University of St.Gallen

Fundamental Stock Screener

May 26, 2019

University of St. Gallen

Skills: Programming with Advanced Computer Languages

Group ID: 1106

Authors:	**Student-ID:**
Till Folger	14-617-765
Verena Konzett	18-613-315
Andreas Kröbl	17-601-014

Contents

1 Task and programming procedure

The purpose of the program is to identify stocks which appear to be fundamentally undervalued. The program scans the S&P 500 or the Dow Jones Industrial Index for stocks which meet the user's criteria. A common ratio to screen for is the Price to Book ratio, which compares the price of a stock to the book value of the company's assets per share. Our program enables the user to screen for three fundamental ratios: Price to Book, Price to Earnings and Debt to Equity. After the screening process the user has the option to print the identified stocks with their respective ratios to an Excel File for future use.

For a solid investment decision, a peer group benchmarking is essential. Therefore, we have implemented a peer benchmarking with the most popular valuation benchmarks. The user can either add peer stocks to the benchmarking by manually entering stock tickers or by using peers, which are recommended by the program itself. By looking at the mean and median of the peer valuation multiples the user can quickly get a picture whether the stock is overpriced or underpriced.

In order to get further insight, the program creates a visual comparison of the stock price movement compared to a selected industry index and its peers. As an additional feature the user can investigate the stock price development and compare it to two moving averages. Finally, two plots targeting volatility shall give the user an understanding of the historic volatility and the rolling volatility. With the help of the program the user can quickly screen through hundreds of stocks and can get a first understanding of a potential investment target.

2 Code

We recommend using Spyder to execute the code.

Preamble:
Necessary PIPs in Anaconda Prompt:

Pandas:	pip install pandas
Numpy:	pip install numpy
Request	pip install request
Matplotlib:	pip install matplotlib
Beautiful Soup:	pip install beautifulsoup4
Selenium:	pip install selenium
Datetime	pip install datetime
Time	pip install time

The package 'Selenium' requires the user to install the latest google chrome driver and to specify the executable path. See line 16.

Link to download chrome driver: http://chromedriver.chromium.org/downloads

The function print_full() was created in order to display the whole set of possible peer companies, which the user can access by typing in help when adding peer companies. Without *print_full* only 60 companies instead of 500 would be shown.

```
1.  # Load libraries
2.  import pandas as pd
3.  from pandas_datareader import data as pdr
4.  import numpy as np
5.  import datetime as dt # package for dates
6.  import time
7.  import matplotlib.pyplot as plt # for graphs
8.  
9.  
10. # packages for scraping:
11. import bs4 as bs # scraping
12. import requests # get request
13. from selenium import webdriver # scraping
14. from selenium.common.exceptions import NoSuchElementException # error handling of scraping
15. 
16. # set executable_path
17. executable_path=r"C:\Users\Andi\Desktop\test2\chromedriver.exe"
18. #executable_path=r"C:\Users\Till\Desktop\test\chromedriver_win32\chromedriver.exe"
19. 
20. #define a function to print whole data set
21. def print_full(x) -> None:
22.     pd.options.display.max_columns = x.shape[1] # display all columns
23.     pd.options.display.max_rows = x.shape[0] # display all rows
24.     print(x)
```

Step 1:

As a first step the indexes are defined, and its constituents are included in an array for each index. The test list has the purpose to efficiently check the code without screening too many companies. With *pd.read_excel('Ticker_List.xlsx')* we access an Excel file, which includes all possible tickers and respective company names.

```
###################################
## STEP 1: PROVIDE STOCK INDICES
###################################

'''''
We provide two current stock indices with tickers and a short test list where you can choose from.
Additionally, we provide a ticker list with company names from which the user can choose in STEP 3.
'''

# Current S&P Companies
SP500 = ['A', 'AAL', 'AAP', 'AAPL', 'ABBV', 'ABC', 'ABMD', 'ABT', 'ACN', 'ADBE'…]
```

```

# Current Dow Jones Industrial Companies
DJI = ['MMM', 'AXP', 'AAPL', 'CAT', 'CVX',  'CSCO', 'DWDP', 'XOM', 'INTC', 'IBM', …]

# test list
test = ['GT', 'IVZ', 'LNC', 'MET', 'MOS', 'MPC', 'MRO', 'PRU', 'TAP', 'UNM', 'WRK']

# load ticker data
Ticker_List = pd.read_excel('Ticker_List.xlsx')
```

Step 2:

In a second step the user selects an index and inputs all the relevant key metrics. The program scrapes the required information from yahoo finance. We use the function *soup.find_all* from beautiful soup as it works well for html codes and is fast. The user further can save the results from the scrape into an excel file called "MeetRequirements".

```
###############################
# STEP 2: STOCK SCREENER
###############################

###############################
# STEP 2.1 ASK USER FOR INPUT
###############################

# print welcome message
print("\nWelcome to the Fundamental Stock Screener!\nThis tool helps you to efficiently
 filter through stocks for your fundamental stock analysis and stock picking!")

# Ask user for index
print("\nYou can choose between two stock indices: Dow Jones Industrial (DJI) and S&P50
0. Which one do you want to choose?")

# Check for user input
while True:
    try:
        # first get index from user:
        index = input(("Please enter either DJI or SP500 here: "))
        # check for input
        if index.lower() == "dji": #check if input is equal to index dji
            # use DJI later
            usedindex = DJI
            break # everything ends
        elif index.lower() == "sp500": #check if input is equal to index sp500
            # use SP500 later
            usedindex = SP500
            break # everthing ends
        elif index.lower() == "test": #check if input is equal to index test
            # use test later
            usedindex = test
            break
        #otherwise, give user again chance for input
        else:
            print("\nError. You must insert exactly one of the indices - DJI or SP500 o
r test - given. Please try again.") # print error message if input is not a number
            continue # ask user again to give an input


    # brauchen wir das überhaupt? (ist bei den anderen loops gleich...)
    # any other error:
    except:
        print("\nError. You must insert exactly one of the indices - DJI or SP500 or test - given. Please try again.") # print error message if input is not a number
        continue # ask user again to give an input

print("\nThe stock index " + index + " will be analysed now. You now have to set the minimum requirements a stock must meet.")


# lists to save data:
statistics_names = []
statistics_values = []

# ask user for maximum pbr
print("\nWhat is the maximum Price to Book Ratio a stock may have? Note that investors usually set a number around 2 here. ")
while True:
    try:
        # let user insert a number
        max_pbr = float(input(("Please insert the maximum number of the Price to Book Ratio here: ")))
        # Append ratio name and value to lists:
        statistics_values.append(max_pbr)
        statistics_names.append(' Maximum Price to Book Ratio')
        break
    except ValueError:
        print("\nError. You have to enter a number. Make sure to use a dot to separate decimals. Please try again.") # print error message if input is not a number
        continue # ask user again to give an input which size will be checked again


# ask user for maximum p/e
print("\nWhat is the maximum Price to Earnings Ratio a stock may have? Note that investors usually set here a number around 20.")
while True:
    try:
        # let user insert a number for ratio
        max_pe = float(input(("Please insert the maximum number of the Price to Earnings Ratio here: ")))
        # Append ratio name and value to lists
        statistics_values.append(max_pe)
        statistics_names.append('Maximum Price to Earnings Ratio')
        break
    except ValueError:
        print("\nError. You have to enter a number. Please try again.") # print error message if input is not a number
        continue # ask user again to give an input which will be checked again


# ask user for maximum d/e
print("\nWhat is the maximum Debt to Equity Ratio a stock may have? Set it around 100 to get a result.")
while True:
    try:
        # let user insert a number for ratio
        max_de = float(input(("Please insert the maximum number of the Debt to Equity Ratio here: ")))
        # Append ratio name and value to lists
```

```python
        statistics_values.append(max_de)
        statistics_names.append('Maximum Debt to Equity Ratio')
        break
    except ValueError:
        print("\nError. You have to enter a number. Please try again.") # print error message if input is not a number
        continue # ask user again to give an input which size will be checked again

# print message what follows
print("\nGreat! You will now get the stocks that meet the given requirements: \n")


        Statistics_Frame = pd.DataFrame({
                      'Ratio Name': statistics_names,
                      'Ratio Value': statistics_values})

        print(Statistics_Frame)

        print("\nThis process may take a few minutes. Please be patient.")


        ####################################################
        ## STEP 2.2: SCRAPE DATA AND CHECK FOR REQUIREMENTS
        ####################################################

        # Ouput: List of stocks that meet requirements

        # create empty lists to save results in function
        meetrequirements = []
        meetrequirements_stock_name = []
        PBR_list = []
        PE_list = []
        DE_list = []
        PEG_list = []
        error_list = []

        #Webscrapper using Yahoo HTML Code / Here we use BeautifulSoup as it is faster than Selenium and works accurately for this data
        def yahooKeystats(stock):
            try:
                # url to be visited for each stock
                url = "https://finance.yahoo.com/quote/" + stock + "/key-statistics"
                # send get request for url and parse text
                r=requests.get(url)
                soup = bs.BeautifulSoup(r.text,"html.parser")
                pbr = soup.find_all('td',{'class':'Fz(s) Fw(500) Ta(end)'})[6].text
                stock_name = soup.find_all('h1',{'class':'D(ib) Fz(16px) Lh(18px)'})[0].text

                # Check whether stocks meet some investment requirements set by user

                # 1.) Price to Book Ratio must be smaller than 1
                if float(pbr) < float(max_pbr):
                    #Used PE not PEG as Yahoo doesn't have this for a lot of stocks = producing errors
                    # then scrape p/e:
                    PE = soup.find_all('td',{'class':'Fz(s) Fw(500) Ta(end)'})[2].text

                    # 2.) PE ratio must be smaller than some value
                    if float(PE) < float(max_pe):
                        # then scrape debt to equity ratio:
```

```python
                    DE = soup.find_all('td',{'class':'Fz(s) Fw(500) Ta(end)'})[26].text

                    # 3.) DE ratio must be smaller than some value
                    if float(DE) < float(max_de):
                        PEG5 = soup.find_all('td',{'class':'Fz(s) Fw(500) Ta(end)'})[4].text

                        # Output given to the user:
                        print(stock_name + ' meets the requirements:')
                        print('----------------------------------------')
                        print('Price to Book Ratio: ', pbr)
                        print('Price to Earnings Ratio: ', PE)
                        print('Total Debt to Equity: ', DE)
                        print('PEG Ratio: ', PEG5)
                        print('----------------------------------------')

                        # Append ratios and the ticker of a stock to lists created above
                        meetrequirements.append(stock)
                        meetrequirements_stock_name.append(stock_name)
                        PBR_list.append(pbr)
                        PE_list.append(PE)
                        DE_list.append(DE)
                        PEG_list.append(PEG5)

        # Error handling
        except Exception as e:
          error_list.append(e) #write a list with all error messages

    # Loop for each stock in the index list
    for eachStock in usedindex:
        yahooKeystats(eachStock)
        time.sleep(.1)


    # print list of stocks that meet the requirements set
    print("The tickers for the filtered stocks are: ")
    print(meetrequirements)


    # Create dataframe with all companies that meet requirements with all ratios
    MeetRequirements_Frame = pd.DataFrame({
                  'Company': meetrequirements_stock_name,
                  'Price to Book': PBR_list,
                  'Price to Earnings': PE_list,
                  'Total Debt to Equity': DE_list,
                  'PEG Ratio': PEG_list,})


    # Ask user whether to print an excel file of the stocks that meet the requirements
    while True:
        try:
            requirements_excel = input(("Do you want to have an Excel file of all stocks that meet the requirements including ratios? Insert Yes or No: "))
            if requirements_excel.lower() == "no":
                break # nothing happens

            elif requirements_excel.lower() == "yes":
                # write excel file
                with pd.ExcelWriter('MeetRequirements.xlsx') as writer:
                    MeetRequirements_Frame.to_excel(writer, sheet_name='Stocks with Ratios')
                    Statistics_Frame.to_excel(writer, sheet_name='Given Requirements')
                # give user information
                print("\nWe have now saved the file named 'MeetRequirements.xlsx' in your current working directory.")
                break # everthing ends

            else:
                print("\nError. Please insert 'Yes' or 'No'. Please try again.") # print error message
                continue  # ask user again to give an input again after error message

        # error handling for any other error
        except:
            print("\nError. Please insert 'Yes' or 'No'. Please try again.") # print error message
            continue  # ask user again to give an input again after error message
```

Step 3:

In the third step the user decides on whether to progress with a peer analysis or to exit. Peers are selected manually or recommended by the program. By entering ‘help’ in the manual setting the user gets an overview of all available peer companies with their tickers and can therefore easily find the ticker for the desired peer company. The recommendation is based on data from NASDAQ and scraped with the package ‘Selenium’. ‘Beautiful Soup’ in this case was not able to scrape the data as the website is not html coded. Selenium also is extremely easy to use by coping xpaths in to the function *driver.find_element_by_xpath()* to target data on the respective website.

```
##############################
# STEP 3: PEER GROUP ANALYSIS
##############################

'''''
Here you get the chance to conduct a peer group analysis with some stock. The peers
can be either manually inserted or you get a recommendation of peers.
'''

# Ask user whether to conduct a peer group analysis with the stock the user inserted:
print("\nDo you want to continue with a peer-group analysis for a chosen stock? ")

# Check for user input and conduct the analysis if requested
while True:
    try:
        group_an = input(("Please type Yes or No: "))
        if group_an.lower() == "no":
            # Give user information:
            print("\nNo peer group analysis provided.")
            break #no fundamental stock analysis ends

        elif group_an.lower() == "yes":

            ###########################
            # START PEER GROUP ANALYSIS
            ###########################
```

```python

            # Ask user which ticker to further analyse
            while True:
                try:
                    # Ask for user input
                    target_stock = input(("Please pick a stock out of the filtered stocks from above or from the SP500 / DJI and insert its ticker: "))
                    # check if ticker is in SP500
                    if target_stock.upper() in SP500:
                        # continue with analysis
                        break
                    # check if ticker is in DJI
                    elif target_stock.upper() in DJI:
                        break
                    # Let user type in another ticker which is in index
                    else:
                        print("Error. Please insert a ticker from the given indices.") # print error message
                        continue  # ask user again to give an input again after error message
                # error handling
                except:
                    print("\nError. Please insert a ticker from the given indices.") # print error message
                    continue

            # Ask user how to compare stock:
            print("\nDo you want to insert stocks for comparison manually or continue with recommended stocks?")
            while True:
                try:
                    option_no = input(("Please type 1 for manual insertion or 2 for recommendations: "))

                    #######################################
                    # OPTION 1: MANUAL PEER GROUP ANALYSIS
                    ######################################
                    if option_no == "1":
                        while True:
                            try:
                                # Ask the user for input
                                #print("\nA list of all possible stocks and its tickers can be found here:")
                                #print_full(Ticker_List)

                                peers = input(("\nIf you insert 'help' a full list with tickers will show up where you can choose from. Otherwise, directly insert the tickers here separated by commas: "))
                                # copy input to new variable to check if user wants help

                                # Transform user input
                                # split input by comma to get a list
                                peers = peers.split(',')
                                # transform each ticker to upper letters:
                                peers = [i.upper() for i in peers]
                                # delete spaces in each entry of the list
                                peers = [i.strip(' ') for i in peers]
                                # delete duplicates from peers list
                                peers = list(dict.fromkeys(peers))

                                # merge indices
                                mergedindices = SP500 + DJI
                                # delete duplicates from mergedindices list
                                mergedindices = list(dict.fromkeys(mergedindices))


                                # user needs help
                                if peers == ['HELP']:
                                    # print full Ticker_List (data loaded in preamble)

                                    print_full(Ticker_List)
                                    continue

                                # user inserted something else than help
                                else:
                                    # test if all tickers exist in list check = true, otherwise, check = false
                                    check = False
                                    check = any(elem in peers for elem in mergedindices)

                                    # if all tickers exist in list (check = true)
                                    if check is True:
                                        # Stocks we will analyse now:
                                        print("\nWe now continue with the peer group analysis with the following tickers:")
                                        print(peers)
                                               break # if check is true ends

                                           # if tickers do not exist in our list (check = false), give user chance to inser new tickers:
                                           else:
                                               print("\nError. The tickers do not exist in our list. Please try again.")
                                               continue # with user input of peer stocks (option 1)

                                   # error handling
                                   except:
                                       print("\nError. The tickers do not exist in our list. Please try again. ") # error message
                                       continue  # with user input of peer stocks (option 1)

                               break # everything ends of option number 1

                           ################################
                           # OPTION 2: PEER RECOMMENDATION
                           ################################
                           elif option_no == "2":
                               print("\nPlease wait a moment while we identify your peer stocks. \nThis may take a few seconds.")

                               # SCRAPER SETTINGS
                               # set options for driver
                               options = webdriver.ChromeOptions()
                               options.add_argument('--ignore-certificate-errors')
                               options.add_argument("--test-type")
                               options.binary_location = "/usr/bin/chromium"

                               # start driver
                               driver = webdriver.Chrome(executable_path=executable_path)
```

```

# url to be visited
url = ('https://www.nasdaq.com/symbol/' + target_stock + '/stock-comparison')

# get request
driver.get(url)

# Wait for 5 seconds
time.sleep(5)

# handle cookies
cookies_list = driver.get_cookies()
cookies_dict = []
for cookie in cookies_list:
    cookies_dict.append([cookie['name'],cookie['value']])

# create empty lists where we add peer tickers
peers = []
peer_names = []

#loop to scrape tickers
for i in range(1, 5):
    # scrape element
    try:
        # scrape peer ticker
        peer_ticker = driver.find_element_by_xpath('//*[@id="quotes_content_left_SC_Symbol'+str(i)+'"]')
        # get the attribute
        peer_ticker = peer_ticker.get_attribute('value')

        # add ticker to peers list
        peers.append(peer_ticker)

        # scrape peer name
        peer_name = driver.find_element_by_xpath('//*[@id="quotes_content_left_SC_CompanyPanel"]/table/tbody/tr[1]/td['+str(i+1)+']/div[2]/a')

        # add name to peer_names list
        peer_names.append(peer_name.text)

    # error handling if there does not exist such an element
    except NoSuchElementException:
        print("Peer number" + str(i) + "out of 5 not found!")
        pass

# quit / close river
driver.quit()

# create dataframe with stocks and tickers
Peer_Recommendation_Frame = pd.DataFrame({
              'Peer Name': peer_names},
              index = peers)
# set ticker as index
Peer_Recommendation_Frame.index.name = 'Ticker'

# print list of peer tickers
print("\nWe identified the following peer stocks for you:")
```

```
                                print(Peer_Recommendation_Frame)


                                # Message if no peers found:
                                if (len(peers) == 0):
                                    print("Sorry, no peers found! Enter your own peers if you want to.")

                                break # option number 2 ends

                            # if input is neither 1 or 2, give user chance for new input:
                            else:
                                print("\nError. Please insert a number: 1 for manual insertion or 2 for recommendations. Please try again.") # print error message if input is not a number
                                continue  # ask user again to give an input whose size will be checked again

                        # ERROR HANDLING
                        except:
                            print("\nError. Please insert a number: 1 for manual insertion or 2 for recommendations. Please try again.") # print error message if input is not a number
                            continue  # ask user again to give an input
```

Step 4:
In the fifth step a benchmark table is created and filled with data from the selected peers. The benchmark table makes it extremely easy to get an understanding of the target's valuation compared to its peers.

```
#################################
# STEP 4: PEER GROUP VALUATIONS
#################################

#Here you will be provided with financials of the peers from STEP 3 and the company you wanted to analyse. The output will be tables of bechmark valuations.


###########################
# STEP 4.1: PEER BENCHMARK
###########################

            print("\nBelow you can find some information about the benchmark stocks: \n")

            # create empty lists to be filled by loop later
            Forward_PE_list = []
            EV_Revenue_list = []
            EV_EBITDA_list = []

            # Loop to scrape key statistics for chosen stocks
            for targetstock in peers:
                # url to be visited for each stock
                url = "https://finance.yahoo.com/quote/" + targetstock + "/key-statistics"
                # send get request for url
                r=requests.get(url)
                # parse text
                soup = bs.BeautifulSoup(r.text,"html.parser")

                # SCRAPE KEY STATISTICS
                # Forward / PE
                Forward_PE = soup.find_all('td',{'class':'Fz(s) Fw(500) Ta(end)'})[3].text
                # append element to list
                Forward_PE_list.append(Forward_PE)

                # EV / Revenue
                EV_Revenue = soup.find_all('td',{'class':'Fz(s) Fw(500) Ta(end)'})[7].text
                # append element to list
                EV_Revenue_list.append(EV_Revenue)

                # EV / EBITDA
                EV_EBITDA = soup.find_all('td',{'class':'Fz(s) Fw(500) Ta(end)'})[8].text
                # append element to list
                EV_EBITDA_list.append(EV_EBITDA)

            # Create dataframe out of lists
            benchmarks = pd.DataFrame({
                          'Forward PE': Forward_PE_list,
                          'EV / Revenue': EV_Revenue_list,
                          'EV / EBITDA': EV_EBITDA_list},
                          index = peers)

            # give index name
            benchmarks.index.name = 'Company Ticker'

            # Change 'N/A' to nan by pandas
            benchmarks = benchmarks.replace('N/A', np.nan)

            # create list of all statistics
            statistics = ['Forward PE', 'EV / Revenue', 'EV / EBITDA']

            # loop over each statistics:
            for stat in statistics:
                # change datatype for each statistics
                benchmarks[stat] = pd.to_numeric(benchmarks[stat])

            # Compute mean and median and add it to the end of the dataframe
            benchmarks.loc['Average'] = benchmarks[:(len(benchmarks))].mean()
            benchmarks.loc['Median'] = benchmarks[:(len(benchmarks)-1)].median()

            # Show user benchmark analysis
            print_full(benchmarks)



#######################################
# STEP 4.2: TARGET COMPANY FINANCIALS
#######################################

            # create a list of index
            Target_Stock = []
            Target_Stock.append(target_stock.upper())

            # Create dataframe out of lists
            r=requests.get("https://finance.yahoo.com/quote/" + target_stock + "/key-statistics?p=" + target_stock)
            # parse text
            soup = bs.BeautifulSoup(r.text,"xml")
            # scrape PE Ratio
            Forward_PE_target = soup.find_all('td',{'class':'Fz(s) Fw(500) Ta(end)'})[3].text
            # scrape EV-Revenue
            EV_Revenue_target = soup.find_all('td',{'class':'Fz(s) Fw(500) Ta(end)'})[7].text
            # Scrape EV-EBITDA
            EV_EBITDA_target = soup.find_all('td',{'class':'Fz(s) Fw(500) Ta(end)'})[8].text

            # Create dataframe
            Target_Frame = pd.DataFrame({
                          'Forward PE': Forward_PE_target,
                          'EV / Revenue': EV_Revenue_target,
                          'EV / EBITDA': EV_EBITDA_target},
                          index = Target_Stock)

                  Target_Frame.index.name = 'Company Ticker'

                  Target_Frame = Target_Frame.replace('N/A', np.nan)

                  #Empty row for better overview
                  print("\n", Target_Frame)

```

Step 5:

In the fifth step the user can select an industry benchmark against which the target stock and the peers are graphically compared. Afterwards the user can select to have an absolute price chart including a 50 day and 100 day moving average and a historical volatility plot and a rolling historical volatility plot.

```
##################################################
# STEP 5: PLOTS
#################################################
#################################################
# STEP 5.1: INDEXED STOCK PLOT
################################################

    start = dt.datetime(2015,1,1)
    end = dt.datetime.today()

    print("\nFor a good comparison of stock price development, we will show an indexed stock plot.\nDo you want to add an industry index for that? ")

    # Check for user input
    while True:
      try:
        # Ask user for answer
        industry_index = input(("Please type Yes or No: "))

        # if user doesn't want to add an industry index
        if industry_index.lower() == "no":
          print("The plot will be provided without an industry index.")
          break # no index ends

        # if user wants to add an industry index
        elif industry_index.lower() == "yes":
```

```python

                        # define industry indices
                        tech_index = 'IYW'
                        industrial_index = 'IYJ'
                        pharma_index = 'IHE'
                        finance_index='IYF'
                        energy_index='IYE'
                        telcom_index='IYZ'

                        # let user select industry index:
                        while True:
                            try:
                                add_index = input(("Please select your industry index. \nType 1 for Tech, 2 for Industrials, 3 for Pharma, 4 for Financials, 5 for Energy, 6 for Telecommunications. "))
                                if add_index == "1":
                                    peers.append(tech_index)
                                    print('\nYour selected tech index is: iShares Dow Jones US Technology')
                                    print('\nThe Ticker of your selected Index is: ' + tech_index)
                                    break
                                elif add_index == "2":
                                    peers.append(industrial_index)
                                    print('\nYour selected industrial index is: iShares Dow Jones US Industrial')
                                    print('\nThe Ticker of your selected Index is: ' + industrial_index)
                                    break
                                elif add_index == "3":
                                    peers.append(pharma_index)
                                    print('\nYour selected pharma index is: iShares US Pharmaceuticals')
                                    print('\nThe Ticker of your selected Index is: ' + pharma_index)
                                    break
                                elif add_index == "4":
                                    peers.append(finance_index)
                                    print('\nYour selected finance index is: iShares Dow Jones US Financial Sector')
                                    print('\nThe Ticker of your selected Index is: ' + finance_index)
                                    break
                                elif add_index == "5":
                                    peers.append(energy_index)
                                    print('\nYour selected energy index is: iShares Dow Jones US Financials')
                                    print('\nThe Ticker of your selected Index is: ' + energy_index)
                                    break
                                elif add_index == "6":
                                    peers.append(telcom_index)
                                    print('\nYour selected telecom index is: iShares Dow Jones US Telecom')
                                    print('\nThe Ticker of your selected Index is: ' + telcom_index)
                                    break
                                else:
                                    print("\nError. Please insert an integer between 1 and 6. Please try again.") # print error message if input is not a number
                                    continue  # ask user again to give an input

                            # error handling for industry index selection
                            except:
                                print("\nError. Please insert an integer between 1 and 6. Please try again.") # print error message if input is not a number
                                continue  # ask user again to give an input

                        #Target Stock is added to the peer array in order to be displayed in the graph
                        peers.append(target_stock)
                        break # index choice ends

                    # if input is neither yes nor no:
                    else:
```

```python
                print("\nError. Please insert 'yes' or 'no'. Please try again.") # print error message if input is not a number

                continue  # ask user again to give an input

        # error handling
        except:
            print("\nError. Please insert 'yes' or 'no'. Please try again.") # print error message if input is not a number
            continue  # ask user again to give an input

    # GET DATA AND PLOT IT
    # function to get data from yahoo api
    def yahoodata(ticker_list):
        # Get data from yahoo
        try:
            series = pdr.DataReader(ticker_list, 'yahoo', start, end)
            return series
        # Error handling
        except Exception as e:
            print('Sorry, data is currently not available: ' + str(e))

    # get data
    df = yahoodata(peers)

    # use adj close
    adj_price = df['Adj Close']

    # compute stock returns
    stock_return = adj_price.apply(lambda x: x / x[0])
    stock_return.head() - 1

    # Figure Settings
    # change figure size
    plt.rcParams['figure.figsize'] = [13.5, 9]
    # axis, grids and lines:
    stock_return.plot(grid = True).axhline(y = 1, color = "black", lw = 1)
    # title
    plt.title("Indexed stock price performance comparison", fontsize=14, fontweight='bold')
    # label of x-axis
    plt.xlabel("time")
    # label of y-axis
    plt.ylabel("Indexed stock prices")
    # show plot
    plt.show()
    # store figure
    stock_return_figure = plt.gcf()


################################################
# STEP 5.2: TARGET STOCK & VOLA PLOTS
###############################################

    # Ask user whether he likes additional technical stock data plot
    print("\nDo you want to get additional technical stock data?")
    while True:
        try:
            # let user answer
            tech_stock_data = input("Please write Yes or No: ")
            # if user wants additional technical stock data:
            if tech_stock_data.lower() == "yes":
                # get data from yahoo
                stock_data = yahoodata(target_stock)
                stock_data_close = pd.DataFrame(stock_data.Close)

                # compute means
                stock_data_close['MA_50'] = stock_data_close.Close.rolling(50).mean()
                stock_data_close['MA_100'] = stock_data_close.Close.rolling(100).mean()

                # Figure settings
                # figure size
                plt.figure(figsize = (15,10))
                # activate grid
                plt.grid(True)
                # title
                plt.title("Stock price with 100 day MA and 50 day MA", fontsize=14, fontweight='bold')
                # Plot series with respective names
                plt.plot(stock_data_close['Close'], label= 'Closing Price')
                plt.plot(stock_data_close['MA_100'], label= 'MA 100 day')
                plt.plot(stock_data_close['MA_50'], label= 'MA 50 day')
                plt.legend(loc=2)
                # label x-axis
                plt.xlabel("time")
                # label y-axis
                plt.ylabel("stock price")
                # show plot
                plt.show()
                # store plot
                ma_figure = plt.gcf()

                #VOLATILITY
                # 1. Historic Volatility
                stock_data_close['change'] = np.log(stock_data_close['Close'] / stock_data_close['Close'].shift())

                # Figure settings
                # figure size
                plt.figure(figsize = (10,5))
                # title
                plt.title("Rolling historical volatility", fontsize=14, fontweight='bold')
                # Plot series
                plt.plot(stock_data_close.change, color ='g')
                # label x-axis
                plt.xlabel("time")
                # label y-axis
                plt.ylabel("change")
                # show plot
                plt.show()
                # store plot
                hist_vola_figure = plt.gcf()


                # 2. Rolling historically volatility
                stock_data_close['volatility'] = stock_data_close.change.rolling(21).std().shift()

                # Figure settings
                # figure size
                plt.figure(figsize = (10,5))
                # title
                plt.title("Rolling historical volatility",fontsize=14, fontweight='bold')
                # plot series
                plt.plot(stock_data_close.volatility, color ='orange')
                # label x-axis
                plt.xlabel("time")
                # label y-axis
                plt.ylabel("change")
                # show plot
```

```
                        plt.show()
                        # store plot
                        roll_hist_vola_figure = plt.gcf()

                        break # additional technical data ends

                    # if user does not want an additional analysis:
                    elif tech_stock_data.lower() == "no":
                        print("\nNo additional technical data will be provided.")
                        break # no additional analysis ends


                    else:
                       print("Error. You have to enter Yes or No. Please try again.") # print error message if input is not possible
                       continue # ask user again to give an input

                except ValueError: # error handling for technical analysis
                    print("Error. Data currently not available. Please try again later. ") # print error message if input is not possible
                    continue # ask user again to give an input

                break # fundamental stock analysis ends (yes case)

        # if continuation input for peer-group analysis is neither yes or no, give user another chance:
        else:
            print("\nError. Please insert 'yes' or 'no'. Please try again.") # print error message
            continue  # ask user again to give an input

    # error handling for peer-group analysis
    except ValueError:
        print("\nError. Please insert 'yes' or 'no'. Please try again.") # print error message
        continue  # ask user again to give an input

    break #everything ends

print("\nThanks for using the tool. The program will now end.")
```

3 Example

```
Welcome to the Fundamental Stock Screener!
This tool helps you to efficiently filter through stocks for your fundamental stock analysis and stock picking!

You can choose between two stock indices: Dow Jones Industrial (DJI) and S&P500. Which one do you want to choo
se?
Please enter either DJI or SP500 here: DJI

The stock index DJI will be analysed now. You now have to set the minimum requirements a stock must meet.

What is the maximum Price to Book Ratio a stock may have? Note that investors usually set a number around 2 here.
Please insert the maximum number of the Price to Book Ratio here: 2

What is the maximum Price to Earnings Ratio a stock may have? Note that investors usually set here a number aroun
d 20.
Please insert the maximum number of the Price to Earnings Ratio here: 20

What is the maximum Debt to Equity Ratio a stock may have? Set it around 100 to get a result.
Please insert the maximum number of the Debt to Equity Ratio here: 100

Great! You will now get the stocks that meet the given requirements:
```

	Ratio Name	Ratio Value
0	Maximum Price to Book Ratio	2.0
1	Maximum Price to Earnings Ratio	20.0
2	Maximum Debt to Equity Ratio	100.0

```
This process may take a few minutes. Please be patient.
CVX - Chevron Corporation meets the requirements:
----------------------------------------
Price to Book Ratio:  1.46
Price to Earnings Ratio:  16.41
Total Debt to Equity:  23.84
PEG Ratio:  0.60
----------------------------------------
XOM - Exxon Mobil Corporation meets the requirements:
----------------------------------------
Price to Book Ratio:  1.64
Price to Earnings Ratio:  17.07
Total Debt to Equity:  22.46
PEG Ratio:  1.40
----------------------------------------
TRV - The Travelers Companies, Inc. meets the requirements:
```

```
----------------------------------------
Price to Book Ratio:  1.59
Price to Earnings Ratio:  15.03
Total Debt to Equity:  30.57
PEG Ratio:  0.88
----------------------------------------
WBA - Walgreens Boots Alliance, Inc. meets the requirements:
----------------------------------------
Price to Book Ratio:  1.91
Price to Earnings Ratio:  9.72
Total Debt to Equity:  71.05
PEG Ratio:  1.71
----------------------------------------
The tickers for the filtered stocks are:
['CVX', 'XOM', 'TRV', 'WBA']
```

Do you want to have an Excel file of all stocks that meet the requirements including ratios? Insert Yes or No: No

Do you want to continue with a peer-group analysis for a chosen stock?
Please type Yes or No: Yes

Please pick a stock out of the filtered stocks from above or from the SP500 / DJI and insert its ticker: XOM

Do you want to insert stocks for comparison manually or continue with recommended stocks?
Please type 1 for manual insertion or 2 for recommendations: 2

We identified the following peer stocks for you:

Ticker	Peer Name
CVX	Chevron Corporation
COP	ConocoPhillips
SU	Suncor Energy Inc.
BP	BP p.l.c.

Below you can find some information about the benchmark stocks:

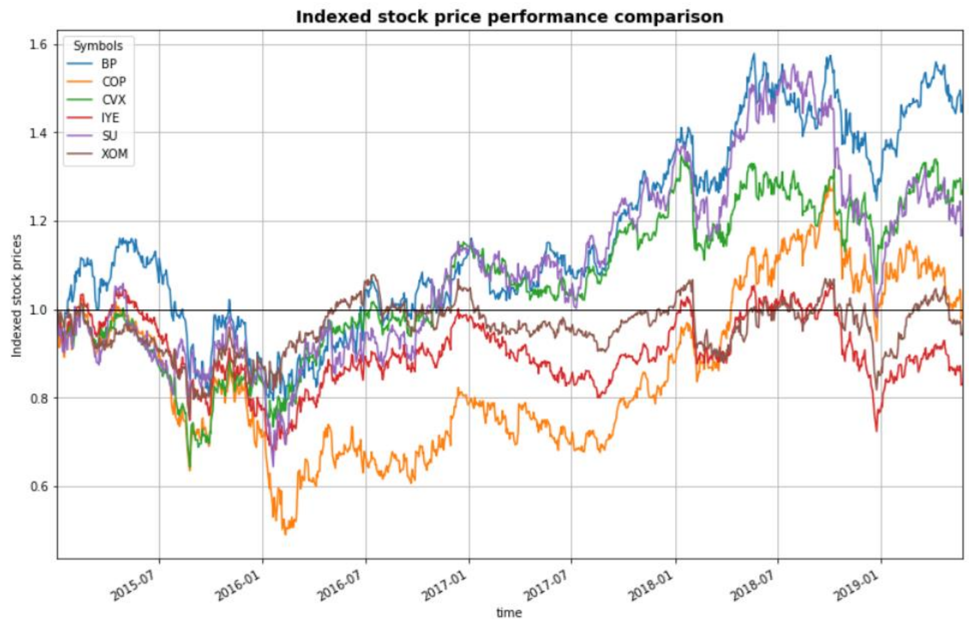
Company Ticker	Forward PE	EV / Revenue	EV / EBITDA
CVX	13.3200	1.65	7.810000
COP	12.8500	2.09	5.010000
SU	15.5500	NaN	NaN
BP	10.2900	0.67	5.910000
Average	13.0025	1.47	6.243333
Median	13.0850	1.65	5.910000

Company Ticker	Forward PE	EV / Revenue	EV / EBITDA
XOM	13.55	1.34	9.81

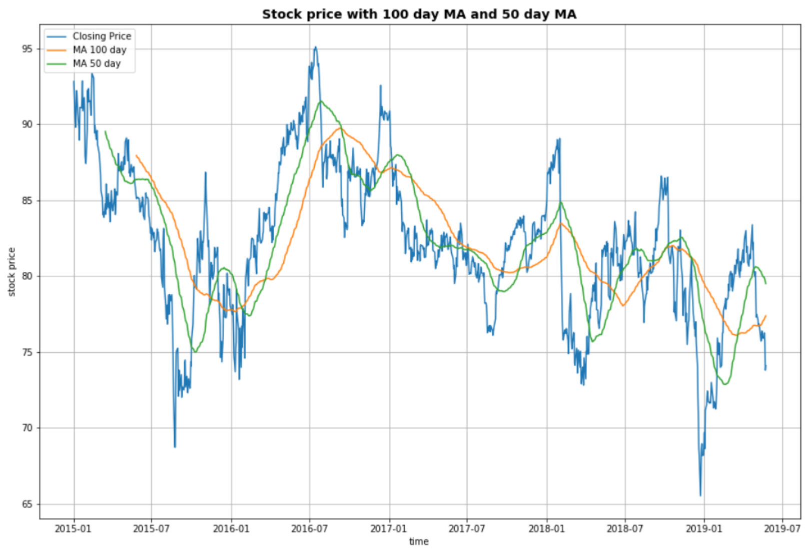
For a good comparison of stock price development, we will show an indexed stock plot.
Do you want to add an industry index for that?

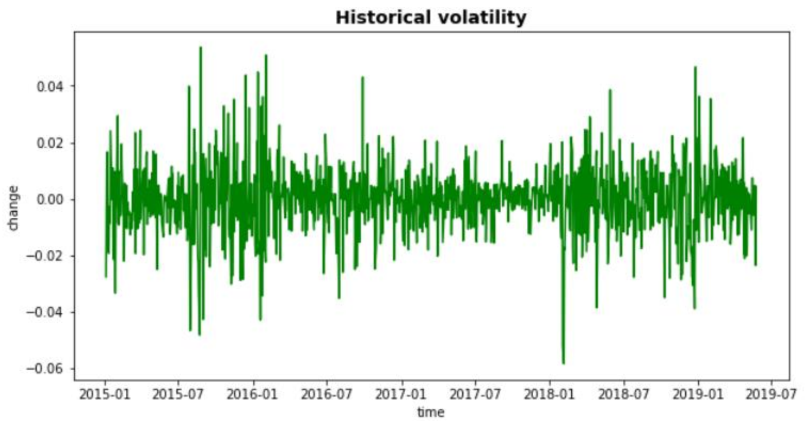
Please select your industry index.
Type 1 for Tech, 2 for Industrials, 3 for Pharma, 4 for Financials, 5 for Energy, 6 for Telecommunications. 5

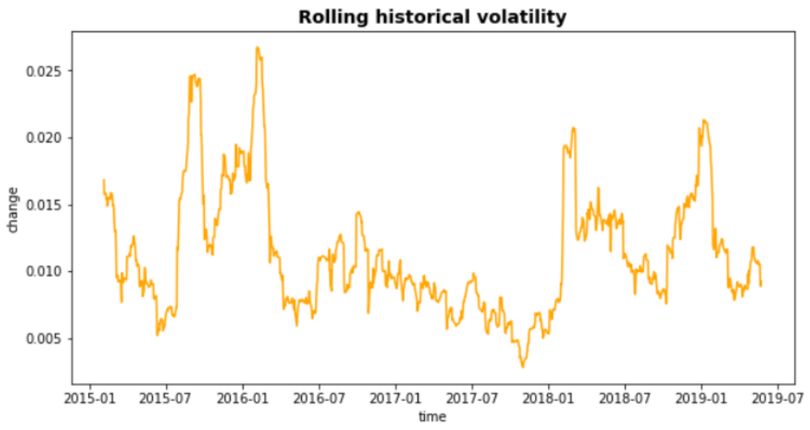
Your selected energy index is: iShares Dow Jones US Financials
The Ticker of your selected Index is: IYE



Do you want to get additional technical stock data?
Please write Yes or No: Yes







Thanks for using the tool. The program will now end.