Milestone II

Michael Perrine

DSC 540

Professor Williams

Milestone 1

Market Analysis

The purpose of this analysis is to research market data. I will gather data from several sources. The goal is to analyze the effect of economic data on stock performance. I will also use the S&P 500 as the benchmark for my stock performance. The sources of my data will come from various sources. I will pull the stock data from FMP a stock price API. The web address for FMP is Documentation V2 - API Reference | FMP. The economic data will come from the FRED. The FRED is managed by the St. louis Fed. The web address is Federal Reserve Economic Data | FRED | St. Louis Fed. Finally, I will pull S&P data from Wikipedia. The web address is S&P 500 - Wikipedia. Economic data will show a relationship between the stock data and the economy. And the S&P will show the relationship between the stock index and the market.
The plan for my analysis will be to add the different data sets into one data frame. I will run an exploratory analysis on the data. I will need to transform the data and determine the variables that make the greatest impact on the analysis. It will be important for me to build graphs to determine the connection between the variables. After I build my charts, I can run a regression analysis and test my model. For the regression analysis I will need to split my data into a training and a testing set. Along with any analysis there are challenges that I will have to manage.

Some challenges that I can foresee are issues with combining the different data sets. Since I am working with data sets from multiple sources, I'll have to make sure they are formatted appropriately. There may also be problems with loading the data into the program. The data will be stored in different formats, so I'll have to ensure that they are formatted in a manner that is compatible across data sets. Another challenge that I can anticipate is overfitting the data. I need to make that the training and testing sets are not in danger of overfitting the data.

Ethical concerns are always a factor when manipulating data. It is important to implement safeguards to prevent ethical. One ethical concern is the misrepresentation of the data. It is important that I don't allow the data to misrepresent the results. Another ethical issue is bias. It's important that I don't allow my personal bias to cause me to see relationships that don't exist.

Milestone 2

The first series of code will load all the libraries for the analysis

In [100…

```
# Load Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from functools import reduce
import yfinance as yf
from datetime import datetime, timedelta
from bs4 import BeautifulSoup
import requests
import lxml
import warnings
import re
import csv
```

In [2]:

```
warnings.filterwarnings('ignore')
```

1. Prep the flat files

The next series of code loads the data from the FRED. I will use three data sets for my analysis. The first data set is the Coincident Economic Activity Index. This index measures the US economic condition. It takes into account non-farm payroll employment, the unemployment rate, average hours worked in manufacturing, and wages and salaries. The second data set is the Consumer Price Index. This measures the price changes of goods and services and is a leading indicator of inflation. The final data set from the FRED is the One Year T-Bill. Fluctuations in T-Bill rates indicate direction of the US Monetary Policy.

In [3]:

```
# This code loads in the cea data, creates a pandas dataframe, and displays the fir
cea = pd.read_csv(r"coincident economic activity index.csv")
cea = pd.DataFrame(cea)
cea.head()
```

Out[3]:

	observation_date	USPHCI
0	1/1/1979	45.36
1	2/1/1979	45.51
2	3/1/1979	45.75
3	4/1/1979	45.82
4	5/1/1979	46.05

In [4]:
```
# This code loads in the cpi data , creates a pandas dataframe, and displays the fi
cpi = pd.read_csv(r"monthly cpi.csv")
cpi = pd.DataFrame(cpi)
cpi.head()
```

Out[4]:

	observation_date	CPIAUCSL
0	1/1/1947	21.48
1	2/1/1947	21.62
2	3/1/1947	22.00
3	4/1/1947	22.00
4	5/1/1947	21.95

In [5]:
```
# This code loads in the t-bill data , creates a pandas dataframe,and displays the
t_bill = pd.read_csv(r"one year t-bill.csv")
t_bill = pd.DataFrame(t_bill)
t_bill.head()
```

Out[5]:

	observation_date	TB1YR
0	7/1/1959	4.34
1	8/1/1959	4.31
2	9/1/1959	4.83
3	10/1/1959	4.69
4	11/1/1959	4.54

The next step is to combine the three data frames into one. This will give me the opportunity to do further analysis. The following series of codes will prep and combine the three data frames.

In [6]:
```
# The first line of code will store all three data frames into one object
dfs = (cea, cpi, t_bill)
```

In [7]:
```
# The second line of code combines all the data into one new data frame.
new_data = reduce(lambda left, right: pd.merge(left, right, on = "observation_date"
```

In [8]:
```
# The third line of code renames the columns in the data frame and displays the com
new_data = new_data.set_axis(["date", "cea", "cpi" , "t_bill"], axis=1)
new_data
```

Out[8]:

	date	cea	cpi	t_bill
0	1/1/1947	NaN	21.480	NaN
1	1/1/1948	NaN	23.680	NaN
2	1/1/1949	NaN	24.010	NaN
3	1/1/1950	NaN	23.510	NaN
4	1/1/1951	NaN	25.380	NaN
...	...	...	...	...
934	9/1/2020	122.45	259.997	0.13
935	9/1/2021	130.21	273.942	0.07
936	9/1/2022	136.71	296.421	3.73
937	9/1/2023	140.69	307.374	5.15
938	9/1/2024	144.20	314.851	3.87

939 rows × 4 columns

Now that I have one data frame I want to reverse the order of the elements. Currently they are in ascending order. I want them in descending order that way when I add my stock data I'll see everything from the most current date to the oldest date.

In [9]:
```
# This line of code reverses the order of the elements
new_data = new_data.iloc[::-1].reset_index(drop=True)
new_data
```

Out[9]:

	date	cea	cpi	t_bill
0	9/1/2024	144.20	314.851	3.87
1	9/1/2023	140.69	307.374	5.15
2	9/1/2022	136.71	296.421	3.73
3	9/1/2021	130.21	273.942	0.07
4	9/1/2020	122.45	259.997	0.13
...	...	...	...	...
934	1/1/1951	NaN	25.380	NaN
935	1/1/1950	NaN	23.510	NaN
936	1/1/1949	NaN	24.010	NaN
937	1/1/1948	NaN	23.680	NaN
938	1/1/1947	NaN	21.480	NaN

939 rows × 4 columns

Now I want to view the data types in my data frame. It's important to make sure all the data types are appropriate for the elements. It looks like I need to change the date data type. It's listed as an object and should be listed as date/time.

In [10]:

```
# This code displays the data type
new_data.dtypes
```

Out[10]:

```
date       object
cea       float64
cpi       float64
t_bill    float64
dtype: object
```

In [11]:

```
new_data.head()
```

Out[11]:

	date	cea	cpi	t_bill
0	9/1/2024	144.20	314.851	3.87
1	9/1/2023	140.69	307.374	5.15
2	9/1/2022	136.71	296.421	3.73
3	9/1/2021	130.21	273.942	0.07
4	9/1/2020	122.45	259.997	0.13

In [12]:

```
# This code changes the date data type from object to date/time and displays the re
new_data["date"] = pd.to_datetime(new_data["date"],format='mixed')
new_data.dtypes
```

Out[12]:
```
date      datetime64[ns]
cea              float64
cpi              float64
t_bill           float64
dtype: object
```

In [13]:
```
# This code displays the new date format
new_data.head()
```

Out[13]:

	date	cea	cpi	t_bill
0	2024-09-01	144.20	314.851	3.87
1	2023-09-01	140.69	307.374	5.15
2	2022-09-01	136.71	296.421	3.73
3	2021-09-01	130.21	273.942	0.07
4	2020-09-01	122.45	259.997	0.13

In [14]:
```
# This code drops NaN values. I chose to drop the
new_data_1 = new_data.dropna(how="any")
new_data_1
```

Out[14]:

	date	cea	cpi	t_bill
0	2024-09-01	144.20	314.851	3.87
1	2023-09-01	140.69	307.374	5.15
2	2022-09-01	136.71	296.421	3.73
3	2021-09-01	130.21	273.942	0.07
4	2020-09-01	122.45	259.997	0.13
...	...	...	...	...
902	1983-01-01	47.12	97.900	8.01
903	1982-01-01	47.54	94.400	12.77
904	1981-01-01	47.16	87.200	12.62
905	1980-01-01	46.74	78.000	10.96
906	1979-01-01	45.36	68.500	9.54

475 rows × 4 columns

In [15]:
```
# This code validates the dimensions of the cleaned data frame
new_data_1.shape
```

Out[15]:
```
(475, 4)
```

Milestone 3

In this section I am scraping data from Wikipedia. My goal is to extract the S&P500 table. To accomplish this goal I will use beautiful soup and requests to import the data. I will need to find the right html tags to create the desired table. The issue with source code is they are not always pretty or easy to work with. Challenges that I face are finding the right tags and dealing with missing or incomplete data.

In [16]:
```
# This code creates a url object
url = 'https://en.wikipedia.org/wiki/S%26P_500'
```

In [17]:
```
# This code requests data from the website
data = requests.get(url)
```

In [ ]:
```
# This code extracts the html from wikipedia and displays it
soup = BeautifulSoup(data.text, 'html.parser')
print(soup.prettify())
```

In [ ]:
```
# This code locates the desired table
table = soup.find_all('table')[1]
table
```

In [ ]:
```
# This code parses the columns for the dataframe
data_columns = soup.find_all('th')[10:20]
data_columns
```

In [ ]:
```
# This code cleans the column titles
data_columns_titles = [title.text.strip() for title in data_columns]
data_columns_titles
```

In [ ]:
```
# This code creates my empty data frame
s_p_index = pd.DataFrame(columns=[data_columns_titles])
```

In [ ]:
```
# I removed the annualized return over column because it is not needed for my analy
s_p_index.drop(columns=['Annualized Return over'], inplace=True)
```

In [ ]:
```
# This code displays the empty dataframe
s_p_index
```

Out[ ]:

Year	Change inIndex	TotalAnnual Return,includingdividends	Value of $1.00invested onJanuary 1, 1970	5 years	10 years	15 years	20 years	25 years

In [ ]:
```
# This code renames the columns
s_p_index.rename(columns={'Year':'year', 'Change inIndex':'change_in_index', 'Total
            'Value of $1.00invested onJanuary 1, 1970':'Value_of_1_invested_on_
```

```
                '5 years':'5_years', '10 years':'10_years', '15 years':'15_years','
                '25 years':'25_years'}, inplace = True)
```

In [ ]:

```
# this code displays the renamed columns
s_p_index
```

Out[ ]:

	year	change_in_index	tot_ann_ret	Value_of_1_invested_on_jan_1_1970	5_years	10_years

In [ ]:

```
# This code locates all the row data for my S&P 500 table
column_data = table.find_all('tr')
column_data
```

In [134…

```
# This code iterates through the rows to find the row data
for row in column_data:
    row_data = row.find_all('td')

    ind_row_data = [data.text.strip() for data in row_data]
    print(ind_row_data)
```

```
[]
[]
['1961', '23.13%', '-', '-', '-', '-', '-', '-', '-']
['1962', '-11.81%', '-', '-', '-', '-', '-', '-', '-']
['1963', '18.89%', '-', '-', '-', '-', '-', '-', '-']
['1964', '12.97%', '-', '-', '-', '-', '-', '-', '-']
['1965', '9.06%', '-', '-', '-', '-', '-', '-', '-']
['1966', '-13.09%', '-', '-', '-', '-', '-', '-', '-']
['1967', '20.09%', '-', '-', '-', '-', '-', '-', '-']
['1968', '7.66%', '-', '-', '-', '-', '-', '-', '-']
['1969', '-11.36%', '-', '-', '-', '-', '-', '-', '-']
['1970', '0.10%', '4.01%', '$1.04', '-', '-', '-', '-', '-']
['1971', '10.79%', '14.31%', '$1.19', '-', '-', '-', '-', '-']
['1972', '15.63%', '18.98%', '$1.41', '-', '-', '-', '-', '-']
['1973', '-17.37%', '-14.66%', '$1.21', '-', '-', '-', '-', '-']
['1974', '-29.72%', '-26.47%', '$0.89', '-2.35%', '-', '-', '-', '-']
['1975', '31.55%', '37.20%', '$1.22', '3.21%', '-', '-', '-', '-']
['1976', '19.15%', '23.84%', '$1.51', '4.87%', '-', '-', '-', '-']
['1977', '-11.50%', '-7.18%', '$1.40', '-0.21%', '-', '-', '-', '-']
['1978', '1.06%', '6.56%', '$1.49', '4.32%', '-', '-', '-', '-']
['1979', '12.31%', '18.44%', '$1.77', '14.76%', '5.86%', '-', '-', '-']
['1980', '25.77%', '32.50%', '$2.34', '13.96%', '8.45%', '-', '-', '-']
['1981', '-9.73%', '-4.92%', '$2.23', '8.10%', '6.47%', '-', '-', '-']
['1982', '14.76%', '21.55%', '$2.71', '14.09%', '6.70%', '-', '-', '-']
['1983', '17.27%', '22.56%', '$3.32', '17.32%', '10.63%', '-', '-', '-']
['1984', '1.40%', '6.27%', '$3.52', '14.81%', '14.78%', '8.76%', '-', '-']
['1985', '26.33%', '31.73%', '$4.64', '14.67%', '14.32%', '10.49%', '-', '-']
['1986', '14.62%', '18.67%', '$5.51', '19.87%', '13.83%', '10.76%', '-', '-']
['1987', '2.03%', '5.25%', '$5.80', '16.47%', '15.27%', '9.86%', '-', '-']
['1988', '12.40%', '16.61%', '$6.76', '15.31%', '16.31%', '12.17%', '-', '-']
['1989', '27.25%', '31.69%', '$8.90', '20.37%', '17.55%', '16.61%', '11.55%', '-']
['1990', '-6.56%', '-3.10%', '$8.63', '13.20%', '13.93%', '13.94%', '11.16%', '-']
['1991', '26.31%', '30.47%', '$11.26', '15.36%', '17.59%', '14.34%', '11.90%', '-']
['1992', '4.46%', '7.62%', '$12.11', '15.88%', '16.17%', '15.47%', '11.34%', '-']
['1993', '7.06%', '10.08%', '$13.33', '14.55%', '14.93%', '15.72%', '12.76%', '-']
['1994', '-1.54%', '1.32%', '$13.51', '8.70%', '14.38%', '14.52%', '14.58%', '10.9
8%']
['1995', '34.11%', '37.58%', '$18.59', '16.59%', '14.88%', '14.81%', '14.60%', '12.2
2%']
['1996', '20.26%', '22.96%', '$22.86', '15.22%', '15.29%', '16.80%', '14.56%', '12.5
5%']
['1997', '31.01%', '33.36%', '$30.48', '20.27%', '18.05%', '17.52%', '16.65%', '13.0
7%']
['1998', '26.67%', '28.58%', '$39.19', '24.06%', '19.21%', '17.90%', '17.75%', '14.9
4%']
['1999', '19.53%', '21.04%', '$47.44', '28.56%', '18.21%', '18.93%', '17.88%', '17.2
5%']
['2000', '-10.14%', '-9.10%', '$43.12', '18.33%', '17.46%', '16.02%', '15.68%', '15.
34%']
['2001', '-13.04%', '-11.89%', '$37.99', '10.70%', '12.94%', '13.74%', '15.24%', '1
3.78%']
['2002', '-23.37%', '-22.10%', '$29.60', '-0.59%', '9.34%', '11.48%', '12.71%', '12.
98%']
['2003', '26.38%', '28.68%', '$38.09', '-0.57%', '11.07%', '12.22%', '12.98%', '13.8
4%']
['2004', '8.99%', '10.88%', '$42.23', '-2.30%', '12.07%', '10.94%', '13.22%', '13.5
```

```
4%']
['2005', '3.00%', '4.91%', '$44.30', '0.54%', '9.07%', '11.52%', '11.94%', '12.48%']
['2006', '13.62%', '15.79%', '$51.30', '6.19%', '8.42%', '10.64%', '11.80%', '13.3
7%']
['2007', '3.53%', '5.49%', '$54.12', '12.83%', '5.91%', '10.49%', '11.82%', '12.7
3%']
['2008', '-38.49%', '-37.00%', '$34.09', '-2.19%', '-1.38%', '6.46%', '8.43%', '9.7
7%']
['2009', '23.45%', '26.46%', '$43.11', '0.41%', '-0.95%', '8.04%', '8.21%', '10.5
4%']
['2010', '12.78%', '15.06%', '$49.61', '2.29%', '1.41%', '6.76%', '9.14%', '9.94%']
['2011', '-0.00%', '2.11%', '$50.65', '-0.25%', '2.92%', '5.45%', '7.81%', '9.28%']
['2012', '13.41%', '16.00%', '$58.76', '1.66%', '7.10%', '4.47%', '8.22%', '9.71%']
['2013', '29.60%', '32.39%', '$77.79', '17.94%', '7.40%', '4.68%', '9.22%', '10.2
6%']
['2014', '11.39%', '13.69%', '$88.44', '15.45%', '7.67%', '4.24%', '9.85%', '9.62%']
['2015', '-0.73%', '1.38%', '$89.66', '12.57%', '7.30%', '5.00%', '8.19%', '9.82%']
['2016', '9.54%', '11.96%', '$100.38', '14.66%', '6.94%', '6.69%', '7.68%', '9.15%']
['2017', '19.42%', '21.83%', '$122.30', '15.79%', '8.49%', '9.92%', '7.19%', '9.6
9%']
['2018', '-6.24%', '-4.38%', '$116.94', '8.49%', '13.12%', '7.77%', '5.62%', '9.0
7%']
['2019', '28.88%', '31.49%', '$153.76', '11.70%', '13.56%', '9.00%', '6.06%', '10.2
2%']
['2020', '16.26%', '18.40%', '$182.06', '15.22%', '13.89%', '9.88%', '7.47%', '9.5
6%']
['2021', '26.89%', '28.71%', '$234.33', '18.48%', '16.55%', '10.66%', '9.52%', '9.7
6%']
['2022', '-19.44%', '-18.11%', '$191.89', '9.43%', '12.56%', '8.80%', '9.80%', '7.6
4%']
['2023', '24.23%', '26.29%', '$242.34', '15.69%', '12.03%', '13.97%', '9.69%', '7.5
6%']
['2024', '23.31%', '25.02%', '$302.97', '14.53%', '13.10%', '13.88%', '10.35%', '7.7
0%']
[]
[]
[]
[]
[]
```

After locating all the data for my analysis, I ran into a snag. I have empty lists in my data and was unable to append it to my empty data frame that I created earlier in the analysis. I tried to find a way to remove the empty lists and was unsuccessful. My solution is to append the list to a csv file and import the file to continue my preprocessing.

In [ ]:

```
# This code displays a list of my columns from my s_p_index dataframe
s_p_index.columns
```

Out[ ]:
```
MultiIndex([(                               'year',),
            (                    'change_in_index',),
            (                        'tot_ann_ret',),
            ('Value_of_1_invested_on_jan_1_1970',),
            (                            '5_years',),
            (                           '10_years',),
            (                           '15_years',),
            (                           '20_years',),
            (                           '25_years',)],
           )
```

In [ ]:
```
# This series of code creates my csv file and imports the data into the file
csv_file= open('s_p_scrape.csv', 'w')
csv_writer = csv.writer(csv_file)
csv_writer.writerow(['year','change_in_index',  'tot_ann_ret', 'Value_of_1_invested
                    '5_years', '10_years', '15_years', '20_years', '25_years'])
for row in column_data[2:]:
    row_data = row.find_all('td')

    ind_row_data = [data.text.strip() for data in row_data]
    #print(ind_row_data)
    csv_writer.writerow(ind_row_data)
csv_file.close()
```

In [ ]:
```
# This code imports the csv file previously created
gspc = pd.read_csv(r's_p_scrape.csv')
```

In [ ]:
```
# This code creates a dataframe for the csv
gspc=pd.DataFrame(gspc)
gspc.head(5)
```

Out[ ]:

	year	change_in_index	tot_ann_ret	Value_of_1_invested_on_jan_1_1970	5_years	10_yea
0	NaN	NaN	NaN	NaN	NaN	Na
1	1961.0	23.13%	-	-	-	
2	NaN	NaN	NaN	NaN	NaN	Na
3	1962.0	-11.81%	-	-	-	
4	NaN	NaN	NaN	NaN	NaN	Na

In [ ]:
```
# This code clears the duplicate rows in the data frame
gspc_cleaned = gspc.drop_duplicates()
gspc_cleaned.head()
```

Out[ ]:

	year	change_in_index	tot_ann_ret	Value_of_1_invested_on_jan_1_1970	5_years	10_yea
0	NaN	NaN	NaN	NaN	NaN	Na
1	1961.0	23.13%	-	-	-	
3	1962.0	-11.81%	-	-	-	
5	1963.0	18.89%	-	-	-	
7	1964.0	12.97%	-	-	-	

In [ ]:

```
# This code drops the first index in the data frame
gspc_cleaned.drop(gspc_cleaned.index[0], inplace= True)
```

In [ ]:

```
# This code shows the cleaned data ready for analysis
gspc_cleaned.head()
```

Out[ ]:

	year	change_in_index	tot_ann_ret	Value_of_1_invested_on_jan_1_1970	5_years	10_yea
1	1961.0	23.13%	-	-	-	
3	1962.0	-11.81%	-	-	-	
5	1963.0	18.89%	-	-	-	
7	1964.0	12.97%	-	-	-	
9	1965.0	9.06%	-	-	-	