Northeastern University
College of Professional Studies

Final Presentation

NYC Taxi and Limousine Commission (TLC) Green taxi rides

Course: ALY 6110
Course Name: Data Management and Big Data
Professor: Patrick Kanza

Submitted By:
Harsh Samani - 001403994
Jainik Majmudar - 001878085
Sai Srujan Kumar Majeti - 001497341

Academic Term: Winter 2019
Date of Submission: 02/14/2019

Description:

The dataset provides below in the link were collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers authorized under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP). The trip data was not created by the TLC, and TLC makes no representations as to the accuracy of these data.

•Mission statement is to provide access to New Yorkers and Visitor

•Types of taxis – Green / Yellow / For-Hire

•Data to NYC TLC is provided by Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP)

Analysis:

- Importing necessary libraries

```
In [3]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        sns.set_style('whitegrid')
        %matplotlib inline
```

- We have loaded the dataset using Pandas library in Python

```
taxi = pd.read_csv('/Users/jainikmajmudar/Downloads/green_tripdata_2018-06.csv')
```

- Use head function to view first 5 values and to view the type of data under respective columns

```
taxi.head(5)
```

	VendorID	lpep_pickup_datetime	lpep_dropoff_datetime	store_and_fwd_flag	RatecodeID	PULocationID	DOLocationID	passenger_count	trip_distance	fare_a
0	2	2018-06-01 00:33:55	2018-06-01 00:36:13	N	1	66	33	5	0.51	
1	2	2018-06-01 00:40:36	2018-06-01 00:49:46	N	1	25	49	5	1.97	
2	2	2018-06-01 00:57:12	2018-06-01 01:02:58	N	1	61	49	5	1.40	
3	2	2018-06-01 00:10:13	2018-06-01 00:16:27	N	1	49	97	1	1.36	
4	1	2018-06-01 00:32:08	2018-06-01 00:52:06	N	1	75	127	1	7.90	

- Use describe function to get the basic statistical parameters for all the columns present in the dataset.

```
taxi.describe()
```

	VendorID	RatecodeID	PULocationID	DOLocationID	passenger_count	trip_distance	fare_amount	extra	mta_tax	tip_am
count	739373.000000	739373.000000	739373.000000	739373.000000	739373.000000	739373.000000	739373.000000	739373.000000	739373.000000	739373.00
mean	1.838727	1.065254	111.230031	129.260628	1.357878	3.298843	13.906104	0.325185	0.489056	1.02
std	0.367783	0.496413	74.388120	76.823702	1.041089	3.738948	12.779641	0.396634	0.081254	2.04
min	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	-80.000000	-4.500000	-0.500000	-2.00
25%	2.000000	1.000000	51.000000	62.000000	1.000000	1.100000	6.500000	0.000000	0.500000	0.00
50%	2.000000	1.000000	82.000000	129.000000	1.000000	1.990000	10.000000	0.000000	0.500000	0.00
75%	2.000000	1.000000	166.000000	193.000000	1.000000	3.970000	17.000000	0.500000	0.500000	1.70
max	2.000000	6.000000	265.000000	265.000000	9.000000	143.100000	2113.000000	4.500000	0.500000	444.44

- By using is NULL function, we can understand if we have any NULL values populated in any of the columns.

```
taxi.isnull().sum()
```

```
VendorID                      0
lpep_pickup_datetime          0
lpep_dropoff_datetime         0
store_and_fwd_flag            0
RatecodeID                    0
PULocationID                  0
DOLocationID                  0
passenger_count               0
trip_distance                 0
fare_amount                   0
extra                         0
mta_tax                       0
tip_amount                    0
tolls_amount                  0
ehail_fee                739373
improvement_surcharge         0
total_amount                  0
payment_type                  0
trip_type                     0
dtype: int64
```

- To understand the types of Vendors in the dataset, use value_counts function.

```
# Number of records provided by each of the LPEP provider 1 = Creative Mobile Technologies, LLC; 2= VeriFone Inc.
taxi['VendorID'].value_counts()
```

```
2    620132
1    119241
Name: VendorID, dtype: int64
```

- Now drop unnecessary columns which could saturate our analysis.

```
# Dropping the column ehail_fee as the whole column has nan values
taxi.drop(['ehail_fee'], axis=1,inplace = True)
```

- Since the column with datetime is an object datatype which needs to be changed in datetime datatype.

```
# The column with datetime is an object datatype which needs to be changed in datetime datatype
taxi['lpep_pickup_datetime'].dtype
```

```
dtype('O')
```

- Now, changing timestamp to datatime datatype

```
# changing to datatime datatype
taxi['lpep_pickup_datetime'] = pd.to_datetime(taxi['lpep_pickup_datetime'])
taxi['lpep_pickup_datetime'].dtype
```

```
dtype('<M8[ns]')
```

- Creating new columns in the dataframe such as Hour_pickup, Month_pickup and Day of Week_pickup.

```
# splitting the lpep_pickup_datetime to a seperate column inti hours, month aand day of the week.

taxi['Hour_pickup'] = taxi['lpep_pickup_datetime'].apply(lambda time: time.hour)
taxi['Month_pickup'] = taxi['lpep_pickup_datetime'].apply(lambda time: time.month)
taxi['Day of Week_pickup'] = taxi['lpep_pickup_datetime'].apply(lambda time: time.dayofweek)
```

```
# Use the .map() with this dictionary to map the actual string names to the day of the week:
dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
taxi['Day of Week_pickup'] = taxi['Day of Week_pickup'].map(dmap)
taxi.head(5)
```

	VendorID	lpep_pickup_datetime	lpep_dropoff_datetime	RatecodeID	PULocationID	DOLocationID	passenger_count	trip_distance	fare_amount	tip_amount
0	2	2018-06-01 00:33:55	2018-06-01 00:36:13	1	66	33	5	0.51	4.0	0.70
1	2	2018-06-01 00:40:36	2018-06-01 00:49:46	1	25	49	5	1.97	9.0	2.06
2	2	2018-06-01 00:57:12	2018-06-01 01:02:58	1	61	49	5	1.40	6.5	0.00
3	2	2018-06-01 00:10:13	2018-06-01 00:16:27	1	49	97	1	1.36	7.0	0.00
4	1	2018-06-01 00:32:08	2018-06-01 00:52:06	1	75	127	1	7.90	24.0	6.30

```
#Now performing same task for lpep_dropoff_datetime column and making into seperate column hours, month aand day of the
taxi['Hour_drop'] = taxi['lpep_dropoff_datetime'].apply(lambda time: time.hour)
taxi['Month_drop'] = taxi['lpep_dropoff_datetime'].apply(lambda time: time.month)
taxi['Day of Week_drop'] = taxi['lpep_dropoff_datetime'].apply(lambda time: time.dayofweek)
```

```
# Again use of the .map() function with this dictionary to map the actual string names to the day of the week:
dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
taxi['Day of Week_drop'] = taxi['Day of Week_drop'].map(dmap)
taxi.head(5)
```

	VendorID	lpep_pickup_datetime	lpep_dropoff_datetime	RatecodeID	PULocationID	DOLocationID	passenger_count	trip_distance	fare_amount	tip_amount
0	2	2018-06-01 00:33:55	2018-06-01 00:36:13	1	66	33	5	0.51	4.0	0.70
1	2	2018-06-01 00:40:36	2018-06-01 00:49:46	1	25	49	5	1.97	9.0	2.06
2	2	2018-06-01 00:57:12	2018-06-01 01:02:58	1	61	49	5	1.40	6.5	0.00
3	2	2018-06-01 00:10:13	2018-06-01 00:16:27	1	49	97	1	1.36	7.0	0.00
4	1	2018-06-01 00:32:08	2018-06-01 00:52:06	1	75	127	1	7.90	24.0	6.30

- **To Analyze trends**

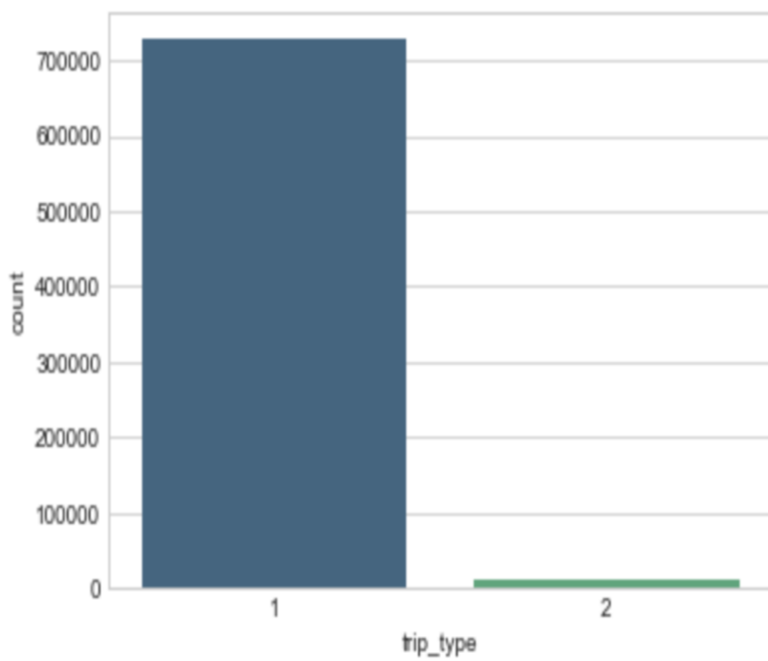
For the analysis, trying to find some similarities, trends among the two type of trips
For the trip_type column 1 -> Street-hail type of trip, 2 -> Dispatch type of trip

```
# 1 = street hail, 2 = dispatch, counting trips by each of the type
taxi['trip_type'].value_counts()
```

```
1    727734
2     11639
Name: trip_type, dtype: int64
```

```
# Visuaaly showing the distribution of both the type of trips
sns.countplot(x='trip_type',data=taxi,palette='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10b7a2be0>
```



```
# At what time in tearms of hours is highest number of taxi rides took by the people.
time_hr1 = taxi['Hour_pickup'].value_counts().head(1)
print('Highest number of taxi rides requested in the particular hour : ', time_hr1)
```

```
Highest number of taxi rides requested in the particular hour :  18    51162
Name: Hour_pickup, dtype: int64
```

```
# At what time in tearms of hours is highest number of Street hail type of trip (taxi rides) took by the people.
time_hr2 = taxi[taxi['trip_type'] == 1]['Hour_pickup'].value_counts().head(1)
print('Highest number of Street hail type of trip in the particular hour : ', time_hr2)
```

```
Highest number of Street hail type of trip in the particular hour :  18    50575
Name: Hour_pickup, dtype: int64
```

```
# At what time in tearms of hours is highest number of dispatch type of trip (taxi rides) took by the people.
time_hr3 = taxi[taxi['trip_type'] == 2]['Hour_pickup'].value_counts().head(1)
print('Highest number of dispatch type of trip in the particular hour : ', time_hr3)
```

```
Highest number of dispatch type of trip in the particular hour :  8    642
Name: Hour_pickup, dtype: int64
```

Which is the most busiest day for taxi rides looking at dispatch type of trip and Street hail type of trip

```
busy_ride1= taxi[taxi['trip_type'] == 1]['Day of Week_drop'].value_counts()
print('The busiest day for Street Hail type ride is on :',busy_ride1)
```

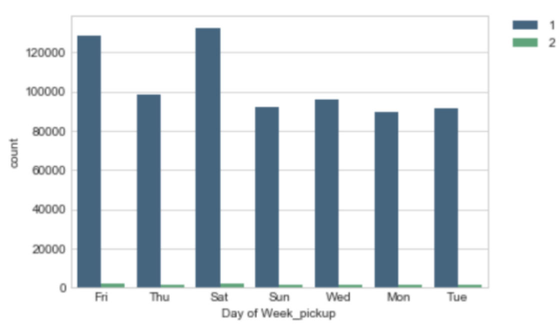
```
The busiest day for Street Hail type ride is on : Sat    131738
Fri    127527
Thu     98399
Wed     95876
Sun     93109
Tue     91600
Mon     89485
Name: Day of Week_drop, dtype: int64
```

```
busy_ride2= taxi[taxi['trip_type'] == 2]['Day of Week_drop'].value_counts()
print('The busiest day for dispatch type ride is on :',busy_ride2)
```

```
The busiest day for dispatch type ride is on : Sat    2311
Fri    2037
Sun    1623
Mon    1491
Thu    1445
Wed    1426
Tue    1306
Name: Day of Week_drop, dtype: int64
```

```
#countplot of the Day of Week column with the hue based off of the trip_type (dispatch & Street Hail) column.
sns.countplot(x='Day of Week_pickup',data=taxi,hue='trip_type',palette='viridis')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

```
<matplotlib.legend.Legend at 0x10a5ac9b0>
```



- Which location is famous for the pickup for both the trip_type

```
d3= taxi[taxi['trip_type'] == 1]['PULocationID'].value_counts().head(1)
print('The famous pickup for Street-hail type ride is on :',d3)
```

```
The most Street-hail type ride is on : 74    47137
Name: PULocationID, dtype: int64
```

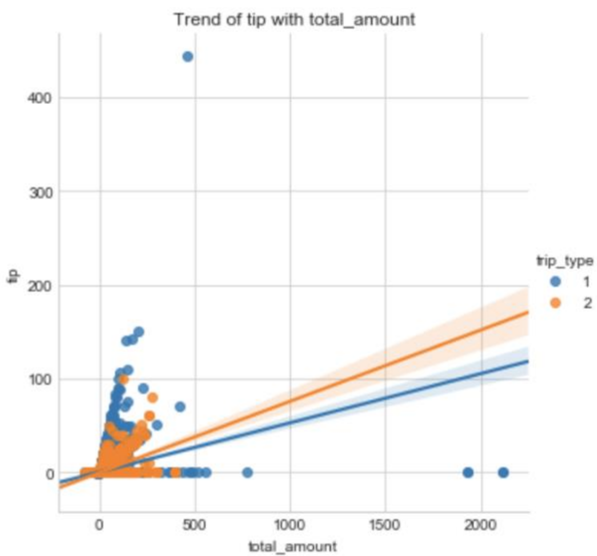
```
d4= taxi[taxi['trip_type'] == 2]['PULocationID'].value_counts().head(1)
print('The famous pickup for dispatch type ride is on :',d4)
```

```
The famous pickup for dispatch type ride is on : 244    554
Name: PULocationID, dtype: int64
```

- Finding the relationship between the total_amount and tip_amount

```
sns.lmplot(x='total_amount', y='tip_amount',hue='trip_type', data=taxi)
plt.xlabel('total_amount')
plt.ylabel('tip')
plt.title('Trend of tip with total_amount')
```

```
Text(0.5,1,'Trend of tip with total_amount')
```



```
taxi.loc[taxi['tip_amount'].idxmax()]
```

```
VendorID                                   2
lpep_pickup_datetime     2018-06-30 16:54:07
lpep_dropoff_datetime    2018-06-30 17:08:24
RatecodeID                                 1
PULocationID                             166
DOLocationID                             246
passenger_count                            1
trip_distance                           4.48
fare_amount                               16
tip_amount                            444.44
total_amount                          461.24
payment_type                               1
trip_type                                  1
Hour_pickup                               16
Month_pickup                               6
Day of Week_pickup                       Sat
Hour_drop                                 17
Month_drop                                 6
Day of Week_drop                         Sat
Name: 727733, dtype: object
```

- The rides which are requested are for how many people and is there any different between number of people take the ride from both trip type.

```
d5 = taxi[taxi['trip_type'] == 1]['passenger_count'].value_counts()
d5
```

```
1    614765
2     55770
5     24801
6     13580
3     12553
4      4776
0      1487
8         1
7         1
Name: passenger_count, dtype: int64
```

```
d6 = taxi[taxi['trip_type'] == 2]['passenger_count'].value_counts()
d6
```

```
1    9238
2    1462
3     454
4     185
5     184
0      85
8      12
7      10
6       7
9       2
Name: passenger_count, dtype: int64
```

```
#by Street hail type
d7 = taxi[taxi['trip_type'] == 1]['payment_type'].value_counts()
d7
```

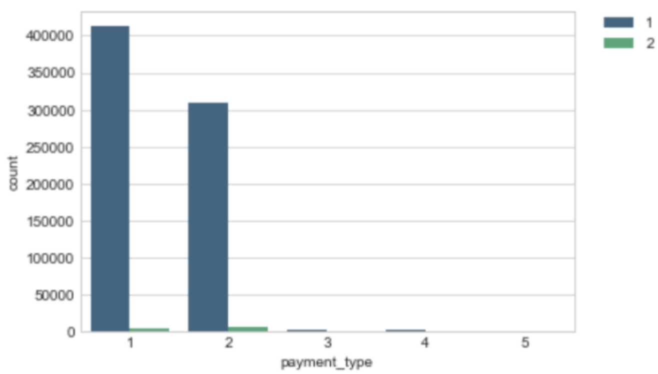
```
1    412479
2    310159
3      3384
4      1687
5        25
Name: payment_type, dtype: int64
```

```
#by dispatch type
d7 = taxi[taxi['trip_type'] == 2]['payment_type'].value_counts()
d7
```

```
2    6773
1    4597
3     158
4     107
5       4
Name: payment_type, dtype: int64
```

```
#countplot of the Day of Week column of the trip_type column using seaborn ploting.
sns.countplot(x='payment_type',data=taxi,hue='trip_type',palette='viridis')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

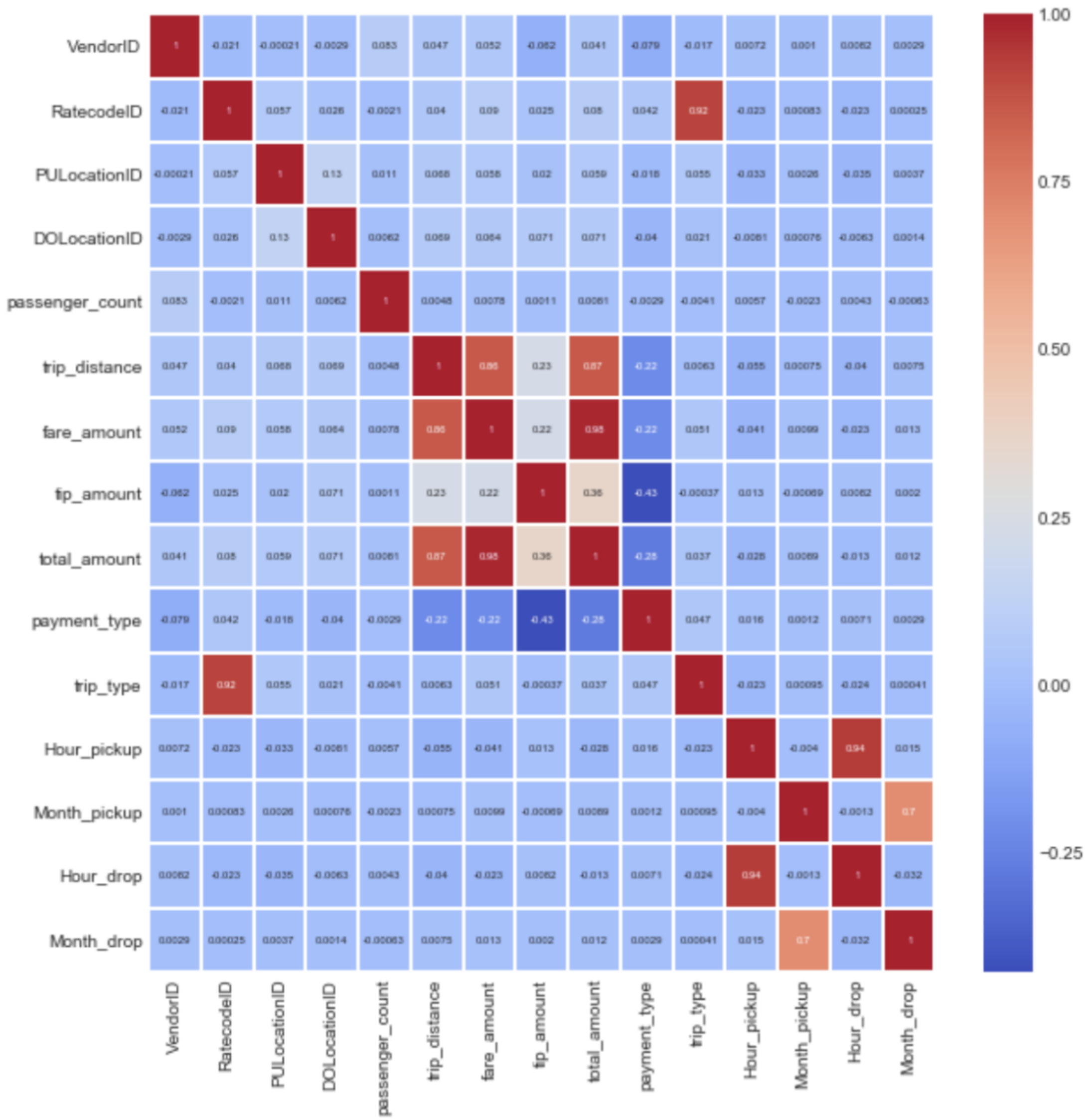
```
<matplotlib.legend.Legend at 0x10dbb57f0>
```



- Predicting Model

```
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(taxi.corr(),cmap='coolwarm',annot=True, linewidths=1, annot_kws={"size":6})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10dc0d748>
```



- Now understanding our features and labels.

```
X = taxi.iloc[:,[0,3,4,5,6,7,8,9,10,11,13,14,16,17]].values
y = taxi.iloc[:,12].values
```

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

```
/Users/jainikmajmudar/anaconda3/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: This
module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes an
d functions are moved. Also note that the interface of the new CV iterators are different from that of this module. T
his module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
X_train
```

```
array([[  2.,   1., 255., ...,   6.,   6.,   6.],
       [  2.,   1., 129., ...,   6.,  16.,   6.],
       [  2.,   1., 185., ...,   6.,  10.,   6.],
       ...,
       [  2.,   1.,  74., ...,   6.,  17.,   6.],
       [  1.,   1.,  43., ...,   6.,  19.,   6.],
       [  2.,   1., 244., ...,   6.,  12.,   6.]])
```

```
X_test
```

```
array([[ 2.,  1., 95., ...,  6., 22.,  6.],
       [ 2.,  1., 41., ...,  6., 10.,  6.],
       [ 2.,  1., 47., ...,  6., 14.,  6.],
       ...,
       [ 2.,  1., 51., ...,  6., 17.,  6.],
       [ 2.,  1., 43., ...,  6., 19.,  6.],
       [ 2.,  1., 61., ...,  6.,  2.,  6.]])
```

```
y_train
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
y_test
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

- Created Classification Algorithm

```
#Creating a classification algorithm - Logistic Regression model, to predict the trip type
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=0, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
y_pred = classifier.predict(X_test)
```

```
y_pred
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

- Created Confusion Matrix using Classification Algorithm

```
# Creating confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
cm
```

```
array([[181806,    129],
       [   256,   2653]])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.997917162580338
```

```
#Creating a classification algorithm - Random forest classifier model, to predict the trip type
from sklearn.ensemble import RandomForestClassifier
classifier1 = RandomForestClassifier(random_state = 0)
classifier1.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

```
y_pred1 = classifier1.predict(X_test)
```

```
y_pred1
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
from sklearn.metrics import confusion_matrix
cm1 = confusion_matrix(y_test, y_pred1)
```

```
cm1
```

```
array([[181869,     66],
       [   243,   2666]])
```

```
accuracy_score(y_test, y_pred1)
```

```
0.9983283200969466
```

```
Y_predicted_trip_type = pd.DataFrame(y_pred1, columns=['predictions_Trip_Type'])
```

Conclusion:

- In this we found, relationship between tips and total amount, we also found the trip type details what is the relation and which trip type is better on which time.
- Predicted which method will be the best for the predicting the trip and trip type.
- We can take both the methods for the consideration while predicting the results as for Logistic Regression we found accuracy to be 99.79% and for random forest classifier it is 99.83%. We just have a difference of 0.04% which doesn't make us fully confident to predict the best method.

References:

http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

http://www.nyc.gov/html/tlc/downloads/pdf/data_dictionary_trip_records_green.pdf