Assignment 1:

Installation

1. Open a command prompt in the folder, and type ‘pip install –r requirements.txt’ to make sure that all the dependencies are installed

2. To run the file, on a command prompt type ‘python a1.py’

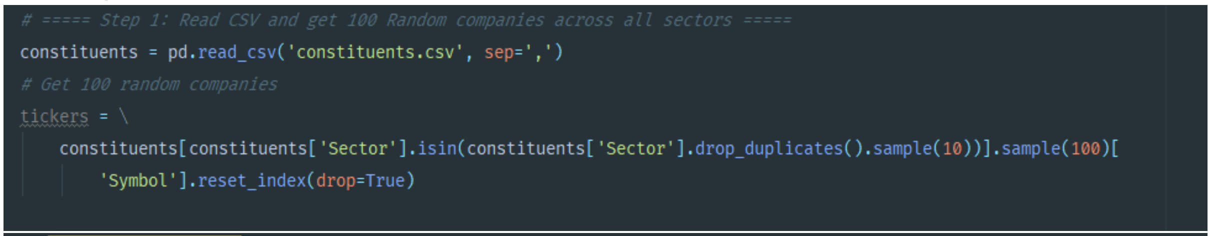
3. Please note: This script makes use of fix_yahoo_finance, which may bug out during execution and fetch blank dataframes. Please re-run the program

Explanation

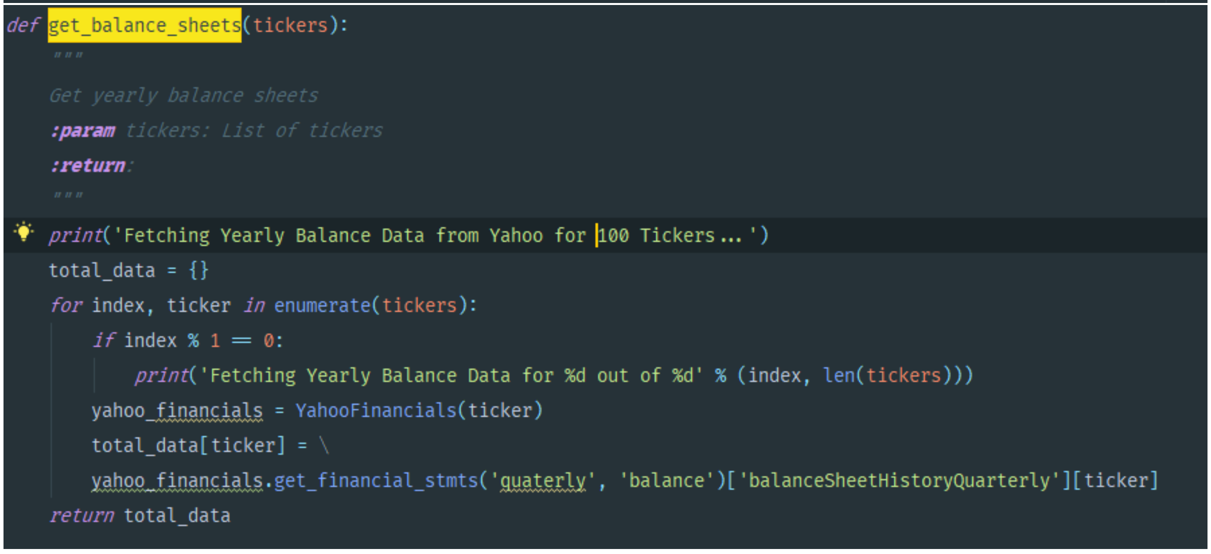
NOTE: I have commented out the parts that connect to yahoo finance and instead use a pickle file which I saved upon fetching the data that I get from connecting to yahoo financial. This helps save time as the process of fetching the data is long. One can always uncomment the function call to verify whether everything is working fine. I’ve kept it that way to save time

NOTE2: For Book to Market, I only took 10 stocks as it takes quite some time for generating B2M

1. **Select a universe of 100 stocks spread across different industry verticals – information technology, utilities, banking and financial services, midcaps, large caps etc**
   Constituents CSV was downloaded from Morningstar. This was then filtered according to sectors, and random 100 stocks were chosen from the 11 sectors. This is depicted by the following code:



```python
# ===== Step 1: Read CSV and get 100 Random companies across all sectors =====
constituents = pd.read_csv('constituents.csv', sep=',')
# Get 100 random companies
tickers = \
    constituents[constituents['Sector'].isin(constituents['Sector'].drop_duplicates().sample(10))].sample(100)[
        'Symbol'].reset_index(drop=True)
```

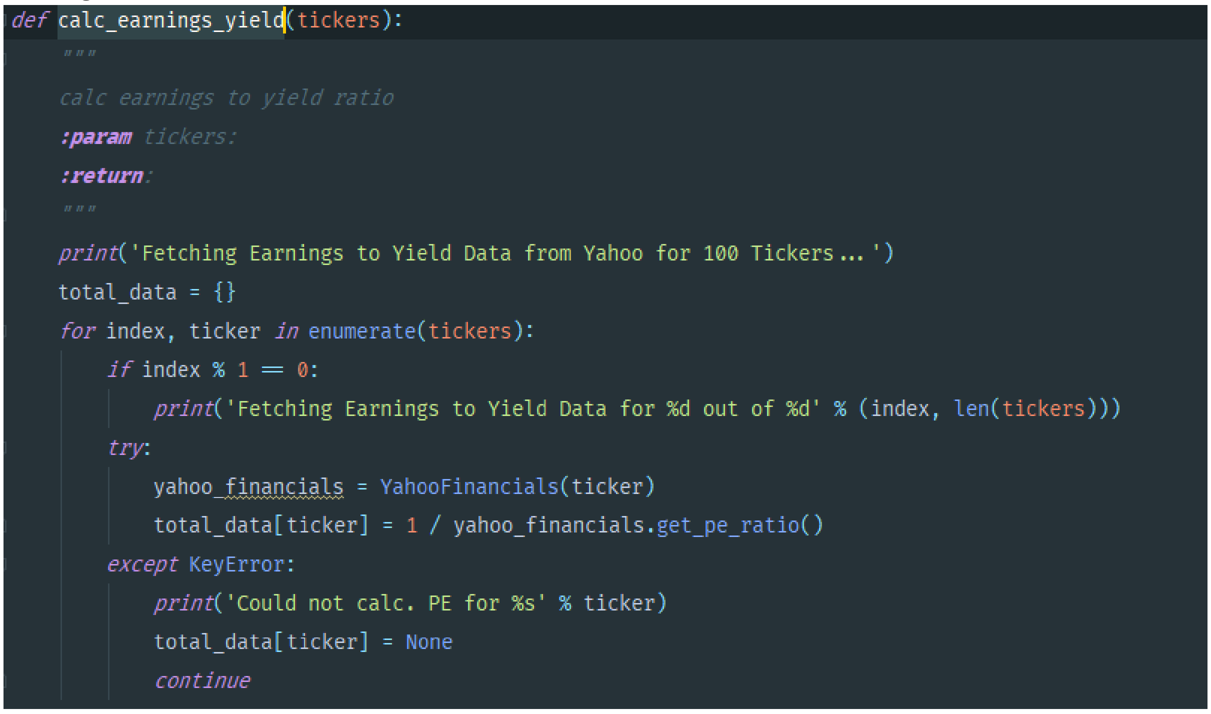


```python
def get_balance_sheets(tickers):
    """

    Get yearly balance sheets
    :param tickers: List of tickers
    :return:
    """
    print('Fetching Yearly Balance Data from Yahoo for 100 Tickers...')
    total_data = {}
    for index, ticker in enumerate(tickers):
        if index % 1 == 0:
            print('Fetching Yearly Balance Data for %d out of %d' % (index, len(tickers)))
        yahoo_financials = YahooFinancials(ticker)
        total_data[ticker] = \
        yahoo_financials.get_financial_stmts('quaterly', 'balance')['balanceSheetHistoryQuarterly'][ticker]
    return total_data
```

2. **Download previous year’s balance sheets for all and calculate the following metrics:**
   - **Earnings Yield**
   - **EBITA**

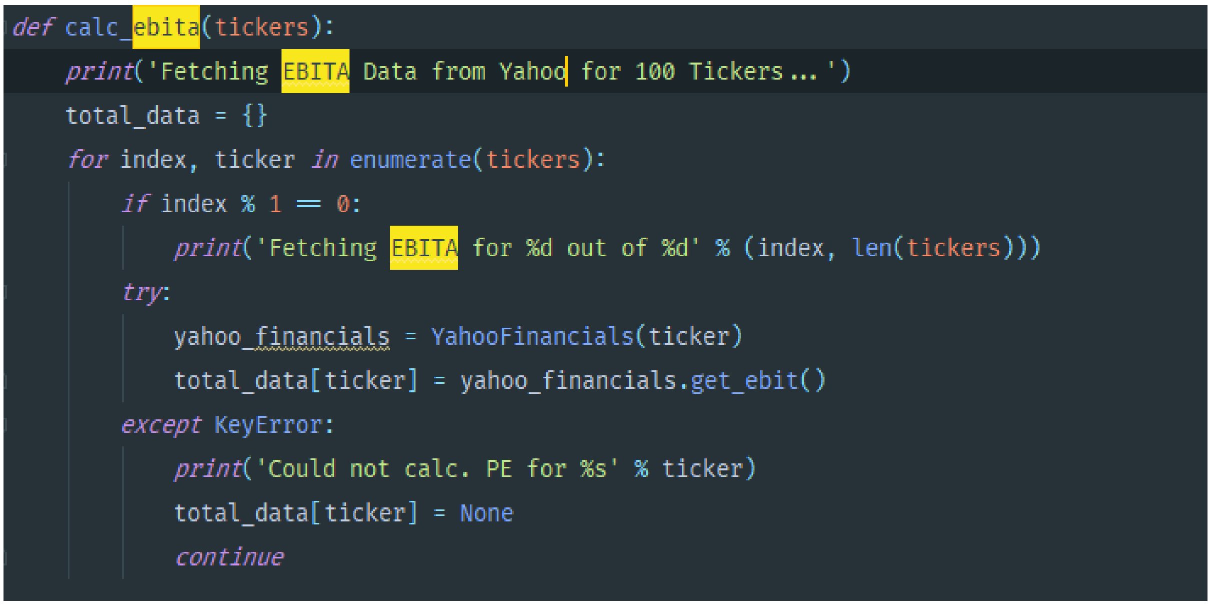
- **Free cash flow yield**
- **Return on Capital Employed**
- **Book to Market**

Earnings Yield is calculated as follows:



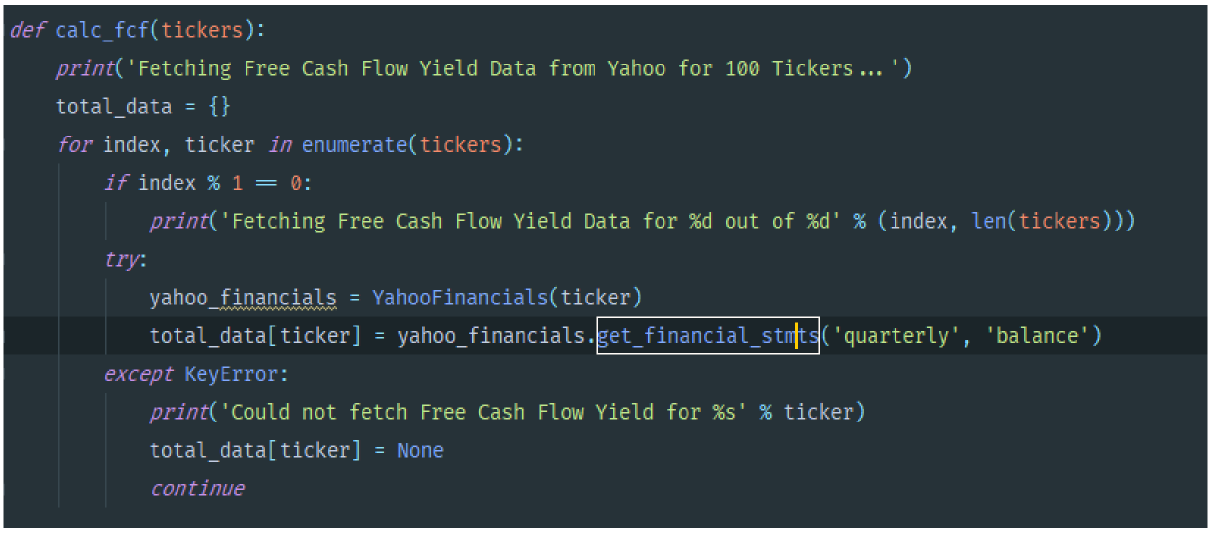
```python
def calc_earnings_yield(tickers):
    """
    calc earnings to yield ratio
    :param tickers:
    :return:
    """
    print('Fetching Earnings to Yield Data from Yahoo for 100 Tickers...')
    total_data = {}
    for index, ticker in enumerate(tickers):
        if index % 1 == 0:
            print('Fetching Earnings to Yield Data for %d out of %d' % (index, len(tickers)))
        try:
            yahoo_financials = YahooFinancials(ticker)
            total_data[ticker] = 1 / yahoo_financials.get_pe_ratio()
        except KeyError:
            print('Could not calc. PE for %s' % ticker)
            total_data[ticker] = None
            continue
```

EBITA



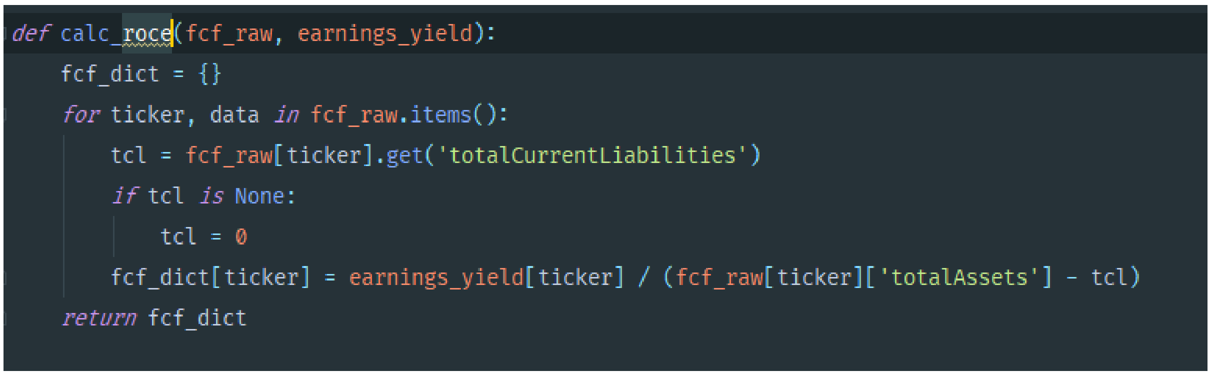
```python
def calc_ebita(tickers):
    print('Fetching EBITA Data from Yahoo for 100 Tickers...')
    total_data = {}
    for index, ticker in enumerate(tickers):
        if index % 1 == 0:
            print('Fetching EBITA for %d out of %d' % (index, len(tickers)))
        try:
            yahoo_financials = YahooFinancials(ticker)
            total_data[ticker] = yahoo_financials.get_ebit()
        except KeyError:
            print('Could not calc. PE for %s' % ticker)
            total_data[ticker] = None
            continue
```

FCF ():



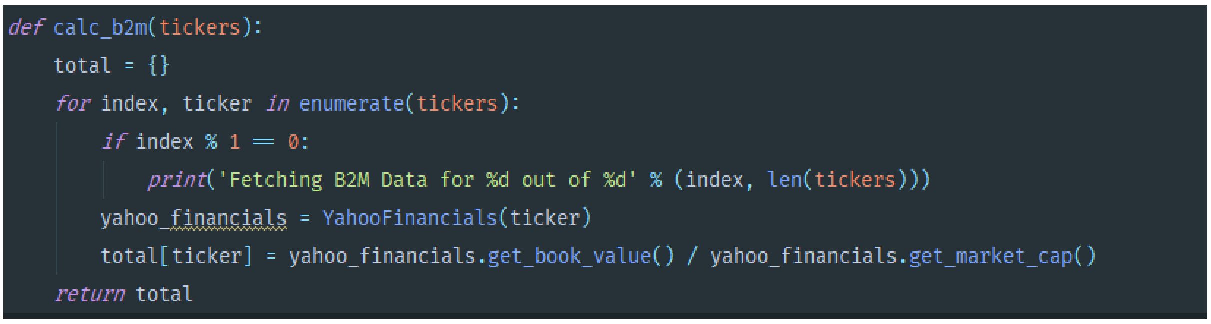
```python
def calc_fcf(tickers):
    print('Fetching Free Cash Flow Yield Data from Yahoo for 100 Tickers... ')
    total_data = {}
    for index, ticker in enumerate(tickers):
        if index % 1 == 0:
            print('Fetching Free Cash Flow Yield Data for %d out of %d' % (index, len(tickers)))
        try:
            yahoo_financials = YahooFinancials(ticker)
            total_data[ticker] = yahoo_financials.get_financial_stmts('quarterly', 'balance')
        except KeyError:
            print('Could not fetch Free Cash Flow Yield for %s' % ticker)
            total_data[ticker] = None
            continue
```

ROCE:

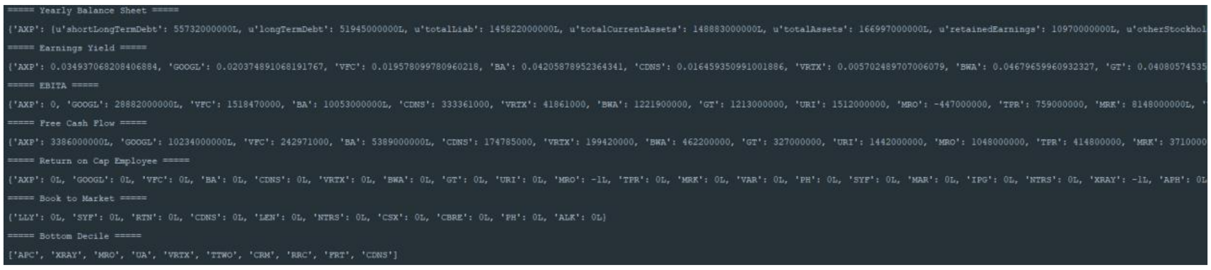


```python
def calc_roce(fcf_raw, earnings_yield):
    fcf_dict = {}
    for ticker, data in fcf_raw.items():
        tcl = fcf_raw[ticker].get('totalCurrentLiabilities')
        if tcl is None:
            tcl = 0
        fcf_dict[ticker] = earnings_yield[ticker] / (fcf_raw[ticker]['totalAssets'] - tcl)
    return fcf_dict
```

B2M:

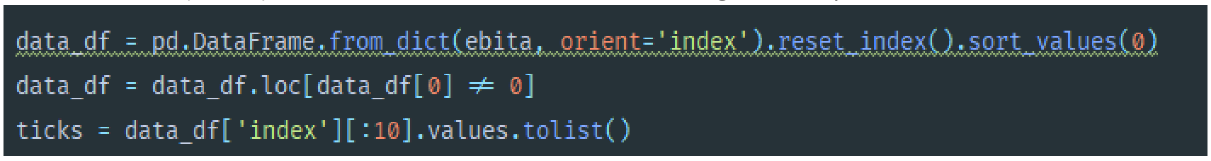


```python
def calc_b2m(tickers):
    total = {}
    for index, ticker in enumerate(tickers):
        if index % 1 == 0:
            print('Fetching B2M Data for %d out of %d' % (index, len(tickers)))
        yahoo_financials = YahooFinancials(ticker)
        total[ticker] = yahoo_financials.get_book_value() / yahoo_financials.get_market_cap()
    return total
```



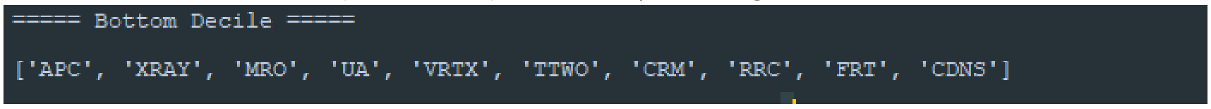
```
===== Yearly Balance Sheet =====
{'AXP': {u'shortLongTermDebt': 55732000000L, u'longTermDebt': 51945000000L, u'totalLiab': 145822000000L, u'totalCurrentAssets': 148883000000L, u'totalAssets': 166997000000L, u'retainedEarnings': 10970000000L, u'otherStockhol
===== Earnings Yield =====
{'AXP': 0.034937068208406884, 'GOOGL': 0.020374891068191767, 'VFC': 0.019578099780960218, 'BA': 0.04205878952364341, 'CDNS': 0.016459350991001886, 'VRTX': 0.005702489707006079, 'BWA': 0.04679659960932327, 'GT': 0.04080574535
===== EBITA =====
{'AXP': 0, 'GOOGL': 28882000000L, 'VFC': 1518470000, 'BA': 10053000000L, 'CDNS': 333361000, 'VRTX': 41861000, 'BWA': 1221900000, 'GT': 1213000000, 'URI': 1512000000, 'MRO': -447000000, 'TPR': 759000000, 'MRK': 8148000000L, '
===== Free Cash Flow =====
{'AXP': 3386000000L, 'GOOGL': 10234000000L, 'VFC': 242971000, 'BA': 5389000000L, 'CDNS': 174785000, 'VRTX': 199420000, 'BWA': 462200000, 'GT': 327000000, 'URI': 1442000000, 'MRO': 1048000000, 'TPR': 414800000, 'MRK': 3710000
===== Return on Cap Employee =====
{'AXP': 0L, 'GOOGL': 0L, 'VFC': 0L, 'BA': 0L, 'CDNS': 0L, 'VRTX': 0L, 'BWA': 0L, 'GT': 0L, 'URI': 0L, 'MRO': -1L, 'TPR': 0L, 'MRK': 0L, 'VAR': 0L, 'PH': 0L, 'SYF': 0L, 'MAR': 0L, 'IPG': 0L, 'NTRS': 0L, 'XRAY': -1L, 'APH': 0L
===== Book to Market =====
{'LLY': 0L, 'SYF': 0L, 'RTN': 0L, 'CDNS': 0L, 'LEN': 0L, 'NTRS': 0L, 'CSX': 0L, 'CBRE': 0L, 'PH': 0L, 'ALK': 0L}
===== Bottom Decile =====
['APC', 'XRAY', 'MRO', 'UA', 'VRTX', 'TTWO', 'CRM', 'RRC', 'FRT', 'CDNS']
```

3. **Arrange stocks in deciles in according to the value of these metrics**
   I took a metric (EBITA), and sorted the stocks based on it to get the top and bottom decile.



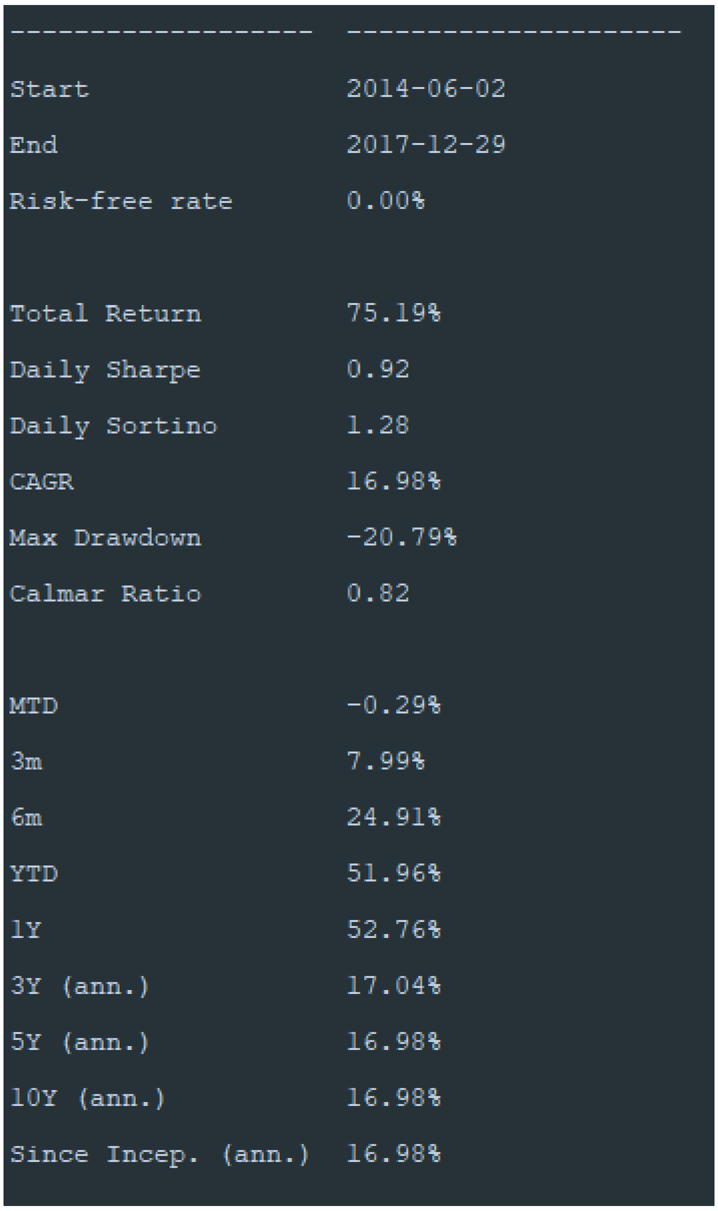
```
data_df = pd.DataFrame.from_dict(ebita, orient='index').reset_index().sort_values(0)
data_df = data_df.loc[data_df[0] ≠ 0]
ticks = data_df['index'][:10].values.tolist()
```

   Then I took the value decile (bottom one) for further processing

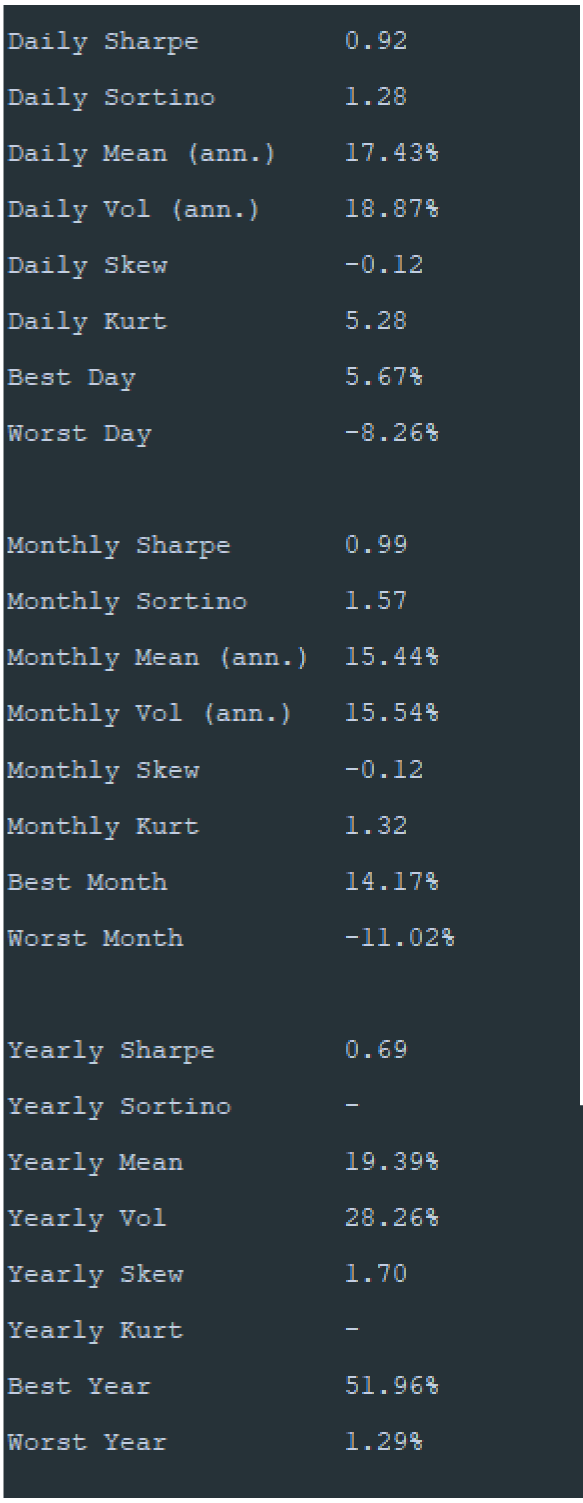


```
===== Bottom Decile =====

['APC', 'XRAY', 'MRO', 'UA', 'VRTX', 'TTWO', 'CRM', 'RRC', 'FRT', 'CDNS']
```

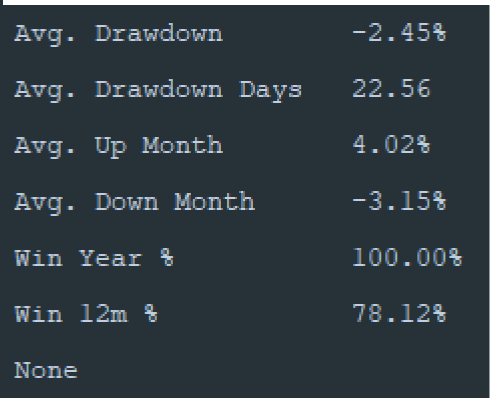
4. **For the value decile stocks, calculate the following (from the time of publishing of last Annual report to date) risk and return metrics**
   I used FFN library to get the results for CAGR etc.
   The results are as follows:



```
-------------------  ---------------------
Start                2014-06-02
End                  2017-12-29
Risk-free rate       0.00%

Total Return         75.19%
Daily Sharpe         0.92
Daily Sortino        1.28
CAGR                 16.98%
Max Drawdown         -20.79%
Calmar Ratio         0.82

MTD                  -0.29%
3m                   7.99%
6m                   24.91%
YTD                  51.96%
1Y                   52.76%
3Y (ann.)            17.04%
5Y (ann.)            16.98%
10Y (ann.)           16.98%
Since Incep. (ann.)  16.98%
```

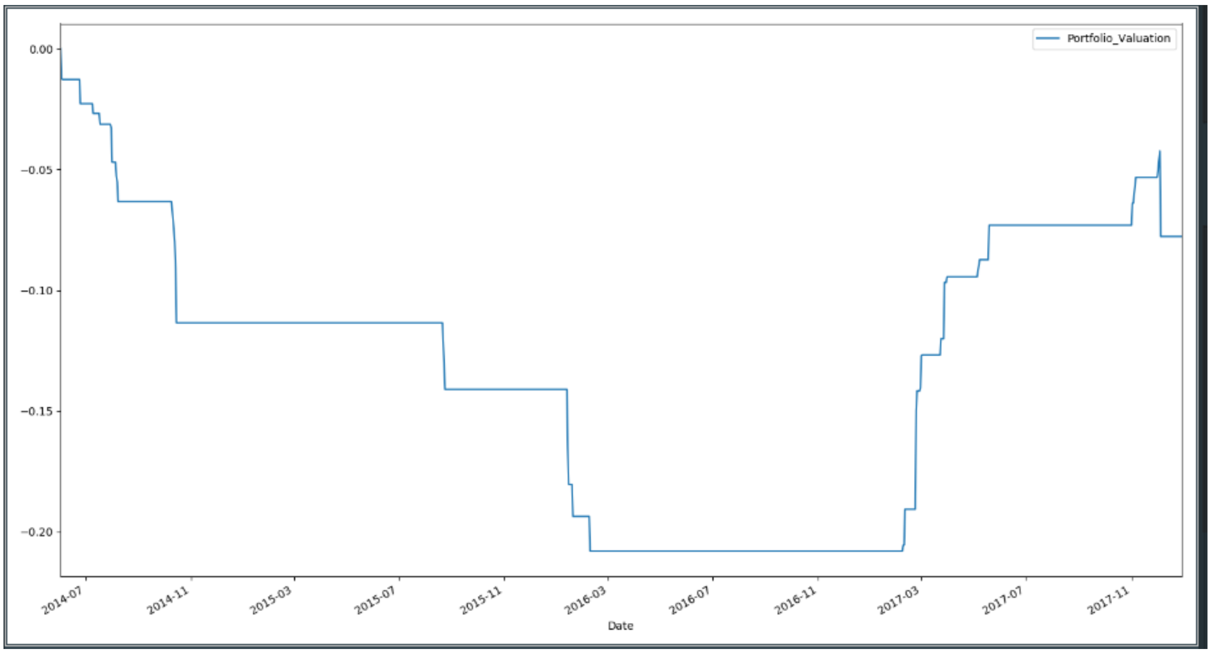


```
Daily Sharpe             0.92
Daily Sortino            1.28
Daily Mean (ann.)        17.43%
Daily Vol (ann.)         18.87%
Daily Skew               -0.12
Daily Kurt               5.28
Best Day                 5.67%
Worst Day                -8.26%

Monthly Sharpe           0.99
Monthly Sortino          1.57
Monthly Mean (ann.)      15.44%
Monthly Vol (ann.)       15.54%
Monthly Skew             -0.12
Monthly Kurt             1.32
Best Month               14.17%
Worst Month              -11.02%

Yearly Sharpe            0.69
Yearly Sortino           -
Yearly Mean              19.39%
Yearly Vol               28.26%
Yearly Skew              1.70
Yearly Kurt              -
Best Year                51.96%
Worst Year               1.29%
```

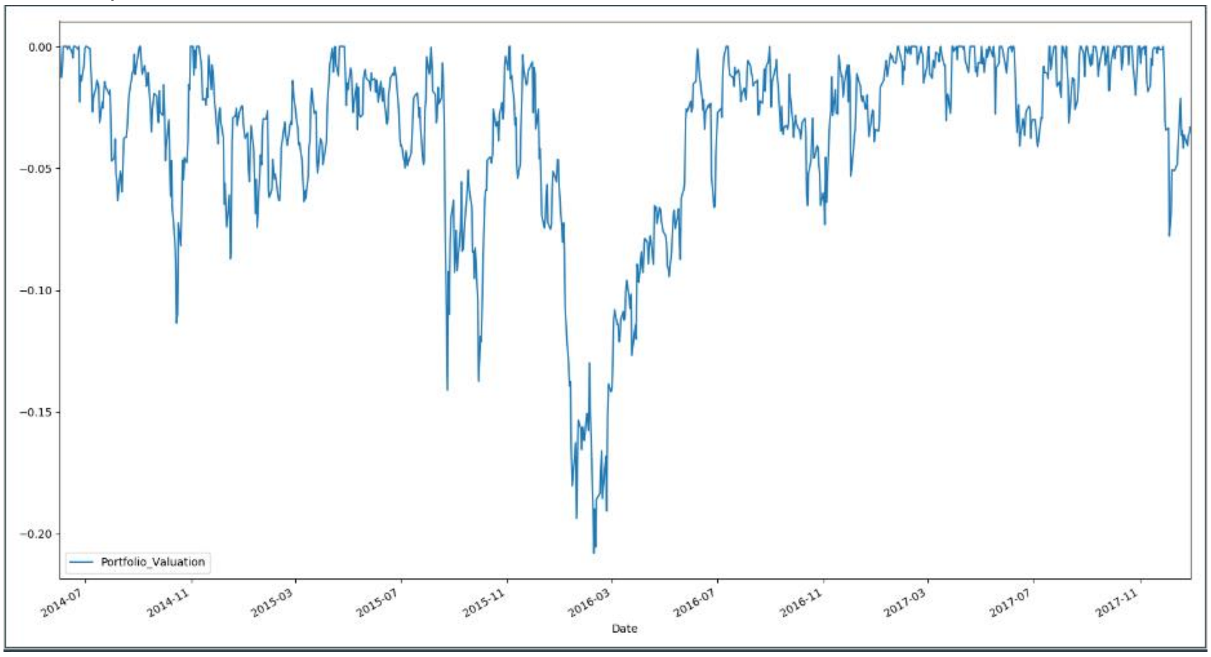


```
Avg. Drawdown            -2.45%
Avg. Drawdown Days       22.56
Avg. Up Month            4.02%
Avg. Down Month          -3.15%
Win Year %               100.00%
Win 12m %                78.12%
None
```

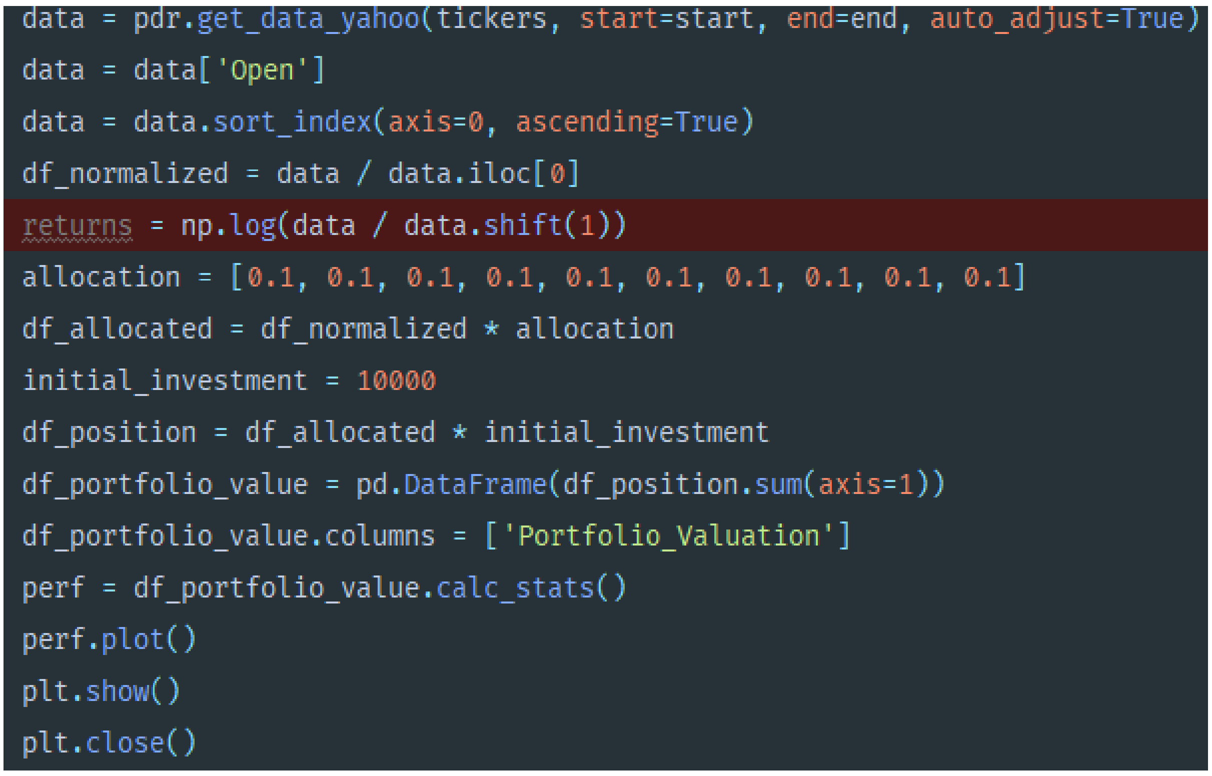
A few plots depicting drawdown are as follows:
Max Drawdown:



Max Daily Drawdown:



The code that finds the metrics is as follows:



```
data = pdr.get_data_yahoo(tickers, start=start, end=end, auto_adjust=True)
data = data['Open']
data = data.sort_index(axis=0, ascending=True)
df_normalized = data / data.iloc[0]
returns = np.log(data / data.shift(1))
allocation = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
df_allocated = df_normalized * allocation
initial_investment = 10000
df_position = df_allocated * initial_investment
df_portfolio_value = pd.DataFrame(df_position.sum(axis=1))
df_portfolio_value.columns = ['Portfolio_Valuation']
perf = df_portfolio_value.calc_stats()
perf.plot()
plt.show()
plt.close()
```