JACOBS UNIVERSITY, BREMEN

DEPARTMENT OF COMPUTER SCIENCES AND SOFTWARE ENGINEERING

Final Project Report

Data Analysis on New York Taxi Dataset

Student Name: Yahya Nasser Ali Salem Mansoor

Matriculation number: 30006969

Contents:

Introduction

The New York yellow taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts recorded in the year 2016 for the month of February. The data used in the attached datasets 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 focus of this project is developing machine learning models that can accurately predict the GoodTip. Good tips are usually a motivator for taxi haling services in densely populated cites, hence it's important for the driver to know the probability of getting a good tip to provide better service. In this project we are going to find the factors which are responsible for good tip, we will use Logistic Regression and Random Forest Classifier and we will evaluate them to find the best model to predict GoodTip.

Background of the Data

Description:

The gool of the project is to predict the GoodTip for a taxi ride in New York City given other features listed below. The Data set is taken from the TCL New York city Taxi data collection we are looking at the time period of 2016 Feb.

The dataset contains the following features:

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. o Creative Mobile Technologies o VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
Pickup_longitude	Longitude where the meter was engaged.
Pickup_latitude	Latitude where the meter was engaged.
RateCodeID	The final rate code in effect at the end of the trip. o Standard rate o JFK o Newark o Nassau or Westchester o Negotiated fare o Group ride
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server. o Y= store and forward trip o N= not a store and forward trip

Dropoff_longitude	Longitude where the meter was disengaged.
Dropoff_ latitude	Latitude where the meter was disengaged.
Payment_type	A numeric code signifying how the passenger paid for the trip. o Credit card o Cash o No charge o Dispute o Unknown o Voided trip
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes. the $0.50 and $1 rush hour and overnight charges.
MTA_tax	0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.
Tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.
GoodTip	Categorical variable indicating an above average tip
Extra	An indicator for additional charges included.
Cash	An indicator whether payment was made by cash or not

Data Preprocessing

Importing the required libraries:

```python
# Importing the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import preprocessing

from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from matplotlib import pyplot as plt
```

Loading The Data:

```python
# Load the dataset.
df = pd.read_csv('/content/exam_data.csv')
df.head()
```

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	store_and_fwd_flag
0	1	2	2016-02-07 16:20:15	2016-02-07 16:29:07	1	1.58	-74.004936	40.740719	1	N
1	2	2	2016-02-19 20:51:20	2016-02-19 21:11:27	6	3.04	-73.973763	40.763351	1	N
2	3	1	2016-02-19 20:53:25	2016-02-19 20:56:17	1	0.60	-73.961571	40.811527	1	N
3	4	1	2016-02-19 20:54:47	2016-02-19 21:06:08	1	2.10	-73.960197	40.770557	1	N
4	5	2	2016-02-19 20:55:31	2016-02-19 21:06:03	1	1.79	-73.986656	40.762379	1	N

5 rows × 23 columns

Comment: we can read the dataset using function read_csv()

Structure and content:

```
[79]  1 # Check the number of rows and columns in the 'dataset'.
      2 df.shape
```

```
(28454, 23)
```

```
 1 # Apply the 'info()' function on the 'df' DataFrame.
 2 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28454 entries, 0 to 28453
Data columns (total 23 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             28454 non-null  int64
 1   VendorID               28454 non-null  int64
 2   tpep_pickup_datetime   28454 non-null  object
 3   tpep_dropoff_datetime  28454 non-null  object
 4   passenger_count        28454 non-null  int64
 5   trip_distance          28454 non-null  float64
 6   pickup_longitude       28454 non-null  float64
 7   pickup_latitude        28454 non-null  float64
 8   RatecodeID             28454 non-null  int64
 9   store_and_fwd_flag     28454 non-null  object
 10  dropoff_longitude      28454 non-null  float64
 11  dropoff_latitude       28454 non-null  float64
 12  payment_type           28454 non-null  int64
 13  fare_amount            28454 non-null  float64
 14  extra                  28454 non-null  float64
 15  mta_tax                28454 non-null  float64
 16  tip_amount             28454 non-null  float64
 17  tolls_amount           28454 non-null  float64
 18  improvement_surcharge  28454 non-null  float64
 19  total_amount           28454 non-null  float64
 20  GoodTip                28454 non-null  bool
 21  Extra                  28454 non-null  bool
 22  Cash                   28454 non-null  bool
dtypes: bool(3), float64(12), int64(5), object(3)
memory usage: 4.4+ MB
```

```
[104]  1 df.describe()
```

	VendorID	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	dropoff_longitude	dropoff_latitude	payment_type	fare_amount	extra	mta_tax	tip_amount	tolls_amount	improvement_
count	28454.000000	28454.000000	28454.000000	28454.000000	28454.000000	28454.000000	28454.000000	28454.000000	28454.000000	28454.000000	28454.000000	28454.000000	28454.000000	28454.000000	28
mean	1.531208	1.647326	2.861006	-72.856299	40.135362	1.043087	-72.926336	40.174826	1.330709	12.349264	0.330762	0.497786	1.777649	0.275885	
std	0.499034	1.306208	3.688742	9.025006	4.971765	0.866217	8.741253	4.815591	0.484680	10.562863	0.443076	0.035253	2.733388	1.323608	
min	1.000000	1.000000	0.000000	-74.272102	0.000000	1.000000	-74.325638	0.000000	1.000000	-52.000000	-1.000000	-0.500000	-2.700000	0.000000	
25%	1.000000	1.000000	1.000000	-73.991783	40.736351	1.000000	-73.991241	40.734728	1.000000	6.500000	0.000000	0.500000	0.000000	0.000000	
50%	2.000000	1.000000	1.650000	-73.981628	40.753462	1.000000	-73.979713	40.753820	1.000000	9.000000	0.000000	0.500000	1.350000	0.000000	
75%	2.000000	2.000000	3.030000	-73.966759	40.768070	1.000000	-73.962448	40.769656	2.000000	14.000000	0.500000	0.500000	2.350000	0.000000	
max	2.000000	6.000000	180.100000	0.000000	40.884964	99.000000	0.000000	41.241875	4.000000	456.000000	4.500000	0.500000	220.000000	31.000000	

Comment: We can see here the shape of the dataset contains 28454 rows × 23 columns and from info we can see we have different data types 3 boolean, 12 float, 5 integer and 3 object. From the description of the dataframe we can see the mean, minimum and maximum values for our numerical features.

Cleaning the data:

In order to get a better understanding of the data, we need to make sure we don't have missing or duplicated values.

```
1 # Check for the missing values in the 'df' DataFrame.
2 df.isnull().sum()
```

```
Unnamed: 0               0
VendorID                 0
tpep_pickup_datetime     0
tpep_dropoff_datetime    0
passenger_count          0
trip_distance            0
pickup_longitude         0
pickup_latitude          0
RatecodeID               0
store_and_fwd_flag       0
dropoff_longitude        0
dropoff_latitude         0
payment_type             0
fare_amount              0
extra                    0
mta_tax                  0
tip_amount               0
tolls_amount             0
improvement_surcharge    0
total_amount             0
GoodTip                  0
Extra                    0
Cash                     0
dtype: int64
```

```
1 df.nunique()
```

```
Unnamed: 0               28454
VendorID                     2
tpep_pickup_datetime     28254
tpep_dropoff_datetime    28271
passenger_count              6
trip_distance             1587
pickup_longitude          9240
pickup_latitude          15492
RatecodeID                   6
store_and_fwd_flag           2
dropoff_longitude        10251
dropoff_latitude         16755
payment_type                 4
fare_amount                196
extra                        6
mta_tax                      3
tip_amount                 691
tolls_amount                31
improvement_surcharge        3
total_amount              1302
GoodTip                      2
Extra                        2
Cash                         2
dtype: int64
```

```
[83]  1 df.duplicated().sum()
```

```
0
```

Observation: we notice that we don't have missing or duplicated values

[84]

```
# Drop the 'Unnamed: 0' columns from the 'df' DataFrame.
df = df.drop(columns=['Unnamed: 0'], axis=1)
```

```
# Get the list of columns present in the 'df' DataFrame after removing the 'Unnamed: 0' columns.
df.columns
```

```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'pickup_longitude',
       'pickup_latitude', 'RatecodeID', 'store_and_fwd_flag',
       'dropoff_longitude', 'dropoff_latitude', 'payment_type', 'fare_amount',
       'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
       'improvement_surcharge', 'total_amount', 'GoodTip', 'Extra', 'Cash'],
      dtype='object')
```

[86]

```
#coveret tpep_pickup_datetime and tpep_dropoff_datetime from object to datetime
df['tpep_pickup_datetime'] = pd.to_datetime(df.tpep_pickup_datetime)
df['tpep_dropoff_datetime'] = pd.to_datetime(df.tpep_dropoff_datetime)
```

[87]

```
##coveret GoodTip, Extra and Cash from boolean to integer

le = preprocessing.LabelEncoder()
le.fit(df["GoodTip"])
le.fit(df["Extra"])
le.fit(df["Cash"])

df["GoodTip"] = le.transform(df["GoodTip"])
df["Extra"] = le.transform(df["Extra"])
df["Cash"] = le.transform(df["Cash"])
```

```
#checking agian the dataset after convert all the values to numerical values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28454 entries, 0 to 28453
Data columns (total 22 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   VendorID               28454 non-null  int64
 1   tpep_pickup_datetime   28454 non-null  datetime64[ns]
 2   tpep_dropoff_datetime  28454 non-null  datetime64[ns]
 3   passenger_count        28454 non-null  int64
 4   trip_distance          28454 non-null  float64
 5   pickup_longitude       28454 non-null  float64
 6   pickup_latitude        28454 non-null  float64
 7   RatecodeID             28454 non-null  int64
 8   store_and_fwd_flag     28454 non-null  object
 9   dropoff_longitude      28454 non-null  float64
 10  dropoff_latitude       28454 non-null  float64
 11  payment_type           28454 non-null  int64
 12  fare_amount            28454 non-null  float64
 13  extra                  28454 non-null  float64
 14  mta_tax                28454 non-null  float64
 15  tip_amount             28454 non-null  float64
 16  tolls_amount           28454 non-null  float64
 17  improvement_surcharge  28454 non-null  float64
 18  total_amount           28454 non-null  float64
 19  GoodTip                28454 non-null  int64
 20  Extra                  28454 non-null  int64
 21  Cash                   28454 non-null  int64
dtypes: datetime64[ns](2), float64(12), int64(7), object(1)
memory usage: 4.8+ MB
```

Comment: We made some modifications to the dataset to get better results while we are modelling. We dropped the Unnecessary column ‘Unnamed’, convert datetime datatype features from object to datetime, also Good Tip, Extra and Cash from Boolean to integer. Because we need to convert the features to numerical datatype.

Imbalanced Data:

As our target 'GoodTip' has two classes, then balanced data would mean 50% observations for each class. Let us calculate the number of observations for each class.

```
[89]  # Print the number of records in each label and their percentage in the 'GoodTip' column
      # Print the number of records below and above avreage tip
      print("Number of records in each label are")
      print(df['GoodTip'].value_counts())

      # Print the percentage of each label
      print("\nPercentage of records in each label are")
      print(df['GoodTip'].value_counts() * 100 / df.shape[0])
```

```
Number of records in each label are
0    17490
1    10964
Name: GoodTip, dtype: int64

Percentage of records in each label are
0    61.467632
1    38.532368
Name: GoodTip, dtype: float64
```

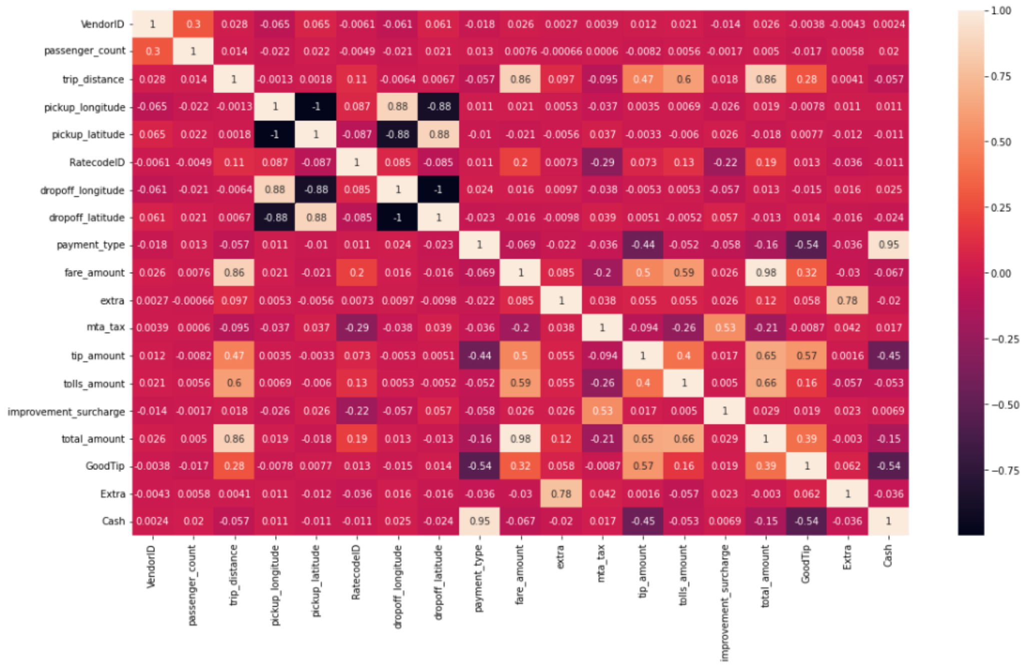
Observation: We can observe that the number of observations for each class is approximately 61% and 39%. This means that our dataset is imbalanced, so we need to do oversampling to make our dataset balanced.

Data Exploration and Data Analysis

Correlation Matrix:

A correlation matrix is simply a table which displays the correlation coefficients for different variables. The matrix depicts the correlation between all the possible pairs of values in a table. It is a powerful tool to summarize a large dataset and to identify and visualize patterns in the given data.

```
plt.figure(figsize= (18, 10))
sns.heatmap(df.corr(), annot= True);
```



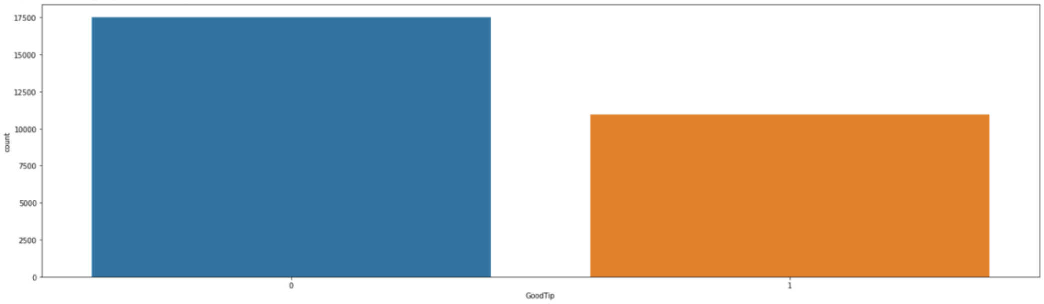
Observation:

- We can observe that the good tip variable negative correlation with the variable payment type and cash
- good tip has positive correlation with variable like trip distance, fare amount, tip amount and total amount
- Hence these variables have positive correlation are important for machine learning models to learn to predict good tip.

Understanding the distribution of GoodTip classes:

```
1 plt.figure(figsize=(25,7))
2 sns.countplot(x=df["GoodTip"])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f01689f6b50>

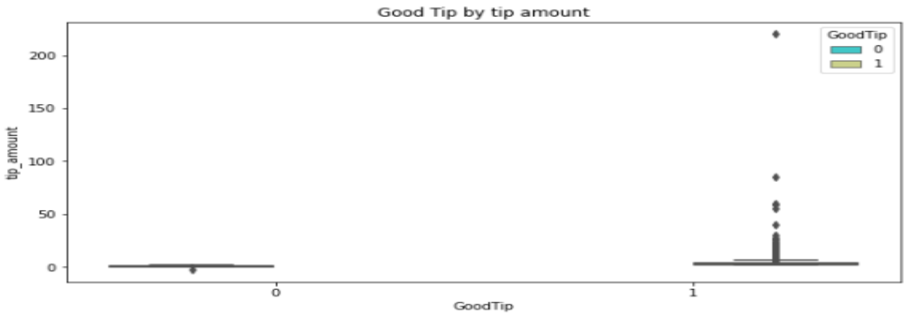


Observation: GoodTip has two classes ‘0’ for tip below the average and ‘1’ for tip above the average. And we can see from the plot the tips below the average more than above the average.

Boxplots for understanding the tip amount:

```
1 plt.figure(figsize=(10,5))
2 sns.boxplot(x='GoodTip',y='tip_amount',data= df, palette='rainbow', hue='GoodTip')
3 plt.title("Good Tip by tip amount")
```

Text(0.5, 1.0, 'Good Tip by tip amount')

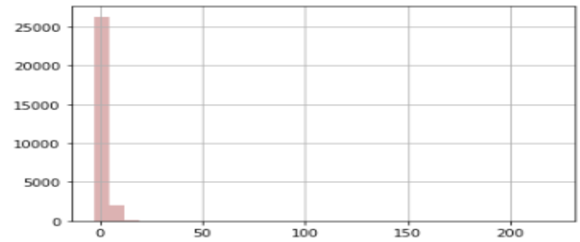


Observation: from this plot we can see the tips above average there is one tip more than 200$ but usually less than 50$.

Plotting the Distribution of Tip Amount:

```
[93] 1 df['tip_amount'].hist(bins=30,color='darkred',alpha=0.3)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f0168a5c880>

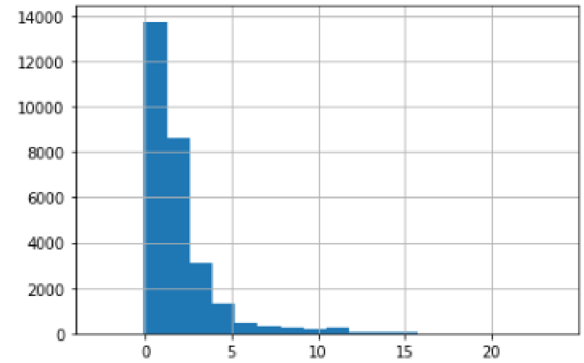


Observation: we can see from this plot most of the riders tip the Taxi driver less than 10$ but still the plot is not clear.

Plotting the Distribution of Tip Amount less than 25$:

```
[17]
print("Number of cases higher than 25 dollars:", len(df[df["tip_amount"]> 25]))

Starting_from_25 = df[df["tip_amount"] < 25]

Starting_from_25["tip_amount"].hist(bins=20)
```

```
Number of cases higher than 20 dollars: 11
<matplotlib.axes._subplots.AxesSubplot at 0x7fdec9114310>
```



Observation: we specify the tip amount to be less than 25$ then we can see the plot clearer than the previous one, so we can see the most tips around 1$ to 5$. And we can see that when we specify the tip amount less than 25$ there are only 11 cases tipping the driver Taxi more than 25$.

Modeling:

Before modeling we need to balance our dataset using OverSampling technique (SMOTE), then we will split the data to training and testing.

SMOTE:

SMOTE is a technique to up-sample the minority classes while avoiding overfitting. It does this by generating new synthetic examples close to the other points (belonging to the minority class) in feature space.

```
[95] # Split the DataFrame into the train and test sets.
X = df.drop(['GoodTip','tpep_pickup_datetime','tpep_dropoff_datetime','store_and_fwd_flag'],axis=1)   # DataFrame consisting of other features
y = df['GoodTip'] # DataFrame containing the GoodTip variable
# Oversampling technique- SMOTE
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)
```

Splitting the Data

```
[96] # Split the DataFrame into the train and test sets such that test set has 30% of the values.
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size = 0.30)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(24486, 18) (10494, 18) (24486,) (10494,)
```

Comment: Y is GoodTip feature and our target. X is all other feature and we dropped date and time and store_and_fwd_flag.

Logistic Regression:

Logistic Regression is a type of classification algorithm which classifies or categorises a given set of data into different class labels. In the context of our dataset, logistic regression will classify the GoodTip either as 1 (above the average) or as 0 (below the average).

Logistic Regression is used to predict the probability of an outcome for an event. It calculates a threshold probability value. If the probability of an outcome is less than the threshold probability, then logistic regression classifies that outcome as 0, otherwise as 1

```
[97]  # Deploy the 'LogisticRegression' model using the 'fit()' function.
      log_reg = LogisticRegression(n_jobs = -1)
      log_reg.fit(X_train, y_train)
      log_reg.score(X_train, y_train)

0.9914645103324349
```

Observation: score is 0.991 which means the model is good

Evaluation of Logistic Regression:
We will evaluate the model using Confusion matrix and Classification report.

```
[98]  y_pred = log_reg.predict(X_test)
      print("Confusion Matrix")
      print(confusion_matrix(y_test, y_pred))
      print("*"*100)
      print("Classification report")
      print(classification_report(y_test, y_pred))

Confusion Matrix
[[5142   73]
 [  18 5261]]
************************************************************
Classification report
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      5215
           1       0.99      1.00      0.99      5279

    accuracy                           0.99     10494
   macro avg       0.99      0.99      0.99     10494
weighted avg       0.99      0.99      0.99     10494
```

Observation: from Classification report the f1-score is high (0.99) for class 1 and class 0 which are true positives and true negative are correctly obtained through Logistic Regression but still we have some False Positives and False Negatives values as we can see from confusion matrix.

Decision Tree:

Decision tree learning is a supervised learning approach used in statistics, data mining and machine learning. In this formalism, a classification or regression decision tree is used as a predictive model to draw conclusions about a set of observations.

```
[99]  clf = DecisionTreeClassifier(random_state=0)
      clf.fit(X_train,y_train)

DecisionTreeClassifier(random_state=0)
```

Evaluation of Decision Tree:

We will evaluate the model using Confusion matrix and Classification report.

```
y_pred = clf.predict(X_test)
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))
print("*"*100)
print("Classification report")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix
[[5215    0]
 [   0 5279]]
*******************************************************
Classification report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5215
           1       1.00      1.00      1.00      5279

    accuracy                           1.00     10494
   macro avg       1.00      1.00      1.00     10494
weighted avg       1.00      1.00      1.00     10494
```

Observation: from Classification report the f1-score is high (1.00) for class 1 and class 0 which are true positives and true negative are correctly obtained through Logistic Regression and we don’t have False Positives and False Negatives values as we can see from confusion matrix**.**

Random Forest Classifier:

Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees

```
[101] # Build the Random Forest Classifier prediction model.
      rf_clf = RandomForestClassifier(n_jobs = -1, n_estimators = 100)
      rf_clf.fit(X_train, y_train)

RandomForestClassifier(n_jobs=-1)
```

Evaluation of Random Forest:

We will evaluate the model using Confusion matrix and Classification report.

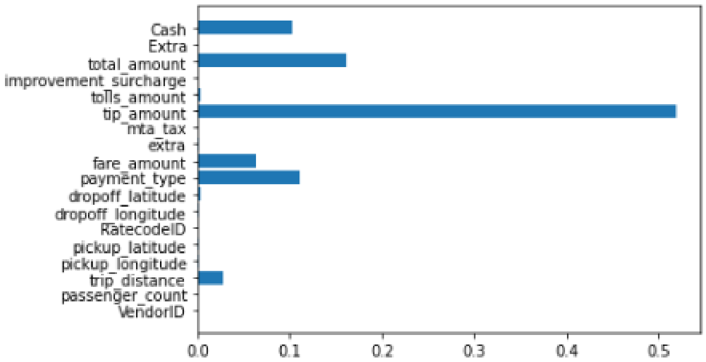
```
[102] #Evaluation of Random Forest Classifier
      rf_y_pred = rf_clf.predict(X_test)
      print("Confusion Matrix")
      print(confusion_matrix(y_test, rf_y_pred))
      print("*"*100)
      print("Classification report")
      print(classification_report(y_test, rf_y_pred))

Confusion Matrix
[[5215    0]
 [   0 5279]]
*******************************************************
Classification report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5215
           1       1.00      1.00      1.00      5279

    accuracy                           1.00     10494
   macro avg       1.00      1.00      1.00     10494
weighted avg       1.00      1.00      1.00     10494
```

Observation: from Classification report the f1-score is high (1.00) for class 1 and class 0 which are true positives and true negative are correctly obtained through Logistic Regression and we don’t have False Positives and False Negatives values as we can see from confusion matrix.

Understanding the most important features of according to Random Forest Classifier:

```
[103]  1 plt.barh(X.columns, rf_clf.feature_importances_)
```

```
<BarContainer object of 18 artists>
```



Observation:

- It can be observed that features like Cash, fare amount, payment type, trip distance and tip amount are very important to classify if the tip awarded was good or not.
- All these features make logical sense in determining the GoodTip.

Conclusions

The aim of this project is to develop machine learning models that can accurately predict the GoodTip. After analyzing the dataset I figure out the dataset was imbalanced, so I used OverSampling technique (SMOTE) to balance the dataset. I used Logistic Regression, Decision Tree and Random Forest Classifier. After evaluating these models using Confusion matrix and Classification report I find out that Decision Tree and Random Forest Classifier are satisfied. According to Random Forest Classifier the most important features are cash, fare amount, payment type, trip distance and tip amount are very important to classify if the tip awarded was good or not which all these features make logical sense in determining the GoodTip.

References

1. Synthetic Minority Over-sampling TEchnique (SMOTE) | by Cory Maklin | Medium
2. Random forest - Wikipedia
3. Decision tree learning - Wikipedia