Problem statement

predicting the house price in USA.To create a model to help him 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")
```

To display top 10 rows

In [3]:
```
1 df.head(10)
```

Out[3]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	p
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	
5	44470845	2011-02-12 02:27:09.0000006	4.9	2011-02-12 02:27:09 UTC	-73.969019	40.755910	-73.969019	40.755910	
6	48725865	2014-10-12 07:04:00.0000002	24.5	2014-10-12 07:04:00 UTC	-73.961447	40.693965	-73.871195	40.774297	
7	44195482	2012-12-11 13:52:00.00000029	2.5	2012-12-11 13:52:00 UTC	0.000000	0.000000	0.000000	0.000000	
8	15822268	2012-02-17 09:32:00.00000043	9.7	2012-02-17 09:32:00 UTC	-73.975187	40.745767	-74.002720	40.743537	
9	50611056	2012-03-29 19:06:00.000000273	12.5	2012-03-29 19:06:00 UTC	-74.001065	40.741787	-73.963040	40.775012	

Data Cleaning And Pre-Processing

In [4]:
```
1 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 [5]:
```
1 # Display the statistical summary
2 df.describe()
```

Out[5]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200000.000000
mean	2.771250e+07	11.359955	-72.527638	39.935885	-72.525292	39.923890	1.684535
std	1.601382e+07	9.901776	11.437787	7.720539	13.117408	6.794829	1.385997
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	1.382535e+07	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	2.774550e+07	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	4.155530e+07	12.500000	-73.967154	40.767158	-73.963658	40.768001	2.000000
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

In [6]:
```
# To display the col headings
df.columns
```

Out[6]:
```
Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

In [7]:
```
cols=df.dropna()
cols
```

Out[7]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitu
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.7232
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.7503
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.7726
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.8033
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.7612
...	...	...	...	...	...	...	...	...
199995	42598914	2012-10-28 10:49:00.00000053	3.0	2012-10-28 10:49:00 UTC	-73.987042	40.739367	-73.986525	40.7402
199996	16382965	2014-03-14 01:09:00.0000008	7.5	2014-03-14 01:09:00 UTC	-73.984722	40.736837	-74.006672	40.7396
199997	27804658	2009-06-29 00:42:00.00000078	30.9	2009-06-29 00:42:00 UTC	-73.986017	40.756487	-73.858957	40.6925
199998	20259894	2015-05-20 14:56:25.0000004	14.5	2015-05-20 14:56:25 UTC	-73.997124	40.725452	-73.983215	40.6954
199999	11951496	2010-05-15 04:08:00.00000076	14.1	2010-05-15 04:08:00 UTC	-73.984395	40.720077	-73.985508	40.7687

199999 rows × 9 columns

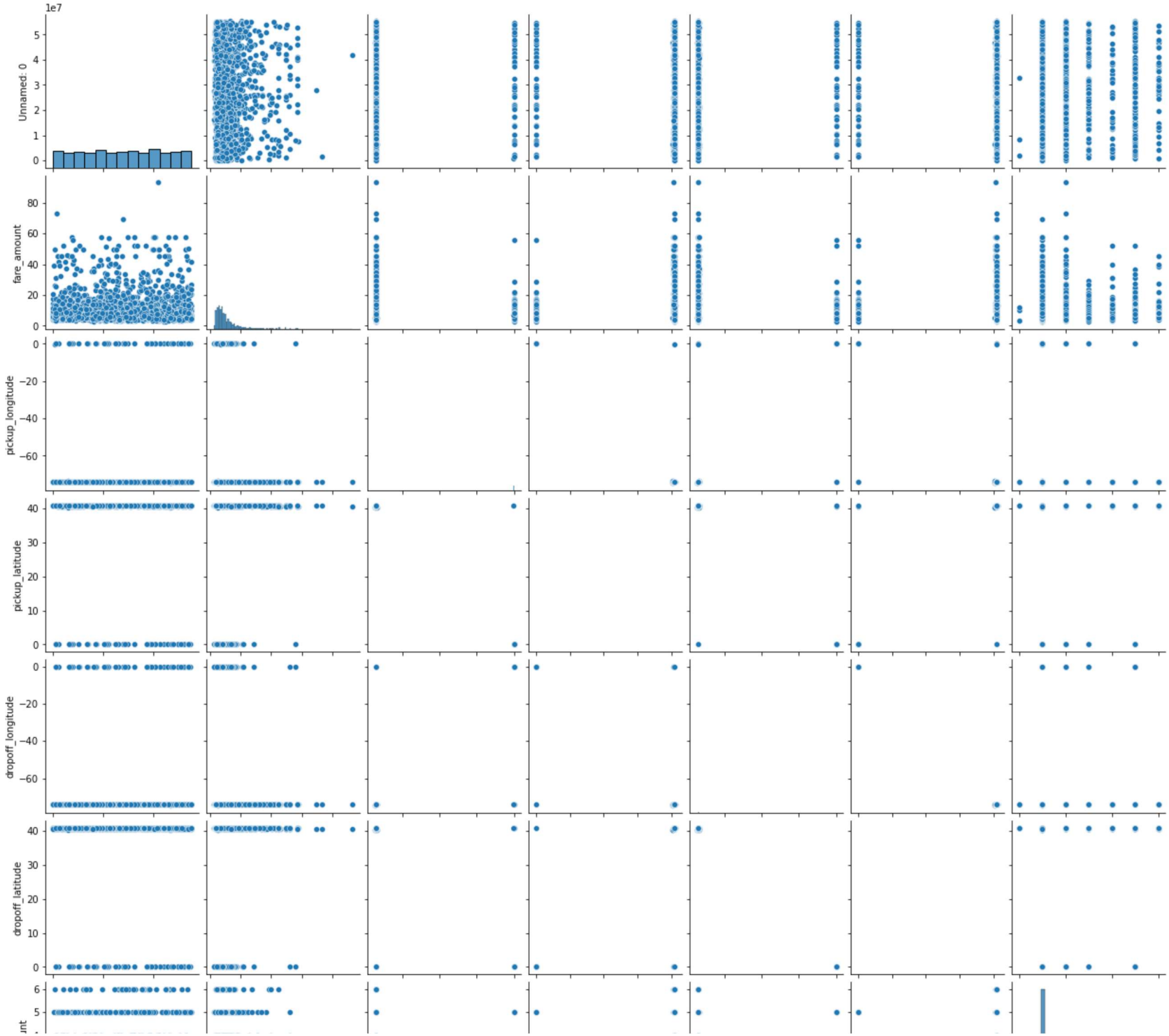
In [8]:
```
cols.columns
cols1=cols[['pickup_longitude', 'pickup_latitude', 'dropoff_longitude']]
```

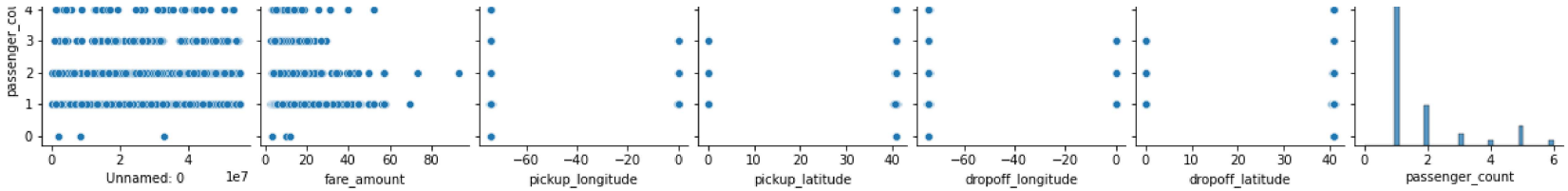
EDA and Visualization

In [9]:
```
d=df.head(2000)
```

In [10]:
```
1 sns.pairplot(d)
```

Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x23221032f10>
```

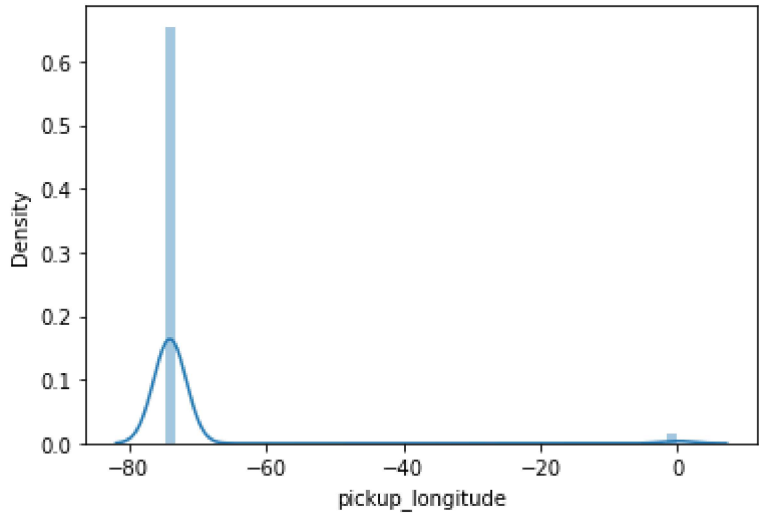

1e7
Unnamed: 0
fare_amount
pickup_longitude
pickup_latitude
dropoff_longitude
dropoff_latitude



In [11]:
```
1 # We use displot in older version we get distplot use displot
2 sns.distplot(d['pickup_longitude'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a dep
recated function and will be removed in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[11]: <AxesSubplot:xlabel='pickup_longitude', ylabel='Density'>



In [12]:

```
df1=df[[ 'pickup_longitude', 'pickup_latitude', 'passenger_count']]
df1
```

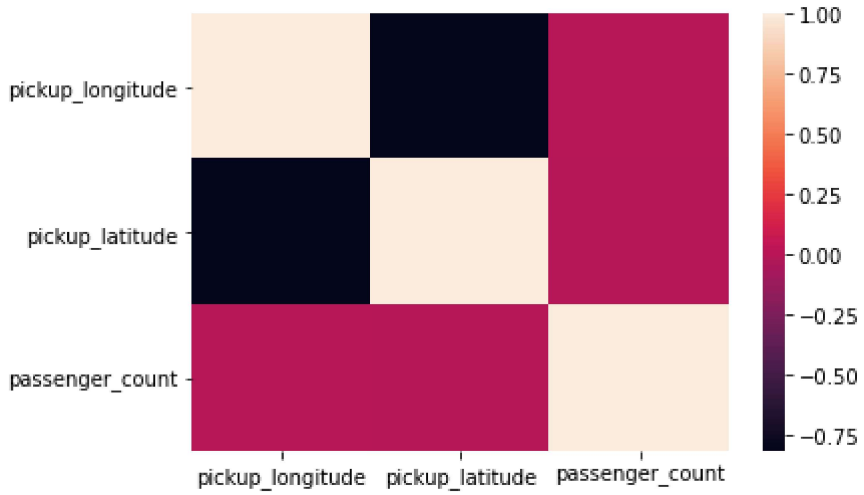
Out[12]:

	pickup_longitude	pickup_latitude	passenger_count
0	-73.999817	40.738354	1
1	-73.994355	40.728225	1
2	-74.005043	40.740770	1
3	-73.976124	40.790844	3
4	-73.925023	40.744085	5
...	...	...	...
199995	-73.987042	40.739367	1
199996	-73.984722	40.736837	1
199997	-73.986017	40.756487	2
199998	-73.997124	40.725452	1
199999	-73.984395	40.720077	1

200000 rows × 3 columns

In [13]:
```
1 sns.heatmap(df1.corr())
```

Out[13]: <AxesSubplot:>



To train the model - MODEL BUILD

Going to train linear regression model;We split our data into 2 variables x and y where x is independent var(input) and y is dependent on x(output), we could ignore address col as it is not required for our model

In [14]:
```
1 x=df1[[ 'pickup_longitude', 'pickup_latitude', 'passenger_count']]
2 y=df1[['pickup_latitude']]
```

To split the dataset into test data

In [15]:
```
1 # importing lib for splitting test data
2 from sklearn.model_selection import train_test_split
```

In [16]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

In [17]:
```
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[17]:
```
LinearRegression()
```

In [18]:
```
print(lr.intercept_)
```

```
[-2.13162821e-14]
```

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

```
1.0
```

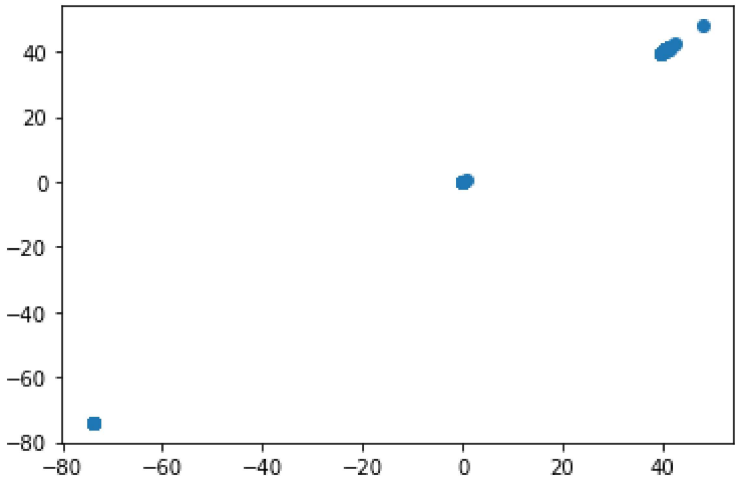
In [20]:
```
coeff=pd.DataFrame(lr.coef_)
coeff
```

Out[20]:

	0	1	2
0	1.599853e-15	1.0	-1.195019e-17

In [21]:
```
pred = lr.predict(x_test)
plt.scatter(y_test,pred)
```

Out[21]: <matplotlib.collections.PathCollection at 0x2324a6cc4f0>



In [22]:
```
from sklearn.linear_model import Ridge,Lasso
```

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

Out[23]: Ridge(alpha=10)

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

Out[24]: 0.9999999999992462

In [25]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[25]: Lasso(alpha=10)

In [26]:
```
la.score(x_test,y_test)
```

Out[26]: 0.9825873230264505

ELASTIC NET

In [27]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[27]:
```
ElasticNet()
```

In [28]:
```
print(en.coef_)
```

```
[-0.01115547  0.97354557 -0.        ]
```

In [29]:
```
print(en.intercept_)
```

```
[0.24739935]
```

In [30]:
```
prediction=en.predict(x_test)
prediction
```

Out[30]:
```
array([40.72539311, 40.70381651, 40.76041058, ..., 40.75356262,
       40.80343105, 40.75741364])
```

In [31]:
```
print(en.score(x_test,y_test))
```

```
0.9999002858020514
```

EVALUATION METRICS

In [32]:
```
from skleprint("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))arn import metrics
```

In [33]:
```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 0.01089527258153302
```

In [34]:
```
1 print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 0.0035466293278402 06
```

In [35]:
```
1 print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 0.059553583669164745
```