INSTITUTE FOR ADVANCED COMPUTING AND SOFTWARE DEVELOPMENT AKURDI, PUNE

Documentation On

"**New York Taxi Fare Price Prediction**"

PG-DBDA Mar 2022

Submitted By:

Group No: 13

Harshita Paliwal 223322

Akash Hingane 223323

Mr. Prashant Karhale
Centre Coordinator

Mr. Akshay Tilekar
Project Guide

INDEX

Sr. No	Chapter Name		Page No
1	**Introduction**		3 - 3
	1.1	Problem Statement	
	1.2	Abstract	
	1.3	Product Scope	
	1.4	Aim And objective	
2	**Overall Description**		4 - 19
	2.1	Work Flow of Project	
	2.2	Dataset Description	
	2.3	Data Preprocessing	
	2.4	Exploratory Data Analysis	
3	**Model Building**		20 - 32
	3.1	XGBOOST Algorithm	
	3.2	Random Forest Algorithm	
	3.3	Artificial Neural Network	
	3.4	Model Comparison	

4	**User Interface**		33 – 39
	4.1	Streamlit API	
	4.2	AWS Cloud Deployment	
5	**Requirement Specification**		40 – 40
	5.1	Hardware Requirement	
	5.2	Software Requirement	
	5.3	Libraries	
6	**Future Work**		41 - 41
7	**Conclusion**		41 – 41
8	**Project Closure Report**		42 – 42
	8.1	General Project Information	
	8.2	Project Closure report stagewise	
9	**References**		43 - 43

1. Introduction

1.1 Problem Statement - New York City Taxi Fare Price Prediction

Taxi companies: Companies can maximize their utilization by diverting the cabs into the locations during specific times.

New York citizen: Citizen can book or get a taxi with right fare amount. They should also know how much amount of money should be spend for fare before travel.

1.2 Abstract -

New York City, being the most populous city in the United States. It has a vast and complex transportation system, including one of the largest subway systems in the world. The subway system digests the lion share of NYC's public transport use, but the 54% of NYC's residents that don't own a car and therefore rely on public transportation still take almost 200 million taxi trips per year.

The primary objective of this project is to predict fare price throughout New York City as it changes from day to day and hour to hour. So, given a specific location, time and passenger count we predict the number of pickups and drop-off in that location.

1.3 Product Scope -

The main use of this regression model is to predict the right fare amount for a trip by the user in our web interface. The user will input the pickup location, drop-off location, date, time and number of passengers and after pressing the Predict Button, this model will predict the fare amount for the trip he wants to travel and send that amount back to the user.

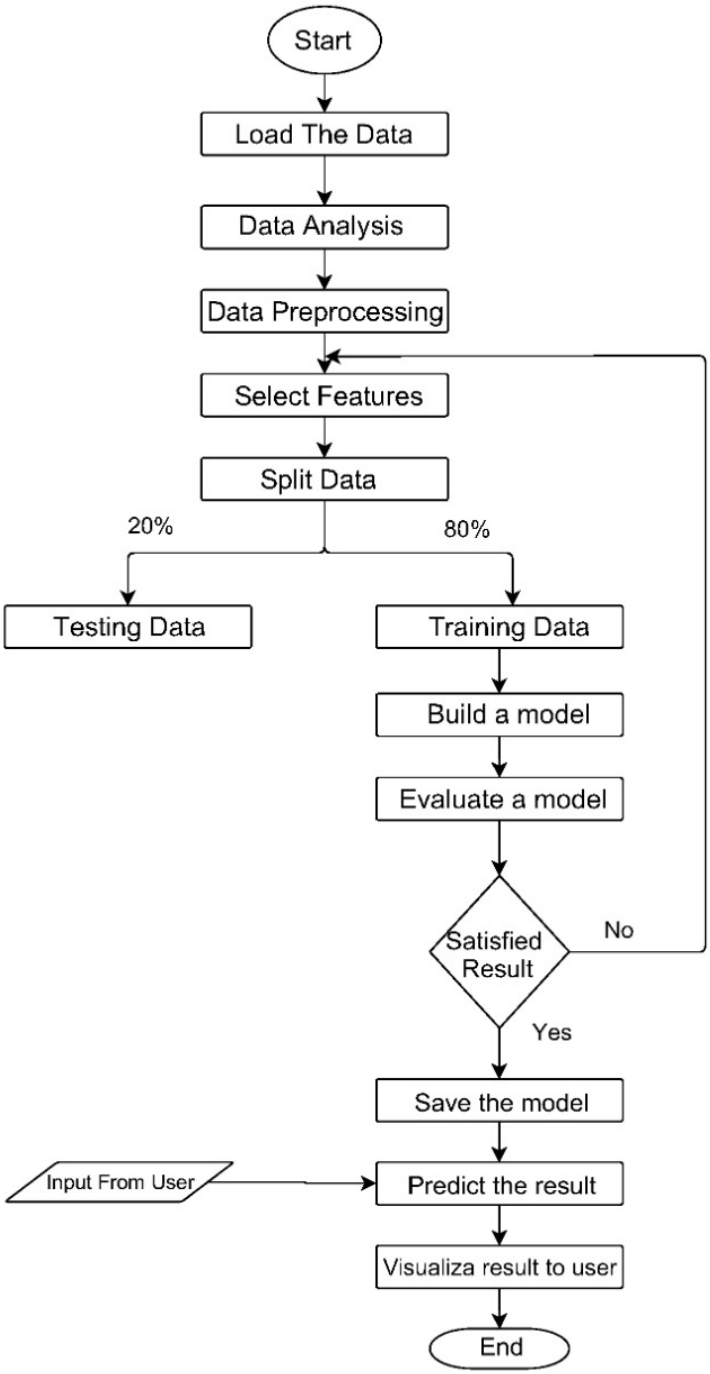
1.4 Aim And objective -

We are going to solve this problem by developing a web interface for end user to predict the fare amount for a taxi ride in New York City on giving the time, number of passenger, pickup and drop-off locations. Predicting fare can help passengers decide when is the optimal time to start their commute, or help drivers decide which of the potential rides will be more profitable.

2.Overall Description

2.1 Workflow of project

This below diagram shows the workflow of project.



2.2 Dataset Description

-- Train.csv

Feature name	Explanation	Variable type
fare_amount Dependent variable (y)	Cost of the taxi ride in dollars. This is the value to be predicted.	Continuous
pickup_datetime	pickup date and time information	Categorical
pickup_longitude	pickup longitude coordinates in decimals	Categorical
pickup_latitude	pickup latitude coordinates in decimals	Categorical
dropoff_longitude	dropoff longitude coordinate in decimals	Categorical
dropoff_latitude	dropoff latitude coordinate in decimals	Categorical
passenger_count	Number of passengers in taxi	Categorical

2.3 Data Sampling and cleaning

2.3.1 Data preprocessing challenge

We are given a training set of 50M Taxi trips in New York 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.

In any analytics project 80% of the time and effort is spent on data cleaning, exploratory analysis and deriving new features. we aim to clean the data, visualize the relationship between variables and also figure out new features that are better predictors of taxi fare.

2.3.2 Methodology

The objective of this project is to predict the fare price of NYC taxi in US. The data set is taken from Kaggle and has 1 csv files named train.csv .

Six major steps involved before feeding data to model

1. Loading the dataset

2. Analysis of data

3. Split data

4. Preprocess data

5. Sample of preprocessed data

6. Merge data

1. Loading the dataset using pandas

For analysis purpose initially we have taken 20M rows .

```
nyct_data  = pd.read_csv(r"Dataset\\raw\\train.csv",nrows=20000000)
```

Python

2. Analysis

The data which we are going to use is to be analyzed first to checkout whether any irrelevant data present inside our dataset which can make our model inaccurate.

2.1. Null values analysis

2.2. The rows which have all the zero values inside every field

2.3. Invalid fare amount

2.4. Invalid locations

2.5. Invalid pickup point

2.6. Invalid drop point

2.7. Outside location

2.8. Invalid passenger

3.Splitting the dataset

After analysis we have split the data year wise .

For 2009 -

```
nyct_data[nyct_data['pickup_datetime'].dt.year == 2009].to_csv(splited_path + "2009_data.csv",index=False)
```

For 2010-

```
nyct_data[nyct_data['pickup_datetime'].dt.year == 2010].to_csv(splited_path + "2010_data.csv",index=False)
```

For 2011-

```
nyct_data[nyct_data['pickup_datetime'].dt.year == 2011].to_csv(splited_path + "2011_data.csv",index=False)
```

For 2012-

```
nyct_data[nyct_data['pickup_datetime'].dt.year == 2012].to_csv(splited_path + "2012_data.csv",index=False)
```

For 2013-

```
nyct_data[nyct_data['pickup_datetime'].dt.year == 2013].to_csv( splited_path + "2013_data.csv",index=False)
```

For 2014-

```
nyct_data[nyct_data['pickup_datetime'].dt.year == 2014].to_csv(splited_path + "2014_data.csv",index=False)
```

For 2015-

```
nyct_data[nyct_data['pickup_datetime'].dt.year == 2015].to_csv(splited_path + "2015_data.csv",index=False)
```

4. Preprocessing data:

Distribution of fare amount:

We performed denoising from the fare amount by removing those fare amount whose values are less than or equal to zero. Since, cost of a trip cannot be negative or zero so we have removed such instances from the data.

Distribution of Trip Distance:

Using the pickup and drop-off coordinates we calculate the trip distance in miles based on Haversine Distance Formula.

Haversine Distance Formula -

The Haversine formula calculates the shortest distance between two points on a sphere using their latitudes and longitudes measured along the surface.

or

$$d = 2rsin^{-1}\left(\sqrt{sin^2\left(\frac{\Phi_2-\Phi_1}{2}\right) + cos(\Phi_1)cos(\Phi_2)sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right)$$

where r is the radius of the earth(6371 km)

d is the distance between two points, , \phi_2 is the latitude of the two points, and \lambda_1, \lambda_2 is the longitude of the two points respectively.

Distribution of Pickup date time :

To Analyze how the fares have changed over time, we need to create features like hour, day of the week, day, month, year from pickup datetime.

Distribution of Passenger Count :

In the New York City we have found that a passenger count for a taxi should not be greater than five. So we removed the data points which is less than or equal to zero and greater than or equal to 5.

5.Sampling Dataset

After performing preprocessing we found that processing such a large amount of data is quite difficult and taking too much time .

So we performed stratified sampling over the train.csv and took the sample size of around 1000000 of each year .

```
try :
    years=[2009,2010,2011,2012,2013,2014,2015]

    for yy in years:
        print(f"\n >> Created sample dataset of {yy}.")
        file=dataset_dir+"\\"+f"filter_dataset_{yy}.csv"
        #load data
        df=pd.read_csv(file).sample(1000000)
        #print(df.columns)
        # create file
        file=sample_dir+"\\"+f"Sample_dataset_of_{yy}.csv"
        # store sample dataset
        df.to_csv(file,index=False)


except Exception as error:
    print("\n Error : ",error)
```

6.Merging

After sampling the data we finally merge the data into a single dataset named final_data.csv in which we have taken around 7000000 row samples in our dataset.

```
try :
    years=[2009,2010,2011,2012,2013,2014,2015]
    df=None
    for yy in years:
        print(f"\n >> load data of {yy}.")
        file=parent_dir+"\\"+f"Sample_dataset_of_{yy}.csv"
        # load data
        if df is not None:
            df1 =  pd.read_csv(file,dtype=dtype)
            df = df.append(df1)
            del df1
            print(f"{yy} merged!!!")
            continue
        df=pd.read_csv(file,dtype=dtype)

    file=final+"\\"+f"finalDataset.csv"
        # store Final merged data
    df.to_csv(file,index=False)
except Exception as error:
    print("\n Error : ",error)
```

2.4 Exploratory Data Analysis:

Exploratory Data Analysis (EDA) is an approach to analyze the data using visual techniques. It is used to discover trends, patterns, or to check assumptions with the help of statistical summary and graphical representations. EDA is an important first step in any data analysis.

Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods. It helps determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions.

It can also help determine if the statistical techniques you are considering for data analysis are appropriate. Originally developed by American mathematician John Tukey in the 1970s, EDA techniques continue to be a widely used method in the data discovery process today.

The main purpose of EDA is to help look at data before making any assumptions. It can help identify obvious errors, as well as better understand patterns within the data, detect outliers or anomalous events, find interesting relations among the variables.

Data scientists can use exploratory analysis to ensure the results they produce are valid and applicable to any desired business outcomes and goals. EDA also helps stakeholders by confirming they are asking the right questions. EDA can help answer questions about standard deviations, categorical variables, and confidence intervals. Once EDA is complete and insights are drawn, its features can then be used for more sophisticated data analysis or modeling, including machine learning.

Dataset Used –

The data used in this project is all subsets of New York City Taxi trip data, which contains observations on around 5 crores taxi rides in New York City between 2009 and 2015. To build the models, a random subset of 70,00,000 observations were used which contains 10,00,000 rows per year.

The original dataset contains features as pickup and drop-off locations, as longitude and latitude coordinates, time and date of pickup , ride fare and passenger count . The data was processed to extract separate features for year, month, day, weekday, hour and minute from the date and time of each ride, as well as trip distance between drop-off and pickup location.

Dependent Variable – Fare_Amount

Independent Variable - pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, passenger_count, year, month, day, weekday, pickup_datetime_hour

Following are some plots we used to extract some useful information:

Explanation – here we can see our dependent variable fare amount have outlier

IQR Based Outlier detection and removing invalid data –

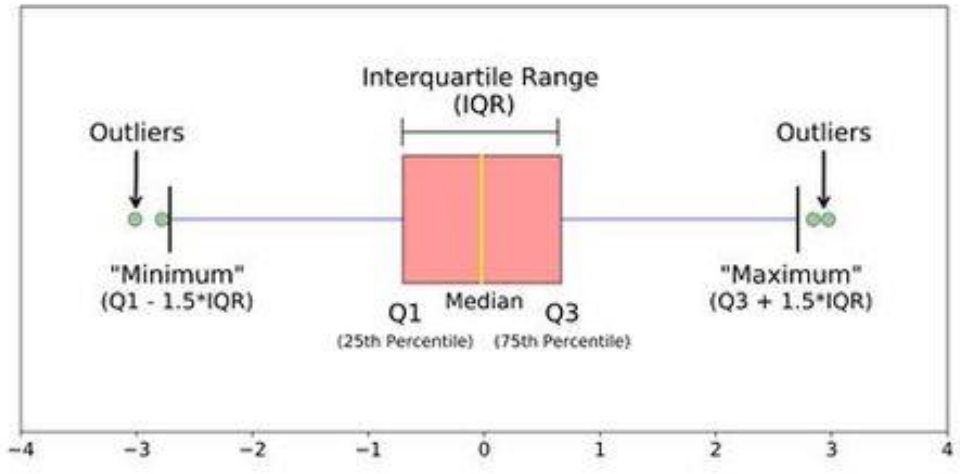
The interquartile range rule is useful in detecting the presence of outliers. Outliers are individual values that fall outside of the overall pattern of a data set. This definition is somewhat vague and

subjective, so it is helpful to have a rule to apply when determining whether a data point is truly an outlier—this is where the interquartile range rule comes in.

Interquartile Range, IQR = Q3 – Q1

Min_value = Q1 – 1.5*IQR ➔ Min Fare Amount = $ 2.5

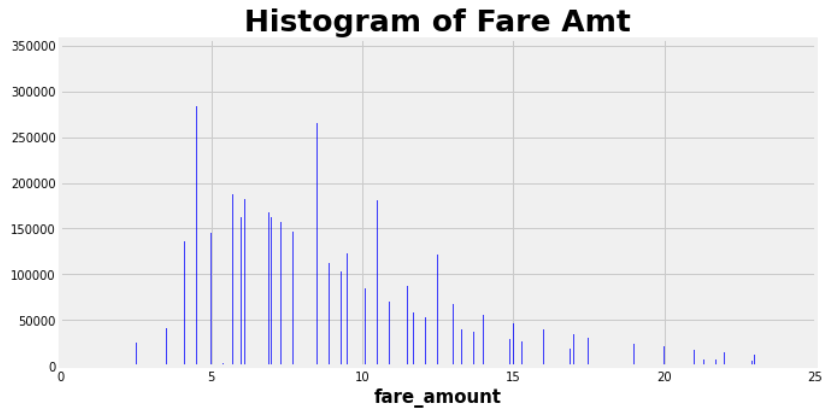
Max_value = Q3 + 1.5*IQR ➔ 12.9 +1.5*(12.9-6) = $ 23.25



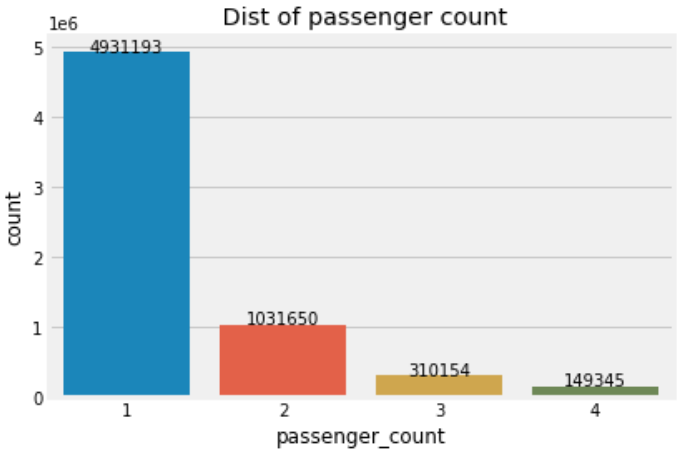
The interquartile range shows how the data is spread about the median. It is less susceptible than the range to outliers and can, therefore, be more helpful.

We can see 5-point analysis of fare_amount using describe function as follows –

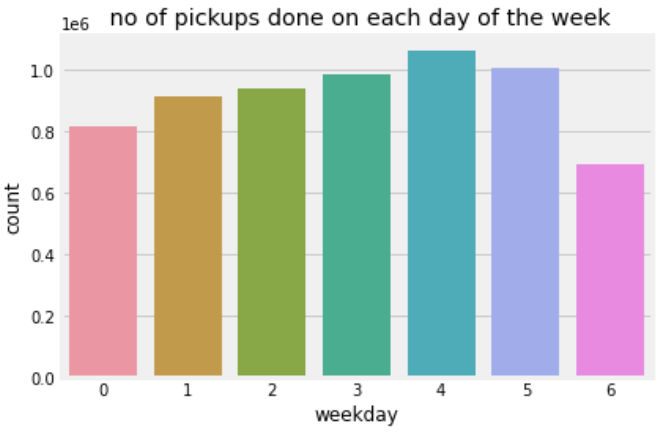
	fare_amount
count	7000000.00000
mean	11.47714
std	43.60017
min	0.01000
25%	6.00000
50%	8.50000
75%	12.90000
max	93963.36000



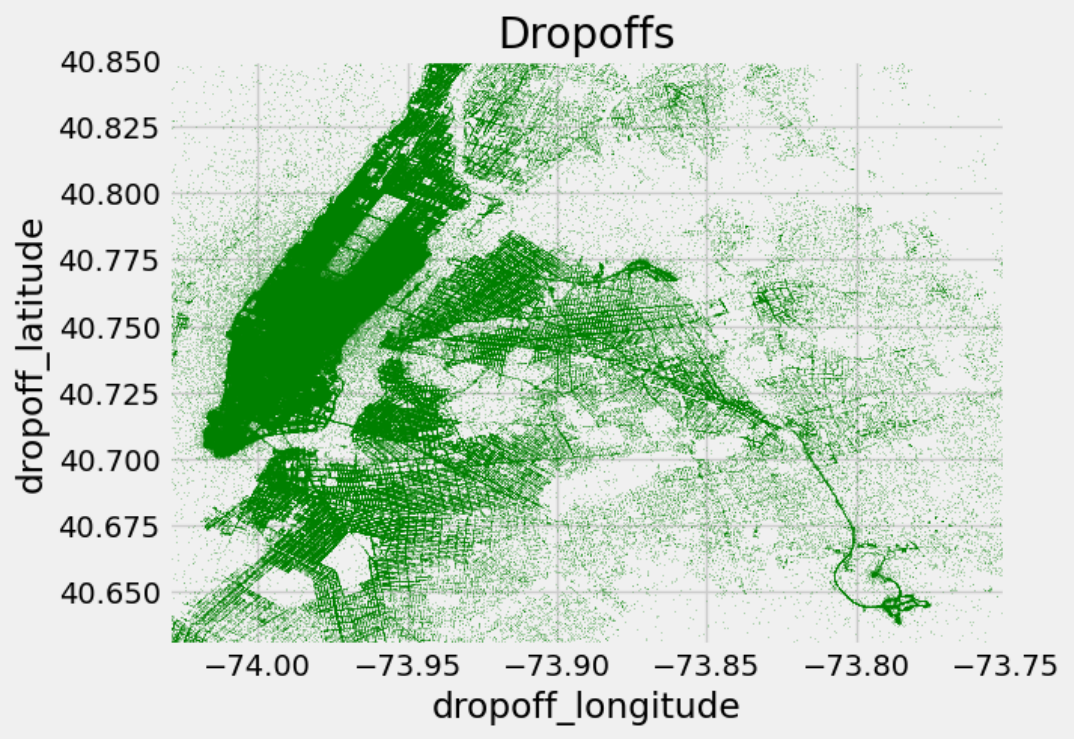
Explanation – Here we can observe that fare_amount is distributed from $2.5 to $23.25



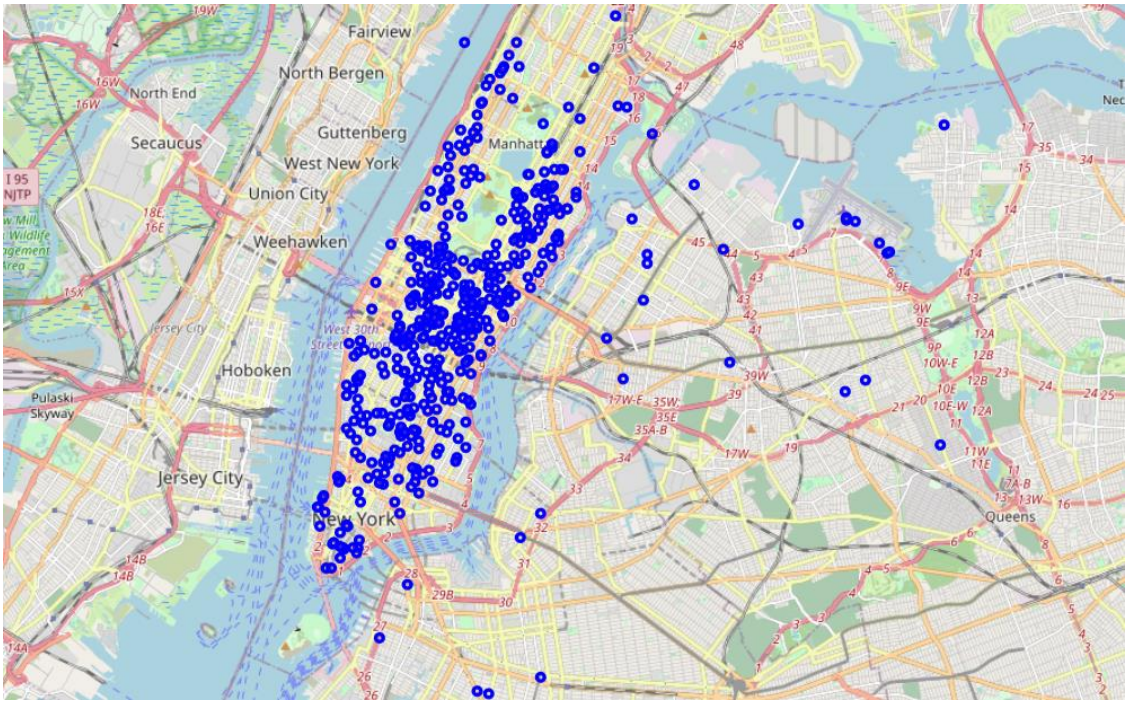
Explanation – Above figure showing that passengers mostly select the solo trip over the grouped trip.



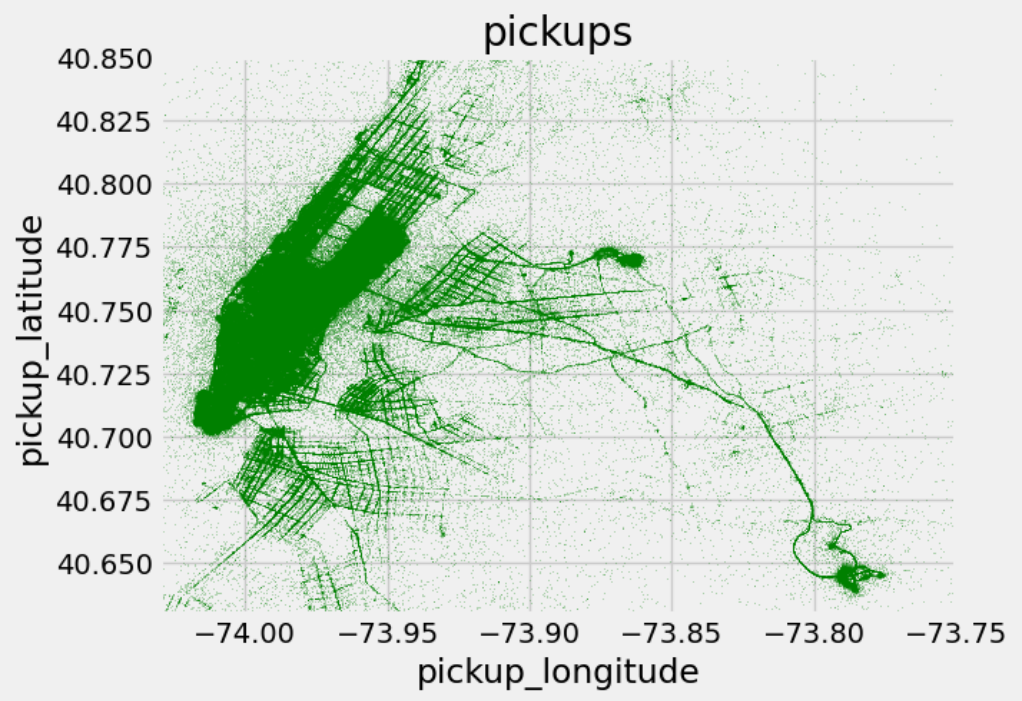
Explanation – Above fig shows the no of pickups done on each day of the week, its clearly shows that on Thursday there are more pickups followed by Friday.



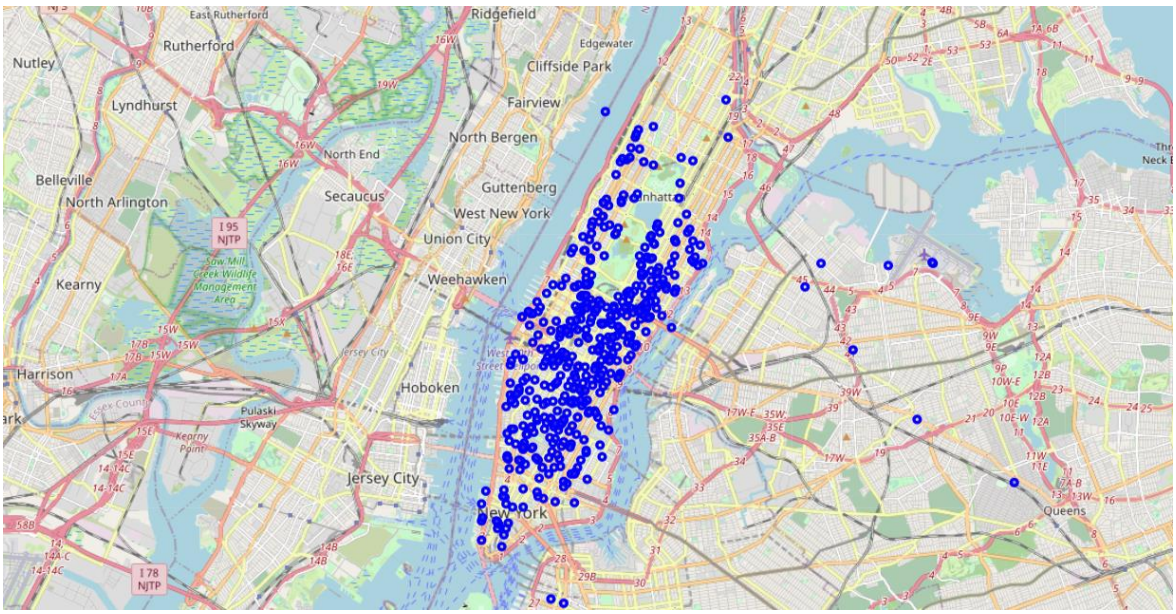
Explanation – This above fig gives idea about the most preferable drop-offs locations by the customer in the New York city.



Explanation – This above fig shows the actual dropoff locations on map for 1st 500 trips from the dataset.



Explanation – This above fig gives idea about the most preferable pickups locations by the customer in the New York city.



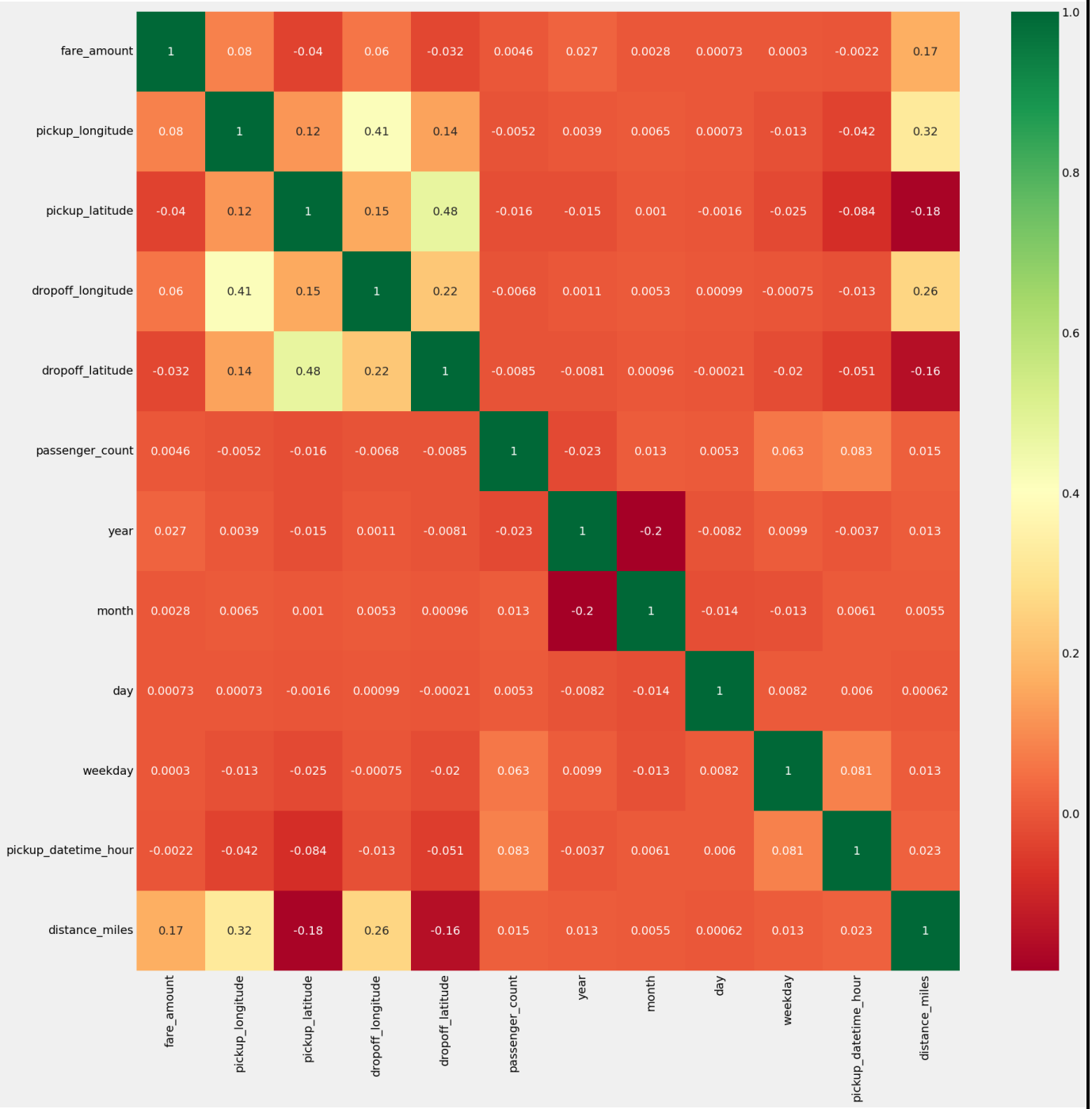
Explanation – This above fig shows the actual pickup locations on map for 1st 500 trips from the dataset.

To calculate the distance between pickup and drop point we create below function and added new column as distance_km in dataset

```
def distance(lat1, lon1, lat2, lon2):
    """ Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)

    All args must be of equal length.    """
    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a =np.sin(dlat/2.0)**2 + np.cos(lat1)*np.cos(lat2)*np.sin(dlon/2.0)**2

    c = 2 * np.arcsin(np.sqrt(a))
    km = 6317 * c
    return km
df_raw['distance_km'] = distance(df_raw.pickup_latitude, df_raw.pickup_longitude,df_raw.dropoff_latitude, df_raw.dropoff_longitude)
```

Using corr() function ,Co-relation between Fare amount and other features are as follows-

	Feature	Corr() coefficient	Correlation in %
Dependent Variable	fare_amount	1.000000	100%
Independent Variables	pickup_longitude	0.080057	8 %
	pickup_latitude	-0.039722	3.9 %
	dropoff_longitude	0.059824	5.9%
	dropoff_latitude	-0.032449	3.2 %
	passenger_count	0.004641	0.4 %
	year	0.027288	2.7 %
	month	0.002797	0.27 %
	day	0.000729	0.072 %
	weekday	0.000301	0.03 %
	pickup_datetime_hour	-0.002159	0.2 %
	distance_km	0.167923	16.79 %



Explanation – Above fig is of heat map it shows the relation between dependent feature and independent features.

This shows that the distance_km has a greater influence on fare amount.

Removing Outlier and invalid data-

As per exploratory data analysis method the fare amount lies outside this (2.5, 23.25) range, it will be considered as a outlier. It must be removed before going for model creation.

Also, passenger count must not be less than or equal to zero and considering the New York city longitude (-75 to 72) and latitude (40 to 42) range, we filtered out invalid rows using below python code-

```
df_raw = df_raw[((df_raw['pickup_longitude'] > -75) & (df_raw['pickup_longitude'] < -72)) &
    ((df_raw['dropoff_longitude'] > -75) & (df_raw['dropoff_longitude'] < -72)) &
    ((df_raw['pickup_latitude'] > 40) & (df_raw['pickup_latitude'] < 42)) &
    ((df_raw['dropoff_latitude'] > 40) & (df_raw['dropoff_latitude'] < 42)) &
    (df_raw['passenger_count'] > 0) &
    ((df_raw['fare_amount'] >=2.5) & (df_raw['fare_amount'] <=23.25))]
```

Now this cleaned data after EDA is used for further machine learning model building for predicting the fare price of taxi.

3. Model Building

Train/ Test Split evaluation –

The train-test split is a technique for evaluating the performance of a machine learning algorithm.

It can be used for classification or regression problems and can be used for any supervised learning algorithm.

The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

- Train Dataset: Used to fit the machine learning model.
- Test Dataset: Used to evaluate the fit machine learning model.

The objective is to estimate the performance of the machine learning model on new data: data not used to train the model.

This is how we expect to use the model in practice. Namely, to fit it on available data with known inputs and outputs, then make predictions on new examples in the future where we do not have the expected output or target values.

The train-test procedure is appropriate when there is a sufficiently large dataset available.

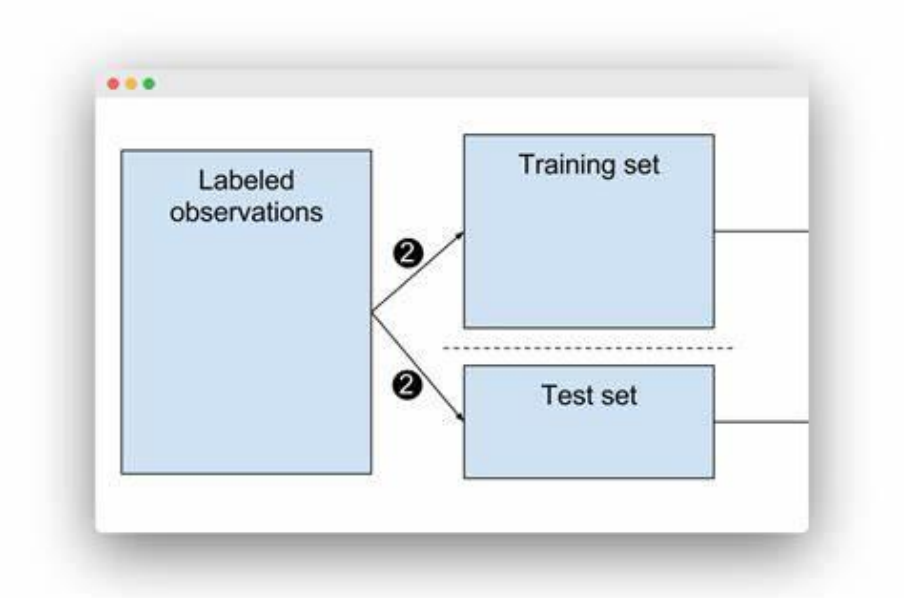
The procedure has one main configuration parameter, which is the size of the train and test sets. This is most commonly expressed as a percentage between 0 and 1 for either the train or test datasets. For example, a training set with the size of 0.67 (67 percent) means that the remainder percentage 0.33 (33 percent) is assigned to the test set.

There is no optimal split percentage. You must choose a split percentage that meets your project's objectives with considerations that include:

- Computational cost in training the model.
- Computational cost in evaluating the model.
- Training set representativeness.
- Test set representativeness.

Nevertheless, common split percentages include:

- Train: 80%, Test: 20%
- Train: 70%, Test: 30%
- Train: 50%, Test: 50%



Advantages of train/test split:

- Model can be trained and tested on different data than the one used for training.
- Response values are known for the test dataset, hence predictions can be evaluated
- Testing accuracy is a better estimate than training accuracy of out-of-sample performance.

In [8]:
```
X= df.drop(['fare_amount','pickup_datetime'],axis=1)
Y= df[['fare_amount']]
```

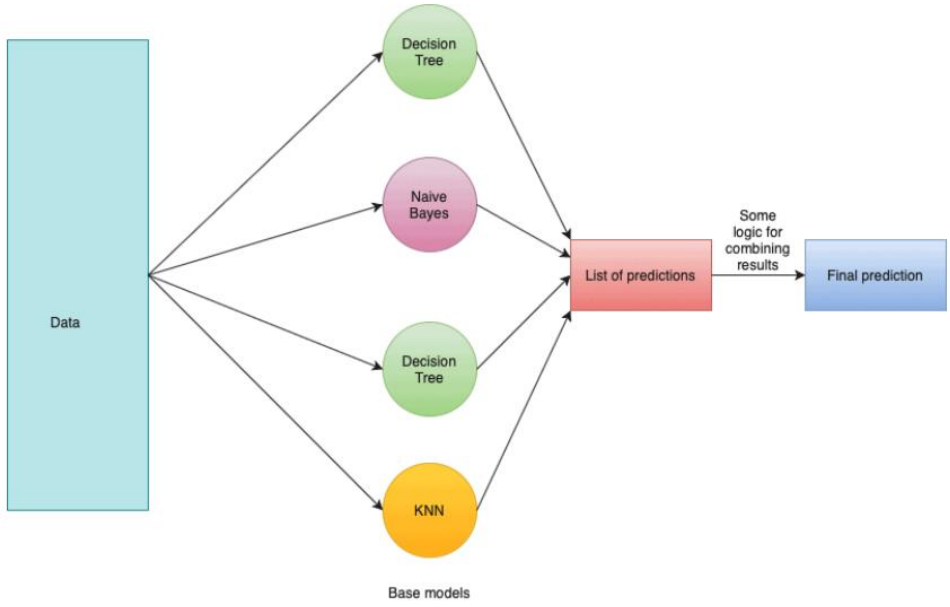
##Train Test split

In [9]:
```
from sklearn.model_selection import train_test_split

X_train,X_test,Y_train,Y_test = train_test_split(X,Y, test_size=0.3, random_state=21)
X_train.shape,X_test.shape,Y_train.shape,Y_test.shape
```

Out[9]: ((4495639, 12), (1926703, 12), (4495639, 1), (1926703, 1))

Ensemble Learning –

The general idea of ensemble learning is quite simple. You should train multiple ML algorithms and combine their predictions in some way. Such an approach tends to make more accurate predictions than any individual model. An Ensemble model is a model that consists of many base models.



So, ensemble learning is a process where multiple ML models are generated and combined to solve a particular problem. In general, ensemble learning is used to obtain better performance results and reduce the likelihood of selecting a poor model.

Overall, Bagging is a nice technique that helps to handle overfitting and reduce variance.

Prediction Models -

Machine learning consists of algorithms that can automate analytical model building. Using algorithms that iteratively learn from data, machine learning models facilitate computers to find hidden insights from Big Data without being explicitly programmed where to look. We have used the following three algorithms to build predictive model.

1. XGBOOST Regression Model
2. Random Forest Regression Model
3. Neural Network for regression

3.1. XGBOOST Regression Model-

Extreme Gradient Boosting (XGBoost) is an open-source library that provides an efficient and effective implementation of the gradient boosting algorithm.

Shortly after its development and initial release, XGBoost became the go-to method and often the key component in winning solutions for a range of problems in machine learning competitions.

Regression predictive modeling problems involve predicting a numerical value such as a dollar amount or a height. XGBoost can be used directly for regression predictive modeling.
In this tutorial, you will discover how to develop and evaluate XGBoost regression models in Python.

- XGBoost is an efficient implementation of gradient boosting that can be used for regression predictive modeling.
- How to evaluate an XGBoost regression model using the best practice technique of repeated k-fold cross-validation.
- How to fit a final model and use it to make a prediction on new data.

XGBoost (Extreme Gradient Boosting) belongs to a family of boosting algorithms and uses the gradient boosting (GBM) framework at its core. It is an optimized distributed gradient boosting library.

Hyper-parameter tunning using Grid Search CV –

There are various hyperparameters used to train a model, and selecting the best hyperparameter plays an important role in getting the best output performance. So, to overcome such problems we use Grid Search.

Scikit-learn's GridSearchCV (where CV stands for Cross Validation) trains the model for different hyperparameters entered by the user and outputs the best-suited hyperparameter for the corresponding training data and the model.

In GridSearchCV approach, machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for best set of hyperparameters from a grid of hyperparameters values.

Python Code – XGBOOST with GridSearchCV –

Step 1 → Importing Required Libraries

```
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
import pickle
```

Step 2 → GridSearchCV

```
xgb = XGBRegressor(random_state=21)
```

```
param_grid = {
'n_estimators' : [30, 50, 70],
'max_depth' :[9,12,14],
'learning_rate':[0.2,0.3]
}
```

```
gscv =GridSearchCV(estimator = xgb, param_grid = param_grid, cv=2, verbose=2)
```

```
gscv.fit(X_train,Y_train)
gscv.best_params_
```

This best_params function gives the best hyperparameter among the given parameter grid which will further used for the model

Step 3 → XGBoost Model

```
xgb = XGBRegressor(random_state=21,max_depth=14,n_estimators=70,learning_rate=0.3)
```

```
xgb.fit(X_train,Y_train)
```

```
XGBRegressor
```

Step 4 → Evaluate the model

```
def eval_fun(model,X_train,Y_train,Y_test,X_test):

    Y_pred_train = model.predict(X_train)
    Y_pred_test = model.predict(X_test)

    r2_train = r2_score(Y_train,Y_pred_train)
    mse_train = mean_squared_error(Y_train,Y_pred_train)
    mae_train = mean_absolute_error(Y_train,Y_pred_train)
    print(f"r2_train_score ==> {r2_train} , mse_train_score ==> {mse_train} , mae_train_score ==> {mae_train}")

    r2_test = r2_score(Y_test,Y_pred_test)
    mse_test = mean_squared_error(Y_test,Y_pred_test)
    mae_test = mean_absolute_error(Y_test,Y_pred_test)
    print(f"r2_test_score ==> {r2_test} , mse_test_score ==> {mse_test} , mae_test_score ==> {mae_test}")
```

```
eval_fun(xgb,X_train,Y_train,Y_test,X_test)
```

```
r2_train_score ==> 0.8992850094416598 , mse_train_score ==> 1.847928362494491 , mae_train_score ==> 0.9060416440308677
r2_test_score ==> 0.8306207582823122 , mse_test_score ==> 3.113190471140647 , mae_test_score ==> 1.1305629783378306
```

```
import pickle

pickle.dump(xgb, open('xgb_model.pkl','wb'))
```

Result –

XGBOOST model giving R2 score for train dataset – 89.92 % and for testing – 83.06%

3.2. Random Forest Regression model –

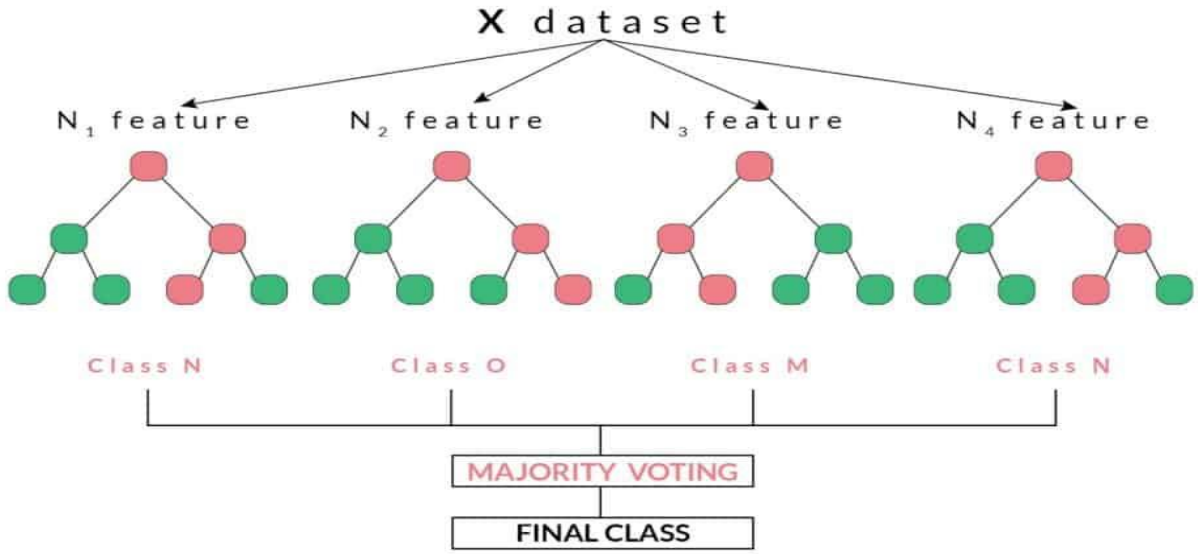
Random Forest is a Supervised learning algorithm that is based on the ensemble learning method and many Decision Trees. Random Forest is a Bagging technique, so all calculations are run in parallel and there is no interaction between the Decision Trees when building them. RF can be used to solve both Classification and Regression tasks.

The name "Random Forest" comes from the Bagging idea of data randomization (Random) and building multiple Decision Trees (Forest). Overall, it is a powerful ML algorithm that limits the disadvantages of a Decision Tree model.

To make things clear let's take a look at the exact algorithm of the Random Forest:

1. So, you have your original dataset D, you want to have K Decision Trees in our ensemble. Additionally, you have a number N – you will build a Tree until there are less or equal to N samples in each node (for the Regression, task N is usually equal to 5). Moreover, you have a number F – number of features that will be randomly selected in each node of the Decision Tree. The feature that will be used to split the node is picked from these F features (for the Regression task, F is usually equal to sqrt(number of features of the original dataset D)
2. Everything else is rather simple. Random Forest creates K subsets of the data from the original dataset D. Samples that do not appear in any subset are called "out-of-bag" samples.
3. K trees are built using a single subset only. Also, each tree is built until there are fewer or equal to N samples in each node. Moreover, in each node F features are randomly selected. One of them is used to split the node
4. K trained models form an ensemble and the final result for the Regression task is produced by averaging the predictions of the individual trees

In the picture below you might see the Random Forest algorithm for Classification.



Python Code – Random Forest with GridSearchCV –

Step 1→ Importing Required Library

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
```

Step 2 → Tunning with GridSearchCV

```
param_grid_rf = {
'n_estimators' : [50, 70],
'max_depth' :[12,14],
}
```

```
rf = RandomForestRegressor(random_state=21)
gscv =GridSearchCV(estimator = rf, param_grid = param_grid_rf, cv=2, verbose=2)
```

```
gscv.fit(X_train,Y_train)
```

This best_params function gives the best hyperparameter among the given parameter grid which will further used for the model.

Step 3 → Random Forest Model

```
rf1 = RandomForestRegressor(n_estimators=70,max_depth=14,oob_score=True,random_state=21,verbose = 2)
```

```
rf1.fit(X_train,Y_train)
```

Step 4 → Evaluate the model

```
eval_fun(rf1,X_train,Y_train,Y_test,X_test)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.9s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:  2.4min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.9s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:  1.0min finished
r2_train_score ==> 0.8468981923225898 , mse_train_score ==> 2.809126736623959 , mae_train_score ==> 1.1449012361444413
r2_test_score ==> 0.8059844485872251 , mse_test_score ==> 3.5660058445537026 , mae_test_score ==> 1.2631481416769716
```

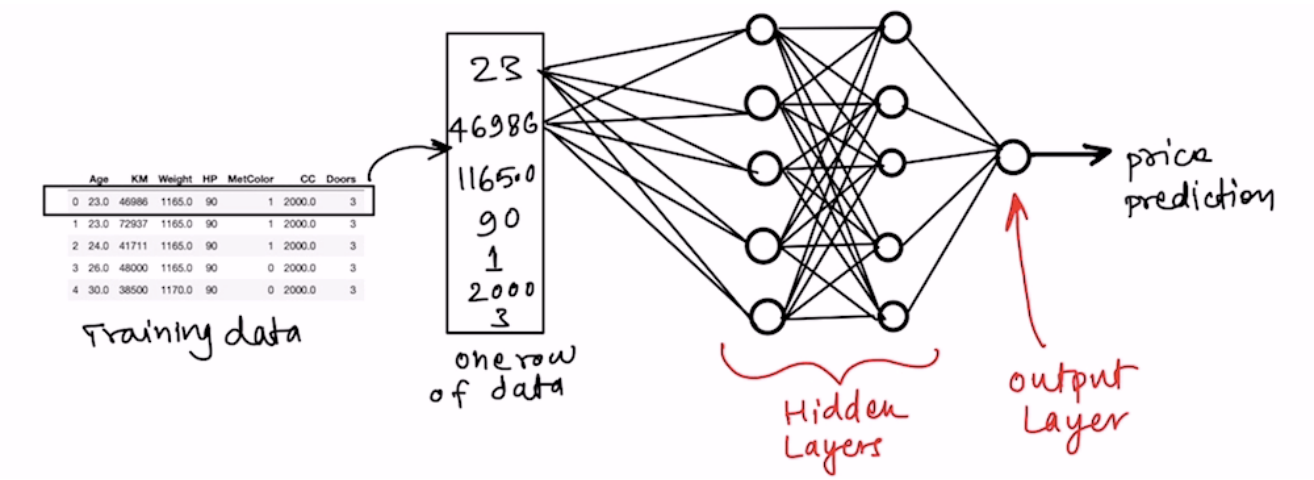
Step 5 → Result

Random Forest model giving R2 score for train dataset – 84.68 % and for testing – 80.59%

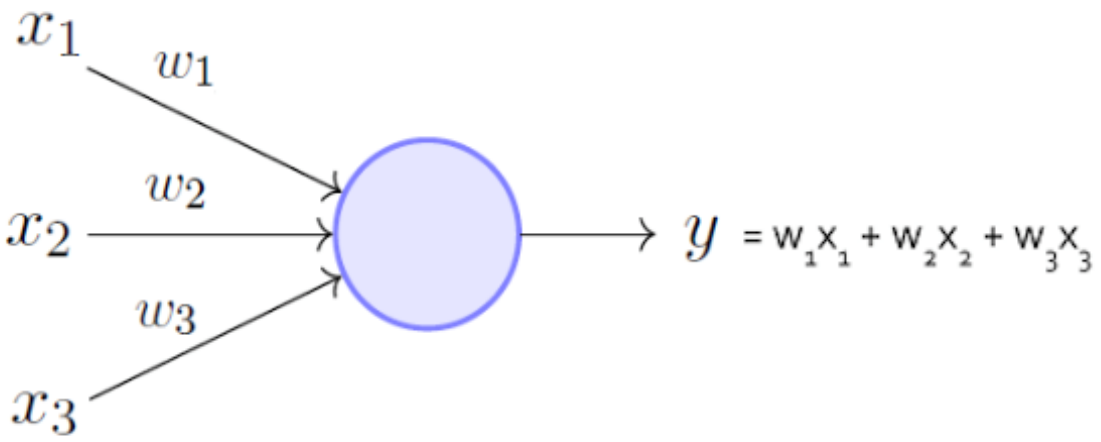
3.3. Artificial Neural Network

Artificial Neural Network (ANN) is probably the first stop for anyone who enters into the field of Deep Learning. Inspired by the structure of Natural Neural Network present in our body, ANN mimics a similar structure and learning mechanism.

ANN is just an algorithm to build an efficient predictive model. Because the algorithm and so its implementation resembles a typical neural network, it is named so.



The structure of a perceptron can be visualized as below:



The process of producing outputs, calculating errors, feeding them back again to produce a better output is generally a confusing process..

KerasTuner-

KerasTuner is an easy-to-use, scalable hyperparameter optimization framework that solves the pain points of hyperparameter search. Easily configure your search space with a define-by-run syntax, then leverage one of the available search algorithms to find the best hyperparameter values for your models. KerasTuner comes with Bayesian Optimization, Hyperband, and Random Search algorithms built-in, and is also designed to be easy for researchers to extend in order to experiment with new search algorithms.

Python Code – Neural Network Model with Keras Tuner –

Step 1 → Importing Required Libraries.

```
!pip install -q -U keras-tuner
```

```
from tensorflow import keras
from tensorflow.keras import layers
from kerastuner.tuners import RandomSearch
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

Step 2 → Keras Model Tunning –

```
def build_model(hp):
    model = keras.Sequential()
    for i in range(hp.Int('num_layers', 2, 20)):
        model.add(layers.Dense(units=hp.Int('units_' + str(i),
                                            min_value=32,
                                            max_value=512,
                                            step=32),
                               activation='relu'))
    model.add(layers.Dense(1, activation='linear'))
    model.compile(
        optimizer=keras.optimizers.Adam(
            hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])),
        loss='mean_absolute_error',
        metrics=['mean_absolute_error'])
    return model
```

```
tuner = RandomSearch(
    build_model,
    objective='val_mean_absolute_error',
    max_trials=5,
    executions_per_trial=1,
    directory='project',
    project_name='nyc Project')
```

```
tuner.search_space_summary()
```

Step 3 → Best Parameters

```
tuner.search(X_train, Y_train,
             epochs=5,
             validation_data=(X_test, Y_test))
```

```
tuner.results_summary()
```

```
Results summary
Results in project/nyc Project
Showing 10 best trials
<keras_tuner.engine.objective.Objective object at 0x7fef45d08b50>
Trial summary
Hyperparameters:
num_layers: 5
units_0: 128
units_1: 128
learning_rate: 0.001
units_2: 192
units_3: 320
units_4: 224
units_5: 448
units_6: 32
units_7: 384
units_8: 160
units_9: 352
Score: 1.5258374214172363
```

Step 4 → ANN model

```
n_features = X_train.shape[1]
n_features
```

```
11
```

```
model1 = Sequential()
model1.add(Dense(11, activation='relu', input_shape=(n_features,)))
model1.add(Dense(128, activation='relu'))
model1.add(Dense(128, activation='relu'))
model1.add(Dense(192, activation='relu'))
model1.add(Dense(320, activation='relu'))
model1.add(Dense(224, activation='relu'))
model1.add(Dense(448, activation='relu'))
model1.add(Dense(32, activation='relu'))
model1.add(Dense(384, activation='relu'))
model1.add(Dense(160, activation='relu'))
model1.add(Dense(352, activation='relu'))
model1.add(Dense(1, activation='linear'))
```

```
model1.summary()
```

Step 5 → Fitting the model

```
model1.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

```
from tensorflow.keras.callbacks import EarlyStopping
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=30)
```

```
history = model1.fit(X_train, Y_train, epochs=10, batch_size=32, validation_split=0.3, verbose=1)#,callbacks=[es])
```

```
Epoch 1/10
98343/98343 [==============================] - 882s 9ms/step - loss: 6.3804 - mae: 1.8098 - val_loss: 6.0004 - val_mae: 1.
7070
Epoch 2/10
98343/98343 [==============================] - 885s 9ms/step - loss: 5.8857 - mae: 1.7398 - val_loss: 6.2663 - val_mae: 1.
```

Step 6 → Model Evaluation

```
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
def eval_fun(model,X_train,Y_train,Y_test,X_test):

    Y_pred_train = model.predict(X_train)
    Y_pred_test = model.predict(X_test)

    r2_train = r2_score(Y_train,Y_pred_train)
    mse_train = mean_squared_error(Y_train,Y_pred_train)
    mae_train = mean_absolute_error(Y_train,Y_pred_train)
    print(f"r2_train_score ==> {r2_train} , mse_train_score ==> {mse_train} , mae_train_score ==> {mae_train}")

    r2_test = r2_score(Y_test,Y_pred_test)
    mse_test = mean_squared_error(Y_test,Y_pred_test)
    mae_test = mean_absolute_error(Y_test,Y_pred_test)
    print(f"r2_test_score ==> {r2_test} , mse_test_score ==> {mse_test} , mae_test_score ==> {mae_test}")
```

```
eval_fun(model1,X_train,Y_train,Y_test,X_test)
```

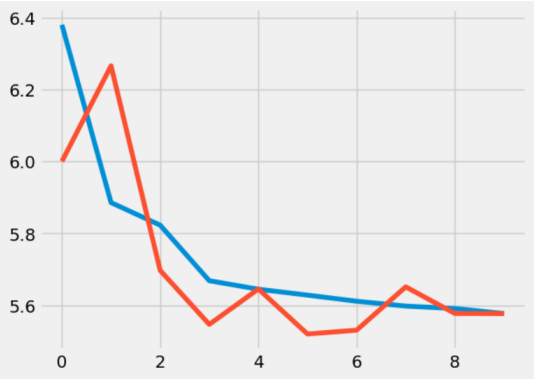
```
140489/140489 [==============================] - 483s 3ms/step
60210/60210 [==============================] - 532s 9ms/step
r2_train_score ==> 0.6951253645021067 , mse_train_score ==> 5.593869222629566 , mae_train_score ==> 1.7316769581589837
r2_test_score ==> 0.6951710944770294 , mse_test_score ==> 5.602755298574723 , mae_test_score ==> 1.7322260146082025
```

In [16]:
```
loss, acc = model1.evaluate(X_test, Y_test, verbose=0)
print('Test Accuracy: %.3f' % acc)
```

```
Test Accuracy: 1.732
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x2669a2724d0>]
```



3.4 Model Comparison –

	Train			Test		
Model	R2 score	MSE	MAE	R2 score	MSE	MAE
XGBOOST	0.89	1.847	0.906	0.83	3.314	1.13
Random Forest	0.846	2.809	1.14	0.8059	3.566	1.263
ANN	0.69	5.5938	1.73	0.69	5.5938	1.73

The XGBoost model outperform all other models used, as it manages to score high on training and testing dataset.

XGBoost model giving higher prediction accuracy of 96 % for train dataset and 82 % for test dataset.

So, we concluded that this model is more efficient and reliable in predicting the fare price.

4. User Interface and Deployment

4.1. Streamlit API

GUI is made using Streamlit. Streamlit is an open-source app framework in Python language. It helps us create web apps for data science and machine learning in a short time.

It is compatible with major Python libraries such as scikit-learn, Keras, PyTorch, SymPy(latex), NumPy, pandas, Matplotlib etc. With Streamlit, no callbacks are needed since widgets are treated as variables. Data caching simplifies and speeds up computation pipelines. Streamlit turns data scripts into shareable web apps in minutes. All in pure Python. No front-end experience required.

We can instantly develop web apps and deploy them easily using Streamlit. Streamlit allows you to write an app the same way you write a python code.

Installing Streamlit

1. Make sure you have python installed in your system

2. Use the following command to install streamlit,

pip install streamlit

Running a streamlit app

First, you create a python script with streamlit commands and execute the script using the following command,

streamlit run <yourscript.py>

You can see in the user interface there are five user inputs columns :

1.Pickup point

2.Dropoff Point

3.Booking Date

4.Booking time

5.Passanger count

We have set some validation rules like there must be only one pickup and dropoff point to be entered . If user may input two pickup point or dropoff point simultaneously then it will send a message to the user that “Please Select only one pickup and dropoff point”.

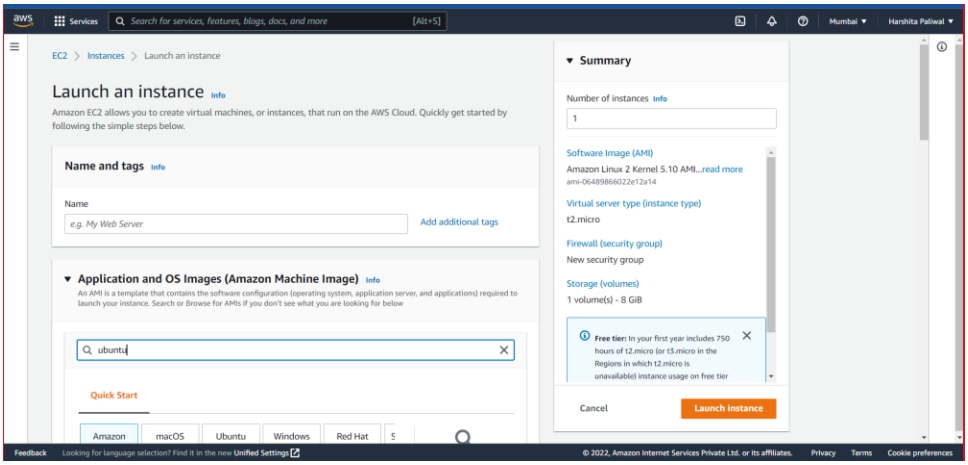
Also the booking date Calendar is showing from the current time onwards so that user can select the current date or future date .

Similarily the same is done for Booking time also .

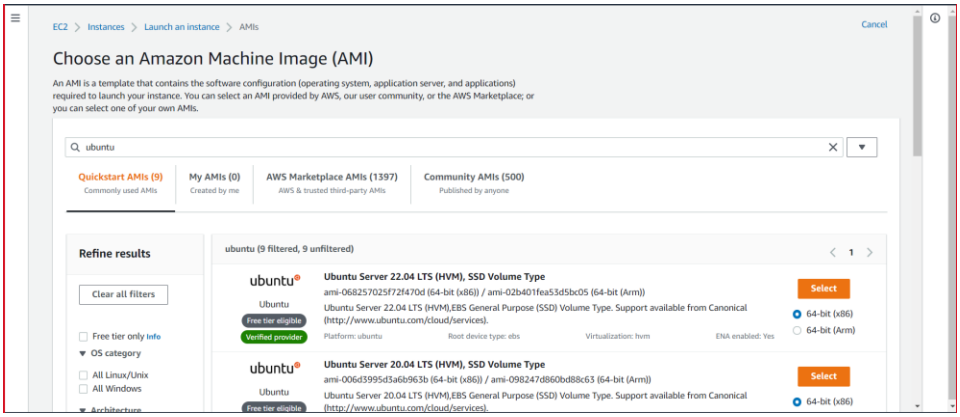
For Passenger count user are allowed to enter the number of passenger upto five only.

4.2. Deployment -

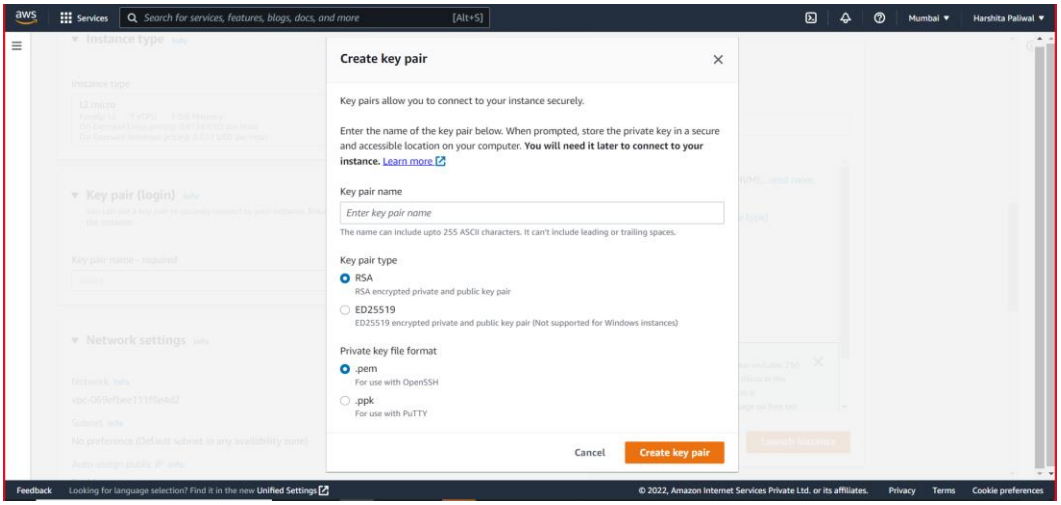
1. Creating a new EC2 instance



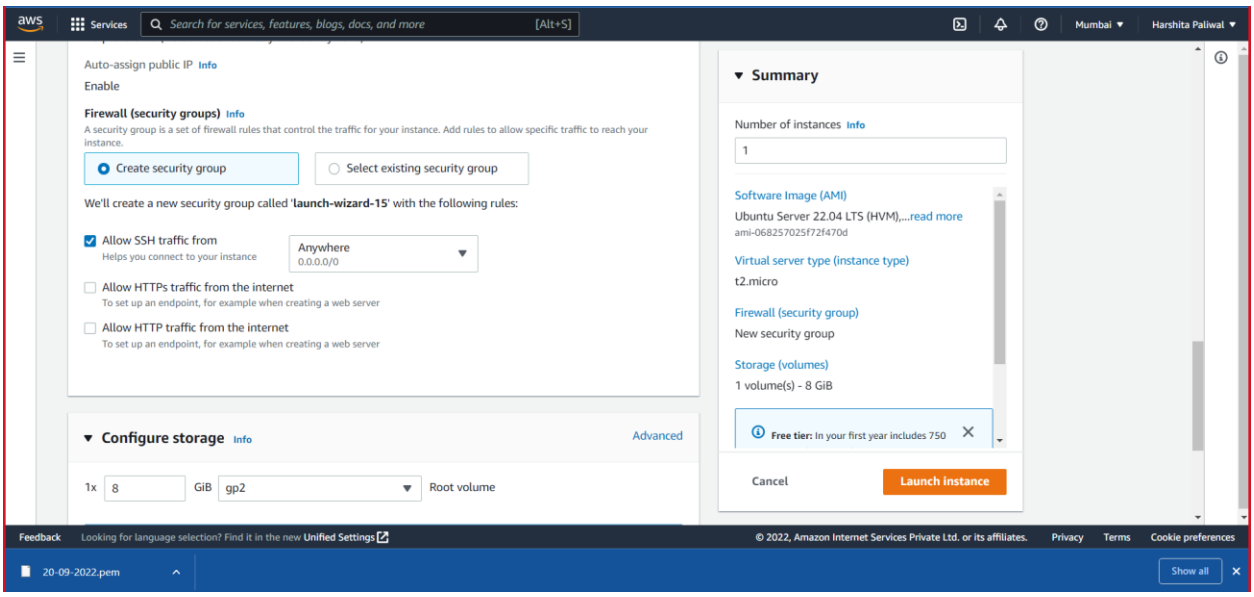
2.Choosing an AMI => ubuntu (free tier eligible)

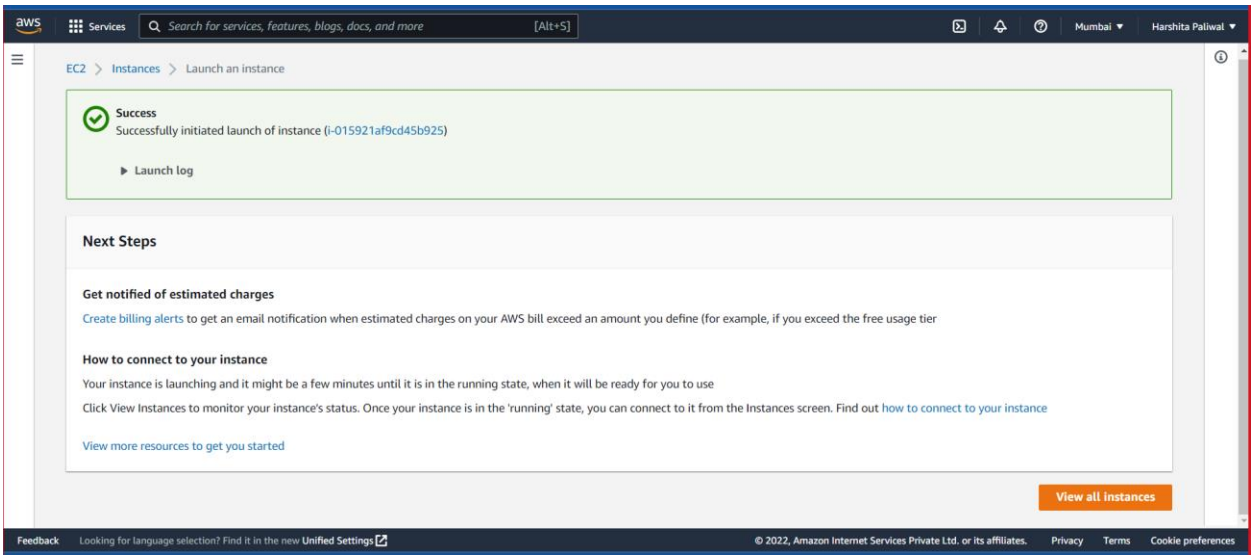


3. Creating Key-Pair

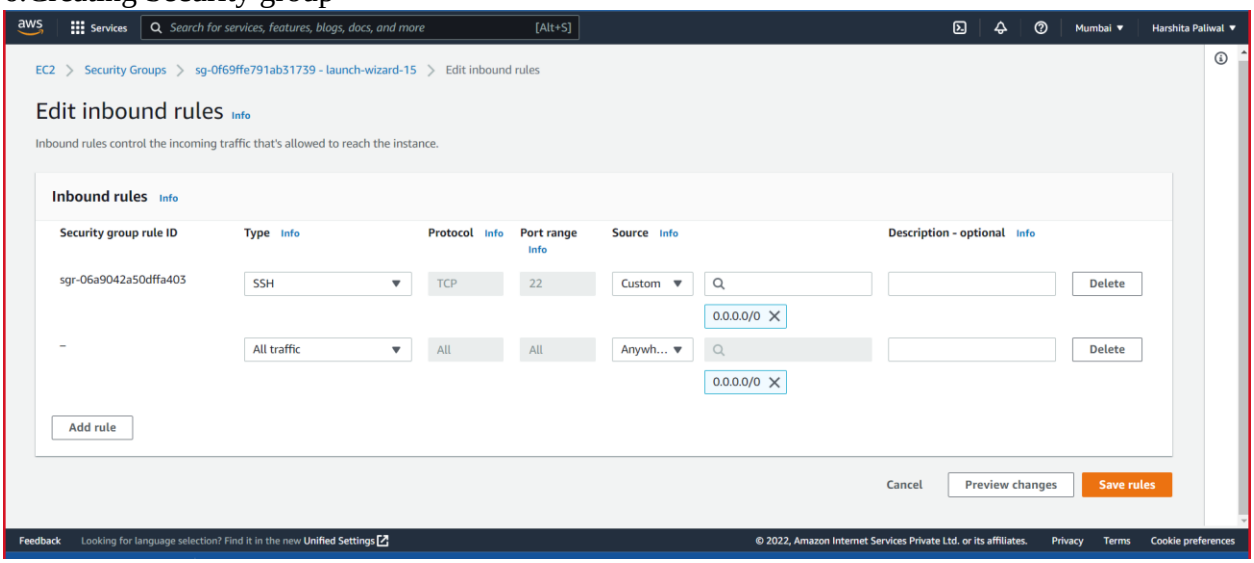


4.Launch Instance

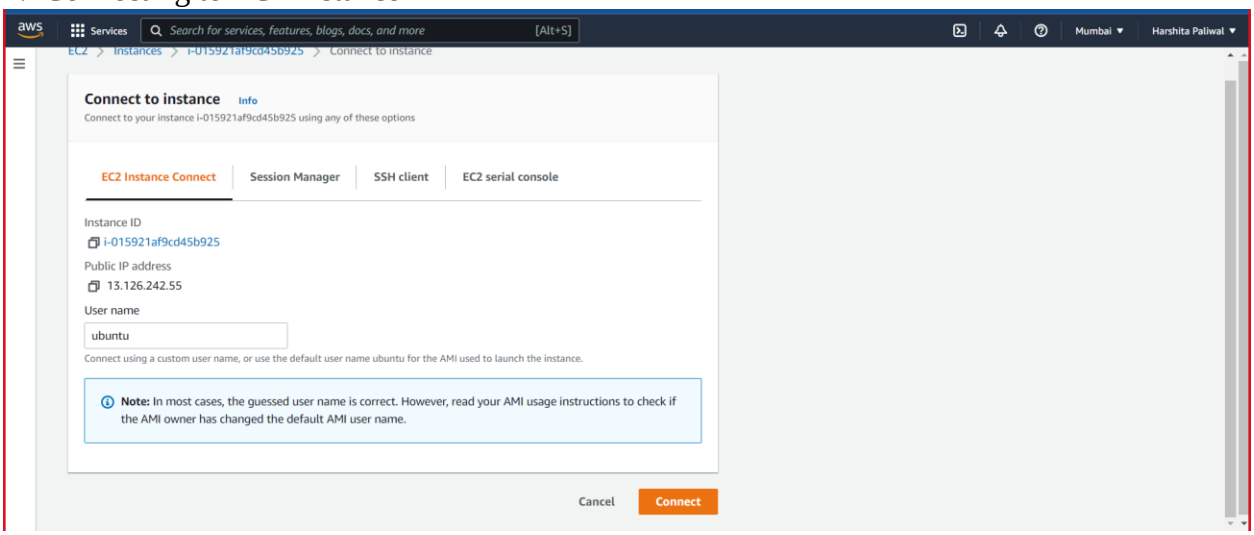


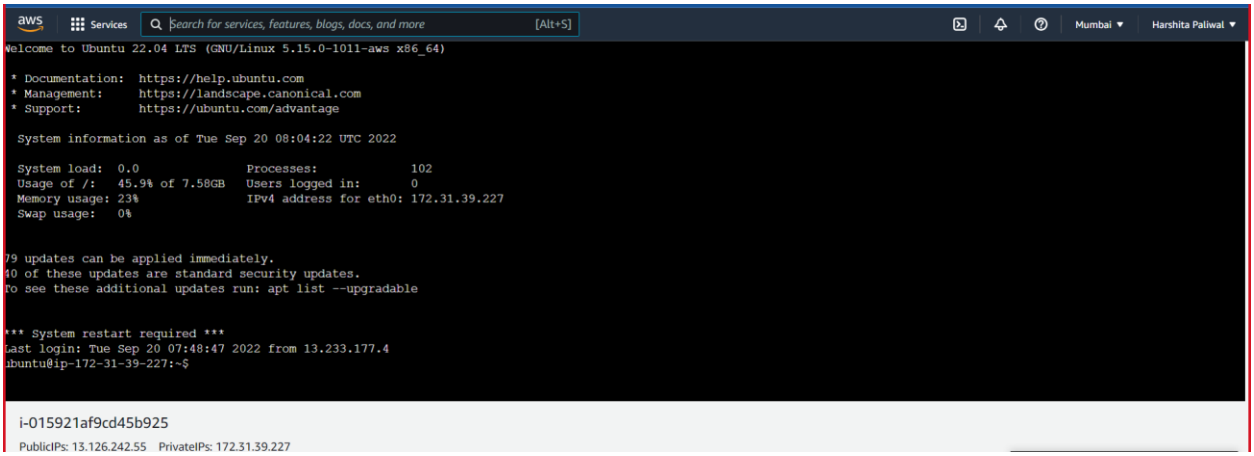


6.Creating Security group

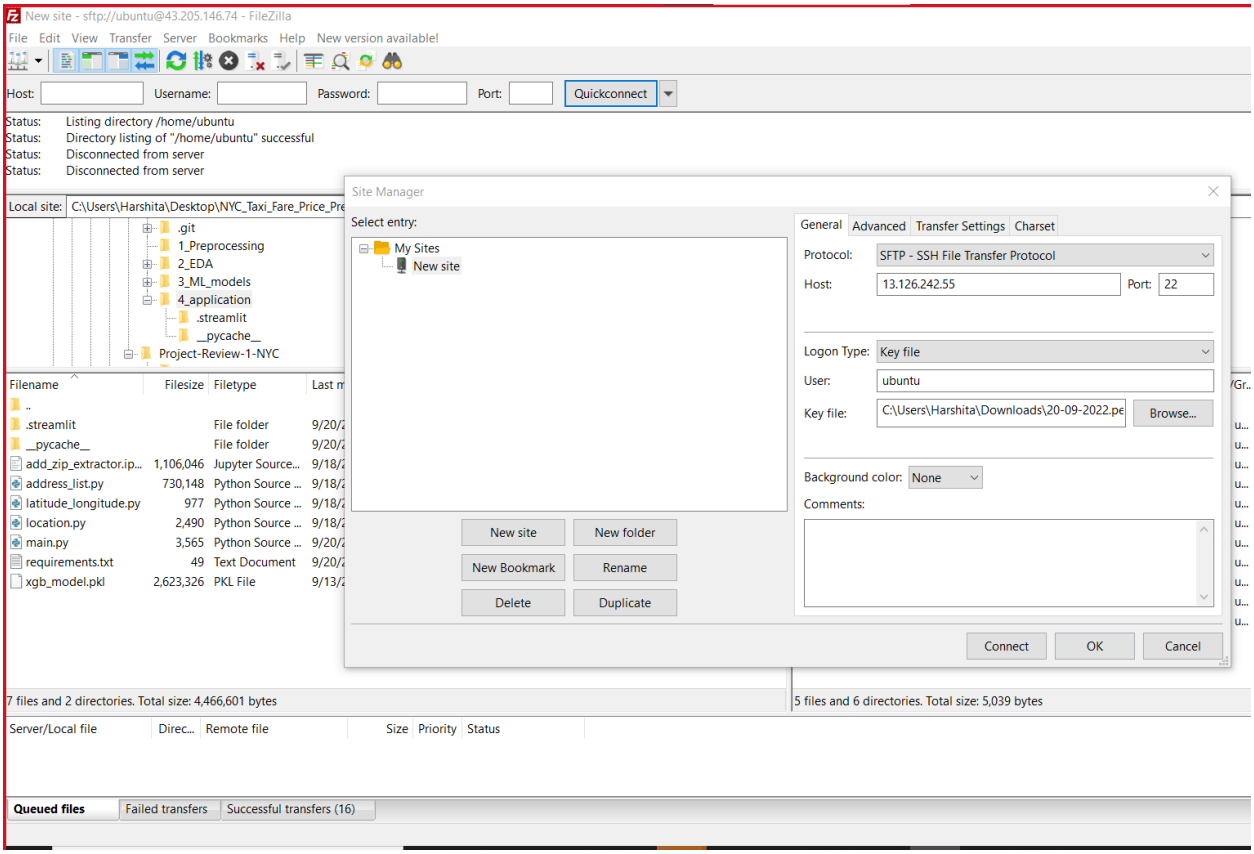


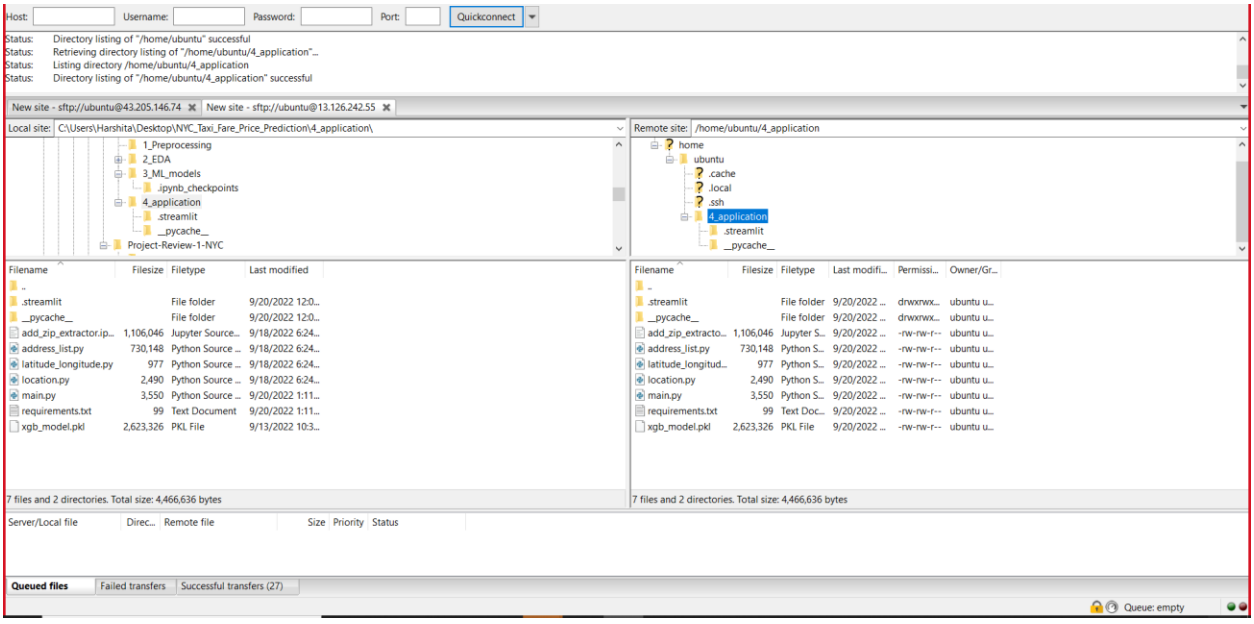
7. Connecting to EC2 instance





10.Transferring files from local to Aws machine





11. Library setup

ubuntu@ip-172-31-39-227:~$ sudo apt-get update

ubuntu@ip-172-31-39-227:~$ sudo apt-get install python3

ubuntu@ip-172-31-39-227:~$ sudo apt-get install python3-pip

ubuntu@ip-172-31-39-227:~$ pip3 install --upgrade pip

ubuntu@ip-172-31-39-227:~/4_application$ cat requirements.txt

pandas

scikit-learn

streamlit

numpy

streamlit_option_menu

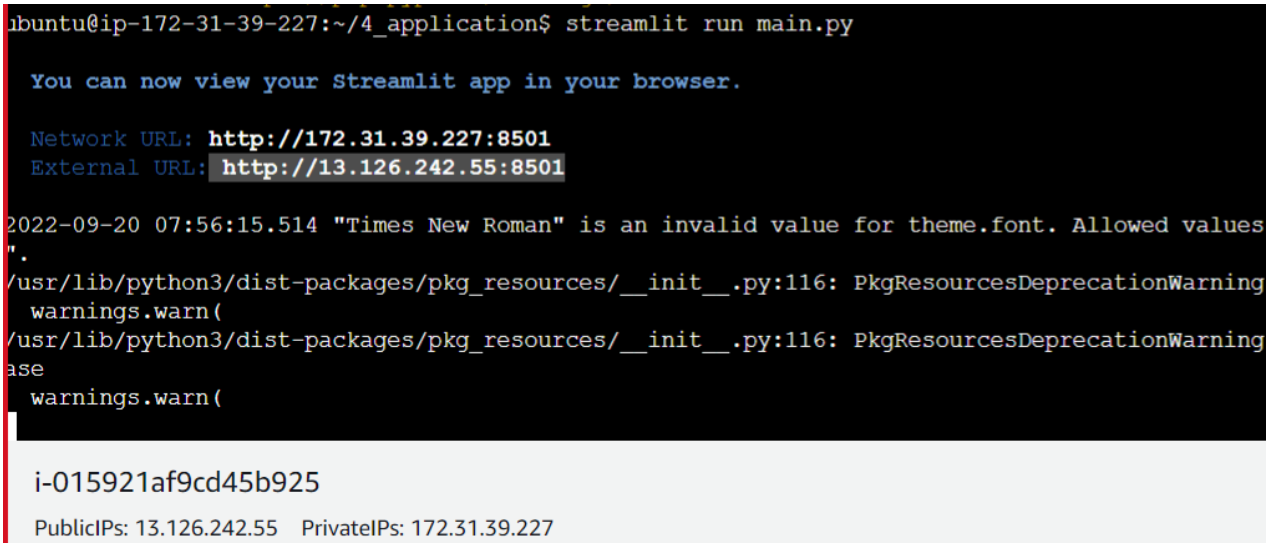
streamlit.components.v1

ubuntu@ip-172-31-39-227:~/4_application$ pip3 install -r requirements.txt

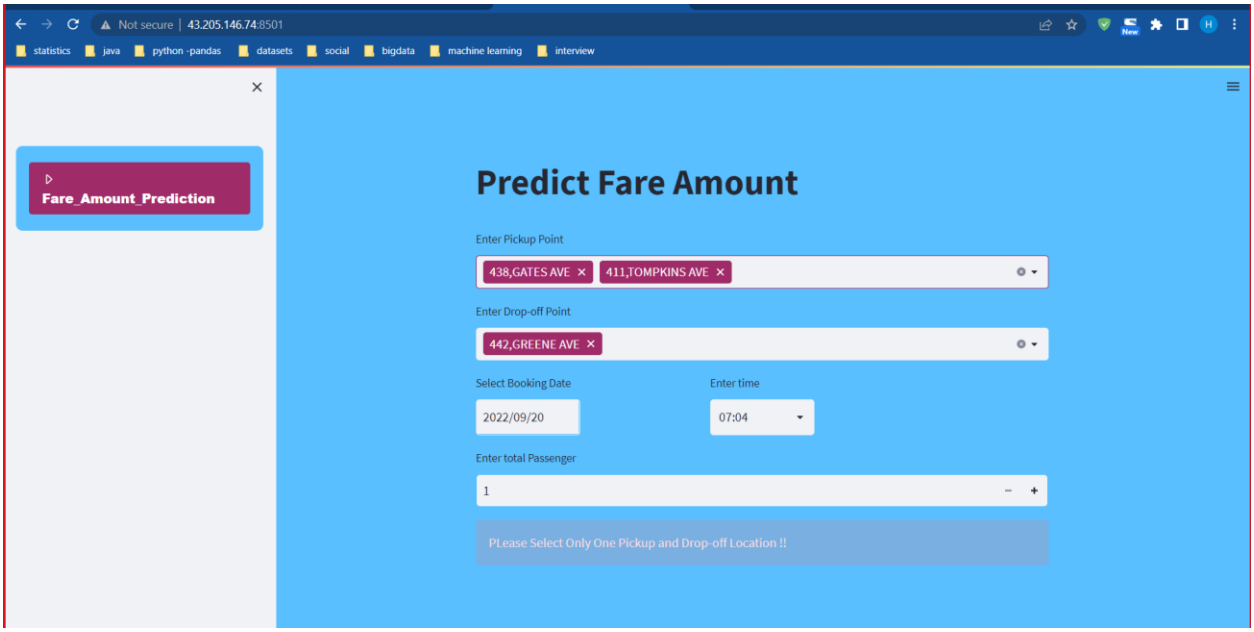
ubuntu@ip-172-31-39-227:~/4_application$ sudo pip3 install xgboost

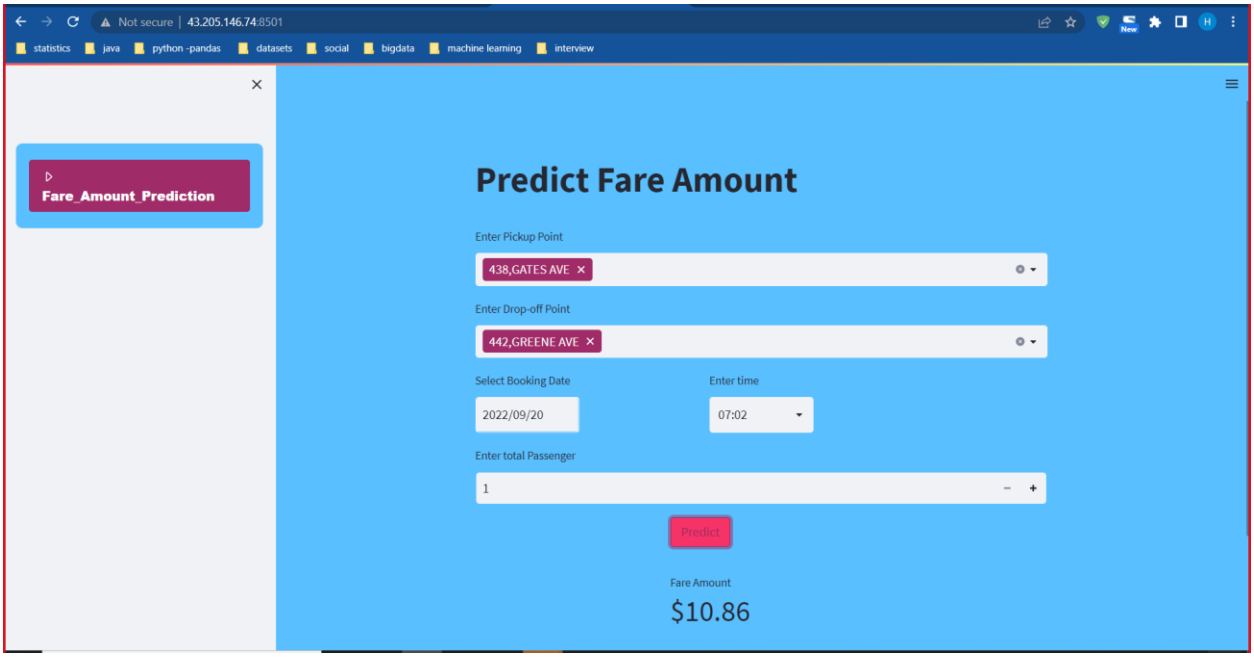
ubuntu@ip-172-31-39-227:~/4_application$ streamlit run main.py

12.Running Streamlit application



13. User Interface after accessing it with public IP address





5. Requirements Specification

5.1 Hardware Requirement:

- 500 GB hard drive (Minimum requirement)
- 16 GB RAM (Minimum requirement)
- CPU core i5 7gen (minimum requirement)

5.2 Software Requirement:

- Windows/Mac/Linux
- Python-3.10.6
- VS Code/Anaconda/Spyder/Jupyter-Notebook
- Python Extension for VS Code
- EC2 instance of AWS cloud platform

5.3 Libraries:

- NumPy 1.23.2
- Pandas 1.4.4
- Matplotlib 3.5.3
- Scikit-learn 1.1.2
- Streamlit
- Xgboost

- Any Modern Web Browser like Google Chrome-
  To access the web application written in Streamlit

6. Future Work:

As a part of the future work, we will be going to implement more features in User interface such as:
User can book a cab for travelling across New York city.
After Booking a cab it will get notification via message or email about the predicted fare amount and the time required to travel.

We will integrate google map API through which a user can track the path where he wants to go. At last, in the model enhancement part, the enhancements to the K-Means Clustering algorithm could be provided by encompassing additional features such as distance to the closest metro station, number of restaurants and eateries in a given zone, etc.

As a further extension we are trying to reduce the training time of regression model using PySpark, so that we can able to train the model with minimal time and higher accuracy.

7.Conclusion:

Once we cleaned our data, removed outliers, removed errors, and did featured engineering also for pickup location and drop-off location to get the distance between two points, afterwards we implemented our data into a Machine Learning Model.

We used various Machine Learning Models such as Linear regression, Random Forest, XGBoost and ANN with hyperparameters tunning models like GridSearchCV and Keras model to get high prediction accuracy.

We are successfully able to implement various algorithms on the New York City Taxi Trip Duration dataset and able to draw certain conclusions from several inferences.

We found that XGBoost is outperforming other algorithm as it shows a slightly better accuracy comparing to other models.

This in turn helps to conclude that XGBoost Model is more efficient and reliable as it predicting fairly accurate result.

To further improve the prediction accuracy, more variabilities need to be considered and modeled such as drop-off time etc.

8. General Closure Report

8.1 General Project Information

	Description
Project Name	New York city taxi fare price prediction
Project Description	Web interface developed using streamlit API able to predict fare price between given locations, date and time
Project Domain	Machine Learning, Python, AWS
Prepared By	Harshita Paliwal Akash Hingane

8.2 Project Closure report stagewise

Sr. No	**Stage**	**Start Date**	**Finish Date**	**Description**
1	Preprocessing	03-09-2022	07-09-2022	Basic data analysis, sampling, Exploratory data analysis
2	Regression Model	08-09-2022	18-09-2022	Model training, evaluation and comparison
3	Streamlit API	19-09-2022	22-09-2022	Backend integration with user interface
4	AWS cloud deployment	23-09-2022	25-09-2022	Deployment of application on AWS EC2 instance

8.References:

- https://www.kaggle.com/competitions/new-york-city-taxi-fare-prediction/data
- https://www.geeksforgeeks.org/
- https://pandas.pydata.org/pandas-docs/version/0.23/api.html
- https://www.youtube.com/
- https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
- https://streamlit.io/
- https://www.youtube.com/watch?v=UN4DaSAZel4&list=PLuU3eVwK0I9PT48ZBYAHdKPFazhXg76h5
- https://pypi.org/project/keras-tuner/