ex13

January 4, 2024

[13]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
sns.set_style("darkgrid", {"grid.color": ".6",
                           "grid.linestyle": ":"})

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import Lasso

from sklearn.ensemble import RandomForestRegressor
# from xgboost import XGBRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
```

[16]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv('/Users/thutranghoa/Code/Data_analysis/Data/data_cleaned.
 ↪csv')
data.head(10)
```

[16]:
```
  country  year  cereal_yield  fdi_perc_gdp  en_per_gdp    en_per_cap  \
0     AGO  1991         417.4      5.449515  179.271884    565.451027
1     ARE  1991        1594.0      0.076475  245.977706  12262.388130
2     ARG  1991        2666.1      1.285579  173.122857   1434.960601
3     AUS  1991        1603.3      1.306912  208.686644   4926.727783
4     AUT  1991        5463.0      0.209142  128.939160   3381.073790
5     BGD  1991        2585.7      0.004491  154.496130    116.511476
6     BGR  1991        3990.0      0.510803  367.387480   2560.054449
```

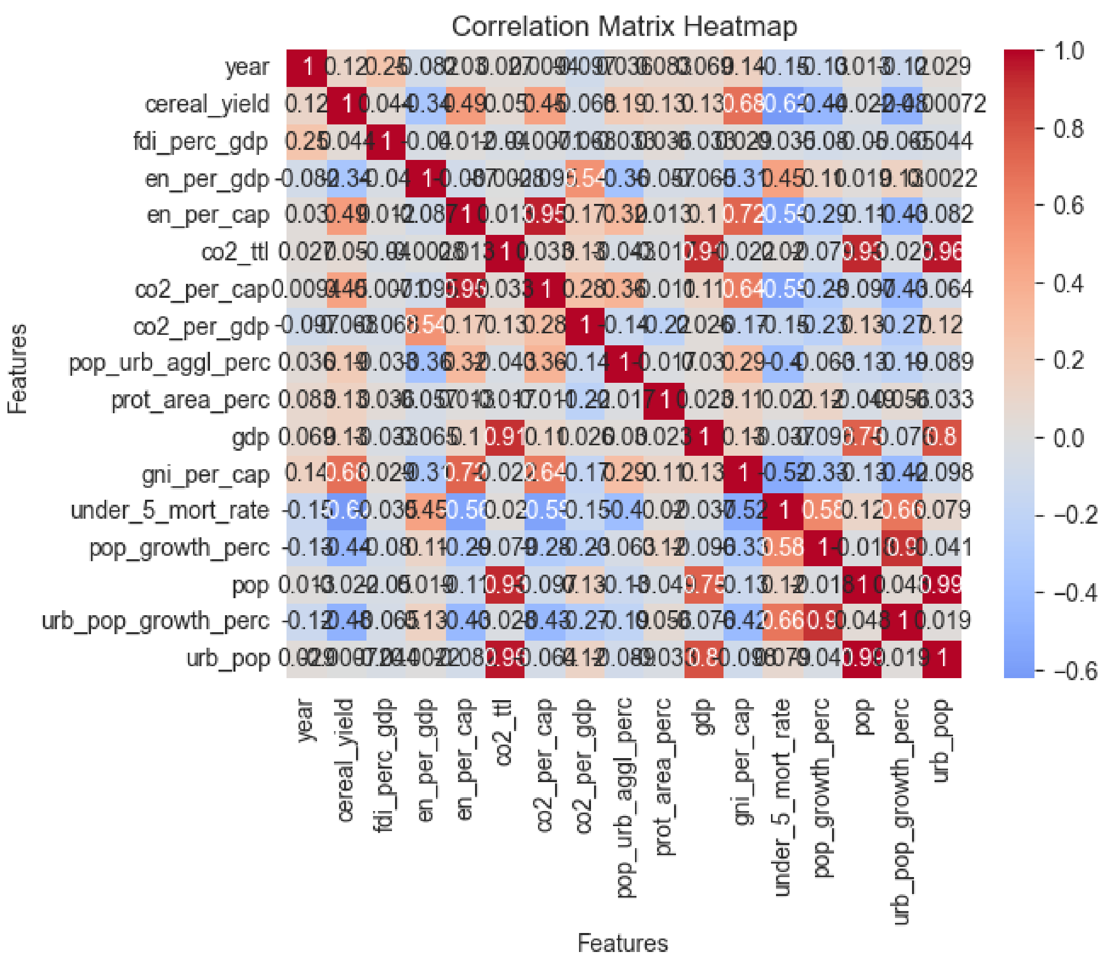
```
7     BOL  1991        1358.0      0.973189  124.898118    394.957523
8     BRA  1991        1850.6      0.270783  131.112804    939.256647
9     CAN  1991        2580.7      0.480500  283.945526   7388.563674

      co2_ttl  co2_per_cap  co2_per_gdp  pop_urb_aggl_perc  prot_area_perc  \
0    4367.397     0.409949   129.971142          15.290728       12.399822
1   57010.849    29.851550   598.807980          26.377204        0.266886
2  117021.304     3.536073   426.614517          39.119646        4.772468
3  281530.258    16.288490   689.948873          60.356798        7.915273
4   65888.656     8.448456   322.186648          19.746121       20.991143
5   15940.449     0.147913   196.135682           9.443704        1.537922
6   59706.094     6.916832   992.618530          13.789261        2.416870
7    5779.192     0.848156   268.213775          25.655982        8.583622
8  219330.604     1.441571   201.231977          35.137914       10.994541
9  449053.486    15.939889   612.576462          39.968292        6.016160

            gdp  gni_per_cap  under_5_mort_rate  pop_growth_perc  \
0  1.219375e+10        820.0              239.1         3.034866
1  3.391964e+10      19340.0               20.5         5.442852
2  1.897200e+11       3960.0               25.8         1.372593
3  3.299655e+11      18380.0                8.6         1.274577
4  1.721664e+11      21200.0                8.9         1.134999
5  3.095744e+10        300.0              137.9         2.359199
6  1.094355e+10       1620.0               22.3        -0.991363
7  5.343259e+09        760.0              116.7         2.306643
8  4.073378e+11       2870.0               57.1         1.654581
9  5.982081e+11      20420.0                8.0         1.360506

            pop  urb_pop_growth_perc       urb_pop
0  1.065352e+07             6.687032  4.099473e+06
1  1.909812e+06             5.265704  1.507988e+06
2  3.309358e+07             1.762636  2.890393e+07
3  1.728400e+07             1.438378  1.478473e+07
4  7.798899e+06             1.134999  5.131676e+06
5  1.077687e+08             4.260207  2.174773e+07
6  8.632000e+06            -0.570562  5.755818e+06
7  6.813834e+06             3.664292  3.840277e+06
8  1.521469e+08             2.453520  1.147188e+08
9  2.817168e+07             1.647301  2.164149e+07
```

[9]:
```
data.shape
```

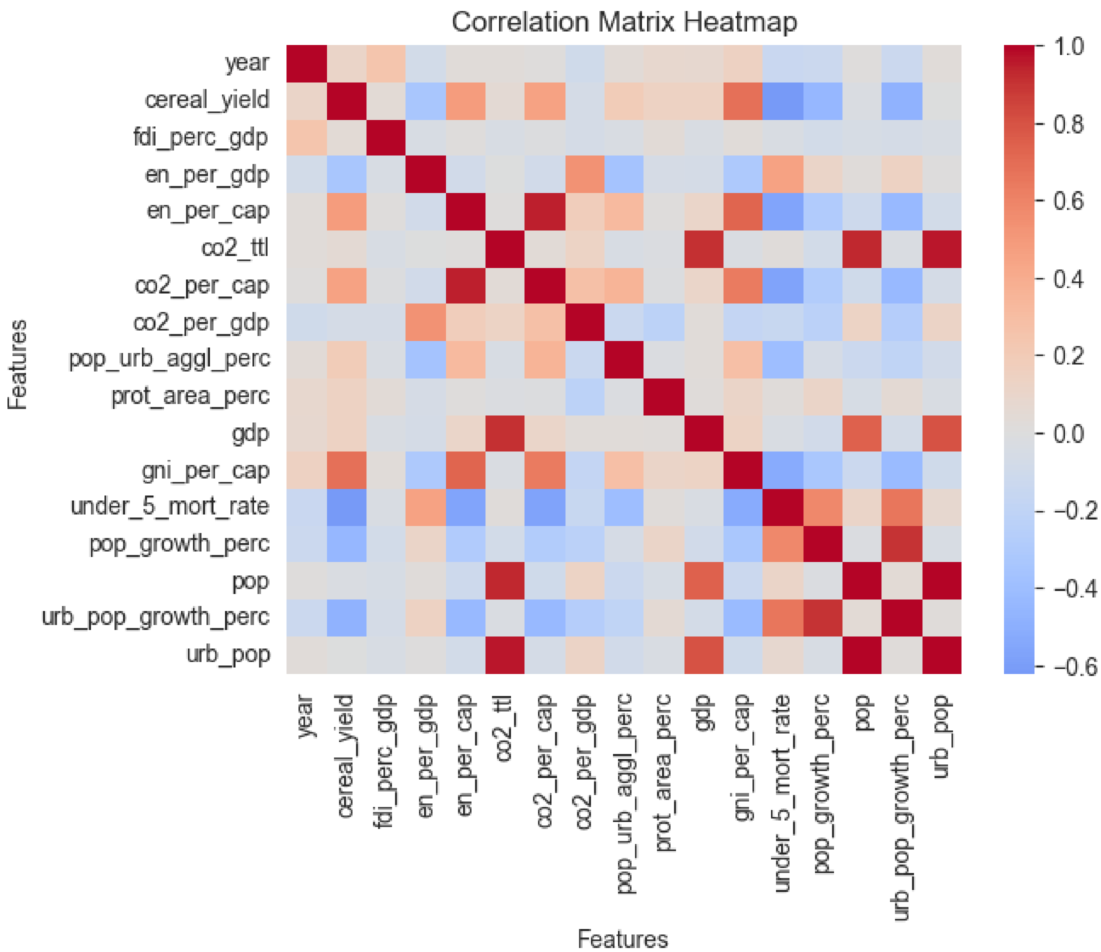
[9]: (1700, 18)

[30]:
```
# Calculate correlation matrix

df = data.drop (['country'], axis=1)
```

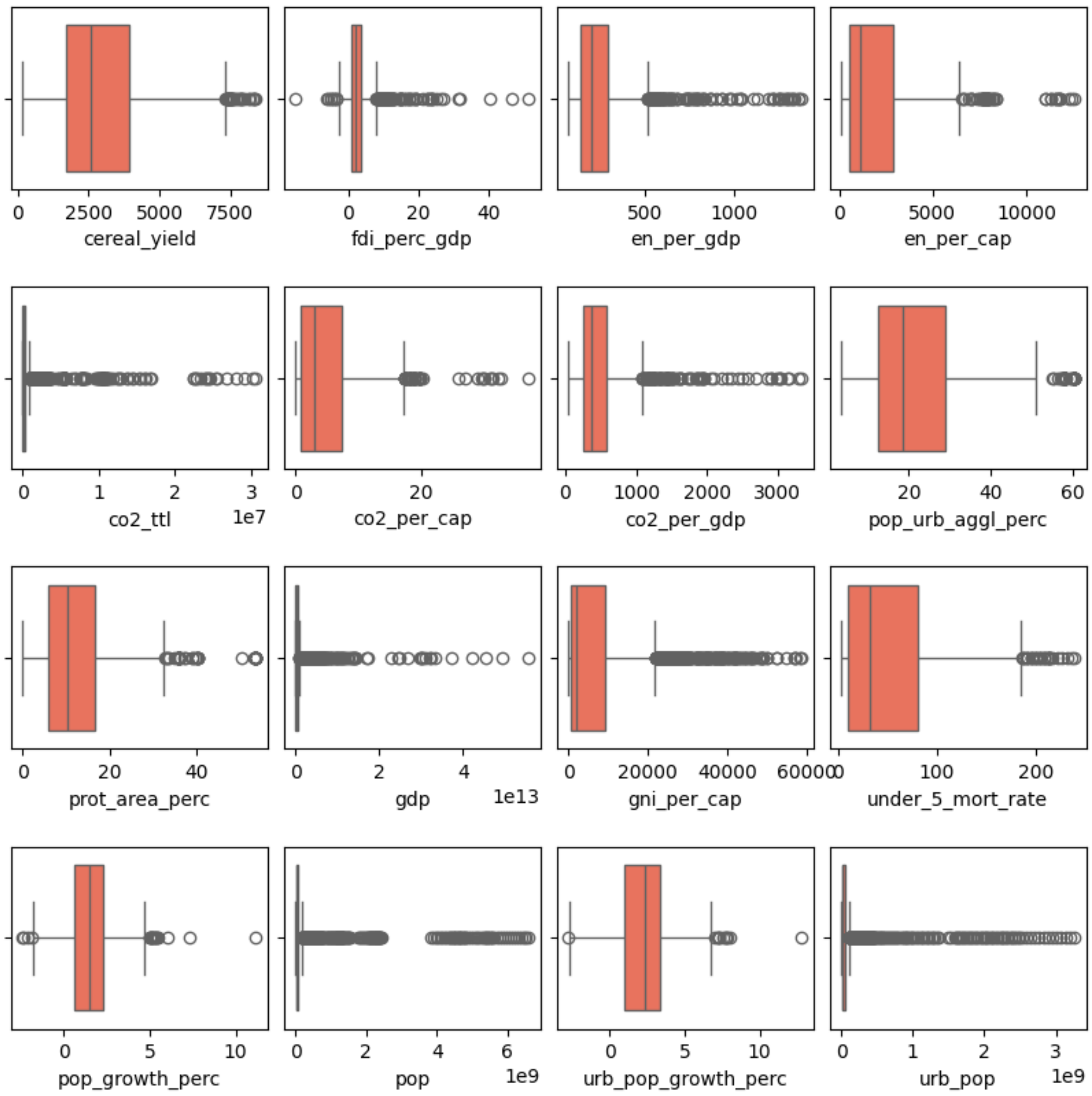
```python
correlation  = df.corr()

# Create heatmap
sns.heatmap(correlation, cmap='coolwarm',
            center=0, annot=True)

# Set title and axis labels
plt.title('Correlation Matrix Heatmap')
plt.xlabel('Features')
plt.ylabel('Features')
# plt.yticks(rotation = 30)

# Show plot
plt.show()
```



[29]:
```
df = data.drop (['country'], axis=1)
correlation  = df.corr()

# Create heatmap
sns.heatmap(correlation, cmap='coolwarm',
            center=0, annot=False)

# Set title and axis labels
plt.title('Correlation Matrix Heatmap')
plt.xlabel('Features')
plt.ylabel('Features')
# plt.yticks(rotation = 10)

# Show plot
plt.show()
```



[11]:
```
fig = plt.figure(figsize=(8, 8))
temp = data.drop(['country', 'year'], axis=1).columns.tolist()
for i, item in enumerate(temp):
    plt.subplot(4, 4, i+1)
    sns.boxplot(data=data, x=item, color='tomato')
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=2.0)
plt.show()
```



[12]:
```
def outlier_removal(column):
    # Capping the outlier rows with Percentiles
    upper_limit = column.quantile(.95)
    # set upper limit to 95percentile
    lower_limit = column.quantile(.05)
```

```
    # set lower limit to 5 percentile
    column.loc[(column > upper_limit)] = upper_limit
    column.loc[(column < lower_limit)] = lower_limit
    return column
```

[17]:
```
# select the features and target variable
X = data.drop(['en_per_gdp', 'co2_per_gdp', 'under_5_mort_rate', 'pop',␣
 ↪'urb_pop', 'country', 'year'], axis=1)

y = data['co2_ttl']
# dividing dataset in to train test
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

[18]:
```
X
```

[18]:
```
      cereal_yield  fdi_perc_gdp    en_per_cap     co2_ttl  co2_per_cap  \
0            417.4      5.449515    565.451027    4367.397     0.409949
1           1594.0      0.076475  12262.388130   57010.849    29.851550
2           2666.1      1.285579   1434.960601  117021.304     3.536073
3           1603.3      1.306912   4926.727783  281530.258    16.288490
4           5463.0      0.209142   3381.073790   65888.656     8.448456
…              …           …            …           …           …
1695        5064.2     10.611056    694.872260  127384.246     1.496485
1696         939.1      5.775544    320.226397   23384.459     1.033494
1697        4055.3      3.503662   3074.597450  435877.955     8.933203
1698         771.5     14.798970    356.023405    2816.256     0.045078
1699        2144.0      6.410991    614.851742    1888.505     0.152550

      pop_urb_aggl_perc  prot_area_perc           gdp  gni_per_cap  \
0             15.290728       12.399822  1.219375e+10        820.0
1             26.377204        0.266886  3.391964e+10      19340.0
2             39.119646        4.772468  1.897200e+11       3960.0
3             60.356798        7.915273  3.299655e+11      18380.0
4             19.746121       20.991143  1.721664e+11      21200.0
…                   …              …             …           …
1695          12.016197        6.166789  9.027376e+10        920.0
1696           9.356326        0.520661  2.691736e+10        970.0
1697          33.234023        6.859550  2.752787e+11       5860.0
1698          17.352516        9.986567  1.166838e+10        160.0
1699          11.109605       35.983018  1.464079e+10        970.0

      pop_growth_perc  urb_pop_growth_perc
0            3.034866             6.687032
1            5.442852             5.265704
2            1.372593             1.762636
3            1.274577             1.438378
4            1.134999             1.134999
```

```
…             …                  …
1695          1.064356           2.803530
1696          3.049598           4.960694
1697          1.104057           1.897450
1698          2.763286           4.605834
1699          2.653956           3.049996

[1700 rows x 11 columns]
```

[19]:

```
scaler = StandardScaler()

# Fit the StandardScaler on the training dataset
scaler.fit(x_train)

# Transform the training dataset
# using the StandardScaler
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

[20]:

```
from xgboost import XGBRegressor

# Create an instance of the XGBRegressor model
model_xgb = XGBRegressor()

# Fit the model to the training data
model_xgb.fit(x_train_scaled, y_train)

# Print the R-squared score on the training data
print("Xgboost Accuracy =", r2_score(
    y_train, model_xgb.predict(x_train_scaled)))
```

```
Xgboost Accuracy = 0.9999997679060656
```

[21]:

```
# Print the R-squared score on the test data
print("Xgboost Accuracy on test data =",
      r2_score(y_test,
               model_xgb.predict(x_test_scaled)))
```

```
Xgboost Accuracy on test data = 0.9981765844439002
```

[23]:

```
y_test = list (y_test)

plt.plot(y_test, color='red', label = 'Actual Value')
plt.plot(model_xgb.predict(x_test_scaled), color='green', label='Predicted␣
 ↪Value')
plt.title('Actual vs Predicted Price')
plt.xlabel('Number of values')
plt.ylabel('GLD Price')
```

```
plt.legend()
plt.show()
```