Python-Code

In [1]:*# loading the required libraries*

**import** **os**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **scipy.stats** **as** **stats**

**from** **fancyimpute** **import** KNN

**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** Linear Regression

**from** **sklearn.model\_selection** **import** GridSearchCV

**from** **sklearn.model\_selection** **import** RandomizedSearchCV

**from** **sklearn.model\_selection** **import** cross\_val\_score

**from** **sklearn.ensemble** **import** RandomForest Regressor

**from** **sklearn.tree** **import** DecisionTree Regressor

**from** **xgboost** **import** XGB Regressor

**import** **xgboost** **as** **xgb**

**from** **sklearn.externals** **import** joblib

In [2]: *# set the working directory*

os.chdir('C:/Users/user/Python3')

os.getcwd()

Out[2]: ‘C:/Users/user/Python3’

**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 [3]:*# Importing data*

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

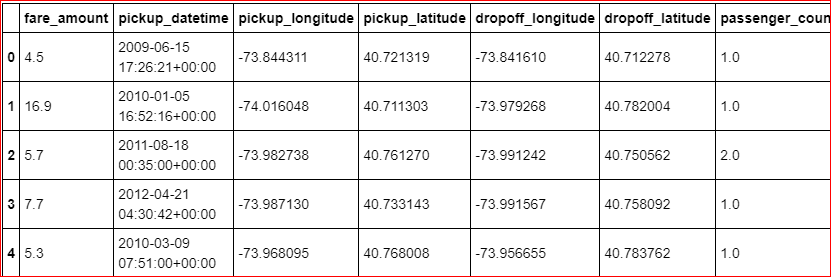
test = pd.read\_csv('test.csv')

data=[train,test]

**for** i **in** data:i['pickup\_datetime'] = pd.to\_datetime(i['pickup\_datetime'],errors='coerce')

train.head(5)

Out[3]:



In [4]:train.info()

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

RangeIndex: 16067 entries, 0 to 16066

Data columns (total 7 columns):

fare\_amount 16042 non-null float64

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

pickup\_longitude 16067 non-null float64

pickup\_latitude 16067 non-null float64

dropoff\_longitude 16067 non-null float64

dropoff\_latitude 16067 non-null float64

passenger\_count 16012 non-null float64

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

memory usage: 878.7 KB

In [6]:test.head(5)

Out[6]: 

In [7]:test.info()

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

RangeIndex: 9914 entries, 0 to 9913

Data columns (total 6 columns):

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

pickup\_longitude 9914 non-null float64

pickup\_latitude 9914 non-null float64

dropoff\_longitude 9914 non-null float64

dropoff\_latitude 9914 non-null float64

passenger\_count 9914 non-null int64

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

memory usage: 464.8 KB

**EDA**

* we will convert passenger\_count into a categorical variable because passenger\_count is not a continuous variable.
* passenger\_count cannot take continuous values. and also they are limited in number if its a cab.

In [73]:cat\_var=['passenger\_count']num\_var=['fare\_amount','pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']

**Graphical EDA - Data Visualization**

In [38]:*# setting up the sns for plots*

sns.set(style='darkgrid',palette='Set1')

**Some histogram plots from seaborn library**

In [75]: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()

* **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 [76]: \_ = sns.jointplot(x='fare\_amount',y='pickup\_longitude',data=train,kind = 'reg')\_.annotate(stats.pearsonr)

*# plt.savefig('jointfplo.png')*

plt.show()

In [77]:\_ = sns.jointplot(x='fare\_amount',y='pickup\_latitude',data=train,kind = 'reg')

\_.annotate(stats.pearsonr)

*# plt.savefig('jointfpla.png')*

plt.show()

In [78]: \_ = sns.jointplot(x='fare\_amount',y='dropoff\_longitude',data=train,kind = 'reg')\_.annotate(stats.pearsonr)

*# plt.savefig('jointfdlo.png')*

plt.show()

In [79]:\_ = sns.jointplot(x='fare\_amount',y='dropoff\_latitude',data=train,kind = 'reg')

\_.annotate(stats.pearsonr)

*# plt.savefig('jointfdla.png')*

plt.show()

Pairwise plots for all numerical variables

In [81]: \_ =sns.pairplot(data=train[num\_var],kind='scatter',dropna=**True**)

\_.fig.suptitle('Pairwise plot of all numerical variables')

*# plt.savefig('Pairwise.png')*

plt.show()

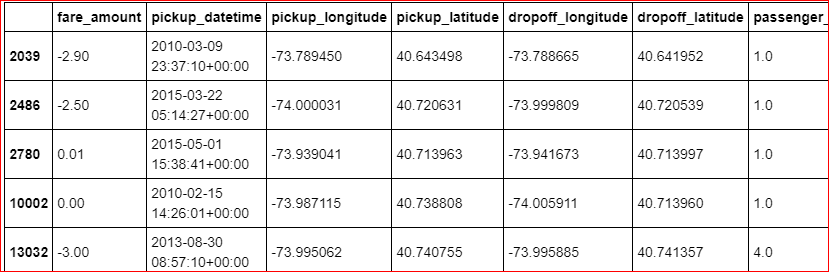
**Removing values which are not within desired range(outliers) depending upon basic understanding of dataset.**

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)

Out[23]: 5

In [24]: train[train['fare\_amount']<1]

Out[24]: 

In [4]: train = train.drop(train[train['fare\_amount']<1].index, axis=0)

2. Passenger count cannot be <1 or >6. Removing the outliers

In [79]: **for** i **in** range(4,11):

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

passenger\_count above4=1367

passenger\_count above5=322

passenger\_count above6=20

passenger\_count above7=20

passenger\_count above8=20

passenger\_count above9=20

passenger\_count above10=20

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

In [5]: train = train.drop(train[train['passenger\_count']>6].index, axis=0)

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

In [89]: sum(train['passenger\_count']>6)

Out[89]: 0

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

In [10]:

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)))

pickup\_longitude above 180=0

pickup\_longitude below -180=0

pickup\_latitude above 90=1

pickup\_latitude below -90=0

dropoff\_longitude above 180=0

dropoff\_longitude below -180=0

dropoff\_latitude below -90=0

dropoff\_latitude above 90=0

* **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 [11]:

**for** i **in** ['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']:

print(i,'equal to 0=**{}**'.format(sum(train[i]==0)))

pickup\_longitude equal to 0=315

pickup\_latitude equal to 0=315

dropoff\_longitude equal to 0=314

dropoff\_latitude equal to 0=312

**These are values which are equal to 0. We will remove them.**

In [6]: 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 [7]: train.shape

Out[7]: (15661, 7)

**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 latitudes and longitudes.
* Univariate Boxplots: Boxplots for all Numerical Variables including target variable.

In [41]: plt.figure (figsize=(20,5))

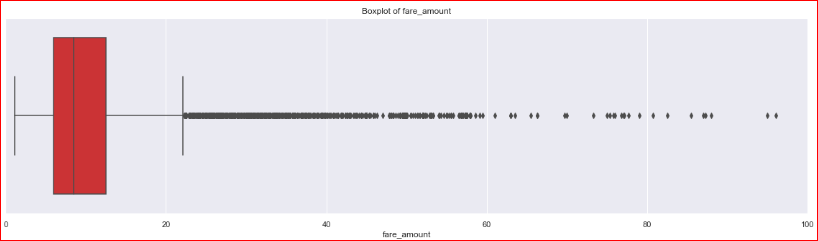
plt.xlim(0,100)

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 [42]:

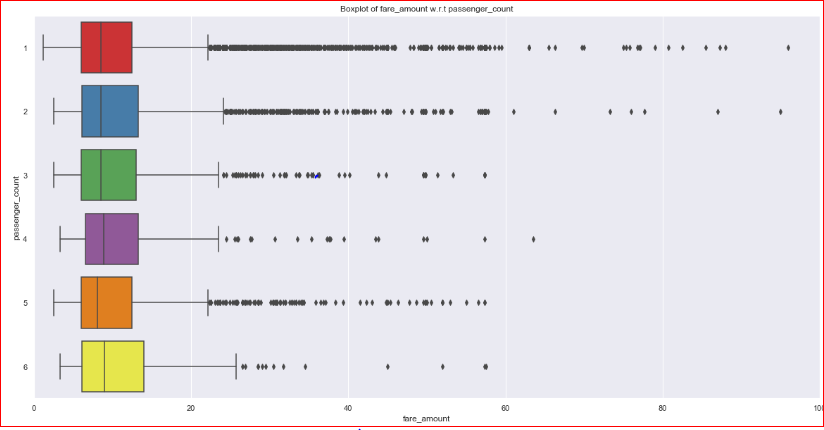
plt.figure (figsize=(20,10))

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')*

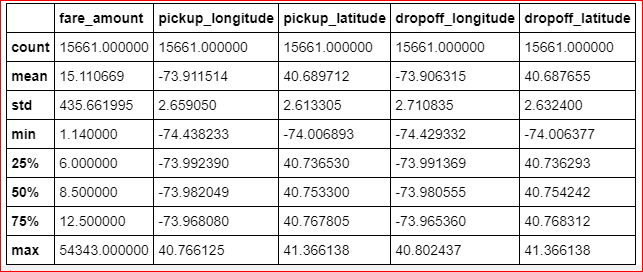
plt.show()



In [43]:

train.describe()

Out[43]:



In [44]:

train['passenger\_count'].describe()

Out[44]:

count 15661

unique 6

top 1

freq 11056

Name: passenger\_count, dtype: int64

**Outlier Treatment**

* As we can see from the above Boxplots there are outliers in the train dataset.
* Reconsider pickup\_longitude, etc.

In [51]:

**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 [52]:

*# for i in num\_var:*

outlier\_treatment('fare\_amount')

*# outlier\_treatment('pickup\_longitude')*

*# outlier\_treatment('pickup\_latitude')*

*# outlier\_treatment('dropoff\_longitude')*

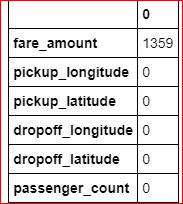
*# outlier\_treatment('dropoff\_latitude')*

12.5 6.0

-3.75 22.25

In [53]: pd.DataFrame(train.isnull().sum())

Out[53]:



In [54]: train.std()

Out[54]:

fare\_amount 4.136102

pickup\_longitude 2.659050

pickup\_latitude 2.613305

dropoff\_longitude 2.710835

dropoff\_latitude 2.632400

passenger\_count 1.264322

dtype: float64

In [55]:*#Imputing with missing values using KNN*

train = pd.DataFrame(KNN(k = 3).fit\_transform(train), columns = train.columns, index=train.index)

In [56]: train.std()

Out[56]:

fare\_amount 4.476970

pickup\_longitude 2.659050

pickup\_latitude 2.613305

dropoff\_longitude 2.710835

dropoff\_latitude 2.632400

passenger\_count 1.264322

dtype: float64

In [57]:

train['passenger\_count'].describe()

Out[57]:

count 15661.000000

mean 1.649192

std 1.264322

min 1.000000

25% 1.000000

50% 1.000000

75% 2.000000

max 6.000000

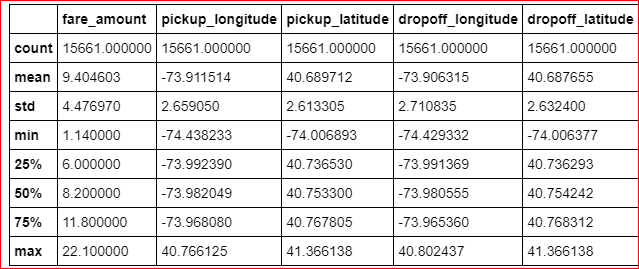
Name: passenger\_count, dtype: float64

In [58]:

train['passenger\_count']=train['passenger\_count'].astype('int').round().astype('object').astype('category')

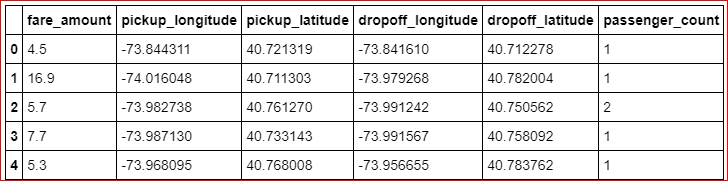
In [59]: train.describe()

Out[59]:



In [60]: train.head()

Out[60]:



In [61]:

df2 = train.copy()

*# train= df2.copy()*

In [62]: train.shape

Out[62]: (15661, 6)

**Missing Value Analysis**

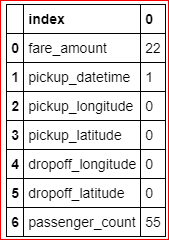
In [8]: *#Create dataframe with missing percentage*

missing\_val = pd.DataFrame(train.isnull().sum())

*#Reset index*

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

Out[8]:



* As we can see there are some missing values in the data.
* 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 [9]: *#Rename variable*

missing\_val = missing\_val.rename(columns = {'index': 'Variables', 0: 'Missing\_percentage'})missing\_val

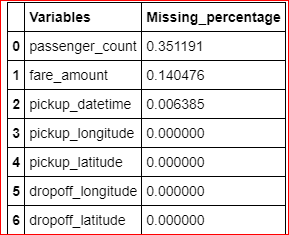
*#Calculate percentage*

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

Out[9]:



* 1. For Passenger\_count:
* Actual value = 1
* Mode = 1
* KNN = 2

In [13]: *# Choosing a random values to replace it as NA*

train['passenger\_count'].loc[1000]

Out[13]: 1.0

In [14]:*# Replacing 1.0 with NA*

train['passenger\_count'].loc[1000] = np.nan

train['passenger\_count'].loc[1000]

Out[14]: nan

In [15]:*# Impute with mode*

train['passenger\_count'].fillna(train['passenger\_count'].mode()[0]).loc[1000]

Out[15]:1.0

* 1. For fare\_amount:
* Actual value = 7.0,
* Mean = 15.117,
* Median = 8.5,
* KNN = 7.369801

In [16]: *# Choosing a random values to replace it as NA*

a=train['fare\_amount'].loc[1000]

print('fare\_amount at loc-1000:**{}**'.format(a))

*# Replacing 7.0 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]))

**Fare\_amount at loc-1000: 7.0**

**Value after replacing with nan: nan**

**Value if imputed with mean: 15.118196060877201**

**Value if imputed with median: 8.5**

In [17]: train.std()

Out[17]:

fare\_amount 435.982171

pickup\_longitude 2.659050

pickup\_latitude 2.613305

dropoff\_longitude 2.710835

dropoff\_latitude 2.632400

passenger\_count 1.266096

dtype: float64

In [18]: 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 [19]: pickup\_datetime= pd.DataFrame(train['pickup\_datetime'])

In [20]: *# Imputing with missing values using KNN*

train = pd.DataFrame(KNN(k = 19).fit\_transform(train.drop('pickup\_datetime',axis=1),columns=columns, index= train.index)

In [21]: train.std()

Out[21]:

fare\_amount 435.661995

pickup\_longitude 2.659050

pickup\_latitude 2.613305

dropoff\_longitude 2.710835

dropoff\_latitude 2.632400

passenger\_count 1.264138

dtype: float64

In [23]:

train['passenger\_count'].head(5)

Out[23]:

0 1.0

1 1.0

2 2.0

3 1.0

4 1.0

Name: passenger\_count, dtype: float64

In [24]:

train['passenger\_count']=train['passenger\_count'].astype('int')

In [26]:

train['passenger\_count'].unique()

Out[26]:

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

In [27]:

train['passenger\_count']= train['passenger\_count'].round().astype('object').astype('category',ordered=**True**)

In [28]:

train['passenger\_count'].unique()

Out[28]:

[1, 2, 3, 6, 5, 4]

Categories (6, int64): [1 < 2 < 3 < 4 < 5 < 6]

In [29]: train.loc[1000]

Out[29]:

fare\_amount 7.3698

pickup\_longitude -73.9954

pickup\_latitude 40.7597

dropoff\_longitude -73.9876

dropoff\_latitude 40.7512

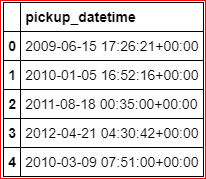
passenger\_count 2

Name: 1000, dtype: object

* 1. Now about missing value in pickup\_datetime

In [30]: pickup\_datetime.head()

Out[30]:



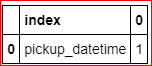
In [31]:*#Create dataframe with missing percentage*

missing\_val = pd.DataFrame(pickup\_datetime.isnull().sum())

*#Reset index*

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

Out[31]:

****

In [32]: pickup\_datetime.shape

Out[32]: (15661, 1)

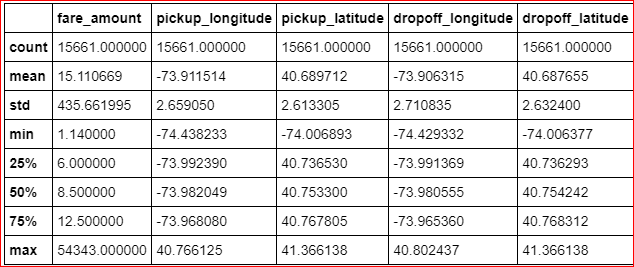
In [33]: train.shape

Out[33]: (15661, 6)

* **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 16041 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 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 [36]: train.describe()

Out[36]:



**Feature Engineering**

**1. Feature Engineering for (pickup\_datetime** ) **timestamp variable**

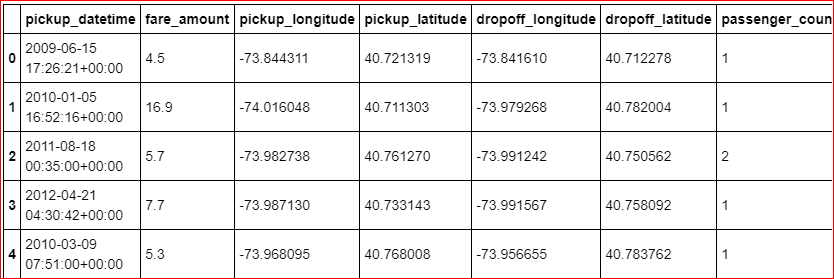
* we will derive new features from pickup\_datetime variable
* new features will be year, month, day\_of\_week, hour

In [63]: *# we will Join 2 Dataframes pickup\_datetime and train*

train = pd.merge(pickup\_datetime,train,right\_index=**True**, left\_index=**True**)

train.head(5)

Out[63]:



In [64]: train.shape

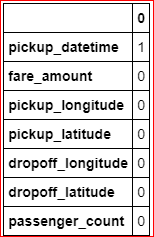
Out[64]: (15661, 7)

In [65]: train=train.reset\_index(drop=**True**)

**As we discussed in Missing value imputation step about dropping missing value, we will do it now.**

In [66]: pd.DataFrame(train.isna().sum())

Out[66]:



In [67]: train= train.dropna()

In [68]: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 [72]: 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))

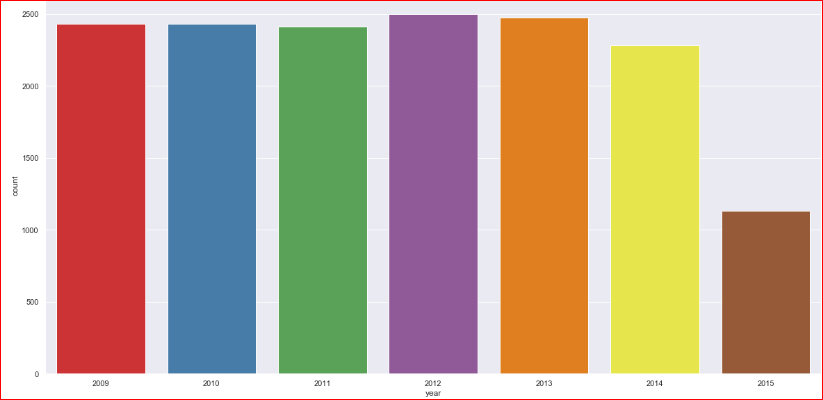
sns.countplot(train['day\_of\_week'])

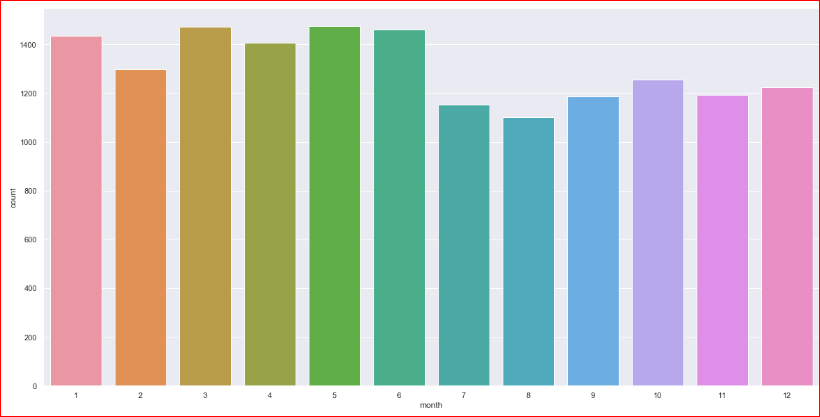
*# plt.savefig('day\_of\_week.png')*

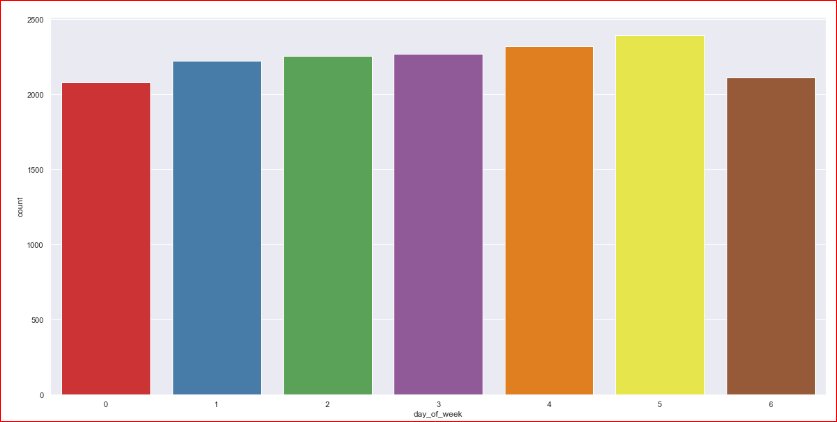
plt.figure(figsize=(20,10))

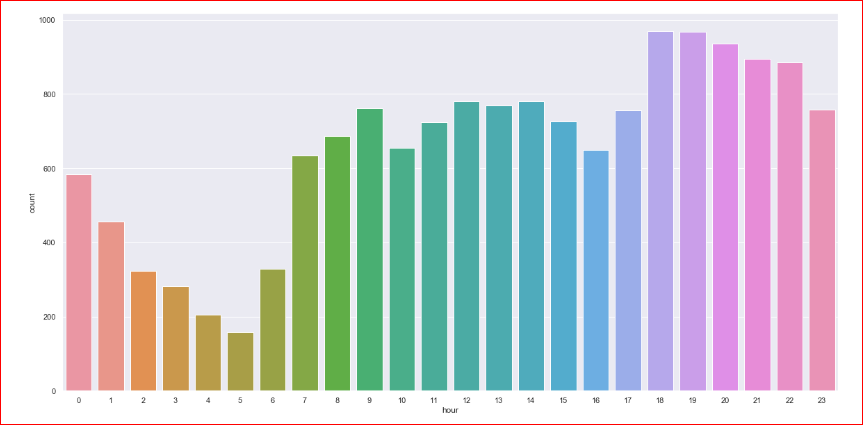
sns.countplot(train['hour'])

*# 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 [73]: **def** f(x):

*''' for sessions in a day using hour column '''*

**if** (x >=5) **and** (x <= 11):

**return** 'morning'

**else if** (x >=12) **and** (x <=16 ):

**return** 'afternoon'

**else if** (x >= 17) **and** (x <= 20):

**return** 'evening'

**else if** (x >=21) **and** (x <= 23) :

**return** 'night\_PM'

**else if** (x >=0) **and** (x <=4):

**return** 'night\_AM'

In [74]: **def** g(x):

*''' for seasons in a year using month column'''*

**if** (x >=3) **and** (x <= 5):

**return** 'spring'

**else if** (x >=6) **and** (x <=8 ):

**return** 'summer'

**else if** (x >= 9) **and** (x <= 11):

**return** 'fall'

**else if** (x >=12)|(x <= 2) :

**return** 'winter'

In [75]: **def** h(x):

*''' for week:weekday/weekend in a day\_of\_week column '''*

**if** (x >=0) **and** (x <= 4):

**return** 'weekday'

**else if** (x >=5) **and** (x <=6 ):

**return** 'weekend'

In [76]: train['session'] = train['hour'].apply(f)

test['session'] = test['hour'].apply(f)

In [77]:

train['seasons'] = train['month'].apply(g)

test['seasons'] = test['month'].apply(g)

In [78]:

train['week'] = train['day\_of\_week'].apply(h)

test['week'] = test['day\_of\_week'].apply(h)

In [79]: train.shape

Out[79]: (15660, 14)

In [80]: test. shape

Out[80]: (9914, 13)

**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 drop-off location.

In [91]:*# Calculate distance the cab travelled from pickup and dropoff location using great\_circle and geodesic from geopy library*

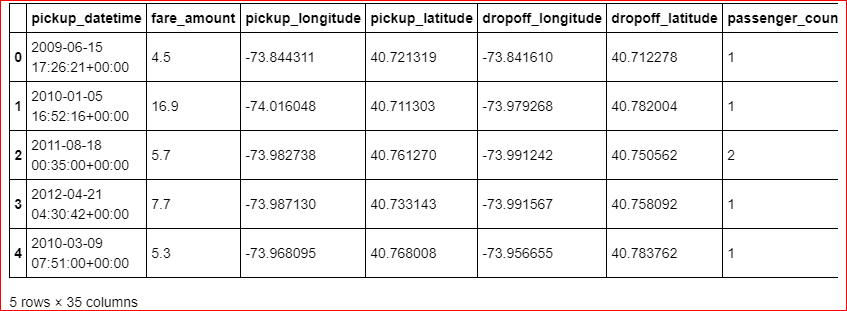
data = [train, test]

**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)

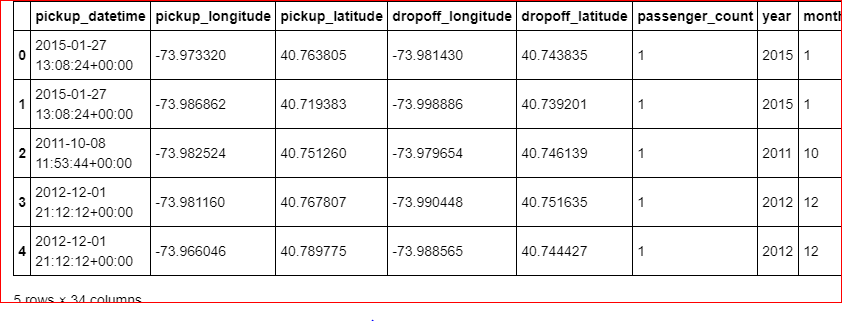
i['geodesic']=i.apply(**lambda** x: geodesic((x['pickup\_latitude'],x['pickup\_longitude']), (x['dropoff\_latitude'], x['dropoff\_longitude'])).miles, axis=1)

In [92]: train.head(5)

Out[92]: 5 rows × 35 columns

In [93]: test.head()

Out[93]:

5 rows × 34 columns

**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 for feature engineering.**

In [94]: pd.DataFrame(train.isna().sum())

In [95]: pd.DataFrame(test.isna().sum())

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

In [96]: 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 [97] train.shape, test.shape

Out[97]: ((15660, 21), (9914, 20))

In [98]: train.columns

Out[98]:

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')

In [99]: test.columns

Out[99]:

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',

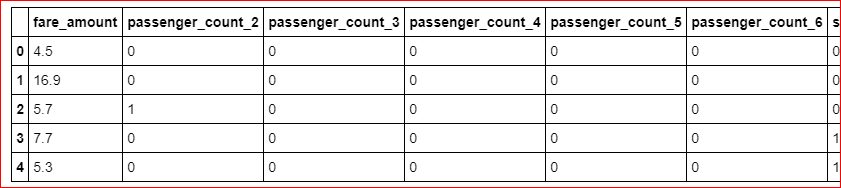
'year\_2011', 'year\_2012', 'year\_2013', 'year\_2014', 'year\_2015',

'geodesic'],

dtype='object')

In [100]: train.head()

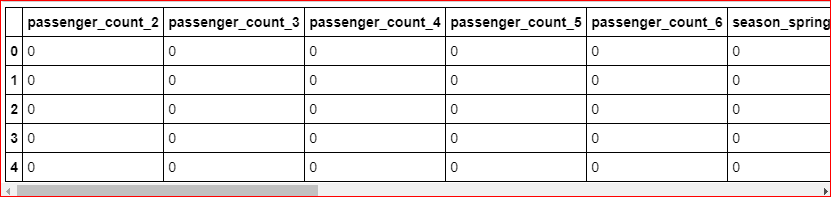
Out[100]:



5 rows × 21 columns

In [101]: test.head()

Out[101]:

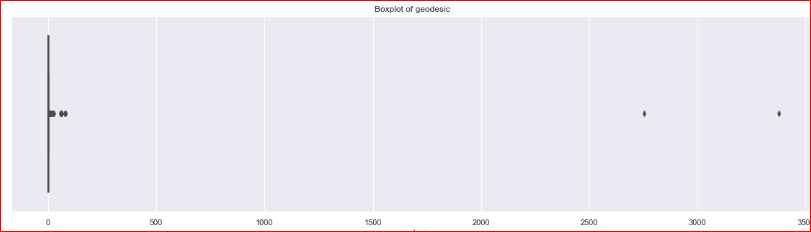
****

In [102]: plt.figure (figsize=(20,5))

sns.boxplot(x=train['geodesic'],data=train, orient='h')

plt.title ('Boxplot of geodesic ')

plt.show()



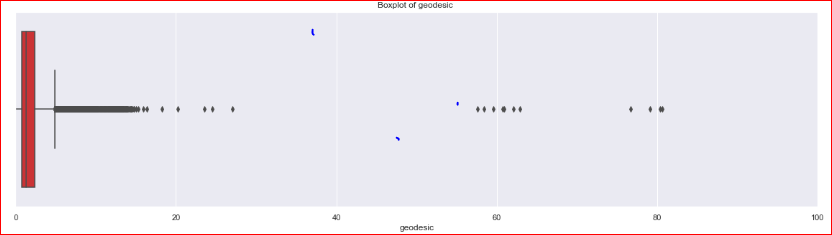
In [105]: plt.figure(figsize=(20,5))

plt.xlim(0,100)

sns.boxplot(x=train['geodesic'],data=train, orient='h')

plt.title('Boxplot of geodesic ')

plt.show()



In [107]: outlier\_treatment('geodesic')

2.425668049965582 0.7815214474966259

-1.6846984562068081 4.891887953669016

In [108]: pd.DataFrame(train.isnull().sum())

In [109]: *#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 [110]: 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.**

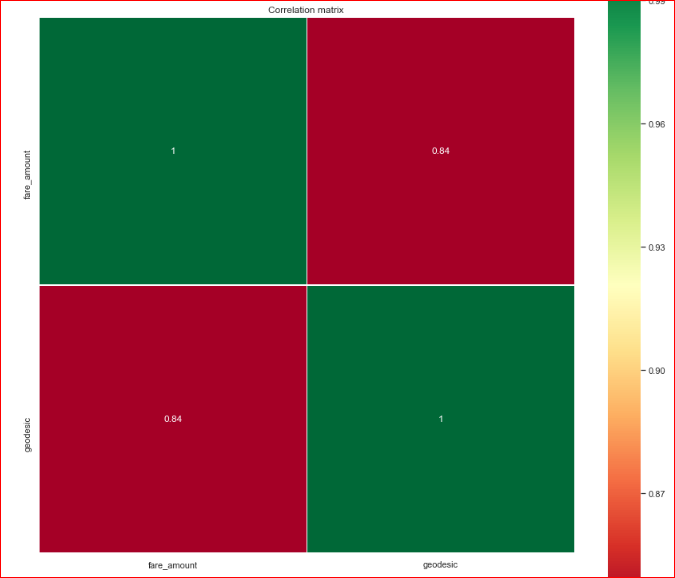
In [111]: *# heatmap using correlation matrix*

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()



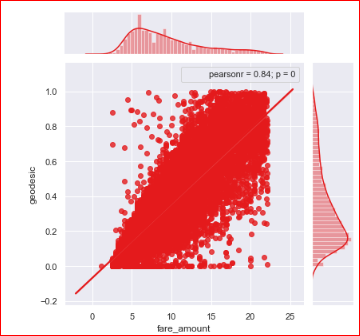
As we can see from above correlation plot fare\_amount and geodesic is correlated to each other.

* 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 correlation coefficient and p value.

In [196]: \_ = 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 dependent.
* 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 less than 0.05.
* And we will keep that variable whose p-value with other variable is more than 0.05

In [ ]: *#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 [113]: train.columns

Out[113]: Index(['fare\_amount', 'passenger\_count\_2', 'passenger\_count\_3','passenger\_count\_4', 'passenger\_count\_5', 'passenger\_count\_6',

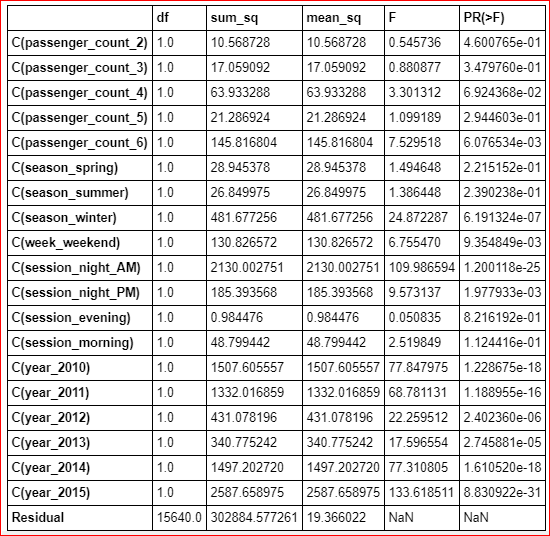
'season\_spring', 'season\_summer', 'season\_winter', 'week\_weekend', 'session\_evening', 'session\_morning', 'session\_night\_AM', 'session\_night\_PM', 'year\_2010', 'year\_2011', 'year\_2012', 'year\_2013','year\_2014', 'year\_2015', 'geodesic'], dtype='object')

In [114]:

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

Out[114]:



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 [115]:

*# \_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*

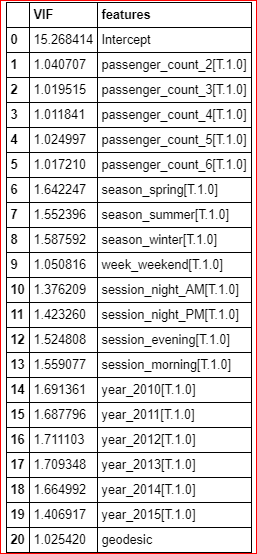
vif = pd.DataFrame()

vif["VIF"] = [variance\_inflation\_factor(predictors.values, i) **for** i **in** range(predictors. shape[1])]

vif["features"] = predictors.columns

vif

Out[115]:



We have checked for multicollinearity in our Dataset and all VIF values are below 5. So we have no or very low multicollinearity.

**Feature Scaling Check with or without normalization**

In [117]: train[num\_var].var()

Out[117]:

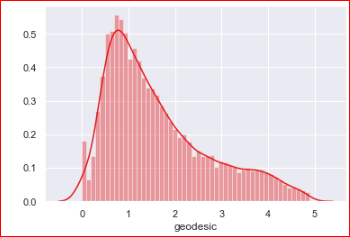
fare\_amount 20.044259

geodesic 1.232252

dtype: float64

In [118]: sns.distplot(train['geodesic'],bins=50)

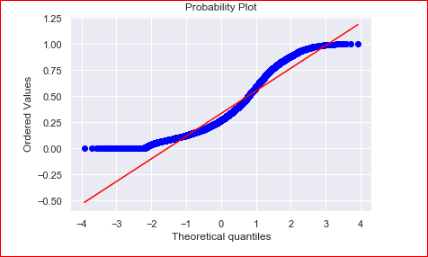
*# plt.savefig('distplot.png')*



In [123]: plt.figure()

stats.probplot(train['geodesic'], dist='norm', fit=**True**, plot=plt)

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



In [120]: *#Normalization*

train['geodesic'] = (train['geodesic'] - min(train['geodesic']))/(max(train['geodesic']) - min(train['geodesic']))

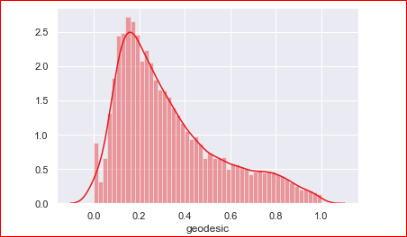
test['geodesic'] = (test['geodesic'] - min(test['geodesic']))/(max(test['geodesic']) - min(test['geodesic']))

In [121]: train['geodesic'].var()

Out[121]: 0.05155889734372662

In [122]:sns.distplot(train['geodesic'],bins=50)

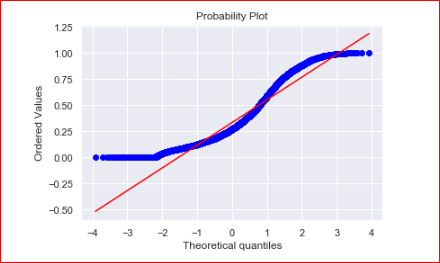
plt.savefig('distplot.png')



In [124]: plt.figure()

stats.probplot(train['geodesic'], dist='norm', fit=**True**,plot=plt)

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



In [125]: train.columns

Out[125]: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')

In [155]:*# df4=train.copy()*

train=df4.copy()

*# f4=test.copy()*

test=f4.copy()

In [161]: train=train.drop(['passenger\_count\_2'],axis=1)

test=test.drop(['passenger\_count\_2'],axis=1)

In [156]: train.columns

Out[156]: 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')

**Splitting train into train and validation dataset**

* X\_train y\_train--are train subset
* X\_test y\_test--are validation subset

In [162]:

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 = 0.25, random\_state=42)

print(train.shape, X\_train.shape, X\_test.shape,y\_train.shape,y\_test.shape)

(15660, 20) (11745, 19) (3915, 19) (11745,) (3915,)

In [163]:

**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 [191]: *# Setup the parameters and distributions to sample from: param\_dist*

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

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

*# Instantiate a Decision reg classifier: reg*

reg = LinearRegression()

*# Instantiate the gridSearchCV object: reg\_cv*

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

*# Fit it to the data*

reg\_cv.fit(X, y)

*# Print the tuned parameters and score*

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

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

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

**Best score is 0.7354470072210966**

In [164]:

*# 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()

R^2: 0.7419452241961578

Root Mean Squared Error: 2.30024202422658

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

r square 0.7343038486646876

Adjusted r square:0.7338732962659352

MAPE:18.736942495849092

MSE: 5.284481967778569

RMSE: 2.2988001147943615

RMSLE: 0.2166043577109918

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

r square 0.7419452241961578

Adjusted r square:0.7406864204117489

MAPE:18.96299602213701

MSE: 5.291113370017995

RMSE: 2.30024202422658

RMSLE: 0.2154998534679604

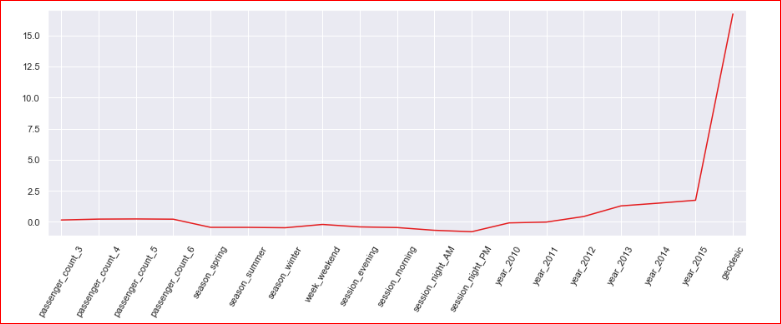
[ 1.54315806e-01 2.24592306e-01 2.37785373e-01 2.18257530e-01

-4.37104209e-01 -4.40131450e-01 -4.70078285e-01 -2.04192062e-01

-4.01234682e-01 -4.55112890e-01 -6.75183037e-01 -7.89462469e-01

-7.98897397e-02 -1.38003960e-02 4.33441511e-01 1.28797506e+00

1.50682737e+00 1.74253369e+00 1.67270577e+01]



In [193]:

**from** **sklearn.model\_selection** **import** cross\_val\_score

*# Create a linear regression object: reg*

reg = Linear Regression()

*# 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)))

[-5.30945311 -5.33924713 -5.10740884 -5.30298879 -5.42849547]

Average 5-Fold CV Score: -5.297518668356715

**Decision Tree Regression**

In [59]:

train.info()

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

Int64Index: 15660 entries, 0 to 15660

Data columns (total 7 columns):

fare\_amount 15660 non-null float64

passenger\_count 15660 non-null category

year 15660 non-null category

month 15660 non-null category

day\_of\_week 15660 non-null category

hour 15660 non-null category

geodesic 15660 non-null float64

dtypes: category(5), float64(2)

memory usage: 445.7 KB

In [169]:

*# 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 regressor: tree*

tree = DecisionTreeRegressor()

*# Instantiate the gridSearchCV object: tree\_cv*

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

*# Fit it to the data*

tree\_cv.fit(X, y)

*# Print the tuned parameters and score*

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

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

Tuned Decision Tree Parameters: {'max\_depth': 6, 'min\_samples\_split': 2}

Best score is 0.7313489270203365



In [175]:

*# 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)

[2.84444261e-04 0.00000000e+00 0.00000000e+00 6.62482083e-04

1.61062372e-04 0.00000000e+00 7.12401788e-04 5.18985478e-04

3.70483099e-04 0.00000000e+00 0.00000000e+00 1.43606742e-04

6.34699189e-04 0.00000000e+00 0.00000000e+00 1.12205718e-02

1.19156808e-02 1.10211120e-02 9.62354470e-01]



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

r square 0.7471531379429529

Adjusted r square:0.7467434074202166

MAPE:18.54709262686637

MSE: 5.028919976577308

RMSE: 2.2425253569530286

RMSLE: 0.2087118879388543

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

r square 0.7408782468914072

Adjusted r square:0.739614238339658

MAPE:19.072219163205943

MSE: 5.312990500038495

RMSE: 2.304992516265182

RMSLE: 0.2123390923855204

**Random Forest Regression**

In [171]:

*# Create the random grid*

random\_grid = {'n\_estimators': range(100,500,100),'max\_depth': range(5,20,1), 'min\_samples\_leaf':range(2,5,1),'max\_features':['auto','sqrt','log2'], 'bootstrap': [**True**, **False**],'min\_samples\_split': range(2,5,1)}

*# Instantiate a Decision Forest classifier: Forest*

Forest = RandomForestRegressor()

*# Instantiate the gridSearchCV object: Forest\_cv*

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

*# Fit it to the data*

Forest\_cv.fit(X, y)

*# Print the tuned parameters and score*

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

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

Tuned Random Forest Parameters: {'n\_estimators': 100, 'min\_samples\_split': 2, 'min\_samples\_leaf': 4, 'max\_features': 'auto', 'max\_depth': 9, 'bootstrap': True}

Best score is 0.7449373558797026

In [172]:

*# 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)

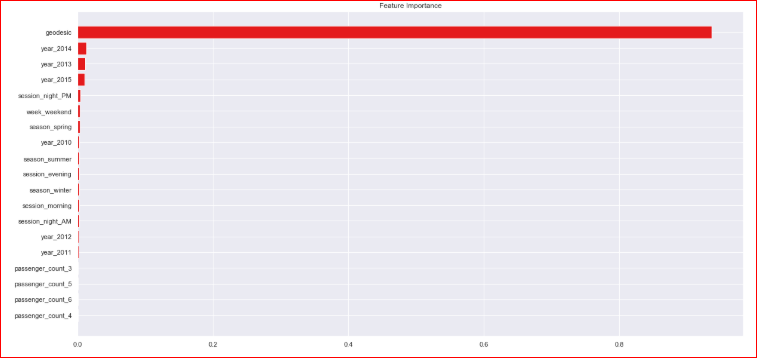
[8.38184957e-04 4.94135920e-04 7.11899115e-04 5.38216645e-04

2.80513240e-03 2.56688669e-03 2.45344190e-03 3.36652102e-03

2.55846601e-03 2.24548406e-03 2.10589565e-03 3.96233505e-03

2.60693816e-03 9.26615145e-04 1.73210920e-03 1.08242321e-02

1.24468718e-02 9.75989798e-03 9.37056736e-01]



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

r square 0.7893248033533271

Adjusted r square:0.7889834107105734

MAPE:16.95972203709288

MSE: 4.190159594493088

RMSE: 2.0469879321806195

RMSLE: 0.19142097538465247

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

r square 0.7542305881615392

Adjusted r square: 0.7530317129818394

MAPE: 18.565011130411502

MSE: 5.039216255034242

RMSE: 2.2448198714004297

RMSLE: 0.20710042575083681

In [173]:

**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)))

[-5.19821639 -5.18058997 -5.11306209 -5.15194135 -5.14644304]

Average 5-Fold CV Score: -5.158050568861664

**Improving accuracy using XGBOOST**

* Improve Accuracy a) Algorithm Tuning b) Ensembles

In [176]:

data\_dmatrix = xgb.DMatrix(data=X,label=y)

dtrain = xgb.DMatrix(X\_train, label=y\_train)

dtest = xgb.DMatrix(X\_test)

In [177]:

dtrain,dtest, data\_dmatrix

Out[177]:

(<xgboost.core.DMatrix at 0x1860091a0f0>,

<xgboost.core.DMatrix at 0x18600a11198>,

<xgboost.core.DMatrix at 0x18603856d68>)

In [179]:

params = {"objective":"reg:linear",'colsample\_bytree': 0.3,'learning\_rate': 0.1,

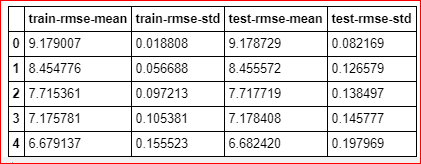
'max\_depth': 5, 'alpha': 10}

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()

Out[179]:



In [180]:

*# the final boosting round metric*

print((cv\_results["test-rmse-mean"]).tail(1))

49 2.685997

Name: test-rmse-mean, dtype: float64

In [181]:

Xgb = XGBRegressor()

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

*# pred\_xgb = model\_xgb.predict(X\_test)*

test\_scores(Xgb)

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

r square 0.7608853411883156

Adjusted r square: 0.760497863276382

MAPE: 17.83934902657697

MSE: 4.7557975392996665

RMSE: 2.1807791129088856

RMSLE: 0.2012889741509688

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

r square 0.7598553804648028

Adjusted r square:0.7586839432963385

MAPE:18.226158089842883

MSE: 4.923886423735188

RMSE: 2.2189831959109534

RMSLE: 0.20444191558083114

In [182]:

*# 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\_))

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

In [183]:

*# 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*

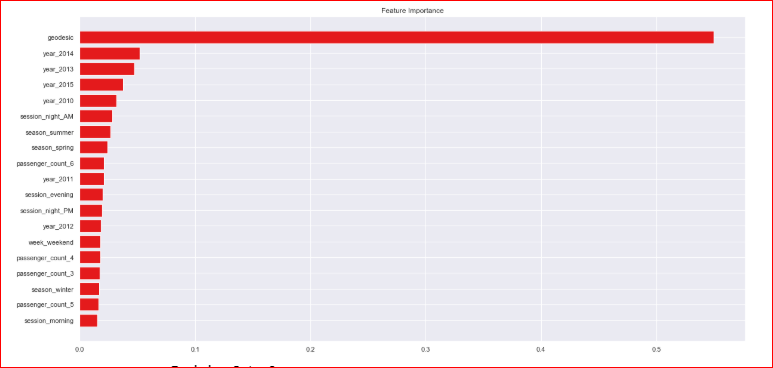
test\_scores(Xgb)

[0.01751499 0.01810841 0.01622883 0.02114999 0.02419355 0.02690452

0.0170655 0.01812746 0.01990535 0.01504988 0.02823743 0.01952868

0.03171849 0.02106621 0.01852263 0.04723873 0.05193354 0.03770861

0.5497972 ]



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

r square 0.7542254447407496

Adjusted r square:0.7538271746725257

MAPE:18.15955543560476

MSE: 4.8882575034638425

RMSE: 2.2109404115588105

RMSLE: 0.20479785952792007

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

r square 0.7587122269269724

Adjusted r square:0.7575352133997868

MAPE:18.37149036944938

MSE: 4.947325458913502

RMSE: 2.2242584065062005

RMSLE: 0.20534539640609023

**Finalize model**

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

In [184]:

**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 [185]:

test.columns

Out[185]:

Index(['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')

In [186]:

train.columns

Out[186]:

Index(['fare\_amount', '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')

In [194]:

train.shape

Out[194]:

(15660, 20)

In [195]:

test.shape

Out[195]:

(9914, 19)

In [187]:

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

In [188]:

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

In [192]:

*# 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})

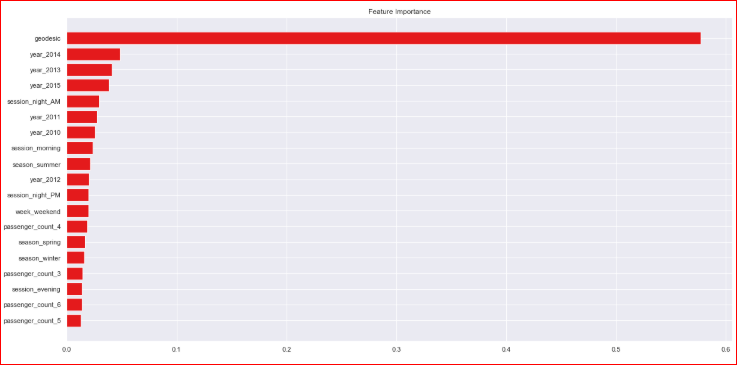
pred\_results\_wrt\_date.to\_csv("predictions\_xgboost.csv",index=**False**)

[0.01415076 0.01866597 0.01274582 0.01364598 0.01662278 0.0213572

0.01617201 0.01989506 0.01398486 0.02389306 0.02976851 0.02008079

0.02590016 0.02744447 0.02047656 0.0408919 0.04868058 0.03846469

0.57715887]



<<<------------------- 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

In [236]:

pred\_results\_wrt\_date

Out[236]:

9914 rows × 2 columns

In [193]:

*# 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')*

Out[193]:

['cab\_fare\_xgboost\_model.pkl']