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 [78]:
```
#import the dataset
data=pd.read_csv(r"C:\Users\user\Desktop\Vicky\17_student_marks.csv")[0:500]
```

In [79]:
```
#to display top 10 rows
data.head()
```

Out[79]:

	Student_ID	Test_1	Test_2	Test_3	Test_4	Test_5	Test_6	Test_7	Test_8	Test_9	Test_10	T
0	22000	78	87	91	91	88	98	94	100	100	100	
1	22001	79	71	81	72	73	68	59	69	59	60	
2	22002	66	65	70	74	78	86	87	96	88	82	
3	22003	60	58	54	61	54	57	64	62	72	63	
4	22004	99	95	96	93	97	89	92	98	91	98	

In [80]:
```
#to display null values
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56 entries, 0 to 55
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Student_ID  56 non-null     int64
 1   Test_1      56 non-null     int64
 2   Test_2      56 non-null     int64
 3   Test_3      56 non-null     int64
 4   Test_4      56 non-null     int64
 5   Test_5      56 non-null     int64
 6   Test_6      56 non-null     int64
 7   Test_7      56 non-null     int64
 8   Test_8      56 non-null     int64
 9   Test_9      56 non-null     int64
 10  Test_10     56 non-null     int64
 11  Test_11     56 non-null     int64
 12  Test_12     56 non-null     int64
dtypes: int64(13)
memory usage: 5.8 KB
```

In [81]:
```
data.shape
```

Out[81]:
```
(56, 13)
```

In [82]:
```
#to display summary of statistics
data.describe()
```

Out[82]:

	Student_ID	Test_1	Test_2	Test_3	Test_4	Test_5	Test_6	
count	56.000000	56.000000	56.000000	56.000000	56.000000	56.000000	56.000000	5
mean	22027.500000	70.750000	69.196429	68.089286	67.446429	67.303571	66.000000	6
std	16.309506	17.009356	17.712266	18.838333	19.807179	20.746890	21.054043	2
min	22000.000000	40.000000	34.000000	35.000000	28.000000	26.000000	29.000000	2
25%	22013.750000	57.750000	55.750000	53.000000	54.500000	53.750000	50.250000	4
50%	22027.500000	70.500000	68.500000	70.000000	71.500000	69.000000	65.500000	6
75%	22041.250000	84.000000	83.250000	85.000000	84.000000	85.250000	83.750000	8
max	22055.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	10

In [83]:
```
#to display columns name
data.columns
```

Out[83]:
```
Index(['Student_ID', 'Test_1', 'Test_2', 'Test_3', 'Test_4', 'Test_5',
       'Test_6', 'Test_7', 'Test_8', 'Test_9', 'Test_10', 'Test_11',
       'Test_12'],
      dtype='object')
```

In [45]:
```
data.fillna(value=5)
```

Out[45]:

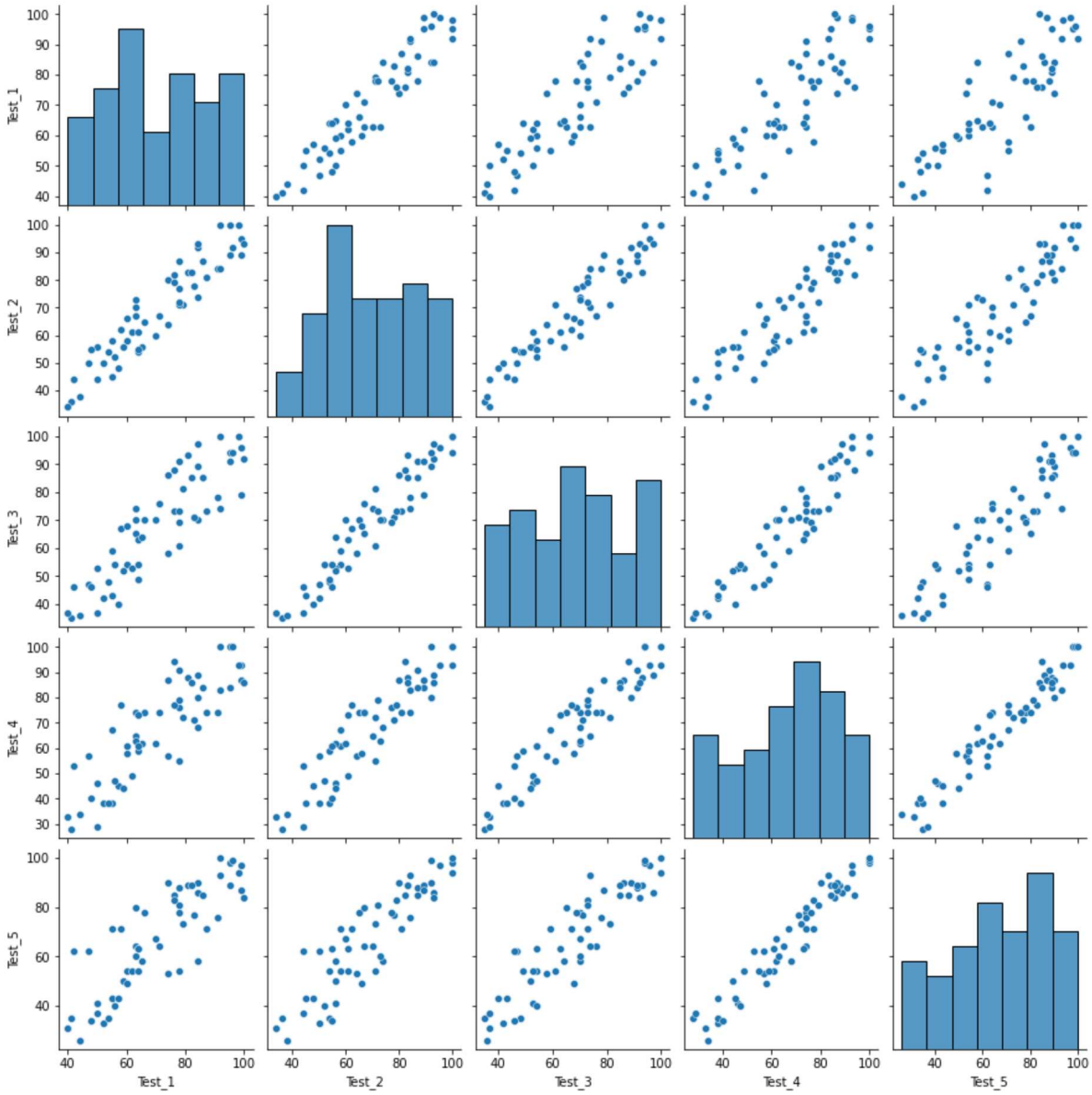
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [84]:
```
data1=data[['Test_1', 'Test_2', 'Test_3', 'Test_4', 'Test_5']]
```

In [85]:
```
sns.pairplot(data1)
```

Out[85]:
```
<seaborn.axisgrid.PairGrid at 0x1257cd70d90>
```

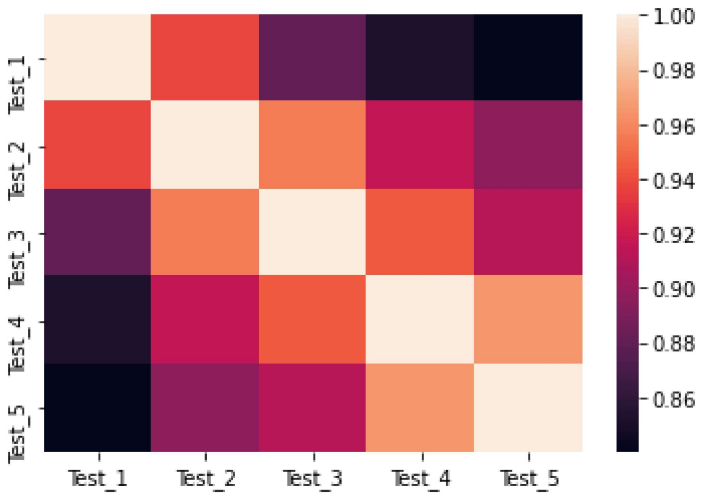


EDA and Visualization

In [86]:
```
#sns.distplot(data['Co2-Emissions'])
```

In [87]:
```
sns.heatmap(data1.corr())
```

Out[87]: <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 [111]:
```
x=data[['Test_2', 'Test_3', 'Test_4', 'Test_5',
       'Test_6', 'Test_7', 'Test_8', 'Test_9', 'Test_10', 'Test_11',
       'Test_12']]
y=data1['Test_1']
```

In [112]:
```
#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 [113]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[113]: LinearRegression()

In [114]:
```
lr.intercept_
```

Out[114]: 8.037827993311254

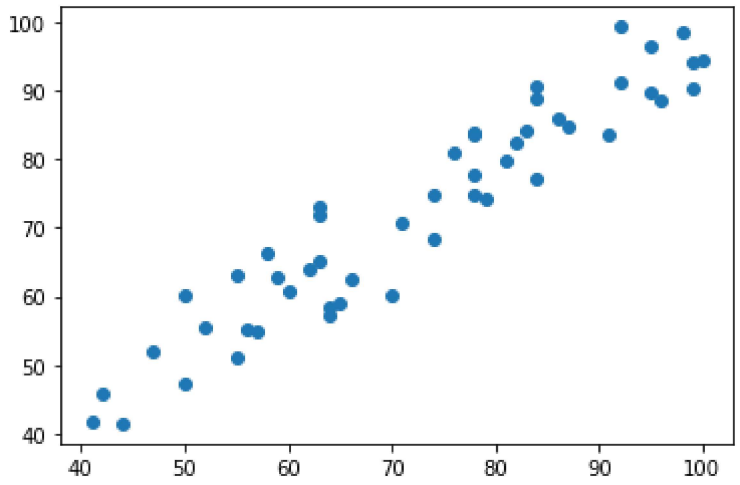
In [115]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=["Co-efficient"])
coeff
```

Out[115]:

	Co-efficient
Test_2	1.246687
Test_3	-0.180468
Test_4	-0.067759
Test_5	0.168814
Test_6	-0.084408
Test_7	-0.359143
Test_8	0.083592
Test_9	0.188246
Test_10	-0.066289
Test_11	-0.117327
Test_12	0.089670

In [116]:
```
prediction = lr.predict(x_train)
plt.scatter(y_train,prediction)
```

Out[116]: `<matplotlib.collections.PathCollection at 0x1257fe1a430>`



In [117]:
```
lr.score(x_test,y_test)
```

Out[117]: `0.5198538480792071`

In [118]:
```
lr.score(x_train,y_train)
```

Out[118]: `0.9015962733027583`

In [119]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [120]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
rr.score(x_test,y_test)
```

Out[120]:
```
0.5279347842005535
```

In [121]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
la.score(x_test,y_test)
```

Out[121]:
```
0.6895001439973342
```

In [122]:
```
from sklearn.linear_model import ElasticNet
en= ElasticNet()
en.fit(x_train,y_train)
```

Out[122]:
```
ElasticNet()
```

In [123]:
```
print(en.coef_)
```

```
[ 1.13113803 -0.12391794 -0.00637278  0.12624958 -0.08150145 -0.27237985
  0.0307049   0.1560741  -0.04507002 -0.03903421  0.01253983]
```

In [124]:
```
print(en.intercept_)
```

```
9.192973877887766
```

In [125]:
```
prediction = en.predict(x_test)
prediction
```

Out[125]:
```
array([57.62294502, 70.82109935, 61.87583691, 76.45798479, 37.45064913,
       60.8023169 ])
```

In [126]:
```
print(en.score(x_test,y_train))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-126-48d1f0543252> 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: [50,
6]
```

In [127]:
```
from sklearn import metrics
```

In [128]:
```
print("Mean Absolute error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute error: 5.754150005141302
```

In [129]:
```
print("Mean Absolute Square error:",metrics.mean_squared_error(y_test,predicti
```

```
Mean Absolute Square error: 56.61581478020323
```

In [130]:
```
print("Root mean Square error:",np.sqrt(metrics.mean_squared_error(y_test,pred
```

```
Root mean Square error: 7.524348129918181
```

In [ ]:

In [ ]: