Gemini Documentation

Release 1.0.0

Anthony Federico

Jun 25, 2021

Contents

CHAPTER 1

Setup

1.1 Repository Clone

Currently the best way to using Gemini is to clone the Github repository.

```
git clone https://github.com/anfederico/Gemini
```

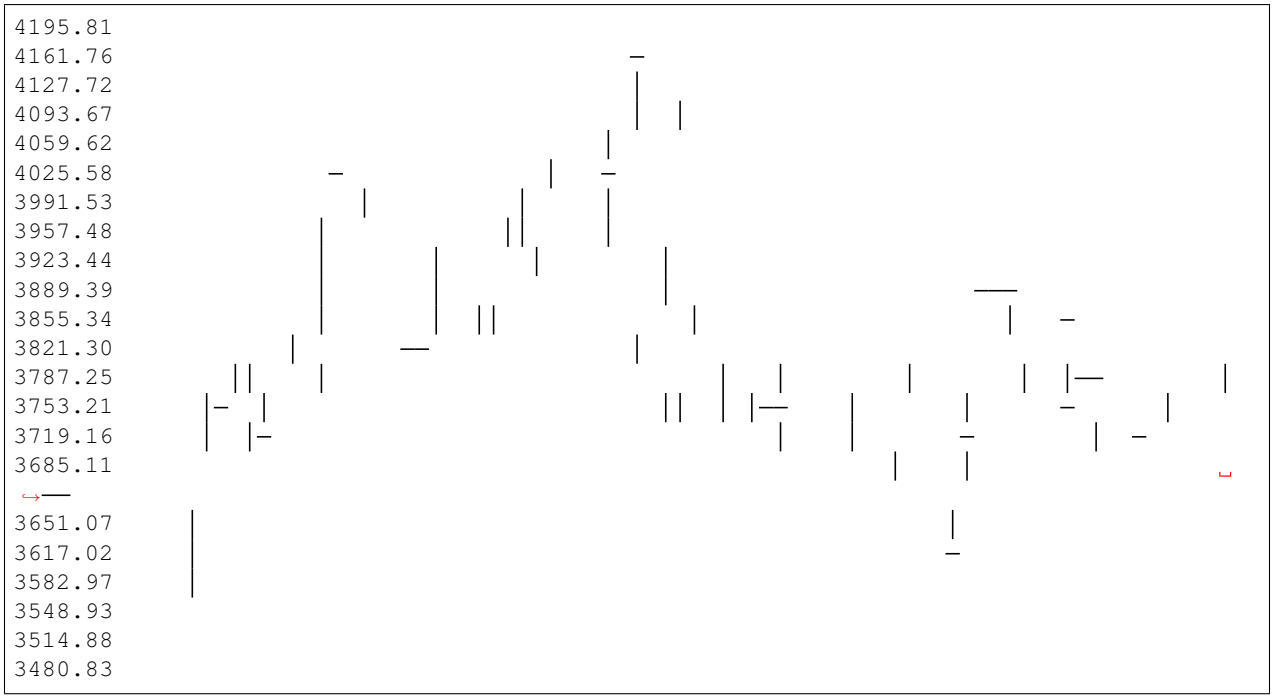
1.2 Quick Example

For a quick example that shows how one would run a backtest using a clone of Gemini, run the following.

```
cd Gemini/examples

python mean_reversion.py
```

CHAPTER 2

Historical Data



2.1 Importing Data

If you have your own data that has/hasn't been processed, you should conform to the following structure. Basically, load your data into a Pandas dataframe object and be sure to convert the dates to datetime format and include the following lowercase column titles.

```
                  date         high          low         open        close
0  2017-07-08 11:00:00  2480.186778  2468.319314  2477.279567  2471.314030
1  2017-07-08 11:30:00  2471.314030  2455.014057  2471.202796  2458.073602
2  2017-07-08 12:00:00  2480.000000  2456.000000  2458.073602  2480.000000
3  2017-07-08 12:30:00  2489.004639  2476.334333  2479.402768  2481.481258
4  2017-07-08 13:00:00  2499.000000  2476.621873  2481.458643  2491.990000
5  2017-07-08 13:30:00  2503.503479  2490.314610  2492.440289  2496.005562
6  2017-07-08 14:00:00  2525.000000  2491.062741  2494.449524  2520.775500
7  2017-07-08 14:30:00  2521.500036  2510.000000  2520.775500  2518.450645
8  2017-07-08 15:00:00  2519.817394  2506.054360  2518.451000  2514.484009
```

2.2 Downloading Data

If you don't have your own data, we've included useful functions for grabbing low and high timeframe historical data from crypto exchanges. These helper functions will automatically resample your datasets to any desired timeframe and return a Gemini-compatible dataframe.

```
import data

# Higher timeframes (>= daily)
df = data.get_htf_candles("BTC_USD", "Bitfinex", "3-DAY", "2019-01-12 00:00:00",
↪"2019-02-01 00:00:00")

# Lower timeframes (< daily)
df = data.get_ltf_candles("USDC_BTC", "30-MIN", "2019-01-12 00:00:00", "2019-02-01␣
↪00:00:00")
```

CHAPTER 3

Using Gemini

3.1 Initialize Backtest

Once your import or download data, you must initialize that backtesting engine with your data. This will simply create a backtesting class with your data pre-loaded.

```
import data

# Higher timeframes (>= daily)
df = data.get_htf_candles("BTC_USD", "Bitfinex", "3-DAY", "2019-01-12 00:00:00",
↪"2019-02-01 00:00:00")

# Lower timeframes (< daily)
df = data.get_ltf_candles("USDC_BTC", "30-MIN", "2019-01-12 00:00:00", "2019-02-01␣
↪00:00:00")

# Loading data into the backtester
import engine

backtest = engine.backtest(df)
```

3.2 Define Stategy

In addition to loading the data, you must define the strategy you want to test. To do this, we'll create a logic function that can be passed to the backtester when you start. The backtester will proceed step-wise through the dataset, copying the current/past datapoints into a variable called "Lookback" to prevent lookahead bias. If the data hasn't already been processed, you may process it within the logic function (this makes the simulation more accurate but significantly increases runtime). You can then use the helper class called "Period" to conveniently reference current and past datapoints. With those, you may execute long, sell, short, and cover positions directly on the "Account" class based on your strategy.

3.2.1 Basic

A basic mock strategy example

```
def logic(account, lookback):
    try:
        # Process dataframe to collect signals
        lookback = helpers.get_signals(lookback)

        # Load into period class to simplify indexing
        lookback = helpers.period(lookback)

        today = lookback.loc(0) # Current candle
        yesterday = lookback.loc(-1) # Previous candle

        if today['signal'] == "down":
            if yesterday['signal'] == "down":
                exit_price = today['close']
                for position in acount.positions:
                    if position.type == 'long':
                        account.close_position(position, 0.5, exit_price)

        if today['signal'] == "up":
            if yesterday['signal'] == "up":
                risk          = 0.03
                entry_price   = today['close']
                entry_capital = account.buying_power*risk
                if entry_capital >= 0:
                    account.enter_position('long', entry_capital, entry_price)

    except ValueError:
        pass # Handles lookback errors in beginning of dataset
```

3.2.2 Advanced

A real mean reversion strategy example

```
import pandas as pd
import numpy as np
from talib.abstract import *

def bands(df, timeperiod=26, nbdevup=2.6, nbdevdn=2.6, matype=0):
    cols = ['high', 'low', 'open', 'close', 'volume']
    HLOCV = {key: df[key].values for key in df if key in cols}
    u, m, l = BBANDS(HLOCV, timeperiod=timeperiod, nbdevup=nbdevup, nbdevdn=nbdevdn,␣
↪matype=matype)
    df['upper']  = u
    df['middle'] = m
    df['lower']  = l
    return df

def touches(df):
    df['touch_upper'] = df.high >= df.upper
    df['touch_lower'] = df.low  <= df.lower
    df['crossing_dn'] = (df.close < df.middle) & (df.open > df.middle)
    df['crossing_up'] = (df.close > df.middle) & (df.open < df.middle)
```

(continues on next page)

(continued from previous page)

```
    return df

def logic(account, lookback):
    try:
        lookback = bands(lookback)
        lookback = touches(lookback)
        lookback = helpers.period(lookback)
        today = lookback.loc(0)

        # Selling
        if today.touch_upper:
            exit_price = today.upper
            for position in account.positions:
                if position.type_ == 'long':
                    account.close_position(position, 1, exit_price)

        if today.crossing_up:
            exit_price = today.close
            for position in account.positions:
                if position.type_ == 'long':
                    account.close_position(position, 1, exit_price)

        # Buying
        if today.touch_lower | today.crossing_dn:
            risk          = 1
            entry_price   = today.lower
            entry_capital = account.buying_power*risk
            if entry_capital > 0:
                account.enter_position('long', entry_capital, entry_price)

        if today.crossing_dn:
            risk          = 1
            entry_price   = today.close
            entry_capital = account.buying_power*risk
            if entry_capital > 0:
                account.enter_position('long', entry_capital, entry_price)

    except Exception as e:
        print(e)
    pass # Handles lookback errors in beginning of dataset
```

3.3 Run Backtest

Running the backtest is as simple as running the following one-liner.

```
backtest.start(1000, logic)
```

3.4 Performance Statistics

After the backtest, you can analyze your strategy by printing the results to console. As of now, these include simple statistics of your run but we plan to implement more complicated metrics for a stronger understanding of performance.

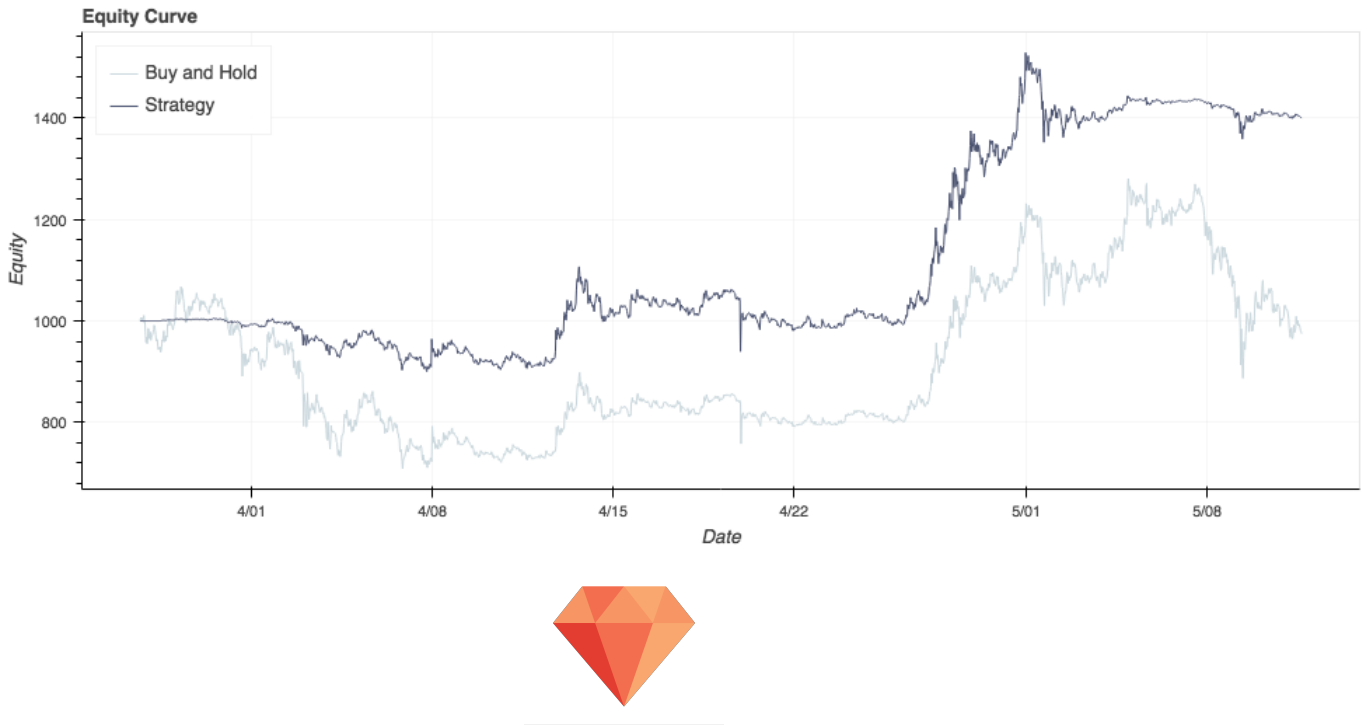
```
backtest.results()
```

```
-------------- Results ----------------

Buy and Hold : 534.15%
Net Profit   : 5341.47
Strategy     : 3811.75%
Net Profit   : 38117.46
Longs        : 30
sells        : 31
shorts       : 0
covers       : 0
--------------------
Total Trades : 61

---------------------------------------
```

3.5 Equity Curve

You can visualize the performance of your strategy by comparing the equity curve with a buy and hold baseline. The equity curve simply tracks your account value throughout the backtest and will optionally show where your algorithm made its trades including longs, sells, shorts, and covers.

```
backtest.chart()
```



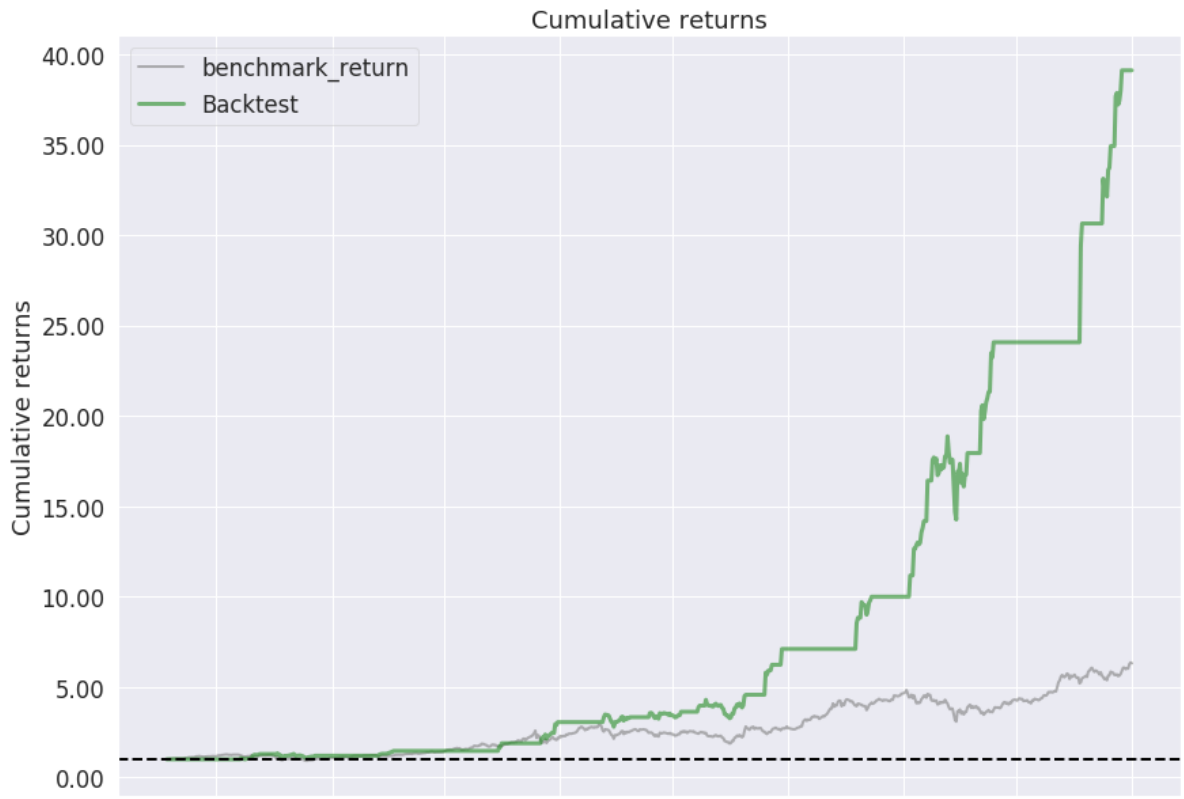
CHAPTER 4

Exporting Results

4.1 Pyfolio Example

4.2 Raw Statistics

4.3 Tearsheet Charts



Cumulative returns volatility matched to benchmark

CHAPTER 5

API Reference

5.1 gemini

5.1.1 gemini package

5.1.1.1 Submodules

gemini.data module

gemini.engine module

gemini.exchange module

gemini.helpers module

gemini.ptable module

gemini.settings module

5.1.1.2 Module contents