FIN42120 - Programming for Financial Data Science

Group Project:

Presented By: Group 5

Kuntoji, Rohan - 22202093 (MSc. Financial Data Science)

Nagar, Shraddha - 22200477 (MSc. Financial Data Science)

Fernandes, William - 22201553 (MSc. Financial Data Science)

Pant, Vidushi - 22200808 (MSc. Financial Data Science)

Parary, Aman - 18417714 (MSc. Financial Data Science)

May 12, 2023

Introduction

The objective of this project is to analyze the performance of the U.S. stock and bond returns over the period from December 1979 to December 2021, and to generate out-of-sample excess return forecasts based on different predictive variables and models. The analysis includes computing various statistics for the stock and bond returns, generating time-series of monthly out-of-sample constant expected excess return forecasts using a recursive estimate approach and rolling window estimate approach, and using five plausible predictors to generate monthly out-of-sample excess return forecasts for each asset class.

The five plausible predictors used in the analysis of equity market are the inflation (infl), book-to-market ratio (BM), stock variance (SVAR), net equity expansion (NTIS), and dividend-payout ratio (D/E). Additionally, for the bond market, the analysis includes the variables of inflation (infl), Treasury bill rate (TBL), long-term yield (LTY), long-term return (LTR) and default yield spread (DFY).

The importance of analyzing the performance of the U.S. stock and bond returns lies in their impact on the economy and financial markets. Investors use stock and bond returns as a measure of their investment performance, while policy-makers use them as a gauge of the overall health of the economy. Additionally, the project's analysis of the predictive models and variables can provide insights into the factors that drive stock and bond returns, helping investors make informed investment decisions.

The data used in the project includes monthly prices of the S&P 500 stock market index (SP500) (hereafter referred as the stock index) from the U.S. Federal Reserve Economic Data (FRED) website, monthly prices of the Bloomberg Barclays U.S. Aggregate Bond Index (LBUSTRUU) (hereafter referred as the bond index) from Bloomberg, and monthly data on the risk-free rate of return from Professor Kenneth French's data library at Dartmouth College. The variables used in the analysis include annualized mean/average, annualized volatility, annualized Sharpe ratio, skewness, and kurtosis.The analysis is carried out in two parts- in Part A - Recursive Estimate Approach is used and in Part B – Rolling Window Estimate Approach is adopted.

Overall, the project aims to provide a comprehensive analysis of the U.S. stock and bond returns and their predictive models, with the goal of helping investors make informed investment decisions.

Part A: Recursive Estimate Approach

Recursive estimation involves training a model on a historical period from $y(0)$ to $y(n)$ and predicting the value of $\hat{y}(n+1)$. The model is then retrained using the extended historical period from $y(0)$ to $y(n+1)$ to predict $\hat{y}(n+2)$, and this process is repeated until the end of the sample period. The window of historical data used for training the model increases with each prediction, and only one-step-ahead predictions are made.

Question 1

Statistic	Stock Index	Bond Index
Mean	0.062294	0.032172
Volatility	0.150755	0.051107
Sharpe	0.413216	0.629497
Skew	-0.632931	0.420890
Kurtosis	5.084584	9.168519

Table 1: Summary Statistic U.S. stock and bond simple excess returns

- Over the period from December 1979 to December 2021, the stock index provided higher mean excess returns (6.23% per annum) compared to the bond index (3.22% per annum).
- However, the stock index was also more volatile (15.08%) than the bond index (5.11%), which resulted in a lower Sharpe ratio (0.41) compared to the bond index (0.63).

- The skewness of the excess returns was negative for the stock index and positive for the bond index, indicating that there were more negative returns than positive returns for the stock index and more positive returns than negative returns for the bond index over the period.
- The kurtosis of the excess returns was high for both indices, indicating that the distribution of returns was fat-tailed and had more extreme values than a normal distribution.
- Overall, the bond index provided a more attractive investment opportunity in terms of the Sharpe ratio, which considers both the excess returns and the volatility of the returns.

Question 2

The total sample has been divided into two periods: an in-sample from January 1980-December 1999 and an out-of-sample period from January 2000-December 2021. To generate a time-series of monthly out-of-sample constant expected (mean) excess return forecasts (call it the mean benchmark forecast) for each of the two asset classes, a recursive estimate approach has been used. A Python function has been written to do this computation. The function returns a pandas DataFrame (mean_benchmark_forecasts_df) containing the mean benchmark forecasts for both asset classes for each month in the out-of-sample period.

Question 3

Monthly data for five plausible predictors of the asset class excess returns have been downloaded. Using the same recursive estimation approach, monthly out-of-sample excess return forecasts have been generated for each of the asset classes as below:

(i) OLS predictive regression model for each of the five predictors
The output shows Out-of-sample OLS Predictor forecasts for each asset class using OLS predictive regression models fitted on the five predictors

(ii) Combination forecasts of excess returns that is a simple average of the forecasts based on the five predictors from the OLS model.
A combination mean forecasts using the out-of-sample OLS predictor forecasts for the two asset classes is generated. The function 'generate_combination_mean_forecasts' takes as inputs the OLS predictor forecasts for the stock and bond markets, and the names of the two assets. The resulting data frame contains the combination mean forecasts for the two assets as Out-of-sample Combination Mean forecasts.

(iii) Any 2 penalized linear regressions

o LASSO regression performs variable selection by shrinking the regression coefficients of some predictors to zero. It is particularly useful when the number of predictors is large, and some of them may not be relevant in explaining the variation in the response variable.

o Ridge regression, on the other hand, adds a penalty term to the least squares estimation, which is proportional to the square of the magnitude of the coefficients. The penalty term helps to prevent overfitting by shrinking the coefficients towards zero but not necessarily to zero. Ridge regression is often used when the predictors are highly correlated, and there is a risk of multicollinearity.

The stock index and bond index out-of-sample return predictions using LASSO and Ridge regression techniques provides monthly excess return forecasts for a particular regression approach. For instance, the Lasso and Ridge projections for the excess returns of the stock index in January 2000 are 1.56% and 1.02%, respectively. This suggests that when compared to the Ridge model, the Lasso model forecasts a larger excess return for the stock index. The Lasso and Ridge forecasts for bonds are similarly -0.0028% and 0.0027%, indicating that the two models are forecasting somewhat different returns.

Predictor No.	Variable Name	β_0	β_1	t-stat	p-value	R-Squared
1	Infl	0.0067	-0.6608	-1.184	0.237	0.003
2	BM	0.0052	-0.0003	-0.038	0.969	0.000
3	SVAR	0.0057	0.2584	-0.784	0.434	0.001
4	NTIS	0.0051	-0.0167	-0.172	0.863	0.000
5	D/E	0.0069	0.0024	0.437	0.662	0.000

Table 2: Recursive Predictive Models for SP500

Predictor No.	Variable Name	β_0	β_1	t-stat	p-value	R-Squared
1	Infl	0.0038	-0.3971	-2.119	0.035	0.009
2	TBL	0.0024	0.0093	0.506	0.613	0.001
3	LTY	0.0010	0.0282	1.397	0.163	0.004
4	LTR	0.0023	0.0653	3.253	0.001	0.021
5	DFY	0.0002	0.2395	1.673	0.095	0.006

Table 3: Recursive Predictive Models for LBUSTRUU

The analysis of predictive models for stock index and bond index revealed limited statistical significance and explanatory power for the examined variables. The variables tested in both markets showed weak relationships and low explanatory power, indicating their limited impact on return variations.

Further the benchmark and eight forecast model predictions are combined for each asset class using a recursive approach. For both asset classes, the benchmark model and the other eight recursive predictive models' MSFE (Mean Squared Forecast Error) values are computed. The average forecast error for each model is measured by the MSFE values, which may be used to assess how well the models estimate the returns on the assets. The other models employ a variety of methods, including Penalized Linear Regression (PLR), Lasso, and Ridge regression, while the benchmark model is based on historical average returns. By comparing the models' results, the MSFE values make it possible to choose the most effective model for forecasting the returns of each asset class.

Also the ratio of MSFE's of predictive models to benchmark MSFE values is calculated. This allows for a comparison of model accuracy, with a ratio >1 indicating poorer performance and <1 indicating better performance.

Model(SP500)	MSFE	Ratios of MSFE	DM t-stat	DM p-value
Benchmark	0.001903	1	-	-
Predictor 1	0.001960	1.0299	1.8761	0.0617
Predictor 2	0.001914	1.0058	2.3015	0.0221
Predictor 3	0.001968	1.0338	0.5989	0.5497
Predictor 4	0.001927	1.0126	2.0183	0.0445
Predictor 5	0.001936	1.0173	0.7946	0.4275
Combined	0.001924	1.0110	0.9427	0.3466
PLR_1 (LASSO)	0.001909	1.0029	0.9516	0.3421
PLR_2 (RIDGE)	0.001932	1.0153	0.9702	0.3328
Model (LBUSTRUU)	**MSFE**	**Ratios of MSFE**	**DM t-stat**	**DM p-value**
Benchmark	0.000094	1	-	-
Predictor 1	0.000093	0.9900	-0.2706	0.7868
Predictor 2	0.000095	1.0112	1.4657	0.1439
Predictor 3	0.000097	1.0265	1.2875	0.1990
Predictor 4	0.000096	1.0166	0.4365	0.6628
Predictor 5	0.000093	0.9858	-0.9819	0.3270
Combined	0.000093	0.9857	-0.9904	0.3228
PLR_1 (LASSO)	0.000100	1.0653	1.1748	0.2411
PLR_2 (RIDGE)	0.000095	1.0123	0.5245	0.6003

Table 4: Ratios of MSFE for predictive models relative to benchmark for SP500 & LBUSTRUU

The Diebold-Mariano (DM) test in Table 4 was utilized to assess the equal predictive ability of all eight recursive predictive models compared to the mean benchmark forecasts for both asset classes.

The null hypothesis for the Diebold-Mariano (DM) test is that there is no difference in the predictive accuracy of two models, while the alternative hypothesis is that one model has superior predictive accuracy compared to the other. In other words, the DM test is used to determine whether the difference in forecast accuracy between two models is statistically significant, with the null hypothesis assuming that there is no significant difference, and the alternative hypothesis assuming that there is a significant difference.

Upon reviewing the p-values of stock index forecast models, it appears that most of the models don't exhibit statistically significant differences in forecast accuracy compared to the benchmark. The p-values are relatively high, suggesting that the differences observed in the mean squared forecast error (MSFE) values are likely due to random variations rather than meaningful distinctions in predictive ability.

The MSFE values and ratios for the predictive models, including the benchmark, for bond index, suggest that the models do not significantly outperform or underperform the benchmark in terms of forecast accuracy. The differences in MSFE values and ratios are relatively small, indicating similar performance between the models and the benchmark.

The two subplots in Figure 1 visualizes the out-of-sample recursive forecasts for the stock index (S&P 500) and bond index (US Aggregate Bond Index). The selected models, including the benchmark, combination mean, Lasso, and Ridge. For stock index, high volatility (dip in returns) is noted around 2008 crisis under all the models, but rest of the return series appear fairly stable, but on the contrary volatility has been noted throughout the return series for bonds index.

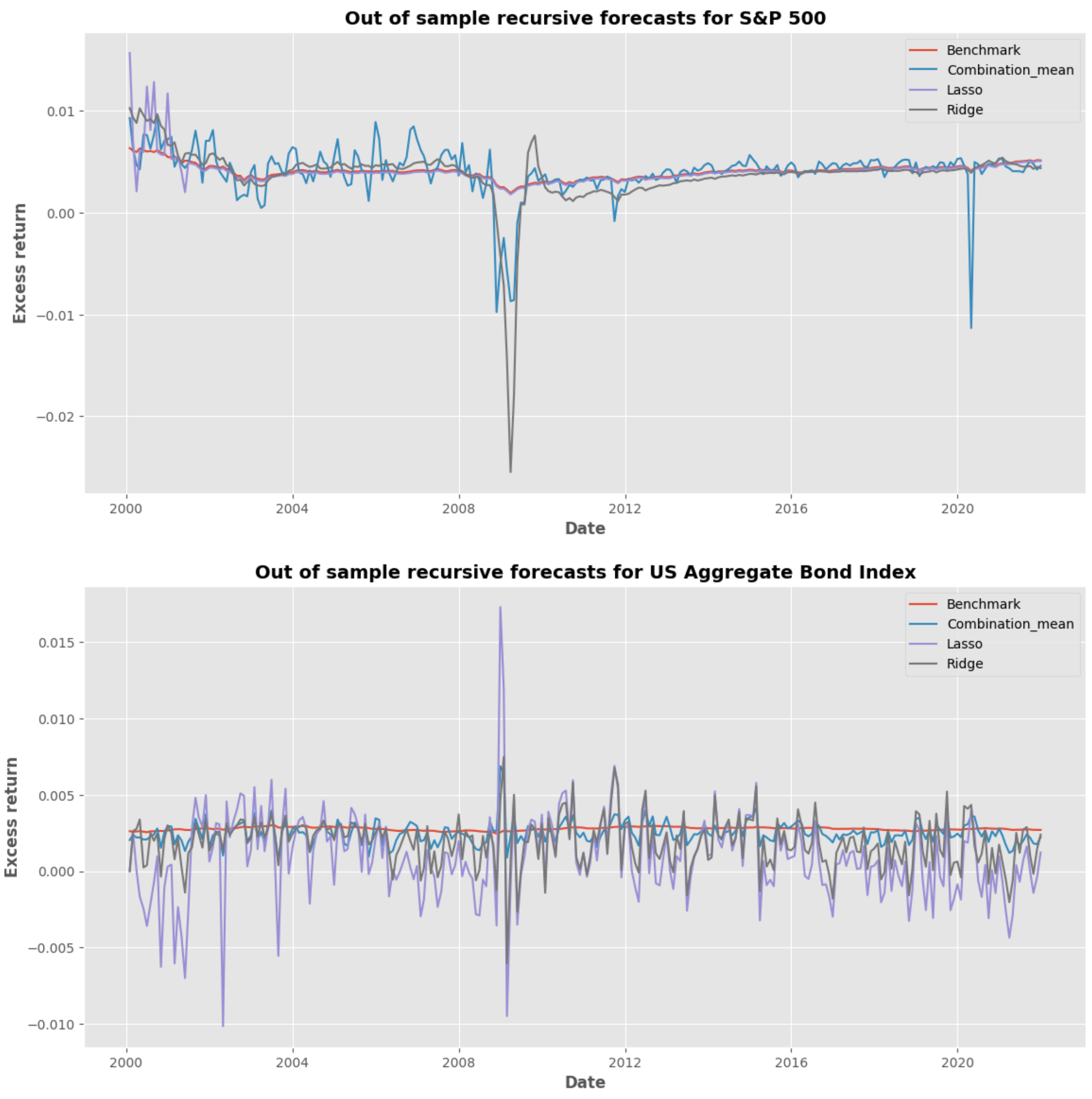


Figure 1: Recursive Forecast for S&P500 and LBUSTRUU

Question 4

The recursive method to generate out-of-sample covariance matrix forecasts for a portfolio. The portfolio consists of two assets, namely stock index and bond index, represented by their excess returns. The covariance matrix forecast provides valuable information about the volatility and the relationship between the returns of the portfolio assets. The generate_portfolio_var_cov_mat_forecast function is responsible for computing these forecasts, utilizing the recursive approach and a specified breakpoint for training and testing data. The resulting forecasts are stored in the portf_cov_mat_forecasts_df data frame. The generated covariance matrix forecasts offer insights into future risk and the interdependencies of the portfolio assets. These insights are crucial for portfolio managers and investors as they assist in asset allocation decisions and risk management.

Question 5

The out-of-sample excess return forecasts and corresponding weights for a portfolio using multiple predictive models in a recursive manner is generated. It creates separate data frames for each model's forecasts and computes the OTP (one-period) excess returns and weights based on these forecasts. The resulting OTP excess return forecasts for all nine models are displayed in the otp_excess_ret_all_models data frame. This code facilitates the evaluation and comparison of the predictive performance of different models for portfolio management.

The summary statistics is calculated for the out-of-sample excess returns of a portfolio across all nine recursive predictive models. Further the statistics measures such as mean, volatility, Sharpe ratio, skewness, and kurtosis are calculated.

Statistic	Benchmark	Pred_1	Pred_2	Pred_3	Pred_4	Pred_5	Comb	PLR_1	PLR_2
Mean	0.0352	0.0427	0.0324	0.0208	0.0355	0.0342	0.0327	0.0284	0.0268
Volatility	0.0004	0.0076	0.0012	0.0070	0.0070	0.0019	0.0023	0.0056	0.0051
Sharpe	90.6522	5.6139	26.0899	2.9658	5.0313	18.1112	13.9734	5.0676	5.2440

Table 5: OTP out-of-sample excess returns summary statistics (all 9 recursive predictive models)

- Mean: The mean excess returns for the benchmark and predictive models range from 2.68% to 4.27%, with Pred_1 having the highest mean return and Pred_3 the lowest.
- Volatility: The benchmark has the lowest volatility (0.04%), while Pred_1 and Pred_4 show relatively higher volatilities compared to other models.
- Sharpe: The Sharpe ratios, measuring risk-adjusted returns, vary significantly. The benchmark has a very high Sharpe ratio of 90.6522, while Pred_2 and Pred_5 also exhibit relatively high ratios. Pred_4 has the lowest Sharpe ratio.

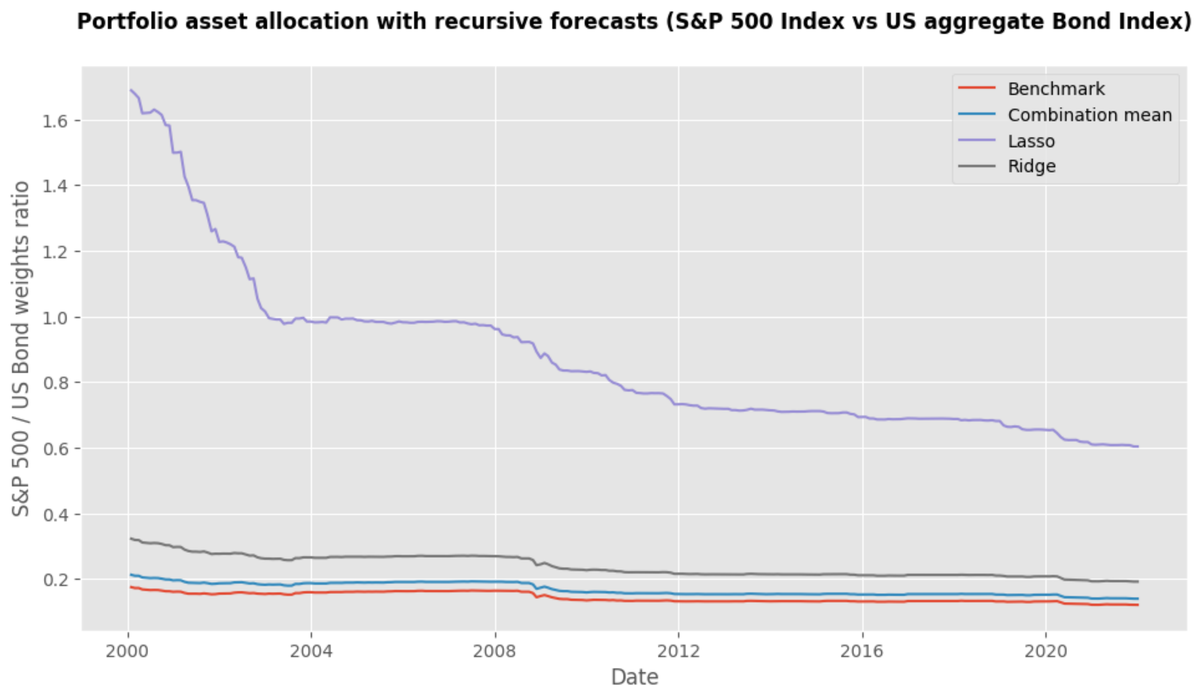


Figure 2: The figure showcases the recursive forecasts for portfolio asset allocation between the S&P500 Index and the US Aggregate Bond Index.

The ratio of asset allocation weights in the two asset class portfolio under the LASSO model indicates higher allocation to stocks from year 2000 and rapidly decreasing to become equally weighted around the year 2003. From there on the allocation to bonds has been higher than stocks. With the other models the asset allocation has been fairly with portfolio being overweighted with bonds.

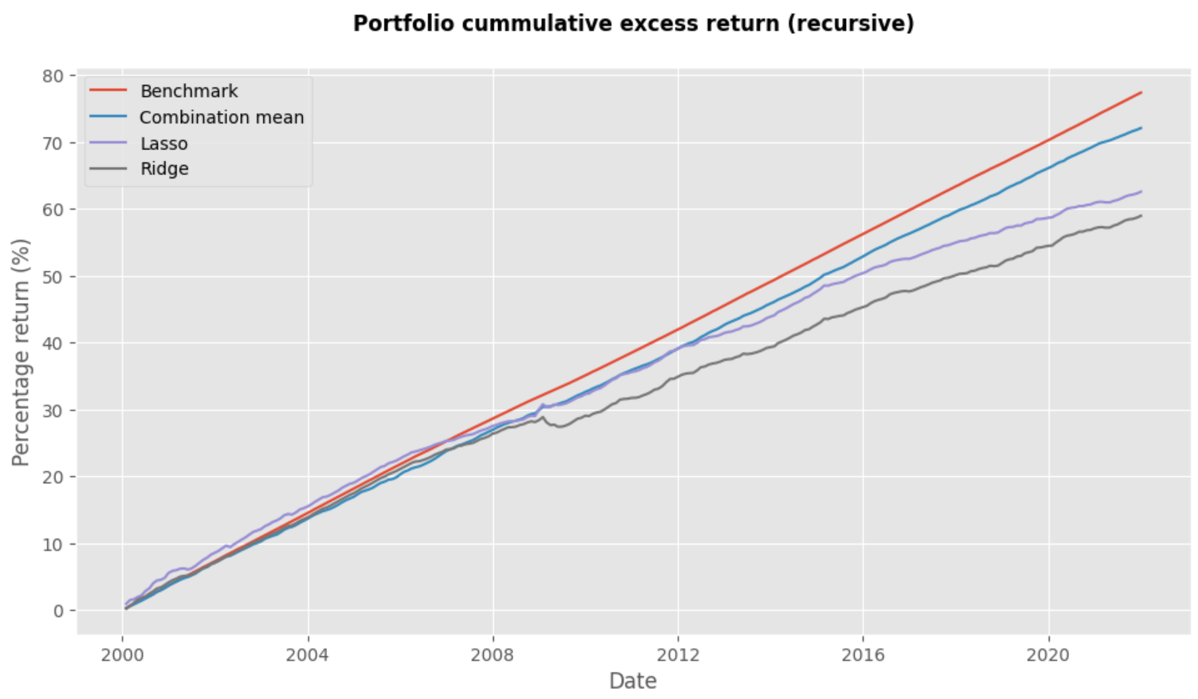


Figure 3: Recursive portfolio cumulative excess return over time.

The cumulative excess return of the portfolio has been below the benchmark, though increasing over the years. The portfolio excess return closely followed the benchmark until the year 2008, but has been below the benchmark thereafter but trending upwards over time.

Part B: Rolling Window Estimate Approach

Rolling estimation, on the other hand, involves training a model on a fixed window of historical data from $y(0)$ to $y(n)$ and predicting the value of $\hat{y}(n+1)$. The model is then retrained using the next window of historical data from $y(1)$ to $y(n+1)$ to predict $\hat{y}(n+2)$, and this process is repeated until the end of the sample period. The size of the historical window used for training the model remains fixed, and only one-step-ahead predictions are made.

Question 1

The summary statistic for stock index and bond index simple excess returns is independent of the recursive or rolling approach. This statistic is derived from the entire dataset, including both in-sample and out-of-sample data. Therefore, the calculation of this statistic is not influenced by the specific method used to process the data, hence is same as calculated before.

Question 2

The mean benchmark's out-of-sample rolling forecasts are calculated using a given window size. It generates the rolling forecasts using a rolling window approach and takes the simple excess returns of the stock index and bond index as input. The results are kept in the variable mean_benchmark_rolling_forecasts and are presented, reflecting the out-of-sample mean benchmark predictions with the selected window size. This analysis provides valuable insights into the performance of the mean benchmark over time.

Question 3

Monthly data for five plausible predictors of the asset class excess returns have been used in the rolling window estimation approach, monthly out-of-sample excess return forecasts have been generated for each of the asset classes as below:

(i) OLS predictive regression model for each of the five predictors
The output shows Out-of-sample OLS Predictor forecasts for each asset class using OLS predictive regression models fitted on the five predictors under rolling window estimation approach.

(ii) Combination forecasts of excess returns that is a simple average of the forecasts based on the five predictors from the OLS model.
The combination mean's out-of-sample rolling forecasts are calculated using a predetermined window size. It makes predictions by combining the stock and bond asset rolling forecasts. The variable combination_mean_rolling_forecasts contains the combination mean rolling forecasts that are produced. The computed rolling forecasts are then displayed after a header describing the projections is printed. This research offers insightful information about the combination mean's performance over time.

(iii) Any 2 penalized linear regressions
Under the rolling window estimation approach, the two types of penalized regression statistical models—LASSO (Least Absolute Shrinkage and Selection Operator) and Ridge are once more applied.

Predictor No.	Variable Name	β_0	β_1	t-stat	p-value	R-Squared
1	Infl	0.0049	0.4563	1.594	0.525	0.002
2	BM	-0.0229	0.0970	1.988	0.048	0.016
3	SVAR	0.0057	-0.0052	-0.013	0.989	0.000
4	NTIS	0.0066	0.1875	1.245	0.214	0.006
5	D/E	0.0076	0.0023	0.376	0.707	0.001

Table 6: Rolling Predictive Models for SP500

Predictor No.	Variable Name	β_0	β_1	t-stat	p-value	R-Squared
1	Infl	0.0034	-0.4075	-2.494	0.013	0.026
2	TBL	0.0032	-0.0469	-1.068	0.287	0.005
3	LTY	0.0004	0.0641	1.337	0.183	0.007
4	LTR	0.0025	0.0296	1.500	0.135	0.009
5	DFY	-0.0015	0.3925	2.686	0.008	0.030

Table 7: Rolling Predictive Models for LBUSTRUU

The results for the rolling window predictive models indicate that most of the predictors are not statistically significant in explaining the returns of stock index and bond index. Only a few predictors show moderate significance, but their explanatory power is limited. Overall, the models have low to very limited explanatory power in predicting the returns of both asset classes.

Mean Squared Forecast Error (MSFE) figures for several predictive models, including the benchmark, for the stock index and bond index using the rolling window technique. Calculations are also made on the ratios of MSFE for each model in comparison to the benchmark. The Diebold-Mariano (DM) test is used to determine whether the models and the benchmark have equivalent predictive power. This analysis enables a comparison of the predictive models' accuracy in predicting returns for the stock index and bond index.

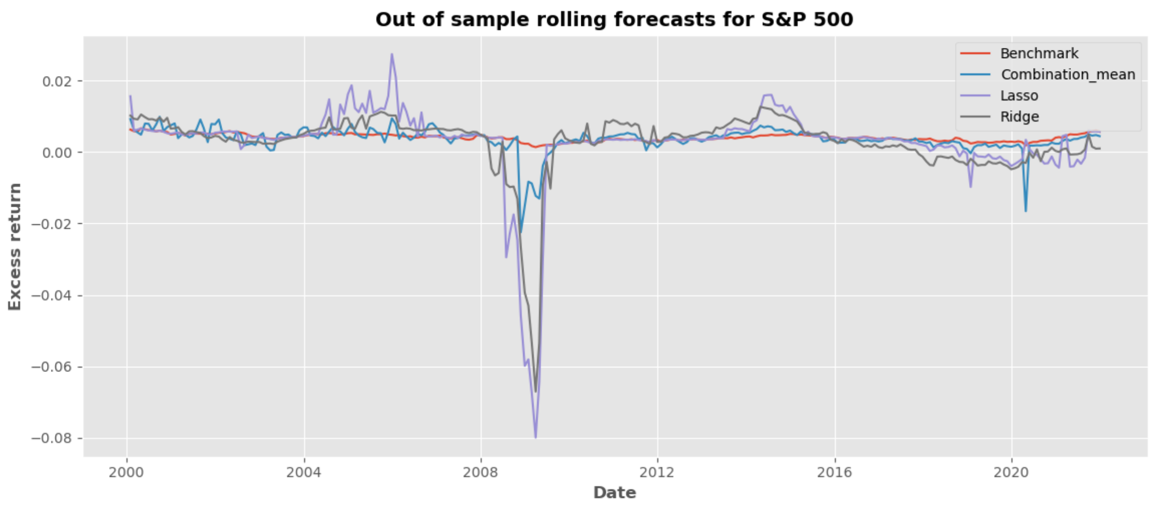
Model(SP500)	MSFE	Ratios of MSFE	DM t-stat	DM p-value
Benchmark	0.001913	1	-	-
Predictor 1	0.001946	1.0172	1.2660	0.2066
Predictor 2	0.001918	1.0027	0.2466	0.8054
Predictor 3	0.002050	1.0718	1.0619	0.2893
Predictor 4	0.001930	1.0093	0.6040	0.5464
Predictor 5	0.001950	1.0196	0.6590	0.5104
Combined	0.001932	1.0102	0.6468	0.5183
PLR_1 (LASSO)	0.001978	1.0343	0.5388	0.5904
PLR_2 (RIDGE)	0.001964	1.0268	0.5346	0.5934
Model (LBUSTRUU)	**MSFE**	**Ratios of MSFE**	**DM t-stat**	**DM p-value**
Benchmark	0.000094	1	-	-
Predictor 1	0.000093	0.9942	-0.1585	0.8742
Predictor 2	0.000097	1.0350	1.8035	0.0725
Predictor 3	0.000098	1.0432	1.6843	0.0933
Predictor 4	0.000095	1.0075	0.3174	0.7512
Predictor 5	0.000094	1.0048	0.1791	0.8580
Combined	0.000094	0.9973	-0.1855	0.8530
PLR_1 (LASSO)	0.000101	1.0749	1.6361	0.1030
PLR_2 (RIDGE)	0.000095	1.0168	0.8638	0.3885

Table 8: Ratios of MSFE for predictive models relative to benchmark for SP500 & LBUSTRUU

The MSFE values and ratios of MSFE for the predictive models, including the benchmark, in predicting stock index returns show varying performance. Some predictors have ratios of MSFE close to 1, indicating similar performance to the benchmark, while others have slightly higher or lower ratios. Overall, the models have relatively similar forecasting accuracy compared to the benchmark.

The results for bond index suggest that most of the predictive models, including the benchmark, do not exhibit statistically significant differences in performance based on the p-values. The MSFE ratios indicate only slight variations in performance compared to the benchmark. Overall, the models' forecasting accuracy is comparable to the benchmark for predicting the returns of bond index.

The two subplots in Figure 4 visualize the out-of-sample rolling forecasts for the stock index (S&P 500) and bond index (US Aggregate Bond Index). The forecasts for selected models, including the benchmark, combination mean, Lasso, and Ridge, are plotted using the corresponding data frames. For stock index, high volatility (dip in returns) is noted around 2008 crisis under all the models, but rest of the return series appear fairly stable, but on the contrary volatility has been noted throughout the return series for bonds. This observation is consistent with the recursive approach.



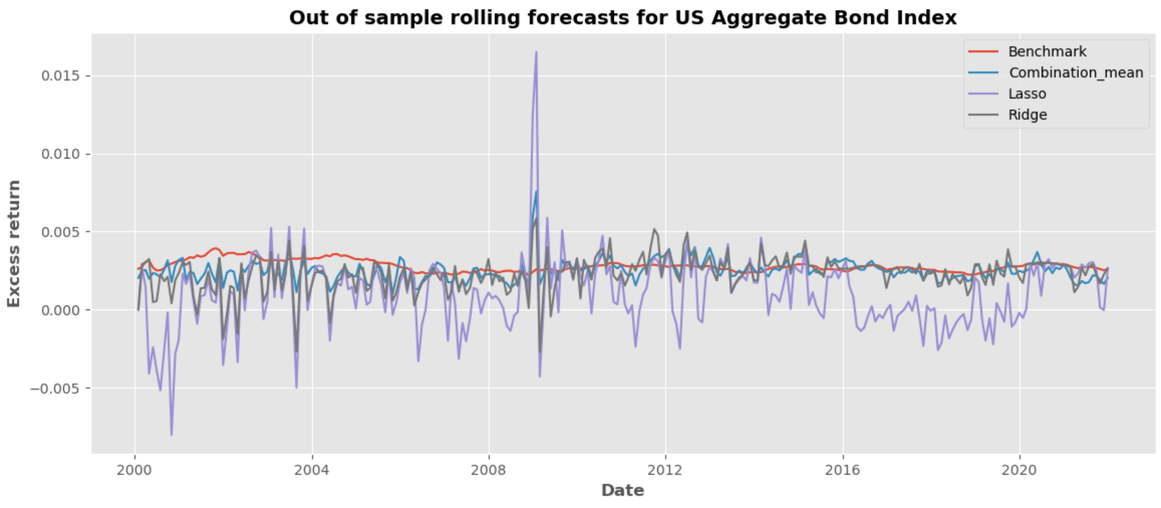


Figure 4: Rolling Forecast for S&P500 and LBUSTRUU

Question 4

The out-of-sample rolling forecasts for the variance-covariance matrix of a portfolio is calculated. It uses a rolling window approach to compute these forecasts, taking into account the simple excess returns of the stock index and bond index. The resulting forecasts capture the changing risk and interdependencies among the portfolio assets over time. This analysis provides valuable insights into the dynamic nature of the portfolio's risk characteristics and aids in portfolio management and risk assessment.

Question 5

The code computes out-of-sample excess returns for multiple rolling predictive models. It generates individual data frames to store the forecasts for each model by combining the rolling forecasts of the stock index and bond index. The code then calculates the monthly out-of-sample OTP excess returns and weights for each model using the forecasts and the rolling variance-covariance matrix. The results provide valuable insights into the performance of the rolling predictive models.

The summary statistics for the out-of-sample excess returns of a portfolio using rolling predictive models is calculated. These statistics include measures such as mean, volatility, Sharpe ratio, skewness, and kurtosis.

Statistic	Benchmark	Pred_1	Pred_2	Pred_3	Pred_4	Pred_5	Comb	PLR_1	PLR_2
Mean	0.0347	0.0369	0.0329	0.0203	0.0323	0.0356	0.0311	0.0160	0.0275
Volatility	0.0013	0.0069	0.0038	0.0058	0.0053	0.0040	0.0023	0.0077	0.0041
Sharpe	26.9846	5.3844	8.5905	3.5051	6.0617	8.8331	13.7661	2.0799	6.7567

Table 9: OTP out-of-sample excess returns summary statistics (all 9 rolling predictive models)

- Mean: The benchmark has an average excess return of 3.47%. The other models have varying mean excess returns, with Pred_1 and Pred_5 exhibiting relatively higher returns compared to the benchmark.
- Volatility: The benchmark has a volatility of 0.13%. while other models show relatively higher volatilities.
- Sharpe: The Sharpe ratio is highest for the benchmark.

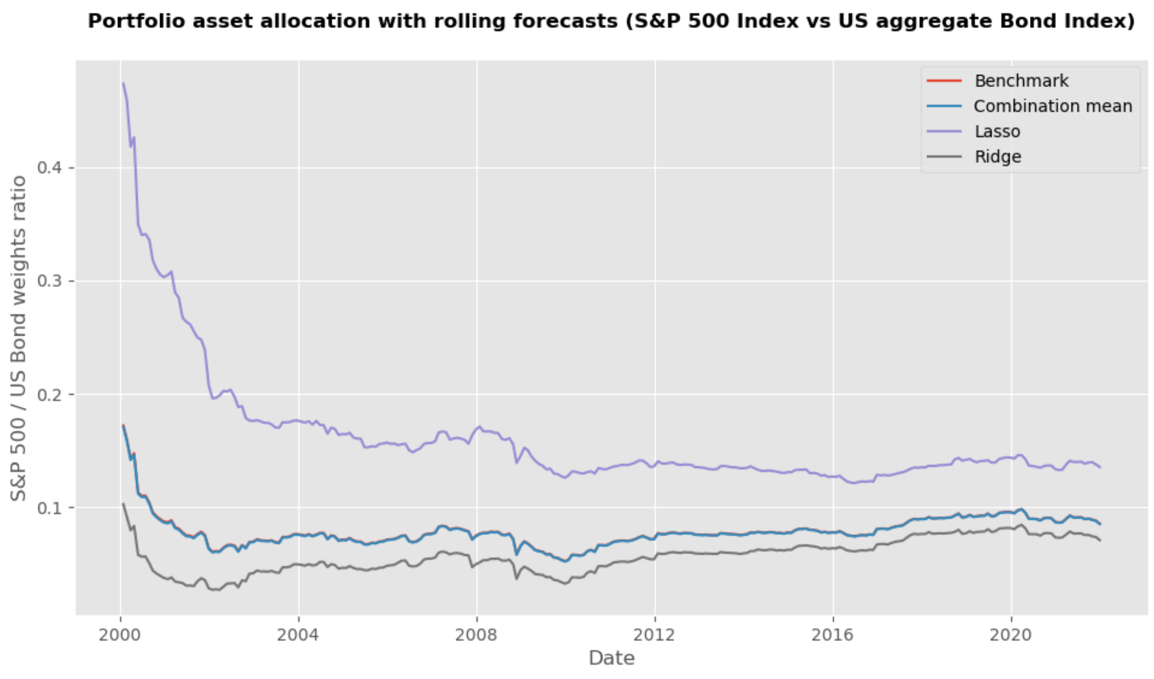


Figure 5: The figure showcases the rolling forecasts for portfolio asset allocation between the S&P500 Index and the US Aggregate Bond Index.

The ratio of asset allocation weights in the two asset class portfolio under the LASSO model, using rolling window estimation, shows a similar pattern to the recursive model, however the allocation to bonds have been higher than stocks since the ratio is less than 1. The allocation under the combination and benchmark models mirror each other.

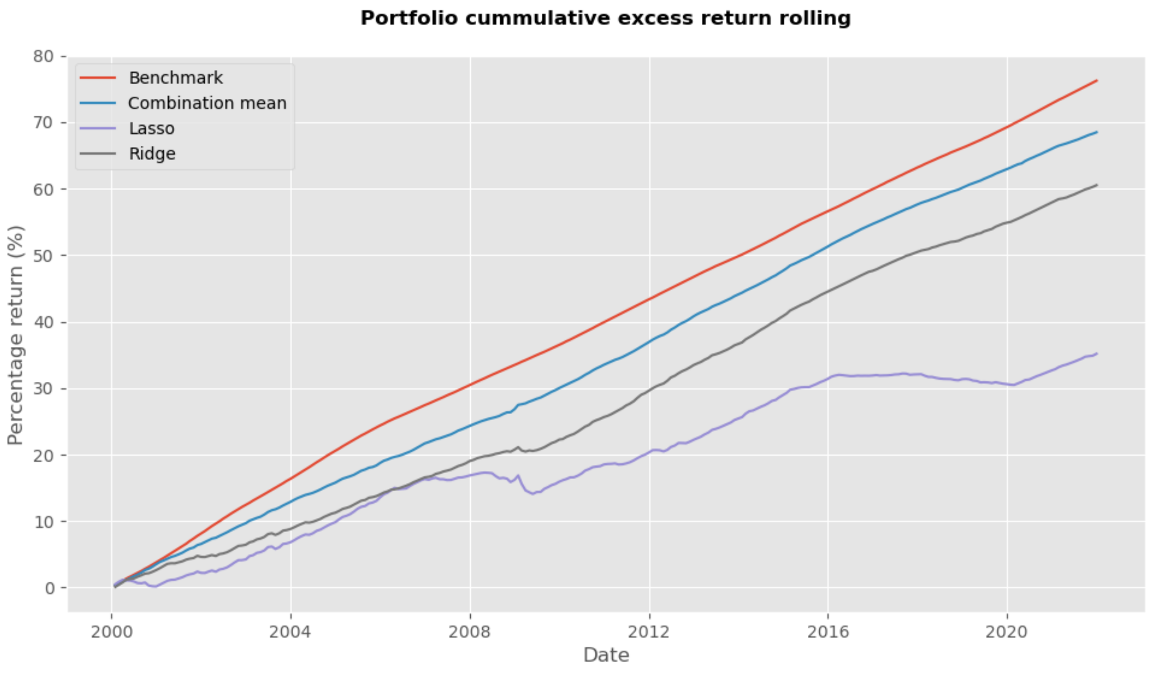


Figure 6: Rolling portfolio cumulative excess return over time.

Under the rolling window estimation approach, the cumulative excess return of the portfolio has been below the benchmark, though increasing over the years in the rolling. The magnitude of the upward movement varies across models with LASSO indicating lower returns over time. The returns have been more dispersed under rolling window estimation approach then recursive approach.

Conclusion

In conclusion, the asset allocation weights in the two asset class portfolio under the LASSO model shift from stocks to bonds around the early 2000s. The portfolio's cumulative excess return has been below the benchmark but increasing over time. The performance of the portfolio varies across models, with the LASSO model indicating lower returns. Further refinement of the models and asset allocation strategies is needed to improve portfolio performance and potentially outperform the benchmark.

References

[1] Diebold, F. X., Mariano, R. S., 1995. Comparing predictive accuracy. Journal of Business & economic statistics 13, 253–263.

[2] Lin, H., Wu, C., Zhou, G., 2017. Forecasting corporate bond returns with a large set of predictors: An iterated combination approach. Management Science .

[3] Rapach, D. E., Ringgenberg, M. C., Zhou, G., 2016. Short interest and aggregate stock returns. Journal of Financial Economics 121, 46–65.

Group 5: Contributions

Name and Surname	Student Number	Type of Contribution	Contribution out of 100%
Kuntoji, Rohan	22202090	10% of Research; 60% of implementing functions; 35% of Code modularisation;	26.25%
Shraddha, Nagar	22200477	45% of Research; 40% of implementing functions;	21.25%
Fernandes, William	22201553	45% of Research; 40% of report writing;	21.25%
Pant, Vidushi	22200808	40% of report writing	10%
Aman, Parary	18417714	65% of Code modularisation; 20% of report writing	21.25%

Python Code

```python
import datetime as dt
import pandas as pd
import numpy as np
from scipy import stats
from scipy.stats import t
import statsmodels.api as sm
import collections
from sklearn.metrics import mean_squared_error
import argparse

from utils import *

import matplotlib.pyplot as plt
plt.style.use('ggplot') # fivethirtyeight, ggplot, dark_background, classic,
import warnings
warnings.filterwarnings("ignore")


class ForecastingAnalysis:
    """
    This class examines various statistical time series forecasting methods using historical
    financial markets data.
    """

    def __init__(self, sp500_full_df, bond_full_df, rf_ret_df):
        """
        Initialize the ForecastingAnalysis object with S&P 500, bond, and risk-free rate data.

        Parameters
        ----------
        sp500_full_df : pd.DataFrame
            Monthly S&P 500 stock index price data from Dec 1979 to Dec 2021.
        bond_full_df : pd.DataFrame
            Monthly Bloomberg Barclays U.S. Aggregate Bond Index price data from
            Dec 1979 to Dec 2021.
        rf_ret_df : pd.DataFrame
            Monthly risk-free rate data from Jan 1980 to Dec 2021.
        """
        self.sp500_full_df = sp500_full_df
        self.bond_full_df = bond_full_df
        self.rf_ret_df = rf_ret_df

    def question2(self, simp_ret_df, breakpoint, method, window_size=None):
        """
        Generate mean benchmark forecasts for the given data and parameters.

        Parameters
        ----------
        simp_ret_df : pd.DataFrame
            A DataFrame containing the simple excess returns data for two assets:
            'SP500_excess' and 'LBUSTRUU_excess'
        breakpoint : dt.datetime
            The breakpoint used for splitting the data into training and testing sets representing
            a date in the format
        method : str
            The method used for generating the benchmark forecast.
            Supported methods: 'recursive' or 'rolling'.
        window_size : int, optional
            The size of the rolling window for calculating rolling method.
            This parameter is required if the method is 'rolling'.
            Default is None.

        Returns
        -------
        mean_benchmark_forecasts_df : pd.DataFrame
            A DataFrame containing the mean benchmark forecasts for the given data and parameters.
        """
        mean_benchmark_forecasts_df = generate_mean_benchmark_forecast(simp_ret_df.loc[:,
        ['SP500_excess', 'LBUSTRUU_excess']], breakpoint, method=method, window_size=window_size if
        window_size is not None else None)
        return mean_benchmark_forecasts_df

    def question3(self, simp_ret_df, predictors_df, breakpoint, asset_tickers,
    mean_benchmark_forecasts_df, method, window_size=None):

        """
        Generate various forecasts, compute Mean Squared Forecast Errors (MSFE),
        and perform Diebold-Mariano (DM) tests for equal predictive ability.

        Paramters
        ---------
        simp_ret_df : pd.DataFrame
            A DataFrame containing the simple excess returns data for two assets:
            'SP500_excess' and 'LBUSTRUU_excess'
        predictors_df: pd.DataFrame
            A DataFrame containing the predictor variables.
        breakpoint : dt.datetime
            The breakpoint used for splitting the data into training and testing sets representing
            a date in the format
        asset_tickers: list
            A list of asset tickers, e.g. ['SP500', 'LBUSTRUU'].
        mean_benchmark_forecasts_df: pd.DataFrame
            A DataFrame containing the mean benchmark forecasts for the assets.
        method : str
            The method used for generating the benchmark forecast.
            Supported methods: 'recursive' or 'rolling'.
        window_size : int, optional
            The size of the rolling window for calculating rolling method.
            This parameter is required if the method is 'rolling'.
            Default is None.

        Returns
        -------
        A tuple containing the following DataFrames:
        1) ols_predictor_forecasts_df: Forecasts for each asset using OLS fit on predictors.
        2) combination_mean_forecasts_df: Forecasts for each asset
        using the combination mean method.
        3) plr_predictor_forecasts_df: Forecasts for each asset using PLR fit on predictors.
        4) msfe_df: Mean Squared Forecast Errors for all predictive models and corresponding
        benchmarks for both assets.
        5) msfe_ratios_to_benchmark: Ratios of MSFEs of predictive models
        to corresponding benchmark MSFE values.
        6) dm_test_stats: Diebold-Mariano test statistics and p-values for checking the equal
        predictive ability of all predictive models with the benchmark.
```

```
7) sp500_all_models: DataFrame containing all forecasts for S&P 500.
8) bond_all_models: DataFrame containing all forecasts for US Aggregate Bond Index.

"""

# SP500 predictors
sp500_predictors = predictors_df.loc[:, ['infl', 'b/m', 'svar', 'ntis']]
sp500_predictors['div_payout'] = np.log(predictors_df['D12']/predictors_df['E12'])

# Bond predictors
bond_predictors = predictors_df.loc[:, ['infl', 'tbl', 'lty', 'ltr']]
bond_predictors['DFY'] = predictors_df['BAA'] - predictors_df['AAA']

# out-of-sample return forecasts for SP500 predicted using
# OLS fit on 5 different predictors
sp500_ols_forecasts_df = generate_ols_predictor_forecast(simp_ret_df.loc[:,
['SP500_excess']], sp500_predictors,breakpoint=breakpoint, method=method,
window_size=window_size if window_size is not None else None)

# out-of-sample return forecasts for bonds predicted using OLS fit
# on 5 different predictors
bond_ols_forecasts_df = generate_ols_predictor_forecast(simp_ret_df.loc[:,
['LBUSTRUU_excess']], bond_predictors,breakpoint=breakpoint, method=method,
window_size=window_size if window_size is not None else None)

# Rename the columns in each data frame
sp500_ols_forecasts_df.columns = pd.MultiIndex.from_product([[asset_tickers[0]],
sp500_ols_forecasts_df.columns])
bond_ols_forecasts_df.columns = pd.MultiIndex.from_product([[asset_tickers[1]],
bond_ols_forecasts_df.columns])
# Concatenate the data frames along the columns axis and create a multi-level column index
ols_predictor_forecasts_df = pd.concat([sp500_ols_forecasts_df, bond_ols_forecasts_df],
axis=1)


# Compute Combination mean forecasts
combination_mean_forecasts_df = generate_combination_mean_forecasts(sp500_ols_forecasts_df,
bond_ols_forecasts_df, assets_names=asset_tickers)

# out-of-sample return forecasts for SP500 predicted using PLR fit on predictors
sp500_plr_forecasts_df = generate_plr_forecast(simp_ret_df.loc[:, ['SP500_excess']],
sp500_predictors,breakpoint=breakpoint, method=method,
window_size=window_size if window_size is not None else None)

# out-of-sample return forecasts for bonds predicted using PLR fit on predictors
bond_plr_forecasts_df = generate_plr_forecast(simp_ret_df.loc[:, ['LBUSTRUU_excess']],
bond_predictors,breakpoint=breakpoint, method=method,
window_size=window_size if window_size is not None else None)

# Rename the columns in each data frame
sp500_plr_forecasts_df.columns = pd.MultiIndex.from_product([[asset_tickers[0]],
sp500_plr_forecasts_df.columns])
bond_plr_forecasts_df.columns = pd.MultiIndex.from_product([[asset_tickers[1]],
bond_plr_forecasts_df.columns])
# Concatenate the data frames along the columns axis and create a multi-level column index
plr_predictor_forecasts_df = pd.concat([sp500_plr_forecasts_df, bond_plr_forecasts_df],
axis=1)
```

```
# combine benchamark & all 8 forecast model predictions for each asset class
sp500_all_models = pd.concat([mean_benchmark_forecasts_df.loc[:, ['SP500_MB']],
sp500_ols_forecasts_df['SP500'], combination_mean_forecasts_df.loc[:, ['SP500_Comb_Mean']],
sp500_plr_forecasts_df['SP500']], axis=1)

sp500_all_models.rename(columns={'SP500_MB': 'Benchmark', 'SP500_Comb_Mean':
'Combination_mean'}, inplace=True)
bond_all_models = pd.concat([mean_benchmark_forecasts_df.loc[:, ['LBUSTRUU_MB']],
bond_ols_forecasts_df['LBUSTRUU'], combination_mean_forecasts_df.loc[:,
['LBUSTRUU_Comb_Mean']], bond_plr_forecasts_df['LBUSTRUU']], axis=1)

bond_all_models.rename(columns={'LBUSTRUU_MB': 'Benchmark', 'LBUSTRUU_Comb_Mean':
'Combination_mean'}, inplace=True)

# Compute MSFE for all predictive models & corresponding benchmarks for both the assets
msfe_df = pd.DataFrame(index=asset_tickers, columns=['Benchmark', 'OLS_1', 'OLS_2', 'OLS_3',
'OLS_4', 'OLS_5', 'CM', 'PLR_1', 'PLR_2'])
msfe_df.loc[asset_tickers[0]] = [mean_squared_error(simp_ret_df.loc[simp_ret_df.index >
breakpoint, ['SP500_excess']], sp500_all_models[x])
                        for x in sp500_all_models.columns]
msfe_df.loc[asset_tickers[1]] = [mean_squared_error(simp_ret_df.loc[simp_ret_df.index >
breakpoint, ['LBUSTRUU_excess']], bond_all_models[x])
                        for x in bond_all_models.columns]

# Compute the ratios of MSFEs of predictive models to corresponding benchmark MSFE values
msfe_ratios_to_benchmark = pd.DataFrame(index=asset_tickers, columns=['OLS_1', 'OLS_2',
'OLS_3', 'OLS_4', 'OLS_5', 'CM', 'PLR_1', 'PLR_2'])
msfe_ratios_to_benchmark.loc[asset_tickers[0]] = [msfe_df.loc[asset_tickers[0],
[x]].values[0] / msfe_df.loc[asset_tickers[0], ['Benchmark']].values[0] for x in
msfe_df.columns[1:]]
msfe_ratios_to_benchmark.loc[asset_tickers[1]] = [msfe_df.loc[asset_tickers[1],
[x]].values[0] / msfe_df.loc[asset_tickers[1], ['Benchmark']].values[0] for x in
msfe_df.columns[1:]]

# Check for equal predictive ability of all 8 predictive models with the
# benchmark using DM test
dm_test_stats = pd.DataFrame(index=pd.MultiIndex.from_product([asset_tickers, ['dm_tstat',
'p_val']]), columns=['OLS_1', 'OLS_2', 'OLS_3', 'OLS_4', 'OLS_5', 'CM', 'PLR_1', 'PLR_2'])
dm_test_stats.loc[(asset_tickers[0], 'dm_tstat')] =
[dm_test(real_values=simp_ret_df.loc[simp_ret_df.index > breakpoint,
'SP500_excess'].values,pred1=sp500_all_models[col].values,
pred2=sp500_all_models['Benchmark'].values)[0]for col in sp500_all_models.columns[1:]]
dm_test_stats.loc[(asset_tickers[0], 'p_val')] =
[dm_test(real_values=simp_ret_df.loc[simp_ret_df.index > breakpoint,
'SP500_excess'].values,pred1=sp500_all_models[col].values,
pred2=sp500_all_models['Benchmark'].values)[1] for col in sp500_all_models.columns[1:]]
dm_test_stats.loc[(asset_tickers[1], 'dm_tstat')] =
[dm_test(real_values=simp_ret_df.loc[simp_ret_df.index > breakpoint,
'LBUSTRUU_excess'].values,pred1=bond_all_models[col].values,
pred2=bond_all_models['Benchmark'].values)[0] for col in bond_all_models.columns[1:]]
dm_test_stats.loc[(asset_tickers[1], 'p_val')] =
[dm_test(real_values=simp_ret_df.loc[simp_ret_df.index > breakpoint,
'LBUSTRUU_excess'].values, pred1=bond_all_models[col].values,
pred2=bond_all_models['Benchmark'].values)[1] for col in bond_all_models.columns[1:]]

return ols_predictor_forecasts_df, combination_mean_forecasts_df,
plr_predictor_forecasts_df, msfe_df, msfe_ratios_to_benchmark,
dm_test_stats, sp500_all_models, bond_all_models
```

```
def plot_forecast(self, sp500_all_models, bond_all_models, method):

    """
    Plot the out-of-sample forecasts for S&P 500 and US Aggregate Bond Index.

    Parameters
    ----------
    sp500_all_models : pd.DataFrame
        A DataFrame containing the out-of-sample forecasts for the S&P 500.
    bond_all_models : pd.DataFrame
        A DataFrame containing the out-of-sample forecasts for the US Aggregate Bond Index.
    method : str
        The method used for generating the benchmark forecast.
        Supported methods: 'recursive' or 'rolling'.

    Returns
    -------
    This function returns a plot showing out-of-sample forecasts for S&P 500 and US Aggregate
    Bond Index.
    """

    plt.figure(figsize=(12, 12))
    desired_models = ['Benchmark', 'Combination_mean', 'Lasso', 'Ridge']

    plt.subplot(2, 1, 1)
    plt.plot(sp500_all_models.index, sp500_all_models.loc[:, desired_models].values)
    plt.title(f'Out of sample {method} forecasts for S&P 500',
    fontweight='bold', fontsize=14)
    plt.xlabel('Date', weight='bold', fontsize=12)
    plt.ylabel('Excess return', weight='bold', fontsize=12)
    plt.legend(desired_models)

    plt.subplot(2, 1, 2)
    plt.plot(bond_all_models.index, bond_all_models.loc[:, desired_models].values)
    plt.title(f'Out of sample {method} forecasts for US Aggregate Bond Index', fontweight='bold'
    fontsize=14)
    plt.xlabel('Date', weight='bold', fontsize=12)
    plt.ylabel('Excess return', weight='bold', fontsize=12)
    plt.legend(desired_models)

    plt.tight_layout(pad=2.0)
    plt.show()

def question4(self, simp_ret_df, breakpoint, method, window_size=None):
    """
    Generates a timeseries of monthly out-of-sample variance-covariance
    matrix forecasts for a portfolio of assets.

    Parameters
    ----------
    simp_ret_df : pd.DataFrame
        A DataFrame containing the simple excess returns data for two assets:
        'SP500_excess' and 'LBUSTRUU_excess'
    breakpoint : dt.datetime
        The breakpoint used for splitting the data into training and testing sets representing
        a date in the format
    method : str
```

```
            The method used for generating the benchmark forecast.
            Supported methods: 'recursive' or 'rolling'.
        window_size : int, optional
            The size of the rolling window for calculating rolling method.
            This parameter is required if the method is 'rolling'.
            Default is None.

        Returns
        -------
        portf_cov_mat_forecasts_df : pd.DataFrame
            Monthly out-of-sample forecast of variance-covariance matrices
            for a portfolio of assets.
        """

        portf_cov_mat_forecasts_df = generate_portfolio_var_cov_mat_forecast(simp_ret_df.loc[:,
        ['SP500_excess', 'LBUSTRUU_excess']], breakpoint=breakpoint, method=method,
        window_size=window_size if window_size is not None else None)
        return portf_cov_mat_forecasts_df

    def question5(self, sp500_all_models, bond_all_models, portf_cov_mat_forecasts_df):

        """
        Generate OTP excess returns, summary statistics, and out-of-sample weights for various
        forecast models.

        Parameters
        ----------
        sp500_all_models : pd.DataFrame
            A DataFrame containing all forecasts for S&P 500.
        bond_all_models : pd.DataFrame
            A DataFrame containing all forecasts for US Aggregate Bond Index.
        portf_cov_mat_forecasts_df : pd.DataFrame
            A DataFrame containing the portfolio covariance matrix forecasts.

        Returns
        -------
        A tuple containing the following DataFrames:
        1) otp_excess_ret_all_models: A DataFrame with the OTP excess
        returns for each forecast model.
        2) portf_excess_ret_summary_stats: A DataFrame with the summary
        statistics of the portfolio excess returns for each forecast model.
        3) otp_oos_weights_all_models: A DataFrame with the out-of-sample
        weights for each forecast model.
        """


        # placeholder for storing list of individual DFs with asset return forecasts
        # computed using all the models
        temp_list = [pd.concat([sp500_all_models.iloc[:, i], bond_all_models.iloc[:, i]], axis=1)
        for i in range(len(sp500_all_models.columns))]
        # placeholder for OTP excess return (for all model forecasts)
        otp_excess_ret_all_models = pd.DataFrame(index=sp500_all_models.index,
                                                 columns=['Benchmark', 'OLS_1', 'OLS_2', 'OLS_3',
                                                 'OLS_4', 'OLS_5', 'CM', 'PLR_1', 'PLR_2'])
        # placeholder for OTP out-of-sample weights (for all model forecasts)
        otp_oos_weights_all_models = otp_excess_ret_all_models.copy()
        for i in range(len(temp_list)):
            # compute time-series of monthly OTP excess return for all the forecast models
            otp_excess_ret_all_models.iloc[:, i], otp_oos_weights_all_models.iloc[:, i] =
            generate_OTP_excess_ret(temp_list[i], portf_cov_mat_forecasts_df)

        # Compute Portfolio excess return summary statistics
        portf_excess_ret_summary_stats = pd.DataFrame(index=['Mean', 'Volatility', 'Sharpe', 'Skew',
        'Kurtosis'], columns=otp_excess_ret_all_models.columns)
        for i in range(len(portf_excess_ret_summary_stats.columns)):
            portf_excess_ret_summary_stats.iloc[:, i] =
            compute_stat_measures(otp_excess_ret_all_models.iloc[:, i].values)

        return otp_excess_ret_all_models, portf_excess_ret_summary_stats,
        otp_oos_weights_all_models

    def plot_portfolio_asset_allocation(self, otp_oos_weights_all_models, method):
        """
        Plot the portfolio asset allocation for various forecast models.

        Parameters
        ----------
        otp_oos_weights_all_models : pd.DataFrame
            A DataFrame containing the out-of-sample weights for each forecast model.
        method : str
            The method used for generating the benchmark forecast.
            Supported methods: 'recursive' or 'rolling'.

        Returns
        -------
        This function returns a plot showing the asset allocation with the specified method's
        forecasts.

        """
        # Plot portfolio asset allocation for various forecast models
        plt.figure(figsize=(10, 6))
        plt.title(f"Portfolio asset allocation with {method} forecasts '
        f'(S&P 500 Index vs US aggregate Bond Index)\n", fontweight='bold', fontsize=12)
        plt.plot(otp_oos_weights_all_models.index, [x[0]/x[1] for x in
        otp_oos_weights_all_models['Benchmark'].values], label='Benchmark')
        plt.plot(otp_oos_weights_all_models.index, [x[0]/x[1] for x in
        otp_oos_weights_all_models['CM'].values], label='Combination mean')
        plt.plot(otp_oos_weights_all_models.index, [x[0]/x[1] for x in
        otp_oos_weights_all_models['PLR_1'].values], label='Lasso')
        plt.plot(otp_oos_weights_all_models.index, [x[0]/x[1] for x in
        otp_oos_weights_all_models['PLR_2'].values], label='Ridge')
        plt.ylabel("S&P 500 / US Bond weights ratio")
        plt.xlabel("Date")
        plt.tight_layout(pad=2.0)
        plt.legend(loc='best')
        plt.show()

    def plot_portfolio_returns(self, otp_excess_ret_all_models, method):
        """
        Plot the portfolio excess returns for various forecast models.

        Parameters
        ----------
        otp_excess_ret_all_models : pd.DataFrame
            A DataFrame containing the excess returns for each forecast model.
        method : str
```

```
            The method used for generating the benchmark forecast.
            Supported methods: 'recursive' or 'rolling'.

        Returns
        -------
        This function returns a plot showing the portfolio
        excess returns for various forecast models.

        """
        # Plot portfolio daily and cummulative returns for various forecast models
        plt.figure(figsize=(10, 6))
        plt.title(f"Portfolio cummulative excess return {method}\n", fontweight='bold', fontsize=12)
        plt.plot(otp_excess_ret_all_models.index, otp_excess_ret_all_models['Benchmark'].cumsum() *
        100, label='Benchmark')
        plt.plot(otp_excess_ret_all_models.index, otp_excess_ret_all_models['CM'].cumsum() * 100,
        label='Combination mean')
        plt.plot(otp_excess_ret_all_models.index, otp_excess_ret_all_models['PLR_1'].cumsum() * 100,
        label='Lasso')
        plt.plot(otp_excess_ret_all_models.index, otp_excess_ret_all_models['PLR_2'].cumsum() * 100,
        label='Ridge')
        plt.ylabel("Percentage return (%)")
        plt.xlabel("Date")
        plt.tight_layout(pad=2.0)
        plt.legend(loc='best')
        plt.show()

def main(method, rolling_window_size):

    """
    Perform forecasting analysis for S&P 500 and US Aggregate Bond index using various predictive
    models.

    Parameters
    ----------
    method : str
        The method to use for forecasting analysis: either 'recursive' or 'rolling'.
    rolling_window_size : int
        The size of the rolling window for the rolling method. Ignored if method is 'recursive'.
    """

    # Load data
    sp500_full_df, bond_full_df, rf_ret_df, predictors_df = get_market_data()
    breakpoint = dt.datetime(2000, 1, 1)
    asset_tickers = ['SP500', 'LBUSTRUU']
    simp_ret_df = generate_asset_simple_rets(sp500_full_df, bond_full_df, rf_ret_df)

    # Create instance of ForecastingAnalysis class
    forecasting_analysis = ForecastingAnalysis(sp500_full_df, bond_full_df, rf_ret_df)


    ##---Question 1---##
    sp_mean, sp_vol, sp_sharpe, sp_skew, sp_kurt =
    compute_stat_measures(simp_ret_df['SP500_excess'].values)
    bond_mean, bond_vol, bond_sharpe, bond_skew, bond_kurt =
    compute_stat_measures(simp_ret_df['LBUSTRUU_excess'].values)

    asset_ret_summary_stats = pd.DataFrame({'SP500': [sp_mean, sp_vol, sp_sharpe, sp_skew, sp_kurt],
                                            'LBUSTRUU': [bond_mean, bond_vol, bond_sharpe,
                                            bond_skew,bond_kurt]},
                                            index=['Mean', 'Volatility', 'Sharpe', 'Skew',
                                            'Kurtosis'])
    print('\nAsset class excess return summary statistics (total time period)')
    print('-' * 100)
    print(asset_ret_summary_stats)
    ##---End of Question 1---##

    if method == 'recursive':
        ##---Question 2---##
        mean_benchmark_forecasts_df = forecasting_analysis.question2(simp_ret_df, breakpoint,
        method=method)
        print(f'\nOut-of-sample Mean Benchmark {method} forecasts')
        print('-' * 100)
        print(mean_benchmark_forecasts_df)
        ##---End of Question 2---##

        ##---Question 3---##
        ols_predictor_forecasts_df, combination_mean_forecasts_df, plr_predictor_forecasts_df,
        msfe_df, msfe_ratios_to_benchmark, dm_test_stats, sp500_all_models, bond_all_models =
        forecasting_analysis.question3(simp_ret_df, predictors_df, breakpoint, asset_tickers,
        mean_benchmark_forecasts_df, method)
        print(f'\nOut-of-sample OLS Predictor {method} forecasts')
        print('-' * 100)
        print(ols_predictor_forecasts_df)
        print(f'\nOut-of-sample Combination Mean {method} forecasts')
        print('-' * 100)
        print(combination_mean_forecasts_df)
        print(f'\nOut-of-sample Penalised Linear Regression {method} forecasts (using all'
        f'Predictors)')
        print('-' * 100)
        print(plr_predictor_forecasts_df)
        print(f'\nMSFE values for all 9 {method} predictive models including benchmark (both asset'
        f'classes)')
        print('-' * 100)
        print(msfe_df)
        print(f'\nRatios of MSFE values of all 8 {method} predictive models to corresponding'
        f'benchmark forecasts MSFEs (both asset classes)')
        print('-' * 100)
        print(msfe_ratios_to_benchmark)
        print(f'\nDM test to check for equal predictive ability relative to mean'
        f' benchmark forecasts (all 8 {method} predictive models for both asset classes)')
        print('-' * 100)
        print(dm_test_stats)
        forecasting_analysis.plot_forecast(sp500_all_models, bond_all_models, method)
        ##---End of Question 3---##

        ##---Question 4---##
        portf_cov_mat_forecasts_df = forecasting_analysis.question4(simp_ret_df, breakpoint, method)
        print(f'\nOut-of-sample variance-covariance matrix {method} forecasts')
        print('-' * 100)
        print(portf_cov_mat_forecasts_df)
        ##---End of Question 4---##


        ##---Question 5---##
        otp_excess_ret_all_models, portf_excess_ret_summary_stats, otp_oos_weights_all_models =
        forecasting_analysis.question5(sp500_all_models, bond_all_models,
        portf_cov_mat_forecasts_df)
        print(f'\nOTP out-of-sample excess returns (all 9 {method} predictive models)')
        print('-' * 100)
        print(otp_excess_ret_all_models)
        print(f'\nOTP out-of-sample excess returns summary statistics (all 9 {method} predictive'
        f'models)')
        print('-' * 100)
        print(portf_excess_ret_summary_stats)
        forecasting_analysis.plot_portfolio_asset_allocation(otp_oos_weights_all_models, method)
        forecasting_analysis.plot_portfolio_returns(otp_excess_ret_all_models, method)
        ##---End of Question 5---##

    else:

        ##---Question 6---##
        mean_benchmark_forecasts_df = forecasting_analysis.question2(simp_ret_df, breakpoint,
        method=method, window_size=rolling_window_size)
        print(f'\nOut-of-sample Mean Benchmark {method} forecasts'
        f'window size {rolling_window_size}')
        print('-' * 100)
        print(mean_benchmark_forecasts_df)

        ols_predictor_forecasts_df, combination_mean_forecasts_df, plr_predictor_forecasts_df,
        msfe_df, msfe_ratios_to_benchmark, dm_test_stats, sp500_all_models, bond_all_models =
        forecasting_analysis.question3(simp_ret_df, predictors_df, breakpoint, asset_tickers,
        mean_benchmark_forecasts_df, method, rolling_window_size)
        print(f'\nOut-of-sample OLS Predictor {method} forecasts window size {rolling_window_size}')
        print('-' * 100)
        print(ols_predictor_forecasts_df)
        print(f'\nOut-of-sample Combination Mean {method} forecasts window size'
        f'{rolling_window_size}')
        print('-' * 100)
        print(combination_mean_forecasts_df)
        print(f'\nOut-of-sample Penalised Linear Regression {method} forecasts'
        f'(using all Predictors;window size {rolling_window_size})')
        print('-' * 100)
        print(plr_predictor_forecasts_df)
        print(f'\nMSFE values for all 9 {method} predictive models including benchmark (both asset'
        f'classes)')
        print('-' * 100)
        print(msfe_df)
        print(f'\nRatios of MSFE values of all 8 {method} predictive models to corresponding'
        f'benchmark forecasts MSFEs (both asset classes)')
        print('-' * 100)
        print(msfe_ratios_to_benchmark)
        print(f'\nDM test to check for equal predictive ability relative to'
        f'mean benchmark forecasts(all 8 {method} predictive models for both'
        f'asset classes withwindow size'
        f'{rolling_window_size})')
        print('-' * 100)
        print(dm_test_stats)
        forecasting_analysis.plot_forecast(sp500_all_models, bond_all_models, method)

        portf_cov_mat_forecasts_df = forecasting_analysis.question4(simp_ret_df, breakpoint, method,
        rolling_window_size)
        print(f'\nOut-of-sample variance-covariance matrix {method} forecasts window size
        f'{rolling_window_size}')
        print('-' * 100)
        print(portf_cov_mat_forecasts_df)

        otp_excess_ret_all_models, portf_excess_ret_summary_stats, otp_oos_weights_all_models =
        forecasting_analysis.question5(sp500_all_models, bond_all_models,
        portf_cov_mat_forecasts_df)

        print(f'\nOTP out-of-sample excess returns (all 9 {method} predictive models'
        f'with window size{rolling_window_size})')
        print('-' * 100)
        print(otp_excess_ret_all_models)
        print(f'\nOTP out-of-sample excess returns summary statistics (all 9 {method} predictive'
        f'models with window size {rolling_window_size})')
        print('-' * 100)
        print(portf_excess_ret_summary_stats)
        forecasting_analysis.plot_portfolio_asset_allocation(otp_oos_weights_all_models, method)
        forecasting_analysis.plot_portfolio_returns(otp_excess_ret_all_models, method)
        ##---End of Question 6---##


def parse_arguments():
    """
    Parse command-line arguments for the forecasting analysis script.

    Returns
    -------
    args : argparse.Namespace
        An object containing the parsed command-line arguments.
    """
    parser = argparse.ArgumentParser(description="Forecasting Analysis")
    parser.add_argument("-m", "--method", choices=["recursive", "rolling"], default="recursive",
    help="Forecasting method: 'recursive' or 'rolling'. Default is 'recursive'.")
    parser.add_argument("-w", "--window", type=int, default=240,
    help="Rolling window size (only used if method is 'rolling'). Default is 240.")
    args = parser.parse_args()
    return args

if __name__ == "__main__":
    # Parse command-line arguments and call the main function with the provided arguments.
    args = parse_arguments()
    main(args.method, args.window)
```

```
##-------UTILS FUNCTION-------##
import datetime as dt
import pandas as pd
import numpy as np
from scipy import stats
from scipy.stats import t
import statsmodels.api as sm
import collections
from sklearn.linear_model import LassoCV, RidgeCV
from sklearn.metrics import mean_squared_error

def get_market_data():
    """
    Fetch historical market data on asset classes (stock & bond index),
    risk-free rate & predictors.

    Returns:
    --------
    sp500_full_df : pd.Dataframe
        Monthly S&P 500 stock index price data from Dec 1979 to Dec 2021.
    bond_full_df : pd.Dataframe
        Monthly Bloomberg Barclays U.S. Aggregate Bond Index price data from Dec 1979 to Dec 2021.
    rf_ret_df : pd.Dataframe
        Monthly risk-free rate data from Jan 1980 to Dec 2021.
    predictors_df : pd.Dataframe
        Monthly predictors data from Jan 1980 to Dec 2021.
    """
    # fetch data for S&P 500 stock index (SP500) & Bloomberg Barclays U.S. Aggregate Bond Index
    (LBUSTRUU)
    sp500_full_df = pd.read_excel('./data/S&P_500_stock_index.xlsx')
    bond_full_df = pd.read_excel('./data/US_Aggregate_Bond_index.xlsx')
    # fetch risk-free rate from Professor Kenneth French's data library
    rf_ret_df = pd.read_excel('./data/Risk-free_rate_of_return.xlsx')

    # fetch predictors data
    predictors_df = pd.read_excel('./data/PredictorData2022.xlsx', sheet_name='Monthly')
    predictors_df['Date'] = pd.to_datetime(predictors_df['yyyymm'], format='%Y%m')
    predictors_df.drop(columns=['yyyymm'], inplace=True)
    predictors_df.set_index(['Date'], drop=True, inplace=True)
    predictors_df = predictors_df[(predictors_df.index > dt.datetime(1979, 12, 1)) &
    (predictors_df.index < dt.datetime(2022, 1, 1))]

    return sp500_full_df, bond_full_df, rf_ret_df, predictors_df

def generate_asset_simple_rets(sp500_data: pd.DataFrame, bond_data: pd.DataFrame, rf_data:
pd.DataFrame):
    """
    Generate simple returns and excess returns for the defined asset classes.

    Parameters:
    -----------
    sp500_data : pd.DataFrame
        S&P 500 stock index price data.
    bond_data : pd.DataFrame
        Bloomberg Barclays U.S. Aggregate Bond Index price data.
    rf_data : pd.DataFrame
        risk-free rate data.

    Returns:
    --------
    simp_ret_df : pd.Dataframe
        Monthly simple return & excess return data for both the assets from Jan 1980 to Dec 2021.
        All data combined in a single dataframe for ease of use.
    """
    assets_df = pd.merge(left=sp500_data, right=bond_data, on='Dates', how='inner')
    assets_df.set_index(['Dates'], drop=True, inplace=True)
    # compute simple percentage return
    simp_ret_df = assets_df.pct_change()
    simp_ret_df.dropna(inplace=True)
    simp_ret_df = pd.merge(left=simp_ret_df, right=rf_data, left_on='Dates', right_on='Date',
    how='inner')
    simp_ret_df.set_index(['Date'], drop=True, inplace=True)
    # compute excess return over risk-free rate
    simp_ret_df['SP500_excess'] = simp_ret_df['SP500 index'] -
    simp_ret_df['Risk free rate of return ']

    simp_ret_df['LBUSTRUU_excess'] = simp_ret_df['LBUSTRUU Index '] -
    simp_ret_df['Risk free rate of return ']

    return simp_ret_df

def compute_stat_measures(values):
    """
    Computes summary statistics for input return data.

    Parameters:
    -----------
    values : ndarray
        Asset (excess) returns data.

    Returns:
    --------
    mean : float
        Annualised mean of excess return.
    vol : float
        Annualised volatility of excess return.
    sharpe : float
        Annualised sharpe ratio.
    skew : float
        Skewness of excess return.
    kurt : float
        Kurtosis (Pearson) of excess return.
    """
    # Compute annualised mean: mean
    mean = np.nanmean(values) * 12
    # Compute annualised vol: std
    vol = np.nanstd(values, ddof=1) * np.sqrt(12)
    # Compute annualised sharpe ratio: mean/std
    sharpe = mean / vol
    # Compute skew: skew
    skew = stats.skew(values)
    # Compute kurtosis: kurtosis
    kurt = stats.kurtosis(values, fisher=False)

    return mean, vol, sharpe, skew, kurt

def generate_mean_benchmark_forecast(excess_ret: pd.DataFrame, breakpoint: dt.datetime):
    """
    Generates a timeseries of monthly out-of-sample mean excess return forecasts
    for the defined assets using recursive estimation approach.

    Parameters:
    -----------
    excess_ret : pd.DataFrame
        Assets (excess) returns data (for both asset classes).
    breakpoint : datetime
        datetime to split the data into in-sample & out-of-sample.

    Returns:
    --------
    mean_forecast : pd.DataFrame
        Monthly out-of-sample mean excess forecasts for all assets.
    """
    # define out-of-sample period by splitting data at the breakpoint
    out_of_sample_period = excess_ret[excess_ret.index > breakpoint].index
    # placeholder for out-of-sample mean forecasts
    mean_forecast = pd.DataFrame(index=out_of_sample_period)

    # Loop over the out-of-sample period and compute
    # mean forecasts using recursive estimation approach
    for t in out_of_sample_period:
        # Update the in-sample period to be considered
        # recursively to include all data up to period t
        curr_in_sample = excess_ret[excess_ret.index < t]
        # Compute the expected (mean) excess return for each asset using the current
        # in-sample period
        for asset in curr_in_sample.columns:
            mean_forecast.loc[t, f"{asset.split('_')[0]}_MB"] =
            np.nanmean(curr_in_sample[asset].values)

    return mean_forecast

def generate_ols_predictor_forecast(asset_excess_ret: pd.DataFrame, predictors: pd.DataFrame,
breakpoint: dt.datetime):
    """
    Generates a timeseries of monthly out-of-sample OLS return forecasts
    using the defined predictors for a specific asset class.

    Parameters:
    -----------
    asset_excess_ret : pd.DataFrame
        Asset (excess) returns data (for one asset class).
    predictors : pd.DataFrame
        Predictors data.
    breakpoint : datetime
        datetime to split the data into in-sample & out-of-sample.

    Returns:
    --------
    ols_forecasts : pd.DataFrame
        Monthly out-of-sample OLS forecasts generated by
        fitting over all individual predictors.
    """
    # define out-of-sample period by splitting data at the breakpoint
    out_of_sample_period = asset_excess_ret[asset_excess_ret.index > breakpoint].index
    # placeholder for out-of-sample OLS forecasts
    ols_forecasts = pd.DataFrame(index=out_of_sample_period, columns=predictors.columns)

    for pred in predictors.columns:
        pred_data = predictors.loc[:, [pred]]
        pred_forecasts = []

        for t in out_of_sample_period:
            # Update the in-sample period to be considered
            # recursively to include all data up to period t
            curr_in_sample_asset_ret = asset_excess_ret[asset_excess_ret.index < t]
            curr_in_sample_pred = pred_data.iloc[:asset_excess_ret.index.get_loc(t)]
            curr_in_sample_len = len(curr_in_sample_pred)
```

```python
            regressor = curr_in_sample_pred.iloc[:curr_in_sample_len-1]
            regressand = curr_in_sample_asset_ret.iloc[1:curr_in_sample_len]
            X = sm.add_constant(regressor.values) # add constant to regressor
            y = regressand.values
            ols_model = sm.OLS(y, X) # define OLS model
            ols_res = ols_model.fit() # fit data with defined model
            # predict the very next out-of-sample return forecast
            # (using the last in-sample predictor value to forecast)
            X_hat = sm.add_constant(curr_in_sample_pred.iloc[curr_in_sample_len-1].values,
            has_constant='add')
            forecast = ols_res.predict(X_hat) # ols_model.predict(params=[ols_res.params[0],
            ols_res.params[1]], exog=X_hat)
            pred_forecasts.append(forecast[0])

        ols_forecasts[pred] = pred_forecasts
    return ols_forecasts

def generate_combination_mean_forecasts(asset1_pred_forecasts: pd.DataFrame, asset2_pred_forecasts:
pd.DataFrame, assets_names: list):
    """
    Generates a timeseries of monthly out-of-sample return forecasts by taking
    mean of the combination of corresponding predictor forecasts of each asset class.
    Finally returns a dataframe with combination mean forecasts for both asset classes.

    Parameters:
    -----------
    asset1_pred_forecasts : pd.DataFrame
        Monthly out-of-sample OLS predictor forecats (5 models) for asset class 1 (SP500).
    asset2_pred_forecasts : pd.DataFrame
        Monthly out-of-sample OLS predictor forecats (5 models) for asset class 2 (LBUSTRUU).
    assets_names : list
        List of asset names (tickers).

    Returns:
    --------
    combination_mean_forecasts : pd.DataFrame
        Monthly out-of-sample Combination mean forecasts generated by
        taking mean of individual OLS predictor forecasts. (both asset classes)
    """
    assert len(asset1_pred_forecasts) == len(asset2_pred_forecasts)
    # rename the asset names to reflect that these correspond to combination mean forecasts
    assets = [f'{asset}_Comb_Mean' for asset in assets_names]
    combination_mean_forecasts = pd.DataFrame(index=asset1_pred_forecasts.index, columns=assets)
    combination_mean_forecasts[assets[0]] = np.nanmean(asset1_pred_forecasts, axis=1)
    combination_mean_forecasts[assets[1]] = np.nanmean(asset2_pred_forecasts, axis=1)

    return combination_mean_forecasts

def generate_plr_forecast(asset_excess_ret: pd.DataFrame, predictors: pd.DataFrame, breakpoint:
dt.datetime):
    """
    Generates a timeseries of monthly out-of-sample return forecasts
    predicted using penalised linear regression models fit on
    defined predictors for a specific asset class

    Parameters:
    -----------
    asset_excess_ret : pd.DataFrame
        Asset (excess) returns data (for one asset class).
    predictors : pd.DataFrame
        Predictors data.
    breakpoint : datetime
        datetime to split the data into in-sample & out-of-sample.

    Returns:
    --------
    plr_forecasts : pd.DataFrame
        Monthly out-of-sample Penalised Linear Regression forecasts generated by
        fitting over all predictors (both models Lasso & Ridge - for one asset class only).
    """
    # define out-of-sample period by splitting data at the breakpoint
    out_of_sample_period = asset_excess_ret[asset_excess_ret.index > breakpoint].index
    # placeholder for out-of-sample lasso & ridge (plr) forecasts
    plr_forecasts = pd.DataFrame(index=out_of_sample_period, columns=['Lasso', 'Ridge'])
    forecasts = []

    for t in out_of_sample_period:
        # Update the in-sample period to be considered
        # recursively to include all data up to period t
        curr_in_sample_asset_ret = asset_excess_ret.loc[asset_excess_ret.index < t]
        curr_in_sample_pred = predictors.iloc[:asset_excess_ret.index.get_loc(t)]
        curr_in_sample_len = len(curr_in_sample_pred)
        X = curr_in_sample_pred.iloc[:curr_in_sample_len-1].values
        y = curr_in_sample_asset_ret.iloc[1:curr_in_sample_len].values

        # predict the very next out-of-sample return forecast (using the last in-sample predictor
        values to forecast)
        X_hat = curr_in_sample_pred.iloc[curr_in_sample_len-1].values.reshape(1, -1)
        # fit Lasso model on data
        lasso_model = LassoCV()
        lasso_model.fit(X, y)
        lasso_forecast = lasso_model.predict(X_hat)
        # fit Ridge model on data
        ridge_model = RidgeCV()
        ridge_model.fit(X, y)
        ridge_forecast = ridge_model.predict(X_hat)

        forecasts.append([lasso_forecast[0], ridge_forecast[0][0]])

    plr_forecasts.loc[:, :] = forecasts
    return plr_forecasts

def dm_test(real_values, pred1, pred2, h=1):
    """
    Diebold-Mariano test for equal predictive ability.

    Parameters
    ----------
    real_values : array-like of shape (n_samples,)
        Ground truth (correct) target values.
    pred1 : array-like of shape (n_samples,)
        Predicted values from model 1.
    pred2 : array-like of shape (n_samples,)
        Predicted values from model 2.
    h : int, optional (default=1)
```

```
        Forecast horizon.

    Returns
    -------
    tuple : named tuple
        DM test statistics and p-value.
    """
    # placeholders for square differences & loss differential
    e1_lst = []
    e2_lst = []
    d_lst = []
    real_values = pd.Series(real_values).apply(lambda x: float(x)).tolist()
    pred1 = pd.Series(pred1).apply(lambda x: float(x)).tolist()
    pred2 = pd.Series(pred2).apply(lambda x: float(x)).tolist()

    # Length of forecasts
    T = float(len(real_values))
    # Construct loss differential according to error criterion (MSE)
    for real, p1, p2 in zip(real_values, pred1, pred2):
        e1_lst.append((real - p1)**2)
        e2_lst.append((real - p2)**2)
    for e1, e2 in zip(e1_lst, e2_lst):
        d_lst.append(e1 - e2)
    # Mean of loss differential
    mean_d = pd.Series(d_lst).mean()

    # Calculate autocovariance
    def autocovariance(Xi, N, k, Xs):
        autoCov = 0
        T = float(N)
        for i in np.arange(0, N-k):
            autoCov += ((Xi[i+k])-Xs) * (Xi[i]-Xs)
        return (1/(T))*autoCov

    # Calculate the denominator of DM stat
    gamma = []
    for lag in range(0, h):
        gamma.append(autocovariance(d_lst, len(d_lst), lag, mean_d))
    V_d = (gamma[0] + 2*sum(gamma[1:]))/T

    # Calculate DM stat
    DM_stat = V_d**(-0.5) * mean_d

    # Calculate and apply Harvey adjustement
    # It applies a correction for small sample
    harvey_adj = ((T+1 - 2*h + h*(h-1)/T)/T)**(0.5)
    DM_stat = harvey_adj*DM_stat
    # Calculate p-value
    p_value = 2 * t.cdf(-abs(DM_stat), df=T - 1)

    dm_return = collections.namedtuple('dm_return', 'DM p_value')
    result = dm_return(DM=DM_stat, p_value=p_value)

    return result

def generate_portfolio_var_cov_mat_forecast(excess_ret: pd.DataFrame, breakpoint: dt.datetime):
    """
    Generates a timeseries of monthly out-of-sample variance-covariance
    matrix forecasts for a portfolio of assets.

    Parameters:
    -----------
    excess_ret : pd.DataFrame
        Assets (excess) returns data (for both asset classes).
    breakpoint : datetime
        datetime to split the data into in-sample & out-of-sample.

    Returns:
    --------
    portf_var_cov_forecast : pd.DataFrame
        Monthly out-of-sample forecast of variance-covariance matrices for a portfolio of assets.
    """
    out_of_sample_period = excess_ret[excess_ret.index > breakpoint].index
    # define out-of-sample period by splitting data at the breakpoint
    # placeholder for out-of-sample
    portf_var_cov_forecast = pd.DataFrame(index=out_of_sample_period)
    portfolio var-cov forecasts
    cov_mats = []

    # Loop over the out-of-sample period and compute
    # portfolio var-cov matrice forecasts using recursive estimation approach
    for t in out_of_sample_period:
        # Update the in-sample period to be considered
        # recursively to include all data up to period t
        curr_in_sample = excess_ret[excess_ret.index < t]
        # Compute the portfolio var-cov matrices using the current in-sample period
        cov_mat = np.cov(curr_in_sample.values, rowvar=False, ddof=1)
        cov_mats.append(cov_mat)

    portf_var_cov_forecast['portf_asset_cov_mat'] = cov_mats
    return portf_var_cov_forecast

def generate_OTP_weights(excessMean, cov):
    """
    Calculate the weights of the Optimal Tangency Portfolio.

    Parameters:
    -----------
    excessMean : ndarray
        Excess mean returns array.

    cov : ndarray
        Covariance of asset returns.

    Returns:
    --------
    weights_TP : array_like
        The weights computed for OTP using the input excesss mean & cov matrix.
    """
    inv_cov = np.linalg.inv(cov)
    num = inv_cov @ excessMean.T
    den = np.ones(cov.shape[0]) @ inv_cov @ excessMean.T
    weights_TP = num/den
    return weights_TP

def generate_OTP_excess_ret(assets_ret_forecasts: pd.DataFrame, cov_mat_forecasts: pd.DataFrame):
```

```python
    """
    Generates a timeseries of monthly out-of-sample Optimal
    Tangency Portfolio excess return forecasts for a portfolio of assets.

    Parameters:
    -----------
    assets_ret_forecasts : pd.DataFrame
        Monthly out-of-sample return forecasts predicted by a specific model (for both assets).
    breakpoint : datetime
        datetime to split the data into in-sample & out-of-sample.

    Returns:
    --------
    otp_excess_ret : NDArray
        Monthly out-of-sample excess portfolio returns.
    all_weights : List of portfolio weights
        Monthly out-of-sample portfolio weights.
    """
    # define a placeholder for time-series of OTP excess return
    otp_excess_ret = pd.Series(index=assets_ret_forecasts.index)
    excess_ret_mean = np.array(assets_ret_forecasts.mean())
    all_weights = []

    # iterate over the entire period
    for i in range(len(otp_excess_ret)):
        # generate out-of-sample OTP weights
        otp_weights = generate_OTP_weights(excessMean=excess_ret_mean,
        cov=cov_mat_forecasts.iloc[i].values[0])
        all_weights.append(otp_weights)
        # Compute the portfolio excess return
        otp_excess_ret.iloc[i] = assets_ret_forecasts.iloc[i].values @ otp_weights.T

    return otp_excess_ret.values, all_weights
```