A real estate agent want help to predict the house price for regions in Usa.he gave us the dataset to work on to use linear Regression model.Create a model that helps him to estimate

Data Collection

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

In [155]:
```
#import the dataset
data=pd.read_csv(r"C:\Users\user\Desktop\Vicky\18_world-data-2023.csv")[0:50]
```

In [156]:
```
#to display top 10 rows
data.head()
```

Out[156]:

	Country	Density\n(P/Km2)	Abbreviation	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Calling Code	Capital/Major City	Co2 Emissions
0	Afghanistan	60	AF	58.10%	652,230	323,000	32.49	93.0	Kabul	8,67
1	Albania	105	AL	43.10%	28,748	9,000	11.78	355.0	Tirana	4,53
2	Algeria	18	DZ	17.40%	2,381,741	317,000	24.28	213.0	Algiers	150,00
3	Andorra	164	AD	40.00%	468	NaN	7.20	376.0	Andorra la Vella	46
4	Angola	26	AO	47.50%	1,246,700	117,000	40.73	244.0	Luanda	34,69

5 rows × 35 columns

In [157]:
```
#to display null values
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 35 columns):
 #   Column                                   Non-Null Count  Dtype  
---  ------                                   --------------  -----  
 0   Country                                  50 non-null     object 
 1   Density
(P/Km2)                          50 non-null     object 
 2   Abbreviation                             49 non-null     object 
 3   Agricultural Land( %)                    50 non-null     object 
 4   Land Area(Km2)                           50 non-null     object 
 5   Armed Forces size                        47 non-null     object 
 6   Birth Rate                               50 non-null     float64
 7   Calling Code                             50 non-null     float64
 8   Capital/Major City                       50 non-null     object 
 9   Co2-Emissions                            50 non-null     object 
 10  CPI                                      47 non-null     object 
 11  CPI Change (%)                           48 non-null     object 
 12  Currency-Code                            46 non-null     object 
 13  Fertility Rate                           50 non-null     float64
 14  Forested Area (%)                        50 non-null     object 
 15  Gasoline Price                           48 non-null     object 
 16  GDP                                      50 non-null     object 
 17  Gross primary education enrollment (%)   49 non-null     object 
 18  Gross tertiary education enrollment (%)  48 non-null     object 
 19  Infant mortality                         50 non-null     float64
 20  Largest city                             49 non-null     object 
 21  Life expectancy                          49 non-null     float64
 22  Maternal mortality ratio                 48 non-null     float64
 23  Minimum wage                             42 non-null     object 
 24  Official language                        50 non-null     object 
 25  Out of pocket health expenditure         49 non-null     object 
 26  Physicians per thousand                  50 non-null     float64
 27  Population                               50 non-null     object 
 28  Population: Labor force participation (%)  47 non-null     object 
 29  Tax revenue (%)                          44 non-null     object 
 30  Total tax rate                           48 non-null     object 
 31  Unemployment rate                        47 non-null     object 
 32  Urban_population                         50 non-null     object 
 33  Latitude                                 50 non-null     float64
 34  Longitude                                50 non-null     float64
dtypes: float64(9), object(26)
memory usage: 13.8+ KB
```

In [158]:
```
data.shape
```

Out[158]: (50, 35)

In [159]:
```
#to display summary of statistics
data.describe()
```

Out[159]:

	Birth Rate	Calling Code	Fertility Rate	Infant mortality	Life expectancy	Maternal mortality ratio	Physicians per thousand	Latitude	Longitude
count	50.00000	50.000000	50.00000	50.000000	49.000000	48.000000	50.000000	50.000000	50.000000
mean	19.64860	291.820000	2.62600	22.618000	72.312245	174.041667	1.929800	17.670807	4.900287
std	10.67511	272.353663	1.41232	22.042368	7.988498	248.707549	1.782451	24.015082	58.380156
min	7.20000	1.000000	1.27000	1.900000	52.800000	2.000000	0.040000	-38.416097	-106.346771
25%	10.75000	56.250000	1.66000	5.225000	66.600000	13.750000	0.272500	5.080929	-57.638718
50%	14.89000	240.000000	1.94000	11.750000	74.900000	50.000000	1.665000	16.799808	15.336481
75%	24.68500	375.750000	2.98250	35.375000	78.100000	242.750000	2.972500	39.017239	32.552116
max	42.17000	994.000000	5.92000	84.500000	82.700000	1140.000000	8.420000	56.263920	133.775136

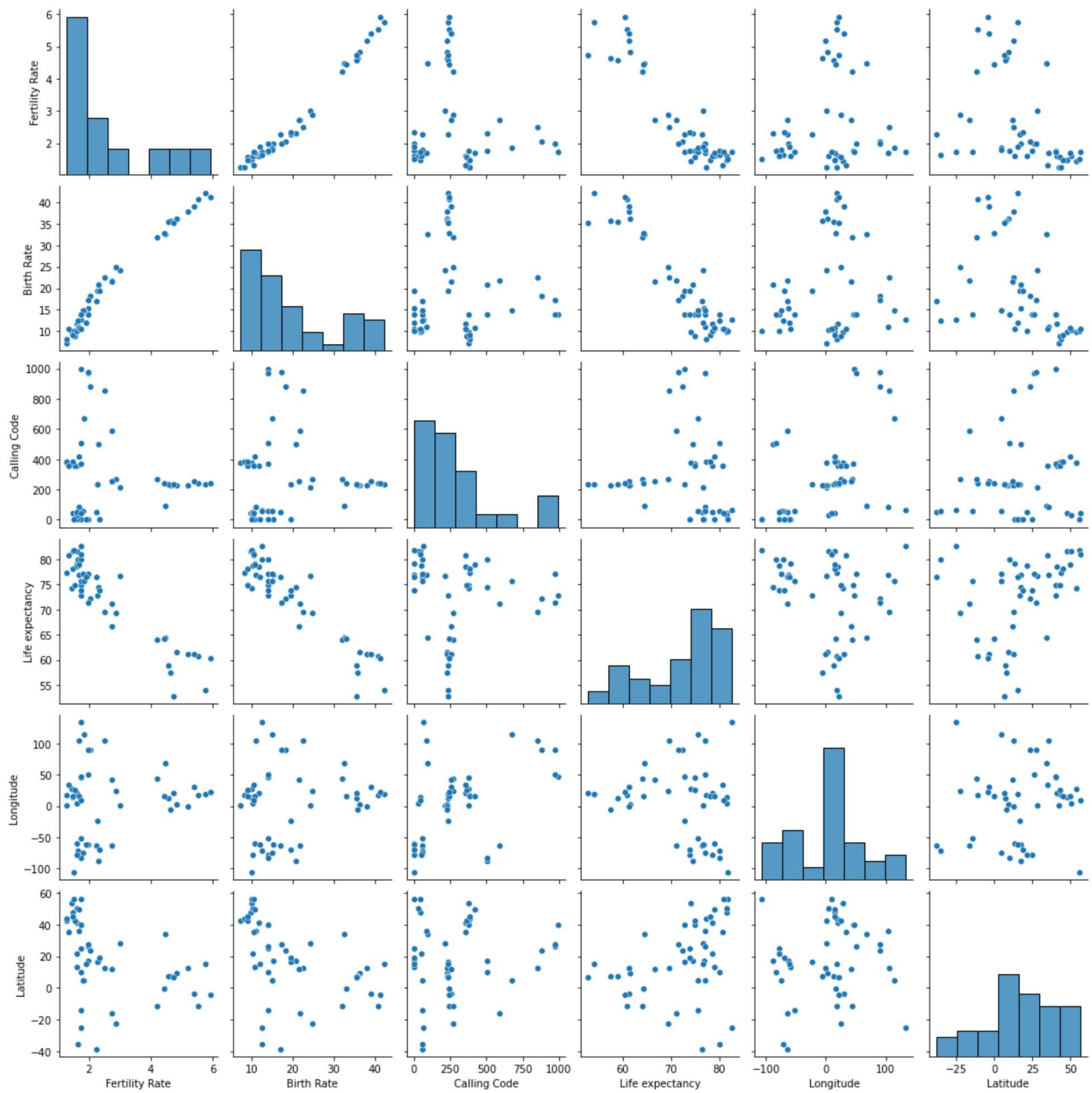
In [160]:
```
#to display columns name
data.columns
```

Out[160]:
```
Index(['Country', 'Density\n(P/Km2)', 'Abbreviation', 'Agricultural Land( %)',
       'Land Area(Km2)', 'Armed Forces size', 'Birth Rate', 'Calling Code',
       '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 expenditure',
       'Physicians per thousand', 'Population',
       'Population: Labor force participation (%)', 'Tax revenue (%)',
       'Total tax rate', 'Unemployment rate', 'Urban_population', 'Latitude',
       'Longitude'],
      dtype='object')
```

In [161]:
```
data1=data[['Fertility Rate','Birth Rate','Calling Code','Life expectancy','Longitude','Latitude
```

In [162]:
```
sns.pairplot(data1)
```

Out[162]: <seaborn.axisgrid.PairGrid at 0x2389a4c3eb0>

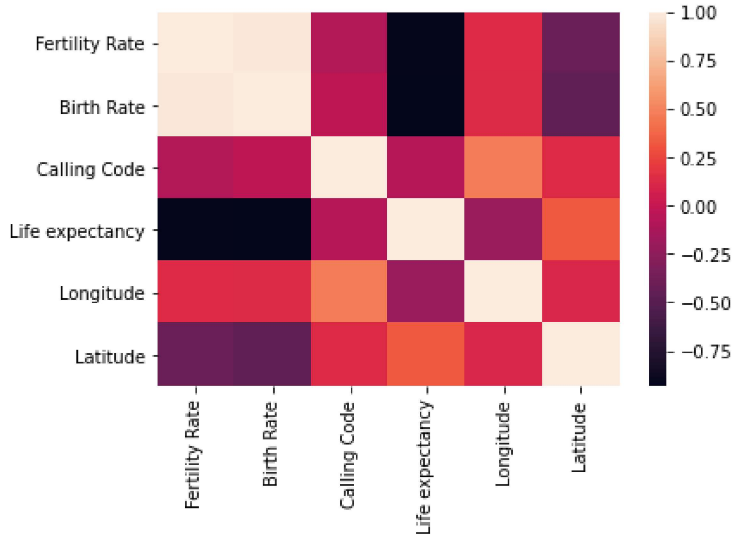


EDA and Visualization

In [163]:
```
#sns.distplot(data['Co2-Emissions'])
```

In [164]:
```
sns.heatmap(data1.corr())
```

Out[164]: <AxesSubplot:>



To train the model

we are going to train the linear regression model ;We need to split the two variable x and y where x in independent variable (input) and y is dependent of x(output) so we could ignore address columns as it is not requires for our model

In [165]:
```
x=data[['Fertility Rate','Birth Rate','Calling Code','Life expectancy','Longitude','Latitude']]
y=data1['Birth Rate']
```

In [166]:
```

#To split test and train data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.6)
```

In [167]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[167]: LinearRegression()

In [168]:
```
lr.intercept_
```

Out[168]: -1.8829382497642655e-13

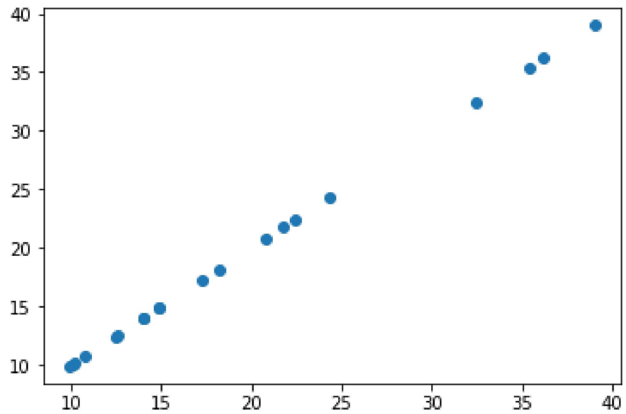
In [169]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=["Co-efficient"])
coeff
```

Out[169]:

	Co-efficient
Fertility Rate	-6.809439e-14
Birth Rate	1.000000e+00
Calling Code	1.248655e-16
Life expectancy	1.512232e-15
Longitude	-1.506115e-16
Latitude	1.800967e-16

In [170]:
```
prediction = lr.predict(x_train)
plt.scatter(y_train,prediction)
```

Out[170]: <matplotlib.collections.PathCollection at 0x23899d56cd0>



In [171]:
```
lr.score(x_test,y_test)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-171-1785cf3deb61> in <module>
----> 1 lr.score(x_test,y_test)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py in score(self, X, y, sample_weight)
    551
    552         from .metrics import r2_score
--> 553         y_pred = self.predict(X)
    554         return r2_score(y, y_pred, sample_weight=sample_weight)
    555

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_base.py in predict(self, X)
    236             Returns predicted values.
    237         """
--> 238         return self._decision_function(X)
    239
    240     _preprocess_data = staticmethod(_preprocess_data)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_base.py in _decision_function
(self, X)
    218         check_is_fitted(self)
    219
--> 220         X = check_array(X, accept_sparse=['csr', 'csc', 'coo'])
    221         return safe_sparse_dot(X, self.coef_.T,
    222                                dense_output=True) + self.intercept_

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwar
gs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, ac
cept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, en
sure_min_samples, ensure_min_features, estimator)
    661
    662         if force_all_finite:
--> 663             _assert_all_finite(array,
    664                                allow_nan=force_all_finite == 'allow-nan')
    665

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in _assert_all_finite(X,
allow_nan, msg_dtype)
    101                 not allow_nan and not np.isfinite(X).all()):
    102             type_err = 'infinity' if allow_nan else 'NaN, infinity'
--> 103             raise ValueError(
    104                     msg_err.format
    105                     (type_err,

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

In [172]:
```
lr.score(x_train,y_train)
```

Out[172]:
```
1.0
```

In [176]:
```
#from sklearn.linear_model import Ridge,Lasso
```

In [177]:
```
""""""rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
rr.score(x_test,y_test)""""
```

```
  File "<ipython-input-177-7221d7f5e5ba>", line 1
    """"""rr=Ridge(alpha=10)
          ^
SyntaxError: invalid syntax
```

In [175]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
la.score(x_test,y_test)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-175-3bb63ba76728> in <module>
      1 la=Lasso(alpha=10)
      2 la.fit(x_train,y_train)
----> 3 la.score(x_test,y_test)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py in score(self, X, y, sample_weight)
    551 
    552         from .metrics import r2_score
--> 553         y_pred = self.predict(X)
    554         return r2_score(y, y_pred, sample_weight=sample_weight)
    555 

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_base.py in predict(self, X)
    236             Returns predicted values.
    237         """
--> 238         return self._decision_function(X)
    239 
    240     _preprocess_data = staticmethod(_preprocess_data)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py in _deci
sion_function(self, X)
    896                                    dense_output=True) + self.intercept_
    897         else:
--> 898             return super()._decision_function(X)
    899 
    900 

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_base.py in _decision_function
(self, X)
    218         check_is_fitted(self)
    219 
--> 220         X = check_array(X, accept_sparse=['csr', 'csc', 'coo'])
    221         return safe_sparse_dot(X, self.coef_.T,
    222                                dense_output=True) + self.intercept_

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwar
gs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64 
     65             # extra_args > 0

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, ac
cept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, en
sure_min_samples, ensure_min_features, estimator)
    661 
    662         if force_all_finite:
--> 663             _assert_all_finite(array,
    664                                allow_nan=force_all_finite == 'allow-nan')
    665 

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in _assert_all_finite(X,
allow_nan, msg_dtype)
    101                 not allow_nan and not np.isfinite(X).all()):
    102             type_err = 'infinity' if allow_nan else 'NaN, infinity'
--> 103             raise ValueError(
    104                     msg_err.format
    105                     (type_err,

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

In [ ]:

In [ ]:

In [ ]: