In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

14. DataSet Iris ¶

In [2]:

```
a=pd.read_csv(r"C:\Users\user\Downloads\14_Iris.csv")
a
```

Out[2]:

	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 [3]:

```
b=a.head(10)
b
```

Out[3]:

	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
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

In [4]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [5]:

```
a.describe()
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [6]:

```
a.columns
```

Out[6]:

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidth
Cm',
       'Species'],
      dtype='object')
```

In [7]:

```
c=b[['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
c
```

Out[7]:

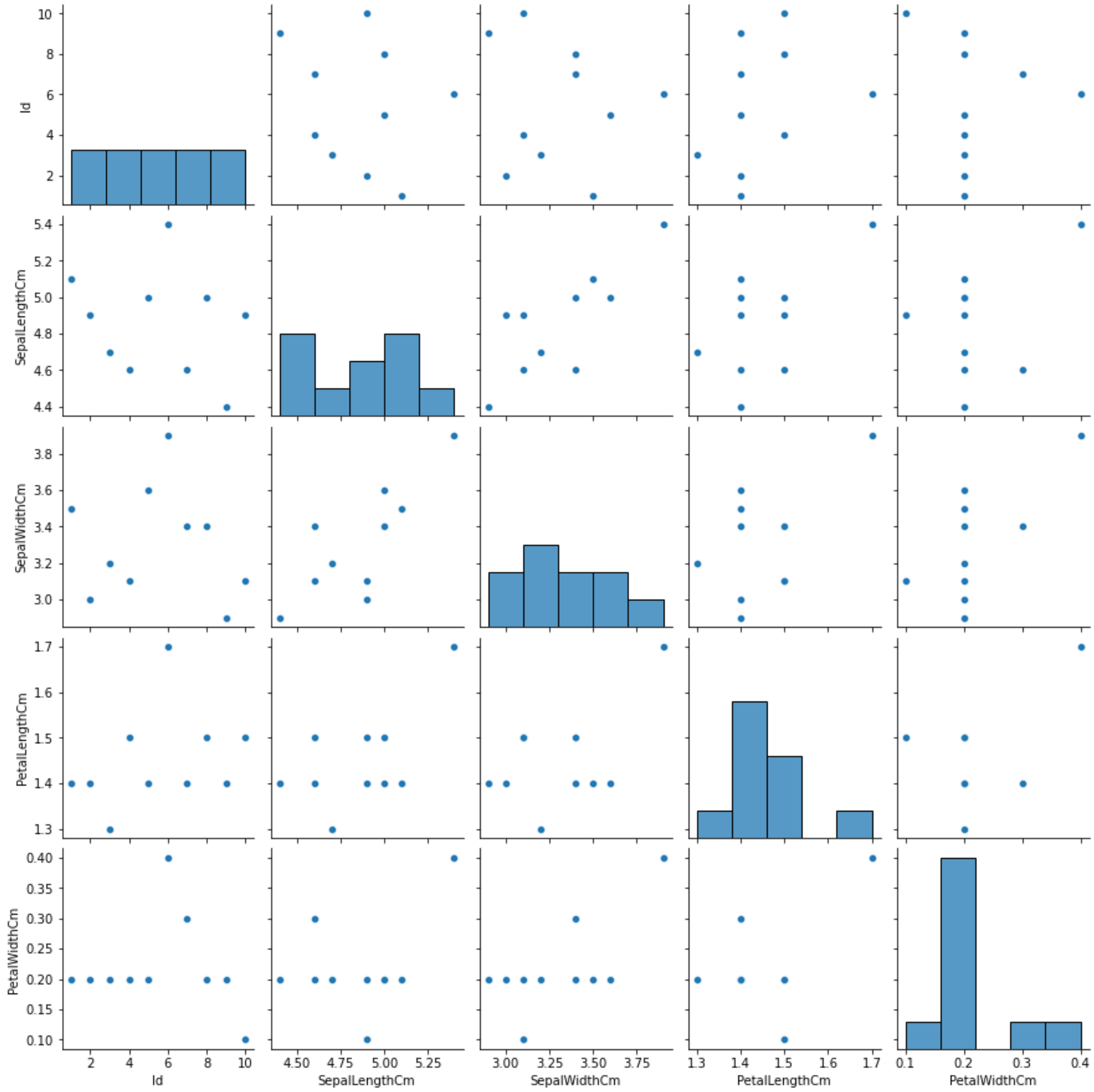
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	1	5.1	3.5	1.4	0.2
1	2	4.9	3.0	1.4	0.2
2	3	4.7	3.2	1.3	0.2
3	4	4.6	3.1	1.5	0.2
4	5	5.0	3.6	1.4	0.2
5	6	5.4	3.9	1.7	0.4
6	7	4.6	3.4	1.4	0.3
7	8	5.0	3.4	1.5	0.2
8	9	4.4	2.9	1.4	0.2
9	10	4.9	3.1	1.5	0.1

In [8]:

```
sns.pairplot(c)
```

Out[8]:

<seaborn.axisgrid.PairGrid at 0x217d2587ac0>



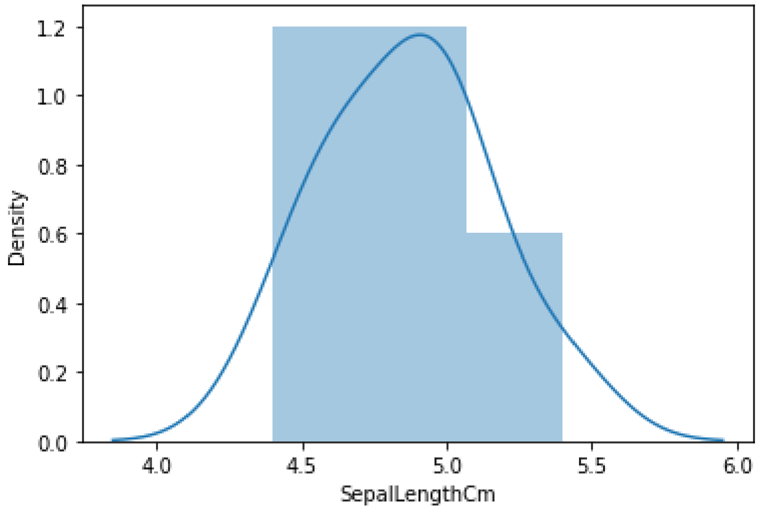
In [9]:

```
sns.distplot(c['SepalLengthCm'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated 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 fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[9]:

```
<AxesSubplot:xlabel='SepalLengthCm', ylabel='Density'>
```

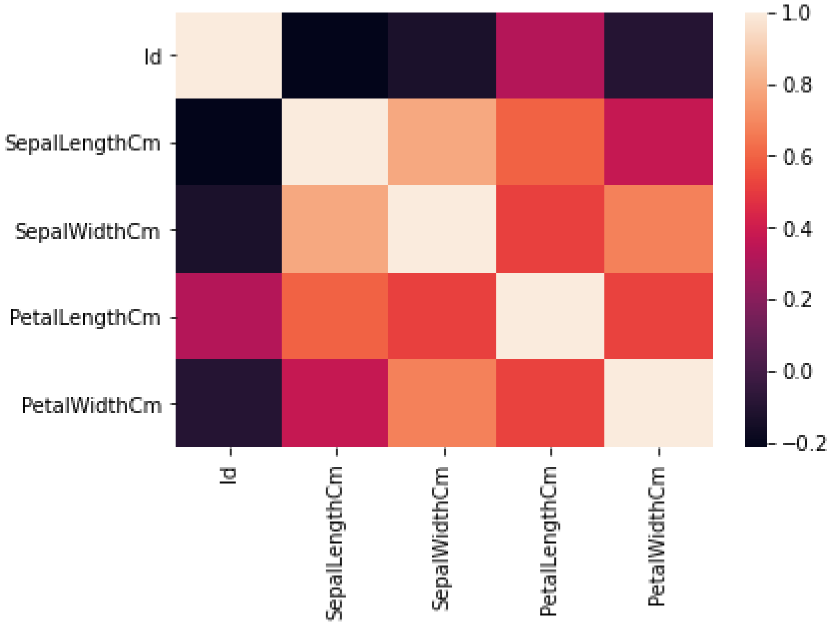


In [10]:

```
sns.heatmap(c.corr())
```

Out[10]:

```
<AxesSubplot:>
```



In [11]:

```
x=b[['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
y=b['SepalWidthCm']
```

In [12]:

```
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 [13]:

```
from sklearn.linear_model import LinearRegression

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

Out[13]:

```
LinearRegression()
```

In [14]:

```
print(lr.intercept_)
```

```
-8.881784197001252e-16
```

In [15]:

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

Out[15]:

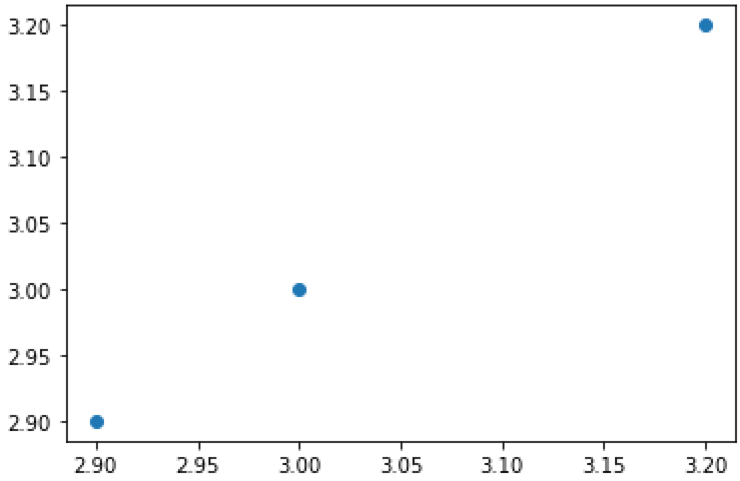
	Co-efficient
Id	0.000000e+00
SepalLengthCm	-4.724091e-18
SepalWidthCm	1.000000e+00
PetalLengthCm	-3.819093e-16
PetalWidthCm	1.138951e-16

In [16]:

```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[16]:

<matplotlib.collections.PathCollection at 0x217d45a1c40>



In [17]:

```
print(lr.score(x_test,y_test))
```

1.0

In [18]:

```
from sklearn.linear_model import Ridge,Lasso
```

In [19]:

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[19]:

Ridge(alpha=10)

In [20]:

```
rr.score(x_test,y_test)
```

Out[20]:

-9.430062712045332

In [21]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[21]:

Lasso(alpha=10)

In [22]:

```
la.score(x_test,y_test)
```

Out[22]:

```
-10.042274052478106
```

In [23]:

```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[23]:

```
ElasticNet()
```

In [24]:

```
print(en.coef_)
```

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

In [25]:

```
print(en.intercept_)
```

```
3.4285714285714284
```

In [26]:

```
prediction=en.predict(x_test)
print(en.score(x_test,y_test))
```

```
-10.042274052478106
```

Evaluation Metrics

In [27]:

```
from sklearn import metrics
```

In [28]:

```
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 0.395238095238095
```

In [29]:

```
print("mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
mean Squared Error: 0.17176870748299308
```

In [30]:

```
print("Root mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root mean Squared Error: 0.414449885369743
```

15. DataSet Horse_Racing

In [31]:

```
a=pd.read_csv(r"C:\Users\user\Downloads\15_Horse Racing Results.CSV - 15_Horse Racing Res
a
```

Out[31]:

	Dato	Track	Race Number	Distance	Surface	Prize money	Starting position	Jockey	Jockey weight	C
0	03.09.2017	Sha Tin	10	1400	Gress	1310000	6	K C Leung	52	
1	16.09.2017	Sha Tin	10	1400	Gress	1310000	14	C Y Ho	52	
2	14.10.2017	Sha Tin	10	1400	Gress	1310000	8	C Y Ho	52	
3	11.11.2017	Sha Tin	9	1600	Gress	1310000	13	Brett Prebble	54	
4	26.11.2017	Sha Tin	9	1600	Gress	1310000	9	C Y Ho	52	
...	...	...	...	...	...	...	...	...	...	
27003	14.06.2020	Sha Tin	11	1200	Gress	1450000	6	A Hamelin	59	A
27004	21.06.2020	Sha Tin	2	1200	Gress	967000	7	K C Leung	57	A
27005	21.06.2020	Sha Tin	4	1200	Gress	967000	6	Blake Shinn	57	A
27006	21.06.2020	Sha Tin	5	1200	Gress	967000	14	Joao Moreira	57	Z
27007	21.06.2020	Sha Tin	11	1200	Gress	1450000	7	C Schofield	55	Z

27008 rows × 21 columns

In [32]:

```
b=a.head(10)
b
```

Out[32]:

	Dato	Track	Race Number	Distance	Surface	Prize money	Starting position	Jockey	Jockey weight	Count
0	03.09.2017	Sha Tin	10	1400	Gress	1310000	6	K C Leung	52	Sveri
1	16.09.2017	Sha Tin	10	1400	Gress	1310000	14	C Y Ho	52	Sveri
2	14.10.2017	Sha Tin	10	1400	Gress	1310000	8	C Y Ho	52	Sveri
3	11.11.2017	Sha Tin	9	1600	Gress	1310000	13	Brett Prebble	54	Sveri
4	26.11.2017	Sha Tin	9	1600	Gress	1310000	9	C Y Ho	52	Sveri
5	10.12.2017	Sha Tin	1	1800	Gress	1310000	4	C Y Ho	52	Sveri
6	01.01.2018	Sha Tin	9	1800	Gress	1310000	9	C Schofield	54	Sveri
7	04.02.2018	Sha Tin	5	1800	Gress	1310000	6	Joao Moreira	57	Sveri
8	03.03.2018	Sha Tin	8	1800	Gress	1310000	3	C Y Ho	56	Sveri
9	11.03.2018	Sha Tin	10	1600	Gress	1310000	8	C Y Ho	57	Sveri

10 rows × 21 columns

In [33]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27008 entries, 0 to 27007
Data columns (total 21 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Dato               27008 non-null  object
 1   Track              27008 non-null  object
 2   Race Number        27008 non-null  int64 
 3   Distance           27008 non-null  int64 
 4   Surface            27008 non-null  object
 5   Prize money        27008 non-null  int64 
 6   Starting position  27008 non-null  int64 
 7   Jockey             27008 non-null  object
 8   Jockey weight      27008 non-null  int64 
 9   Country            27008 non-null  object
 10  Horse age          27008 non-null  int64 
 11  TrainerName        27008 non-null  object
 12  Race time          27008 non-null  object
 13  Path               27008 non-null  int64 
 14  Final place        27008 non-null  int64 
 15  FGrating           27008 non-null  int64 
 16  Odds               27008 non-null  object
 17  RaceType           27008 non-null  object
 18  HorseId            27008 non-null  int64 
 19  JockeyId           27008 non-null  int64 
 20  TrainerID          27008 non-null  int64 
dtypes: int64(12), object(9)
memory usage: 4.3+ MB
```

In [34]:

```
a.describe()
```

Out[34]:

	Race Number	Distance	Prize money	Starting position	Jockey weight	Horse age
count	27008.000000	27008.000000	2.700800e+04	27008.000000	27008.000000	27008.000000
mean	5.268624	1401.666173	1.479445e+06	6.741447	55.867373	5.246408
std	2.780088	276.065045	2.162109e+06	3.691071	2.737006	1.519880
min	1.000000	1000.000000	6.600000e+05	1.000000	47.000000	2.000000
25%	3.000000	1200.000000	9.200000e+05	4.000000	54.000000	4.000000
50%	5.000000	1400.000000	9.670000e+05	7.000000	56.000000	5.000000
75%	8.000000	1650.000000	1.450000e+06	10.000000	58.000000	6.000000
max	11.000000	2400.000000	2.800000e+07	14.000000	63.000000	12.000000

In [35]:

```
a.columns
```

Out[35]:

```
Index(['Dato', 'Track', 'Race Number', 'Distance', 'Surface', 'Prize mone
y',
       'Starting position', 'Jockey', 'Jockey weight', 'Country', 'Horse a
ge',
       'TrainerName', 'Race time', 'Path', 'Final place', 'FGrating', 'Odd
s',
       'RaceType', 'HorseId', 'JockeyId', 'TrainerID'],
      dtype='object')
```

In [36]:

```
c=b[['Race Number', 'Distance', 'Surface', 'Prize money',
       'Starting position']]
c
```

Out[36]:

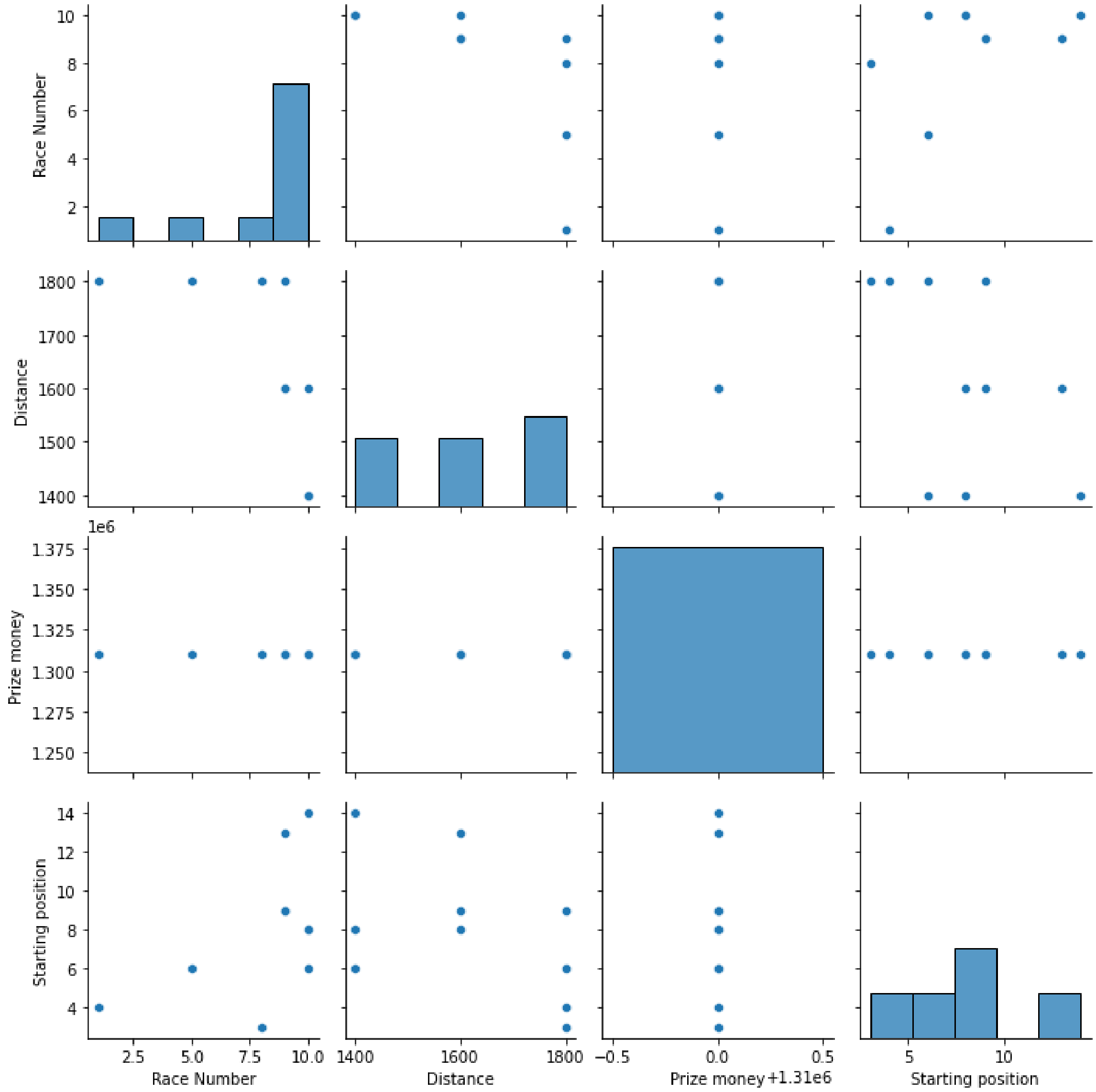
	Race Number	Distance	Surface	Prize money	Starting position
0	10	1400	Gress	1310000	6
1	10	1400	Gress	1310000	14
2	10	1400	Gress	1310000	8
3	9	1600	Gress	1310000	13
4	9	1600	Gress	1310000	9
5	1	1800	Gress	1310000	4
6	9	1800	Gress	1310000	9
7	5	1800	Gress	1310000	6
8	8	1800	Gress	1310000	3
9	10	1600	Gress	1310000	8

In [37]:

```
sns.pairplot(c)
```

Out[37]:

```
<seaborn.axisgrid.PairGrid at 0x217d4a2c160>
```



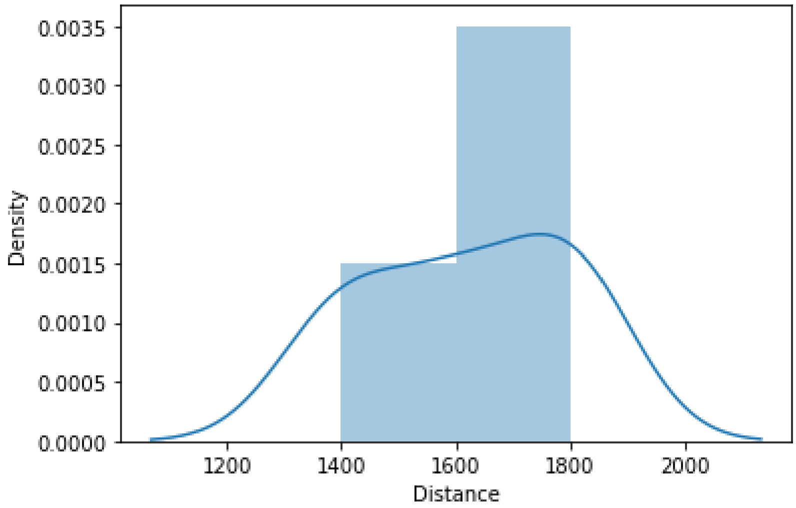
In [38]:

```
sns.distplot(c['Distance'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated 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 fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[38]:

<AxesSubplot:xlabel='Distance', ylabel='Density'>



In [39]:

```
sns.heatmap(c.corr())
```

Out[39]:

<AxesSubplot:>

In [40]:

```
x=b[['Race Number', 'Distance', 'Prize money',
       'Starting position']]
y=b['Path']
```

In [41]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [42]:

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

Out[42]:

```
LinearRegression()
```

In [43]:

```
print(lr.intercept_)
```

```
-16.134030197444815
```

In [44]:

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

Out[44]:

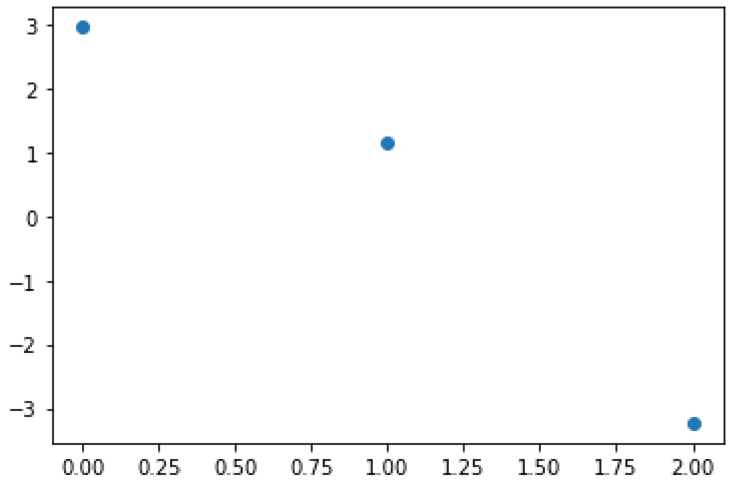
	Co-efficient
Race Number	0.660163
Distance	0.006267
Prize money	0.000000
Starting position	0.240418

In [45]:

```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[45]:

```
<matplotlib.collections.PathCollection at 0x217d57e49a0>
```



In [46]:

```
print(lr.score(x_test,y_test))
```

```
-17.078783064286586
```

In [47]:

```
from sklearn.linear_model import Ridge,Lasso
```

In [48]:

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[48]:

```
Ridge(alpha=10)
```

In [49]:

```
rr.score(x_test,y_test)
```

Out[49]:

```
-7.724029079501507
```

In [50]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[50]:

```
Lasso(alpha=10)
```

In [51]:

```
la.score(x_test,y_test)
```

Out[51]:

```
-0.8690625000000003
```

In [52]:

```
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[52]:

```
ElasticNet()
```

In [53]:

```
print(en.coef_)
```

```
[0.24550125 0.00280497 0.         0.18344861]
```

In [54]:

```
print(en.intercept_)
```

```
-6.434565790561453
```

In [55]:

```
prediction=en.predict(x_test)
print(en.score(x_test,y_test))
```

```
-5.48654541826254
```

Evaluation Metrics

In [56]:

```
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 1.8230175459888072
```

In [57]:

```
print("mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
mean Squared Error: 4.324363612175026
```

In [58]:

```
print("Root mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root mean Squared Error: 2.0795104260799047
```

16. DataSet Sleep_Health

In [59]:

```
a=pd.read_csv(r"C:\Users\user\Downloads\16_Sleep_health_and_lifestyle_dataset.csv")
a
```

Out[59]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	Pr
0	1	Male	27	Software Engineer	6.1	6	42	6	Overweight	
1	2	Male	28	Doctor	6.2	6	60	8	Normal	
2	3	Male	28	Doctor	6.2	6	60	8	Normal	
3	4	Male	28	Sales Representative	5.9	4	30	8	Obese	
4	5	Male	28	Sales Representative	5.9	4	30	8	Obese	
...	...	...	...	...	...	...	...	...	...	
369	370	Female	59	Nurse	8.1	9	75	3	Overweight	
370	371	Female	59	Nurse	8.0	9	75	3	Overweight	
371	372	Female	59	Nurse	8.1	9	75	3	Overweight	
372	373	Female	59	Nurse	8.1	9	75	3	Overweight	
373	374	Female	59	Nurse	8.1	9	75	3	Overweight	

374 rows × 13 columns

In [60]:

```
b=a.head(10)
b
```

Out[60]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category	B Pres
0	1	Male	27	Software Engineer	6.1	6	42	6	Overweight	12
1	2	Male	28	Doctor	6.2	6	60	8	Normal	12
2	3	Male	28	Doctor	6.2	6	60	8	Normal	12
3	4	Male	28	Sales Representative	5.9	4	30	8	Obese	14
4	5	Male	28	Sales Representative	5.9	4	30	8	Obese	14
5	6	Male	28	Software Engineer	5.9	4	30	8	Obese	14
6	7	Male	29	Teacher	6.3	6	40	7	Obese	14
7	8	Male	29	Doctor	7.8	7	75	6	Normal	12
8	9	Male	29	Doctor	7.8	7	75	6	Normal	12
9	10	Male	29	Doctor	7.8	7	75	6	Normal	12

In [61]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374 entries, 0 to 373
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Person ID                374 non-null    int64  
 1   Gender                   374 non-null    object 
 2   Age                      374 non-null    int64  
 3   Occupation               374 non-null    object 
 4   Sleep Duration           374 non-null    float64
 5   Quality of Sleep         374 non-null    int64  
 6   Physical Activity Level  374 non-null    int64  
 7   Stress Level             374 non-null    int64  
 8   BMI Category             374 non-null    object 
 9   Blood Pressure           374 non-null    object 
 10  Heart Rate               374 non-null    int64  
 11  Daily Steps              374 non-null    int64  
 12  Sleep Disorder           374 non-null    object 
dtypes: float64(1), int64(7), object(5)
memory usage: 38.1+ KB
```

In [62]:

```
a.describe()
```

Out[62]:

	Person ID	Age	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	Heart Rate
count	374.000000	374.000000	374.000000	374.000000	374.000000	374.000000	374.000000
mean	187.500000	42.184492	7.132086	7.312834	59.171123	5.385027	70.165775
std	108.108742	8.673133	0.795657	1.196956	20.830804	1.774526	4.135676
min	1.000000	27.000000	5.800000	4.000000	30.000000	3.000000	65.000000
25%	94.250000	35.250000	6.400000	6.000000	45.000000	4.000000	68.000000
50%	187.500000	43.000000	7.200000	7.000000	60.000000	5.000000	70.000000
75%	280.750000	50.000000	7.800000	8.000000	75.000000	7.000000	72.000000
max	374.000000	59.000000	8.500000	9.000000	90.000000	8.000000	86.000000

In [63]:

```
a.columns
```

Out[63]:

```
Index(['Person ID', 'Gender', 'Age', 'Occupation', 'Sleep Duration',
       'Quality of Sleep', 'Physical Activity Level', 'Stress Level',
       'BMI Category', 'Blood Pressure', 'Heart Rate', 'Daily Steps',
       'Sleep Disorder'],
      dtype='object')
```

In [64]:

```
c=b[['Sleep Duration',
       'Quality of Sleep', 'Physical Activity Level', 'Stress Level']]
c
```

Out[64]:

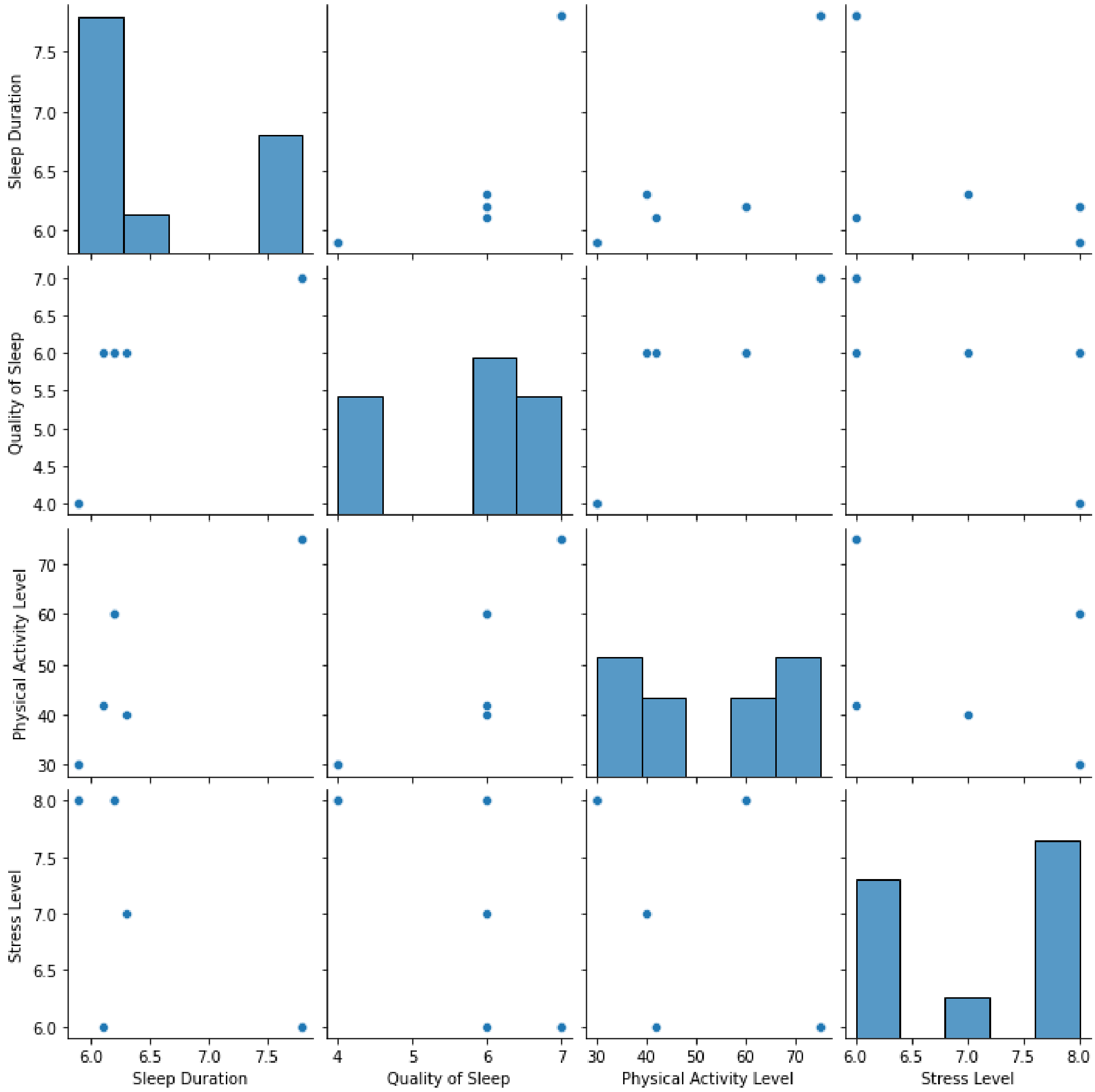
	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level
0	6.1	6	42	6
1	6.2	6	60	8
2	6.2	6	60	8
3	5.9	4	30	8
4	5.9	4	30	8
5	5.9	4	30	8
6	6.3	6	40	7
7	7.8	7	75	6
8	7.8	7	75	6
9	7.8	7	75	6

In [65]:

```
sns.pairplot(c)
```

Out[65]:

<seaborn.axisgrid.PairGrid at 0x217d583d400>



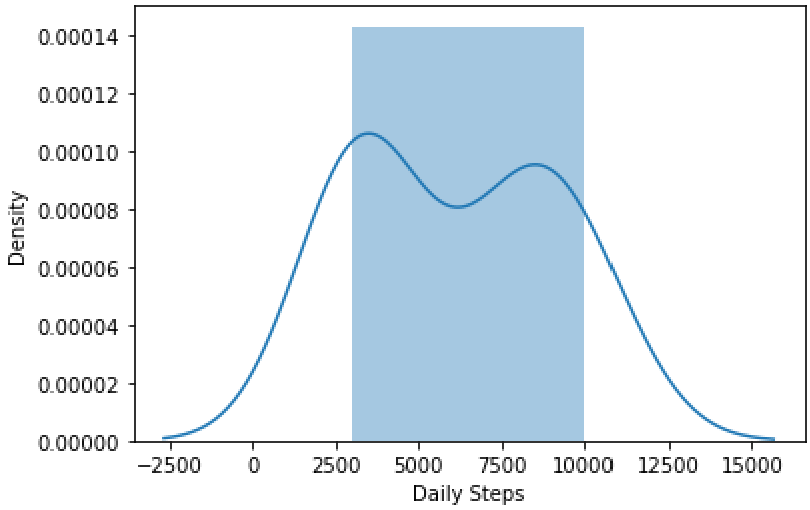
In [66]:

```
sns.distplot(b['Daily Steps'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated 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 fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[66]:

```
<AxesSubplot:xlabel='Daily Steps', ylabel='Density'>
```

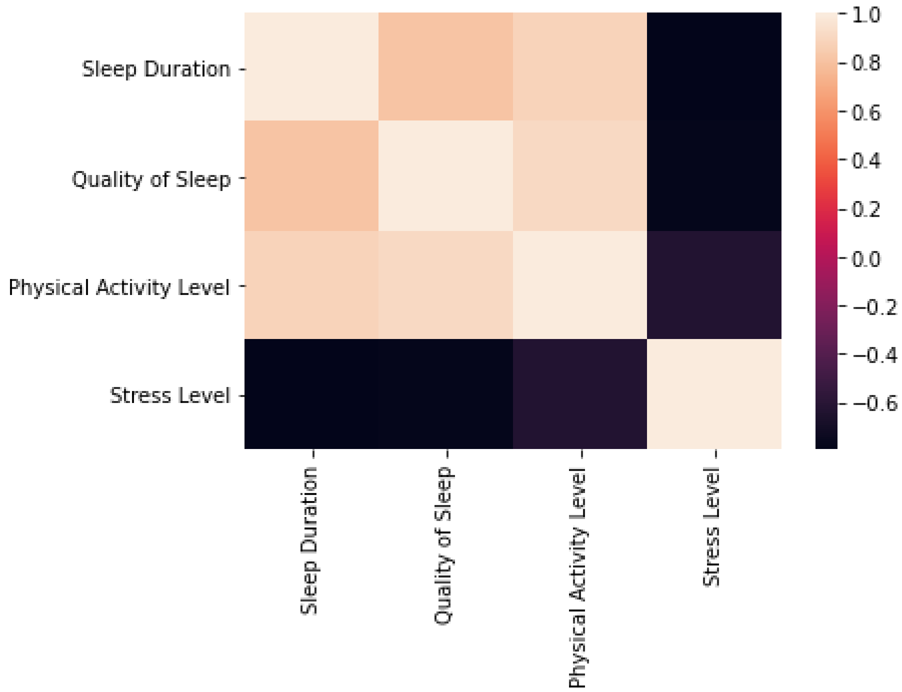


In [67]:

```
sns.heatmap(c.corr())
```

Out[67]:

```
<AxesSubplot:>
```



In [68]:

```
x=b[['Age', 'Sleep Duration',
       'Quality of Sleep', 'Physical Activity Level', 'Stress Level']]
y=b['Heart Rate']
```

In [69]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [70]:

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

Out[70]:

```
LinearRegression()
```

In [71]:

```
print(lr.intercept_)
```

```
92.64886316724176
```

In [72]:

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

Out[72]:

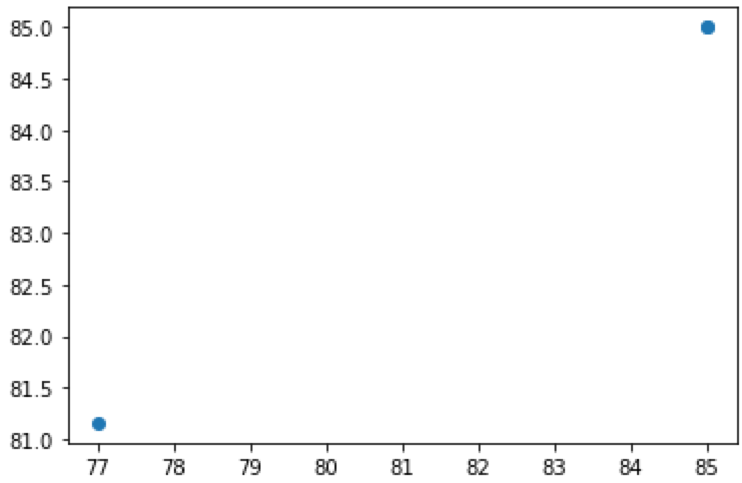
	Co-efficient
Age	0.085580
Sleep Duration	-0.068389
Quality of Sleep	0.196109
Physical Activity Level	-0.345723
Stress Level	-0.006792

In [73]:

```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[73]:

```
&lt;matplotlib.collections.PathCollection at 0x217d3cc96d0&gt;
```



In [74]:

```
print(lr.score(x_test,y_test))
```

```
0.5948164949109942
```

In [75]:

```
from sklearn.linear_model import Ridge,Lasso
```

In [76]:

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[76]:

```
Ridge(alpha=10)
```

In [77]:

```
rr.score(x_test,y_test)
```

Out[77]:

```
0.6107776395910929
```

In [78]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[78]:

```
Lasso(alpha=10)
```

In [79]:

```
la.score(x_test,y_test)
```

Out[79]:

```
0.676799228718691
```

In [80]:

```
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[80]:

```
ElasticNet()
```

In [81]:

```
print(en.coef_)
```

```
[ 0.         -0.          0.         -0.33422547 -0.        ]
```

In [82]:

```
print(en.intercept_)
```

```
95.100509964194
```

In [83]:

```
prediction=en.predict(x_test)
print(en.score(x_test,y_test))
```

```
0.6128318822957425
```

Evaluation Metrics

In [84]:

```
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 1.403510675491584
```

In [85]:

```
print("mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
mean Squared Error: 5.50639100734944
```

In [86]:

```
print("Root mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root mean Squared Error: 2.3465700516603887
```

17. DataSet Student_Marks

In [87]:

```
a=pd.read_csv(r"C:\Users\user\Downloads\17_student_marks.csv")
a
```

Out[87]:

	Student_ID	Test_1	Test_2	Test_3	Test_4	Test_5	Test_6	Test_7	Test_8	Test_9	Test_
0	22000	78	87	91	91	88	98	94	100	100	1
1	22001	79	71	81	72	73	68	59	69	59	[illegible]
2	22002	66	65	70	74	78	86	87	96	88	[illegible]
3	22003	60	58	54	61	54	57	64	62	72	[illegible]
4	22004	99	95	96	93	97	89	92	98	91	[illegible]
5	22005	41	36	35	28	35	36	27	26	19	[illegible]
6	22006	47	50	47	57	62	64	71	75	85	[illegible]
7	22007	84	74	70	68	58	59	56	56	64	[illegible]
8	22008	74	64	58	57	53	51	47	45	42	[illegible]
9	22009	87	81	73	74	71	63	53	45	39	[illegible]
10	22010	40	34	37	33	31	35	39	38	40	[illegible]
11	22011	91	84	78	74	76	80	80	73	75	[illegible]
12	22012	81	83	93	88	89	90	99	99	95	[illegible]
13	22013	52	50	42	38	33	30	28	22	12	[illegible]
14	22014	63	67	65	74	80	86	95	96	92	[illegible]
15	22015	76	82	88	94	85	76	70	60	50	[illegible]
16	22016	83	78	71	71	77	72	66	75	66	[illegible]
17	22017	55	45	43	38	43	35	44	37	45	[illegible]
18	22018	71	67	76	74	64	61	57	64	61	[illegible]
19	22019	62	61	53	49	54	59	68	74	65	[illegible]
20	22020	44	38	36	34	26	34	39	44	36	[illegible]
21	22021	50	56	53	46	41	38	47	39	44	[illegible]
22	22022	57	48	40	45	43	36	26	19	9	
23	22023	59	56	52	44	50	40	45	46	54	[illegible]
24	22024	84	92	89	80	90	80	84	74	68	[illegible]
25	22025	74	80	86	87	90	100	95	87	85	[illegible]
26	22026	92	84	74	83	93	83	75	82	81	[illegible]
27	22027	63	70	74	65	64	55	61	58	48	[illegible]
28	22028	78	77	69	76	78	74	67	69	78	[illegible]
29	22029	55	58	59	67	71	62	53	61	67	[illegible]
30	22030	54	54	48	38	35	45	46	47	41	[illegible]
31	22031	84	93	97	89	86	95	100	100	100	[illegible]
32	22032	95	100	94	100	98	99	100	90	80	[illegible]
33	22033	64	61	63	73	63	68	64	58	50	[illegible]
34	22034	76	79	73	77	83	86	95	89	90	[illegible]
35	22035	78	71	61	55	54	48	41	32	41	[illegible]
36	22036	95	89	91	84	89	94	85	91	100	1

	Student_ID	Test_1	Test_2	Test_3	Test_4	Test_5	Test_6	Test_7	Test_8	Test_9	Test_
37	22037	99	89	79	87	87	81	82	74	64	
38	22038	82	83	85	86	89	80	88	95	87	
39	22039	65	56	64	62	58	51	61	68	70	
40	22040	100	93	92	86	84	76	82	74	79	
41	22041	78	72	73	79	81	73	71	77	83	
42	22042	98	100	100	93	94	92	100	100	98	
43	22043	58	62	67	77	71	63	64	73	83	
44	22044	96	92	94	100	99	95	98	92	84	
45	22045	86	87	85	84	85	91	86	82	85	
46	22046	48	55	46	40	34	29	37	34	39	
47	22047	56	52	54	47	40	35	43	44	40	
48	22048	42	44	46	53	62	59	57	53	43	
49	22049	64	54	49	59	54	55	57	59	63	
50	22050	50	44	37	29	37	46	53	57	55	
51	22051	70	60	70	62	67	67	68	67	72	
52	22052	63	73	70	63	60	67	61	59	52	
53	22053	92	100	100	100	100	100	92	87	94	1
54	22054	64	55	54	61	63	57	47	37	44	
55	22055	60	66	68	58	49	47	39	29	39	

In [88]:

```
b=a.head(10)
b
```

Out[88]:

	Student_ID	Test_1	Test_2	Test_3	Test_4	Test_5	Test_6	Test_7	Test_8	Test_9	Test_1
0	22000	78	87	91	91	88	98	94	100	100	10
1	22001	79	71	81	72	73	68	59	69	59	6
2	22002	66	65	70	74	78	86	87	96	88	8
3	22003	60	58	54	61	54	57	64	62	72	6
4	22004	99	95	96	93	97	89	92	98	91	9
5	22005	41	36	35	28	35	36	27	26	19	2
6	22006	47	50	47	57	62	64	71	75	85	8
7	22007	84	74	70	68	58	59	56	56	64	7
8	22008	74	64	58	57	53	51	47	45	42	4
9	22009	87	81	73	74	71	63	53	45	39	4

In [89]:

```
a.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 [90]:

```
a.describe()
```

Out[90]:

	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
mean	22027.500000	70.750000	69.196429	68.089286	67.446429	67.303571	66.000000
std	16.309506	17.009356	17.712266	18.838333	19.807179	20.746890	21.054043
min	22000.000000	40.000000	34.000000	35.000000	28.000000	26.000000	29.000000
25%	22013.750000	57.750000	55.750000	53.000000	54.500000	53.750000	50.250000
50%	22027.500000	70.500000	68.500000	70.000000	71.500000	69.000000	65.500000
75%	22041.250000	84.000000	83.250000	85.000000	84.000000	85.250000	83.750000
max	22055.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000

In [91]:

```
a.columns
```

Out[91]:

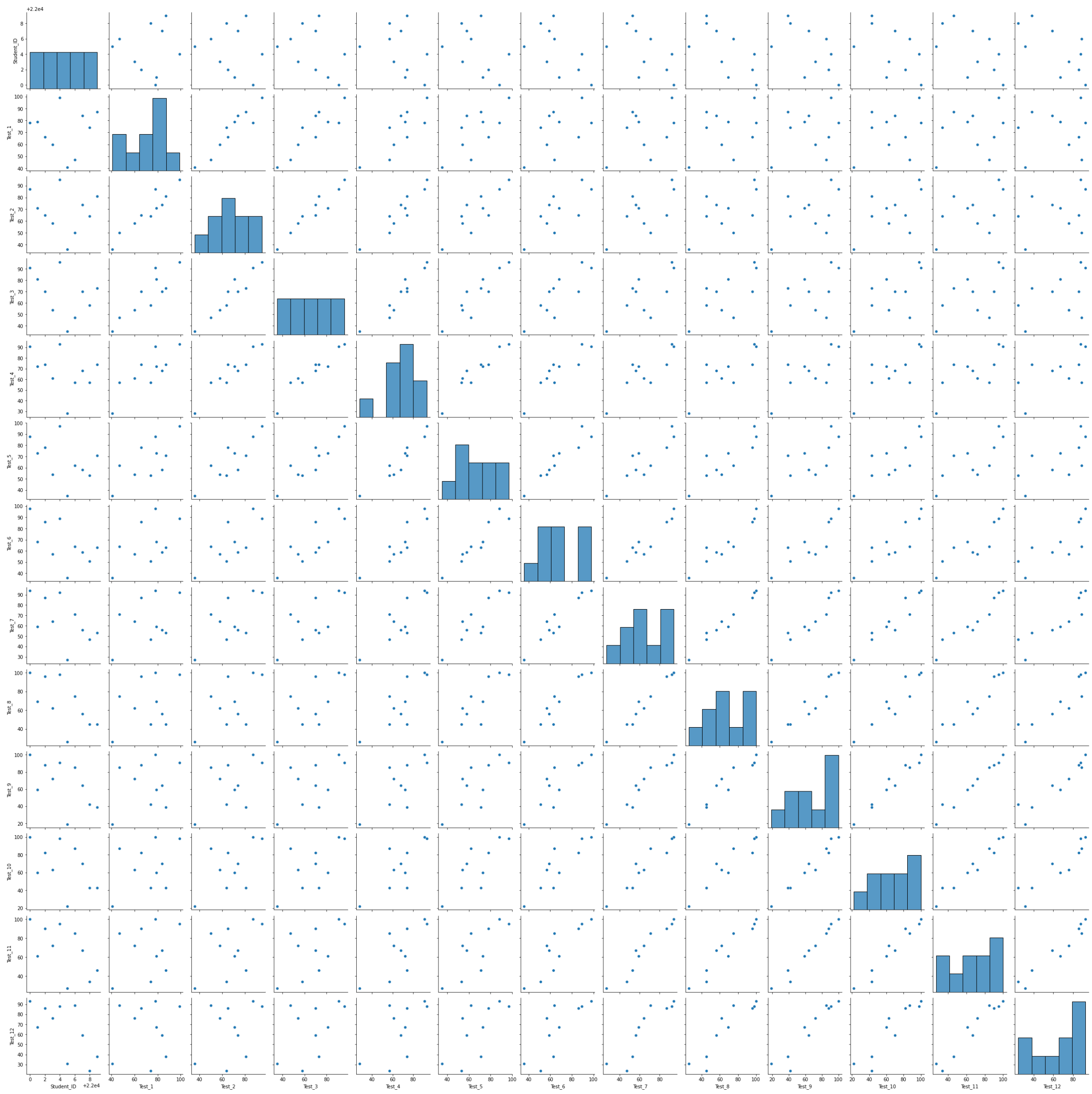
```
Index(['Student_ID', 'Test_1', 'Test_2', 'Test_3', 'Test_4', 'Test_5',
       'Test_6', 'Test_7', 'Test_8', 'Test_9', 'Test_10', 'Test_11',
       'Test_12'],
      dtype='object')
```

In [92]:

```
sns.pairplot(b)
```

Out[92]:

```
<seaborn.axisgrid.PairGrid at 0x217d65aa6d0>
```



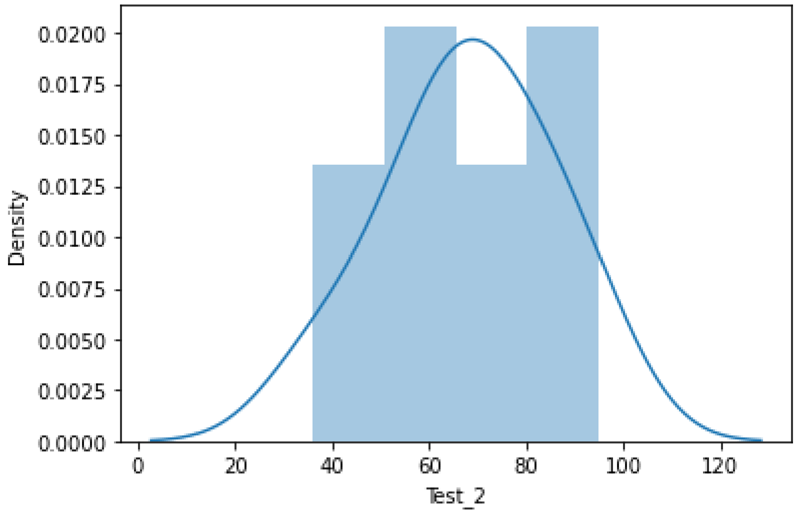
In [93]:

```
sns.distplot(b['Test_2'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated 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 fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[93]:

<AxesSubplot:xlabel='Test_2', ylabel='Density'>

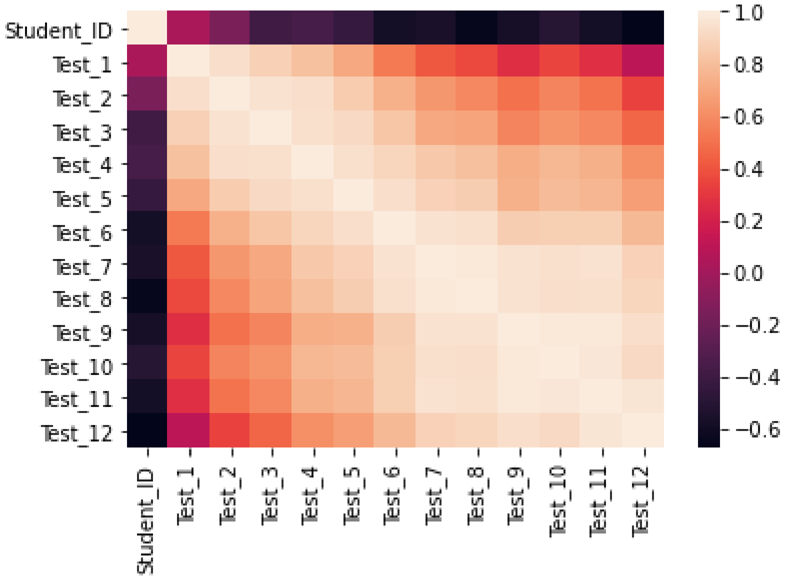


In [94]:

```
sns.heatmap(b.corr())
```

Out[94]:

<AxesSubplot:>



In [95]:

```
x=b[['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']]
y=b['Test_6']
```

In [96]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [97]:

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

Out[97]:

```
LinearRegression()
```

In [98]:

```
print(lr.intercept_)
```

```
13.200685689397119
```

In [99]:

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

Out[99]:

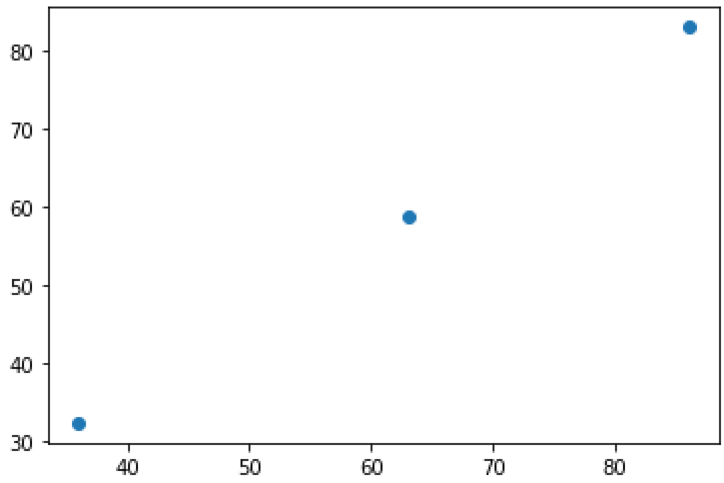
	Co-efficient
Student_ID	-0.000308
Test_1	-0.279774
Test_2	0.055882
Test_3	0.299643
Test_4	0.157631
Test_5	-0.019157
Test_6	0.500782
Test_7	0.008546
Test_8	0.139421
Test_9	0.108753
Test_10	0.073561
Test_11	-0.014964
Test_12	-0.128484

In [100]:

```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[100]:

```
<matplotlib.collections.PathCollection at 0x217df4ccd60>
```



In [101]:

```
print(lr.score(x_test,y_test))
```

```
0.9695653774078994
```

In [102]:

```
from sklearn.linear_model import Ridge,Lasso
```

In [103]:

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[103]:

```
Ridge(alpha=10)
```

In [104]:

```
rr.score(x_test,y_test)
```

Out[104]:

```
0.9644727349653939
```

In [105]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[105]:

```
Lasso(alpha=10)
```

In [106]:

```
la.score(x_test,y_test)
```

Out[106]:

```
0.9988004699481617
```

In [107]:

```
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[107]:

```
ElasticNet()
```

In [108]:

```
print(en.coef_)
```

```
[-0.         -0.          0.          0.01747778  0.          0.
  0.93487531  0.01167405  0.03041345  0.          0.          0.
 -0.        ]
```

In [109]:

```
print(en.intercept_)
```

```
0.28096938877563105
```

In [110]:

```
prediction=en.predict(x_test)
print(en.score(x_test,y_test))
```

```
0.9996346626033297
```

Evaluation Metrics

In [111]:

```
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 0.35516758709910806
```

In [112]:

```
print("mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
mean Squared Error: 0.15254865963191833
```

In [113]:

```
print("Root mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root mean Squared Error: 0.39057478110077504
```

18. DataSet World_Data

In [114]:

```
a=pd.read_csv(r"C:\Users\user\Downloads\18_world-data-2023.csv")
a
```

Out[114]:

	Country	Density\n(P/Km2)	Abbreviation	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Call Co
0	Afghanistan	60	AF	58.10%	652,230	323,000	32.49	9
1	Albania	105	AL	43.10%	28,748	9,000	11.78	35
2	Algeria	18	DZ	17.40%	2,381,741	317,000	24.28	21
3	Andorra	164	AD	40.00%	468	NaN	7.20	37
4	Angola	26	AO	47.50%	1,246,700	117,000	40.73	24
...	...	...	...	...	...	...	...	
190	Venezuela	32	VE	24.50%	912,050	343,000	17.88	5
191	Vietnam	314	VN	39.30%	331,210	522,000	16.75	8
192	Yemen	56	YE	44.60%	527,968	40,000	30.45	96
193	Zambia	25	ZM	32.10%	752,618	16,000	36.19	26
194	Zimbabwe	38	ZW	41.90%	390,757	51,000	30.68	26

195 rows × 35 columns

In [115]:

```
b=a.fillna(value=51)
b
```

Out[115]:

	Country	Density\n(P/Km2)	Abbreviation	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Call Co
0	Afghanistan	60	AF	58.10%	652,230	323,000	32.49	9
1	Albania	105	AL	43.10%	28,748	9,000	11.78	35
2	Algeria	18	DZ	17.40%	2,381,741	317,000	24.28	21
3	Andorra	164	AD	40.00%	468	51	7.20	37
4	Angola	26	AO	47.50%	1,246,700	117,000	40.73	24
...	...	...	...	...	...	...	...	
190	Venezuela	32	VE	24.50%	912,050	343,000	17.88	5
191	Vietnam	314	VN	39.30%	331,210	522,000	16.75	8
192	Yemen	56	YE	44.60%	527,968	40,000	30.45	96
193	Zambia	25	ZM	32.10%	752,618	16,000	36.19	26
194	Zimbabwe	38	ZW	41.90%	390,757	51,000	30.68	26

195 rows × 35 columns

In [116]:

```
c=b.head(10)
c
```

Out[116]:

	Country	Density\n(P/Km2)	Abbreviation	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Callin Cod
0	Afghanistan	60	AF	58.10%	652,230	323,000	32.49	93.
1	Albania	105	AL	43.10%	28,748	9,000	11.78	355.
2	Algeria	18	DZ	17.40%	2,381,741	317,000	24.28	213.
3	Andorra	164	AD	40.00%	468	51	7.20	376.
4	Angola	26	AO	47.50%	1,246,700	117,000	40.73	244.
5	Antigua and Barbuda	223	AG	20.50%	443	0	15.33	1.
6	Argentina	17	AR	54.30%	2,780,400	105,000	17.02	54.
7	Armenia	104	AM	58.90%	29,743	49,000	13.99	374.
8	Australia	3	AU	48.20%	7,741,220	58,000	12.60	61.
9	Austria	109	AT	32.40%	83,871	21,000	9.70	43.

10 rows × 35 columns

In [117]:

```
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 35 columns):
 #   Column                                    Non-Null Count  Dtype  
---  ------                                    --------------  -----  
 0   Country                                   195 non-null    object 
 1   Density
(P/Km2)                          195 non-null    object 
 2   Abbreviation                              188 non-null    object 
 3   Agricultural Land( %)                     188 non-null    object 
 4   Land Area(Km2)                            194 non-null    object 
 5   Armed Forces size                         171 non-null    object 
 6   Birth Rate                                189 non-null    float64
 7   Calling Code                              194 non-null    float64
 8   Capital/Major City                        192 non-null    object 
 9   Co2-Emissions                             188 non-null    object 
 10  CPI                                       178 non-null    object 
 11  CPI Change (%)                            179 non-null    object 
 12  Currency-Code                             180 non-null    object 
 13  Fertility Rate                            188 non-null    float64
 14  Forested Area (%)                         188 non-null    object 
 15  Gasoline Price                            175 non-null    object 
 16  GDP                                       193 non-null    object 
 17  Gross primary education enrollment (%)    188 non-null    object 
 18  Gross tertiary education enrollment (%)   183 non-null    object 
 19  Infant mortality                          189 non-null    float64
 20  Largest city                              189 non-null    object 
 21  Life expectancy                           187 non-null    float64
 22  Maternal mortality ratio                  181 non-null    float64
 23  Minimum wage                              150 non-null    object 
 24  Official language                         194 non-null    object 
 25  Out of pocket health expenditure          188 non-null    object 
 26  Physicians per thousand                   188 non-null    float64
 27  Population                                194 non-null    object 
 28  Population: Labor force participation (%)  176 non-null    object 
 29  Tax revenue (%)                           169 non-null    object 
 30  Total tax rate                            183 non-null    object 
 31  Unemployment rate                         176 non-null    object 
 32  Urban_population                          190 non-null    object 
 33  Latitude                                  194 non-null    float64
 34  Longitude                                 194 non-null    float64
dtypes: float64(9), object(26)
memory usage: 53.4+ KB
```

In [118]:

```
a.describe()
```

Out[118]:

	Birth Rate	Calling Code	Fertility Rate	Infant mortality	Life expectancy	Maternal mortality ratio	Physicians per thousand
count	189.000000	194.000000	188.000000	189.000000	187.000000	181.000000	188.000000
mean	20.214974	360.546392	2.698138	21.332804	72.279679	160.392265	1.839840
std	9.945774	323.236419	1.282267	19.548058	7.483661	233.502024	1.684261
min	5.900000	1.000000	0.980000	1.400000	52.800000	2.000000	0.010000
25%	11.300000	82.500000	1.705000	6.000000	67.000000	13.000000	0.332500
50%	17.950000	255.500000	2.245000	14.000000	73.200000	53.000000	1.460000
75%	28.750000	506.750000	3.597500	32.700000	77.500000	186.000000	2.935000
max	46.080000	1876.000000	6.910000	84.500000	85.400000	1150.000000	8.420000

In [119]:

```
c.columns
```

Out[119]:

```
Index(['Country', 'Density\n(P/Km2)', 'Abbreviation', 'Agricultural Land(
%)',
       'Land Area(Km2)', 'Armed Forces size', 'Birth Rate', 'Calling Cod
e',
       'Capital/Major City', 'Co2-Emissions', 'CPI', 'CPI Change (%)',
       'Currency-Code', 'Fertility Rate', 'Forested Area (%)',
       'Gasoline Price', 'GDP', 'Gross primary education enrollment (%)',
       'Gross tertiary education enrollment (%)', 'Infant mortality',
       'Largest city', 'Life expectancy', 'Maternal mortality ratio',
       'Minimum wage', 'Official language', 'Out of pocket health expendit
ure',
       'Physicians per thousand', 'Population',
       'Population: Labor force participation (%)', 'Tax revenue (%)',
       'Total tax rate', 'Unemployment rate', 'Urban_population', 'Latitud
e',
       'Longitude'],
      dtype='object')
```

In [120]:

```
d=c[['Density\n(P/Km2)','Urban_population', 'Latitude',
       'Longitude']]
d
```

Out[120]:

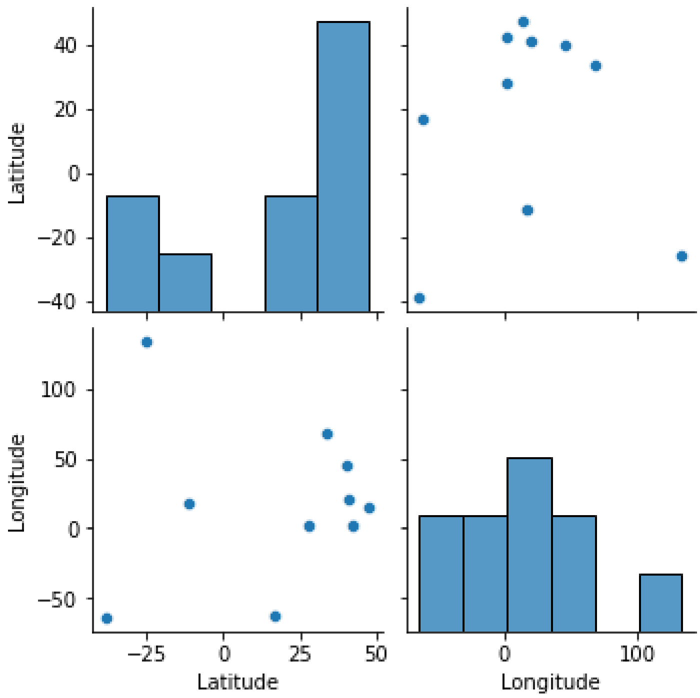
	Density\n(P/Km2)	Urban_population	Latitude	Longitude
0	60	9,797,273	33.939110	67.709953
1	105	1,747,593	41.153332	20.168331
2	18	31,510,100	28.033886	1.659626
3	164	67,873	42.506285	1.521801
4	26	21,061,025	-11.202692	17.873887
5	223	23,800	17.060816	-61.796428
6	17	41,339,571	-38.416097	-63.616672
7	104	1,869,848	40.069099	45.038189
8	3	21,844,756	-25.274398	133.775136
9	109	5,194,416	47.516231	14.550072

In [121]:

```
sns.pairplot(d)
```

Out[121]:

```
<seaborn.axisgrid.PairGrid at 0x217e09599a0>
```



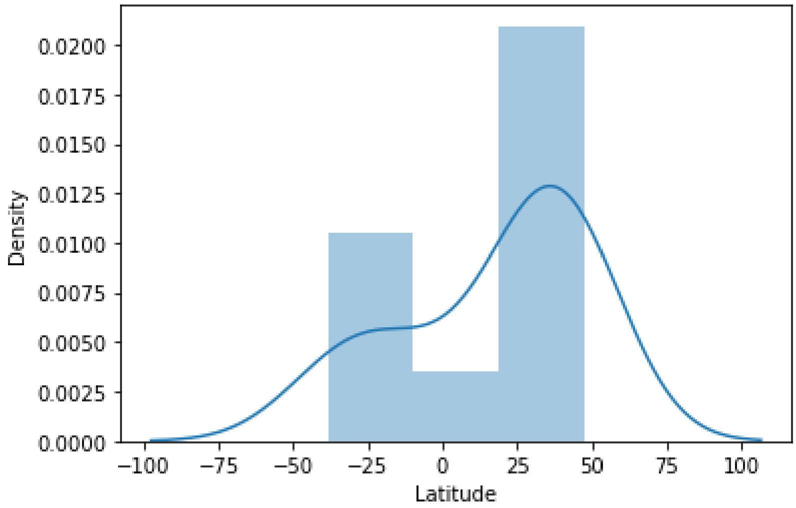
In [122]:

```
sns.distplot(d['Latitude'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated 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 fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[122]:

```
<AxesSubplot:xlabel='Latitude', ylabel='Density'>
```

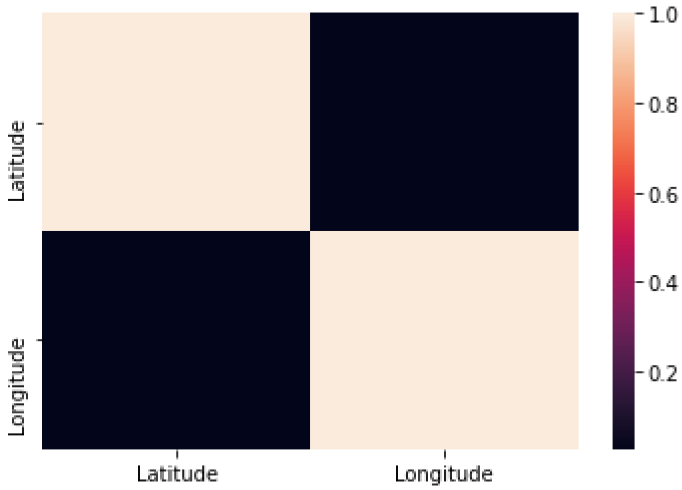


In [123]:

```
sns.heatmap(d.corr())
```

Out[123]:

```
<AxesSubplot:>
```



In [124]:

```
x=d[['Density\n(P/Km2)','Latitude',
       'Longitude']]
y=d['Latitude']
```

In [125]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [126]:

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

Out[126]:

```
LinearRegression()
```

In [127]:

```
print(lr.intercept_)
```

```
-1.0658141036401503e-14
```

In [128]:

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

Out[128]:

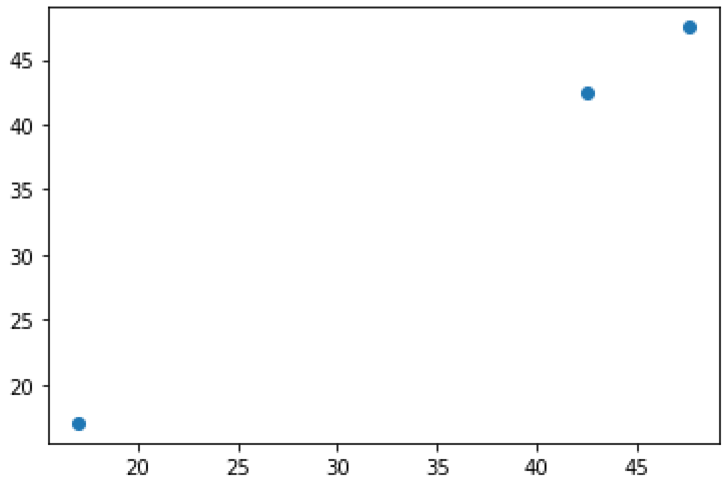
	Co-efficient
Density\n(P/Km2)	2.906770e-16
Latitude	1.000000e+00
Longitude	-5.332036e-17

In [129]:

```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[129]:

<matplotlib.collections.PathCollection at 0x217e0c2fcd0>



In [130]:

```
print(lr.score(x_test,y_test))
```

1.0

In [131]:

```
from sklearn.linear_model import Ridge,Lasso
```

In [132]:

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[132]:

Ridge(alpha=10)

In [133]:

```
rr.score(x_test,y_test)
```

Out[133]:

0.9996940751514699

In [134]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[134]:

Lasso(alpha=10)

In [135]:

```
la.score(x_test,y_test)
```

Out[135]:

```
0.9994946274260796
```

In [136]:

```
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[136]:

```
ElasticNet()
```

In [137]:

```
print(en.coef_)
```

```
[9.68190444e-04 9.98012216e-01 0.00000000e+00]
```

In [138]:

```
print(en.intercept_)
```

```
-0.02666247441751146
```

In [139]:

```
prediction=en.predict(x_test)
print(en.score(x_test,y_test))
```

```
0.9999500554619002
```

Evaluation Metrics

In [140]:

```
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 0.07284664765712161
```

In [141]:

```
print("mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
mean Squared Error: 0.008879604769335657
```

In [142]:

```
print("Root mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root mean Squared Error: 0.0942316548158614
```

In [ ]:

```

```