In [1]:
```
import os
import pandas as pd
import numpy as np
import collections as defaultdict
from scipy.stats import hmean
from scipy.spatial.distance import cdist
from scipy import stats
import numbers
from fancyimpute import KNN
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
import seaborn as sns
from random import randrange, uniform
```

```
Using TensorFlow backend.
```

In [2]:
```
################# Set current Workind Directory #########################
os.chdir("C:/Users/User/Desktop/Project 3")
os.getcwd()
```

Out[2]:
```
'C:\\Users\\User\\Desktop\\Project 3'
```

In [3]:
```
#################### Load CSV #################################
cab_train = pd.read_csv("train_cab.csv")
```

In [4]:
```
cab_train.head()
```

Out[4]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_la
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.7
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.7
2	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.7
3	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.7
4	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.7

In [5]:
```
cab_train.shape
```

Out[5]:
```
(16067, 7)
```

A. Exploratory Data Analysis

In [6]:
```
############################## Find The Datatypes ##############################
#
cab_train.dtypes
```

Out[6]:
```
fare_amount           object
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count      float64
dtype: object
```

In [7]:
```
############################## convert catagoric to numeric ###################
#
cab_train['fare_amount'] = cab_train['fare_amount'].convert_objects(convert_nu
meric=True)
```

```
C:\Users\User\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarn
ing: convert_objects is deprecated.  To re-infer data dtypes for object colum
ns, use Series.infer_objects()
For all other conversions use the data-type specific converters pd.to_datetim
e, pd.to_timedelta and pd.to_numeric.
```

In [8]:
```
############################## Convert object to datetime ####################
#
from datetime import datetime
import calendar
cab_train.pickup_datetime = pd.to_datetime(cab_train.pickup_datetime, errors=
'coerce')
```

In [9]:
```
############## get info of all attributes #################
cab_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16067 entries, 0 to 16066
Data columns (total 7 columns):
fare_amount          16042 non-null float64
pickup_datetime      16066 non-null datetime64[ns]
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](1), float64(6)
memory usage: 878.7 KB
```

In [10]:
```
######################## Split Data time ##########################
cab_train['pickup_year']= cab_train['pickup_datetime'].dt.year
cab_train['pickup_month']=cab_train['pickup_datetime'].dt.month
cab_train['pickup_day_of_month']=cab_train['pickup_datetime'].dt.day
cab_train['pickup_hour']=cab_train['pickup_datetime'].dt.hour
cab_train['pickup_minute']=cab_train['pickup_datetime'].dt.minute
cab_train['pickup_second']=cab_train['pickup_datetime'].dt.second
#cab_train['pickup_weekday_name']=cab_train['pickup_datetime'].dt.weekday_name
```

In [11]:
```
cab_train.head()
```

Out[11]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_la
0	4.5	2009-06-15 17:26:21	-73.844311	40.721319	-73.841610	40.7
1	16.9	2010-01-05 16:52:16	-74.016048	40.711303	-73.979268	40.7
2	5.7	2011-08-18 00:35:00	-73.982738	40.761270	-73.991242	40.7
3	7.7	2012-04-21 04:30:42	-73.987130	40.733143	-73.991567	40.7
4	5.3	2010-03-09 07:51:00	-73.968095	40.768008	-73.956655	40.7

In [12]:
```
##################### Drop pickup_datetime #####################

cab_train = cab_train.drop(['pickup_datetime'], axis=1)
```

In [13]:
```
cab_train = cab_train[cab_train['pickup_longitude']!=0]
cab_train = cab_train[cab_train['pickup_latitude']!=0]
cab_train = cab_train[cab_train['dropoff_longitude']!=0]
cab_train = cab_train[cab_train['dropoff_latitude']!=0]
```

In [14]:
```
##################### Calutlate the distance using latlong #################
#####

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

#Calculate the great circle distance between two points on the earth (specifie
d in decimal degrees)

def haversine_np(lon1, lat1, lon2, lat2):

    # Convert latitude and longitude to radians
    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])

    # Find the differences
    dlon = lon2 - lon1
    dlat = lat2 - lat1

    # Apply the formula
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2

    # Calculate the angle (in radians)
    c = 2 * np.arcsin(np.sqrt(a))

    # Convert to kilometers
    km = 6367 * c

    return km

cab_train['Trip_distance_KM'] =  haversine_np(cab_train['pickup_longitude'], c
ab_train['pickup_latitude'],
                        cab_train['dropoff_longitude'], cab_train['dropoff_lat
itude'])
```

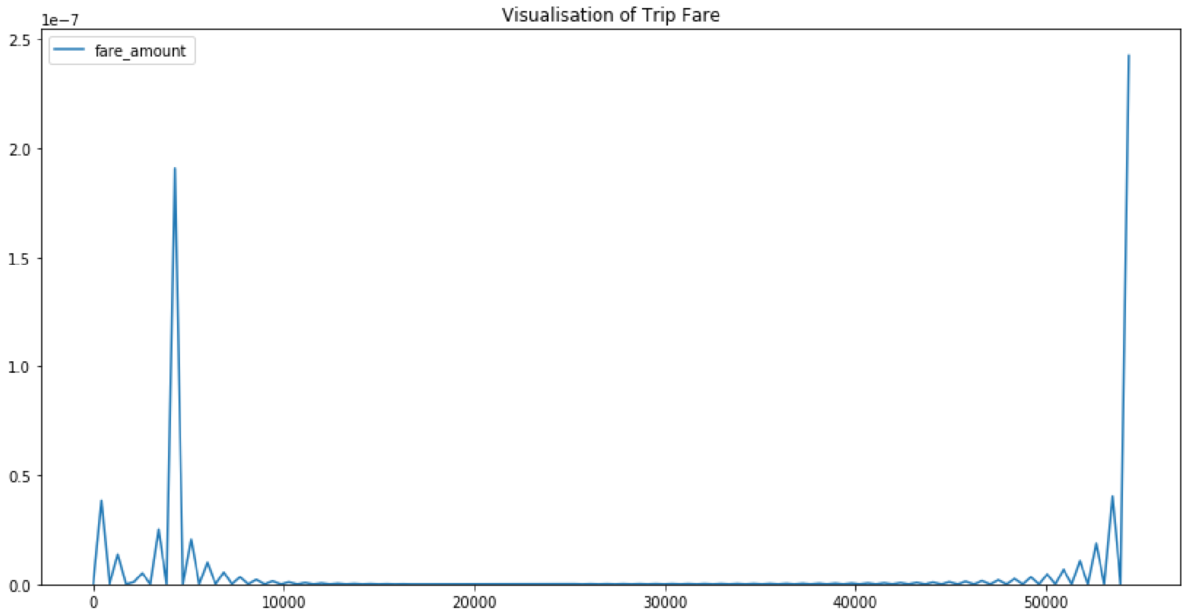
In [15]:
```
################## Trip_distance_KM describtion ################

cab_train["Trip_distance_KM"].describe()
```

Out[15]:
```
count    15736.000000
mean         3.371577
std          4.127624
min          0.000000
25%          1.256179
50%          2.168648
75%          3.892358
max        101.031147
Name: Trip_distance_KM, dtype: float64
```

In [17]:
```
##################### Visualisatin of Trip Fare ###################
plt.figure(figsize=(14,7))
sns.kdeplot(cab_train['fare_amount']).set_title("Visualisation of Trip Fare")
cab_train.loc[cab_train['fare_amount']<0].shape
cab_train["fare_amount"].describe()
```

Out[17]:
```
count    15713.000000
mean        15.085124
std        434.940822
min         -3.000000
25%          6.000000
50%          8.500000
75%         12.500000
max      54343.000000
Name: fare_amount, dtype: float64
```

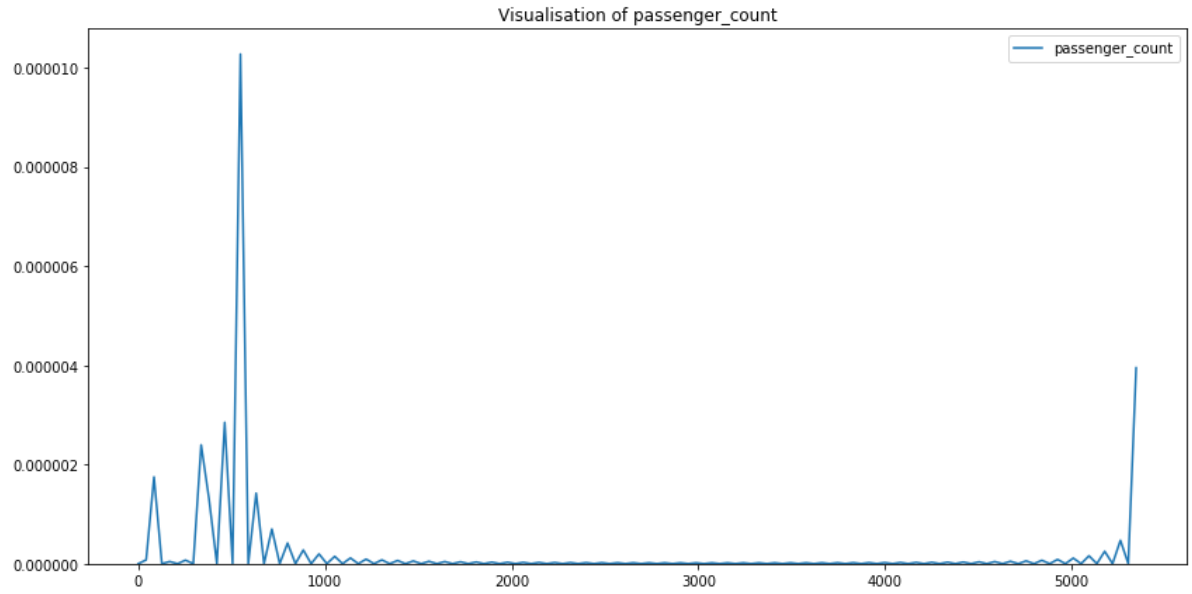


In [18]:
```
########### Trip fare should not be -ve so here we drop the -ve values #######
#############
cab_train=cab_train.loc[cab_train['fare_amount']>0]
cab_train.shape
```

Out[18]:
```
(15709, 13)
```

In [19]:
```
####################################### Visualisation of passenger_count ######
##########################
plt.figure(figsize=(14,7))
sns.kdeplot(cab_train['passenger_count']).set_title("Visualisation of passenge
r_count")
```

Out[19]:
```
Text(0.5, 1.0, 'Visualisation of passenger_count')
```

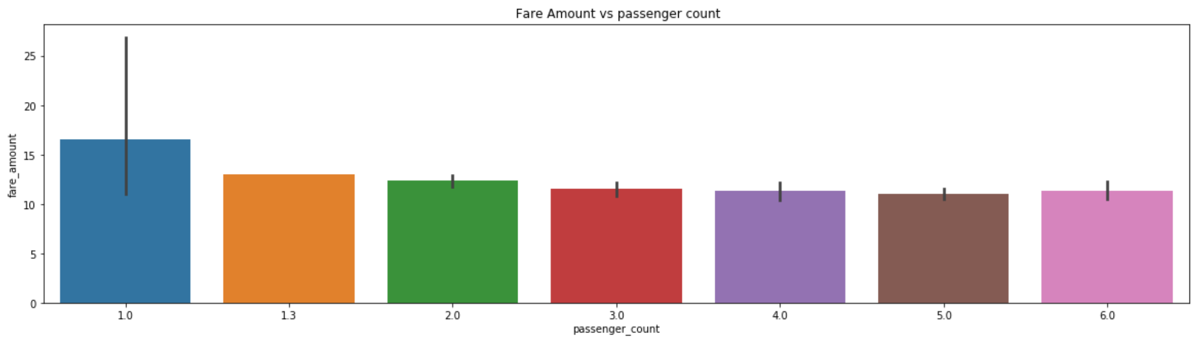


In [20]:
```
################################# Passenger count is in between 1 to 6 ########
################
cab_train=cab_train[cab_train['passenger_count']<=6]
cab_train=cab_train[cab_train['passenger_count']>=1]
cab_train["passenger_count"].describe()
```

Out[20]:
```
count    15581.000000
mean         1.650298
std          1.265902
min          1.000000
25%          1.000000
50%          1.000000
75%          2.000000
max          6.000000
Name: passenger_count, dtype: float64
```

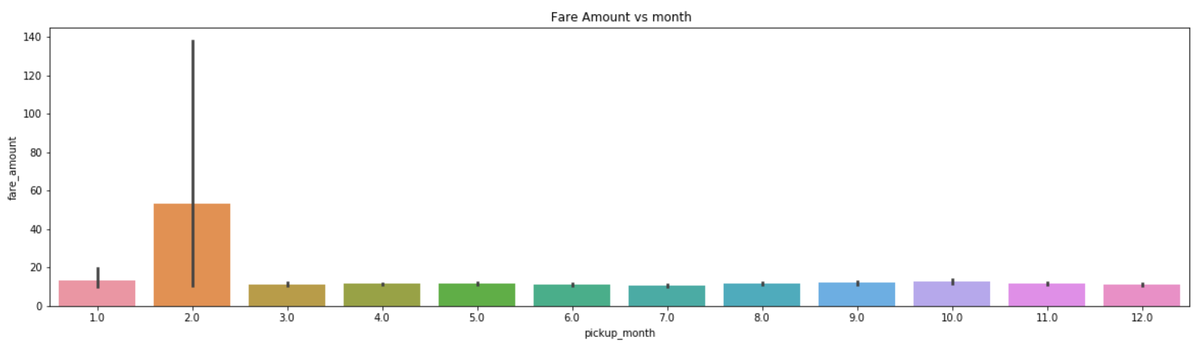
In [21]:
```
############################### Bar plot of fare amount vs passenger count ####
#################################
plt.figure(figsize=(20,5))
sns.barplot(x='passenger_count',y='fare_amount',data=cab_train).set_title(" Fa
re Amount vs passenger count")
```

Out[21]: Text(0.5, 1.0, ' Fare Amount vs passenger count')



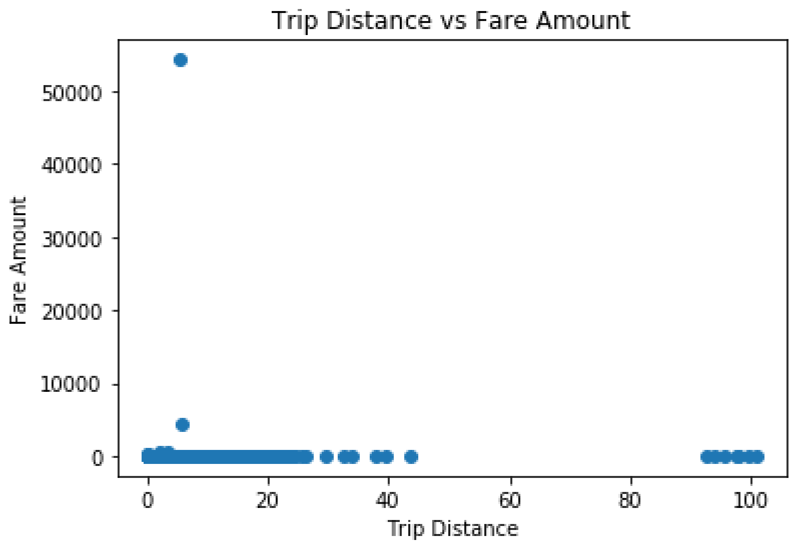
In [22]:
```
############################### Bar plot of fare amount vs Pickup year ########
#############################
plt.figure(figsize=(20,5))
sns.barplot(x='pickup_month',y='fare_amount',data=cab_train).set_title(" Fare
 Amount vs month")
```

Out[22]: Text(0.5, 1.0, ' Fare Amount vs month')



In [23]:
```
## Trip distance vs fare amount
plt.scatter(x=cab_train['Trip_distance_KM'],y=cab_train['fare_amount'])
plt.xlabel("Trip Distance")
plt.ylabel("Fare Amount")
plt.title("Trip Distance vs Fare Amount")
```

Out[23]: Text(0.5, 1.0, 'Trip Distance vs Fare Amount')



In [24]:
```
############################ storing the EDA data in df ######################
#####
bf = cab_train
#cab_train = bf
```

In [165]:
```
#cab_train.to_csv("train_sample1.csv",index=False)
```

In [156]:
```
cab_train.head()
```

Out[156]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_
0	4.5	-73.844311	40.721319	-73.841610	40.712278	
1	16.9	-74.016048	40.711303	-73.979268	40.782004	
2	5.7	-73.982738	40.761270	-73.991242	40.750562	
3	7.7	-73.987130	40.733143	-73.991567	40.758092	
4	5.3	-73.968095	40.768008	-73.956655	40.783762	

B. Data Preprocessing

1. Missing Value Analysis

In [25]:
```
###################################### MISSING VALUE ANALYSIS ################
##################################################

#Create dataframe with missing percentage
missing_val = pd.DataFrame(cab_train.isnull().sum())
#Reset index
missing_val = missing_val.reset_index()
#Rename variable
missing_val = missing_val.rename(columns = {'index': 'Variables', 0: 'Missing_
percentage'})
#Calculate percentage
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(cab
_train))*100
#descending order
missing_val = missing_val.sort_values('Missing_percentage', ascending = False)
.reset_index(drop = True)
```

In [26]:
```
missing_val
```

Out[26]:

	Variables	Missing_percentage
0	pickup_year	0.006418
1	pickup_month	0.006418
2	pickup_day_of_month	0.006418
3	pickup_hour	0.006418
4	pickup_minute	0.006418
5	pickup_second	0.006418
6	fare_amount	0.000000
7	pickup_longitude	0.000000
8	pickup_latitude	0.000000
9	dropoff_longitude	0.000000
10	dropoff_latitude	0.000000
11	passenger_count	0.000000
12	Trip_distance_KM	0.000000

In [27]:
```
# Actual = 40.774138
#Mean = 39.914672005073626
#Median = 40.752603
#Mode = 41.25456
cab_train['pickup_latitude'].loc[7]
```

Out[27]: 40.774138

In [127]:
```
#create missing value
cab_train['pickup_latitude'].loc[7] = np.nan
```

In [28]:
```
################ Here We select median method ####################
cab_train['pickup_year']= cab_train['pickup_year'].fillna(cab_train['pickup_ye
ar'].median())
cab_train['pickup_month']= cab_train['pickup_month'].fillna(cab_train['pickup_
month'].median())
cab_train['pickup_day_of_month']= cab_train['pickup_day_of_month'].fillna(cab_
train['pickup_day_of_month'].median())
cab_train['pickup_hour']= cab_train['pickup_hour'].fillna(cab_train['pickup_ho
ur'].median())
cab_train['pickup_minute']= cab_train['pickup_minute'].fillna(cab_train['picku
p_minute'].median())
cab_train['pickup_second']= cab_train['pickup_second'].fillna(cab_train['picku
p_second'].median())
```

In [29]:
```
cab_train.isnull().sum()
```

Out[29]:
```
fare_amount            0
pickup_longitude       0
pickup_latitude        0
dropoff_longitude      0
dropoff_latitude       0
passenger_count        0
pickup_year            0
pickup_month           0
pickup_day_of_month    0
pickup_hour            0
pickup_minute          0
pickup_second          0
Trip_distance_KM       0
dtype: int64
```

In [130]:
```
#################  MEAN METHOD #################
#cab_train['pickup_latitude']= cab_train['pickup_latitude'].fillna(cab_train
['pickup_latitude'].mean())
################# MODE METHOD #################
#cab_train['pickup_latitude']= cab_train['pickup_latitude'].fillna(cab_train
['pickup_latitude'].median())
```

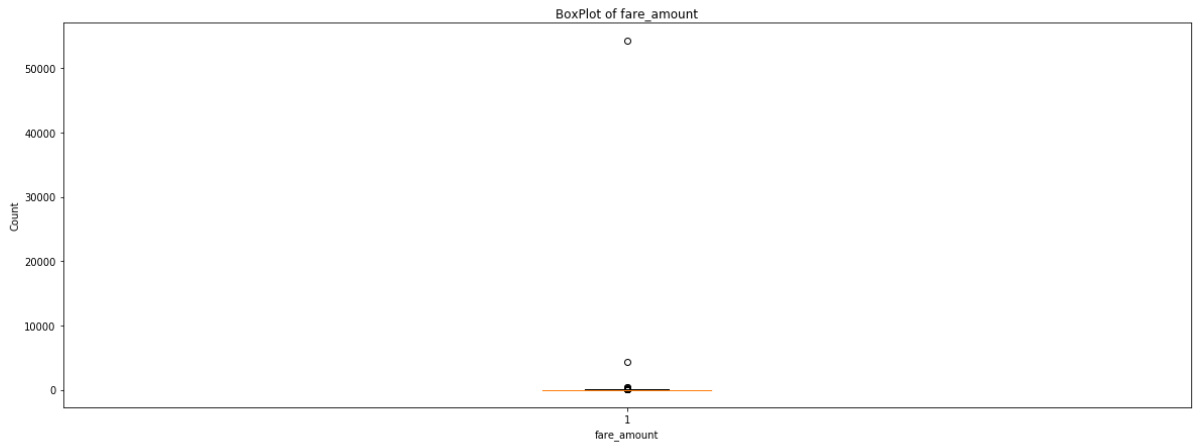
In [131]:
```
#cab_train = cab_train(KNN(k = 3).complete(cab_train), columns = cab_train.col
umns)
```

2. Outlier Analysis

In [30]:
```
########################### CANTINUOUS VARIABLES WITH TARGETVARIABLE##########
###################
cnames = ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_la
titude','Trip_distance_KM','fare_amount']
```

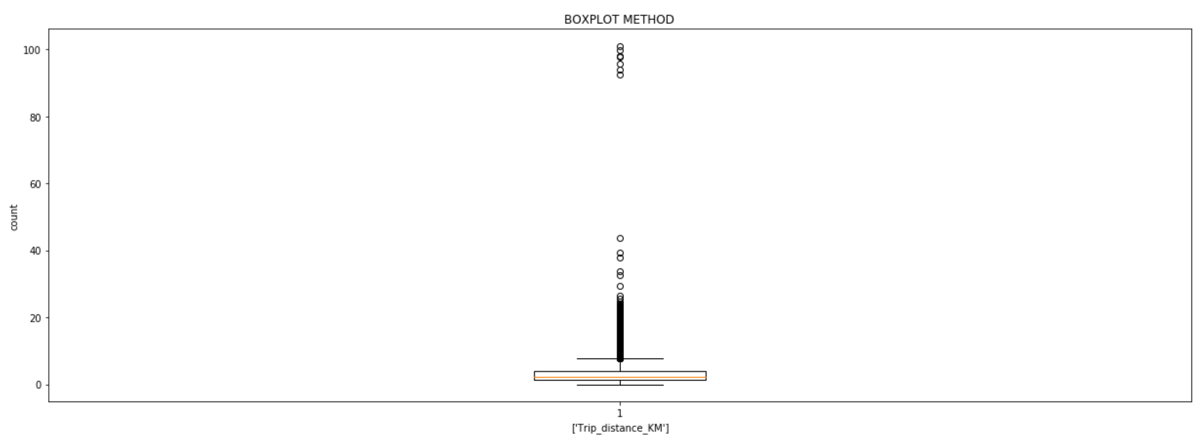
In [31]:
```
########################################## OUTLIER ANALYSIS ###################
#####################
plt.figure(figsize=(20,7))
plt.boxplot(cab_train['fare_amount'])
plt.xlabel('fare_amount')
plt.ylabel('Count')
plt.title("BoxPlot of fare_amount")
```

Out[31]: Text(0.5, 1.0, 'BoxPlot of fare_amount')



In [32]:
```
##### BOX PLOT FOR TRIP DISTANCE IN KM
plt.figure(figsize=(21,7))
plt.boxplot([cab_train['Trip_distance_KM']])
plt.title('BOXPLOT METHOD')
plt.xlabel(['Trip_distance_KM'])
plt.ylabel('count')
```

Out[32]: Text(0, 0.5, 'count')



In [33]:
```
############################### Box plot method to remove the outliers ########
######################
for i in cnames:
         print(i)
         q75, q25 = np.percentile(cab_train.loc[:,i], [75 ,25])
         print("75% ="+ str(q75))
         print("25% ="+ str(q25))
         iqr = q75 - q25
         print("IQR ="+ str(iqr))
         min = q25 - (iqr*1.5)
         max = q75 + (iqr*1.5)
         print("Min="+ str(min))
         print("Max="+ str(max))

# To remove the Outliers
cab_train = cab_train.drop(cab_train[cab_train.loc[:,i] < min].index)
cab_train = cab_train.drop(cab_train[cab_train.loc[:,i] > max].index)
```

```
pickup_longitude
75% =-73.968057
25% =-73.992372
IQR =0.02431500000000142
Min=-74.0288445
Max=-73.9315845
pickup_latitude
75% =40.76780319
25% =40.73655
IQR =0.03125319000000104
Min=40.689670215
Max=40.814682975000004
dropoff_longitude
75% =-73.96536
25% =-73.99136800000001
IQR =0.026008000000004472
Min=-74.03038000000001
Max=-73.92634799999999
dropoff_latitude
75% =40.768312
25% =40.736302
IQR =0.03200999999999965
Min=40.688287
Max=40.816327
Trip_distance_KM
75% =3.8960724377489284
25% =1.2566335303977378
IQR =2.639438907351191
Min=-2.7025248306290486
Max=7.855230798775715
fare_amount
75% =12.5
25% =6.0
IQR =6.5
Min=-3.75
Max=22.25
```

In [34]:
```
cab_train["Trip_distance_KM"].describe()
```

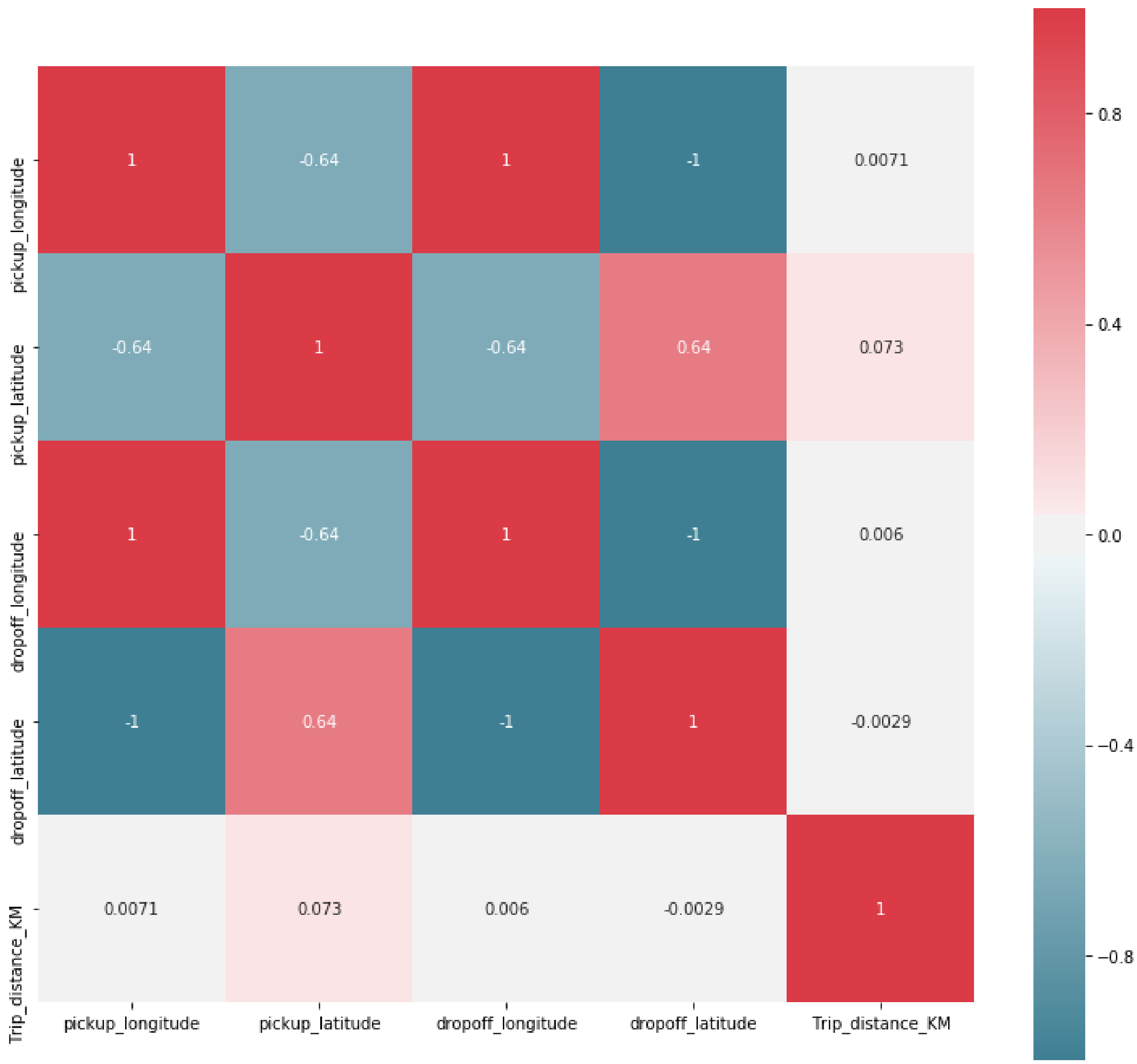
Out[34]:
```
count    14226.000000
mean         2.554514
std          2.708438
min          0.000000
25%          1.201627
50%          1.986696
75%          3.295632
max         99.708938
Name: Trip_distance_KM, dtype: float64
```

3. Feature Selection

In [35]:
```
########################### CANTINUOUS VARIABLES WITH TARGETVARIABLE##########
###################
cnames = ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_la
titude','Trip_distance_KM']
```

In [36]:
```
################################################ Feature Selection ##########
############################################
#Correlation analysis
#Correlation plot
df_corr = cab_train.loc[:,cnames]
#Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(13, 12))
#Generate correlation matrix
corr = df_corr.corr()
#Plot using seaborn library
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_
palette(220, 10, as_cmap=True), annot = True, square=True, ax=ax)
plt.plot()
```

Out[36]: []



In [37]:
```
########################## CATAGARICAL VARIABLES #############################
##
cat_names = ['pickup_year','pickup_month','pickup_day_of_month','pickup_hour',
'pickup_minute','pickup_second','passenger_count']
```

In [38]:
```
#Here we are using ANOVA test for catagorical attributes
for i in cat_names:
    f, p = stats.f_oneway(cab_train[i], cab_train["fare_amount"])
    print("P value for variable "+str(i)+" is "+str(p))
    print("f value for variable "+str(i)+" is "+str(f))
```

```
P value for variable pickup_year is 0.0
f value for variable pickup_year is 2770443965.632115
P value for variable pickup_month is 0.0
f value for variable pickup_month is 3520.1177754289
P value for variable pickup_day_of_month is 0.0
f value for variable pickup_day_of_month is 7013.444824065899
P value for variable pickup_hour is 0.0
f value for variable pickup_hour is 5194.470021689829
P value for variable pickup_minute is 0.0
f value for variable pickup_minute is 19509.690429726423
P value for variable pickup_second is 0.0
f value for variable pickup_second is 1885.7189632049021
P value for variable passenger_count is 0.0
f value for variable passenger_count is 40121.009130591825
```

In [39]:
```
# drop the variables those who are highly correlated
cab_train = cab_train.drop(['pickup_latitude','pickup_longitude','dropoff_long
itude','dropoff_latitude'], axis=1)
```

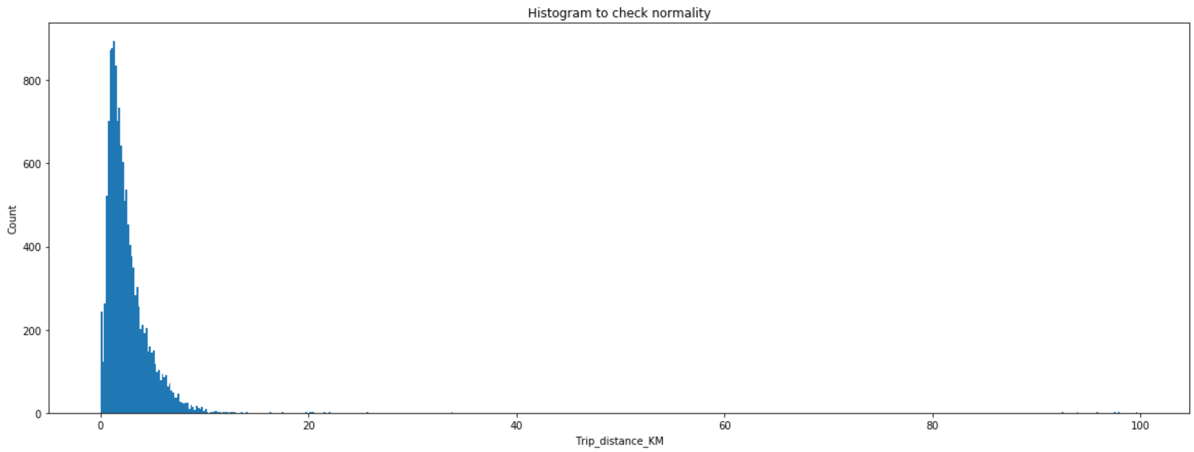
In [40]:
```
P = cab_train
```

In [143]:
```
P.to_csv("train_Visualisation.csv",index=False)
```

4. Feature Scaling

In [41]:
```
#Normality check
plt.figure(figsize=(20,7))
plt.hist(cab_train['Trip_distance_KM'], bins='auto')
plt.xlabel('Trip_distance_KM')
plt.ylabel('Count')
plt.title('Histogram to check normality')
```

Out[41]: Text(0.5, 1.0, 'Histogram to check normality')



In [42]:
```
cnames = ['Trip_distance_KM', 'fare_amount']
```

In [43]:
```
# As our variables are left sckew here we select Nomalisation method
for i in cnames:
    print(i)
    if i == 'fare_amount':
        continue
    cab_train[i] = (cab_train[i] - cab_train[i].min())/(cab_train[i].max()-cab
_train[i].min())
```

```
Trip_distance_KM
fare_amount
```

In [44]:
```
#################################### saveing data of preprocessing in pf #####
################################

#cab_train.to_csv("train_sample1.csv",index=False)
pf= cab_train
cab_train.shape
#cab_train = pf
```

Out[44]: (14226, 9)

5. Dummy Variables

In [45]:
```
###################### Get dummy variables for categorical variables  ########
##########################
df = pd.get_dummies(data = cab_train, columns = cat_names)
df = df.drop(['passenger_count_1.3'], axis=1)
df.shape
#df_A = df
```

Out[45]: (14226, 202)

In [52]:
```
df_A = df
```

In [26]:
```
df.to_csv("train_sample.csv",index=False)
```

In [46]:
```
######################### Devide data into train and test ####################
####
from sklearn.model_selection import train_test_split
Y = df['fare_amount']
df.drop(['fare_amount'], inplace = True, axis=1)
X = df
# Using train_test_split sampling function for test and train data split
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2)
```

In [47]:
```
X_test.shape
```

Out[47]: (2846, 201)

C. Modeling

1.Random Forest

In [48]:
```
##############################    RF       ###################################
##

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

model = RandomForestRegressor(n_estimators=100, random_state=0).fit(X_train, y
_train)

y_pred = model.predict(X_train)
print('Root Mean Squared Error for train:', np.sqrt(mean_squared_error(y_train
, y_pred)))

test_pred = model.predict(X_test)
print('Root Mean Squared Error for test:', np.sqrt(mean_squared_error(y_test,t
est_pred)))

## R2 ##
print("R^2 Score = "+str(r2_score(y_test,test_pred)))
```

```
Root Mean Squared Error for train: 0.8342412966487998
Root Mean Squared Error for test: 2.2222441831632613
R^2 Score = 0.6885217505821997
```

2. Linear regression

In [49]:
```
#Import libraries for LR
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Train the model using the training sets
model = sm.OLS(y_train, X_train).fit()

# predictions for train model
predictions_LR = model.predict(X_train)
# Calulating RMSE
print('Root Mean Squared Error of train:', np.sqrt(mean_squared_error(y_train,
predictions_LR)))

# predictions for train model
predictions_LR = model.predict(X_test)
# Calulating RMSE
print('Root Mean Squared Error of test:', np.sqrt(mean_squared_error(y_test,pr
edictions_LR)))

## R2 ##
print("R^2 Score = "+str(r2_score(y_test,predictions_LR)))
```

```
Root Mean Squared Error of train: 3.406019630290123
Root Mean Squared Error of test: 3.315523866502557
R^2 Score = 0.30665644006299575
```

In [50]:
```
# Print out the statistics
model.summary()
```

Out[50]:

OLS Regression Results

Dep. Variable:	fare_amount	**R-squared:**	0.334
Model:	OLS	**Adj. R-squared:**	0.323
Method:	Least Squares	**F-statistic:**	28.97
Date:	Sun, 05 May 2019	**Prob (F-statistic):**	0.00
Time:	17:21:03	**Log-Likelihood:**	-30094.
No. Observations:	11380	**AIC:**	6.058e+04
Df Residuals:	11185	**BIC:**	6.201e+04
Df Model:	194		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Trip_distance_KM	83.2748	1.192	69.878	0.000	80.939	85.611
pickup_year_2009.0	1.0929	0.081	13.460	0.000	0.934	1.252
pickup_year_2010.0	1.1649	0.081	14.415	0.000	1.007	1.323
pickup_year_2011.0	1.3477	0.081	16.616	0.000	1.189	1.507
pickup_year_2012.0	1.8088	0.081	22.232	0.000	1.649	1.968
pickup_year_2013.0	2.6240	0.081	32.444	0.000	2.465	2.783
pickup_year_2014.0	2.6891	0.085	31.801	0.000	2.523	2.855
pickup_year_2015.0	3.1019	0.119	26.071	0.000	2.869	3.335
pickup_month_1.0	0.6167	0.104	5.955	0.000	0.414	0.820
pickup_month_2.0	1.0181	0.109	9.310	0.000	0.804	1.232
pickup_month_3.0	1.1285	0.103	11.009	0.000	0.928	1.329
pickup_month_4.0	1.1579	0.106	10.968	0.000	0.951	1.365
pickup_month_5.0	1.3534	0.104	13.060	0.000	1.150	1.557
pickup_month_6.0	1.0284	0.103	9.995	0.000	0.827	1.230
pickup_month_7.0	1.0427	0.114	9.124	0.000	0.819	1.267
pickup_month_8.0	0.6958	0.120	5.810	0.000	0.461	0.931
pickup_month_9.0	1.4867	0.112	13.247	0.000	1.267	1.707
pickup_month_10.0	1.5606	0.113	13.844	0.000	1.340	1.782
pickup_month_11.0	1.4124	0.114	12.351	0.000	1.188	1.637
pickup_month_12.0	1.3281	0.113	11.788	0.000	1.107	1.549
pickup_day_of_month_1.0	0.7279	0.183	3.982	0.000	0.370	1.086
pickup_day_of_month_2.0	0.3728	0.174	2.144	0.032	0.032	0.714
pickup_day_of_month_3.0	0.4764	0.188	2.537	0.011	0.108	0.844
pickup_day_of_month_4.0	0.3469	0.181	1.917	0.055	-0.008	0.702
pickup_day_of_month_5.0	0.4517	0.182	2.475	0.013	0.094	0.809

pickup_day_of_month_6.0	0.2805	0.175	1.600	0.110	-0.063	0.624
pickup_day_of_month_7.0	0.6163	0.171	3.603	0.000	0.281	0.952
pickup_day_of_month_8.0	0.4226	0.166	2.551	0.011	0.098	0.747
pickup_day_of_month_9.0	0.4336	0.174	2.491	0.013	0.092	0.775
pickup_day_of_month_10.0	0.5747	0.169	3.401	0.001	0.243	0.906
pickup_day_of_month_11.0	0.2471	0.175	1.412	0.158	-0.096	0.590
pickup_day_of_month_12.0	0.0758	0.176	0.431	0.666	-0.269	0.420
pickup_day_of_month_13.0	0.2237	0.173	1.290	0.197	-0.116	0.563
pickup_day_of_month_14.0	0.2534	0.187	1.352	0.176	-0.114	0.621
pickup_day_of_month_15.0	0.5657	0.181	3.130	0.002	0.211	0.920
pickup_day_of_month_16.0	0.4871	0.164	2.976	0.003	0.166	0.808
pickup_day_of_month_17.0	0.5530	0.179	3.096	0.002	0.203	0.903
pickup_day_of_month_18.0	0.6379	0.176	3.626	0.000	0.293	0.983
pickup_day_of_month_19.0	0.7017	0.174	4.044	0.000	0.362	1.042
pickup_day_of_month_20.0	0.6037	0.175	3.452	0.001	0.261	0.947
pickup_day_of_month_21.0	0.7259	0.171	4.245	0.000	0.391	1.061
pickup_day_of_month_22.0	0.3532	0.171	2.067	0.039	0.018	0.688
pickup_day_of_month_23.0	0.7231	0.175	4.125	0.000	0.380	1.067
pickup_day_of_month_24.0	0.4284	0.170	2.525	0.012	0.096	0.761
pickup_day_of_month_25.0	0.1934	0.178	1.089	0.276	-0.155	0.541
pickup_day_of_month_26.0	0.3489	0.185	1.885	0.059	-0.014	0.712
pickup_day_of_month_27.0	0.4228	0.183	2.308	0.021	0.064	0.782
pickup_day_of_month_28.0	0.5321	0.174	3.058	0.002	0.191	0.873
pickup_day_of_month_29.0	0.3423	0.195	1.753	0.080	-0.041	0.725
pickup_day_of_month_30.0	0.1468	0.189	0.776	0.438	-0.224	0.518
pickup_day_of_month_31.0	0.5598	0.231	2.426	0.015	0.107	1.012
pickup_hour_0.0	0.8793	0.167	5.272	0.000	0.552	1.206
pickup_hour_1.0	0.9111	0.193	4.728	0.000	0.533	1.289
pickup_hour_2.0	0.7473	0.220	3.395	0.001	0.316	1.179
pickup_hour_3.0	1.0322	0.232	4.454	0.000	0.578	1.486
pickup_hour_4.0	0.7332	0.288	2.550	0.011	0.170	1.297
pickup_hour_5.0	-0.0796	0.325	-0.245	0.807	-0.717	0.558
pickup_hour_6.0	-0.5216	0.223	-2.344	0.019	-0.958	-0.085
pickup_hour_7.0	0.0101	0.159	0.064	0.949	-0.302	0.323
pickup_hour_8.0	0.6531	0.152	4.306	0.000	0.356	0.950
pickup_hour_9.0	0.7919	0.145	5.462	0.000	0.508	1.076
pickup_hour_10.0	0.3865	0.157	2.463	0.014	0.079	0.694

pickup_hour_11.0	0.4470	0.151	2.954	0.003	0.150	0.744
pickup_hour_12.0	0.8402	0.144	5.828	0.000	0.558	1.123
pickup_hour_13.0	0.7723	0.145	5.339	0.000	0.489	1.056
pickup_hour_14.0	0.7762	0.144	5.384	0.000	0.494	1.059
pickup_hour_15.0	0.8387	0.151	5.560	0.000	0.543	1.134
pickup_hour_16.0	0.5690	0.161	3.541	0.000	0.254	0.884
pickup_hour_17.0	0.5352	0.146	3.664	0.000	0.249	0.822
pickup_hour_18.0	0.7869	0.129	6.110	0.000	0.534	1.039
pickup_hour_19.0	0.4769	0.127	3.754	0.000	0.228	0.726
pickup_hour_20.0	0.4942	0.132	3.738	0.000	0.235	0.753
pickup_hour_21.0	0.5738	0.132	4.336	0.000	0.314	0.833
pickup_hour_22.0	0.6960	0.135	5.142	0.000	0.431	0.961
pickup_hour_23.0	0.4796	0.146	3.296	0.001	0.194	0.765
pickup_minute_0.0	0.5234	0.251	2.089	0.037	0.032	1.014
pickup_minute_1.0	0.2924	0.244	1.196	0.232	-0.187	0.771
pickup_minute_2.0	0.4381	0.248	1.766	0.077	-0.048	0.924
pickup_minute_3.0	0.1290	0.263	0.490	0.624	-0.387	0.645
pickup_minute_4.0	-0.3532	0.251	-1.406	0.160	-0.846	0.139
pickup_minute_5.0	0.2808	0.240	1.169	0.242	-0.190	0.751
pickup_minute_6.0	0.1847	0.259	0.714	0.475	-0.323	0.692
pickup_minute_7.0	0.2664	0.257	1.037	0.300	-0.237	0.770
pickup_minute_8.0	0.3633	0.275	1.320	0.187	-0.176	0.903
pickup_minute_9.0	0.0847	0.260	0.326	0.745	-0.425	0.594
pickup_minute_10.0	0.2376	0.243	0.979	0.328	-0.238	0.713
pickup_minute_11.0	0.5995	0.242	2.479	0.013	0.125	1.074
pickup_minute_12.0	0.0625	0.252	0.248	0.804	-0.431	0.556
pickup_minute_13.0	-0.0366	0.241	-0.152	0.880	-0.509	0.436
pickup_minute_14.0	0.4086	0.246	1.659	0.097	-0.074	0.891
pickup_minute_15.0	-0.3195	0.252	-1.266	0.206	-0.814	0.175
pickup_minute_16.0	0.5409	0.258	2.093	0.036	0.034	1.048
pickup_minute_17.0	0.3000	0.267	1.126	0.260	-0.222	0.822
pickup_minute_18.0	-0.0562	0.264	-0.213	0.831	-0.574	0.461
pickup_minute_19.0	0.4567	0.248	1.842	0.065	-0.029	0.943
pickup_minute_20.0	0.3048	0.235	1.299	0.194	-0.155	0.765
pickup_minute_21.0	0.4081	0.250	1.629	0.103	-0.083	0.899
pickup_minute_22.0	0.1328	0.241	0.551	0.582	-0.340	0.605
pickup_minute_23.0	0.2660	0.245	1.087	0.277	-0.214	0.746

pickup_minute_24.0	0.3919	0.246	1.596	0.110	-0.089	0.873
pickup_minute_25.0	0.4152	0.258	1.608	0.108	-0.091	0.921
pickup_minute_26.0	0.2493	0.245	1.018	0.309	-0.231	0.729
pickup_minute_27.0	0.2925	0.238	1.230	0.219	-0.174	0.759
pickup_minute_28.0	0.1583	0.239	0.662	0.508	-0.310	0.627
pickup_minute_29.0	-0.1629	0.244	-0.669	0.504	-0.641	0.315
pickup_minute_30.0	0.0651	0.257	0.253	0.800	-0.439	0.570
pickup_minute_31.0	0.1907	0.244	0.781	0.435	-0.288	0.669
pickup_minute_32.0	0.4874	0.239	2.036	0.042	0.018	0.957
pickup_minute_33.0	0.5572	0.241	2.312	0.021	0.085	1.030
pickup_minute_34.0	0.3422	0.267	1.282	0.200	-0.181	0.865
pickup_minute_35.0	0.0849	0.252	0.336	0.737	-0.410	0.580
pickup_minute_36.0	0.3998	0.274	1.459	0.145	-0.137	0.937
pickup_minute_37.0	0.1946	0.241	0.809	0.419	-0.277	0.666
pickup_minute_38.0	0.3606	0.255	1.416	0.157	-0.139	0.860
pickup_minute_39.0	0.4019	0.246	1.633	0.103	-0.081	0.885
pickup_minute_40.0	0.0936	0.268	0.349	0.727	-0.432	0.620
pickup_minute_41.0	0.0967	0.241	0.401	0.688	-0.376	0.569
pickup_minute_42.0	0.4647	0.243	1.911	0.056	-0.012	0.941
pickup_minute_43.0	-0.3331	0.238	-1.402	0.161	-0.799	0.133
pickup_minute_44.0	0.6148	0.250	2.462	0.014	0.125	1.104
pickup_minute_45.0	0.1429	0.245	0.584	0.559	-0.337	0.622
pickup_minute_46.0	-0.0230	0.244	-0.094	0.925	-0.502	0.456
pickup_minute_47.0	0.0177	0.240	0.074	0.941	-0.454	0.489
pickup_minute_48.0	0.1737	0.247	0.705	0.481	-0.310	0.657
pickup_minute_49.0	0.5796	0.246	2.360	0.018	0.098	1.061
pickup_minute_50.0	0.1053	0.252	0.418	0.676	-0.389	0.599
pickup_minute_51.0	0.4115	0.235	1.751	0.080	-0.049	0.872
pickup_minute_52.0	0.3106	0.252	1.233	0.217	-0.183	0.804
pickup_minute_53.0	-0.3125	0.244	-1.283	0.200	-0.790	0.165
pickup_minute_54.0	0.1880	0.241	0.779	0.436	-0.285	0.661
pickup_minute_55.0	0.3270	0.261	1.252	0.211	-0.185	0.839
pickup_minute_56.0	0.3466	0.246	1.412	0.158	-0.135	0.828
pickup_minute_57.0	0.0779	0.249	0.313	0.754	-0.410	0.566
pickup_minute_58.0	0.5250	0.257	2.043	0.041	0.021	1.029
pickup_minute_59.0	0.0791	0.247	0.320	0.749	-0.405	0.563
pickup_second_0.0	0.3496	0.068	5.135	0.000	0.216	0.483

pickup_second_1.0	0.2321	0.334	0.695	0.487	-0.423	0.887
pickup_second_2.0	0.4064	0.342	1.188	0.235	-0.264	1.077
pickup_second_3.0	-0.0436	0.324	-0.135	0.893	-0.678	0.591
pickup_second_4.0	0.0764	0.318	0.240	0.810	-0.547	0.700
pickup_second_5.0	0.3299	0.342	0.964	0.335	-0.341	1.001
pickup_second_6.0	0.3489	0.334	1.044	0.297	-0.306	1.004
pickup_second_7.0	0.6495	0.319	2.038	0.042	0.025	1.274
pickup_second_8.0	0.4219	0.307	1.375	0.169	-0.179	1.023
pickup_second_9.0	0.0038	0.341	0.011	0.991	-0.664	0.672
pickup_second_10.0	0.4370	0.357	1.225	0.221	-0.262	1.136
pickup_second_11.0	0.7833	0.336	2.332	0.020	0.125	1.442
pickup_second_12.0	0.6091	0.330	1.846	0.065	-0.038	1.256
pickup_second_13.0	0.9195	0.327	2.811	0.005	0.278	1.561
pickup_second_14.0	0.2897	0.338	0.857	0.391	-0.373	0.952
pickup_second_15.0	0.6458	0.314	2.055	0.040	0.030	1.262
pickup_second_16.0	-0.4846	0.338	-1.434	0.152	-1.147	0.178
pickup_second_17.0	0.1769	0.321	0.551	0.582	-0.453	0.806
pickup_second_18.0	0.6474	0.344	1.880	0.060	-0.028	1.322
pickup_second_19.0	0.1751	0.378	0.463	0.643	-0.566	0.916
pickup_second_20.0	0.2001	0.336	0.596	0.551	-0.458	0.858
pickup_second_21.0	-0.0240	0.328	-0.073	0.942	-0.667	0.619
pickup_second_22.0	-0.0836	0.346	-0.242	0.809	-0.762	0.594
pickup_second_23.0	0.1509	0.328	0.460	0.645	-0.492	0.794
pickup_second_24.0	0.0301	0.334	0.090	0.928	-0.625	0.685
pickup_second_25.0	0.5138	0.350	1.468	0.142	-0.172	1.200
pickup_second_26.0	-0.7587	0.331	-2.291	0.022	-1.408	-0.110
pickup_second_27.0	0.5538	0.298	1.856	0.064	-0.031	1.139
pickup_second_28.0	-0.5207	0.337	-1.547	0.122	-1.180	0.139
pickup_second_29.0	0.1083	0.333	0.326	0.745	-0.544	0.761
pickup_second_30.0	0.0554	0.330	0.168	0.867	-0.592	0.703
pickup_second_31.0	0.2627	0.336	0.783	0.434	-0.395	0.921
pickup_second_32.0	0.5691	0.339	1.678	0.093	-0.096	1.234
pickup_second_33.0	0.1182	0.351	0.337	0.736	-0.570	0.806
pickup_second_34.0	-0.0270	0.325	-0.083	0.934	-0.664	0.610
pickup_second_35.0	0.2716	0.314	0.865	0.387	-0.344	0.888
pickup_second_36.0	-0.8739	0.353	-2.475	0.013	-1.566	-0.182
pickup_second_37.0	0.8274	0.343	2.415	0.016	0.156	1.499

pickup_second_38.0	0.6140	0.325	1.888	0.059	-0.024	1.252
pickup_second_39.0	0.6342	0.371	1.708	0.088	-0.094	1.362
pickup_second_40.0	0.4824	0.331	1.457	0.145	-0.167	1.132
pickup_second_41.0	0.4743	0.337	1.409	0.159	-0.185	1.134
pickup_second_42.0	0.4059	0.344	1.179	0.239	-0.269	1.081
pickup_second_43.0	0.4783	0.348	1.374	0.170	-0.204	1.161
pickup_second_44.0	-0.3969	0.343	-1.157	0.247	-1.069	0.275
pickup_second_45.0	0.2467	0.351	0.703	0.482	-0.442	0.935
pickup_second_46.0	0.2170	0.336	0.646	0.518	-0.441	0.876
pickup_second_47.0	0.3142	0.357	0.880	0.379	-0.386	1.014
pickup_second_48.0	-0.1176	0.338	-0.348	0.728	-0.779	0.544
pickup_second_49.0	0.2460	0.365	0.674	0.500	-0.469	0.961
pickup_second_50.0	0.7103	0.336	2.115	0.034	0.052	1.369
pickup_second_51.0	0.2413	0.328	0.735	0.462	-0.402	0.885
pickup_second_52.0	0.0553	0.346	0.160	0.873	-0.624	0.734
pickup_second_53.0	-0.5908	0.334	-1.766	0.077	-1.246	0.065
pickup_second_54.0	-0.2220	0.324	-0.685	0.493	-0.857	0.413
pickup_second_55.0	0.2464	0.327	0.754	0.451	-0.394	0.887
pickup_second_56.0	0.2373	0.353	0.673	0.501	-0.454	0.929
pickup_second_57.0	0.2861	0.350	0.818	0.413	-0.399	0.972
pickup_second_58.0	-0.1014	0.327	-0.310	0.756	-0.742	0.539
pickup_second_59.0	1.0206	0.331	3.079	0.002	0.371	1.670
passenger_count_1.0	2.1540	0.057	37.828	0.000	2.042	2.266
passenger_count_2.0	2.3667	0.088	26.859	0.000	2.194	2.539
passenger_count_3.0	2.4116	0.148	16.286	0.000	2.121	2.702
passenger_count_4.0	2.4424	0.200	12.199	0.000	2.050	2.835
passenger_count_5.0	2.2205	0.126	17.656	0.000	1.974	2.467
passenger_count_6.0	2.2340	0.213	10.510	0.000	1.817	2.651

Omnibus:	12149.518	**Durbin-Watson:**	2.020
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6621930.506
Skew:	-4.648	**Prob(JB):**	0.00
Kurtosis:	120.809	**Cond. No.**	2.29e+16

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.58e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

3. XGboost

In [51]:
```
######################################## Gradient Boosting  ####################
##################33

# Importing library
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Building model on top of training dataset
fit_GB = GradientBoostingRegressor().fit(X_train, y_train)

# Calculating RMSE for training data to check for over fitting
pred_train = fit_GB.predict(X_train)
# Calulating RMSE
print('Root Mean Squared Error of train:', np.sqrt(mean_squared_error(y_train,
pred_train)))

# Calculating RMSE for test data to check accuracy
pred_test = fit_GB.predict(X_test)
# Calulating RMSE
print('Root Mean Squared Error of test:', np.sqrt(mean_squared_error(y_test,pr
ed_test)))
print("R^2 Score for test(coefficient of determination) = "+str(r2_score(y_tes
t,pred_test)))
```

```
Root Mean Squared Error of train: 2.054400325878095
Root Mean Squared Error of test: 2.1215625146618264
R^2 Score for test(coefficient of determination) = 0.7161062472440993
```

D. Predicting the new test case

In [53]:
```
train = df_A
```

In [ ]:
```
y_train= df_A['fare_amount']
df_A.drop(['fare_amount'], inplace = True, axis=1)
X_train = df
```

In [33]:
```
#################### Load CSV #################################
test = pd.read_csv("test_sample.csv")
```

In [51]:
```
X_test = test
X_test.shape
```

Out[51]:
```
(9914, 201)
```

In [41]:
```
######################################## Gradient Boosting  #####################
##################33

# Importing library
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Building model on top of training dataset
fit_GB = GradientBoostingRegressor().fit(X_train, y_train)
#
pred_test = fit_GB.predict(X_test)
```

```
Root Mean Squared Error of train: 2.0616956653625538
```

In [53]:
```
Test_prediction = pd.DataFrame(pred_test)
```

In [60]:
```
Test_prediction.describe()
```

Out[60]:

	0
count	9914.000000
mean	9.553086
std	3.972477
min	3.839144
25%	6.427308
50%	8.492642
75%	11.819192
max	21.007886

In [61]:
```
test['Test_prediction'] = Test_prediction
```

In [63]:
```
test.head()
```

Out[63]:

	Trip_distance_KM	pickup_year_2009	pickup_year_2010	pickup_year_2011	pickup_year_2012
0	0.023234	0	0	0	0
1	0.024254	0	0	0	0
2	0.006187	0	0	1	0
3	0.019611	0	0	0	1
4	0.053875	0	0	0	1

5 rows × 202 columns

In [64]:
```
test.to_csv("test_predict.csv",index=False)
```

E. Data Visualization

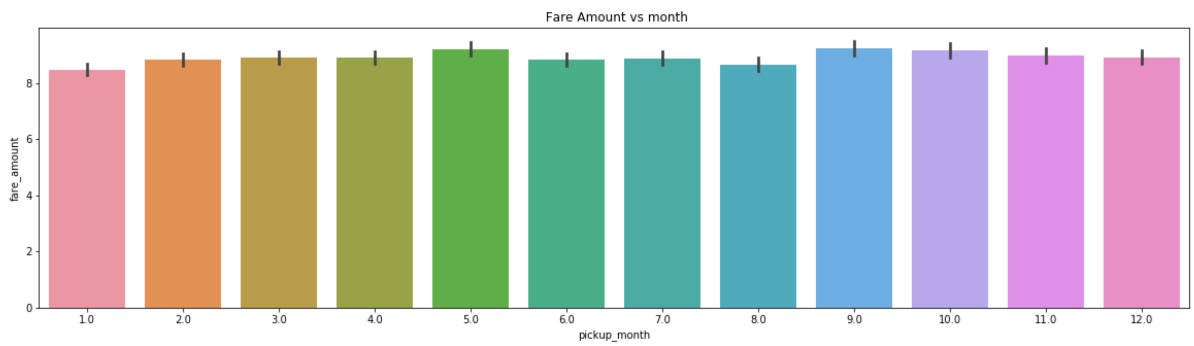
In [92]:
```
pf.head()
```

Out[92]:

	fare_amount	passenger_count	pickup_year	pickup_month	pickup_day_of_month	pickup_hou
0	4.5	1.0	2009.0	6.0	15.0	17.
1	16.9	1.0	2010.0	1.0	5.0	16.
2	5.7	2.0	2011.0	8.0	18.0	0.
3	7.7	1.0	2012.0	4.0	21.0	4.
4	5.3	1.0	2010.0	3.0	9.0	7.

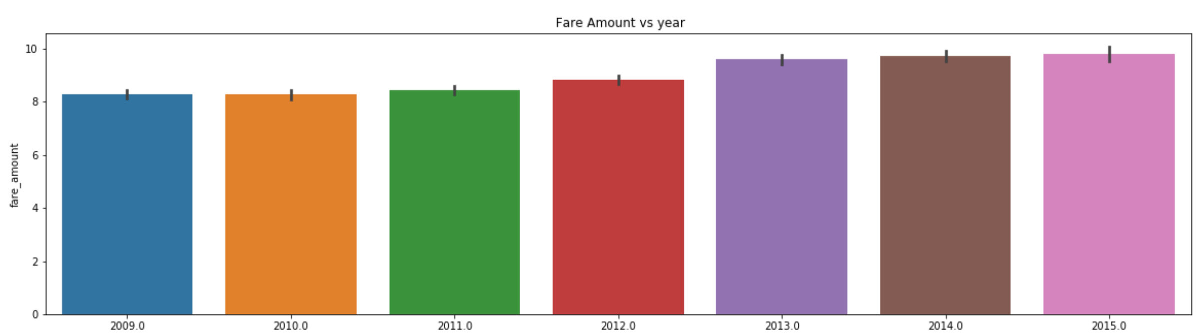
In [93]:
```
############################## Bar plot of fare amount vs Pickup month #######
##############################
plt.figure(figsize=(20,5))
sns.barplot(x='pickup_month',y='fare_amount',data=cab_train).set_title(" Fare
 Amount vs month")
```

Out[93]: Text(0.5, 1.0, ' Fare Amount vs month')



In [95]:
```
############################## Bar plot of fare amount vs Pickup year ########
##############################
plt.figure(figsize=(20,5))
sns.barplot(x='pickup_year',y='fare_amount',data=cab_train).set_title(" Fare A
mount vs year")
```

Out[95]: Text(0.5, 1.0, ' Fare Amount vs year')



In [96]:
```
############################## Bar plot of fare amount vs Pickup month days ##
##################################
plt.figure(figsize=(20,5))
sns.barplot(x='passenger_count',y='fare_amount',data=cab_train).set_title(" Fa
re Amount vs passenger_count")
```

Out[96]: Text(0.5, 1.0, ' Fare Amount vs month')