Quant Trading Project

- Anthony Puffenbarger (12338832)
- Cole Stecyk (12333742)
- Jacob Sprecher (12327719)
- Kelvin Wang (12283999)

1. Introduction

Why study ETF?

An ETF could be defined as "an investment product that allows an investor to buy and sell shares in a single security that represents a fractional ownership of a portfolio of securities."

1. **Liquidity**: ETFs traded like stocks, investors purchased and sold shares in ETFs (fund shares) on secondary securities markets through broker-dealers throughout the day

2. **Lower Trading costs**: ETFs offered investors inexpensive access to a diversified portfolio of securities through index investing.

1. **Transparency**: The SEC required ETFs to disclose all portfolio holdings on a daily basis. In comparison, mutual funds generally disclosed their holdings on a quarterly basis, with a lag of up to 60 days after the balance sheet date. Hedge funds were generally more opaque. ETFs have a more efficient primary market mechanism

1. **Creation Redemption Mechanism and Authorized Participants**:

   Since ETFs, trade on an exchange there are numerous market making firms quoting prices to provide liquidity. However, ETFs also contract with Authorized Participants (APs) to efficiently create large blocks of new shares (creation units) or redeem existing ones through a process known as the Creation/Redemption mechanism. This can be thought of as following steps: 1. At ETF inception, the fund publishes a basket of securities that constitutes a creation unit 2. An AP acquires in the open market and delivers them to the ETF in-kind 3. The ETF delivers newly created shares to the AP for them to sell 4. Markets continue functioning without APs until there is a large buy/sell order imbalance 5. If buys exceed sells, market makers contract with APs to facilitate another creation 6. If sells exceed buys, APs place a redemption order with the fund manager who cancels existing shares and delivers securities out of the portfolio to the AP (to raise cash) But APs are not actually dormant until liquidity is needed. Since the ETF is still a regulated, open-end investment company, it strikes an official NAV per share at the end of the day that reflects a fair value estimate of its underlying holdings. *Additionally*, the fund disseminates an **iNAV** (intraday net asset value) every 15 seconds that reflects what an official NAV would be at the most recent prices of underlying. Often the price in the secondary market will deviate from this value (NAV Premium/Discount) and create a potential source of arbitrage profit. This phenomenon and specifically its information value motivates our strategy.

NAV Premium/Discount and Authorized Participant Arbitrage

drawing

Suppose an S&P 500 ETF experiences a large volume of buy order flow that pushes the price well above the most recently published iNAV. Anyone making markets in the ETF and its constituents could potentially profit by selling the overvalued ETF shares, buying the basket of underlying securities, and hedging any residual risk. APs can further reduce risk/costs by closing the trade with an ETF creation in the primary market at NAV. If realized net purchase prices for the basket closely track prices in the ETF NAV, the AP may realize a true arbitrage profit. Therefore, the NAV Premium/Discount should be bounded by the costs of an AP to conduct this arbitrage.

S&P 500 constituents are highly liquid and easy to trade with high price transparency, so any actual arbitrage opportunity should quickly disappear as it is exploited. But what about ETFs with less **liquidity** and less **on-market execution price certainty** of the underlying assets (such as **high yield corporate bonds**)?

The below figure compares the per minute iNAV Premium/Discounts on 2/25/2022 for the SPY S&P500 ETF (SPY) vs. the iShares iBoxx High Yield Corporate Bond ETF (HYG). SPY never closes more than $1‰$ away from it's NAV, however HYG can close as much 1% or more above or below its NAV. Additionally, it appears as though NAV Premium/Discount can persist. The existence of these deviations from NAV does not alone represent an arbitrage opportunity and may be justified given the dynamics of the ETF and its underlying assets.

title title

Drivers of the ETF Prices

Given the primary and secondary market dynamics for ETFs, the following factors (primarily) influence the ETF prices:

1. **Underlying securities(fair value of the ETF)**
   - What price is in the NAV for a security and how much is in the basket?
   - How do the securities trade in the open-market (minimum trade sizes, OTC, etc.)

1. **Level of ETF supply and demand in the secondary market**
   - More buyer initiated trades (positive flow) may indicate the price will move up.
   - **Quote book** of the ETF suggests the supply and demand for the ETF
   - ETFs may see NAV Premium/Discounts as prices get bid up or pushed down.
   - APs may conduct primary market transactions to balance order flow

1. **Creation/Redemption mechanism costs**
   - Some fund managers charge explicit fees (although these are typically small)
   - Underlying asset prices in the NAV vs. prices at which the APs transact may be different

1. **Level of underlying asset market volatility and liquidity**
   - Exactly how confident is the AP that it can transact in the underlying securities on the open market at prices equal to or better than NAV prices?

The previous figure of SPY vs. HYG NAV Premium/Discounts generalizes for ETFs on assets with high standardization and liquidity vs. other assets. Larger, more persistent, and more volatile NAV Premiums/Discounts tend to exist where the underlying assets are characterized by less liquidity and more price uncertainty.

In this framework, we find US-listed **corporate bond ETFs** to be a particularly interesting area of study and especially in the year **2020** with the pandemic involving a lot of illiquidity.

Explicit NAV Premium/Discount Model for Bond ETFs

Some previous research exists suggesting evidence of the ability of NAV Premium/Discount and ETF primary market flows to predict ETF returns. Brown, Davies, and Ringgenberg in "ETF Arbitrage, Non-Fundamental Demand, and Return Predictability" find changes in **ETF shares outstanding** to be negatively correlated with **future ETF returns**. They also find returns from conditioning on NAV Premium/Discount to be positive but not statistically different from zero.

We intend to study similar dynamics in our project however our measurements of ETF flows and NAV Premium/Discount will be slightly different in higher frequency.

Model features

- **flow**
  We will measure ETF trade flows in the secondary market with actual trade level information from WRDS.

  As trade flow becomes more imbalanced, APs are more likely to create or redeem shares in the primary market, and those costs are more likely to be reflected in the NAV Premium/Discount. Flow is an interesting as it is a potential indicator of forward returns over the short term. It uses the order book to measure the volume of trades that are seller initiated vs buyer initiated. If more trades are seller initiated, this means that the market participants that are selling shares are more willing to cross the bid-ask spread to initiate a trade compared to the market participants that are buying shares. This imbalance will potentially drive the price lower if flow is negative and will potentially drive the price higher if flow is positive.

- **iNAV Premium Discount**
  We will measure intraday NAV Premium/Discount using iNAV data published in Bloomberg. This information is generally available every 15 seconds, but our frequency will be on the order of minutes to limit the volume of data.

- **average bid ask price spread**: measure of illiquidity

- **Size Imbalance**
  In addition, we will look at size imbalance. Size imbalance is a measure of volume that is being offered at the best ask price compared to the volume willing to be purchased at the best bid price. If size imbalance is positive, then (A) quite a few people are trying to buy and (B) it would not take many of them crossing the book to eliminate all the remaining offer orders, thereby rising it one level that will potentially drive the price higher until the market returns to equilibrium.

We plan to study the performance of long-only portfolios formed on each signal individually as well as the cross-section of flow and NAV Premium/Discount.

Feature Variables Calculation Formulas

In addition to the column already present i the trade flow data we calculated identified the following as possible additional important features:

- **flow** (sum and ewma) = volume of buyer initiated trades - volume of seller initiated trades

$$V_B - V_S$$

  Flow is an important measure of buyer/seller activity at a given point in time. If there has been heavy buyer initiated trades or heavy seller initiated trades in the past time interval, flow will reflect this.

- **dollar flow**: Similar to flow only in dollar amounts

- **average bid ask price spread**: measure of illiquidity

$$\frac{1}{n}\sum_{t=1}^{n}\frac{(NBO_t - NBB_t)}{Price_t}$$

- **exponential weighted moving average volatility**:

  Assuming log return has mean equals 0:

$$\sqrt{EWMA\{log(\frac{P_t}{P_t-\tau})^2]\}}$$

- **size volume**:

$$\sum_{t=T-\tau}^{T} tradesize_t$$

- **size imbalance**: relative difference between best bid size and best ask size

$$\frac{Size_{bestbid} - Size_{bestoffer}}{Size_{bestbid} + Size_{bestoffer}}$$

- **forward return**(simple): including delay L of 50 milliseconds

$$\frac{P_{t+\tau+L} - P_{t+L}}{P_{t+L}}$$

- **vwap forward return**: To avoid the tick-sensitivity problem, we calculated the vwap forward return over the simple that calculated a volume weighted average price = exponential weighted mean of foward return with half life t

$$\frac{VWAP_{t+\tau+L} - VWAP_{t+L}}{VWAP_{t+L}}$$

Each of the features above was calculated with in τ interval using a 1, 2, 3, 5, 15, 20, 25, 30, 45, and 60 mintue interval.

Forward Return is calculated in 3 different ways:

- no transaction costs are assumed and trade price is used as the entry and exit
- no transaction costs are explicitly assumed and the vwap price is used as the entry and exit
- transction costs are assumed using the NBB and NBO
  1. long positions are assumed to be opened at the ask and closed at the bid.
  2. short position are assumed to be opened at the bid and closed at the ask.(long_rtn, short_rtn)

Data

Import and Functions

In [1]:
```python
import pandas as pd
import numpy as np
import requests
import io
import pickle
import statsmodels.api as sm
import warnings
import matplotlib.pyplot as plt
import calendar
import matplotlib.dates as mdates
from itertools import islice
from matplotlib.backends.backend_pdf import PdfPages
from matplotlib.dates import HourLocator, DateFormatter
import matplotlib.ticker as tick
warnings.filterwarnings('ignore')
import datetime
from scipy.stats import norm
from collections import Counter
import seaborn as sns
from sklearn.metrics import mean_squared_error
from util.strategy_analysis import FeatureSelection
from util.strategy_analysis import ExponentialWeightedReg
from util.nav_util import *

from IPython.core.display import HTML
HTML("""
<style>
.output_png {
    display: table-cell;
    text-align: center;
    vertical-align: middle;
}
</style>
""")
```

Out[1]:

Load and Clean Data

Pricing and iNAV Data

Read in raw data exported from Bloomberg. The raw data is 5 minute price and iNAV data, as well as daily coporate actions (dividends and splits) for 8 etfs: BKLN, HYG, HYGH, JNK, PFF, PGX, SPHY, SRLN. The export from Bloomberg was done in yearly incrememts from Sep 2019 - Dec 2020. The following code assumes the raw data .xlsx files have already been saved locally. ETF price data, iNav, and coporate actions are loaded seperatly and then combined into one cohesive dataframe.

In [2]:
```python
pricesPath = "HYG_JNK_Prices/"
navPath = "HYG_JNK_NAV/"
corpActsPath = "HYG_JNK_Corporate_Actions/"
```

In [3]:
```python
# Load HYG, JNK Prices
hygPriceDf, jnkPriceDf = loadData(pricesPath)
# format dfs
hygPriceDf, jnkPriceDf = cleanPriceDF(hygPriceDf), cleanPriceDF(jnkPriceDf)
```

In [4]:
```python
# Load HYG, JNK iNAVs
hygNavDf, jnkNavDf = loadData(navPath)
# format dfs
hygNavDf, jnkNavDf = cleanNavDF(hygNavDf), cleanNavDF(jnkNavDf)
```

The pricing data for JNK and HYG are already split adjusted so we just backwards adjust the nav data (which isn't split adjusted). Dividends are noted on the first candle (9:30) on their payout day.

In [5]:
```python
# Load HYG, JNK dividends and splits
jnkNavDf = adjCorpActions(corpActsPath+"jnk_corporate_actions.xlsx", jnkNavDf)
hygNavDf = adjCorpActions(corpActsPath+"hyg_corporate_actions.xlsx", hygNavDf)
```

The pricing, nav, and corporate actions dataframes are combined.

In [6]:
```python
# drop duplicate indexes
jnkPriceDf = jnkPriceDf[~jnkPriceDf.index.duplicated(keep='last')]
jnkNavDf = jnkNavDf[~jnkNavDf.index.duplicated(keep='last')]
hygPriceDf = hygPriceDf[~hygPriceDf.index.duplicated(keep='last')]
hygNavDf = hygNavDf[~hygNavDf.index.duplicated(keep='last')]

# combine dataframes
hygDf = pd.concat([hygPriceDf, hygNavDf], axis=1)
jnkDf = pd.concat([jnkPriceDf, jnkNavDf], axis=1)
```

In [7]:
```python
hygDf.head()
```

Out[7]:

	close_etf	net_chg_etf	open_etf	high_etf	low_etf	tick_cnt_etf	volume_etf	close_nav	net_chg_nav	open_nav	high_nav	low_nav	ti
2017-02-17 09:30:00	87.5850	-0.0950	87.590	87.6000	87.51	2471.0	971205	87.44	-0.06	87.45	87.45	87.44	
2017-02-17 10:00:00	87.5950	0.0100	87.585	87.6200	87.58	1880.0	349642	87.45	0.01	87.44	87.45	87.43	
2017-02-17 10:30:00	87.5875	-0.0075	87.595	87.6077	87.58	1004.0	187134	87.44	-0.01	87.45	87.45	87.44	
2017-02-17 11:00:00	87.5850	-0.0025	87.590	87.6100	87.57	993.0	247301	87.46	0.02	87.44	87.46	87.44	
2017-02-17 11:30:00	87.5900	0.0050	87.585	87.5900	87.56	620.0	146345	87.46	0.00	87.46	87.46	87.46	

In [8]:
```python
jnkDf.head()
```

Out[8]:

	close_etf	net_chg_etf	open_etf	high_etf	low_etf	tick_cnt_etf	volume_etf	close_nav	net_chg_nav	open_nav	high_nav	low_
2017-02-17 09:30:00	110.700	-0.120	110.730	110.73	110.610	658.0	91255	110.524011	-0.0759	110.532711	110.533611	110.522
2017-02-17 10:00:00	110.700	0.000	110.700	110.73	110.670	593.0	91427	110.531211	0.0072	110.524311	110.531211	110.522
2017-02-17 10:30:00	110.679	-0.021	110.700	110.73	110.677	663.0	201649	110.524611	-0.0066	110.531211	110.531211	110.523
2017-02-17 11:00:00	110.670	-0.009	110.685	110.70	110.670	197.0	28538	110.538711	0.0141	110.524611	110.538711	110.524
2017-02-17 11:30:00	110.687	0.017	110.685	110.70	110.670	210.0	21321	110.542311	0.0036	110.538711	110.544711	110.537

Order Book Data

In [9]:
```python
import wrds
# connecting to wrds
db = wrds.Connection(wrds_username='cstecyk')
HYG_total=db.raw_sql("""Select date, time_m, bid, bidsiz, ask, asksiz,qu_cancel from taqmsec.cqm_20200103
                where sym_root='HYG'
                limit 1
               """)
```

```
Loading library list...
Done
```

WRDS privides milliseconds order book and trade book data each trading day. We fetch 12 fields from the Quote dataset.The quote data is structured as shown below:

> $DATE$: Date of quote (DATE) $TIME_M$: Time of Trade or Quote with nanoseconds (HHMMSSXXX) (TIME_M) SYM_ROOT: Security symbol root (SYM_ROOT) SYM_SUFFIX: Security symbol suffix (SYM_SUFFIX) EX: Exchange that issued the quote (EX) BID: double Bid price (BID) $BIDSIZ$: double Bid size in units of trade (BIDSIZ) ASK: double Ask price (ASK) $ASKSIZ$: double Ask size in units of trade (ASKSIZ) QU_COND: string Condition of quote issued (QU_COND) QU_SEQNUM: double Quote Sequence Number (QU_SEQNUM) $NATBBO_IND$: string National BBO Indicator (NATBBO_IND)

In [10]:
```python
order_book_samp = pd.read_csv('JNK-2020-03-02-sample.csv')
order_book_samp['DATE'] = pd.to_datetime(order_book_samp['DATE'], format = '%Y%m%d')
order_book_samp['DateTime'] = order_book_samp['DATE'].astype(str)+' '+order_book_samp['TIME_M'].astype(str)
order_book_samp['DateTime'] = order_book_samp['DateTime'].apply(pd.Timestamp)
order_book_samp = order_book_samp.set_index('DateTime')
order_book_samp.head()
```

Out[10]:

DateTime	DATE	TIME_M	BID	BIDSIZ	ASK	ASKSIZ	QU_COND	QU_SEQNUM	NATBBO_IND	QU_CANCEL	SYM_ROOT
2020-03-02 04:00:00.008130813	2020-03-02	4:00:00.008130813	0.00	0	115.0	1	R	601	G	NaN	JNK
2020-03-02 04:00:00.008146336	2020-03-02	4:00:00.008146336	100.93	1	115.0	1	R	701	G	NaN	JNK
2020-03-02 04:00:00.008146336	2020-03-02	4:00:00.008146336	103.40	1	115.0	1	R	702	G	NaN	JNK
2020-03-02 04:00:00.008146336	2020-03-02	4:00:00.008146336	106.60	1	115.0	1	R	703	G	NaN	JNK
2020-03-02 04:00:00.008146336	2020-03-02	4:00:00.008146336	106.60	1	108.6	1	R	704	G	NaN	JNK

Trade Book Data

WRDS also provides millisencond trade book data each day.

In [11]:
```python
trade_book_samp=pd.read_csv('JNK trade 2020-03-02.csv')
```

In [12]:
```python
trade_book_samp['DATE'] = pd.to_datetime(trade_book_samp['DATE'], format = '%Y%m%d')
trade_book_samp['DateTime'] = trade_book_samp['DATE'].astype(str)+' '+trade_book_samp['TIME_M'].astype(str)
trade_book_samp['DateTime'] = trade_book_samp['DateTime'].apply(pd.Timestamp)
trade_book_samp = trade_book_samp.set_index('DateTime')
trade_book_samp.head()
```

Out[12]:

DateTime	DATE	TIME_M	EX	SYM_ROOT	SYM_SUFFIX	TR_SCOND	SIZE	PRICE	TR_CORR	TR_SEQNUM	TR_ID
2020-03-02 04:00:00.008137321	2020-03-02	4:00:00.008137321	P	JNK	NaN	T	500	107.09	0	12501	52983525027894
2020-03-02 04:33:45.205750338	2020-03-02	4:33:45.205750338	P	JNK	NaN	T	100	106.85	0	17701	52983525028170
2020-03-02 04:36:25.953104090	2020-03-02	4:36:25.953104090	P	JNK	NaN	T	400	106.82	0	18101	52983525028173
2020-03-02 04:36:25.953127230	2020-03-02	4:36:25.953127230	P	JNK	NaN	T	100	106.80	0	18201	52983525028174
2020-03-02 06:00:09.350430160	2020-03-02	6:00:09.350430160	P	JNK	NaN	TI	5	106.52	0	35001	52983525028940

Trade Matching: Lee and Ready Algorithm

The raw trade book data from WRDS doesn't directly incidate the side of the trade (bid, aks or mid). To determine the side we use the Lee and Ready algorithm. This algorithm is comprised of two parts:

1. **Quote test** compare current trade with the immediate quote's mid-point
   - if trade price > (quote+bid)/2 then buyer-initiated
   - if trade price < (quote+bid)/2 then seller-initiated
   - if trade price = (quote+bid)/2 undecided

   Originally, Lee and Ready (1991) applies a so-called 5-second rule to adjust potential delay of the reporting time for trades. However, the latency in quote issuance and trade execution has been decreasing over time, and more recent research has recommended the use of 2 seconds or 0 seconds lags instead.

   Here we use 0 second delay.

   The SAS environment on WRDS proved to be a much more efficent method for querying and merging data. This SAS code was devloped to quickly query and caclulate trade side:

   %let tables=wct_20190105;
   DATA temp;
   set taqmsec.&tables : ;
   where sym_root in ('HYG','JNK','BKLN','SRLN','PFF','PGX', 'HYGH', 'SPHY') and price>0 and size>0;
   LeeReady=sign(Price-(NBO+NBB)/2);
   run;

2. **Tick test**:

   For the undecided trades we run a second test:- compare the trade price with the previous price
   - if trade price > previous trade price then buyer-initiated
   - if trade price < previous trade price then seller-initiated
   - if trade price = previous trade price then repeat the tick test with an additoinal lag

   We mark the trades that we weren't able to assign a side to after applying two lags with a 0. An example of how we applied the tick test to the trade data is shown below.

   Lee Ready sign is 1 when it is buyer initiated.

   Lee Ready sign is -1 when it is seller initiated

In [13]:
```python
computeLeeReady("trade2019-2020subsets/HYGH.csv").head()
```

Out[13]:

	DATE	TIME_M	SYM_ROOT	SYM_SUFFIX	type	EX	TR_SCOND	SIZE	PRICE	TR_SEQNUM	QTime	NBO	NBB
0	20191001	9:30:00.127748718	HYGH	NaN	T	P	I	85	88.3100	157201	9:29:49.026170117	88.38	88.30
1	20191001	9:30:00.127786153	HYGH	NaN	T	P	Q	85	88.3100	157301	9:29:49.026170117	88.38	88.30
2	20191001	10:00:07.240263571	HYGH	NaN	T	P	I	3	88.1900	3963101	10:00:06.892941766	88.20	88.10
3	20191001	10:18:42.689757089	HYGH	NaN	T	D	I	1	87.9108	7906601	10:18:33.353313771	87.95	87.88
4	20191001	10:23:10.964038036	HYGH	NaN	T	D	I	2	87.9400	8605801	10:23:05.711584539	87.94	87.87

Feature Selection

Altogether, we have 79 features. This may be too much for regression and most of them are correlated to each other. Therefore, we perform 2 feature selection process under **two** criteria:

- `Mutual Infromation Ratio`
  Mutual information between two random variables is a non-negative value, which measures the dependency between the variables. It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency. We estimate mutual information for a continuous target variable.
- `Lasso Regression`
  Lasso regression performs L1 regularization, which adds a penalty equal to the absolute value of the magnitude of coefficients. This type of regularization can result in sparse models with few coefficients; Some coefficients can become zero and eliminated from the model.

For feature selection, we group similar categories of variables together and regress 5 minute forward returns on the data using the lasso model to identify potentially significant variables. We select the top 3 variables from each group by mutual information criteria. Shown below are these groups and importance scores for the top 3 variables in each category for HYG. The top rated features were very similar across all etfs.

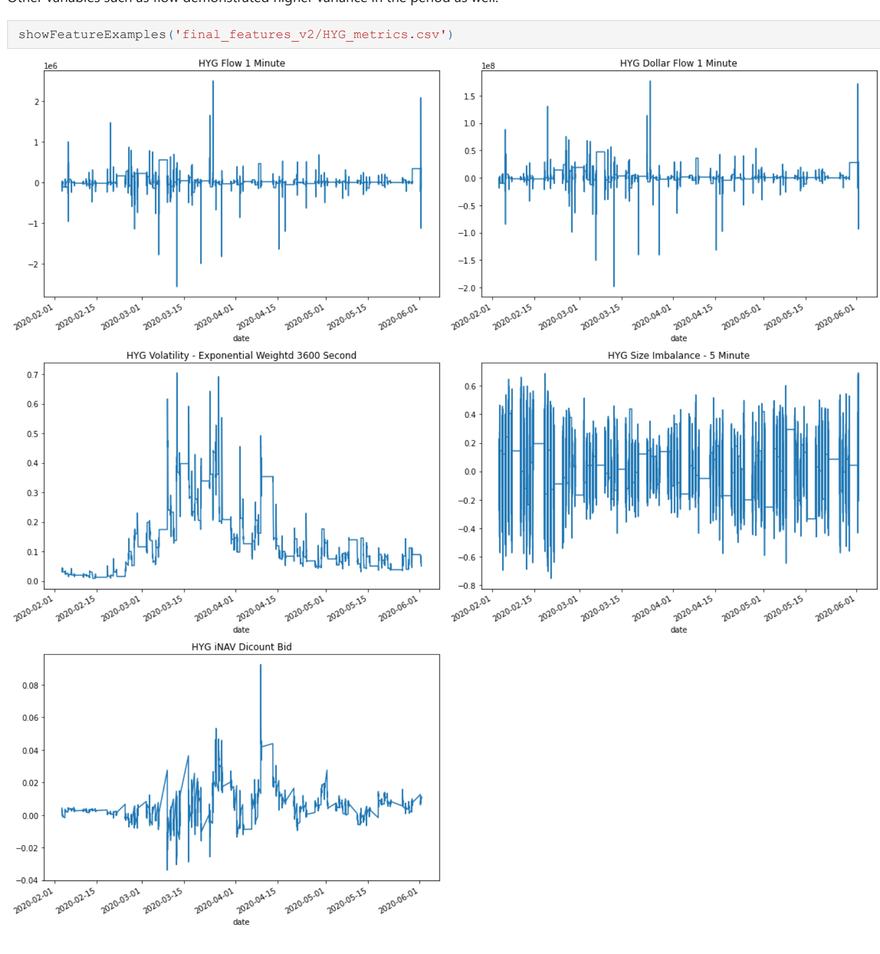
Following this process we selected 5 features for regression:

- flow_1min
- dollar_flow_1min
- ewm_vol_3600s
- size_imbalance_5min
- nav_discount_bid

To add a quick note, here we can observe how the high frequency variables change across time. Most notably, our volatility measure Exponential Weighted 3600 Second is quite a good measure of the 2020. In 2020 March, it shows a significant **spike** of volatility.

Other variables such as flow demonstrated higher variance in the period as well.

In [14]:
```python
showFeatureExamples('final_features_v2/HYG_metrics.csv')
```



Mutual Information Regression for Feature Selection

In [15]:
```python
feature_selection = FeatureSelection("HYG")
```

In [16]:
```python
feature_selection.mutual_information_selection()
```

	importance
dollar_flow_1min	0.702962
flow_1min	0.699934
flow_2min	0.693063

	importance
ewm_vol_3600s	0.973798
ewm_vol_1800s	0.969190
ewm_vol_2700s	0.968501

	importance
size_imbalance_4min	0.702879
size_imbalance_10min	0.699468
size_imbalance_15min	0.699113

	importance
bid_ask_price_1min	1.155059
bid_ask_price_2min	0.965098
bid_ask_price_5min	0.812138

	importance
nav_discount_bid	0.94232

Lasso Regression for Feature Selection

In [17]:
```python
feature_selection.lasso_selection()
```

	importance
flow_5min	0.000011
flow_4min	0.000001
flow_1min	0.000000

	importance
ewm_vol_60s	0.000233
ewm_vol_120s	0.000107
ewm_vol_3600s	0.000087

	importance
size_imbalance_1min	0.000031
size_imbalance_4min	0.000022
size_imbalance_5min	0.000015

	importance
bid_ask_5min	0.000654
bid_ask_price_10min	0.000645
bid_ask_price_5min	0.000565

Matching iNAV to Trade Data

There is a frequency **mismatch** in our data sources as iNAV is updated at best every 15 seconds. However, trade data and order book data is almost continuous flow up to nanoseconds.

In our model, we choose an iNAV frequency of every 5 minutes and a corresponding intraday trading interval of 5 minutes as well. We match the high frequency feature calculation to the low frequency iNAV update.

As showed below, at time **T**, the iNAV and the filtration of trade/quote book data is generated as of time T. We receive these information at **T+25ms** with a network delay and our model outputs signal that places trades in the market at **T+50ms** including network delay.

how did we match iNAV to trade data

title

Preliminary Analysis/Visualization

HYG Premium Discount Monthly Trends

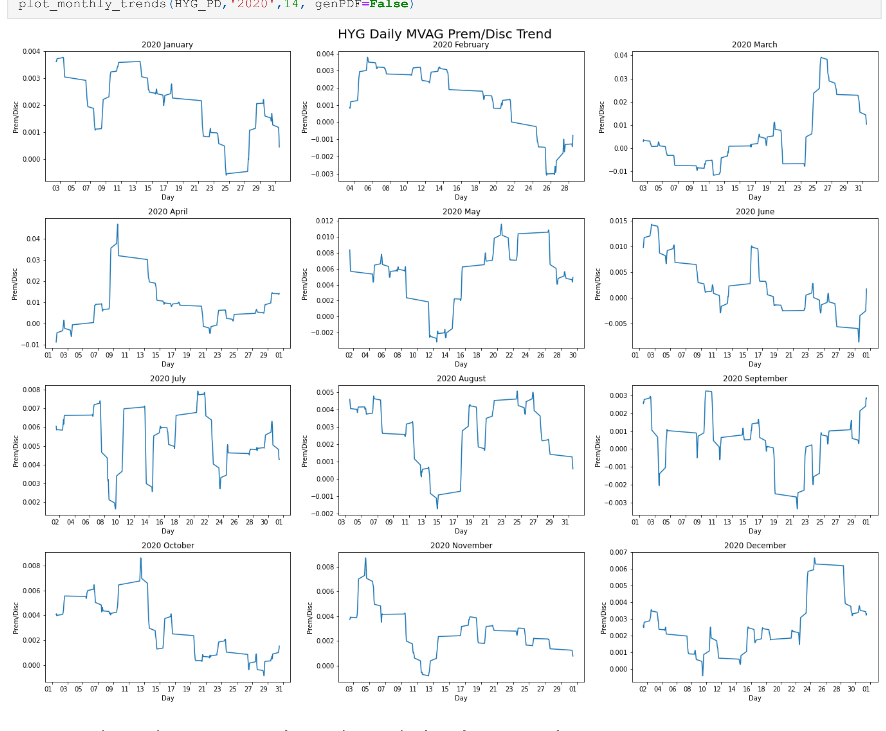
In [18]:
```python
token ='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'  # github token
# load hyg price, nav data
HYG_url = 'https://github.com/jsprecher/trading-project/blob/main/HYG_PRICE_NAV.pkl?raw=true'
HYG_pkl = io.BytesIO(github(HYG_url,token).content)
HYG = pickle.load(HYG_pkl)

# load jnk price, nav data
JNK_url = 'https://github.com/jsprecher/trading-project/blob/main/JNK_PRICE_NAV.pkl?raw=true'
JNK_pkl = io.BytesIO(github(JNK_url,token).content)
JNK = pickle.load(JNK_pkl)
```

The metric we will use heavily for our project is premium discount. This is calculated by:

$$PremiumDiscount = \frac{Price_{etf} - Price_{nav}}{Price_{nav}}$$

In [19]:
```python
# calculate premium discount
HYG_PD = ((HYG['close_etf'] - HYG['close_nav'])/HYG['close_nav']).dropna()
JNK_PD = ((JNK['close_etf'] - JNK['close_nav'])/JNK['close_nav']).dropna()
HYG_PD.name = 'HYG'
JNK_PD.name = 'JNK'
```

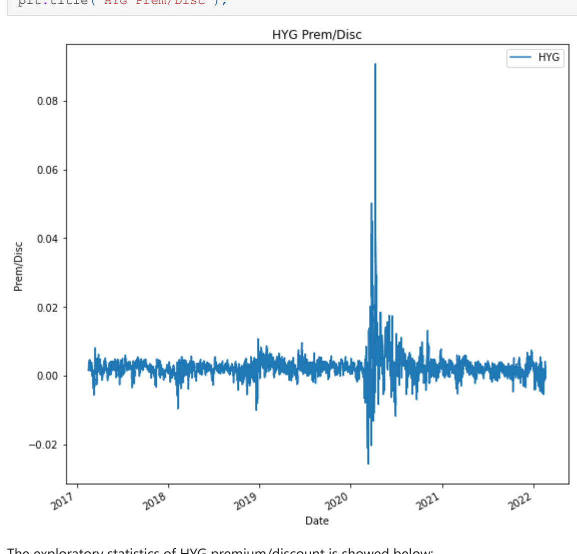
In [20]:
```python
plot_monthly_trends(HYG_PD,'2020',14, genPDF=False)
```



HYG Premium Discount Over the Entire Period (Feb2017 - Feb2022)

Year 2020 has extremely high premium discount as showed in the below graph, we are interested to know whether we can exploit from the illiquidity in the period

In [21]:
```python
HYG_PD = HYG_PD.to_frame()
HYG_PD.plot(figsize=(9,9));
plt.xlabel("Date")
plt.ylabel("Prem/Disc")
plt.title("HYG Prem/Disc");
```



The exploratory statistics of HYG premium/discount is showed below:

In [22]:
```python
Metrics(HYG_PD)
```

Out[22]:

	Std	Min	Q .05	Mean	Median	Q .95	Max	Samples Cnt
HYG	0.003326	-0.025761	-0.001166	0.002293	0.002176	0.005714	0.090814	16353

HYG, JNK NAV Premium Discount by Minute (Intraday)

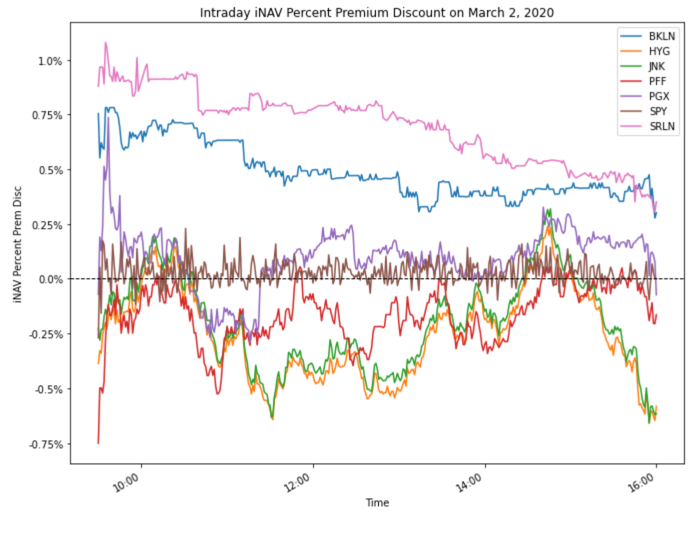
We also have access to shorter 1 minture time frame price, NAV data.

We see that except SPY ETF which quickly self-corrects, all the fixed income ETFs have persistent premiums or discounts on 2/25.In addition, **HYG**(orange)/**JNK**(green) follows similar premium/discount patterns as they track similar indices.

In [25]:
```python
hyg_intraday = pd.read_csv('hyg_discounts_1day.csv', index_col = 'date', parse_dates=True)
hyg_intraday = hyg_intraday.rename({'px_close':'Price','discount':'HYG Price - iNAV'},axis=1)

jnk_intraday = pd.read_csv('jnk_discounts_1day.csv', index_col = 'date', parse_dates=True)
jnk_intraday = jnk_intraday.rename({'px_close':'Price','discount':'JNK Price - iNAV','date':'time'},axis=1)
```

In [26]:
```python
minute_discounts_data = pd.read_csv('minute_series/discounts_1minute.csv')
minute_discounts_data['date'] = minute_discounts_data['date'].apply(pd.Timestamp)
minute_discounts_data.set_index('date',inplace=True)
minute_discounts_data["pct_discount"] = minute_discounts_data.discount/minute_discounts_data.iNAV  # add pct di
```

In [27]:
```python
iNavMarchPremiumDiscountPlot(minute_discounts_data)
```

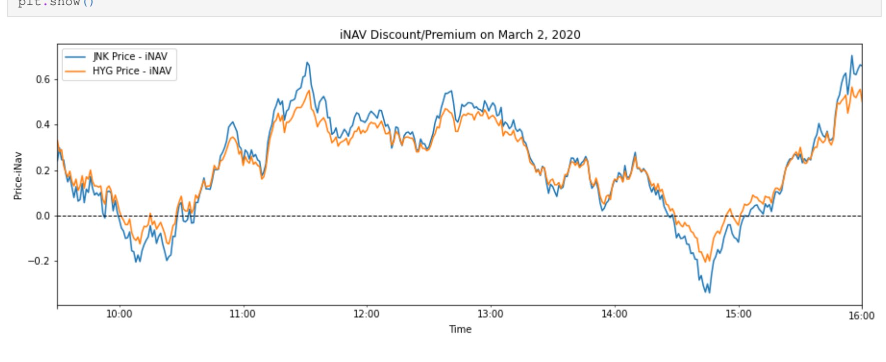


How does the premium discount of two ETFs with similar components move?

The first graph is the price and iNAV of JNK.

As we see below in the second graph, as the HYG/JNK both track US High Yield Coroporate Bond Indices, their premium/discount performs very similarly and comove with each other very closely.This pattern may be used to discover some trading strategies.

In [29]:
```
hy_discounts = jnk_intraday[['JNK Price - iNAV']].join(hyg_intraday[['HYG Price - iNAV']])
hy_discounts = hy_discounts * -1 # fix misslabeling
hy_discounts['2020-03-02'].plot(title = 'iNAV Discount/Premium on March 2, 2020', figsize=(15,5))
plt.xlabel('Time')
plt.ylabel('Price-iNav')
plt.axhline(y=0, color='black', linestyle='--', linewidth=1)  # add dashed line at 0
plt.show()
```



Model

Exponential Weighted Regression with daily update

After consideration, we moved on to exponential weighted regression or discounted least square regression that has substantial benefits

1. **No train test split - Update model each day with new information**

   In some sense, splitting the data to a train period to train the model and predict the model in a separate testing period is ignoring a lot of information.

   First of all, it assumes a constant model within the entire 6 months test period. However, new information arrived every day and assuming the same model for 6 months does not allow us to deal with extreme market events in 2020.

   We **update** the model **each day after the market closes** with today's new trade data. Applying the Sherman-Morrison inversion formula

   On each day after market close, we update P

$$P_{new} = \frac{1}{\lambda}(P - Px(\lambda + x^*Px)^{-1}x^*P$$

   Our new coefficients are then

$$\begin{aligned}\beta_{new} &= \beta + Px(\lambda + x^*Px)^{-1}(y - x^*\beta) \\ &= \beta + Pxf\lambda^{-1}h\end{aligned}$$

2. **Give more weight to recent data**

   Second, it assumes the data on recent and past have similar information. But clearly more recent data carries much more information. This is why we moved onto exponential weighted regression, also called discounted least-squares regression.

1. **Avoid look ahead bias as we find the best coefficients**

   We use the model to predict and output strategy signal in the next day. This way as we are doing a daily update model, we are not choosing the best coeffient over the whole period and act on previous dates.

In [ ]:

Model Parameter Tuning

In discounted least square regression, although we do not define a BoxCox window, there is the λ discount factor with which we discounted past data. So first, we should choose a best discount factor for the model.

Parameter Tuning on HYG

Here we did a grid search of discount factor from 0.2-0.9 and calculate the corresponding MSE for the prediction.

We can see that with different decay factors, MSE do not vary much. But the performance drops as we choose extremely fast decay rate like 0.3, 0.2. Since there is no material difference, we assume a factor of 0.7 in the following analysis.

In [15]:
```
model = ExponentialWeightedReg("HYG")
ks = [.2, .3, .4, .5, .6, .7, .8, .9]
MSE = pd.DataFrame(index=ks, columns=["MSE"])
for k in ks:
    model.train(['flow_1min','dollar_flow_1min','ewm_vol_3600s','size_imbalance_5min','nav_discount_bid'],'fwd_
    pred = model.estimate
    actual = model.data["2020-01-01":]['fwd_rtn_5min']
    MSE.loc[k, "MSE"] = mean_squared_error(actual, pred)

MSE.sort_values(by="MSE", ascending = True)
```

Out[15]:

	MSE
0.9	0.000001
0.8	0.000001
0.7	0.000001
0.6	0.000001
0.5	0.000001
0.4	0.000001
0.3	0.000002
0.2	0.000002

Signal Transformation

After defining the models, we need to transform our estimate or prediction into signals. We define

$$Position_t = \begin{cases} 1 & \text{go long if predicted return is higher than threshhold j} \\ -1 & \text{go short if predicted return is lower than threshhold -j} \\ 0 & \text{do not take position otherwise} \end{cases}$$

Risk Control

We employ a quite simple risk control to each ETF intraday trading strategy

1. Stop loss at 2% per day at a strategy level which will close positions and no more trading intraday

2. Maximum strategy drawdown limited to 5%

Strategy Backtesting with Different Thresholds (j)

Long only buy one share

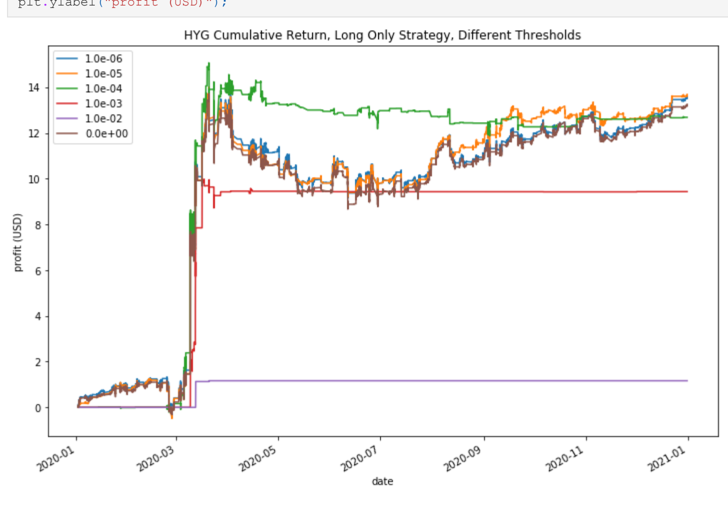
We start from a simple strategy which is **long only**, **ignoring transaction costs** and buying **one** share for every buy signal to test the performance for different signal cutoff j.

Note Per share price of HYG is around $86

In [99]:
```
PL_df = pd.DataFrame()
model = ExponentialWeightedReg("HYG")
model.train(['flow_1min','dollar_flow_1min','ewm_vol_3600s','size_imbalance_5min','nav_discount_bid'],'fwd_rtn_
for threshold in [.000001*pow(10, i) for i in range(5)] + [0]:
    model.get_signal(threshold)
    pl,_,_,_ = model.backtest()
    PL_df["{:.1e}".format(threshold)] = pl.cumsum()
longOnly = PL_df['1.0e-05']
```

From the graph below, we see that for high cutoffs, there are only a few trades happening and not high pnl, but for really low cutoffs, the strategy suffered from the noise resulting in high volatility.

In [100…
```
PL_df.plot(figsize=(12,8))
plt.title("HYG Cumulative Return, Long Only Strategy, Different Thresholds");
plt.ylabel("profit (USD)");
```

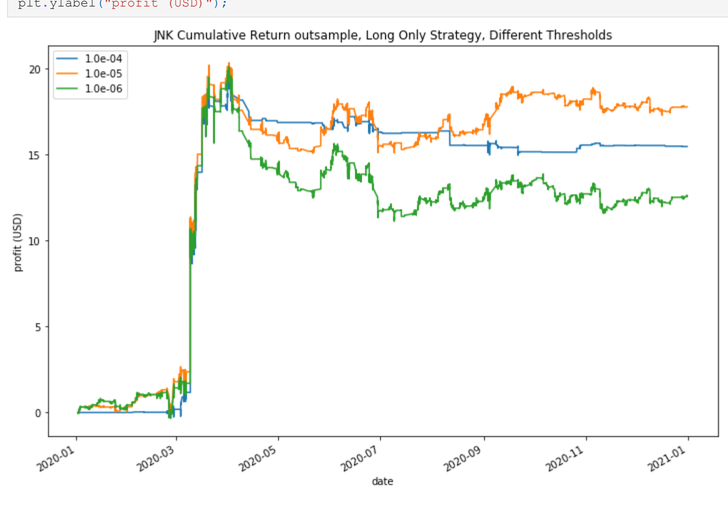


Out sample testing for choosing the best meta parameter

From the preliminary analysis of premium discount and price analysis, we know that ETF HYG and JNK are very similar in movement of prices, we choose the best three thresholds 1e-4,1e-5,1e-6 from in-sample testing of HYG to perform out sample testing on JNK.

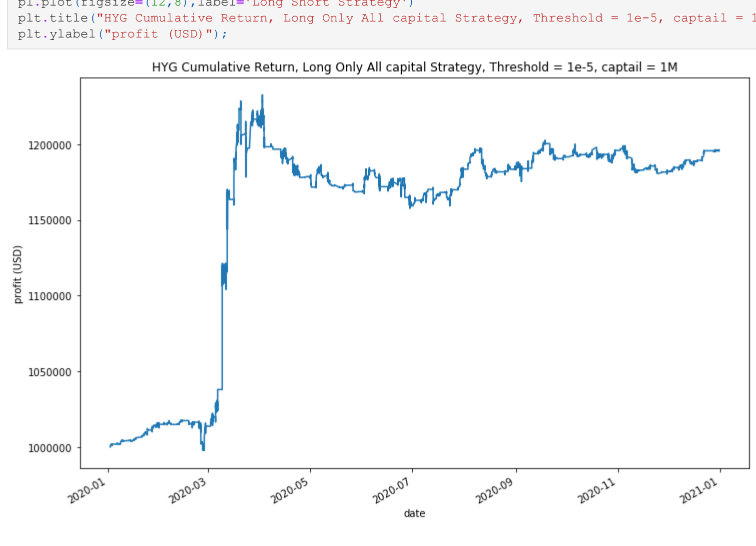
As showed below, the best parameter is 1e-5.

In [102…
```
PL_df = pd.DataFrame()
model = ExponentialWeightedReg("JNK")
model.train(['flow_1min','dollar_flow_1min','ewm_vol_3600s','size_imbalance_5min','nav_discount_bid'],'fwd_rtn_
for threshold in [0.0001,0.00001,0.000001]:
    model.get_signal(threshold)
    pl,_,_,_ = model.backtest()
    PL_df["{:.1e}".format(threshold)] = pl.cumsum()

PL_df.plot(figsize=(12,8))
plt.title("JNK Cumulative Return outsample, Long Only Strategy, Different Thresholds");
plt.ylabel("profit (USD)");
```



Long Only buy with all capital

In [19]:
```
model = ExponentialWeightedReg("HYG")
model.train(['flow_1min','dollar_flow_1min','ewm_vol_3600s','size_imbalance_5min','nav_discount_bid'],'fwd_rtn_
model.get_signal(1e-5)
pl,_,_,_ = model.backtest(long_only=True, all_in_capital=True)
longOnly = pl
pl.plot(figsize=(12,8),label='Long Short Strategy')
plt.title("HYG Cumulative Return, Long Only All capital Strategy, Threshold = 1e-5, captail = 1M");
plt.ylabel("profit (USD)");
```



Long short with all capital

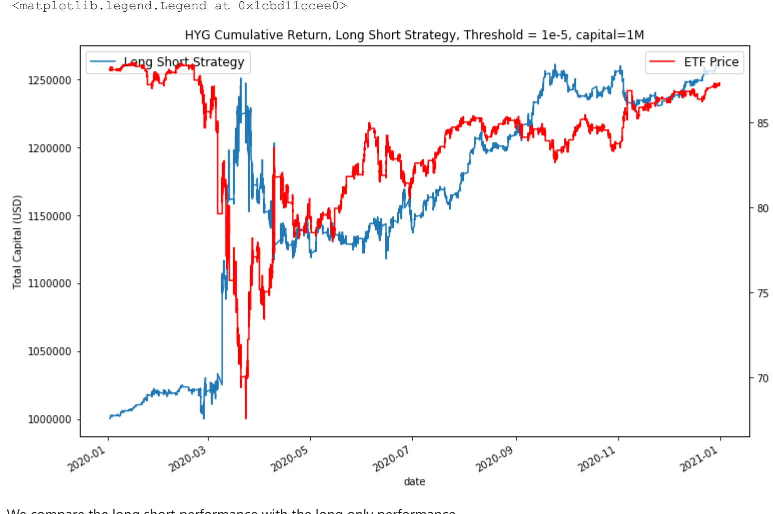
Next we try a long short strategy with buying or shorting with all capital(leverage ratio = 1). We use the best threshold found above which is 1e-5. Below we plot the cumulative PnL of long short strategy. What is more insteresting is that we compare it to the price of the ETF.

It seemed that although our strategy suffered from an intial loss around end February and early March. When ETF price continue to fall in March during the pandemic. The model seemed to have learned from the period and have positive PnL.

On the other hand, when ETF price started to rally in April. Our model is still using the information from March and did not recover from the shift until end of June.

In [18]:
```
model = ExponentialWeightedReg("HYG")
model.train(['flow_1min','dollar_flow_1min','ewm_vol_3600s','size_imbalance_5min','nav_discount_bid'],'fwd_rtn_
model.get_signal(1e-5)
pl,_,_,_ = model.backtest(long_only=False, all_in_capital=True)
longShort = pl
ax=longShort.plot(figsize=(12,8),label='Long Short Strategy')
plt.title("HYG Cumulative Return, Long Short Strategy, Threshold = 1e-5, capital=1M");
plt.ylabel("Total Capital (USD)");

ax2=ax.twinx()
model.data['2020':]['PRICE'].plot(ax=ax2,C='R',label='ETF Price')
ax.legend(loc=2, prop={'size': 12})
plt.legend(loc=1, prop={'size': 12})
```

Out[18]: <matplotlib.legend.Legend at 0x1cbd11ccee0>



We compare the long short performance with the long only performance

In [29]:
```
df = pd.concat([longOnly, longShort], axis=1)
df.rename(columns={"fwd_rtn_5min":"long only", 0:"long short"}, inplace=True)
ax=df.plot(figsize=(12,8));
plt.title("Long Only vs Long Short, Threshold = 1e-5, Capital=1M")
plt.ylabel("profit (USD)")
plt.legend(loc=0, prop={'size': 12})
```

Out[29]: <matplotlib.legend.Legend at 0x1cbd438f2e0>

We compare the long-short with the long-only

For the best threshold of 1e-5, the long short strategy performs better than long only. However it also have a larger drawdown as model recovers from the pandemic. **Especially during March when everyone wants to short**, it is highly likely that makes shorting unaccessible.

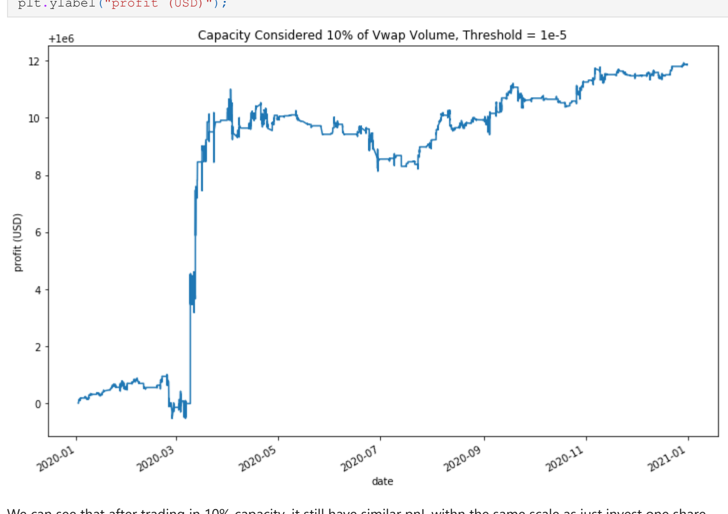
Therefore below in the sections, we only consider long only.

Estimating Capacity

Previously we assumed we can only buy or sell one share, but this is far from reality. We are able to trade multiple shares as long as we are within the capacity of the market.

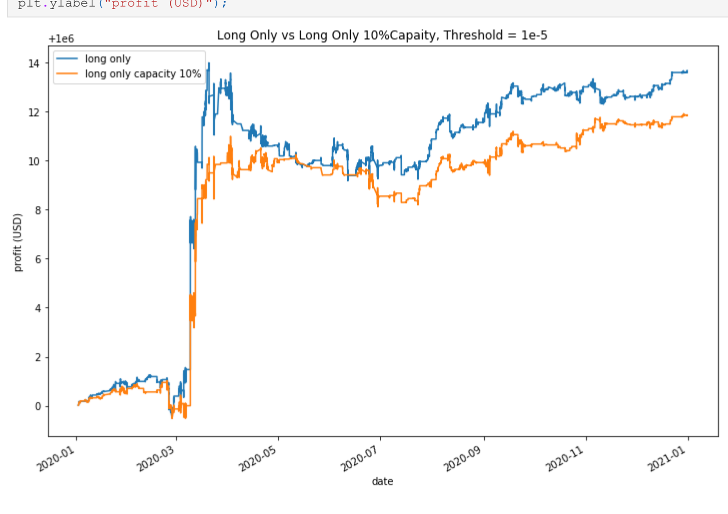
Here we limit **capacity** is defined as the best bid(if sell) or best ask(if buy) quantity of the most recent trade. We asuume we can invest in **10%** of the capacity at any given time and can take on fractional shares. Performing this on the long only position strategy produces the following results.

In [63]:
```
model.get_signal(1e-5)
pl,_,_,_ = model.backtest(sizing_method='vwap_volume_0.1')
longOnlyCapacity = pl.cumsum()+1e6
longOnlyCapacity.plot(figsize=(12,8));
plt.title("Capacity Considered 10% of Vwap Volume, Threshold = 1e-5");
plt.ylabel("profit (USD)");
```



We can see that after trading in 10% capacity, it still have similar pnL withn the same scale as just invest one share.

In [64]:
```
df = pd.concat([longOnly, longOnlyCapacity], axis=1)
df.rename(columns={"1.0e-05":"long only", 0:"long only capacity 10%"}, inplace=True)
df.plot(figsize=(12,8));
plt.title("Long Only vs Long Only 10%Capaity, Threshold = 1e-5");
plt.ylabel("profit (USD)");
```



Transaction Cost

For ETF trading, there are three kind of transaction costs:

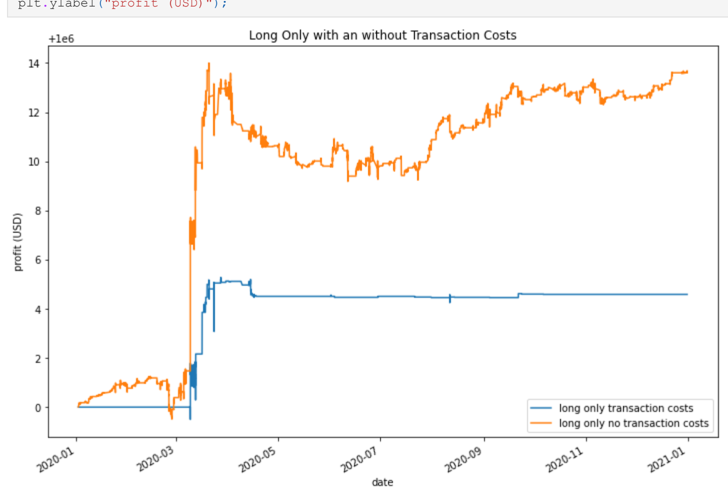
1. crossing bid ask spread
2. exchange commission(neglect)
3. management fee: already deducted from iNAV.

Therefore, the primary cost we need to pay attention to is the bid ask spread.

- transaction costs are assumed using the NBB and NBO
  1. long positions are assumed to be opened at the ask and closed at the bid.
  2. short position are assumed to be opened at the bid and closed at the ask.(long_rtn, short_rtn)

We can see that including transaction cost does hurt performance a lot. As showed below, transaction cost takes away half of the profit

In [86]:
```
k= .0005
#model.train(['flow_1min','dollar_flow_1min','ewm_vol_3600s','size_imbalance_5min','nav_discount_bid'],'fwd_rtn
model.get_signal(k)
pl,_,_,_ = model.backtest(sizing_method='Fixed', trading_cost=True)
longOnly_2 = (pl.cumsum()+1e6)
df = pd.concat([longOnly_2, longOnly], axis=1)
df.rename(columns={"1.0e-05":"long only no transaction costs", 0:"long only transaction costs"}, inplace=True)
df.plot(figsize=(12,8));
plt.title("Long Only with an without Transaction Costs");
plt.ylabel("profit (USD)");
```

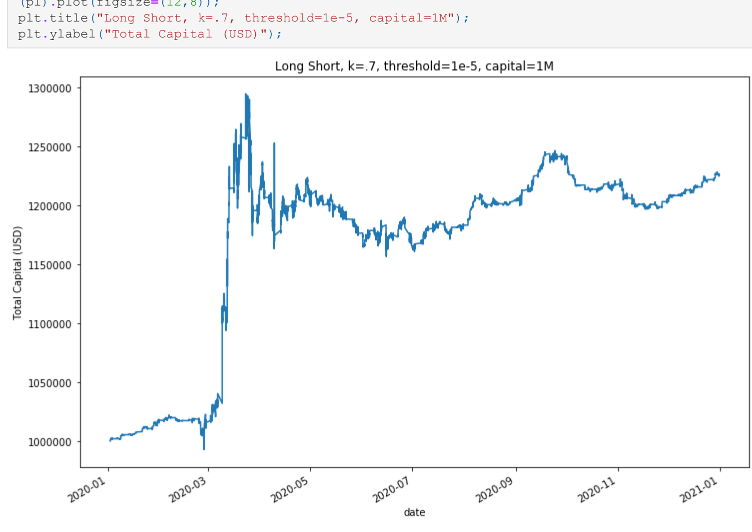


In [ ]:

Return distribution

Overall our best strategy is long short excluding transaction cost

In [31]:
```
model.train(['flow_1min','dollar_flow_1min','ewm_vol_3600s','size_imbalance_5min','nav_discount_bid'],'fwd_rtn_
model.get_signal(1e-5)
pl, tradeNum, trades, posSig = model.backtest(long_only=False, all_in_capital=True)
(pl).plot(figsize=(12,8));
plt.title("Long Short, k=.7, threshold=1e-5, capital=1M");
plt.ylabel("Total Capital (USD)");
```



Below are the return metrics of the strategy. HYG long short strategy has sharpe ratio of 1.36. With max drawdown of 0.1.

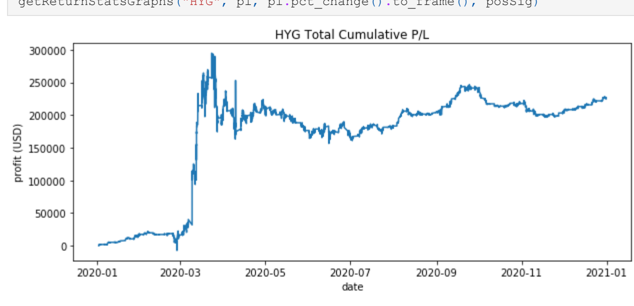
In [41]:
```
name = "HYG"
pl.name = name
trades=pl.pct_change()
trades.name = name
trades = trades.to_frame()
display(perfMetrics(trades, annualization=6.5*60/5*252))
display(tailMetrics(trades))
```

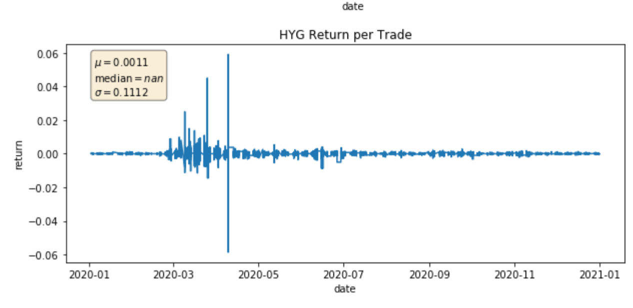
	Sharpe	Vol	Min	Lower Quartile	Median	Mean	Upper Quartile	Max
HYG	1.362323	0.155843	-0.058709	-0.000117	0.0	0.212309	0.000118	0.059153

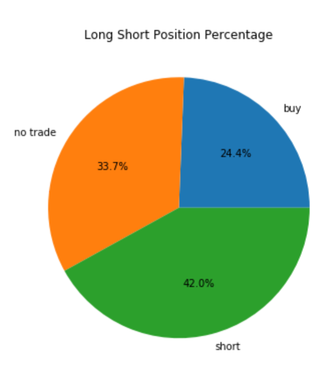
	Max Drawdown	Peak	Bottom	Recover	Skew	Kurtosis	VaR (.05)	CVaR (.05)
HYG	-0.106711	2020-03-23 10:30:00	2020-06-15 14:10:00	None	4.919012	966.664397	-0.000941	-0.002022

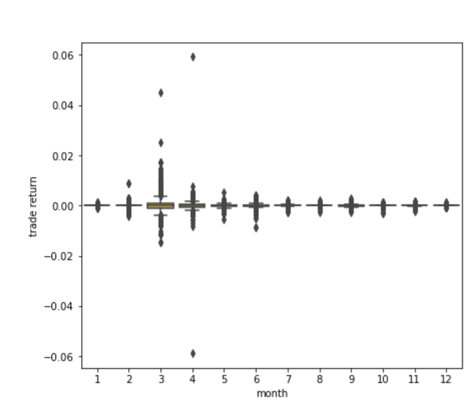
Below we look at the distribution of the strategy returns. We hold split roughly equally between long short and no trades.

In [46]:
```
pl.name = "HYG"
plotCumPL("HYG", pl.diff(), pl.pct_change())
getReturnStatsGraphs("HYG", pl, pl.pct_change().to_frame(), posSig)
```









Model Performance On Each ETF Individually

Here we look at the performance of each ETF long only on a standalone basis. We can see that the strategy with the highest Sharpe ratio is SRLN. The strategy with the lowest drawdown is HYG.

All strategies have more than 1 sharpe ratio some close

In [68]:
```
etfs = ["BKLN", "HYG", "HYGH", "JNK", "PFF", "PGX", "SPHY", "SRLN"]
etfModelInfo = getEtfModelInfo(etfs, 100)  # returns a dict
```

In [74]:
```
return_table=pd.DataFrame()
pnl_table=pd.DataFrame()
for name, etfData in etfModelInfo.items():
    pl, tradeNum, trades, posSig = etfData
    trades.name = name
    trades = trades.to_frame()
    return_table=return_table.append(perfMetrics(trades, annualization=6.5*60/5*252))
    pnl_table=pnl_table.append(tailMetrics(trades))
```

HYGH AND SRLN have the highest sharpe ratio

In [76]:
```
return_table
```

Out[76]:

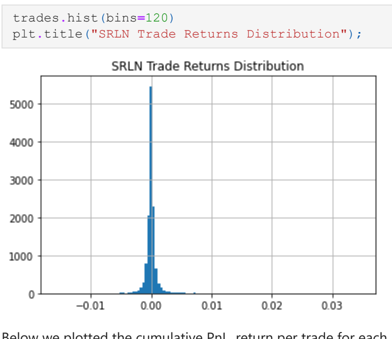
	Sharpe	Vol	Min	Lower Quartile	Median	Mean	Upper Quartile	Max
BKLN	1.339026	0.208856	-0.055025	-0.000230	0.0	0.279664	0.000229	0.040385
HYG	0.959402	0.157857	-0.059153	-0.000232	0.0	0.151449	0.000234	0.058709
HYGH	4.732068	0.201305	-0.007903	0.000000	0.0	0.952591	0.000000	0.050144
JNK	1.318730	0.137080	-0.034324	-0.000195	0.0	0.180771	0.000202	0.032100
PFF	1.447666	0.238075	-0.084548	-0.000262	0.0	0.344653	0.000265	0.084793
PGX	1.573833	0.266957	-0.089172	-0.000334	0.0	0.420145	0.000334	0.099401
SPHY	2.442244	0.330594	-0.030854	-0.000161	0.0	0.807391	0.000004	0.064530
SRLN	3.103366	0.191491	-0.015769	-0.000226	0.0	0.594267	0.000227	0.034664

In [77]:
```
pnl_table
```

Out[77]:

	Max Drawdown	Peak	Bottom	Recover	Skew	Kurtosis	VaR (.05)	CVaR (.05)
BKLN	-0.108337	2020-04-07 10:00:00	2020-12-31 09:40:00	NaT	2.918170	345.881314	-0.000961	-0.002513
HYG	-0.078816	2020-04-02 10:35:00	2020-04-09 14:00:00	NaT	-1.577152	928.423759	-0.001010	-0.002064
HYGH	-0.128880	2020-03-30 14:05:00	2020-04-23 14:10:00	2020-08-14 12:00:00	18.452622	623.733765	-0.000954	-0.002371
JNK	-0.086614	2020-04-09 09:55:00	2020-09-24 09:35:00	NaT	3.313559	236.150070	-0.001004	-0.002004
PFF	-0.191663	2020-03-16 10:45:00	2020-03-18 13:30:00	2020-03-30 12:50:00	3.147320	734.886145	-0.001121	-0.002911
PGX	-0.169414	2020-05-26 15:20:00	2020-12-29 12:30:00	NaT	4.811287	783.041318	-0.001232	-0.003051
SPHY	-0.249394	2020-03-27 14:55:00	2020-12-22 12:55:00	NaT	12.192234	304.301792	-0.001772	-0.003712
SRLN	-0.159958	2020-04-09 10:10:00	2020-12-29 13:00:00	NaT	6.838809	153.948423	-0.001129	-0.002506

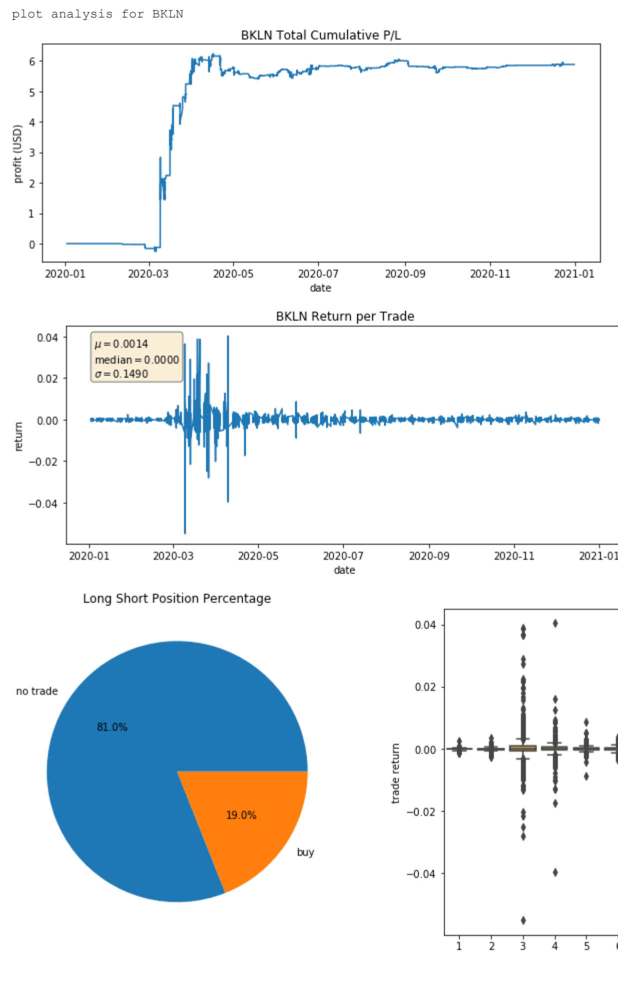
In [99]:
```
trades.hist(bins=120)
plt.title("SRLN Trade Returns Distribution");
```



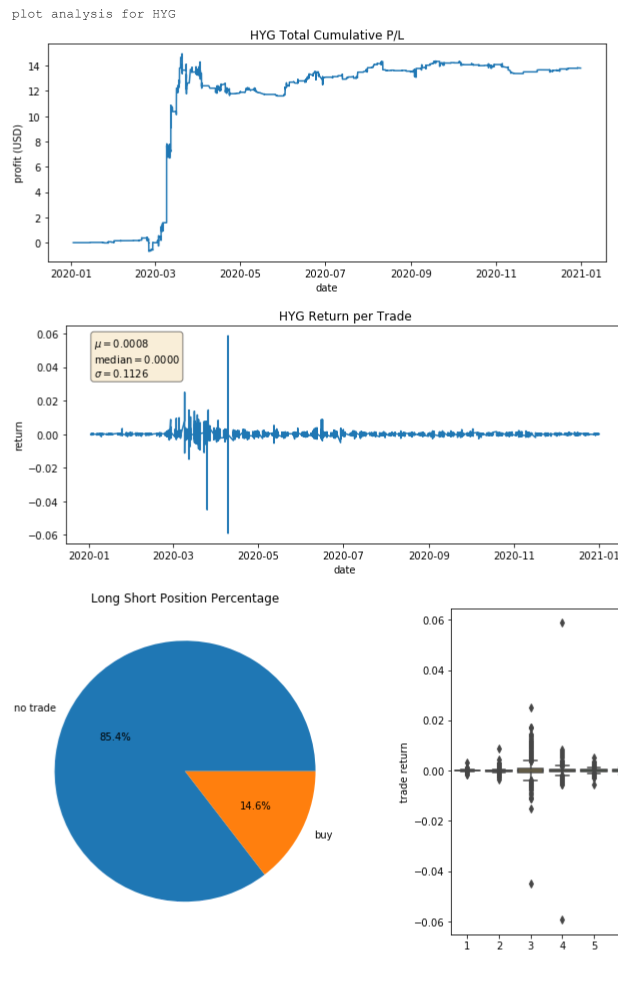
Below we plotted the cumulative PnL, return per trade for each ETF, position holdings over the entire period and the box plot of pnl per month. All ETFs have positive PnL.

In [79]:
```
for name, etfData in etfModelInfo.items():
    pl, tradeNum, trades, posSig = etfData
    pl.name = name
    pl = pl.to_frame()
    trades.name = name
    trades = trades.to_frame()
    print('**********************************************************************************************
    print('plot analysis for', name)
    plotCumPL(name, pl, trades)
    getReturnStatsGraphs(name, pl, trades, posSig)
    print("\n\n\n")
```

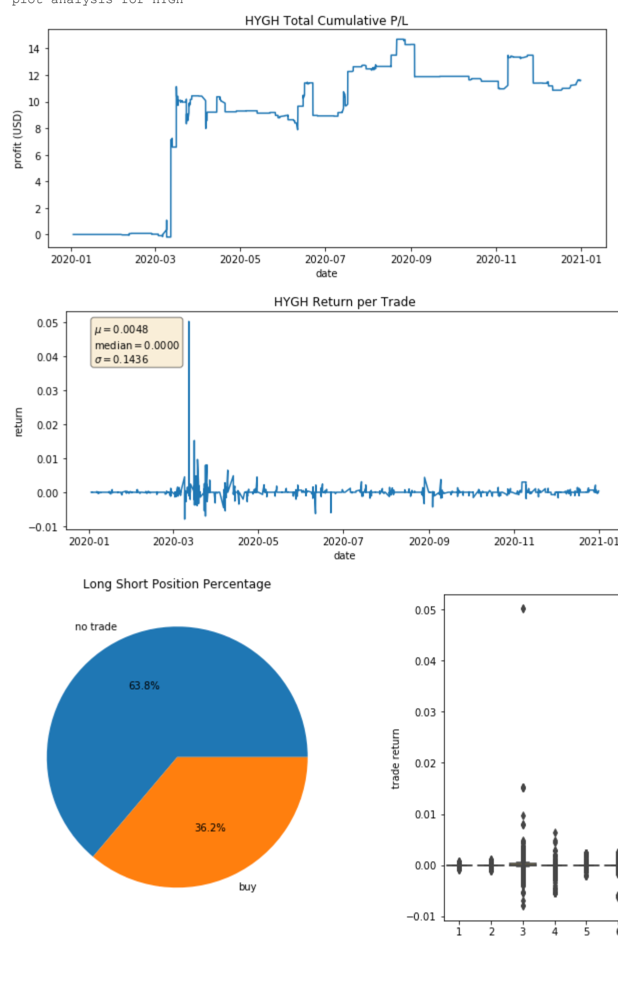
```
****************************************************************************************************
plot analysis for BKLN
```



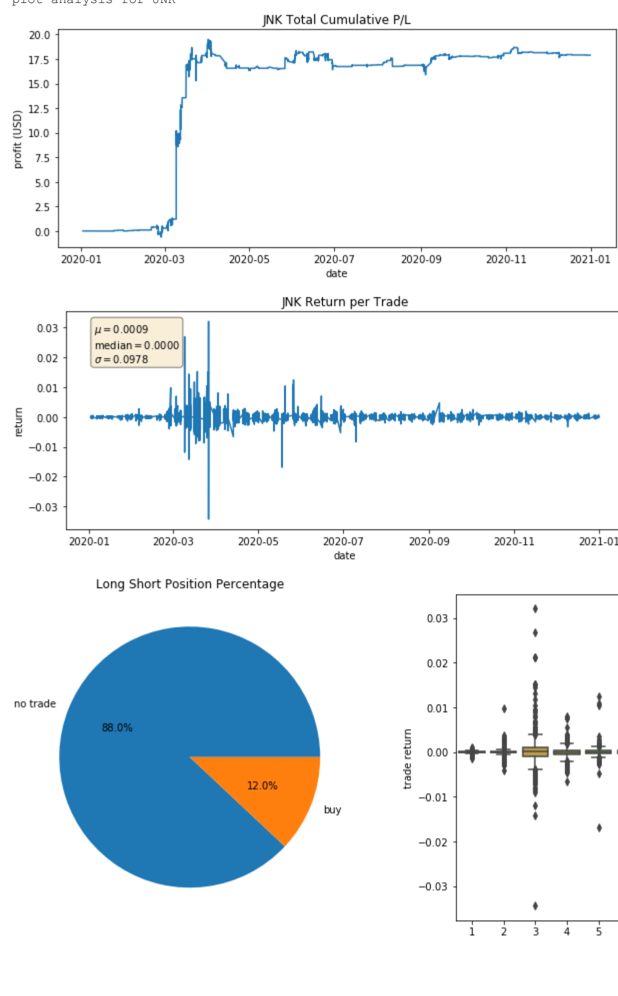

```
****************************************************************************************************
plot analysis for HYG
```



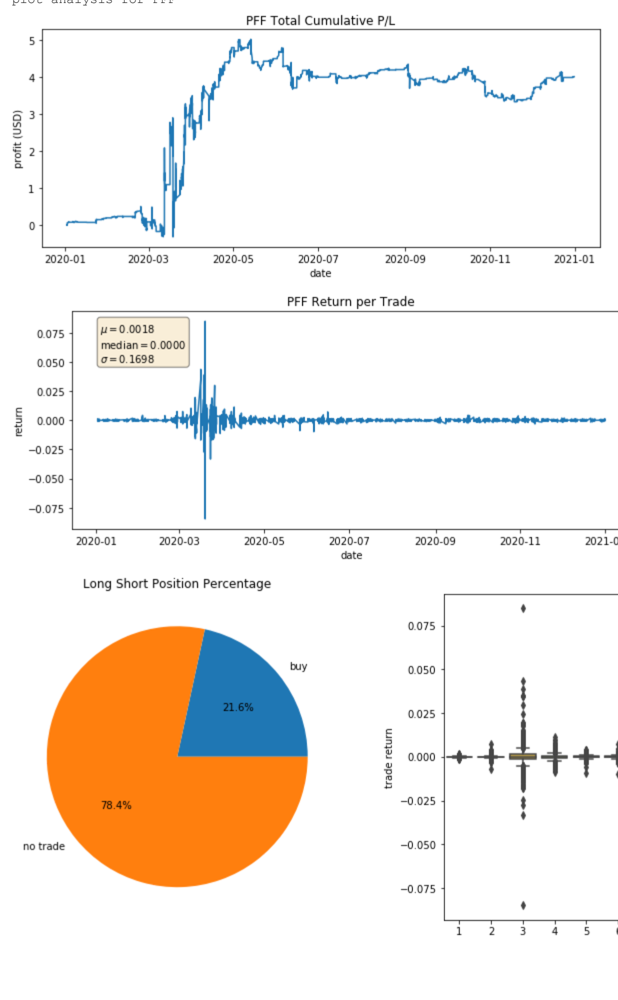

```
****************************************************************************************************
plot analysis for HYGH
```



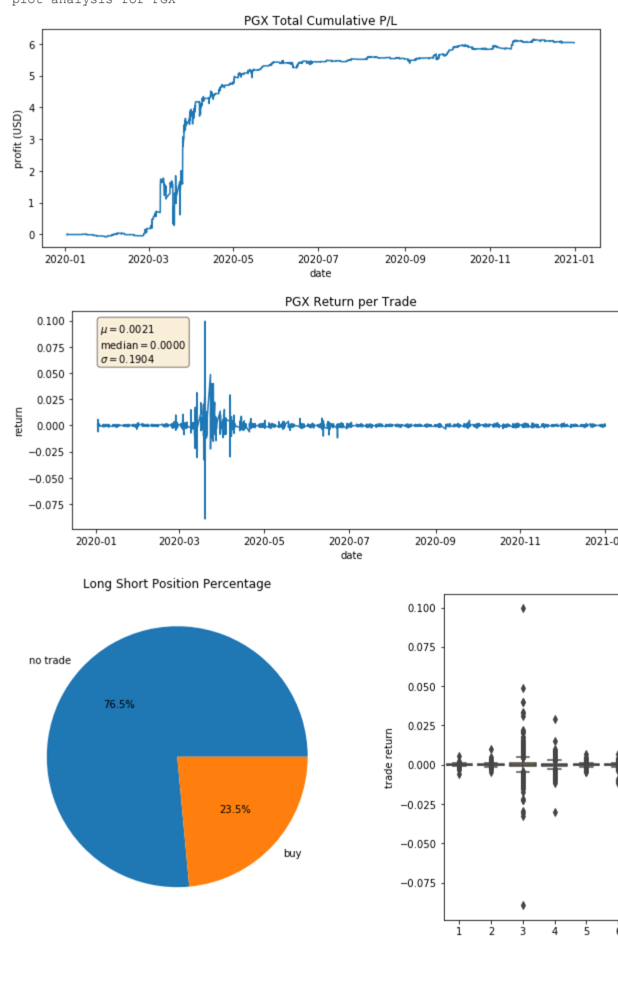

```
****************************************************************************************************
plot analysis for JNK
```



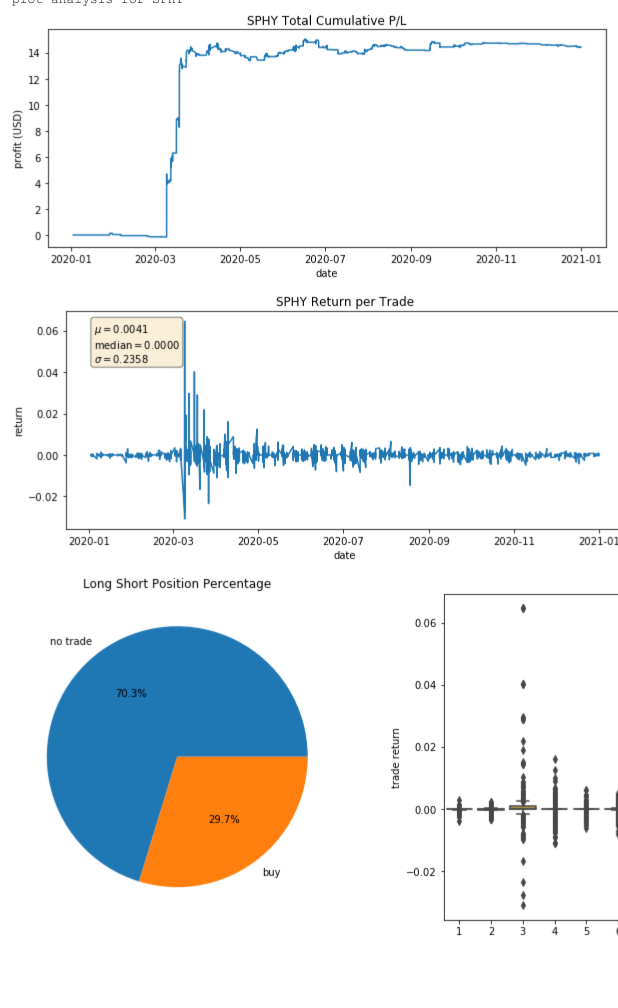

```
****************************************************************************************************
plot analysis for PFF
```



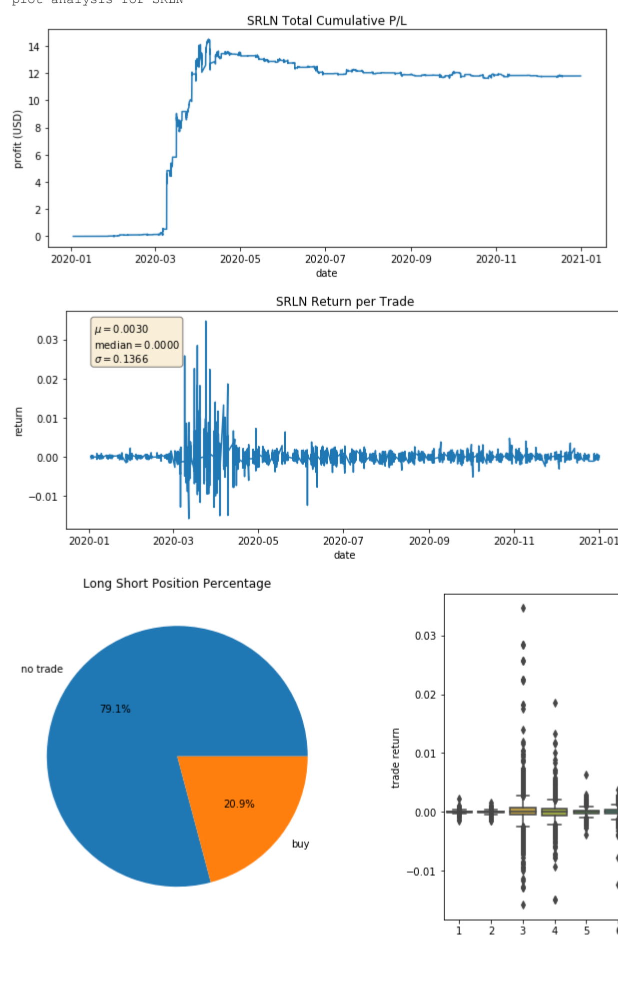

```
****************************************************************************************************
plot analysis for PGX
```




```
****************************************************************************************************
plot analysis for SPHY
```




```
****************************************************************************************************
plot analysis for SRLN
```



Portfolio Construction

Generally, our strategy is a high frequency timing strategy that do not involve portfolio construction and can work on a standalone basis.

However, just a simple equal weighted portfolio is able to outrun a equal weighted underlying ETF return, suggesting our strategy is able to profit from the intraday trades.

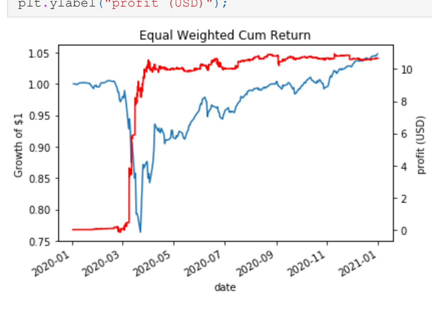
Performance on Equal Weighted Portfolio

In [80]:

```
import quandl
```

In [91]:

```
quandl.ApiConfig.api_key = "myyxRsQkumGXg-LNmd4x"
def fetch_quandl(tickers, columns, start_date, end_date):
    qdata = quandl.get_table('QUOTEMEDIA/PRICES', ticker = list(tickers),
                             qopts = {'columns': list(columns)},
                             date = {'gte': start_date, 'lte': end_date},
                             paginate=True)
    qdata.set_index(['date','ticker'],inplace=True)
    qdata.sort_index(inplace=True)
    return qdata

etfs = ('BKLN','HYG','JNK','PFF','PGX','SPHY','SRLN','HYGH')
etf_data = fetch_quandl(tickers = etfs,
                        columns = ('ticker','date','adj_close'),
                        start_date = '2020-01-01',
                        end_date = '2020-12-31')

etf_returns = (etf_data.groupby(level=1).pct_change()).dropna()
weights = 1/len(etfs)
benchmark_rets = (etf_returns*weights).groupby(level=0).sum()
ax=(1+benchmark_rets).cumprod().plot(title='Equal Weighted Cum Return',
                                     ylabel='Growth of $1',
                                     legend=False)

weights = [1/len(etfs)] * len(etfs)
performance = pd.concat([v[0] for k,v in etfModelInfo.items()], axis=1)
eqWeightPort = (performance.fillna(0) @ weights).cumsum()
ax2=ax.twinx()
eqWeightPort.plot(ax=ax2, c='r');
plt.ylabel("profit (USD)");
```



Conclusion

We think it is an very interesting idea that we can combine high frequency supply and demand to the ETF premium discount. As we all know excess demand for ETF will drive up premium, it is an interesting perspective to link the phenomenon to micromarket structure.

From our analysis, our assumption that flow, volatility, illiquidity, size imbalance are related to ETF return are all proved correct as we perform backtesting on all ETFs and yield positive pnl with sharpe ratio and relatively low drawdown ratios. The strategy works both under long-short and long-only.

We do need to be cautious of the transaction costs that may be imposed during illiquid times. The results in section shows that including transaction cost significantly hurt strategy performance

In [ ]: