Cab Fare Prediction Project

Submitted by: *Kunal Choudhary*

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.

Dataset

Attributes: -

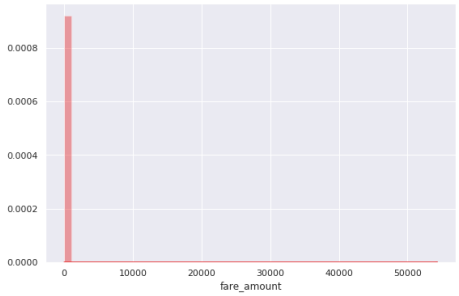
- pickup_datetime - timestamp value indicating when the cab ride started.
- pickup_longitude - float for longitude coordinate of where the cab ride started.
- pickup_latitude - float for latitude coordinate of where the cab ride started.
- dropoff_longitude - float for longitude coordinate of where the 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.

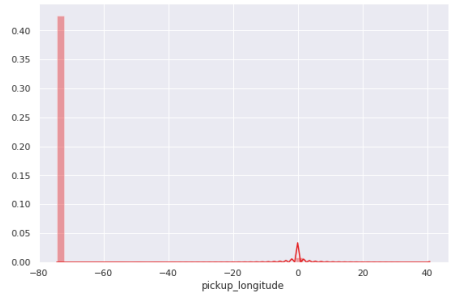
Pre-processing

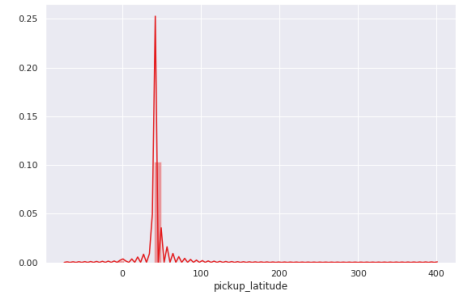
Data pre-processing is the first stage of any type of project. In this stage we get the feel of the data. We do this by looking at plots of independent variables vs target variables. If the data is messy, we try to improve it by sorting deleting extra rows and columns. This stage is called as Exploratory 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.

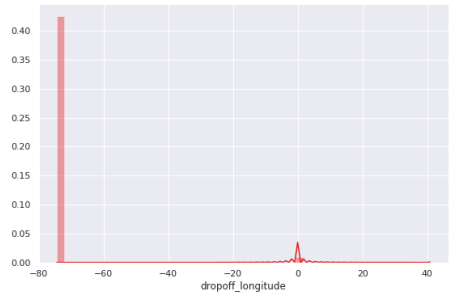
Further we will look at what Pre-Processing steps do this project was involved in. Some Histogram plots from seaborn library for each individual variable created using distplot() method. We can also see some Jointplots:

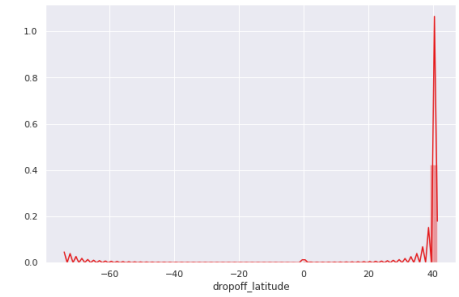
- They are used for Bivariate Analysis.
- Here we have plotted Scatter plot with Regression line between 2 variables along with separate Bar plots of both variables.
- Also, we have annotated Pearson correlation coefficient and p value.
- Plotted only for numerical/continuous variables
- Target variable ‘fare_amount’ Vs each numerical variable.

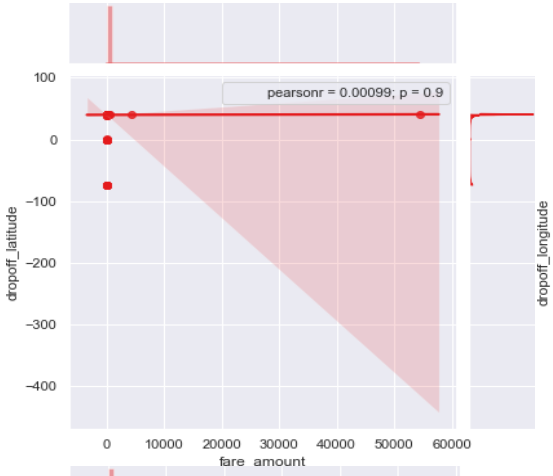

0.0008
0.0006
0.0004
0.0002
0.0000
0
10000
20000
30000
40000
50000
fare_amount

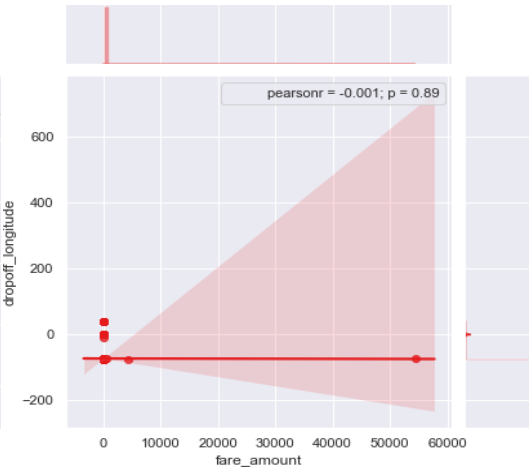

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−80
−60
−40
−20
0
20
40
pickup_longitude

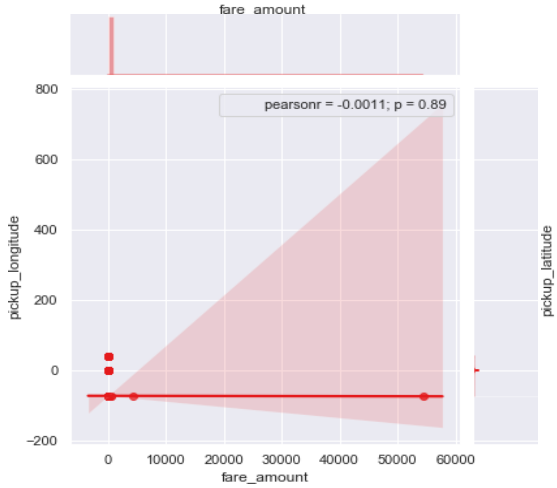

0.25
0.20
0.15
0.10
0.05
0.00
0
100
200
300
400
pickup_latitude

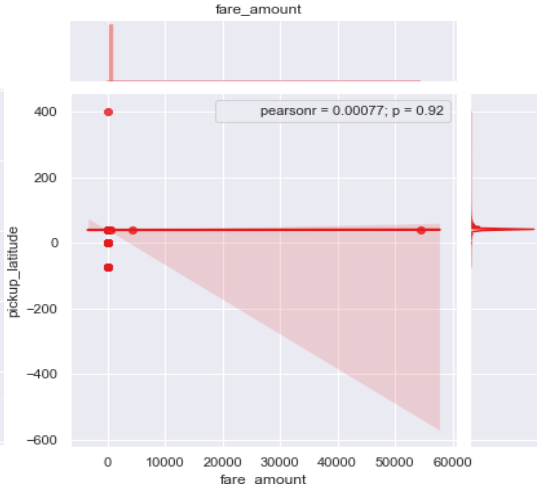

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−80
−60
−40
−20
0
20
40
dropoff_longitude


1.0
0.8
0.6
0.4
0.2
0.0
−60
−40
−20
0
20
40
dropoff_latitude

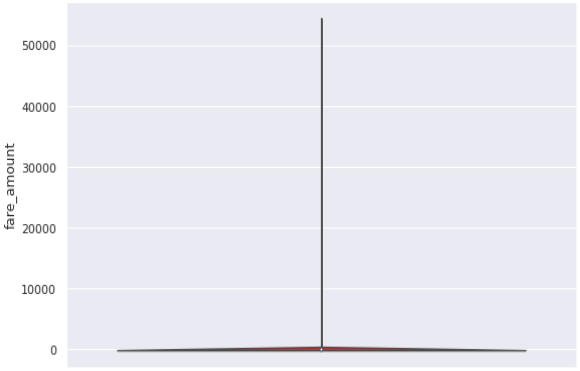

pearsonr = 0.00099; p = 0.9
100
0
−100
−200
−300
−400
dropoff_latitude
0
10000
20000
30000
40000
50000
60000
fare_amount

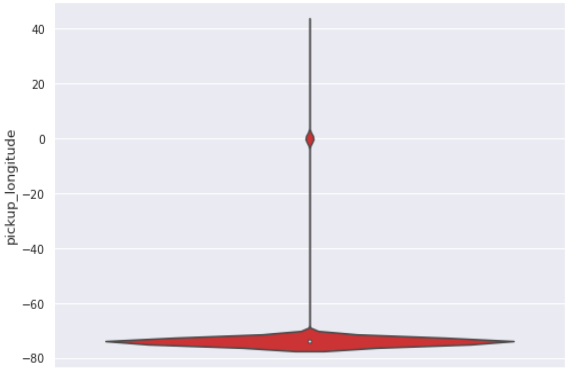

pearsonr = -0.001; p = 0.89
600
400
200
0
−200
dropoff_longitude
0
10000
20000
30000
40000
50000
60000
fare_amount

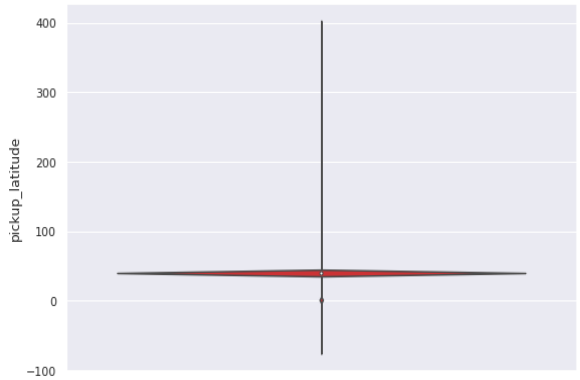

pearsonr = -0.0011; p = 0.89
800
600
400
200
0
−200
pickup_longitude
0
10000
20000
30000
40000
50000
60000
fare_amount

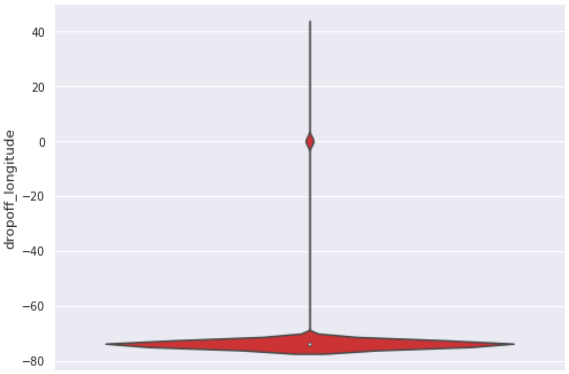

pearsonr = 0.00077; p = 0.92
400
200
0
−200
−400
−600
pickup_latitude
0
10000
20000
30000
40000
50000
60000
fare_amount

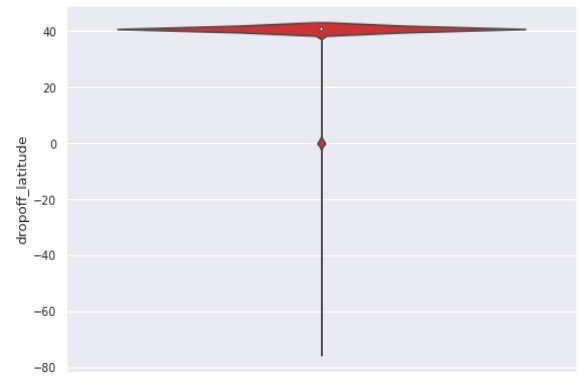
Some Violin Plots to get the idea about till what range is the variables is spread.



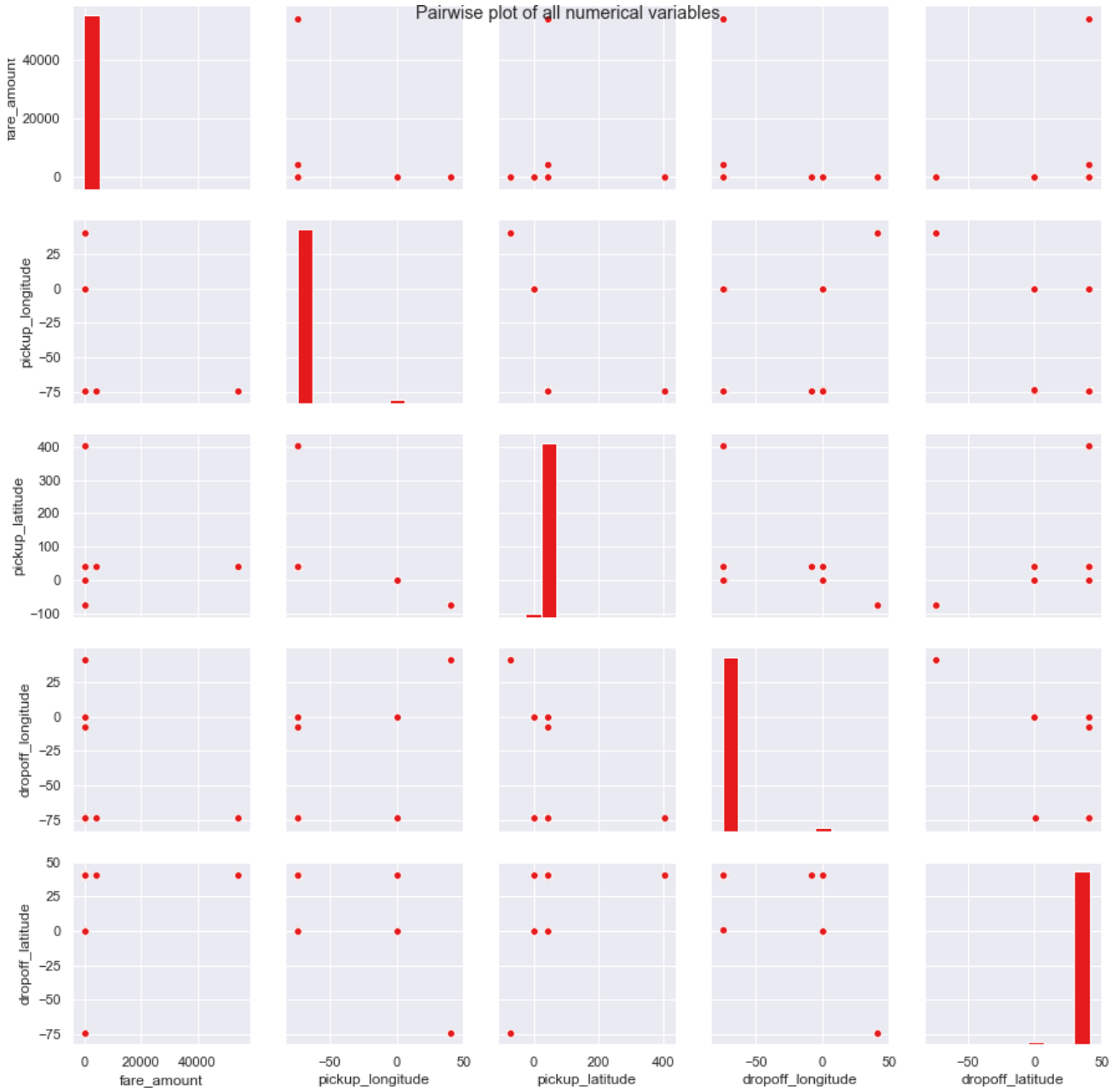








Pairwise Plots for all Numerical variables:



Outlier Analysis

In this step we will remove values in each variable which are not within desired range and we will consider them as outliers depending upon basic understanding of all the variables. You would think why haven't made those values NA instead of removing them well I did make them NA but it turned out to be a lot of missing values (NA's) in the dataset. Missing values percentage becomes very much high and then there will be no point of using that imputed data. Look at below 3 scenarios—

- If everything beyond range is made nan also except latitudes and longitudes, then:

Variables	Missing_percentage	
0	passenger_count	29.563702
1	pickup_latitude	1.966764
2	pickup_longitude	1.960540
3	dropoff_longitude	1.954316
4	dropoff_latitude	1.941868
5	fare_amount	0.186718
6	pickup_datetime	0.006224

After imputing above mentioned missing values kNN algorithm imputes every value to 0 at a particular row which was made nan using np.nan method:

```
fare_amount          0.0
pickup_longitude     0.0
pickup_latitude      0.0
dropoff_longitude    0.0
dropoff_latitude     0.0
passenger_count      0.0
Name: 1000, dtype: float64
```

- And If everything is dropped which are beyond range then below are the missing percentages 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

After imputing above mentioned missing values kNN algorithm values at a particular row which was made nan using np.nan method.

```
fare_amount          7.3698
pickup_longitude   -73.9954
pickup_latitude      40.7597
dropoff_longitude  -73.9876
dropoff_latitude     40.7512
passenger_count      2
Name: 1000, dtype: object
```

- If everything beyond range is made nan except passenger_count:

Variables	Missing_percentage	
0	pickup_latitude	1.951342
1	dropoff_longitude	1.951342
2	pickup_longitude	1.945087
3	dropoff_latitude	1.938833
4	passenger_count	0.343986
5	fare_amount	0.181375
6	pickup_datetime	0.006254

After imputing above mentioned missing values kNN algorithm imputes every value to 0 at a particular row which was made nan using np.nan method:

```
fare_amount    0.0
pickup_longitude     0.0
pickup_latitude      0.0
dropoff_longitude    0.0
dropoff_latitude     0.0
passenger_count      0.0
Name: 1000, dtype: float64
```

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

And below is the Standard deviation of particular variable which has missing values in them:

fare_amount 435. 982171

passenger_count 1.266096

dtype: float64

We'd tried central statistical methods and algorithmic method--KNN to impute missing values in the dataset:

1. For Passenger_count:

    Actual value = 1

    Mode = 1

    KNN = 2

We will choose the KNN method here because it maintains the standard deviation of variable. We will not use Mode method because whole variable will be more biased towards 1 passenger_count also passenger_count has maximum value equals to 1

2. For fare_amount:

Actual value = 7.0,

Mean = 15.117,

Median = 8.5,

KNN = 7.369801

We will Choose KNN method here because it imputes value closest to actual value also it maintains the Standard Deviation of the variable.

Standard deviation for passenger_count and fare_amount after KNN imputation:

fare_amount 435.661995

passenger_count 1.264322

dtype: float64

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,

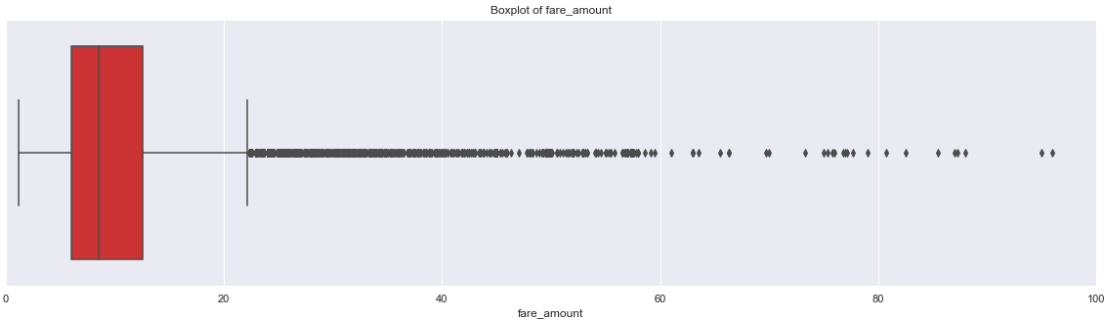
I. We replaced them with NaN values or we can say created missing values.

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 laitudes and longitudes.

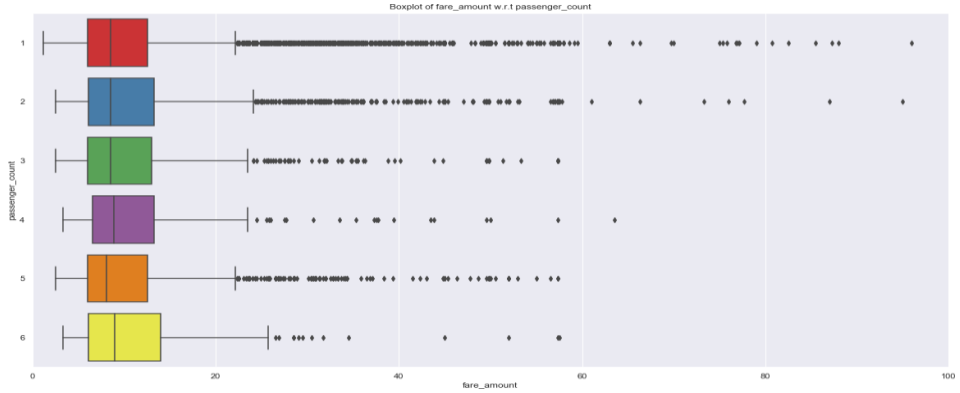
Univariate Boxplot

Boxplot for target variable ie. fare_amount.



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 imputation method and K value is 3.

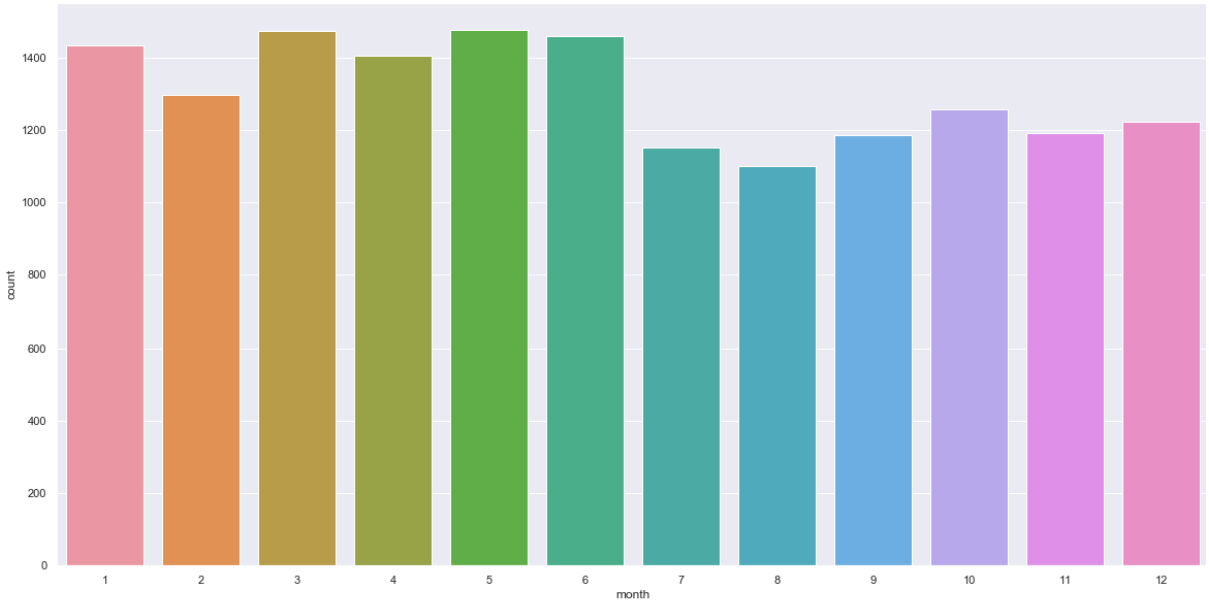
Feature Engineering

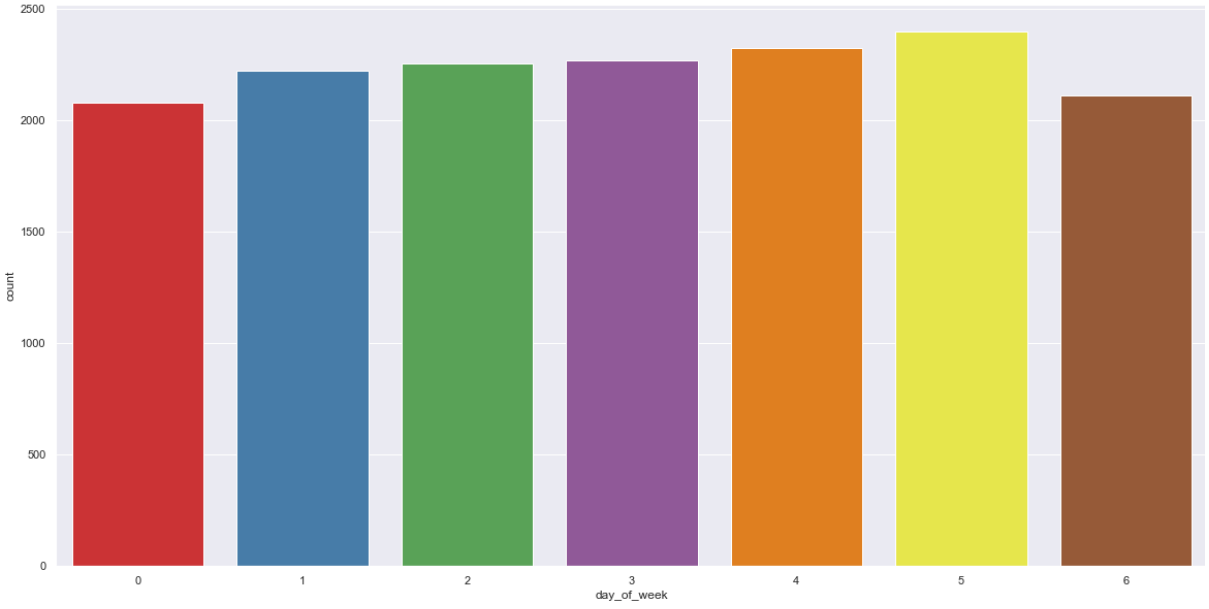
Feature Engineering is used to drive new features from existing features.

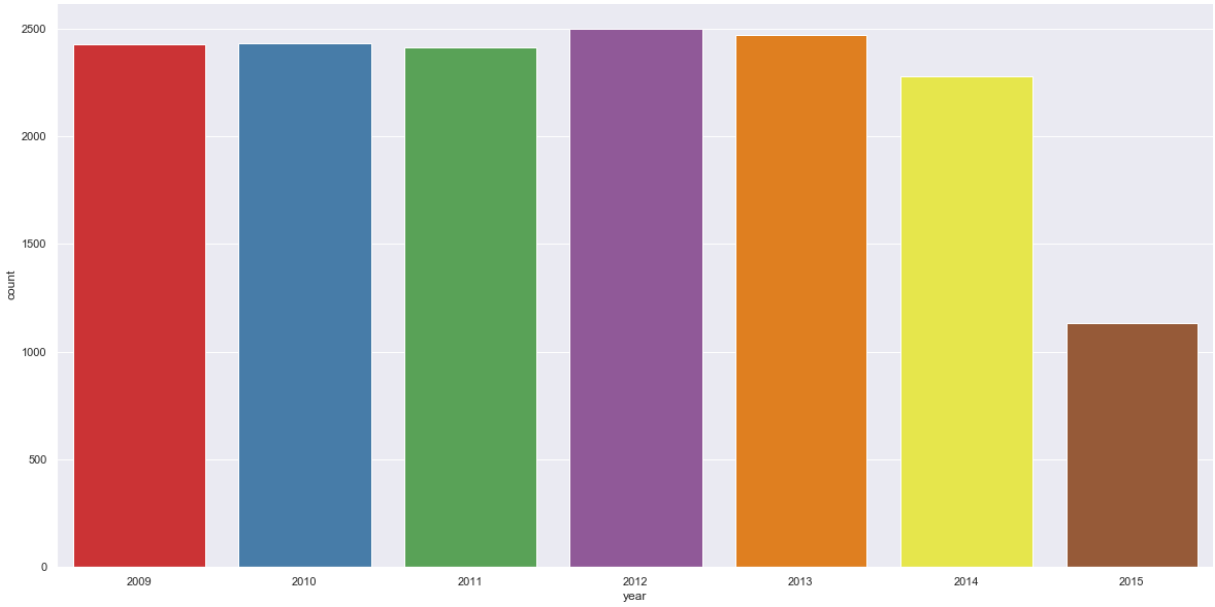
1. *For ‘pickup_datetime’ variable:*

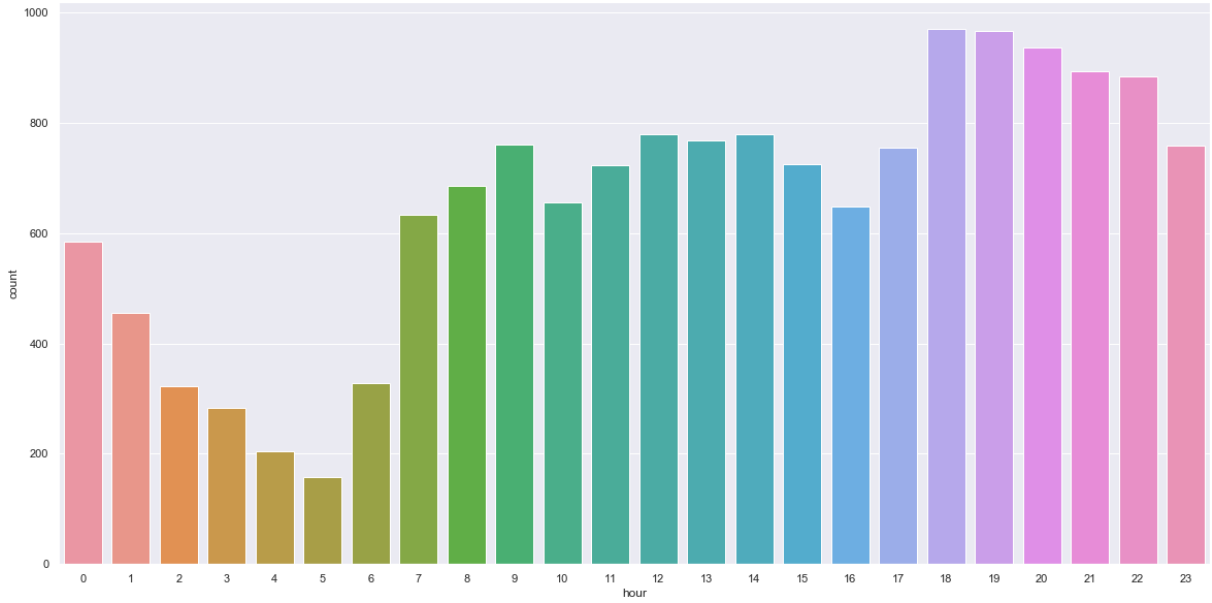
We will use this timestamp variable to create new variables. New features will be year, month, day_of_week, hour.

- ‘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.

2. *For 'passenger_count' variable:*

As passenger_count is a categorical variable we will one-hot-encode it.

3. *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.

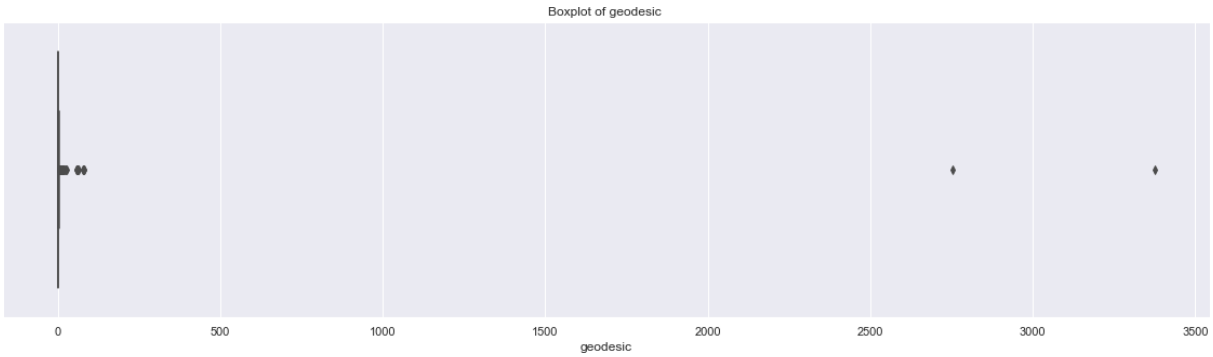
Columns in training data after feature engineering:

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')

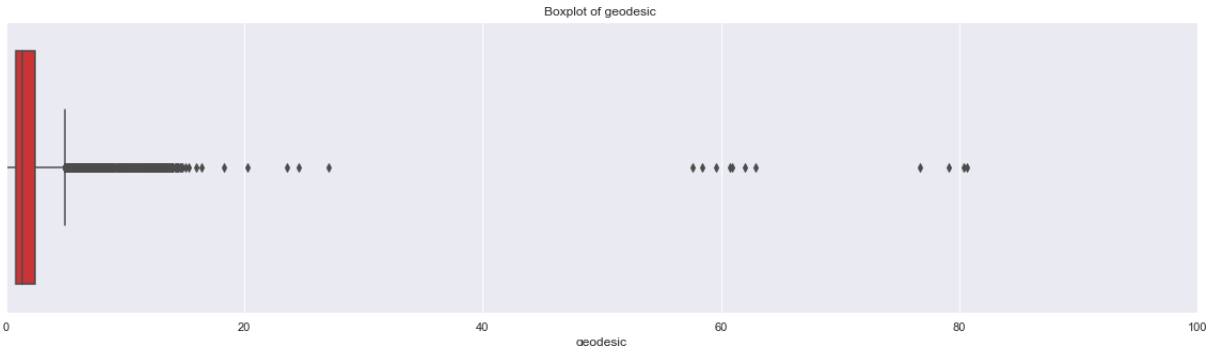
Columns in testing data after feature engineering:

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 a cab cannot go up to 3400 miles. Boxplot of ‘geodesic’ for range 0 to 100 miles.



We will treat these outliers like we previously did.

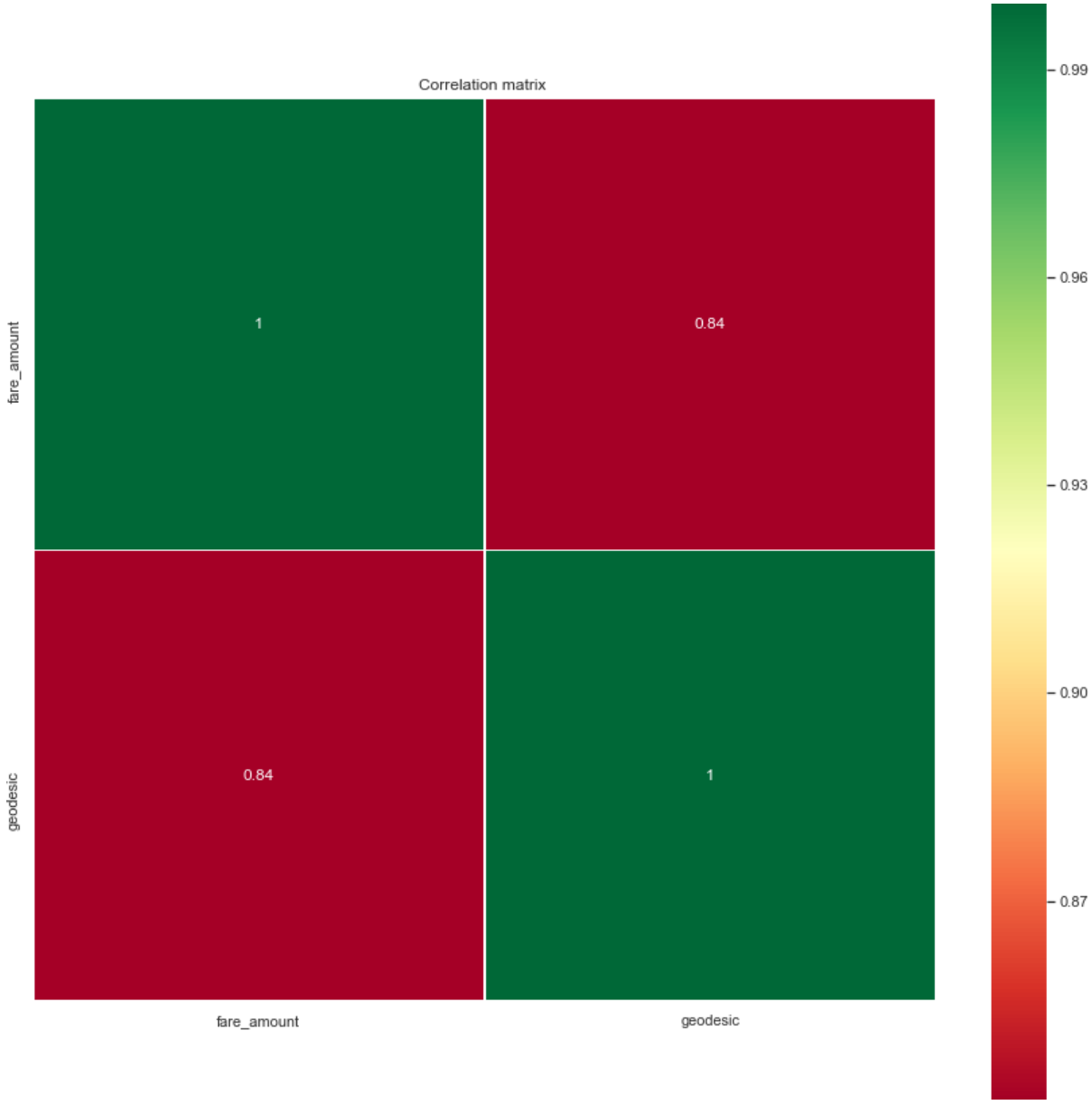
Feature Selection

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.

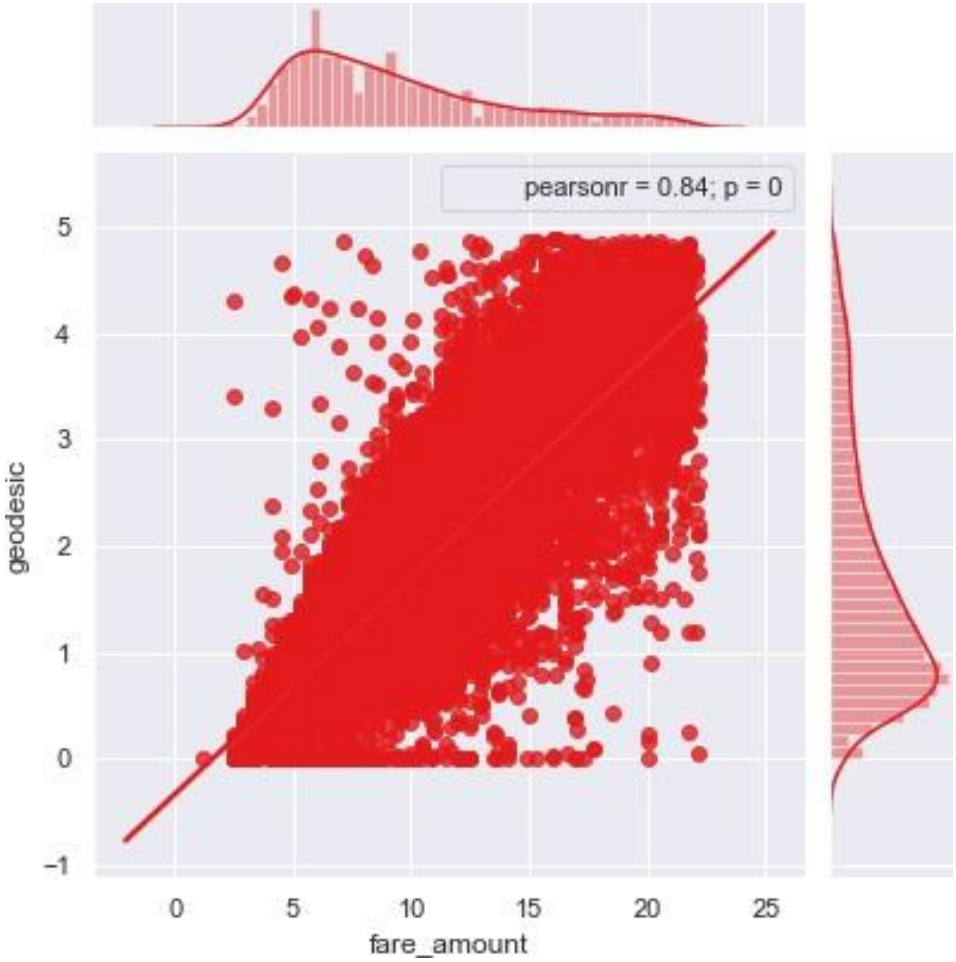
Further below are some types of test involved for feature selection:

1. *Correlation analysis* – This requires only numerical variables. Therefore, we will filter out 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.



Jointplot between ‘geodesic’ and ‘fare_amount’:



2. *Chi-Square test of independence* – Unlike correlation analysis we will filter out only categorical 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.

    i. Assumption for chi-square test: Dependency between Independent variable and dependent variable should be high and there should be no dependency among independent variables.
    ii. Before proceeding to calculate chi-square statistic, we do the hypothesis testing: Null hypothesis: 2 variables are independent.

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

I. For theorical 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.
II. 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.

3. *Analysis of Variance (ANOVA) Test* –

i. It is carried out to compare between each group in a categorical variable.
ii. ANOVA only lets us know the means for different groups are same or not. It

This 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.
- 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.

4. *Multicollinearity* - In regression, "multicollinearity" refers to predictors that are correlated 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.

- Multicollinearity increases the standard errors of the coefficients.
- Increased standard errors in turn means that coefficients for some independent variables may be found not to be significantly different from 0. 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.
- 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.
- 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
0	15.268789	Intercept
1	1.040670	passenger_count_2[T.1.0]
2	1.019507	passenger_count_3[T.1.0]
3	1.011836	passenger_count_4[T.1.0]
4	1.024990	passenger_count_5[T.1.0]
5	1.017206	passenger_count_6[T.1.0]
6	1.642247	season_spring[T.1.0]
7	1.552411	season_summer[T.1.0]
8	1.587588	season_winter[T.1.0]
9	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.

Feature Scaling

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

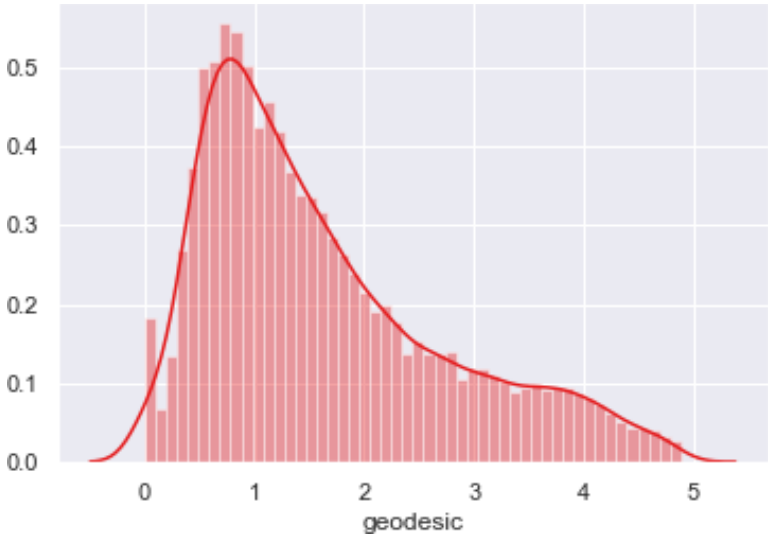
- *Normalization:* Normalization refer to the dividing of a vector by its length. normalization normalizes the data in the range of 0 to 1. It is generally used when we are planning to use distance method for our model development purpose such as KNN. Normalizing the data improves convergence of such algorithms. Normalization 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.

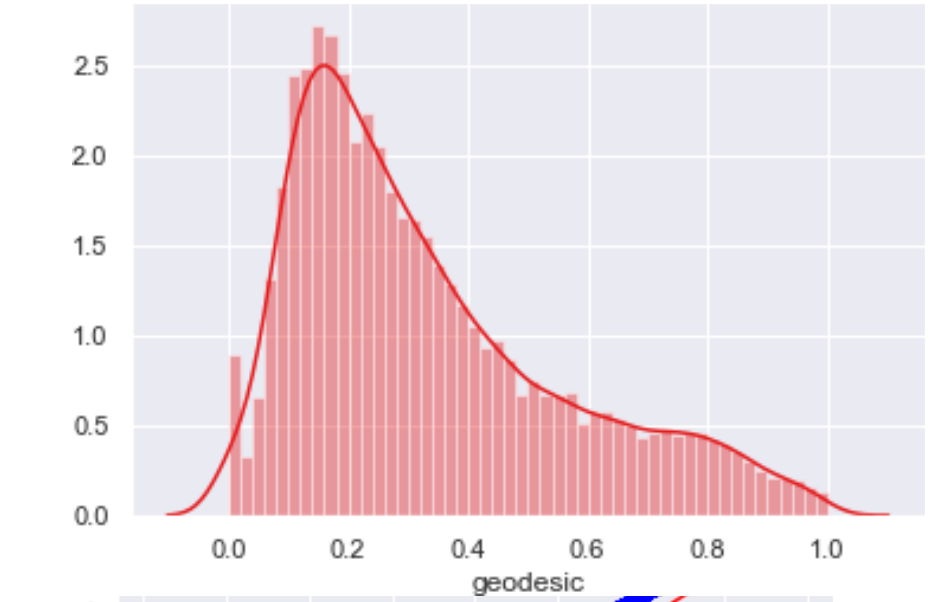
- We have checked variance for each column in dataset before Normalization.
- High variance will affect the accuracy of the model. So, we want to normalize that variance.

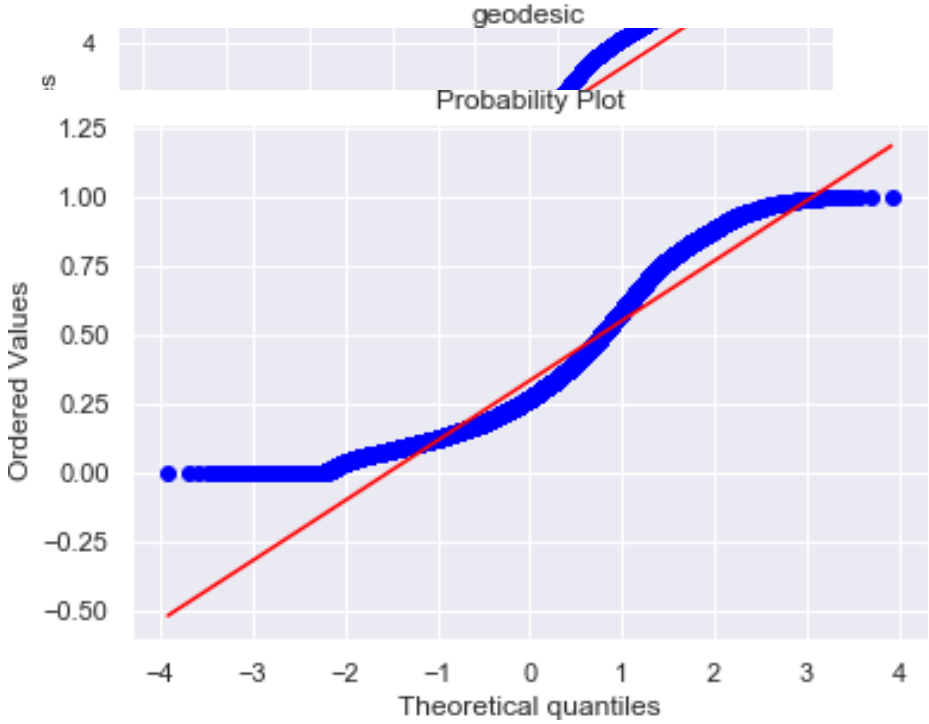
Graphs based on which standardization was chosen. It is performed only on continuous variables. distplot() for 'geodesic' feature before normalization is seen below:



qq-probability plot before normalization is below:

distplot() for 'geodesic' feature after normalization:





qq probability plot after normalization:

Train and Test datasets

- We have used sklearn's *train_test_split()* method to divide whole Dataset into train and validation datset.
- 25% is in validation dataset and 75% is in training data.
- 11745 observations in training and 3915 observations in validation dataset.
- We will test the performance of model on validation dataset.
- The model which performs best will be chosen to perform on test dataset provided along with original train dataset.
- X_train, y_train -- are train subset.

- X_test, y_test -- are test subset.

Hyperparameter Optimization

To find the optimal hyperparameter we have used sklearn.model_selection.GridSearchCV. and sklearn.model_selection.RandomizedSearchCV.

GridSearchCV tries all the parameters that we provide it and then returns the best suited parameter for data. We gave parameter dictionary to GridSearchCV which contains keys which are parameter names and values are the values of parameters which we want to try for.

Below are best hyperparameter we found for different models:

- Multiple Linear Regression: Tuned Decision reg Parameters: *{'copy_X': True, 'fit_intercept': True}.* Best score is 0.7354470072210966

- Ridge Regression: Tuned Decision ridge Parameters: *{'alpha': 0.0005428675439323859, 'max_iter': 500, 'normalize': True}*. Best score is 0.7354637543642097

- Lasso Regression: Tuned Decision lasso Parameters: {'alpha': 0.00021209508879201905, 'max_iter': 1000, 'normalize': False} Best score is 0.40677751497154

- Decision Tree Regression: Tuned Decision Tree Parameters: *{'max_depth': 6, 'min_samples_split': 2}* Best score is 0.7313489270203365.

- Random Forest Regression: Tuned Decision Forest Parameters: *{'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 4, 'max_features': 'auto', 'max_depth': 9, 'bootstrap': True}.* Best score is 0.7449373558797026

- Xgboost regression: 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}*. Best score is 0.7489532917329004

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:

- Linear Regression
- Ridge Regression
- Lasso Regression
- Decision Tree
- Random Forest
- Xgboost Regression

We will evaluate performance on validation dataset which was generated using Sampling. We will deal with specific error metrics like the following:

- R-squared
- Adjusted R-squared

- MAPE (Mean Absolute Percentage Error)
- MSE (Mean square Error)
- RMSE (Root Mean Square Error)
- RMSLE (Root Mean Squared Log Error)

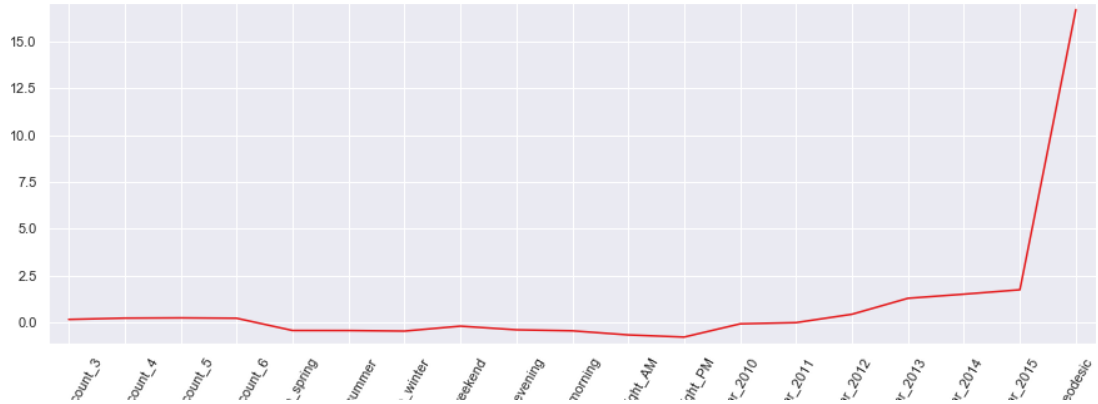
Model Performance

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

- Multiple Linear Regression:

Error Metrics	R-squared	Adj r squared	MAPE	MSE	RMSE	RMSLE
Train	0.734	0.733	18.73	5.28	2.29	0.21
Test	0.719	0.7406	18.96	5.29	2.30	0.21

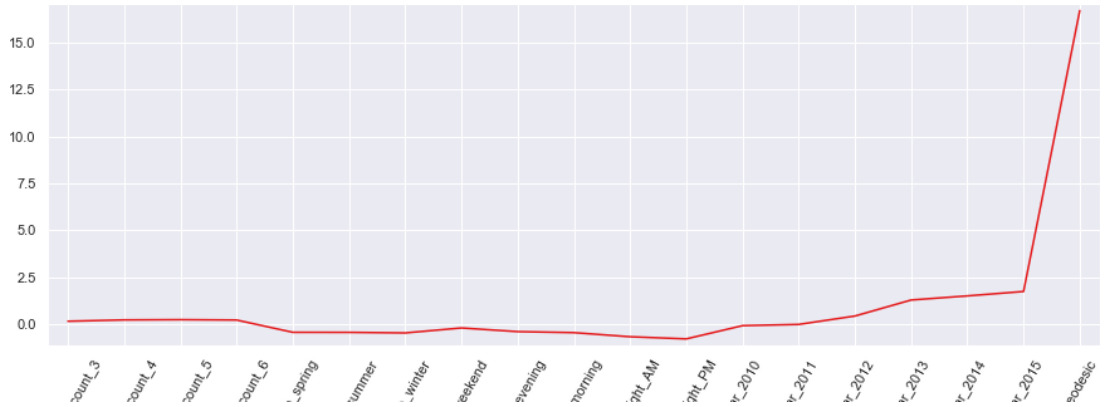
Line Plot for Coefficients of Multiple Linear regression:



- Ridge Regression:

Error Metrics	R-squared	Adj. R-squared	MAPE	MSE	RMSE	RMSLE
Train	0.7343	0.733	18.74	5.28	2.29	0.21
Test	0.7419	0.7406	18.96	5.29	2.3	0.21

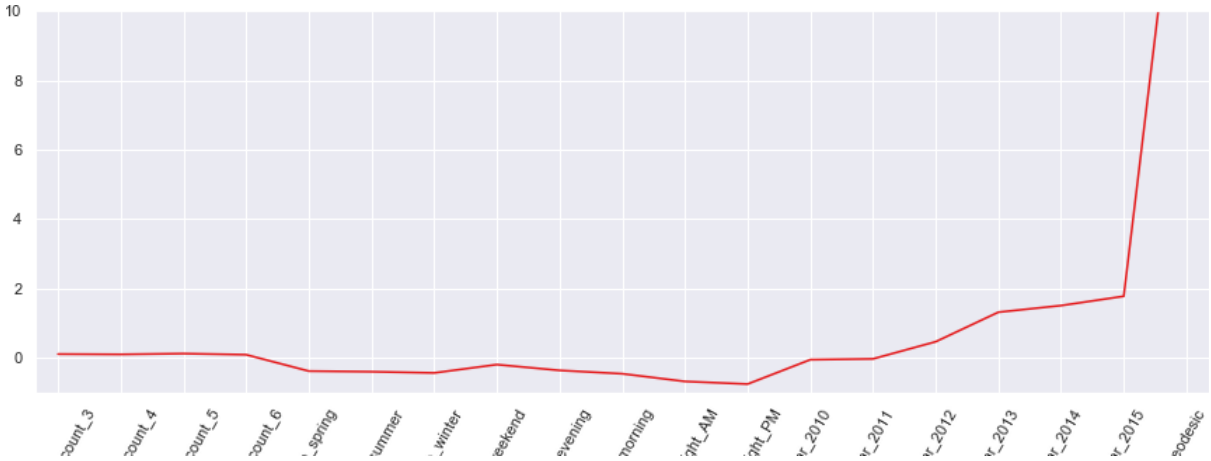
Line Plot for Coefficients of Ridge regression:



- Lasso Regression

Error Metrics	R-squared	Adj R-squared	MAPE	MSE	RMSE	RMSLE
Train	0.7341	0.7337	18.75	5.28	2.29	0.21
Test	0.7427	0.7415	18.95	5.27	2.29	0.21

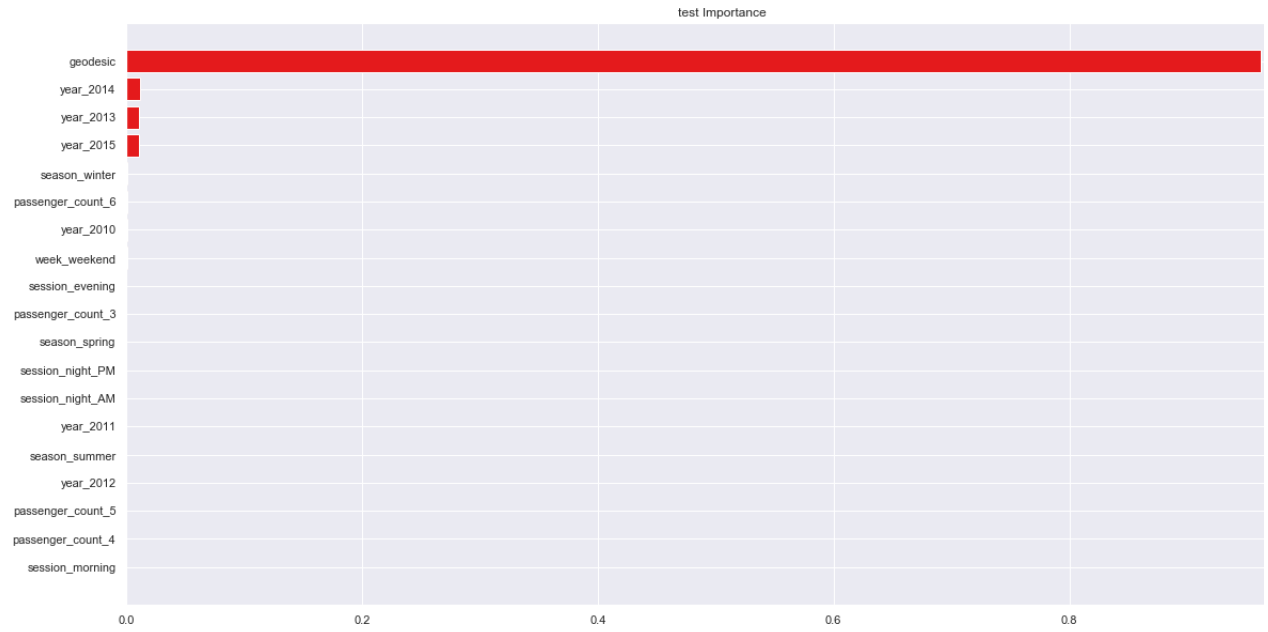
Line Plot for Coefficients of Lasso regression:



- Decision Tree Regression

Error Metrics	R-squared	Adj R-squared	MAPE	MSE	RMSE	RMSLE
Train	0.7471	0.7467	18.54	5.02	2.24	0.20
Test	0.7408	0.7396	19.07	5.31	2.30	0.21

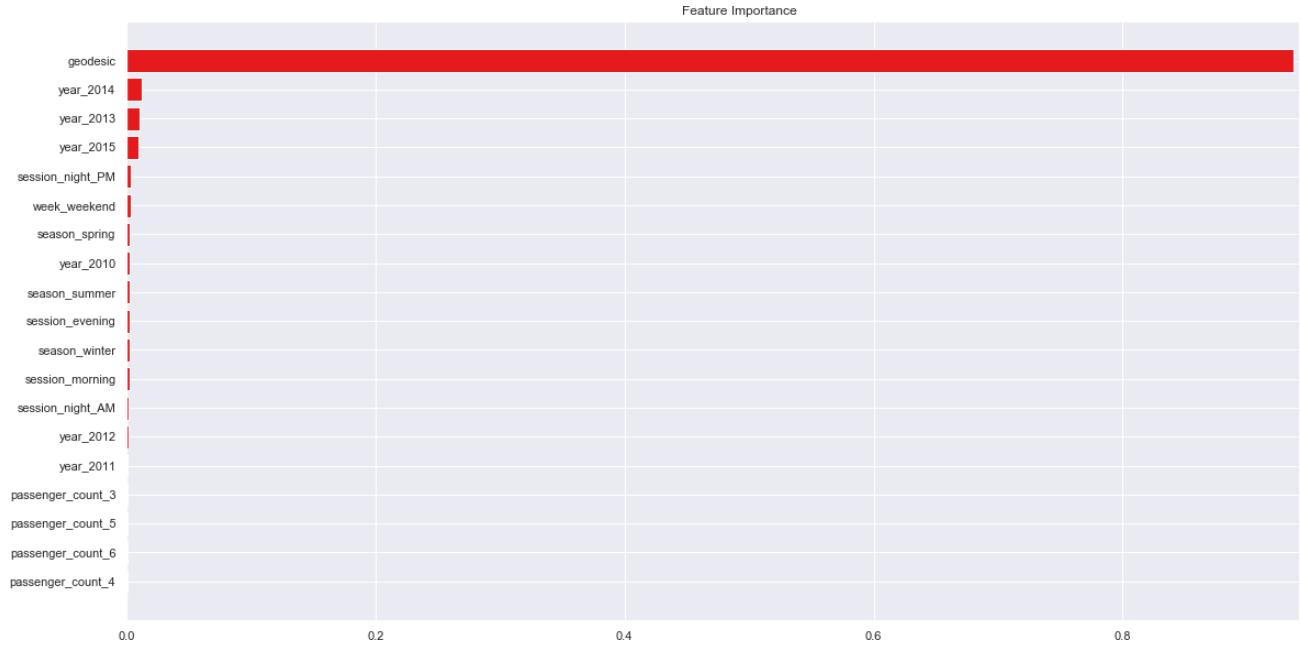
Bar Plot of Decision tree Feature Importance:



- Random Forest Regression

Error Metrics	R-squared	Adj R-squared	MAPE	MSE	RMSE	RMSLE
Train	0.7893	0.7889	16.95	4.19	2.04	0.19
Test	0.7542	0.7530	18.56	5.09	2.24	0.20

Bar Plot of Random Forest Feature Importance:



Cross validation scores: *[-5.19821639 -5.18058997 -5.11306209 -5.15194135 -5.14644304]*

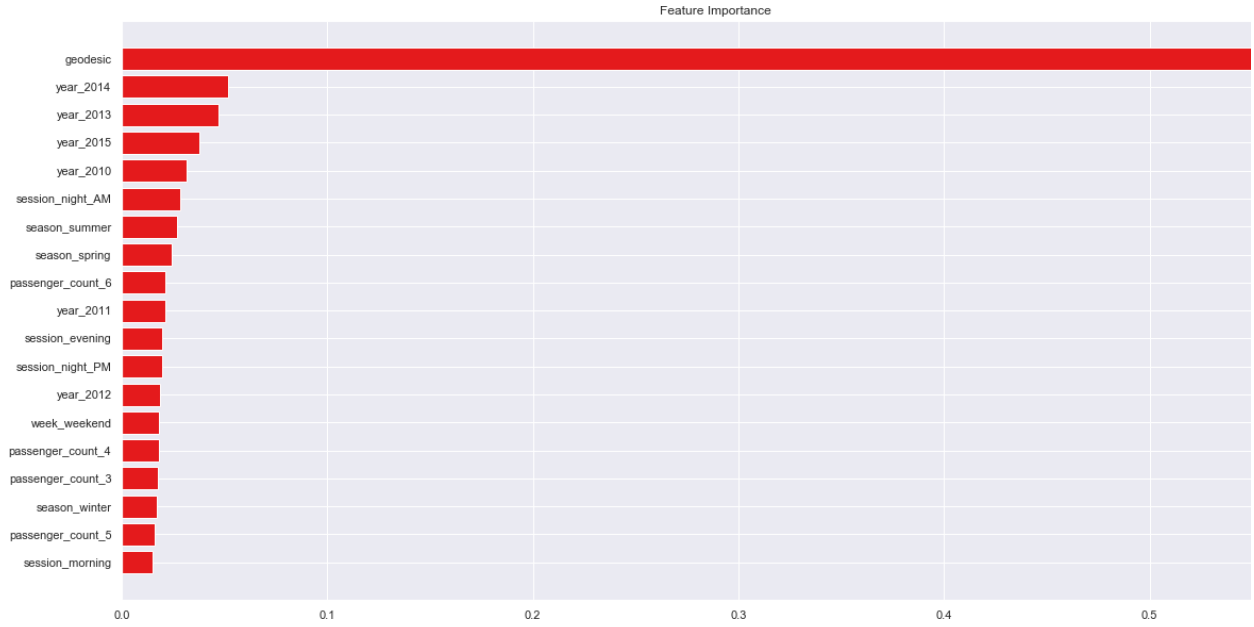
Average 5-Fold CV Score: *-5.158050568861664*

- XGboost Regression (Improves accuracy)

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}*

Error Metrics	R-squared	Adj R-squared	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:



Finalized model

We will create standalone model on entire training dataset and save model for later use.

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

r square 0.7564292952182666

Adjusted r square:0.7561333973032505 MAPE:18.100202501103993

MSE: 4.881882644209386

RMSE: 2.2094982788428204

RMSLE: 0.2154998534679604

RMSLE: 0.20415655796958632

Feature importance for the finalized model: