```
# pip install arch
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import cvxpy as cp
from arch import arch_model
```

Skip this if already have the data file

```
# pip install wrds
```

```
import wrds
```

Double-click (or enter) to edit

```
# Connect to WRDS
db = wrds.Connection(wrds_username='simengut')
```

```
Enter your WRDS username [simengut]:simengut
Enter your password:··········
WRDS recommends setting up a .pgpass file.
Create .pgpass file now [y/n]?: y
Created .pgpass file successfully.
You can create this file yourself at any time with the create_pgpass_file() function.
Loading library list...
Done
```

```
renewwable_tickers = ['ICLN', 'PBW', 'QCLN']
nonrenewable_tickers = ['XLE']
benchmark = ["SPY"]

# SQL query to fetch the data
sql_query = """
SELECT
    a.permno,
    a.date,
    b.ticker,
    b.comnam,
    b.cusip,
    a.prc,
    a.vol,
    a.ret
FROM
    crsp.dsf AS a
JOIN
    crsp.dsenames AS b
ON
    a.permno = b.permno
WHERE
    b.ticker IN ('ICLN', 'PBW', 'QCLN', 'XLE', 'SPY')
    AND a.date BETWEEN '2007-01-01' AND '2024-12-31'
    AND a.date >= b.namedt
    AND a.date <= b.nameendt
ORDER BY
    b.ticker,
    a.date;
"""

# Execute query and get results
df = db.raw_sql(sql_query)

# Display the first few rows
df.head()
```

	permno	date	ticker	comnam	cusip	prc	vol	ret
0	92720	2008-06-25	ICLN	ISHARES TRUST	46428822	52.77	1935.0	<NA>
1	92720	2008-06-26	ICLN	ISHARES TRUST	46428822	51.06	8722.0	-0.032405
2	92720	2008-06-27	ICLN	ISHARES TRUST	46428822	50.157	7588.0	-0.017685
3	92720	2008-06-30	ICLN	ISHARES TRUST	46428822	50.25	16765.0	0.001854
4	92720	2008-07-01	ICLN	ISHARES TRUST	46428822	48.797	14393.0	-0.028915

```
# Pivot the DataFrame
returns_df = df.pivot(index='date', columns='ticker', values='ret')

# Sort the index (dates)
returns_df = returns_df.sort_index()

# Display the first few rows
print("Returns DataFrame (dates as index, tickers as columns):")
returns_df.head()
```

```
Returns DataFrame (dates as index, tickers as columns):
```

ticker	ICLN	PBW	QCLN	SPY	XLE
date					
2007-01-03	<NA>	-0.005774	<NA>	-0.001765	-0.034965
2007-01-04	<NA>	0.009292	<NA>	0.002122	-0.018204
2007-01-05	<NA>	-0.023015	<NA>	-0.007976	0.006661
2007-01-08	<NA>	0.006478	<NA>	0.004625	0.0
2007-01-09	<NA>	-0.005266	<NA>	-0.00085	-0.008584

```
# Find the latest start date among all ETFs
latest_start_date = df.groupby('ticker')['date'].min().max()
print(f"Latest start date among all ETFs: {latest_start_date}")

# Filter the data to start from the latest start date
df_filtered = df[df['date'] >= latest_start_date]

# Pivot the filtered data
returns_df = df_filtered.pivot(index='date', columns='ticker', values='ret')

# Sort the index (dates)
returns_df = returns_df.sort_index()

# Display the first few rows
print("\nReturns DataFrame (starting from latest common date):")
returns_df.head()
```

```
Latest start date among all ETFs: 2008-06-25

Returns DataFrame (starting from latest common date):
```

ticker	ICLN	PBW	QCLN	SPY	XLE
date					
2008-06-25	<NA>	0.008435	0.004221	0.004726	-0.003774
2008-06-26	-0.032405	-0.047398	-0.037065	-0.02716	-0.007231
2008-06-27	-0.017685	-0.015122	-0.012302	-0.005459	0.008787
2008-06-30	0.001854	-0.031699	-0.006669	0.003529	0.014098
2008-07-01	-0.028915	-0.014322	-0.021599	0.003126	0.004182

```
# check for null values
returns_df.isnull().sum()
returns_df.dropna(inplace=True)
returns_df.head()
returns_df.isnull().sum()
returns_df.dropna(inplace=True)
returns_df
```

ticker	ICLN	PBW	QCLN	SPY	XLE
date					
2008-06-26	-0.032405	-0.047398	-0.037065	-0.02716	-0.007231
2008-06-27	-0.017685	-0.015122	-0.012302	-0.005459	0.008787
2008-06-30	0.001854	-0.031699	-0.006669	0.003529	0.014098
2008-07-01	-0.028915	-0.014322	-0.021599	0.003126	0.004182
2008-07-02	-0.026375	-0.05397	-0.04382	-0.017137	-0.030163
...	...	...	...	...	...
2024-12-24	0.003448	0.026706	0.017939	0.011115	0.008459
2024-12-26	-0.003436	0.026012	-0.001399	0.000067	-0.000827
2024-12-27	-0.006035	-0.016901	-0.015966	-0.010527	-0.000118
2024-12-30	-0.008673	-0.021012	-0.018787	-0.011412	-0.000118
2024-12-31	-0.004374	-0.023902	-0.015666	-0.003638	0.013128

4157 rows × 5 columns

```
# Create separate DataFrames for renewable and non-renewable ETFs
nonrenewable_df = returns_df[['XLE', 'SPY']].copy()
renewable_df = returns_df.drop('XLE', axis=1).copy()
renewable_df = renewable_df.drop('SPY', axis=1).copy()

# Save both DataFrames to separate CSV files
nonrenewable_df.to_csv('nonrenewable_etf_returns.csv')
renewable_df.to_csv('renewable_etfs_returns.csv')

print("Non-renewable ETF (XLE) data:")
print(nonrenewable_df.head())
print("\nRenewable ETFs data:")
print(renewable_df.head())
```

```
Non-renewable ETF (XLE) data:
ticker           XLE       SPY
date
2008-06-26 -0.007231  -0.02716
2008-06-27  0.008787 -0.005459
2008-06-30  0.014098  0.003529
2008-07-01  0.004182  0.003126
2008-07-02 -0.030163 -0.017137

Renewable ETFs data:
ticker          ICLN       PBW      QCLN
date
2008-06-26 -0.032405 -0.047398 -0.037065
2008-06-27 -0.017685 -0.015122 -0.012302
2008-06-30  0.001854 -0.031699 -0.006669
2008-07-01 -0.028915 -0.014322 -0.021599
2008-07-02 -0.026375  -0.05397  -0.04382
```

```
# write the returns code to a file
nonrenewable_df.to_csv('data/nonrenewable_etfs_returns.csv')
renewable_df.to_csv('data/renewable_etfs_returns.csv')
```

Retrieve data from csv files

```
from google.colab import drive
drive.mount('/content/drive/')
```

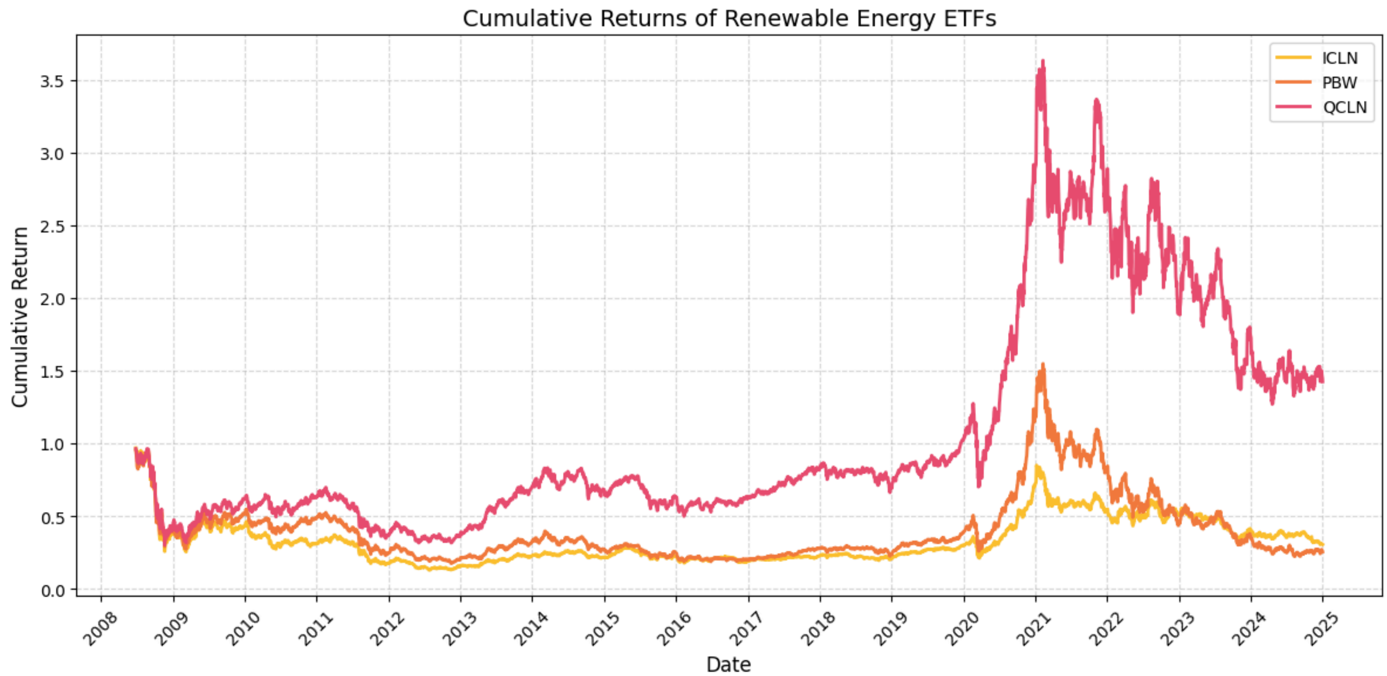
```
Mounted at /content/drive/
```

```
# simen location
# cd drive/MyDrive/Math583/FinalProject
```

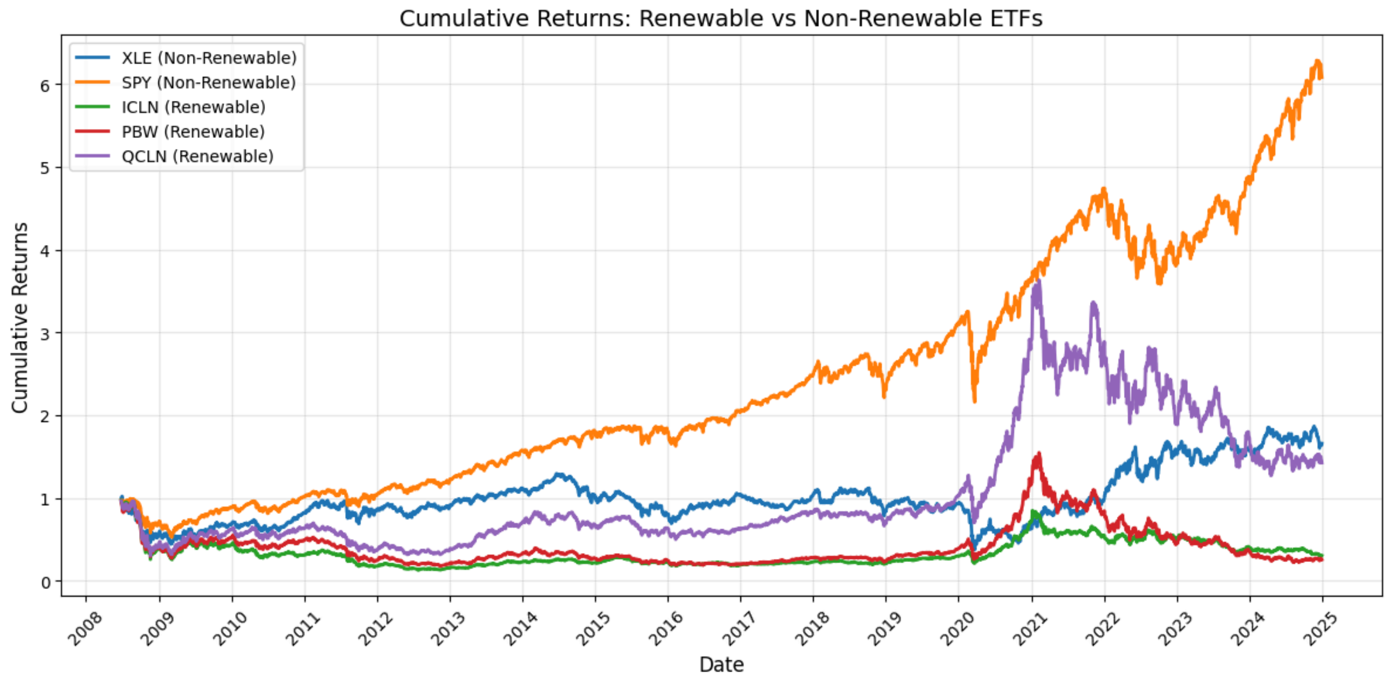
```
# nonrenewable_df = pd.read_csv('data/nonrenewable_etfs_returns.csv', index_col=0, parse_dates=True)
# renewable_df = pd.read_csv('data/renewable_etfs_returns.csv', index_col=0, parse_dates=True)
returns_df = pd.read_csv('daily_returns_df.csv', index_col=0, parse_dates=True)


renewable_df = returns_df[['ICLN', 'PBW', 'QCLN']]
nonrenewable_df = returns_df[['XLE', 'SPY']]
```

Exploratory analysis

```
import matplotlib.dates as mdates

# Compute cumulative returns
cum_returns_r = (1 + renewable_df).cumprod()
cum_returns_nr = (1 + nonrenewable_df).cumprod()


dates_nr = pd.to_datetime(cum_returns_nr.index)
dates_r = pd.to_datetime(cum_returns_r.index)

# Plot
plt.figure(figsize=(12, 6))

# Set custom colors
colors = {
    "ICLN": "#FDBF2D",  # golden
    "PBW": "#F47C3C",   # orange
    "QCLN": "#E94B6E",  # red-pink
}

for col in cum_returns_r.columns:
    plt.plot(cum_returns_r.index, cum_returns_r[col], label=col, color=colors.get(col, None), linewidth=2)

# Styling
plt.title("Cumulative Returns of Renewable Energy ETFs", fontsize=14)
plt.ylabel("Cumulative Return", fontsize=12)
plt.xlabel("Date", fontsize=12)
plt.grid(True, which='major', linestyle='--', alpha=0.5)

# Set x-ticks to show only every 100th date
ax = plt.gca()
ax.xaxis.set_major_locator(mdates.YearLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.xticks(rotation=45)


plt.legend()
plt.tight_layout()
plt.show()
```




```
plt.figure(figsize=(12, 6))

# Convert indices to datetime if they're not already
dates_nr = pd.to_datetime(cum_returns_nr.index)
dates_r = pd.to_datetime(cum_returns_r.index)

# Plot both cumulative returns series
plt.plot(dates_nr, cum_returns_nr['XLE'], label='XLE (Non-Renewable)', linewidth=2)
plt.plot(dates_nr, cum_returns_nr['SPY'], label='SPY (Non-Renewable)', linewidth=2)
plt.plot(dates_r, cum_returns_r['ICLN'], label='ICLN (Renewable)', linewidth=2)
plt.plot(dates_r, cum_returns_r['PBW'], label='PBW (Renewable)', linewidth=2)
plt.plot(dates_r, cum_returns_r['QCLN'], label='QCLN (Renewable)', linewidth=2)

# Customize the plot
plt.title('Cumulative Returns: Renewable vs Non-Renewable ETFs', fontsize=14)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Cumulative Returns', fontsize=12)
plt.grid(True, alpha=0.3)

# Set x-ticks to show only every 100th date
ax = plt.gca()
ax.xaxis.set_major_locator(mdates.YearLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.xticks(rotation=45)

# Add legend
plt.legend(fontsize=10)

plt.tight_layout()
plt.show()
```

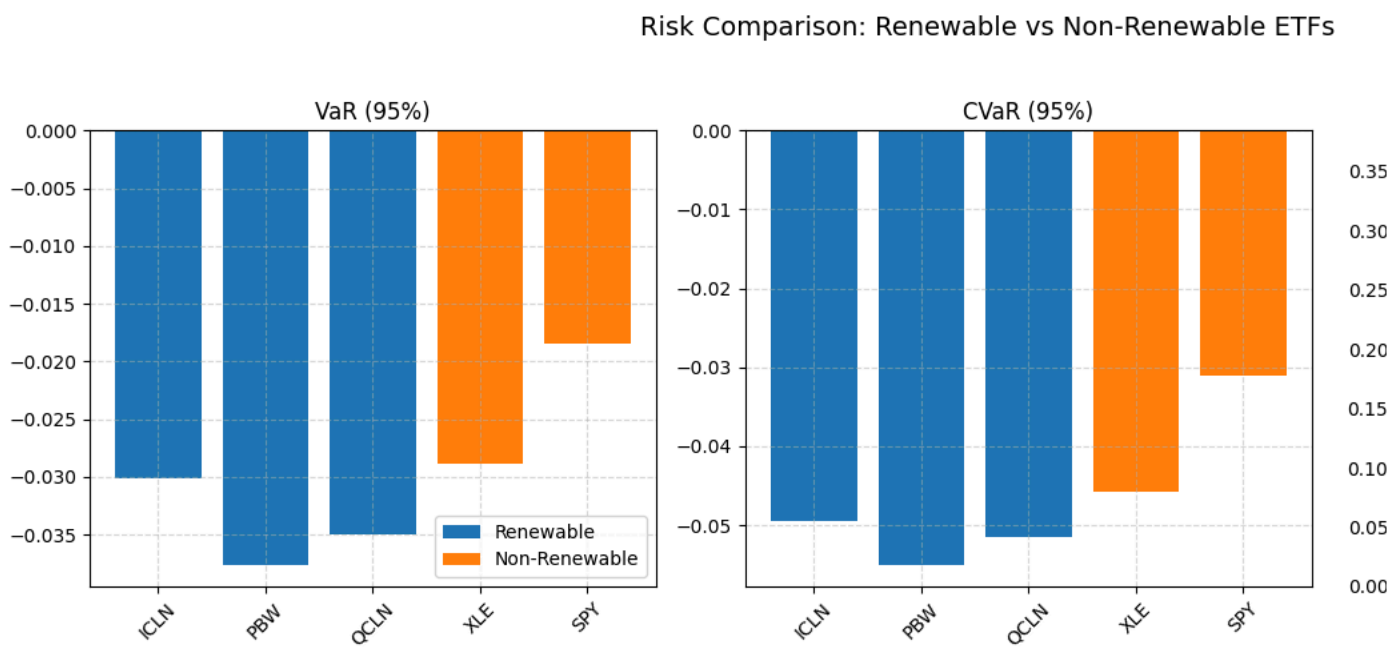


Compute VaR, CVaR and Max Drawdown for ETFs

```
import warnings
warnings.filterwarnings("ignore")

# 1. Compute risk metrics
def compute_risk_metrics(df):
    metrics = {}
    for col in df.columns:
        returns = df[col].dropna()
        var_95 = returns.quantile(0.05)
        cvar_95 = returns[returns <= var_95].mean()
        mdd = (returns.cummax() - returns).max()
        metrics[col] = {
            'VaR (95%)': round(var_95, 4),
            'CVaR (95%)': round(cvar_95, 4),
            'Max Drawdown': round(mdd, 4)
        }
    return pd.DataFrame(metrics).T

# 2. Generate risk metric tables
renewable_risk = compute_risk_metrics(renewable_df)
nonrenewable_risk = compute_risk_metrics(nonrenewable_df)

print(renewable_risk)
print(nonrenewable_risk)

# 3. Combine with group labels
combined_risk = pd.concat(
    [renewable_risk, nonrenewable_risk],
    keys=["Renewable", "Non-Renewable"]
)
combined_risk.index.names = ["Group", "ETF"]

# 4. Melt into tidy format for plotting
melted = combined_risk.reset_index().melt(
    id_vars=["Group", "ETF"],
    var_name="Metric",
    value_name="Value"
)

# 5. Plot
```

```
plt.figure(figsize=(15, 5))

metrics = melted["Metric"].unique()
for i, metric in enumerate(metrics):
    ax = plt.subplot(1, 3, i + 1)
    subset = melted[melted["Metric"] == metric]

    # Create ordered list of ETFs
    etf_order = ['ICLN', 'PBW', 'QCLN', 'XLE', 'SPY']

    # Filter and sort the data according to the ETF order
    subset = subset[subset['ETF'].isin(etf_order)]
    subset['ETF'] = pd.Categorical(subset['ETF'], categories=etf_order, ordered=True)
    subset = subset.sort_values('ETF')

    for group in subset["Group"].unique():
        group_data = subset[subset["Group"] == group]
        ax.bar(group_data["ETF"], group_data["Value"], label=group)

    ax.set_title(metric)
    ax.set_xticklabels(etf_order, rotation=45)
    ax.grid(True, linestyle="--", alpha=0.5)
    if i == 0:
        ax.legend()

plt.suptitle("Risk Comparison: Renewable vs Non-Renewable ETFs", fontsize=14)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

```
        VaR (95%)  CVaR (95%)  Max Drawdown
ICLN    -0.0301     -0.0495        0.3275
PBW     -0.0376     -0.0550        0.3162
QCLN    -0.0350     -0.0515        0.3040
      VaR (95%)  CVaR (95%)  Max Drawdown
XLE    -0.0288     -0.0458        0.3662
SPY    -0.0185     -0.0310        0.2546
```

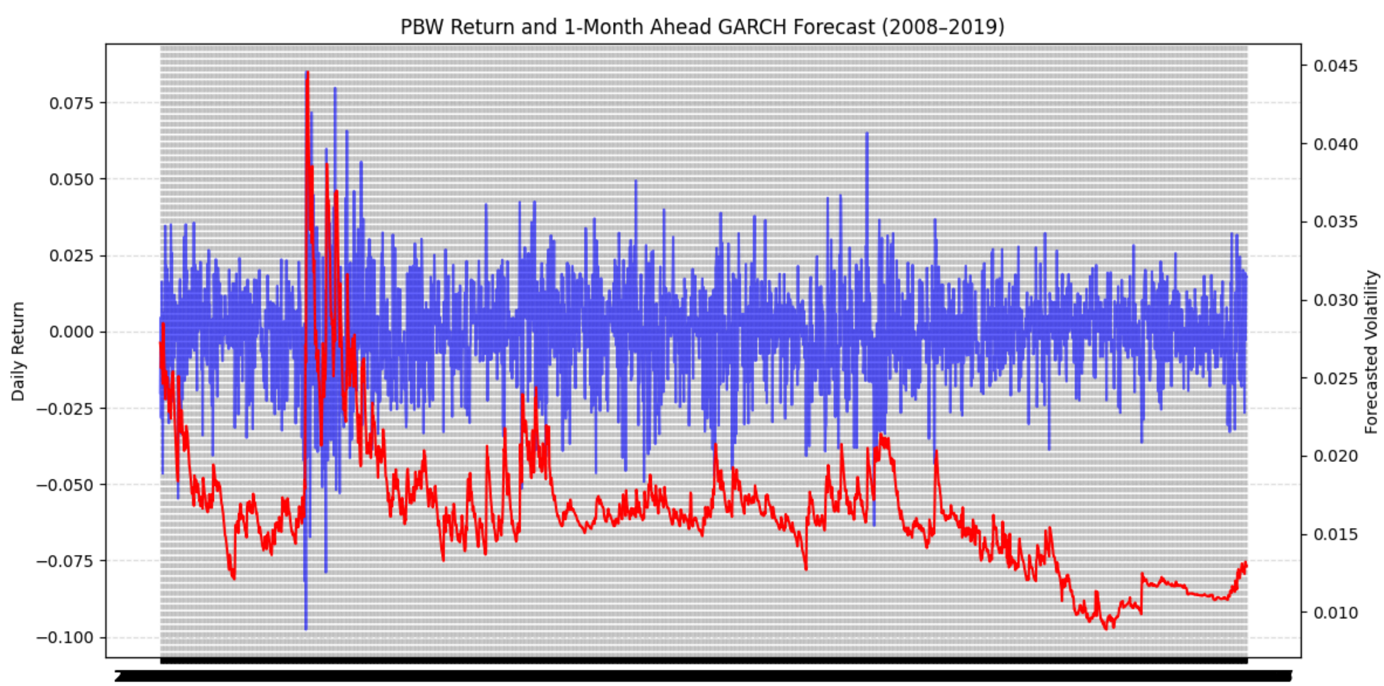


GARCH Volatility testing to get volatility signals

make training and test set to later test performance of volatility signals

```
# Step 1: Limit data to 2008–2019 (development phase)
subset_df = renewable_df.loc["2008":"2019"]
ticker = "PBW"
returns = subset_df[ticker].dropna() * 100  # GARCH expects percentage returns

# Step 2: Fit GARCH on full available history up to each point and forecast 1 month ahead
from arch import arch_model
```

```
import matplotlib.pyplot as plt
import numpy as np

# Define rolling forecast window and forecast horizon
rolling_window = 500  # use 500 days (~2 years) to fit GARCH
forecast_horizon = 22  # forecast 22 trading days (1 month)

# Store results
forecast_vols = []

# Only forecast when we have enough data
for i in range(rolling_window, len(returns) - forecast_horizon):
    window_data = returns.iloc[i - rolling_window:i]
    model = arch_model(window_data, vol='Garch', p=1, q=1)
    try:
        res = model.fit(disp="off")
        forecast = res.forecast(horizon=forecast_horizon)
        # Take average volatility over the 22-day forecast period
        avg_forecast_vol = np.mean(np.sqrt(forecast.variance.values[-1]) / 100)
        forecast_vols.append(avg_forecast_vol)
    except:
        forecast_vols.append(np.nan)

# Create time-aligned Series of forecasted volatilities
valid_index = returns.index[rolling_window:-forecast_horizon]
garch_vol_forecast_1M = pd.Series(forecast_vols, index=valid_index, name="1M GARCH Forecast Vol")

# Combine with actual returns
aligned_returns = returns.loc[valid_index] / 100  # convert back to decimal

# Plot
fig, ax1 = plt.subplots(figsize=(12, 6))
ax1.plot(aligned_returns.index, aligned_returns, label=f"{ticker} Return", color="blue", alpha=0.6)
ax2 = ax1.twinx()
ax2.plot(garch_vol_forecast_1M.index, garch_vol_forecast_1M, label="1M Forecasted Vol", color="red")
ax1.set_title(f"{ticker} Return and 1-Month Ahead GARCH Forecast (2008–2019)")
ax1.set_ylabel("Daily Return")
ax2.set_ylabel("Forecasted Volatility")
ax1.grid(True, linestyle="--", alpha=0.4)
plt.tight_layout()
plt.show()
```



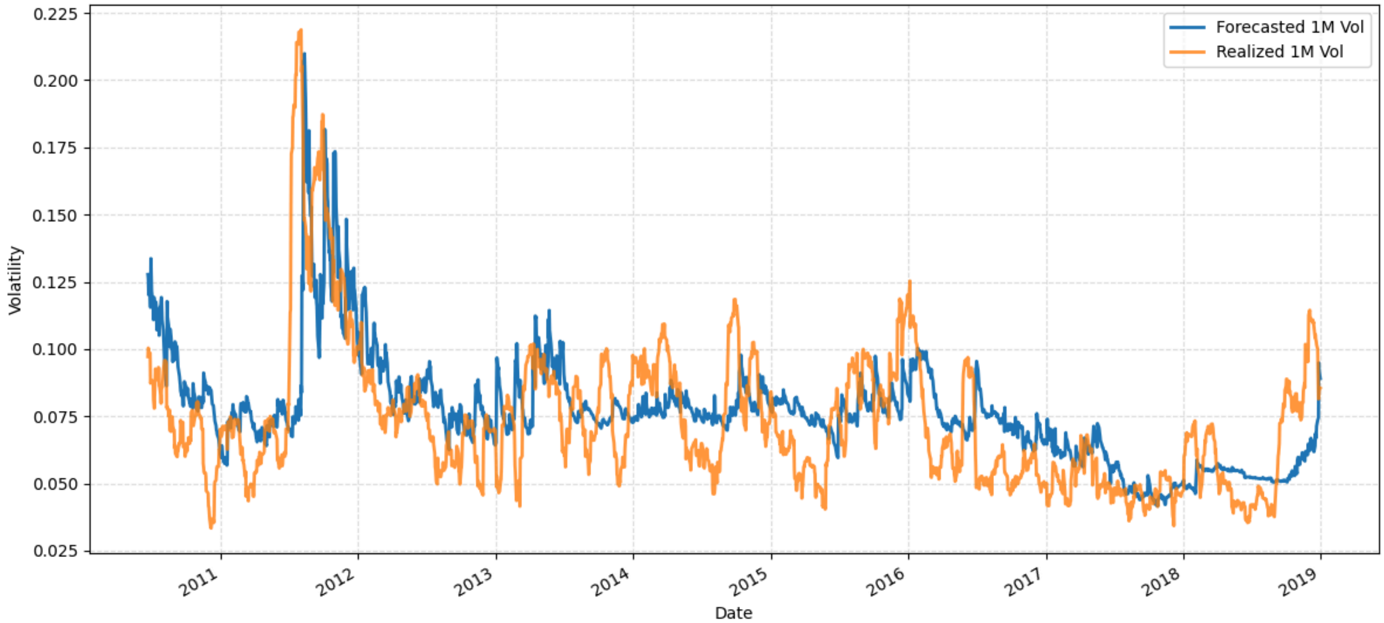
```python
def compute_forecast_and_realized_vol(
    returns_df: pd.DataFrame,
    ticker: str,
    start: str,
    end: str,
    rolling_window: int = 500,
    forecast_horizon: int = 22
) -> (pd.Series, pd.Series):
    """
    Compute 1-month ahead GARCH(1,1) forecasted volatility and realized volatility.

    Parameters:
    - returns_df: DataFrame of decimal returns, index is datetime, columns are tickers
    - ticker: the column name to analyze
    - start, end: date strings like '2008-01-01'
    - rolling_window: number of days for in-sample GARCH fit
    - forecast_horizon: days ahead to forecast & realized window (e.g. 22)

    Returns:
    - forecast_vol: Series of forecasted vol, indexed by forecast date
    - realized_vol: Series of realized vol, same index
    """
    # Ensure datetime index
    df = returns_df.copy()
    df.index = pd.to_datetime(df.index)

    # Subset and drop missing
    series = df[ticker].loc[start:end].dropna()

    # Convert to percent scale for GARCH
    pct_returns = series * 100

    forecast_dates = []
    forecast_vols = []
    # Rolling GARCH forecasts
    for i in range(rolling_window, len(pct_returns) - forecast_horizon):
        window = pct_returns.iloc[i - rolling_window : i]
        model = arch_model(window, vol='Garch', p=1, q=1)
        res = model.fit(disp='off')
        fc = res.forecast(horizon=forecast_horizon, reindex=False)
        # Sum daily variances (pct^2) and convert to decimal var
        var_fore = fc.variance.values[-1]
        monthly_var_dec = var_fore.sum() / 10000
        forecast_vols.append(np.sqrt(monthly_var_dec))
        forecast_dates.append(pct_returns.index[i])

    forecast_vol = pd.Series(forecast_vols, index=forecast_dates, name='Forecasted 1M Vol')

    # Compute realized vol by integer slicing
    realized_vols = []
    for dt in forecast_dates:
        pos = series.index.get_loc(dt)
        window = series.iloc[pos+1 : pos+1+forecast_horizon]
        realized_var = (window ** 2).sum()
        realized_vols.append(np.sqrt(realized_var))

    realized_vol = pd.Series(realized_vols, index=forecast_dates, name='Realized 1M Vol')
    return forecast_vol, realized_vol



def plot_forecast_vs_realized_vol(
    forecast_vol: pd.Series,
    realized_vol: pd.Series,
    ticker: str,
    start_date: str,
    end_date: str,
    figsize=(12, 6),
    linewidth=2,
    alpha=0.8):
    """
    Plot forecasted vs realized 1-month vol for a specific date range.

    Parameters:
    -----------
    forecast_vol : pd.Series
```

```
        Series of forecasted volatilities
    realized_vol : pd.Series
        Series of realized volatilities
    ticker : str
        ETF ticker symbol
    start_date : str
        Start date for the plot (format: 'YYYY-MM-DD')
    end_date : str
        End date for the plot (format: 'YYYY-MM-DD')
    figsize : tuple
        Figure size (width, height)
    linewidth : float
        Width of the lines
    alpha : float
        Transparency of the realized volatility line
    """
    # Convert dates to datetime if they aren't already
    start_date = pd.to_datetime(start_date)
    end_date = pd.to_datetime(end_date)

    # Filter data for the specified date range
    forecast_vol = forecast_vol.loc[start_date:end_date]
    realized_vol = realized_vol.loc[start_date:end_date]

    # Create the plot
    plt.figure(figsize=figsize)
    # Plot the lines
    plt.plot(forecast_vol.index, forecast_vol,
             label='Forecasted 1M Vol',
             linewidth=linewidth)
    plt.plot(realized_vol.index, realized_vol,
             label='Realized 1M Vol',
             linewidth=linewidth,
             alpha=alpha)

    # Add title and labels
    plt.title(f'{ticker}: Forecasted vs Realized 1-Month Volatility\n{start_date.strftime("%Y-%m-%d")} to {end_date.strftime("%Y
    plt.xlabel('Date')
    plt.ylabel('Volatility')
    # Add legend and grid
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.4)
    # Format x-axis dates
    plt.gcf().autofmt_xdate()
    plt.tight_layout()
    plt.show()
```

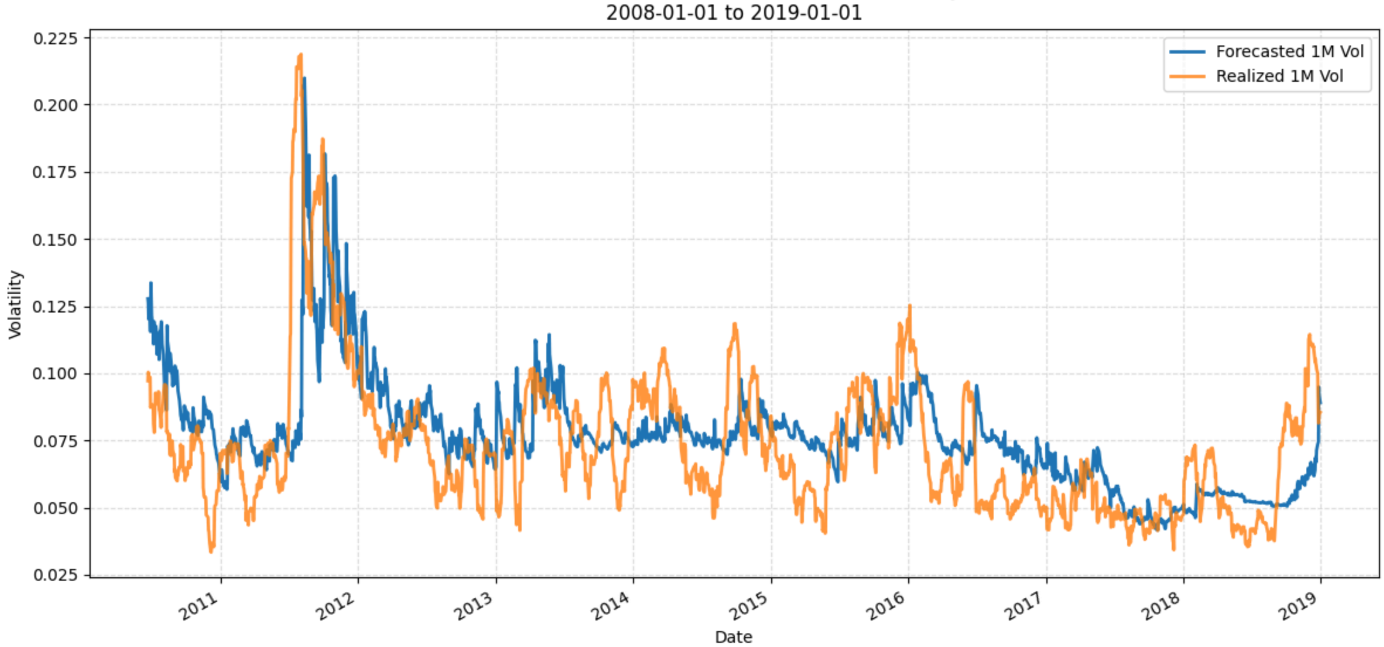
```
forecast_ICLN, realized_ICLN = compute_forecast_and_realized_vol(renewable_df, 'ICLN', '2008-01-01', '2019-12-31')
forecast_PBW, realized_PBW = compute_forecast_and_realized_vol(renewable_df, 'PBW', '2008-01-01', '2019-12-31')
forecast_QCLN, realized_QCLN = compute_forecast_and_realized_vol(renewable_df, 'QCLN', '2008-01-01', '2019-12-31')
```

```
plot_forecast_vs_realized_vol(forecast_PBW, realized_PBW, 'ICLN', '2008', '2019')
plot_forecast_vs_realized_vol(forecast_PBW, realized_PBW, 'PBW', '2008', '2019')
plot_forecast_vs_realized_vol(forecast_PBW, realized_PBW, 'QCLN', '2008', '2019')
```

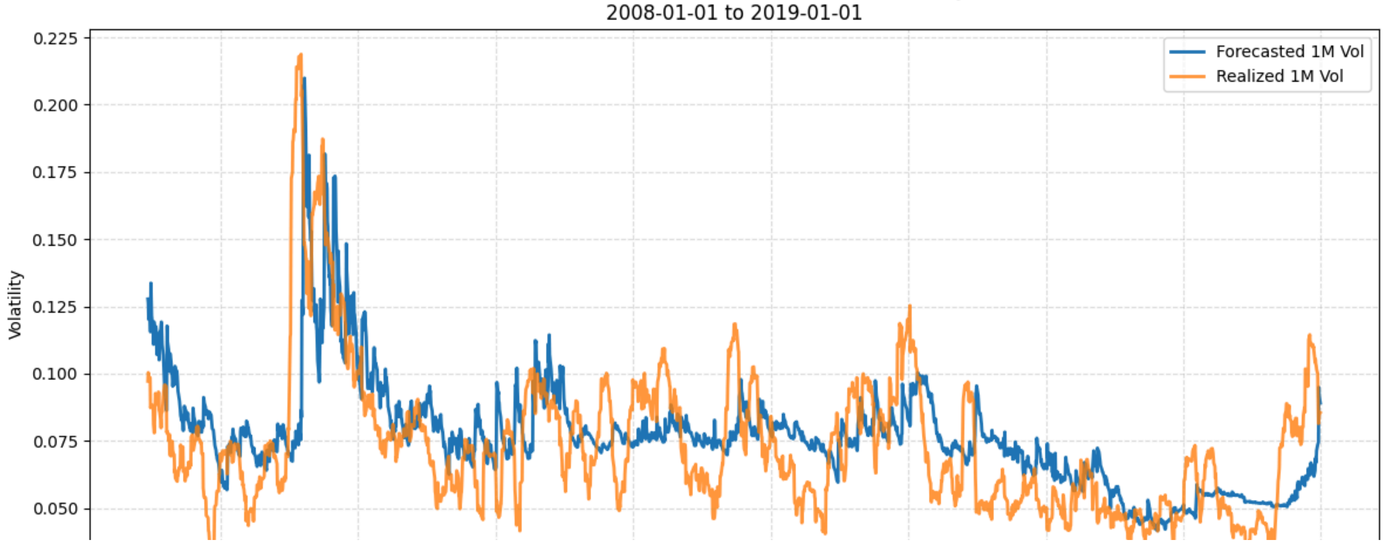
ICLN: Forecasted vs Realized 1-Month Volatility
2008-01-01 to 2019-01-01

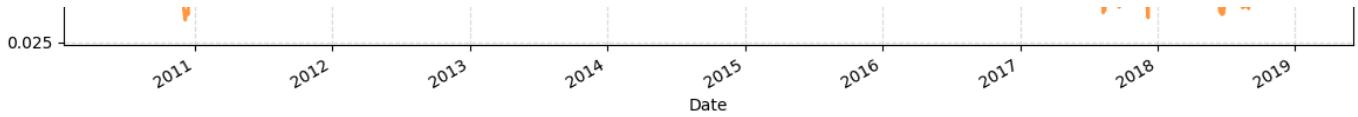


PBW: Forecasted vs Realized 1-Month Volatility
2008-01-01 to 2019-01-01



QCLN: Forecasted vs Realized 1-Month Volatility
2008-01-01 to 2019-01-01





GARCH Hedging

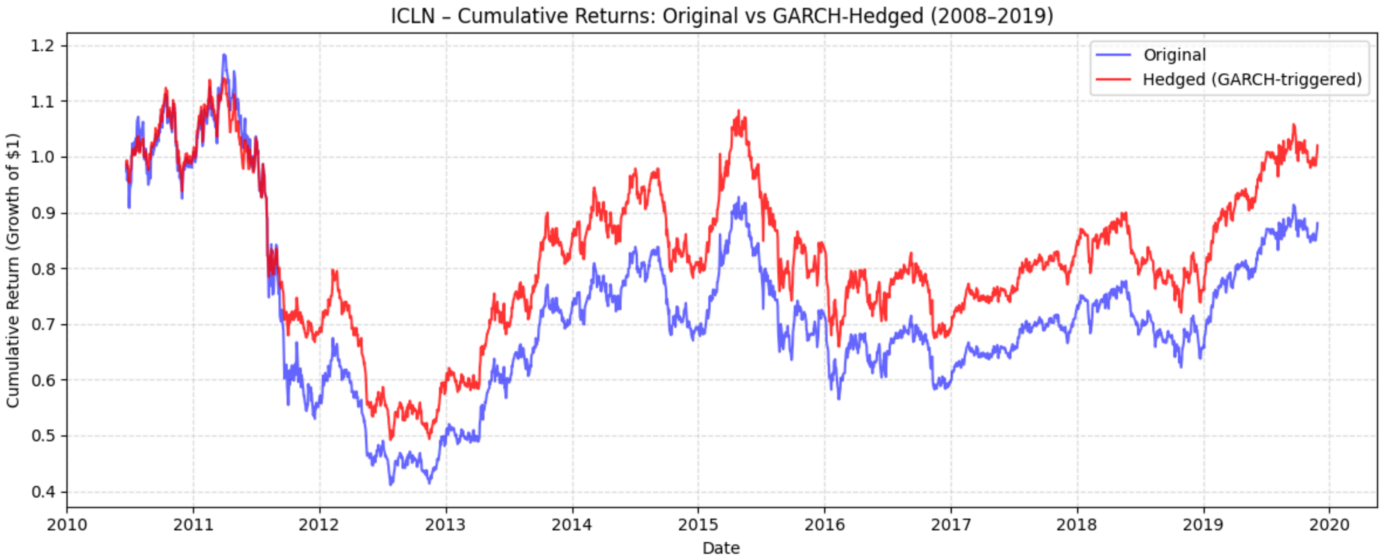
```
import pandas as pd
import numpy as np
from arch import arch_model
import matplotlib.pyplot as plt

# Parameters
rolling_window = 500
forecast_horizon = 22
de_risk_pct = 0.5
percentile_threshold = 0.9

# Restrict data to 2008–2019
renewable_df.index = pd.to_datetime(renewable_df.index)
renewable_subset = renewable_df.loc["2008":"2019"]

hedged_returns = {}
hedge_log = {}  # Store decision logs
hedge_trigger_dates_dict = {}  # NEW: stores dates with hedge activation

for col in renewable_subset.columns:
    print(f"Processing {col}...")
    returns = renewable_subset[col].dropna() * 100
    forecast_vols = []
    decisions = []
    hedge_dates = []

    for i in range(rolling_window, len(returns) - forecast_horizon):
        window = returns.iloc[i - rolling_window:i]
        model = arch_model(window, vol='Garch', p=1, q=1)
        try:
            res = model.fit(disp="off")
            forecast = res.forecast(horizon=forecast_horizon)
            avg_vol = np.mean(np.sqrt(forecast.variance.values[-1]) / 100)
            forecast_vols.append(avg_vol)
        except:
            forecast_vols.append(np.nan)

    valid_index = returns.index[rolling_window:-forecast_horizon]
    vol_signal = pd.Series(forecast_vols, index=valid_index)
    threshold = vol_signal.quantile(percentile_threshold)

    raw_returns = returns.loc[valid_index] / 100
    hedged = raw_returns.copy()

    for date in valid_index:
        vol = vol_signal.loc[date]
        if vol > threshold:
            decision = f"{date.date()}: De-risked (Vol={vol:.4f} > {threshold:.4f})"
            hedged.loc[date] *= (1 - de_risk_pct)
            hedge_dates.append(date)  # store hedge activation date
        else:
            decision = f"{date.date()}: Full exposure (Vol={vol:.4f} <= {threshold:.4f})"
        decisions.append(decision)

    hedged_returns[col] = hedged
```

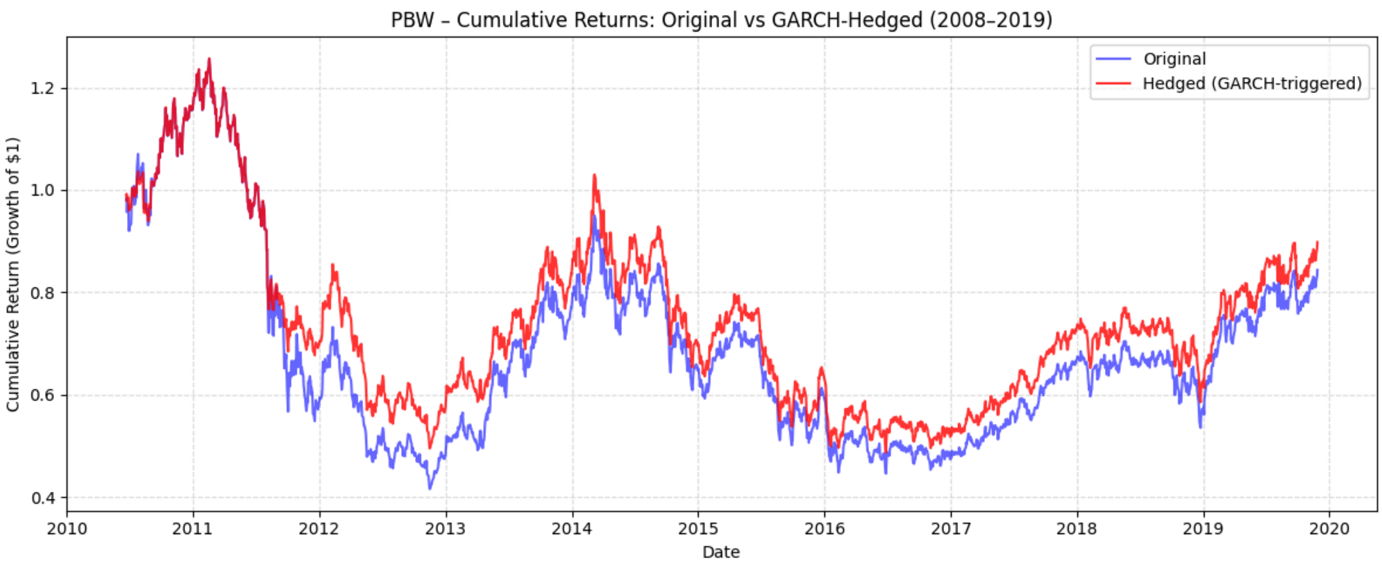
```
    hedge_log[col] = decisions
    hedge_trigger_dates_dict[col] = hedge_dates  # save hedge dates per ETF

# Save returns
hedged_renewable_df = pd.DataFrame(hedged_returns)
hedged_renewable_df.to_csv("renewable_hedged_returns.csv")

# Save logs
with open("hedge_decision_log.txt", "w") as f:
    for col, logs in hedge_log.items():
        f.write(f"\n=== {col} Hedge Decision Log ===\n")
        for entry in logs:
            f.write(entry + "\n")

# Save hedge trigger dates
with open("hedge_trigger_dates.txt", "w") as f:
    for col, dates in hedge_trigger_dates_dict.items():
        f.write(f"\n=== {col} Hedge Trigger Dates ===\n")
        for d in dates:
            f.write(str(d.date()) + "\n")

print("✅ Saved hedged returns, decision logs, and hedge trigger dates.")
```

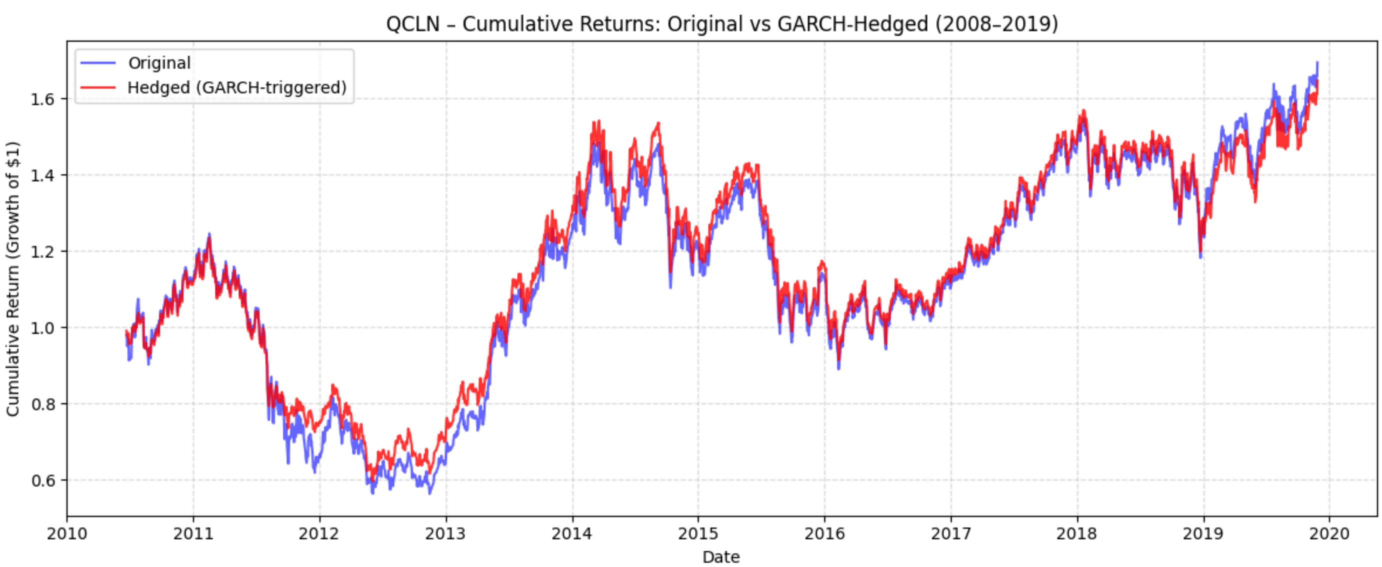
```
Processing ICLN...
Processing PBW...
Processing QCLN...
✅ Saved hedged returns, decision logs, and hedge trigger dates.
```

Start coding or generate with AI.

```
import pandas as pd
import matplotlib.pyplot as plt

# Load data
renewable_df = pd.read_csv("renewable_etfs_returns.csv", parse_dates=["date"], index_col="date")
hedged_renewable_df = pd.read_csv("renewable_hedged_returns.csv", parse_dates=["date"], index_col="date")

# Align date range
renewable_df = renewable_df.loc[hedged_renewable_df.index]

# Plot cumulative returns for each ETF
for col in hedged_renewable_df.columns:
    cum_orig = (1 + renewable_df[col]).cumprod()
    cum_hedged = (1 + hedged_renewable_df[col]).cumprod()

    plt.figure(figsize=(12, 5))
    plt.plot(cum_orig.index, cum_orig, label="Original", color="blue", alpha=0.6)
    plt.plot(cum_hedged.index, cum_hedged, label="Hedged (GARCH-triggered)", color="red", alpha=0.8)
    plt.title(f"{col} — Cumulative Returns: Original vs GARCH-Hedged (2008–2019)")
    plt.xlabel("Date")
    plt.ylabel("Cumulative Return (Growth of $1)")
    plt.grid(True, linestyle="--", alpha=0.4)
    plt.legend()
    plt.tight_layout()
    plt.show()
```


ICLN – Cumulative Returns: Original vs GARCH-Hedged (2008–2019)
Original
Hedged (GARCH-triggered)
Cumulative Return (Growth of $1)
Date
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020


PBW – Cumulative Returns: Original vs GARCH-Hedged (2008–2019)
Original
Hedged (GARCH-triggered)
Cumulative Return (Growth of $1)
Date
0.4
0.6
0.8
1.0
1.2
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020


QCLN – Cumulative Returns: Original vs GARCH-Hedged (2008–2019)
Original
Hedged (GARCH-triggered)
Cumulative Return (Growth of $1)
Date
0.6
0.8
1.0
1.2
1.4
1.6
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020

```
def run_garch_hedging_strategy(renewable_df,
                               etf_name,
                               rolling_window=500,
                               forecast_horizon=22,
                               de_risk_pct=0.5,
                               percentile_threshold=0.9,
                               start_date="2008",
                               end_date="2019"
                               ):
    """
    Run GARCH-based hedging strategy for a single ETF.

    Parameters:
    -----------
    renewable_df : pandas.DataFrame
        DataFrame containing ETF returns
    etf_name : str
        Name of the ETF to analyze
    rolling_window : int
        Window size for GARCH estimation
    forecast_horizon : int
        Number of days to forecast volatility
    de_risk_pct : float
        Percentage to de-risk when volatility is high
    percentile_threshold : float
        Percentile threshold for volatility (0-1)
    start_date : str
        Start date for analysis
    end_date : str
        End date for analysis
    Returns:
    --------
    dict
        Dictionary containing results and metrics
    """

    # Ensure datetime index
    renewable_df.index = pd.to_datetime(renewable_df.index)
    renewable_subset = renewable_df.loc[start_date:end_date]

    print(f"Processing {etf_name}...")
    returns = renewable_subset[etf_name].dropna() * 100
    forecast_vols = []
    decisions = []
    hedge_trigger_dates = pd.DataFrame(columns=['ETF', 'Date', 'Volatility', 'Threshold'])

    # Fit GARCH model and forecast volatility
    for i in range(rolling_window, len(returns) - forecast_horizon):
        window = returns.iloc[i - rolling_window:i]
        model = arch_model(window, vol='Garch', p=1, q=1)
        try:
            res = model.fit(disp="off")
            forecast = res.forecast(horizon=forecast_horizon)
            avg_vol = np.mean(np.sqrt(forecast.variance.values[-1]) / 100)
            forecast_vols.append(avg_vol)
        except:
            forecast_vols.append(np.nan)

    # Create volatility signal and determine threshold
    valid_index = returns.index[rolling_window:-forecast_horizon]
    vol_signal = pd.Series(forecast_vols, index=valid_index)
    threshold = vol_signal.quantile(percentile_threshold)

    # Apply hedging strategy
    raw_returns = returns.loc[valid_index] / 100
    hedged = raw_returns.copy()

    for date in valid_index:
        vol = vol_signal.loc[date]
        if vol > threshold:
            decision = f"{date.date()}: De-risked (Vol={vol:.4f} > {threshold:.4f})"
            hedged.loc[date] *= (1 - de_risk_pct)
            # Add to hedge trigger dates DataFrame
            hedge_trigger_dates = pd.concat([hedge_trigger_dates, pd.DataFrame({
                'ETF': [etf_name],
```

```
                'Date': [date],
                'Volatility': [vol],
                'Threshold': [threshold]
            })], ignore_index=True)
        else:
            decision = f"{date.date()}: Full exposure (Vol={vol:.4f} <= {threshold:.4f})"
        decisions.append(decision)

    # Create DataFrame of hedged returns
    hedged_returns = pd.DataFrame({
        'raw_returns': raw_returns,
        'hedged_returns': hedged
    })

    # Sort hedge trigger dates by date
    hedge_trigger_dates = hedge_trigger_dates.sort_values('Date')

    print(f"Finished Processing {etf_name}")
    # Return results dictionary
    return {
        'etf_name': etf_name,
        'hedged_returns': hedged_returns,
        'hedge_log': decisions,
        'hedge_trigger_dates': hedge_trigger_dates,
        'volatility_signal': vol_signal,
        'threshold': threshold,
        'parameters': {
            'rolling_window': rolling_window,
            'forecast_horizon': forecast_horizon,
            'de_risk_pct': de_risk_pct,
            'percentile_threshold': percentile_threshold,
            'start_date': start_date,
            'end_date': end_date
        }

    }


def compute_garch_volatility(renewable_df,
                             etf_name,
                             rolling_window=500,
                             forecast_horizon=22,
                             start_date="2008",
                             end_date="2019"):
    """
    Compute GARCH volatility forecasts for a single ETF.

    Parameters:
    -----------
    renewable_df : pandas.DataFrame
        DataFrame containing ETF returns
    etf_name : str
        Name of the ETF to analyze
    rolling_window : int
        Window size for GARCH estimation
    forecast_horizon : int
        Number of days to forecast volatility
    start_date : str
        Start date for analysis
    end_date : str
        End date for analysis

    Returns:
    --------
    dict
        Dictionary containing volatility forecasts and parameters
    """
    # Ensure datetime index
    renewable_df.index = pd.to_datetime(renewable_df.index)
    renewable_subset = renewable_df.loc[start_date:end_date]

    print(f"Computing GARCH volatility for {etf_name}...")
    returns = renewable_subset[etf_name].dropna() * 100
    forecast_vols = []

    # Fit GARCH model and forecast volatility
    for i in range(rolling_window, len(returns) - forecast_horizon):
```

```
        window = returns.iloc[i - rolling_window:i]
        model = arch_model(window, vol='Garch', p=1, q=1)
        try:
            res = model.fit(disp="off")
            forecast = res.forecast(horizon=forecast_horizon)
            avg_vol = np.mean(np.sqrt(forecast.variance.values[-1]) / 100)
            forecast_vols.append(avg_vol)
        except:
            forecast_vols.append(np.nan)

    # Create volatility signal
    valid_index = returns.index[rolling_window:-forecast_horizon]
    vol_signal = pd.Series(forecast_vols, index=valid_index)

    print(f"Finished computing GARCH volatility for {etf_name}")
    return {
        'etf_name': etf_name,
        'volatility_signal': vol_signal,
        'raw_returns': returns.loc[valid_index] / 100,
        'parameters': {
            'rolling_window': rolling_window,
            'forecast_horizon': forecast_horizon,
            'start_date': start_date,
            'end_date': end_date
        }
    }

def apply_hedging_strategy(garch_results,
                           de_risk_pct=0.5,
                           percentile_threshold=0.9):
    """
    Apply hedging strategy based on GARCH volatility forecasts.

    Parameters:
    -----------
    garch_results : dict
        Results from compute_garch_volatility function
    de_risk_pct : float
        Percentage to de-risk when volatility is high
    percentile_threshold : float
        Percentile threshold for volatility (0-1)

    Returns:
    --------
    dict
        Dictionary containing hedging results and metrics
    """
    etf_name = garch_results['etf_name']
    vol_signal = garch_results['volatility_signal']
    raw_returns = garch_results['raw_returns']

    print(f"Applying hedging strategy for {etf_name}...")

    # Determine threshold
    threshold = vol_signal.quantile(percentile_threshold)

    # Apply hedging strategy
    hedged = raw_returns.copy()
    decisions = []
    hedge_trigger_dates = pd.DataFrame(columns=['ETF', 'Date', 'Volatility', 'Threshold'])

    for date in vol_signal.index:
        vol = vol_signal.loc[date]
        if vol > threshold:
            decision = f"{date.date()}: De-risked (Vol={vol:.4f} > {threshold:.4f})"
            hedged.loc[date] *= (1 - de_risk_pct)
            # Add to hedge trigger dates DataFrame
            hedge_trigger_dates = pd.concat([hedge_trigger_dates, pd.DataFrame({
                'ETF': [etf_name],
                'Date': [date],
                'Volatility': [vol],
                'Threshold': [threshold]
            })], ignore_index=True)
        else:
            decision = f"{date.date()}: Full exposure (Vol={vol:.4f} <= {threshold:.4f})"
        decisions.append(decision)
```

```python
    # Create DataFrame of hedged returns
    hedged_returns = pd.DataFrame({
        'raw_returns': raw_returns,
        'hedged_returns': hedged
    })

    # Sort hedge trigger dates by date
    hedge_trigger_dates = hedge_trigger_dates.sort_values('Date')

    print(f"Finished applying hedging strategy for {etf_name}")
    return {
        'etf_name': etf_name,
        'hedged_returns': hedged_returns,
        'hedge_log': decisions,
        'hedge_trigger_dates': hedge_trigger_dates,
        'volatility_signal': vol_signal,
        'threshold': threshold,
        'parameters': {
            **garch_results['parameters'],
            'de_risk_pct': de_risk_pct,
            'percentile_threshold': percentile_threshold
        }
    }
```

```python
# Compute GARCH volatilities
# ICLN
garch_vol_ICLN = compute_garch_volatility(
     renewable_df=renewable_df,
     etf_name='ICLN',
     rolling_window=500,
     forecast_horizon=22,
     start_date='2008-01-01',
     end_date='2019-12-31'
)
# PBW
garch_vol_PBW = compute_garch_volatility(
     renewable_df=renewable_df,
     etf_name='PBW',
     rolling_window=500,
     forecast_horizon=22,
     start_date='2008-01-01',
     end_date='2019-12-31'
)
# QCLN
garch_vol_QCLN = compute_garch_volatility(
     renewable_df=renewable_df,
     etf_name='QCLN',
     rolling_window=500,
     forecast_horizon=22,
     start_date='2008-01-01',
     end_date='2019-12-31'
)
```

```
Computing GARCH volatility for ICLN...
Finished computing GARCH volatility for ICLN
Computing GARCH volatility for PBW...
Finished computing GARCH volatility for PBW
Computing GARCH volatility for QCLN...
Finished computing GARCH volatility for QCLN
```

```python
# Perform hedge strategies with 50% derisk and 90% threshold
garch_hedge_ICLN = apply_hedging_strategy(
     garch_results=garch_vol_ICLN,
     de_risk_pct=0.5,
     percentile_threshold=0.9
)

garch_hedge_PBW = apply_hedging_strategy(
     garch_results=garch_vol_PBW,
     de_risk_pct=0.5,
     percentile_threshold=0.9
)

garch_hedge_QCLN = apply_hedging_strategy(
     garch_results=garch_vol_QCLN,
     de_risk_pct=0.5,
```

```
    percentile_threshold=0.9
)
```

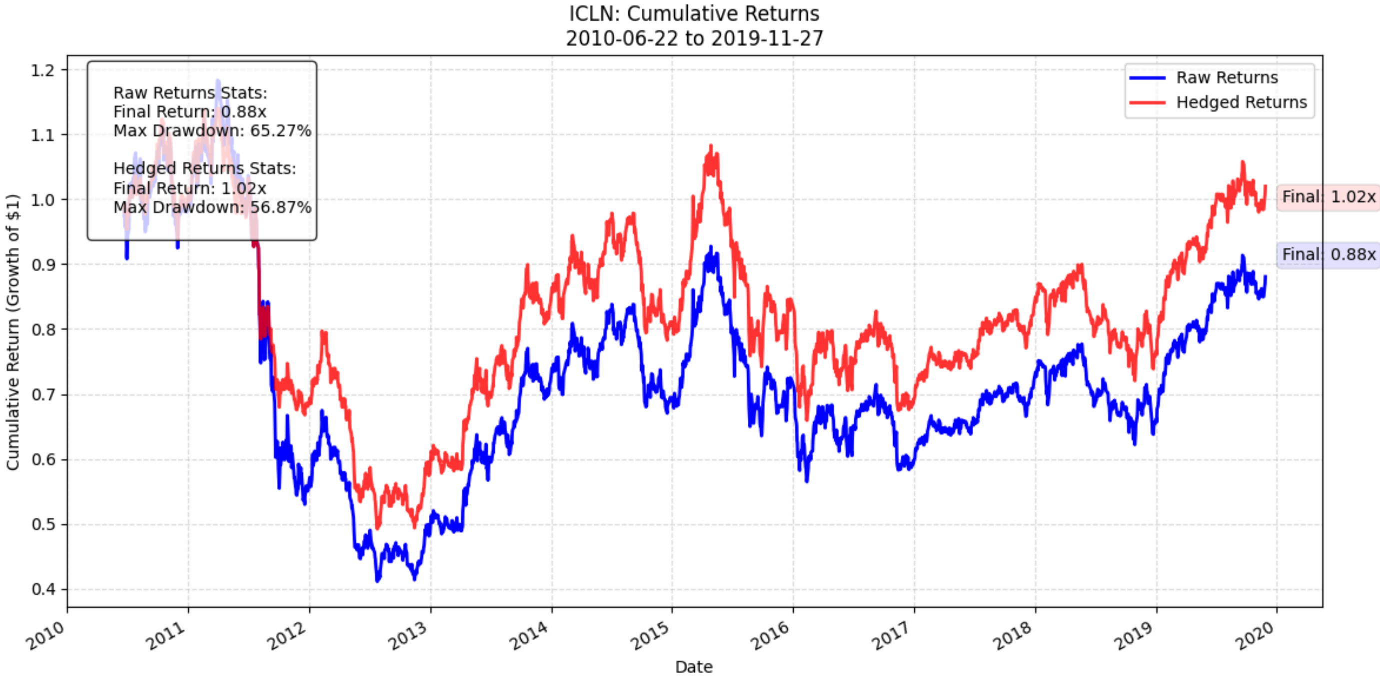
```
Applying hedging strategy for ICLN...
Finished applying hedging strategy for ICLN
Applying hedging strategy for PBW...
Finished applying hedging strategy for PBW
Applying hedging strategy for QCLN...
Finished applying hedging strategy for QCLN
```

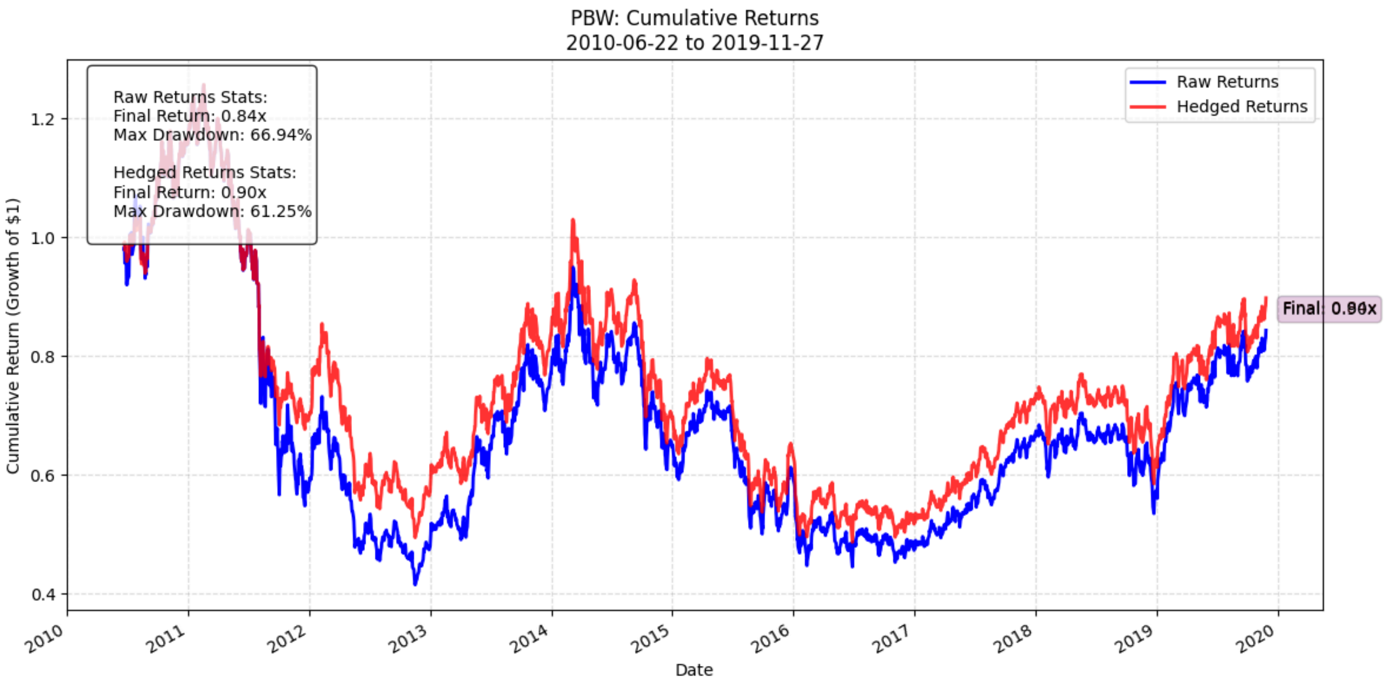
Start coding or generate with AI.

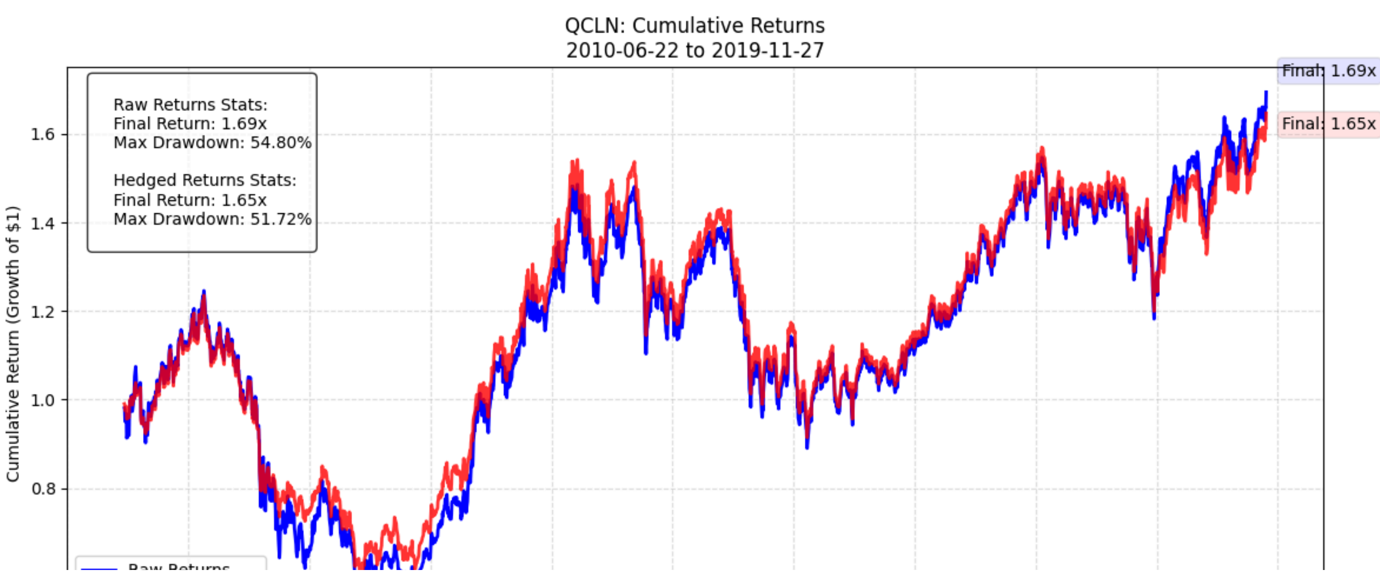
Need to experiment with de_risk_pct and percentile_threshold to minimize value at risk++ (while maintainint some sort of returns)

```
def plot_cumulative_returns(returns_df,
                          etf_name,
                          start_date=None,
                          end_date=None,
                          figsize=(12, 6),
                          linewidth=2,
                          alpha=0.8):
    """
    Plot cumulative returns for raw and hedged strategies.

    Parameters:
    -----------
    returns_df : pandas.DataFrame
        DataFrame with 'raw_returns' and 'hedged_returns' columns
    etf_name : str
        Name of the ETF for the plot title
    start_date : str, optional
        Start date for the plot (format: 'YYYY-MM-DD')
    end_date : str, optional
        End date for the plot (format: 'YYYY-MM-DD')
    figsize : tuple
        Figure size (width, height)
    linewidth : float
        Width of the lines
    alpha : float
        Transparency of the lines
    """
    # Filter by date range if provided
    if start_date:
        returns_df = returns_df.loc[start_date:]
    if end_date:
        returns_df = returns_df.loc[:end_date]

    # Calculate cumulative returns
    cum_raw = (1 + returns_df['raw_returns']).cumprod()
    cum_hedged = (1 + returns_df['hedged_returns']).cumprod()

    # Create the plot
    plt.figure(figsize=figsize)

    # Plot the lines
    plt.plot(cum_raw.index, cum_raw,
            label='Raw Returns',
            linewidth=linewidth,
            color='blue')

    plt.plot(cum_hedged.index, cum_hedged,
            label='Hedged Returns',
            linewidth=linewidth,
            alpha=alpha,
            color='red')

    # Add title and labels
    plt.title(f'{etf_name}: Cumulative Returns\n{returns_df.index[0].strftime("%Y-%m-%d")} to {returns_df.index[-1].strftime("%Y
    plt.xlabel('Date')
    plt.ylabel('Cumulative Return (Growth of $1)')

    # Add legend and grid
```

```python
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.4)

    # Format x-axis dates
    plt.gcf().autofmt_xdate()

    # Add final values as text annotations
    final_raw = cum_raw.iloc[-1]
    final_hedged = cum_hedged.iloc[-1]

    plt.annotate(f'Final: {final_raw:.2f}x',
                 xy=(cum_raw.index[-1], final_raw),
                 xytext=(10, 10), textcoords='offset points',
                 bbox=dict(boxstyle='round', facecolor='blue', alpha=0.1))

    plt.annotate(f'Final: {final_hedged:.2f}x',
                 xy=(cum_hedged.index[-1], final_hedged),
                 xytext=(10, -10), textcoords='offset points',
                 bbox=dict(boxstyle='round', facecolor='red', alpha=0.1))

    # Add some statistics as text
    stats_text = f"""
    Raw Returns Stats:
    Final Return: {final_raw:.2f}x
    Max Drawdown: {((cum_raw.cummax() - cum_raw) / cum_raw.cummax()).max():.2%}

    Hedged Returns Stats:
    Final Return: {final_hedged:.2f}x
    Max Drawdown: {((cum_hedged.cummax() - cum_hedged) / cum_hedged.cummax()).max():.2%}
    """

    # Add text box with statistics
    plt.text(0.02, 0.98, stats_text,
             transform=plt.gca().transAxes,
             verticalalignment='top',
             bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

    plt.tight_layout()
    plt.show()

    # Return the statistics
    return {
        'raw_stats': {
            'final_return': final_raw,
            'max_drawdown': ((cum_raw.cummax() - cum_raw) / cum_raw.cummax()).max()
        },
        'hedged_stats': {
            'final_return': final_hedged,
            'max_drawdown': ((cum_hedged.cummax() - cum_hedged) / cum_hedged.cummax()).max()
        }
    }



# Example usage:
plot_cumulative_returns(
    returns_df= garch_hedge_ICLN['hedged_returns'],
    etf_name='ICLN',
     start_date='2008-01-01',
     end_date='2019-12-31'
)
# Example usage:
plot_cumulative_returns(
    returns_df= garch_hedge_PBW['hedged_returns'],
    etf_name='PBW',
     start_date='2008-01-01',
     end_date='2019-12-31'
)
# Example usage:
plot_cumulative_returns(
    returns_df= garch_hedge_QCLN['hedged_returns'],
    etf_name='QCLN',
     start_date='2008-01-01',
     end_date='2019-12-31'
)
```


ICLN: Cumulative Returns
2010-06-22 to 2019-11-27
Raw Returns Stats:
Final Return: 0.88x
Max Drawdown: 65.27%
Hedged Returns Stats:
Final Return: 1.02x
Max Drawdown: 56.87%
Raw Returns
Hedged Returns
Final: 1.02x
Final: 0.88x
Cumulative Return (Growth of $1)
Date


PBW: Cumulative Returns
2010-06-22 to 2019-11-27
Raw Returns Stats:
Final Return: 0.84x
Max Drawdown: 66.94%
Hedged Returns Stats:
Final Return: 0.90x
Max Drawdown: 61.25%
Raw Returns
Hedged Returns
Cumulative Return (Growth of $1)
Date


QCLN: Cumulative Returns
2010-06-22 to 2019-11-27
Raw Returns Stats:
Final Return: 1.69x
Max Drawdown: 54.80%
Hedged Returns Stats:
Final Return: 1.65x
Max Drawdown: 51.72%
Final: 1.69x
Final: 1.65x
Cumulative Return (Growth of $1)


```
{'raw_stats': {'final_return': np.float64(1.6942908731104076),
  'max_drawdown': 0.5480178345644655},
 'hedged_stats': {'final_return': np.float64(1.6467953906668724),
  'max_drawdown': 0.517240033686768}}
```

```python
def risk_metrics(returns):
    """
    Calculate risk metrics for a returns series.

    Parameters:
    -----------
    returns : array-like
        Array or Series of returns

    Returns:
    --------
    dict
        Dictionary of risk metrics
    """
    # Convert to pandas Series if it's not already
    if not isinstance(returns, pd.Series):
        returns = pd.Series(returns)

    # Calculate metrics
    var = returns.quantile(0.05)
    cvar = returns[returns <= var].mean()
    mdd = (returns.cummax() - returns).max()

    # Return metrics dictionary
    return {
        "VaR (95%)": round(var, 4),
        "CVaR (95%)": round(cvar, 4),
        "Max Drawdown": round(mdd, 4)
    }


# Function to compare unhedged and hedged metrics
def compare_risk_metrics(returns_df):
    """
    Compare risk metrics between unhedged and hedged returns.

    Parameters:
    -----------
    returns_df : pandas.DataFrame
        DataFrame with 'raw_returns' and 'hedged_returns' columns

    Returns:
    --------
    pandas.DataFrame
        Comparison table of risk metrics
    """
    # Calculate metrics for both strategies
    unhedged_metrics = risk_metrics(returns_df['raw_returns'])
    hedged_metrics = risk_metrics(returns_df['hedged_returns'])

    # Create comparison DataFrame
    comparison = pd.DataFrame({
        'Unhedged': unhedged_metrics,
        'Hedged': hedged_metrics
    })
```

```
    # Calculate percent changes
    percent_changes = ((comparison['Hedged'] - comparison['Unhedged']) /
                       comparison['Unhedged'] * 100).round(2)
    comparison['Change (%)'] = percent_changes

    return comparison
```

```
compare_risk_metrics(garch_hedge_ICLN['hedged_returns'])
```

	Unhedged	Hedged	Change (%)
VaR (95%)	-0.0244	-0.0217	-11.0700
CVaR (95%)	-0.0344	-0.0296	-13.9500
Max Drawdown	0.1425	0.1222	-14.2500

```
compare_risk_metrics(garch_hedge_PBW['hedged_returns'])
```

	Unhedged	Hedged	Change (%)
VaR (95%)	-0.0278	-0.0255	-8.2700
CVaR (95%)	-0.0378	-0.0337	-10.8500
Max Drawdown	0.1639	0.1286	-21.5400

```
compare_risk_metrics(garch_hedge_QCLN['hedged_returns'])
```

	Unhedged	Hedged	Change (%)
VaR (95%)	-0.0275	-0.0252	-8.3600
CVaR (95%)	-0.0368	-0.0332	-9.7800
Max Drawdown	0.1291	0.1125	-12.8600

Hidden Markov Model on Realized Volatiliti

2-state Gaussigan HMM

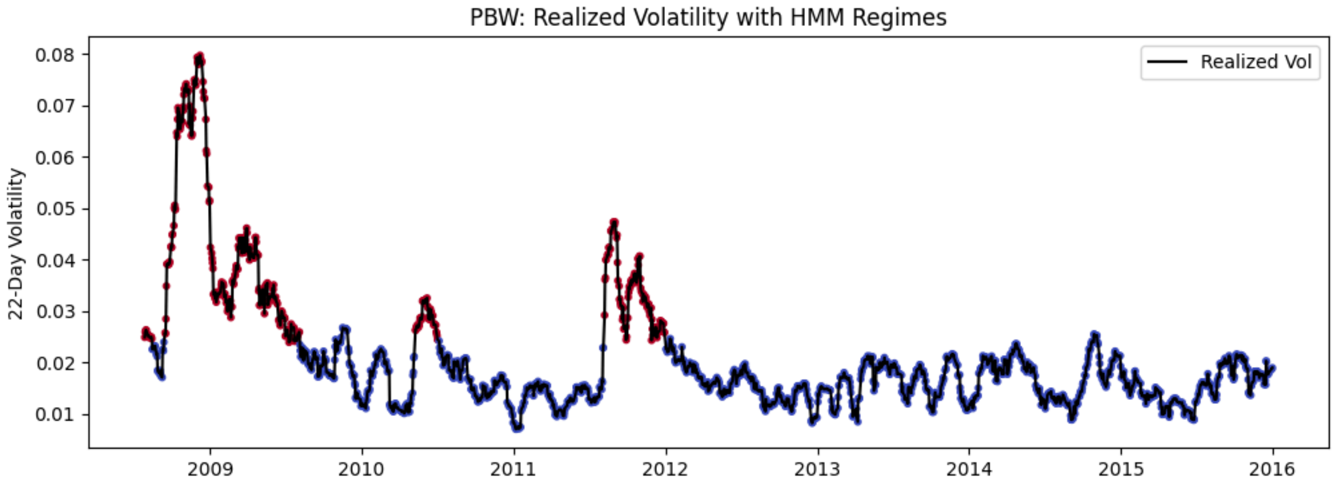
```
pip install hmmlearn
```

```
Collecting hmmlearn
  Downloading hmmlearn-0.3.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.0 kB)
Requirement already satisfied: numpy>=1.10 in /usr/local/lib/python3.11/dist-packages (from hmmlearn) (2.0.2)
Requirement already satisfied: scikit-learn!=0.22.0,>=0.16 in /usr/local/lib/python3.11/dist-packages (from hmmlearn) (1.6.1
Requirement already satisfied: scipy>=0.19 in /usr/local/lib/python3.11/dist-packages (from hmmlearn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn!=0.22.0,>=0.16->h
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn!=0.22.0,>=
Downloading hmmlearn-0.3.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (165 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 165.9/165.9 kB 3.0 MB/s eta 0:00:00
Installing collected packages: hmmlearn
Successfully installed hmmlearn-0.3.3
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from hmmlearn.hmm import GaussianHMM

# 1) Prepare your realized volatility series
#    Use your daily PBW returns (decimal) to compute a 22-day rolling std
ticker = 'PBW'
ret = renewable_df[ticker].loc['2008':'2015'].dropna()
realized_vol = ret.rolling(window=22).std().dropna()

# 2) Reshape for HMM (n_samples, n_features)
X = realized_vol.values.reshape(-1, 1)

# 3) Fit a 2-state Gaussian HMM
model = GaussianHMM(n_components=2, covariance_type='full', n_iter=100, random_state=42)
model.fit(X)
```

```
# 4) Decode the most likely state sequence
states = model.predict(X)
state_series = pd.Series(states, index=realized_vol.index)

# 5) Inspect the regimes
for i in [0,1]:
    mu = model.means_[i][0]
    sigma = np.sqrt(model.covars_[i][0][0])
    print(f"State {i}: mean vol = {mu:.4f},   std vol = {sigma:.4f}")

# 6) Plot realized vol colored by regime
plt.figure(figsize=(12,4))
plt.plot(realized_vol.index, realized_vol, label='Realized Vol', color='black')
plt.scatter(realized_vol.index, realized_vol, c=states, cmap='coolwarm', s=10)
plt.title(f"{ticker}: Realized Volatility with HMM Regimes")
plt.ylabel("22-Day Volatility")
plt.legend()
plt.show()
```

```
State 0: mean vol = 0.0161,   std vol = 0.0046
State 1: mean vol = 0.0384,   std vol = 0.0154
```

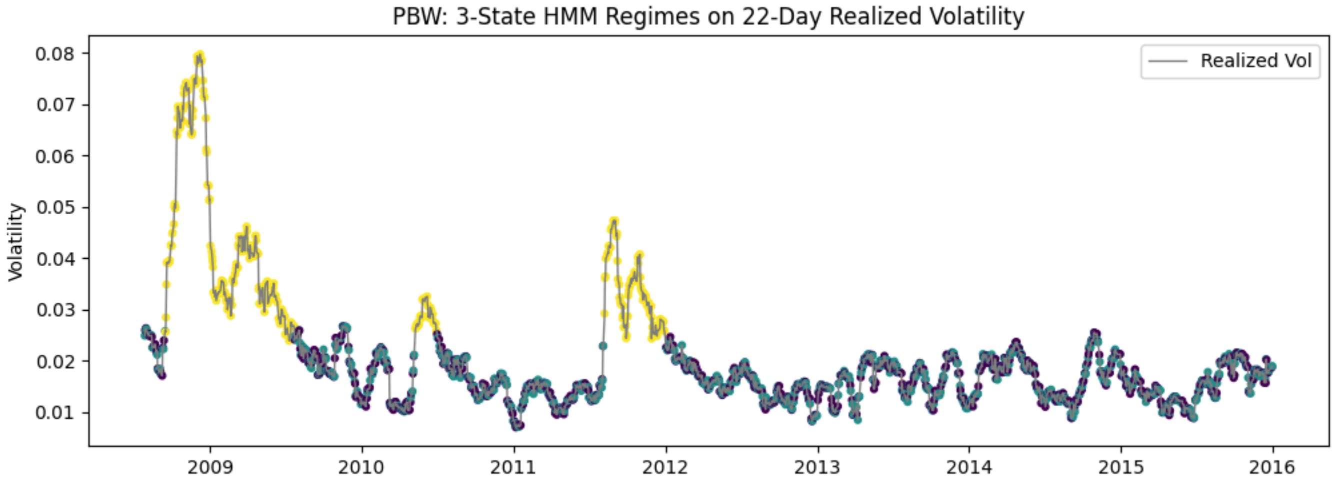


3-state Gaussian HMM

```
# 1) Compute 22-day realized volatility for PBW (2008–2019)
ticker = 'PBW'
ret = renewable_df[ticker].loc['2008':'2015'].dropna()
realized_vol = ret.rolling(window=22).std().dropna()

# 2) Reshape for HMM
X = realized_vol.values.reshape(-1, 1)

# 3) Fit a 3-state Gaussian HMM
model3 = GaussianHMM(
    n_components=3,
    covariance_type='full',
    n_iter=200,
    random_state=42
)
model3.fit(X)

# 4) Decode the most likely state sequence
states3 = model3.predict(X)
state_series3 = pd.Series(states3, index=realized_vol.index)

# 5) Inspect each regime’s volatility characteristics
print("Regime means & std devs:")
for i in range(3):
    mu    = model3.means_[i][0]
    sigma = np.sqrt(model3.covars_[i][0][0])
    print(f" State {i}: mean = {mu:.4f},  std = {sigma:.4f}")

# 6) Plot realized vol colored by regime
plt.figure(figsize=(12,4))
plt.plot(realized_vol.index, realized_vol, color='grey', linewidth=1, label='Realized Vol')
```

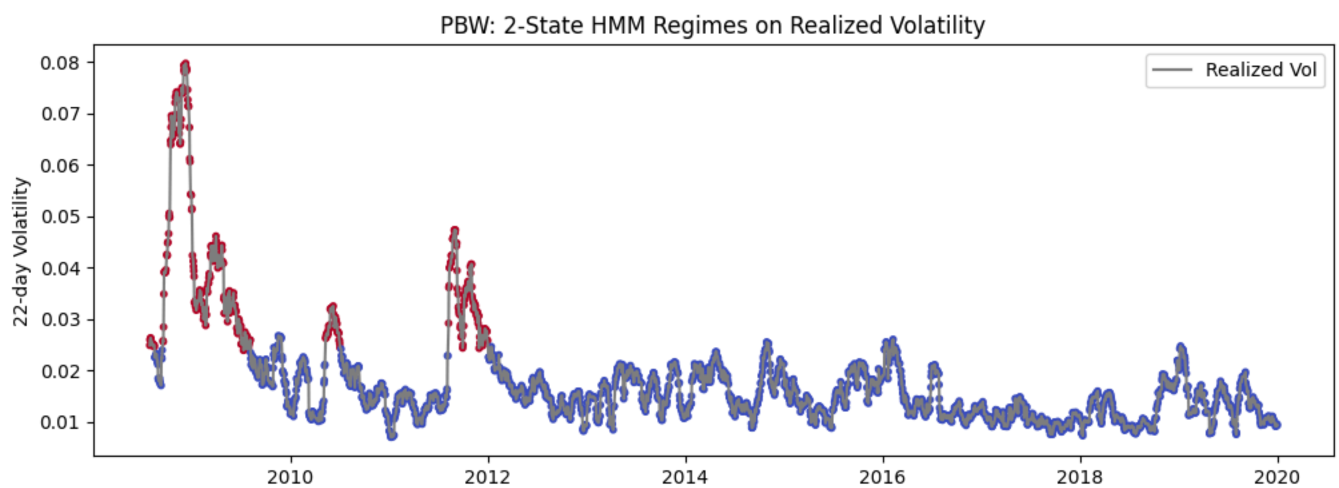
```
plt.scatter(realized_vol.index, realized_vol, c=states3, cmap='viridis', s=12)
plt.title(f"{ticker}: 3-State HMM Regimes on 22-Day Realized Volatility")
plt.ylabel("Volatility")
plt.legend()
plt.show()
```

```
Regime means & std devs:
 State 0: mean = 0.0164,  std = 0.0054
 State 1: mean = 0.0164,  std = 0.0054
 State 2: mean = 0.0400,  std = 0.0155
```




```
# Fit both models on the 2008–2015 train set.
# Validate on 2016–2018:


# Try to import hmmlearn, else instruct user to install
try:
    from hmmlearn.hmm import GaussianHMM
except ImportError as e:
    raise ImportError("hmmlearn not installed. Install via 'pip install hmmlearn' to use HMM functionality.") from e

# Parameters
ticker = 'PBW'
start_train = '2008-01-01'
end_train = '2015-12-31'
end_full = '2019-12-31'
forecast_horizon = 22
de_risk_pct = 0.5  # 50% reduction in exposure in high-vol state

# Ensure datetime index
returns_df = renewable_df.copy()
returns_df.index = pd.to_datetime(returns_df.index)

# 1) Compute 22-day realized volatility series
ret_full = returns_df[ticker].loc['2008':end_full].dropna()
realized_vol_full = ret_full.rolling(window=forecast_horizon).std().dropna()

# 2) Split realized volatility into train and test
rv_train = realized_vol_full.loc[start_train:end_train]
rv_test = realized_vol_full.loc['2016-01-01':end_full]

# 3) Fit 2-state Gaussian HMM on training data
X_train = rv_train.values.reshape(-1, 1)
model_hmm = GaussianHMM(n_components=2, covariance_type='full', n_iter=200, random_state=42)
model_hmm.fit(X_train)

# Identify high-vol state (state with higher mean)
means = model_hmm.means_.flatten()
high_vol_state = np.argmax(means)

# 4) Decode states for full period
X_full = realized_vol_full.values.reshape(-1, 1)
states_full = model_hmm.predict(X_full)
state_series = pd.Series(states_full, index=realized_vol_full.index, name='Regime')

# 5) Plot realized volatility with regimes
plt.figure(figsize=(12,4))
plt.plot(realized_vol_full.index, realized_vol_full, color='grey', label='Realized Vol')
```

```
plt.plot(realized_vol_full.index, realized_vol_full, color='grey', label='Realized Vol')
plt.scatter(realized_vol_full.index, realized_vol_full, c=states_full, cmap='coolwarm', s=10)
plt.title(f"{ticker}: 2-State HMM Regimes on Realized Volatility")
plt.ylabel("22-day Volatility")
plt.legend()
plt.show()

# 6) Construct hedged returns: reduce exposure when in high-vol state
raw_returns = ret_full.loc[state_series.index]  # align dates
hedged_returns = raw_returns.copy()
hedged_returns[state_series == high_vol_state] *= (1 - de_risk_pct)

# 7) Evaluate risk metrics on test period
def compute_metrics(returns):
    var = returns.quantile(0.05)
    cvar = returns[returns <= var].mean()
    mdd = (returns.cummax() - returns).max()
    return var, cvar, mdd

# Align test returns
test_returns = raw_returns.loc[rv_test.index]
test_hedged = hedged_returns.loc[rv_test.index]

metrics_orig = compute_metrics(test_returns)
metrics_hedged = compute_metrics(test_hedged)

# Display metrics
metrics_df = pd.DataFrame({
    'Original': metrics_orig,
    'Hedged': metrics_hedged
}, index=['VaR (95%)','CVaR (95%)','Max Drawdown'])

metrics_df
```



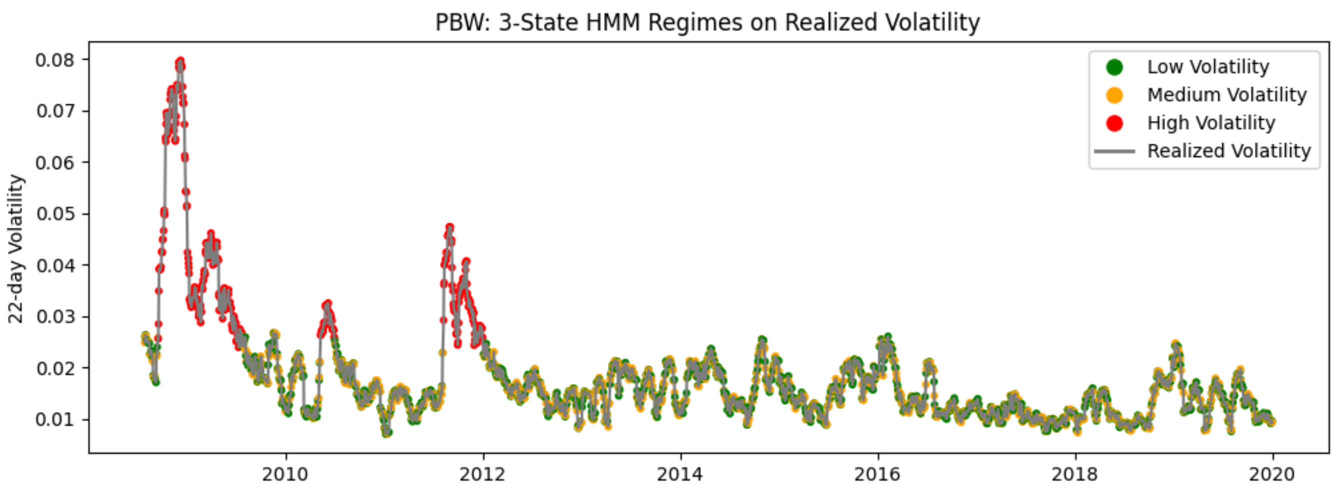
	Original	Hedged
VaR (95%)	-0.0214	-0.0214
CVaR (95%)	-0.0299	-0.0299
Max Drawdown	0.1002	0.1002

Next steps: Generate code with `metrics_df` View recommended plots New interactive sheet

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Try to import hmmlearn, else instruct user to install
try:
    from hmmlearn.hmm import GaussianHMM
except ImportError as e:
    raise ImportError("hmmlearn not installed. Install via 'pip install hmmlearn' to use HMM functionality.") from e

# Parameters
ticker = 'PBW'
start_train = '2008-01-01'
```

```
start_train = '2008-01-01'
end_train = '2015-12-31'
end_full = '2019-12-31'
forecast_horizon = 22

# De-risking percentages for each state
de_risk_high = 0.50  # 50% reduction in high volatility state
de_risk_medium = 0.25  # 25% reduction in medium volatility state
de_risk_low = 0.00  # 0% reduction in low volatility state

# Ensure datetime index
returns_df = renewable_df.copy()
returns_df.index = pd.to_datetime(returns_df.index)

# 1) Compute 22-day realized volatility series
ret_full = returns_df[ticker].loc['2008':end_full].dropna()
realized_vol_full = ret_full.rolling(window=forecast_horizon).std().dropna()

# 2) Split realized volatility into train and test
rv_train = realized_vol_full.loc[start_train:end_train]
rv_test = realized_vol_full.loc['2016-01-01':end_full]

# 3) Fit 3-state Gaussian HMM on training data
X_train = rv_train.values.reshape(-1, 1)
model_hmm = GaussianHMM(n_components=3, covariance_type='full', n_iter=200, random_state=42)
model_hmm.fit(X_train)

# Identify states by their mean volatility (low to high)
means = model_hmm.means_.flatten()
sorted_states = np.argsort(means)  # States ordered from low to high volatility
low_vol_state = sorted_states[0]
medium_vol_state = sorted_states[1]
high_vol_state = sorted_states[2]

# 4) Decode states for full period
X_full = realized_vol_full.values.reshape(-1, 1)
states_full = model_hmm.predict(X_full)
state_series = pd.Series(states_full, index=realized_vol_full.index, name='Regime')

# 5) Plot realized volatility with regimes
# 5) Plot realized volatility with regimes
plt.figure(figsize=(12,4))
plt.plot(realized_vol_full.index, realized_vol_full, color='grey', label='Realized Vol')

# Create a custom colormap for the three states
colors = ['green', 'orange', 'red']  # Low, Medium, High volatility
state_colors = [colors[state] for state in states_full]

# Plot the scatter points with custom colors
scatter = plt.scatter(realized_vol_full.index, realized_vol_full,
                     c=state_colors, s=10)

# Create custom legend
from matplotlib.lines import Line2D
legend_elements = [
    Line2D([0], [0], marker='o', color='w', markerfacecolor='green', markersize=10, label='Low Volatility'),
    Line2D([0], [0], marker='o', color='w', markerfacecolor='orange', markersize=10, label='Medium Volatility'),
    Line2D([0], [0], marker='o', color='w', markerfacecolor='red', markersize=10, label='High Volatility'),
    Line2D([0], [0], color='grey', lw=2, label='Realized Volatility')
]

plt.title(f"{ticker}: 3-State HMM Regimes on Realized Volatility")
plt.ylabel("22-day Volatility")
plt.legend(handles=legend_elements)
plt.show()

# 6) Construct hedged returns with state-dependent de-risking
raw_returns = ret_full.loc[state_series.index]  # align dates
hedged_returns = raw_returns.copy()

# Apply different de-risking based on state
hedged_returns[state_series == high_vol_state] *= (1 - de_risk_high)
hedged_returns[state_series == medium_vol_state] *= (1 - de_risk_medium)
hedged_returns[state_series == low_vol_state] *= (1 - de_risk_low)

# 7) Evaluate risk metrics on test period
def compute_metrics(returns):
    var = returns.quantile(0.05)
```

```
    cvar = returns[returns <= var].mean()
    mdd = (returns.cummax() - returns).max()
    return var, cvar, mdd

# Align test returns
test_returns = raw_returns.loc[rv_test.index]
test_hedged = hedged_returns.loc[rv_test.index]

metrics_orig = compute_metrics(test_returns)
metrics_hedged = compute_metrics(test_hedged)

# Display metrics
metrics_df = pd.DataFrame({
    'Original': metrics_orig,
    'Hedged': metrics_hedged
}, index=['VaR (95%)','CVaR (95%)','Max Drawdown'])

print("State Means (Low to High Volatility):")
print(f"Low Volatility State: {means[low_vol_state]:.4f}")
print(f"Medium Volatility State: {means[medium_vol_state]:.4f}")
print(f"High Volatility State: {means[high_vol_state]:.4f}")

print("\nRisk Metrics Comparison:")
metrics_df
```



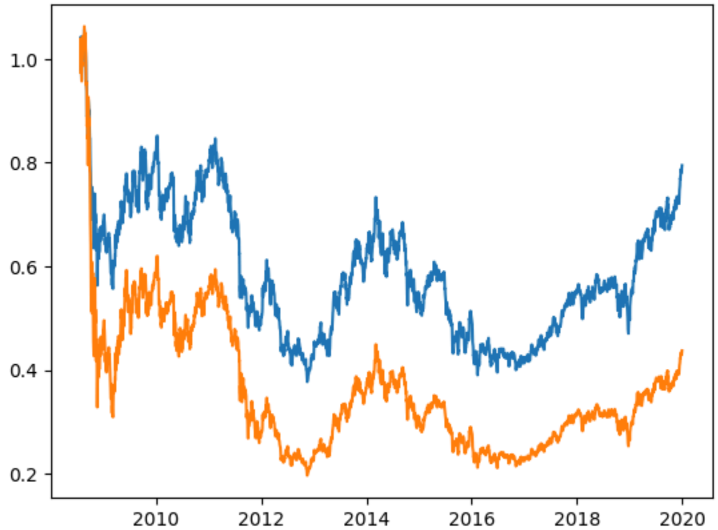

```
State Means (Low to High Volatility):
Low Volatility State: 0.0164
Medium Volatility State: 0.0164
High Volatility State: 0.0400

Risk Metrics Comparison:
```

	Original	Hedged
VaR (95%)	-0.0214	-0.0197
CVaR (95%)	-0.0299	-0.0267
Max Drawdown	0.1002	0.1002

Next steps: Generate code with metrics_df | View recommended plots | New interactive sheet

```
plt.plot((1 + hedged_returns).cumprod())
plt.plot((1 + raw_returns).cumprod())
```

[<matplotlib.lines.Line2D at 0x7cb378147350>]




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from hmmlearn.hmm import GaussianHMM

# Parameters
ticker = 'QCLN'
start_train = '2010-01-01'
end_train = '2015-12-31'
start_test = '2019-01-01'
end_full = '2024-12-31'
forecast_horizon = 22

# De-risking percentages for each state
de_risk_high = 0.50  # 50% reduction in high volatility state
de_risk_medium = 0.25  # 25% reduction in medium volatility state
de_risk_low = 0.00  # 0% reduction in low volatility state

# Ensure datetime index
returns_df = renewable_df.copy()
returns_df.index = pd.to_datetime(returns_df.index)

# 1) Compute 22-day realized volatility series
ret_full = returns_df[ticker].loc[start_train:end_full].dropna()
realized_vol_full = ret_full.rolling(window=forecast_horizon).std().dropna()

def rolling_hmm_states(realized_vol, window=1000, n_states=3):
    states = np.full(len(realized_vol), np.nan)
    idx = realized_vol.index
    X = realized_vol.values.reshape(-1,1)

    # Clean the data - remove any remaining NaN or inf values
    valid_mask = ~np.isnan(X.flatten()) & ~np.isinf(X.flatten())
    X_clean = X[valid_mask]
    idx_clean = idx[valid_mask]

    for t in range(window, len(X_clean)):
        # Get the window of data
        window_data = X_clean[t-window:t]

        # Skip if window contains NaN or inf values
        if np.any(np.isnan(window_data)) or np.any(np.isinf(window_data)):
            continue

        try:
            model = GaussianHMM(n_components=n_states,
                              covariance_type='diag',  # Changed to diag for better stability
                              n_iter=100,
                              tol=1e-2,
                              random_state=0)
            model.fit(window_data)
            states[t] = model.predict(X_clean[t:t+1])[0]
        except:
```

```
            continue

    return pd.Series(states, index=idx_clean)

# Get rolling HMM states
states_series = rolling_hmm_states(realized_vol_full, window=750, n_states=3)

# Identify states by their mean volatility in the most recent window
def get_state_means(vol_series, states_series, window=750):
    X = vol_series.values.reshape(-1,1)
    valid_mask = ~np.isnan(X.flatten()) & ~np.isinf(X.flatten())
    X_clean = X[valid_mask]

    if len(X_clean) < window:
        window = len(X_clean)

    model = GaussianHMM(n_components=3,
                        covariance_type='diag',  # Changed to diag for better stability
                        n_iter=100,
                        tol=1e-2,
                        random_state=0)
    model.fit(X_clean[-window:])
    means = model.means_.flatten()
    sorted_states = np.argsort(means)  # States ordered from low to high volatility
    return sorted_states

# Get state means and identify high/medium/low states
sorted_states = get_state_means(realized_vol_full, states_series)
low_vol_state = sorted_states[0]
medium_vol_state = sorted_states[1]
high_vol_state = sorted_states[2]

# Plot the states
plt.figure(figsize=(12,4))
plt.plot(realized_vol_full.index, realized_vol_full, color='grey', label='Realized Vol')

# Create a custom colormap for the three states
colors = ['green', 'orange', 'red']  # Low, Medium, High volatility
state_colors = [colors[int(state)] if not np.isnan(state) else 'grey' for state in states_series]

# Plot the scatter points with custom colors
scatter = plt.scatter(states_series.index, realized_vol_full.loc[states_series.index],
                     c=state_colors, s=10)

# Create custom legend
from matplotlib.lines import Line2D
legend_elements = [
    Line2D([0], [0], marker='o', color='w', markerfacecolor='green', markersize=10, label='Low Volatility'),
    Line2D([0], [0], marker='o', color='w', markerfacecolor='orange', markersize=10, label='Medium Volatility'),
    Line2D([0], [0], marker='o', color='w', markerfacecolor='red', markersize=10, label='High Volatility'),
    Line2D([0], [0], color='grey', lw=2, label='Realized Volatility')
]

plt.title(f"{ticker}: Rolling 3-State HMM Regimes on Realized Volatility")
plt.ylabel("22-day Volatility")
plt.legend(handles=legend_elements)
plt.show()

# Construct hedged returns with state-dependent de-risking
raw_returns = ret_full.loc[states_series.index]  # align dates
hedged_returns = raw_returns.copy()

# Apply different de-risking based on state
hedged_returns[states_series == high_vol_state] *= (1 - de_risk_high)
hedged_returns[states_series == medium_vol_state] *= (1 - de_risk_medium)
hedged_returns[states_series == low_vol_state] *= (1 - de_risk_low)

# Evaluate risk metrics on test period
def compute_metrics(returns):
    var = returns.quantile(0.05)
    cvar = returns[returns <= var].mean()
    mdd = (returns.cummax() - returns).max()
    return var, cvar, mdd

# Align test returns
test_returns = raw_returns.loc[start_test:end_full]
test_hedged = hedged_returns.loc[start_test:end_full]
```

```
metrics_orig = compute_metrics(test_returns)
metrics_hedged = compute_metrics(test_hedged)


# Display metrics
metrics_df = pd.DataFrame({
    'Original': metrics_orig,
    'Hedged': metrics_hedged
}, index=['VaR (95%)','CVaR (95%)','Max Drawdown'])

print("State Means (Low to High Volatility):")
print(f"Low Volatility State: {sorted_states[0]}")
print(f"Medium Volatility State: {sorted_states[1]}")
print(f"High Volatility State: {sorted_states[2]}")

print("\nRisk Metrics Comparison:")
metrics_df
```

```
Model is not converging.  Current: 2664.650859185498 is not greater than 2664.6522742732336. Delta is -0.0014150877354950353
Model is not converging.  Current: 2663.9994175063694 is not greater than 2664.0007101904366. Delta is -0.001292684067266236
Model is not converging.  Current: 2663.3724187562393 is not greater than 2663.3734703173936. Delta is -0.0010515611543326918
Model is not converging.  Current: 2662.7017264495203 is not greater than 2662.7026225649074. Delta is -0.0008961153871149620
Model is not converging.  Current: 2662.091515100676 is not greater than 2662.091941352391. Delta is -0.0004262517145434685
Model is not converging.  Current: 2661.518601978697 is not greater than 2661.518744119714. Delta is -0.00014214101702236803
Model is not converging.  Current: 2661.4722656771864 is not greater than 2661.472532349826. Delta is -0.0002666726395545993
Model is not converging.  Current: 2662.0637508840136 is not greater than 2662.0644126845928. Delta is -0.0006618005791109694
Model is not converging.  Current: 2665.828234169133 is not greater than 2665.82858724485. Delta is -0.00035307571715748054
Model is not converging.  Current: 2666.478318288456 is not greater than 2666.4784575393983. Delta is -0.00013925094208838905
Model is not converging.  Current: 2666.7680534406522 is not greater than 2666.768178938376. Delta is -0.0001254977237294952
Model is not converging.  Current: 2666.953421596827 is not greater than 2666.9537071259774. Delta is -0.0002855291504602064
Model is not converging.  Current: 2667.13868111561 is not greater than 2667.1394014336975. Delta is -0.0007203180875876569
Model is not converging.  Current: 2667.23106650818 is not greater than 2667.231342570647. Delta is -0.00027606246703726356
Model is not converging.  Current: 2667.2279841195223 is not greater than 2667.2285648631514. Delta is -0.0005807436291433398
Model is not converging.  Current: 2667.276511593538 is not greater than 2667.2771005447516. Delta is -0.0005889512135581754
Model is not converging.  Current: 2667.4217918898335 is not greater than 2667.4223537391176. Delta is -0.0005618492841167692
Model is not converging.  Current: 2668.204109492316 is not greater than 2668.205375743872. Delta is -0.001266251555895837
Model is not converging.  Current: 2668.8947798955874 is not greater than 2668.8959891547875. Delta is -0.0012092592000095402
Model is not converging.  Current: 2669.6743600764053 is not greater than 2669.6757092999687. Delta is -0.0013492235633465784
Model is not converging.  Current: 2676.6558582634616 is not greater than 2676.656887370209. Delta is -0.001029106747409969
Model is not converging.  Current: 2677.3555452896685 is not greater than 2677.355562660422. Delta is -1.737075353958062e-05
Model is not converging.  Current: 2687.039216321803 is not greater than 2687.0392567132835. Delta is -4.039148052470409e-05
Model is not converging.  Current: 2689.428159986071 is not greater than 2689.4288786133716. Delta is -0.0007186273005557273
Model is not converging.  Current: 2693.1958957261845 is not greater than 2693.1961147677052. Delta is -0.000219041520722385
Model is not converging.  Current: 2816.8899580274683 is not greater than 2816.8931493066743. Delta is -0.003191279206021136
Model is not converging.  Current: 2860.922103798721 is not greater than 2860.9231751944035. Delta is -0.0010713956826293725
Model is not converging.  Current: 2870.1886212598247 is not greater than 2870.190029406528. Delta is -0.001408146703397506
Model is not converging.  Current: 2876.795149932971 is not greater than 2876.8033023706444. Delta is -0.00815243767328866
Model is not converging.  Current: 2879.7097533201395 is not greater than 2879.7230450284087. Delta is -0.013291708269207447
Model is not converging.  Current: 2883.259676450303 is not greater than 2883.2657709821465. Delta is -0.006094531843700679
Model is not converging.  Current: 2897.53764103426 is not greater than 2897.56928718882. Delta is -0.03164615455989406
Model is not converging.  Current: 3005.466966377353 is not greater than 3005.4707364987426. Delta is -0.003770121389607084
Model is not converging.  Current: 2984.8215832286833 is not greater than 2985.5541473046665. Delta is -0.7325640759831913
Model is not converging.  Current: 2905.632700019678 is not greater than 2905.885792612606. Delta is -0.2530925929281693
Model is not converging.  Current: 2935.109956659063 is not greater than 2935.156014711031. Delta is -0.04605805196797519
Model is not converging.  Current: 2932.584192456584 is not greater than 2932.647955188872. Delta is -0.06376273228806895
Model is not converging.  Current: 2926.7421302758926 is not greater than 2928.171826996125. Delta is -1.4296967202326414
Model is not converging.  Current: 2922.608087402272 is not greater than 2923.603095337513. Delta is -0.9950079352411194
Model is not converging.  Current: 2920.4996351650057 is not greater than 2921.475322636899. Delta is -0.9756874718932522
Model is not converging.  Current: 2915.6672326398234 is not greater than 2916.0947202944863. Delta is -0.42748765466285477
Model is not converging.  Current: 2313.4287602898344 is not greater than 2313.474137735864. Delta is -0.045377446029760904
Model is not converging.  Current: 2301.7724075003603 is not greater than 2301.9080227468944. Delta is -0.1356152465341438
Model is not converging.  Current: 2300.789170532197 is not greater than 2300.9306998656425. Delta is -0.14152933344530538
Model is not converging.  Current: 2299.895590207075 is not greater than 2300.042818349563. Delta is -0.14722814248762006
Model is not converging.  Current: 2298.909515302413 is not greater than 2299.0629076659543. Delta is -0.15339236354111563
Model is not converging.  Current: 2291.3769352666063 is not greater than 2291.5541027651807. Delta is -0.17716749857436298
Model is not converging.  Current: 2290.870378465795 is not greater than 2291.0477918237434. Delta is -0.17741335794835322
Model is not converging.  Current: 2290.096007213549 is not greater than 2290.272720742794. Delta is -0.17671352924480743
Model is not converging.  Current: 2289.404690351921 is not greater than 2289.5802827684365. Delta is -0.17559241651542834
Model is not converging.  Current: 2289.127413710624 is not greater than 2289.302666405136. Delta is -0.17525269451198255
Model is not converging.  Current: 2288.935815276277 is not greater than 2289.110200739854. Delta is -0.17438546357698215
Model is not converging.  Current: 2288.4446061811245 is not greater than 2288.617965617693. Delta is -0.17335943656826203
Model is not converging.  Current: 2287.9182421966566 is not greater than 2288.0897928005. Delta is -0.17155060384357057
Model is not converging.  Current: 2287.660844226393 is not greater than 2287.8314539528064. Delta is -0.1706097264136588
Model is not converging.  Current: 2287.2587419496363 is not greater than 2287.428299007209. Delta is -0.1695570575725469
Model is not converging.  Current: 2286.8668441216946 is not greater than 2287.0348457806463. Delta is -0.16800165895165264
Model is not converging.  Current: 2286.5058209065 is not greater than 2286.6725834915546. Delta is -0.16676258505458463
Model is not converging.  Current: 2286.1800757062956 is not greater than 2286.3453185854514. Delta is -0.16524287915581226
Model is not converging.  Current: 2285.525954020255 is not greater than 2285.687001082891. Delta is -0.16104706263604385
Model is not converging.  Current: 2285.537164294001 is not greater than 2285.6977282358976. Delta is -0.16056394189672574
Model is not converging.  Current: 2896.899880864427 is not greater than 2896.9000453369017. Delta is -0.0001644724748075532
Model is not converging.  Current: 2285.5762530463053 is not greater than 2285.734336551401. Delta is -0.158083505095874
Model is not converging.  Current: 2285.58407151557 is not greater than 2285.7414638610358. Delta is -0.15739234546572334
Model is not converging.  Current: 2285.6229910526554 is not greater than 2285.779145199425. Delta is -0.15615414676949513
Model is not converging.  Current: 2285.6607547523035 is not greater than 2285.816365762694. Delta is -0.1556110103906576
Model is not converging.  Current: 2285.6733089930576 is not greater than 2285.827989964411. Delta is -0.15468097135317294
Model is not converging.  Current: 2285.711955294858 is not greater than 2285.8655381829344. Delta is -0.15358288807647114
Model is not converging.  Current: 2285.998413898823 is not greater than 2286.1470432282845. Delta is -0.14862932946152796
Model is not converging.  Current: 2875.158986727231 is not greater than 2875.204183418853. Delta is -0.045196691622095386
Model is not converging.  Current: 2286.3731837358346 is not greater than 2286.517550958848. Delta is -0.14436722301343252
Model is not converging.  Current: 2286.437644284274 is not greater than 2286.5812331171073. Delta is -0.14358883283330215
Model is not converging.  Current: 2836.661784546166 is not greater than 2836.8998931304714. Delta is -0.238108584305337
Model is not converging.  Current: 2821.32666706914 is not greater than 2821.6241094016686. Delta is -0.29744233252858976
Model is not converging.  Current: 2821.3036401560144 is not greater than 2822.94101043758. Delta is -1.6373702815658362
Model is not converging.  Current: 2816.7756034120002 is not greater than 2818.9003801866074. Delta is -2.1247767746071986
Model is not converging.  Current: 2814.478882766246 is not greater than 2816.4137865731514. Delta is -1.9349038069053677
Model is not converging.  Current: 2813.753887250843 is not greater than 2815.0254990319618. Delta is -1.271611781118736
Model is not converging.  Current: 2814.9100209654807 is not greater than 2814.967420602095. Delta is -0.05739963661426373
Model is not converging.  Current: 2809.1925543286075 is not greater than 2809.641629117008. Delta is -0.4490747884005941
Model is not converging.  Current: 2808.805835689087 is not greater than 2808.924277231521. Delta is -0.11844154243408411
Model is not converging.  Current: 2281.549323653421 is not greater than 2281.6197284495856. Delta is -0.07040479616443918
Model is not converging.  Current: 2283.55695435404 is not greater than 2283.593575474839. Delta is -0.03662112079882718
Model is not converging.  Current: 2713.912601793748 is not greater than 2714.088851981801. Delta is -0.17625018805301806
Model is not converging.  Current: 2267.0882101169814 is not greater than 2267.088437712489. Delta is -0.0002275955075674574
```

```
Model is not converging.  Current: 2267.0882101169814 is not greater than 2267.0884377712489. Delta is -0.0002275955075674574
Model is not converging.  Current: 2265.1817691638694 is not greater than 2265.182577114578. Delta is -0.0008079507088041282
Model is not converging.  Current: 2263.293177039332 is not greater than 2263.294769950887. Delta is -0.001592911554780585
Model is not converging.  Current: 2261.0016588554745 is not greater than 2261.0046037776883. Delta is -0.002944922213828249
Model is not converging.  Current: 2258.817742575603 is not greater than 2258.8214687989. Delta is -0.003726223296780517 6
Model is not converging.  Current: 2244.756751830613 is not greater than 2244.766982510317. Delta is -0.010230679703909118
Model is not converging.  Current: 2242.4473120526754 is not greater than 2242.4584840684247. Delta is -0.01117201574925275
Model is not converging.  Current: 2234.551971870735 is not greater than 2234.5566484251403. Delta is -0.004676554405250499
Model is not converging.  Current: 2232.6982005959744 is not greater than 2232.7028910236077. Delta is -0.004690427633249783
Model is not converging.  Current: 2223.251088979757 is not greater than 2223.2695233109357. Delta is -0.018434331178923458
Model is not converging.  Current: 2223.0100796455986 is not greater than 2223.028672766444. Delta is -0.018593120845252997
Model is not converging.  Current: 2222.765193287416 is not greater than 2222.78393217248. Delta is -0.01873888506406729
Model is not converging.  Current: 2222.1602718495706 is not greater than 2222.1795230754647. Delta is -0.019251225894095114
Model is not converging.  Current: 2222.0952867834203 is not greater than 2222.114612173141. Delta is -0.01932538972050679
Model is not converging.  Current: 2222.073171764604 is not greater than 2222.0924937933046. Delta is -0.019322028700571536
Model is not converging.  Current: 2222.1687425307373 is not greater than 2222.188145539819. Delta is -0.019403009081543132
Model is not converging.  Current: 2222.2711378040367 is not greater than 2222.290682216737. Delta is -0.019544412700270186
Model is not converging.  Current: 2222.3558451051367 is not greater than 2222.3753578040996. Delta is -0.019512698962898867
Model is not converging.  Current: 2222.5014373011104 is not greater than 2222.5209939993965. Delta is -0.019556698286322372
Model is not converging.  Current: 2222.751528352415 is not greater than 2222.771066263495. Delta is -0.01953791108007863
Model is not converging.  Current: 2223.015875352742 is not greater than 2223.035417977872. Delta is -0.01954262512981586
Model is not converging.  Current: 2223.28788719608 is not greater than 2223.307434957873. Delta is -0.019547761792637175
Model is not converging.  Current: 2223.5593901328284 is not greater than 2223.5789332346576. Delta is -0.019543101829185616
Model is not converging.  Current: 2223.8361513220666 is not greater than 2223.8557351456325. Delta is -0.019583823565881175
Model is not converging.  Current: 2224.124248231495 is not greater than 2224.1437936557063. Delta is -0.019545424211173668
Model is not converging.  Current: 2224.385216475905 is not greater than 2224.4047860579394. Delta is -0.01956958203436443
Model is not converging.  Current: 2224.66963861107 is not greater than 2224.6891850843954. Delta is -0.01954647332559034
Model is not converging.  Current: 2224.9303412098097 is not greater than 2224.94988574221. Delta is -0.01954453240023213
Model is not converging.  Current: 2225.2295064699606 is not greater than 2225.2490575618644. Delta is -0.019551091903849738
Model is not converging.  Current: 2225.440738592505 is not greater than 2225.4603652655474. Delta is -0.019626673042239418
Model is not converging.  Current: 2225.6665763842257 is not greater than 2225.686215072144. Delta is -0.01963868791835921
Model is not converging.  Current: 2226.1735088248633 is not greater than 2226.1932466306394. Delta is -0.019737805776600337
Model is not converging.  Current: 2226.409980901821 is not greater than 2226.429733439982. Delta is -0.019752538160901167
Model is not converging.  Current: 2226.712260328953 is not greater than 2226.7319892347105. Delta is -0.019728905757347093
Model is not converging.  Current: 2227.476041941543 is not greater than 2227.495756474205. Delta is -0.019714532661964768
Model is not converging.  Current: 2227.89804192946 is not greater than 2227.9177265735393. Delta is -0.019684644079461577
Model is not converging.  Current: 2228.337048594635 is not greater than 2228.356587400592. Delta is -0.019538805957381555
Model is not converging.  Current: 2228.846318473173 is not greater than 2228.865778678566. Delta is -0.019460205392988428
Model is not converging.  Current: 2229.9335624646233 is not greater than 2229.9529960044943. Delta is -0.019433539870988817
Model is not converging.  Current: 2230.5285324040233 is not greater than 2230.5336498924153. Delta is -0.005117488391988445
Model is not converging.  Current: 2231.8627433492184 is not greater than 2231.8677171903123. Delta is -0.00497384109394261
Model is not converging.  Current: 2232.5312823928252 is not greater than 2232.536500099618. Delta is -0.005217706792791432
Model is not converging.  Current: 2233.1394650289185 is not greater than 2233.1446607846137. Delta is -0.005195755695240223
Model is not converging.  Current: 2233.7193427501807 is not greater than 2233.7320940781988. Delta is -0.012751328018111963
Model is not converging.  Current: 2234.257035065275 is not greater than 2234.2626758665397. Delta is -0.005640801264689799
Model is not converging.  Current: 2234.6870886844035 is not greater than 2234.6997295023184. Delta is -0.012640817914871l5
Model is not converging.  Current: 2235.1818233186805 is not greater than 2235.194425560974. Delta is -0.012602242293723975
Model is not converging.  Current: 2235.608605070899 is not greater than 2235.621245464175. Delta is -0.012640393275887618
Model is not converging.  Current: 2235.98701307494 is not greater than 2235.9996479082793. Delta is -0.01263483333923432
Model is not converging.  Current: 2236.375289755494 is not greater than 2236.38798839737. Delta is -0.012698641875886096
Model is not converging.  Current: 2236.7374858753096 is not greater than 2236.743267952544. Delta is -0.005782077234471217
Model is not converging.  Current: 2237.056743483746 is not greater than 2237.0624822332834. Delta is -0.00573874953761333
Model is not converging.  Current: 2237.3995702583593 is not greater than 2237.4121214734496. Delta is -0.012551215090297774
Model is not converging.  Current: 2237.7587127351344 is not greater than 2237.7772117468917. Delta is -0.01849901175728519
Model is not converging.  Current: 2238.1234714697775 is not greater than 2238.142029532976. Delta is -0.018558063198270247
Model is not converging.  Current: 2238.509037714305 is not greater than 2238.527633783785. Delta is -0.018596069479826838
Model is not converging.  Current: 2238.877908563268 is not greater than 2238.890513981444. Delta is -0.012605418176008243
Model is not converging.  Current: 2239.218671148481 is not greater than 2239.2371118210895. Delta is -0.01844067260844895
Model is not converging.  Current: 2239.601983328059 is not greater than 2239.6084851583146. Delta is -0.006501830255729146
Model is not converging.  Current: 2239.898331968613 is not greater than 2239.916742453965. Delta is -0.018410485351978423
Model is not converging.  Current: 2240.1836330937404 is not greater than 2240.201969706821. Delta is -0.018336613080464303
Model is not converging.  Current: 2240.5244135297016 is not greater than 2240.542751376643. Delta is -0.018337846941449243
Model is not converging.  Current: 2240.846760459788 is not greater than 2240.8519935547347. Delta is -0.005233094946561323
Model is not converging.  Current: 2241.190312960558 is not greater than 2241.195566481133. Delta is -0.005253520575024595
Model is not converging.  Current: 2241.5641920563185 is not greater than 2241.5694387143208. Delta is -0.005246658002306503
Model is not converging.  Current: 2242.004721464614 is not greater than 2242.0109934814286. Delta is -0.006272016814818926
Model is not converging.  Current: 2242.4306169522174 is not greater than 2242.44853908573. Delta is -0.01792213351245664
Model is not converging.  Current: 2242.8768621830113 is not greater than 2242.894742677367. Delta is -0.017880494355722476
Model is not converging.  Current: 2243.3204963736316 is not greater than 2243.3258107771812. Delta is -0.005314403549618873
Model is not converging.  Current: 2243.7660958388797 is not greater than 2243.7714381870896. Delta is -0.005342348209978809
Model is not converging.  Current: 2244.2817117702507 is not greater than 2244.2992599410436. Delta is -0.017548170792906603
Model is not converging.  Current: 2244.7471800171634 is not greater than 2244.7525575267246. Delta is -0.005377509561185434
Model is not converging.  Current: 2245.2708149043083 is not greater than 2245.2883249717293. Delta is -0.01751006742097161
Model is not converging.  Current: 2245.7247974652355 is not greater than 2245.7422624073993. Delta is -0.017464942163769592
Model is not converging.  Current: 2246.178702747582 is not greater than 2246.196313855925. Delta is -0.01761110834286228
Model is not converging.  Current: 2246.550221078354 is not greater than 2246.5677455329546. Delta is -0.017524454600788886
Model is not converging.  Current: 2246.867073947013 is not greater than 2246.8724893557683. Delta is -0.005415408755197859
Model is not converging.  Current: 2247.254412386781 is not greater than 2247.2719566697506. Delta is -0.01754428296953847
Model is not converging.  Current: 2247.590167141268 is not greater than 2247.6077007216004. Delta is -0.01753358033238328
Model is not converging.  Current: 2247.904664946154 is not greater than 2247.9102119160666. Delta is -0.005546969912757049
Model is not converging.  Current: 2248.253463785365 is not greater than 2248.2709265490807. Delta is -0.01746276371568456
Model is not converging.  Current: 2248.59890768579 is not greater than 2248.604489779042. Delta is -0.005582093252087361
Model is not converging.  Current: 2248.9658893505307 is not greater than 2248.971488890104. Delta is -0.005599539573267975
Model is not converging.  Current: 2249.3371096348646 is not greater than 2249.342717213863. Delta is -0.005607578998478857
Model is not converging.  Current: 2249.670042792015 is not greater than 2249.675680076043. Delta is -0.005637284028125578
```

```
Model is not converging.  Current: 2250.005178787457 is not greater than 2250.0108360923005. Delta is -0.005657304843225575
Model is not converging.  Current: 2250.3307283261083 is not greater than 2250.3363948858873. Delta is -0.005666559779001545
Model is not converging.  Current: 2250.6502757630924 is not greater than 2250.655963737179. Delta is -0.0056879740864133055
Model is not converging.  Current: 2251.5664979802336 is not greater than 2251.5835518765675. Delta is -0.017053896333891316
Model is not converging.  Current: 2252.1462907142327 is not greater than 2252.16329562432. Delta is -0.017004910087507596
Model is not converging.  Current: 2252.422153010202 is not greater than 2252.4389331823677. Delta is -0.01678017216590888
Model is not converging.  Current: 2253.0680587375614 is not greater than 2253.0848670554988. Delta is -0.01680831793737525
Model is not converging.  Current: 2253.384984329567 is not greater than 2253.4015388762095. Delta is -0.016554546642510104
Model is not converging.  Current: 2253.7524134486484 is not greater than 2253.7691084533108. Delta is -0.01669500466232421
Model is not converging.  Current: 2254.7605428011943 is not greater than 2254.7768996323234. Delta is -0.016356831129087368
Model is not converging.  Current: 2255.117993704807 is not greater than 2255.134278218526. Delta is -0.01628451371925621
Model is not converging.  Current: 2255.4442271786734 is not greater than 2255.460421539509. Delta is -0.016194360835470434
Model is not converging.  Current: 2255.789077447358 is not greater than 2255.805250452328. Delta is -0.016173004969914473
Model is not converging.  Current: 2256.1272208515193 is not greater than 2256.143355196736. Delta is -0.016134345216869406
Model is not converging.  Current: 2256.484239309447 is not greater than 2256.5002656176753. Delta is -0.016026308228447306
Model is not converging.  Current: 2256.8664955659733 is not greater than 2256.8822075608796. Delta is -0.015711994906268956
Model is not converging.  Current: 2257.2575943460465 is not greater than 2257.2733982040995. Delta is -0.01580385805300466
Model is not converging.  Current: 2257.6682130542154 is not greater than 2257.6839063879006. Delta is -0.015693333685248945
Model is not converging.  Current: 2258.044584541364 is not greater than 2258.050298675512. Delta is -0.005714134147638106
Model is not converging.  Current: 2258.473798175842 is not greater than 2258.479514089201. Delta is -0.0057159133589266276
Model is not converging.  Current: 2259.136429007996 is not greater than 2259.142140937055. Delta is -0.005711929059089016
Model is not converging.  Current: 2259.6221736721236 is not greater than 2259.6279014675715. Delta is -0.005727795447910466
Model is not converging.  Current: 2260.1045395627557 is not greater than 2260.110208104536. Delta is -0.0056685417803237215
Model is not converging.  Current: 2260.58091314008 is not greater than 2260.5866147911847. Delta is -0.005701651104573102
Model is not converging.  Current: 2260.9721244218367 is not greater than 2260.977807768516. Delta is -0.005683346679234091
Model is not converging.  Current: 2261.380942412209 is not greater than 2261.3865790932214. Delta is -0.005636681012219924
Model is not converging.  Current: 2261.8169694349886 is not greater than 2261.82264466604. Delta is -0.005675231051554874
Model is not converging.  Current: 2262.2199399934193 is not greater than 2262.2256491712237. Delta is -0.005709177804419596
Model is not converging.  Current: 2262.875932861216 is not greater than 2262.881604380386. Delta is -0.005671519169936801
Model is not converging.  Current: 2263.2313603895363 is not greater than 2263.237051787598. Delta is -0.005691398061571817
Model is not converging.  Current: 2265.7394602349927 is not greater than 2265.7452368694476. Delta is -0.005776634454832674
Model is not converging.  Current: 2266.386713293099 is not greater than 2266.4008737420745. Delta is -0.014160448975417239
Model is not converging.  Current: 2266.761856720686 is not greater than 2266.7762469767804. Delta is -0.014390256094429787
Model is not converging.  Current: 2267.191242538294 is not greater than 2267.2055303596817. Delta is -0.014287821387824806
Model is not converging.  Current: 2267.275448632276 is not greater than 2267.2812913651196. Delta is -0.005842732843575504
Model is not converging.  Current: 2267.542856130743 is not greater than 2267.5571195520815. Delta is -0.014263421338455373
Model is not converging.  Current: 2267.664667993896 is not greater than 2267.670556652132. Delta is -0.005888658236017363
Model is not converging.  Current: 2268.414839873765 is not greater than 2268.420741630181. Delta is -0.0059017564162786584
Model is not converging.  Current: 2268.400927989525 is not greater than 2268.4068289480724. Delta is -0.005900958547499613
Model is not converging.  Current: 2268.4012692699725 is not greater than 2268.4156611235467. Delta is -0.01439185357412498
Model is not converging.  Current: 2268.3516169574145 is not greater than 2268.3575203906803. Delta is -0.005903433265757485
Model is not converging.  Current: 2267.9529216831934 is not greater than 2267.9588355705273. Delta is -0.005913887333917955
Model is not converging.  Current: 2267.7235225843624 is not greater than 2267.729435596013. Delta is -0.005913011650591216
Model is not converging.  Current: 2267.630786930503 is not greater than 2267.6366850835543. Delta is -0.005898153051475674
Model is not converging.  Current: 2267.54522105669 is not greater than 2267.5511328307716. Delta is -0.005911774081596377
Model is not converging.  Current: 2267.469212857426 is not greater than 2267.4751231059317. Delta is -0.005910248505642812
Model is not converging.  Current: 2267.388552835849 is not greater than 2267.3944548349446. Delta is -0.0059019990953856905
Model is not converging.  Current: 2267.310800726724 is not greater than 2267.316720220088. Delta is -0.00591949336376274
Model is not converging.  Current: 2266.8730421935165 is not greater than 2266.8887251338424. Delta is -0.015682940325859818
Model is not converging.  Current: 2263.2459807034456 is not greater than 2263.260535479804. Delta is -0.014554776358181698
Model is not converging.  Current: 2268.710556339901 is not greater than 2268.7307305294044. Delta is -0.02017418950345018
Model is not converging.  Current: 2269.5190068548122 is not greater than 2269.538905622491. Delta is -0.01989876767856913
Model is not converging.  Current: 2270.33009335182 is not greater than 2270.3504239270987. Delta is -0.020330575278876495
Model is not converging.  Current: 2271.1455338006085 is not greater than 2271.1652525764093. Delta is -0.019718775800811272
Model is not converging.  Current: 2271.956520272363 is not greater than 2271.9758973158155. Delta is -0.019377043452550424
Model is not converging.  Current: 2272.8936628315014 is not greater than 2272.9134995694303. Delta is -0.019836737928926595
Model is not converging.  Current: 2274.4403125373824 is not greater than 2274.4596645495844. Delta is -0.01935201220203453
Model is not converging.  Current: 2275.2072977512994 is not greater than 2275.2269371403822. Delta is -0.019639389082840353
Model is not converging.  Current: 2275.9882056042948 is not greater than 2276.007368009684. Delta is -0.019162405389124615
Model is not converging.  Current: 2276.752158982868 is not greater than 2276.771154031476. Delta is -0.01899504860784873
Model is not converging.  Current: 2277.535714043129 is not greater than 2277.5551909038427. Delta is -0.019476860713893984
Model is not converging.  Current: 2278.045099078945 is not greater than 2278.064599384255. Delta is -0.019500305309975374
Model is not converging.  Current: 2278.346026805469 is not greater than 2278.3655363794. Delta is -0.019509573930918123
Model is not converging.  Current: 2278.638902036873 is not greater than 2278.658427514412. Delta is -0.019525477538991254
Model is not converging.  Current: 2277.1828251798984 is not greater than 2277.199486585493. Delta is -0.01666140559473206
Model is not converging.  Current: 2276.414305304563 is not greater than 2276.4311973388963. Delta is -0.016892034333068295
Model is not converging.  Current: 2274.9065920774688 is not greater than 2274.927545500096. Delta is -0.020953422627371765
Model is not converging.  Current: 2274.3259023377873 is not greater than 2274.3469270424844. Delta is -0.021024704697083507
Model is not converging.  Current: 2273.619020728382 is not greater than 2273.64022459943. Delta is -0.02120387104787369
Model is not converging.  Current: 2272.9453416040583 is not greater than 2272.966692425815. Delta is -0.02135082175664138
Model is not converging.  Current: 2271.362481613658 is not greater than 2271.3840734606842. Delta is -0.021591847026229516
Model is not converging.  Current: 2268.5573903633795 is not greater than 2268.5736595519757. Delta is -0.016269188596197637
Model is not converging.  Current: 2266.8489861752887 is not greater than 2266.8709711178212. Delta is -0.021984942532526475
Model is not converging.  Current: 2266.315614358183 is not greater than 2266.3321113486436. Delta is -0.01649699046038222
Model is not converging.  Current: 2265.7795243987357 is not greater than 2265.7960425385677. Delta is -0.016518139831987355
Model is not converging.  Current: 2265.635489676947 is not greater than 2265.652025907682. Delta is -0.016536230734800483
Model is not converging.  Current: 2265.4825866676188 is not greater than 2265.50428435884. Delta is -0.021697691221106652
Model is not converging.  Current: 2264.99839387016 is not greater than 2265.014871863431. Delta is -0.01647799327110988
Model is not converging.  Current: 2265.0199910315223 is not greater than 2265.041920799083. Delta is -0.02192976756077769
Model is not converging.  Current: 2264.804016408307 is not greater than 2264.8204706211823. Delta is -0.01645421287548743
Model is not converging.  Current: 2264.5688915727887 is not greater than 2264.5908215131826. Delta is -0.02192994039387486
Model is not converging.  Current: 2264.818697250883 is not greater than 2264.8410090675084. Delta is -0.02231181662546078
Model is not converging.  Current: 2265.0864816492954 is not greater than 2265.1088478802617. Delta is -0.02236623096632684
Model is not converging.  Current: 2265.3443754437994 is not greater than 2265.3667375631653. Delta is -0.022362119365880062
Model is not converging.  Current: 2265.7752362545193 is not greater than 2265.7974490923225. Delta is -0.022212837803181174
```