```
from tda import auth, client
import json
import pandas
import datetime
import sys

pandas.set_option('display.max_columns', None)  # don't truncate
displays of wide Pandas data frames

# "log in" using OAuth2.0 protocol to TD Ameritrade to obtain "bearer
token"
# bearer token is stored in token_path for use in future runs of the
program
token_path = 'token.pickle'
api_key = 'YPLV570YA3I5WZA5BB6DPGJDI49HZ19J@AMER.OAUTHAP'
redirect_uri = 'https://localhost'
try:
    c = auth.client_from_token_file(token_path, api_key)
except FileNotFoundError:
    from selenium import webdriver
    with webdriver.Firefox() as driver:
        c = auth.client_from_login_flow(driver, api_key, redirect_uri,
token_path)

# Sample: Retrieve AAPL stock price history
r = c.get_price_history('AAPL',

period_type=client.Client.PriceHistory.PeriodType.YEAR,

period=client.Client.PriceHistory.Period.ONE_YEAR,                # 1
year

frequency_type=client.Client.PriceHistory.FrequencyType.MONTHLY,

frequency=client.Client.PriceHistory.Frequency.MONTHLY)           #
monthly bars
assert r.status_code == 200, r.raise_for_status()
# print(json.dumps(r.json(), indent=3))

# Sample: Retrieve SPY option chain for a few specified month range, 2
strikes around the money
s = c.get_option_chain(symbol='SPY',
                       strike_count=(2),
                       from_date=datetime.date(2021, 9, 17),
                       to_date=datetime.date(2021, 10, 15))
assert s.status_code == 200, s.raise_for_status()

# Sample: Retrieve SPY option chain for a single specified month
August, 2 strikes around the money
s = c.get_option_chain(symbol='SPY',
```

```
                        strike_count=(2),

exp_month=(client.Client.Options.ExpirationMonth.AUGUST))
assert s.status_code == 200, s.raise_for_status()

# Sample: Retrieve SPY option chain for a few months (all strike
prices, all/weekly expiration cycles)
s = c.get_option_chain(symbol='SPY',
                        from_date=datetime.date(2021, 9, 17),
                        to_date=datetime.date(2021, 10, 15))
assert s.status_code == 200, s.raise_for_status()

response_json = s.json()
response_json_underlying_symbol = response_json['symbol']
response_json_underlying_price = response_json['underlyingPrice']
response_json_puts = response_json['putExpDateMap']
response_json_calls = response_json['callExpDateMap']

# gather_details
# inputs: options listing (puts or calls), chain symbol
# output: dictionary of option chain and list of column names
def gather_details(chain, chain_symbol):
    accumulated = []
    one_by_one_df = pandas.DataFrame()
    for k1, v1 in sorted(chain.items()):
        for k2, v2 in sorted(v1.items()):
            col_names = list(v2[0].keys())
            one_by_one_df = pandas.json_normalize(chain,
record_path=[[k1, k2]])  # parse efficently into Pandas frame of one
row
            df_values = one_by_one_df.T.to_dict('list')[0]  # convert
back into dictionary
            df_values[:0] = [chain_symbol, k1[0:k1.rindex(':')],
int(k1[k1.rindex(':')+1:]), float(k2)]  # add latest into collection
            col_names = ['underlying_symbol', 'expiration',
'days_to_exp', 'strike_price'] + col_names
            accumulated.append(df_values)
    return {'option_details': accumulated, 'column_names': col_names}


gather_puts = gather_details(response_json_puts,
response_json_underlying_symbol)
gather_calls = gather_details(response_json_calls,
response_json_underlying_symbol)
all_options = gather_puts['option_details'] +
gather_calls['option_details']
column_names = gather_puts['column_names']
df_options = pandas.DataFrame(data=all_options,
columns=(column_names))  # create Pandas data frame from list
```

```
# Sample: For each expiration and side calls vs. pus, output
cumulative open interest
expiry_dates = sorted(set(df_options.expiration))
for exp_date in expiry_dates:
    for side in ['CALL', 'PUT']:
        sub_data = df_options[(df_options['expiration'] == exp_date) &
(df_options['putCall'] == side)]
        sub_data_OI_sum = sub_data['openInterest'].sum()
        print('Cumulative open interest for',
response_json_underlying_symbol,
              'for expiration', exp_date,
              'side', side,
              'is', sub_data_OI_sum)

# Sample: combined weighted average open interest based predictor
(CWOP)
#
# Paper reference:
#   Rafiqul Bhuyan and Mo Chaudhury,
#   "Trading on the information content of open interest: Evidence
from the US equity options market", 2005
#
# Equation 3 on page 20: (applied for a given underlying and
expiration)
# Combined Weighted average Open interest based Predictor (CWOP):
# Expected equilibrium stock price E(S sub t) =
# [Sum over all strike prices, for calls (open interest at specific
strike price * specific strike price) +
#  Sum over all strike prices, for puts  (open interest at specific
strike price * specific strike price)] /
# [Sum over all strike prices, for calls (open interest) + Sum over
all strike prices, for puts (open interest)]
#
# Trading Assumption:
#    CWOP is "correct" and
#    current price will move to CWOP price at expiration
# Trading Method: Pair trade CWOP vs. current price
#    trade bullish if current stock is less than    CWOP
#    trade bearish if current stock is greater than CWOP

# iterate over all expiration periods
bc_expiry_dates = sorted(set(df_options.expiration))
for exp_date in bc_expiry_dates:
    # compute denominator
    # accumulate open interest for calls
    sub_data = df_options[(df_options['expiration'] == exp_date) &
(df_options['putCall'] == 'CALL')]
    sub_data_OI_sum_calls = sub_data['openInterest'].sum()
    # accumulate open interest for puts
    sub_data = df_options[(df_options['expiration'] == exp_date) &
```

```
(df_options['putCall'] == 'PUT')]
    sub_data_OI_sum_puts = sub_data['openInterest'].sum()
    # accumulate total open interest
    sub_data_OI_sum = sub_data_OI_sum_calls + sub_data_OI_sum_puts

    # compute numerator
    bc_expiration_subset = df_options[(df_options['expiration'] ==
exp_date)]
    # print('bc_expiration_subset for', exp_date, 'is',
bc_expiration_subset)
    bc_strike_prices =
sorted(set(bc_expiration_subset['strike_price']))
    # print('bc_expiration_prices', bc_strike_prices, 'for', exp_date)

    bc_sum_oi_price = {}
    bc_sum_oi_price['CALL'] = 0
    bc_sum_oi_price['PUT'] = 0
    for strike_price in bc_strike_prices:

        # strike prices (either calls or puts) in given expiration
        bc_strikes_subset =
bc_expiration_subset[(bc_expiration_subset['strike_price'] ==
strike_price)]

        # accumulate fractional open interest * strike price
        def oi_accum_for_strike_side(side):
            bc_specific_subset =
bc_strikes_subset[(bc_strikes_subset['putCall'] == side)]
            bc_oi = int(bc_specific_subset['openInterest'])
            bc_sum_oi_price[side] += bc_oi * strike_price

        oi_accum_for_strike_side('CALL')
        oi_accum_for_strike_side('PUT')

    # compute CWOP
    price_estimator = ((bc_sum_oi_price['CALL'] +
bc_sum_oi_price['PUT']) / sub_data_OI_sum)
    print('Bhuyan Chaudhury estimate for',
response_json_underlying_symbol,
          'on expiration', exp_date,
          'with underying price', response_json_underlying_price,
          'is', price_estimator)


# sample: all possible iron condors
#
# within each expiration cycle, configure all iron condors with
# short options a given number of dollars away from the money,
# and a given dollar distance between the short option to the long
option,
```

```
# and sort them by credit/risk ratio
#
# a key issue to deal with is that there may NOT be an option with a
given target strike price,
# so we have to compute the NEAREST away from the money strike price
# and this has to performed in mirror fashion for put vs. call sides
#
apic_spot_to_short_target = [1, 5, 10]  # list of dollar distances
between current price to short option
apic_short_to_wing_target = [1, 5, 10]  # list of dollar distances
between short option strike price and long option
apic_expiry_dates = sorted(set(df_options['expiration']))
rows_list = []
for exp_date in apic_expiry_dates:

    apic_expiration_subset = df_options[(df_options['expiration'] ==
exp_date)]
    # print('apic_expiration_subset for', exp_date, 'is',
apic_expiration_subset)

    apic_strike_prices =
sorted(set(apic_expiration_subset['strike_price']))
    # print('apic_expiration_prices', apic_strike_prices, 'for',
exp_date)

    # compute successive short option strike prices away from the
money
    for spot_to_short_target in apic_spot_to_short_target:

        # target prices
        apic_call_distance_underying_to_spot_to_short_target =
(response_json_underlying_price + spot_to_short_target)
        apic_call_distance_spot_to_short_target = [(apic_sp, apic_sp -
apic_call_distance_underying_to_spot_to_short_target) for apic_sp in
apic_strike_prices]

        apic_put_distance_underying_to_spot_to_short_target =
(response_json_underlying_price - spot_to_short_target)
        apic_put_distance_spot_to_short_target = [(apic_sp, apic_sp -
apic_put_distance_underying_to_spot_to_short_target) for apic_sp in
apic_strike_prices]

        # beginning with target short option, pick strike prices that
are FURTHER away from the money
        apic_call_distance_spot_to_short_target_goal = [v for v in
apic_call_distance_spot_to_short_target if (v[1] >= 0)]
        apic_put_distance_spot_to_short_target_goal = [v for v in
apic_put_distance_spot_to_short_target if (v[1] <= 0)]

        # pick the FIRST strike price beyond the target further away
```

```
from the money
        apic_call_closest_short_strike =
min(apic_call_distance_spot_to_short_target_goal, key=lambda t: t[1])
[0]
        apic_put_closest_short_strike =
max(apic_put_distance_spot_to_short_target_goal, key=lambda t: t[1])
[0]

        apic_call_expiration_subset_side =
apic_expiration_subset[(apic_expiration_subset['strike_price'] ==
apic_call_closest_short_strike) &

(apic_expiration_subset['putCall'] == 'CALL')]
        apic_put_expiration_subset_side =
apic_expiration_subset[(apic_expiration_subset['strike_price'] ==
apic_put_closest_short_strike) &

(apic_expiration_subset['putCall'] == 'PUT')]

        # when selling be conservative and sell at bid
        apic_call_closest_short_strike_price =
float(apic_call_expiration_subset_side['bid'])
        apic_put_closest_short_strike_price =
float(apic_put_expiration_subset_side['bid'])

        # compute successive long option strike prices from selected
short options
        for short_to_wing_target in apic_short_to_wing_target:

            apic_call_distance_short_to_short_to_wing_target =
(apic_call_closest_short_strike + short_to_wing_target)
            apic_call_distance_short_to_wing_target = [(apic_sp,
apic_sp - apic_call_distance_short_to_short_to_wing_target) for
apic_sp in apic_strike_prices]

            apic_put_distance_short_to_short_to_wing_target =
(apic_put_closest_short_strike - short_to_wing_target)
            apic_put_distance_short_to_wing_target = [(apic_sp,
apic_sp - apic_put_distance_short_to_short_to_wing_target) for apic_sp
in apic_strike_prices]

            # pick strike price that is further from the money
            apic_call_distance_short_to_wing_target_goal = [v for v in
apic_call_distance_short_to_wing_target if (v[1] >= 0)]
            apic_put_distance_short_to_wing_target_goal = [v for v in
apic_put_distance_short_to_wing_target if (v[1] <= 0)]
            apic_call_closest_wing_strike =
min(apic_call_distance_short_to_wing_target_goal, key=lambda t: t[1])
[0]
            apic_put_closest_wing_strike =
```

```
max(apic_put_distance_short_to_wing_target_goal, key=lambda t: t[1])
[0]

            apic_call_expiration_subset_side =
apic_expiration_subset[(apic_expiration_subset['strike_price'] ==
apic_call_closest_wing_strike) &

(apic_expiration_subset['putCall'] == 'CALL')]
            apic_put_expiration_subset_side =
apic_expiration_subset[(apic_expiration_subset['strike_price'] ==
apic_put_closest_wing_strike) &

(apic_expiration_subset['putCall'] == 'PUT')]

            # when buying be conservative and buy at ask
            apic_call_closest_wing_strike_price =
float(apic_call_expiration_subset_side['ask'])
            apic_put_closest_wing_strike_price =
float(apic_put_expiration_subset_side['ask'])

            # compose iron condor from put vertical + call vertical

            # attributes of call vertical
            apic_call_vertical_value =
(apic_call_closest_short_strike_price -
apic_call_closest_wing_strike_price)
            apic_call_width = (apic_call_closest_wing_strike -
apic_call_closest_short_strike)
            apic_call_spread_risk = (apic_call_width -
apic_call_vertical_value)
            apic_call_value_over_risk = (apic_call_vertical_value /
apic_call_spread_risk)

            # attributes of put vertical
            apic_put_vertical_value =
(apic_put_closest_short_strike_price -
apic_put_closest_wing_strike_price)
            apic_put_width = (apic_put_closest_short_strike -
apic_put_closest_wing_strike)
            apic_put_spread_risk = (apic_put_width -
apic_put_vertical_value)
            apic_put_value_over_risk = (apic_put_vertical_value /
apic_put_spread_risk)

            # restrict to iron condors whose component verticals are
credits and where the width on both sides are the same
            if ((apic_call_value_over_risk > 0) and
(apic_put_value_over_risk > 0) and (apic_call_width ==
apic_put_width)):
                apic_ic_value = (apic_call_vertical_value +
```

```
apic_put_vertical_value)
                apic_ic_width = max(apic_call_width, apic_put_width)
# but the same
                apic_ic_spread_risk = (apic_ic_width - apic_ic_value)
                apic_ic_value_over_risk = (apic_ic_value /
apic_ic_spread_risk)

                rows_list.append({'underlying_symbol':
response_json_underlying_symbol,
                                  'response_json_underlying_price':
response_json_underlying_price,
                                  'exp_date': exp_date,
                                  'spot_to_short_target':
spot_to_short_target,
                                  'short_to_wing_target':
short_to_wing_target,
                                  'apic_call_closest_short_strike':
apic_call_closest_short_strike,

'apic_call_closest_short_strike_price':
apic_call_closest_short_strike_price,
                                  'apic_call_closest_wing_strike':
apic_call_closest_wing_strike,

'apic_call_closest_wing_strike_price':
apic_call_closest_wing_strike_price,
                                  'apic_call_width': apic_call_width,
                                  'apic_call_spread_risk':
apic_call_spread_risk,
                                  'apic_call_vertical_value':
apic_call_vertical_value,
                                  'apic_call_value_over_risk':
apic_call_value_over_risk,
                                  'apic_put_closest_short_strike':
apic_put_closest_short_strike,

'apic_put_closest_short_strike_price':
apic_put_closest_short_strike_price,
                                  'apic_put_closest_wing_strike':
apic_put_closest_wing_strike,

'apic_put_closest_wing_strike_price':
apic_put_closest_wing_strike_price,
                                  'apic_put_width': apic_put_width,
                                  'apic_put_spread_risk':
apic_put_spread_risk,
                                  'apic_put_vertical_value':
apic_put_vertical_value,
                                  'apic_put_value_over_risk':
apic_put_value_over_risk,
```

```
                                    'apic_ic_width': apic_ic_width,
                                    'apic_ic_spread_risk':
apic_ic_spread_risk,
                                    'apic_ic_value': apic_ic_value,
                                    'apic_ic_value_over_risk':
apic_ic_value_over_risk})

df1 = pandas.DataFrame(rows_list)
df1.sort_values(by=['apic_ic_value_over_risk'], inplace=True)
```