Problem Statement:

A real estate agent want to help to predict the house price for regions in USA.He gave us the dataset to work on to use Linear Regression modelCreate a Model that helps him to estimate of what the house would sell for

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv("uber.csv")
df
```

Out[2]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_la
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.7
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.7
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.7
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.7
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.7
...	...	...	...	...	...	
199995	42598914	2012-10-28 10:49:00.00000053	3.0	2012-10-28 10:49:00 UTC	-73.987042	40.7
199996	16382965	2014-03-14 01:09:00.0000008	7.5	2014-03-14 01:09:00 UTC	-73.984722	40.7
199997	27804658	2009-06-29 00:42:00.00000078	30.9	2009-06-29 00:42:00 UTC	-73.986017	40.7
199998	20259894	2015-05-20 14:56:25.0000004	14.5	2015-05-20 14:56:25 UTC	-73.997124	40.7
199999	11951496	2010-05-15 04:08:00.00000076	14.1	2010-05-15 04:08:00 UTC	-73.984395	40.7

200000 rows × 9 columns

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Unnamed: 0         200000 non-null  int64  
 1   key                200000 non-null  object 
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object 
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

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

Out[4]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085

Data cleaning and Pre-Processing

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Unnamed: 0         200000 non-null  int64  
 1   key                200000 non-null  object 
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object 
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

In [6]:
```
df.describe()
```

Out[6]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_
count	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.000000	19999
mean	2.771250e+07	11.359955	-72.527638	39.935885	-72.525292	3
std	1.601382e+07	9.901776	11.437787	7.720539	13.117408	
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300	-88
25%	1.382535e+07	6.000000	-73.992065	40.734796	-73.991407	4
50%	2.774550e+07	8.500000	-73.981823	40.752592	-73.980093	4
75%	4.155530e+07	12.500000	-73.967154	40.767158	-73.963658	4
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603	87

In [7]:
```
a= df.dropna(axis='columns')
a
```

Out[7]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_la
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.7
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.7
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.7
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.7
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.7
...	...	...	...	...	...	
199995	42598914	2012-10-28 10:49:00.00000053	3.0	2012-10-28 10:49:00 UTC	-73.987042	40.7
199996	16382965	2014-03-14 01:09:00.0000008	7.5	2014-03-14 01:09:00 UTC	-73.984722	40.7
199997	27804658	2009-06-29 00:42:00.00000078	30.9	2009-06-29 00:42:00 UTC	-73.986017	40.7
199998	20259894	2015-05-20 14:56:25.0000004	14.5	2015-05-20 14:56:25 UTC	-73.997124	40.7
199999	11951496	2010-05-15 04:08:00.00000076	14.1	2010-05-15 04:08:00 UTC	-73.984395	40.7

200000 rows × 7 columns

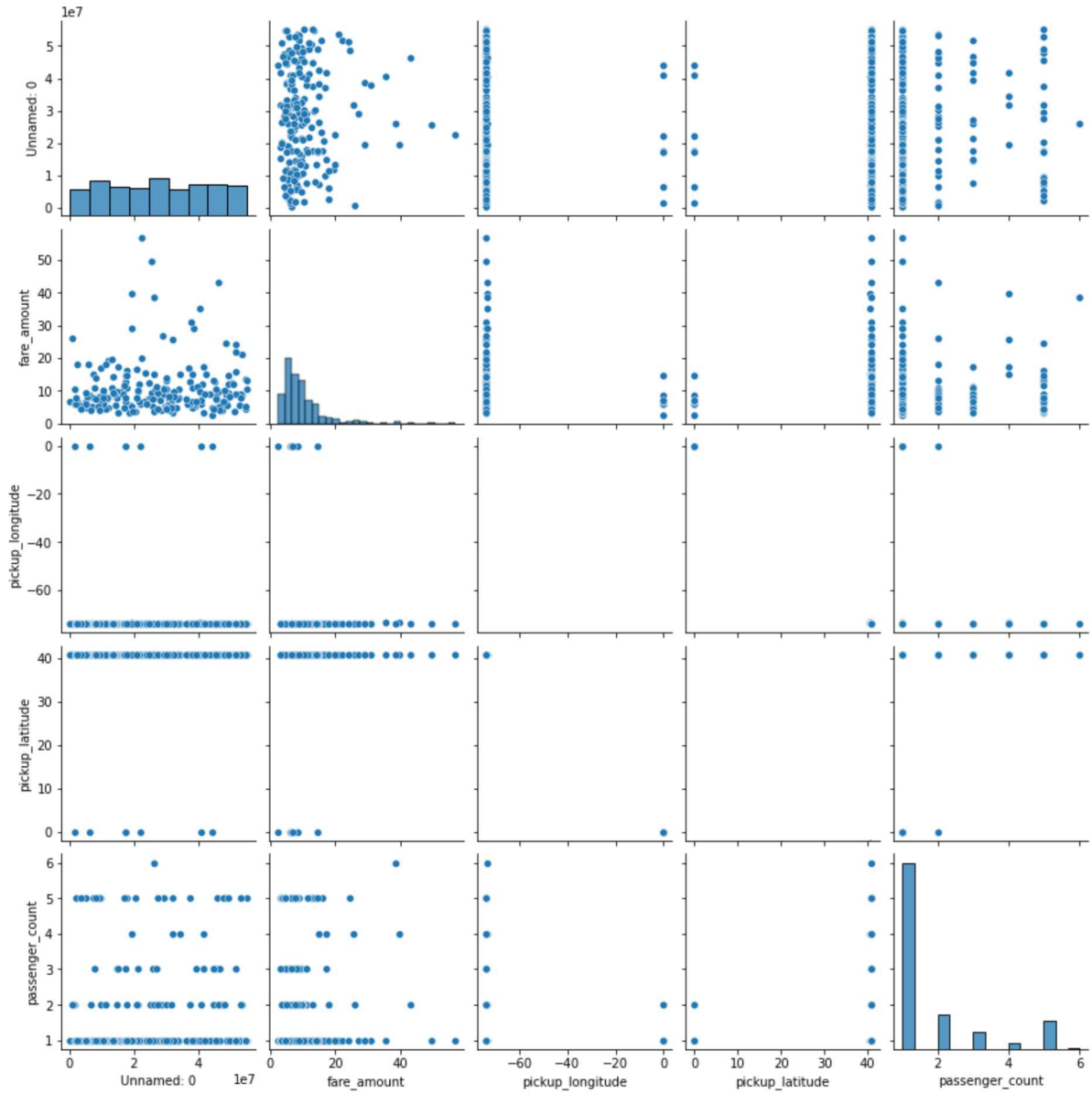
In [8]:
```
a.columns
```

Out[8]:
```
Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'passenger_count'],
      dtype='object')
```

EDA and VISUALIZATION

In [9]:
```
b = a.head(200)
sns.pairplot(b)
```

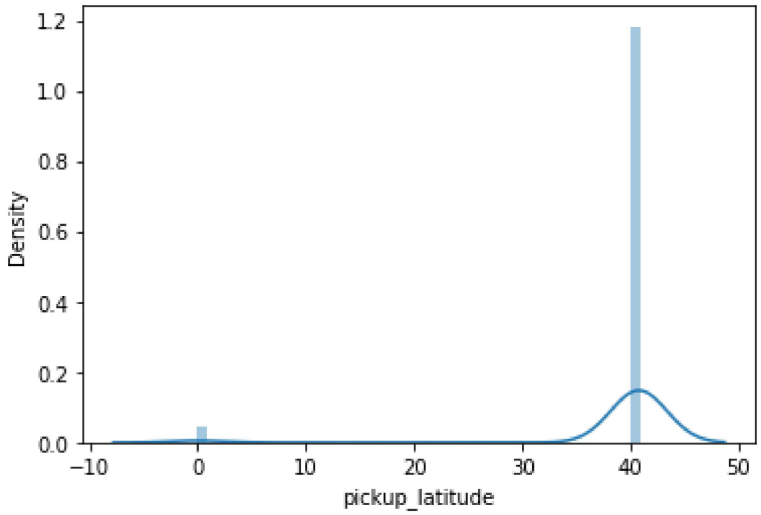
Out[9]: <seaborn.axisgrid.PairGrid at 0x266412da490>



In [10]:
```
sns.distplot(b['pickup_latitude'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function for hi
stograms).
  warnings.warn(msg, FutureWarning)
```

Out[10]: <AxesSubplot:xlabel='pickup_latitude', ylabel='Density'>

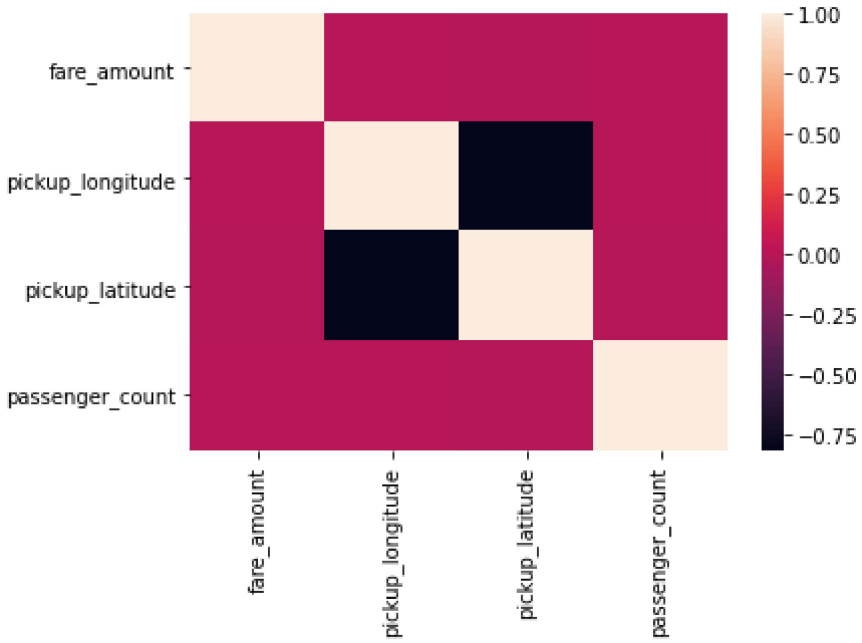


In [11]:
```
df1=df[['key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'passenger_count']]
```

Plot Using Heat Map

In [12]:
```
sns.heatmap(df1.corr())
```

Out[12]: <AxesSubplot:>



To Train The Model-Model Building

we are going to train Linera Regression Model;We need to split out data into two variables x and y where x is independent variable(input) and y is dependent on x(output) we could ignore address column as it required for our model

In [13]:
```
x=df1[['pickup_longitude', 'pickup_latitude' ]]
y=df1[ 'fare_amount']
```

To Split my dataset into training and test data

In [14]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [15]:
```
from sklearn.linear_model import LinearRegression
lr= LinearRegression()
lr.fit(x_train,y_train)
```

Out[15]: LinearRegression()

In [16]:
```
lr.intercept_
```

Out[16]: 12.130267854317225

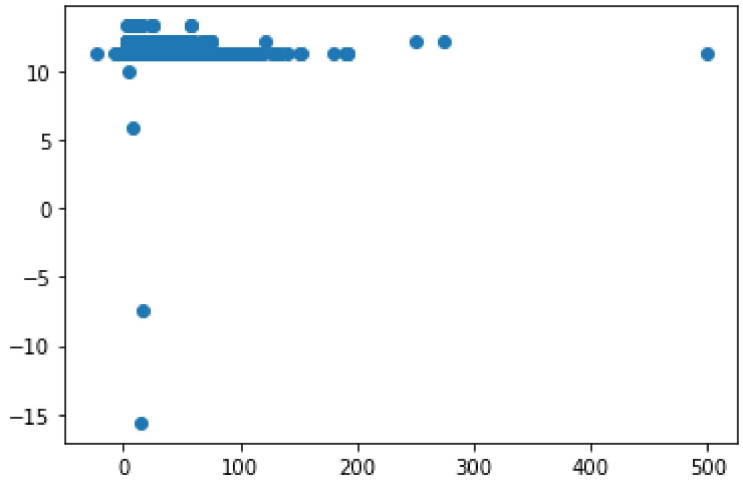
In [17]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[17]:

	Co-efficient
pickup_longitude	0.002004
pickup_latitude	-0.015234

In [18]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[18]: <matplotlib.collections.PathCollection at 0x266536e9ca0>



In [19]:
```
lr.score(x_test,y_test)
```

Out[19]: -0.000151990148022918

In [20]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [21]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
rr.score(x_test,y_test)
rr.score(x_train,y_train)
```

Out[21]: 0.00013844175700927774

In [22]:
```
rr.score(x_test,y_test)
```

Out[22]: -0.00015198723486120613

In [23]:
```
la = Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[23]:
```
Lasso(alpha=10)
```

In [24]:
```
la.score(x_test,y_test)
```

Out[24]:
```
-3.14586989444976e-05
```

In [34]:
```
from sklearn.linear_model import ElasticNet
en = ElasticNet()
en.fit(x_train,y_train)
```

Out[34]:
```
ElasticNet()
```

In [26]:
```
print(en.coef_)
```

```
[ 0.00565512 -0.        ]
```

In [27]:
```
print(en.intercept_)
```

```
11.7869775114203
```

In [28]:
```
print(en.predict(x_test))
```

```
[11.36871276 11.36862459 11.36857745 ... 11.36866601 11.36884279
 11.36871709]
```

In [29]:
```
print(en.score(x_test,y_test))
```

```
4.858561128329164e-05
```

In [30]:
```
# Evaluation Metrics
from sklearn import metrics
```

In [31]:
```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 6.027735821055768
```

In [32]:
```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 100.72094055920147
```

In [33]:
```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prec
```

```
Root Mean Squared Error: 10.035982291694294
```