Metrics and Analytics Case Study

Pallab Dey
Hansi Hu

Draft – Dated: 29 March, 2018

Overview

Learning outcome: Metrics creation process in MQD and creating research case studies using Jupyter.

Pre-requisite: Uptick training, Market Microstructure knowledge, MQD and Workflow training session 1 to 6.

Contents Summary:

- Introduction
- Metrics Creation Process
- Metrics Test Cases
- Review and Deployment Process
- Analytics Case Studies

Introduction

What are MQD Metrics?

- Generate Financial data catering to research areas such as analysis of Market Efficiency and Fairness, Systemic risk, Market Fragmentation, Fundamental Analysis, Fixed Income Analysis, Energy Market Analysis.

- Based on Python language and uptick API.

- Available at the security level, a group of securities level, index level, a decile of securities level or a market wide level.

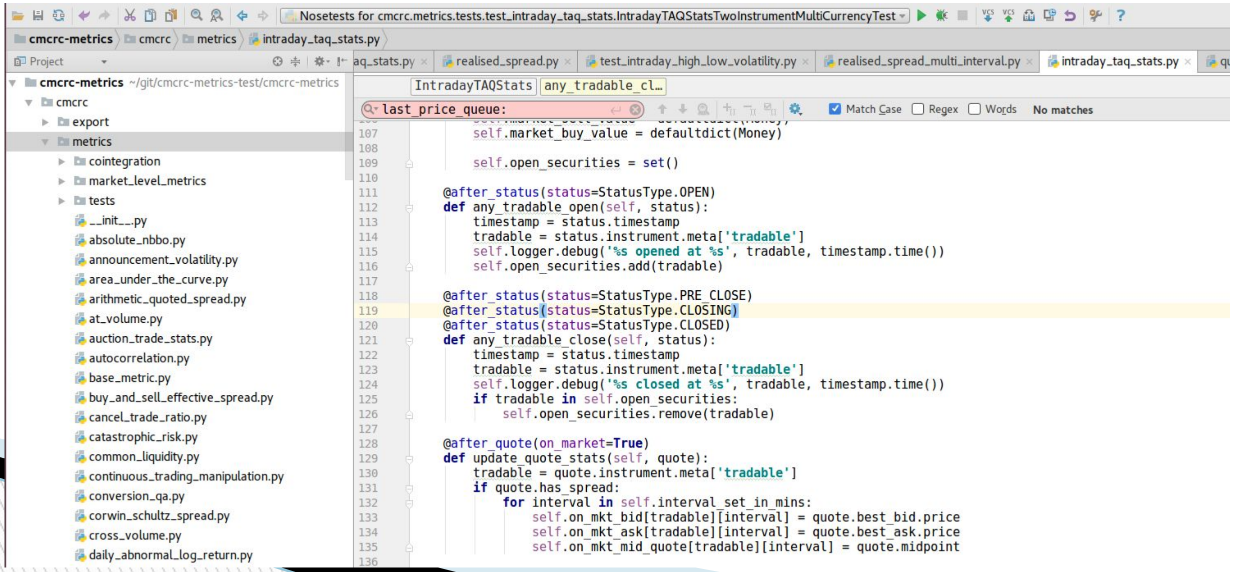
Click here for MQD Link for metric details

Click here for MQD metrics spec (needs login)

Metrics Creation Process

Tools and Environment

- Tools and Languages you need
  - Python – MQD Uptick Metrics is based on Python 3.6
  - Setup and build CMCRC-Metrics Repo https://wiki.cmcrc.com/display/MQ/Work+Environment+Set+Up+for+Uptick+metric
  - Where do we write metrics? Pycharm IDE
    - Easy-to-use and development friendly tool to write code in Python. One stop place to write, verify, test and submit the code for review.



cmcrc-metrics ~/git/cmcrc-metrics-test/cmcrc-metrics
cmcrc
export
metrics
cointegration
market_level_metrics
tests
__init__.py
absolute_nbbo.py
announcement_volatility.py
area_under_the_curve.py
arithmetic_quoted_spread.py
at_volume.py
auction_trade_stats.py
autocorrelation.py
base_metric.py
buy_and_sell_effective_spread.py
cancel_trade_ratio.py
catastrophic_risk.py
common_liquidity.py
continuous_trading_manipulation.py
conversion_qa.py
corwin_schultz_spread.py
cross_volume.py
daily_abnormal_log_return.py

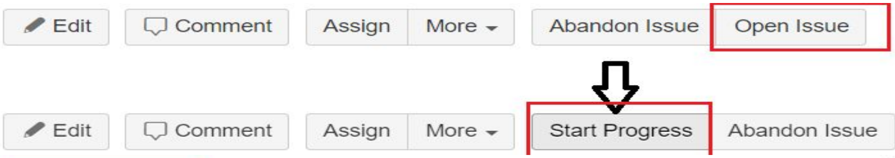
```
        self.market_buy_value = defaultdict(Money)

        self.open_securities = set()

    @after_status(status=StatusType.OPEN)
    def any_tradable_open(self, status):
        timestamp = status.timestamp
        tradable = status.instrument.meta['tradable']
        self.logger.debug('%s opened at %s', tradable, timestamp.time())
        self.open_securities.add(tradable)

    @after_status(status=StatusType.PRE_CLOSE)
    @after_status(status=StatusType.CLOSING)
    @after_status(status=StatusType.CLOSED)
    def any_tradable_close(self, status):
        timestamp = status.timestamp
        tradable = status.instrument.meta['tradable']
        self.logger.debug('%s closed at %s', tradable, timestamp.time())
        if tradable in self.open_securities:
            self.open_securities.remove(tradable)

    @after_quote(on_market=True)
    def update_quote_stats(self, quote):
        tradable = quote.instrument.meta['tradable']
        if quote.has_spread:
            for interval in self.interval_set_in_mins:
                self.on_mkt_bid[tradable][interval] = quote.best_bid.price
                self.on_mkt_ask[tradable][interval] = quote.best_ask.price
                self.on_mkt_mid_quote[tradable][interval] = quote.midpoint
```

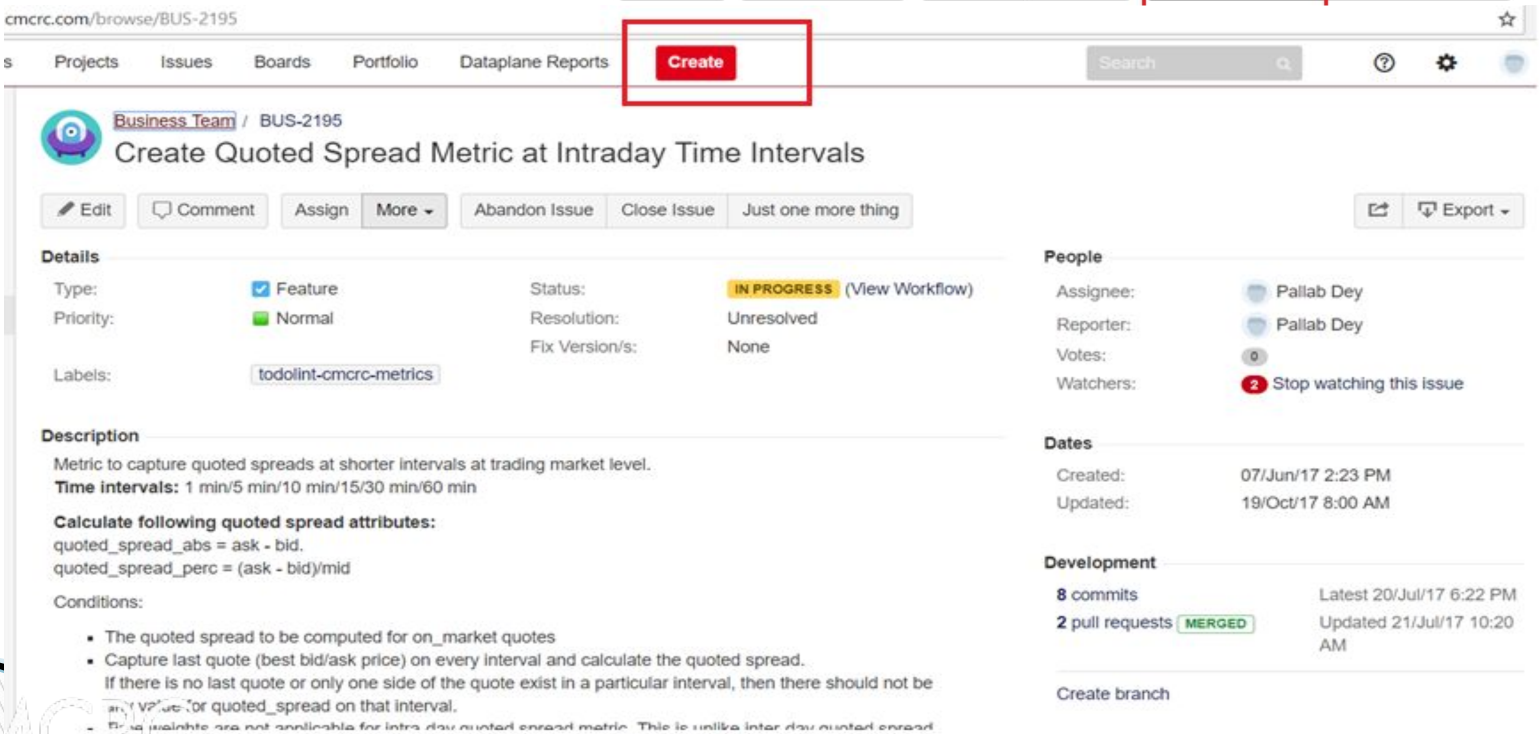
Steps to create and deploy a metric

- Create JIRA for the metric
- Ensure you are working on latest code repository (update master branch using “git pull”
- Create new branch from Master (JIRA name should be part of branch name)
- Create New Metric class file
- Create Test case file
- Run the test case file to test for success
- Run the metric on real trading data
- Submit the metric code and test case code for review
- Changes to display in MQD UI

Create JIRA (Overview)

- Create JIRA using “Create” Button
- Fill-in JIRA title, description and add any attachments (if applicable)
- Assign the JIRA to the “Assignee” who will work on the JIRA
- Update the status to “Open” followed by “Start Progress” before starting to implement.





Business Team / BUS-2195

Create Quoted Spread Metric at Intraday Time Intervals

Details

Type:	☑ Feature	Status:	IN PROGRESS (View Workflow)
Priority:	🟩 Normal	Resolution:	Unresolved
		Fix Version/s:	None
Labels:	todolint-cmcrc-metrics		

Description

Metric to capture quoted spreads at shorter intervals at trading market level.
Time intervals: 1 min/5 min/10 min/15/30 min/60 min

Calculate following quoted spread attributes:
quoted_spread_abs = ask - bid.
quoted_spread_perc = (ask - bid)/mid

Conditions:

- The quoted spread to be computed for on_market quotes
- Capture last quote (best bid/ask price) on every interval and calculate the quoted spread.
  If there is no last quote or only one side of the quote exist in a particular interval, then there should not be any value for quoted_spread on that interval.
- Time weights are not applicable for intra day quoted spread metric. This is unlike inter day quoted spread

People

Assignee:	Pallab Dey
Reporter:	Pallab Dey
Votes:	0
Watchers:	2 Stop watching this issue

Dates

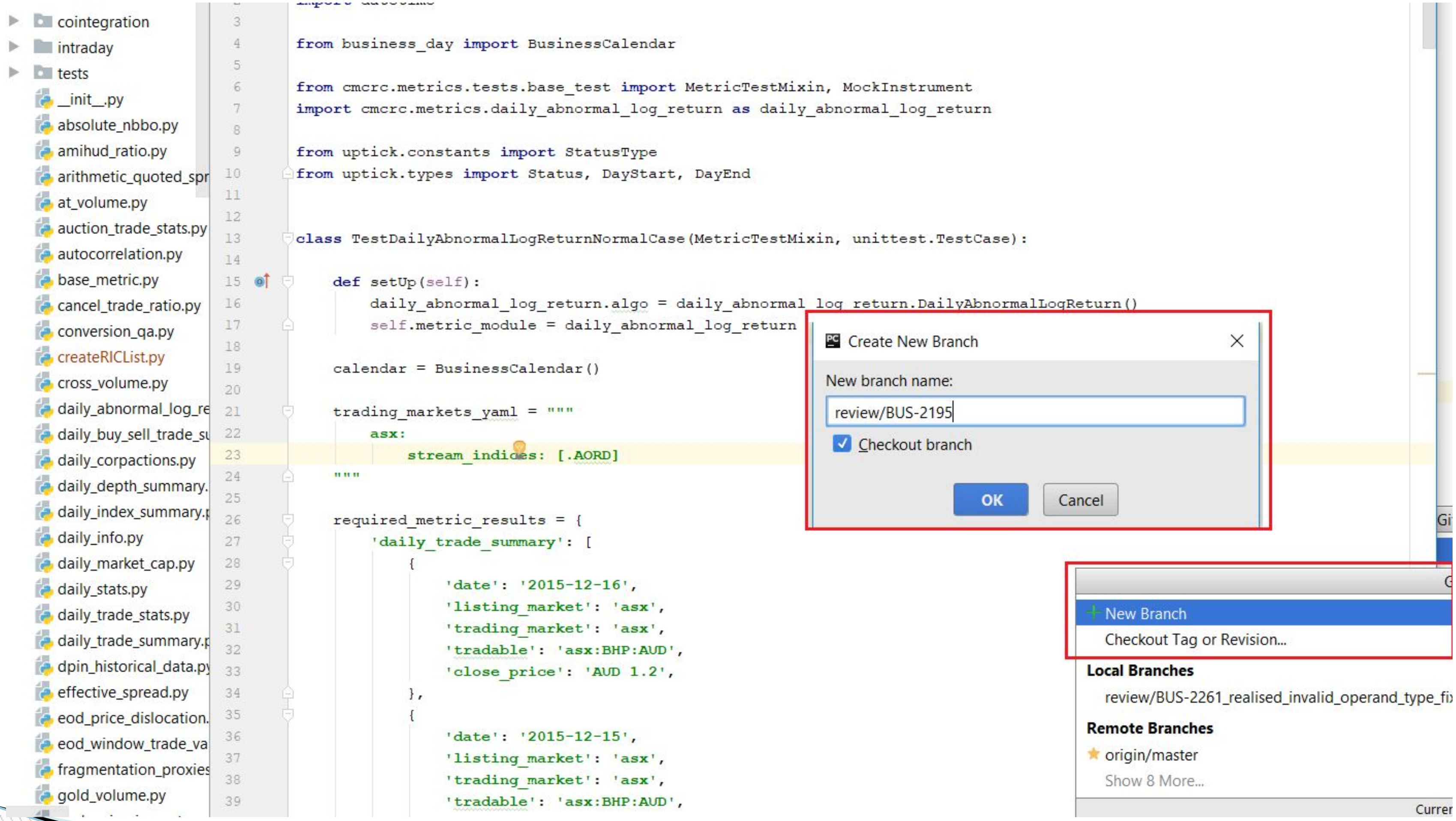
Created:	07/Jun/17 2:23 PM
Updated:	19/Oct/17 8:00 AM

Development

8 commits	Latest 20/Jul/17 6:22 PM
2 pull requests MERGED	Updated 21/Jul/17 10:20 AM

Create branch

Create New Branch



```
▶ cointegration
▶ intraday
▶ tests
  _init_.py
  absolute_nbbo.py
  amihud_ratio.py
  arithmetic_quoted_spr
  at_volume.py
  auction_trade_stats.py
  autocorrelation.py
  base_metric.py
  cancel_trade_ratio.py
  conversion_qa.py
  createRICList.py
  cross_volume.py
  daily_abnormal_log_re
  daily_buy_sell_trade_su
  daily_corpactions.py
  daily_depth_summary.
  daily_index_summary.p
  daily_info.py
  daily_market_cap.py
  daily_stats.py
  daily_trade_stats.py
  daily_trade_summary.p
  dpin_historical_data.py
  effective_spread.py
  eod_price_dislocation.
  eod_window_trade_va
  fragmentation_proxies
  gold_volume.py
```

```python
import datetime

from business_day import BusinessCalendar

from cmcrc.metrics.tests.base_test import MetricTestMixin, MockInstrument
import cmcrc.metrics.daily_abnormal_log_return as daily_abnormal_log_return

from uptick.constants import StatusType
from uptick.types import Status, DayStart, DayEnd


class TestDailyAbnormalLogReturnNormalCase(MetricTestMixin, unittest.TestCase):

    def setUp(self):
        daily_abnormal_log_return.algo = daily_abnormal_log_return.DailyAbnormalLogReturn()
        self.metric_module = daily_abnormal_log_return

    calendar = BusinessCalendar()

    trading_markets_yaml = """
        asx:
            stream_indices: [.AORD]
    """

    required_metric_results = {
        'daily_trade_summary': [
            {
                'date': '2015-12-16',
                'listing_market': 'asx',
                'trading_market': 'asx',
                'tradable': 'asx:BHP:AUD',
                'close_price': 'AUD 1.2',
            },
            {
                'date': '2015-12-15',
                'listing_market': 'asx',
                'trading_market': 'asx',
                'tradable': 'asx:BHP:AUD',
```

Create New Branch

New branch name:

review/BUS-2195

☑ Checkout branch

OK | Cancel

+ New Branch

Checkout Tag or Revision...

Local Branches

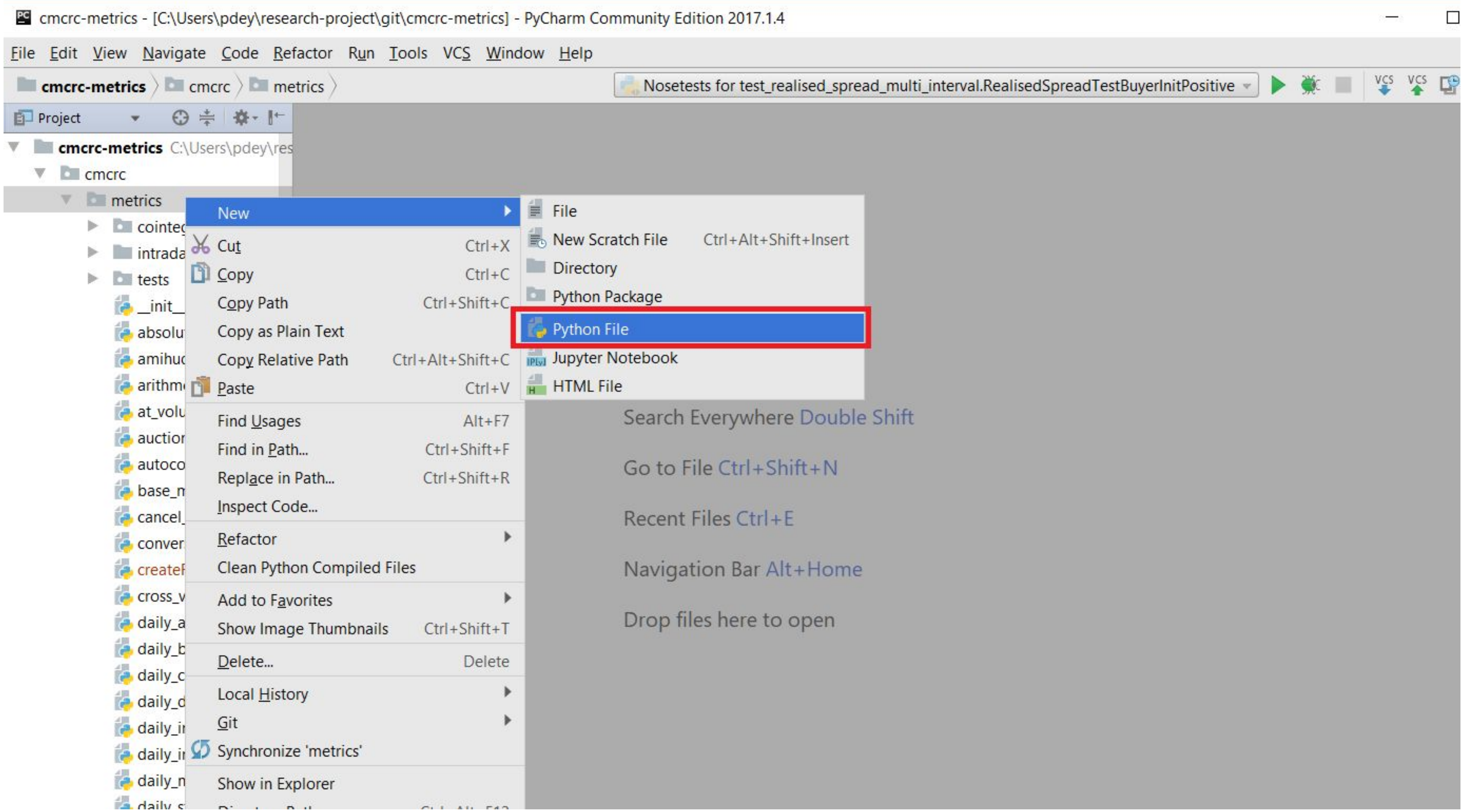
review/BUS-2261_realised_invalid_operand_type_fi

Remote Branches

★ origin/master

Show 8 More...

Create New MQD Metric File


cmcrc-metrics - [C:\Users\pdey\research-project\git\cmcrc-metrics] - PyCharm Community Edition 2017.1.4
File Edit View Navigate Code Refactor Run Tools VCS Window Help
cmcrc-metrics 〉 cmcrc 〉 metrics 〉
Nosetests for test_realised_spread_multi_interval.RealisedSpreadTestBuyerInitPositive
Project
cmcrc-metrics C:\Users\pdey\res
cmcrc
metrics
cointeg
intrada
tests
init
absolu
amihu
arithm
at_volu
auctior
autoco
base_n
cancel_
conver
createl
cross_v
daily_a
daily_b
daily_c
daily_d
daily_i
daily_i
daily_n
daily_s
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy as Plain Text
Copy Relative Path Ctrl+Alt+Shift+C
Paste Ctrl+V
Find Usages Alt+F7
Find in Path... Ctrl+Shift+F
Replace in Path... Ctrl+Shift+R
Inspect Code...
Refactor
Clean Python Compiled Files
Add to Favorites
Show Image Thumbnails Ctrl+Shift+T
Delete... Delete
Local History
Git
Synchronize 'metrics'
Show in Explorer
File
New Scratch File Ctrl+Alt+Shift+Insert
Directory
Python Package
Python File
Jupyter Notebook
HTML File
Search Everywhere Double Shift
Go to File Ctrl+Shift+N
Recent Files Ctrl+E
Navigation Bar Alt+Home
Drop files here to open

- Create new metric and test case file using this option.
- Test case file should start with “test” and then metric file name. For example, if metric is named as quoted_spread, test case should be named as ‘test_quoted_spread’.

Key Metric Elements

- Import packages – uptick and python specific packages

- Python variables (dictionary, list, set, Boolean etc.)

- Output fields definition: attributes that will be in output csv file.

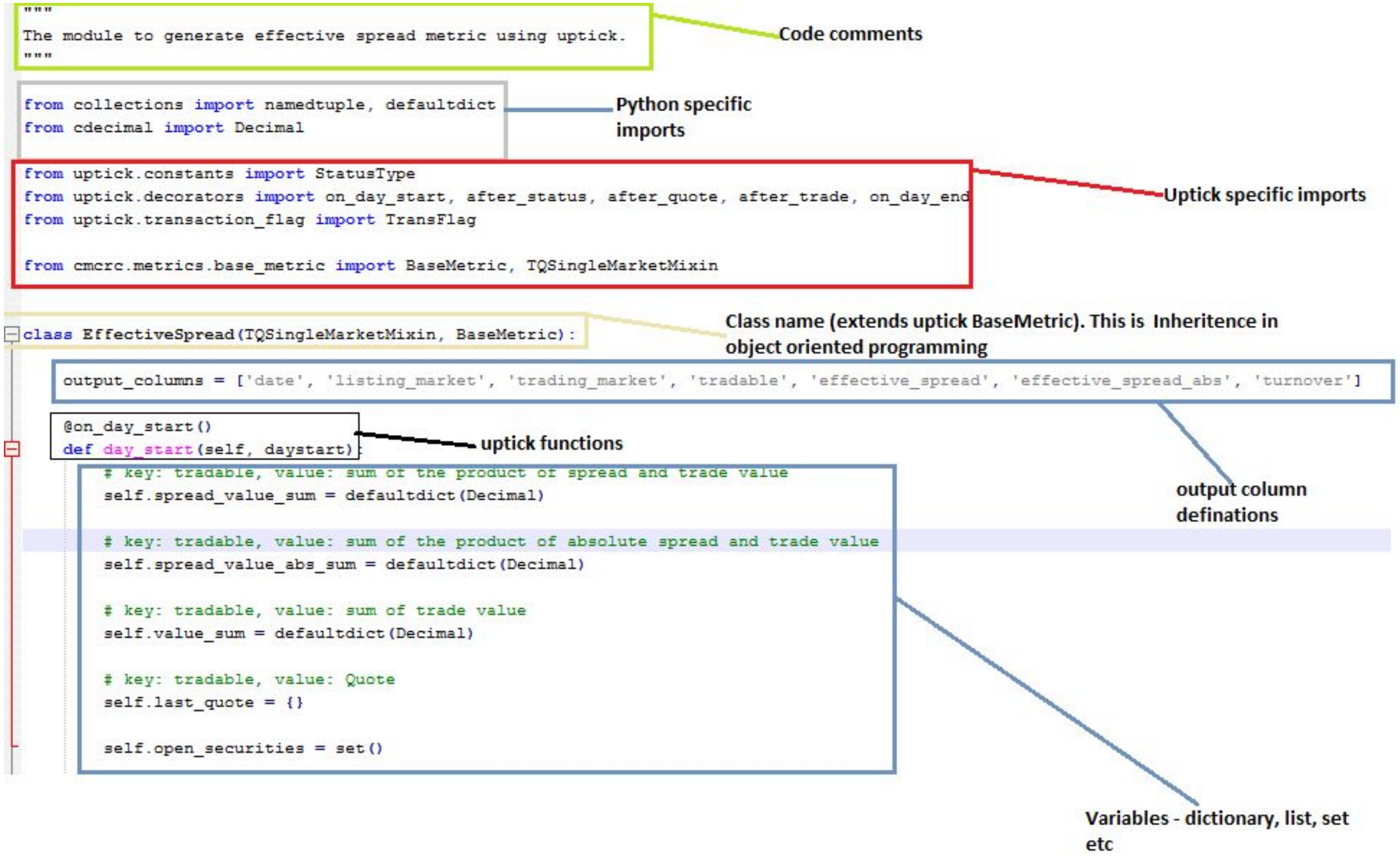
- How do we construct the instrument list for which metric needs to generate the result?
  - **Scenario 1:** Using the instruments opened for trading.
  - **Scenario 2:** Using the list of instruments from a metric output (click here for example)

- Uptick functions –
  - Event Triggers. For example: @on_day_start, @after_status (flag input as StatusType.OPEN/PRE_OPEN/PRE_CLOSE/CLOSING/CLOSED), after_quote, after_trade (takes input such as on_market = True/False, @on_day_end

    In python terms these are called as 'decorator'.

  - Uptick pre-defined input objects(for e.g. trade, quote, status). Provides information such as:
    Security information, trade/quote , timestamp, trade price and volume, bid/ask price and volume information.
    For details on uptick pre-defined objects & variables, click here

Key Metric Elements (Cont...)



```
"""
The module to generate effective spread metric using uptick.
"""

from collections import namedtuple, defaultdict
from cdecimal import Decimal

from uptick.constants import StatusType
from uptick.decorators import on_day_start, after_status, after_quote, after_trade, on_day_end
from uptick.transaction_flag import TransFlag

from cmcrc.metrics.base_metric import BaseMetric, TQSingleMarketMixin

class EffectiveSpread(TQSingleMarketMixin, BaseMetric):

    output_columns = ['date', 'listing_market', 'trading_market', 'tradable', 'effective_spread', 'effective_spread_abs', 'turnover']

    @on_day_start()
    def day_start(self, daystart):
        # key: tradable, value: sum of the product of spread and trade value
        self.spread_value_sum = defaultdict(Decimal)

        # key: tradable, value: sum of the product of absolute spread and trade value
        self.spread_value_abs_sum = defaultdict(Decimal)

        # key: tradable, value: sum of trade value
        self.value_sum = defaultdict(Decimal)

        # key: tradable, value: Quote
        self.last_quote = {}

        self.open_securities = set()
```

Code comments

Python specific imports

Uptick specific imports

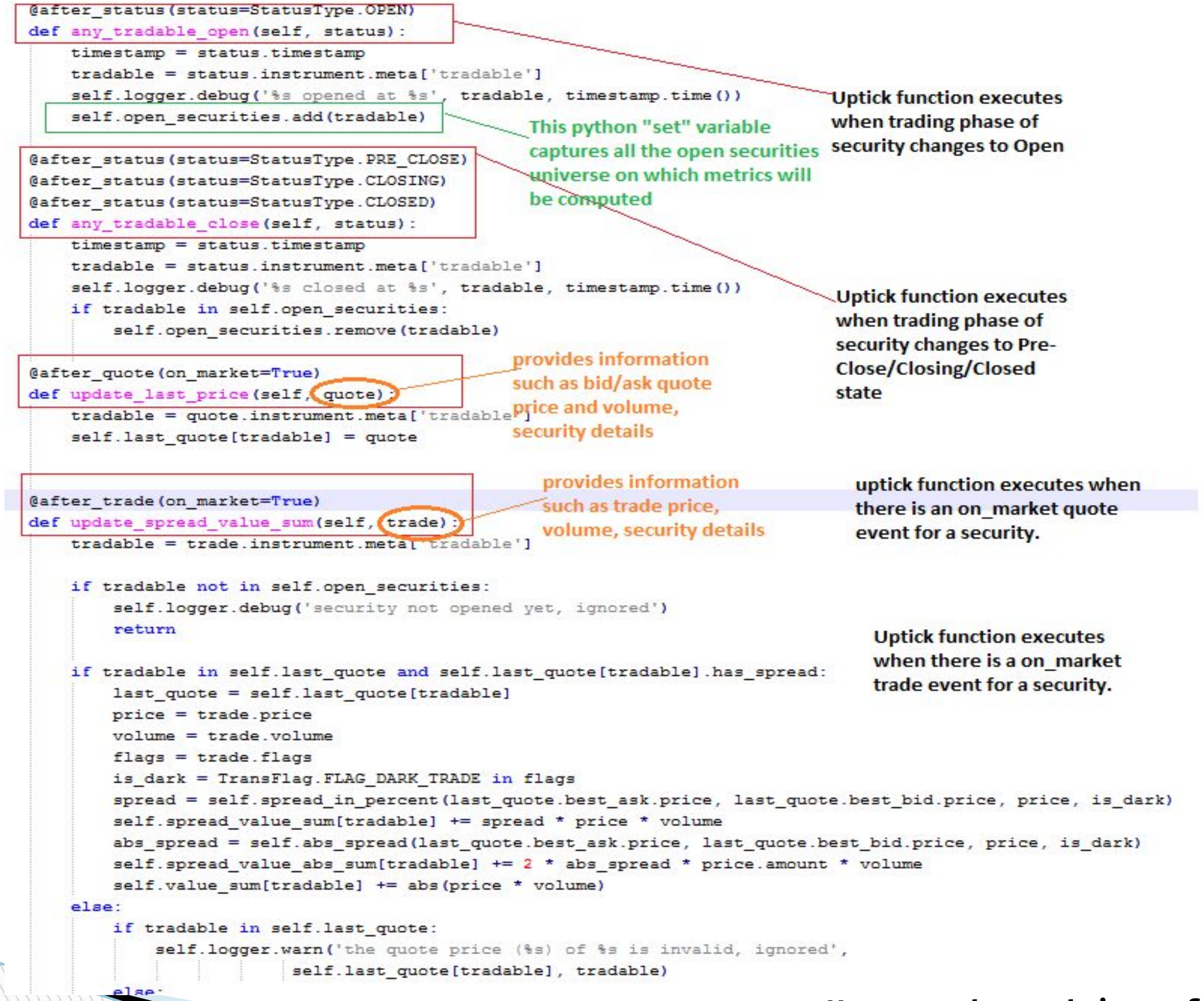
Class name (extends uptick BaseMetric). This is Inheritence in object oriented programming

uptick functions

output column definations

Variables - dictionary, list, set etc

Key Metric Elements (Cont…)



```
@after_status(status=StatusType.OPEN)
def any_tradable_open(self, status):
    timestamp = status.timestamp
    tradable = status.instrument.meta['tradable']
    self.logger.debug('%s opened at %s', tradable, timestamp.time())
    self.open_securities.add(tradable)

@after_status(status=StatusType.PRE_CLOSE)
@after_status(status=StatusType.CLOSING)
@after_status(status=StatusType.CLOSED)
def any_tradable_close(self, status):
    timestamp = status.timestamp
    tradable = status.instrument.meta['tradable']
    self.logger.debug('%s closed at %s', tradable, timestamp.time())
    if tradable in self.open_securities:
        self.open_securities.remove(tradable)

@after_quote(on_market=True)
def update_last_price(self, quote):
    tradable = quote.instrument.meta['tradable']
    self.last_quote[tradable] = quote

@after_trade(on_market=True)
def update_spread_value_sum(self, trade):
    tradable = trade.instrument.meta['tradable']

    if tradable not in self.open_securities:
        self.logger.debug('security not opened yet, ignored')
        return

    if tradable in self.last_quote and self.last_quote[tradable].has_spread:
        last_quote = self.last_quote[tradable]
        price = trade.price
        volume = trade.volume
        flags = trade.flags
        is_dark = TransFlag.FLAG_DARK_TRADE in flags
        spread = self.spread_in_percent(last_quote.best_ask.price, last_quote.best_bid.price, price, is_dark)
        self.spread_value_sum[tradable] += spread * price * volume
        abs_spread = self.abs_spread(last_quote.best_ask.price, last_quote.best_bid.price, price, is_dark)
        self.spread_value_abs_sum[tradable] += 2 * abs_spread * price.amount * volume
        self.value_sum[tradable] += abs(price * volume)
    else:
        if tradable in self.last_quote:
            self.logger.warn('the quote price (%s) of %s is invalid, ignored',
                             self.last_quote[tradable], tradable)
        else:
```

Uptick function executes when trading phase of security changes to Open

This python "set" variable captures all the open securities universe on which metrics will be computed

Uptick function executes when trading phase of security changes to Pre-Close/Closing/Closed state

provides information such as bid/ask quote price and volume, security details

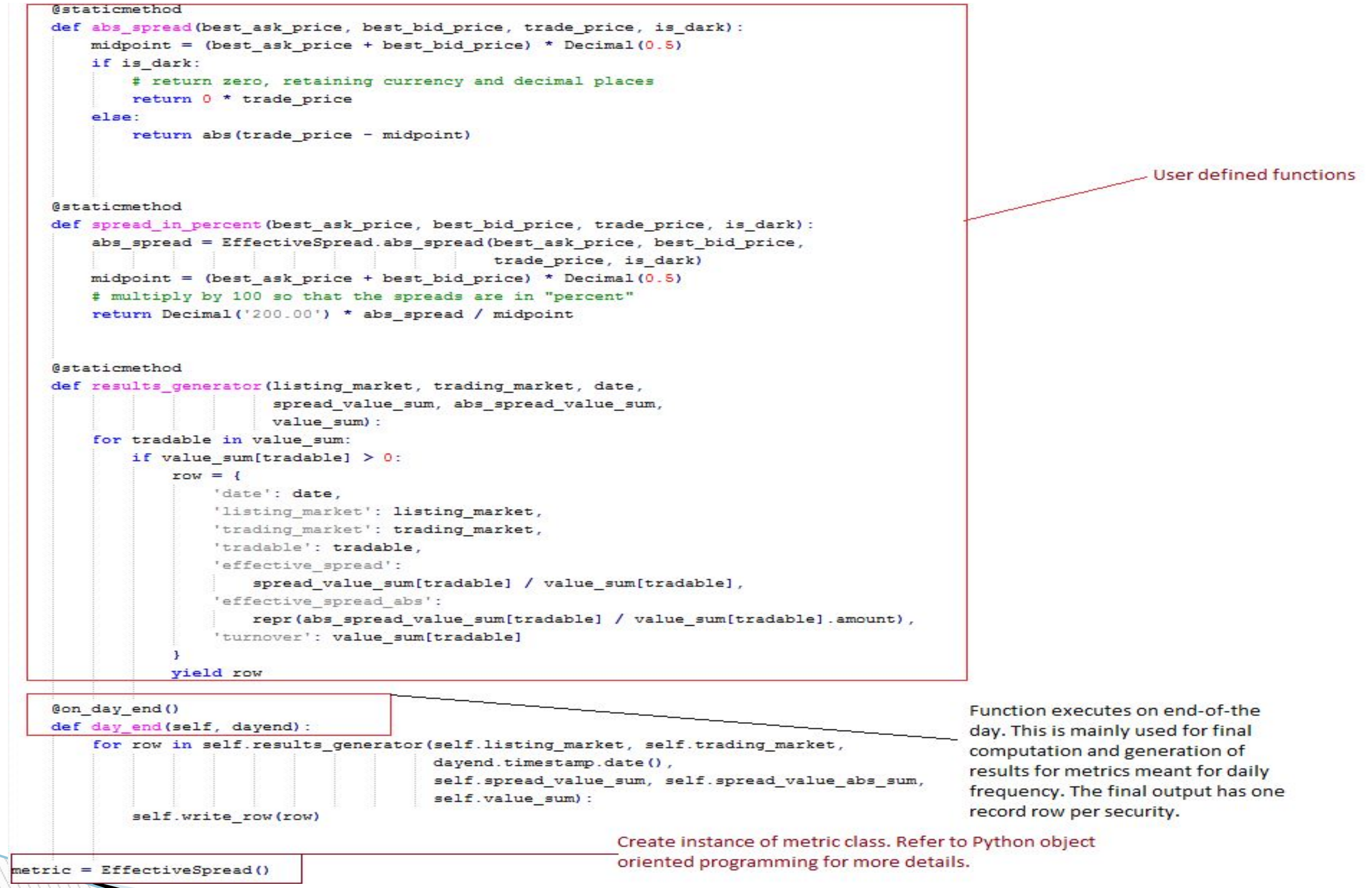
provides information such as trade price, volume, security details

uptick function executes when there is an on_market quote event for a security.

Uptick function executes when there is a on_market trade event for a security.

Users can choose their own function names (which starts with "def …). However, for good naming style, this should ideally be self-explanatory in terms of purpose of the function

Key Metric Elements (Cont…)



```
    @staticmethod
    def abs_spread(best_ask_price, best_bid_price, trade_price, is_dark):
        midpoint = (best_ask_price + best_bid_price) * Decimal(0.5)
        if is_dark:
            # return zero, retaining currency and decimal places
            return 0 * trade_price
        else:
            return abs(trade_price - midpoint)


    @staticmethod
    def spread_in_percent(best_ask_price, best_bid_price, trade_price, is_dark):
        abs_spread = EffectiveSpread.abs_spread(best_ask_price, best_bid_price,
                                                trade_price, is_dark)
        midpoint = (best_ask_price + best_bid_price) * Decimal(0.5)
        # multiply by 100 so that the spreads are in "percent"
        return Decimal('200.00') * abs_spread / midpoint


    @staticmethod
    def results_generator(listing_market, trading_market, date,
                          spread_value_sum, abs_spread_value_sum,
                          value_sum):
        for tradable in value_sum:
            if value_sum[tradable] > 0:
                row = {
                    'date': date,
                    'listing_market': listing_market,
                    'trading_market': trading_market,
                    'tradable': tradable,
                    'effective_spread':
                        spread_value_sum[tradable] / value_sum[tradable],
                    'effective_spread_abs':
                        repr(abs_spread_value_sum[tradable] / value_sum[tradable].amount),
                    'turnover': value_sum[tradable]
                }
                yield row

    @on_day_end()
    def day_end(self, dayend):
        for row in self.results_generator(self.listing_market, self.trading_market,
                                          dayend.timestamp.date(),
                                          self.spread_value_sum, self.spread_value_abs_sum,
                                          self.value_sum):
            self.write_row(row)

metric = EffectiveSpread()
```

User defined functions

Function executes on end-of-the day. This is mainly used for final computation and generation of results for metrics meant for daily frequency. The final output has one record row per security.

Create instance of metric class. Refer to Python object oriented programming for more details.

User defined functions does not have uptick decorators (like after_quote , after_trade, after_status etc) and take any set of user defined input variables. When would you usually like to define your own functions?

When do we use variables or functions with self

- Self refers to current instance of the "Class"
- Variables starting with "self" means the variables values belong to instance of the class. The values can be accessed anywhere within any functions having access to self.
- In function, it refers to the instance whose method is called.

So do all functions and variables need to have self?

Quick facts: *Concept of using "self" is common to object oriented program (OOP) languages (Java has something called "this").*

Definition of Class: *A class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables a.k.a attributes), and implementations of behavior (member functions a.k.a methods) (reference: https://brilliant.org/wiki/classes-oop/)*

Coding Best Practices and Due Diligence

- Is the data structure appropriate? For example correct usage of dictionary definitions. Common dictionary key is tradable. Dictionary could be multi-dimensional (i.e. having multiple keys).

  Exercise: in which case do you think we will use interval as another dictionary key along with tradable? Click here for an example. Can you provide example of keys?

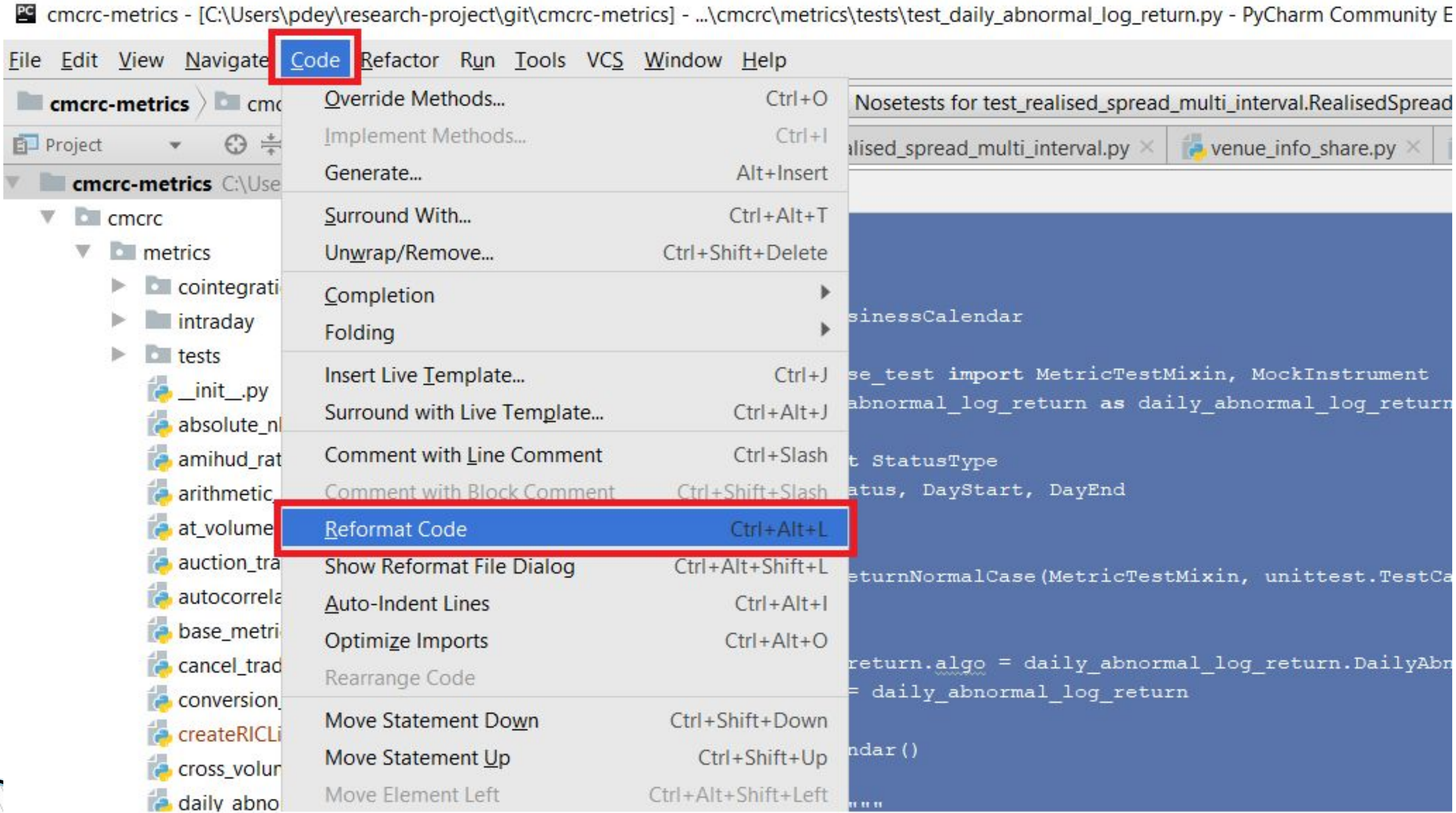
- Are we missing any trading phases?

- Are we using multiple nested loops causing potential performance bottleneck?

- What if any data is not available – exceptions scenarios? Is metric code robust to both liquid and illiquid stocks.

Coding Best Practices and Due Diligence (Cont...)

Are we following appropriate code style/formatting (Pycharm IDE can help you!)


cmcrc-metrics - [C:\Users\pdey\research-project\git\cmcrc-metrics] - ...\cmcrc\metrics\tests\test_daily_abnormal_log_return.py - PyCharm Community E
File
Edit
View
Navigate
Code
Refactor
Run
Tools
VCS
Window
Help
cmcrc-metrics
cmc
Project
cmcrc-metrics C:\Use
cmcrc
metrics
cointegrati
intraday
tests
__init__.py
absolute_n
amihud_rat
arithmetic
at_volume
auction_tra
autocorrela
base_metri
cancel_trad
conversion
createRICLi
cross_volur
daily_abno
Override Methods... Ctrl+O
Implement Methods... Ctrl+I
Generate... Alt+Insert
Surround With... Ctrl+Alt+T
Unwrap/Remove... Ctrl+Shift+Delete
Completion
Folding
Insert Live Template... Ctrl+J
Surround with Live Template... Ctrl+Alt+J
Comment with Line Comment Ctrl+Slash
Comment with Block Comment Ctrl+Shift+Slash
Reformat Code Ctrl+Alt+L
Show Reformat File Dialog Ctrl+Alt+Shift+L
Auto-Indent Lines Ctrl+Alt+I
Optimize Imports Ctrl+Alt+O
Rearrange Code
Move Statement Down Ctrl+Shift+Down
Move Statement Up Ctrl+Shift+Up
Move Element Left Ctrl+Alt+Shift+Left
Nosetests for test_realised_spread_multi_interval.RealisedSpread
alised_spread_multi_interval.py
venue_info_share.py

```
sinessCalendar

se_test import MetricTestMixin, MockInstrument
abnormal_log_return as daily_abnormal_log_return

t StatusType
atus, DayStart, DayEnd

eturnNormalCase(MetricTestMixin, unittest.TestCa

return.algo = daily_abnormal_log_return.DailyAbn
= daily_abnormal_log_return

ndar()

"""
```

Metrics Case Study Examples

Metric to measure Quote Volatility and Effective Spread:

- Which uptick functions we need to use?
- What are the key user defined variables and uptick pre-defined variables that are required?
- What are the key considerations for using trade and quote?

  Handling one sided quote, invalid spread.
- Which trading phases are involved in the computation?
- What happens when there is trade halt/suspension?
- How do we perform the computations? For example, we "online algorithms" for standard deviations, autocorrelation could be faster.

Example Exception checks:

- Is the quote valid (ask >bid)
- What happens if a stock has few sample sizes?
- Singular matrix problems (for example price series for illiquid stocks)

Quote Volatility code, Effective Spread Code

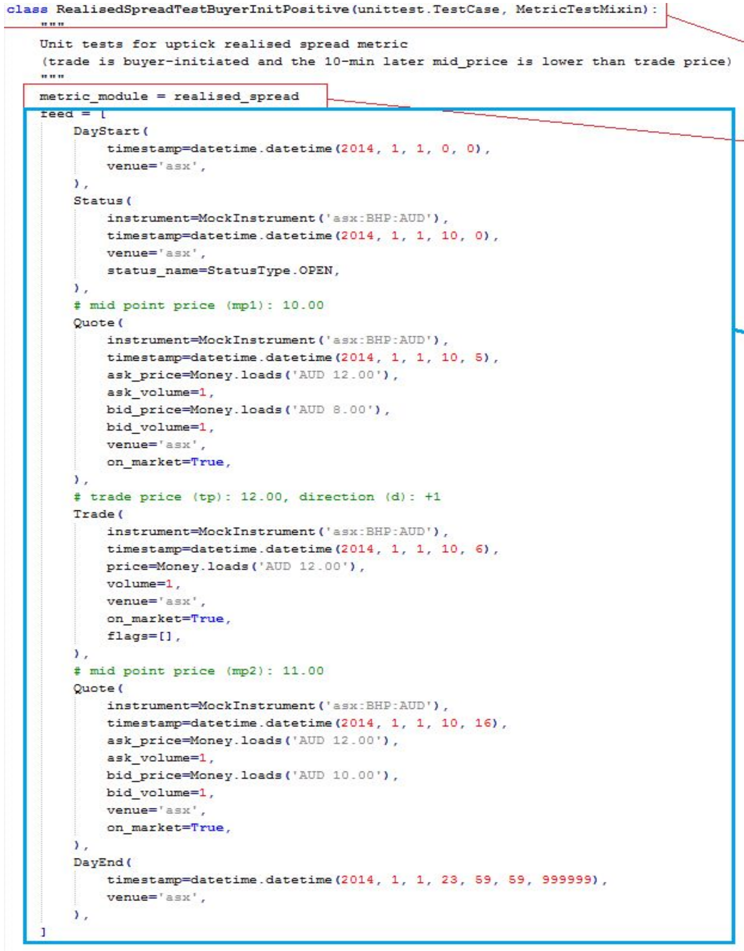
Metrics Test Cases

Metrics Test Cases

- Why unit test matters?
  - Minimize bugs in the code.
  - Make code robust in handling various input scenarios.
  - Improves quality of code.

- Creating metrics test cases and key elements of the test case:
  - Create new test case starting with test_<metric_file_name>.py.
  - Define test case python class per test scenario.
  - What should test scenario cover?
    - Given test data feed, what should be the expected output?
  - How many scenarios? Break up metric code into logical subparts – does test case at least covers key scenarios handled by the code? Less unit test coverage could lead to more issues during integration testing and end output.

Test Case Elements



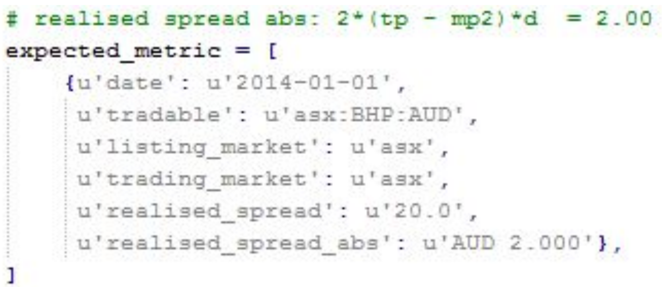
```
class RealisedSpreadTestBuyerInitPositive(unittest.TestCase, MetricTestMixin):
    """
    Unit tests for uptick realised spread metric
    (trade is buyer-initiated and the 10-min later mid_price is lower than trade price)
    """
    metric_module = realised_spread
    feed = [
        DayStart(
            timestamp=datetime.datetime(2014, 1, 1, 0, 0),
            venue='asx',
        ),
        Status(
            instrument=MockInstrument('asx:BHP:AUD'),
            timestamp=datetime.datetime(2014, 1, 1, 10, 0),
            venue='asx',
            status_name=StatusType.OPEN,
        ),
        # mid point price (mp1): 10.00
        Quote(
            instrument=MockInstrument('asx:BHP:AUD'),
            timestamp=datetime.datetime(2014, 1, 1, 10, 5),
            ask_price=Money.loads('AUD 12.00'),
            ask_volume=1,
            bid_price=Money.loads('AUD 8.00'),
            bid_volume=1,
            venue='asx',
            on_market=True,
        ),
        # trade price (tp): 12.00, direction (d): +1
        Trade(
            instrument=MockInstrument('asx:BHP:AUD'),
            timestamp=datetime.datetime(2014, 1, 1, 10, 6),
            price=Money.loads('AUD 12.00'),
            volume=1,
            venue='asx',
            on_market=True,
            flags=[],
        ),
        # mid point price (mp2): 11.00
        Quote(
            instrument=MockInstrument('asx:BHP:AUD'),
            timestamp=datetime.datetime(2014, 1, 1, 10, 16),
            ask_price=Money.loads('AUD 12.00'),
            ask_volume=1,
            bid_price=Money.loads('AUD 10.00'),
            bid_volume=1,
            venue='asx',
            on_market=True,
        ),
        DayEnd(
            timestamp=datetime.datetime(2014, 1, 1, 23, 59, 59, 999999),
            venue='asx',
        ),
    ]
```

Test case 'Class'. Represents Test Scenario. Extends unittest.TestCase (Python inheritence)

Metric file name where the metric is coded. This test case will test this metric.

Input data feed when which metric will be tested

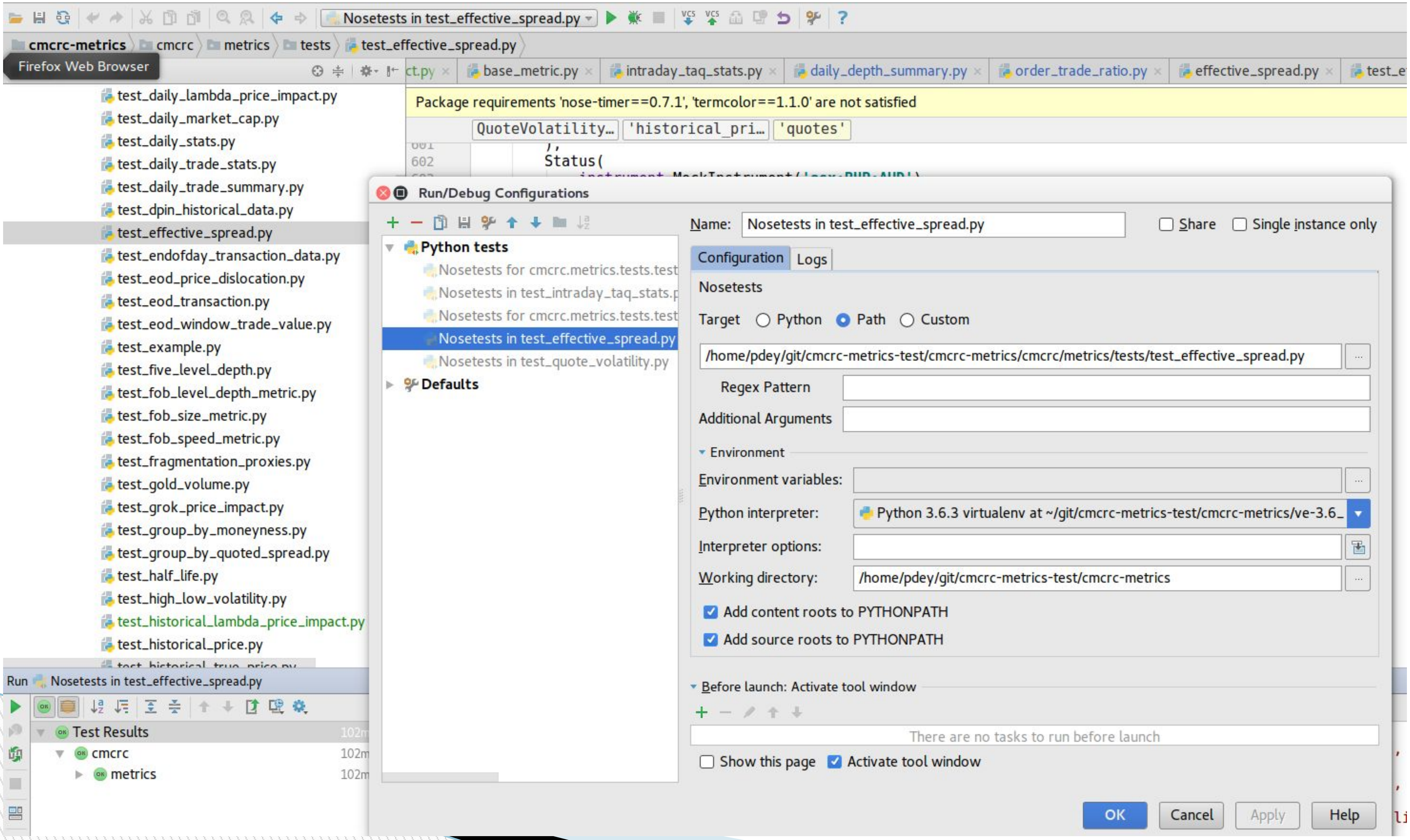
Expected output



```
# realised spread abs: 2*(tp - mp2)*d  = 2.00
expected_metric = [
    {u'date': u'2014-01-01',
     u'tradable': u'asx:BHP:AUD',
     u'listing_market': u'asx',
     u'trading_market': u'asx',
     u'realised_spread': u'20.0',
     u'realised_spread_abs': u'AUD 2.000'},
]
```

Always manually verify the expected output values based on your test data!

Running Test Case



test_daily_lambda_price_impact.py
test_daily_market_cap.py
test_daily_stats.py
test_daily_trade_stats.py
test_daily_trade_summary.py
test_dpin_historical_data.py
test_effective_spread.py
test_endofday_transaction_data.py
test_eod_price_dislocation.py
test_eod_transaction.py
test_eod_window_trade_value.py
test_example.py
test_five_level_depth.py
test_fob_level_depth_metric.py
test_fob_size_metric.py
test_fob_speed_metric.py
test_fragmentation_proxies.py
test_gold_volume.py
test_grok_price_impact.py
test_group_by_moneyness.py
test_group_by_quoted_spread.py
test_half_life.py
test_high_low_volatility.py
test_historical_lambda_price_impact.py
test_historical_price.py
test_historical_true_price.py

Package requirements 'nose-timer==0.7.1', 'termcolor==1.1.0' are not satisfied

```
QuoteVolatility…  'historical_pri…  'quotes'
602         Status(
```

Run/Debug Configurations

- Nosetests for cmcrc.metrics.tests.test
- Nosetests in test_intraday_taq_stats.p
- Nosetests for cmcrc.metrics.tests.test
- Nosetests in test_effective_spread.py
- Nosetests in test_quote_volatility.py

Name: Nosetests in test_effective_spread.py

Nosetests

Target ○ Python ● Path ○ Custom

/home/pdey/git/cmcrc-metrics-test/cmcrc-metrics/cmcrc/metrics/tests/test_effective_spread.py

Regex Pattern

Additional Arguments

Environment

Environment variables:

Python interpreter: Python 3.6.3 virtualenv at ~/git/cmcrc-metrics-test/cmcrc-metrics/ve-3.6_

Interpreter options:

Working directory: /home/pdey/git/cmcrc-metrics-test/cmcrc-metrics

☑ Add content roots to PYTHONPATH

☑ Add source roots to PYTHONPATH

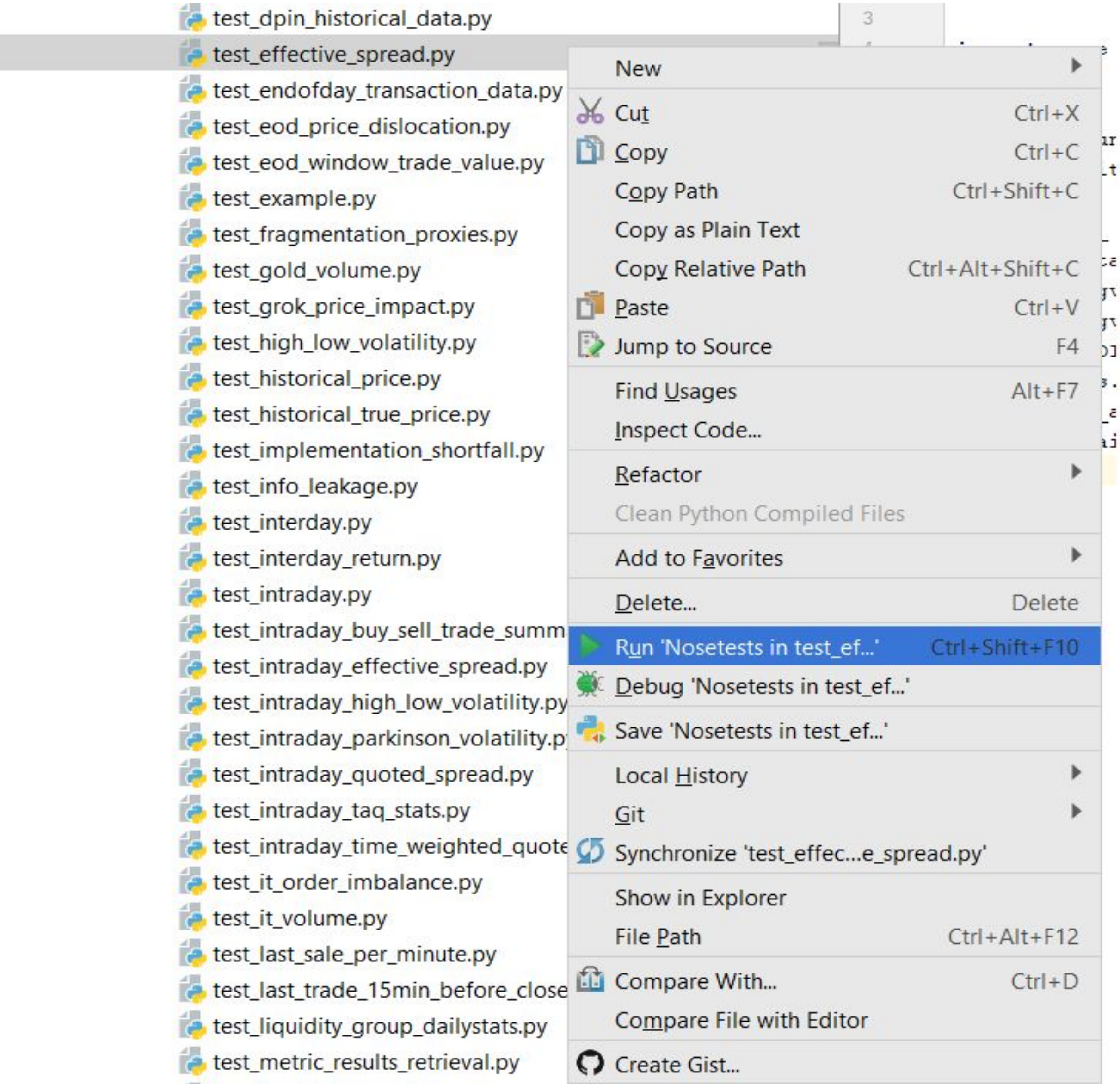
Before launch: Activate tool window

There are no tasks to run before launch

□ Show this page ☑ Activate tool window

Run Nosetests in test_effective_spread.py

Running Test Case (Cont...)


test_dpin_historical_data.py
test_effective_spread.py
test_endofday_transaction_data.py
test_eod_price_dislocation.py
test_eod_window_trade_value.py
test_example.py
test_fragmentation_proxies.py
test_gold_volume.py
test_grok_price_impact.py
test_high_low_volatility.py
test_historical_price.py
test_historical_true_price.py
test_implementation_shortfall.py
test_info_leakage.py
test_interday.py
test_interday_return.py
test_intraday.py
test_intraday_buy_sell_trade_summ
test_intraday_effective_spread.py
test_intraday_high_low_volatility.py
test_intraday_parkinson_volatility.p
test_intraday_quoted_spread.py
test_intraday_taq_stats.py
test_intraday_time_weighted_quote
test_it_order_imbalance.py
test_it_volume.py
test_last_sale_per_minute.py
test_last_trade_15min_before_close
test_liquidity_group_dailystats.py
test_metric_results_retrieval.py
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy as Plain Text
Copy Relative Path Ctrl+Alt+Shift+C
Paste Ctrl+V
Jump to Source F4
Find Usages Alt+F7
Inspect Code...
Refactor
Clean Python Compiled Files
Add to Favorites
Delete... Delete
Run 'Nosetests in test_ef...' Ctrl+Shift+F10
Debug 'Nosetests in test_ef...'
Save 'Nosetests in test_ef...'
Local History
Git
Synchronize 'test_effec...e_spread.py'
Show in Explorer
File Path Ctrl+Alt+F12
Compare With... Ctrl+D
Compare File with Editor
Create Gist...

Test Scenarios case study (Effective Spread)

- What is Effective Spread MQD Metric? It measures how much the actual trade price deviates from the mid-point. It represents the actual, round-trip cost of trading to the liquidity demander.

  $$EffSpr_{v,s,d} = \sum 200 * D_{v,s,d,t} * \frac{P_{v,s,d,t} - Mid_{v,s,d,t}}{Mid_{v,s,d,t}} * W_{v,s,d,t}$$

  P = trade price, M = mid point price, W = value weights, D = direction of trade

- What scenarios you can think of?

  What should be the Expected output when market has:

  - Both buy and sell trades
  - Only buy trades or only sell trades
  - No trades
  - Holiday
  - What could be other scenario?

- Input feed

  a) Check if it is dependant on any other metrics?

  b) Create test data feed

- What could be output Attribute?

Click here to go to the test code

Test Scenarios case study (Quote Volatility)

- What is Quote Volatility MQD Metric?

  Standard deviation of 1 minute midpoint price return.

- What scenarios you can think of?

  What should be the expected output when market has:
  - Series of on-market quotes (simple case)
  - Invalid spread (example: bid>=ask)
  - Bid or Ask side is unavailable
  - What could be other scenarios?
- Input feed
  - Check if it is dependant on any other metrics?
  - Create test data feed
- What could be output Attribute?

Click here to go to the test code

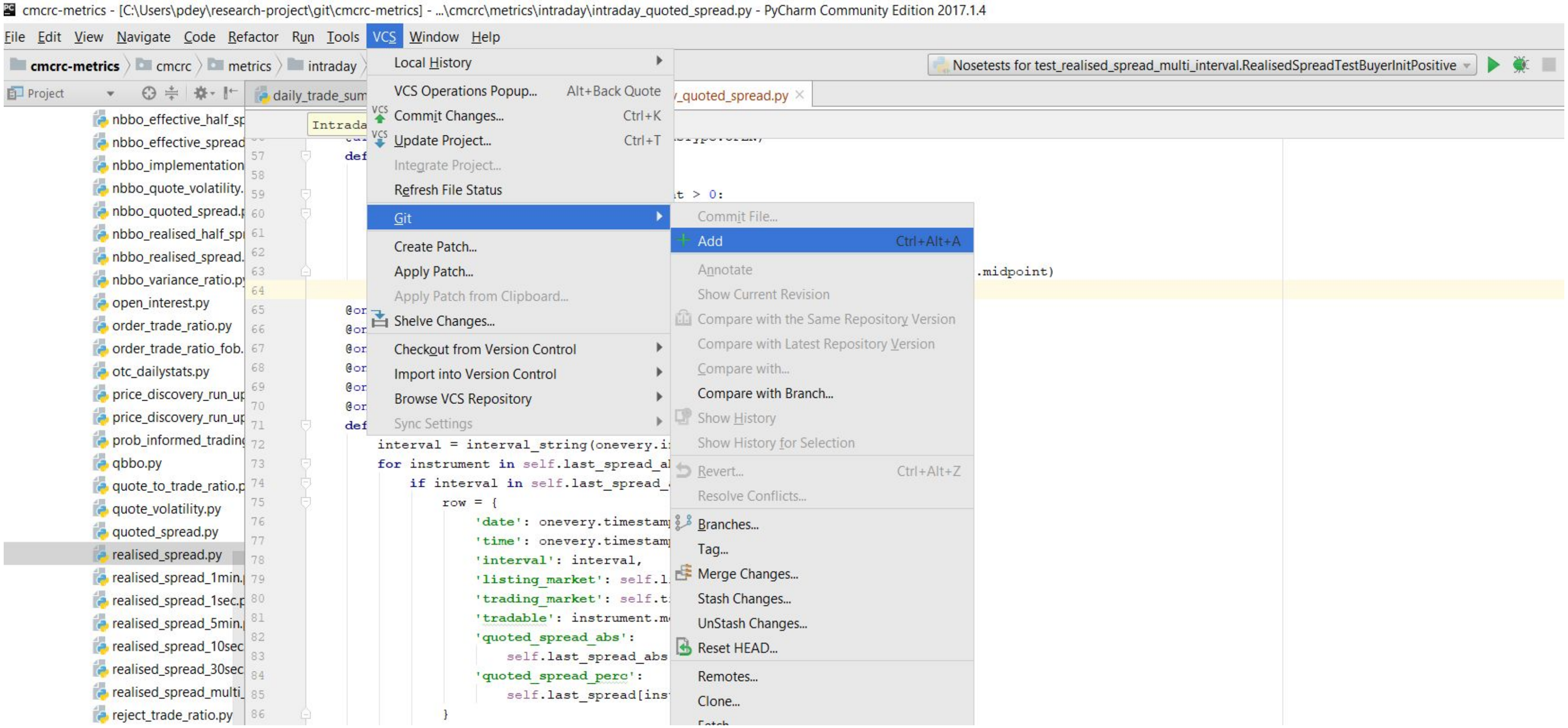
Review and Deployment Process

- Ensure code is complete and unit tested.
- Mark JIRA status "In Progress" before committing the changes" to git repository.
- Submitting changes to code repository for review. Tutorial about learning about Git tool in pycharm and git (in general), please refer to "https://www.jetbrains.com/help/pycharm/using-git-tutorial.html#d387025e2"
  - Use git to commit and push the changes
  - Add appropriate code comments including the JIRA number.
- Code is submitted for review in bit-bucket tool. The changes will be merged to Master repository once reviewed/accepted by all code reviewers.

Bit bucket link

Submitting Metric Changes

If new file, then perform git–>add, else directly perform git “commit” or “git push” so that changes are submitted to remote repository

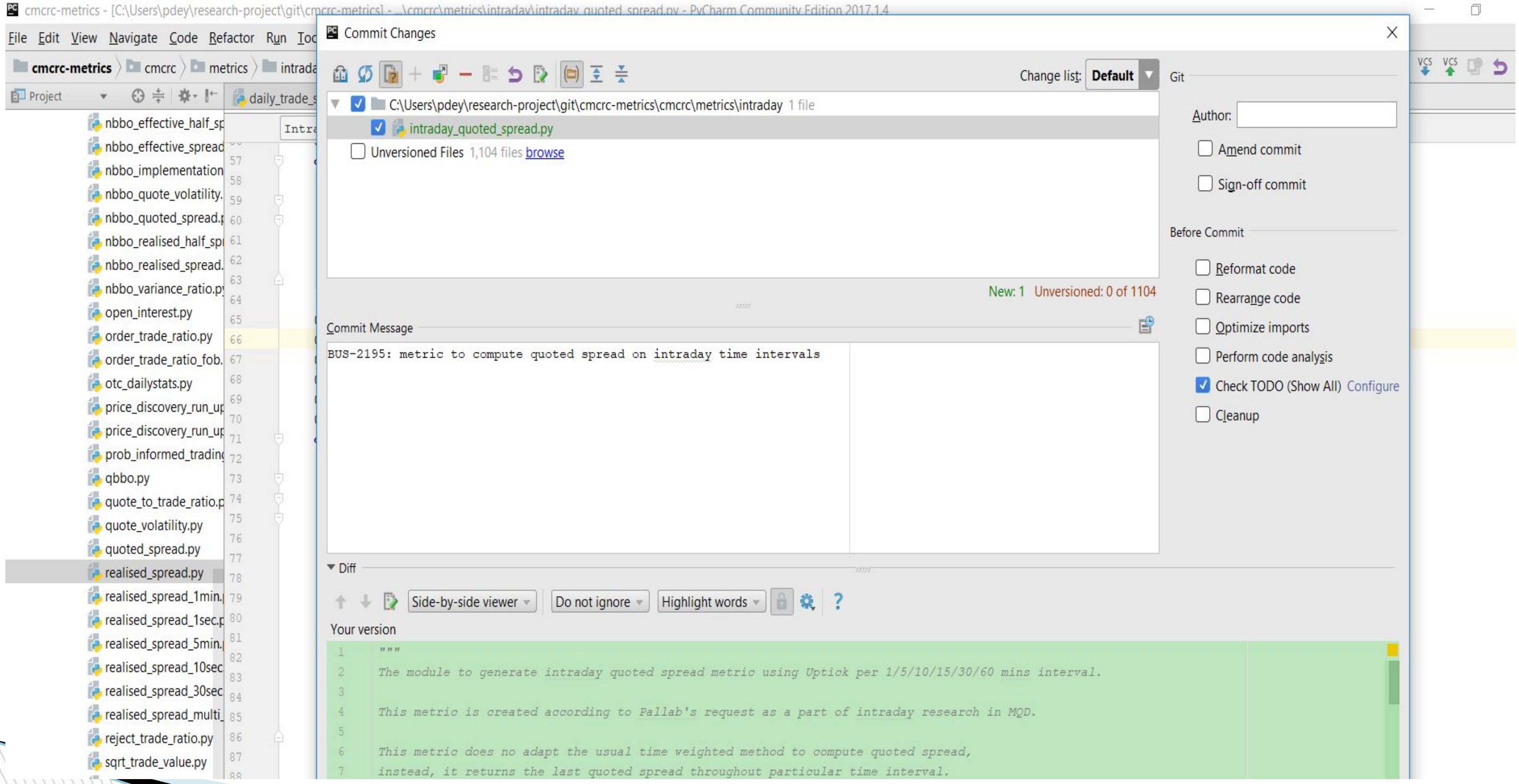


cmcrc-metrics - [C:\Users\pdey\research-project\git\cmcrc-metrics] - ...\cmcrc\metrics\intraday\intraday_quoted_spread.py - PyCharm Community Edition 2017.1.4

```
interval = interval_string(onevery.i
for instrument in self.last_spread_a
    if interval in self.last_spread_
        row = {
            'date': onevery.timestam
            'time': onevery.timestam
            'interval': interval,
            'listing_market': self.l
            'trading_market': self.t
            'tradable': instrument.m
            'quoted_spread_abs':
                self.last_spread_abs
            'quoted_spread_perc':
                self.last_spread[ins
        }
```

Submitting Metric Changes (Cont..)

Sample git commit operation. The commit message should start with JIRA name, colon “:” followed by user defined commit message. Commit message should briefly highlights the change



Commit Changes

Change list: Default

☑ C:\Users\pdey\research-project\git\cmcrc-metrics\cmcrc\metrics\intraday 1 file

☑ intraday_quoted_spread.py

□ Unversioned Files 1,104 files browse

New: 1 Unversioned: 0 of 1104

Commit Message

```
BUS-2195: metric to compute quoted spread on intraday time intervals
```

Git

Author:

□ Amend commit

□ Sign-off commit

Before Commit

□ Reformat code

□ Rearrange code

□ Optimize imports

□ Perform code analysis

☑ Check TODO (Show All) Configure

□ Cleanup

Diff

Your version

```
"""
The module to generate intraday quoted spread metric using Uptick per 1/5/10/15/30/60 mins interval.

This metric is created according to Pallab's request as a part of intraday research in MQD.

This metric does no adapt the usual time weighted method to compute quoted spread,
instead, it returns the last quoted spread throughout particular time interval.
```

Mqdashboard Analytics

MQD Analytics

What is mqd analytics?

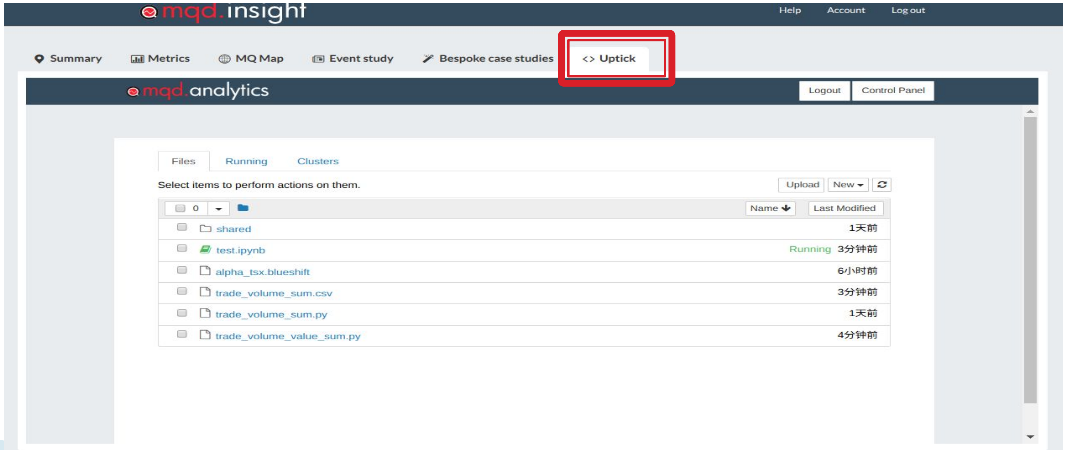
It is a python framework in a jupyter platform for financial metric calculation and result display

What is Jupyter?

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code

https://jupyter.readthedocs.io/en/latest/

How to use mqd analytics?



Select items to perform actions on them.

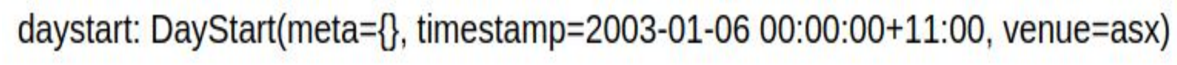
```
shared                          1天前
test.ipynb              Running 3分钟前
alpha_tsx.blueshift             6小时前
trade_volume_sum.csv            3分钟前
trade_volume_sum.py             1天前
trade_volume_value_sum.py       4分钟前
```

How can you do research in mqd analytic?

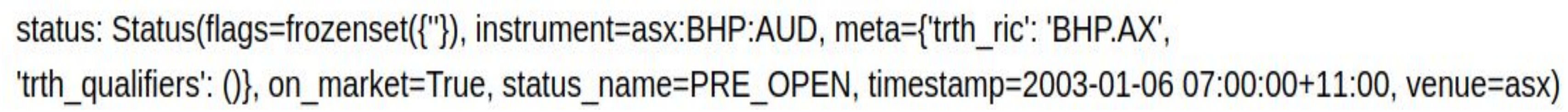
1. Read and investigate converted data
2. Write New Metric or obtain existing metric result directly
3. Customized research report

1. Read and Investigate Data

- 1) Day Start

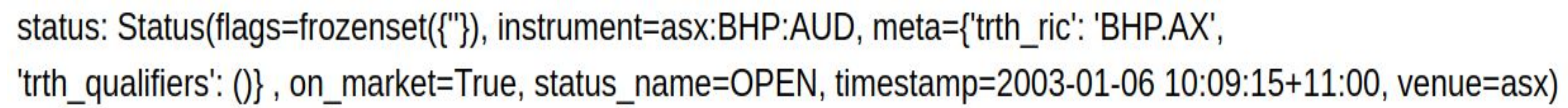


```
daystart: DayStart(meta={}, timestamp=2003-01-06 00:00:00+11:00, venue=asx)
```



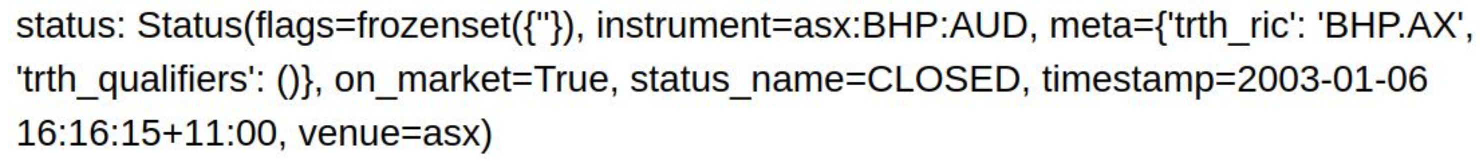
```
status: Status(flags=frozenset({''}), instrument=asx:BHP:AUD, meta={'trth_ric': 'BHP.AX',
'trth_qualifiers': ()}, on_market=True, status_name=PRE_OPEN, timestamp=2003-01-06 07:00:00+11:00, venue=asx)
```

- 2) Open



```
status: Status(flags=frozenset({''}), instrument=asx:BHP:AUD, meta={'trth_ric': 'BHP.AX',
'trth_qualifiers': ()} , on_market=True, status_name=OPEN, timestamp=2003-01-06 10:09:15+11:00, venue=asx)
```

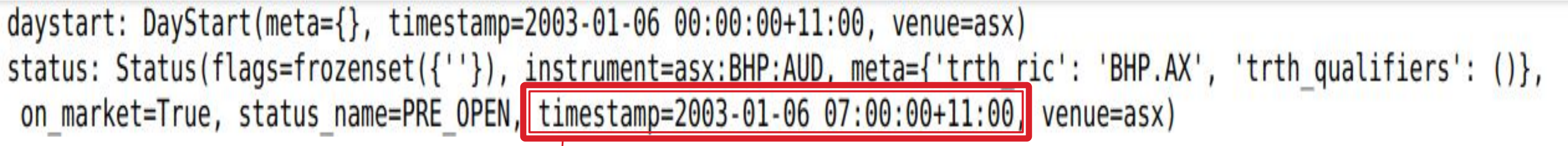
- 3) Closed



```
status: Status(flags=frozenset({''}), instrument=asx:BHP:AUD, meta={'trth_ric': 'BHP.AX',
'trth_qualifiers': ()}, on_market=True, status_name=CLOSED, timestamp=2003-01-06
16:16:15+11:00, venue=asx)
```

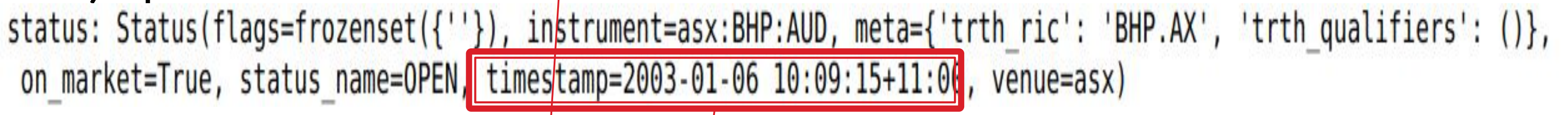
1. Read and Investigate Data (Cont..)

- 1) Day Start and Pre-Open



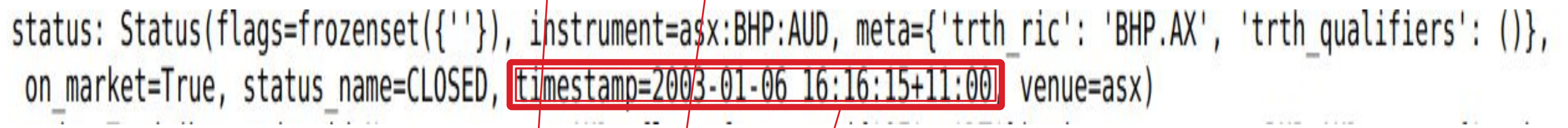
```
daystart: DayStart(meta={}, timestamp=2003-01-06 00:00:00+11:00, venue=asx)
status: Status(flags=frozenset({''}), instrument=asx:BHP:AUD, meta={'trth_ric': 'BHP.AX', 'trth_qualifiers': ()},
 on_market=True, status_name=PRE_OPEN, timestamp=2003-01-06 07:00:00+11:00, venue=asx)
```

- 2) Open

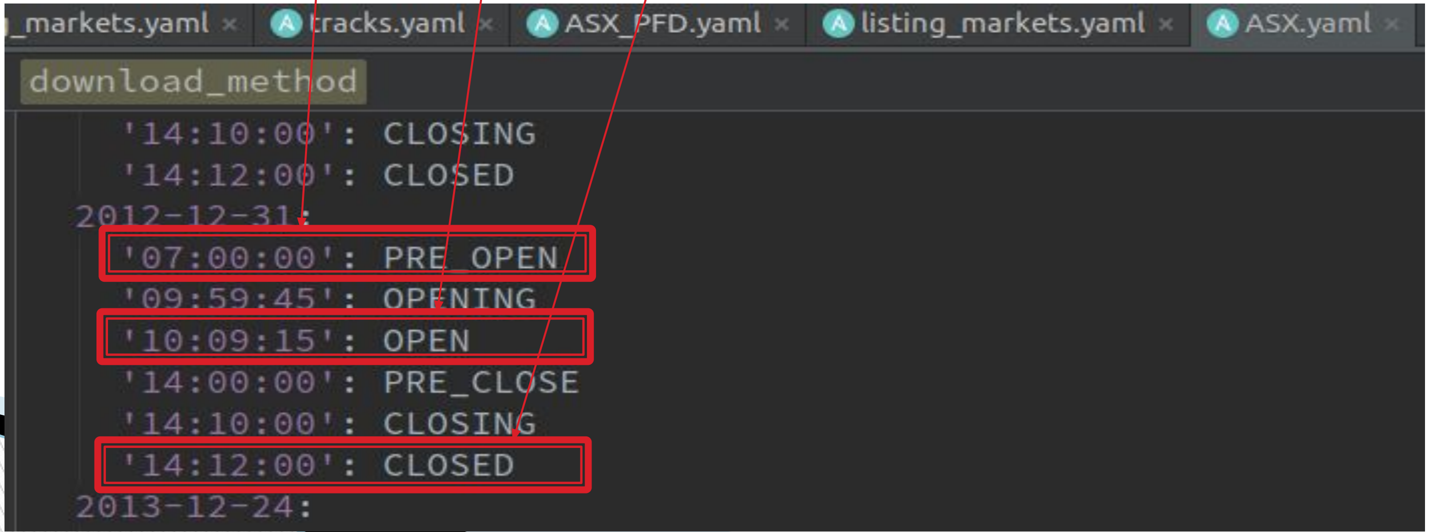


```
status: Status(flags=frozenset({''}), instrument=asx:BHP:AUD, meta={'trth_ric': 'BHP.AX', 'trth_qualifiers': ()},
 on_market=True, status_name=OPEN, timestamp=2003-01-06 10:09:15+11:00, venue=asx)
```

- 3) Closed



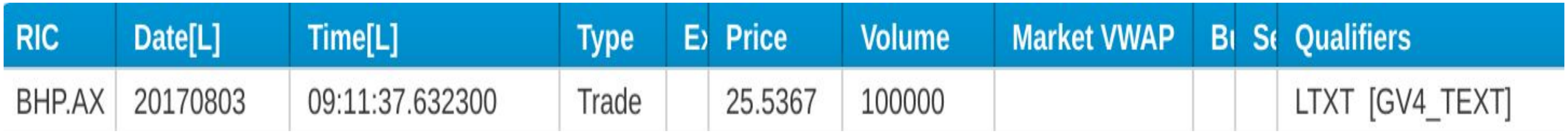
```
status: Status(flags=frozenset({''}), instrument=asx:BHP:AUD, meta={'trth_ric': 'BHP.AX', 'trth_qualifiers': ()},
 on_market=True, status_name=CLOSED, timestamp=2003-01-06 16:16:15+11:00, venue=asx)
```



```
    '14:10:00': CLOSING
    '14:12:00': CLOSED
  2012-12-31:
    '07:00:00': PRE_OPEN
    '09:59:45': OPENING
    '10:09:15': OPEN
    '14:00:00': PRE_CLOSE
    '14:10:00': CLOSING
    '14:12:00': CLOSED
  2013-12-24:
```

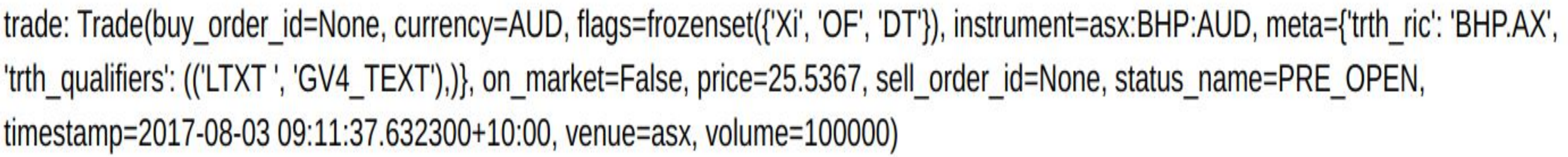
1. Read and Investigate Data (Cont..)

- 4) Trade
- A) From TRTH



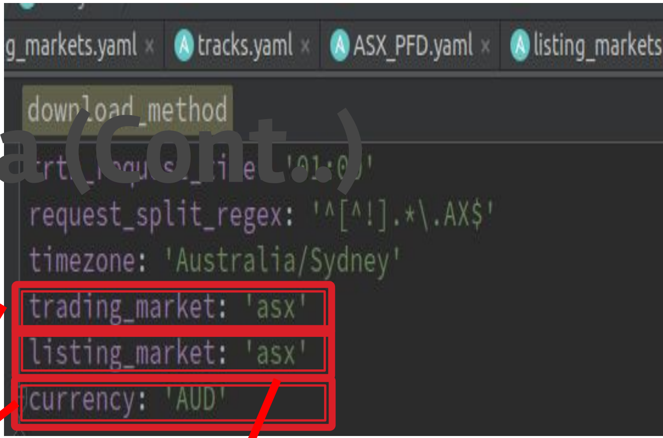
RIC	Date[L]	Time[L]	Type	Ex	Price	Volume	Market VWAP	B	Se	Qualifiers
BHP.AX	20170803	09:11:37.632300	Trade		25.5367	100000				LTXT [GV4_TEXT]

- B) From blueshift



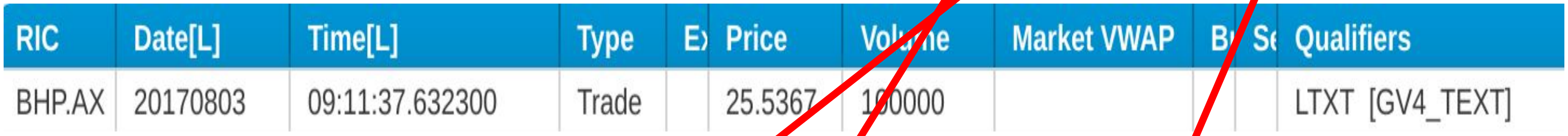
```
trade: Trade(buy_order_id=None, currency=AUD, flags=frozenset({'Xi', 'OF', 'DT'}), instrument=asx:BHP:AUD, meta={'trth_ric': 'BHP.AX',
'trth_qualifiers': (('LTXT ', 'GV4_TEXT'),)}, on_market=False, price=25.5367, sell_order_id=None, status_name=PRE_OPEN,
timestamp=2017-08-03 09:11:37.632300+10:00, venue=asx, volume=100000)
```

1. Read and Investigate Data (Cont.)



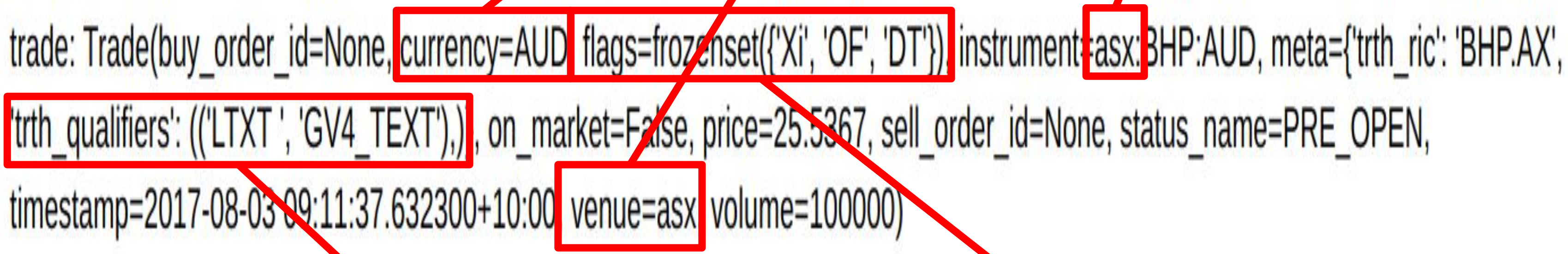
```
g_markets.yaml × | tracks.yaml × | ASX_PFD.yaml × | listing_markets
download_method
trth_request_time: '01:00'
request_split_regex: '^[^!].*\.AX$'
timezone: 'Australia/Sydney'
trading_market: 'asx'
listing_market: 'asx'
currency: 'AUD'
```

- 4) Trade
- A) From TRTH

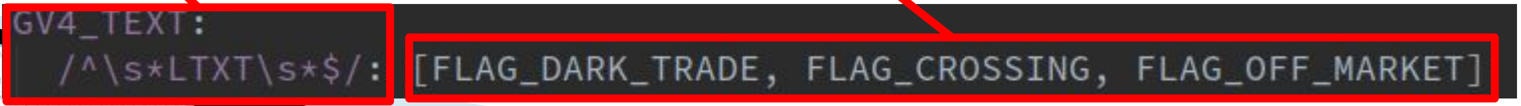


RIC	Date[L]	Time[L]	Type	E>	Price	Volume	Market VWAP	B	Se	Qualifiers
BHP.AX	20170803	09:11:37.632300	Trade		25.5367	100000				LTXT [GV4_TEXT]

- B) From blueshift



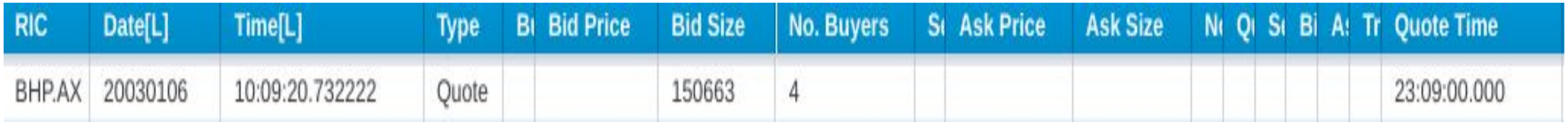
```
trade: Trade(buy_order_id=None, currency=AUD, flags=frozenset({'Xi', 'OF', 'DT'}), instrument=asx:BHP:AUD, meta={'trth_ric': 'BHP.AX',
'trth_qualifiers': (('LTXT ', 'GV4_TEXT'),)}, on_market=False, price=25.5367, sell_order_id=None, status_name=PRE_OPEN,
timestamp=2017-08-03 09:11:37.632300+10:00, venue=asx, volume=100000)
```



```
GV4_TEXT:
  /^\s*LTXT\s*$/: [FLAG_DARK_TRADE, FLAG_CROSSING, FLAG_OFF_MARKET]
```

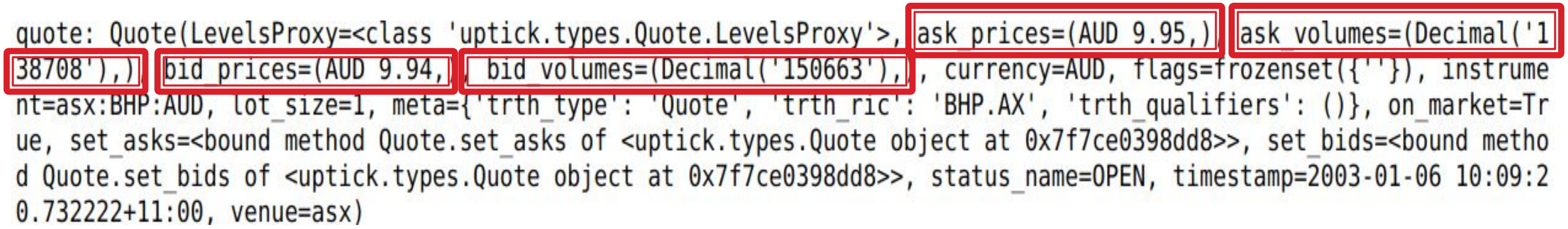
1. Read and Investigate Data (Cont..)

- 5) Quote
- A) From TRTH



RIC	Date[L]	Time[L]	Type	Bi	Bid Price	Bid Size	No. Buyers	S	Ask Price	Ask Size	N	Q	S	Bi	A	Tr	Quote Time
BHP.AX	20030106	10:09:20.732222	Quote			150663	4										23:09:00.000

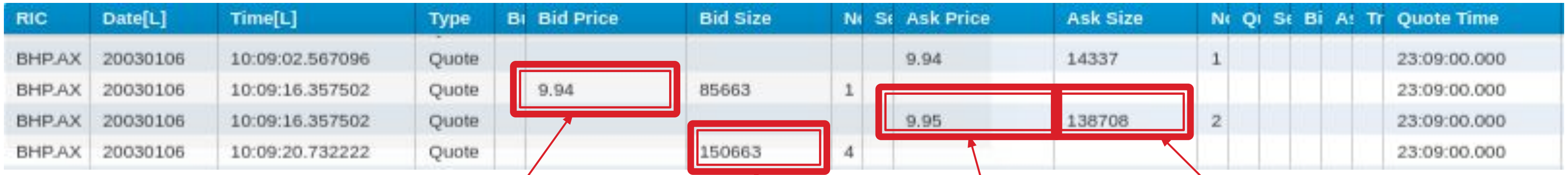
- B) From blueshift



```
quote: Quote(LevelsProxy=<class 'uptick.types.Quote.LevelsProxy'>, ask_prices=(AUD 9.95,), ask_volumes=(Decimal('1
38708'),), bid_prices=(AUD 9.94,), bid_volumes=(Decimal('150663'),), currency=AUD, flags=frozenset({''}), instrume
nt=asx:BHP:AUD, lot_size=1, meta={'trth_type': 'Quote', 'trth_ric': 'BHP.AX', 'trth_qualifiers': ()}, on_market=Tr
ue, set_asks=<bound method Quote.set_asks of <uptick.types.Quote object at 0x7f7ce0398dd8>>, set_bids=<bound metho
d Quote.set_bids of <uptick.types.Quote object at 0x7f7ce0398dd8>>, status_name=OPEN, timestamp=2003-01-06 10:09:2
0.732222+11:00, venue=asx)
```

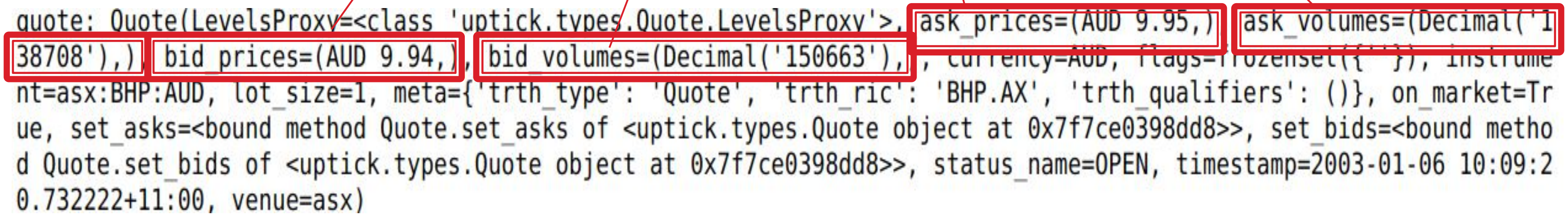
1. Read and Investigate Data (Cont..)

- 5) Quote
- A) From TRTH



RIC	Date[L]	Time[L]	Type	Bı	Bid Price	Bid Size	Nı	Sı	Ask Price	Ask Size	Nı	Qı	Sı	Bi	A:	Tr	Quote Time
BHP.AX	20030106	10:09:02.567096	Quote						9.94	14337	1						23:09:00.000
BHP.AX	20030106	10:09:16.357502	Quote		9.94	85663	1										23:09:00.000
BHP.AX	20030106	10:09:16.357502	Quote						9.95	138708	2						23:09:00.000
BHP.AX	20030106	10:09:20.732222	Quote			150663	4										23:09:00.000

- B) From blueshift



```
quote: Quote(LevelsProxy=<class 'uptick.types.Quote.LevelsProxy'>, ask_prices=(AUD 9.95,), ask_volumes=(Decimal('1
38708'),), bid_prices=(AUD 9.94,), bid_volumes=(Decimal('150663'),), currency=AUD, flags=frozenset({''}), instrume
nt=asx:BHP:AUD, lot_size=1, meta={'trth_type': 'Quote', 'trth_ric': 'BHP.AX', 'trth_qualifiers': ()}, on_market=Tr
ue, set_asks=<bound method Quote.set_asks of <uptick.types.Quote object at 0x7f7ce0398dd8>>, set_bids=<bound metho
d Quote.set_bids of <uptick.types.Quote object at 0x7f7ce0398dd8>>, status_name=OPEN, timestamp=2003-01-06 10:09:2
0.732222+11:00, venue=asx)
```

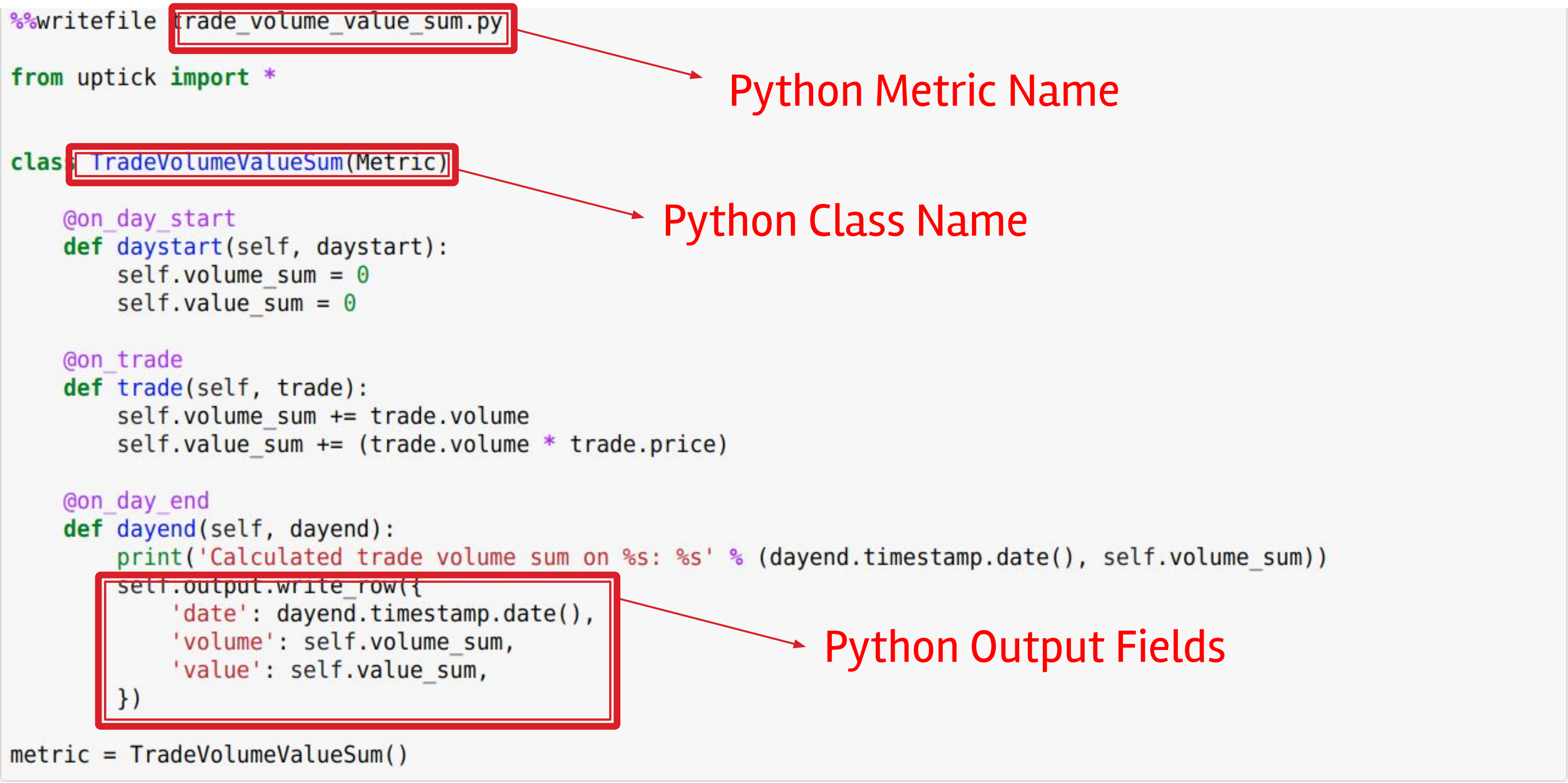
2. Obtain Metric Result

1. Write a new metric on blueshift or your own data

2. Obtain existing metric result stored in mqdashboard data base

2. Obtain Metric Result (Cont..)

1) Write and run your own metric



```
%%writefile trade_volume_value_sum.py

from uptick import *


class TradeVolumeValueSum(Metric)

    @on_day_start
    def daystart(self, daystart):
        self.volume_sum = 0
        self.value_sum = 0

    @on_trade
    def trade(self, trade):
        self.volume_sum += trade.volume
        self.value_sum += (trade.volume * trade.price)

    @on_day_end
    def dayend(self, dayend):
        print('Calculated trade volume sum on %s: %s' % (dayend.timestamp.date(), self.volume_sum))
        self.output.write_row({
            'date': dayend.timestamp.date(),
            'volume': self.volume_sum,
            'value': self.value_sum,
        })

metric = TradeVolumeValueSum()
```

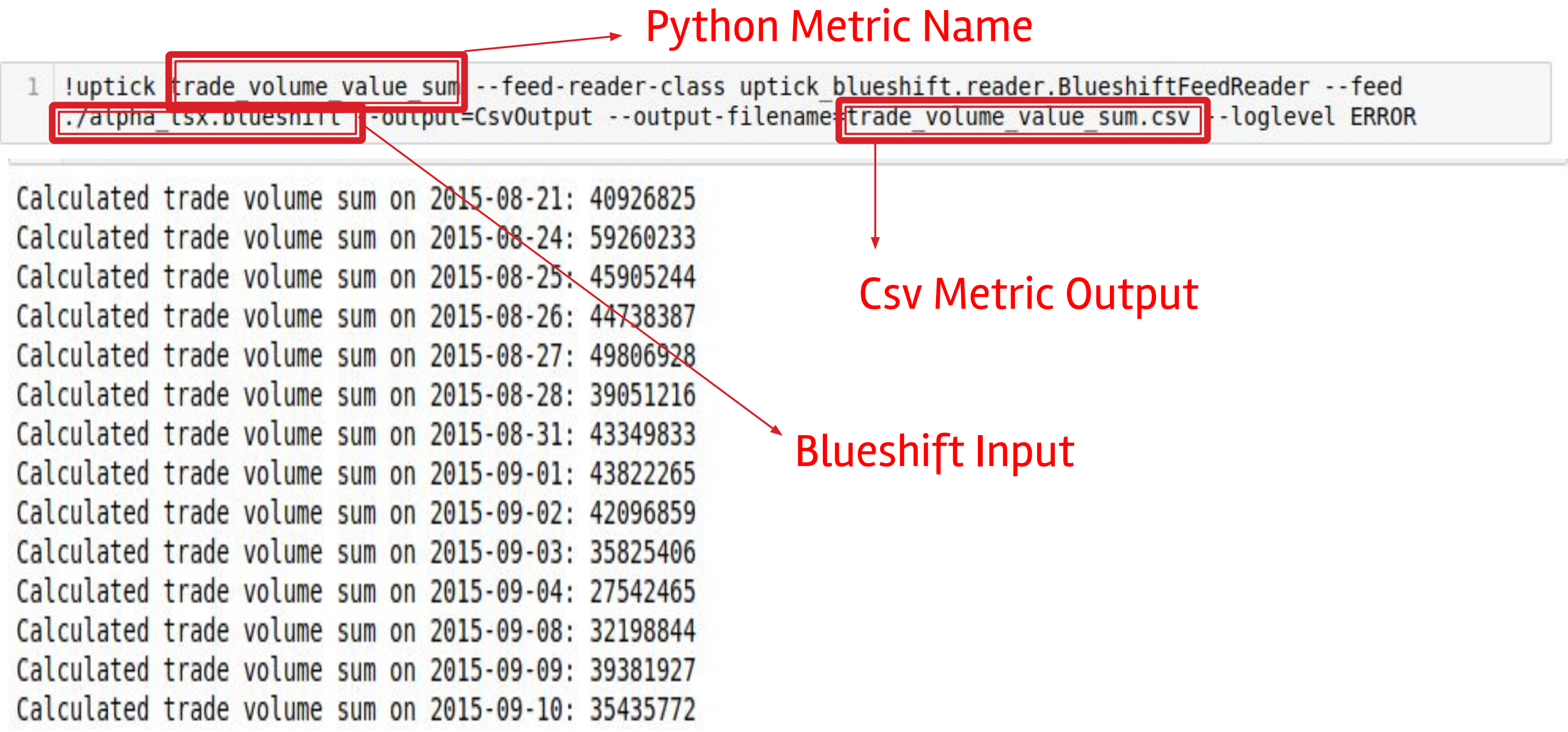
Python Metric Name

Python Class Name

Python Output Fields

2. Obtain Metric Result (Cont..)

1) Write and run your own metric



```
!uptick trade_volume_value_sum --feed-reader-class uptick_blueshift.reader.BlueshiftFeedReader --feed
./alpha_tsx.blueshift --output=CsvOutput --output-filename=trade_volume_value_sum.csv --loglevel ERROR
```

```
Calculated trade volume sum on 2015-08-21: 40926825
Calculated trade volume sum on 2015-08-24: 59260233
Calculated trade volume sum on 2015-08-25: 45905244
Calculated trade volume sum on 2015-08-26: 44738387
Calculated trade volume sum on 2015-08-27: 49806928
Calculated trade volume sum on 2015-08-28: 39051216
Calculated trade volume sum on 2015-08-31: 43349833
Calculated trade volume sum on 2015-09-01: 43822265
Calculated trade volume sum on 2015-09-02: 42096859
Calculated trade volume sum on 2015-09-03: 35825406
Calculated trade volume sum on 2015-09-04: 27542465
Calculated trade volume sum on 2015-09-08: 32198844
Calculated trade volume sum on 2015-09-09: 39381927
Calculated trade volume sum on 2015-09-10: 35435772
```

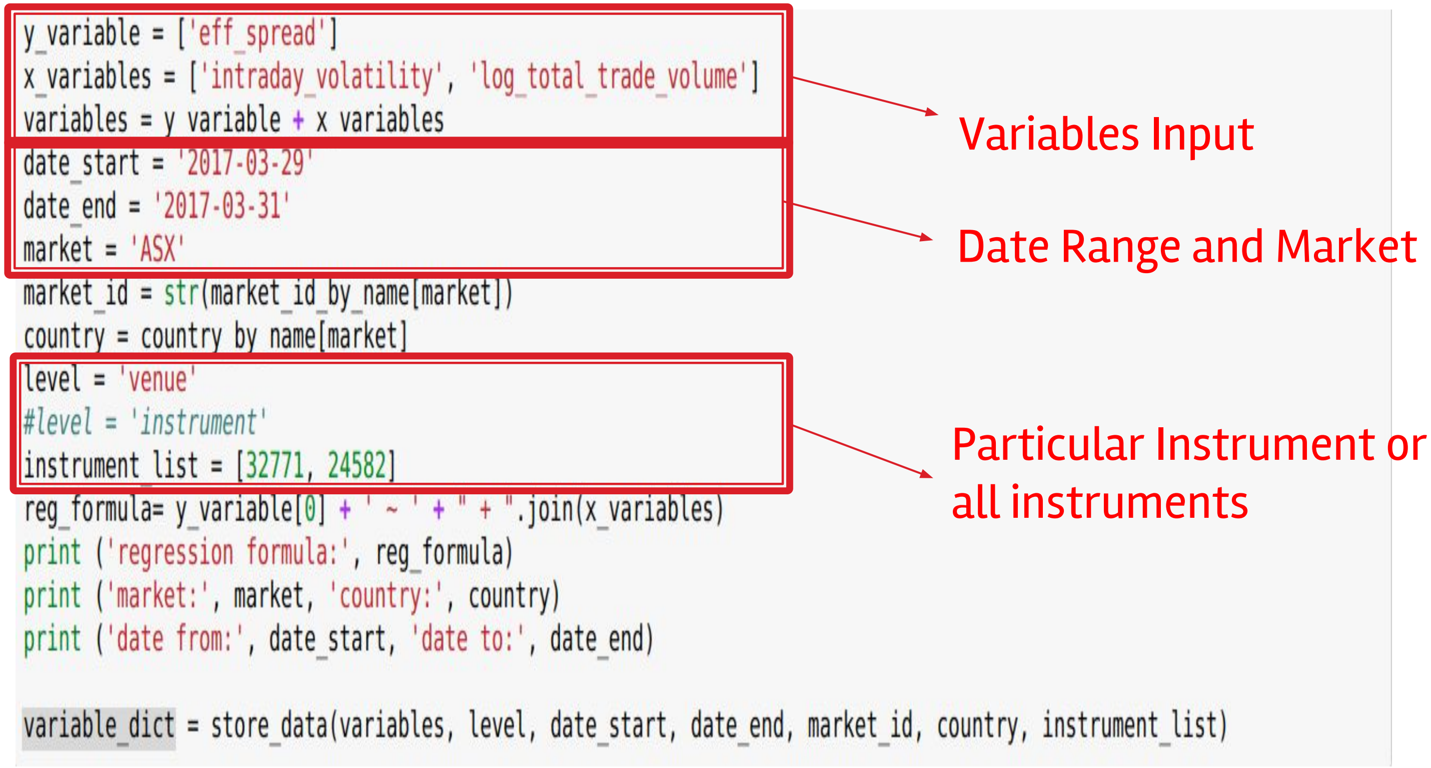
Python Metric Name

Csv Metric Output

Blueshift Input

2. Obtain Metric Result (Cont..)

2) Obtain existing metric result in database



```
y_variable = ['eff_spread']
x_variables = ['intraday_volatility', 'log_total_trade_volume']
variables = y_variable + x_variables
date_start = '2017-03-29'
date_end = '2017-03-31'
market = 'ASX'
market_id = str(market_id_by_name[market])
country = country_by_name[market]
level = 'venue'
#level = 'instrument'
instrument_list = [32771, 24582]
reg_formula= y_variable[0] + ' ~ ' + " + ".join(x_variables)
print ('regression formula:', reg_formula)
print ('market:', market, 'country:', country)
print ('date from:', date_start, 'date to:', date_end)

variable_dict = store_data(variables, level, date_start, date_end, market_id, country, instrument_list)
```

Variables Input

Date Range and Market

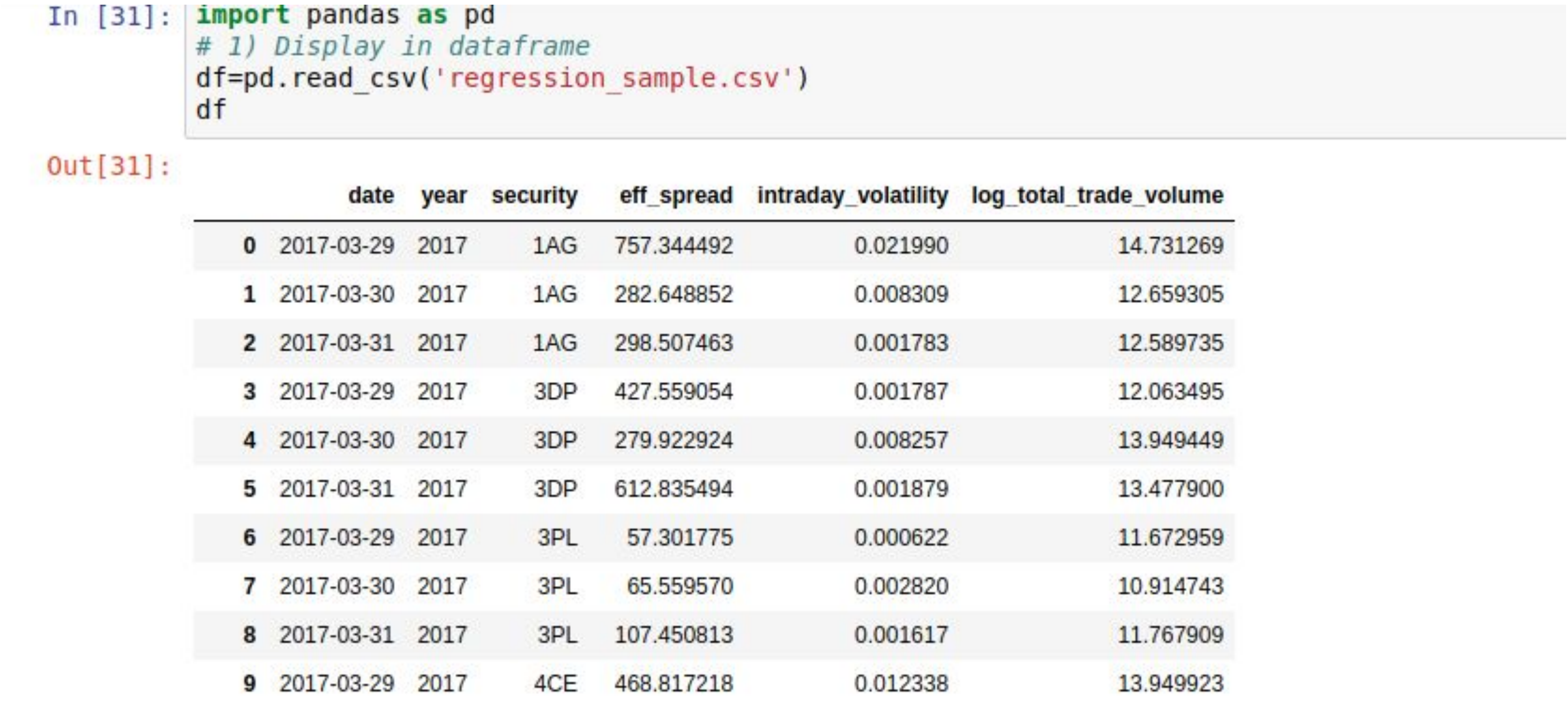
Particular Instrument or all instruments

3. Customize Your Report

1. Display the result in dataframe
   https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html
2. Show descriptive statistics
   http://hamelg.blogspot.com.au/2015/11/python-for-data-analysis-part-21.html
3. Show regression result
   https://www.statsmodels.org/stable/index.html
4. Construct charts
   https://www.highcharts.com/

4. Display Your Research Results

- 1) DataFrame



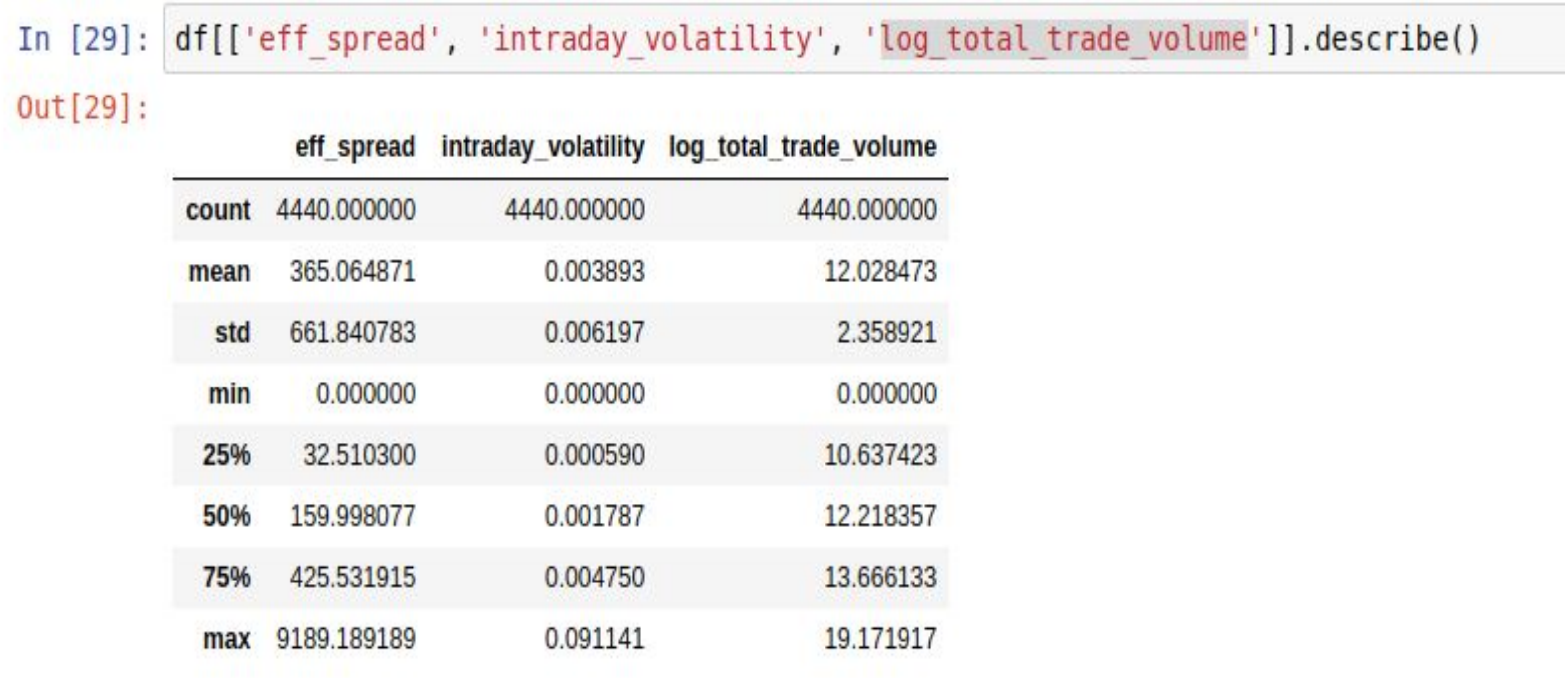
In [31]:
```
import pandas as pd
# 1) Display in dataframe
df=pd.read_csv('regression_sample.csv')
df
```

Out[31]:

	date	year	security	eff_spread	intraday_volatility	log_total_trade_volume
0	2017-03-29	2017	1AG	757.344492	0.021990	14.731269
1	2017-03-30	2017	1AG	282.648852	0.008309	12.659305
2	2017-03-31	2017	1AG	298.507463	0.001783	12.589735
3	2017-03-29	2017	3DP	427.559054	0.001787	12.063495
4	2017-03-30	2017	3DP	279.922924	0.008257	13.949449
5	2017-03-31	2017	3DP	612.835494	0.001879	13.477900
6	2017-03-29	2017	3PL	57.301775	0.000622	11.672959
7	2017-03-30	2017	3PL	65.559570	0.002820	10.914743
8	2017-03-31	2017	3PL	107.450813	0.001617	11.767909
9	2017-03-29	2017	4CE	468.817218	0.012338	13.949923

4. Display Your Research Results (Cont..)

- 2) Descriptive Statistics



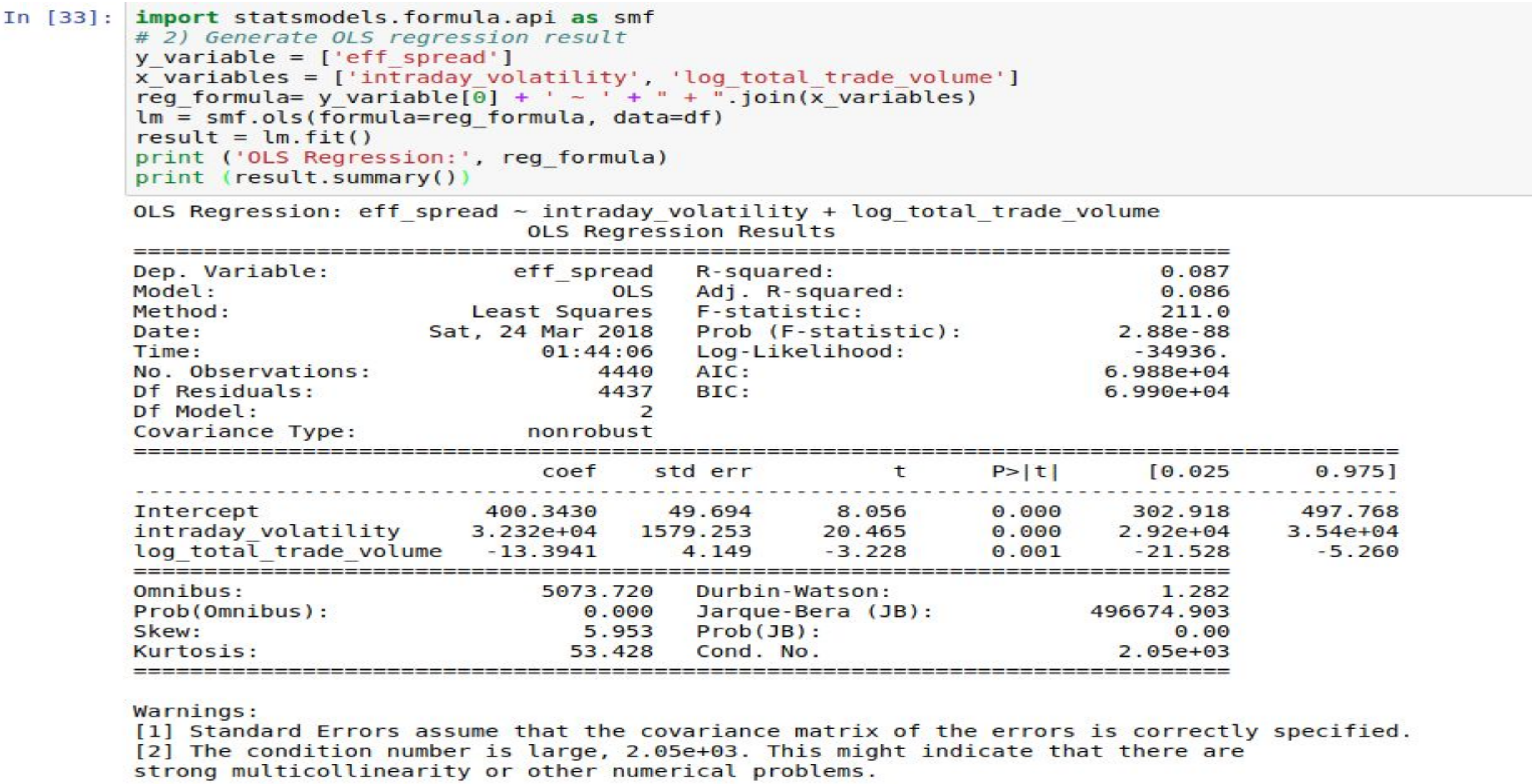
```
In [29]: df[['eff_spread', 'intraday_volatility', 'log_total_trade_volume']].describe()
```

Out[29]:

	eff_spread	intraday_volatility	log_total_trade_volume
count	4440.000000	4440.000000	4440.000000
mean	365.064871	0.003893	12.028473
std	661.840783	0.006197	2.358921
min	0.000000	0.000000	0.000000
25%	32.510300	0.000590	10.637423
50%	159.998077	0.001787	12.218357
75%	425.531915	0.004750	13.666133
max	9189.189189	0.091141	19.171917

4. Display Your Research Results (Cont..)

- 3) Regression



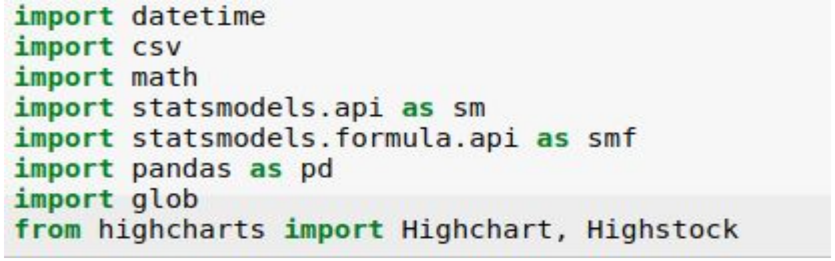
```
In [33]: import statsmodels.formula.api as smf
         # 2) Generate OLS regression result
         y_variable = ['eff_spread']
         x_variables = ['intraday_volatility', 'log_total_trade_volume']
         reg_formula= y_variable[0] + ' ~ ' + " + ".join(x_variables)
         lm = smf.ols(formula=reg_formula, data=df)
         result = lm.fit()
         print ('OLS Regression:', reg_formula)
         print (result.summary())
```

```
OLS Regression: eff_spread ~ intraday_volatility + log_total_trade_volume
                            OLS Regression Results
==============================================================================
Dep. Variable:             eff_spread   R-squared:                       0.087
Model:                            OLS   Adj. R-squared:                  0.086
Method:                 Least Squares   F-statistic:                     211.0
Date:                Sat, 24 Mar 2018   Prob (F-statistic):           2.88e-88
Time:                        01:44:06   Log-Likelihood:                -34936.
No. Observations:                4440   AIC:                         6.988e+04
Df Residuals:                    4437   BIC:                         6.990e+04
Df Model:                           2
Covariance Type:            nonrobust
==========================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------
Intercept                400.3430     49.694      8.056      0.000     302.918     497.768
intraday_volatility     3.232e+04   1579.253     20.465      0.000    2.92e+04    3.54e+04
log_total_trade_volume   -13.3941      4.149     -3.228      0.001     -21.528      -5.260
==============================================================================
Omnibus:                     5073.720   Durbin-Watson:                   1.282
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           496674.903
Skew:                           5.953   Prob(JB):                         0.00
Kurtosis:                      53.428   Cond. No.                     2.05e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.05e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

4. Display Your Research Results (Cont..)

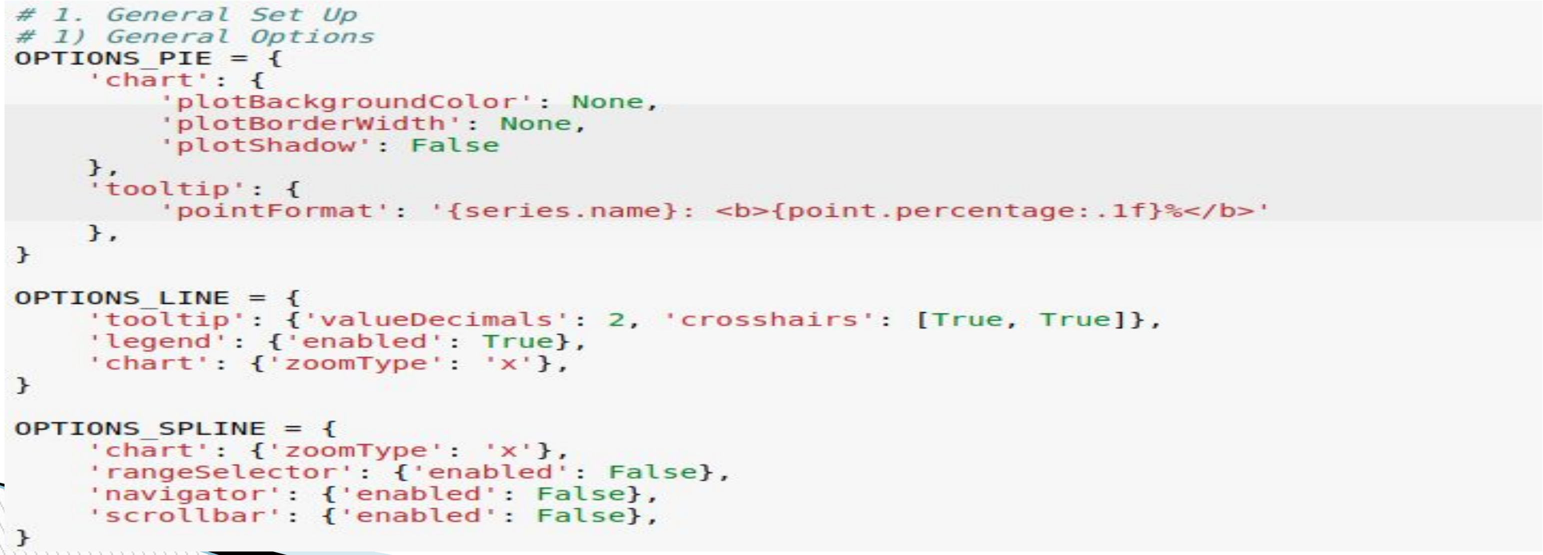
- 4) Chart

- Required Packages



```python
import datetime
import csv
import math
import statsmodels.api as sm
import statsmodels.formula.api as smf
import pandas as pd
import glob
from highcharts import Highchart, Highstock
```

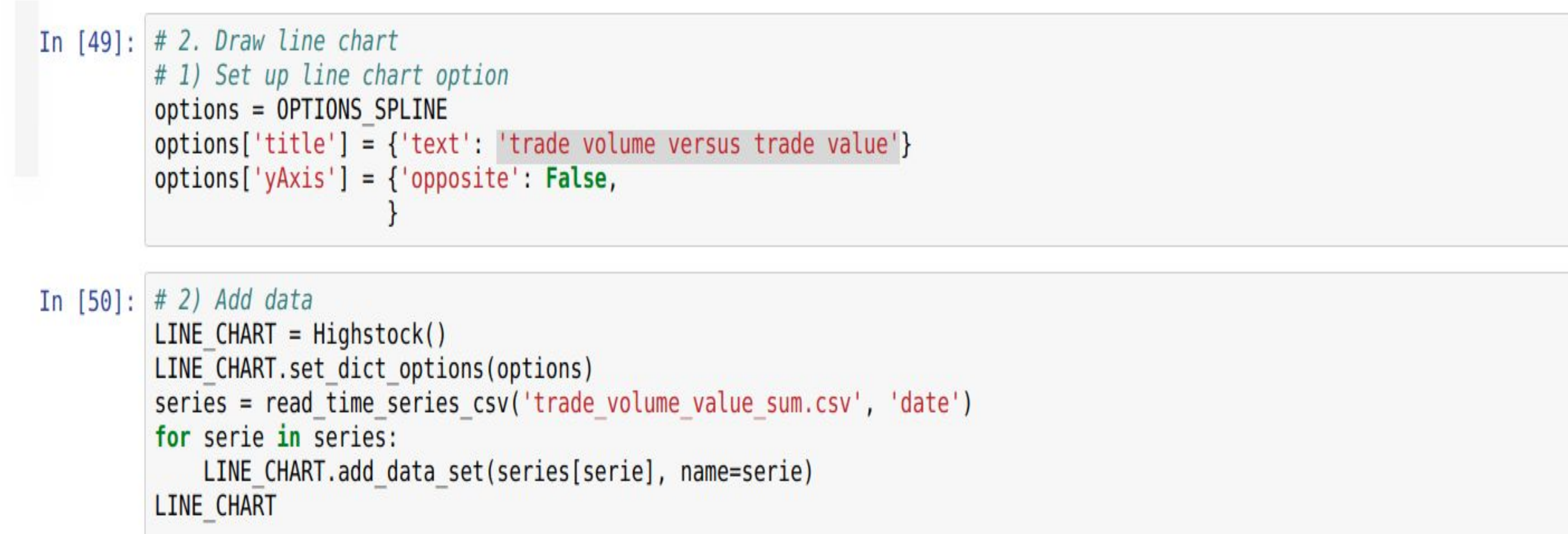
- General Option



```python
# 1. General Set Up
# 1) General Options
OPTIONS_PIE = {
    'chart': {
        'plotBackgroundColor': None,
        'plotBorderWidth': None,
        'plotShadow': False
    },
    'tooltip': {
        'pointFormat': '{series.name}: <b>{point.percentage:.1f}%</b>'
    },
}

OPTIONS_LINE = {
    'tooltip': {'valueDecimals': 2, 'crosshairs': [True, True]},
    'legend': {'enabled': True},
    'chart': {'zoomType': 'x'},
}

OPTIONS_SPLINE = {
    'chart': {'zoomType': 'x'},
    'rangeSelector': {'enabled': False},
    'navigator': {'enabled': False},
    'scrollbar': {'enabled': False},
}
```

4. Display Your Research Results (Cont..)

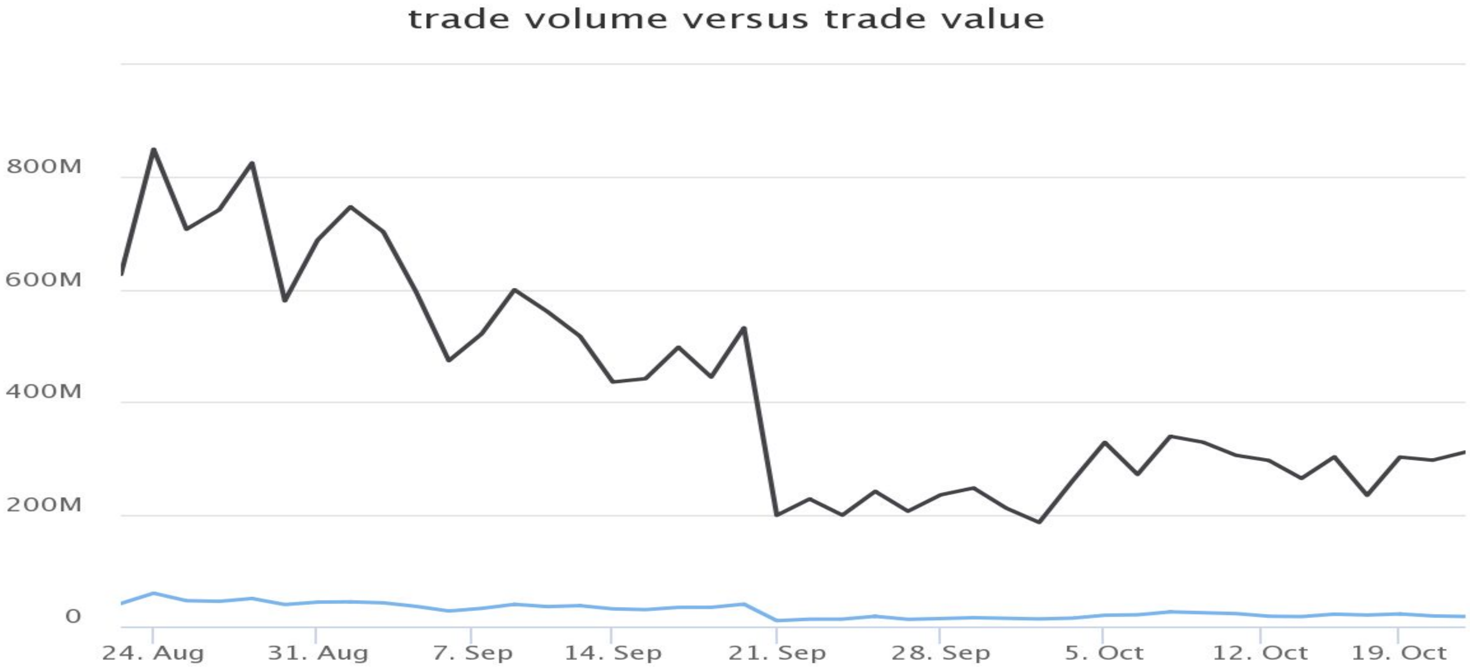
- a) Line Chart



```
In [49]: # 2. Draw line chart
         # 1) Set up line chart option
         options = OPTIONS_SPLINE
         options['title'] = {'text': 'trade volume versus trade value'}
         options['yAxis'] = {'opposite': False,
                            }

In [50]: # 2) Add data
         LINE_CHART = Highstock()
         LINE_CHART.set_dict_options(options)
         series = read_time_series_csv('trade_volume_value_sum.csv', 'date')
         for serie in series:
             LINE_CHART.add_data_set(series[serie], name=serie)
         LINE_CHART
```

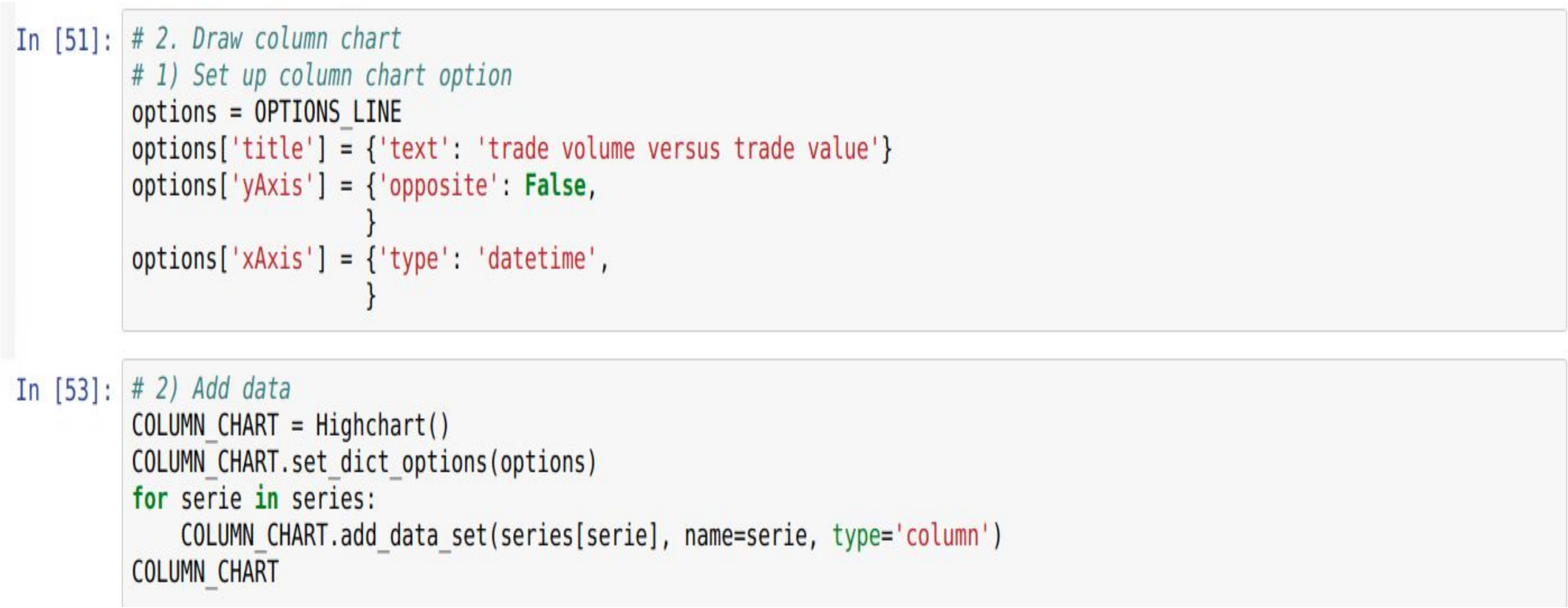
4. Display Your Research Results (Cont..)

- a) Line Chart



4. Display Your Research Results (Cont..)

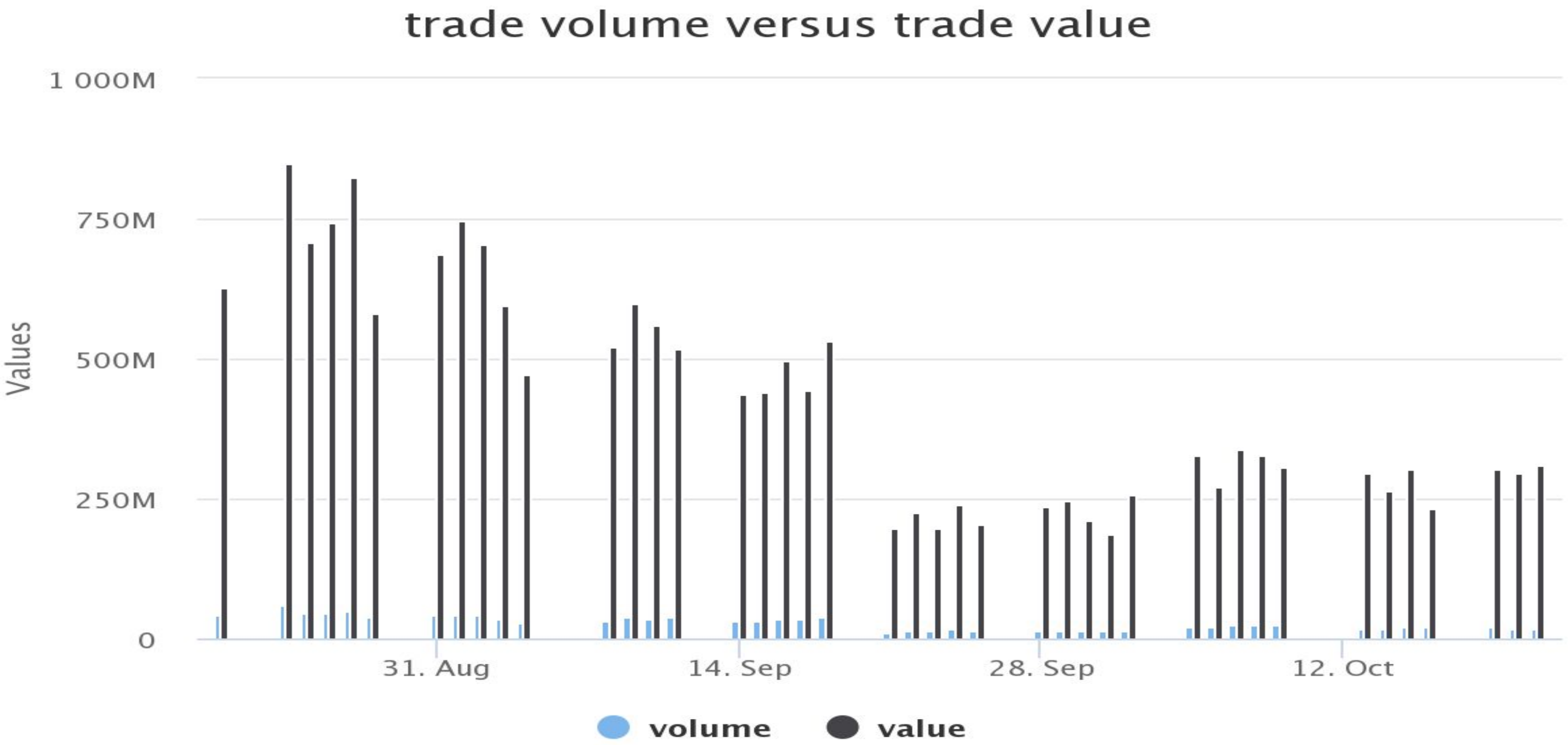
- b) Column Chart



```
In [51]: # 2. Draw column chart
         # 1) Set up column chart option
         options = OPTIONS_LINE
         options['title'] = {'text': 'trade volume versus trade value'}
         options['yAxis'] = {'opposite': False,
                            }
         options['xAxis'] = {'type': 'datetime',
                            }

In [53]: # 2) Add data
         COLUMN_CHART = Highchart()
         COLUMN_CHART.set_dict_options(options)
         for serie in series:
             COLUMN_CHART.add_data_set(series[serie], name=serie, type='column')
         COLUMN_CHART
```

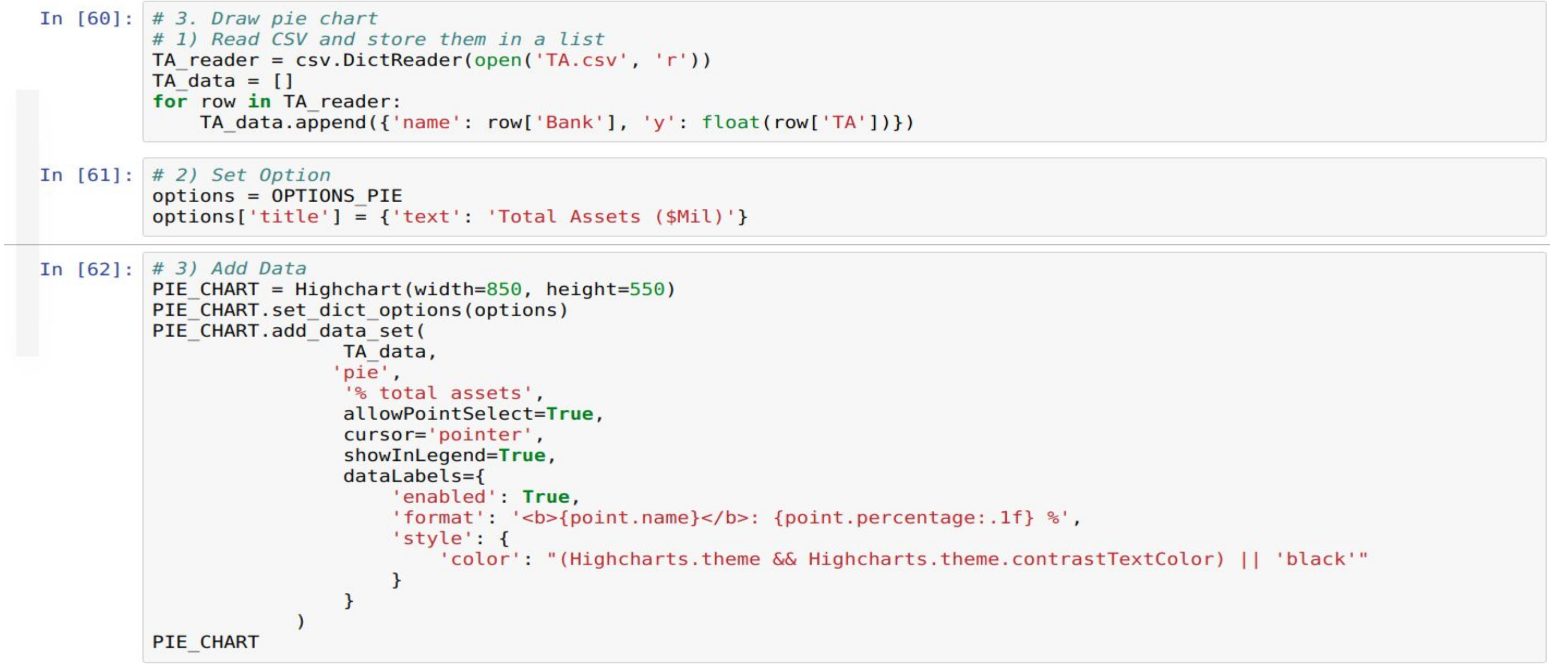
4. Display Your Research Results (Cont..)

- b) Column Chart



4. Display Your Research Results (Cont..)

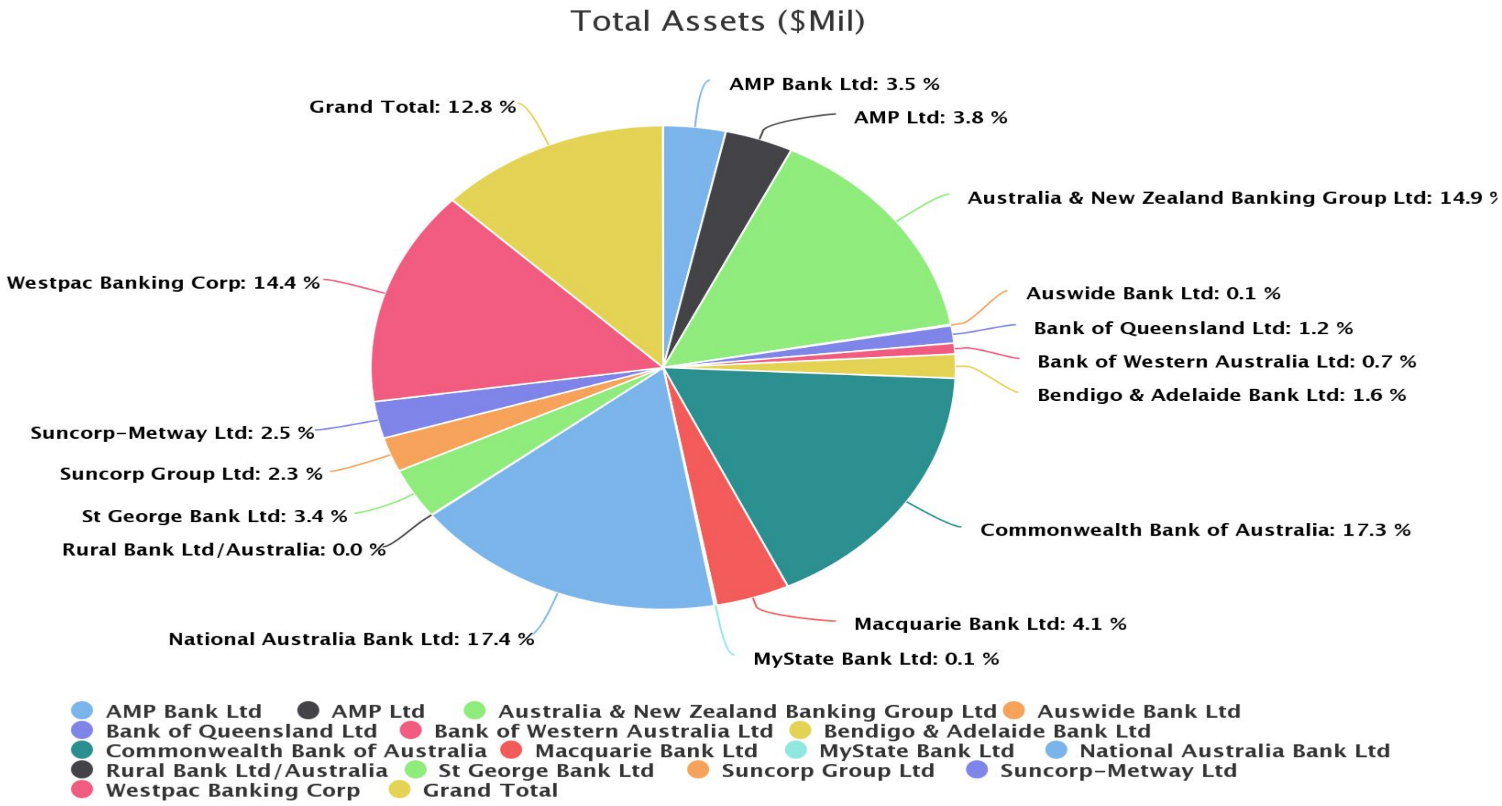
- c) Pie Chart



```
In [60]: # 3. Draw pie chart
         # 1) Read CSV and store them in a list
         TA_reader = csv.DictReader(open('TA.csv', 'r'))
         TA_data = []
         for row in TA_reader:
             TA_data.append({'name': row['Bank'], 'y': float(row['TA'])})

In [61]: # 2) Set Option
         options = OPTIONS_PIE
         options['title'] = {'text': 'Total Assets ($Mil)'}

In [62]: # 3) Add Data
         PIE_CHART = Highchart(width=850, height=550)
         PIE_CHART.set_dict_options(options)
         PIE_CHART.add_data_set(
                         TA_data,
                        'pie',
                         '% total assets',
                         allowPointSelect=True,
                         cursor='pointer',
                         showInLegend=True,
                         dataLabels={
                             'enabled': True,
                             'format': '<b>{point.name}</b>: {point.percentage:.1f} %',
                             'style': {
                                 'color': "(Highcharts.theme && Highcharts.theme.contrastTextColor) || 'black'"
                             }
                         }
                     )
         PIE_CHART
```

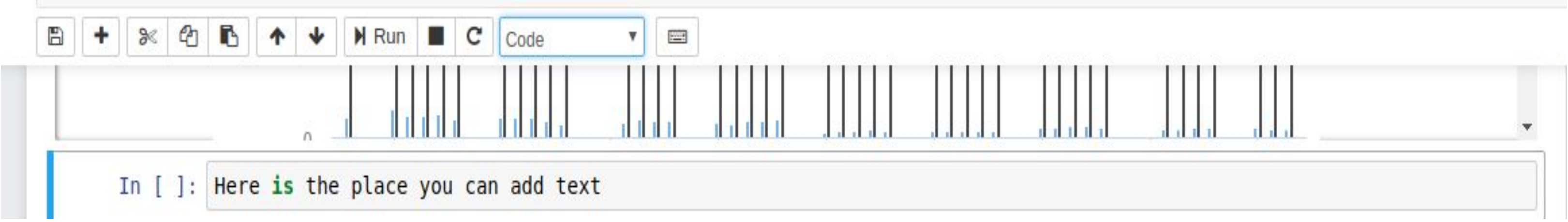
4. Display Your Research Results (Cont..)

- c) Pie Chart



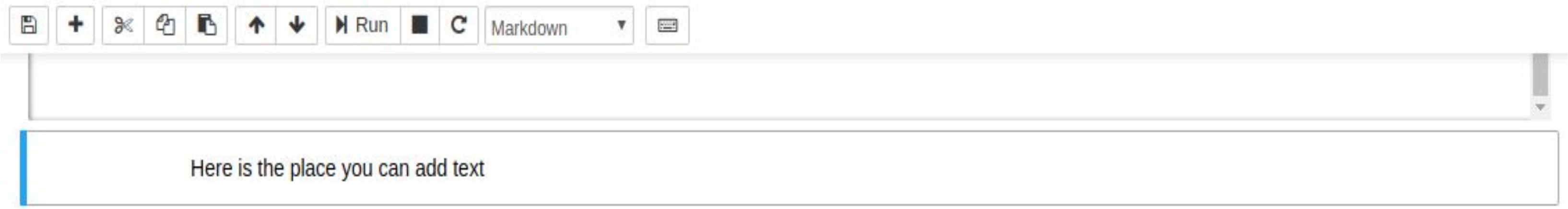
4. Display Your Research Results (Cont..)

- 5) Add text
- Before adding markdown text



```
In [ ]: Here is the place you can add text
```

- After adding markdown text



Here is the place you can add text

Questions?