

“Cab Fare Prediction”

(A data science project report)

**Submitted By**

**Arun Kumar**

**arunk.recs.cse@gmail.com**

**To**

**edWisor.com**

**Contents**

1. Introduction …………………………… 3
   1. Problem Statement …………………………… 3
   2. Data …………………………… 3
2. Methodology …………………………… 5

2.1. Data Pre-processing …………………………… 5

2.1.0. Exploratory Data Analysis …………………………… 5

2.1.1. Missing Value Analysis …………………………… 6

2.1.2. Outlier Analysis …………………………… 7

2.1.3. Feature Engineering …………………………… 8

2.1.4. Feature Selection ……………………………. 11

2.1.5. Feature Scaling ……………………………. 15

2.1.6. Data after EDA and pre-processing ………………………… 17

2.2. Model Development ……………………………. 18

2.2.0 Linear Regression …………………………….. 18

2.2.1 Ridge Regression …………………………….. 18

2.2.2 Lasso Regression ……………………………... 18

2.2.3 Decision tree regression …………………………….. 18

2.2.4 XGboost …………………………….. 18

1. Conclusion …………………………….. 19

3.1. Model Evaluation ……………………………. 21

3.2. Model Selection …………………………….. 22

1. Codes …………………………….. 25

4.1 Python Code …………………………….. 25

4.2 R code ……………………………... 60

1. Reference ……………………………... 69

Chapter 1

Introduction

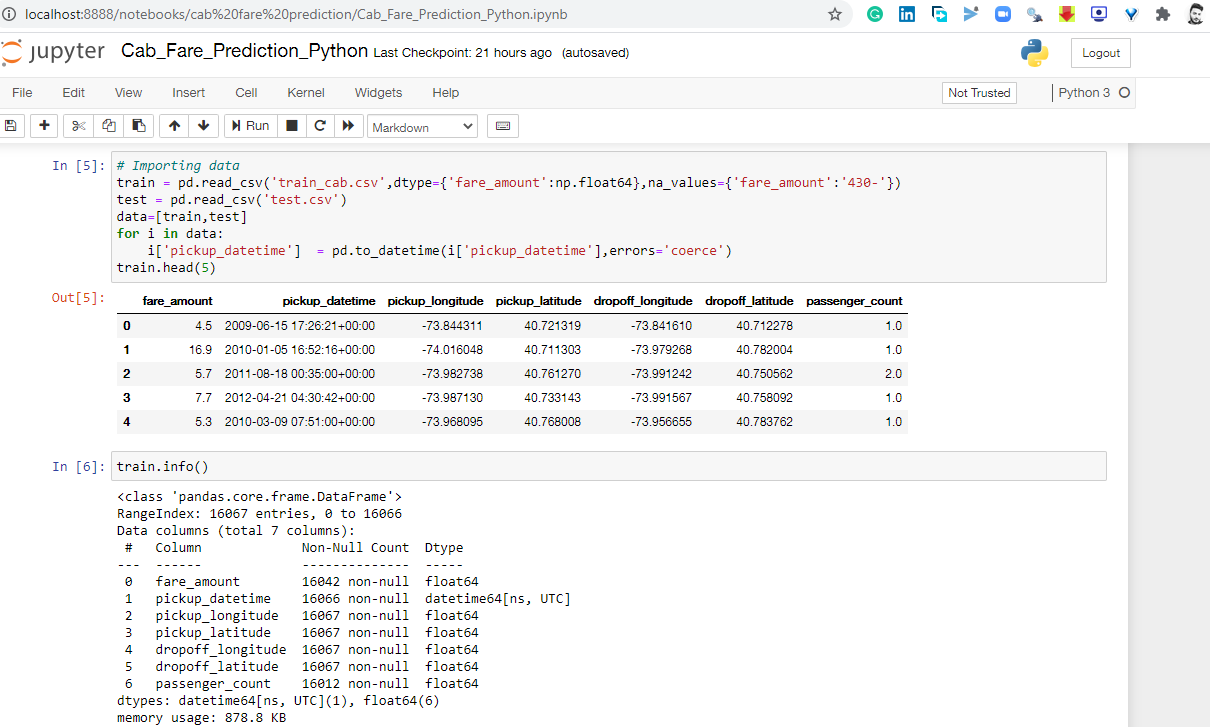


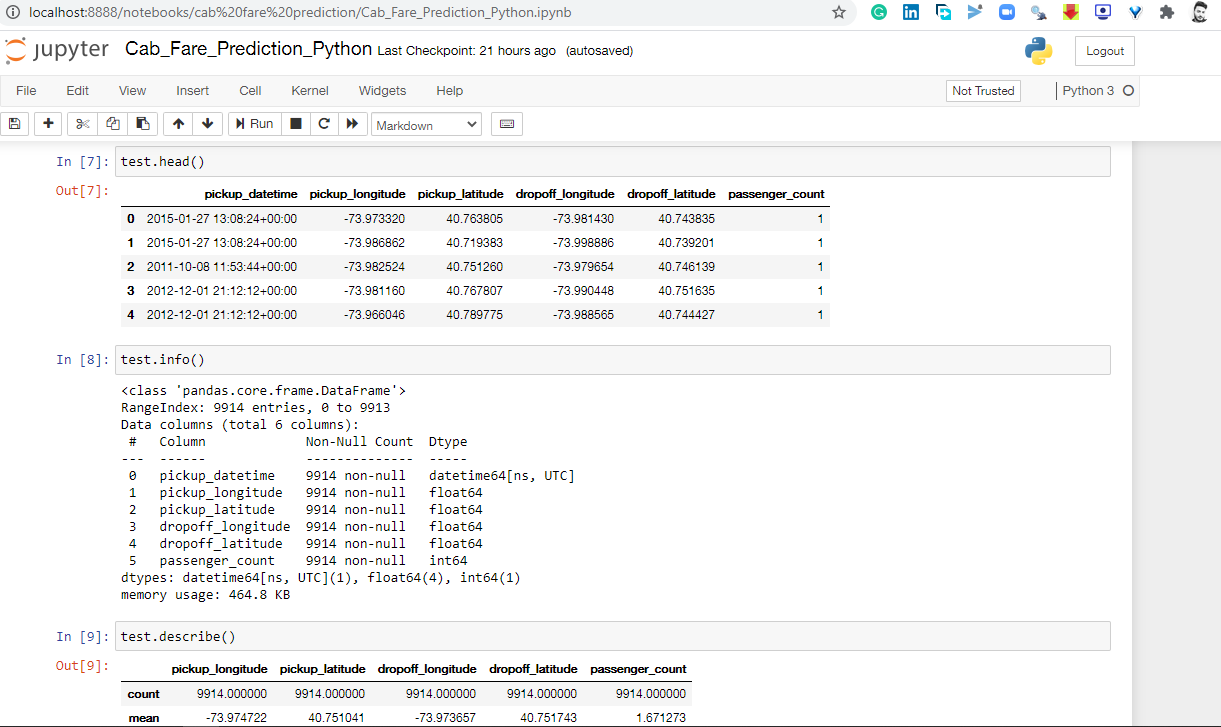
1.1 Problem Statement

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 Data

We have total 7 data columns in train data and 6 data columns in test data. For more details look below pictures.





Attributes:-

* pickup\_datetime - timestamp value indicating when the cab ride started.
* pickup\_longitude - float for longitude coordinate of where the cab rides started.
* pickup\_latitude - float for latitude coordinate of where the cab rides 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.

Chapter 2

Methodology



**2.1 Data Pre-processing**

Data pre-processing is the first stage of any type of project. In this stage we get the feel of the data. We do this by looking at plots of independent variables vs target variables. If the data is messy, we try to improve it by sorting deleting extra rows and columns. This stage is called as Exploratory Data Analysis. This stage generally involves data cleaning, merging, sorting, looking for outlier analysis, looking for missing values in the data, imputing missing values if found by various methods such as mean, median, mode, KNN imputation, etc.

Further we will look into what Pre-Processing steps do this project was involved in.

**2.1.0 Exploratory Data Analysis**

Exploratory Data Analysis (EDA) is an approach to analysing data sets and

Summarize their main characteristics. Our train data consists 16067 observation and 7 variables. Data type of all variables is either int64 or float64 or datetime64. As per the data analysis we have to find which variables are the categorical variables, continuous variables and target variable. Data types need to be change accordingly. We have distributed the variables on the basis of continuous and categorical variables. Target variable is continuous. We have total 16067 observation, but as per above summary tables total observation is <16067 in some variables. Its means there is missing values present in our dataset. Missing value analysis is required to further understand the data.

Train data summery

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

RangeIndex: 16067 entries, 0 to 16066

Data columns (total 7 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 fare\_amount 16042 non-null float64

1 pickup\_datetime 16066 non-null datetime64[ns, UTC]

2 pickup\_longitude 16067 non-null float64

3 pickup\_latitude 16067 non-null float64

4 dropoff\_longitude 16067 non-null float64

5 dropoff\_latitude 16067 non-null float64

6 passenger\_count 16012 non-null float64

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

memory usage: 878.8 KB

test data summery

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

RangeIndex: 9914 entries, 0 to 9913

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 pickup\_datetime 9914 non-null datetime64[ns, UTC]

1 pickup\_longitude 9914 non-null float64

2 pickup\_latitude 9914 non-null float64

3 dropoff\_longitude 9914 non-null float64

4 dropoff\_latitude 9914 non-null float64

5 passenger\_count 9914 non-null int64

dtypes: datetime64[ns, UTC](1), float64(4), int64(1)

memory usage: 464.8 KB

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

1. Fare amount has a negative value, which doesn't make sense. A price amount cannot be -ve and also cannot be 0. So we will remove these fields.
2. Passenger\_count variable drop where passenger count less than 0 or greater than 6.
3. Latitudes range from -90 to 90.Longitudes range from -180 to 180. Removing which does not satisfy these ranges. And there is only one value and just drop it.

So, we lost 16067-15661 = 406 observations because of non-sensual values.

## 2.1.1 Missing Value Analysis

In this step we look for missing values in the dataset like empty row column cell which was left after removing special characters and punctuation marks. Some missing values are in form of NA. missing values left behind after outlier analysis; missing values can be in any form. Unfortunately, in this dataset we have found some missing values. Therefore, we will do some missing value analysis. Before imputed we selected random row no-1000 and made it NA, so that we will compare original value with imputed value and choose best method which will impute value closer to actual value.

|  | **index** | **0** |
| --- | --- | --- |
| **0** | fare\_amount | 22 |
| **1** | pickup\_datetime | 1 |
| **2** | pickup\_longitude | 0 |
| **3** | pickup\_latitude | 0 |
| **4** | dropoff\_longitude | 0 |
| **5** | dropoff\_latitude | 0 |
| **6** | passenger\_count | 55 |

We will impute values for fare\_amount and passenger\_count both of them has missing values 22 and 55 respectively. We will drop 1 value in pickup\_datetime i.e. it will be an entire row to drop.

Below are the missing value percentage for each variable:

| **Variables** | **Missing\_percentage** |
| --- | --- |
| **0** | passenger\_count | 0.351191 |
| **1** | fare\_amount | 0.140476 |
| **2** | pickup\_datetime | 0.006385 |
| **3** | pickup\_longitude | 0.000000 |
| **4** | pickup\_latitude | 0.000000 |
| **5** | dropoff\_longitude | 0.000000 |
| **6** | dropoff\_latitude | 0.000000 |

## 

**2.1.2. Outlier Analysis**

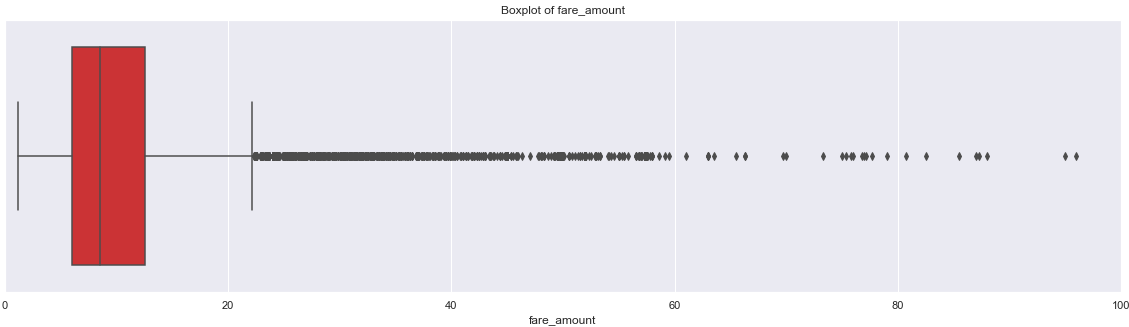
We look for outlier in the dataset by plotting Boxplots. There are outliers present in the data. We have removed these outliers. This is how we done,

I. We replaced them with Nan values or we can say created missing values.

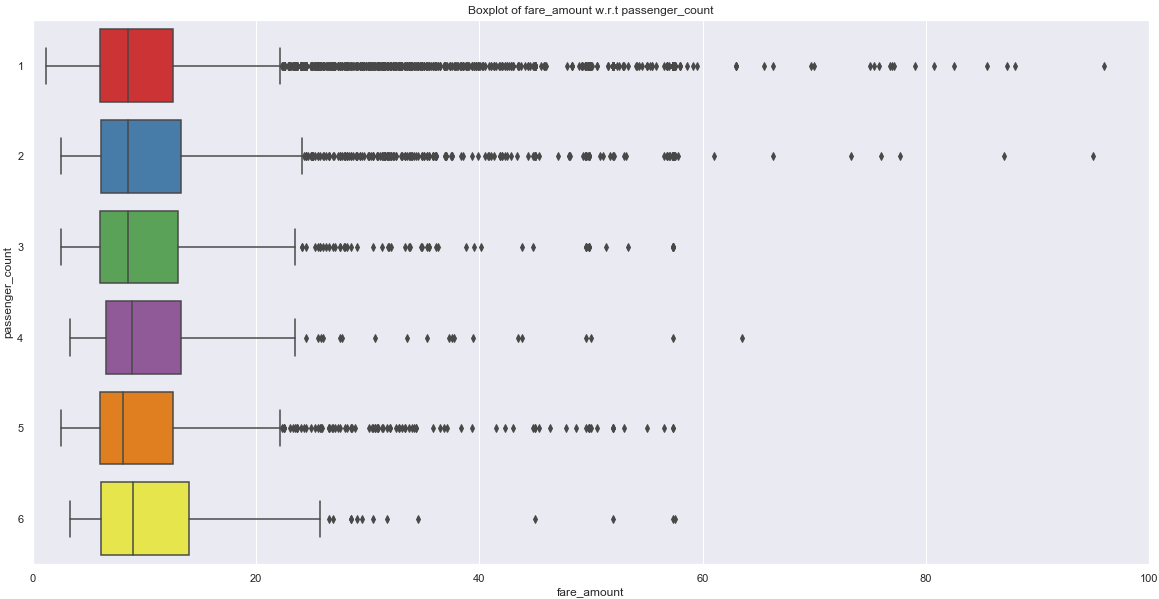
II. Then we imputed those missing values with KNN method.

* We will do Outlier Analysis only on Fare\_amount just for now and we will do outlier analysis after feature engineering latitudes and longitudes.
* Univariate Boxplots: Boxplots for target variable.

Univariate Boxplots: Boxplots for all Numerical Variables also for target variable



Bivariate Boxplots: Boxplots for all fare\_amount Variables Vs all passenger\_count variable.



From above Boxplots we see that ‘fare\_amount ’have outliers in it: ‘fare\_amount’ has 1359 outliers. We successfully imputed these outliers with KNN and K value is 3

**2.1.3. Feature Engineering**

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

1. **For ‘pickup\_datetime’ variable:**

We will use this timestamp variable to create new variables.

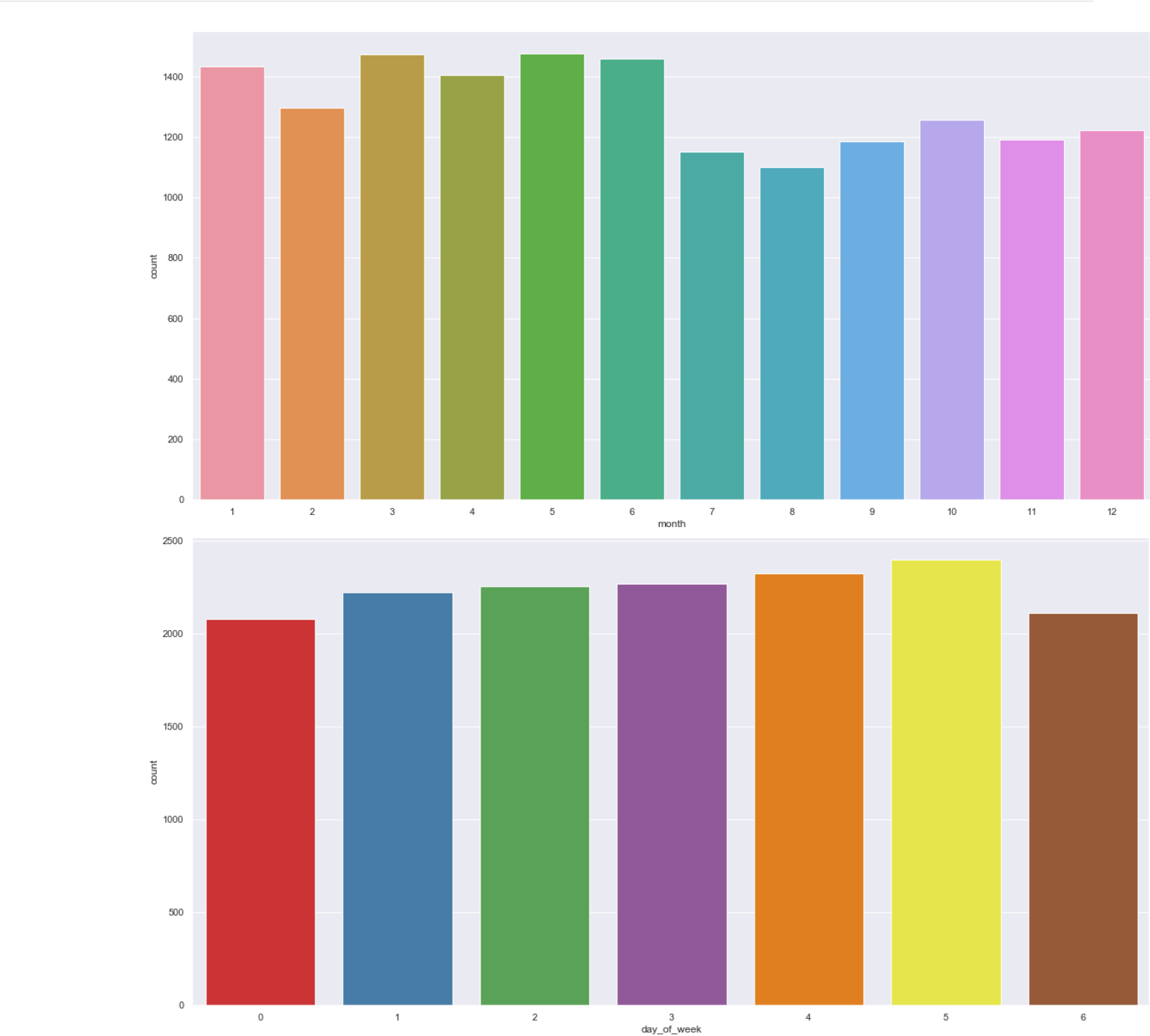
New features will be year, month, day\_of\_week, and hour.

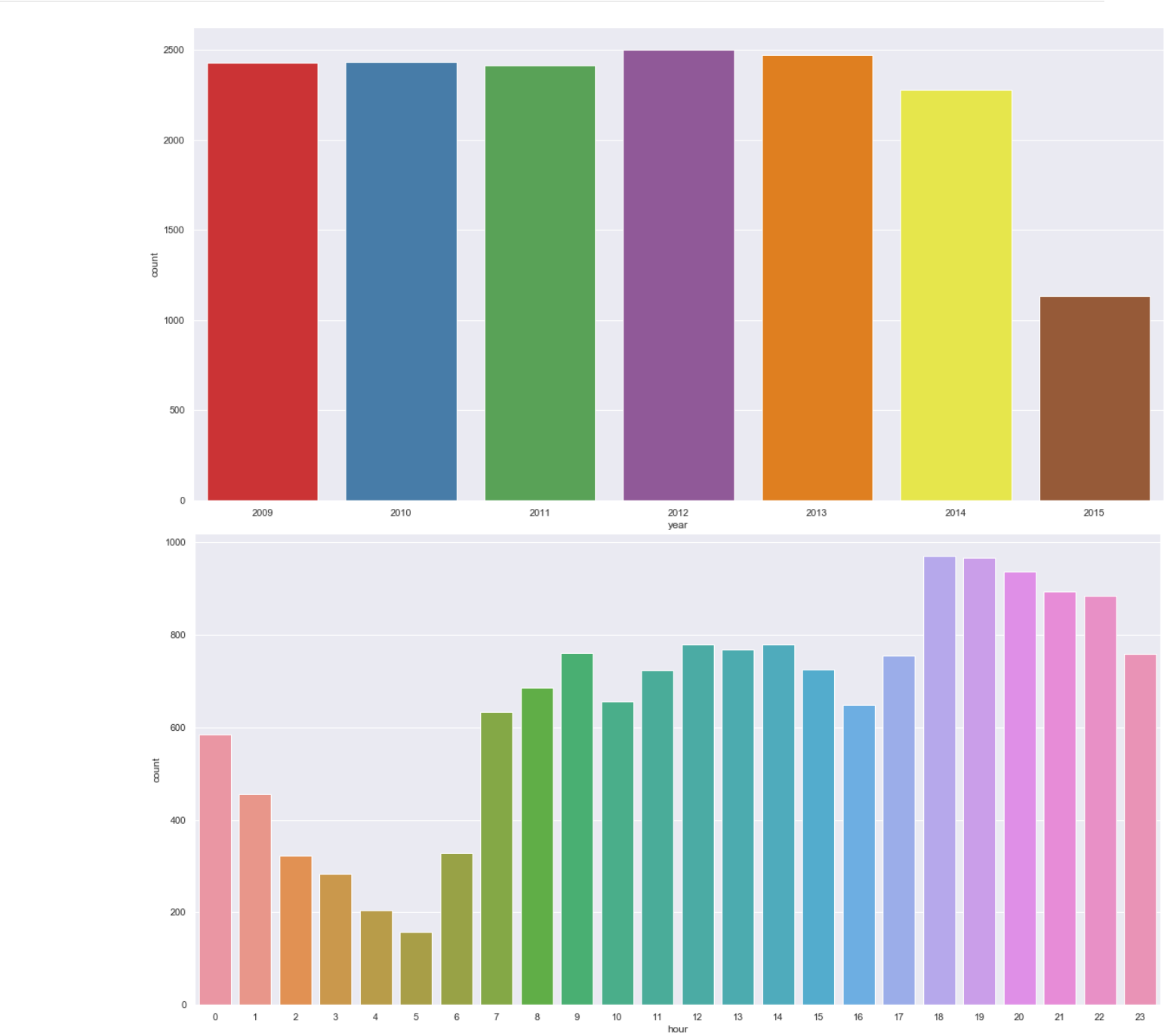
‘year’ will contain only years from pickup\_datetime. For ex. 2009, 2010, 2011, etc.

‘month’ will contain only months from pickup\_datetime. For ex. 1 for January, 2 for February, etc.

‘day\_of\_week’ will contain only week from pickup\_datetime. For ex. 1 which is for Monday,2 for Tuesday,etc.

‘hour’ will contain only hours from pickup\_datetime. For ex. 1, 2, 3, etc.





As we have now these new variables we will categorize them to new variables like Session from hour column, seasons from month column, week:weekday/weekend from day\_of\_week variable.

So, session variable which will contain categories—morning, afternoon, evening, night\_PM, night\_AM.

Seasons variable will contain categories—spring, summer, fall, winter.

Week will contain categories—weekday, weekend.

We will one-hot-encode session, seasons, week variable.

1. **For ‘passenger\_count’ variable:**

As passenger\_count is a categorical variable we will one-hot-encode it.

1. **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.

Columns in training data after feature engineering:

Index(['fare\_amount', 'passenger\_count\_2', 'passenger\_count\_3',

'passenger\_count\_4', 'passenger\_count\_5', 'passenger\_count\_6',

'season\_spring', 'season\_summer', 'season\_winter', 'week\_weekend',

'session\_evening', 'session\_morning', 'session\_night\_AM',

'session\_night\_PM', 'year\_2010', 'year\_2011', 'year\_2012', 'year\_2013',

'year\_2014', 'year\_2015', 'geodesic'],

dtype='object')

Columns in testing data after feature engineering:

Index(['passenger\_count\_2', 'passenger\_count\_3', 'passenger\_count\_4',

'passenger\_count\_5', 'passenger\_count\_6', 'season\_spring',

'season\_summer', 'season\_winter', 'week\_weekend', 'session\_evening',

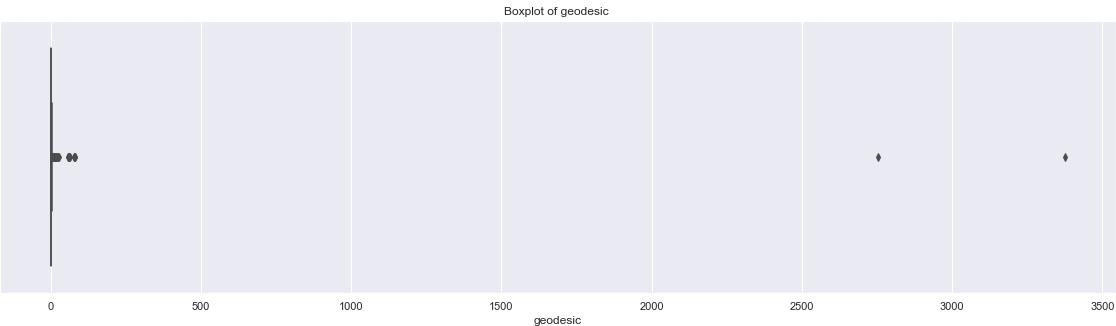
'session\_morning', 'session\_night\_AM', 'session\_night\_PM', 'year\_2010',

'year\_2011', 'year\_2012', 'year\_2013', 'year\_2014', 'year\_2015',

'geodesic'],

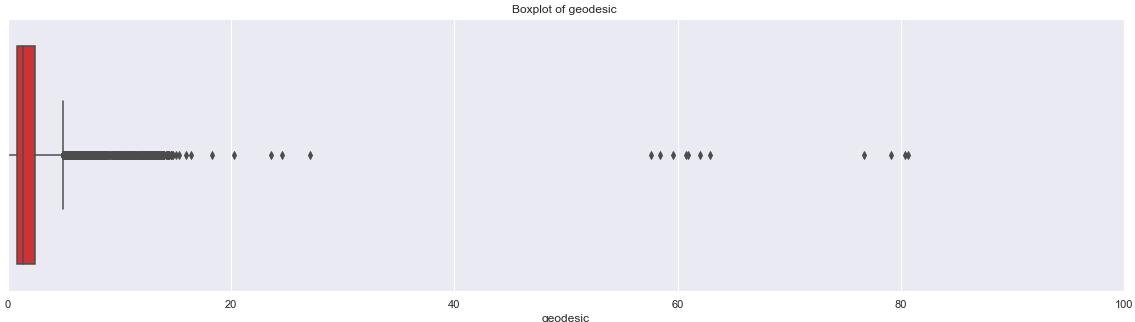
dtype='object')

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



We see that there are outliers in ‘geodesic’ and also a cab cannot go upto 3400 miles

Boxplot of ‘geodesic’ for range 0 to 100 miles.



We will treat these outliers like we previously did.

**2.1.4. Feature Selection**

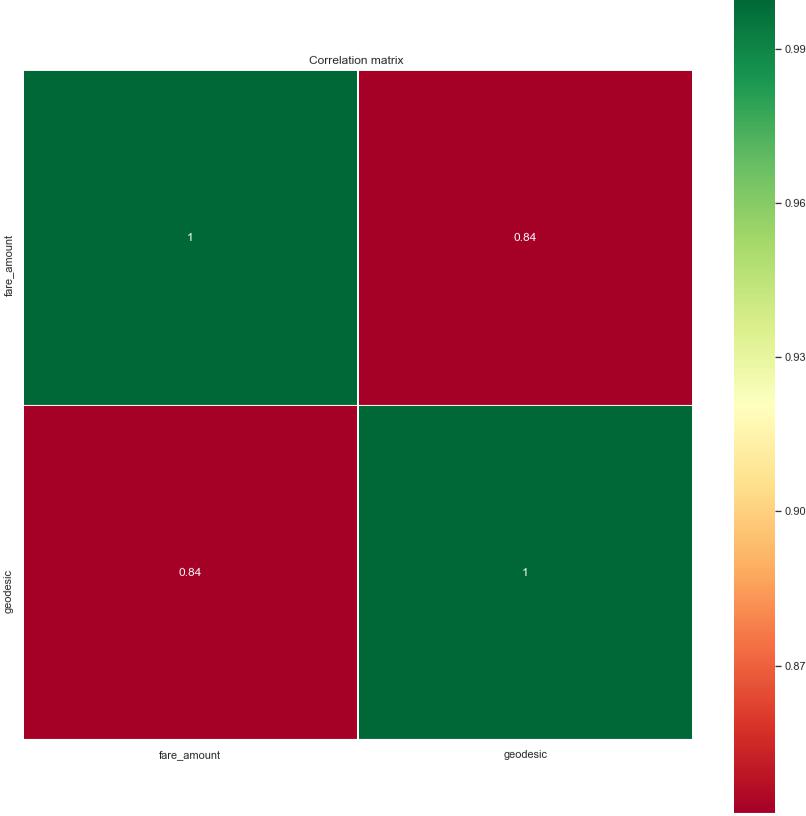
In this step we would allow only to pass relevant features to further steps. We remove irrelevant features from the dataset. We do this by some statistical techniques, like we look for features which will not be helpful in predicting the target variables. In this dataset we have to predict the fare\_amount.

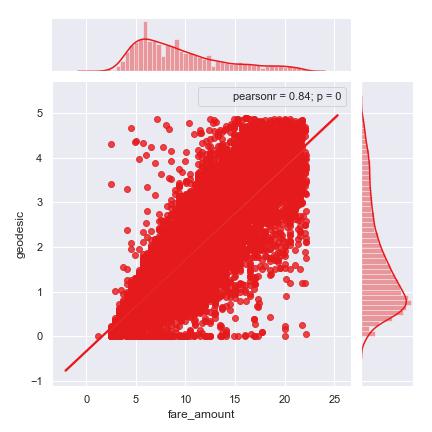
Further below are some types of test involved for feature selection:

**Correlation analysis** –This requires only numerical variables. Therefore, we will filterout only numerical variables and feed it to correlation analysis. We do this by plotting correlation plot for all numerical variables. There should be no correlation between independent variables but there should be high correlation between independent variable and dependent variable. So, we plot the correlation plot. we can see that in correlation plot faded colour like skin colour indicates that 2 variables are highly correlated with each other. As the colour fades correlation values increases.

From below correlation plot we see that:

* + 'fare\_amount' and 'geodesic' are very highly correlated with each other.
  + As fare\_amount is the target variable and ‘geodesic’ is independent variable we will keep ‘geodesic’ because it will help to explain variation in fare\_amount.

Correlation Plot:

Jointplot between ‘geodesic’ and ‘fare\_amount’:

1. **Chi-Square test of independence** –Unlike correlation analysis we will filter out onlycategorical variables and pass it to Chi-Square test. Chi-square test compares 2 categorical variables in a contingency table to see if they are related or not.
   1. Assumption for chi-square test: Dependency between Independent variable and dependent variable should be high and there should be no dependency among

independent variables.

1. Before proceeding to calculate chi-square statistic, we do the hypothesis testing: Null hypothesis: 2 variables are independent.

Alternate hypothesis: 2 variables are not independent. The interpretation of chi-square test:

* 1. For the orical or excel sheet purpose: If chi-square statistics is greater than critical value then reject the null hypothesis saying that 2 variables are dependent and if

it’s less, then accept the null hypothesis saying that 2 variables are independent.

1. While programming: If p-value is less than 0.05 then we reject the null hypothesis saying that 2 variables are dependent and if p-value is greater than 0.05 then we

accept the null hypothesis saying that 2 variables are independent.

Here we did the test between categorical independent variables pairwise.

* If p-value<0.05 then remove the variable,
* If p-value>0.05 then keep the variable.

1. **Analysis of Variance(Anova) Test** –

I. It is carried out to compare between each group in a categorical variable.

1. ANOVA only lets us know the means for different groups are same or not. It doesn’t help us identify which mean is different.

Hypothesis testing:

* + **Null Hypothesis**: mean of all categories in a variable are same.
  + **Alternate Hypothesis**: mean of at least one category in a variable is different.
* If p-value is less than 0.05 then we reject the null hypothesis.
* And if p-value is greater than 0.05 then we accept the null hypothesis.

Below is the anova analysis table for each categorical variable:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | **df** |  | **sum\_sq** |  | **mean\_sq** |  | **F** |  | **PR(>F)** |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(passenger\_count\_2)** |  | 1.0 |  | 10.881433 |  | 10.881433 |  | 0.561880 |  | 4.535152e-01 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(passenger\_count\_3)** |  | 1.0 |  | 17.098139 |  | 17.098139 |  | 0.882889 |  | 3.474262e-01 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(passenger\_count\_4)** |  | 1.0 |  | 63.987606 |  | 63.987606 |  | 3.304099 |  | 6.912635e-02 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(passenger\_count\_5)** |  | 1.0 |  | 21.227640 |  | 21.227640 |  | 1.096122 |  | 2.951349e-01 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(passenger\_count\_6)** |  | 1.0 |  | 145.904989 |  | 145.904989 |  | 7.534030 |  | 6.061341e-03 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(season\_spring)** |  | 1.0 |  | 28.961298 |  | 28.961298 |  | 1.495461 |  | 2.213894e-01 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(season\_summer)** |  | 1.0 |  | 26.878639 |  | 26.878639 |  | 1.387920 |  | 2.387746e-01 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(season\_winter)** |  | 1.0 |  | 481.664803 |  | 481.664803 |  | 24.871509 |  | 6.193822e-07 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(week\_weekend)** |  | 1.0 |  | 130.676545 |  | 130.676545 |  | 6.747686 |  | 9.395730e-03 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(session\_night\_AM)** |  | 1.0 |  | 2130.109284 |  | 2130.109284 |  | 109.991494 |  | 1.197176e-25 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(session\_night\_PM)** |  | 1.0 |  | 185.382247 |  | 185.382247 |  | 9.572500 |  | 1.978619e-03 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(session\_evening)** |  | 1.0 |  | 0.972652 |  | 0.972652 |  | 0.050224 |  | 8.226762e-01 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(session\_morning)** |  | 1.0 |  | 48.777112 |  | 48.777112 |  | 2.518682 |  | 1.125248e-01 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(year\_2010)** |  | 1.0 |  | 1507.533635 |  | 1507.533635 |  | 77.843835 |  | 1.231240e-18 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(year\_2011)** |  | 1.0 |  | 1332.003332 |  | 1332.003332 |  | 68.780056 |  | 1.189600e-16 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(year\_2012)** |  | 1.0 |  | 431.018841 |  | 431.018841 |  | 22.256326 |  | 2.406344e-06 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(year\_2013)** |  | 1.0 |  | 340.870175 |  | 340.870175 |  | 17.601360 |  | 2.738958e-05 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(year\_2014)** |  | 1.0 |  | 1496.882424 |  | 1496.882424 |  | 77.293844 |  | 1.624341e-18 |  |
|  |  |  | |  | |  | |  | |  |  |  |
|  | **C(year\_2015)** |  | 1.0 |  | 2587.637234 |  | 2587.637234 |  | 133.616659 |  | 8.839097e-31 |  |
|  |  |  | |  | |  | |  |  |  |  |  |
|  | **Residual** |  | 15640.0 |  | 302886.232626 |  | 19.366127 |  | NaN |  | NaN |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Looking at above table every variable has p value less than 0.05 so reject the null hypothesis.

1. **Multicollinearity**–In regression, "multicollinearity" refers to predictors that arecorrelated with other predictors. Multicollinearity occurs when your model includes multiple factors that are correlated not just to your response variable, but also to each other.
2. Multicollinearity increases the standard errors of the coefficients.
3. Increased standard error in turn means that coefficients for some independent variables may be found not to be significantly different from 0.
4. In other words, by overinflating the standard errors, multicollinearity makes some variables statistically insignificant when they should be significant.

Without multicollinearity (and thus, with lower standard errors), those coefficients might be significant.

IV. VIF is always greater or equal to 1.

if VIF is 1 --- Not correlated to any of the variables.

if VIF is between 1-5 --- Moderately correlated.

if VIF is above 5 --- Highly correlated.

If there are multiple variables with VIF greater than 5, only remove the variable with the highest VIF.

1. And if the VIF goes above 10, you can assume that the regression coefficients are poorly estimated due to multicollinearity.

Below is the table for VIF analysis for each independent variable:

**VIF** **features**

1. ****15.268789****Intercept
2. ****1.040670 ****passenger\_count\_2[T.1.0]
3. ****1.019507 ****passenger\_count\_3[T.1.0]
4. ****1.011836 ****passenger\_count\_4[T.1.0]
5. ****1.024990 ****passenger\_count\_5[T.1.0]
6. ****1.017206 ****passenger\_count\_6[T.1.0]
7. ****1.642247 ****season\_spring[T.1.0]
8. ****1.552411 ****season\_summer[T.1.0]
9. ****1.587588 ****season\_winter[T.1.0]
10. ****1.050786 ****week\_weekend[T.1.0]

**10**1.376197 session\_night\_AM[T.1.0]

**11** 1.423255 session\_night\_PM[T.1.0]

**12** 1.524790 session\_evening[T.1.0]

**13**1.559080 session\_morning[T.1.0]

**14**1.691361 year\_2010[T.1.0]

**15**1.687794 year\_2011[T.1.0]

**16**1.711100 year\_2012[T.1.0]

**17**1.709348 year\_2013[T.1.0]

**18** 1.665000 year\_2014[T.1.0]

**19**1.406916 year\_2015[T.1.0]

**20**1.025425 geodesic

We have checked for multicollinearity in our Dataset and all VIF values are below

5.

**2.1.5. Feature Scaling**

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

**Normalization**: Normalization refers 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.

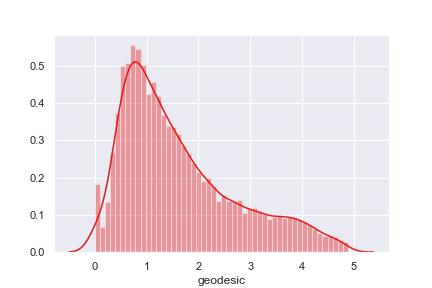
Linear Models assume that the data you are feeding are related in a linear fashion, or can be measured with a linear distance metric. Also, our independent numerical variable ‘geodesic’ is not distributed normally so we had chosen normalization over standardization.

* We have checked variance for each column in dataset before Normalisation
* High variance will affect the accuracy of the model. So, we want to normalise that variance.

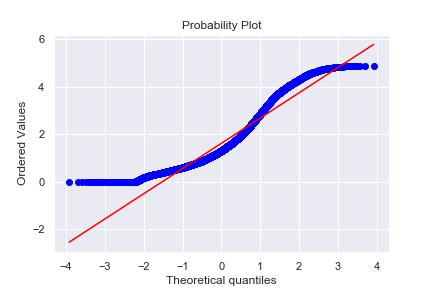
Graphs based on which standardization was chosen:

Note: It is performed only on Continuous variables.

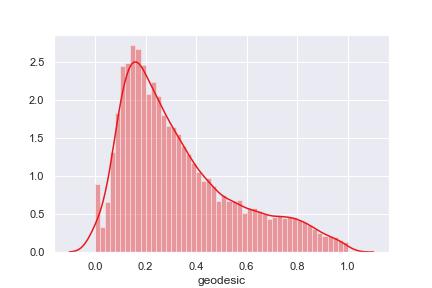
distplot() for ‘geodesic’ feature before normalization:

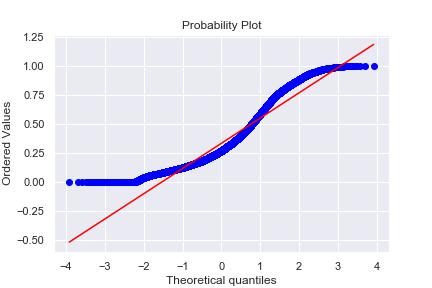


qq probability plot before normalization:



distplot() for ‘geodesic’ feature after normalization:



qq probability plot after normalization:

**2.1.6. Data after EDA and pre-processing**

Save this data after EDA and data pre-processing for further steps.

**2.2. Model Development**

Our problem statement wants us to predict the fare\_amount. This is a

Regression problem. So, we are going to build regression models on training data and predict it on test data. In this project I have built models using 5 Regression Algorithms:

I. Linear Regression

II. Ridge Regression

III. Lasso Regression

IV. Decision Tree

V. Random Forest

VI. Xgboost Regression

We will evaluate performance on validation dataset which was generated

using Sampling. We will deal with specific error metrics like –

Regression metrics for our Models:

1. r square
2. Adjusted r square
3. MAPE(Mean Absolute Percentage Error)
4. MSE(Mean square Error)
5. RMSE(Root Mean Square Error)
6. RMSLE( Root Mean Squared Log Error)

Chapter 3

Conclusion



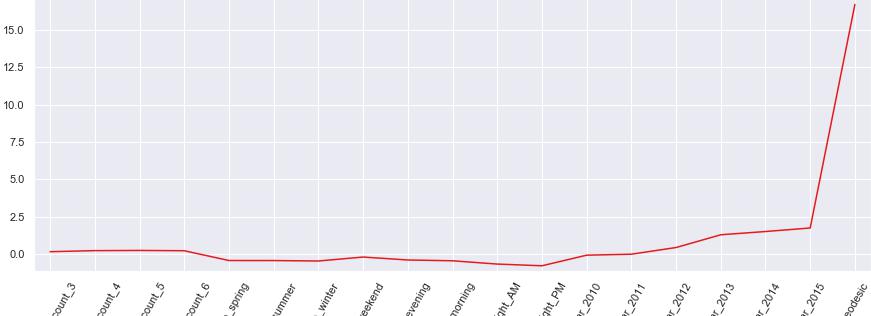
**3.1. Model Evaluation**

Here, we will evaluate the performance of different Regression models based on different Error Metrics

1. Multiple Linear Regression:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Error Metrics | r square | Adj r sq | MAPE | MSE | RMSE | RMSLE |
|  |  |  |  |  |  |  |
| Train | 0.734 | 0.733 | 18.73 | 5.28 | 2.29 | 0.21 |
| Validation | 0.719 | 0.7406 | 18.96 | 5.29 | 2.30 | 0.21 |

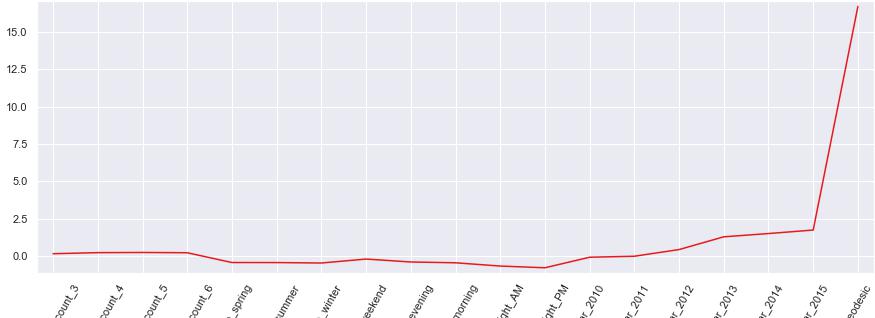
Line Plot for Coefficients of Multiple Linear regression:



1. Ridge Regression:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Error Metrics | r square | Adj r sq | MAPE | MSE | RMSE | RMSLE |
|  |  |  |  |  |  |  |
| Train | 0.7343 | 0.733 | 18.74 | 5.28 | 2.29 | 0.21 |
| validation | 0.7419 | 0.7406 | 18.96 | 5.29 | 2.3 | 0.21 |

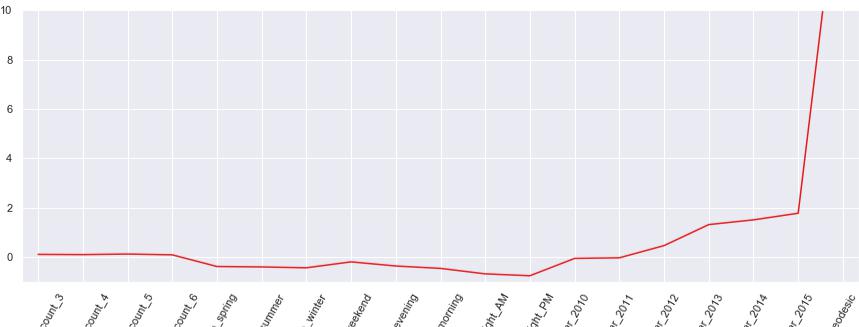
Line Plot for Coefficients of Ridge regression:



1. Lasso Regression:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Error Metrics | r square | Adj r sq | MAPE | MSE | RMSE | RMSLE |
|  |  |  |  |  |  |  |
| Train | 0.7341 | 0.7337 | 18.75 | 5.28 | 2.29 | 0.21 |
| Validation | 0.7427 | 0.7415 | 18.95 | 5.27 | 2.29 | 0.21 |

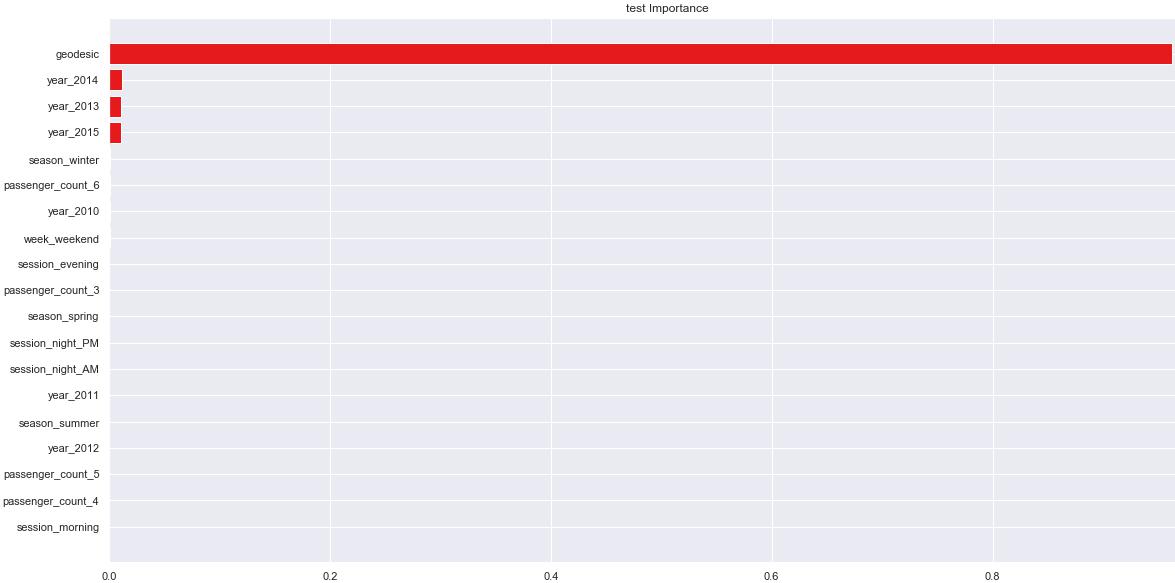
Line Plot for Coefficients of Lasso regression:



IV. Decision Tree Regression:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Error Metrics | r square | Adj r sq | MAPE | MSE | RMSE | RMSLE |
|  |  |  |  |  |  |  |
| Train | 0.7471 | 0.7467 | 18.54 | 5.02 | 2.24 | 0.20 |
| Validation | 0.7408 | 0.7396 | 19.07 | 5.31 | 2.30 | 0.21 |

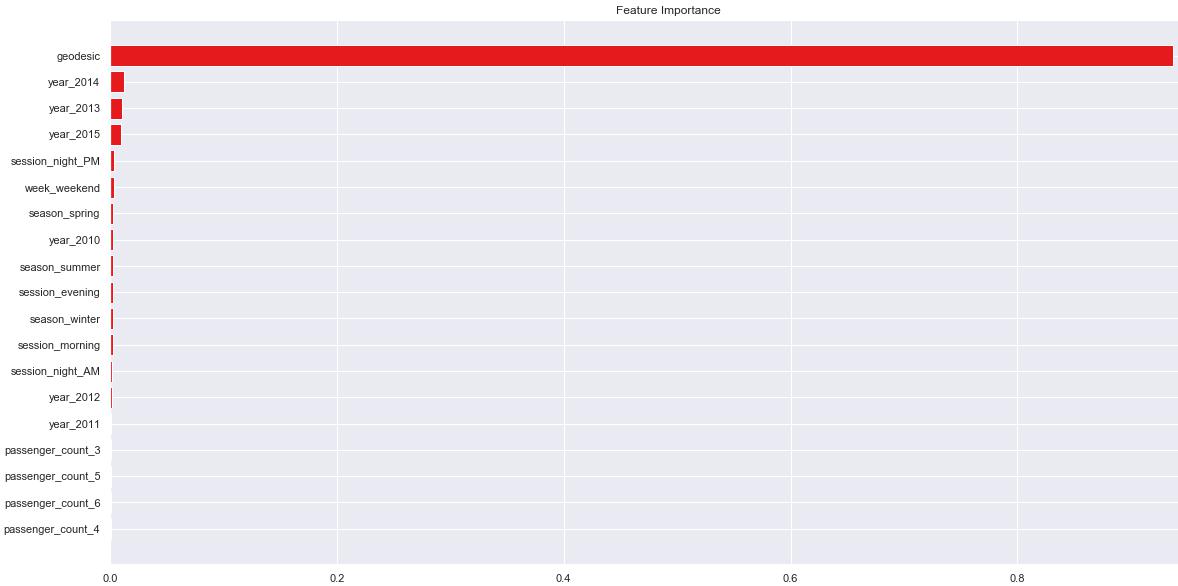
Bar Plot of Decision tree Feature Importance:



1. Random Forest Regression:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Error Metrics | r square | Adj r sq | MAPE | MSE | RMSE | RMSLE |
|  |  |  |  |  |  |  |
| Train | 0.7893 | 0.7889 | 16.95 | 4.19 | 2.04 | 0.19 |
| Validation | 0.7542 | 0.7530 | 18.56 | 5.09 | 2.24 | 0.20 |

Bar Plot of Random Forest Feature Importance:



**Improving accuracy**

* Improve Accuracy

a) Algorithm Tuning

b) Ensembles

* We have used XGboost as an ensemble technique.

Xgboost hyperparameters tuned parameters:Tuned Xgboost Parameters: {'subsample': 0.1,

'reg\_alpha': 0.08685113737513521, 'n\_estimators': 200, 'max\_depth': 3, 'learning\_rate': 0.05,

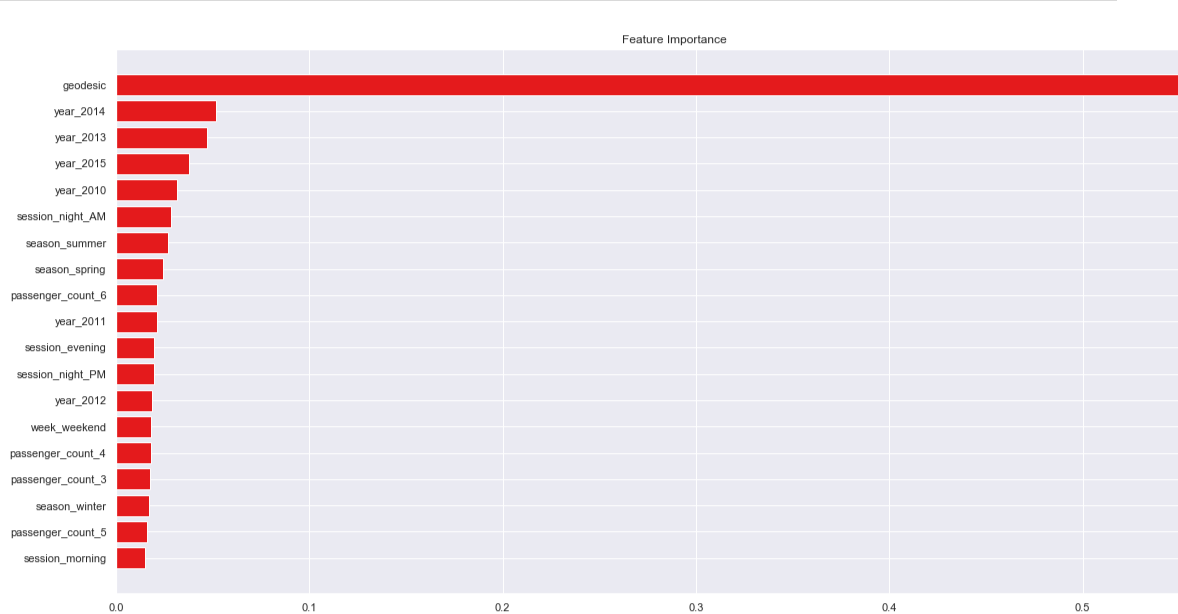
'colsample\_bytree': 0.7000000000000001, 'colsample\_bynode': 0.7000000000000001,

'colsample\_bylevel': 0.9000000000000001}

Xgboost Regression:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Error Metrics | r square | Adj r sq | MAPE | MSE | RMSE | RMSLE |
|  |  |  |  |  |  |  |
| Train | 0.7542 | 0.7538 | 18.15 | 4.88 | 2.21 | 0.20 |
| Validation | 0.7587 | 0.7575 | 18.37 | 4.96 | 2.22 | 0.20 |

Bar Plot of Xgboost Feature Importance:



Chapter 3

Conclusion



3.2. Model Selection

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

We have trained an XGboost model on entire training dataset and used that model to predict on test data. Also, we have saved model for later use.

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

r square 0.7536990357805177

Adjusted r square:0.7521952702534707

MAPE:18.477615127632426

MSE: 5.044311731381777

RMSE: 2.24595452567094

RMSLE: 0.20634512637457236

Feature importance:



Chapter 4

Codes





4.1. Python code

# ### Problem Statement  
# 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.  
  
# In[4]:  
  
  
# loading the required libraries   
import os  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import matplotlib.pyplot as plt  
import scipy.stats as stats  
from fancyimpute import KNN  
import warnings  
warnings.filterwarnings('ignore')  
from geopy.distance import geodesic  
from geopy.distance import great\_circle  
from scipy.stats import chi2\_contingency  
import statsmodels.api as sm  
from statsmodels.formula.api import ols  
from patsy import dmatrices  
from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import mean\_squared\_error  
from sklearn import metrics  
from sklearn.linear\_model import LinearRegression,Ridge,Lasso  
from sklearn.model\_selection import GridSearchCV  
from sklearn.model\_selection import RandomizedSearchCV  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.tree import DecisionTreeRegressor  
from xgboost import XGBRegressor  
import xgboost as xgb  
from sklearn.externals import joblib   
  
  
# In[5]:  
  
  
#Get working dirctory  
os.getcwd()  
  
  
# In[6]:  
  
  
# set the working directory  
os.chdir('C:\\Users\\Arun Kumar\\Desktop\\cab fare prediction')  
  
  
# The details of data attributes in the dataset are as follows:  
# - 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.  
  
# Predictive modeling machine learning project can be broken down into below workflow:   
# 1. Prepare Problem   
# a) Load libraries b) Load dataset   
# 2. Summarize Data a) Descriptive statistics b) Data visualizations   
# 3. Prepare Data a) Data Cleaning b) Feature Selection c) Data Transforms   
# 4. Evaluate Algorithms a) Split-out validation dataset b) Test options and evaluation metric c) Spot Check Algorithms d) Compare Algorithms   
# 5. Improve Accuracy a) Algorithm Tuning b) Ensembles   
# 6. Finalize Model a) Predictions on validation dataset b) Create standalone model on entire training dataset c) Save model for later use  
  
# In[7]:  
  
  
# Importing data  
train = pd.read\_csv('train\_cab.csv',dtype={'fare\_amount':np.float64},na\_values={'fare\_amount':'430-'})  
test = pd.read\_csv('test.csv')  
data=[train,test]  
for i in data:  
 i['pickup\_datetime'] = pd.to\_datetime(i['pickup\_datetime'],errors='coerce')  
train.head(5)  
  
  
# In[8]:  
  
  
train.info()  
  
  
# In[9]:  
  
  
test.head()  
  
  
# In[10]:  
  
  
test.info()  
  
  
# In[11]:  
  
  
test.describe()  
  
  
# In[12]:  
  
  
train.describe()  
  
  
# ## EDA   
  
# - we will convert passenger\_count into a categorical variable because passenger\_count is not a continuous variable.  
# - passenger\_count cannot take continous values. and also they are limited in number if its a cab.  
  
# In[13]:  
  
  
cat\_var=['passenger\_count']  
num\_var=['fare\_amount','pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']  
  
  
# ## Graphical EDA - Data Visualization   
  
# In[14]:  
  
  
# setting up the sns for plots  
sns.set(style='darkgrid',palette='Set1')  
  
  
# Some histogram plots from seaborn library  
  
# In[15]:  
  
  
plt.figure(figsize=(20,20))  
plt.subplot(321)  
\_ = sns.distplot(train['fare\_amount'],bins=50)  
plt.subplot(322)  
\_ = sns.distplot(train['pickup\_longitude'],bins=50)  
plt.subplot(323)  
\_ = sns.distplot(train['pickup\_latitude'],bins=50)  
plt.subplot(324)  
\_ = sns.distplot(train['dropoff\_longitude'],bins=50)  
plt.subplot(325)  
\_ = sns.distplot(train['dropoff\_latitude'],bins=50)  
# plt.savefig('hist.png')  
plt.show()  
  
  
# Some Bee Swarmplots  
  
# In[16]:  
  
  
# plt.figure(figsize=(25,25))  
# \_ = sns.swarmplot(x='passenger\_count',y='fare\_amount',data=train)  
# plt.title('Cab Fare w.r.t passenger\_count')  
  
  
# - Jointplots for Bivariate Analysis.  
# - Here Scatter plot has regression line between 2 variables along with separate Bar plots of both variables.  
# - Also its annotated with pearson correlation coefficient and p value.  
  
# In[17]:  
  
  
\_ = sns.jointplot(x='fare\_amount',y='pickup\_longitude',data=train,kind = 'reg')  
\_.annotate(stats.pearsonr)  
# plt.savefig('jointfplo.png')  
plt.show()  
  
  
# In[18]:  
  
  
\_ = sns.jointplot(x='fare\_amount',y='pickup\_latitude',data=train,kind = 'reg')  
\_.annotate(stats.pearsonr)  
# plt.savefig('jointfpla.png')  
plt.show()  
  
  
# In[19]:  
  
  
\_ = sns.jointplot(x='fare\_amount',y='dropoff\_longitude',data=train,kind = 'reg')  
\_.annotate(stats.pearsonr)  
# plt.savefig('jointfdlo.png')  
plt.show()  
  
  
# In[20]:  
  
  
\_ = sns.jointplot(x='fare\_amount',y='dropoff\_latitude',data=train,kind = 'reg')  
\_.annotate(stats.pearsonr)  
# plt.savefig('jointfdla.png')  
plt.show()  
  
  
# Some Violinplots to see spread of variables  
  
# In[21]:  
  
  
plt.figure(figsize=(20,20))  
plt.subplot(321)  
\_ = sns.violinplot(y='fare\_amount',data=train)  
plt.subplot(322)  
\_ = sns.violinplot(y='pickup\_longitude',data=train)  
plt.subplot(323)  
\_ = sns.violinplot(y='pickup\_latitude',data=train)  
plt.subplot(324)  
\_ = sns.violinplot(y='dropoff\_longitude',data=train)  
plt.subplot(325)  
\_ = sns.violinplot(y='dropoff\_latitude',data=train)  
plt.savefig('violin.png')  
plt.show()  
  
  
# Pairplot for all numerical variables  
  
# In[22]:  
  
  
\_ =sns.pairplot(data=train[num\_var],kind='scatter',dropna=True)  
\_.fig.suptitle('Pairwise plot of all numerical variables')  
# plt.savefig('Pairwise.png')  
plt.show()  
  
  
# ## Removing values which are not within desired range(outlier) depending upon basic understanding of dataset.  
  
# 1.Fare amount has a negative value, which doesn't make sense. A price amount cannot be -ve and also cannot be 0. So we will remove these fields.  
  
# In[23]:  
  
  
sum(train['fare\_amount']<1)  
  
  
# In[24]:  
  
  
train[train['fare\_amount']<1]  
  
  
# In[25]:  
  
  
train = train.drop(train[train['fare\_amount']<1].index, axis=0)  
  
  
# In[26]:  
  
  
# train.loc[train['fare\_amount'] < 1,'fare\_amount'] = np.nan  
  
  
# 2.Passenger\_count variable  
  
# In[27]:  
  
  
for i in range(4,11):  
 print('passenger\_count above' +str(i)+'={}'.format(sum(train['passenger\_count']>i)))  
  
  
# so 20 observations of passenger\_count is consistenly above from 6,7,8,9,10 passenger\_counts, let's check them.  
  
# In[28]:  
  
  
train[train['passenger\_count']>6]  
  
  
# Also we need to see if there are any passenger\_count<1  
  
# In[29]:  
  
  
train[train['passenger\_count']<1]  
  
  
# In[30]:  
  
  
len(train[train['passenger\_count']<1])  
  
  
# In[31]:  
  
  
test['passenger\_count'].unique()  
  
  
# - passenger\_count variable conatins values which are equal to 0.  
# - 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 remove those 0 values.  
# - Also, We will remove 20 observation which are above 6 value because a cab cannot hold these number of passengers.  
  
# In[32]:  
  
  
train = train.drop(train[train['passenger\_count']>6].index, axis=0)  
train = train.drop(train[train['passenger\_count']<1].index, axis=0)  
  
  
# In[33]:  
  
  
# train.loc[train['passenger\_count'] >6,'passenger\_count'] = np.nan  
# train.loc[train['passenger\_count'] >1,'passenger\_count'] = np.nan  
  
  
# In[34]:  
  
  
sum(train['passenger\_count']>6)  
  
  
# 3.Latitudes range from -90 to 90.Longitudes range from -180 to 180.  
# Removing which does not satisfy these ranges  
  
# In[35]:  
  
  
print('pickup\_longitude above 180={}'.format(sum(train['pickup\_longitude']>180)))  
print('pickup\_longitude below -180={}'.format(sum(train['pickup\_longitude']<-180)))  
print('pickup\_latitude above 90={}'.format(sum(train['pickup\_latitude']>90)))  
print('pickup\_latitude below -90={}'.format(sum(train['pickup\_latitude']<-90)))  
print('dropoff\_longitude above 180={}'.format(sum(train['dropoff\_longitude']>180)))  
print('dropoff\_longitude below -180={}'.format(sum(train['dropoff\_longitude']<-180)))  
print('dropoff\_latitude below -90={}'.format(sum(train['dropoff\_latitude']<-90)))  
print('dropoff\_latitude above 90={}'.format(sum(train['dropoff\_latitude']>90)))  
  
  
# - There's only one outlier which is in variable pickup\_latitude.So we will remove it with nan.  
# - Also we will see if there are any values equal to 0.  
  
# In[36]:  
  
  
for i in ['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']:  
 print(i,'equal to 0={}'.format(sum(train[i]==0)))  
  
  
# there are values which are equal to 0. we will remove them.  
  
# In[37]:  
  
  
train = train.drop(train[train['pickup\_latitude']>90].index, axis=0)  
for i in ['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']:  
 train = train.drop(train[train[i]==0].index, axis=0)  
  
  
# In[38]:  
  
  
# for i in ['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']:  
# train.loc[train[i]==0,i] = np.nan  
# train.loc[train['pickup\_latitude']>90,'pickup\_latitude'] = np.nan  
  
  
# In[39]:  
  
  
train.shape  
  
  
# So, we lossed 16067-15661=406 observations because of non-sensical values.  
  
# In[40]:  
  
  
df=train.copy()  
# train=df.copy()  
  
  
# ## Missing Value Analysis   
  
# In[41]:  
  
  
#Create dataframe with missing percentage  
missing\_val = pd.DataFrame(train.isnull().sum())  
#Reset index  
missing\_val = missing\_val.reset\_index()  
missing\_val  
  
  
# - As we can see there are some missing values in the data.  
# - Also pickup\_datetime variable has 1 missing value.   
# - We will impute missing values for fare\_amount,passenger\_count variables except pickup\_datetime.  
# - And we will drop that 1 row which has missing value in pickup\_datetime.  
  
# In[43]:  
  
  
#Rename variable  
missing\_val = missing\_val.rename(columns = {'index': 'Variables', 0: 'Missing\_percentage'})  
missing\_val  
#Calculate percentage  
missing\_val['Missing\_percentage'] = (missing\_val['Missing\_percentage']/len(train))\*100  
#descending order  
missing\_val = missing\_val.sort\_values('Missing\_percentage', ascending = False).reset\_index(drop = True)  
missing\_val  
  
  
# 1.For Passenger\_count:  
# - Actual value = 1  
# - Mode = 1  
# - KNN = 2  
  
# In[44]:  
  
  
# Choosing a random values to replace it as NA  
train['passenger\_count'].loc[1000]  
  
  
# In[45]:  
  
  
# Replacing 1.0 with NA  
train['passenger\_count'].loc[1000] = np.nan  
train['passenger\_count'].loc[1000]  
  
  
# In[46]:  
  
  
# Impute with mode  
train['passenger\_count'].fillna(train['passenger\_count'].mode()[0]).loc[1000]  
  
  
# We can't use mode method because data will be more biased towards passenger\_count=1  
  
# 2.For fare\_amount:   
# - Actual value = 7.0,  
# - Mean = 15.117,  
# - Median = 8.5,  
# - KNN = 7.369801  
  
# In[47]:  
  
  
# for i in ['fare\_amount','pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']:  
# # Choosing a random values to replace it as NA  
# a=train[i].loc[1000]  
# print(i,'at loc-1000:{}'.format(a))  
# # Replacing 1.0 with NA  
# train[i].loc[1000] = np.nan  
# print('Value after replacing with nan:{}'.format(train[i].loc[1000]))  
# # Impute with mean  
# print('Value if imputed with mean:{}'.format(train[i].fillna(train[i].mean()).loc[1000]))  
# # Impute with median  
# print('Value if imputed with median:{}\n'.format(train[i].fillna(train[i].median()).loc[1000]))  
  
  
# In[48]:  
  
  
# Choosing a random values to replace it as NA  
a=train['fare\_amount'].loc[1000]  
print('fare\_amount at loc-1000:{}'.format(a))  
# Replacing 1.0 with NA  
train['fare\_amount'].loc[1000] = np.nan  
print('Value after replacing with nan:{}'.format(train['fare\_amount'].loc[1000]))  
# Impute with mean  
print('Value if imputed with mean:{}'.format(train['fare\_amount'].fillna(train['fare\_amount'].mean()).loc[1000]))  
# Impute with median  
print('Value if imputed with median:{}'.format(train['fare\_amount'].fillna(train['fare\_amount'].median()).loc[1000]))  
  
  
# In[49]:  
  
  
train.std()  
  
  
# In[50]:  
  
  
columns=['fare\_amount', 'pickup\_longitude', 'pickup\_latitude','dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count']  
  
  
# we will separate pickup\_datetime into a different dataframe and then merge with train in feature engineering step.  
  
# In[51]:  
  
  
pickup\_datetime=pd.DataFrame(train['pickup\_datetime'])  
  
  
# In[52]:  
  
  
# Imputing with missing values using KNN  
# Use 19 nearest rows which have a feature to fill in each row's missing features  
train = pd.DataFrame(KNN(k = 19).fit\_transform(train.drop('pickup\_datetime',axis=1)),columns=columns, index=train.index)  
  
  
# In[53]:  
  
  
train.std()  
  
  
# In[54]:  
  
  
train.loc[1000]  
  
  
# In[55]:  
  
  
train['passenger\_count'].head()  
  
  
# In[56]:  
  
  
train['passenger\_count']=train['passenger\_count'].astype('int')  
  
  
# In[57]:  
  
  
train.std()  
  
  
# In[58]:  
  
  
train['passenger\_count'].unique()  
  
  
# In[59]:  
  
  
train['passenger\_count']=pd.Categorical(train['passenger\_count'], categories=[1,2,3,4,5,6],ordered=True)  
  
  
# In[60]:  
  
  
train['passenger\_count'].unique()  
  
  
# In[61]:  
  
  
train.loc[1000]  
  
  
# - Now about missing value in pickup\_datetime  
  
# In[62]:  
  
  
pickup\_datetime.head()  
  
  
# In[63]:  
  
  
#Create dataframe with missing percentage  
missing\_val = pd.DataFrame(pickup\_datetime.isnull().sum())  
#Reset index  
missing\_val = missing\_val.reset\_index()  
missing\_val  
  
  
# In[64]:  
  
  
pickup\_datetime.shape  
  
  
# In[65]:  
  
  
train.shape  
  
  
# - We will drop 1 row which has missing value for pickup\_datetime variable after feature engineering step because if we drop now, pickup\_datetime dataframe will have 16040 rows and our train has 1641 rows, then if we merge these 2 dataframes then pickup\_datetime variable will gain 1 missing value.  
# - And if we merge and then drop now then we would require to split again before outlier analysis and then merge again in feature engineering step.  
# - So, instead of doing the work 2 times we will drop 1 time i.e. after feature engineering process.  
  
# In[66]:  
  
  
df1 = train.copy()  
train=df1.copy()  
  
  
# In[67]:  
  
  
train['passenger\_count'].describe()  
  
  
# In[68]:  
  
  
train.describe()  
  
  
# ## Outlier Analysis using Boxplot  
# - We Will do Outlier Analysis only on Fare\_amount just for now and we will do outlier analysis after feature engineering laitudes and longitudes.  
  
# - Univariate Boxplots: Boxplots for all Numerical Variables including target variable.  
  
# In[69]:  
  
  
plt.figure(figsize=(20,5))   
plt.xlim(0,100)  
sns.boxplot(x=train['fare\_amount'],data=train,orient='h')  
plt.title('Boxplot of fare\_amount')  
# plt.savefig('bp of fare\_amount.png')  
plt.show()  
  
  
# In[70]:  
  
  
sum(train['fare\_amount']<22.5)/len(train['fare\_amount'])\*100  
  
  
# - Bivariate Boxplots: Boxplot for Numerical Variable Vs Categorical Variable.  
  
# In[71]:  
  
  
plt.figure(figsize=(20,10))  
plt.xlim(0,100)  
\_ = sns.boxplot(x=train['fare\_amount'],y=train['passenger\_count'],data=train,orient='h')  
plt.title('Boxplot of fare\_amount w.r.t passenger\_count')  
# plt.savefig('Boxplot of fare\_amount w.r.t passenger\_count.png')  
plt.show()  
  
  
# In[72]:  
  
  
train.describe()  
  
  
# In[73]:  
  
  
train['passenger\_count'].describe()  
  
  
# ## Outlier Treatment  
# - As we can see from the above Boxplots there are outliers in the train dataset.  
# - Reconsider pickup\_longitude,etc.  
  
# In[74]:  
  
  
def outlier\_treatment(col):  
 *''' calculating outlier indices and replacing them with NA '''* #Extract quartiles  
 q75, q25 = np.percentile(train[col], [75 ,25])  
 print(q75,q25)  
 #Calculate IQR  
 iqr = q75 - q25  
 #Calculate inner and outer fence  
 minimum = q25 - (iqr\*1.5)  
 maximum = q75 + (iqr\*1.5)  
 print(minimum,maximum)  
 #Replace with NA  
 train.loc[train[col] < minimum,col] = np.nan  
 train.loc[train[col] > maximum,col] = np.nan  
  
  
# In[75]:  
  
  
# for i in num\_var:  
outlier\_treatment('fare\_amount')  
# outlier\_treatment('pickup\_longitude')  
# outlier\_treatment('pickup\_latitude')  
# outlier\_treatment('dropoff\_longitude')  
# outlier\_treatment('dropoff\_latitude')  
  
  
# In[76]:  
  
  
pd.DataFrame(train.isnull().sum())  
  
  
# In[77]:  
  
  
train.std()  
  
  
# In[78]:  
  
  
#Imputing with missing values using KNN  
train = pd.DataFrame(KNN(k = 3).fit\_transform(train), columns = train.columns, index=train.index)  
  
  
# In[79]:  
  
  
train.std()  
  
  
# In[80]:  
  
  
train['passenger\_count'].describe()  
  
  
# In[81]:  
  
  
train['passenger\_count']=train['passenger\_count'].astype('int').round().astype('object').astype('category')  
  
  
# In[82]:  
  
  
train.describe()  
  
  
# In[83]:  
  
  
train.head()  
  
  
# In[84]:  
  
  
df2 = train.copy()  
# train=df2.copy()  
  
  
# In[85]:  
  
  
train.shape  
  
  
# ## Feature Engineering  
  
# #### 1.Feature Engineering for timestamp variable  
# - we will derive new features from pickup\_datetime variable  
# - new features will be year,month,day\_of\_week,hour  
  
# In[86]:  
  
  
# we will Join 2 Dataframes pickup\_datetime and train  
train = pd.merge(pickup\_datetime,train,right\_index=True,left\_index=True)  
train.head()  
  
  
# In[87]:  
  
  
train.shape  
  
  
# In[88]:  
  
  
train=train.reset\_index(drop=True)  
  
  
# As we discussed in Missing value imputation step about dropping missing value, we will do it now.  
  
# In[89]:  
  
  
pd.DataFrame(train.isna().sum())  
  
  
# In[90]:  
  
  
train=train.dropna()  
  
  
# In[91]:  
  
  
data = [train,test]  
for i in data:  
 i["year"] = i["pickup\_datetime"].apply(lambda row: row.year)  
 i["month"] = i["pickup\_datetime"].apply(lambda row: row.month)  
# i["day\_of\_month"] = i["pickup\_datetime"].apply(lambda row: row.day)  
 i["day\_of\_week"] = i["pickup\_datetime"].apply(lambda row: row.dayofweek)  
 i["hour"] = i["pickup\_datetime"].apply(lambda row: row.hour)  
  
  
# In[92]:  
  
  
# train\_nodummies=train.copy()  
# train=train\_nodummies.copy()  
  
  
# In[93]:  
  
  
plt.figure(figsize=(20,10))  
sns.countplot(train['year'])  
# plt.savefig('year.png')  
  
plt.figure(figsize=(20,10))  
sns.countplot(train['month'])  
# plt.savefig('month.png')  
  
plt.figure(figsize=(20,10))  
sns.countplot(train['day\_of\_week'])  
# plt.savefig('day\_of\_week.png')  
  
plt.figure(figsize=(20,10))  
sns.countplot(train['hour'])  
# plt.savefig('hour.png')  
  
  
# Now we will use month,day\_of\_week,hour to derive new features like sessions in a day,seasons in a year,week:weekend/weekday  
  
# In[94]:  
  
  
def f(x):  
 *''' for sessions in a day using hour column '''* if (x >=5) and (x <= 11):  
 return 'morning'  
 elif (x >=12) and (x <=16 ):  
 return 'afternoon'  
 elif (x >= 17) and (x <= 20):  
 return'evening'  
 elif (x >=21) and (x <= 23) :  
 return 'night\_PM'  
 elif (x >=0) and (x <=4):  
 return'night\_AM'  
  
  
# In[95]:  
  
  
def g(x):  
 *''' for seasons in a year using month column'''* if (x >=3) and (x <= 5):  
 return 'spring'  
 elif (x >=6) and (x <=8 ):  
 return 'summer'  
 elif (x >= 9) and (x <= 11):  
 return'fall'  
 elif (x >=12)|(x <= 2) :  
 return 'winter'  
  
  
# In[96]:  
  
  
def h(x):  
 *''' for week:weekday/weekend in a day\_of\_week column '''* if (x >=0) and (x <= 4):  
 return 'weekday'  
 elif (x >=5) and (x <=6 ):  
 return 'weekend'  
  
  
# In[97]:  
  
  
train['session'] = train['hour'].apply(f)  
test['session'] = test['hour'].apply(f)  
# train\_nodummies['session'] = train\_nodummies['hour'].apply(f)  
  
  
# In[98]:  
  
  
train['seasons'] = train['month'].apply(g)  
test['seasons'] = test['month'].apply(g)  
# train['seasons'] = test['month'].apply(g)  
  
  
# In[99]:  
  
  
train['week'] = train['day\_of\_week'].apply(h)  
test['week'] = test['day\_of\_week'].apply(h)  
  
  
# In[100]:  
  
  
train.shape  
  
  
# In[101]:  
  
  
test.shape  
  
  
# #### 2.Feature Engineering for passenger\_count variable  
# - Because models in scikit learn require numerical input,if dataset contains categorical variables then we have to encode them.  
# - We will use one hot encoding technique for passenger\_count variable.  
  
# In[102]:  
  
  
train['passenger\_count'].describe()  
  
  
# In[103]:  
  
  
#Creating dummies for each variable in passenger\_count and merging dummies dataframe to both train and test dataframe  
temp = pd.get\_dummies(train['passenger\_count'], prefix = 'passenger\_count')  
train = train.join(temp)  
temp = pd.get\_dummies(test['passenger\_count'], prefix = 'passenger\_count')  
test = test.join(temp)  
temp = pd.get\_dummies(train['seasons'], prefix = 'season')  
train = train.join(temp)  
temp = pd.get\_dummies(test['seasons'], prefix = 'season')  
test = test.join(temp)  
temp = pd.get\_dummies(train['week'], prefix = 'week')  
train = train.join(temp)  
temp = pd.get\_dummies(test['week'], prefix = 'week')  
test = test.join(temp)  
temp = pd.get\_dummies(train['session'], prefix = 'session')  
train = train.join(temp)  
temp = pd.get\_dummies(test['session'], prefix = 'session')  
test = test.join(temp)  
temp = pd.get\_dummies(train['year'], prefix = 'year')  
train = train.join(temp)  
temp = pd.get\_dummies(test['year'], prefix = 'year')  
test = test.join(temp)  
  
  
# In[104]:  
  
  
train.head()  
  
  
# In[105]:  
  
  
test.head()  
  
  
# we will drop one column from each one-hot-encoded variables  
  
# In[106]:  
  
  
train.columns  
  
  
# In[107]:  
  
  
train=train.drop(['passenger\_count\_1','season\_fall','week\_weekday','session\_afternoon','year\_2009'],axis=1)  
test=test.drop(['passenger\_count\_1','season\_fall','week\_weekday','session\_afternoon','year\_2009'],axis=1)  
  
  
# #### 3.Feature Engineering for latitude and longitude variable  
# - As we have latitude and longitude data for pickup and dropoff, we will find the distance the cab travelled from pickup and dropoff location.  
  
# In[108]:  
  
  
# train.sort\_values('pickup\_datetime')  
  
  
# In[109]:  
  
  
# def haversine(coord1, coord2):  
# '''Calculate distance the cab travelled from pickup and dropoff location using the Haversine Formula'''  
# data = [train, test]  
# for i in data:  
# lon1, lat1 = coord1  
# lon2, lat2 = coord2  
# R = 6371000 # radius of Earth in meters  
# phi\_1 = np.radians(i[lat1])  
# phi\_2 = np.radians(i[lat2])  
# delta\_phi = np.radians(i[lat2] - i[lat1])  
# delta\_lambda = np.radians(i[lon2] - i[lon1])  
# a = np.sin(delta\_phi / 2.0) \*\* 2 + np.cos(phi\_1) \* np.cos(phi\_2) \* np.sin(delta\_lambda / 2.0) \*\* 2  
# c = 2 \* np.arctan2(np.sqrt(a), np.sqrt(1 - a))  
# meters = R \* c # output distance in meters  
# km = meters / 1000.0 # output distance in kilometers  
# miles = round(km, 3)/1.609344  
# i['distance'] = miles  
# # print(f"Distance: {miles} miles")  
# # return miles  
  
  
# In[110]:  
  
  
# haversine(['pickup\_longitude','pickup\_latitude'],['dropoff\_longitude','dropoff\_latitude'])  
  
  
# In[111]:  
  
  
# Calculate distance the cab travelled from pickup and dropoff location using great\_circle from geopy library  
data = [train, 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)  
  
  
# In[112]:  
  
  
train.head()  
  
  
# In[113]:  
  
  
test.head()  
  
  
# As Vincenty is more accurate than haversine. Also vincenty is prefered for short distances.Therefore we will drop great\_circle. we will drop them together with other variables which were used to feature engineer.  
  
# In[114]:  
  
  
pd.DataFrame(train.isna().sum())  
  
  
# In[115]:  
  
  
pd.DataFrame(test.isna().sum())  
  
  
# #### We will remove the variables which were used to feature engineer new variables  
  
# In[116]:  
  
  
# train\_nodummies=train\_nodummies.drop(['pickup\_datetime','pickup\_longitude', 'pickup\_latitude',  
# 'dropoff\_longitude', 'dropoff\_latitude','great\_circle'],axis = 1)  
# test\_nodummies=test.drop(['pickup\_datetime','pickup\_longitude', 'pickup\_latitude',  
# 'dropoff\_longitude', 'dropoff\_latitude','passenger\_count\_1', 'passenger\_count\_2', 'passenger\_count\_3',  
# 'passenger\_count\_4', 'passenger\_count\_5', 'passenger\_count\_6',  
# 'season\_fall', 'season\_spring', 'season\_summer', 'season\_winter',  
# 'week\_weekday', 'week\_weekend', 'session\_afternoon', 'session\_evening',  
# 'session\_morning', 'session\_night (AM)', 'session\_night (PM)',  
# 'year\_2009', 'year\_2010', 'year\_2011', 'year\_2012', 'year\_2013',  
# 'year\_2014', 'year\_2015', 'great\_circle'],axis = 1)  
  
  
# In[117]:  
  
  
train=train.drop(['pickup\_datetime','pickup\_longitude', 'pickup\_latitude',  
 'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count', 'year',  
 'month', 'day\_of\_week', 'hour', 'session', 'seasons', 'week','great\_circle'],axis=1)  
test=test.drop(['pickup\_datetime','pickup\_longitude', 'pickup\_latitude',  
 'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count', 'year',  
 'month', 'day\_of\_week', 'hour', 'session', 'seasons', 'week','great\_circle'],axis=1)  
  
  
# In[118]:  
  
  
train.shape,test.shape  
  
  
# In[119]:  
  
  
# test\_nodummies.columns  
  
  
# In[120]:  
  
  
# train\_nodummies.columns  
  
  
# In[121]:  
  
  
train.columns  
  
  
# In[122]:  
  
  
test.columns  
  
  
# In[123]:  
  
  
train.head()  
  
  
# In[124]:  
  
  
test.head()  
  
  
# In[125]:  
  
  
plt.figure(figsize=(20,5))   
sns.boxplot(x=train['geodesic'],data=train,orient='h')  
plt.title('Boxplot of geodesic ')  
# plt.savefig('bp geodesic.png')  
plt.show()  
  
  
# In[126]:  
  
  
plt.figure(figsize=(20,5))   
plt.xlim(0,100)  
sns.boxplot(x=train['geodesic'],data=train,orient='h')  
plt.title('Boxplot of geodesic ')  
# plt.savefig('bp geodesic.png')  
plt.show()  
  
  
# In[127]:  
  
  
outlier\_treatment('geodesic')  
  
  
# In[128]:  
  
  
pd.DataFrame(train.isnull().sum())  
  
  
# In[129]:  
  
  
#Imputing with missing values using KNN  
train = pd.DataFrame(KNN(k = 3).fit\_transform(train), columns = train.columns, index=train.index)  
  
  
# ## Feature Selection  
# 1.Correlation Analysis  
#   
# Statistically correlated: features move together directionally.  
# Linear models assume feature independence.  
# And if features are correlated that could introduce bias into our models.  
  
# In[130]:  
  
  
cat\_var=['passenger\_count\_2',  
 'passenger\_count\_3', 'passenger\_count\_4', 'passenger\_count\_5',  
 'passenger\_count\_6', 'season\_spring', 'season\_summer',  
 'season\_winter', 'week\_weekend',  
 'session\_evening', 'session\_morning', 'session\_night\_AM',  
 'session\_night\_PM', 'year\_2010', 'year\_2011',  
 'year\_2012', 'year\_2013', 'year\_2014', 'year\_2015']  
num\_var=['fare\_amount','geodesic']  
train[cat\_var]=train[cat\_var].apply(lambda x: x.astype('category') )  
test[cat\_var]=test[cat\_var].apply(lambda x: x.astype('category') )   
  
  
# - We will plot a Heatmap of correlation whereas, correlation measures how strongly 2 quantities are related to each other.  
  
# In[131]:  
  
  
# heatmap using correlation matrix  
plt.figure(figsize=(15,15))  
\_ = sns.heatmap(train[num\_var].corr(), square=True, cmap='RdYlGn',linewidths=0.5,linecolor='w',annot=True)  
plt.title('Correlation matrix ')  
# plt.savefig('correlation.png')  
plt.show()  
  
  
# As we can see from above correlation plot fare\_amount and geodesic is correlated to each other.  
  
# - Jointplots for Bivariate Analysis.  
# - Here Scatter plot has regression line between 2 variables along with separate Bar plots of both variables.  
# - Also its annotated with pearson correlation coefficient and p value.  
  
# In[132]:  
  
  
\_ = sns.jointplot(x='fare\_amount',y='geodesic',data=train,kind = 'reg')  
\_.annotate(stats.pearsonr)  
# plt.savefig('jointct.png')  
plt.show()  
  
  
# ### Chi-square test of Independence for Categorical Variables/Features  
  
# - Hypothesis testing :  
# - Null Hypothesis: 2 variables are independent.  
# - Alternate Hypothesis: 2 variables are not independent.  
# - If p-value is less than 0.05 then we reject the null hypothesis saying that 2 variables are dependent.  
# - And if p-value is greater than 0.05 then we accept the null hypothesis saying that 2 variables are independent.   
# - There should be no dependencies between Independent variables.  
# - So we will remove that variable whose p-value with other variable is low than 0.05.  
# - And we will keep that variable whose p-value with other variable is high than 0.05  
  
# In[133]:  
  
  
#loop for chi square values  
for i in cat\_var:  
 for j in cat\_var:  
 if(i != j):  
 chi2, p, dof, ex = chi2\_contingency(pd.crosstab(train[i], train[j]))  
 if(p < 0.05):  
 print(i,"and",j,"are dependent on each other with",p,'----Remove')  
 else:  
 print(i,"and",j,"are independent on each other with",p,'----Keep')  
  
  
# ## Analysis of Variance(Anova) Test  
# - It is carried out to compare between each groups in a categorical variable.  
# - ANOVA only lets us know the means for different groups are same or not. It doesn’t help us identify which mean is different.  
# - Hypothesis testing :  
# - Null Hypothesis: mean of all categories in a variable are same.  
# - Alternate Hypothesis: mean of at least one category in a variable is different.  
# - If p-value is less than 0.05 then we reject the null hypothesis.  
# - And if p-value is greater than 0.05 then we accept the null hypothesis.  
  
# In[134]:  
  
  
train.columns  
  
  
# In[135]:  
  
  
#ANOVA \_1)+C(passenger\_count\_2)+C(passenger\_count\_3)+C(passenger\_count\_4)+C(passenger\_count\_5)+C(passenger\_count\_6)  
model = ols('fare\_amount ~ C(passenger\_count\_2)+C(passenger\_count\_3)+C(passenger\_count\_4)+C(passenger\_count\_5)+C(passenger\_count\_6)+C(season\_spring)+C(season\_summer)+C(season\_winter)+C(week\_weekend)+C(session\_night\_AM)+C(session\_night\_PM)+C(session\_evening)+C(session\_morning)+C(year\_2010)+C(year\_2011)+C(year\_2012)+C(year\_2013)+C(year\_2014)+C(year\_2015)',data=train).fit()  
   
aov\_table = sm.stats.anova\_lm(model)  
aov\_table  
  
  
# Every variable has p-value less than 0.05 therefore we reject the null hypothesis.  
  
# ## Multicollinearity Test  
# - VIF is always greater or equal to 1.  
# - if VIF is 1 --- Not correlated to any of the variables.  
# - if VIF is between 1-5 --- Moderately correlated.  
# - if VIF is above 5 --- Highly correlated.  
# - If there are multiple variables with VIF greater than 5, only remove the variable with the highest VIF.  
  
# In[136]:  
  
  
# \_1+passenger\_count\_2+passenger\_count\_3+passenger\_count\_4+passenger\_count\_5+passenger\_count\_6  
outcome, predictors = dmatrices('fare\_amount ~ geodesic+passenger\_count\_2+passenger\_count\_3+passenger\_count\_4+passenger\_count\_5+passenger\_count\_6+season\_spring+season\_summer+season\_winter+week\_weekend+session\_night\_AM+session\_night\_PM+session\_evening+session\_morning+year\_2010+year\_2011+year\_2012+year\_2013+year\_2014+year\_2015',train, return\_type='dataframe')  
# calculating VIF for each individual Predictors  
vif = pd.DataFrame()  
vif["VIF"] = [variance\_inflation\_factor(predictors.values, i) for i in range(predictors.shape[1])]  
vif["features"] = predictors.columns  
vif  
  
  
# So we have no or very low multicollinearity  
  
# ## Feature Scaling Check with or without normalization of standarscalar  
  
# In[137]:  
  
  
train[num\_var].var()  
  
  
# In[138]:  
  
  
sns.distplot(train['geodesic'],bins=50)  
# plt.savefig('distplot.png')  
  
  
# In[139]:  
  
  
plt.figure()  
stats.probplot(train['geodesic'], dist='norm', fit=True,plot=plt)  
# plt.savefig('qq prob plot.png')  
  
  
# In[140]:  
  
  
#Normalization  
train['geodesic'] = (train['geodesic'] - min(train['geodesic']))/(max(train['geodesic']) - min(train['geodesic']))  
test['geodesic'] = (test['geodesic'] - min(test['geodesic']))/(max(test['geodesic']) - min(test['geodesic']))  
  
  
# In[141]:  
  
  
train['geodesic'].var()  
  
  
# In[142]:  
  
  
sns.distplot(train['geodesic'],bins=50)  
plt.savefig('distplot.png')  
  
  
# In[143]:  
  
  
plt.figure()  
stats.probplot(train['geodesic'], dist='norm', fit=True,plot=plt)  
# plt.savefig('qq prob plot.png')  
  
  
# In[144]:  
  
  
train.columns  
  
  
# In[145]:  
  
  
df4=train.copy()  
train=df4.copy()  
f4=test.copy()  
test=f4.copy()  
  
  
# In[146]:  
  
  
train=train.drop(['passenger\_count\_2'],axis=1)  
test=test.drop(['passenger\_count\_2'],axis=1)  
  
  
# In[147]:  
  
  
train.columns  
  
  
# ## Splitting train into train and validation subsets  
# - X\_train y\_train--are train subset  
# - X\_test y\_test--are validation subset  
  
# In[149]:  
  
  
X = train.drop('fare\_amount',axis=1).values  
y = train['fare\_amount'].values  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state=42)  
print(train.shape, X\_train.shape, X\_test.shape,y\_train.shape,y\_test.shape)  
  
  
# In[150]:  
  
  
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('MSE:', metrics.mean\_squared\_error(y, y\_))  
 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('RMSLE:',rmsle(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)  
 print('RMSLE:',rmsle(y\_test,y\_pred))  
  
  
# ## Multiple Linear Regression  
  
# In[151]:  
  
  
# 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\_))  
  
  
# In[152]:  
  
  
# 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(test.columns)), reg\_coef)  
plt.xticks(range(len(test.columns)), test.columns.values, rotation=60)  
plt.margins(0.02)  
plt.savefig('linear coefficients')  
plt.show()  
  
  
# In[153]:  
  
  
from sklearn.model\_selection import cross\_val\_score  
# Create a linear regression object: reg  
reg = LinearRegression()  
  
# Compute 5-fold cross-validation scores: cv\_scores  
cv\_scores = cross\_val\_score(reg,X,y,cv=5,scoring='neg\_mean\_squared\_error')  
  
# Print the 5-fold cross-validation scores  
print(cv\_scores)  
  
print("Average 5-Fold CV Score: {}".format(np.mean(cv\_scores)))  
  
  
# ## Ridge Regression  
  
# In[154]:  
  
  
# Setup the parameters and distributions to sample from: param\_dist  
param\_dist = {'alpha':np.logspace(-4, 0, 50),  
 'normalize':[True,False],  
 'max\_iter':range(500,5000,500)}  
# Instantiate a Decision ridge classifier: ridge  
ridge = Ridge()  
  
# Instantiate the gridSearchCV object: ridge\_cv  
ridge\_cv = GridSearchCV(ridge, param\_dist, cv=5,scoring='r2')  
  
# Fit it to the data  
ridge\_cv.fit(X, y)  
  
# Print the tuned parameters and score  
print("Tuned Decision ridge Parameters: {}".format(ridge\_cv.best\_params\_))  
print("Best score is {}".format(ridge\_cv.best\_score\_))  
  
  
# In[155]:  
  
  
# Instantiate a ridge regressor: ridge  
ridge = Ridge(alpha=0.0005428675439323859, normalize=True,max\_iter = 500)  
  
# Fit the regressor to the data  
ridge.fit(X\_train,y\_train)  
  
# Compute and print the coefficients  
ridge\_coef = ridge.coef\_  
print(ridge\_coef)  
  
# Plot the coefficients  
plt.figure(figsize=(15,5))  
plt.plot(range(len(test.columns)), ridge\_coef)  
plt.xticks(range(len(test.columns)), test.columns.values, rotation=60)  
plt.margins(0.02)  
# plt.savefig('ridge coefficients')  
plt.show()  
test\_scores(ridge)  
  
  
# lasso can be used feature selection  
  
# ## Lasso Regression  
  
# In[156]:  
  
  
# Setup the parameters and distributions to sample from: param\_dist  
param\_dist = {'alpha':np.logspace(-4, 0, 50),  
 'normalize':[True,False],  
 'max\_iter':range(500,5000,500)}  
# Instantiate a Decision lasso classifier: lasso  
lasso = Lasso()  
  
# Instantiate the gridSearchCV object: lasso\_cv  
lasso\_cv = GridSearchCV(lasso, param\_dist, cv=5,scoring='r2')  
  
# Fit it to the data  
lasso\_cv.fit(X, y)  
  
# Print the tuned parameters and score  
print("Tuned Decision lasso Parameters: {}".format(lasso\_cv.best\_params\_))  
print("Best score is {}".format(lasso\_cv.best\_score\_))  
  
  
# In[157]:  
  
  
# Instantiate a lasso regressor: lasso  
lasso = Lasso(alpha=0.00021209508879201905, normalize=False,max\_iter = 500)  
  
# Fit the regressor to the data  
lasso.fit(X,y)  
  
# Compute and print the coefficients  
lasso\_coef = lasso.coef\_  
print(lasso\_coef)  
  
# Plot the coefficients  
plt.figure(figsize=(15,5))  
plt.ylim(-1,10)  
plt.plot(range(len(test.columns)), lasso\_coef)  
plt.xticks(range(len(test.columns)), test.columns.values, rotation=60)  
plt.margins(0.02)  
plt.savefig('lasso coefficients')  
plt.show()  
test\_scores(lasso)  
  
  
# ## Decision Tree Regression  
  
# In[159]:  
  
  
# 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\_))  
  
  
# In[160]:  
  
  
# Instantiate a tree regressor: tree  
tree = DecisionTreeRegressor(max\_depth= 6, min\_samples\_split=2)  
  
# 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 = [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)  
  
  
# ## Random Forest Regression  
  
# In[163]:  
  
  
# 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\_))  
  
  
# In[164]:  
  
  
# Instantiate a Forest regressor: Forest  
Forest = RandomForestRegressor(n\_estimators=100, min\_samples\_split= 2, min\_samples\_leaf=4, max\_features='auto', max\_depth=9, bootstrap=True)  
  
# 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 = [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)  
  
  
# In[165]:  
  
  
from sklearn.model\_selection import cross\_val\_score  
# Create a random forest regression object: Forest  
Forest = RandomForestRegressor(n\_estimators=400, min\_samples\_split= 2, min\_samples\_leaf=4, max\_features='auto', max\_depth=12, bootstrap=True)  
  
# Compute 5-fold cross-validation scores: cv\_scores  
cv\_scores = cross\_val\_score(Forest,X,y,cv=5,scoring='neg\_mean\_squared\_error')  
  
# Print the 5-fold cross-validation scores  
print(cv\_scores)  
  
print("Average 5-Fold CV Score: {}".format(np.mean(cv\_scores)))  
  
  
# ## Improving accuracy using XGBOOST  
#   
# - a) Algorithm Tuning   
# - b) Ensembles  
  
# In[166]:  
  
  
data\_dmatrix = xgb.DMatrix(data=X,label=y)  
dtrain = xgb.DMatrix(X\_train, label=y\_train)  
dtest = xgb.DMatrix(X\_test)  
  
  
# In[167]:  
  
  
dtrain,dtest,data\_dmatrix  
  
  
# In[168]:  
  
  
params = {"objective":"reg:linear",'colsample\_bytree': 0.3,'learning\_rate': 0.1,  
 'max\_depth': 5, 'alpha': 10}  
  
cv\_results = xgb.cv(dtrain=data\_dmatrix, params=params, nfold=5,  
 num\_boost\_round=50,early\_stopping\_rounds=10,metrics="rmse", as\_pandas=True, seed=123)  
cv\_results.head()  
  
  
# In[169]:  
  
  
# the final boosting round metric  
print((cv\_results["test-rmse-mean"]).tail(1))  
  
  
# In[170]:  
  
  
Xgb = XGBRegressor()  
Xgb.fit(X\_train,y\_train)  
# pred\_xgb = model\_xgb.predict(X\_test)  
test\_scores(Xgb)  
  
  
# In[171]:  
  
  
# 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\_))  
  
  
# In[172]:  
  
  
# Instantiate a xgb regressor: xgb  
Xgb = XGBRegressor(subsample= 0.1, reg\_alpha= 0.08685113737513521, n\_estimators= 200, max\_depth= 3, learning\_rate=0.05, colsample\_bytree= 0.7000000000000001, colsample\_bynode=0.7000000000000001, colsample\_bylevel=0.9000000000000001)  
  
# 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 = [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)  
  
  
# ## Finalize model  
# - Create standalone model on entire training dataset  
# - Save model for later use  
  
# In[173]:  
  
  
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\_)))  
 print('RMSLE:',rmsle(y\_test,y\_pred))  
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))   
  
  
# In[174]:  
  
  
test.columns  
  
  
# In[175]:  
  
  
train.columns  
  
  
# In[176]:  
  
  
train.shape  
  
  
# In[177]:  
  
  
test.shape  
  
  
# In[178]:  
  
  
a=pd.read\_csv('test.csv')  
  
  
# In[179]:  
  
  
test\_pickup\_datetime=a['pickup\_datetime']  
  
  
# In[180]:  
  
  
# Instantiate a xgb regressor: xgb  
Xgb = XGBRegressor(subsample= 0.1, reg\_alpha= 0.08685113737513521, n\_estimators= 200, max\_depth= 3, learning\_rate=0.05, colsample\_bytree= 0.7000000000000001, colsample\_bynode=0.7000000000000001, colsample\_bylevel=0.9000000000000001)  
  
# 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 = [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)  
  
# Predictions  
pred = Xgb.predict(test.values)  
pred\_results\_wrt\_date = pd.DataFrame({"pickup\_datetime":test\_pickup\_datetime,"fare\_amount" : pred})  
pred\_results\_wrt\_date.to\_csv("predictions\_xgboost.csv",index=False)  
  
  
# In[181]:  
  
  
pred\_results\_wrt\_date  
  
  
# In[182]:  
  
  
# Save the model as a pickle in a file   
joblib.dump(Xgb, 'cab\_fare\_xgboost\_model.pkl')   
   
# # Load the model from the file   
# Xgb\_from\_joblib = joblib.load('cab\_fare\_xgboost\_model.pkl')   
  
  
# In[ ]:

Chapter 4

Codes





4.2 R Code

# Cab Fare Prediction

#clear envernment

rm(list = ls())

#get working dir

getwd()

#set working dir

setwd("C:/Users/Arun Kumar/Desktop/cab fare prediction")

# #loading Libraries

x = c("ggplot2", "corrgram", "DMwR", "usdm", "caret", "randomForest", "e1071",

"DataCombine", "doSNOW", "inTrees", "rpart.plot", "rpart",'MASS','xgboost','stats')

#load Packages

lapply(x, require, character.only = TRUE)

rm(x)

# The details of data attributes in the dataset are as follows:

# 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.

# loading datasets

train = read.csv("train\_cab.csv", header = T, na.strings = c(" ", "", "NA"))

test = read.csv("test.csv")

test\_pickup\_datetime = test["pickup\_datetime"]

# Structure of data

str(train)

str(test)

summary(train)

summary(test)

head(train,5)

head(test,5)

############# Exploratory Data Analysis #######################

# Changing the data types of variables

train$fare\_amount = as.numeric(as.character(train$fare\_amount))

train$passenger\_count=round(train$passenger\_count)

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

# 1.Fare amount has a negative value, which doesn't make sense. A price amount cannot be -ve and also cannot be 0. So we will remove these fields.

train[which(train$fare\_amount < 1 ),]

nrow(train[which(train$fare\_amount < 1 ),])

train = train[-which(train$fare\_amount < 1 ),]

#2.Passenger\_count variable

for (i in seq(4,11,by=1)){

print(paste('passenger\_count above ' ,i,nrow(train[which(train$passenger\_count > i ),])))

}

# so 20 observations of passenger\_count is consistenly above from 6,7,8,9,10 passenger\_counts, let's check them.

train[which(train$passenger\_count > 6 ),]

# Also we need to see if there are any passenger\_count==0

train[which(train$passenger\_count <1 ),]

nrow(train[which(train$passenger\_count <1 ),])

# We will remove these 58 observations and 20 observation which are above 6 value because a cab cannot hold these number of passengers.

train = train[-which(train$passenger\_count < 1 ),]

train = train[-which(train$passenger\_count > 6),]

# 3.Latitudes range from -90 to 90.Longitudes range from -180 to 180.Removing which does not satisfy these ranges

print(paste('pickup\_longitude above 180=',nrow(train[which(train$pickup\_longitude >180 ),])))

print(paste('pickup\_longitude above -180=',nrow(train[which(train$pickup\_longitude < -180 ),])))

print(paste('pickup\_latitude above 90=',nrow(train[which(train$pickup\_latitude > 90 ),])))

print(paste('pickup\_latitude above -90=',nrow(train[which(train$pickup\_latitude < -90 ),])))

print(paste('dropoff\_longitude above 180=',nrow(train[which(train$dropoff\_longitude > 180 ),])))

print(paste('dropoff\_longitude above -180=',nrow(train[which(train$dropoff\_longitude < -180 ),])))

print(paste('dropoff\_latitude above -90=',nrow(train[which(train$dropoff\_latitude < -90 ),])))

print(paste('dropoff\_latitude above 90=',nrow(train[which(train$dropoff\_latitude > 90 ),])))

# There's only one outlier which is in variable pickup\_latitude.So we will remove it with nan.

# Also we will see if there are any values equal to 0.

nrow(train[which(train$pickup\_longitude == 0 ),])

nrow(train[which(train$pickup\_latitude == 0 ),])

nrow(train[which(train$dropoff\_longitude == 0 ),])

nrow(train[which(train$pickup\_latitude == 0 ),])

# there are values which are equal to 0. we will remove them.

train = train[-which(train$pickup\_latitude > 90),]

train = train[-which(train$pickup\_longitude == 0),]

train = train[-which(train$dropoff\_longitude == 0),]

# Make a copy

df=train

# train=df

############# Missing Value Analysis #############

missing\_val = data.frame(apply(train,2,function(x){sum(is.na(x))}))

missing\_val$Columns = row.names(missing\_val)

names(missing\_val)[1] = "Missing\_percentage"

missing\_val$Missing\_percentage = (missing\_val$Missing\_percentage/nrow(train)) \* 100

missing\_val = missing\_val[order(-missing\_val$Missing\_percentage),]

row.names(missing\_val) = NULL

missing\_val = missing\_val[,c(2,1)]

missing\_val

unique(train$passenger\_count)

unique(test$passenger\_count)

train[,'passenger\_count'] = factor(train[,'passenger\_count'], labels=(1:6))

test[,'passenger\_count'] = factor(test[,'passenger\_count'], labels=(1:6))

# 1.For Passenger\_count:

# Actual value = 1

# Mode = 1

# KNN = 1

train$passenger\_count[1000]

train$passenger\_count[1000] = NA

getmode <- function(v) {

uniqv <- unique(v)

uniqv[which.max(tabulate(match(v, uniqv)))]

}

# Mode Method

getmode(train$passenger\_count)

# We can't use mode method because data will be more biased towards passenger\_count=1

# 2.For fare\_amount:

# Actual value = 18.1,

# Mean = 15.117,

# Median = 8.5,

# KNN = 18.28

sapply(train, sd, na.rm = TRUE)

# fare\_amount pickup\_datetime pickup\_longitude

# 435.968236 4635.700531 2.659050

# pickup\_latitude dropoff\_longitude dropoff\_latitude

# 2.613305 2.710835 2.632400

# passenger\_count

# 1.266104

train$fare\_amount[1000]

train$fare\_amount[1000]= NA

# Mean Method

mean(train$fare\_amount, na.rm = T)

#Median Method

median(train$fare\_amount, na.rm = T)

# kNN Imputation

train = knnImputation(train, k = 181)

train$fare\_amount[1000]

train$passenger\_count[1000]

sapply(train, sd, na.rm = TRUE)

# fare\_amount pickup\_datetime pickup\_longitude

# 435.661952 4635.700531 2.659050

# pickup\_latitude dropoff\_longitude dropoff\_latitude

# 2.613305 2.710835 2.632400

# passenger\_count

# 1.263859

sum(is.na(train))

str(train)

summary(train)

df1=train

# train=df1

##################### Outlier Analysis ##################

# We Will do Outlier Analysis only on Fare\_amount just for now and we will do outlier analysis after feature engineering laitudes and longitudes.

# Boxplot for fare\_amount

pl1 = ggplot(train,aes(x = factor(passenger\_count),y = fare\_amount))

pl1 + geom\_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,outlier.size=1, notch=FALSE)+ylim(0,100)

# Replace all outliers with NA and impute

vals = train[,"fare\_amount"] %in% boxplot.stats(train[,"fare\_amount"])$out

train[which(vals),"fare\_amount"] = NA

#lets check the NA's

sum(is.na(train$fare\_amount))

#Imputing with KNN

train = knnImputation(train,k=3)

# lets check the missing values

sum(is.na(train$fare\_amount))

str(train)

df2=train

# train=df2

################## Feature Engineering ##########################

# 1.Feature Engineering for timestamp variable

# we will derive new features from pickup\_datetime variable

# new features will be year,month,day\_of\_week,hour

#Convert pickup\_datetime from factor to date time

train$pickup\_date = as.Date(as.character(train$pickup\_datetime))

train$pickup\_weekday = as.factor(format(train$pickup\_date,"%u"))# Monday = 1

train$pickup\_mnth = as.factor(format(train$pickup\_date,"%m"))

train$pickup\_yr = as.factor(format(train$pickup\_date,"%Y"))

pickup\_time = strptime(train$pickup\_datetime,"%Y-%m-%d %H:%M:%S")

train$pickup\_hour = as.factor(format(pickup\_time,"%H"))

#Add same features to test set

test$pickup\_date = as.Date(as.character(test$pickup\_datetime))

test$pickup\_weekday = as.factor(format(test$pickup\_date,"%u"))# Monday = 1

test$pickup\_mnth = as.factor(format(test$pickup\_date,"%m"))

test$pickup\_yr = as.factor(format(test$pickup\_date,"%Y"))

pickup\_time = strptime(test$pickup\_datetime,"%Y-%m-%d %H:%M:%S")

test$pickup\_hour = as.factor(format(pickup\_time,"%H"))

sum(is.na(train))# there was 1 'na' in pickup\_datetime which created na's in above feature engineered variables.

train = na.omit(train) # we will remove that 1 row of na's

train = subset(train,select = -c(pickup\_datetime,pickup\_date))

test = subset(test,select = -c(pickup\_datetime,pickup\_date))

# Now we will use month,weekday,hour to derive new features like sessions in a day,seasons in a year,week:weekend/weekday

# f = function(x){

# if ((x >=5)& (x <= 11)){

# return ('morning')

# }

# if ((x >=12) & (x <= 16)){

# return ('afternoon')

# }

# if ((x >=17) & (x <= 20)){

# return ('evening')

# }

# if ((x >=21) & (x <= 23)){

# return ('night (PM)')

# }

# if ((x >=0) & (x <= 4)){

# return ('night (AM)')

# }

# }

# 2.Calculate the distance travelled using longitude and latitude

deg\_to\_rad = function(deg){

(deg \* pi) / 180

}

haversine = function(long1,lat1,long2,lat2){

#long1rad = deg\_to\_rad(long1)

phi1 = deg\_to\_rad(lat1)

#long2rad = deg\_to\_rad(long2)

phi2 = deg\_to\_rad(lat2)

delphi = deg\_to\_rad(lat2 - lat1)

dellamda = deg\_to\_rad(long2 - long1)

a = sin(delphi/2) \* sin(delphi/2) + cos(phi1) \* cos(phi2) \*

sin(dellamda/2) \* sin(dellamda/2)

c = 2 \* atan2(sqrt(a),sqrt(1-a))

R = 6371e3

R \* c / 1000 #1000 is used to convert to meters

}

# Using haversine formula to calculate distance fr both train and test

train$dist = haversine(train$pickup\_longitude,train$pickup\_latitude,train$dropoff\_longitude,train$dropoff\_latitude)

test$dist = haversine(test$pickup\_longitude,test$pickup\_latitude,test$dropoff\_longitude,test$dropoff\_latitude)

# We will remove the variables which were used to feature engineer new variables

train = subset(train,select = -c(pickup\_longitude,pickup\_latitude,dropoff\_longitude,dropoff\_latitude))

test = subset(test,select = -c(pickup\_longitude,pickup\_latitude,dropoff\_longitude,dropoff\_latitude))

str(train)

summary(train)

################ Feature selection ###################

numeric\_index = sapply(train,is.numeric) #selecting only numeric

numeric\_data = train[,numeric\_index]

cnames = colnames(numeric\_data)

#Correlation analysis for numeric variables

corrgram(train[,numeric\_index],upper.panel=panel.pie, main = "Correlation Plot")

#ANOVA for categorical variables with target numeric variable

#aov\_results = aov(fare\_amount ~ passenger\_count \* pickup\_hour \* pickup\_weekday,data = train)

aov\_results = aov(fare\_amount ~ passenger\_count + pickup\_hour + pickup\_weekday + pickup\_mnth + pickup\_yr,data = train)

summary(aov\_results)

# pickup\_weekdat has p value greater than 0.05

train = subset(train,select=-pickup\_weekday)

#remove from test set

test = subset(test,select=-pickup\_weekday)

####################### Feature Scaling ###############################

#Normality check

# qqnorm(train$fare\_amount)

# histogram(train$fare\_amount)

library(car)

# dev.off()

par(mfrow=c(1,2))

qqPlot(train$fare\_amount) # qqPlot, it has a x values derived from gaussian distribution, if data is distributed normally then the sorted data points should lie very close to the solid reference line

truehist(train$fare\_amount) # truehist() scales the counts to give an estimate of the probability density.

lines(density(train$fare\_amount)) # Right skewed # lines() and density() functions to overlay a density plot on histogram

#Normalisation

print('dist')

train[,'dist'] = (train[,'dist'] - min(train[,'dist']))/

(max(train[,'dist'] - min(train[,'dist'])))

# #check multicollearity

# library(usdm)

# vif(train[,-1])

#

# vifcor(train[,-1], th = 0.9)

#################### Splitting train into train and validation subsets ###################

set.seed(1000)

tr.idx = createDataPartition(train$fare\_amount,p=0.75,list = FALSE) # 75% in trainin and 25% in Validation Datasets

train\_data = train[tr.idx,]

test\_data = train[-tr.idx,]

rmExcept(c("test","train","df",'df1','df2','df3','test\_data','train\_data','test\_pickup\_datetime'))

###################Model Selection################

#Error metric used to select model is RMSE

############# Linear regression #################

lm\_model = lm(fare\_amount ~.,data=train\_data)

summary(lm\_model)

str(train\_data)

plot(lm\_model$fitted.values,rstandard(lm\_model),main = "Residual plot",

xlab = "Predicted values of fare\_amount",

ylab = "standardized residuals")

lm\_predictions = predict(lm\_model,test\_data[,2:6])

qplot(x = test\_data[,1], y = lm\_predictions, data = test\_data, color = I("blue"), geom = "point")

regr.eval(test\_data[,1],lm\_predictions)

# mae mse rmse mape

# 3.5303114 19.3079726 4.3940838 0.4510407

############# Decision Tree #####################

Dt\_model = rpart(fare\_amount ~ ., data = train\_data, method = "anova")

summary(Dt\_model)

#Predict for new test cases

predictions\_DT = predict(Dt\_model, test\_data[,2:6])

qplot(x = test\_data[,1], y = predictions\_DT, data = test\_data, color = I("blue"), geom = "point")

regr.eval(test\_data[,1],predictions\_DT)

# mae mse rmse mape

# 1.8981592 6.7034713 2.5891063 0.2241461

############# Random forest #####################

rf\_model = randomForest(fare\_amount ~.,data=train\_data)

summary(rf\_model)

rf\_predictions = predict(rf\_model,test\_data[,2:6])

qplot(x = test\_data[,1], y = rf\_predictions, data = test\_data, color = I("blue"), geom = "point")

regr.eval(test\_data[,1],rf\_predictions)

# mae mse rmse mape

# 1.9053850 6.3682283 2.5235349 0.2335395

######## Improving Accuracy by using Ensemble technique ---- XGBOOST ############

train\_data\_matrix = as.matrix(sapply(train\_data[-1],as.numeric))

test\_data\_data\_matrix = as.matrix(sapply(test\_data[-1],as.numeric))

xgboost\_model = xgboost(data = train\_data\_matrix,label = train\_data$fare\_amount,nrounds = 15,verbose = FALSE)

summary(xgboost\_model)

xgb\_predictions = predict(xgboost\_model,test\_data\_data\_matrix)

qplot(x = test\_data[,1], y = xgb\_predictions, data = test\_data, color = I("blue"), geom = "point")

regr.eval(test\_data[,1],xgb\_predictions)

# mae mse rmse mape

# 1.6183415 5.1096465 2.2604527 0.1861947

############# Finalizing and Saving Model for later use ##############

# In this step we will train our model on whole training Dataset and save that model for later use

train\_data\_matrix2 = as.matrix(sapply(train[-1],as.numeric))

test\_data\_matrix2 = as.matrix(sapply(test,as.numeric))

xgboost\_model2 = xgboost(data = train\_data\_matrix2,label = train$fare\_amount,nrounds = 15,verbose = FALSE)

# Saving the trained model

saveRDS(xgboost\_model2, "./final\_Xgboost\_model\_using\_R.rds")

# loading the saved model

super\_model <- readRDS("./final\_Xgboost\_model\_using\_R.rds")

print(super\_model)

# Lets now predict on test dataset

xgb = predict(super\_model,test\_data\_matrix2)

xgb\_pred = data.frame(test\_pickup\_datetime,"predictions" = xgb)

# Now lets write(save) the predicted fare\_amount in disk as .csv format

write.csv(xgb\_pred,"xgb\_predictions\_R.csv",row.names = FALSE)

Chapter 5

Reference



<https://stackoverflow.com/>

<https://www.kaggle.com/>

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