Capstone ReadMe

Kevin Crossley

This ReadMe describes the steps involved in completing the DevUp Data Science Capstone Project, as completed in CrossleyCapstone.py.

Table of Contents

[Script Structure 2](#_Toc517125311)

[Import Libraries 2](#_Toc517125312)

[Source Bike Data 2](#_Toc517125313)

[Create Graph of Manhattan 2](#_Toc517125314)

[Source Weather Data 2](#_Toc517125315)

[Combine Bike Data 3](#_Toc517125316)

[Initial Analysis and Visualization 3](#_Toc517125317)

[Basic Statistics 3](#_Toc517125318)

[Question 1 4](#_Toc517125319)

[Question 2 5](#_Toc517125320)

[Question 3 6](#_Toc517125321)

[Question 4 7](#_Toc517125322)

[Question 5 9](#_Toc517125323)

[Create and Evaluate Prediction Model 11](#_Toc517125324)

[Data Load Script 12](#_Toc517125325)

[Clean and Prepare Data 12](#_Toc517125326)

[Call Functions to Run Results 13](#_Toc517125327)

# Script Structure

Generally, my Capstone script is structured as separate functions that complete the major tasks of the project. Some universal steps (required by all tasks) are accomplished in the script itself near the end of the file (See: *Data Load Script* and *Clean and Prepare Data*).

In this way, I do not have to run through each task every time I run the script; I can select which question or task I am currently developing or interested in, and run it. The functions are organized in essentially chronological order. The first task required of the project comes first in the script. The following sections describe the steps in each task.

# Import Libraries

The first step in any script is to import any external libraries to be used. In this project, I relied on BeautifulSoup and Requests for aquiring data, NumPy and Pandas for data structure and manipulation, MatPlotLib, seaborn, OSMNX, and NetworkX for visualization, and SKLearn for modelling.

# Source Bike Data

The first step in this project was to gather the required Citi Bike Data. In the save\_data() function, I download zip files from the Citi Bike data archive website. The urls follow a simple format, and I use a for loop to run through the 12 months of data, download the zip file, and extract the data to a csv.

# Create Graph of Manhattan

To draw the visualizations I had in mind, the next data I needed to download was a street map of Manhattan. In the manhattan\_data() function, I use the OMSNX library to download drivable street data from Open Street Map and save it as a mathematical graph (a series of nodes and connections). This is saved to my local disk to avoid making API requests every time I run the script.

# Source Weather Data

The final data I used for this project was historic New York City weather data. I found a weather website that reported 2017 weather information in a data table. In the scrape\_weather() function, I used Beautiful Soup to extract the relevant data from the webpage. I used a triple-nested for loop to accomplish this.

The first step was to create the 12 urls needed to get each month of historic data. Then, the first level of the extraction loop cycles through each url. The next level makes the GET request to the webpage, and stores the site as a Beautiful Soup object. Then I select only the data table in question (obsTable) and find the table body elements within it. The second loop cycles through each body element to find each row of the table, and the third loop cycles through each row to capture each data element. Each data element is then appended to a list, which is then converting that list to a pandas data frame, and saving to csv for easy access.

# Combine Bike Data

The final data preparation step occurs in the combine\_data() function. Here, each month of Citi Bike data is read and appended to the January file. I create strings representing each month’s filepath, then read in each file to a data frame, then append each data frame to January. This creates one csv file with all the bike data.

# Initial Analysis and Visualization

In this section of the script, I define individual functions to calculate some basic statistics, and to answer each project question (1-5).

## Basic Statistics

With basic\_stats(df), I calculate and print out basic statistics of the data set.

1. Total Rides – this is calculated by simply calling the df.shape method, which returns the number of rows (equal to the number of individual rides) in the data set.
2. Total Ride Time – this is calculated by simply taking the sum of the entire ‘Trip Duration’ row. I convert to years for easier comprehension.
3. Total Bikes – This item uses the nunique() method to calculate the number of unique Bike IDs
4. Total Uses – This is used to calculate the percentage of rides by subscribers and customers. I use the value\_counts() method to count the number of each type of user.
5. Subscriber Percentage – simply number of subscribers divided by total uses
6. Customer Percentage – same but for customers.

|  |  |
| --- | --- |
| Total Rides | 16,364,660 |
| Total Ride Time | 516 years |
| Total Bikes | 14,204 |
| % of Rides by Subscribers | 89% |
| % of Rides by Customers | 11% |

## Question 1

Question 1 asks me to calculate and visualize the top 5 stations with the most starts. I accomplish this in the question1(df, G) function. This is easy to calculate with the value\_counts() method. Then, since I calculated the top stations using the Start Station ID column, I need to correlate that ID with the address (for presentation), and latitude and longitude (for visualization). I loop through my top 5 list of stations and find the first station in the data set with a matching ID, then grab the necessary info and append to a new data frame.

Next, I use the OSMNX library to create a plot of every street in Manhattan, and place a scatter plot of the station coordinates on top of it. This achieves a visualization that shows the top 5 stations on a map.

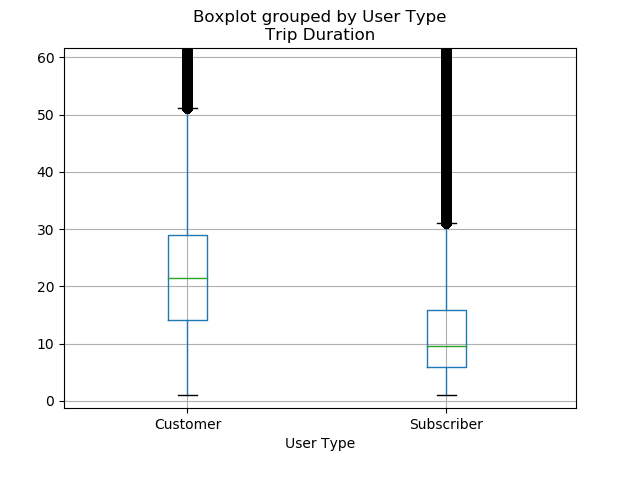
|  |  |  |  |
| --- | --- | --- | --- |
| **Rank** | **Station Area** | **Station Address** | **Count** |
| 1 | Grand Central | Pershing Square North | 162,716 |
| 2 | Union Square | E 17 St & Broadway | 112,218 |
| 3 | Madison Square | Broadway & E 22 St | 108,590 |
| 4 |  | W 21 St & 6 Ave | 107,133 |
| 5 | Hudson Greenway | West St & Chambers St | 105,610 |



## Question 2

For Question 2, I am asked to understand trip durations by the two different types of users: Customers (one-time payment users), and Subscribers (monthly payments). In the question2(df) function, first I drop any row in the data set that has an unknown user type and convert Trip Duration to minutes. The pandas groupby() method lets me easily calculate basic statistics of Trip Duration, grouped by the two user types. I use the pandas box plot function to easily create a box plot of the data, and zoom in to get a better understanding of the information without outliers.

|  |  |  |
| --- | --- | --- |
|  | **Customer** | **Subscriber** |
| Count | 1,769,423 | 14,579,325 |
| Mean | 42 min | 14 min |
| STD | 667 min | 166 min |
| Min | 1 min | 1 min |
| 25% | 14 min | 6 min |
| 50% | 21 min | 10 min |
| 75% | 29 min | 16 min |
| Max | 111 days | 113 days |

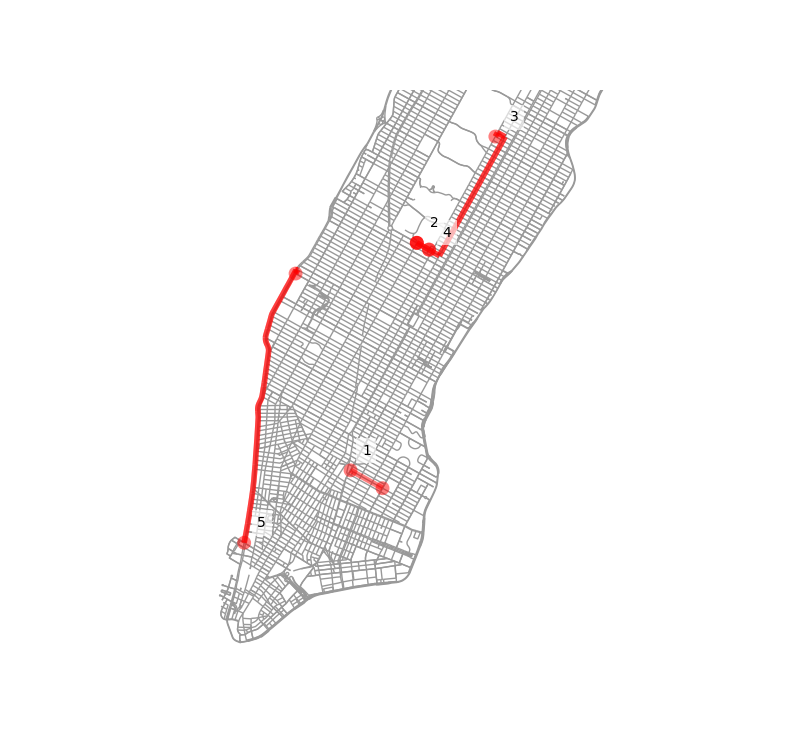


## Question 3

Question 3 asks that I calculate and visualize the most popular trips. The question3(df, G) function includes a modified function plot\_graph\_route() from the imported OSMNX library. This function is copied directly from the source code and is slightly modified on the last line to avoid plotting the route immediately, and instead simply return the line collection to be plotted. This simple modification allows me to stack multiple routes on one map of Manhattan.

I begin this function by using groupby() again to count unique pairs of start and end stations. I use the same method as question 1 to source the name and coordinates of the station pair. This lets me merge the two tables together into a data frame that contains the count as well as name and coordinates. Then, I use the get\_nearest\_node() function to find the nearest node on the graph of Manhattan (street intersection) to each start and end station. I then calculate the shortest path between then and use my modified plot function to return all 5 paths. Then I add the routes to the plot and annotate them with rank numbers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rank** | **Area** | **Start Address** | **End Address** | **Count** |
| 1 | East Village | E 7 St & Avenue A | Cooper Square & E 7 St | 7994 |
| 2 | Central Park | Central Park S & 6 Ave | Central Park S & 6 Ave | 7169 |
| 3 | Central Park | Central Park S & 6 Ave | 5 Ave & E 88 St | 6318 |
| 4 | Central Park | Grand Army Plaza & Central Park S | Grand Army Plaza & Central Park S | 5670 |
| 5 | Hudson Greenway | 12 Ave & W 40 St | West St & Chambers St | 5403 |



## Question 4

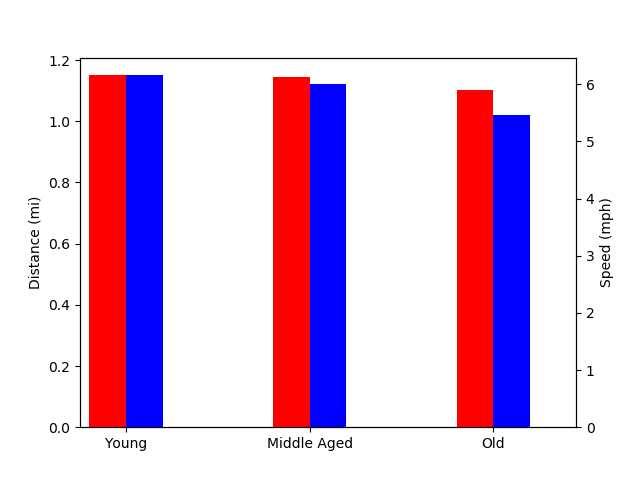
Question 4 analyzes trip distance and speed by age and gender. To do this, I bucketed the ages into three categories: Young (0-30), Middle Aged (31-50), and Old (50-118). I calculated the ages themselves in the universal script (see *Clean and Prepare Data*).

The challenge with this question was how to most efficiently calculate the distance between two stations. After I set up the list of coordinates, I use a for loop to apply the distance.distance() function from the GeoPy library. I attempted to use other functions from the OSMNX library, but the speed gains from those attempts resulted in data so inaccurate that the distances were often being calculated as 0. Therefore, this method was the only accurate one I could find. The run time is **extremely** long. In the future, I would spend much more time investigate a more efficient way of calculating distance. One final problem is that I would encounter an error every time I tried to save the calculated distance column to csv. For this reason, I did not have access to this data column when building my predictive model, which I believe hurt its accuracy.

After I append the distance data to the data set, it is easy to calculate speed as distance divided by Trip Duration. Then I can calculate the mean of both speed and distance columns for my age buckets and gender using the groupby method again.

The final step is to plot the data as a bar chart with multiple axes using matplotlib.

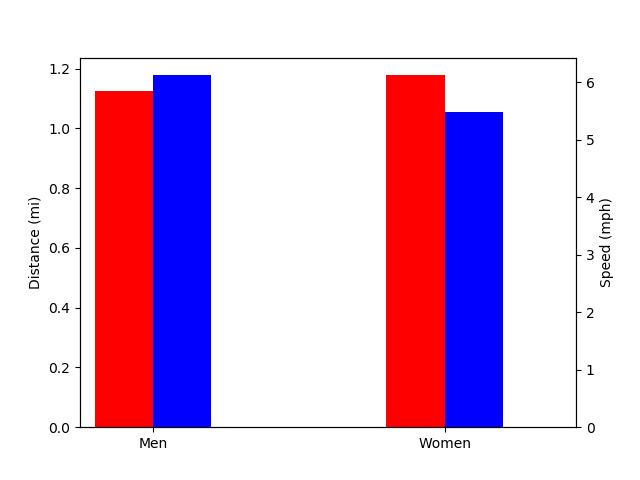
|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Young** | **Middle Aged** | **Old** |
| Distance (mi) | 1.15 | 1.14 | 1.10 |
| Speed (mph) | 6.16 | 6.00 | 5.46 |



speed

distance

|  |  |  |
| --- | --- | --- |
| **Metric** | **Men** | **Women** |
| Distance (mi) | 1.13 | 1.18 |
| Speed (mph) | 6.12 | 5.49 |



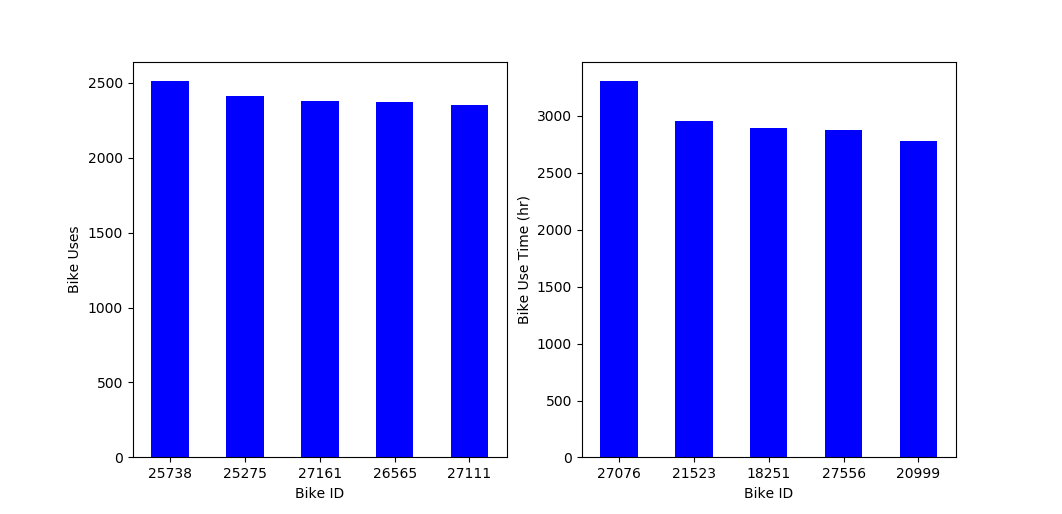
speed

distance

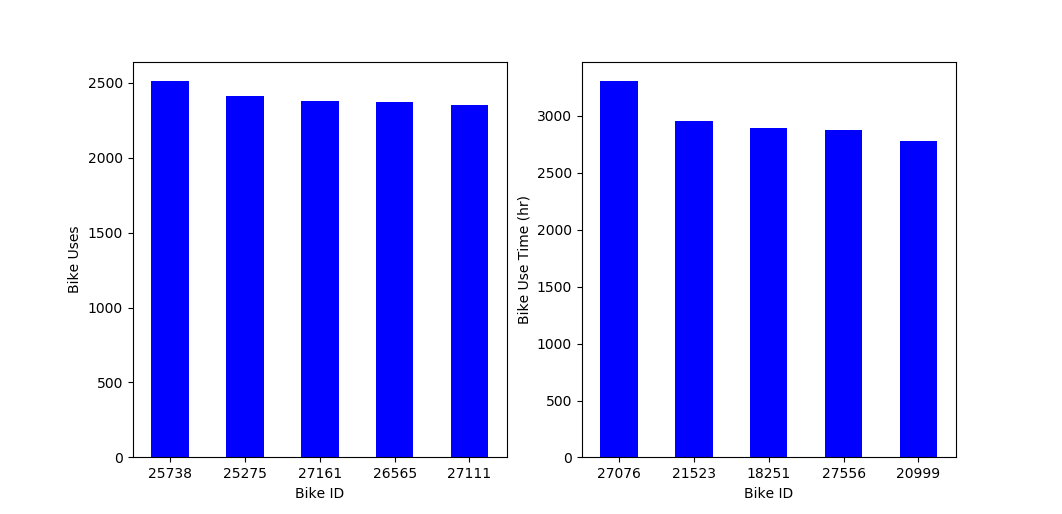
## Question 5

Question 5 analyzes the busiest bike. I calculated “busiest” using 2 different metrics: number of uses and total use time. The first method uses the value\_counts() method used often in this project. Calculating bike use time uses groupby() on only the Trip Duration column and then sums up the total Trip Duration time for each Bike ID. The final step is a simple bar graph for each metric.

|  |  |  |
| --- | --- | --- |
| **Rank** | **Bike ID** | **Uses** |
| 1 | 25738 | 2514 |
| 2 | 25275 | 2409 |
| 3 | 27161 | 2376 |
| 4 | 26565 | 2370 |
| 5 | 27111 | 2349 |



|  |  |  |
| --- | --- | --- |
| **Rank** | **Bike ID** | **Time in Use (hr)** |
| 1 | 27076 | 3310 |
| 2 | 21523 | 2958 |
| 3 | 18251 | 2893 |
| 4 | 27556 | 2880 |
| 5 | 20999 | 2782 |



# Create and Evaluate Prediction Model

In this section, I define the model1(df\_model, f) function to hold the code I use to build the predictive travel time model. This initial model uses many basic techniques to construct a simple linear regression based on only significant variables. The first step is to standardize the columns to the range 0 to 1 to ensure bias-free modelling. I standardize Start Time by creating a column representing the time as minutes after midnight, then scale by total minutes in a day to get the required range.

Next, I create one-hot columns for weather event, gender (which wound up unused), and User Type. The get\_dummies() method creates new columns with a 0 if the row has the attribute and a 1 if it does. This allows categorical data to be used in the regression.

After dropping unnecessary rows, I scale Age, High Temperature, and Humidity (unused) by their maximum values to fit them to the correct range. The final caclulation before building the model is to calculate the median trip time between any two stations. This uses the groupby method and the transform(‘median’) method to create a new column containing this information. I also drop any row where the start and end station are equal, as a trip of 0 distance will be very unreliable to model.

To build the model, I first create a smaller random sample of the data (with the fraction of the total data set to use given by f in the model1 input) to allow for rapid testing. Then I assign Trip Duration as the target variable and the rest of the relevant columns (Median Trip Time, User Type, and High Temperature) to the predictor variable. Then I use the sklearn train\_test\_split function to split the data set into a testing set (where the model knows the correct answer) and a training set (where the model will predict a value without knowing the correct one). The final step is to actually create the model by creating a Linear Regression object, fitting the data, and scoring the predictions. In my testing, more complicated sklearn models like Lasso or Support Value Regression took prohibitively long to run.

The final step is to evaulate the performance by calculating the cross value score. The cross value score computes an R2 value using different pieces of the data set as the test and train groups in order to gives a more realistic understanding of true model performance. I also create a seaborn regplot using the Trip Duration and Median Travel Time data to quickly visualize how well a fit I created.

Model Results

* Average Cross Value Score: 69%
* Model Components
  + Median Trip Time, User Type (Subscriber/Customer), Daily High Temperature

As I mentioned with question 4, the distance between each station pair data was unable to be saved to disk, and therefore was unavailable for this model. With more time, I would isolate the save to\_csv bug to ensure access to this valuable data.

# Data Load Script

The actual script that runs every time CrossleyCapstone is called begins in this section of the code (line 950). Here, I load the complete data set from file (with the option of only grabbing n number of rows for quicker iteration and testing). I also load the saved graph of Manhattan and the weather data csv. In this case, the weather data was easier to manipulate to the desired form in Excel, so I load the excel-modified csv that rearranged the data scraped from the web into a data stucture that was easy to use.

# Clean and Prepare Data

The script that runs every time continues in this section. Here, I convert the data frame columns into the correct type (category, pandas datetime format, etc.). In order to prep the bike and weather data sets for merging, I need to create a column of only the date of a ride, not the time.

I also create the ‘Age’ column here. First I drop any row without an age (mostly the customer data). Then, when I reviewed the summary statistics of the Birth Year column, I noticed several data elements from the 1800s. Since few, if any, humans cabale of riding a bike were born in the 1800s, I assumed these were user error and the user mistyped 8 instead of 9. Therefore, I add 100 to each birth year in a new column, then replace only those years less than 1900 with the modified year. Age is calculated by subtracting 2017 from the modified birth year, and the temporary columns are dropped.

The next step in this script is to prepare the weather data for merging, again by assigning the correct type and creating a column of just date, not time. Finally, the data set used for the predictive model is created by merging daily high temperature, humidity, and weather event data with the date of each ride in the data set. Columns not needed for the analysis are dropped.

# Call Functions to Run Results

In this final section of the script, I can call any of the functions as I test and iterate through the project. This saves time and avoids calculating values I already have saved.