MIRC Sector Momentum Sample Algo (Part 1)

This notebook was created as a demonstration for **McMaster's MIRC** quant club. For this project, we hypothesize that a **company's earning reports** contains information about the **performance of rest of the sector**, and will attempt to **exploit any alpha** if such a relationship exists.

As an example, if we believe that Apple will post good earnings numbers (earnings per share, growth in revenue, etc.) then other tech companies also will post good earnings. Generally good earnings correlate with a jump in the stock price the next day (and vice versa with poor earnings numbers), so we will use Apple as an indicator for the rest of the sector (assuming Apple is the first in the sector to report).

Importing Relevant Packages

Here I will place any important packages that are neccessary for the functions in the notebook to run.

In [4]:

```
from quantopian.pipeline import Pipeline # Used to import from Quantop
ian
from quantopian.research import run_pipeline #
from quantopian.pipeline.factors import Returns, MarketCap # Want stoc
k performance and mkt cap (SP500)
from quantopian.pipeline.data import USEquityPricing, Fundamentals # W
ant stock pricing and earnings

from quantopian.pipeline.filters import QTradableStocksUS, StaticAsset
s # Want to know if the stock

# is tradeable

from quantopian.pipeline.classifiers.fundamentals import Sector # Need
the sector of each company

# From Part 2 #########################################
from quantopian.pipeline.data.factset.estimates import Actuals
from quantopian.pipeline import Pipeline
import quantopian.pipeline.data.factset.estimates as fe
from quantopian.pipeline.domain import US_EQUITIES
from quantopian.research import run_pipeline
#######################################################


# Other conventional packages ########################
import numpy as np # Numeric calculations
import pandas as pd # Dataframe calculations
import matplotlib.pyplot as plt # Plots and charts
```

Importing the Data

We will use **Quantopian's API** to store the constituents of the **S&P500** and save the data of each date into a separate dictionary entry. Since the S&P500 is an index that tracks the performance of the 500 largest companies by market cap, it is easy to create such a pull.

In [5]:
```
def make_pipeline():

    # Pipeline factors
    close_price = USEquityPricing.close.latest # Close price of each s
tock
    market_cap = MarketCap() # Market cap of each stock

    # Pipeline Filters
    QTU = QTradableStocksUS()
    top_500_market_cap = market_cap.top(500)

    QTU_top_500 = QTU & top_500_market_cap

    # Want to make sure that the stock is listed and tradeable on that
day
    has_pricing_data = close_price.notnull()

    return Pipeline(
        # columns takes a dict of {'name of factor goes here':  factor
_variable} arguments (usually)
        columns={'close_price': close_price},
        # screen takes filter arguments
        screen=QTU_top_500 & has_pricing_data
    )
```

Test the Data

Let us take a look at the output and **make sure we have everything we need** before we dedicate the time to pull a much larger dataset.

In [6]:
```
# Want to have this in its own cell so it only needs to be run once
pipeline_test = run_pipeline(make_pipeline(), '2017-1-1', '2017-1-5')
```

Pipeline Execution Time: 21.67 Seconds

In [7]:
```
pipeline_test.keys()
```

Out[7]:
```
Index(['close_price'], dtype='object')
```

So it looks like the **pipeline_test dict** stores everything in the key ["close_price"]. We can **change/add other outputs** (which we will need later) in the columns arg in the **return Pipeline() part of the function above.** Now we will look further into the dict to see how we can access more of the data.

```
In [8]: pipeline_test["close_price"].keys()[0]
```

```
Out[8]: (Timestamp('2017-01-03 00:00:00+0000', tz='UTC', offset='C'),
         Equity(24 [AAPL]))
```

Since I picked a short time frame at the beginning of the year, there was only one day where the market was open, which is why we see only one key here. We will explore further:

```
In [9]: pipeline_test["close_price"]["2017-01-03"].head() # .head() shows just
the first 5 entries
```

```
Out[9]: Equity(24 [AAPL])    115.84
        Equity(62 [ABT])      38.42
        Equity(64 [GOLD])     15.99
        Equity(67 [ADSK])     74.01
        Equity(76 [TAP])      97.33
        Name: close_price, dtype: float64
```

So it seems like this is what we are looking for. Above are the close prices for some of the stocks listed on the S&P500 on that day. For example, we now know that Apple's close price (AAPL) on January 3, 2017 was $115.84.

We will continue with our research project in part 2 of the MIRC tutorials.

Part 2 -- Wrangling the Data

In Part 1 we learned how to **initialize a pipeline**, use that pipeline to **pull basic price data**, and **read that data into a dictionary** for further analysis.

If we are trying to look at **sector momentum*, we will need** more data**, both in terms of** range **(rule of thumb is something like 10 years of data) and in terms of** what data **we need (for sector momentum we will need some sort of gauge of "success," one example being** earnings surprise **i.e. how much the company** beat/missed earnings** previously predicted by analysts).

Creating our Factors

We will take the pipeline function we defined earlier and modify it to match the needs listed above (using the documentation found here [https://www.quantopian.com/docs/data-reference/estimates_actuals (https://www.quantopian.com/docs/data-reference/estimates_actuals)](https://www.quantopian.com/docs/data-reference/estimates_actuals))

Note there are more packages that need to be downloaded for the next step (see the link). These will be added to the cell at the very top under the Part 2 comment

First, we must create our earnings surprise gauge using the pipeline (literally copied and pasted from the link):

In [10]:
```
# Slice the PeriodicConsensus and Actuals DataSetFamilies into DataSet
s. In this context,
# fq0_eps_cons is a DataSet containing consensus estimates data (e.g.
what analysts on Wall St and
# beyond believe the EPS of the company will be in the next quarter) a
bout EPS for the
# most recently reported fiscal quarter. fq0_eps_act is a DataSet cont
aining the actual
# reported EPS for the most recently reported quarter.
fq0_eps_cons = fe.PeriodicConsensus.slice('EPS', 'qf', 0)
fq0_eps_act = fe.Actuals.slice('EPS', 'qf', 0)

# Get the latest mean consensus EPS estimate for the last reported qua
rter.
fq0_eps_cons_mean = fq0_eps_cons.mean.latest

# Get the EPS value from the last reported quarter.
fq0_eps_act_value = fq0_eps_act.actual_value.latest

# Define a surprise factor to be the relative difference between the e
stimated and
# reported EPS.
fq0_surprise = (fq0_eps_act_value - fq0_eps_cons_mean) / fq0_eps_cons_
mean
```

Next we define a custom factor for sector (see link [https://www.quantopian.com/posts/figure-out-sector (https://www.quantopian.com/posts/figure-out-sector)](https://www.quantopian.com/posts/figure-out-sector)):

In [15]:
```
sector = Fundamentals.morningstar_sector_code.latest
```

Now that we have "made" our factors, we will plug it into the function that we declared in Part 1 and change a few things to match our goals.

In [53]:
```
# Add the surprise factor to the pipeline.
def make_pipeline():

    # Pipeline factors -- same as before
    close_price = USEquityPricing.close.latest # Close price of each s
tock
    market_cap = MarketCap() # Market cap of each stock

    # Pipeline Filters -- same as before
    QTU = QTradableStocksUS()
    top_500_market_cap = market_cap.top(500)
    QTU_top_500 = QTU & top_500_market_cap

    # Want to make sure that the stock is listed and tradeable on that
day
    # same as before
    has_pricing_data = close_price.notnull()

    # this is the only part that needs to be changed, add our new fact
or to the list
    return Pipeline(
        columns={
        'eps_surprise_factor': fq0_surprise,# New factor
        'sector': sector, # new factor
        'actuals': fq0_eps_act_value, # separate factor for actuals, w
ill see why later
        'close_price': close_price # Same as before
    },
    screen=QTU_top_500 & has_pricing_data)
```

In [54]:
```
# Want to have this in its own cell so it only needs to be run once
# Same as before, just to test to make sure this is what we are lookin
g for
pipeline_test = run_pipeline(make_pipeline(), '2017-1-1', '2017-1-5')
```

Pipeline Execution Time: 3.75 Seconds

In [55]:
```
pipeline_test.keys()
```

Out[55]:
```
Index(['actuals', 'close_price', 'eps_surprise_factor', 'sector'], d
type='object')
```

Let us now look at what our new eps factor looks like:

In [19]:
```
pipeline_test["eps_surprise_factor"]["2017-01-03"].head()
```

Out[19]:
```
Equity(24 [AAPL])     0.007769
Equity(62 [ABT])      0.014876
Equity(64 [GOLD])     0.191487
Equity(67 [ADSK])    -0.248384
Equity(76 [TAP])      0.007497
Name: eps_surprise_factor, dtype: float64
```

We can see that the estimate average for AAPL comes very close in this case to the actual realized value (estimates were off the actuals by less than a percent (I wonder if this is a reasonable assumption for all the estimates?)) vs, for example Autodesk (ADSK) where the estimate average was off by almost 25%.

And now we will look at what the sector factor looks like:

In [22]:
```
pipeline_test["sector"]["2017-01-03"].head()
```

Out[22]:
```
Equity(24 [AAPL])     311
Equity(62 [ABT])      206
Equity(64 [GOLD])     101
Equity(67 [ADSK])     311
Equity(76 [TAP])      205
Name: sector, dtype: int64
```

There seems to be a code that Morningstar/FactSet uses for their sectors, so I will create a dictionary to easily map the code to a string (see https://spotlightstockmarket.com/media/6145/morningstar.pdf (https://spotlightstockmarket.com/media/6145/morningstar.pdf)):

In [23]:
```
# Find the codes (GICS Level 1 usually has only 11)
pipeline_test["sector"]["2017-01-03"].unique()
```

Out[23]:
```
array([311, 206, 101, 205, 207, 103, 309, 310, 102, 308, 104])
```

In [56]:
```
sectorNumToString = {}
sectorNumToString[311] = "Technology"
sectorNumToString[206] = "Healthcare"
sectorNumToString[101] = "Basic Materials"
sectorNumToString[205] = "Consumer Staple" # aka Consumer Defensive
sectorNumToString[207] = "Utilities"
sectorNumToString[103] = "Financial Services"
sectorNumToString[309] = "Energy"
sectorNumToString[310] = "Industrials"
sectorNumToString[102] = "Consumer Cyclical"
sectorNumToString[308] = "Communication Services"
sectorNumToString[104] = "Real Estate" # iirc this one is tricky becau
se it was introduced later

# quick test
sectorNumToString[104]
```

Out[56]:
```
'Real Estate'
```

Finally, we will look at the actuals. We need this factor to find the first company to report in a given quarter. Since companies usually report on different days, often days or weeks after the official quarter end, we must find the first company to report for every quarter.

In [60]:
```
pipeline_test["actuals"]["2017-01-03"].head()
```

Out[60]:
```
Equity(24 [AAPL])     1.67
Equity(62 [ABT])      0.59
Equity(64 [GOLD])     0.24
Equity(67 [ADSK])    -0.18
Equity(76 [TAP])      1.03
Name: actuals, dtype: float64
```

Looking at this data, we can say that AAPL's reported Earnings per Share on January 3, 2017 was $1.67. The value itself is not important, it is the reporting date that is.

Reading, Consolidating, and Wrangling the Data

Now that we have created all the factors that we need and have sanity checked to make sure our data pull gives us what we want, we can now pull the whole range of data and "wrangle" it to format our algorithm later

In [61]:
```
# In a separate cell so we only need to run it once
data = run_pipeline(make_pipeline(), '2010-1-1', '2019-1-1')
```

Pipeline Execution Time: 48.62 Seconds

Next, I will create separate DataFrames for each to make wrangling easier:

In [69]:
```
data_close = pd.DataFrame(data["close_price"])
data_close = data_close.unstack() # removes the structure imposed by t
he pipeline
data_close.index.names = ["Date"] # creates a title for the date index
data_close.columns = data_close.columns.droplevel(0) # removes the ext
ra unneccessary column level
data_close.head()
```

Out[69]:

	Equity(2 [ARNC])	Equity(24 [AAPL])	Equity(53 [ABMD])	Equity(62 [ABT])	Equity(64 [GOLD])	Equity(67 [ADSK])	Equity(76 [TAP])	Equity([ADI
Date								
2010-01-04 00:00:00+00:00	16.12	210.84	NaN	53.95	39.40	NaN	45.14	3
2010-01-05 00:00:00+00:00	16.64	214.18	NaN	54.49	40.37	NaN	45.96	3
2010-01-06 00:00:00+00:00	16.12	214.38	NaN	53.99	40.88	NaN	45.34	3
2010-01-07 00:00:00+00:00	16.94	210.94	NaN	54.34	41.77	NaN	45.28	3
2010-01-08 00:00:00+00:00	16.59	210.58	NaN	54.78	41.18	NaN	44.60	3

5 rows × 595 columns

Do the same for the other factors:

In [71]:
```
data_surp = pd.DataFrame(data["eps_surprise_factor"]).unstack()
data_surp.index.names = ["Date"]
data_surp.columns = data_surp.columns.droplevel(0)
data_surp.head()
```

Out[71]:

	Equity(2 [ARNC])	Equity(24 [AAPL])	Equity(53 [ABMD])	Equity(62 [ABT])	Equity(64 [GOLD])	Equity(67 [ADSK])	Equity(76 [TAP])	Equit [AI
Date								
2010-01-04 00:00:00+00:00	-1.486871	0.238709	NaN	0.024963	1.267621	NaN	0.18213	0.0:
2010-01-05 00:00:00+00:00	-1.486871	0.238709	NaN	0.024963	1.267621	NaN	0.18213	0.0:
2010-01-06 00:00:00+00:00	-1.486871	0.238709	NaN	0.024963	1.267621	NaN	0.18213	0.0:
2010-01-07 00:00:00+00:00	-1.486871	0.238709	NaN	0.024963	1.267621	NaN	0.18213	0.0:
2010-01-08 00:00:00+00:00	-1.486871	0.238709	NaN	0.024963	1.267621	NaN	0.18213	0.0:

5 rows × 595 columns

The actuals dataframe will be a little different because we want to include the sector of the company in this dataframe to make later calculations easier.

In [111]:
```
data_actu = pd.DataFrame(data["actuals"]).unstack()
data_actu.index.names = ["Date"]
```

In [134]:
```
data_actu.head()
```

Out[134]:

	Equity(2 [ARNC])	Equity(24 [AAPL])	Equity(53 [ABMD])	Equity(62 [ABT])	Equity(64 [GOLD])	Equity(67 [ADSK])	Equity(76 [TAP])	Equity([ADI
Date								
2010-01-04 00:00:00+00:00	0.12	1.82	NaN	0.919999	0.54	NaN	1.14	
2010-01-05 00:00:00+00:00	0.12	1.82	NaN	0.919999	0.54	NaN	1.14	
2010-01-06 00:00:00+00:00	0.12	1.82	NaN	0.919999	0.54	NaN	1.14	
2010-01-07 00:00:00+00:00	0.12	1.82	NaN	0.919999	0.54	NaN	1.14	
2010-01-08 00:00:00+00:00	0.12	1.82	NaN	0.919999	0.54	NaN	1.14	

5 rows × 595 columns

We want to be able to **group the data above by sectors**, so we will take the sector data that we have pulled and turn it into a single-column dataframe where **index=stock/company name and value=sector**.

In [128]:
```
data_sector = pd.DataFrame(data["sector"]).unstack()
data_sector = data_sector.bfill() # Backwards fill so each company's s
ector code is the first row
data_sector.columns = data_sector.columns.droplevel(0)
data_sector = data_sector.T["2010-01-04"]
```

In [129]:
```
data_sector.head()
```

Out[129]:
```
Equity(2 [ARNC])     101.0
Equity(24 [AAPL])    311.0
Equity(53 [ABMD])    206.0
Equity(62 [ABT])     206.0
Equity(64 [GOLD])    101.0
Name: 2010-01-04 00:00:00+00:00, dtype: float64
```

In [131]:
```
for x in data_sector.index:
    data_sector[x] = sectorNumToString[data_sector[x]]
```

In [135]:
```
data_sector.head()
```

Out[135]:
```
Equity(2 [ARNC])     Basic Materials
Equity(24 [AAPL])         Technology
Equity(53 [ABMD])         Healthcare
Equity(62 [ABT])          Healthcare
Equity(64 [GOLD])    Basic Materials
Name: 2010-01-04 00:00:00+00:00, dtype: object
```

I now realize I've done a lot here, so I will break for today and hopefully continue tomorrow.