A real estate agent want help to predict the house price for regions in Usa.he gave us the dataset to work on to use linear Regression model.Create a model that helps him to estimate

Data Collection

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

In [2]:
```
#import the dataset
data=pd.read_csv(r"C:\Users\user\Desktop\Vicky\7_uber.csv")[0:500]
```

In [3]:
```
#to display top 10 rows
data.head()
```

Out[3]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitud
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.73835
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.72822
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.74077
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.79084
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.74408

In [4]:
```
#to display null values
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         500 non-null    int64  
 1   key                500 non-null    object 
 2   fare_amount        500 non-null    float64
 3   pickup_datetime    500 non-null    object 
 4   pickup_longitude   500 non-null    float64
 5   pickup_latitude    500 non-null    float64
 6   dropoff_longitude  500 non-null    float64
 7   dropoff_latitude   500 non-null    float64
 8   passenger_count    500 non-null    int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 35.3+ KB
```

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

Out[5]:
```
(500, 9)
```

In [6]:
```
#to display summary of statistics
data.describe()
```

Out[6]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_l
count	5.000000e+02	500.000000	500.000000	500.000000	500.000000	500.
mean	2.737940e+07	10.708720	-72.053865	39.692497	-72.201155	39.
std	1.607155e+07	8.334145	11.784239	6.491541	11.333432	6.
min	1.862090e+05	2.500000	-74.030417	0.000000	-74.027813	0.
25%	1.250293e+07	6.000000	-73.992804	40.735994	-73.991571	40.
50%	2.749836e+07	8.100000	-73.982352	40.752445	-73.980784	40.
75%	4.157492e+07	12.500000	-73.968724	40.765865	-73.965878	40.
max	5.519870e+07	57.330000	0.001782	40.850558	0.000875	40.

In [7]:
```
#to display columns name
data.columns
```

Out[7]:
```
Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

In [8]:
```
data.fillna(value=5)
```

Out[8]:

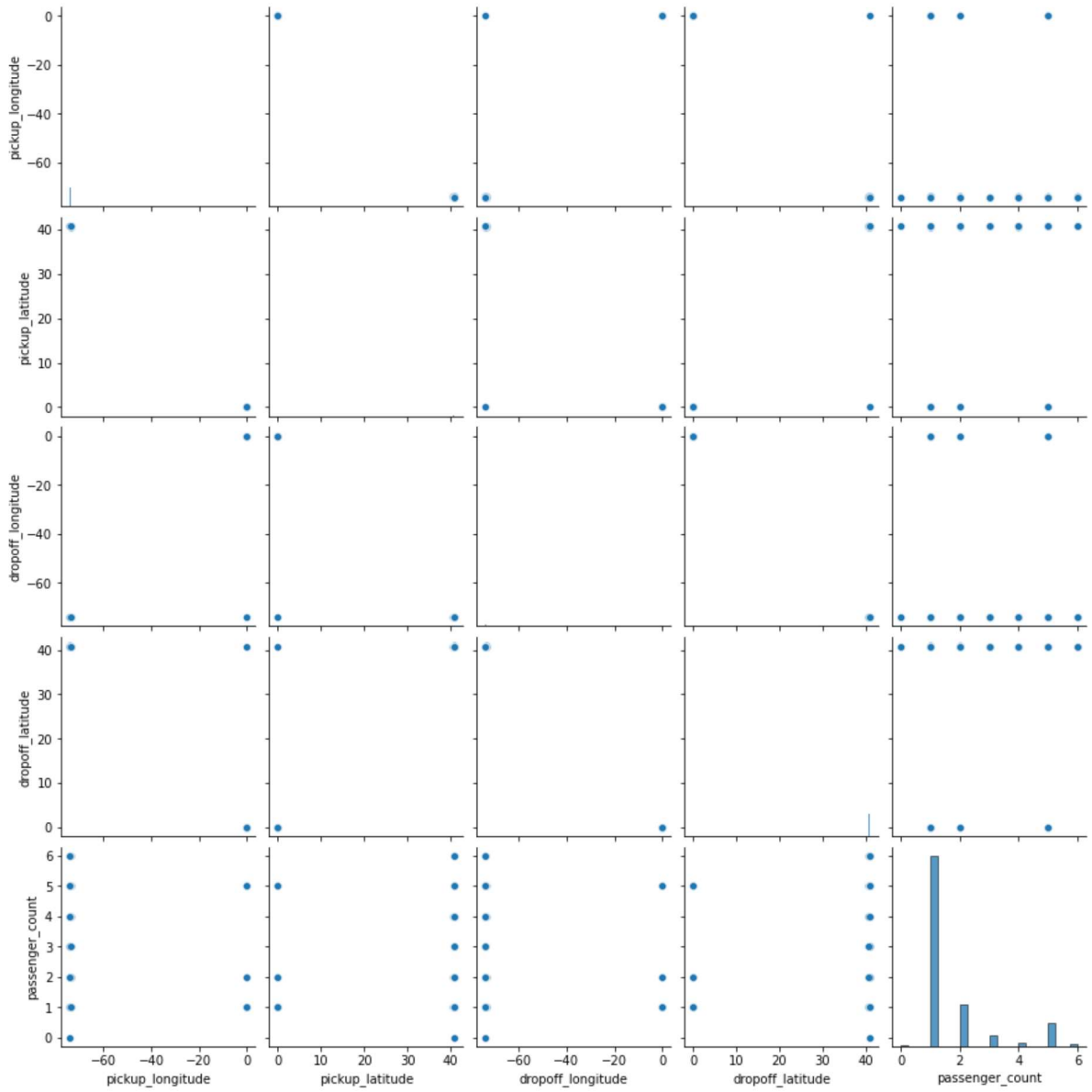
	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitu
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744
...	...	...	...	...	...	
495	1204312	2012-06-03 12:18:02.0000001	25.7	2012-06-03 12:18:02 UTC	-73.862765	40.770
496	2511529	2014-12-24 05:54:45.0000001	8.0	2014-12-24 05:54:45 UTC	-73.918530	40.743
497	24116460	2010-01-18 02:18:16.0000001	10.5	2010-01-18 02:18:16 UTC	-74.005734	40.743
498	42607669	2015-03-30 10:58:37.0000001	5.5	2015-03-30 10:58:37 UTC	-74.001648	40.740
499	36533403	2015-03-09 16:16:21.0000006	10.0	2015-03-09 16:16:21 UTC	-73.960037	40.780

500 rows × 9 columns

In [9]:
```
data1=data[[
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count']]
```

In [10]:
```
sns.pairplot(data1)
```

Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x12542ce5bb0>
```

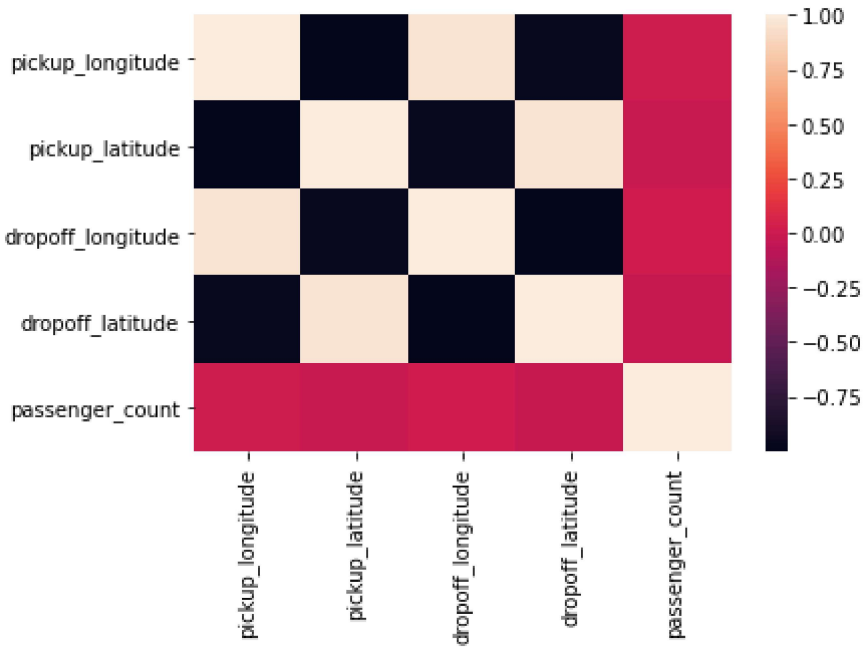


EDA and Visualization

In [11]:
```
#sns.distplot(data['Co2-Emissions'])
```

In [12]:
```
sns.heatmap(data1.corr())
```

Out[12]: <AxesSubplot:>



To train the model

we are going to train the linear regression model ;We need to split the two variable x and y where x in independent variable (input) and y is dependent of x(output) so we could ignore address columns as it is not requires for our model

In [14]:
```
x=data[['Unnamed: 0', 'fare_amount']]
y=data1['passenger_count']
```

In [15]:
```
#To split test and train data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.1)
```

In [16]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[16]: LinearRegression()

In [17]:
```
lr.intercept_
```

Out[17]: 1.6783025895224049

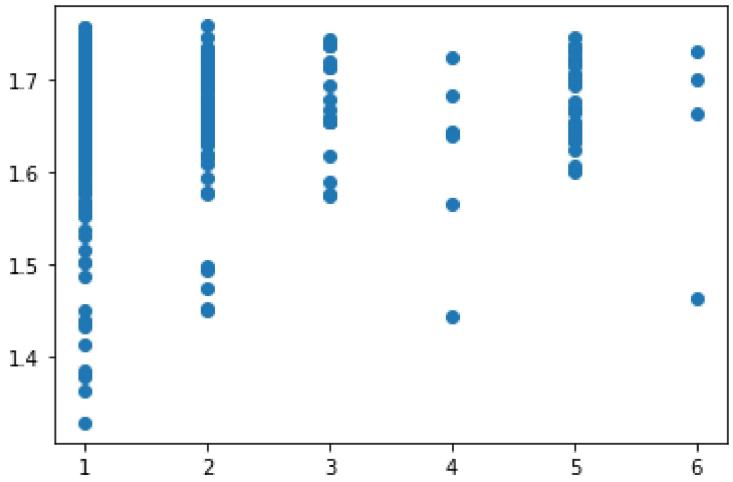
In [18]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=["Co-efficient"])
coeff
```

Out[18]:

	Co-efficient
Unnamed: 0	2.007656e-09
fare_amount	-6.933661e-03

In [19]:
```
prediction = lr.predict(x_train)
plt.scatter(y_train,prediction)
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x1255e9d74c0>
```



In [20]:
```
lr.score(x_test,y_test)
```

Out[20]:
```
0.01310552361153583
```

In [21]:
```
lr.score(x_train,y_train)
```

Out[21]:
```
0.0027974104736769867
```

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

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

Out[23]:
```
0.013104477168466522
```

In [24]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
la.score(x_test,y_test)
```

Out[24]:
```
0.008595632994847224
```

In [25]:
```
from sklearn.linear_model import ElasticNet
en= ElasticNet()
en.fit(x_train,y_train)
```

Out[25]:
```
ElasticNet()
```

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

```
[ 1.86449265e-09 -0.00000000e+00]
```

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

```
1.6086119779780188
```

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

Out[28]:
```
array([1.61329471, 1.65785767, 1.69316902, 1.67935531, 1.65381762,
       1.68127248, 1.62494177, 1.6479163 , 1.66999062, 1.63084741,
       1.61034369, 1.65022449, 1.70034984, 1.66500622, 1.67181656,
       1.69412468, 1.69185324, 1.62220566, 1.69310704, 1.66725315,
       1.71102598, 1.61565701, 1.70118729, 1.68253549, 1.69192796,
       1.65481454, 1.64738271, 1.61473906, 1.64176514, 1.61259816,
       1.6830947 , 1.67588723, 1.62749539, 1.62137754, 1.691492  ,
       1.62445871, 1.62566793, 1.62998135, 1.68416525, 1.66374382,
       1.68995066, 1.62007831, 1.66272789, 1.62672891, 1.64748724,
       1.66585756, 1.66458106, 1.66746733, 1.63287422, 1.63283758])
```

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-29-48d1f0543252> in <module>
----> 1 print(en.score(x_test,y_train))

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py in score(self, X,
y, sample_weight)
    552         from .metrics import r2_score
    553         y_pred = self.predict(X)
--> 554         return r2_score(y, y_pred, sample_weight=sample_weight)
    555
    556     def _more_tags(self):

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in inn
er_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py in
r2_score(y_true, y_pred, sample_weight, multioutput)
    674     -3.0
    675     """
--> 676     y_type, y_true, y_pred, multioutput = _check_reg_targets(
    677         y_true, y_pred, multioutput)
    678     check_consistent_length(y_true, y_pred, sample_weight)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py in
_check_reg_targets(y_true, y_pred, multioutput, dtype)
     86         the dtype argument passed to check_array.
     87     """
---> 88     check_consistent_length(y_true, y_pred)
     89     y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
     90     y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in che
ck_consistent_length(*arrays)
    260     uniques = np.unique(lengths)
    261     if len(uniques) > 1:
--> 262         raise ValueError("Found input variables with inconsistent num
bers of"
    263                          " samples: %r" % [int(l) for l in lengths])
    264

ValueError: Found input variables with inconsistent numbers of samples: [450,
50]
```

In [30]:
```
from sklearn import metrics
```

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

```
Mean Absolute error: 1.0211566873953835
```

In [32]:
```
print("Mean Absolute Square error:",metrics.mean_squared_error(y_test,predicti
```

```
Mean Absolute Square error: 2.072034759627673
```

In [33]:
```
print("Root mean Square error:",np.sqrt(metrics.mean_squared_error(y_test,pred
```

```
Root mean Square error: 1.4394564111593213
```

In [ ]:
```

```