Quantitative Momentum Strategy

-Chayan Roy

"Momentum investing" means investing in the stocks that have increased in price the most.

For this project, we're going to build an investing strategy that selects the 50 stocks with the highest price momentum. From there, we will calculate recommended trades for an equal-weight portfolio of these 50 stocks.

Library Imports

First we will import the open-source software libraries that we'll be using in this project.

```
import numpy as np #The Numpy numerical computing library
import pandas as pd #The Pandas data science library
import requests #The requests library for HTTP requests in Python
import xlsxwriter #The XlsxWriter libarary for
import math #The Python math module
from scipy import stats #The SciPy stats module
```

Importing Our List of Stocks

Here we'll import our list of stocks and API token for IEX cloud platform. API token is saved as variable in secrets.py file. During version controll this file is not uploaded making your token secrect and secure.

```
stocks = pd.read_csv('sp_500_stocks.csv')
from secrets import IEX_CLOUD_API_TOKEN
```

This is a sample API Call

Here we are using sandbox version of iex cloud. This is a sample api call where details of one stock, APPLE is loaded.

```
symbol = 'AAPL'
api_url = f'https://sandbox.iexapis.com/stable/stock/{symbol}/stats?token={IEX_CLOUD_API_TOKEN}'
data = requests.get(api_url).json()
data
```

```
{'week52change': 1.271858,
 'week52high': 462.02,
 'week52low': 206.85,
 'marketcap': 1937104274094,
 'employees': 137265,
 'day200MovingAvg': 309.44,
 'day50MovingAvg': 390.09,
 'float': 4440629054,
 'avg10Volume': 54435185.2,
 'avg30Volume': 39067154.1,
 'ttmEPS': 13.7084,
 'ttmDividendRate': 3.22,
 'companyName': 'Apple, Inc.',
 'sharesOutstanding': 4331609946,
 'maxChangePercent': 452.5766,
 'year5ChangePercent': 3.0546,
 'year2ChangePercent': 1.1867,
 'year1ChangePercent': 1.186376,
 'ytdChangePercent': 0.512578,
 'month6ChangePercent': 0.407457,
 'month3ChangePercent': 0.485051,
 'month1ChangePercent': 0.19254,
 'day30ChangePercent': 0.253108,
 'day5ChangePercent': -0.008155,
 'nextDividendDate': '2020-08-16',
 'dividendYield': 0.007235663525925464,
 'nextEarningsDate': '2020-10-17',
 'exDividendDate': '2020-08-06',
 'peRatio': 34.17,
 'beta': 1.15885673879414}
```


```
data['year1ChangePercent']
```

```
1.186376
```

Executing A Batch API Call & Building Our DataFrame in Pandas

Now we execute several batch API calls and add the information we need to our DataFrame.

It contains a function called `chunks` that we can use to divide our list of securities into groups of 100 because using batch api call we can get details of atmost 100 stocks at a time.

```
# Function sourced from
# https://stackoverflow.com/questions/312443/how-do-you-split-a-list-into-evenly-sized-chunks
def chunks(lst, n):
    """Yield successive n-sized chunks from lst."""
    for i in range(0, len(lst), n):
        yield lst[i:i + n]

symbol_groups = list(chunks(stocks['Ticker'], 100))
symbol_strings = []
for i in range(0, len(symbol_groups)):
    symbol_strings.append(','.join(symbol_groups[i]))
#     print(symbol_strings[i])

my_columns = ['Ticker', 'Price', 'One-Year Price Return', 'Number of Shares to Buy']
```

Now we need to create a blank DataFrame and add our data to the data frame one-by-one.

```
final_dataframe = pd.DataFrame(columns = my_columns)

for symbol_string in symbol_strings:
#     print(symbol_strings)
    batch_api_call_url = f'https://sandbox.iexapis.com/stable/stock/market/batch/?types=stats,quote&symbols={symbol_string}&token={I
    data = requests.get(batch_api_call_url).json()
    for symbol in symbol_string.split(','):
        final_dataframe = final_dataframe.append(
                                        pd.Series([symbol,
                                                   data[symbol]['quote']['latestPrice'],
                                                   data[symbol]['stats']['year1ChangePercent'],
                                                   'N/A'
                                                   ],
                                                  index = my_columns),
                                        ignore_index = True)


final_dataframe
```

	Ticker	Price	One-Year Price Return	Number of Shares to Buy
0	A	101.50	0.452986	N/A
1	AAL	13.65	-0.527621	N/A
2	AAP	157.72	0.088479	N/A
3	AAPL	453.87	1.172528	N/A
4	ABBV	95.71	0.476493	N/A
...	...	...	...	...
500	YUM	93.50	-0.208066	N/A
501	ZBH	138.91	0.003031	N/A
502	ZBRA	287.51	0.369427	N/A
503	ZION	35.73	-0.162236	N/A
504	ZTS	165.46	0.288770	N/A

505 rows × 4 columns

Removing Low-Momentum Stocks

The investment strategy that we're building seeks to identify the 50 highest-momentum stocks in the S&P 500.

Because of this, the next thing we need to do is remove all the stocks in our DataFrame that fall below this momentum threshold. We'll sort the DataFrame by the stocks' one-year price return, and drop all stocks outside the top 50.

```
final_dataframe.sort_values('One-Year Price Return', ascending = False, inplace = True)
final_dataframe = final_dataframe[:51]
final_dataframe.reset_index(drop = True, inplace = True)
final_dataframe
```

	Ticker	Price	One-Year Price Return	Number of Shares to Buy
0	NVDA	458.14	2.015964	N/A
1	DXCM	441.29	1.721219	N/A
2	AMD	85.46	1.632817	N/A
3	CARR	29.94	1.489900	N/A
4	AAPL	453.87	1.172528	N/A
5	REGN	624.50	1.031728	N/A
6	SWKS	153.26	0.921570	N/A
7	WST	283.20	0.921000	N/A
8	LRCX	393.68	0.892740	N/A
9	QRVO	132.38	0.858187	N/A
10	PYPL	196.30	0.829338	N/A
11	AMZN	3192.88	0.759898	N/A
12	ATVI	83.82	0.710587	N/A
13	ALGN	313.73	0.709093	N/A
14	NEM	63.72	0.698936	N/A

	Ticker	Price	One-Year Price Return	Number of Shares to Buy
15	ODFL	196.24	0.690263	N/A
16	ROL	55.90	0.676072	N/A
17	NOW	439.05	0.656782	N/A
18	LOW	159.19	0.646186	N/A
19	DPZ	405.67	0.644884	N/A
20	FBHS	85.93	0.635603	N/A
21	FAST	49.98	0.624303	N/A
22	TGT	136.10	0.616877	N/A
23	QCOM	119.34	0.609546	N/A
24	URI	181.40	0.590406	N/A
25	CHTR	602.51	0.586074	N/A
26	MSCI	369.93	0.583839	N/A
27	VAR	174.79	0.569642	N/A
28	ROK	247.30	0.560821	N/A
29	BIO	517.62	0.558503	N/A
30	KSU	193.03	0.548382	N/A
31	KLAC	216.00	0.547432	N/A
32	ADBE	449.01	0.540159	N/A
33	ABMD	307.07	0.537076	N/A
34	CDNS	111.60	0.532542	N/A
35	NFLX	486.12	0.531236	N/A
36	ADSK	240.90	0.526803	N/A
37	FTNT	132.50	0.522105	N/A
38	MSFT	215.73	0.518900	N/A
39	EA	145.30	0.517087	N/A
40	TMO	422.95	0.513970	N/A
41	KR	36.29	0.502626	N/A
42	DHI	73.64	0.499570	N/A
43	CTXS	141.53	0.496854	N/A
44	LEN	77.72	0.495581	N/A
45	MAS	61.01	0.493611	N/A
46	TMUS	119.48	0.490167	N/A
47	BBY	107.33	0.486775	N/A
48	VRTX	280.26	0.486553	N/A
49	PWR	50.22	0.482556	N/A
50	OTIS	67.90	0.481000	N/A

Calculating the Number of Shares to Buy

We now need to calculate the number of shares we need to buy. The one change we're going to make is wrapping this functionality inside a function, since we'll be using it again later in this Jupyter Notebook.

```
def portfolio_input():
    global portfolio_size
    portfolio_size = input("Enter the value of your portfolio:")


    try:
        val = float(portfolio_size)
    except ValueError:
        print("That's not a number! \n Try again:")
        portfolio_size = input("Enter the value of your portfolio:")

portfolio_input()
print(portfolio_size)
```

```
Enter the value of your portfolio:1000000
1000000
```

```
position_size = float(portfolio_size) / len(final_dataframe.index)
for i in range(0, len(final_dataframe['Ticker'])):
    final_dataframe.loc[i, 'Number of Shares to Buy'] = math.floor(position_size / final_dataframe['Price'][i])
final_dataframe
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/pandas/core/indexing.py:494: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-
  self.obj[item] = s
```

	Ticker	Price	One-Year Price Return	Number of Shares to Buy
0	NVDA	458.14	2.015964	42
1	DXCM	441.29	1.721219	44
2	AMD	85.46	1.632817	229
3	CARR	29.94	1.489900	654
4	AAPL	453.87	1.172528	43
5	REGN	624.50	1.031728	31
6	SWKS	153.26	0.921570	127
7	WST	283.20	0.921000	69
8	LRCX	393.68	0.892740	49
9	QRVO	132.38	0.858187	148
10	PYPL	196.30	0.829338	99
11	AMZN	3192.88	0.759898	6
12	ATVI	83.82	0.710587	233
13	ALGN	313.73	0.709093	62
14	NEM	63.72	0.698936	307
15	ODFL	196.24	0.690263	99
16	ROL	55.90	0.676072	350
17	NOW	439.05	0.656782	44
18	LOW	159.19	0.646186	123
19	DPZ	405.67	0.644884	48
20	FBHS	85.93	0.635603	228
21	FAST	49.98	0.624303	392
22	TGT	136.10	0.616877	144
23	QCOM	119.34	0.609546	164
24	URI	181.40	0.590406	108
25	CHTR	602.51	0.586074	32
26	MSCI	369.93	0.583839	53
27	VAR	174.79	0.569642	112
28	ROK	247.30	0.560821	79
29	BIO	517.62	0.558503	37
30	KSU	193.03	0.548382	101
31	KLAC	216.00	0.547432	90
32	ADBE	449.01	0.540159	43
33	ABMD	307.07	0.537076	63
34	CDNS	111.60	0.532542	175
35	NFLX	486.12	0.531236	40
36	ADSK	240.90	0.526803	81
37	FTNT	132.50	0.522105	147
38	MSFT	215.73	0.518900	90
39	EA	145.30	0.517087	134
40	TMO	422.95	0.513970	46
41	KR	36.29	0.502626	540
42	DHI	73.64	0.499570	266
43	CTXS	141.53	0.496854	138
44	LEN	77.72	0.495581	252
45	MAS	61.01	0.493611	321
46	TMUS	119.48	0.490167	164

	Ticker	Price	One-Year Price Return	Number of Shares to Buy
47	BBY	107.33	0.486775	182
48	VRTX	280.26	0.486553	69
49	PWR	50.22	0.482556	390
50	OTIS	67.90	0.481000	288

Building a Better (and More Realistic) Momentum Strategy

Real-world quantitative investment firms differentiate between "high quality" and "low quality" momentum stocks:

- High-quality momentum stocks show "slow and steady" outperformance over long periods of time
- Low-quality momentum stocks might not show any momentum for a long time, and then surge upwards.

To identify high-quality momentum, we're going to build a strategy that selects stocks from the highest percentiles of:

- 1-month price returns
- 3-month price returns
- 6-month price returns
- 1-year price returns

We have used the abbreviation `hqm` often. It stands for `high-quality momentum`.

```
hqm_columns = [
                'Ticker',
                'Price',
                'Number of Shares to Buy',
                'One-Year Price Return',
                'One-Year Return Percentile',
                'Six-Month Price Return',
                'Six-Month Return Percentile',
                'Three-Month Price Return',
                'Three-Month Return Percentile',
                'One-Month Price Return',
                'One-Month Return Percentile',
                'HQM Score'
                ]

hqm_dataframe = pd.DataFrame(columns = hqm_columns)

for symbol_string in symbol_strings:
#     print(symbol_strings)
    batch_api_call_url = f'https://sandbox.iexapis.com/stable/stock/market/batch/?types=stats,quote&symbols={symbol_string}&token={I
    data = requests.get(batch_api_call_url).json()
    for symbol in symbol_string.split(','):
        hqm_dataframe = hqm_dataframe.append(
                                        pd.Series([symbol,
                                                   data[symbol]['quote']['latestPrice'],
                                                   'N/A',
                                                   data[symbol]['stats']['year1ChangePercent'],
                                                   'N/A',
                                                   data[symbol]['stats']['month6ChangePercent'],
                                                   'N/A',
                                                   data[symbol]['stats']['month3ChangePercent'],
                                                   'N/A',
                                                   data[symbol]['stats']['month1ChangePercent'],
                                                   'N/A',
                                                   'N/A'
                                                   ],
                                                  index = hqm_columns),
                                        ignore_index = True)

hqm_dataframe.columns
```

```
Index(['Ticker', 'Price', 'Number of Shares to Buy', 'One-Year Price Return',
       'One-Year Return Percentile', 'Six-Month Price Return',
       'Six-Month Return Percentile', 'Three-Month Price Return',
       'Three-Month Return Percentile', 'One-Month Price Return',
       'One-Month Return Percentile', 'HQM Score'],
      dtype='object')
```

Calculating Momentum Percentiles

We need to calculate percentile scores for the following metrics for every stock:

- One-Year Price Return
- Six-Month Price Return
- Three-Month Price Return
- One-Month Price Return

```
time_periods = [
                'One-Year',
                'Six-Month',
                'Three-Month',
                'One-Month'
                ]

for row in hqm_dataframe.index:
    for time_period in time_periods:
        hqm_dataframe.loc[row, f'{time_period} Return Percentile'] = stats.percentileofscore(hqm_dataframe[f'{time_period} Price Ret

# Print each percentile score to make sure it was calculated properly
for time_period in time_periods:
    print(hqm_dataframe[f'{time_period} Return Percentile'])

#Print the entire DataFrame
hqm_dataframe
```

```
0       0.885149
1      0.0237624
2       0.578218
3       0.992079
4        0.89703
         ...
500     0.211881
501     0.457426
502     0.843564
503     0.255446
504     0.772277
Name: One-Year Return Percentile, Length: 505, dtype: object
0       0.837624
1      0.0158416
2       0.839604
3       0.968317
4       0.629703
         ...
500     0.405941
501      0.39604
502     0.906931
503     0.227723
504     0.776238
Name: Six-Month Return Percentile, Length: 505, dtype: object
0      0.473267
1      0.908911
2      0.643564
3      0.887129
4       0.19802
         ...
500    0.374257
501    0.544554
502    0.611881
503     0.70297
504    0.665347
Name: Three-Month Return Percentile, Length: 505, dtype: object
0       0.530693
1       0.827723
2       0.742574
3       0.879208
4      0.0693069
         ...
500     0.370297
501     0.762376
502     0.641584
503     0.312871
504     0.792079
Name: One-Month Return Percentile, Length: 505, dtype: object
```

	Ticker	Price	Number of Shares to Buy	One-Year Price Return	One-Year Return Percentile	Six-Month Price Return	Six-Month Return Percentile	Three-Month Price Return	Three-Month Return Percentile	One-Month Price Return	One-Month Return Percentile	HQM Score
0	A	98.19	N/A	0.444090	0.885149	0.147456	0.837624	0.221461	0.473267	0.093820	0.530693	N/A
1	AAL	13.89	N/A	-0.526494	0.0237624	-0.564540	0.0158416	0.509431	0.908911	0.172430	0.827723	N/A
2	AAP	161.13	N/A	0.088066	0.578218	0.148378	0.839604	0.295700	0.643564	0.144608	0.742574	N/A
3	AAPL	467.65	N/A	1.171724	0.992079	0.401695	0.968317	0.474900	0.887129	0.189840	0.879208	N/A
4	ABBV	97.29	N/A	0.478770	0.89703	0.001711	0.629703	0.077880	0.19802	-0.024533	0.0693069	N/A
...	...	...	...	...	...	...	...	...	...	...	...	...
500	YUM	93.51	N/A	-0.214524	0.211881	-0.116757	0.405941	0.161550	0.374257	0.068180	0.370297	N/A
501	ZBH	140.16	N/A	0.003007	0.457426	-0.127491	0.39604	0.250906	0.544554	0.151414	0.762376	N/A
502	ZBRA	285.97	N/A	0.373952	0.843564	0.223856	0.906931	0.283668	0.611881	0.115379	0.641584	N/A
503	ZION	34.95	N/A	-0.161814	0.255446	-0.255398	0.227723	0.328190	0.70297	0.055265	0.312871	N/A
504	ZTS	163.66	N/A	0.290969	0.772277	0.102747	0.776238	0.307237	0.665347	0.157549	0.792079	N/A

505 rows × 12 columns

Calculating the HQM Score

We'll now calculate our `HQM Score`, which we'll use to filter for stocks in this investing strategy.

The `HQM Score` will be the arithmetic mean of the 4 momentum percentile scores that we calculated in the last section.

To calculate arithmetic mean, we will use the `mean` function from Python's built-in `statistics` module.

```
from statistics import mean

for row in hqm_dataframe.index:
    momentum_percentiles = []
    for time_period in time_periods:
        momentum_percentiles.append(hqm_dataframe.loc[row, f'{time_period} Return Percentile'])
    hqm_dataframe.loc[row, 'HQM Score'] = mean(momentum_percentiles)
```

Selecting the 50 Best Momentum Stocks

We get the top 50 best performing momentum stock by sorting the DataFrame on the `HQM Score` column in desc order and drop all but top 50 rows.

```
hqm_dataframe.sort_values(by = 'HQM Score', ascending = False)
hqm_dataframe = hqm_dataframe[:51]
```

Calculating the Number of Shares to Buy

We'll use the `portfolio_input` function to accept our portfolio size.

```
portfolio_input()
```

```
Enter the value of your portfolio:1000000
```

```
position_size = float(portfolio_size) / len(hqm_dataframe.index)
for i in range(0, len(hqm_dataframe['Ticker'])-1):
    hqm_dataframe.loc[i, 'Number of Shares to Buy'] = math.floor(position_size / hqm_dataframe['Price'][i])
hqm_dataframe
```

	Ticker	Price	Number of Shares to Buy	One-Year Price Return	One-Year Return Percentile	Six-Month Price Return	Six-Month Return Percentile	Three-Month Price Return	Three-Month Return Percentile	One-Month Price Return	One-Month Return Percentile	HQM Score
0	A	98.19	199	0.444090	0.885149	0.147456	0.837624	0.221461	0.473267	0.093820	0.530693	0.681683
1	AAL	13.89	1411	-0.526494	0.0237624	-0.564540	0.0158416	0.509431	0.908911	0.172430	0.827723	0.444059
2	AAP	161.13	121	0.088066	0.578218	0.148378	0.839604	0.295700	0.643564	0.144608	0.742574	0.70099
3	AAPL	467.65	41	1.171724	0.992079	0.401695	0.968317	0.474900	0.887129	0.189840	0.879208	0.931683
4	ABBV	97.29	201	0.478770	0.89703	0.001711	0.629703	0.077880	0.19802	-0.024533	0.0693069	0.448515
5	ABC	104.97	186	0.163705	0.651485	0.100112	0.774257	0.241867	0.522772	0.066990	0.364356	0.578218
6	ABMD	305.78	64	0.546138	0.942574	0.874783	0.99802	0.645795	0.954455	0.145153	0.746535	0.910396
7	ABT	102.94	190	0.161073	0.647525	0.139130	0.823762	0.093917	0.233663	0.081815	0.457426	0.540594
8	ACN	237.88	82	0.197484	0.69505	0.085519	0.762376	0.279570	0.605941	0.067045	0.366337	0.607426
9	ADBE	453.91	43	0.532839	0.934653	0.196705	0.879208	0.252382	0.550495	0.006829	0.150495	0.628713
10	ADI	120.52	162	0.056525	0.522772	0.002920	0.631683	0.149173	0.360396	0.017244	0.174257	0.422277
11	ADM	46.62	420	0.178916	0.679208	-0.018869	0.580198	0.328413	0.70495	0.123837	0.687129	0.662871
12	ADP	139.82	140	-0.174896	0.243564	-0.227657	0.265347	0.037317	0.134653	-0.042030	0.0435644	0.171782
13	ADSK	234.20	83	0.528696	0.928713	0.116990	0.794059	0.312257	0.677228	-0.001259	0.122772	0.630693
14	AEE	85.23	230	0.073499	0.556436	-0.044706	0.538614	0.202938	0.447525	0.086675	0.483168	0.506436
15	AEP	87.88	223	-0.069760	0.368317	-0.186159	0.326733	0.084255	0.217822	-0.007456	0.106931	0.25495
16	AES	17.54	1117	0.141966	0.635644	-0.177281	0.336634	0.510037	0.910891	0.186968	0.873267	0.689109
17	AFL	39.27	499	-0.298263	0.116832	-0.283044	0.190099	0.178662	0.4	0.082055	0.459406	0.291584

	Ticker	Price	Number of Shares to Buy	One-Year Price Return	One-Year Return Percentile	Six-Month Price Return	Six-Month Return Percentile	Three-Month Price Return	Three-Month Return Percentile	One-Month Price Return	One-Month Return Percentile	HQM Score
18	AIG	31.76	617	-0.468456	0.0356436	-0.398200	0.0673267	0.242767	0.528713	0.042150	0.271287	0.225743
19	AIV	39.00	502	-0.260956	0.168317	-0.323003	0.142574	0.096556	0.243564	0.004109	0.140594	0.173762
20	AIZ	129.93	150	0.016575	0.475248	-0.116835	0.40396	0.418900	0.839604	0.254327	0.948515	0.666832
21	AJG	111.00	176	0.195680	0.689109	-0.011750	0.6	0.253483	0.554455	0.092902	0.520792	0.591089
22	AKAM	110.00	178	0.206310	0.70495	0.075705	0.752475	0.125834	0.312871	-0.027015	0.0594059	0.457426
23	ALB	94.85	206	0.339750	0.823762	0.036265	0.691089	0.528245	0.918812	0.120477	0.673267	0.776733
24	ALGN	309.25	63	0.692338	0.970297	0.131757	0.815842	0.554489	0.928713	0.143933	0.738614	0.863366
25	ALK	38.63	507	-0.387440	0.0633663	-0.438559	0.0534653	0.519770	0.916832	0.103833	0.576238	0.402475
26	ALL	99.53	197	-0.079870	0.354455	-0.230788	0.261386	0.045750	0.146535	0.088925	0.49703	0.314851
27	ALLE	105.13	186	0.071934	0.542574	-0.246470	0.241584	0.112362	0.277228	0.024073	0.20396	0.316337
28	ALXN	107.60	182	-0.079570	0.356436	-0.006381	0.60396	0.003280	0.0792079	-0.040910	0.0455446	0.271287
29	AMAT	68.10	287	0.389577	0.855446	-0.013556	0.594059	0.301391	0.651485	0.082413	0.461386	0.640594
30	AMCR	11.52	1702	0.073093	0.552475	0.105582	0.778218	0.238793	0.510891	0.077579	0.433663	0.568812
31	AMD	85.23	230	1.590065	0.99604	0.532176	0.984158	0.597011	0.948515	0.560304	0.99802	0.981683
32	AME	104.38	187	0.177719	0.677228	-0.002080	0.615842	0.305470	0.659406	0.147190	0.752475	0.676238
33	AMGN	253.49	77	0.176726	0.671287	0.086140	0.764356	0.020895	0.112871	-0.034769	0.0534653	0.400495
34	AMP	166.96	117	0.254633	0.736634	-0.101617	0.429703	0.385995	0.80198	0.089628	0.50099	0.617327
35	AMT	252.70	77	0.137500	0.631683	-0.022213	0.574257	0.091099	0.231683	-0.018941	0.0871287	0.381188
36	AMZN	3267.82	6	0.753428	0.978218	0.471720	0.974257	0.339324	0.728713	0.018781	0.178218	0.714851
37	ANET	225.98	86	-0.065031	0.374257	-0.093866	0.449505	0.011798	0.0910891	0.024541	0.207921	0.280693
38	ANSS	321.00	61	0.451591	0.887129	0.058949	0.722772	0.229415	0.487129	0.044242	0.283168	0.59505
39	ANTM	293.84	66	-0.013031	0.439604	-0.053364	0.514851	0.058803	0.164356	0.104273	0.582178	0.425248
40	AON	196.84	99	0.006102	0.463366	-0.183042	0.332673	0.010224	0.0891089	-0.026247	0.0633663	0.237129
41	AOS	50.31	389	0.101690	0.59802	0.141115	0.825743	0.260257	0.564356	0.032405	0.231683	0.55495
42	APA	16.40	1195	-0.291451	0.124752	-0.441406	0.0514851	0.485613	0.893069	0.260429	0.952475	0.505446
43	APD	295.10	66	0.242925	0.728713	0.124499	0.807921	0.273219	0.592079	0.040389	0.265347	0.598515
44	APH	110.83	176	0.255236	0.738614	0.053364	0.714851	0.342563	0.740594	0.168114	0.819802	0.753465
45	APTV	91.40	214	0.072118	0.544554	-0.011731	0.60198	0.490744	0.89505	0.172564	0.829703	0.717822
46	ARE	174.20	112	0.208731	0.706931	-0.001613	0.619802	0.214849	0.465347	0.078630	0.441584	0.558416
47	ATO	108.98	179	-0.039357	0.415842	-0.125705	0.4	0.089492	0.229703	0.053203	0.306931	0.338119
48	ATVI	82.88	236	0.706749	0.976238	0.310780	0.950495	0.110910	0.273267	0.048812	0.289109	0.622277
49	AVB	156.62	125	-0.263600	0.162376	-0.336301	0.120792	0.012586	0.0930693	0.005162	0.142574	0.129703
50	AVGO	336.73	N/A	0.177460	0.675248	0.030386	0.679208	0.278870	0.60198	0.074210	0.405941	0.590594

Formatting Our Excel Output

We will be using the XlsxWriter library for Python to create nicely-formatted Excel files.

```
writer = pd.ExcelWriter('momentum_strategy.xlsx', engine='xlsxwriter')
hqm_dataframe.to_excel(writer, sheet_name='Momentum Strategy', index = False)
```

Creating the Formats We'll Need For Our .xlsx File

```
background_color = '#0a0a23'
font_color = '#ffffff'

string_template = writer.book.add_format(
        {
            'font_color': font_color,
            'bg_color': background_color,
            'border': 1
        }
    )

dollar_template = writer.book.add_format(
        {
            'num_format':'$0.00',
            'font_color': font_color,
            'bg_color': background_color,
            'border': 1
        }
    )

integer_template = writer.book.add_format(
        {
            'num_format':'0',
            'font_color': font_color,
            'bg_color': background_color,
            'border': 1
        }
    )

percent_template = writer.book.add_format(
        {
            'num_format':'0.0%',
            'font_color': font_color,
            'bg_color': background_color,
            'border': 1
        }
    )
```

```
column_formats = {
                    'A': ['Ticker', string_template],
                    'B': ['Price', dollar_template],
                    'C': ['Number of Shares to Buy', integer_template],
                    'D': ['One-Year Price Return', percent_template],
                    'E': ['One-Year Return Percentile', percent_template],
                    'F': ['Six-Month Price Return', percent_template],
                    'G': ['Six-Month Return Percentile', percent_template],
                    'H': ['Three-Month Price Return', percent_template],
                    'I': ['Three-Month Return Percentile', percent_template],
                    'J': ['One-Month Price Return', percent_template],
                    'K': ['One-Month Return Percentile', percent_template],
                    'L': ['HQM Score', integer_template]
                    }

for column in column_formats.keys():
    writer.sheets['Momentum Strategy'].set_column(f'{column}:{column}', 20, column_formats[column][1])
    writer.sheets['Momentum Strategy'].write(f'{column}1', column_formats[column][0], string_template)
```

Saving Our Excel Output

```
writer.save()
```