

**Contents**

**1. Introduction**

[1.1 Problem Statement](#_Toc29919) 3

[1.2 Dataset](#_Toc29920) 3

[1.3 Exploratory Data Analysis](#_Toc29921) 3

[**2. Methodology**](#_Toc29922)

[2.1 Data Pre Processing](#_Toc29923) 8

[2.1.1 Missing Value Analysis](#_Toc29924) 8

[2.1.2 Outlier Analysis](#_Toc29925) 8

[2.1.3 Feature Engineering](#_Toc29926) 11

[2.1.4 Feature Scaling](#_Toc29927) 17

2.2 Splitting train and test Dataset 18

2.3 Hyperparameter Optimization 19

[2.4 Model Development](#_Toc29928) 19

[2.4.1 Decision Tree](#_Toc29929) 19

[2.4.2 Linear Regression 20](#_Toc29930)

[2.4.3 Random Forest 21](#_Toc29931)

[2.4.4 XG Boosting 22](#_Toc29932)

[**3. Conclusion**](#_Toc29933)

[3.1 Model Evaluation](#_Toc29934) 23

[3.2 Model Selection](#_Toc29935) 24

[**4. Coding**](#_Toc29937)

[4.1 Python Coding](#_Toc29939) 24

**[References](#_Toc29940)**65

**1. Introduction**

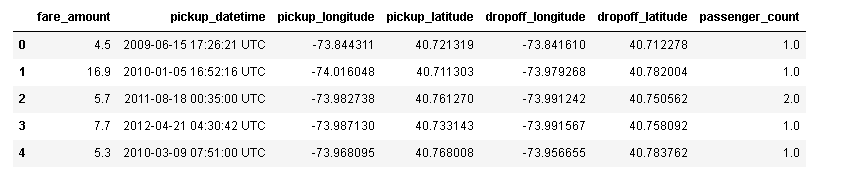
##### **1.1 Problem Statement**

The objective of this project is to predict Cab Fare amount.

You are a cab rental start-up company. You have successfully run the pilot project and now want to launch your cab service across the country. You have collected the historical data from your pilot project and now have a requirement to apply analytics for fare prediction. You need to design a system that predicts the fare amount for a cab ride in the city.

**1.2 Dataset**

Sample Dataset-



**Attribute Information:**

* pickup\_datetime - timestamp value indicating when the cab ride started.
* pickup\_longitude - float for longitude coordinate of where the cab ride started.
* pickup\_latitude - float for latitude coordinate of where the cab ride started.
* dropoff\_longitude - float for longitude coordinate of where the cab ride ended.
* dropoff\_latitude - float for latitude coordinate of where the cab ride ended.
* passenger\_count - an integer indicating the number of passengers in the cab ride.

Dataset has 7 variables in which 6 variables are independent and 1 (fare\_amount) is dependent variable. Since target variable is continuous in nature, this is a regression problem.

### **1.3 Exploratory Data Analysis**

Any predictive modelling requires that we look at the data before we start modelling. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis. To start this process, we will first try and look at all the probability distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the probability distributions or probability density functions of the variable.

Exploratory Data Analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics. In the given data set there are 7 variables and data types of all variables are object,float64 and int64. There are 16067 observations and 7 columns in our data set. Missing value is also present in our data.

We start this process after importing the train\_cab.csv file in python jupyter notebook or

R Studio.Before importing the data, having a look on the excel file pf data can help us draw many conclusions.

After importing the csv file we start cleaning the data. We have also the the test.csv file along with it.

The data comprises a date time column with UTC time zone. For proper analysis we can convert it to a datetime

format so that we can further use it for future extraction.

fare\_amount 16042 non-null float64

pickup\_datetime 16067 non-null object

pickup\_longitude 16067 non-null float64

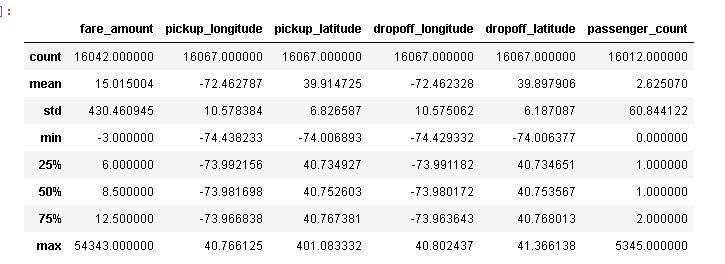
pickup\_latitude 16067 non-null float64

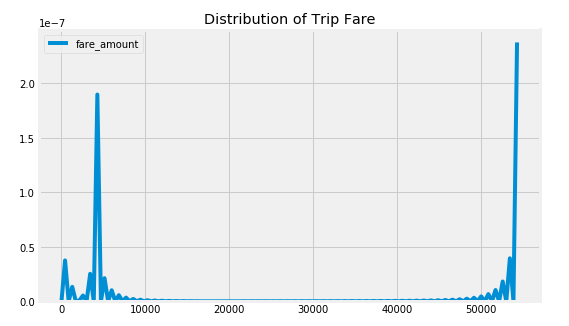
dropoff\_longitude 16067 non-null float64

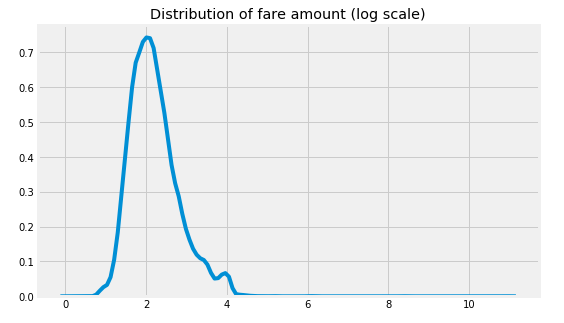
dropoff\_latitude 16067 non-null float64

passenger\_count 16012 non-null float64

The distribution of given attributes is like below:





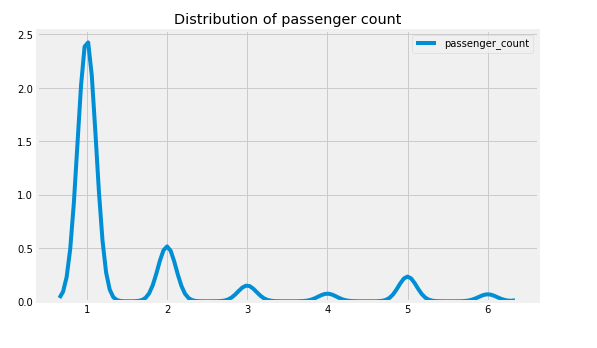


**Removing values which are not within desired range(outlier) depending upon basic nderstanding of dataset.**

In this step we will remove values in each variable which are not within desired range and we will consider them as outliers depending upon basic understanding of all the variables.

For the **fare\_amount** variable there are few records with negative fare and 0 fare it doesn't make any sence.so we will remove these records from the data.

In **passenger\_count** variable, we observed that few values are less than 1 and more than 7 in cab it is not possible. And in test data does not contain passenger\_count=0 . So if we feature engineer passenger\_count of train dataset then it will create a dummy variable for passenger\_count=0 which will be an extra feature compared to test dataset. so we will delete this passenger count values which are less than 1 and greater than 7.



For geographical Features like **pickup\_longitude,pickup\_latitude,dropoff\_longitude and dropoff\_latitude**

### Ranges in test data are below

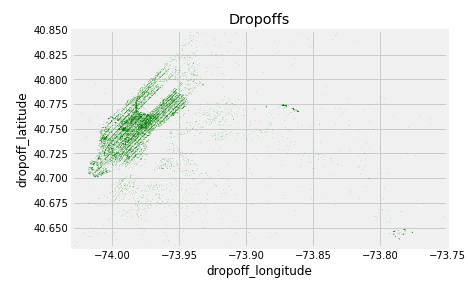
1) Longitude Boundary in test data

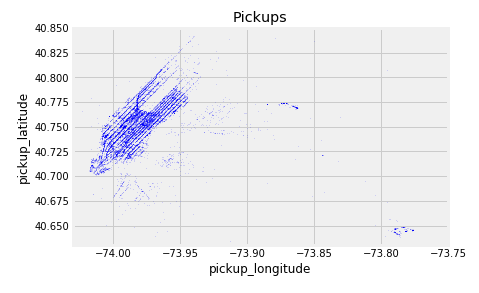
(-74.263242, -72.986532

2) Latitude Boundary in test data

(40.573143, 41.709555)

Based on test data boundaries, in train data we have outliers in pickup\_latitude variable. We will remove those values.





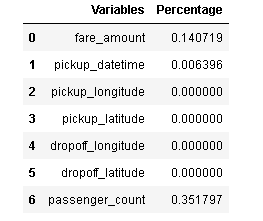
# **2. Methodology**

Before feeding the data to the model we need to clean the data and convert it to a proper format. It is the most crucial part of data science. In this we have to apply different preprocessing techniques to clean the data and to convert it into proper format.

##### **2.1 Data Pre-Processing**

#### **2.1.1 Missing Value Analysis**

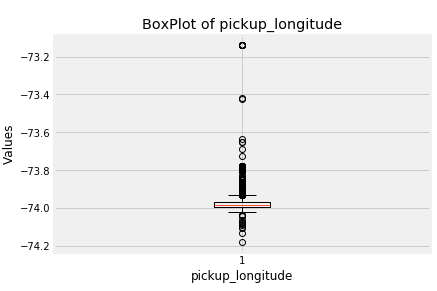
In statistics, missing data, or *missing values*, occur when no *data value* is stored for the variable in an observation. If a column has more than 30% of data as missing value either we ignore the entire column, or we ignore those observations. In the given data the maximum percentage of missing value is 0.351797% for passenger\_count column. And for fare\_amount has 0.140719% of missing values So, we have decided to drop all the missing values from the dataset.

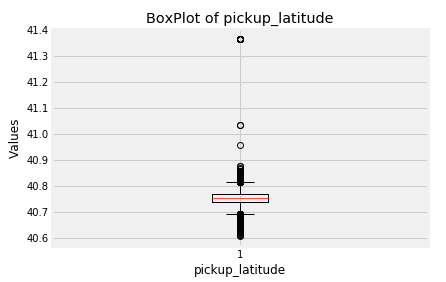


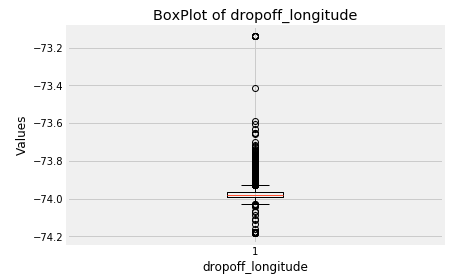
**2.1.2 Outlier Analysis**

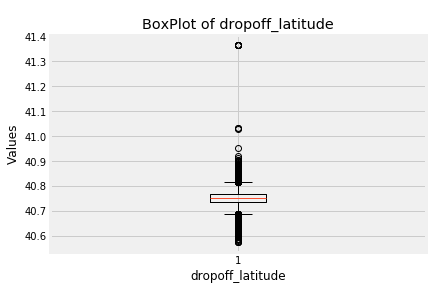
We can clearly observe from these probability distributions that most of the variables are skewed. The skew in these distributions can be most likely explained by the presence of outliers and extreme values in the data. One of the other steps of pre-processing apart from checking for normality is the presence of outliers. In this case we use a classic approach of removing outliers. We visualize the outliers using boxplots.

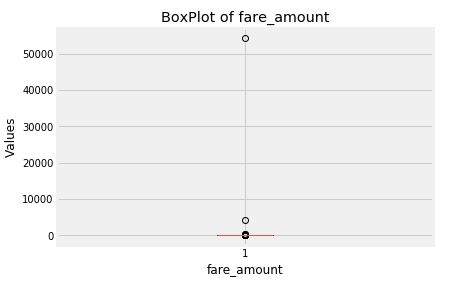
In figure we have plotted the boxplots of the 6 predictor variables with respect to target variable **fare\_amount** and detect the outliers by visualization.











From outlier analysis we remove all the outliers present in the data.

From the graph of outliers it is prominent that 0 lat and long is an outlier present in all the four variables. Thus double checking for the removal of all zeroes lat and long should be done. All such observations are removed due to outlier analysis.

There are outliers in passenger count as well. After finding the unique values of passenger count in dataframe we find the significant range for passenger count is from 1 to 7. Also the count less than 1 is found which are outliers and should be removed. The range of passenger count in test set is from 1 to 7.

**2.1.3 Feature Engineering**

Feature Engineering is used to drive new features from existing features.

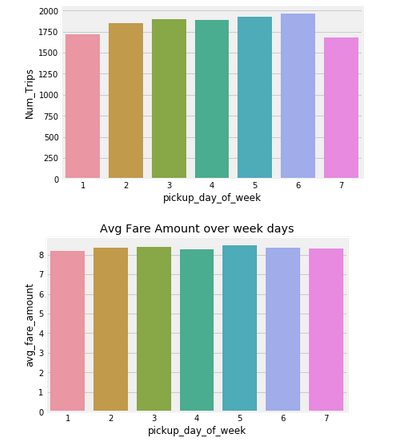
For **pickup\_datetime**

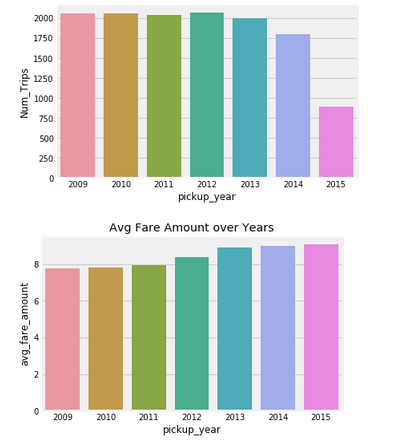
We will use this timestamp variable to create new variables.

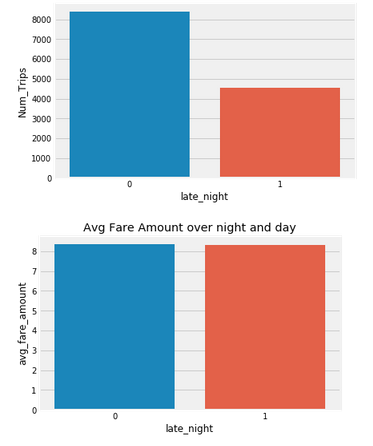
New features will be pickup\_date, pickup\_day, pickup\_hour, pickup\_day\_of\_week, pickup\_month, pickup\_year,and late\_night.

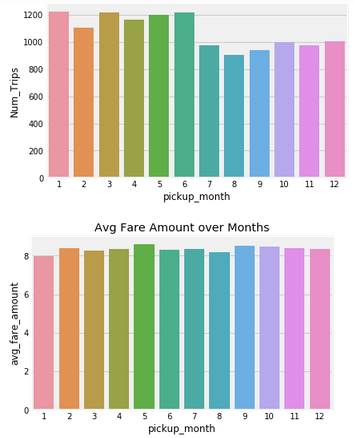
* ‘pickup\_date will contain only dates from pickup\_datetime. For ex. 1-3-2015 etc
* ‘pickup\_day’ will contain only dates from pickup\_datetime. For ex. 12 ,16,20 up to 31st..
* ‘pickup\_day\_of\_week ’ will contain only week from pickup\_datetime. For ex. 1 which is for Monday,2 for Tuesday,etc.
* ‘pickup\_hour’ will contain only hours from pickup\_datetime. For ex. 1, 2, 3, etc.
* ‘pickup\_month’ will contain only month from pickup\_datetime , for ex: 1(JAN) to 12(DEC).
* ‘pickup\_year’ will contain only year. For ex:2012, 2019 etc.
* late\_night will contain the ride is during late night or not for ex : 1 or 0.

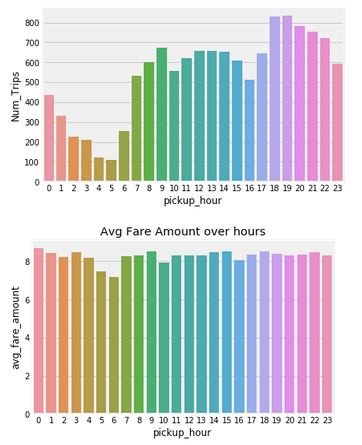
**We will see below, how extracted variables distributed for num\_trips and fare amount.**

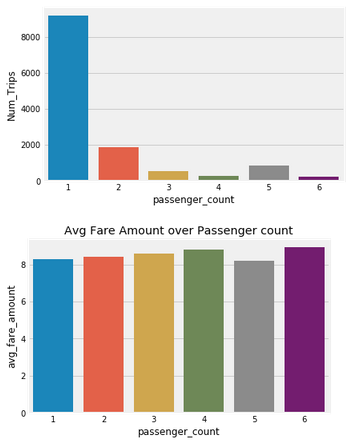


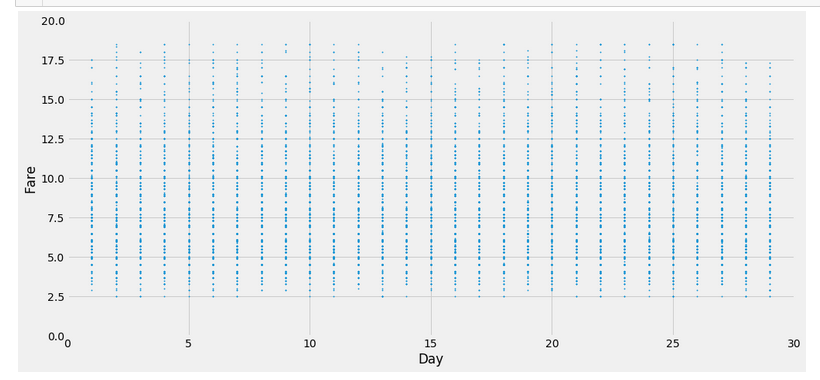


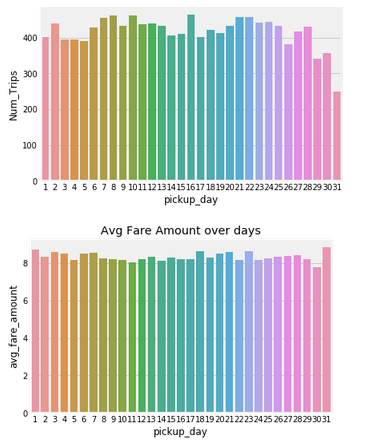












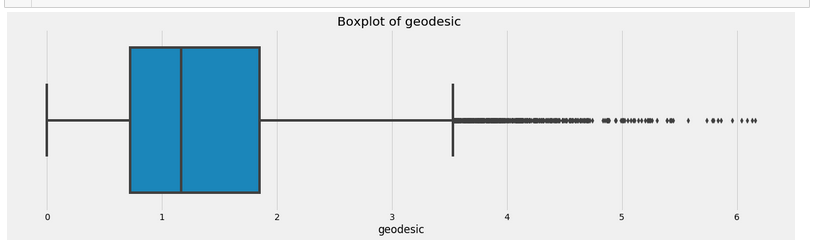
**For ‘Latitudes’ and ‘Longitudes’ variables:**

As we have latitude and longitude data for pickup and dropoff, we will find the distance the cab travelled from pickup and dropoff location.

We will use both haversine and vincenty methods to calculate distance. For haversine, variable name will be ‘great\_circle’ and for vincenty, new variable name will be ‘geodesic’.

As Vincenty is more accurate than haversine. Also, vincenty is prefered for short distances. Therefore, we will drop great\_circle.

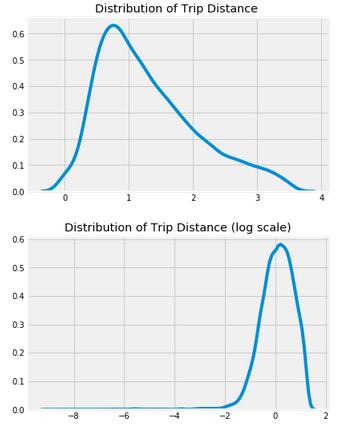
we will plot boxplot for our new variable ‘geodesic’:



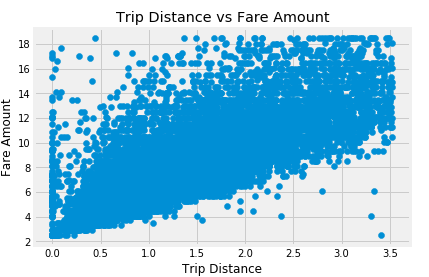
We delete these outliers

The scatter plot shows that most pickups and drops are from Manhattan in New York.

The distribution of the newly added feature distance is normally distributed.



Now let’s see the scatter plot of distance covered against its fare amount.



We see a linear distribution of fare amount with the distance covered except for few cases where

the fare amount is less even though the distance covered is large.

# **2.1.4 Feature Scaling**

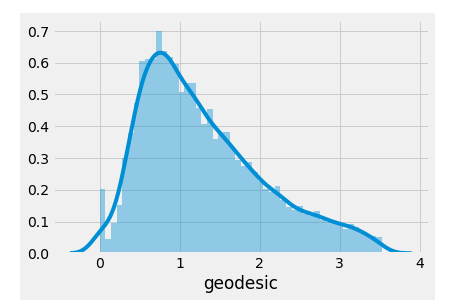
Data Scaling methods are used when we want our variables in data to scaled on common ground. It is performed only on continuous variables.

* • **Normalization**: Normalization refer to the dividing of a vector by its length. normalization normalizes the data in the range of 0 to 1. It is generally used when we are planning to use distance method for our model development purpose such as KNN. Normalizing the data improves convergence of such algorithms. Normalisation of data scales the data to a very small interval, where outliers can be loosed.
* • **Standardization**: Standardization refers to the subtraction of mean from individual point and then dividing by its SD. Z is negative when the raw score is below the mean and Z is positive when above mean. When the data is distributed normally you should go for standardization.

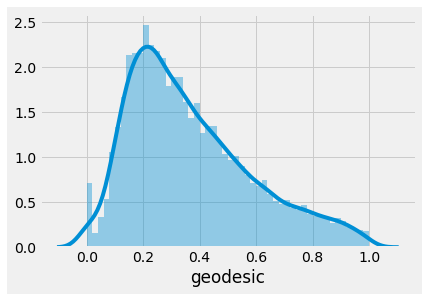
Linear Models assume that the data you are feeding are related in a linear fashion, or can be measured with a linear distance metric.

Also, our independent numerical variable ‘geodesic’ is not distributed normally so we had chosen normalization over standardization. Note: It is performed only on Continuous variables.

distplot() for ‘geodesic’ feature before normalization:



distplot() for ‘geodesic’ feature after normalization:



**2.2 Splitting train and Validation Dataset**

1. a) We have used sklearn’s train\_test\_split() method to divide whole Dataset into train and validation datset.
2. b) 25% is in validation dataset and 75% is in training data.
3. c) 9361 observations in training and 3121 observations in validation dataset.
4. d) We will test the performance of model on validation datset.
5. e) The model which performs best will be chosen to perform on test dataset provided along with original train dataset.
6. f) X\_train y\_train--are train subset.
7. g) X\_test y\_test--are validation subset.

**2.3 Hyperparameter Optimization**

1. a. To find the optimal hyperparameter we have used
2. sklearn.model\_selection.GridSearchCV. and sklearn.model\_selection.RandomizedSearchCV
3. b. GridSearchCV tries all the parameters that we provide it and then returns the best suited parameter for data.
4. c. We gave parameter dictionary to GridSearchCV which contains keys which are parameter names and values are the values of parameters which we want to try for.

Below are best hyperparameter we found for different models

1. **Decision Tree Regression**

'max\_depth': 6, 'min\_samples\_split': 2

Best score is 0.6381460733369655

**2)Linear Regression**

'copy\_X': True, 'fit\_intercept': True}

Best score is 0.649584211349103

**3)Random Forest**

**4)XGBoost**

##### **2.4 Model Development**

After Data pre-processing the next step is to develop a model using a train or historical data Which can perform to predict accurate result on test data or new data. Here we have tried with different model and will choose the model which will provide the most accurate values.

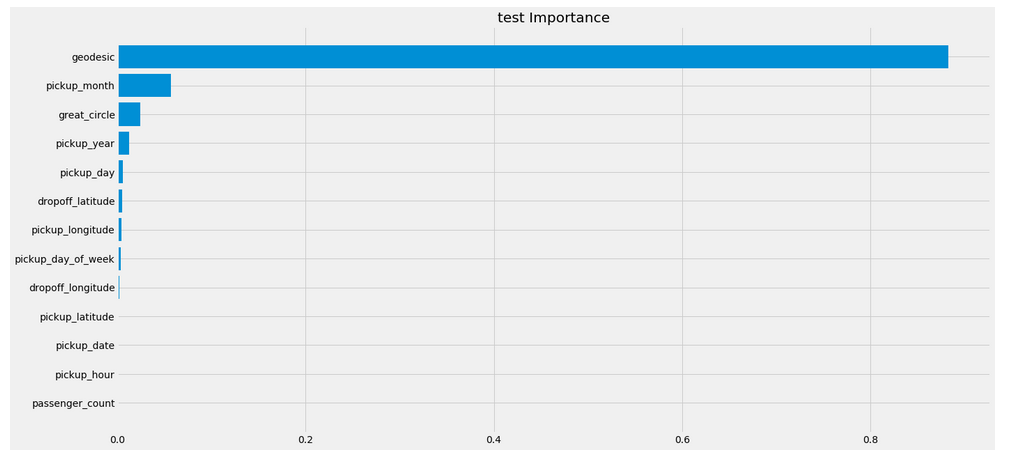
**2.4.1 Decision Tree**

Decision Tree is a supervised machine learning algorithm, which is used to predict the data for classification and regression. It accepts both continuous and categorical variables. A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. Each branch connects nodes with “and” and multiple branches are connected by “or”. Extremely easy to understand by the business users. It provides its output in the form of rule, which can easily understood by a non-technical person also.

We have prepared a model by using decision tree algorithm and calculate RMSE value and R^2 value for our project in R and Python are –

|  |  |  |
| --- | --- | --- |
| **Decision Tree** | **R** | **PYTHON** |
| **RMSE** | 2.2482 | 1.9336 |
| **R^2** | 0.6571 | 0.6391 |

Bar Plot of Decision Tree Feature Importance:



**2.4.2 Linear Regression**

The first step seeing the dependence of various features on the target variable shows a linearity in

behaviour. So the simplest model we use to develop on our data is linear regression.

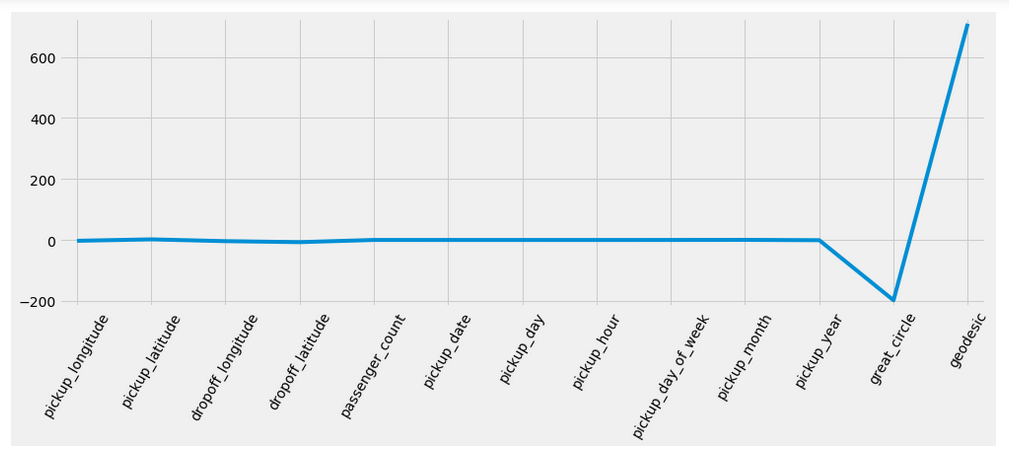
Before applying Linear Regression we find the variance inflation factor to find the multicollinearity in

the data. After checking we found that that there is no collinearity present.

We predict the fare amount using linear regression.

The RMSE value and R^2 value for our project in R and Python are –

|  |  |  |
| --- | --- | --- |
| Linear Regression | **R** | **PYTHON** |
| **RMSE** | 2.033 | 1.9076 |
| **R^2** | 0.7195 | 0.6509 |



**2.4.3 Random Forest**

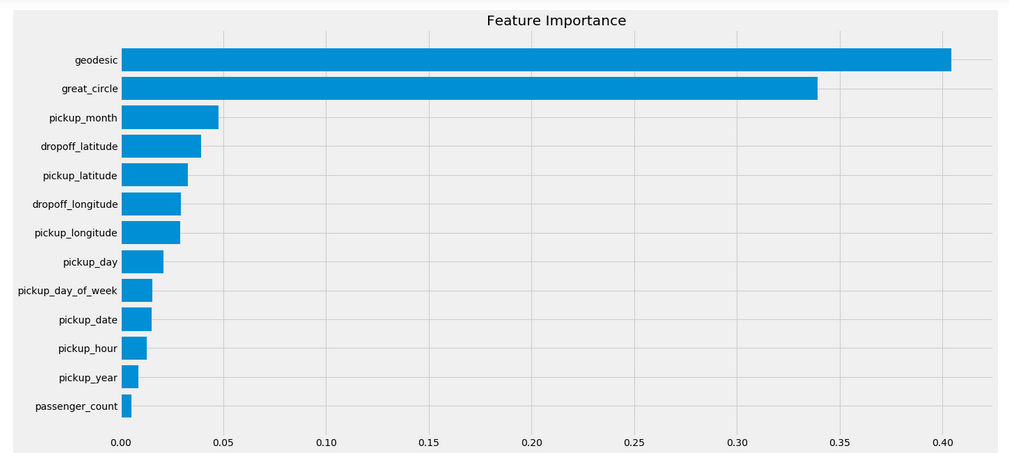
Random Forest is an ensemble technique that consists of many decision trees. The idea behind Random Forest is to build n number of trees to have more accuracy in dataset. It is called random forest as we are building n number of trees randomly. In other words,

to build the decision trees it selects randomly n number of variables and n number of observations. It means to build each decision tree on random forest we are not going to use the same data. The higher no of trees in the random forest will give higher no of accuracy, so in random forest we can go for multiple trees. It can handle large no of independent variables without variable deletion and it will give the estimates that what variables are important.

The RMSE value and R^2 value for our project in R and Python are –

|  |  |  |
| --- | --- | --- |
| **Random Forest** | **R** | **PYTHON** |
| **RMSE** | 2.0031 | 1.8197 |
| **R^2** | 0.7408 | 0.6804 |

Bar Plot of Random Forest Feature Importance:



**2.4.4 XGBoost**

[XGBoost](https://xgboost.ai/?source=post_page---------------------------)is a decision-tree-based ensemble Machine Learning algorithm that uses a [gradient boosting](https://en.wikipedia.org/wiki/Gradient_boosting?source=post_page---------------------------) framework. Decision trees, in their simplest form, are easy-to-visualize and fairly interpretable algorithms but building intuition for the next-generation of tree-based algorithms can be a bit tricky.

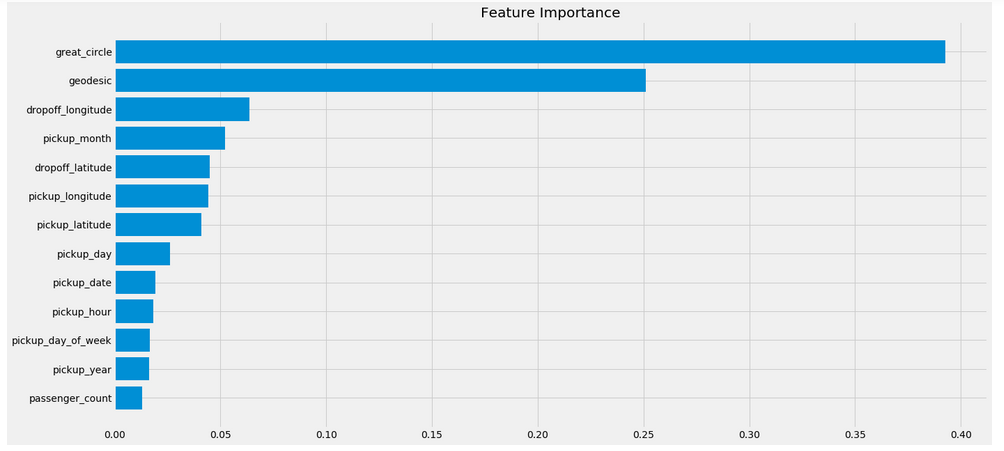
Think of XGBoost as gradient boosting on ‘steroids’ (well it is called ‘Extreme Gradient Boosting’ for a reason!). It is a perfect combination of software and hardware optimization techniques to yield superior results using less computing resources in the shortest amount of time.

XGBoost and Gradient Boosting Machines (GBMs) are both ensemble tree methods that apply the principle of boosting weak learners ([CARTs](https://www.datasciencecentral.com/profiles/blogs/introduction-to-classification-regression-trees-cart?source=post_page---------------------------)generally) using the gradient descent architecture. However, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.

The RMSE value and R^2 value for our project in R and Python are –

|  |  |  |
| --- | --- | --- |
| **XGBoost** | **R** | **PYTHON** |
| **RMSE** | 1.9499 | 1.7565 |
| **R^2** | 0.7428 | 0.7022 |

Bar Plot of XGBoost Feature Importance:



# **3. Conclusion**

In methodology we have done data cleaning and then applied different machine learning algorithms on the data set to check the performance of each model, now in conclusion we will finalize the model of Employee Absenteeism dataset.

**3.1 Model Evaluation**

In the previous chapter we have applied four algorithms on our dataset and calculate the

Root Mean Square Error (RMSE) and R-Squared Value for all the models. Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are, RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. RMSE is an absolute measure of fit. RMSE can be interpreted as the standard deviation of the unexplained variance, and has the useful property of being in the same units as the response variable. R-squared is a relative measure of fit. R-squared is basically explains the degree to which input variable explain the variation of the output. In simple words R-squared tells how much variance of dependent variable explained by the independent variable. It is a measure if goodness of fit in regression line. Value of R-squared between 0-1, where 0 means independent variable unable to explain the target variable and 1 means target variable is completely explained by the independent variable. So, Lower values of RMSE and higher value of R-Squared Value indicate better fit of model.

**3.2 Model Selection**

From the observation of all **RMSE Value** and **R-Squared** Value we have concluded that **XGBoost** has minimum value of RMSE (**1.7565)** and it’s **R-Squared** Value is also maximum (**0.74**).Means, By XGBoost algorithm predictor are explain 74% to the target variable on the test data. The RMSE value of Test data and Train does not differs a lot this implies that it is not the case of overfitting.

**4.Coding**

**4.1 Python Coding**

import os

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import warnings

from datetime import datetime

import calendar

from math import sin, cos, sqrt, atan2, radians,asin

import folium

from folium import FeatureGroup, LayerControl, Map, Marker

from folium.plugins import HeatMap

from folium.plugins import TimestampedGeoJson

from folium.plugins import MarkerCluster

from geopy.distance import great\_circle

import matplotlib.dates as mdates

import matplotlib as mpl

from datetime import timedelta

import datetime as dt

warnings.filterwarnings('ignore')

pd.set\_option('display.max\_colwidth', -1)

plt.style.use('fivethirtyeight')

import folium

from sklearn.cluster import KMeans

from sklearn import preprocessing

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import Imputer

from sklearn import linear\_model

from sklearn.metrics import mean\_squared\_error

from sklearn import metrics

from sklearn.ensemble import RandomForestRegressor

from haversine import haversine

import pickle

from geopy.distance import geodesic

from xgboost import XGBRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.linear\_model import LinearRegression

#Set working directory

os.chdir("C:/Data science/Project/Cab\_fare\_prediction")

os.getcwd()

'C:\\Data science\\Project\\Cab\_fare\_prediction'

#load the data

Cab\_train=pd.read\_csv("train\_cab.csv",dtype={'fare\_amount':np.float64},na\_values={'fare\_amount':'430-'})

Cab\_test=pd.read\_csv("test.csv")

Cab\_train.head()

|  | **fare\_amount** | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.5 | 2009-06-15 17:26:21 UTC | -73.844311 | 40.721319 | -73.841610 | 40.712278 | 1.0 |
| **1** | 16.9 | 2010-01-05 16:52:16 UTC | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1.0 |
| **2** | 5.7 | 2011-08-18 00:35:00 UTC | -73.982738 | 40.761270 | -73.991242 | 40.750562 | 2.0 |
| **3** | 7.7 | 2012-04-21 04:30:42 UTC | -73.987130 | 40.733143 | -73.991567 | 40.758092 | 1.0 |
| **4** | 5.3 | 2010-03-09 07:51:00 UTC | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1.0 |

Cab\_train.shape

(16067, 7)

Cab\_test.head()

|  | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 2015-01-27 13:08:24 UTC | -73.973320 | 40.763805 | -73.981430 | 40.743835 | 1 |
| **1** | 2015-01-27 13:08:24 UTC | -73.986862 | 40.719383 | -73.998886 | 40.739201 | 1 |
| **2** | 2011-10-08 11:53:44 UTC | -73.982524 | 40.751260 | -73.979654 | 40.746139 | 1 |
| **3** | 2012-12-01 21:12:12 UTC | -73.981160 | 40.767807 | -73.990448 | 40.751635 | 1 |
| **4** | 2012-12-01 21:12:12 UTC | -73.966046 | 40.789775 | -73.988565 | 40.744427 | 1 |

Cab\_train.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 16067 entries, 0 to 16066

Data columns (total 7 columns):

fare\_amount 16042 non-null float64

pickup\_datetime 16067 non-null object

pickup\_longitude 16067 non-null float64

pickup\_latitude 16067 non-null float64

dropoff\_longitude 16067 non-null float64

dropoff\_latitude 16067 non-null float64

passenger\_count 16012 non-null float64

dtypes: float64(6), object(1)

memory usage: 878.7+ KB

#Convert pickup\_datetime from Object to Datetime object

Cab\_train['pickup\_datetime']=pd.to\_datetime(Cab\_train['pickup\_datetime'],format='%Y-%m-%d %H:%M:%S UTC',errors='coerce')

#Cab\_train.head()

Cab\_test['pickup\_datetime']=pd.to\_datetime(Cab\_test['pickup\_datetime'],format='%Y-%m-%d %H:%M:%S UTC',errors='coerce')

Cab\_train.isnull().sum()

fare\_amount 25

pickup\_datetime 1

pickup\_longitude 0

pickup\_latitude 0

dropoff\_longitude 0

dropoff\_latitude 0

passenger\_count 55

dtype: int64

Cab\_train['pickup\_datetime'].count()

16066

Cab\_train.describe()

|  | **fare\_amount** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- | --- |
| **count** | 16042.000000 | 16067.000000 | 16067.000000 | 16067.000000 | 16067.000000 | 16012.000000 |
| **mean** | 15.015004 | -72.462787 | 39.914725 | -72.462328 | 39.897906 | 2.625070 |
| **std** | 430.460945 | 10.578384 | 6.826587 | 10.575062 | 6.187087 | 60.844122 |
| **min** | -3.000000 | -74.438233 | -74.006893 | -74.429332 | -74.006377 | 0.000000 |
| **25%** | 6.000000 | -73.992156 | 40.734927 | -73.991182 | 40.734651 | 1.000000 |
| **50%** | 8.500000 | -73.981698 | 40.752603 | -73.980172 | 40.753567 | 1.000000 |
| **75%** | 12.500000 | -73.966838 | 40.767381 | -73.963643 | 40.768013 | 2.000000 |
| **max** | 54343.000000 | 40.766125 | 401.083332 | 40.802437 | 41.366138 | 5345.000000 |

## Exploratory Data Analysis

Distribution of Trip Fare

Cab\_train.loc[Cab\_train['fare\_amount']<1].shape

(5, 7)

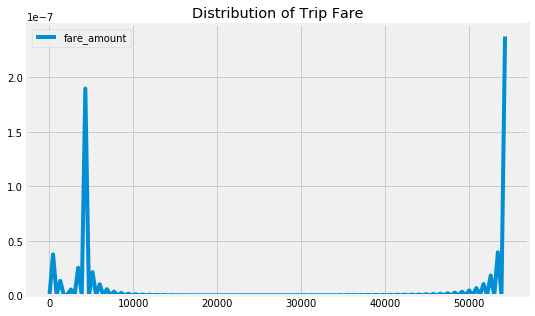
There are few records with negative fare and 0 fare it doesn't make any sence,so We will remove these records from the data

Cab\_train = Cab\_train.drop(Cab\_train[Cab\_train['fare\_amount']<1].index, axis=0)

plt.figure(figsize=(8,5))

sns.kdeplot(Cab\_train['fare\_amount']).set\_title("Distribution of Trip Fare")

Text(0.5, 1.0, 'Distribution of Trip Fare')

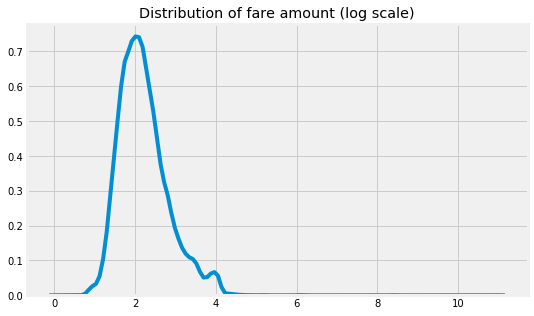
****

We can see there are outliers in fare amount we will delete those in outlier analysis

plt.figure(figsize=(8,5))

sns.kdeplot(np.log(Cab\_train['fare\_amount'].values)).set\_title("Distribution of fare amount (log scale)")

Text(0.5, 1.0, 'Distribution of fare amount (log scale)')

****

Distributaion of Passenger\_count

#Checking passenger\_count less than 1

Cab\_train.loc[Cab\_train['passenger\_count']<1].shape

(58, 7)

#Checking passenger\_count greater than 7

Cab\_train.loc[Cab\_train['passenger\_count']>7].shape

(20, 7)

#Checking passenger\_count range

for i in range(4,11):

print('passenger\_count above' +str(i)+'={}'.format(sum(Cab\_train['passenger\_count']>i)))

passenger\_count above4=1367

passenger\_count above5=322

passenger\_count above6=20

passenger\_count above7=20

passenger\_count above8=20

passenger\_count above9=20

passenger\_count above10=20

Cab\_test['passenger\_count'].unique()

array([1, 2, 3, 4, 5, 6], dtype=int64)

In passenger\_count variable, we observerd that few values are less than 1 and more than 7 in cab it is not possible And test data does not contain passenger\_count=0 . So if we feature engineer passenger\_count of train dataset then it will create a dummy variable for passenger\_count=0 which will be an extra feature compared to test dataset. so we will delete this passenger count values which are less than 1 and greater than 7

Cab\_train = Cab\_train.drop(Cab\_train[Cab\_train['passenger\_count']>7].index, axis=0)

Cab\_train = Cab\_train.drop(Cab\_train[Cab\_train['passenger\_count']<1].index, axis=0)

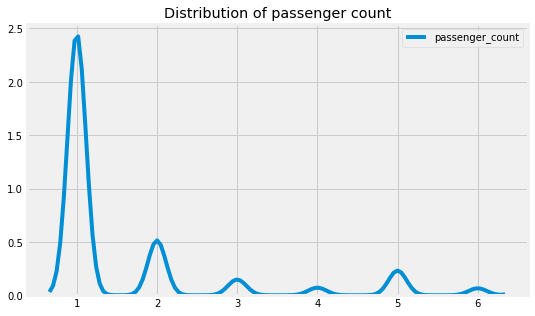
Cab\_train.loc[Cab\_train['passenger\_count']<1].shape

(0, 7)

plt.figure(figsize=(8,5))

sns.kdeplot(Cab\_train['passenger\_count']).set\_title("Distribution of passenger count")

Text(0.5, 1.0, 'Distribution of passenger count')

****

### Let us look at Geographical Features

Distribution of Pickup and Dropoff Lat Lng

#Checking latitude and longitude ranges in train data

print("Range of Pickup Latitude is ", (min(Cab\_train['pickup\_latitude']),max(Cab\_train['pickup\_latitude'])))

Range of Pickup Latitude is (-74.006893, 401.083332)

print("Range of Dropoff Latitude is ", (min(Cab\_train['dropoff\_latitude']),max(Cab\_train['dropoff\_latitude'])))

Range of Dropoff Latitude is (-74.006377, 41.366138)

print("Range of Dropoff Latitude is ", (min(Cab\_train['pickup\_longitude']),max(Cab\_train['pickup\_longitude'])))

Range of Dropoff Latitude is (-74.438233, 40.766125)

print("Range of Dropoff Latitude is ", (min(Cab\_train['dropoff\_longitude']),max(Cab\_train['dropoff\_longitude'])))

Range of Dropoff Latitude is (-74.42933199999999, 40.802437)

#Checking latitude and longitude ranges in test data

print("Longitude Boundary in test data")

min(Cab\_test.pickup\_longitude.min(), Cab\_test.dropoff\_longitude.min()),max(Cab\_test.pickup\_longitude.max(), Cab\_test.dropoff\_longitude.max())

Longitude Boundary in test data

(-74.263242, -72.986532)

print("Latitude Boundary in test data")

min(Cab\_test.pickup\_latitude.min(), Cab\_test.pickup\_latitude.min()),max(Cab\_test.pickup\_latitude.max(), Cab\_test.pickup\_latitude.max())

Latitude Boundary in test data

(40.573143, 41.709555)

Based on test data boundaries, in train data we have outliers in pickup\_latitude variable

#checking whether these values having 0 values

Cab\_train[(Cab\_train.pickup\_latitude==0) | (Cab\_train.pickup\_longitude)==0 | (Cab\_train.dropoff\_latitude==0)|(Cab\_train.dropoff\_longitude==0)].shape

(312, 7)

boundary={'min\_lng':-74.263242,

'min\_lat':40.573143,

'max\_lng':-72.986532,

'max\_lat':41.709555}

Cab\_train.loc[~((Cab\_train.pickup\_longitude >= boundary['min\_lng'] ) & (Cab\_train.pickup\_longitude <= boundary['max\_lng']) &

(Cab\_train.pickup\_latitude >= boundary['min\_lat']) & (Cab\_train.pickup\_latitude <= boundary['max\_lat']) &

(Cab\_train.dropoff\_longitude >= boundary['min\_lng']) & (Cab\_train.dropoff\_longitude <= boundary['max\_lng']) &

(Cab\_train.dropoff\_latitude >=boundary['min\_lat']) & (Cab\_train.dropoff\_latitude <= boundary['max\_lat'])),'is\_outlier\_loc']=1

Cab\_train.loc[((Cab\_train.pickup\_longitude >= boundary['min\_lng'] ) & (Cab\_train.pickup\_longitude <= boundary['max\_lng']) &

(Cab\_train.pickup\_latitude >= boundary['min\_lat']) & (Cab\_train.pickup\_latitude <= boundary['max\_lat']) &

(Cab\_train.dropoff\_longitude >= boundary['min\_lng']) & (Cab\_train.dropoff\_longitude <= boundary['max\_lng']) &

(Cab\_train.dropoff\_latitude >=boundary['min\_lat']) & (Cab\_train.dropoff\_latitude <= boundary['max\_lat'])),'is\_outlier\_loc']=0

print("Outlier vs Non Outlier Counts")

print(Cab\_train['is\_outlier\_loc'].value\_counts())

# Let us drop rows, where location is outlier

Cab\_train=Cab\_train.loc[Cab\_train['is\_outlier\_loc']==0]

Cab\_train.drop(['is\_outlier\_loc'],axis=1,inplace=True)

Outlier vs Non Outlier Counts

0.0 15634

1.0 350

Name: is\_outlier\_loc, dtype: int64

#Plot Heatmap of Pickups and Dropoffs

city\_long\_border = (-74.03, -73.75)

city\_lat\_border = (40.63, 40.85)

Cab\_train.plot(kind='scatter', x='dropoff\_longitude', y='dropoff\_latitude',

color='green',

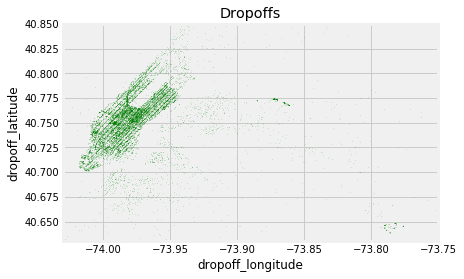
s=.02, alpha=.6)

plt.title("Dropoffs")

plt.ylim(city\_lat\_border)

plt.xlim(city\_long\_border)

(-74.03, -73.75)

****

Cab\_train.plot(kind='scatter', x='pickup\_longitude', y='pickup\_latitude',

color='blue',

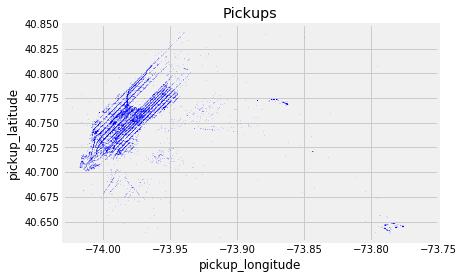
s=.02, alpha=.6)

plt.title("Pickups")

plt.ylim(city\_lat\_border)

plt.xlim(city\_long\_border)

(-74.03, -73.75)

****

# Missing Value Analysis

#Missing values in each variable-

missing\_val=pd.DataFrame(Cab\_train.isnull().sum())

missing\_val=(missing\_val/len(Cab\_train))\*100

missing\_val.columns=['Percentage']

missing\_val=missing\_val.reset\_index()

#missing\_val

missing\_val=missing\_val.rename(columns={'index':'Variables',})

missing\_val

|  | **Variables** | **Percentage** |
| --- | --- | --- |
| **0** | fare\_amount | 0.140719 |
| **1** | pickup\_datetime | 0.006396 |
| **2** | pickup\_longitude | 0.000000 |
| **3** | pickup\_latitude | 0.000000 |
| **4** | dropoff\_longitude | 0.000000 |
| **5** | dropoff\_latitude | 0.000000 |
| **6** | passenger\_count | 0.351797 |

#We decided to drop all the missing values in data set

Cab\_train=Cab\_train.dropna()

# Outlier Analysis

#copy of data

df1= Cab\_train.copy()

#Cab\_train= df1.copy()

df1.shape

(15556, 7)

Cab\_train.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 15556 entries, 0 to 16065

Data columns (total 7 columns):

fare\_amount 15556 non-null float64

pickup\_datetime 15556 non-null datetime64[ns]

pickup\_longitude 15556 non-null float64

pickup\_latitude 15556 non-null float64

dropoff\_longitude 15556 non-null float64

dropoff\_latitude 15556 non-null float64

passenger\_count 15556 non-null float64

dtypes: datetime64[ns](1), float64(6)

memory usage: 972.2 KB

#saving all continuous variables in one data frame

cnames=['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude','fare\_amount']

##Plot boxplot to visulazie outliers-

for i in cnames:

print(i)

plt.boxplot(Cab\_train[i])

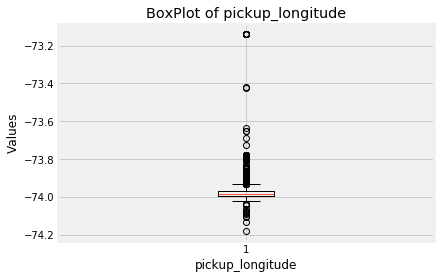
plt.xlabel(i)

plt.ylabel('Values')

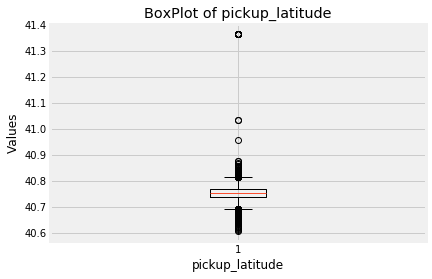
plt.title("BoxPlot of "+i)

plt.show()

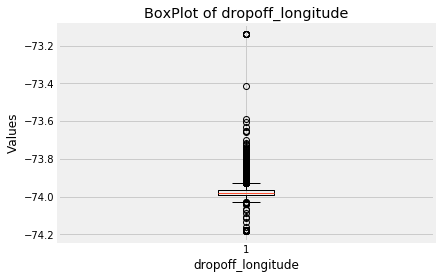
pickup\_longitude

****

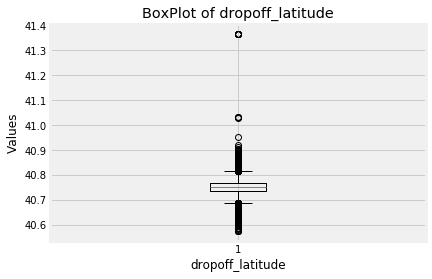
pickup\_latitude

****

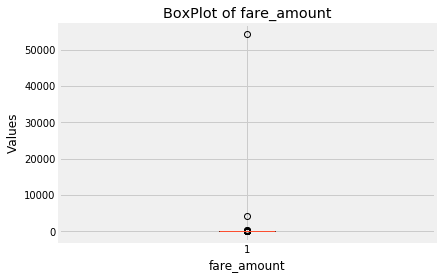
dropoff\_longitude

****

dropoff\_latitude

****

fare\_amount

****

##Calculate iqr, lower fence and upper fence-

for i in cnames:

print(i)

q75,q25= np.percentile(Cab\_train.loc[:,i],[75,25])

iqr= q75-q25

minimum= q25-(iqr\*1.5)

maximum= q75+(iqr\*1.5)

print(minimum)

print(maximum)

print(iqr)

Cab\_train=Cab\_train.drop(Cab\_train[Cab\_train.loc[:,i] < minimum].index)

Cab\_train=Cab\_train.drop(Cab\_train[Cab\_train.loc[:,i] > maximum].index)

pickup\_longitude

-74.02877313125

-73.93169652125002

0.02426915249999695

pickup\_latitude

40.69273876874998

40.81111313875001

0.029593592500006594

dropoff\_longitude

-74.02877425625002

-73.92960984625

0.024791102500003603

dropoff\_latitude

40.69245499

40.81290835

0.030113339999999766

fare\_amount

-2.1000000000000005

18.700000000000003

5.2

Cab\_train.isnull().sum()

fare\_amount 0

pickup\_datetime 0

pickup\_longitude 0

pickup\_latitude 0

dropoff\_longitude 0

dropoff\_latitude 0

passenger\_count 0

dtype: int64

# Feature Extraction

Create datetime features based on pickup\_datetime

def encodeDays(day\_of\_week):

day\_dict={'Sunday':7,'Monday':1,'Tuesday':2,'Wednesday':3,'Thursday':4,'Friday':5,'Saturday':6}

return day\_dict[day\_of\_week]

def late\_night (row):

if (row['pickup\_hour'] <= 6) or (row['pickup\_hour'] >= 20):

return 1

else:

return 0

Cab\_train['pickup\_date']= Cab\_train['pickup\_datetime'].dt.date

Cab\_train['pickup\_day']=Cab\_train['pickup\_datetime'].apply(lambda x:x.day)

Cab\_train['pickup\_hour']=Cab\_train['pickup\_datetime'].apply(lambda x:x.hour)

Cab\_train['pickup\_day\_of\_week']=Cab\_train['pickup\_datetime'].apply(lambda x:calendar.day\_name[x.weekday()])

Cab\_train['pickup\_day\_of\_week']=Cab\_train['pickup\_day\_of\_week'].apply(lambda x:encodeDays(x))

Cab\_train['pickup\_month']=Cab\_train['pickup\_datetime'].apply(lambda x:x.month)

Cab\_train['pickup\_year']=Cab\_train['pickup\_datetime'].apply(lambda x:x.year)

Cab\_train['late\_night'] = Cab\_train.apply (lambda x: late\_night(x), axis=1)

Cab\_train['passenger\_count']=Cab\_train['passenger\_count'].astype('int64')

#Extracting features from test data as well

def encodeDays(day\_of\_week):

day\_dict={'Sunday':7,'Monday':1,'Tuesday':2,'Wednesday':3,'Thursday':4,'Friday':5,'Saturday':6}

return day\_dict[day\_of\_week]

def late\_night (row):

if (row['pickup\_hour'] <= 6) or (row['pickup\_hour'] >= 20):

return 1

else:

return 0

Cab\_test['pickup\_date']=Cab\_test['pickup\_datetime'].dt.date

Cab\_test['pickup\_day']=Cab\_test['pickup\_datetime'].apply(lambda x:x.day)

Cab\_test['pickup\_hour']=Cab\_test['pickup\_datetime'].apply(lambda x:x.hour)

Cab\_test['pickup\_day\_of\_week']=Cab\_test['pickup\_datetime'].apply(lambda x:calendar.day\_name[x.weekday()])

Cab\_test['pickup\_day\_of\_week']=Cab\_test['pickup\_day\_of\_week'].apply(lambda x:encodeDays(x))

Cab\_test['pickup\_month']=Cab\_test['pickup\_datetime'].apply(lambda x:x.month)

Cab\_test['pickup\_year']=Cab\_test['pickup\_datetime'].apply(lambda x:x.year)

Cab\_test['late\_night'] = Cab\_test.apply (lambda x: late\_night(x), axis=1)

Cab\_test['passenger\_count']=Cab\_test['passenger\_count'].astype('int64')

Feature Distribution against Target variable

#pickup\_day\_of\_week

No\_trips\_weekday=pd.DataFrame(Cab\_train['pickup\_day\_of\_week'].value\_counts())

No\_trips\_weekday=No\_trips\_weekday.reset\_index()

No\_trips\_weekday=No\_trips\_weekday.rename(columns={'index':'pickup\_day\_of\_week','pickup\_day\_of\_week':'Num\_Trips'})

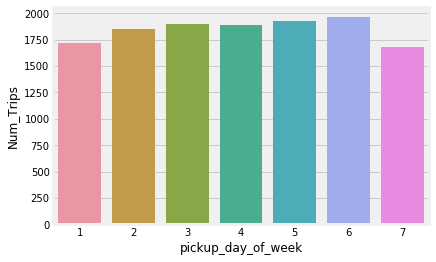
sns.barplot(x='pickup\_day\_of\_week',y='Num\_Trips',data=No\_trips\_weekday)

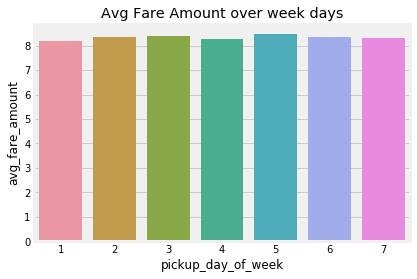
plt.show()

trips\_weekday\_fareamount=Cab\_train.groupby(['pickup\_day\_of\_week'])['fare\_amount'].mean().reset\_index().rename(columns={'fare\_amount':'avg\_fare\_amount'})

sns.barplot(x='pickup\_day\_of\_week',y='avg\_fare\_amount',data=trips\_weekday\_fareamount).set\_title("Avg Fare Amount over week days")

plt.show()

****

****

#pickup\_year

no\_trips\_year=pd.DataFrame(Cab\_train['pickup\_year'].value\_counts())

no\_trips\_year=no\_trips\_year.reset\_index()

no\_trips\_year=no\_trips\_year.rename(columns={'index':'pickup\_year','pickup\_year':'Num\_Trips'})

no\_trips\_year.head()

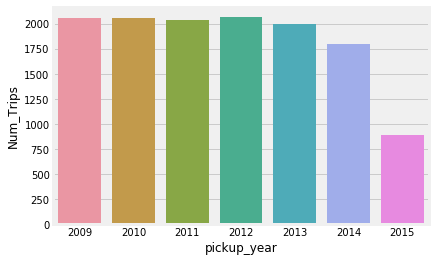
sns.barplot(x='pickup\_year',y='Num\_Trips',data=no\_trips\_year)

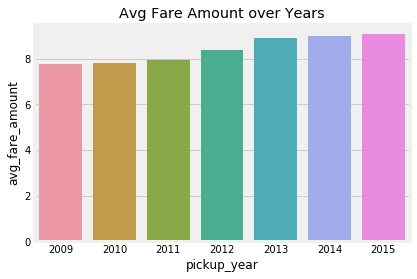
plt.show()

trips\_year\_fareamount=Cab\_train.groupby(['pickup\_year'])['fare\_amount'].mean().reset\_index().rename(columns={'fare\_amount':'avg\_fare\_amount'})

sns.barplot(x='pickup\_year',y='avg\_fare\_amount',data=trips\_year\_fareamount).set\_title("Avg Fare Amount over Years")

plt.show()

****

****

#late\_night

no\_trips\_night=pd.DataFrame(Cab\_train['late\_night'].value\_counts())

no\_trips\_night=no\_trips\_night.reset\_index()

no\_trips\_night=no\_trips\_night.rename(columns={'index':'late\_night','late\_night':'Num\_Trips'})

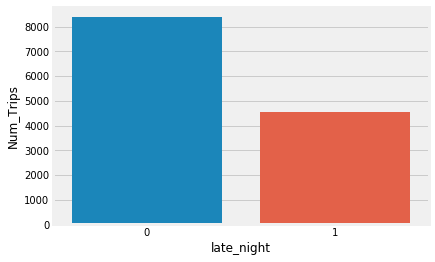
sns.barplot(x='late\_night',y='Num\_Trips',data=no\_trips\_night)

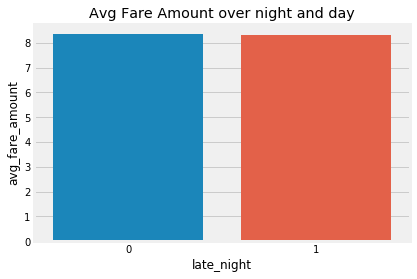
plt.show()

trips\_night\_fareamount=Cab\_train.groupby(['late\_night'])['fare\_amount'].mean().reset\_index().rename(columns={'fare\_amount':'avg\_fare\_amount'})

sns.barplot(x='late\_night',y='avg\_fare\_amount',data=trips\_night\_fareamount).set\_title("Avg Fare Amount over night and day")

plt.show()

****

****

#pickup\_month

no\_trips\_month=pd.DataFrame(Cab\_train['pickup\_month'].value\_counts())

no\_trips\_month=no\_trips\_month.reset\_index()

no\_trips\_month=no\_trips\_month.rename(columns={'index':'pickup\_month','pickup\_month':'Num\_Trips'})

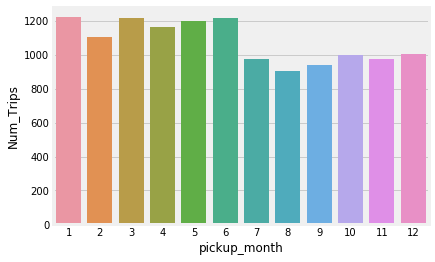
sns.barplot(x='pickup\_month',y='Num\_Trips',data=no\_trips\_month)

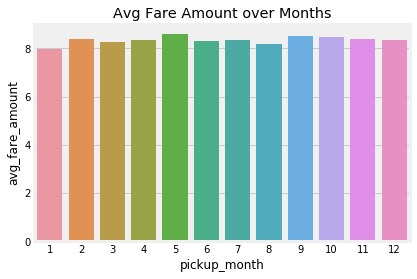
plt.show()

trips\_month\_fareamount=Cab\_train.groupby(['pickup\_month'])['fare\_amount'].mean().reset\_index().rename(columns={'fare\_amount':'avg\_fare\_amount'})

sns.barplot(x='pickup\_month',y='avg\_fare\_amount',data=trips\_month\_fareamount).set\_title("Avg Fare Amount over Months")

plt.show()

****

****

#pickup\_hour

trips\_hour=pd.DataFrame(Cab\_train['pickup\_hour'].value\_counts())

trips\_hour=trips\_hour.reset\_index()

trips\_hour=trips\_hour.rename(columns={'index':'pickup\_hour','pickup\_hour':'Num\_Trips'})

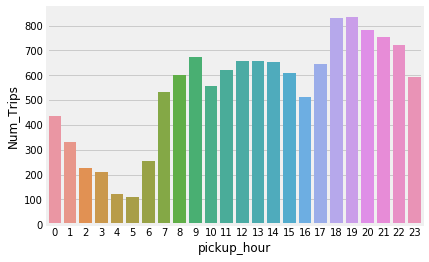
sns.barplot(x='pickup\_hour',y='Num\_Trips',data=trips\_hour)

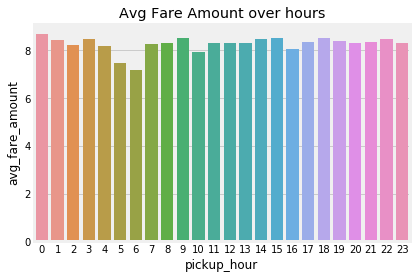
plt.show()

trips\_hour\_fareamount=Cab\_train.groupby(['pickup\_hour'])['fare\_amount'].mean().reset\_index().rename(columns={'fare\_amount':'avg\_fare\_amount'})

sns.barplot(x='pickup\_hour',y='avg\_fare\_amount',data=trips\_hour\_fareamount).set\_title("Avg Fare Amount over hours")

plt.show()

****

****

#passenger\_count

pass\_count=pd.DataFrame(Cab\_train['passenger\_count'].value\_counts())

pass\_count=pass\_count.reset\_index()

pass\_count=pass\_count.rename(columns={'index':'passenger\_count','passenger\_count':'Num\_Trips'})

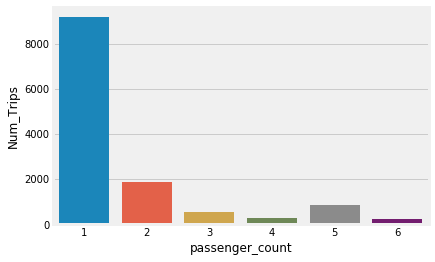
sns.barplot(x='passenger\_count',y='Num\_Trips',data=pass\_count)

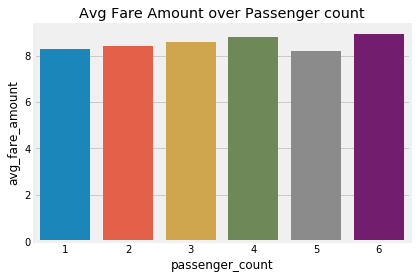
plt.show()

pass\_count\_fareamount=Cab\_train.groupby(['passenger\_count'])['fare\_amount'].mean().reset\_index().rename(columns={'fare\_amount':'avg\_fare\_amount'})

sns.barplot(x='passenger\_count',y='avg\_fare\_amount',data=pass\_count\_fareamount).set\_title("Avg Fare Amount over Passenger count")

plt.show()

****

****

##pickup\_day

plt.figure(figsize=(15,7))

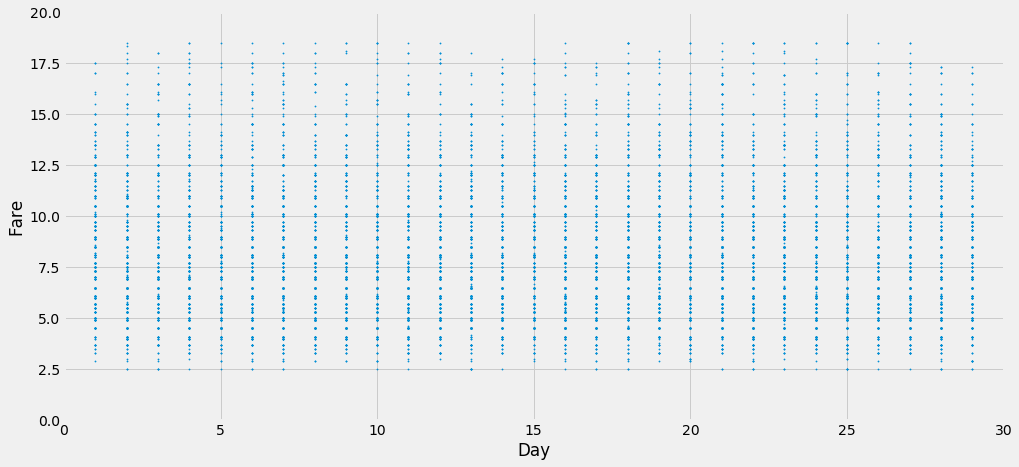
plt.scatter(x=Cab\_train['pickup\_day'], y=Cab\_train['fare\_amount'], s=1.5)

plt.xlabel('Day')

plt.ylabel('Fare')

plt.axis([0, 30, 0, 20])

plt.show()

****

##pickup\_day

trips\_day=pd.DataFrame(Cab\_train['pickup\_day'].value\_counts())

trips\_day=trips\_day.reset\_index()

trips\_day=trips\_day.rename(columns={'index':'pickup\_day','pickup\_day':'Num\_Trips'})

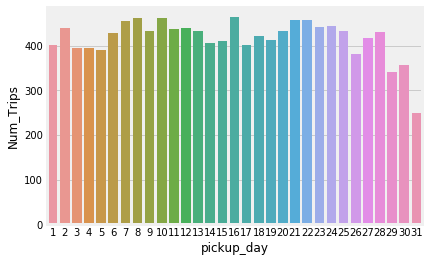
sns.barplot(x='pickup\_day',y='Num\_Trips',data=trips\_day)

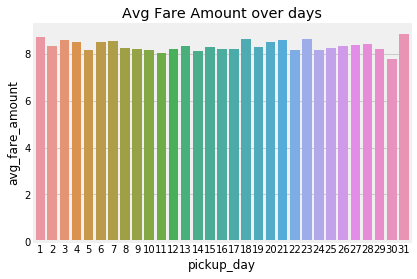
plt.show()

trips\_day\_fareamount=Cab\_train.groupby(['pickup\_day'])['fare\_amount'].mean().reset\_index().rename(columns={'fare\_amount':'avg\_fare\_amount'})

sns.barplot(x='pickup\_day',y='avg\_fare\_amount',data=trips\_day\_fareamount).set\_title("Avg Fare Amount over days")

plt.show()

****

****

# Calculate distance the cab travelled from pickup and dropoff location using great\_circle from geopy library

data = [Cab\_train, Cab\_test]

for i in data:

i['great\_circle']=i.apply(lambda x: great\_circle((x['pickup\_latitude'],x['pickup\_longitude']), (x['dropoff\_latitude'], x['dropoff\_longitude'])).miles, axis=1)

i['geodesic']=i.apply(lambda x: geodesic((x['pickup\_latitude'],x['pickup\_longitude']), (x['dropoff\_latitude'], x['dropoff\_longitude'])).miles, axis=1)

Cab\_train.describe()

|  | **fare\_amount** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **pickup\_day** | **pickup\_hour** | **pickup\_day\_of\_week** | **pickup\_month** | **pickup\_year** | **late\_night** | **great\_circle** | **geodesic** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 12934.000000 | 12934.000000 | 12934.000000 | 12934.000000 | 12934.000000 | 12934.000000 | 12934.000000 | 12934.000000 | 12934.000000 | 12934.000000 | 12934.000000 | 12934.000000 | 12934.000000 | 12934.000000 |
| **mean** | 8.333746 | -73.981747 | 40.753049 | -73.980880 | 40.753515 | 1.648137 | 15.677362 | 13.671486 | 4.011288 | 6.217102 | 2011.684862 | 0.351631 | 1.395957 | 1.395887 |
| **std** | 3.447371 | 0.016008 | 0.021059 | 0.016536 | 0.021860 | 1.267014 | 8.692780 | 6.342838 | 1.959348 | 3.452771 | 1.861672 | 0.477498 | 0.919137 | 0.918645 |
| **min** | 2.500000 | -74.018108 | 40.692830 | -74.019535 | 40.692483 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 1.000000 | 2009.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 5.700000 | -73.992737 | 40.738713 | -73.991953 | 40.739216 | 1.000000 | 8.000000 | 9.000000 | 2.000000 | 3.000000 | 2010.000000 | 0.000000 | 0.723547 | 0.724068 |
| **50%** | 7.700000 | -73.982636 | 40.753610 | -73.982016 | 40.754677 | 1.000000 | 16.000000 | 14.000000 | 4.000000 | 6.000000 | 2012.000000 | 0.000000 | 1.165315 | 1.166058 |
| **75%** | 10.500000 | -73.971350 | 40.766820 | -73.970679 | 40.767386 | 2.000000 | 23.000000 | 19.000000 | 6.000000 | 9.000000 | 2013.000000 | 1.000000 | 1.846101 | 1.845964 |
| **max** | 18.500000 | -73.931787 | 40.811077 | -73.929738 | 40.812595 | 6.000000 | 31.000000 | 23.000000 | 7.000000 | 12.000000 | 2015.000000 | 1.000000 | 6.164333 | 6.158285 |

#Checking outliers in geodesic

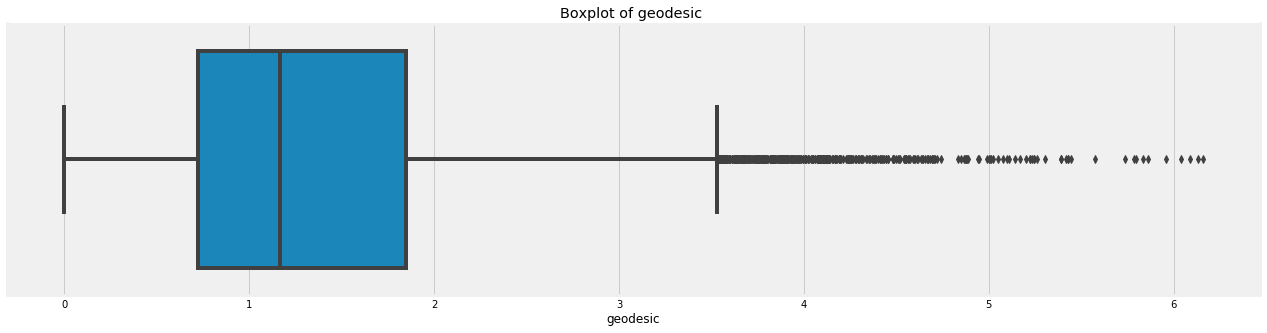
plt.figure(figsize=(20,5))

sns.boxplot(x=Cab\_train['geodesic'],data=Cab\_train,orient='h')

plt.title('Boxplot of geodesic ')

# plt.savefig('bp geodesic.png')

plt.show()

****

# #Detect and delete outliers from data

q75, q25 = np.percentile(Cab\_train.loc[:,'geodesic'], [75 ,25])

iqr = q75 - q25

min = q25 - (iqr\*1.5)

max = q75 + (iqr\*1.5)

print(min)

print(max)

Cab\_train = Cab\_train.drop(Cab\_train[Cab\_train.loc[:,'geodesic'] < min].index)

Cab\_train = Cab\_train.drop(Cab\_train[Cab\_train.loc[:,'geodesic'] > max].index)

-0.9587762759928581

3.5288073949811958

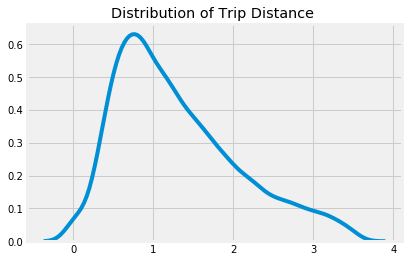
#Plotting the distribution of Distance covered

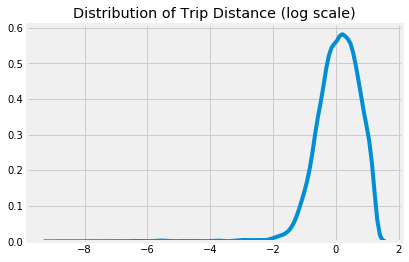
sns.kdeplot((Cab\_train['geodesic'].values)).set\_title("Distribution of Trip Distance ")

plt.show()

sns.kdeplot(np.log(Cab\_train['geodesic'].values)).set\_title("Distribution of Trip Distance (log scale)")

plt.show()

****

****

#Plot of Distance VS Fare

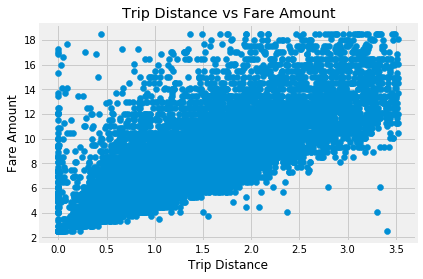
plt.scatter(x=Cab\_train['geodesic'],y=Cab\_train['fare\_amount'])

plt.xlabel("Trip Distance")

plt.ylabel("Fare Amount")

plt.title("Trip Distance vs Fare Amount")

plt.show()

****

df2=Cab\_train.copy()

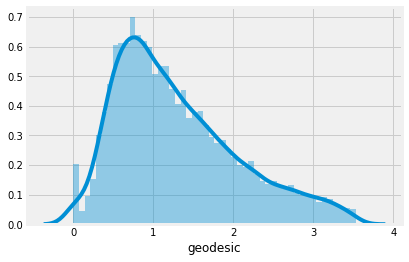
dfT=Cab\_test.copy()

#cab\_train=df2.copy()

sns.distplot(Cab\_train['geodesic'],bins=50)

# plt.savefig('distplot.png')

<matplotlib.axes.\_subplots.AxesSubplot at 0x81a9b70>

****

import scipy.stats as stats

plt.figure()

stats.probplot(Cab\_train['geodesic'], dist='norm', fit=True,plot=plt)

# plt.savefig('qq prob plot.png')

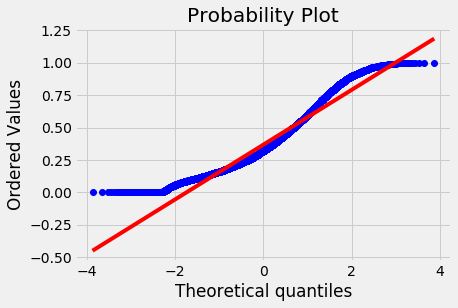
((array([-3.86506329, -3.64290312, -3.52107171, ..., 3.52107171,

3.64290312, 3.86506329]),

array([0. , 0. , 0. , ..., 0.99860303, 0.99871514,

1. ])),

(0.21176140857217782, 0.3679344582637862, 0.9715741545543277))

****

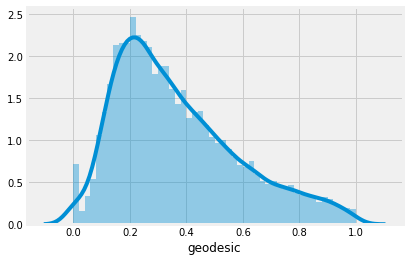
#Normalization

Cab\_train['geodesic'] = (Cab\_train['geodesic'] - np.min(Cab\_train['geodesic']))/(np.max(Cab\_train['geodesic']) - np.min(Cab\_train['geodesic']))

Cab\_test['geodesic'] = (Cab\_test['geodesic'] - np.min(Cab\_test['geodesic']))/(np.max(Cab\_test['geodesic']) - np.min(Cab\_test['geodesic']))

sns.distplot(Cab\_train['geodesic'],bins=50)

<matplotlib.axes.\_subplots.AxesSubplot at 0x80ae550>

****

Cab\_train['geodesic'].var()

0.047481249795667466

#Cab\_train = Cab\_train.drop(['pickup\_datetime'], axis=1)

Cab\_train = Cab\_train.drop(['pickup\_date'],axis=1)

#Drop unnecessary variables and divide Training set into train and validation set

#Cab\_train = Cab\_train.drop(['pickup\_datetime'], axis=1)

#Cab\_train = Cab\_train.drop(['pickup\_date']

y = Cab\_train['fare\_amount']

X = Cab\_train.drop(['fare\_amount'],axis = 1)

###Drop columns on test data

#Cab\_test = Cab\_test.drop(['pickup\_datetime'],axis = 1)

Cab\_test = Cab\_test.drop(['pickup\_date'],axis=1)

Cab\_train.shape

(12482, 14)

#25% Validation set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,random\_state=42, test\_size=0.25)

Cab\_train.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 12482 entries, 2 to 16065

Data columns (total 14 columns):

fare\_amount 12482 non-null float64

pickup\_longitude 12482 non-null float64

pickup\_latitude 12482 non-null float64

dropoff\_longitude 12482 non-null float64

dropoff\_latitude 12482 non-null float64

passenger\_count 12482 non-null int64

pickup\_day 12482 non-null int64

pickup\_hour 12482 non-null int64

pickup\_day\_of\_week 12482 non-null int64

pickup\_month 12482 non-null int64

pickup\_year 12482 non-null int64

late\_night 12482 non-null int64

great\_circle 12482 non-null float64

geodesic 12482 non-null float64

dtypes: float64(7), int64(7)

memory usage: 1.4 MB

# Model Development

def rmsle(y,y\_):

log1 = np.nan\_to\_num(np.array([np.log(v + 1) for v in y]))

log2 = np.nan\_to\_num(np.array([np.log(v + 1) for v in y\_]))

calc = (log1 - log2) \*\* 2

return np.sqrt(np.mean(calc))

def scores(y, y\_):

print('r square ', metrics.r2\_score(y, y\_))

print('Adjusted r square:{}'.format(1 - (1-metrics.r2\_score(y, y\_))\*(len(y)-1)/(len(y)-X\_train.shape[1]-1)))

print('MAPE:{}'.format(np.mean(np.abs((y - y\_) / y))\*100))

print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y, y\_)))

def test\_scores(model):

print('<<<------------------- Training Data Score --------------------->')

print()

#Predicting result on Training data

y\_pred = model.predict(X\_train)

scores(y\_train,y\_pred)

print()

print('<<<------------------- Test Data Score --------------------->')

print()

# Evaluating on Test Set

y\_pred = model.predict(X\_test)

scores(y\_test,y\_pred)

## Decision Tree Regression

# Setup the parameters and distributions to sample from: param\_dist

param\_dist = {'max\_depth': range(2,16,2),

'min\_samples\_split': range(2,16,2)}

# Instantiate a Decision Tree classifier: tree

tree = DecisionTreeRegressor()

# Instantiate the gridSearchCV object: tree\_cv

tree\_cv = GridSearchCV(tree, param\_dist, cv=5)

# Fit it to the data

tree\_cv.fit(X, y)

# Print the tuned parameters and score

print("Tuned Decision Tree Parameters: {}".format(tree\_cv.best\_params\_))

print("Best score is {}".format(tree\_cv.best\_score\_))

Tuned Decision Tree Parameters: {'max\_depth': 6, 'min\_samples\_split': 4}

Best score is 0.6378003594889797

# Instantiate a tree regressor: tree

tree = DecisionTreeRegressor(max\_depth= 6, min\_samples\_split=4)

# Fit the regressor to the data

tree.fit(X\_train,y\_train)

# Compute and print the coefficients

tree\_features = tree.feature\_importances\_

print(tree\_features)

# Sort test importances in descending order

indices = np.argsort(tree\_features)[::1]

# Rearrange test names so they match the sorted test importances

names = [Cab\_test.columns[i] for i in indices]

# Creating plot

fig = plt.figure(figsize=(20,10))

plt.title("test Importance")

# Add horizontal bars

plt.barh(range(pd.DataFrame(X\_train).shape[1]),tree\_features[indices],align = 'center')

plt.yticks(range(pd.DataFrame(X\_train).shape[1]), names)

plt.savefig('tree test importance')

plt.show()

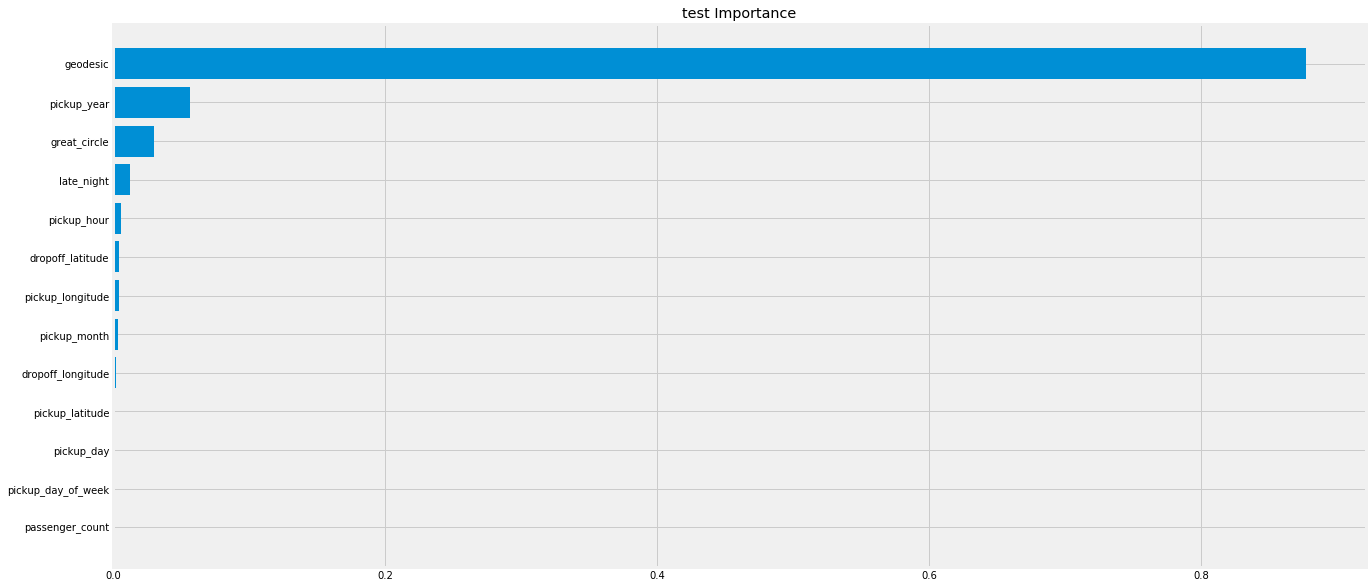
# Make predictions and cal error

test\_scores(tree)

[0.00435383 0.0009874 0.00208352 0.00480357 0. 0.00091674

0.0059383 0. 0.00354377 0.05702772 0.0126548 0.03036813

0.87732222]

****

<<<------------------- Training Data Score --------------------->

r square 0.6759851252920315

Adjusted r square:0.6755344787347186

MAPE:17.24420035961592

RMSE: 1.8182678215178951

<<<------------------- Test Data Score --------------------->

r square 0.6391940260795843

Adjusted r square:0.6376843776531391

MAPE:17.994384710217158

RMSE: 1.9336111504059643

## Linear Regression

# Setup the parameters and distributions to sample from: param\_dist

param\_dist = {'copy\_X':[True, False],

'fit\_intercept':[True,False]}

# Instantiate a Decision reg classifier: reg

reg = LinearRegression()

# Instantiate the gridSearchCV object: reg\_cv

reg\_cv = GridSearchCV(reg, param\_dist, cv=5,scoring='r2')

# Fit it to the data

reg\_cv.fit(X, y)

# Print the tuned parameters and score

print("Tuned Decision reg Parameters: {}".format(reg\_cv.best\_params\_))

print("Best score is {}".format(reg\_cv.best\_score\_))

Tuned Decision reg Parameters: {'copy\_X': True, 'fit\_intercept': True}

Best score is 0.6495842113491038

# Create the regressor: reg\_all

reg\_all = LinearRegression(copy\_X= True, fit\_intercept=True)

# Fit the regressor to the training data

reg\_all.fit(X\_train,y\_train)

# Predict on the test data: y\_pred

y\_pred = reg\_all.predict(X\_test)

# Compute and print R^2 and RMSE

print("R^2: {}".format(reg\_all.score(X\_test, y\_test)))

rmse = np.sqrt(mean\_squared\_error(y\_test,y\_pred))

print("Root Mean Squared Error: {}".format(rmse))

test\_scores(reg\_all)

# Compute and print the coefficients

reg\_coef = reg\_all.coef\_

print(reg\_coef)

# Plot the coefficients

plt.figure(figsize=(15,5))

plt.plot(range(len(Cab\_test.columns)), reg\_coef)

plt.xticks(range(len(Cab\_test.columns)), Cab\_test.columns.values, rotation=60)

plt.margins(0.02)

plt.savefig('linear coefficients')

plt.show()

R^2: 0.6488035889552841

Root Mean Squared Error: 1.9076878625659281

<<<------------------- Training Data Score --------------------->

r square 0.6509166970439839

Adjusted r square:0.6504311848006514

MAPE:17.702833767265542

RMSE: 1.8872955556283124

<<<------------------- Test Data Score --------------------->

r square 0.6488035889552841

Adjusted r square:0.6473341479048877

MAPE:17.92035141825923

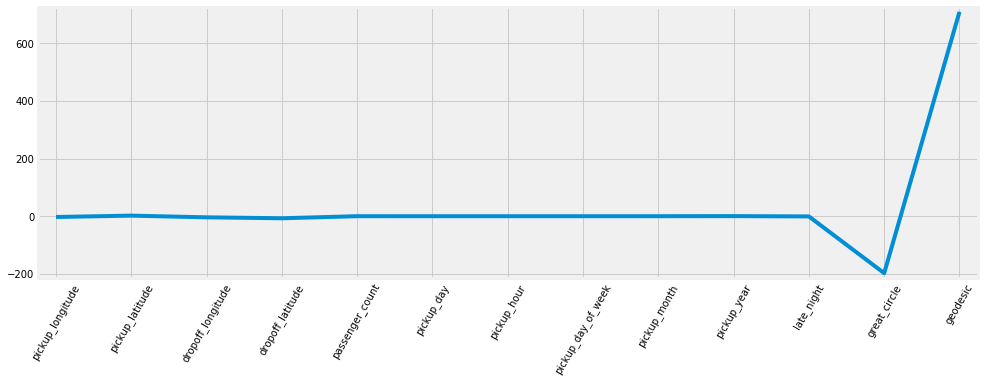
RMSE: 1.9076878625659281

[-2.70678267e+00 1.96591877e+00 -3.88307790e+00 -7.16360499e+00

5.10266990e-02 2.07277436e-03 1.75873709e-02 -1.71608003e-02

4.64318316e-02 3.26702486e-01 -7.62660901e-01 -1.97626535e+02

7.08920724e+02]

****

## Random Forest

# Create the random grid

random\_grid = {'n\_estimators': range(100,500,100),

'max\_depth': range(5,20,1),

'min\_samples\_leaf':range(2,5,1),

'max\_features':['auto','sqrt','log2'],

'bootstrap': [True, False],

'min\_samples\_split': range(2,5,1)}

# Instantiate a Decision Forest classifier: Forest

Forest = RandomForestRegressor()

# Instantiate the gridSearchCV object: Forest\_cv

Forest\_cv = RandomizedSearchCV(Forest, random\_grid, cv=5)

# Fit it to the data

Forest\_cv.fit(X, y)

# Print the tuned parameters and score

print("Tuned Random Forest Parameters: {}".format(Forest\_cv.best\_params\_))

print("Best score is {}".format(Forest\_cv.best\_score\_))

Tuned Random Forest Parameters: {'n\_estimators': 300, 'min\_samples\_split': 4, 'min\_samples\_leaf': 3, 'max\_features': 'sqrt', 'max\_depth': 18, 'bootstrap': False}

Best score is 0.6873276470522499

# Instantiate a Forest regressor: Forest

Forest = RandomForestRegressor(n\_estimators=300, min\_samples\_split= 4, min\_samples\_leaf=3, max\_features='sqrt', max\_depth=18, bootstrap=False)

# Fit the regressor to the data

Forest.fit(X\_train,y\_train)

# Compute and print the coefficients

Forest\_features = Forest.feature\_importances\_

print(Forest\_features)

# Sort feature importances in descending order

indices = np.argsort(Forest\_features)[::1]

# Rearrange feature names so they match the sorted feature importances

names = [Cab\_test.columns[i] for i in indices]

# Creating plot

fig = plt.figure(figsize=(20,10))

plt.title("Feature Importance")

# Add horizontal bars

plt.barh(range(pd.DataFrame(X\_train).shape[1]),Forest\_features[indices],align = 'center')

plt.yticks(range(pd.DataFrame(X\_train).shape[1]), names)

plt.savefig('Random forest feature importance')

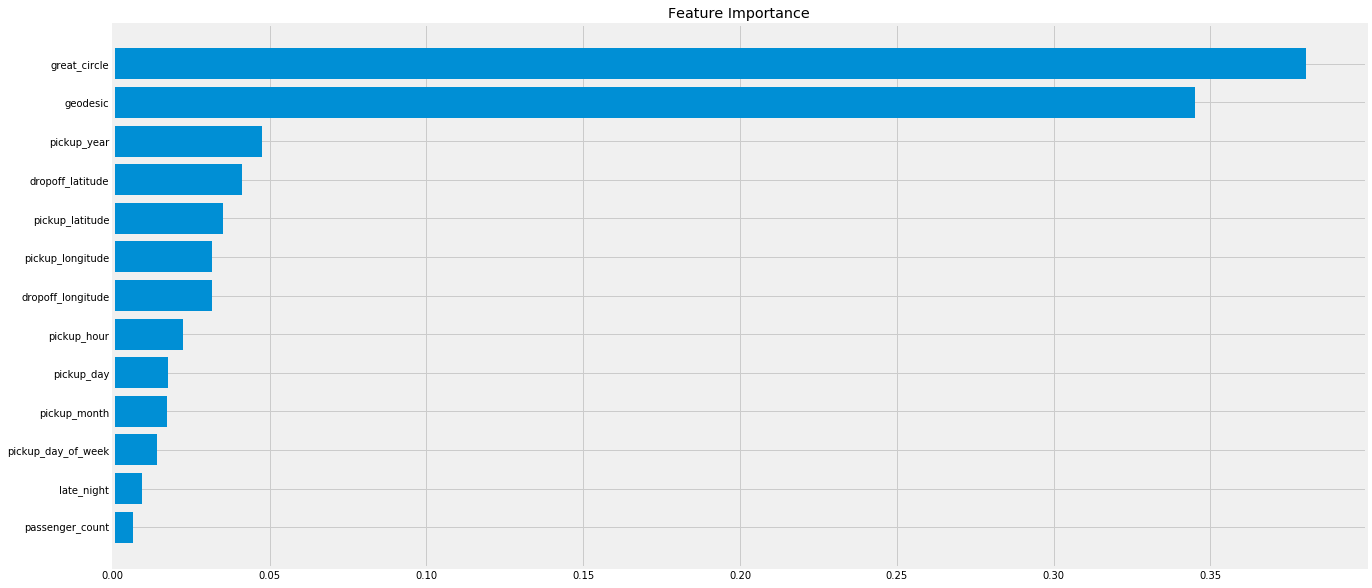
plt.show()# Make predictions

test\_scores(Forest)

[0.03164039 0.03502175 0.03163833 0.04112095 0.00635663 0.01769075

0.02249496 0.01420423 0.01725168 0.04763963 0.00928393 0.38046165

0.34519513]

****

<<<------------------- Training Data Score --------------------->

r square 0.9397106332444995

Adjusted r square:0.9396267815522109

MAPE:6.744788724206718

RMSE: 0.7843244255515974

<<<------------------- Test Data Score --------------------->

r square 0.6804525962598367

Adjusted r square:0.6791155778341456

MAPE:16.90922917401351

RMSE: 1.8197005597982328

## XGBOOST

# Create the random grid

para = {'n\_estimators': range(100,500,100),

'max\_depth': range(3,10,1),

'reg\_alpha':np.logspace(-4, 0, 50),

'subsample': np.arange(0.1,1,0.2),

'colsample\_bytree': np.arange(0.1,1,0.2),

'colsample\_bylevel': np.arange(0.1,1,0.2),

'colsample\_bynode': np.arange(0.1,1,0.2),

'learning\_rate': np.arange(.05, 1, .05)}

# Instantiate a Decision Forest classifier: Forest

Xgb = XGBRegressor()

# Instantiate the gridSearchCV object: Forest\_cv

xgb\_cv = RandomizedSearchCV(Xgb, para, cv=5)

# Fit it to the data

xgb\_cv.fit(X, y)

# Print the tuned parameters and score

print("Tuned Xgboost Parameters: {}".format(xgb\_cv.best\_params\_))

print("Best score is {}".format(xgb\_cv.best\_score\_))

[16:39:26] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:39:29] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:39:33] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:39:36] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:39:40] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:39:46] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:39:49] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:39:54] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:01] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:09] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:15] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:17] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:19] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:20] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:21] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:23] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:28] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:33] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:39] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:44] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:49] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:51] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:54] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:56] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:40:58] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:00] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:01] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:02] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:03] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:04] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:05] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:08] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:11] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:14] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:18] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:20] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:22] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:23] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:25] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:27] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:28] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:29] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:30] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:31] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:32] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:33] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:34] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:37] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:38] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:40] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[16:41:42] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

Tuned Xgboost Parameters: {'subsample': 0.9000000000000001, 'reg\_alpha': 0.0011513953993264468, 'n\_estimators': 400, 'max\_depth': 5, 'learning\_rate': 0.15000000000000002, 'colsample\_bytree': 0.7000000000000001, 'colsample\_bynode': 0.5000000000000001, 'colsample\_bylevel': 0.5000000000000001}

Best score is 0.7044461748943669

# Instantiate a xgb regressor: xgb

Xgb = XGBRegressor(subsample= 0.9000000000000001, reg\_alpha= 0.0011513953993264468, n\_estimators= 400, max\_depth= 5, learning\_rate= 0.15000000000000002,

colsample\_bytree= 0.7000000000000001, colsample\_bynode=0.5000000000000001, colsample\_bylevel=0.5000000000000001)

# Fit the regressor to the data

Xgb.fit(X\_train,y\_train)

# Compute and print the coefficients

xgb\_features = Xgb.feature\_importances\_

print(xgb\_features)

# Sort feature importances in descending order

indices = np.argsort(xgb\_features)[::1]

# Rearrange feature names so they match the sorted feature importances

names = [Cab\_test.columns[i] for i in indices]

# Creating plot

fig = plt.figure(figsize=(20,10))

plt.title("Feature Importance")

# Add horizontal bars

plt.barh(range(pd.DataFrame(X\_train).shape[1]),xgb\_features[indices],align = 'center')

plt.yticks(range(pd.DataFrame(X\_train).shape[1]), names)

plt.savefig(' xgb feature importance')

plt.show()# Make predictions

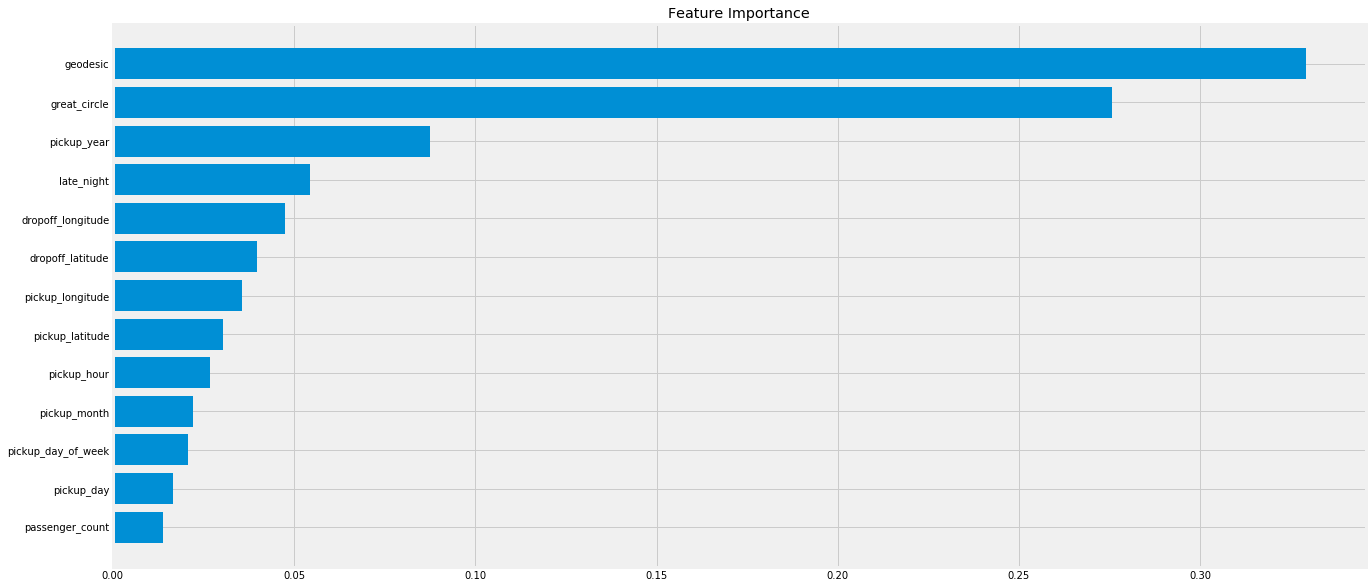
test\_scores(Xgb)

[16:43:53] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[0.03558831 0.03031121 0.04751259 0.03965991 0.01379091 0.01648034

0.02691714 0.02087018 0.02207658 0.08754644 0.05452408 0.27554253

0.32917985]

****

<<<------------------- Training Data Score --------------------->

r square 0.8973133744273231

Adjusted r square:0.8971705557547602

MAPE:9.793984328712876

RMSE: 1.0236048152025392

<<<------------------- Test Data Score --------------------->

r square 0.7022548565324075

Adjusted r square:0.7010090609530453

MAPE:16.470292626182633

RMSE: 1.756526172479898

# Finalize model

Create standalone model on entire training dataset and Save model for later use

def rmsle(y,y\_):

log1 = np.nan\_to\_num(np.array([np.log(v + 1) for v in y]))

log2 = np.nan\_to\_num(np.array([np.log(v + 1) for v in y\_]))

calc = (log1 - log2) \*\* 2

return np.sqrt(np.mean(calc))

def score(y, y\_):

print('r square ', metrics.r2\_score(y, y\_))

print('Adjusted r square:{}'.format(1 - (1-metrics.r2\_score(y, y\_))\*(len(y)-1)/(len(y)-X\_train.shape[1]-1)))

print('MAPE:{}'.format(np.mean(np.abs((y - y\_) / y))\*100))

print('MSE:', metrics.mean\_squared\_error(y, y\_))

print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y, y\_)))

def scores(model):

print('<<<------------------- Training Data Score --------------------->')

print()

#Predicting result on Training data

y\_pred = model.predict(X)

score(y,y\_pred)

print('RMSLE:',rmsle(y,y\_pred))

test\_data=pd.read\_csv('test.csv')

test\_pickup\_datetime=test\_data['pickup\_datetime']

# Instantiate a xgb regressor: xgb

Xgb = XGBRegressor(subsample= 0.9000000000000001, reg\_alpha= 0.0011513953993264468, n\_estimators= 400, max\_depth= 5, learning\_rate= 0.15000000000000002,

colsample\_bytree= 0.7000000000000001, colsample\_bynode=0.5000000000000001, colsample\_bylevel=0.5000000000000001)

# Fit the regressor to the data

Xgb.fit(X,y)

# Compute and print the coefficients

xgb\_features = Xgb.feature\_importances\_

print(xgb\_features)

# Sort feature importances in descending order

indices = np.argsort(xgb\_features)[::1]

# Rearrange feature names so they match the sorted feature importances

names = [Cab\_test.columns[i] for i in indices]

# Creating plot

fig = plt.figure(figsize=(20,10))

plt.title("Feature Importance")

# Add horizontal bars

plt.barh(range(pd.DataFrame(X\_train).shape[1]),xgb\_features[indices],align = 'center')

plt.yticks(range(pd.DataFrame(X\_train).shape[1]), names)

plt.savefig(' xgb1 feature importance')

plt.show()

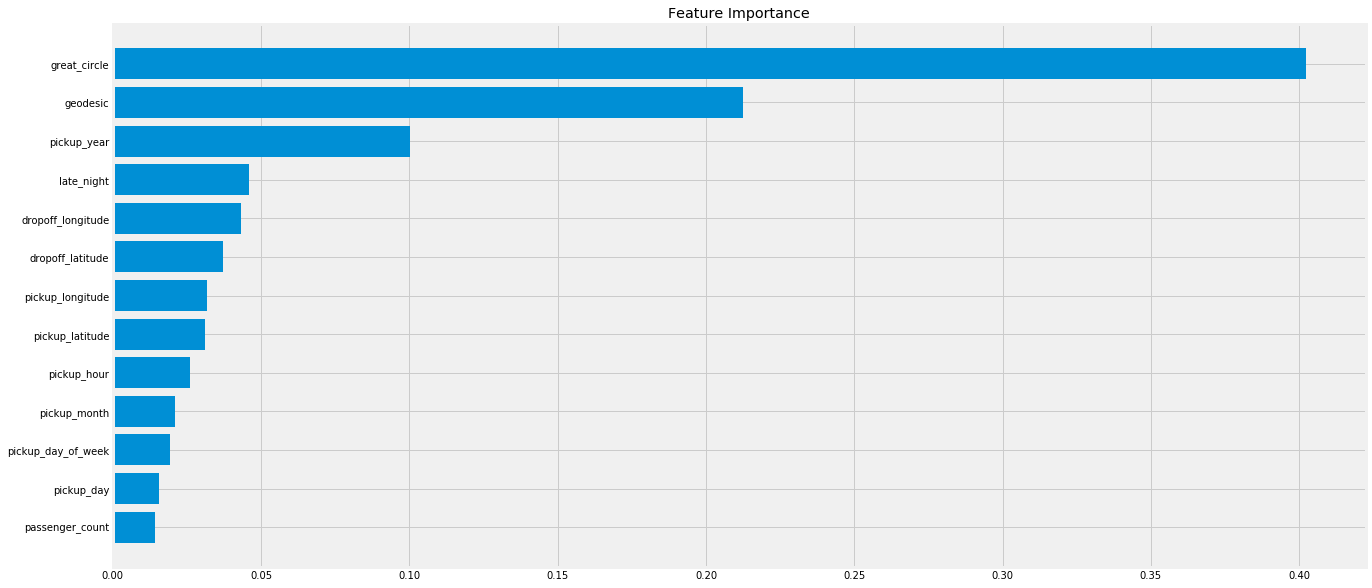
scores(Xgb)

[16:50:25] WARNING: C:/Jenkins/workspace/xgboost-win64\_release\_0.90/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

[0.03163542 0.03105691 0.04312714 0.03716621 0.01413951 0.01558933

0.02593329 0.01943849 0.02088184 0.10014623 0.04591875 0.40232328

0.21264356]

****

<<<------------------- Training Data Score --------------------->

r square 0.8769807106603733

Adjusted r square:0.8768524422322842

MAPE:10.72824650048165

MSE: 1.2601635015105397

RMSE: 1.1225700430309637

RMSLE: 0.12174482797619744

Thank you

**Reference:**

<https://learning.edwisor.com/>

<https://matplotlib.org/index.html>

<https://stackoverflow.com/>

<https://towardsdatascience.com/>