DATA COLLECTION

In [23]:
```python
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [24]:
```python
# To Import Dataset
sd=pd.read_csv(r"c:\Users\user\Downloads\18_world-data-20231.csv")
sd
```

Out[24]:

	Country	Density\n(P/Km2)	Abbreviation	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Calling Code
0	Afghanistan	60	AF	0.581	652230.0	323000.0	32.49	93.0
1	Albania	105	AL	0.431	28748.0	9000.0	11.78	355.0
2	Algeria	18	DZ	0.174	2381741.0	317000.0	24.28	213.0
3	Andorra	164	AD	0.400	468.0	NaN	7.20	376.0
4	Angola	26	AO	0.475	1246700.0	117000.0	40.73	244.0
...	...	...	...	...	...	...	...	...
190	Venezuela	32	VE	0.245	912050.0	343000.0	17.88	58.0
191	Vietnam	314	VN	0.393	331210.0	522000.0	16.75	84.0
192	Yemen	56	YE	0.446	527968.0	40000.0	30.45	967.0
193	Zambia	25	ZM	0.321	752618.0	16000.0	36.19	260.0
194	Zimbabwe	38	ZW	0.419	390757.0	51000.0	30.68	263.0

195 rows × 35 columns

```
In [25]: # to display top 10 rows
         sd.head(10)
```

Out[25]:

	Country	Density\n(P/Km2)	Abbreviation	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Calling Code
0	Afghanistan	60	AF	0.581	652230.0	323000.0	32.49	93.0
1	Albania	105	AL	0.431	28748.0	9000.0	11.78	355.0
2	Algeria	18	DZ	0.174	2381741.0	317000.0	24.28	213.0
3	Andorra	164	AD	0.400	468.0	NaN	7.20	376.0
4	Angola	26	AO	0.475	1246700.0	117000.0	40.73	244.0
5	Antigua and Barbuda	223	AG	0.205	443.0	0.0	15.33	1.0
6	Argentina	17	AR	0.543	2780400.0	105000.0	17.02	54.0
7	Armenia	104	AM	0.589	29743.0	49000.0	13.99	374.0
8	Australia	3	AU	0.482	7741220.0	58000.0	12.60	61.0
9	Austria	109	AT	0.324	83871.0	21000.0	9.70	43.0

10 rows × 35 columns

DATA CLEANING AND PRE_PROCESSING

```
In [26]: sd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 35 columns):
 #   Column                                     Non-Null Count  Dtype  
---  ------                                     --------------  -----  
 0   Country                                    195 non-null    object 
 1   Density
(P/Km2)                          195 non-null    int64  
 2   Abbreviation                               188 non-null    object 
 3   Agricultural Land( %)                      188 non-null    float64
 4   Land Area(Km2)                             194 non-null    float64
 5   Armed Forces size                          171 non-null    float64
 6   Birth Rate                                 189 non-null    float64
 7   Calling Code                               194 non-null    float64
 8   Capital/Major City                         192 non-null    object 
 9   Co2-Emissions                              188 non-null    float64
 10  CPI                                        178 non-null    float64
 11  CPI Change (%)                             179 non-null    float64
 12  Currency-Code                              180 non-null    object 
 13  Fertility Rate                             188 non-null    float64
 14  Forested Area (%)                          188 non-null    float64
 15  Gasoline Price                             175 non-null    object 
 16  GDP                                        193 non-null    object 
 17  Gross primary education enrollment (%)     188 non-null    float64
 18  Gross tertiary education enrollment (%)    183 non-null    float64
 19  Infant mortality                           189 non-null    float64
 20  Largest city                               189 non-null    object 
 21  Life expectancy                            187 non-null    float64
 22  Maternal mortality ratio                   181 non-null    float64
 23  Minimum wage                               150 non-null    object 
 24  Official language                          194 non-null    object 
 25  Out of pocket health expenditure           188 non-null    float64
 26  Physicians per thousand                    188 non-null    float64
 27  Population                                 194 non-null    float64
 28  Population: Labor force participation (%)  176 non-null    float64
 29  Tax revenue (%)                            169 non-null    float64
 30  Total tax rate                             183 non-null    float64
 31  Unemployment rate                          176 non-null    float64
 32  Urban_population                           190 non-null    float64
 33  Latitude                                   194 non-null    float64
 34  Longitude                                  194 non-null    float64
dtypes: float64(25), int64(1), object(9)
memory usage: 53.4+ KB
```

In [27]:
```
# to display summary of statistics
sd.describe()
```

Out[27]:

	Density\n(P/Km2)	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Calling Code	E
count	195.000000	188.000000	1.940000e+02	1.710000e+02	189.000000	194.000000	1.880
mean	356.764103	0.391176	6.896244e+05	1.592749e+05	20.214974	360.546392	1.777
std	1982.888967	0.217831	1.921609e+06	3.806288e+05	9.945774	323.236419	8.387
min	2.000000	0.006000	0.000000e+00	0.000000e+00	5.900000	1.000000	1.100
25%	35.500000	0.217000	2.382825e+04	1.100000e+04	11.300000	82.500000	2.304
50%	89.000000	0.396000	1.195110e+05	3.100000e+04	17.950000	255.500000	1.230
75%	216.500000	0.553750	5.242560e+05	1.420000e+05	28.750000	506.750000	6.388
max	26337.000000	0.826000	1.709824e+07	3.031000e+06	46.080000	1876.000000	9.893

8 rows × 26 columns

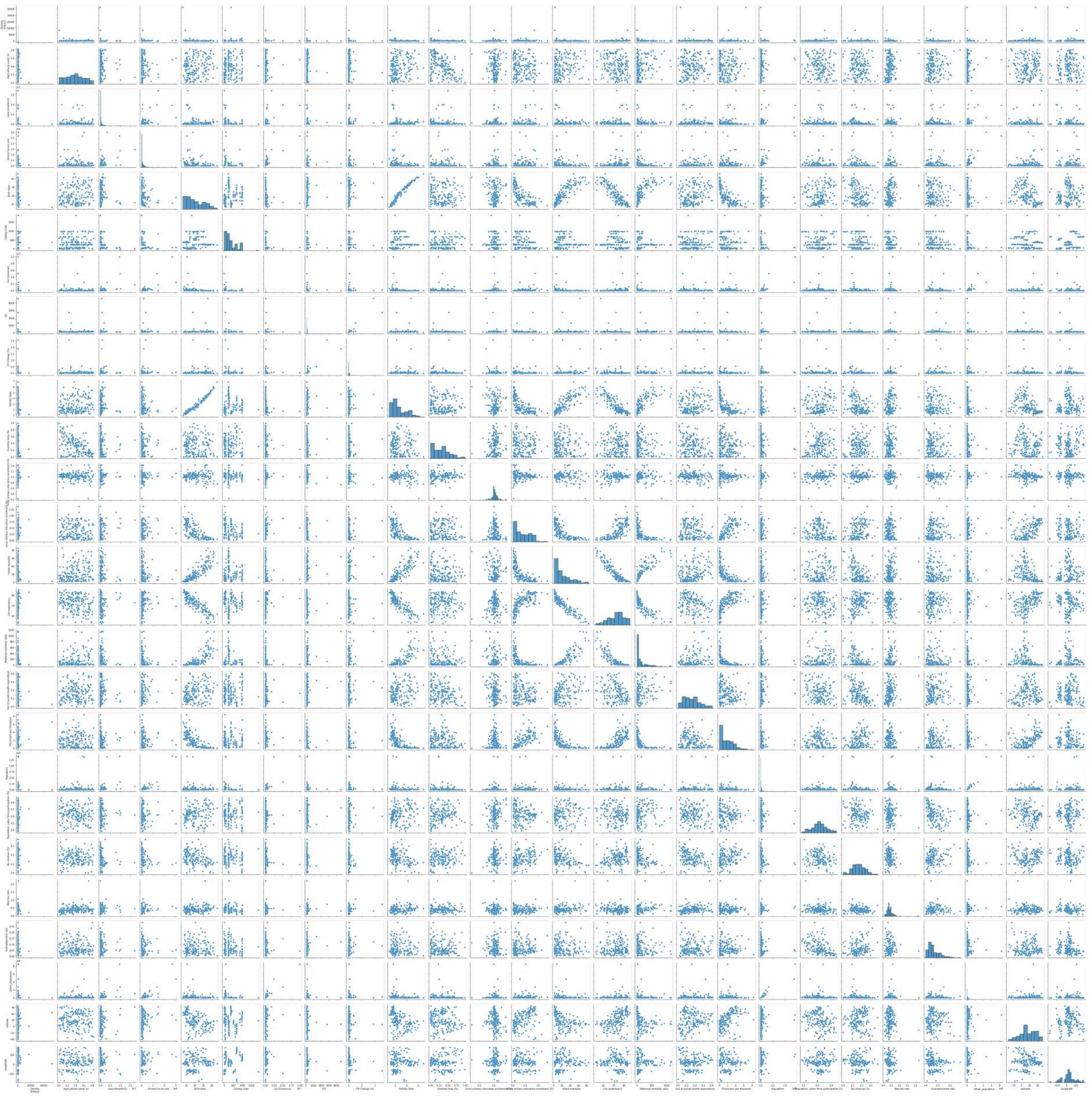
In [28]:
```
#to display colums heading
sd.columns
```

Out[28]:
```
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 expenditur
e',
       'Physicians per thousand', 'Population',
       'Population: Labor force participation (%)', 'Tax revenue (%)',
       'Total tax rate', 'Unemployment rate', 'Urban_population', 'Latitude',
       'Longitude'],
      dtype='object')
```

EDA and visualization

```
In [29]: sns.pairplot(sd)
```

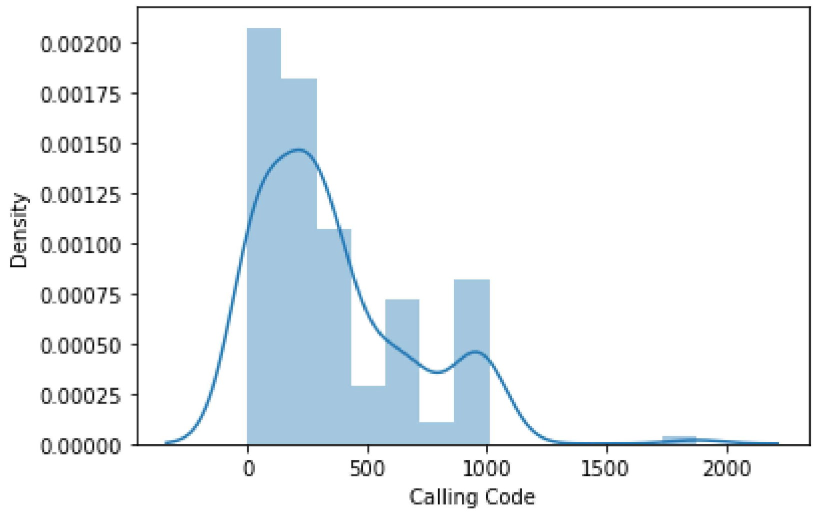
```
Out[29]: <seaborn.axisgrid.PairGrid at 0x24ff6f3fb50>
```



```
In [32]: sns.distplot(sd['Calling Code'])
```

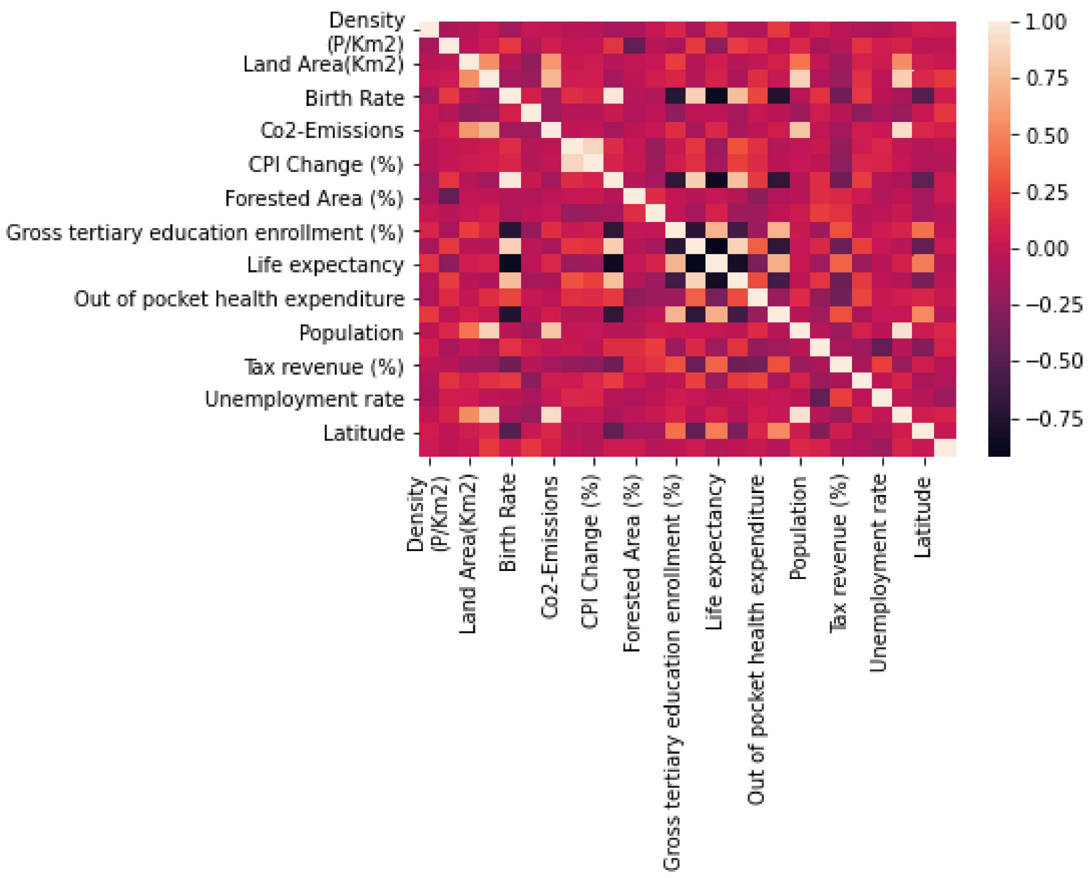
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function for hi
stograms).
  warnings.warn(msg, FutureWarning)
```

```
Out[32]: <AxesSubplot:xlabel='Calling Code', ylabel='Density'>
```



```
In [33]: sns.heatmap(sd.corr())
```

```
Out[33]: <AxesSubplot:>
```



```
In [38]: sd1=sd[[ 'Density\n(P/Km2)','Land Area(Km2)', 'Calling Code']]
```

TO TRAIN THE MODEL _MODEL BUILDING

we are goint train Liner Regression model; we need to split out the data into two varibles x and y where x is independent on x (output) and y is dependent on x(output) adress coloumn as it is not required our model

```
In [39]: x= sd1[['Density\n(P/Km2)','Land Area(Km2)']]
         y=sd1['Calling Code']
```

```
In [40]: # To split my dataset  into training data 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 [41]: from sklearn.linear_model import LinearRegression

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

```
Out[41]: LinearRegression()
```

In [42]:
```
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(x_train,y_train)
```
Out[42]:
```
LinearRegression()
```
In [43]:
```
print(lr.intercept_)
```
```
415.0872883901774
```
In [44]:
```
coeff= pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```
Out[44]:

	Co-efficient
Density\n(P/Km2)	-0.003028
Land Area(Km2)	-0.000038

```
In [45]: prediction = lr.predict(x_test)
         plt.scatter(y_test,prediction)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-45-10d398fd7dc3> in <module>
----> 1 prediction = lr.predict(x_test)
      2 plt.scatter(y_test,prediction)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_base.py in p
redict(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 inn
er_f(*args, **kwargs)
     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 che
ck_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force
_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, es
timator)
    661 
    662         if force_all_finite:
--> 663             _assert_all_finite(array,
    664                                allow_nan=force_all_finite == 'allow-n
an')
    665 

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in _as
sert_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('floa
t64').
```

```
In [46]: print(lr.score(x_test,y_test))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-46-6bc23016a4ce> in <module>
----> 1 print(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 p
redict(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 inn
er_f(*args, **kwargs)
     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 che
ck_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force
_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, es
timator)
    661 
    662         if force_all_finite:
--> 663             _assert_all_finite(array,
    664                                allow_nan=force_all_finite == 'allow-n
an')
    665 

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in _as
sert_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('floa
t64').
```

In [47]:
```
lr.score(x_train,y_train)
```

Out[47]:
```
0.04117891531509299
```

In [48]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [49]:
```
dr=Ridge(alpha=10)
dr.fit(x_train,y_train)
```

Out[49]:
```
Ridge(alpha=10)
```

In [51]:
```
dr.score(x_train,y_train)
```

Out[51]:
```
0.04117891531509288
```

In [52]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[52]:
```
Lasso(alpha=10)
```

In [55]:
```
la.score(x_train,y_train)
```

Out[55]:
```
0.04117891515707395
```

In [ ]:

In [ ]: