Appendix A

User Guide

1. How to run the algorithm

The Algorithm is developed in Quantopian Platform. In order to execute algorithm, you must execute it in Quantopian platform.

Step 1: Go to quantopian.com

Step 2: Sign Up for an account

Step 3: After Log in, In Research Tab, click into Algorithm

Step 4: Delete all initialised file and paste the algorithm into the text section

Step 5: Choose the date range you want to test and the initial capital you want to have.

My default date testing range is from 01-01-2003(1st January 2019) to 01-04-2019(1st April 2019)

My default capital is $100,000

Step 6: Wait for the backtest to execute, you can then view all the portfolio analysis in notebook section, after the backtest is finished

Note: My_Magic_Formula.py and My_Value_Long_only.py are developed by me, while Acquirer_Multiples_Developed_from_Black_Cat.py and F_Score_Developed_from_Fleury.py are developed from works of two members on Quantopian Community.

Links of their work:

Acquirer_Multiples_Developed_from_Black_Cat: https://www.quantopian.com/posts/acquirers-multiple-based-on-deep-value-number-fundamentals (BlackCat Nov 26 2018)

F-Score_Developed_from_Fleury:https://www.quantopian.com/posts/piotroskis-f-score-algorithm to buy stocks based on F-Score (Guy Fleury July 17 2018)

Appendix B

Source Code

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary.

Quang Dung Nguyen
22nd April 2019

1.Value Long-Only Investing Algorithm

```
"""
Author Quang Dung Nguyen
Â
Create a scoring-system based on valuation ratio
Trading universe is defined by Tradable US Securities, top 2000 market cap and Sector Defined,
It is then be monitored by the momentum and volatility
"""
from quantopian.algorithm import attach_pipeline, pipeline_output
from quantopian.pipeline import Pipeline
from quantopian.pipeline import CustomFactor
from quantopian.pipeline.data.builtin import USEquityPricing
from quantopian.pipeline.data import morningstar
from quantopian.pipeline.classifiers.morningstar import Sector
from quantopian.pipeline.filters import QTradableStocksUS
from quantopian.pipeline.factors.fundamentals import MarketCap
import pandas as pd
import numpy as np

# Constraint Parameters
MAX_GROSS_LEVERAGE = 1.0
MAX_LONG_POSITION_SIZE = 0.04   # 4% about 25 stocks
"""
Initialize the function
"""
def initialize(context):
    # Schedule my rebalance function at beginning of each month 30 minutes after market open
    schedule_function(rebalance,
                      date_rules.month_start(days_offset=0),
                      time_rules.market_open(hours=0,minutes=30))

    # Schedule my plotting function everyday
    schedule_function(func=record_vars,
                      date_rule=date_rules.every_day(),
                      time_rule=time_rules.market_close())

    # initialize our stock selector
    my_pipe = make_pipeline()
    attach_pipeline(my_pipe, 'filtered_top_stocks')

    # set my leverage
    context.long_leverage = 1

    # set the portfolio to only take the long positions, never take short position
    set_long_only()

    # set the slippage and commision for the portfolio
    set_slippage_and_commisions()

"""
    A function to create our dynamic stock selector (pipeline).
"""
def make_pipeline():

    pipe = Pipeline()

    # Set the universe to the QTradableStocksUS & stocks with Sector and Top 2000 by MarketCap
    universe = MarketCap().top(2000, mask = QTradableStocksUS()) & Sector().notnull()

    #filter more with momentum and volarility filter(lowest 600 volatility stocks)

    momentum = Momentum()

    volatility = Volatility()
```

```
    volatility_rank = volatility.rank(mask=universe, ascending=True)

    pipe.set_screen(universe  & (volatility_rank.top(600)) & (momentum>1))

    # Creating Price_to_book,Price_to_earnings, and return_on_assets, return_on_Equity, Return
 on Invested Capital Objects, and Dividend_yield Object and Rank them

    #Create Price to book and Price to Earning and rank them, the lower the ratio, the better
    pb = Price_to_Book()
    pb_rank = pb.rank(mask=universe, ascending=True)

    pe = Price_to_Earnings()
    pe_rank = pe.rank(mask=universe, ascending=True)

    #Create Return on Assets, Return on Equity, Return on Invested Capital and Dividend Yield
Class and rank them,the higher the ratio, the better
    roa = Return_on_Assets()
    roa_rank = roa.rank(mask=universe, ascending=False)

    roe = Return_on_Equity()
    roe_rank = roe.rank(mask=universe, ascending=False)

    roic = Return_on_Invested_Capital()
    roic_rank = roic.rank(mask=universe, ascending=False)

    earnings_yield = Earnings_Yield()
    EY_rank   = earnings_yield.rank(ascending=False, mask=universe)


    dy = Dividend_Yield()
    dy_rank = dy.rank(mask=universe, ascending=False)


    #Give 1 weight forall metrics such as P/e,P/B,Dividend Yield,Return on Assets,Equity and I
nvested Capital
    the_ranking_score = (pb_rank+pe_rank+dy_rank+roa_rank+roe_rank+roic_rank*2 + EY_rank)/8

    # Rank the combo_raw and add that to the pipeline
    pipe.add(the_ranking_score.rank(mask=universe), 'ranking_score')

    return pipe
"""
    Called every day before market open.
"""
def before_trading_start(context, data):
    # Call pipeline_output to get the output
    context.output = pipeline_output('filtered_top_stocks')

    # Narrow down the securities to only the top 25 & update my universe
    context.long_list = context.output.sort_values(['ranking_score'], ascending=True).iloc[:25
].dropna()
"""
    Execute orders according to our schedule_function() timing.
"""
def rebalance(context,data):

    if len(context.long_list) != 0:
        long_weight = MAX_GROSS_LEVERAGE/ float(len(context.long_list))
    else:
        long_weight = 0

    #maximum weight per single stock
```

```
    if long_weight > MAX_LONG_POSITION_SIZE :
        long_weight = 0.04

    # if the stock is in the long_list, buy the stock
    for long_stock in context.long_list.index:
        if data.can_trade(long_stock):
            order_target_percent(long_stock, long_weight)

    # if the stock is currently in the porfolio
    # but not longer in the long_list,sell the stock
    for stock in context.portfolio.positions:
        if stock not in context.long_list.index:
            if data.can_trade(stock):
                order_target(stock, 0)
"""
counting number of positions
"""
def position_count(context, data):
    num_of_position = 0
    for stock,position in context.portfolio.positions.items():
        if position.amount > 0:
            num_of_position +=1
    return num_of_position
"""
    Plot variables at the end of each day.
"""
def record_vars(context, data):

     # Record and plot the leverage and number of positions of our portfolio over time.
    record(leverage = context.account.leverage,
          exposure = context.account.net_leverage,
          number_of_position=position_count(context, data))
"""
    Define slippage and commission. Fixed slippage at 5 basis point and volume_limit is 0.1%
    Commission is set at Interactive Broker Rate: $0.05 per share with minimum trading cost of
 $1 per           transaction
"""
def set_slippage_and_commisions():
    set_slippage(
        us_equities=slippage.FixedBasisPointsSlippage(basis_points=5, volume_limit=0.1))

    set_commission(commission.PerShare(cost = 0.05, min_trade_cost = 1))

"""
Custom Class Momentum Price of 10 days ago/ Price of 30 days ago.
"""
class Momentum(CustomFactor):

    # Pre-declare inputs and window_length
    inputs = [USEquityPricing.close]
    window_length = 30
    # Last row of 10-days ago / First Row (30 days ago)
    def compute(self, today, assets, out, close):
        out[:] = close[-10]/close[0]
"""
Custom Class Volatility
"""
class Volatility(CustomFactor):
    inputs = [USEquityPricing.close]
    window_length = 15
    # compute standard deviation for the last 15-day
    def compute(self, today, assets, out, close):
```

```
        out[:] = np.std(close, axis=0)
"""
Custom Class Price to Book
"""
class Price_to_Book(CustomFactor):

    inputs = [morningstar.valuation_ratios.pb_ratio]
    window_length = 1


    def compute(self, today, assets, out, pb):
        pb = nanfill(pb)
        out[:] = pb
"""
Custom Class Price to Earnings
"""
class Price_to_Earnings(CustomFactor):

    inputs = [morningstar.valuation_ratios.pe_ratio]
    window_length = 1


    def compute(self, today, assets, out, pe):
        pe = nanfill(pe)
        out[:] = pe
"""
Custom Class Return on Assets
"""
class Return_on_Assets(CustomFactor):

    inputs = [morningstar.operation_ratios.roa]
    window_length = 1

    def compute(self, today, assets, out, roa):
        roa = nanfill(roa)
        out[:] = roa
"""
Custom Class Return on Equity
"""
class Return_on_Equity(CustomFactor):

    inputs = [morningstar.operation_ratios.roe]
    window_length = 1


    def compute(self, today, assets, out, roe):
        roe = nanfill(roe)
        out[:] = roe[-1]
"""
Custom Class Return on Invested Capital
"""
class Return_on_Invested_Capital(CustomFactor):

    # Pre-declare inputs and window_length
    inputs = [morningstar.operation_ratios.roic]
    window_length = 1


    def compute(self, today, assets, out, roic):
        roic = nanfill(roic)
        out[:] = roic
"""
Custom Class Dividend Yield
```

```
"""
class Dividend_Yield(CustomFactor):
    inputs = [morningstar.valuation_ratios.dividend_yield]
    window_length = 1
    def compute(self, today, assets, out, d_y):
        dy = nanfill(d_y)
        out[:] = dy
"""
Custom class Earnings_Yield
"""
class Earnings_Yield(CustomFactor):
    inputs = [morningstar.income_statement.ebit, morningstar.valuation.enterprise_value]
    window_length = 1
    def compute(self, today, assets, out, ebit, ev):
        ey = ebit[-1] / ev[-1]
        out[:] = ey
"""
Fillout nan values by  Blue Seahawk
Links :https://www.quantopian.com/posts/forward-filling-nans-in-pipeline-custom-factors
"""
def nanfill(arr):
    mask = np.isnan(arr)
    idx  = np.where(~mask,np.arange(mask.shape[1]),0)
    np.maximum.accumulate(idx,axis=1, out=idx)
    arr[mask] = arr[np.nonzero(mask)[0], idx[mask]]
    return arr
```

2. My Magic Formula Algorithm

```
"""
Author :Quang Dung Nguyen
Create Magic Formula by Joel Greenblatt
    1. Establish a minimum market capitalization of $50 million.
    2. Exclude utility and financial sectors.
    3. Calculate company's earnings yield = EBIT / enterprise value.
    4. Calculate company's return on capital = EBIT / (net fixed assets + working capital).
    5. Rank all companies by highest earnings yield and highest return on capital.
    6. Invest in 25 highest ranked companies, buying in January and sell in December before th
e year ends
"""
from quantopian.pipeline.classifiers.fundamentals import Sector
from quantopian.algorithm import attach_pipeline, pipeline_output
from quantopian.pipeline import Pipeline
from quantopian.pipeline.data.builtin import USEquityPricing
from quantopian.pipeline.factors import AverageDollarVolume
from quantopian.pipeline import CustomFactor
from quantopian.pipeline.filters import QTradableStocksUS
from quantopian.pipeline.data import morningstar, Fundamentals
from quantopian.pipeline.filters.morningstar import IsPrimaryShare, IsDepositaryReceipt
import quantopian.pipeline.factors.fundamentals
import pandas as pd
import numpy as np

MAX_GROSS_LEVERAGE = 1.0
MAX_LONG_POSITION_SIZE = 0.04   # 4% per position or about 25 stocks

def initialize(context):

    # Create our dynamic stock selector.
    context.capacity = 25

    # initialize our stock selector
    my_pipe = make_pipeline()
    attach_pipeline(my_pipe, 'my_pipeline')

    # set the portfolio to only take the long positions, never take short position
    set_long_only()

    # set the slippage and commision for the portfolio
    set_slippage_and_commisions()

    #schedule for buying a week after the year start
    schedule_function(func= buying_in_January,
                      date_rule=date_rules.month_start(4),
                      time_rule=time_rules.market_open())

    # Schedule Selling Function at the beginning of each December before the year start
    schedule_function(sell_in_December,date_rules.every_day(),time_rules.market_close())

    #Schedule my plotting function
    schedule_function(record_vars,date_rules.every_day(), time_rules.market_open())

def make_pipeline():
    # Set the universe to the QTradableStocksUS & stocks with Sector defined
    universe  = QTradableStocksUS() & Sector().notnull()

    # A Filterfor market cap greater than 1  billion
    medium_cap = Fundamentals.market_cap.latest > 1000000000

    # We don't choose from Financial and Ultility Sector
    not_Finan_and_Ulti = ((Sector != 103) & (Sector != 207))
```

```
    # the final universe
    universe = universe & medium_cap & not_Finan_and_Ulti

    #Create an object of Earnings_Yield and rank them in descending order
    earnings_yield = Earnings_Yield()
    EY_rank   = earnings_yield.rank(ascending=False, mask=universe)

    #Create an object of Return on invested Capital and rank them in descending order
    roic= Return_on_Invested_Capital()
    roic_rank = roic.rank(ascending=False, mask=universe)

    # the rank of magic formula is equal to the sum of rank of Earning Yields and Return on in
vested capital
    Magic_Formula_rank   = EY_rank + roic_rank

    pipe = Pipeline(columns = {
        'earnings_yield': earnings_yield,
        'Earnings_yield_rank':EY_rank,
        'roic'          : roic,
        'Roic_rank': roic_rank,
        'MagicFormula_rank': Magic_Formula_rank,
    }, screen = universe )
    return pipe

"""
    Called every day before market open.
"""
def before_trading_start(context, data):
    context.output=pipeline_output('my_pipeline')
    context.buy_list = context.output.sort_values(['MagicFormula_rank'], ascending=True).head(
25).dropna()

    # equal weighting of a position, = 1.0 leverage / number of stocks in buy list
    if len(context.buy_list) != 0:
        context.weight = MAX_GROSS_LEVERAGE / float(len(context.buy_list))
    else:
        context.weight = 0

    #maximum weight per single stock
    if context.weight > MAX_LONG_POSITION_SIZE :
        context.weight = 0.04

"""
Buying on January the top 25 stocks in the list
"""
def buying_in_January(context, data):

    today = get_datetime('US/Eastern')
    if today.month == 1:
        for stock in context.buy_list.index:
            if data.can_trade(stock):
                    order_target_percent(stock, context.weight)
"""
Sell at the beginning of December all positions in the portfolio
"""
def sell_in_December(context,data):
    today = get_datetime('US/Eastern')
    if today.month == 12 and context.portfolio.positions_value != 0:
        for stock in context.portfolio.positions.iterkeys():
            if stock not in context.buy_list.index:
                if data.can_trade(stock):
                    order_target(stock, 0)
```

```
"""
    Define slippage and commission. Fixed slippage at 5 basis point and volume_limit is 0.1%
    Commission is set at Interactive Broker Rate: $0.05 per share with minimum trading cost of
 $1 per transaction
"""
def set_slippage_and_commisions():
    set_slippage(
        us_equities=slippage.FixedBasisPointsSlippage(basis_points=5, volume_limit=0.1))

    set_commission(commission.PerShare(cost = 0.05, min_trade_cost = 1))
"""
counting number of positions
"""
def position_count(context, data):
    num_of_position = 0
    for stock,position in context.portfolio.positions.items():
        if position.amount > 0:
            num_of_position +=1
    return num_of_position

"""
    Plot variables at the end of each day,record leverage of the portfolio and number of posit
ions
"""
def record_vars(context, data):

     # Record and plot the leverage and number of positions our portfolio over time.
    record(leverage = context.account.leverage,
          number_of_positions=position_count(context, data))


"""
Custom class Earnings_Yield
"""
class Earnings_Yield(CustomFactor):
    inputs = [morningstar.income_statement.ebit, morningstar.valuation.enterprise_value]
    window_length = 1

    def compute(self, today, assets, out, ebit, ev):
        ey = ebit[-1] / ev[-1]
        out[:] = ey

"""
Custom Class Return on Invested Capital
"""
class Return_on_Invested_Capital(CustomFactor):

    # Pre-declare inputs and window_length
    inputs = [morningstar.operation_ratios.roic]
    window_length = 1


    def compute(self, today, assets, out, roic):
        out[:] = roic[-1]
```

3. The Acquirer Multiple Algorithms

Note: This is a Modified Version from BlackCat Nov 26 2018 at https://www.quantopian.com/posts/acquirers-multiple-based-on-deep-value-number-fundamentals

Adding interactive broker commission rates in set_slippage_and_commissions as well records_vars to record watched variables

```
"""
A Modified Version from BlackCat Nov 26 2018 at  https://www.quantopian.com/posts/acquirers-multiple-based-on-deep-value-number-fundamentals
Adding interactive broker commission rates in set_slippage_and_commissions as well records_vars to record watched variables
"""

"""
Rank stocks by 'The Acquirer's Multiple'
Â
Picks the top 25 stocks with the lowest EV/EBIT. Equal weight for each stock.  Rebalances every year.
Â
Instructions to Use
--------------------
Backtest start date should be on the first day of a month:
  - It will buy stocks on that day (or the first trading day after, if a holiday).
  - On the same date next year, it will sell all those stocks and buy new ones.
Â
eg:
  Start on 1st Jan, it will rebalance on 1st Jan (or first trading day of every Jan).
  Start on 1st April, it will rebalance on 1st April (or first trading day of every April).
Â
Â
Â
Algorithim
-----------
1) Operates on QTradableStocksUS universe
2) Filters out tradable stocks (https://www.quantopian.com/posts/pipeline-trading-universe-best-practice).
3) Filters out financial companies.
4) Get TTM EBIT from past 4 quarterly results
5) Filters out any stocks with -ve EBIT
6) The stocks from lowest EV/EBIT to highest.  Buy the first NUM_STOCKS_TO_BUY stocks.
7) Rebalance eery year.
Â
Â
"""
import quantopian.algorithm as algo
from quantopian.pipeline import Pipeline, CustomFactor
from quantopian.pipeline.data.builtin import USEquityPricing
from quantopian.pipeline.filters import QTradableStocksUS
from quantopian.pipeline.filters.morningstar import IsPrimaryShare
from quantopian.pipeline.data import Fundamentals
import numpy as np
import pandas
import scipy.stats as stats
import quantopian.optimize as opt
from quantopian.optimize import TargetWeights

def initialize(context):
    """
    Called once at the start of the algorithm.
    """
    # Call rebalance() every month, 1 hour after market open.
    algo.schedule_function(
        rebalance,
        algo.date_rules.month_start(days_offset=0),
        algo.time_rules.market_open(minutes=60),
    )

    # Record tracking variables at the end of each day.
    algo.schedule_function(
```

```
        record_vars,
        algo.date_rules.every_day(),
        algo.time_rules.market_close(),
    )

    # Stores which month we do our rebalancing.  Will be assigned the month of the first day w
e are run.
    context.month_to_run = -1

    context.NUM_STOCKS_TO_BUY = 25

    # Create our dynamic stock selector.
    algo.attach_pipeline(make_pipeline(context), 'pipeline')

    # Set the slippage and commisions function
    set_slippage_and_commisions()

# From Doug Baldwin: https://www.quantopian.com/posts/trailing-twelve-months-ttm-with-as-of-da
te
#
# Takes a quarterly factor and changes it to TTM.
class TrailingTwelveMonths(CustomFactor):
    window_length=400
    window_safe = True  # OK as long as we dont use per share fundamentals: https://www.quanto
pian.com/posts/how-to-make-factors-be-the-input-of-customfactor-calculation
    outputs=['factor','asof_date']

    # TODO: what if there are more than 4 unique elements.

    def compute(self, today, assets, out, values, dates):
        out.factor[:] = [    # Boolean masking of arrays
            (v[d + np.timedelta64(52, 'W') > d[-1]])[
                np.unique(
                    d[d + np.timedelta64(52, 'W') > d[-1]],
                    return_index=True
                )[1]
            ].sum()
            for v, d in zip(values.T, dates.T)
        ]
        out.asof_date[:] = dates[-1]

def make_pipeline(context):

    #  Convert quarterly fundamental data to TTM.
    (
        ebit_ttm,
        ebit_ttm_asof_date
    ) = TrailingTwelveMonths(
        inputs=[
            Fundamentals.ebit,
            Fundamentals.ebit_asof_date,
        ]
    )

    ev = Fundamentals.enterprise_value.latest


    # If we dont use it, there are around 4000 stocks.  But we must
    # screen for price & liquidity ourselves. And I think Quantopian
    # may only support their own stock universe(s) later.
    base_universe = QTradableStocksUS()

    # Filter tradable stocks.
```

```
    # https://www.quantopian.com/posts/pipeline-trading-universe-best-practice

    # Filter for primary share equities. IsPrimaryShare is a built-in filter.
    primary_share = IsPrimaryShare()

    # Equities listed as common stock (as opposed to, say, preferred stock).
    # 'ST00000001' indicates common stock.
    common_stock = Fundamentals.security_type.latest.eq('ST00000001')

    # Non-depositary receipts. Recall that the ~ operator inverts filters,
    # turning Trues into Falses and vice versa
    not_depositary = ~Fundamentals.is_depositary_receipt.latest

    # Equities not trading over-the-counter.
    not_otc = ~Fundamentals.exchange_id.latest.startswith('OTC')

    # Not when-issued equities.
    not_wi = ~Fundamentals.symbol.latest.endswith('.WI')

    # Equities without LP in their name, .matches does a match using a regular
    # expression
    not_lp_name = ~Fundamentals.standard_name.latest.matches('.* L[. ]?P.?$')

    # Equities with a null value in the limited_partnership Morningstar
    # fundamental field.
    not_lp_balance_sheet = Fundamentals.limited_partnership.latest.isnull()

    # Equities whose most recent Morningstar market cap is not null have
    # fundamental data and therefore are not ETFs.
    have_market_cap = Fundamentals.market_cap.latest.notnull()

    ############################
    # Filters specific to EV.
    ############################

    # We cannot have -ve EBIT.  This will give is a -ve EBIT/EV, which is meaningless.
    negative_earnings = ebit_ttm < 0

    # Screen out financials.  EV is meaningless for companies that lend money out as part of t
heir business.
    is_a_financial_company = (Fundamentals.morningstar_sector_code.latest.eq(103) )

    # Filter for stocks that pass all of our previous filters.
    tradeable_stocks = (
    primary_share
    & common_stock
    & not_depositary
    & not_otc
    & not_wi
    & not_lp_name
    & not_lp_balance_sheet
    & have_market_cap
    & ~negative_earnings
    & ~is_a_financial_company
    )

    # EV/EBIT ratio.  Lower values are better, like a PE ratio.
    # I use this instead of EBIT/EV (earnings yield) because
    # earnings yield will not make sense if EV is -ve.
    ev_over_ebit = ev / ebit_ttm

    Lowest_Ev_over_ebit_stocks = ev_over_ebit.bottom(context.NUM_STOCKS_TO_BUY, mask=base_univ
erse & tradeable_stocks)
```

```
    return Pipeline(
        columns={
            'ebit_ttm': ebit_ttm,
            'ebit_ttm_asof_date': ebit_ttm_asof_date,
            'ev': ev,
            'ev_over_ebit': ev_over_ebit,
        },
        screen = Lowest_Ev_over_ebit_stocks
    )

def before_trading_start(context, data):
    """
    Called every day before market open.
    """

    # Get pipeline output
    context.output = algo.pipeline_output('pipeline')
    context.output['ebit_ttm_asof_date'] =context.output['ebit_ttm_asof_date'].astype('datetim
e64[ns]')

    # Sort: lowest ev/ebit first
    context.output = context.output.sort_values('ev_over_ebit');

    # These are the securities that we are interested in trading each day.
    context.security_list = context.output.index

def rebalance(context, data):
    """
    Execute orders according to our schedule_function() timing.
    """


    # We will rebalance on the 1st day of the month that we started running on.
    rebalanceNow = False;

    today = get_datetime('US/Eastern')
    if (context.month_to_run == -1):
        context.month_to_run = today.month
        print str("Rebalancing will be done on 1st day of month " + str( context.month_to_run
) )
        rebalanceNow = True;
    else:
        if (context.month_to_run == today.month):
            rebalanceNow = True;

    if (rebalanceNow):
        print "REBALANCING."

        # Rank the stocks withthe lowest EV/EBIT ratio first.
        context.output['ev_over_ebit_rank'] = context.output['ev_over_ebit'].rank(ascending=Tr
ue)

        # Use Q's order_optimal_portfolio API.
        # Use equal weights for all items in pipeline
        context.weights = {}
        for sec in context.security_list:
            if data.can_trade(sec):
                context.weights[sec] = 0.99/context.NUM_STOCKS_TO_BUY

        objective=TargetWeights(context.weights)
        algo.order_optimal_portfolio(
            objective=objective,
```

```
            constraints=[],
        )

"""
    Define slippage and commission. Fixed slippage at 5 basis point and volume_limit is 0.1%
    Commission is set at Interactive Broker Rate: $0.05 per share with minimum trading cost of
 $1 per transaction
"""
def set_slippage_and_commisions():
    set_slippage(
        us_equities=slippage.FixedBasisPointsSlippage(basis_points=5, volume_limit=0.1))

    set_commission(commission.PerShare(cost = 0.05, min_trade_cost = 1))
"""
counting number of positions
"""
def position_count(context, data):
    num_of_position = 0
    for stock,position in context.portfolio.positions.items():
        if position.amount > 0:
            num_of_position +=1
    return num_of_position

"""
    Plot variables at the end of each day,record leverage of the portfolio and number of posit
ions
"""
def record_vars(context, data):

     # Record and plot the leverage and number of positions our portfolio over time.
    record(leverage = context.account.leverage,
          number_of_positions=position_count(context, data))
```

4. The Piotroski’s F-Score algorithm

Note: This is a Modified Version from Guy Fleury on July 17 2018 at https://www.quantopian.com/posts/piotroskis-f-score-algorithm to buy stocks based on F-Score

Adding interactive broker commission rates in set_slippage_and_commissions as well as records_vars to record watched variables and move up the speed by deleting handle_data function

```
"""
A Modified Version from Guy Fleury at July 17 2018 at https://www.quantopian.com/posts/piotros
kis-f-score-algorithm to buy stocks based on F-Score
Adding interactive broker commission rates in set_slippage_and_commissions as well records_var
s to record watched variables
"""

"""
    Creating an algorithm based off the Piotroski Score index which is based off of a score (0
-9)
    Each of the following points in Profitablity, Leverage & Operating Effificieny means one p
oint.
    We are going to select

    Profitability
    - Positive ROA
    - Positive Operating Cash Flow
    - Higher ROA in current year versus last year
    - Cash flow from operations > ROA of current year

    Leverage
    - Current ratio of long term debt < last year's ratio of long term debt
    - Current year's current_ratio > last year's current_ratio
    - No new shares issued this year

    Operating Efficiency
    - Higher gross margin compared to previous year
    - Higher asset turnover ratio compared to previous year

"""

import numpy as np
from quantopian.pipeline.data.builtin import USEquityPricing
from quantopian.pipeline import CustomFactor, Pipeline
from quantopian.pipeline.factors import SimpleMovingAverage
from quantopian.algorithm import attach_pipeline, pipeline_output
from quantopian.pipeline.data import morningstar
from quantopian.pipeline.filters import QTradableStocksUS



class Piotroski(CustomFactor):
    inputs = [
        morningstar.operation_ratios.roa,
        morningstar.cash_flow_statement.operating_cash_flow,
        morningstar.cash_flow_statement.cash_flow_from_continuing_operating_activities,

        morningstar.operation_ratios.long_term_debt_equity_ratio,
        morningstar.operation_ratios.current_ratio,
        morningstar.valuation.shares_outstanding,

        morningstar.operation_ratios.gross_margin,
        morningstar.operation_ratios.assets_turnover,
    ]
    window_length = 22

    def compute(self, today, assets, out,
                roa, cash_flow, cash_flow_from_ops,
                long_term_debt_ratio, current_ratio, shares_outstanding,
                gross_margin, assets_turnover):
        # Calculate firm profit ability making
        profit = (
            (roa[-1] > 0).astype(int) +
```

```
            (cash_flow[-1] > 0).astype(int) +
            (roa[-1] > roa[0]).astype(int) +
            (cash_flow_from_ops[-1] > roa[-1]).astype(int)
        )
        # Calculate different ratio in terms of firm's debt
        leverage = (
            (long_term_debt_ratio[-1] < long_term_debt_ratio[0]).astype(int) +
            (current_ratio[-1] > current_ratio[0]).astype(int) +
            (shares_outstanding[-1] <= shares_outstanding[0]).astype(int)
        )
        # Calculating the efficiency of operation of the firms
        operating = (
            (gross_margin[-1] > gross_margin[0]).astype(int) +
            (assets_turnover[-1] > assets_turnover[0]).astype(int)
        )

        out[:] = profit + leverage + operating


def initialize(context):

    pipe = Pipeline()
    pipe = attach_pipeline(pipe, name='piotroski')

    profit = ROA() + ROAChange() + CashFlow() + CashFlowFromOps()
    leverage = LongTermDebtRatioChange() + CurrentDebtRatioChange() + SharesOutstandingChange(
)
    operating = GrossMarginChange() + AssetsTurnoverChange()
    piotroski = profit + leverage + operating

    ev_ebitda = morningstar.valuation_ratios.ev_to_ebitda.latest > 0
    market_cap = morningstar.valuation.market_cap > 1e9

    pipe.add(piotroski, 'piotroski')
    pipe.set_screen(((piotroski >= 7) | (piotroski <= 3)) & ev_ebitda & market_cap)
    context.is_month_end = False
    schedule_function(set_month_end, date_rules.month_end(1))
    schedule_function(trade_long, date_rules.month_end(), time_rules.market_open())
    schedule_function(trade_short, date_rules.month_end(), time_rules.market_open())
    schedule_function(trade, date_rules.month_end(), time_rules.market_close())

    # set the slippage and commision for the portfolio
    set_slippage_and_commisions()

# Set month end to true
def set_month_end(context, data):

    context.is_month_end = True

# Execute before trading is started
def before_trading_start(context, data):
    if context.is_month_end:
        context.results = pipeline_output('piotroski')
        try:
            context.long_stocks = context.results.sort_values('piotroski', ascending=False).he
ad(10)
            context.short_stocks = context.results.sort_values('piotroski', ascending=True).he
ad(10)
        except:
            print ("In exception")
#placing long position
def trade_long(context, data):
```

```
    for stock in context.long_stocks.index:
        if data.can_trade(stock):
            order_target_percent(stock, 1.0/20)
#placing short position
def trade_short(context, data):
    for stock in context.short_stocks.index:
        if data.can_trade(stock):
            order_target_percent(stock, -1.0/20)

# Selling stocks if it is not in both list
def trade(context, data):

    for stock in context.portfolio.positions:
        if stock not in context.long_stocks.index and stock not in context.short_stocks.index:
            order_target_percent(stock, 0)
    context.is_month_end = False
## Different custom class to calculate the  Triotski score

class ROA(CustomFactor):
    window_length = 1
    inputs = [morningstar.operation_ratios.roa]

    def compute(self, today, assets, out, roa):
        out[:] = (roa[-1] > 0).astype(int)

class ROAChange(CustomFactor):
    window_length = 22
    inputs = [morningstar.operation_ratios.roa]

    def compute(self, today, assets, out, roa):
        out[:] = (roa[-1] > roa[0]).astype(int)

class CashFlow(CustomFactor):
    window_length = 1
    inputs = [morningstar.cash_flow_statement.operating_cash_flow]

    def compute(self, today, assets, out, cash_flow):
        out[:] = (cash_flow[-1] > 0).astype(int)

class CashFlowFromOps(CustomFactor):
    window_length = 1
    inputs = [morningstar.cash_flow_statement.cash_flow_from_continuing_operating_activities,
morningstar.operation_ratios.roa]

    def compute(self, today, assets, out, cash_flow_from_ops, roa):
        out[:] = (cash_flow_from_ops[-1] > roa[-1]).astype(int)

class LongTermDebtRatioChange(CustomFactor):
    window_length = 22
    inputs = [morningstar.operation_ratios.long_term_debt_equity_ratio]

    def compute(self, today, assets, out, long_term_debt_ratio):
        out[:] = (long_term_debt_ratio[-1] < long_term_debt_ratio[0]).astype(int)

class CurrentDebtRatioChange(CustomFactor):
    window_length = 22
    inputs = [morningstar.operation_ratios.current_ratio]

    def compute(self, today, assets, out, current_ratio):
        out[:] = (current_ratio[-1] > current_ratio[0]).astype(int)

class SharesOutstandingChange(CustomFactor):
    window_length = 22
```

```
    inputs = [morningstar.valuation.shares_outstanding]

    def compute(self, today, assets, out, shares_outstanding):
        out[:] = (shares_outstanding[-1] <= shares_outstanding[0]).astype(int)

class GrossMarginChange(CustomFactor):
    window_length = 22
    inputs = [morningstar.operation_ratios.gross_margin]

    def compute(self, today, assets, out, gross_margin):
        out[:] = (gross_margin[-1] > gross_margin[0]).astype(int)

class AssetsTurnoverChange(CustomFactor):
    window_length = 22
    inputs = [morningstar.operation_ratios.assets_turnover]

    def compute(self, today, assets, out, assets_turnover):
        out[:] = (assets_turnover[-1] > assets_turnover[0]).astype(int)
"""
counting number of positions
"""
def position_count(context, data):
    num_of_position = 0
    for stock,position in context.portfolio.positions.items():
        if position.amount > 0:
            num_of_position +=1
    return num_of_position
"""
    Plot variables at the end of each day.
"""
def record_vars(context, data):

     # Record and plot the leverage and number of positions of our portfolio over time.
    record(leverage = context.account.leverage,
          exposure = context.account.net_leverage,
          number_of_position=position_count(context, data))
"""
    Define slippage and commission. Fixed slippage at 5 basis point and volume_limit is 0.1%
    Commission is set at Interactive Broker Rate: $0.05 per share with minimum trading cost of
 $1 per           transaction
"""
def set_slippage_and_commisions():


    set_slippage(
        us_equities=slippage.FixedBasisPointsSlippage(basis_points=5, volume_limit=0.1))

    set_commission(commission.PerShare(cost = 0.05, min_trade_cost = 1))
```