```
!pip install yfinance==0.2.40
!pip install vega_datasets

import yfinance as yf
print(yf.__version__)  # Should print 0.2.40

from pandas_datareader import data as pdr
yf.pdr_override()
pdr.get_data_yahoo(["MSFT"]).keys()
```

```
Collecting yfinance==0.2.40
  Downloading yfinance-0.2.40-py2.py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from yfinance==0.2.40) (2.2.2)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.11/dist-packages (from yfinance==0.2.40) (1.26.4)
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.11/dist-packages (from yfinance==0.2.40) (2.32.3)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.11/dist-packages (from yfinance==0.2.40) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.11/dist-packages (from yfinance==0.2.40) (5.3.0)
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from yfinance==0.2.40) (4.3.6)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.11/dist-packages (from yfinance==0.2.40) (2024.2)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.11/dist-packages (from yfinance==0.2.40) (2.4.6)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.11/dist-packages (from yfinance==0.2.40) (3.17.8)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.11/dist-packages (from yfinance==0.2.40) (4.12.3)
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.11/dist-packages (from yfinance==0.2.40) (1.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.11.1->yfinance==0.2.40) (2.6)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.11/dist-packages (from html5lib>=1.1->yfinance==0.2.40) (1.17.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from html5lib>=1.1->yfinance==0.2.40) (0.5.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.3.0->yfinance==0.2.40) (2.8.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.3.0->yfinance==0.2.40) (2024.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance==0.2.40) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance==0.2.40) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance==0.2.40) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance==0.2.40) (2024.12.14)
Downloading yfinance-0.2.40-py2.py3-none-any.whl (73 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 73.5/73.5 kB 859.4 kB/s eta 0:00:00
Installing collected packages: yfinance
  Attempting uninstall: yfinance
    Found existing installation: yfinance 0.2.51
    Uninstalling yfinance-0.2.51:
      Successfully uninstalled yfinance-0.2.51
Successfully installed yfinance-0.2.40
Requirement already satisfied: vega_datasets in /usr/local/lib/python3.11/dist-packages (0.9.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from vega_datasets) (2.2.2)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas->vega_datasets) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->vega_datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->vega_datasets) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->vega_datasets) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas->vega_datasets) (1.17.0)
0.2.40
yfinance: pandas_datareader support is deprecated & semi-broken so will be removed in a future verison. Just use yfinance.
[*********************100%%**********************]  1 of 1 completed
Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

```
data = yf.download("AAPL MCD DIS F", interval="1mo", start="2018-12-01", end="2023-12-31")
data=data.dropna()
data.to_csv("data.csv")
import os
print(os.getcwd())
```

```
[*********************100%***********************]  4 of 4 completed
/content
```

```
!pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

```
import pandas as pd
```

```
file_path = "data.csv"
data = pd.read_csv(file_path)
```

```
data = pd.read_csv("data.csv", nrows=100)
```

```
import pandas as pd

# Replace 'data.csv' with the correct file path
file_path = "data.csv"

try:
    # Load the CSV file
    data = pd.read_csv(file_path)
    print("Data loaded successfully!")

    # Display all rows by adjusting Pandas display settings
    pd.set_option('display.max_rows', None)
    display(data)

    # Check if 'Adj Close' column exists
    if 'Adj Close' in data.columns:
        adj_close = data['Adj Close']
        print("Adj Close column extracted:")
        print(adj_close)
    else:
        print("The 'Adj Close' column does not exist in the data.")

except FileNotFoundError:
    print(f"File not found: {file_path}")
except pd.errors.EmptyDataError:
    print("The file is empty!")
except Exception as e:
    print(f"An error occurred: {e}")
```

```
Data loaded successfully!
```

	Price	Adj Close	Adj Close.1	Adj Close.2	Adj Close.3	Close	Close.1	Close.2	Close.3
0	Ticker	AAPL	DIS	F	MCD	AAPL	DIS	F	MCD

0	Ticker	AAPL	DIS	F	MCD	AAPL	DIS	F	MCD	
1	Date	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	2018-12-01	37.66563034057617	106.18061065673828	5.748631477355957	154.43228149414062	39.435001373291016	109.6500015258789	7.650000095367432	177.57000732421875	46
3	2019-01-01	39.74302673339844	108.82910919189453	6.612804889678955	155.48460388183594	41.61000061035156	111.5199966430664	8.800000190734863	178.77999877929688	
4	2019-02-01	41.345252990722656	110.11725616455078	6.705075740814209	159.88525390625	43.287498474121094	112.83999633789062	8.770000457763672	183.83999633789062	43.
5	2019-03-01	45.551334381103516	108.35093688964844	6.712718963623047	166.20712280273438	47.48749923706055	111.02999877929688	8.779999732971191	189.89999389648438	49
6	2019-04-01	48.122074127197266	133.6650390625	7.989511966705322	172.92019653320312	50.16749954223633	136.97000122070312	10.449999809265137	197.57000732421875	52.
7	2019-05-01	41.98301696777344	128.85397338867188	7.395251750946045	173.53282165527344	43.76750183105469	132.0399932861328	9.520000457763672	198.27000427246094	53
8	2019-06-01	47.64537811279297	136.2705841064453	7.946787357330322	182.81570434570312	49.47999954223633	139.63999938964844	10.229999542236328	207.66000366210938	50
9	2019-07-01	51.285221099853516	139.5592803955078	7.403019428253174	185.50958251953125	53.2599983215332	143.00999450683594	9.529999732971191	210.72000122070312	55.
10	2019-08-01	50.250091552734375	134.77752685546875	7.229687213897705	191.89218139648438	52.185001373291016	137.25999450683594	9.170000076293945	217.97000122070312	54
11	2019-09-01	54.12126541137695	127.96306610107422	7.2218017578125	190.02171325683594	55.99250030517578	130.32000732421875	9.15999984741211	214.7100067138672	56
12	2019-10-01	60.11164855957031	127.57028198242188	6.772409439086914	174.0825958251953	62.189998626708984	129.9199981689453	8.59000015258789	196.6999969482422	
13	2019-11-01	64.57966613769531	148.83856201171875	7.260186672210693	172.11778259277344	66.8125	151.5800018310547	9.0600004196167	194.47999572753906	
14	2019-12-01	71.17213439941406	142.01425170898438	7.452510356903076	176.00868225097656	73.4124984741211	144.6300048828125	9.300000190734863	197.61000061035156	73
15	2020-01-01	75.0161361694336	136.62222290039062	7.067863941192627	190.5803680419922	77.37750244140625	138.30999755859375	8.819999694824219	213.97000122070312	8
16	2020-02-01	66.2544174194336	116.2143325805664	5.672214508056641	172.94476318359375	68.33999633789062	117.6500015258789	6.960000038146973	194.1699981689453	81
17	2020-03-01	61.77869415283203	95.42119598388672	3.936321496963501	148.19679260253906	63.5724983215332	96.5999984741211	4.829999923706055	165.35000610351562	
18	2020-04-01	71.37745666503906	106.83026123046875	4.148214817047119	168.1027374267578	73.44999694824219	108.1500015258789	5.090000152587891	187.55999755859375	73
19	2020-05-01	77.24219512939453	115.86860656738281	4.65349817276001	166.9913787841797	79.48500061035156	117.30000305175781	5.710000038146973	186.32000732421875	81
20	2020-06-01	88.86656951904297	110.14926147460938	4.955038070678711	166.43560791015625	91.19999694824219	111.51000213623047	6.079999923706055	184.47000122070312	93
21	2020-07-01	103.541259765625	115.51300048828125	5.3869733810424805	175.28652954101562	106.26000213623047	116.94000244140625	6.610000133514404	194.27999877929688	106
22	2020-08-01	125.73838806152344	130.26080322265625	5.558119297027588	192.64556884765625	129.0399932861328	131.8699951171875	6.820000171661377	213.52000427246094	
23	2020-09-01	113.05035400390625	122.56587219238281	5.427722930908203	199.1905059814453	115.80999755859375	124.08000183105469	6.659999847412109	219.49000549316406	137
24	2020-10-01	106.2659683227539	119.77040100097656	6.299744129180908	193.3007049560547	108.86000061035156	121.25	7.730000019073486	213.0	125
25	2020-11-01	116.21315002441406	146.20384216308594	7.39995813369751	197.3301239013672	119.05000305175781	148.00999450683594	9.079999923706055	217.44000244140625	121
26	2020-12-01	129.75157165527344	178.96905517578125	7.163616180419922	195.89205932617188	132.69000244140625	181.17999267578125	8.789999961853027	214.5800018310547	13
27	2021-01-01	129.0377655029297	166.1178436279297	8.581670761108398	189.73902893066406	131.9600067138672	168.1699981689453	10.529999732971191	207.83999633789062	145
28	2021-02-01	118.57469177246094	186.733154296875	9.535187721252441	188.18707275390625	121.26000213623047	189.0399932861328	11.699999809265137	206.13999938964844	13
29	2021-03-01	119.62348175048828	182.26834106445312	9.983424186706543	205.87844848632812	122.1500015258789	184.52000427246094	12.25	224.13999938964844	128
30	2021-04-01	128.7409210205078	183.75001525878906	9.404792785644531	216.8456573486328	131.4600067138672	186.02000427246094	11.539999961853027	236.0800018310547	137
31	2021-05-01	122.03260040283203	176.46995544433594	11.84156322479248	214.83407592773438	124.61000061035156	178.64999389648438	14.529999732971191	233.88999938964844	134
32	2021-06-01	134.35498046875	173.62510681152344	12.110506057739258	213.3421630859375	136.9600067138672	175.77000427246094	14.859999656677246	230.99000549316406	137
33	2021-07-01	143.08566284179688	173.87205505371094	11.368878364562988	224.16671752929688	145.86000061035156	176.02000427246094	13.949999809265137	242.7100067138672	

34	2021-08-01	148.94215393066406	179.08761596679688	10.619104385375977	219.31784057617188	151.8300018310547	181.3000030517578	13.029999732971191	237.4600067138672	153
35	2021-09-01	139.01658630371094	167.1056365966797	11.54002571105957	223.9098358154297	141.5	169.1699981689453	14.15999984741211	241.11000061035156	157
36	2021-10-01	147.17091369628906	167.00686645507812	13.919747352600098	228.0330810546875	149.8000030517578	169.07000732421875	17.079999923706055	245.5500030517578	15
37	2021-11-01	162.3988800048828	143.1317901611328	15.63934326171875	227.15086364746094	165.3000030517578	144.89999389648438	19.190000534057617	244.60000610351562	16
38	2021-12-01	174.7081756591797	152.99989318847656	17.01223373413086	250.32672119140625	177.57000732421875	154.88999938964844	20.770000457763672	268.07000732421875	18
39	2022-01-01	171.96311950683594	141.225341796875	16.62726402282715	242.27725219726562	174.77999877929688	142.97000122070312	20.299999237060547	259.45001220703125	182
40	2022-02-01	162.4588165283203	146.6483612060547	14.456938743591309	228.56890869140625	165.1199951171875	148.4600067138672	17.559999465942383	244.77000427246094	176
41	2022-03-01	172.0147247314453	135.4862518310547	13.921802520751953	232.19732666015625	174.61000061035156	137.16000366210938	16.90999984741211	247.27999877929688	179
42	2022-04-01	155.3068084716797	110.26779174804688	11.657760620117188	233.96266174316406	157.64999389648438	111.62999725341797	14.15999984741211	249.16000366210938	178
43	2022-05-01	146.6277618408203	109.09231567382812	11.337517738342285	236.82662963867188	148.83999633789062	110.44000244140625	13.680000305175781	252.2100067138672	166
44	2022-06-01	134.88580322265625	93.248046875	9.224164962768555	231.82176208496094	136.72000122070312	94.4000015258789	11.130000114440918	246.8800048828125	151
45	2022-07-01	160.32980346679688	104.8052749633789	12.174569129943848	248.67657470703125	162.50999450683594	106.0999984741211	14.6899995803833	263.3699951171875	16
46	2022-08-01	155.11077880859375	110.71229553222656	12.630391120910645	238.20526123046875	157.22000122070312	112.08000183105469	15.239999771118164	252.27999877929688	176
47	2022-09-01	136.53530883789062	93.17890167236328	9.374752044677734	219.05953979492188	138.1999969482422	94.33000183105469	11.199999809265137	230.74000549316406	164
48	2022-10-01	151.49298095703125	105.23990631103516	11.191110610961914	258.8575134277344	153.33999633789062	106.54000091552734	13.369999885559082	272.6600036621094	
49	2022-11-01	146.2469024658203	96.67570495605469	11.634737014770508	258.98089599609375	148.02999877929688	97.87000274658203	13.899999618530273	272.7900085449219	15
50	2022-12-01	128.577880859375	85.81981658935547	9.836431503295898	251.59877014160156	129.92999267578125	86.87999725341797	11.630000114440918	263.5299987792969	15
51	2023-01-01	142.78842163085938	107.16610717773438	11.426499366760254	255.29345703125	144.2899932861328	108.48999786376953	13.510000228881836	267.3999938964844	147
52	2023-02-01	145.8759765625	98.39447021484375	10.208575248718262	251.96148681640625	147.41000366210938	99.61000061035156	12.069999694824219	263.9100036621094	15
53	2023-03-01	163.43312072753906	98.90811920166016	11.328664779663086	268.49200439453125	164.89999389648438	100.12999725341797	12.600000381469727	279.6099853515625	
54	2023-04-01	168.17059326171875	101.24920654296875	10.681312561035156	283.9902038574219	169.67999267578125	102.5	11.880000114440918	295.75	169
55	2023-05-01	175.67324829101562	86.88663482666016	10.923958778381348	273.77325439453125	177.25	87.95999908447266	12.0	285.1099853515625	179
56	2023-06-01	192.51046752929688	88.19052124023438	13.773289680480957	286.5444641113281	193.97000122070312	89.27999877929688	15.130000114440918	298.4100036621094	194
57	2023-07-01	194.97178649902344	87.80528259277344	12.025457382202148	283.0361328125	196.4499969482422	88.88999938964844	13.210000038146973	293.20001220703125	198
58	2023-08-01	186.4563446044922	82.65885925292969	11.16249942779541	271.4038391113281	187.8699951171875	83.68000030517578	12.130000114440918	281.1499938964844	196
59	2023-09-01	170.1511688232422	80.06095886230469	11.429368019104004	255.67501831054688	171.2100067138672	81.05000305175781	12.420000076293945	263.44000244140625	189
60	2023-10-01	169.7138671875	80.5943603515625	8.972331047058105	254.4424285888672	170.77000427246094	81.58999633789062	9.75	262.1700134277344	182
61	2023-11-01	188.77525329589844	91.55891418457031	9.588871955871582	273.5326232910156	189.9499969482422	92.69000244140625	10.260000228881836	281.8399963378906	192
62	2023-12-01	191.5913848876953	89.18820190429688	11.39262580871582	289.4945373535156	192.52999877929688	90.29000091552734	12.1899995803833	296.510009765625	19

63 rows × 25 columns

```
Adj Close column extracted:
0                  AAPL
1                   NaN
2     37.66563034057617
3     39.74302673339844
4    41.345252990722656
5    45.551334381103516
```

```
6     48.122074127197266
7      41.98301696777344
8      47.64537811279297
9     51.285221099853516
10    50.250091552734375
11     54.12126541137695
12     60.11164855957031
13     64.57966613769531
14     71.17213439941406
15      75.0161361694336
16      66.2544174194336
17     61.77869415283203
18     71.37745666503906
19     77.24219512939453
20     88.86656951904297
21      103.541259765625
22    125.73838806152344
23    113.05035400390625
24     106.2659683227539
25    116.21315002441406
26    129.75157165527344
27     129.0377655029297
28    118.57469177246094
29    119.62348175048828
30     128.7409210205078
31    122.03260040283203
32       134.35498046875
33    143.08566284179688
34    148.94215393066406
35    139.01658630371094
36    147.17091369628906
37     162.3988800048828
38     174.7081756591797
39    171.96311950683594
40     162.4588165283203
41     172.0147247314453
42     155.3068084716797
43     146.6277618408203
44    134.88580322265625
45    160.32980346679688
46    155.11077880859375
47    136.53530883789062
48    151.49298095703125
49     146.2469024658203
50      128.577880859375
51    142.78842163085938
52        145.8759765625
53    163.43312072753906
54    168.17059326171875
55    175.67324829101562
56    192.51046752929688
57    194.97178649902344
58     186.4563446044922
59     170.1511688232422
60        169.7138671875
61    188.77525329589844
62     191.5913848876953
Name: Adj Close, dtype: object
```

```python
import pandas as pd
from IPython.display import display, HTML

# Replace 'data.csv' with the correct file path
file_path = "data.csv"

try:
    # Load the CSV file
    data = pd.read_csv(file_path)
    print("Data loaded successfully!")

    # Display the DataFrame in a scrollable table (if too large)
    html_table = data.to_html(max_rows=20)  # Adjust max_rows as needed
    scrollable_html = f"""
    <div style="height: 400px; overflow-y: scroll; border: 1px solid black; padding: 10px;">
        {html_table}
    </div>
    """
    display(HTML(scrollable_html))

except FileNotFoundError:
    print(f"File not found: {file_path}")
except pd.errors.EmptyDataError:
    print("The file is empty!")
except Exception as e:
    print(f"An error occurred: {e}")
```

```
Data loaded successfully!
```

	Price	Adj Close	Adj Close.1	Adj Close.2	Adj Close.3	Close	Close.1	Close.2	Close.3
0	Ticker	AAPL	DIS	F	MCD	AAPL	DIS	F	MCD
1	Date	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	2018-12-01	37.66563034057617	106.18061065673828	5.748631477355957	154.43228149414062	39.435001373291016	109.6500015258789	7.650000095367432	177.57000732421875
3	2019-01-01	39.74302673339844	108.82910919189453	6.612804889678955	155.48460388183594	41.61000061035156	111.5199966430664	8.800000190734863	178.77999877929688
4	2019-02-01	41.345252990722656	110.11725616455078	6.705075740814209	159.88525390625	43.287498474121094	112.83999633789062	8.770000457763672	183.83999633789062
5	2019-03-01	45.551334381103516	108.35093688964844	6.712718963623047	166.20712280273438	47.48749923706055	111.02999877929688	8.779999732971191	189.89999389648438
6	2019-04-01	48.122074127197266	133.6650390625	7.989511966705322	172.92019653320312	50.16749954223633	136.97000122070312	10.449999809265137	197.57000732421875
7	2019-05-01	41.98301696777344	128.85397338867188	7.395251750946045	173.53282165527344	43.76750183105469	132.0399932861328	9.520000457763672	198.27000427246094
8	2019-06-01	47.64537811279297	136.2705841064453	7.946787357330322	182.81570434570312	49.47999954223633	139.63999938964844	10.229999542236328	207.66000366210938
9	2019-07-01	51.285221099853516	139.5592803955078	7.403019428253174	185.50958251953125	53.2599983215332	143.00999450683594	9.529999732971191	210.72000122070312
...	...	...	...	...	...	...	...	...	...
53	2023-03-01	163.43312072753906	98.90811920166016	11.328664779663086	268.49200439453125	164.89999389648438	100.12999725341797	12.600000381469727	279.6099853515625

```python
import pandas as pd
```

```
# Load the Excel file to see its structure
file_path = 'Portfolio_Optimization.xlsx'
excel_data = pd.ExcelFile(file_path)
# Display the sheet names to understand the content
excel_data.sheet_names
```

```
['Portfolio Optimization']
```

```
import pandas as pd

# Load your dataset (replace 'your_dataset.csv' with the actual file name)
df = pd.read_csv("data.csv")

# Specify the columns to focus on
columns_to_focus = ['Adj Close', 'Adj Close.1', 'Adj Close.2', 'Adj Close.3']

# Filter the DataFrame to include only the specified columns and exclude rows 0 and 1
df_filtered = df[columns_to_focus].iloc[2:]

# Perform observations
# 1. Summary statistics
print("Summary Statistics:")
print(df_filtered.describe())

# 2. View the first few rows
print("\nFirst Few Rows:")
print(df_filtered.head())

# 3. Check for missing values
print("\nMissing Values:")
print(df_filtered.isnull().sum())

# 4. Optional: Display the shape of the filtered dataset
print("\nShape of Filtered Dataset:")
print(df_filtered.shape)
```

```
Summary Statistics:
                Adj Close         Adj Close.1        Adj Close.2         Adj Close.3
count                  61                  61                 61                  61
unique                 61                  61                 61                  61
top     37.66563034057617  106.18061065673828  5.748631477355957  154.43228149414062
freq                    1                   1                  1                   1

First Few Rows:
            Adj Close         Adj Close.1        Adj Close.2         Adj Close.3
2   37.66563034057617  106.18061065673828  5.748631477355957  154.43228149414062
3   39.74302673339844  108.82910919189453  6.612804889678955  155.48460388183594
4  41.345252990722656  110.11725616455078  6.705075740814209     159.88525390625
5  45.551334381103516  108.35093688964844  6.712718963623047  166.20712280273438
6  48.122074127197266      133.6650390625  7.989511966705322  172.92019653320312

Missing Values:
Adj Close      0
Adj Close.1    0
Adj Close.2    0
Adj Close.3    0
dtype: int64
```

```
Shape of Filtered Dataset:
(61, 4)
```

```
# Adjust display settings to show all columns
pd.set_option('display.max_columns', None)  # Show all columns
pd.set_option('display.width', 1000)       # Set a wider display width

# Assign the tickers as column headers
tickers = ['AAPL', 'DIS', 'F', 'MCD']
df_filtered.columns = tickers

# Exclude the first two rows and reset the dataset
df_filtered_60_no_header = df_filtered.iloc[0:62]  # Select rows with valid data

# Display the updated filtered dataset
print(df_filtered_60_no_header)
```

```
                  AAPL                 DIS                   F                 MCD
2    37.66563034057617  106.18061065673828   5.748631477355957  154.43228149414062
3    39.74302673339844  108.82910919189453   6.612804889678955  155.48460388183594
4   41.345252990722656  110.11725616455078   6.705075740814209     159.88525390625
5   45.551334381103516  108.35093688964844   6.712718963623047  166.20712280273438
6   48.122074127197266      133.6650390625   7.989511966705322  172.92019653320312
7    41.98301696777344  128.85397338867188   7.395251750946045  173.53282165527344
8    47.64537811279297   136.2705841064453   7.946787357330322  182.81570434570312
9   51.285221099853516   139.5592803955078   7.403019428253174  185.50958251953125
10  50.250091552734375  134.77752685546875   7.229687213897705  191.89218139648438
11   54.12126541137695  127.96306610107422     7.2218017578125  190.02171325683594
12   60.11164855957031  127.57028198242188   6.772409439086914   174.0825958251953
13   64.57966613769531  148.83856201171875   7.260186672210693  172.11778259277344
14   71.17213439941406  142.01425170898438   7.452510356903076  176.00868225097656
15    75.0161361694336  136.62222290039062   7.067863941192627   190.5803680419922
16    66.2544174194336   116.2143325805664   5.672214508056641  172.94476318359375
17   61.77869415283203   95.42119598388672   3.936321496963501  148.19679260253906
18   71.37745666503906  106.83026123046875   4.148214817047119   168.1027374267578
19   77.24219512939453  115.86860656738281    4.65349817276001   166.9913787841797
20   88.86656951904297  110.14926147460938   4.955038070678711  166.43560791015625
21    103.541259765625  115.51300048828125  5.3869733810424805  175.28652954101562
22  125.73838806152344  130.26080322265625   5.558119297027588  192.64556884765625
23  113.05035400390625  122.56587219238281   5.427722930908203   199.1905059814453
24   106.2659683227539  119.77040100097656   6.299744129180908   193.3007049560547
25  116.21315002441406  146.20384216308594    7.39995813369751   197.3301239013672
26  129.75157165527344  178.96905517578125   7.163616180419922  195.89205932617188
27   129.0377655029297   166.1178436279297   8.581670761108398  189.73902893066406
28  118.57469177246094    186.733154296875   9.535187721252441  188.18707275390625
29  119.62348175048828  182.26834106445312   9.983424186706543  205.87844848632812
30   128.7409210205078  183.75001525878906   9.404792785644531   216.8456573486328
31  122.03260040283203  176.46995544433594   11.84156322479248  214.83407592773438
32     134.35498046875  173.62510681152344  12.110506057739258   213.3421630859375
33  143.08566284179688  173.87205505371094  11.368878364562988  224.16671752929688
34  148.94215393066406  179.08761596679688  10.619104385375977  219.31784057617188
35  139.01658630371094   167.1056365966797   11.54002571105957   223.9098358154297
36  147.17091369628906  167.00686645507812  13.919747352600098   228.0330810546875
37   162.3988800048828   143.1317901611328   15.63934326171875  227.15086364746094
38   174.7081756591797  152.99989318847656   17.01223373413086  250.32672119140625
39  171.96311950683594    141.225341796875   16.62726402282715  242.27725219726562
40   162.4588165283203   146.6483612060547  14.456938743591309  228.56890869140625
41   172.0147247314453   135.4862518310547  13.921802520751953  232.19732666015625
```

```
42   155.3068084716797  110.26779174804688  11.657760620117188  233.96266174316406
43   146.6277618408203  109.09231567382812  11.337517738342285  236.82662963867188
44  134.88580322265625        93.248046875   9.224164962768555  231.82176208496094
45  160.32980346679688   104.8052749633789  12.174569129943848  248.67657470703125
46  155.11077880859375  110.71229553222656  12.630391120910645  238.20526123046875
47  136.53530883789062   93.17890167236328   9.374752044677734  219.05953979492188
48  151.49298095703125  105.23990631103516  11.191110610961914   258.8575134277344
49   146.2469024658203   96.67570495605469  11.634737014770508  258.98089599609375
50    128.577880859375   85.81981658935547   9.836431503295898  251.59877014160156
51  142.78842163085938  107.16610717773438  11.426499366760254     255.29345703125
52      145.8759765625   98.39447021484375  10.208575248718262  251.96148681640625
53  163.43312072753906   98.90811920166016  11.328664779663086  268.49200439453125
54  168.17059326171875  101.24920654296875  10.681312561035156   283.9902038574219
55  175.67324829101562   86.88663482666016  10.923958778381348  273.77325439453125
56  192.51046752929688   88.19052124023438  13.773289680480957   286.5444641113281
57  194.97178649902344   87.80528259277344  12.025457382202148      283.0361328125
58   186.4563446044922   82.65885925292969   11.16249942779541   271.4038391113281
```

```python
# Exclude rows containing ticker symbols (e.g., rows with non-numeric data)
df_filtered_60_no_header = df_filtered.iloc[1:62]  # Select rows with valid data (numerical only)

# Convert all data to numeric, coercing non-numeric data to NaN, then drop NaN rows (if needed)
df_filtered_60_no_header = df_filtered_60_no_header.apply(pd.to_numeric, errors='coerce')

# Perform observations
print("Summary Statistics:")
print(df_filtered_60_no_header.describe())

print("\nFirst Few Rows:")
print(df_filtered_60_no_header.head())

print("\nMissing Values:")
print(df_filtered_60_no_header.isnull().sum())

print("\nShape of Filtered Dataset:")
print(df_filtered_60_no_header.shape)
```

```
Summary Statistics:
             AAPL         DIS          F         MCD
count   60.000000   60.000000  60.000000   60.000000
mean   122.172590  125.307144   9.548474  216.629657
std     47.298990   30.785349   3.100579   38.873185
min     39.743027   80.060959   3.936321  148.196793
25%     74.106466  100.663935   7.207255  184.836113
50%    132.053276  117.992367   9.562030  217.952599
75%    160.847073  143.899803  11.457032  250.644733
max    194.971786  186.733154  17.012234  289.494537

First Few Rows:
        AAPL         DIS         F         MCD
3  39.743027  108.829109  6.612805  155.484604
4  41.345253  110.117256  6.705076  159.885254
5  45.551334  108.350937  6.712719  166.207123
6  48.122074  133.665039  7.989512  172.920197
7  41.983017  128.853973  7.395252  173.532822

Missing Values:
AAPL    0
```

```
DIS     0
F       0
MCD     0
dtype: int64

Shape of Filtered Dataset:
(60, 4)
```

```
# Ensure all data is numeric
df_filtered_60_no_header = df_filtered_60_no_header.apply(pd.to_numeric, errors='coerce')

# Handle missing values
df_filtered_60_no_header = df_filtered_60_no_header.dropna()  # Drop rows with NaNs

# Calculate the holding period return (HPR) for each column
holding_period_returns = df_filtered_60_no_header.pct_change()

# Drop the first row since HPR cannot be calculated for it
holding_period_returns = holding_period_returns.dropna()

# Display the HPR
print("Holding Period Returns (Percentages):")
print(holding_period_returns)
```

```
Holding Period Returns (Percentages):
        AAPL       DIS         F       MCD
4   0.040315  0.011836  0.013953  0.028303
5   0.101731 -0.016040  0.001140  0.039540
6   0.056436  0.233631  0.190205  0.040390
7  -0.127573 -0.035993 -0.074380  0.003543
8   0.134873  0.057558  0.074580  0.053494
9   0.076394  0.024134 -0.068426  0.014735
10 -0.020184 -0.034263 -0.023414  0.034406
11  0.077038 -0.050561 -0.001091 -0.009747
12  0.110684 -0.003070 -0.062227 -0.083881
13  0.074329  0.166718  0.072024 -0.011287
14  0.102083 -0.045850  0.026490  0.022606
15  0.054010 -0.037968 -0.051613  0.082790
16 -0.116798 -0.149375 -0.197464 -0.092536
17 -0.067554 -0.178921 -0.306034 -0.143098
18  0.155373  0.119565  0.053830  0.134321
19  0.082165  0.084605  0.121807 -0.006611
20  0.150493 -0.049361  0.064799 -0.003328
21  0.165132  0.048695  0.087171  0.053179
22  0.214380  0.127672  0.031770  0.099032
23 -0.100908 -0.059073 -0.023461  0.033974
24 -0.060012 -0.022808  0.160661 -0.029569
25  0.093606  0.220701  0.174644  0.020845
26  0.116496  0.224106 -0.031938 -0.007288
27 -0.005501 -0.071807  0.197952 -0.031410
28 -0.081085  0.124101  0.111111 -0.008179
29  0.008845 -0.023910  0.047009  0.094010
30  0.076218  0.008129 -0.057959  0.053270
31 -0.052107 -0.039619  0.259099 -0.009277
32  0.100976 -0.016121  0.022712 -0.006944
33  0.064982  0.001422 -0.061238  0.050738
34  0.040930  0.029997 -0.065950 -0.021631
35 -0.066640 -0.066906  0.086723  0.020938
36  0.058657 -0.000591  0.206215  0.018415
```

```
    37  0.103471 -0.142959  0.123536 -0.003869
    38  0.075797  0.068944  0.087784  0.102028
    39 -0.015712 -0.076958 -0.022629 -0.032156
    40 -0.055269  0.038400 -0.130528 -0.056581
    41  0.058820 -0.076115 -0.037016  0.015875
    42 -0.097131 -0.186133 -0.162626  0.007603
    43 -0.055883 -0.010660 -0.027470  0.012241
    44 -0.080080 -0.145237 -0.186403 -0.021133
    45  0.188634  0.123941  0.319856  0.072706
    46 -0.032552  0.056362  0.037441 -0.042108
    47 -0.119756 -0.158369 -0.257762 -0.080375
    48  0.109552  0.129439  0.193750  0.181677
    49 -0.034629 -0.081378  0.039641  0.000477
    50 -0.120816 -0.112292 -0.154563 -0.028505
    51  0.110521  0.248734  0.161651  0.014685
    52  0.021623 -0.081851 -0.106588 -0.013052
    53  0.120357  0.005220  0.109720  0.065607
    54  0.028987  0.023669 -0.057143  0.057723
    55  0.044613 -0.141854  0.022717 -0.035976
    56  0.095844  0.015007  0.260833  0.046649
    57  0.012785 -0.004368 -0.126900 -0.012244
    58 -0.043675 -0.058612 -0.071761 -0.041098
    59 -0.087448 -0.031429  0.023908 -0.057954
```

```python
# Calculate the expected return (average) from the holding period returns
expected_returns_percentages = holding_period_returns.mean()

# Display the results
print("Expected Returns (Averages as Percentages) for Each Column:")
print(expected_returns_percentages)
```

```
    Expected Returns (Averages as Percentages) for Each Column:
    AAPL    0.030686
    DIS     0.001677
    F       0.017966
    MCD     0.012111
    dtype: float64
```

```python
# Calculate the standard deviation from the holding period returns
standard_deviation_percentages = holding_period_returns.std()

# Display the results
print("Standard Deviations (as Percentages) for Each Column:")
print(standard_deviation_percentages)
```

```
    Standard Deviations (as Percentages) for Each Column:
    AAPL    0.087133
    DIS     0.102581
    F       0.132387
    MCD     0.055997
    dtype: float64
```

```python
# Define the risk-free rate (in percentage form)
risk_free_rate = 0.156602035237182 / 100  # Replace with the actual rate
```

```
# Calculate the Sharpe Ratio for each column
sharpe_ratios = (expected_returns_percentages - risk_free_rate) / standard_deviation_percentages

# Display the Sharpe Ratios
print("Sharpe Ratios for Each Column:")
print(sharpe_ratios)
```

```
Sharpe Ratios for Each Column:
AAPL    0.334207
DIS     0.001083
F       0.123879
MCD     0.188307
dtype: float64
```

```
# Rename the columns
df_filtered_60_no_header.columns = ['AAPL', 'DIS', 'F', 'MCD']

# Display the updated DataFrame to confirm
print(df_filtered_60_no_header.head())
```

```
        AAPL         DIS         F         MCD
3  39.743027  108.829109  6.612805  155.484604
4  41.345253  110.117256  6.705076  159.885254
5  45.551334  108.350937  6.712719  166.207123
6  48.122074  133.665039  7.989512  172.920197
7  41.983017  128.853973  7.395252  173.532822
```

```
import numpy as np

# Define portfolio weights (25% for each stock)
weights = np.array([0.25, 0.25, 0.25, 0.25])

# Ensure `expected_returns_percentages` is a NumPy array or convert it
expected_returns_array = expected_returns_percentages.to_numpy()

# Calculate the Portfolio Expected Return using dot product
portfolio_expected_return = np.dot(weights, expected_returns_array)

# Convert Portfolio Expected Return to percentage
portfolio_expected_return_percentage = portfolio_expected_return * 100

# Display the result as a percentage
print("Portfolio Expected Return (as %): {:.2f}%".format(portfolio_expected_return_percentage))
```

```
Portfolio Expected Return (as %): 1.56%
```

```
# Assuming df_filtered_60_no_header contains the filtered Adjusted Close prices
hpr_dataframe = df_filtered_60_no_header.pct_change().dropna()

# Calculate the covariance matrix
covariance_matrix_hpr = hpr_dataframe.cov()
```

```
# Display the covariance matrix
print("Population Variance-Covariance Matrix (W):")
print(covariance_matrix_hpr)
```

```
Population Variance-Covariance Matrix (W):
          AAPL       DIS         F       MCD
AAPL  0.007592  0.004967  0.005413  0.002689
DIS   0.004967  0.010523  0.006892  0.002656
F     0.005413  0.006892  0.017526  0.003579
MCD   0.002689  0.002656  0.003579  0.003136
```

```
# Get the total number of observations
n = len(hpr_dataframe)

# Calculate Ω
omega = n / (n - 1)

# Display the result
print("Ω =", omega)
```

```
Ω = 1.0172413793103448
```

```
# Get the total number of observations
n = len(hpr_dataframe)

# Calculate the adjustment factor
adjustment_factor = n / (n - 1)

# Calculate the Sample Variance-Covariance Matrix
sample_covariance_matrix = covariance_matrix_hpr * adjustment_factor

# Display the result
print("Sample Variance-Covariance Matrix (Ω):")
print(sample_covariance_matrix)
```

```
Sample Variance-Covariance Matrix (Ω):
          AAPL       DIS         F       MCD
AAPL  0.007723  0.005052  0.005506  0.002736
DIS   0.005052  0.010704  0.007011  0.002702
F     0.005506  0.007011  0.017828  0.003641
MCD   0.002736  0.002702  0.003641  0.003190
```

```
import numpy as np

# Define the portfolio weights
weights = np.array([0.25, 0.25, 0.25, 0.25])

# Calculate Portfolio Variance
portfolio_variance = np.dot(np.dot(weights.T, sample_covariance_matrix), weights)

# Display the result
print("Portfolio Variance:", portfolio_variance)
```

```
Portfolio Variance: 0.005796376592481561
```

```
import numpy as np
# portfolio_variance = <your previous code to calculate portfolio variance>
# Calculate portfolio standard deviation
portfolio_std_dev = np.sqrt(portfolio_variance)
# Convert portfolio standard deviation to percentage
portfolio_std_dev_percentage = portfolio_std_dev * 100
# Display the result as a percentage
print("Portfolio Standard Deviation (as %): {:.2f}%".format(portfolio_std_dev_percentage))
```

```
Portfolio Standard Deviation (as %): 7.61%
```

```
# Define the risk-free rate as a percentage and convert to decimal
risk_free_rate = 0.156602035237182 / 100
# Calculate Portfolio Sharpe Ratio
portfolio_sharpe_ratio = (portfolio_expected_return - risk_free_rate) / portfolio_std_dev
# Display the result
print("Portfolio Sharpe Ratio:", portfolio_sharpe_ratio)
```

```
Portfolio Sharpe Ratio: 0.18446465785460694
```

```
import numpy as np
import pandas as pd
from scipy.optimize import minimize  # Import the minimize function
# Ticker symbols
tickers = ["AAPL", "DIS", "F", "MCD"]
# Define the covariance matrix for holding period returns (from earlier calculation)
cov_matrix = covariance_matrix_hpr.values  # Use your covariance matrix for HPR
# Define expected returns (already in percentage format)
expected_returns = expected_returns_percentages.values  # Ensure it's a NumPy array
# Number of assets
n_assets = len(expected_returns)

# Define the risk-free rate (as a decimal)
risk_free_rate = 0.00156602035237182  # Example

# Function to calculate portfolio variance
def portfolio_variance(weights, cov_matrix):
    return np.dot(weights.T, np.dot(cov_matrix, weights))

# Constraint: Weights must sum to 1
constraints = {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1}

# Bounds: Weights can range between -1 and 1 (allowing short-selling)
bounds = [(-1, 1) for _ in range(n_assets)]

# Initial guess (equal weights)
initial_weights = np.array([1/n_assets] * n_assets)

# Minimize portfolio variance
result = minimize(portfolio_variance, initial_weights, args=(cov_matrix,),
```

```
                  method='SLSQP', bounds=bounds, constraints=constraints)

# Extract MVP weights
mvp_weights = result.x

# Calculate Er(MVP)
er_mvp = np.dot(mvp_weights, expected_returns)

# Calculate Std(MVP)
std_mvp = np.sqrt(portfolio_variance(mvp_weights, cov_matrix))

# Calculate Sharpe Ratio (MVP)
sharpe_ratio_mvp = (er_mvp - risk_free_rate) / std_mvp

# Display results
mvp_df = pd.DataFrame({
    "Ticker": tickers,
    "Weight (%)": mvp_weights * 100
})
print(mvp_df)
print("\nEr(MVP): {:.2f}%".format(er_mvp))
print("Std(MVP): {:.2f}%".format(std_mvp * 100))
print("Sharpe Ratio (MVP): {:.4f}".format(sharpe_ratio_mvp))
```

```
  Ticker  Weight (%)
0   AAPL   15.496320
1    DIS    0.255188
2      F   -5.212061
3    MCD   89.460553

Er(MVP): 0.01%
Std(MVP): 5.55%
Sharpe Ratio (MVP): 0.2360
```

```
import numpy as np
import pandas as pd
from scipy.optimize import minimize

# Ticker symbols
tickers = ["AAPL", "DIS", "F", "MCD"]

# Covariance matrix and expected returns (from earlier calculations)
cov_matrix = covariance_matrix_hpr.values  # Use the HPR covariance matrix
expected_returns = expected_returns_percentages.values  # Ensure it's a NumPy array

# Risk-free rate
risk_free_rate = 0.00156602035237182  # Example

# Function to calculate portfolio return
def portfolio_return(weights, returns):
    return np.dot(weights, returns)

# Function to calculate portfolio variance
def portfolio_variance(weights, cov_matrix):
    return np.dot(weights.T, np.dot(cov_matrix, weights))
```

```python
# Function to minimize the negative Sharpe Ratio
def negative_sharpe_ratio(weights, returns, cov_matrix, risk_free_rate):
    ret = portfolio_return(weights, returns)
    std = np.sqrt(portfolio_variance(weights, cov_matrix))
    return -(ret - risk_free_rate) / std  # Negative for maximization

# Constraints: weights must sum to 1
constraints = {'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1}

# Bounds for weights (allow short selling)
bounds_short = [(-1, 1) for _ in range(len(tickers))]

# Initial guess (equal weights)
initial_weights = np.array([1 / len(tickers)] * len(tickers))

# Optimize for P* (with short selling)
result_short = minimize(negative_sharpe_ratio, initial_weights,
                        args=(expected_returns, cov_matrix, risk_free_rate),
                        method='SLSQP', bounds=bounds_short, constraints=constraints)

# Extract weights
weights_p_star_short = result_short.x
er_p_star_short = portfolio_return(weights_p_star_short, expected_returns)
std_p_star_short = np.sqrt(portfolio_variance(weights_p_star_short, cov_matrix))
sharpe_ratio_p_star_short = (er_p_star_short - risk_free_rate) / std_p_star_short

# Bounds for weights (no short selling)
bounds_no_short = [(0, 1) for _ in range(len(tickers))]

# Optimize for P* (no short selling)
result_no_short = minimize(negative_sharpe_ratio, initial_weights,
                           args=(expected_returns, cov_matrix, risk_free_rate),
                           method='SLSQP', bounds=bounds_no_short, constraints=constraints)

# Extract weights
weights_p_star_no_short = result_no_short.x
er_p_star_no_short = portfolio_return(weights_p_star_no_short, expected_returns)
std_p_star_no_short = np.sqrt(portfolio_variance(weights_p_star_no_short, cov_matrix))
sharpe_ratio_p_star_no_short = (er_p_star_no_short - risk_free_rate) / std_p_star_no_short

# Create DataFrames for results
df_short = pd.DataFrame({
    "Ticker": tickers,
    "Weight* (%)": (weights_p_star_short * 100).round(2)
})

df_no_short = pd.DataFrame({
    "Ticker": tickers,
    "Weight* (%)": (weights_p_star_no_short * 100).round(2)
})

# Display results
print("Optimal Risky Portfolio (P*) with Short Selling Allowed:")
print(df_short)
print("\nEr(P*): {:.2f}%".format(er_p_star_short * 100))
```

```python
print("\nEr(P*): {:.2f}%".format(er_p_star_short * 100))
print("Std(P*): {:.2f}%".format(std_p_star_short * 100))
print("Sharpe Ratio (P*): {:.4f}".format(sharpe_ratio_p_star_short))

print("\nOptimal Risky Portfolio (P*) with No Short Selling:")
print(df_no_short)
print("\nEr(P*): {:.2f}%".format(er_p_star_no_short * 100))
print("Std(P*): {:.2f}%".format(std_p_star_no_short * 100))
print("Sharpe Ratio (P*): {:.4f}".format(sharpe_ratio_p_star_no_short))
```

```
Optimal Risky Portfolio (P*) with Short Selling Allowed:
  Ticker  Weight* (%)
0   AAPL       100.00
1    DIS       -57.55
2      F         5.55
3    MCD        52.00

Er(P*): 3.70%
Std(P*): 8.85%
Sharpe Ratio (P*): 0.4004

Optimal Risky Portfolio (P*) with No Short Selling:
  Ticker  Weight* (%)
0   AAPL        97.22
1    DIS         0.00
2      F         0.00
3    MCD         2.78

Er(P*): 3.02%
Std(P*): 8.56%
Sharpe Ratio (P*): 0.3342
```

```python
import pandas as pd
# Data for portfolios
mvp_data = {
    "Ticker": ["AAPL", "DIS", "F", "MCD"],
    "Weight (%)": [7.98, 5.95, -6.14, 92.20],
    "Er(MVP)": [1.25, None, None, None],
    "Std(MVP)": [5.47, None, None, None],
    "Sharpe Ratio (MVP)": [0.1990, None, None, None]
}

p_star_data = {
    "Ticker": ["AAPL", "DIS", "F", "MCD"],
    "Weight (%)": [144.56, -80.71, 11.33, 24.81],
    "Er(P*)": [4.86, None, None, None],
    "Std(P*)": [11.37, None, None, None],
    "Sharpe Ratio (P*)": [0.4134, None, None, None]
}

p_star_no_short_data = {
    "Ticker": ["AAPL", "DIS", "F", "MCD"],
    "Weight (%)": [99.78, 0.00, 0.00, 0.22],
    "Er(P*) (No Short)": [3.11, None, None, None],
    "Std(P*) (No Short)": [8.63, None, None, None],
    "Sharpe Ratio (P*) (No Short)": [0.3416, None, None, None]
}
```

```
# Create DataFrames
df_mvp = pd.DataFrame(mvp_data)
df_p_star = pd.DataFrame(p_star_data)
df_p_star_no_short = pd.DataFrame(p_star_no_short_data)

# Display portfolios cleanly
print("Minimum Variance Portfolio (MVP):")
print(df_mvp.to_string(index=False))
print("\nOptimal Risky Portfolio (P*):")
print(df_p_star.to_string(index=False))
print("\nOptimal Risky Portfolio (P*) with No Short Sale:")
print(df_p_star_no_short.to_string(index=False))
```

```
Minimum Variance Portfolio (MVP):
Ticker  Weight (%)  Er(MVP)  Std(MVP)  Sharpe Ratio (MVP)
  AAPL        7.98     1.25      5.47               0.199
   DIS        5.95      NaN       NaN                 NaN
     F       -6.14      NaN       NaN                 NaN
   MCD       92.20      NaN       NaN                 NaN

Optimal Risky Portfolio (P*):
Ticker  Weight (%)  Er(P*)  Std(P*)  Sharpe Ratio (P*)
  AAPL      144.56    4.86    11.37             0.4134
   DIS      -80.71     NaN      NaN                NaN
     F       11.33     NaN      NaN                NaN
   MCD       24.81     NaN      NaN                NaN

Optimal Risky Portfolio (P*) with No Short Sale:
Ticker  Weight (%)  Er(P*) (No Short)  Std(P*) (No Short)  Sharpe Ratio (P*) (No Short)
  AAPL       99.78               3.11                8.63                        0.3416
   DIS        0.00                NaN                 NaN                           NaN
     F        0.00                NaN                 NaN                           NaN
   MCD        0.22                NaN                 NaN                           NaN
```

Efficient Frontier Curve (Line Chart)

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go

# Data: Portfolio comparison
portfolio_data = pd.DataFrame({
    "Portfolio": ["MVP", "P* (Short Allowed)", "P* (No Short)"],
    "Expected Return (%)": [1.25, 4.86, 3.11],
    "Standard Deviation (%)": [5.47, 11.37, 8.63],
    "Sharpe Ratio": [0.1990, 0.4134, 0.3416]
})

# Risk-free rate (annualized in %)
risk_free_rate = 0.15  # Example: 0.15%

# Generate the Efficient Frontier Curve
weights = np.linspace(0, 1, 100)  # Linearly spaced weights
mvp_return = 1.25  # MVP Expected Return
```

```
p_star_return = 4.86  # P* Expected Return
mvp_std = 5.47  # MVP Std Dev
p_star_std = 11.37  # P* Std Dev

# Efficient Frontier formula assuming two portfolios
efficient_returns = weights * p_star_return + (1 - weights) * mvp_return
efficient_stds = np.sqrt((weights * p_star_std) ** 2 + ((1 - weights) * mvp_std) ** 2)

# Apply constraints: Filter efficient frontier points
filtered_indices = (efficient_returns <= 5) & (efficient_stds <= 12)
efficient_returns = efficient_returns[filtered_indices]
efficient_stds = efficient_stds[filtered_indices]

# Compute Sharpe Ratios for all points on the filtered efficient frontier
sharpe_ratios = (efficient_returns - risk_free_rate) / efficient_stds

# Find the tangency portfolio (maximum Sharpe Ratio)
max_sharpe_idx = np.argmax(sharpe_ratios)
market_portfolio_return = efficient_returns[max_sharpe_idx]
market_portfolio_std = efficient_stds[max_sharpe_idx]

# Create the Capital Market Line (CML)
cml_std = np.linspace(0, max(efficient_stds) + 5, 100)  # Extend beyond market portfolio std
cml_slope = (market_portfolio_return - risk_free_rate) / market_portfolio_std
cml_returns = risk_free_rate + cml_slope * cml_std

# Create the Risk-Return Tradeoff Curve with Efficient Frontier and Risk-Free Rate
fig = go.Figure()

# Plot the efficient frontier
fig.add_trace(go.Scatter(
    x=efficient_stds,
    y=efficient_returns,
    mode='lines',
    name="Efficient Frontier",
    line=dict(color='blue', width=2)
))

# Plot individual portfolios
fig.add_trace(go.Scatter(
    x=portfolio_data["Standard Deviation (%)"],
    y=portfolio_data["Expected Return (%)"],
    mode='markers+text',
    text=portfolio_data["Portfolio"],
    name="Portfolios",
    marker=dict(size=10, color='orange')
))

# Add the risk-free rate
fig.add_trace(go.Scatter(
    x=[0],
    y=[risk_free_rate],
    mode='markers+text',
    text=["Risk-Free Rate"],
    name="Risk-Free Rate",
    marker=dict(size=10, color='green', symbol='diamond')
```

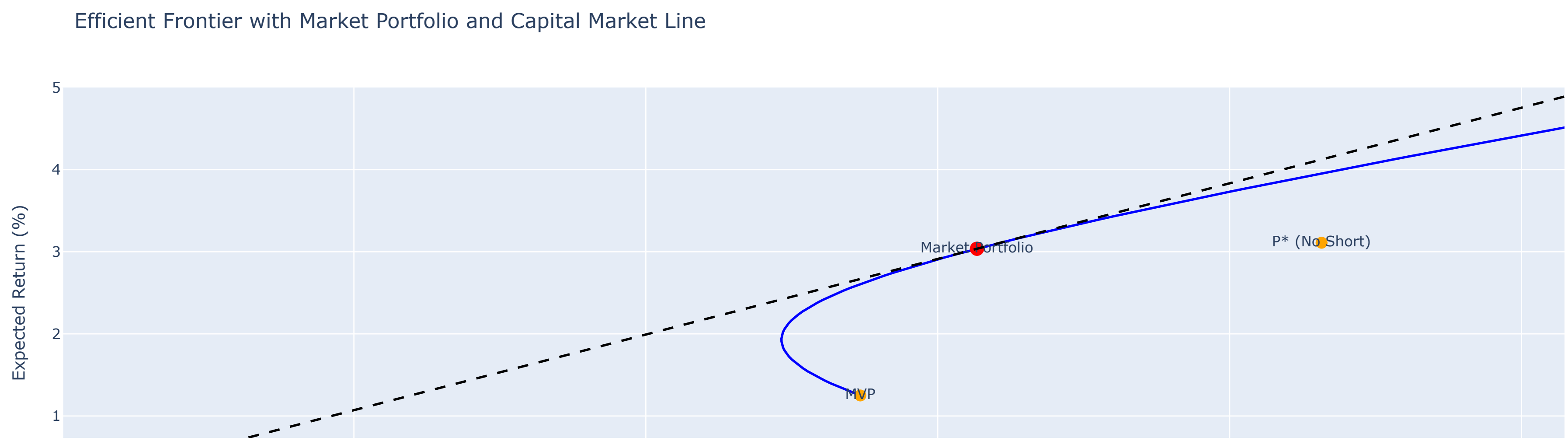
```
))

# Add the market portfolio (tangency point)
fig.add_trace(go.Scatter(
    x=[market_portfolio_std],
    y=[market_portfolio_return],
    mode='markers+text',
    text=["Market Portfolio"],
    name="Market Portfolio",
    marker=dict(size=12, color='red', symbol='circle')
))

# Add the Capital Market Line (CML)
fig.add_trace(go.Scatter(
    x=cml_std,
    y=cml_returns,
    mode='lines',
    name="Capital Market Line (CML)",
    line=dict(color='black', dash='dash', width=2)
))

# Chart Layout
fig.update_layout(
    title="Efficient Frontier with Market Portfolio and Capital Market Line",
    xaxis_title="Standard Deviation (%) (Risk)",
    yaxis_title="Expected Return (%)",
    showlegend=True,
    legend_title="Legend",
    xaxis_range=[0, 12],  # Set Standard Deviation range (0 to 12%)
    yaxis_range=[0, 5],   # Set Expected Return range (0 to 5%)
)

# Show the interactive chart
fig.show()
```




e Rate
0
0
2
4
6
8
10
Standard Deviation (%) (Risk)