PROJECT 4 FINAL REPORT

ENGR 130 - Transforming Ideas to Innovation I & II

Section 001

DETECTING INSIDER TRADING SIGNALS

Layla Le
le306@purdue.edu
Team 12
04/28/2024

1. Introduction

1.1 Background

In 2012, the STOCK Act (Stop Trading on Congressional Knowledge Act) came into force to prohibit lawmakers from using nonpublic information to their own personal benefit, that is, in most cases, financial gain. It is known as the law to combat insider trading.

Insider trading refers to an unfair financial transaction in which the investor has an informational advantage over others. This applies, for example, to cases in which congress members who - through their exclusive status - knows before the general public that taxes for a specific commodity would rise, thus selling all the stocks they hold in companies that involve in the manufacturing and distributing of that commodity.

There are inherent drawbacks to this practice: insider trading makes the market more volatile and vulnerable to impulsive decisions - panic buying/selling. Moreover, when the game is "rigged" in such a way, the public would lose faith in the system - both Congress and the market itself. In addition, these would create biases that drive lawmakers to vote for the sake of their stock portfolio, not the people. A congressman might not pass the bill to tax electric cars not because they believe it does harm to their fellow citizens, but their because their stocks in Ford would crash.

The STOCK Act, however, could not eliminate all concerns: this legislation does not outright ban stock ownership. There are potential loopholes around this, which led to speculations that congressional insider trading took place when the market plunged in 2020, right before lockdown regulations were enforced. This led to the introduction of several bills that aim for a complete ban, such as Bipartisan Ban on Congressional Stock Ownership Act of 2023, but none of these have been passed.

Today, there are plenty of suspicions around insider trading and the true motives behind legislative moves in Congress. This has resulted in major doubts in market fairness, politicians' integrity, and social, financial, and political system in general.

1.2.. Existing solutions

a. The STOCK Act

Since 2012, U.S. Senators, Representatives, and their immediate families must disclose any trade over $1000 within 45 days of the transaction. Accordingly, are required to file Periodic Transaction Reports (PTRs) that would be publicly accessible for assets including stocks, crypto, bonds, etc.

a. Unusual Whales

In 2023, Unusual Whales released two exchange-traded funds (ETFs) that monitor Democratic and Republican Congress members' trades, respectively called NANC and KRUZ, using the PTRs required under the Stock Act.

b. Limitations & Opportunities

The mentioned existing solutions stop at reporting and tracking, and neither go a step further to calculate financial and statistical metrics to narrow down the exact windows of abnormal returns, correlate it with legislative actions, nor create plots for visual comparisons.

1.3. Project Goal and Statement of Interests

As a student double majoring in Business Analytics & Information Management and First Year Engineering going into Electrical Engineering, I am interested in developing an algorithm that can help detect patterns that indicate signs of insider trading in Congress to promote transparency in the political landscape of the stock market of the USA.

I aim to achieve this by leveraging, not competing with the currently available tools. In particular, I want to develop a program in Python that assists in monitoring any irregular

trading patterns in the US through Unusual Whale's NANC and KRUZ ETF and compares them against the market, reflected by the S&P 500. The program fetches these indexes' returns, goes on to evaluate some key financial metrics, then creates some visual plots for better trend and pattern comparison. It will also conduct event studies to flag unusual abnormal returns on dates of specific legislative actions that serve to narrow down signals of potential insider trading.

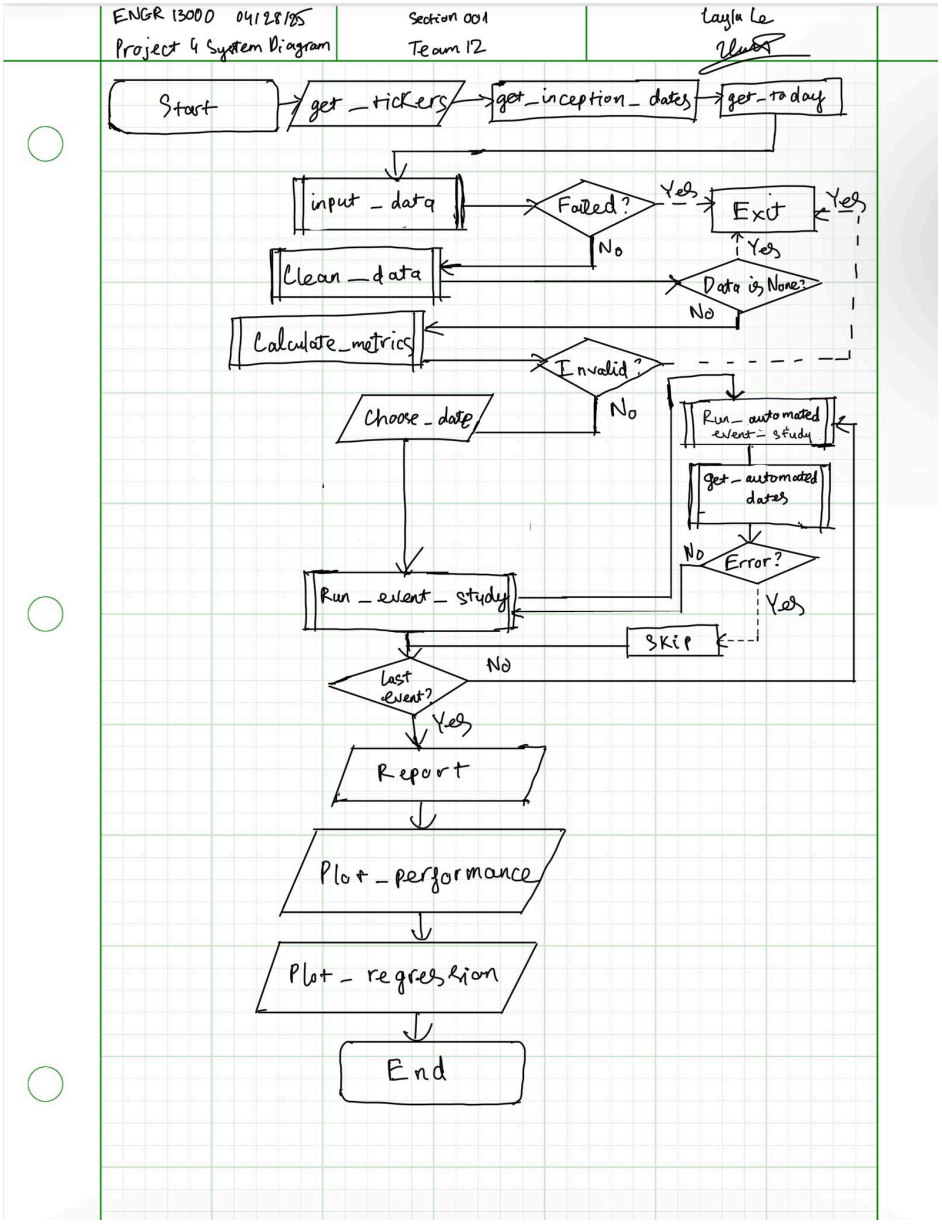
1.4. Specifications & Requirements

In order to fully meet the course requirements, the program has to be able to run without errors. It also has to include:

- More than 70 lines of code (Python)
- At least 3 out of 4 below:
    + for loop
    + while loop
    + list/array/vector/matrix
    + if decisional structure (at least one else or elseif/elif)
- Nested structures
- Functions
- Input and output
- Error checking
- Required comments

2. Program Description

2.1. System Diagram - Flowcharting



2.2. Libraries

```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import scipy.stats as stats
from datetime import datetime, timedelta
import requests #processing API requests
```

yfinance: Unofficial library of Yahoo Finance. Used to retrieve closing prices of specified stickers without having to manually input data or upload files.
pandas: Used to manipulate different data types
numpy: Used to perform numerical operations
matplotlib.pyplot: Used for plotting and data visualization
sklearn.linear_model.LinearRegression: Used to build linear regression model
datetime: Date formatting
requests: access Web and APIs

2.3. Inputs

The program automatically fetches the data of historical ETF and other indexes in the portfolio from Yahoo Finance yfinance and information of legislative bilsl through the Congress.gov API via requests).

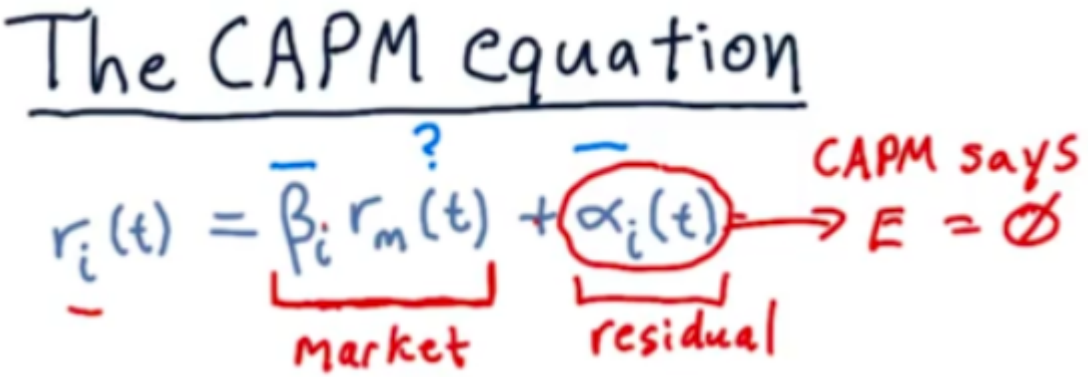
The usual manually a date on which the program should conduct event study on in specified format. Input in the wrong format will trigger the program to ask for input again, repeated until the program accepts the input date.

```
Enter event date (YYYY-MM-DD): 1
Invalid date format. Please use YYYY-MM-DD.
```

2.4. Outputs:

a. Calculated financial metrics:

Volatility: Explain index's fluctuation rates
Sharpe ratio: Risk-adjusted return
Beta: Variable explained by the CAPM model. How sensitive are the returns compared to the market index.
Alpha: Variable explained by the CAPM model. How much did the index outperform or underperform compared to the market index



b. Event study results on manually and automatically chosen event dates

Cumulative Abnormal Returns (CAR): The total amount of return is unexplained by the market.

t-statistics, p-values: Used to explain whether an event's impact is significant. For this program, p-value under 0.05 indicates statistical significance (Confidence Interval 95%)

For automatic event study, only the statistics of significant event dates are displayed

c. Plots

Performance plot: Comparing general trend of 2 ETFs against the market

Linear Regression Plot: Explaining the relationship between 2 ETFs and the market

2.5. User-defined functions

a. Input

get_tickers():

- Parameters: None
- Returns a hardcoded list of 24 tickers including bipartisan ETFs (NANC, KRUZ), the S&P 500 index (^GSPC), and some other index that we care about.

get_inception_dates():

- Parameters: None
- Return a dictionary assigning inception date, hardcoded to be NANC's inception date, to each ticker
- Prevent faulty analysis from non-existent data

get_today():

- Parameters: None
- Uses the datetime library to return today's date as a string in 'YYYY-MM-DD' format
- Dynamically sets the end date of the analysis window and ensures consistent data update

choose_date():

- Parameters: None
- Prompts the user to enter a date for manual event study testing and make sure it is in the right format

input_data():

- Parameters: None

- Return raw downloaded price DataFrame for further processing

- Uses yfinance to download adjusted daily close prices of chosen from Yahoo Finance starting from inception dates to current date. Stores inception dates in the DataFrame's attrs attribute for later use. Check for data fetching fail error.

b. Computing

clean_data(data):

- Parameters: data (raw price data downloaded from yfinance)
- Returns: Cleaned pandas DataFrame for further processing
- Cleans the raw data by removing price entries before each ticker's inception date, use forward fills to replace missing values, drop rows with all empty values, and reassign inception dates to the cleaned data set.

run_event_study(data, ticker, event_date, event_window, estimation_window, market_col):
- Parameters:
  - data: cleaned pandas DataFrame
  - ticker: strings of tickers in portfolio
  - event_date: event day
  - event_window: tuple defining how long the event last
  - estimation_window: tuple defining normal period for comparison
  - market_col: string of market index ticker (^GSPC)
- Returns: Statistic metrics to determine causes significant abnormal return
  - car (Cumulative Abnormal Return)
  - t_stat
  - p_value
- Performs a linear regression of ETF returns against S&P 500 returns during an estimation window. Calculates expected returns in the event window, computes abnormal returns, and sums them to obtain CAR.
- Uses scipy to calculate the t-statistic and p-value to test if the CAR is statistically significant.

get_automated_dates(api_key, max_pages = 20):
- Parameters:
  - api_key: string (for Congress.gov access)
  - max_pages: how many pages of results to fetch (Hardcoded to be 20)

- Returns a list of dictionaries containing event name, ticker being analyzed and event data to feed to prepare for event study analysis

- Request bill data from Congress.gov API Alternates ETF assignment between NANC and KRUZ for analysis. Filters out invalid dates and stores event metadata.

run_automated_event_study(api_key, data, p_thresh = 0.05):

- Parameters:
  - api_key: string (for Congress.gov access)
  - data: cleaned price DataFrame

  - p-thresh: significance threshold (hardcoded to be 0.05)

- Filters events with valid event date. Applies the run_event_study to each valid event prints significant events

calculate_metrics(data):

- Parameters: cleaned price DataFrame

- Returns: Dictionary of financial metrics per ticker to derive relationship between ETFs and the broader market
  - Annualized volatility
  - Sharpe ratio (using risk-free rate of 2%)
  - Maximum drawdown
  - Regression metrics: alpha, beta, R^2
  - Fits a linear regression of each ETF against the S&P 500 to estimate market sensitivity

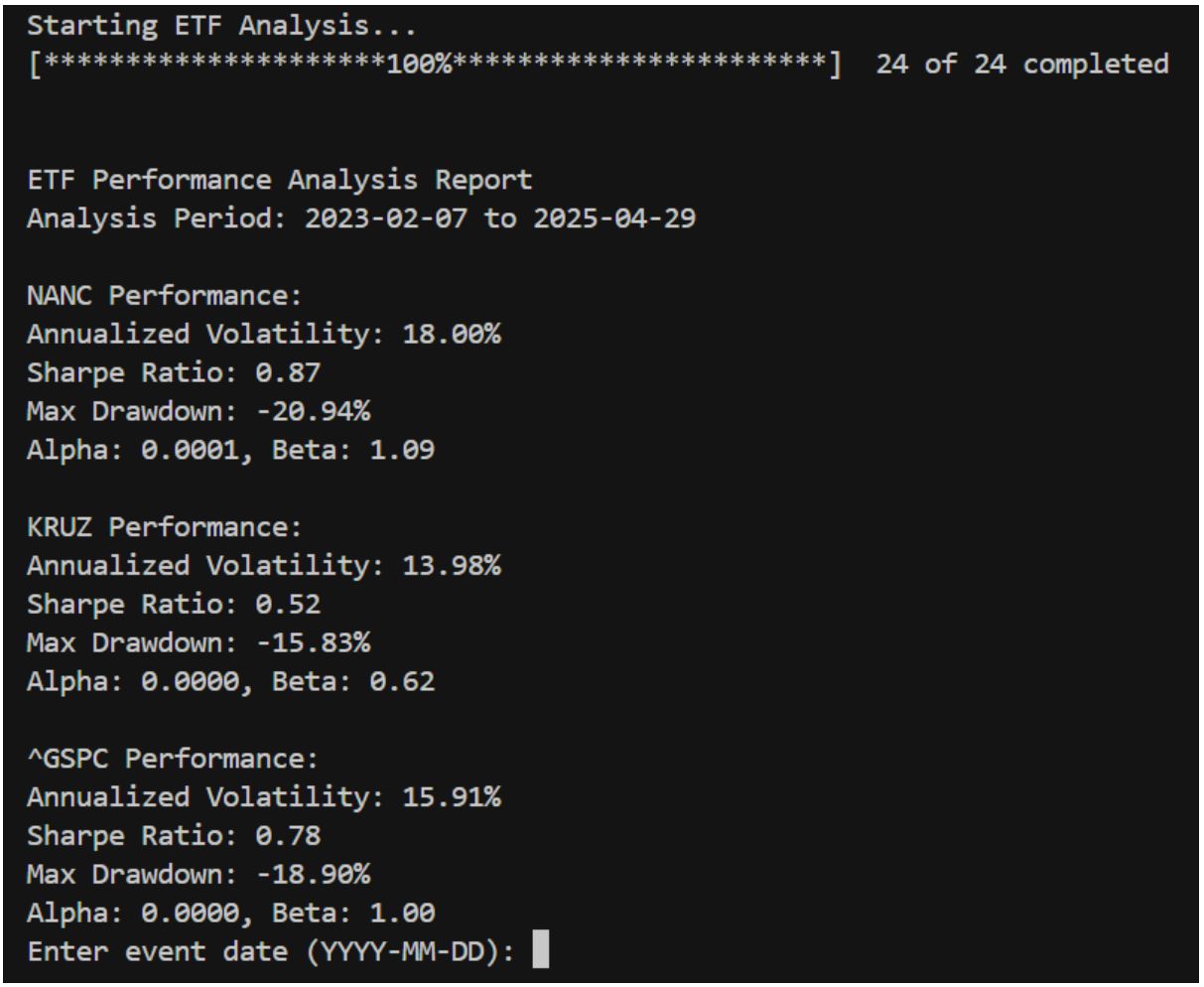
c. Output

plot_performance(data):
- Parameters: cleaned price DataFrame
- Normalize data to 100 and plot side-by-side performance comparisons of NANC and KRUZ to visualize how they performed relative to the market over time

plot_regression(results, data):
- Parameters:
  - results: financial metrics dictionary from calculate_metrics
- data: cleaned return DataFrame
- Creates scatter plots of daily ETF returns vs. S&P 500 returns and adds regression lines using sklearn predictions.

3. User manual

The program automatically starts calculating financial metrics of NANC, KRUZ and S&P 500.



The program will then ask the user to enter the event date in (YYYY-MM-DD) format. If the date format is incorrect, the program will repeat asking until a valid date format is accepted.

```
Enter event date (YYYY-MM-DD): 1
Invalid date format. Please use YYYY-MM-DD.
Enter event date (YYYY-MM-DD): 3
Invalid date format. Please use YYYY-MM-DD.
Enter event date (YYYY-MM-DD): 2025-01-01


Manual Back-test '2025-01-01':
NANC error on '2025-01-01': Event date not in record
KRUZ error on '2025-01-01': Event date not in record
Skipping S&P 500 vs itself
QQQM error on '2025-01-01': Event date not in record
XOM error on '2025-01-01': Event date not in record
LMT error on '2025-01-01': Event date not in record
TEMP error on '2025-01-01': Event date not in record
AAPL error on '2025-01-01': Event date not in record
MSFT error on '2025-01-01': Event date not in record
```

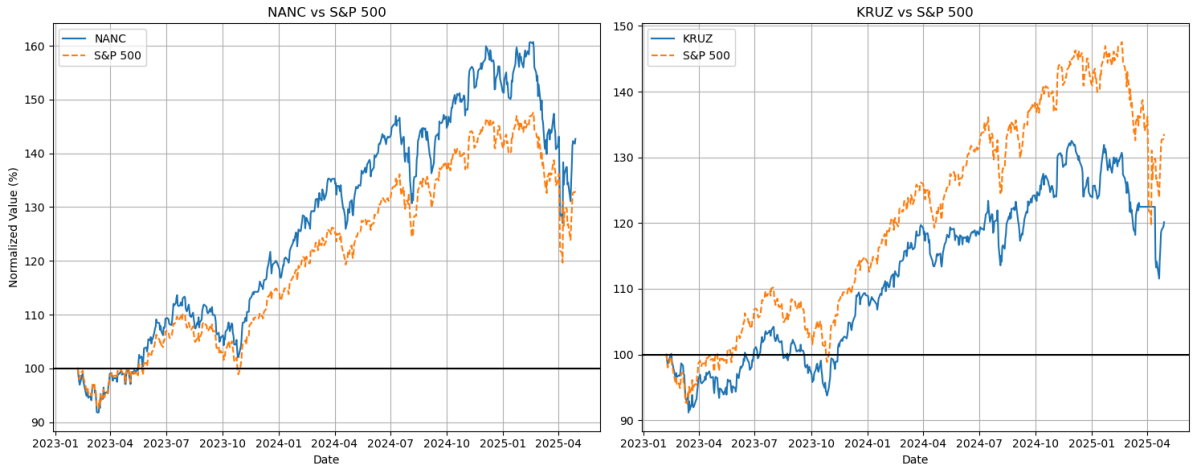
If the date chosen is in valid format but has no recorded data, the program displays: “Event date not in record”. This is usually because the day is a weekend, holiday, etc. - when Yahoo Finance Platform temporarily halts price recording. The user can stop the program and enter another date, or not, the program will continue anyway.

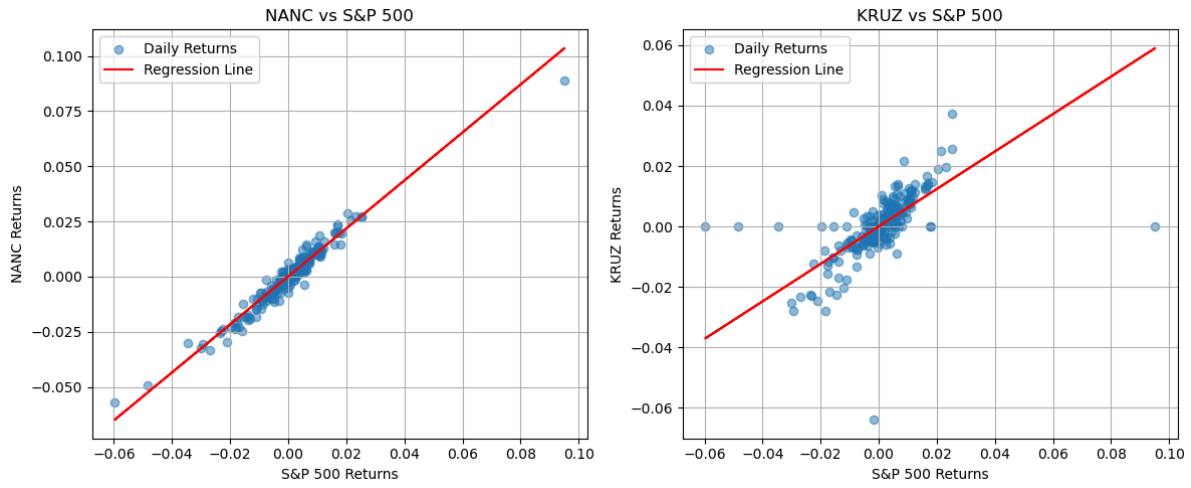
```
Enter event date (YYYY-MM-DD): 2024-12-30


Manual Back-test '2024-12-30':
NANC on '2024-12-30' → CAR -0.37%, t-stat -0.40, p-value 0.689 Not significant
KRUZ on '2024-12-30' → CAR 0.77%, t-stat 0.47, p-value 0.636 Not significant
Skipping S&P 500 vs itself
QQQM on '2024-12-30' → CAR -0.13%, t-stat -0.09, p-value 0.932 Not significant
XOM on '2024-12-30' → CAR 4.45%, t-stat 0.84, p-value 0.406 Not significant
LMT on '2024-12-30' → CAR -1.12%, t-stat -0.21, p-value 0.835 Not significant
TEMP on '2024-12-30' → CAR -1.19%, t-stat -0.41, p-value 0.681 Not significant
AAPL on '2024-12-30' → CAR -5.48%, t-stat -1.31, p-value 0.193 Not significant
MSFT on '2024-12-30' → CAR -1.55%, t-stat -0.43, p-value 0.666 Not significant
AMZN on '2024-12-30' → CAR -1.10%, t-stat -0.19, p-value 0.848 Not significant
GOOGL on '2024-12-30' → CAR 5.20%, t-stat 1.01, p-value 0.313 Not significant
META on '2024-12-30' → CAR 2.11%, t-stat 0.35, p-value 0.725 Not significant
NVDA on '2024-12-30' → CAR 2.71%, t-stat 0.29, p-value 0.775 Not significant
TSLA on '2024-12-30' → CAR -6.51%, t-stat -0.43, p-value 0.666 Not significant
BRK-B on '2024-12-30' → CAR -0.58%, t-stat -0.15, p-value 0.883 Not significant
JPM on '2024-12-30' → CAR 4.69%, t-stat 0.72, p-value 0.474 Not significant
V on '2024-12-30' → CAR -2.32%, t-stat -0.49, p-value 0.622 Not significant
UNH on '2024-12-30' → CAR 6.98%, t-stat 1.04, p-value 0.302 Not significant
PG on '2024-12-30' → CAR -7.07%, t-stat -1.58, p-value 0.117 Not significant
HD on '2024-12-30' → CAR 1.27%, t-stat 0.27, p-value 0.791 Not significant
MA on '2024-12-30' → CAR -3.07%, t-stat -0.78, p-value 0.435 Not significant
DIS on '2024-12-30' → CAR -5.96%, t-stat -1.02, p-value 0.309 Not significant
XLV on '2024-12-30' → CAR 2.89%, t-stat 1.09, p-value 0.278 Not significant
VDE on '2024-12-30' → CAR 10.29%, t-stat 2.21, p-value 0.029 ✅ Significant
```

If the input date is in record, the statistical values of all the indexes in the portfolio will be shown, along with a claim of whether the event on that date is significant to that index.

The plot will not be impacted by user input, and automatically updates with real-time data.





After the plots are displayed, the user should close the figure tabs so that the program continues with the automated event study. Only significant events will be displayed. The output should look something like this:

```
Significant Events (p < 0.05)
To redesignate certain facilities at Paterson Great Falls National Historical Park in honor of Congressman Bill Pascrell, Jr. (KRUZ 2025-04-17): CAR -3.57%, t-stat -2.13, p-value 0.035 ✅ Signi
ficant
To amend the Aquifer Recharge Flexibility Act to clarify a provision relating to conveyances for aquifer recharge purposes. (KRUZ 2025-04-17): CAR -3.57%, t-stat -2.13, p-value 0.035 ✅ Signifi
cant
To direct the Secretary of the Treasury to issue Clean Energy Victory Bonds. (KRUZ 2025-04-17): CAR -3.57%, t-stat -2.13, p-value 0.035 ✅ Significant
To amend the John D. Dingell, Jr. Conservation, Management, and Recreation Act to establish the Cerro de la Olla Wilderness in the Río Grande del Norte National Monument and to modify the bound
ary of the Río Grande del Norte National Monument. (KRUZ 2025-04-17): CAR -3.57%, t-stat -2.13, p-value 0.035 ✅ Significant
To amend title 5, United States Code, to provide for the publication, by the Office of Information and Regulatory Affairs, of information relating to rules, and for other purposes. (KRUZ 2025-0
4-17): CAR -3.57%, t-stat -2.13, p-value 0.035 ✅ Significant
To amend the Public Health Service Act to support the development and implementation of programs using data analysis to identify and facilitate strategies to improve outcomes for children in ge
ographic areas with a high prevalence of trauma from exposure to adverse childhood experiences, and for other purposes. (KRUZ 2025-04-17): CAR -3.57%, t-stat -2.13, p-value 0.035 ✅ Significant
To establish a Commission on the Federal Regulation of Cannabis to study a prompt and plausible pathway to the Federal regulation of cannabis, and for other purposes. (KRUZ 2025-04-17): CAR -3.
57%, t-stat -2.13, p-value 0.035 ✅ Significant
Providing for congressional disapproval under chapter 8 of title 5, United States Code, of the rule submitted by the Office of the Comptroller of the Currency of the Department of the Treasury
relating to the review of applications under the Bank Merger Act. (NANC 2025-04-10): CAR 2.36%, t-stat 2.03, p-value 0.045 ✅ Significant
COAST Anti-Drilling Act of 2025 (NANC 2025-04-10): CAR 2.36%, t-stat 2.03, p-value 0.045 ✅ Significant
Cape Fox Land Entitlement Finalization Act of 2025 (NANC 2025-04-10): CAR 2.36%, t-stat 2.03, p-value 0.045 ✅ Significant
RESILIENCE Act of 2025 (NANC 2025-04-10): CAR 2.36%, t-stat 2.03, p-value 0.045 ✅ Significant
To continue Executive Order 14220 in effect indefinitely. (NANC 2025-04-10): CAR 2.36%, t-stat 2.03, p-value 0.045 ✅ Significant
```

```
47/380 events significant (12.4% of total valid events)
```

The user can modify the size of data they want to study by changing the max_pages parameter (set to 20 at default) in get_automated_data user-defined function. The more pages of bills are studied, the more events are involved.

```
# CONGRESS.GOV SIGNIFICANT EVENTS
def get_automated_dates(api_key, max_pages=70):
```

```
Significant Events (p < 0.05)
To redesignate certain facilities at Paterson Great Falls National Historical Park in honor of Congressman Bill Pascrell, Jr. (KRUZ 2025-04-17): CAR -3.57%, t-stat -2.13, p-value 0.035 ✅ Signi
ficant
To amend the Aquifer Recharge Flexibility Act to clarify a provision relating to conveyances for aquifer recharge purposes. (KRUZ 2025-04-17): CAR -3.57%, t-stat -2.13, p-value 0.035 ✅ Signifi
cant
To direct the Secretary of the Treasury to issue Clean Energy Victory Bonds. (KRUZ 2025-04-17): CAR -3.57%, t-stat -2.13, p-value 0.035 ✅ Significant
To amend the John D. Dingell, Jr. Conservation, Management, and Recreation Act to establish the Cerro de la Olla Wilderness in the Río Grande del Norte National Monument and to modify the bound
ary of the Río Grande del Norte National Monument. (KRUZ 2025-04-17): CAR -3.57%, t-stat -2.13, p-value 0.035 ✅ Significant
To amend title 5, United States Code, to provide for the publication, by the Office of Information and Regulatory Affairs, of information relating to rules, and for other purposes. (KRUZ 2025-0
4-17): CAR -3.57%, t-stat -2.13, p-value 0.035 ✅ Significant
To amend the Public Health Service Act to support the development and implementation of programs using data analysis to identify and facilitate strategies to improve outcomes for children in ge
```

```
86/1380 events significant (6.2% of total valid events)
```

4. Appendix

```
"""
================================================================================
========
ENGR 13000 Spring 2025

Program Description
    This program analyzes, compares and visualizes the performance of
two bipartisan ETFs, NANC and KRUZ, against the S&P 500 index.
    It performs event studies to manually and automatically using
Congress.gov API to identify significant events that impact the
performance of these ETFs.
Assignment Information
    Assignment:     Ind Project 4
    Author:         Layla Le, le306@purdue.edu
    Team ID:        001 - 12

Contributor:    Yi Ding (ECE 20875 instructor), yiding@purdue.edu
    My contributor(s) helped me:
    [X] understand the assignment expectations without
        telling me how they will approach it.
    [X] understand different ways to think about a solution
        without helping me plan my solution.
    [ ] think through the meaning of a specific error or
        bug present in my code without looking at my code.
Contributor:    Lulu Zeng (MGMT 41310 instructor), zeng135@purdue.edu
    My contributor(s) helped me:
    [X] understand the assignment expectations without
        telling me how they will approach it.

    Note that if you helped somebody else with their code, you
    have to list that person as a contributor here as well.

ACADEMIC INTEGRITY STATEMENT
I have not used source code obtained from any other unauthorized
source, either modified or unmodified. Neither have I provided
access to my code to another. The project I am submitting
is my own original work.
```

```
================================================================================
========
"""


# Write any import statements here (and delete this comment).
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import scipy.stats as stats
from datetime import datetime, timedelta
import requests #processing API requests



#-------------------------------------------------
#  Inputs
#HARD-CODED TICKERS FOR SAMPLE PORTFOLIO INPUT
def get_tickers():
    return ['NANC', 'KRUZ', '^GSPC', 'QQQM', 'XOM',
            'LMT', 'TEMP', 'AAPL', 'MSFT', 'AMZN',
            'GOOGL', 'META', 'NVDA', 'TSLA', 'BRK-B',
            'JPM', 'V', 'UNH', 'PG', 'HD', 'MA',
            'DIS', 'XLV', 'VDE']

# ASSIGN INCEPTION DATES FOR SAMPLE PORTFOLIO
def get_inception_dates():
    tickers = get_tickers()
    return {tkr: '2023-02-07' for tkr in tickers}

#UPDATE TODAY'S DATE
def get_today():
    return datetime.today().strftime('%Y-%m-%d')

# USER INPUTS EVENT DATE FOR MANUAL EVENT STUDY
def choose_date():
    date = input("Enter event date (YYYY-MM-DD): ")
    try:
        return pd.to_datetime(date).strftime('%Y-%m-%d') #Convert
string to datetime
    except ValueError:
```

```
        print("Invalid date format. Please use YYYY-MM-DD.")
        return choose_date() #recurring function to ask for date again

# PORTFOLIO DATA INPUT
def input_data():
    try:
        data = yf.download(
            list(get_tickers()), #call function
            start=get_inception_dates()['NANC'], #Start date from
inception date",
            end=get_today(),
            auto_adjust=False
        )['Close'] #Retrieve closing prices of tickers from Yahoo
Finance
        data.attrs['inception_dates'] = get_inception_dates()
        # Here, I had to store a dictionary of ticker with their
corresponding inception dates in attrs attribute of data because pandas
does not allow adding attributes to data frame
        return data
    except Exception as e: #Error checking
        print(f"Data fetch failed: {e}")
        return None

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


#--------------------------------------------------
#  Computations
# DATA CLEANING
def clean_data(data):
    cleaned = data.copy() # Create a copy to clean to avoid changing
original data

    # Access inception_date dictionary from attrs
    inception_dates = data.attrs.get('inception_dates', {})

    for tkr, start in inception_dates.items():
        start_dt = pd.to_datetime(start) # Convert string to datetime
        cleaned.loc[cleaned.index < start_dt, tkr] = np.nan  # Ignore
data before inception dates

    cleaned = cleaned.ffill().dropna(how='all')  # Forward fill missing
values and drop rows with all NaNs
```

```
    # Reassign inception_dates back to attrs
    cleaned.attrs['inception_dates'] = inception_dates

    return cleaned


# EVENT STUDY
def run_event_study(data, ticker, event_date,
                    event_window=(-5, 10), estimation_window=(-120,
-21),
                    market_col='^GSPC'):
    #market_col = '^GSPC' #S&P 500 as market index
    event_date = pd.to_datetime(event_date) #Convert string to datetime
    returns = data[[ticker, market_col]].pct_change().dropna()
    #Calculate daily returns of the market index and other ticker
columns in the portfolio data frame by computing market changes in
percentage and skip empty rows

    if event_date not in returns.index:
        raise ValueError(f"Event date not in record") #Error checking

    # Estimation window to calculate the normal relationship between
stick and market returns
    loc = returns.index.get_loc(event_date) #Use pandas get_loc to find
the position of the event date in data frame
    est_start = loc + estimation_window[0] #Estimation window start
    est_end = loc + estimation_window[1] #Estimation window end

    # Process data in the estimation window
    estimation_returns = returns.iloc[est_start:est_end].dropna() #Drop
empty rows in the estimation window
    r_stock = estimation_returns[ticker] #Daily returns of the ticker
in the estimation window
    r_market = estimation_returns[market_col] #Daily returns of the
market index in the estimation window

    # Linear regression model
    model = LinearRegression().fit(r_market.values.reshape(-1, 1),
r_stock.values)
    alpha, beta = model.intercept_, model.coef_[0]

    # Event window
```

```
    evt_start = loc + event_window[0]
    evt_end = loc + event_window[1]
    event_returns = returns.iloc[evt_start:evt_end + 1].dropna() #last
event is inclusive
    r_evt_stock = event_returns[ticker].values
    r_evt_market = event_returns[market_col].values

#FINANCIAL METRICS IN CAPITAL ASSET PRICING MODEL, MGMT 41310
    expected = alpha + beta * r_evt_market
    abnormal = r_evt_stock - expected

    car = abnormal.sum()
    resid = r_stock.values - (alpha + beta * r_market.values)
    stderr = np.std(resid, ddof=1)
    se_car = stderr * np.sqrt(len(abnormal))
    t_stat = car / se_car
    p_value = 2 * (1 - stats.t.cdf(abs(t_stat), df=len(r_stock)-2))
#ECE 20875

    return car, t_stat, p_value


# STATISTICAL METRICS TO COMPARE BIPARTISAN ETFs VS MARKET
def calculate_metrics(data):
    returns = data.pct_change().dropna(how = 'all') #Calculate daily
returns, skip empty rows
    results = {
        'volatility': returns.std() * np.sqrt(252),
        'sharpe': (returns.mean() * 252 - 0.02) / (returns.std() *
np.sqrt(252)),
        'max_drawdown': (data / data.cummax() - 1).min(),
        'cumulative_returns': (returns + 1).cumprod()
    }
    X = returns['^GSPC'].values.reshape(-1, 1) #Fit market returns to
the model
    for etf in ['NANC', 'KRUZ', '^GSPC']: #Only run regression for the
two bipartisan ETFs and S&P 500
        y = returns[etf].values
        m = LinearRegression().fit(X, y)
        results.update({
            f'{etf}_alpha': m.intercept_,
            f'{etf}_beta': m.coef_[0],
            f'{etf}_r2': m.score(X, y),
```

```
            f'{etf}_model': m
        })
    return results


# CONGRESS.GOV SIGNIFICANT EVENTS
def get_automated_dates(api_key, max_pages=20):
    source =
f"https://api.congress.gov/v3/bill?api_key={api_key}&format=json"
#Calls API for congress.gov
    events = []
    ticker_idx = 0 #step count
    start_evt = data.index[0]  # first date in the dataset
    end_evt = data.index[-1]   # last date in the dataset
    for page in range(max_pages):
        url = f"{source}&offset={page*250}" #250 bills per page
        request = requests.get(url)
        if request.status_code != 200:
            print(f"API request failed:{request.status_code}") #Error
checking
            break

        bills = request.json().get("bills", []) #convert API response
from JSON to Python dictionary

        for bill in bills:
            date_str = bill.get("latestAction", {}).get("actionDate")
or bill.get("updateDate") #Get latest action date or update date
            if not date_str: #Error checking: if no bill is found for
this date
                print(f"No recorded bill of this date {date_str}")
                continue #skip to next bill
            event_date = datetime.strptime(date_str, "%Y-%m-%d")
            if not (start_evt <= event_date <= end_evt): #Eroror
checking: if event date out of range
                print(f"Event date {event_date} out of range")
                continue #skip to next bill

            ticker = ['NANC', 'KRUZ'][ticker_idx % 2] #alternate
between the two ETFs only after each step
            ticker_idx += 1 #step count update
            #store event list
            label = bill.get("title", "Untitled Bill")
```

```
            events.append({"label": label, "ticker": ticker,
"event_date": event_date.strftime("%Y-%m-%d")})
    return events

# BACKTEST SIGNIFICANT EVENTS

def run_automated_event_study(api_key, data, p_thresh=0.05):
    all_events = get_automated_dates(api_key)
    last_date = data.index[-1] #Last date data is recorded
    post_days = 10 #Post-event window (Events too recent might not have
enough data and low p-value regardless)
    valid_events = [] #List of valid events

# Filter out events that don't have a full post-event window (full
analysis of p-value not available)
    for evt in all_events:
        ev_dt = datetime.strptime(evt['event_date'], "%Y-%m-%d")
        if ev_dt + timedelta(days=post_days) <= last_date:
            valid_events.append(evt)

    print("\n\nSignificant Events (p < 0.05)")
    sig_count = 0
    for evt in valid_events:
        try:
            car, t_stat, p_val = run_event_study(data, evt['ticker'],
evt['event_date'])
            if p_val < p_thresh:
                sig_count += 1
                print(f"{evt['label']} ({evt['ticker']}
{evt['event_date']}): CAR {car:.2%}, "
                      f"t-stat {t_stat:.2f}, p-value {p_val:.3f} ✅
Significant")
        except Exception: #Error checking
            continue

# Summary of significant events
    total = len(valid_events)
    if total > 0:
        pct = sig_count / total * 100
        print(f"{sig_count}/{total} events significant ({pct:.1f}% of
total valid events)")
    else:
        print("No events with a full post-event window to analyze.")
```

```
#-----------------------------------------------------



#-----------------------------------------------------
#  Outputs
# VISUALIZATION
def plot_performance(data):
    fig, axes = plt.subplots(1, 2, figsize=(15, 6))
    for ax, etf in zip(axes, ['NANC', 'KRUZ']):
        start = pd.to_datetime(get_inception_dates()[etf]) #Start date
from inception date
        etf_px = data[etf].loc[start:] #Access ETF data since its
inception date
        first = etf_px.first_valid_index() #Valid starting point
        spx_px = data['^GSPC'].loc[first:]  # Get S&P 500 data from the
same date
        norm_etf = etf_px / etf_px.loc[first] * 100  # Normalize ETF
data
        norm_spx = spx_px / spx_px.loc[first] * 100  # Normalize S&P
500 data

        # Plot normalized data
        ax.plot(norm_etf, label=etf)
        ax.plot(norm_spx, '--', label='S&P 500')
        ax.axhline(100, color='black', linestyle='-')  # Baseline at
100%

        # Add labels and title
        ax.set_title(f"{etf} vs S&P 500")
        ax.set_xlabel('Date')
        ax.legend()
        ax.grid(True)

    # Add a shared y-axis label
    axes[0].set_ylabel('Normalized Value (%)')

    plt.tight_layout()
    plt.show()

#REGRESSION AND SCATTER PLOT
def plot_regression(results, data):
```

```
    returns = data.pct_change().dropna()
    X = returns['^GSPC'].values.reshape(-1, 1)
    fig, axes = plt.subplots(1, 2, figsize=(12, 5))

    for ax, etf in zip(axes, ['NANC', 'KRUZ']):
        model = results[f'{etf}_model'] #regression model for each ETF
        y = returns[etf].values
        y_pred = model.predict(X)

        ax.scatter(X, y, alpha=0.5, label='Daily Returns')
        ax.plot(X, y_pred, color='red', label='Regression Line')
        ax.set_title(f'{etf} vs S&P 500')
        ax.set_xlabel('S&P 500 Returns')
        ax.set_ylabel(f'{etf} Returns')
        ax.legend()
        ax.grid(True)

    plt.tight_layout()
    plt.show()

# MAIN FUNCTION
if __name__ == "__main__":
    print("Starting ETF Analysis...")
    raw = input_data()
    if raw is None:
        print("Data fetch failed. Exiting.")
        exit() #Error checking: Stop program if data input failed

    data = clean_data(raw)
    if data is None:
        print("Data cleaning failed.")
        exit()
    metrics = calculate_metrics(data)
    print(f"\n\nETF Performance Analysis Report")
    print(f"Analysis Period: {data.index[0].date()} to
{data.index[-1].date()}")
    for t in ['NANC', 'KRUZ', '^GSPC']:
        print(f"\n{t} Performance:")
        print(f"Annualized Volatility: {metrics['volatility'][t]:.2%}")
        print(f"Sharpe Ratio: {metrics['sharpe'][t]:.2f}")
        print(f"Max Drawdown: {metrics['max_drawdown'][t]:.2%}")
        print(f"Alpha: {metrics[f'{t}_alpha']:.4f}, Beta:
{metrics[f'{t}_beta']:.2f}")
```

```
    # Single-event study example
    date = choose_date()
    print(f"\n\nManual Back-test '{date}':")
    for ticker in get_inception_dates().keys():
        if ticker == '^GSPC':
            print("Skipping S&P 500 vs itself")
            continue
        try:
            car, t_stat, p_val = run_event_study(data, ticker, date)
            flag = "✅ Significant" if p_val < 0.05 else "Not
significant"
            print(f"{ticker} on '{date}' → CAR {car:.2%}, t-stat
{t_stat:.2f}, p-value {p_val:.3f} {flag}")
        except Exception as e:
            print(f"{ticker} error on '{date}': {e}")


run_automated_event_study("hNI3oldMUe5knB7iQ7fhftN04Orr5MGlI9a1YgNY",
data)
    plot_performance(data)
    plot_regression(metrics, data)
#-----------------------------------------------------
```