

*Quant Invests*

*Where the Quant Invests For You!*

Author:

Joe Chamoun

Mobile:

403-463-8072

Website:

quantinvests.com

Email:

Joechamoun23@gmail.com

LinkedIn:

<https://ca.linkedin.com/company/quantinvests>

======================================================== **Details of the investment service app:**

=============================================

**App Name: Quant Invests**

**Tagline: Where the Quant Invests For You!**

**Author: Joe Chamoun**

**Contact Number: 403-463-8072**

**Website: quantinvests.com**

**Email:** [**Joechamoun23@gmail.com**](mailto:Joechamoun23@gmail.com)

**LinkedIn:**

[**https://ca.linkedin.com/company/quantinvests**](https://ca.linkedin.com/company/quantinvests)

**Logo:**



Front End:

| *Quant Invests* |

[Logo] Where the Quant Invests for You

[Investment App Wireframe] |

------------------------------------------------| [Navigation Menu] |

| [User Profile] [Settings] [Logout] |

|----------------------------------------------|

| [Sidebar] | Main Content Area |

| [Parameter Input Form] | |

|-------------|--------------------------------|

|- UserName | Login

|- Profile display | [Submit Button]

| -Starting Capital |[Pie Table Bullish Positions by percent]

| -Risk Tolerance |[Pie Table Bearish Positions by percent] |

|- Investment Horizon | |- register |

| | |

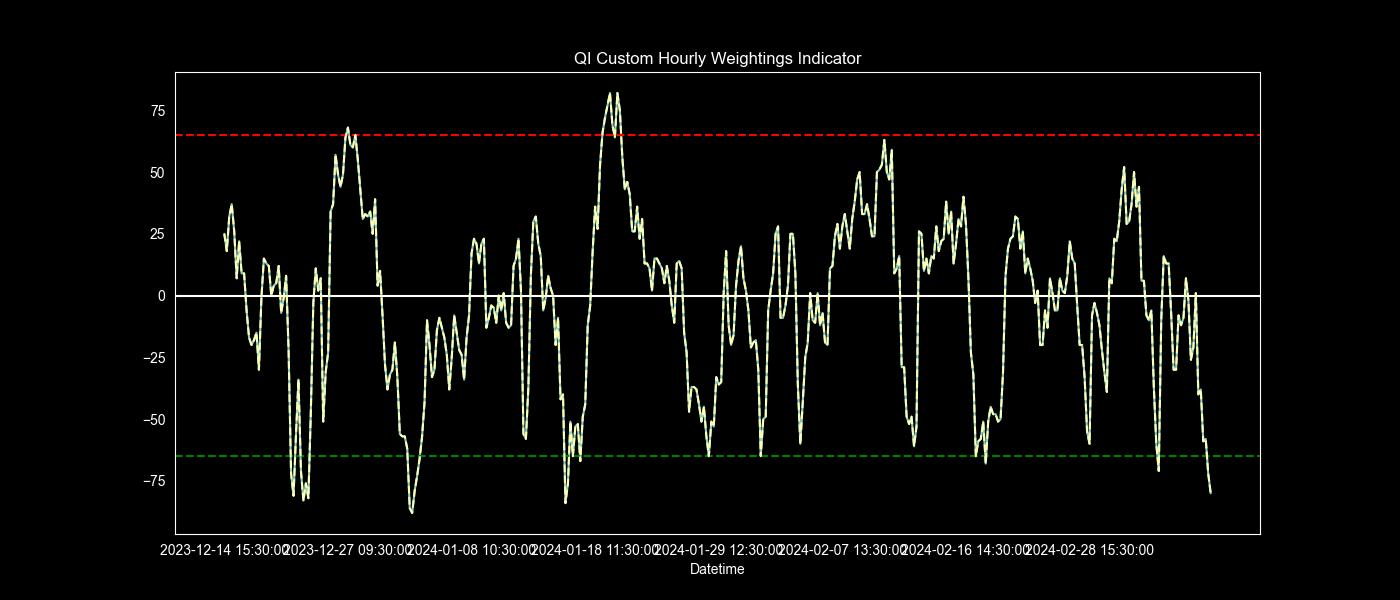
| | [Recommendation Area] |

| | - QI’s Portfolio Breakdown |

| | - Performance Charts |

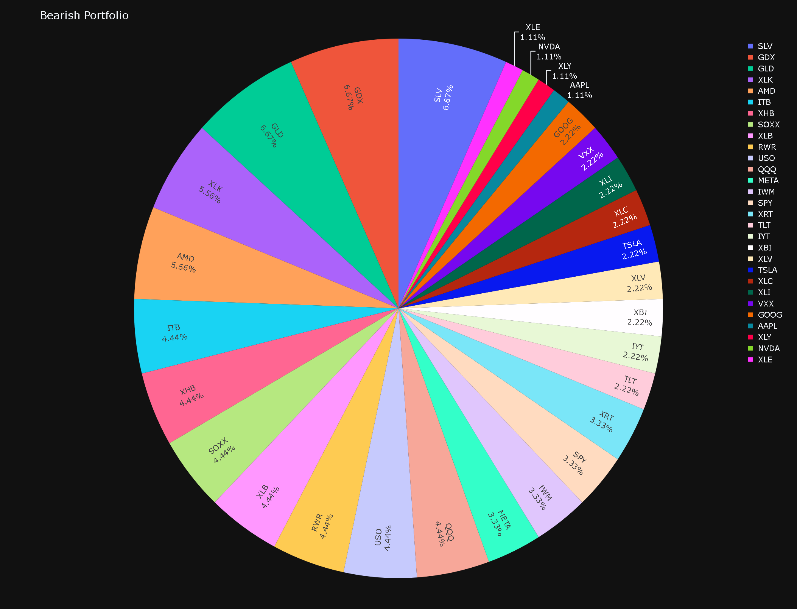
| | - Custom Multiplier Chart

QI’s Custom Multiplier Summation Indicator:



Example of the home page pie charts: side by side

A pie chart with different colored circles

Description automatically generated

Shows a breakdown of the portfolio allocations in percent. Followed by

**Details of the variables, forms, routes, and Style within the app:**

Styling-CSS:

-fonts = Great Vibes for headings and titles

Form content:

- Starting Capital

- Risk Tolerance

- Log In

- Register

- Investment Horizon

Backend:

Next is data storage and api retrieval

CI, IDX, Div ID, IMO, signals, portfolio allocation, Fibonacci Confluence zones and machine learning.

**CI, IDX, Corr:**

Run initialize

**INSTALL\_BASE\_PATH="$HOME/local"**

**cd ~**

**mkdir build**

**cd build**

**[ -f Python-3.11.6.tgz ] || wget <https://www.python.org/ftp/python/3.11.6/Python-3.11.6.tgz>**

**tar -zxvf Python-3.11.6.tgz**

**[ -f sqlite-autoconf-3240000.tar.gz ] || wget <https://www.sqlite.org/2018/sqlite-autoconf-3240000.tar.gz>**

**tar -zxvf sqlite-autoconf-3240000.tar.gz**

**cd sqlite-autoconf-3240000**

**./configure --prefix=${INSTALL\_BASE\_PATH}**

**make**

**make install**

**cd ../Python-3.11.6**

**LD\_RUN\_PATH=${INSTALL\_BASE\_PATH}/lib configure**

**LDFLAGS="-L ${INSTALL\_BASE\_PATH}/lib"**

**CPPFLAGS="-I ${INSTALL\_BASE\_PATH}/include"**

**LD\_RUN\_PATH=${INSTALL\_BASE\_PATH}/lib make**

**./configure --prefix=${INSTALL\_BASE\_PATH}**

**make**

**make install**

**cd ~**

**LINE\_TO\_ADD="export PATH=${INSTALL\_BASE\_PATH}/bin:\$PATH"**

**if grep -q -v "${LINE\_TO\_ADD}" $HOME/.bash\_profile; then echo "${LINE\_TO\_ADD}" >> $HOME/.bash\_profile; fi**

**source $HOME/.bash\_profile**

**docker build --pull . -t quantinvests\_2025\_devcontainer-app:latest**

**pip install pipenv**

**# The below command installs dependencies in the pipenv's virtualenv**

**pipenv install --dev**

**pip install virtualenv**

**# Activates virtualenv subshell**

**python -m pip install pip-tools**

**python -m ensurepip --upgrade**

**pip install --upgrade pip**

**pip install virtualenv**

**pip install -r requirements.txt**

**set FLASK\_APP app:"app.py"**

**cd source/repos/quantinvests\_2025**

**python3 -m venv env**

**pip install flask**

**c:\Users\joech\source\repos\quantinvests\_2025\env\Scripts\activate**

**pip install --upgrade --force-reinstall -r requirements.txt**

**https://download.visualstudio.microsoft.com/download/pr/a099e4b6-a6a8-4d34-bf95-b00739d35bb7/cdad50779717ba0e56caf89a3ba29ab1/dotnet-sdk-7.0.403-win-x64.exe**

**# <!-- pip install --upgrade --force-reinstall -r requirements.txt -->**

**cd source/repos/quantinvests\_2025**

**pip install virtualenv**

**pip install -r requirements.txt**

**set FLASK\_APP app:"app.py"**

**python -m venv env**

**git config core.autocrlf true**

**set env/bin/activate**

**set FLASK\_ENV=development**

**set FLASK\_DEBUG=1**

**run\_app.bat**

**@echo off**

**REM Navigate to your project directory**

**cd C:\Users\joech\source\repos\quantinvests\_2025**

**REM Activate the virtual environment**

**call .virtualenvs\quantinvests\_2025-clal\_mt2\Scripts\activate**

**cd C:\Users\joech\source\repos\quantinvests\_2025\.devcontainer**

**pip3.11 install nbformat**

**REM Run Flask app**

**flask run**

**@REM pip install --upgrade pip setuptools wheel**

Pipenv

export FLASK\_APP=app.py

export FLASK\_ENV=development

docker build -t FINAL\_QI\_2025 .

docker run -p 3000:3000 -d FINAL\_QI\_2025

python -m ensurepip --upgrade

pip install virtualenv

python -m venv env

pipenv install requests

pipenv install flask\_caching

pipenv install flask\_marshmallow

pipenv install flask\_marshmallow

pipenv install flask\_caching

pipenv install Flask-Caching as flask\_caching

pipenv install flask\_bcrypt

pipenv install flask-bcrypt

pipenv update

pipenv shell

flask run

**Github Checkout**

***git*** checkout Buddha23-RGB-QI

**# Make your changes**

***git*** add .

***git*** commit -m "Your commit message Aplir8"

***git*** push origin Buddha23-RGB-QI

Setup.bat

@echo off

REM Set Flask app

set FLASK\_APP=app.py

REM Set Flask environment variables

set FLASK\_ENV=development

set FLASK\_DEBUG=1

REM Navigate to your project directory

cd C:\Users\joech\source\repos\quantinvests\_2025

REM Create a Python virtual environment

python -m venv .virtualenvs\quantinvests\_2025-clal\_mt2

REM Activate the virtual environment

call .virtualenvs\quantinvests\_2025-clal\_mt2\Scripts\activate

REM Upgrade pip

python -m pip install --upgrade pip

REM Install required Python packages

pip install -r requirements.txt

REM Check the Python environment

python --version

pip list

REM Run Flask app

flask run

**README**

**Quant**

**Quantitative Strategies: Detailed description of your quantitative strategies.**

**Historical Data: Relevant historical financial data for backtesting.**

**Risk Parameters: Your risk tolerance and any specific risk parameters.**

**Investment Universe: The assets or securities you are interested in.**

**Trading Constraints: Any constraints such as transaction costs, liquidity, etc.**

**Performance Metrics: The metrics you want to use to evaluate the performance of the strategy.**

**pip install pyinstaller**

**cd C:\Users\joech\source\repos\quantinvests\_2025**

**REM Activate the virtual environment**

**call .virtualenvs\quantinvests\_2025-clal\_mt2\Scripts\activate**

**Navigate to the directory containing your Python script (your\_script.py) in the terminal.**

**/myproject**

**/myproject**

**\_\_init\_\_.py**

**app.py**

**config.py**

**/models**

**\_\_init\_\_.py**

**user.py**

**portfolio.py**

**transaction.py**

**/forms**

**\_\_init\_\_.py**

**registration\_form.py**

**login\_form.py**

**transaction\_form.py**

**portfolio\_form.py**

**/templates**

**index.html**

**login.html**

**register.html**

**portfolio.html**

**user\_portfolio.html**

**disclaimer.html**

**privacy.html**

**contact.html**

**chart.html**

**about.html**

**/tables**

**symbol.html**

**/static**

**/css**

**style.css**

**/js**

**script.js**

**/images**

**symbol.png**

**run.py**

**requirements.txt**

**In this structure:**

**app.py is your main Flask application script.**

**config.py contains your Flask configuration classes.**

**The models directory contains your SQLAlchemy models.**

**The forms directory contains your Flask-WTF forms.**

**The templates directory contains your Jinja2 templates.**

**The static directory contains your static files, such as CSS, JavaScript, and images.**

**run.py is the script you run to start your Flask application.**

**requirements.txt lists the Python packages that your project depends on.**

**Remember to adjust this structure based on your actual project structure and requirements.**

**Run PyInstaller:**

**bash**

**pyinstaller --onefile app.py**

**This command tells PyInstaller to create a single executable file (--onefile) from your\_script.py.**

**After running the command, PyInstaller creates a dist directory in the same directory as your script. Inside dist, you'll find your standalone executable file named your\_script.exe.**

**You can now distribute your\_script.exe as a PC application. Users can run it on their machine without needing to have Python installed.**

**Remember to test your executable thoroughly on the target operating system to ensure it works as expected. Also, be aware that complex applications with many dependencies might require additional steps or options with PyInstaller.**

**ased on the information you've provided, your Flask application seems to have the necessary components. However, a typical Flask application might include the following files and directories:**

**Templates: A templates directory that contains HTML templates. You've mentioned portfolio.html, register.html, login.html, and index.html, so it seems you have this covered.**

**Static Files: A static directory that contains static files such as CSS stylesheets, JavaScript files, and images.**

**Forms: A Python file (e.g., forms.py) that defines WTForms form classes. You've mentioned RegistrationForm and LoginForm, so it seems you have this covered.**

**Models: A Python file (e.g., models.py) that defines SQLAlchemy model classes. You've mentioned User and Portfolio, so it seems you have this covered.**

**Blueprints: A Python file (e.g., portfolio.py) that defines a Flask blueprint. You've mentioned portfolio\_bp, so it seems you have this covered.**

**Database: A SQLite database file (e.g., stocks.db). You've mentioned this in your app.py file, so it seems you have this covered.**

**Main Application File: A Python file (e.g., app.py) that creates the Flask application and defines routes. You've provided this file, so it seems you have this covered.**

**Requirements File: A requirements.txt file that lists the Python packages your application depends on. This makes it easier for others (or yourself, in the future) to install the necessary packages.**

**Configuration File: A Python file (e.g., config.py) that contains configuration variables for your application. This can include things like the secret key, database URI, and any third-party API keys.**

**Quant Invests:**

**==============**

This project is a financial application that allows users to manage their stock portfolios. It uses Python, SQLAlchemy, and SQLite to create and manage a database of users and their stock portfolios.

Features

User Management: The application allows the creation of user accounts with a username, hashed password, starting capital, and current capital.

Stock Signals: The application reads stock signal data from CSV files. Each stock signal includes a symbol, signal price, sig\_div, sig\_ci, sig\_idx, multiplier, mult\_change, and position.

Portfolio Management: The application allows users to manage their stock portfolios. Each portfolio includes a user ID, symbol, company, shares, value, cash, time, and running value.

Technologies

Python: The application is written in Python, a powerful and flexible programming language.

SQLAlchemy: SQLAlchemy is a SQL toolkit and Object-Relational Mapping (ORM) system for Python. It provides a full suite of well-known enterprise-level persistence patterns, designed for efficient and high-performing database access.

SQLite: SQLite is a C library that provides a lightweight disk-based database. It allows accessing the database using a nonstandard variant of the SQL query language.

Pandas: Pandas is a software library for Python used for data manipulation and analysis. It is used in this application to read CSV files and manipulate data frames.

Werkzeug: Werkzeug is a comprehensive WSGI web application library. It is used in this application to generate password hashes.

How It Works

The application first creates a new user and adds the user to the database. It then reads the last 14 periods from the CSV files and converts the data into a list of StockSignal objects. These StockSignal objects are added to the session and committed to the database. The application can then query the database to retrieve the user and their portfolio.

**Commit\_changes.bat**

**# github**

@echo off

REM Navigate to your project directory

cd C:\Users\joech\source\repos\quantinvests\_2025

REM Commit changes to Git

git add .

git commit -m "Your commit message: Dec 2023"

git push

REM Set the upstream branch for the first time

git push --set-upstream origin quantinvests\_2025

git config --global core.autocrlf true

REM Configure Git to automatically set the upstream branch for new branches

@REM git config --global push.default current

git push --set-upstream origin quantinvests\_2025

**app.spec**

# -\*- mode: python ; coding: utf-8 -\*-

a = Analysis(

['app\\app.py'],

pathex=[],

binaries=[],

datas=[],

hiddenimports=[],

hookspath=[],

hooksconfig={},

runtime\_hooks=[],

excludes=[],

noarchive=False,

)

pyz = PYZ(a.pure)

exe = EXE(

pyz,

a.scripts,

a.binaries,

a.datas,

[],

name='app',

debug=False,

bootloader\_ignore\_signals=False,

strip=False,

upx=True,

upx\_exclude=[],

runtime\_tmpdir=None,

console=True,

disable\_windowed\_traceback=False,

argv\_emulation=False,

target\_arch=None,

codesign\_identity=None,

entitlements\_file=None,

)

Importing Our List of Stock Symbols

==============================

* The first thing we need to do is import the constituents of the stocks and markets.
* These constituents change over time, so in an ideal world you would connect directly to the index provider (Standard & Poor's) and pull their real-time constituents on a regular basis.
* Currently, filtered and sorted 955 symbols, need further categorize them into type(ie. Etfs, benchmarks or dependants
* Symbol List, Sector w/ underlying companies symbols and stats

As of 2-6-2022 here is a list of the companies being considered:

====================================================

['A','AA','AAL','AAP','AAPL','ABBV','ABC','ABMD','ABT','ACC','ACGL','ACHC','ACM','ACN','ADBE','ADI','ADM','ADNT','ADP','ADS','ADSK','ADT','AEE','AEP','AES','AFG','AFL','AGCO','AGIO','AGNC','AGO','AGR','AIG','AIV','AIZ','AJG','AKAM','AL','ALB','ALGN','ALK','ALKS','ALL','ALLE','ALLY','ALNY','ALSN','AMAT','AMCR','AMCX','AMD','AME','AMG','AMGN','AMH','AMP','AMT','AMZN','AN','ANAT','ANET','ANSS','ANTM','AON','AOS','APA','APD','APH','APLE','APTV','AR','ARE','ARMK','ARNC','ARW','ASB','ASH','ASML','ATH','ATO','ATR','ATVI','AVB','AVGO','AVT','AVY','AWI','AWK','AXP','AXS','AXTA','AYI','AZO','AZPN','BA','BAC','BAH','BAX','BBY','BC','BDN','BDX','BEN','BERY','BFAM','BG','BGCP','BHF','BIG','BIIB','BIO','BJ','BK','BKI','BKNG','BKR','BKU','BLD','BLK','BLL','BLUE','BMRN','BMY','BOH','BOKF','BPOP','BR','BRKR','BRO','BRX','BSX','BURL','BWA','BWXT','BXP','BYND',‘C','CABO','CACC','CACI','CAG','CAH','CARR','CASY','CAT','CB','CBOE','CBRE','CBSH','CBT','CC','CCI','CCK','CCL','CDAY','CDEV','CDK','CDNS','CDW','CE','CERN','CF','CFG','CFR','CFX','CGNX','CHD','CHE','CHH','CHK','CHTR','CHWY','CI','CIM','CINF','CIT','CL','CLH','CLR','CLX','CMA','CMCSA','CME','CMG','CMI','CMS','CNA','CNC','CNDT','CNK','CNP','CNQ','CNX','COF','COLM','COMM','CONE','COO','COP','COR','COST','COTY','CPA','CPB','CPRI','CPRT','CPT','CR','CRI','CRL','CRM','CSCO','CSGP','CSL','CSX','CTAS','CTLT','CTSH','CTVA','CTXS','CUBE','CVS','CVX','CW','CXP','CZR','D','DAL','DCI','DD','DE','DEI','DELL','DFS','DG','DGX','DHC','DHI','DHR','DIS','DISCA','DISCK','DISH','DKS','DLB','DLR','DLTR','DNB','DOCU','DOV','DOW','DOX','DPZ','DRE','DRI','DTE','DUK','DVA','DVN','DXC','DXCM','EA','EAF','EBAY','ECL','ED','EEFT','EFX','EHC','EIX','EL','ELS','EMN','EMR','ENR','EOG','EPAM','EPR','EQC','EQH','EQIX','EQR','EQT','ERIE','ES','ESI','ESRT','ESS','ETN','ETR','ETSY','EVBG','EVR','EVRG','EW','EWBC','EXAS','EXC','EXEL','EXP','EXPD','EXPE','EXR','F','FAF','FANG','FAST','FB','FBHS','FCNCA','FCX','FDS','FDX','FE','FFIV','FHB','FHN','FICO','FIS','FISV','FITB','FL','FLO','FLR','FLS','FLT','FMC','FNB','FND','FNF','FOX','FOXA','FRC','FRT','FSLR','FTI','FTNT','FTV','FWONA','FWONK','G','GD','GDDY','GDX','GE','GGG','GHC','GILD','GIS','GL','GLD','GLPI','GLW','GM','GNTX','GOOG','GOOGL','GPC','GPK','GPN','GPS','GRUB','GS','GT','GTES','GWRE','GWW','H','HAIN','HAL','HAS','HBAN','HBI','HCA','HD','HE','HEI','HES','HFC','HGV','HHC','HIG','HII','HIW','HLF','HLT','HOG','HOOD','HOLX','HON','HP','HPE','HPP','HPQ','HRB','HRC','HRL','HSIC','HST','HSY','HTA','HUBB','HUM','HUN','HXL','IAC','IART','IBKR','IBM','ICE','ICUI','IDXX','IEX','IFF','IGT','ILMN','INCY','INFO','INGR','INTC','INTU','INVH','IONS','IP','IPG','IPGP','IQV','IR','IRM','ISRG','IT','ITB','ITT','ITW','IVZ','IWM','IYT','J','JAZZ','JBGS','JBHT','JBL','JBLU','JCI','JEF','JKHY','JLL','JNJ','JNPR','JPM','JWN','K','KAR','KBE','KDP','KEX','KEY','KEYS','KHC','KIM','KLAC','KMB','KMI','KMX','KNX','KO','KOS','KR','KRC','KSS','KSU','L','LAMR','LAZ','LBRDA','LBRDK','LDOS','LEA','LECO','LEG','LEN','LFUS','LH','LHX','LII','LIN','LKQ','LLY','LMT','LNC','LNG','LNT','LOPE','LOW','LPLA','LRCX','LSI','LSTR','LSXMA','LSXMK','LULU','LUMN','LUV','LVS','LW','LYB','LYFT','LYV','M','MA','MAA','MAC','MAN','MANH','MAR','MAS','MASI','MAT','MCD','MCHP','MCK','MCO','MCY','MD','MDLZ','MDT','MDU','MET','MFA','MGM','MHK','MIC','MIDD','MKC','MKL','MKSI','MKTX','MLM','MMC','MMM','MNST','MO','MOH','MORN','MOS','MPC','MPW','MPWR','MRK','MRNA','MRO','MRVL','MS','MSCI','MSFT','MSI','MSM','MTB','MTCH','MTD','MTN','MU','MUR','NATI','NAVI','NBIX','NBR','NCLH','NCR','NDAQ','NDSN','NEE','NEM','NEU','NFG','NFLX','NI','NKE','NKTR','NLOK','NLSN','NLY','NNN','NOC','NOV','NOW','NRG','NRZ','NSC','NTAP','NTNX','NTRS','NUAN','NUE','NUS','NVDA','NVR','NVT','NWL','NWS','NWSA','NXPI','NYCB','O','OC','ODFL','OFC','OGE','OHI','OI','OKE','OKTA','OLED','OLLI','OLN','OMC','OMF','ON','ORCL','ORI','ORLY','OSK','OTIS','OUT','OXY','OZK','PACW','PAG','PANW','PAYC','PAYX','PB','PBCT','PBF','PCAR','PCG','PEAK','PEG','PEGA','PEN','PEP','PETS','PFE','PFG','PG','PGR','PGRE','PH','PHM','PII','PINC','PK','PKG','PKI','PLD','PM','PNC','PNFP','PNR','PNW','PODD','POOL','POST','PPC','PPG','PPL','PRGO','PRU','PSA','PSTG','PSX','PTC','PTEN','PVH','PWR','PXD','PYPL','QCOM','QGEN','QQQ','QRTEA','QRVO','R','RAD','RCL','RE','REG','REGN','RES','RF','RGA','RGLD','RHI','RIG','RJF','RL','RLGY','RMD','RNG','RNR','ROK','ROL','ROP','ROST','RPM','RRC','RS','RSG','RTX','RWR','RYN','SABR','SAGE','SATS','SBAC','SBNY','SBUX','SC','SCCO','SCHW','SCI','SEB','SEE','SEIC','SFM','SGEN','SHW','SIRI','SITC','SIVB','SIX','SJM','SKX','SLB','SLG','SLGN','SLM','SLV','SM','SMG','SNA','SNDR','SNPS','SNV','SO','SON','SOXX','SPB','SPG','SPGI','SPLK','SPR','SPY','SQ','SRC','SRCL','SRE','SRPT','SSNC','ST','STE','STL','STLD','STOR','STT','STWD','STX','STZ','SUI','SVC','SWCH','SWK','SWKS','SYF','SYK','SYY','T','TAP','TCBI','TDC','TDG','TDOC','TDS','TDY','TEAM','TECH','TEL','TER','TEX','TFC','TFSL','TFX','TGT','THG','THO','THS','TJX','TKR','TLT','TMO','TMUS','TOL','TPR','TPX','TREX','TRGP','TRIP','TRMB','TRN','TROW','TRU','TRV','TSCO','TSLA','TSN','TTC','TTWO','TWLO','TWO','TWOU','TWTR','TXN','TXT','TYL','UA','UAA','UAL','UBER','UDR','UFS','UGI','UHAL','UHS','UI','ULTA','UMPQ','UNH','UNIT','UNM','UNP','UNVR','UPS','URBN','URI','USB','USFD','USM','USO','UTHR','UUP','V','VC','VEEV','VFC','VIAC','VIACA','VICI','VIRT','VLO','VMC','VMI','VMW','VNO','VOYA','VRSK','VRSN','VRTX','VST','VTR','VVV','VXX','VZ','W','WAB','WAL','WAT','WBA','WBS','WBT','WCC','WDAY','WDC','WEC','WELL','WEN','WEX','WFC','WH','WHR','WLK','WLL','WLTW','WM','WMB','WMT','WPC','WRB','WRK','WSM','WSO','WST','WTFC','WTM','WTRG','WU','WY','WYNN','X','XBI','XEL','XHB','XLB','XLC','XLE','XLF','XLI','XLK','XLNX','XLP','XLU','XLV','XLY','XOM','XPO','XRAY','XRT','XRX','XYL','Y','YUM','YUMC','Z','ZBH', 'ZBRA','ZEN','ZG','ZION','ZNGA','ZTS']

## Congestion Index

## ================

## Identify Stockcycle:

## --------------------

## 1) If the Stock's Cycle is Trending

## - the absolute value of the CI has to be greater than 20

## 2) If the Stock's Cycle is Congestion Index

## - the absolute value of the CI has to be less than 20,

## - separate them from the stocks experiencing congestion in price Action

## Equation

## ========

## C = Closing Price

## X = Time Segment

## H = Highest High for the chosen Time Period

## L = Lowest Low for the chosen Time Period

## Noise minimization = Further Smoothed by a 3 Day EMA

## Index Fluctuations & Range of Readings:

## --------------------------------------

## Range is from 100 to -100

## Trending instrument's CI reading must be great than 20 for a long position

## less than -20 indicates a downturn in the instrument, short position

## Output

## ======

## List of Filtered/Usable Trending Ticker Symbols

## """

## # %%

## """

## Congestion Index

## ================

## """

## Trend Identification

## Congestion Index: +/-20 reading indicates market is trending (otherwise sideways/congested price action.

## Text Description automatically generated

C= Closing price, X=Time, H=Highest high for the time period, L= Lowest low. To minimize noise, add an additional 3 day EMA. (optional).

Python Code:

def congestion\_index(df, window):

col = 'CI\_' + str(window)

df['HH'] = df['High'].rolling(window=window).max()

df['LL'] = df['Low'].rolling(window=window).min()

df['ROC'] = (df['Close'] - df.Close[-window]) / df.Close[-window]

df['Div'] = (df['HH'] - df['LL']) / df['LL']

df[col] = df['ROC']/df['Div']

return df[col].ewm(span=3, adjust=False, min\_periods=3, ignore\_na=True).mean()

**Excel Formula:**

HH(E)=MAX (B22:B49) B=high column

LL(F)=MIN (C22:C49) C=Low Column

ROC(G)=(D49-D22)/D22\*100 D=Close Column

(HH-LL)/LL (H)=(E49-F49)/(f49+0.01)

CI (I)=G49/H49

**DX**

**def IDX(df, window):**

**col = 'IDX\_' + str(window)**

**HH = df.High.rolling(window=window).max()**

**LL = df.Low.rolling(window=window).min()**

**df[col] = ((df.Close - LL)\*100) / (HH-LL)**

**df = pd.DataFrame(df[col])**

**return df[col].ewm(span=3, adjust=False, min\_periods=3, ignore\_na=True).mean()**

**Intermarket Momentum Indicator**

**==============================**

**Indicator Oscillator**

**--------------------**

**1) If the Indicator has reached a peak above 80 and then falls below 80**

**- A Short signal is generated**

**2) If the Indicator has reached a trough below 20 and then rises above 20**

**- the absolute value of the CI has to be less than 20,**

**- separate them from the stocks experiencing congestion in price Action**

**Equation**

**========**

**I = Indicator current value**

**X = Time Segment**

**H = Highest indicator High for the chosen Time Period**

**L = Lowest indicator Low for the chosen Time Period**

**Noise minimization = Further Smoothed by a 3 Day EMA**

**Index Fluctuations & Range of Readings:**

**--------------------------------------**

**Range is from 100 to 0**

**Trending instrument's IMO reading must be less than 20 at n-1 & great than 20 at n**

**for a long position**

**Trending instrument's IMO reading must be great than 80 at n-1 & then less than 80**

**at n period to qualify for a short position**

**Output**

**======**

**List of Filtered/Usable Trending Ticker Symbols**

**def IMO(indicator, window):**

**cols = []**

**col = 'IMO\_' + str(indicator) + str(window)**

**HH = indicator.rolling(window=window).max()**

**LL = indicator.rolling(window=window).min()**

**df = pd.DataFrame(index=indicator.index)**

**df[col] = ((indicator - LL)\*100) / (HH-LL)**

**cols.append(col)**

**return df[cols].ewm(span=3, adjust=False, min\_periods=3, ignore\_na=True).mean()**

Quant

You can explain the starting portfolio to the user as follows:

The portfolio consists of various stocks, each represented by a symbol. For example, 'AAPL' represents Apple Inc.

The 'Price' column shows the current price of each stock.

The 'Signal\_div', 'Signal\_idx', 'Signal\_ci', 'Signal\_ds', and 'Signal\_cor' columns represent different signals or indicators that might influence your decision to buy or sell the stock. The meaning of these signals depends on your specific trading strategy.

The 'Mult\_change' column represents the change in the multiplier, which is a factor that determines how much money you want to invest in each stock.

The 'Multiplier' column represents the current multiplier for each stock. The multiplier directly influences the number of shares you hold for each stock. The higher the multiplier, the more shares you hold.

The 'Shares' column (which you calculated based on the price and the multiplier) represents the number of shares you currently hold for each stock.

The value of your portfolio is the sum of the value of all shares you hold. You can calculate the value of each stock by multiplying the number of shares by the current price, and then sum up these values to get the total value of your portfolio.

Please note that this is a simplified explanation. The actual details might vary depending on your specific trading strategy and the meaning of the signals and the multiplier in your strategy.

*from* dotenv *import* load\_dotenv

*import* os

*# !pip install schedule*

*# !pip install matplotlib*

*# Load environment variables from .env file if it exists*

load\_dotenv("C:/workspaces/Congestion/hourly/.github/.env")

*import* time

*import* yfinance *as* yf

*import* pandas *as* pd

*import* numpy *as* np

*import* commons

*from* commons *import* \*

*import* matplotlib.pyplot *as* plt

*#%%*

symbols = commons.short\_list

windows = commons.windows

style = matplotlib.style.use('dark\_background')

plt.style.use('dark\_background')

*# create our custom\_dark theme from the plotly\_dark template*

plt\_io.templates["custom\_dark"] = plt\_io.templates["plotly\_dark"]

*# set the paper\_bgcolor and the plot\_bgcolor to a new color*

plt\_io.templates["custom\_dark"]['layout']['paper\_bgcolor'] = '#30404D'

plt\_io.templates["custom\_dark"]['layout']['plot\_bgcolor'] = '#30404D'

*# you may also want to change gridline colors if you are modifying background*

plt\_io.templates['custom\_dark']['layout']['yaxis']['gridcolor'] = '#4f687d'

plt\_io.templates['custom\_dark']['layout']['xaxis']['gridcolor'] = '#4f687d'

sns.set\_style('whitegrid')

*class* CongestionIndexTrader:

*def* \_\_init\_\_(self, symbol, start, now, interval, windows):

        self.symbol = symbol

        self.start = start

        self.now = now

        self.interval = interval

        self.windows = windows

        self.df = None

*def* IMO\_function(self, indicator, n):

        LL = indicator.rolling(window=n).min()

        HH = indicator.rolling(window=n).max()

        IMO\_val = (indicator - LL) / (HH - LL) \* 100

*return* IMO\_val.ewm(span=2, min\_periods=3).mean()

*def* signal\_gen(self, IMO\_df):

        Trade\_ID = np.where(

            ((IMO\_df < 75) & (IMO\_df.shift(1) > 75)) | (IMO\_df < 25), -1, 0)

        Trade\_ID = np.where(

            ((IMO\_df > 25) & (IMO\_df.shift(1) < 25)) | (IMO\_df > 75), 1, Trade\_ID)

*return* Trade\_ID

*def* calculate\_window\_indicators(self):

*for* window in self.windows:

            HH = self.df['High'].rolling(window=window).max()

            LL = self.df['Low'].rolling(window=window).min()

            CI = self.df['Close'].pct\_change(window) / (HH.pct\_change(window) + 1e-9) \* 100

            IDX = (self.df['Close'] - LL) / (HH - LL) \* 100

            self.df[*f*'CI\_{window}'] = CI

            self.df[*f*'IDX\_{window}'] = IDX

            self.df[*f*'CI\_IMO\_{window}'] = self.IMO\_function(CI, window)

            self.df[*f*'IDX\_IMO\_{window}'] = self.IMO\_function(IDX, window)

            self.df[*f*'CI\_signal\_{window}'] = self.signal\_gen(self.df[*f*'CI\_IMO\_{window}'])

            self.df[*f*'IDX\_signal\_{window}'] = self.signal\_gen(self.df[*f*'IDX\_IMO\_{window}'])

            self.df[*f*'CI\_trend\_{window}'] = np.where(CI > 20, "Bullish", np.where(CI < -20, "Bearish", "Congested"))

            self.df[*f*"ROC\_{window}"] = ((self.df.Close - self.df.Close[-window]) / self.df.Close[-window]) \* 100

            std = self.df['Close'].rolling(window=window, min\_periods=5).std()

            mid\_band = self.df['Close'].rolling(window=window, min\_periods=5).mean()

            ATR = self.df['TR'].rolling(window=window).mean()

            DS = ((self.df['Close'] - mid\_band)/mid\_band) \* 100

            ID = ((self.df['Close']-mid\_band) + (2\*std))/(4\*std)

            self.df[*f*'std\_{window}'] = std

            self.df[*f*'mid\_band\_{window}'] = mid\_band

            self.df[*f*'upper\_band\_{window}'] = mid\_band + 2\*std

            self.df[*f*'lower\_band\_{window}'] = mid\_band - 2\*std

            self.df[*f*'ATR\_{window}'] = ATR

            self.df[*f*'lower\_keltner\_{window}'] = mid\_band - (ATR \* 1.5)

            self.df[*f*'upper\_keltner\_{window}'] = mid\_band + (ATR \* 1.5)

            self.df[*f*'DS\_{window}'] = DS.ewm(span=3, adjust=False).mean()

            self.df[*f*'ID\_{window}'] = ID.ewm(span=3, adjust=False).mean()

*def* snapshot(self):

        self.df = yf.download(self.symbol, start=self.start, end=self.now, interval=self.interval)

        self.df['date'] = self.df.index

        self.df['returns'] = np.log(self.df['Adj Close'] / self.df['Adj Close'].shift(1))

        self.df['log\_returns'] = np.log(self.df['Close'] / self.df['Close'].shift(1))

        self.df['TR'] = abs(self.df['High'] - self.df['Low'])

        self.calculate\_window\_indicators()

*return* self.df.fillna(0)

*def* plot\_trends\_and\_signals(self, window):

*# Ensure the DataFrame is not empty*

*if* self.df is None or self.df.empty:

                print("DataFrame is empty. No data to plot.")

*return*

*# Slice the DataFrame to the last 'slice\_' rows*

        slice\_ = 255

        df\_slice = self.df[-slice\_:]

*# Calculate the buffer for y-axis limits*

        buffer\_percent = 0.05  *# 5% buffer above and below the price range*

        price\_min = df\_slice['Close'].min()

        price\_max = df\_slice['Close'].max()

        price\_buffer = (price\_max - price\_min) \* buffer\_percent

*# Plot the closing price*

*# Save the figure and axis reference*

        fig, ax = plt.subplots(figsize=(14, 7))

        ax.plot(df\_slice.index, df\_slice['Close'],

                label='Closing Price', color='skyblue')

*# Plot CI\_signal as markers*

        ci\_long\_signals = df\_slice[df\_slice[*f*'CI\_signal\_{window}'] > 0]

        ci\_short\_signals = df\_slice[df\_slice[*f*'CI\_signal\_{window}'] < 0]

        ax.scatter(ci\_long\_signals.index,

                ci\_long\_signals['Close'], label='CI Long Signal', marker='^', color='lime')

        ax.scatter(ci\_short\_signals.index,

                ci\_short\_signals['Close'], label='CI Short Signal', marker='v', color='red')

*# Plot IDX\_signal as markers*

        idx\_long\_signals = df\_slice[df\_slice[*f*'IDX\_signal\_{window}'] > 0]

        idx\_short\_signals = df\_slice[df\_slice[*f*'IDX\_signal\_{window}'] < 0]

        ax.scatter(idx\_long\_signals.index,

                idx\_long\_signals['Close'], label='IDX Long Signal', marker='o', color='aqua', alpha=0.5)

        ax.scatter(idx\_short\_signals.index,

                idx\_short\_signals['Close'], label='IDX Short Signal', marker='x', color='fuchsia', alpha=0.5)

*# Highlight CI\_trend areas with more opaque colors*

        bullish\_trend = df\_slice[df\_slice[*f*'CI\_trend\_{window}'] == 'Bullish']

        bearish\_trend = df\_slice[df\_slice[*f*'CI\_trend\_{window}'] == 'Bearish']

        congested\_trend = df\_slice[df\_slice[*f*'CI\_trend\_{window}'] == 'Congested']

*# Plot bullish trend areas*

        ax.fill\_between(

            bullish\_trend.index, bullish\_trend['Close'], color='green', alpha=0.3, label='Bullish Trend')

*# Plot bearish trend areas*

        ax.fill\_between(

            bearish\_trend.index, bearish\_trend['Close'], color='red', alpha=0.5, label='Bearish Trend')

*# Plot congested trend areas*

        ax.fill\_between(congested\_trend.index,

                        congested\_trend['Close'], color='orange', alpha=0.3, label='Congested Trend')

*# Customize and show the plot*

        ax.set\_title(

*f*'Closing Price with CI and IDX Signals for {self.symbol}')

        ax.set\_xlabel('Date')

        ax.set\_ylabel('Price')

        ax.legend()

*# Set y-axis limits with buffer*

        ax.set\_ylim(price\_min - price\_buffer, price\_max + price\_buffer)

*# Save the figure*

        chart\_filename = *f*'/workspaces/Congestion/hourly/charts/trading\_chart\_{self.symbol}.jpg'

        fig.savefig(chart\_filename)

        plt.close(fig)  *# Close the figure to free up memory*

*# %%*

"""Plot"""

*# Set up date variables*

now = datetime.datetime.now()

start\_daily = now - datetime.timedelta(days=2500)

start\_hourly = now - datetime.timedelta(days=720)

start\_quarter = now - datetime.timedelta(days=80)

*# Set the style to 'dark\_background'*

plt.style.use('dark\_background')

*# Usage*

*# Initialize a DataFrame to store the best window for each symbol*

best\_windows\_df = pd.DataFrame(

    columns=['Symbol', 'Best Window', 'Performance'])

windows = commons.windows

*for* symbol in short\_list:

*# Initialize the trader with the necessary parameters*

    trader = CongestionIndexTrader(

        symbol=symbol, start=start\_hourly, now=now, interval='1h', windows=windows)

*# Download the data and calculate indicators*

    df\_snapshot = trader.snapshot()

    df = pd.DataFrame(df\_snapshot)

*# Plot trends and signals*

*# Use the first window size for plotting*

    trend\_path = *f*"C:/workspaces/Congestion/hourly/tables/{symbol}.csv"

    df.to\_csv(trend\_path)

*# Assuming this method exists in your class*

    trader.plot\_trends\_and\_signals(window=40)



DIVERGENCE

Bollinger Band Basics Ags Calculations

*# %%*

*import* cProfile

*from* commons *import* table\_path

*import* os

*import* commons

*from* commons *import* \*

*import* datetime

*# from hourly\_congestion import CongestionIndexTrader*

*import* pandas *as* pd

*# Usage# Usage*

symbols = commons.short\_list

*# Set up date variables*

now = datetime.datetime.now()

start\_hourly = now - datetime.timedelta(days=720)

*# Ignore warnings*

warnings.filterwarnings('ignore')

*# Set the style to 'dark\_background'*

plt.style.use('dark\_background')

*# %%*

*def* IMO\_function(indicator, n):

    LL\_200 = indicator.rolling(window=n).min()

    HH\_200 = indicator.rolling(window=n).max()

    IMO\_val = (indicator - LL\_200)\*100 / (HH\_200-LL\_200)

*return* IMO\_val.ewm(span=2, min\_periods=3).mean()

*def* reverse\_signal\_gen(Signal\_ID\_data):

    Trade\_ID = np.where(((Signal\_ID\_data < 75) & (

        Signal\_ID\_data.shift(1) > 75)) | (Signal\_ID\_data < 25), 1, 0)

    Trade\_ID = np.where(((Signal\_ID\_data > 25) & (Signal\_ID\_data.shift(1) < 25)) | (

        Signal\_ID\_data > 75), -1, Trade\_ID)

*return* Trade\_ID

*def* signal\_gen(Signal\_ID\_data):

    Trade\_ID = np.where(((Signal\_ID\_data < 75) & (

        Signal\_ID\_data.shift(1) > 75)) | (Signal\_ID\_data < 25), -1, 0)

    Trade\_ID = np.where(((Signal\_ID\_data > 25) & (Signal\_ID\_data.shift(1) < 25)) | (

        Signal\_ID\_data > 75), 1, Trade\_ID)

*return* Trade\_ID

*def* figures\_to\_html(figs, filename="dashboard.html"):

    dashboard = open(filename, 'w')

    dashboard.write("<html><head></head><body>" + "\n")

*for* fig in figs:

        inner\_html = fig.to\_html().split('<body>')[1].split('</body>')[0]

        dashboard.write(inner\_html)

*# linger Bands:   THIS is already accounted for in the congestion class*

*# def bollinger\_bands(df):*

*#     global windows*

*#     import numpy as np*

*#     Cols = df.columns*

*#     df = pd.DataFrame(df, columns=Cols, index=df.index)*

*#     df['returns'] = np.log(df['Adj Close']/df['Adj Close'].shift(1))*

*#     df['ewma'] = df['Adj Close'].ewm(span=40, min\_periods=40).mean()*

*#     df['pct\_returns'] = df['Adj Close'].pct\_change(1) \* 100*

*#     df['price'] = df['Adj Close'].copy()*

*#     df['log\_returns'] = np.log(df.Close/df.Close.shift(1))*

*#     df['direction'] = np.sign(df.log\_returns)*

*#     df['abs\_price\_diff'] = df.Close.diff()*

*#     df['std'] = df['Close'].rolling(window=40, min\_periods=2).std()*

*#     df['cusum\_exp'] = df.log\_returns.cumsum().apply(np.exp)*

*#     df['pct\_cumsum'] = df.pct\_returns.cumsum()*

*#     # df = pd.concat([df, adx, macd, rsi, atr], axis=1)*

*#     df = df.dropna()*

*#     df = df.rename(columns=lambda x: x)*

*#     data = copy.deepcopy(df)*

*#     data['TR'] = abs(data['High'] - data['Low'])*

*#     for n in windows:*

*#         data[f'std\_{n}'] = data['Close'].rolling(window=n, min\_periods=5).std()*

*#         data[f'mid\_band\_{n}'] = data['Close'].rolling(*

*#             window=n, min\_periods=5).mean()*

*#         data[f'upper\_band\_{n}'] = data[f'mid\_band\_{n}']+2\*data[f'std\_{n}']*

*#         data[f'lower\_band\_{n}'] = data[f'mid\_band\_{n}']-2\*data[f'std\_{n}']*

*#         data[f'ATR\_{n}'] = data['TR'].rolling(window=n).mean()*

*#         data[f'lower\_keltner\_{n}'] = data[f'mid\_band\_{n}'] - \*

*#             (data[f'ATR\_{n}'] \* 1.5)*

*#         data[f'upper\_keltner\_{n}'] = data[f'mid\_band\_{n}'] + \*

*#             (data[f'ATR\_{n}'] \* 1.5)*

*#         data[f'DS\_{n}'] = (data['Close'] - data[f'mid\_band\_{n}']*

*#                            )/data[f'mid\_band\_{n}'] \* 100*

*#         data[f'ID\_{n}'] = ((data['Close']-data[f'mid\_band\_{n}']) +*

*#                            (2\*data[f'std\_{n}']))/(4\*data[f'std\_{n}'])*

*#         data[f'DS\_{n}'] = data[f'DS\_{n}'].ewm(span=3, adjust=False).mean()*

*#         data[f'ID\_{n}'] = data[f'ID\_{n}'].ewm(span=3, adjust=False).mean()*

*#         # data[f'Strategy\_Results\_CI\_{n}'] = data[f'Signal\_CI\_{n}'].shift(*

*#         #     1) \* data.log\_returns*

*#         # data[f'Strategy\_Results\_IDX\_{n}'] = data[f'Signal\_IDX\_{n}'].shift(*

*#         #     1) \* data.log\_returns*

*#     return data.fillna(0)*

# **DIVERGENCE STRATEGIES:**

The selected Python code is part of a script for a trading strategy based on divergence indices. It uses the pandas library for data manipulation and plotly for visualization. The script is divided into several parts:

1. **Imports and setup**: The script imports necessary libraries and sets up some variables. It also sets the style for any plots to 'dark\_background'.
2. **Function definitions**: Two functions are defined:
   * figures\_to\_html(figs, filename="dashboard.html"): This function takes a list of plotly figures and a filename, and writes the figures to an HTML file.
   * DivergenceIndexTrader: This is a class that encapsulates the trading strategy. It has several methods:
     + \_\_init\_\_(self, start, now, interval, windows): The constructor sets up the trader with a start and end time, an interval, and a window size.
     + get\_data(self, symbol): This method reads a CSV file for a given symbol into a pandas DataFrame.
     + IMO\_function(self, indicator, n): This method calculates the IMO (Index of Market Overbought/Oversold) value for a given indicator and window size.
     + reverse\_signal\_gen(self, Signal\_ID\_data) and signal\_gen(self, Signal\_ID\_data): These methods generate trading signals based on the IMO values.
     + main(self, df\_dep, df\_ben, dependant, benchmark, n): This method calculates several divergence indices and trading signals for a given dependant and benchmark, and adds them to the dependant's DataFrame.
3. **Main script**: The main part of the script initializes a DivergenceIndexTrader and uses it to calculate divergence indices and trading signals for a list of symbols against a list of benchmarks. The results are saved to CSV files.

Hourly Divergence.py file:

*def figures\_to\_html(figs, filename="dashboard.html"):*

*dashboard = open(filename, 'w')*

*dashboard.write("<html><head></head><body>" + "\n")*

*for fig in figs:*

*inner\_html = fig.to\_html().split('<body>')[1].split('</body>')[0]*

*dashboard.write(inner\_html)*

*class DivergenceIndexTrader:*

*def \_\_init\_\_(self, start, now, interval, windows):*

*self.start = start*

*self.now = now*

*self.interval = interval*

*self.windows = windows*

*self.data = {}*

*def get\_data(self, symbol):*

*trend\_path = f"C:/workspaces/Congestion/hourly/tables/{symbol}.csv"*

*df = pd.read\_csv(trend\_path, index\_col='Datetime', parse\_dates=True)*

*self.data = pd.DataFrame(df)*

*return self.data*

*def IMO\_function(self, indicator, n):*

*LL\_200 = indicator.rolling(window=n).min()*

*HH\_200 = indicator.rolling(window=n).max()*

*IMO\_val = (indicator - LL\_200)\*100 / (HH\_200-LL\_200)*

*return IMO\_val.ewm(span=2, min\_periods=3).mean()*

*def reverse\_signal\_gen(self, Signal\_ID\_data):*

*Trade\_ID = np.where(((Signal\_ID\_data < 75) & (*

*Signal\_ID\_data.shift(1) > 75)) | (Signal\_ID\_data < 25), 1, 0)*

*Trade\_ID = np.where(((Signal\_ID\_data > 25) & (Signal\_ID\_data.shift(1) < 25)) | (*

*Signal\_ID\_data > 75), -1, Trade\_ID)*

*return Trade\_ID*

*def signal\_gen(self, Signal\_ID\_data):*

*Trade\_ID = np.where(((Signal\_ID\_data < 75) & (*

*Signal\_ID\_data.shift(1) > 75)) | (Signal\_ID\_data < 25), -1, 0)*

*Trade\_ID = np.where(((Signal\_ID\_data > 25) & (Signal\_ID\_data.shift(1) < 25)) | (*

*Signal\_ID\_data > 75), 1, Trade\_ID)*

*return Trade\_ID*

*def main(self, df\_dep, df\_ben, dependant, benchmark, n):*

*corr = df\_ben[f'ROC\_{n}'].rolling(n).corr(df\_dep[f'ROC\_{n}'])*

*corr\_ewm = corr.ewm(span=3, min\_periods=3).mean()*

*DIV\_ID = df\_ben[f'ID\_{n}']-df\_dep[f'ID\_{n}']*

*ID = DIV\_ID.divide(df\_dep[f'ID\_{n}']) \* 100*

*DS = df\_dep[f'DS\_{n}'] - df\_ben[f'DS\_{n}']*

*df\_dep[f'Signal\_DS\_{benchmark}\_{n}'] = np.multiply(DS, corr\_ewm)*

*df\_dep[f'Signal\_ID\_{benchmark}\_{n}'] = np.multiply(ID, corr\_ewm)*

*df\_dep[f'IMO\_ID\_{benchmark}\_{n}'] = self.IMO\_function(*

*df\_dep[f'Signal\_ID\_{benchmark}\_{n}'], n)*

*df\_dep[f'IMO\_DS\_{benchmark}\_{n}'] = self.IMO\_function(*

*df\_dep[f'Signal\_DS\_{benchmark}\_{n}'], n)*

*df\_dep[f'IMO\_corr\_{benchmark}\_{n}'] = self.IMO\_function(corr\_ewm, n)*

*df\_dep[f'Trade\_ID\_{benchmark}\_{n}'] = self.reverse\_signal\_gen(*

*df\_dep[f'IMO\_ID\_{benchmark}\_{n}'])*

*df\_dep[f'Trade\_DS\_{benchmark}\_{n}'] = self.reverse\_signal\_gen(*

*df\_dep[f'IMO\_DS\_{benchmark}\_{n}'])*

*df\_dep[f'Trade\_corr\_{benchmark}\_{n}'] = self.signal\_gen(*

*df\_dep[f'IMO\_corr\_{benchmark}\_{n}'])*

*return df\_dep[[f'Trade\_corr\_{benchmark}\_{n}', f'Trade\_DS\_{benchmark}\_{n}', f'Trade\_ID\_{benchmark}\_{n}']]*

*# Initialize DivergenceIndexTrader*

*divergence\_trader = DivergenceIndexTrader(start\_hourly, now, "1h", windows=40)*

*# Initialize an empty dictionary to store benchmark data*

*benchmark\_data = {}*

*# Loop over each benchmark to get the data*

*for benchmark in commons.benchmarks:*

*benchmark\_data[benchmark] = divergence\_trader.get\_data(benchmark)*

*for symbol in symbols:*

*print(f"Processing symbol: {symbol}")*

*dependant\_data = divergence\_trader.get\_data(symbol)*

*results = []*

*# Loop over each benchmark*

*for benchmark in commons.benchmarks:*

*divergence\_result = divergence\_trader.main(*

*dependant\_data, benchmark\_data[benchmark], symbol, benchmark, 40)*

*results.append(divergence\_result)*

*# Concatenate results*

*df\_results = pd.concat(results, axis=1)*

*# Concatenate results with dependant\_data*

*df\_final = pd.concat([dependant\_data, df\_results], axis=1)*

*# Save to a CSV file*

*table\_path = f"C:/workspaces/Congestion/hourly/signals/{symbol}.csv"*

*df\_final.to\_csv(table\_path)*

*# Initialize the DivergenceIndexTrader class.*

"""

Loop over each symbol in symbols.

For each symbol, it will get the data for that symbol and store it in dependant.

It will then loop over each benchmark in commons.benchmarks.

For each benchmark, it will get the data for that benchmark and store it in bench\_mark.

It will then calculate the divergence between dependant and bench\_mark using the main method, and append the result to the results list.

After looping over all benchmarks, it will concatenate all the results into a single DataFrame df.

It will then save df to a CSV file in the C:/workspaces/Congestion/hourly/signals/ directory, with the filename being the symbol name.

In this revised class:

The calculate\_window\_indicators method is now properly defined within the class and is responsible for calculating the indicators for each window size.

The snapshot method has been updated to call calculate\_window\_indicators for each window size.

The string formatting for dynamic window size in DataFrame column names has been corrected (replaced %7Bwindow%7D with {window}).

The backtest and analyze\_trade\_windows methods remain unchanged and should be included in the class as well.

Make sure to include the backtest and analyze\_trade\_windows methods in the class if they are not already there. The usage example demonstrates how to use

the class to perform a snapshot, backtest, and analyze the trade windows.

The \_\_init\_\_ method initializes the class with the stock symbol, start and end dates, interval, and window sizes for the analysis.

The IMO\_function method calculates the Inertia/Momentum Oscillator (IMO) value for a given indicator and window size.

The signal\_gen method generates trading signals based on the IMO values.

The snapshot method downloads the stock data using yfinance, calculates various metrics, and applies the CI and IMO calculations for each window size.

# By using a class, you encapsulate all related functionality, making the code more organized and easier to maintain. You can also easily extend the

# class with additional methods or modify existing ones without affecting the rest of the code.

- We import the necessary modules and classes.

- We set the `rel\_path` to the directory where the data is stored.

- We instantiate the `DivergenceIndexTrader` class with the appropriate parameters.

- We call the `diverge` method to calculate divergence for the specified ticker.

- We backtest the strategy using the `backtest` method.

- If the `trade\_log\_df` DataFrame is not empty, we analyze the trade windows and print the results.

- We identify the most profitable window based on the total profit/loss and print it.

Flask APP:

Quant Invests Where the Quant Invests For You! Author: Joe Chamoun Mobile: 403-463-8072 Website: quantinvests.com Email: Joechamoun23@gmail.com LinkedIn: https://ca.linkedin.com/company/quantinvests ======================================================== Details of the investment service app: ============================================= App Name: Quant Invests Tagline: Where the Quant Invests For You! Author: Joe Chamoun Contact Number: 403-463-8072 Website: quantinvests.com Email: Joechamoun23@gmail.com LinkedIn: https://ca.linkedin.com/company/quantinvests Logo: Front End: | Quant Invests | [Logo] Where the Quant Invests for You [Investment App Wireframe] | ------------------------------------------------| [Navigation Menu] | | [User Profile] [Settings] [Logout] | |----------------------------------------------| | [Sidebar] | Main Content Area | | [Parameter Input Form] | | |-------------|--------------------------------| |- UserName | Login |- Profile display | [Submit Button] | -Starting Capital |[Pie Table Bullish Positions by percent] | -Risk Tolerance |[Pie Table Bearish Positions by percent] | |- Investment Horizon | |- register | | | | | | [Recommendation Area] | | | - QI’s Portfolio Breakdown | | | - Performance Charts | | | - Custom Multiplier Chart QI’s Custom Multiplier Summation Indicator: provide a flask app that can provide the front end

Copilot

To create a Flask app that serves as the frontend for your investment service application, you’ll need to set up a Flask project with the necessary routes, templates, and static files. Here’s a blueprint to get you started:

1. **Project Structure**:
   * Organize your project into directories for templates, static files, and the main application.
2. QuantInvestsApp/
3. ├── app/
4. │ ├── templates/
5. │ │ ├── base.html
6. │ │ ├── index.html
7. │ │ └── dashboard.html
8. │ ├── static/
9. │ │ ├── css/
10. │ │ ├── js/
11. │ │ └── images/
12. │ ├── \_\_init\_\_.py
13. │ ├── routes.py
14. │ └── forms.py
15. ├── config.py
16. ├── run.py
17. **Flask Application Setup** (\_\_init\_\_.py):
    * Initialize your Flask app and configure it.

**Python**

from flask import Flask

def create\_app():

app = Flask(\_\_name\_\_)

# Additional configuration if needed

from . import routes

app.register\_blueprint(routes.bp)

return app

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

1. **Routes** (routes.py):
   * Define the routes for your application.

**Python**

from flask import Blueprint, render\_template

bp = Blueprint('main', \_\_name\_\_)

@bp.route('/')

def index():

return render\_template('index.html')

@bp.route('/dashboard')

def dashboard():

return render\_template('dashboard.html')

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

1. **Templates**:
   * Create HTML templates using Jinja2 syntax.
   * base.html will contain the common layout for your app.
   * index.html and dashboard.html will extend base.html and contain specific content for each page.
2. **Static Files**:
   * Place CSS, JavaScript, and image files in the static directory.
   * Link to these files in your HTML templates to style your frontend and add interactivity.
3. **Forms** (forms.py, optional):
   * If your app requires user input, define Flask-WTF forms.

**Python**

from flask\_wtf import FlaskForm

from wtforms import StringField, SubmitField

class InvestmentForm(FlaskForm):

starting\_capital = StringField('Starting Capital')

risk\_tolerance = StringField('Risk Tolerance')

investment\_horizon = StringField('Investment Horizon')

submit = SubmitField('Submit')

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

1. **Run the Application** (run.py):
   * Create a script to run your Flask app.

**Python**

from app import create\_app

app = create\_app()

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

1. **Dependencies**:
   * Ensure you have Flask and Flask-WTF installed.
   * Use a requirements.txt file to manage your Python dependencies.

This blueprint provides a basic structure for your Flask app. You can expand upon this by adding more routes, templates, and static files as needed for your application’s functionality. Good luck with your project! 🚀📈

Deploying a Flask app to a server involves several steps. Here’s a high-level overview of the process:

1. **Prepare Your App for Deployment**:
   * Ensure your app is production-ready by removing any development environment settings.
   * Use a production WSGI server like Gunicorn or uWSGI instead of the built-in Flask server.
2. **Create a Distribution File**:
   * Generate a wheel distribution file of your app using setuptools.
   * Install the wheel library if not already installed: pip install wheel.
   * Run python setup.py bdist\_wheel to create the distribution file.
3. **Choose a Hosting Service**:
   * Decide on a hosting service that supports Python applications, such as Heroku, AWS, or a VPS.
4. **Set Up the Server**:
   * Install the necessary software on the server (e.g., Nginx, Apache).
   * Configure a reverse proxy to pass requests to your Flask app.
5. **Secure Your App**:
   * Set up SSL/TLS for HTTPS.
   * Configure firewalls and security groups as needed.
6. **Deploy Your App**:
   * Transfer the distribution file to your server.
   * Set up a virtual environment on the server and install your app using pip.
   * Configure environment variables, including the SECRET\_KEY.
7. **Run Your App**:
   * Start your Flask app using the production WSGI server.
   * Ensure it’s running on the correct port and is accessible.
8. **Monitor and Maintain**:
   * Set up monitoring tools to keep track of your app’s performance and uptime.
   * Regularly update your app and server software to patch security vulnerabilities.

[For detailed instructions, you can refer to the Flask documentation on deploying to production1](https://flask.palletsprojects.com/en/2.2.x/tutorial/deploy/)[, or follow a tutorial specific to your chosen hosting service2](https://www.freecodecamp.org/news/how-to-build-a-web-application-using-flask-and-deploy-it-to-the-cloud-3551c985e492/). Remember to test your deployment thoroughly to ensure everything works as expected. Good luck with your Flask app deployment! 🚀📈