DV_Project_UberDataAnalysis

February 15, 2024

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.dates as mdates
import plotly.express as px
```

[2]:
```
# Load the Drive helper and mount
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[3]:
```
# Import csv files:
data_apr= pd.read_csv('/content/drive/MyDrive/MS/Data Visualization/Project/
 ↪Uber_Data/uber-raw-data-apr14.csv')
data_may= pd.read_csv('/content/drive/MyDrive/MS/Data Visualization/Project/
 ↪Uber_Data/uber-raw-data-may14.csv')
data_jun= pd.read_csv('/content/drive/MyDrive/MS/Data Visualization/Project/
 ↪Uber_Data/uber-raw-data-jun14.csv')
data_jul= pd.read_csv('/content/drive/MyDrive/MS/Data Visualization/Project/
 ↪Uber_Data/uber-raw-data-jul14.csv')
data_aug= pd.read_csv('/content/drive/MyDrive/MS/Data Visualization/Project/
 ↪Uber_Data/uber-raw-data-aug14.csv')
data_sep= pd.read_csv('/content/drive/MyDrive/MS/Data Visualization/Project/
 ↪Uber_Data/uber-raw-data-sep14.csv')
uber_data= pd.concat([data_apr,data_may,data_jun, data_jul,data_aug,data_sep])
```

[5]:
```
uber_data.count()
```

[5]:
```
Date/Time    4534327
Lat          4534327
Lon          4534327
Base         4534327
dtype: int64
```

[6]:
```
uber_data.head(10)
```

[6]:

	Date/Time	Lat	Lon	Base
0	4/1/2014 0:11:00	40.7690	-73.9549	B02512
1	4/1/2014 0:17:00	40.7267	-74.0345	B02512
2	4/1/2014 0:21:00	40.7316	-73.9873	B02512
3	4/1/2014 0:28:00	40.7588	-73.9776	B02512
4	4/1/2014 0:33:00	40.7594	-73.9722	B02512
5	4/1/2014 0:33:00	40.7383	-74.0403	B02512
6	4/1/2014 0:39:00	40.7223	-73.9887	B02512
7	4/1/2014 0:45:00	40.7620	-73.9790	B02512
8	4/1/2014 0:55:00	40.7524	-73.9960	B02512
9	4/1/2014 1:01:00	40.7575	-73.9846	B02512

0.1 Examining data for Sept 2014

[ ]:
```
# Convert Date/Time column to datetime type:
data_sep['Date/Time'] = pd.to_datetime(data_sep['Date/Time'])

# Add two new columns of Date and Hours:
data_sep['Date'] = data_sep['Date/Time'].dt.date
data_sep['Month']=data_sep['Date/Time'].dt.month
data_sep['Day']=data_sep['Date/Time'].dt.day
data_sep['Hour'] = data_sep['Date/Time'].dt.hour
data_sep['Minute']=data_sep['Date/Time'].dt.minute
data_sep['Day_of_Week']= data_sep['Date/Time'].dt.strftime('%A')

# Calculate counts of occurrences of Dates and Hours:
date_counts = data_sep['Date'].value_counts().sort_index()
hour_counts = data_sep['Hour'].value_counts().sort_index()
day_counts = data_sep['Day_of_Week'].value_counts()
```

[ ]:
```
data_sep.head()
```

[ ]:

	Date/Time	Lat	Lon	Base	Date	Month	Day	Hour	Minute	Day_of_Week
0	2014-09-01 00:01:00	40.2201	-74.0021	B02512	2014-09-01	9	1	0	1	Monday
1	2014-09-01 00:01:00	40.7500	-74.0027	B02512	2014-09-01	9	1	0	1	Monday
2	2014-09-01 00:03:00	40.7559	-73.9864	B02512	2014-09-01	9	1	0	3	Monday
3	2014-09-01 00:06:00	40.7450	-73.9889	B02512	2014-09-01	9	1	0	6	Monday
4	2014-09-01 00:11:00	40.8145	-73.9444	B02512	2014-09-01	9	1	0	11	Monday

[ ]:
```
date_counts.head()
```

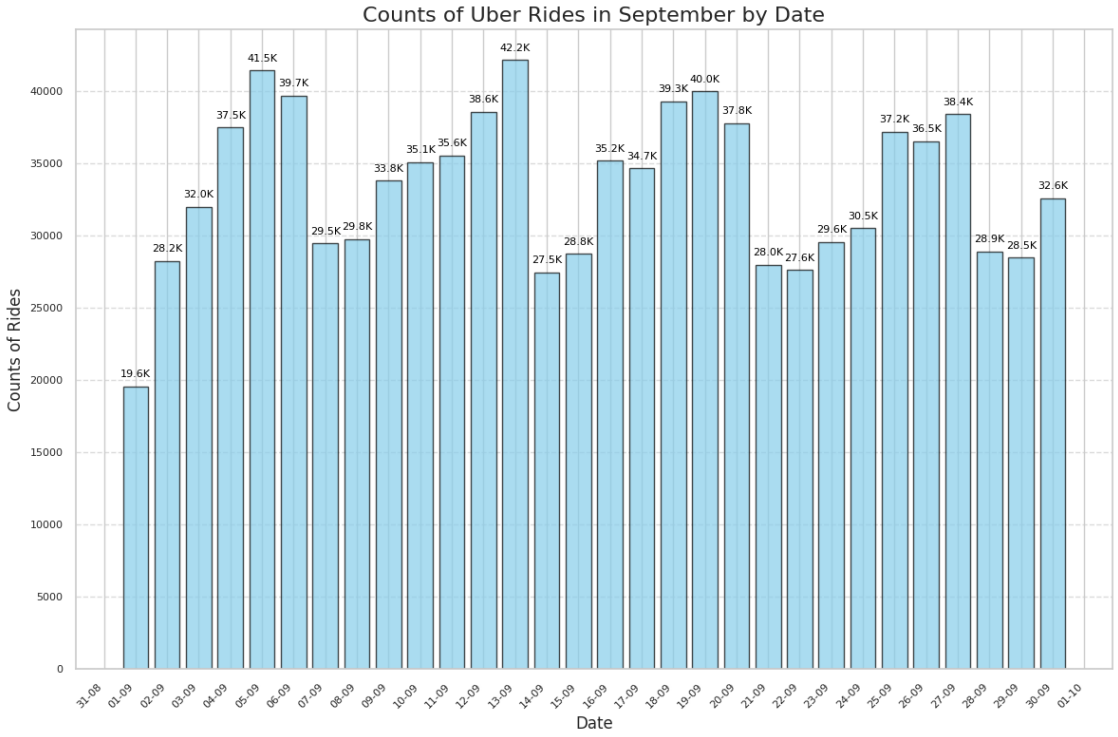
[ ]:
```
2014-09-01    19581
2014-09-02    28239
2014-09-03    32007
2014-09-04    37507
2014-09-05    41457
Name: Date, dtype: int64
```

[ ]:
```
duplicates = data_sep.duplicated()
print(data_sep[duplicates].head())
```

```
Empty DataFrame
Columns: [Date/Time, Lat, Lon, Base, Date, Month, Day, Hour, Minute,
Day_of_Week]
Index: []
```

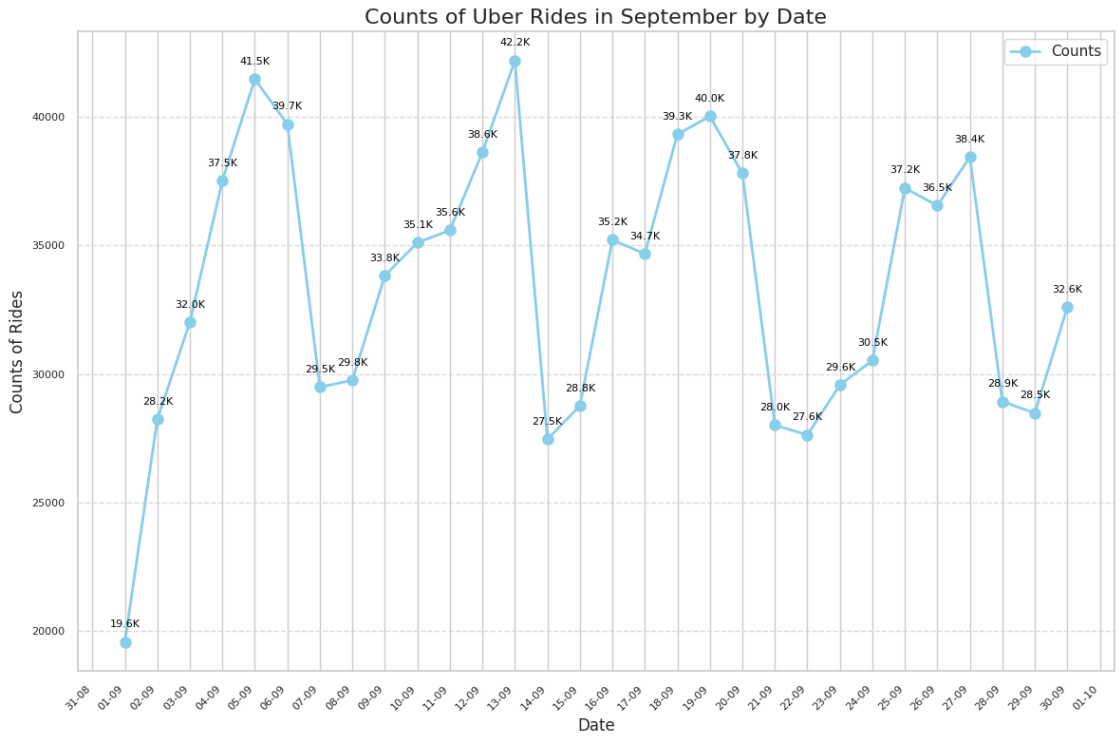
[ ]:
```
data_sep.drop_duplicates(inplace=True)
```

[ ]:
```
# Create a bar plot of Dates:
plt.figure(figsize=(12, 8))
plt.bar(date_counts.index, date_counts.values, color='skyblue',␣
 ↪edgecolor='black', alpha=0.7)

plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d-%m'))

plt.xlabel('Date')
plt.ylabel('Counts of Rides')

for i, value in enumerate(date_counts.values):
    plt.text(date_counts.index[i], value + 500, f"{value/1000:.1f}K",␣
 ↪ha='center', va='bottom', fontsize=8, color='black')

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.title('Counts of Uber Rides in September by Date', fontsize=16)
plt.xticks(rotation=45, ha='right', fontsize=8)
plt.yticks(fontsize=8)

plt.tight_layout()

plt.show()
```



[ ]:
```
# Create a line plot of Dates:
plt.figure(figsize=(12, 8))
plt.plot(date_counts.index, date_counts.values, '-o', color='skyblue',␣
 ↪linewidth=2, markersize=8, label='Counts')

plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d-%m'))

plt.xlabel('Date')
plt.ylabel('Counts of Rides')

for i, value in enumerate(date_counts.values):
    plt.text(date_counts.index[i], value + 500, f"{value/1000:.1f}K",␣
 ↪ha='center', va='bottom', fontsize=8, color='black')

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.title('Counts of Uber Rides in September by Date', fontsize=16)
plt.xticks(rotation=45, ha='right', fontsize=8)
plt.yticks(fontsize=8)

plt.legend()
```

```
plt.tight_layout()

plt.show()
```



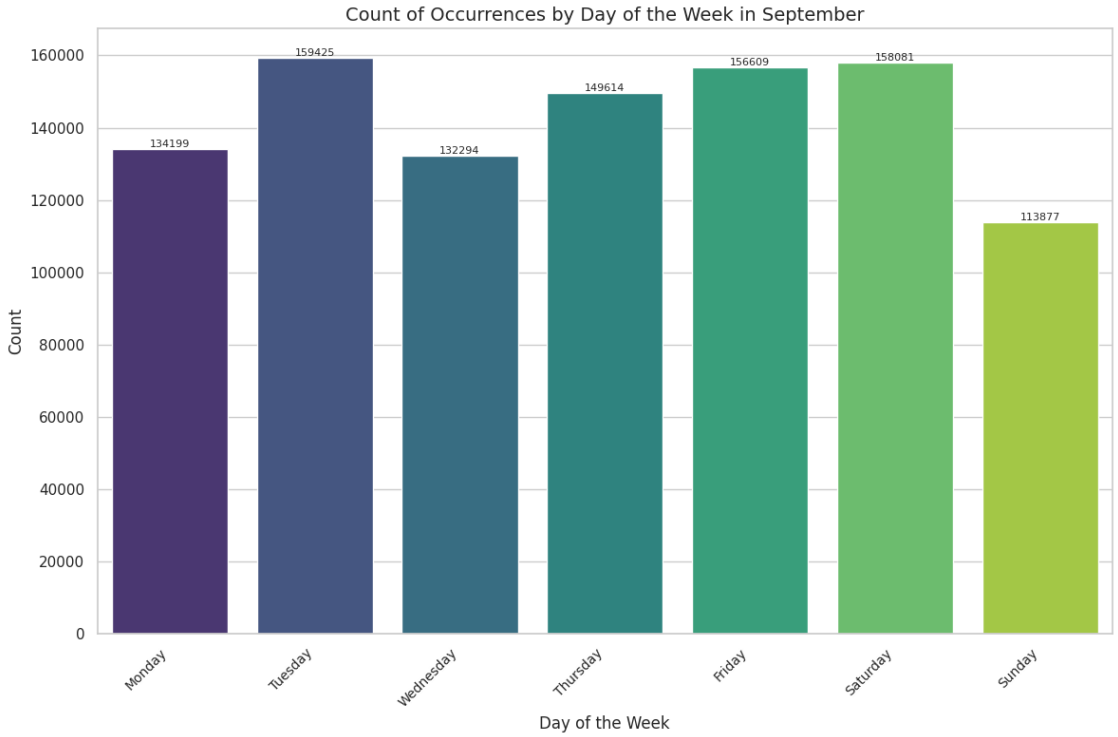
[ ]:
```
import plotly.graph_objects as go
import pandas as pd


formatted_dates = date_counts.index.astype(str)

fig = go.Figure()

fig.add_trace(go.Scatter(x=formatted_dates, y=date_counts.values,␣
 ↪mode='lines+markers',
                         name='Counts of Rides'))

fig.update_xaxes(
    tickmode='array',
    tickvals=formatted_dates,
    ticktext=formatted_dates,
    tickangle=45,
    title='Date'
)
```

```
for i, value in enumerate(date_counts.values):
    fig.add_annotation(x=formatted_dates[i], y=value + 500,
                       text=f"{value/1000:.1f}K",
                       showarrow=True,
                       arrowhead=3,
                       ax=0,
                       ay=-30,
                       font=dict(size=10, color='black'))

fig.update_layout(title='Counts of Uber Rides in September by Date',␣
 ↪xaxis_title='Date', yaxis_title='Counts of Rides')

fig.show()
```

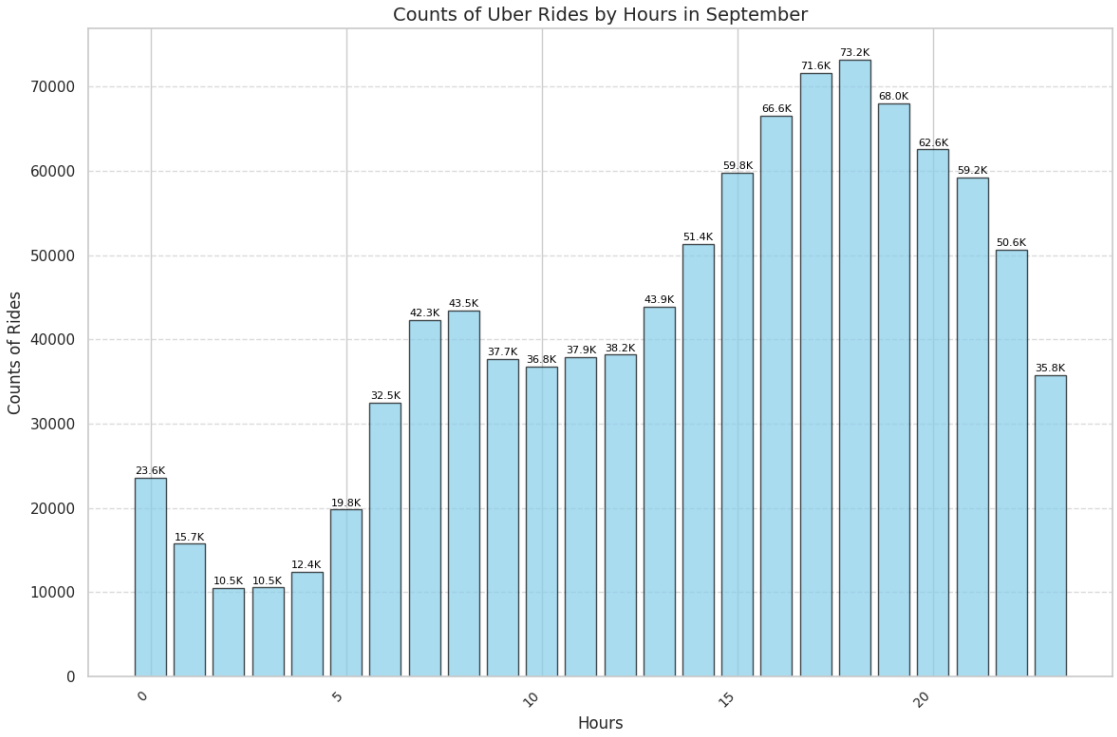
[ ]:
```
import plotly.graph_objects as go
import plotly.express as px
from datetime import datetime


fig = px.line(x=formatted_dates, y=date_counts.values, labels={'x': 'Date', 'y':
 ↪ 'Counts of Rides'},
              title='Counts of Uber Rides in September by Date',␣
 ↪line_shape='linear')

fig.update_xaxes(
    tickmode='array',
    tickvals=formatted_dates,
    ticktext=formatted_dates,
    tickangle=45,
    title='Date'
)

# Add annotation on top of each data point with Day of the Week
for i, (date, value) in enumerate(zip(date_counts.index, date_counts.values)):
    day_of_week = date.strftime('%A')  # Get the day of the week
    fig.add_annotation(x=date, y=value + 500,
                       text=f"{day_of_week}<br>{value/1000:.1f}K",
                       showarrow=True,
                       arrowhead=3,
                       ax=0,
                       ay=-30,
                       font=dict(size=10, color='black'))


fig.update_layout(title='Counts of Uber Rides in September by Date',␣
 ↪xaxis_title='Date', yaxis_title='Counts of Rides')
```

```
fig.show()
```

[ ]:
```
# Sort the counts of Days from Monday to Sunday:
days_in_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
 ↪'Saturday', 'Sunday']
day_counts = day_counts.reindex(days_in_order)

# Set a Seaborn style for a more visually appealing plot
sns.set(style="whitegrid")


plt.figure(figsize=(12, 8))
sns.barplot(x=day_counts.index, y=day_counts.values, palette="viridis")


plt.xlabel('Day of the Week', fontsize=12)
plt.xticks(rotation=45, ha='right', fontsize=10)

plt.ylabel('Count', fontsize=12)


for i, value in enumerate(day_counts.values):
    plt.text(i, value + 5, str(value), ha='center', va='bottom', fontsize=8)

plt.title('Count of Occurrences by Day of the Week in September', fontsize=14)

plt.tight_layout()

plt.show()
```



[ ]:
```
sns.set(style="whitegrid")

plt.figure(figsize=(12, 8))
plt.bar(hour_counts.index, hour_counts.values, color='skyblue',␣
 ↪edgecolor='black', alpha=0.7)

plt.xlabel('Hours', fontsize=12)
plt.xticks(rotation=45, ha='right', fontsize=10)

plt.ylabel('Counts of Rides', fontsize=12)

for i, value in enumerate(hour_counts.values):
    plt.text(hour_counts.index[i], value + 200, f"{value/1000:.1f}K",␣
 ↪ha='center', va='bottom', fontsize=8, color='black')

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.title('Counts of Uber Rides by Hours in September', fontsize=14)

plt.tight_layout()

plt.show()
```



[ ]:
```
fig = px.bar(x=hour_counts.index, y=hour_counts.values,
             labels={'x': 'Hours', 'y': 'Counts of Rides'},
             title='Counts of Uber Rides by Hours in September',
             color_discrete_sequence=['skyblue'],
             text=[f"{val/1000:.1f}K" for val in hour_counts.values])


fig.update_xaxes(tickangle=45, tickmode='array', tickvals=hour_counts.index,␣
 ↪ticktext=hour_counts.index)


fig.update_layout(xaxis_title='Hours', yaxis_title='Counts of Rides',␣
 ↪title_font_size=14)


fig.show()
```

Observation 1 – from April 2014 Data Based on the plot of count of Uber rides by day of the week in April 2014 in NYC,

it appears that Monday and Sunday have the least counts, while Tuesday and Wednesday have the most rides.

Additionally, the plot indicates that approximately 60% of rides occurred between 14:00 - 21:00.

0.2 Examing All Data from April to September

[ ]:
```
uber_data['Date/Time'] = pd.to_datetime(uber_data['Date/Time'])

uber_data['Date'] = uber_data['Date/Time'].dt.date
uber_data['Month']=uber_data['Date/Time'].dt.month
uber_data['Day']=uber_data['Date/Time'].dt.day
uber_data['Hour'] = uber_data['Date/Time'].dt.hour
uber_data['Minute']=uber_data['Date/Time'].dt.minute
uber_data['Day_of_Week']= uber_data['Date/Time'].dt.strftime('a%A')
uber_data['Day_of_Week']= uber_data['Day_of_Week'].str[1:]

date_counts_concat = uber_data['Date'].value_counts().sort_index()
hour_counts_concat = uber_data['Hour'].value_counts().sort_index()

day_counts_concat = uber_data['Day_of_Week'].value_counts()
```

[ ]:
```
duplicates = data_sep.duplicated()

print(data_sep[duplicates].head())
```

```
             Date/Time      Lat      Lon    Base
76    9/1/2014 8:59:00  40.6950 -74.1780  B02512
153   9/1/2014 11:24:00  40.7249 -74.0354  B02512
205   9/1/2014 12:30:00  40.7214 -74.0409  B02512
435   9/1/2014 17:47:00  40.6875 -74.1905  B02512
500   9/1/2014 19:35:00  40.6947 -74.1777  B02512
```

[ ]:
```
data_sep.drop_duplicates(inplace=True)
```

[ ]:
```
uber_data.head()
```

[ ]:
```
             Date/Time      Lat      Lon    Base        Date  Month  Day  Hour  \
0 2014-04-01 00:11:00  40.7690 -73.9549  B02512  2014-04-01      4    1     0
1 2014-04-01 00:17:00  40.7267 -74.0345  B02512  2014-04-01      4    1     0
2 2014-04-01 00:21:00  40.7316 -73.9873  B02512  2014-04-01      4    1     0
3 2014-04-01 00:28:00  40.7588 -73.9776  B02512  2014-04-01      4    1     0
4 2014-04-01 00:33:00  40.7594 -73.9722  B02512  2014-04-01      4    1     0

   Minute Day_of_Week
0      11     Tuesday
1      17     Tuesday
2      21     Tuesday
3      28     Tuesday
4      33     Tuesday
```

[ ]:
```
unique_bases = uber_data['Base'].unique()
print(unique_bases)
```

```
['B02512' 'B02598' 'B02617' 'B02682' 'B02764']
```

[ ]:
```
base_map= {
    'B02512': 'Unter',
    'B02598': 'Hinter',
    'B02617': 'Weiter',
    'B02682': 'Schmecken',
    'B02764': 'Danach-NY'}

uber_data['Base']= uber_data['Base'].map(base_map)
```

[ ]:
```
uber_data.head()
```

[ ]:
```
             Date/Time      Lat      Lon   Base        Date  Month  Day  Hour  \
0 2014-04-01 00:11:00  40.7690 -73.9549  Unter  2014-04-01      4    1     0
1 2014-04-01 00:17:00  40.7267 -74.0345  Unter  2014-04-01      4    1     0
2 2014-04-01 00:21:00  40.7316 -73.9873  Unter  2014-04-01      4    1     0
3 2014-04-01 00:28:00  40.7588 -73.9776  Unter  2014-04-01      4    1     0
4 2014-04-01 00:33:00  40.7594 -73.9722  Unter  2014-04-01      4    1     0

   Minute Day_of_Week
0      11     Tuesday
1      17     Tuesday
2      21     Tuesday
3      28     Tuesday
4      33     Tuesday
```

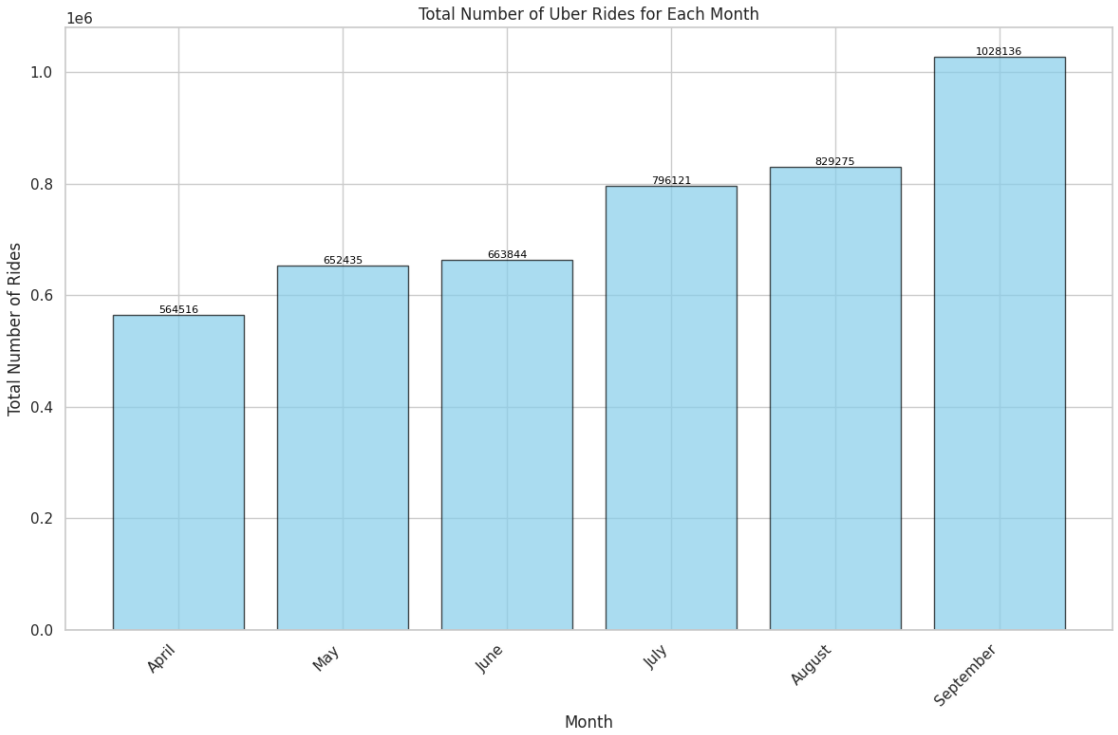
###01. Which month and day of the month sees the highest number of uber trips?

[ ]:
```
df = pd.DataFrame(uber_data)

weekday = pd.DataFrame(df[['Day', 'Month']].value_counts()).reset_index()
weekday.columns = ['Day', 'Month', 'Count']
weekday['Day'] = pd.Categorical(weekday['Day'], categories=range(1, 32),␣
 ↪ordered=True)
weekday['Month'] = pd.Categorical(weekday['Month'], categories=[4, 5, 6, 7, 8,␣
 ↪9], ordered=True)

fig = px.bar(weekday,
             x='Day',
             y='Count',
             color='Month',
             template='seaborn',
             labels={'Count': 'Number of Trips', 'Day': 'Day of the Month'},
```

```
            width=1500,
            height=600,
            category_orders={"Month": [4, 5, 6, 7, 8, 9]},
            color_discrete_sequence=['#2C2C3E', '#2E5467', '#1E7F84',␣
 ↪'#33AC8D', '#78D584', '#D1FA74'],
            text='Count')

fig.update_traces(texttemplate='%{text:.2s}', textposition='outside')
fig.update_layout(uniformtext_minsize=8, uniformtext_mode='hide')
fig.update_layout(title_text='Uber trip by Months and Days', title_x=0.5)

fig.show()
```

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import calendar


uber_data['Date'] = pd.to_datetime(uber_data['Date'])

uber_data['Month'] = uber_data['Date'].dt.month
uber_data['Year'] = uber_data['Date'].dt.year

monthly_counts = uber_data.groupby(['Year', 'Month']).size().
 ↪reset_index(name='Number of Rides')

monthly_counts['Month-Year'] = monthly_counts['Year'].astype(str) + '-' +␣
 ↪monthly_counts['Month'].astype(str)

monthly_counts = monthly_counts.sort_values(['Year', 'Month'])

month_names = [calendar.month_name[i] for i in monthly_counts['Month']]

plt.figure(figsize=(12, 8))
bars = plt.bar(month_names, monthly_counts['Number of Rides'], color='skyblue',␣
 ↪edgecolor='black', alpha=0.7)

plt.xlabel('Month')
plt.ylabel('Total Number of Rides')
plt.title('Total Number of Uber Rides for Each Month')
plt.xticks(rotation=45, ha='right')

for bar in bars:
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height(), str(int(bar.
 ↪get_height())),
             ha='center', va='bottom', fontsize=8, color='black')
```

```
plt.tight_layout()
plt.show()
```



[ ]:
```
date_counts_concat.index = date_counts_concat.index.astype(str)

fig = px.line(x=date_counts_concat.index, y=date_counts_concat.values,
              labels={'x': 'Date', 'y': 'Counts of Rides'},
              title='Counts of Uber Rides from April to September by Date')

fig.update_traces(marker=dict(size=10, color='skyblue'), line_shape='linear',␣
 ↪line=dict(color='darkblue'))

fig.update_traces(hovertemplate='<b>Date</b>: %{x}<br><b>Count</b>: %{y}')

fig.update_xaxes(type='category')

fig.update_layout(
    title=dict(text='Counts of Uber Rides from April to September by Date', x=0.
 ↪5, y=0.98),
    xaxis_title=dict(standoff=10),
    yaxis_title=dict(standoff=10),
    font=dict(family='Arial', size=12, color='black'),
    paper_bgcolor='lightgray',
```

```
)

fig.show()
```

13th September 2014 sees the highest number of trips!

###02. Which is the busiest hour in the day for uber cabs?

[ ]:
```
hour=pd.DataFrame(uber_data['Hour'].value_counts()).reset_index()
hour.columns=['Hour','Count']
hour=hour.sort_values(by='Hour')
```

[ ]:
```
fig2=px.bar(hour,
      x='Hour',
      y='Count',
      template='seaborn',
      labels={'Count':'Number of Trips'},
      height=600,
      width=1500,
      text='Count'
      )
#fig2.update_traces(texttemplate='%{text:.1s}', textposition='outside')
#fig2.update_layout(title_text='Uber Rides by hour',title_x=0.5)
fig2.show()
```

Maximum number of rides are taken between 4-7 PM in a day

0.2.1 03. Distribution of Trips By Days in a Month

[ ]:
```
trips_by_days=pd.DataFrame(uber_data['Day'].value_counts()).reset_index()
trips_by_days.columns=['Days','Number of Trips']
trips_by_days=trips_by_days.sort_values(by='Days')
```

[ ]:
```
fig3 = px.histogram(trips_by_days,
                    x='Days',
                    y='Number of Trips',
                    height=600,
                    width=1500,
                    nbins=31,
                    template='seaborn',
                    color_discrete_sequence=['#C78845'])


for day, trips in zip(trips_by_days['Days'], trips_by_days['Number of Trips']):

    trips_text = f'{trips/1000:.0f}K'

    fig3.add_annotation(
```

```
        x=day,
        y=trips,
        text=trips_text,
        showarrow=True,
        arrowhead=3,
        ax=0,
        ay=-30
    )

fig3.update_layout(bargap=0.2)
fig3.update_layout(title_text='Distribution of trips by days in a Month',␣
 ↪title_x=0.5)
fig3.show()
```

[ ]:
```
fig_line = px.line(trips_by_days, x='Days', y='Number of Trips',␣
 ↪template='seaborn',
                   labels={'Number of Trips': 'Trips'},
                   title='Distribution of trips by days in a Month')
fig_line.show()
```

Day 30 has the miximum number of trips

0.2.2 04. Base locations with highest number of pickups

[ ]:
```
trips_by_loc=uber_data[['Base','Hour']].value_counts().reset_index()
trips_by_loc.columns=['Base','Hour','Number of Trips']
trips_by_loc
```

[ ]:
```
          Base  Hour  Number of Trips
0       Weiter    17           107590
1       Hinter    17           104759
2       Weiter    18           104196
3       Hinter    18           101050
4       Weiter    16            99353
..         …   …              …
115  Danach-NY     3             2798
116      Unter     1             2149
117      Unter     4             1815
118      Unter     3             1582
119      Unter     2             1466

[120 rows x 3 columns]
```

[ ]:
```
fig4=px.scatter(trips_by_loc,
            x='Hour',
            y='Number of Trips',
            color='Base',
```

```
            template='plotly_dark',
           ␣
 ↪color_discrete_sequence=['#50F9F1','#6AE5A8','#96C96A','#B7AA47','#C78845'])
fig4.update_layout(title_text='Trips by location and time of the day',␣
 ↪title_x=0.5)
fig4.show()
```

0.2.3 05. Cross Analysis between hours and weekdays

[ ]:
```
hour_week=uber_data.groupby(['Day_of_Week','Hour']).count()['Date/Time']
hour_week
```

[ ]:
```
Day_of_Week  Hour
Friday       0       13716
             1        8163
             2        5350
             3        6930
             4        8806
                     …
Wednesday    19      47017
             20      47772
             21      44553
             22      32868
             23      18146
Name: Date/Time, Length: 168, dtype: int64
```

[ ]:
```
pivot=hour_week.unstack()
pivot
```

[ ]:
```
Hour            0      1      2      3     4      5      6      7      8   \
Day_of_Week
Friday       13716   8163   5350   6930  8806  13450  23412  32061  31509
Monday        6436   3737   2938   6232  9640  15032  23746  31159  29265
Saturday     27633  19189  12710   9542  6846   7084   8579  11014  14411
Sunday       32877  23015  15436  10597  6374   6169   6596   8728  12128
Thursday      9293   5290   3719   5637  8505  14169  27065  37038  35431
Tuesday       6237   3509   2571   4494  7548  14241  26872  36599  33934
Wednesday     7644   4324   3141   4855  7511  13794  26943  36495  33826

Hour            9   …     14     15     16     17     18     19     20  \
Day_of_Week         …
Friday       25230  …  36206  43673  48169  51961  54762  49595  43542
Monday       22197  …  28157  32744  38770  42023  37000  34159  32849
Saturday     17669  …  31418  38769  43512  42844  45883  41098  38714
Sunday       16401  …  28151  31112  33038  31521  28291  25948  25076
Thursday     27812  …  36699  44442  50560  56704  55825  51907  51990
```

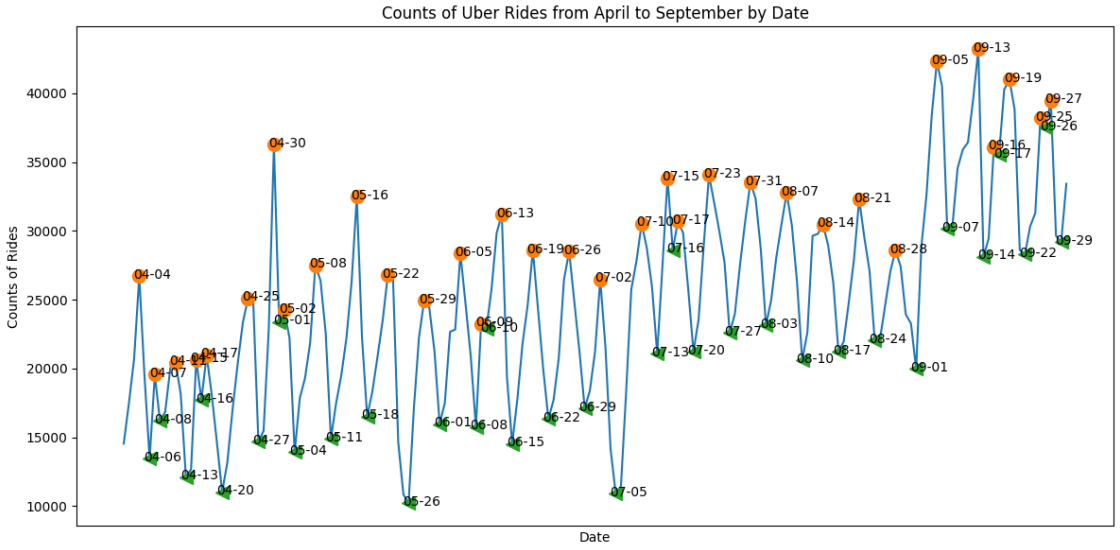
```
Tuesday      25023  …  34846  41338  48667  55500  50186  44789  44661
Wednesday    25635  …  35148  43388  50684  55637  52732  47017  47772

Hour            21     22     23
Day_of_Week
Friday       48323  49409  41260
Monday       28925  20158  11811
Saturday     43826  47951  43174
Sunday       23967  19566  12166
Thursday     51953  44194  27764
Tuesday      39913  27712  14869
Wednesday    44553  32868  18146

[7 rows x 24 columns]
```

[ ]:
```
fig = px.imshow(pivot,
                color_continuous_scale='blues',
                labels=dict(x="Hour", y="Day of Week", color="Values"))


fig.update_layout(title='Interactive Heatmap', autosize=False, width=800,␣
 ↪height=800)

fig.show()
```

0.3 Peaks and Pits

[ ]:
```
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(date_counts_concat.index, date_counts_concat.values)
ax.set_xlabel('Date')
ax.set_ylabel('Counts of Rides')
ax.set_title('Counts of Uber Rides from April to September by Date')

from scipy.signal import find_peaks
peaks, _ = find_peaks(date_counts_concat.values)
pits, _ = find_peaks(-date_counts_concat.values)
peaks_and_pits = np.sort(np.concatenate([peaks, pits]))

ax.plot(date_counts_concat.index[peaks], date_counts_concat.values[peaks], 'o',␣
 ↪markersize=10, label='Peaks')
ax.plot(date_counts_concat.index[pits], date_counts_concat.values[pits], '<',␣
 ↪markersize=10, label='Pits')
for i in range(len(peaks_and_pits)):
    date_str = date_counts_concat.index[peaks_and_pits[i]]
    date_str_without_year = date_str[5:]  # Assuming the date is formatted as␣
 ↪'YYYY-MM-DD'
```

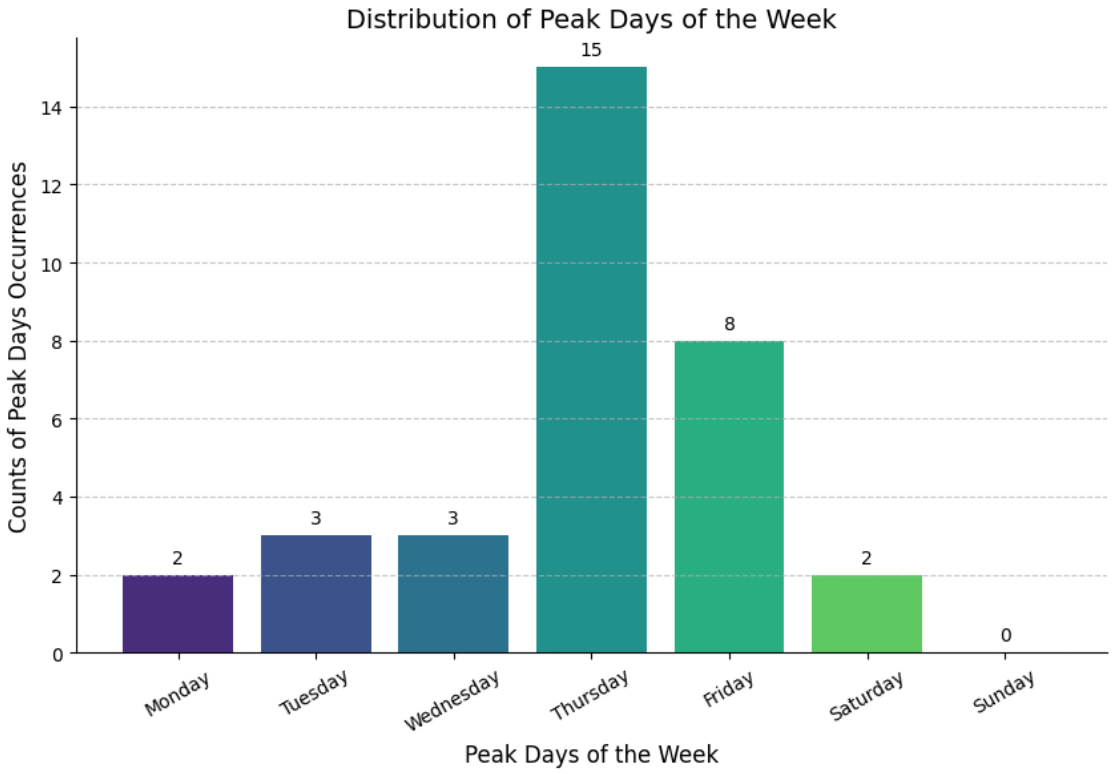
```
    ax.annotate(date_str_without_year,
                xy=(date_counts_concat.index[peaks_and_pits[i]],␣
 ↪date_counts_concat.values[peaks_and_pits[i]]),
                xytext=(10, 6),
                textcoords='offset points',
                fontsize=10,
                ha='center',
                va='top')

ax.set_xticks([])
plt.tight_layout()

plt.show()
```



[ ]:
```
peak_days_of_week= []

for i in peaks:
    peak_date= date_counts_concat.index[i]
    peak_day_of_week= peak_date.strftime('%A')
    peak_days_of_week.append(peak_day_of_week)

peak_days, counts= np.unique(peak_days_of_week, return_counts= True)

peak_df= pd.DataFrame(data=counts, index=peak_days, columns=['Counts'])
days_label= ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
 ↪'Saturday', 'Sunday']

peak_df= peak_df.reindex(days_label)
```

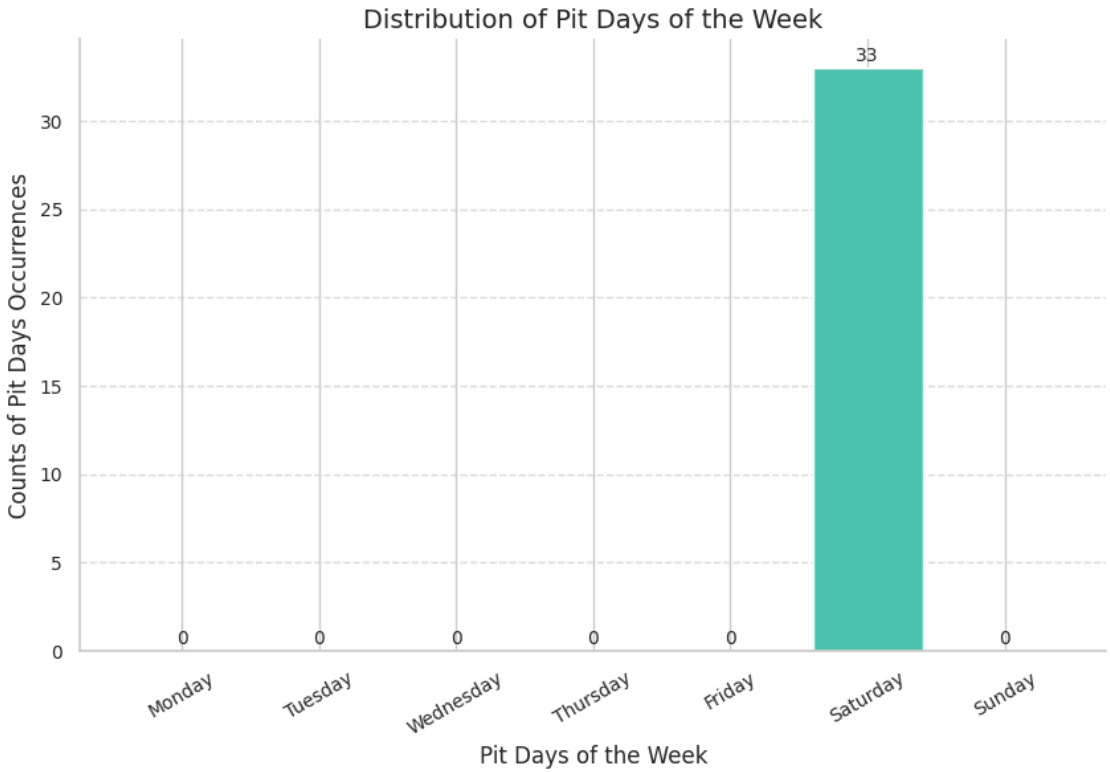
```
peak_df.fillna(0, inplace=True)
peak_df['Counts']= peak_df['Counts'].astype(int)
```

[ ]:
```
colors = sns.color_palette("viridis", len(peak_df))

plt.figure(figsize=(10, 6))
bars = plt.bar(peak_df.index, peak_df['Counts'], color=colors)

for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.2, int(yval),␣
 ↪ha='center', va='bottom', fontsize=10)

plt.xlabel('Peak Days of the Week', fontsize=12)
plt.ylabel('Counts of Peak Days Occurrences', fontsize=12)
plt.title('Distribution of Peak Days of the Week', fontsize=14)
plt.xticks(rotation=30, ha='center', fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)


sns.despine()

plt.show()
```



[ ]:
```
pits_days_of_week= []

for i in pits:
    pits_date= date_counts_concat.index[i]
    pits_day_of_week= peak_date.strftime('%A')
    pits_days_of_week.append(pits_day_of_week)

pits_days, counts= np.unique(pits_days_of_week, return_counts= True)

pits_df= pd.DataFrame(data=counts, index=pits_days, columns=['Counts'])
days_label= ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
 ↪'Saturday', 'Sunday']

pits_df= pits_df.reindex(days_label)
pits_df.fillna(0, inplace=True)
pits_df['Counts']= pits_df['Counts'].astype(int)
```

[ ]:
```
colors = sns.color_palette("mako", len(pits_df))

plt.figure(figsize=(10, 6))
bars = plt.bar(pits_df.index, pits_df['Counts'], color=colors)

for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.2, int(yval),␣
 ↪ha='center', va='bottom', fontsize=10)

plt.xlabel('Pit Days of the Week', fontsize=12)
plt.ylabel('Counts of Pit Days Occurrences', fontsize=12)
plt.title('Distribution of Pit Days of the Week', fontsize=14)
plt.xticks(rotation=30, ha='center', fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)

sns.despine()

plt.show()
```



Observation – from Peak and Pit Pattern:

Based on the visualizations of Uber ride data in New York City from April to September 2014, it was found that:

Thursdays and Fridays were the most frequent peak ride days, with the highest number of rides occurring on these days. This suggests that there may be increased demand for Uber rides on Thursdays and Fridays, possibly due to higher travel activity, events, or other factors during these weekdays.

All 33 pit days, or days with the lowest ride counts, were observed on Sunday, indicating relatively lower demand for Uber rides on Sunday during this period.

These observations from the half-year data align with our findings from April 2014, indicating that the observed patterns are not seasonal but rather continuous.

This also supports our proposal that higher commute needs during weekdays result in increased ride volume, while reduced activities on weekends lead to decreased ride demands.

Heatmap Analysis of Uber Ride Patterns in NYC: Validating Proposals To further validate our proposal regarding weekday commute needs and reduced ride demands on weekends –

We will create heatmaps using the 'Lat' and 'Lon' columns from the original data to visualize ride patterns on Thursday (peak day) and Sunday (off-peak day).

We will name these heatmaps Thursday_heatmap and Sunday_heatmap, respectively, and by comparing them, we can gain valuable insights.

[ ]:
```
import folium
from folium.plugins import HeatMap
from folium import plugins
```

[ ]:
```
thursday_data= uber_data[uber_data['Day_of_Week']=='Thursday']

heatmap_thursday= folium.Map(location= [40.7128,-74.0060], zoom_start= 12)

HeatMap(thursday_data[['Lat','Lon']].values).add_to(heatmap_thursday)

heatmap_thursday
```

[ ]:
```
<folium.folium.Map at 0x7ea64e37b7c0>
```

[ ]:
```
sunday_data= uber_data[uber_data['Day_of_Week']=='Sunday']

heatmap_sunday= folium.Map(location= [40.7128,-74.0060], zoom_start= 12)

HeatMap(sunday_data[['Lat','Lon']].values).add_to(heatmap_sunday)

heatmap_sunday
```

[ ]:
```
<folium.folium.Map at 0x7ea60f8dc340>
```

Observation 3 – from Heatmap:

Our observation of Uber ride patterns in NYC reveals that Thursdays consistently show the highest ride volume throughout the year –

The Thursday heatmap indicating a wider range of rides extending to suburban areas such as Bridgeport, Clverton, and Bridgewater Township.

In contrast, on Sundays, the rides are concentrated in areas like New Brunswick and downtown.

These two heatmaps provide compelling evidence to support our proposal that weekday commute needs are the primary drivers of ride demand, while reduced activity on weekends represents a key pain point.

0.3.1 Spatial Analysis - Map Visulization - Sept

[ ]:
```
from folium.plugins import HeatMapWithTime
```

[ ]:
```
# Calculating the mean latitude and longitude for the center of the map
mean_lat, mean_lon = data_sep['Lat'].mean(), data_sep['Lon'].mean()

uber_map = folium.Map(location=[mean_lat, mean_lon], zoom_start=12)
```

```
# Adding a heatmap to the map
heat_data = [[row['Lat'], row['Lon']] for index, row in data_sep.iterrows()]
HeatMap(heat_data).add_to(uber_map)
```

[ ]: <folium.plugins.heat_map.HeatMap at 0x7ea60f9210c0>

[ ]:
```
# uber_map.save("/mnt/data/uber_pickups_heatmap.html")

uber_map
```

[ ]: <folium.folium.Map at 0x7ea60f9226b0>

[ ]:
```
data_sep['Date/Time'] = pd.to_datetime(data_sep['Date/Time'])

# Create a map centered around the mean latitude and longitude
uber_map = folium.Map(location=[mean_lat, mean_lon], zoom_start=12)

# Prepare data for HeatMapWithTime
heat_data_with_time = [[[row['Lat'], row['Lon']] for index, row in␣
 ↪data_sep[data_sep['Date/Time'] == timestamp].iterrows()]
                       for timestamp in sorted(data_sep['Date/Time'].unique())]

HeatMapWithTime(heat_data_with_time, auto_play=True, radius=15).add_to(uber_map)

uber_map

uber_map.save("uber_Sept_heatmap_with_time.html")
```

[ ]:
```
data_sep['Date/Time'] = pd.to_datetime(data_sep['Date/Time'])

# Group the data by hour and prepare data for HeatMapWithTime
grouped_data = data_sep.groupby(data_sep['Date/Time'].dt.hour)[['Lat', 'Lon']].
 ↪apply(lambda x: x.values.tolist()).tolist()

# Create a map centered around the mean latitude and longitude
mean_lat, mean_lon = data_sep['Lat'].mean(), data_sep['Lon'].mean()
uber_map = folium.Map(location=[mean_lat, mean_lon], zoom_start=12)

# Adding a HeatMapWithTime to the map
HeatMapWithTime(grouped_data, auto_play=True, radius=15).add_to(uber_map)

uber_map

uber_map.save("uber_Sept_Hour_heatmap_with_time.html")
```

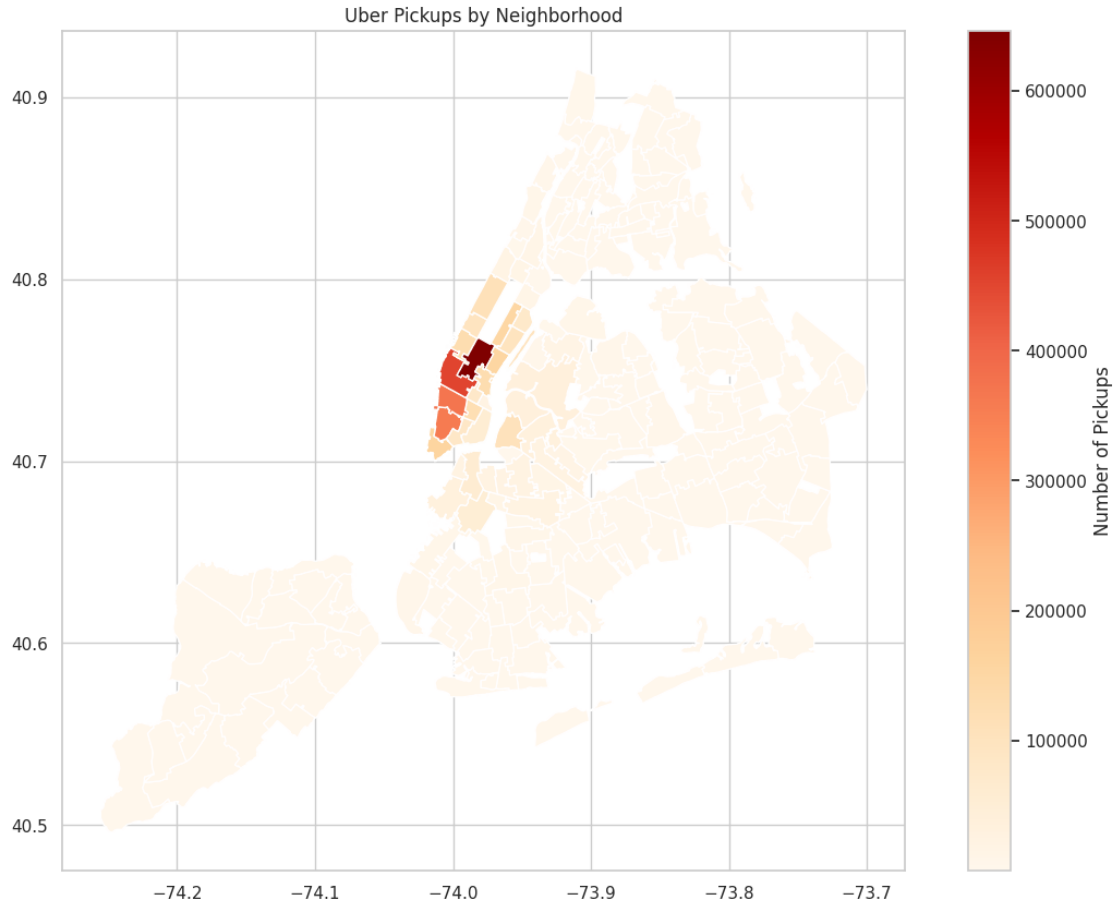
[ ]:
```
uber_map
```

[ ]: <folium.folium.Map at 0x7ea60f8fcac0>

[ ]:
```
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt
from shapely.geometry import Point

uber_data['geometry'] = uber_data.apply(lambda row: Point(row['Lon'],␣
 ↪row['Lat']), axis=1)
gdf = gpd.GeoDataFrame(uber_data, geometry='geometry')

neighborhoods = gpd.read_file('/content/drive/MyDrive/MS/Data Visualization/
 ↪Project/Uber_Data/nyc-neighborhoods.geo.json')

joined = gpd.sjoin(gdf, neighborhoods, how="inner", op='within')

pickup_counts = joined.groupby('name').size()

neighborhoods = neighborhoods.set_index('name').join(pickup_counts.
 ↪rename('pickup_count'))

fig, ax = plt.subplots(1, 1, figsize=(15, 10))
neighborhoods.plot(column='pickup_count', ax=ax, legend=True,
                   legend_kwds={'label': "Number of Pickups"},
                   cmap='OrRd')  # Or any other colormap
plt.title('Uber Pickups by Neighborhood')
plt.show()
```

```
WARNING:fiona._env:Non closed ring detected. To avoid accepting it, set the
OGR_GEOMETRY_ACCEPT_UNCLOSED_RING configuration option to NO
WARNING:fiona._env:Non closed ring detected. To avoid accepting it, set the
OGR_GEOMETRY_ACCEPT_UNCLOSED_RING configuration option to NO
WARNING:fiona._env:Non closed ring detected. To avoid accepting it, set the
OGR_GEOMETRY_ACCEPT_UNCLOSED_RING configuration option to NO
WARNING:fiona._env:Non closed ring detected. To avoid accepting it, set the
OGR_GEOMETRY_ACCEPT_UNCLOSED_RING configuration option to NO
/usr/local/lib/python3.10/dist-packages/IPython/core/interactiveshell.py:3473:
FutureWarning:

The `op` parameter is deprecated and will be removed in a future release. Please
use the `predicate` parameter instead.

<ipython-input-80-2a8ae9bffa2d>:13: UserWarning:

CRS mismatch between the CRS of left geometries and the CRS of right geometries.
Use `to_crs()` to reproject one of the input geometries to match the CRS of the
other.
```

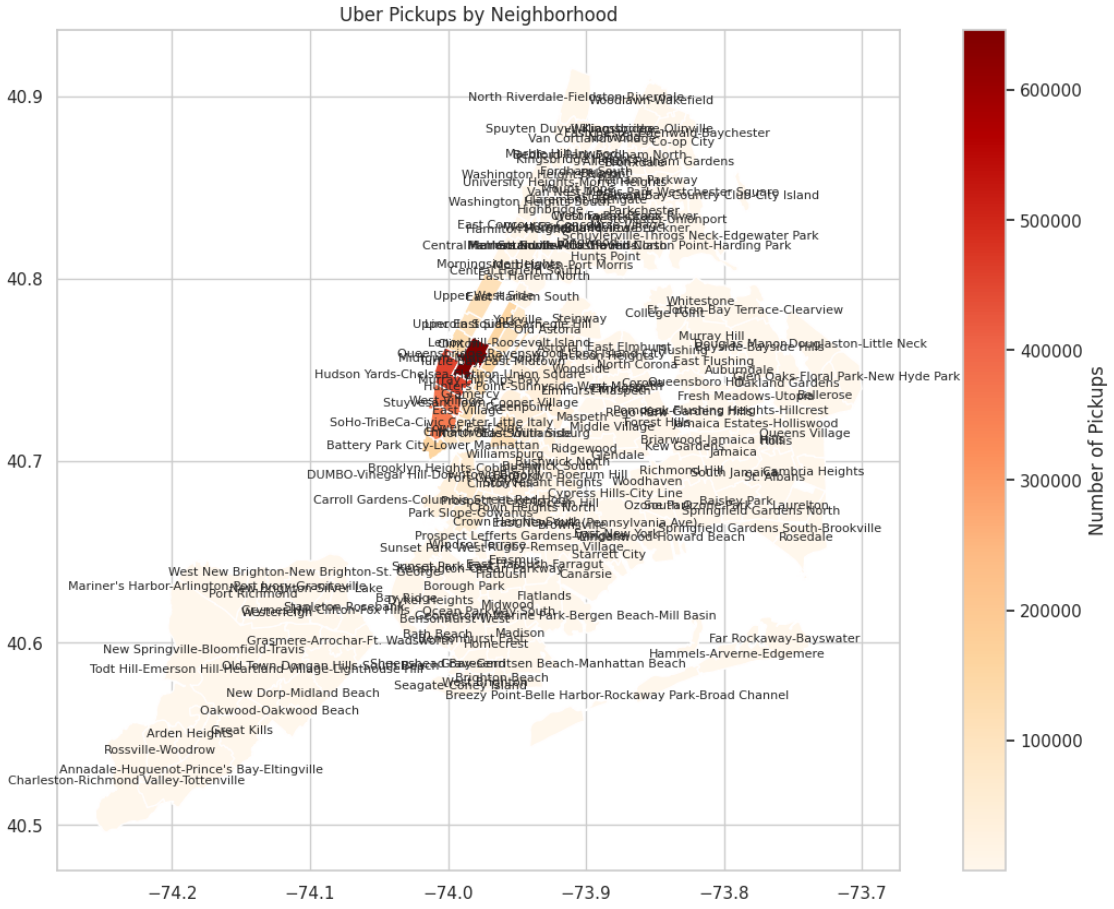
```
Left CRS: None
Right CRS: EPSG:4326
```



[ ]:
```
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt
from shapely.geometry import Point

uber_data['geometry'] = uber_data.apply(lambda row: Point(row['Lon'],␣
 ↪row['Lat']), axis=1)
gdf = gpd.GeoDataFrame(uber_data, geometry='geometry')

neighborhoods = gpd.read_file('/content/drive/MyDrive/MS/Data Visualization/
 ↪Project/Uber_Data/nyc-neighborhoods.geo.json')

joined = gpd.sjoin(gdf, neighborhoods, how="inner", op='within')
```

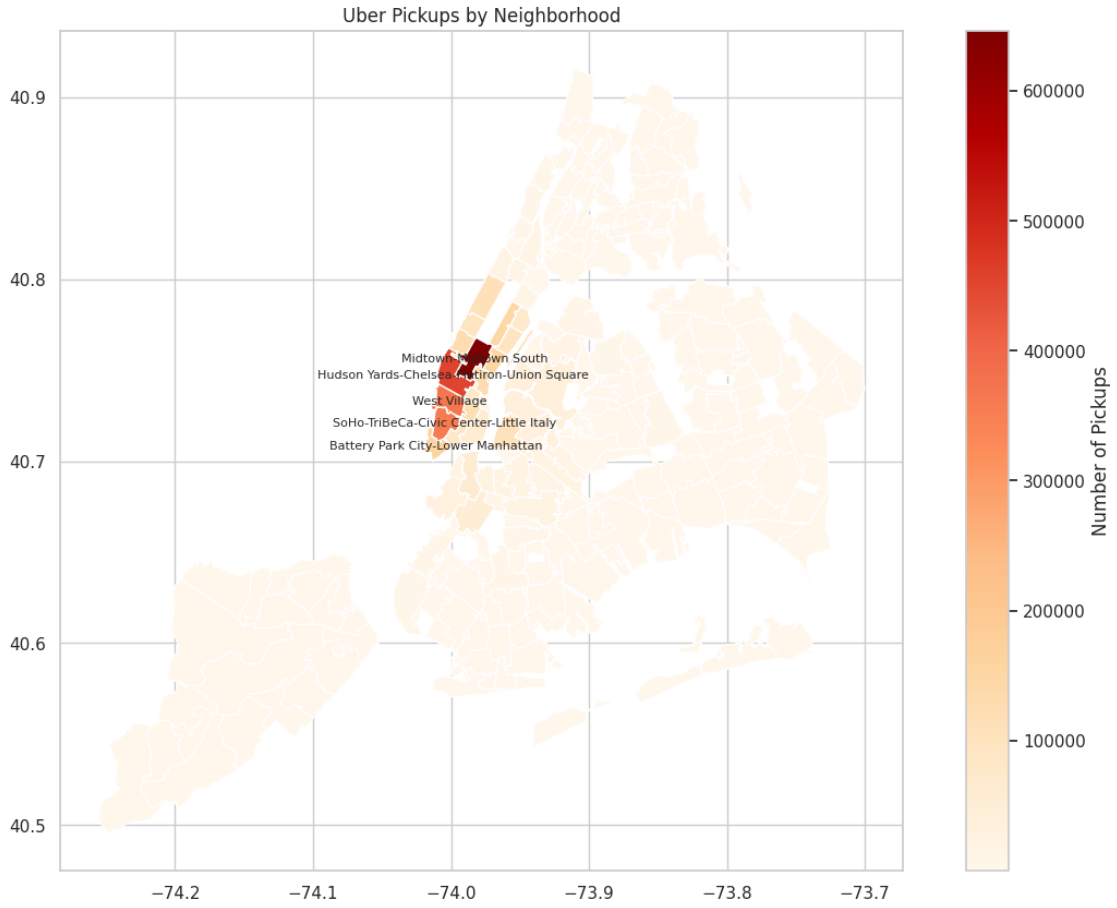
```
pickup_counts = joined.groupby('name').size()

neighborhoods = neighborhoods.set_index('name').join(pickup_counts.
 ↪rename('pickup_count'))

fig, ax = plt.subplots(1, 1, figsize=(15, 10))
neighborhoods.plot(column='pickup_count', ax=ax, legend=True,
                   legend_kwds={'label': "Number of Pickups"},
                   cmap='OrRd')

for x, y, label in zip(neighborhoods.geometry.centroid.x, neighborhoods.
 ↪geometry.centroid.y, neighborhoods.index):
    ax.text(x, y, label, fontsize=8, ha='center', va='center')

plt.title('Uber Pickups by Neighborhood')
plt.show()
```

```
WARNING:fiona._env:Non closed ring detected. To avoid accepting it, set the
OGR_GEOMETRY_ACCEPT_UNCLOSED_RING configuration option to NO
WARNING:fiona._env:Non closed ring detected. To avoid accepting it, set the
OGR_GEOMETRY_ACCEPT_UNCLOSED_RING configuration option to NO
WARNING:fiona._env:Non closed ring detected. To avoid accepting it, set the
OGR_GEOMETRY_ACCEPT_UNCLOSED_RING configuration option to NO
WARNING:fiona._env:Non closed ring detected. To avoid accepting it, set the
OGR_GEOMETRY_ACCEPT_UNCLOSED_RING configuration option to NO
/usr/local/lib/python3.10/dist-packages/IPython/core/interactiveshell.py:3473:
FutureWarning:

The `op` parameter is deprecated and will be removed in a future release. Please
use the `predicate` parameter instead.

<ipython-input-66-71f28e80aea4>:14: UserWarning:

CRS mismatch between the CRS of left geometries and the CRS of right geometries.
Use `to_crs()` to reproject one of the input geometries to match the CRS of the
other.

Left CRS: None
Right CRS: EPSG:4326


<ipython-input-66-71f28e80aea4>:29: UserWarning:

Geometry is in a geographic CRS. Results from 'centroid' are likely incorrect.
Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before this
operation.
```



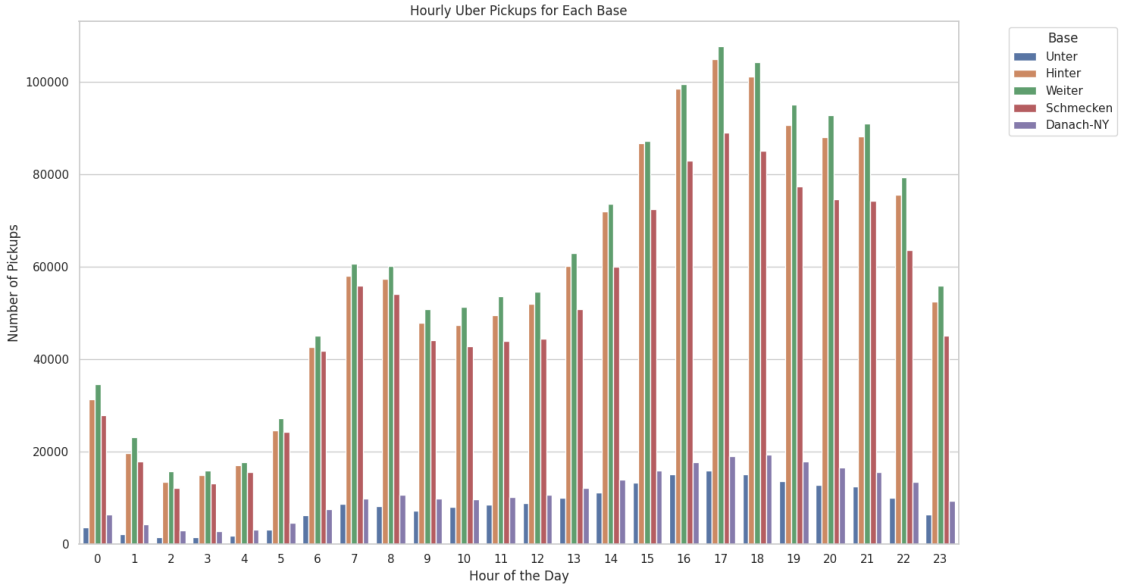
[ ]:
```
top_neighborhoods = neighborhoods.nlargest(5, 'pickup_count')

# Plotting the choropleth map
fig, ax = plt.subplots(1, 1, figsize=(15, 10))
neighborhoods.plot(column='pickup_count', ax=ax, legend=True,
                   legend_kwds={'label': "Number of Pickups"},
                   cmap='OrRd')  # Or any other colormap

# Add annotations for top 5 neighborhood names
for x, y, label in zip(top_neighborhoods.geometry.centroid.x, top_neighborhoods.
  ↪geometry.centroid.y, top_neighborhoods.index):
    ax.text(x, y, label, fontsize=8, ha='center', va='center')

plt.title('Uber Pickups by Neighborhood')
plt.show()
```

```
<ipython-input-81-07ca0e31f0cf>:32: UserWarning:

Geometry is in a geographic CRS. Results from 'centroid' are likely incorrect.
Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before this
operation.
```

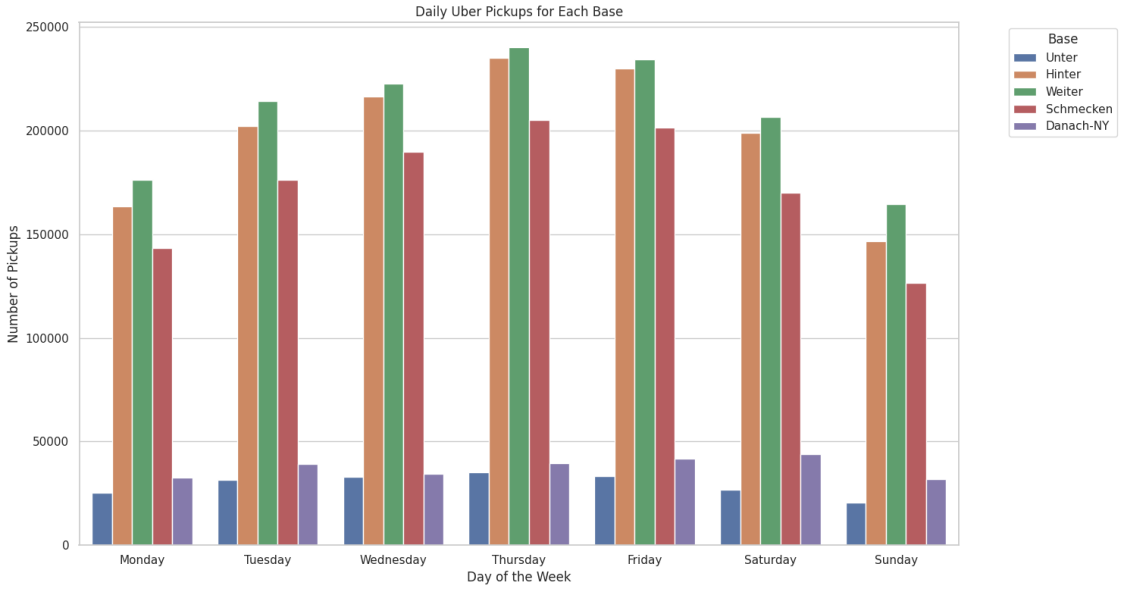


0.3.2 Base Code Analysis:

[ ]:
```
# Plotting hourly trends for each base
plt.figure(figsize=(15, 8))
sns.countplot(x='Hour', hue='Base', data=uber_data)
plt.title('Hourly Uber Pickups for Each Base')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Pickups')
plt.legend(title='Base', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



[ ]:
```
# Plotting daily trends for each base
plt.figure(figsize=(15, 8))
sns.countplot(x='Day_of_Week', hue='Base', data=uber_data, order=['Monday',␣
↪'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
plt.title('Daily Uber Pickups for Each Base')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Pickups')
plt.legend(title='Base', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



0.3.3 Combining Temporal and Spatial Insights

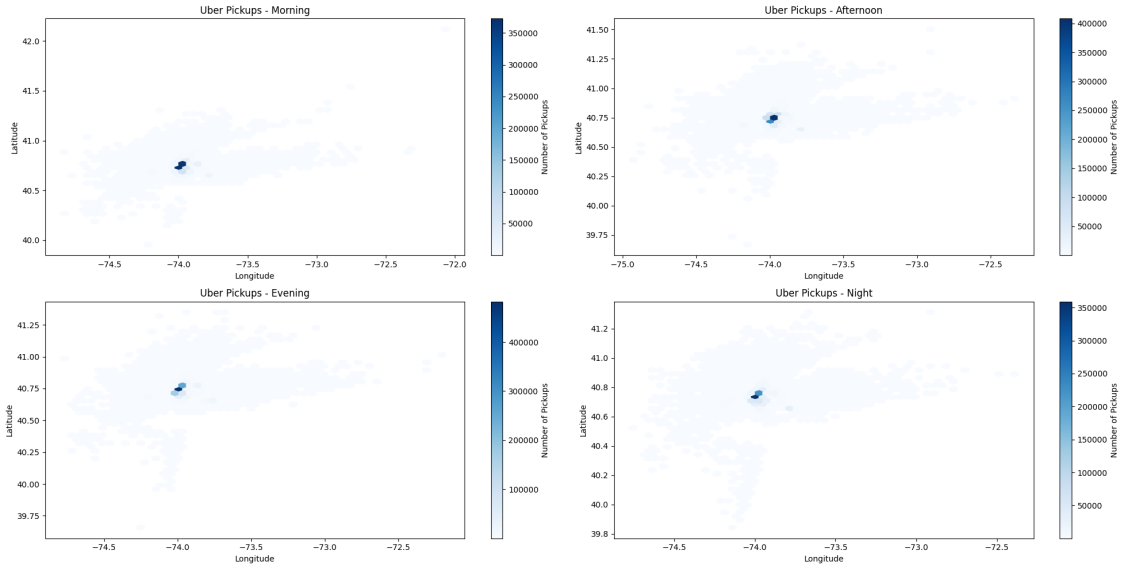
[ ]:
```
# Define time periods
def time_period(hour):
    if 5 <= hour < 12:
        return 'Morning'
    elif 12 <= hour < 17:
        return 'Afternoon'
    elif 17 <= hour < 21:
        return 'Evening'
    else:
        return 'Night'

# Apply the function to create a new column
uber_data['TimePeriod'] = uber_data['Hour'].apply(time_period)
```

[ ]:
```
uber_data.head()
```

[ ]:
```
            Date/Time      Lat      Lon    Base        Date  Month  Day  Hour  \
0 2014-04-01 00:11:00  40.7690 -73.9549  Unter  2014-04-01      4    1     0
1 2014-04-01 00:17:00  40.7267 -74.0345  Unter  2014-04-01      4    1     0
2 2014-04-01 00:21:00  40.7316 -73.9873  Unter  2014-04-01      4    1     0
3 2014-04-01 00:28:00  40.7588 -73.9776  Unter  2014-04-01      4    1     0
4 2014-04-01 00:33:00  40.7594 -73.9722  Unter  2014-04-01      4    1     0

   Minute Day_of_Week TimePeriod
0      11     Tuesday      Night
```

```
1       17      Tuesday       Night
2       21      Tuesday       Night
3       28      Tuesday       Night
4       33      Tuesday       Night
```

[ ]:
```python
# Define function for plotting
def plot_time_period(time_period):
    subset = uber_data[uber_data['TimePeriod'] == time_period]
    plt.hexbin(subset['Lon'], subset['Lat'], gridsize=50, cmap='Blues',␣
 ↪mincnt=1)
    plt.colorbar(label='Number of Pickups')
    plt.title(f'Uber Pickups - {time_period}')
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')

# Create plots for each time period
plt.figure(figsize=(20, 10))

for i, period in enumerate(['Morning', 'Afternoon', 'Evening', 'Night'], 1):
    plt.subplot(2, 2, i)
    plot_time_period(period)

plt.tight_layout()
plt.show()
```



[ ]:
```
!pip install bokeh
```

```
Requirement already satisfied: bokeh in /usr/local/lib/python3.10/dist-packages
```

```
(3.3.1)
Requirement already satisfied: Jinja2>=2.9 in /usr/local/lib/python3.10/dist-
packages (from bokeh) (3.1.2)
Requirement already satisfied: contourpy>=1 in /usr/local/lib/python3.10/dist-
packages (from bokeh) (1.2.0)
Requirement already satisfied: numpy>=1.16 in /usr/local/lib/python3.10/dist-
packages (from bokeh) (1.23.5)
Requirement already satisfied: packaging>=16.8 in
/usr/local/lib/python3.10/dist-packages (from bokeh) (23.2)
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.10/dist-
packages (from bokeh) (1.5.3)
Requirement already satisfied: pillow>=7.1.0 in /usr/local/lib/python3.10/dist-
packages (from bokeh) (9.4.0)
Requirement already satisfied: PyYAML>=3.10 in /usr/local/lib/python3.10/dist-
packages (from bokeh) (6.0.1)
Requirement already satisfied: tornado>=5.1 in /usr/local/lib/python3.10/dist-
packages (from bokeh) (6.3.2)
Requirement already satisfied: xyzservices>=2021.09.1 in
/usr/local/lib/python3.10/dist-packages (from bokeh) (2023.10.1)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from Jinja2>=2.9->bokeh) (2.1.3)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.2->bokeh) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.2->bokeh) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.1->pandas>=1.2->bokeh) (1.16.0)
```

[ ]:
```python
from bokeh.plotting import figure, show, output_notebook
from bokeh.models import ColumnDataSource, HoverTool
from bokeh.tile_providers import get_provider, CARTODBPOSITRON
```

```
BokehDeprecationWarning: 'tile_providers module' was deprecated in Bokeh 3.0.0
and will be removed, use 'add_tile directly' instead.
```

[ ]:
```python
# Convert Lat/Lon to Web Mercator coordinates for Bokeh plotting
def wgs84_to_web_mercator(df, lon="Lon", lat="Lat"):
    k = 6378137
    df["x"] = df[lon] * (k * np.pi/180.0)
    df["y"] = np.log(np.tan((90 + df[lat]) * np.pi/360.0)) * k
    return df

uber_data = wgs84_to_web_mercator(uber_data)

source = ColumnDataSource(uber_data)
```

```
output_notebook()
p = figure(title="Uber Pickups", x_axis_type="mercator", y_axis_type="mercator",
           x_axis_label='Longitude', y_axis_label='Latitude')
tile_provider = get_provider(CARTODBPOSITRON)
p.add_tile(tile_provider)


p.circle(x="x", y="y", source=source, size=8, color="blue", alpha=0.5)


hover = HoverTool()
hover.tooltips=[("Base", "@Base"), ("Hour", "@Hour"), ("Day", "@Day_of_Week")]
p.add_tools(hover)


show(p)
```

```
BokehDeprecationWarning: 'get_provider' was deprecated in Bokeh 3.0.0 and will
be removed, use 'add_tile directly' instead.
```

[ ]:
```
show(p)
```

[ ]:
```
sampled_data = uber_data.sample(frac=0.01, random_state=42)
```

[ ]:
```
import plotly.express as px

def interactive_plot_time_period(data, time_period):
    subset = data[data['TimePeriod'] == time_period]
    fig = px.density_mapbox(subset, lat='Lat', lon='Lon', z='Hour', radius=10,
                            center=dict(lat=40.75, lon=-73.97), zoom=10,
                            mapbox_style="stamen-terrain")
    fig.update_layout(title=f'Uber Pickups - {time_period}')
    return fig

for period in ['Morning', 'Afternoon', 'Evening', 'Night']:
    fig = interactive_plot_time_period(sampled_data, period)
    fig.show()
```

[ ]:
```
import plotly.express as px

def interactive_plot_time_period(data, time_period):
    subset = data[data['TimePeriod'] == time_period]
    fig = px.scatter_mapbox(subset, lat='Lat', lon='Lon', color='Hour',␣
 ↪size='Hour',
                            hover_data=['Hour'],
                            center=dict(lat=40.75, lon=-73.97), zoom=10,
                            mapbox_style="stamen-terrain")
```

```
    fig.update_layout(title=f'Uber Pickups - {time_period}')
    return fig

for period in ['Morning', 'Afternoon', 'Evening', 'Night']:
    fig = interactive_plot_time_period(sampled_data, period)
    fig.show()
```

[ ]:
```
import folium

def interactive_map_time_period(data, time_period):
    subset = data[data['TimePeriod'] == time_period]

    map_center = [subset['Lat'].mean(), subset['Lon'].mean()]
    uber_map = folium.Map(location=map_center, zoom_start=12)

    for _, row in subset.iterrows():
        folium.Marker(location=[row['Lat'], row['Lon']],
                      popup=f"Hour: {row['Hour']}",
                      icon=folium.Icon(color='blue')).add_to(uber_map)

    return uber_map

for period in ['Morning', 'Afternoon', 'Evening', 'Night']:
    uber_map = interactive_map_time_period(sampled_data, period)
    display(uber_map)
```

0.4 Part 2: Predictive Modeling

[4]:
```
# from sklearn.utils import shuffle
# uber_data = shuffle(uber_data)

uber_data = uber_data.sample(frac=1, random_state=42).reset_index(drop=True)
```

[5]:
```
uber_data
```

[5]:

	Date/Time	Lat	Lon	Base
0	4/10/2014 20:15:00	40.7588	-73.9726	B02617
1	7/20/2014 11:56:00	40.7653	-73.9724	B02682
2	6/17/2014 8:02:00	40.7444	-73.9771	B02598
3	4/28/2014 16:30:00	40.6449	-73.7824	B02682
4	9/17/2014 20:40:00	40.7636	-73.9798	B02598
…	…	…	…	…
4534322	5/21/2014 6:15:00	40.7195	-73.9996	B02682
4534323	6/14/2014 16:50:00	40.7392	-73.9972	B02617
4534324	7/11/2014 6:05:00	40.7489	-73.9769	B02617
4534325	9/22/2014 15:22:00	40.7489	-73.9762	B02682
4534326	6/3/2014 18:56:00	40.7088	-74.0024	B02682

```
[4534327 rows x 4 columns]
```

[6]:
```
uber_data.describe()
```

[6]:
```
                Lat           Lon
count  4.534327e+06  4.534327e+06
mean   4.073926e+01 -7.397302e+01
std    3.994991e-02  5.726670e-02
min    3.965690e+01 -7.492900e+01
25%    4.072110e+01 -7.399650e+01
50%    4.074220e+01 -7.398340e+01
75%    4.076100e+01 -7.396530e+01
max    4.211660e+01 -7.206660e+01
```

[7]:
```
uber_data['Date/Time'] = pd.to_datetime(uber_data['Date/Time'], format='%m/%d/
 ↪%Y %H:%M:%S')

# Create separate columns for date and time
uber_data['date'] = uber_data['Date/Time'].dt.date
uber_data['time'] = uber_data['Date/Time'].dt.time
uber_data['Day'] = uber_data['Date/Time'].dt.day
uber_data['DayOfWeek'] = uber_data['Date/Time'].dt.day_name()
uber_data['MonthName'] = uber_data['Date/Time'].dt.month_name()

# Display the result
uber_data.head()
```

[7]:
```
            Date/Time      Lat      Lon    Base        date      time  Day  \
0 2014-04-10 20:15:00  40.7588 -73.9726  B02617  2014-04-10  20:15:00   10
1 2014-07-20 11:56:00  40.7653 -73.9724  B02682  2014-07-20  11:56:00   20
2 2014-06-17 08:02:00  40.7444 -73.9771  B02598  2014-06-17  08:02:00   17
3 2014-04-28 16:30:00  40.6449 -73.7824  B02682  2014-04-28  16:30:00   28
4 2014-09-17 20:40:00  40.7636 -73.9798  B02598  2014-09-17  20:40:00   17

   DayOfWeek  MonthName
0   Thursday      April
1     Sunday       July
2    Tuesday       June
3     Monday      April
4  Wednesday  September
```

[9]:
```
avg_rides_per_day = uber_data.groupby(['MonthName', 'Day']).size().
 ↪groupby('MonthName').mean()
```

[10]:
```
avg_rides_per_day
```

[10]:
```
MonthName
April        18817.200000
August       26750.806452
July         25681.322581
June         22128.133333
May          21046.290323
September    34271.200000
dtype: float64
```

[11]:
```
avg_rides_per_day.round(0)
```

[11]:
```
MonthName
April        18817.0
August       26751.0
July         25681.0
June         22128.0
May          21046.0
September    34271.0
dtype: float64
```

[12]:
```
uber_data.head()
```

[12]:

	Date/Time	Lat	Lon	Base	date	time	Day	DayOfWeek	MonthName
0	2014-04-10 20:15:00	40.7588	-73.9726	B02617	2014-04-10	20:15:00	10	Thursday	April
1	2014-07-20 11:56:00	40.7653	-73.9724	B02682	2014-07-20	11:56:00	20	Sunday	July
2	2014-06-17 08:02:00	40.7444	-73.9771	B02598	2014-06-17	08:02:00	17	Tuesday	June
3	2014-04-28 16:30:00	40.6449	-73.7824	B02682	2014-04-28	16:30:00	28	Monday	April
4	2014-09-17 20:40:00	40.7636	-73.9798	B02598	2014-09-17	20:40:00	17	Wednesday	September

[13]:
```
uber_data.MonthName.value_counts()
```

[13]:
```
September    1028136
August        829275
July          796121
June          663844
May           652435
April         564516
Name: MonthName, dtype: int64
```

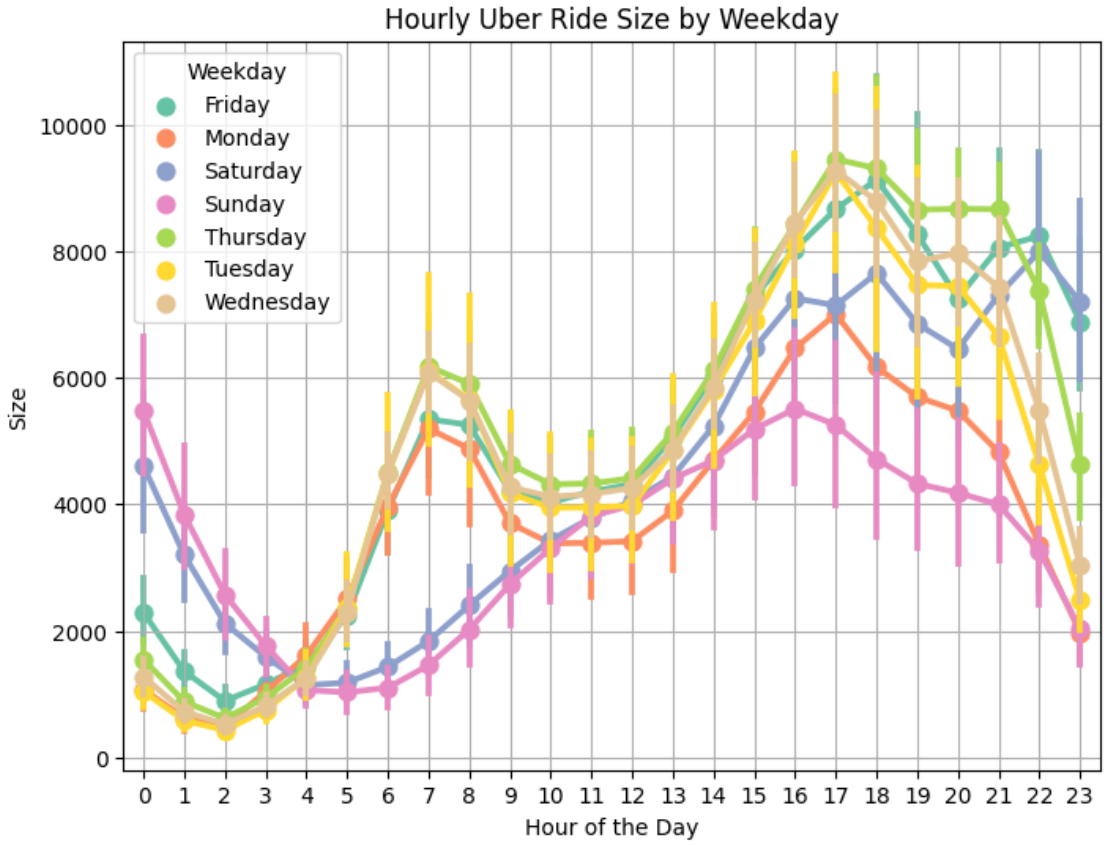
[14]:
```
uber_data['hour'] = uber_data['Date/Time'].dt.hour

summary = uber_data.groupby(['MonthName','DayOfWeek', 'hour'], as_index=False).
 ↪size()
summary
```

[14]:

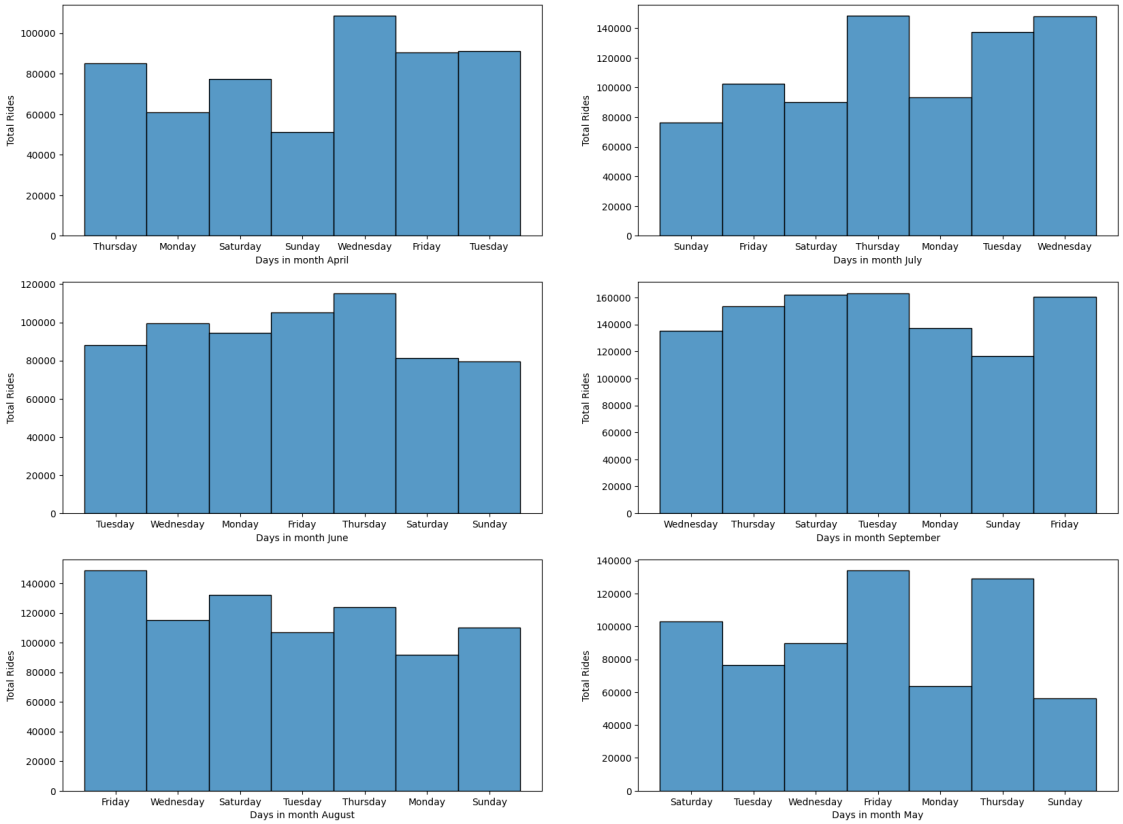
	MonthName	DayOfWeek	hour	size
0	April	Friday	0	1367
1	April	Friday	1	760
2	April	Friday	2	513
3	April	Friday	3	736
4	April	Friday	4	932
…	…	…	…	…
1003	September	Wednesday	19	9268
1004	September	Wednesday	20	9108
1005	September	Wednesday	21	7951
1006	September	Wednesday	22	6179
1007	September	Wednesday	23	3408

[1008 rows x 4 columns]

[15]:
```
import seaborn as sns

# Set the figure size
plt.figure(figsize=(8, 6))

# Create the point plot
sns.pointplot(x="hour", y="size", hue="DayOfWeek", data=summary, palette="Set2")

# Add labels and title
plt.xlabel("Hour of the Day")
plt.ylabel("Size")
plt.title("Hourly Uber Ride Size by Weekday")

# Customize the legend
plt.legend(title="Weekday")

# Show the plot
plt.grid(True)  # Add grid lines
plt.show()
```



[16]:
```
plt.figure(figsize=(20,15))
for i,month in enumerate(uber_data['MonthName'].unique()):
    plt.subplot(3,2,i+1)
    sns.histplot(data=uber_data,x=uber_data[uber_data['MonthName'] ==␣
 ↪month]['DayOfWeek'],bins=30)
    plt.xlabel('Days in month {}'.format(month))
    plt.ylabel('Total Rides')
```



[17]:
```
uber_trend = uber_data.groupby(['Lat', 'Lon'], as_index=False).size()
uber_trend
```

[17]:

	Lat	Lon	size
0	39.6569	-74.2258	1
1	39.6686	-74.1607	1
2	39.7214	-74.2446	1
3	39.8416	-74.1512	1
4	39.9055	-74.0791	1
…	…	…	…
574553	41.3730	-72.9237	1
574554	41.3737	-73.7988	1
574555	41.5016	-72.8987	1
574556	41.5276	-72.7734	2
574557	42.1166	-72.0666	1

[574558 rows x 3 columns]

[18]:
```
import folium
from folium.plugins import HeatMap
```

```
basemap = folium.Map()
HeatMap(uber_trend).add_to(basemap)
basemap
```

[18]:
```
<folium.folium.Map at 0x78ea01c55510>
```

[19]:
```
trips_by_loc=uber_data[['Base','hour']].value_counts().reset_index()
trips_by_loc.columns=['Base','Hour','Number of Trips']
trips_by_loc
```

[19]:

	Base	Hour	Number of Trips
0	B02617	17	107590
1	B02598	17	104759
2	B02617	18	104196
3	B02598	18	101050
4	B02617	16	99353
..	…	…	…
115	B02764	3	2798
116	B02512	1	2149
117	B02512	4	1815
118	B02512	3	1582
119	B02512	2	1466

[120 rows x 3 columns]

[20]:
```
# Sample 1000 random rows
sampled_data = uber_data.sample(n=50000, random_state=42)
```

Trying to findout the Busiest Day

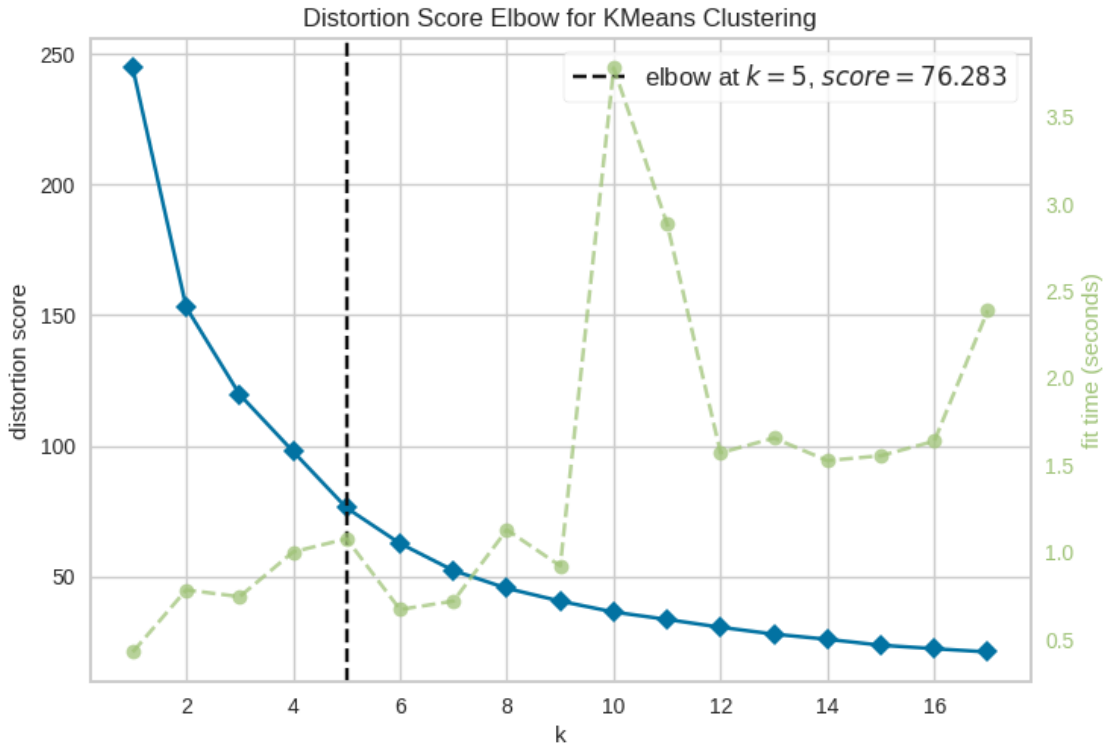
[21]:
```
clus_k_ori = sampled_data[['Lat', 'Lon']]
clus_k_ori.dtypes
```

[21]:
```
Lat    float64
Lon    float64
dtype: object
```

[23]:
```
import warnings
warnings.filterwarnings('ignore')
```

[24]:
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer

model_ori = KMeans()
visualizer = KElbowVisualizer(model_ori, k = (1, 18)) #k = 1 to 17
visualizer.fit(clus_k_ori)
```

```
visualizer.show()
```



[24]:
```
<Axes: title={'center': 'Distortion Score Elbow for KMeans Clustering'},
xlabel='k', ylabel='distortion score'>
```

[25]:
```
kmeans_ori = KMeans(n_clusters = 5, random_state = 0) #k = 5
kmeans_ori.fit(clus_k_ori)
```

[25]:
```
KMeans(n_clusters=5, random_state=0)
```

[26]:
```
centroids_k_ori = kmeans_ori.cluster_centers_
centroids_k_ori
```

[26]:
```
array([[ 40.66512325, -73.76473521],
       [ 40.71515673, -73.98928692],
       [ 40.7973785 , -73.88378307],
       [ 40.76140565, -73.97768552],
       [ 40.69569619, -74.20342886]])
```

[27]:
```
clocation_k_ori = pd.DataFrame(centroids_k_ori, columns = ['Latitude',␣
↪'Longitude'])
```

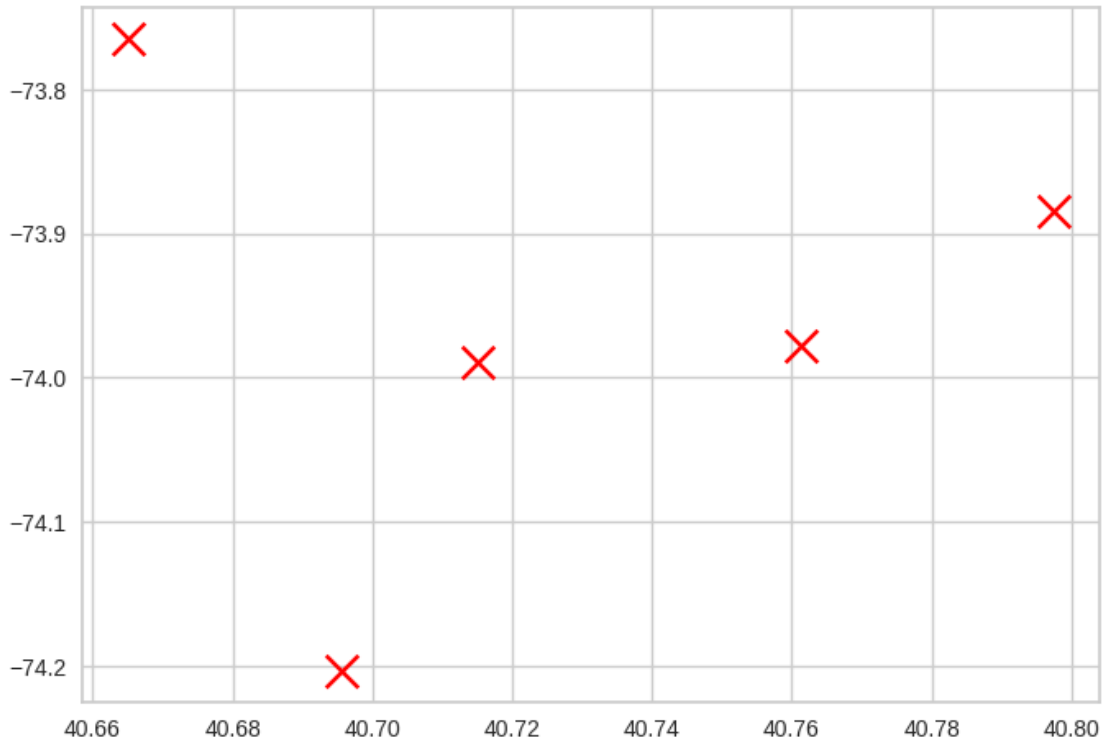
[28]:
```
clocation_k_ori
```

[28]:

	Latitude	Longitude
0	40.665123	-73.764735
1	40.715157	-73.989287
2	40.797379	-73.883783
3	40.761406	-73.977686
4	40.695696	-74.203429

[29]:
```
plt.scatter(clocation_k_ori['Latitude'], clocation_k_ori['Longitude'],␣
 ↪marker="x", color='red', s=200)
```

[29]: <matplotlib.collections.PathCollection at 0x78e9e5a67a00>



[30]:
```
centroid_k_ori = clocation_k_ori.values.tolist()

map_k_ori = folium.Map(location = [40.71600413400166, -73.98971408426613],␣
 ↪zoom_start = 10)
for point in range(0, len(centroid_k_ori)):
    folium.Marker(centroid_k_ori[point], popup = centroid_k_ori[point]).
 ↪add_to(map_k_ori)

map_k_ori.save("map_k_ori_scatter.html")

map_k_ori
```

[30]: <folium.folium.Map at 0x78ea03d93cd0>

Finding top 10 locations in each cluster

[31]:
```
import pandas as pd
import folium
from folium.plugins import MarkerCluster
from sklearn.cluster import KMeans
from IPython.display import display

# Assuming uber_data is your DataFrame with 'Lat' and 'Lon' columns
# Modify the code accordingly based on your actual DataFrame structure

# Fit KMeans clustering
kmeans = KMeans(n_clusters=5, random_state=42)
sampled_data['Cluster'] = kmeans.fit_predict(sampled_data[['Lat', 'Lon']])

# Create a folium map centered around the mean latitude and longitude
map_center = [sampled_data['Lat'].mean(), sampled_data['Lon'].mean()]
my_map = folium.Map(location=map_center, zoom_start=7)

# Loop through each cluster
for cluster_num in range(5):
    # Filter data for the current cluster
    cluster_data = sampled_data[sampled_data['Cluster'] == cluster_num]

    # Group by latitude and longitude, count occurrences, and sort by frequency
    coordinates_freq = cluster_data.groupby(['Lat', 'Lon']).size().
 ↪reset_index(name='Frequency')
    top_coordinates = coordinates_freq.nlargest(10, 'Frequency')

    # Add markers for the top 10 coordinates in the current cluster to the map
    marker_cluster = MarkerCluster().add_to(my_map)
    for index, row in top_coordinates.iterrows():
        folium.Marker([row['Lat'], row['Lon']],
                      popup=f"Cluster: {cluster_num}, Frequency:␣
 ↪{row['Frequency']}").add_to(marker_cluster)


my_map.save('hot_ten_location_in_each_cluster.html')

# Display the map in the notebook
display(my_map)
```

<folium.folium.Map at 0x78ea03d910f0>

[32]:
```
daily_pickups = uber_data.groupby(['MonthName', 'Day'])['hour'].count()
print('Busiest Day: {}'.format(daily_pickups.idxmax()))
print('Number of pickups: {}'.format(daily_pickups.max()))
```

```
Busiest Day: ('September', 13)
Number of pickups: 43205
```

[33]:
```
from collections import defaultdict
from datetime import datetime
from collections import OrderedDict


busiest_day = uber_data[
    (uber_data['MonthName'] == 'September') & (uber_data['Day'] == 13)]

# Extracting all pickups for a given hour
hourly = defaultdict(list)
for pickup in busiest_day.itertuples():
    pickup_time = datetime.strptime(
        '2014-9-13  {}'.format(pickup.hour), '%Y-%m-%d %H')
    hourly[str(pickup_time)].append([pickup.Lat, pickup.Lon])
hourly = OrderedDict(sorted(hourly.items(), key=lambda t: t[0]))
```

[34]:
```
import folium
from folium.plugins import HeatMapWithTime
from IPython.display import display

# Assuming hourly is a dictionary with timestamps as keys and corresponding␣
 ↪data as values
# Modify the code accordingly based on your actual data structure

# Create a Folium map with 'OpenStreetMap' as the base map
pickup_map = folium.Map(location=[uber_data['Lat'].mean(), uber_data['Lon'].
 ↪mean()], zoom_start=12, tiles="OpenStreetMap")

# Create HeatMapWithTime
hourly_pickups = HeatMapWithTime(
    data=list(hourly.values()),
    index=list(hourly.keys()),
    radius=10,
    auto_play=True,
    max_opacity=0.4
)
hourly_pickups.add_to(pickup_map)

# Display the map in the notebook
display(pickup_map)
```

```
<folium.folium.Map at 0x78e9e5beb760>
```

[35]:
```
# Function to make predictions
def make_prediction(user_input_lat, user_input_lon, model, threshold=20):
    user_input = [[user_input_lat, user_input_lon]]
    user_cluster = model.predict(user_input)[0]
    cluster_pickups = sampled_data[sampled_data['Cluster'] == user_cluster].
 ↪shape[0]

    if cluster_pickups > threshold:
        return f"Accept the ride. Historical pickups in this cluster:␣
 ↪{cluster_pickups}"
    else:
        return f"Reject the ride. Historical pickups in this cluster:␣
 ↪{cluster_pickups}"

# User input
user_input_lat = float(input("Enter the latitude: "))
user_input_lon = float(input("Enter the longitude: "))

# Example usage
prediction = make_prediction(user_input_lat, user_input_lon, kmeans,␣
 ↪threshold=20)
print(prediction)
```

```
Enter the latitude: 1
Enter the longitude: 34
Accept the ride. Historical pickups in this cluster: 1713
```

[36]:
```
from IPython.display import display, HTML, clear_output
import ipywidgets as widgets
import folium

# Create widgets for latitude, longitude, and radius input
latitude_input = widgets.FloatText(value=40.75, description='Latitude:')
longitude_input = widgets.FloatText(value=-73.99, description='Longitude:')
radius_input = widgets.FloatText(value=1, description='Radius (miles):')
submit_button = widgets.Button(description='Find Pickups')

# Function to handle button click event
def on_submit_button_clicked(b):
    clear_output()
    display(latitude_input, longitude_input, radius_input, submit_button)

    # Get the input values
    input_lat = latitude_input.value
    input_lon = longitude_input.value
```

```
    radius = radius_input.value

    # Create a Folium map centered around the input coordinates
    pickup_map = folium.Map(location=[input_lat, input_lon], zoom_start=14)

    # Add a marker for the entered location
    folium.Marker([input_lat, input_lon], popup=f"Entered Location:␣
 ↪{input_lat}, {input_lon}", icon=folium.Icon(color='blue')).add_to(pickup_map)

    # Display the map
    display(pickup_map)

    # Example usage of make_prediction function
    prediction = make_prediction(input_lat, input_lon, kmeans, threshold=20)
    print(prediction)

# Bind the button click event to the function
submit_button.on_click(on_submit_button_clicked)

# Display the widgets
display(latitude_input, longitude_input, radius_input, submit_button)
```

```
FloatText(value=40.75, description='Latitude:')
```

```
FloatText(value=-73.99, description='Longitude:')
```

```
FloatText(value=1.0, description='Radius (miles):')
```

```
Button(description='Find Pickups', style=ButtonStyle())
```

```
<folium.folium.Map at 0x78e9e5aa67d0>
```

```
Accept the ride. Historical pickups in this cluster: 33882
```

[37]:
```
from IPython.display import display, HTML, clear_output
import ipywidgets as widgets
from ipyleaflet import Map, Marker
from sklearn.cluster import KMeans

# # Sample DataFrame
# data = {'Date/Time': ['4/1/2014 0:11:00', '4/1/2014 0:17:00', '4/1/2014 0:21:
 ↪00', '4/1/2014 0:28:00', '4/1/2014 0:33:00'],
#         'Lat': [40.7690, 40.7267, 40.7316, 40.7588, 40.7594],
#         'Lon': [-73.9549, -74.0345, -73.9873, -73.9776, -73.9722],
#         'Base': ['B02512', 'B02512', 'B02512', 'B02512', 'B02512']}

# df = pd.DataFrame(data)
# df['Date/Time'] = pd.to_datetime(df['Date/Time'])
```

```
# # KMeans Clustering
# kmeans = KMeans(n_clusters=3, random_state=42)
# sampled_data['Cluster'] = kmeans.fit_predict(df[['Lat', 'Lon']])

# Function to make predictions
def make_prediction(user_input_lat, user_input_lon, model, threshold=2000):
    user_input = [[user_input_lat, user_input_lon]]
    user_cluster = model.predict(user_input)[0]
    cluster_pickups = sampled_data[sampled_data['Cluster'] == user_cluster].
 ↪shape[0]

    if cluster_pickups > threshold:
        return f"Accept the ride. Historical pickups in this cluster:␣
 ↪{cluster_pickups}"
    else:
        return f"Reject the ride. Historical pickups in this cluster:␣
 ↪{cluster_pickups}"

# Function to handle marker movement
def handle_marker_move(change):
    clear_output()
    display(map_widget, submit_button)

    # Get the new marker location
    lat, lon = marker.location

    # Update the latitude and longitude input widgets
    latitude_input.value = lat
    longitude_input.value = lon

# Create the map widget
initial_location = [40.75, -73.99]
map_widget = Map(center=initial_location, zoom=14)
marker = Marker(location=initial_location, draggable=True)
map_widget.add_layer(marker)

# Set the marker move event handler using observe
marker.observe(handle_marker_move, names='location')

# Create widgets for latitude, longitude, and radius input
latitude_input = widgets.FloatText(value=initial_location[0],␣
 ↪description='Latitude:')
longitude_input = widgets.FloatText(value=initial_location[1],␣
 ↪description='Longitude:')
radius_input = widgets.FloatText(value=1, description='Radius (miles):')
submit_button = widgets.Button(description='Decision')
```

```
# Function to handle button click event
def on_submit_button_clicked(b):
    clear_output()
    display(map_widget, submit_button)

    # Get the input values
    input_lat = latitude_input.value
    input_lon = longitude_input.value
    radius = radius_input.value

    # Example usage of make_prediction function
    prediction = make_prediction(input_lat, input_lon, kmeans, threshold=2000)
    print(prediction)

# Bind the button click event to the function
submit_button.on_click(on_submit_button_clicked)

# Display the widgets
display(map_widget, submit_button)
```

```
Map(bottom=1576606.0, center=[40.75, -73.99],␣
  ↪controls=(ZoomControl(options=['position', 'zoom_in_text', 'zoom…
```

```
Button(description='Decision', style=ButtonStyle())
```

```
Accept the ride. Historical pickups in this cluster: 33882
```

[ ]: