הגשה סופית – פרויקט סופי שיטות חיזוי בפינטק - אביב 2024

מגישים:
אלון פלדמן 316352103, זאב קפינוס 313590697, אלינור בורוכוב 325539120, רועי לצרס 316353903

<u>נכס</u>: מחיר הגז הטבעי ביום פתיחת הבורסה
מחיר הגז הטבעי מהווה מרכיב מרכזי בשוק האנרגיה העולמי, וכתוצאה מכך, תחזיות של מחירים אלו יכולות לספק מידע לסוחרים, משקיעים וחברות בתחום האנרגיה. היכולת לחזות את המחיר בפתיחת יום המסחר יכולה להשפיע על החלטות השקעה ולסייע בהבנת תנודות השוק.

<u>שוק</u>: שוק האנרגיה
שוק האנרגיה כולל מקורות אנרגיה מסורתיים כמו נפט גולמי וגז טבעי, מקורות מתחדשים כמו אנרגיית רוח וסולארית, ומקורות גרעיניים. הוא משפיע על כלכלות עולמיות, מדיניות ממשלתית, שווקים פיננסיים, תעשייה, תחבורה, והסביבה. שינויים במחירי האנרגיה יכולים להשפיע על צמיחה כלכלית, אינפלציה, ומטבעות לאומיים.

<u>הפיצ'רים</u>:
המודלים לחיזוי מחירי הגז הטבעי מסתמכים על מגוון רחב של פיצ'רים שמייצגים את הגורמים המרכזיים המשפיעים על השוק ומתבססים על מאמרים שמצאנו:

1. מחיר הנפט הגולמי: מכיוון שהנפט והגז הטבעי הם תחליפים פוטנציאליים, שינויים במחיר הנפט יכולים להשפיע על מחיר הגז.
2. שער הדולר: התנודתיות בשער הדולר משפיעה על מחירי האנרגיה, שכן הגז נסחר בדולרים בשוק הבינלאומי.
3. S&P 500: המדד משמש כאינדיקטור כללי למצב השוק והכלכלה, והשפעתו על נכסי אנרגיה קשורה לשינויים כלליים באמון המשקיעים.
4. מזג אוויר: טמפרטורות קיצוניות ועונת החורף בפרט משפיעות על הביקוש לגז טבעי לצורכי חימום וכן, מזג האוויר במדינות אשר אחראיות על ייצוא ועיבוד הגז הטבעי יכול להשפיע על מחירו.

<u>טווח זמן</u>: יומי
נרצה לחזות את מחיר הגז הטבעי ביום הפתיחה של הבורסה.
<u>מקורות הנתונים</u>:

- Yahoo Finance
- ספריית Meteostat

<u>הנדסת הנתונים</u>**:**

1. ניקוי נתונים: הסרת ערכים חסרים.
2. נרמול נתונים: הפיכת הפיצ'רים למדדים בסקאלה אחידה, מאחר שהמחירים בבורסה משתנים בהתאם למוצר, למשל, נפט גולמי מופיע כדולר לחבית וגז טבעי כדולר למיליון BTU.
3. הזזת נתונים: שימוש בנתוני עבר (lag) כדי להפיק פיצ'רים מבוססי היסטוריה.
4. שימוש במודל וחישוב מדדים סטטיסטים ומדדי סיכון תוך התייחסות רק לימי המסחר.

<u>קוד</u>: הקוד מכיל 3 קבצים של עבור כל אחד מהמודלים וקובץ נוסף של מודל ARIMA עם מזג אוויר, כאשר המידע ההיסטורי הוא מ- 1.1.2013 עד היום.
מופיע בנספחים ומצורף בקובץ הזיפ.
<u>תוצאות הרצה</u>:
הרצנו 3 מודלי חיזוי בהתאם למאמרים שמצאנו, רצינו לבדוק האם מודל פשוט של רגרסיה לינארית מקבל תוצאות טובות יותר מהמודלים האחרים, לאחר מכן למודל הטוב ביותר הוספנו מידע של נתוני מזג האוויר.

עבור מודל VAR התוצאות שקיבלנו היו לא מספיק טובות כאשר R^2 נמוך מאוד:
Mean Absolute Error: 1.4823521005905365
Root Mean Squared Error: 1.669165313885016
R^2 score: -11.498496590823581
True Volatility: 0.041740555884290105
Predicted Volatility: 0.0015904762761845188
True Value at Risk (VaR): -0.06194216850319372
Predicted Value at Risk (VaR): 0.00027983807380702486

עבור מודל הרגרסיה הלינארית קיבלנו תוצאות פחות טובות מבחינת $R^2, MAE, RMSE$ כפי ששיערנו:
Mean Absolute Error: 2.2513406817630006
Root Mean Squared Error: 2.3245427059652437
R^2 score: -23.240084656969838
True Volatility: 0.041740555884290105
Predicted Volatility: 0.017326465771336487
True Value at Risk (VaR): -0.06194216850319372
Predicted Value at Risk (VaR): -0.02933502173965448

עבור מודל ARIMA קיבלנו את התוצאות הטובות ביותר, אך עדיין R^2 שלילי:
Mean Absolute Error: 0.44742580736162374
Root Mean Squared Error: 0.5751860249078142
R^2 score: -0.4841429203375702
True Volatility: 0.041740555884290105
Predicted Volatility: 0.00013738589393414658
True Value at Risk (VaR): -0.06194216850319372
Predicted Value at Risk (VaR): -1.4943046800055156e-11

לפי המדדים, מודל ARIMA יצא הטוב ביותר, לכן החלטנו להוסיף למודל זה את נתוני מזג האוויר מערים נבחרות בארה"ב, למשל, יוסטון, טקסס נבחרה כי היא מעבדת ומייצאת גז טבעי וניו יורק סיטי, ניו יורק היא צרכנית גדולה וייתכן כי שינויי מזג האוויר ישפיעו השימוש בגז הטבעי ובכך על מחירו.
עקב הוספת נתוני מזג האוויר שיערנו כי התרדד הצורך בשימוש במדד ה- S&P 500 כאינדיקטור לביקושים בשוק האנרגיה, תוצאות המודל אוששו השערה זו כך שללא מדד זה ועם נתוני מזג האוויר קיבלנו תוצאות טובות יותר כאשר R^2 חיובי:

Mean Absolute Error: 0.14158198706930666
Root Mean Squared Error: 0.17369267773569044
R^2 Score: 0.8653569105397222
True Volatility: 0.04397971820549371
Predicted Volatility: 0.029209313747337522
True Value at Risk (VaR): -0.059738565784045516
Predicted Value at Risk (VaR): -0.03812107938488393
Sharpe Ratio: 0.07126134575776769
Sortino Ratio: 0.11887257188214163
Empirical Sharpe Ratio: 0.024339227531447863
Empirical Sortino Ratio: 0.046531240499548085

מודלי חיזוי (תיאור מפורט מתחת למדדי הסיכון):

תחזיות המחירים בוצעו בעזרת מספר מודלי חיזוי סטטיסטיים:

1. VAR**:** הוא מודל סטטיסטי שנעשה בו שימוש כדי לתפוס את הקשרים בין מספר משתנים בזמן כפי שהם משתנים עם הזמן וסוג של מודל תהליך סטוכסטי.
2. Linear Regression: מודל פשוט ובסיסי המתבסס על קשר ליניארי בין הפיצ'רים לבין מחיר הגז.
3. ARIMA**:** מודל סטטיסטי שנעשה בו שימוש לניתוח וחיזוי סדרות עתיות (רצף של תצפיות שנאמדו זו אחר זו, במרווח זמן נתון).
4. ARIMA with weather: הוספת פיצ'ר של מזג האוויר בערים מסוימות בארה"ב למודל ARIMA

מדדים סטטיסטיים:

כדי להעריך את ביצועי המודלים, נשתמש במספר מדדים סטטיסטיים:

1. Mean Absolute Error: שגיאה ממוצעת מוחלטת:
   - מחשב את הממוצע של ההבדלים המוחלטים בין הערכים החזויים לערכים האמיתיים
   - יעזור להבנה עד כמה התחזיות קרובות לערכים האמיתיים של מחירי הגז
   - ערך נמוך יותר מעיד על דיוק גבוה יותר
2. Root Mean Squared Error שורש ממוצע ריבועי השגיאות:
   - מחשב את השורש הריבועי של הממוצע של ריבועי ההפרשים בין הערכים החזויים לערכים האמיתיים
   - מדד זה רגיש לשגיאות גדולות ולכן יכול להדגיש מודלים שמייצרים תחזיות שגויות בצורה משמעותית
   - ערך נמוך יותר מעיד על דיוק גבוה יותר
3. R-squared (R^2) Score:
   - מדד המייצג את אחוז השונות
   - אם ה R^2-גבוה, זה מציין שהמודל מצליח להסביר חלק גדול מהשונות במחירי הגז
   - ערך של 1 מצביע על התאמה מצוינת, ערך של 0 מציין שהמודל לא מסביר את השונות מעבר לממוצע. ערך שלילי מעיד על חיזוי פחות טוב מהמודל הבסיסי

מדדי סיכון:

מדדי הסיכון יעזרו לנו להבין את הסיכון בהשקעה על פי המודל ולהבין כיצד המודל מתנהג ביחס לעולם האמיתי.

1. Volatility תנודתיות:
   - חישוב סטיית התקן של התשואות (שינוי באחוזים בערכי היעד)
   - משמש להערכת התנודתיות של השוק ויסייע לנו להעריך את רמת הסיכון הקשורה לחיזויים של מחירי הגז
   - ביצענו חישוב של הערך האמיתי לעומת המודל
2. מדד Value at Risk – VaR:
   - משמש כדי להעריך את הסיכון הכספי הצפוי של השקעות בגז טבעי בתקופה מסוימת.
   - שימושי להבין את הסיכון הפוטנציאלי של ההשקעות
   - ביצענו חישוב של הערך האמיתי לעומת המודל
3. מדד שארפ:
   - משמש כדי לאפיין עד כמה התשואה של נכס מפצה את המשקיע על הסיכון שהוא לוקח.
   - כאשר משווים שני נכסים, הנכס בעל מדד שארפ הגבוה יותר נותן תשואה גבוהה יותר באותה רמת סיכון. או לחלופין, הוא נותן אותה תשואה אך בסיכון נמוך יותר. בדרך כלל מומלץ למשקיעים לבחור השקעות בעלות מדד שארפ גבוה יותר.
4. מדד סורטינו:
   - בודק את עודף התשואה של התיק ליחידת סיכון ומאוד דומה למדד שארפ ההבדל בין המדדים הוא שמדד זה מתייחס לתנודתיות של התיק רק בחודשים עם התשואה השלילית
   - ככל שעודף התשואה ליחידת סיכון יהיה גבוה יותר מדד סורטינו יהיה גבוה יותר

<u>תיאורים מפורטים של המודלים</u>:

מודל VAR (Vector AutoRegression):

מודל VAR הוא מודל אוטו רגרסיבי רב-משתני, שבו כל משתנה תלוי באחרים מתוך קבוצה של משתנים.
כל משתנה במודל תלוי בזמן הנוכחי ובערכים קודמים של כל המשתנים האחרים במודל.
המודל משמש לניתוח סדרות עתיות של מספר משתנים שמקיימים יחסי תלות הדדיים.
המודל יוצר משתנים חדשים שמייצגים את הערכים הקודמים של המשתנים המקוריים, על מנת להציג את הקשרים העקיפים בין המשתנים, לאחר מכן, המודל מתאים את הקשרים בין כל המשתנים בהתבסס על מספר עיכובים מוגדרים (lags) מהעבר.
לבסוף, המודל משתמש בקשרים שהותאמו כדי לחזות ערכים עתידיים של המשתנים.

רגרסיה ליניארית:

רגרסיה ליניארית היא שיטה לניתוח קשרים בין משתנה תלוי לבין משתנים בלתי תלויים. המודל מניח קשר ליניארי בין המשתנים, כלומר, המשתנה התלוי הוא קו ליניארי של המשתנים הבלתי תלויים.
המודל בונה את הקשר הליניארי בין המשתנה התלוי (מחיר הגז הטבעי) לבין המשתנים הבלתי תלויים (סוגי סחורות ומדדים כלכליים).
המודל עושה שימוש בקשרים הליניאריים שנוצרו כדי לחזות ערכים עתידיים של המשתנה התלוי.

מודל ARIMA (AutoRegressive Integrated Moving Average):

מודל ARIMA הוא מודל לסדרות עתיות שנועד לחזות ערכים עתידיים על סמך הערכים הקודמים של הסדרה.
המודל כולל שלושה מרכיבים עיקריים: אוטורגרסיה ,(AR) אינטגרציה (I) וממוצע נודד(MA)

- אוטורגרסיה (AR): משתמש בערכים הקודמים של הסדרה לחיזוי ערכים עתידיים.
- אינטגרציה (I): מבצע הבדל (differencing) כדי להפוך את הסדרה לסטציונרית, כלומר לגרום לכך שסטטיסטיקות בסיסיות של הסדרה לא ישתנו עם הזמן.
- ממוצע נודד (MA): כולל את השגיאות הקודמות של המודל בתהליך החיזוי כדי לשפר את הדיוק.

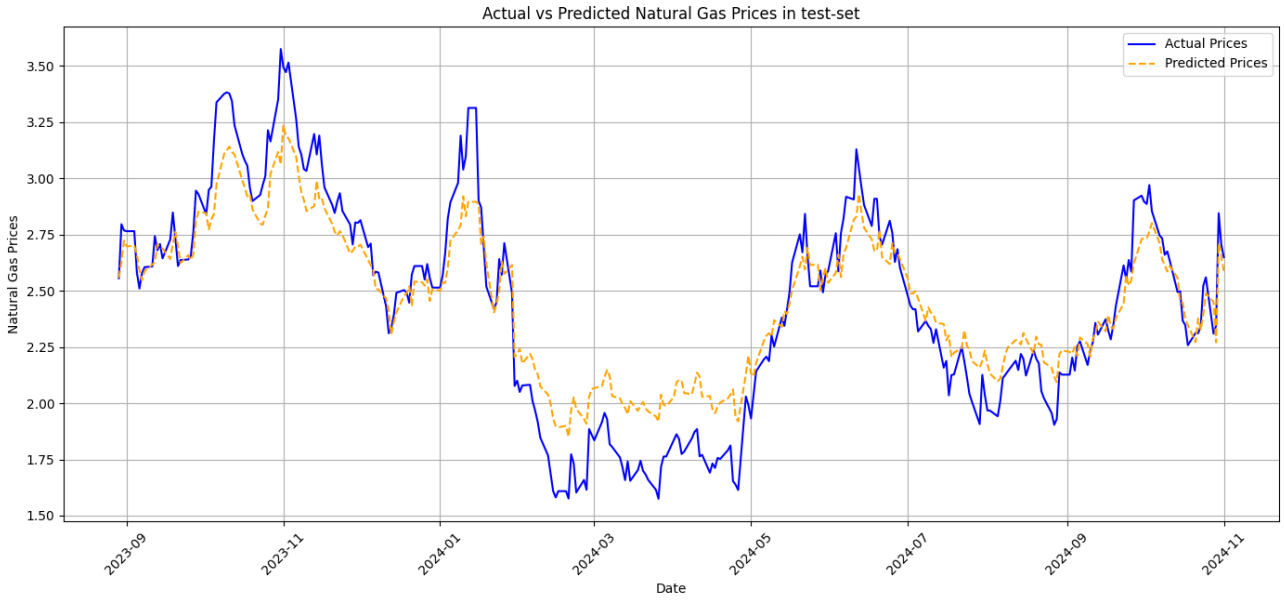
מודלים אלו נבחרו מכמה סיבות, ראשית, מפני שמחירי הגז נבדקו במודל על פני הזמן, הם למעשה סדרת עתית של תצפיות אשר נאמדו במרווח זמן נתון זו אחר זו, כאשר במקרה שלנו מרווח הזמן הוא יום עסקים.
מודל ARIMA כפי שפירטנו לעיל הוא מודל המותאם לסדרות עתיות, בנוסף, מאחר שהמודל שלנו מתבסס לא רק על נתוני העבר של מחירי הגז אלא גם על משתנים נוספים כמו מחיר הנפט, מזג האוויר ושער הדולר מודל זה התאים לצורך שלנו כי הוא מאפשר להוסיף משתנים חיצוניים.
שוק האנרגיה הוא תנודתי ולכן מודל שמתמודד עם מגמות עולות או יורדות באמצעות אינטגרציה כמו מודל ARIMA מתאים לניתוח שרצינו לבצע.
מודל VAR מתאים כאשר ישנם מספר משתנים שמשפיעים זה על זה וקיוונו כי המודל יתאים גם לחיזוי שלנו, בנוסף, המודל הופיע בכמה מאמרים שמצאנו בנוגע לחיזוי מחירי גז טבעי והאמנו כי הוא יתאים גם לצרכים שלנו.
כאמור, בהתאם לתוצאות מודל ARIMA סיפק את התוצאות הטובות ביותר לפי הערכת הביצועים שביצענו ובהתאם לכך הוספנו לנו את נתוני מזג האוויר.

<u>תיאור התוצאות</u>:

כעת נסביר את התוצאות שקיבלנו במודל הסופי המשלב את מודל ARIMA עם נתוני מזג אוויר:
קיבלנו מדדי שגיאה נמוכים שמעידות שהמודל מתמודד בצורה סבירה עם חריגות, שגיאת שורש ממוצע ריבועי RMSE מעט גבוהה יותר מה שיכול להעיד ששגיאות גדולות מתרחשות לעיתים רחוקות.
קיבלנו R^2 קרוב ל-1 שמצביע שהמודל מסביר כ-86.5% מהשונות בנתונים התלויים, כלומר המודל מתאים לנתונים, זהו הבדל משמעותי לעומת המודלים ללא נתוני מזג האוויר שהראו R^2 שלילי.
התנודתיות החזויה Volatility נמוכה מעט מהתנודתיות האמיתית, מה שמעיד על כך שהמודל עשוי להמעיט מעט בהערכת הסיכון. עם זאת, הקרבה בין הערכים מראה שהמודל מצליח ללכוד את מגמת התנודתיות הכללית בצורה טובה.

מדד VaR החזוי פחות שלילי מה VaR-האמיתי, מה שמצביע על כך שהמודל מספק הערכת סיכון מעט שמרנית (כלומר, תחזית להפסד קיצוני נמוך יותר).
למרות שזה לא אידיאלי לניהול סיכונים מדויק, התחזית היא בטווח סביר ומספקת הערכה זהירה.
נסתכל על מדד שארפ וסורטינו ביחד, שניהם חיוביים בעוד מדד סורטינו מעט גבוה יותר, כלומר ייתכן שהמודל מתפקד טוב יותר כאשר מתחשבים רק בתנודתיות שלילית.
כאשר מסתכלים על הנתונים האמפיריים של מדדים אלו קיבלנו כי הם נמוכים מהתיאורטיים, מה שמצביע על כך שהתשואות בפועל, לאחר התאמה לתנודתיות, עלולות להיות נמוכות יותר בתנאי שוק אמיתיים.
ייתכן שזה נובע מהערכת הסיכון השמרנית של המודל או מגורמים חיצוניים שלא נכללו במודל.

גרף התפלגות מחירי הגז הטבעי של חיזוי סט המבחן לעומת המחירים האמיתיים:



נספחים:

קוד:

ARIMA Model with weather:

```python
import pandas as pd
import numpy as np
from datetime import date, datetime
import yfinance as yf
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from meteostat import Stations, Daily
from geopy.geocoders import Nominatim
from tqdm import tqdm  # For progress bars
import matplotlib.pyplot as plt

1 usage
def get_business_day_data(ticker_symbol, start_date, end_date):
    """
    Fetches historical data for the given ticker symbol from Yahoo Finance,
    ensuring that only business days are considered.
    """
    data = yf.download(ticker_symbol, start=start_date, end=end_date)
    data = data.asfreq('B').dropna()  # Keep only business days
    return data

1 usage
def get_weather_data(states, start_date, end_date):
    """
    Fetches historical weather data for the specified states
    between start_date and end_date using the Meteostat library.
    Aggregates the data by averaging across selected stations in each state.
    """
    # Initialize geolocator
    geolocator = Nominatim(user_agent="natural_gas_weather_app")

    # Define a dictionary to hold state-wise average weather data
    weather_data = pd.DataFrame()

    for state, city in states.items():
        print(f"Fetching weather data for {state} ({city})")
        # Geocode the city to get latitude and longitude
        location = geolocator.geocode(city + ", " + state)
        if not location:
            print(f"Could not geocode city: {city}, {state}")
            continue
        lat, lon = location.latitude, location.longitude

        # Find nearby weather stations
        stations = Stations()
        stations = stations.nearby(lat, lon)
        stations = stations.inventory(freq='daily', required=(datetime.strptime(start_date, '%Y-%m-%d'),
                                      datetime.strptime(end_date, '%Y-%m-%d')))
        station = stations.fetch(1)

        if station.empty:
            print(f"No weather stations found for {city}, {state}")
            continue

        station_id = station.index[0]

        # Fetch daily weather data
        data = Daily(station_id, start=datetime.strptime(start_date, '%Y-%m-%d'),
                     end=datetime.strptime(end_date, '%Y-%m-%d'))
        data = data.fetch()

        if data.empty:
            print(f"No weather data found for station {station_id} in {state}")
            continue

        # Define the required columns
        required_columns = ['temp', 'tmin', 'tmax', 'prcp', 'wspd', 'pres', 'rhum']

        # Check which required columns are present
        present_columns = [col for col in required_columns if col in data.columns]
        if not present_columns:
            print(f"No required weather data available for station {station_id} in {state}")
            continue

        # Select relevant columns
        selected_data = data[present_columns].copy()

        # Rename all present columns by prefixing with the state name to ensure uniqueness
        rename_dict = {col: f'{state}_{col}' for col in present_columns}
        selected_data = selected_data.rename(columns=rename_dict)

        # Handle missing data
        selected_data = selected_data.fillna(method='ffill').fillna(method='bfill')

        # Aggregate by date
        if weather_data.empty:
            weather_data = selected_data.copy()
        else:
            weather_data = weather_data.join(
                selected_data, how='outer'
            )

    # After fetching all states, aggregate the features by taking the mean across states
    # Identify averaged features based on renamed columns
    temp_cols = [col for col in weather_data.columns if col.endswith('_temp') or col.endswith('_tmin') or col.endswith('_tmax')]
    precip_cols = [col for col in weather_data.columns if col.endswith('_prcp')]
    humid_cols = [col for col in weather_data.columns if col.endswith('_rhum')]
    wspd_cols = [col for col in weather_data.columns if col.endswith('_wspd')]
    pres_cols = [col for col in weather_data.columns if col.endswith('_pres')]

    # Calculate average for each feature category if columns exist
    if temp_cols:
        weather_data['Avg_Temp'] = weather_data[temp_cols].mean(axis=1)
    if precip_cols:
        weather_data['Avg_Precipitation'] = weather_data[precip_cols].mean(axis=1)
    if humid_cols:
        weather_data['Avg_Humidity'] = weather_data[humid_cols].mean(axis=1)
    if wspd_cols:
        weather_data['Avg_Wind_Speed'] = weather_data[wspd_cols].mean(axis=1)
    if pres_cols:
        weather_data['Avg_Pressure'] = weather_data[pres_cols].mean(axis=1)

    # Keep only the averaged features that were created
    averaged_features = []
    if temp_cols:
        averaged_features.append('Avg_Temp')
    if precip_cols:
        averaged_features.append('Avg_Precipitation')
    if humid_cols:
        averaged_features.append('Avg_Humidity')
    if wspd_cols:
        averaged_features.append('Avg_Wind_Speed')
    if pres_cols:
        averaged_features.append('Avg_Pressure')

    weather_data = weather_data[averaged_features]

    # Handle any remaining missing data
    weather_data = weather_data.fillna(method='ffill').fillna(method='bfill')
    return weather_data
```

```python
# Define the ticker symbols for the required data
symbols = {
    'Crude_Oil': 'CL=F',
    # 'SP500': '^GSPC',
    'USD_to_Euro': 'EURUSD=X',
    'Natural_Gas': 'NG=F'
}

# Define the date range
start_date = '2013-01-01'
end_date = date.today().strftime("%Y-%m-%d")

# Fetch financial data for each ticker symbol and keep only business days
data_frames = {}
for key, symbol in symbols.items():
    print(f"Downloading data for {key} ({symbol})")
    data_frames[key] = get_business_day_data(symbol, start_date, end_date)

# Combine the relevant 'Close' columns into a single DataFrame
data = pd.DataFrame({
    'Crude_Oil_Close': data_frames['Crude_Oil']['Close'],
    # 'SP500_Close': data_frames['SP500']['Close'],
    'USD_to_Euro_Close': data_frames['USD_to_Euro']['Close'],
    'Natural_Gas_Close': data_frames['Natural_Gas']['Close'],
    'Natural_Gas_Open': data_frames['Natural_Gas']['Open'],
})

# Handle missing data by forward filling and then dropping any remaining NaNs
data = data.ffill().dropna()

# Define the states and their representative cities
states = {
    'Texas': 'Houston',          # Major natural gas production and export hub
    'Louisiana': 'New Orleans',  # Significant natural gas production and trading
    'Oklahoma': 'Oklahoma City',  # Important production area
    'New York': 'New York City',  # Major consumer market
    'Pennsylvania': 'Pittsburgh', # Key region for Marcellus Shale gas production
    'Colorado': 'Denver',         # Important gas production state
    'Alaska': 'Anchorage',        # Major supplier of natural gas
    'California': 'Los Angeles',  # Large consumer market with import facilities
    'Illinois': 'Chicago',        # Major market for natural gas
    'Ohio': 'Cleveland',          # Emerging natural gas production area
    'West Virginia': 'Charleston', # Significant producer in the Appalachian region
    'Kentucky': 'Louisville',     # Natural gas production and consumption
    'Florida': 'Miami',           # Major consumer with import facilities
    'Alabama': 'Birmingham',      # Consumer and transporter
    'Michigan': 'Detroit',        # Significant natural gas consumer
    'Maryland': 'Baltimore',      # Consumption and distribution hub
    'Virginia': 'Richmond',       # Important for gas distribution
    'Tennessee': 'Nashville'      # Consumption and transportation hub
    # Add more states and cities as needed
}


# Fetch weather data
weather_data = get_weather_data(states, start_date, end_date)

# Merge weather data with financial data
# Ensure that the weather_data index is in datetime format and matches the financial data index
weather_data.index = pd.to_datetime(weather_data.index)
data.index = pd.to_datetime(data.index)

# Resample weather data to business days by forward filling
weather_data = weather_data.asfreq('B').fillna(method='ffill')

# Merge on the index (dates)
data = data.join(weather_data, how='left')

# Handle any missing weather data
data = data.fillna(method='ffill').fillna(method='bfill')

# Define features and target
X = data.drop(columns=['Natural_Gas_Close'])
y = data['Natural_Gas_Close']

# Split data into train and test sets
train_size = int(len(data) * 0.9)  # 90% training data
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Standardize the features
scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

# Standardize the target
scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1)).flatten()
y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1)).flatten()

# Fit the ARIMA model on the training set with exogenous variables
arima_order = (4, 1, 0)  # Example ARIMA order, you can optimize this
arima_model = ARIMA(y_train_scaled, order=arima_order, exog=X_train_scaled)
arima_model_fit = arima_model.fit()
# Forecast the values for the test set
forecast_scaled = arima_model_fit.forecast(steps=len(y_test), exog=X_test_scaled)

# Convert forecasts back to original scale
forecast_unscaled = pd.Series(scaler_y.inverse_transform(forecast_scaled.reshape(-1, 1)).flatten(),
                              index=y_test.index)

# Calculate and print performance metrics
mae = mean_absolute_error(y_test, forecast_unscaled)
rmse = np.sqrt(mean_squared_error(y_test, forecast_unscaled))
r2 = r2_score(y_test, forecast_unscaled)

print(f"Mean Absolute Error: {mae}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R^2 Score: {r2}")

# Calculate returns and volatility
true_returns = np.log(y_test / y_test.shift(1)).dropna()
predicted_returns = np.log(forecast_unscaled / forecast_unscaled.shift(1)).dropna()

true_volatility = true_returns.std()
predicted_volatility = predicted_returns.std()

print(f"True Volatility: {true_volatility}")
print(f"Predicted Volatility: {predicted_volatility}")

# Calculate Value at Risk (VaR)
confidence_level = 0.95
VaR_true = np.percentile(true_returns, (1 - confidence_level) * 100)
VaR_predicted = np.percentile(predicted_returns, (1 - confidence_level) * 100)

print(f"True Value at Risk (VaR): {VaR_true}")
print(f"Predicted Value at Risk (VaR): {VaR_predicted}")
```

```python
# Bonus:
# Existing imports...

# Additional imports for trading strategy calculations
from scipy.stats import norm


# Define trading strategy based on forecast
1 usage
def trading_strategy(predicted_prices, actual_prices):
    """
    Implements a simple trading strategy based on the forecasted natural gas prices.
    Buy if the forecast price is higher than the current price, sell otherwise.
    """
    # Generate signals: 1 for buy, -1 for sell, 0 for hold
    signals = np.where(predicted_prices > actual_prices.shift(1), 1, -1)

    # Calculate returns based on signals
    daily_returns = actual_prices.pct_change().shift(-1)  # Shift to align with signals
    strategy_returns = signals * daily_returns  # Multiply signals by returns

    return strategy_returns.dropna()


# Calculate risk-return metrics
2 usages
def calculate_metrics(strategy_returns, risk_free_rate=0.0):
    """
    Calculates performance metrics for the trading strategy.
    """
    mean_return = strategy_returns.mean()  # Average return
    volatility = strategy_returns.std()  # Standard deviation of returns

    # Calculate Sharpe Ratio
    sharpe_ratio = (mean_return - risk_free_rate) / volatility

    # Calculate Sortino Ratio
    downside_returns = strategy_returns[strategy_returns < 0]
    downside_volatility = downside_returns.std()
    sortino_ratio = (mean_return - risk_free_rate) / downside_volatility if downside_volatility != 0 else np.nan

    return sharpe_ratio, sortino_ratio


# Create a DataFrame for actual prices and predicted prices
predicted_prices = forecast_unscaled
actual_prices = y_test

# Generate strategy returns
strategy_returns = trading_strategy(predicted_prices, actual_prices)

# Calculate risk-return metrics
sharpe_ratio, sortino_ratio = calculate_metrics(strategy_returns)

# Print results
print(f"Sharpe Ratio: {sharpe_ratio}")
print(f"Sortino Ratio: {sortino_ratio}")

# Compare empirical values
empirical_returns = y_test.pct_change().dropna()
empirical_sharpe_ratio, empirical_sortino_ratio = calculate_metrics(empirical_returns)

print(f"Empirical Sharpe Ratio: {empirical_sharpe_ratio}")
print(f"Empirical Sortino Ratio: {empirical_sortino_ratio}")

# Graph Actual vs Predicted Natural Gas Prices in test-set

# Plot actual vs predicted prices over time
plt.figure(figsize=(14, 7))
plt.plot( *args: y_test.index, y_test, label='Actual Prices', color='blue')
plt.plot( *args: forecast_unscaled.index, forecast_unscaled, label='Predicted Prices', color='orange', linestyle='--')

# Formatting the x-axis to show months and years
plt.xlabel('Date')
plt.ylabel('Natural Gas Prices')
plt.title('Actual vs Predicted Natural Gas Prices in test-set')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.tight_layout()

# Display the plot
plt.show()
```

:VAR Model

```python
import pandas as pd
import numpy as np
from datetime import date
import yfinance as yf
from statsmodels.tsa.vector_ar.var_model import VAR
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

1 usage  ± hagKiev +1
def get_business_day_data(ticker_symbol, start_date, end_date):
    """
    Fetches historical data for the given ticker symbol from Yahoo Finance,
    ensuring that only business days are considered.
    """
    data = yf.download(ticker_symbol, start=start_date, end=end_date)
    data = data.asfreq('B').dropna()  # Keep only business days
    return data

# Define the ticker symbols for the required data
symbols = {
    'Crude_Oil': 'CL=F',
    'SP500': '^GSPC',
    'USD_to_Euro': 'EURUSD=X',
    'Natural_Gas': 'NG=F'
}

# Define the date range
start_date = '2013-01-01'
end_date = date.today()

# Fetch data for each ticker symbol and keep only business days
data_frames = {}
for key, symbol in symbols.items():
    data_frames[key] = get_business_day_data(symbol, start_date, end_date)

# Combine the relevant 'Close' and 'Open' columns into a single DataFrame
data = pd.DataFrame({
    'Crude_Oil_Close': data_frames['Crude_Oil']['Close'],
    'SP500_Close': data_frames['SP500']['Close'],
    'USD_to_Euro_Close': data_frames['USD_to_Euro']['Close'],
    'Natural_Gas_Close': data_frames['Natural_Gas']['Close'],
    'Crude_Oil_Open': data_frames['Crude_Oil']['Open'],
    'SP500_Open': data_frames['SP500']['Open'],
    'USD_to_Euro_Open': data_frames['USD_to_Euro']['Open'],
    'Natural_Gas_Open': data_frames['Natural_Gas']['Open']
})

# Handle missing data by forward filling and then dropping any remaining NaNs
data = data.ffill().dropna()

# Define the number of lags to use
num_lags = 5  # You can choose a different number of lags based on model performance

# Create lagged features
for i in range(1, num_lags + 1):
    data[f'Crude_Oil_Close_Lag_{i}'] = data['Crude_Oil_Close'].shift(i)
    data[f'SP500_Close_Lag_{i}'] = data['SP500_Close'].shift(i)
    data[f'USD_to_Euro_Close_Lag_{i}'] = data['USD_to_Euro_Close'].shift(i)
    data[f'Crude_Oil_Open_Lag_{i}'] = data['Crude_Oil_Open'].shift(i)
    data[f'SP500_Open_Lag_{i}'] = data['SP500_Open'].shift(i)
    data[f'USD_to_Euro_Open_Lag_{i}'] = data['USD_to_Euro_Open'].shift(i)

# Drop rows with NaN values (which will be created by lagging)
data = data.dropna()

# Separate features and target
X = data.drop(columns=['Natural_Gas_Close'])
y = data['Natural_Gas_Close']

# Split data into train and test sets
train_size = int(len(data) * 0.9)  # 90% training data
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Standardize the features
scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1)).flatten()
y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1)).flatten()

# Combine training data into a single DataFrame for VAR model
train_data = pd.DataFrame(X_train_scaled, columns=X.columns)
train_data['Natural_Gas_Close'] = y_train_scaled

# Fit the VAR model
model = VAR(train_data)
lag_order = model.select_order(maxlags=15)
var_model = model.fit(lag_order.aic)

# Forecast the values for the test set
test_data = pd.DataFrame(X_test_scaled, columns=X.columns)
test_data['Natural_Gas_Close'] = np.nan  # Placeholder for actual values

forecast_scaled = var_model.forecast(train_data.values[-var_model.k_ar:], steps=len(X_test))

# Convert forecasts back to original scale
forecast_unscaled = pd.DataFrame(scaler_y.inverse_transform(forecast_scaled[:, -1].reshape(-1, 1)),
                                 index=y_test.index,
                                 columns=['Natural_Gas'])

# Calculate and print performance metrics
mae = mean_absolute_error(y_test, forecast_unscaled)
rmse = np.sqrt(mean_squared_error(y_test, forecast_unscaled))
r2 = r2_score(y_test, forecast_unscaled)

print(f"Mean Absolute Error: {mae}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R^2 Score: {r2}")

# Calculate returns and volatility
true_returns = np.log(y_test / y_test.shift(1)).dropna()
predicted_returns = np.log(forecast_unscaled['Natural_Gas'] / forecast_unscaled['Natural_Gas'].shift(1)).dropna()

true_volatility = true_returns.std()
predicted_volatility = predicted_returns.std()

print(f"True Volatility: {true_volatility}")
print(f"Predicted Volatility: {predicted_volatility}")

# Calculate Value at Risk (VaR)
confidence_level = 0.95
VaR_true = np.percentile(true_returns, (1 - confidence_level) * 100)
VaR_predicted = np.percentile(predicted_returns, (1 - confidence_level) * 100)

print(f"True Value at Risk (VaR): {VaR_true}")
print(f"Predicted Value at Risk (VaR): {VaR_predicted}")
```

:Linear Regression Model

```python
import pandas as pd
import numpy as np
from datetime import date
import yfinance as yf
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

1 usage   Alon Feldman
def get_business_day_data(ticker_symbol, start_date, end_date):
    """
    Fetches historical data for the given ticker symbol from Yahoo Finance,
    ensuring that only business days are considered.
    """
    data = yf.download(ticker_symbol, start=start_date, end=end_date)
    data = data.asfreq('B').dropna()  # Keep only business days
    return data

# Define the ticker symbols for the required data
symbols = {
    'Crude_Oil': 'CL=F',
    'SP500': '^GSPC',
    'USD_to_Euro': 'EURUSD=X',
    'Natural_Gas': 'NG=F'
}

# Define the date range
start_date = '2013-01-01'
end_date = date.today()

# Fetch data for each ticker symbol and keep only business days
data_frames = {}
for key, symbol in symbols.items():
    data_frames[key] = get_business_day_data(symbol, start_date, end_date)

# Combine the relevant 'Close' columns into a single DataFrame
data = pd.DataFrame({
    'Crude_Oil_Close': data_frames['Crude_Oil']['Close'],
    'SP500_Close': data_frames['SP500']['Close'],
    'USD_to_Euro_Close': data_frames['USD_to_Euro']['Close'],
    'Natural_Gas_Close': data_frames['Natural_Gas']['Close'],
})

# Handle missing data by forward filling and then dropping any remaining NaNs
data = data.ffill().dropna()

# Define features and target
X = data.drop(columns=['Natural_Gas_Close'])
y = data['Natural_Gas_Close']

# Split data into train and test sets
train_size = int(len(data) * 0.9)  # 90% training data
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Standardize the features and target
scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1)).flatten()
y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1)).flatten()

# Fit the Linear Regression model
lin_reg_model = LinearRegression()
lin_reg_model.fit(X_train_scaled, y_train_scaled)

# Predict the test set
predicted_scaled = lin_reg_model.predict(X_test_scaled)

# Convert predictions back to original scale
predicted_unscaled = scaler_y.inverse_transform(predicted_scaled.reshape(-1, 1)).flatten()

# Calculate and print performance metrics
mae = mean_absolute_error(y_test, predicted_unscaled)
rmse = np.sqrt(mean_squared_error(y_test, predicted_unscaled))
r2 = r2_score(y_test, predicted_unscaled)

print(f"Mean Absolute Error: {mae}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R^2 Score: {r2}")

# Calculate returns and volatility
true_returns = np.log(y_test / y_test.shift(1)).dropna()
predicted_returns = np.log(pd.Series(predicted_unscaled, index=y_test.index) / pd.Series(predicted_unscaled, index=y_test.index).shift(1)).dropna()

true_volatility = true_returns.std()
predicted_volatility = predicted_returns.std()

print(f"True Volatility: {true_volatility}")
print(f"Predicted Volatility: {predicted_volatility}")

# Calculate Value at Risk (VaR)
confidence_level = 0.95
VaR_true = np.percentile(true_returns, (1 - confidence_level) * 100)
VaR_predicted = np.percentile(predicted_returns, (1 - confidence_level) * 100)

print(f"True Value at Risk (VaR): {VaR_true}")
print(f"Predicted Value at Risk (VaR): {VaR_predicted}")
```

:ARIMA Model

```python
import pandas as pd
import numpy as np
from datetime import date
import yfinance as yf
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

1 usage  Alon Feldman
def get_business_day_data(ticker_symbol, start_date, end_date):
    """
    Fetches historical data for the given ticker symbol from Yahoo Finance,
    ensuring that only business days are considered.
    """
    data = yf.download(ticker_symbol, start=start_date, end=end_date)
    data = data.asfreq('B').dropna()  # Keep only business days
    return data

# Define the ticker symbols for the required data
symbols = {
    'Crude_Oil': 'CL=F',
    'SP500': '^GSPC',
    'USD_to_Euro': 'EURUSD=X',
    'Natural_Gas': 'NG=F'
}

# Define the date range
start_date = '2013-01-01'
end_date = date.today()

# Fetch data for each ticker symbol and keep only business days
data_frames = {}
for key, symbol in symbols.items():
    data_frames[key] = get_business_day_data(symbol, start_date, end_date)

# Combine the relevant 'Close' columns into a single DataFrame
data = pd.DataFrame({
    'Crude_Oil_Close': data_frames['Crude_Oil']['Close'],
    'SP500_Close': data_frames['SP500']['Close'],
    'USD_to_Euro_Close': data_frames['USD_to_Euro']['Close'],
    'Natural_Gas_Close': data_frames['Natural_Gas']['Close'],
})

# Handle missing data by forward filling and then dropping any remaining NaNs
data = data.ffill().dropna()

# Define features and target (only using close prices for ARIMA)
X = data.drop(columns=['Natural_Gas_Close'])
y = data['Natural_Gas_Close']

# Split data into train and test sets
train_size = int(len(data) * 0.9)  # 90% training data
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Standardize the target
scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1)).flatten()
y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1)).flatten()

# Fit the ARIMA model on the training set
arima_order = (5, 1, 0)  # Example ARIMA order, you can optimize this
arima_model = ARIMA(y_train_scaled, order=arima_order)
arima_model_fit = arima_model.fit()

# Forecast the values for the test set
forecast_scaled = arima_model_fit.forecast(steps=len(y_test))

# Convert forecasts back to original scale
forecast_unscaled = pd.Series(scaler_y.inverse_transform(forecast_scaled.reshape(-1, 1)).flatten(),
                              index=y_test.index)

# Calculate and print performance metrics
mae = mean_absolute_error(y_test, forecast_unscaled)
rmse = np.sqrt(mean_squared_error(y_test, forecast_unscaled))
r2 = r2_score(y_test, forecast_unscaled)

print(f"Mean Absolute Error: {mae}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R^2 Score: {r2}")

# Calculate returns and volatility
true_returns = np.log(y_test / y_test.shift(1)).dropna()
predicted_returns = np.log(forecast_unscaled / forecast_unscaled.shift(1)).dropna()

true_volatility = true_returns.std()
predicted_volatility = predicted_returns.std()

print(f"True Volatility: {true_volatility}")
print(f"Predicted Volatility: {predicted_volatility}")

# Calculate Value at Risk (VaR)
confidence_level = 0.95
VaR_true = np.percentile(true_returns, (1 - confidence_level) * 100)
VaR_predicted = np.percentile(predicted_returns, (1 - confidence_level) * 100)

print(f"True Value at Risk (VaR): {VaR_true}")
print(f"Predicted Value at Risk (VaR): {VaR_predicted}")
```

מאמרים:

1. Hartley, P., Medlock, K., & Rosthal, J. (2007). The relationship between crude oil and natural gas prices. *Rice University, Baker Institute Working Paper*.
2. Zamani, N. (2016). How the crude oil market affects the natural gas market? Demand and supply shocks. *International Journal of Energy Economics and Policy*, *6*(2), 217-221.
3. Saltik, O., Degirmen, S., & Ural, M. (2016). Volatility modelling in crude oil and natural gas prices. *Procedia economics and finance*, *38*, 476-491.
4. Hartley, P. R., & Medlock III, K. B. (2014). The relationship between crude oil and natural gas prices: The role of the exchange rate. *The Energy Journal*, *35*(2), 25-44
5. Szafranek, K., & Rubaszek, M. (2024). Have European natural gas prices decoupled from crude oil prices? Evidence from TVP-VAR analysis. *Studies in Nonlinear Dynamics & Econometrics*, *28*(3), 507-530.
6. Mu, X. (2007). Weather, storage, and natural gas price dynamics: Fundamentals and volatility. *Energy Economics*, *29*(1), 46-63.
7. Erdős, P., & Ormos, M. (2012). Natural gas prices on three continents. Energies, 5(10), 4040-4056
8. Brown, S. P., & Yttcel, M. K. (2008). What drives natural gas prices?. *The Energy Journal*, *29*(2), 45-60
9. Natural Gas Gross Withdrawals and Production - https://www.eia.gov/dnav/ng/ng_prod_sum_a_EPG0_FPD_mmcf_a.htm
10. Natural Gas Consumption by End Use - https://www.eia.gov/dnav/ng/ng_cons_sum_a_EPG0_VC0_mmcf_a.htm