2_Boston_Blue_Bikes_Feb2020

February 26, 2020

In this project, I look at all bike rides taken by Boston Blue Bike subscribers in 2019, with the goal of determining if the infrastructure is optimally set up. Bike stations are setup throughout the city with a varying number of docks at each station, where one dock holds one bike. The system can only run efficiently and be successful if subscribers can rent a bike when and where they want to and don't have to worry about having a dock to drop it off when they are done. Since certain stations obviously see more activity and certain routes are traveled more, optimally distributing docks to ensure riders will always have a bike available to rent or an open dock to leave a bike after a ride is critical.

Start with package imports:

[1]:
```
import pandas as pd
import numpy as np
import os
import networkx as nx
import matplotlib as mpl
import matplotlib.pyplot as plt
```

This function reads in the data - one file per month. It returns the data compiled into one dataframe (and a copy of the dataframe in case the original gets corrupted in the manipulations).

[2]:
```
def load_data():
    '''
    Parameters
    ----------
    None
        Loads data into dataframe and returns df and df_copy

    Returns
    -------
    df, copy of df

    '''
    df = pd.DataFrame()

    for file in os.listdir('Data'):
        if '2019' in file:
            print('...Reading file "{}"...'.format(file))
            df = df.append(pd.read_csv('Data/' + file,
```

```
                                         parse_dates = ['starttime',
                                                        'stoptime']))
    df = df.reset_index(drop = True)

    return df, df.copy()
```

[3]:
```
df, df_copy = load_data()
```

```
…Reading file "201901-bluebikes-tripdata.csv"…
…Reading file "201902-bluebikes-tripdata.csv"…
…Reading file "201903-bluebikes-tripdata.csv"…
…Reading file "201904-bluebikes-tripdata.csv"…
…Reading file "201905-bluebikes-tripdata.csv"…
…Reading file "201906-bluebikes-tripdata.csv"…
…Reading file "201907-bluebikes-tripdata.csv"…
…Reading file "201908-bluebikes-tripdata.csv"…
…Reading file "201909-bluebikes-tripdata.csv"…
…Reading file "201910-bluebikes-tripdata.csv"…
…Reading file "201911-bluebikes-tripdata.csv"…
…Reading file "201912-bluebikes-tripdata.csv"…
```

[4]:
```
df.shape
```

[4]:
```
(2522537, 15)
```

2,522,537 rides were taken in 2019 - that's a really amazing number of rides taken in just one year! Let's also take a peak at the dataframe as well as one unique ride to get an idea of what the data looks like.

A couple things to note about the dataset that aren't immediately obvious: - `tripduration` is measured in seconds - in the `gender` column, 0 = "Prefer not to say", 1 = "Male", 2 = "Female" - stations have both a numeric ID and a name. This will become important later

[5]:
```
df.head()
```

[5]:
```
   tripduration               starttime                stoptime  \
0           371 2019-01-01 00:09:13.798 2019-01-01 00:15:25.336
1           264 2019-01-01 00:33:56.182 2019-01-01 00:38:20.880
2           458 2019-01-01 00:41:54.600 2019-01-01 00:49:33.273
3           364 2019-01-01 00:43:32.571 2019-01-01 00:49:37.426
4           681 2019-01-01 00:49:56.464 2019-01-01 01:01:17.701

   start station id                         start station name  \
0                80     MIT Stata Center at Vassar St / Main St
1               117                       Binney St / Sixth St
2                68       Central Square at Mass Ave / Essex St
3                89   Harvard Law School at Mass Ave / Jarvis St
4                73     Harvard Square at Brattle St / Eliot St
```

```
   start station latitude  start station longitude  end station id  \
0               42.362131               -71.091156             179
1               42.366162               -71.086883             189
2               42.365070               -71.103100              96
3               42.379011               -71.119945             334
4               42.373231               -71.120886             367

                                    end station name  end station latitude  \
0                                    MIT Vassar St             42.355601
1                                         Kendall T             42.362428
2  Cambridge Main Library at Broadway / Trowbridg…            42.373379
3                        Mass Ave at Hadley/Walden             42.391210
4                        Vassal Lane at Tobin/VLUS             42.383932

   end station longitude  bikeid    usertype  birth year  gender
0             -71.103945    3689  Subscriber        1987       1
1             -71.084955    4142  Subscriber        1990       1
2             -71.111075    1628  Subscriber        1977       1
3             -71.122608    2969  Subscriber        1993       1
4             -71.139613    3469  Subscriber        1979       2
```

Example of one ride:

[6]:
```
df.iloc[0]
```

[6]:
```
tripduration                                            371
starttime                        2019-01-01 00:09:13.798000
stoptime                         2019-01-01 00:15:25.336000
start station id                                         80
start station name          MIT Stata Center at Vassar St / Main St
start station latitude                              42.3621
start station longitude                            -71.0912
end station id                                          179
end station name                              MIT Vassar St
end station latitude                                42.3556
end station longitude                              -71.1039
bikeid                                                 3689
usertype                                         Subscriber
birth year                                             1987
gender                                                    1
Name: 0, dtype: object
```

The following function adds additional columns for the starting hour, starting day of the week, and starting month for each ride. This will help me get an idea of when the service is seeing the most use.

[7]:
```python
def setup_dataframe(df):
    '''
    Parameters
    ----------
    df: input dataframe.
        Adds new columns for start_hour, start_day

    Returns
    -------
    adjusted dataframe

    '''
    # Ride start hour
    print('...Collecting starting hour...')
    df['start_hour'] = df['starttime'].apply(lambda x: x.hour)

    # Ride start day
    print('\n...Collecting start day...')
    df['start_day'] = df['starttime'].apply(lambda x: x.weekday())

    # Ride start month
    print('\n...Collecting starting month...')
    df['start_month'] = df['starttime'].apply(lambda x: x.month)

    return df
```

[8]:
```python
df = setup_dataframe(df)
```

```
…Collecting starting hour…

…Collecting start day…

…Collecting starting month…
```

Take another look at the dataframe and notice the additional 3 columns that were added at the end:

[9]:
```python
df.head()
```

[9]:
```
   tripduration               starttime                stoptime  \
0           371 2019-01-01 00:09:13.798 2019-01-01 00:15:25.336
1           264 2019-01-01 00:33:56.182 2019-01-01 00:38:20.880
2           458 2019-01-01 00:41:54.600 2019-01-01 00:49:33.273
3           364 2019-01-01 00:43:32.571 2019-01-01 00:49:37.426
4           681 2019-01-01 00:49:56.464 2019-01-01 01:01:17.701

   start station id                          start station name  \
0                80     MIT Stata Center at Vassar St / Main St
```

```
1              117                         Binney St / Sixth St
2               68        Central Square at Mass Ave / Essex St
3               89  Harvard Law School at Mass Ave / Jarvis St
4               73     Harvard Square at Brattle St / Eliot St

   start station latitude  start station longitude  end station id  \
0               42.362131               -71.091156             179
1               42.366162               -71.086883             189
2               42.365070               -71.103100              96
3               42.379011               -71.119945             334
4               42.373231               -71.120886             367

                                 end station name  end station latitude  \
0                                   MIT Vassar St             42.355601
1                                       Kendall T             42.362428
2  Cambridge Main Library at Broadway / Trowbridg…           42.373379
3                         Mass Ave at Hadley/Walden             42.391210
4                         Vassal Lane at Tobin/VLUS             42.383932

   end station longitude  bikeid    usertype  birth year  gender  start_hour  \
0             -71.103945    3689  Subscriber        1987       1           0
1             -71.084955    4142  Subscriber        1990       1           0
2             -71.111075    1628  Subscriber        1977       1           0
3             -71.122608    2969  Subscriber        1993       1           0
4             -71.139613    3469  Subscriber        1979       2           0

   start_day  start_month
0          1            1
1          1            1
2          1            1
3          1            1
4          1            1
```

The `describe` function below returns summary statistics on the dataframe. There are a couple things that jump out here: - maximum trip duration is 3,581,049 seconds (995 hours or nearly 1.5 months) which doesn't seem realistic. - minimum birth year is 1886. While I would love to believe there is a 134 year old using a Blue Bike to commute to the grocery store, this again seems unrealistic.

The `remove_outliers` function therefore removes any rides over 1 day in length or any rider registering as >100 years old

[10]:
```
df.describe().T
```

[10]:
```
                              count         mean           std        min  \
tripduration              2522537.0  1471.833556  21908.019456  61.000000
start station id          2522537.0   142.295763    118.322374   1.000000
start station latitude    2522537.0    42.357457      0.055846   0.000000
```

```
start station longitude  2522537.0   -71.087947      0.093027   -71.166491
end station id           2522537.0   141.630204    118.058385     1.000000
end station latitude     2522537.0    42.357301      0.081703     0.000000
end station longitude    2522537.0   -71.087471      0.136653   -71.166491
bikeid                   2522537.0  3637.692047   1287.279049     1.000000
birth year               2522537.0  1984.724999     11.548306  1886.000000
gender                   2522537.0     1.124794      0.573825     0.000000
start_hour               2522537.0    13.862679      4.828993     0.000000
start_day                2522537.0     2.842870      1.934989     0.000000
start_month              2522537.0     7.298545      2.708392     1.000000

                                 25%          50%          75%           max
tripduration              418.000000   707.000000  1185.000000  3.581049e+06
start station id           55.000000    99.000000   190.000000  4.460000e+02
start station latitude     42.348706    42.358100    42.365994  4.241480e+01
start station longitude   -71.104412   -71.089811   -71.068922  0.000000e+00
end station id             54.000000    98.000000   190.000000  4.460000e+02
end station latitude       42.348706    42.358100    42.365994  4.241480e+01
end station longitude     -71.104412   -71.088220   -71.067811  0.000000e+00
bikeid                   2746.000000  3670.000000  4497.000000  6.173000e+03
birth year               1977.000000  1989.000000  1994.000000  2.003000e+03
gender                      1.000000     1.000000     1.000000  2.000000e+00
start_hour                 10.000000    15.000000    18.000000  2.300000e+01
start_day                   1.000000     3.000000     4.000000  6.000000e+00
start_month                 5.000000     8.000000     9.000000  1.200000e+01
```

[11]:
```python
def remove_outliers(df):
    '''
    Parameters
    ----------
    df: input dataframe.
        Removes trips >24 hours
        Removes rides with birth year <1920
        Respect to anyone 100+ still riding but...

    Returns
    -------
    Trimmed dataframe

    '''
    long_trips = df[df['tripduration'] > 24 * 60 * 60]
    too_old = df[df['birth year'] < 1920]
    df = df.drop(index = long_trips.index.append(too_old.index))
    return df
```

[12]:
```python
df = remove_outliers(df)
```

Now that I have removed the outliers, I can compare the dataframes from before and after: - mean trip duration decreases by 400+ seconds, but the standard deviation is also reduced by a factor of 10+, suggesting we removed some serious statistical outliers. - on the other hand, the mean birth year didn't change (1984) so removing the extreme birth years had minimal impact.

[13]:
```
df.describe().T
```

[13]:
```
                             count         mean          std          min  \
tripduration             2519648.0  1045.876419  1965.292020    61.000000
start station id         2519648.0   142.275172   118.315887     1.000000
start station latitude   2519648.0    42.357463     0.055872     0.000000
start station longitude  2519648.0   -71.087955     0.093076   -71.166491
end station id           2519648.0   141.642643   118.050610     1.000000
end station latitude     2519648.0    42.357300     0.081745     0.000000
end station longitude    2519648.0   -71.087478     0.136729   -71.166491
bikeid                   2519648.0  3637.908047  1287.217123     1.000000
birth year               2519648.0  1984.756855    11.454521  1923.000000
gender                   2519648.0     1.125284     0.573436     0.000000
start_hour               2519648.0    13.862478     4.827832     0.000000
start_day                2519648.0     2.842376     1.934858     0.000000
start_month              2519648.0     7.298616     2.708384     1.000000

                                 25%          50%          75%           max
tripduration              418.000000   707.000000  1183.000000  86339.000000
start station id           55.000000    99.000000   190.000000    446.000000
start station latitude     42.348706    42.358100    42.365994     42.414802
start station longitude   -71.104412   -71.089811   -71.068922      0.000000
end station id             54.000000    98.000000   190.000000    446.000000
end station latitude       42.348706    42.358100    42.365994     42.414802
end station longitude     -71.104412   -71.088220   -71.067811      0.000000
bikeid                   2746.000000  3670.000000  4498.000000   6173.000000
birth year               1977.000000  1989.000000  1994.000000   2003.000000
gender                      1.000000     1.000000     1.000000      2.000000
start_hour                 10.000000    15.000000    18.000000     23.000000
start_day                   1.000000     3.000000     4.000000      6.000000
start_month                 5.000000     8.000000     9.000000     12.000000
```

The following `get_station_ids` function associates numeric station IDs with their respective station names which helps at various points in the analysis.

[14]:
```
def get_station_ids(df):
    '''
    Parameters
    ----------
    df: dataframe to gather unique stations ids

    Returns
    -------
```

```
    dict: {station ID: station name}

    '''
    id_df = df[['start station id', 'start station name']].\
                drop_duplicates().reset_index(drop = True)
    id_dict = {id_df['start station id'].loc[i]:
               id_df['start station name'].loc[i]
               for i in range(id_df.shape[0])}

    return id_dict
```

The next function is used to find the top `n` starting and ending stations. It can then either print out the list (default: `show = True`) to be viewed or not. Let's see a list of the top 5 starting/ending stations based on rides taken in 2019 as an example.

[15]:
```
def top_stations(df, n, show = True):
    '''
    Parameters
    ----------
    df: input dataframe
    n: number of top stations to look at
        Finds top starting and ending stations
        Plots top starting and ending stations
    show: whether to print out (default: True)

    Returns
    -------
    top_start_stations, top_end_stations

    '''
    station_ids = get_station_ids(df)

    top_start_stations = df['start station id'].value_counts()[:n]
    top_end_stations = df['end station id'].value_counts()[:n]

    if show:
        print('--- Top {} Starting Stations ---'.format(n))
        [print('{}. [{}] {}'.format(
         i, j, station_ids[j]))
         for i, j in enumerate(top_start_stations.index, 1)]

        print('\n--- Top {} Ending Stations ---'.format(n))
        [print('{}. [{}] {}'.format(
         i, j, station_ids[j]))
         for i, j in enumerate(top_end_stations.index, 1)]

    return top_start_stations, top_end_stations
```

[16]:
```
top_start_stations, top_end_stations = top_stations(df, 5, True)
```

```
--- Top 5 Starting Stations ---
1. [67] MIT at Mass Ave / Amherst St
2. [68] Central Square at Mass Ave / Essex St
3. [80] MIT Stata Center at Vassar St / Main St
4. [22] South Station - 700 Atlantic Ave
5. [107] Ames St at Main St

--- Top 5 Ending Stations ---
1. [67] MIT at Mass Ave / Amherst St
2. [68] Central Square at Mass Ave / Essex St
3. [107] Ames St at Main St
4. [190] Nashua Street at Red Auerbach Way
5. [80] MIT Stata Center at Vassar St / Main St
```

I am now just starting to draw a picture of where Blue Bikes are used most often. It looks like the MIT/Central Square area sees a lot of traffic. This is just a very preliminary look, so shortly I will drill down further to really get a better understanding of where these rides are being taken.

The next two functions are simple utility functions that translate numerical representations of days of the weeks or months of the year into actual days and months (e.g. day 0 is "Monday" and month 1 is "January"). This will be helpful when we visualize the data.

[17]:
```
def get_weekday(day_num):
    '''
    Parameters
    ----------
    day_num: int
        Returns string of day of week

    Returns
    -------
    Day of week (string)

    '''
    day = {0: 'Monday',
           1: 'Tuesday',
           2: 'Wednesday',
           3: 'Thursday',
           4: 'Friday',
           5: 'Saturday',
           6: 'Sunday'}

    return day[int(day_num)]
```

[18]:
```
def get_month(month_num):
    '''
```

```
    Parameters
    ----------
    month_num: int
        Returns string of month

    Returns
    -------
    Month (string)

    '''
    month = {1: 'January',
             2: 'February',
             3: 'March',
             4: 'April',
             5: 'May',
             6: 'June',
             7: 'July',
             8: 'August',
             9: 'September',
             10: 'October',
             11: 'November',
             12: 'December'}

    return month[int(month_num)]
```

The next 4 functions plot the most traveled months/days/hours, and the total number of rides taken based on gender. Together, these 4 plots are fed into the `matrix_plot` function to display as a 2x2 chart for easy comparison.

[19]:
```
def most_traveled_months(df, ax):
    '''
    Parameters
    ----------
    df: adjusted dataframe with start_month column added

    Returns
    -------
    bar chart of most traveled months
    '''
    months, rides = np.unique(df['start_month'], return_counts = True)

    colors = mpl.cm.summer(rides / max(rides))
    bar = ax.bar(x = [get_month(month) for month in months],
                 height = rides,
                 color = colors,
                 zorder = 2)
    ax.grid(axis = 'y', zorder = 1)
```

```
    ax.tick_params(axis = 'x', labelrotation = 70)
    ax.set_xlabel('Month')
    ax.set_ylabel('Total Rides')
    ax.set_title('Total Rides per Month')
    ax.set_facecolor('lightgray')
    return bar
```

[20]:
```
def most_traveled_days(df, ax):
    '''
    Parameters
    ----------
    df: adjusted dataframe with start_days column added

    Returns
    -------
    bar chart of most traveled days
    '''
    days, rides = np.unique(df['start_day'], return_counts = True)

    colors = mpl.cm.summer(rides / max(rides))
    bar = ax.bar(x = [get_weekday(day) for day in days],
                 height = rides,
                 color = colors,
                 zorder = 2)
    ax.grid(axis = 'y', zorder = 1)
    ax.tick_params(axis = 'x', labelrotation = 70)
    ax.set_xlabel('Day of Week')
    ax.set_ylabel('Total Rides')
    ax.set_title('Total Rides per Day')
    ax.set_facecolor('lightgray')
    return bar
```

[21]:
```
def most_traveled_times(df, ax):
    '''
    Parameters
    ----------
    df: adjusted dataframe with start_hour column added

    Returns
    -------
    bar chart of most traveled hours
    '''
    hours, rides = np.unique(df['start_hour'], return_counts = True)

    colors = mpl.cm.summer(rides / max(rides))
    bar = ax.bar(x = hours,
                 height = rides,
```

```
                 color = colors,
                 zorder = 2)
    ax.grid(axis = 'y', zorder = 1)
    ax.set_xlabel('Hour of the Day')
    ax.set_ylabel('Total Rides')
    ax.set_xticks(np.arange(0, 23, 3))
    ax.set_title('Total Rides per Hour')
    ax.set_facecolor('lightgray')
    return bar
```

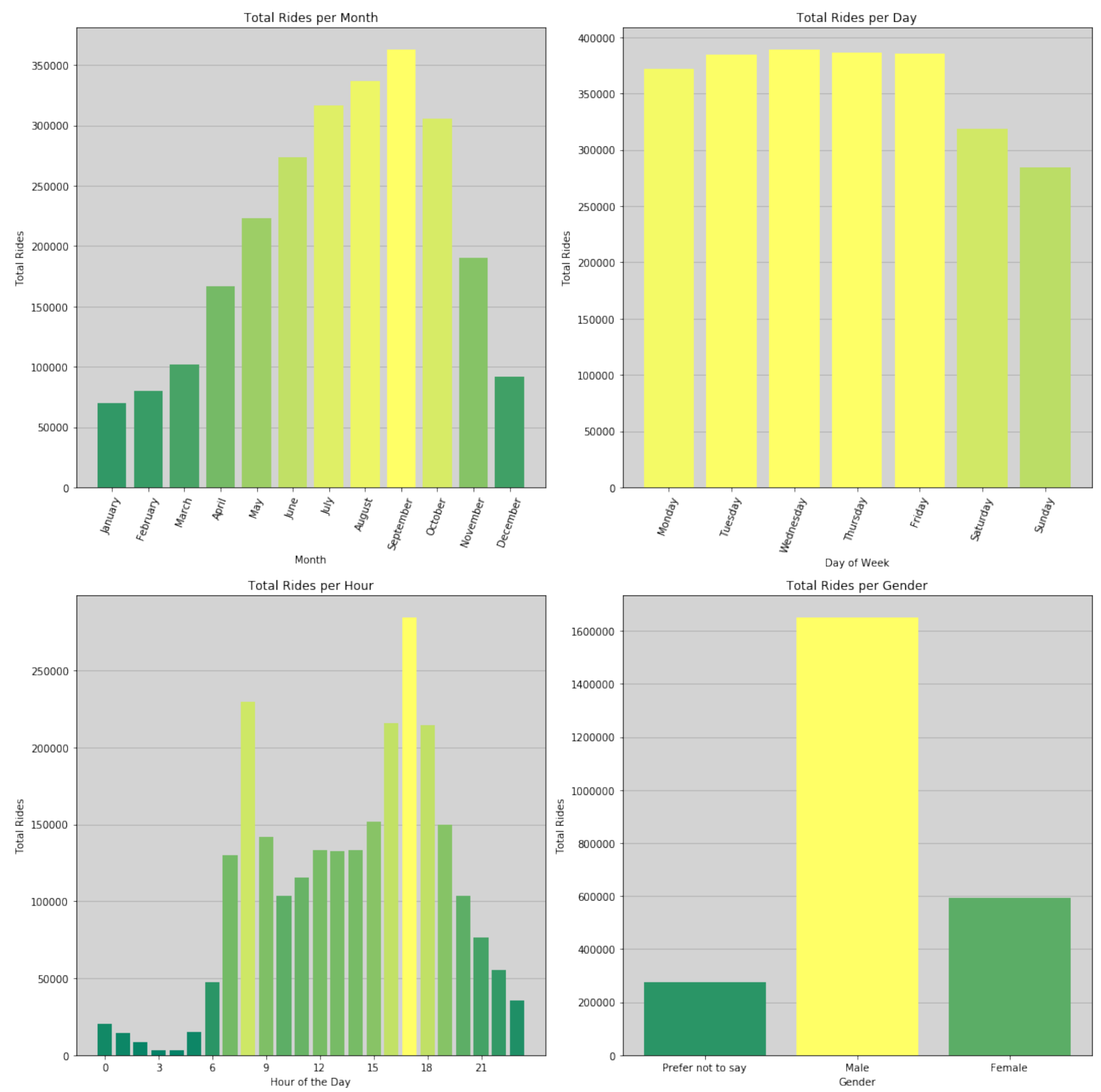
[22]:
```
def rides_by_gender(df, ax):
    '''
    Parameters
    ----------
    df: adjusted dataframe

    Returns
    -------
    bar chart of rides by gender
    '''
    gender, rides = np.unique(df['gender'], return_counts = True)

    colors = mpl.cm.summer(rides / max(rides))
    bar = ax.bar(x = gender,
                 height = rides,
                 color = colors,
                 zorder = 2)
    ax.grid(axis = 'y', zorder = 1)
    ax.tick_params(axis = 'x', labelrotation = 0)
    ax.set_xticks([0, 1, 2])
    ax.set_xticklabels(['Prefer not to say', 'Male', 'Female'])
    ax.set_xlabel('Gender')
    ax.set_ylabel('Total Rides')
    ax.set_title('Total Rides per Gender')
    ax.set_facecolor('lightgray')
    return bar
```

[23]:
```
def matrix_plot():
    '''
    Parameters
    ----------
    df: adjusted dataframe

    Returns
    -------
    2 x 2 subplots
    '''
```

```
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows = 2,
                                                 ncols = 2,
                                                 figsize = (15, 15))
    most_traveled_months(df, ax1)
    most_traveled_days(df, ax2)
    most_traveled_times(df, ax3)
    rides_by_gender(df, ax4)
    plt.tight_layout()
```

I now call the `matrix_plot` function to plot all of these charts. There are no big surprises related to the time when the bikes are most used: the 8AM and 5PM hours on weekdays during the warmer months. What does come as a major suprise, however, is how many more men are using the service than women. Men are riding at a rate of more than 2.5 times that of women! That discrepancy was certainly not something I expected to see when I started this investigation, and may warrant more investigation in a different analysis.

[24]:
```
matrix_plot()
```



The next `plot_legend` function is used internally in the `network_graph` and `rides_per_dock` functions to keep the plots similarly stylized.

[25]:
```
def plot_legend(df, sta_nums):
    '''
    Parameters
    ----------
    df: adjusted dataframe
    sta_nums: set of station numbers to be plotted

    Returns
    -------
    Nothing.
```

```
    Adds consistent stylized legend with station IDs/names to graphs.

    '''
    station_ids = get_station_ids(df)
    handles = [mpl.patches.Patch(facecolor = 'k',
                                 edgecolor = 'k',
                                 label = '{}: {}'.format(num,
                                                         station_ids[num]))
               for num in sorted(list(sta_nums))]
    leg = plt.legend(handles = handles,
                     loc = 'upper center',
                     bbox_to_anchor = (0.5, -0.05),
                     shadow = True,
                     ncol = 2,
                     handlelength = 0,
                     handletextpad = 0,
                     fancybox = True)
    for item in leg.legendHandles:
        item.set_visible(False)
    leg.get_frame().set_facecolor('lightblue')
```

I now draw a directed network graph of the top `n` routes, based on rides between each station. Station IDs are nodes in the graph and are plotted in a circle, while colored arrows represent the edges and point from the starting station towards the ending station.

It is important to note that the color of the arrow represents how popular the specific route is, but is based on how many routes are plotted. For example, below I plot the top 20 routes. The most popular route is at the top of the color bar and will always be red, but if I were to replot with a different number of routes (e.g. top 50 routes instead) the colors would shift accordingly to accomodate more or less routes.

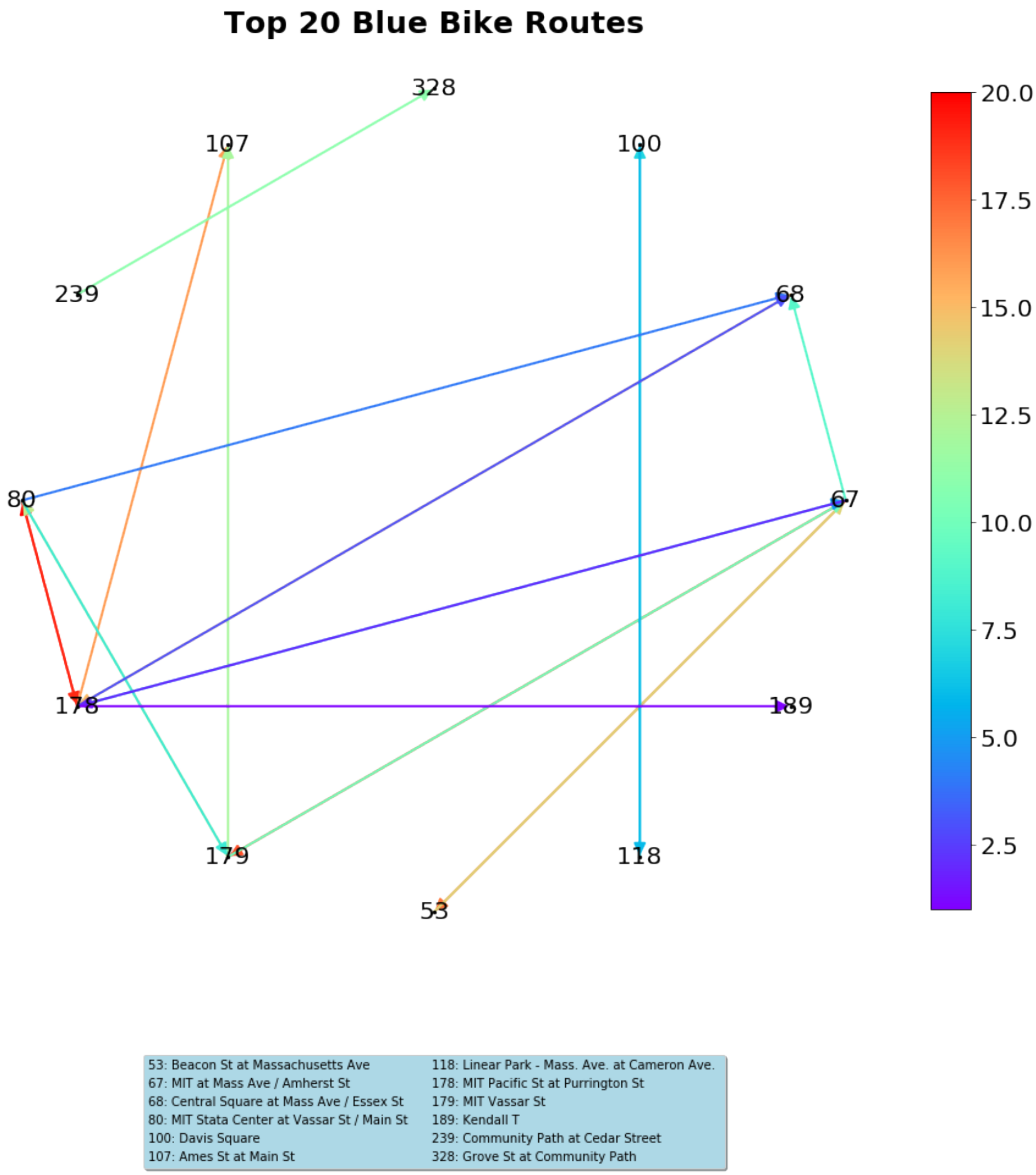
This graph can quickly become very busy, and is somewhat difficult to interpret when busy routes travel in both directions (for instance route 178 to 80 vs. 80 to 178) since it is directed and the arrows will overlap. Therefore, while this is a great way to get a quick visual of the busiest routes and stations, I will explicitly print out the busiest routes further below.

[26]:
```
def network_graph(df, n):
    '''
    Parameters
    ----------
    df: adjusted dataframe
    n: (int) number of rides to plot

    Returns
    -------
    network graph of top n routes
    '''
    grouped = df.groupby(['start station id', 'end station id']).count()\
```

```
                 ['tripduration']
grouped = grouped.nlargest(n)

station_numbers = set()
edge_collection = []
for i in grouped.index:
    edge_collection.append(i)
    station_numbers.update(i)
edge_colors = range(len(edge_collection), 0, -1)

G = nx.DiGraph()
G.add_nodes_from(station_numbers)

pos = nx.layout.circular_layout(G)

plt.figure(figsize = (15, 15))
nodes = nx.draw_networkx_nodes(G,
                               pos,
                               node_size = 5,
                               node_color = 'black')
edges = nx.draw_networkx_edges(G,
                               pos,
                               node_size = 5,
                               arrowstyle = '-|>',
                               arrowsize = 20,
                               edgelist = edge_collection,
                               edge_color = edge_colors,
                               edge_cmap = plt.cm.rainbow,
                               width = 2)

pc = mpl.collections.PatchCollection(edges,
                                     cmap = plt.cm.rainbow)
pc.set_array(edge_colors)
cbar = plt.colorbar(pc,
                    fraction = 0.04,
                    pad = 0.01)
cbar.ax.tick_params(labelsize = 20)
nx.draw_networkx_labels(G,
                        pos,
                        font_size = 20)

ax = plt.gca()
ax.set_axis_off()
plot_legend(df, station_numbers)
plt.axis('equal')
plt.title('Top {} Blue Bike Routes'.format(n),
          fontdict = {'fontsize': 25,
```

```
                                'fontweight': 'bold'},
                  pad = -25)
    plt.show()
```

[27]:
```
network_graph(df, 20)
```



I now specifically print out the top `n` routes. Whereas the network graph gives a good visual representation of which stations and routes see a lot of traffic, sometimes it helps just to explicitly print the text out.

It is very interesting to compare the busiest routes (which obviously include two separate stations) with the busiest individual starting and ending stations. For instance, if you look back at the top ending stations, station 190 (Nashua Street at Red Auerbach Way) was the 4th busiest ending station but does not even show up as part of the top 20 routes. That suggests that while station 190 is a highly popular ending station, people are getting there from any number of stations all over the city.

On the other hand, the route between stations 80 and 178 is the most popular route in both directions, but station 178 doesn't even register in the top 5 starting or ending stations. This suggests that while this specific route is extremely popular, station 178 doesn't actually see as much traffic in or out as other stations.

[28]:
```python
def top_trips(df, n):
    '''
    Parameters
    ----------
    df: adjusted dataframe
    n (int): number of top trips to print

    Returns
    -------
    Prints top n trips

    '''
    station_ids = get_station_ids(df)
    grouped = df.groupby(['start station id', 'end station id']).count()\
                 ['tripduration']
    grouped = grouped.nlargest(n)

    edges = []
    for i in grouped.index:
        edges.append(i)
    for num, (i, j) in enumerate(edges, 1):
        print('{}: [{}] {} to [{}] {}'.format(num,
                                             i, station_ids[i],
                                             j, station_ids[j]))
```

[29]:
```python
top_trips(df, 20)
```

```
1: [178] MIT Pacific St at Purrington St to [80] MIT Stata Center at Vassar St /
Main St
2: [80] MIT Stata Center at Vassar St / Main St to [178] MIT Pacific St at
Purrington St
3: [67] MIT at Mass Ave / Amherst St to [179] MIT Vassar St
4: [67] MIT at Mass Ave / Amherst St to [53] Beacon St at Massachusetts Ave
5: [178] MIT Pacific St at Purrington St to [107] Ames St at Main St
6: [68] Central Square at Mass Ave / Essex St to [178] MIT Pacific St at
Purrington St
7: [53] Beacon St at Massachusetts Ave to [67] MIT at Mass Ave / Amherst St
```

```
8: [179] MIT Vassar St to [80] MIT Stata Center at Vassar St / Main St
9: [179] MIT Vassar St to [107] Ames St at Main St
10: [239] Community Path at Cedar Street to [328] Grove St at Community Path
11: [179] MIT Vassar St to [67] MIT at Mass Ave / Amherst St
12: [67] MIT at Mass Ave / Amherst St to [68] Central Square at Mass Ave / Essex
St
13: [80] MIT Stata Center at Vassar St / Main St to [179] MIT Vassar St
14: [118] Linear Park - Mass. Ave. at Cameron Ave.  to [100] Davis Square
15: [100] Davis Square to [118] Linear Park - Mass. Ave. at Cameron Ave.
16: [178] MIT Pacific St at Purrington St to [67] MIT at Mass Ave / Amherst St
17: [80] MIT Stata Center at Vassar St / Main St to [68] Central Square at Mass
Ave / Essex St
18: [178] MIT Pacific St at Purrington St to [68] Central Square at Mass Ave /
Essex St
19: [67] MIT at Mass Ave / Amherst St to [178] MIT Pacific St at Purrington St
20: [178] MIT Pacific St at Purrington St to [189] Kendall T
```

Finally, I look at how many rides are taken per dock, starting with the highest number of rides per dock. Recall that one dock can house one bike, and the number of docks per station can vary. By understanding where the busiest routes and stations are, as well as knowing how many rides (starting or ending) are taken per individual bike dock, I can get a very good understanding of how to optimize the layout of bike docks at stations around the city.

Below I read in the data set which includes how many docks are at each station. We can take a look at the first couple of rows in the dataset:

[30]:
```
station_df = pd.read_csv('Data/current_bluebikes_stations.csv', skiprows = 1)
station_df.head()
```

[30]:
```
   Number                                     Name   Latitude  Longitude  \
0  A32019                         175 N Harvard St  42.363796 -71.129164
1  S32035                            191 Beacon St  42.380323 -71.108786
2  S32023                              30 Dane St  42.381001 -71.104025
3  M32026  359 Broadway - Broadway at Fayette Street  42.370803 -71.104412
4  M32054                           699 Mt Auburn St  42.375002 -71.148716

     District Public  Total docks
0      Boston    Yes           18
1  Somerville    Yes           19
2  Somerville    Yes           15
3   Cambridge    Yes           23
4   Cambridge    Yes           25
```

I now plot the number of rides per dock.

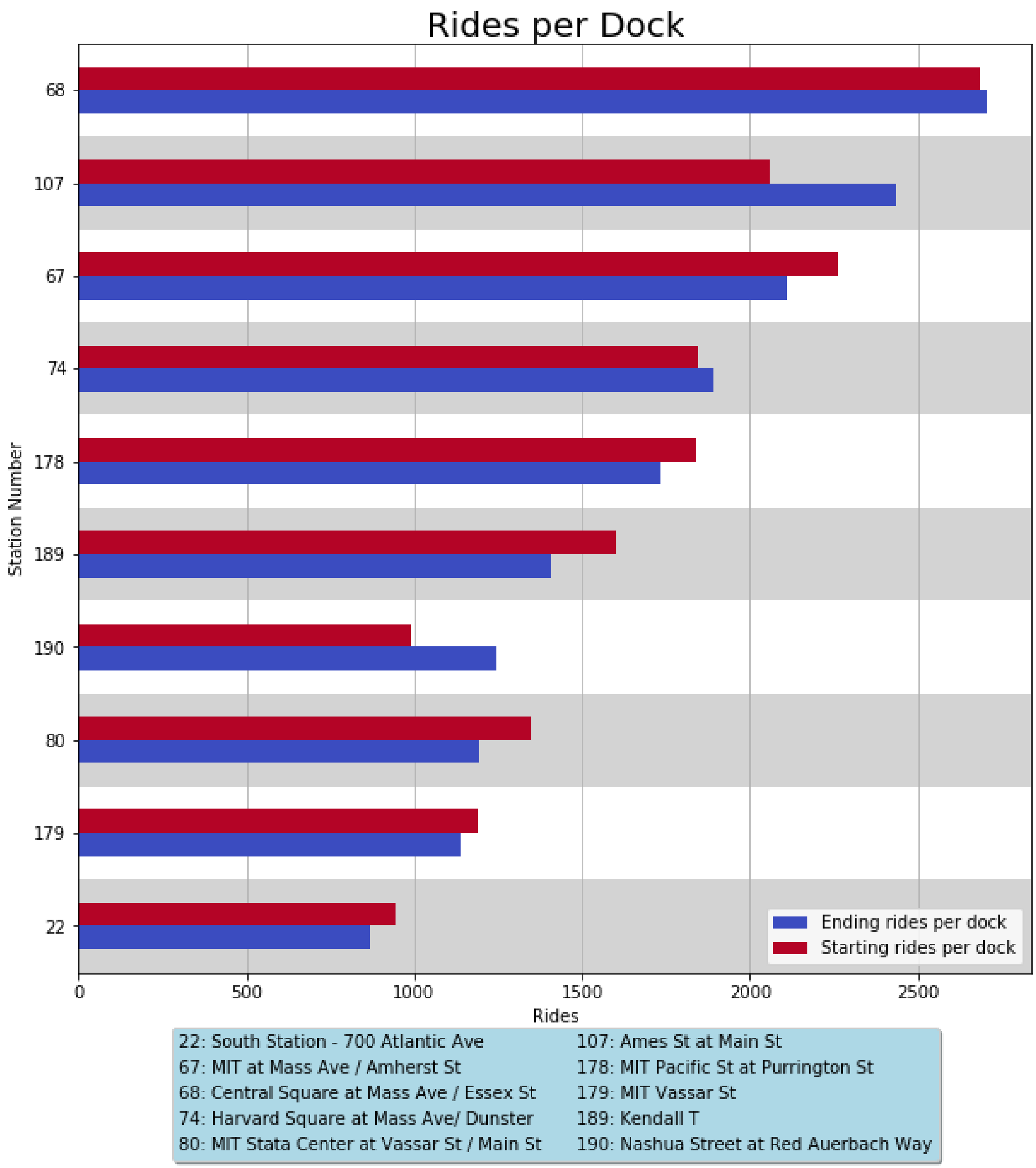
From this plot we can see station 74 (Harvard Square at Mass Ave / Dunster) has signficantly more rides per dock than station 80 (MIT Stata Center at Vassar St / Main St), but station 74 doesn't even register in the top 20 routes or top 5 starting/ending stations. On the other hand, station 68 (Central Square at Mass Ave / Essex St), 107 (Ames St at Main St) and 67 (MIT at Mass Ave /

Amherst St) see the most rides per dock, are among the top 5 starting/ending stations, and are included in many of the most popular routes. I would therefore recommend focusing on adding more docks to these high-traffic stations before focusing on others.

[31]:
```python
def rides_per_dock(df, sta_df, n):
    '''
    Parameters
    ----------
    df: adjusted dataframe
    station_df: dataframe with station names and total docks
    n: (int) number of stations to look at
    start: (bool) whether to look at starting (default) or ending stations

    Returns
    -------
    bar chart of rides per station for top n -ending- stations
    '''
    station_ids = get_station_ids(df)

    # Concatenate top starting/ending stations on station ID
    top_df = pd.concat(top_stations(df, -1, False), axis = 1).reset_index()
    top_df = top_df.nlargest(n, 'end station id')
    top_df.columns = ['ID', 'Start rides', 'End rides']

    # Station_df uses names instead of numbers, so need to join on names
    top_df['Name'] = top_df['ID'].apply(lambda x: station_ids[x])
    top_df = top_df.join(sta_df.set_index('Name'), on = 'Name')

    # Some stations have 0 docks. Delete, but worth investigating separately
    top_df = top_df[top_df['Total docks'] != 0]

    # Calculate rides per dock for starting/ending stations
    top_df['Starting rides per dock'] = \
        top_df.apply(lambda x: x['Start rides']/x['Total docks'], axis = 1)
    top_df['Ending rides per dock'] = \
        top_df.apply(lambda x: x['End rides']/x['Total docks'], axis = 1)
    top_df = top_df.sort_values(by = 'Ending rides per dock')

    # Pull out useful columns
    top_df = top_df[['ID', 'Ending rides per dock',
                     'Starting rides per dock']].set_index('ID')

    # Plot
    top_df.plot(figsize = (10, 10),
                kind = 'barh',
                colormap = 'coolwarm',
                zorder = 2)
```

```
    plt.title('Rides per Dock',
              fontdict = {'fontsize': 20},
              pad = -25)
    plt.xlabel('Rides')
    plt.ylabel('Station Number')
    plt.grid(which = 'major', axis = 'x')
    #plt.tick_params(axis = 'both', labelsize = '15')

    # Shade every other station
    for i, j in enumerate(top_df.index):
        if i % 2 == 0:
            plt.axhspan(i - 0.5,
                        i + 0.5,
                        facecolor = 'lightgray',
                        zorder = 1)

    # Plot 2 legends
    ax = plt.gca()
    ax.add_artist(plt.legend(loc = 'lower right'))
    station_numbers = set()
    for i in top_df.index:
        station_numbers.add(i)
    plot_legend(df, station_numbers)
    plt.show()
```

[32]:
```
rides_per_dock(df, station_df, 10)
```



Based on this analysis, I can get a good idea of where to expand the Blue Bikes stations by adding more bike docks. In order for the service to be successful, subscribers must feel confident that they can rent a bike where and when they want, and easily drop it off where and when they are done.

There is one critical piece which I did not take into account in this analysis, and that is the actual logistics of adding stations. I walk by station 68 (Central Square at Mass Ave / Essex St) every day on my way to work, and it is very often entirely without bikes, but it is right in the middle of Central Square. Blue Bikes cannot just slap extra docks onto any station they want as the docks take up physical space on the sidewalk and space in the city is at an absolute premium. Therefore, this analysis should serve as guidance about stations to look into, but physical space constraints must also be considered before expanding.