NYC Taxi Trip Time Prediction

Supervised Machine Learning Regression Model

LAKSHMI KEERTHANA
TITO VARGHESE

M/L Pipeline / Points to Discuss

- Problem Statement
- Data Summary
- Data Cleaning
- Feature Engineering
- Exploratory Data Analysis
- Feature Selection
- Model Building
- Comparing Different Models
- Evaluation Metrics
- HyperParameter Tuning
- Conclusion
- Challenges

Problem Statement

To clean,extract,analyse and to build a regression model based on the 2016 NYC yellow cab data set given to us-

The data was originally published by the NYC Taxi and Limousine Commission (TLC). The data was sampled and cleaned for the purposes of this project. Based on individual trip attributes, you should predict the duration of each trip in the test set.

NYC Taxi Data.csv - the training set (contains 1458644 trip records/rows)

NYC Taxi Data.csv - the training set (contains 11 features/columns)

Independent Features

```
'id'                 'Pickup_longitude'
'vendor_id'          'pickup_latitude'
'pickup_datetime'    'dropoff_longitude'
'dropoff_datetime'   'dropoff_latitude'
'passenger_count'    'store_and_fwd_flag'
```

Target Variable/Feature

```
‘trip_duration’
(It’s a continuous
variable hence we will
use a regression model)
```

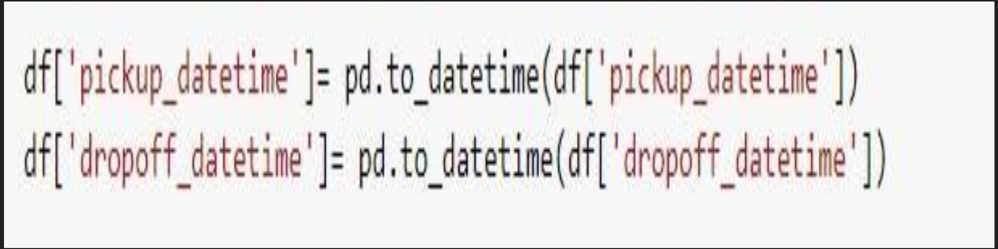
Data Summary

The dataset info gives us some crucial insights into our Features data type and Non Null Count:

```
RangeIndex: 1458644 entries, 0 to 1458643

Data columns (total 11 columns):

 #   Column              Non-Null Count    Dtype
---  ------              --------------    -----
 0   id                  1458644 non-null  object
 1   vendor_id           1458644 non-null  int64
 2   pickup_datetime     1458644 non-null  object
 3   dropoff_datetime    1458644 non-null  object
 4   passenger_count     1458644 non-null  int64
 5   pickup_longitude    1458644 non-null  float64
 6   pickup_latitude     1458644 non-null  float64
 7   dropoff_longitude   1458644 non-null  float64
 8   dropoff_latitude    1458644 non-null  float64
 9   store_and_fwd_flag  1458644 non-null  object
 10  trip_duration       1458644 non-null  int64

dtypes: float64(4), int64(3), object(4)
```

Converted the datatype of pickup_date_time and dropoff_date_time to datetime datatype

```
df['pickup_datetime']=
pd.to_datetime(df['pickup_datetime'])
df['dropoff_datetime']=
pd.to_datetime(df['dropoff_datetime'])
```



```
df['pickup_datetime']= pd.to_datetime(df['pickup_datetime'])
df['dropoff_datetime']= pd.to_datetime(df['dropoff_datetime'])
```

Data Cleaning

Data Cleaning is a crucial step before EDA as it will remove the ambiguous data that can affect the outcome of Model Accuracy.

While cleaning data we performed the following steps:

1) Handling missing values- Since the data given to us was sampled and cleaned for the purposes of this project,it saved lot of our time.

2) Handling duplicate rows - We haven't came across any duplicate records in this data set.

3) Handling Outliers - There were few outliers were found in distance,trip_duration ,trip_speed and Pickup latitude and longitude,we removed it successfully by keeping all the records within np.percentile 99.8 value and np.percentile 1 values.

4) Handling Irrelevant records - The passenger count values 0 and more than 6 were removed from the records.

The final shape of the dataset after cleaning the records (1373771,27)

Feature Engineering

New Features Derived

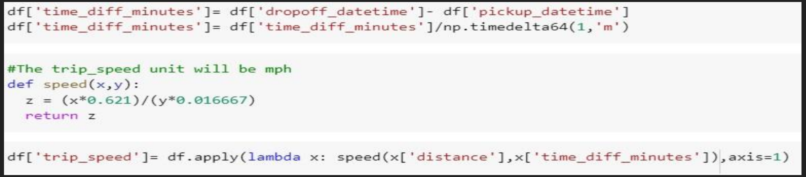
We have derived few features from pickup/dropoff datetime columns after converting its datatype to datetime.

Few new features derived are-

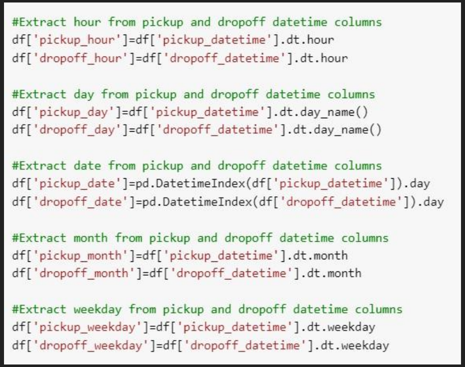
pickup/dropoff month,day,weekday,hour,timezone and date.

We have derived 2 new feature (distance and trip_direction) using the geographical latitude and longitude features in our dataset.

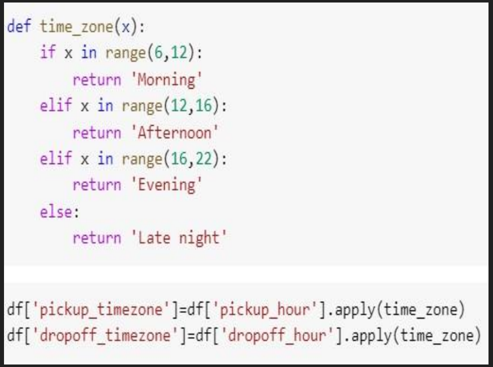
We created another new features trip_speed using trip_distance(km) and trip_duration(seconds). We have used 0.621 to convert the trip distance to miles and 0.00278 to convert seconds to hour.



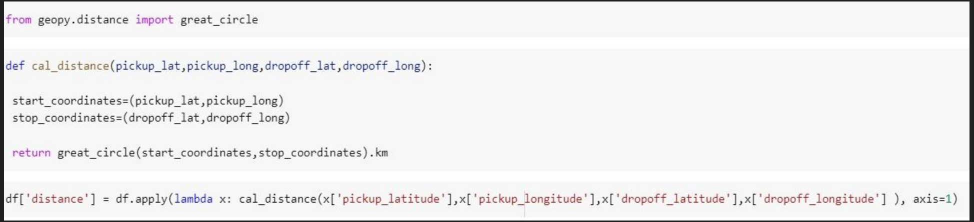
```
df['time_diff_minutes']= df['dropoff_datetime']- df['pickup_datetime']
df['time_diff_minutes']= df['time_diff_minutes']/np.timedelta64(1,'m')

#The trip_speed unit will be mph
def speed(x,y):
  z = (x*0.621)/(y*0.016667)
  return z

df['trip_speed']= df.apply(lambda x: speed(x['distance'],x['time_diff_minutes']),axis=1)
```



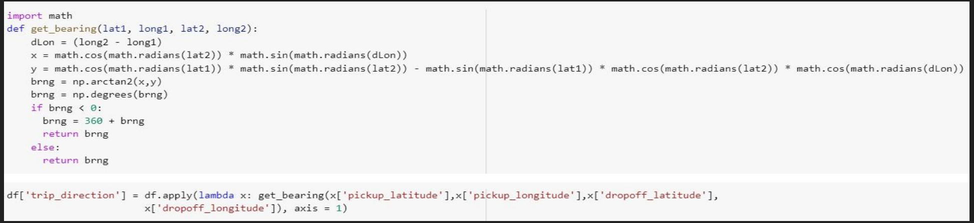
```
#Extract hour from pickup and dropoff datetime columns
df['pickup_hour']=df['pickup_datetime'].dt.hour
df['dropoff_hour']=df['dropoff_datetime'].dt.hour

#Extract day from pickup and dropoff datetime columns
df['pickup_day']=df['pickup_datetime'].dt.day_name()
df['dropoff_day']=df['dropoff_datetime'].dt.day_name()

#Extract date from pickup and dropoff datetime columns
df['pickup_date']=pd.DatetimeIndex(df['pickup_datetime']).day
df['dropoff_date']=pd.DatetimeIndex(df['dropoff_datetime']).day

#Extract month from pickup and dropoff datetime columns
df['pickup_month']=df['pickup_datetime'].dt.month
df['dropoff_month']=df['dropoff_datetime'].dt.month

#Extract weekday from pickup and dropoff datetime columns
df['pickup_weekday']=df['pickup_datetime'].dt.weekday
df['dropoff_weekday']=df['dropoff_datetime'].dt.weekday
```



```
def time_zone(x):
    if x in range(6,12):
        return 'Morning'
    elif x in range(12,16):
        return 'Afternoon'
    elif x in range(16,22):
        return 'Evening'
    else:
        return 'Late night'

df['pickup_timezone']=df['pickup_hour'].apply(time_zone)
df['dropoff_timezone']=df['dropoff_hour'].apply(time_zone)
```



```
from geopy.distance import great_circle

def cal_distance(pickup_lat,pickup_long,dropoff_lat,dropoff_long):

 start_coordinates=(pickup_lat,pickup_long)
 stop_coordinates=(dropoff_lat,dropoff_long)

 return great_circle(start_coordinates,stop_coordinates).km

df['distance'] = df.apply(lambda x: cal_distance(x['pickup_latitude'],x['pickup_longitude'],x['dropoff_latitude'],x['dropoff_longitude'] ), axis=1)
```



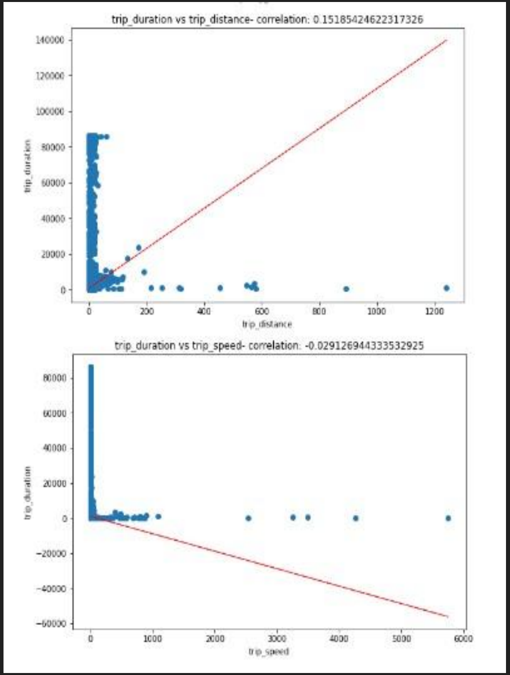
```
import math
def get_bearing(lat1, long1, lat2, long2):
    dLon = (long2 - long1)
    x = math.cos(math.radians(lat2)) * math.sin(math.radians(dLon))
    y = math.cos(math.radians(lat1)) * math.sin(math.radians(lat2)) - math.sin(math.radians(lat1)) * math.cos(math.radians(lat2)) * math.cos(math.radians(dLon))
    brng = np.arctan2(x,y)
    brng = np.degrees(brng)
    if brng < 0:
      brng = 360 + brng
      return brng
    else:
      return brng

df['trip_direction'] = df.apply(lambda x: get_bearing(x['pickup_latitude'],x['pickup_longitude'],x['dropoff_latitude'],
                       x['dropoff_longitude']), axis = 1)
```

Feature Engineering

Checking for Linearity in Independant and Target Feature

The distance shows a positive linear relationship with trip_duration

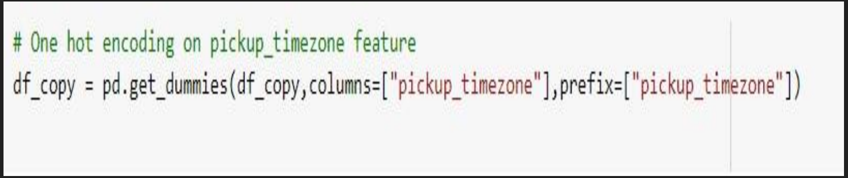
On the Other Hand, the trip_speed shows a negative linear relationship with trip_duration



Feature Encoding

Encoding leads to a better model and most algorithms cannot handle the categorical variables unless they are converted into a numerical value.

In this project we have used hot encoding techniques.

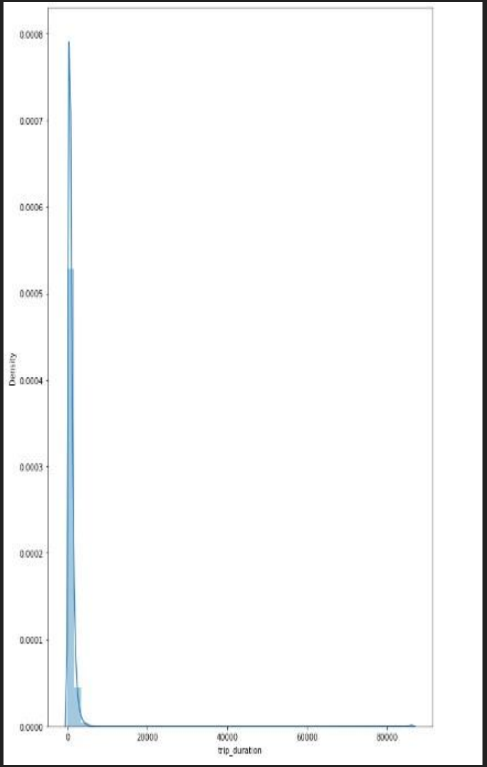


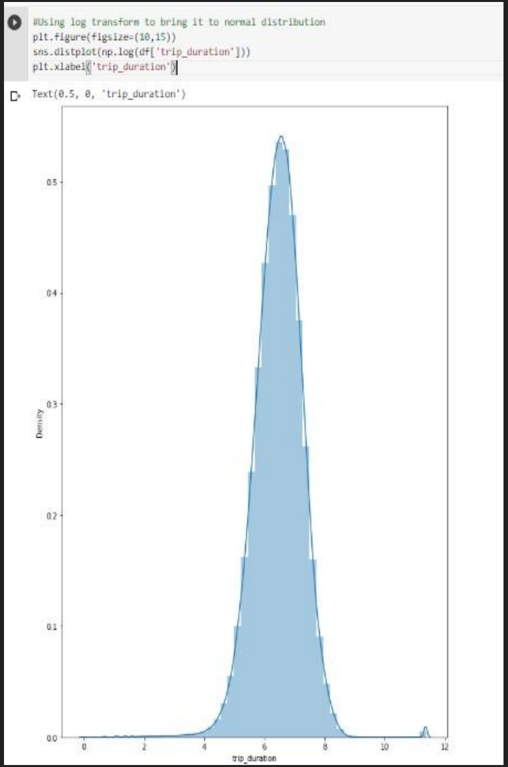
```
# One hot encoding on pickup_timezone feature
df_copy = pd.get_dummies(df_copy,columns=["pickup_timezone"],prefix=["pickup_timezone"])
```

Feature Engineering

Handling Skewness in Features Distribution

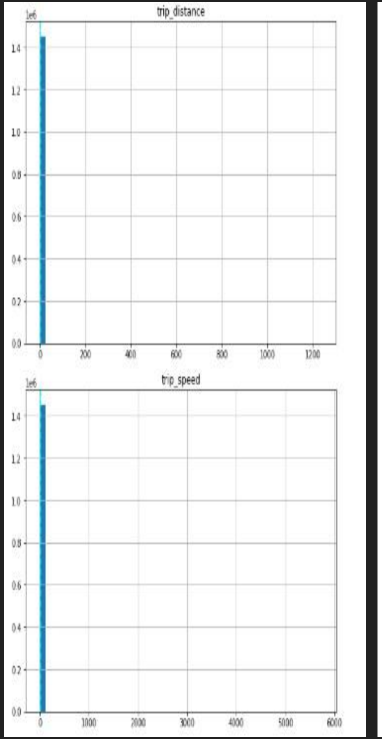
We have used log transform to handle the skewness on the target variable ,distance features

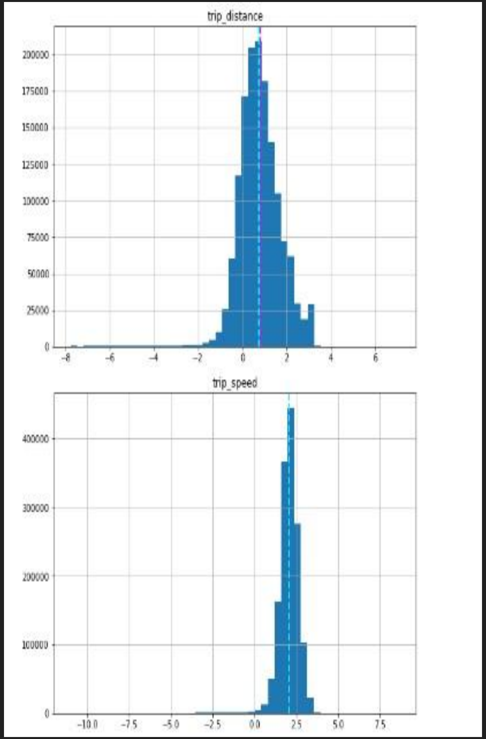




```
#Using log transform to bring it to normal distribution
plt.figure(figsize=(10,15))
sns.distplot(np.log(df['trip_duration']))
plt.xlabel('trip_duration')
```

```
Text(0.5, 0, 'trip_duration')
```

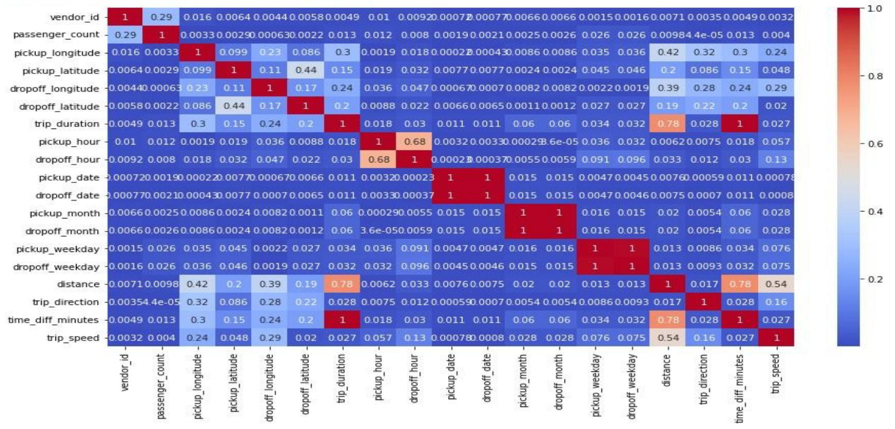


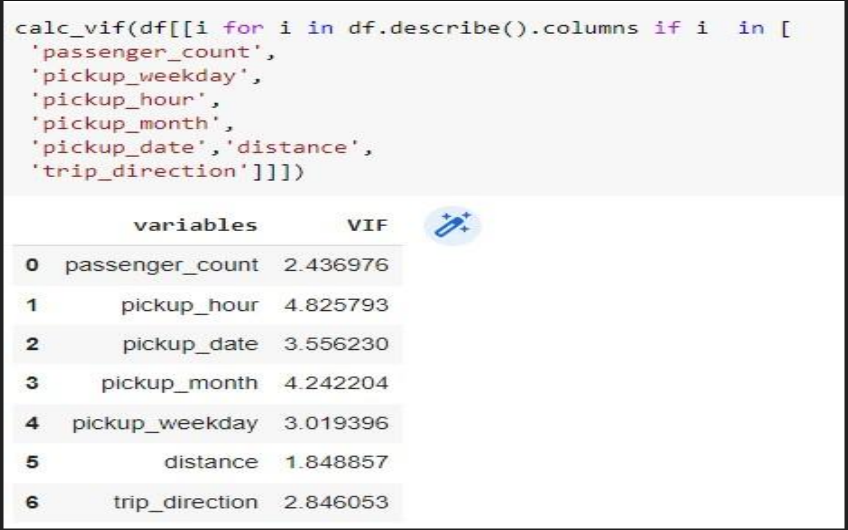


Multicollinearity

```python
## Correlation
plt.figure(figsize=(15,8))
correlation=df.corr()
sns.heatmap(abs(correlation), annot=True, cmap='coolwarm')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f84046e7310>
```



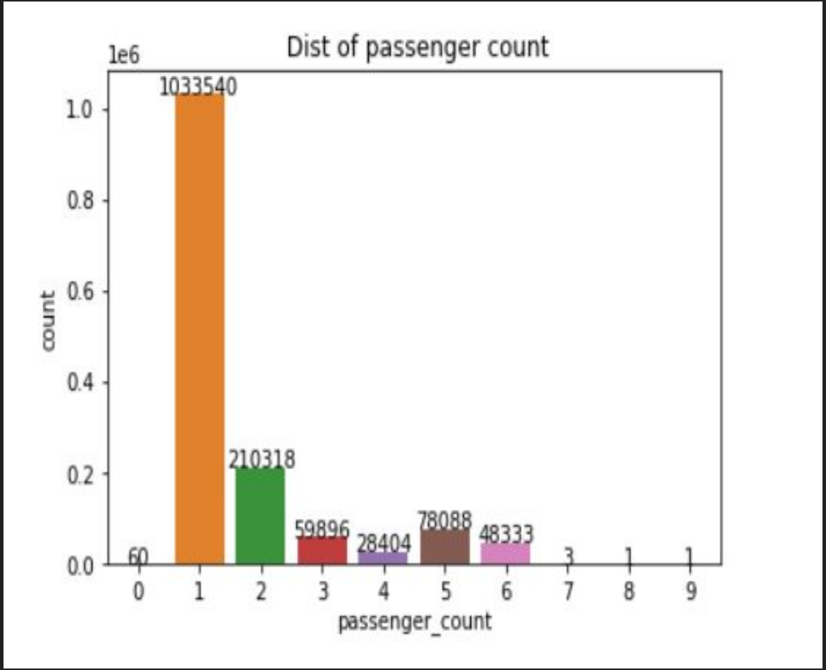


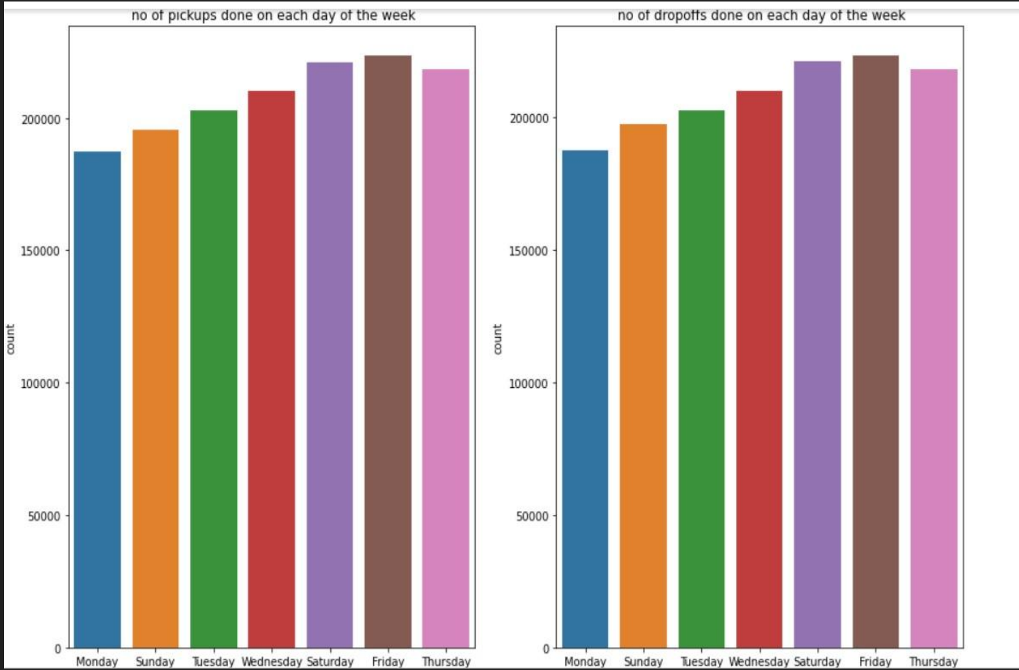
```python
calc_vif(df[[i for i in df.describe().columns if i  in [
 'passenger_count',
 'pickup_weekday',
 'pickup_hour',
 'pickup_month',
 'pickup_date','distance',
 'trip_direction']]])
```

	variables	VIF
0	passenger_count	2.436976
1	pickup_hour	4.825793
2	pickup_date	3.556230
3	pickup_month	4.242204
4	pickup_weekday	3.019396
5	distance	1.848857
6	trip_direction	2.846053

1. We can see a high correlation between pickup_month and dropoff_month,pickup_date and dropoff_date,pickup_weekday and dropoff_weekday.
2. Our target variable shows 0.78 percent correlation with distance.
3. pickup_hour and dropoff hour shows a correlation of 0.68.
4. We drop time_diff_minutes since its highly correlated with trip_duration.
5. In all highly correlated features we can only keep the pickup details and drop the drop off details to remove multicollinearity.

Exploratory Data Analysis (Univariate-Analysis)



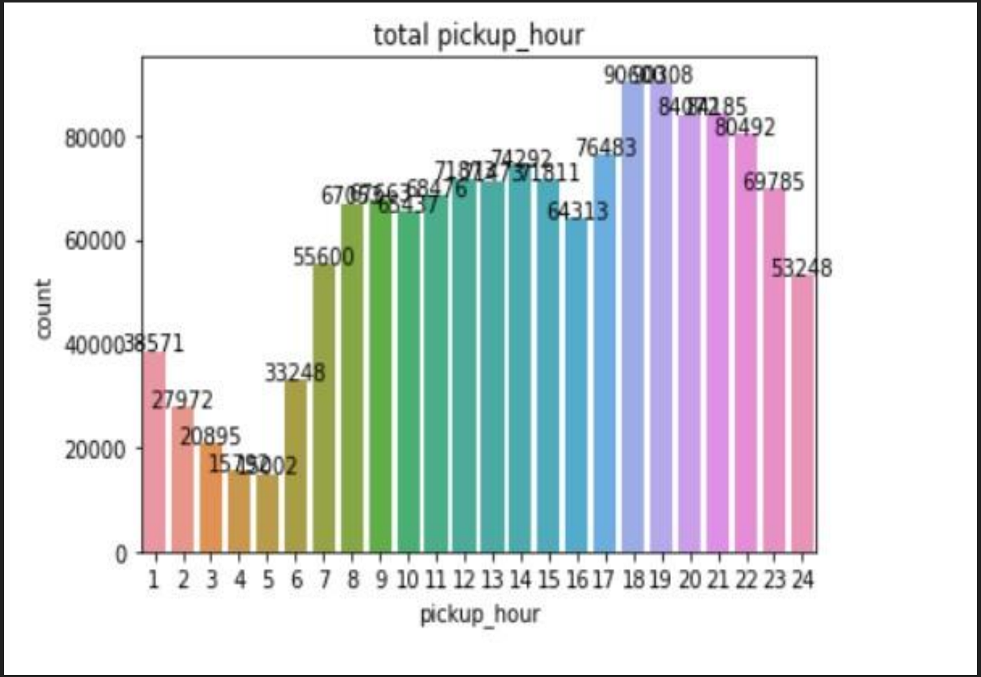


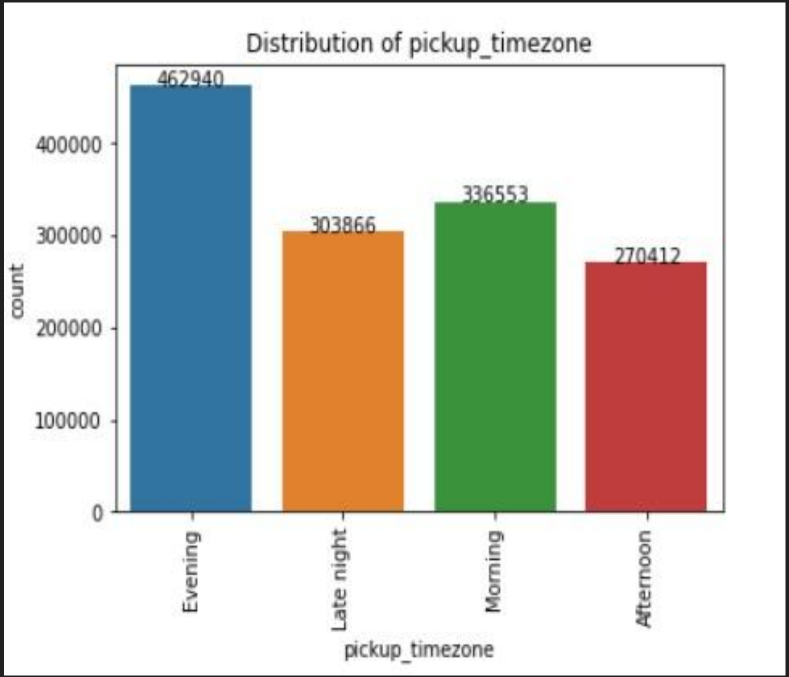
The above plot tells us that the mostly the taxi trip passengers count is one.

It depicts that taxi are mostly prefered by those who likes to travel alone.

The plot tells us that mostly on friday's we can see a high demand for taxi trip and the least number of taxi trip demand on Monday.

EDA (continued)





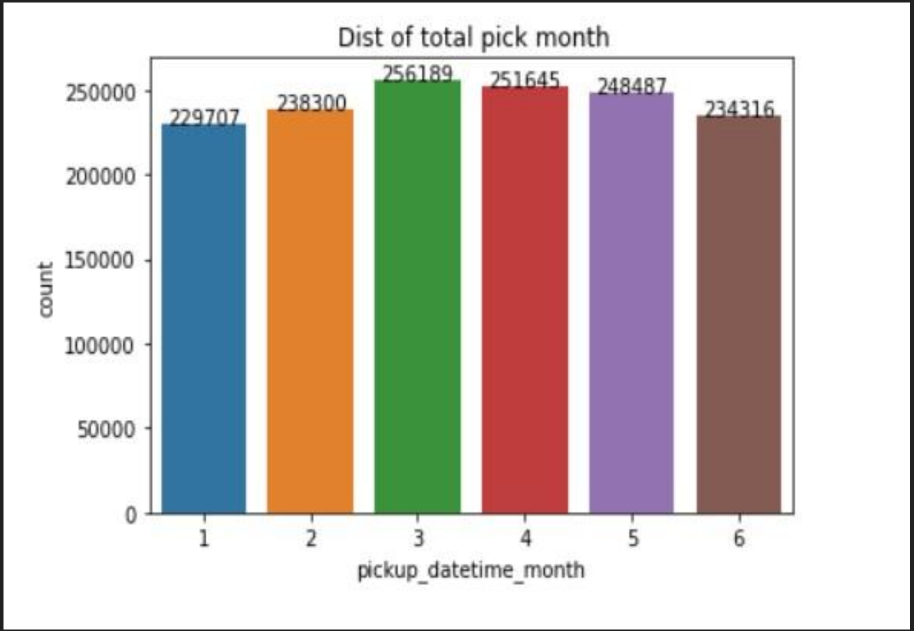
The most busy hours for taxi trip were between evening

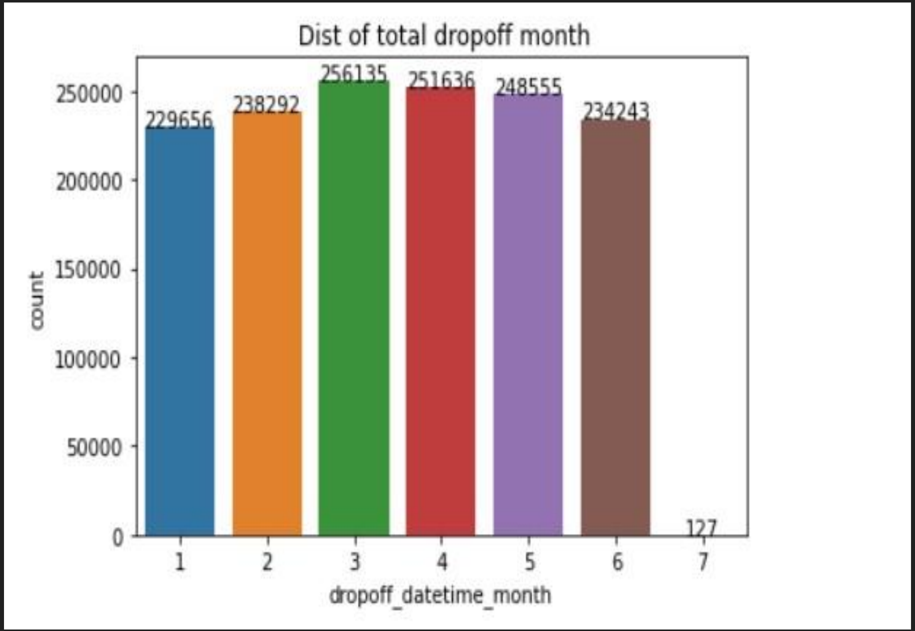
18-22 hr.

The least busy hours were between early morning 2-5 hr.

1. The high demand for taxi trip is in the evening timezone.
2. The least demand for taxi trip is in the afternoon timezone

EDA (continued)

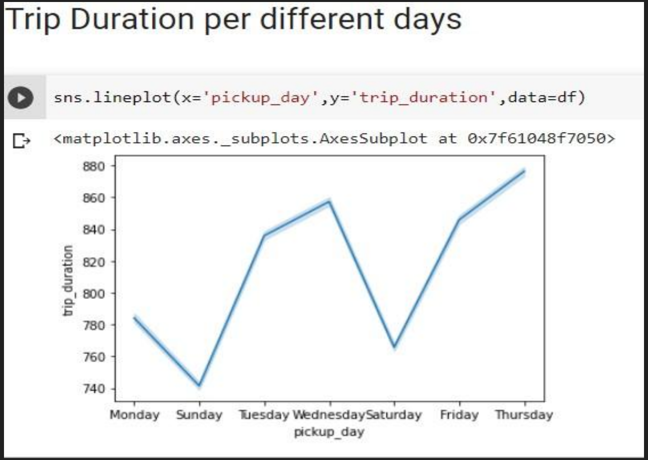




1.The month of March has received the highest number of trips followed by April for both pickup/dropoff.

2.The least number of trips done in the month of January and July.

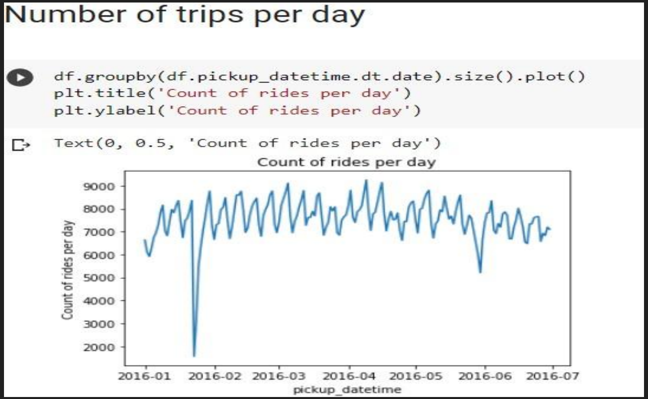
EDA (Bivariate Analysis)



Trip Duration per different days

```
sns.lineplot(x='pickup_day',y='trip_duration',data=df)
```

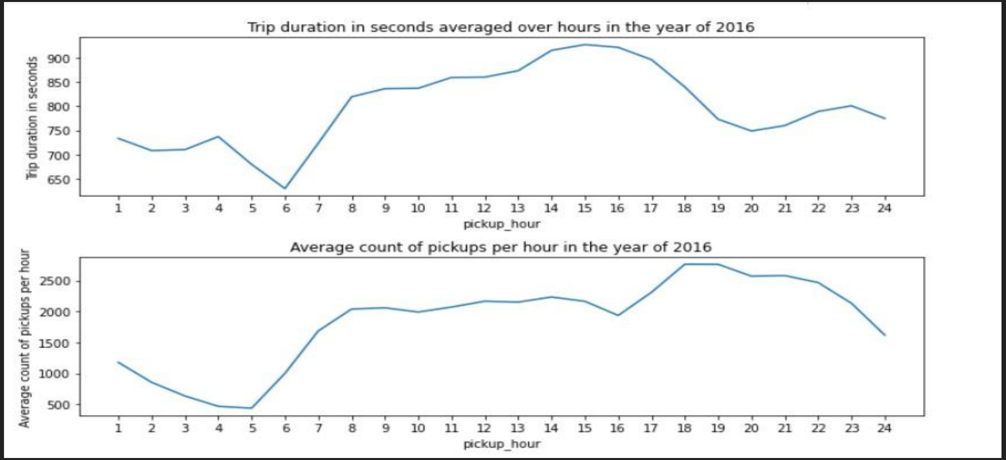
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f61048f7050>
```



Number of trips per day

```
df.groupby(df.pickup_datetime.dt.date).size().plot()
plt.title('Count of rides per day')
plt.ylabel('Count of rides per day')
```

```
Text(0, 0.5, 'Count of rides per day')
```



- Rides increase steadily from 5am to 8am only to flatten out between 8am to 4pm and then again steeply increase from 4pm to 6pm to fall down later.
- The sudden dip in the taxi rides between the 20th and 26th Jan was because of the heavy snow that was observed during that period.
- The trip duration is the maximum in Wednesday and lowest on Sunday

Feature Selection (Preparing dataset for modelling)

Feature Selection is the method of reducing the input variable to your model by using only relevant data and getting rid of noise in data.

To train an optimal model, we need to make sure that we use only the essential features. If we have too many features, the model can capture the unimportant patterns and learn from noise.

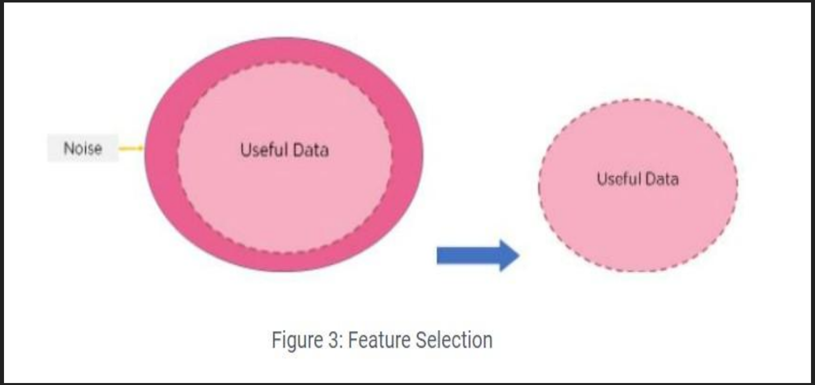
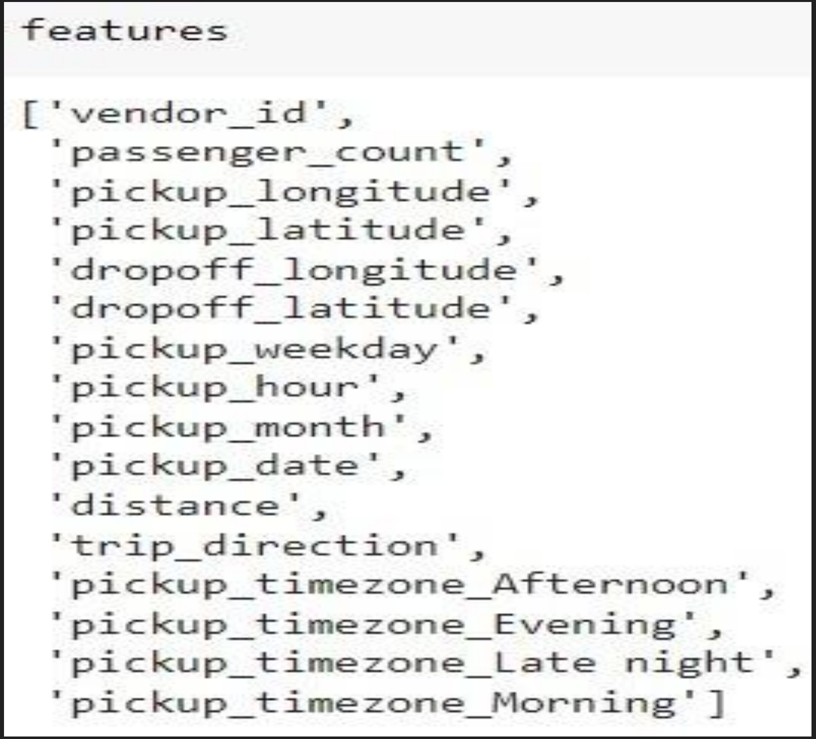


Figure 3: Feature Selection



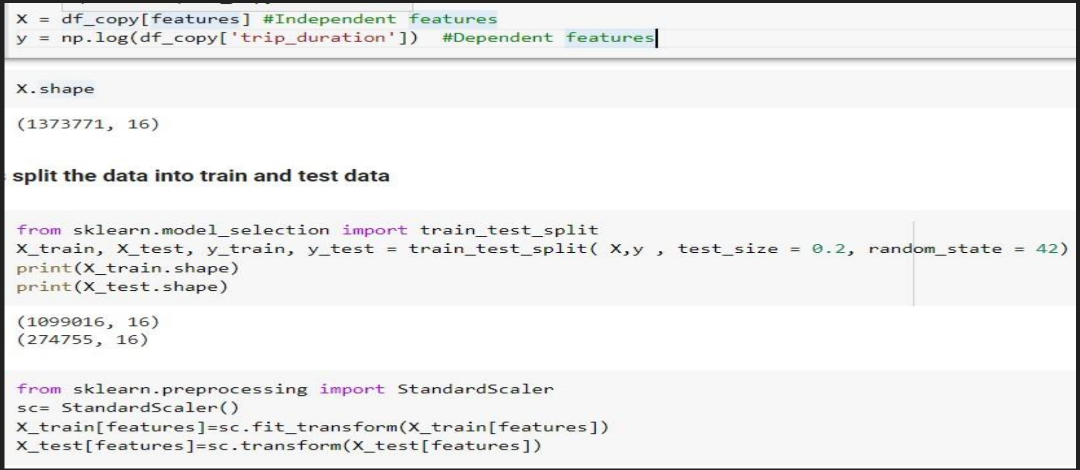
```
features

['vendor_id',
 'passenger_count',
 'pickup_longitude',
 'pickup_latitude',
 'dropoff_longitude',
 'dropoff_latitude',
 'pickup_weekday',
 'pickup_hour',
 'pickup_month',
 'pickup_date',
 'distance',
 'trip_direction',
 'pickup_timezone_Afternoon',
 'pickup_timezone_Evening',
 'pickup_timezone_Late night',
 'pickup_timezone_Morning']
```

Task - Regression Model

Train set - (1099016, 16)

Test set - (274755, 16)

Model Building



```
X = df_copy[features] #Independent features
y = np.log(df_copy['trip_duration'])  #Dependent features
```

```
X.shape
```

```
(1373771, 16)
```

split the data into train and test data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X,y , test_size = 0.2, random_state = 42)
print(X_train.shape)
print(X_test.shape)
```

```
(1099016, 16)
(274755, 16)
```

```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
X_train[features]=sc.fit_transform(X_train[features])
X_test[features]=sc.transform(X_test[features])
```

The Initial Step for Model building will be to split the dataset into train and test data

Then,we have performed Standard Scaler scaling technique on the training set using fit.transform and thereafter transformed the test data .

If our data in any conditions has data points far from each other, scaling is a technique to make them closer to each other or in simpler words, we can say that the scaling is used for making data points generalized so that the distance between them will be lower.

Different Regression Machine Learning Algorithm/Models

The Regression Models which we were used-

1.Linear Regression Model (Simple/Base Model)

2. Lasso and Ridge Regularized Regression Models

3. Decision Tree Model

4. Random Forest (Ensemble Bagging Model)

5. Xgboost (Ensemble Boosting Model)

We have many variables such as the pickup/dropoff month, weekday, hour of the day, trip direction which are not linear variables and can be difficult for a linear regression (LR) algorithm to model without first converting these variables to appropriate forms that can be fed to the LR models

Another alternative is we can use non-linear methods such as decision trees to fit the data. Decision tree doesn't require us to convert the variables because it can split across any values of the variables

But before moving onto decision trees or random forest let's check the performance of LR using only few significant variables such as the distance, hour and weekday.

Evaluation Metrics

MAE is the easiest to understand, because it's the average error.

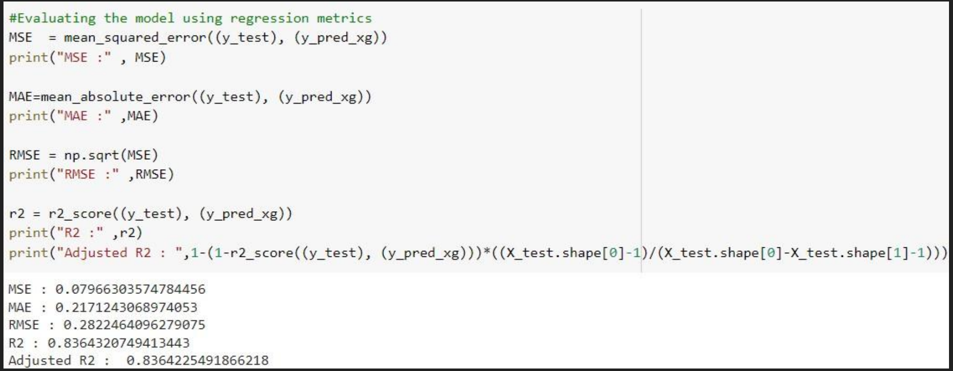
MSE is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.

RMSE is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are loss functions, because we want to minimize them.

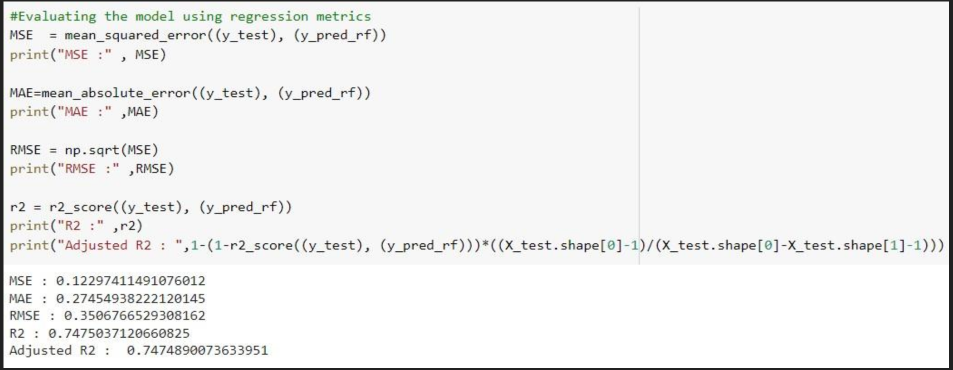
R-square(R^2) is also known as the *coefficient of determination*, It is the proportion of variation in Y explained by the independent variables X. It is the measure of goodness of fit of the model.

Similar to R^2, the Adjusted R^2 measures the variation in the dependent variable.



```
#Evaluating the model using regression metrics
MSE  = mean_squared_error((y_test), (y_pred_xg))
print("MSE :" , MSE)

MAE=mean_absolute_error((y_test), (y_pred_xg))
print("MAE :" ,MAE)

RMSE = np.sqrt(MSE)
print("RMSE :" ,RMSE)

r2 = r2_score((y_test), (y_pred_xg))
print("R2 :" ,r2)
print("Adjusted R2 : ",1-(1-r2_score((y_test), (y_pred_xg)))*((X_test.shape[0]-1)/(X_test.shape[0]-X_test.shape[1]-1)))
```

```
MSE : 0.07966303574784456
MAE : 0.2171243068974053
RMSE : 0.2822464096279075
R2 : 0.8364320749413443
Adjusted R2 :  0.8364225491866218
```

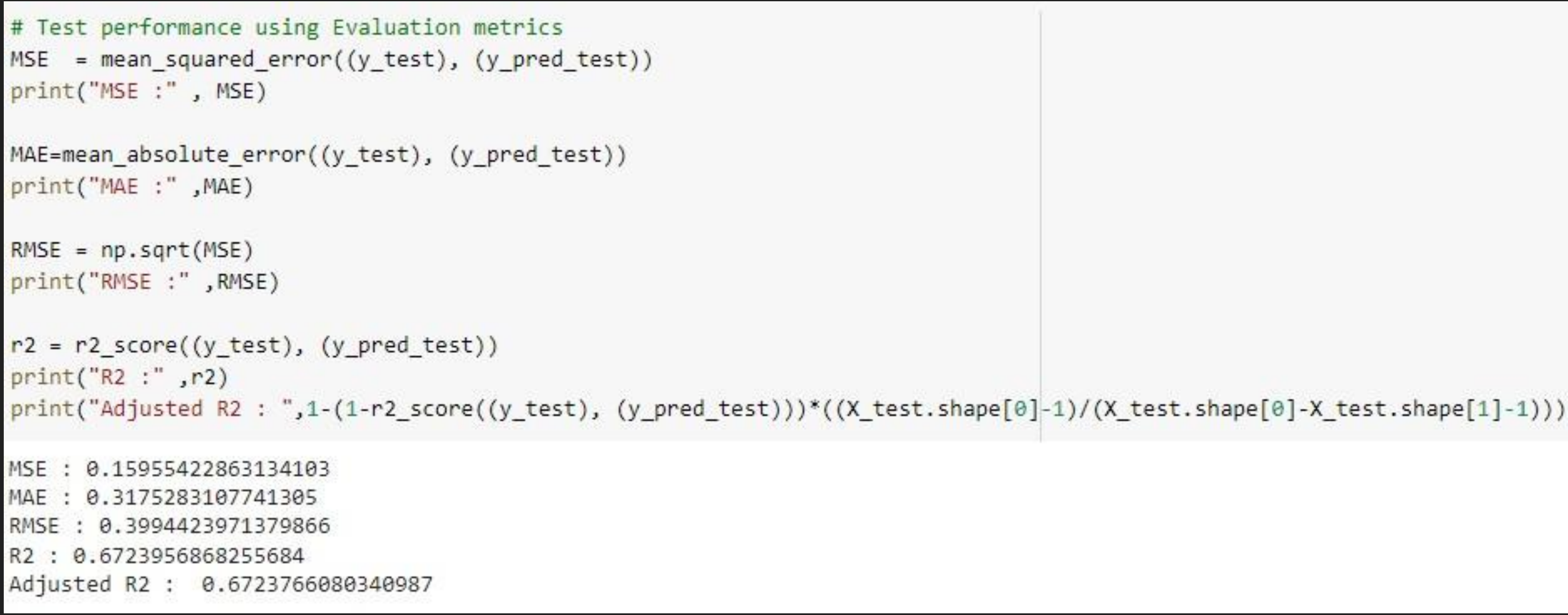


```
#Evaluating the model using regression metrics
MSE  = mean_squared_error((y_test), (y_pred_rf))
print("MSE :" , MSE)

MAE=mean_absolute_error((y_test), (y_pred_rf))
print("MAE :" ,MAE)

RMSE = np.sqrt(MSE)
print("RMSE :" ,RMSE)

r2 = r2_score((y_test), (y_pred_rf))
print("R2 :" ,r2)
print("Adjusted R2 : ",1-(1-r2_score((y_test), (y_pred_rf)))*((X_test.shape[0]-1)/(X_test.shape[0]-X_test.shape[1]-1)))
```

```
MSE : 0.12297411491076012
MAE : 0.27454938222120145
RMSE : 0.3506766529308162
R2 : 0.7475037120660825
Adjusted R2 :  0.7474890073633951
```

Applying Model (Baseline Model)

Linear Regression Model



```
# Test performance using Evaluation metrics
MSE  = mean_squared_error((y_test), (y_pred_test))
print("MSE :" , MSE)

MAE=mean_absolute_error((y_test), (y_pred_test))
print("MAE :" ,MAE)

RMSE = np.sqrt(MSE)
print("RMSE :" ,RMSE)

r2 = r2_score((y_test), (y_pred_test))
print("R2 :" ,r2)
print("Adjusted R2 : ",1-(1-r2_score((y_test), (y_pred_test)))*((X_test.shape[0]-1)/(X_test.shape[0]-X_test.shape[1]-1)))
```

```
MSE : 0.15955422863134103
MAE : 0.3175283107741305
RMSE : 0.3994423971379866
R2 : 0.6723956868255684
Adjusted R2 :  0.6723766080340987
```

Model Validation and Selection

S.No	Model Name	Train Accuracy	Test Accuracy	M.S.E	R.M.S.E	MAE	Adjusted r2 score
1	Linear Regression	0.67	0.67	0.160	0.399	0.317	0.67
2	Ridge	0.7	0.7	0.146	0.382	0.302	0.69
3	Lasso	0.69	0.69	0.147	0.383	0.303	0.69
4	Decision Tree	0.66	0.65	0.165	0.407	0.312	0.65
5	Random Forest	0.74	0.74	0.123	0.351	0.275	0.74
6	Xgboost	0.84	0.83	0.079	0.2822	0.217	0.83

Model Validation and Selection

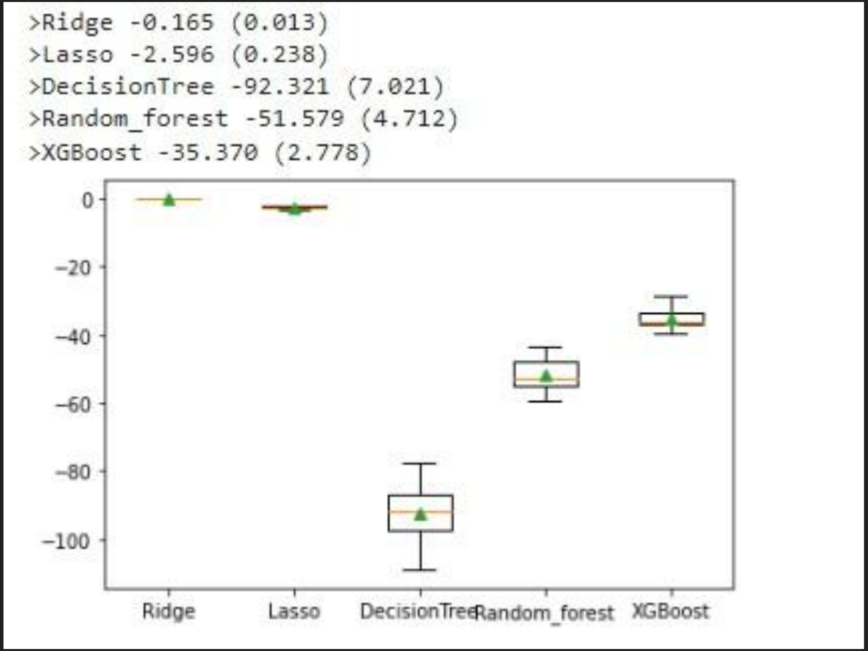
Based on negative mean square error metrics

We can say from looking at the boxplot that Ridge and Lasso Models were the Best Model.

The second best model is Xgboost model followed by Random Forest Model giving third least negative mean square error.

The Decision Tree Model has given the highest negative mean square error and hence underperformed model comparatively to other regression models.

Negative Mean Square Error (BoxPlot)



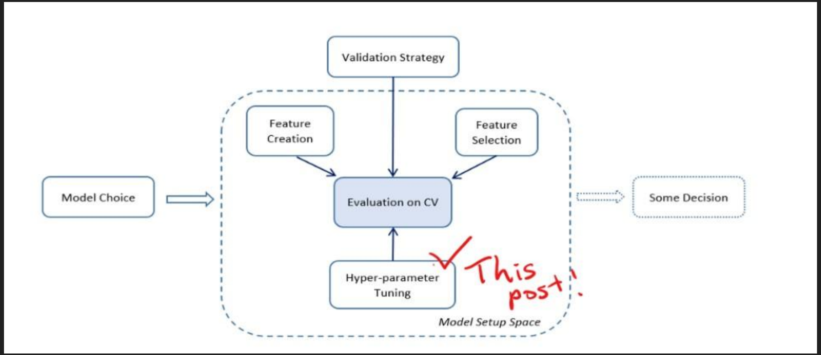
```
>Ridge -0.165 (0.013)
>Lasso -2.596 (0.238)
>DecisionTree -92.321 (7.021)
>Random_forest -51.579 (4.712)
>XGBoost -35.370 (2.778)
```

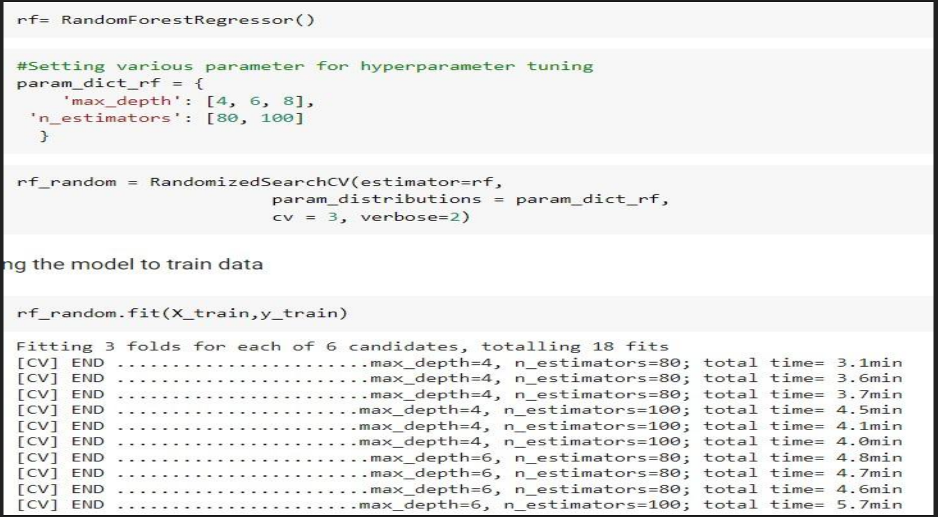
Hyperparameter tuning / Cross- Validation

We have already learned to use train_test_split to split the data, so you can measure model quality on the test data. Compared to train_test_split, cross-validation gives you a more reliable measure of your model's quality, though it takes longer to run.

Hyperparameter Tuning is a method to tune the parameters in a model to optimize the model & give a best accuracy score by selecting the best parameter using cross validation.

In this project,we have used Randomizedsearchcv for hyperparameter tuning,because its comparatively faster than GridSearchCV.



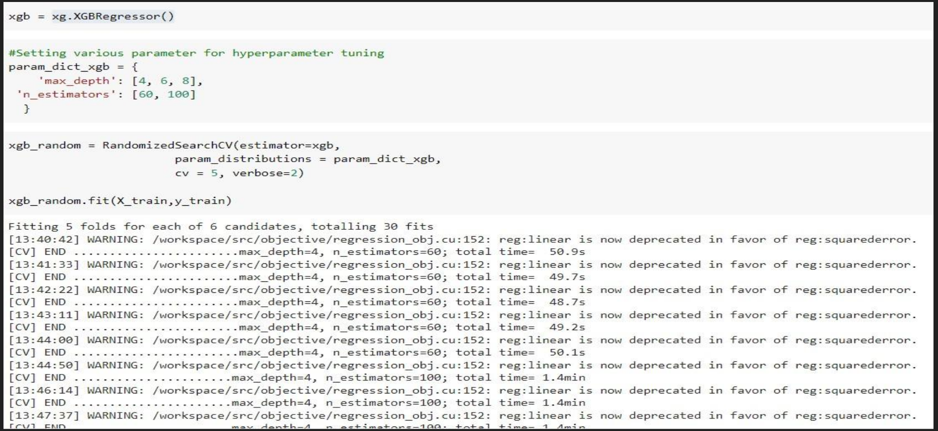


```
rf= RandomForestRegressor()

#Setting various parameter for hyperparameter tuning
param_dict_rf = {
     'max_depth': [4, 6, 8],
 'n_estimators': [80, 100]
  }

rf_random = RandomizedSearchCV(estimator=rf,
                               param_distributions = param_dict_rf,
                               cv = 3, verbose=2)
```

ng the model to train data

```
rf_random.fit(X_train,y_train)
```

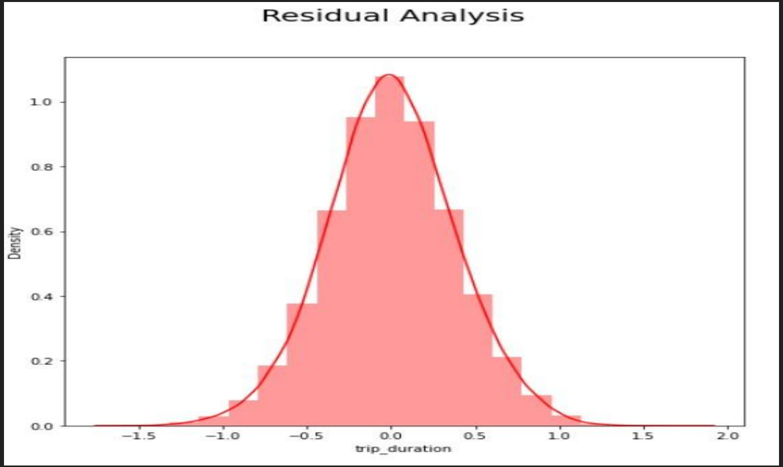
```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
[CV] END .......................max_depth=4, n_estimators=80; total time= 3.1min
[CV] END .......................max_depth=4, n_estimators=80; total time= 3.6min
[CV] END .......................max_depth=4, n_estimators=80; total time= 3.7min
[CV] END ......................max_depth=4, n_estimators=100; total time= 4.5min
[CV] END ......................max_depth=4, n_estimators=100; total time= 4.1min
[CV] END ......................max_depth=4, n_estimators=100; total time= 4.0min
[CV] END .......................max_depth=6, n_estimators=80; total time= 4.8min
[CV] END .......................max_depth=6, n_estimators=80; total time= 4.7min
[CV] END .......................max_depth=6, n_estimators=80; total time= 4.6min
[CV] END ......................max_depth=6, n_estimators=100; total time= 5.7min
```

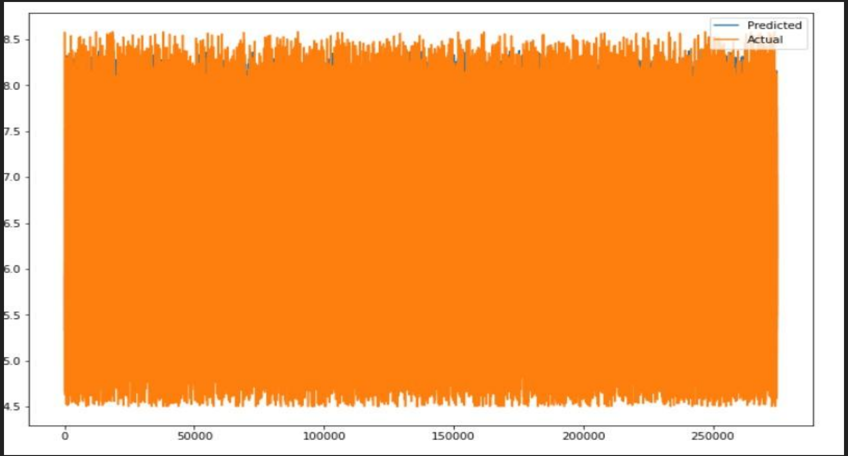


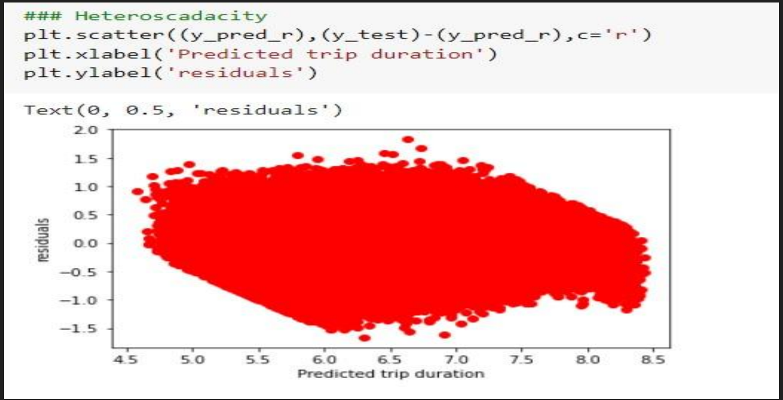
```
xgb = xg.XGBRegressor()

#Setting various parameter for hyperparameter tuning
param_dict_xgb = {
    'max_depth': [4, 6, 8],
 'n_estimators': [60, 100]
  }

xgb_random = RandomizedSearchCV(estimator=xgb,
                                param_distributions = param_dict_xgb,
                                cv = 5, verbose=2)

xgb_random.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[13:40:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV] END .......................max_depth=4, n_estimators=60; total time=  50.9s
[13:41:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV] END .......................max_depth=4, n_estimators=60; total time=  49.7s
[13:42:22] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV] END .......................max_depth=4, n_estimators=60; total time=  48.7s
[13:43:11] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV] END .......................max_depth=4, n_estimators=60; total time=  49.2s
[13:44:00] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV] END .......................max_depth=4, n_estimators=60; total time=  50.1s
[13:44:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV] END ......................max_depth=4, n_estimators=100; total time= 1.4min
[13:46:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV] END ......................max_depth=4, n_estimators=100; total time= 1.4min
[13:47:37] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV] END ......................max_depth=4, n_estimators=100; total time= 1.4min
```

Scatter Plot / Residual Plot / Heteroscedasticity Plot

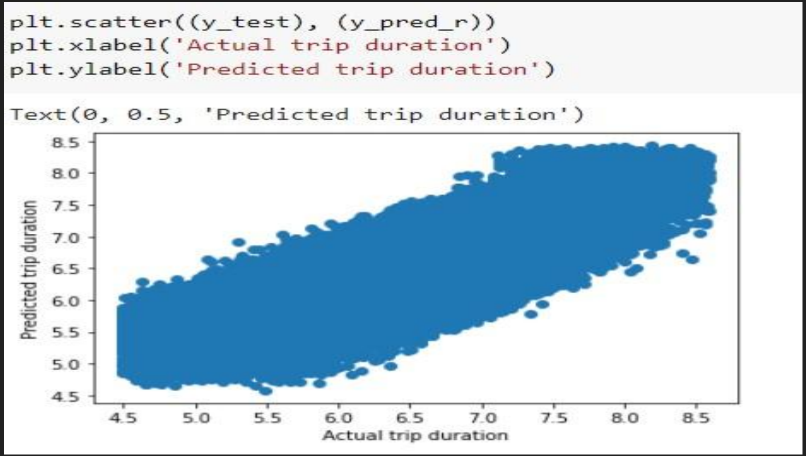






```
### Heteroscadacity
plt.scatter((y_pred_r),(y_test)-(y_pred_r),c='r')
plt.xlabel('Predicted trip duration')
plt.ylabel('residuals')
```

```
Text(0, 0.5, 'residuals')
```



```
plt.scatter((y_test), (y_pred_r))
plt.xlabel('Actual trip duration')
plt.ylabel('Predicted trip duration')
```

```
Text(0, 0.5, 'Predicted trip duration')
```

Conclusion

1.We can also use stacking algorithm over here to find the accuracy,but due to the high computation time we didn't used it.

2.We got the best model accuracy in Xgboost model,r2 score of 0.84 in training set and 0.83 in test data..The RMSE score of Xgboost model is 0.282

3.Our second best model is Random Forest based with a r2 score of 0.7485 accuracy in training and 0.7475 accuracy in test data. The RMSE score of random forest is 0.350

4.Our base model (Linear Regression) gave us a r2_score of 0.67 in train and test data and keep this model accuracy in reference to compare other model.

5.Decision Tree has given us least score in evaluation compared to all other algorithms based on negative mean absolute error.

6.We just used ensemble techniques over here to demonstrate various other options for these kind of regression problems

7.We can also do Model Explainability of random forest and Xgboost model over here using SHAP or LIME

Challenges

- Extracting new features from existing features were a bit tedious job to do.
- Handling Outliers in Independent features was bit lengthy process.
- There were few irrelevant data records present in the dataset.
- The Randomised Search cv took nearly more than a hour to run ,so it was really time consuming task.

Thank You!!!