Data preparation examples

2021-22 coursework specification

The specification has been modified this year. In particular I have tried to avoid using 'data preparation and exploration' as this seemed to cause confusion last year.

Last year's specification:

Data exploration and preparation

This aspect requires you to write Python code as well as document your work in the README.md.

Document the steps you took to explore and clean the data. Describe and explain your findings and the decisions that you took as a result.

While exploring the data you may wish to consider the data in terms of its size, shape, data types etc as well as creating some exploratory charts. Save any charts you create as an image file (e.g. jpg, png) and include the image in the README.

To clean and prepare the data you will need to consider aspects such as missing data, outliers, text fields etc. You may also need to wrangle and parse out important information held in the original data to ensure the structure of the resulting data can more easily be used by your Dashboard application.

Write Python code to explore, clean and structure the data. Empty files are included in the repo though you can use as many or few files as you wish.

Consider the quality of the code. Does it adhere to python naming convention? Are function and method names meaningful? Is the code easy to understand?

Save both the initial data set and the cleaned and prepared data set in your repository. Most students are likely to save this as a .csv or .xlsx file.

Good response

Tutor comment

I am not going to put the full code for a good response since it would be far too easy to simply follow that student's structure and just change it slightly. Instead I am providing two extracts from it to illustrate why their code was good.

The students' code quality was high and their docstrings exemplary.

Note: The student used a Jupyter notebook which isn't allowed this year. The text would have been written in markdown.

Import required libraries

In [ ]:
```
import os
import datetime
import json

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from collections import OrderedDict
from dateutil.parser import parse
```

Exploration of meta_data.csv

Loading meta_data.csv using a pandas data frame

In [ ]:
```
meta_data = pd.read_csv(os.path.join("./data/", "meta_data.csv"), encoding='unicode_escape')
```

Number of rows and columns in meta_data.csv

In [ ]:
```
print("Number of rows:", meta_data.shape[0], "\nNumber of columns:", meta_data.shape[1])
```

First 5 rows of meta_data.csv

In [ ]:
```
meta_data.head()
```

By observing the first 5 rows of the data, it seems as if meta_data gives information regarding the characteristics of the pod, for example, the name/kind of location or the pollutant it senses.

Column labels, null values and data types

In [ ]:
```
meta_data.info()
```

There are 16 columns and a maximum of 181 rows in meta_data.

Only `relocate_date_utc` and `distance_from_road` have missing values as they have 18 and 170 non-null values, respectively, while the other columns have 181. These columns, along with the remaining columns, are explored in the following section to determine if they are useful.

Exploration of all columns in meta_data to determine if they are useful

pod_id_location

In [ ]:
```
meta_data.sort_values(["pod_id_location"]).head()
```

Inferring from the name, `pod_id_location` seems to be the unique identifier of each pod's location. From the first 5 lines of the data sorted with respect to `pod_id_location`, it can be observed that a single `pod_id_location` has the same `latitude` and `longitude` but different `pollutant`. This could imply that each pod could sense 2 different gases.

The following is done to determine the number of times the same `pod_id_location` has occured:

In [ ]:
```
meta_data.value_counts(["pod_id_location"])
```

There are 106 unique values of `pod_id_location`. According to the website, there are 100 pods placed in London, which is close to the number of unique values `pod_id_location` that was acquired. Therefore, it is safe to conclude that that `pod_id_location` is an identifier for location. However, to uniquely identify each pod's location, `latitude` and `logitude` are better features to use as they are more universal. Hence, this column is classified as **not useful**.

start_date_utc

In [ ]:
```
meta_data.sort_values(["start_date_utc"]).head()
```

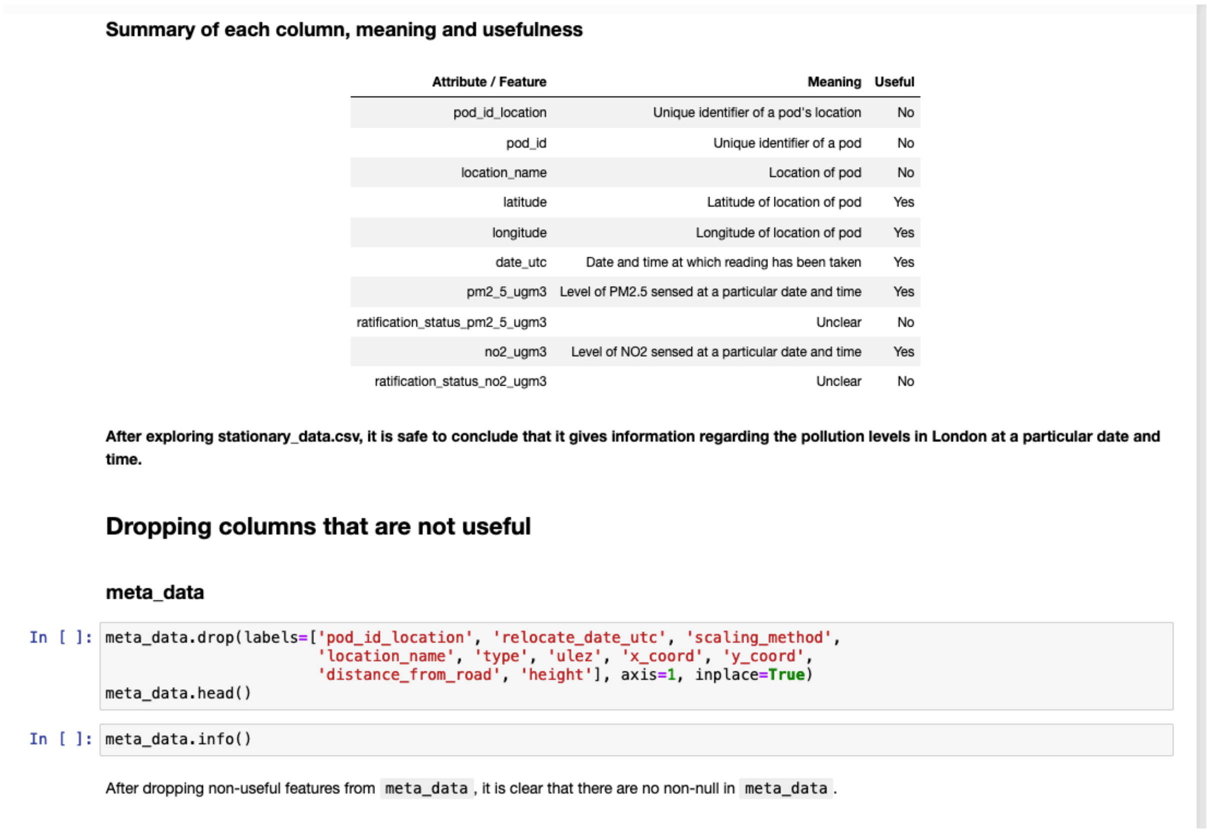
The first 5 lines of data sorted with respect to `start_date_utc` does not offer much insight.

In [ ]:
```
meta_data.value_counts(["start_date_utc"])
```

There are 138 unique values of `start_date_utc`. Although, this does not offer much insight once again, the significance of this column can be inferred

The student continued to systematically understand and explore the data. Ultimately they didn't need to deal with many issues however they still used the pandas functions to determine that this wasn't necessary.

An example of a decision they took based on their exploration of the data is illustrated in the next screenshot. Here they decide which data to keep based on their preceding analysis and then took steps to remove the data they didn't need.



Summary of each column, meaning and usefulness

Attribute / Feature	Meaning	Useful
pod_id_location	Unique identifier of a pod's location	No
pod_id	Unique identifier of a pod	No
location_name	Location of pod	No
latitude	Latitude of location of pod	Yes
longitude	Longitude of location of pod	Yes
date_utc	Date and time at which reading has been taken	Yes
pm2_5_ugm3	Level of PM2.5 sensed at a particular date and time	Yes
ratification_status_pm2_5_ugm3	Unclear	No
no2_ugm3	Level of NO2 sensed at a particular date and time	Yes
ratification_status_no2_ugm3	Unclear	No

After exploring stationary_data.csv, it is safe to conclude that it gives information regarding the pollution levels in London at a particular date and time.

Dropping columns that are not useful

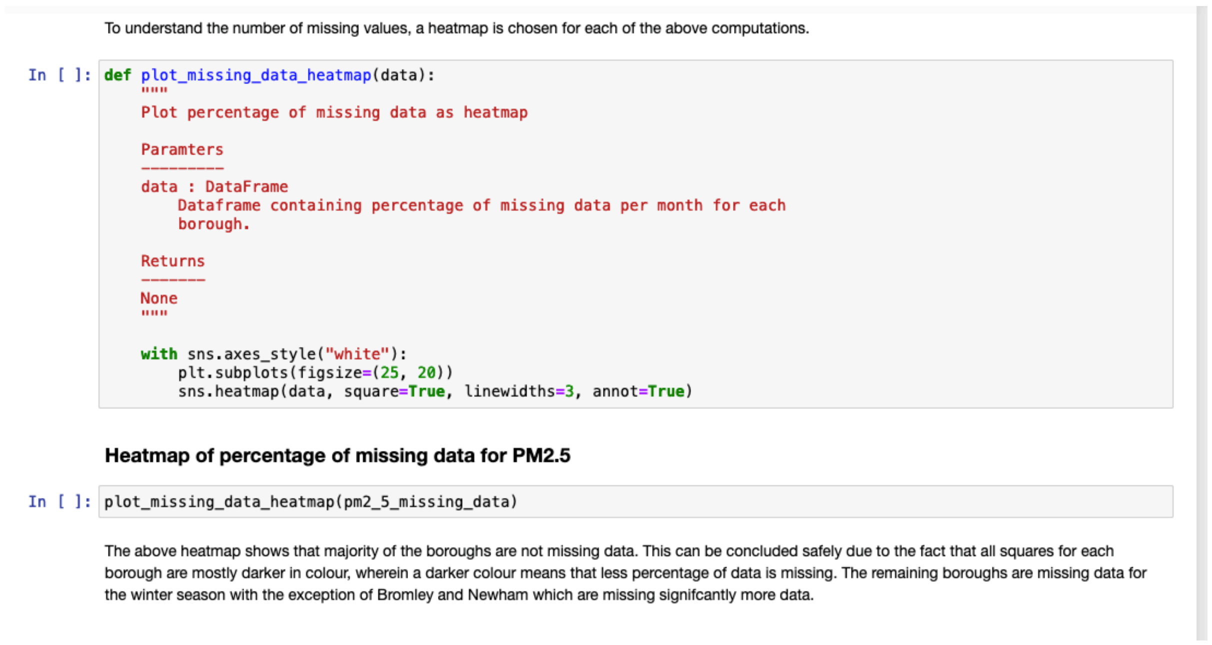
meta_data

```
In [ ]: meta_data.drop(labels=['pod_id_location', 'relocate_date_utc', 'scaling_method',
                               'location_name', 'type', 'ulez', 'x_coord', 'y_coord',
                               'distance_from_road', 'height'], axis=1, inplace=True)
        meta_data.head()
```

```
In [ ]: meta_data.info()
```

After dropping non-useful features from `meta_data`, it is clear that there are no non-null in `meta_data`.

This student was exceptional and from this point when far beyond the course expectations. However, I've included the next screenshot as while there is a lot of code you don't see, what the student did was to create a chart that helped to understand the data. In this case to analyse missing data:


To understand the number of missing values, a heatmap is chosen for each of the above computations.

```
In [ ]: def plot_missing_data_heatmap(data):
            """
            Plot percentage of missing data as heatmap

            Paramters
            ---------
            data : DataFrame
                Dataframe containing percentage of missing data per month for each
                borough.

            Returns
            -------
            None
            """

            with sns.axes_style("white"):
                plt.subplots(figsize=(25, 20))
                sns.heatmap(data, square=True, linewidths=3, annot=True)
```

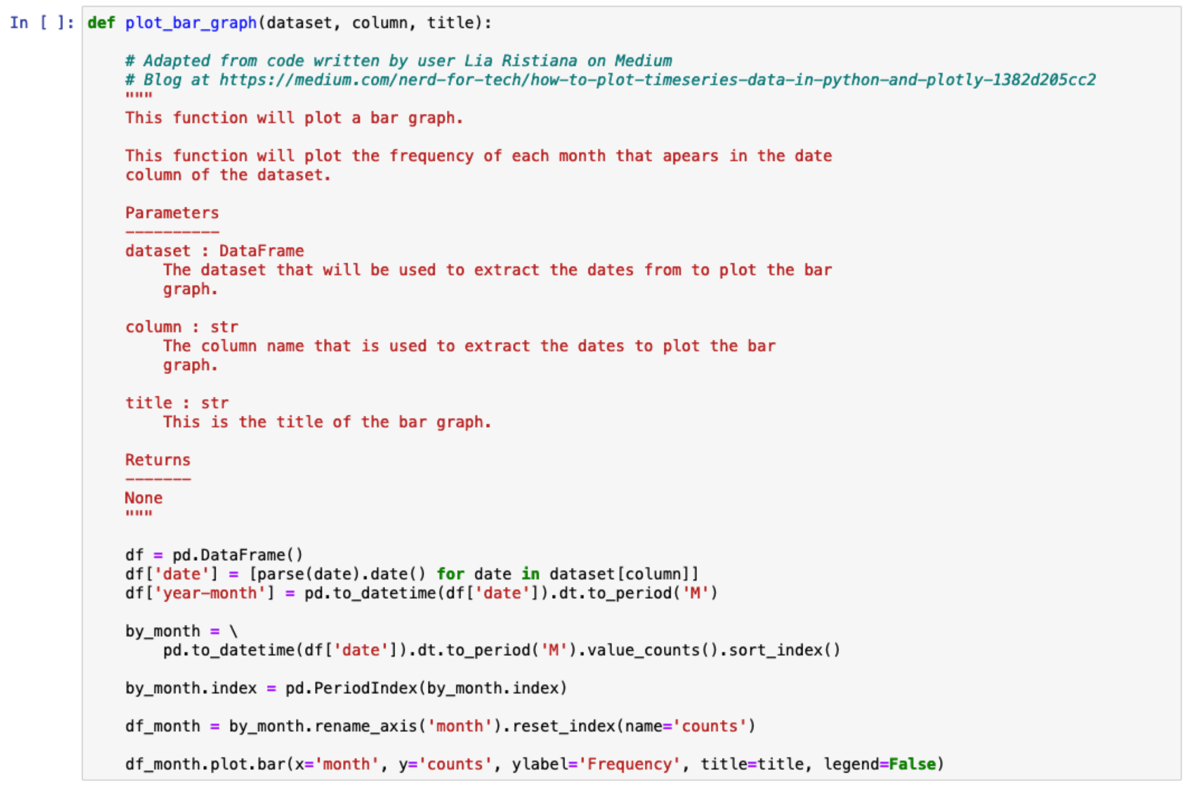
Heatmap of percentage of missing data for PM2.5

```
In [ ]: plot_missing_data_heatmap(pm2_5_missing_data)
```

The above heatmap shows that majority of the boroughs are not missing data. This can be concluded safely due to the fact that all squares for each borough are mostly darker in colour, wherein a darker colour means that less percentage of data is missing. The remaining boroughs are missing data for the winter season with the exception of Bromley and Newham which are missing signifcantly more data.

This was a good use of a chart used to prepare and understand the data, rather than to try and answer questions for the target audience (see weak response).

Finally, this is an example of one of their functions with docstrings and also includes an appropriate reference in the code:



```
In [ ]: def plot_bar_graph(dataset, column, title):

            # Adapted from code written by user Lia Ristiana on Medium
            # Blog at https://medium.com/nerd-for-tech/how-to-plot-timeseries-data-in-python-and-plotly-1382d205cc2
            """
            This function will plot a bar graph.

            This function will plot the frequency of each month that apears in the date
            column of the dataset.

            Parameters
            ----------
            dataset : DataFrame
                The dataset that will be used to extract the dates from to plot the bar
                graph.

            column : str
                The column name that is used to extract the dates to plot the bar
                graph.

            title : str
                This is the title of the bar graph.

            Returns
            -------
            None
            """

            df = pd.DataFrame()
            df['date'] = [parse(date).date() for date in dataset[column]]
            df['year-month'] = pd.to_datetime(df['date']).dt.to_period('M')

            by_month = \
                pd.to_datetime(df['date']).dt.to_period('M').value_counts().sort_index()

            by_month.index = pd.PeriodIndex(by_month.index)

            df_month = by_month.rename_axis('month').reset_index(name='counts')

            df_month.plot.bar(x='month', y='counts', ylabel='Frequency', title=title, legend=False)
```

Weak response

The student misunderstood what was required for the data preparation and exploration and instead tried to create charts to answer the questions which was not what was required. Their code was a little difficult to follow in places. They could have used more meaningful variable names rather than df1.. df58.

The student wrote a lot of code but didn't really use any of the pandas functions to examine the structure of the data, the data types, any missing data, any fields that needed to be added/removed etc.

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

#Firstly read through the csv file, only read until the row that still contains values.
df = pd.read_csv('Data1.csv', skiprows=0, nrows=35)

#Define the country specifed cost percentage values
#This is for Australia Cost Percentage
df1 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Cost (% of Warehouse
value)') & (
            df['Country Name'] == 'Australia')]
df2 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Australia': [df1['2016'], df1['2017'], df1['2018'], df1['2019'], df1['2020']]})

#This is for China Cost Percentage
df3 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Cost (% of Warehouse
value)') & (
            df['Country Name'] == 'China')]
df4 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'China': [df3['2016'], df3['2017'], df3['2018'], df3['2019'], df3['2020']]})

#This is for France Cost Percentage
df5 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Cost (% of Warehouse
value)') & (
            df['Country Name'] == 'France')]
df6 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'France': [df5['2016'], df5['2017'], df5['2018'], df5['2019'], df5['2020']]})

#This is for Germany Cost Percentage
df7 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Cost (% of Warehouse
value)') & (
            df['Country Name'] == 'Germany')]
df8 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Germany': [df7['2016'], df7['2017'], df7['2018'], df7['2019'], df7['2020']]})

#This is for Japan Cost Percentage
df9 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Cost (% of Warehouse
value)') & (
            df['Country Name'] == 'Japan')]
df10 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Japan': [df9['2016'], df9['2017'], df9['2018'], df9['2019'], df9['2020']]})

#This is for United Kingdom Cost Percentage
df11 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Cost (% of Warehouse
value)') & (
            df['Country Name'] == 'United Kingdom')]
df12 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'United Kingdom': [df11['2016'], df11['2017'], df11['2018'], df11['2019'],
df11['2020']]})

#This is for United States Cost Percentage
df13 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Cost (% of Warehouse
value)') & (
            df['Country Name'] == 'United States')]
df14 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'United States': [df13['2016'], df13['2017'], df13['2018'], df13['2019'], df13['2020']]})

#Use matplotlib.pyplot library to plot the lines for the dataframe defined above
#All the basic elements: x-axis name, y-axis name, title name, legend, label name, the marker
to show the value
plt.figure(figsize=(10,6), dpi=120)
plt.plot(df2.Year, df2.Australia, label="Australia", marker='.')
plt.plot(df4.Year, df4.China, label="China", marker='.')
plt.plot(df6.Year, df6.France, label="France", marker='.')
plt.plot(df8.Year, df8.Germany, label="Germany", marker='.')
plt.plot(df10.Year, df10.Japan, label="Japan", marker='.')
plt.plot(df12.Year, df12['United Kingdom'], label="United Kingdom", marker='.')
plt.plot(df14.Year, df14['United States'], label="United States", marker='.')
plot1 = plt.figure(1)
plt.legend()
plt.xlabel('Years')
plt.ylabel('Cost Percentage')
plt.title('Dealing with construction permits: Cost (% of Warehouse value)',
          fontdict ={'fontname':'Comic Sans Ms'})


#Use numpy function to define the xticks function as it doesn't make sense to show decimal
numbers for years
yearvalue = df2.Year
array2 = np.array(yearvalue)
plt.xticks(array2)
plt.show()


#Define the country specifed Dealing with construction permits: Time (days)
#This is for Australia Time (days)
df15 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Time (days)') & (
            df['Country Name'] == 'Australia')]
df16 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Australia': [df15['2016'], df15['2017'], df15['2018'], df15['2019'], df15['2020']]})

#This is for China Time (days)
df17 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Time (days)') & (
            df['Country Name'] == 'China')]
df18 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'China': [df17['2016'], df17['2017'], df17['2018'], df17['2019'], df17['2020']]})

#This is for France Time (days)
df19 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Time (days)') & (
            df['Country Name'] == 'France')]
df20 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'France': [df19['2016'], df19['2017'], df19['2018'], df19['2019'], df19['2020']]})

#This is for Germany Time (days)
df21 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Time (days)') & (
            df['Country Name'] == 'Germany')]
df22 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Germany': [df21['2016'], df21['2017'], df21['2018'], df21['2019'], df21['2020']]})

#This is for Japan Time (days)
df23 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Time (days)') & (
            df['Country Name'] == 'Japan')]
df24 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Japan': [df23['2016'], df23['2017'], df23['2018'], df23['2019'], df23['2020']]})

#This is for United Kingdom Time (days)
df25 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Time (days)') & (
            df['Country Name'] == 'United Kingdom')]
df26 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'United Kingdom': [df25['2016'], df25['2017'], df25['2018'], df25['2019'],
df25['2020']]})

#This is for United States Time (days)
df27 = df.loc[(df['Series Name'] == 'Dealing with construction permits: Time (days)') & (
            df['Country Name'] == 'United States')]
df28 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'United States': [df27['2016'], df27['2017'], df27['2018'], df27['2019'], df27['2020']]})

#Use matplotlib.pyplot library to plot the lines for the dataframe defined above
#All the basic elements: x-axis name, y-axis name, title name, legend, label name, the marker
to show the value
```

```
plt.figure(figsize=(10,6), dpi=120)
plt.plot(df16.Year, df16.Australia, label="Australia", marker='.')
plt.plot(df18.Year, df18.China, label="China", marker='.')
plt.plot(df20.Year, df20.France, label="France", marker='.')
plt.plot(df22.Year, df22.Germany, label="Germany", marker='.')
plt.plot(df24.Year, df24.Japan, label="Japan", marker='.')
plt.plot(df26.Year, df26['United Kingdom'], label="United Kingdom", marker='.')
plt.plot(df28.Year, df28['United States'], label="United States", marker='.')

plt.legend()
plt.xlabel('Years')
plt.ylabel('Days')
plt.title('Dealing with construction permits: Time (days))',
          fontdict ={'fontname':'Comic Sans Ms'})


#Use numpy function to define the xticks function as it doesn't make sense to show decimal
numbers for years
yearvalue = df2.Year
array2 = np.array(yearvalue)
plt.xticks(array2)
plt.show()


#Define the country specifed Cost to get electricity (% of income per capita)
#This is for Australia Cost to get electricity
df29 = df.loc[(df['Series Name'] == 'Getting electricity: Cost to get electricity (% of
income per capita)') & (
            df['Country Name'] == 'Australia')]
df30 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Australia': [df29['2016'], df29['2017'], df29['2018'], df29['2019'], df29['2020']]})

#This is for China Cost to get electricity
df31 = df.loc[(df['Series Name'] == 'Getting electricity: Cost to get electricity (% of
income per capita)') & (
            df['Country Name'] == 'China')]
df32 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'China': [df31['2016'], df31['2017'], df31['2018'], df31['2019'], df31['2020']]})

#This is for France Cost to get electricity
df33 = df.loc[(df['Series Name'] == 'Getting electricity: Cost to get electricity (% of
income per capita)') & (
            df['Country Name'] == 'France')]
df34 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'France': [df33['2016'], df33['2017'], df33['2018'], df33['2019'], df33['2020']]})

#This is for Germany Cost to get electricity
df35 = df.loc[(df['Series Name'] == 'Getting electricity: Cost to get electricity (% of
income per capita)') & (
            df['Country Name'] == 'Germany')]
df36 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Germany': [df35['2016'], df35['2017'], df35['2018'], df35['2019'], df35['2020']]})

#This is for Japan Cost to get electricity
df37 = df.loc[(df['Series Name'] == 'Getting electricity: Cost to get electricity (% of
income per capita)') & (
            df['Country Name'] == 'Japan')]
df38= pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Japan': [df37['2016'], df37['2017'], df37['2018'], df37['2019'], df37['2020']]})

#This is for United Kingdom Cost to get electricity
df39 = df.loc[(df['Series Name'] == 'Getting electricity: Cost to get electricity (% of
income per capita)') & (
            df['Country Name'] == 'United Kingdom')]
df40 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'United Kingdom': [df39['2016'], df39['2017'], df39['2018'], df39['2019'],
df39['2020']]})

#This is for United States Cost to get electricity
df41 = df.loc[(df['Series Name'] == 'Getting electricity: Cost to get electricity (% of
income per capita)') & (
            df['Country Name'] == 'United States')]
df42 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'United States': [df41['2016'], df41['2017'], df41['2018'], df41['2019'], df41['2020']]})

#Use matplotlib.pyplot library to plot the lines for the dataframe defined above
#All the basic elements: x-axis name, y-axis name, title name, legend, label name, the marker
to show the value
plt.figure(figsize=(10,6), dpi=120)
plt.plot(df16.Year, df30.Australia, label="Australia", marker='.')
plt.plot(df18.Year, df32.China, label="China", marker='.')
plt.plot(df20.Year, df34.France, label="France", marker='.')
plt.plot(df22.Year, df36.Germany, label="Germany", marker='.')
plt.plot(df24.Year, df38.Japan, label="Japan", marker='.')
plt.plot(df26.Year, df40['United Kingdom'], label="United Kingdom", marker='.')
plt.plot(df28.Year, df42['United States'], label="United States", marker='.')

plt.legend()
plt.xlabel('Years')
plt.ylabel('Cost Percentage')
plt.title('Getting electricity: Cost to get electricity (% of income per capita)',
          fontdict ={'fontname':'Comic Sans Ms'})


#Use numpy function to define the xticks function as it doesn't make sense to show decimal
numbers for years
yearvalue = df2.Year
array2 = np.array(yearvalue)
plt.xticks(array2)
plt.show()


#Define the country specifed Paying taxes: Total tax and contribution rate (% of profit)
#This is for Australia Total tax and contribution rate
df43 = df.loc[(df['Series Name'] == 'Paying taxes: Total tax and contribution rate (% of
profit)') & (
            df['Country Name'] == 'Australia')]
df44 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Australia': [df43['2016'], df43['2017'], df43['2018'], df43['2019'], df43['2020']]})

#This is for China Total tax and contribution rate
df45 = df.loc[(df['Series Name'] == 'Paying taxes: Total tax and contribution rate (% of
profit)') & (
            df['Country Name'] == 'China')]
df46 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'China': [df45['2016'], df45['2017'], df45['2018'], df45['2019'], df45['2020']]})

#This is for France Total tax and contribution rate
df47 = df.loc[(df['Series Name'] == 'Paying taxes: Total tax and contribution rate (% of
profit)') & (
            df['Country Name'] == 'France')]
df48 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'France': [df47['2016'], df47['2017'], df47['2018'], df47['2019'], df47['2020']]})

#This is for Germany Total tax and contribution rate
df49 = df.loc[(df['Series Name'] == 'Paying taxes: Total tax and contribution rate (% of
profit)') & (
            df['Country Name'] == 'Germany')]
df50 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Germany': [df49['2016'], df49['2017'], df49['2018'], df49['2019'], df49['2020']]})

#This is for Japan Total tax and contribution rate
df51 = df.loc[(df['Series Name'] == 'Paying taxes: Total tax and contribution rate (% of
profit)') & (
            df['Country Name'] == 'Japan')]
df52= pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Japan': [df51['2016'], df51['2017'], df51['2018'], df51['2019'], df51['2020']]})

#This is for United Kingdom Total tax and contribution rate
df53 = df.loc[(df['Series Name'] == 'Paying taxes: Total tax and contribution rate (% of
profit)') & (
            df['Country Name'] == 'United Kingdom')]
```

```
df54 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'United Kingdom': [df53['2016'], df53['2017'], df53['2018'], df53['2019'],
df53['2020']]})

#This is for United States Total tax and contribution rate
df55 = df.loc[(df['Series Name'] == 'Paying taxes: Total tax and contribution rate (% of
profit)') & (
            df['Country Name'] == 'United States')]
df56 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'United States': [df55['2016'], df55['2017'], df55['2018'], df55['2019'], df55['2020']]})

#Use matplotlib.pyplot library to plot the lines for the dataframe defined above
#All the basic elements: x-axis name, y-axis name, title name, legend, label name, the marker
to show the value
plt.figure(figsize=(10,6), dpi=120)
plt.plot(df16.Year, df44.Australia, label="Australia", marker='.')
plt.plot(df18.Year, df46.China, label="China", marker='.')
plt.plot(df20.Year, df48.France, label="France", marker='.')
plt.plot(df22.Year, df50.Germany, label="Germany", marker='.')
plt.plot(df24.Year, df52.Japan, label="Japan", marker='.')
plt.plot(df26.Year, df54['United Kingdom'], label="United Kingdom", marker='.')
plt.plot(df28.Year, df56['United States'], label="United States", marker='.')

plt.legend()
plt.xlabel('Years')
plt.ylabel('% of profit')
plt.title('Paying taxes: Total tax and contribution rate (% of profit)',
          fontdict ={'fontname':'Comic Sans Ms'})


#Use numpy function to define the xticks function as it doesn't make sense to show decimal
numbers for years
yearvalue = df2.Year
array2 = np.array(yearvalue)
plt.xticks(array2)
plt.show()

############################################################################################
###
############################################################################################
###
############################################################################################
###

#Define the country specifed Trading across borders
#This is for Australia Time to import
df57 = df.loc[(df['Series Name'] == 'Trading across borders: Time to import: Border
compliance (hours) (DB16-20 methodology) - Score') & (
            df['Country Name'] == 'Australia')]
df58 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Australia': [df57['2016'], df57['2017'], df57['2018'], df57['2019'], df57['2020']]})

#This is for China Time to import
df59 = df.loc[(df['Series Name'] == 'Trading across borders: Time to import: Border
compliance (hours) (DB16-20 methodology) - Score') & (
            df['Country Name'] == 'China')]
df60 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'China': [df59['2016'], df59['2017'], df59['2018'], df59['2019'], df59['2020']]})

#This is for France Time to import
df61 = df.loc[(df['Series Name'] == 'Trading across borders: Time to import: Border
compliance (hours) (DB16-20 methodology) - Score') & (
            df['Country Name'] == 'France')]
df62 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'France': [df61['2016'], df61['2017'], df61['2018'], df61['2019'], df61['2020']]})

#This is for Germany Time to import
df63 = df.loc[(df['Series Name'] == 'Trading across borders: Time to import: Border
compliance (hours) (DB16-20 methodology) - Score') & (
            df['Country Name'] == 'Germany')]
df64 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Germany': [df63['2016'], df63['2017'], df63['2018'], df63['2019'], df63['2020']]})

#This is for Japan Time to import
df65 = df.loc[(df['Series Name'] == 'Trading across borders: Time to import: Border
compliance (hours) (DB16-20 methodology) - Score') & (
            df['Country Name'] == 'Japan')]
df66 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'Japan': [df65['2016'], df65['2017'], df65['2018'], df65['2019'], df65['2020']]})

#This is for United Kingdom Time to import
df67 = df.loc[(df['Series Name'] == 'Trading across borders: Time to import: Border
compliance (hours) (DB16-20 methodology) - Score') & (
            df['Country Name'] == 'United Kingdom')]
df68 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'United Kingdom': [df67['2016'], df67['2017'], df67['2018'], df67['2019'],
df67['2020']]})

#This is for United States Time to import
df69 = df.loc[(df['Series Name'] == 'Trading across borders: Time to import: Border
compliance (hours) (DB16-20 methodology) - Score') & (
            df['Country Name'] == 'United States')]
df70 = pd.DataFrame({
    'Year': [2016, 2017, 2018, 2019, 2020],
    'United States': [df69['2016'], df69['2017'], df69['2018'], df69['2019'], df69['2020']]})

#Use matplotlib.pyplot library to plot the lines for the dataframe defined above
#All the basic elements: x-axis name, y-axis name, title name, legend, label name, the marker
to show the value
plt.figure(figsize=(10,6), dpi=120)
plt.plot(df16.Year, df58.Australia, label="Australia", marker='.')
plt.plot(df18.Year, df60.China, label="China", marker='.')
plt.plot(df20.Year, df62.France, label="France", marker='.')
plt.plot(df22.Year, df64.Germany, label="Germany", marker='.')
plt.plot(df24.Year, df66.Japan, label="Japan", marker='.')
plt.plot(df26.Year, df68['United Kingdom'], label="United Kingdom", marker='.')
plt.plot(df28.Year, df70['United States'], label="United States", marker='.')

plt.legend()
plt.xlabel('Years')
plt.ylabel('% of profit')
plt.title('Trading across borders: Time to import: Border compliance (hours) (DB16-20
methodology) - Score)',
          fontdict ={'fontname':'Comic Sans Ms'})


#Use numpy function to define the xticks function as it doesn't make sense to show decimal
numbers for years
yearvalue = df2.Year
array2 = np.array(yearvalue)
plt.xticks(array2)
plt.show()
```