Problem statement

predicting the house price in USA.To create a model to help him estimate of what the house would sell for.

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

In [2]:
```
df=pd.read_csv("2015")
```

To display top 10 rows

In [3]:

```
df.head(10)
```

Out[3]:

	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
5	Finland	Western Europe	6	7.406	0.03140	1.29025	1.31826	0.88911	0.64169	0.41372	0.23351	2.61955
6	Netherlands	Western Europe	7	7.378	0.02799	1.32944	1.28017	0.89284	0.61576	0.31814	0.47610	2.46570
7	Sweden	Western Europe	8	7.364	0.03157	1.33171	1.28907	0.91087	0.65980	0.43844	0.36262	2.37119
8	New Zealand	Australia and New Zealand	9	7.286	0.03371	1.25018	1.31967	0.90837	0.63938	0.42922	0.47501	2.26425
9	Australia	Australia and New Zealand	10	7.284	0.04083	1.33358	1.30923	0.93156	0.65124	0.35637	0.43562	2.26646

Data Cleaning And Pre-Processing

In [4]:

```
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 [5]:

```
# Display the statistical summary
df.describe()
```

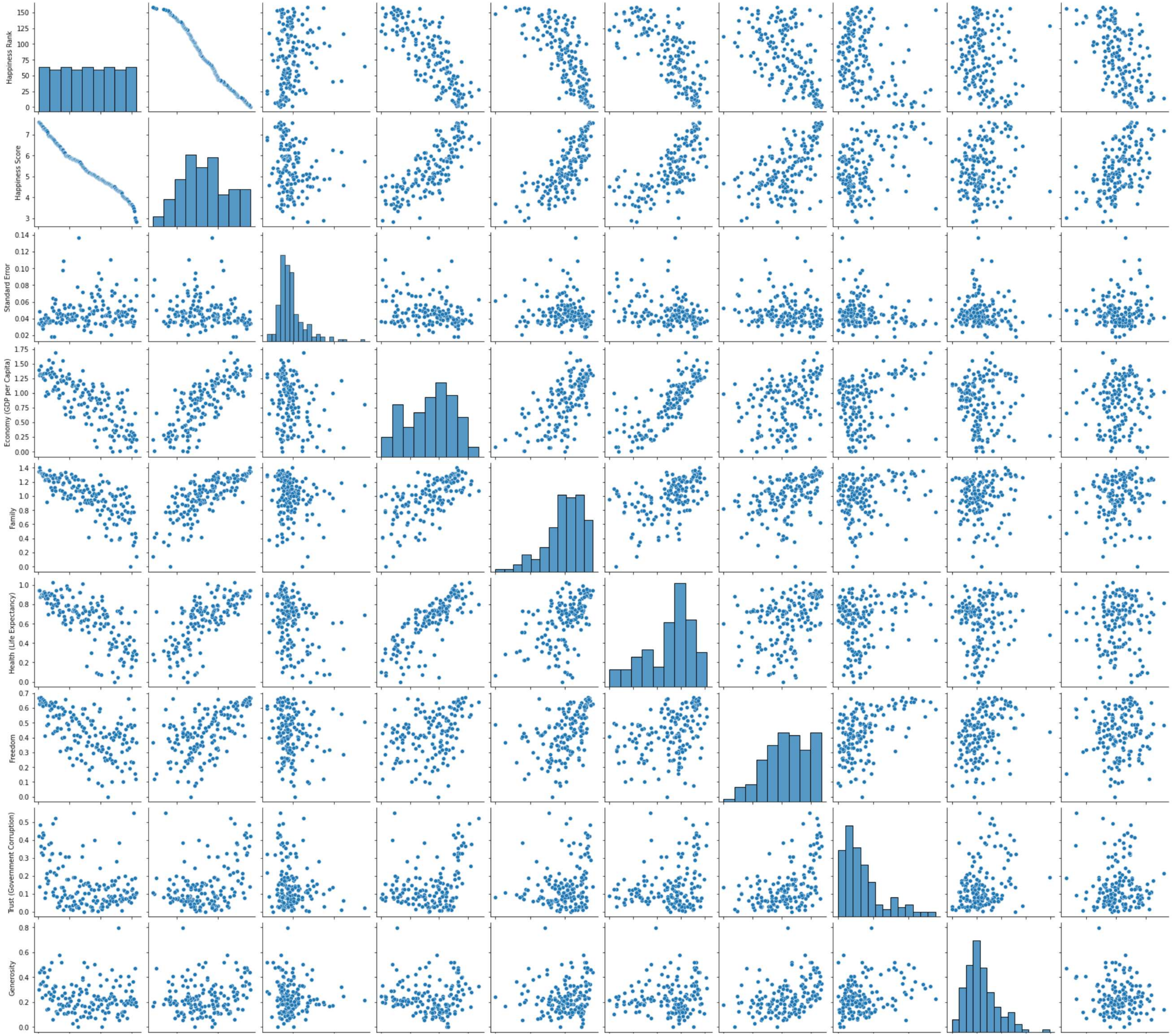
Out[5]:

	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

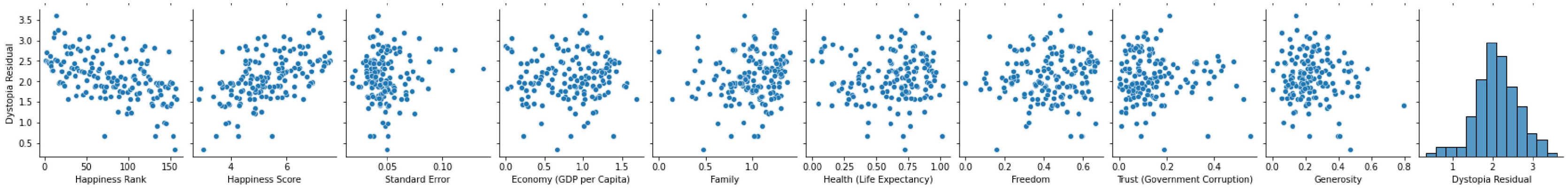
In [6]:
```
# To display the col headings
df.columns
```
Out[6]:
```
Index(['Country', 'Region', 'Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family',
       'Health (Life Expectancy)', 'Freedom', 'Trust (Government Corruption)',
       'Generosity', 'Dystopia Residual'],
      dtype='object')
```
In [7]:
```
cols=df.dropna(axis=1)
```
In [8]:
```
cols.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(cols)
```
Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x231ad840cd0>
```

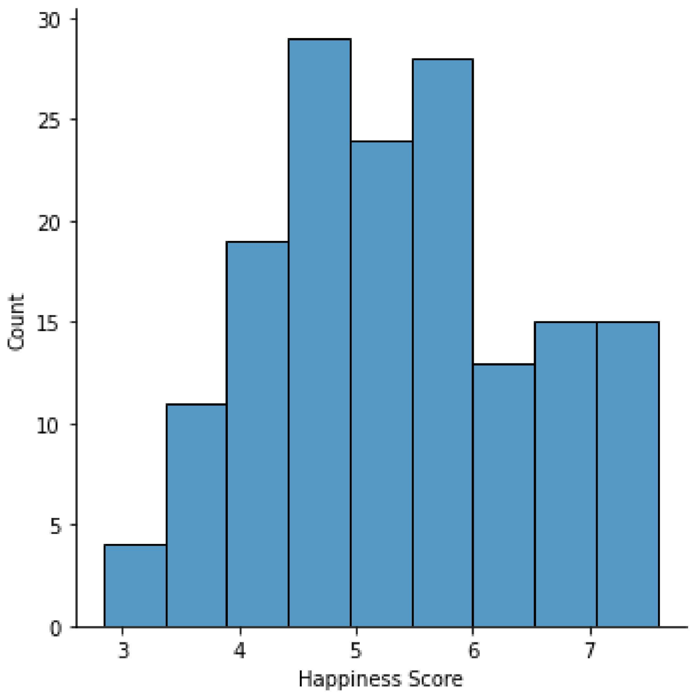


Happiness Rank
Happiness Score
Standard Error
Economy (GDP per Capita)
Family
Health (Life Expectancy)
Freedom
Trust (Government Corruption)
Generosity



In [10]:
```
sns.displot(df['Happiness Score'])
```

Out[10]: <seaborn.axisgrid.FacetGrid at 0x231b297b310>



In [11]:

```
# We use displot in older version we get distplot use displot
sns.distplot(df['Happiness Score'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a dep
recated 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[11]:

```
<AxesSubplot:xlabel='Happiness Score', ylabel='Density'>
```

In [12]:

```python
df1=cols[['Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family']]
df1
```

Out[12]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family
0	1	7.587	0.03411	1.39651	1.34951
1	2	7.561	0.04884	1.30232	1.40223
2	3	7.527	0.03328	1.32548	1.36058
3	4	7.522	0.03880	1.45900	1.33095
4	5	7.427	0.03553	1.32629	1.32261
...	...	...	...	...	...
153	154	3.465	0.03464	0.22208	0.77370
154	155	3.340	0.03656	0.28665	0.35386
155	156	3.006	0.05015	0.66320	0.47489
156	157	2.905	0.08658	0.01530	0.41587
157	158	2.839	0.06727	0.20868	0.13995

158 rows × 5 columns

In [13]:
```python
sns.heatmap(df1.corr())
```

Out[13]: <AxesSubplot:>

To train the model - MODEL BUILD

Going to train linear regression model;We split our data into 2 variables x and y where x is independent var(input) and y is dependent on x(output), we could ignore address col as it is not required for our model

In [14]:
```python
x=df1[['Happiness Rank', 'Happiness Score',
       'Standard Error', 'Economy (GDP per Capita)', 'Family']]
y=df1[['Happiness Rank']]
```

To split the dataset into test data

In [15]:

```
# importing lib for splitting test data
from sklearn.model_selection import train_test_split
```

In [16]:

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

In [17]:

```
from sklearn.linear_model import LinearRegression

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

Out[17]:

```
LinearRegression()
```

In [18]:

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

```
[1.42108547e-14]
```

In [19]:

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

```
1.0
```

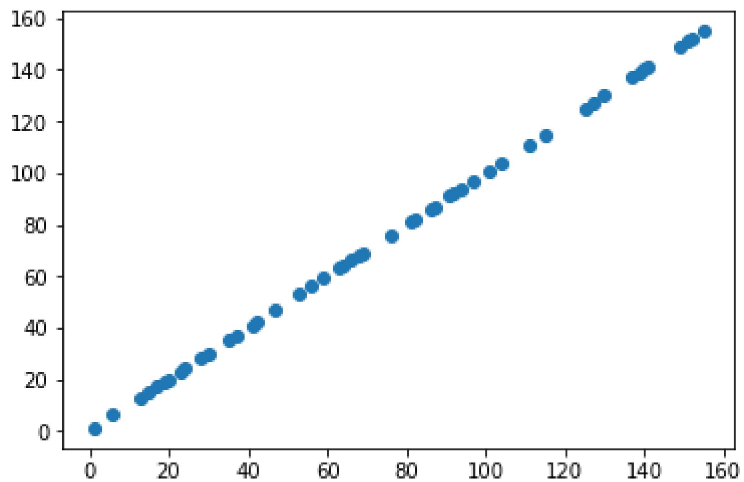
In [20]:

```
coeff=pd.DataFrame(lr.coef_)
coeff
```

Out[20]:

	0	1	2	3	4
0	1.0	-1.576490e-15	-1.097963e-13	-5.413673e-16	8.067434e-15

In [21]:
```
pred = lr.predict(x_test)
plt.scatter(y_test,pred)
```
Out[21]: <matplotlib.collections.PathCollection at 0x231b42bb9a0>



In [22]:
```
from sklearn.linear_model import Ridge,Lasso
```

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

In [24]:
```
rr.score(x_test,y_test)
```
Out[24]: 0.9999999946793897

In [25]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```
Out[25]: Lasso(alpha=10)

In [26]:
```
la.score(x_test,y_test)
```
Out[26]: 0.9999767157462808

ELASTIC NET

In [28]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[28]: ElasticNet()

In [29]:
```
print(en.coef_)
```

```
[ 0.99952014 -0.          0.         -0.         -0.        ]
```

In [30]:
```
print(en.intercept_)
```

```
[0.03882978]
```

In [41]:
```
prediction=en.predict(x_test)
prediction
```

Out[41]:
```
array([ 69.00571912,  81.99948087,  96.9922829 ,  41.01915533,
        24.02731303, 129.97644737,  63.0085983 ,  13.03259154,
       148.96732994, 136.97308831,   1.03834992, 126.97788696,
        68.00619898,  47.01627614,  35.02203452,  28.02539357,
        37.02107479,  15.03163181,   6.03595059, 103.98892385,
       110.9855648 ,  80.99996074,  85.99756141,  66.00715871,
       124.97884669,  42.01867546,  17.03067208,  64.00811844,
       139.97164872, 150.96637021, 154.96445075, 140.97116885,
        53.01339695,  76.00236006,  93.9937225 ,  86.99708155,
       114.98364534,  20.02923249,  30.02443384, 151.96589034,
        19.02971235, 138.97212858,  23.02779289, 100.99036344,
        59.01051776,  56.01195736,  91.99468223,  90.99516209])
```

In [42]:
```
print(en.score(x_test,y_test))
```

```
0.9999997672691823
```

EVALUATION METRICS

In [34]:
```python
from sklearn import metrics
```

In [43]:
```python
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 0.018763074744266837
```

In [44]:
```python
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 0.00047885567774766103
```

In [46]:
```python
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 0.021882771253834855
```

In [ ]:
```python

```