NYC Taxis Trips Analysis - Over 1 Million Records

Wanyun Yang, Wanzhen Li

Introduction

Transportation is an important part of our lives. Our research focused on the taxis trips in New York City (NYC). Compared to other transportation means, the most noticeable advantage of taxi service was a combination of both: a high degree of mobility and schedule flexibility. This is essential especially in NYC, most people use a form of taxi service every day. For our project, we used four datasets to develop our research about taxi services in NYC. Three out of four datasets came from Kaggle. The first dataset gave us the weather data of NYC from 2016. For example, the weather dataset contains, 'average temperature', 'maximum temperature', 'minimum temperature'. The second dataset provided information about NYC taxi drives in 2016. The values found consist of, 'pickup_datatime', 'dropoff_datatime', and 'pickup_latitude'. The last one shows NYC restaurants' inspection record including zip codes. The fourth dataset came from GitHub, and provides zip code, latitude, and longitude in the USA.

Questions

1.How do different factors affect the trip duration (weather, distance, and weekend or not)?

2.Do the peak hours for the weekday and the weekend distribute differently?

3.What areas have more dense pick-up spots for taxi drivers? Why?

Description and presentation of the tasks

For every dataset we converted data types, removed outliers to prepared all data for the join process, and used 'group by' to calculate new variables that we need. The first thing we did to the weather dataset was, drop all other columns but 'date' and 'average temperature'. Then for taxi drives dataset, we removed all extra spaces, converted the date column to the DateTime format, and generated three columns to represent pickup, date, and day. Then, we joined these two dataframes. After they were joined, we created a dummy variable for the weekend and weekdays and defined the distance function by latitude and longitude. Then, we changed the type of columns related to longitude and latitude to float and dropped three useless columns. For the USA zipcode dataset, we changed the columns' names to 'ZIP CODE', 'LAT', and 'LONG' during the join process. For the restaurants' dataset, we deleted the extra records for the same restaurants and dropped rows that have missing values. We used 'groupby' to calculate the number of restaurants by zip code and then joined the third and fourth datasets.

In [2]:
```
# Import the modules
import glob
import pandas as pd
import numpy as np
from math import sin, cos, sqrt, atan2, radians
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import plotly.graph_objects as go
import datetime as dt
```

In [3]:
```
# Check the current path
%pwd
```

Out[3]:
```
'C:\\Users\\yang7'
```

In [4]:
```
# Change the path
%cd C:/Users/yang7/OneDrive/Desktop/python
```

```
C:\Users\yang7\OneDrive\Desktop\python
```

In [5]:
```
# weather dataset import
weather = pd.read_csv (r'C:/Users/yang7/OneDrive/Desktop/python/weather_data_n
yc_centralpark_2016.csv')
weather.drop(["maximum temerature", "minimum temperature","precipitation","sno
w fall","snow depth"], axis = 1, inplace = True)
weather.head()
```

Out[5]:

	date	average temperature
0	01-01-16	39.0
1	02-01-16	35.0
2	03-01-16	39.5
3	04-01-16	24.0
4	05-01-16	19.5

In [6]:
```
# 2nd dataset of taxi drives in NYC from 2016 Jan to 2016 June
sample = pd.read_csv (r'C:/Users/yang7/OneDrive/Desktop/python/NYC Taxi Data S
et.csv')
sample.head()
```

Out[6]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude
0	id2875421	2	14-03-2016 17:24	14-03-2016 17:32	1	-73.982155
1	id2377394	1	12-06-2016 00:43	12-06-2016 00:54	1	-73.980415
2	id3858529	2	19-01-2016 11:35	19-01-2016 12:10	1	-73.979027
3	id3504673	2	06-04-2016 19:32	06-04-2016 19:39	1	-74.010040
4	id2181028	2	26-03-2016 13:30	26-03-2016 13:38	1	-73.973053

In [7]:
```
sample.shape
```

Out[7]:
```
(1048575, 11)
```

In [8]:
```
# Delete the space in front of the start_date column if any
sample.pickup_datetime = sample.pickup_datetime.str.lstrip()
```

In [9]:
```
# Extract a part of date as a new list for future use
# Convert all kinds of object in pickup time column to datetime with one unifo
rm format
import datetime
taxiDate = []
for i in range(1048575):
    if sample.iat[i,2][1]=='/' and sample.iat[i,2][3]=='/' :
        taxiDate.append(datetime.datetime.strptime(sample.iat[i,0][0:8], '%m/%
d/%Y').strftime('%d-%m-%y'))
    elif sample.iat[i,2][2]=='/' and sample.iat[i,2][4]=='/' :
        taxiDate.append(datetime.datetime.strptime(sample.iat[i,0][0:9], '%m/%
d/%Y').strftime('%d-%m-%y'))
    elif sample.iat[i,2][2]=='/' and sample.iat[i,2][5]=='/' :
        taxiDate.append(datetime.datetime.strptime(sample.iat[i,0][0:10], '%m/
%d/%Y').strftime('%d-%m-%y'))
    else:
        taxiDate.append(datetime.datetime.strptime(sample.iat[i,2][0:10], '%d-
%m-%Y').strftime('%d-%m-%y'))
```

In [9]:
```
# Generate three columns of pickup hour, date and the day of that day.
sample=sample.assign(pickup = ' ')
sample=sample.assign(date = ' ')
sample=sample.assign(day = ' ')
for i in range(1048575):
    sample.iat[i, -3] = sample.iat[i,3][-5:-3]
    sample.iat[i, -2] = taxiDate[i]
    sample.iat[i, -1] = datetime.datetime.strptime(sample.iat[i,-2], '%d-%m-%
y').strftime('%A')
sample.head()
```

Out[9]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude
0	id2875421	2	14-03-2016 17:24	14-03-2016 17:32	1	-73.982155
1	id2377394	1	12-06-2016 00:43	12-06-2016 00:54	1	-73.980415
2	id3858529	2	19-01-2016 11:35	19-01-2016 12:10	1	-73.979027
3	id3504673	2	06-04-2016 19:32	06-04-2016 19:39	1	-74.010040
4	id2181028	2	26-03-2016 13:30	26-03-2016 13:38	1	-73.973053

In [10]:
```
# Join two tables
data=pd.merge(weather, sample, on='date', how='right')
```

In [11]:
```
# Create a variable for weekend/weekdays
data=data.assign(weekend = 'weekdays')
for i in range(1048575):
    if data.iat[i, -2] == 'Saturday' or data.iat[i, -2] == 'Sunday':
        data.iat[i, -1]= 'weekends'
```

In [12]:
```
# define the distance function by lans and lons
def distance(lat1,lon1,lat2,lon2):
    # approximate radius of earth in km
    R = 6373.0
    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c
    return distance
```

In [13]:
```
# Create a new column and convert the lons and lats to floats
data=data.assign(distance_km = 0)
data[["pickup_longitude", "pickup_latitude","dropoff_longitude","dropoff_latit
ude"]] = data[["pickup_longitude", "pickup_latitude","dropoff_longitude","drop
off_latitude"]].astype(float)
data.dtypes
```

Out[13]:
```
date                    object
average temperature    float64
id                      object
vendor_id                int64
pickup_datetime         object
dropoff_datetime        object
passenger_count          int64
pickup_longitude       float64
pickup_latitude        float64
dropoff_longitude      float64
dropoff_latitude       float64
store_and_fwd_flag      object
trip_duration            int64
pickup                  object
day                     object
weekend                 object
distance_km              int64
dtype: object
```

In [14]:
```
# Transform to distance
for i in range(1048575):
    data.iat[i, -1]=distance(data.iat[i, 8],data.iat[i, 7],data.iat[i, 10],dat
a.iat[i, 9])
```

In [15]:
```
# Drop the useless cols.
data.drop(["store_and_fwd_flag", "vendor_id","id","dropoff_datetime"], axis =
1, inplace = True)
data.dropna()
data.head()
```

Out[15]:

	date	average temperature	pickup_datetime	passenger_count	pickup_longitude	pickup_latitude	drop
0	01-01-16	39.0	01-01-2016 10:45	1	-74.001610	40.740810	
1	01-01-16	39.0	01-01-2016 00:09	2	-73.984360	40.748985	
2	01-01-16	39.0	01-01-2016 03:49	2	-73.953148	40.791618	
3	01-01-16	39.0	01-01-2016 15:48	6	-74.005363	40.722340	
4	01-01-16	39.0	01-01-2016 09:47	1	-73.994072	40.751282	

In [16]:
```
# Import the 3rd dataset that has zipcode and locations
zipLocation = pd.read_csv('https://gist.githubusercontent.com/erichurst/788266
6/raw/5bdc46db47d9515269ab12ed6fb2850377fd869e/US%2520Zip%2520Codes%2520from%2
5202013%2520Government%2520Data')
# Change the column names for the join process
zipLocation.columns = ['ZIPCODE', 'LAT','LONG']
```

In [17]:
```
# Import the 4th dataset that has NYC restaurants' inspection record
# We will only use the zipcode to locate and calculate the sum of restaurants
 in that area
restaurants = pd.read_csv (r'C:/Users/yang7/OneDrive/Desktop/python/New_York_C
ity_Restaurant_Inspection_Results.csv')
restaurants.head()
```

Out[17]:

	CAMIS	DBA	BORO	BUILDING	STREET	ZIPCODE	PHONE	C DESCR
0	40511702	NOTARO RESTAURANT	MANHATTAN	635	SECOND AVENUE	10016.0	2126863400	
1	40511702	NOTARO RESTAURANT	MANHATTAN	635	SECOND AVENUE	10016.0	2126863400	
2	50046354	VITE BAR	QUEENS	2507	BROADWAY	11106.0	3478134702	
3	50061389	TACK'S CHINESE TAKE OUT	STATEN ISLAND	11C	HOLDEN BLVD	10314.0	7189839854	
4	41516263	NO QUARTER	BROOKLYN	8015	5 AVENUE	11209.0	7187019180	A

In [18]:
```
# Delete the extra records of the same restaurants
restaurants = restaurants.drop_duplicates(subset=['CAMIS'], keep='first')
restaurants = restaurants[['CAMIS','DBA','ZIPCODE']].copy()
# Drop the rows with missing value
restaurants.dropna()
restaurants.head()
```

Out[18]:

	CAMIS	DBA	ZIPCODE
0	40511702	NOTARO RESTAURANT	10016.0
2	50046354	VITE BAR	11106.0
3	50061389	TACK'S CHINESE TAKE OUT	10314.0
4	41516263	NO QUARTER	11209.0
5	50015855	KABAB HOUSE NYC	11355.0

In [19]:
```
# Calculate the num of restaurants by zip code
restaurantsZip=restaurants.groupby(by='ZIPCODE')['DBA'].describe()
# Prepare for join
restaurantsZip=restaurantsZip.reset_index()
```

In [20]:
```
# Join
restaurantsLocation=pd.merge(restaurantsZip, zipLocation, on='ZIPCODE', how='l
eft')
# Filter the zipcode with too less resturants
restaurantsLocation=restaurantsLocation[restaurantsLocation['count']>5]
# Convert the count column data type.
restaurantsLocation['count'] = restaurantsLocation[['count']].astype(float)
```

Description and presentation of your analysis

For question 1: How do different factors affect the trip duration (weather, distance, and weekend or not)?

Since our goal is to know the relationship among trip duration and weather/ distance/ weekday/ weekend/ passenger number, we did a linear regression model about the log (trip duration) and chose ‘average temperature’, ‘distance_km’, ‘weekend’, ‘passenger_count’ as the independent variables. Below are OLS Regression Results and the residual plot.

In [21]:
```
# Make a subset copy to use in this question
regression=data[['average temperature', 'distance_km', 'weekend','trip_duratio
n','passenger_count']].copy()
regression.head()
```

Out[21]:

	average temperature	distance_km	weekend	trip_duration	passenger_count
0	39.0	1	weekdays	383	1
1	39.0	3	weekdays	622	2
2	39.0	1	weekdays	185	2
3	39.0	2	weekdays	280	6
4	39.0	0	weekdays	166	1

In [22]:
```
# Determine the independent variables
cols = ['average temperature', 'distance_km', 'weekend','passenger_count']
cat_cols = ['weekend']
```

In [23]:
```
# Set up variables for multiple linear regression model
X = pd.get_dummies(regression[cols], columns=cat_cols, prefix='', prefix_sep=
'', drop_first=True)
X = sm.add_constant(X)
y = np.log(regression['trip_duration'])
```

```
C:\Users\yang7\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389: Fu
tureWarning:

Method .ptp is deprecated and will be removed in a future version. Use numpy.
ptp instead.
```

In [24]:
```
pd.concat([X, y], axis=1).head()
```

Out[24]:

	const	average temperature	distance_km	passenger_count	weekends	trip_duration
0	1.0	39.0	1	1	0	5.948035
1	1.0	39.0	3	2	0	6.432940
2	1.0	39.0	1	2	0	5.220356
3	1.0	39.0	2	6	0	5.634790
4	1.0	39.0	0	1	0	5.111988

In [25]:
```
# Fit linear regression model and report results
model = sm.OLS(endog=y, exog=X)
results = model.fit()
results.summary()
```

Out[25]:

OLS Regression Results

Dep. Variable:	trip_duration	**R-squared:**	0.317
Model:	OLS	**Adj. R-squared:**	0.317
Method:	Least Squares	**F-statistic:**	1.218e+05
Date:	Wed, 11 Dec 2019	**Prob (F-statistic):**	0.00
Time:	21:21:52	**Log-Likelihood:**	-1.0536e+06
No. Observations:	1048575	**AIC:**	2.107e+06
Df Residuals:	1048570	**BIC:**	2.107e+06
Df Model:	4		
Covariance Type:	nonrobust		

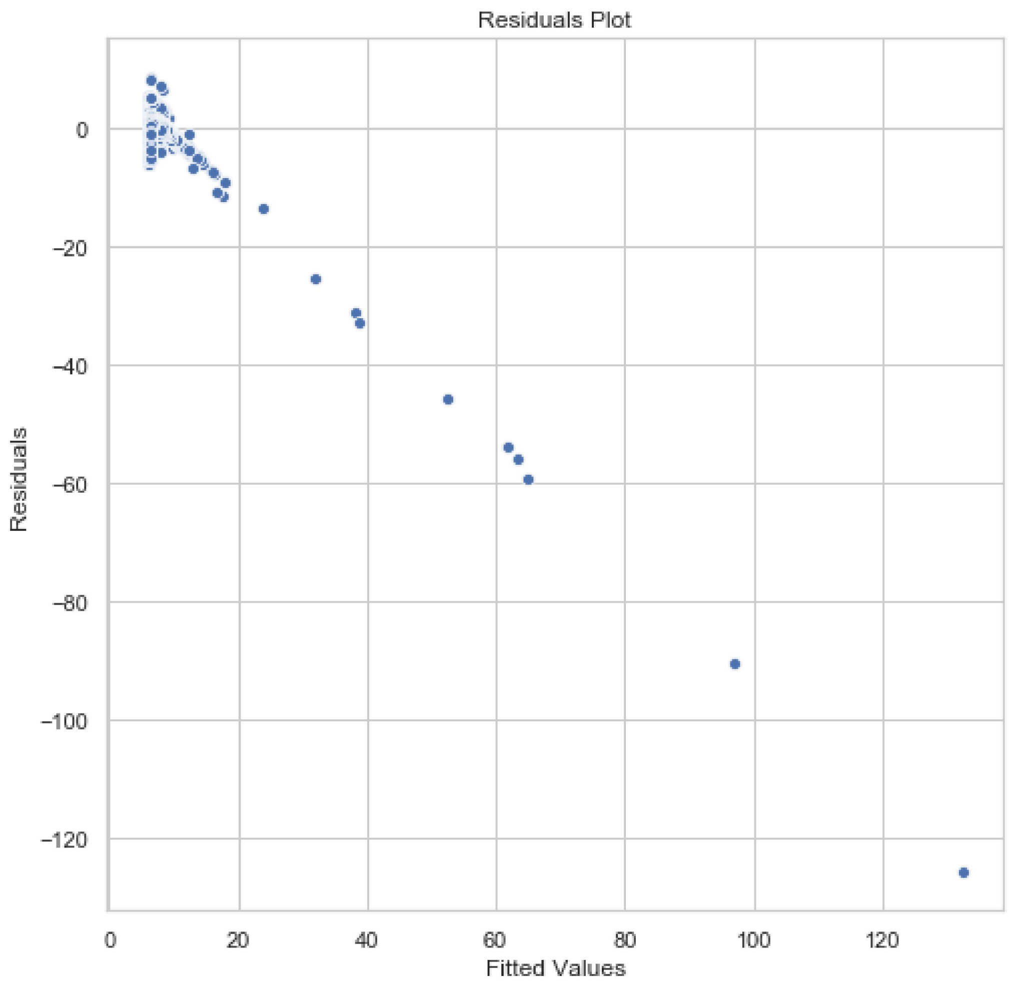
	coef	std err	t	P>|t|	[0.025	0.975]
const	6.0928	0.002	2540.385	0.000	6.088	6.098
average temperature	0.0017	4.26e-05	40.870	0.000	0.002	0.002
distance_km	0.1017	0.000	692.267	0.000	0.101	0.102
passenger_count	0.0103	0.000	20.876	0.000	0.009	0.011
weekends	-0.1046	0.001	-73.099	0.000	-0.107	-0.102

Omnibus:	1904852.198	**Durbin-Watson:**	1.982
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	137896785520.390
Skew:	-11.915	**Prob(JB):**	0.00
Kurtosis:	1779.413	**Cond. No.**	192.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [26]:
```
# Plot the residual information
plt.figure(figsize=(8,8))
sns.set(style="whitegrid")
ax = sns.scatterplot(x=results.fittedvalues,y=results.resid, marker='o', data=
regression)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals Plot')
```

Out[26]:
```
Text(0.5, 1.0, 'Residuals Plot')
```



As above, we tried the log-level regression model. First, we can tell that R-square is very small at 0.32, which means that only 32% of variation can be explained by our model. Second, although the P-values of all variables are extremely close to zero, which means the coefficients are significant, the coefficients of passenger count and average temperatures are small. Third, the residual plots showed many outliers in the dataset. It is supposed to show a bunch of random points around 0. However, we can see over 10 points with extreme error values. Therefore, we need to adjust the considered variables, also try to remove outliers and rerun the model.

In [27]:
```
# Make a copy
noOutlier=data
```

In [28]:
```
# Define a funtion to remove the outliers
def remove_outlier(df_in, col_name):
    q1 = df_in[col_name].quantile(0.25)
    q3 = df_in[col_name].quantile(0.75)
    iqr = q3-q1 #Interquartile range
    fence_low  = q1-1.5*iqr
    fence_high = q3+1.5*iqr
    df_out = df_in.loc[(df_in[col_name] > fence_low) & (df_in[col_name] < fenc
e_high)]
    return df_out
```

In [29]:
```
# Apply to the variables
noOutlier=remove_outlier(noOutlier,'average temperature')
noOutlier=remove_outlier(noOutlier,'passenger_count')
noOutlier=remove_outlier(noOutlier,'trip_duration')
noOutlier=remove_outlier(noOutlier,'distance_km')
```

In [30]:
```
noOutlier.shape
```

Out[30]:
```
(798405, 13)
```

In [31]:
```
cols = ['distance_km', 'weekend','passenger_count']
cat_cols = ['weekend']
```

In [32]:
```
# Set up variables for multiple linear regression model
X = pd.get_dummies(noOutlier[cols], columns=cat_cols, prefix='', prefix_sep=''
, drop_first=True)
X = sm.add_constant(X)
y = np.log(noOutlier['trip_duration'])
```

In [33]:
```
# Fit linear regression model and report results
model = sm.OLS(endog=y, exog=X)
results2 = model.fit()
results2.summary()
```

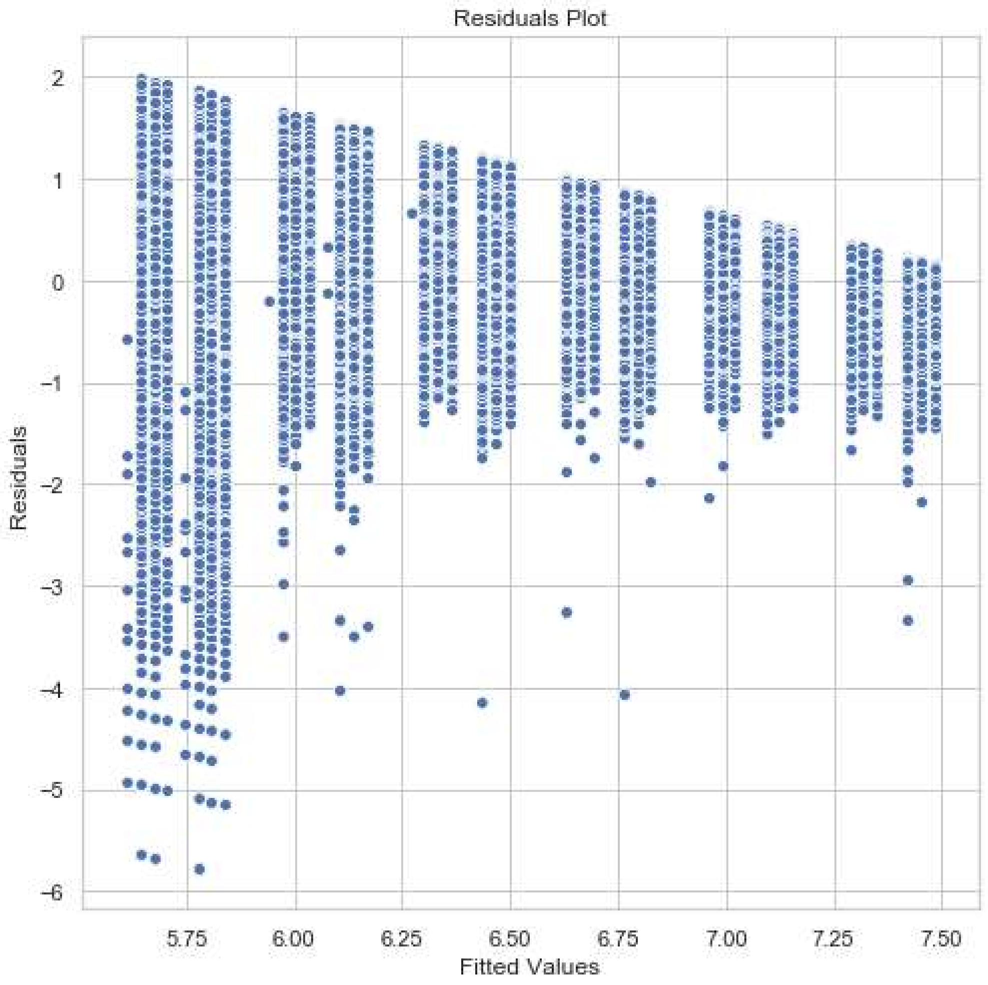
Out[33]:
```
OLS Regression Results

     Dep. Variable:     trip_duration        R-squared:       0.407
            Model:                OLS   Adj. R-squared:       0.407
           Method:      Least Squares      F-statistic:   1.827e+05
             Date:   Wed, 11 Dec 2019 Prob (F-statistic):      0.00
             Time:           21:22:21   Log-Likelihood: -6.3381e+05
No. Observations:             798405              AIC:   1.268e+06
    Df Residuals:             798401              BIC:   1.268e+06
         Df Model:                  3
  Covariance Type:          nonrobust

                    coef  std err          t  P>|t|  [0.025  0.975]
          const   5.7455    0.002   3347.456  0.000   5.742   5.749
     distance_km  0.3289    0.000    736.314  0.000   0.328   0.330
 passenger_count  0.0312    0.001     27.496  0.000   0.029   0.033
        weekends -0.1338    0.001   -100.471  0.000  -0.136  -0.131

       Omnibus: 278066.786     Durbin-Watson:       1.974
 Prob(Omnibus):      0.000  Jarque-Bera (JB): 2833956.075
          Skew:     -1.384          Prob(JB):        0.00
      Kurtosis:     11.805          Cond. No.        8.41


Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

When the distance increase by 1 km, the trip duration will increase by 32.89%. When the passenger number increases by 1, the trip duration will increase by 3.12%. When it is weekend and other variables remain the same, the trip duration will decrease 13.38% compared to weekdays.

Although the R-square is only 40.7%, I think the included independent variables do have an impact on trip duration, especially the direct distance between pick-up and drop-off spots, whether it is weekend or not and the passenger number. We just need more information about other potential factors to make it better.

In [34]:
```
# Plot the residual
plt.figure(figsize=(8,8))
sns.set(style="whitegrid")
ax = sns.scatterplot(x=results2.fittedvalues,y=results2.resid, marker='o', dat
a=noOutlier)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals Plot')
```

Out[34]:
```
Text(0.5, 1.0, 'Residuals Plot')
```



Here are the new OLS Regression Results and residual plot (ignored ‘average temperature’ since the coefficient is small and remove outliers).After, re-running the same model, we achieved a much better residual plot roughly within the range of -6 to.2. We can tell when the expected trip duration increases, we can get a more accurate result. When the expected trip duration is under 6, there is a bigger chance that the actual duration is shorter. This statistic result shows that all p-values look good to make sure coefficients in the model are significantly different from 0.

In [35]:
```
results2.params
```

Out[35]:
```
const              5.745510
distance_km        0.328904
passenger_count    0.031246
weekends          -0.133822
dtype: float64
```

In [36]:
```
#Confidence intervals ([0.025, 0.975])
results2.conf_int()
```

Out[36]:

	0	1
const	5.742146	5.748874
distance_km	0.328028	0.329779
passenger_count	0.029019	0.033473
weekends	-0.136433	-0.131212

In [37]:
```
#Calcualte the error bar lengths for confidence intervals.
err_series = results2.params - results2.conf_int()[0]
err_series
```

Out[37]:
```
const              0.003364
distance_km        0.000875
passenger_count    0.002227
weekends           0.002611
dtype: float64
```

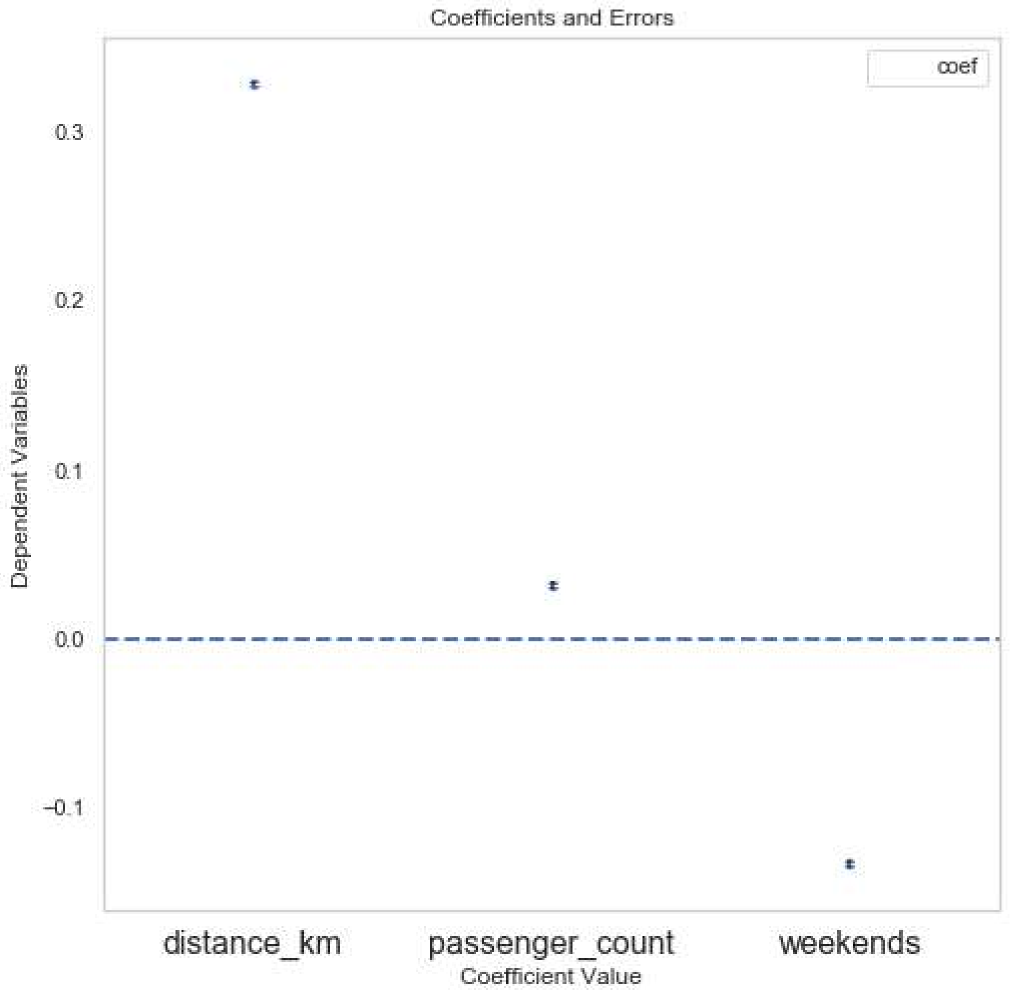
In [38]:
```
coef_df = pd.DataFrame({'coef': results2.params.values[1:],
                        'err': err_series.values[1:],
                        'varname': err_series.index.values[1:]
                       })

coef_df
```

Out[38]:

	coef	err	varname
0	0.328904	0.000875	distance_km
1	0.031246	0.002227	passenger_count
2	-0.133822	0.002611	weekends

In [39]:
```
# Plot the coeffients with their error bar
fig, ax = plt.subplots(figsize=(8, 8))
sns.set(style="white")
coef_df.plot(x='varname', y='coef', kind='bar',
             ax=ax, color='none',
             yerr='err', legend=False)
ax=sns.scatterplot(x=pd.np.arange(coef_df.shape[0]),y=coef_df['coef'], marker=
'o', data=coef_df)
ax.set_ylabel('Dependent Variables')
ax.set_xlabel('Coefficient Value')
ax.set_title('Coefficients and Errors')
ax.axhline(y=0, linestyle='--', color='b', linewidth=2)
ax.xaxis.set_ticks_position('none')
ax.set_xticklabels(['distance_km', 'passenger_count', 'weekends'],
                   rotation=0, fontsize=16)
```

Out[39]:
```
[Text(0, 0, 'distance_km'),
 Text(0, 0, 'passenger_count'),
 Text(0, 0, 'weekends')]
```



Then, we calculated the confidence interval and error for the confidence interval to check if the new regression is accurate. Based on the errors and coefficients of 'distance_km', 'weekend', 'passenger_count', we created a plot. We see that the errors are very small, which means the regression coefficient is measured precisely.

For question 2: Do the peak hours for the weekday and the weekend distribute differently?

First, we sorted values by ‘pickup’. If ‘day’ is Saturday or ‘Sunday’, we counted it as ‘weekends’. According to 7 days in a week, we used ‘groupby’ to count passengers each day and then one plot. For the plot, the x-axis shows pick-up time and the y-axis shows the total passenger number.

In [40]:
```
noOutlier.shape
```
Out[40]:
```
(798405, 13)
```

In [41]:
```
noOutlier.head()
```
Out[41]:

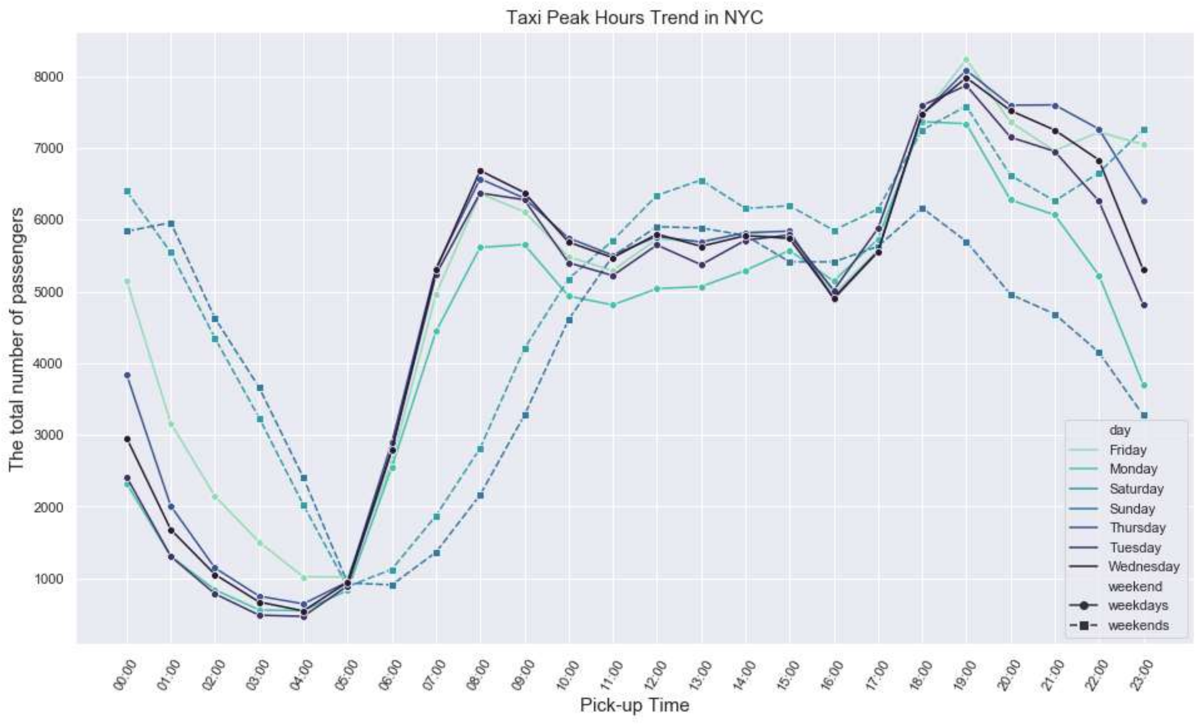
	date	average temperature	pickup_datetime	passenger_count	pickup_longitude	pickup_latitude	drop
0	01-01-16	39.0	01-01-2016 10:45	1	-74.001610	40.740810	
1	01-01-16	39.0	01-01-2016 00:09	2	-73.984360	40.748985	
2	01-01-16	39.0	01-01-2016 03:49	2	-73.953148	40.791618	
4	01-01-16	39.0	01-01-2016 09:47	1	-73.994072	40.751282	
5	01-01-16	39.0	01-01-2016 03:20	1	-73.996376	40.748837	

In [42]:
```
# Convert the pickup column into a datetime
for i in range(798405):
    noOutlier.iat[i,-4]=dt.datetime.strptime(noOutlier.iat[i,-4], '%H')
```

In [43]:
```
# Sort by the pickup hour
peak=noOutlier.sort_values('pickup')
```

In [44]:
```
# Make a dateframe of every day(monday/tuesday/...) and time and there passeng
er count information
peak.pickup=peak.pickup.dt.strftime("%H:00")
allPeak=peak.groupby(by=['pickup','day'])['passenger_count'].describe()
allPeak=allPeak.reset_index()
allPeak=allPeak.assign(weekend = 'weekdays')
for i in range(168):
    if allPeak.iat[i, 1] == 'Saturday' or allPeak.iat[i, 1] == 'Sunday':
        allPeak.iat[i, -1]= 'weekends'
```

In [45]:
```python
plt.figure(figsize = (16,9))
sns.set(style="darkgrid")
palette = sns.color_palette("mako_r", 7)
# Plot the overall peak hours changes in a graph
# Use different ways to show weekdays or weekends
# Use different color to show different days
overall=sns.lineplot(x='pickup', y="count",
             hue='day',style='weekend',
                     palette = palette,markers = ["o", "s"],
             data=allPeak)
for item in overall.get_xticklabels():
    item.set_rotation(60)
plt.title("Taxi Peak Hours Trend in NYC", fontsize = 15)
plt.xlabel("Pick-up Time", fontsize = 15)
plt.ylabel("The total number of passengers", fontsize = 15)
plt.show()
```



From the plot, we can tell that from Mondays to Thursdays the numbers of passengers are following the same pattern/trend, which is different from the weekend's pattern. Peak hours for Friday, Saturday, and Sunday tend to be later. Friday nights around 11 pm the number of passengers remains the same level instead of decreases like weekdays. The demand for a taxi at midnight goes up since Thursday night and then gets much bigger on weekends. No matter which day, the lowest demand is around 4 is and the highest demand is around 7 pm. However, for weekdays the second biggest peak is around 8 am, when is a reasonable time for going to work. For weekends, there are other two small peaks. One is around 1 am, when people hang out, and the other is 1 pm, probably implies the trips for lunch. Saturdays and Sundays have very similar trends overall except for the night. They both have a higher demand from 12 am to 4 am than weekdays.

For question 3: What areas have more dense pick-up spots for taxi drivers? Why?

We decided to use a mapboxto create a map for this question. We generated a column of random values first. Then we sorted the table by the random numbers. Using for loop, we added a dummy variable called ‘sample’ that tags every 20th row. The random sampling helped us decrease the huge dataset we got to 5% of its sample size. After sampling, the sample size became 79841. When loading the data by using mapbox, we can see an NYC map that shows pick up spots and restaurants density by zip codes.

In [46]:
```
# Your mapbox token
mapbox_access_token = 'pk.eyJ1Ijoid2FueXVuLXlhbmciLCJhIjoiY2syb3E4cTU5MTZhbDNt
bzNyejRxZDAzbSJ9.V9aZq1zuZ7bovxHrjfce6g'
```

In [47]:
```
# Add a dummy variable to the DataFrame called "sample" that tags every 20th r
ow to be included in the sample
# 10% of total data size
mapSample=noOutlier.assign(sample = 0)
```

In [48]:
```
mapSample.shape
```

Out[48]:
```
(798405, 14)
```

In [49]:
```
# Assign random numbers
np.random.seed( 30 )
mapSample['random number'] = np.random.randint(0,1000,size=(len(mapSample),1))
```

In [50]:
```
mapSample=mapSample.sort_values('random number')
```

In [51]:
```
# Random sampling
for i in range(0,len(mapSample)):
    if i % 20 == 0:
        mapSample.iat[i, -2] = 1
```

In [52]:
```
mapSample
```

Out[52]:

	date	average temperature	pickup_datetime	passenger_count	pickup_longitude	pickup_latitude
498668	28-03-16	49.5	28-03-2016 10:25	1	-73.980820	40.763691
945001	12-06-16	72.0	12-06-2016 04:34	3	-73.989082	40.720360
394948	10-03-16	59.5	10-03-2016 15:00	2	-73.959854	40.773434
811834	19-05-16	60.0	19-05-2016 22:11	1	-73.978241	40.773174
988568	20-06-16	71.0	20-06-2016 13:19	1	-74.001160	40.746826
626106	17-04-16	53.5	17-04-2016 23:18	1	-74.005928	40.740181
301677	24-02-16	48.0	24-02-2016 21:18	1	-73.988449	40.723351
163451	31-01-16	42.5	31-01-2016 22:08	1	-73.966888	40.693226
900102	04-06-16	72.5	04-06-2016 22:16	1	-73.979568	40.743835
942816	11-06-16	61.5	11-06-2016 08:59	1	-73.955444	40.772587
700725	30-04-16	53.5	30-04-2016 20:17	1	-73.965698	40.795399
811781	19-05-16	60.0	19-05-2016 08:31	1	-73.974159	40.737232
256903	16-02-16	43.5	16-02-2016 13:19	1	-73.974922	40.761871
360763	04-03-16	33.0	04-03-2016 16:31	1	-73.964813	40.767376
206400	08-02-16	32.0	08-02-2016 18:06	1	-73.988472	40.716572
414093	13-03-16	53.0	13-03-2016 00:27	1	-73.989197	40.740849
707662	02-05-16	50.5	02-05-2016 11:04	1	-73.973961	40.762833

	date	average temperature	pickup_datetime	passenger_count	pickup_longitude	pickup_latitude
893149	03-06-16	64.0	03-06-2016 22:28	3	-74.006668	40.732964
487448	26-03-16	42.5	26-03-2016 13:02	2	-74.001137	40.741528
411555	13-03-16	53.0	13-03-2016 18:21	1	-73.982063	40.731926
294664	23-02-16	36.5	23-02-2016 13:58	1	-73.991974	40.744694
591598	12-04-16	49.0	12-04-2016 20:53	1	-73.967583	40.762550
913516	06-06-16	74.5	06-06-2016 20:43	1	-73.989800	40.741398
612395	15-04-16	48.5	15-04-2016 08:35	1	-73.978218	40.748432
883633	01-06-16	70.5	01-06-2016 12:51	1	-73.982803	40.735485
693664	29-04-16	49.0	29-04-2016 19:08	1	-73.973084	40.748653
954001	13-06-16	66.5	13-06-2016 22:16	1	-73.987190	40.760696
181364	04-02-16	48.0	04-02-2016 23:33	2	-74.000000	40.734440
148135	29-01-16	37.0	29-01-2016 11:00	1	-73.994072	40.751255
317679	26-02-16	32.5	26-02-2016 13:16	1	-73.997818	40.756374
...	...	...	...	...	...	...
1027372	27-06-16	74.0	27-06-2016 05:32	1	-74.001038	40.736794
307723	25-02-16	50.0	25-02-2016 23:30	2	-73.975372	40.751640
193917	06-02-16	27.5	06-02-2016 14:08	1	-73.997139	40.735806

	date	average temperature	pickup_datetime	passenger_count	pickup_longitude	pickup_latitude
44744	09-01-16	42.0	09-01-2016 02:17	1	-73.975204	40.750736
108672	20-01-16	32.5	20-01-2016 09:41	1	-73.964943	40.772690
879844	31-05-16	75.0	31-05-2016 18:42	1	-73.978371	40.778881
192897	05-02-16	32.0	05-02-2016 21:07	1	-73.980087	40.755692
557353	06-04-16	36.0	06-04-2016 08:36	1	-73.950485	40.775146
996335	21-06-16	78.5	21-06-2016 09:01	1	-73.995827	40.694527
965526	15-06-16	71.5	15-06-2016 05:19	1	-73.951286	40.783569
71817	14-01-16	29.0	14-01-2016 15:44	2	-73.954010	40.771175
618499	16-04-16	53.5	16-04-2016 11:48	1	-74.001122	40.725910
14234	03-01-16	39.5	03-01-2016 21:22	3	-73.970749	40.764141
285483	21-02-16	47.0	21-02-2016 13:42	3	-73.982140	40.776459
1006589	23-06-16	70.0	23-06-2016 09:21	1	-73.953598	40.778591
557465	07-04-16	51.5	07-04-2016 08:56	1	-73.961159	40.766762
415542	13-03-16	53.0	13-03-2016 01:08	1	-73.982384	40.756676
993387	21-06-16	78.5	21-06-2016 23:31	1	-74.004585	40.748039
432556	16-03-16	50.0	16-03-2016 10:01	1	-73.976486	40.744205
809991	19-05-16	60.0	19-05-2016 18:05	1	-73.972168	40.761551

	date	average temperature	pickup_datetime	passenger_count	pickup_longitude	pickup_latitude
236573	13-02-16	14.0	13-02-2016 20:34	1	-73.977585	40.783932
599602	13-04-16	42.5	13-04-2016 18:58	3	-73.984398	40.757740
870655	29-05-16	75.0	29-05-2016 13:20	1	-74.006683	40.730572
981648	18-06-16	70.0	18-06-2016 19:50	1	-73.970131	40.756718
593425	12-04-16	49.0	12-04-2016 18:06	2	-73.961151	40.765114
484190	25-03-16	52.0	25-03-2016 09:17	1	-73.954247	40.767220
360498	04-03-16	33.0	04-03-2016 00:29	1	-73.982201	40.755962
867236	29-05-16	75.0	29-05-2016 18:42	1	-74.017693	40.706825
14290	03-01-16	39.5	03-01-2016 13:17	1	-73.963905	40.774357
977491	17-06-16	69.5	17-06-2016 03:07	1	-73.991760	40.726112

798405 rows × 15 columns

In [53]:
```
mapSample=mapSample[mapSample['sample']==1]
```

In [54]:
```
mapSample['pickup_longitude'].describe()
# the mean longitude of the sample is - -73.979750, which will be used as zoom
center later.
```

Out[54]:
```
count    39921.000000
mean       -73.979689
std          0.021787
min        -74.449074
25%        -73.991974
50%        -73.982086
75%        -73.969620
max        -73.631432
Name: pickup_longitude, dtype: float64
```

In [55]:
```
mapSample['pickup_latitude'].describe()
# the mean latitude of the sample is  40.753085, which will be used as zoom ce
nter later.
```

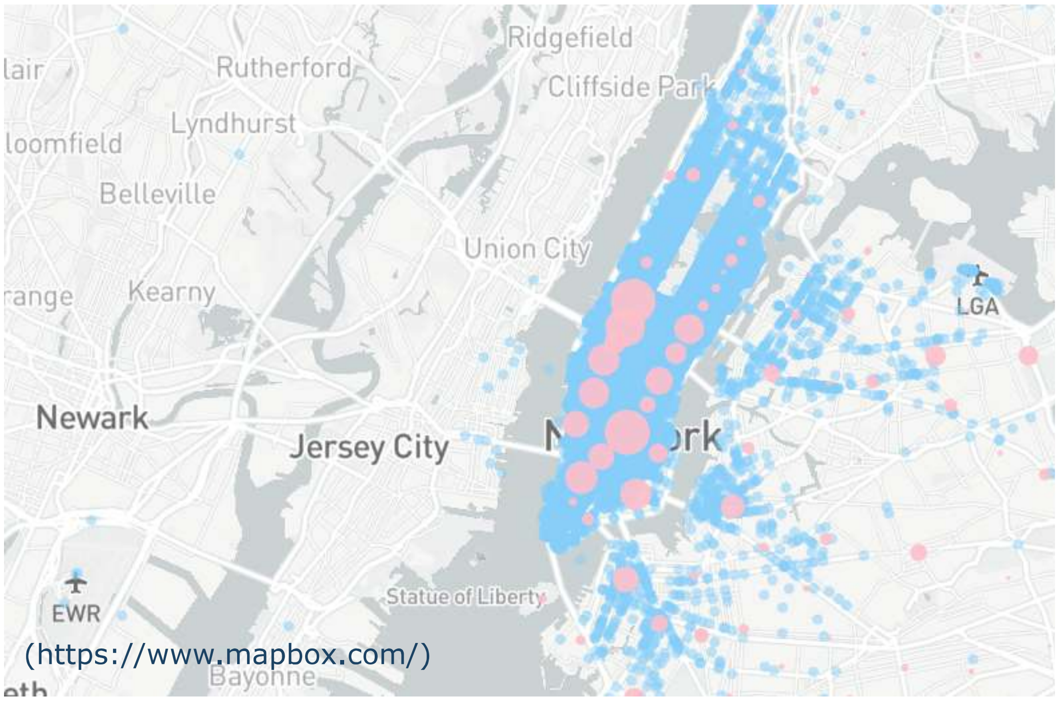
Out[55]:
```
count    39921.000000
mean        40.753251
std          0.023457
min         40.496201
25%         40.739223
50%         40.754421
75%         40.767834
max         40.899853
Name: pickup_latitude, dtype: float64
```

In [56]:
```
mapSample.shape
```

Out[56]:
```
(39921, 15)
```

In [57]:
```
taxi_map_data = go.Scattermapbox(
        lon = mapSample ['pickup_longitude'],
        lat = mapSample ['pickup_latitude'],
        mode = 'markers',
        marker = dict(
                    color = 'lightskyblue',
                    symbol = 'circle',
                    opacity = .5
                ),
    name = "Taxi pickup locations"
)
taxi_map_data2 = go.Scattermapbox(
        lon = restaurantsLocation['LONG'],
        lat = restaurantsLocation['LAT'],
        mode = 'markers',
        text = restaurantsLocation['count'],
        hoverinfo='text',
        marker = dict(
                    color = 'pink',
                    size = restaurantsLocation['count']/30,
                    symbol = 'circle',
                    opacity = .9,

                ),
        name = "Restaurant density by zipcode "
)

taxi_map_layout = go.Layout(
        title = 'Taxi Pickup Locations & Restaurants Density in NYC (Size: The
number of restaurants)',
        mapbox=go.layout.Mapbox(
            accesstoken=mapbox_access_token,
            zoom=1
        )
    )

taxi_map = go.Figure(data=[taxi_map_data,taxi_map_data2], layout=taxi_map_layo
ut)
taxi_map.update_layout(
    hovermode='closest',
    mapbox=go.layout.Mapbox(
        accesstoken=mapbox_access_token,
        bearing=0,
        center=go.layout.mapbox.Center(
            lat=40.753085,
            lon= -73.979750
        ),
        pitch=0,
        zoom=10
    )
)
taxi_map.show()
```

Taxi Pickup Locations & Restaurants Density in NYC (Size: The n

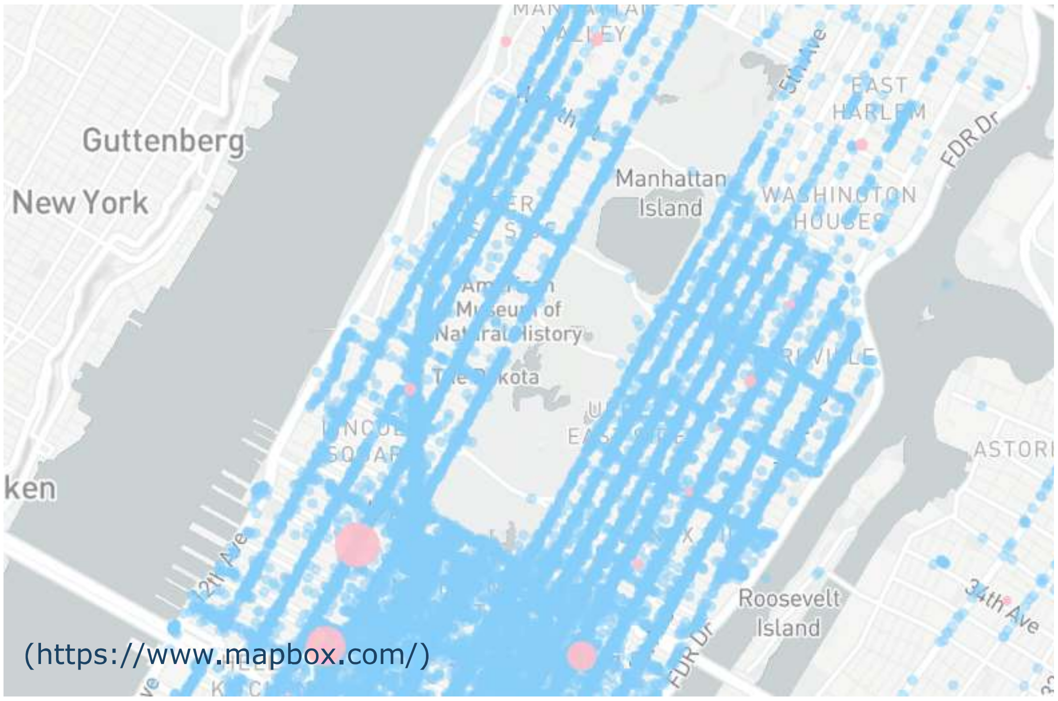


(https://www.mapbox.com/)

Manhattan is the most popular pick-up spots. The pink points in Manhattan are the densest. Also, the Brooklyn area has many spots along the river. Some points are around LGA airports. From the images, we can tell that the southern part of Manhattan has both more restaurants, which is a likely factor that shapes the pattern of the taxis' pick-up locations. Other pick up locations that are not in Manhattan also tend to be within areas that have more restaurants. We can conclude that areas with more restaurants are more popular for people to use taxi service.

In [58]:
```
# Zoom in Manhattan to look at the pattern closer
# The center is the central park in NYC
taxi_map2 = go.Figure(data=[taxi_map_data,taxi_map_data2], layout=taxi_map_lay
out)
taxi_map2.update_layout(
    hovermode='closest',
    mapbox=go.layout.Mapbox(
        accesstoken=mapbox_access_token,
        bearing=0,
        center=go.layout.mapbox.Center(
            lat=40.778424,
            lon=-73.96175
        ),
        pitch=0,
        zoom=12
    )
)
taxi_map2.show()
```

Taxi Pickup Locations & Restaurants Density in NYC (Size: The n



(https://www.mapbox.com/)

Then, we do a zoom-in to see the pattern closer. Within Manhattan, the pick-up spots in the southern part are more intense than the northern part in general. The most popular pick-up spots are located in the several blocks south of the Center Park.

Conclusion

In conclusion, our analysis shows that ‘distance_km’, ‘passenger_count’ has a positive impact on trip duration, and ‘weekend’ has a negative impact on trip duration. It also shows that the peak hours from Monday to Thursday distribute similarly and a little bit different from that for Friday. Peak hours for weekdays and weekends distribute differently. Manhattan is truly a popular pick-up area. Within it, the southern part tends to be more popular than other parts. After conducting this research, our finding can help the pick-up process more efficient for taxi drivers. They will know how to arrange their schedule based on the regression model and peak hours analysis to gain a better profit in an effective way. Also, they can pay more attention to the area with higher demand, so that customer’s waiting time can be minimized, and drivers can make more trips as well.

References

https://www.kaggle.com/mathijs/weather-data-in-new-york-city-2016#weather_data_nyc_centralpark_2016(1).csv (https://www.kaggle.com/mathijs/weather-data-in-new-york-city-2016#weather_data_nyc_centralpark_2016(1).csv)

https://www.kaggle.com/c/nyc-taxi-trip-duration/data (https://www.kaggle.com/c/nyc-taxi-trip-duration/data)

https://www.kaggle.com/new-york-city/nyc-inspections (https://www.kaggle.com/new-york-city/nyc-inspections)

https://gist.githubusercontent.com/erichurst/7882666/raw/5bdc46db47d9515269ab12ed6fb2850377fd869e/US%25
(https://gist.githubusercontent.com/erichurst/7882666/raw/5bdc46db47d9515269ab12ed6fb2850377fd869e/US%2

END