Analyzing the Effects of Space and Time on Bikeshare Use: A Case Study in Washington, DC

Matthew Wigginton Conway

December 14, 2013

Bikesharing is a relatively new form of shared transportation wherein bikes are deposited at stations throughout a city. Users pay an annual fee and they can then take bikes from any station and return them to any station. These systems generate a wealth of data. The stations are electronic, so each time a trip is taken, a record is stored in a database with the origin, destination, start and end times of that trip. For some systems, this data is freely available online. This project aims to use data on the approximately 4.5 million trips taken on Washington, DC's Capital Bikeshare (CaBi) system from 2010 through the present to examine how time of day and station location affect the usage of the bikeshare system.

1 Data Processing

Before analysis could be undertaken, the data needed to be obtained and cleaned. The data are available from the CaBi website[1] as a series of quarterly CSV files. Data from the fourth quarter of 2010 through the 2nd quarter of 2013 were used. Each record contains the origin and destination of the trip, the start and end times, and ancillary information. The script fetchData.sh (page 12) was used to download the CSV files. The files were then merged using csvMerge.py (page 13). This script also cleaned the data; the column names in the CSV files from different quarters differ slightly, so the script contains code to normalize them. It also renames the columns to remove whitespace, making it simpler to use the data in R.

There are no spatial data present in the trip history files. To remedy this, station locations were retrieved from the CaBi real-time API[2] and merged with the trip history data using expandFileWithXY.py (page 15). This script also projects the station locations to Universal Transverse Mercator so that Euclidean math can be used.

1. `http://capitalbikeshare.com/system-data`
2. `http://capitalbikeshare.com/data/stations/bikeStations.xml`

Finally, load˙data.R (page 19) was used to load the data into R, removing trips longer than 20km or 2 hours, assuming these trips to be errors. This script also implicitly removes trips that are missing either origin or destination coordinates; these trips likely began or ended at stations that have been removed. Since we derive the geographic coordinates from the real-time station information feed, stations that are no longer active have no coordinates. There are 7168 such trips; this count can be determined either by the output of expandFileWithXY.py (page 15) or the dedicated script findTripsNoCoords.py (page 17). After cleaning (and labeling, described in the next section), 4,485,213 trips were left for analysis.

2 Time of Day Effects

One of the chief difficulties of any bikeshare system is keeping bicycles balanced across the system. The problem is doubly constrained, because the system operator not only needs to keep enough bikes available at all stations, but also needs to prevent stations from becoming completely full and thus preventing people from parking the bikes. This is usually accomplished via a fleet of trucks which pick up bikes from overfull stations and rebalance them to empty or nearly empty stations.

For the time of day portion of this project, the data on bike share trips was used to determine whether the distribution of start and end stations differs significantly between time periods. Eight time periods were defined: morning (6a–9a), midday (9a–3p), afternoon (3p–7p) and overnight (7p–9p) for both weekdays and weekends. The boundaries of the time periods are the same as those used in the Metropolitan Washington Council of Governments travel model, although MWCOG does not further divide the time periods into weekday and weekend patterns (Metropolitan Washington Council of Governments 2013, 14).

2.1 Methodology

Each trip in the data was first assigned a time period based on its start time using labeling.R (page 20). For each time period, an origin-destination matrix was created, showing the number of trips between each station pair using relabel.R (page 22). The matrices were then compared pairwise, comparing each time period to every other time period. The following test statistic was computed for each pairwise comparison.

$$\frac{\sum_i \sum_j (t_{ij,1} - t_{ij,2})^2}{(\sum_i \sum_j t_{ij,2})^2} \tag{1}$$

Where i and j represent origin and destination stations, respectively, $t_{ij,1}$ is trips from origin i to destination j in time period 1, and $t_{ij,2}$ is trips from origin i to destination j in time period 2. The denominator of the equation is to scale the test statistic based on the total number of trips taken in the time period, so that the magnitude of the test statistic is not affected by the absolute number of trips taken in the time period. Additionally, matrix 2 is scaled before calculation such that the total number of trips in each matrix are the same. That is,

$$\sum_i \sum_j t_{ij,1} = \sum_i \sum_j t_{ij,2} \tag{2}$$

Once test statistics were computed for each pair of time periods, a Monte Carlo simulation was undertaken to determine whether the time periods differ significantly. The trips that were originally used to generate the origin-destination matrices were randomly reassigned to different time periods. The number of trips in each time period was held constant. Since this worked by relabeling the existing trips, the distribution of trips to origins and destinations was constant over all time periods though it varied within time periods. Origin-destination matrices were then calculated using the relabeled trips, and the same pairwise comparison was done and test statistics computed. This process was repeated 999 times. This simulation was performed by relabel.R (page 22). If there is no significant difference in the origin-destination matrices between time periods, one would expect the test statistics from the observed data to fall in the middle of the distribution.

2.2 Results

It was found that there is an effect of time of day on the origin-destination matrices. For every pair, there was no test statistic from the Monte Carlo simulation higher than the test statistic from the observed pair. The p-values between time periods are show below.

	WkMr	WkMd	WkAf	WkNt	WeMr	WeMd	WeAf	WeNt
Weekday Morning (6a–9a)	0.999	0.000*	0.000*	0.000*	0.000*	0.000*	0.000*	0.000*
Weekday Midday (9a–3p)	0.000*	0.999	0.000*	0.000*	0.000*	0.000*	0.000*	0.000*
Weekday Afternoon (3p–7p)	0.000*	0.000*	0.999	0.000*	0.000*	0.000*	0.000*	0.000*
Weekday Overnight (7p–6a) †	0.000*	0.000*	0.000*	0.999	0.000*	0.000*	0.000*	0.000*
Weekend Morning (6a–9a)	0.000*	0.000*	0.000*	0.000*	0.999	0.000*	0.000*	0.000*
Weekend Midday (9a–3p)	0.000*	0.000*	0.000*	0.000*	0.000*	0.999	0.000*	0.000*
Weekend Afternoon (3p–7p)	0.000*	0.000*	0.000*	0.000*	0.000*	0.000*	0.999	0.000*
Weekend Overnight (7p–6a) †	0.000*	0.000*	0.000*	0.000*	0.000*	0.000*	0.000*	0.999

* Statistically significant at $\alpha = 0.05$ level

† Friday night is a weekend night, Sunday night is a weekday night.

The p-value represents the probability that the observed differences between two time periods could have occurred by chance. As we can see, there is a statistically significant difference between every time period and every other time period ($p < 0.05$).

2.3 Discussion

Such a result should not be surprising. To take a trivial example, commuters may ride from a residential area to a Metro stop each morning, and the reverse each evening. Weekend trips may represent people using the system for pleasure or entertainment rather than commuting. The data confirm that usage patterns (measured by station origin-destination matrices) differ for each time period.

This intensifies the need for rebalancing. Bikeshare demand is often not in equilibrium, so bikes must be moved from station to station by the system operator in order to ensure that there are available bikes and available docks at all stations (Instituto para la Diversificación y Ahorro de la Energía 2007, 108). That is, if most people ride bikes from residential areas to commercial areas in the morning, there will not be enough bikes available in residential areas, and there will not be enough docks available in commercial areas. The problem is doubly-constrained, because there need to both be bikes available at each station in order for users to take a bike, and empty docks in order to return bikes. An interesting project would be to determine to what extent demand is cyclical. If the demand is cyclical, with people biking to commercial areas in the morning and back to residential areas in the afternoon, providing a sufficient number of bikes would ameliorate the need for rebalancing. There may, however, be a general trend as well. For instance, people may prefer to bikeshare to work if they are in a hurry and bikeshare is faster than transit,[3] and take transit home if they are tired. In this case, there could be a general trend of bikes moving towards commercial areas. This could be tested using the trip history data, but is beyond the scope of this project.

3 Effects of Space on Bikeshare Use

Some bikeshare stations are, of course, more popular than others. It was hypothesized that station popularities (defined here as the average number of bike movements—both pickups and dropoffs—per day) are spatially autocorrelated. That is, stations near each other would tend to have similar popularities. To test this, a Moran's I statistic was computed to evaluate whether there is significant autocorrelation between station popularities.

3. Using this same trip history data, it was found that many trips in the data are faster than comparable bikeshare trips; people tend to use bikeshare for trips where it is faster than transit (Wong 2012).

3.1 Methodology

Data were loaded into R, and were then summarized to get a count of bike movements for each station. This process was repeated twice to get counts for both the number of trips originating and the number of trips terminating at each station; the results were then summed by station. This number of bike movements was then normalized by dividing by the number of days each station had been open. The count of bike movements is skewed right, with many stations having relatively few bike movements per day, and a few stations having a large number of bike movements per day. In order to better analyze the data, a Box-Cox transformation was undertaken to normalize the distribution. Using the Shapiro-Wilks estimator, the Box-Cox parameter was estimated as $\lambda = 0.33$. This did serve to make the distribution more symmetric (see fig. 1 for a comparison between the untransformed and transformed distribution). The transformed distribution, however, is bimodal. This should not affect the Moran's I calculation, but explanation of this would be a worthwhile direction for future research.

To calculate I, a weight matrix was first calculated. This was done by creating a Delaunay triangulation of the stations using the `tri2nb` function from the `spdep` package. Links longer than 4km were removed, which left two graph components, one in Washington, Arlington and Alexandria, and one in Montgomery County, Maryland. The component in Montgomery County was removed for the purposes of this analysis. 4km was chosen as a the maximum link length visually, removing the most outrageous links while still keeping some degree of connectivity. A Shiny application was created to visualize the network with different thresholds for breaking links; a slider can be used to adjust the threshold while seeing results in real time (see Figure 3, server.R (page 28) and ui.R (page 28)). The adjacency graph defined is shown in figure 2.

The adjacency graph was then converted to a weight matrix in row-standardized form (that is, all rows sum to 1). Thus the neighborhood value of a point can be interpreted as a local mean. Moran's I was calculated, and a Moran plot was generated for further interpretation.

3.2 Results

A Moran's I value of 0.78 was found ($p < 0.05$), indicating strong positive spatial autocorrelation between stations. That is, stations near each other tend to have similar popularities. Upon examining the Moran plot (fig. 4), we see clearly the positive trend. We also see that the influential points (marked ⟐) are fairly evenly distributed about the scatter. These influential points are somewhat spatially dispersed as well (see fig. 5).

It should be noted that the coordinates may not be exactly right for all of the trips in the data. We are using the current coordinates of each station, drawn from the real-time station information feed. The stations are moved short distances from time to time (Wong 2012). It is believed that this does not significantly impact the results; the weight matrix

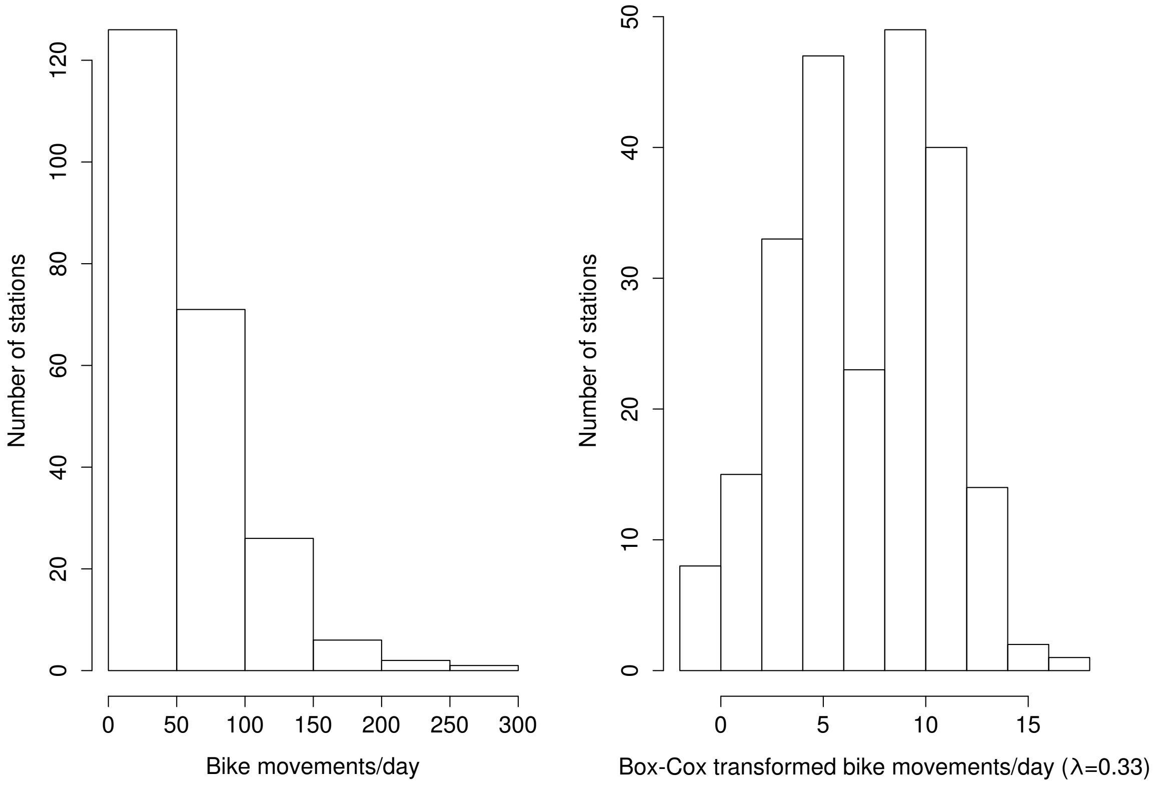


Figure 1: Station popularity, before and after Box-Cox transformation.

is based on adjacency rather than distance.

3.3 Discussion

There is significant positive spatial autocorrelation in the popularities of bikeshare stations. This makes sense intuitively, in terms of both first- and second-order effects. On the first-order side, areas with many attractions are likely to have more bike movements at all stations in the area. On the second-order side, each trip requires two stations within biking distance of each other; more stations closer together means more options for trips. One could also hypothesize an inhibition effect at very fine scales in some situations: if there are two stations very near each other, but one is preferable, it may inhibit use of the other one. For example, there are two stations near the San Francisco train station in the Bay Area Bikeshare system. One is directly in front of the station, and one is across the

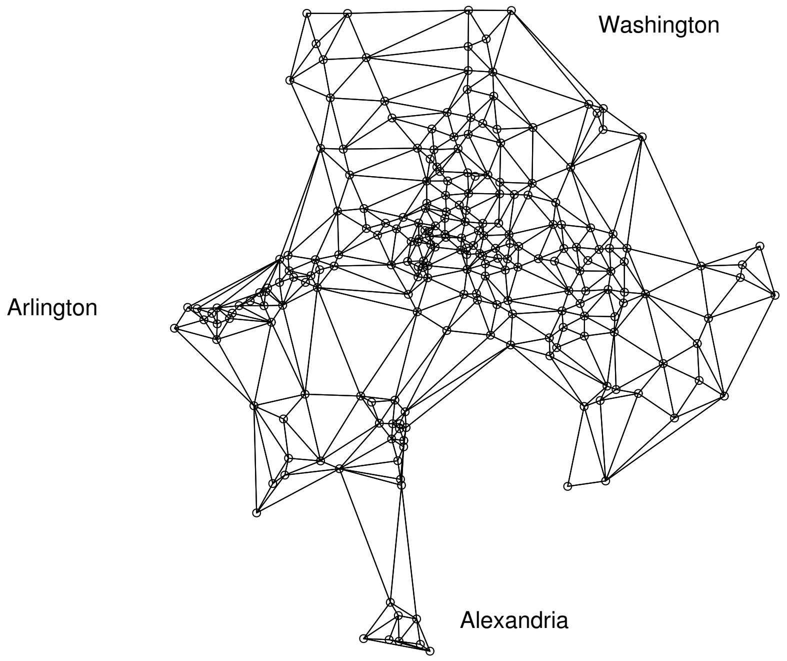
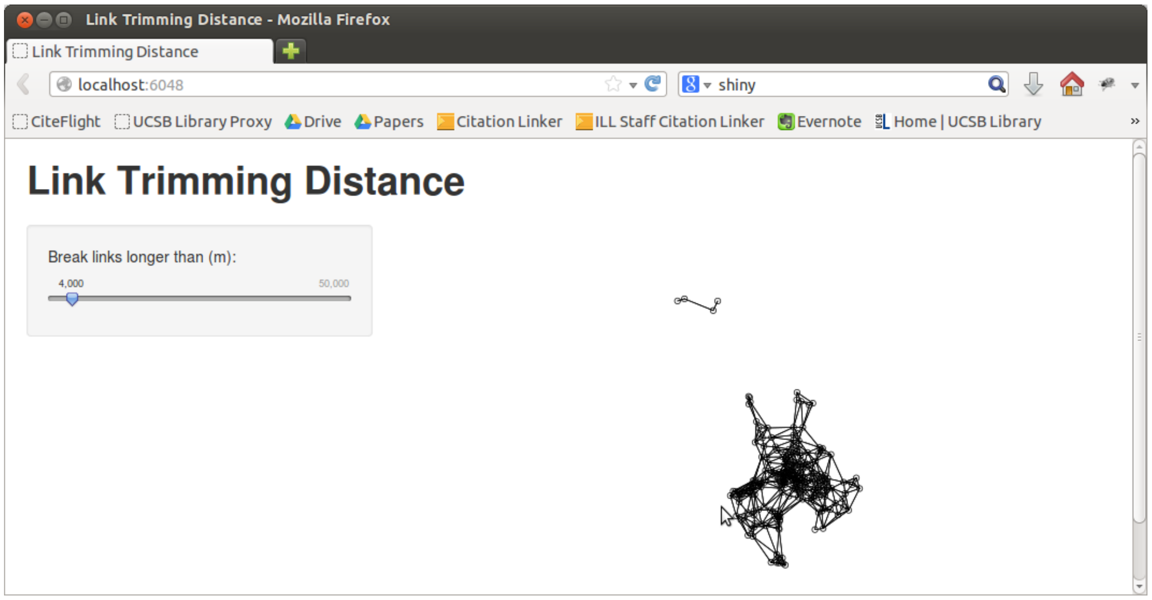


Figure 2: Station adjacency graph: Delaunay triangulation with links longer than 4km removed.

street. It is likely that the one across the street is less popular because people prefer to use the one in front of the station when bikes or docks are available. This project did not attempt to differentiate between first- and second-order effects.

In general, the locations of the influential points are not surprising. For example, station 31258, one of the stations that is much more popular than those around it, is located at the Lincoln Memorial, a popular destination for tourists. It is also a neighbor to the not-particularly-popular station 31211, despite being separated from it by freeways; this artificially pulls down the local average and makes station 31258 more influential. This is a downside of using pure Euclidean planar geometry to define the neighbor matrix. Station 31211 is itself another good example; it is much less popular than its neighbors. Some of this is due to the aforementioned connection with the station at the Lincoln Memorial. This station is also located at the Kennedy Center, a performing-arts center which is on the



Link Trimming Distance

Break links longer than (m):

4,000 50,000

Figure 3: A Shiny application for interactively visualizing networks with different link lengths; adjusting the slider changes the maximum link length in the network, breaking more or fewer links in the original triangulation.

Potomac and separated from much of Washington by Interstate 66. It is not particularly accessible by bicycle; this is likely the reason it is not particularly popular. Finally, the aforementioned inhibition effect of stations very close to each other may be occurring near DuPont Circle. Station 31200 is located directly on DuPont Circle, while station 31234 is located about a block away and around a corner. People, especially tourists, traveling to DuPont Circle likely ride to the more obvious station; this may inhibit use of the less obvious station.

4 General Discussion and Further Research

This project found that both time and space exert significant effects on bikeshare use. The distributions of trips between stations differ significantly at different times of day. There also tends to be spatial autocorrelation; popular stations tend to be near each other.

One interesting topic for further research would be to look further into the origin-destination matrices at different times of day. One could attempt to determine the direction of movement of the bikes at different times of day, and the determinants of the movement (for instance, are people from Metro stations to downtown areas to downtown areas in the morning due to the commute?). This research would be useful in that it could inform rebalancing. One team has developed statistical models for predicting bikeshare station

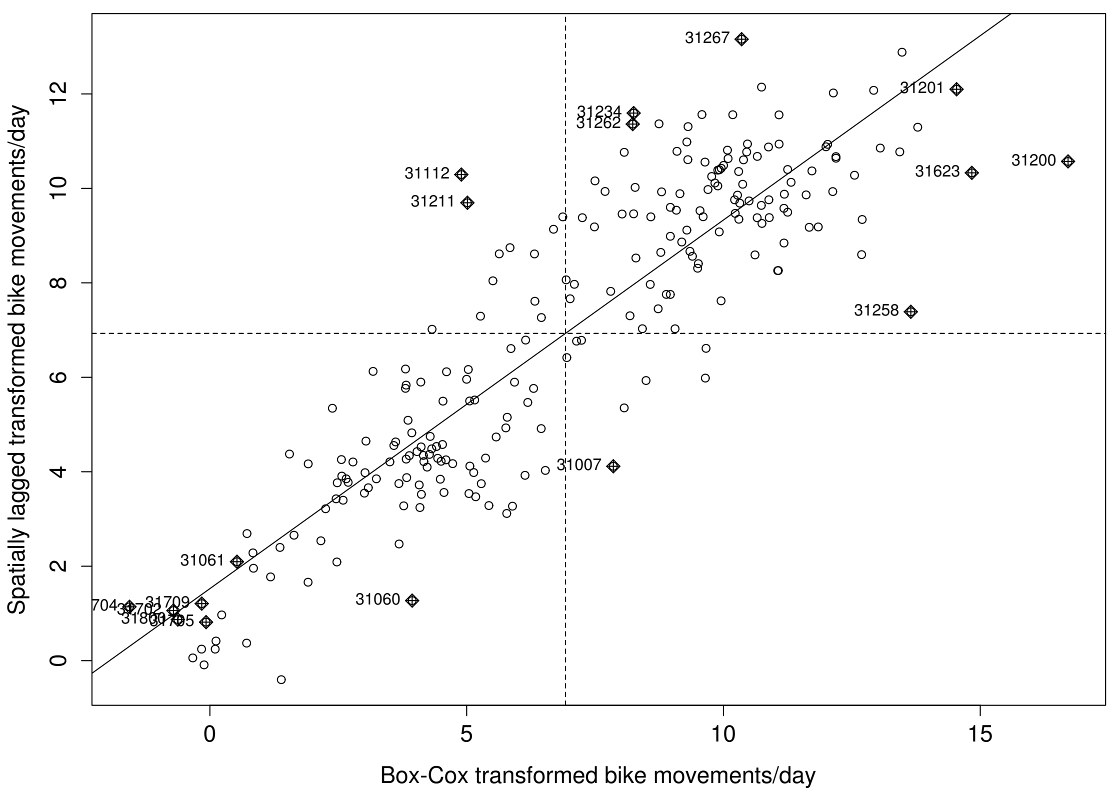


Figure 4: Moran plot of station popularity.

use, but their approach does not model the flow of bikes between stations because they did not have full trip data available in Chicago (Dempsey et al. 2013). Using origin-destination matrices would be an alternate way to model bikeshare use.

The bikeshare trip data provides a wealth of information for analysis. Very rarely do researchers have access to complete origin-destination matrices for a particular mode. This research confirms the value of geography in explaining the use of this new transportation tool. Further research could do more with this data, creating predictive tools to assist bikeshare system operators and planners.

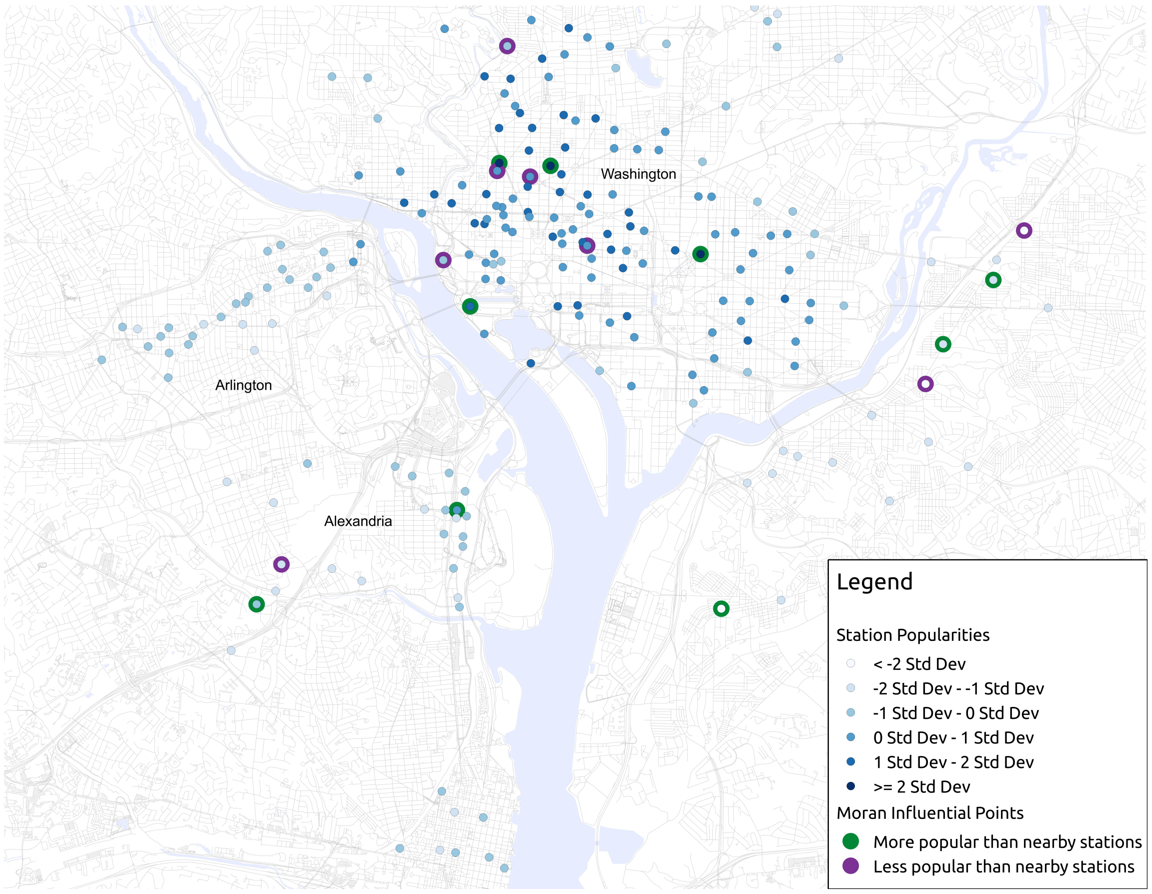


Figure 5: Map of station popularities and Moran influential points.

References

Dempsey, Walter, Juan-Pablo Velez, Adam Fishman, Jette Henderson, Breanna Miller, and Vidhur Vohra. 2013. *Methodology - dssg/bikeshare wiki.* `https://github.com/dssg/bikeshare/wiki/methodology`.

Instituto para la Diversificación y Ahorro de la Energía. 2007. *Guía metodológica para la implantación de sistemas de bicicletas públicas en España.* Madrid: Instituto para la Diversificación y Ahorro de la Energía. ISBN: 978-84-96680-24-1. `http://www.idae.es/index.php/mod.documentos/mem.descarga?file=/documentos_Guia_Bicicletas_8367007d.pdf`.

Metropolitan Washington Council of Governments. 2013. *User's Guide for the MWCOG/NCRTPB Travel Forecasting Model, Version 2.3, Build 52: Draft Report.* Technical report. Washington: Metropolitan Washington Council of Governments. `http://www.mwcog.org/transportation/activities/models/files/V2.3.52_Users_Guide_v2_w_appA.pdf`.

Wong, James. 2012. "When is Bikeshare Faster than Transit?" *Greater Greater Washington.* `http://www.greatergreaterwashington.org/post/15168/`.

A Source Code Listings

This appendix contains source code listings for the code used in this project. Data management and retrieval code is primarily Python, whereas analysis code is written in R. All of the code used in this project is licensed under the Apache License, and is also available at `https://www.github.com/mattwigway/bikeshare-analysis`.

A.1 Data Management

Scripts in this section are used to retrieve data from Capital Bikeshare and process it into a format suitable for analysis.

A.1.1 fetchData.sh

This shell script simply calls the other scripts to retrieve and process the data.

```
#!/bin/sh
# fetch and process all of the capital bikeshare data

# Copyright (C) 2013 Matthew Wigginton Conway.

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#    http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

BASEDIR=$(dirname $0)

wget http://capitalbikeshare.com/assets/files/trip-history-data/2010-4th-quarter.
    csv
wget http://capitalbikeshare.com/assets/files/trip-history-data/2011-1st-quarter.
    csv
wget http://capitalbikeshare.com/assets/files/trip-history-data/2011-2nd-quarter.
    csv
wget http://capitalbikeshare.com/assets/files/trip-history-data/2011-3rd-quarter.
    csv
wget http://capitalbikeshare.com/assets/files/trip-history-data/2011-4th-quarter.
    csv
wget http://capitalbikeshare.com/assets/files/trip-history-data/2012-1st-quarter.
    csv
wget http://capitalbikeshare.com/assets/files/trip-history-data/2012-2nd-quarter.
    csv
```

```
wget http://capitalbikeshare.com/assets/files/trip-history-data/2012-3rd-quarter.
    csv
wget http://capitalbikeshare.com/assets/files/trip-history-data/2012-4th-quarter.
    csv
wget http://capitalbikeshare.com/assets/files/trip-history-data/2013-1st-quarter.
    csv
wget http://capitalbikeshare.com/assets/files/trip-history-data/2013-2nd-quarter.
    csv

# for the latitude and longitude
wget http://capitalbikeshare.com/data/stations/bikeStations.xml

# merge the csv files
${BASEDIR}/csvMerge.py 201?-???-quarter.csv all-trips-aspatial.csv

# expand the csv file
${BASEDIR}/expandFileWithXY.py all-trips-aspatial.csv bikeStations.xml all-trips.
    csv
```

A.1.2 csvMerge.py

This script merges all of the quarterly CSV files into one large CSV file, merging columns that are the same but have different names in data files from different time periods.

```
#!/usr/bin/python
# Merge CSV files with the same columns

# Copyright (C) 2013 Matthew Wigginton Conway.

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#    http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from sys import argv
import csv
import re

# get the last argument
outfileName = argv[-1]
```

```
fieldnames = ['duration', 'duration_sec', 'start_date', 'start_station', 'start_
    terminal', 'end_date', 'end_station', 'end_terminal', 'bike_num', 'subscription
    _type', 'bike_key']
out = csv.DictWriter(open(outfileName, 'w'), fieldnames)
out.writeheader()

def renameKey(row, key, newKey):
    if row.has_key(key):
        row[newKey] = row[key]
        del row[key]
        return row

count = 0
for infile in argv[1:-1]:
    print '\nProcessing %s' % infile

    reader = csv.DictReader(open(infile))

    for row in reader:
        # Standardize field names
        renameKey(row, 'Member Type', 'Type')
        renameKey(row, 'Start station', 'Start Station')
        renameKey(row, 'End station', 'End Station')
        renameKey(row, 'Duration (Sec)', 'Duration(Sec)')
        renameKey(row, 'Duration (sec)', 'Duration(Sec)')
        renameKey(row, 'Start terminal', 'Start Terminal')
        renameKey(row, 'End terminal', 'End Terminal')
        renameKey(row, 'Subscriber Type', 'Subscription Type')
        renameKey(row, 'Type', 'Subscription Type')

        # Make field names R-friendly
        renameKey(row, 'Duration', 'duration')
        renameKey(row, 'Duration(Sec)', 'duration_sec')
        renameKey(row, 'Start date', 'start_date')
        renameKey(row, 'Start Station', 'start_station')
        renameKey(row, 'Start Terminal', 'start_terminal')
        renameKey(row, 'End date', 'end_date')
        renameKey(row, 'End Station', 'end_station')
        renameKey(row, 'End Terminal', 'end_terminal')
        renameKey(row, 'Bike#', 'bike_num')
        renameKey(row, 'Subscription Type', 'subscription_type')
        renameKey(row, 'Start time', 'start_date')
        renameKey(row, 'Bike Key', 'bike_key')

        # Sometimes they don't explicitly record the terminals
        if not row.has_key('start_terminal'):
            # rstrip ensures there is no trailing white space
            row['start_terminal'] = row['start_station'].rstrip()[-6:-1]
        if not row.has_key('end_terminal'):
            row['end_terminal'] = row['end_station'].rstrip()[-6:-1]

        # Back out the duration
        if not row.has_key('duration_sec'):
            m = re.search('([0-9]+)h[ .]+([0-9]+)mi?n?[ .]+([0-9]+)s', row['
                duration'])
            if m == None:
                print 'Unable to parse duration %s' % row['duration']
            else:
                row['duration_sec'] = int(m.group(1)) * 3600 + int(m.group(2)) *
                    60 + int(m.group(3))

        out.writerow(row)

        count += 1
        if count % 50000 == 0:
            print '%s . . . ' % count,

print '%s total rows processed' % count
```

A.1.3 expandFileWithXY.py

Station locations are not included in the trip history feed, only their names and numbers. This script takes a trip history file and adds spatial coordinates of the stations, based on the locations of stations in the CaBi XML API. It also projects the location to Universal Transverse Mercator so that Euclidean geometry can be used in calculations.

```
#!/usr/bin/python
# Add X and Y coordinates to a Capital Bikeshare trip history file

# Copyright (C) 2013 Matthew Wigginton Conway.

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#     http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import csv
from xml.dom.minidom import parse
from sys import argv
from pyproj import Proj, transform

sourceProj = Proj(init='epsg:4326')
# WGS 84 UTM Zone 18N meters
destProj = Proj(init='epsg:32618')

# process arguments

inCsvFileName = argv[1]
xmlFileName = argv[2]
outCsvFileName = argv[3]

# Build the station database
stationFile = parse(xmlFileName)

stations = dict()

# convenience
def getTagContents(parent, tagName):
    return station.getElementsByTagName(tagName)[0].firstChild.nodeValue

for station in stationFile.getElementsByTagName('station'):
    stId = getTagContents(station, 'terminalName')

    # todo: project
    lat = float(getTagContents(station, 'lat'))
    lon = float(getTagContents(station, 'long'))

    x, y = transform(sourceProj, destProj, lon, lat)

    stations[stId] = dict(
        x=int(round(x)),
        y=int(round(y))
        )

# read CSV
inCsv = csv.DictReader(open(inCsvFileName))
outFieldNames = [n for n in inCsv.fieldnames]
outFieldNames.append('start_x')
outFieldNames.append('start_y')
outFieldNames.append('end_x')
outFieldNames.append('end_y')
outCsv = csv.DictWriter(open(outCsvFileName, 'w'), outFieldNames)
outCsv.writeheader()

unmatched = 0
count = 0
```

```
try:
    for row in inCsv:
        try:
            row['start_x'] = stations[row['start_terminal']]['x']
            row['start_y'] = stations[row['start_terminal']]['y']
            row['end_x'] = stations[row['end_terminal']]['x']
            row['end_y'] = stations[row['end_terminal']]['y']

        except KeyError:
            unmatched += 1
            row['start_x'] = ''
            row['start_y'] = ''
            row['end_x'] = ''
            row['end_y'] = ''

        outCsv.writerow(row)

        count += 1
        if count % 50000 == 0:
            print '%s trips processed' % count


finally:
    print '%s / %s trips unmatched' % (unmatched, count)
```

A.1.4 findTripsNoCoords.py

This script simply counts the number of trips with no coordinates.

```
#!/usr/bin/python
# Count the number of trips with no geo coordinates

# Copyright (C) 2013 Matthew Wigginton Conway.

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#    http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from csv import DictReader
```

```
r = DictReader(open('all-trips.csv'))

nospatial = 0
i = 0
for line in r:
    i += 1
    if i % 100000 == 0:
        print '%s (%s) . . . ' % (i, nospatial)

    if line['start_x'] == '' or line['end_x'] == '' or\
            line['start_y'] == '' or line['end_y'] == '':
        nospatial += 1

print
print 'Trips with no spatial coordinates: %s' % nospatial
```

A.2 Effects of Time on Bikeshare Use

This section contains scripts used to evaluate the effects of time on bikeshare use.

A.2.1 periods.R

This code simply defines how the time periods are coded. They are stored as single numbers in the file to reduce the file size.

```
# Time periods

# Copyright (C) 2013 Matthew Wigginton Conway.

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#     http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# We use the same time-of-day spans as used by MWCOG in their four-step model,
    page 14:
# Metropolitan Washington Council of Governments. User's Guide for the MWCOG/
    NCRTPB Travel
# Forecasting Model, Version 2.3, Build 52: Draft Report. Washington, 2013.
# http://www.mwcog.org/transportation/activities/models/files/V2.3.52_Users_Guide_
    v2_w_appA.pdf.
```

```
period.all <- 1:8
period.wkmorn <- 1
period.wkmid <-2
period.wkeve <- 3
period.wknight <- 4
period.wemorn <- 5
period.wemid <- 6
period.weeve <- 7
period.wenight <- 8
period.count <- 8
```

A.2.2 load_data.R

This code loads and filters the trip data from CaBi, dropping trips over 20 km or 24 hours.

```
# load-data.R: Abstract data loading

# Copyright (C) 2013 Matthew Wigginton Conway.

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#    http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# If the argument is true, then load just a 1 percent sample of the data
load_data <- function(sample=F) {
  if (sample) {
    data_raw <- read.csv('data/sample-trips-.1pct.csv')
  }
  else {
    data_raw <- read.csv('data/all-trips.csv')
  }

  # grab only trips 2 hours or less
  data_subset <- subset(data_raw, data_raw["duration_sec"] <= 7200)

  # and 20 km or less
  # This also implicitly removes trips that are missing start and/or end
  # coordinates; the sqrt function will return NA and the subset function
  # will exclude that record.
  data_subset <- subset(data_subset, sqrt((data_subset["start_x"] - data_subset["
      end_x"])^2
                                            + (data_subset["start_y"] - data_subset[
                                               "end_y"])^2)
                          <= 20000)

  # Remove trips with null start times
  data_subset <- subset(data_subset, data_subset["start_date"] != '')

  return(data_subset)
}
```

A.2.3 labeling.R

This code takes the Capital Bikeshare trip data, cleaned by the above scripts, and labels it based on trip start time period.

```
# Crosstab the trips by origin, destination and time of day, and compare the test
    statistics

# Copyright (C) 2013 Matthew Wigginton Conway.

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#    http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# Constants
TIMEZONE="America/New_York"

# hiram, hortense
setwd('/media/matthewc/Excelsior/GEOG172/bikeshare-analysis')
# Starlab
#setwd('E:/GEOG172/bikeshare-analysis')

source('analysis/load_data.R')
source('analysis/periods.R')

data <- load_data(sample=F)

label <- function(rawDate) {
  date <- strptime(rawDate, format='%m/%d/%Y %H:%M', tz=TIMEZONE)

  # first check the time
  hr <- date$hour

  if (hr >= 6 && hr < 9) {
    period <- period.wkmorn
  } else if (hr >= 9 && hr < 15) {
    period <- period.wkmid
  } else if (hr >= 15 && hr < 19) {
    period <- period.wkeve
  } else {
    period <- period.wknight
  }

  # check for weekend
  if (date$wday == 0 || date$wday == 6) {
    period <- period + 4
  }

  # Apply correction for Friday nights (weekends) and Sunday nights (weekday)
  if (date$wday == 0 && hr >= 19) {
    period <- period.wknight
  } else if (date$wday == 5 && hr >= 19) {
    period <- period.wenight
  }

  return(period)
}

# label all of the data
# It would seem simple to just use vapply, but that crashes R. So we chunk it into
    groups of
# 10000
# data$label <- apply(data, )
dataLen <- length(data$start_date)
# Writing directly into a data frame is really slow
data_label <- rep(NA, dataLen)

# don't store a huge vector that is really just an index
i <- 1
while (i <= dataLen) {
  if (i %% 100000 == 0)
    cat(i, ' . . . . ')
  data_label[i] <- label(data$start_date[i])

  i <- i + 1
}
```

```
data$label <- data_label

# store labeled data
write.csv(data, file="data/data-cleaned-labeled.csv")
```

A.2.4 relabel.R

This script calculates test statistics for the observed distribution of trips then randomizes the distribution of trips to time periods, conducting a Monte Carlo simulation to determine the p-value of the observed trip distribution.

```
# Relabel the data and test for an effect of time

# Copyright (C) 2013 Matthew Wigginton Conway.

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#    http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# Number of simulations for the relabel
NSIMS = 999

source('analysis/periods.R')

# This builds a 3-dimensional Xtab, with x being the label, Y being the start
   station, and Z being the end station
buildTripMatrix <- function(data) {
  return(xtabs(~label + start_terminal + end_terminal, data))
}

# This randomizes the order of the character vector passed to it
randomizeOrder <- function(vector) {
  # We order by random numbers
  orderBy <- runif(length(vector))
  return(vector[order(orderBy)])
}

# This computes the test statistic for two matrices
calcTs <- function (observed, expected) {
  return(sum((observed - expected)^2)/sum(expected)^2)
```

```
}

# This computes pairwise test statistics for each time period relative to every
   other
computePairwiseStats <- function (tripMatrix) {
  # Build a matrix for the test statistics
  pairwiseStats <- matrix(NA, nrow=period.count, ncol=period.count)

  # compare each one to all others
  for (i in period.all) {
    for (j in period.all) {
      obs <- tripMatrix[i,,]
      ex  <- tripMatrix[j,,]

      # scale so sums are same
      pairwiseStats[i,j] <- calcTs(obs, ex * (sum(ex)/sum(obs)))
    }
  }

  return(pairwiseStats)
}

data <- read.csv('data/data-cleaned-labeled.csv')
orig <- buildTripMatrix(data)

origTS <- computePairwiseStats(orig)

# Make an array to store the test statistics in
simulatedTS <- array(NA, dim=c(NSIMS, period.count, period.count))

for (i in 1:NSIMS) {
  cat('Repetition', i, '\n')

  # Randomize the labels, preserving the marginals
  data$label <- randomizeOrder(data$label)

  tripMatrix <- buildTripMatrix(data)

  # Compute the pairwise stats and store them
  simulatedTS[i,,] <- computePairwiseStats(tripMatrix)
}

# Find the p-values
pvals <- matrix(NA, nrow=period.count, ncol=period.count)
for (i in period.all) {
  for (j in period.all) {
     pvals[i,j] <- sum(simulatedTS[,i,j] >= origTS[i,j]) / (NSIMS + 1)
  }
```

```
}

# Write a CSV file that is used to generate the table in the TeX writeup
write.csv(pvals, 'writeup/pvals.csv')
```

A.3 Effects of Space on Bikeshare Use

A.3.1 morans_i.R

This script was used to calculate Moran's I to calculate how spatially autocorrelated the popularity of bikeshare stations is.

```
# Calculate Moran's I for bike share data; are station popularities autocorrelated
   ?

# Copyright (C) 2013 Matthew Wigginton Conway.

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#    http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

TIMEZONE <- 'America/New_York'
BREAK_LINKS_OVER <- 4000 # 4000 m, empirically determined; much less than 4000m
   and we start seeing more than
                          # two clusters, and we really do want to keep it
                             clustered to just DC/Arlington/Alexandria
                          # and Montgomery County.
EXCLUDE_STATIONS <- c('32004', '32009', '32005', '32007') # Exclude the four
   stations in Montgomery County
# StarLab
setwd('E:/GEOG172/bikeshare-analysis')
library(plyr)
library(spatstat)
library(spdep)
library(AID)

data <- read.csv('data/data-cleaned-labeled.csv')

# First, calculate station popularities
first <- function (vect) { return(vect[1]) }
```

```
# This function returns the time as seconds since January 1, 1970
epoch <- function (date) {
  return(as.integer(as.POSIXct(date)))
}

# Calculate the lowest date in a vector of dates. Return an epoch time.
minDate <- function (rawDates) {
  lowest <- NA

  for (rawDate in rawDates) {
    date <- strptime(rawDate, format='%m/%d/%Y_%H:%M', tz=TIMEZONE)

    if (is.na(lowest)) {
      lowest <- epoch(date)
    }
    else if (date < lowest) {
      lowest <- epoch(date)
    }
  }

  return(lowest)
}

# Calculate the largest date in a vector of dates. Return an epoch time.
maxDate <- function (rawDates) {
  greatest <- NA

  for (rawDate in rawDates) {
    date <- strptime(rawDate, format='%m/%d/%Y_%H:%M', tz=TIMEZONE)

    if (is.na(greatest)) {
      greatest <- epoch(date)
    }
    else if (date > greatest) {
      greatest <- epoch(date)
    }
  }

  return(greatest)
}

# pop can be interpreted as trips per day, iff you're using population data
# First we summarise the bike movements at the start and end of trips
popularityDest <- ddply(data, c('end_terminal'), summarise,
                                  # No need to have x's and y's here, they come from
                                      orig
                        NDest=length(end_x),
                        firstDest=minDate(start_date),
                                     lastDest=maxDate(start_date)
)

popularityOrig <- ddply(data, c('start_terminal'), summarise,
                             x=first(start_x),
                             y=first(start_y),
                             NOrig=length(start_x),
                             firstOrig=minDate(start_date),
                                     lastOrig=maxDate(start_date)
)

# Now combine the origins and the destinations
# Full inner join; remove any station that doesn't have both trip origins and trip
     terminations
popularity <- merge(popularityOrig, popularityDest, all=F, by.x='start_terminal',
   by.y='end_terminal')
popularity <- rename(popularity, c("start_terminal"="terminal"))

# Remove stations
popularity <- subset(popularity, !(terminal %in% EXCLUDE_STATIONS))

attach(popularity)

# Sum up the number of bike movements and divide by the number of days the station
     is open
# First, calculate span
# The matrix bit gets the first movement recorded at the station, either an origin
     or a destionation.
# We can use min/max because all times are represented as seconds since Jan 1,
   1970 at this point.
lastOpDate <- apply(matrix(c(lastOrig, lastDest), ncol=2), 1, max, na.rm=T)
firstOpDate <- apply(matrix(c(firstOrig, firstDest), ncol=2), 1, min, na.rm=T)
spans <- (((lastOpDate - firstOpDate) / 86400) + 1)
popularity$pop <- (NOrig + NDest) / spans

# Normalize the popularity as much as possible
# We use the Shapiro-Wilk method because it gave a number close to the average
# when used on sample data. Need to check with Dr. Sweeney regarding the best
# way to choose a method.
bctransform <- function (data, lambda) {
  if (lambda == 0) {
    return(log(data))
  }
  else {
    return((data^lambda - 1)/lambda)
  }
}
bclam <- boxcoxnc(popularity$pop, method='sw')
```

```
popularity$bcpop <- bctransform(popularity$pop, bclam$result[1])
cat('Box-Cox p-values:', bclam$result[2:4,])

detach(popularity)

# save the popularities to avoid recalculation later
write.csv(popularity, 'data/station-popularities.csv')
# Re-read popularities: start from here if you're repeating the analysis.
popularity <- read.csv('data/station-popularities.csv')

attach(popularity)

# Make a plot for the writeup showing why we used Box-Cox
layout(matrix(1:2, 1, 2))
hist(pop, main=NA, xlab='Bike movements/day', ylab='Number of stations')
hist(bcpop, main=NA, xlab=paste('Box-Cox transformed bike movements/day (lambda=',
    bclam$result[1], ')', sep=''), ylab='Number of stations')
graphics.off()

# build the neighbor matrix
nbmat <- tri2nb(popularity[,c('x','y')], row.names=terminal)

# Drop really long links
for (i in 1:length(nbmat)) {
  newNb <- c()
  for (j in nbmat[[i]]) {
    dist <- sqrt((x[i] - x[j])^2 + (y[i] - y[j])^2)
    if (dist <= BREAK_LINKS_OVER) {
      newNb <- c(newNb, as.integer(j))
    }
  }
  nbmat[[i]] <- newNb
}

weights <- nb2listw(nbmat, style='W')

# Plot the triangulation
plot(weights, coords=popularity[,c('x','y')], main="Station adjacency")

# labels
text(locator(), labels=c('Washington', 'Arlington', 'Alexandria'))

graphics.off()

# Calulate moran's I
moran.test(bcpop, weights)
moran.plot(bcpop, weights, xlab='Box-Cox transformed bike movements/day', ylab='
    Spatially lagged transformed bike movements/day')
```

A.3.2 ui.R

This script is the UI for the Shiny application used to evaluate the link threshold distance (see figure 3).

```
# Copyright (C) 2013 Matthew Wigginton Conway.

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#    http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

library(shiny)

shinyUI(pageWithSidebar(
  headerPanel("Link Trimming Distance"),
  sidebarPanel(
      sliderInput('BREAK_LINKS', 'Break links longer than (m):', min=100, max
         =50000, value=4000, step=100)
    ),
  mainPanel(plotOutput('links'))
  ))
```

A.3.3 server.R

This is the server component of the Shiny application used to evaluate the link threshold distance.

```
# Copyright (C) 2013 Matthew Wigginton Conway.

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#    http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

library(shiny)
library(spdep)
```

```
# Load the data and build the initial triangulation
setwd("/media/matthewc/Excelsior/GEOG172/bikeshare-analysis")
stations <- read.csv('data/stations.csv')
attach(stations)
globalNbmat <- tri2nb(stations[,c('x','y')])

shinyServer(function(input, output) {
  output$links <- renderPlot({
    nbmat <- globalNbmat
    # Drop really long links
    for (i in 1:length(nbmat)) {
      newNb <- c()
      for (j in nbmat[[i]]) {
        dist <- sqrt((x[i] - x[j])^2 + (y[i] - y[j])^2)
        if (dist <= input$BREAK_LINKS) {
          newNb <- c(newNb, as.integer(j))
        }
      }
      nbmat[[i]] <- newNb
    }

    weights <- nb2listw(nbmat, style='W')

    plot(weights, stations[,c('x','y')])
  })
})
```