Unemployment rate vs export analysis

DOCUMENTATION

Overview

This document is a summary of the thought process related to task number 2. In this documentation, I will explain the way all the functions I've created work and why I've designed them that way. Some of the functions I've used, in particular, those involved in the data downloading and conversion into a dataset have been already explained in the file "*task_1_thought process.docx*", so in this document, I' will just speak about the modifications.

PROJECT STRUCTURE

This project is structured as follows:

- there is the main file, titled '`main.ipynb`' which includes the project workflow and all the data visualizations
- all the functions I've created to download the data and turn them into a dataset are contained in the file '`manage_data.py`'
- the function I've created to plot the data are contained in the file '`plot_data.py`'
- All the libraries I've used for the project have been installed in a separate virtual environment, you can find the full list in the file '`REQUIREMENTS.txt`'

HOW THE CODE WORKS, OVERVIEW

The main file is divided into n captions, the following list sums up what each caption is competent of:

1. `Import libraries:` here all the libraries are loaded, including the modules I've created.
2. `Import data:` Here the program downloads all the data related to the exports and unemployment rate and loads those data into two separate data frames
3. `Clean and merge the data frames:` Here the program merges the data frames, to do it needs to make the data frames compatible being sure the countries are named the same way.
4. `Visualize data and their correlation:` In this section, the code invokes a function to plot the data along with their correlations.
5. `Study exports and unemployment correlation:` Here in the concluding part of the code I came up with two different methods to try to explain the correlation observed in the data.

manage_data.py

This module contains all the functions used in the main file to download the data from the UN comtrade database and turn them into a data frame.

download_jsons ( )

This function is the same used in task 1, the only That has been modified is the query, to download only the export data, all the other parts of the function are the same so I'll refer to the file "task_1_thought process.docx".

create_df ( )

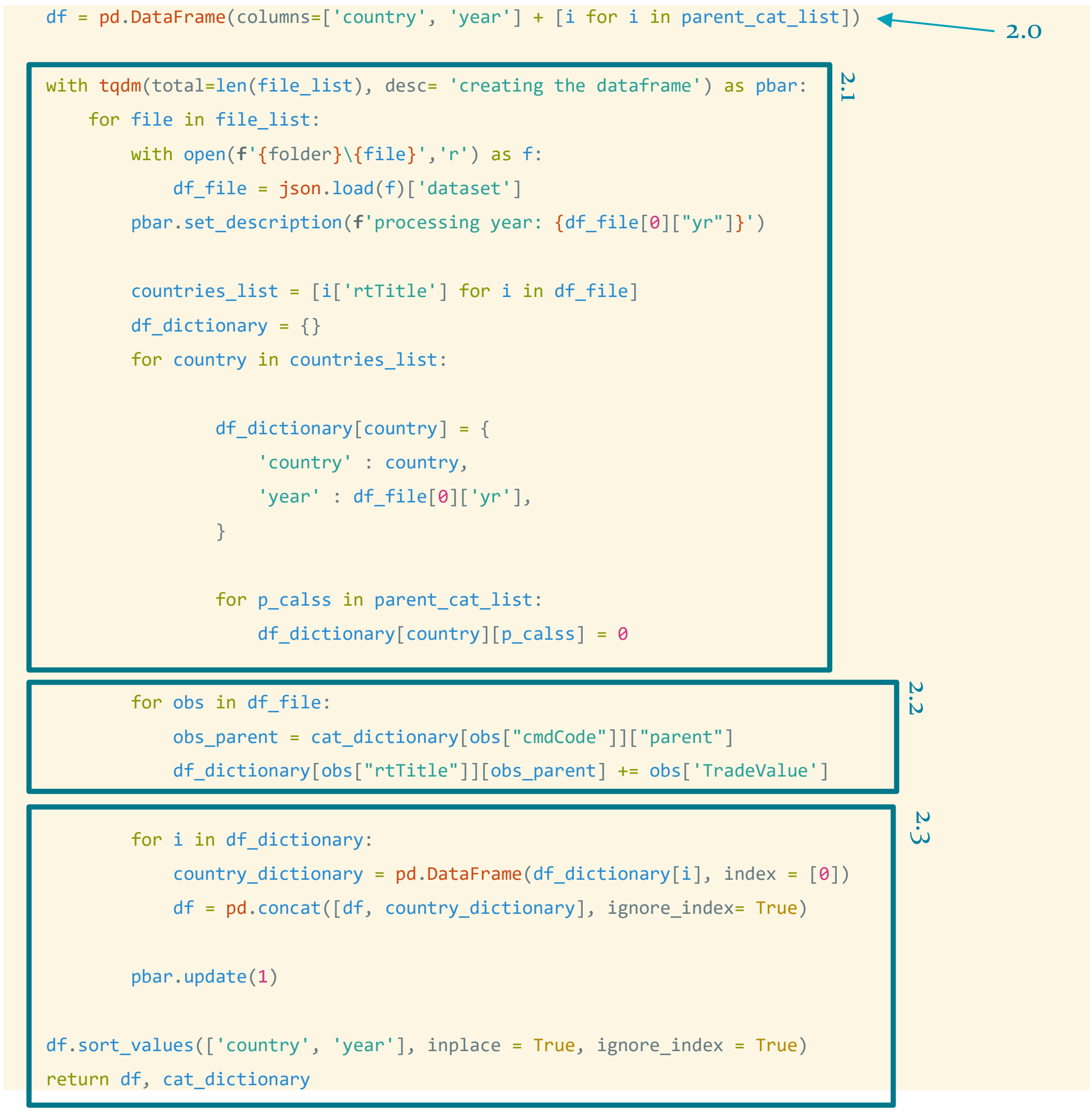
Arguments:

- *folder* (str): this argument passes to the function the name of the folder that contains the files

```
def create_df (folder : str):

    file_list = os.listdir(folder)                                      1.0

    with open('category_hash_map.json') as file:                        1.1
        cat_list = json.load(file)['results']

    cat_dictionary = {}
    for i in cat_list:
        cat_dictionary[i['id']] = {
            'category' : i['text'].split(' ')[0],
            'description' : ' '.join(i['text'].split(' ')[1:]),
            'parent' : i['parent']
        }

    with open('cat_dictionary.json', 'w') as f:                         1.2
        json.dump(cat_dictionary, f, indent=3)
    parent_cat_list = []
    for i in cat_dictionary:
        if cat_dictionary[i]['parent'] == '200' or cat_dictionary[i]['parent'] == '999':
            parent_cat_list.append(i)
```

In this first part, the function takes from the given folder the list of all the files stored (1.0).

Then It loads the hash map of all the categories and creates a dictionary where the keys are the class IDs mapped with their description and parent class (1.1). Next, the function saves the dictionary and creates a list that contains all the parent classes which are the classes that have as a parent the class 200 or 999 (1.2)



```
df = pd.DataFrame(columns=['country', 'year'] + [i for i in parent_cat_list])

with tqdm(total=len(file_list), desc= 'creating the dataframe') as pbar:
    for file in file_list:
        with open(f'{folder}\{file}','r') as f:
            df_file = json.load(f)['dataset']
        pbar.set_description(f'processing year: {df_file[0]["yr"]}')

        countries_list = [i['rtTitle'] for i in df_file]
        df_dictionary = {}
        for country in countries_list:

                df_dictionary[country] = {
                    'country' : country,
                    'year' : df_file[0]['yr'],
                }

                for p_calss in parent_cat_list:
                    df_dictionary[country][p_calss] = 0

        for obs in df_file:
            obs_parent = cat_dictionary[obs["cmdCode"]]["parent"]
            df_dictionary[obs["rtTitle"]][obs_parent] += obs['TradeValue']

        for i in df_dictionary:
            country_dictionary = pd.DataFrame(df_dictionary[i], index = [0])
            df = pd.concat([df, country_dictionary], ignore_index= True)

        pbar.update(1)

df.sort_values(['country', 'year'], inplace = True, ignore_index = True)
return df, cat_dictionary
```

Here the function initializes the data frame (2.0) that will contain the data and goes through each file creating a dictionary for each country that will contain a value for each parent class (initialized at 0) (2.1). Then for each file, the function takes the value of each class and adds it to the cumulative value of the respective parent class (2.2), so at the end, we will have a list of parent classes that are the sum of all the sub-classes. In the last part, the function concatenate the dictionary to the main data frame and return it along with the category dictionary (2.3).

plot_data.py

This module contains the function I've created to plot the data along with the correlations between classes and the unemployment rate.

create_df ( )

this function has been designed to be used in a pywidget so the program passes it to a widgets.interact function along with its arguments.

Arguments:

- `country` (str): this argument passes to the function country data we want to plot.
- `normalized` (bool): If set to True the data will be normalized before they're plotted
- `df` (pd.DataFrame): this argument passes to the function the data frame that contains the data.
- `cat_map` (dictionary): this argument passes to the function the map of all the categories, used to assign the name in the plot label

```
def plot_series(country, normalized, df, cat_map):

    df_to_plot = df.loc[df['country'] == country]                      1.0

    fig, ax = plt.subplots(1,2,figsize=(20,8))                          1.1

    ax[0].set_xlabel('year')
    ax[0].set_ylabel('unemployment')

    if normalized:                                                      1.2
        normalized_data = preprocessing.normalize([df_to_plot['value']])
        ax[0].plot(df_to_plot['year'], normalized_data[0], linewidth=3, ls = '--')

    else:                                                               1.3
        ax[0].plot(df_to_plot['year'], df_to_plot['value'], linewidth=3, ls = '--')
```

In this first part the function extract from the dataset the country we want to plot (1.0), then It initializes the axes (1.1) and plots the unemployment data, if normalized is set to True the function will normalize de data before plotting them (1.2) otherwise It will simply plot them as they are (1.3).

2.0

```
ax2 = ax[0].twinx()
ax2.set_ylabel('exports')
```

2.1

```
cmap = plt.get_cmap('tab10')
colors = [cmap(i/5) for i in range(1,6)]
```

2.2

```
for cat, color in zip(df_to_plot.iloc[:,2:7], colors):
    if normalized:

        normalized_data = preprocessing.normalize([df_to_plot[cat][df_to_plot[cat].notna()]])
        ax2.plot(df_to_plot['year'][df_to_plot[cat].notna()], normalized_data[0], label =
f"({cat}) {cat_map[cat]['description']}", color = color)
```

2.3

```
    else:
        ax2.plot(df_to_plot['year'], df_to_plot[cat], label = f"({cat})
{cat_map[cat]['description']}", color = color)
```

Here the function creates a second ax and links its x-ax to the ax used to plot the unemployment data (2.0) and pick a color map for the exports categories (2.1). Then the function plot the data for each category, if normalized is set to True before doing that It will normalize the data (2.2), otherwise, It will plot the data as they are (2.3).

3.0

```
plt.legend(title = 'exports categories', title_fontproperties = {'weight' : 'bold'})
```

3.1

```
fig.tight_layout()

data_columns = ['value', '205', '236', '245', '253', '268']
corr_mat = df_to_plot[data_columns].corr()
sns.heatmap(corr_mat, ax=ax[1], annot = True, cmap = 'viridis')
```

In the end, the function adds the legend (3.0), calculates the correlation between all the data, and plots them as a haet-map along with the chart (3.1).