Predictive Analysis on Uber Data and other travel services

SUMMARY

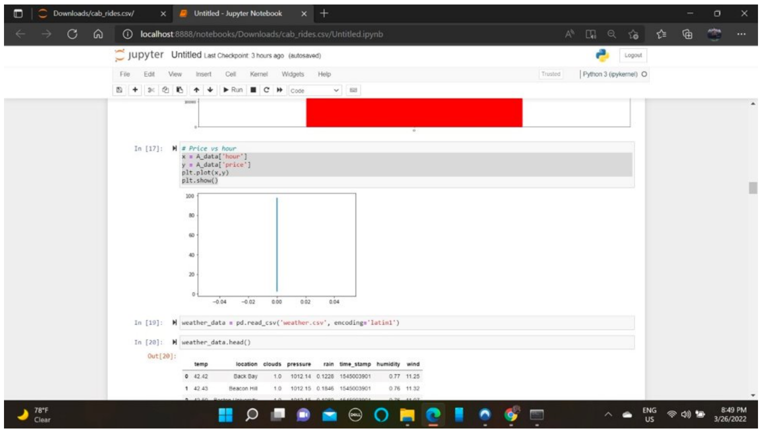
To examine the Uber and Lyft data set in order to establish how reliable their services are to

users, as well as to study their characteristics and other attributes, as well as fare prediction, This project can help uber and other travel services analysis determines which services need to be enhanced based on conditions and location without requiring outside help, and it can also be used to predict fare requirements at specific times and to expand business in a variety of situations, We use the following models (Logistic Regression, Rainforest, and Linear Regression) to analyze and predict the outcome using the data to attain these factors.

GOALS

The main aim of this report is to determine how several factors like Weather, Time, Distance..., affect the price of different travel services and to obtain a better understanding of the statistics and build a strategy for proposing a solution to Uber by creating a system that will assist them in picking the suitable service and enhancing performance by recognizing data trends. Our goal is to outperform current estimates based on analysis.

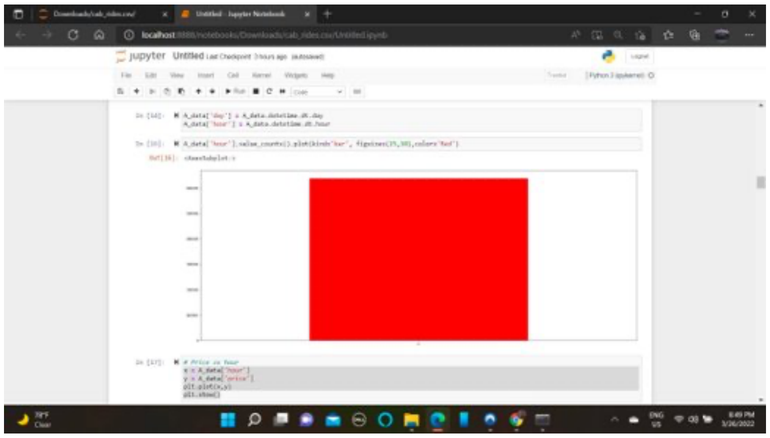
- How is the fare for rides based on hours?



```
In [17]:  # Price vs hour
          x = A_data['hour']
          y = A_data['price']
          plt.plot(x,y)
          plt.show()
```

```
In [19]:  weather_data = pd.read_csv('weather.csv', encoding='latin1')

In [20]:  weather_data.head()

Out[20]:
     temp    location  clouds  pressure    rain  time_stamp  humidity   wind
  0  42.42   Back Bay     1.0   1012.14  0.1228  1545003901      0.77  11.25
  1  42.43  Beacon Hill   1.0   1012.15  0.1846  1545003901      0.76  11.32
```

- What is the peak time for Rides?



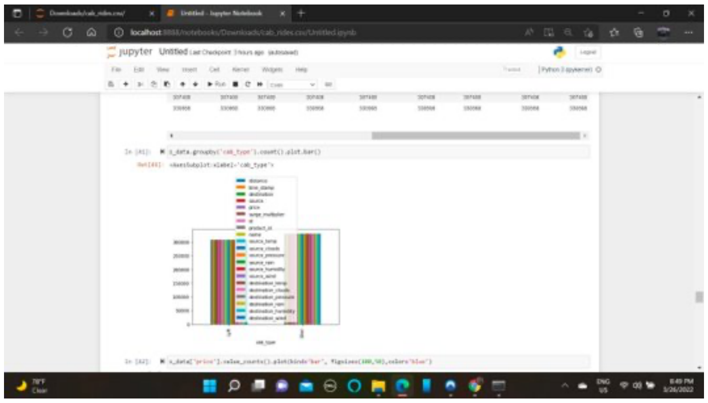
```
In [14]:  A_data['day'] = A_data.datetime.dt.day
          A_data['hour'] = A_data.datetime.dt.hour

In [16]:  A_data['hour'].value_counts().plot(kind='bar', figsize=(15,10),color='Red')

Out[16]:  <AxesSubplot:>
```

```
In [17]:  # Price vs hour
          x = A_data['hour']
          y = A_data['price']
          plt.plot(x,y)
          plt.show()
```

- Which cab service is more providing better service based on all circumstances?
- What type of product is most often selected?

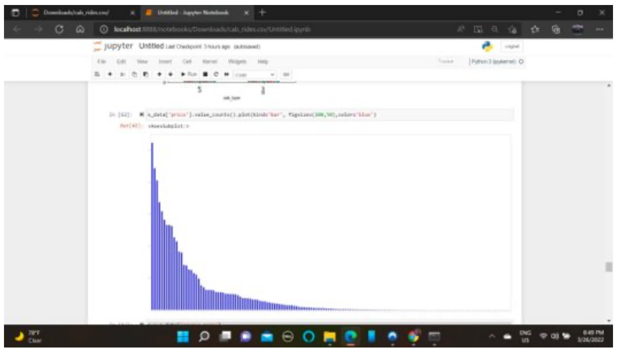
Data set: It's from the Kaggle and some raw data that we converted to a proper data set.

These are some of the data analysis which have been performed on data set



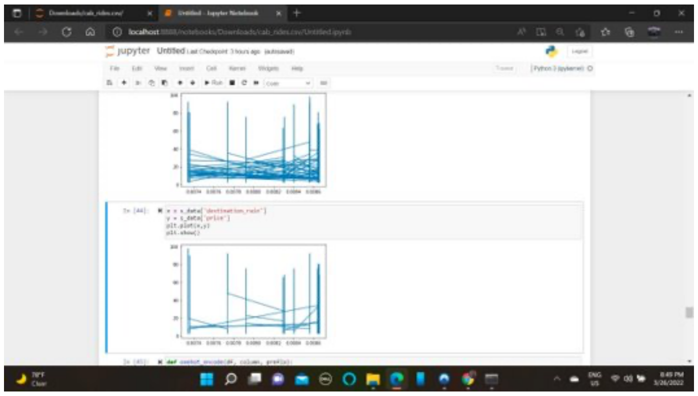
```
In [41]: s_data.groupby('cab_type').count().plot.bar()
Out[41]: <AxesSubplot:xlabel='cab_type'>
```

♦ What is the price in certain locations?



```
In [42]: s_data['price'].value_counts().plot(kind='bar', figsize=(100,50),color='blue')
Out[42]: <AxesSubplot:>
```

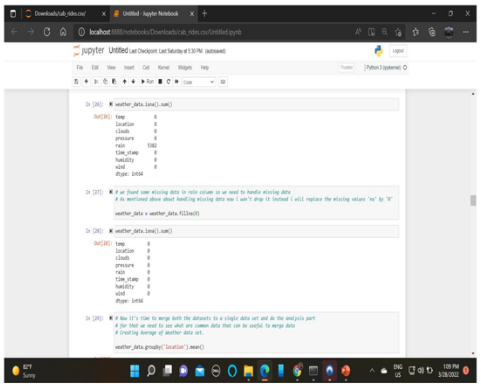
♦ How the price varies for source and destination based on the weather conditions?



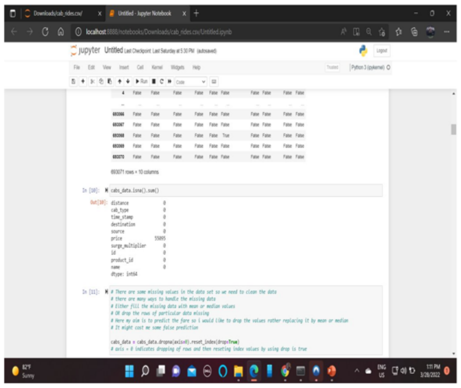
```
In [44]: x = s_data['destination_rain']
         y = s_data['price']
         plt.plot(x,y)
         plt.show()
```

DATA PRE-PROCESSING

Before performing analysis through models, the raw data set which we have considered to be first raw data set is first preprocessed by cleaning and handling the null values.



```
In [26]: weather_data.isna().sum()

Out[26]: temp            0
         location        0
         clouds          0
         pressure        0
         rain         5382
         time_stamp      0
         humidity        0
         wind            0
         dtype: int64

In [27]: # we found some missing data in rain column so we need to handle missing data
         # As mentioned above about handling missing data now i won't drop it instead i will replace the missing values 'na' by '0'

         weather_data = weather_data.fillna(0)

In [28]: weather_data.isna().sum()

Out[28]: temp          0
         location      0
         clouds        0
         pressure      0
         rain          0
         time_stamp    0
         humidity      0
         wind          0
         dtype: int64

In [29]: # Now it's time to merge both the datasets to a single data set and do the analysis part
         # for that we need to see what are common data that can be useful to merge data
         # Creating Average of Weather data set.

         weather_data.groupby('location').mean()
```



```
     4  False  False  False  False  False  False  False  False  False  False
   ...    ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
693066  False  False  False  False  False  False  False  False  False  False
693067  False  False  False  False  False  False  False  False  False  False
693068  False  False  False  False  False   True  False  False  False  False
693069  False  False  False  False  False  False  False  False  False  False
693070  False  False  False  False  False  False  False  False  False  False

693071 rows × 10 columns

In [10]: cabs_data.isna().sum()

Out[10]: distance                0
         cab_type                0
         time_stamp              0
         destination             0
         source                  0
         price               55095
         surge_multiplier        0
         id                      0
         product_id              0
         name                    0
         dtype: int64

In [11]: # There are some missing values in the data set so we need to clean the data
         # there are many ways to handle the missing data
         # Either fill the missing data with mean or median values
         # OR drop the rows of particular data missing
         # Here my aim is to predict the fare so i would like to drop the values rather replacing it by mean or median
         # It might cost me some false prediction

         cabs_data = cabs_data.dropna(axis=0).reset_index(drop=True)
         # axis = 0 indicates dropping of rows and then reseting index values by using drop is true
```

FUNCTIONS OF PRE-PROCESSING

One-hot-encoder returns a vector for each unique value of the categorical column. Each such vector contains only one ‘1’ while all other values in the vector are ‘0’.

Once this is performed, we split the data into testing and training.

One hot encoding: since it is not a categorical data of (two types) we used One hot encoding to transform the categorical values.

METHODS

Linear Regression

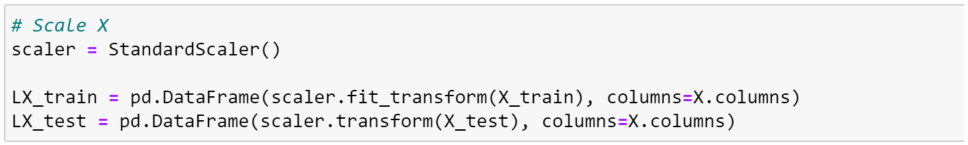
In statistics and machine learning, linear regression is one of the most well-known and well-understood techniques.

Linear regression is the process of identifying a line that best matches the data points on the plot so that we can use it to forecast output values for inputs that aren't present in the data set we have, with the assumption that those outputs will fall on the line.

As mentioned above once the preprocessing is done since the linear regression model only considers 0 and 1 values to predict the data, we have used the Standard scalar function.

Standard Scalar:

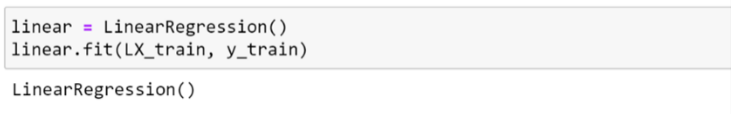
Standard Scalar is used to resize the distribution of values so that the mean of the observed values is 0 and the standard deviation is 1.



```
# Scale X
scaler = StandardScaler()

LX_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X.columns)
LX_test = pd.DataFrame(scaler.transform(X_test), columns=X.columns)
```

After splitting the data set into Train and Test, reshaping for X Train and Y Train is done.

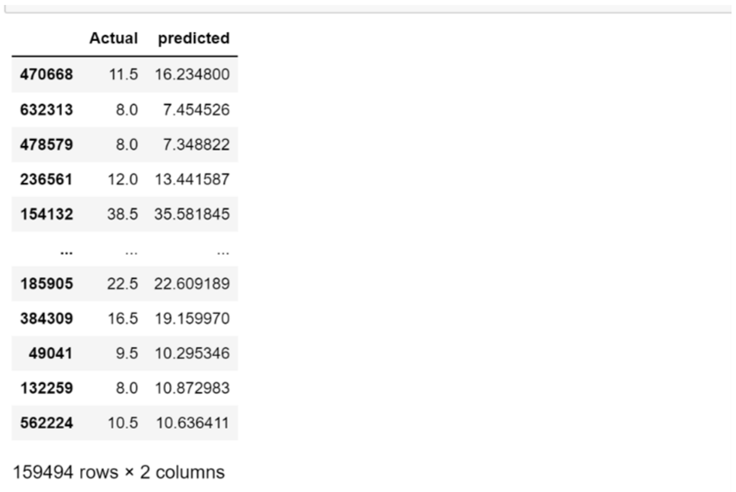
We fit the data to conduct linear regression after reshaping it, and we print the actual and anticipated values.



```
linear = LinearRegression()
linear.fit(LX_train, y_train)
```

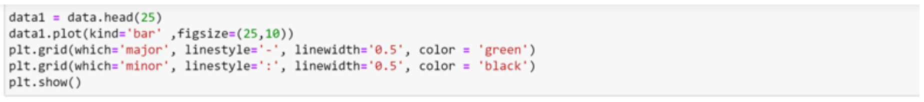
```
LinearRegression()
```

The actual and predicted values, as well as the difference, were shown on the graph.

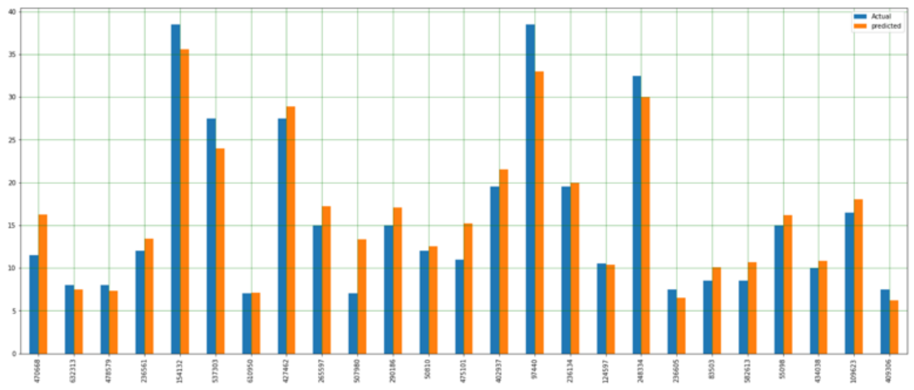


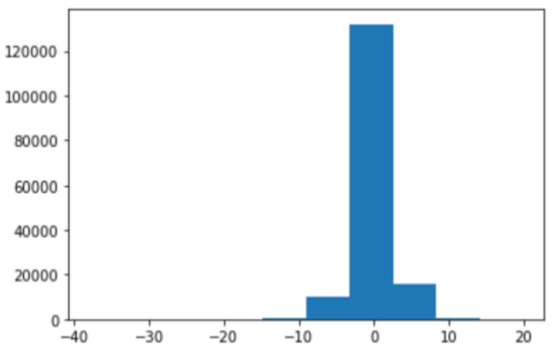
	Actual	predicted
470668	11.5	16.234800
632313	8.0	7.454526
478579	8.0	7.348822
236561	12.0	13.441587
154132	38.5	35.581845
...	...	...
185905	22.5	22.609189
384309	16.5	19.159970
49041	9.5	10.295346
132259	8.0	10.872983
562224	10.5	10.636411

159494 rows × 2 columns



```
data1 = data.head(25)
data1.plot(kind='bar' ,figsize=(25,10))
plt.grid(which='major', linestyle='-', linewidth='0.5', color = 'green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color = 'black')
plt.show()
```





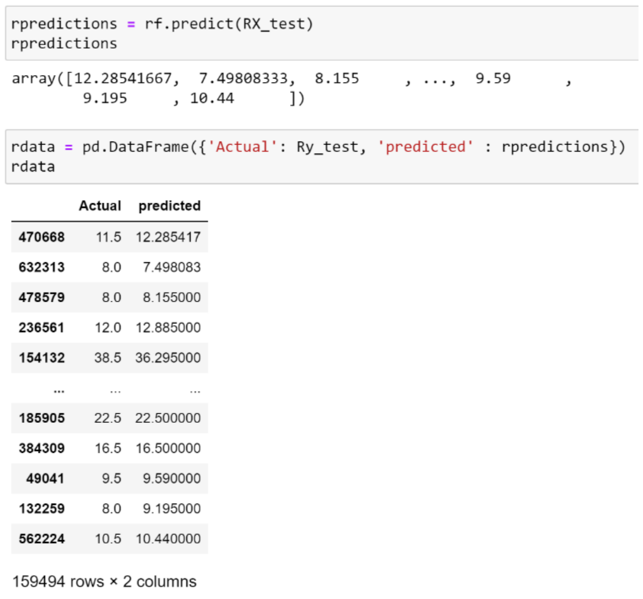
Random Forest

Random forest is a supervised machine learning algorithm that is commonly used to solve classification and regression problems. It creates decision trees from various samples, using the majority vote for classification and the average for regression.

One of the most important characteristics of the Random Forest Algorithm is that it can handle data sets with both continuous and categorical variables, as in regression and classification. For classification difficulties, it produces superior results. Cross-validation ensures a better level of accuracy. This will be handled by a random forest classifier.

Once the preprocessing is done, we split the data set into Train and Test, and reshaping for X_Train and Y_Train is done.

Following the restructuring of the data,



```
rpredictions = rf.predict(RX_test)
rpredictions
```

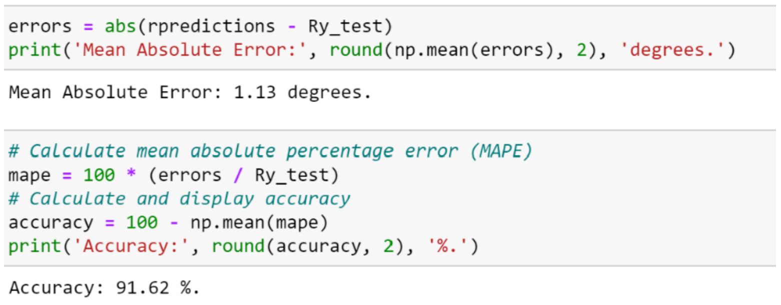
```
array([12.28541667,  7.49808333,  8.155     , ...,  9.59      ,
        9.195     , 10.44      ])
```

```
rdata = pd.DataFrame({'Actual': Ry_test, 'predicted' : rpredictions})
rdata
```

	Actual	predicted
470668	11.5	12.285417
632313	8.0	7.498083
478579	8.0	8.155000
236561	12.0	12.885000
154132	38.5	36.295000
...	...	...
185905	22.5	22.500000
384309	16.5	16.500000
49041	9.5	9.590000
132259	8.0	9.195000
562224	10.5	10.440000

159494 rows × 2 columns

The mean absolute error for the predicted values, as well as the accuracy and differences for predicted values with respect to the test, are then calculated.



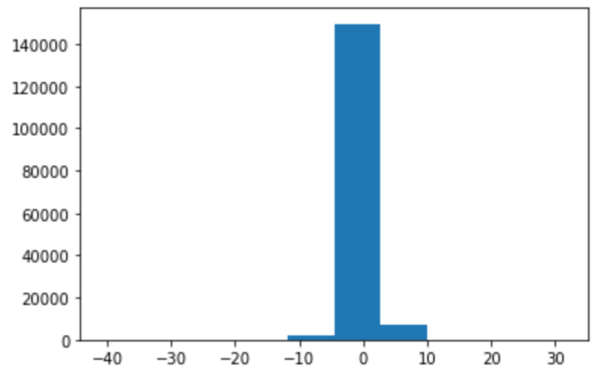
```
errors = abs(rpredictions - Ry_test)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 1.13 degrees.
```

```
# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / Ry_test)
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Accuracy: 91.62 %.
```

We made a graph for the calculated differences.

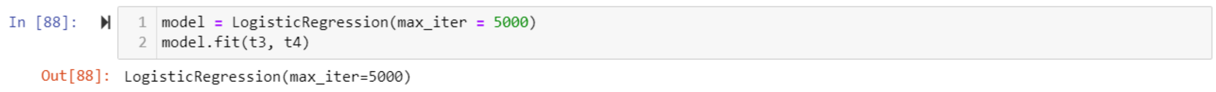


Logistic Regression:

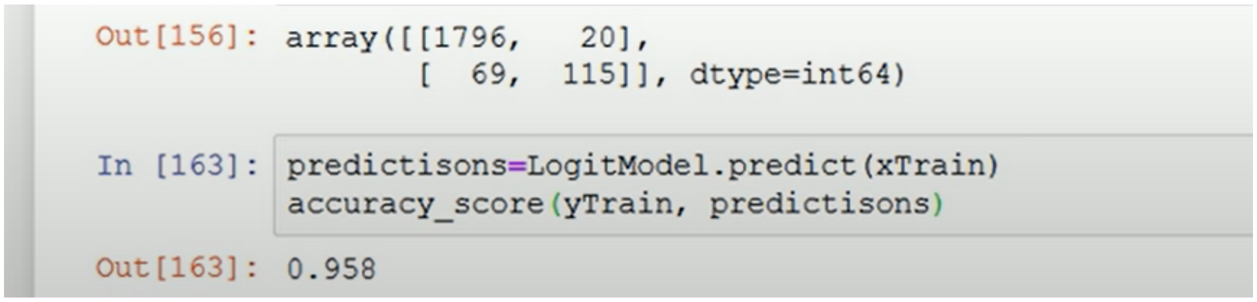
The supervised learning classification algorithm logistic regression is used to predict the probability of a target variable. The target or dependent variable is discrete in nature, which means there are only two possible classes. It is generally used in statistical software to understand the relationship between the dependent variable and one or more independent variables by estimating probabilities using a logistic regression equation.

We type cast the training data into integers after preprocessing the data in logistic regression since the data prediction is only done on integer values.

We fit the data to perform logistic regression, output the actual and predicted ed values, and determine the accuracy score as well as the difference in prediction after reshaping the data.

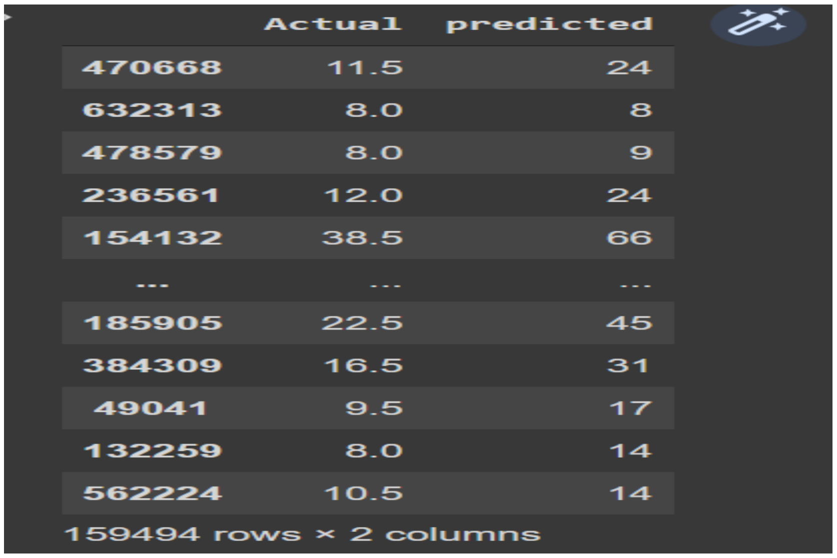


```
In [88]:  model = LogisticRegression(max_iter = 5000)
          model.fit(t3, t4)

Out[88]:  LogisticRegression(max_iter=5000)
```



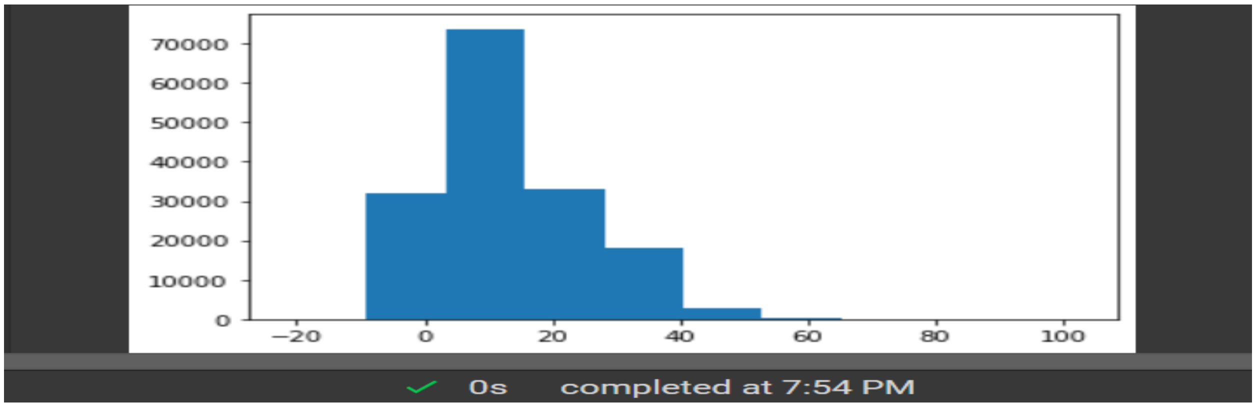
```
Out[156]: array([[1796,   20],
                 [  69,  115]], dtype=int64)

In [163]: predictisons=LogitModel.predict(xTrain)
          accuracy_score(yTrain, predictisons)

Out[163]: 0.958
```

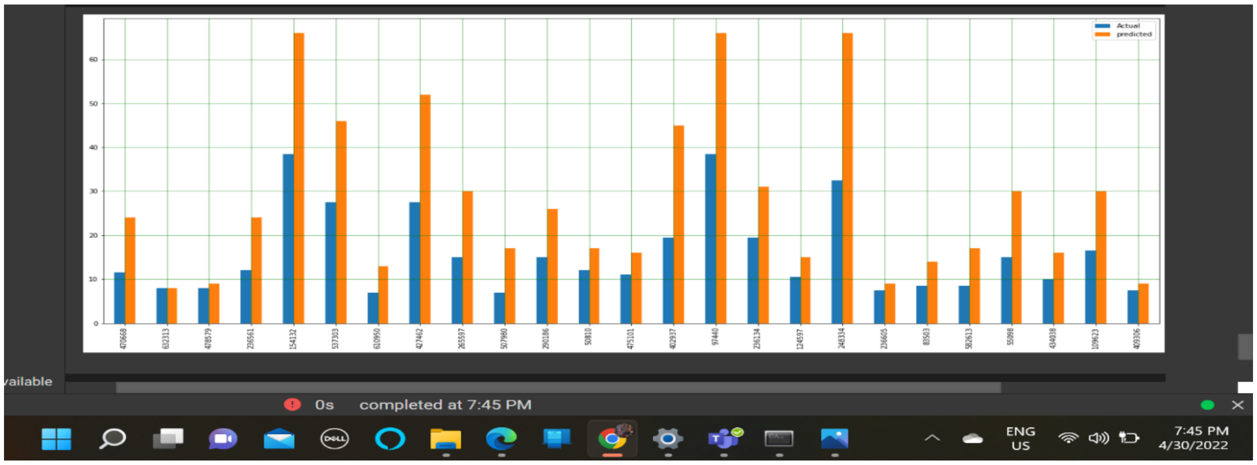
Actual and predicted values have been plotted on the graph.



	Actual	predicted
470668	11.5	24
632313	8.0	8
478579	8.0	9
236561	12.0	24
154132	38.5	66
...	...	...
185905	22.5	45
384309	16.5	31
49041	9.5	17
132259	8.0	14
562224	10.5	14

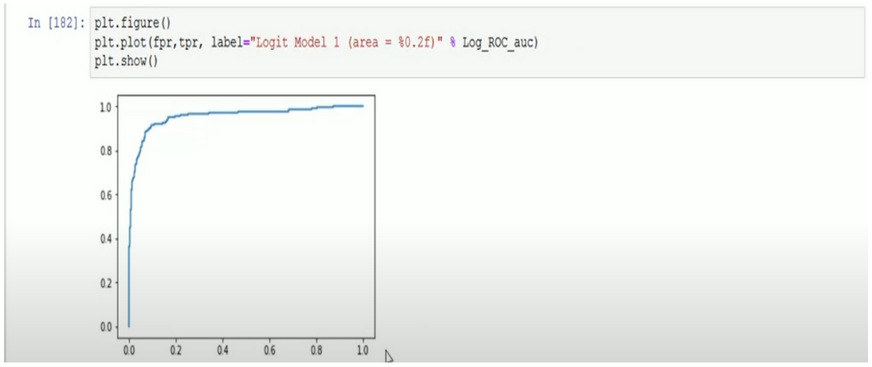
159494 rows × 2 columns



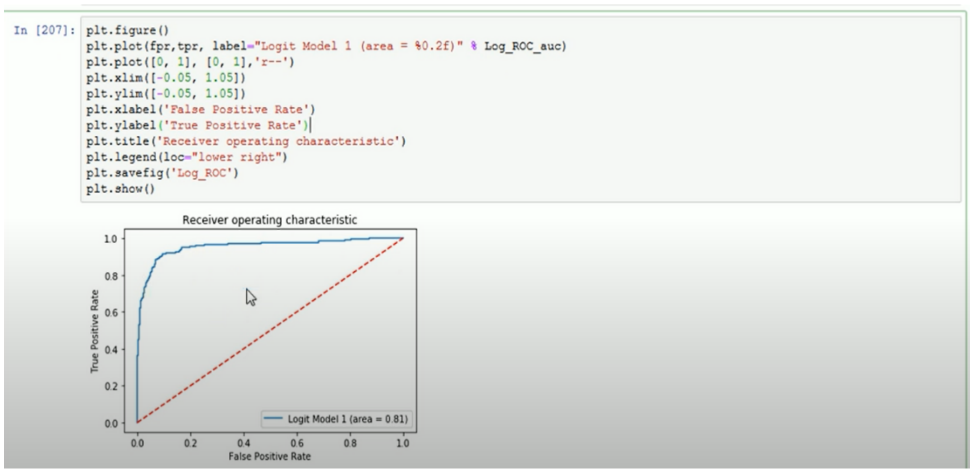


```
0s    completed at 7:45 PM
```

ROC Curve for logistic regression :



```
In [182]: plt.figure()
          plt.plot(fpr,tpr, label="Logit Model 1 (area = %0.2f)" % Log_ROC_auc)
          plt.show()
```



```
In [207]: plt.figure()
          plt.plot(fpr,tpr, label="Logit Model 1 (area = %0.2f)" % Log_ROC_auc)
          plt.plot([0, 1], [0, 1],'r--')
          plt.xlim([-0.05, 1.05])
          plt.ylim([-0.05, 1.05])
          plt.xlabel('False Positive Rate')
          plt.ylabel('True Positive Rate')
          plt.title('Receiver operating characteristic')
          plt.legend(loc="lower right")
          plt.savefig('Log_ROC')
          plt.show()
```

References

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.hl

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

https://www.statology.org/plot-roc-curve-python/

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html