National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Janvi Rajesh Rajani
Student ID: X20148712

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Janvi Rajesh Rajani		
Student ID:	X20148712		
Programme:	MSc. Data Analytics	**Year:**	2021
Module:	MSc. Research Project		
Lecturer:	Dr. Catherine Mulwa		
Submission Due Date:	31/01/2022		
Project Title:	Taxi Trip Time and Trajectory Prediction Using Machine Learning		
Word Count:	832 **Page Count:** 15		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Janvi Rajesh Rajani
……

Date: 31/01/2022……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Janvi Rajesh Rajani
X20148712

1 Hardware Setup

The hardware setup for the research is shown in Table 1. It features a RAM of 8.00 GB and runs on a 64-bit operating system. AMD Ryzen 5 is the processor used.

Table 1. Hardware Setup

Processor	AMD Ryzen 5 4500U with Radeon Graphics
RAM	8.00 GB
System Type	64-bit operating system, x64-based processor

2 Software Setup

The Software used in this research is Anaconda- Jupyter Notebook. It is downloaded from its official website[i]. It has multiple applications, Jupyter Notebook is used in this particular research as shown in the Figure 1.

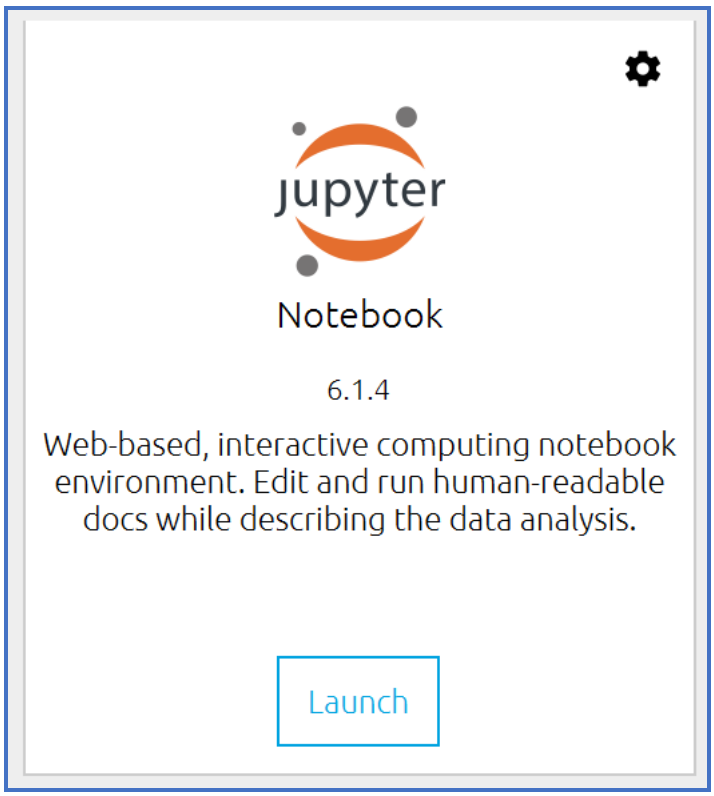

Figure 1: Jupyter Notebook

After Launching the Jupyter Notebook application, a new window will be launched and on the right-hand corner click on New Python 3 Notebook as shown in the Figure 2. This will open a new Python Notebook.

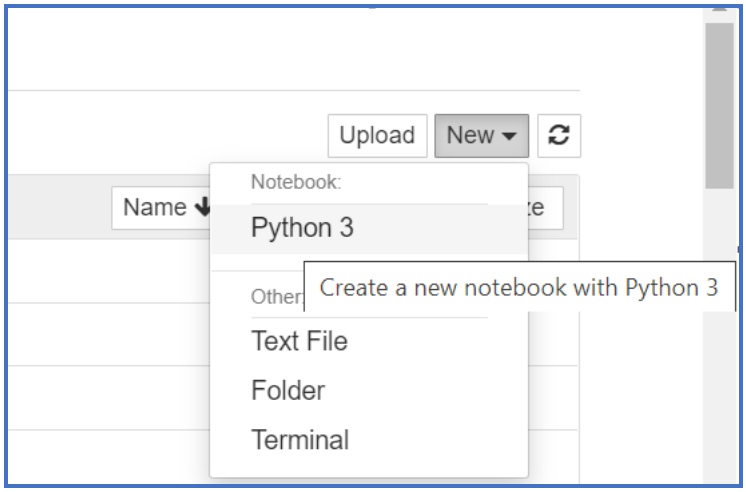

Figure 2. Python Notebook

3 Package Requirement and Instalment

The primary software used for the research is Python. Table 2 shows the list of packages used:

Table 2. Required Python Packages

Numpy
Pandas
Matplotlib
Sklearn
Math
Datetime
Seaborn

A list of libraries required to complete this project are provided. Ensure the libraries are installed on Python to ensure a smooth operation.

4 Data Preparation and Transformation

The data used in this research is publicly available on kaggle[ii]. The Figure 3 shows the data loading in python.

```
#Storing the data in Csv file
#Loading the data from Csv file
import pandas as pd
taxi_df = pd.read_csv("train.csv")
taxi_df
```

Figure 3. Data Loading

The Figure 4 shows the libraries imported for the smooth execution of the project.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score
from math import sqrt
from sklearn.metrics import mean_absolute_error as mae
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn import neighbors
```

Figure 4. Essential Packages Imported

5 Data Pre-Processing

The Figure 5 shows data pre-processing where the NULL values are replaced with 0, so that the NULL values does not affect the performance of the machine learning model.

```
taxi_df['ORIGIN_CALL']=taxi_df['ORIGIN_CALL'].fillna(0)
taxi_df['ORIGIN_STAND']=taxi_df['ORIGIN_STAND'].fillna(0)
```

Figure 5. NULL replaced with zero

The Figure 6 shows how timestamp column is divided into year, month, month day, hour and week day. Later, these features are used in the Machine Learning techniques.

```
#Extracting year, month, hour from TIMESTAMP column
taxi_df['year'] = taxi_df['TIMESTAMP'].apply(lambda x :datetime.datetime.fromtimestamp(x).year)
taxi_df['month'] = taxi_df['TIMESTAMP'].apply(lambda x :datetime.datetime.fromtimestamp(x).month)
taxi_df['month_day'] = taxi_df['TIMESTAMP'].apply(lambda x :datetime.datetime.fromtimestamp(x).day)
taxi_df['hour'] = taxi_df['TIMESTAMP'].apply(lambda x :datetime.datetime.fromtimestamp(x).hour)
taxi_df['week_day'] = taxi_df['TIMESTAMP'].apply(lambda x :datetime.datetime.fromtimestamp(x).weekday())
```

Figure 6: Timestamp Divided into Other Attributes

The Figure 7 shows using polyline feature and lambda function, new attribute is made Polyline Length. It depicts the total length of the taxi trip in seconds.

```
taxi_df['Polyline Length'] = taxi_df['POLYLINE'].apply(lambda x : len(eval(x))-1)
```

```
taxi_df['Trip Time(sec)'] = taxi_df['Polyline Length'].apply(lambda x : x * 15)
```

Figure 7: Polyline Length Feature

6 Additional Exploratory Data Visualization

The Figure 8 shows the bar chart of the total trips for each month. It is seen that May has highest number of taxi trips.

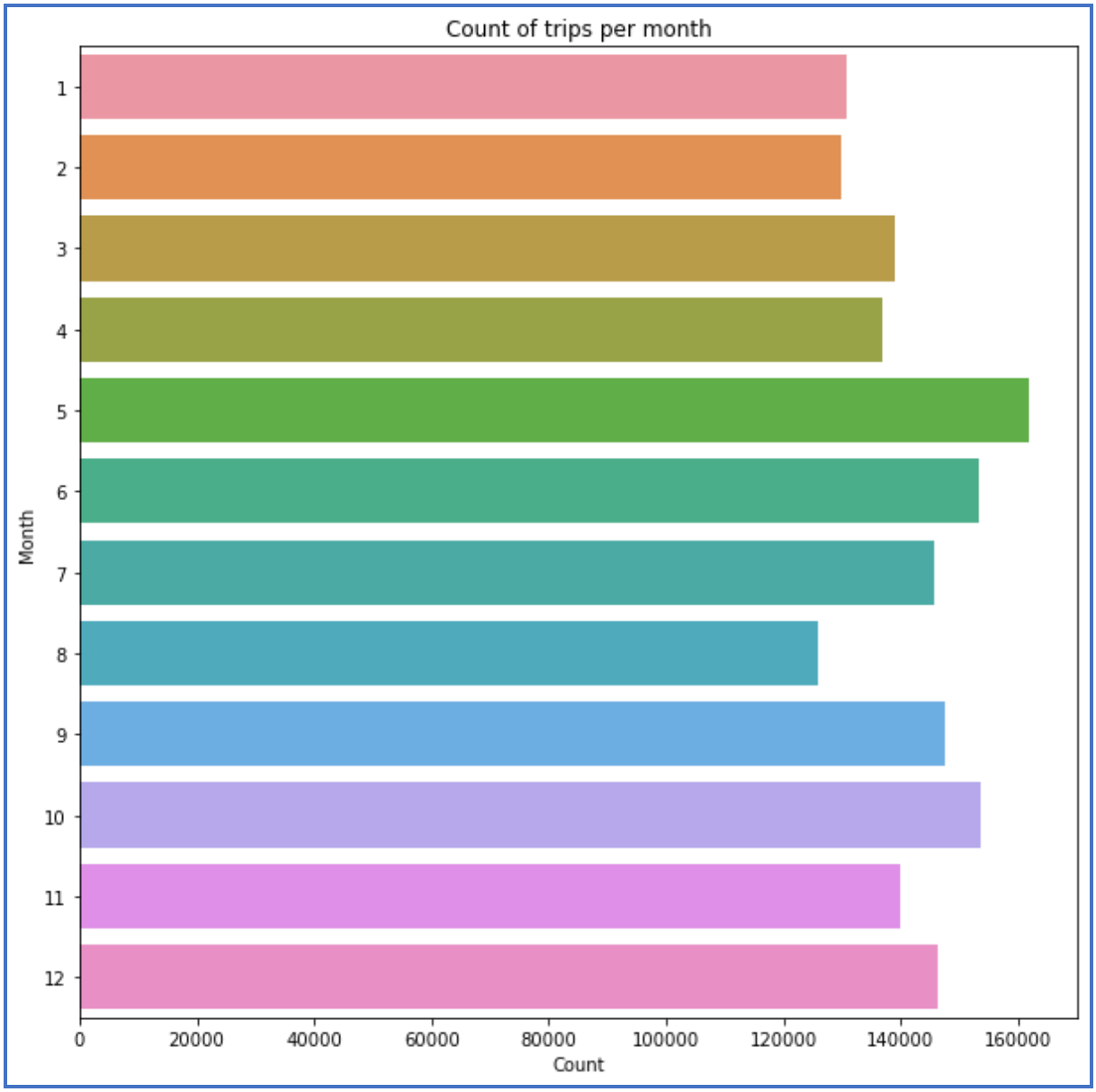

Figure 8. Trips Per Month

The Figure 9 shows the bar chart of the total trips for entire day. It is seen that there are less trips in the morning whereas it is substantially increased 5 throughout the day and then gradually decreases at night. It means that maximum taxi trips are taken in the day.

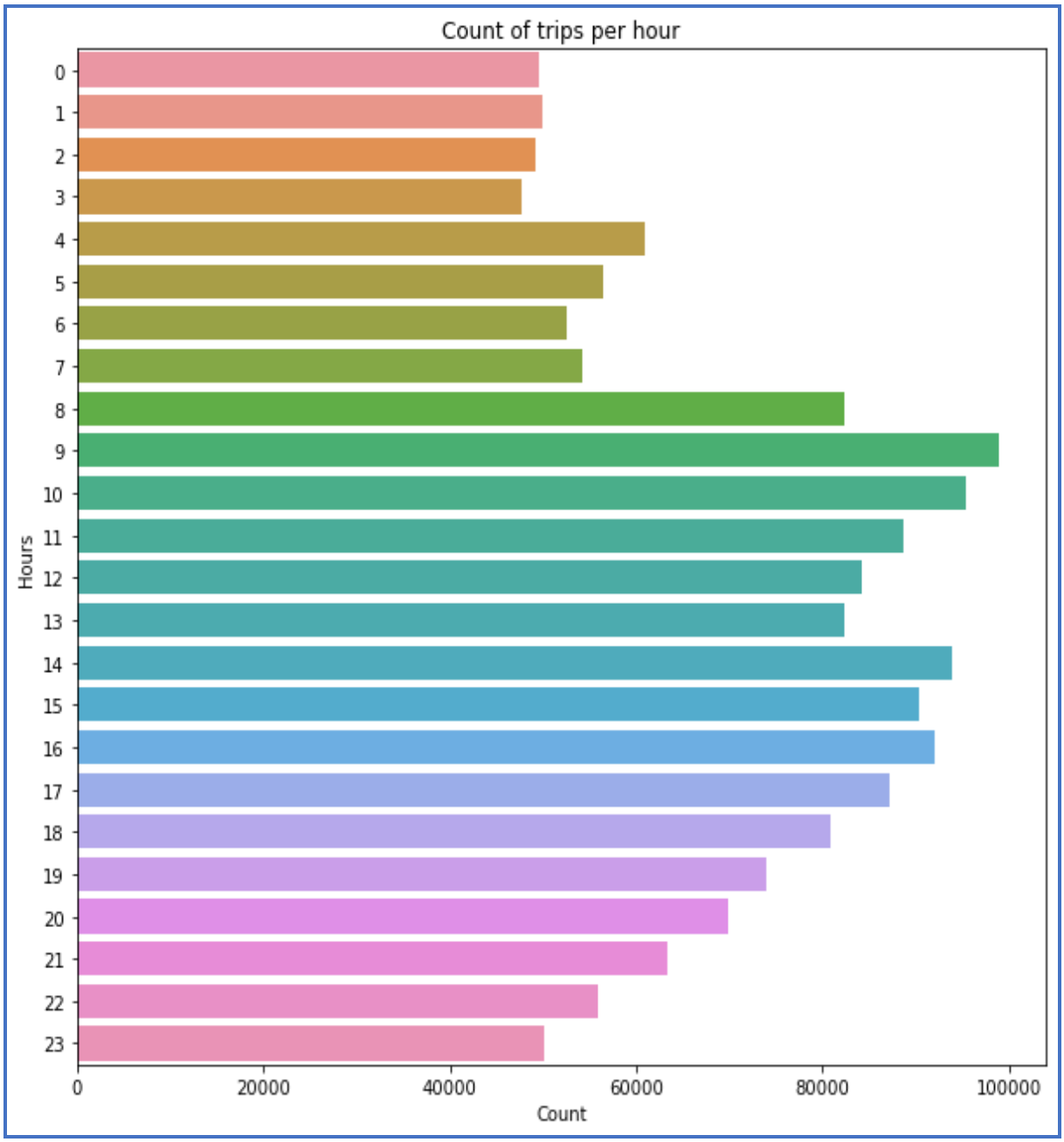

Figure 9. Trips per Hour

7 Data Standardization

The data is divided X and y. Initially all the values are considered for X and only the dependent value is taken for y as shown in the Figure 10. The data is then Standardized using the StandardScaler function. Data Standardization is essential since it improves the quality of data and gives consistency to the data which in turn makes it easy for use.

```
taxi_df = taxi_df.reset_index()
```

```
X = taxi_df[[ 'ORIGIN_CALL', 'ORIGIN_STAND', 'TIMESTAMP',
              'year', 'month', 'month_day', 'hour', 'week_day', 'lon_1st', 'lat_1st', 'lon_last', 'lat_last', 'delta_lon',
              'delta_lat', 'Polyline Length', 'CALL_TYPE_A', 'CALL_TYPE_B', 'CALL_TYPE_C','DAY_TYPE_A']]
y = taxi_df['Trip Time(sec)']
```

Data Standardization

```
s = StandardScaler()
X = s.fit_transform(X)
```

```
print(np.mean(X))
np.std(X)
```

```
1.9355573050760493e-14

0.9176629354822471
```

Figure 10. Data Standardization

8 Feature Selection

The Figure 11. shows the feature selection using Linear Regression. It uses importance feature and it gives features score.

```
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot
model=LinearRegression()
model.fit(X,y)
importance=model.coef_
for i,v in enumerate(importance):
    print('Feature:%0d,Score:%.5f'% (i,v))
pyplot.bar([x for x in range(len(importance))],importance)
pyplot.show()
```

Figure 11. Feature Selection using Linear Regression

The Figure 12. shows the score of each feature and the unwanted features are given 0 score by the Linear Regression model.

```
Feature:0,Score:-0.00000
Feature:1,Score:-0.00000
Feature:2,Score:0.00000
Feature:3,Score:-0.15368
Feature:4,Score:-0.03916
Feature:5,Score:-0.00000
Feature:6,Score:-0.00000
Feature:7,Score:-0.00000
Feature:8,Score:10.06428
Feature:9,Score:-6.61688
Feature:10,Score:-12.03310
Feature:11,Score:8.97671
Feature:12,Score:13.52261
Feature:13,Score:-9.97386
Feature:14,Score:685.91455
Feature:15,Score:3.52336
Feature:16,Score:4.27800
Feature:17,Score:3.87050
Feature:18,Score:0.00000
```

Figure 12. Feature Score

Figure 13. shows the feature selection using co-relation matrix.

```
plt.figure(figsize=(18,15))
cor = taxi_df_tra.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```

Figure 13. Co-relation Matrix

Figure 14. shows the feature selection using Co-relation Matrix.

```
#Correlation with output variable
cor_target = abs(cor["lon_last"])
#Selecting highly correlated features
relevant_features = cor_target[cor_target>0.1]
relevant_features
```

Figure 14. Features Using Co-relation Matrix

9 Splitting into Test and Train

Figure 15. shows the test train data split taxi trip time Prediction

```
X = taxi_df[[  'year', 'month','lon_1st', 'lat_1st', 'lon_last', 'lat_last', 'delta_lon',
            'delta_lat', 'Polyline Length', 'CALL_TYPE_A', 'CALL_TYPE_B', 'CALL_TYPE_C']]
y = taxi_df['Trip Time(sec)']
```

```
X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size = 0.2)
```

```
print("The size of training input is", X_train.shape)
print("The size of training output is", y_train.shape)
print(50 *'*')
print("The size of testing input is", X_test.shape)
print("The size of testing output is", y_test.shape)
```

```
The size of training input is (40000, 12)
The size of training output is (40000,)
**************************************************
The size of testing input is (10000, 12)
The size of testing output is (10000,)
```

Figure 15. Data Split for taxi trip time Prediction

Figure 16. shows the test train data split taxi trip trajectory Prediction

```
X_tra = taxi_df_tra[["lon_1st", "delta_lon","delta_lon"]]

y_tra = taxi_df_tra[["lon_last","lat_last"]]

X_train_tra, X_test_tra, y_train_tra, y_test_tra = train_test_split(X_tra, y_tra, test_size=0.3, random_state=1)
```

Figure 16. Data Split for taxi trip Trajectory

10 Machine Learning Models to Predict Taxi Trip Time

Figure 17. shows the implementation of the Baseline Model

```
y_train_pred = np.ones(X_train.shape[0]) * y_train.mean() #Predicting the train results

y_test_pred = np.ones(X_test.shape[0]) * y_test.mean() #Predicting the test results

print("Test Results for Baseline Model:")
print(50 * '-')
print("Root mean squared error: ", sqrt(mse(y_test, y_test_pred)))
print("R-squared: ", r2_score(y_test, y_test_pred))
print("Mean Squared Error:",mse(y_test, y_test_pred))
print("Mean Absolute Error:",mae(y_test, y_test_pred))
```

Figure 17. Baseline Model

Figure 18. shows the implementation of KNN Regressor.

```
rmse_val = [] #to store rmse values for different k
for K in range(20):
    K = K+1
    knn_regressor = neighbors.KNeighborsRegressor(n_neighbors = K)

    knn_regressor.fit(X_train,y_train)  #fit the model
    y_test_pred=knn_regressor.predict(X_test) #make prediction on test set
    error_test = sqrt(mse(y_test,y_test_pred)) #calculate rmse on test set
    rmse_val.append(error_test) #store test rmse values
    print('RMSE value for k= ' , K , 'is:', error_test)
```

Figure 18. KNN regressor

Figure 19. shows implementation of Lasso Regressor

```
params ={'alpha' :[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]}
lasso_regressor =GridSearchCV(Lasso(), params ,cv =15,scoring = 'neg_mean_absolute_error', n_jobs =-1)
lasso_regressor.fit(X_train,y_train)
```

Figure 19. Lasso regression

Figure 20. shows the implementation of Decision Tree Regressor

```
depth  =list(range(3,30))
param_grid =dict(max_depth =depth)
regr =GridSearchCV(DecisionTreeRegressor(),param_grid,cv =10)
regr.fit(X_train,y_train)
```

Figure 20. Decision Tree Regressor

Figure 21. and Figure 22. Shows the visualization of Decision tree Regressor

```
import graphviz
from sklearn.tree import export_graphviz
from sklearn import tree

dot_data = tree.export_graphviz(regr.best_estimator_, out_file=None,

                                filled=True)
graphviz.Source(dot_data, format="png")
```

Figure 21. Decision Tree Regression visualization

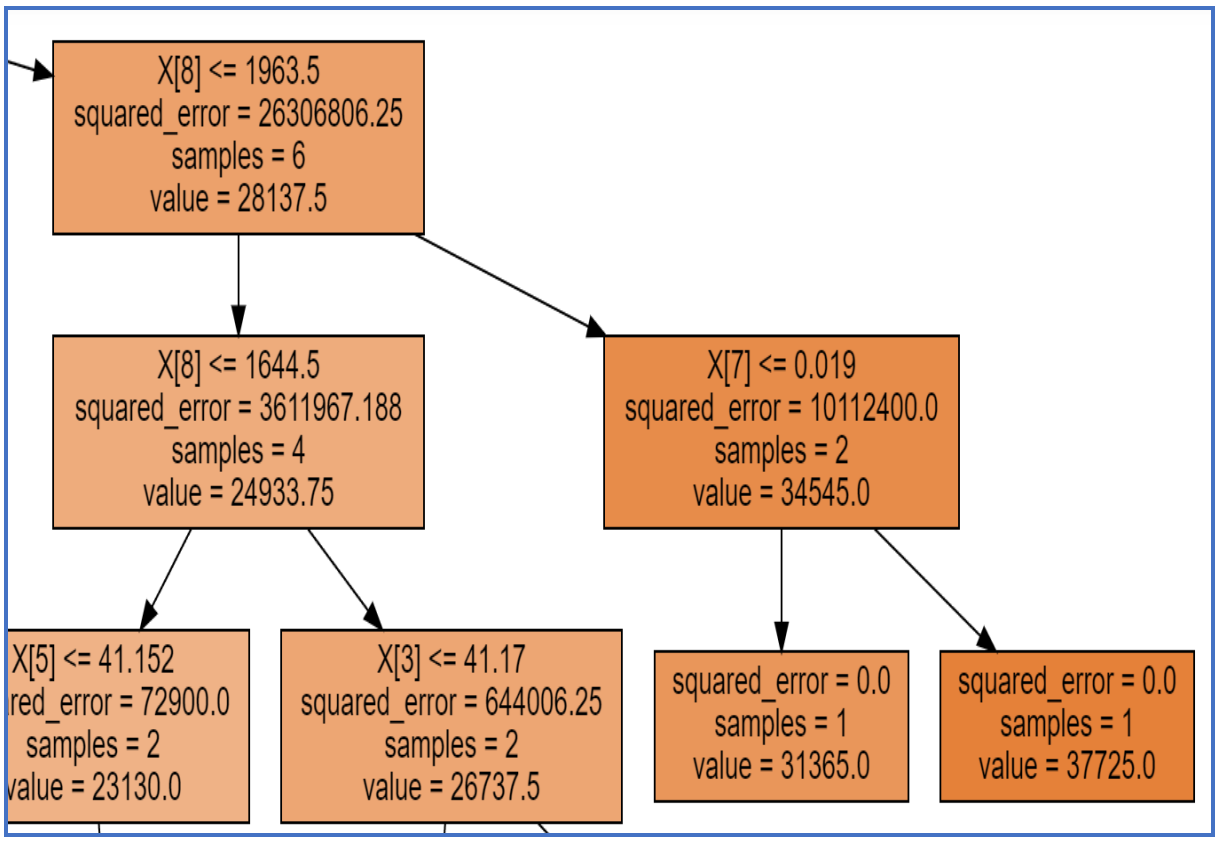

Figure 22. Decision tree regressor Visualization

Figure 23. shows XGBoost Regression implementation

```
tuned_params = {'max_depth': [1, 2, 3, 4, 5], 'learning_rate': [0.01, 0.05, 0.1], 'n_estimators': [100, 200, 300, 400, 500]}
model = RandomizedSearchCV(XGBRegressor(), tuned_params, n_iter=20, scoring = 'neg_mean_absolute_error', cv=5, n_jobs=-1)
model.fit(X_train, y_train)
```

Figure 23. XGBoost Regression

Figure 24. shows implementation Random Forest Regression

```
tuned_params = {'n_estimators': [100, 200, 300, 400, 500], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4]}
random_regressor = RandomizedSearchCV(RandomForestRegressor(), tuned_params, n_iter = 20, scoring = 'neg_mean_absolute_error',
random_regressor.fit(X_train, y_train)
```

Figure 24. Random Forest Regression

11 Machine Learning Algorithms to Predict Taxi Trip Trajectory

11.1 Normality and Histogram for Checking Linearity

Figure 25. shows the first target variable lon_last histogram and Normality plot.

```
#Normality
#histogram and normal probability plot
from scipy.stats import norm
from scipy import stats
sns.set(rc={'figure.figsize':(7,5)})
sns.distplot(taxi_df_tra['lon_last'], fit=norm);
fig = plt.figure()
res = stats.probplot(taxi_df_tra['lon_last'], plot=plt)
```

Figure 25. Histogram and Normality plot for lon_last

Figure 26. shows the first target variable lat_last histogram and Normality plot.

```
#Normality
#histogram and normal probability plot
from scipy.stats import norm
from scipy import stats
sns.set(rc={'figure.figsize':(7,5)})
sns.distplot(taxi_df_tra['lat_last'], fit=norm);
fig = plt.figure()
res = stats.probplot(taxi_df_tra['lat_last'], plot=plt)
```

Figure 26. Histogram and Normality plot for lat_last

Figure 27. shows Multiple Linear Regression implementation

```
from sklearn.linear_model import LinearRegression
#Fitting the Multiple Linear Regression model
mlr = LinearRegression()
mlr.fit(X_train_tra, y_train_tra)
```

```
LinearRegression()
```

Figure 27. Multiple Linear Regression

Figure 28. shows Gradient Boosting Regression implementation

```
from sklearn.multioutput import MultiOutputRegressor
from sklearn.ensemble import GradientBoostingRegressor
bosting = MultiOutputRegressor(GradientBoostingRegressor(random_state=0))

# Fitting
bosting = bosting.fit(X_train_tra, y_train_tra)
y_train_pred_tra = bosting.predict(X_train_tra)
y_test_pred_tra = bosting.predict(X_test_tra)
```

Figure 28. Gradient Boosting Regression

12 Evaluation Metrices

Figure 29. shows the evaluation metrices used to evaluate all the Machine Learning Models.

```
print("Root mean squared error: ", sqrt(mse(y_test, y_test_pred)))
print("R-squared: ", r2_score(y_test, y_test_pred))
print("Mean Squared Error:",mse(y_test, y_test_pred))
print("Mean Absolute Error:",mae(y_test, y_test_pred))
```

Figure 29. Evaluation Metrices

[i] https://www.anaconda.com/products/individual

[ii] https://www.kaggle.com/crailtap/taxi-trajectory