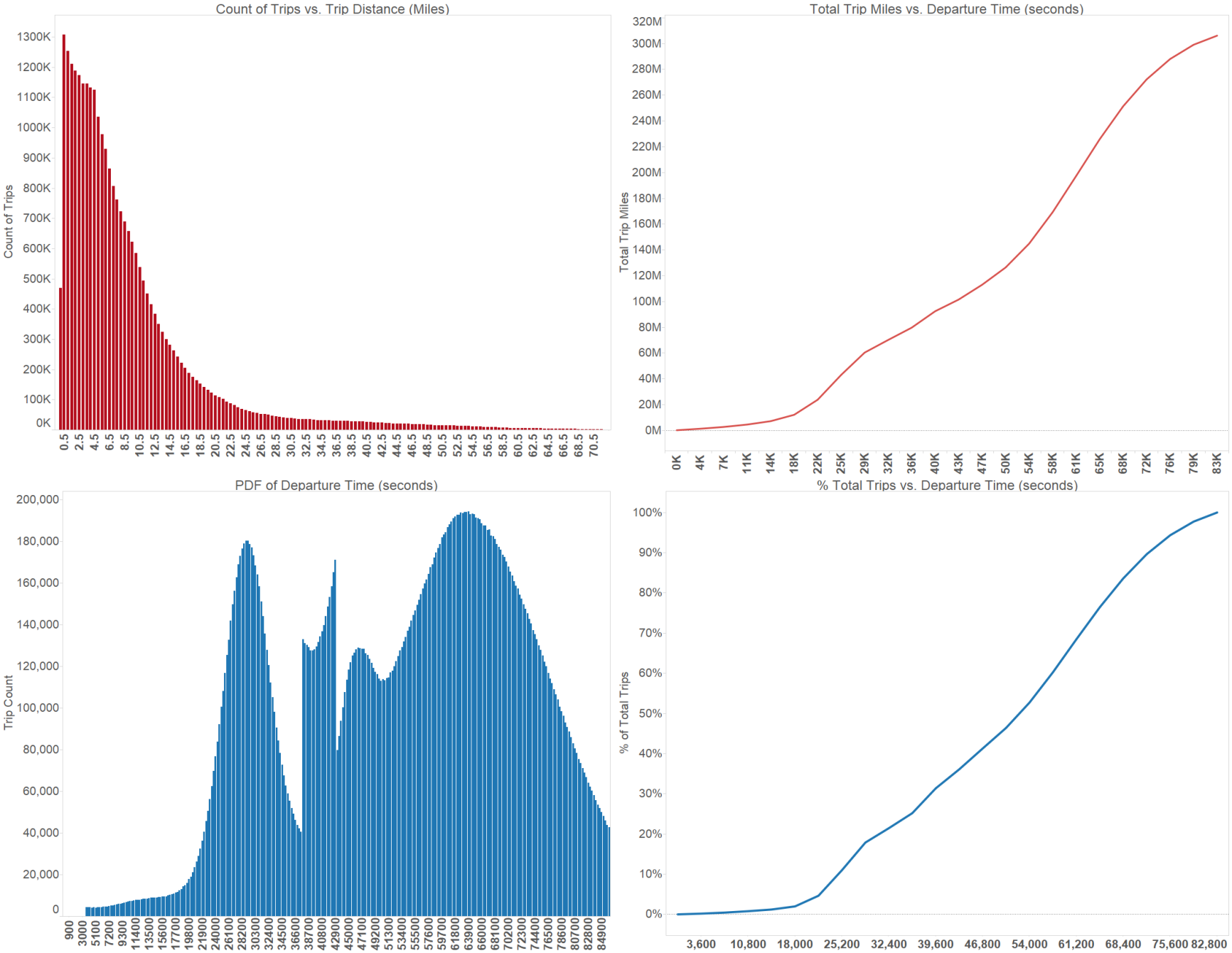

Figure A.84: New_Jersey: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

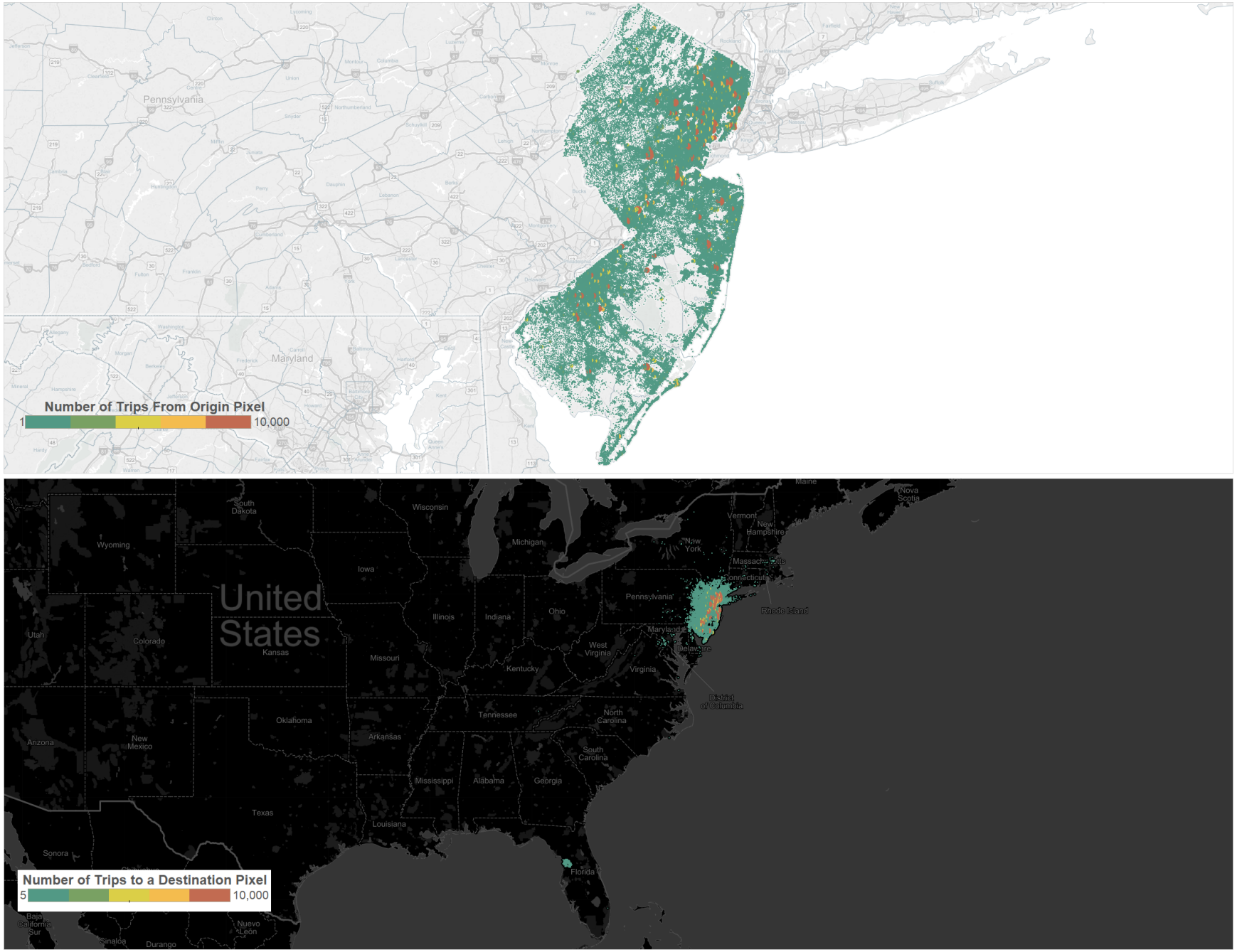

Figure A.85: New_Jersey: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

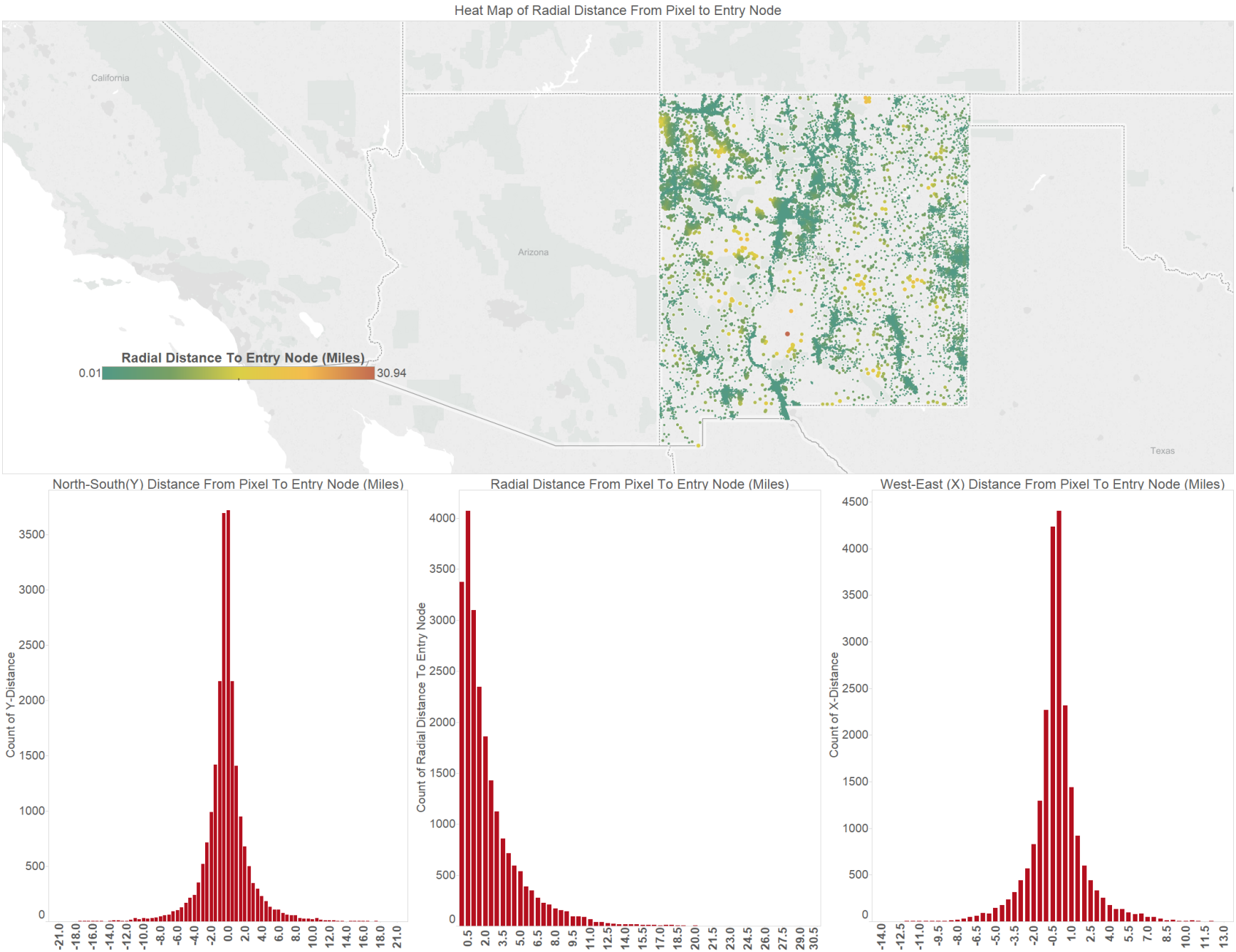

Figure A.86: New_Mexico: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

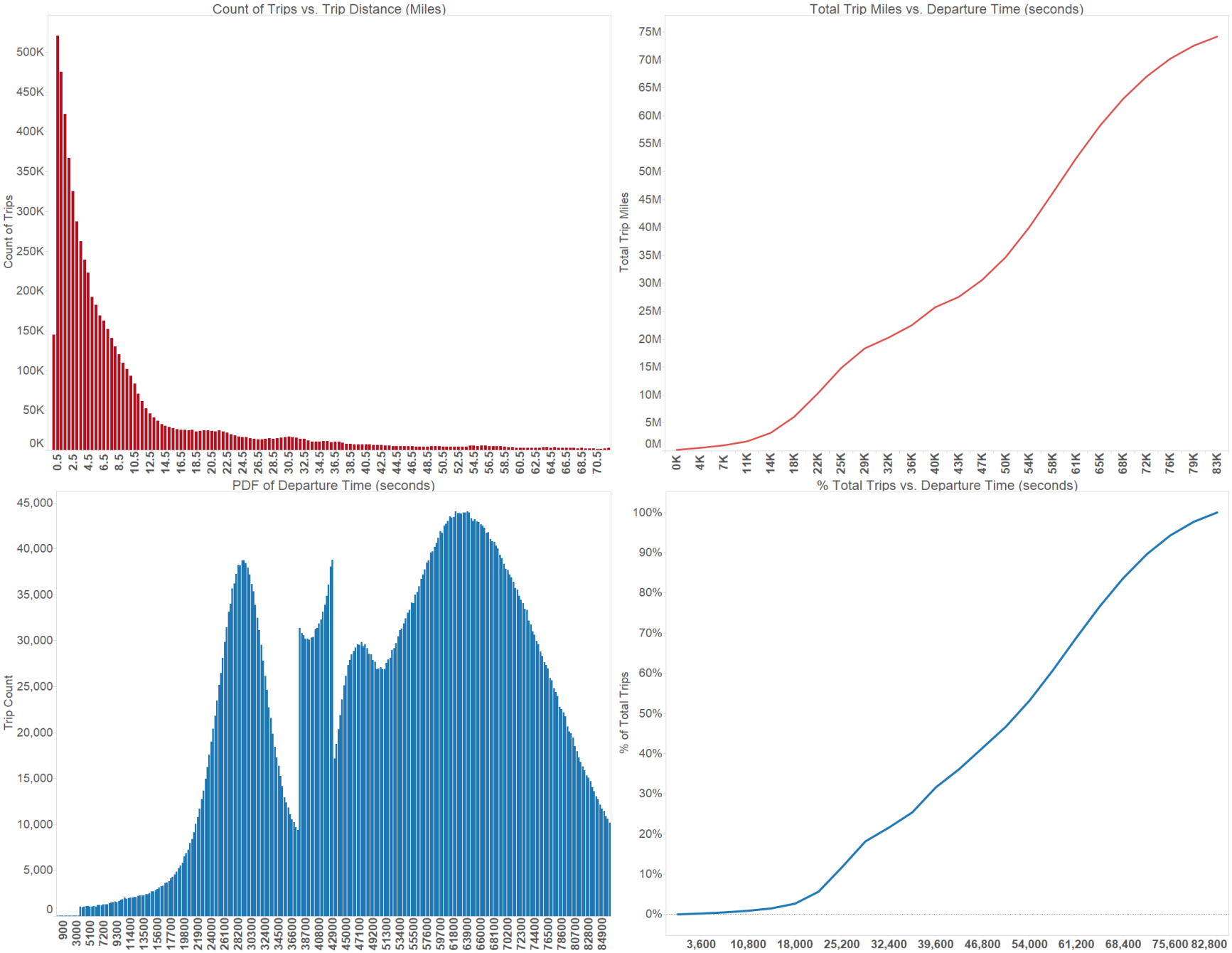

Figure A.87: New_Mexico: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

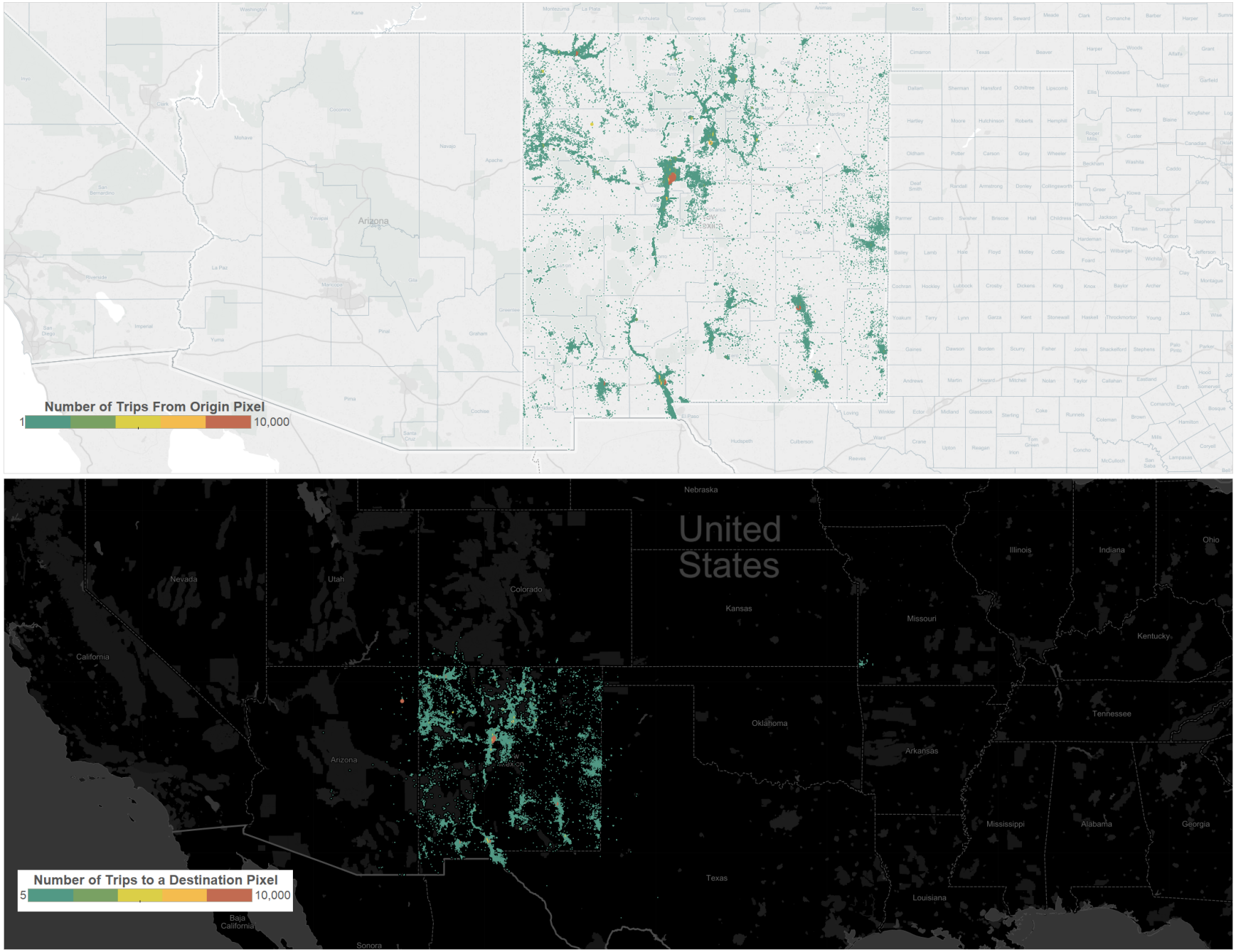

Figure A.88: New_Mexico: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

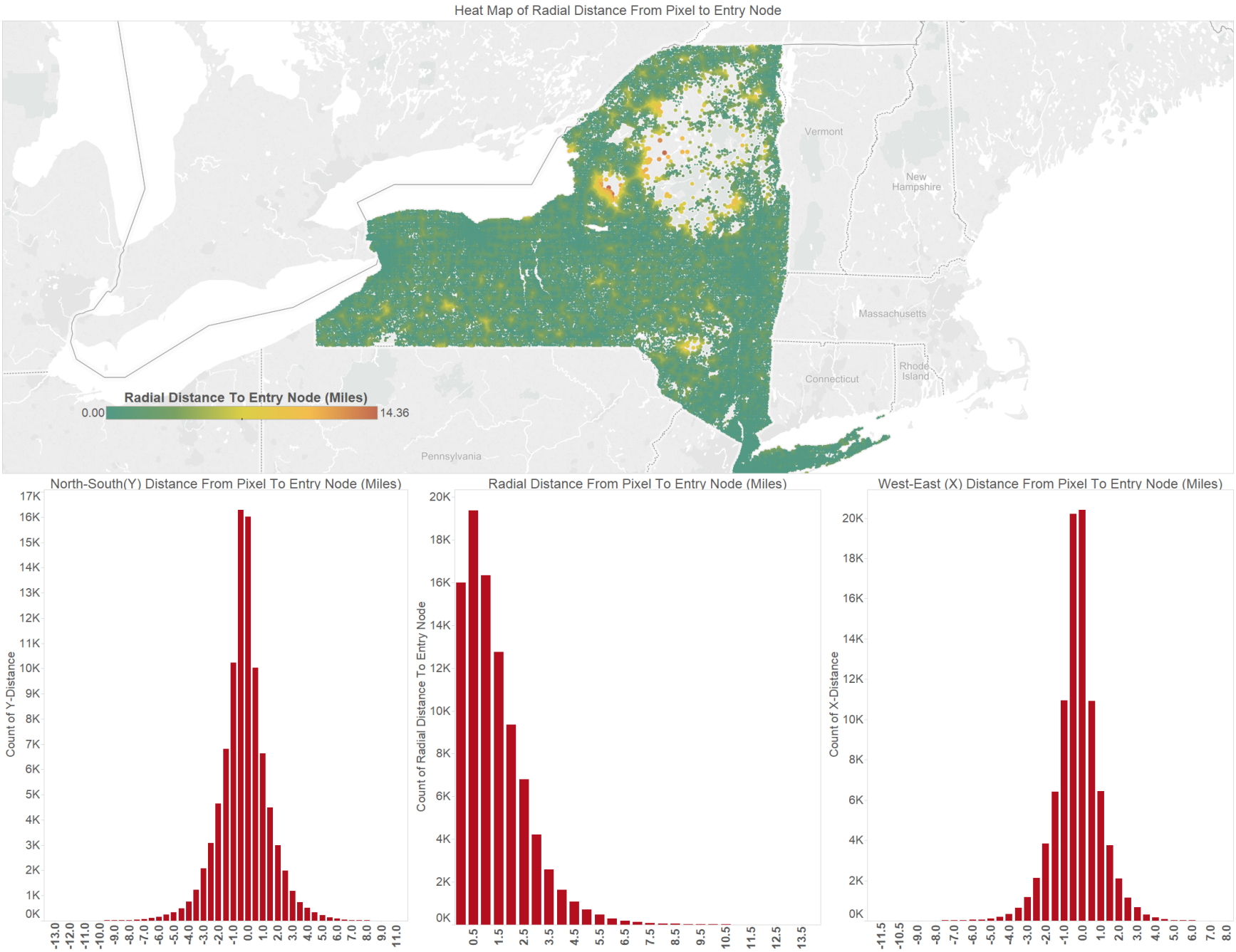

Figure A.89: New_York: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

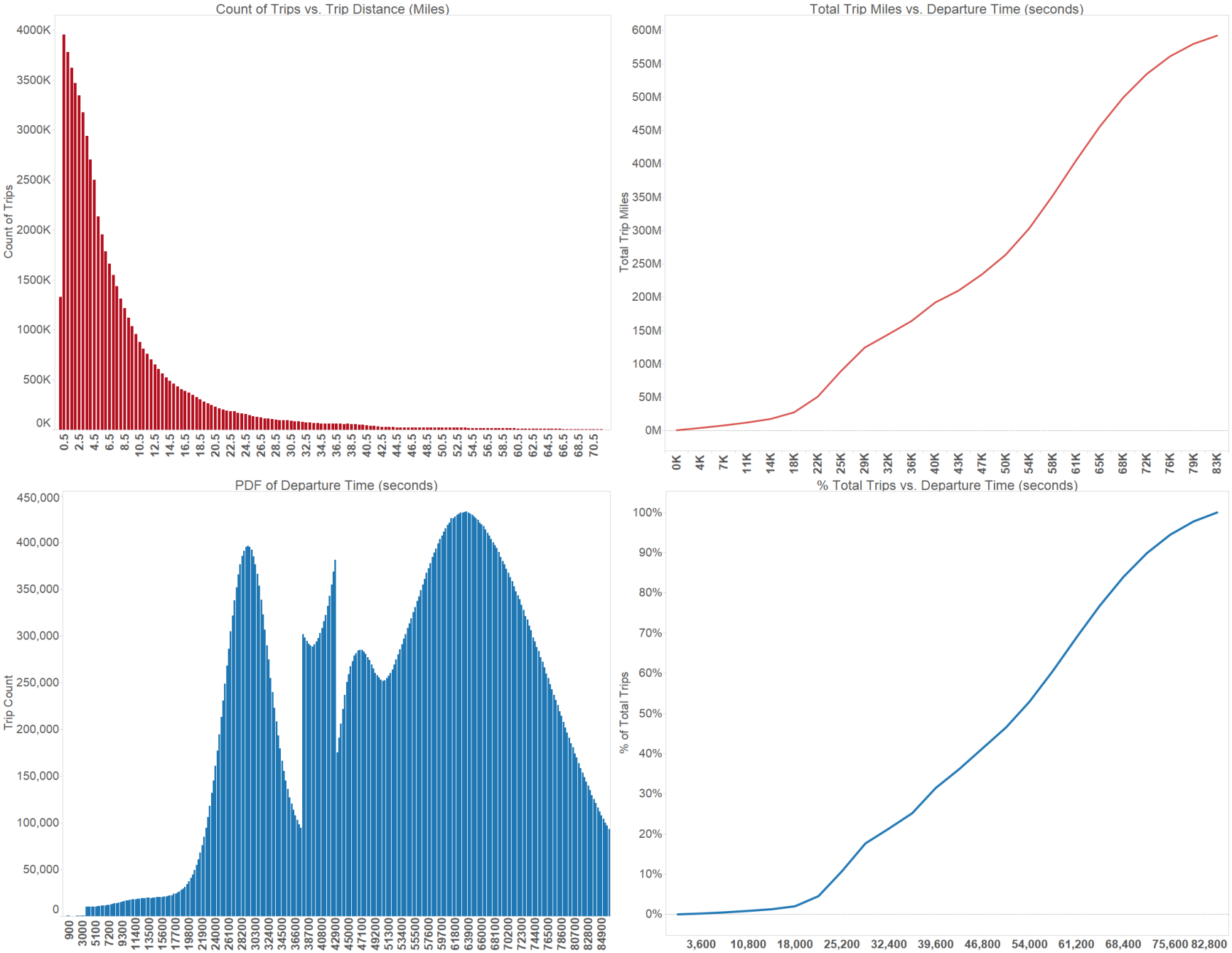

Figure A.90: New_York: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

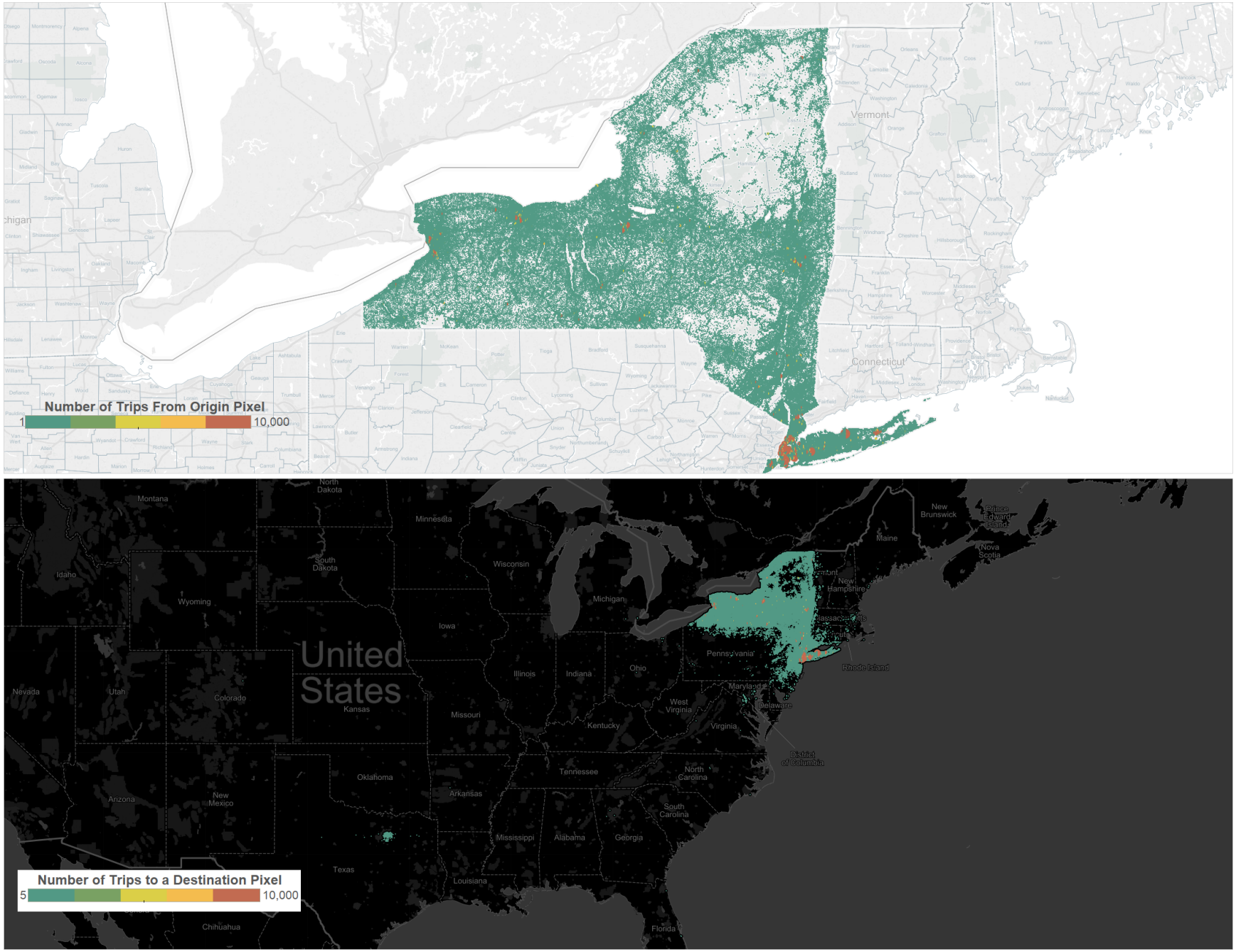

Figure A.91: New_York: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

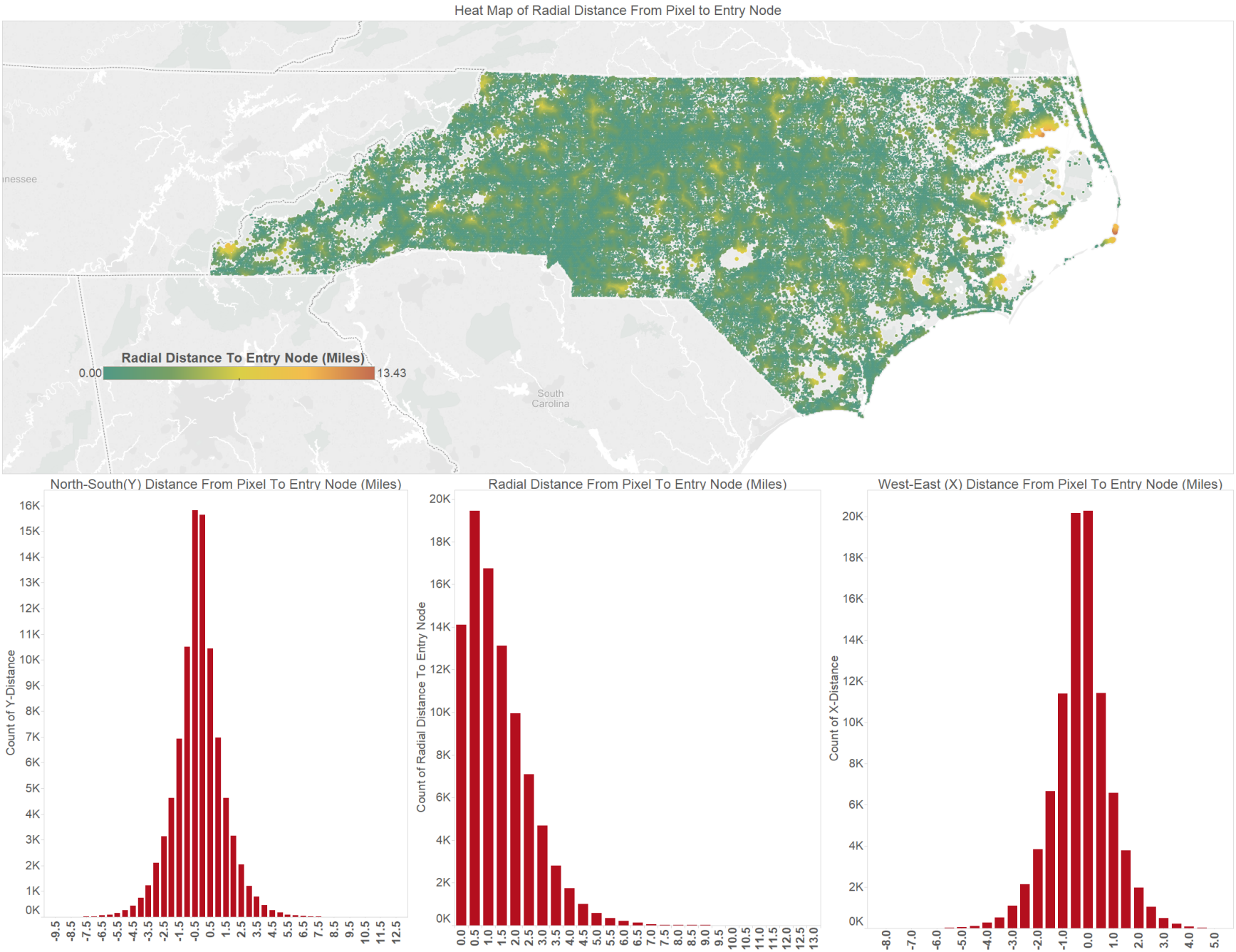

Figure A.92: North_Carolina: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

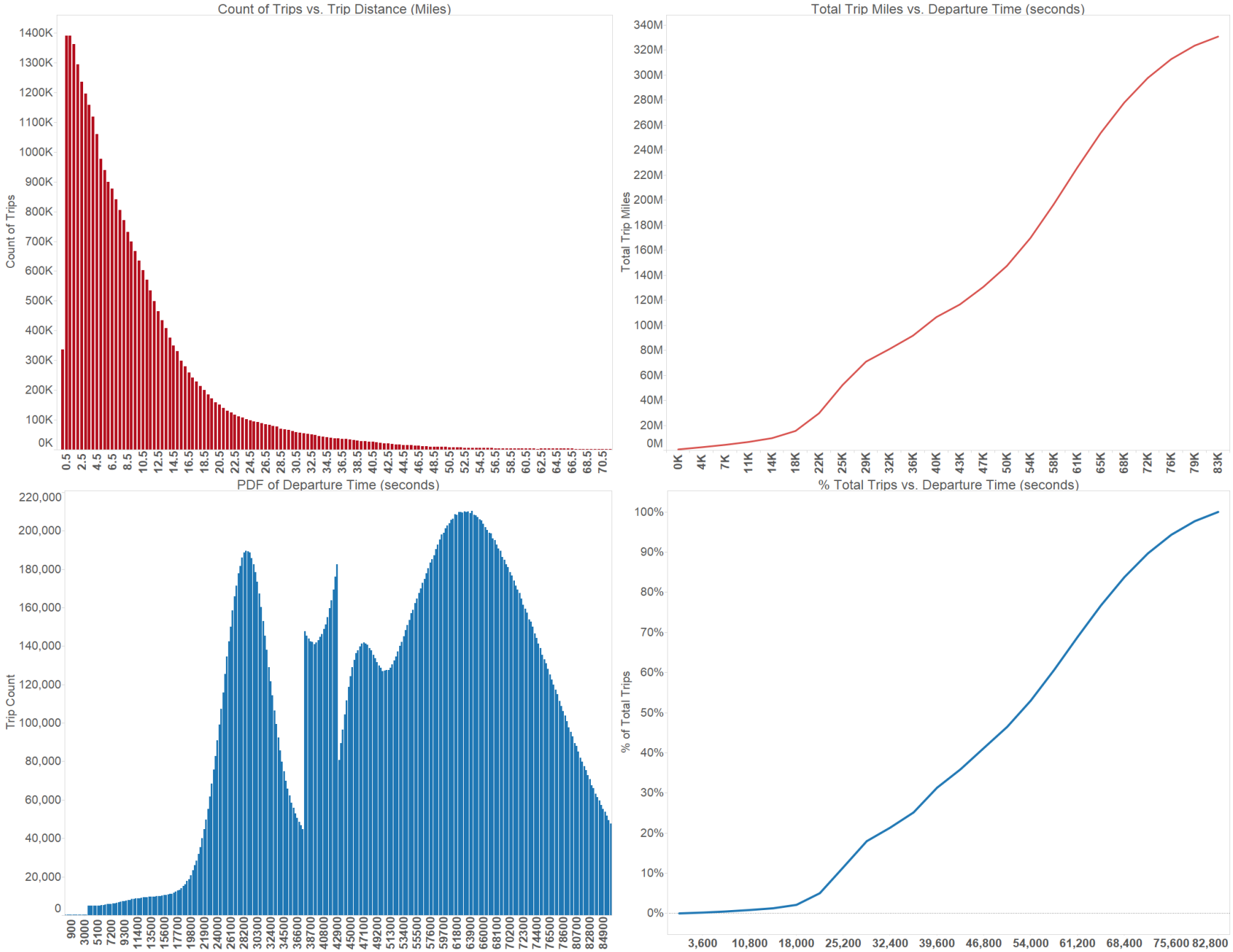

Figure A.93: North_Carolina: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

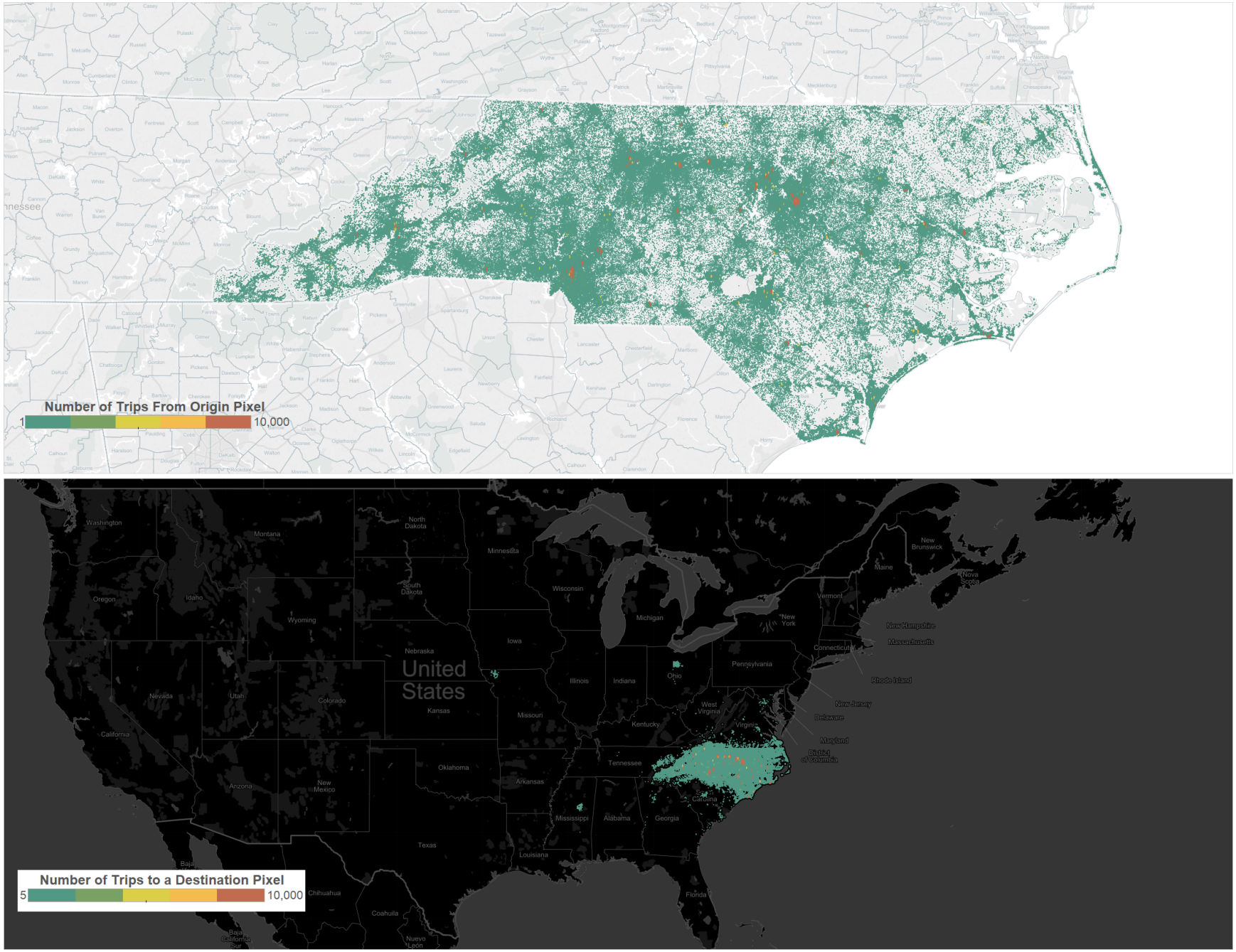

Figure A.94: North_Carolina: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

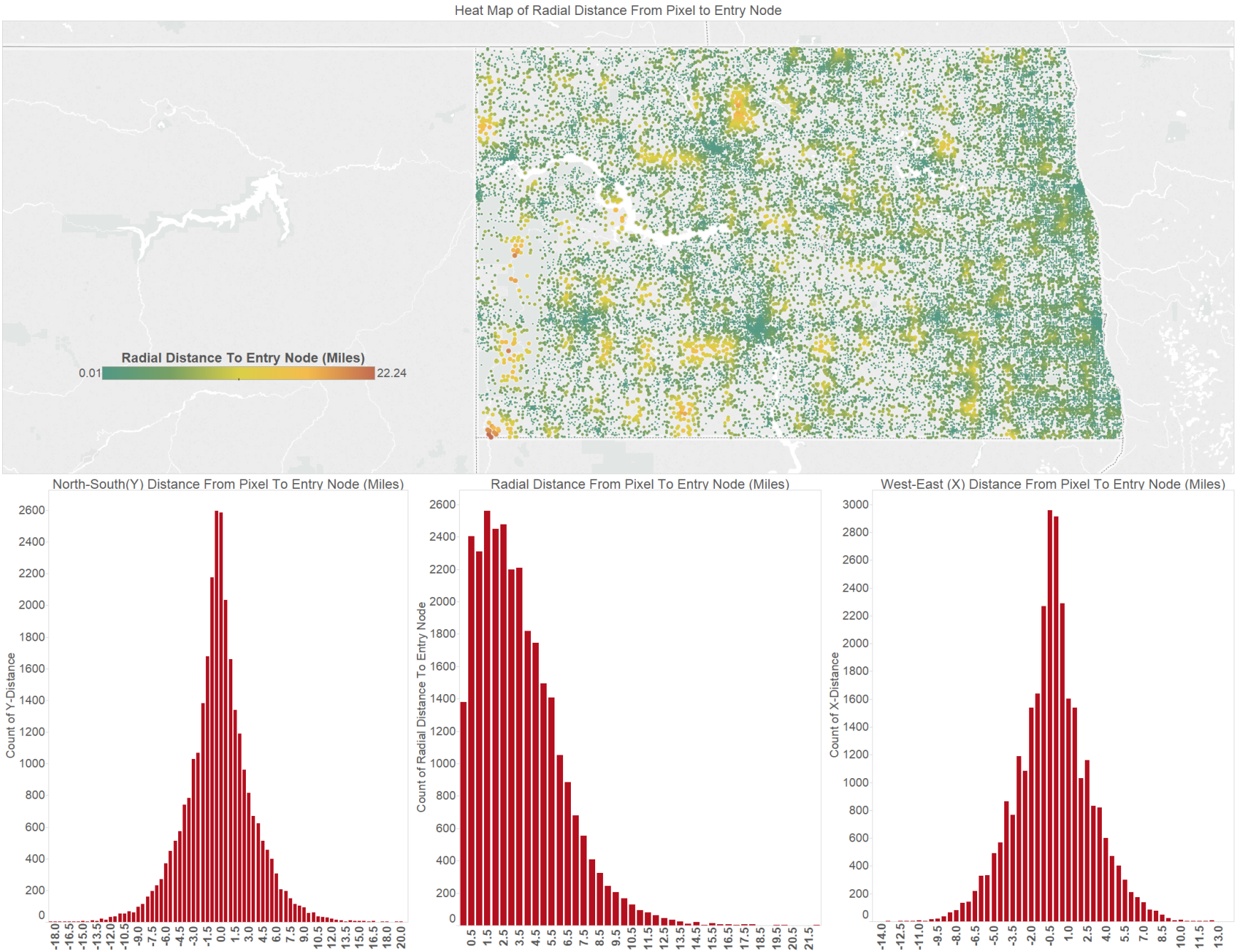

Figure A.95: North_Dakota: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

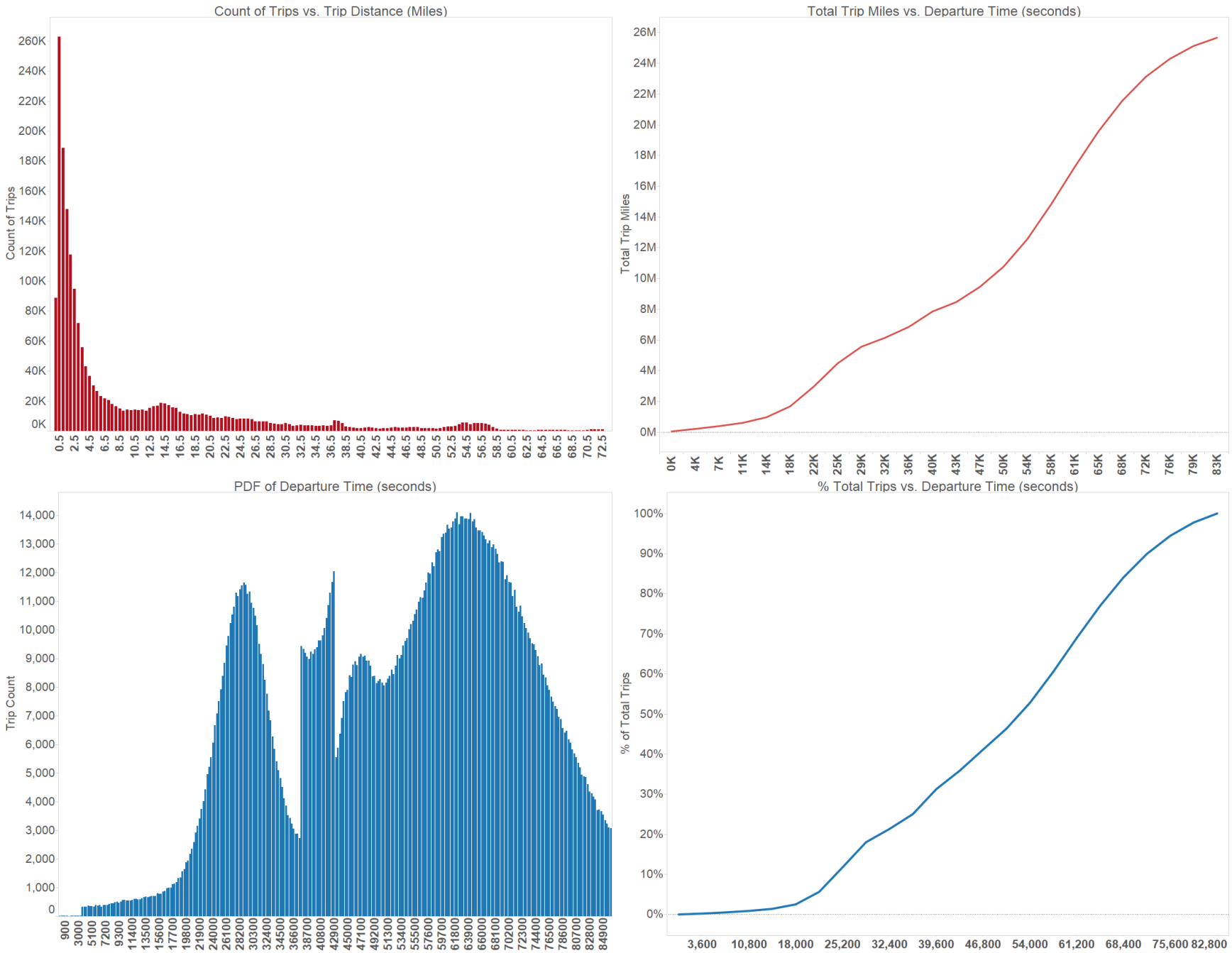

Figure A.96: North_Dakota: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

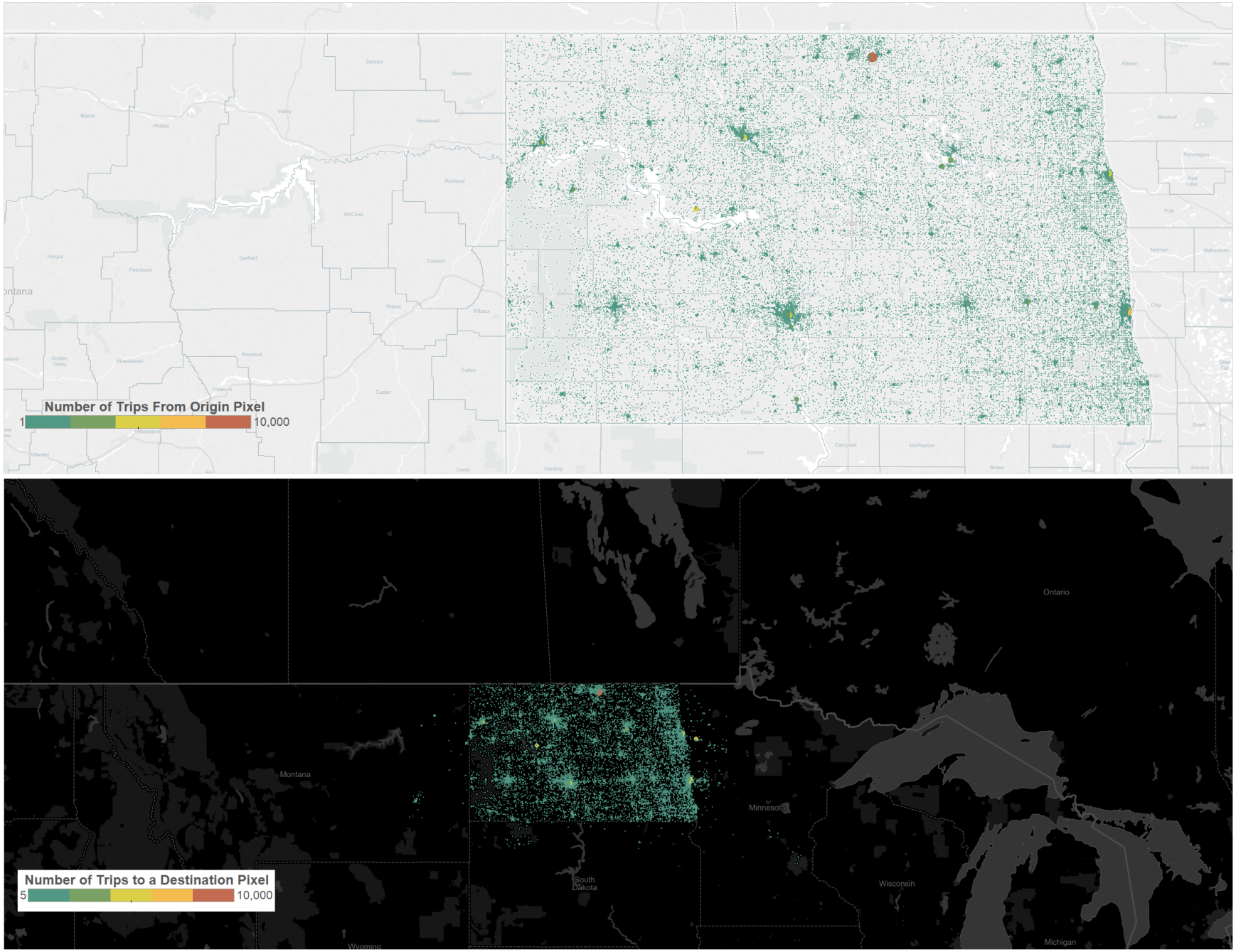

Figure A.97: North_Dakota: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

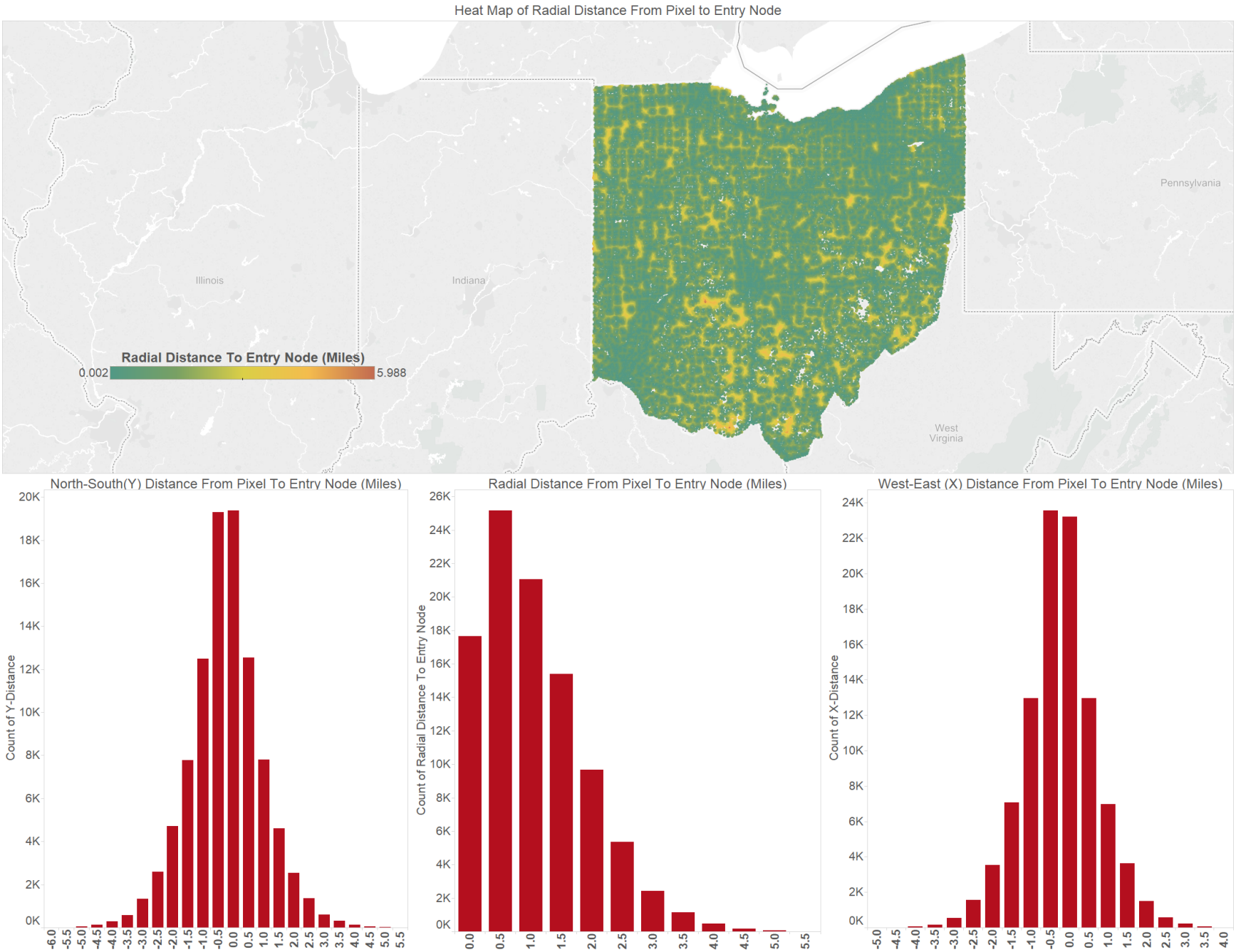

Figure A.98: Ohio: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

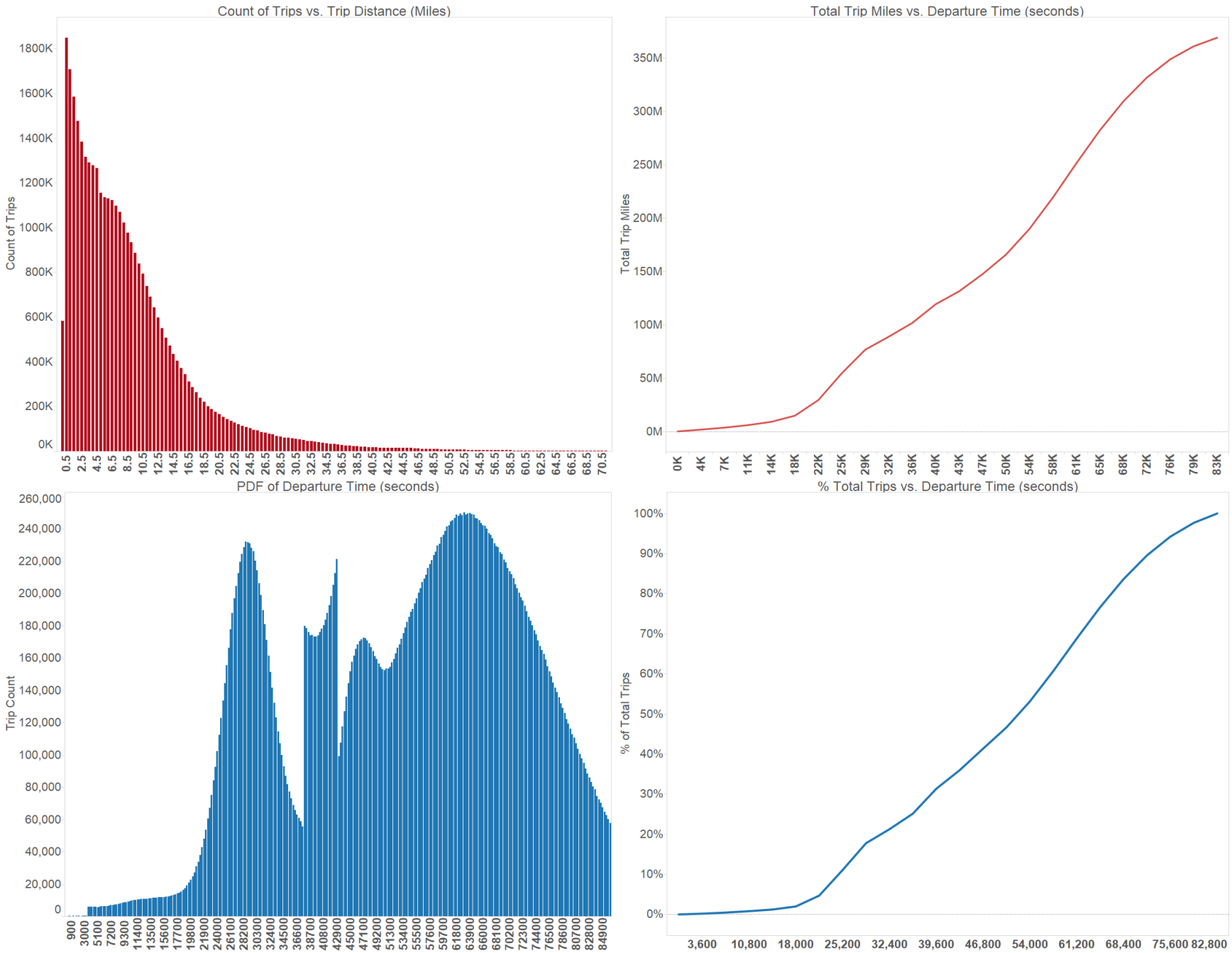

Figure A.99: Ohio: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

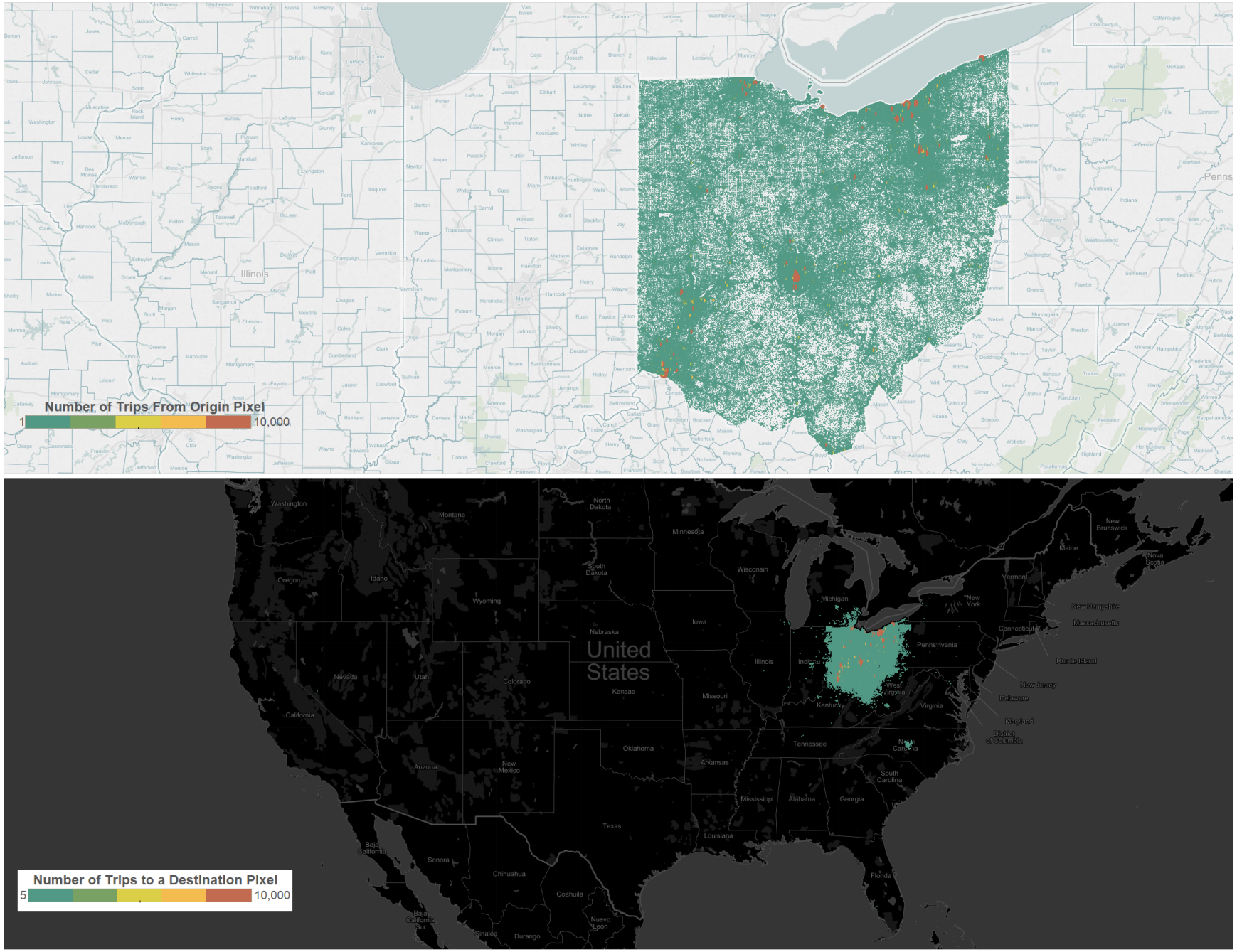

Figure A.100: Ohio: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

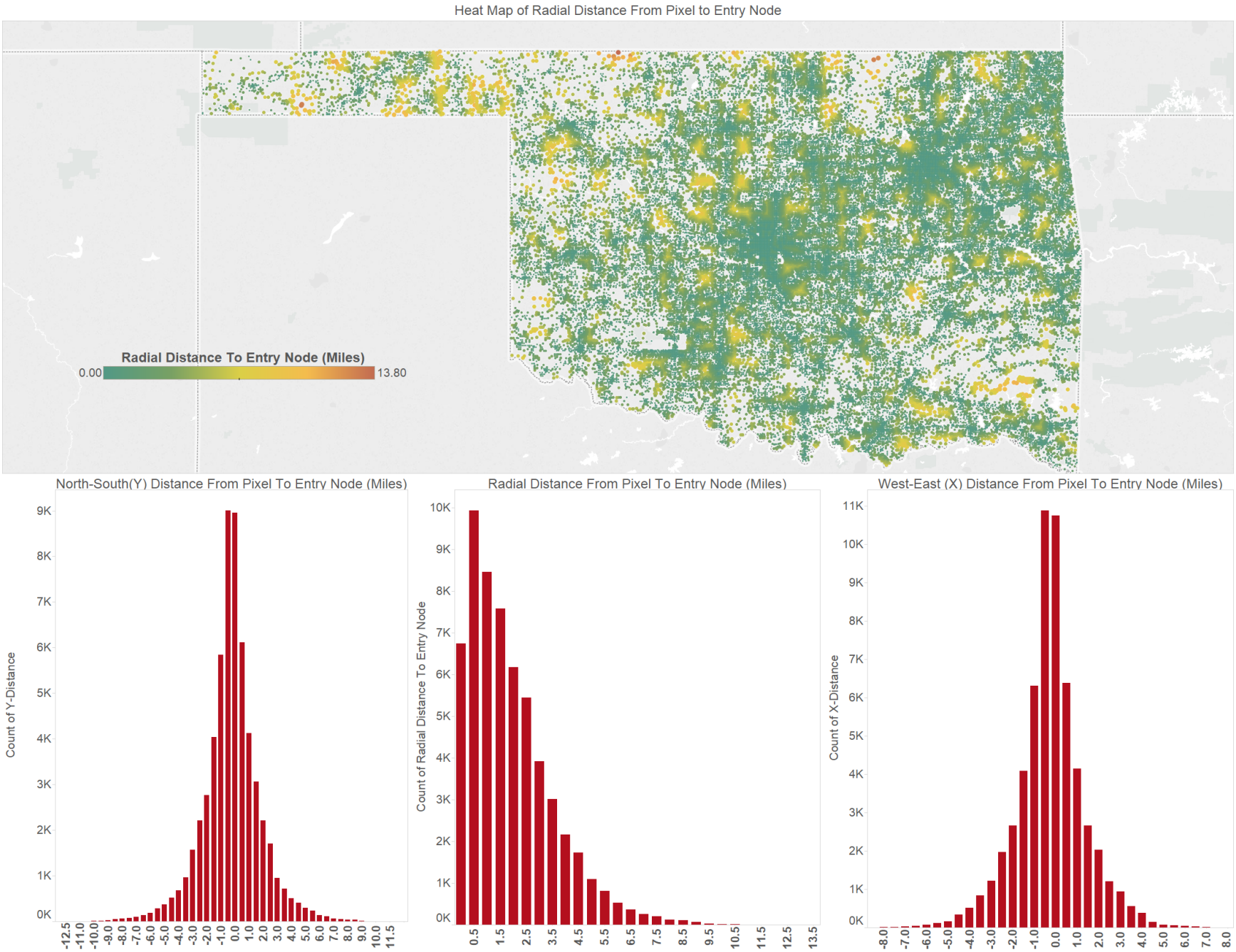

Figure A.101: Oklahoma: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

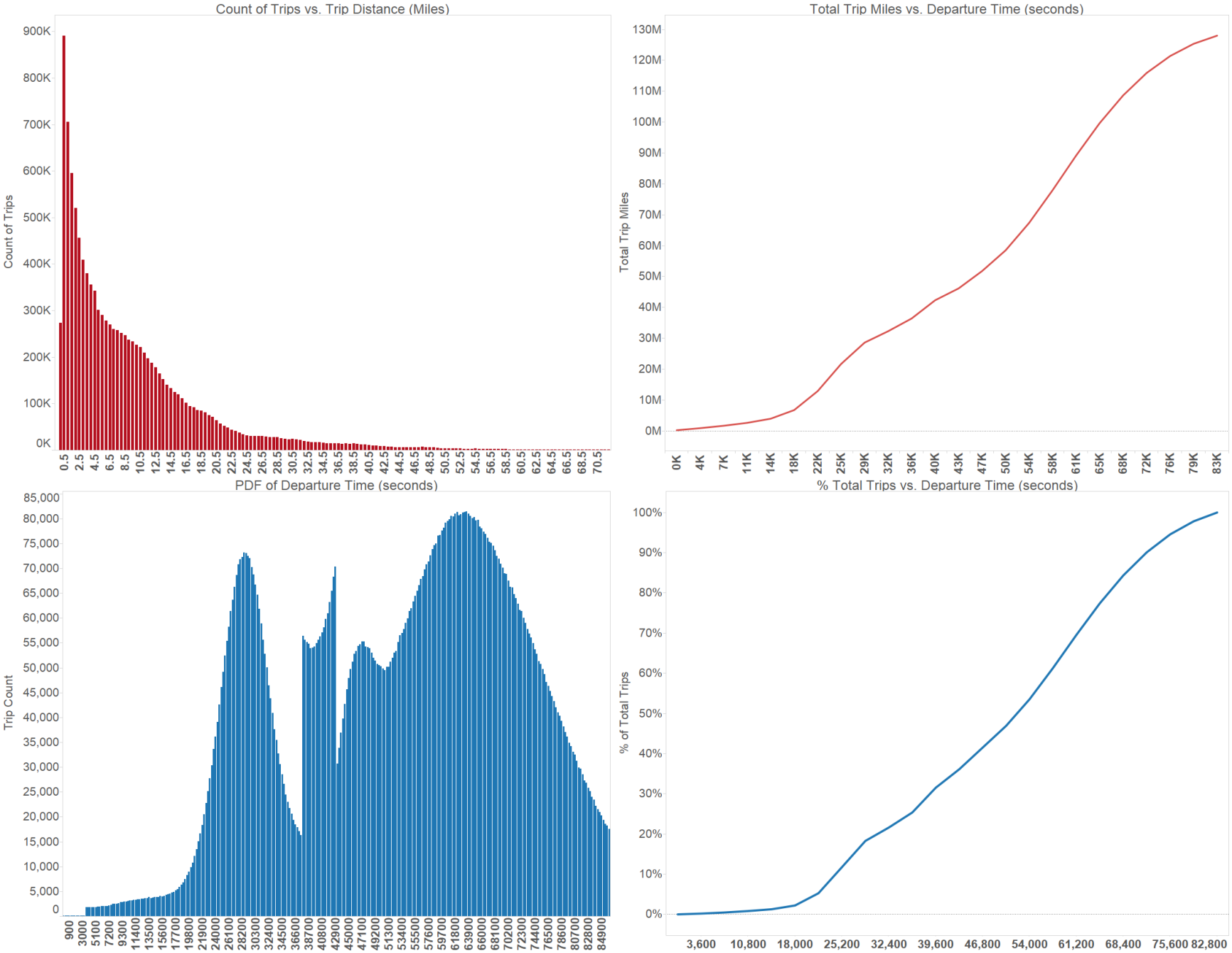

Figure A.102: Oklahoma: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

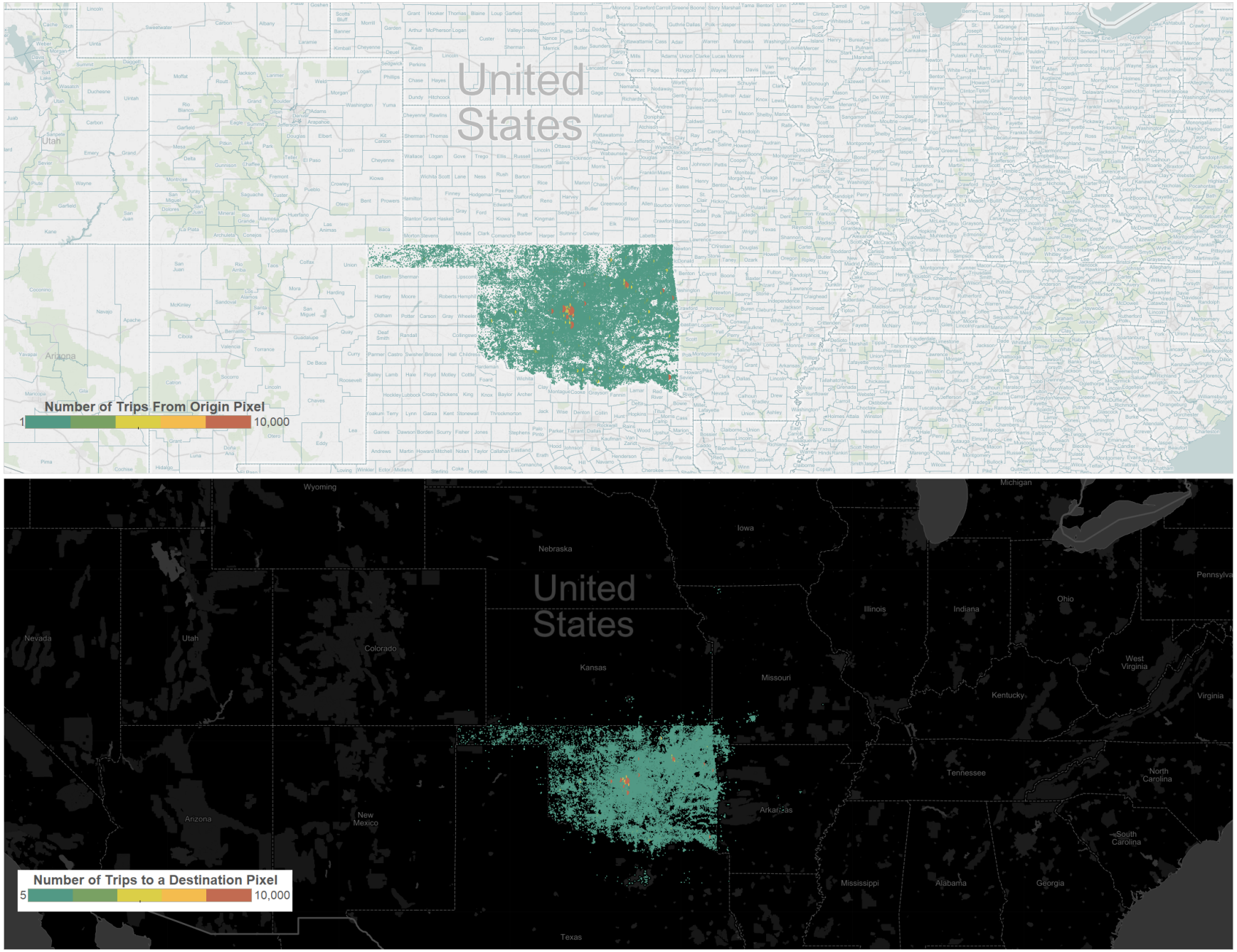

Figure A.103: Oklahoma: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

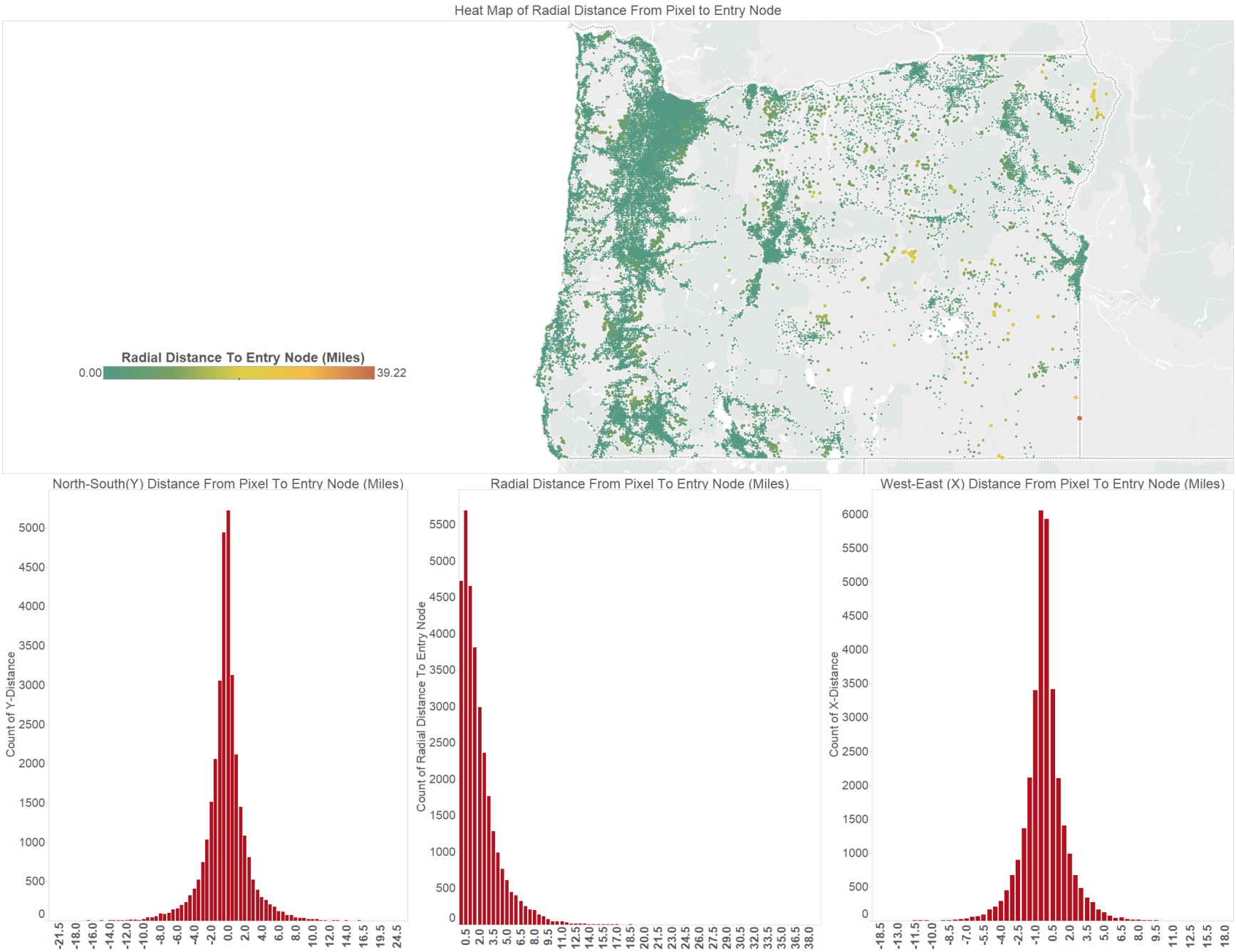

Figure A.104: Oregon: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

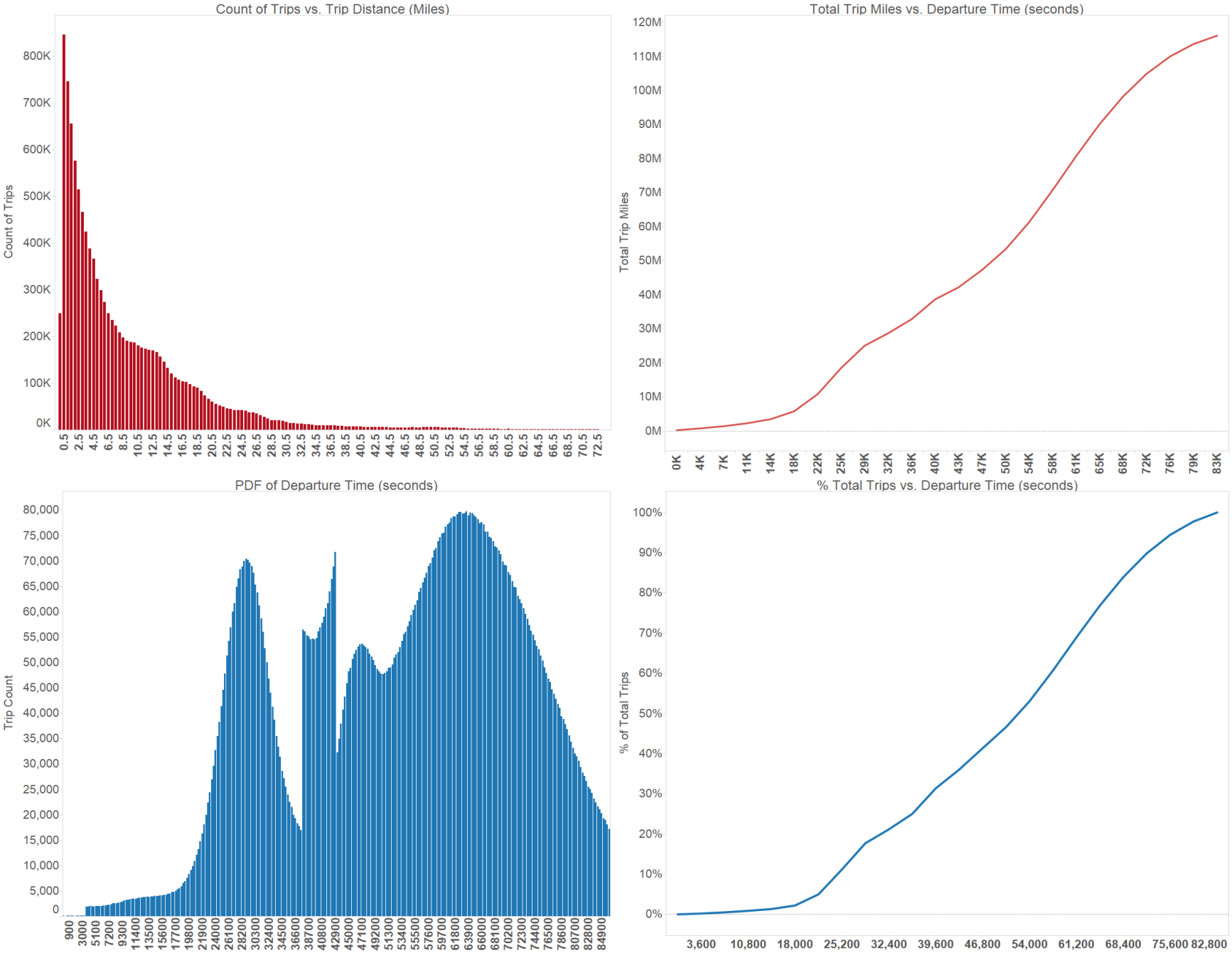

Figure A.105: Oregon: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

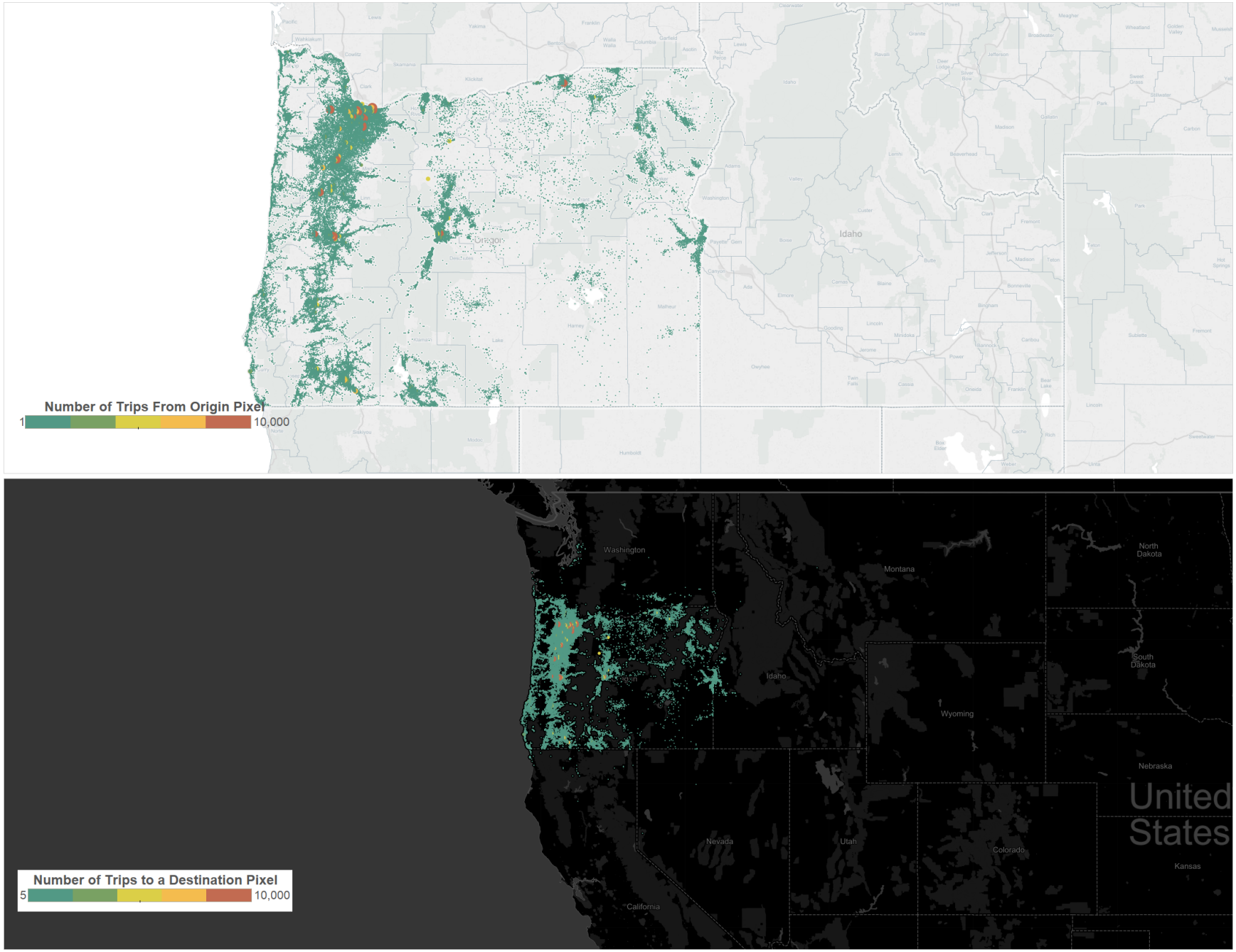

Figure A.106: Oregon: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

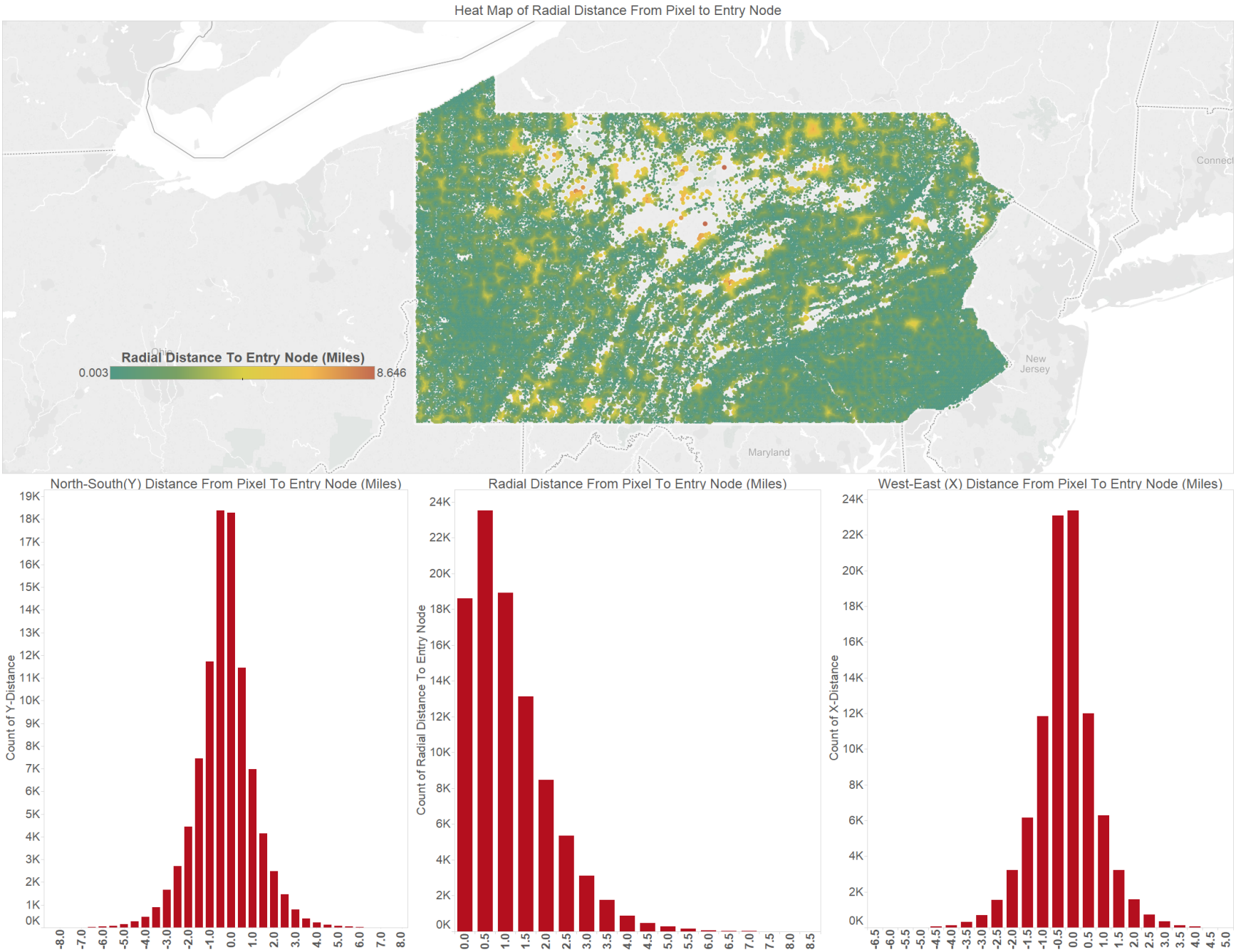

Figure A.107: Pennsylvania: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

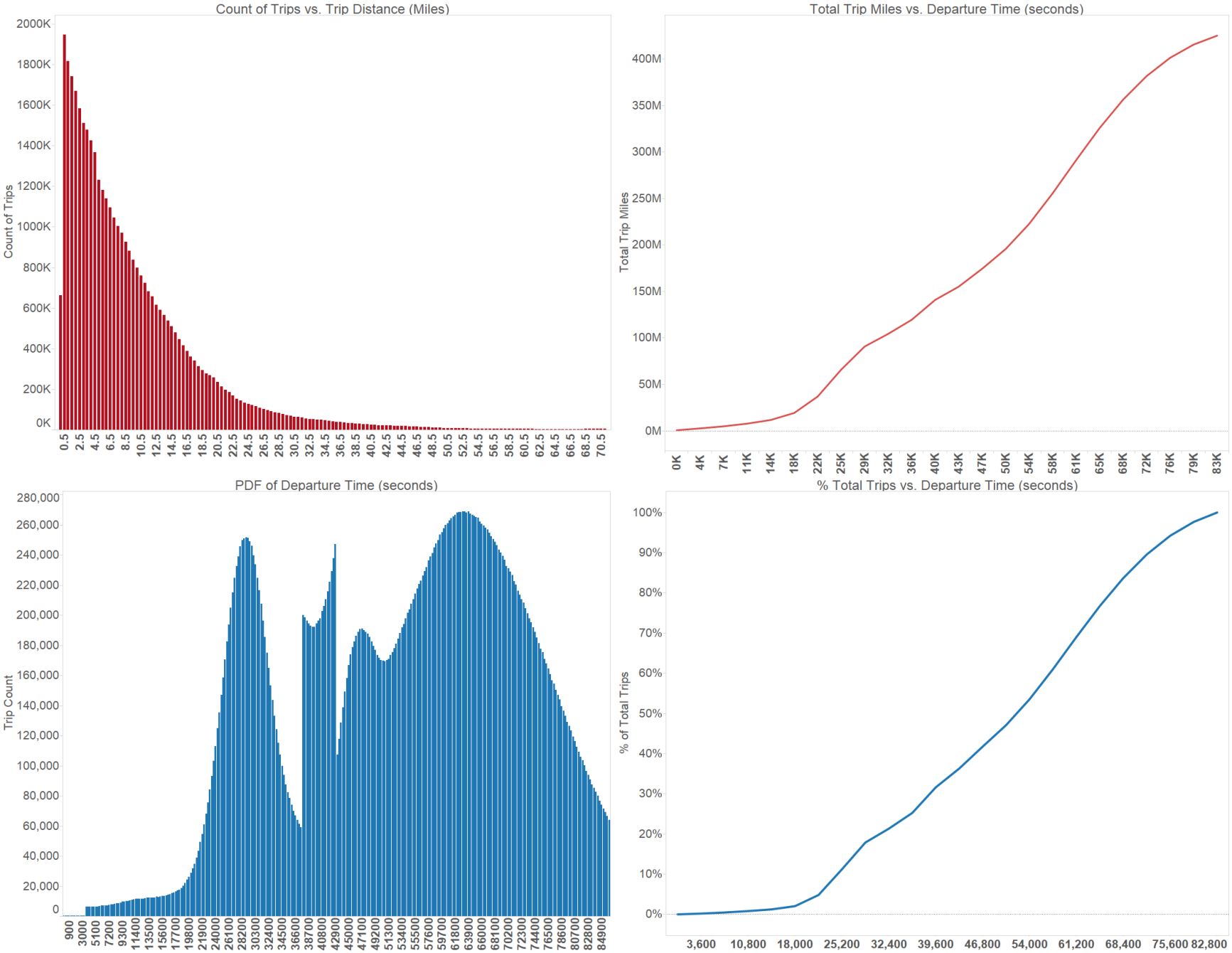

Figure A.108: Pennsylvania: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

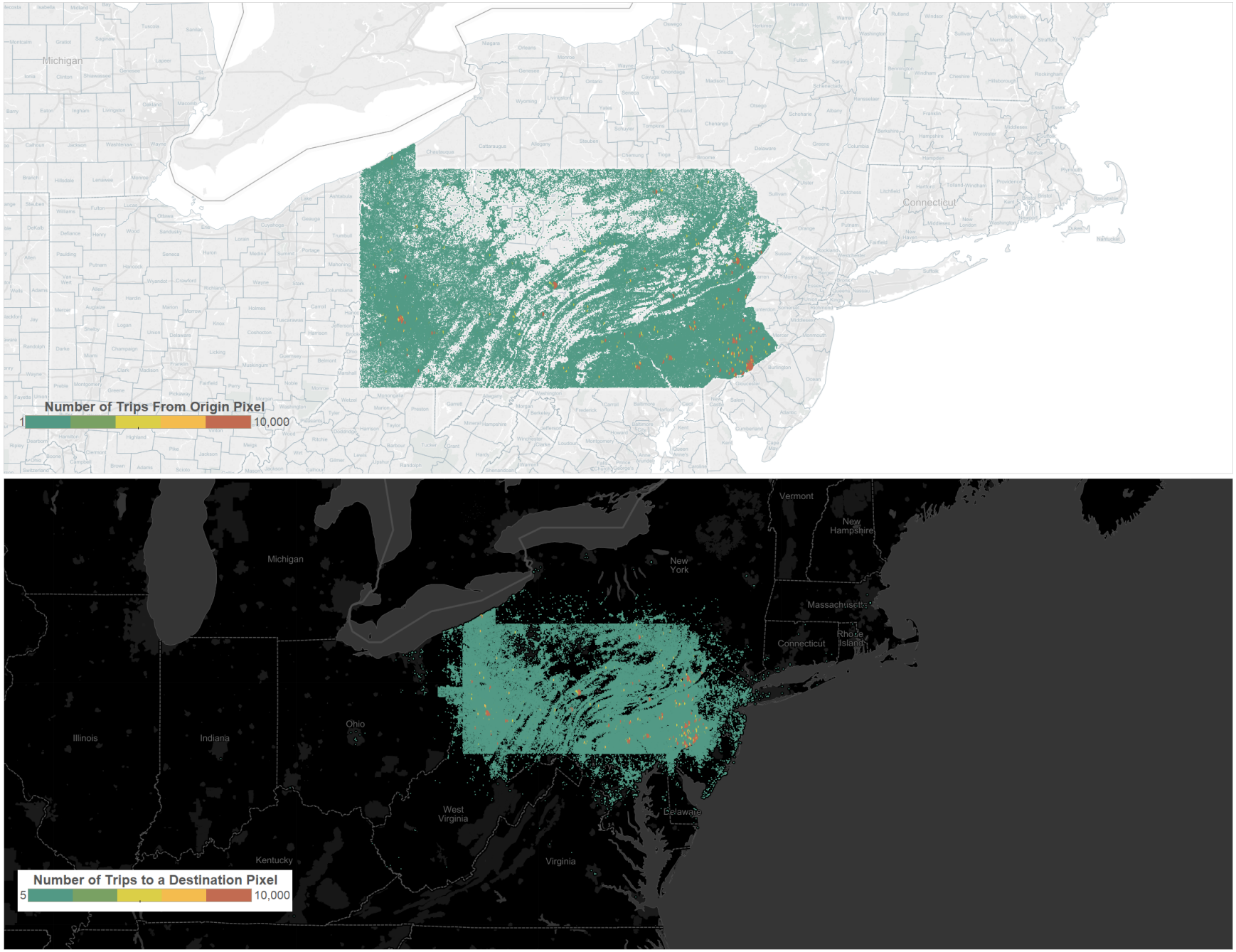

Figure A.109: Pennsylvania: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

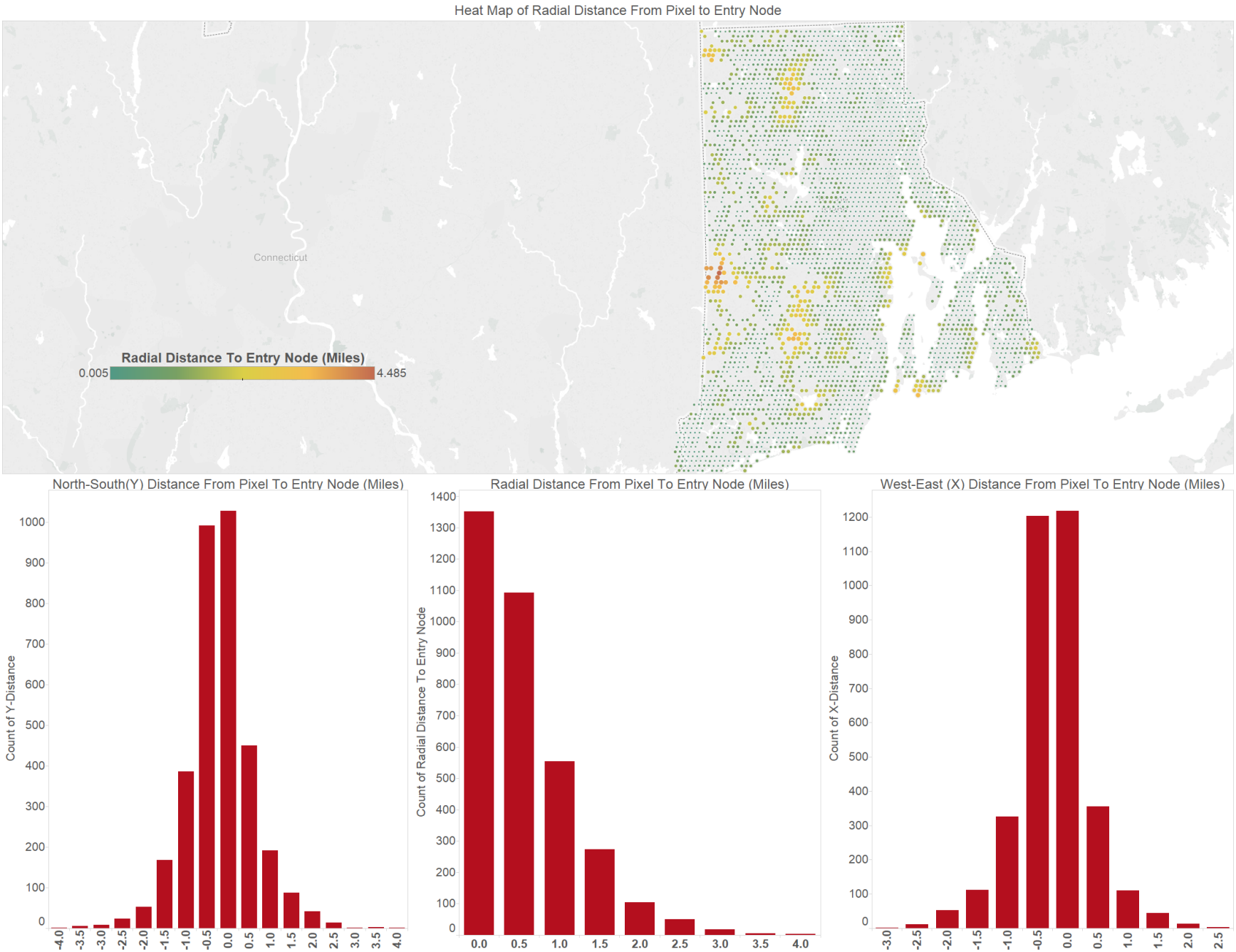

Figure A.110: Rhode_Island: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

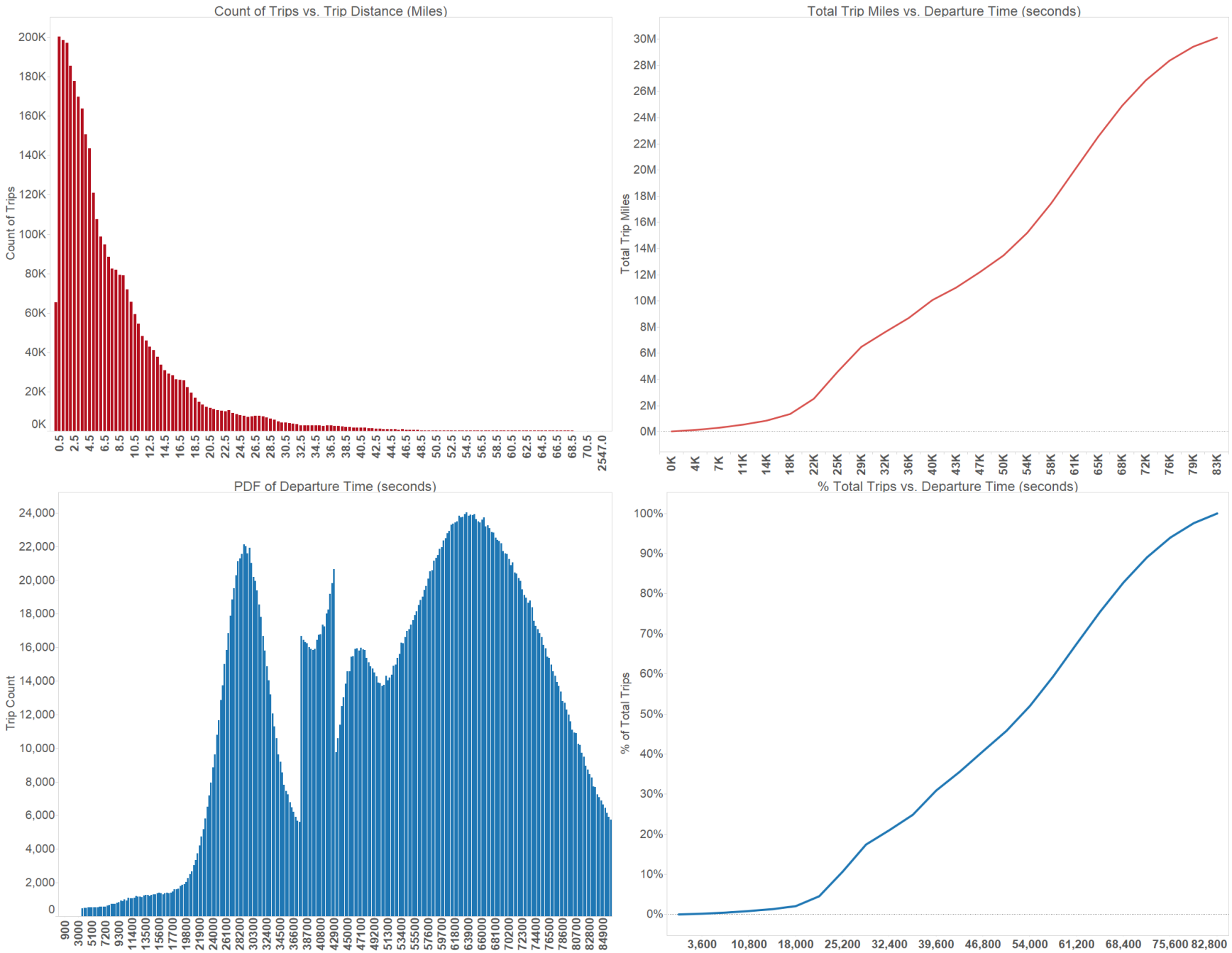

Figure A.111: Rhode_Island: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

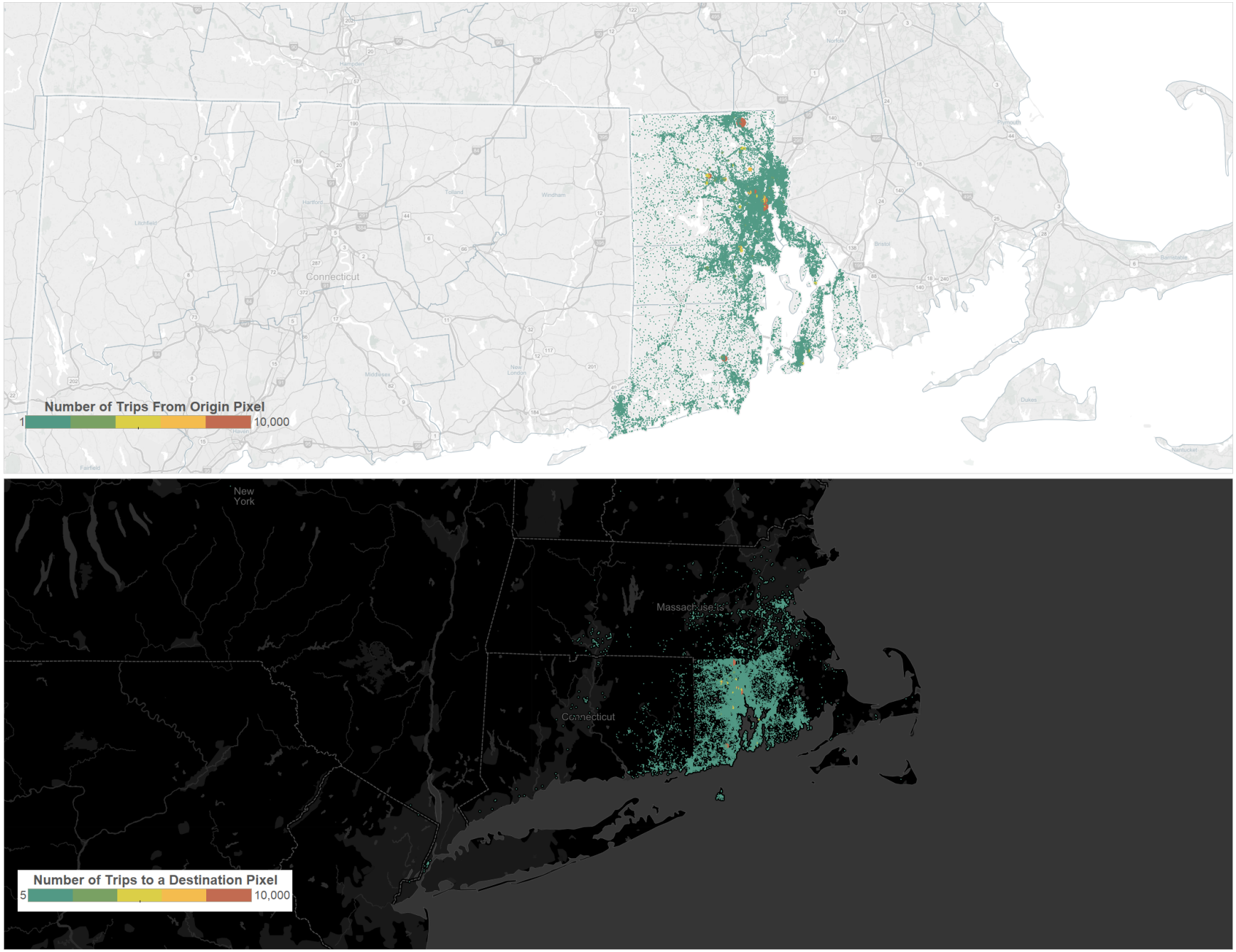

Figure A.112: Rhode_Island: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

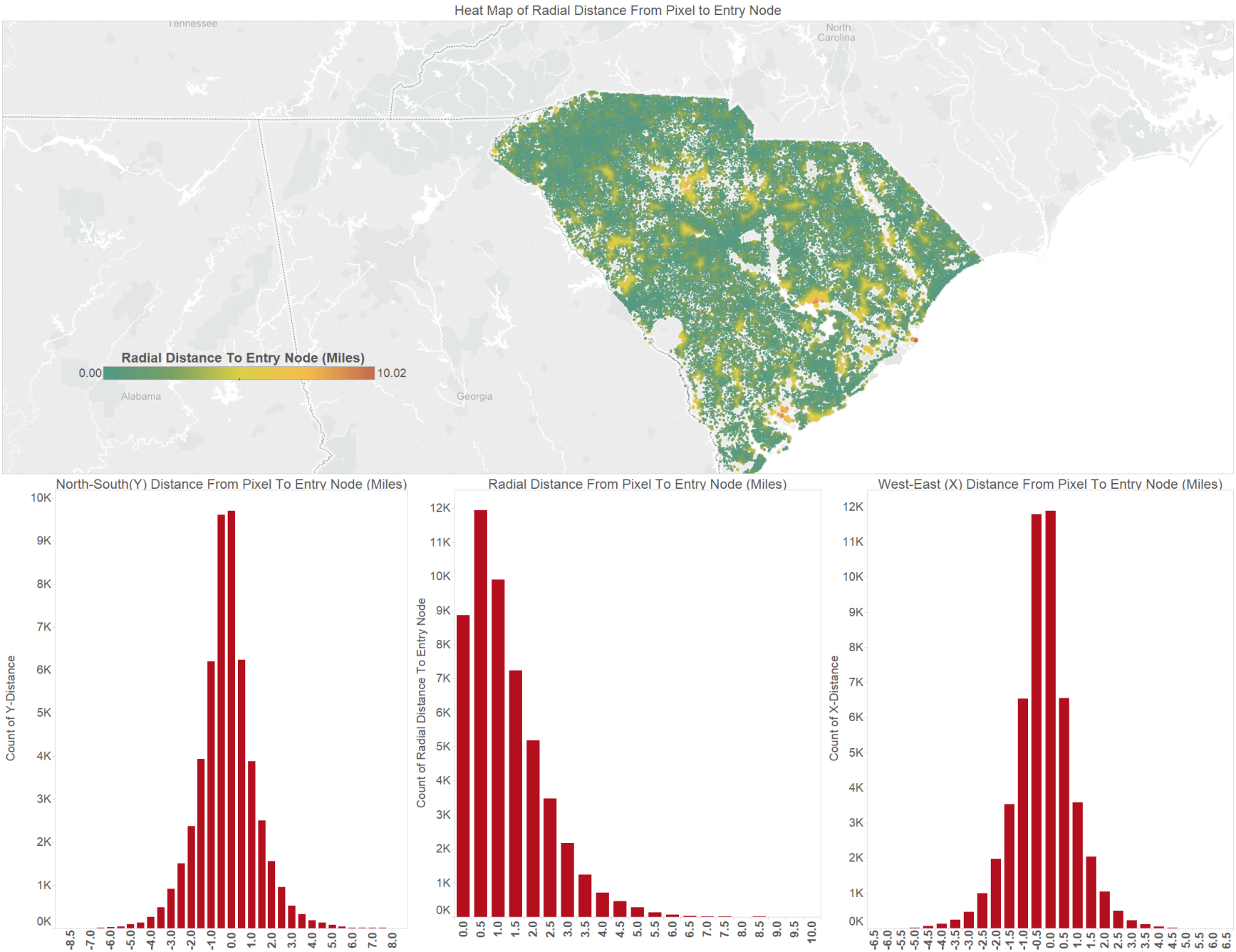

Figure A.113: South_Carolina: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

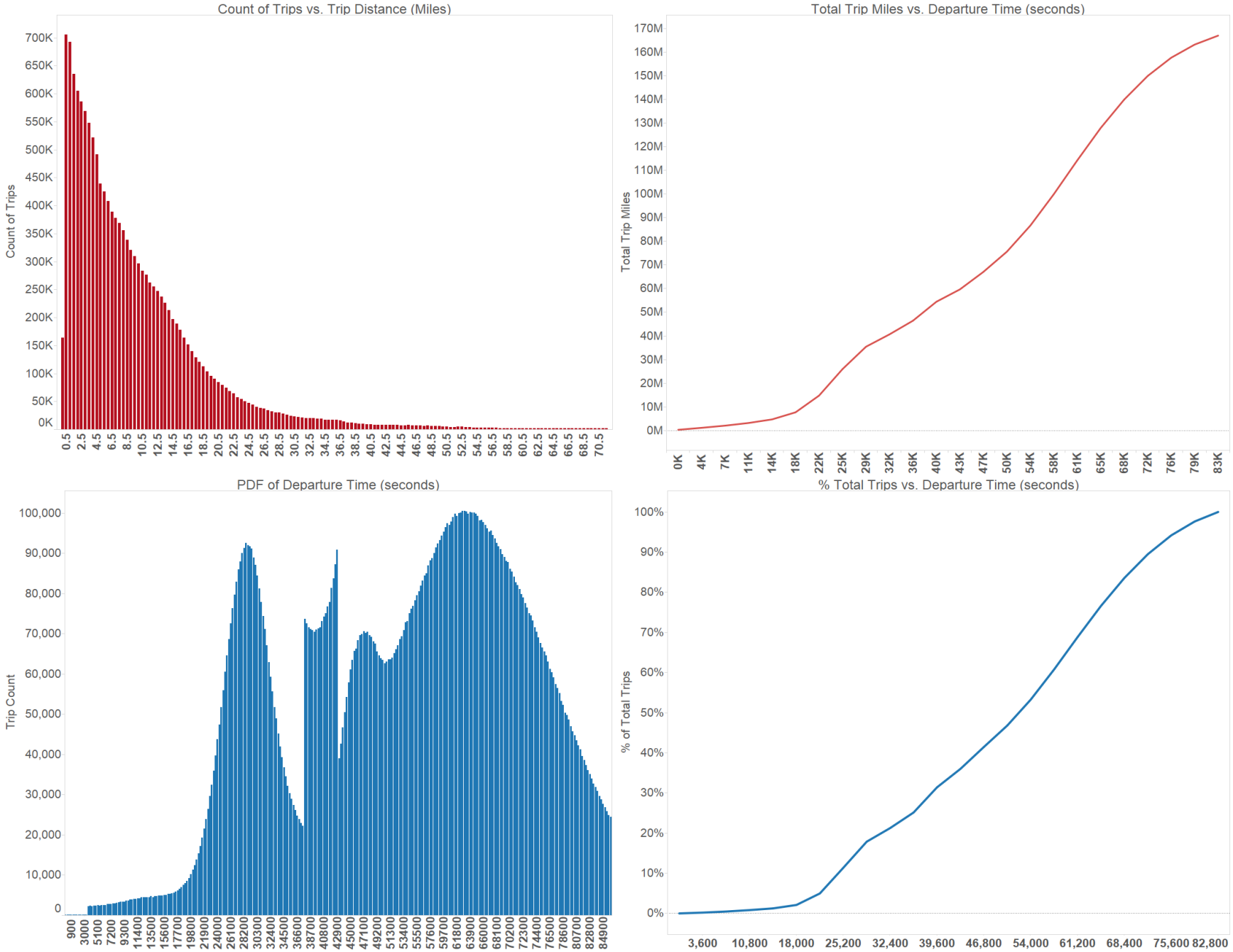

Figure A.114: South_Carolina: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

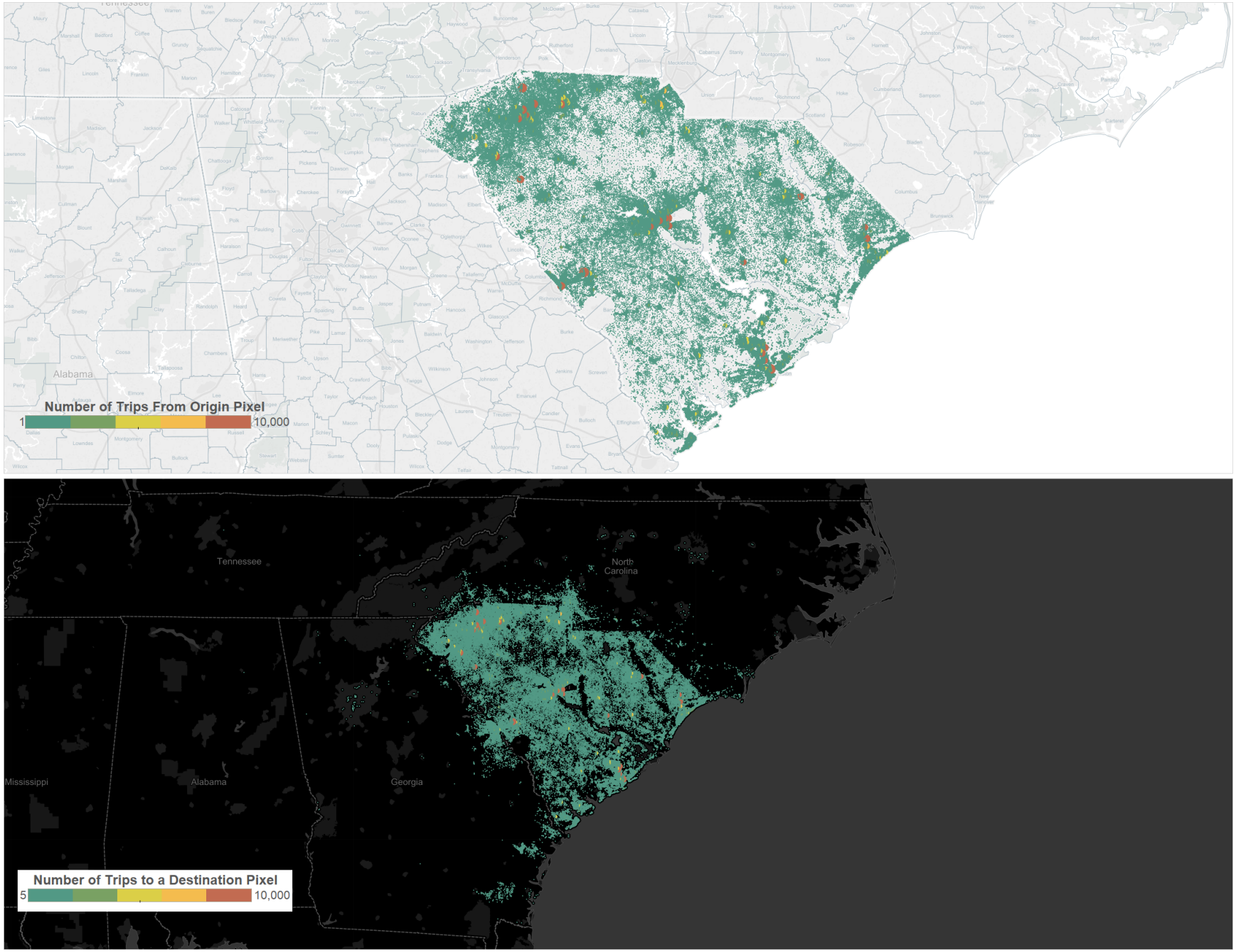

Figure A.115: South_Carolina: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

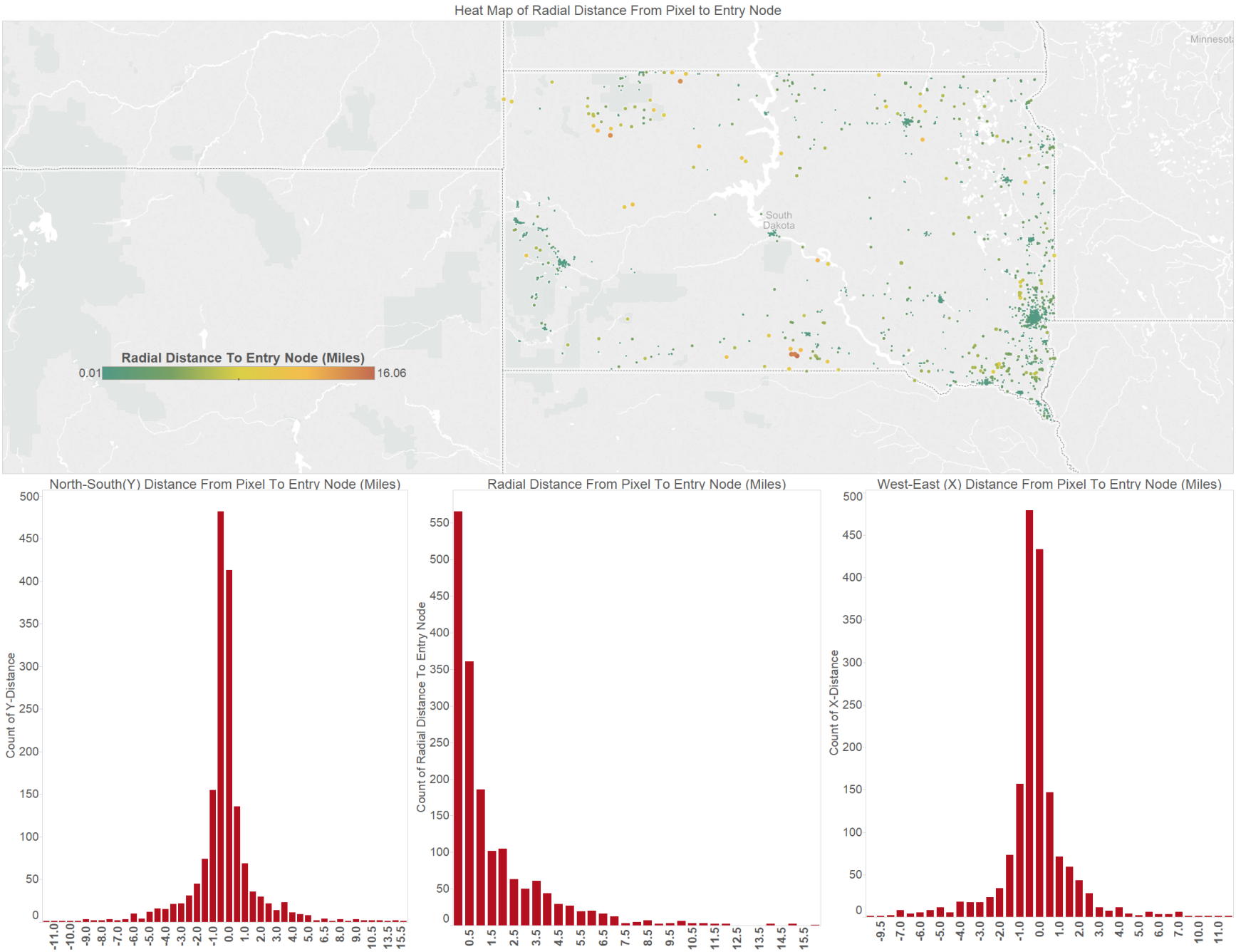

Figure A.116: South_Dakota: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

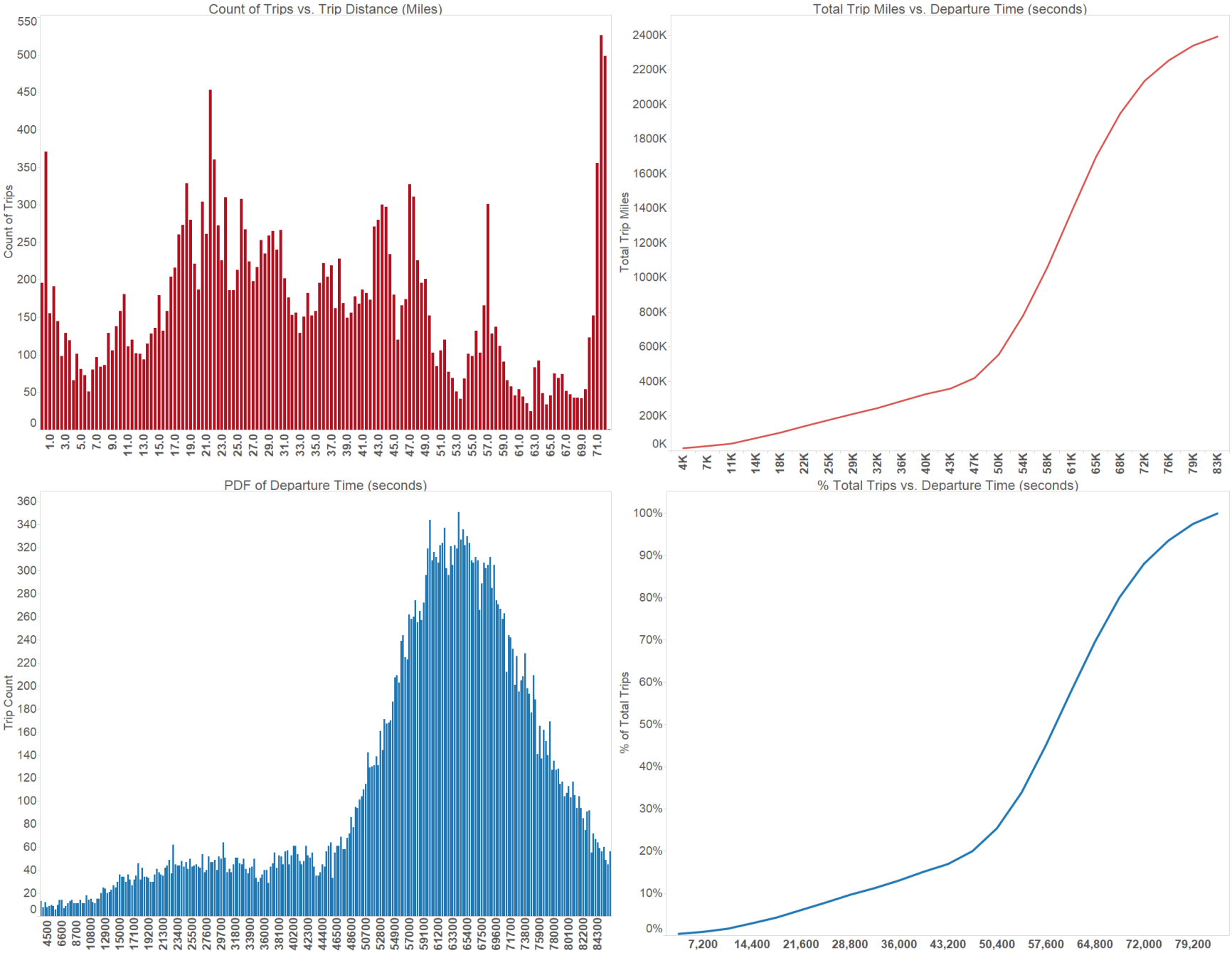

Figure A.117: South_Dakota: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

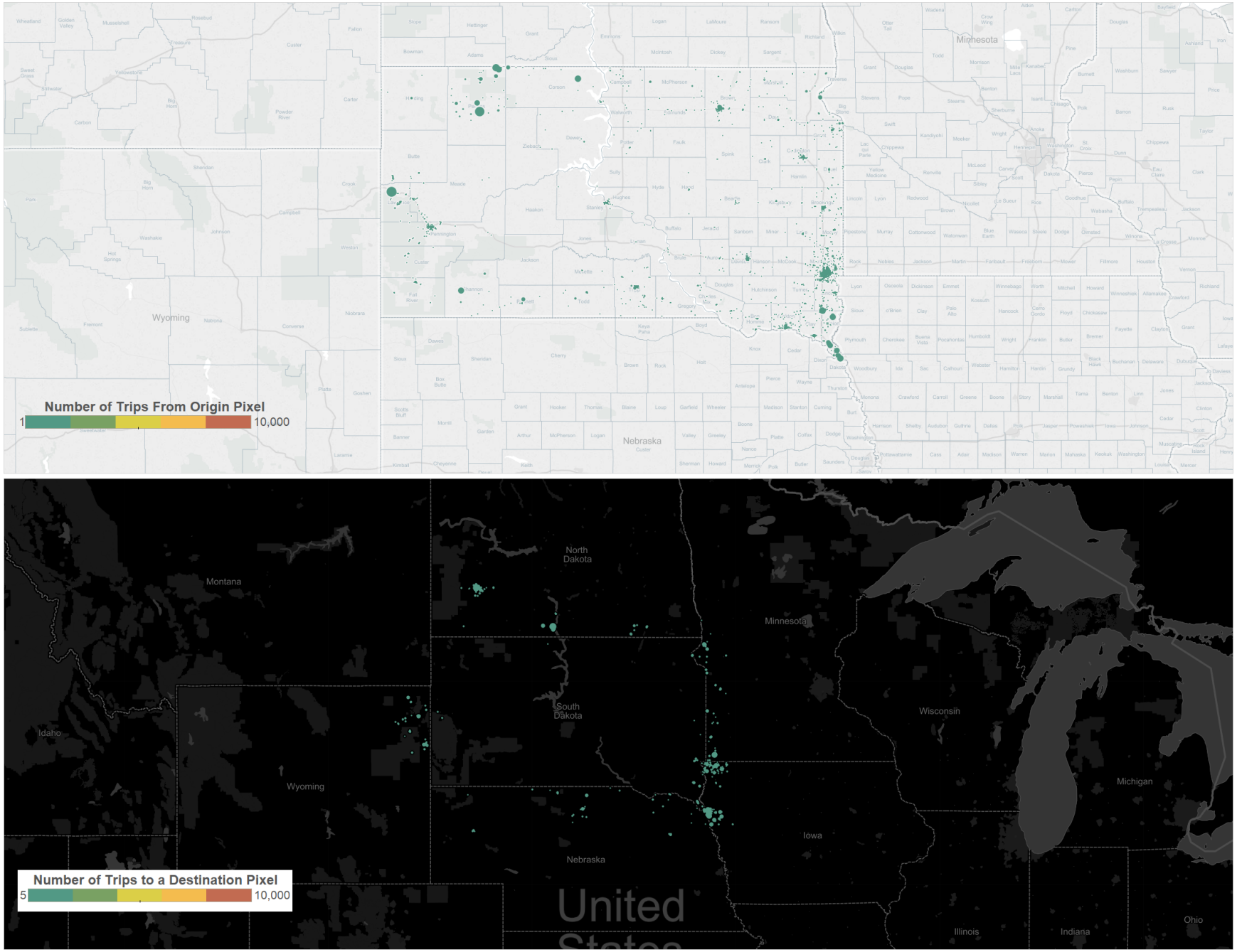

Figure A.118: South_Dakota: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

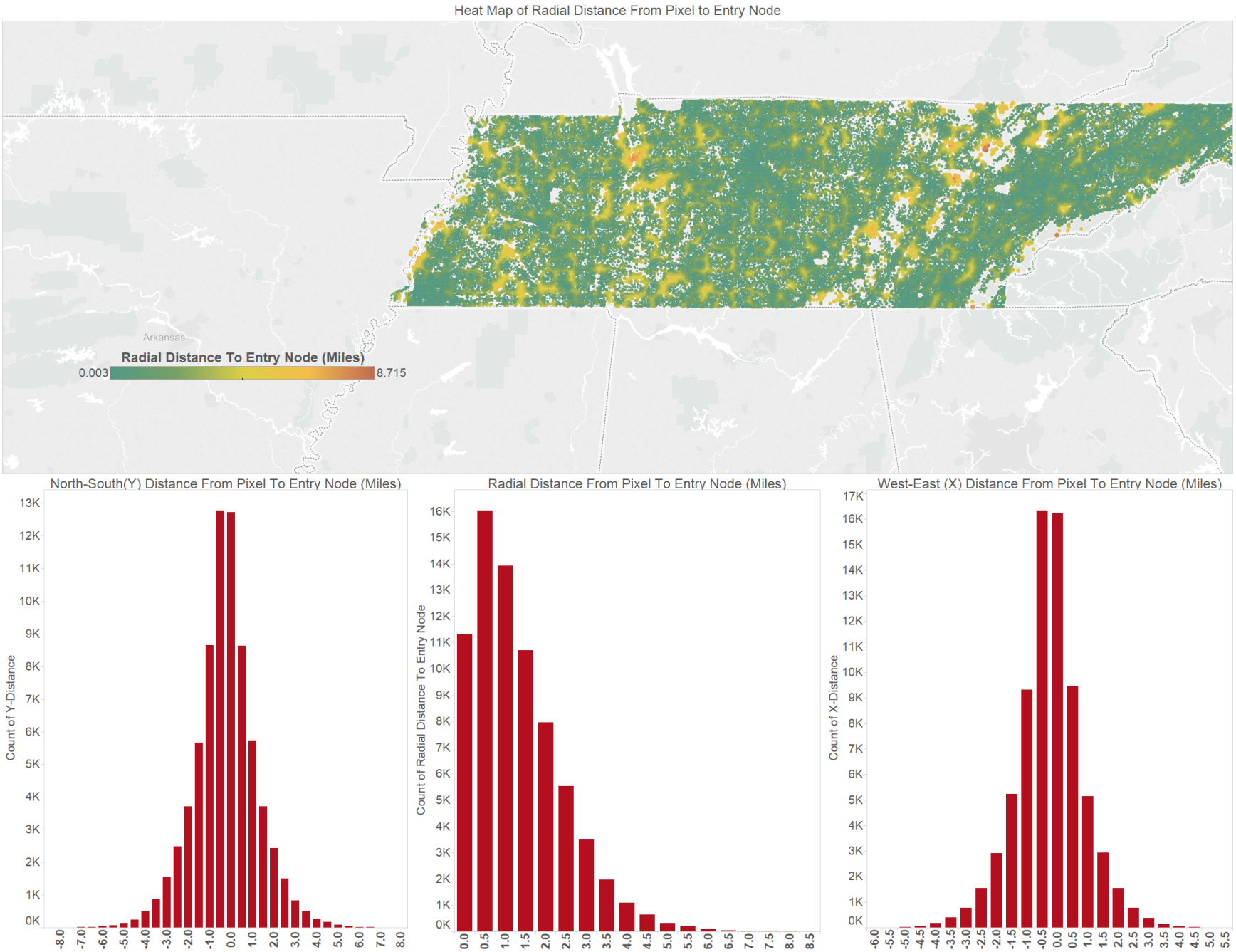

Figure A.119: Tennessee: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

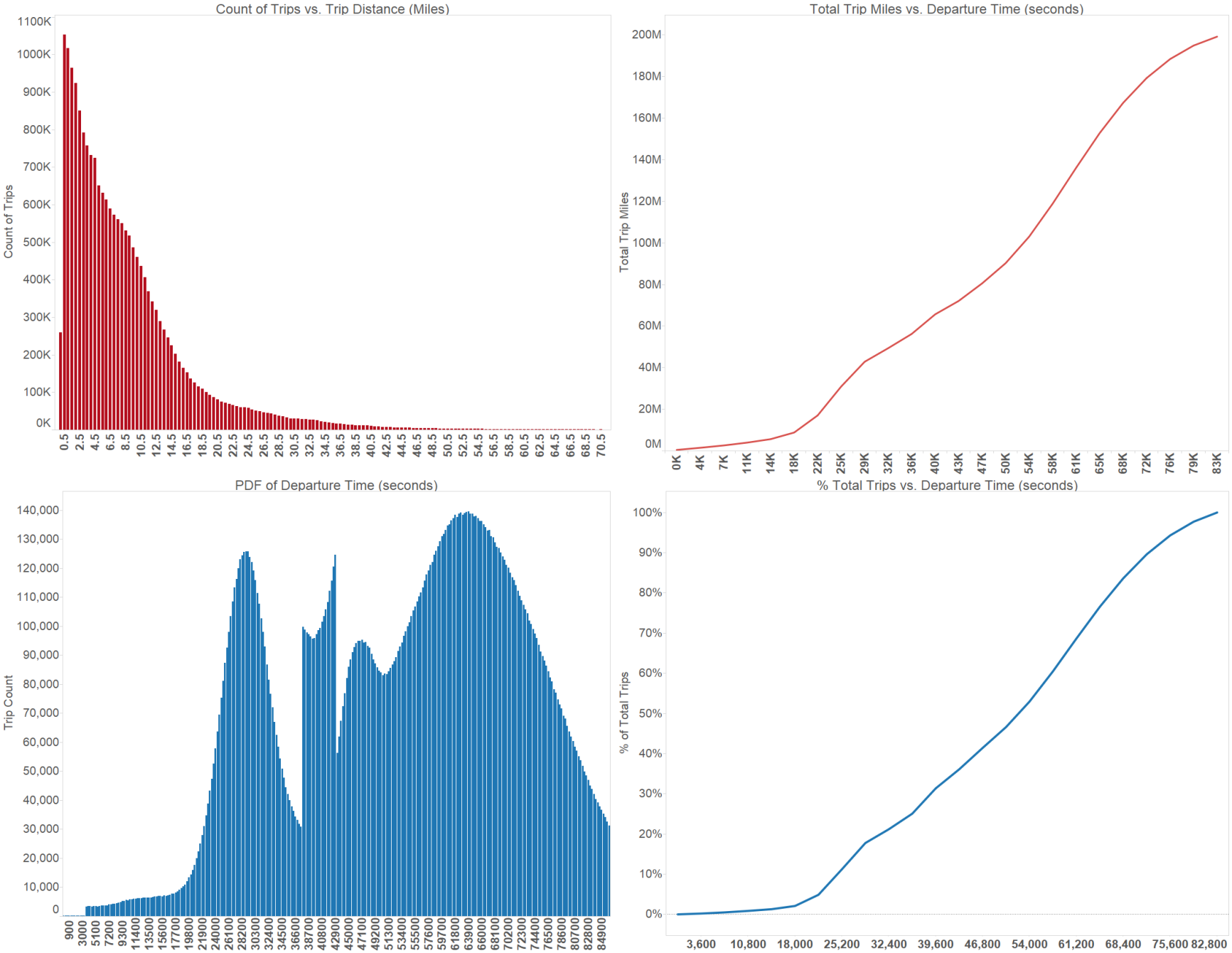

Figure A.120: Tennessee: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

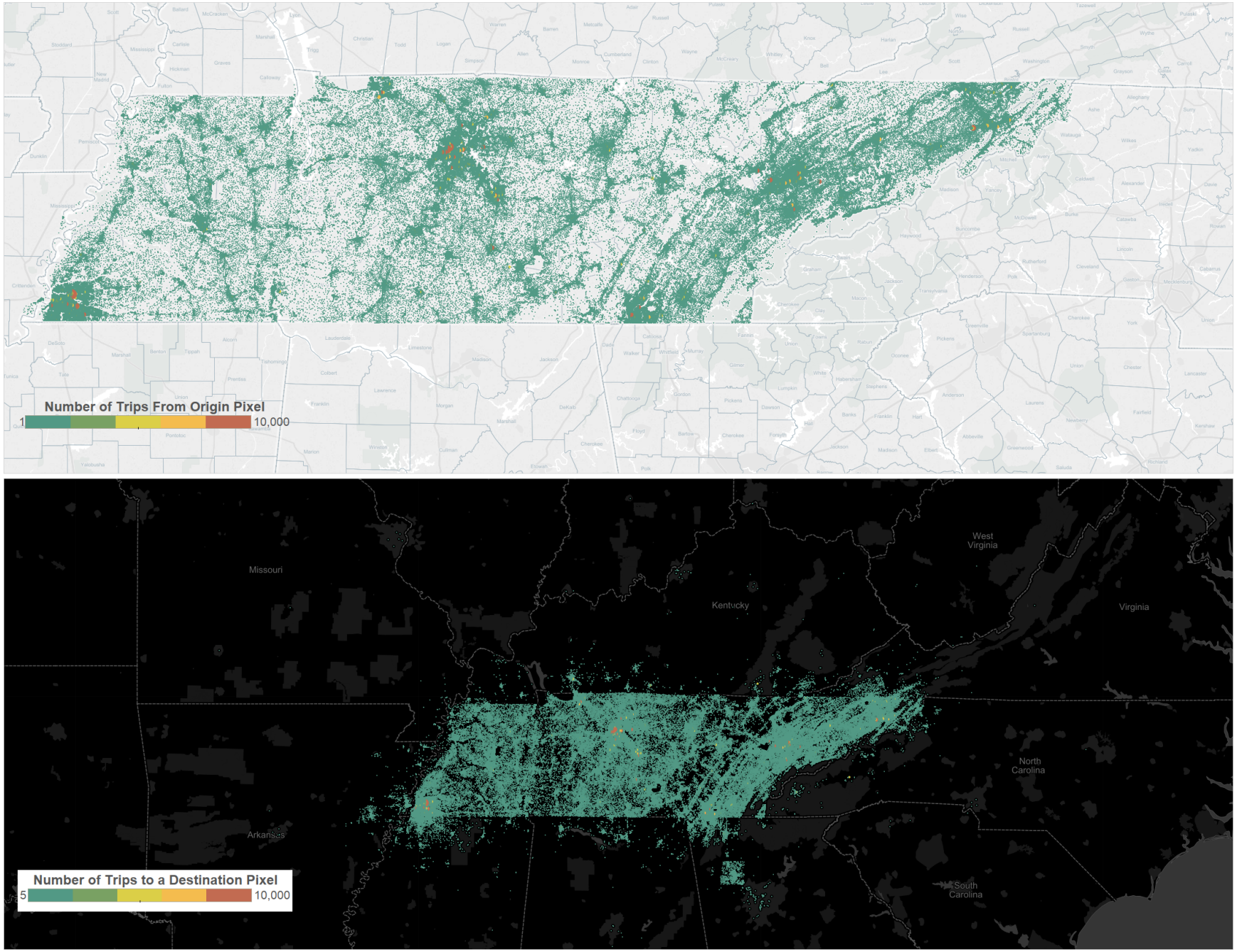

Figure A.121: Tennessee: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

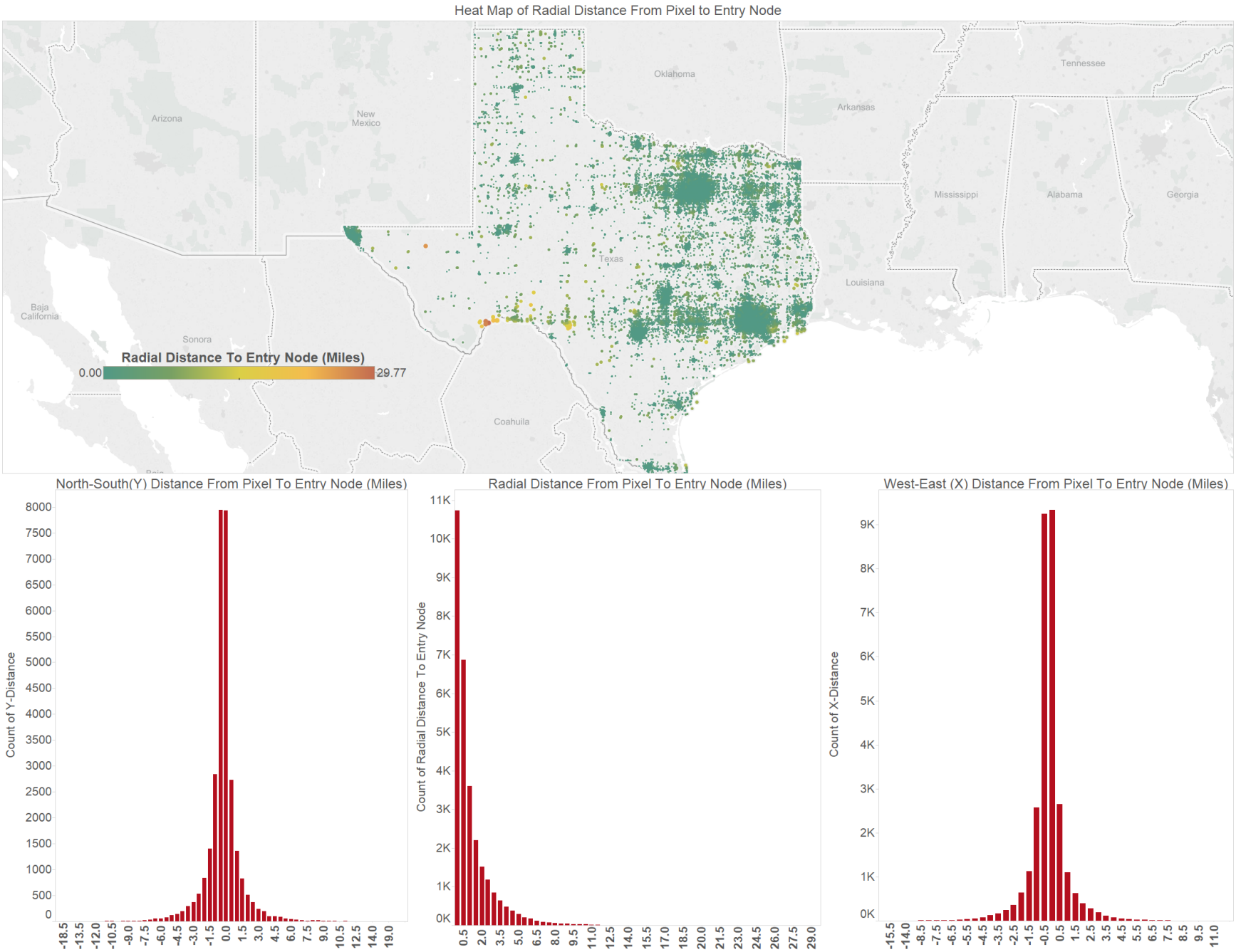

Figure A.122: Texas: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

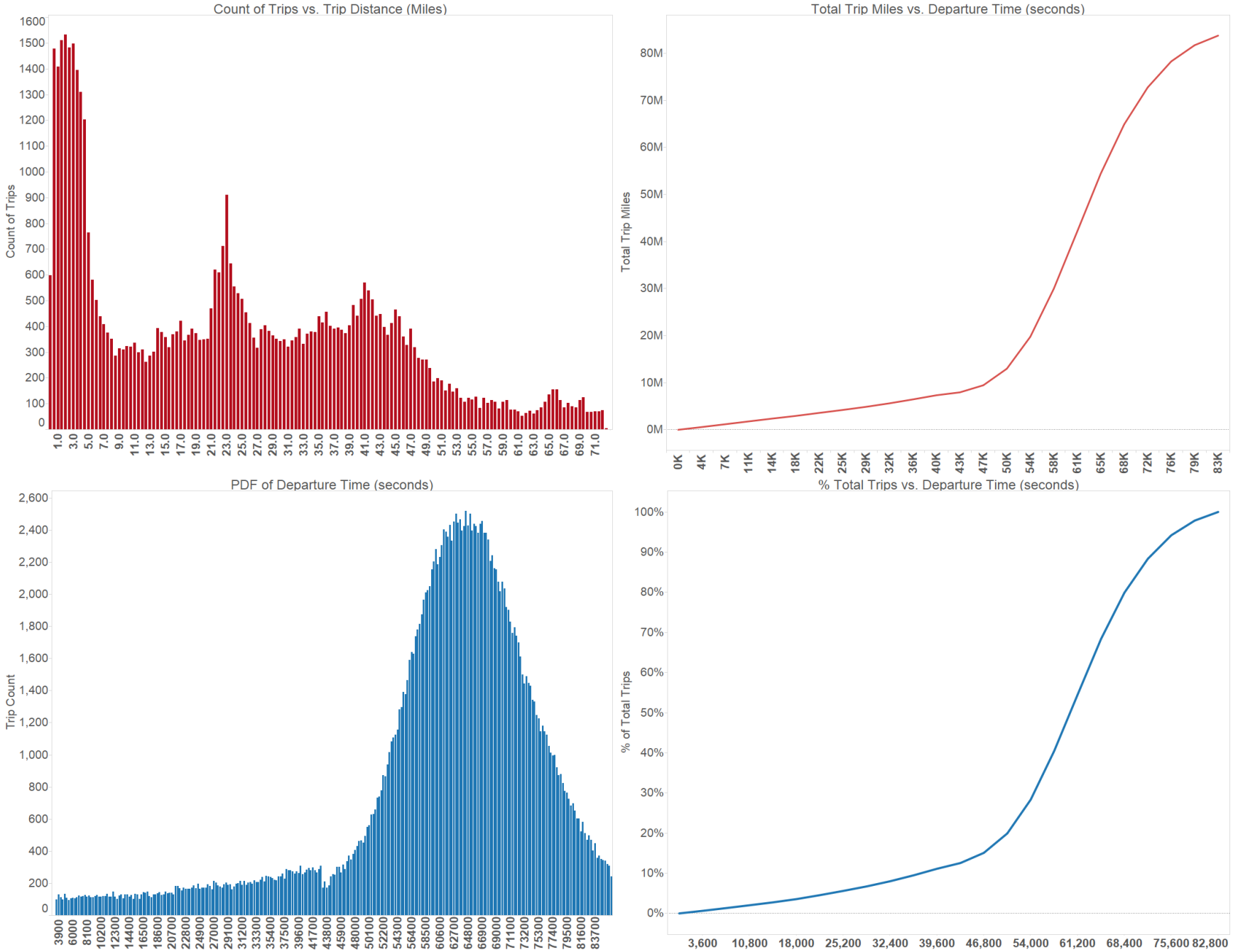

Figure A.123: Texas: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

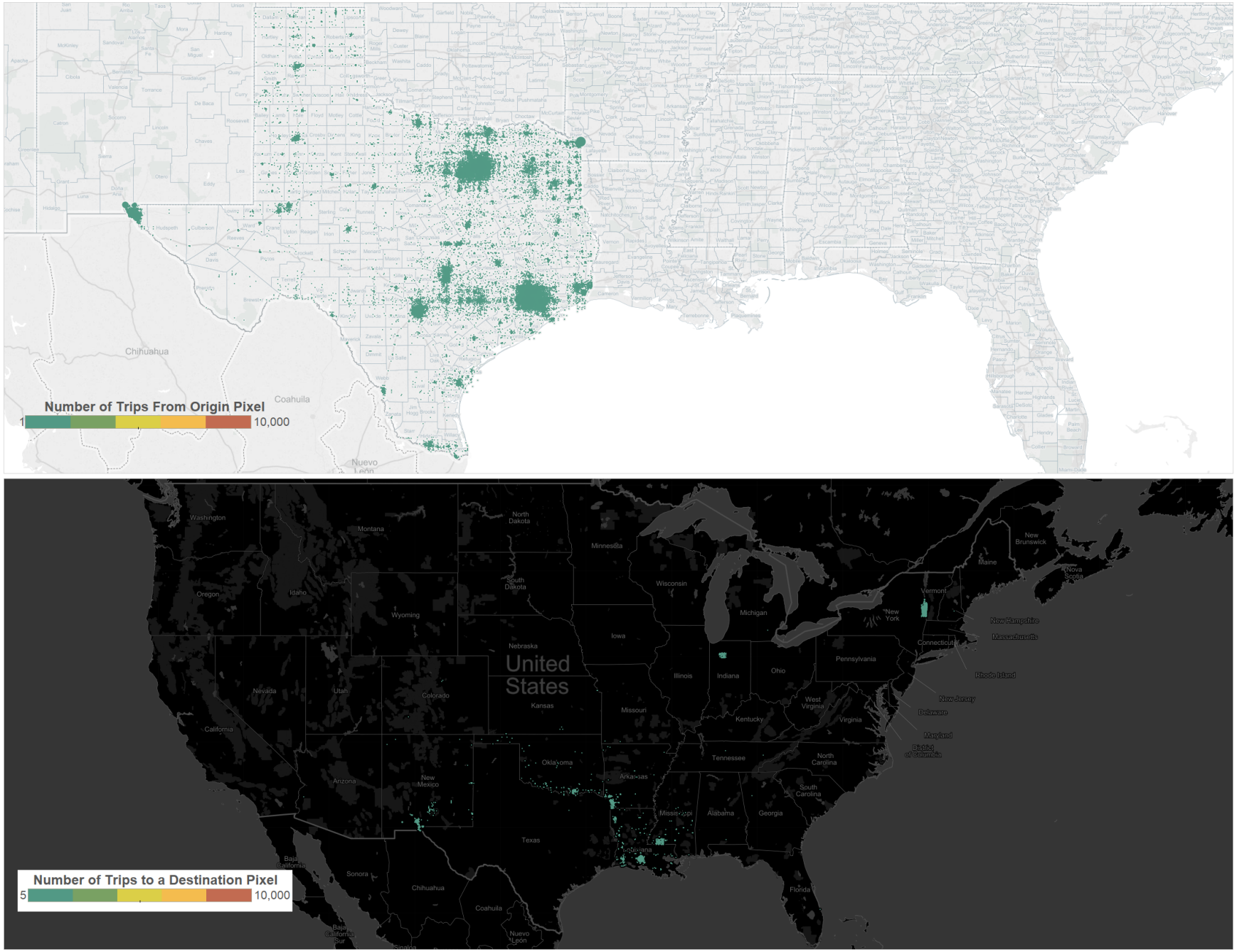

Figure A.124: Texas: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

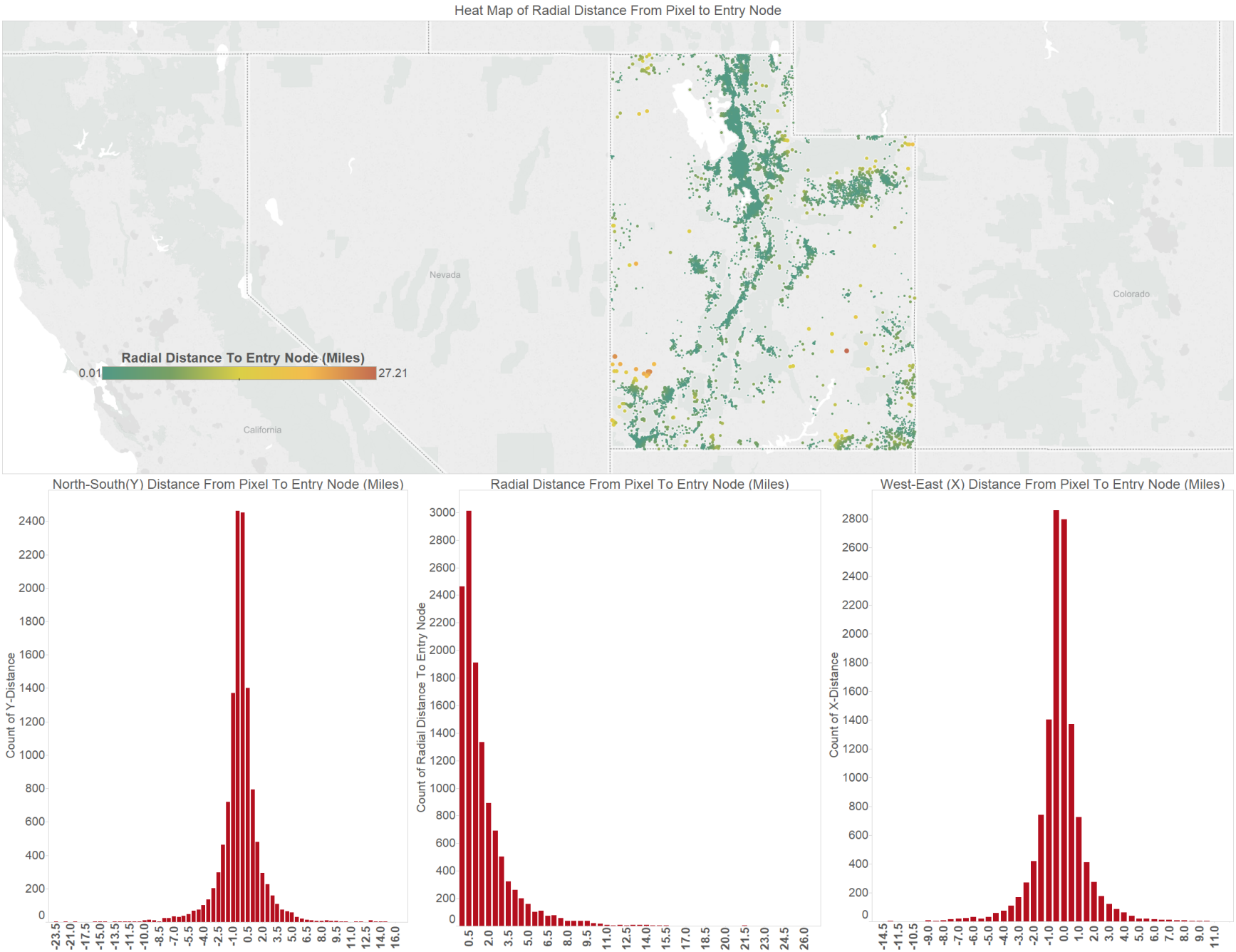

Figure A.125: Utah: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

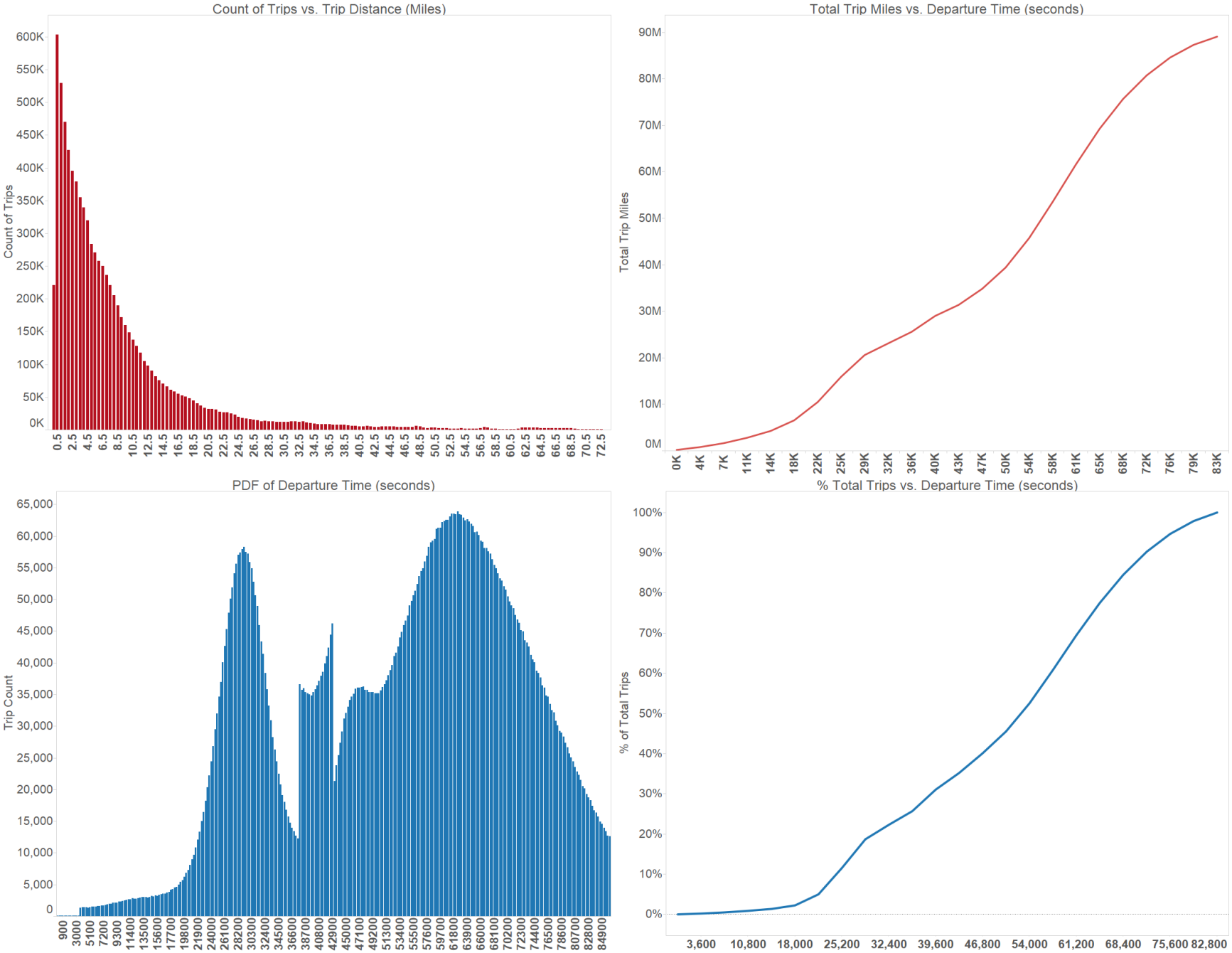

Figure A.126: Utah: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

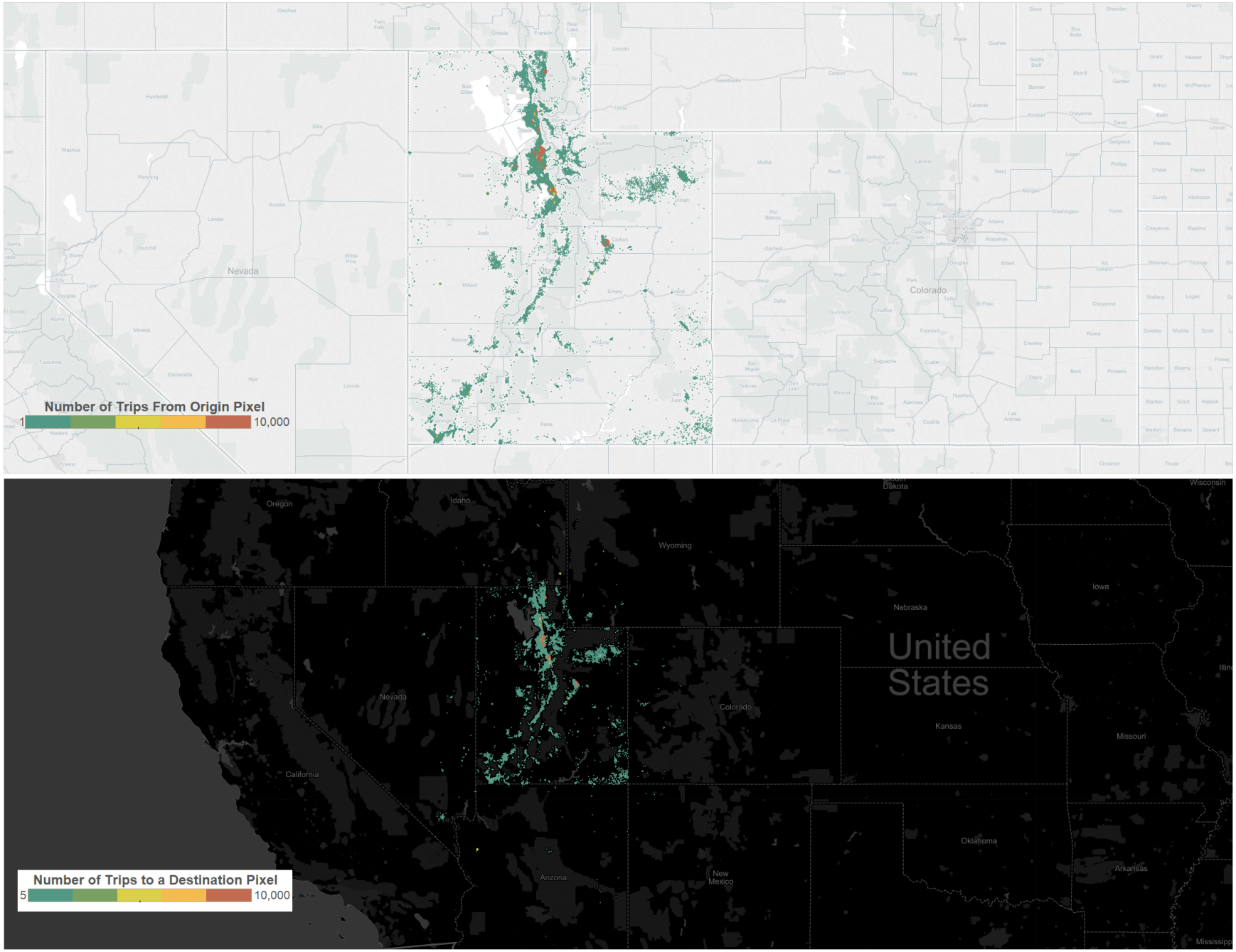

Figure A.127: Utah: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

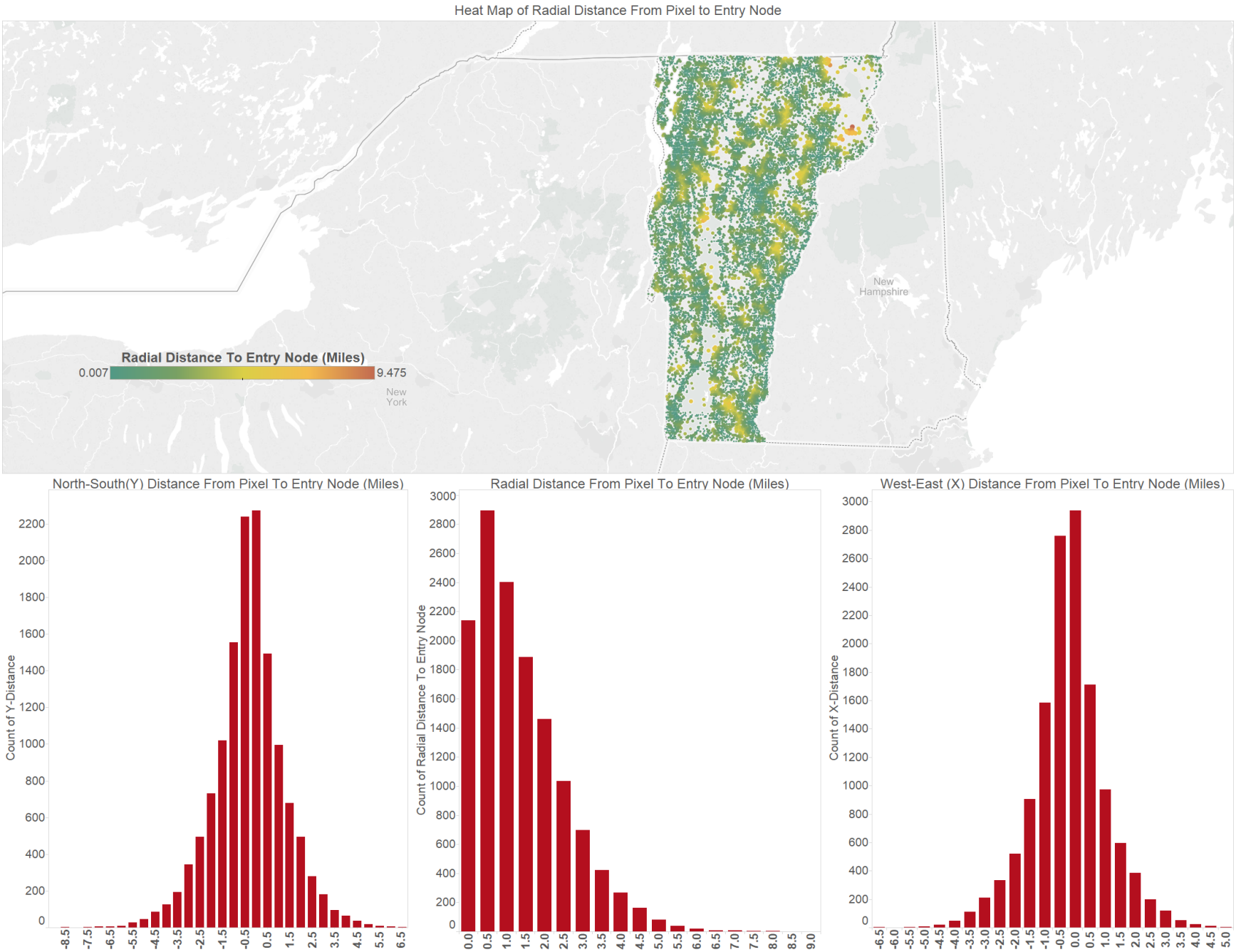

Figure A.128: Vermont: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

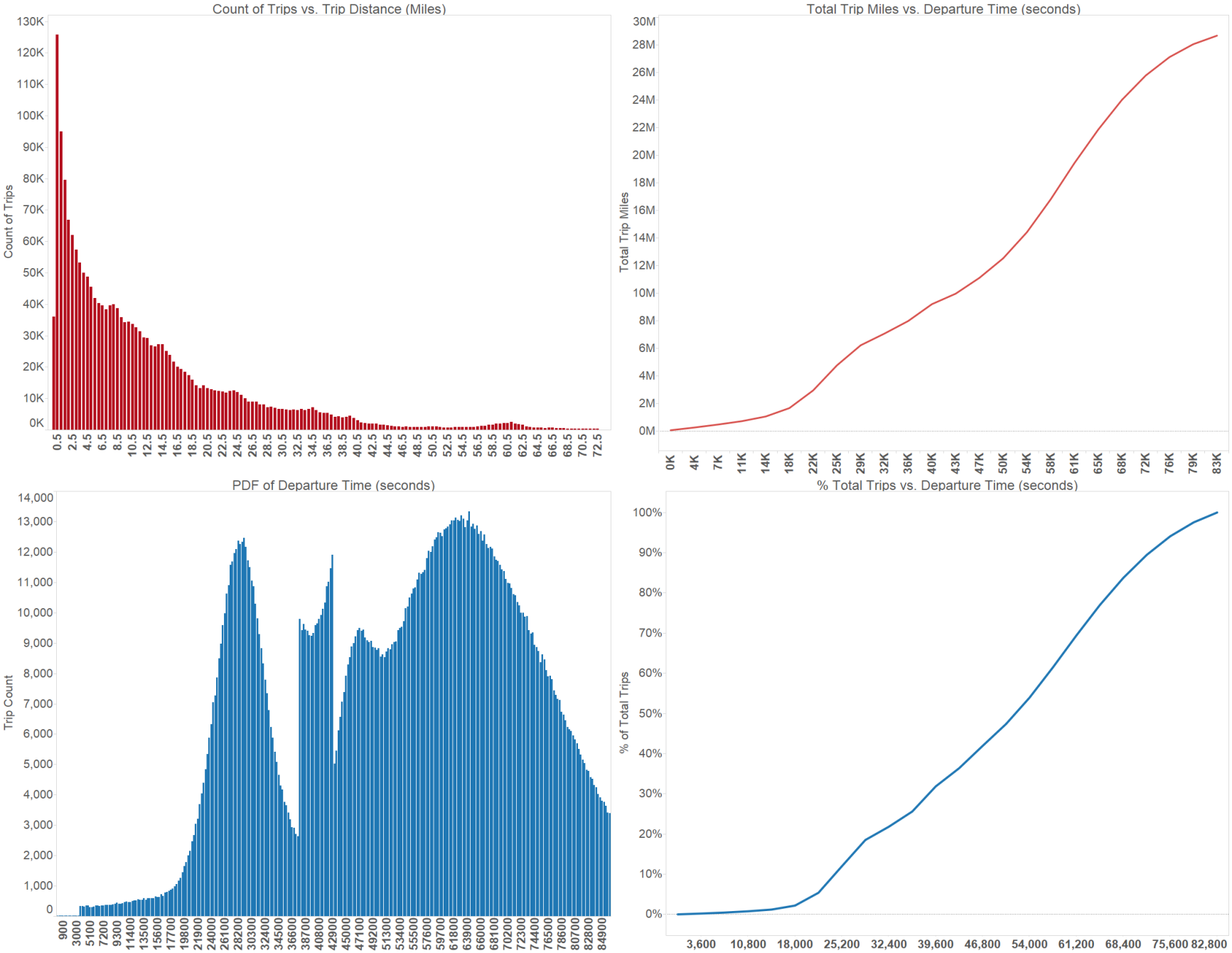

Figure A.129: Vermont: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

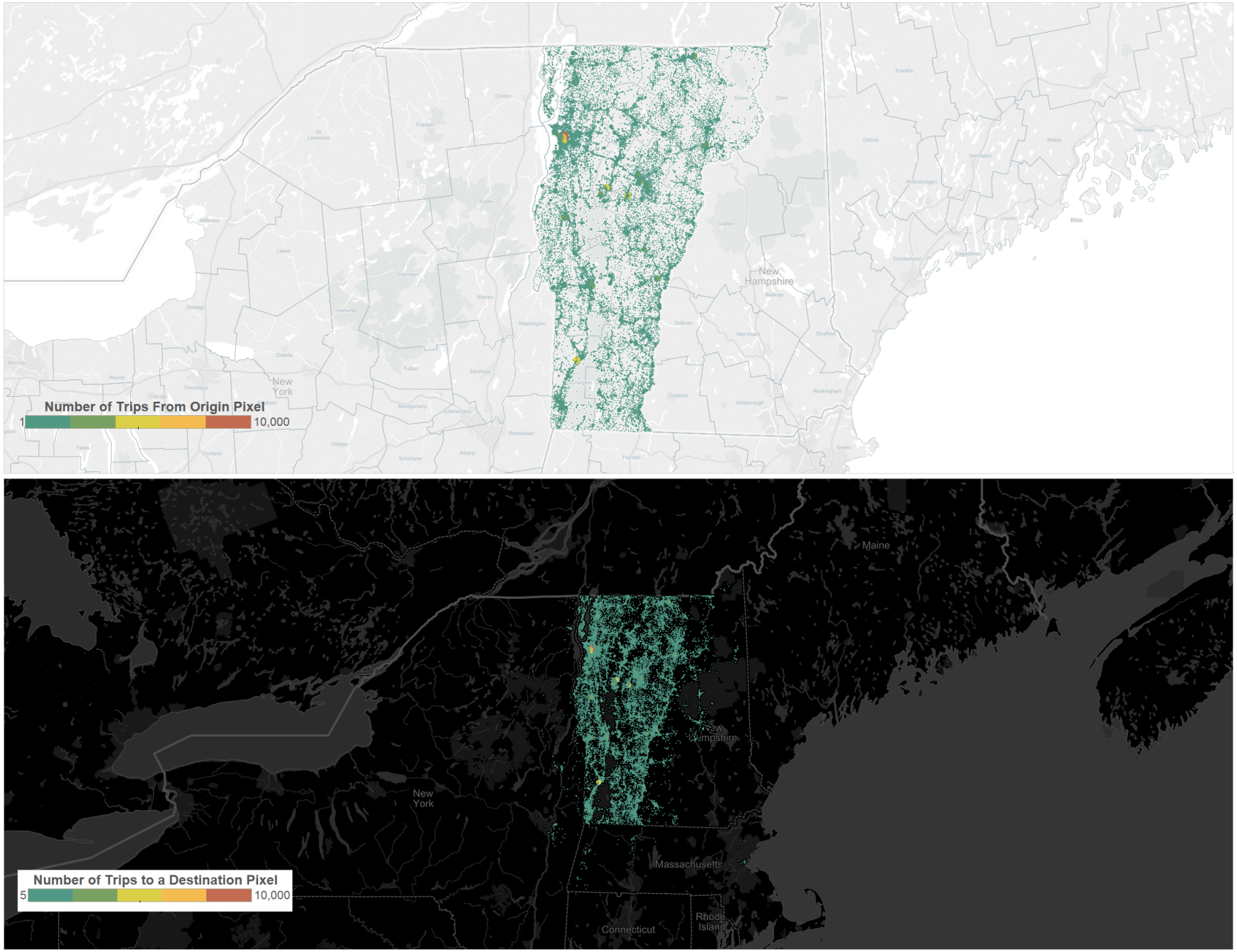

Figure A.130: Vermont: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

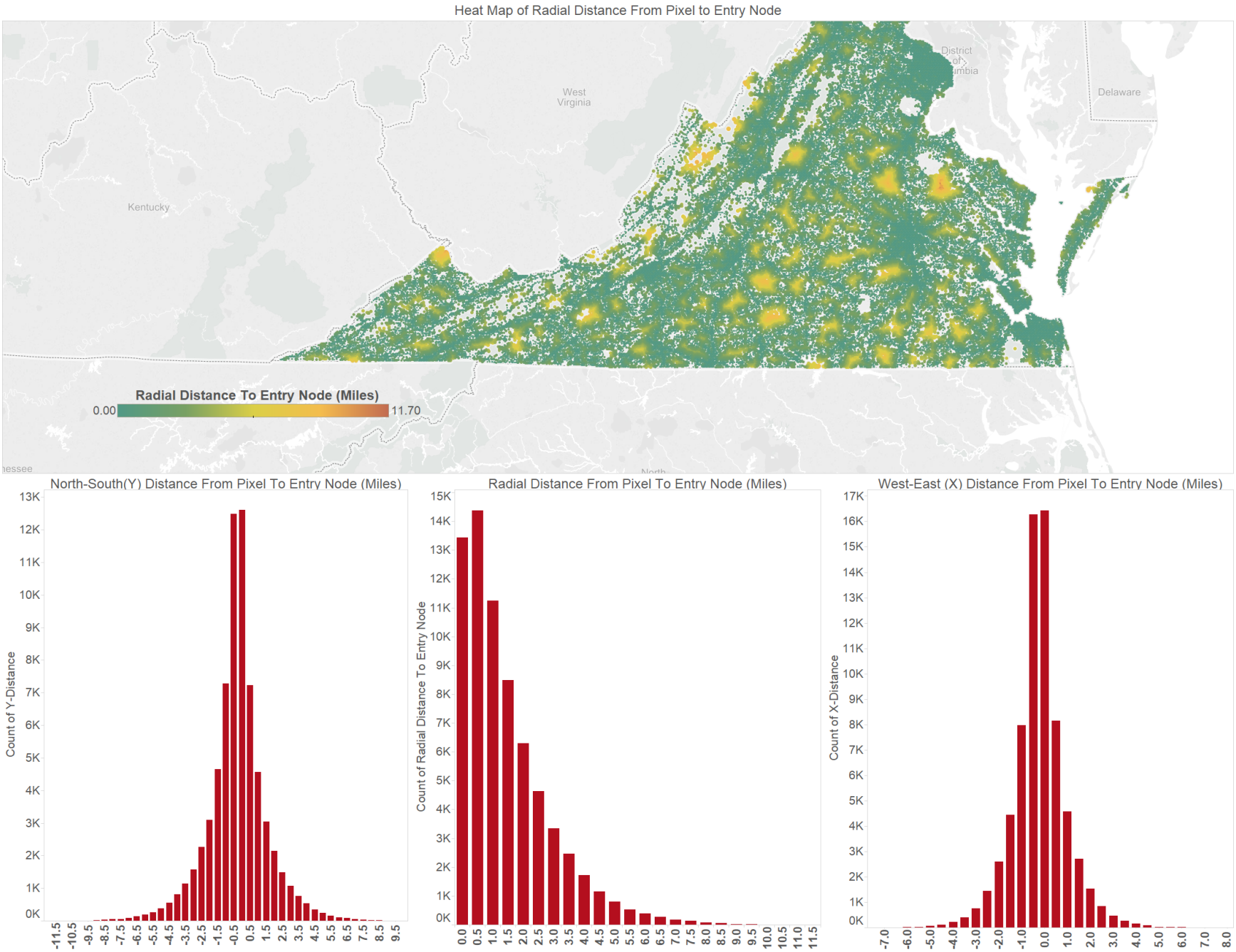

Figure A.131: Virginia: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

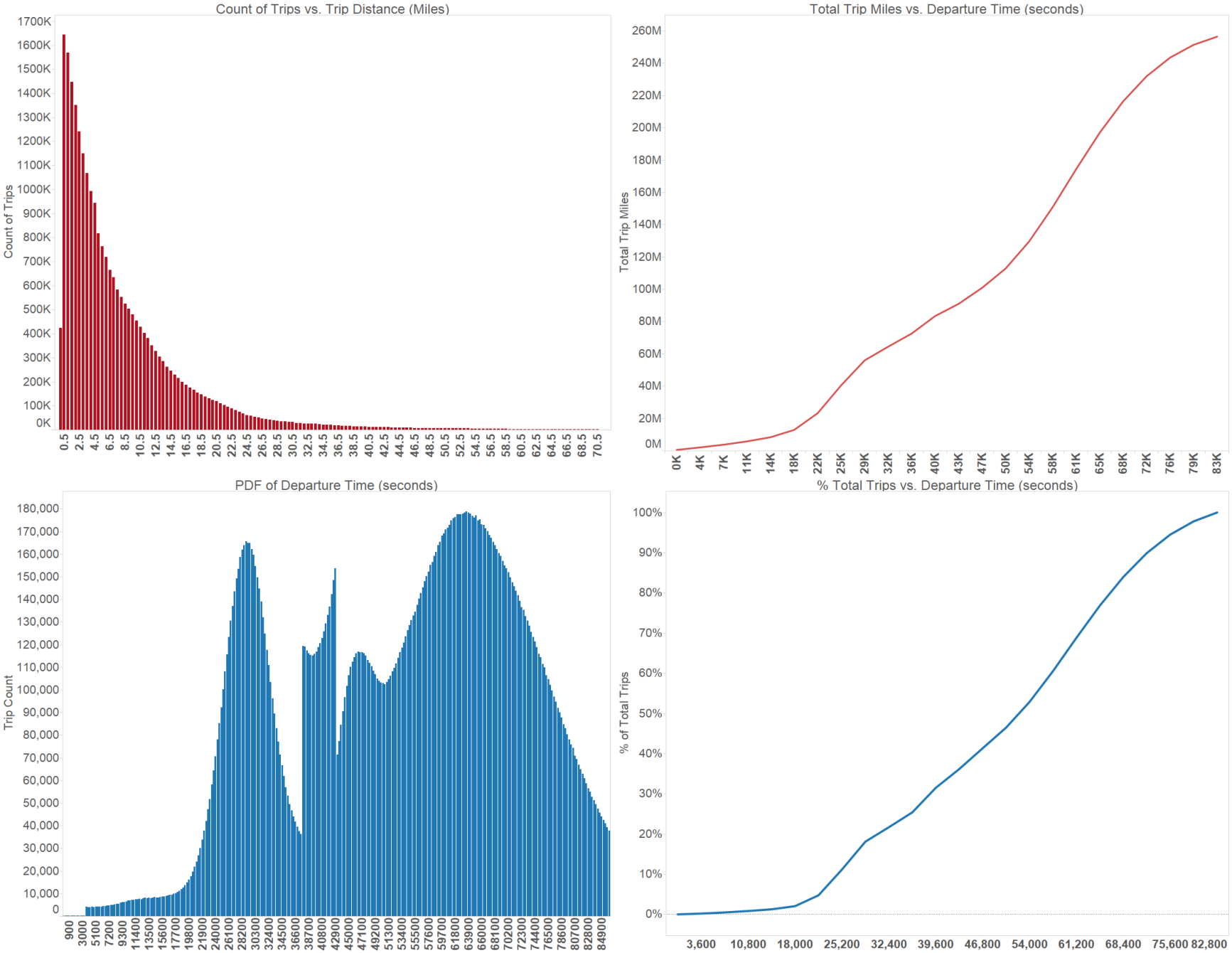

Figure A.132: Virginia: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

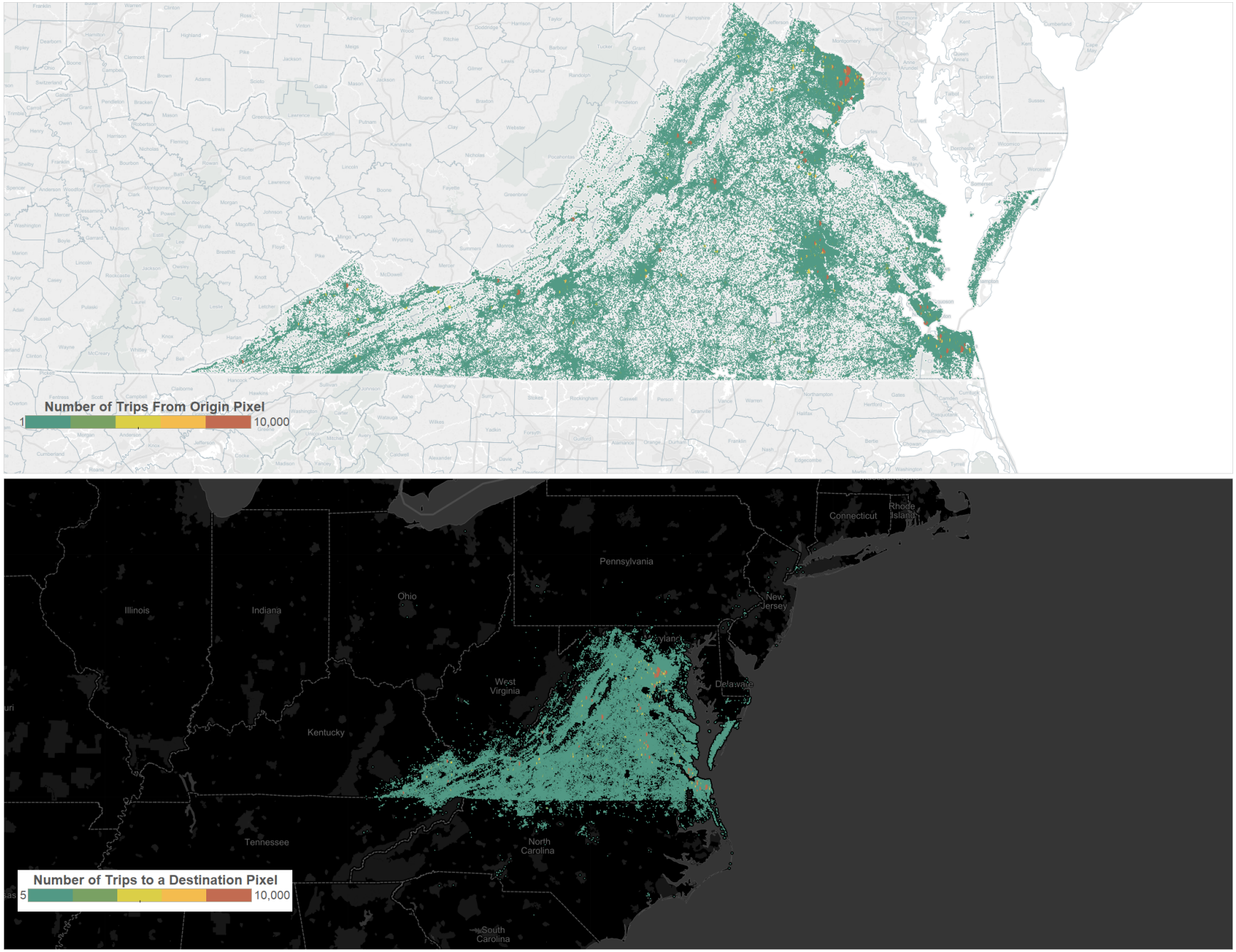

Figure A.133: Virginia: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

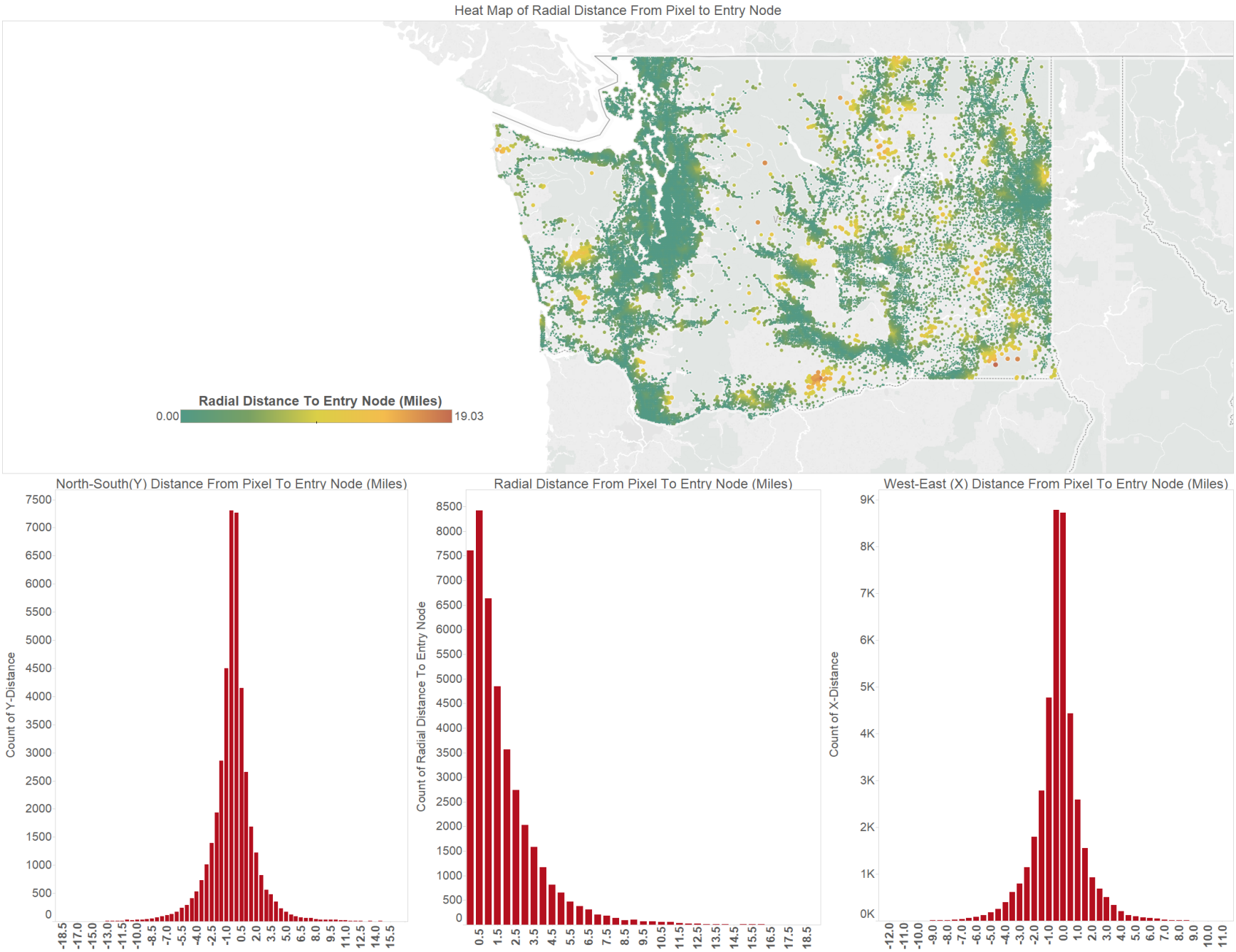

Figure A.134: Washington: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

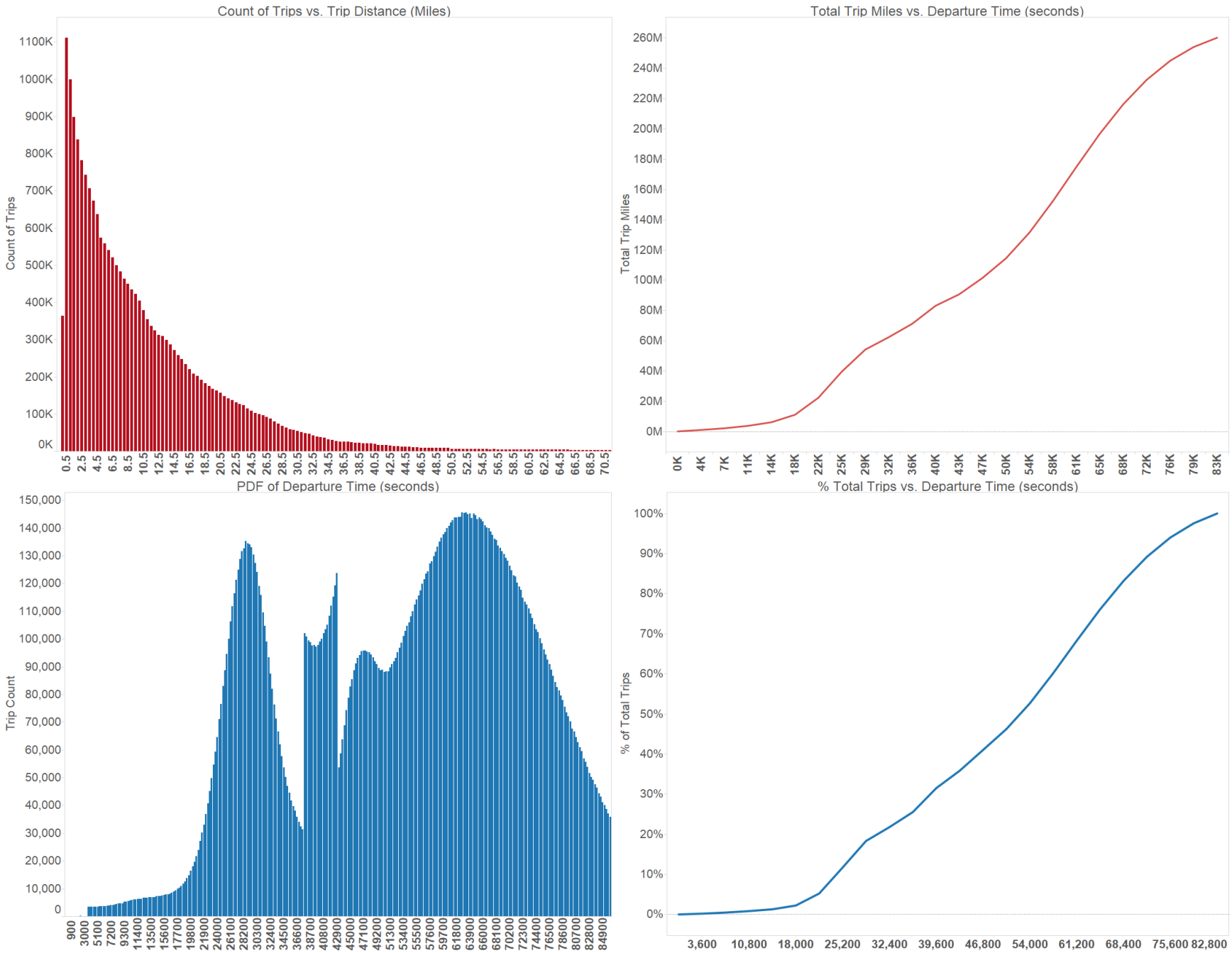

Figure A.135: Washington: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

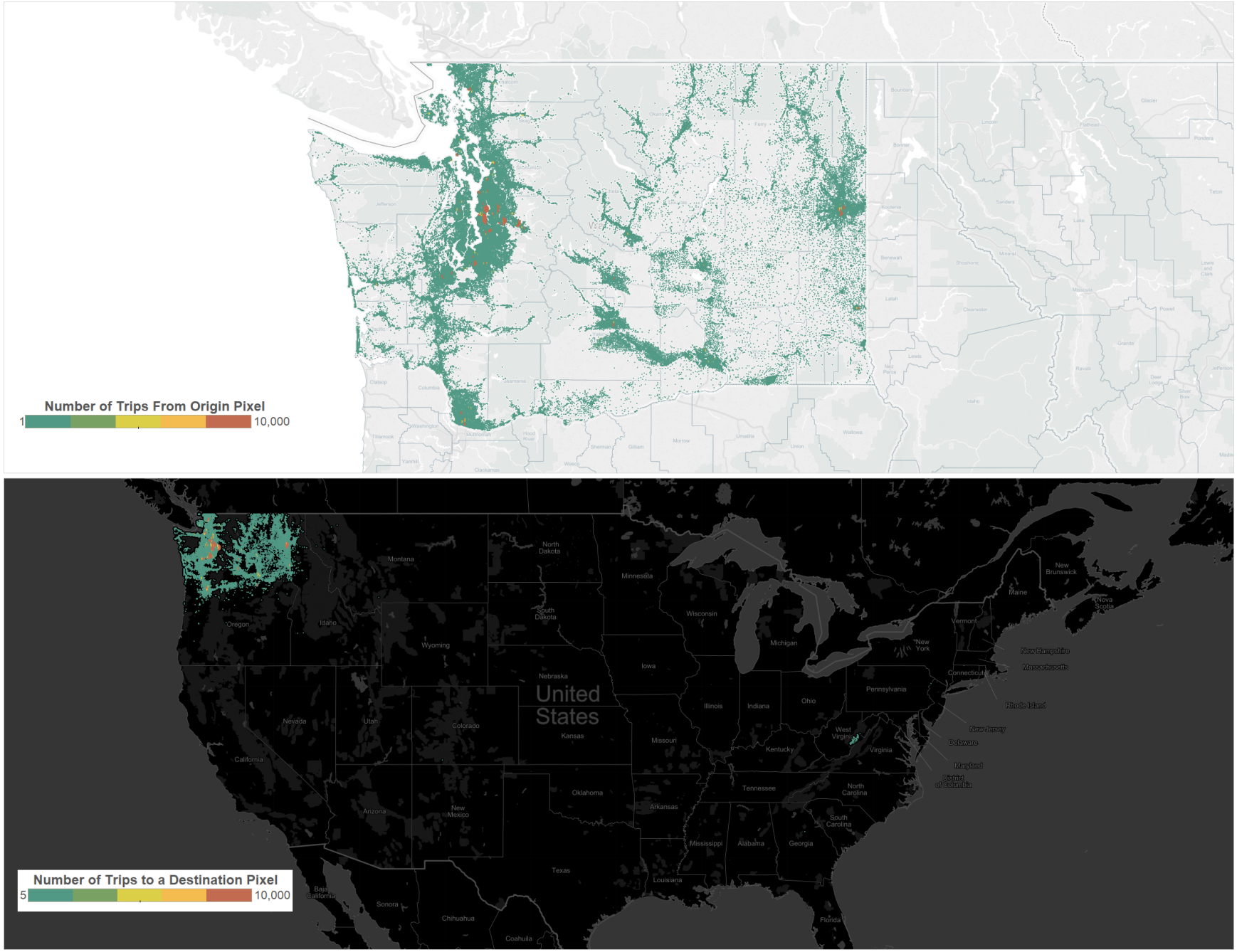

Figure A.136: Washington: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

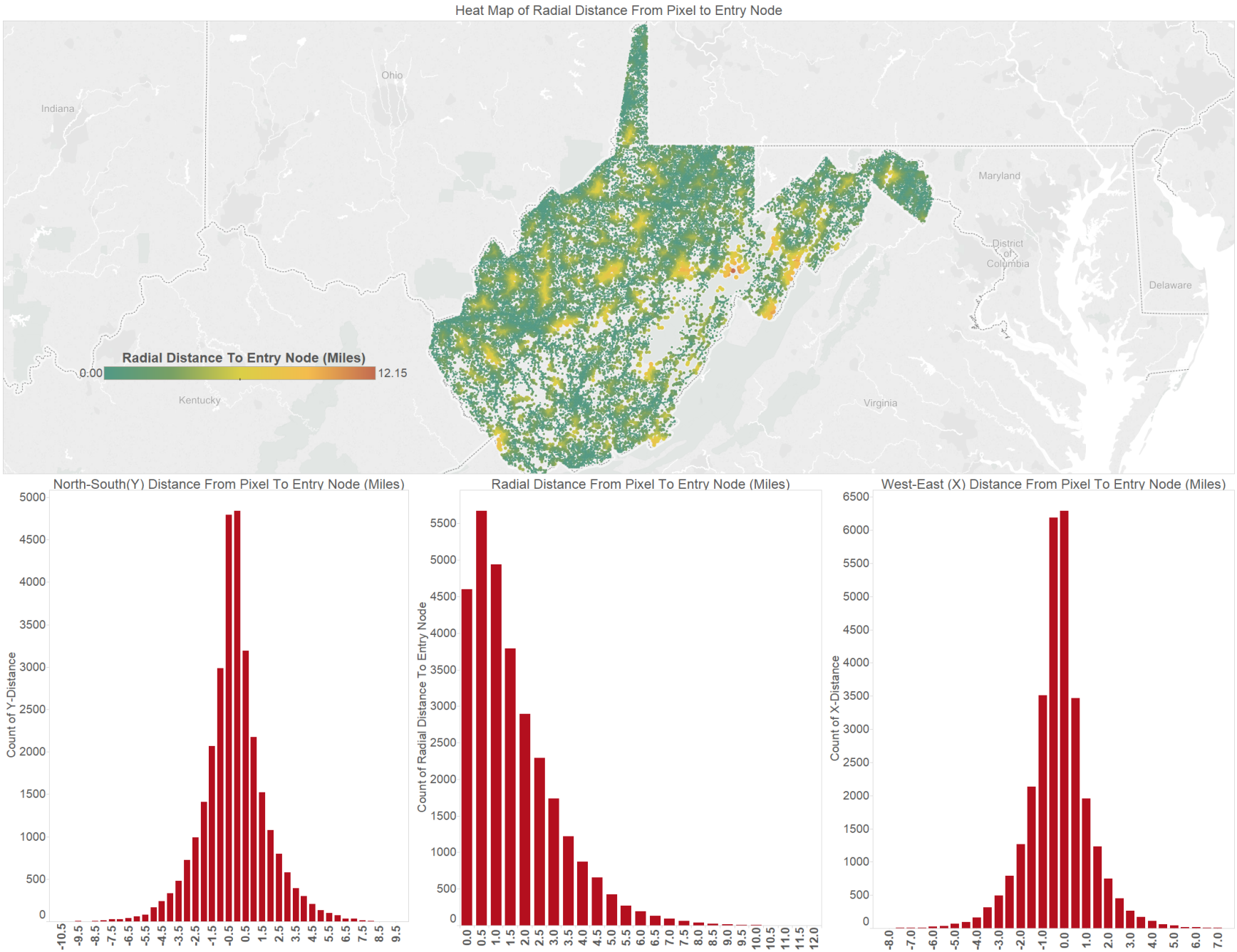

Figure A.137: West_Virginia: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

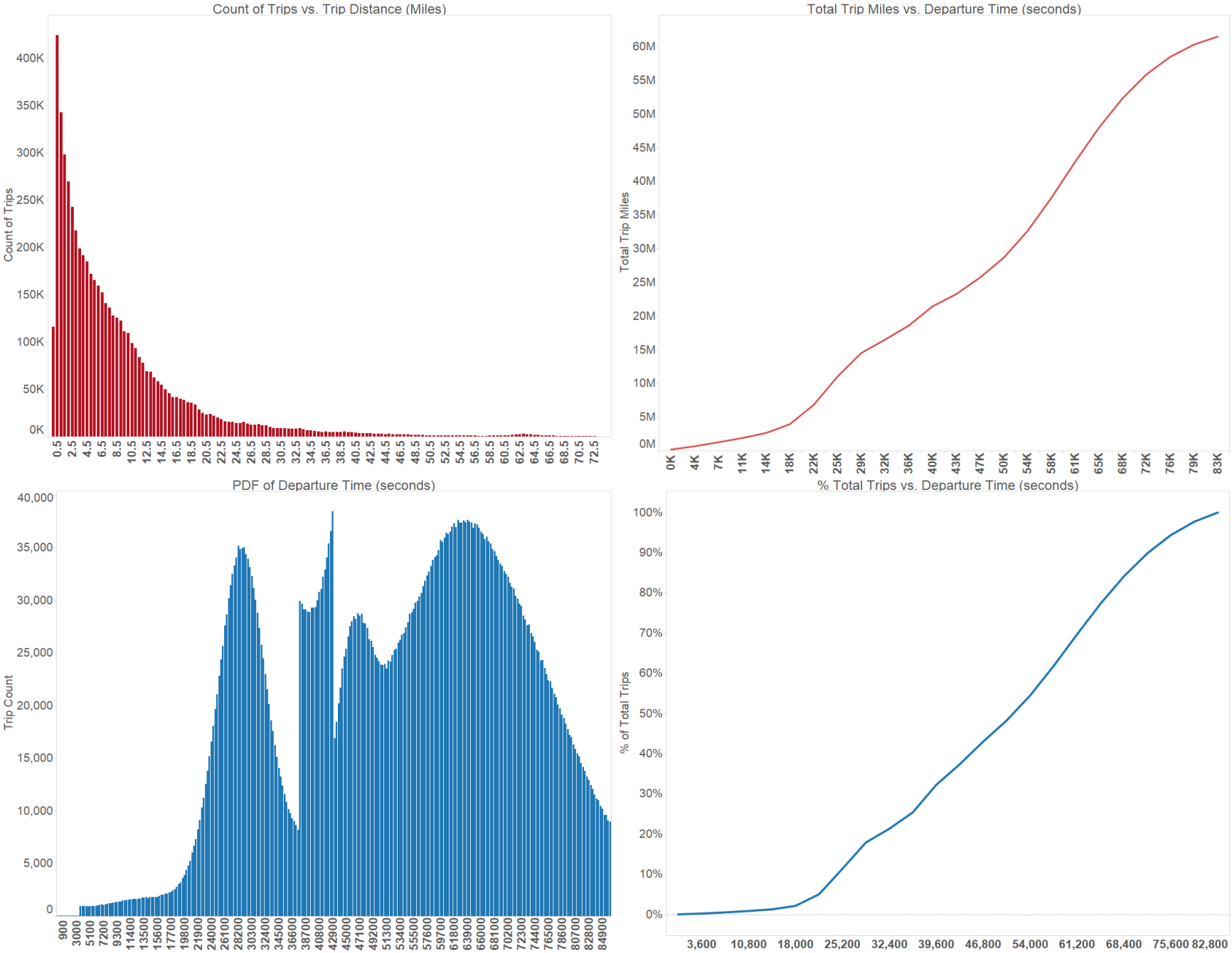

Figure A.138: West_Virginia: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

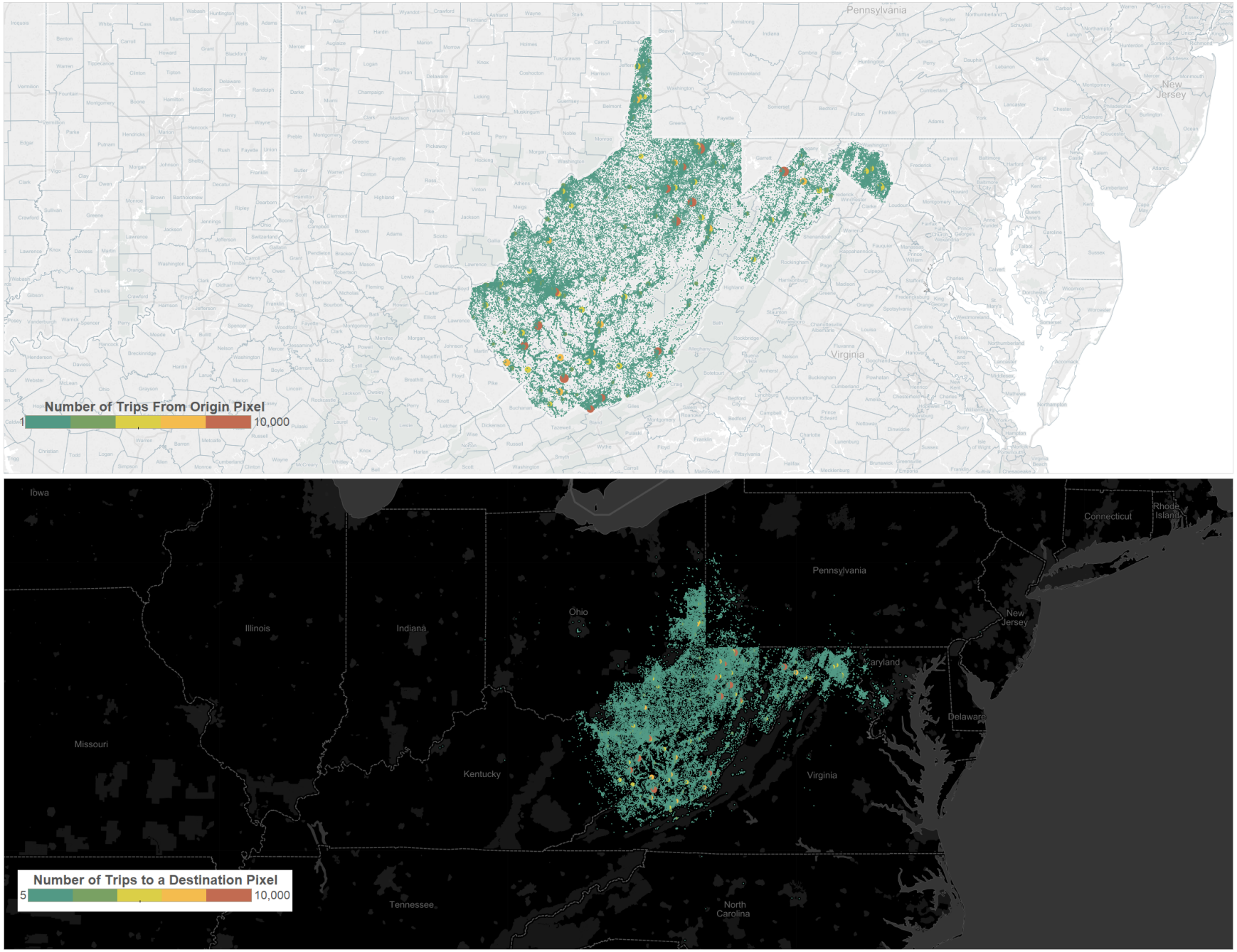

Figure A.139: West_Virginia: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

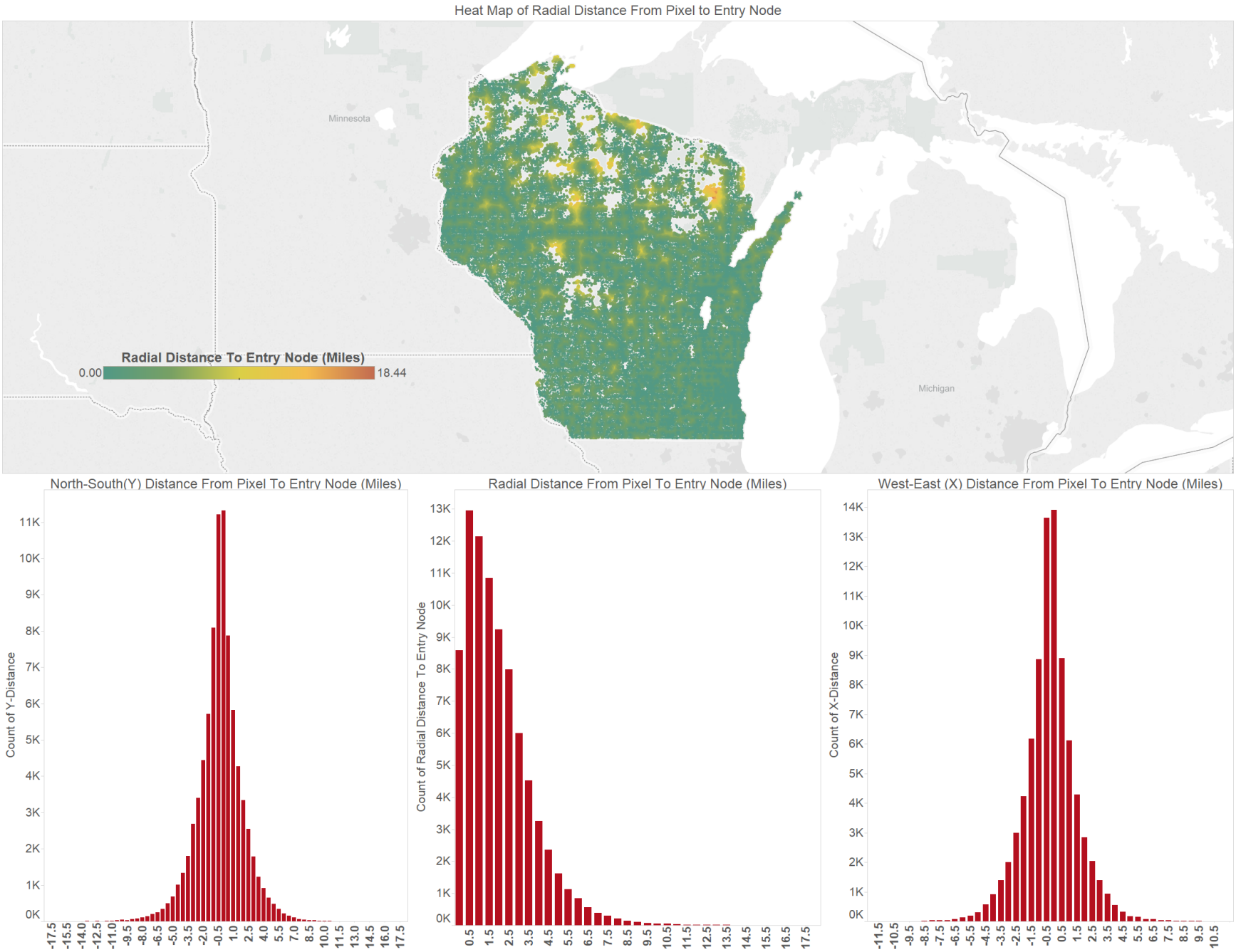

Figure A.140: Wisconsin: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

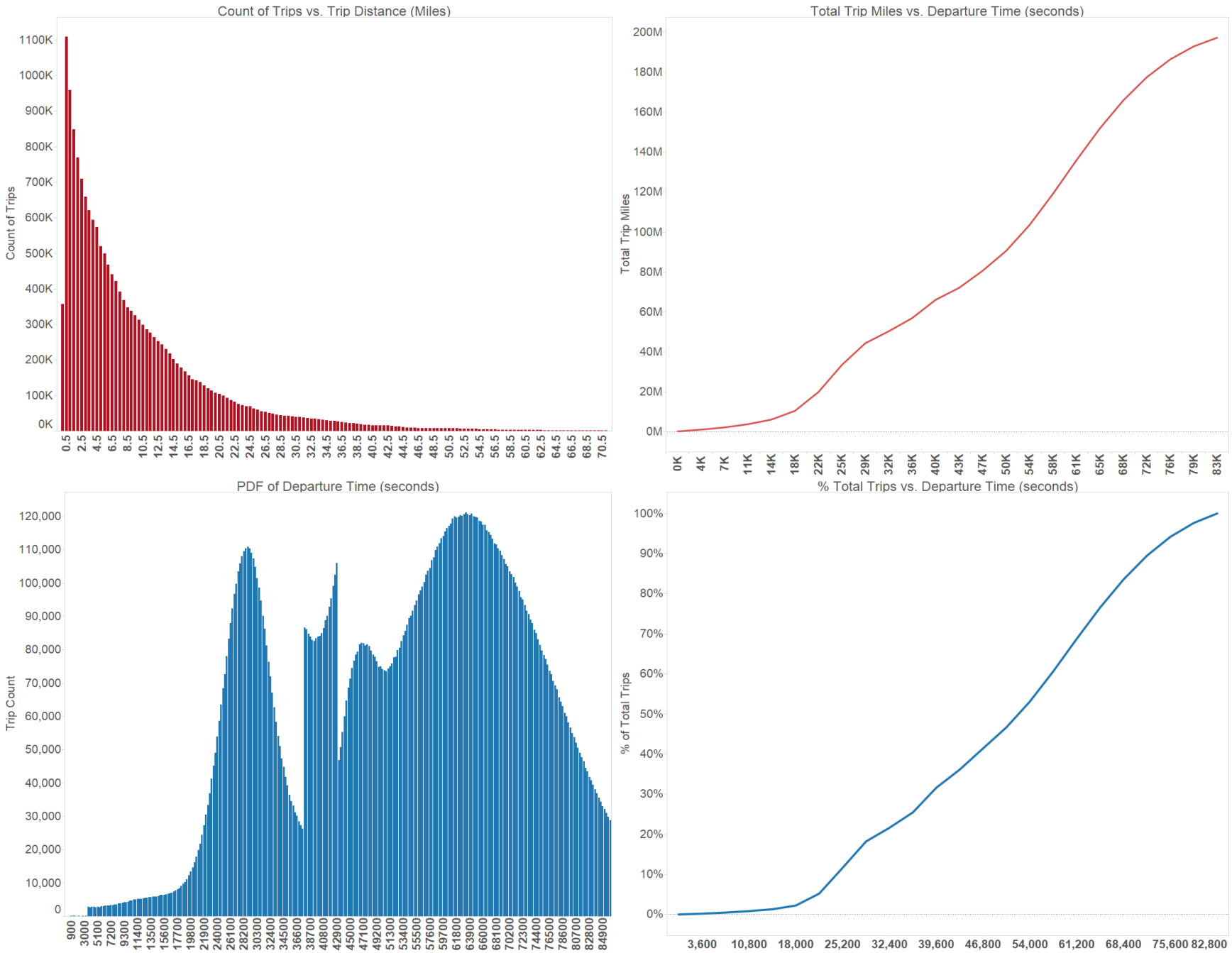

Figure A.141: Wisconsin: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

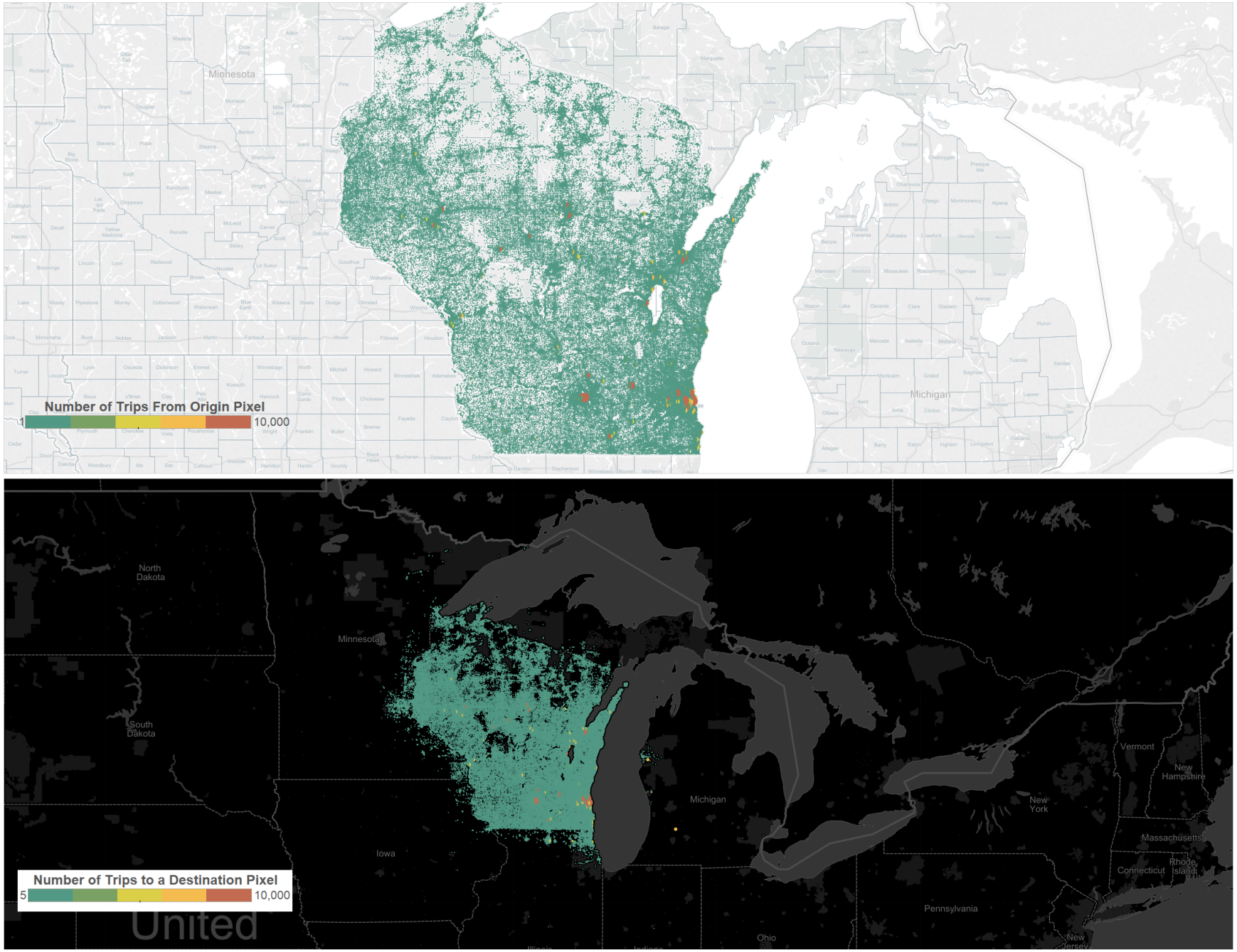

Figure A.142: Wisconsin: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

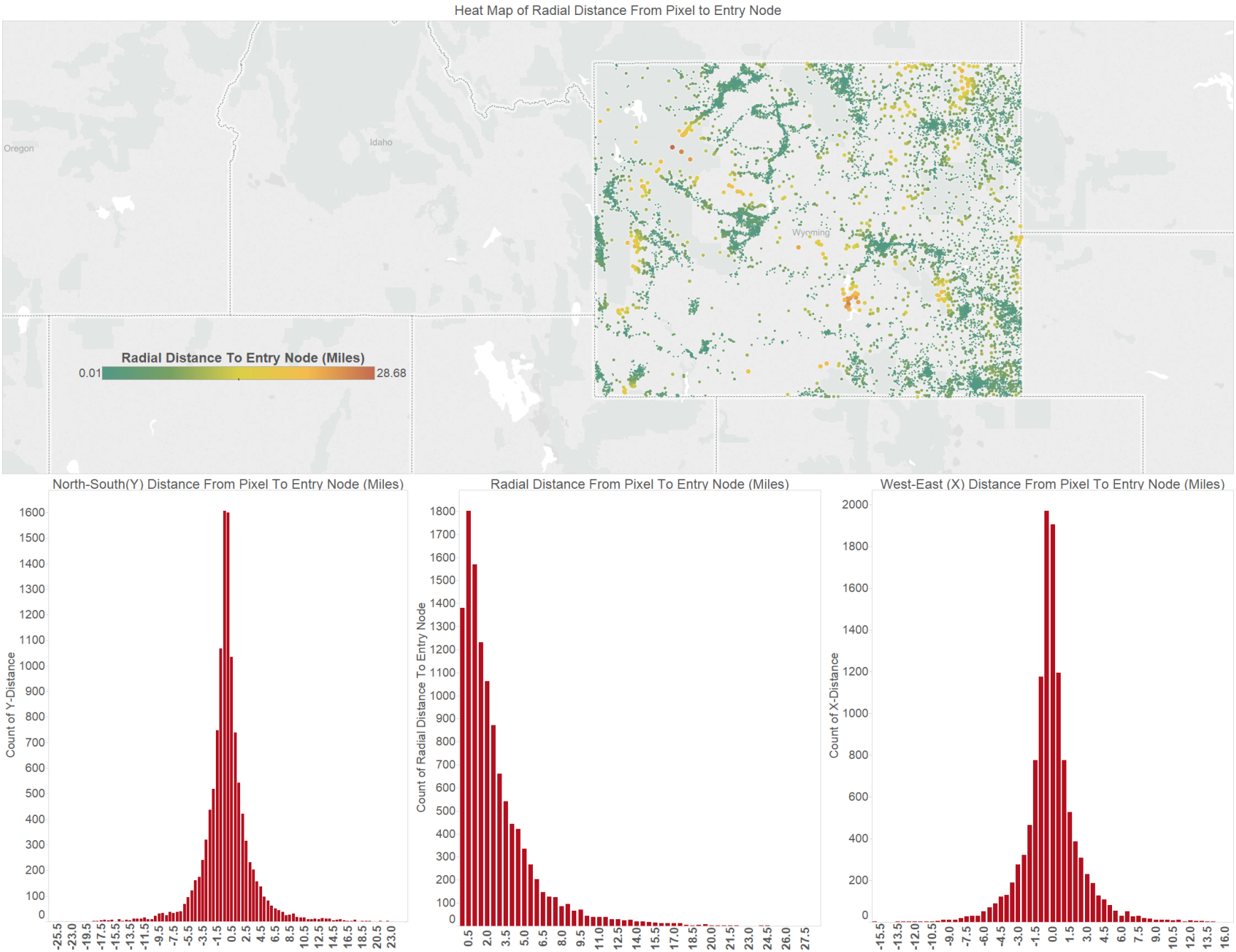

Figure A.143: Wyoming: Heat map of Radial Distance oPixel to Entry Node
Distribution of Vertical Distance from EntryNode, Distribution of Radial Distance from EntryNode, Distribution of Horizontal Distance from EntryNode

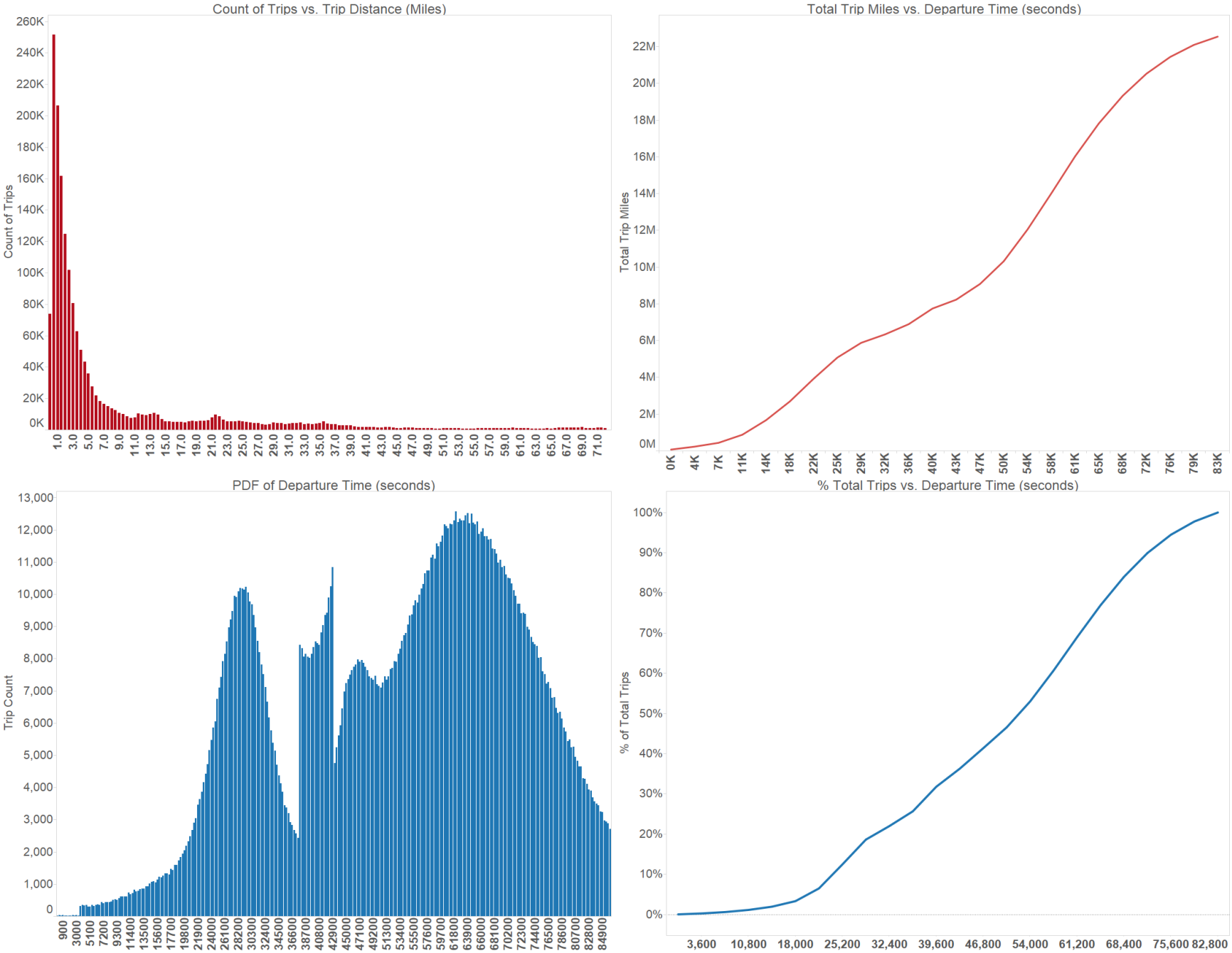

Figure A.144: Wyoming: Trip Distance Distribution, Cumulative Distribution of Trip Miles
PDF of Trip Departure Times, Cumulative Trip Departure Times

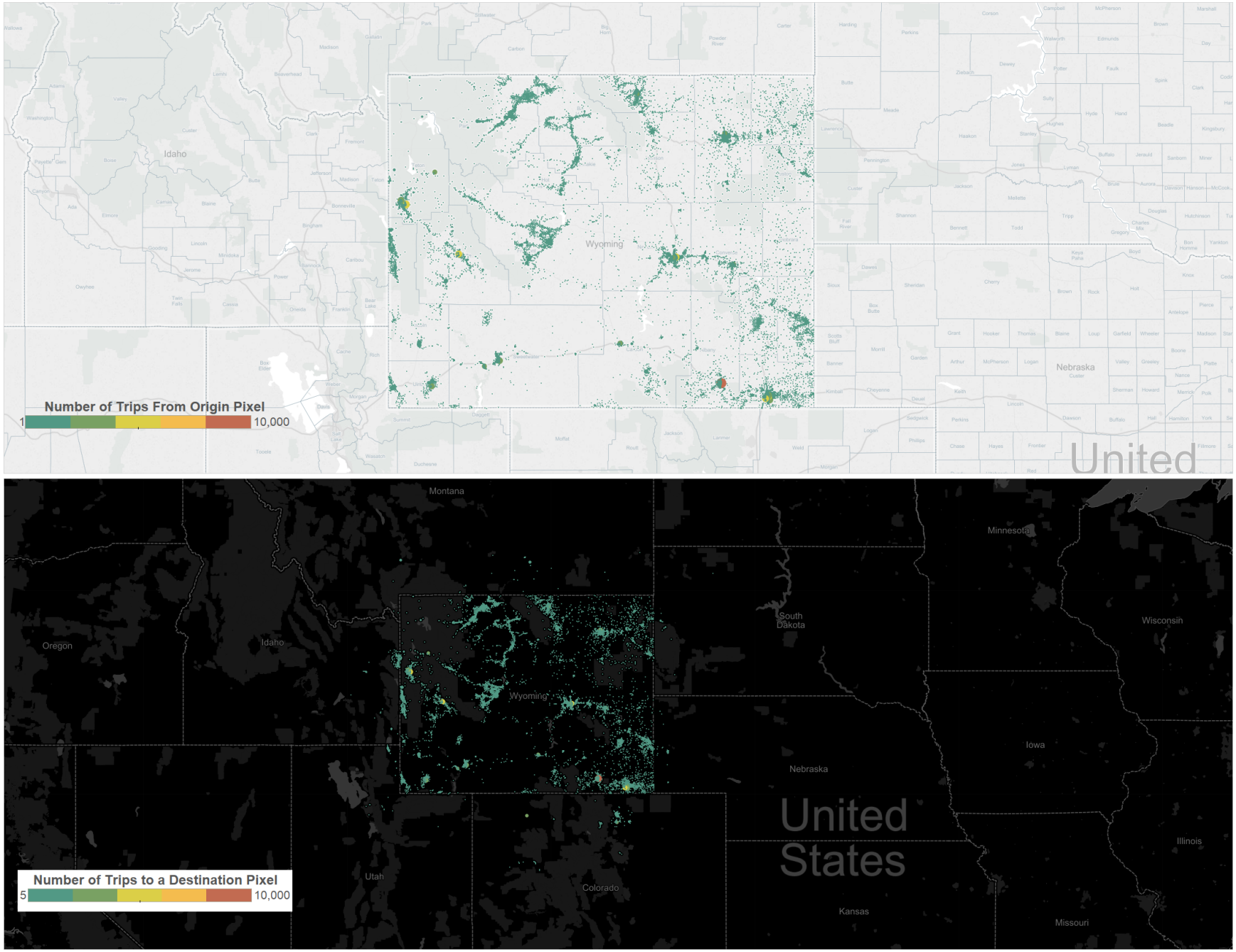

Figure A.145: Wyoming: Heat Map of Trip Generation by Pixel
Heat Map of Trips to dPixel

Appendix B

Code

Java			Page 187
Section B.1.1	Grid Creation		Page 187
	CreateGrid.java	Page 187	
	Pixelization.java	Page 188	
	XMLReader.java	Page 190	
	CreateSqlTables.java	Page 191	
	TestFrame.java	Page 192	
Section B.1.2	oTrip Generation		Page 195
	TripGenerator.java	Page 195	
	CreateTable.java	Page 196	
Section B.1.3	Path File Generation		Page 197
	PathFinder.java	Page 197	
	FindEntryNodes.java	Page 199	
	NodeObject.java	Page 200	
	LinkObject.java	Page 201	
	LinkDij.java	Page 202	
Section B.1.4	Data Visualization		Page 203
	CreateJsonFiles.java	Page 203	
Section B.1.5	Database Assist and Data Structures		Page 204
	KdTreeST.java	Page 204	
	CreateTables.java	Page 208	
JavaScript and HTML			Page 208
Section B.2.1	WebGL Globe Animation		Page 208
	WebGL_Globe.html	Page 208	
	Globe.js	Page 210	
Python and Bash Scripts			Page 213
Section B.3.1	Network Data Cleaning		Page 213
	Nodeclean.py	Page 213	
	Linkclean.py	Page 213	
	Restclean.py	Page 213	

B.1 Java Code

B.1.1 Grid Creation

CreateGrid.java

```java

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

/**
 *  Created by: Tharald S. Fongaard & Jacob Perricone
 *  Advisor: Alain Kornhauser
 *  Date: 05/05/2015
 *
 *  Part Of Project: ORF 376 - Independent Work
 *
 *  Princeton University    2015
 */
public class CreateGrid {
    // Protected Private Objects
    private SeparateChainingHashST<String,SeparateChainingHashST<Double,
          Pixelization>> pixels;
    private XMLReader xmlReader;
    private SeparateChainingHashST<Double,String> keyState;

    // Constructor
    public CreateGrid() throws ClassNotFoundException, SQLException {
        xmlReader = new XMLReader();
        pixels = new SeparateChainingHashST<String, SeparateChainingHashST<Double
              , Pixelization>>();
        keyState = new SeparateChainingHashST<Double, String>();

        for(String state: xmlReader.getAllStates()){
            SeparateChainingHashST<Double,Pixelization> stateTable = new
                  SeparateChainingHashST<Double, Pixelization>();
            pixels.put(state,stateTable);
        }

        int iStart;
        int iEnd;
        int jStart;
        int jEnd;

        double initiallat = 24.544091;
        double initallon = -120.5;
        double finallon = -106.25;
        double finallat = 48.987386;

        iStart = (int) Math.floor(138.348*(initallon + 97.5)*Math.cos(Math.
              toRadians(initiallat)));
        jStart = (int) Math.floor(138.348*(initiallat - 37.0));
        iEnd = (int) Math.floor(138.348*(finallon + 97.5)*Math.cos(Math.toRadians
              (finallat)));
        jEnd = (int) Math.floor(138.348*(finallat - 37.0));

        iStart = 2000;
        iEnd = 2421;

        StdOut.println("iStart:" + iStart);
        StdOut.println("iEnd:" + iEnd);
        StdOut.println("jStart:" + jStart);
        StdOut.println("jEnd:" + jEnd);
        Pixelization firstP = new Pixelization(iStart,jStart);
        Point2D fBl = firstP.get_bl();

        Pixelization centerP = new Pixelization(0,0);
        Point2D centerBl = centerP.get_bl();
        Point2D centerBr = centerP.get_br();
        StdOut.println(centerBr.toString());
        int a = 0;

        String url = "jdbc:mysql://127.0.0.1:3306/Grid";

        for(int i = iEnd; i >= iStart; i--){
            Class.forName("com.mysql.jdbc.Driver");
            Connection m_Connection = DriverManager.getConnection(url, "tharald",
                  "putin");
            StdOut.println("" + i + " / " + iEnd);
            for(int j = jStart; j <= jEnd; j++){
                Pixelization pixel = new Pixelization(i,j);
                double key = pixel.get_id();
                Point2D cent = pixel.get_centroid();
                String state = xmlReader.getClosestState(cent);

                Point2D bl = pixel.get_bl();
                Point2D br = pixel.get_br();
                Point2D tr = pixel.get_tr();
                Point2D tl = pixel.get_tl();
                int x = i;
                int y = j;

                PreparedStatement total = m_Connection.prepareStatement("REPLACE
                      INTO "+ state + " VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?)");

                //ID
                total.setDouble(1, key);

                //X/Y
                total.setInt(2, x);
                total.setInt(3, y);

                //Centroid Long/Lat
                total.setDouble(4, cent.x());
                total.setDouble(5, cent.y());

                //Bottom Left Long/Lat
                total.setDouble(6, bl.x());
                total.setDouble(7, bl.y());

                //Bottom Right Long/Lat
                total.setDouble(8, br.x());
                total.setDouble(9, br.y());

                //Top Left Long/Lat
```

```
                total.setDouble(10, tl.x());
                total.setDouble(11, tl.y());

                //Top Right Long/Lat
                total.setDouble(12, tr.x());
                total.setDouble(13, tr.y());

                // Store Pixel
                total.executeUpdate();
            }
            m_Connection.close();
        }
    }

    /*------------------------------ PUBLIC METHODS
        ------------------------------*/

    // Get All States In The Grid
    public Iterable<String> getStates(){
        return pixels.keys();
    }

    // Get Number of Pixels In The Grid
    public double Size(){
        return pixels.size();
    }

    // Get All Pixels Mapped to State
    public Iterable<Double> getKeys(String state){
        return pixels.get(state).keys();
    }

    // Get X Coordinate to Pixel
    public int getX(double key){
        String state = keyState.get(key);
        Pixelization pixel = pixels.get(state).get(key);
        return pixel.get_x();
    }

    // Get Y Coordinate to Pixel
    public int getY(double key){
        String state = keyState.get(key);
        Pixelization pixel = pixels.get(state).get(key);
        return pixel.get_y();
    }

    // Get Point Centroid to Pixel
    public Point2D getPixelCentroid(double key){
        String state = keyState.get(key);
        Pixelization pixel = pixels.get(state).get(key);
        Point2D returnPoint = pixel.get_centroid();
        return returnPoint;
    }

    // Get Point Bottom Right to Pixel
    public Point2D getPixelBr(double key){
        String state = keyState.get(key);
        Pixelization pixel = pixels.get(state).get(key);
        Point2D returnPoint = pixel.get_br();
        return returnPoint;
    }

    // Get Point Bottom Left to Pixel
    public Point2D getPixelBl(double key){
        String state = keyState.get(key);
        Pixelization pixel = pixels.get(state).get(key);
        Point2D returnPoint = pixel.get_bl();
        return returnPoint;
    }

    // Get Point Top Right to Pixel
    public Point2D getPixelTr(double key){
        String state = keyState.get(key);
        Pixelization pixel = pixels.get(state).get(key);
        Point2D returnPoint = pixel.get_tr();
        return returnPoint;
    }

    // Get Point Top Left to Pixel
    public Point2D getPixelTl(double key){
        String state = keyState.get(key);
        Pixelization pixel = pixels.get(state).get(key);
        Point2D returnPoint = pixel.get_tl();
        return returnPoint;
    }

    // Test Execution
    public static void main(String[] args) {

        CreateGrid grid = null;
        try {
            grid = new CreateGrid();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        StdOut.println("Done");
    }
}
```

./Code/CreateGrid.java

Pixelization.java

```
import java.lang.Math;

public class Pixelization {
  // Protected Private Objects
    double id;
    int x;
    int y;
    double CENTERLON = -97.5;
    double CENTERLAT = 37;
    double YHEIGHT = 0.00722814;
    double XWIDTH =  0.00944344;
```

```java
    Point2D bottomleft;
    Point2D bottomright;
    Point2D topleft;
    Point2D topright;
    Point2D centroid;

  // Constructor
  public Pixelization(int x, int y) {
    this.x = x;
    this.y = y;
    this.id = createid(x, y);
    this.bottomleft = createbottomleft(x, y);
    this.bottomright = createbottomright(x,y);
    this.topleft = createtopleft(x,y);
    this.topright = createtopright(x,y);
    this.centroid = createcentroid();
  }
  // Cantor Pairing Function with offset
  private double createid(int a, int b) {
        double returnn = 0.5*(a+195593+b+1724)*(a+195593+b+1724+1)+b+1724;
        return returnn;
    }

  // Given X,Y coordinates, create BL Corner
  private Point2D createbottomleft(int i, int j) {
    double lon, lat;
    double cosangle = Math.cos(Math.toRadians(CENTERLAT + j*YHEIGHT));
    lon = (CENTERLON + (YHEIGHT*i)/cosangle);
    lat = (CENTERLAT + YHEIGHT*j);
    Point2D p = new Point2D(lon, lat);
    return p;
  }

  // Given X,Y coordinates, create BR Corner
  private Point2D createbottomright(int i, int j) {
    i++;
    double lon, lat;
    double cosangle = Math.cos(Math.toRadians(CENTERLAT + j*YHEIGHT));
    lon = (CENTERLON + (YHEIGHT*i)/cosangle);
    lat = (CENTERLAT + YHEIGHT*j);
    Point2D p = new Point2D(lon, lat);
    return p;
  }

  // Given X,Y coordinates, create TL Corner
  private Point2D createtopleft(int i, int j) {
    j++;
    double lon, lat;
    double cosangle = Math.cos(Math.toRadians(CENTERLAT + j*YHEIGHT));
    lon = (CENTERLON + (YHEIGHT*i)/cosangle);
    lat = (CENTERLAT + YHEIGHT*j);
    Point2D p = new Point2D(lon, lat);
    return p;
  }

  // Given X,Y coordinates, create TR Corner
  private Point2D createtopright(int i, int j) {
    j++;
    i++;
    double lon, lat;
    double cosangle = Math.cos(Math.toRadians(CENTERLAT + j*YHEIGHT));
    lon = (CENTERLON + (YHEIGHT*i)/cosangle);
    lat = (CENTERLAT + YHEIGHT*j);
    Point2D p = new Point2D(lon, lat);
    return p;
  }

  // Given X,Y coordinates, create centroid
  private Point2D createcentroid() {
    double lon, lat;
    lon = (bottomleft.x() + bottomright.x())/2;
    lat = (bottomleft.y() + topleft.y())/2;
    Point2D p = new Point2D(lon, lat);
    return p;
  }

  // Get X Coordinate of Pixel
  public int get_x() {
    return x;
  }

  // Get Y Coordinate of Pixel
  public int get_y() {
    return y;
  }

  // Get ID of Pixel
  public double get_id() {
    return id;
  }

  // Get Bottom Right of Pixel
  public Point2D get_br() {
    return bottomright;
  }

  // Get Bottom Left of Pixel
  public Point2D get_bl() {
    return bottomleft;
  }

  // Get Top Left of Pixel
  public Point2D get_tl() {
    return topleft;
  }

  // Get Top Right of Pixel
  public Point2D get_tr() {
    return topright;
  }

    // Get Centroid of Pixel
    public Point2D get_centroid() {
        return centroid;
    }
}
```

./Code/Pixelization.java

XMLReader.java

```java
import jdk.internal.org.xml.sax.SAXException;
import org.w3c.dom.Document;
import org.w3c.dom.*;



import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import java.awt.*;
import java.io.IOException;

/**
 *  Created by: Tharald S. Fongaard & Jacob Perricone
 *  Advisor: Alain Kornhauser
 *  Date: 05/05/2015
 *
 *  Part Of Project: ORF 376 - Independent Work
 *
 *  Princeton University    2015
 */
public class XMLReader {

    // Protected Private Objects
    private KdTreeST<Polygon> treeST;
    private SeparateChainingHashST<String,Polygon> states;

    // Constructor, fetches static XML data (state boundaries and centroids)
    public XMLReader(){
        treeST = new KdTreeST<Polygon>();
        states = new SeparateChainingHashST<String, Polygon>();

        DocumentBuilderFactory builderFactory = DocumentBuilderFactory.
             newInstance();

        try{
            DocumentBuilder dBuilder = builderFactory.newDocumentBuilder();
            Document document = dBuilder.parse(XMLReader.class.
                 getResourceAsStream("/states.xml"));
            Document centroidsDocument = dBuilder.parse(XMLReader.class.
                 getResourceAsStream("/centroids.xml"));
            document.normalize();
            centroidsDocument.normalize();

            NodeList centroids = centroidsDocument.getElementsByTagName("states")
                 ;
            Node centroidRoot = centroids.item(0);
            Element centroidElementRoot = (Element) centroidRoot;

            NodeList rootNodes = document.getElementsByTagName("states");
            org.w3c.dom.Node rootNode = rootNodes.item(0);
            org.w3c.dom.Element rootElement = (org.w3c.dom.Element) rootNode;

            NodeList statesList = rootElement.getElementsByTagName("state");
            NodeList centroidList = centroidElementRoot.getElementsByTagName("
                 state");

            for(int i = 0; i < statesList.getLength();i++){
                org.w3c.dom.Node theState = statesList.item(i);
                org.w3c.dom.Element stateElement = (org.w3c.dom.Element) theState
                     ;
                //StdOut.println("This state is " + stateElement.getAttribute("
                     name"));
                Polygon newState = new Polygon(stateElement.getAttribute("name"),
                      stateElement.getAttribute("colour"));

                NodeList pointList = stateElement.getElementsByTagName("point");
                for(int j = 0; j < pointList.getLength();j++){
                    org.w3c.dom.Element pointElement = (org.w3c.dom.Element)
                         pointList.item(j);
                    Point2D newPoint = new Point2D(Double.parseDouble(
                         pointElement.getAttribute("lng")),Double.parseDouble(
                         pointElement.getAttribute("lat")));
                    newState.add(newPoint);
                    //StdOut.println("Lat: " + pointElement.getAttribute("lat") +
                          "Long: " + pointElement.getAttribute("lng"));
                }
                Node statecentroid = centroidList.item(i);
                Element centroidElement = (Element) statecentroid;
                Point2D centroid = new Point2D(Double.parseDouble(centroidElement
                     .getAttribute("lon")),Double.parseDouble(centroidElement.
                     getAttribute("lat")));
                treeST.insert(centroid,newState);
                states.put(stateElement.getAttribute("name"),newState);

            }
            Polygon nostate = new Polygon("NoState","#ff0000");
            states.put("NoState", nostate);

        }
        catch (ParserConfigurationException e){
            e.printStackTrace();
        }
        catch (org.xml.sax.SAXException e){
            e.printStackTrace();
        }
        catch (IOException e){
            e.printStackTrace();
        }
    }

    // Get State Boundary Polygon
    public Polygon getStatePolygon(String state){
        return states.get(state);
    }

    // Get All State Boundary Polygons
    public Iterable<Polygon> getAllPolygons(){
        return treeST.getAllValues();
    }

    // Get Most Likely State Boundary Polygon Given Query Point q
    public String getClosestState(Point2D queryPoint){
        Point2D nearestPoint = treeST.nearest(queryPoint);
        Polygon nearestState = treeST.get(nearestPoint);
        if(nearestState.contains(queryPoint)){
            return nearestState.getName();
        }
        else{
```

```
            for(Polygon pol:treeST.getAllValues()){
                if(pol.contains(queryPoint)){
                    return pol.getName();
                }
            }
        }
        return "NoState";
    }

    // Get Number of States in XML file
    public int size(){
        return treeST.size();
    }

    // Iterable of State Names (SQL Friendly)
    public Iterable<String> getAllStates() {
        return states.keys();
    }

    // Iterable of State Centroids
    public Iterable<Point2D> getAllPoints(){
        return treeST.getAllPoints();
    }

    // Draw all Polygons (Test Method)
    public void drawTree(){
        treeST.draw();
    }

    // Main Test Execution
    public static void main(String[] args){
       XMLReader xmlReader = new XMLReader();
        StdOut.println(xmlReader.size());
       // StdDraw.setXscale(-136,-66);
        // StdDraw.setYscale(20.54,53);
        for(String s: xmlReader.getAllStates()){
            StdOut.println(s);
        }
        /*
        StdDraw.setPenColor(Color.GREEN);
        int points= 0;
        for(Point2D p:xmlReader.getAllPoints()){
            p.draw();
            points++;
        }

        for(Polygon p:xmlReader.getAllPolygons()){
            StdDraw.setPenColor(p.color());
            p.draw();
        }
        //StdOut.println(points);
        //xmlReader.drawTree();
        StdDraw.setPenColor(Color.BLACK);
        StdDraw.setPenRadius(.006);

        for(Point2D p:xmlReader.getAllPoints()){
            p.draw();
        }
        */
    }
}
```

./Code/XMLReader.java

CreateSqlTables.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

/**
 *  Created by: Tharald S. Fongaard & Jacob Perricone
 *  Advisor: Alain Kornhauser
 *  Date: 05/05/2015
 *
 *  Part Of Project: ORF 376 - Independent Work
 *
 *  Princeton University    2015
 */
public class CreateSqlTables {

    // Main Execution
    public static void main(String[] args){
        CreateSqlTables obj = new CreateSqlTables();
        try {
            obj.updateTable();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Private Helper Function, Creates All Pixels in Grid
    private void updateTable() throws ClassNotFoundException, SQLException {

        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://127.0.0.1:3306/Grid";
        Connection m_Connection = DriverManager.getConnection(url, "tharald", "
            putin");
        StdOut.println("Connection Successful");

        CreateGrid grid = new CreateGrid();

        StdOut.println("Grid Created");

        for(String state: grid.getStates()) {
            for (double key : grid.getKeys(state)) {
                Point2D centroid = grid.getPixelCentroid(key);
                Point2D bl = grid.getPixelBl(key);
                Point2D br = grid.getPixelBr(key);
                Point2D tr = grid.getPixelTr(key);
                Point2D tl = grid.getPixelTl(key);
                int x = grid.getX(key);
                int y = grid.getX(key);
```

```
                PreparedStatement total = m_Connection.prepareStatement("REPLACE
                     INTO "+ state + " VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?)");

                //ID
                total.setDouble(1, key);

                //X/Y
                total.setInt(2, x);
                total.setInt(3, y);

                //Centroid Long/Lat
                total.setDouble(4, centroid.x());
                total.setDouble(5, centroid.y());

                //Bottom Left Long/Lat
                total.setDouble(6, bl.x());
                total.setDouble(7, bl.y());

                //Bottom Right Long/Lat
                total.setDouble(8, br.x());
                total.setDouble(9, br.y());

                //Top Left Long/Lat
                total.setDouble(10, tl.x());
                total.setDouble(11, tl.y());

                //Top Right Long/Lat
                total.setDouble(12, tr.x());
                total.setDouble(13, tr.y());

                total.executeUpdate();
            }
        }
        m_Connection.close();
    }
}
```

./Code/CreateSqlTables.java

TestFrame.java

```
/**
 *  Created by: Tharald S. Fongaard & Jacob Perricone
 *  Advisor: Alain Kornhauser
 *  Date: 05/05/2015
 *
 *  Part Of Project: ORF 376 - Independent Work
 *
 *  Princeton University    2015
 */

import org.openstreetmap.gui.jmapviewer.*;
import org.openstreetmap.gui.jmapviewer.events.JMVCommandEvent;
import org.openstreetmap.gui.jmapviewer.interfaces.*;
import org.openstreetmap.gui.jmapviewer.tilesources.BingAerialTileSource;
import org.openstreetmap.gui.jmapviewer.tilesources.MapQuestOpenAerialTileSource;
import org.openstreetmap.gui.jmapviewer.tilesources.MapQuestOsmTileSource;
import org.openstreetmap.gui.jmapviewer.tilesources.OsmTileSource;

import java.awt.*;
import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.IOException;
import java.sql.*;
import java.util.*;
import java.util.List;

import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class TestFrame extends JFrame implements JMapViewerEventListener {

    // Protected Private Objects
    private static final long serialVersionUID = 1L;
    private JMapViewerTree treeMap = null;
    private JLabel zoomLabel=null;
    private JLabel zoomValue=null;
    private JLabel mperpLabelName=null;
    private JLabel mperpLabelValue = null;

    // Constructor
    public TestFrame() {

        super("TrafficFlow");
        XMLReader xmlreader = new XMLReader();
        setSize(400, 400);

        treeMap = new JMapViewerTree("Zones");

        // Listen to the map viewer for user operations so components will
        // recieve events and update
        map().addJMVListener(this);

        // Set Layout to fit screen, add state selection button
        setLayout(new BorderLayout());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setExtendedState(JFrame.MAXIMIZED_BOTH);
        JPanel panel = new JPanel();
        JPanel panelTop = new JPanel();
        JPanel panelBottom = new JPanel();
        JPanel helpPanel = new JPanel();
        JPanel sqlPanel = new JPanel();

        // Create Information Pane
        mperpLabelName=new JLabel("Meters/Pixels: ");
        mperpLabelValue=new JLabel(String.format("%s",map().getMeterPerPixel()));
```

```java

        zoomLabel=new JLabel("Zoom: ");
        zoomValue=new JLabel(String.format("%s", map().getZoom()));

        // Add Information Pane
        add(panel, BorderLayout.NORTH);
        add(helpPanel, BorderLayout.SOUTH);
        panel.setLayout(new BorderLayout());
        panel.add(panelTop, BorderLayout.NORTH);
        panel.add(panelBottom, BorderLayout.SOUTH);
        add(sqlPanel,BorderLayout.EAST);
        JLabel helpLabel = new JLabel("Use right mouse button to move,\n "
                + "left double click or mouse wheel to zoom.");
        helpPanel.add(helpLabel);
        JButton button = new JButton("setDisplayToFitMapMarkers");

        // Add all Event Listeners (zoom, select state, layers)
        button.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                map().setDisplayToFitMapMarkers();
            }
        });
        JComboBox<TileSource> tileSourceSelector = new JComboBox<>(new TileSource
             [] { new OsmTileSource.Mapnik(),
                new OsmTileSource.CycleMap(), new BingAerialTileSource(), new
                     MapQuestOsmTileSource(), new MapQuestOpenAerialTileSource()
                     });
        tileSourceSelector.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                map().setTileSource((TileSource) e.getItem());
            }
        });
        JComboBox<TileLoader> tileLoaderSelector;
        try {
            tileLoaderSelector = new JComboBox<>(new TileLoader[] { new
                 OsmFileCacheTileLoader(map()), new OsmTileLoader(map()) });
        } catch (IOException e) {
            tileLoaderSelector = new JComboBox<>(new TileLoader[] { new
                 OsmTileLoader(map()) });
        }
        tileLoaderSelector.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                map().setTileLoader((TileLoader) e.getItem());
            }
        });
        map().setTileLoader((TileLoader) tileLoaderSelector.getSelectedItem());
        panelTop.add(tileSourceSelector);
        panelTop.add(tileLoaderSelector);
        final JCheckBox showMapMarker = new JCheckBox("Map markers visible");
        showMapMarker.setSelected(map().getMapMarkersVisible());
        showMapMarker.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                map().setMapMarkerVisible(showMapMarker.isSelected());
            }
        });
        panelBottom.add(showMapMarker);
        ///
        final JCheckBox showTreeLayers = new JCheckBox("Tree Layers visible");
        showTreeLayers.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                treeMap.setTreeVisible(showTreeLayers.isSelected());
            }
        });
        panelBottom.add(showTreeLayers);
        ///
        final JCheckBox showToolTip = new JCheckBox("ToolTip visible");
        showToolTip.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                map().setToolTipText(null);
            }
        });
        panelBottom.add(showToolTip);
        ///
        final JCheckBox showTileGrid = new JCheckBox("Tile grid visible");
        showTileGrid.setSelected(map().isTileGridVisible());
        showTileGrid.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                map().setTileGridVisible(showTileGrid.isSelected());
            }
        });
        panelBottom.add(showTileGrid);
        final JCheckBox showZoomControls = new JCheckBox("Show zoom controls");
        showZoomControls.setSelected(map().getZoomContolsVisible());
        showZoomControls.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                map().setZoomContolsVisible(showZoomControls.isSelected());
            }
        });
        panelBottom.add(showZoomControls);
        final JCheckBox scrollWrapEnabled = new JCheckBox("Scrollwrap enabled");
        scrollWrapEnabled.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                map().setScrollWrapEnabled(scrollWrapEnabled.isSelected());
            }
        });

        // Add Scrolling Functionality
        panelBottom.add(scrollWrapEnabled);
        panelBottom.add(button);

        // Add Zooming Functionality
        panelTop.add(zoomLabel);
        panelTop.add(zoomValue);
        panelTop.add(mperpLabelName);
        panelTop.add(mperpLabelValue);

        // Add Layer Functionality
        add(treeMap, BorderLayout.CENTER);

        // Add States Layers
        LayerGroup pixelGroup = new LayerGroup("States");
        pixelGroup.setVisible(true);
        SeparateChainingHashST<String, Layer>  states = new
             SeparateChainingHashST<String,Layer>();

        // All States Layers
        String[] options = new String[51];
        int i = 0;
        for(String s: xmlreader.getAllStates()) {
            Layer layer = pixelGroup.addLayer(s);
           states.put(s,layer);
```

```
            Polygon pol = xmlreader.getStatePolygon(s);

            List<Coordinate> list = new ArrayList<Coordinate>();
            for(Point2D p: pol.getAll()){
                list.add(c(p));
            }
            MapPolygon thisState = new MapPolygonImpl(layer,s,list);
            map().addMapPolygon(thisState);
            treeMap.addLayer(layer);
            options[i] = s;
            i++;
        }
        // Combobox for selecting states
        JComboBox jComboBox = new JComboBox(options);
        sqlPanel.add(jComboBox);

        // Load pixels for given state from Grid Database
        JButton getPixels = new JButton();
        getPixels.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                try {
                    Class.forName("com.mysql.jdbc.Driver");
                } catch (ClassNotFoundException e1) {
                    e1.printStackTrace();
                }
                String url = "jdbc:mysql://127.0.0.1:3306/Grid";
                try {
                    Connection m_Connection = DriverManager.getConnection(url, "
                         tharald", "putin");
                    PreparedStatement total = m_Connection.prepareStatement("
                         SELECT * FROM " + jComboBox.getSelectedItem());
                    ResultSet resultSet = total.executeQuery();
                    while (resultSet.next()) {
                        double bl_long = resultSet.getDouble(6);
                        double bl_lat = resultSet.getDouble(7);
                        double br_long = resultSet.getDouble(8);
                        double br_lat = resultSet.getDouble(9);
                        double tl_long = resultSet.getDouble(10);
                        double tl_lat = resultSet.getDouble(11);
                        double tr_long = resultSet.getDouble(12);
                        double tr_lat = resultSet.getDouble(13);
                        String state = (String) jComboBox.getSelectedItem();
                        List<Coordinate> points = new ArrayList<Coordinate>();
                        points.add(c(bl_lat, bl_long));
                        points.add(c(br_lat, br_long));
                        points.add(c(tr_lat, tr_long));
                        points.add(c(tl_lat, tl_long));

                        MapPolygonImpl pix = new MapPolygonImpl(states.get(state)
                             , points);
                        map().addMapPolygon(pix);
                    }

                } catch (SQLException e1) {
                    e1.printStackTrace();
                }
            }
        });

        // Add all pixels fetched
        sqlPanel.add(getPixels);

        // Add Mouse Capabilities
        map().addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (e.getButton() == MouseEvent.BUTTON1) {
                    map().getAttribution().handleAttribution(e.getPoint(), true);
                }
            }
        });

        map().addMouseMotionListener(new MouseAdapter() {
            @Override
            public void mouseMoved(MouseEvent e) {
                Point p = e.getPoint();
                boolean cursorHand = map().getAttribution().
                     handleAttributionCursor(p);
                if (cursorHand) {
                    map().setCursor(new Cursor(Cursor.HAND_CURSOR));
                } else {
                    map().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
                }
                if(showToolTip.isSelected()) map().setToolTipText(map().
                     getPosition(p).toString());
            }
        });
    }

    @Override
    public void processCommand(JMVCommandEvent command) {
        if (command.getCommand().equals(JMVCommandEvent.COMMAND.ZOOM) ||
                command.getCommand().equals(JMVCommandEvent.COMMAND.MOVE)) {
            updateZoomParameters();
        }
    }

    // Update Zoom Level on Select
    private void updateZoomParameters() {
        if (mperpLabelValue!=null)
            mperpLabelValue.setText(String.format("%s",map().getMeterPerPixel()))
                ;
        if (zoomValue!=null)
            zoomValue.setText(String.format("%s", map().getZoom()));
    }

    // Private Constructor Helper Functions
    private JMapViewer map(){
        return treeMap.getViewer();
    }
    private static Coordinate c(double lat, double lon){
        return new Coordinate(lat, lon);
    }
    private static Coordinate c(Point2D p){
        return new Coordinate(p.y(),p.x());
    }

    // Main Execution
    public static void main(String[] args) {
        // java.util.Properties systemProperties = System.getProperties();
        // systemProperties.setProperty("http.proxyHost", "localhost");
```

```
        // systemProperties.setProperty("http.proxyPort", "8008");
        new TestFrame().setVisible(true);
    }
}
```

./Code/TestFrame.java

B.1.2 oTrip Generation

TripGenerator.java

```
/**
 *  Created by: Tharald S. Fongaard & Jacob Perricone
 *  Advisor: Alain Kornhauser
 *  Date: 05/05/2015
 *
 *  Part Of Project: ORF 376 - Independent Work
 *
 *  Princeton University    2015
 */

import java.util.*;
import java.io.*;
import java.io.FileNotFoundException;
import java.sql.*;

public class TripGenerator  {

    // Constructor
    public TripGenerator(File filename) throws ClassNotFoundException,
         SQLException {
        generateTrips(filename);
    }

    // Private Helper Function, Creates oTrips From CSV file
    private void generateTrips(File filename) throws ClassNotFoundException,
         SQLException {
        BufferedReader reader = null;
        String line = "";
        String csvSplitBy = ",";
        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://127.0.0.1:3306/Global";
        Connection m_Connection = DriverManager.getConnection(url, "tharald", "
             putin");
        try{
            reader = new BufferedReader(new FileReader(filename));
            reader.readLine();
            while((line = reader.readLine()) != null) {
                double atime;
                String[] info = line.split(csvSplitBy);
                double person_id = Long.parseLong(info[5]);
                if(Double.parseDouble(info[12]) == 0.0 || Double.parseDouble(info
                     [13])==0){
                    continue;
                }
                int k = 0;
                for (int i = 1; i <= 7;i++) {
                    StdOut.println(i);
                    k++;
                    boolean skip = false;
                    int j = i;
                    int index = (i-1)*9 + 12;
                    String oType = info[index - 5];
                    String dType = info[index - 3];
                    if(oType.equals("NA") || dType.equals("N")){
                        break;
                    }
                    if(info[index - 5].equals("NA") || info[index - 3].equals("N"
                         )){
                        break;
                    }
                    double unique_id = createId(person_id, k);
                    double otime = Double.parseDouble(info[index + 2]);
                    double dtime = Double.parseDouble(info[index + 3]);
                    double olat = Double.parseDouble(info[index]);
                    double olon = Double.parseDouble(info[index + 1]);
                    int ox_pixel = get_x(olat, olon);
                    int oy_pixel = get_y(olat, olon);
                    int dx_pixel;
                    int dy_pixel;
                    double dlat;
                    double dlon;
                    if(i==7){
                        atime = Double.parseDouble(info[index + 3]);
                        dlat  = Double.parseDouble(info[12]);
                        dlon = Double.parseDouble(info[13]);
                        dx_pixel = get_x(dlat,dlon);
                        dy_pixel = get_y(dlat,dlon);
                    }else {
                        atime = Double.parseDouble(info[index + 11]);
                         dlat = Double.parseDouble(info[index + 9]);
                        dlon = Double.parseDouble(info[index + 10]);
                        if(dlat == 0.0 || dlon == 0.0) {
                            for(j = i + 1; (dlat == 0.0 || dlon == 0.0)  && j <
                                 7; j++) {
                                int tempindex = (j - 1) * 9 + 12;
                                if(info[tempindex - 5].equals("NA") || info[
                                     tempindex - 3].equals("N")){
                                    skip = true;
                                    break;
                                }
                                atime = Double.parseDouble(info[tempindex +
                                     11]);
                                dlat = Double.parseDouble(info[tempindex +
                                     9]);
                                dlon = Double.parseDouble(info[tempindex +
                                     10]);
                                dType = info[tempindex - 3];
                                StdOut.println(dlat + " " + dlon + " " +
                                     dType + "first" + j + "i: " + i);
                            }
                            if(skip){
                                break;
                            }
                            if(dlat == 0.0 || dlon == 0.0) {
```

```
                                dType = "H";
                                atime = 0;
                                dlat = Double.parseDouble(info[12]);
                                dlon = Double.parseDouble(info[13]);
                            }
                            i = j-1;
                        }
                        StdOut.println(dlat + " " + dlon + " " + dType + j);
                        dx_pixel = get_x(dlat, dlon);
                        dy_pixel = get_y(dlat, dlon);
                    }

                    double oId = fetchPixelId(ox_pixel, oy_pixel);
                    double aId = fetchPixelId(dx_pixel, dy_pixel);

                    String s = String.format("SELECT State from Global.Info WHERE
                          ID = %f", oId);
                    StdOut.println(s);
                    StdOut.println(person_id + " " + Double.parseDouble(info[
                          index]) +" " + Double.parseDouble(info[index + 1]));

                    PreparedStatement total = m_Connection.prepareStatement(s);
                    ResultSet result = total.executeQuery();
                    if(!result.next()){
                        continue;
                    }

                    String state = result.getString(1);
                    PreparedStatement insert = m_Connection.prepareStatement("
                          REPLACE INTO oTrip."+ state +" VALUES
                          (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)");
                    insert.setDouble(1, unique_id); insert.setDouble(2, person_id
                          );
                    insert.setInt(3, k); insert.setDouble(4, oId); insert.setInt
                          (5,ox_pixel);
                    insert.setInt(6,oy_pixel); insert.setDouble(7, olat); insert.
                          setDouble(8,olon);
                    insert.setDouble(9, aId); insert.setInt(10, dx_pixel); insert
                          .setInt(11,dy_pixel);
                    insert.setDouble(12,dlat); insert.setDouble(13,dlon);
                    insert.setDouble(14, otime);  insert.setDouble(15, dtime);
                    insert.setDouble(16, atime);
                    insert.setString(17, oType); insert.setString(18, dType);
                    insert.executeUpdate();
                    /*
                    System.out.printf("Trip Number: %d, Person_id: %f, Unique Id:
                          %f, oPixelid: %f, aPixelid: %f, oTime: %f, dTime: %f,
                          aTime: %f, otype: %s, dType: %s \n",
                            i, person_id, unique_id, oId, aId, otime, dtime,
                                atime, oType, dType);
                    */
                }
            }
        }
        catch(FileNotFoundException e){
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Combine Person ID and trip segment to create unique Trip ID
    private double createId(double pid, double trip) {
        double id = pid + (trip/10);
        return id;
    }

    // Get oPixel given coordinates a and b
    private double fetchPixelId(int a, int b) {
        double pixelId = 0.5*(a+195593+b+1724)*(a+195593+b+1724+1)+b+1724;
        // StdOut.println(returnn);
        return pixelId;
    }

    // Get x coordiate given latitide and longitude of oTrip
    private int get_x(double lat, double lon){
        int x = (int) Math.floor(138.348 * (lon + 97.5) * Math.cos(Math.toRadians
              (lat)));
        return x;
    }

    // Get y coordiate given latitide and longitude of oTrip
    private int get_y(double lat, double lon){
        int y = (int) Math.floor(138.348 * (lat - 37.0));
        return y;
    }

    // Main Execution, Fills in oTrip database given NN file in csv
    public static void main(String[] args) throws SQLException,
          ClassNotFoundException {
        String filePath = "/Users/jacobperricone/Desktop/triptest.csv";
        File inputFile = new File(filePath);
        TripGenerator gen = new TripGenerator(inputFile.getAbsoluteFile());
    }
}
```

./Code/TripGenerator.java

CreateTable.java

```
import java.sql.*;
import java.lang.*;

/**
 *  Created by: Tharald S. Fongaard & Jacob Perricone
 *  Advisor: Alain Kornhauser
 *  Date: 05/05/2015
 *
 *  Part Of Project: ORF 376 - Independent Work
 *
 *  Princeton University    2015
 */
public class CreateTable {

    // Constructor
    public CreateTable() throws ClassNotFoundException, SQLException {
```

```java
        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://127.0.0.1:3306/Grid";
        Connection m_Connection = DriverManager.getConnection(url, "tharald", "
             putin");
        PreparedStatement total = m_Connection.prepareStatement("SELECT
             TABLE_NAME from information_schema.TABLES WHERE TABLE_SCHEMA LIKE ('
             Grid');");
        ResultSet result = total.executeQuery();

        while (result.next()){
            String state = result.getString(1);
            System.out.println(state);
            PreparedStatement get = m_Connection.prepareStatement("SELECT ID FROM
                  Grid." + state +";");
            ResultSet ids = get.executeQuery();
            while (ids.next()){
                double id = ids.getDouble(1);
                PreparedStatement insert = m_Connection.prepareStatement("REPLACE
                      INTO Global.Info VALUES(?,?)");
                insert.setDouble(1, id);
                insert.setString(2, state);
                insert.executeUpdate();
            }
        }
        m_Connection.close();
    }

    // Main Exectuiton, Creates Table Global.Info
    public static void main(String[] args){
        try {
            CreateTable go = new CreateTable();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }

    }
}
```

./Code/CreateTable.java

B.1.3 Path File Generation

PathFinderjava

```java

import java.sql.*;

/**
 * Created by tharald on 30/04/15.
 */
public class PathFinder {

    private LinkObject linkObject;

    public PathFinder(int from, int to) throws SQLException,
         ClassNotFoundException {
        linkObject = new LinkObject();
        readNodeTable(from, to);
    }

    private void readNodeTable(int from, int to) throws ClassNotFoundException,
         SQLException {
        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://127.0.0.1:3306/Global";
        String urlTrip = "jdbc:mysql://127.0.0.1:3306/oTrip";
        String pathFinder = "jdbc:mysql://127.0.0.1:3306/Path";
        Connection m_Connection = DriverManager.getConnection(url, "tharald", "
             putin");
        Connection trip_Connection = DriverManager.getConnection(urlTrip, "
             tharald", "putin");
        Connection path_connetction = DriverManager.getConnection(pathFinder, "
             tharald", "putin");

        PreparedStatement total = m_Connection.prepareStatement("SELECT Pixel,
             EntryNode FROM EntryNodes WHERE (EntryNode > "+from+" AND EntryNode
             < "+to+") order by EntryNode, Pixel");
        ResultSet resultSet = total.executeQuery();

        // Find Paths
        // Create empty linkDij
        LinkDij linkDij = new LinkDij();
        int prevEntryNode = -1;
        //StdOut.println("ID \t PersonID \t Segment \t aNode \t bNode \t LinkID \
             t LinkName \t oPixel \t dPixel \t dTime \t aTime \t oType \t dType \
             t a_lat \t a_Long \t b_Lat \t b_Long \t aNode_Time \t bNode_Time");
        while (resultSet.next()){
            double pixel = resultSet.getDouble(1);
            int entryNode = resultSet.getInt(2);
            if(entryNode != prevEntryNode){
                linkDij = getDijkstra(entryNode);
            }
            prevEntryNode = entryNode;

            // Find State
            PreparedStatement getState = m_Connection.prepareStatement("SELECT
                 State FROM Info WHERE (ID="+pixel+")");
            ResultSet resultState = getState.executeQuery();
            if(!resultState.next()){
                resultState.close();
                getState.close();
                StdOut.printf(" NO STATE FOUND FOR PIXEL: %.2f", pixel);
                StdOut.println();
                continue;
            }
            String state = resultState.getString(1);
            resultState.close();
            getState.close();

            //Find trips
            PreparedStatement getTrips = trip_Connection.prepareStatement("SELECT
                 ID, PersonID,oPixel,dPixel, dTime, aTime,oType, dType, oLat,
                oLong FROM "+ state + " Where (oPixel = "+ pixel +")");
            ResultSet resultTrips = getTrips.executeQuery();
```

```
            if(!resultTrips.next()){
                resultTrips.close();
                getTrips.close();
                StdOut.printf(" NO TRIP FOUND FOR STATE: " + state + "PIXEL: %.2f
                     ", pixel);
                StdOut.println();
                continue;
            }
            while (resultTrips.next()){
                double tripID = resultTrips.getDouble(1);
                double personID = resultTrips.getDouble(2);
                double oPixel = resultTrips.getDouble(3);
                double dPixel = resultTrips.getDouble(4);
                double dTime = resultTrips.getDouble(5);
                double aTime = resultTrips.getDouble(6);
                String oType = resultTrips.getString(7);
                String dType = resultTrips.getString(8);
                double oLat = resultTrips.getDouble(9);
                double oLong = resultTrips.getDouble(10);

                if(linkObject.getNode(dPixel) == null){
                    StdOut.println("NO Node FOUND FOR PIXEL: " + dPixel);
                    continue;
                }
                int destNode = linkObject.getNode(dPixel);

                Iterable<DirectedEdge> path = linkDij.pathTo(destNode);

                if (path == null){
                    StdOut.print("NO PATH FOUND FOR: ");
                    StdOut.printf("Trip: %.1f oPix: %.1f dPixel %.1f ", tripID,
                         oPixel, dPixel);
                    StdOut.println("from: " + entryNode+ " to: " + destNode);
                    continue;
                }

                int segment = 0;
                double aNode_Time = dTime;

                PreparedStatement pathInsert = path_connetction.prepareStatement(
                     "REPLACE INTO " + state + " VALUES
                     (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)");

                for(DirectedEdge e: path) {

                    if (segment == 0) {
                        double a_lat = oLat;
                        double a_long = oLong;
                        double b_lat = linkObject.getPoint(e.from()).y();
                        double b_long = linkObject.getPoint(e.from()).x();
                        int a_node = -1;
                        int  b_node = e.from();
                        int linkId = linkObject.getLinkId(e);
                        String linkName = linkObject.getLinkName(e);

                        PreparedStatement pathInsert0 = path_connetction.
                             prepareStatement("REPLACE INTO " + state + " VALUES
                             (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)");
                        pathInsert0.setDouble(1, tripID);
                        pathInsert0.setDouble(2, personID);
                        pathInsert0.setInt(3, segment);
                        pathInsert0.setInt(4, a_node);
                        pathInsert0.setInt(5, b_node);
                        pathInsert0.setInt(6, linkId);
                        pathInsert0.setString(7, linkName);
                        pathInsert0.setDouble(8, oPixel);
                        pathInsert0.setDouble(9, dPixel);
                        pathInsert0.setDouble(10, dTime);
                        pathInsert0.setDouble(11, aTime);
                        pathInsert0.setString(12, oType);
                        pathInsert0.setString(13, dType);
                        pathInsert0.setDouble(14, a_lat);
                        pathInsert0.setDouble(15, a_long);
                        pathInsert0.setDouble(16, b_lat);
                        pathInsert0.setDouble(17, b_long);
                        pathInsert0.setDouble(18, aNode_Time);
                        pathInsert0.setDouble(19, aNode_Time);
                        pathInsert0.executeUpdate();
                        segment++;
                        pathInsert0.close();

                    }

                    int linkId = linkObject.getLinkId(e);
                    String linkName = linkObject.getLinkName(e);
                    double bNode_Time = aNode_Time + e.weight();
                    double a_lat = linkObject.getPoint(e.from()).y();
                    double a_long = linkObject.getPoint(e.from()).x();
                    double b_lat = linkObject.getPoint(e.to()).y();
                    double b_long = linkObject.getPoint(e.to()).x();

                    pathInsert.setDouble(1, tripID);
                    pathInsert.setDouble(2, personID);
                    pathInsert.setInt(3, segment);
                    pathInsert.setInt(4, e.from());
                    pathInsert.setInt(5, e.to());
                    pathInsert.setInt(6, linkId);
                    pathInsert.setString(7, linkName);
                    pathInsert.setDouble(8, oPixel);
                    pathInsert.setDouble(9, dPixel);
                    pathInsert.setDouble(10, dTime);
                    pathInsert.setDouble(11, aTime);
                    pathInsert.setString(12, oType);
                    pathInsert.setString(13, dType);
                    pathInsert.setDouble(14, a_lat);
                    pathInsert.setDouble(15, a_long);
                    pathInsert.setDouble(16, b_lat);
                    pathInsert.setDouble(17, b_long);
                    pathInsert.setDouble(18, aNode_Time);
                    pathInsert.setDouble(19, bNode_Time);
                    pathInsert.executeUpdate();
                    aNode_Time = bNode_Time;
                    segment++;
                    pathInsert.clearParameters();
                }
                pathInsert.close();
            }
            resultTrips.close();
            getTrips.close();
        }
        resultSet.close();
```

```java
        total.close();

        m_Connection.close();
        trip_Connection.close();
        path_connetction.close();
    }

    private LinkDij getDijkstra(int source){
        Point2D focal = linkObject.getPoint(source);
        Point2D NE = getBearing(focal, 45.0);
        Point2D SW = getBearing(focal, 230.0);
        //StdOut.println("Square: Center: " + focal.toString()+ " NE: "+NE.toString()+ " SW: "+ SW.toString());
        double xMin = SW.x();
        double yMin = SW.y();
        double xMax = NE.x();
        double yMax = NE.y();
        Iterable<DirectedEdge> edges = linkObject.getEdges(xMin, yMin, xMax, yMax);

        LinkDij linkDij = new LinkDij(edges,source);
        return linkDij;
    }

    private Point2D getBearing(Point2D focal, double bearingDeg){
        double focalLat = focal.y();
        double focalLong = focal.x();
        focalLat = Math.toRadians(focalLat);
        focalLong = Math.toRadians(focalLong);
        double bearing = Math.toRadians(bearingDeg);
        double ratio = 250.0/3963.1676;

        double newLat = Math.asin(Math.sin(focalLat) * Math.cos(ratio) + Math.cos(focalLat) * Math.sin(ratio) * Math.cos(bearing));
        double newLong = focalLong + Math.atan2(Math.sin(bearing)*Math.sin(ratio)*Math.cos(focalLat), Math.cos(ratio)-Math.sin(focalLat)*Math.sin(newLat));

        newLat = Math.toDegrees(newLat);
        newLong = Math.toDegrees(newLong);

        Point2D p = new Point2D(newLong,newLat);
        return p;
    }

    public static void main(String args[]) throws SQLException, ClassNotFoundException {
        int from = Integer.parseInt(args[0]);
        int to = Integer.parseInt(args[1]);
        new PathFinder(from, to);
    }
}
```

./Code/PathFinder.java

FindEntryNodes.java

```java
import java.sql.*;

/**
 *  Created by: Tharald S. Fongaard & Jacob Perricone
 *  Advisor: Alain Kornhauser
 *  Date: 05/05/2015
 *
 *  Part Of Project: ORF 376 - Independent Work
 *
 *  Princeton University    2015
 */
public class FindEntryNodes {
    // Protected Private Objects
    private KdTreeST<Integer> nodes;
    private XMLReader xmlReader;

    // Constructor
    public FindEntryNodes() throws ClassNotFoundException, SQLException {
        nodes = new KdTreeST<Integer>();
        xmlReader = new XMLReader();

        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://127.0.0.1:3306/Global";
        Connection m_Connection = DriverManager.getConnection(url, "tharald", "putin");

        PreparedStatement total = m_Connection.prepareStatement("SELECT * FROM Nodes");
        ResultSet resultSet = total.executeQuery();

        while(resultSet.next()){
            int id = resultSet.getInt(1);
            double lat = resultSet.getDouble(2);
            double lng = resultSet.getDouble(3);
            Point2D point = new Point2D(lng,lat);
            nodes.insert(point,id);
        }
        resultSet.close();
        total.close();
        System.out.println("Created KdTree");
    }

    // Get all states in static XML dataset associated with project
    public Iterable<String> getAllStates(){
        return xmlReader.getAllStates();
    }

    // Get closest node given closest point p
    public Integer iFindClosest(Point2D q){
        return nodes.get(q);
    }

    // Find closest node given query point q
    public Point2D pFindClosest(Point2D p){
        return nodes.nearest(p);
    }

    // Test Execution
```

```java
    public static void main(String args[]) throws ClassNotFoundException, SQLException{
        FindEntryNodes entryNodes = new FindEntryNodes();

        for(String state : entryNodes.getAllStates()){
            if(state.equals("NoState")){
                System.out.println("State finished:" + state);
                continue;
            }
            Class.forName("com.mysql.jdbc.Driver");
            String url = "jdbc:mysql://127.0.0.1:3306";
            String urlGlobal = "jdbc:mysql://127.0.0.1:3306/Global";
            Connection m_Connection = DriverManager.getConnection(url, "tharald", "putin");
            Connection c_Connection = DriverManager.getConnection(urlGlobal, "tharald", "putin");
            PreparedStatement total = m_Connection.prepareStatement("SELECT oTrip." + state + ".oPixel, Grid." + state + ".C_Long, Grid." + state + ".C_Lat from oTrip." + state + " LEFT JOIN Grid." + state + " ON oTrip." + state + ".oPixel = Grid." + state + ".ID GROUP BY oPixel");
                ResultSet resultSet = total.executeQuery();

            while(resultSet.next()){
                double id = resultSet.getDouble(1);
                double c_lng = resultSet.getDouble(2);
                double c_lat = resultSet.getDouble(3);
                Point2D qPoint = new Point2D(c_lng,c_lat);
                Point2D mPoint = entryNodes.pFindClosest(qPoint);
                int entryN = entryNodes.iFindClosest(mPoint);
                double m_lng = mPoint.x();
                double m_lat = mPoint.y();
                PreparedStatement insert = c_Connection.prepareStatement("REPLACE INTO EntryNodes VALUES(?,?,?,?,?,?)");
                insert.setDouble(1, id);
                insert.setInt(2, entryN);
                insert.setDouble(3, c_lat);
                insert.setDouble(4, c_lng);
                insert.setDouble(5, m_lat);
                insert.setDouble(6, m_lng);
                insert.execute();
                insert.close();
            }
            m_Connection.close();
            c_Connection.close();
            System.out.println("State finished:" + state);
        }
    }
}
```

./Code/FindEntryNodes.java

NodeObject.java

```java
import java.sql.*;

/**
 *  Created by: Tharald S. Fongaard & Jacob Perricone
 *  Advisor: Alain Kornhauser
 *  Date: 05/05/2015
 *
 *  Part Of Project: ORF 376 - Independent Work
 *
 *  Princeton University    2015
 */
public class NodeObject {
    // Protected Private Objects
    private KdTreeST<Integer> nodes;
    private SeparateChainingHashST<Integer,Point2D> nodesPoints;
    private SeparateChainingHashST<Double, Integer> pixelPoints;

    // Constructor
    public NodeObject() throws SQLException, ClassNotFoundException {
        nodes = new KdTreeST<Integer>();
        nodesPoints = new SeparateChainingHashST<Integer,Point2D>();
        pixelPoints = new SeparateChainingHashST<Double,Integer>();
        readNodes();
    }

    // Private helper method to connect to Node Database and store links memory
    private void readNodes() throws ClassNotFoundException, SQLException {
        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://127.0.0.1:3306/Global";
        Connection m_Connection = DriverManager.getConnection(url, "tharald", "putin");

        // Get Nodes
        PreparedStatement total = m_Connection.prepareStatement("SELECT * FROM Nodes");
        ResultSet resultSet = total.executeQuery();

        while(resultSet.next()){
            int id = resultSet.getInt(1);
            double lat = resultSet.getDouble(2);
            double lng = resultSet.getDouble(3);
            Point2D point = new Point2D(lng,lat);
            nodes.insert(point,id);
            nodesPoints.put(id,point);
        }
        resultSet.close();
        total.close();

        // Get entrynodes
        PreparedStatement eNodes = m_Connection.prepareStatement("SELECT Pixel, EntryNode FROM EntryNodes");
        ResultSet resultNodes = eNodes.executeQuery();

        while(resultNodes.next()){
            double pixel = resultNodes.getDouble(1);
            int node = resultNodes.getInt(2);
            pixelPoints.put(pixel,node);
        }

        resultNodes.close();
        eNodes.close();
        m_Connection.close();
```

```java
    }

    // Get geo-location of node
    public Point2D getPoint(Integer node){
        return nodesPoints.get(node);
    }

    // Get node ID of point
    public Integer getNode(Point2D point){
        return nodes.get(point);
    }

    // Get node ID associated with pixel
    public Integer getNode(double pixel){
        return pixelPoints.get(pixel);
    }
}
```

./Code/NodeObject.java

LinkObject.java

```java
import java.sql.*;
import java.util.*;

/**
 * Created by tharald on 01/05/15.
 */
public class LinkObject {
    // Protected Private Objects
    private NodeObject nodeObject;
    private KdTreeST<Bag<DirectedEdge>> links;
    private SeparateChainingHashST<DirectedEdge, Integer> edgeToLink;
    private SeparateChainingHashST<DirectedEdge,String> edgeToName;

    // Constructor
    public LinkObject() throws SQLException, ClassNotFoundException {
        links = new KdTreeST<Bag<DirectedEdge>>();
        nodeObject = new NodeObject();
        edgeToLink = new SeparateChainingHashST<DirectedEdge,Integer>();
        edgeToName = new SeparateChainingHashST<DirectedEdge,String>();
        readLinks();
    }

    // Private helper method to connect to Link Database and store links memory
    private void readLinks() throws ClassNotFoundException, SQLException {
        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://127.0.0.1:3306/Global";
        Connection m_Connection = DriverManager.getConnection(url, "tharald", "
            putin");

        PreparedStatement total = m_Connection.prepareStatement("SELECT * FROM
            Links");
        ResultSet resultSet = total.executeQuery();

        while (resultSet.next()) {
            int id = resultSet.getInt(1);
            int nodeA = resultSet.getInt(2);
            int nodeB = resultSet.getInt(3);
            double distance = resultSet.getDouble(4);
            int roadType = resultSet.getInt(5);
            int nodeType = resultSet.getInt(6);
            String name = resultSet.getString(7);

            double weight;
            if (roadType == 1) {
                weight = (distance / 65) * 3600;
            } else if (roadType == 2) {
                weight = (distance / 65) * 3600;
            } else if (roadType == 3) {
                weight = (distance / 55) * 3600;
            } else if (roadType == 4) {
                weight = (distance / 40) * 3600;
            } else if (roadType == 5) {
                weight = (distance / 5) * 3600;
            } else if (roadType == 6) {
                weight = (distance / 30) * 3600;
            } else if (roadType == 7) {
                weight = (distance / 35) * 3600;
            } else {
                weight = (distance / 20) * 3600;
            }

            if (nodeType == 1) {
                DirectedEdge edgeA = new DirectedEdge(nodeA, nodeB, weight);
                DirectedEdge edgeB = new DirectedEdge(nodeB, nodeA, weight);
                addLink(edgeA, id, name);
                addLink(edgeB, id, name);

            } else if (nodeType == 2) {
                DirectedEdge edge = new DirectedEdge(nodeB, nodeA, weight);
                addLink(edge, id, name);
            } else if (nodeType == 3) {
                DirectedEdge edgeA = new DirectedEdge(nodeA, nodeB, weight);
                DirectedEdge edgeB = new DirectedEdge(nodeB, nodeA, weight);
                addLink(edgeA, id, name);
                addLink(edgeB, id, name);
            }
        }
        resultSet.close();
        total.close();
        m_Connection.close();
    }

    // Get all links within query rectangle xmin,xmax,ymin,ymax
    public Iterable<DirectedEdge> getEdges(double xmin, double ymin, double xmax,
        double ymax){
        //StdOut.println("xmin: " + xmin + " ymin: " + ymin + " xmax:" + xmax + "
            ymax: " + ymax);
        RectHV rectHV = new RectHV(xmin, ymin, xmax, ymax);
        Queue<DirectedEdge> directedEdgeQueue = new Queue<DirectedEdge>();
         for (Point2D p: links.range(rectHV)){
             Bag<DirectedEdge> bag = links.get(p);
             for(DirectedEdge e: bag){
                 directedEdgeQueue.enqueue(e);
             }
```

```
        }
        return directedEdgeQueue;
    }

    // Add link to datase, check for duplicates
    private void addLink(DirectedEdge e, int linkID, String linkName){
        if (hasEdge(e) == false){
            int aNode = e.from();

            Point2D p = nodeObject.getPoint(aNode);

            if(links.contains(p)){
                Bag<DirectedEdge> edgebag = links.get(p);
                edgebag.add(e);
                links.insert(p,edgebag);
            }
            else{
                Bag<DirectedEdge> edgebag = new Bag<DirectedEdge>();
                edgebag.add(e);
                links.insert(p,edgebag);
            }
            edgeToLink.put(e,linkID);
            edgeToName.put(e,linkName);
        }
    }

    // Does the data structure already contain the link?
    private boolean hasEdge(DirectedEdge e){
        if (edgeToLink.contains(e)){
            return true;
        }
        else {
            return false;
        }

    }

    // Get geo-location of node
    public Point2D getPoint(int node){
        return nodeObject.getPoint(node);
    }

    // Get id of node associated with pixel
    public Integer getNode(double pixel){
        return nodeObject.getNode(pixel);
    }

    // Get Link ID of edge object e
    public Integer getLinkId(DirectedEdge e){
        return edgeToLink.get(e);
    }

    // Get Link Name of edge object e
    public String getLinkName(DirectedEdge e){
        return edgeToName.get(e);
    }

    // Test Execution
    public static void main(String args[]) throws SQLException,
         ClassNotFoundException {
        LinkObject linkObject = new LinkObject();
    }
}
```

./Code/LinkObject.java

LinkDij.java

```
/**
 *  Created by: Tharald S. Fongaard & Jacob Perricone
 *  Advisor: Alain Kornhauser
 *  Date: 05/05/2015
 *
 *  Part Of Project: ORF 376 - Independent Work
 *
 *  Princeton University    2015
 */
public class LinkDij {
    // Protected Private Objects
    private EdgeWeightedDigraph digraph;
    private SET<Integer> nodes;
    private DijkstraSP dijkstraSP;

    // Empty Overloaded Constructor
    public LinkDij(){
    }

    // Constructor
    public LinkDij(Iterable<DirectedEdge> iterable, Integer source){
        nodes = new SET<Integer>();
        //497713 is the max N of node possible
        digraph = new EdgeWeightedDigraph(497713);
        for (DirectedEdge e: iterable){
            digraph.addEdge(e);
        }
        dijkstraSP = new DijkstraSP(digraph, source);
    }

    // Return Path Destination Node
    public Iterable<DirectedEdge> pathTo(int dest){
        return dijkstraSP.pathTo(dest);
    }

    // Return Distance To Destination Node
    public  double distTo(int dest){
        return dijkstraSP.distTo(dest);
    }
}
```

./Code/LinkDij.java

B.1.4 Data Visualization

CreateJsonFiles.java

```java
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.PreparedStatement;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 *  Created by: Tharald S. Fongaard & Jacob Perricone
 *  Advisor: Alain Kornhauser
 *  Date: 05/05/2015
 *
 *  Part Of Project: ORF 376 - Independent Work
 *
 *  Princeton University    2015
 */
public class CreateJsonFiles {
    private XMLReader xmlReader;

    //Constructor
    public CreateJsonFiles(){
        //initialize XML reader
        xmlReader = new XMLReader();
    }

    //Main
    public static void main(String[] args){
        CreateJsonFiles obj = new CreateJsonFiles();
        try {
            obj.createJson();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    //private method that creates the JSON output for globe.js
    private void createJson() throws ClassNotFoundException, SQLException {
        //initialize sql connection
        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://127.0.0.1:3306/oTrip";
        Connection m_Connection = DriverManager.getConnection(url, "tharald", "
             putin");
        double max = 2500000.0;
        System.out.print("[");
        for (int p = 0; p <= 24; p = p + 3) {
            int k = p;
            System.out.print("[" + "\"" +  + "\"" +  "," + "[");
            int statecount = 0;
            for(String state: xmlReader.getAllStates()) {
                //Skip the No State
                if(state.equals("NoState")) {
                    continue;
                }
                int lat = 0;
                int lon = 0; double magnitude = 0;
                statecount++;
                int j = 1;
                PreparedStatement preparedStatement = m_Connection.
                     prepareStatement("select oX, oY, oLat, oLong, count(*)/"+max
                     +" from "+state+" where (dTime < " + (p + )*3600 +" and
                     dTime >= " + (p)*3600+") group by oPixel order by oX, oY" );
                ResultSet resultSet2 = preparedStatement.executeQuery();
                while (resultSet2.next()) {
                    if(j < 2501) {
                        if (j == 1250) {
                            lat = (int)Math.round(resultSet2.getDouble(3));
                            lon = (int)Math.round(resultSet2.getDouble(4));
                        }
                        magnitude += resultSet2.getDouble(5);
                        if(resultSet2.isLast() && statecount == 50) {
                            magnitude = Math.round(magnitude * 1000);
                            magnitude = magnitude/1000;
                            System.out.print(lat + "," + lon + "," + magnitude);
                        }
                        else if (resultSet2.isLast()){
                            magnitude = magnitude/j;
                            magnitude = Math.round(magnitude * 1000);
                            magnitude = magnitude/1000;

                            System.out.print(lat + "," + lon + "," + magnitude+ "
                                 ,");
                        }
                        j++;
                    }
                    else {
                        if(resultSet2.isLast() && statecount == 50) {
                            magnitude = Math.round(magnitude * 1000);
                            magnitude = magnitude/1000;
                            System.out.print(lat + "," + lon + "," + magnitude);
                        }
                        else {
                            magnitude = Math.round(magnitude * 1000);
                            magnitude = magnitude/1000;
                            System.out.print(lat + "," + lon + ","  + magnitude +
                                 ",");
                        }
                        magnitude = 0;
                        magnitude += resultSet2.getDouble(5);
                        lat = (int)Math.round(resultSet2.getDouble(3));
                        lon = (int)Math.round(resultSet2.getDouble(4));
                        j = 2;
                    }
                }
                resultSet2.close();
            }
            if(p != 24)
                System.out.print("]],");
            else{
                System.out.print("]]");
            }
        }
        System.out.print("]");
        m_Connection.close();
    }
}
```

./Code/CreateJsonFiles.java

B.1.5 Database Assist & Data Structures

KdTreeST.java

```
/**
 *  Created by: Tharald S. Fongaard & Alex Kubiak
 *
 *  Assignment For Class: COS 226 - Algorithms and Data Structures
 *
 *  Princeton University    2014
 *
 *  Altered by: Tharald S. Fongaard & Jacob Perricone
 *  Advisor: Alain Kornhauser
 *  Date: 05/05/2015
 *
 *  Part Of Project: ORF 376 - Independent Work
 *
 *  Princeton University    2015
 */
public class KdTreeST<Value>
{
    // flags for red or blue lines
    private static final boolean RED = true;
    private static final boolean BLUE = false;

    private int size; // number of nodes in tree
    private boolean isEmpty; // is the tree empty?
    private Node root; // root node

    // Node data type in a 2d-tree
    private class Node
    {
        private Point2D p; // the point
        private Value value; // the symbol table maps the point to this value
        private RectHV rect; // the axis-aligned rectangle corresponding to this node
        private Node lb; // the left/bottom subtree
        private Node rt; // the right/top subtree
        private boolean color; // true = RED or false = BLUE
    }

    // construct an empty symbol table of points
    public KdTreeST()
    {
        root = new Node();
        // root.p = new Point2D(null, null);
        root.value = null;
        // root.rect = new RectHV(Double.NEGATIVE_INFINITY, Double.NEGATIVE_INFINITY,
        //                        Double.POSITIVE_INFINITY, Double.POSITIVE_INFINITY);
        root.rect = new RectHV(-1000, -1000, 1000, 1000);
        root.lb = null;
        root.rt = null;
        size = 0;
        isEmpty = true;
    }

    // is the symbol table empty?
    public boolean isEmpty()
    {
        return isEmpty;
    }

    // number of points in the ST
    public int size()
    {
        return size;
    }

    // add the point p to the ST or if it already exists, update
    public void insert(Point2D p, Value v)
    {
        // is this the first node to be inserted in the tree?
        if (isEmpty())
        {
            isEmpty = false;
            root.p = p;
            root.value = v;
            root.color = RED;
            size++;
            return;
        }
        // call private helper insert method
        insertR(root, p, v);
    }

    // private helper method to insert new node into ST
    private void insertR(Node prev, Point2D point, Value value)
    {
        // checks if same point
        if (point.compareTo(prev.p) == 0)
        {
            prev.value = value; // update value
            return;
        }

        if (prev.color == RED) // color of parent node is red
        {
            if (xGreaterThan(point, prev.p)) // node will go right
            {
                if (prev.rt == null) // at an empty place in the tree
                {
                    // create the new node
                    prev.rt = makeChild(prev, point, value, true);
                    return;
                }
                insertR(prev.rt, point, value); // recursively insert the node
            }
            else
```

```
            {
                // at an empty place in the tree
                if (prev.lb == null)
                {
                    // create the new node
                    prev.lb = makeChild(prev, point, value, false);
                    return;
                }
                insertR(prev.lb, point, value); // recursively insert the node
            }
        }

        else // color of parent node is blue
        {
            if (yGreaterThan(point, prev.p)) // node will go right
            {
                if (prev.rt == null) // at an empty place in the tree
                {
                    // create the new node
                    prev.rt = makeChild(prev, point, value, true);
                    return;
                }
                insertR(prev.rt, point, value); // recursively insert the node
            }
            else
            {
                // at an empty place in the tree
                if (prev.lb == null)
                {
                    // create the new node
                    prev.lb = makeChild(prev, point, value, false);
                    return;
                }
                insertR(prev.lb, point, value); // recursively insert the node
            }
        }
    }

    // helper function to create a new Node to be inserted
    private Node makeChild(Node prev, Point2D p, Value v, boolean lr)
    {
        Node child = new Node();
        child.p = p;
        child.value = v;
        child.rect = prev.rect;

        if (prev.color == RED) // y-determined coordinates -> x-determined
             rectangle
        {
            if (lr) // greater than; going right
                child.rect = new RectHV(prev.p.x(), prev.rect.ymin(),
                        prev.rect.xmax(), prev.rect.ymax());
            else // less than; going left
                child.rect = new RectHV(prev.rect.xmin(), prev.rect.ymin(),
                        prev.p.x(), prev.rect.ymax());
            child.color = BLUE;
        }
        else // x-determined coordinates -> y-determined rectangle
        {
            if (lr) // greater than; going right
                child.rect = new RectHV(prev.rect.xmin(), prev.p.y(),
                        prev.rect.xmax(), prev.rect.ymax());
            else // less than; going left
                child.rect = new RectHV(prev.rect.xmin(), prev.rect.ymin(),
                        prev.rect.xmax(), prev.p.y());
            child.color = RED;
        }
        child.lb = null;
        child.rt = null;
        size++;
        return child;
    }

    // if point a's x-coord < point b's x-coord, move left; else move right
    private boolean xGreaterThan(Point2D a, Point2D b)
    {
        if (a.x() < b.x()) return false; // go left
        return true; // go right
    }

    // if point a's y-coord < point b's y-coord, move left; else move right
    private boolean yGreaterThan(Point2D a, Point2D b)
    {
        if (a.y() < b.y()) return false; // go left
        return true; // go right
    }

    // returns value mapped to by p
    public Value get(Point2D p)
    {
        if (size == 0) return null;
        // if coordinates are equal, return value
        if ((root.p.x() == p.x()) && (root.p.y() == p.y())) return root.value;

        Node n = root;
        while (n != null)
        {
            // if coordinates are equal, return value
            if ((n.p.x() == p.x()) && (n.p.y() == p.y())) return n.value;
            if (n.color == RED)
            {
                if (xGreaterThan(p, n.p)) n = n.rt; // go right
                else n = n.lb; // go left
            }
            else
            {
                if (yGreaterThan(p, n.p)) n = n.rt; // go right
                else n = n.lb; // go left
            }
        }
        return null;
    }

    // does the ST contain the point p?
    public boolean contains(Point2D p)
    {
        return (get(p) != null);
    }

    // draw points in the unit square to standard draw
    public void draw()
    {
```

```java
        // StdDraw
        StdDraw.setPenColor(StdDraw.BLACK);
        StdDraw.setPenRadius(0.01);
        Node prev = new Node();
        prev = root;

        // iterate through nodes
        for (Node n : iterate())
        {
            n.p.draw();
            StdDraw.setPenRadius();
            if (n.color == RED)
            {
                StdDraw.setPenColor(StdDraw.RED);
                StdDraw.line(n.p.x(), n.rect.ymin(), n.p.x(), n.rect.ymax());
                prev = n;
            }
            else
            {
                StdDraw.setPenColor(StdDraw.BLUE);
                StdDraw.line(n.rect.xmin(), n.p.y(), n.rect.xmax(), n.p.y());
                prev = n;
            }
            StdDraw.setPenColor(StdDraw.BLACK);
            StdDraw.setPenRadius(0.01);
        }
    }

    // all points in the ST that are inside the rectangle
    public Iterable<Point2D> range(RectHV rect)
    {
        // stack to store valid points
        Stack<Point2D> tempstack = new Stack<Point2D>();

        // call recursive helper function
        recursive(root, tempstack, rect);
        return tempstack;
    }

    // helper function to iterate recursively through all points
    private void recursive(Node x, Stack<Point2D> tempstack, RectHV r)
    {
        if (x == null) return; // base case

        if (r.contains(x.p)) tempstack.push(x.p); // point exists in rectangle

        // recurse; travel down/left in the tree
        if ((x.lb != null) && x.lb.rect.intersects(r))
            recursive(x.lb, tempstack, r);

        // recurse; travel down/right in the tree
        if ((x.rt != null) && x.rt.rect.intersects(r))
            recursive(x.rt, tempstack, r);

        return;
    }

    // iterable to iterate through all points in the ST
    private Iterable<Node> iterate()
    {
        // structure to hold valid points
        Stack<Node> tempstack = new Stack<Node>();

        // call recursive helper function
        recurNode(root, tempstack);
        return tempstack;
    }

    public Iterable<Point2D> getAllPoints(){
        Stack<Point2D> returnStack = new Stack<Point2D>();
        for(Node n:iterate()){
            returnStack.push(n.p);
        }
        return returnStack;
    }

    public Iterable<Value> getAllValues(){
        Stack<Value> returnStack = new Stack<Value>();
        for(Node n:iterate()){
            returnStack.push(n.value);
        }
        return returnStack;
    }

    private void recurNode(Node x, Stack<Node> tempstack)
    {
        if (x == null) return; // base case

        // no children, dead end
        if ((x.lb == null) && (x.rt == null))
        {
            tempstack.push(x);
            return;
        }

        // check left child, if exists
        if (x.lb != null)
        {
            tempstack.push(x);
            recurNode(x.lb, tempstack);
        }

        // check right child, if exists
        if (x.rt != null)
        {
            tempstack.push(x);
            recurNode(x.rt, tempstack);
        }
        return;
    }

    // a nearest neighbor in the ST to p; null if set is empty
    public Point2D nearest(Point2D p)
    {
        if (isEmpty) return null; // base case

        Point2D nearest = new Point2D(100, 100); // base faraway test point
        Node n = root;

        // call recursive helper function
        nearest = recurNear(n, nearest, p);
        return nearest;
```

```java
    }

    // recursive helper function to find shortest distance to a point
    private Point2D recurNear(Node current, Point2D nearest, Point2D p)
    {
        // initial point comparison
        if (p.distanceSquaredTo(current.p) < p.distanceSquaredTo(nearest))
            nearest = current.p;

        if (current.color == RED)
        {
            if (p.x() < current.p.x()) // compare left then right
            {
                // set new shortest distance if found
                if ((current.lb != null)
                        && (current.lb.rect.distanceSquaredTo(p)
                        < p.distanceSquaredTo(nearest)))
                    nearest = recurNear(current.lb, nearest, p);

                if ((current.rt != null)
                        && (current.rt.rect.distanceSquaredTo(p)
                        < p.distanceSquaredTo(nearest)))
                    nearest = recurNear(current.rt, nearest, p);
            }
            else // compare right then left
            {
                // set new shortest distance if found
                if ((current.rt != null)
                        && (current.rt.rect.distanceSquaredTo(p)
                        < p.distanceSquaredTo(nearest)))
                    nearest = recurNear(current.rt, nearest, p);

                if ((current.lb != null)
                        && (current.lb.rect.distanceSquaredTo(p)
                        < p.distanceSquaredTo(nearest)))
                    nearest = recurNear(current.lb, nearest, p);
            }
        }

        else // BLUE
        {
            if (p.y() < current.p.y()) // compare left then right
            {
                // set new shortest distance if found
                if ((current.lb != null)
                        && (current.lb.rect.distanceSquaredTo(p)
                        < p.distanceSquaredTo(nearest)))
                    nearest = recurNear(current.lb, nearest, p);

                if ((current.rt != null)
                        && (current.rt.rect.distanceSquaredTo(p)
                        < p.distanceSquaredTo(nearest)))
                    nearest = recurNear(current.rt, nearest, p);
            }
            else // compare right then left
            {
                // set new shortest distance if found
                if ((current.rt != null)
                        && (current.rt.rect.distanceSquaredTo(p)
                        < p.distanceSquaredTo(nearest)))
                    nearest = recurNear(current.rt, nearest, p);

                if ((current.lb != null)
                        && (current.lb.rect.distanceSquaredTo(p)
                        < p.distanceSquaredTo(nearest)))
                    nearest = recurNear(current.lb, nearest, p);
            }
        }
        return nearest;
    }


    // unit testing of the methods
    public static void main(String[] args)
    {
        /*Point2D a = new Point2D(.5, .5);
        Point2D a = new Point2D(0.206107, 0.095492);
        Point2D b = new Point2D(0.975528, 0.654508);
        Point2D c = new Point2D(0.024472, 0.345492);
        Point2D d = new Point2D(0.793893, 0.095492);
        Point2D e = new Point2D(0.793893, 0.904508);
        Point2D f = new Point2D(0.975528, 0.345492);
        Point2D g = new Point2D(0.206107, 0.904508);
        Point2D h = new Point2D(0.500000, 0.000000);
        Point2D i = new Point2D(0.024472, 0.654508);
        Point2D j = new Point2D(0.500000, 1.000000);
        KdTreeST tree = new KdTreeST();
        tree.insert(a, "dummya");
        tree.insert(b, "dummyb");
        tree.insert(c, "dummyc");
        tree.insert(d, "dummyd");
        tree.insert(e, "dummye");
        tree.insert(f, "dummyf");
        tree.insert(g, "dummyg");
        tree.insert(h, "dummyh");
        tree.insert(i, "dummyi");
        tree.insert(j, "dummyj");
        StdOut.println("size is " + tree.size());
        StdOut.println("isEmpty? " + tree.isEmpty());
        Point2D sample = new Point2D(0.81, 0.3);
        Point2D sample2 = new Point2D(0.206107, 0.904508);
        StdOut.println("should be false " + tree.contains(sample));
        StdOut.println("should be true " + tree.contains(sample2));


        StdOut.println(tree.contains(a));
        StdOut.println(tree.contains(b));
        StdOut.println(tree.get(c));
        StdOut.println(tree.get(d));
        StdOut.println(tree.get(e));
        tree.draw();*/
    }
}
```

./Code/KdTreeST.java

CreateTables.java

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.*;

/**
 *  Created by: Tharald S. Fongaard & Jacob Perricone
 *  Advisor: Alain Kornhauser
 *  Date: 05/05/2015
 *
 *  Part Of Project: ORF 376 - Independent Work
 *
 *  Princeton University    2015
 */
public class CreateTables {

    // Main Execution, Creates Database with all state tables and given preferences
    public static void main(String args[]) throws ClassNotFoundException, SQLException {
        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://127.0.0.1:3306/Path";
        Connection m_Connection = DriverManager.getConnection(url, "tharald", "putin");
        XMLReader xmlReader = new XMLReader();
        for(String state: xmlReader.getAllStates()) {
            PreparedStatement total = m_Connection.prepareStatement("CREATE TABLE " + state + "( ID double unsigned NOT NULL,PersonID double DEFAULT NULL, Segment int(11) NOT NULL, aNode int(11) DEFAULT NULL,bNode int(11) DEFAULT NULL, LinkID int(11) DEFAULT NULL, LinkName text,oPixel double DEFAULT NULL,dPixel double DEFAULT NULL, dTime double DEFAULT NULL, aTime double DEFAULT NULL, oType text, dType text, a_Lat double DEFAULT NULL, a_Long double DEFAULT NULL, b_Lat double DEFAULT NULL, b_Long double DEFAULT NULL, aNode_Time double DEFAULT NULL, bNode_Time double DEFAULT NULL, PRIMARY KEY (ID,Segment)) ENGINE=InnoDB DEFAULT CHARSET=latin1");
            total.executeUpdate();
        }
    }
}
```

./Code/CreateTables.java

B.2 JavaScript & HTML Code

B.2.1 WebGL Globe Animation

WebGL_Globe.html

```html
<!DOCTYPE HTML>
<html lang="en">
<head>
    <title>WebGL Globe</title>
    <meta charset="utf-8">
    <style type="text/css">
        html {
            height: 100%;
        }

        body {
            margin: 0;
            padding: 0;
            background: #000000 url(loading.gif) center center no-repeat;
            color: #ffffff;
            font-family: sans-serif;
            font-size: 13px;
            line-height: 20px;
            height: 100%;
        }
        #info {
            font-size: 11px;
            position: absolute;
            bottom: 5px;
            background-color: rgba(0,0,0,0.8);
            border-radius: 3px;
            right: 10px;
            padding: 10px;
        }
        #currentInfo {
            width: 270px;
            position: absolute;
            left: 20px;
            top: 63px;
            background-color: rgba(0,0,0,0.2);
            border-top: 1px solid rgba(255,255,255,0.4);
            padding: 10px;
        }
        a {
            color: #aaa;
            text-decoration: none;
        }
        a:hover {
            text-decoration: underline;
        }
        .bull {
            padding: 0 5px;
            color: #555;
        }
        #title {
```

```
            position: absolute;
            top: 20px;
            width: 270px;
            left: 20px;
            background-color: rgba(0,0,0,0.2);
            border-radius: 3px;
            font: 20px Georgia;
            padding: 10px;
        }
        .year {
            font: 16px Georgia;
            line-height: 26px;
            height: 30px;
            text-align: center;
            float: left;
            margin-right: 10px;
            color: rgba(255, 255, 255, 0.4);
            cursor: pointer;
            -webkit-transition: all 0.1s ease-out;
        }
        .year:hover, .year.active {
            font-size: 23px;
            color: #fff;
        }
        #ce span {
            display: none;
        }
        #ce {
            width: 107px;
            height: 55px;
            display: block;
            position: absolute;
            bottom: 15px;
            left: 20px;
            background: url(ce.png);
        }
        .globe {
            display: none;
        }
        .show {
            display: block;
        }
    </style>
</head>
<body>
<div id="container0" class="globe"></div>
<div id="container3" class="globe"></div>
<div id="container6" class="globe"></div>
<div id="container9" class="globe"></div>
<div id="container12" class="globe"></div>
<div id="container15" class="globe"></div>
<div id="container18" class="globe"></div>
<div id="container21" class="globe"></div>

<div id="title">
    Trips by Time Period
</div>
<div id="currentInfo">
    <span id="year0" class="year">12:00-3:00am</span>
    <span id="year3" class="year">3:00-6:00am</span>
    <span id="year6" class="year">6:00-9:00am</span>
    <span id="year9" class="year">9:00-12:00pm</span>
    <span id="year12" class="year">12:00-3:00pm</span>
    <span id="year15" class="year">3:00-6:00pm</span>
    <span id="year18" class="year">6:00-9:00pm</span>
    <span id="year21" class="year">9:00-12:00am</span>
</div>
<script type="text/javascript" src="/globe/third-party/ThreeWebGL.js"></script>
<script type="text/javascript" src="/globe/third-party/ThreeExtras.js"></script>
<script type="text/javascript" src="/globe/third-party/RequestAnimationFrame.js">
     </script>
<script type="text/javascript" src="/globe/third-party/Detector.js"></script>
<script type="text/javascript" src="/globe/third-party/Tween.js"></script>
<script type="text/javascript" src="/globe/third-party/three.min.js"></script>
<script type="text/javascript" src="/globe/third-party/jquery.min.js"></script>
<script type="text/javascript" src="globe.js"></script>
<script type="text/javascript">
    if(!Detector.webgl){
        Detector.addGetWebGLMessage();
    } else {
        //initialize buckers
        years = ["0",  "3",  "6",  "9", "12", "15", "18", "21"];
        //create the array of div elements
        containers = new Array(years.length);
        //vreate the array of globes
        var globes = new Array(years.length);

        var i, tweens = [];

        //initialize Tween
        var xhr;
        TWEEN.start();
        //event listener function on each time span
        settime = function(i) {
            return function() {
                console.log(i);
                $(document.getElementById("container" + years[i])).addClass("show
                     ");
                for(j = i + 1; j < years.length; j++) {
                    // console.log("hide"); console.log(document.getElementById("
                         container" + years[j]));
                    $(document.getElementById('container' + years[j])).
                         removeClass("show");
                }
            };
        }
        // initialize even listeners
        for(var i = 0; i<years.length; i++) {
            y = document.getElementById('year'+years[i]);
            console.log(y);
            y.addEventListener('mouseover', settime(i), false);
        }

        //load in the json data
        xhr = new XMLHttpRequest();
        xhr.open('GET', '/globe/new8.json', true);
        xhr.onreadystatechange = function(e) {
            if (xhr.readyState === 4) {
                if (xhr.status === 200 || xhr.status==304) {
                    data = JSON.parse(xhr.responseText);
                    window.data = data;
                    //add data
```

```
                    for (i=0;i<data.length;i++) {
                        container = document.getElementById('container' + years[i
                            ]);
                        globes[i] = new DAT.Globe(container);
                        console.log(data[i][0]);
                        console.log(data[i][1]);
                        globes[i].addData(data[i][1], {format: 'magnitude', name:
                            data[i][0], animated: false});
                    }
                    //create points
                    for(var i = 0; i<years.length; i++) {
                        console.log("creating and animating" + i);
                        globes[i].createPoints();
                        globes[i].animate();
                    }
                    console.log("triggering");
                }
            }
        };
        xhr.send(null);
        console.log("triggering");
        //settime(0);

    }
</script>
</body>
</html>
```

./Code/WebGL_Globe.html

Globe.js

```
//Declare DAT Instance
var DAT = DAT || {};
DAT.Globe = function(container, opts) {
    //declare global variables
    opts = opts || {};

    var colorFn = opts.colorFn || function(x) {
            var c = new THREE.Color();
            c.setHSL( ( 0.6 - ( x * 0.5 ) ), 1.0, 0.5 );
            return c;
        };
    var imgDir = opts.imgDir || '/globe/';
//give texture to  the earth object
    var Shaders = {
        'earth' : {
            uniforms: {
                'texture': { type: 't', value: null }
            },
            vertexShader: [
                'varying vec3 vNormal;',
                'varying vec2 vUv;',
                'void main() {',
                'gl_Position = projectionMatrix * modelViewMatrix * vec4(
                    position, 1.0 );',
                'vNormal = normalize( normalMatrix * normal );',
                'vUv = uv;',
                '}'
            ].join('\n'),
            fragmentShader: [
                'uniform sampler2D texture;',
                'varying vec3 vNormal;',
                'varying vec2 vUv;',
                'void main() {',
                'vec3 diffuse = texture2D( texture, vUv ).xyz;',
                'float intensity = 1.05 - dot( vNormal, vec3( 0.0, 0.0, 1.0 ) )
                    ;',
                'vec3 atmosphere = vec3( 1.0, 1.0, 1.0 ) * pow( intensity, 3.0 )
                    ;',
                'gl_FragColor = vec4( diffuse + atmosphere, 1.0 );',
                '}'
            ].join('\n')
        },
        'atmosphere' : {
            uniforms: {},
            vertexShader: [
                'varying vec3 vNormal;',
                'void main() {',
                'vNormal = normalize( normalMatrix * normal );',
                'gl_Position = projectionMatrix * modelViewMatrix * vec4(
                    position, 1.0 );',
                '}'
            ].join('\n'),
            fragmentShader: [
                'varying vec3 vNormal;',
                'void main() {',
                'float intensity = pow( 0.8 - dot( vNormal, vec3( 0, 0, 1.0 ) ),
                    12.0 );',
                'gl_FragColor = vec4( 1.0, 1.0, 1.0, 1.0 ) * intensity;',
                '}'
            ].join('\n')
        }
    };
    //initialize camera, scene, mesh, etc.
    var camera, scene, renderer, w, h;
    var mesh, atmosphere, point;
    var overRenderer;
    var curZoomSpeed = 0;
    var zoomSpeed = 50;
    //initialize intitiale position
    var mouse = { x: 0, y: 0 }, mouseOnDown = { x: 0, y: 0 };
    rotation = { x: -Math.PI*1, y: 0 },
        target = { x: Math.PI, y: Math.PI / 6.0 },
        targetOnDown = { x: 0, y: 0 };
    var distance = 100000, distanceTarget = 100000;
    var padding = 40;
    var PI_HALF = Math.PI / 2;
    //
    function init() {
        container.style.color = '#fff';
        container.style.font = '13px/20px Arial, sans-serif';
        var shader, uniforms, material;
        w = container.offsetWidth || window.innerWidth;
        h = container.offsetHeight || window.innerHeight;
        //create new Camera
        camera = new THREE.PerspectiveCamera(30, w / h, 1, 10000);
```

```
        camera.position.z = distance;
        //Create new schene
        scene = new THREE.Scene();
        //create a 3D geometry
        var geometry = new THREE.SphereGeometry(200, 40, 30);
        shader = Shaders['earth'];
        uniforms = THREE.UniformsUtils.clone(shader.uniforms);
        uniforms['texture'].value = THREE.ImageUtils.loadTexture(imgDir+'world.jpg');
        // shade the material
        material = new THREE.ShaderMaterial({
            uniforms: uniforms,
            vertexShader: shader.vertexShader,
            fragmentShader: shader.fragmentShader
        });
        //create a mesh from the geometry and material and
        //add too scene
        mesh = new THREE.Mesh(geometry, material);
        mesh.rotation.y = Math.PI;
        scene.add(mesh);
        shader = Shaders['atmosphere'];
        uniforms = THREE.UniformsUtils.clone(shader.uniforms);
        material = new THREE.ShaderMaterial({
            uniforms: uniforms,
            vertexShader: shader.vertexShader,
            fragmentShader: shader.fragmentShader,
            side: THREE.BackSide,
            blending: THREE.AdditiveBlending,
            transparent: true
        });
        //create outer layer
        mesh = new THREE.Mesh(geometry, material);
        mesh.scale.set( 1.1, 1.1, 1.1 );
        scene.add(mesh);
        //creat box cover
        geometry = new THREE.BoxGeometry(0.75, 0.75, 1);
        geometry.applyMatrix(new THREE.Matrix4().makeTranslation(0,0,-0.5));
        //intialize a point on the geometry
        point = new THREE.Mesh(geometry);
        //intiialize the render
        renderer = new THREE.WebGLRenderer({antialias: true});
        renderer.setSize(w, h);
        //add event listeners
        renderer.domElement.style.position = 'absolute';
        container.appendChild(renderer.domElement);
        container.addEventListener('mousedown', onMouseDown, olse);
        container.addEventListener('mousewheel', onMouseWheel, false);
        document.addEventListener('keydown', onDocumentKeyDown, false);
        window.addEventListener('resize', onWindowResize, false);
        container.addEventListener('mouseover', function() {
            overRenderer = true;
        }, false);
        container.addEventListener('mouseout', function() {
            overRenderer = false;
        }, false);
    }
//add data to the globe
    function addData(data, opts) {
        var lat, lng, size, color, i, step, colorFnWrapper;
        opts.animated = opts.animated || false;
        this.is_animated = opts.animated;
        opts.format = opts.format || 'magnitude'; // other option is 'legend'
        if (opts.format === 'magnitude') {
            step = 3;
            colorFnWrapper = function(data, i) { return colorFn(data[i+2]); }
        } else if (opts.format === 'legend') {
            step = 4;
            colorFnWrapper = function(data, i) { return colorFn(data[i+3]); }
        } else {
            throw('error: format not supported: '+opts.format);
        }
        // add size component if inputed
        if (opts.animated) {
            if (this._baseGeometry === undefined) {
                this._baseGeometry = new THREE.Geometry();
                for (i = 0; i < data.length; i += step) {
                    lat = data[i];
                    lng = data[i + 1];
//        size = data[i + 2];
                    color = colorFnWrapper(data,i);
                    size = 0;
                    addPoint(lat, lng, size, color, this._baseGeometry);
                }
            }
            if(this._morphTargetId === undefined) {
                this._morphTargetId = 0;
            } else {
                this._morphTargetId += 1;
            }
            opts.name = opts.name || 'morphTarget'+this._morphTargetId;
        }
        //create the size columns
        var subgeo = new THREE.Geometry();
        for (i = 0; i < data.length; i += step) {
            lat = data[i];
            lng = data[i + 1];
            color = colorFnWrapper(data,i);
            size = data[i + 2];
            size = size*200;
            addPoint(lat, lng, size, color, subgeo);
        }
        if (opts.animated) {
            this._baseGeometry.morphTargets.push({'name': opts.name, vertices: subgeo.vertices});
        } else {
            this._baseGeometry = subgeo;
        }
    };
    //create points on the mesh for all data
    function createPoints() {
        if (this._baseGeometry !== undefined) {
            if (this.is_animated === false) {
                this.points = new THREE.Mesh(this._baseGeometry, new THREE.MeshBasicMaterial({
                    color: 0xffffff,
                    vertexColors: THREE.FaceColors,
                    morphTargets: false
                }));
            } else {
                if (this._baseGeometry.morphTargets.length < 8) {
                    console.log('t l',this._baseGeometry.morphTargets.length);
                    var padding = 8-this._baseGeometry.morphTargets.length;
```

```
                    console.log('padding', padding);
                    for(var i=0; i<=padding; i++) {
                        console.log('padding',i);
                        this._baseGeometry.morphTargets.push({'name': '
                             morphPadding'+i, vertices: this._baseGeometry.
                             vertices});
                    }
                }
                this.points = new THREE.Mesh(this._baseGeometry, new THREE.
                     MeshBasicMaterial({
                    color: 0xffffff,
                    vertexColors: THREE.FaceColors,
                    morphTargets: true
                }));
            }
            scene.add(this.points);
        }
    }
//add singular point
    function addPoint(lat, lng, size, color, subgeo) {
        var phi = (90 - lat) * Math.PI / 180;
        var theta = (180 - lng) * Math.PI / 180;
        point.position.x = 200 * Math.sin(phi) * Math.cos(theta);
        point.position.y = 200 * Math.cos(phi);
        point.position.z = 200 * Math.sin(phi) * Math.sin(theta);
        point.lookAt(mesh.position);
        point.scale.z = Math.max( size, 0.1 ); // avoid non-invertible matrix
        point.updateMatrix();
        for (var i = 0; i < point.geometry.faces.length; i++) {
            point.geometry.faces[i].color = color;
        }
        if(point.matrixAutoUpdate){
            point.updateMatrix();
        }
        subgeo.merge(point.geometry, point.matrix);
    }
//give 3d capabailities to the app by triggering event listeners
    function onMouseDown(event) {
        event.preventDefault();
        container.addEventListener('mousemove', onMouseMove, false);
        container.addEventListener('mouseup', onMouseUp, false);
        container.addEventListener('mouseout', onMouseOut, false);
        mouseOnDown.x = - event.clientX;
        mouseOnDown.y = event.clientY;
        targetOnDown.x = target.x;
        targetOnDown.y = target.y;
        container.style.cursor = 'move';
    }
//give 3d capabailities to the app by triggering event listeners
    function onMouseMove(event) {
        mouse.x = - event.clientX;
        mouse.y = event.clientY;
        var zoomDamp = distance/1000;
        target.x = targetOnDown.x + (mouse.x - mouseOnDown.x) * 0.005 * zoomDamp;
        target.y = targetOnDown.y + (mouse.y - mouseOnDown.y) * 0.005 * zoomDamp;
        target.y = target.y > PI_HALF ? PI_HALF : target.y;
        target.y = target.y < - PI_HALF ? - PI_HALF : target.y;
    }
//give 3d capabailities to the app by triggering event listeners
    function onMouseUp(event) {
        container.removeEventListener('mousemove', onMouseMove, false);
        container.removeEventListener('mouseup', onMouseUp, false);
        container.removeEventListener('mouseout', onMouseOut, false);
        container.style.cursor = 'auto';
    }
//give 3d capabailities to the app by triggering event listeners
    function onMouseOut(event) {
        container.removeEventListener('mousemove', onMouseMove, false);
        container.removeEventListener('mouseup', onMouseUp, false);
        container.removeEventListener('mouseout', onMouseOut, false);
    }
//give 3d capabailities to the app by triggering event listeners
    function onMouseWheel(event) {
        event.preventDefault();
        if (overRenderer) {
            zoom(event.wheelDeltaY * 0.3);
        }
        return false;
    }
//give 3d capabailities to the app by triggering event listeners
    function onDocumentKeyDown(event) {
        switch (event.keyCode) {
            case 38:
                zoom(100);
                event.preventDefault();
                break;
            case 40:
                zoom(-100);
                event.preventDefault();
                break;
        }
    }
//give 3d capabailities to the app by triggering event listeners
    function onWindowResize( event ) {
        camera.aspect = container.offsetWidth / container.offsetHeight;
        camera.updateProjectionMatrix();
        renderer.setSize( container.offsetWidth, container.offsetHeight );
    }
//resize on zoome
    function zoom(delta) {
        distanceTarget -= delta;
        distanceTarget = distanceTarget > 1000 ? 1000 : distanceTarget;
        distanceTarget = distanceTarget < 350 ? 350 : distanceTarget;
    }
//render the graph
    function animate() {
        requestAnimationFrame(animate);
        render();
    }
//initialize the camera position
    function render() {

        zoom(curZoomSpeed);
        rotation.x += (target.x - rotation.x) * 0.1;
        rotation.y += (target.y - rotation.y) * 0.1;
        distance += (distanceTarget - distance) * 0.3;
        camera.position.x = distance * Math.sin(rotation.x) * Math.cos(rotation.y
             );
        camera.position.y = distance * Math.sin(rotation.y);
        camera.position.z = distance * Math.cos(rotation.x) * Math.cos(rotation.y
             );
        camera.lookAt(mesh.position);
```

```
        renderer.render(scene, camera);
    }
    init()
    this.animate = animate;
    this.__defineGetter__('time', function() {
        return this._time || 0;
    });
    //more event listeners.
    this.__defineSetter__('time', function(t) {
        var validMorphs = [];
        var morphDict = this.points.morphTargetDictionary;
        for(var k in morphDict) {
            if(k.indexOf('morphPadding') < 0) {
                validMorphs.push(morphDict[k]);
            }
        }
        validMorphs.sort();
        var l = validMorphs.length-1;
        var scaledt = t*l+1;
        var index = Math.floor(scaledt);
        for (i=0;i<validMorphs.length;i++) {
            this.points.morphTargetInfluences[validMorphs[i]] = 0;
        }
        var lastIndex = index - 1;
        var leftover = scaledt - index;
        if (lastIndex >= 0) {
            this.points.morphTargetInfluences[lastIndex] = 1 - leftover;
        }
        this.points.morphTargetInfluences[index] = leftover;
        this._time = t;
    });
    this.addData = addData;
    this.createPoints = createPoints;
    this.renderer = renderer;
    this.scene = scene;
    return this;
};
```

./Code/Globe.js

B.3 Python & Bash Scripts

B.3.1 Network Data Cleaning

Nodeclean.py

```
import sys
import string
def main():
  line = sys.stdin.readline()
  mylist = [];
  while(line != ""):
    mylist = line.split(",");
    print ",".join(map(str, mylist[:5])),
    rest = mylist[5].split();
    if len(rest) == 1:
      print "," + rest[0] + ",";
    else:
      b = rest[1:];
      print "," + rest[0] + "," + " ".join(map(str, b));
    line = sys.stdin.readline()
main();
```

./Code/Nodeclean.py

Linkclean.py

```
def main():
  line = sys.stdin.readline()
  mylist = [];
  i = 0;
  while(line != ""):
    while("Skip" in line):
      line = sys.stdin.readline();
    line.strip("\r");
    mylist = line.split();
    if len(mylist) == 4:
      print str(i) + ", " + mylist[0] + ", " + mylist[1] + ", " + mylist[2] + ",
            " + mylist[3];
    elif len(mylist) == 5:
      print str(i) + ", " + mylist[0] + ", " + mylist[1] + ", " + mylist[2] + ",
            " + mylist[3] + ", " + mylist[4];
    elif len(mylist) > 5:
      print str(i) + ", " + mylist[0] + ", " + mylist[1] + ", " + mylist[2] + ",
            " + mylist[3] + ", ",
      print ' '.join(map(str, mylist[4:]));
    i = i + 1;
    line = sys.stdin.readline()
main();
```

./Code/Linkclean.py

Restclean.py

```
import sys
import string
def main():
  line = sys.stdin.readline()
  mylist = [];
```

```
  while(line != ""):
    mylist = line.split(",");
    print ",".join(map(str, mylist[:5])),
    rest = mylist[5].split();
    if len(rest) == 1:
      print "," + rest[0] + ",";
    else:
      b = rest[1:];
      print "," + rest[0] + "," + " ".join(map(str, b));
    line = sys.stdin.readline()
main();
```

./Code/Restclean.py