About the Project

The Project is **New York City Taxi Fare Prediction** where in you need to predict the taxi fares (*fare amount*) for the gi

```
from google.colab import drive
```

```
drive.mount('/content/drive', force_remount=True)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189

Enter your authorization code:
··········
Mounted at /content/drive
```

```
ls
```

```
drive/  sample_data/
```

```
cd drive
```

```
/content/drive
```

```
cd My/Drive
```

```
[Errno 2] No such file or directory: 'My/Drive'
/content/drive
```

```
cd My\ Drive
```

```
/content/drive/My Drive
```

```
cd New\ York\ Taxi\ Fare\ Project
```

```
/content/drive/My Drive/New York Taxi Fare Project
```

Let us start with first importing the required libraries

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
import seaborn as sea
import numpy as np
import datetime as dt
```

As the dataset given is about 55 million rows which takes up a lot of memory and processing time, we initially take l

Reading the data using pandas

```
data=pd.read_csv("train.csv",nrows=1000000)
test=pd.read_csv("test.csv")
```

Data Exploration and Cleaning

The following things which exploration and cleaning pertains are done:

- Datatype checks
- Shape checks
- Check for Nan values
- Check for outliers
- Cleaning the data pertaining to each attribute in the training set

```
data.shape
```

```
(1000000, 8)
```

Our cell contains 1M rows as we specified and 8 columns.

```
data.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lat
count	1000000.000000	1000000.000000	1000000.000000	999990.000000	999990.0
mean	11.348079	-72.526640	39.929008	-72.527860	39.9
std	9.822090	12.057937	7.626154	11.324494	8.2
min	-44.900000	-3377.680935	-3116.285383	-3383.296608	-3114.3
25%	6.000000	-73.992060	40.734965	-73.991385	40.7
50%	8.500000	-73.981792	40.752695	-73.980135	40.7
75%	12.500000	-73.967094	40.767154	-73.963654	40.7
max	500.000000	2522.271325	2621.628430	45.581619	1651.5

By calling the describe method we can get the overlay of the data in the columns.

Now let us check the data types of our attributes (columns)

```
data.dtypes
```

```
key                   object
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

Now let us check if all the colums have data or some have null, we call the ' *isnull( )* ' function for that

```
null_data=data.isnull().sum()
print(null_data)
```

```
key                   0
fare_amount           0
pickup_datetime       0
pickup_longitude      0
pickup_latitude       0
dropoff_longitude    10
dropoff_latitude     10
passenger_count       0
dtype: int64
```

Here we can see that there are 10 each missing values in dropoff_latitude and dropoff_longitude, we will drop those

```
data.shape
```

```
(1000000, 8)
```

```
data = data.dropna(how = 'any', axis = 'rows')
```

```
data.shape
```

```
(999990, 8)
```

As we can see, we have dropped 10 rows which means that there are missing values in both the columns concurren

Let us check each attributes for outliers and remove or append them with certain values

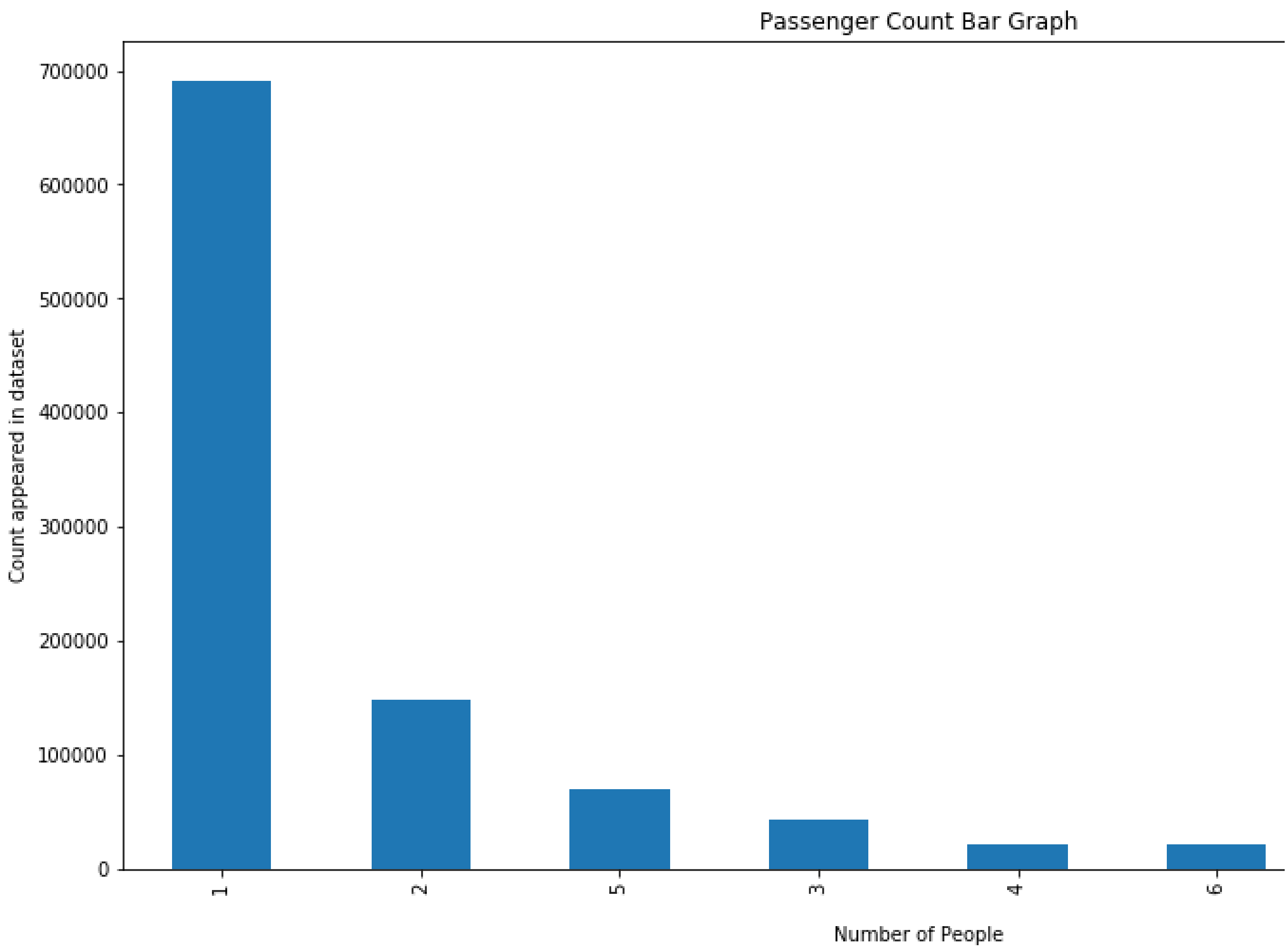
- First let us take a look at the ' passenger_count ' column

```
data['passenger_count'].describe()
```

```
count    999990.000000
mean          1.684941
std           1.323907
min           0.000000
25%           1.000000
50%           1.000000
75%           2.000000
max         208.000000
Name: passenger_count, dtype: float64
```

```
#To get a gist of the data in the column, we plot the data into a bar chart
plt.figure(figsize=(15,8))
data['passenger_count'].value_counts().plot.bar()
plt.xlabel('Number of People')
plt.ylabel('Count appeared in dataset')
plt.title('Passenger Count Bar Graph')
```

```
Text(0.5, 1.0, 'Passenger Count Bar Graph')
```



- As we can see the max value of ' passenger_count ' is about 208 which is quite absurd as even if we consider would be of around 60.
- ' passenger_count ' for 1 to 6 have a significant count, Let's try to find the rides which have more than 6 passe

```
data[data['passenger_count']>6]
```

	key	fare_amount	pickup_datetime	pickup_longitud
929022	2009-07-30 11:54:00.000000193	3.3	2009-07-30 11:54:00 UTC	0.

- We have two rows with 9 and 208, which are definetly outliers, so we remove these from our data.

```
data.shape
```

```
(999990, 8)
```

```
data=data[data['passenger_count']<=6]
```

```
data.shape
```

```
(999989, 8)
```

- Now we decribe the ' passenger_count ' column and check if the max passengers are 6.

```
data['passenger_count'].describe()
```

```
count    999989.000000
mean          1.684735
std           1.307733
min           0.000000
25%           1.000000
50%           1.000000
75%           2.000000
max           6.000000
Name: passenger_count, dtype: float64
```

Note:

- We have small amount of rides with passengers 0 too, this might be the case of cancelling the ride after the t
  drop this data as these might have potential information.
- We will deal with this case when we are cleaning latitude and longitude data

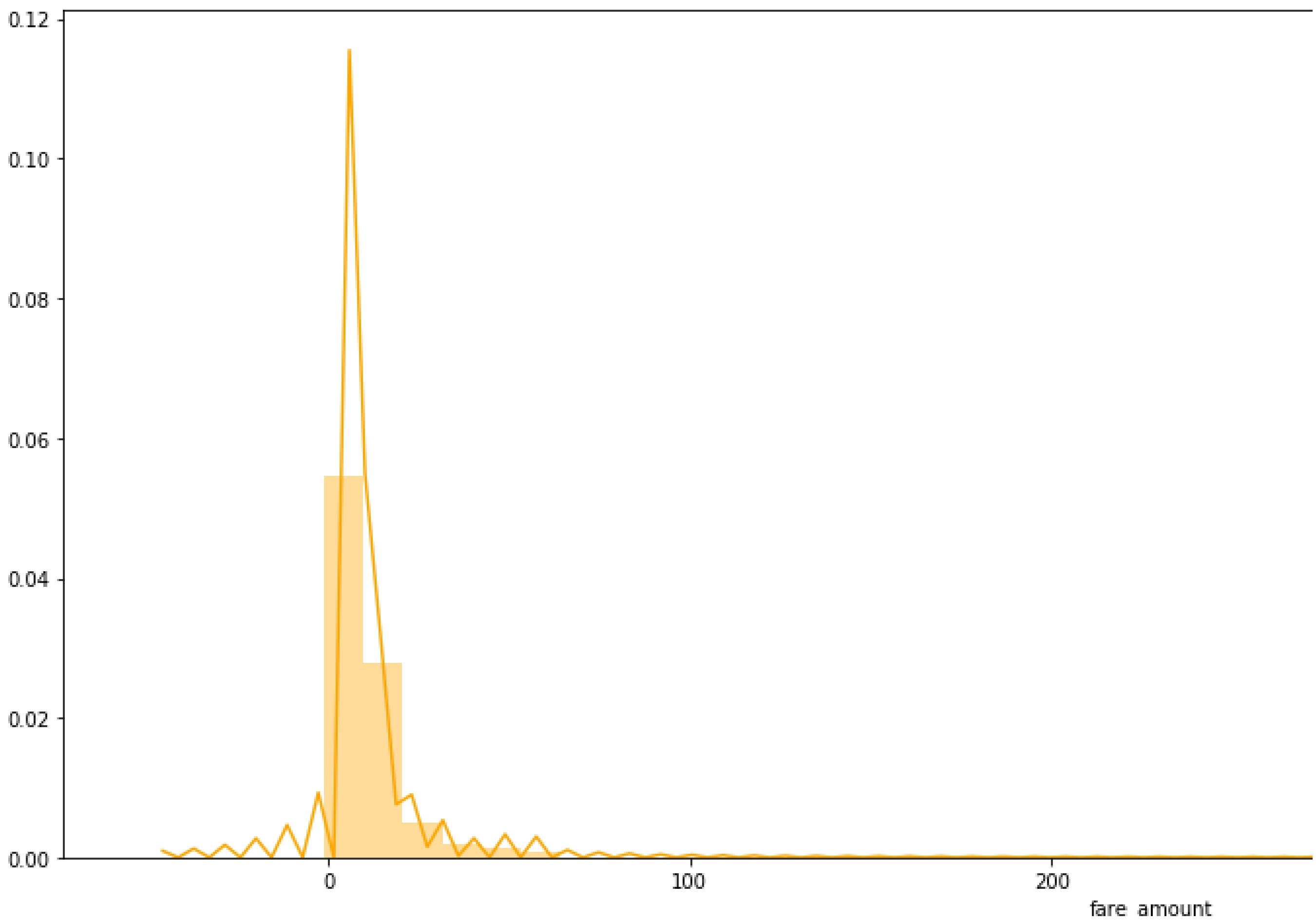
- Now let us take a look at the ' fare_amount ' column

```
data['fare_amount'].describe()
```

```
count    999989.000000
mean         11.347961
std           9.821791
min         -44.900000
25%           6.000000
50%           8.500000
75%          12.500000
max         500.000000
Name: fare_amount, dtype: float64
```

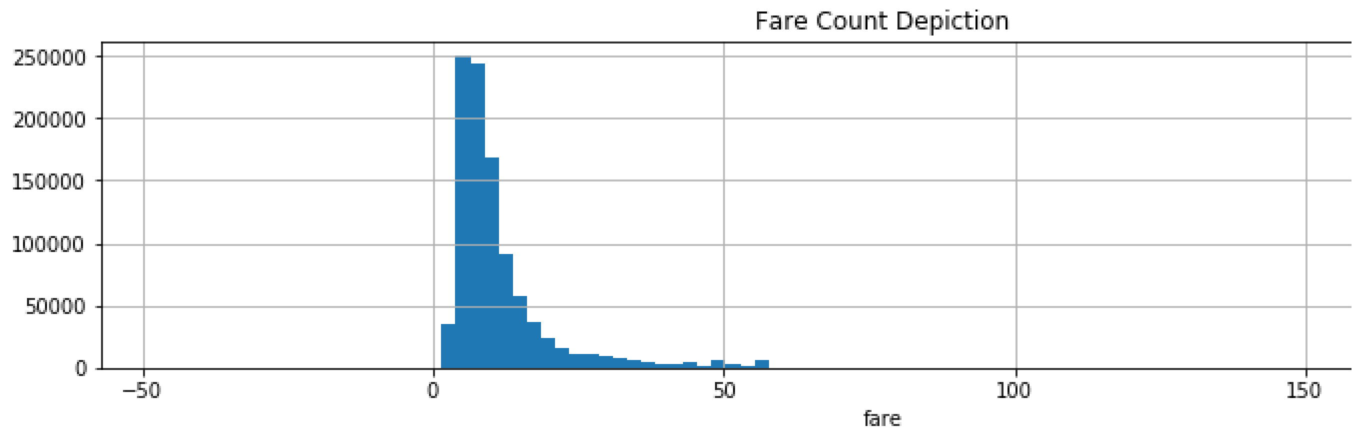
```
#Let us plot a graph to get a gist of the column
plt.figure(figsize=(20,8))
sea.distplot(data.fare_amount,color="orange")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbeb586cba8>
```



- As we can see the ' fare_amount ' is not normally distributed
- A lot of fares agglomerate near the range of [0 , 80]
- the max fare is about a whooping more than $1200

```
data[data.fare_amount<200].fare_amount.hist(bins=100, figsize=(14,3))
plt.xlabel('fare')
plt.title('Fare Count Depiction');
```




```
fare_less_than_zero=data[data['fare_amount']<0]
```

```
fare_less_than_zero.shape
```

```
(38, 8)
```

- We have 38 rows with fare less than zero, we drop them as there cannot be any fare less than zero

```
data=data[data['fare_amount']>=0]
```

```
data.shape
```

```
(999951, 8)
```

```
data['fare_amount'].sort_values(ascending=False)
```

```
101885    500.00
247671    495.00
287638    450.00
233874    450.00
329010    450.00
451974    400.00
361793    400.00
951810    347.54
578919    287.08
130460    281.05
309769    263.25
719764    262.04
142550    255.00
888472    250.25
351584    250.00
217225    245.41
786490    243.00
149769    240.00
168218    235.00
202499    225.00
612128    220.00
806692    220.00
196990    217.00
784935    215.00
225249    215.00
416989    215.00
285659    214.33
110337    212.00
215662    211.44
979151    210.00
            ...
2780        0.01
211499      0.00
942215      0.00
211455      0.00
27891       0.00
957590      0.00
105051      0.00
47302       0.00
895361      0.00
436658      0.00
175352      0.00
930680      0.00
760662      0.00
788466      0.00
266485      0.00
495273      0.00
938020      0.00
681342      0.00
10002       0.00
561786      0.00
386734      0.00
762802      0.00
949564      0.00
689250      0.00
897211      0.00
489767      0.00
```

```
331597      0.00
520715      0.00
431819      0.00
670254      0.00
Name: fare_amount, Length: 999951, dtype: float64
```

Now let us consider the pickup and dropoff longitudes and latitudes

```
data.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati
count	999951.000000	999951.000000	999951.000000	999951.000000	999951.00
mean	11.348624	-72.526792	39.929090	-72.528173	39.92
std	9.821251	12.057574	7.626025	11.323551	8.20
min	0.000000	-3377.680935	-3116.285383	-3383.296608	-3114.33
25%	6.000000	-73.992060	40.734965	-73.991385	40.73
50%	8.500000	-73.981792	40.752695	-73.980135	40.75
75%	12.500000	-73.967095	40.767154	-73.963654	40.76
max	500.000000	2522.271325	2621.628430	45.581619	1651.55

- As we check the data we have min values ranging around -3400 which is not defined and these pertain to outl
- The max values also have a certain outliers being max values around 3000.

- Now what we try to do is that we check the min and max values of latitudes and longitudes of the test data ar max] range.
- Let us find the min and max of the test data.

```
test.describe()
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger
count	9914.000000	9914.000000	9914.000000	9914.000000	9914
mean	-73.974722	40.751041	-73.973657	40.751743	1
std	0.042774	0.033541	0.039072	0.035435	1
min	-74.252193	40.573143	-74.263242	40.568973	1
25%	-73.992501	40.736125	-73.991247	40.735254	1
50%	-73.982326	40.753051	-73.980015	40.754065	1
75%	-73.968013	40.767113	-73.964059	40.768757	2
max	-72.986532	41.709555	-72.990963	41.696683	6

```
min(test.pickup_longitude.min(), test.dropoff_longitude.min()), \
max(test.pickup_longitude.max(), test.dropoff_longitude.max())
```

```
(-74.263242, -72.986532)
```

```
min(test.pickup_latitude.min(), test.dropoff_latitude.min()), \
max(test.pickup_latitude.max(), test.dropoff_latitude.max())
```

```
(40.568973, 41.709555)
```

- Now we got the extremes of the test data of long' and lat', we now restrict our train data rows to this range.
- We use Dataframe.loc to access the group with the limits we put and only keep them in data.

```
data = data.loc[data['pickup_longitude'].between(-74.3, -72.9)]
data = data.loc[data['dropoff_longitude'].between(-74.3, -72.9)]
data = data.loc[data['dropoff_latitude'].between(40.5, 41.7)]
data = data.loc[data['pickup_latitude'].between(40.5, 41.7)]
```

```
data.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati
count	978953.000000	978953.000000	978953.000000	978953.000000	978953.00
mean	11.325728	-73.975085	40.751116	-73.974191	40.75
std	9.689830	0.038426	0.029531	0.037559	0.03
min	0.000000	-74.299372	40.503982	-74.299372	40.50
25%	6.000000	-73.992268	40.736583	-73.991569	40.73
50%	8.500000	-73.982082	40.753417	-73.980590	40.75
75%	12.500000	-73.968313	40.767582	-73.965321	40.76
max	500.000000	-72.940862	41.696852	-72.900000	41.64

```
data.dtypes
```

```
key                    object
fare_amount           float64
pickup_datetime        object
pickup_longitude      float64
pickup_latitude       float64
dropoff_longitude     float64
dropoff_latitude      float64
passenger_count         int64
dtype: object
```

- The pickup_datetime is in ' object ' type, we convert it into datetime format so as to parse various day functio

```
def convert_to_datetime(df):
    test_time = df['pickup_datetime'].astype(str).str[:-4]
    df['date_time'] =  pd.to_datetime(test_time, format='%Y%m%d %H:%M:%S')
    return df

data = convert_to_datetime(data)
test = convert_to_datetime(test)
```

```
data.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati
count	978953.000000	978953.000000	978953.000000	978953.000000	978953.00
mean	11.325728	-73.975085	40.751116	-73.974191	40.75
std	9.689830	0.038426	0.029531	0.037559	0.03
min	0.000000	-74.299372	40.503982	-74.299372	40.50
25%	6.000000	-73.992268	40.736583	-73.991569	40.73
50%	8.500000	-73.982082	40.753417	-73.980590	40.75
75%	12.500000	-73.968313	40.767582	-73.965321	40.76
max	500.000000	-72.940862	41.696852	-72.900000	41.64

```
def extract_date(data):
    data['hour'] = data['date_time'].dt.hour
    data['day'] = data['date_time'].dt.day
    data['month'] = data['date_time'].dt.month
    data['year'] = data['date_time'].dt.year
    data['day_of_week'] = data['date_time'].dt.weekday
   # data = data.drop(['date_time','pickup_datetime'], axis=1)
    return data

data = extract_date(data)
test = extract_date(test)
```

```
data.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati
count	978953.000000	978953.000000	978953.000000	978953.000000	978953.00
mean	11.325728	-73.975085	40.751116	-73.974191	40.75
std	9.689830	0.038426	0.029531	0.037559	0.03
min	0.000000	-74.299372	40.503982	-74.299372	40.50
25%	6.000000	-73.992268	40.736583	-73.991569	40.73
50%	8.500000	-73.982082	40.753417	-73.980590	40.75
75%	12.500000	-73.968313	40.767582	-73.965321	40.76
max	500.000000	-72.940862	41.696852	-72.900000	41.64

```
test.describe()
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger
count	9914.000000	9914.000000	9914.000000	9914.000000	9914
mean	-73.974722	40.751041	-73.973657	40.751743	1
std	0.042774	0.033541	0.039072	0.035435	1
min	-74.252193	40.573143	-74.263242	40.568973	1
25%	-73.992501	40.736125	-73.991247	40.735254	1
50%	-73.982326	40.753051	-73.980015	40.754065	1
75%	-73.968013	40.767113	-73.964059	40.768757	2
max	-72.986532	41.709555	-72.990963	41.696683	6

Distance Calculation

- Let us calculate the distance as it serves as a key element in determining the fare of the trip.
- Now we have pickup and dropoff longitudes, latitudes to calculate the distance of the trip.
- We try to use haversine distance which is a slight tweek to euclidean distance considering the cosine angle o equator and descreases as we approach noth pole.
- The reason for this consideration is that, it might be simple to use euclidean on a high level but the distance b is a lot.
- As distance is key feature in training our model, we have to be precise.
- There is a specific method in ' geopy ' library for distance calculation but takes too much of time.

```
#https://stackoverflow.com/questions/19412462/getting-distance-between-two-points-based-on-latitude-
def distance(lat1, long1, lat2, long2):
    x = [data, test]
    for i in x:
        #R = 6371  #radius of earth in kilometers
        R = 3959 #radius of earth in miles
        phi1 = np.radians(i[lat1])
        phi2 = np.radians(i[lat2])

        delta_phi = np.radians(i[lat2]-i[lat1])
        delta_lambda = np.radians(i[long2]-i[long1])

        #a = sin²((φB - φA)/2) + cos φA . cos φB . sin²((λB - λA)/2)
        a = np.sin(delta_phi / 2.0) ** 2 + np.cos(phi1) * np.cos(phi2) * np.sin(delta_lambda / 2.0)

        #c = 2 * atan2( √a, √(1-a) )
        c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))

        #d = R*c
        d = (R * c) #in miles
        i['distance'] = d
    return d
```

```
distance('pickup_latitude', 'pickup_longitude', 'dropoff_latitude', 'dropoff_longitude')
```

```
0        1.443696
1        1.507137
2        0.384421
3        1.218604
4        3.347720
5        2.002523
6        0.577663
7       13.385224
8        2.407317
9        0.683422
10       1.440232
11       2.994683
12       0.449074
13       1.041174
14       1.556030
15       3.178743
16       0.185701
17       1.572879
18       0.484976
19       0.265517
20       1.026535
21       1.223686
22       0.811214
23       1.181482
24       3.591573
25       0.697027
26       8.890220
27       5.858568
28       0.711221
29       2.918369
           ...
9884     9.395273
9885     3.110925
9886     4.583985
9887     1.044416
9888     0.000000
9889     1.477539
9890     1.829791
9891     3.699151
9892     1.833382
9893     2.942607
9894     1.927215
9895     5.592887
9896     1.490240
9897     3.157794
9898     0.345060
9899     3.486422
9900     1.174842
9901     0.950830
9902     2.418231
9903     1.428817
9904     9.377854
9905     5.974105
9906     0.229615
9907     6.166867
9908     1.016876
9909     1.320417
```

```
9910     2.032611
9911    11.921084
9912     5.184722
9913     0.733776
Length: 9914, dtype: float64
```

```
data.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati
count	978953.000000	978953.000000	978953.000000	978953.000000	978953.00
mean	11.325728	-73.975085	40.751116	-73.974191	40.75
std	9.689830	0.038426	0.029531	0.037559	0.03
min	0.000000	-74.299372	40.503982	-74.299372	40.50
25%	6.000000	-73.992268	40.736583	-73.991569	40.73
50%	8.500000	-73.982082	40.753417	-73.980590	40.75
75%	12.500000	-73.968313	40.767582	-73.965321	40.76
max	500.000000	-72.940862	41.696852	-72.900000	41.64

```
test.describe()
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger
count	9914.000000	9914.000000	9914.000000	9914.000000	9914
mean	-73.974722	40.751041	-73.973657	40.751743	1
std	0.042774	0.033541	0.039072	0.035435	1
min	-74.252193	40.573143	-74.263242	40.568973	1
25%	-73.992501	40.736125	-73.991247	40.735254	1
50%	-73.982326	40.753051	-73.980015	40.754065	1
75%	-73.968013	40.767113	-73.964059	40.768757	2
max	-72.986532	41.709555	-72.990963	41.696683	6

Pearson Correlation

- Distance of the ride and the taxi fare
- Time of day and distance traveled
- Time of day and the taxi fare

```
#importing the scipy library for finding pearson correlation directly.
import scipy.stats as stats
```

- We use the ' stats.pearsonr ' to calculate the pearson coefficient.
- pearsonr() returns a two-tuple consisting of the correlation coefficient and the corresponding p-value.
- The correlation coefficient can range from -1 to +1.
- The null hypothesis is that the two variables are uncorrelated. The p-value is a number between zero and one would have arisen if the null hypothesis were true.

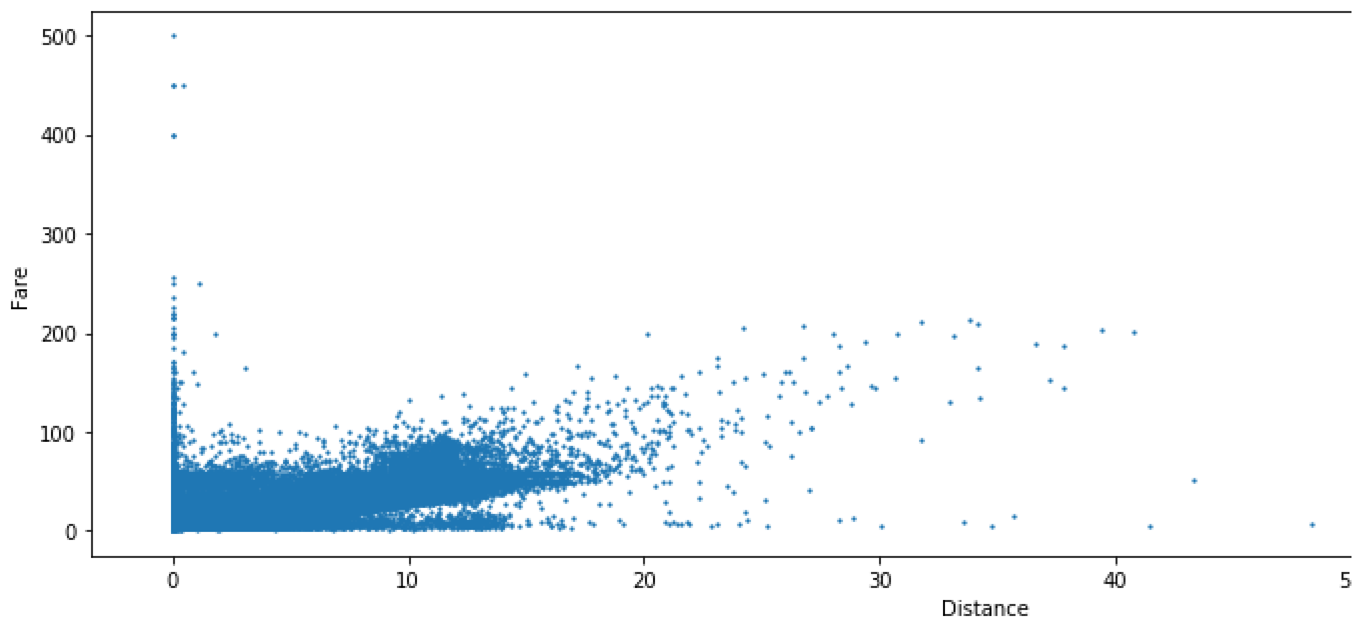
```
stats.pearsonr(data['distance'], data['fare_amount'])
```

```
(0.8169228364017578, 0.0)
```

Let us plot a visual representation of these features.

```
plt.figure(figsize=(16,5))
plt.scatter(x=data['distance'], y=data['fare_amount'], s=1.5)
plt.xlabel('Distance')
plt.ylabel('Fare')
```

```
Text(0, 0.5, 'Fare')
```



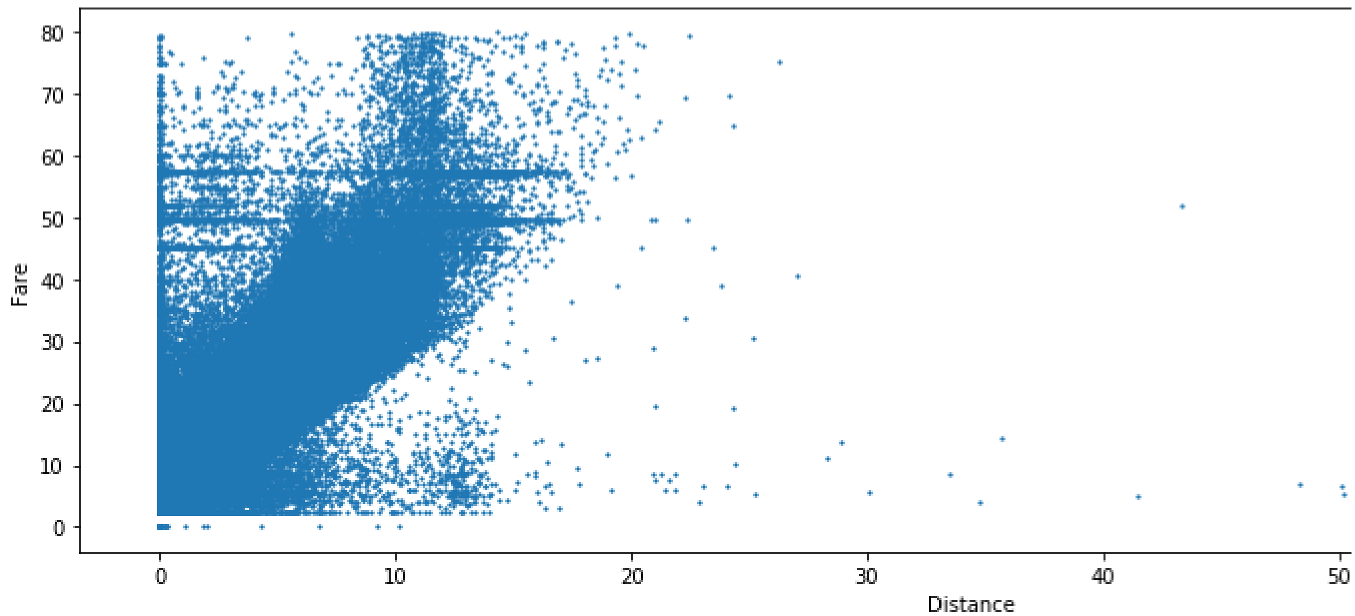
From the plot it is clearly evident that: ***Linear Relation***

- There are trips with zero distance but with a non-zero fare. How can a trip end at the same location of pickup? pickup location mistakely or something else
- There are quite a few trips with distances of ~60 miles but the fare is pretty less. There might be different rea applied discount coupon', 'outstation trips which cost less for more miles we travel as fare/mile ratio will be d
- Overall, we generically say that there is a *linear relationship* between distance and fare amount since pearsonı related) and the plot evidently checks out as a linear relationship.
- Although there might be a vague discussion of the relationship of the right side fewer dots but generically we

The scatterplot in the distance range of [0 , 20] is not clear, let us try to limit the distance and fare values so as to g

```
rep_data=data[data['distance'] < 30]
rep_data=data[data['fare_amount'] < 80]
plt.figure(figsize=(16,5))
plt.scatter(x= rep_data['distance'], y=rep_data['fare_amount'], s=1.5)
plt.xlabel('Distance')
plt.ylabel('Fare')
```

```
Text(0, 0.5, 'Fare')
```



Discovery:

- When we observe carefully we can see three lines horizontally near fare ~45, ~50, ~57 for a distance of ~18 r
- These might be the fixed taxi prices for airports as the airport transfer will be having a fixed price relieving the

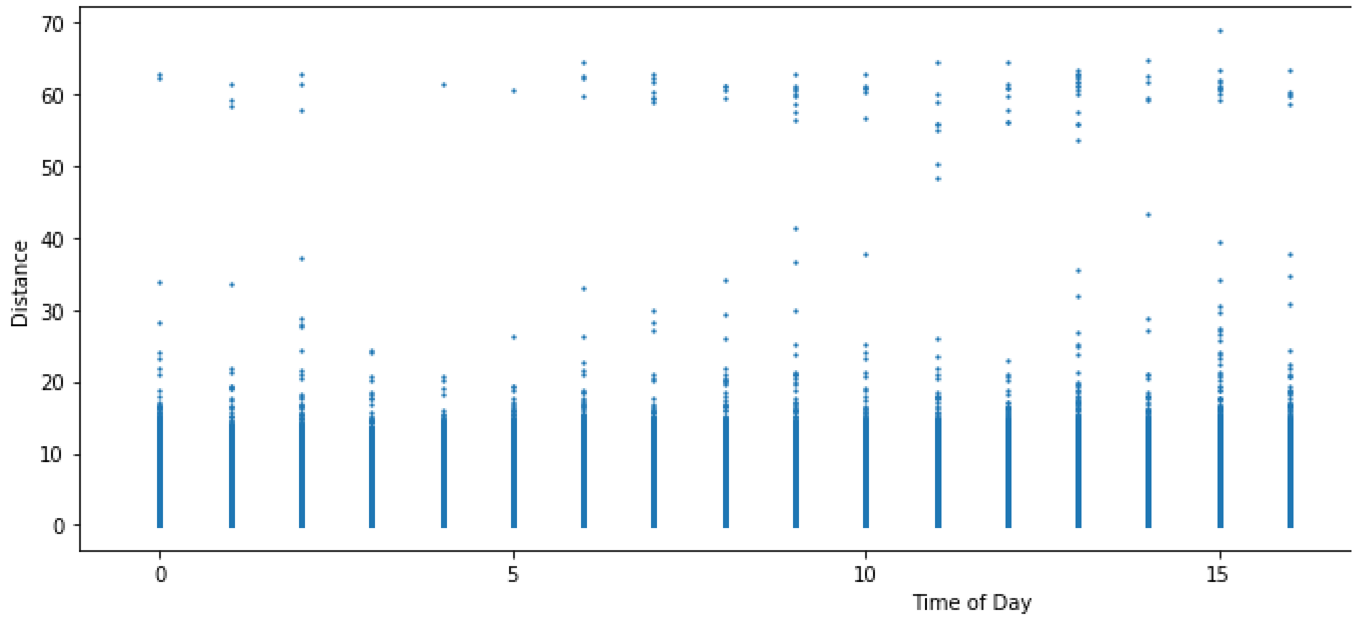
```
stats.pearsonr(data['hour'], data['distance'])
```

```
(-0.030101850860392312, 5.248414374040616e-195)
```

Let us plot a visual representation of these features.

```
plt.figure(figsize=(16,5))
plt.scatter(x=data['hour'], y=data['distance'], s=1.5)
plt.xlabel('Time of Day')
plt.ylabel('Distance')
```

```
Text(0, 0.5, 'Distance')
```



From the plot it is evident that: Non Linear

- The relationship is vague and cannot be accurately determined as we cannot clearly depict it is a linear relatic
- We can observe that people travelled less distance between 3 AM to 5 AM. This is obvious because most pec
- People travelled more distances during 9 AM to 10 AM in the morning and 3PM to 5PM. This can be justified l office and might hail a cab from long distances to compensate the time. In the evening people might leave ea feel tired to travel.
- If we limit the distance to < 40 miles we can say it will be a non- linear relationship as the graph does not fit in

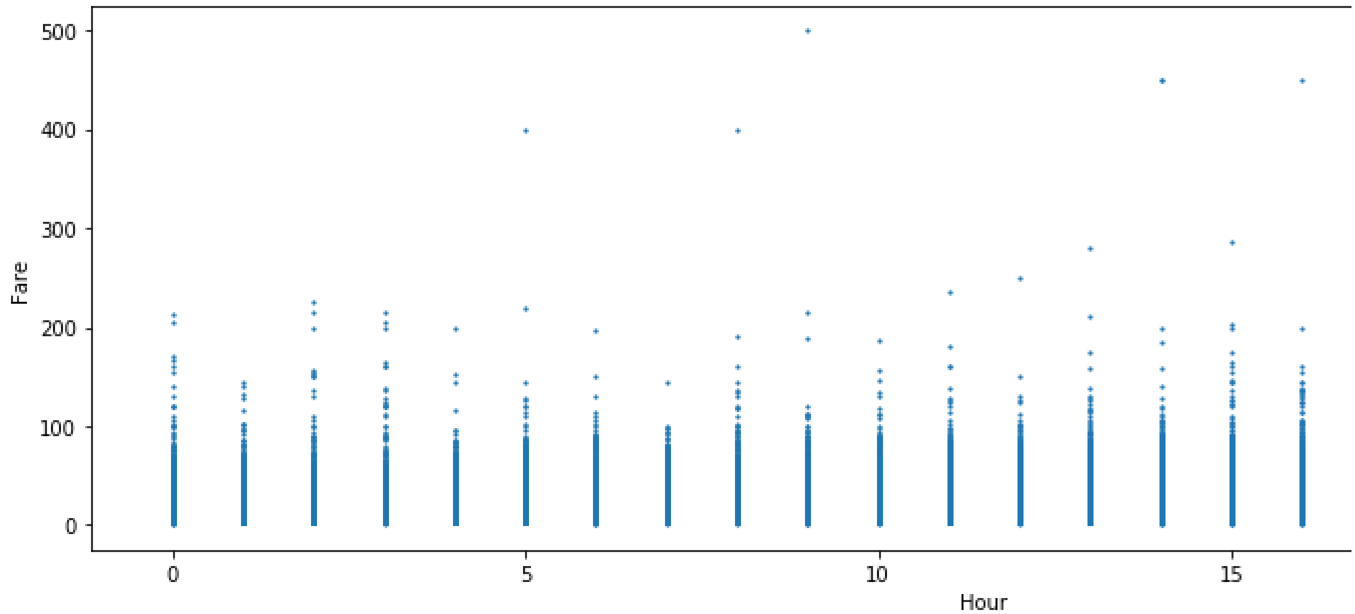
```
stats.pearsonr(data['hour'], data['fare_amount'])
```

```
(-0.019295638779025792, 2.873013197646852e-81)
```

Let us plot a visual representation of these features.

```
plt.figure(figsize=(16,5))
plt.scatter(x=data['hour'], y=data['fare_amount'], s=1.5)
plt.xlabel('Hour')
plt.ylabel('Fare')
```

```
Text(0, 0.5, 'Fare')
```



From the plot it is evident that: Non Linear

- The relationship is vague and cannot be accurately determined as we cannot clearly depict it is a linear relatic
- We can observe that the fares at 12AM, 5AM to 10AM and 2PM to 5PM, 8PM are high.
- This can be the reason of surge in the area, like at 12AM it might be a friday or saturday night people might gc cabs were at a surge due to unavailability and they book it anyway as they might be wasted!!!
- 5AM to 10AM, 2PM to 5PM and 8PM, the new york taxis will be on a definte surge as these are the time range

Highest Correlation : ' Distance ' and ' Fare Amount '

- The correlation between distance and fare_amount is the highest about 0.82 (pearson coefficient).
- This is clearly obvious as the fare amount is and will always be dependent on distance travelled.
- Basic formula for fare calculation is,

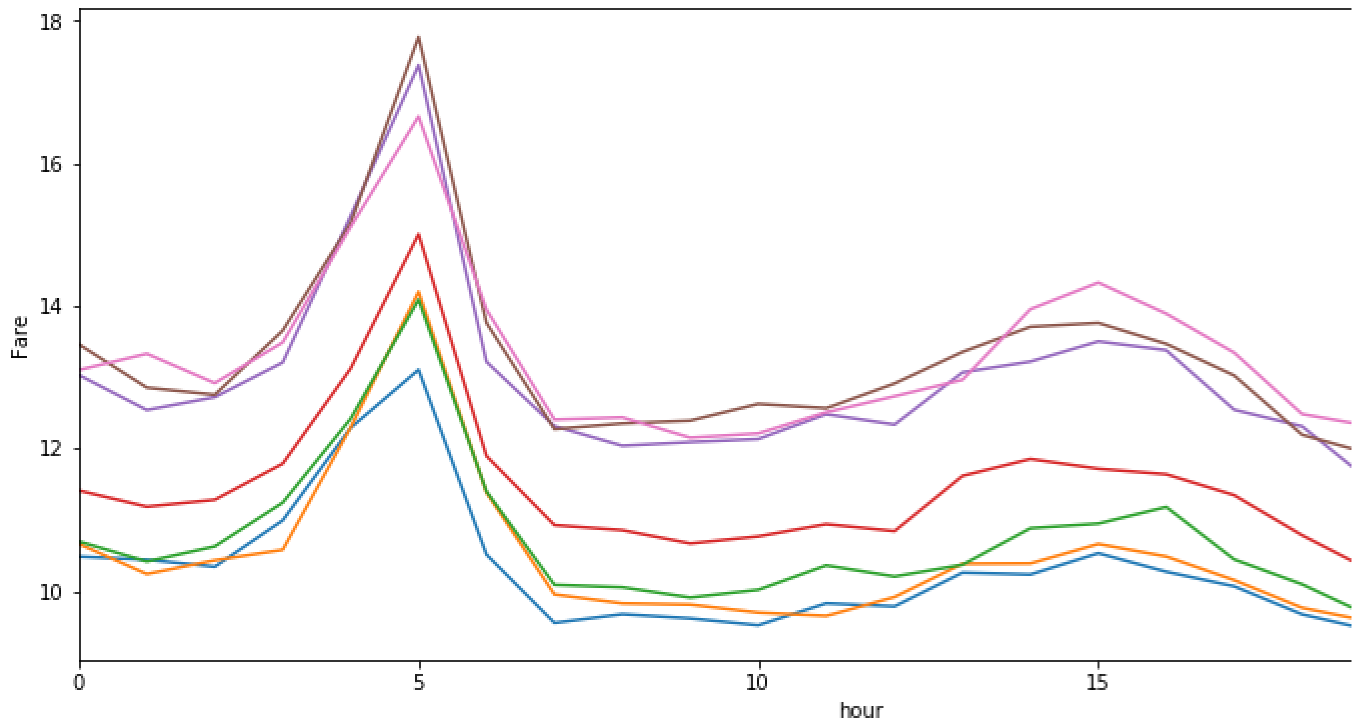
```
fare amount = ( base fare ) + ( distance ) * (generalized rate / 1 mile)
```

- The fare also depends on other metrics like tolls, traffic waiting time and few other things.

Let us try to analyze the the feature dependencies by exploring the metrics and plotting them to find a trend.

- Let us try to plot the 'fare amount' for a 'day' for all the 'years' in the training data.

```
data.pivot_table('fare_amount', index='hour', columns='year').plot(figsize=(14,6))
plt.ylabel('Fare');
```



From the plot it is evident that:

- There is a huge increare in the taxi fare rates from the year 2009 to 2015.
- The peak time (in this plot ~5AM and 2PM to 5PM) the plotting lines over the years overlap due to surge prici pretty consistent.
- This will cause a problem in predicting the fare amounts for the test data as in the fare amount will be incons cab is taken.
- We either need to model using each year seperately or group the years which have a less gradient or add a litt in the model for more accurate predictions.

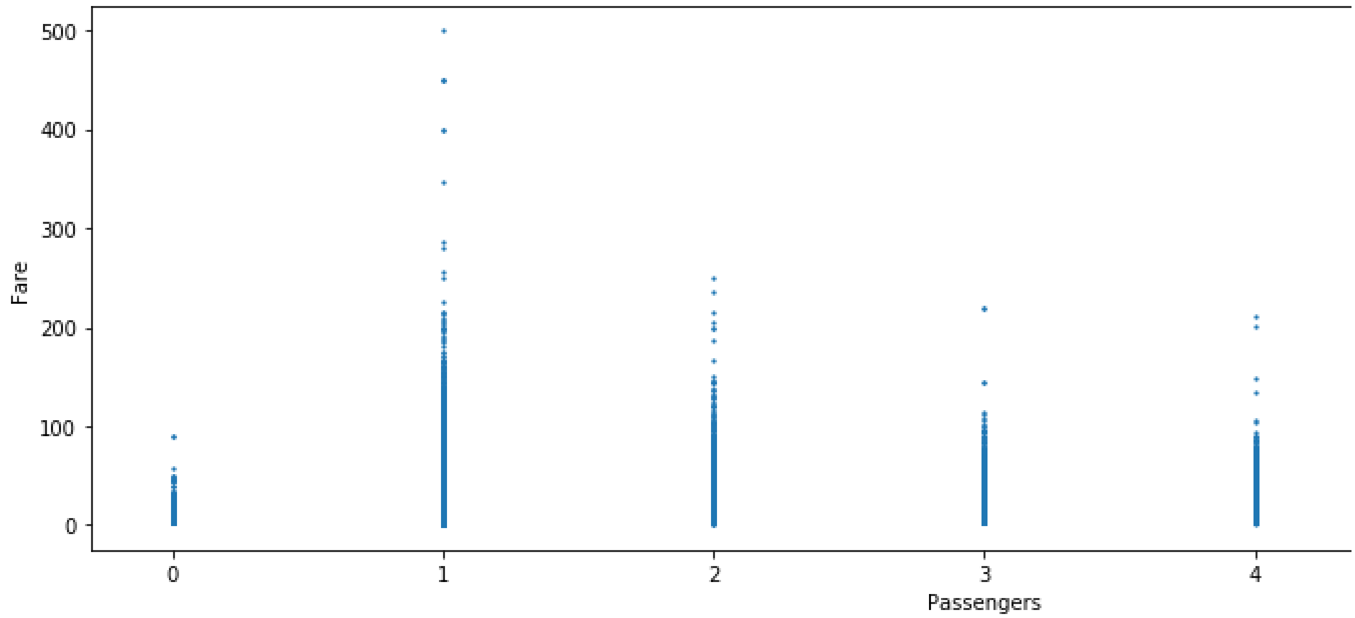
```
skew_data = data.skew()
plt.figure(figsize=(20,8))
sea.distplot(skew_data, bins=10, kde=True, color="orange")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbeb56b2be0>
```

- From the above graph we can observe that there is slight bit of positive skew in our training data i.e the mean the right.

```
plt.figure(figsize=(16,5))
plt.scatter(x=data['passenger_count'], y=data['fare_amount'], s=1.5)
plt.xlabel('Passengers')
plt.ylabel('Fare')
```

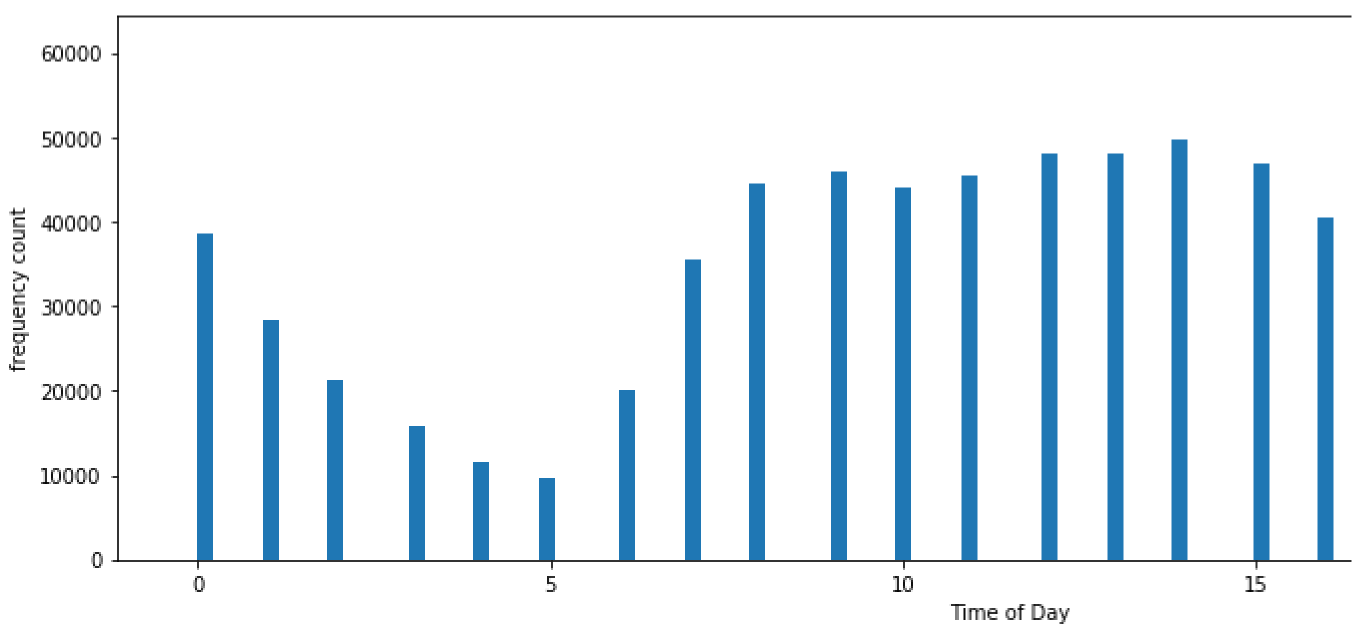
```
Text(0, 0.5, 'Fare')
```



- This is the graph plotted between the fare_amount and the passenger_count to get a gist of the relation.
- We cannot identify any specific trend in the plotting but we can see that single passengers are the ones who t the taxi with single passenger.

```
plt.figure(figsize=(16,5))
plt.hist(data['hour'], bins=100)
plt.xlabel('Time of Day')
plt.ylabel('frequency count')
```

```
Text(0, 0.5, 'frequency count')
```



- This is the plot for the time of day frequency count in our training dataset.

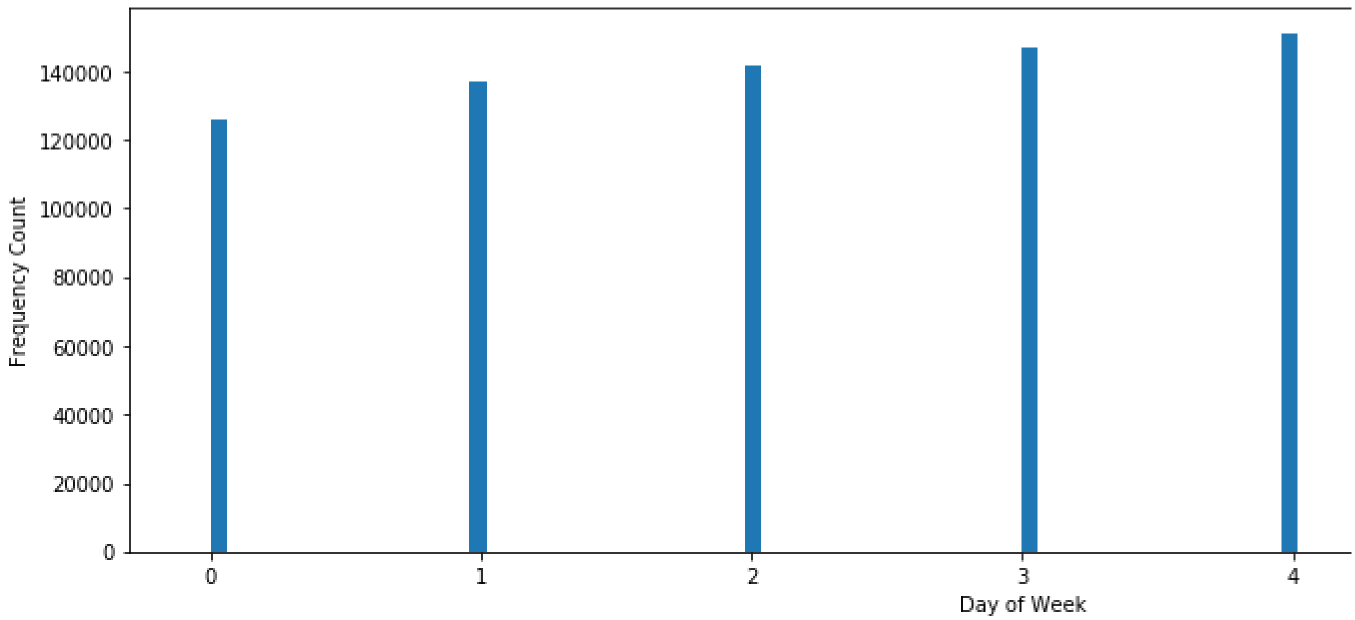
- In the previous plot between time of day and fare amount we observed the fare is high during peak period anc unavailability.
- Now the above graph justifies our assumption of surge pricing, as we can see that the frequency of taxis take from 6PM and peaks at 8PM.
- The taxis requested are more and thus the surge as the taxis will be less further on.

Let us create a plot for day_of_week metric:

- This is from the assumption that most of the taxi companies have different fares for weekdays and on the we

```
plt.figure(figsize=(16,5))
plt.hist(data['day_of_week'], bins=100)
plt.xlabel('Day of Week')
plt.ylabel('Frequency Count')
```

```
Text(0, 0.5, 'Frequency Count')
```



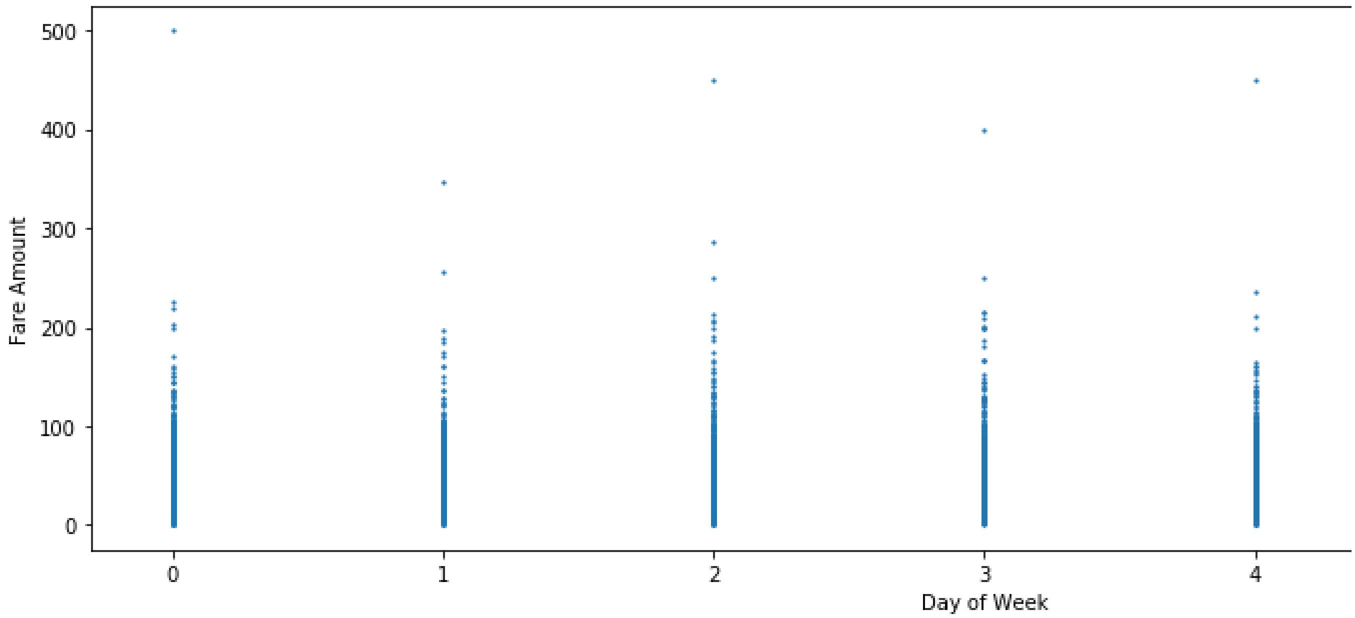

```
0 - Sunday ; 1 - Monday ; 2 - Tuesday ; 3 - Wednesday ; 4 - Thursday ; 5 - Friday
```

- From the above graph we can see that there is not clear cut assumption we can presume, rather that Thursda Sunday, Saturday there are less, may be people just taking rest for the weekend due to hectic work or not trav

- Now let us plot the day_of_week with the fare_amount, let's see we can find something.

```
plt.figure(figsize=(16,5))
plt.scatter(x=data['day_of_week'], y=data['fare_amount'], s=1.5)
plt.xlabel('Day of Week')
plt.ylabel('Fare Amount')
```

```
Text(0, 0.5, 'Fare Amount')
```



- The fares on Sunday and Wednesday seem to be highest and are the least on Monday.

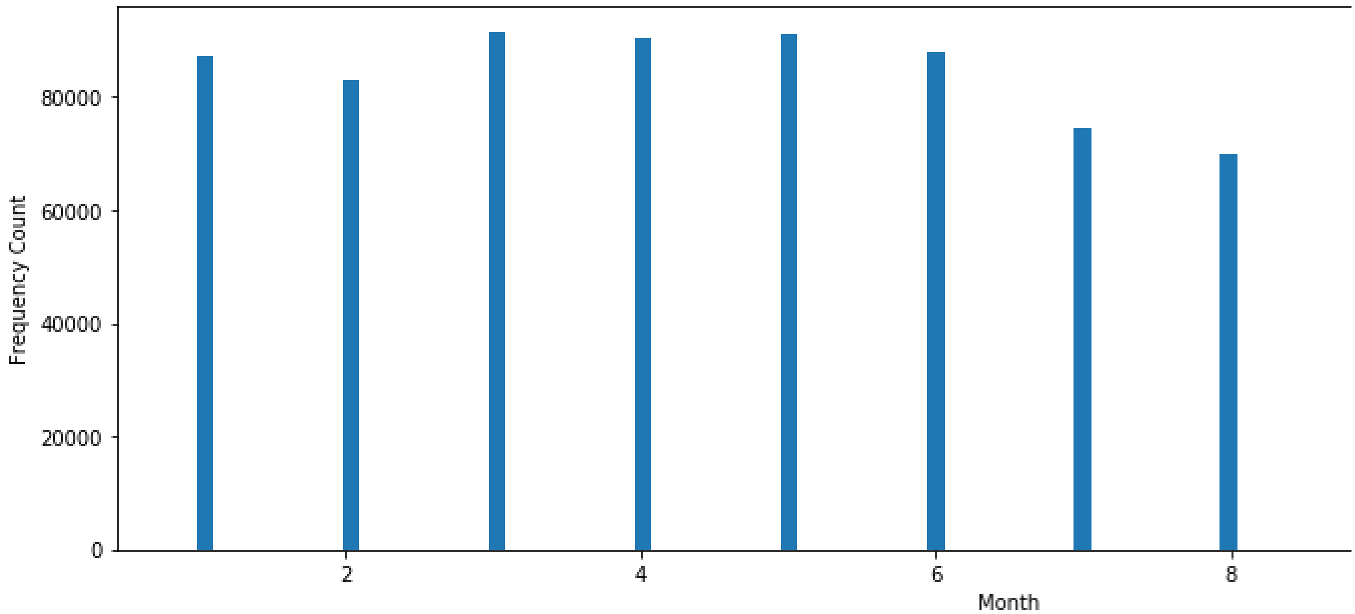
```
data.columns
```

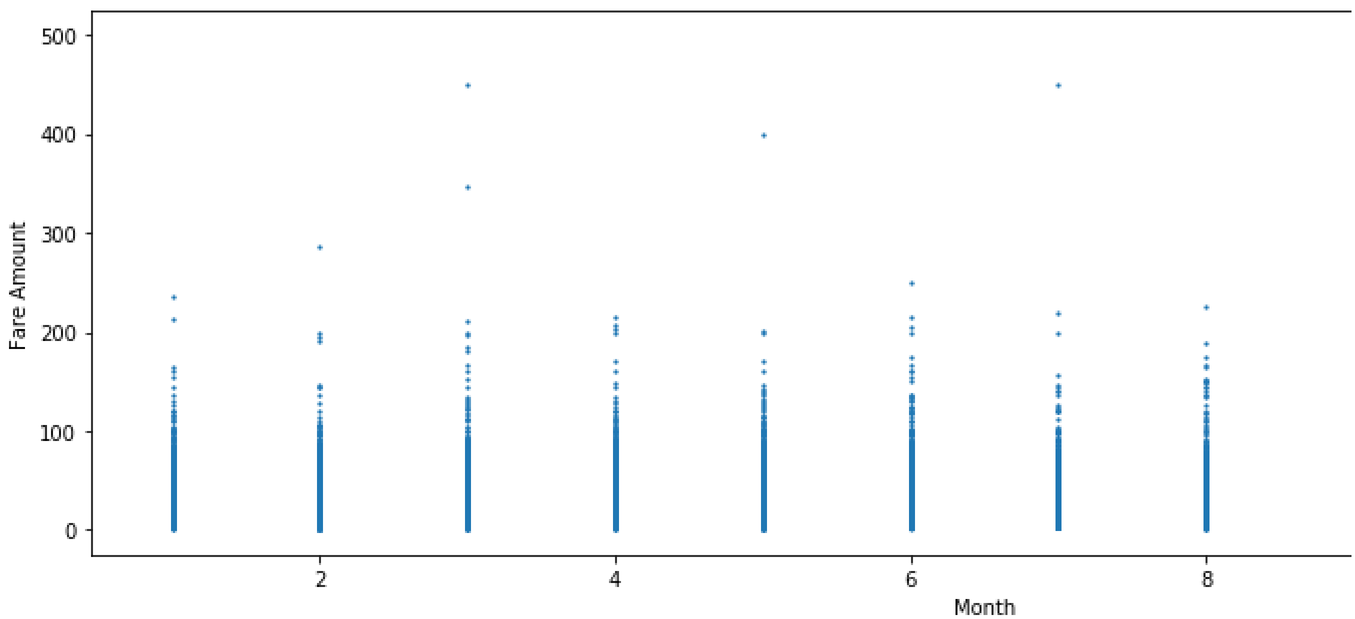
```
Index(['key', 'fare_amount', 'pickup_datetime', 'pickup_longitude',
       'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude',
       'passenger_count', 'date_time', 'hour', 'day', 'month', 'year',
       'day_of_week', 'distance'],
      dtype='object')
```

```
# Frequency count of the taxi rides in specific months
plt.figure(figsize=(16,5))
plt.hist(data['month'], bins=100)
plt.xlabel('Month')
plt.ylabel('Frequency Count')

# Plot of taxis fare amounts to the respective months
plt.figure(figsize=(16,5))
plt.scatter(x=data['month'], y=data['fare_amount'], s=1.5)
plt.xlabel('Month')
plt.ylabel('Fare Amount')
```

```
Text(0, 0.5, 'Fare Amount')
```





- The above graphs are plotted to get a sense or gist of the seasonal pricing.
- But unfortunately we cannot detect any trends in the fare amounts categorical to month.
- We can find the taxi fares to be reaching maximun in the month of September, may be due to a long weekend

Let us calculate absolute difference between latitudes and longitudes

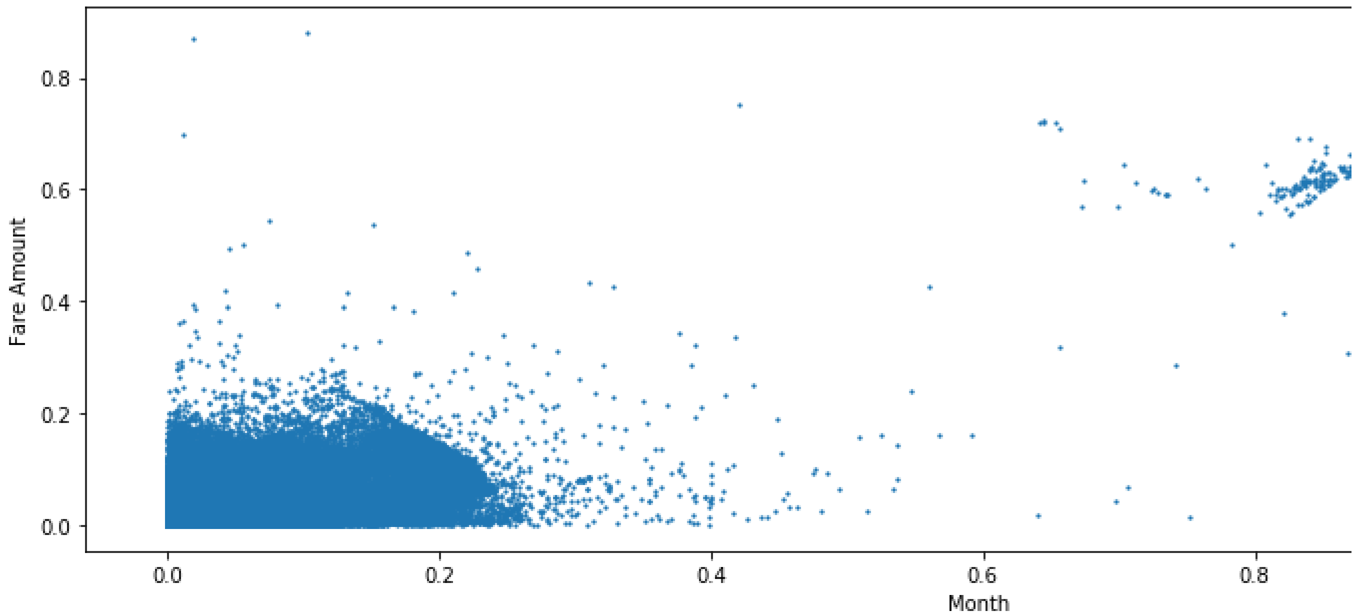
- This is because we have distance=0 but fare is not 0, which is absurd

```
data['abs_diff_longitude'] = (data['pickup_longitude'] - data['dropoff_longitude']).abs()
data['abs_diff_latitude'] = (data['pickup_latitude'] - data['dropoff_latitude']).abs()

test['abs_diff_longitude'] = (test['pickup_longitude'] - test['dropoff_longitude']).abs()
test['abs_diff_latitude'] = (test['pickup_latitude'] - test['dropoff_latitude']).abs()
```

```
plt.figure(figsize=(16,5))
plt.scatter(x=data['abs_diff_longitude'], y=data['abs_diff_latitude'], s=1.5)
plt.xlabel('Month')
plt.ylabel('Fare Amount')
```

```
Text(0, 0.5, 'Fare Amount')
```




```
stats.pearsonr(data['abs_diff_longitude'], data['fare_amount'])
```

```
(0.7720957909438255, 0.0)
```

```
stats.pearsonr(data['abs_diff_latitude'], data['fare_amount'])
```

```
(0.6414606842096722, 0.0)
```

```
dist_zero=data[data['distance']==0]
print(dist_zero)
```

```
                              key  ...  abs_diff_latitude
105       2009-03-25 00:08:52.0000001  ...                0.0
191       2014-01-08 21:55:58.0000006  ...                0.0
270       2012-08-25 01:53:42.0000005  ...                0.0
290      2009-12-14 12:33:00.00000075  ...                0.0
396       2014-03-12 18:12:44.0000006  ...                0.0
503      2010-01-19 01:10:00.00000012  ...                0.0
657       2009-08-25 01:50:21.0000001  ...                0.0
737     2014-08-11 19:37:00.000000174  ...                0.0
808       2010-10-22 02:24:53.0000001  ...                0.0
843      2014-07-19 04:42:00.00000041  ...                0.0
1124      2010-10-14 21:12:17.0000004  ...                0.0
1214      2011-08-16 07:04:05.0000001  ...                0.0
1222    2011-06-22 16:02:00.000000186  ...                0.0
1245      2010-04-01 13:14:41.0000001  ...                0.0
1265      2013-03-25 10:52:26.0000004  ...                0.0
1419      2013-07-21 00:10:23.0000001  ...                0.0
1630      2013-07-15 11:21:17.0000002  ...                0.0
1662     2013-03-12 19:02:00.00000016  ...                0.0
1723      2011-07-12 16:25:25.0000003  ...                0.0
1774      2012-08-27 15:24:00.0000007  ...                0.0
1906      2011-06-16 16:59:06.0000001  ...                0.0
2281    2011-08-29 08:24:00.000000130  ...                0.0
2346    2009-02-07 00:14:00.000000223  ...                0.0
2510      2010-03-30 13:21:00.0000001  ...                0.0
2656      2012-04-30 07:15:16.0000002  ...                0.0
2763     2009-08-26 18:55:00.00000092  ...                0.0
3587     2010-12-29 17:59:00.00000021  ...                0.0
3945    2010-03-28 03:33:00.000000140  ...                0.0
4086      2009-06-29 12:31:54.0000001  ...                0.0
4240     2012-10-01 20:24:00.00000074  ...                0.0
...                               ...  ...                ...
997917    2013-07-23 13:28:42.0000001  ...                0.0
997965    2009-02-10 23:02:46.0000004  ...                0.0
998113    2009-08-03 13:36:13.0000002  ...                0.0
998140    2009-04-12 13:10:03.0000004  ...                0.0
998163   2011-11-28 16:56:00.00000023  ...                0.0
998239    2015-01-21 20:14:58.0000009  ...                0.0
998257   2009-09-18 08:49:00.00000012  ...                0.0
998282   2011-06-22 19:35:00.00000024  ...                0.0
998416    2010-03-22 23:08:54.0000002  ...                0.0
998598   2012-02-27 20:45:00.00000060  ...                0.0
998641  2012-02-10 21:19:00.000000185  ...                0.0
998883    2011-06-29 18:20:21.0000004  ...                0.0
999006    2011-12-29 06:51:35.0000002  ...                0.0
999052   2011-01-30 00:26:00.00000086  ...                0.0
999146   2012-12-25 01:42:00.00000020  ...                0.0
999183  2010-05-25 11:59:00.000000112  ...                0.0
999342   2013-03-09 15:56:00.00000049  ...                0.0
999356    2013-02-07 09:55:22.0000001  ...                0.0
999365    2012-02-29 10:51:34.0000002  ...                0.0
999406    2009-11-10 17:41:21.0000006  ...                0.0
999461   2012-01-21 14:58:00.00000089  ...                0.0
999467    2010-04-11 19:50:29.0000003  ...                0.0
999537  2014-09-25 22:28:00.000000176  ...                0.0
999641   2009-11-30 10:11:00.00000010  ...                0.0
999686    2009-11-07 10:01:09.0000003  ...                0.0
```

```
999727    2012-04-02 18:23:00.0000007  ...              0.0
999827   2010-11-19 10:34:00.00000044  ...              0.0
999931  2012-03-05 22:22:00.000000181  ...              0.0
999988   2011-05-14 07:21:00.00000014  ...              0.0
999996    2010-09-20 14:50:37.0000002  ...              0.0

[10472 rows x 17 columns]
```

```
dist_zero.shape
```

```
(10472, 17)
```

- There are 20776 rows which is a huge number to just delete the data, but can we do is that as we know fare a below mentioned formula,

```
fare amount = ( base fare ) + ( distance )* (fare / mile)
```

- From here, we can calculate distance as,

```
distance = ((fare amount) - ( base fare) ) / (fare / mile)
```

- All we have to know is the fare amount, base fare and the fare/mile cost.

As per the source http://www.nyc.gov/html/tlc/html/passenger/taxicab_rate.shtml ,

- Base fare (initial charge) is $2.5. - - - farepermileisabout$1.5.
- There is a daily 50-cent surcharge from 8pm to 6am.
- There is a $1 surcharge from 4pm to 8pm on weekdays.

First we will set the minimun fare_amount of the taxi rides to be $2.5.

```
data.shape
```

```
(978953, 17)
```

```
fare_less_than_base = data.loc[((data['distance']==0) & (data['fare_amount'] < 2.5))]
fare_less_than_base.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitu
count	3.0	3.000000	3.000000	3.000000	3.0000
mean	0.0	-74.006640	40.736437	-74.006640	40.7364
std	0.0	0.032410	0.054271	0.032410	0.0542
min	0.0	-74.043442	40.679971	-74.043442	40.6799
25%	0.0	-74.018784	40.710551	-74.018784	40.7105
50%	0.0	-73.994125	40.741131	-73.994125	40.7411
75%	0.0	-73.988240	40.764669	-73.988240	40.7646
max	0.0	-73.982354	40.788208	-73.982354	40.7882

```
data=data.drop(fare_less_than_base.index, axis=0)
```

```
data.shape
```

```
(978950, 17)
```

```
data=data[data['fare_amount']>=2.5]
```

```
data.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati
count	978922.000000	978922.000000	978922.000000	978922.000000	978922.00
mean	11.326086	-73.975087	40.751116	-73.974192	40.75
std	9.689774	0.038421	0.029530	0.037554	0.03
min	2.500000	-74.299372	40.503982	-74.299372	40.50
25%	6.000000	-73.992268	40.736583	-73.991569	40.73
50%	8.500000	-73.982082	40.753417	-73.980590	40.75
75%	12.500000	-73.968313	40.767581	-73.965322	40.76
max	500.000000	-72.940862	41.696852	-72.900000	41.64

- Now we need to append the fare_amount where the distance is zero due to same pickup and dropoff longitud
- We have 3 time periods,

```
Weekdays ( 6AM to 4PM ) & Weekends - 2.5 + distance * 1.5
Weekdays ( 4PM to 8PM ) - 2.5 + 1 + distance * 1.5
Weekdays & Weekends ( 8PM to 6AM) - 2.5 + 0.5 + distance * 1.5
```

```
data_dist_zero=data.loc[(data['distance']==0)]
```

Now we generalize the fare for every time and just set base fare as 2.5 dollars and fare per mile will be around 1.59

```
data_dist_zero.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latit
count	10469.000000	10469.000000	10469.000000	10469.000000	10469.000
mean	10.972748	-73.944035	40.761166	-73.944035	40.761
std	14.869779	0.133932	0.098780	0.133932	0.098
min	2.500000	-74.299372	40.525964	-74.299372	40.525
25%	4.900000	-73.990385	40.734292	-73.990385	40.734
50%	6.900000	-73.975640	40.752426	-73.975640	40.752
75%	10.500000	-73.948726	40.767326	-73.948726	40.767
max	500.000000	-73.137393	41.513024	-73.137393	41.513

```
data_dist_zero['distance'] = data_dist_zero.apply(
lambda row: ((row['fare_amount']-2.50)/1.59), axis=1
)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexi
```

```
data_dist_zero.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latit
count	10469.000000	10469.000000	10469.000000	10469.000000	10469.000
mean	10.972748	-73.944035	40.761166	-73.944035	40.761
std	14.869779	0.133932	0.098780	0.133932	0.098
min	2.500000	-74.299372	40.525964	-74.299372	40.525
25%	4.900000	-73.990385	40.734292	-73.990385	40.734
50%	6.900000	-73.975640	40.752426	-73.975640	40.752
75%	10.500000	-73.948726	40.767326	-73.948726	40.767
max	500.000000	-73.137393	41.513024	-73.137393	41.513

```
data.update(data_dist_zero)
```

```
data.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati
count	978922.000000	978922.000000	978922.000000	978922.000000	978922.00
mean	11.326086	-73.975087	40.751116	-73.974192	40.75
std	9.689774	0.038421	0.029530	0.037554	0.03
min	2.500000	-74.299372	40.503982	-74.299372	40.50
25%	6.000000	-73.992268	40.736583	-73.991569	40.73
50%	8.500000	-73.982082	40.753417	-73.980590	40.75
75%	12.500000	-73.968313	40.767581	-73.965322	40.76
max	500.000000	-72.940862	41.696852	-72.900000	41.64

```
data=data[data['fare_amount']>3.0]
```

```
data.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati
count	971347.000000	971347.000000	971347.000000	971347.000000	971347.00
mean	11.393055	-73.975159	40.751127	-73.974258	40.75
std	9.697626	0.038206	0.029400	0.037357	0.03
min	3.300000	-74.299372	40.503982	-74.299372	40.50
25%	6.000000	-73.992277	40.736595	-73.991573	40.73
50%	8.500000	-73.982093	40.753420	-73.980601	40.75
75%	12.500000	-73.968372	40.767580	-73.965387	40.76
max	500.000000	-72.940862	41.696852	-72.900000	41.64

- This is just an intuation step that the data will be linearly giving out the fare amounts.
- So, just considering the fare to be less than 200 and distance is limited to 100.

```
data=data[data['fare_amount']<250]
```

```
data=data[data['distance']<150]
```

```
data=data[data['passenger_count']>0]
```

```
data=data[data['distance']>3]
```

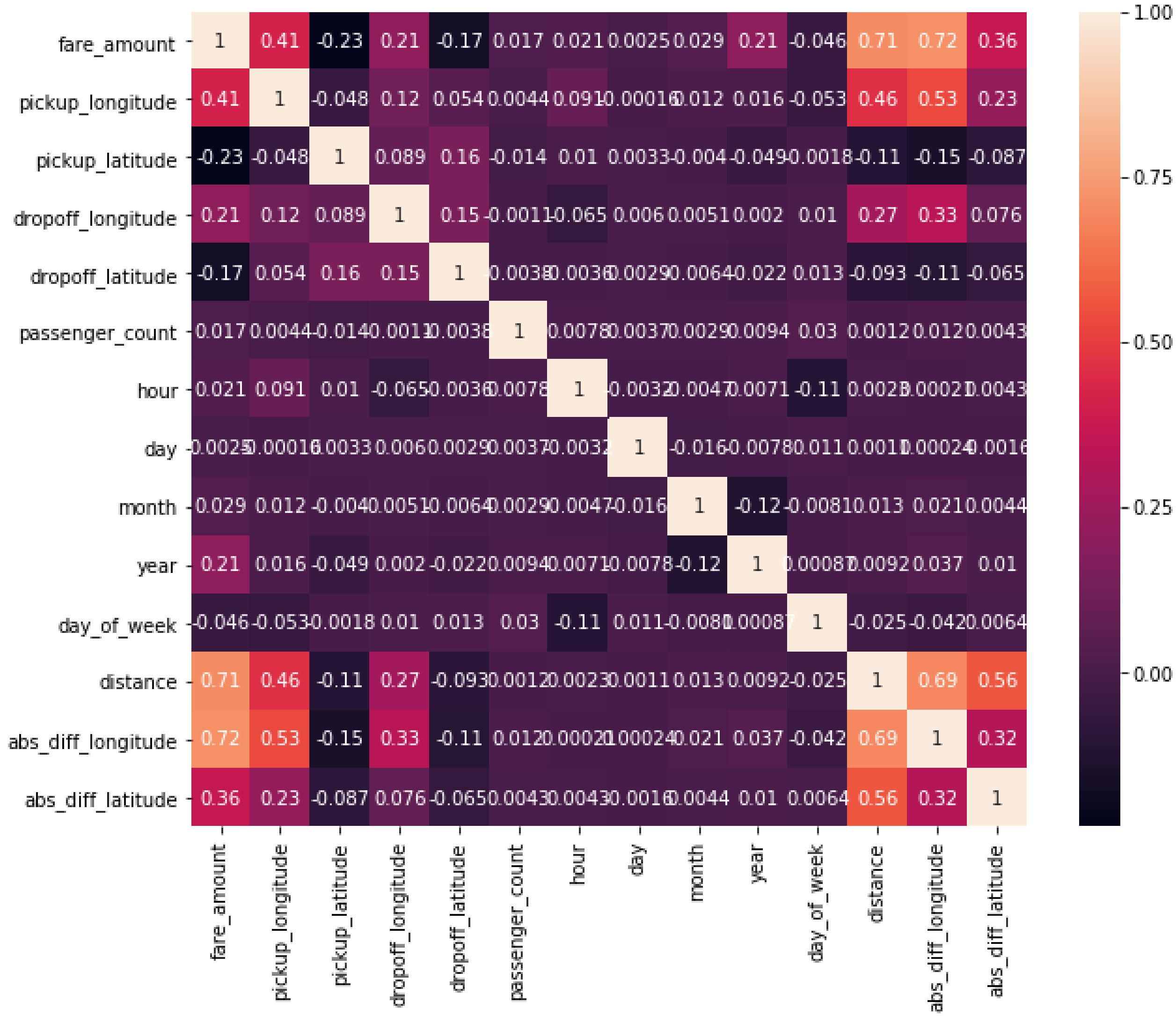
Modelling

Let us start training our data and use Linear Regression Basemodel to predict the fare amounts

- Initially we first check on the training data and check the rmse and then try to predict the values of the test da
- We plot a heat map which gives the correlation coefficients so we can select features through this and the an

```
plt.figure(figsize=(10,8))
sea.heatmap(data.corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbeb51af7f0>
```




```
data.columns
```

```
Index(['key', 'fare_amount', 'pickup_datetime', 'pickup_longitude',
       'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude',
       'passenger_count', 'date_time', 'hour', 'day', 'month', 'year',
       'day_of_week', 'distance', 'abs_diff_longitude', 'abs_diff_latitude'],
      dtype='object')
```

```
test.columns
```

```
Index(['key', 'pickup_datetime', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count', 'date_time',
       'hour', 'day', 'month', 'year', 'day_of_week', 'distance',
       'abs_diff_longitude', 'abs_diff_latitude'],
      dtype='object')
```

Let us import the required model and metrics from sklearn for modelling

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```

- Now we select columns required to be used for training our model.

```
columns_train=['distance','day_of_week','passenger_count','year', 'month','abs_diff_longitude','abs_
columns_target=['fare_amount']
```

```
X=data[columns_train]
Y=data[columns_target]
```

- Assigning the values to the train set for X_train and Y_train.
- The training data is split into train and test randomly using sklearn train_test_split

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.15, random_state = 7)

# Check shape
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(157873, 11)
(27861, 11)
(157873, 1)
(27861, 1)
```

Fitting the Baseline Regression model.

Ignore

5 cells hidden

Ignore

5 cells hidden

Prediction of test values:

- This is the final part for which we have been performing different perspective analysis on our data.
- Assign training data values to trainX and trainY which contains features and target value respectively.

```
trainX=data[columns_train]
trainY=data[columns_target]
```

Now assign the test data to the ' test ' variable for which we need to predit the fare amount.

```
test=test[columns_train]
```

Fit the model with the training data.

```
pf=LinearRegression()
pf.fit(trainX,trainY)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Predict the fare amount values for the data in the test set.

```
fare_predict=pf.predict(test)
```

Final Steps:

- Now the only step left is that to write the predicted values into the sample_submission.csv file and then overv which we have predicted.
- Pandas to_csv() is used to write the submissions on to the file.

```
submission = pd.read_csv('sample_submission.csv')
submission['fare_amount'] = fare_predict
submission.to_csv('full_functional__extra_cleaned_submission_linear_reg.csv', index=False)
submission.tail(20)
```

	key	fare_amount
9894	2013-09-25 22:00:00.000000153	11.403262
9895	2013-09-25 22:00:00.000000241	22.510509
9896	2013-09-25 22:00:00.000000127	10.692350
9897	2015-02-20 11:08:29.0000001	13.741110
9898	2015-01-12 15:36:37.0000002	6.866251
9899	2015-06-07 00:38:14.0000002	18.913982
9900	2015-04-12 21:56:22.0000005	9.438104
9901	2015-04-10 11:56:54.0000004	8.331925
9902	2015-06-25 01:01:46.0000002	15.987964
9903	2015-05-29 10:02:42.0000001	10.201975
9904	2015-06-30 20:03:50.0000002	44.123814
9905	2015-02-27 19:36:02.0000006	22.239211
9906	2015-06-15 01:00:06.0000002	8.566482
9907	2015-02-03 09:00:58.0000001	28.521944
9908	2015-05-19 13:58:11.0000001	10.298492
9909	2015-05-10 12:37:51.0000002	9.450211
9910	2015-01-12 17:05:51.0000001	11.205466
9911	2015-04-19 20:44:15.0000001	50.385502
9912	2015-01-31 01:05:19.0000005	21.318508
9913	2015-01-18 14:06:23.0000006	8.628577

- Our submission scored a 5.23 rmse which is ok, considering it (Linear Regression) is a basline model.
- Now let us try to predict the fare_amount using other complex models

K-Nearest Neighbour Regression

- Import the regressor from sklearn
- fit the data
- predict the values

```
from sklearn.neighbors import KNeighborsRegressor
```

Fit the data model with the training data.

```
knr = KNeighborsRegressor()
knr.fit(X_train, Y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

Predict the values for the provided data in the X_test sample.

```
knr_predict = knr.predict(X_test)
```

Calculate the rmse for the predicted values and compare with the baseline models.

```
rmse_knr = np.sqrt(mean_squared_error(Y_test , knr_predict))
print("Root Mean Squared Error:",rmse_knr)
```

```
Root Mean Squared Error: 7.455189424922282
```

Let us try to predict the fare amounts for the data in the test set and write to a submission file to check the rmse we

```
knr_final = KNeighborsRegressor()
knr_final.fit(trainX, trainY)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=5, p=2,
          weights='uniform')
```

```
fare_predict_knr=knr_final.predict(test)
```

```
submission = pd.read_csv('sample_submission.csv')
submission['fare_amount'] = fare_predict_knr
submission.to_csv('full_functional_extra_cleaned_submission_nearest_neighbors_reg.csv', index=False)
submission.tail(20)
```

	key	fare_amount
9894	2013-09-25 22:00:00.000000153	11.300
9895	2013-09-25 22:00:00.000000241	27.232
9896	2013-09-25 22:00:00.000000127	9.100
9897	2015-02-20 11:08:29.0000001	19.000
9898	2015-01-12 15:36:37.0000002	4.700
9899	2015-06-07 00:38:14.0000002	17.800
9900	2015-04-12 21:56:22.0000005	7.600
9901	2015-04-10 11:56:54.0000004	8.700
9902	2015-06-25 01:01:46.0000002	13.900
9903	2015-05-29 10:02:42.0000001	10.800
9904	2015-06-30 20:03:50.0000002	44.316
9905	2015-02-27 19:36:02.0000006	39.198
9906	2015-06-15 01:00:06.0000002	5.300
9907	2015-02-03 09:00:58.0000001	30.698
9908	2015-05-19 13:58:11.0000001	8.000
9909	2015-05-10 12:37:51.0000002	8.200
9910	2015-01-12 17:05:51.0000001	11.700
9911	2015-04-19 20:44:15.0000001	56.390
9912	2015-01-31 01:05:19.0000005	18.900
9913	2015-01-18 14:06:23.0000006	5.500

- Our submission scored a 4.36 rmse, we reduced it by ~17%.
- Now let us try to predict the fare_amount using Random Forest Regression.

Random Forest Regression

- Import the regressor from sklearn
- Fit the data
- Predict the values

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

```
data = data.drop(['key'], axis=1)
```

```
x_train = data.iloc[:,data.columns!='fare_amount']
y_train = data['fare_amount'].values
x_test = test
```

```
data.dtypes
```

```
fare_amount          float64
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count      float64
hour                 float64
day                  float64
month                float64
year                 float64
day_of_week          float64
distance             float64
abs_diff_longitude   float64
abs_diff_latitude    float64
dtype: object
```

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

```
test = test.drop(['key'], axis=1)
```

```
test.dtypes
```

```
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
hour                   int64
day                    int64
month                  int64
year                   int64
day_of_week            int64
distance             float64
abs_diff_longitude   float64
abs_diff_latitude    float64
dtype: object
```

```
from sklearn.ensemble import RandomForestRegressor
```

```
random_fr = RandomForestRegressor()
random_fr.fit(x_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

```
random_fr_predict = random_fr.predict(test)
```

```
rmse_random_forest = np.sqrt(mean_squared_error(Y_test , random_fr_predict))
print("Root Mean Squared Error:",rmse_random_forest)
```

```
('Root Mean Squared Error:', 3.4829943327788238)
```

```
rd_final = RandomForestRegressor(n_estimators=100)
rd_final.fit(x_train, y_train)
```

```
/anaconda2/lib/python2.7/site-packages/ipykernel_launcher.py:2: DataConversionWarning: A

RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

```
fare_predict_rd=random_fr.predict(x_test)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-59-94291bfa7323> in <module>()
----> 1 fare_predict_rd=random_fr.predict(x_test)

3 frames
/anaconda2/lib/python2.7/site-packages/sklearn/utils/validation.pyc in check_array(array
    431                                       force_all_finite)
    432     else:
--> 433         array = np.array(array, dtype=dtype, order=order, copy=copy)
    434 
    435         if ensure_2d:

TypeError: float() argument must be a string or a number
```

SEARCH STACK OVERFLOW

```
submission = pd.read_csv('sample_submission.csv')
submission['fare_amount'] = random_fr_predict
submission.to_csv('final_date.csv', index=False)
submission.tail(20)
```

	key	fare_amount
9894	2013-09-25 22:00:00.000000153	17.750
9895	2013-09-25 22:00:00.000000241	25.550
9896	2013-09-25 22:00:00.000000127	16.683
9897	2015-02-20 11:08:29.0000001	18.150
9898	2015-01-12 15:36:37.0000002	19.850
9899	2015-06-07 00:38:14.0000002	16.300
9900	2015-04-12 21:56:22.0000005	14.550
9901	2015-04-10 11:56:54.0000004	21.083
9902	2015-06-25 01:01:46.0000002	15.630
9903	2015-05-29 10:02:42.0000001	21.550
9904	2015-06-30 20:03:50.0000002	45.800
9905	2015-02-27 19:36:02.0000006	22.050
9906	2015-06-15 01:00:06.0000002	14.550
9907	2015-02-03 09:00:58.0000001	44.930
9908	2015-05-19 13:58:11.0000001	21.400
9909	2015-05-10 12:37:51.0000002	16.600
9910	2015-01-12 17:05:51.0000001	19.516
9911	2015-04-19 20:44:15.0000001	55.727
9912	2015-01-31 01:05:19.0000005	15.800
9913	2015-01-18 14:06:23.0000006	18.300

Gradient Boosting Regression

- Gradient boosting is a machine learning technique for regression and classification problems, which produce: of weak prediction models.
- It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowin function.

- Garadient Boosting Regression is readily availabale in sklearn.
- We import ensemble from sklearn from which we can apply the Regression algorithm.
- Fit the training data into the model.

```
from sklearn import ensemble
grad_test = ensemble.GradientBoostingRegressor(n_estimators=100)
grad_test.fit(X_train,Y_train)
```

```
/anaconda2/lib/python2.7/site-packages/sklearn/utils/validation.py:578: DataConversionWa
  y = column_or_1d(y, warn=True)
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.1, loss='ls', max_depth=3, max_features=None,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=1,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=100, presort='auto', random_state=None,
             subsample=1.0, verbose=0, warm_start=False)
```

Now predict the fare amount for the sample test values created.

```
grad_predict=grad_test.predict(X_test)
```

- Calculate the rmse between the predicted and the actual fare values.
- We should be observing a smaller rmse out of all the above regression models we used as the Gradient Boos with more precision than the above models.

```
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(Y_test , grad_predict))
print("Root Mean Squared Error:",rmse)
```

```
('Root Mean Squared Error:', 3.575135147206956)
```

```
f=['distance','day_of_week','passenger_count','year', 'month','abs_diff_longitude','abs_diff_latitud
t=['fare_amount']
ls_trainX=data[f]
ls_trainY=data[t]
test=test[f]
```

Fit all the training data into the Gradient Boosting Regressor.

```
gd_final = ensemble.GradientBoostingRegressor(n_estimators=100)
gd_final.fit(ls_trainX,ls_trainY)
```

```
/anaconda2/lib/python2.7/site-packages/sklearn/utils/validation.py:578: DataConversionWa
  y = column_or_1d(y, warn=True)
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.1, loss='ls', max_depth=3, max_features=None,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=1,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=100, presort='auto', random_state=None,
             subsample=1.0, verbose=0, warm_start=False)
```

Predict the fare amounts for the values in the test dataset.

```
fare_predict_gd=gd_final.predict(test)
```

Write the submissions to the csv file and submitting it to kaggle.

```
submission = pd.read_csv('sample_submission.csv')
submission['fare_amount'] = fare_predict_gd
submission.to_csv('full_functional_extra_cleaned_submission_gradiant_boosting_reg_123.csv', index=Fa
submission.tail(20)
```

	key	fare_amount
9894	2013-09-25 22:00:00.000000153	11.636090
9895	2013-09-25 22:00:00.000000241	25.486449
9896	2013-09-25 22:00:00.000000127	11.306746
9897	2015-02-20 11:08:29.0000001	16.359373
9898	2015-01-12 15:36:37.0000002	5.511266
9899	2015-06-07 00:38:14.0000002	19.129598
9900	2015-04-12 21:56:22.0000005	8.416444
9901	2015-04-10 11:56:54.0000004	7.970733
9902	2015-06-25 01:01:46.0000002	15.973977
9903	2015-05-29 10:02:42.0000001	9.827504
9904	2015-06-30 20:03:50.0000002	47.619005
9905	2015-02-27 19:36:02.0000006	26.552371
9906	2015-06-15 01:00:06.0000002	5.769054
9907	2015-02-03 09:00:58.0000001	37.569472
9908	2015-05-19 13:58:11.0000001	8.320748
9909	2015-05-10 12:37:51.0000002	8.838086
9910	2015-01-12 17:05:51.0000001	11.903043
9911	2015-04-19 20:44:15.0000001	56.399752
9912	2015-01-31 01:05:19.0000005	22.642865
9913	2015-01-18 14:06:23.0000006	6.595519

```
data.describe()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-2bb0b18689d4> in <module>()
----> 1 data.describe()

NameError: name 'data' is not defined
```

SEARCH STACK OVERFLOW