Data Quality and Data Wrangling

25.08.2023

Task 2: Scrape the Web

Silva, Bruno – UPS10620925

Professor Qaadan, Sahar

iu INTERNATIONALE HOCHSCHULE

Table of Contents

List of Figures

Introduction

In an era where information is key for company success and innovation, web scraping takes an important role when it comes to acquiring that so valuable information. In many cases, the scraped data is not necessarily information, it is just a lot of raw material impossible for the human brain to understand. It is the job of the data analyst to transform the raw data into readable information to its end user.

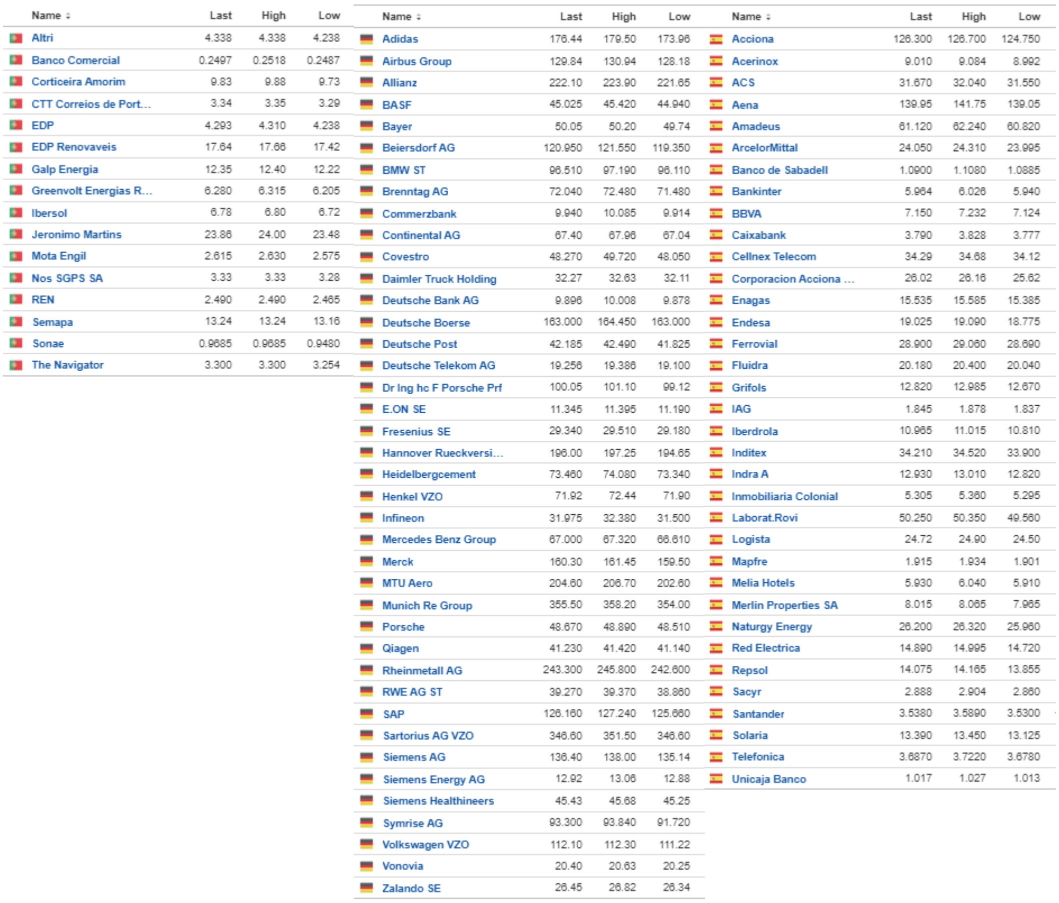
This assignment challenged me to precisely create that, a web scraper capable of scraping data from tree URLs and afterwards store it, manipulate it, and transform it into information, in this case, I am going to create some charts. As required by this assignment, I am going to store the data in an HDF5 file and manipulate it using "pandas" library and for the charts we are also using "pandas" and "matplotlib" libraries. For the scraping, I will be using the "requests" and "BeautifulSoup" libraries. This data will be collected once every day for two weeks so that I have some temporal information to create the charts with some materialistic meaning. The outputs will be discussed as well.

One thing to have in consideration when web scraping, is the Terms and Conditions of that web page. In some cases, any type of scraping is completely forbidden, so that is one thing I had to have in mind when selecting the pages to scrape. For this project I am going to scrape tree tables from tree URLs in the investing.com website where the Term and Conditions only specify that it is forbidden collect data and information about other users or third parties without their consent.

1. The Web Scraper

1.1 Targeted Data

As said in the introduction, for this scraper I targeted 3 tables from 3 URLs in the investing.com website, as required by the assignment. The data is stock market values from the PSI20, DAX and IBEX 35 indexes which corresponds to, Portugal, Germany, and Spain. Bellow you can see the tables and the type of data to be acquired:



	Name ⇕	Last	High	Low
🇵🇹	Altri	4.338	4.338	4.238
🇵🇹	Banco Comercial	0.2497	0.2518	0.2487
🇵🇹	Corticeira Amorim	9.83	9.88	9.73
🇵🇹	CTT Correios de Port...	3.34	3.35	3.29
🇵🇹	EDP	4.293	4.310	4.238
🇵🇹	EDP Renovaveis	17.64	17.66	17.42
🇵🇹	Galp Energia	12.35	12.40	12.22
🇵🇹	Greenvolt Energias R...	6.280	6.315	6.205
🇵🇹	Ibersol	6.78	6.80	6.72
🇵🇹	Jeronimo Martins	23.86	24.00	23.48
🇵🇹	Mota Engil	2.615	2.630	2.575
🇵🇹	Nos SGPS SA	3.33	3.33	3.28
🇵🇹	REN	2.490	2.490	2.465
🇵🇹	Semapa	13.24	13.24	13.16
🇵🇹	Sonae	0.9685	0.9685	0.9480
🇵🇹	The Navigator	3.300	3.300	3.254

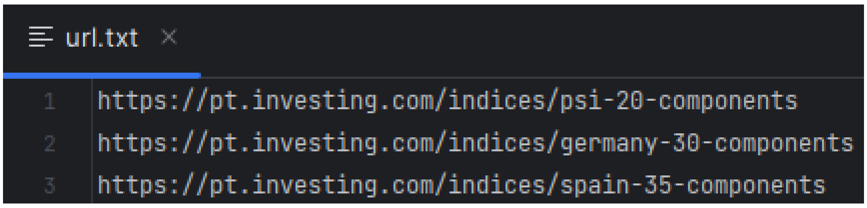
	Name ⇕	Last	High	Low
🇩🇪	Adidas	176.44	179.50	173.96
🇩🇪	Airbus Group	129.84	130.94	128.18
🇩🇪	Allianz	222.10	223.90	221.65
🇩🇪	BASF	45.025	45.420	44.940
🇩🇪	Bayer	50.05	50.20	49.74
🇩🇪	Beiersdorf AG	120.950	121.550	119.350
🇩🇪	BMW ST	96.510	97.190	96.110
🇩🇪	Brenntag AG	72.040	72.480	71.480
🇩🇪	Commerzbank	9.940	10.085	9.914
🇩🇪	Continental AG	67.40	67.96	67.04
🇩🇪	Covestro	48.270	49.720	48.050
🇩🇪	Daimler Truck Holding	32.27	32.63	32.11
🇩🇪	Deutsche Bank AG	9.896	10.008	9.878
🇩🇪	Deutsche Boerse	163.000	164.450	163.000
🇩🇪	Deutsche Post	42.185	42.490	41.825
🇩🇪	Deutsche Telekom AG	19.256	19.386	19.100
🇩🇪	Dr Ing hc F Porsche Prf	100.05	101.10	99.12
🇩🇪	E.ON SE	11.345	11.395	11.190
🇩🇪	Fresenius SE	29.340	29.510	29.180
🇩🇪	Hannover Rueckversi...	196.00	197.25	194.65
🇩🇪	Heidelbergcement	73.460	74.080	73.340
🇩🇪	Henkel VZO	71.92	72.44	71.90
🇩🇪	Infineon	31.975	32.380	31.500
🇩🇪	Mercedes Benz Group	67.000	67.320	66.610
🇩🇪	Merck	160.30	161.45	159.50
🇩🇪	MTU Aero	204.60	206.70	202.60
🇩🇪	Munich Re Group	355.50	358.20	354.00
🇩🇪	Porsche	48.670	48.890	48.510
🇩🇪	Qiagen	41.230	41.420	41.140
🇩🇪	Rheinmetall AG	243.300	245.800	242.600
🇩🇪	RWE AG ST	39.270	39.370	38.860
🇩🇪	SAP	126.160	127.240	125.660
🇩🇪	Sartorius AG VZO	346.60	351.50	346.60
🇩🇪	Siemens AG	136.40	138.00	135.14
🇩🇪	Siemens Energy AG	12.92	13.06	12.88
🇩🇪	Siemens Healthineers	45.43	45.68	45.25
🇩🇪	Symrise AG	93.300	93.840	91.720
🇩🇪	Volkswagen VZO	112.10	112.30	111.22
🇩🇪	Vonovia	20.40	20.63	20.25
🇩🇪	Zalando SE	26.45	26.82	26.34

	Name ⇕	Last	High	Low
🇪🇸	Acciona	126.300	126.700	124.750
🇪🇸	Acerinox	9.010	9.084	8.992
🇪🇸	ACS	31.670	32.040	31.550
🇪🇸	Aena	139.95	141.75	139.05
🇪🇸	Amadeus	61.120	62.240	60.820
🇪🇸	ArcelorMittal	24.050	24.310	23.995
🇪🇸	Banco de Sabadell	1.0900	1.1080	1.0885
🇪🇸	Bankinter	5.964	6.026	5.940
🇪🇸	BBVA	7.150	7.232	7.124
🇪🇸	Caixabank	3.790	3.828	3.777
🇪🇸	Cellnex Telecom	34.29	34.68	34.12
🇪🇸	Corporacion Acciona ...	26.02	26.16	25.62
🇪🇸	Enagas	15.535	15.585	15.385
🇪🇸	Endesa	19.025	19.090	18.775
🇪🇸	Ferrovial	28.900	29.060	28.690
🇪🇸	Fluidra	20.180	20.400	20.040
🇪🇸	Grifols	12.820	12.985	12.670
🇪🇸	IAG	1.845	1.878	1.837
🇪🇸	Iberdrola	10.965	11.015	10.810
🇪🇸	Inditex	34.210	34.520	33.900
🇪🇸	Indra A	12.930	13.010	12.820
🇪🇸	Inmobiliaria Colonial	5.305	5.360	5.295
🇪🇸	Laborat.Rovi	50.250	50.350	49.560
🇪🇸	Logista	24.72	24.90	24.50
🇪🇸	Mapfre	1.915	1.934	1.901
🇪🇸	Melia Hotels	5.930	6.040	5.910
🇪🇸	Merlin Properties SA	8.015	8.065	7.965
🇪🇸	Naturgy Energy	26.200	26.320	25.960
🇪🇸	Red Electrica	14.890	14.995	14.720
🇪🇸	Repsol	14.075	14.165	13.855
🇪🇸	Sacyr	2.888	2.904	2.860
🇪🇸	Santander	3.5380	3.5890	3.5300
🇪🇸	Solaria	13.390	13.450	13.125
🇪🇸	Telefonica	3.6870	3.7220	3.6780
🇪🇸	Unicaja Banco	1.017	1.027	1.013

Figure 1 - Targeted Data

1.2 The code to scrape

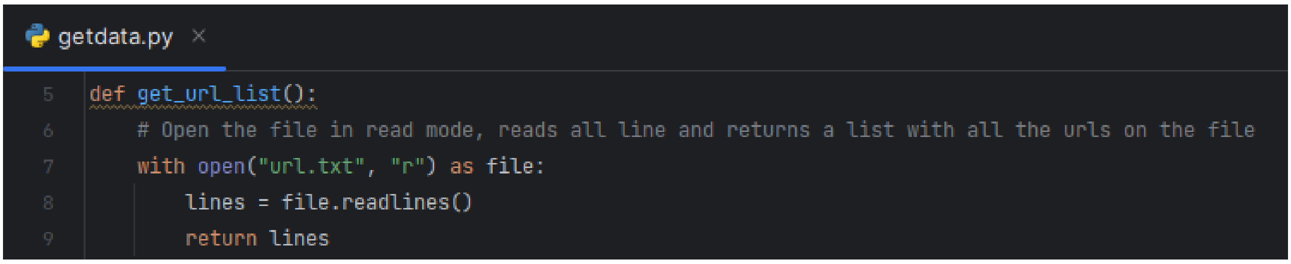
I created a text file with the tree links to scrape. This is useful because, in the future, if I want to change to any other index, I just have to change or add the URLs on this text file, see the figure bellow:



```
https://pt.investing.com/indices/psi-20-components
https://pt.investing.com/indices/germany-30-components
https://pt.investing.com/indices/spain-35-components
```

Figure 2 - URLs Text File

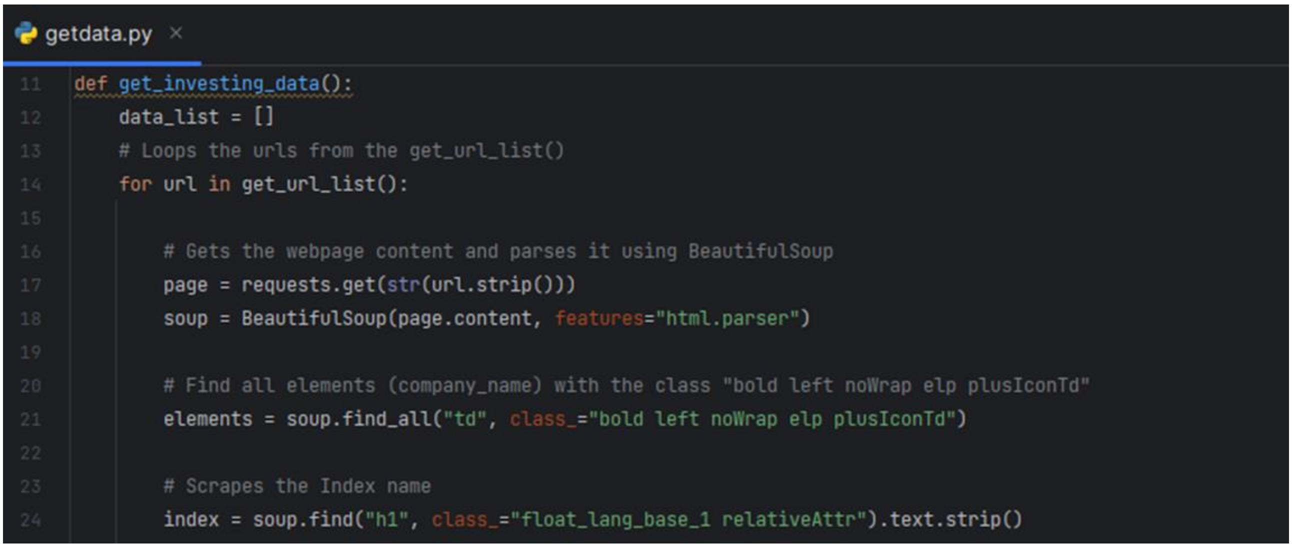
Basically, I created a function to read and create a list with the URLs contained on the text file, see bellow:



```python
def get_url_list():
    # Open the file in read mode, reads all line and returns a list with all the urls on the file
    with open("url.txt", "r") as file:
        lines = file.readlines()
        return lines
```

Figure 3 - get_url_list Function

After that, another function was created to loop through each individual URL and acquire the data. This is a big function which I will be deconstructing and explaining it. Additionally, "requests" and BeautifulSoup" libraries are required, see bellow:

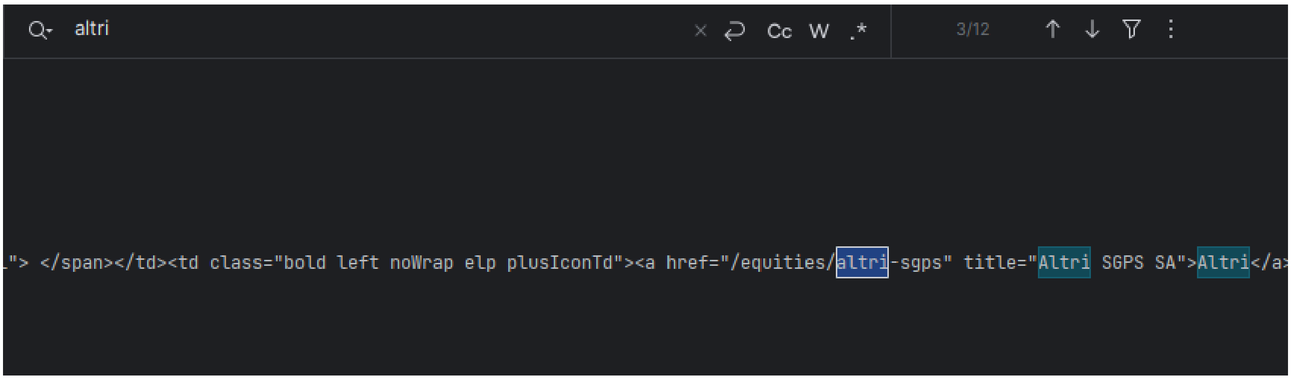


```python
11  def get_investing_data():
12      data_list = []
13      # Loops the urls from the get_url_list()
14      for url in get_url_list():
15
16          # Gets the webpage content and parses it using BeautifulSoup
17          page = requests.get(str(url.strip()))
18          soup = BeautifulSoup(page.content, features="html.parser")
19
20          # Find all elements (company_name) with the class "bold left noWrap elp plusIconTd"
21          elements = soup.find_all("td", class_="bold left noWrap elp plusIconTd")
22
23          # Scrapes the Index name
24          index = soup.find("h1", class_="float_lang_base_1 relativeAttr").text.strip()
```

Figure 4 - get_investing_data Function part 1/2

On line 17 and 18 of the Figure 4, a request is being sent to the specified URL and the server response contains the HTML content of that web page. This content is further processes and parsed by the beautifulSoup to make it possible to search, extract and manipulate specific elements and data. At this point, if we print "soup" we get the entire "soup" object which is essentially the parsed

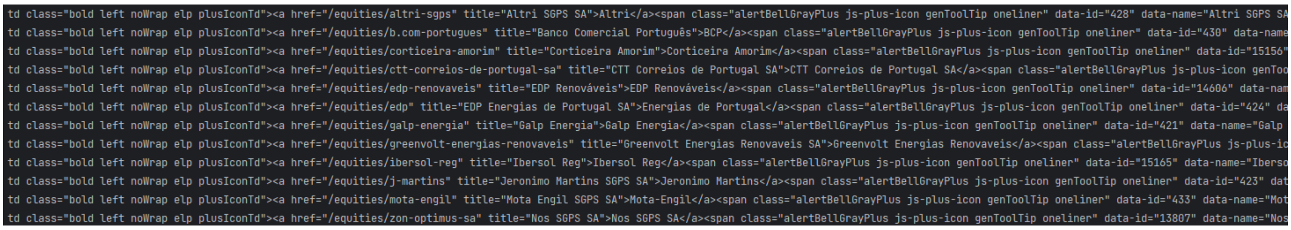
HTML content and structure of the web page. By going through this “soup” we can see where one of the companies of the targeted data is:



```
"> </span></td><td class="bold left noWrap elp plusIconTd"><a href="/equities/altri-sgps" title="Altri SGPS SA">Altri</a
```

Figure 5 - Discovering the span and the class of the desired data

So, we now know that the “span” of our table is “td” and the “class” is “bold left noWrap elp plusIconTd” and by running the code “elements = soup.find_all(“td”, class_= “bold left noWrap elp plusIconTd”)” on line 21 we get this data in memory:



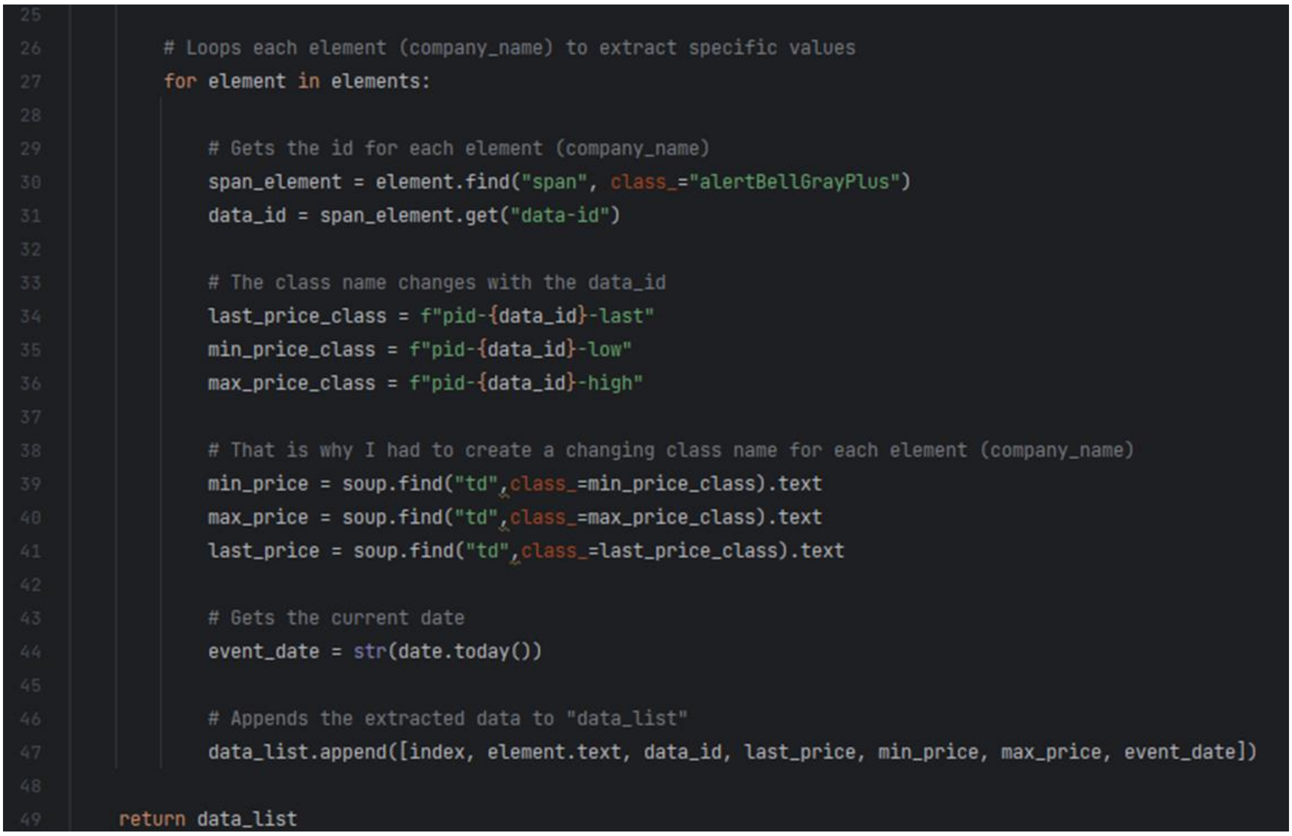
```
td class="bold left noWrap elp plusIconTd"><a href="/equities/altri-sgps" title="Altri SGPS SA">Altri</a><span class="alertBellGrayPlus js-plus-icon genToolTip oneliner" data-id="428" data-name="Altri SGPS SA
td class="bold left noWrap elp plusIconTd"><a href="/equities/b.com-portugues" title="Banco Comercial Português">BCP</a><span class="alertBellGrayPlus js-plus-icon genToolTip oneliner" data-id="430" data-name
td class="bold left noWrap elp plusIconTd"><a href="/equities/corticeira-amorim" title="Corticeira Amorim">Corticeira Amorim</a><span class="alertBellGrayPlus js-plus-icon genToolTip oneliner" data-id="15156"
td class="bold left noWrap elp plusIconTd"><a href="/equities/ctt-correios-de-portugal-sa" title="CTT Correios de Portugal SA">CTT Correios de Portugal SA</a><span class="alertBellGrayPlus js-plus-icon genToo
td class="bold left noWrap elp plusIconTd"><a href="/equities/edp-renovaveis" title="EDP Renováveis">EDP Renováveis</a><span class="alertBellGrayPlus js-plus-icon genToolTip oneliner" data-id="14606" data-nam
td class="bold left noWrap elp plusIconTd"><a href="/equities/edp" title="EDP Energias de Portugal SA">Energias de Portugal</a><span class="alertBellGrayPlus js-plus-icon genToolTip oneliner" data-id="424" da
td class="bold left noWrap elp plusIconTd"><a href="/equities/galp-energia" title="Galp Energia">Galp Energia</a><span class="alertBellGrayPlus js-plus-icon genToolTip oneliner" data-id="421" data-name="Galp 
td class="bold left noWrap elp plusIconTd"><a href="/equities/greenvolt-energias-renovaveis" title="Greenvolt Energias Renovaveis SA">Greenvolt Energias Renovaveis</a><span class="alertBellGrayPlus js-plus-ic
td class="bold left noWrap elp plusIconTd"><a href="/equities/ibersol-reg" title="Ibersol Reg">Ibersol Reg</a><span class="alertBellGrayPlus js-plus-icon genToolTip oneliner" data-id="15165" data-name="Iberso
td class="bold left noWrap elp plusIconTd"><a href="/equities/j-martins" title="Jeronimo Martins SGPS SA">Jeronimo Martins</a><span class="alertBellGrayPlus js-plus-icon genToolTip oneliner" data-id="423" dat
td class="bold left noWrap elp plusIconTd"><a href="/equities/mota-engil" title="Mota Engil SGPS SA">Mota-Engil</a><span class="alertBellGrayPlus js-plus-icon genToolTip oneliner" data-id="433" data-name="Mot
td class="bold left noWrap elp plusIconTd"><a href="/equities/zon-optimus-sa" title="Nos SGPS SA">Nos SGPS SA</a><span class="alertBellGrayPlus js-plus-icon genToolTip oneliner" data-id="13807" data-name="Nos
```

Figure 6 - All the elements on the table

At this point we are getting the names of all the companies and the “data-id” (which will be required shortly), but we are not getting the “Last” values for example. That is because these values are under another “class”.

For storing purpose we are also scraping the index name in the code “index = soup.find(“h1”, class_=”float_lang_base_1 relativeAttr”).text.strip() which is outside of the table and has a “span” of “h1” and a class of “float_lang_base_1 relativeAttr”.

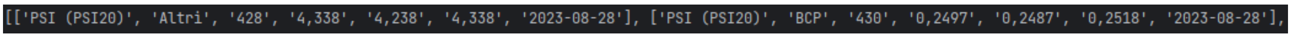
Continuing our big function, we are going to loop through each element (each company) on our elements (all companies) variable. See bellow:



```
        # Loops each element (company_name) to extract specific values
        for element in elements:

            # Gets the id for each element (company_name)
            span_element = element.find("span", class_="alertBellGrayPlus")
            data_id = span_element.get("data-id")

            # The class name changes with the data_id
            last_price_class = f"pid-{data_id}-last"
            min_price_class = f"pid-{data_id}-low"
            max_price_class = f"pid-{data_id}-high"

            # That is why I had to create a changing class name for each element (company_name)
            min_price = soup.find("td",class_=min_price_class).text
            max_price = soup.find("td",class_=max_price_class).text
            last_price = soup.find("td",class_=last_price_class).text

            # Gets the current date
            event_date = str(date.today())

            # Appends the extracted data to "data_list"
            data_list.append([index, element.text, data_id, last_price, min_price, max_price, event_date])

    return data_list
```

Figure 7 - get_investing_data Function part 2/2

This is the moment where we are going to use the “data-id” because the name of the class of “Last”, “High” and “Low” values changes accordingly. We start by the getting the “data-id” of the company we are looping at that moment. We then create 3 variables, “last_price_class”, “min_price_class” and “max_price_class” that stores in memory a formatted string with the class name. Afterwards, we find the specific values using the “find” function again where the “span” is “td” and the “class” is our predefined variables. We are also getting the date of the scraping using the “datetime” library. To finish, we are appending all this data to a list containing the elements index name, company name, company id, last price, minimum price and max price for that day and date. We got something like this in memory (list of lists):

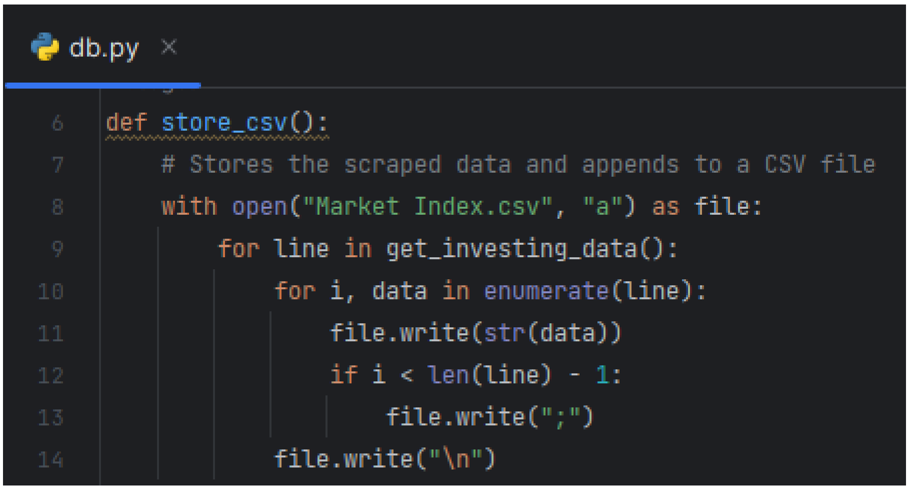


```
[['PSI (PSI20)', 'Altri', '428', '4,338', '4,238', '4,338', '2023-08-28'], ['PSI (PSI20)', 'BCP', '430', '0,2497', '0,2487', '0,2518', '2023-08-28'],
```

Figure 8 - Returned data from get_investing_data

1.3 The code to store

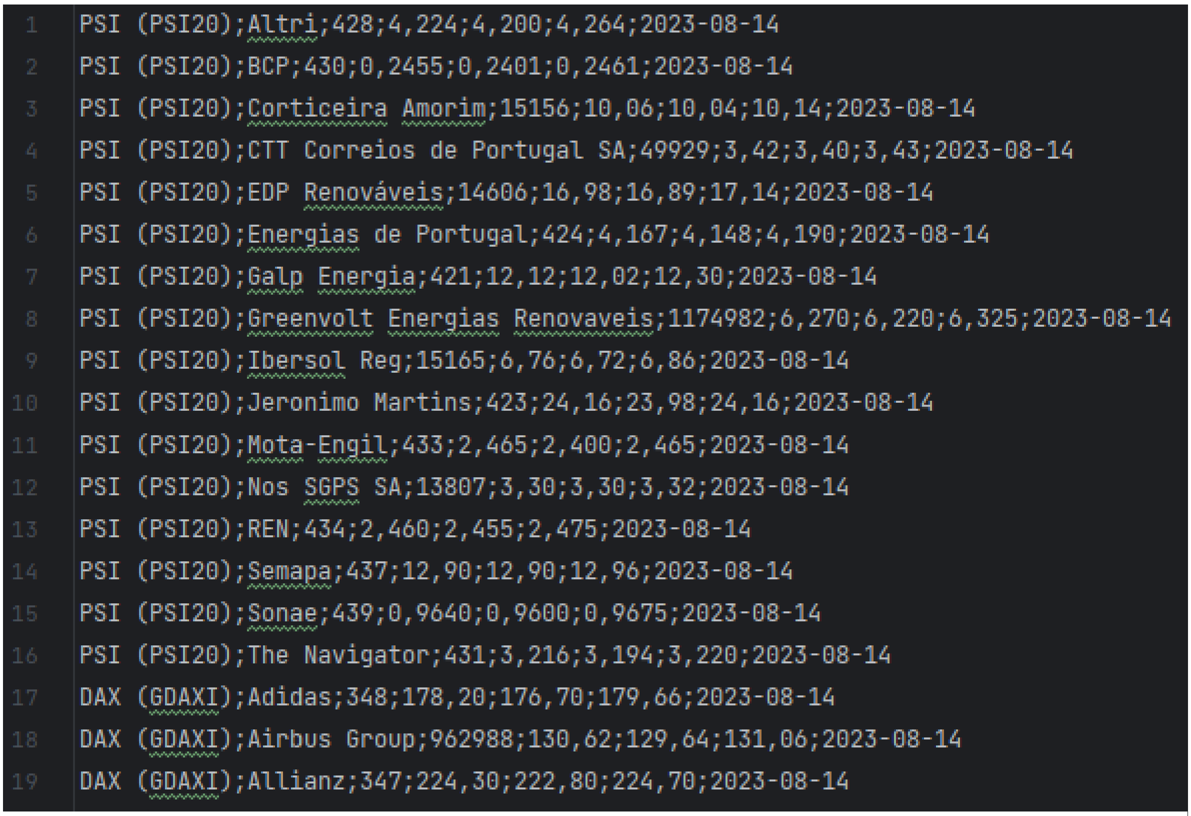
For this project, our data storage approach involved using an HDF5 file. However, I chose to initially append this data to a CSV file before subsequently writing it to the HDF5 file. That way, I could see better if there was anything wrong on the csv file beforehand. To append the scraped data to a csv file I used the following function:



```python
def store_csv():
    # Stores the scraped data and appends to a CSV file
    with open("Market Index.csv", "a") as file:
        for line in get_investing_data():
            for i, data in enumerate(line):
                file.write(str(data))
                if i < len(line) - 1:
                    file.write(";")
            file.write("\n")
```

Figure 9 - store_csv Function

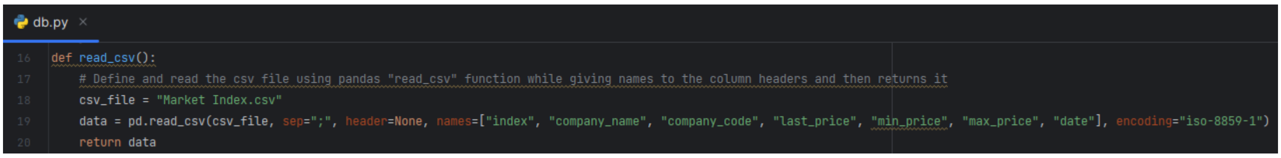
We start by opening a csv file named “Market Index” in append mode (if there is no file with that name, one will be created and opened). The function uses a loop to iterate over the data obtained from the function “get_investing_data”. Within this loop, the code iterates over each element within each “line” (each line consists of index name, company name, company id, last price, minimum price and max price for that day and date). We convert the data to a string to write to the file and after each element the code writes “;” as separator between elements. When the code is finished with the “data” of one “line”, it writes “\n” to the file, inserting a new line to start over the loop on a new “line”. The csv file looks like this:



```
PSI (PSI20);Altri;428;4,224;4,200;4,264;2023-08-14
PSI (PSI20);BCP;430;0,2455;0,2401;0,2461;2023-08-14
PSI (PSI20);Corticeira Amorim;15156;10,06;10,04;10,14;2023-08-14
PSI (PSI20);CTT Correios de Portugal SA;49929;3,42;3,40;3,43;2023-08-14
PSI (PSI20);EDP Renováveis;14606;16,98;16,89;17,14;2023-08-14
PSI (PSI20);Energias de Portugal;424;4,167;4,148;4,190;2023-08-14
PSI (PSI20);Galp Energia;421;12,12;12,02;12,30;2023-08-14
PSI (PSI20);Greenvolt Energias Renovaveis;1174982;6,270;6,220;6,325;2023-08-14
PSI (PSI20);Ibersol Reg;15165;6,76;6,72;6,86;2023-08-14
PSI (PSI20);Jeronimo Martins;423;24,16;23,98;24,16;2023-08-14
PSI (PSI20);Mota-Engil;433;2,465;2,400;2,465;2023-08-14
PSI (PSI20);Nos SGPS SA;13807;3,30;3,30;3,32;2023-08-14
PSI (PSI20);REN;434;2,460;2,455;2,475;2023-08-14
PSI (PSI20);Semapa;437;12,90;12,90;12,96;2023-08-14
PSI (PSI20);Sonae;439;0,9640;0,9600;0,9675;2023-08-14
PSI (PSI20);The Navigator;431;3,216;3,194;3,220;2023-08-14
DAX (GDAXI);Adidas;348;178,20;176,70;179,66;2023-08-14
DAX (GDAXI);Airbus Group;962988;130,62;129,64;131,06;2023-08-14
DAX (GDAXI);Allianz;347;224,30;222,80;224,70;2023-08-14
```

Figure 10 - CSV File

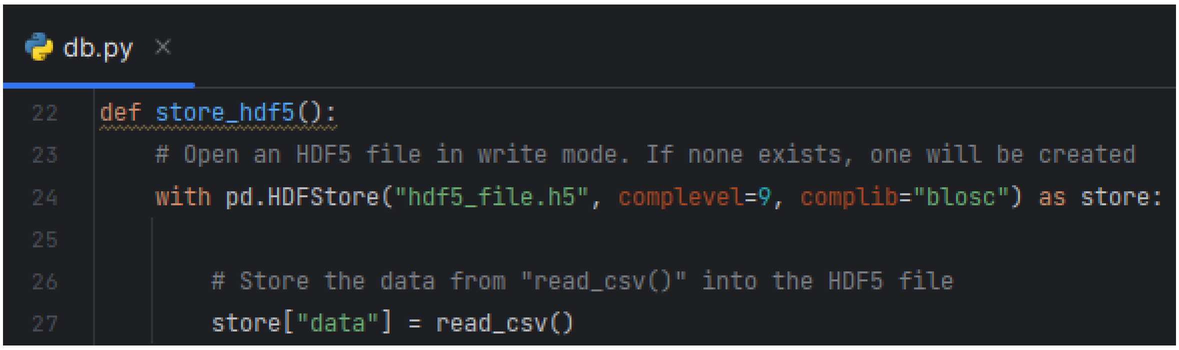
To now store the csv file into an HDF5 file, I had to create a function to read the csv file. Note that I am using the pandas library to read the file, see bellow:



```python
def read_csv():
    # Define and read the csv file using pandas "read_csv" function while giving names to the column headers and then returns it
    csv_file = "Market Index.csv"
    data = pd.read_csv(csv_file, sep=";", header=None, names=["index", "company_name", "company_code", "last_price", "min_price", "max_price", "date"], encoding="iso-8859-1")
    return data
```

Figure 11 - read_csv Function

The function reads data from the "Market Index" csv file using pandas, assigning column headers, and returning a dataframe containing the read data. The encoding used for reading the file is "iso-8859-1". We are now ready to store the data into an HDF5 file, see the code bellow:



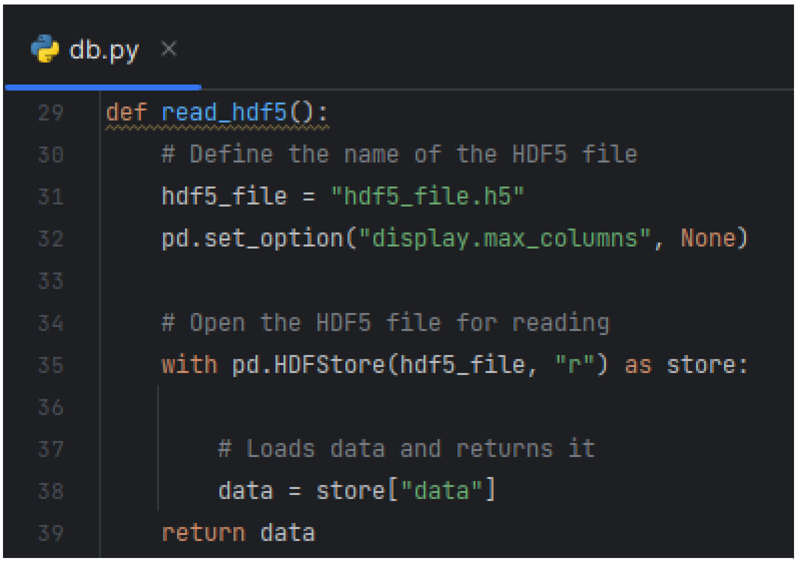
```python
def store_hdf5():
    # Open an HDF5 file in write mode. If none exists, one will be created
    with pd.HDFStore("hdf5_file.h5", complevel=9, complib="blosc") as store:

        # Store the data from "read_csv()" into the HDF5 file
        store["data"] = read_csv()
```

Figure 12 - store_hdf5 Function

The function opens or creates an HDF5 file named "hdf5_file" in write mode using pandas. The data being stored is obtained from the above function ("read_csv"). It also defines a compression level and a compression library. The higher the compression level, the smaller the size of the files, but that also results in a slower process because it requires more computational resources to compress and decompress the data. Even though this was not required, I wanted to see the difference between compression levels. The file with a compression level of 0 had 1082 KB and the file with a compression level of 9 (max) had 1077 KB so, the size difference between files is almost inexistent, but our data is relatively small. As for the speed of compressing data, the difference was unnoticeable. We also had to specify the compression library, in this case "blosc". In the references will be a link to this specific pandas documentation (pandas library, 2019).

1.4 The code to read, manipulate and show charts

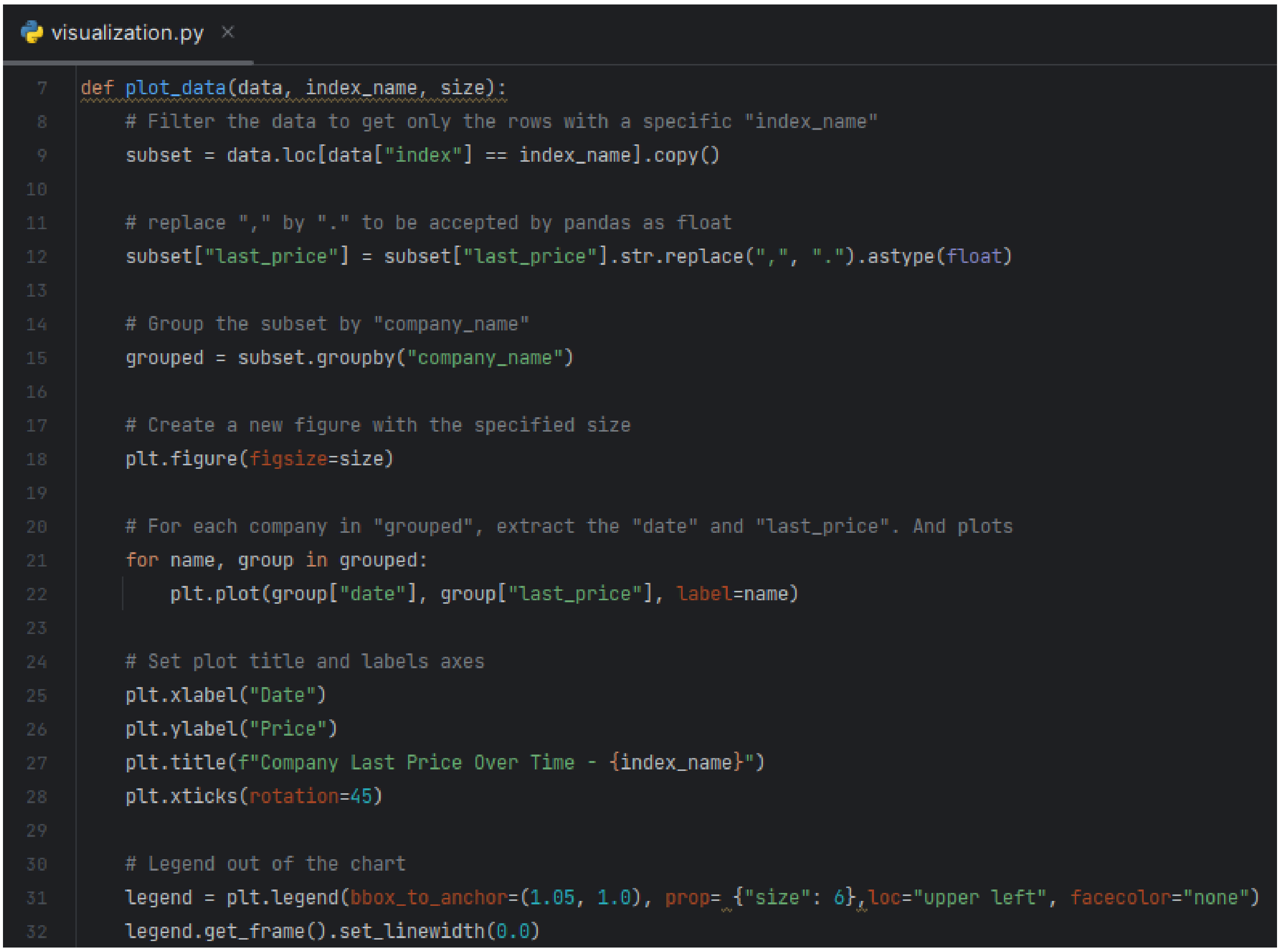
On this section I am going to create some line charts, one for each index, meaning, one for each country. To start up, we need a function to read the HDF5 file like the one bellow:



```python
def read_hdf5():
    # Define the name of the HDF5 file
    hdf5_file = "hdf5_file.h5"
    pd.set_option("display.max_columns", None)

    # Open the HDF5 file for reading
    with pd.HDFStore(hdf5_file, "r") as store:

        # Loads data and returns it
        data = store["data"]
    return data
```

Figure 13 - read_hdf5 Function

With this function, we are basically opening the file in read mode and loading all the data into memory. Now that we have the data, we need a function to plot it into a chart. To do that, I am going to use the “pandas” and “matplotlib” libraries, see bellow:



```python
def plot_data(data, index_name, size):
    # Filter the data to get only the rows with a specific "index_name"
    subset = data.loc[data["index"] == index_name].copy()

    # replace "," by "." to be accepted by pandas as float
    subset["last_price"] = subset["last_price"].str.replace(",", ".").astype(float)

    # Group the subset by "company_name"
    grouped = subset.groupby("company_name")

    # Create a new figure with the specified size
    plt.figure(figsize=size)

    # For each company in "grouped", extract the "date" and "last_price". And plots
    for name, group in grouped:
        plt.plot(group["date"], group["last_price"], label=name)

    # Set plot title and labels axes
    plt.xlabel("Date")
    plt.ylabel("Price")
    plt.title(f"Company Last Price Over Time - {index_name}")
    plt.xticks(rotation=45)

    # Legend out of the chart
    legend = plt.legend(bbox_to_anchor=(1.05, 1.0), prop= {"size": 6},loc="upper left", facecolor="none")
    legend.get_frame().set_linewidth(0.0)
```

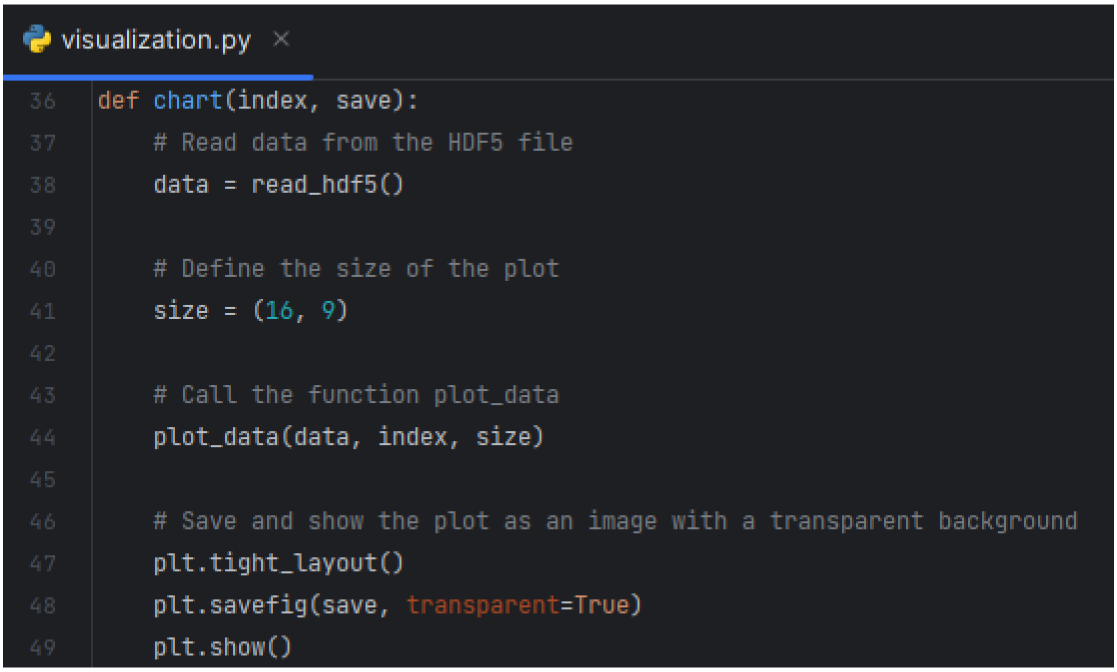
Figure 14 - plot_data Function

Note that the function has 3 arguments, data (the data provided by the read_hdf5 function), index_name (the name of the index accordingly with the name in our data, “PSI (PSI20)”, “DAX (GDAXI)” and “IBEX 35 (IBEX)”) and size (the size of our plot). These arguments are going to be passed when we run another function called “chart” which I will be explaining later, for now, let’s focus on this one.

We start by creating a subset which will have the data for a specific “index_name”. After that, I had to create a function to replace the “,” to “.” on our data values and convert it to a float. Converting it to a float without changing the “,” to “.” would show strange numbers. Afterwards, a “grouped” variable is created with the subset being grouped by the “company_name” (header column name).

We then create and define the size of our figure and we start plotting using the “matplotlib” library. On the “x” axis will be dates and on “y” axis will be the value in Euros and the function plots one dot for each combination of “company_name”, “date” and “last_price”. Because we want a line chart, the dots will be connected. Next, we are defining the plot tittle, we label axis and we create a legend for each line (outside of the actual plot).

I will now explain and show the function “chart” mentioned above. This function serves to pass the 3 arguments to the “plot_data” function and is responsible for the outcome of our plot, see bellow:



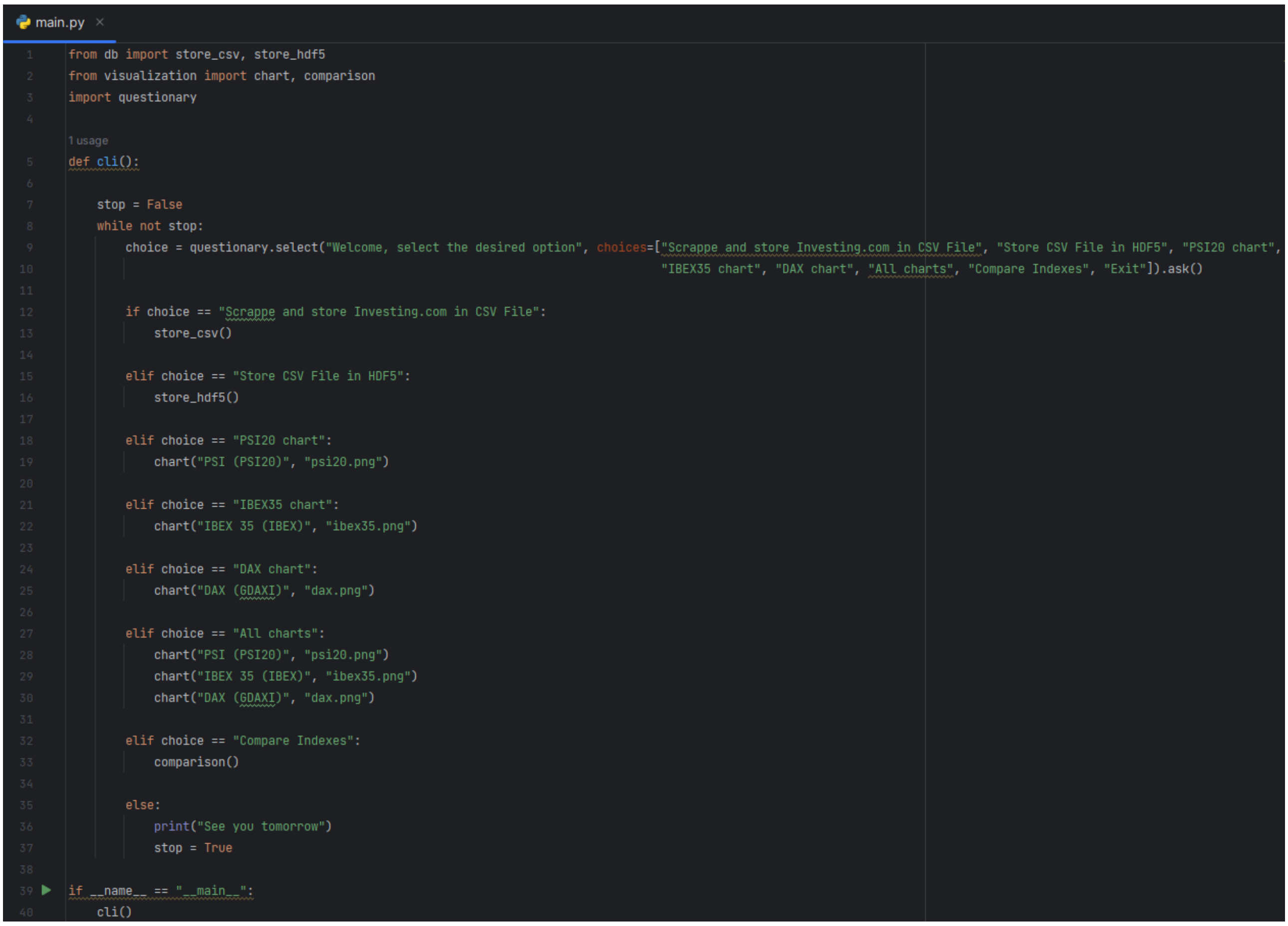
```python
def chart(index, save):
    # Read data from the HDF5 file
    data = read_hdf5()

    # Define the size of the plot
    size = (16, 9)

    # Call the function plot_data
    plot_data(data, index, size)

    # Save and show the plot as an image with a transparent background
    plt.tight_layout()
    plt.savefig(save, transparent=True)
    plt.show()
```

Figure 15 - chart Function

This function takes 2 arguments which comes from our main.py, those being index (index name) and save (save name of our plot figure). We start by loading the data from the “read_hdf5” function and “storing” it in our data variable, we define the size of our plot and then we call the function “plot_data” with those 3 required arguments. After that, we define to save the figure with a transparent background on the same path as our project and to finish, we show the chart on screen. For now I will not be showing charts because those will be discussed at the chapter 2.

1.5 The code of main with CLI

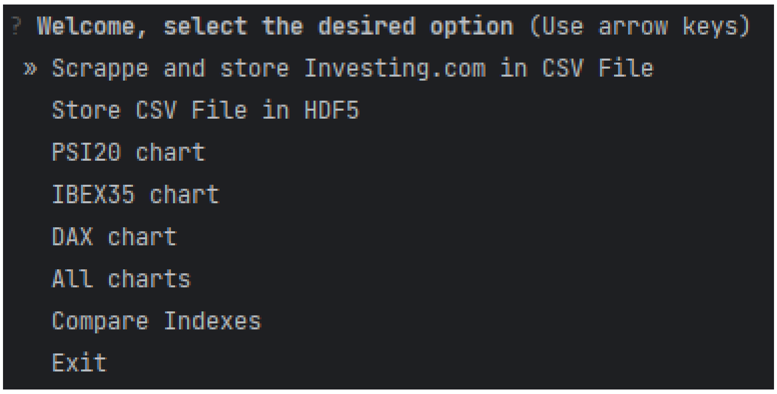
In main.py I created a CLI using the “questionary” library with the following design:



```python
from db import store_csv, store_hdf5
from visualization import chart, comparison
import questionary

def cli():

    stop = False
    while not stop:
        choice = questionary.select("Welcome, select the desired option", choices=["Scrappe and store Investing.com in CSV File", "Store CSV File in HDF5", "PSI20 chart",
                                                                                   "IBEX35 chart", "DAX chart", "All charts", "Compare Indexes", "Exit"]).ask()

        if choice == "Scrappe and store Investing.com in CSV File":
            store_csv()

        elif choice == "Store CSV File in HDF5":
            store_hdf5()

        elif choice == "PSI20 chart":
            chart("PSI (PSI20)", "psi20.png")

        elif choice == "IBEX35 chart":
            chart("IBEX 35 (IBEX)", "ibex35.png")

        elif choice == "DAX chart":
            chart("DAX (GDAXI)", "dax.png")

        elif choice == "All charts":
            chart("PSI (PSI20)", "psi20.png")
            chart("IBEX 35 (IBEX)", "ibex35.png")
            chart("DAX (GDAXI)", "dax.png")

        elif choice == "Compare Indexes":
            comparison()

        else:
            print("See you tomorrow")
            stop = True

if __name__ == "__main__":
    cli()
```

Figure 16 - main.py code

Basically a menu will be presented when running the project with the options “"Scrappe and store Investing.com in CSV File", "Store CSV File in HDF5", "PSI20 chart", "IBEX35 chart", "DAX chart", "All charts", "Compare Indexes", "Exit". Each option calls different functions to be executed accordingly to what I have been explaining in the previous chapters. This is what it looks like when run:



```
? Welcome, select the desired option (Use arrow keys)
 » Scrappe and store Investing.com in CSV File
   Store CSV File in HDF5
   PSI20 chart
   IBEX35 chart
   DAX chart
   All charts
   Compare Indexes
   Exit
```

Figure 17 - CLI Menu

The user can then use the arrow keys to navigate through the menu.

2. Outputs Discussion

These charts contain PSI20 (Portugal), DEX (Germany) and IBEX35 (Spain) company values over a 2 week period. These markets open early in the day and close at the end of the afternoon, that mean the values stay the same until the start of the next day. I conducted data scraping during the closed period, which is why the term “last_value” is meaningful as it represents the value just prior to closure. The corresponding charts will be included in the appendices section located at the end of this report.

2.1 PSI20 Chart

Starting with Portugal which the chart with less companies:

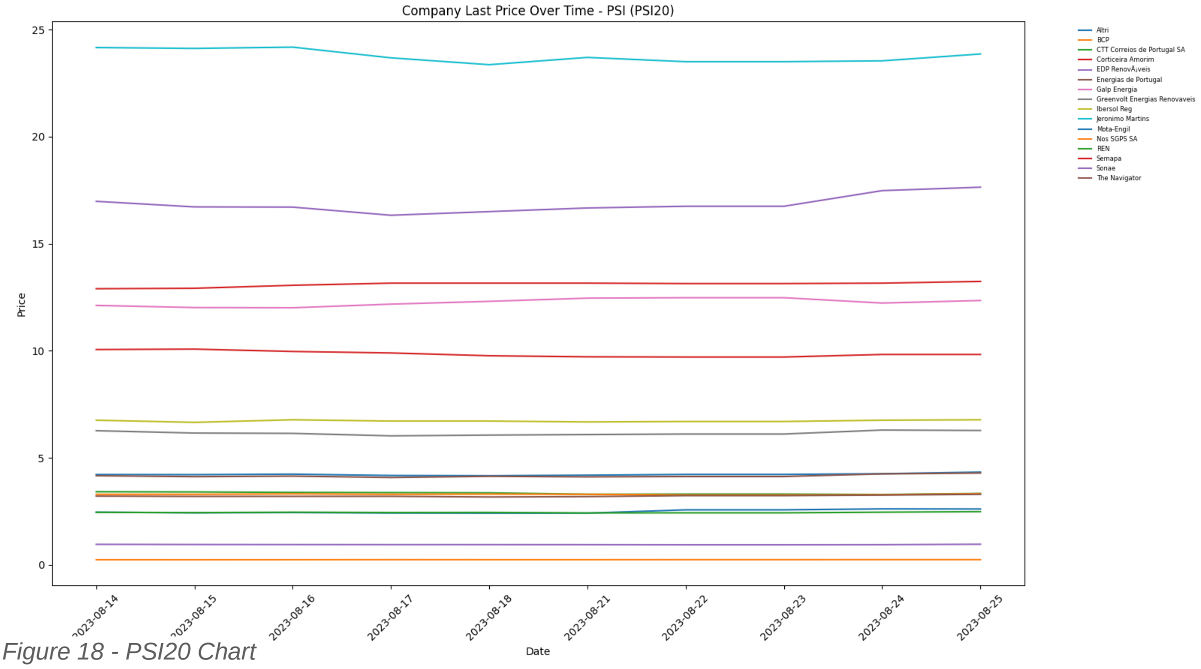


Figure 18 - PSI20 Chart

Even though for this assignment we were only required to scrape data for about 7 days, I decided to extend this period because, as we can see, there are no big changes on the lines, therefore there is not many things to say about. One thing I can say is that the top companies like Jeronimo Martins and EDP Renovaveis, have the biggest flat fluctuations. For companies like Banco Comercial Portugues where its value is around 0,25 €, the changes are not visible because the value is so low and the fluctuations are so small.

When I was in the phase of deciding what to scrape, I envisioned a different kind of chart. I used to work in a company (funny enough, one company in the PSI20) and I was responsible to create and innovate the financial and operational report for the company administration. Sometimes our vision (without seeing the end result) doesn't align with the end product. This is exactly what happened here, in my mind, I was going to obtain a chart where some companies overlap the others, instead I have a chart where it looks like they ended where they started. Now it makes sense, for this kind of data to be more meaningful, the period had to be longer.

2.2 IBEX35 Chart

Followed by Spain, IBEX35:

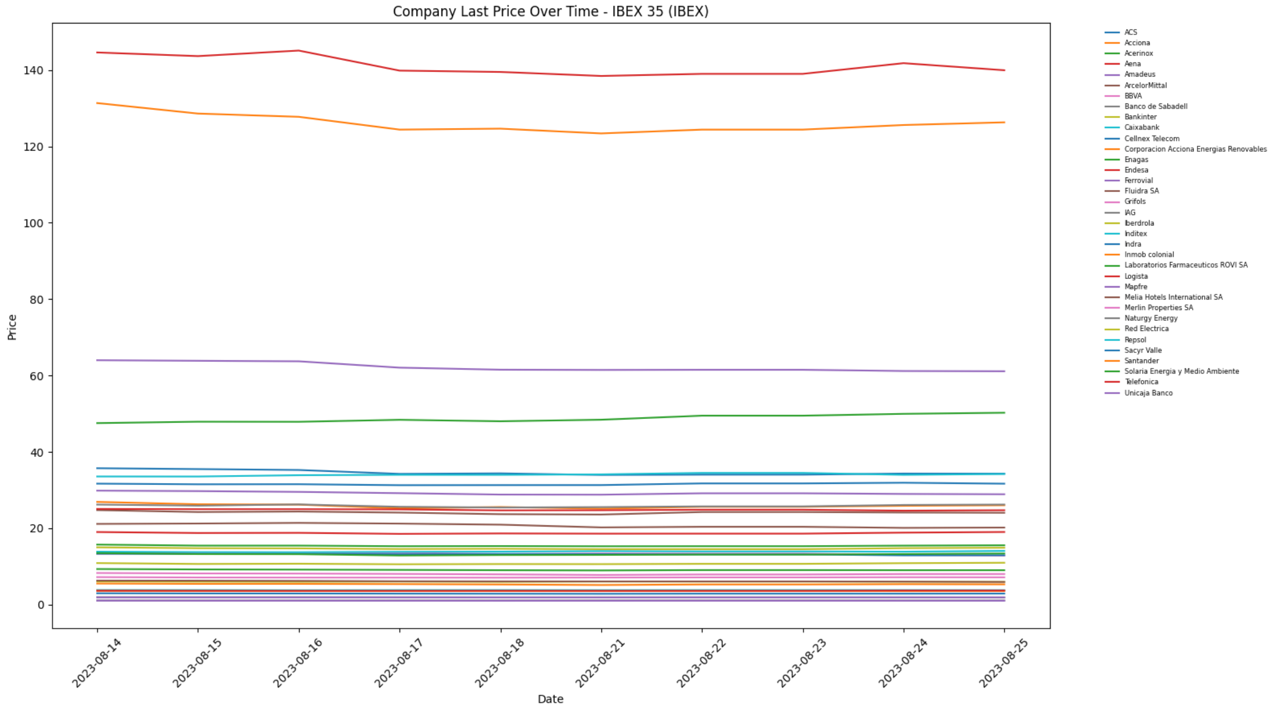


Figure 19 - IBEX35 Chart

IBEX35 with its 35 companies in a single chart was, definitely, not a good idea. We can see that this index has 2 major companies that are really far from the others, those being, Acciona (red color) and Aena (orange color). The rest of the chart, we can say that it is a beautiful rainbow. If we remove the 2 top companies the others would be more visible, so let's do it.

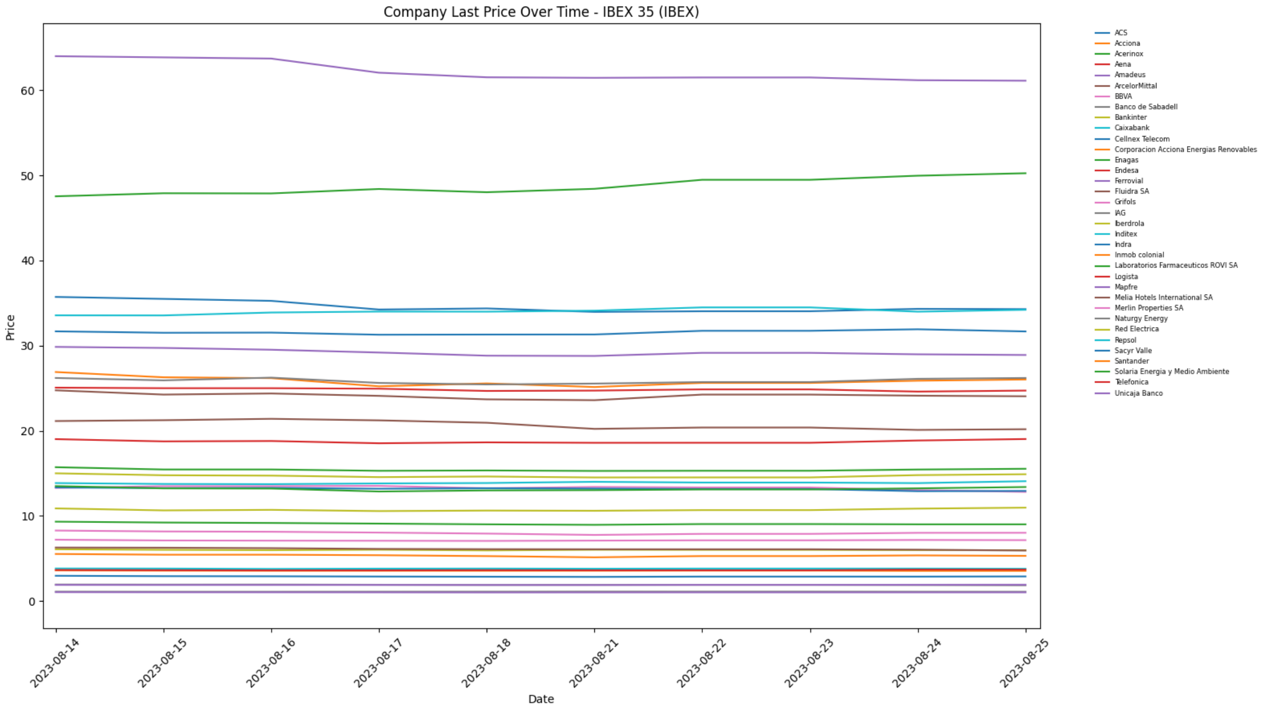


Figure 20 - IBEX35 without 2 top companies

As we can see, we now have another 2 top companies, those being Amadeus and Laboratorios Farmaceuticos ROVI. After that, we are left with 31 companies with its values in between 40 € and 0 €. It is not visible, but there are 2 lines inside of each other on the lowest position, those being, Unicaja Banco and Banco de Sabadell with its values around 1 €.

2.3 DAX Chart

Followed by Germany, DAX with its 40 companies. At this point, if Spain was not an easy chart to evaluate because it had too much information with 35 companies, we can only expect for this one to be even harder:

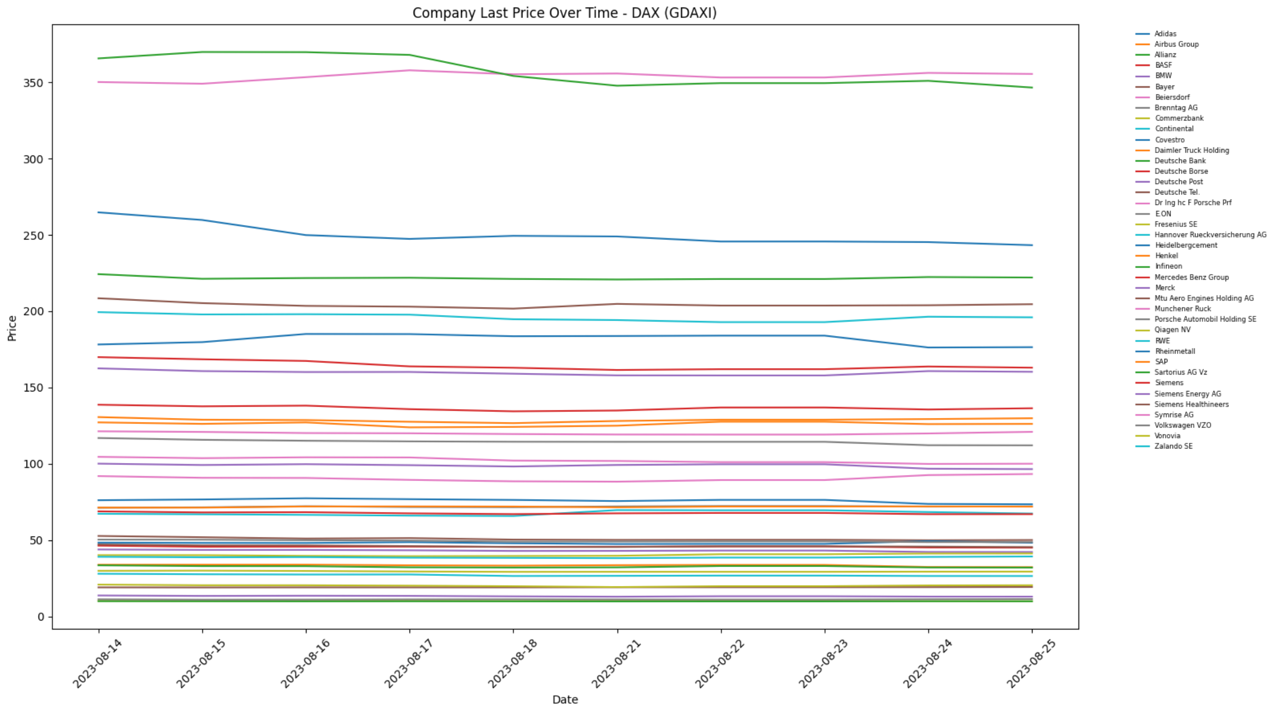


Figure 21 - DAX Chart

One thing we can immediately see is that once again there are 2 major companies, Sartorius AG Vz and Munchener Ruck. We can also say that these 2 companies have more than double value of the 2 top companies from Spain. In fact, there are 9 German companies above the highest Spanish company, and 34 German companies above the highest Portuguese company. Because I was not very happy with the end result of my charts, I decided to create another one with a comparison between countries.

2.4 Comparison Chart

The creation of this chart followed the same logic as the other ones, except that this one has the average of all the companies in each index. The code for the chart is going to be presented in the appendices.

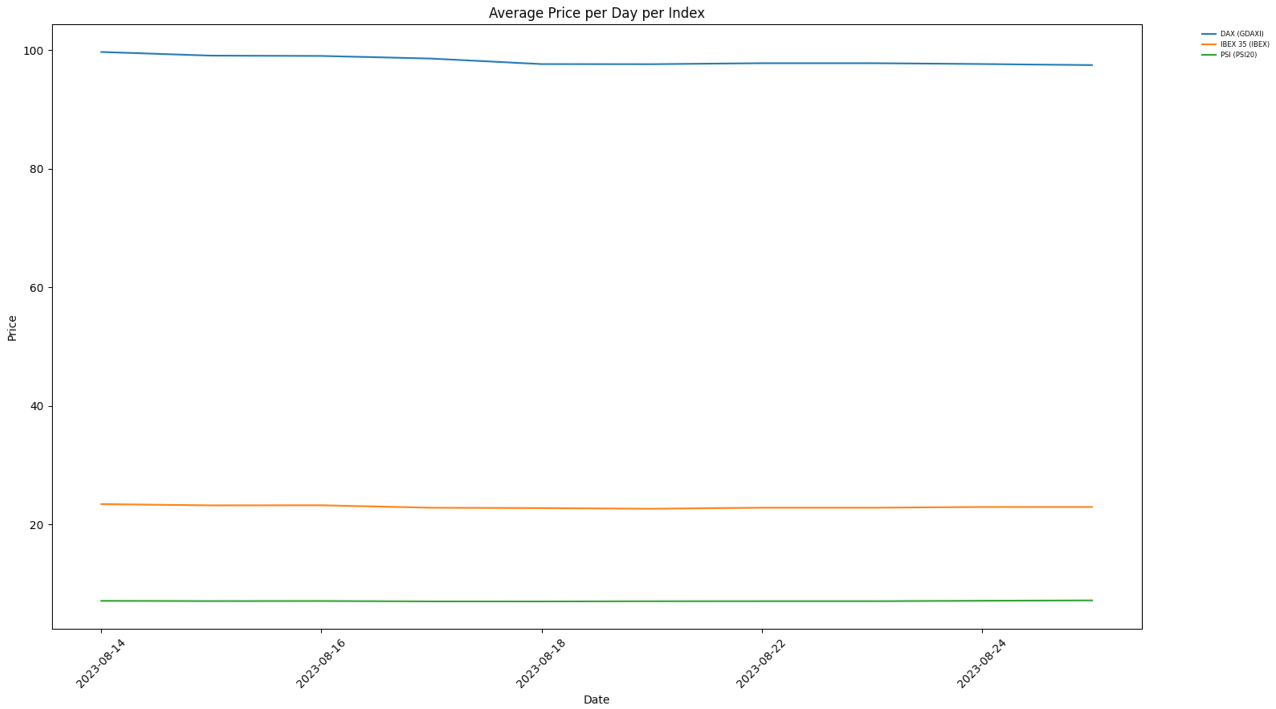


Figure 22 - Index comparison

Portugal boasts an average of 7,18 €, while Spain demonstrates an average of 22,93 €, and Germany exhibits a substantially higher average of 97,47 €. This indicates a remarkable contrast, with Germany's average value being 13,58 times greater than that of Portugal, and Spain's average standing 4,25 times lower than Germany's. These disparities are particularly striking considering that these countries are all in the European Union.

Conclusion

As a programming learner, it was very challenging to create this project. I had experience in manipulating data to create charts in excel, but never in a programming language. The good thing about this python project, is that everything is done in the same place, the web scraping, and the charts themselves.

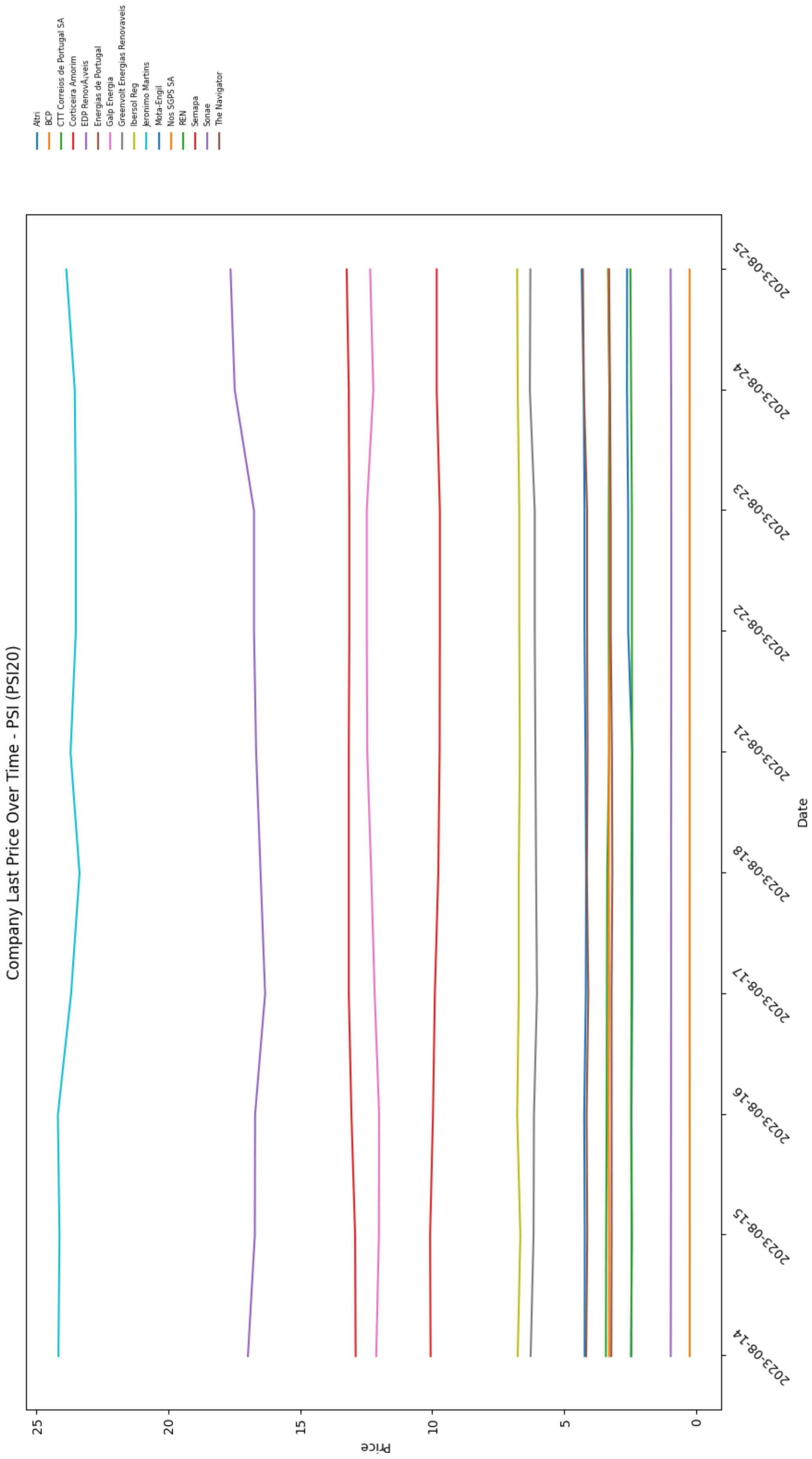
As I said in the main body of this report, I was not happy with the end result of the charts created. I am happy that I learned to do it successfully, but I was not happy with the information value provided by the charts. When I imagined the graphics, in my mind, they would be better for analysis however, sometimes this happens, situations where the creator visualizes something that in fact, when created, do not correspond to the creator's vision.

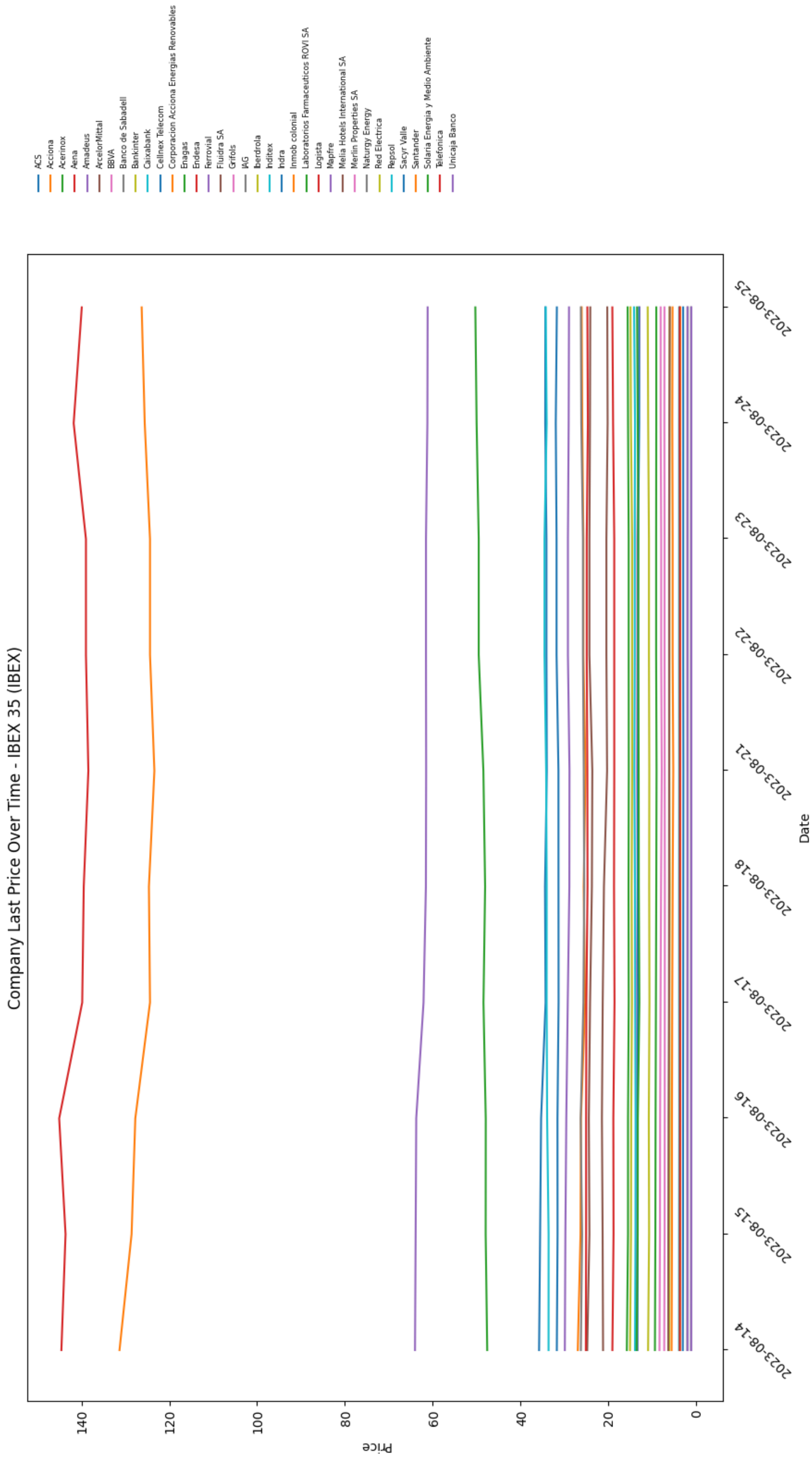
In conclusion, I would like to emphasize that I not only expanded my proficiency in web scraper development but also gained valuable insights into utilizing libraries such as pandas and matplotlib to enhance my skills.

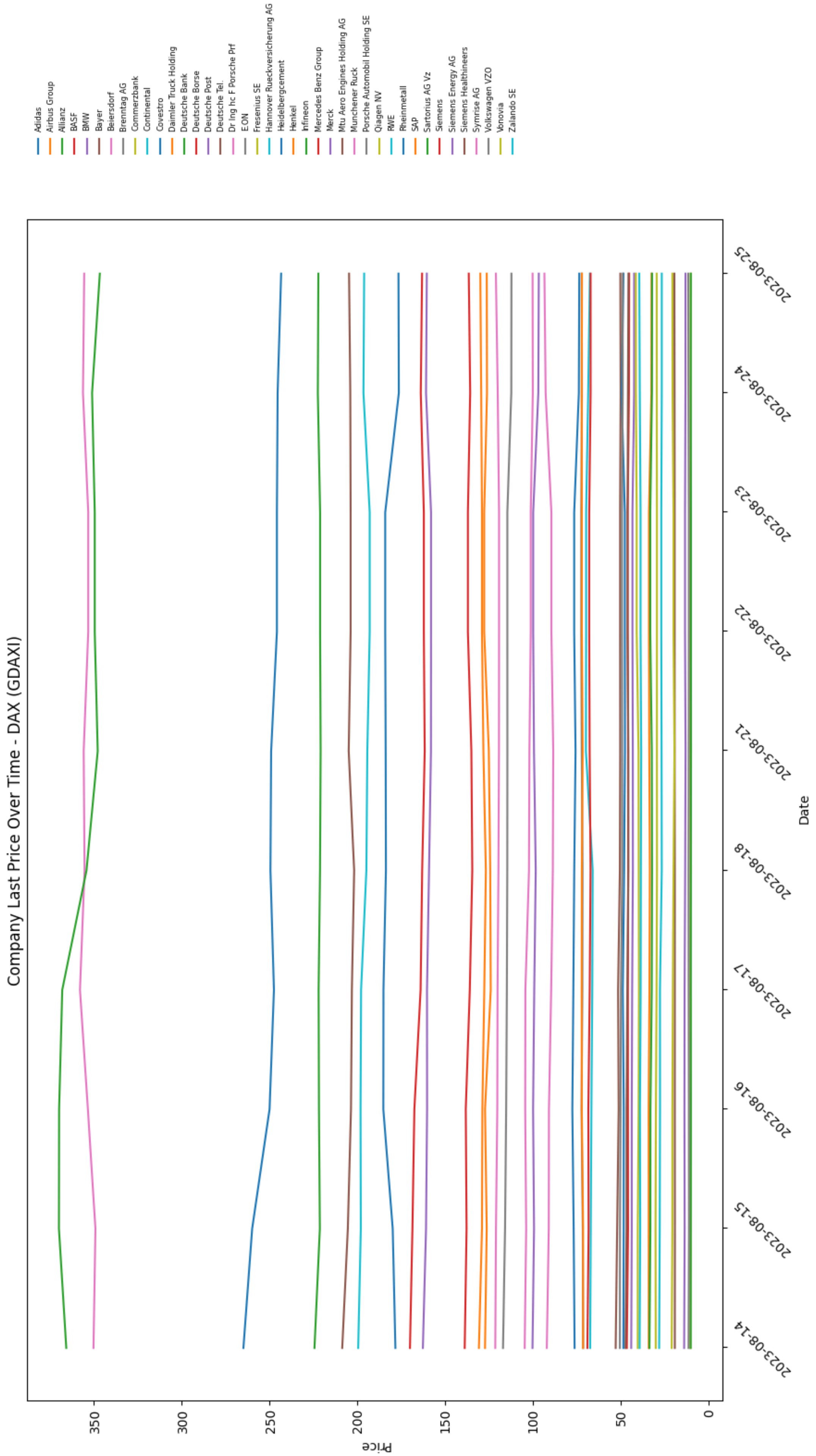
Bibliography

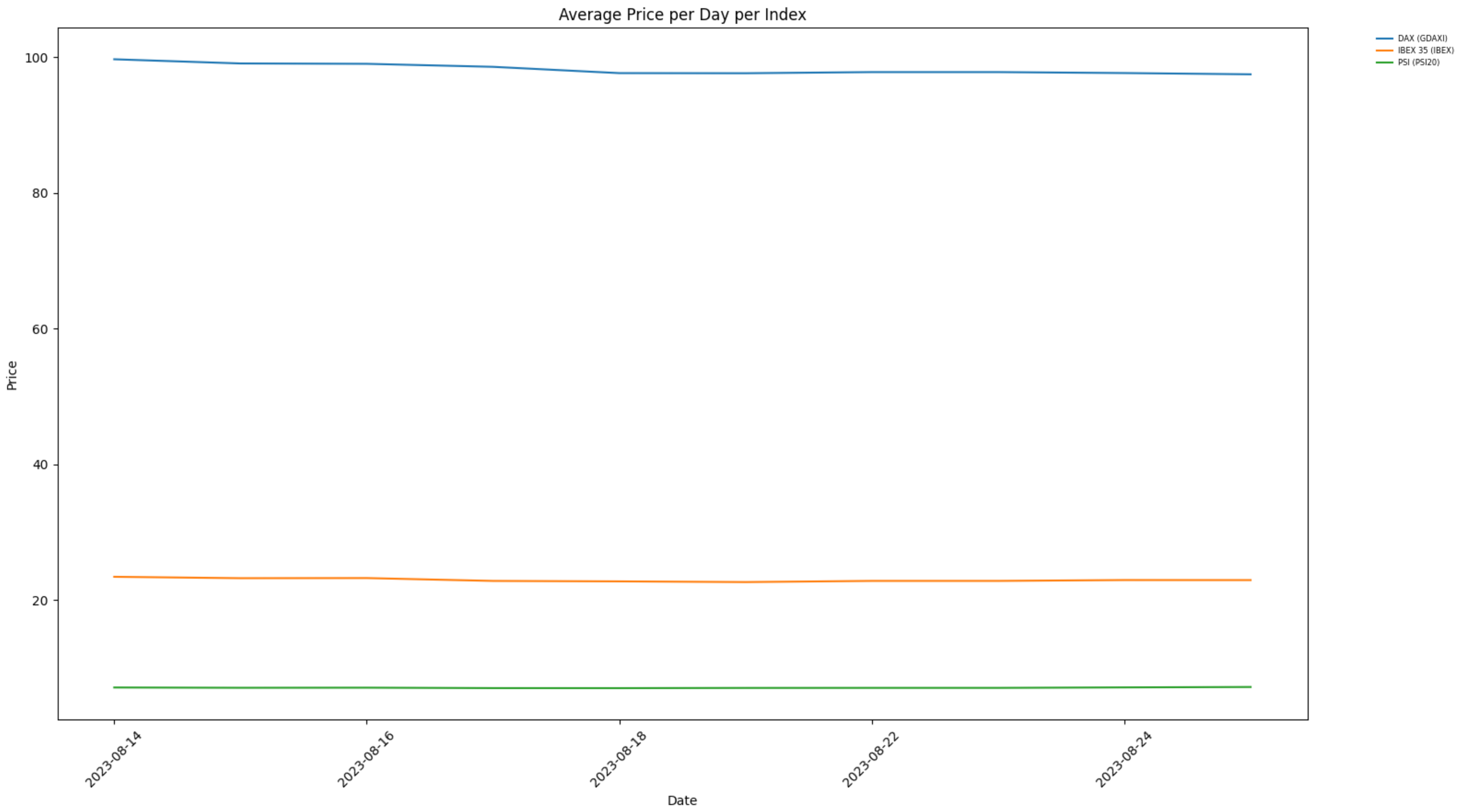
pandas library. (2019, 07 18). Retrieved from pandas 0.25.0 documentation: https://pandas.pydata.org/pandas-docs/version/0.25.0/reference/api/pandas.DataFrame.to_hdf.html

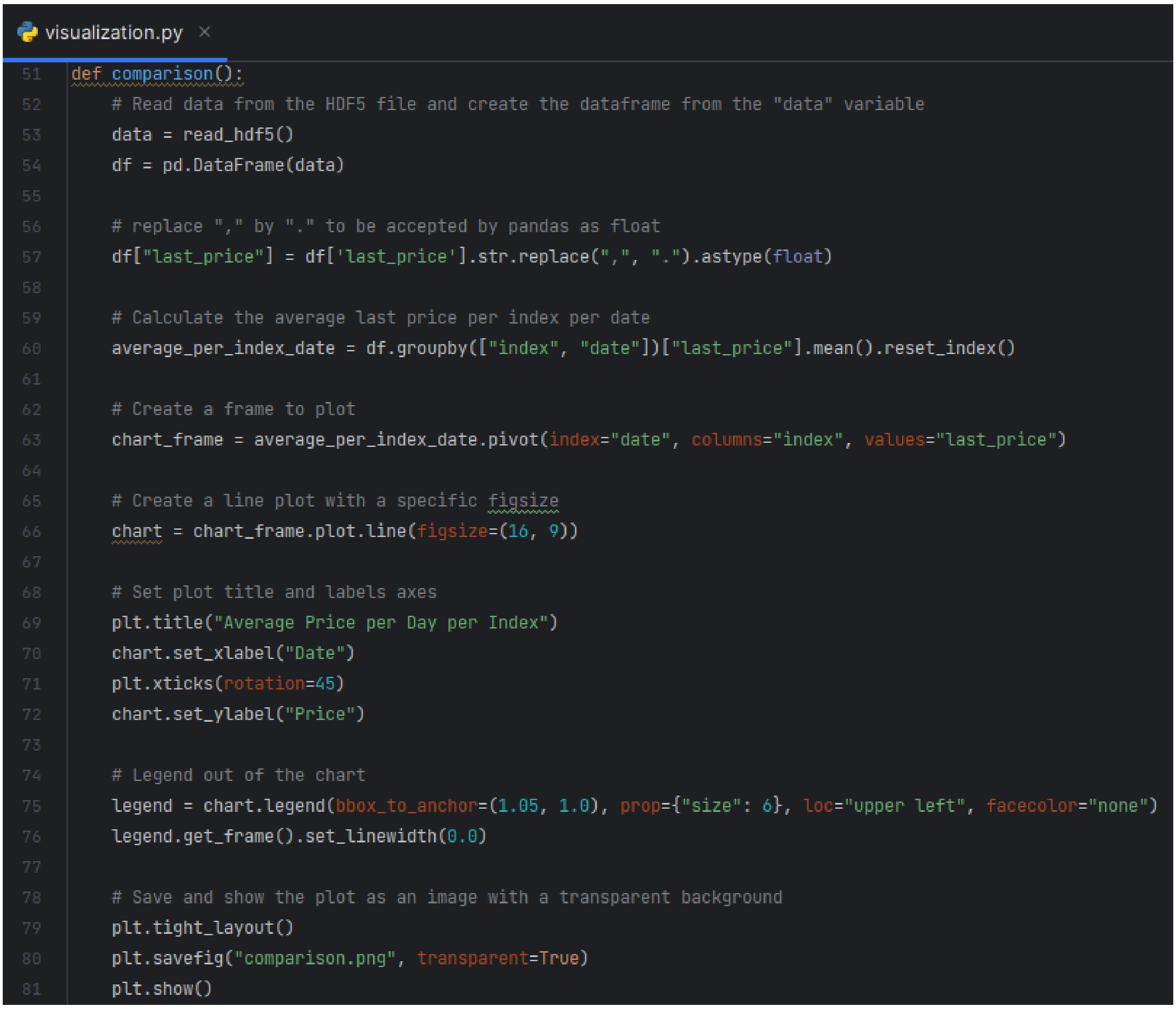
Appendices


Company Last Price Over Time - PSI (PSI20)
Altri
BCP
CTT Correios de Portugal SA
Corticeira Amorim
EDP RenovÃ¡veis
Energias de Portugal
Galp Energia
Greenvolt Energias Renovaveis
Ibersol Reg
Jeronimo Martins
Mota-Engil
Nos SGPS SA
REN
Semapa
Sonae
The Navigator
Price
25
20
15
10
5
0
Date
2023-08-14
2023-08-15
2023-08-16
2023-08-17
2023-08-18
2023-08-21
2023-08-22
2023-08-23
2023-08-24
2023-08-25


Company Last Price Over Time - IBEX 35 (IBEX)
Price
Date
0
20
40
60
80
100
120
140
2023-08-14
2023-08-15
2023-08-16
2023-08-17
2023-08-18
2023-08-21
2023-08-22
2023-08-23
2023-08-24
2023-08-25
ACS
Acciona
Acerinox
Aena
Amadeus
ArcelorMittal
BBVA
Banco de Sabadell
Bankinter
Caixabank
Cellnex Telecom
Corporacion Acciona Energias Renovables
Enagas
Endesa
Ferrovial
Fluidra SA
Grifols
IAG
Iberdrola
Inditex
Indra
Inmob colonial
Laboratorios Farmaceuticos ROVI SA
Logista
Mapfre
Melia Hotels International SA
Merlin Properties SA
Naturgy Energy
Red Electrica
Repsol
Sacyr Valle
Santander
Solaria Energia y Medio Ambiente
Telefonica
Unicaja Banco


Company Last Price Over Time - DAX (GDAXI)
Adidas
Airbus Group
Allianz
BASF
BMW
Bayer
Beiersdorf
Brenntag AG
Commerzbank
Continental
Covestro
Daimler Truck Holding
Deutsche Bank
Deutsche Borse
Deutsche Post
Deutsche Tel.
Dr Ing hc F Porsche Prf
E ON
Fresenius SE
Hannover Rueckversicherung AG
Heidelbergcement
Henkel
Infineon
Mercedes Benz Group
Merck
Mtu Aero Engines Holding AG
Munchener Ruck
Porsche Automobil Holding SE
Qiagen NV
RWE
Rheinmetall
SAP
Sartorius AG Vz
Siemens
Siemens Energy AG
Siemens Healthineers
Symrise AG
Volkswagen VZO
Vonovia
Zalando SE
Price
350
300
250
200
150
100
50
0
Date
2023-08-14
2023-08-15
2023-08-16
2023-08-17
2023-08-18
2023-08-21
2023-08-22
2023-08-23
2023-08-24
2023-08-25


Average Price per Day per Index
DAX (GDAXI)
IBEX 35 (IBEX)
PSI (PSI20)
Price
100
80
60
40
20
Date
2023-08-14
2023-08-16
2023-08-18
2023-08-22
2023-08-24



```python
def comparison():
    # Read data from the HDF5 file and create the dataframe from the "data" variable
    data = read_hdf5()
    df = pd.DataFrame(data)

    # replace "," by "." to be accepted by pandas as float
    df["last_price"] = df['last_price'].str.replace(",", ".").astype(float)

    # Calculate the average last price per index per date
    average_per_index_date = df.groupby(["index", "date"])["last_price"].mean().reset_index()

    # Create a frame to plot
    chart_frame = average_per_index_date.pivot(index="date", columns="index", values="last_price")

    # Create a line plot with a specific figsize
    chart = chart_frame.plot.line(figsize=(16, 9))

    # Set plot title and labels axes
    plt.title("Average Price per Day per Index")
    chart.set_xlabel("Date")
    plt.xticks(rotation=45)
    chart.set_ylabel("Price")

    # Legend out of the chart
    legend = chart.legend(bbox_to_anchor=(1.05, 1.0), prop={"size": 6}, loc="upper left", facecolor="none")
    legend.get_frame().set_linewidth(0.0)

    # Save and show the plot as an image with a transparent background
    plt.tight_layout()
    plt.savefig("comparison.png", transparent=True)
    plt.show()
```