Type *Markdown* and LaTeX: α^2

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

In [2]:
```
#import dataset
df=pd.read_csv(r"E:\154\2015 - 2015.csv",low_memory=False).dropna(axis='column
df
```

Out[2]:

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

158 rows × 12 columns

In [3]:
```
df.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 [4]:
```
#to display top 5 rows
df.head()
```

Out[4]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freed
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	0.665
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	0.628
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	0.649
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	0.669
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	0.632

Data cleaning and Pre-Processing

In [5]:
```
#To find null values
df.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 [6]:
```
# To display summary of statistics
df.describe()
```

Out[6]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	(Gc C
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	1
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615	
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693	
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000	
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330	
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515	
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092	
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730	

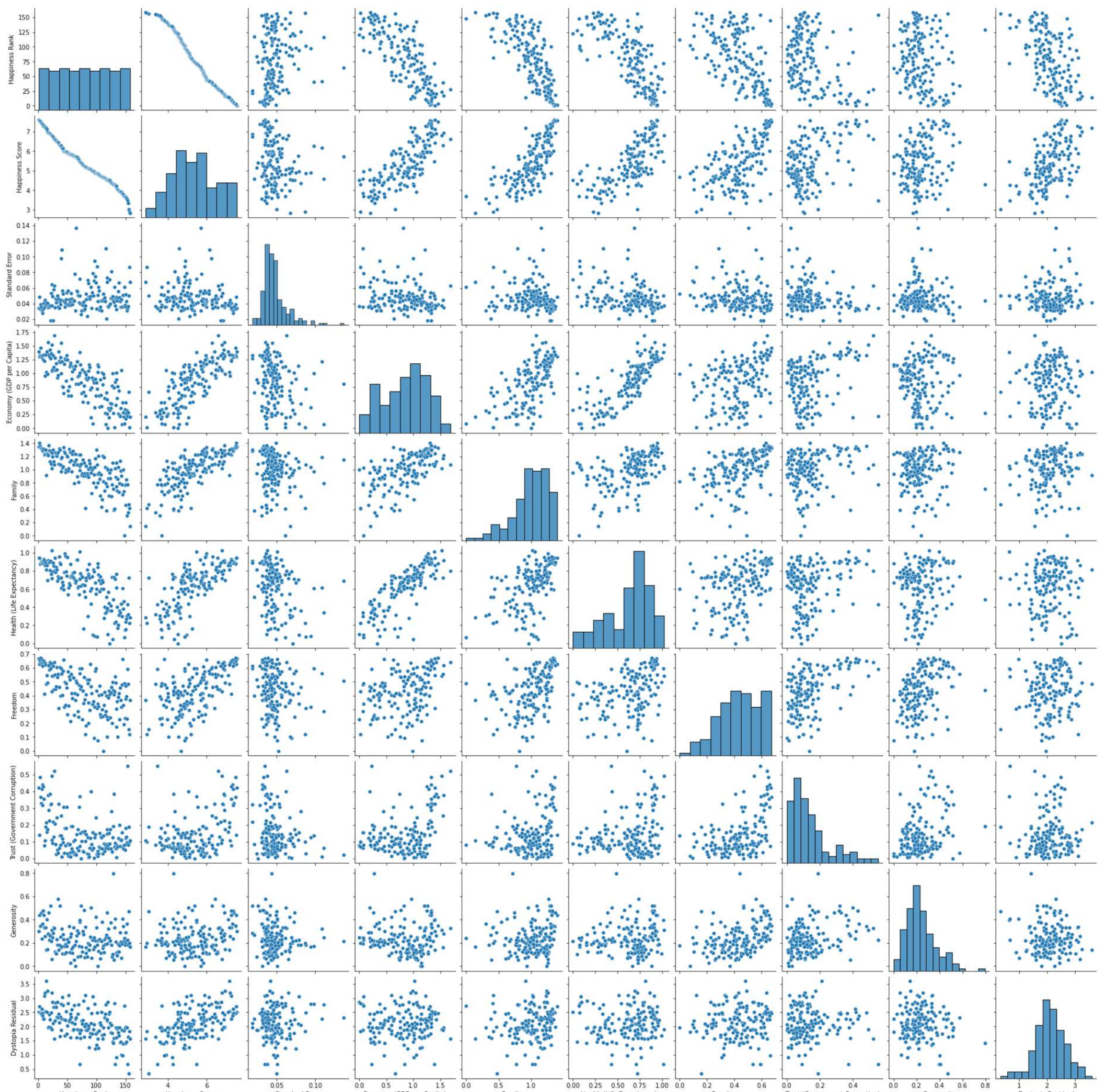
In [7]:
```
#To Display column heading
df.columns
```

Out[7]:
```
Index(['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruptio
n)',
       'Generosity', 'Dystopia Residual'],
      dtype='object')
```

EDA and VISUALIZATION

In [8]:
```
sns.pairplot(df)
```

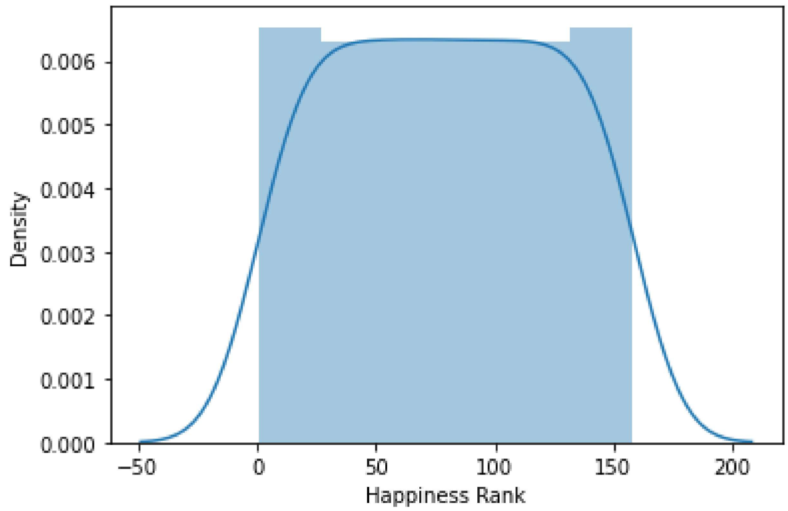
Out[8]: <seaborn.axisgrid.PairGrid at 0x16532709ac0>



In [9]:
```
sns.distplot(df['Happiness Rank'])
```

```
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 function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[9]: <AxesSubplot:xlabel='Happiness Rank', ylabel='Density'>

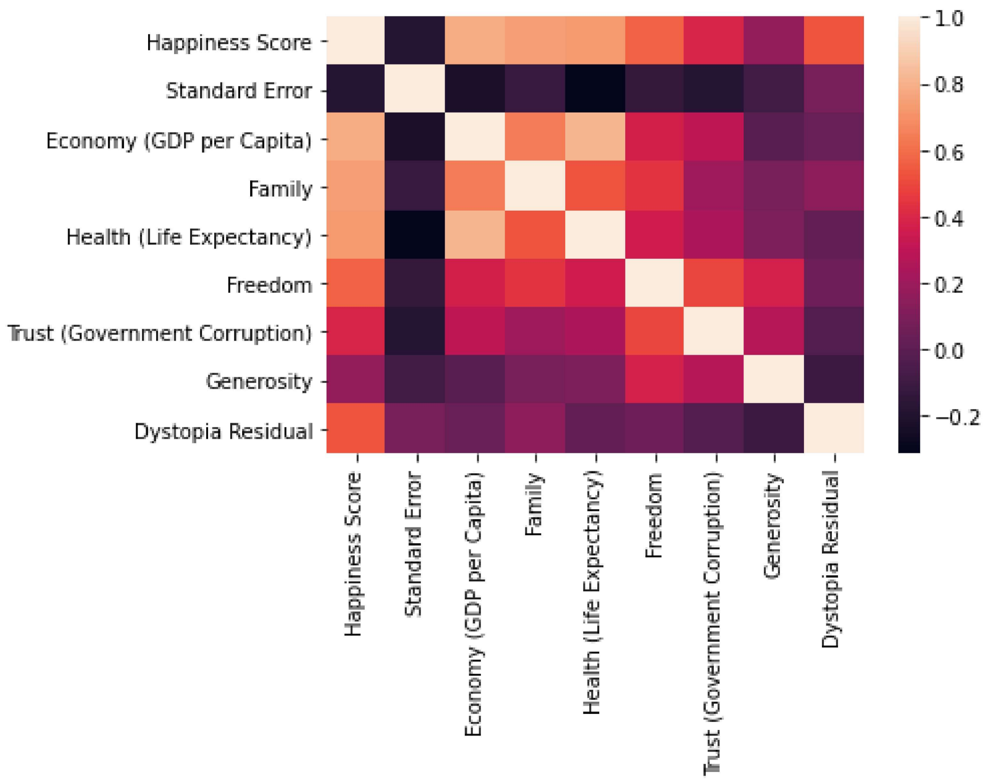


In [10]:
```
df1=df[['Country', 'Region', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual']]
```

Plot Using Heat Map

In [11]:
```
sns.heatmap(df1.corr())
```

Out[11]: <AxesSubplot:>



To Train The Model-Model Building

we are going to train Linera Regression Model;We need to split out data into two variables x and y where x is independent variable(input) and y is dependent on x(output) we could ignore address column as it required for our model

In [12]:
```
x=df1[['Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual']]
y=df1['Family']
```

To Split my dataset into training and test data

In [13]:
```

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]:
```
lr.intercept_
```

Out[15]: -0.0001609872532650769

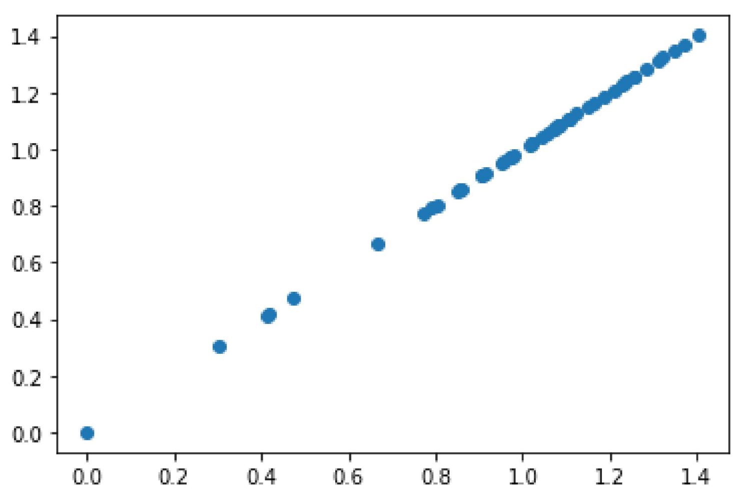
In [16]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[16]:

	Co-efficient
Happiness Score	0.999970
Standard Error	-0.000543
Economy (GDP per Capita)	-1.000072
Health (Life Expectancy)	-0.999828
Freedom	-0.999520
Trust (Government Corruption)	-0.999829
Generosity	-1.000024
Dystopia Residual	-0.999954

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

Out[17]: <matplotlib.collections.PathCollection at 0x165390f2be0>



Accuracy

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

Out[18]: 0.9999988920925807

In [19]:
```
lr.score(x_train,y_train)
```
Out[19]:
```
0.9999989896054637
```

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

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

In [22]:
```
rr.score(x_test,y_test)
```
Out[22]:
```
0.7009415594590706
```

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

In [24]:
```
la.score(x_test,y_test)
```
Out[24]:
```
-1.3557709450662259e-05
```

ElasticNet

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

In [27]:
```
print(en.intercept_)
```
```
0.9907238181818183
```

In [28]:
```
print(en.predict(x_test))
```
```
[0.99072382 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382
 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382
 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382
 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382
 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382
 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382
 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382
 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382 0.99072382]
```

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

```
-1.3557709450662259e-05
```

Evaluation Metrics ¶

In [31]:
```
from sklearn import metrics
```

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

```
Mean Absolute Error 0.00026943036268065024
```

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

```
Mean Squared Error: 9.187860411525314e-08
```

In [34]:
```
print("Root Mean Absolute Error:",np.sqrt(metrics.mean_squared_error(y_test,pr
```

```
Root Mean Absolute Error: 0.00030311483651456783
```

In [ ]:
```

```