

Problem statement

Data collection

Importing libraries

In [1]:

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

Importing dataset

In [2]:

```
data=pd.read_csv(r"C:\Users\user\Downloads\VE.CSV.csv")
data
```

Out[2]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66557
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62877
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64938
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66973
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63297
...
153	Rwanda	Sub-Saharan Africa	154	3.465	0.03464	0.22208	0.77370	0.42864	0.59201
154	Benin	Sub-Saharan Africa	155	3.340	0.03656	0.28665	0.35386	0.31910	0.48450
155	Syria	Middle East and Northern Africa	156	3.006	0.05015	0.66320	0.47489	0.72193	0.15684
156	Burundi	Sub-Saharan	157	2.905	0.08658	0.01530	0.41587	0.22396	0.11850

Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	
Africa									
157	Togo	Sub-Saharan Africa	158	2.839	0.06727	0.20868	0.13995	0.28443	0.36453

158 rows × 12 columns

head

In [3]:

```
# to display first 8 dataset values
da=data.head(8)
da
```

Out[3]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	(GDP per Capita)
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66557	
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62877	
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64938	
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66973	
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63297	
5	Finland	Western Europe	6	7.406	0.03140	1.29025	1.31826	0.88911	0.64169	
6	Netherlands	Western Europe	7	7.378	0.02799	1.32944	1.28017	0.89284	0.61576	
7	Sweden	Western Europe	8	7.364	0.03157	1.33171	1.28907	0.91087	0.65980	

info

In [4]:

```
# to identify missing values
data.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
```

describe

In [5]:

```
# to display summary of the dataset
data.describe()
```

Out[5]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)
count	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	0.14342
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693	0.12003
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000	0.000000
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330	0.06167
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515	0.10722
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092	0.18025
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730	0.55191

columns

In [6]:

```
# to display headings of the dataset
data.columns
```

Out[6]:

```
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 [7]:

```
a=data.dropna(axis=1)
a
```

Out[7]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.66557
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.62877
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.64938
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.66973
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.63297
...
153	Rwanda	Sub-Saharan Africa	154	3.465	0.03464	0.22208	0.77370	0.42864	0.59201
154	Benin	Sub-Saharan Africa	155	3.340	0.03656	0.28665	0.35386	0.31910	0.48450
155	Syria	Middle East and Northern Africa	156	3.006	0.05015	0.66320	0.47489	0.72193	0.15684
156	Burundi	Sub-Saharan Africa	157	2.905	0.08658	0.01530	0.41587	0.22396	0.11850
157	Togo	Sub-Saharan Africa	158	2.839	0.06727	0.20868	0.13995	0.28443	0.36453

158 rows × 12 columns



In [8]:

a.columns

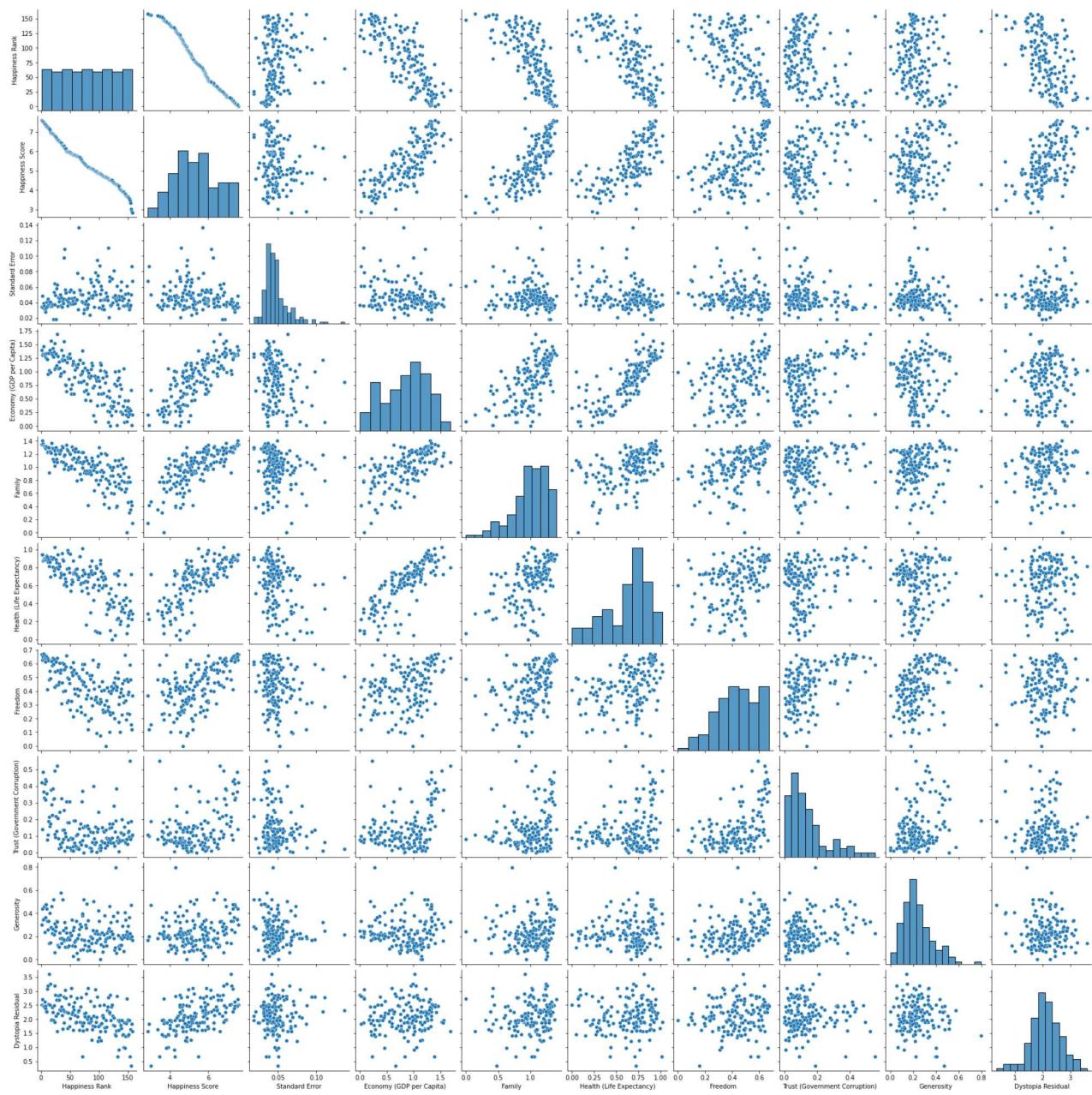
```
Out[8]: Index(['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual'],
      dtype='object')
```

EDA and Visualization

In [9]:

sns.pairplot(a)

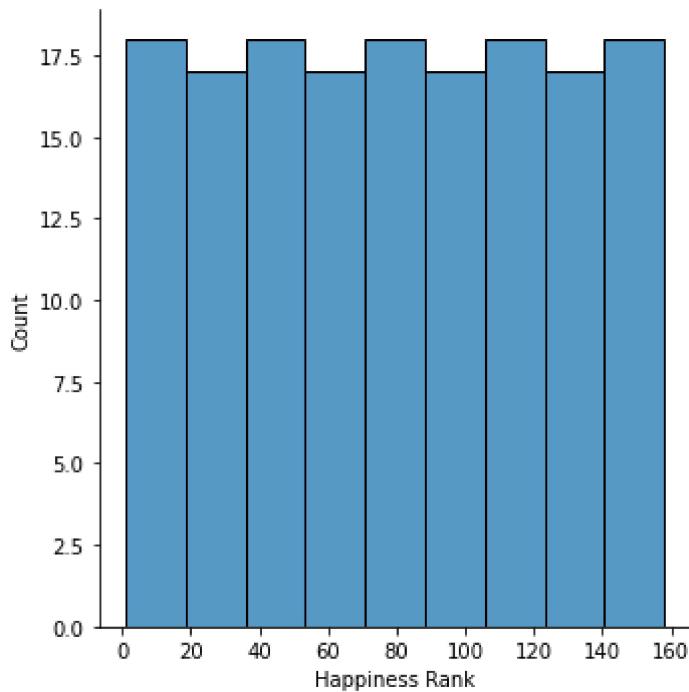
```
Out[9]: <seaborn.axisgrid.PairGrid at 0x1fc1f8e2d30>
```



distribution plot

```
In [10]: sns.displot(a["Happiness Rank"])
```

```
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1fc24ca9940>
```



correlation

```
In [11]: dat=data[['Country', 'Region', 'Happiness Rank', 'Happiness Score',
   'Standard Error', 'Economy (GDP per Capita)', 'Family',
   'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
   'Generosity', 'Dystopia Residual']]
sns.heatmap(dat.corr())
```

Out[11]: <AxesSubplot:>



To train the model-Model Building

```
In [12]:  
x=a[['Happiness Rank']]  
y=a['Happiness Rank']
```

```
In [13]:  
# to split my dataset into training and test data  
from sklearn.model_selection import train_test_split  
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

```
In [14]:  
from sklearn.linear_model import LinearRegression  
lr=LinearRegression()  
lr.fit(x_train,y_train)
```

```
Out[14]: LinearRegression()
```

```
In [15]:  
print(lr.intercept_)
```

```
-1.4210854715202004e-14
```

```
In [16]:  
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])  
coeff
```

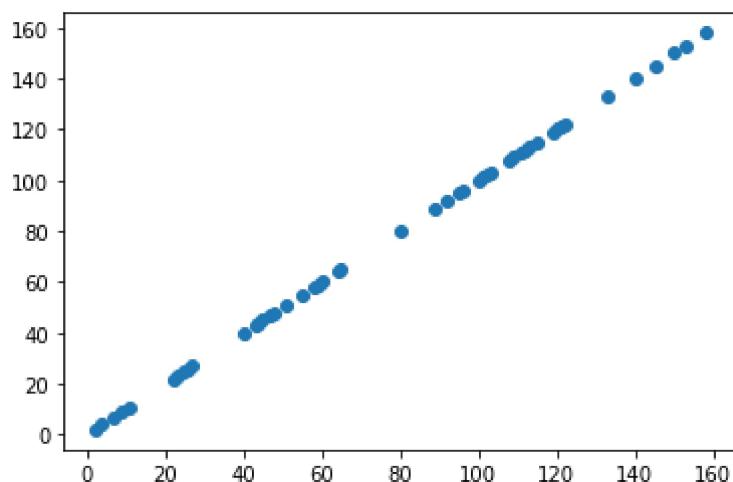
```
Out[16]:  


|                | Co-efficient |
|----------------|--------------|
| Happiness Rank | 1.0          |


```

```
In [17]:  
prediction=lr.predict(x_test)  
plt.scatter(y_test,prediction)
```

```
Out[17]: <matplotlib.collections.PathCollection at 0x1fc262c8250>
```



```
In [18]:  
print(lr.score(x_test,y_test))
```

1.0

```
In [19]: lr.score(x_train,y_train)
```

```
Out[19]: 1.0
```

Ridge regression

```
In [20]: from sklearn.linear_model import Ridge,Lasso
```

```
In [21]: rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
rr.score(x_test,y_test)
```

```
Out[21]: 0.9999999981583578
```

```
In [22]: rr.score(x_train,y_train)
```

```
Out[22]: 0.9999999981656488
```

Lasso regression

```
In [23]: la=Lasso(alpha=10)
la.fit(x_train,y_train)
la.score(x_train,y_train)
```

```
Out[23]: 0.9999778024483262
```

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

```
Out[24]: 0.9999777142203998
```

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

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

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

```
[0.99952897]
```

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

```
0.03784526534779786
```

```
In [31]: predict=en.predict(x_test)
```

```
In [28]: print(en.score(x_test,y_test))
```

```
0.9999997772471647
```

```
In [29]: from sklearn import metrics
```

```
In [32]: print("Mean Absolute error:",metrics.mean_absolute_error(y_test,predict))
```

```
Mean Absolute error: 0.018566504073749796
```

```
In [33]: print("Mean Squared error:",metrics.mean_squared_error(y_test,predict))
```

```
Mean Squared error: 0.0004405675959306349
```

```
In [34]: print("Root squared error:",np.sqrt(metrics.mean_squared_error(y_test,predict)))
```

```
Root squared error: 0.020989702140112303
```