Linear Regression

Data1

In [95]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [96]:

```
data1 = pd.read_csv(r"C:\Users\DELL\Downloads\1_2015.csv")
```

In [97]:

```
data1.describe()
```

Out[97]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615	0.143422	0.237296	2.098977
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693	0.120034	0.126685	0.553550
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.328580
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330	0.061675	0.150553	1.759410
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515	0.107220	0.216130	2.095415
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092	0.180255	0.309883	2.462415
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730	0.551910	0.795880	3.602140

In [98]:

```
data1.head()
```

Out[98]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	0.41978	0.29678	2.51738
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	0.14145	0.43630	2.70201
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	0.48357	0.34139	2.49204
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	0.36503	0.34699	2.46531
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	0.32957	0.45811	2.45176

In [99]:

```
data1.tail()
```

Out[99]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generosity	Dystopia Residual
153	Rwanda	Sub-Saharan Africa	154	3.465	0.03464	0.22208	0.77370	0.42864	0.59201	0.55191	0.22628	0.67042
154	Benin	Sub-Saharan Africa	155	3.340	0.03656	0.28665	0.35386	0.31910	0.48450	0.08010	0.18260	1.63328
155	Syria	Middle East and Northern Africa	156	3.006	0.05015	0.66320	0.47489	0.72193	0.15684	0.18906	0.47179	0.32858
156	Burundi	Sub-Saharan Africa	157	2.905	0.08658	0.01530	0.41587	0.22396	0.11850	0.10062	0.19727	1.83302
157	Togo	Sub-Saharan Africa	158	2.839	0.06727	0.20868	0.13995	0.28443	0.36453	0.10731	0.16681	1.56726

In [100…

```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158 entries, 0 to 157
Data columns (total 12 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Country                        158 non-null    object
 1   Region                         158 non-null    object
 2   Happiness Rank                 158 non-null    int64
 3   Happiness Score                158 non-null    float64
 4   Standard Error                 158 non-null    float64
 5   Economy (GDP per Capita)       158 non-null    float64
 6   Family                         158 non-null    float64
 7   Health (Life Expectancy)       158 non-null    float64
 8   Freedom                        158 non-null    float64
 9   Trust (Government Corruption)  158 non-null    float64
 10  Generosity                     158 non-null    float64
 11  Dystopia Residual              158 non-null    float64
dtypes: float64(9), int64(1), object(2)
memory usage: 14.9+ KB
```

In [101…

```
data1.columns
```

Out[101…

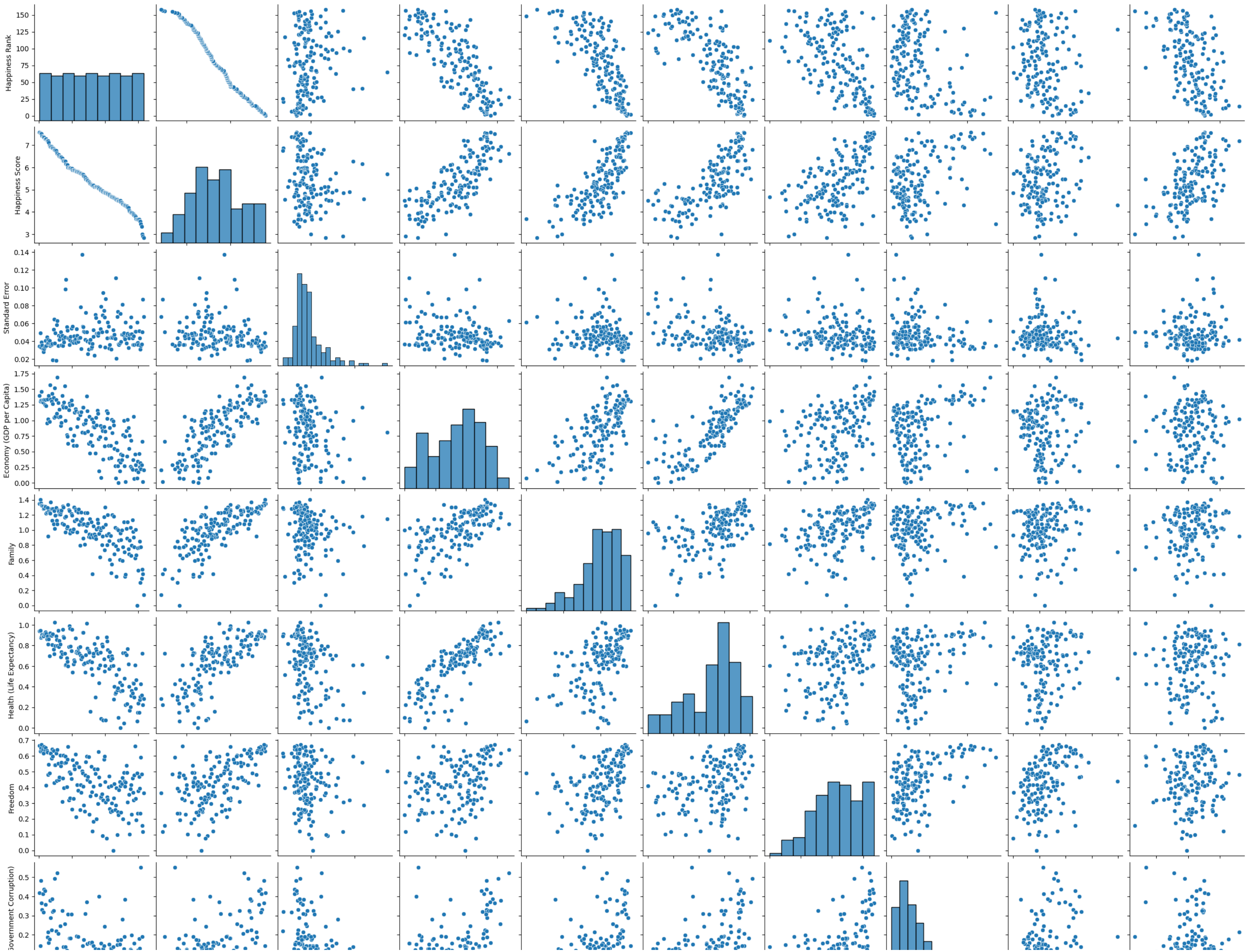
```
Index(['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual'],
      dtype='object')
```

In [102…

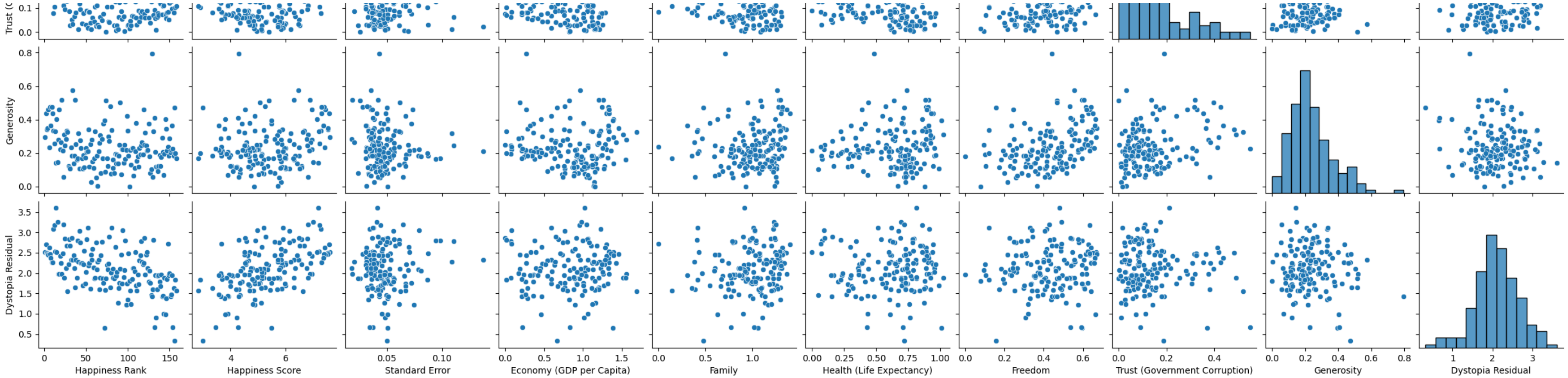
```
sns.pairplot(data1)
```

Out[102…

```
<seaborn.axisgrid.PairGrid at 0x1915c2b6a10>
```



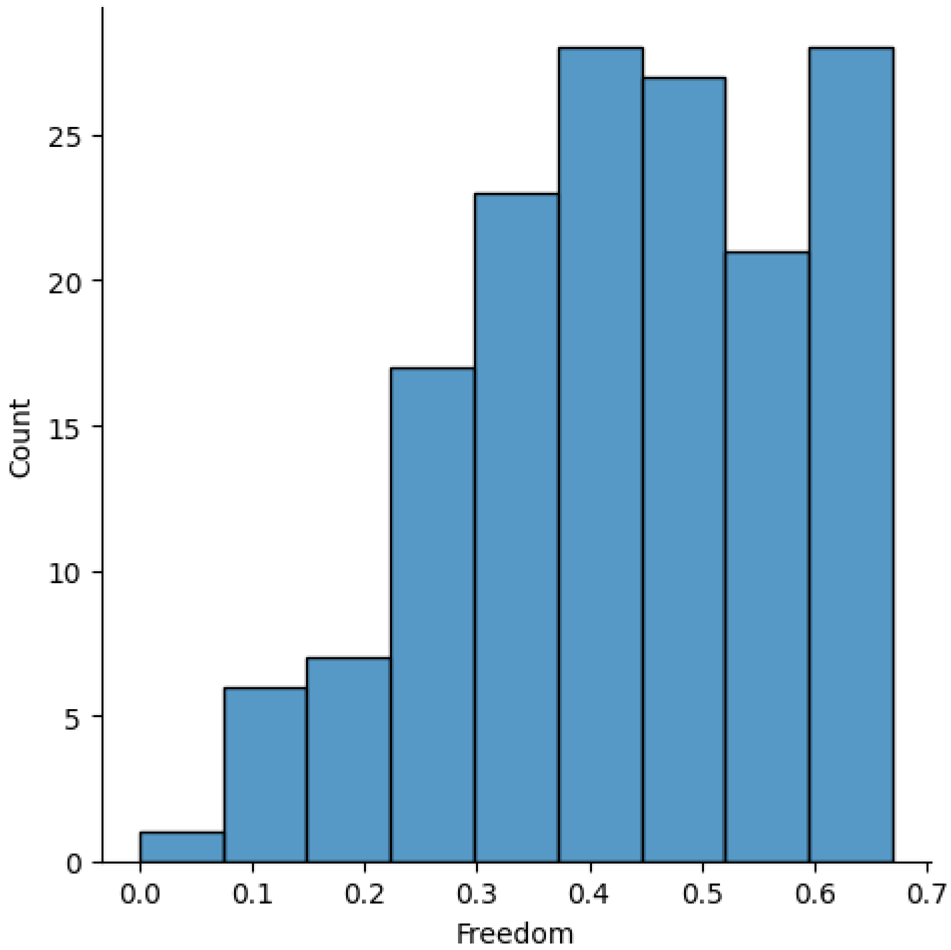
Happiness Rank
Happiness Score
Standard Error
Economy (GDP per Capita)
Family
Health (Life Expectancy)
Freedom
Government Corruption)



In [103…

```
sns.displot(data1['Freedom'])
```

Out[103… <seaborn.axisgrid.FacetGrid at 0x191612abcd0>



In [104…

```
new_data1 = data1[['Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual']]
```

In [105…

```
sns.heatmap(new_data1.corr())
```

Out[105…

```
<Axes: >
```

Model Building for data1

In [106…

```
X = new_data1[['Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual']]
y = data1['Freedom']
```

In [107…

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30)
```

In [108…

```
from sklearn.linear_model import LinearRegression

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

Out[108…

```
▾ LinearRegression
LinearRegression()
```

In [109…

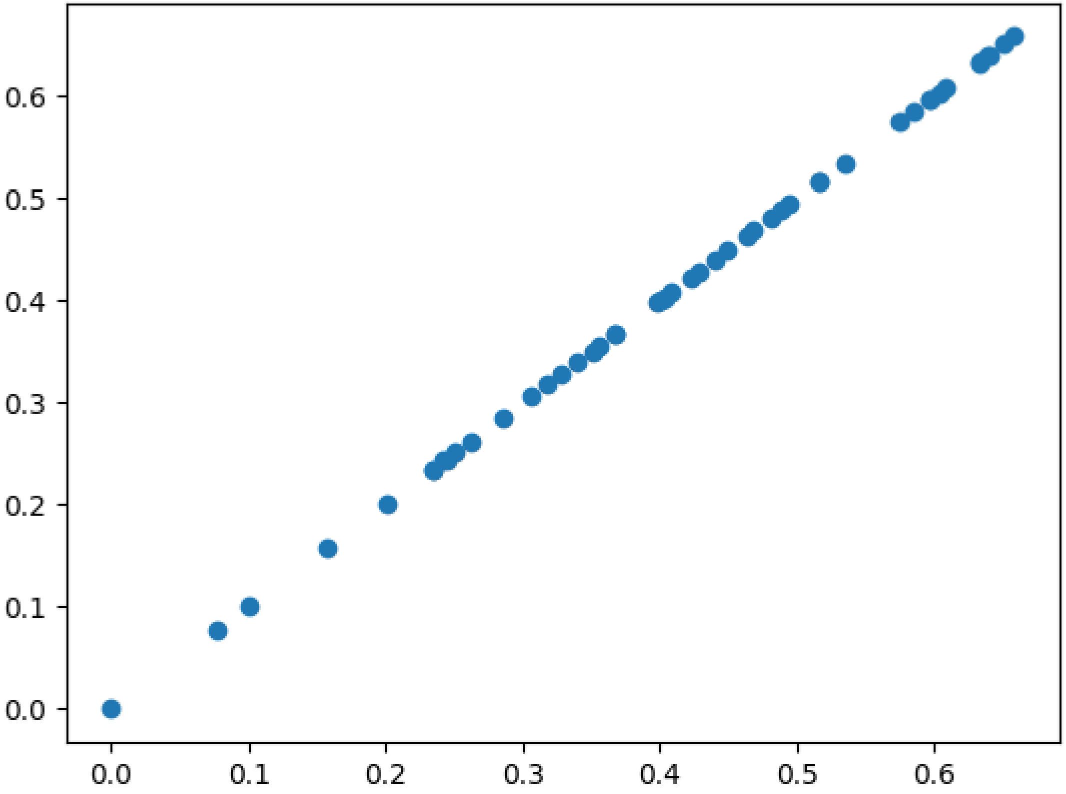
```
#Prediction
predX = lr.predict(X_test)
print(predX)
```

```
[ 3.50407110e-01  2.44321421e-01  1.00084532e-01  3.38894543e-01
  5.74406594e-01  4.81495985e-01  3.98253082e-01  3.66877443e-01
  4.63518325e-01  6.32808900e-01  4.01515420e-01  4.22991209e-01
  6.40115426e-01  2.51400993e-01 -5.13552999e-04  6.58497821e-01
  4.08211552e-01  3.55978398e-01  1.56400903e-01  6.03509006e-01
  4.40255808e-01  2.34561294e-01  5.96088111e-01  6.39404358e-01
  4.89078180e-01  5.84163166e-01  2.00812246e-01  2.62036660e-01
  3.67982987e-01  3.06310627e-01  2.34776412e-01  6.51279830e-01
  6.33118993e-01  4.94933368e-01  4.03251358e-01  3.17292650e-01
  6.08913517e-01  2.42704504e-01  4.28763410e-01  3.28104681e-01
  2.84746087e-01  5.96281492e-01  4.68650295e-01  4.49628123e-01
  5.34870933e-01  5.16594173e-01  5.16275924e-01  7.62404792e-02]
```

In [110…

```
#Accuracy
print(lr.score(X_test,y_test))
```

```
0.999996578194038
```

In [111…
```
plt.scatter(y_test,predX)
```
Out[111…
```
<matplotlib.collections.PathCollection at 0x1916362db50>
```


Data2

In [112…
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```
In [113…
```
data2 = pd.read_csv(r"C:\Users\DELL\Downloads\4_Drug200.csv")
```

In [114…

```
data2.describe()
```

Out[114…

	Age	Na_to_K
count	200.000000	200.000000
mean	44.315000	16.084485
std	16.544315	7.223956
min	15.000000	6.269000
25%	31.000000	10.445500
50%	45.000000	13.936500
75%	58.000000	19.380000
max	74.000000	38.247000

In [115…

```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          200 non-null    int64  
 1   Sex          200 non-null    object 
 2   BP           200 non-null    object 
 3   Cholesterol  200 non-null    object 
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object 
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

In [116…

```
data2.head()
```

Out[116...

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY

In [117...

```
data2.tail()
```

Out[117...

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
195	56	F	LOW	HIGH	11.567	drugC
196	16	M	LOW	HIGH	12.006	drugC
197	52	M	NORMAL	HIGH	9.894	drugX
198	23	M	NORMAL	NORMAL	14.020	drugX
199	40	F	LOW	NORMAL	11.349	drugX

In [118...

```
data2.columns
```

Out[118...

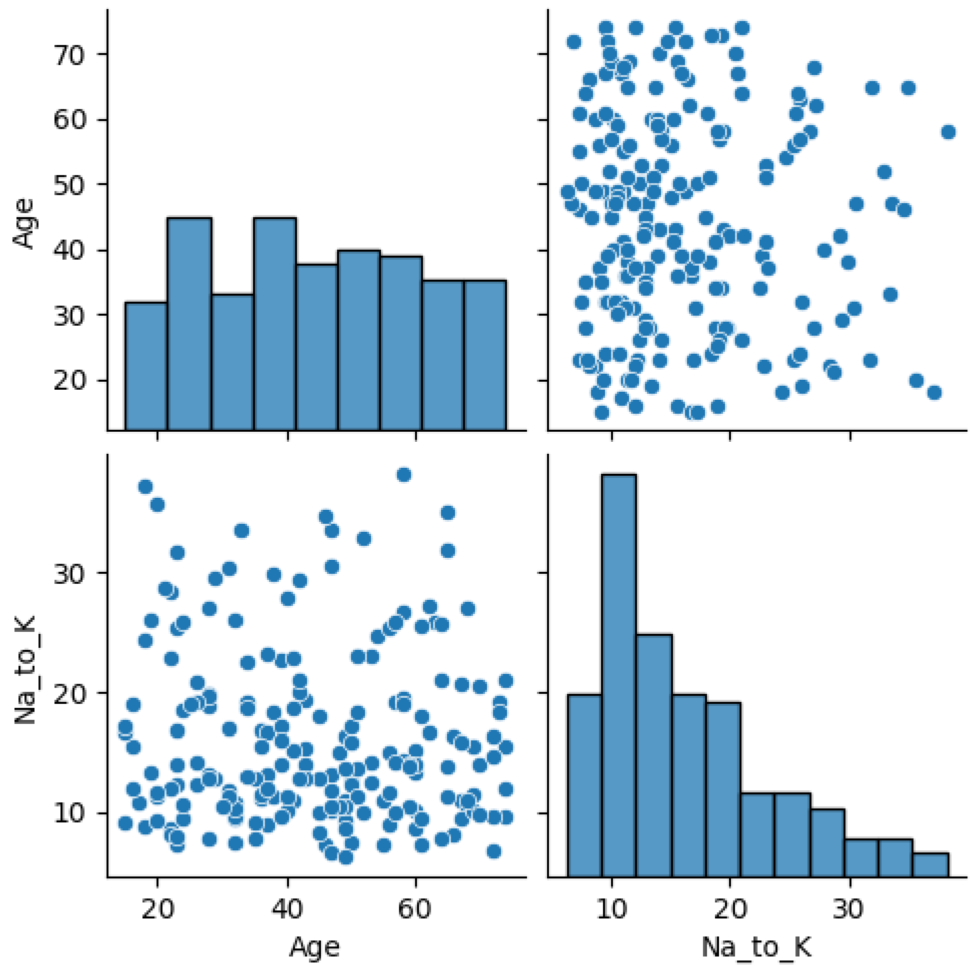
```
Index(['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K', 'Drug'], dtype='object')
```

In [119...

```
sns.pairplot(data2)
```

Out[119...

```
<seaborn.axisgrid.PairGrid at 0x191618b9450>
```



In [120…

```
#Changing High, Normal and Low to 2, 1 and 0 respectively...
BP = {"BP":{"LOW":0,"NORMAL":1,"HIGH":2}}
data2 = data2.replace(BP)
Cholesterol = {"Cholesterol":{"LOW":0,"NORMAL":1,"HIGH":2}}
data2 = data2.replace(Cholesterol)
data2
```

Out[120…

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	2	2	25.355	drugY
1	47	M	0	2	13.093	drugC
2	47	M	0	2	10.114	drugC
3	28	F	1	2	7.798	drugX
4	61	F	0	2	18.043	drugY
...	...	...	...	...	...	...
195	56	F	0	2	11.567	drugC
196	16	M	0	2	12.006	drugC
197	52	M	1	2	9.894	drugX
198	23	M	1	1	14.020	drugX
199	40	F	0	1	11.349	drugX

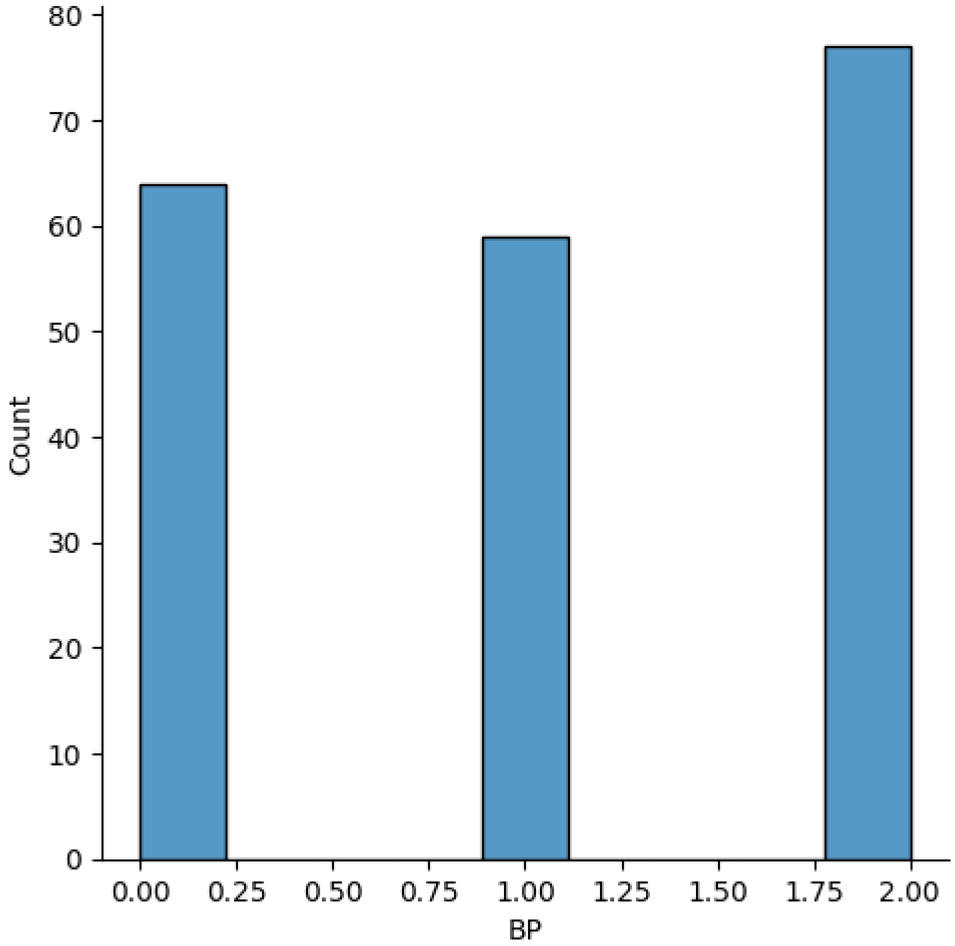
200 rows × 6 columns

In [121…

```
sns.displot(data2['BP'])
```

Out[121…

```
<seaborn.axisgrid.FacetGrid at 0x1916386eb90>
```



In [122…

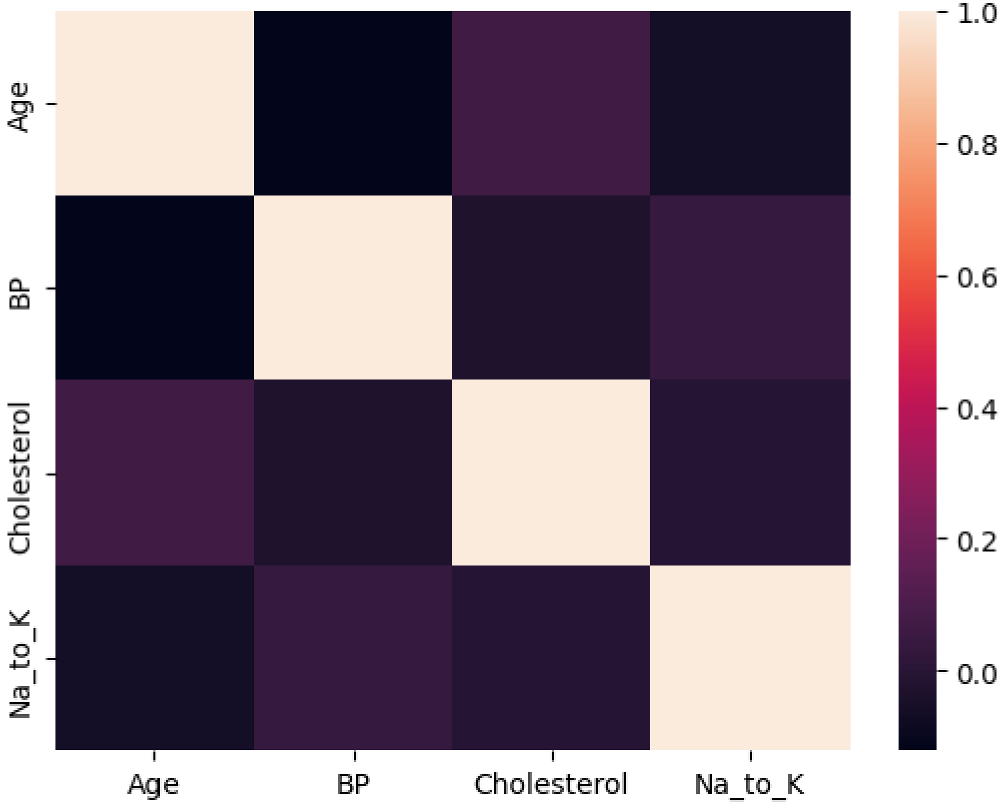
```
new_data2 = data2[['Age', 'BP', 'Cholesterol', 'Na_to_K']]
```

In [163…

```
sns.heatmap(new_data2.corr())
```

Out[163…

```
<Axes: >
```



Model Building for data2

In [123…
```
X = new_data2[['Age', 'Cholesterol', 'Na_to_K']]
y = data2['BP']
```

In [124…
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30)
```

In [125…
```
from sklearn.linear_model import LinearRegression

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

Out[125… ▾ LinearRegression

```
LinearRegression()
```

In [126…

```
#Prediction
predX = lr.predict(X_test)
print(predX)
```

```
[1.17948255 0.87149886 1.15403066 0.91369289 1.29715605 0.93036102
 1.07669148 1.02138302 0.98194584 1.24379266 1.13599336 0.9358394
 0.90889546 1.04976238 1.1943776  0.96194822 1.08720137 1.12351748
 0.93842582 0.81695131 0.88438205 1.17738574 0.81834615 0.84618828
 1.18118745 0.94032171 0.97170533 1.12901312 0.9279675  1.18862678
 0.82059716 0.94063383 0.90512497 0.96906673 1.05121723 0.98785583
 1.04615116 0.97260591 0.91316062 1.36035689 1.13687056 1.01932303
 0.78111628 1.13674687 0.87549402 1.05806675 1.25921844 0.89251658
 0.99617131 1.28819313 1.20077506 1.18823655 1.25721308 1.25337888
 0.96876595 1.03890291 1.21508083 1.32087403 0.79351868 1.27920144]
```

In [127…

```
#Accuracy
print(lr.score(X_test,y_test))
```

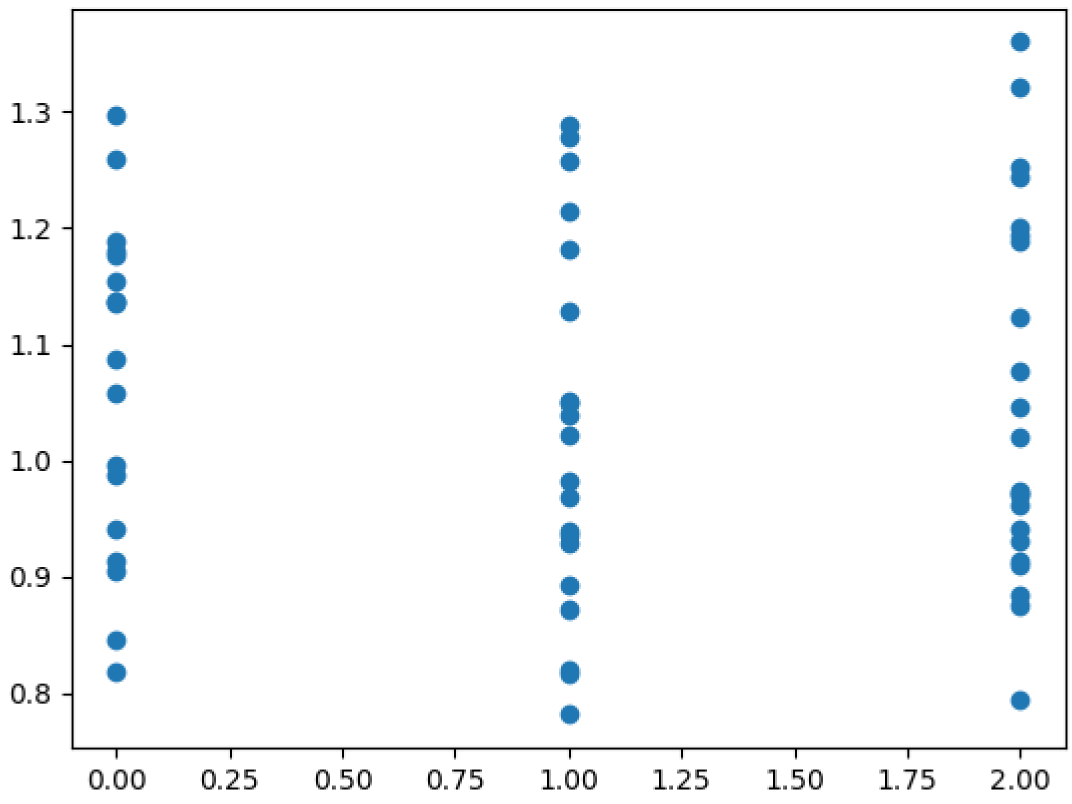
```
-0.047925459897639966
```

In [128…

```
plt.scatter(y_test,predX)
```

Out[128…

```
<matplotlib.collections.PathCollection at 0x19163909610>
```



Data 3

In [129…

```
data3 = pd.read_csv(r"C:\Users\DELL\Downloads\7_Uber.csv")
data3
```

Out[129…

	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

200000 rows × 9 columns

In [130…

```
data3.describe()
```

Out[130…

	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 [131…

```
data3.head()
```

Out[131…

	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	

In [132…

```
data3.tail()
```

Out[132…

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latituc
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

In [133…

```
data3.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 [134…

```
data3.columns
```

Out[134…

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

Random Forest

In [135…

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [136…

```
df1=pd.read_csv(r"C:\Users\DELL\Downloads\C1_Ionosphere.csv")
df1
```

Out[136…

	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.1	0.03760	...	-0.51171	0.41078	-0.46168	0.21266	-0.34
0	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.26569	-0.20468	-0.18401	-0.19040	-0.11
1	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.40220	0.58984	-0.22145	0.43100	-0.17
2	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...	0.90695	0.51613	1.00000	1.00000	-0.20
3	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...	-0.65158	0.13290	-0.53206	0.02431	-0.62
4	1	0	0.02337	-0.00592	-0.09924	-0.11949	-0.00763	-0.11824	0.14706	0.06637	...	-0.01535	-0.03240	0.09223	-0.07859	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
345	1	0	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	...	-0.04202	0.83479	0.00123	1.00000	0.12
346	1	0	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	...	0.01361	0.93522	0.04925	0.93159	0.08
347	1	0	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	...	0.03193	0.92489	0.02542	0.92120	0.02
348	1	0	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	...	-0.02099	0.89147	-0.07760	0.82983	-0.17
349	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	...	-0.15114	0.81147	-0.04822	0.78207	-0.00

350 rows × 35 columns

In [137…

```
df1.describe()
```

Out[137…

	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.1	0.03760	...	0.56811
count	350.000000	350.0	350.000000	350.000000	350.000000	350.000000	350.000000	350.000000	350.000000	350.000000	...	350.000000
mean	0.891429	0.0	0.640330	0.044667	0.600350	0.116154	0.549284	0.120779	0.510453	0.181756	...	0.395643
std	0.311546	0.0	0.498059	0.442032	0.520431	0.461443	0.493124	0.520816	0.507117	0.484482	...	0.579206
min	0.000000	0.0	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	...	-1.000000
25%	1.000000	0.0	0.471517	-0.065388	0.412555	-0.024868	0.209105	-0.053483	0.086785	-0.049003	...	0.000000
50%	1.000000	0.0	0.870795	0.016700	0.808620	0.021170	0.728000	0.015085	0.682430	0.017550	...	0.549175
75%	1.000000	0.0	1.000000	0.194727	1.000000	0.335317	0.970445	0.451572	0.950555	0.536192	...	0.907165
max	1.000000	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000

8 rows × 34 columns

In [138…

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 350 entries, 0 to 349
Data columns (total 35 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   1          350 non-null    int64  
 1   0          350 non-null    int64  
 2   0.99539    350 non-null    float64
 3   -0.05889   350 non-null    float64
 4   0.85243    350 non-null    float64
 5   0.02306    350 non-null    float64
 6   0.83398    350 non-null    float64
 7   -0.37708   350 non-null    float64
 8   1.1        350 non-null    float64
 9   0.03760    350 non-null    float64
 10  0.85243.1  350 non-null    float64
 11  -0.17755   350 non-null    float64
 12  0.59755    350 non-null    float64
 13  -0.44945   350 non-null    float64
 14  0.60536    350 non-null    float64
 15  -0.38223   350 non-null    float64
 16  0.84356    350 non-null    float64
 17  -0.38542   350 non-null    float64
 18  0.58212    350 non-null    float64
 19  -0.32192   350 non-null    float64
 20  0.56971    350 non-null    float64
 21  -0.29674   350 non-null    float64
 22  0.36946    350 non-null    float64
 23  -0.47357   350 non-null    float64
 24  0.56811    350 non-null    float64
 25  -0.51171   350 non-null    float64
 26  0.41078    350 non-null    float64
 27  -0.46168   350 non-null    float64
 28  0.21266    350 non-null    float64
 29  -0.34090   350 non-null    float64
 30  0.42267    350 non-null    float64
 31  -0.54487   350 non-null    float64
 32  0.18641    350 non-null    float64
 33  -0.45300   350 non-null    float64
 34  g          350 non-null    object 
```

```
dtypes: float64(32), int64(2), object(1)
memory usage: 95.8+ KB
```

In [139…

```python
g = {"g":{"g":1,"b":2}}
df1 = df1.replace(g)
df1
```

Out[139…

	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.1	0.03760	...	-0.51171	0.41078	-0.46168	0.21266	-0.34
0	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.26569	-0.20468	-0.18401	-0.19040	-0.11
1	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.40220	0.58984	-0.22145	0.43100	-0.17
2	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...	0.90695	0.51613	1.00000	1.00000	-0.20
3	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...	-0.65158	0.13290	-0.53206	0.02431	-0.62
4	1	0	0.02337	-0.00592	-0.09924	-0.11949	-0.00763	-0.11824	0.14706	0.06637	...	-0.01535	-0.03240	0.09223	-0.07859	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
345	1	0	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	...	-0.04202	0.83479	0.00123	1.00000	0.12
346	1	0	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	...	0.01361	0.93522	0.04925	0.93159	0.08
347	1	0	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	...	0.03193	0.92489	0.02542	0.92120	0.02
348	1	0	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	...	-0.02099	0.89147	-0.07760	0.82983	-0.17
349	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	...	-0.15114	0.81147	-0.04822	0.78207	-0.00

350 rows × 35 columns

In [140…

```python
df1["g"].value_counts()
```

Out[140…

```
g
1    224
2    126
Name: count, dtype: int64
```

In [141…

```
x = df1.drop("g",axis=1)
y = df1["g"]
```

In [142…

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.40)
```

In [143…

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[143…

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [144…

```
rf = RandomForestClassifier()
```

In [145…

```
params = {"max_depth":[1,2,3,4,5],
         "min_samples_leaf":[2,4,6,8,10],
         "n_estimators":[1,3,5,7]
        }
```

In [146…

```
from sklearn.model_selection import GridSearchCV
gs = GridSearchCV(estimator=rf,param_grid=params,cv=2,scoring='accuracy')
gs.fit(x_train,y_train)
```

Out[146…

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [147…

```
rf_best = gs.best_estimator_
rf_best
```

Out[147…

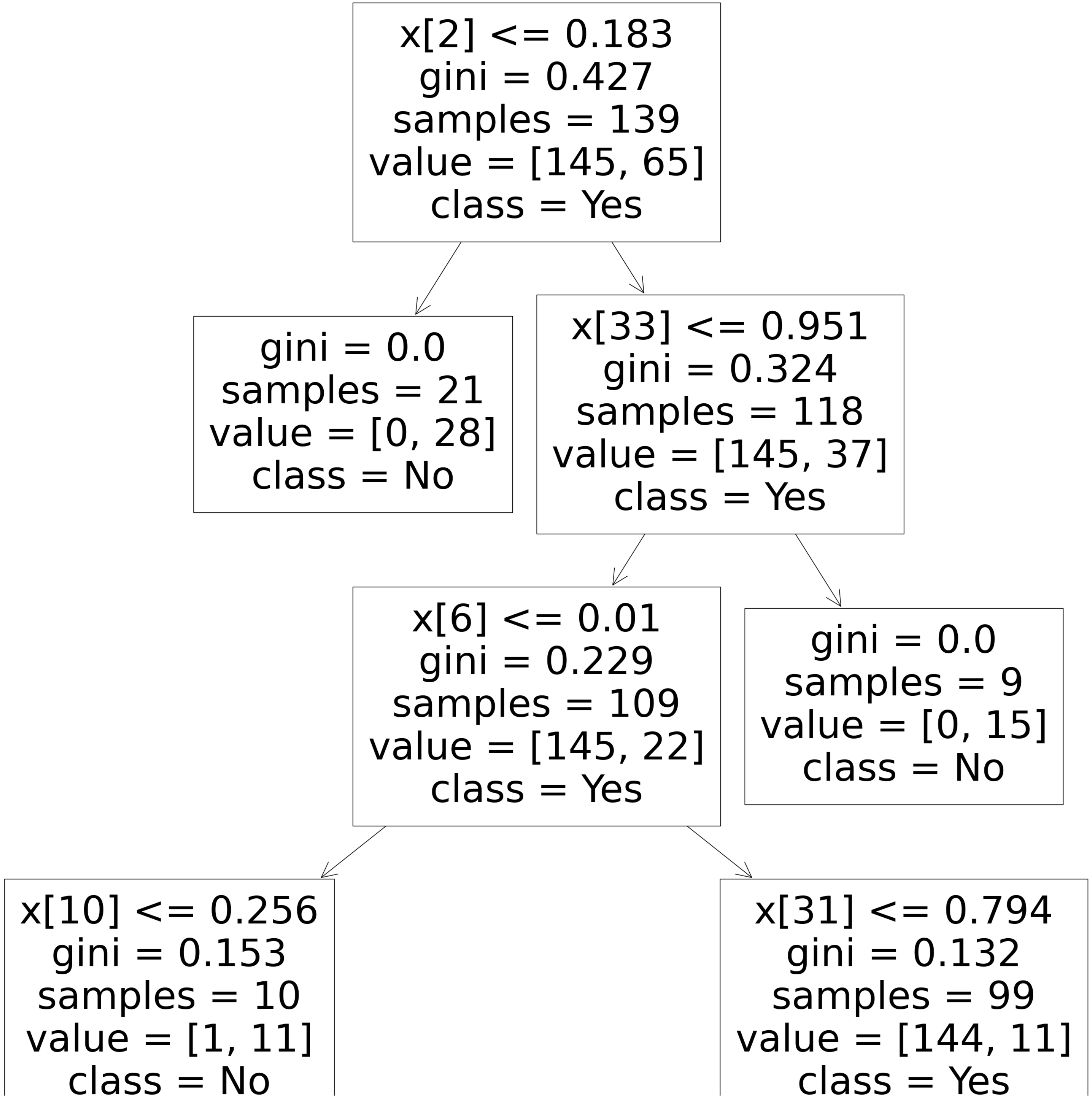
```
RandomForestClassifier

RandomForestClassifier(max_depth=4, min_samples_leaf=4, n_estimators=7)
```

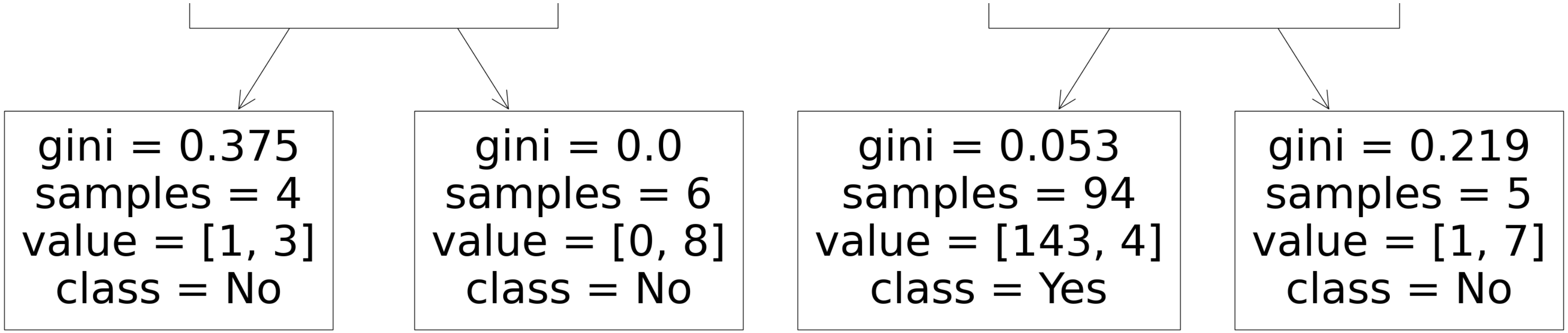
In [148…

```python
from sklearn.tree import plot_tree
plt.figure(figsize=(40,40))
plot_tree(rf_best.estimators_[4],feature_names=None,class_names=['Yes','No'])
```

Out[148…

```
[Text(0.5, 0.9, 'x[2] <= 0.183\ngini = 0.427\nsamples = 139\nvalue = [145, 65]\nclass = Yes'),
 Text(0.375, 0.7, 'gini = 0.0\nsamples = 21\nvalue = [0, 28]\nclass = No'),
 Text(0.625, 0.7, 'x[33] <= 0.951\ngini = 0.324\nsamples = 118\nvalue = [145, 37]\nclass = Yes'),
 Text(0.5, 0.5, 'x[6] <= 0.01\ngini = 0.229\nsamples = 109\nvalue = [145, 22]\nclass = Yes'),
 Text(0.25, 0.3, 'x[10] <= 0.256\ngini = 0.153\nsamples = 10\nvalue = [1, 11]\nclass = No'),
 Text(0.125, 0.1, 'gini = 0.375\nsamples = 4\nvalue = [1, 3]\nclass = No'),
 Text(0.375, 0.1, 'gini = 0.0\nsamples = 6\nvalue = [0, 8]\nclass = No'),
 Text(0.75, 0.3, 'x[31] <= 0.794\ngini = 0.132\nsamples = 99\nvalue = [144, 11]\nclass = Yes'),
 Text(0.625, 0.1, 'gini = 0.053\nsamples = 94\nvalue = [143, 4]\nclass = Yes'),
 Text(0.875, 0.1, 'gini = 0.219\nsamples = 5\nvalue = [1, 7]\nclass = No'),
 Text(0.75, 0.5, 'gini = 0.0\nsamples = 9\nvalue = [0, 15]\nclass = No')]
```


x[2] <= 0.183
gini = 0.427
samples = 139
value = [145, 65]
class = Yes
gini = 0.0
samples = 21
value = [0, 28]
class = No
x[33] <= 0.951
gini = 0.324
samples = 118
value = [145, 37]
class = Yes
x[6] <= 0.01
gini = 0.229
samples = 109
value = [145, 22]
class = Yes
gini = 0.0
samples = 9
value = [0, 15]
class = No
x[10] <= 0.256
gini = 0.153
samples = 10
value = [1, 11]
class = No
x[31] <= 0.794
gini = 0.132
samples = 99
value = [144, 11]
class = Yes



Random Forest for data2

In [195…

```
df2=pd.read_csv(r"C:\Users\DELL\Downloads\C10_Loan1.csv")
df2
```

Out[195…

	Home Owner	Marital Status	Annual Income	Defaulted Borrower
0	Yes	Single	125	No
1	No	Married	100	No
2	No	Single	70	No
3	Yes	Married	120	No
4	No	Divorced	95	Yes
5	No	Married	60	No
6	Yes	Divorced	220	No
7	No	Single	85	Yes
8	No	Married	75	No
9	No	Single	90	Yes

In [196…

```
df2.describe()
```

Out[196…

	Annual Income
count	10.000000
mean	104.000000
std	45.631373
min	60.000000
25%	77.500000
50%	92.500000
75%	115.000000
max	220.000000

In [197…

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Home Owner         10 non-null     object
 1   Marital Status     10 non-null     object
 2   Annual Income      10 non-null     int64 
 3   Defaulted Borrower 10 non-null     object
dtypes: int64(1), object(3)
memory usage: 452.0+ bytes
```

In [198…

```
Home_Owner = {"Home Owner":{"Yes":1,"No":2}}
df2 = df2.replace(Home_Owner)
Defaulted_Borrower = {"Defaulted Borrower":{"Yes":1,"No":2}}
df2 = df2.replace(Defaulted_Borrower)
Marital_Status = {"Marital Status":{"Divorced":0,"Single":1,"Married":2}}
df2 = df2.replace(Marital_Status)
df2
```

Out[198...

	Home Owner	Marital Status	Annual Income	Defaulted Borrower
0	1	1	125	2
1	2	2	100	2
2	2	1	70	2
3	1	2	120	2
4	2	0	95	1
5	2	2	60	2
6	1	0	220	2
7	2	1	85	1
8	2	2	75	2
9	2	1	90	1

In [199...

```
df2["Home Owner"].value_counts()
```

Out[199...

```
Home Owner
2    7
1    3
Name: count, dtype: int64
```

In [200...

```
df2["Defaulted Borrower"].value_counts()
```

Out[200...

```
Defaulted Borrower
2    7
1    3
Name: count, dtype: int64
```

In [201...

```
x = df2.drop("Marital Status",axis=1)
y = df2["Marital Status"]
```

In [202…

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.40)
```

In [203…

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[203…

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [204…

```
rf = RandomForestClassifier()
```

In [205…

```
params = {"max_depth":[1,2,3,4,5],
         "min_samples_leaf":[2,4,6,8,10],
         "n_estimators":[1,3,5,7]
        }
```

In [206…

```
from sklearn.model_selection import GridSearchCV
gs = GridSearchCV(estimator=rf,param_grid=params,cv=2,scoring='accuracy')
gs.fit(x_train,y_train)
```

Out[206…

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [207…

```
rf_best = gs.best_estimator_
rf_best
```

Out[207…

```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=1, min_samples_leaf=2, n_estimators=5)
```

In [208…

```
from sklearn.tree import plot_tree
plt.figure(figsize=(40,40))
plot_tree(rf_best.estimators_[4],feature_names=None,class_names=['Yes','No'])
```

Out[208…

```
[Text(0.5, 0.5, 'gini = 0.278\nsamples = 3\nvalue = [0, 5, 1]\nclass = No')]
```

gini = 0.278
samples = 3
value = [0, 5, 1]
class = No

In [ ]:

```

```