Title - Regression Algorithms on Yellow Cab Dataset

Group No - 23

Participants Name:

1. AlWasi Khan
2. Vikas Dutta
3. Sandipta Subir Khare

PROBLEM STATEMENT

The goal of this challenge is to predict the fare of a taxi trip given information about the pickup and drop off locations, the pickup date time and number of passengers travelling. We aim to clean the data, visualize the relationship between variables and also figure out new features that are better predictors of taxi fare.

There are six predictor variables and one target variable which are listed as follows: Predictors:

1) Pickup_datetime: timestamp value indicating when the cab ride started.

2) Pickup_longitude: float for longitude coordinate of where the cab ride started.

3) Pickup_latitude: float for latitude coordinate of where the cab ride started.

4) Dropoff_longitude: float for longitude coordinate of where the cab ride ended.

5) Dropoff_latitude: float for latitude coordinate of where the cab ride ended.

6) Passenger_count: an integer indicating the number of passengers in the cab ride.

7) Total_fare: an integer indicating the total amount of the cab ride.

Target: total_amount

Summary Of Solution:

We have the following data in our file, with the given columns.

```
In [4]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9710124 entries, 0 to 9710123
Data columns (total 17 columns):
VendorID                 int64
tpep_pickup_datetime     object
tpep_dropoff_datetime    object
passenger_count          int64
trip_distance            float64
RatecodeID               int64
store_and_fwd_flag       object
PULocationID             int64
DOLocationID             int64
payment_type             int64
fare_amount              float64
extra                    float64
mta_tax                  float64
tip_amount               float64
tolls_amount             float64
improvement_surcharge    float64
total_amount             float64
dtypes: float64(8), int64(6), object(3)
memory usage: 1.2+ GB
```

The different types of variables used in our dataset are as follows with their proper details.

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. **1= Creative Mobile Technologies, LLC; 2= 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. **1= Standard rate** **2=JFK** **3=Newark** **4=Nassau or Westchester** **5=Negotiated fare** **6=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. **Y= store and forward trip** **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. **1= Credit card** **2= Cash** **3= No charge** **4= Dispute** **5= Unknown** **6= 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.

After explaining all the details of the columns

```
In [5]: df.describe()
```

Out[5]:

	VendorID	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type	fare_amount	extra	mta_tax
count	9.710124e+06	9.710124e+06	9.710124e+06	9.710124e+06	9.710124e+06	9.710124e+06	9.710124e+06	9.710124e+06	9.710124e+06	9.710124e+06
mean	1.547079e+00	1.628982e+00	2.813899e+00	1.039581e+00	1.641065e+02	1.617627e+02	1.337541e+00	1.237423e+01	3.234861e-01	4.975229e-01
std	4.977787e-01	1.271994e+00	3.611680e+00	5.059084e-01	6.664998e+01	7.067207e+01	4.913703e-01	2.652315e+02	4.425577e-01	4.881278e-02
min	1.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	-3.500000e+02	-5.520000e+01	-5.000000e-01
25%	1.000000e+00	1.000000e+00	9.500000e-01	1.000000e+00	1.140000e+02	1.070000e+02	1.000000e+00	6.500000e+00	0.000000e+00	5.000000e-01
50%	2.000000e+00	1.000000e+00	1.600000e+00	1.000000e+00	1.620000e+02	1.620000e+02	1.000000e+00	9.000000e+00	0.000000e+00	5.000000e-01
75%	2.000000e+00	2.000000e+00	2.900000e+00	1.000000e+00	2.330000e+02	2.340000e+02	2.000000e+00	1.350000e+01	5.000000e-01	5.000000e-01
max	2.000000e+00	9.000000e+00	2.647100e+02	9.900000e+01	2.650000e+02	2.650000e+02	5.000000e+00	6.259008e+05	5.554000e+01	5.650000e+01

Since the values of the data extracted is so variable and large, we try to reduce the no. of values and required data from our data file.

```
In [6]: # Since the data is very huge we need to filter the data based on our needs
df1 = df[(df['RatecodeID']==1) & (df['total_amount']<75) & (df['payment_type']==1) & ((df['trip_distance']!=0) & df['total_amoun
df1.info()
```

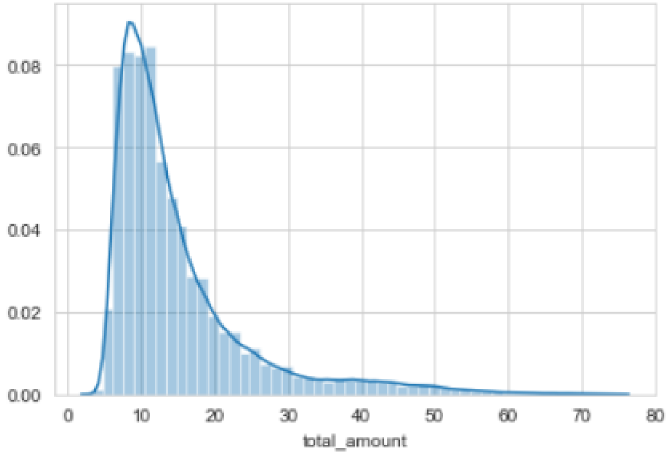
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6311410 entries, 0 to 9710123
Data columns (total 17 columns):
VendorID                 int64
tpep_pickup_datetime     object
tpep_dropoff_datetime    object
passenger_count          int64
trip_distance            float64
RatecodeID               int64
store_and_fwd_flag       object
PULocationID             int64
DOLocationID             int64
payment_type             int64
fare_amount              float64
extra                    float64
mta_tax                  float64
tip_amount               float64
tolls_amount             float64
improvement_surcharge    float64
total_amount             float64
dtypes: float64(8), int64(6), object(3)
memory usage: 866.7+ MB
```

Here we cleanse our data and after cleaning No. of values in our data file which was “9710123” gets reduced to “6311410”, which is the required amount of data for our regression.

We first looked at the distribution of fare amount and found that there were few records where the fare was negative. Since, cost of a trip cannot be negative we removed such instances from the data. Also, fare amount follows long tail distribution. To understand the distribution of fare amount better we take a log transformation after removing the negative fares- this makes the distribution close to normal.

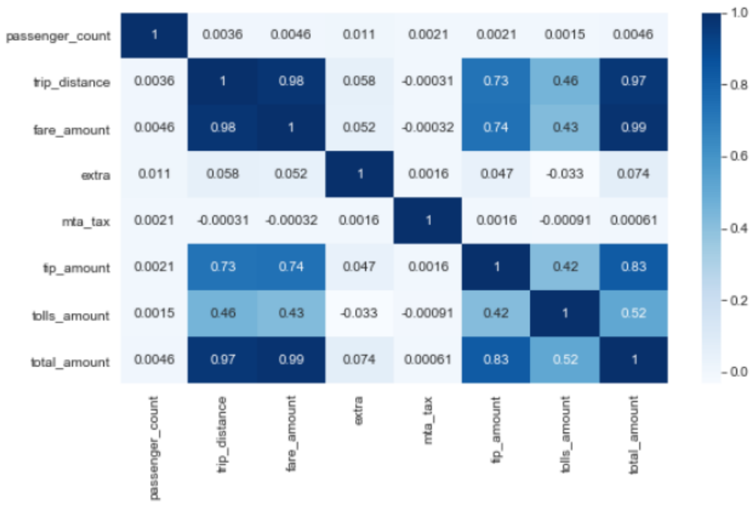
```
In [7]: df1 = df1.drop(df1.index[500000:],0)
```

```
In [8]: sns.distplot(df1['total_amount'])
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x26ee66b4320>
```



Hence, we draw a distribution chart and find that our data follows a normal distribution which is a good a conclusion for the model.

```
In [9]: plt.figure(figsize=(9,5))
        #Removed all the categorical value columns to get the actual data to work on
        sns.heatmap(df1[['passenger_count', 'trip_distance', 'fare_amount', 'extra',
               'mta_tax', 'tip_amount', 'tolls_amount','total_amount']].corr(),annot=True, cmap='Blues')
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x26eb12096a0>
```



We also construct a heat map which gives us correlation of all the columns.

Here we can see that few columns are highly correlated to each other.

Because all the variables are numeric the important features are extracted using the correlation matrix. All the variables are important for predicting the fare_amount since none of the variables have a high correlation factor (considering the threshold as 0.9), so all the variables for model building are kept.

```
In [32]: cdf
```

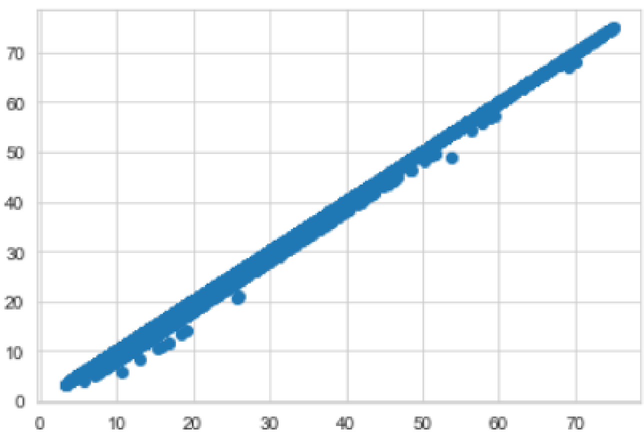
Out[32]:

	Coeff
passenger_count	0.000926
trip_distance	-0.000260
fare_amount	1.000546
extra	1.000426
mta_tax	1.010594
tip_amount	0.999434
tolls_amount	0.998972

```
In [33]: y_pred = lm.predict(x_test)

In [34]: y_pred

Out[34]: array([ 8.16276845,  7.25379628, 34.10940296, ..., 15.95520953,
        20.17005137, 18.3088633 ])
```

After finding correlation between the columns we perform prediction and get the coefficient for all the columns.

```
In [45]: plt.scatter(y_test,y_pred)

Out[45]: <matplotlib.collections.PathCollection at 0x26f04ea8898>
```



The scatter plot shows that the model is very good with very few outliers.

Model Selection

In the early stages of analysis during pre-processing, it is understood that fare_amount is dependent on multiple behaviours. Therefore, it's important to build a model in such a way that it takes in all the required inputs and fits the model in such a way that it gives the most accurate result amongst all the other models. The dependent variable can fall in any of the four categories: Nominal, Ordinal, Interval, and Ratio. Three approaches are taken and compared:

A. Decision Tree

A decision tree is a tree-like graph with nodes representing the place where an attribute is picked and queried; edges represent the answers to the query, and the leaves represent the actual output or class label. Decision trees are nonlinear . Decision Tree algorithms are referred to as Classification and Regression Trees (CART) .

Max Depth: larger the dataset harder to visualize so the maximum branching is taken as five, and Herein, the maxDepth is chosen as 5.

B. Random Forest

Random forest is a tree-based algorithm, which involves building several trees (decision trees), then combining their output to improve the generalization ability of the model. The method of combining trees is known as an ensemble method. The ensemble is a combination of weak learners (individual trees) to produce a strong learner. Random Forest can be used to solve regression and classification problems. In regression problems, the dependent variable is continuous. In classification problems, the dependent variable is categorical.

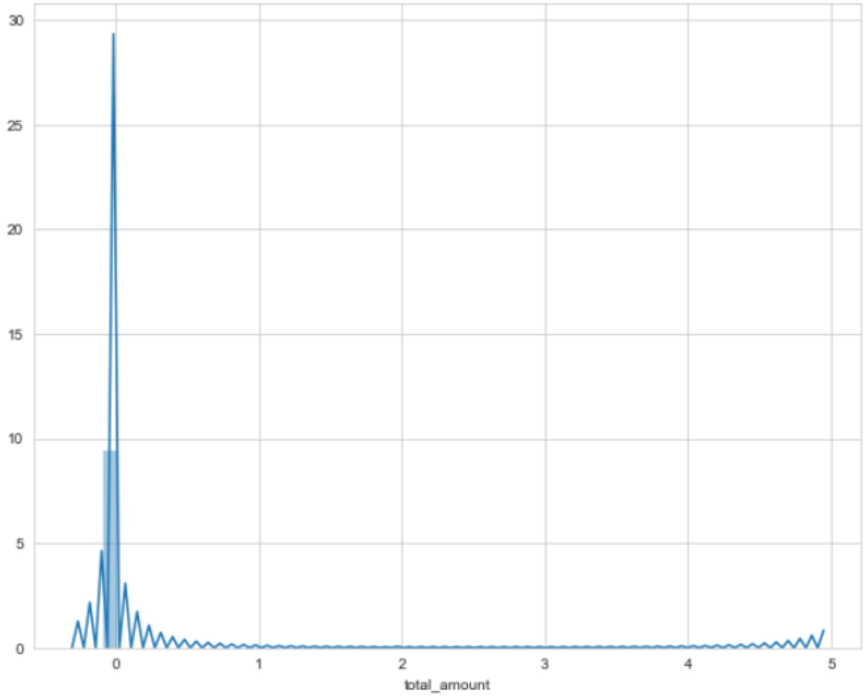
C. SVM REGRESSION

Support Vector Machines are perhaps one of the most popular and talked about machine learning algorithms. The model produced by support-vector classification (as described above) depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.

D. MULTIPLE LINEAR RGRESSION

Multiple linear regression (MLR), also known simply as **multiple regression**, is a statistical technique that uses **several** explanatory variables to predict the outcome of a response variable. **Multiple regression** is an extension of **linear** (OLS) **regression** that uses just one explanatory variable. It is used **when we** want to predict the value of a variable based on the value of two or more other variables. The variable **we** want to predict is called the dependent variable

```
In [46]: plt.figure(figsize=(10,8))
         sns.distplot(y_test-y_pred,bins=50)

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x26f051a5d30>
```



We create a distribution plot and notice that our values are following normal distribution which is a very good model

RANDOM FOREST PREDICTION ERROR

```
In [37]: print("MAE is:\t",metrics.mean_absolute_error(y_test,rfr_pred))
         print("MSE is:\t",metrics.mean_squared_error(y_test, rfr_pred))
         print("RMSE is:",np.sqrt(metrics.mean_squared_error(y_test, rfr_pred)))

MAE is:  0.020019466471557994
MSE is:  0.040103591873415355
RMSE is: 0.20025881222412
```

DECISION TREE PREDICTION ERROR

```
In [30]:  print("MAE is:\t",metrics.mean_absolute_error(y_test,dtree_pred))
          print("MSE is:\t",metrics.mean_squared_error(y_test, dtree_pred))
          print("RMSE is:",np.sqrt(metrics.mean_squared_error(y_test, dtree_pred)))

MAE is:  0.021253174104991474
MSE is:  0.049260350532359894
RMSE is: 0.22194672904181287
```

Multiple LINEAR Regression ERROR

```
In [25]:  print("MAE is:\t",metrics.mean_absolute_error(y_test,y_pred))
          print("MSE is:\t",metrics.mean_squared_error(y_test, y_pred))
          print("RMSE is:",np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

MAE is:  0.01090101412486229
MSE is:  0.01177316900270165
RMSE is: 0.1085042349528425
```

SVM REGRESSION ERROR

```
In [22]:  print("MAE is:\t",metrics.mean_absolute_error(y_test,y_pred))
          print("MSE is:\t",metrics.mean_squared_error(y_test, y_pred))
          print("RMSE is:",np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

MAE is:  6.224111586311541
MSE is:  105.30701201610825
RMSE is: 10.26192048381336
```

MODEL EVALUATION

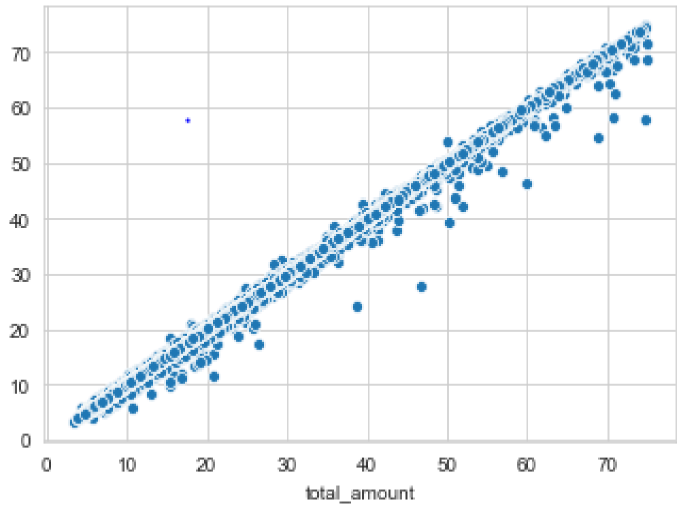
The quality of a regression model is how well its predictions match up against actual values, and Error metrics are used to judge the quality of a model, which enables us to compare regressions against other regressions with varied parameters .

A. Root Mean Squared Error (RMSE) The Root Mean Squared Error (RMSE) and R-Squared are used for dealing with time series forecasting and continuous variables. The RMSE indicates the absolute fit of the model to the data, whereas R-Squared is a relative measure of fit. RMSE must be compared with the dependent variable as RMSE is in the same units as the dependent variable.

Model	RMSE	MAE	MSE
Decision Tree	0.2219467418128	0.21253174104	0.0492603505323
Random Forest	0.20025881222412	0.2001946647155	0.04010359187
Multiple Linear Regression	0.1085042349	0.010901014124	0.01177316

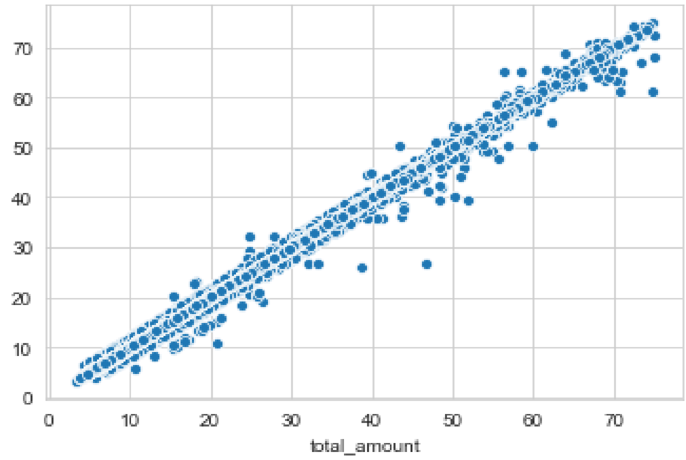
RANDOM FOREST PREDICTION

```
In [37]: sns.scatterplot(x=y_test, y=rfr_pred)

Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x1df0c156860>
```



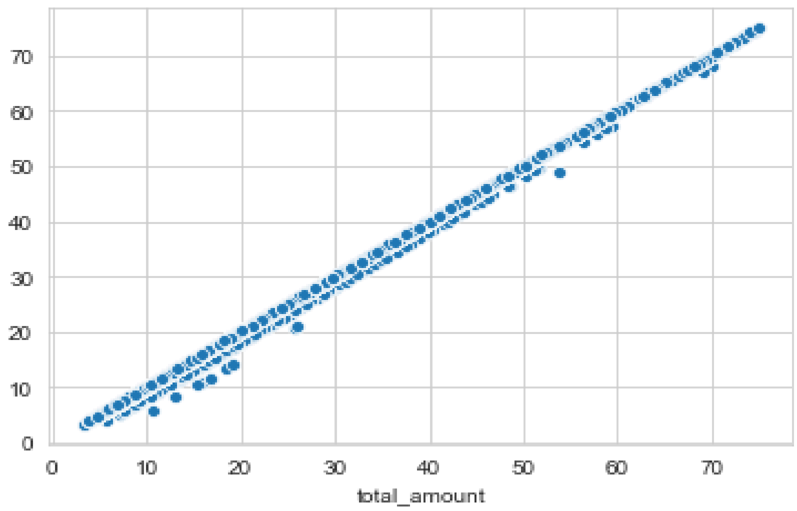
DECISION TREE PREDICTION

```
In [30]: sns.scatterplot(x=y_test, y=dtree_pred)
Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x1df0544bb00>
```



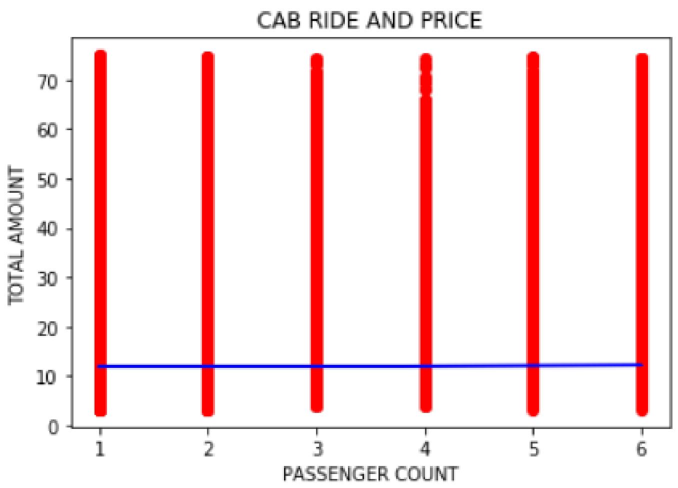
Multiple LINEAR Regression

```
In [22]: sns.scatterplot(x=y_test, y=y_pred)
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1de9d282ac8>
```



Regression Support Vector Machine

In [33]:
```
plt.scatter(x, y, color = 'red')
plt.plot(x_test, regressor.predict(x_test), color = 'blue')
plt.title('CAB RIDE AND PRICE')
plt.xlabel('PASSENGER COUNT')
plt.ylabel('TOTAL AMOUNT')
plt.show()
```



Conclusion:

We started with the data exploration where we got a feeling for the dataset. During this process we used seaborn and matplotlib to do the visualizations.

After performing the algorithms and looking at the metrics we can say that everything performed quiet well. But by considering the metrics for regression we can say that Multiple Linear Regression performed seemingly well.

We finally choose the model proposed by Multiple Linear regression as it has lowest residual errors.