DAV Assignment

Name : Panav Shah

Roll Number : 23B3323

Links to original Google Colab files

- Q1, Q2 : https://colab.research.google.com/drive/1go3ZNHivZT6uYTUN9luy9T-yaRoqV94w?usp=sharing
- Q3, Q4 : https://colab.research.google.com/drive/1L18iM5Ikv9_nUPDT9nUbdyQvi32cq5xQ?usp=sharing
- Q6 : https://colab.research.google.com/drive/1pN89cFP62pxCKvwYNxKW2sA0tXLIlg5-?usp=sharing
- Q7 : https://colab.research.google.com/drive/1qAgQQws0h3xzMxZT3PfRz5oAecLzyXVP?usp=sharing

Links to Google Colab files without the debugging outputs

- Q1, Q2 : https://colab.research.google.com/drive/13GkunQLL1eVTN2JLmBuxlCXDcY9XwmDJ?usp=sharing
- Q3, Q4 : https://colab.research.google.com/drive/1qEjCFV74wzXWvD064UJhImfvezKZTDpZ?usp=sharing
- Q6 : https://colab.research.google.com/drive/1pN89cFP62pxCKvwYNxKW2sA0tXLIlg5-?usp=sharing
- Q7 : https://colab.research.google.com/drive/1q5kq3tOLcAuICc7Z4eqQ-hWIur-7VpZR?usp=sharing

Link to my GitHub repository

https://github.com/PanavShah1/DAV_assignment

- Have uploaded all of the datasets (csv files) to the repository to access in the ipynb code

Predictions for Q3 & Q7

Q3 Edited Consumer Test Dataset - https://drive.google.com/file/d/1ZaSdC6wvsJ2Mk4rxwhQ-Z6T59pMyNbPk/view?usp=sharing

Q7 Edited Email Test Dataset - https://drive.google.com/file/d/1IBFMYjsE7XOsz3dxl0l5O3mJvRl6JXUb/view?usp=sharing

DAV Assignment Q1 and Q2

You are the head of the sales and marketing division of “The Renewables” , a company that sells renewable energy solutions to people. Recognizing India's vast potential for renewable energy growth, you decide to extend your operations into this dynamic market. To make informed decisions, you need to understand the country's energy landscape, including its actual and installed energy capacities. For this you turned to the NDAP for comprehensive data on India's energy sector,and obtained this dataset from there:

Q1: Ask 10 reasonably involved questions and try to answer them by analyzing the Dataset.

```
# Import libraries
import requests
from pathlib import Path
import zipfile
import os


request = requests.get("https://github.com/PanavShah1/DAV_assignment/blob/main/Power_Generation_Dataset.csv?raw=true")
with open("Power_Generation_Dataset.csv", "wb") as f:
  f.write(request.content)
  print("csv file downloaded")
```

```
csv file downloaded
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("Power_Generation_Dataset.csv")
df.head(5)
```

	ROWID	Country	State LGD Code	State	Actual energy generated	Category of Plant	Type of fuel used	Installed Capacity	Ge P
0	1	India	22	Chhattisgarh	0.0	THERMAL	COAL	0.0	
1	2	India	2	Himachal Pradesh	0.0	HYDRO	HYDRO	0.0	
2	3	India	27	Maharashtra	0.0	THERMAL	COAL	0.0	
3	4	India	9	Uttar Pradesh	0.0	THERMAL	COAL	0.0	
4	5	India	22	Chhattisgarh	0.0	THERMAL	COAL	0.0	

Next steps: Generate code with df View recommended plots

```
states = df['State'].unique()
plants = df['Category of Plant'].unique()
fuels = df['Type of fuel used'].unique()
sectors = df['Sector of power plant'].unique()
years = df['YearCode'].unique()
```

Q1: How many Thermal, Hydro and Nuclear Plants are there in India?

A1:

```
num_thermal = len(df.loc[df['Category of Plant'] == 'THERMAL'])
num_hydro = len(df.loc[df['Category of Plant'] == 'HYDRO'])
num_nuclear = len(df.loc[df['Category of Plant'] == 'NUCLEAR'])
print(f'There are {num_thermal} Thermal Plants, {num_hydro} Hydro Plants and {num_nuclear} Nuclear Plants in India')
```

```
There are 26427 Thermal Plants, 19465 Hydro Plants and 783 Nuclear Plants in India
```

Q2: How many plants use the different types of fuels

```
num_fuels = {fuel: 0 for fuel in fuels}

for fuel in num_fuels:
    condition = len(df.loc[df['Type of fuel used'] == fuel])
    num_fuels[fuel] = condition
```

A2:

```
for fuel in num_fuels:
  print(f"There are {num_fuels[fuel]} plants using {fuel}")
```

```
There are 17380 plants using COAL
There are 19465 plants using HYDRO
There are 986 plants using LIGNITE
There are 4795 plants using GAS
There are 783 plants using NUCLEAR
There are 137 plants using MULTI FUEL
There are 1220 plants using NATURAL GAS
There are 20 plants using THERMAL
There are 1177 plants using DIESEL
There are 267 plants using HIGH SPEED DIESEL
There are 445 plants using NAPTHA
```

Q3: What fuels do the Thermal, Hydro and Nuclear plants use

```
plant_data = {plant: {fuel: 0 for fuel in num_fuels} for plant in plants}

for plant in plants:
    for fuel in num_fuels:
        condition = (df['Category of Plant'] == plant) & (df['Type of fuel used'] == fuel)
        length = len(df.loc[condition])
        plant_data[plant][fuel] = length
```

A3:

```
for plant in plants:
    for fuel in num_fuels:
        if plant_data[plant][fuel] == 0:
            continue
        print(f"There are {plant_data[plant][fuel]} {plant} plants using {fuel}")
    print()
```

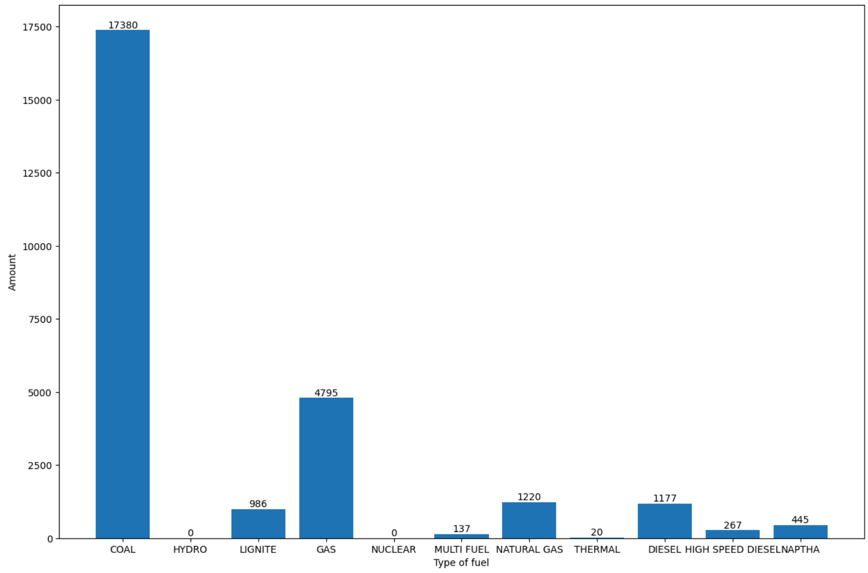
```
There are 17380 THERMAL plants using COAL
There are 986 THERMAL plants using LIGNITE
There are 4795 THERMAL plants using GAS
There are 137 THERMAL plants using MULTI FUEL
There are 1220 THERMAL plants using NATURAL GAS
There are 20 THERMAL plants using THERMAL
There are 1177 THERMAL plants using DIESEL
There are 267 THERMAL plants using HIGH SPEED DIESEL
There are 445 THERMAL plants using NAPTHA

There are 19465 HYDRO plants using HYDRO

There are 783 NUCLEAR plants using NUCLEAR
```

Q4: Graph the various fuels used in a Coal Plant

```
x = fuels
y = [0 for i in range(len(fuels))]
for i, fuel in enumerate(list(fuels)):
  y[i] = plant_data['THERMAL'][fuel]
```

A4:

```
fig = plt.figure(figsize=(15, 10))
plt.bar(x, y)
plt.xlabel("Type of fuel")
plt.ylabel("Amount")
for i, value in enumerate(y):
    plt.text(i, value + 1, str(value), ha='center', va='bottom', fontsize=10)
```



Q5: What is the plant distribution in the states of India

```
num_states = {state: len(df.loc[df['State'] == state]) for state in states }
print("State: Number of plants\n")
for state in sorted(num_states):
    print(f"{state}: {num_states[state]}")
```

```
State: Number of plants

Andaman And Nicobar Islands: 107
Andhra Pradesh: 1785
Arunachal Pradesh: 18
Assam: 666
Bihar: 486
Chhattisgarh: 2476
Delhi: 678
```

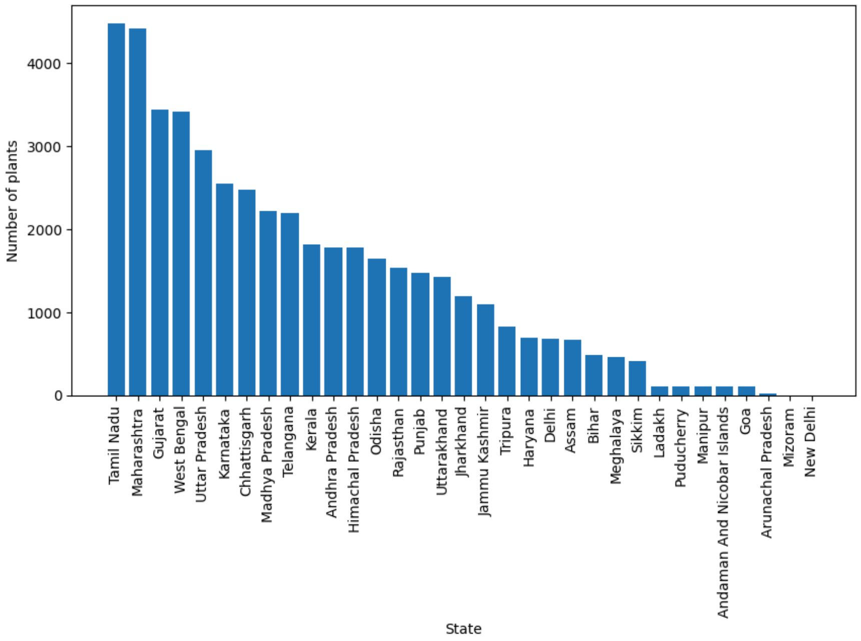
```
Goa: 107
Gujarat: 3443
Haryana: 690
Himachal Pradesh: 1776
Jammu Kashmir: 1103
Jharkhand: 1196
Karnataka: 2545
Kerala: 1819
Ladakh: 107
Madhya Pradesh: 2218
Maharashtra: 4417
Manipur: 107
Meghalaya: 468
Mizoram: 3
New Delhi: 1
Odisha: 1643
Puducherry: 107
Punjab: 1471
Rajasthan: 1532
Sikkim: 412
Tamil Nadu: 4478
Telangana: 2191
Tripura: 827
Uttar Pradesh: 2959
Uttarakhand: 1427
West Bengal: 3412
```

```
num_states = {k: v for k, v in sorted(num_states.items(), key=lambda item: item[1], reverse=True)}
```

A5:

```
fig = plt.figure(figsize=(10, 5))
plt.bar(num_states.keys(), num_states.values())
plt.xticks(rotation='vertical')
plt.xlabel("State")
plt.ylabel("Number of plants")
```

```
Text(0, 0.5, 'Number of plants')
```



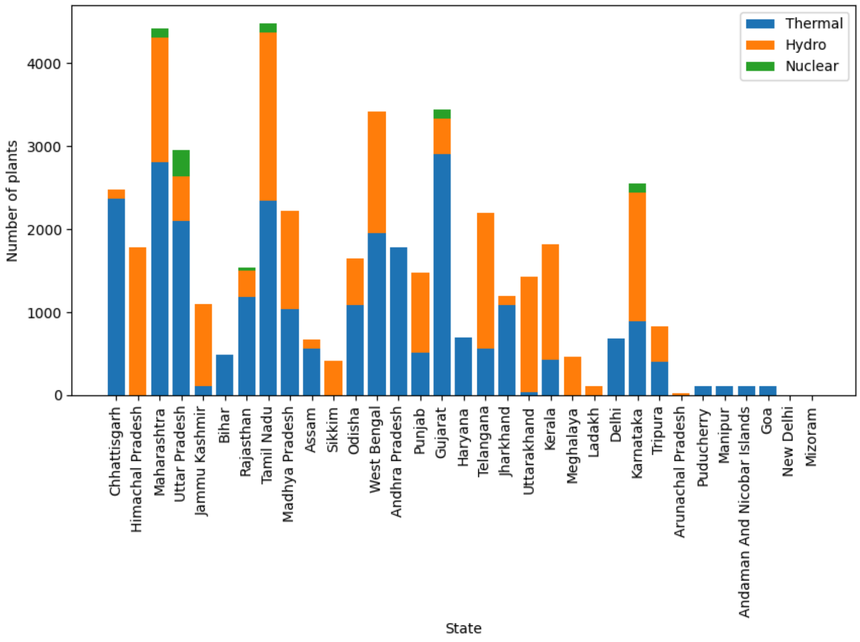
Q6: What is the distribution of the Thermal, Hydro and Nuclear plants in each state

```
state_data = {state: {plant: 0 for plant in plants} for state in states}

for state in states:
    for plant in plants:
        condition = (df['State'] == state) & (df['Category of Plant'] == plant)
        length = len(df.loc[condition])
        state_data[state][plant] = length
```

A6:

```
categories = list(state_data.keys())
thermal_data = [state_data[category]['THERMAL'] for category in categories]
hydro_data = [state_data[category]['HYDRO'] for category in categories]
nuclear_data = [state_data[category]['NUCLEAR'] for category in categories]

fig, ax = plt.subplots(figsize=(10, 5))
ax.bar(categories, thermal_data, label='Thermal')
ax.bar(categories, hydro_data, bottom=thermal_data, label='Hydro')
ax.bar(categories, nuclear_data, bottom=[x + y for x, y in zip(thermal_data, hydro_data)], label='Nuclear')
plt.xticks(rotation='vertical')
plt.xlabel("State")
plt.ylabel("Number of plants")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fe9e36da3e0>
```



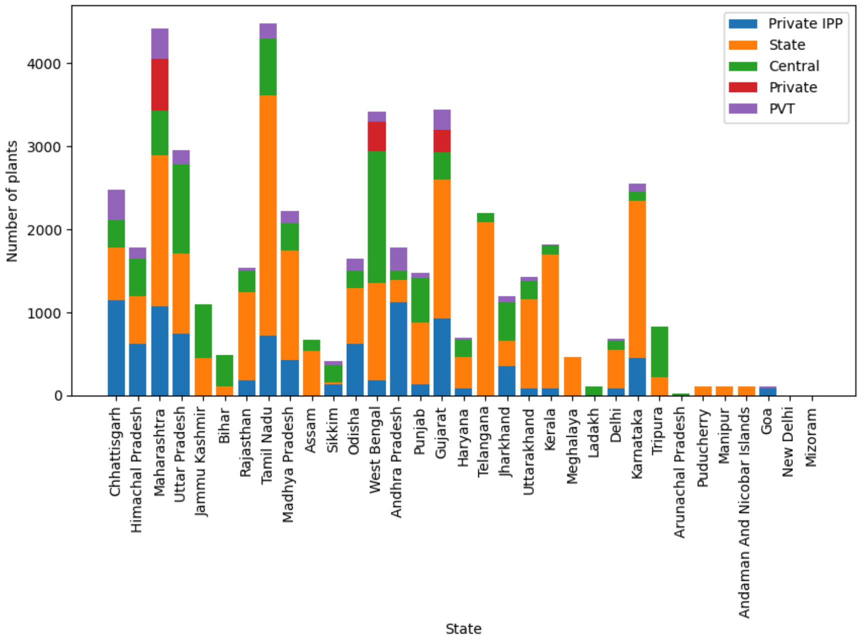
Q7: What's the distribution of the sector of power plants in the states

```
state_sector_data = {state: {sector: 0 for sector in sectors} for state in states}

for state in states:
    for sector in sectors:
        condition = (df['State'] == state) & (df['Sector of power plant'] == sector)
        length = len(df.loc[condition])
        state_sector_data[state][sector] = length
```

A7:

```
categories = list(state_sector_data.keys())
private_ipp = [state_sector_data[category]['PRIVATE IPP'] for category in categories]
state = [state_sector_data[category]['STATE'] for category in categories]
central = [state_sector_data[category]['CENTRAL'] for category in categories]
private = [state_sector_data[category]['PRIVATE'] for category in categories]
pvt = [state_sector_data[category]['PVT'] for category in categories]


fig, ax = plt.subplots(figsize=(10, 5))
ax.bar(categories, private_ipp, label='Private IPP')
ax.bar(categories, state, bottom=private_ipp, label='State')
ax.bar(categories, central, bottom=[x + y for x, y in zip(private_ipp, state)], label='Central')
ax.bar(categories, private, bottom=[x + y + z for x, y, z in zip(private_ipp, state, central)], label='Private')
ax.bar(categories, pvt, bottom=[x + y + z + w for x, y, z, w in zip(private_ipp, state, central, private)], label='PVT')


plt.xticks(rotation='vertical')
plt.xlabel("State")
plt.ylabel("Number of plants")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fe9e36da620>
```



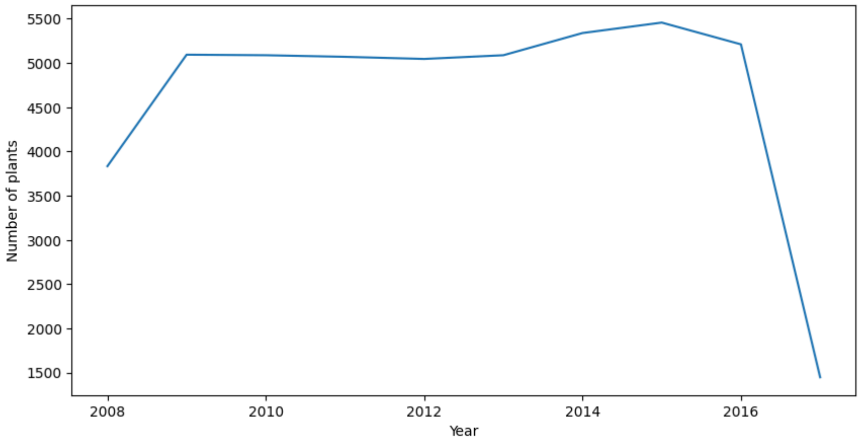
Q8: When were the power plants commissioned

```
year_data = {year: len(df.loc[df['YearCode'] == year]) for year in years}
year_data = dict(sorted(year_data.items(), key=lambda item: item[0]))
```

A8:

```
x = list(year_data.keys())
y = list(year_data.values())
fig = plt.figure(figsize=(10, 5))
plt.plot(x, y)
plt.xlabel("Year")
plt.ylabel("Number of plants")
```

```
Text(0, 0.5, 'Number of plants')
```

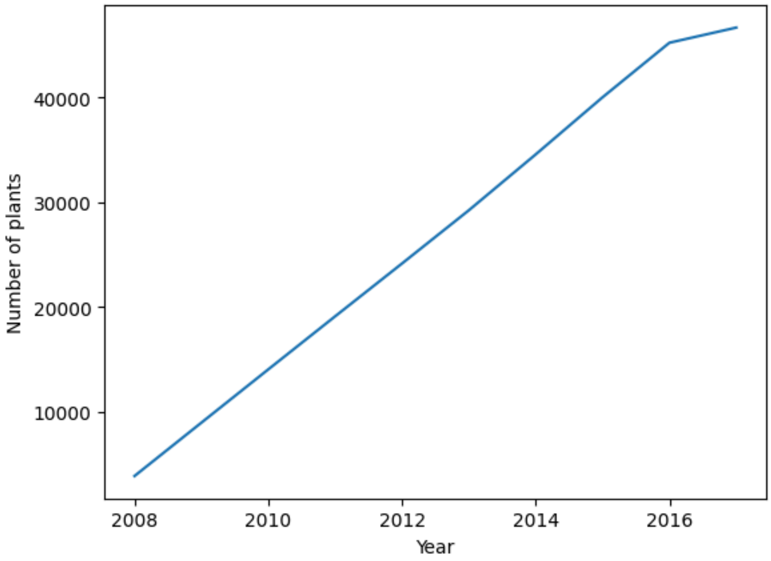


Q9: How many plants were running in that year

A9:

```
def cumulative_sum(nums):
    return [sum(nums[:i+1]) for i in range(len(nums))]

y1 = cumulative_sum(y)
plt.plot(x, y1)
plt.xlabel("Year")
plt.ylabel("Number of plants")
```

```
Text(0, 0.5, 'Number of plants')
```



Q10: Which are the top coal companies

```
companies = df['Name of coal company'].unique()
company_data = {company: len(df.loc[df['Name of coal company'] == company]) for company in companies}
sorted_company_data = {k: v for k, v in sorted(company_data.items(), key=lambda item: item[1], reverse=True)}
```

A10:

```
for i, company in enumerate(sorted_company_data):
    if i == 10:
        break
    print(f"{i+1} : {company} with {sorted_company_data[company]} coal plants")
```

```
1 : TNGDCL with 2889 coal plants
2 : NTPC Ltd. with 2651 coal plants
3 : NHPC with 2045 coal plants
4 : APGENCO with 2002 coal plants
5 : KPCL with 1894 coal plants
6 : MAHAGENCO with 1819 coal plants
7 : KSEB with 1605 coal plants
8 : MPPGCL with 1320 coal plants
9 : GSECL with 1100 coal plants
10 : RRVUNL with 1070 coal plants
```

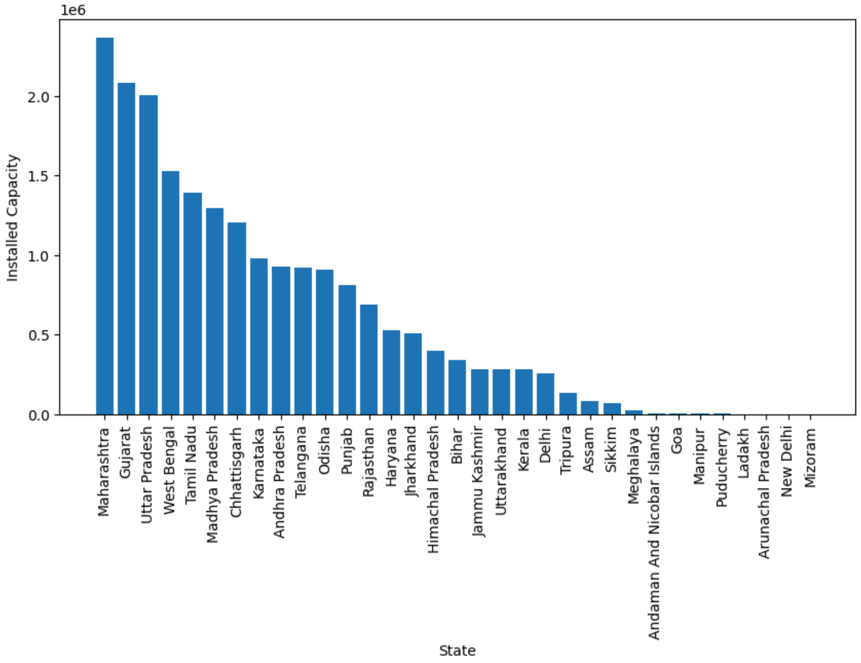
Q11: What is the amount of installed capacity of energy per state

```
installed_capacity = {state: 0 for state in states}
for state in states:
  installed_capacity[state] = df.loc[df['State'] == state]['Installed Capacity'].sum().round(2)
installed_capacity = {k: v for k, v in sorted(installed_capacity.items(), key=lambda item: item[1], reverse=True)}
```

A11:

```
fig = plt.figure(figsize=(10, 5))
plt.bar(list(installed_capacity.keys()), list(installed_capacity.values()))
plt.xlabel("State")
plt.ylabel("Installed Capacity")
plt.xticks(rotation='vertical')
```

```
 11,
 12,
 13,
 14,
 15,
 16,
 17,
 18,
 19,
 20,
 21,
 22,
 23,
 24,
 25,
 26,
 27,
 28,
 29,
 30,
 31,
 32],
[Text(0, 0, 'Maharashtra'),
 Text(1, 0, 'Gujarat'),
 Text(2, 0, 'Uttar Pradesh'),
 Text(3, 0, 'West Bengal'),
 Text(4, 0, 'Tamil Nadu'),
 Text(5, 0, 'Madhya Pradesh'),
 Text(6, 0, 'Chhattisgarh'),
 Text(7, 0, 'Karnataka'),
 Text(8, 0, 'Andhra Pradesh'),
 Text(9, 0, 'Telangana'),
 Text(10, 0, 'Odisha'),
 Text(11, 0, 'Punjab'),
 Text(12, 0, 'Rajasthan'),
 Text(13, 0, 'Haryana'),
 Text(14, 0, 'Jharkhand'),
 Text(15, 0, 'Himachal Pradesh'),
 Text(16, 0, 'Bihar'),
 Text(17, 0, 'Jammu Kashmir'),
 Text(18, 0, 'Uttarakhand'),
 Text(19, 0, 'Kerala'),
 Text(20, 0, 'Delhi'),
 Text(21, 0, 'Tripura'),
 Text(22, 0, 'Assam'),
 Text(23, 0, 'Sikkim'),
 Text(24, 0, 'Meghalaya'),
 Text(25, 0, 'Andaman And Nicobar Islands'),
 Text(26, 0, 'Goa'),
 Text(27, 0, 'Manipur'),
 Text(28, 0, 'Puducherry'),
 Text(29, 0, 'Ladakh'),
 Text(30, 0, 'Arunachal Pradesh'),
 Text(31, 0, 'New Delhi'),
 Text(32, 0, 'Mizoram')])
```



Q2: Seeing high levels of pollution in the country, you get curious about the trends in pollution and try to correlate it with the dataset for power plants you are given. How would you go about doing that? Is the current dataset sufficient to identify pollution trends? If not, what additional data would you need, and where would you obtain it? List the potential sources for acquiring this necessary data.

- Pollution is caused by various sources like combustion devices, motor vehicles, industrial facilites, forest fires, etc.
- But if we look into the distribution of pollution caused by these sources we find that **75.80% of green house gas emmisions is due to Energy Production**.
- Source - https://www.downtoearth.org.in/blog/pollution/5-rise-in-india-s-ghg-emissions-since-2016-driven-by-energy-industrial-sectors-94076
- So it is fair to approximate the trends in pollution to correspond to the amount of emissions released by these energy plants.
- The current dataset can give us an approximate of the amount of GHG emissions which correlates to the amount of pollution in India.
- Although if additional data about IPPU, Agriculture, Waste and Vehicle emissions is given, we can formulate a better approximation.

The amount of GHG emissions produced by the energy plants are:

- Thermal Plants - 584 g CO2/kWh
- Hydro Plants - 23 g CO2/kWh
- Nuclear Plants - 4 g CO2/kWh

Sources

- https://teriin.org/index.php/research-paper/assessment-greenhouse-gas-emissions-coal-and-natural-gas-thermal-power-plants-using#:~:text=The%20total%20GHG%20emission%20from,CO2%20eq%2FkWh%20electricity%20generation
- https://www.hydropower.org/factsheets/greenhouse-gas-emissions#:~:text=The%20results%20published%20in%20Water,consistent%20with%20the%20IPCC%20findings
- https://www.carbonbrief.org/solar-wind-nuclear-amazingly-low-carbon-footprints/

Let's calculate the distibution of GHG emmisions by the States of India

```
def assign_ghg_emission(category):
    if category == 'THERMAL':
        return 584
    elif category == 'HYDRO':
        return 23
    else:
        return 4
df['GHG Emission'] = df['Category of Plant'].apply(assign_ghg_emission)
df.head(5)
```

	ROWID	Country	State LGD Code	State	Actual energy generated	Category of Plant	Type of fuel used	Installed Capacity	Ge P
0	1	India	22	Chhattisgarh	0.0	THERMAL	COAL	0.0	
1	2	India	2	Himachal Pradesh	0.0	HYDRO	HYDRO	0.0	
2	3	India	27	Maharashtra	0.0	THERMAL	COAL	0.0	
3	4	India	9	Uttar Pradesh	0.0	THERMAL	COAL	0.0	
4	5	India	22	Chhattisgarh	0.0	THERMAL	COAL	0.0	

5 rows × 21 columns

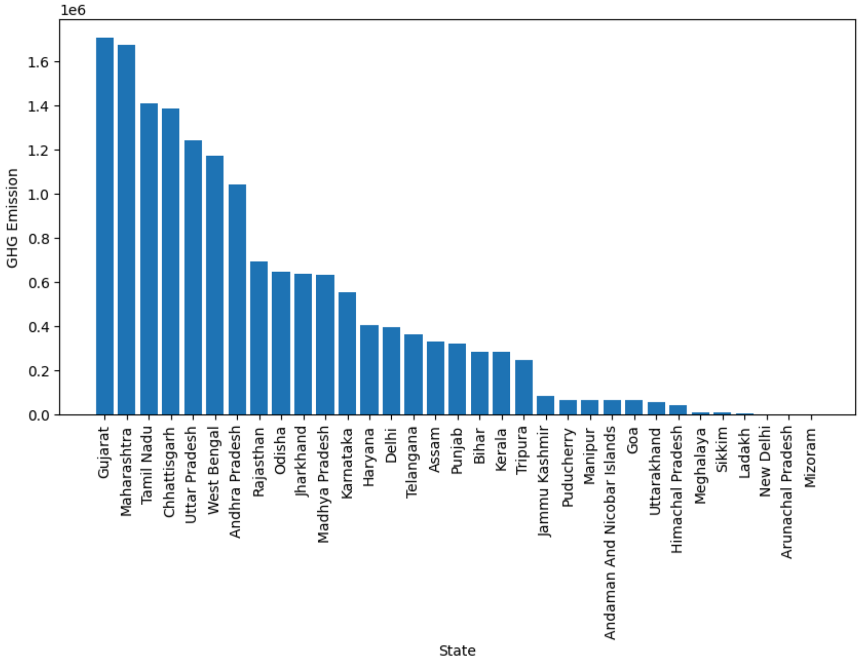
```
emission_states = {state: df.loc[df['State'] == state]['GHG Emission'].sum() for state in states}
emission_states = {k: v for k, v in sorted(emission_states.items(), key=lambda item: item[1], reverse=True)}
```

```
for emission_state in emission_states:
  print(f"{emission_state} produces {emission_states[emission_state]} g CO2/kWh")
```

```
Gujarat produces 1708544 g CO2/kWh
Maharashtra produces 1677090 g CO2/kWh
Tamil Nadu produces 1412579 g CO2/kWh
Chhattisgarh produces 1385957 g CO2/kWh
Uttar Pradesh produces 1241741 g CO2/kWh
West Bengal produces 1174670 g CO2/kWh
Andhra Pradesh produces 1042440 g CO2/kWh
Rajasthan produces 694887 g CO2/kWh
Odisha produces 644791 g CO2/kWh
Jharkhand produces 638437 g CO2/kWh
Madhya Pradesh produces 635015 g CO2/kWh
Karnataka produces 554670 g CO2/kWh
Haryana produces 402960 g CO2/kWh
Delhi produces 395952 g CO2/kWh
Telangana produces 362870 g CO2/kWh
Assam produces 328917 g CO2/kWh
Punjab produces 318821 g CO2/kWh
Bihar produces 283824 g CO2/kWh
Kerala produces 281945 g CO2/kWh
Tripura produces 245104 g CO2/kWh
Jammu Kashmir produces 85396 g CO2/kWh
Puducherry produces 62488 g CO2/kWh
Manipur produces 62488 g CO2/kWh
Andaman And Nicobar Islands produces 62488 g CO2/kWh
Goa produces 62488 g CO2/kWh
Uttarakhand produces 53017 g CO2/kWh
Himachal Pradesh produces 40848 g CO2/kWh
Meghalaya produces 10764 g CO2/kWh
Sikkim produces 9476 g CO2/kWh
Ladakh produces 2461 g CO2/kWh
New Delhi produces 584 g CO2/kWh
Arunachal Pradesh produces 414 g CO2/kWh
Mizoram produces 69 g CO2/kWh
```

```
fig = plt.figure(figsize=(10, 5))
plt.bar(list(emission_states.keys()), list(emission_states.values()))
plt.xlabel("State")
plt.ylabel("GHG Emission")
plt.xticks(rotation='vertical')
```

```
  11,
  12,
  13,
  14,
  15,
  16,
  17,
  18,
  19,
  20,
  21,
  22,
  23,
  24,
  25,
  26,
  27,
  28,
  29,
  30,
  31,
  32],
 [Text(0, 0, 'Gujarat'),
  Text(1, 0, 'Maharashtra'),
  Text(2, 0, 'Tamil Nadu'),
  Text(3, 0, 'Chhattisgarh'),
  Text(4, 0, 'Uttar Pradesh'),
  Text(5, 0, 'West Bengal'),
  Text(6, 0, 'Andhra Pradesh'),
  Text(7, 0, 'Rajasthan'),
  Text(8, 0, 'Odisha'),
  Text(9, 0, 'Jharkhand'),
  Text(10, 0, 'Madhya Pradesh'),
  Text(11, 0, 'Karnataka'),
  Text(12, 0, 'Haryana'),
  Text(13, 0, 'Delhi'),
  Text(14, 0, 'Telangana'),
  Text(15, 0, 'Assam'),
  Text(16, 0, 'Punjab'),
  Text(17, 0, 'Bihar'),
  Text(18, 0, 'Kerala'),
  Text(19, 0, 'Tripura'),
  Text(20, 0, 'Jammu Kashmir'),
  Text(21, 0, 'Puducherry'),
  Text(22, 0, 'Manipur'),
  Text(23, 0, 'Andaman And Nicobar Islands'),
  Text(24, 0, 'Goa'),
  Text(25, 0, 'Uttarakhand'),
  Text(26, 0, 'Himachal Pradesh'),
  Text(27, 0, 'Meghalaya'),
  Text(28, 0, 'Sikkim'),
  Text(29, 0, 'Ladakh'),
  Text(30, 0, 'New Delhi'),
  Text(31, 0, 'Arunachal Pradesh'),
  Text(32, 0, 'Mizoram')])
```



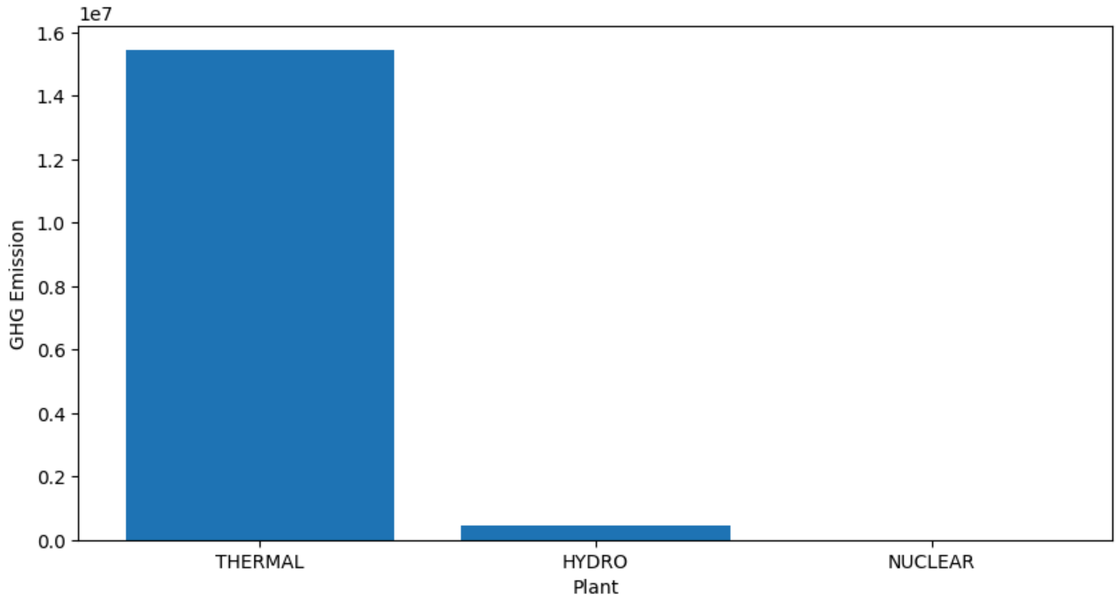
Now let's see which source of energy is worse for the country and by how much

```
emission_plant = {plant: df.loc[df['Category of Plant'] == plant]['GHG Emission'].sum() for plant in plants}
emission_plant
```

```
{'THERMAL': 15433368, 'HYDRO': 447695, 'NUCLEAR': 3132}
```

```
fig = plt.figure(figsize=(10, 5))
plt.bar(list(emission_plant.keys()), list(emission_plant.values()))
plt.xlabel("Plant")
plt.ylabel("GHG Emission")
```

```
Text(0, 0.5, 'GHG Emission')
```



```
emission_plant['THERMAL'] / (emission_plant['THERMAL'] + emission_plant['HYDRO'] + emission_plant['NUCLEAR']) * 100
```

```
97.16178880956825
```

Conclusion

1. The top five states which produce the most pollution are Gujrat, Maharashtra, Tamil Nadu, Chattisgarh and Uttar Pradesh
2. Clearly from the bar graph we can see that the production of energy from thermal plants creates a lot more pollution than the production of energy from hydro or nuclear sources. To be precise, it releases 97.16% of the total emissions released by energy production sources

DAV Assignment Q3 and Q4

After extensive research, suppose you decided to enter the Indian Market with your existing products (P, Q, R, S and T). Your team has determined that the behavior of the new market in India closely mirrors that of your existing market. In your current market, the sales team categorizes all customers into four segments (A, B, C, D). They then tailor their outreach and communication strategies to each specific segment. This approach has proven to be highly effective for them.

They plan to use the same strategy for the Indian markets and have identified about 2500 new potential trial customers. For this they have an existing database of about 8000 customers which they have already classified.

Q3: Implement methods to assist your team in identifying and predicting the appropriate group (A, B, C, or D) for each customer in the test data.

```
# Import libraries
import requests
from pathlib import Path
import zipfile
import os


request = requests.get("https://github.com/PanavShah1/DAV_assignment/blob/main/Consumer_Dataset.csv?raw=true")
with open("Consumer_Dataset.csv", "wb") as f:
  f.write(request.content)
  print("csv file downloaded")

request = requests.get("https://github.com/PanavShah1/DAV_assignment/blob/main/Consumer%20Test%20Dataset.csv?raw=true")
with open("Consumer_Test_Dataset.csv", "wb") as f:
  f.write(request.content)
  print("csv file downloaded")
```

```
csv file downloaded
csv file downloaded
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("Consumer_Dataset.csv")
df.head(5)
```

	Unnamed: 0	Gender	Age	Ever_Married	Family_Size	Profession	Graduated	Wor
0	0	Male	22	No	4.0	Healthcare	No	
1	1	Female	38	Yes	3.0	Engineer	Yes	
2	2	Female	67	Yes	1.0	Engineer	Yes	
3	3	Male	67	Yes	2.0	Lawyer	Yes	
4	4	Female	40	Yes	6.0	Entertainment	Yes	

Next steps: Generate code with df View recommended plots

Get the dataset ready

```
professions = df['Profession'].unique()
renewable = df['Preferred_Renewable'].unique()
avg_work_experience = df['Work_Experience'].mean()
avg_family_size = df['Family_Size'].mean()
```

```
df = df.replace(to_replace="Male", value=0)
df = df.replace(to_replace="Female", value=1)
df = df.replace(to_replace="Yes", value=1)
df = df.replace(to_replace="No", value=0)
df['Work_Experience'] = df['Work_Experience'].fillna(avg_work_experience)
df = df.replace(to_replace="Low", value=0)
df = df.replace(to_replace="Average", value=1)
df = df.replace(to_replace="High", value=2)
df['Family_Size'] = df['Family_Size'].fillna(avg_family_size)

df = df.replace(to_replace="A", value=0)
df = df.replace(to_replace="B", value=1)
df = df.replace(to_replace="C", value=2)
df = df.replace(to_replace="D", value=3)
df.head(5)
```

	Unnamed: 0	Gender	Age	Ever_Married	Family_Size	Profession	Graduated	Wor
0	0	0	22	0.0	4.0	Healthcare	0.0	
1	1	1	38	1.0	3.0	Engineer	1.0	
2	2	1	67	1.0	1.0	Engineer	1.0	
3	3	0	67	1.0	2.0	Lawyer	1.0	
4	4	1	40	1.0	6.0	Entertainment	1.0	

Next steps: Generate code with df View recommended plots

```
for i, source in enumerate(professions):
  df['Profession'][df['Profession'] == source] = i
avg_profession = df['Profession'].mean()
df['Profession'] = df['Profession'].fillna(avg_profession)
avg_profession
```

```
<ipython-input-248-2b2318068495>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Profession'][df['Profession'] == source] = i
<ipython-input-248-2b2318068495>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Profession'][df['Profession'] == source] = i
<ipython-input-248-2b2318068495>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Profession'][df['Profession'] == source] = i
<ipython-input-248-2b2318068495>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Profession'][df['Profession'] == source] = i
<ipython-input-248-2b2318068495>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Profession'][df['Profession'] == source] = i
<ipython-input-248-2b2318068495>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Profession'][df['Profession'] == source] = i
<ipython-input-248-2b2318068495>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Profession'][df['Profession'] == source] = i
<ipython-input-248-2b2318068495>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Profession'][df['Profession'] == source] = i
<ipython-input-248-2b2318068495>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Profession'][df['Profession'] == source] = i
<ipython-input-248-2b2318068495>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Profession'][df['Profession'] == source] = i
3.2775679758308156
```

```
avg_married = df['Ever_Married'].mean()
df['Ever_Married'] = df['Ever_Married'].fillna(avg_married)
avg_married
```

```
0.5856458123107972
```

```
for i, source in enumerate(renewable):
  df['Preferred_Renewable'][df['Preferred_Renewable'] == source] = i
avg_preferred_renewable = df['Preferred_Renewable'].mean()
df['Preferred_Renewable'] = df['Preferred_Renewable'].fillna(avg_preferred_renewable)
avg_preferred_renewable
```

```
<ipython-input-250-cd5c1d51057a>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Preferred_Renewable'][df['Preferred_Renewable'] == source] = i
<ipython-input-250-cd5c1d51057a>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Preferred_Renewable'][df['Preferred_Renewable'] == source] = i
<ipython-input-250-cd5c1d51057a>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Preferred_Renewable'][df['Preferred_Renewable'] == source] = i
<ipython-input-250-cd5c1d51057a>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Preferred_Renewable'][df['Preferred_Renewable'] == source] = i
<ipython-input-250-cd5c1d51057a>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Preferred_Renewable'][df['Preferred_Renewable'] == source] = i
<ipython-input-250-cd5c1d51057a>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Preferred_Renewable'][df['Preferred_Renewable'] == source] = i
<ipython-input-250-cd5c1d51057a>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df['Preferred_Renewable'][df['Preferred_Renewable'] == source] = i
1.4197947947947949
```

```
avg_graduated = df['Graduated'].mean()
df['Graduated'] = df['Graduated'].fillna(avg_graduated)
avg_graduated
```

```
0.6217772215269086
```

```
df = df.drop('Unnamed: 0', axis=1)
df.head(5)
```

	Gender	Age	Ever_Married	Family_Size	Profession	Graduated	Work_Experien
0	0	22	0.0	4.0	0.0	0.0	1.000
1	1	38	1.0	3.0	1.0	1.0	2.641
2	1	67	1.0	1.0	1.0	1.0	1.000
3	0	67	1.0	2.0	2.0	1.0	0.000
4	1	40	1.0	6.0	3.0	1.0	2.641

Next steps: Generate code with df View recommended plots

```
data = df.to_numpy()
data
```

```
array([[ 0., 22.,  0., ...,  0.,  0.,  3.],
       [ 1., 38.,  1., ...,  1.,  0.,  0.],
       [ 1., 67.,  1., ...,  0.,  1.,  1.],
       ...,
       [ 1., 33.,  0., ...,  0.,  1.,  3.],
       [ 1., 27.,  0., ...,  0.,  1.,  1.],
       [ 0., 37.,  1., ...,  1.,  0.,  1.]])
```

```
data.shape
```

```
(8068, 10)
```

```
X = data[:, 0:9]
y = data[:, 9]
X.shape, y.shape
```

```
((8068, 9), (8068,))
```

```
import torch
from torch import nn
```

```
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

```
'cpu'
```

```
X = torch.from_numpy(X)
y = torch.from_numpy(y)
X.shape, y.shape
```

```
(torch.Size([8068, 9]), torch.Size([8068]))
```

```
X = X.type(torch.float32)
y = y.type(torch.long)
X.dtype, y.dtype
```

```
(torch.float32, torch.int64)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
(torch.Size([7261, 9]),
 torch.Size([807, 9]),
 torch.Size([7261]),
 torch.Size([807]))
```

```
from torch.utils.data import Dataset

class ConsumerDataset(Dataset):
  def __init__(self, X, y):
    self.X = X
    self.y = y

  def __len__(self):
    return len(self.X)

  def __getitem__(self, idx):
    return self.X[idx], self.y[idx]

train_dataset = ConsumerDataset(X_train, y_train)
test_dataset = ConsumerDataset(X_test, y_test)
train_dataset, test_dataset
```

```
(<__main__.ConsumerDataset at 0x78e875a6ffd0>,
 <__main__.ConsumerDataset at 0x78e875a6d7b0>)
```

```
from torch.utils.data import DataLoader

BATCH_SIZE = 16
train_dataloader = DataLoader(dataset=train_dataset,
                              batch_size=BATCH_SIZE,
                              shuffle=True)
test_dataloader = DataLoader(dataset=test_dataset,
                             batch_size=BATCH_SIZE,
                             shuffle=False)
train_dataloader, test_dataloader
```

```
(<torch.utils.data.dataloader.DataLoader at 0x78e875a6d750>,
 <torch.utils.data.dataloader.DataLoader at 0x78e875a6f5e0>)
```

```
from torch import nn

class ModelV0(nn.Module):
  def __init__(self,
               input_shape: int,
               hidden_layers: int,
               output_shape: int = 4):
    super().__init__()
    self.layer_stack = nn.Sequential(
        nn.Linear(in_features=input_shape,
                  out_features=hidden_layers),
        nn.ReLU(),
        nn.Linear(in_features=hidden_layers,
                  out_features=hidden_layers),
        # nn.ReLU(),
        # nn.Linear(in_features=hidden_layers,
        #           out_features=hidden_layers),
        nn.ReLU(),
        nn.Linear(in_features=hidden_layers,
                  out_features=output_shape)
    )

  def forward(self, x):
    return self.layer_stack(x)
```

```
model_0 = ModelV0(input_shape=X.shape[1], hidden_layers=30, output_shape=4)
model_0
```

```
ModelV0(
  (layer_stack): Sequential(
    (0): Linear(in_features=9, out_features=30, bias=True)
    (1): ReLU()
    (2): Linear(in_features=30, out_features=30, bias=True)
    (3): ReLU()
    (4): Linear(in_features=30, out_features=4, bias=True)
  )
)
```

```
model_0.state_dict().keys()
```

```
odict_keys(['layer_stack.0.weight', 'layer_stack.0.bias', 'layer_stack.2.weight', 'layer_stack.2.bias',
'layer_stack.4.weight', 'layer_stack.4.bias'])
```

```
def train_step(model: torch.nn.Module,
               dataloader: torch.utils.data.DataLoader,
               loss_fn: torch.nn.Module,
               optimizer: torch.optim.Optimizer,
               device=device):
  net_loss, net_acc = 0, 0
  model.train()
  for batch, (X, y) in enumerate(dataloader):
    X, y = X.to(device), y.to(device)
    y_pred = model(X)
    loss = loss_fn(y_pred, y)
    net_loss += loss
    net_acc += (y_pred.argmax(dim=1) == y).sum().item() / len(y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
  net_loss /= len(dataloader)
  net_acc /= len(dataloader)
  return net_loss, net_acc
```

```
def test_step(model: torch.nn.Module,
              dataloader: torch.utils.data.DataLoader,
              loss_fn: torch.nn.Module,
              device=device):
  net_loss, net_acc = 0, 0
  model.eval()
  with torch.inference_mode():
    for batch, (X, y) in enumerate(dataloader):
      X, y = X.to(device), y.to(device)
      y_pred = model(X)
      net_loss += loss_fn(y_pred, y)
      net_acc += (y_pred.argmax(dim=1) == y).sum().item() / len(y)
    net_loss = net_loss / len(dataloader)
    net_acc = net_acc / len(dataloader)
    return net_loss, net_acc
```

```
from tqdm.auto import tqdm

def train(model: torch.nn.Module,
          train_dataloader: torch.utils.data.DataLoader,
          test_dataloader: torch.utils.data.DataLoader,
          optimizer: torch.optim.Optimizer,
          loss_fn: torch.nn.Module,
          epochs: int = 5,
          device = device):

  results = {"train_loss": [],
             "train_acc": [],
             "test_loss": [],
             "test_acc": []}
  for epoch in range(epochs):
    train_loss, train_acc = train_step(model=model,
                                      dataloader=train_dataloader,
                                      loss_fn=loss_fn,
                                      optimizer=optimizer,
                                      device=device)
    test_loss, test_acc = test_step(model=model,
                                    dataloader=test_dataloader,
                                    loss_fn=loss_fn,
                                    device=device)

    print(f"Epoch: {epoch} | Train loss: {train_loss:.4f} | Train acc: {train_acc:.4f} | Test loss: {test_loss:4f} | Test ac
    results["train_loss"].append(train_loss)
    results["train_acc"].append(train_acc)
    results["test_loss"].append(test_loss)
    results["test_acc"].append(test_acc)

  return results
```

```
NUM_EPOCHS = 50

model_0 = ModelV0(input_shape=X.shape[1],
                  hidden_layers=100,
                  output_shape=4)

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_0.parameters(), lr=0.001)
```

```
results = train(model=model_0,
                train_dataloader=train_dataloader,
                test_dataloader=test_dataloader,
                optimizer=optimizer,
                loss_fn=loss_fn,
                epochs=NUM_EPOCHS)
```

```
Epoch: 0 | Train loss: 1.3058 | Train acc: 0.3889 | Test loss: 1.228549 | Test acc: 0.4039
Epoch: 1 | Train loss: 1.2180 | Train acc: 0.4496 | Test loss: 1.174514 | Test acc: 0.4680
Epoch: 2 | Train loss: 1.1847 | Train acc: 0.4659 | Test loss: 1.232061 | Test acc: 0.4597
Epoch: 3 | Train loss: 1.1680 | Train acc: 0.4749 | Test loss: 1.142212 | Test acc: 0.4762
Epoch: 4 | Train loss: 1.1550 | Train acc: 0.4782 | Test loss: 1.124821 | Test acc: 0.4977
Epoch: 5 | Train loss: 1.1469 | Train acc: 0.4827 | Test loss: 1.133963 | Test acc: 0.4876
Epoch: 6 | Train loss: 1.1410 | Train acc: 0.4898 | Test loss: 1.125877 | Test acc: 0.4912
Epoch: 7 | Train loss: 1.1301 | Train acc: 0.4887 | Test loss: 1.136982 | Test acc: 0.4944
Epoch: 8 | Train loss: 1.1274 | Train acc: 0.4910 | Test loss: 1.137922 | Test acc: 0.4940
Epoch: 9 | Train loss: 1.1185 | Train acc: 0.4987 | Test loss: 1.101650 | Test acc: 0.4998
Epoch: 10 | Train loss: 1.1084 | Train acc: 0.4958 | Test loss: 1.101140 | Test acc: 0.5026
Epoch: 11 | Train loss: 1.1036 | Train acc: 0.4976 | Test loss: 1.101348 | Test acc: 0.5039
Epoch: 12 | Train loss: 1.0972 | Train acc: 0.5048 | Test loss: 1.124134 | Test acc: 0.4891
```

```
Epoch: 13 | Train loss: 1.0945 | Train acc: 0.5012 | Test loss: 1.103670 | Test acc: 0.5100
Epoch: 14 | Train loss: 1.0866 | Train acc: 0.5077 | Test loss: 1.081837 | Test acc: 0.5165
Epoch: 15 | Train loss: 1.0850 | Train acc: 0.5048 | Test loss: 1.095448 | Test acc: 0.5084
Epoch: 16 | Train loss: 1.0781 | Train acc: 0.5163 | Test loss: 1.086476 | Test acc: 0.5030
Epoch: 17 | Train loss: 1.0751 | Train acc: 0.5101 | Test loss: 1.099423 | Test acc: 0.4928
Epoch: 18 | Train loss: 1.0691 | Train acc: 0.5150 | Test loss: 1.092672 | Test acc: 0.4977
Epoch: 19 | Train loss: 1.0696 | Train acc: 0.5176 | Test loss: 1.071761 | Test acc: 0.5054
Epoch: 20 | Train loss: 1.0632 | Train acc: 0.5187 | Test loss: 1.094769 | Test acc: 0.4974
Epoch: 21 | Train loss: 1.0669 | Train acc: 0.5199 | Test loss: 1.080727 | Test acc: 0.5014
Epoch: 22 | Train loss: 1.0603 | Train acc: 0.5200 | Test loss: 1.087723 | Test acc: 0.5124
Epoch: 23 | Train loss: 1.0586 | Train acc: 0.5246 | Test loss: 1.086855 | Test acc: 0.5348
Epoch: 24 | Train loss: 1.0550 | Train acc: 0.5277 | Test loss: 1.095753 | Test acc: 0.5201
Epoch: 25 | Train loss: 1.0513 | Train acc: 0.5294 | Test loss: 1.089631 | Test acc: 0.5014
Epoch: 26 | Train loss: 1.0512 | Train acc: 0.5304 | Test loss: 1.075911 | Test acc: 0.5226
Epoch: 27 | Train loss: 1.0476 | Train acc: 0.5259 | Test loss: 1.085704 | Test acc: 0.5161
Epoch: 28 | Train loss: 1.0425 | Train acc: 0.5317 | Test loss: 1.102374 | Test acc: 0.5312
Epoch: 29 | Train loss: 1.0424 | Train acc: 0.5391 | Test loss: 1.106436 | Test acc: 0.4928
Epoch: 30 | Train loss: 1.0411 | Train acc: 0.5322 | Test loss: 1.118480 | Test acc: 0.5103
Epoch: 31 | Train loss: 1.0395 | Train acc: 0.5324 | Test loss: 1.087901 | Test acc: 0.5112
Epoch: 32 | Train loss: 1.0375 | Train acc: 0.5314 | Test loss: 1.106410 | Test acc: 0.5124
Epoch: 33 | Train loss: 1.0352 | Train acc: 0.5398 | Test loss: 1.112103 | Test acc: 0.4916
Epoch: 34 | Train loss: 1.0321 | Train acc: 0.5374 | Test loss: 1.098401 | Test acc: 0.5140
Epoch: 35 | Train loss: 1.0276 | Train acc: 0.5440 | Test loss: 1.108210 | Test acc: 0.4919
Epoch: 36 | Train loss: 1.0277 | Train acc: 0.5431 | Test loss: 1.113733 | Test acc: 0.5165
Epoch: 37 | Train loss: 1.0249 | Train acc: 0.5399 | Test loss: 1.101434 | Test acc: 0.5088
Epoch: 38 | Train loss: 1.0232 | Train acc: 0.5402 | Test loss: 1.106688 | Test acc: 0.4981
Epoch: 39 | Train loss: 1.0215 | Train acc: 0.5462 | Test loss: 1.100720 | Test acc: 0.5385
Epoch: 40 | Train loss: 1.0195 | Train acc: 0.5437 | Test loss: 1.112374 | Test acc: 0.5137
Epoch: 41 | Train loss: 1.0167 | Train acc: 0.5421 | Test loss: 1.108079 | Test acc: 0.5137
Epoch: 42 | Train loss: 1.0140 | Train acc: 0.5505 | Test loss: 1.120486 | Test acc: 0.5112
Epoch: 43 | Train loss: 1.0124 | Train acc: 0.5527 | Test loss: 1.143582 | Test acc: 0.4977
Epoch: 44 | Train loss: 1.0134 | Train acc: 0.5418 | Test loss: 1.133351 | Test acc: 0.4876
Epoch: 45 | Train loss: 1.0069 | Train acc: 0.5533 | Test loss: 1.102942 | Test acc: 0.5152
Epoch: 46 | Train loss: 1.0048 | Train acc: 0.5490 | Test loss: 1.119493 | Test acc: 0.5075
Epoch: 47 | Train loss: 1.0022 | Train acc: 0.5517 | Test loss: 1.124027 | Test acc: 0.5116
Epoch: 48 | Train loss: 0.9985 | Train acc: 0.5546 | Test loss: 1.111841 | Test acc: 0.5189
Epoch: 49 | Train loss: 0.9993 | Train acc: 0.5532 | Test loss: 1.123729 | Test acc: 0.5075
```

```
from sklearn import tree

clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
0.4510532837670384
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 5, 10],
    'max_features': [None, 'sqrt', 'log2']
}
grid_search = GridSearchCV(estimator=tree.DecisionTreeClassifier(), param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)
print(f"Best parameters found: {grid_search.best_params_}")
best_model = grid_search.best_estimator_
best_model.score(X_test, y_test)
```

```
Best parameters found: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 10, 'min_samples_split': 20}
0.48079306071871125
```

```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X_train, y_train)
rf_clf.score(X_test, y_test)
```

```
0.46964064436183395
```

```
from sklearn.model_selection import cross_val_score

cross_val_scores = cross_val_score(tree.DecisionTreeClassifier(), X_train, y_train, cv=5)
print(f"Cross-validation scores: {cross_val_scores}")
print(f"Average cross-validation score: {cross_val_scores.mean()}")
```

```
Cross-validation scores: [0.44872677 0.45247934 0.43801653 0.44214876 0.43181818]
Average cross-validation score: 0.44263791642256256
```

```
clf = tree.DecisionTreeClassifier(class_weight='balanced')
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
0.44114002478314746
```

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(X_train, y_train)
lr.score(X_test, y_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to conve
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
0.4510532837670384
```

Make predictions

Using the manually created linear regression model

```
df1 = pd.read_csv("Consumer_Test_Dataset.csv")
df2 = df1.copy()
df1.head(5)
```

	Unnamed: 0	Gender	Age	Ever_Married	Family_Size	Profession	Graduated	Wor
0	0	Female	36	Yes	1.0	Engineer	Yes	
1	1	Male	37	Yes	4.0	Healthcare	Yes	
2	2	Female	69	Yes	1.0	NaN	No	
3	3	Male	59	Yes	2.0	Executive	No	
4	4	Female	19	No	4.0	Marketing	No	

Next steps: Generate code with df1 View recommended plots

```
df1 = df1.replace(to_replace="Male", value=0)
df1 = df1.replace(to_replace="Female", value=1)
df1 = df1.replace(to_replace="Yes", value=1)
df1 = df1.replace(to_replace="No", value=0)
df1['Work_Experience'] = df1['Work_Experience'].fillna(avg_work_experience)
df1 = df1.replace(to_replace="Low", value=0)
df1 = df1.replace(to_replace="Average", value=1)
df1 = df1.replace(to_replace="High", value=2)
df1['Family_Size'] = df1['Family_Size'].fillna(avg_family_size)

df1 = df1.replace(to_replace="A", value=0)
df1 = df1.replace(to_replace="B", value=1)
df1 = df1.replace(to_replace="C", value=2)
df1 = df1.replace(to_replace="D", value=3)
df1
```

	Unnamed: 0	Gender	Age	Ever_Married	Family_Size	Profession	Graduated
0	0	1	36	1.0	1.0	Engineer	1.0
1	1	0	37	1.0	4.0	Healthcare	1.0
2	2	1	69	1.0	1.0	NaN	0.0
3	3	0	59	1.0	2.0	Executive	0.0
4	4	1	19	0.0	4.0	Marketing	0.0
...	...	...	...	...	...	...	...
2622	2622	0	29	0.0	4.0	Healthcare	0.0
2623	2623	1	35	0.0	1.0	Doctor	1.0
2624	2624	1	53	0.0	2.0	Entertainment	1.0
2625	2625	0	47	1.0	5.0	Executive	1.0
2626	2626	1	43	0.0	3.0	Healthcare	1.0

2627 rows × 10 columns

Next steps: Generate code with df1 View recommended plots

```
for i, source in enumerate(professions):
  df1['Profession'][df1['Profession'] == source] = i
avg_profession = df1['Profession'].mean()
df1['Profession'] = df1['Profession'].fillna(avg_profession)
avg_profession
```

```
<ipython-input-279-e4db59ec5021>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Profession'][df1['Profession'] == source] = i
<ipython-input-279-e4db59ec5021>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Profession'][df1['Profession'] == source] = i
<ipython-input-279-e4db59ec5021>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Profession'][df1['Profession'] == source] = i
<ipython-input-279-e4db59ec5021>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Profession'][df1['Profession'] == source] = i
<ipython-input-279-e4db59ec5021>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Profession'][df1['Profession'] == source] = i
<ipython-input-279-e4db59ec5021>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Profession'][df1['Profession'] == source] = i
<ipython-input-279-e4db59ec5021>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Profession'][df1['Profession'] == source] = i
<ipython-input-279-e4db59ec5021>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Profession'][df1['Profession'] == source] = i
<ipython-input-279-e4db59ec5021>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Profession'][df1['Profession'] == source] = i
<ipython-input-279-e4db59ec5021>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Profession'][df1['Profession'] == source] = i
3.3151796060254926
```

```
avg_married = df1['Ever_Married'].mean()
df1['Ever_Married'] = df1['Ever_Married'].fillna(avg_married)
avg_married
```

```
0.5898331393092744
```

```
for i, source in enumerate(renewable):
  df1['Preferred_Renewable'][df1['Preferred_Renewable'] == source] = i
avg_preferred_renewable = df1['Preferred_Renewable'].mean()
df1['Preferred_Renewable'] = df1['Preferred_Renewable'].fillna(avg_preferred_renewable)
avg_preferred_renewable
```

```
<ipython-input-281-880a2e04fb0b>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Preferred_Renewable'][df1['Preferred_Renewable'] == source] = i
<ipython-input-281-880a2e04fb0b>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Preferred_Renewable'][df1['Preferred_Renewable'] == source] = i
<ipython-input-281-880a2e04fb0b>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Preferred_Renewable'][df1['Preferred_Renewable'] == source] = i
<ipython-input-281-880a2e04fb0b>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Preferred_Renewable'][df1['Preferred_Renewable'] == source] = i
<ipython-input-281-880a2e04fb0b>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Preferred_Renewable'][df1['Preferred_Renewable'] == source] = i
<ipython-input-281-880a2e04fb0b>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Preferred_Renewable'][df1['Preferred_Renewable'] == source] = i
<ipython-input-281-880a2e04fb0b>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  df1['Preferred_Renewable'][df1['Preferred_Renewable'] == source] = i
1.40616570327553
```

```
avg_graduated = df1['Graduated'].mean()
df1['Graduated'] = df1['Graduated'].fillna(avg_graduated)
avg_graduated
```

```
0.6154437187860161
```

```
df1 = df1.drop('Unnamed: 0', axis=1)
df1.head(5)
```

	Gender	Age	Ever_Married	Family_Size	Profession	Graduated	Work_Experien
0	1	36	1.0	1.0	1.00000	1.0	0.0000
1	0	37	1.0	4.0	0.00000	1.0	8.0000
2	1	69	1.0	1.0	3.31518	0.0	0.0000
3	0	59	1.0	2.0	5.00000	0.0	11.0000
4	1	19	0.0	4.0	8.00000	0.0	2.6416

Next steps: Generate code with df1 View recommended plots

```
data1 = df1.to_numpy()
data1
```

```
array([[ 1.        , 36.        ,  1.        , ...,  0.        ,
         0.        ,  1.        ],
       [ 0.        , 37.        ,  1.        , ...,  8.        ,
         1.        ,  1.        ],
       [ 1.        , 69.        ,  1.        , ...,  0.        ,
        0.        ,  1.        ],
       ...,
       [ 1.        , 53.        ,  0.        , ...,  2.64166321,
         0.        ,  1.        ],
       [ 0.        , 47.        ,  1.        , ...,  1.        ,
         2.        ,  0.        ],
       [ 1.        , 43.        ,  0.        , ...,  9.        ,
         0.        ,  2.        ]])
```

```
data1.shape
```

```
(2627, 9)
```

```
X1 = data1[:, 0:9]
X1.shape
```

```
(2627, 9)
```

```
X1 = torch.from_numpy(X1)
X1.shape
```

```
torch.Size([2627, 9])
```

```
X1 = X1.type(torch.float32)
```

```
predictions = model_0(X1).argmax(dim=1).tolist()
```

```
for i, prediction in enumerate(predictions):
  if prediction == 0:
    predictions[i] = 'A'
  elif prediction == 1:
    predictions[i] = 'B'
  elif prediction == 2:
    predictions[i] = 'C'
  elif prediction == 3:
    predictions[i] = 'D'
  else:
    predictions[i] = 'Error'
predictions[:5]
```

```
['A', 'C', 'B', 'C', 'C']
```

```
df2.head(5)
```

	Unnamed: 0	Gender	Age	Ever_Married	Family_Size	Profession	Graduated	Work_Experience	Energy_Consumption	Preferred_
0	0	Female	36	Yes	1.0	Engineer	Yes	0.0	Low	
1	1	Male	37	Yes	4.0	Healthcare	Yes	8.0	Average	
2	2	Female	69	Yes	1.0	NaN	No	0.0	Low	
3	3	Male	59	Yes	2.0	Executive	No	11.0	High	
4	4	Female	19	No	4.0	Marketing	No	NaN	Low	

Next steps: Generate code with df2 View recommended plots

```
df2['Group'] = predictions
df2.head(5)
```

	Unnamed: 0	Gender	Age	Ever_Married	Family_Size	Profession	Graduated	Work_Experience	Energy_Consumption	Preferred_
0	0	Female	36	Yes	1.0	Engineer	Yes	0.0	Low	
1	1	Male	37	Yes	4.0	Healthcare	Yes	8.0	Average	
2	2	Female	69	Yes	1.0	NaN	No	0.0	Low	
3	3	Male	59	Yes	2.0	Executive	No	11.0	High	
4	4	Female	19	No	4.0	Marketing	No	NaN	Low	

Next steps: Generate code with df2 View recommended plots

```
with open("Edited_Consumer_Test_Dataset.csv", "wb") as f:
  f.write(df2.to_csv(index=False).encode("utf-8"))
  print("csv file created")
```

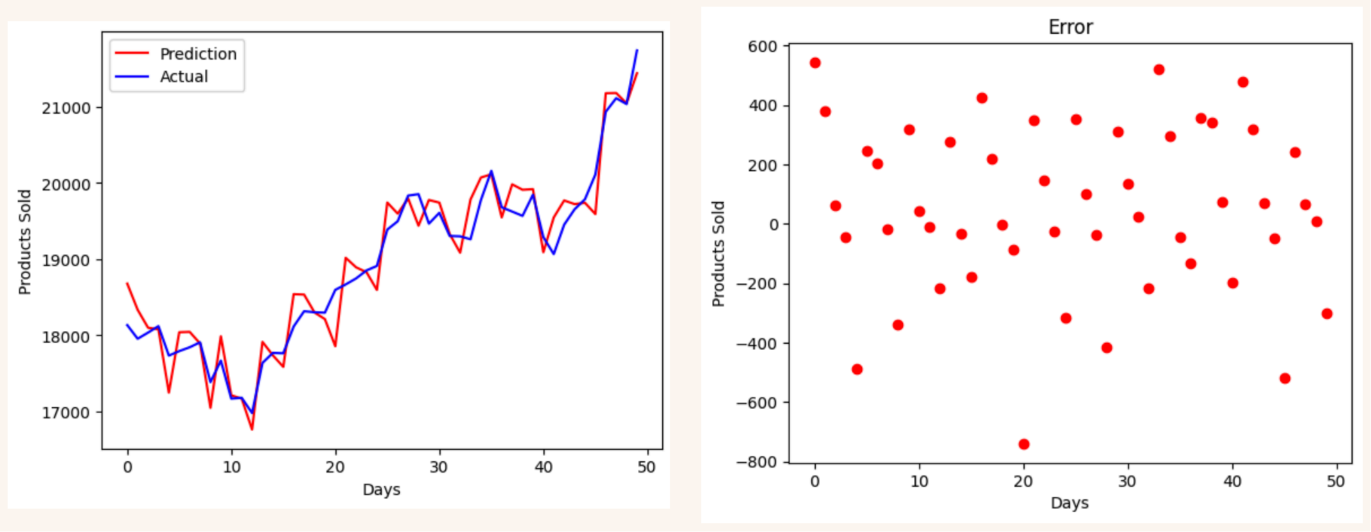
```
csv file created
```

Q4: In the above case your team already had data from which customers were classified in groups. How will you predict the classes if the groups and it’s data weren’t available? State the method you would have used.

Instead of deploying a Deep Learning model which predicts the group for the customers, I would try to find some pattern among the customers and using that write code to distribute them in the four classes

DAV Assignment Q6

After launching your renewable energy products in the Indian market, you notice a positive response from customers. Sales are increasing steadily, and the demand for your products is reaching new heights. To ensure your company can meet this growing demand, you need to predict the number of goods that can be produced over time. This forecast will help you make informed decisions about scaling operations and supply chain management. Given that your company relies on renewable energy sources, daily production can vary based on factors like weather, season, and other variables. Your team has developed a model that accounts for these factors, but its performance on test data has been disappointing, as shown in the accompanying figure.



Q6: What do you think might be causing the poor performance of the model? To improve the model' s accuracy, what steps would you take? Provide a detailed justification for each of your proposed methods.

Causes

1. Model Complexity - The model used might be too simple to capture the underlying patterns in the data.
2. Data Quality and Quantity - The data might have some noise or missing values. Also, the amount of data available might not be sufficient to train a robust model.
3. Data Preprocessing - The data may not have been cleaned, reduced and transform sufficiently.
4. Incorrect Hyperparameter Tuning - The hyperparameters may not have been tuned to achive optimal performance.

Steps to take

1. Increase the Model Complexity
   - Instead of a simple linear regression or a basic time series model, consider using more advanced models like random forests, decision tree or logistic regression models.
   - Use advanced time series models like LSTM.
2. Encorporate More Features
   - Add more features to the dataset.
   - Normalise or transform the features to improve the model's performance.
3. Improve data quality
   - Handle missing values appropriately. Assign the average value of the column to the null field.
   - Augment existing data to create more data.
4. Model tuning
   - Use techniques like grid search or random search to find the optimal hyperparameters.

DAV Assignment Q7

You have been provided with a dataset containing emails categorized as either spam or non-spam and another dataset with emails not yet categorized.

Q7: Develop a model to predict whether each email is spam or not, and use it to classify the uncategorized emails.

```
# Import libraries
import requests
from pathlib import Path
import zipfile
import os


request = requests.get("https://github.com/PanavShah1/DAV_assignment/blob/main/Email_Dataset.csv?raw=true")
with open("Email_Dataset.csv", "wb") as f:
  f.write(request.content)
  print("csv file downloaded")
```

```
csv file downloaded
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("Email_Dataset.csv")
df.head(5)
```

	Unnamed: 0	CATEGORY	MESSAGE
0	1000	Spam	\n\nThe Internet's Online Pharmacy\n\n\n\nViag...
1	1001	Spam	------=_NextPart_000_00B0_35C58D0E.D7267B06\n\...
2	1002	Spam	<html>\n\n\n\n<head>\n\n<meta http-equiv="Cont...
3	1003	Spam	------=_NextPart_000_00E4_86E61E0A.B5488E11\n\...
4	1004	Spam	BARRISTER ADEWALE COKER CHAMBERS\n\nLegal Prac...

Next steps: Generate code with df   View recommended plots

```
import string
def remove_angle_brackets(text):
  return pd.Series(text).str.replace(r'<(.|\n)*>', '', regex=True)

def remove_new_line(text):
  return pd.Series(text).str.replace(r'\n', '', regex=True)

punctuation_list = string.punctuation
def remove_punctuations(text):
    temp = str.maketrans('', '', punctuation_list)
    return text.translate(temp)

df['MESSAGE'] = df['MESSAGE'].apply(remove_angle_brackets)
df['MESSAGE'] = df['MESSAGE'].apply(remove_new_line)
df['MESSAGE'] = df['MESSAGE'].apply(remove_punctuations)
df.drop('Unnamed: 0', axis=1, inplace=True)
df.replace('Spam', value=1, inplace=True)
df.replace('Not Spam', value=0, inplace=True)
df.head(10)
```

	CATEGORY	MESSAGE
0	1	The Internets Online PharmacyViagra Xenical ...
1	1	NextPart00000B035C58D0ED7267B06ContentType tex...
2	1	httpxentcommailmanlistinfofork
3	1	NextPart00000E486E61E0AB5488E11ContentType tex...
4	1	BARRISTER ADEWALE COKER CHAMBERSLegal Practiti...
5	1	DeathToSpamDeathToSpamDeathToSpamThis sfnet em...
6	1	
7	1	
8	1	Just sent you a note with the wrong link Tthe ...
9	1	This is a multipart message in MIME formatNext...

Next steps: Generate code with df View recommended plots

```
X = df['MESSAGE']
y = df['CATEGORY']
X.shape, y.shape
```

```
((4000,), (4000,))
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((3200,), (800,), (3200,), (800,))
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=1, stop_words='english', lowercase=True)
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)
X_train_vectorized.shape, X_test_vectorized.shape
```

```
((3200, 59378), (800, 59378))
```

```
y_train = y_train.astype('int')
y_test = y_test.astype('int')
```

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
```

```
model.fit(X_train_vectorized, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
train_predictions = model.predict(X_train_vectorized)
train_predictions[:10]
```

```
array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0])
```

```
train_accuracy = model.score(X_train_vectorized, y_train)
train_accuracy
```

```
0.9134375
```

```
test_predictions = model.predict(X_test_vectorized)
test_predictions[:10]
```

```
array([1, 0, 1, 0, 0, 0, 0, 1, 0, 1])
```

```
test_accuracy = model.score(X_test_vectorized, y_test)
test_accuracy
```

```
0.8825
```

Predict

```
request = requests.get("https://github.com/PanavShah1/DAV_assignment/blob/main/Email%20Test%20Data.csv?raw=true")
with open("Email_Test_Data.csv", "wb") as f:
  f.write(request.content)
  print("csv file downloaded")
```

```
csv file downloaded
```

```
df1 = pd.read_csv("Email_Test_Data.csv")
df1.head(5)
```

	Unnamed: 0	MESSAGE
0	0	Dear Homeowner,\n\n \n\nInterest Rates are at ...
1	1	ATTENTION: This is a MUST for ALL Computer Use...
2	2	This is a multi-part message in MIME format.\n...
3	3	IMPORTANT INFORMATION:\n\n\n\nThe new domain n...
4	4	This is the bottom line. If you can GIVE AWAY...

Next steps: Generate code with df1 View recommended plots

```
from copy import deepcopy
df2 = deepcopy(df1)
```

```
df1['MESSAGE'] = df1['MESSAGE'].apply(remove_angle_brackets)
df1['MESSAGE'] = df1['MESSAGE'].apply(remove_new_line)
df1['MESSAGE'] = df1['MESSAGE'].apply(remove_punctuations)
df1.head(5)
```

	Unnamed: 0	MESSAGE
0	0	Dear Homeowner Interest Rates are at their low...
1	1	ATTENTION This is a MUST for ALL Computer User...
2	2	This is a multipart message in MIME formatNext...
3	3	IMPORTANT INFORMATIONThe new domain names are ...
4	4	This is the bottom line If you can GIVE AWAY ...

Next steps: Generate code with df1 View recommended plots

```
X1 = df1['MESSAGE']
X1.shape
```

```
(1000,)
```

```
X1_vectorized = vectorizer.transform(X1)
```

```
spam_predictions_binary = model.predict(X1_vectorized)
```

```
spam_predictions = [0 for i in range(len(spam_predictions_binary))]

for i, x in enumerate(spam_predictions_binary):
  if x == 1:
    spam_predictions[i] = 'Spam'
  else:
    spam_predictions[i] = 'Not Spam'

spam_predictions[:10]
```

```
['Spam',
 'Not Spam',
 'Spam',
 'Spam',
 'Not Spam',
 'Spam',
 'Not Spam',
 'Not Spam',
 'Not Spam',
 'Not Spam']
```

```
df2['CATEGORY'] = spam_predictions
df2.head(5)
```

	Unnamed: 0	MESSAGE	CATEGORY
0	0	Dear Homeowner,\n\n \n\nInterest Rates are at ...	Spam
1	1	ATTENTION: This is a MUST for ALL Computer Use...	Not Spam
2	2	This is a multi-part message in MIME format.\n...	Spam
3	3	IMPORTANT INFORMATION:\n\n\n\nThe new domain n...	Spam
4	4	This is the bottom line. If you can GIVE AWAY...	Not Spam

Next steps: Generate code with df2 View recommended plots

```
with open('Edited_Email_Test_Data.csv', 'wb') as f:
  f.write(df2.to_csv(index=False).encode('utf-8'))
  print("csv file created")
```

```
csv file created
```