**Cab Fare Prediction Solution Using Python**

**Report by AMAN ARYA**

**INDEX**

1. Loading libraries
2. Importing data
3. Graphical EDA - Data Visualization

## Missing Value Analysis

## Outlier Analysis using Boxplot

## Outlier Treatment

## Feature Engineering

## Feature Selection

## Analysis of Variance(Anova) Test

## Multicollinearity Test

## Multiple Linear Regression

## Decision Tree Regression

## Random Forest Regression

## Finalize model

**Loading libraries**

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import matplotlib.pyplot as plt

import scipy.stats as stats

from fancyimpute import KNN

import warnings

warnings.filterwarnings('ignore')

from geopy.distance import geodesic

from geopy.distance import great\_circle

from scipy.stats import chi2\_contingency

import statsmodels.api as sm

from statsmodels.formula.api import ols

from patsy import dmatrices

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

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

from sklearn.metrics import mean\_squared\_error

from sklearn import metrics

from sklearn.linear\_model import LinearRegression,Ridge,Lasso

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import RandomizedSearchCV

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

from sklearn.ensemble import RandomForestRegressor

from sklearn.tree import DecisionTreeRegressor

from xgboost import XGBRegressor

import xgboost as xgb

from sklearn.externals import joblib

os.chdir('C:/Users/DELL/Desktop/Cab Fare Prediction')

os.getcwd()

'C:\\Users\\DELL\\Desktop\\Cab Fare Prediction'

**Importing data**

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

test = pd.read\_csv('C:/Users/DELL/Desktop/data/test.csv')

data=[train,test]

for i in data:

i['pickup\_datetime'] = pd.to\_datetime(i['pickup\_datetime'],errors='coerce')

train.head(5)

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

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: 878.8 KB

test.head(5)

| **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- |
| 0 | 2015-01-27 13:08:24+00:00 | -73.973320 | 40.763805 | -73.981430 | 40.743835 | 1 |
| 1 | 2015-01-27 13:08:24+00:00 | -73.986862 | 40.719383 | -73.998886 | 40.739201 | 1 |
| 2 | 2011-10-08 11:53:44+00:00 | -73.982524 | 40.751260 | -73.979654 | 40.746139 | 1 |
| 3 | 2012-12-01 21:12:12+00:00 | -73.981160 | 40.767807 | -73.990448 | 40.751635 | 1 |
| 4 | 2012-12-01 21:12:12+00:00 | -73.966046 | 40.789775 | -73.988565 | 40.744427 | 1 |

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

test.describe()

|  | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- |
| count | 9914.000000 | 9914.000000 | 9914.000000 | 9914.000000 | 9914.000000 |
| mean | -73.974722 | 40.751041 | -73.973657 | 40.751743 | 1.671273 |
| std | 0.042774 | 0.033541 | 0.039072 | 0.035435 | 1.278747 |
| min | -74.252193 | 40.573143 | -74.263242 | 40.568973 | 1.000000 |
| 25% | -73.992501 | 40.736125 | -73.991247 | 40.735254 | 1.000000 |
| 50% | -73.982326 | 40.753051 | -73.980015 | 40.754065 | 1.000000 |
| 75% | -73.968013 | 40.767113 | -73.964059 | 40.768757 | 2.000000 |
| max | -72.986532 | 41.709555 | -72.990963 | 41.696683 | 6.000000 |

train.describe()

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

cat\_var=['passenger\_count']

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

**Graphical EDA - Data Visualization**

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

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)

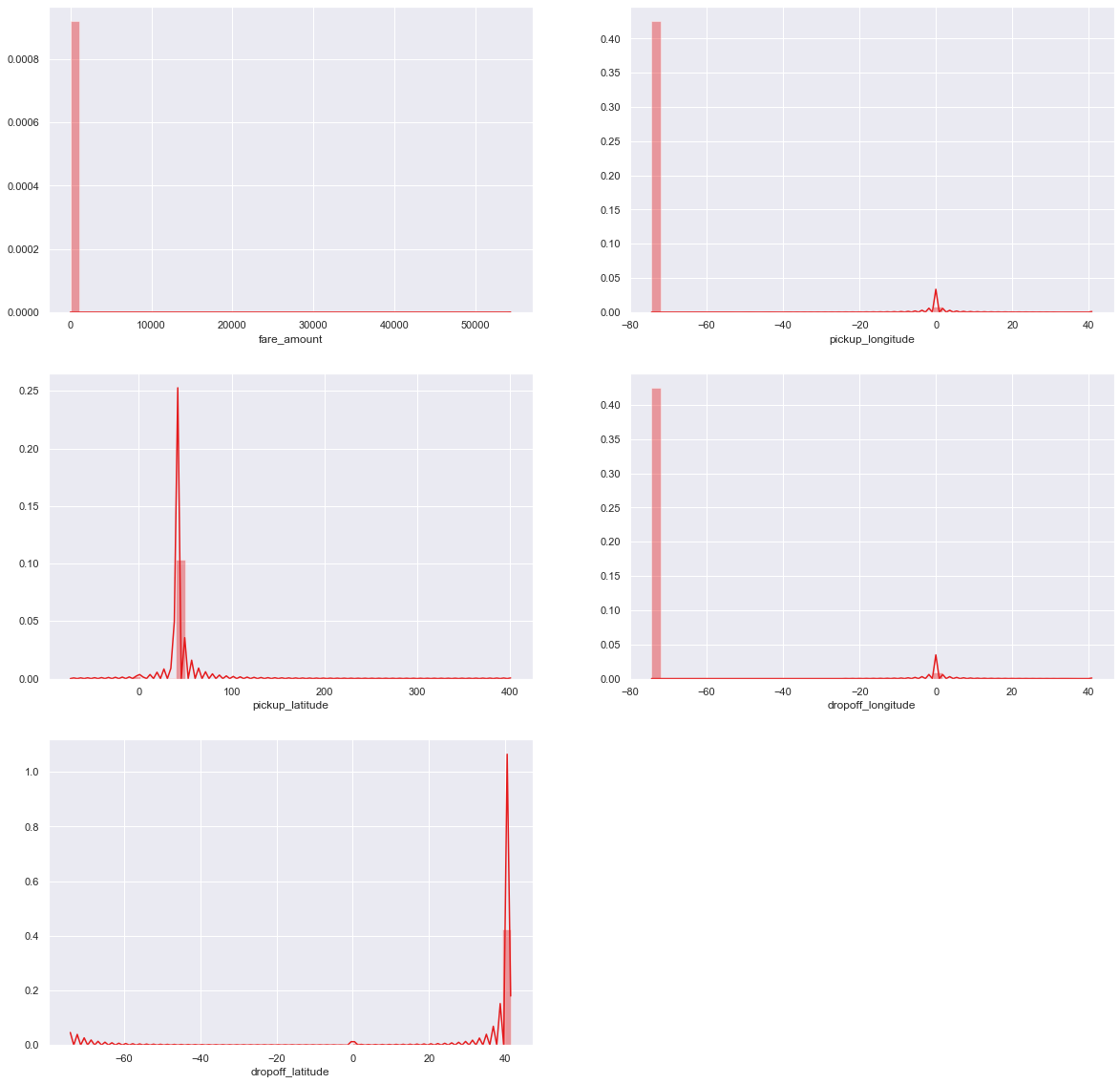
\_ = sns.distplot(train['dropoff\_longitude'],bins=50)

plt.subplot(325)

\_ = sns.distplot(train['dropoff\_latitude'],bins=50)

# plt.savefig('hist.png')

plt.show()

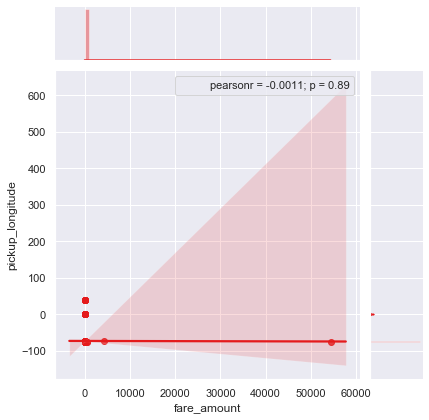


\_ = sns.jointplot(x='fare\_amount',y='pickup\_longitude',data=train,kind = 'reg')

\_.annotate(stats.pearsonr)

# plt.savefig('jointfplo.png')

plt.show()

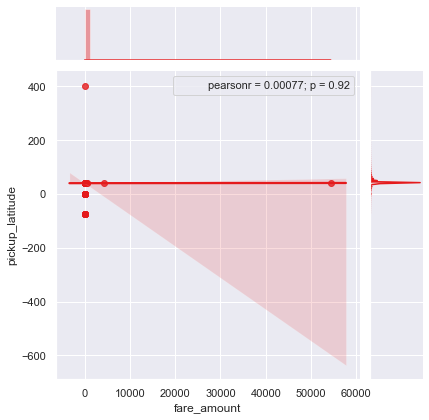


\_ = sns.jointplot(x='fare\_amount',y='pickup\_latitude',data=train,kind = 'reg')

\_.annotate(stats.pearsonr)

# plt.savefig('jointfpla.png')

plt.show()

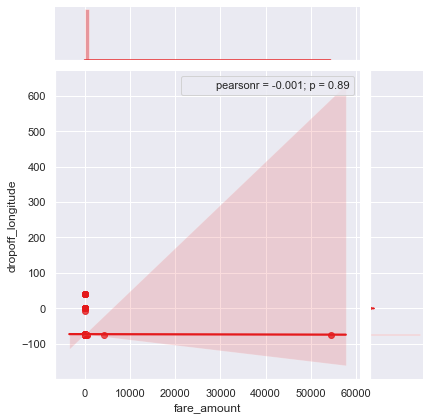


\_ = sns.jointplot(x='fare\_amount',y='dropoff\_longitude',data=train,kind = 'reg')

\_.annotate(stats.pearsonr)

# plt.savefig('jointfdlo.png')

plt.show()

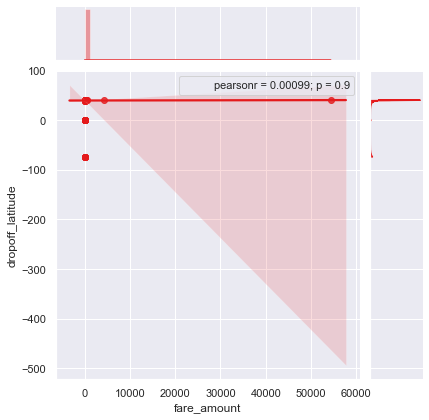


\_ = sns.jointplot(x='fare\_amount',y='dropoff\_latitude',data=train,kind = 'reg')

\_.annotate(stats.pearsonr)

# plt.savefig('jointfdla.png')

plt.show()



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

plt.subplot(321)

\_ = sns.violinplot(y='fare\_amount',data=train)

plt.subplot(322)

\_ = sns.violinplot(y='pickup\_longitude',data=train)

plt.subplot(323)

\_ = sns.violinplot(y='pickup\_latitude',data=train)

plt.subplot(324)

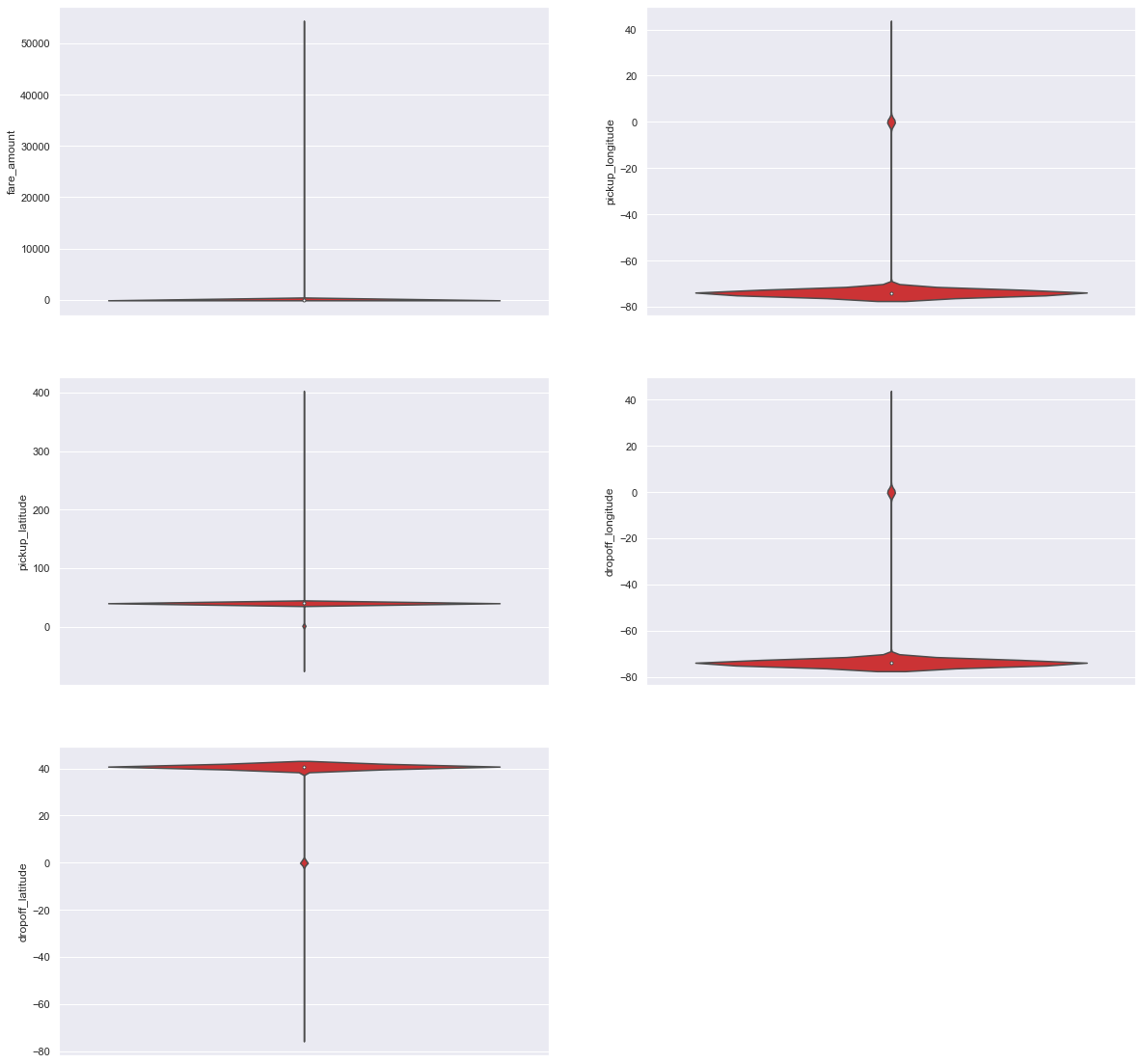
\_ = sns.violinplot(y='dropoff\_longitude',data=train)

plt.subplot(325)

\_ = sns.violinplot(y='dropoff\_latitude',data=train)

plt.savefig('violin.png')

plt.show()

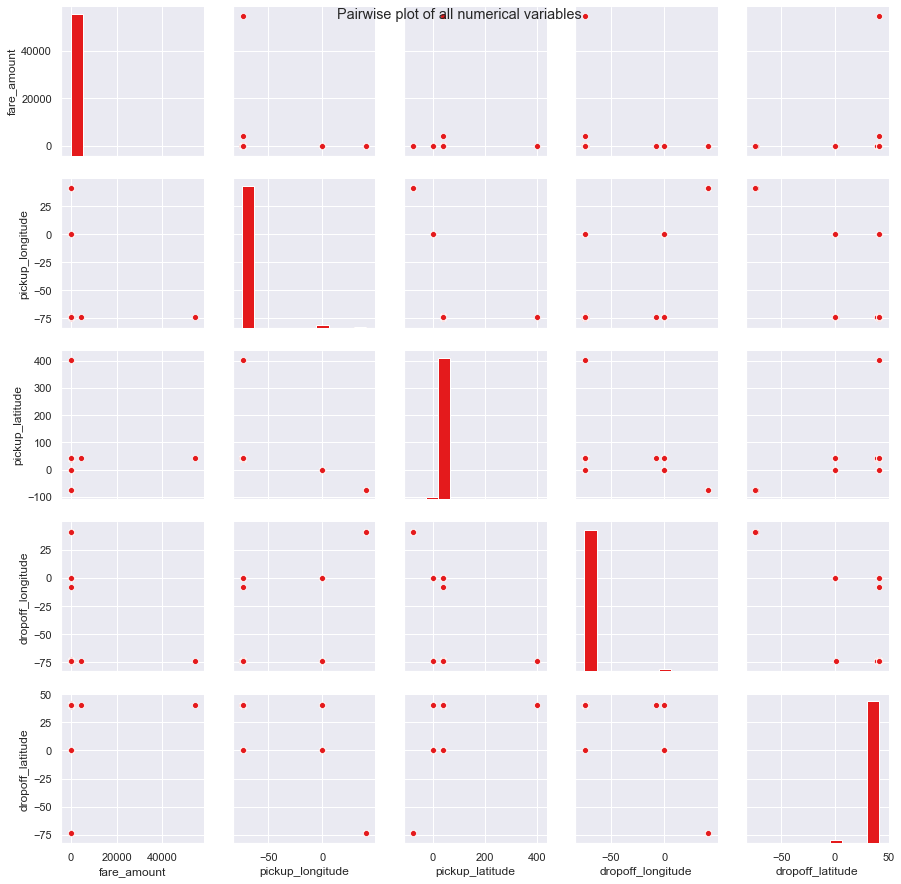


\_ =sns.pairplot(data=train[num\_var],kind='scatter',dropna=True)

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

# plt.savefig('Pairwise.png')

plt.show()



sum(train['fare\_amount']<1)

5

train[train['fare\_amount']<1]

|  | **fare\_amount** | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2039 | -2.90 | 2010-03-09 23:37:10+00:00 | -73.789450 | 40.643498 | -73.788665 | 40.641952 | 1.0 |
| 2486 | -2.50 | 2015-03-22 05:14:27+00:00 | -74.000031 | 40.720631 | -73.999809 | 40.720539 | 1.0 |
| 2780 | 0.01 | 2015-05-01 15:38:41+00:00 | -73.939041 | 40.713963 | -73.941673 | 40.713997 | 1.0 |
| 10002 | 0.00 | 2010-02-15 14:26:01+00:00 | -73.987115 | 40.738808 | -74.005911 | 40.713960 | 1.0 |
| 13032 | -3.00 | 2013-08-30 08:57:10+00:00 | -73.995062 | 40.740755 | -73.995885 | 40.741357 | 4.0 |

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

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

train[train['passenger\_count']>6]

|  | **fare\_amount** | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 233 | 8.5 | 2011-07-24 01:14:35+00:00 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 236.0 |
| 263 | 4.9 | 2010-07-12 09:44:33+00:00 | -73.983249 | 40.734655 | -73.991278 | 40.738918 | 456.0 |
| 293 | 6.1 | 2011-01-18 23:48:00+00:00 | -74.006642 | 40.738927 | -74.010828 | 40.717907 | 5334.0 |
| 356 | 8.5 | 2013-06-18 10:27:05+00:00 | -73.992108 | 40.764203 | -73.973000 | 40.762695 | 535.0 |
| 386 | 8.1 | 2009-08-21 19:35:05+00:00 | -73.960853 | 40.761557 | -73.976335 | 40.748361 | 354.0 |
| 413 | NaN | 2013-09-12 11:32:00+00:00 | -73.982060 | 40.772705 | -73.956213 | 40.771777 | 55.0 |
| 971 | 10.1 | 2010-11-21 01:41:00+00:00 | -74.004500 | 40.742143 | -73.994330 | 40.720412 | 554.0 |
| 1007 | 3.7 | 2010-12-14 14:46:00+00:00 | -73.969157 | 40.759000 | -73.968763 | 40.764617 | 53.0 |
| 1043 | 5.7 | 2012-08-22 22:08:29+00:00 | -73.973573 | 40.760184 | -73.953564 | 40.767392 | 35.0 |
| 1107 | 4.9 | 2009-08-08 21:50:50+00:00 | -73.988977 | 40.721068 | -73.982368 | 40.732064 | 345.0 |
| 1146 | 8.0 | 2014-03-27 08:05:01+00:00 | -73.991098 | 40.770655 | -73.976933 | 40.790070 | 5345.0 |
| 1200 | 9.7 | 2011-08-16 09:29:00+00:00 | -73.980487 | 40.741610 | -73.980617 | 40.746868 | 536.0 |
| 1242 | 5.3 | 2011-10-16 00:22:00+00:00 | -73.981095 | 40.738160 | -73.990587 | 40.740105 | 43.0 |
| 8406 | 6.9 | 2010-08-25 11:41:00+00:00 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 53.0 |
| 8445 | 5.7 | 2009-03-28 22:00:00+00:00 | -73.982413 | 40.751320 | -73.971292 | 40.748502 | 58.0 |
| 8506 | 11.3 | 2010-05-23 20:06:37+00:00 | -73.985424 | 40.738468 | -74.001698 | 40.707758 | 537.0 |
| 8571 | 12.5 | 2011-12-03 03:21:00+00:00 | -73.993718 | 40.762039 | -73.977527 | 40.734024 | 87.0 |
| 8631 | 20.0 | 2012-12-10 22:28:00+00:00 | -73.955445 | 40.670232 | -74.004795 | 40.731477 | 43.0 |
| 8715 | 4.5 | 2009-09-04 09:14:03+00:00 | -73.977518 | 40.758480 | -73.983252 | 40.749837 | 531.2 |
| 8985 | 8.5 | 2015-01-14 15:10:21+00:00 | -73.955444 | 40.787605 | -73.965561 | 40.798691 | 557.0 |

train[train['passenger\_count']<1]

|  | **fare\_amount** | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 314 | 34.0 | 2015-06-02 23:16:15+00:00 | -73.974899 | 40.751095 | -73.908546 | 40.881878 | 0.00 |
| 566 | 4.9 | 2012-01-28 21:33:18+00:00 | -73.955322 | 40.782840 | -73.955797 | 40.773673 | 0.00 |
| 678 | 6.5 | 2012-02-27 07:24:20+00:00 | -73.983397 | 40.738183 | -73.971395 | 40.758023 | 0.00 |
| 1160 | 13.3 | 2011-05-25 23:58:48+00:00 | -73.998360 | 40.740348 | -73.946455 | 40.777348 | 0.00 |
| 1935 | 10.1 | 2011-10-23 11:09:28+00:00 | -73.971400 | 40.795000 | -73.967900 | 40.768600 | 0.00 |
| 2200 | 8.1 | 2011-05-23 16:54:19+00:00 | -73.988008 | 40.748303 | -74.005185 | 40.738733 | 0.00 |
| 2425 | 8.9 | 2011-11-25 22:47:33+00:00 | -73.999900 | 40.738600 | -73.971800 | 40.746300 | 0.00 |
| 3034 | 5.7 | 2011-03-06 12:03:14+00:00 | -73.986557 | 40.745783 | -73.994545 | 40.729995 | 0.00 |
| 3413 | 7.3 | 2011-02-28 06:39:16+00:00 | -73.973413 | 40.743708 | -73.985220 | 40.741583 | 0.00 |
| 3481 | 11.3 | 2011-11-30 17:23:02+00:00 | -73.968100 | 40.762500 | -73.984400 | 40.760900 | 0.00 |
| 3489 | 7.3 | 2011-10-18 08:18:18+00:00 | -73.966100 | 40.794500 | -73.960300 | 40.779800 | 0.00 |
| 3599 | 8.9 | 2011-03-06 18:24:50+00:00 | -73.982657 | 40.746145 | -74.005580 | 40.724402 | 0.00 |
| 4114 | 4.5 | 2011-07-22 05:51:00+00:00 | -73.975500 | 40.760800 | -73.990200 | 40.760000 | 0.00 |
| 4248 | 4.1 | 2012-02-15 11:19:50+00:00 | -73.982112 | 40.721073 | -73.992240 | 40.725313 | 0.00 |
| 4344 | 24.9 | 2011-05-19 13:28:17+00:00 | -73.789230 | 40.646622 | -73.725032 | 40.614140 | 0.00 |
| 4354 | 5.3 | 2012-01-15 12:06:43+00:00 | -73.964400 | 40.767500 | -73.981600 | 40.774000 | 0.00 |
| 5058 | 12.9 | 2011-11-05 18:29:30+00:00 | -74.008900 | 40.709400 | -73.985400 | 40.756200 | 0.00 |
| 5150 | 7.7 | 2011-11-07 22:07:24+00:00 | -73.984900 | 40.675300 | -74.010400 | 40.655200 | 0.00 |
| 5161 | 11.3 | 2011-12-20 06:59:57+00:00 | -73.983700 | 40.775800 | -73.985300 | 40.741300 | 0.00 |
| 5277 | 6.1 | 2012-04-12 09:35:22+00:00 | -73.967003 | 40.772417 | -73.968860 | 40.761147 | 0.00 |
| 5517 | 3.3 | 2012-07-02 16:11:55+00:00 | -74.007110 | 40.743862 | -74.003337 | 40.748877 | 0.00 |
| 5557 | 27.3 | 2011-02-08 13:31:18+00:00 | -73.873318 | 40.773948 | -74.010205 | 40.711158 | 0.00 |
| 5688 | 4.9 | 2011-09-07 21:50:40+00:00 | -73.986700 | 40.761300 | -73.982300 | 40.773900 | 0.00 |
| 5914 | 8.5 | 2011-12-19 22:44:42+00:00 | -73.982000 | 40.755900 | -73.955600 | 40.769600 | 0.00 |
| 6036 | 7.7 | 2011-02-26 12:41:03+00:00 | -73.976715 | 40.775708 | -73.972785 | 40.764467 | 0.00 |
| 6575 | 4.5 | 2012-04-19 19:44:48+00:00 | -73.964598 | 40.807213 | -73.970875 | 40.797955 | 0.00 |
| 6713 | 10.5 | 2012-03-06 15:35:37+00:00 | -73.965115 | 40.766463 | -73.997523 | 40.744878 | 0.00 |
| 6881 | 5.7 | 2011-06-14 10:12:16+00:00 | -73.959000 | 40.781000 | -73.950300 | 40.775400 | 0.00 |
| 7279 | 10.9 | 2012-04-18 18:44:15+00:00 | -73.982930 | 40.722645 | -73.971108 | 40.760172 | 0.00 |
| 7520 | 6.9 | 2011-06-08 13:11:10+00:00 | -73.975100 | 40.755200 | -73.980000 | 40.765900 | 0.00 |
| 7640 | 10.9 | 2012-01-21 16:17:50+00:00 | -73.999505 | 40.725002 | -73.973262 | 40.763290 | 0.00 |
| 7909 | 6.9 | 2011-03-12 17:03:23+00:00 | -73.974793 | 40.759865 | -73.955718 | 40.772732 | 0.00 |
| 8321 | 7.3 | 2012-02-25 19:19:41+00:00 | -74.003725 | 40.742108 | -73.988270 | 40.759308 | 0.00 |
| 8661 | 8.1 | 2012-01-13 13:47:57+00:00 | -73.999700 | 40.721800 | -74.005600 | 40.741100 | 0.00 |
| 8862 | 11.5 | 2013-05-17 07:15:00+00:00 | -73.969835 | 40.753017 | -73.998580 | 40.712927 | 0.12 |
| 8916 | 2.5 | 2011-09-13 18:45:31+00:00 | -73.783300 | 40.648600 | -73.783300 | 40.648600 | 0.00 |
| 8971 | 7.7 | 2011-10-22 20:50:34+00:00 | -73.998600 | 40.761000 | -73.977900 | 40.777200 | 0.00 |
| 9159 | 3.3 | 2011-06-04 10:38:47+00:00 | -73.987300 | 40.729200 | -73.994000 | 40.732100 | 0.00 |
| 9965 | 9.3 | 2012-01-04 22:04:14+00:00 | -73.971000 | 40.754900 | -74.004500 | 40.736100 | 0.00 |
| 10642 | 10.1 | 2012-03-10 18:21:16+00:00 | -73.972272 | 40.754037 | -73.994090 | 40.734440 | 0.00 |
| 10663 | 18.1 | 2011-05-06 13:38:48+00:00 | -73.973818 | 40.789362 | -73.942252 | 40.754120 | 0.00 |
| 10711 | 6.5 | 2012-03-31 20:16:01+00:00 | -74.004323 | 40.724073 | -74.006453 | 40.738420 | 0.00 |
| 11462 | 15.7 | 2011-10-20 23:09:45+00:00 | -73.994800 | 40.750400 | -73.959200 | 40.710600 | 0.00 |
| 11803 | 12.1 | 2011-06-21 08:58:36+00:00 | -73.999000 | 40.724800 | -73.980600 | 40.754900 | 0.00 |
| 12216 | 5.7 | 2012-03-27 16:06:09+00:00 | -73.982017 | 40.756338 | -73.980987 | 40.745442 | 0.00 |
| 12611 | 8.1 | 2011-07-22 23:13:58+00:00 | -73.978000 | 40.752300 | -73.991800 | 40.763900 | 0.00 |
| 13029 | 4.5 | 2012-02-01 21:18:24+00:00 | -73.982117 | 40.770408 | -73.980057 | 40.762155 | 0.00 |
| 13227 | 10.9 | 2011-04-22 18:13:12+00:00 | -73.999512 | 40.722078 | -73.989495 | 40.747422 | 0.00 |
| 13379 | 7.3 | 2012-03-14 07:20:10+00:00 | -73.964512 | 40.771577 | -73.945595 | 40.778032 | 0.00 |
| 13714 | 4.1 | 2011-12-15 07:05:33+00:00 | -73.979400 | 40.731100 | -73.981500 | 40.724800 | 0.00 |
| 13742 | 21.7 | 2011-04-29 12:51:14+00:00 | -74.009472 | 40.702140 | -73.959185 | 40.783245 | 0.00 |
| 14196 | 8.1 | 2011-10-06 23:31:04+00:00 | -73.999400 | 40.743800 | -73.990400 | 40.724800 | 0.00 |
| 14308 | 8.1 | 2011-11-11 12:03:38+00:00 | 0.000000 | 0.000000 | -73.990000 | 40.755400 | 0.00 |
| 14872 | 3.3 | 2011-03-02 19:25:46+00:00 | -73.948553 | 40.773972 | -73.945098 | 40.778413 | 0.00 |
| 15286 | 4.9 | 2012-03-20 22:23:34+00:00 | -73.971808 | 40.760143 | -73.962027 | 40.767677 | 0.00 |
| 15514 | 6.5 | 2012-02-12 02:03:50+00:00 | -73.999342 | 40.718872 | -73.984112 | 40.725242 | 0.00 |
| 15554 | 6.5 | 2011-08-03 08:31:19+00:00 | -73.991700 | 40.750100 | -73.981000 | 40.750900 | 0.00 |
| 15919 | 16.5 | 2011-10-17 08:58:54+00:00 | 0.000000 | 0.000000 | -73.999700 | 40.734500 | 0.00 |

len(train[train['passenger\_count']<1])

58

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

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

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

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

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

0

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

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=311

pickup\_latitude equal to 0=311

dropoff\_longitude equal to 0=312

dropoff\_latitude equal to 0=310

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)

train.shape

(15661, 7)

df=train.copy()

## Missing Value Analysis

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

#Reset index

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

missing\_val

|  | **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 |

#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

|  | **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 |

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

1.0

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

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

nan

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

1.0

# Choosing a random values to replace it as NA

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

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

# Replacing 1.0 with NA

train['fare\_amount'].loc[1000] = np.nan

print('Value after replacing with nan:{}'.format(train['fare\_amount'].loc[1000]))

# Impute with mean

print('Value if imputed with mean:{}'.format(train['fare\_amount'].fillna(train['fare\_amount'].mean()).loc[1000]))

# Impute with median

print('Value if imputed with median:{}'.format(train['fare\_amount'].fillna(train['fare\_amount'].median()).loc[1000]))

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

train.std()

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

columns=['fare\_amount', 'pickup\_longitude', 'pickup\_latitude','dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count']

pickup\_datetime=pd.DataFrame(train['pickup\_datetime'])

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

Imputing row 1/15661 with 0 missing, elapsed time: 61.372

Imputing row 101/15661 with 0 missing, elapsed time: 61.610

Imputing row 201/15661 with 0 missing, elapsed time: 61.616

Imputing row 301/15661 with 0 missing, elapsed time: 61.619

Imputing row 401/15661 with 0 missing, elapsed time: 61.623

Imputing row 501/15661 with 0 missing, elapsed time: 61.629

Imputing row 601/15661 with 0 missing, elapsed time: 61.636

Imputing row 701/15661 with 0 missing, elapsed time: 61.642

Imputing row 801/15661 with 0 missing, elapsed time: 61.646

Imputing row 901/15661 with 0 missing, elapsed time: 61.650

Imputing row 1001/15661 with 0 missing, elapsed time: 61.652

Imputing row 1101/15661 with 0 missing, elapsed time: 61.654

Imputing row 1201/15661 with 0 missing, elapsed time: 61.655

Imputing row 1301/15661 with 0 missing, elapsed time: 61.656

Imputing row 1401/15661 with 0 missing, elapsed time: 61.658

Imputing row 1501/15661 with 0 missing, elapsed time: 61.659

Imputing row 1601/15661 with 0 missing, elapsed time: 61.661

Imputing row 1701/15661 with 0 missing, elapsed time: 61.663

Imputing row 1801/15661 with 0 missing, elapsed time: 61.664

Imputing row 1901/15661 with 0 missing, elapsed time: 61.665

Imputing row 2001/15661 with 0 missing, elapsed time: 61.666

Imputing row 2101/15661 with 0 missing, elapsed time: 61.668

Imputing row 2201/15661 with 0 missing, elapsed time: 61.669

Imputing row 2301/15661 with 0 missing, elapsed time: 61.671

Imputing row 2401/15661 with 0 missing, elapsed time: 61.672

Imputing row 2501/15661 with 0 missing, elapsed time: 61.673

Imputing row 2601/15661 with 0 missing, elapsed time: 61.674

Imputing row 2701/15661 with 0 missing, elapsed time: 61.674

Imputing row 2801/15661 with 0 missing, elapsed time: 61.675

Imputing row 2901/15661 with 0 missing, elapsed time: 61.675

Imputing row 3001/15661 with 0 missing, elapsed time: 61.677

Imputing row 3101/15661 with 0 missing, elapsed time: 61.677

Imputing row 3201/15661 with 0 missing, elapsed time: 61.678

Imputing row 3301/15661 with 0 missing, elapsed time: 61.679

Imputing row 3401/15661 with 0 missing, elapsed time: 61.681

Imputing row 3501/15661 with 0 missing, elapsed time: 61.682

Imputing row 3601/15661 with 0 missing, elapsed time: 61.682

Imputing row 3701/15661 with 0 missing, elapsed time: 61.683

Imputing row 3801/15661 with 0 missing, elapsed time: 61.683

Imputing row 3901/15661 with 0 missing, elapsed time: 61.684

Imputing row 4001/15661 with 0 missing, elapsed time: 61.687

Imputing row 4101/15661 with 0 missing, elapsed time: 61.687

Imputing row 4201/15661 with 0 missing, elapsed time: 61.688

Imputing row 4301/15661 with 0 missing, elapsed time: 61.688

Imputing row 4401/15661 with 0 missing, elapsed time: 61.689

Imputing row 4501/15661 with 0 missing, elapsed time: 61.690

Imputing row 4601/15661 with 0 missing, elapsed time: 61.693

Imputing row 4701/15661 with 0 missing, elapsed time: 61.695

Imputing row 4801/15661 with 0 missing, elapsed time: 61.696

Imputing row 4901/15661 with 0 missing, elapsed time: 61.697

Imputing row 5001/15661 with 0 missing, elapsed time: 61.697

Imputing row 5101/15661 with 0 missing, elapsed time: 61.698

Imputing row 5201/15661 with 0 missing, elapsed time: 61.698

Imputing row 5301/15661 with 0 missing, elapsed time: 61.698

Imputing row 5401/15661 with 0 missing, elapsed time: 61.699

Imputing row 5501/15661 with 0 missing, elapsed time: 61.699

Imputing row 5601/15661 with 0 missing, elapsed time: 61.700

Imputing row 5701/15661 with 0 missing, elapsed time: 61.700

Imputing row 5801/15661 with 0 missing, elapsed time: 61.701

Imputing row 5901/15661 with 0 missing, elapsed time: 61.701

Imputing row 6001/15661 with 0 missing, elapsed time: 61.702

Imputing row 6101/15661 with 0 missing, elapsed time: 61.702

Imputing row 6201/15661 with 0 missing, elapsed time: 61.703

Imputing row 6301/15661 with 0 missing, elapsed time: 61.703

Imputing row 6401/15661 with 0 missing, elapsed time: 61.704

Imputing row 6501/15661 with 0 missing, elapsed time: 61.704

Imputing row 6601/15661 with 0 missing, elapsed time: 61.705

Imputing row 6701/15661 with 0 missing, elapsed time: 61.705

Imputing row 6801/15661 with 0 missing, elapsed time: 61.706

Imputing row 6901/15661 with 0 missing, elapsed time: 61.706

Imputing row 7001/15661 with 0 missing, elapsed time: 61.708

Imputing row 7101/15661 with 0 missing, elapsed time: 61.708

Imputing row 7201/15661 with 0 missing, elapsed time: 61.709

Imputing row 7301/15661 with 0 missing, elapsed time: 61.709

Imputing row 7401/15661 with 0 missing, elapsed time: 61.709

Imputing row 7501/15661 with 0 missing, elapsed time: 61.710

Imputing row 7601/15661 with 0 missing, elapsed time: 61.711

Imputing row 7701/15661 with 0 missing, elapsed time: 61.715

Imputing row 7801/15661 with 0 missing, elapsed time: 61.716

Imputing row 7901/15661 with 0 missing, elapsed time: 61.717

Imputing row 8001/15661 with 0 missing, elapsed time: 61.718

Imputing row 8101/15661 with 0 missing, elapsed time: 61.719

Imputing row 8201/15661 with 0 missing, elapsed time: 61.719

Imputing row 8301/15661 with 0 missing, elapsed time: 61.720

Imputing row 8401/15661 with 0 missing, elapsed time: 61.720

Imputing row 8501/15661 with 0 missing, elapsed time: 61.721

Imputing row 8601/15661 with 0 missing, elapsed time: 61.721

Imputing row 8701/15661 with 0 missing, elapsed time: 61.722

Imputing row 8801/15661 with 0 missing, elapsed time: 61.724

Imputing row 8901/15661 with 0 missing, elapsed time: 61.724

Imputing row 9001/15661 with 0 missing, elapsed time: 61.724

Imputing row 9101/15661 with 0 missing, elapsed time: 61.725

Imputing row 9201/15661 with 0 missing, elapsed time: 61.725

Imputing row 9301/15661 with 0 missing, elapsed time: 61.726

Imputing row 9401/15661 with 0 missing, elapsed time: 61.726

Imputing row 9501/15661 with 0 missing, elapsed time: 61.727

Imputing row 9601/15661 with 0 missing, elapsed time: 61.727

Imputing row 9701/15661 with 0 missing, elapsed time: 61.728

Imputing row 9801/15661 with 0 missing, elapsed time: 61.728

Imputing row 9901/15661 with 0 missing, elapsed time: 61.728

Imputing row 10001/15661 with 0 missing, elapsed time: 61.729

Imputing row 10101/15661 with 0 missing, elapsed time: 61.729

Imputing row 10201/15661 with 0 missing, elapsed time: 61.730

Imputing row 10301/15661 with 0 missing, elapsed time: 61.730

Imputing row 10401/15661 with 0 missing, elapsed time: 61.731

Imputing row 10501/15661 with 0 missing, elapsed time: 61.731

Imputing row 10601/15661 with 0 missing, elapsed time: 61.732

Imputing row 10701/15661 with 0 missing, elapsed time: 61.732

Imputing row 10801/15661 with 0 missing, elapsed time: 61.733

Imputing row 10901/15661 with 0 missing, elapsed time: 61.733

Imputing row 11001/15661 with 0 missing, elapsed time: 61.733

Imputing row 11101/15661 with 0 missing, elapsed time: 61.734

Imputing row 11201/15661 with 0 missing, elapsed time: 61.734

Imputing row 11301/15661 with 0 missing, elapsed time: 61.735

Imputing row 11401/15661 with 0 missing, elapsed time: 61.735

Imputing row 11501/15661 with 0 missing, elapsed time: 61.736

Imputing row 11601/15661 with 0 missing, elapsed time: 61.736

Imputing row 11701/15661 with 0 missing, elapsed time: 61.737

Imputing row 11801/15661 with 0 missing, elapsed time: 61.737

Imputing row 11901/15661 with 0 missing, elapsed time: 61.738

Imputing row 12001/15661 with 0 missing, elapsed time: 61.738

Imputing row 12101/15661 with 0 missing, elapsed time: 61.739

Imputing row 12201/15661 with 0 missing, elapsed time: 61.739

Imputing row 12301/15661 with 0 missing, elapsed time: 61.740

Imputing row 12401/15661 with 0 missing, elapsed time: 61.740

Imputing row 12501/15661 with 0 missing, elapsed time: 61.741

Imputing row 12601/15661 with 0 missing, elapsed time: 61.741

Imputing row 12701/15661 with 0 missing, elapsed time: 61.742

Imputing row 12801/15661 with 0 missing, elapsed time: 61.742

Imputing row 12901/15661 with 0 missing, elapsed time: 61.743

Imputing row 13001/15661 with 0 missing, elapsed time: 61.743

Imputing row 13101/15661 with 0 missing, elapsed time: 61.743

Imputing row 13201/15661 with 0 missing, elapsed time: 61.744

Imputing row 13301/15661 with 0 missing, elapsed time: 61.745

Imputing row 13401/15661 with 0 missing, elapsed time: 61.745

Imputing row 13501/15661 with 0 missing, elapsed time: 61.745

Imputing row 13601/15661 with 0 missing, elapsed time: 61.746

Imputing row 13701/15661 with 0 missing, elapsed time: 61.746

Imputing row 13801/15661 with 0 missing, elapsed time: 61.747

Imputing row 13901/15661 with 0 missing, elapsed time: 61.747

Imputing row 14001/15661 with 0 missing, elapsed time: 61.748

Imputing row 14101/15661 with 0 missing, elapsed time: 61.748

Imputing row 14201/15661 with 0 missing, elapsed time: 61.749

Imputing row 14301/15661 with 0 missing, elapsed time: 61.750

Imputing row 14401/15661 with 0 missing, elapsed time: 61.751

Imputing row 14501/15661 with 0 missing, elapsed time: 61.752

Imputing row 14601/15661 with 0 missing, elapsed time: 61.752

Imputing row 14701/15661 with 0 missing, elapsed time: 61.753

Imputing row 14801/15661 with 0 missing, elapsed time: 61.754

Imputing row 14901/15661 with 0 missing, elapsed time: 61.754

Imputing row 15001/15661 with 0 missing, elapsed time: 61.755

Imputing row 15101/15661 with 0 missing, elapsed time: 61.755

Imputing row 15201/15661 with 0 missing, elapsed time: 61.756

Imputing row 15301/15661 with 0 missing, elapsed time: 61.756

Imputing row 15401/15661 with 0 missing, elapsed time: 61.756

Imputing row 15501/15661 with 0 missing, elapsed time: 61.757

Imputing row 15601/15661 with 0 missing, elapsed time: 61.757

train.std()

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

train.loc[1000]

fare\_amount 7.369801

pickup\_longitude -73.995420

pickup\_latitude 40.759662

dropoff\_longitude -73.987607

dropoff\_latitude 40.751247

passenger\_count 2.544158

Name: 1000, dtype: float64

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

0 1.0

1 1.0

2 2.0

3 1.0

4 1.0

Name: passenger\_count, dtype: float64

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

train.std()

fare\_amount 435.661995

pickup\_longitude 2.659050

pickup\_latitude 2.613305

dropoff\_longitude 2.710835

dropoff\_latitude 2.632400

passenger\_count 1.264322

dtype: float64

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

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

train.loc[1000]

fare\_amount 7.369801

pickup\_longitude -73.995420

pickup\_latitude 40.759662

dropoff\_longitude -73.987607

dropoff\_latitude 40.751247

passenger\_count 2.000000

Name: 1000, dtype: float64

pickup\_datetime.head()

|  | **pickup\_datetime** |
| --- | --- |
| 0 | 2009-06-15 17:26:21+00:00 |
| 1 | 2010-01-05 16:52:16+00:00 |
| 2 | 2011-08-18 00:35:00+00:00 |
| 3 | 2012-04-21 04:30:42+00:00 |
| 4 | 2010-03-09 07:51:00+00:00 |

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

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

missing\_val

|  | **index** | **0** |
| --- | --- | --- |
| 0 | pickup\_datetime | 1 |

pickup\_datetime.shape

(15661, 1)

train.shape

(15661, 6)

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

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

train.describe()

|  | **fare\_amount** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- | --- |
| count | 15661.000000 | 15661.000000 | 15661.000000 | 15661.000000 | 15661.000000 | 15661.000000 |
| mean | 15.110669 | -73.911514 | 40.689712 | -73.906315 | 40.687655 | 1.649192 |
| std | 435.661995 | 2.659050 | 2.613305 | 2.710835 | 2.632400 | 1.264322 |
| min | 1.140000 | -74.438233 | -74.006893 | -74.429332 | -74.006377 | 1.000000 |
| 25% | 6.000000 | -73.992390 | 40.736530 | -73.991369 | 40.736293 | 1.000000 |
| 50% | 8.500000 | -73.982049 | 40.753300 | -73.980555 | 40.754242 | 1.000000 |
| 75% | 12.500000 | -73.968080 | 40.767805 | -73.965360 | 40.768312 | 2.000000 |
| max | 54343.000000 | 40.766125 | 41.366138 | 40.802437 | 41.366138 | 6.000000 |

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

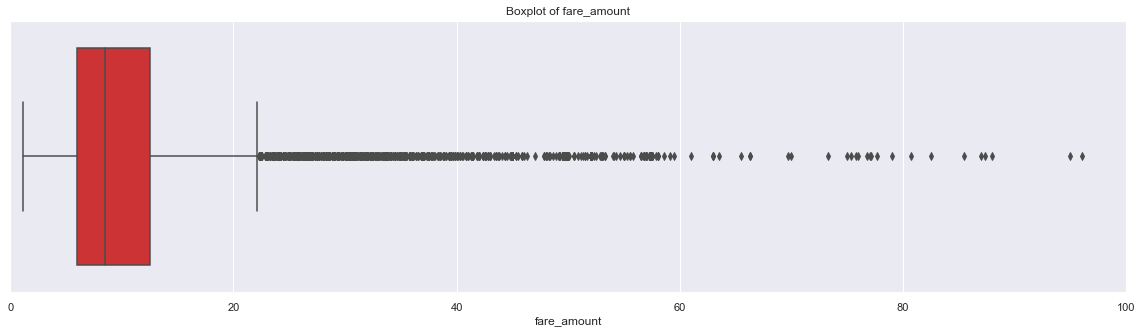
plt.xlim(0,100)

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()



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

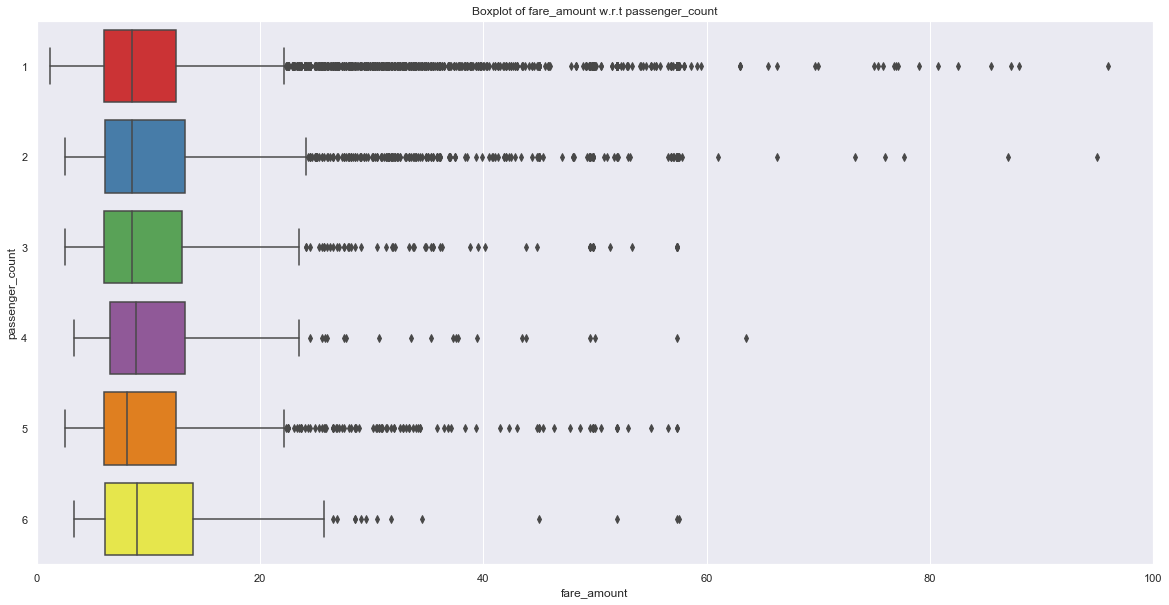
plt.xlim(0,100)

\_ = 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()



train.describe()

|  | **fare\_amount** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- | --- |
| count | 15661.000000 | 15661.000000 | 15661.000000 | 15661.000000 | 15661.000000 | 15661.000000 |
| mean | 15.110669 | -73.911514 | 40.689712 | -73.906315 | 40.687655 | 1.649192 |
| std | 435.661995 | 2.659050 | 2.613305 | 2.710835 | 2.632400 | 1.264322 |
| min | 1.140000 | -74.438233 | -74.006893 | -74.429332 | -74.006377 | 1.000000 |
| 25% | 6.000000 | -73.992390 | 40.736530 | -73.991369 | 40.736293 | 1.000000 |
| 50% | 8.500000 | -73.982049 | 40.753300 | -73.980555 | 40.754242 | 1.000000 |
| 75% | 12.500000 | -73.968080 | 40.767805 | -73.965360 | 40.768312 | 2.000000 |
| max | 54343.000000 | 40.766125 | 41.366138 | 40.802437 | 41.366138 | 6.000000 |

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

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

## Outlier Treatment

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

outlier\_treatment('fare\_amount')

12.5 6.0

-3.75 22.25

pd.DataFrame(train.isnull().sum())

|  | **0** |
| --- | --- |
| fare\_amount | 1359 |
| pickup\_longitude | 0 |
| pickup\_latitude | 0 |
| dropoff\_longitude | 0 |
| dropoff\_latitude | 0 |
| passenger\_count | 0 |

train.std()

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

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

Imputing row 1/15661 with 0 missing, elapsed time: 80.478

Imputing row 101/15661 with 0 missing, elapsed time: 80.481

Imputing row 201/15661 with 0 missing, elapsed time: 80.485

Imputing row 301/15661 with 1 missing, elapsed time: 80.486

Imputing row 401/15661 with 0 missing, elapsed time: 80.489

Imputing row 501/15661 with 0 missing, elapsed time: 80.492

Imputing row 601/15661 with 1 missing, elapsed time: 80.495

Imputing row 701/15661 with 0 missing, elapsed time: 80.497

Imputing row 801/15661 with 0 missing, elapsed time: 80.499

Imputing row 901/15661 with 0 missing, elapsed time: 80.502

Imputing row 1001/15661 with 0 missing, elapsed time: 80.504

Imputing row 1101/15661 with 1 missing, elapsed time: 80.506

Imputing row 1201/15661 with 0 missing, elapsed time: 80.508

Imputing row 1301/15661 with 0 missing, elapsed time: 80.510

Imputing row 1401/15661 with 0 missing, elapsed time: 80.512

Imputing row 1501/15661 with 0 missing, elapsed time: 80.515

Imputing row 1601/15661 with 0 missing, elapsed time: 80.518

Imputing row 1701/15661 with 0 missing, elapsed time: 80.520

Imputing row 1801/15661 with 0 missing, elapsed time: 80.523

Imputing row 1901/15661 with 0 missing, elapsed time: 80.525

Imputing row 2001/15661 with 0 missing, elapsed time: 80.528

Imputing row 2101/15661 with 0 missing, elapsed time: 80.530

Imputing row 2201/15661 with 0 missing, elapsed time: 80.533

Imputing row 2301/15661 with 0 missing, elapsed time: 80.537

Imputing row 2401/15661 with 0 missing, elapsed time: 80.540

Imputing row 2501/15661 with 0 missing, elapsed time: 80.542

Imputing row 2601/15661 with 0 missing, elapsed time: 80.546

Imputing row 2701/15661 with 0 missing, elapsed time: 80.549

Imputing row 2801/15661 with 0 missing, elapsed time: 80.551

Imputing row 2901/15661 with 0 missing, elapsed time: 80.553

Imputing row 3001/15661 with 0 missing, elapsed time: 80.556

Imputing row 3101/15661 with 0 missing, elapsed time: 80.559

Imputing row 3201/15661 with 0 missing, elapsed time: 80.561

Imputing row 3301/15661 with 0 missing, elapsed time: 80.565

Imputing row 3401/15661 with 0 missing, elapsed time: 80.568

Imputing row 3501/15661 with 0 missing, elapsed time: 80.571

Imputing row 3601/15661 with 0 missing, elapsed time: 80.574

Imputing row 3701/15661 with 0 missing, elapsed time: 80.577

Imputing row 3801/15661 with 0 missing, elapsed time: 80.580

Imputing row 3901/15661 with 0 missing, elapsed time: 80.582

Imputing row 4001/15661 with 0 missing, elapsed time: 80.584

Imputing row 4101/15661 with 0 missing, elapsed time: 80.586

Imputing row 4201/15661 with 0 missing, elapsed time: 80.589

Imputing row 4301/15661 with 0 missing, elapsed time: 80.591

Imputing row 4401/15661 with 0 missing, elapsed time: 80.594

Imputing row 4501/15661 with 0 missing, elapsed time: 80.596

Imputing row 4601/15661 with 0 missing, elapsed time: 80.599

Imputing row 4701/15661 with 0 missing, elapsed time: 80.601

Imputing row 4801/15661 with 0 missing, elapsed time: 80.603

Imputing row 4901/15661 with 0 missing, elapsed time: 80.606

Imputing row 5001/15661 with 0 missing, elapsed time: 80.608

Imputing row 5101/15661 with 1 missing, elapsed time: 80.610

Imputing row 5201/15661 with 0 missing, elapsed time: 80.612

Imputing row 5301/15661 with 0 missing, elapsed time: 80.614

Imputing row 5401/15661 with 0 missing, elapsed time: 80.616

Imputing row 5501/15661 with 0 missing, elapsed time: 80.619

Imputing row 5601/15661 with 0 missing, elapsed time: 80.621

Imputing row 5701/15661 with 0 missing, elapsed time: 80.623

Imputing row 5801/15661 with 0 missing, elapsed time: 80.625

Imputing row 5901/15661 with 0 missing, elapsed time: 80.626

Imputing row 6001/15661 with 0 missing, elapsed time: 80.629

Imputing row 6101/15661 with 0 missing, elapsed time: 80.631

Imputing row 6201/15661 with 0 missing, elapsed time: 80.634

Imputing row 6301/15661 with 0 missing, elapsed time: 80.635

Imputing row 6401/15661 with 0 missing, elapsed time: 80.637

Imputing row 6501/15661 with 0 missing, elapsed time: 80.639

Imputing row 6601/15661 with 0 missing, elapsed time: 80.641

Imputing row 6701/15661 with 0 missing, elapsed time: 80.643

Imputing row 6801/15661 with 0 missing, elapsed time: 80.647

Imputing row 6901/15661 with 0 missing, elapsed time: 80.649

Imputing row 7001/15661 with 0 missing, elapsed time: 80.652

Imputing row 7101/15661 with 0 missing, elapsed time: 80.653

Imputing row 7201/15661 with 0 missing, elapsed time: 80.655

Imputing row 7301/15661 with 0 missing, elapsed time: 80.658

Imputing row 7401/15661 with 0 missing, elapsed time: 80.662

Imputing row 7501/15661 with 0 missing, elapsed time: 80.665

Imputing row 7601/15661 with 0 missing, elapsed time: 80.670

Imputing row 7701/15661 with 0 missing, elapsed time: 80.672

Imputing row 7801/15661 with 0 missing, elapsed time: 80.677

Imputing row 7901/15661 with 1 missing, elapsed time: 80.680

Imputing row 8001/15661 with 0 missing, elapsed time: 80.685

Imputing row 8101/15661 with 0 missing, elapsed time: 80.687

Imputing row 8201/15661 with 0 missing, elapsed time: 80.690

Imputing row 8301/15661 with 0 missing, elapsed time: 80.694

Imputing row 8401/15661 with 0 missing, elapsed time: 80.697

Imputing row 8501/15661 with 0 missing, elapsed time: 80.700

Imputing row 8601/15661 with 0 missing, elapsed time: 80.702

Imputing row 8701/15661 with 0 missing, elapsed time: 80.705

Imputing row 8801/15661 with 0 missing, elapsed time: 80.709

Imputing row 8901/15661 with 0 missing, elapsed time: 80.711

Imputing row 9001/15661 with 0 missing, elapsed time: 80.713

Imputing row 9101/15661 with 0 missing, elapsed time: 80.715

Imputing row 9201/15661 with 0 missing, elapsed time: 80.717

Imputing row 9301/15661 with 0 missing, elapsed time: 80.718

Imputing row 9401/15661 with 0 missing, elapsed time: 80.720

Imputing row 9501/15661 with 0 missing, elapsed time: 80.723

Imputing row 9601/15661 with 0 missing, elapsed time: 80.726

Imputing row 9701/15661 with 0 missing, elapsed time: 80.731

Imputing row 9801/15661 with 0 missing, elapsed time: 80.734

Imputing row 9901/15661 with 0 missing, elapsed time: 80.737

Imputing row 10001/15661 with 0 missing, elapsed time: 80.740

Imputing row 10101/15661 with 0 missing, elapsed time: 80.744

Imputing row 10201/15661 with 0 missing, elapsed time: 80.746

Imputing row 10301/15661 with 0 missing, elapsed time: 80.748

Imputing row 10401/15661 with 0 missing, elapsed time: 80.751

Imputing row 10501/15661 with 0 missing, elapsed time: 80.753

Imputing row 10601/15661 with 0 missing, elapsed time: 80.754

Imputing row 10701/15661 with 0 missing, elapsed time: 80.757

Imputing row 10801/15661 with 0 missing, elapsed time: 80.759

Imputing row 10901/15661 with 0 missing, elapsed time: 80.761

Imputing row 11001/15661 with 0 missing, elapsed time: 80.762

Imputing row 11101/15661 with 0 missing, elapsed time: 80.765

Imputing row 11201/15661 with 0 missing, elapsed time: 80.767

Imputing row 11301/15661 with 0 missing, elapsed time: 80.770

Imputing row 11401/15661 with 0 missing, elapsed time: 80.772

Imputing row 11501/15661 with 0 missing, elapsed time: 80.774

Imputing row 11601/15661 with 0 missing, elapsed time: 80.779

Imputing row 11701/15661 with 1 missing, elapsed time: 80.781

Imputing row 11801/15661 with 0 missing, elapsed time: 80.786

Imputing row 11901/15661 with 0 missing, elapsed time: 80.792

Imputing row 12001/15661 with 0 missing, elapsed time: 80.794

Imputing row 12101/15661 with 0 missing, elapsed time: 80.797

Imputing row 12201/15661 with 0 missing, elapsed time: 80.801

Imputing row 12301/15661 with 1 missing, elapsed time: 80.803

Imputing row 12401/15661 with 0 missing, elapsed time: 80.808

Imputing row 12501/15661 with 0 missing, elapsed time: 80.811

Imputing row 12601/15661 with 1 missing, elapsed time: 80.813

Imputing row 12701/15661 with 0 missing, elapsed time: 80.815

Imputing row 12801/15661 with 0 missing, elapsed time: 80.818

Imputing row 12901/15661 with 0 missing, elapsed time: 80.820

Imputing row 13001/15661 with 0 missing, elapsed time: 80.823

Imputing row 13101/15661 with 0 missing, elapsed time: 80.826

Imputing row 13201/15661 with 0 missing, elapsed time: 80.828

Imputing row 13301/15661 with 0 missing, elapsed time: 80.830

Imputing row 13401/15661 with 0 missing, elapsed time: 80.833

Imputing row 13501/15661 with 0 missing, elapsed time: 80.835

Imputing row 13601/15661 with 1 missing, elapsed time: 80.837

Imputing row 13701/15661 with 0 missing, elapsed time: 80.839

Imputing row 13801/15661 with 0 missing, elapsed time: 80.842

Imputing row 13901/15661 with 0 missing, elapsed time: 80.843

Imputing row 14001/15661 with 0 missing, elapsed time: 80.846

Imputing row 14101/15661 with 0 missing, elapsed time: 80.849

Imputing row 14201/15661 with 0 missing, elapsed time: 80.852

Imputing row 14301/15661 with 0 missing, elapsed time: 80.855

Imputing row 14401/15661 with 0 missing, elapsed time: 80.857

Imputing row 14501/15661 with 1 missing, elapsed time: 80.859

Imputing row 14601/15661 with 0 missing, elapsed time: 80.863

Imputing row 14701/15661 with 0 missing, elapsed time: 80.865

Imputing row 14801/15661 with 0 missing, elapsed time: 80.868

Imputing row 14901/15661 with 0 missing, elapsed time: 80.872

Imputing row 15001/15661 with 0 missing, elapsed time: 80.877

Imputing row 15101/15661 with 0 missing, elapsed time: 80.882

Imputing row 15201/15661 with 0 missing, elapsed time: 80.886

Imputing row 15301/15661 with 0 missing, elapsed time: 80.890

Imputing row 15401/15661 with 0 missing, elapsed time: 80.895

Imputing row 15501/15661 with 0 missing, elapsed time: 80.898

Imputing row 15601/15661 with 0 missing, elapsed time: 80.901

train.std()

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

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

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

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

train.describe()

|  | **fare\_amount** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** |
| --- | --- | --- | --- | --- | --- |
| count | 15661.000000 | 15661.000000 | 15661.000000 | 15661.000000 | 15661.000000 |
| mean | 9.404603 | -73.911514 | 40.689712 | -73.906315 | 40.687655 |
| std | 4.476970 | 2.659050 | 2.613305 | 2.710835 | 2.632400 |
| min | 1.140000 | -74.438233 | -74.006893 | -74.429332 | -74.006377 |
| 25% | 6.000000 | -73.992390 | 40.736530 | -73.991369 | 40.736293 |
| 50% | 8.200000 | -73.982049 | 40.753300 | -73.980555 | 40.754242 |
| 75% | 11.800000 | -73.968080 | 40.767805 | -73.965360 | 40.768312 |
| max | 22.100000 | 40.766125 | 41.366138 | 40.802437 | 41.366138 |

train.head()

|  | **fare\_amount** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 4.5 | -73.844311 | 40.721319 | -73.841610 | 40.712278 | 1 |
| 1 | 16.9 | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1 |
| 2 | 5.7 | -73.982738 | 40.761270 | -73.991242 | 40.750562 | 2 |
| 3 | 7.7 | -73.987130 | 40.733143 | -73.991567 | 40.758092 | 1 |
| 4 | 5.3 | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1 |

df2 = train.copy()

train.shape

(15661, 6)

## Feature Engineering

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

train.head()

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

train.shape

(15661, 7)

train=train.reset\_index(drop=True)

pd.DataFrame(train.isna().sum())

|  | **0** |
| --- | --- |
| pickup\_datetime | 1 |
| fare\_amount | 0 |
| pickup\_longitude | 0 |
| pickup\_latitude | 0 |
| dropoff\_longitude | 0 |
| dropoff\_latitude | 0 |
| passenger\_count | 0 |

train=train.dropna()

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)

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))

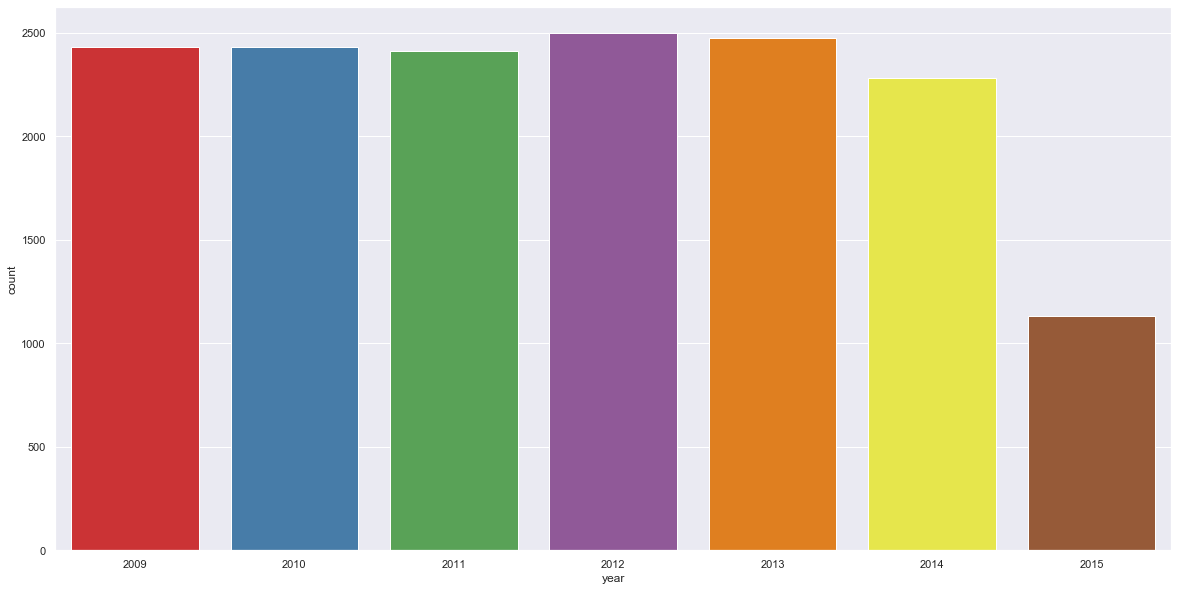
sns.countplot(train['day\_of\_week'])

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

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

sns.countplot(train['hour'])

# plt.savefig('hour.png')



def f(x):

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

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

return 'morning'

elif (x >=12) and (x <=16 ):

return 'afternoon'

elif (x >= 17) and (x <= 20):

return'evening'

elif (x >=21) and (x <= 23) :

return 'night\_PM'

elif (x >=0) and (x <=4):

return'night\_AM'

def g(x):

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

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

return 'spring'

elif (x >=6) and (x <=8 ):

return 'summer'

elif (x >= 9) and (x <= 11):

return'fall'

elif (x >=12)|(x <= 2) :

return 'winter'

def h(x):

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

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

return 'weekday'

elif (x >=5) and (x <=6 ):

return 'weekend'

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

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

# train\_nodummies['session'] = train\_nodummies['hour'].apply(f)

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

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

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

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

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

train.shape

(15660, 13)

test.shape

(9914, 12)

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

count 15660

unique 6

top 1

freq 11055

Name: passenger\_count, dtype: int64

#Creating dummies for each variable in passenger\_count and merging dummies dataframe to both train and test dataframe

temp = pd.get\_dummies(train['passenger\_count'], prefix = 'passenger\_count')

train = train.join(temp)

temp = pd.get\_dummies(test['passenger\_count'], prefix = 'passenger\_count')

test = test.join(temp)

temp = pd.get\_dummies(train['seasons'], prefix = 'season')

train = train.join(temp)

temp = pd.get\_dummies(test['seasons'], prefix = 'season')

test = test.join(temp)

temp = pd.get\_dummies(train['week'], prefix = 'week')

train = train.join(temp)

temp = pd.get\_dummies(test['week'], prefix = 'week')

test = test.join(temp)

temp = pd.get\_dummies(train['session'], prefix = 'session')

train = train.join(temp)

temp = pd.get\_dummies(test['session'], prefix = 'session')

test = test.join(temp)

temp = pd.get\_dummies(train['year'], prefix = 'year')

train = train.join(temp)

temp = pd.get\_dummies(test['year'], prefix = 'year')

test = test.join(temp)

train.head()

| **pickup\_datetime** | **fare\_amount** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **year** | **month** | **day\_of\_week** | **...** | **passenger\_count\_1** | **passenger\_count\_2** | **passenger\_count\_3** | **passenger\_count\_4** | **passenger\_count\_5** | **passenger\_count\_6** | **season\_fall** | **season\_spring** | **season\_summer** | **season\_winter** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 2009-06-15 17:26:21+00:00 | 4.5 | -73.844311 | 40.721319 | -73.841610 | 40.712278 | 1 | 2009 | 6 | 0 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 2010-01-05 16:52:16+00:00 | 16.9 | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1 | 2010 | 1 | 1 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 2011-08-18 00:35:00+00:00 | 5.7 | -73.982738 | 40.761270 | -73.991242 | 40.750562 | 2 | 2011 | 8 | 3 | ... | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 2012-04-21 04:30:42+00:00 | 7.7 | -73.987130 | 40.733143 | -73.991567 | 40.758092 | 1 | 2012 | 4 | 5 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 4 | 2010-03-09 07:51:00+00:00 | 5.3 | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1 | 2010 | 3 | 1 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

5 rows × 23 columns

test.head()

| **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **year** | **month** | **day\_of\_week** | **hour** | **...** | **passenger\_count\_1** | **passenger\_count\_2** | **passenger\_count\_3** | **passenger\_count\_4** | **passenger\_count\_5** | **passenger\_count\_6** | **season\_fall** | **season\_spring** | **season\_summer** | **season\_winter** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 2015-01-27 13:08:24+00:00 | -73.973320 | 40.763805 | -73.981430 | 40.743835 | 1 | 2015 | 1 | 1 | 13 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 2015-01-27 13:08:24+00:00 | -73.986862 | 40.719383 | -73.998886 | 40.739201 | 1 | 2015 | 1 | 1 | 13 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 2011-10-08 11:53:44+00:00 | -73.982524 | 40.751260 | -73.979654 | 40.746139 | 1 | 2011 | 10 | 5 | 11 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 3 | 2012-12-01 21:12:12+00:00 | -73.981160 | 40.767807 | -73.990448 | 40.751635 | 1 | 2012 | 12 | 5 | 21 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 4 | 2012-12-01 21:12:12+00:00 | -73.966046 | 40.789775 | -73.988565 | 40.744427 | 1 | 2012 | 12 | 5 | 21 | ... | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

5 rows × 22 columns

train.columns

Index(['pickup\_datetime', 'fare\_amount', 'pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count', 'year',

'month', 'day\_of\_week', 'hour', 'session', 'seasons',

'passenger\_count\_1', 'passenger\_count\_2', 'passenger\_count\_3',

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

'season\_fall', 'season\_spring', 'season\_summer', 'season\_winter'],

dtype='object')

train=train.drop(['passenger\_count\_1','season\_fall','week\_weekday','session\_afternoon','year\_2009'],axis=1)

test=test.drop(['passenger\_count\_1','season\_fall','week\_weekday','session\_afternoon','year\_2009'],axis=1)

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)

train.head()

|  | **pickup\_datetime** | **fare\_amount** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **year** | **month** | **day\_of\_week** | **...** | **passenger\_count\_3** | **passenger\_count\_4** | **passenger\_count\_5** | **passenger\_count\_6** | **season\_fall** | **season\_spring** | **season\_summer** | **season\_winter** | **great\_circle** | **geodesic** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 2009-06-15 17:26:21+00:00 | 4.5 | -73.844311 | 40.721319 | -73.841610 | 40.712278 | 1 | 2009 | 6 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0.640488 | 0.639764 |
| 1 | 2010-01-05 16:52:16+00:00 | 16.9 | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1 | 2010 | 1 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 5.250677 | 5.246511 |
| 2 | 2011-08-18 00:35:00+00:00 | 5.7 | -73.982738 | 40.761270 | -73.991242 | 40.750562 | 2 | 2011 | 8 | 3 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0.863412 | 0.863167 |
| 3 | 2012-04-21 04:30:42+00:00 | 7.7 | -73.987130 | 40.733143 | -73.991567 | 40.758092 | 1 | 2012 | 4 | 5 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1.739388 | 1.737223 |
| 4 | 2010-03-09 07:51:00+00:00 | 5.3 | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1 | 2010 | 3 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1.242220 | 1.241710 |

5 rows × 25 columns

test.head()

| **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **year** | **month** | **day\_of\_week** | **hour** | **...** | **passenger\_count\_3** | **passenger\_count\_4** | **passenger\_count\_5** | **passenger\_count\_6** | **season\_fall** | **season\_spring** | **season\_summer** | **season\_winter** | **great\_circle** | **geodesic** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 2015-01-27 13:08:24+00:00 | -73.973320 | 40.763805 | -73.981430 | 40.743835 | 1 | 2015 | 1 | 1 | 13 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1.443609 | 1.442197 |
| 1 | 2015-01-27 13:08:24+00:00 | -73.986862 | 40.719383 | -73.998886 | 40.739201 | 1 | 2015 | 1 | 1 | 13 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1.507047 | 1.506080 |
| 2 | 2011-10-08 11:53:44+00:00 | -73.982524 | 40.751260 | -73.979654 | 40.746139 | 1 | 2011 | 10 | 5 | 11 | ... | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0.384398 | 0.384120 |
| 3 | 2012-12-01 21:12:12+00:00 | -73.981160 | 40.767807 | -73.990448 | 40.751635 | 1 | 2012 | 12 | 5 | 21 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1.218531 | 1.217683 |
| 4 | 2012-12-01 21:12:12+00:00 | -73.966046 | 40.789775 | -73.988565 | 40.744427 | 1 | 2012 | 12 | 5 | 21 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 3.347519 | 3.344737 |

5 rows × 24 columns

pd.DataFrame(train.isna().sum())

|  | **0** |
| --- | --- |
| pickup\_datetime | 0 |
| fare\_amount | 0 |
| pickup\_longitude | 0 |
| pickup\_latitude | 0 |
| dropoff\_longitude | 0 |
| dropoff\_latitude | 0 |
| passenger\_count | 0 |
| year | 0 |
| month | 0 |
| day\_of\_week | 0 |
| hour | 0 |
| session | 0 |
| seasons | 0 |
| passenger\_count\_1 | 0 |
| passenger\_count\_2 | 0 |
| passenger\_count\_3 | 0 |
| passenger\_count\_4 | 0 |
| passenger\_count\_5 | 0 |
| passenger\_count\_6 | 0 |
| season\_fall | 0 |
| season\_spring | 0 |
| season\_summer | 0 |
| season\_winter | 0 |
| great\_circle | 0 |
| geodesic | 0 |

pd.DataFrame(test.isna().sum())

|  | **0** |
| --- | --- |
| pickup\_datetime | 0 |
| pickup\_longitude | 0 |
| pickup\_latitude | 0 |
| dropoff\_longitude | 0 |
| dropoff\_latitude | 0 |
| passenger\_count | 0 |
| year | 0 |
| month | 0 |
| day\_of\_week | 0 |
| hour | 0 |
| session | 0 |
| seasons | 0 |
| passenger\_count\_1 | 0 |
| passenger\_count\_2 | 0 |
| passenger\_count\_3 | 0 |
| passenger\_count\_4 | 0 |
| passenger\_count\_5 | 0 |
| passenger\_count\_6 | 0 |
| season\_fall | 0 |
| season\_spring | 0 |
| season\_summer | 0 |
| season\_winter | 0 |
| great\_circle | 0 |
| geodesic | 0 |

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)

train.shape,test.shape

((15660, 25), (9914, 24))

train.columns

Index(['pickup\_datetime', 'fare\_amount', 'pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count', 'year',

'month', 'day\_of\_week', 'hour', 'session', 'seasons',

'passenger\_count\_1', 'passenger\_count\_2', 'passenger\_count\_3',

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

'season\_fall', 'season\_spring', 'season\_summer', 'season\_winter',

'great\_circle', 'geodesic'],

dtype='object')

test.columns

Index(['pickup\_datetime', 'pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count', 'year',

'month', 'day\_of\_week', 'hour', 'session', 'seasons',

'passenger\_count\_1', 'passenger\_count\_2', 'passenger\_count\_3',

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

'season\_fall', 'season\_spring', 'season\_summer', 'season\_winter',

'great\_circle', 'geodesic'],

dtype='object')

train.head()

|  | **pickup\_datetime** | **fare\_amount** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **year** | **month** | **day\_of\_week** | **...** | **passenger\_count\_3** | **passenger\_count\_4** | **passenger\_count\_5** | **passenger\_count\_6** | **season\_fall** | **season\_spring** | **season\_summer** | **season\_winter** | **great\_circle** | **geodesic** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 2009-06-15 17:26:21+00:00 | 4.5 | -73.844311 | 40.721319 | -73.841610 | 40.712278 | 1 | 2009 | 6 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0.640488 | 0.639764 |
| 1 | 2010-01-05 16:52:16+00:00 | 16.9 | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1 | 2010 | 1 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 5.250677 | 5.246511 |
| 2 | 2011-08-18 00:35:00+00:00 | 5.7 | -73.982738 | 40.761270 | -73.991242 | 40.750562 | 2 | 2011 | 8 | 3 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0.863412 | 0.863167 |
| 3 | 2012-04-21 04:30:42+00:00 | 7.7 | -73.987130 | 40.733143 | -73.991567 | 40.758092 | 1 | 2012 | 4 | 5 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1.739388 | 1.737223 |
| 4 | 2010-03-09 07:51:00+00:00 | 5.3 | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1 | 2010 | 3 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1.242220 | 1.241710 |

5 rows × 25 columns

test.head()

|  | **pickup\_datetime** | **pickup\_longitude** | **pickup\_latitude** | **dropoff\_longitude** | **dropoff\_latitude** | **passenger\_count** | **year** | **month** | **day\_of\_week** | **hour** | **...** | **passenger\_count\_3** | **passenger\_count\_4** | **passenger\_count\_5** | **passenger\_count\_6** | **season\_fall** | **season\_spring** | **season\_summer** | **season\_winter** | **great\_circle** | **geodesic** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 2015-01-27 13:08:24+00:00 | -73.973320 | 40.763805 | -73.981430 | 40.743835 | 1 | 2015 | 1 | 1 | 13 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1.443609 | 1.442197 |
| 1 | 2015-01-27 13:08:24+00:00 | -73.986862 | 40.719383 | -73.998886 | 40.739201 | 1 | 2015 | 1 | 1 | 13 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1.507047 | 1.506080 |
| 2 | 2011-10-08 11:53:44+00:00 | -73.982524 | 40.751260 | -73.979654 | 40.746139 | 1 | 2011 | 10 | 5 | 11 | ... | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0.384398 | 0.384120 |
| 3 | 2012-12-01 21:12:12+00:00 | -73.981160 | 40.767807 | -73.990448 | 40.751635 | 1 | 2012 | 12 | 5 | 21 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1.218531 | 1.217683 |
| 4 | 2012-12-01 21:12:12+00:00 | -73.966046 | 40.789775 | -73.988565 | 40.744427 | 1 | 2012 | 12 | 5 | 21 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 3.347519 | 3.344737 |

5 rows × 24 columns

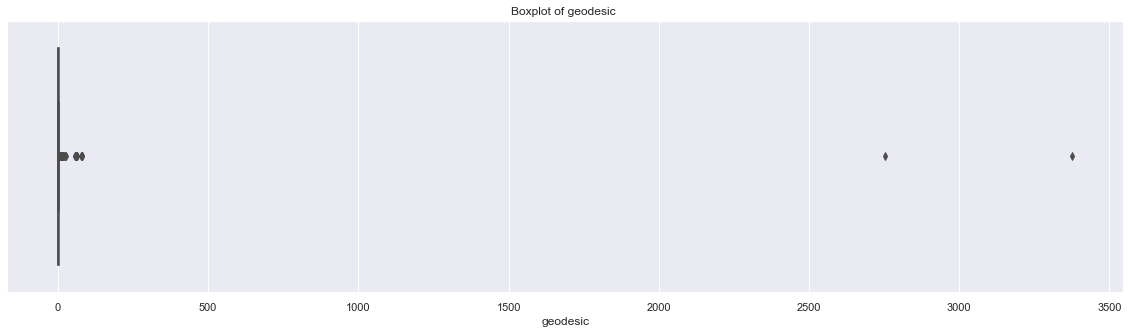
plt.figure(figsize=(20,5))

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

plt.title('Boxplot of geodesic ')

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

plt.show()



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

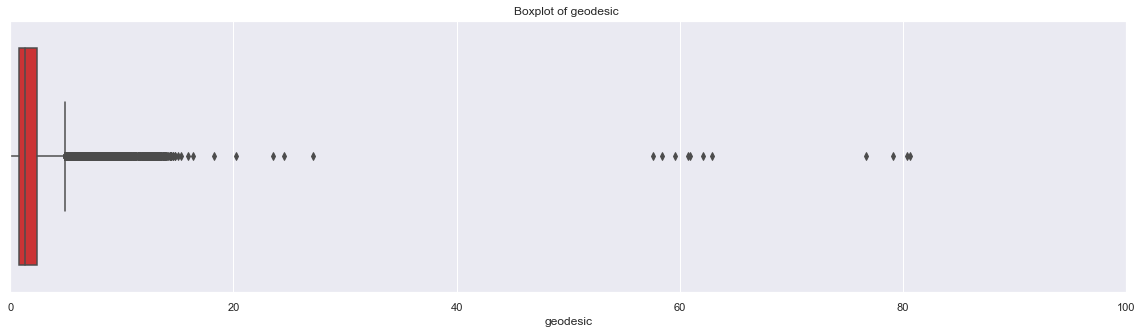
plt.xlim(0,100)

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

plt.title('Boxplot of geodesic ')

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

plt.show()



outlier\_treatment('geodesic')

2.425668049965582 0.7815214474966259

-1.6846984562068081 4.891887953669016

pd.DataFrame(train.isnull().sum())

|  | **0** |
| --- | --- |
| pickup\_datetime | 0 |
| fare\_amount | 0 |
| pickup\_longitude | 0 |
| pickup\_latitude | 0 |
| dropoff\_longitude | 0 |
| dropoff\_latitude | 0 |
| passenger\_count | 0 |
| year | 0 |
| month | 0 |
| day\_of\_week | 0 |
| hour | 0 |
| session | 0 |
| seasons | 0 |
| passenger\_count\_1 | 0 |
| passenger\_count\_2 | 0 |
| passenger\_count\_3 | 0 |
| passenger\_count\_4 | 0 |
| passenger\_count\_5 | 0 |
| passenger\_count\_6 | 0 |
| season\_fall | 0 |
| season\_spring | 0 |
| season\_summer | 0 |
| season\_winter | 0 |
| great\_circle | 0 |
| geodesic | 1348 |

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

## Feature Selection

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') )

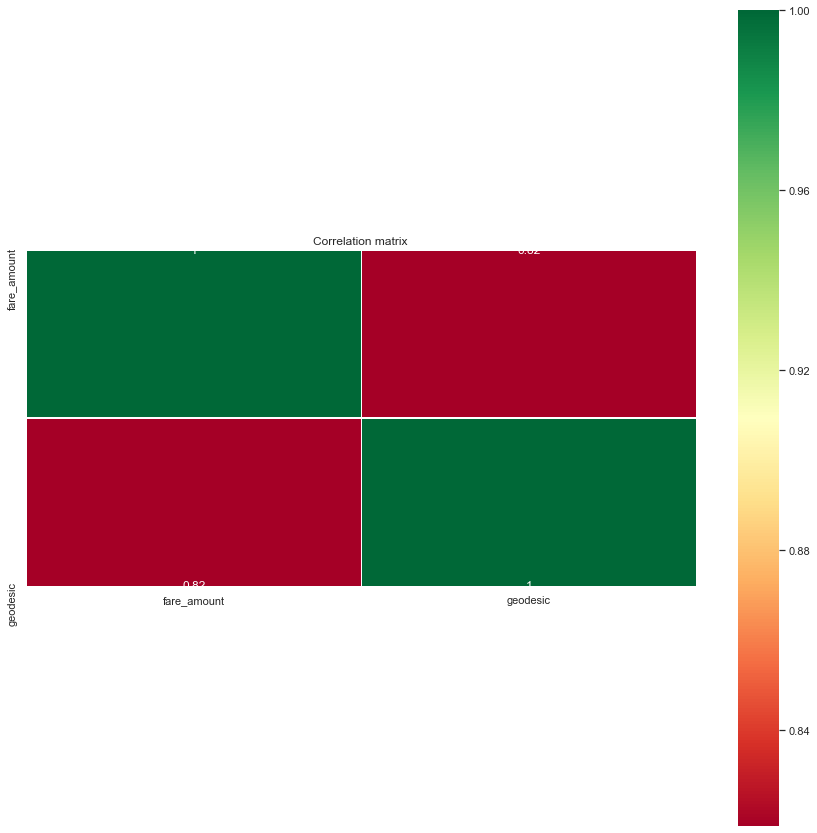
plt.figure(figsize=(15,15))

\_ = 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()

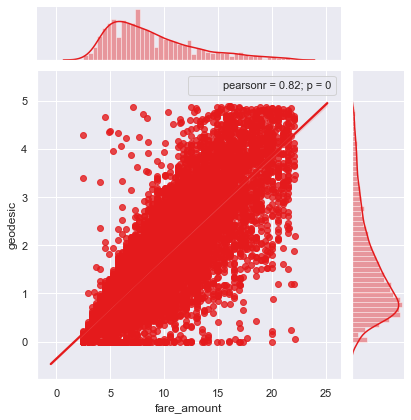


\_ = sns.jointplot(x='fare\_amount',y='geodesic',data=train,kind = 'reg')

\_.annotate(stats.pearsonr)

# plt.savefig('jointct.png')

plt.show()



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')

passenger\_count\_2 and passenger\_count\_3 are dependent on each other with 1.7121597150859386e-27 ----Remove

passenger\_count\_2 and passenger\_count\_4 are dependent on each other with 1.158785630549477e-13 ----Remove

passenger\_count\_2 and passenger\_count\_5 are dependent on each other with 1.02944814727332e-42 ----Remove

passenger\_count\_2 and passenger\_count\_6 are dependent on each other with 1.0340832329142975e-12 ----Remove

passenger\_count\_2 and season\_spring are independent on each other with 0.983869902597337 ----Keep

passenger\_count\_2 and season\_summer are dependent on each other with 0.024589215954971648 ----Remove

passenger\_count\_2 and season\_winter are independent on each other with 0.520383954383566 ----Keep

train.columns

Index(['pickup\_datetime', 'fare\_amount', 'pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count', 'year',

'month', 'day\_of\_week', 'hour', 'session', 'seasons',

'passenger\_count\_1', 'passenger\_count\_2', 'passenger\_count\_3',

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

'season\_fall', 'season\_spring', 'season\_summer', 'season\_winter',

'great\_circle', 'geodesic'],

dtype='object')

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

## Multicollinearity Test

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

|  | **VIF** | **Features** |
| --- | --- | --- |
| **0** | 15.268414 | Intercept |
| **1** | 1.040707 | passenger\_count\_2[T.1.0] |
| **2** | 1.019515 | passenger\_count\_3[T.1.0] |
| **3** | 1.011841 | passenger\_count\_4[T.1.0] |
| **4** | 1.024997 | passenger\_count\_5[T.1.0] |
| **5** | 1.017210 | passenger\_count\_6[T.1.0] |
| **6** | 1.642247 | season\_spring[T.1.0] |
| **7** | 1.552396 | season\_summer[T.1.0] |
| **8** | 1.587592 | season\_winter[T.1.0] |
| **9** | 1.050816 | week\_weekend[T.1.0] |
| **10** | 1.376209 | session\_night\_AM[T.1.0] |
| **11** | 1.423260 | session\_night\_PM[T.1.0] |
| **12** | 1.524808 | session\_evening[T.1.0] |
| **13** | 1.559077 | session\_morning[T.1.0] |
| **14** | 1.691361 | year\_2010[T.1.0] |
| **15** | 1.687796 | year\_2011[T.1.0] |
| **16** | 1.711103 | year\_2012[T.1.0] |
| **17** | 1.709348 | year\_2013[T.1.0] |
| **18** | 1.664992 | year\_2014[T.1.0] |
| **19** | 1.406917 | year\_2015[T.1.0] |
| **20** | 1.025420 | Geodesic |

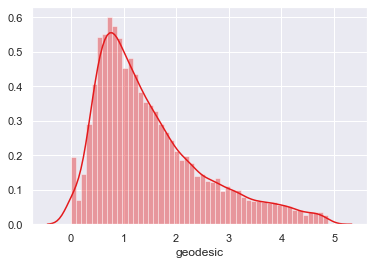
train[num\_var].var()

fare\_amount 20.044259

geodesic 1.078511

dtype: float64

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



plt.figure()

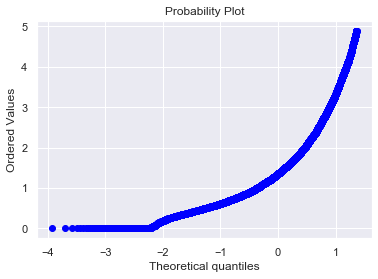
stats.probplot(train['geodesic'], dist='norm', fit=True,plot=plt)

((array([-3.92007182, -3.70085568, -3.58076887, ..., 3.58076887,

3.70085568, 3.92007182]),

array([ 0., 0., 0., ..., nan, nan, nan])),

(nan, nan, nan))



#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']))

train['geodesic'].var()

0.045126199298793135

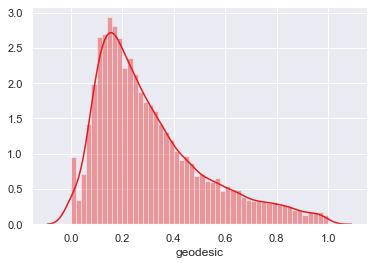
sns.distplot(train['geodesic'],bins=50)

plt.savefig('distplot.png')

plt.figure()

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

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



plt.figure()

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

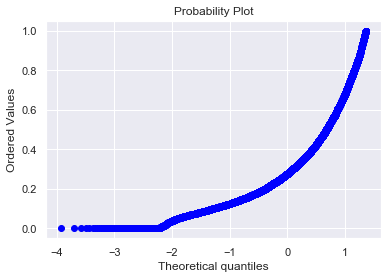
# plt.savefig('qq prob plot.png')

((array([-3.92007182, -3.70085568, -3.58076887, ..., 3.58076887,

3.70085568, 3.92007182]),

array([ 0., 0., 0., ..., nan, nan, nan])),

(nan, nan, nan))



train.columns

Index(['pickup\_datetime', 'fare\_amount', 'pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count', 'year',

'month', 'day\_of\_week', 'hour', 'session', 'seasons',

'passenger\_count\_1', 'passenger\_count\_2', 'passenger\_count\_3',

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

'season\_fall', 'season\_spring', 'season\_summer', 'season\_winter',

'great\_circle', 'geodesic'],

dtype='object')

# df4=train.copy()

train=df4.copy()

# f4=test.copy()

test=f4.copy()

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

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

train.columns

Index(['pickup\_datetime', 'fare\_amount', 'pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count', 'year',

'month', 'day\_of\_week', 'hour', 'session', 'seasons',

'passenger\_count\_1', 'passenger\_count\_3', 'passenger\_count\_4',

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

'season\_spring', 'season\_summer', 'season\_winter', 'great\_circle',

'geodesic'],

dtype='object')

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, 24) (11745, 23) (3915, 23) (11745,) (3915,)

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

# 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

# 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()

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

# Create a linear regression object: reg

reg = LinearRegression()

# Compute 5-fold cross-validation scores: cv\_scores

cv\_scores = cross\_val\_score(reg,X,y,cv=5,scoring='neg\_mean\_squared\_error')

# Print the 5-fold cross-validation scores

print(cv\_scores)

print("Average 5-Fold CV Score: {}".format(np.mean(cv\_scores)))

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

Average 5-Fold CV Score: -5.297518668356715

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

param\_dist = {'alpha':np.logspace(-4, 0, 50),

'normalize':[True,False],

'max\_iter':range(500,5000,500)}

# Instantiate a Decision ridge classifier: ridge

ridge = Ridge()

# Instantiate the gridSearchCV object: ridge\_cv

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

# Fit it to the data

ridge\_cv.fit(X, y)

# Print the tuned parameters and score

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

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

Tuned Decision ridge Parameters: {'alpha': 0.0005428675439323859, 'max\_iter': 500, 'normalize': True}

Best score is 0.7354637543642097

# Instantiate a ridge regressor: ridge

ridge = Ridge(alpha=0.0005428675439323859, normalize=True,max\_iter = 500)

# Fit the regressor to the data

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

# Compute and print the coefficients

ridge\_coef = ridge.coef\_

print(ridge\_coef)

# Plot the coefficients

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

plt.plot(range(len(test.columns)), ridge\_coef)

plt.xticks(range(len(test.columns)), test.columns.values, rotation=60)

plt.margins(0.02)

# plt.savefig('ridge coefficients')

plt.show()

test\_scores(ridge)

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

param\_dist = {'alpha':np.logspace(-4, 0, 50),

'normalize':[True,False],

'max\_iter':range(500,5000,500)}

# Instantiate a Decision lasso classifier: lasso

lasso = Lasso()

# Instantiate the gridSearchCV object: lasso\_cv

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

# Fit it to the data

lasso\_cv.fit(X, y)

# Print the tuned parameters and score

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

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

Tuned Decision lasso Parameters: {'alpha': 0.00021209508879201905, 'max\_iter': 500, 'normalize': False}

Best score is 0.7354642751453719

# Instantiate a lasso regressor: lasso

lasso = Lasso(alpha=0.00021209508879201905, normalize=False,max\_iter = 500)

# Fit the regressor to the data

lasso.fit(X,y)

# Compute and print the coefficients

lasso\_coef = lasso.coef\_

print(lasso\_coef)

# Plot the coefficients

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

plt.ylim(-1,10)

plt.plot(range(len(test.columns)), lasso\_coef)

plt.xticks(range(len(test.columns)), test.columns.values, rotation=60)

plt.margins(0.02)

plt.savefig('lasso coefficients')

plt.show()

test\_scores(lasso)

[ 0.11458721 0.10614226 0.12840196 0.0970443 -0.3752226 -0.39314893

-0.42775038 -0.18824363 -0.35500483 -0.45186563 -0.67320664 -0.7489739

-0.04730828 -0.02357537 0.4703557 1.32109229 1.51204606 1.78037818

16.71624519]

## Decision Tree Regression

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

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

# Instantiate a Decision Tree classifier: tree

tree = DecisionTreeRegressor()

# Instantiate the gridSearchCV object: tree\_cv

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

# Fit it to the data

tree\_cv.fit(X, y)

# Print the tuned parameters and score

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

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

# Instantiate a tree regressor: tree

tree = DecisionTreeRegressor(max\_depth= 6, min\_samples\_split=2)

# Fit the regressor to the data

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

# Compute and print the coefficients

tree\_features = tree.feature\_importances\_

print(tree\_features)

# Sort test importances in descending order

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

# Rearrange test names so they match the sorted test importances

names = [test.columns[i] for i in indices]

# Creating plot

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

plt.title("test Importance")

# Add horizontal bars

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

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

plt.savefig('tree test importance')

plt.show()

# Make predictions and cal error

test\_scores(tree)

## Random Forest Regression

# 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\_))

# 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)

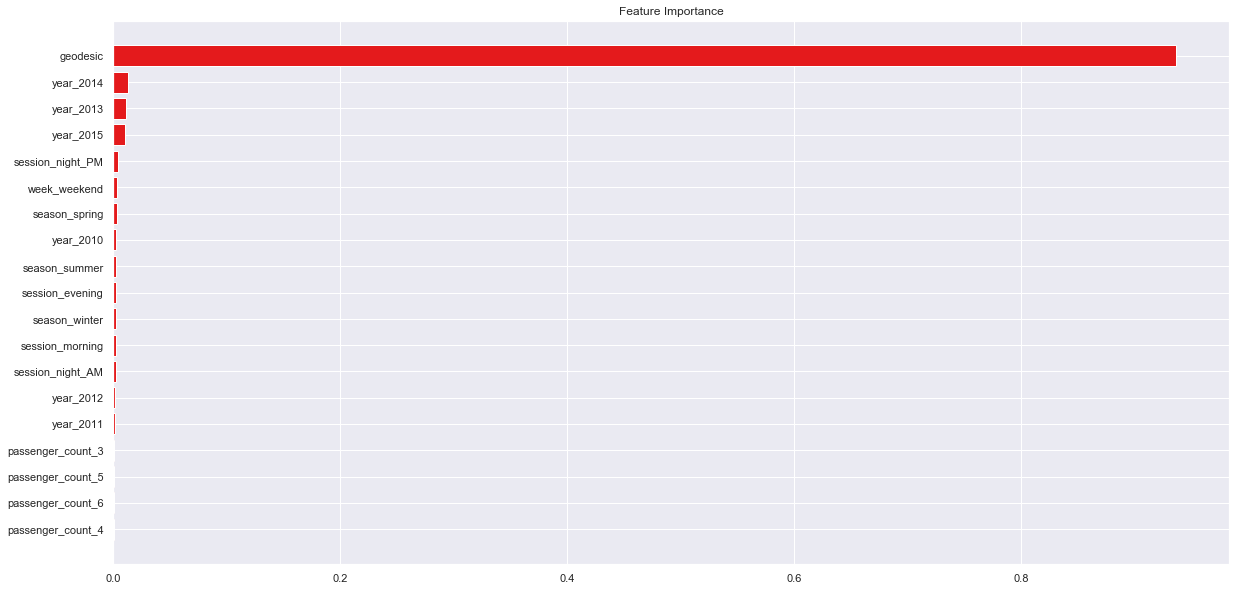
[8.38184957e-04 4.94135920e-04 7.11899115e-04 5.38216645e-04

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]



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)))

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

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

dtest = xgb.DMatrix(X\_test)

dtrain,dtest,data\_dmatrix

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()

|  | **train-rmse-mean** | **train-rmse-std** | **test-rmse-mean** | **test-rmse-std** |
| --- | --- | --- | --- | --- |
| **0** | 9.179007 | 0.018808 | 9.178729 | 0.082169 |
| **1** | 8.454776 | 0.056688 | 8.455572 | 0.126579 |
| **2** | 7.715361 | 0.097213 | 7.717719 | 0.138497 |
| **3** | 7.175781 | 0.105381 | 7.178408 | 0.145777 |
| **4** | 6.679137 | 0.155523 | 6.682420 | 0.197969 |

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

49 2.685997

Name: test-rmse-mean, dtype: float64

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

# 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

# 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

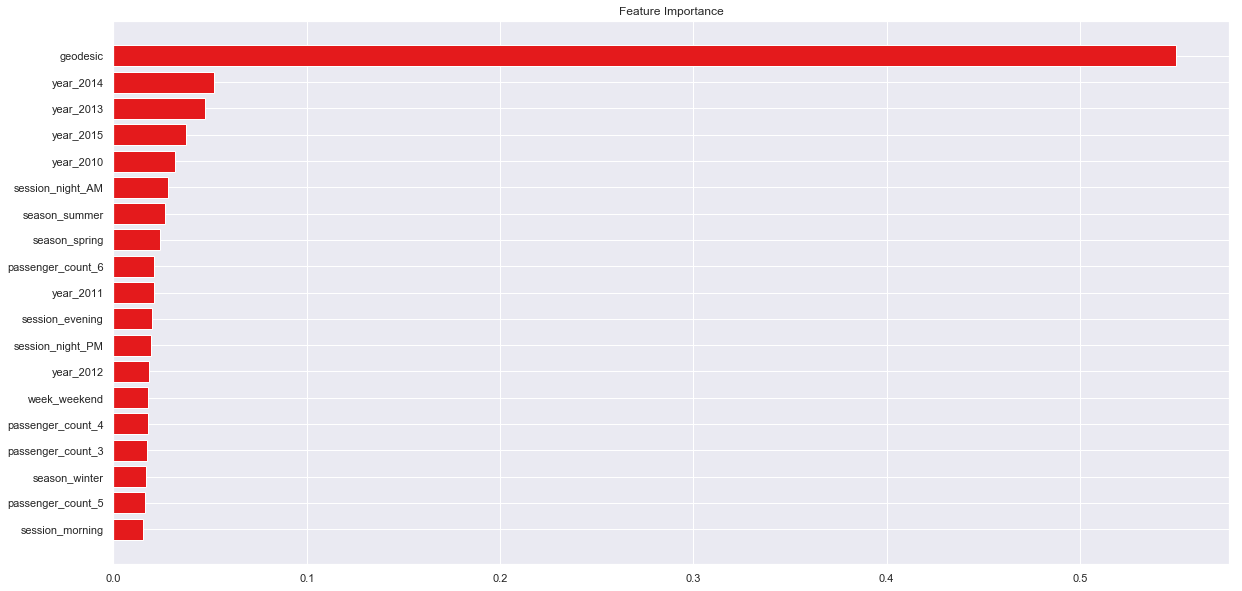
test\_scores(Xgb)

[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 ]



## Finalize model

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))

test.columns

Index(['pickup\_datetime', 'pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count', 'year',

'month', 'day\_of\_week', 'hour', 'session', 'seasons',

'passenger\_count\_1', 'passenger\_count\_3', 'passenger\_count\_4',

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

'season\_spring', 'season\_summer', 'season\_winter', 'great\_circle',

'geodesic'],

dtype='object')

train.columns

Index(['pickup\_datetime', 'fare\_amount', 'pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count', 'year',

'month', 'day\_of\_week', 'hour', 'session', 'seasons',

'passenger\_count\_1', 'passenger\_count\_3', 'passenger\_count\_4',

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

'season\_spring', 'season\_summer', 'season\_winter', 'great\_circle',

'geodesic'],

dtype='object')

train.shape

(15660, 24)

test.shape

(9914, 23)

a=pd.read\_csv('C:/Users/DELL/Desktop/data/test.csv')

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

# 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)

**Submission**

# Predictions

pred = Xgb.predict(test.values)

pred\_results\_wrt\_date = pd.DataFrame({"pickup\_datetime":test\_pickup\_datetime,"fare\_amount" : pred})

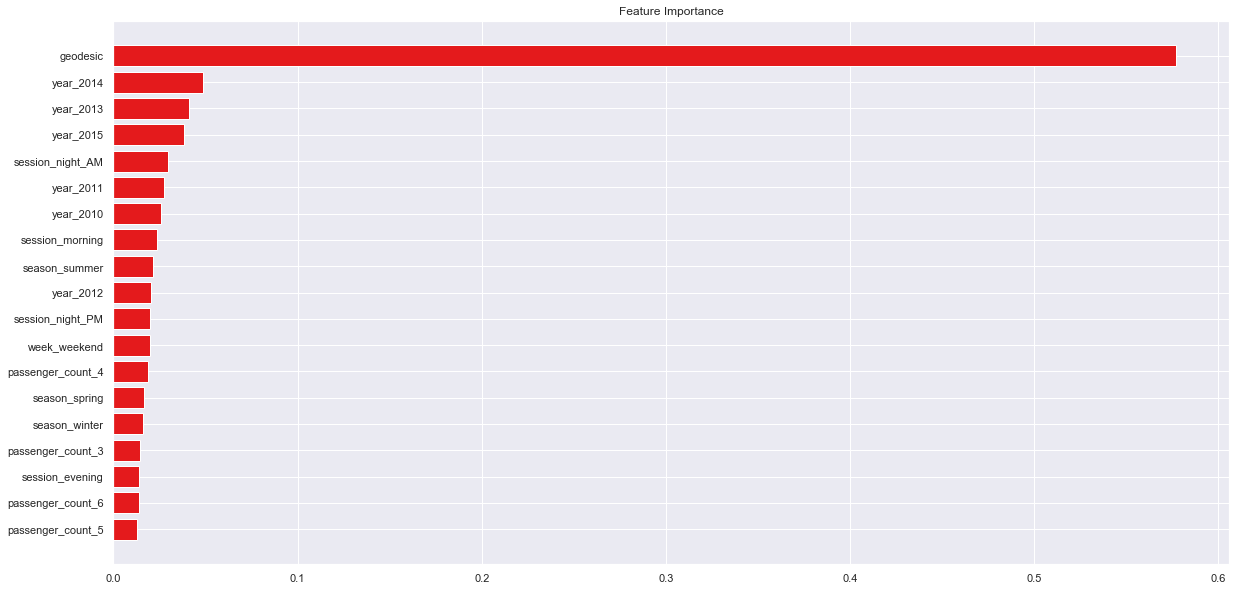
pred\_results\_wrt\_date.to\_csv("predictions\_xgboost.csv",index=False)

[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]



pred\_results\_wrt\_date

|  | **pickup\_datetime** | **prediction** |
| --- | --- | --- |
| **0** | 2015-01-27 13:08:24 UTC | 10.346493 |
| **1** | 2015-01-27 13:08:24 UTC | 10.847954 |
| **2** | 2011-10-08 11:53:44 UTC | 4.352736 |
| **3** | 2012-12-01 21:12:12 UTC | 7.438527 |
| **4** | 2012-12-01 21:12:12 UTC | 15.103196 |
| **5** | 2012-12-01 21:12:12 UTC | 10.278064 |
| **6** | 2011-10-06 12:10:20 UTC | 5.358164 |
| **7** | 2011-10-06 12:10:20 UTC | 15.853908 |
| **8** | 2011-10-06 12:10:20 UTC | 11.389716 |
| **9** | 2014-02-18 15:22:20 UTC | 6.736518 |
| **10** | 2014-02-18 15:22:20 UTC | 9.530075 |
| **11** | 2014-02-18 15:22:20 UTC | 15.778401 |
| **12** | 2010-03-29 20:20:32 UTC | 4.665674 |
| **13** | 2010-03-29 20:20:32 UTC | 6.754781 |
| **14** | 2011-10-06 03:59:12 UTC | 8.571068 |
| **15** | 2011-10-06 03:59:12 UTC | 14.031708 |
| **16** | 2012-07-15 16:45:04 UTC | 4.630556 |
| **17** | 2012-07-15 16:45:04 UTC | 9.081897 |
| **18** | 2012-07-15 16:45:04 UTC | 4.900045 |
| **19** | 2012-07-15 16:45:04 UTC | 4.785850 |
| **20** | 2014-10-29 02:09:56 UTC | 7.832850 |
| **21** | 2014-06-14 13:39:00 UTC | 8.743765 |
| **22** | 2014-06-14 13:39:00 UTC | 7.215349 |
| **23** | 2014-06-14 13:39:00 UTC | 8.504488 |
| **24** | 2014-06-14 13:39:00 UTC | 17.346186 |
| **25** | 2014-06-14 13:39:00 UTC | 6.529103 |
| **26** | 2014-06-14 13:39:00 UTC | 16.912596 |
| **27** | 2014-06-14 13:39:00 UTC | 16.912596 |
| **28** | 2014-06-14 13:39:00 UTC | 6.554900 |
| **29** | 2014-06-14 13:39:00 UTC | 15.397476 |
| **...** | ... | ... |
| **9884** | 2013-09-25 22:00:00 UTC | 16.823784 |
| **9885** | 2013-09-25 22:00:00 UTC | 15.257554 |
| **9886** | 2013-09-25 22:00:00 UTC | 18.848145 |
| **9887** | 2013-09-25 22:00:00 UTC | 7.673367 |
| **9888** | 2013-09-25 22:00:00 UTC | 7.120000 |
| **9889** | 2013-09-25 22:00:00 UTC | 9.490878 |
| **9890** | 2013-09-25 22:00:00 UTC | 10.463439 |
| **9891** | 2013-09-25 22:00:00 UTC | 17.461910 |
| **9892** | 2013-09-25 22:00:00 UTC | 10.463439 |
| **9893** | 2013-09-25 22:00:00 UTC | 14.953842 |
| **9894** | 2013-09-25 22:00:00 UTC | 11.040827 |
| **9895** | 2013-09-25 22:00:00 UTC | 16.823784 |
| **9896** | 2013-09-25 22:00:00 UTC | 9.518365 |
| **9897** | 2015-02-20 11:08:29 UTC | 16.383884 |
| **9898** | 2015-01-12 15:36:37 UTC | 5.795610 |
| **9899** | 2015-06-07 00:38:14 UTC | 17.494331 |
| **9900** | 2015-04-12 21:56:22 UTC | 8.176169 |
| **9901** | 2015-04-10 11:56:54 UTC | 8.239670 |
| **9902** | 2015-06-25 01:01:46 UTC | 13.320436 |
| **9903** | 2015-05-29 10:02:42 UTC | 10.580725 |
| **9904** | 2015-06-30 20:03:50 UTC | 16.939375 |
| **9905** | 2015-02-27 19:36:02 UTC | 16.650532 |
| **9906** | 2015-06-15 01:00:06 UTC | 5.315781 |
| **9907** | 2015-02-03 09:00:58 UTC | 16.534311 |
| **9908** | 2015-05-19 13:58:11 UTC | 8.485254 |
| **9909** | 2015-05-10 12:37:51 UTC | 10.116715 |
| **9910** | 2015-01-12 17:05:51 UTC | 12.283984 |
| **9911** | 2015-04-19 20:44:15 UTC | 18.129465 |
| **9912** | 2015-01-31 01:05:19 UTC | 16.728676 |
| **9913** | 2015-01-18 14:06:23 UTC | 6.801928 |

9914 rows × 2 columns



# Save the model as a pickle in a

# 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')

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