Project 1- 430

Names :

1) Dishant Vaishnav- 106545424

2) Ketki Sawant- 506353861

3) Praharsh Mehrotra- 906306085

4) Teesta Bose- 206549884

Heading: Global CO2 Emissions: Analyzing Economic and Policy Impacts

Motivation Of Project:

One of the most important global policy issues in the 21st century is the question of how to reduce carbon emissions and the impact of global warming. This is because of the immense impact global warming is expected to have on the economy of the future.

In this project, we aim to explore the key factors affecting per capita CO2 emissions across nations. We are interested in examining both how macroeconomic factors such as GDP per capita, share of industry, urbanization, technology growth, and environmental and policy-related factors like fossil fuel subsidies and renewable energy adoption affect carbon emissions. Furthermore, we seek to understand what factors seem to be the most important influencers of CO2 emissions and, thus, could potentially be the focus of environmental policy making.

0. Data Description

The dependent variable in this project is Per Capita CO2 Emissions, while the other variables serve as explanatory factors.

- Per Capita CO2 Emissions: This is the total carbon dioxide emissions generated by a country, divided by its population, representing the average emissions per individual in that country. (Source: Global Carbon Budget (2023); Population based on various sources (2023))
- Fossil Fuel Subsidies per Capita: This variable represents the total financial support provided by the government to fossil fuel producers or consumers. Production subsidies lower the cost of extracting coal, oil, or gas, while consumption subsidies reduce fuel prices for end users.

(Source: International Energy Agency, Organisation for Economic Co-operation and Development and International Monetary Fund via United Nations Global SDG Database) (2021-23)

- Population: The total number of individuals living in a country. (Source: HYDE (2023); Gapminder (2022); UN WPP (2024)) (2021-23)
- Per Capita GDP (PPP, 2017, Constant in $): This is a measure of a nation's economic output per person, calculated as the gross domestic product (GDP) divided by the population. Per capita GDP in PPP (2017 constant dollars) reflects economic output per person, adjusted for both purchasing power differences between countries and inflation over time. (Source: World Bank) (Years: 2020-2021)
- Per Capita Energy Consumption (kWh/person): The total energy consumed by a country, divided by its population, reflecting national energy demand and usage, often tied to industrial activity and efficiency. (Source: U.S. Energy Information Administration (2023); Energy Institute - Statistical Review of World Energy (2024); Population based on various sources (2023)) (Years: 2021-22-23)
- Renewable Energy Share: This variable indicates the share of a country's total energy consumption derived from renewable sources like wind, solar, and hydropower. (Source: Energy Institute - Statistical Review of World Energy (2024)) (Years: 2021-22-23)
- Industry Share: This represents the percentage of a country's economic output contributed by industrial activities. Nations with larger industrial sectors typically have higher energy demands and may produce (Source: World Bank) (Year: 2023)
- GII Technology: This metric, from the Global Innovation Index (GII), reflects a country’s technological capabilities and potential for innovation. (Source: Global Innovation Index, World Intellectual Property Organization) (Years: 2020-21-22)
- Urbanization: The percentage of a country's population living in urban areas. (Source: Multiple sources compiled by World Bank (2024)) (Years: 2020-21-22)

In [182…
```
# Importing Libraries
# Import Modules

# Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from numpy import random; random.seed(10) # pre-setting seed
from scipy import stats
from scipy import optimize  # for Box-Cox calculations
from scipy.stats import norm
from matplotlib import rcParams # for ease of resizing plots
# Numpy version matters for scipy

# For model fitting
import statsmodels.api as sm
import statsmodels.stats.api as sms
import statsmodels.formula.api as smf
from fitter import Fitter # might require install, numpy version matters
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing
from sklearn import metrics
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder
from scipy.stats import binomtest, ks_2samp


from sklearn.ensemble import RandomForestRegressor
from boruta import BorutaPy
from BorutaShap import BorutaShap # use .py file version


# Subset regressions & feature selection
from sklearn.linear_model import LinearRegression
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
from mlxtend.feature_selection import SequentialFeatureSelector as SFS

#from scipy.stats import binomtest, ks_2samp
from ydata_profiling import ProfileReport


#Importing R
import os
os.environ['R_HOME'] = '/Library/Frameworks/R.framework/Resources'
import rpy2

import warnings
warnings.filterwarnings('ignore')

from rpy2.robjects import pandas2ri
import rpy2.rinterface as rinterface
from rpy2 import robjects
pandas2ri.activate()


%load_ext rpy2.ipython
%reload_ext rpy2.ipython
```

```
The rpy2.ipython extension is already loaded. To reload it, use:
  %reload_ext rpy2.ipython
```

In [183…
```
#Import data
raw_data = pd.read_csv("/Users/ketkisawant/Documents/430/data_compiled.csv")

#Cleaning_data

continents = pd.read_csv('/Users/ketkisawant/Documents/430/continents-according-to-our-world-in-data.csv')

raw_data_cleaned = pd.merge(raw_data, continents, on = 'Entity', how = 'inner')
#on =entity means merging both datasets using the common column entity
#how ='inner'specifies the type of merge,keeps only the rows that have matching values in both raw_data and cont
raw_data_cleaned = raw_data_cleaned.drop(['Code', 'Year'], axis = 1)
#removing code and year column from the continent data set and axis=1 means column 1

raw_data_cleaned['fossil_fuel_subsidies'] = pd.to_numeric(raw_data_cleaned['fossil_fuel_subsidies'], errors='co
```

In [184…
```
#KNN
#replace a missing observation with a weighted average of the closest (using the Euclidean distance)
raw_data_knn_test = raw_data_cleaned.copy() #creating a copy of cleaned data and assigning it to new data
#helps to ensure old data is not affected
```

```
raw_data_knn_test = raw_data_knn_test.drop(['Entity'], axis = 1) #removing entity column, axis=1 represents colu

# One-hot encode categorical variables and apply KNN imputation
raw_data_knn_test = pd.get_dummies(raw_data_knn_test, columns=['Continent'])
raw_data_knn_test
# for the computer to understand, .get_dummies will create a new column for every continent and assign values 0


# Create a KNNImputer instance
imputer = KNNImputer(n_neighbors= 3)
#handle missing values by employing KNN imputation
#n=3 takes three near neighbours and takes average to fill it up

# Impute missing values
raw_data_knn_test_imputed = imputer.fit_transform(raw_data_knn_test)
#Fit: It analyzes the data to find the nearest neighbors for each row where there are missing values.
#Transform: It fills in the missing values based on the nearest neighbors' values.
#The result is a NumPy array with all missing values imputed.

# Convert the result back to a dataframe
raw_data_knn_test_imputed = pd.DataFrame(raw_data_knn_test_imputed, columns=raw_data_knn_test.columns)

(raw_data_knn_test_imputed)

raw_data_cleaned = raw_data_knn_test_imputed

raw_data_cleaned = raw_data_cleaned.drop(['Continent_Africa',
                                          'Continent_Oceania', 'Continent_North America',
                                          'Continent_Europe', 'Continent_South America', 'Continent_Asia'],
                                         axis=1)

raw_data_cleaned
```

Out[184…

	per_capita_co2_emissions	fossil_fuel_subsidies	population	per_capita_gdp	Urbanization	renewable_energy_share	per_cap
0	0.302	2.735000	4.067801e+07	1742.307000	26.318700	15.838400	
1	1.737	53.655567	2.829639e+06	14455.506000	62.960000	21.215467	
2	3.972	348.876700	4.546757e+07	11024.048000	74.255300	0.255000	
3	8.900	13.258333	4.838533e+04	28684.551667	87.172700	13.605600	
4	4.673	99.693333	7.965500e+04	33643.285333	87.861700	13.605600	
...	...	...	...	...	...	...	
231	3.625	18.506700	9.965598e+07	10825.124000	38.052700	24.585400	
232	2.258	13.258333	1.149500e+04	10264.042333	82.083667	13.605600	
233	0.342	0.005000	3.825130e+07	30304.380667	38.547300	15.875267	
234	0.440	10.455000	2.016017e+07	3278.982000	45.194000	17.122533	
235	0.523	0.000000	1.606904e+07	2104.474000	32.313300	21.673533	

236 rows × 9 columns

Question 1

Part a) Descriptive Analysis Of Variables

In [185…

```
# data describe

raw_data_cleaned.describe()
```

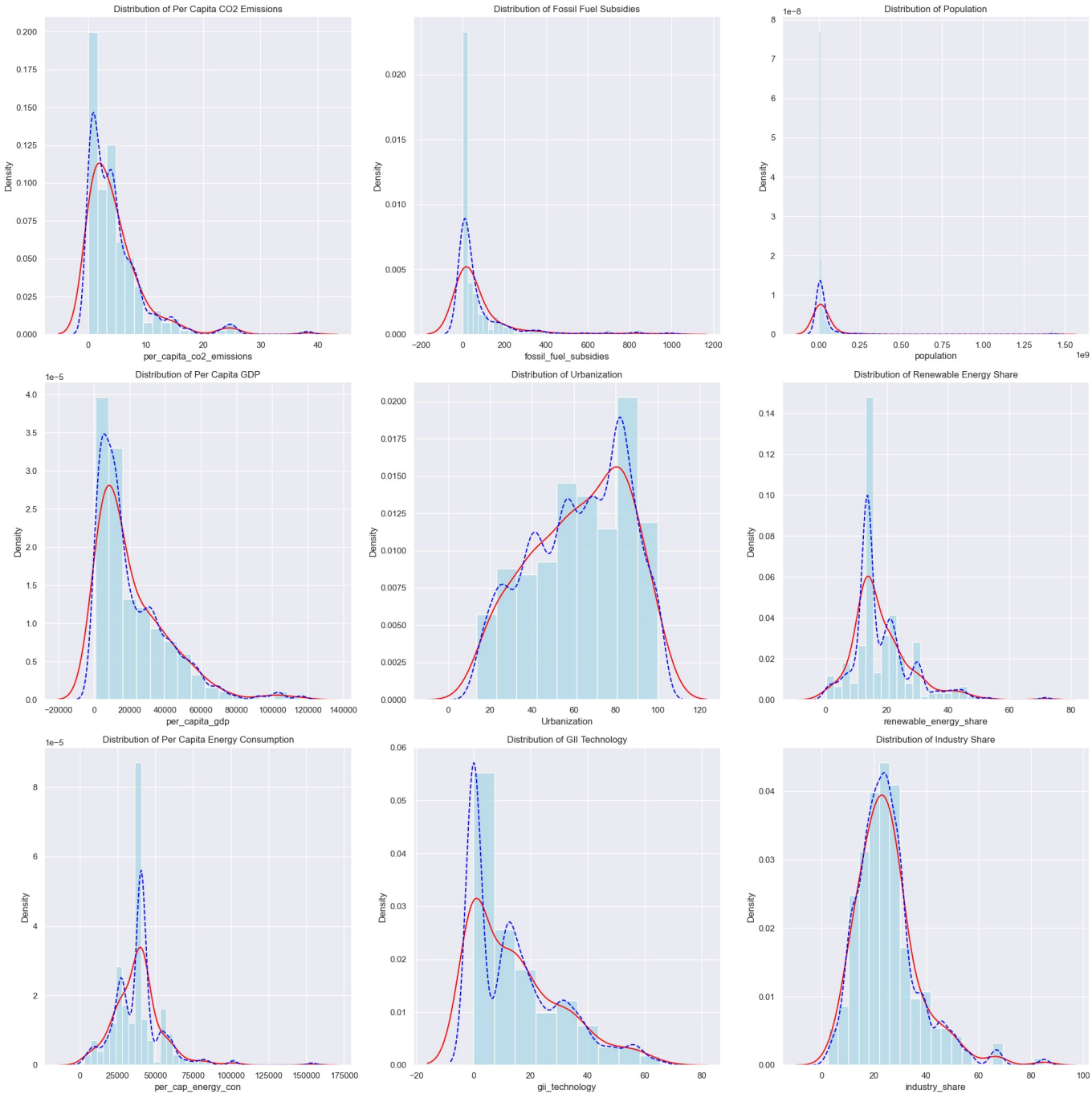
Out[185…

	per_capita_co2_emissions	fossil_fuel_subsidies	population	per_capita_gdp	Urbanization	renewable_energy_share	per_c
count	236.000000	236.000000	2.360000e+02	236.000000	236.000000	236.000000	
mean	4.617240	83.611463	3.399101e+07	21409.295904	62.791892	18.173640	
std	5.191257	167.269398	1.366796e+08	20464.773749	23.276689	9.755889	
min	0.037000	0.000000	5.136670e+02	711.200000	13.461000	0.242400	
25%	1.115750	1.878750	3.950424e+05	5916.773000	43.371400	13.605600	
50%	3.617000	13.258333	5.469677e+06	14226.321500	66.535850	13.605600	
75%	6.097500	77.855825	2.263032e+07	31266.279000	82.083667	21.537400	
max	38.206000	999.630000	1.425899e+09	116177.270000	100.000000	71.720000	

In [186…
```
# Check for NA Values

raw_data_cleaned.isna().sum()
```

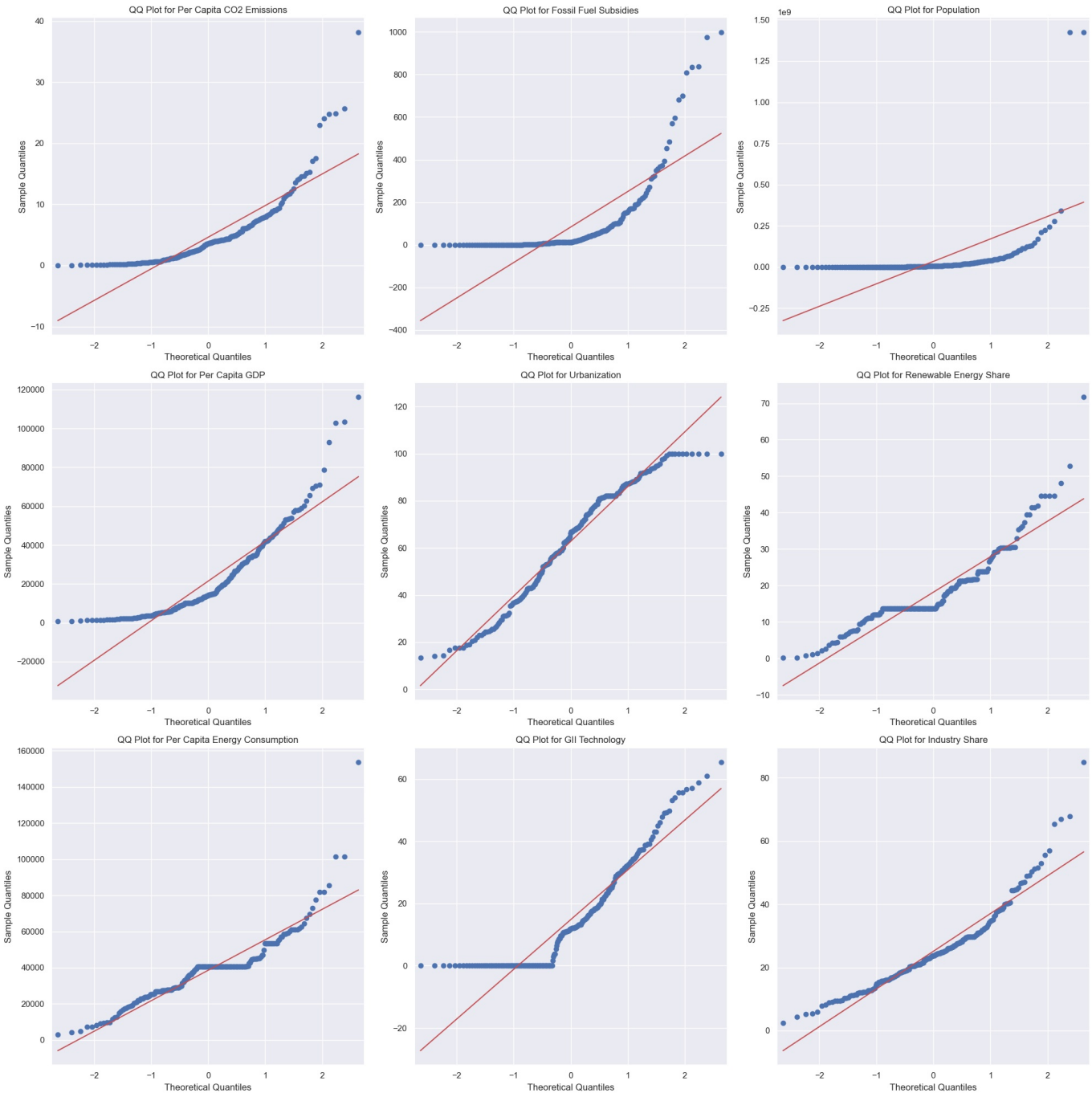
Out[186…
```
per_capita_co2_emissions    0
fossil_fuel_subsidies       0
population                  0
per_capita_gdp              0
Urbanization                0
renewable_energy_share      0
per_cap_energy_con          0
gii_technology              0
industry_share              0
dtype: int64
```

In [187…
```
#Density Plots (Histograms And KDE Plots)
fig, ax = plt.subplots(3,3, figsize = (20,20))

sns.histplot(raw_data_cleaned['per_capita_co2_emissions'], kde = False, stat = 'density', color = 'lightblue', a
sns.kdeplot(raw_data_cleaned['per_capita_co2_emissions'], color = 'red', ax = ax[0,0])
sns.kdeplot(raw_data_cleaned['per_capita_co2_emissions'], bw_adjust = 0.5, color = 'blue', linestyle = '--', ax
ax[0,0].set_title('Distribution of Per Capita CO2 Emissions')

sns.histplot(raw_data_cleaned['fossil_fuel_subsidies'], kde = False, stat = 'density', color = 'lightblue', ax =
sns.kdeplot(raw_data_cleaned['fossil_fuel_subsidies'], color = 'red', ax = ax[0,1])
sns.kdeplot(raw_data_cleaned['fossil_fuel_subsidies'], bw_adjust = 0.5, color = 'blue', linestyle = '--', ax = a
ax[0,1].set_title('Distribution of Fossil Fuel Subsidies')


sns.histplot(raw_data_cleaned['population'], kde = False, stat = 'density', color = 'lightblue', ax = ax[0,2])
sns.kdeplot(raw_data_cleaned['population'], color = 'red', ax = ax[0,2])
sns.kdeplot(raw_data_cleaned['population'], bw_adjust = 0.5, color = 'blue', linestyle = '--', ax = ax[0,2])
ax[0,2].set_title('Distribution of Population')

sns.histplot(raw_data_cleaned['per_capita_gdp'], kde = False, stat = 'density', color = 'lightblue', ax = ax[1,(
sns.kdeplot(raw_data_cleaned['per_capita_gdp'], color = 'red', ax = ax[1,0])
sns.kdeplot(raw_data_cleaned['per_capita_gdp'], bw_adjust = 0.5, color = 'blue', linestyle = '--', ax = ax[1,0]
ax[1,0].set_title('Distribution of Per Capita GDP')

sns.histplot(raw_data_cleaned['Urbanization'], kde = False, stat = 'density', color = 'lightblue', ax = ax[1,1]
sns.kdeplot(raw_data_cleaned['Urbanization'], color = 'red', ax = ax[1,1])
sns.kdeplot(raw_data_cleaned['Urbanization'], bw_adjust = 0.5, color = 'blue', linestyle = '--', ax = ax[1,1])
ax[1,1].set_title('Distribution of Urbanization')

sns.histplot(raw_data_cleaned['renewable_energy_share'], kde = False, stat = 'density', color = 'lightblue', ax
sns.kdeplot(raw_data_cleaned['renewable_energy_share'], color = 'red', ax = ax[1,2])
sns.kdeplot(raw_data_cleaned['renewable_energy_share'], bw_adjust = 0.5, color = 'blue', linestyle = '--', ax =
ax[1,2].set_title('Distribution of Renewable Energy Share')

sns.histplot(raw_data_cleaned['per_cap_energy_con'], kde = False, stat = 'density', color = 'lightblue', ax = a
sns.kdeplot(raw_data_cleaned['per_cap_energy_con'], color = 'red', ax = ax[2,0])
sns.kdeplot(raw_data_cleaned['per_cap_energy_con'], bw_adjust = 0.5, color = 'blue', linestyle = '--', ax = ax[
ax[2,0].set_title('Distribution of Per Capita Energy Consumption')

sns.histplot(raw_data_cleaned['gii_technology'], kde = False, stat = 'density', color = 'lightblue', ax = ax[2,
sns.kdeplot(raw_data_cleaned['gii_technology'], color = 'red', ax = ax[2,1])
sns.kdeplot(raw_data_cleaned['gii_technology'], bw_adjust = 0.5, color = 'blue', linestyle = '--', ax = ax[2,1]
ax[2,1].set_title('Distribution of GII Technology')


sns.histplot(raw_data_cleaned['industry_share'], kde = False, stat = 'density', color = 'lightblue', ax = ax[2,
sns.kdeplot(raw_data_cleaned['industry_share'], color = 'red', ax = ax[2,2])
sns.kdeplot(raw_data_cleaned['industry_share'], bw_adjust = 0.5, color = 'blue', linestyle = '--', ax = ax[2,2]
ax[2,2].set_title('Distribution of Industry Share')

plt.tight_layout()
plt.show()
```



In [181…

```
#Check Whether variables are normally distributed

fig, ax = plt.subplots(3,3, figsize = (20,20))

sm.qqplot(raw_data_cleaned['per_capita_co2_emissions'], line = 's', ax = ax[0,0])
ax[0,0].set_title('QQ Plot for Per Capita CO2 Emissions')
sm.qqplot(raw_data_cleaned['fossil_fuel_subsidies'], line = 's', ax = ax[0,1])
ax[0,1].set_title('QQ Plot for Fossil Fuel Subsidies')
sm.qqplot(raw_data_cleaned['population'], line = 's', ax = ax[0,2])
ax[0,2].set_title('QQ Plot for Population')
sm.qqplot(raw_data_cleaned['per_capita_gdp'], line = 's', ax = ax[1,0])
ax[1,0].set_title('QQ Plot for Per Capita GDP')
sm.qqplot(raw_data_cleaned['Urbanization'], line = 's', ax = ax[1,1])
ax[1,1].set_title('QQ Plot for Urbanization')
sm.qqplot(raw_data_cleaned['renewable_energy_share'], line = 's', ax = ax[1,2])
ax[1,2].set_title('QQ Plot for Renewable Energy Share')
sm.qqplot(raw_data_cleaned['per_cap_energy_con'], line = 's', ax = ax[2,0])
ax[2,0].set_title('QQ Plot for Per Capita Energy Consumption')
sm.qqplot(raw_data_cleaned['gii_technology'], line = 's', ax = ax[2,1])
ax[2,1].set_title('QQ Plot for GII Technology')
sm.qqplot(raw_data_cleaned['industry_share'], line = 's', ax = ax[2,2])
ax[2,2].set_title('QQ Plot for Industry Share')
plt.tight_layout()
plt.show()
```



In [188…

```
#Jarque Bera Test For Normality
# List of variables to test
v = ['per_capita_co2_emissions', 'fossil_fuel_subsidies', 'population', 'per_capita_gdp', 'renewable_energy_sha

# Loop through each variable and run the Jarque-Bera test
for var in v:
    print(f'\n===== Jarque-Bera Test for {var} =====')

# Run Jarque-Bera test on the variable
    data = raw_data_cleaned[var]

    jb_test_stat, jb_test_pvalue, jb_test_skew, jb_test_kurtosis = sms.jarque_bera(data)

# Print the test results
    print(f"Test Statistic: {jb_test_stat}\n"
f"p-value: {jb_test_pvalue}\n"
f"Skewness: {jb_test_skew}\n"
f"Kurtosis: {jb_test_kurtosis}")
```

```
===== Jarque-Bera Test for per_capita_co2_emissions =====
Test Statistic: 1278.5589104931892
p-value: 2.314516026257474e-278
Skewness: 2.649210126174947
Kurtosis: 13.09701356241752

===== Jarque-Bera Test for fossil_fuel_subsidies =====
Test Statistic: 1857.3395216057793
p-value: 0.0
Skewness: 3.3299560361182614
Kurtosis: 15.021961422252343

===== Jarque-Bera Test for population =====
Test Statistic: 81135.24944140471
p-value: 0.0
Skewness: 9.111297989450069
Kurtosis: 91.98864709537294

===== Jarque-Bera Test for per_capita_gdp =====
Test Statistic: 222.79836359187257
p-value: 4.1682144754634425e-49
Skewness: 1.664247993506886
Kurtosis: 6.402730546001467

===== Jarque-Bera Test for renewable_energy_share =====
Test Statistic: 282.7830070154051
p-value: 3.930522064617908e-62
Skewness: 1.562368897295846
Kurtosis: 7.358165628887945

===== Jarque-Bera Test for per_cap_energy_con =====
Test Statistic: 1056.4599375400694
p-value: 3.914167325596546e-230
Skewness: 1.835971660457285
Kurtosis: 12.692958065813812

===== Jarque-Bera Test for gii_technology =====
Test Statistic: 39.59522761479145
p-value: 2.523513129180102e-09
Skewness: 0.9962976088127916
Kurtosis: 3.2370603624693643

===== Jarque-Bera Test for industry_share =====
Test Statistic: 187.67911156390062
p-value: 1.7619709330395206e-41
Skewness: 1.3721993177335203
Kurtosis: 6.399159819003523
```

In [189…

```
#Correlation Matrix
plt.figure(figsize=(13,7))
c= raw_data_cleaned[['per_capita_co2_emissions', 'fossil_fuel_subsidies', 'population', 'per_capita_gdp', 'Urba
print(c)
sns.heatmap(c,annot=True, cmap="Blues")
plt.show()
```

	per_capita_co2_emissions	fossil_fuel_subsidies
per_capita_co2_emissions	1.000000	0.574576
fossil_fuel_subsidies	0.574576	1.000000
population	-0.008211	-0.044207
per_capita_gdp	0.608999	0.397288
Urbanization	0.455464	0.252520
renewable_energy_share	-0.061228	0.027524
per_cap_energy_con	0.165868	0.083499
gii_technology	0.199125	0.176786
industry_share	0.270647	0.316870

	population	per_capita_gdp	Urbanization
per_capita_co2_emissions	-0.008211	0.608999	0.455464
fossil_fuel_subsidies	-0.044207	0.397288	0.252520
population	1.000000	-0.054239	-0.070463
per_capita_gdp	-0.054239	1.000000	0.555398
Urbanization	-0.070463	0.555398	1.000000
renewable_energy_share	-0.093938	0.128459	0.104996
per_cap_energy_con	-0.173104	0.281697	0.176215
gii_technology	0.275055	0.488193	0.239634
industry_share	0.074981	0.069751	0.048577

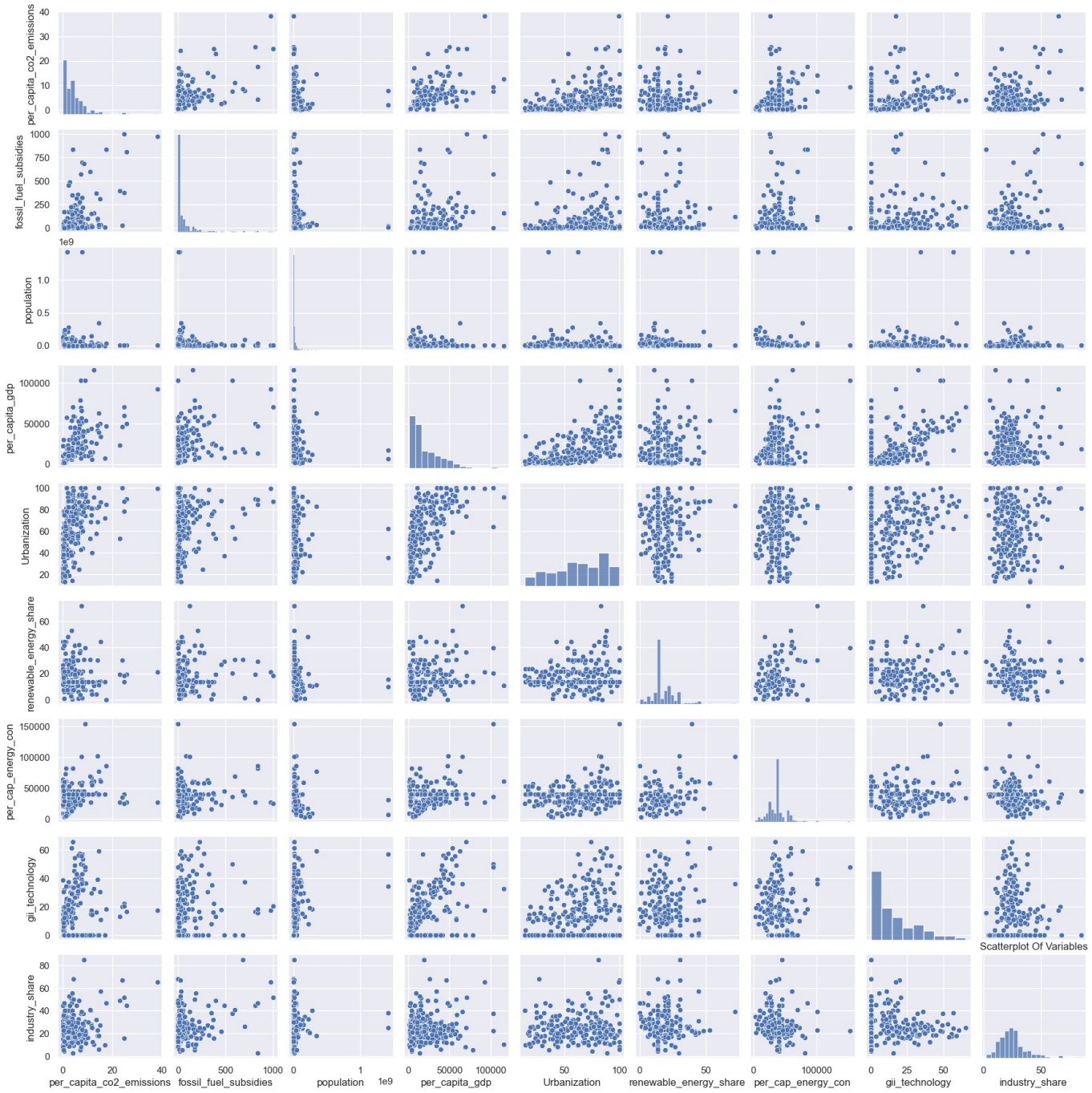
	renewable_energy_share	per_cap_energy_con
per_capita_co2_emissions	-0.061228	0.165868
fossil_fuel_subsidies	0.027524	0.083499
population	-0.093938	-0.173104
per_capita_gdp	0.128459	0.281697
Urbanization	0.104996	0.176215
renewable_energy_share	1.000000	0.319679
per_cap_energy_con	0.319679	1.000000
gii_technology	0.166082	0.024063
industry_share	0.109641	-0.134854

	gii_technology	industry_share
per_capita_co2_emissions	0.199125	0.270647
fossil_fuel_subsidies	0.176786	0.316870
population	0.275055	0.074981
per_capita_gdp	0.488193	0.069751
Urbanization	0.239634	0.048577
renewable_energy_share	0.166082	0.109641
per_cap_energy_con	0.024063	-0.134854
gii_technology	1.000000	0.028744
industry_share	0.028744	1.000000

In [190…

```
# Scatterplot matrix

sns.set()
sns.pairplot(raw_data_cleaned, height = 2.0)
```

```
plt.title('Scatterplot Of Variables')
plt.show()
```



Most regressors (x variables) show little correlation between each other. Exceptions to this is GDP Per capita being correlated to Urbanization and Technology Level (GII Technology)

Part B)- Panda Profiling Report

In [191…
```
# Panda Profiling Report

ProfileReport(raw_data_cleaned)
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Pandas Profiling Report

Overview

Brought to you by YData

Overview | Alerts 9 | Reproduction

Dataset statistics

Number of variables	9
Number of observations	236
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	16.7 KiB
Average record size in memory	72.6 B

Variable types

Numeric	9

Variables

Select Columns

Out[191…

Reconfirms that there are no missing or NA values. Also, shows that Per Capita GDP, fossil fuel subsidies, urbanization and per capita co2 emissions are variables with high pairwise correlations with each other. This could potentially lead to multi-collinearity issues in case of a multi linear regression. But, for pairwise single linear regressions, might be good.

Part C)- Estimate Density Distributions For All Variables

We are analyzing the statistical distribution of each variable to understand how they behave and how values are spread or accumulated over their range. This will help us identify patterns, relationships, or potential biases in the data. Knowing this will guide further analysis or model selection. For example, certain statistical models assume normality, while others can handle skewed data.

Kernel Density Estimation (KDE): We have estimated the empirical density using the KDE method. The KDE is a non-parametric way to estimate the probability density function (PDF) of a variable. Here, instead of assuming that the data follows a known distribution (eg. normal, exponential, etc.), we let the data itself guide the shape of the distribution.

Empirical Cumulative Distribution Function (ECDF): The ECDF is a step function that will provide a view of how data accumulates and

how likely it is that a randomly chosen value will be below a certain threshold. It's a way to understand the *distribution shape*, especially to check for skewed or non-normal data.

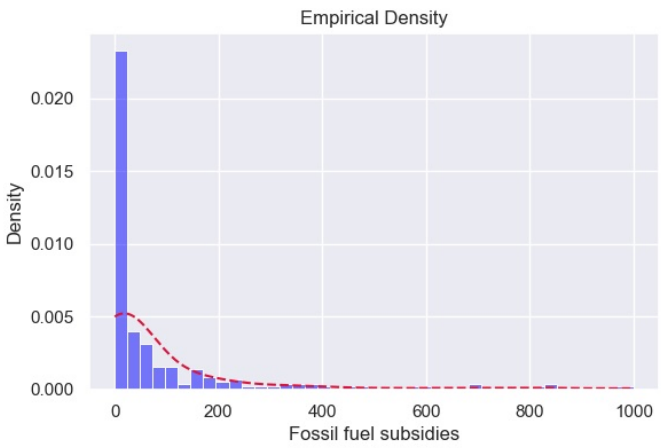
By using both KDE and ECDF we will get an intuitive sense of where your data clusters, its overall spread and how data points build up over time and to see proportions or percentiles

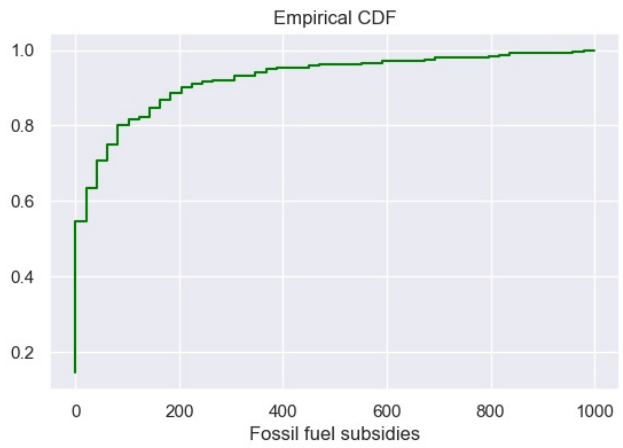
Cullen Frey Graphs: The final step is comparing our data to theoretical distributions (eg. normal, lognormal, exponential, etc.). The Cullen-Frey plots will help us assess the shape of our data distribution by plotting the skewness and kurtosis. It is essentially a diagnostic tool that provides a visual guide for determining whether your data fits common theoretical distributions.

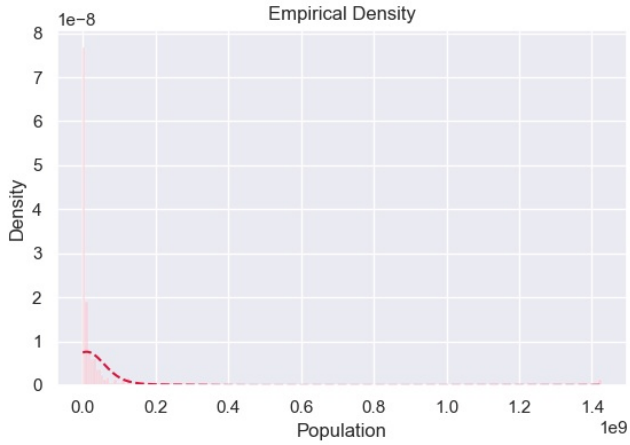
In [194…

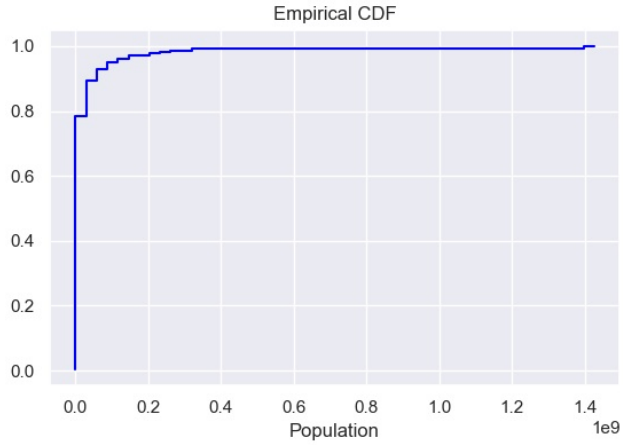
```
# Now fit the ECDF
fossil_fuel_subsidies_ecdf = sm.distributions.ECDF(raw_data_cleaned['fossil_fuel_subsidies'])
fossil_fuel_subsidies_range = np.linspace(min(raw_data_cleaned['fossil_fuel_subsidies']), max(raw_data_cleaned[
fossil_fuel_subsidies_cdf = fossil_fuel_subsidies_ecdf(fossil_fuel_subsidies_range)

population_ecdf = sm.distributions.ECDF(raw_data_cleaned['population'])
population_range = np.linspace(min(raw_data_cleaned['population']), max(raw_data_cleaned['population']))
population_cdf = population_ecdf(population_range)

per_capita_gdp_ecdf = sm.distributions.ECDF(raw_data_cleaned['per_capita_gdp'])
per_capita_gdp_range = np.linspace(min(raw_data_cleaned['per_capita_gdp']), max(raw_data_cleaned['per_capita_gdp
per_capita_gdp_cdf = per_capita_gdp_ecdf(per_capita_gdp_range)

Urbanization_ecdf = sm.distributions.ECDF(raw_data_cleaned['Urbanization'])
Urbanization_range = np.linspace(min(raw_data_cleaned['Urbanization']), max(raw_data_cleaned['Urbanization']))
Urbanization_cdf = Urbanization_ecdf(Urbanization_range)

renewable_energy_share_ecdf = sm.distributions.ECDF(raw_data_cleaned['renewable_energy_share'])
renewable_energy_share_range = np.linspace(min(raw_data_cleaned['renewable_energy_share']), max(raw_data_cleaned
renewable_energy_share_cdf = renewable_energy_share_ecdf(renewable_energy_share_range)

per_cap_energy_con_ecdf = sm.distributions.ECDF(raw_data_cleaned['per_cap_energy_con'])
per_cap_energy_con_range = np.linspace(min(raw_data_cleaned['per_cap_energy_con']), max(raw_data_cleaned['per_ca
per_cap_energy_con_cdf = per_cap_energy_con_ecdf(per_cap_energy_con_range)

gii_technology_ecdf = sm.distributions.ECDF(raw_data_cleaned['gii_technology'])
gii_technology_range = np.linspace(min(raw_data_cleaned['gii_technology']), max(raw_data_cleaned['gii_technology
gii_technology_cdf = gii_technology_ecdf(gii_technology_range)

industry_share_ecdf = sm.distributions.ECDF(raw_data_cleaned['industry_share'])
industry_share_range = np.linspace(min(raw_data_cleaned['industry_share']), max(raw_data_cleaned['industry_share
industry_share_cdf = industry_share_ecdf(industry_share_range)

fig, ax3 = plt.subplots(5, 2, figsize=(12, 20))
# fossil_fuel_subsidies
sns.histplot(raw_data_cleaned['fossil_fuel_subsidies'], stat='density', kde=True, ax=ax3[0, 0], color='blue')
ax3[0, 0].lines[0].set_color('crimson')
ax3[0, 0].lines[0].set_linestyle('--')
ax3[0, 0].set_xlabel('Fossil fuel subsidies')
ax3[0, 0].set_title('Empirical Density')
ax3[0, 1].step(fossil_fuel_subsidies_range, fossil_fuel_subsidies_cdf, color='green')
ax3[0, 1].set_title('Empirical CDF')
ax3[0, 1].set_xlabel('Fossil fuel subsidies')

# population
sns.histplot(raw_data_cleaned['population'], stat='density', kde=True, ax=ax3[1, 0], color='pink')
ax3[1, 0].lines[0].set_color('crimson')
ax3[1, 0].lines[0].set_linestyle('--')
ax3[1, 0].set_xlabel('Population')
ax3[1, 0].set_title('Empirical Density')
ax3[1, 1].step(population_range, population_cdf, color='blue')
ax3[1, 1].set_title('Empirical CDF')
ax3[1, 1].set_xlabel('Population')

# per_capita_gdp
sns.histplot(raw_data_cleaned['per_capita_gdp'], stat='density', kde=True, ax=ax3[2, 0], color='yellow')
ax3[2, 0].lines[0].set_color('crimson')
ax3[2, 0].lines[0].set_linestyle('--')
ax3[2, 0].set_xlabel('GDP per Capita')
ax3[2, 0].set_title('Empirical Density')
ax3[2, 1].step(per_capita_gdp_range, per_capita_gdp_cdf, color='red')
ax3[2, 1].set_title('Empirical CDF')
ax3[2, 1].set_xlabel('GDP per Capita')

# Urbanization
sns.histplot(raw_data_cleaned['Urbanization'], stat='density', kde=True, ax=ax3[3, 0], color='purple')
ax3[3, 0].lines[0].set_color('crimson')
ax3[3, 0].lines[0].set_linestyle('--')
ax3[3, 0].set_xlabel('Urbanization')
ax3[3, 0].set_title('Empirical Density')
ax3[3, 1].step(Urbanization_range, Urbanization_cdf, color='pink')
```

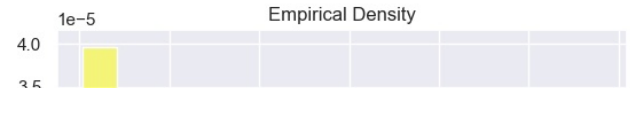
```
ax3[3, 1].set_title('Empirical CDF')
ax3[3, 1].set_xlabel('Urbanization')

# renewable_energy_share
sns.histplot(raw_data_cleaned['renewable_energy_share'], stat='density', kde=True, ax=ax3[4, 0], color='orange'
ax3[4, 0].lines[0].set_color('crimson')
ax3[4, 0].lines[0].set_linestyle('--')
ax3[4, 0].set_xlabel('Renewable Energy Share')
ax3[4, 0].set_title('Empirical Density')
ax3[4, 1].step(renewable_energy_share_range, renewable_energy_share_cdf, color='orange')
ax3[4, 1].set_title('Empirical CDF')
ax3[4, 1].set_xlabel('Renewable Energy Share')

# per_cap_energy_con
sns.histplot(raw_data_cleaned['per_cap_energy_con'], stat='density', kde=True, ax=ax3[4, 0], color='green')
ax3[4, 0].lines[0].set_color('crimson')
ax3[4, 0].lines[0].set_linestyle('--')
ax3[4, 0].set_xlabel('Energy per capita consumption')
ax3[4, 0].set_title('Empirical Density')
ax3[4, 1].step(per_cap_energy_con_range, per_cap_energy_con_cdf, color='orange')
ax3[4, 1].set_title('Empirical CDF')
ax3[4, 1].set_xlabel('Energy per capita consumption')

# gii_technology
sns.histplot(raw_data_cleaned['gii_technology'], stat='density', kde=True, ax=ax3[4, 0], color='blue')
ax3[4, 0].lines[0].set_color('crimson')
ax3[4, 0].lines[0].set_linestyle('--')
ax3[4, 0].set_xlabel('Technology')
ax3[4, 0].set_title('Empirical Density')
ax3[4, 1].step(gii_technology_range, gii_technology_cdf, color='orange')
ax3[4, 1].set_title('Empirical CDF')
ax3[4, 1].set_xlabel('Technology')

# industry_share
sns.histplot(raw_data_cleaned['industry_share'], stat='density', kde=True, ax=ax3[4, 0], color='red')
ax3[4, 0].lines[0].set_color('crimson')
ax3[4, 0].lines[0].set_linestyle('--')
ax3[4, 0].set_xlabel('Share of Industry in GDP')
ax3[4, 0].set_title('Empirical Density')
ax3[4, 1].step(industry_share_range, industry_share_cdf, color='orange')
ax3[4, 1].set_title('Empirical CDF')
ax3[4, 1].set_xlabel('Share of Industry in GDP')

plt.tight_layout()
plt.show()
```

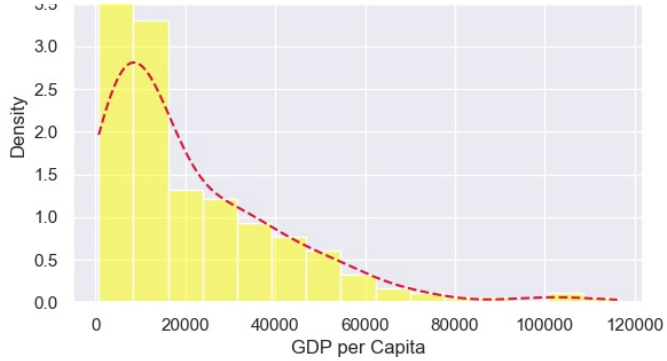


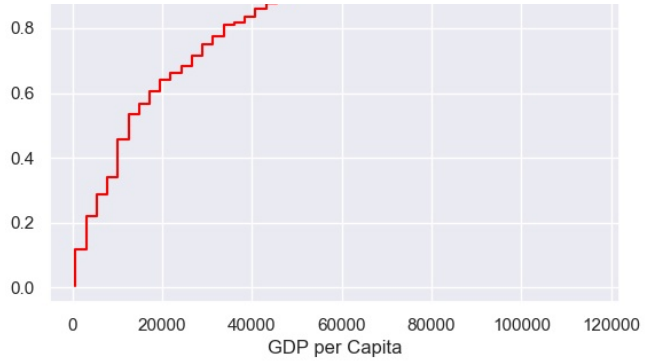


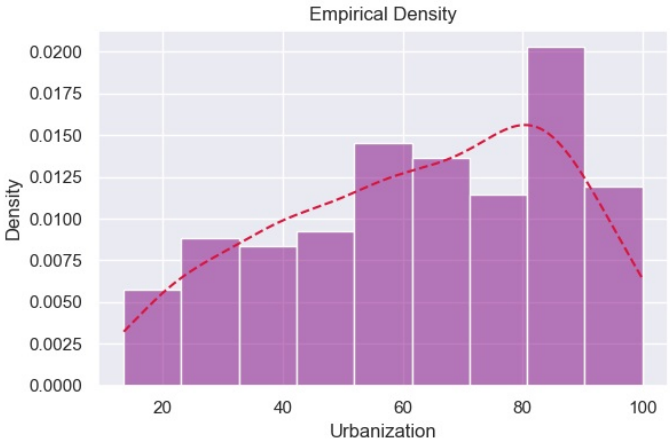


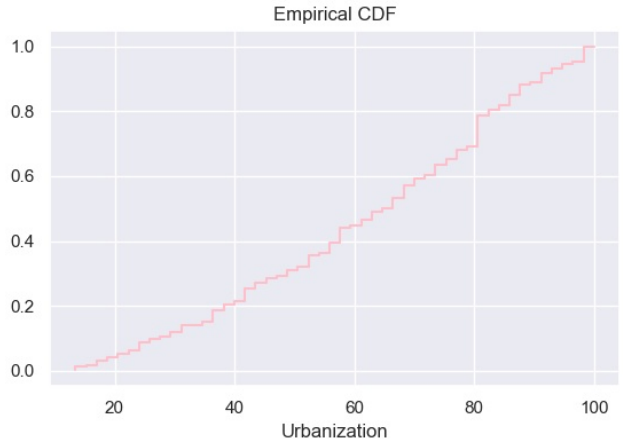


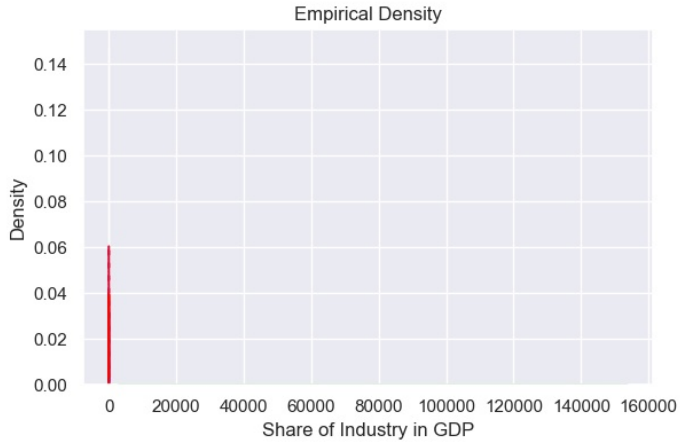


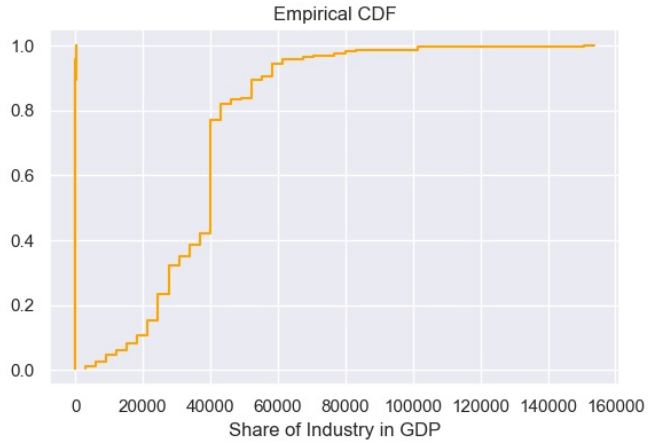








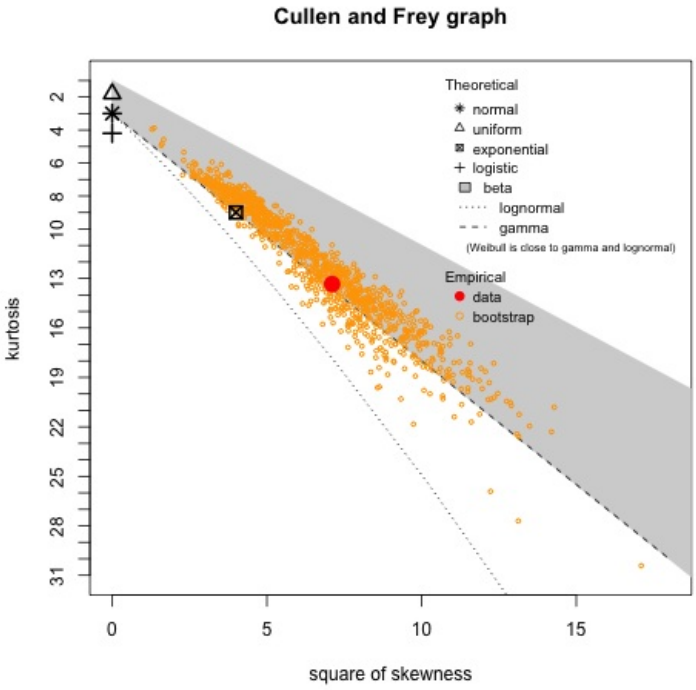




In [195…
```
%R -i raw_data_cleaned #runs as a data-frame in R
```

In [196…
```
%%R

set.seed(10) # for bootstrap
library("fitdistrplus")
# raw_data_cleaned <- read.csv('path/to/your/file.csv')

# Cullen-Frey Graph for per capita CO2 emissions
descdist(raw_data_cleaned$per_capita_co2_emissions, boot = 1000)
```

```
summary statistics
------
min:  0.037   max:  38.206
median:  3.617
mean:  4.61724
estimated sd:  5.191257
estimated skewness:  2.666186
estimated kurtosis:  13.3401
```

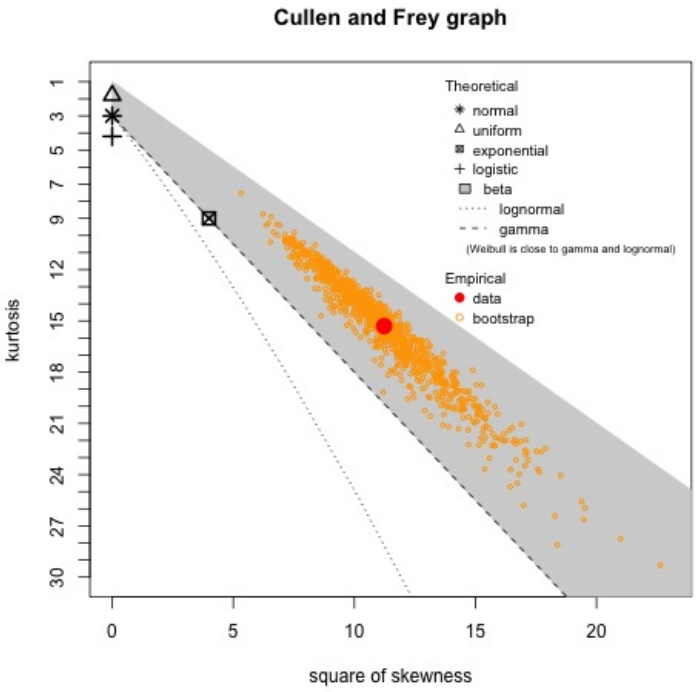


Gamma distribution fits the data best, as the empirical points (in red) are located near the gamma distribution region in the plot. The bootstrap samples (orange dots) also cluster around that region.

In [198…

```
%%R
set.seed(10) # for bootstrap
library("fitdistrplus")

# Cullen-Frey Graph
descdist(raw_data_cleaned$fossil_fuel_subsidies , boot = 1000)
```

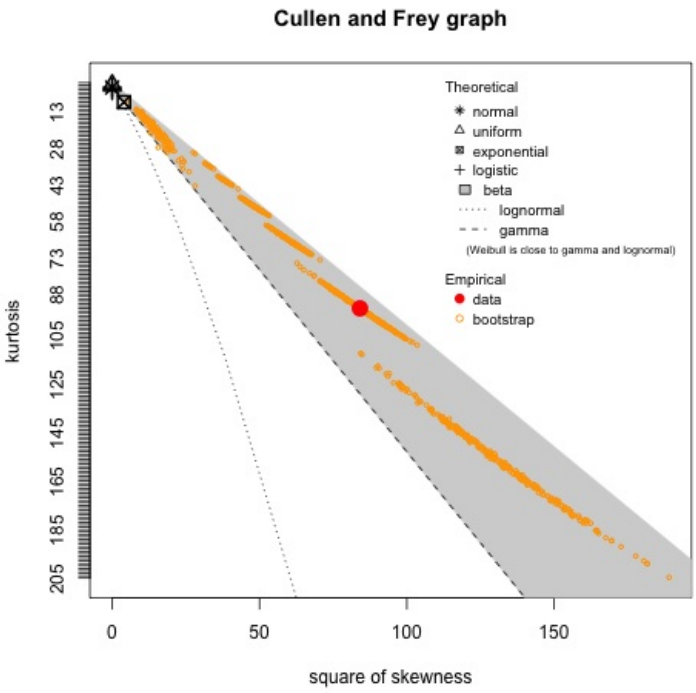
```
summary statistics
------
min:  0   max:  999.63
median:  13.25833
mean:  83.61146
estimated sd:  167.2694
estimated skewness:  3.351294
estimated kurtosis:  15.30647
```



The red point (empirical data) lies near the gamma distribution region, and farther from the normal, logistic, and beta distributions.

In [96]:
```
%%R
set.seed(10) # for bootstrap
library("fitdistrplus")

# Cullen-Frey Graph
descdist(raw_data_cleaned$population    , boot = 1000)
```

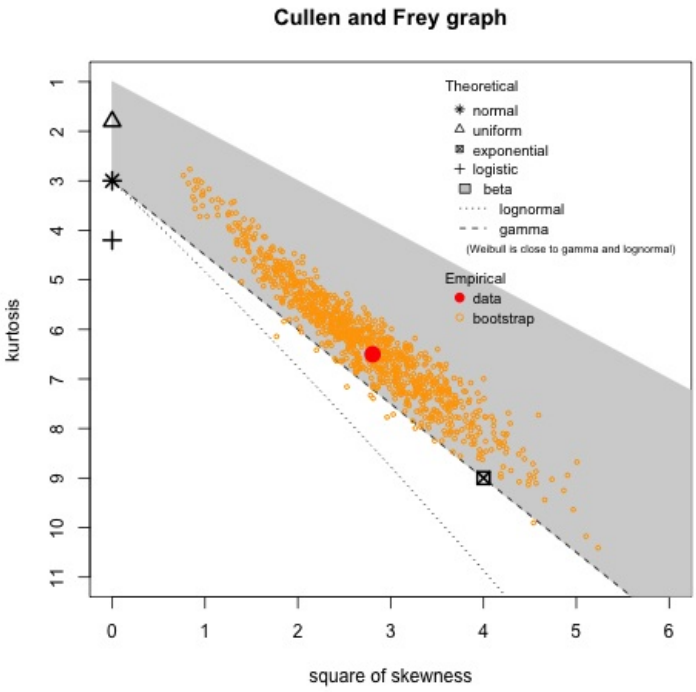
```
summary statistics
------
min:  513.667   max:  1425898899
median:  5469677
mean:  33991010
estimated sd:  136679649
estimated skewness:  9.169683
estimated kurtosis:  93.92903
```



The lognormal distribution would be a good fit here as the red empirical point is closer to the lognormal distribution boundary.

In [97]:
```
%%R
set.seed(10) # for bootstrap
library("fitdistrplus")

# Cullen-Frey Graph
descdist(raw_data_cleaned$per_capita_gdp, boot = 1000)
```

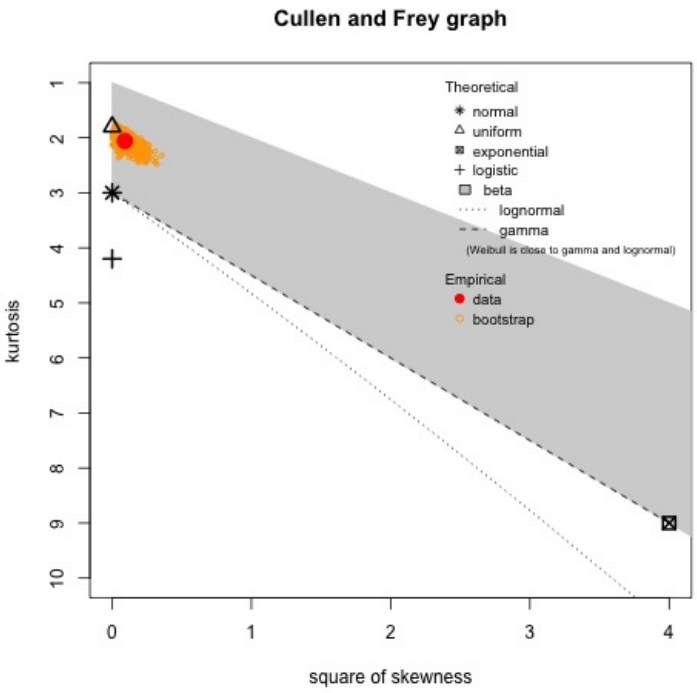
```
summary statistics
------
min:  711.2   max:  116177.3
median:  14226.32
mean:  21409.3
estimated sd:  20464.77
estimated skewness:  1.674912
estimated kurtosis:  6.501799
```



The gamma distribution seems to be the best fit for the data.

In [98]:
```
%%R
set.seed(10) # for bootstrap
library("fitdistrplus")

# Cullen-Frey Graph
descdist(raw_data_cleaned$Urbanization, boot = 1000)
```

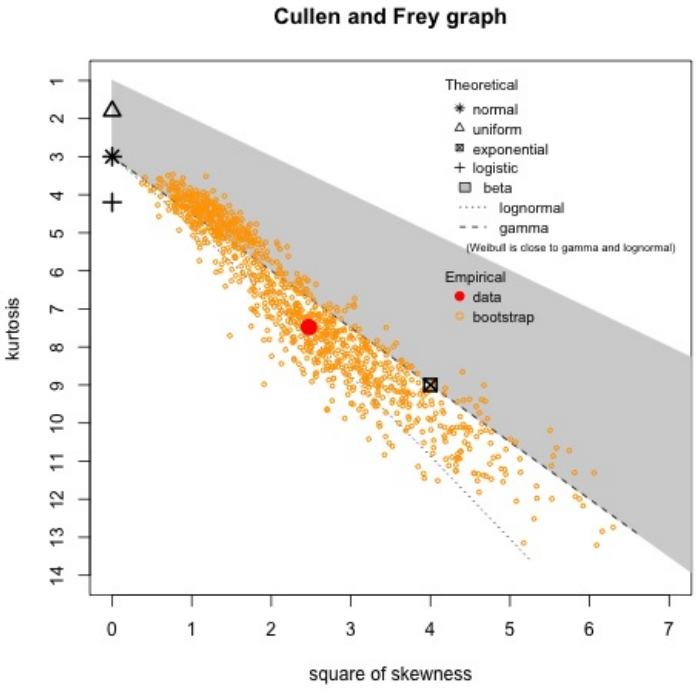
```
summary statistics
------
min:  13.461   max:  100
median:  66.53585
mean:  62.79189
estimated sd:  23.27669
estimated skewness:  -0.3029137
estimated kurtosis:  2.059665
```



The data point falls in the zone corresponding to the log-normal distribution. This suggests that the log-normal distribution is a good fit for the data.

In [99]:
```
%%R
set.seed(10) # for bootstrap
library("fitdistrplus")

# Cullen-Frey Graph
descdist(raw_data_cleaned$renewable_energy_share, boot = 1000)
```

```
summary statistics
------
min:  0.2424   max:  71.72
median:  13.6056
mean:  18.17364
estimated sd:  9.755889
estimated skewness:  1.572381
estimated kurtosis:  7.477789
```

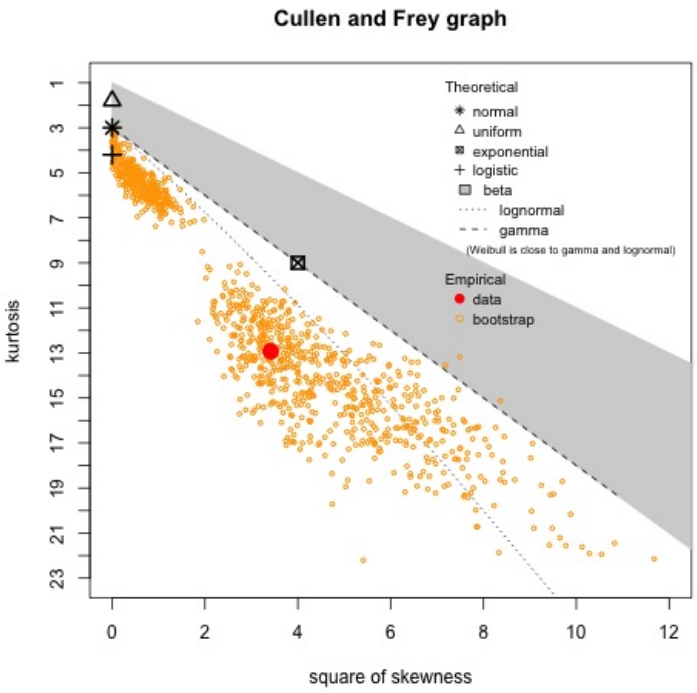


The best fit for the data here is gamma distribution. The data points cluster more closely to the gamma distribution line than the other distributions.

In [100…

```
%%R
set.seed(10) # for bootstrap
library("fitdistrplus")

# Cullen-Frey Graph
descdist(raw_data_cleaned$per_cap_energy_con, boot = 1000)
```

```
summary statistics
------
min:  2957.727   max:  153713.6
median:  40585.63
mean:  38566.49
estimated sd:  16911.48
estimated skewness:  1.847737
estimated kurtosis:  12.92736
```

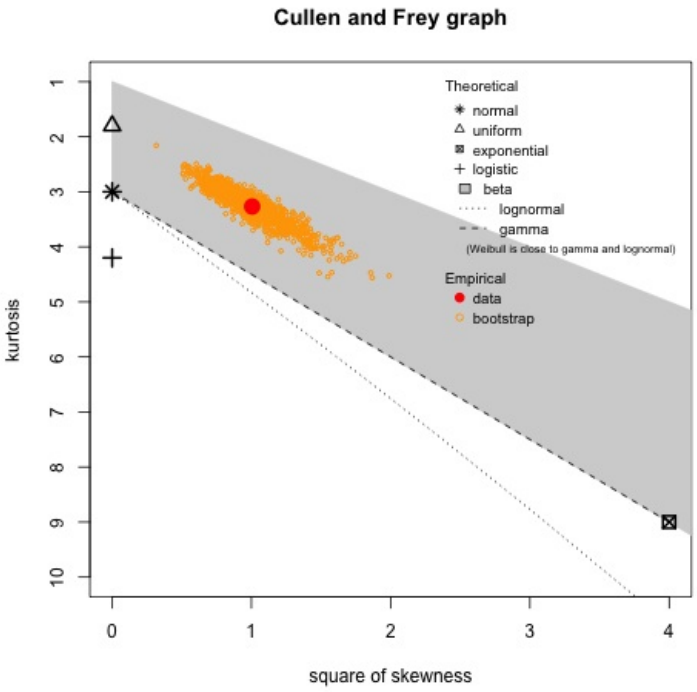


According to the location of the bootstrap data points, the distribution most likely to fit the data is the lognormal distribution.

In [101…

```
%%R
set.seed(10) # for bootstrap
library("fitdistrplus")

# Cullen-Frey Graph
descdist(raw_data_cleaned$gii_technology, boot = 1000)
```

```
summary statistics
------
min:  0   max:  65.5
median:  11.9
mean:  14.84719
estimated sd:  16.03588
estimated skewness:  1.002682
estimated kurtosis:  3.268022
```

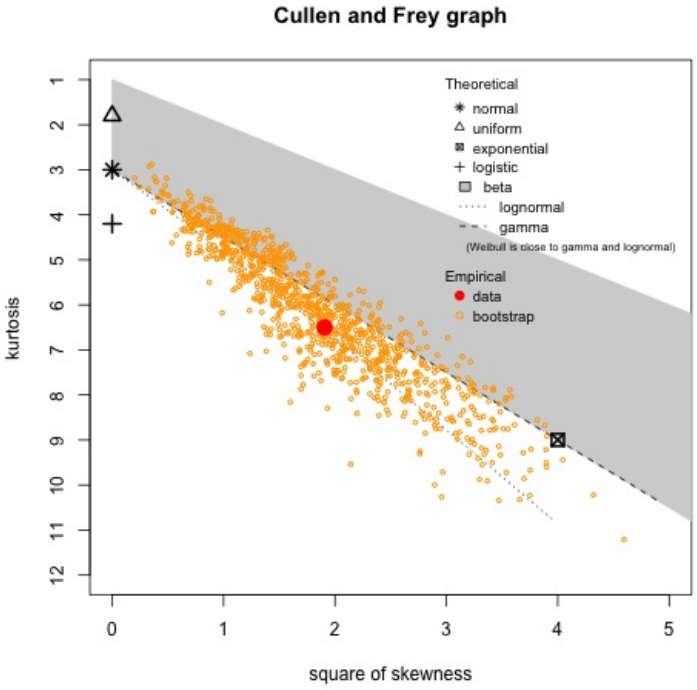


The data here is closest to the gamma and lognormal distribution as the point in the graph lies in the area between those two distributions.

In [102…

```
%%R
set.seed(10) # for bootstrap
library("fitdistrplus")

# Cullen-Frey Graph
descdist(raw_data_cleaned$industry_share, boot = 1000)

# Add which dist fits best
```

```
summary statistics
------
min:  2.4   max:  85
median:  23.75
mean:  25.12105
estimated sd:  11.96803
estimated skewness:  1.380992
estimated kurtosis:  6.498151
```



The data resembles a gamma distribution.

Part D) Identifying Non Linearities. What transformations should you perform? What would happen if you included non-linear variable without transformation

To identify and correct non-linearities in our variables, we applied Box-Cox transformations. The Box-Cox transformation is particularly useful for handling skewed or non-normally distributed variables by identifying the optimal power transformation parameter (λ) to stabilize variance and improve linearity. First, we tested various continuous variables, such as fossil fuel subsidies and per capita GDP, to detect non-linear patterns by examining residual plots for curvature. Once non-linearities were identified, we applied the Box-Cox transformation to those variables, transforming them closer to normality, which improved their fit in the linear regression model. This transformation is crucial because using non-linear variables in regression without adjusting them can lead to misleading results, biased estimates, and inefficiencies. The transformation helped meet the assumptions of ordinary least squares (OLS) regression, ensuring the model's validity. Finally, we reassessed the model to confirm improvements in the linear relationship between the dependent and independent variables.

In [200…

```
# List of variables to transform
variables_to_transform = ['per_capita_co2_emissions','fossil_fuel_subsidies', 'population', 'per_cap_energy_con

# Function to perform Box-Cox transformation and format the output
def boxcox_with_stats(df, variable):
    print(f"===== Results for {variable} =====")

    # Ensure positive values (add 1 if necessary)
    positive_values = df[variable] + 1

    # Drop missing values
    positive_values_no_na = positive_values.dropna()

    # Apply Box-Cox transformation and get the lambda and confidence interval
    transformed_data, lambda_value, confidence_interval = stats.boxcox(positive_values_no_na, alpha=0.05)

    # Print the lambda and confidence interval
    print(f"estimated power = {lambda_value:.4}")
    print(f"rounded power = {np.round(lambda_value, 1)}")
    print(f"Confidence interval = {np.round(confidence_interval, 4)}\n")

    # Log-likelihood computations for different lambdas
    L_opt_ll = stats.boxcox_llf(lmb=lambda_value, data=positive_values_no_na)  # optimal lambda
    L0_ll = stats.boxcox_llf(lmb=0, data=positive_values_no_na)  # lambda = 0 (log transformation)
```

```
    L05_ll = stats.boxcox_llf(lmb=0.5, data=positive_values_no_na)  # lambda = 0.5 (square root)
    L1_ll = stats.boxcox_llf(lmb=1, data=positive_values_no_na)  # lambda = 1 (no transformation)

    # Likelihood-ratio test statistics
    L0_stat = 2 * (L_opt_ll - L0_ll)
    L05_stat = 2 * (L_opt_ll - L05_ll)
    L1_stat = 2 * (L_opt_ll - L1_ll)

    # Compute p-values
    L0_p = 1 - stats.chi2.cdf(L0_stat, df=1)
    L05_p = 1 - stats.chi2.cdf(L05_stat, df=1)
    L1_p = 1 - stats.chi2.cdf(L1_stat, df=1)

    # Print the formatted output for the likelihood-ratio test
    print("Lambda \t Test Stat \t p-value")
    print(f"0 \t {L0_stat:.6} \t {L0_p:.4}")
    print(f"0.5 \t {L05_stat:.6} \t {L05_p:.4}")
    print(f"1 \t {L1_stat:.6} \t {L1_p:.4}\n")

# Loop through each variable and print the results
for var in variables_to_transform:
    boxcox_with_stats(raw_data_cleaned, var)
```

```
===== Results for per_capita_co2_emissions =====
estimated power = -0.08564
rounded power = -0.1
Confidence interval = [-0.2409  0.0654]

Lambda   Test Stat       p-value
0        1.2215          0.2691
0.5      61.5879         4.219e-15
1        220.741         0.0

===== Results for fossil_fuel_subsidies =====
estimated power = -0.008626
rounded power = -0.0
Confidence interval = [-0.0776  0.0593]

Lambda   Test Stat       p-value
0        0.0612159       0.8046
0.5      215.851         0.0
1        755.526         0.0

===== Results for population =====
estimated power = 0.0925
rounded power = 0.1
Confidence interval = [0.0508 0.1345]

Lambda   Test Stat       p-value
0        19.025          1.29e-05
0.5      333.599         0.0
1        1361.49         0.0

===== Results for per_cap_energy_con =====
estimated power = 0.5355
rounded power = 0.5
Confidence interval = [0.3786 0.6953]

Lambda   Test Stat       p-value
0        46.9433         7.307e-12
0.5      0.193833        0.6597
1        31.4185         2.08e-08

===== Results for per_capita_gdp =====
estimated power = 0.1702
rounded power = 0.2
Confidence interval = [0.0571 0.2848]

Lambda   Test Stat       p-value
0        8.75951         0.00308
0.5      30.7969         2.865e-08
1        178.936         0.0

===== Results for renewable_energy_share =====
estimated power = 0.4267
rounded power = 0.4
Confidence interval = [0.2837 0.5768]

Lambda   Test Stat       p-value
0        37.3844         9.7e-10
0.5      0.938734        0.3326
1        49.4556         2.029e-12

===== Results for industry_share =====
```

```
estimated power = 0.2769
rounded power = 0.3
Confidence interval = [0.0828 0.4757]

Lambda   Test Stat       p-value
0        7.90754         0.004923
0.5      4.82613         0.02803
1        48.0367         4.183e-12

===== Results for gii_technology =====
estimated power = 0.1552
rounded power = 0.2
Confidence interval = [0.0421 0.2671]

Lambda   Test Stat       p-value
0        7.20348         0.007276
0.5      36.7044         1.375e-09
1        212.263         0.0

===== Results for Urbanization =====
estimated power = 1.165
rounded power = 1.2
Confidence interval = [0.8534 1.4911]

Lambda   Test Stat       p-value
0        60.1345         8.882e-15
0.5      18.351          1.837e-05
1        1.05693         0.3039
```

After examining the joint plots and conducting Box-Cox transformations, we identified non-linear relationships in several variables, including *per_capita_co2_emissions, fossil_fuel_subsidies, population, per_cap_energy_con, per_capita_gdp, renewable_energy_share, industry_share, gii_technology, and Urbanization*. Based on the Box-Cox test results, we plan to apply specific transformations to these variables to reduce non-linearity and improve model accuracy:

1. *per_capita_co2_emissions: The estimated lambda value is approximately* -0.1**. Although this indicates a log transformation, the confidence interval includes zero, suggesting that minor adjustments might stabilize this variable for linearity without significant distortion.
2. fossil_fuel_subsidies: The estimated lambda rounds to **0*, so a log transformation is appropriate for handling its non-linearity.
3. population: With a lambda of **0.1*, this variable will be transformed with a mild power transformation to improve its linearity.
4. per_cap_energy_con: A lambda of **0.5* indicates a square-root transformation, which we'll apply to reduce non-linear effects.
5. per_capita_gdp: This variable has a lambda of **0.2*, suggesting a moderate transformation to improve model fit while preserving interpretability.
6. renewable_energy_share: A rounded lambda of **0.4* suggests a root transformation close to the fourth root, which is necessary to improve the linearity.
7. industry_share: With a lambda of **0.3*, we'll use a cubic root transformation to stabilize this variable's contribution to the model.
8. gii_technology: A lambda of **0.2* supports a mild transformation, such as a fifth root, to better align this variable with a linear model.
9. Urbanization: This variable has a higher lambda of **1.2*, close to linearity, so no transformation may be necessary, although a slight power transformation could further improve stability.

By applying these transformations, we aim to reduce non-linearity and better approximate a linear relationship between the predictors and the dependent variable, *per_capita_co2_emissions*. If we included these variables without addressing their non-linearities, the model could become misspecified, leading to biased estimates of coefficients and potentially inaccurate interpretations of the relationships between variables.

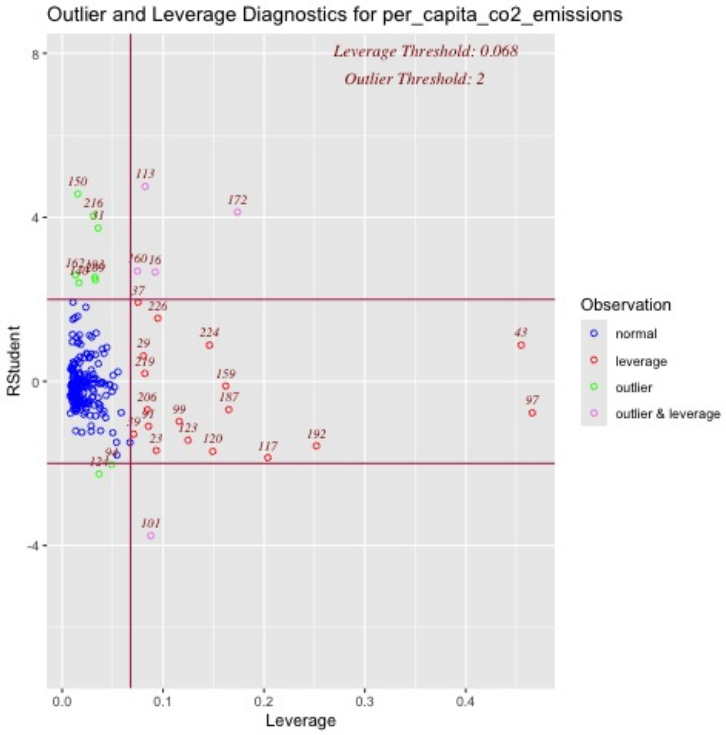
Part E)- Comment On any outliers and/or unusual features of your variable

To examine outliers and unusual features in our regression model, we used several diagnostic tools. First, we generated an Outlier and Leverage Diagnostic plot, which highlights data points with high leverage (those far from the center of the predictor space) and large residuals, identifying observations that could have a significant impact on the regression estimates. This plot shows us whether the data points are potentially influencing the slope of the regression line. We then used the Cook's Distance plot to further assess the influence of individual data points, flagging those that might overly influence the model's predictions by combining both leverage and residual information. Finally, we analyzed Residual plots, which allowed us to examine the distribution of residuals across the fitted values, helping detect any non-random patterns (such as funnel-shaped residuals) that might indicate model misspecification, outliers, or non-linearity in the data. Together, these diagnostics helped us pinpoint specific observations that were either highly influential or showed large deviations from the model's predicted values, guiding further investigation and potential adjustments to the model.

Outlier and Leverage Diagnostics

In [201…

```
%%R
# Load necessary libraries

library(olsrr)

# Fit the linear regression model with the dataset passed from Python to R
mreg.mod = lm(per_capita_co2_emissions ~ fossil_fuel_subsidies + population + per_cap_energy_con + per_capita_g

# Plot the residuals vs leverage
ols_plot_resid_lev(mreg.mod)
```

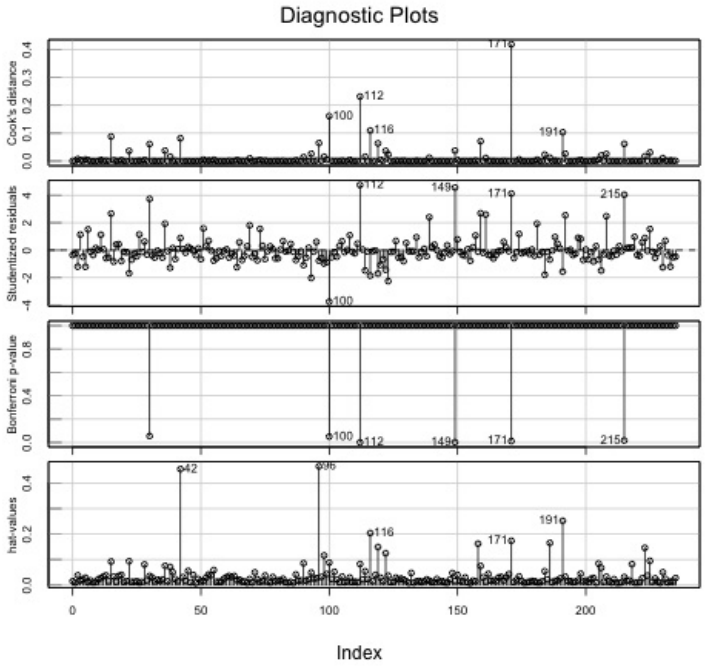


The leverage vs. studentized residuals plot identifies influential observations. Points with high leverage are far from the center of the predictor variable space, and those with high studentized residuals indicate a poor fit for the regression model. Points like 172, 43, and 97 exhibit both high leverage and large studentized residuals. These observations should be carefully inspected as they might unduly influence the regression results. Additionally, points like 101 and 117 have high leverage, which means they significantly affect the regression line but without being extreme in their residuals. Outliers such as 150, 216, and 81 also have large residuals, indicating that the model doesn't fit well for these observations.

Cook's Distance Plot

In [202…

```
%%R

# Load necessary libraries
library(car)
model = lm(per_capita_co2_emissions ~ fossil_fuel_subsidies + population + per_cap_energy_con + per_capita_gdp +
influenceIndexPlot(model, id=list(n=5))
```



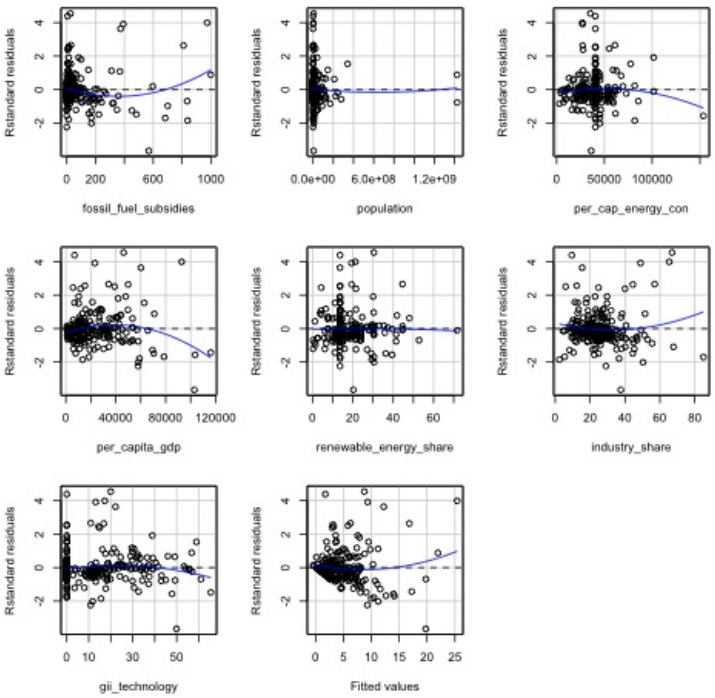
This plot measures the influence of each observation on the overall regression coefficients. Observations with high Cook's distance have a disproportionate effect on the model. Points like 100, 116, 171, and 215 have the highest Cook's distance values, indicating they are highly influential in determining the regression results. These points should be further investigated to determine if they represent valid observations or are outliers that could be distorting the model.

Residuals Plot

In [203…

```
%%R
library(car)
model = lm(per_capita_co2_emissions ~ fossil_fuel_subsidies + population + per_cap_energy_con + per_capita_gdp +
residualPlots(model,type = "rstandard")
```

```
                        Test stat Pr(>|Test stat|)
fossil_fuel_subsidies     2.8354        0.0049905 **
population                0.3822        0.7026454
per_cap_energy_con       -1.3265        0.1860019
per_capita_gdp           -3.3865        0.0008343 ***
renewable_energy_share   -0.1707        0.8645945
industry_share            1.7820        0.0760934 .
gii_technology           -1.8161        0.0706679 .
Tukey test                1.7046        0.0882602 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```



This plot helps check the assumption that the residuals (errors) are randomly distributed. A horizontal line with no distinct pattern suggests that the linear regression model is appropriate. Some of the predictors like fossil_fuel_subsidies, per_capita_gdp, and Urbanization show non-linear patterns in residuals, indicating potential non-linearities in the relationship between these variables and the dependent variable (per_capita_co2_emissions). In particular, per_capita_gdp seems to have the most pronounced non-linear pattern, suggesting the need for transformation or a different model specification.

Part F)- Imputing NA's Using KNN Method

Used KNN method with three neighbours to impute NA values. This was based on taking all x variables into account as well as using a dummy variable for continents to find the closest predictor.

See above for Code

Question 2

Part A) Boruta Algorithm

In [204…

```
### Boruta using BorutaPy

raw_data_cleaned_2 = raw_data_cleaned.copy()

# Prepare data
raw_data_cleaned_y = raw_data_cleaned_2['per_capita_co2_emissions'].values
raw_data_cleaned_x = raw_data_cleaned_2.drop(['per_capita_co2_emissions'] , axis=1).values.astype(np.float64)

#Boruta algorithm
rf = RandomForestRegressor(max_depth=5)
boruta = BorutaPy(rf, n_estimators='auto', verbose=2, random_state=10, two_step=True) # two_step uses Bonferron

# Run algorithm, X and y must by numpy arrays
boruta.fit(raw_data_cleaned_x, raw_data_cleaned_y)
```

```
Iteration:      1 / 100
Confirmed:      0
Tentative:      8
Rejected:       0
Iteration:      2 / 100
Confirmed:      0
Tentative:      8
Rejected:       0
Iteration:      3 / 100
Confirmed:      0
Tentative:      8
Rejected:       0
Iteration:      4 / 100
Confirmed:      0
Tentative:      8
Rejected:       0
Iteration:      5 / 100
Confirmed:      0
Tentative:      8
Rejected:       0
Iteration:      6 / 100
Confirmed:      0
Tentative:      8
Rejected:       0
Iteration:      7 / 100
Confirmed:      0
Tentative:      8
Rejected:       0
Iteration:      8 / 100
Confirmed:      4
Tentative:      2
Rejected:       2
Iteration:      9 / 100
Confirmed:      4
Tentative:      2
Rejected:       2
Iteration:      10 / 100
Confirmed:      4
Tentative:      2
Rejected:       2
Iteration:      11 / 100
Confirmed:      4
Tentative:      2
Rejected:       2
Iteration:      12 / 100
Confirmed:      4
Tentative:      1
```

```
Rejected:       3
Iteration:      13 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      14 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      15 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      16 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      17 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      18 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      19 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      20 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      21 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      22 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      23 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      24 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      25 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      26 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      27 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      28 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      29 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      30 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      31 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      32 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      33 / 100
Confirmed:      4
```

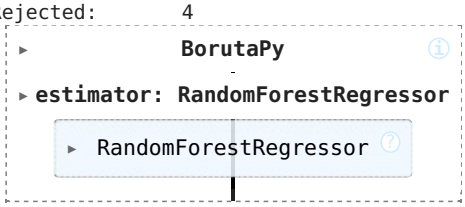
```
Tentative:      1
Rejected:       3
Iteration:      34 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      35 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      36 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      37 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      38 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      39 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      40 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      41 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      42 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      43 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      44 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      45 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      46 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      47 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      48 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      49 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      50 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      51 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      52 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      53 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      54 / 100
```

```
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      55 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      56 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      57 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      58 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      59 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      60 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      61 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      62 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      63 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      64 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      65 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      66 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      67 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      68 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      69 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      70 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      71 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      72 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      73 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      74 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
```

```
Iteration:      75 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      76 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      77 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      78 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      79 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      80 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      81 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      82 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      83 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      84 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      85 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      86 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      87 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      88 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      89 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      90 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      91 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      92 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      93 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      94 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      95 / 100
Confirmed:      4
Tentative:      1
```

```
Rejected:       3
Iteration:      96 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      97 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      98 / 100
Confirmed:      4
Tentative:      1
Rejected:       3
Iteration:      99 / 100
Confirmed:      4
Tentative:      1
Rejected:       3


BorutaPy finished running.

Iteration:      100 / 100
Confirmed:      4
Tentative:      0
Rejected:       4
```

Out[204…



In [205…

```
# Compile feature decisions
boruta_keep = ['Confirmed' if c else
               'Tentative' if t else
               'Reject' for c,t in zip(boruta.support_, boruta.support_weak_)]

# Combine results and print
boruta_features = zip(raw_data_cleaned_2.drop('per_capita_co2_emissions', axis=1).columns, boruta.ranking_, bor
for feat in boruta_features:
    print('Feature: {:<20} Rank: {},  Keep: {}'.format(feat[0], feat[1], feat[2]))
```

```
Feature: fossil_fuel_subsidies Rank: 1,  Keep: Confirmed
Feature: population            Rank: 2,  Keep: Reject
Feature: per_capita_gdp        Rank: 1,  Keep: Confirmed
Feature: Urbanization          Rank: 4,  Keep: Reject
Feature: renewable_energy_share Rank: 3,  Keep: Reject
Feature: per_cap_energy_con    Rank: 1,  Keep: Confirmed
Feature: gii_technology        Rank: 5,  Keep: Reject
Feature: industry_share        Rank: 1,  Keep: Confirmed
```

In [206…

```
### Boruta using BorutaShap
## Note: takes about 10 minutes


raw_data_cleaned2_X2 = raw_data_cleaned_2.drop('per_capita_co2_emissions', axis=1)

# Run Boruta algorithm
Feature_Selector = BorutaShap(importance_measure='shap', classification=False)
Feature_Selector.fit(X=raw_data_cleaned2_X2, y= raw_data_cleaned_y, n_trials=100, random_state=90095)

# Plot results
Feature_Selector.plot(which_features='all')
plt.show()

# Print results
boruta_keep2 = ['Confirmed' if p in Feature_Selector.accepted else
                'Tentative' if p in Feature_Selector.tentative else
                'Reject' for p in Feature_Selector.all_columns]

boruta_features2 = zip(Feature_Selector.all_columns,
                       len(Feature_Selector.shap_values) - np.argsort(Feature_Selector.shap_values),
                       Feature_Selector.shap_values,
                       boruta_keep2)
for feat in boruta_features2:
    print('Feature: {:<20} Rank: {} ({:.3f}),  Keep: {}'.format(feat[0], feat[1], feat[2], feat[3]))
```

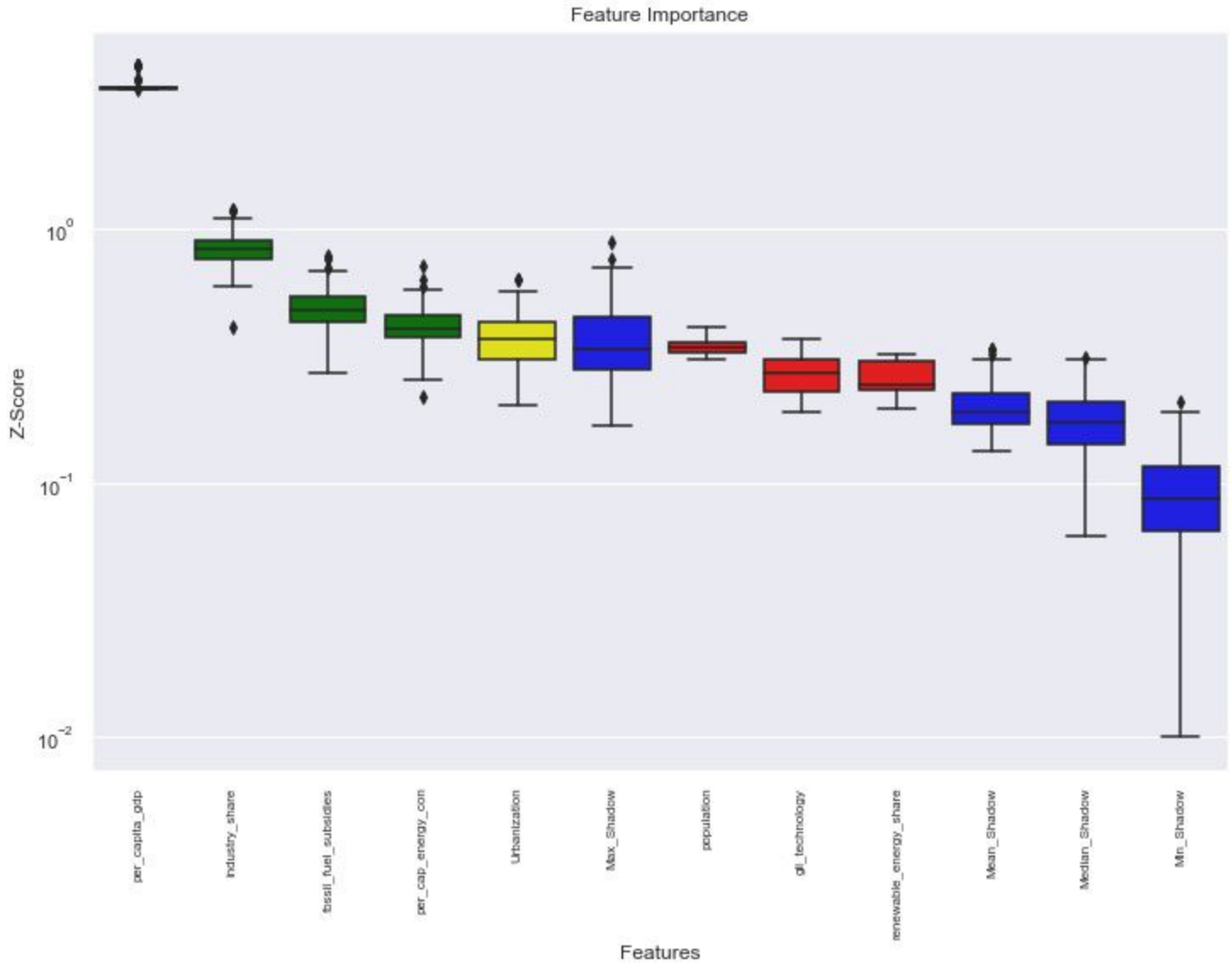
```
  0%|          | 0/100 [00:00<?, ?it/s]
```

In [15]:

```
### Boruta using BorutaShap
## Note: takes about 10 minutes


raw_data_cleaned2_X2 = raw_data_cleaned_2.drop('per_capita_co2_emissions', axis=1)

# Run Boruta algorithm
Feature_Selector = BorutaShap(importance_measure='shap', classification=False)
Feature_Selector.fit(X=raw_data_cleaned2_X2, y= raw_data_cleaned_y, n_trials=100, random_state=90095)

# Plot results
Feature_Selector.plot(which_features='all')
plt.show()

# Print results
boruta_keep2 = ['Confirmed' if p in Feature_Selector.accepted else
                'Tentative' if p in Feature_Selector.tentative else
                'Reject' for p in Feature_Selector.all_columns]

boruta_features2 = zip(Feature_Selector.all_columns,
                       len(Feature_Selector.shap_values) - np.argsort(Feature_Selector.shap_values),
                       Feature_Selector.shap_values,
                       boruta_keep2)
for feat in boruta_features2:
    print('Feature: {:<20} Rank: {} ({:.3f}),  Keep: {}'.format(feat[0], feat[1], feat[2], feat[3]))
```

```
100% |████████████████████████████████████████| 100/100 [01:52<00:00,  1.10it/s]

4 attributes confirmed important: ['per_cap_energy_con', 'industry_share', 'fossil_fuel_subsidies', 'per_capita_gdp']
3 attributes confirmed unimportant: ['population', 'gii_technology', 'renewable_energy_share']
1 tentative attributes remains: ['Urbanization']
```

```
4 attributes confirmed important: ['per_cap_energy_con', 'industry_share', 'fossil_fuel_subsidies', 'per_capita_gdp']
3 attributes confirmed unimportant: ['population', 'gii_technology', 'renewable_energy_share']
1 tentative attributes remains: ['Urbanization']
```




```
Feature: fossil_fuel_subsidies Rank: 3 (0.302),  Keep: Confirmed
Feature: population            Rank: 5 (2.211),  Keep: Reject
Feature: per_capita_gdp        Rank: 4 (0.275),  Keep: Confirmed
Feature: Urbanization          Rank: 1 (0.251),  Keep: Tentative
Feature: renewable_energy_share Rank: 7 (0.445),  Keep: Reject
Feature: per_cap_energy_con    Rank: 8 (0.074),  Keep: Confirmed
Feature: gii_technology        Rank: 10 (0.100),  Keep: Reject
Feature: industry_share        Rank: 2 (0.071),  Keep: Confirmed
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[206], line 9
      7 # Run Boruta algorithm

      8 Feature_Selector = BorutaShap(importance_measure='shap', classification=False)
----> 9 Feature_Selector.fit(X=raw_data_cleaned2_X2, y= raw_data_cleaned_y, n_trials=100, random_state=90095)
     11 # Plot results
     12 Feature_Selector.plot(which_features='all')

File /opt/anaconda3/lib/python3.12/site-packages/BorutaShap.py:473, in BorutaShap.fit(self, X, y, sample_weight,
n_trials, random_state, sample, train_or_test, normalize, verbose, stratify)
    471         self.hits += hits
    472         self.history_hits = np.vstack((self.history_hits, self.hits))
--> 473         self.test_features(iteration=trial+1)
    475 self.store_feature_importance()
    476 self.calculate_rejected_accepted_tentative(verbose=verbose)

File /opt/anaconda3/lib/python3.12/site-packages/BorutaShap.py:934, in BorutaShap.test_features(self, iteration)
    919 def test_features(self, iteration):
    921     """
    922     For each feature with an undetermined importance perform a two-sided test of equality
    923     with the maximum shadow value to determine if it is statistcally better
   (...)
    931         Two arrays of the names of the accepted and rejected columns at that instance
    932     """
--> 934     acceptance_p_values = self.binomial_H0_test(self.hits,
    935                                                 n=iteration,
    936                                                 p=0.5,
    937                                                 alternative='greater')
    939     regect_p_values = self.binomial_H0_test(self.hits,
    940                                             n=iteration,
    941                                             p=0.5,
    942                                             alternative='less')
    944     # [1] as function returns a tuple

File /opt/anaconda3/lib/python3.12/site-packages/BorutaShap.py:885, in BorutaShap.binomial_H0_test(array, n, p,
alternative)
    878 @staticmethod
    879 def binomial_H0_test(array, n, p, alternative):
    880     """
    881     Perform a test that the probability of success is p.
    882     This is an exact, two-sided test of the null hypothesis
    883     that the probability of success in a Bernoulli experiment is p
    884     """
--> 885     return [binom_test(x, n=n, p=p, alternative=alternative) for x in array]

NameError: name 'binom_test' is not defined
```

Based on the Boruta algorithm, we can see in terms of importance, the top 2 predictors in orders of most to least important are:

Per Capita GDP Share of industry as a % of GDP Other important predictors are:

Fossil Fuel Subsidies Per Capita Per Capita Energy Consumption Per Capita Urbanization (Tentative)

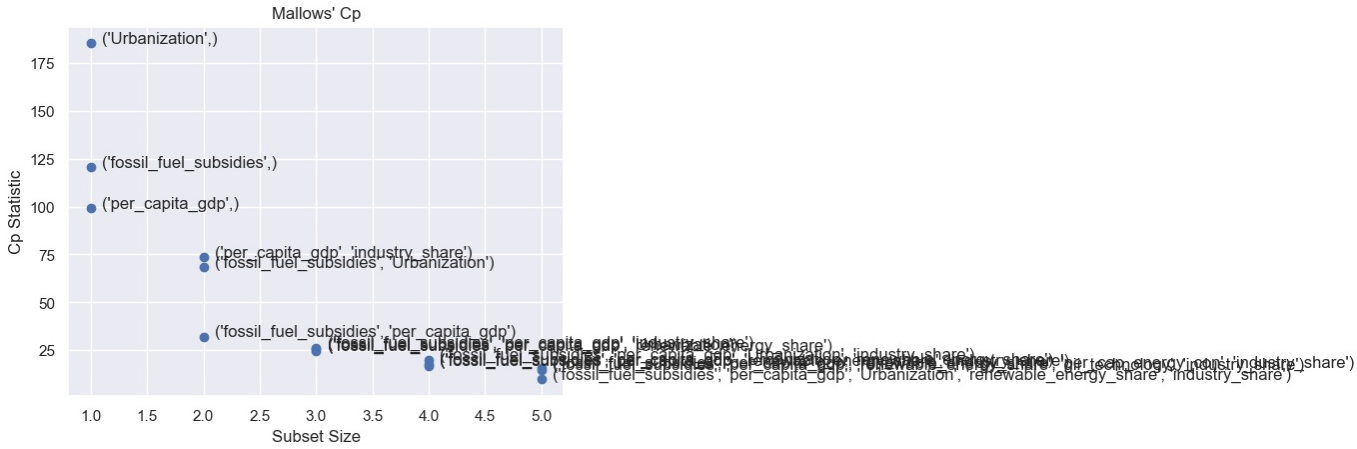
Part B) Mallow's CP

In [207…

```
# Mallow's CP


#full regression
energy_reg = smf.ols('per_capita_co2_emissions ~ fossil_fuel_subsidies + population + per_capita_gdp + Urbaniza
energ_reg_sig2 = energy_reg.mse_resid
energy_n, _ = raw_data_cleaned.shape
energy_reg.summary()


# Negative Mallows' cp, so we want largest
def negative_mallows(estimator, X, y):
    y_pred = estimator.predict(X)
    sse = np.sum((y - y_pred) ** 2)
    p = estimator.n_features_in_ + 1 # number features doesn't include bias term
    cp = sse / energ_reg_sig2 + 2*p - energy_n
    return -cp

# Set up regression
energy_reg_Y = raw_data_cleaned['per_capita_co2_emissions'].values
energy_reg_X = raw_data_cleaned.drop('per_capita_co2_emissions', axis=1)
lr = LinearRegression(fit_intercept=True)

# Using ExhaustiveFeatureSelector from mlxtend
energy_efs = EFS(lr,
```

```
                min_features=1,
                max_features=5,
                scoring=negative_mallows,
                cv=None)

# Run regressions
energy_efs.fit(raw_data_cleaned2_X2, raw_data_cleaned_y)
print('Best subset:', [energy_efs.feature_names[i] for i in energy_efs.best_idx_])

# Extract Mallows' Cp data
energy_efs_features = [energy_efs.subsets_[p]['feature_names'] for p in energy_efs.subsets_]
energy_efs_subset = [len(energy_efs.subsets_[p]['feature_idx']) for p in energy_efs.subsets_]
energy_efs_cp = [ -energy_efs.subsets_[p]['avg_score'] for p in energy_efs.subsets_]
energy_mallows = pd.DataFrame({'subsets':energy_efs_subset, 'Cp':energy_efs_cp}, index=energy_efs_features)

# Top 3 best, by subset
energy_mallows_best = energy_mallows.groupby('subsets')[['subsets','Cp']].apply(lambda x: x.nsmallest(3, 'Cp'))
energy_mallows_best

# Plotting best subsets
plt.scatter(energy_mallows_best['subsets'], energy_mallows_best['Cp'])
plt.title("Mallows' Cp")
plt.xlabel('Subset Size')
plt.ylabel('Cp Statistic')
for i,sub in enumerate(energy_mallows_best['subsets']):
    subset = energy_mallows_best.index[i][1]
    plt.annotate(subset, (sub + 0.1, energy_mallows_best.iloc[i]['Cp'] + 0.3)) # to plot labels
plt.tight_layout()
plt.show()

# And the associated sorted data frame
energy_mallows_best.sort_values('Cp')
```

```
Features: 218/218
Best subset: ['fossil_fuel_subsidies', 'per_capita_gdp', 'Urbanization', 'renewable_energy_share', 'industry_sha
re']
```



Out[207…

		subsets	Cp
subsets			
5	**(fossil_fuel_subsidies, per_capita_gdp, Urbanization, renewable_energy_share, industry_share)**	5	9.804622
	(fossil_fuel_subsidies, per_capita_gdp, renewable_energy_share, gii_technology, industry_share)	5	14.960474
	(fossil_fuel_subsidies, per_capita_gdp, renewable_energy_share, per_cap_energy_con, industry_share)	5	15.955191
4	**(fossil_fuel_subsidies, per_capita_gdp, renewable_energy_share, industry_share)**	4	17.032969
	(fossil_fuel_subsidies, per_capita_gdp, Urbanization, renewable_energy_share)	4	17.735033
	(fossil_fuel_subsidies, per_capita_gdp, Urbanization, industry_share)	4	19.918248
3	**(fossil_fuel_subsidies, per_capita_gdp, renewable_energy_share)**	3	24.843277
	(fossil_fuel_subsidies, per_capita_gdp, Urbanization)	3	25.569007
	(fossil_fuel_subsidies, per_capita_gdp, industry_share)	3	26.287196
2	**(fossil_fuel_subsidies, per_capita_gdp)**	2	31.915509
	(fossil_fuel_subsidies, Urbanization)	2	68.343848
	(per_capita_gdp, industry_share)	2	73.518371
1	**(per_capita_gdp,)**	1	99.047205
	(fossil_fuel_subsidies,)	1	120.486370
	(Urbanization,)	1	185.046489

Top 2 predictors based on Mallow's CP are in order of importance:

1. Per Capita GDP
2. Fossil Fuel Subsidies

A third important factor is:

1. Urbanization

Based on Boruta algorithm and Mallow's CP, taking common factors, we identified 3 important factors that can determine per capita co2 emissions across countries:

1. Per Capita GDP
2. Fossil Fuel Subsidies
3. Urbanization

We will now, use these 3 variables to build untransformed and transformed models (based on box-cox transformations done earlier) and test the models to measure what's the best model to predict per capita co2 emissions.

Question 3

Untransformed Version Of Models

In [208…

```
#Model 1 Untransformed- Per capita gdp
model_1_untrans = smf.ols('per_capita_co2_emissions ~ per_capita_gdp', data = raw_data_cleaned).fit()
print(model_1_untrans.summary())
model_1_untrans.residuals = model_1_untrans.resid
model_1_untrans.residuals

#Parameter

model_1_untrans_b1, model_1_untrans_b2 = model_1_untrans.params # extract estimates

#plotting regression line

plt.scatter(x = raw_data_cleaned['per_capita_gdp'], y = raw_data_cleaned['per_capita_co2_emissions'], color = '
plt.axline(xy1 = (0, model_1_untrans_b1), slope= model_1_untrans_b2, color='blue')
plt.ylabel('Per Capita CO2 Emissions')
plt.xlabel('Per Capita GDP')
plt.title('Per Capita CO2 Emissions Vs Per Capita GDP')
plt.tight_layout()
plt.show()

#Plotting Residuals

plt.scatter(x = raw_data_cleaned['per_capita_gdp'], y = model_1_untrans.resid, color = 'black')
plt.axhline(y = 0, color='red')
plt.ylabel('Residuals')
plt.xlabel('Per Capita GDP')
plt.title('Residuals Vs Per Capita GDP')
plt.tight_layout()
plt.show()
```

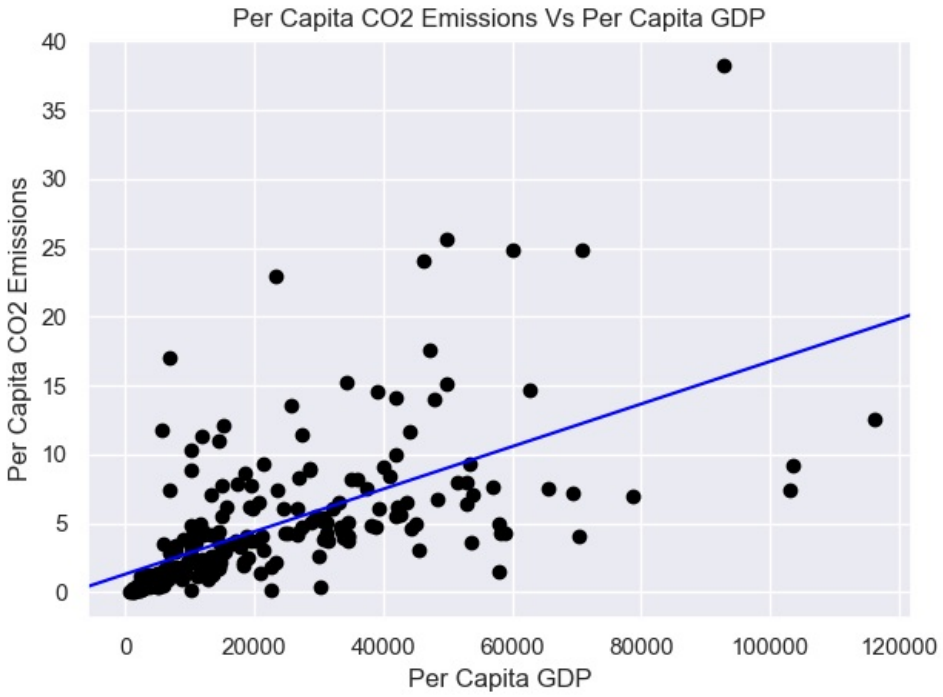
OLS Regression Results

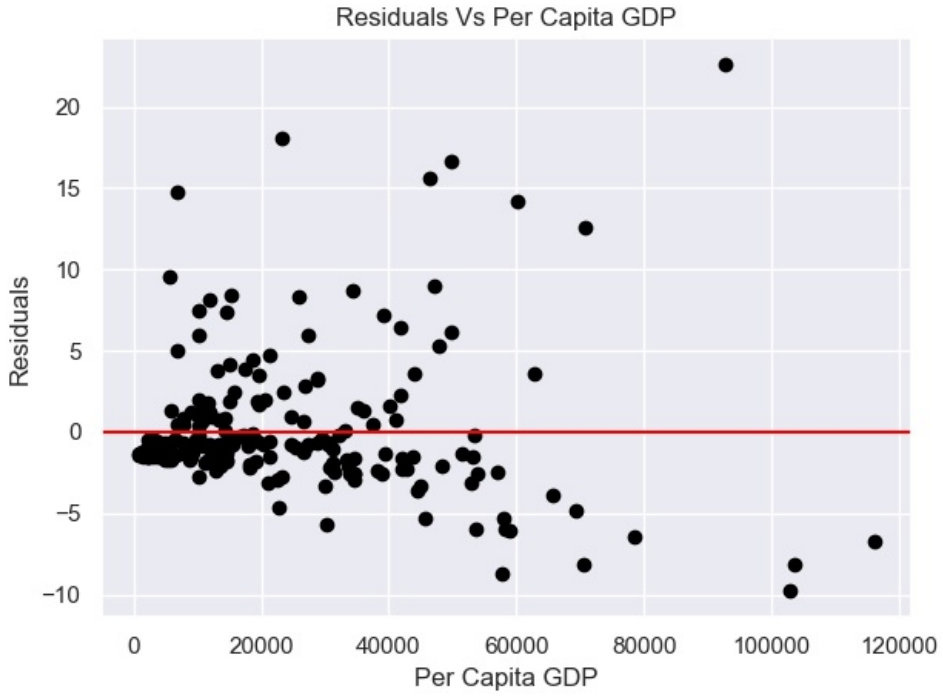
Dep. Variable:	per_capita_co2_emissions	R-squared:	0.371
Model:	OLS	Adj. R-squared:	0.368
Method:	Least Squares	F-statistic:	137.9
Date:	Fri, 25 Oct 2024	Prob (F-statistic):	2.40e-25
Time:	16:38:52	Log-Likelihood:	-668.37
No. Observations:	236	AIC:	1341.
Df Residuals:	234	BIC:	1348.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.3099	0.389	3.366	0.001	0.543	2.077
per_capita_gdp	0.0002	1.32e-05	11.745	0.000	0.000	0.000

Omnibus:	128.825	Durbin-Watson:	2.090
Prob(Omnibus):	0.000	Jarque-Bera (JB):	732.114
Skew:	2.166	Prob(JB):	1.06e-159
Kurtosis:	10.463	Cond. No.	4.29e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.29e+04. This might indicate that there are
strong multicollinearity or other numerical problems.





The untransformed model for Per Capita GDP tells us the following:

1. For every unit increase in per capita GDP, per capita CO_2 emissions are expected to increase by approximately 0.0002, assuming all else remains constant.
2. The high t-value (11.745) and low p-value (0.000) indicate that this coefficient is statistically significant.
3. The R-squared value implies that about 37.1% of the variance in per capita CO_2 emissions is explained by per capita GDP.
4. The F-statistic of (137.9) and its p-value (2.40e-25) show that the model as a whole is statistically significant, meaning that per capita GDP is a meaningful predictor of per capita CO_2 emissions in this context.
5. The residual plot suggests that while the model may provide reasonable predictions for lower GDP values, it struggles to explain the variability in CO_2 emissions for higher GDP levels and also includes few outliers at high Per Capita GDP levels.

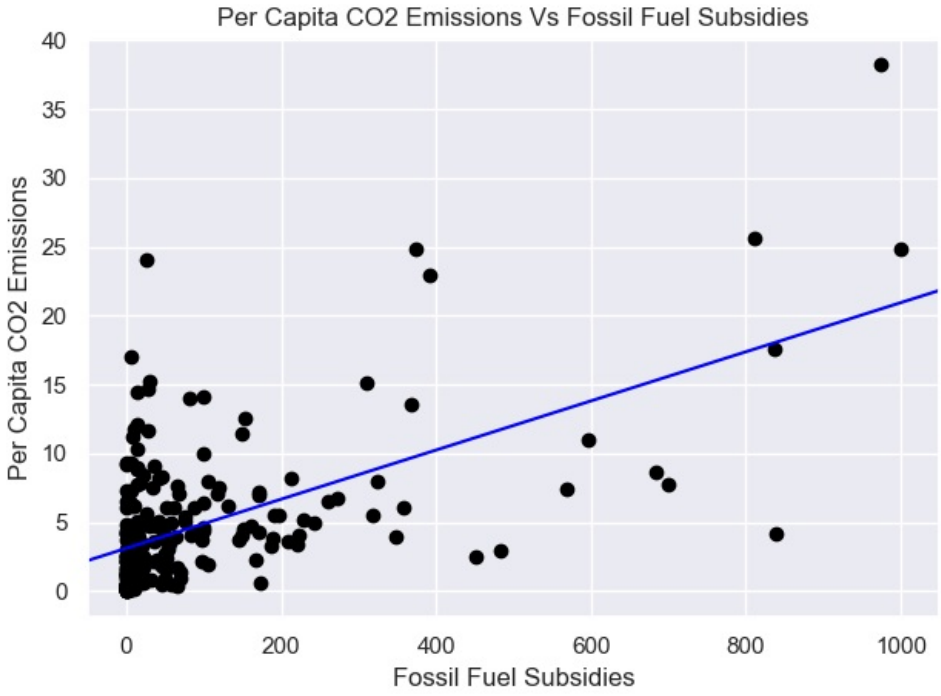
In [209…
```
print(np.sum(model_1_untrans.resid))
np.cov(model_1_untrans.resid, raw_data_cleaned.per_capita_gdp)
```

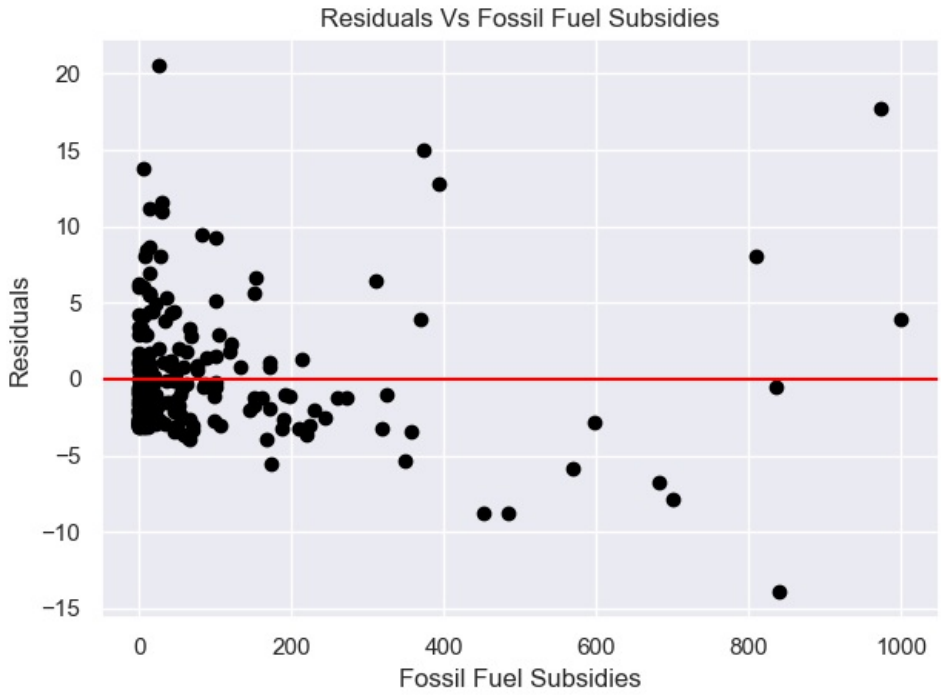
```
1.2079226507921703e-13
```

Out[209…
```
array([[1.69542479e+01, 2.55742170e-11],
       [2.55742170e-11, 4.18806965e+08]])
```

In [210…
```
# Model 2 Untransformed- (Fossil Fuel Subsidies)

model_2_untrans = smf.ols('per_capita_co2_emissions ~ fossil_fuel_subsidies', data = raw_data_cleaned).fit()
print(model_2_untrans.summary())
model_2_untrans.residuals = model_2_untrans.resid
model_2_untrans.residuals

#Parameter

model_2_untrans_b1, model_2_untrans_b2 = model_2_untrans.params # extract estimates

#plotting regression line

plt.scatter(x = raw_data_cleaned['fossil_fuel_subsidies'], y = raw_data_cleaned['per_capita_co2_emissions'], co
plt.axline(xy1 = (0, model_2_untrans_b1), slope= model_2_untrans_b2, color='blue')
plt.ylabel('Per Capita CO2 Emissions')
plt.xlabel('Fossil Fuel Subsidies')
plt.title('Per Capita CO2 Emissions Vs Fossil Fuel Subsidies')
plt.tight_layout()
plt.show()

#Plotting Residuals

plt.scatter(x = raw_data_cleaned['fossil_fuel_subsidies'], y = model_2_untrans.resid, color = 'black')
plt.axhline(y = 0, color='red')
plt.ylabel('Residuals')
plt.xlabel('Fossil Fuel Subsidies')
plt.title('Residuals Vs Fossil Fuel Subsidies')
plt.tight_layout()
plt.show()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     per_capita_co2_emissions   R-squared:                       0.330
Model:                                 OLS   Adj. R-squared:                  0.327
Method:                      Least Squares   F-statistic:                     115.3
Date:                     Fri, 25 Oct 2024   Prob (F-statistic):           3.93e-22
Time:                             16:39:02   Log-Likelihood:                -675.77
No. Observations:                      236   AIC:                             1356.
Df Residuals:                          234   BIC:                             1362.
Df Model:                                1
Covariance Type:                 nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
Intercept                3.1263      0.310     10.085      0.000       2.516       3.737
fossil_fuel_subsidies    0.0178      0.002     10.739      0.000       0.015       0.021
==============================================================================
Omnibus:                       85.041   Durbin-Watson:                   1.964
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              277.873
Skew:                           1.522   Prob(JB):                     4.58e-61
Kurtosis:                       7.358   Cond. No.                         209.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```





The untransformed model for Fossil Fuel Subsidies tells us the following:

1. For each additional unit increase in fossil fuel subsidies, the per capita CO2 emissions are expected to increase by approximately 0.0178 units.
2. The high t-value (10.739) and low p-value (0.000) indicate that this coefficient is statistically significant.
3. The model explains approximately 33% of the variance in per capita CO2 emissions.
4. The F-statistic of (115.3) and its p-value (3.93e-22) show that the model as a whole is statistically significant, meaning that per capita GDP is a meaningful predictor of per capita CO_2 emissions in this context.
5. In the residual graph most of the points are clustered near the zero line at lower values of fossil fuel subsidies,the residuals at higher fossil fuel subsidy levels (values above 400 on the x-axis) show a clear pattern: the residuals start to spread out more, and some residuals become more extreme. Also,there are several data points with large positive or negative residuals, especially as the fossil fuel subsidies increase.

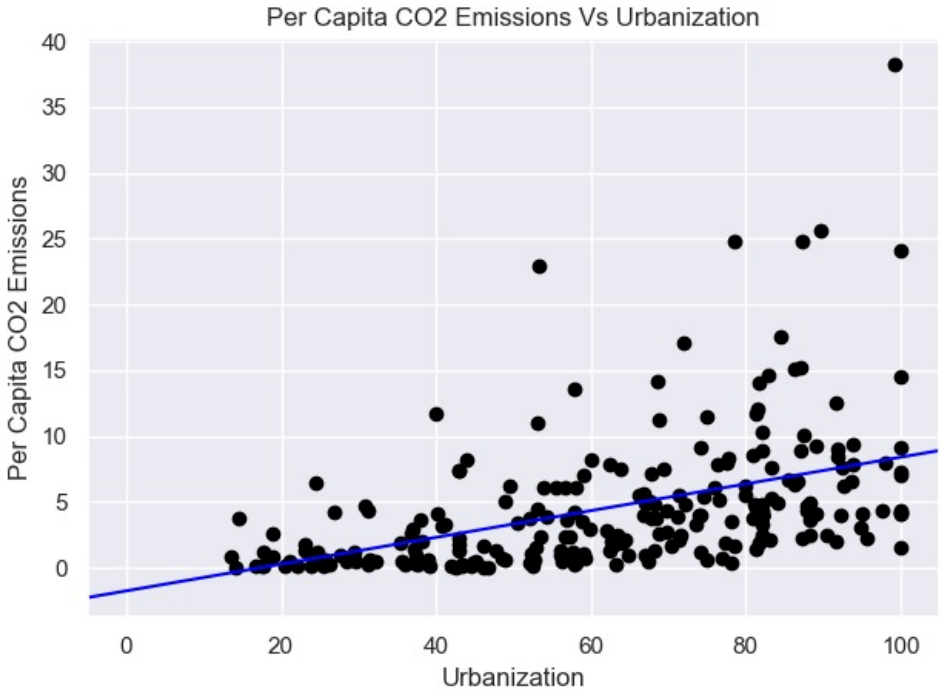
In [211…
```
print(np.sum(model_2_untrans.resid))
np.cov(model_2_untrans.resid, raw_data_cleaned.fossil_fuel_subsidies)
```

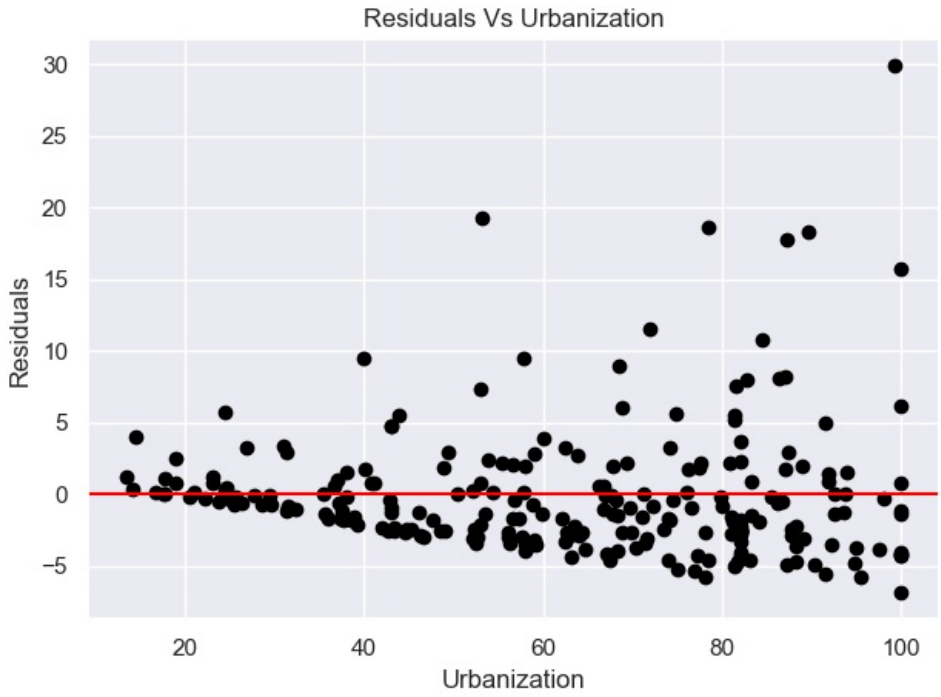
```
5.3290705182007514e-14
```

Out[211…
```
array([[ 1.80522330e+01, -2.32211413e-14],
       [-2.32211413e-14,  2.79790515e+04]])
```

In [212…
```
# Model 3 Untransformed (Urbanization)

model_3_untrans = smf.ols('per_capita_co2_emissions ~ Urbanization', data = raw_data_cleaned).fit()
print(model_3_untrans.summary())
model_3_untrans.residuals = model_3_untrans.resid
model_3_untrans.residuals

#Parameter

model_3_untrans_b1, model_3_untrans_b2 = model_3_untrans.params # extract estimates

#plotting regression line

plt.scatter(x = raw_data_cleaned['Urbanization'], y = raw_data_cleaned['per_capita_co2_emissions'], color = 'bla
plt.axline(xy1 = (0, model_3_untrans_b1), slope= model_3_untrans_b2, color='blue')
plt.ylabel('Per Capita CO2 Emissions')
plt.xlabel('Urbanization')
plt.title('Per Capita CO2 Emissions Vs Urbanization')
plt.tight_layout()
plt.show()

#Plotting Residuals

plt.scatter(x = raw_data_cleaned['Urbanization'], y = model_3_untrans.resid, color = 'black')
plt.axhline(y = 0, color='red')
plt.ylabel('Residuals')
plt.xlabel('Urbanization')
plt.title('Residuals Vs Urbanization')
plt.tight_layout()
plt.show()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     per_capita_co2_emissions   R-squared:                       0.207
Model:                                 OLS   Adj. R-squared:                  0.204
Method:                      Least Squares   F-statistic:                     61.25
Date:                     Fri, 25 Oct 2024   Prob (F-statistic):           1.73e-13
Time:                             16:39:04   Log-Likelihood:                -695.62
No. Observations:                      236   AIC:                             1395.
Df Residuals:                          234   BIC:                             1402.
Df Model:                                1
Covariance Type:                 nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      -1.7611      0.869     -2.027      0.044      -3.473      -0.049
Urbanization    0.1016      0.013      7.826      0.000       0.076       0.127
==============================================================================
Omnibus:                      158.771   Durbin-Watson:                   2.104
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1339.503
Skew:                           2.640   Prob(JB):                    1.35e-291
Kurtosis:                      13.409   Cond. No.                         193.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```





The untransformed model for Urbanization tells us the following:

1. The coefficient for urbanization is approximately 0.1016, indicating that for every one-unit increase in urbanization, per capita CO2

emissions increase by about 0.1016 units.

2. The high t-value (7.826) and low p-value (0.000) indicate that this coefficient is statistically significant.
3. Approximately 20.7% of the variance in per capita CO2 emissions can be explained by urbanization.

In [214…
```
print(np.sum(model_3_untrans.resid))
np.cov(model_3_untrans.resid, raw_data_cleaned.Urbanization)
```

```
-1.3500311979441904e-13
```

Out[214…
```
array([[ 2.13586142e+01, -3.24128431e-14],
       [-3.24128431e-14,  5.41804233e+02]])
```

Transformed Version Of Models

In [215…
```
# Power Transformation
def power_transform(variable, power):
    # For power = 0 (log transformation), handle it explicitly
    if power == 0:
        return np.log(variable)
    else:
        return np.power(variable, power)

# Transformation for Per Capita CO2 Emissions (power = -0.1)
raw_data_cleaned['per_capita_co2_emissions_transformed'] = power_transform(raw_data_cleaned['per_capita_co2_emi

# Apply transformation for Per Capita GDP (power = 0.2)
raw_data_cleaned['per_capita_gdp_transformed'] = power_transform(raw_data_cleaned['per_capita_gdp'], 0.2)

# Apply transformation for Urbanization (power = 1.2)
raw_data_cleaned['Urbanization_transformed'] = power_transform(raw_data_cleaned['Urbanization'], 1.2)

# Apply transformation for Fossil Fuel Subsidies (power = 0)
raw_data_cleaned['fossil_fuel_subsidies_transformed'] = power_transform(raw_data_cleaned['fossil_fuel_subsidies
#raw_data_cleaned[raw_data_cleaned['fossil_fuel_subsidies_transformed'] == -np.inf] = 0
raw_data_cleaned['fossil_fuel_subsidies_transformed'] = raw_data_cleaned['fossil_fuel_subsidies_transformed'].r
```

In [216…
```
# Model 1 Transformed (Per Capita GDP)
model_transformed_1 = smf.ols('per_capita_co2_emissions_transformed ~ per_capita_gdp_transformed', data = raw_d
print(model_transformed_1.summary())
model_transformed_1.residuals = model_transformed_1.resid
model_transformed_1.residuals

#Parameter

model_transformed_1_b1, model_transformed_1_b2 = model_transformed_1.params # extract estimates

# Plotting regression line

plt.scatter(x = raw_data_cleaned['per_capita_gdp_transformed'], y = raw_data_cleaned['per_capita_co2_emissions_
plt.axline(xy1 = (0, model_transformed_1_b1), slope= model_transformed_1_b2, color='blue')
plt.ylabel('Per Capita CO2 Emissions Transformed')
plt.xlabel('Per Capita GDP Transformed')
plt.title('Per Capita CO2 Emissions Transformed Vs Per Capita GDP Transformed')
plt.tight_layout()
plt.show()

# Residuals Plot
plt.scatter(x= raw_data_cleaned.per_capita_gdp_transformed, y= model_transformed_1.resid, color='black')
plt.axhline(y=0, color='red')
plt.ylabel('Residuals')
plt.xlabel('Per Capita GDP Transformed')
plt.title('Residuals Vs Per Capita GDP Transformed')
plt.tight_layout()
plt.show()
```

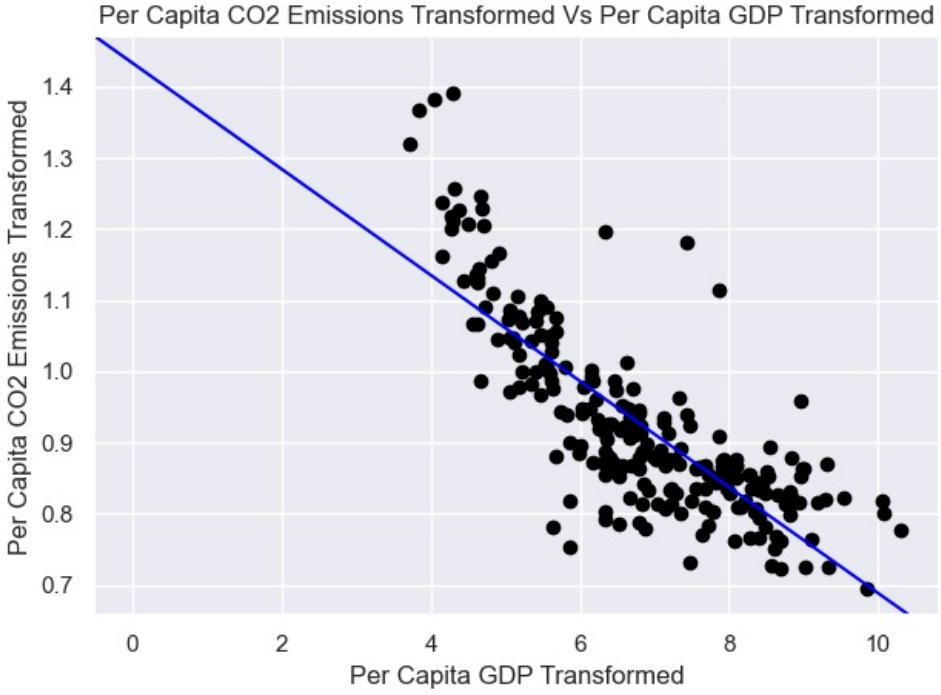
OLS Regression Results

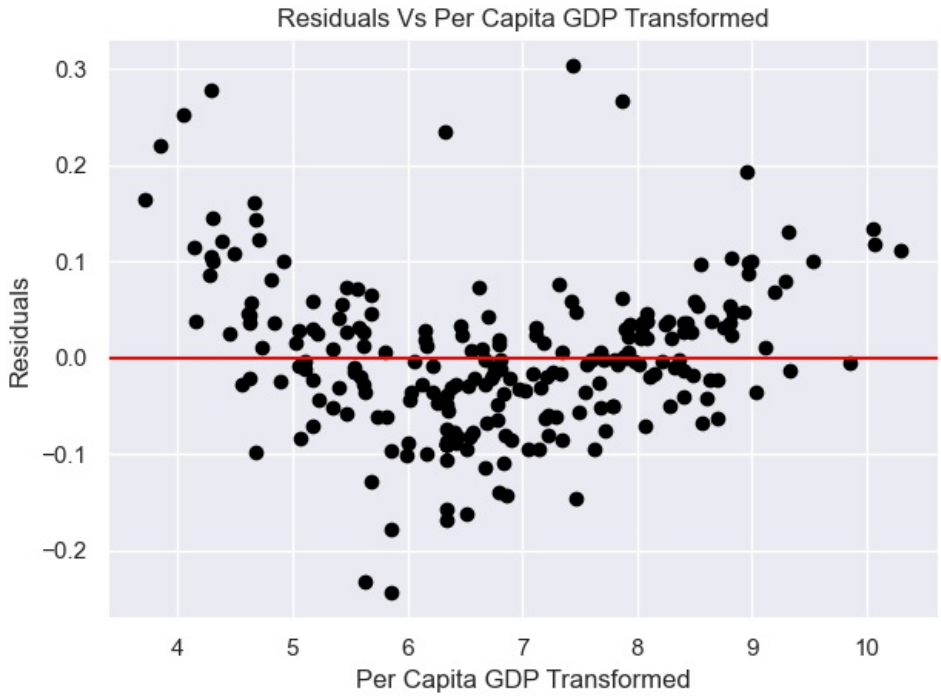
Dep. Variable:	per_capita_co2_emissions_transformed	R-squared:	0.635
Model:	OLS	Adj. R-squared:	0.634
Method:	Least Squares	F-statistic:	407.3
Date:	Fri, 25 Oct 2024	Prob (F-statistic):	3.87e-53
Time:	16:39:23	Log-Likelihood:	259.52
No. Observations:	236	AIC:	-515.0
Df Residuals:	234	BIC:	-508.1
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.4331	0.026	55.842	0.000	1.383	1.484
per_capita_gdp_transformed	-0.0744	0.004	-20.181	0.000	-0.082	-0.067

Omnibus:	26.654	Durbin-Watson:	1.910
Prob(Omnibus):	0.000	Jarque-Bera (JB):	49.908
Skew:	0.598	Prob(JB):	1.45e-11
Kurtosis:	4.909	Cond. No.	34.6

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.





In [ ]:

In [217…

```
# Model 2 Transformed (Fossil Fuel Subsidies)
model_transformed_2 = smf.ols('per_capita_co2_emissions_transformed ~ fossil_fuel_subsidies_transformed', data =
print(model_transformed_2.summary())
model_transformed_2.residuals = model_transformed_2.resid
model_transformed_2.residuals

#Parameter

model_transformed_2_b1, model_transformed_2_b2 = model_transformed_2.params # extract estimates

# Plotting regression line

plt.scatter(x = raw_data_cleaned['fossil_fuel_subsidies_transformed'], y = raw_data_cleaned['per_capita_co2_emi
plt.axline(xy1 = (0, model_transformed_2_b1), slope= model_transformed_2_b2, color='blue')
plt.ylabel('Per Capita CO2 Emissions Transformed')
plt.xlabel('Fossil Fuel Subsidies Transformed')
plt.title('Per Capita CO2 Emissions Transformed Vs Fossil Fuel Subsidies Transformed')
plt.tight_layout()
plt.show()

# Residuals Plot
plt.scatter(x= raw_data_cleaned['fossil_fuel_subsidies_transformed'], y= model_transformed_2.resid, color='blac
plt.xlim(-5,5)
plt.ylim(-5,5)
plt.axhline(y=0, color='red')
plt.ylabel('Residuals')
plt.xlabel('Fossil Fuel Subsidies Transformed')
plt.title('Residuals Vs Fossil Fuel Subsidies Transformed')
plt.tight_layout()
plt.show()
```

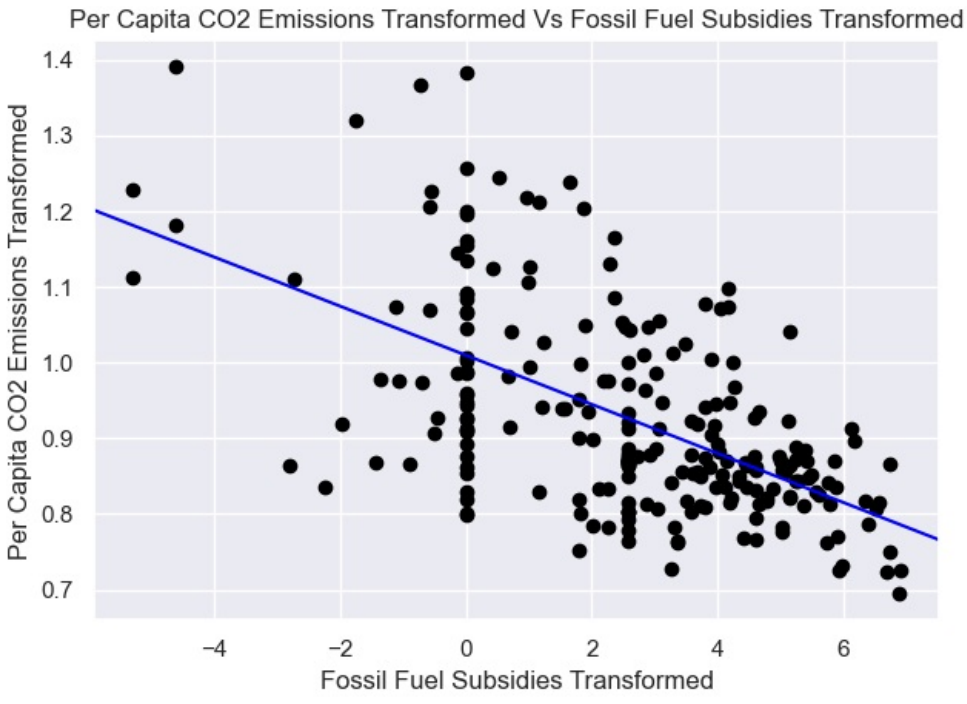
OLS Regression Results

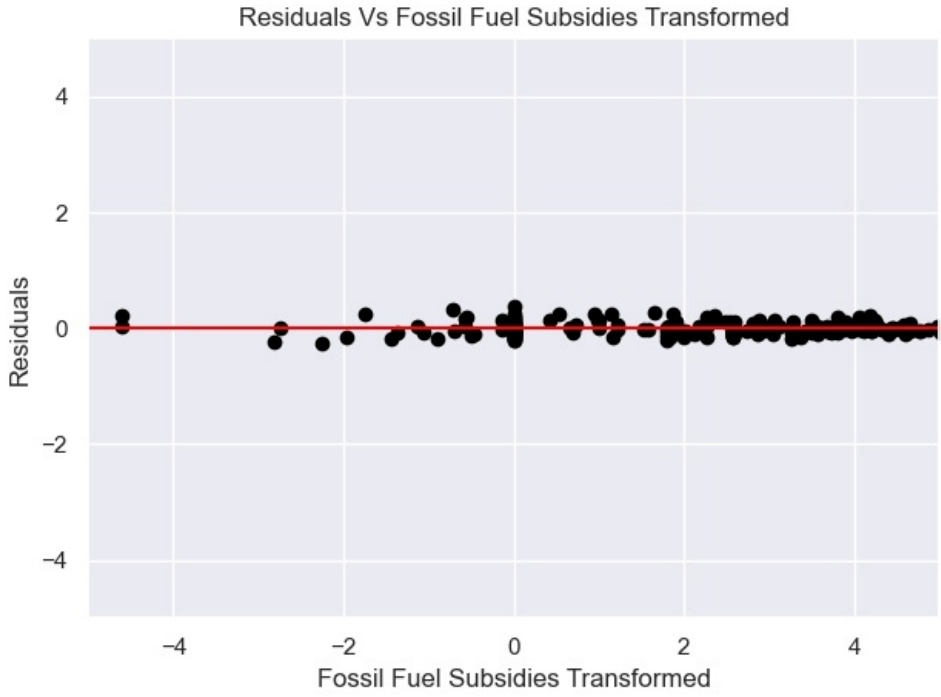
Dep. Variable:	per_capita_co2_emissions_transformed	R-squared:	0.336
Model:	OLS	Adj. R-squared:	0.333
Method:	Least Squares	F-statistic:	118.5
Date:	Fri, 25 Oct 2024	Prob (F-statistic):	1.33e-22
Time:	16:39:25	Log-Likelihood:	188.92
No. Observations:	236	AIC:	-373.8
Df Residuals:	234	BIC:	-366.9
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.0098	0.010	96.528	0.000	0.989	1.030
fossil_fuel_subsidies_transformed	-0.0324	0.003	-10.888	0.000	-0.038	-0.027

Omnibus:	16.746	Durbin-Watson:	2.004
Prob(Omnibus):	0.000	Jarque-Bera (JB):	18.253
Skew:	0.628	Prob(JB):	0.000109
Kurtosis:	3.528	Cond. No.	5.40

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

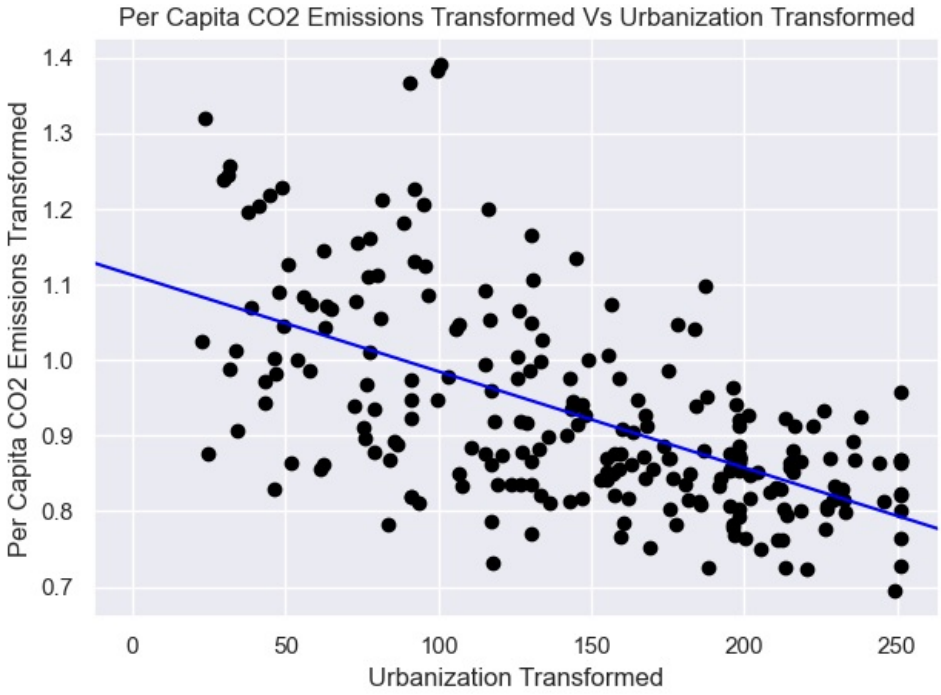


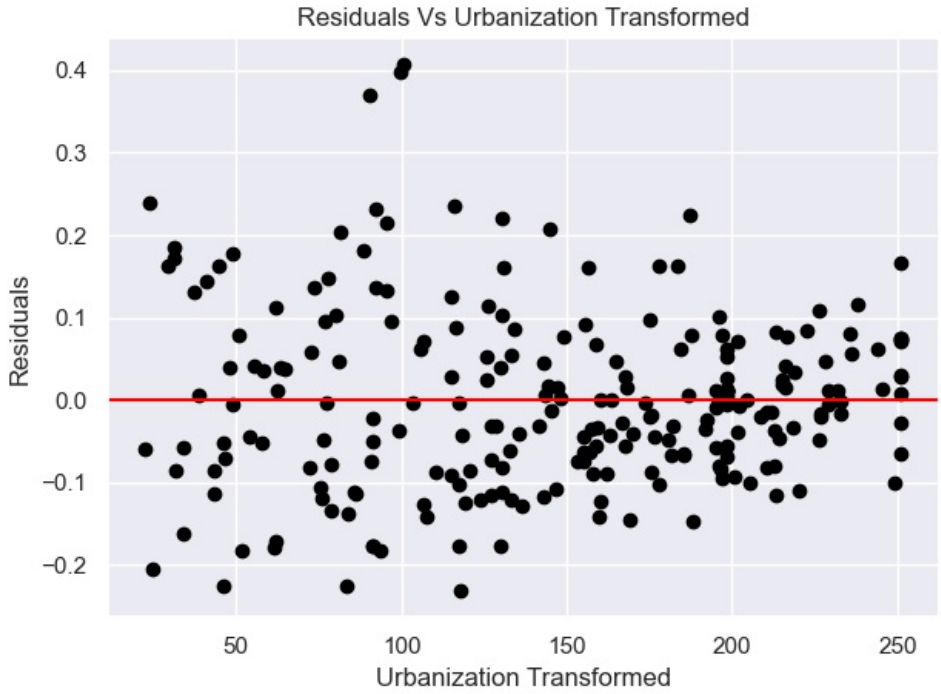


In [218…

```
# Model 3 Transformed (Urbanization)
model_transformed_3 = smf.ols('per_capita_co2_emissions_transformed ~ Urbanization_transformed', data = raw_data
print(model_transformed_3.summary())
model_transformed_3.residuals = model_transformed_3.resid
model_transformed_3.residuals

#Parameter

model_transformed_3_b1, model_transformed_3_b2 = model_transformed_3.params # extract estimates

# Plotting regression line

plt.scatter(y = raw_data_cleaned['per_capita_co2_emissions_transformed'], x = raw_data_cleaned['Urbanization_tra
plt.axline(xy1 = (0, model_transformed_3_b1), slope= model_transformed_3_b2, color='blue')
plt.ylabel('Per Capita CO2 Emissions Transformed')
plt.xlabel('Urbanization Transformed')
plt.title('Per Capita CO2 Emissions Transformed Vs Urbanization Transformed')
plt.tight_layout()
plt.show()

# Residuals Plot
plt.scatter(x = raw_data_cleaned['Urbanization_transformed'], y= model_transformed_3.resid, color='black')
plt.axhline(y = 0, color='red')
plt.ylabel('Residuals')
plt.xlabel('Urbanization Transformed')
plt.title('Residuals Vs Urbanization Transformed')
plt.tight_layout()
plt.show()
```

```
                                OLS Regression Results
==============================================================================================
Dep. Variable:     per_capita_co2_emissions_transformed   R-squared:                       0.357
Model:                                             OLS   Adj. R-squared:                  0.354
Method:                                  Least Squares   F-statistic:                     129.8
Date:                                 Fri, 25 Oct 2024   Prob (F-statistic):           3.23e-24
Time:                                         16:39:30   Log-Likelihood:                 192.65
No. Observations:                                  236   AIC:                            -381.3
Df Residuals:                                      234   BIC:                            -374.4
Df Model:                                            1
Covariance Type:                             nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                       1.1126      0.018     62.553      0.000       1.078       1.148
Urbanization_transformed       -0.0013      0.000    -11.395      0.000      -0.001      -0.001
==============================================================================
Omnibus:                       25.412   Durbin-Watson:                   1.934
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               34.203
Skew:                           0.710   Prob(JB):                     3.74e-08
Kurtosis:                       4.209   Cond. No.                         405.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```





Model interpretation- Based on transformed models, here is the interpretation of the transformed models:

Y Variable Transformation: Per capita CO2 emissions was transformed with a power of -0.1

1. Per Capita GDP: Transformed per capita gdp with a power of 0.8. Downward sloping regression line shows that increase in per capita GDP increases (per capita CO2 emissions)^(-0.1) which means countries with higher per capita GDP have higher per capita co2 emissions.

2. Fossil Fuel Subsidies: Transformed fossil fuel subsidies per capita with a log. Here, 1% increase in fossil fuel subsidies leads to a -0.032 change in per capita co2 emissions^(-0.1) . Thus, increase in fossil fuel subsidies leads to higher co2 emissions per capita.

3. Urbanization: Transformed urbanization with a power of 0.2. Downward sloping regression line shows that increase in per capita GDP increases (per capita CO2 emissions)^(-0.1) which means countries with higher % of population in urban areas leads to higher per capita co2 emissions.

Evaluate transformation do transformed variables fit better

1. Per Capita GDP: Transforming both the variables improves data fit since R^2 increases from 0.371 to 0.635.
2. Fossil Fuel Subsidies: Transforming both the variables improves data fit very marginally since R^2 increases from 0.330 to 0.336.
3. Urbanization: Transforming both the variables improves data a bit since R^2 increases from 0.207 to 0.357.

Overall, in terms of data fit, based on these transformed models, Per capita GDP transformed seems to be the best predictor of Per

Capita CO2 emissions. Moreover, increase in per capita GDP increases per capita CO2 emissions.

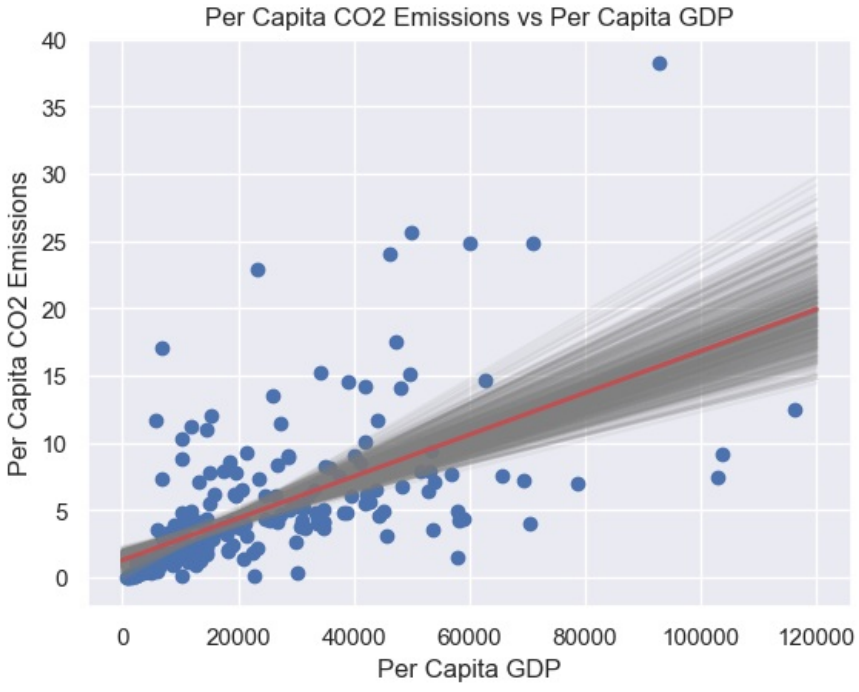
Bootstrapping Regression Models

In this case, to check robustness of our regression model coefficients, we bootstrap both our untransformed and transformed models

Untransformed Models

In [219…

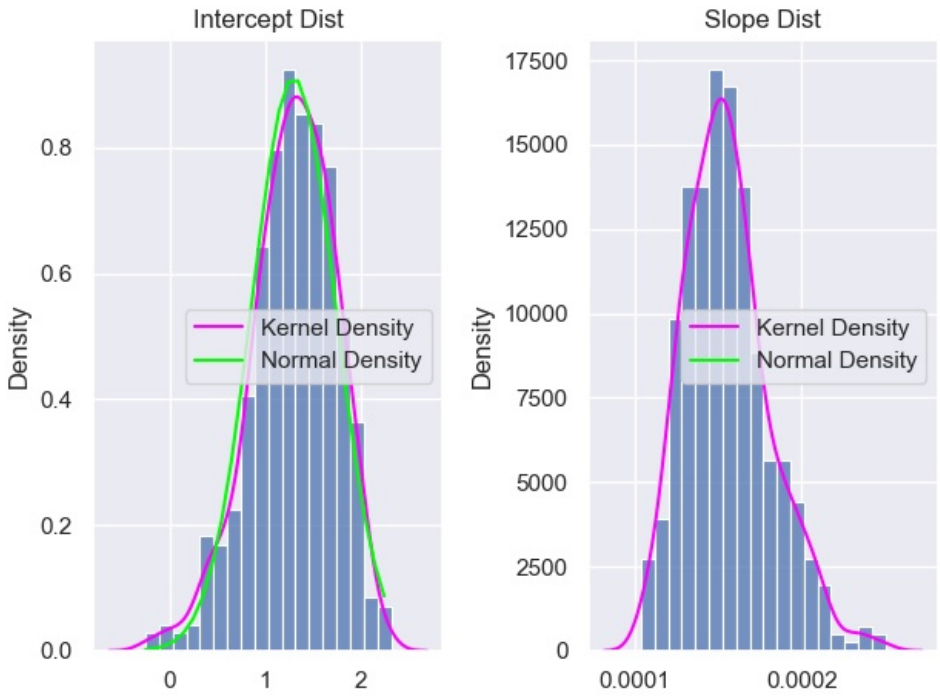
```
# Model 1 Untransformed (Per capita GDP)

random.seed(90095)
n_boot = 500 # number of bootstraps

# initializations
boot_rmse = []
boot_b1 = []
boot_b2 = []
boot_sig2hat = [] # sigma^2 hat estimates (or std error ^2)
boot_r2 = [] # R^2 (no need to adjust)

per_capita_gdp_span_x = pd.DataFrame({'per_capita_gdp':np.linspace(0, 120000, 200)}) # x spanning 3.5 through 3

for iter in range(n_boot):
   # Sampling with replacement
   raw_data_cleaned_sample = raw_data_cleaned.sample(n=raw_data_cleaned.shape[0], replace=True) # sample method
   raw_data_cleaned_sample_ols = smf.ols('per_capita_co2_emissions ~ per_capita_gdp', data=raw_data_cleaned_samp

   # Append coefficients
   boot_b1.append(raw_data_cleaned_sample_ols.params[0])          # intercept
   boot_b2.append(raw_data_cleaned_sample_ols.params[1])          # slope
   # Performance measures
   boot_sig2hat.append(raw_data_cleaned_sample_ols.scale)         # scale = estimated sigma^2
   boot_rmse.append(np.sqrt(np.mean(raw_data_cleaned_sample_ols.resid ** 2))) # RMSE
   boot_r2.append(raw_data_cleaned_sample_ols.rsquared)           # R^2

   # Plot a greyed out line for this iteration
   raw_data_cleaned_sample_yhat = raw_data_cleaned_sample_ols.predict(per_capita_gdp_span_x)
   plt.plot(per_capita_gdp_span_x, raw_data_cleaned_sample_yhat, color='grey', alpha=0.1)

# Fit averaged model
raw_data_cleaned_ols_span_y = np.mean(boot_b1) + per_capita_gdp_span_x * np.mean(boot_b2)

# Plotting the fitted lines
plt.scatter( raw_data_cleaned['per_capita_gdp'], raw_data_cleaned['per_capita_co2_emissions'])
plt.plot(per_capita_gdp_span_x, raw_data_cleaned_ols_span_y, color='r', linewidth=2)
plt.xlabel('Per Capita GDP')
plt.ylabel('Per Capita CO2 Emissions')
plt.title('Per Capita CO2 Emissions vs Per Capita GDP')
plt.show()
```

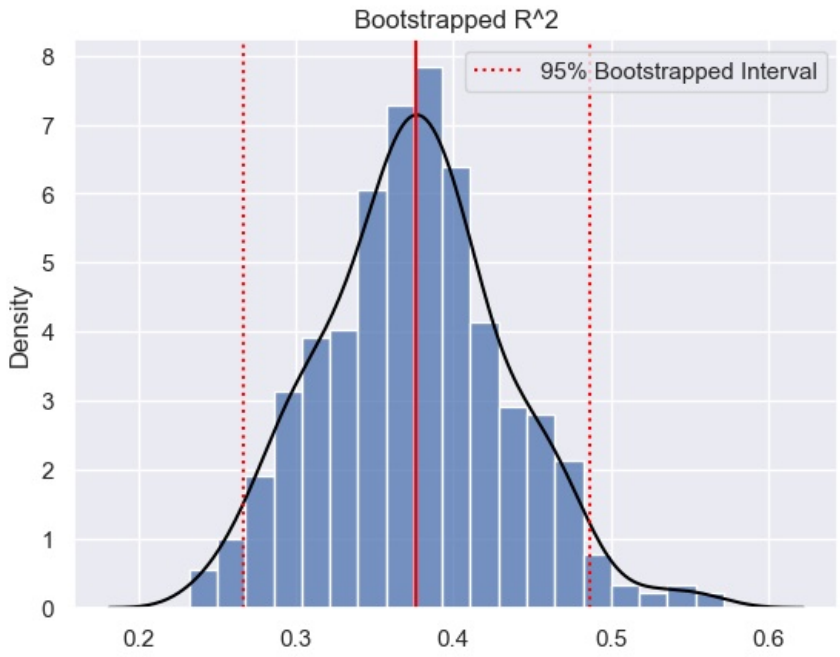


In [220…

```
# Next the distributions of the estimates

# For plotting normal PDFs
```

```
b1_range = np.arange(np.min(boot_b1), np.max(boot_b1), 0.1)
b2_range = np.arange(np.min(boot_b2), np.max(boot_b2), 0.1)

# Plotting beta estimates
fig6, ax6 = plt.subplots(1, 2)
sns.histplot(boot_b1, stat='density', kde=False, ax=ax6[0])
sns.kdeplot(boot_b1, color='fuchsia', ax=ax6[0], label='Kernel Density')
sns.lineplot(x=b1_range, y=norm.pdf(b1_range, np.mean(boot_b1), np.std(boot_b1)), color='lime',ax=ax6[0], label=
ax6[0].set_title('Intercept Dist')
sns.histplot(boot_b2, stat='density', kde=False, ax=ax6[1])
sns.kdeplot(boot_b2, color='fuchsia', ax=ax6[1], label='Kernel Density')
sns.lineplot(x=b2_range, y=norm.pdf(b2_range, np.mean(boot_b2), np.std(boot_b2)), color='lime',ax=ax6[1], label=
ax6[1].set_title('Slope Dist')
plt.tight_layout()
plt.show()

# Bootstrap for distribution of R^2 (coefficient of determination)
sns.histplot(boot_r2, stat='density', kde=False)
sns.kdeplot(boot_r2, color='black')
plt.axvline(x=np.quantile(boot_r2, q=0.025), color='red', linestyle=':', label='95% Bootstrapped Interval')
plt.axvline(x=np.quantile(boot_r2, q=0.5), color='red')
plt.axvline(x=np.quantile(boot_r2, q=0.975), color='red', linestyle=':')
plt.title('Bootstrapped R^2')
plt.legend()
plt.show()

# Bootstrap for estimated residual standard deviation
boot_stdhat = np.sqrt(boot_sig2hat)
print('--- sigma hat ---\n' +
      f'estimate:\t\t{boot_stdhat.mean():.5}\n' +
      f'std error:\t\t{boot_stdhat.std():.5}\n' +
      f'median:\t\t\t{np.quantile(boot_stdhat, q=0.5):.5}\n' +
      f'2.5 percentile:\t\t{np.quantile(boot_stdhat, q=0.025):.5}\n' +
      f'97.5 percentile:\t{np.quantile(boot_stdhat, q=0.975):.5}'
      )
```



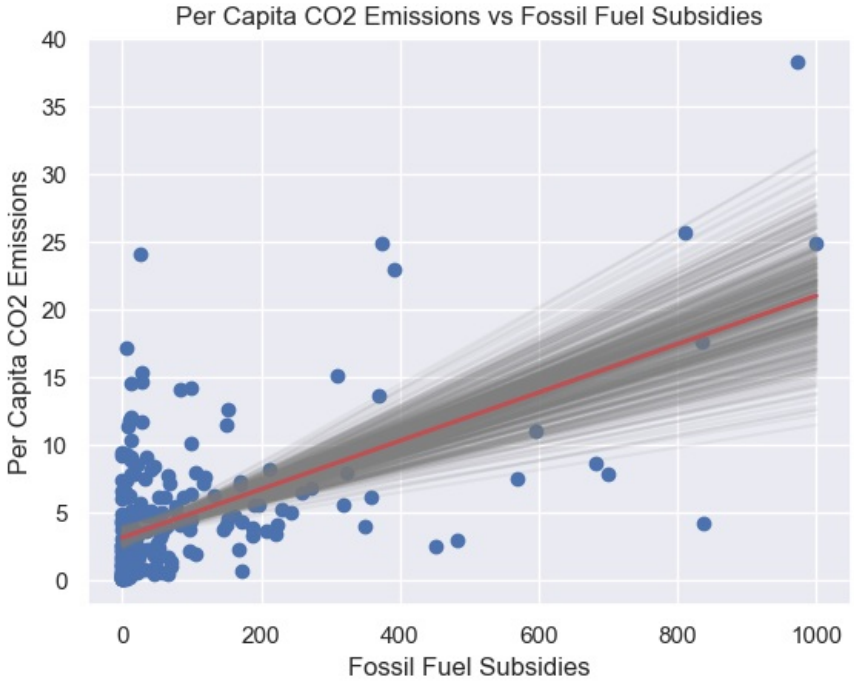



```
--- sigma hat ---
estimate:               4.0857
std error:              0.42277
median:                 4.1116
2.5 percentile:         3.31
97.5 percentile:        4.9073
```

In [221…

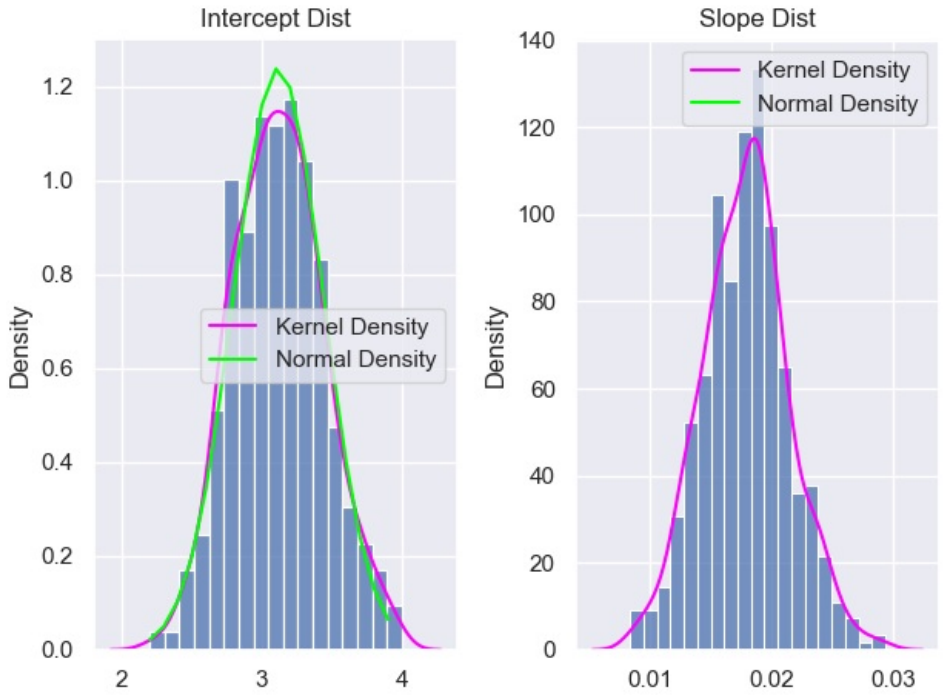
```
# Model 2 Untransformed (Fossil Fuel Subsidies)

random.seed(90095)
n_boot = 500 # number of bootstraps

# initializations
boot_rmse = []
boot_b1 = []
boot_b2 = []
boot_sig2hat = [] # sigma^2 hat estimates (or std error ^2)
boot_r2 = [] # R^2 (no need to adjust)

fossil_fuel_subsidies_span_x = pd.DataFrame({'fossil_fuel_subsidies':np.linspace(0, 1000, 200)}) # x spanning 3

for iter in range(n_boot):
   # Sampling with replacement
   raw_data_cleaned_sample = raw_data_cleaned.sample(n=raw_data_cleaned.shape[0], replace=True) # sample method
   raw_data_cleaned_sample_ols = smf.ols('per_capita_co2_emissions ~ fossil_fuel_subsidies', data=raw_data_clea

   # Append coefficients
   boot_b1.append(raw_data_cleaned_sample_ols.params[0])          # intercept
   boot_b2.append(raw_data_cleaned_sample_ols.params[1])          # slope
   # Performance measures
   boot_sig2hat.append(raw_data_cleaned_sample_ols.scale)         # scale = estimated sigma^2
   boot_rmse.append(np.sqrt(np.mean(raw_data_cleaned_sample_ols.resid ** 2))) # RMSE
   boot_r2.append(raw_data_cleaned_sample_ols.rsquared)           # R^2

   # Plot a greyed out line for this iteration
   raw_data_cleaned_sample_yhat = raw_data_cleaned_sample_ols.predict(fossil_fuel_subsidies_span_x)
   plt.plot(fossil_fuel_subsidies_span_x, raw_data_cleaned_sample_yhat, color='grey', alpha=0.1)
```

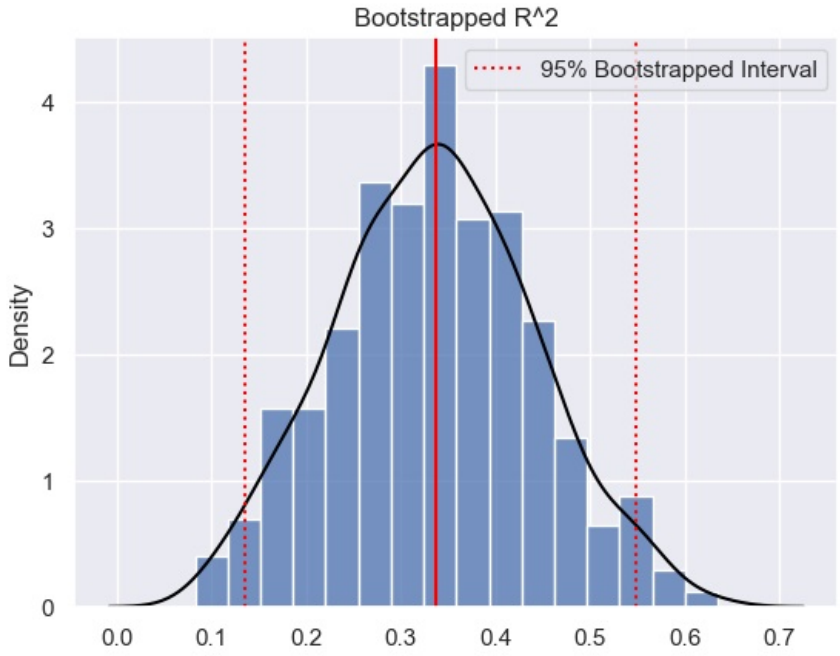
```
# Fit averaged model
raw_data_cleaned_ols_span_y = np.mean(boot_b1) + fossil_fuel_subsidies_span_x * np.mean(boot_b2)

# Plotting the fitted lines
plt.scatter( raw_data_cleaned['fossil_fuel_subsidies'], raw_data_cleaned['per_capita_co2_emissions'])
plt.plot(fossil_fuel_subsidies_span_x, raw_data_cleaned_ols_span_y, color='r', linewidth=2)
plt.xlabel('Fossil Fuel Subsidies')
plt.ylabel('Per Capita CO2 Emissions')
plt.title('Per Capita CO2 Emissions vs Fossil Fuel Subsidies')
plt.show()
```



In [222…

```
# Next the distributions of the estimates

# For plotting normal PDFs
b1_range = np.arange(np.min(boot_b1), np.max(boot_b1), 0.1)
b2_range = np.arange(np.min(boot_b2), np.max(boot_b2), 0.1)

# Plotting beta estimates
fig6, ax6 = plt.subplots(1, 2)
sns.histplot(boot_b1, stat='density', kde=False, ax=ax6[0])
sns.kdeplot(boot_b1, color='fuchsia', ax=ax6[0], label='Kernel Density')
sns.lineplot(x=b1_range, y=norm.pdf(b1_range, np.mean(boot_b1), np.std(boot_b1)), color='lime',ax=ax6[0], label=
ax6[0].set_title('Intercept Dist')
sns.histplot(boot_b2, stat='density', kde=False, ax=ax6[1])
sns.kdeplot(boot_b2, color='fuchsia', ax=ax6[1], label='Kernel Density')
sns.lineplot(x=b2_range, y=norm.pdf(b2_range, np.mean(boot_b2), np.std(boot_b2)), color='lime',ax=ax6[1], label=
ax6[1].set_title('Slope Dist')
plt.tight_layout()
plt.show()

# Bootstrap for distribution of R^2 (coefficient of determination)
sns.histplot(boot_r2, stat='density', kde=False)
sns.kdeplot(boot_r2, color='black')
plt.axvline(x=np.quantile(boot_r2, q=0.025), color='red', linestyle=':', label='95% Bootstrapped Interval')
plt.axvline(x=np.quantile(boot_r2, q=0.5), color='red')
plt.axvline(x=np.quantile(boot_r2, q=0.975), color='red', linestyle=':')
plt.title('Bootstrapped R^2')
plt.legend()
plt.show()

# Bootstrap for estimated residual standard deviation
boot_stdhat = np.sqrt(boot_sig2hat)
print('--- sigma hat ---\n' +
      f'estimate:\t\t{boot_stdhat.mean():.5}\n' +
      f'std error:\t\t{boot_stdhat.std():.5}\n' +
      f'median:\t\t\t{np.quantile(boot_stdhat, q=0.5):.5}\n' +
      f'2.5 percentile:\t\t{np.quantile(boot_stdhat, q=0.025):.5}\n' +
      f'97.5 percentile:\t{np.quantile(boot_stdhat, q=0.975):.5}'
      )
```



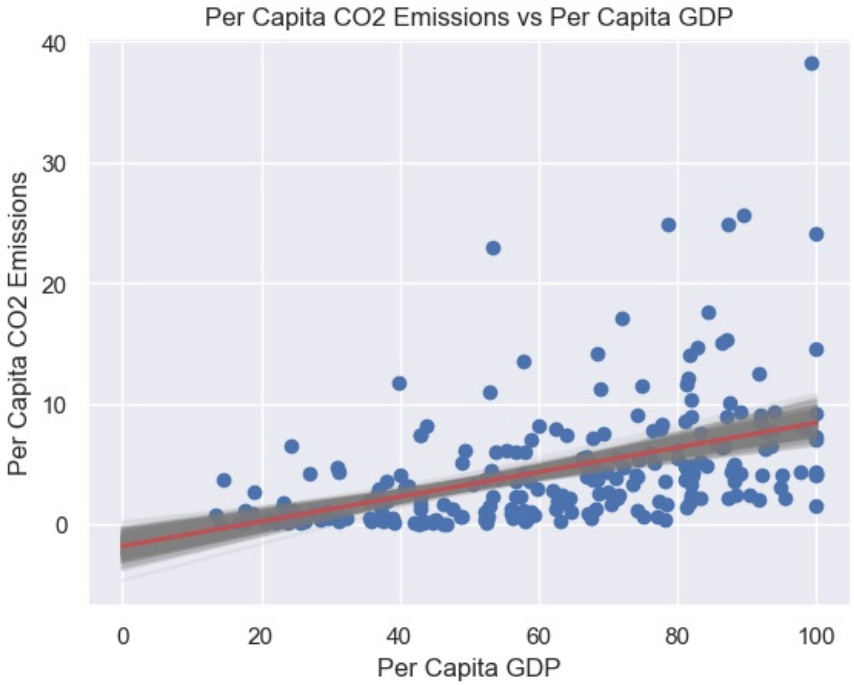



```
--- sigma hat ---
estimate:               4.1882
std error:              0.36054
median:                 4.1876
2.5 percentile:         3.4686
97.5 percentile:        4.8915
```

In [223…

```
# Model 3 Untransformed (Urbanization)

random.seed(90095)
n_boot = 500 # number of bootstraps

# initializations
boot_rmse = []
boot_b1 = []
boot_b2 = []
boot_sig2hat = [] # sigma^2 hat estimates (or std error ^2)
boot_r2 = [] # R^2 (no need to adjust)

Urbanization_span_x = pd.DataFrame({'Urbanization':np.linspace(0, 100, 200)}) # x spanning 3.5 through 34 for g

for iter in range(n_boot):
   # Sampling with replacement
   raw_data_cleaned_sample = raw_data_cleaned.sample(n=raw_data_cleaned.shape[0], replace=True) # sample method
   raw_data_cleaned_sample_ols = smf.ols('per_capita_co2_emissions ~ Urbanization', data=raw_data_cleaned_sample

   # Append coefficients
   boot_b1.append(raw_data_cleaned_sample_ols.params[0])          # intercept
   boot_b2.append(raw_data_cleaned_sample_ols.params[1])          # slope
   # Performance measures
```

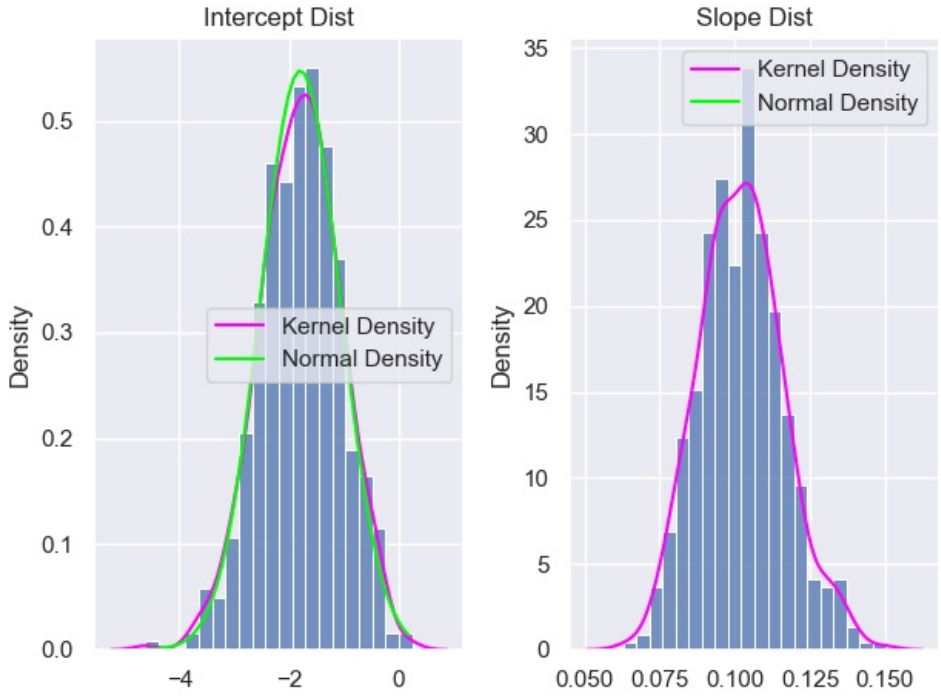
```
    boot_sig2hat.append(raw_data_cleaned_sample_ols.scale)         # scale = estimated sigma^2
    boot_rmse.append(np.sqrt(np.mean(raw_data_cleaned_sample_ols.resid ** 2))) # RMSE
    boot_r2.append(raw_data_cleaned_sample_ols.rsquared)           # R^2

    # Plot a greyed out line for this iteration
    raw_data_cleaned_sample_yhat = raw_data_cleaned_sample_ols.predict(Urbanization_span_x)
    plt.plot(Urbanization_span_x, raw_data_cleaned_sample_yhat, color='grey', alpha=0.1)

# Fit averaged model
raw_data_cleaned_ols_span_y = np.mean(boot_b1) + Urbanization_span_x * np.mean(boot_b2)

# Plotting the fitted lines
plt.scatter( raw_data_cleaned['Urbanization'], raw_data_cleaned['per_capita_co2_emissions'])
plt.plot(Urbanization_span_x, raw_data_cleaned_ols_span_y, color='r', linewidth=2)
plt.xlabel('Per Capita GDP')
plt.ylabel('Per Capita CO2 Emissions')
plt.title('Per Capita CO2 Emissions vs Per Capita GDP')
plt.show()
```

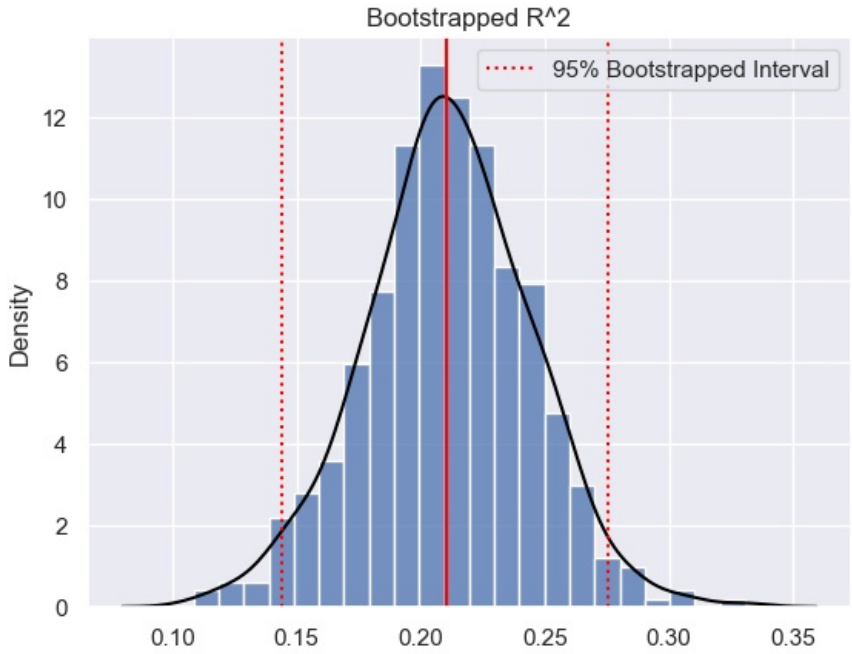


In [224…

```
# Next the distributions of the estimates

# For plotting normal PDFs
b1_range = np.arange(np.min(boot_b1), np.max(boot_b1), 0.1)
b2_range = np.arange(np.min(boot_b2), np.max(boot_b2), 0.1)

# Plotting beta estimates
fig6, ax6 = plt.subplots(1, 2)
sns.histplot(boot_b1, stat='density', kde=False, ax=ax6[0])
sns.kdeplot(boot_b1, color='fuchsia', ax=ax6[0], label='Kernel Density')
sns.lineplot(x=b1_range, y=norm.pdf(b1_range, np.mean(boot_b1), np.std(boot_b1)), color='lime',ax=ax6[0], label=
ax6[0].set_title('Intercept Dist')
sns.histplot(boot_b2, stat='density', kde=False, ax=ax6[1])
sns.kdeplot(boot_b2, color='fuchsia', ax=ax6[1], label='Kernel Density')
sns.lineplot(x=b2_range, y=norm.pdf(b2_range, np.mean(boot_b2), np.std(boot_b2)), color='lime',ax=ax6[1], label=
ax6[1].set_title('Slope Dist')
plt.tight_layout()
plt.show()

# Bootstrap for distribution of R^2 (coefficient of determination)
sns.histplot(boot_r2, stat='density', kde=False)
sns.kdeplot(boot_r2, color='black')
plt.axvline(x=np.quantile(boot_r2, q=0.025), color='red', linestyle=':', label='95% Bootstrapped Interval')
plt.axvline(x=np.quantile(boot_r2, q=0.5), color='red')
plt.axvline(x=np.quantile(boot_r2, q=0.975), color='red', linestyle=':')
plt.title('Bootstrapped R^2')
plt.legend()
plt.show()

# Bootstrap for estimated residual standard deviation
boot_stdhat = np.sqrt(boot_sig2hat)
print('--- sigma hat ---\n' +
      f'estimate:\t\t{boot_stdhat.mean():.5}\n' +
      f'std error:\t\t{boot_stdhat.std():.5}\n' +
      f'median:\t\t\t{np.quantile(boot_stdhat, q=0.5):.5}\n' +
      f'2.5 percentile:\t\t{np.quantile(boot_stdhat, q=0.025):.5}\n' +
      f'97.5 percentile:\t{np.quantile(boot_stdhat, q=0.975):.5}'
```

```
    )
```





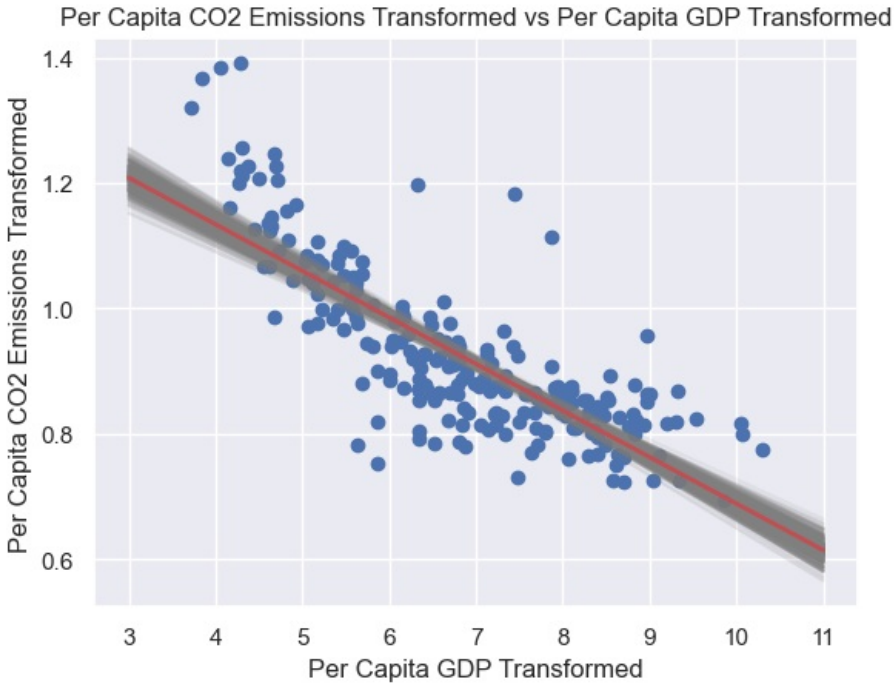

```
--- sigma hat ---
estimate:               4.6079
std error:              0.54397
median:                 4.6073
2.5 percentile:         3.6316
97.5 percentile:        5.7283
```

Transformed Models

In [225…

```
# Model 1 Transformed Model (Per capita GDP Transformed)

random.seed(90095)
n_boot = 500 # number of bootstraps

# initializations
boot_rmse = []
boot_b1 = []
boot_b2 = []
boot_sig2hat = [] # sigma^2 hat estimates (or std error ^2)
boot_r2 = [] # R^2 (no need to adjust)

per_capita_gdp_transformed_span_x = pd.DataFrame({'per_capita_gdp_transformed':np.linspace(3, 11, 200)}) # x spa

for iter in range(n_boot):
   # Sampling with replacement
   raw_data_cleaned_sample = raw_data_cleaned.sample(n=raw_data_cleaned.shape[0], replace=True) # sample method
   raw_data_cleaned_sample_ols = smf.ols('per_capita_co2_emissions_transformed ~ per_capita_gdp_transformed', da
```

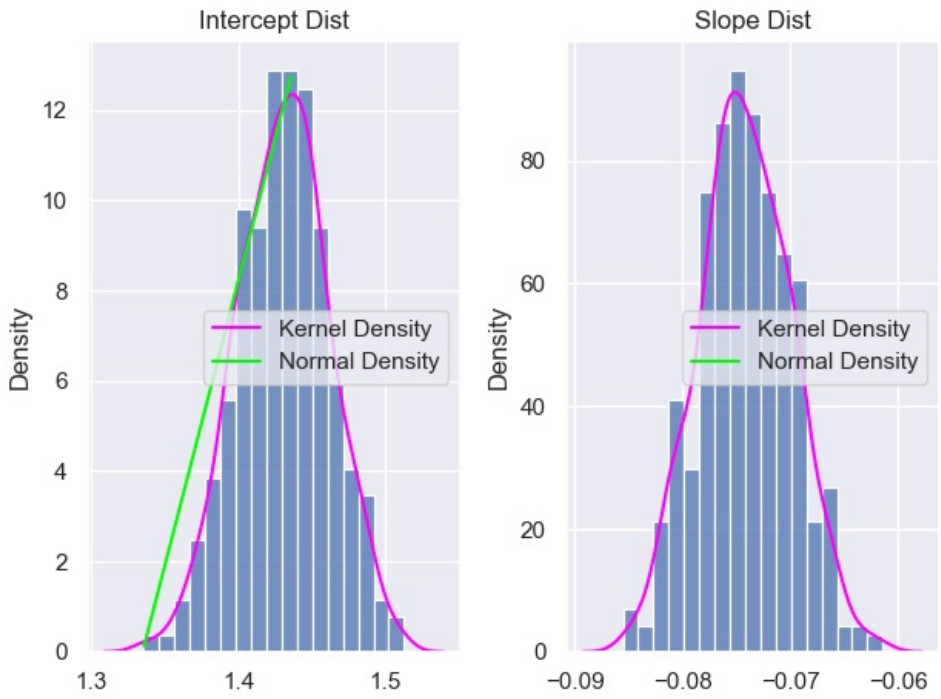
```
    # Append coefficients
    boot_b1.append(raw_data_cleaned_sample_ols.params[0])          # intercept
    boot_b2.append(raw_data_cleaned_sample_ols.params[1])          # slope
    # Performance measures
    boot_sig2hat.append(raw_data_cleaned_sample_ols.scale)         # scale = estimated sigma^2
    boot_rmse.append(np.sqrt(np.mean(raw_data_cleaned_sample_ols.resid ** 2))) # RMSE
    boot_r2.append(raw_data_cleaned_sample_ols.rsquared)           # R^2

    # Plot a greyed out line for this iteration
    raw_data_cleaned_sample_yhat = raw_data_cleaned_sample_ols.predict(per_capita_gdp_transformed_span_x)
    plt.plot(per_capita_gdp_transformed_span_x, raw_data_cleaned_sample_yhat, color='grey', alpha=0.1)

# Fit averaged model
raw_data_cleaned_ols_span_y = np.mean(boot_b1) + per_capita_gdp_transformed_span_x * np.mean(boot_b2)

# Plotting the fitted lines
plt.scatter( raw_data_cleaned['per_capita_gdp_transformed'], raw_data_cleaned['per_capita_co2_emissions_transfo
plt.plot(per_capita_gdp_transformed_span_x, raw_data_cleaned_ols_span_y, color='r', linewidth=2)
plt.xlabel('Per Capita GDP Transformed')
plt.ylabel('Per Capita CO2 Emissions Transformed')
plt.title('Per Capita CO2 Emissions Transformed vs Per Capita GDP Transformed')
plt.show()
```

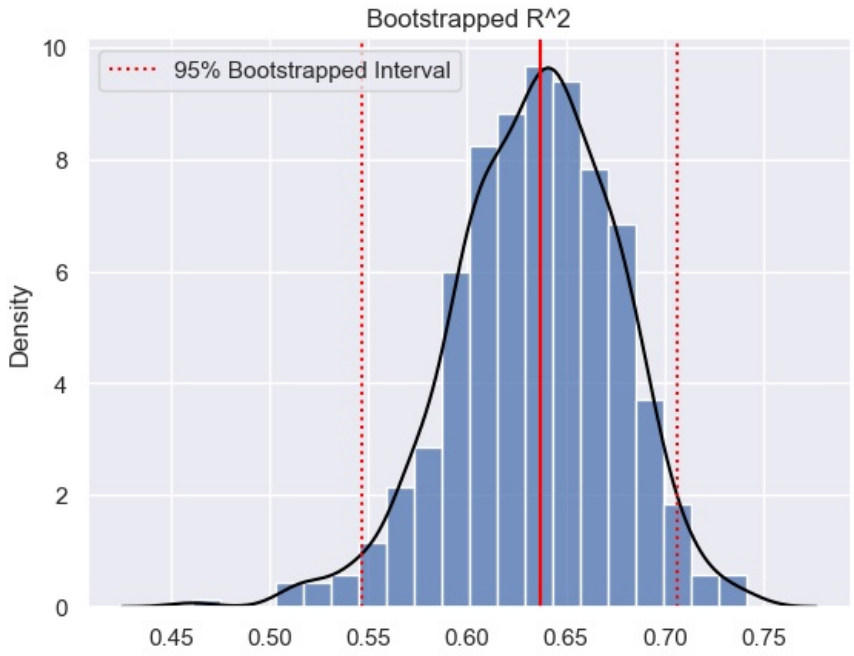


In [226…

```
# Next the distributions of the estimates

# For plotting normal PDFs
b1_range = np.arange(np.min(boot_b1), np.max(boot_b1), 0.1)
b2_range = np.arange(np.min(boot_b2), np.max(boot_b2), 0.1)

# Plotting beta estimates
fig6, ax6 = plt.subplots(1, 2)
sns.histplot(boot_b1, stat='density', kde=False, ax=ax6[0])
sns.kdeplot(boot_b1, color='fuchsia', ax=ax6[0], label='Kernel Density')
sns.lineplot(x=b1_range, y=norm.pdf(b1_range, np.mean(boot_b1), np.std(boot_b1)), color='lime',ax=ax6[0], label=
ax6[0].set_title('Intercept Dist')
sns.histplot(boot_b2, stat='density', kde=False, ax=ax6[1])
sns.kdeplot(boot_b2, color='fuchsia', ax=ax6[1], label='Kernel Density')
sns.lineplot(x=b2_range, y=norm.pdf(b2_range, np.mean(boot_b2), np.std(boot_b2)), color='lime',ax=ax6[1], label=
ax6[1].set_title('Slope Dist')
plt.tight_layout()
plt.show()

# Bootstrap for distribution of R^2 (coefficient of determination)
sns.histplot(boot_r2, stat='density', kde=False)
sns.kdeplot(boot_r2, color='black')
plt.axvline(x=np.quantile(boot_r2, q=0.025), color='red', linestyle=':', label='95% Bootstrapped Interval')
plt.axvline(x=np.quantile(boot_r2, q=0.5), color='red')
plt.axvline(x=np.quantile(boot_r2, q=0.975), color='red', linestyle=':')
plt.title('Bootstrapped R^2')
plt.legend()
plt.show()

# Bootstrap for estimated residual standard deviation
boot_stdhat = np.sqrt(boot_sig2hat)
print('--- sigma hat ---\n' +
```

```
      f'estimate:\t\t{boot_stdhat.mean():.5}\n' +
      f'std error:\t\t{boot_stdhat.std():.5}\n' +
      f'median:\t\t\t{np.quantile(boot_stdhat, q=0.5):.5}\n' +
      f'2.5 percentile:\t\t{np.quantile(boot_stdhat, q=0.025):.5}\n' +
      f'97.5 percentile:\t{np.quantile(boot_stdhat, q=0.975):.5}'
      )
```



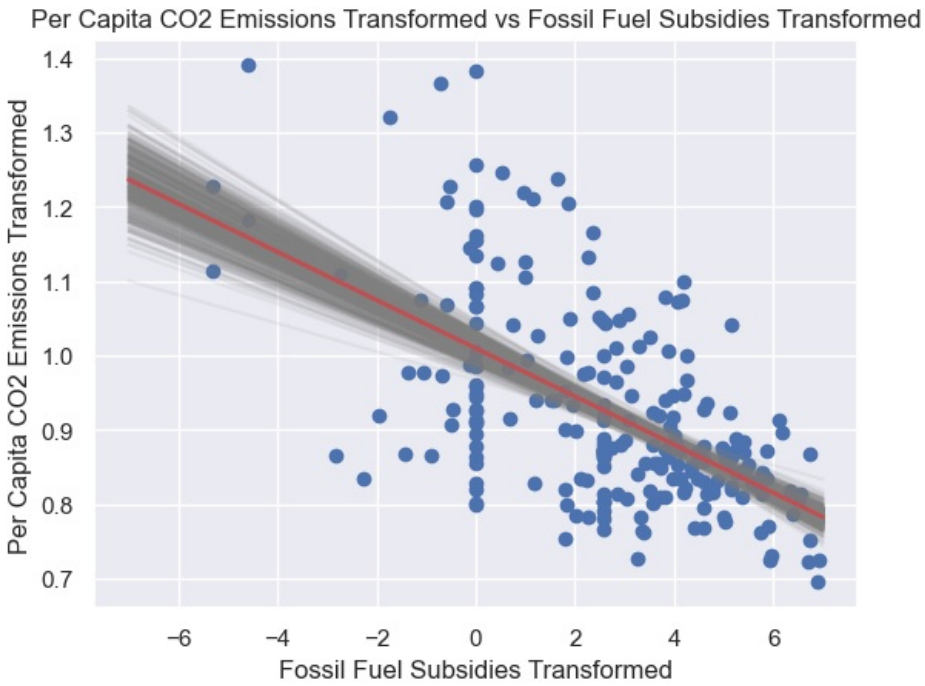



```
--- sigma hat ---
estimate:               0.080374
std error:              0.005258
median:                 0.080113
2.5 percentile:         0.070489
97.5 percentile:        0.091317
```

In [227…

```
# Model 2 Transformed Model (Fossil Fuel Subsidies Transformed)

random.seed(90095)
n_boot = 500 # number of bootstraps

# initializations
boot_rmse = []
boot_b1 = []
boot_b2 = []
boot_sig2hat = [] # sigma^2 hat estimates (or std error ^2)
boot_r2 = [] # R^2 (no need to adjust)

fossil_fuel_subsidies_transformed_span_x = pd.DataFrame({'fossil_fuel_subsidies_transformed':np.linspace(-7, 7,

for iter in range(n_boot):
   # Sampling with replacement
```

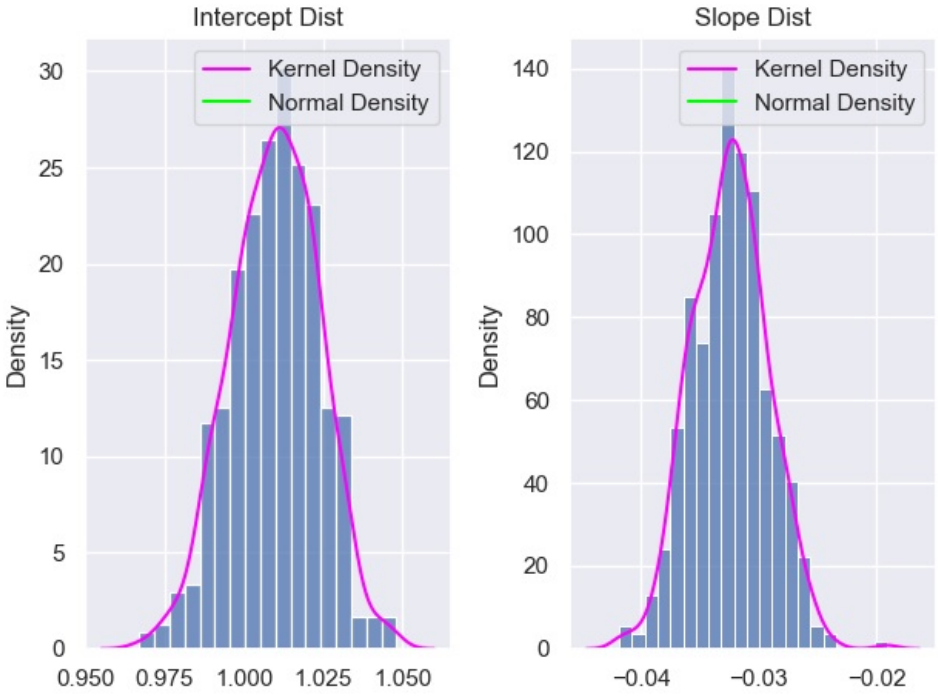
```
    raw_data_cleaned_sample = raw_data_cleaned.sample(n=raw_data_cleaned.shape[0], replace=True) # sample method
    raw_data_cleaned_sample_ols = smf.ols('per_capita_co2_emissions_transformed ~ fossil_fuel_subsidies_transfor

    # Append coefficients
    boot_b1.append(raw_data_cleaned_sample_ols.params[0])          # intercept
    boot_b2.append(raw_data_cleaned_sample_ols.params[1])          # slope
    # Performance measures
    boot_sig2hat.append(raw_data_cleaned_sample_ols.scale)         # scale = estimated sigma^2
    boot_rmse.append(np.sqrt(np.mean(raw_data_cleaned_sample_ols.resid ** 2))) # RMSE
    boot_r2.append(raw_data_cleaned_sample_ols.rsquared)           # R^2

    # Plot a greyed out line for this iteration
    raw_data_cleaned_sample_yhat = raw_data_cleaned_sample_ols.predict(fossil_fuel_subsidies_transformed_span_x)
    plt.plot(fossil_fuel_subsidies_transformed_span_x, raw_data_cleaned_sample_yhat, color='grey', alpha=0.1)

# Fit averaged model
raw_data_cleaned_ols_span_y = np.mean(boot_b1) + fossil_fuel_subsidies_transformed_span_x * np.mean(boot_b2)

# Plotting the fitted lines
plt.scatter( raw_data_cleaned['fossil_fuel_subsidies_transformed'], raw_data_cleaned['per_capita_co2_emissions_
plt.plot(fossil_fuel_subsidies_transformed_span_x, raw_data_cleaned_ols_span_y, color='r', linewidth=2)
plt.xlabel('Fossil Fuel Subsidies Transformed')
plt.ylabel('Per Capita CO2 Emissions Transformed')
plt.title('Per Capita CO2 Emissions Transformed vs Fossil Fuel Subsidies Transformed')
plt.show()
```

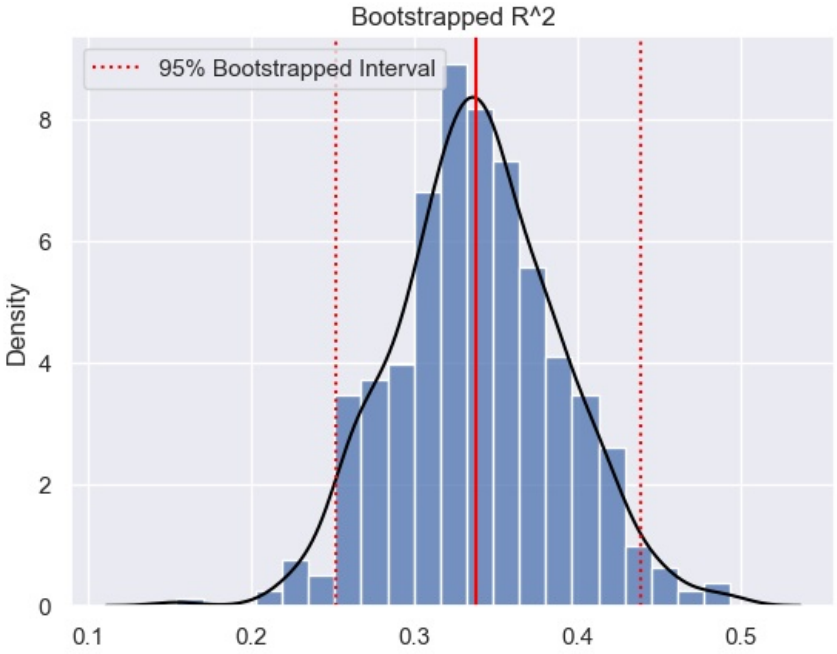


In [228…

```
# Next the distributions of the estimates

# For plotting normal PDFs
b1_range = np.arange(np.min(boot_b1), np.max(boot_b1), 0.1)
b2_range = np.arange(np.min(boot_b2), np.max(boot_b2), 0.1)

# Plotting beta estimates
fig6, ax6 = plt.subplots(1, 2)
sns.histplot(boot_b1, stat='density', kde=False, ax=ax6[0])
sns.kdeplot(boot_b1, color='fuchsia', ax=ax6[0], label='Kernel Density')
sns.lineplot(x=b1_range, y=norm.pdf(b1_range, np.mean(boot_b1), np.std(boot_b1)), color='lime',ax=ax6[0], label=
ax6[0].set_title('Intercept Dist')
sns.histplot(boot_b2, stat='density', kde=False, ax=ax6[1])
sns.kdeplot(boot_b2, color='fuchsia', ax=ax6[1], label='Kernel Density')
sns.lineplot(x=b2_range, y=norm.pdf(b2_range, np.mean(boot_b2), np.std(boot_b2)), color='lime',ax=ax6[1], label=
ax6[1].set_title('Slope Dist')
plt.tight_layout()
plt.show()

# Bootstrap for distribution of R^2 (coefficient of determination)
sns.histplot(boot_r2, stat='density', kde=False)
sns.kdeplot(boot_r2, color='black')
plt.axvline(x=np.quantile(boot_r2, q=0.025), color='red', linestyle=':', label='95% Bootstrapped Interval')
plt.axvline(x=np.quantile(boot_r2, q=0.5), color='red')
plt.axvline(x=np.quantile(boot_r2, q=0.975), color='red', linestyle=':')
plt.title('Bootstrapped R^2')
plt.legend()
plt.show()

# Bootstrap for estimated residual standard deviation
```

```
boot_stdhat = np.sqrt(boot_sig2hat)
print('--- sigma hat ---\n' +
      f'estimate:\t\t{boot_stdhat.mean():.5}\n' +
      f'std error:\t\t{boot_stdhat.std():.5}\n' +
      f'median:\t\t\t{np.quantile(boot_stdhat, q=0.5):.5}\n' +
      f'2.5 percentile:\t\t{np.quantile(boot_stdhat, q=0.025):.5}\n' +
      f'97.5 percentile:\t{np.quantile(boot_stdhat, q=0.975):.5}'
      )
```



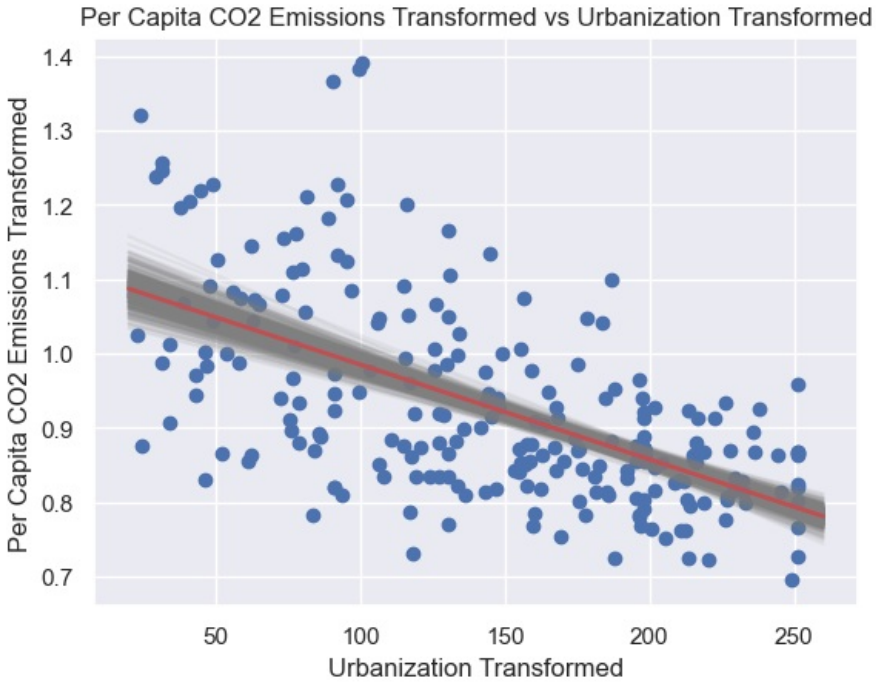



```
--- sigma hat ---
estimate:               0.10823
std error:              0.005698
median:                 0.10819
2.5 percentile:         0.097631
97.5 percentile:        0.11898
```

In [229…

```
# Model 3 Transformed Model (Urbanization Transformed)

random.seed(90095)
n_boot = 500 # number of bootstraps

# initializations
boot_rmse = []
boot_b1 = []
boot_b2 = []
boot_sig2hat = [] # sigma^2 hat estimates (or std error ^2)
boot_r2 = [] # R^2 (no need to adjust)

Urbanization_transformed_span_x = pd.DataFrame({'Urbanization_transformed':np.linspace(20, 260, 200)})
```

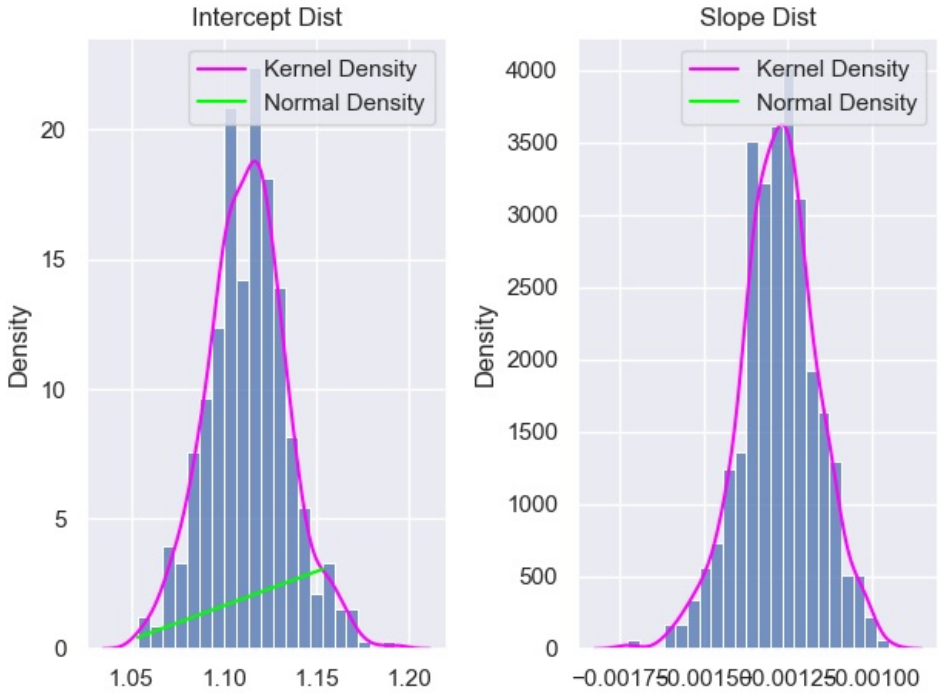
```
for iter in range(n_boot):
    # Sampling with replacement
    raw_data_cleaned_sample = raw_data_cleaned.sample(n=raw_data_cleaned.shape[0], replace=True) # sample method
    raw_data_cleaned_sample_ols = smf.ols('per_capita_co2_emissions_transformed ~ Urbanization_transformed', data

    # Append coefficients
    boot_b1.append(raw_data_cleaned_sample_ols.params[0])          # intercept
    boot_b2.append(raw_data_cleaned_sample_ols.params[1])          # slope
    # Performance measures
    boot_sig2hat.append(raw_data_cleaned_sample_ols.scale)         # scale = estimated sigma^2
    boot_rmse.append(np.sqrt(np.mean(raw_data_cleaned_sample_ols.resid ** 2))) # RMSE
    boot_r2.append(raw_data_cleaned_sample_ols.rsquared)           # R^2

    # Plot a greyed out line for this iteration
    raw_data_cleaned_sample_yhat = raw_data_cleaned_sample_ols.predict(Urbanization_transformed_span_x)
    plt.plot(Urbanization_transformed_span_x, raw_data_cleaned_sample_yhat, color='grey', alpha=0.1)

# Fit averaged model
raw_data_cleaned_ols_span_y = np.mean(boot_b1) + Urbanization_transformed_span_x * np.mean(boot_b2)

# Plotting the fitted lines
plt.scatter( raw_data_cleaned['Urbanization_transformed'], raw_data_cleaned['per_capita_co2_emissions_transforme
plt.plot(Urbanization_transformed_span_x, raw_data_cleaned_ols_span_y, color='r', linewidth=2)
plt.xlabel('Urbanization Transformed')
plt.ylabel('Per Capita CO2 Emissions Transformed')
plt.title('Per Capita CO2 Emissions Transformed vs Urbanization Transformed')
plt.show()
```

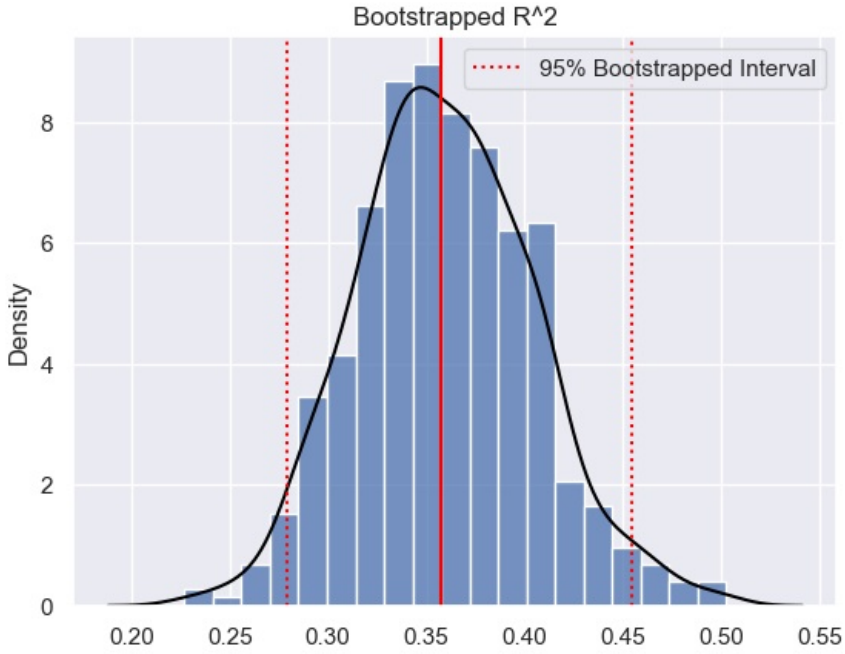


In [230…

```
# Next the distributions of the estimates

# For plotting normal PDFs
b1_range = np.arange(np.min(boot_b1), np.max(boot_b1), 0.1)
b2_range = np.arange(np.min(boot_b2), np.max(boot_b2), 0.1)

# Plotting beta estimates
fig6, ax6 = plt.subplots(1, 2)
sns.histplot(boot_b1, stat='density', kde=False, ax=ax6[0])
sns.kdeplot(boot_b1, color='fuchsia', ax=ax6[0], label='Kernel Density')
sns.lineplot(x=b1_range, y=norm.pdf(b1_range, np.mean(boot_b1), np.std(boot_b1)), color='lime',ax=ax6[0], label=
ax6[0].set_title('Intercept Dist')
sns.histplot(boot_b2, stat='density', kde=False, ax=ax6[1])
sns.kdeplot(boot_b2, color='fuchsia', ax=ax6[1], label='Kernel Density')
sns.lineplot(x=b2_range, y=norm.pdf(b2_range, np.mean(boot_b2), np.std(boot_b2)), color='lime',ax=ax6[1], label=
ax6[1].set_title('Slope Dist')
plt.tight_layout()
plt.show()

# Bootstrap for distribution of R^2 (coefficient of determination)
sns.histplot(boot_r2, stat='density', kde=False)
sns.kdeplot(boot_r2, color='black')
plt.axvline(x=np.quantile(boot_r2, q=0.025), color='red', linestyle=':', label='95% Bootstrapped Interval')
plt.axvline(x=np.quantile(boot_r2, q=0.5), color='red')
plt.axvline(x=np.quantile(boot_r2, q=0.975), color='red', linestyle=':')
plt.title('Bootstrapped R^2')
plt.legend()
plt.show()
```

```
# Bootstrap for estimated residual standard deviation
boot_stdhat = np.sqrt(boot_sig2hat)
print('--- sigma hat ---\n' +
      f'estimate:\t\t{boot_stdhat.mean():.5}\n' +
      f'std error:\t\t{boot_stdhat.std():.5}\n' +
      f'median:\t\t\t{np.quantile(boot_stdhat, q=0.5):.5}\n' +
      f'2.5 percentile:\t\t{np.quantile(boot_stdhat, q=0.025):.5}\n' +
      f'97.5 percentile:\t{np.quantile(boot_stdhat, q=0.975):.5}'
      )
```






```
--- sigma hat ---
estimate:               0.10657
std error:              0.0062562
median:                 0.10635
2.5 percentile:         0.094838
97.5 percentile:        0.1186
```

Based on bootstrapping estimates, we can see good robustness of all the estimates especially in the transformed models, especially since the kernel densities, very closely resemble a normal density distribution and thus, CLT holds.

Additionally, in all the 3 transformed models, the sigma hat value or the estimated standard residuals are significantly lower than their untransformed model versions, indicating that the transformed version model estimates are significantly better at predicting per capita co2 emissions versus the LS version

Additionally, this also shows that the best model in terms of lowest estimated standard residual is the transformed version of regressing per capita GDP to predict per capita CO2 emissions

K- Cross Validation

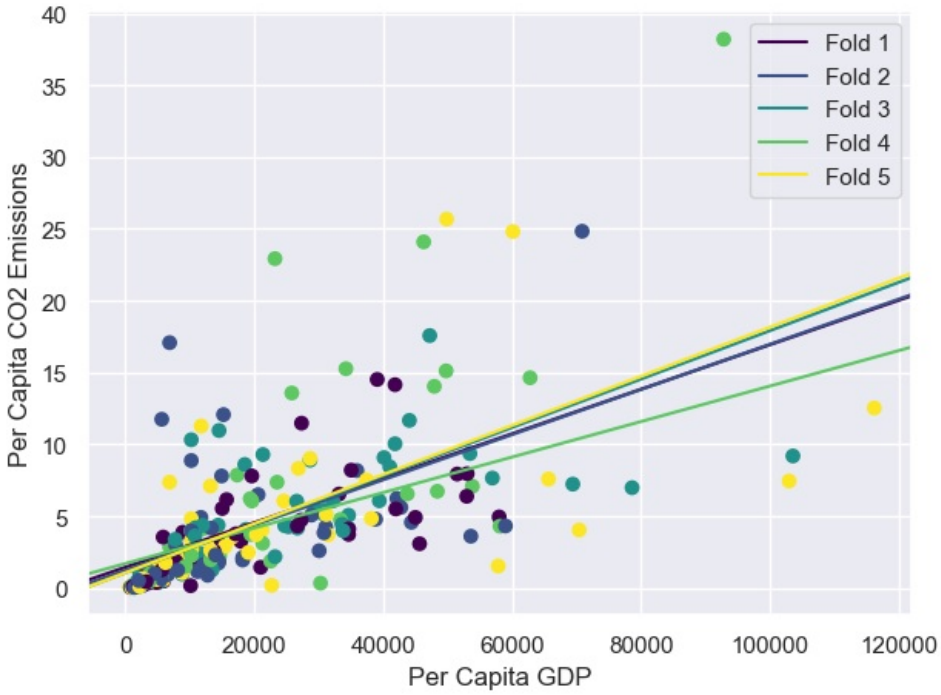
To evaluate our model's performance, we used k-fold cross-validation, a technique that divides the dataset into k equal parts (folds) to ensure a robust performance assessment. We specifically applied 5-fold cross-validation, where the data is split into 5 subsets. In each iteration, the model is trained on 4 folds and tested on the remaining 1, rotating so that each fold serves as the test set once. This method allows us to obtain an average performance measure across all folds, reducing the impact of any single data split and improving the reliability of the evaluation. We computed the Root Mean Squared Error (RMSE) for each fold and then averaged these RMSEs to assess the model's overall predictive accuracy. By using cross-validation, we get a realistic estimate of how the model would perform on unseen data, as it reduces the risk of overfitting and provides insights into model stability and generalizability.

In [231…
```
from sklearn.model_selection import train_test_split

# Split the raw_cleaned_data dataset into training and test sets
energy_train, energy_test = train_test_split(raw_data_cleaned, test_size=0.25, random_state=90095)

# Print the shape of the train and test sets to see sample sizes
print(energy_train.shape, energy_test.shape)
```

```
(177, 13) (59, 13)
```

In [232…
```
# Model 1 Untransformed (Per Capita GDP)
from sklearn.model_selection import KFold, train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

# Assuming 'raw_data_cleaned' is your dataset
# X will be the independent variable (e.g., per_capita_gdp)
# y will be the dependent variable (e.g., per_capita_co2_emissions)

X = raw_data_cleaned['per_capita_gdp'].to_numpy().reshape(-1,1)  # X-variable (independent)
y = raw_data_cleaned['per_capita_co2_emissions'].to_numpy()  # y-variable (dependent)

# Initialize KFold
kf = KFold(n_splits=5, shuffle=True, random_state=10)

# Lists to store fold results
fold_assignments = np.zeros(X.shape[0])
rmse_list = []
intercept_list = []
coefficient_list = []

# K-fold Cross-Validation
for fold, (train_index, test_index) in enumerate(kf.split(X)):
    fold_assignments[test_index] = fold  # Record the fold assignment for each test sample

    # Split into train and test samples for this fold
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train the model
    model = LinearRegression().fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)

    # Compute the Root Mean Squared Error (RMSE)
    fold_rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))

    # Append the RMSE, intercept, and coefficient for this fold
    rmse_list.append(fold_rmse)
    intercept_list.append(model.intercept_)
    coefficient_list.append(model.coef_[0])  # Only one coefficient for simple regression

# Add the fold assignments to the original DataFrame for visualization
raw_data_cleaned_folds = raw_data_cleaned.copy()
raw_data_cleaned_folds['fold'] = fold_assignments

# Plot the true values (colored by fold) and the regression lines for each fold
fold_colors = ['#440154FF', '#3B528BFF', '#21908CFF', '#5DC863FF', '#FDE725FF']
plt.scatter(x=raw_data_cleaned_folds['per_capita_gdp'],
            y=raw_data_cleaned_folds['per_capita_co2_emissions'],
            c=raw_data_cleaned_folds['fold'], cmap='viridis')

# Plot regression lines for each fold
for i, params in enumerate(zip(intercept_list, coefficient_list)):
    plt.axline(xy1=(0, params[0]), slope=params[1], color=fold_colors[i], label=f'Fold {i+1}')

plt.ylabel('Per Capita CO2 Emissions')
plt.xlabel('Per Capita GDP')
plt.legend()
plt.tight_layout()
```

```
# Show the plot
plt.show()

# Print the Cross-Validation results (mean RMSE)
print(f'CV RMSE = {np.mean(rmse_list):.6f}')
```

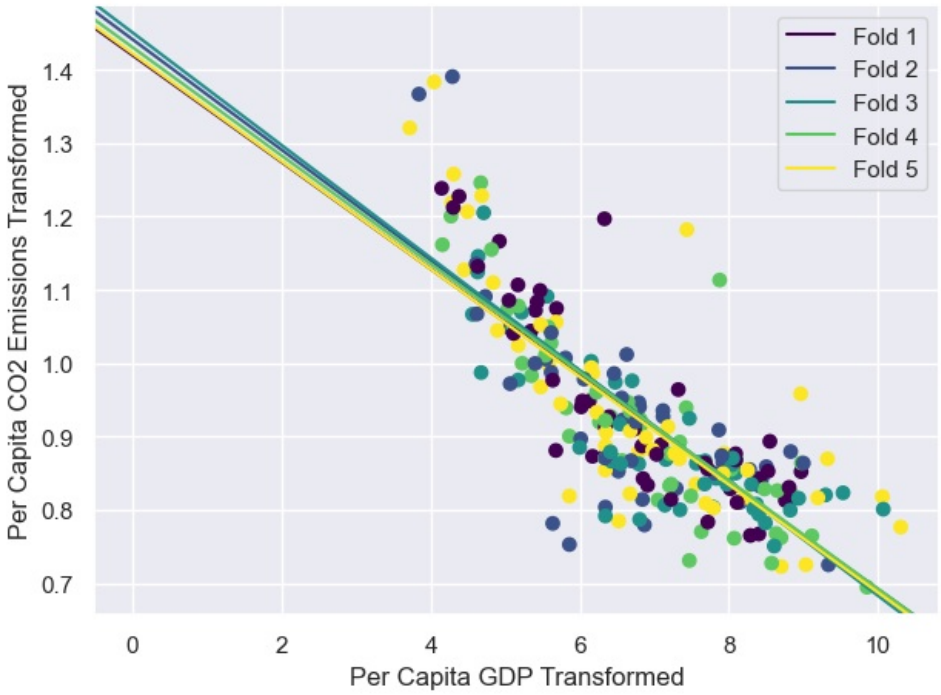



```
CV RMSE = 4.102897
```

In [233…

```
# Model 1 Transformed (Per Capita GDP Transformed)
from sklearn.model_selection import KFold, train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

# Assuming 'raw_data_cleaned' is your dataset
# X will be the independent variable (e.g., per_capita_gdp)
# y will be the dependent variable (e.g., per_capita_co2_emissions)

X = raw_data_cleaned['per_capita_gdp_transformed'].to_numpy().reshape(-1,1)  # X-variable (independent)
y = raw_data_cleaned['per_capita_co2_emissions_transformed'].to_numpy()  # y-variable (dependent)

# Initialize KFold
kf = KFold(n_splits=5, shuffle=True, random_state=10)

# Lists to store fold results
fold_assignments = np.zeros(X.shape[0])
rmse_list = []
intercept_list = []
coefficient_list = []

# K-fold Cross-Validation
for fold, (train_index, test_index) in enumerate(kf.split(X)):
    fold_assignments[test_index] = fold  # Record the fold assignment for each test sample

    # Split into train and test samples for this fold
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train the model
    model = LinearRegression().fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)

    # Compute the Root Mean Squared Error (RMSE)
    fold_rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))

    # Append the RMSE, intercept, and coefficient for this fold
    rmse_list.append(fold_rmse)
    intercept_list.append(model.intercept_)
    coefficient_list.append(model.coef_[0])  # Only one coefficient for simple regression

# Add the fold assignments to the original DataFrame for visualization
raw_data_cleaned_folds = raw_data_cleaned.copy()
raw_data_cleaned_folds['fold'] = fold_assignments
```

```
# Plot the true values (colored by fold) and the regression lines for each fold
fold_colors = ['#440154FF', '#3B528BFF', '#21908CFF', '#5DC863FF', '#FDE725FF']
plt.scatter(x=raw_data_cleaned_folds['per_capita_gdp_transformed'],
            y=raw_data_cleaned_folds['per_capita_co2_emissions_transformed'],
            c=raw_data_cleaned_folds['fold'], cmap='viridis')

# Plot regression lines for each fold
for i, params in enumerate(zip(intercept_list, coefficient_list)):
    plt.axline(xy1=(0, params[0]), slope=params[1], color=fold_colors[i], label=f'Fold {i+1}')

plt.ylabel('Per Capita CO2 Emissions Transformed')
plt.xlabel('Per Capita GDP Transformed')
plt.legend()
plt.tight_layout()

# Show the plot
plt.show()

# Print the Cross-Validation results (mean RMSE)
print(f'CV RMSE = {np.mean(rmse_list):.6f}')
```

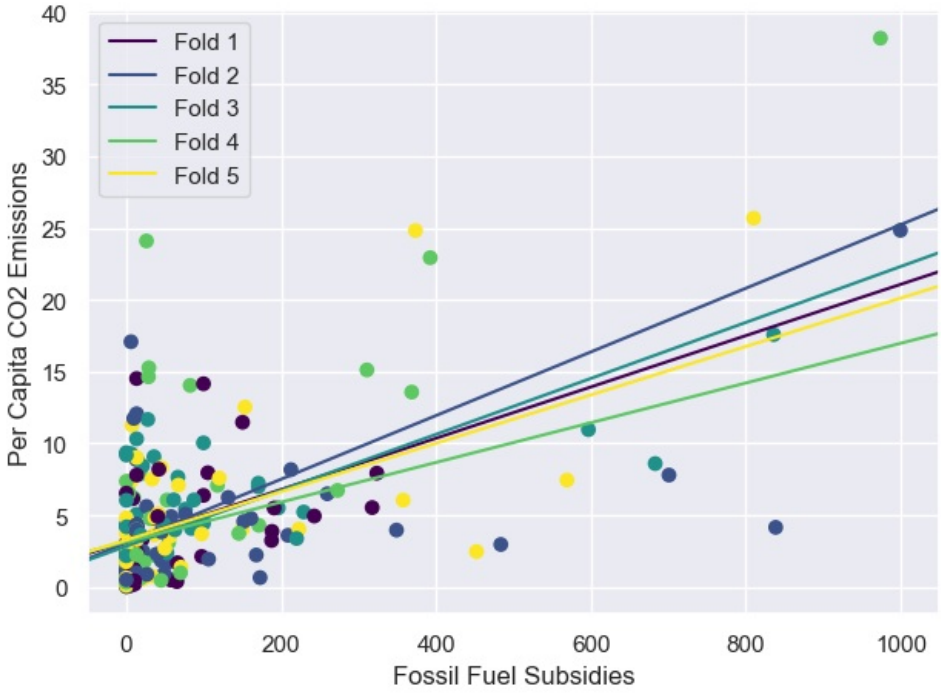



```
CV RMSE = 0.080082
```

In [234…

```
# Model 2 Untransformed (Fossil Fuel Subsidies)
from sklearn.model_selection import KFold, train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

# Assuming 'raw_data_cleaned' is your dataset
# X will be the independent variable (e.g., per_capita_gdp)
# y will be the dependent variable (e.g., per_capita_co2_emissions)

X = raw_data_cleaned['fossil_fuel_subsidies'].to_numpy().reshape(-1,1)  # X-variable (independent)
y = raw_data_cleaned['per_capita_co2_emissions'].to_numpy()  # y-variable (dependent)

# Initialize KFold
kf = KFold(n_splits=5, shuffle=True, random_state=10)

# Lists to store fold results
fold_assignments = np.zeros(X.shape[0])
rmse_list = []
intercept_list = []
coefficient_list = []

# K-fold Cross-Validation
for fold, (train_index, test_index) in enumerate(kf.split(X)):
    fold_assignments[test_index] = fold  # Record the fold assignment for each test sample

    # Split into train and test samples for this fold
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train the model
    model = LinearRegression().fit(X_train, y_train)

    # Predict on the test set
```

```
    y_pred = model.predict(X_test)

    # Compute the Root Mean Squared Error (RMSE)
    fold_rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))

    # Append the RMSE, intercept, and coefficient for this fold
    rmse_list.append(fold_rmse)
    intercept_list.append(model.intercept_)
    coefficient_list.append(model.coef_[0])  # Only one coefficient for simple regression

# Add the fold assignments to the original DataFrame for visualization
raw_data_cleaned_folds = raw_data_cleaned.copy()
raw_data_cleaned_folds['fold'] = fold_assignments

# Plot the true values (colored by fold) and the regression lines for each fold
fold_colors = ['#440154FF', '#3B528BFF', '#21908CFF', '#5DC863FF', '#FDE725FF']
plt.scatter(x=raw_data_cleaned_folds['fossil_fuel_subsidies'],
            y=raw_data_cleaned_folds['per_capita_co2_emissions'],
            c=raw_data_cleaned_folds['fold'], cmap='viridis')

# Plot regression lines for each fold
for i, params in enumerate(zip(intercept_list, coefficient_list)):
    plt.axline(xy1=(0, params[0]), slope=params[1], color=fold_colors[i], label=f'Fold {i+1}')

plt.xlabel('Fossil Fuel Subsidies')
plt.ylabel('Per Capita CO2 Emissions')
plt.legend()
plt.tight_layout()

# Show the plot
plt.show()

# Print the Cross-Validation results (mean RMSE)
print(f'CV RMSE = {np.mean(rmse_list):.6f}')
```

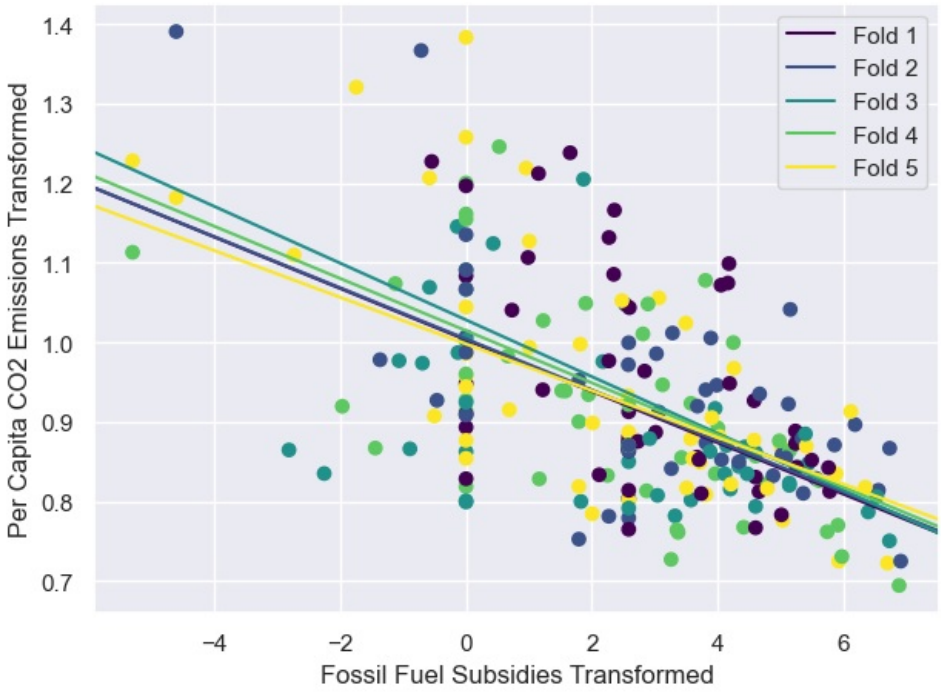



```
CV RMSE = 4.380548
```

In [235…

```
# Model 2 Transformed (Fossil Fuel Subsidies)
from sklearn.model_selection import KFold, train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

# Assuming 'raw_data_cleaned' is your dataset
# X will be the independent variable (e.g., per_capita_gdp)
# y will be the dependent variable (e.g., per_capita_co2_emissions)

X = raw_data_cleaned['fossil_fuel_subsidies_transformed'].to_numpy().reshape(-1,1)  # X-variable (independent)
y = raw_data_cleaned['per_capita_co2_emissions_transformed'].to_numpy()  # y-variable (dependent)

# Initialize KFold
kf = KFold(n_splits=5, shuffle=True, random_state=10)

# Lists to store fold results
fold_assignments = np.zeros(X.shape[0])
rmse_list = []
intercept_list = []
```

```
coefficient_list = []

# K-fold Cross-Validation
for fold, (train_index, test_index) in enumerate(kf.split(X)):
    fold_assignments[test_index] = fold  # Record the fold assignment for each test sample

    # Split into train and test samples for this fold
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train the model
    model = LinearRegression().fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)

    # Compute the Root Mean Squared Error (RMSE)
    fold_rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))

    # Append the RMSE, intercept, and coefficient for this fold
    rmse_list.append(fold_rmse)
    intercept_list.append(model.intercept_)
    coefficient_list.append(model.coef_[0])  # Only one coefficient for simple regression

# Add the fold assignments to the original DataFrame for visualization
raw_data_cleaned_folds = raw_data_cleaned.copy()
raw_data_cleaned_folds['fold'] = fold_assignments

# Plot the true values (colored by fold) and the regression lines for each fold
fold_colors = ['#440154FF', '#3B528BFF', '#21908CFF', '#5DC863FF', '#FDE725FF']
plt.scatter(x=raw_data_cleaned_folds['fossil_fuel_subsidies_transformed'],
            y=raw_data_cleaned_folds['per_capita_co2_emissions_transformed'],
            c=raw_data_cleaned_folds['fold'], cmap='viridis')

# Plot regression lines for each fold
for i, params in enumerate(zip(intercept_list, coefficient_list)):
    plt.axline(xy1=(0, params[0]), slope=params[1], color=fold_colors[i], label=f'Fold {i+1}')

plt.ylabel('Per Capita CO2 Emissions Transformed')
plt.xlabel('Fossil Fuel Subsidies Transformed')
plt.legend()
plt.tight_layout()

# Show the plot
plt.show()

# Print the Cross-Validation results (mean RMSE)
print(f'CV RMSE = {np.mean(rmse_list):.6f}')
```

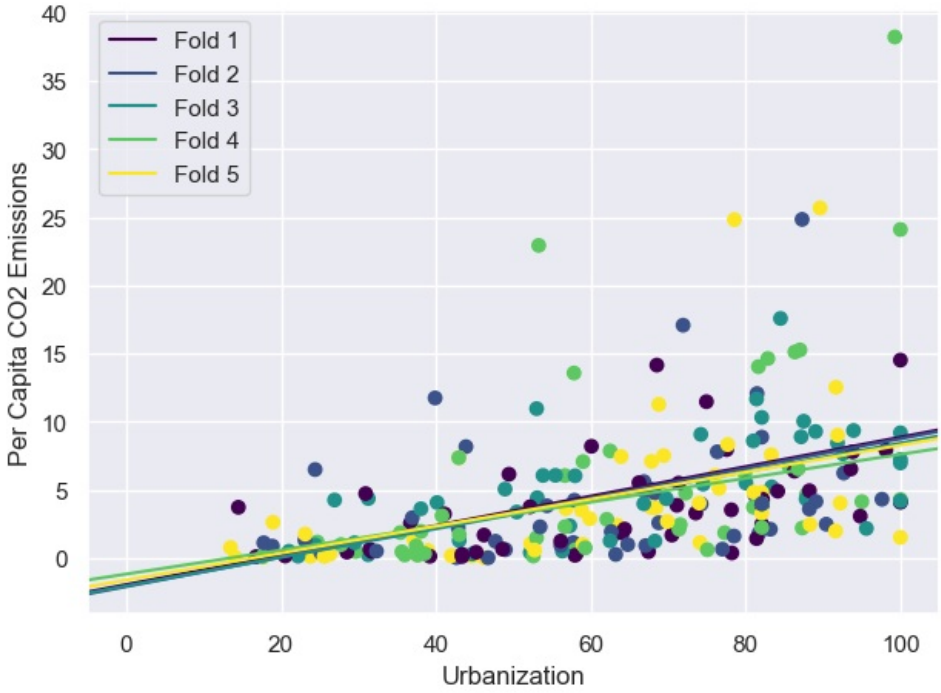



```
CV RMSE = 0.110900
```

In [236…

```
# Model 3 Untransformed (Urbanization)
from sklearn.model_selection import KFold, train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

```
# Assuming 'raw_data_cleaned' is your dataset
# X will be the independent variable (e.g., per_capita_gdp)
# y will be the dependent variable (e.g., per_capita_co2_emissions)

X = raw_data_cleaned['Urbanization'].to_numpy().reshape(-1,1)  # X-variable (independent)
y = raw_data_cleaned['per_capita_co2_emissions'].to_numpy()  # y-variable (dependent)

# Initialize KFold
kf = KFold(n_splits=5, shuffle=True, random_state=10)

# Lists to store fold results
fold_assignments = np.zeros(X.shape[0])
rmse_list = []
intercept_list = []
coefficient_list = []

# K-fold Cross-Validation
for fold, (train_index, test_index) in enumerate(kf.split(X)):
    fold_assignments[test_index] = fold  # Record the fold assignment for each test sample

    # Split into train and test samples for this fold
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train the model
    model = LinearRegression().fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)

    # Compute the Root Mean Squared Error (RMSE)
    fold_rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))

    # Append the RMSE, intercept, and coefficient for this fold
    rmse_list.append(fold_rmse)
    intercept_list.append(model.intercept_)
    coefficient_list.append(model.coef_[0])  # Only one coefficient for simple regression

# Add the fold assignments to the original DataFrame for visualization
raw_data_cleaned_folds = raw_data_cleaned.copy()
raw_data_cleaned_folds['fold'] = fold_assignments

# Plot the true values (colored by fold) and the regression lines for each fold
fold_colors = ['#440154FF', '#3B528BFF', '#21908CFF', '#5DC863FF', '#FDE725FF']
plt.scatter(x=raw_data_cleaned_folds['Urbanization'],
            y=raw_data_cleaned_folds['per_capita_co2_emissions'],
            c=raw_data_cleaned_folds['fold'], cmap='viridis')

# Plot regression lines for each fold
for i, params in enumerate(zip(intercept_list, coefficient_list)):
    plt.axline(xy1=(0, params[0]), slope=params[1], color=fold_colors[i], label=f'Fold {i+1}')

plt.xlabel('Urbanization')
plt.ylabel('Per Capita CO2 Emissions')
plt.legend()
plt.tight_layout()

# Show the plot
plt.show()

# Print the Cross-Validation results (mean RMSE)
print(f'CV RMSE = {np.mean(rmse_list):.6f}')
```

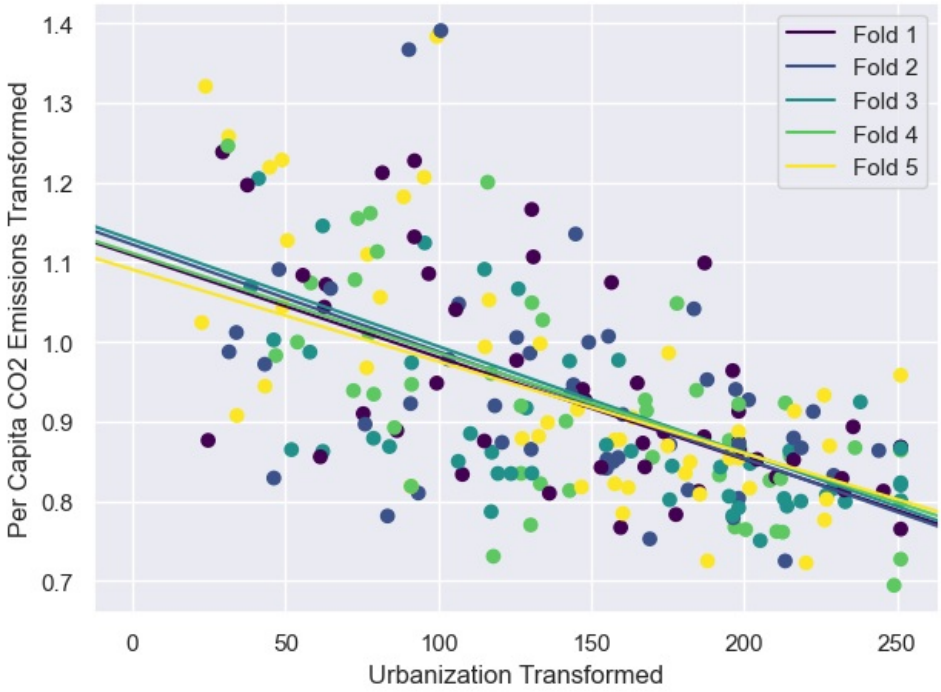



```
CV RMSE = 4.479131
```

In [157…

```python
# Model 3 Transformed (Urbanization)
from sklearn.model_selection import KFold, train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

# Assuming 'raw_data_cleaned' is your dataset
# X will be the independent variable (e.g., per_capita_gdp)
# y will be the dependent variable (e.g., per_capita_co2_emissions)

X = raw_data_cleaned['Urbanization_transformed'].to_numpy().reshape(-1,1)  # X-variable (independent)
y = raw_data_cleaned['per_capita_co2_emissions_transformed'].to_numpy()  # y-variable (dependent)

# Initialize KFold
kf = KFold(n_splits=5, shuffle=True, random_state=10)

# Lists to store fold results
fold_assignments = np.zeros(X.shape[0])
rmse_list = []
intercept_list = []
coefficient_list = []

# K-fold Cross-Validation
for fold, (train_index, test_index) in enumerate(kf.split(X)):
    fold_assignments[test_index] = fold  # Record the fold assignment for each test sample

    # Split into train and test samples for this fold
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train the model
    model = LinearRegression().fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)

    # Compute the Root Mean Squared Error (RMSE)
    fold_rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))

    # Append the RMSE, intercept, and coefficient for this fold
    rmse_list.append(fold_rmse)
    intercept_list.append(model.intercept_)
    coefficient_list.append(model.coef_[0])  # Only one coefficient for simple regression

# Add the fold assignments to the original DataFrame for visualization
raw_data_cleaned_folds = raw_data_cleaned.copy()
raw_data_cleaned_folds['fold'] = fold_assignments

# Plot the true values (colored by fold) and the regression lines for each fold
fold_colors = ['#440154FF', '#3B528BFF', '#21908CFF', '#5DC863FF', '#FDE725FF']
plt.scatter(x=raw_data_cleaned_folds['Urbanization_transformed'],
            y=raw_data_cleaned_folds['per_capita_co2_emissions_transformed'],
            c=raw_data_cleaned_folds['fold'], cmap='viridis')

# Plot regression lines for each fold
```

```
for i, params in enumerate(zip(intercept_list, coefficient_list)):
    plt.axline(xy1=(0, params[0]), slope=params[1], color=fold_colors[i], label=f'Fold {i+1}')

plt.xlabel('Urbanization Transformed')
plt.ylabel('Per Capita CO2 Emissions Transformed')
plt.legend()
plt.tight_layout()

# Show the plot
plt.show()

# Print the Cross-Validation results (mean RMSE)
print(f'CV RMSE = {np.mean(rmse_list):.6f}')
```




```
CV RMSE = 0.107786
```

The significant difference in RMSE values—4.102897 for the original variables versus 0.080082 for the Box-Cox transformed variables—highlights the effectiveness of the transformation. The Box-Cox transformation stabilizes variance and normalizes the distribution, allowing for a better linear relationship between Per Capita CO2 emissions and Per Capita GDP. By reducing non-linearity and mitigating the impact of outliers, the transformation enhances the model's predictive accuracy. Consequently, the model trained on the transformed data demonstrates a much better fit, leading to a substantially lower RMSE. This improvement signifies that using Box-Cox transformed variables provides more reliable insights into the relationship between these key economic indicators.

The RMSE results reveal a substantial difference: 4.380548 for the original variables and 0.110900 for the Box-Cox transformed variables. This dramatic decrease in RMSE suggests that the Box-Cox transformation significantly improved the model's predictive accuracy. By normalizing the distributions of both Per Capita CO2 emissions and fossil fuel subsidies, the transformation enhances the linearity of their relationship. Additionally, the transformation reduces the influence of outliers and stabilizes variance, leading to a better model fit. This improvement indicates that the Box-Cox transformed data aligns more closely with the assumptions of linear regression, resulting in more reliable predictions.

The cross-validation results show a significant difference in RMSE values: 4.479131 for the original variables compared to 0.107786 for the Box-Cox transformed variables. This dramatic reduction in RMSE indicates that the Box-Cox transformation greatly enhanced the model's predictive accuracy. By transforming both Per Capita CO2 emissions and Urbanization, the transformation normalizes their distributions and stabilizes variance, improving the linear relationship between the variables. Additionally, it helps mitigate the effects of outliers, which can adversely impact model performance. As a result, the transformed data aligns better with the assumptions of linear regression, leading to more reliable predictions.

Conclusion

Based on the Estimated standard residuals, comparing model fits and CV RMSE estimates across models, we can conclude that the best model among the ones tested is regressing per capita gdp to the power of 0.2 to predict per capita CO2 emissions to the power of -0.1

In terms of economic significance, this clearly shows that countries with high per capita GDP are more likely to have higher per capita CO2 emissions. This could be caused by several factors, including higher industrialization, richer countries being more likely to use higher transportation coverage, including higher usage of resources causing CO2 emissions. For implications of policy, this could mean, that richer countries should be potentially the focus of global environmental policy and look inward, to understand how they can reduce CO2 emissions and their impact on global warming

Loading [MathJax]/extensions/Safe.js