FIN42120: Programming for Financial Data Science

Group Project

Group 1

Submitted: 12th May, 2023

	Name & Surname	**Student Number**
1.	Gavin Connolly	18308483
2.	Sofia Emelianova	22205187
3.	Evan McGarry	22206023
4.	Ajit Nambiyar	22200172
5.	Fanis-Filippos Papanikolaou	22200286

Task 0 - Individual Contribution

	Name & Surname	Student Number	Type of Contribution	Contribution %
1.	Gavin Connolly	18308483	Coding 40%	20%
2.	Sofia Emelianova	22205187	Writing 40%	20%
3.	Evan McGarry	22206023	Coding 40%	20%
4.	Ajit Nambiyar	22200172	Writing 40%	20%
5.	Fanis-Filippos Papanikolaou	22200286	Coding 20% Writing 20%	20%

Task 1 - Data Collection & Summary Statistics

To conduct the analysis, we collected the monthly prices of the S&P500 from the U.S. Federal Reserve Economic Data (FRED) website and the US Aggregate Bond Index (LBUS-TRUU) from Bloomberg. Furthermore, we obtained the monthly risk-free rate of return data from Professor Kenneth French's data library.

We then imported this data into Python and calculated the summary statistics found in the table below with the code for generating these statistics provided in Appendix A.1:

Table 1: Summary Statistics of the S&P500 Index and US Aggregate Bond Index

Statistics	SPY	Bond
Annualized Mean	0.0641	0.0327
Standard Deviation	0.1508	0.0511
Sharpe Ratio	0.4252	0.6389
Skewness	-0.6348	0.4221
Kurtosis	2.1174	6.2422

The summary statistics show that the S&P500 Index (SPY) offers a greater return but this is at the expense of much higher risk when compared with the US Aggregate Bond Index (Bond). The annualised mean for excess returns of SPY are essentially double that of Bond, but the standard deviation (volatility) is tripled. Unsurprisingly, this result in a higher "*Sharpe Ratio*" for Bond as this is a much less risky investment whilst still offering acceptable returns. SPY is negatively skewed whilst Bond is positively skewed - given the stable nature of bonds relative to stocks this result is within expectations. The high kurtosis of Bond is initially worrying, however this needs to be read in conjunction with the standard deviation. The investment in Bond is still good since the high kurtosis is offset by a low standard deviation. SPY has a relatively lower kurtosis (albeit it is still high) with a much higher standard deviation which makes it a worse investment.

Task 2 - Recursive Estimation of Mean Excess Return Forecasts

The sample was divided into an in-sample period and out-sample period which are defined below:

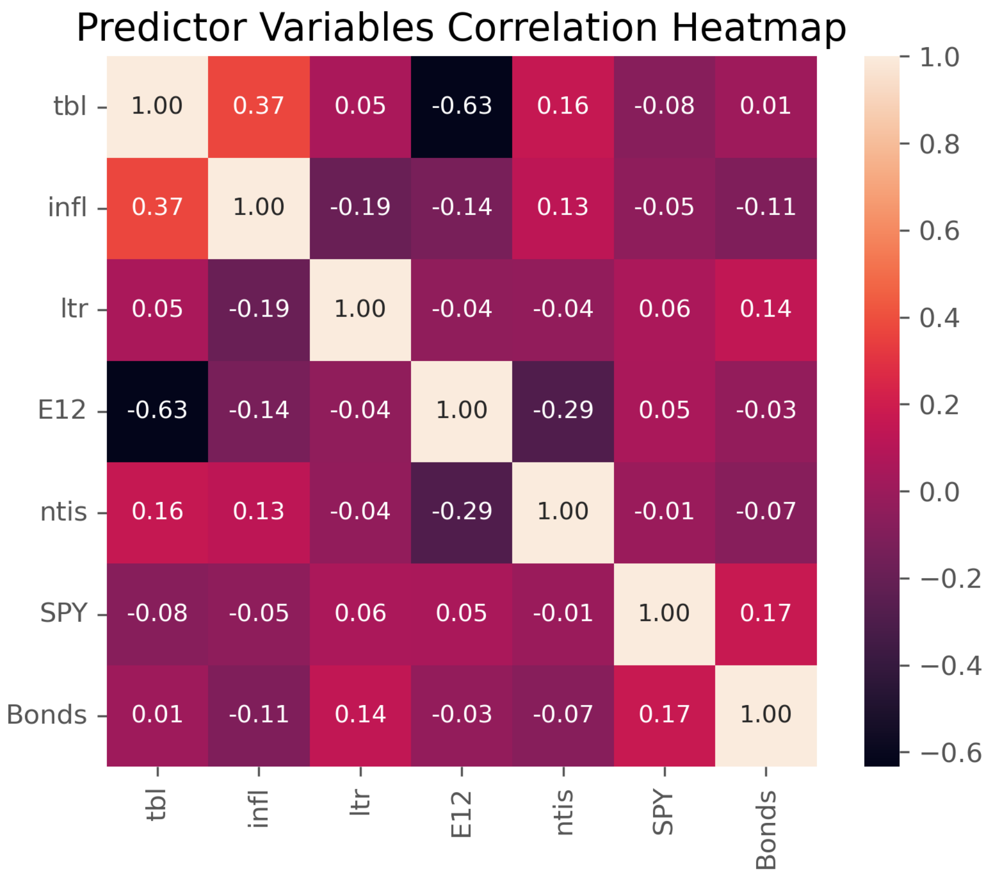
1. In-sample period spanning from January 1980 to December 1999.
2. Out-of-sample period from January 2000 to December 2021.

We then defined a function to compute the mean excess return forecasts, utilizing a recursive estimation technique to generate time-series of monthly out-of-sample constant expected (mean) excess return forecasts for two asset

classes. The function takes the mean of the in-sample period as being the first forecasted value in the out-sample period. Finally, the function returns a recursively forecasted time-series of monthly mean benchmark excess returns for the out-of-sample period, based on the mean of the recursively defined in-sample period.
The mean benchmark forecast is a valuable tool for assessing the performance and accuracy of forecasts for each asset class. The code for these steps can be found in Appendix A.2.

Task 3 - Predictive Models Recursive Forecasts & Comparisons

In order to determine the most suitable variables, a correlation heatmap was constructed to determine which variables were highly correlated with SPY and Bond which is shown below:



These predictors were researched and the correlation was analysed to determine what the best predictor variables for SPY and Bonds were. With our predictors selected the 8 predictive models were created in order to forecast monthly excess returns for each asset class (*i.e.* SPY & Bond). The models are listed below:

1. OLS Predictive Regression Model for each of the 5 Predictors selected above (resulting in 5 individual models with each being a simple linear regression).

2. Combination forecasts taking a simple average of the above 5 models (this is referred to as "*Ensemble*").

3. 2 penalised linear regression:

   (a) Ridge Regression

   (b) Lasso Regression

These models were fitted onto the data and used to forecast excess returns for each asset class. With the models fitted on the data and used for forecasting it was necessary to evaluate and compare the performance of each of these models - this was done by calculating the Mean Squared Forecast Error (MSFE) of each model. In order to compare them with the mean benchmark excess returns, the ratio of each model's MSFE to the mean benchmark MSFE was calculated. This allowed us to consider performance relative to our benchmark and further to this it was necessary to run the Diebold-Mariano Test to determine whether or not the forecasts from each model are significantly different from the mean benchmark:

- H0: The Mean of the Standardised Loss Differential (DM-Stat) = 0
- H1: The Mean of the Standardised Loss Differential (DM-Stat) $\neq 0$

The Null Hypothesis simply states that there is no difference in predictive accuracy between the model's forecasted excess returns and the mean benchmark excess returns. Setting $\alpha = 0.05$ (*i.e.* our significance level equal to 5%) we can evaluate the differences in predictive ability of each model by calculating the test statistics and p-values. This was done firstly for the SPY asset class with the results below:

Table 2: Comparing the Performance of each Predictive Model with the Mean Benchmark Excess Returns for the SPY Asset Class

Model (SPY)	MSFE Ratio	Test Statistic	p-value
forecast: tbl	0.9899	-0.0002	0.5001
forecast: infl	0.9737	-0.0007	0.5003
forecast: ltr	0.9983	-0.0001	0.5000
forecast: E12	0.9835	-0.0004	0.5002
forecast: ntis	0.9873	-0.0003	0.5001
forecast: ensemble	0.9907	-0.0003	0.5001
forecast: ridge	0.9882	-0.0003	0.5001
forecast: lasso	0.9984	-0.0001	0.5000

As we can see above, the MSFE ratios are similar for all the models and are all less than 1 which seems to suggest that each of these models perform better than the mean benchmark. However, we fail to reject the Null Hypothesis for all of the models. In conclusion, for the SPY asset class the models are not significantly different, in terms of predictive ability, than the mean benchmark excess returns.

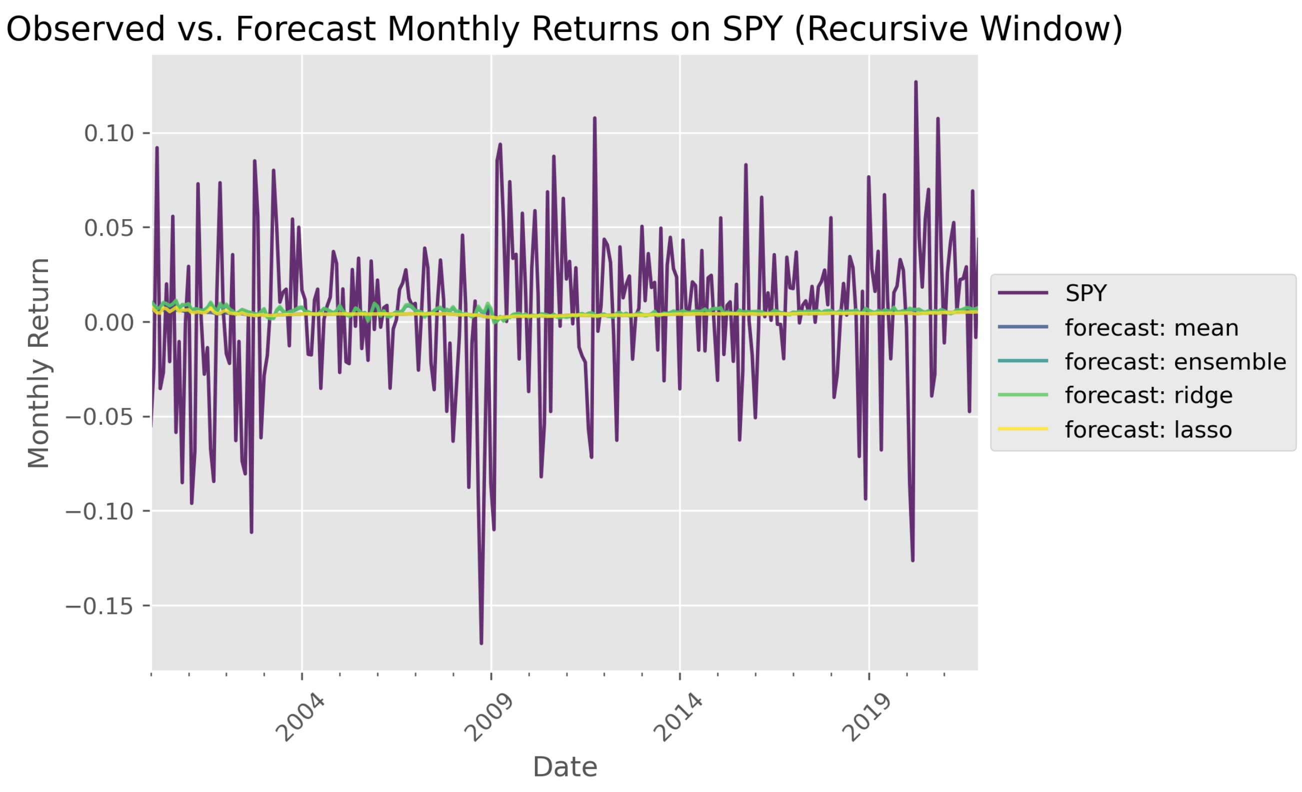
The same process was repeated for the Bond asset class with the results below:

Table 3: Comparing the Performance of each Predictive Model with the Mean Benchmark Excess Returns for the Bond Asset Class

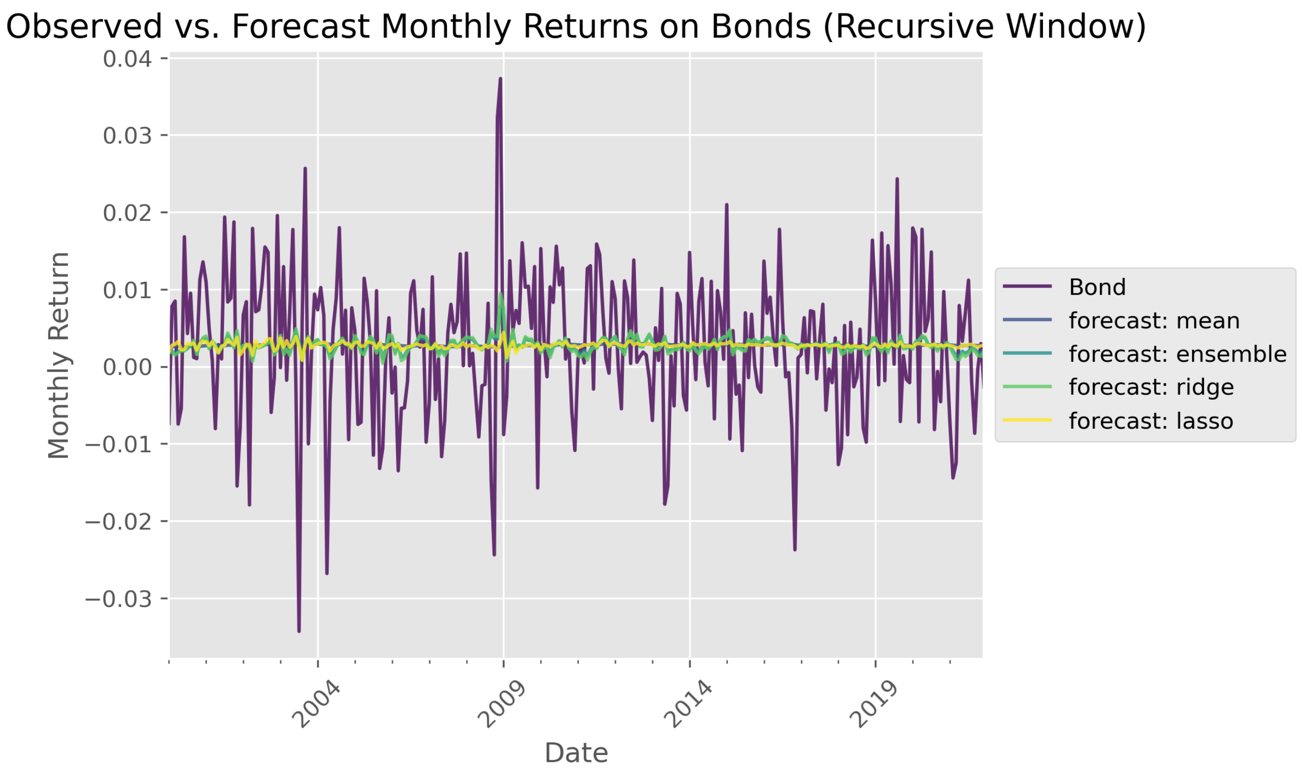
Model (Bond)	MSFE Ratio	Test Statistic	p-value
forecast: tbl	0.9914	-0.0000	0.5000
forecast: infl	1.0081	0.0000	0.5000
forecast: ltr	0.9829	-0.0001	0.5000
forecast: E12	0.9889	-0.0000	0.5000
forecast: ntis	0.9462	-0.0001	0.5000
forecast: ensemble	1.0079	0.0000	0.5000
forecast: ridge	1.0089	0.0000	0.5000
forecast: lasso	0.9986	-0.0000	0.5000

The MSFE Ratios are all similar but some of them are greater than 1 which suggests a higher MSFE than the mean benchmark. However this is irrelevant since we fail to reject the Null Hypothesis for all of the models. In conclusion, for the Bond asset class the models are not significantly different, in terms of predictive ability, than the mean benchmark excess returns.

There were numerous functions created to facilitate the output as seen above for both the SPY and Bond asset classes, the code for which is provided in Appendix A.3. With these results stored, the final step was to create time-series plots that show the *excess returns forecasts* for the mean benchmark, ensemble and the two penalised linear regressions for each of the asset classes:



For the asset class SPY, the models and the benchmark had very similar predictions throughout the in-sample period. This is in line with our failure to reject the Null Hypothesis as we would expect these models to have similar performance in terms of monthly excess returns forecasts.



There are bigger differences in the forecasts of the models for the Bond asset class with the Ridge Regression forecasting much higher excess returns in 2009 than the rest of the models. Overall the models behave in a similar manner which is in line with our rejection of the Null Hypothesis and the slight deviations are partially explained by the differences in the MSFE ratios between the models with the Ridge Regression having a higher MSFE than the mean benchmark. The code for these plots can also be found in Appendix A.3.

Task 4 - Recursive Estimation of Variance-Covariance Forecasts

In order to calculate the out-of-sample forecasts of the variance-covariance matrix, we adopt a similar approach to that used in calculating the mean forecast in Task 2. We calculate the 2x2 covariance matrix for our in-sample period, and then move the right-hand boundary of the in-sample period.

We flatten this 2x2 output before storing it as a row in an array. At the end of the loop, the array is converted to a dataframe, giving us a recursively-generated forecast of the covariance matrix of the returns on SPY & Bonds for each of the 264 months in the out-of-sample period. The code for this process can be found in Appendix A.4.

Task 5 - Construction & Analysis of Recursively Defined Optimal Tangency Portfolios

Using the mean benchmark excess return forecasts and sample variance-covariance matrix forecasts, out-of-sample optimal tangency portfolio weights for a mean-variance investor were constructed. The annualized summary statistics for the optimal portfolio's excess return are described under *forecast: mean* in Table 4.

This table also includes the annualised summary statistics for the alternative optimal tangency portfolios which were calculated in the same way as the original optimal portfolio but instead the forecasts for the monthly excess returns were used in conjunction with the covariance matrix from Q4. The code for calculating the annualised summary statistics for all the portfolios is provided in Appendix A.5 and the results are shown below:

Table 4: Summary Statistics for Optimal Tangency Portfolios.

Model	Mean Excess Return	Std. Dev of Excess Return	Sharpe Ratio
forecast: mean	0.0368	0.0615	0.5989
forecast: tbl	0.0321	0.0701	0.4581
forecast: infl	0.0048	0.1311	0.0369
forecast: ltr	0.0740	0.0914	0.8097
forecast: E12	0.0791	0.1393	0.5677
forecast: ntis	0.0281	0.0650	0.4328
forecast: ensemble	0.0392	0.0647	0.6063
forecast: ridge	0.0396	0.0659	0.6011
forecast: lasso	0.0387	0.0623	0.6209

The results from Table 4 can be interpreted as follows:

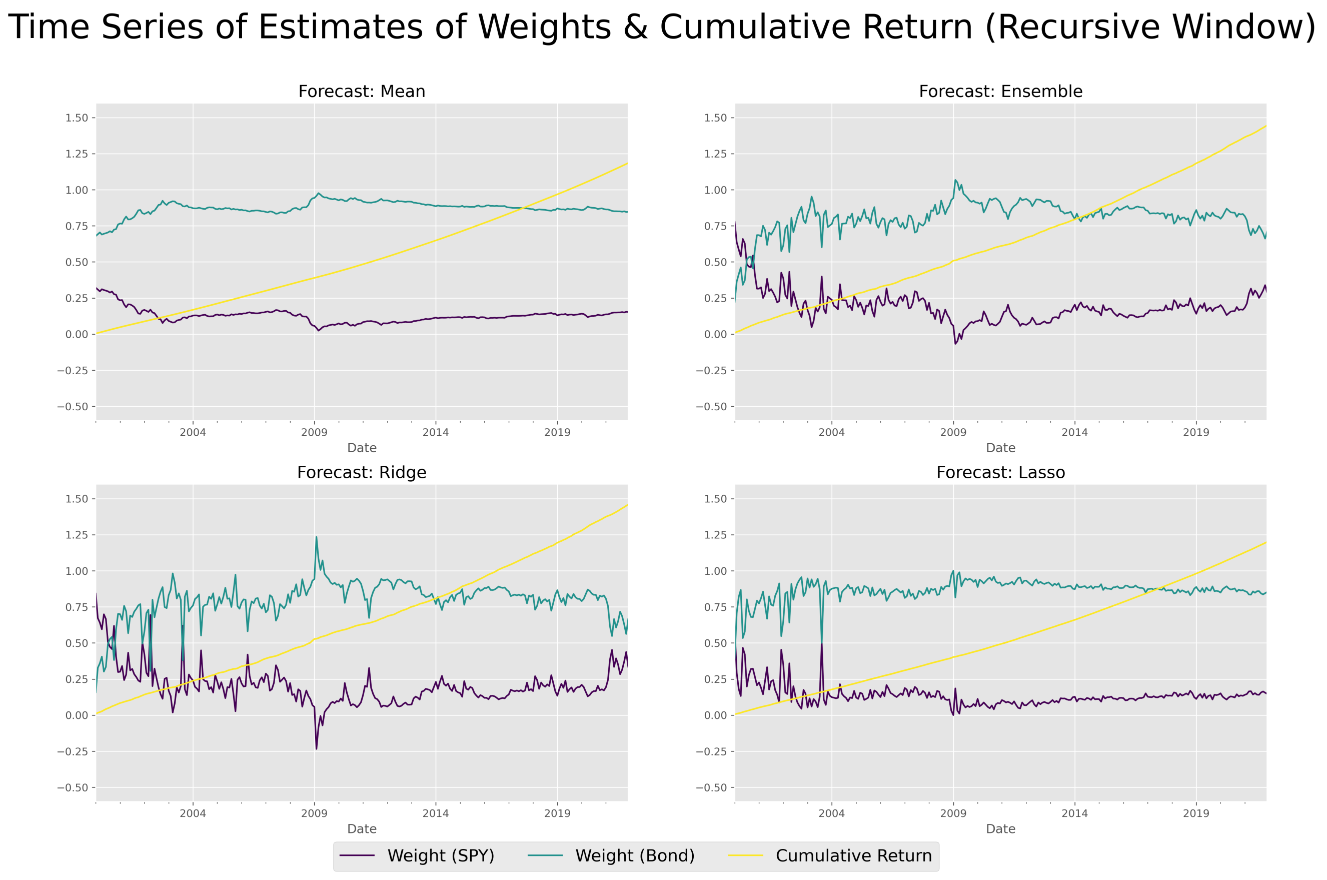
1. **Mean Excess Return:** Summarises the average excess returns for each forecast. The highest mean excess return is associated with *forecast: E12* at 7.9% thus indicating that this method yielded the highest expected return.

2. **Std. Dev of Excess Return:** Represents the standard deviation (*i.e.* volatility) of the excess returns. The simple linear regression using E12 (*forecast: E12*) as the predictor variable yielded the highest mean excess return but it also has the highest volatility which will likely make this model rank poorly when considering Sharpe Ratios. The forecast with the lowest volatility is *forecast: mean* with a standard deviation of 6.1503% but this is coupled with a meagre mean excess return of approx. 3%. In order to determine which model has the best combination of volatility and mean excess return it is necessary to consider the Sharpe Ratio of each portfolio.

3. **Sharpe Ratio:** The Sharpe ratio is the most commonly used method to measure risk-adjusted relative returns and as such will act as a ranking system for each of our optimum tangency portfolios. As we can see above *forecast: ltr* has the highest Sharpe Ratio by a significant amount. It offers high return with relatively low risk. The ensemble, ridge regression and lasso regression all achieve similar ranks and are all ranked above the optimal tangency portfolio of the mean benchmark (*forecast: mean*). Therefore, there is a possibility for

economic gain through investing in one of these portfolios as they all rank above the mean benchmark in terms of risk-adjusted relative return.

A final evaluation was done by visually comparing the plots of 4 portfolios as a time-series of the recursive estimations of weights & cumulative returns:



From these plots we see that the Ensemble and Ridge Regression model have higher cumulative excess return and the tangency portfolio weights have been more volatile. For the mean benchmark and Lasso regression the cumulative excess returns are lower and there has tangency portfolio weights are more stable.

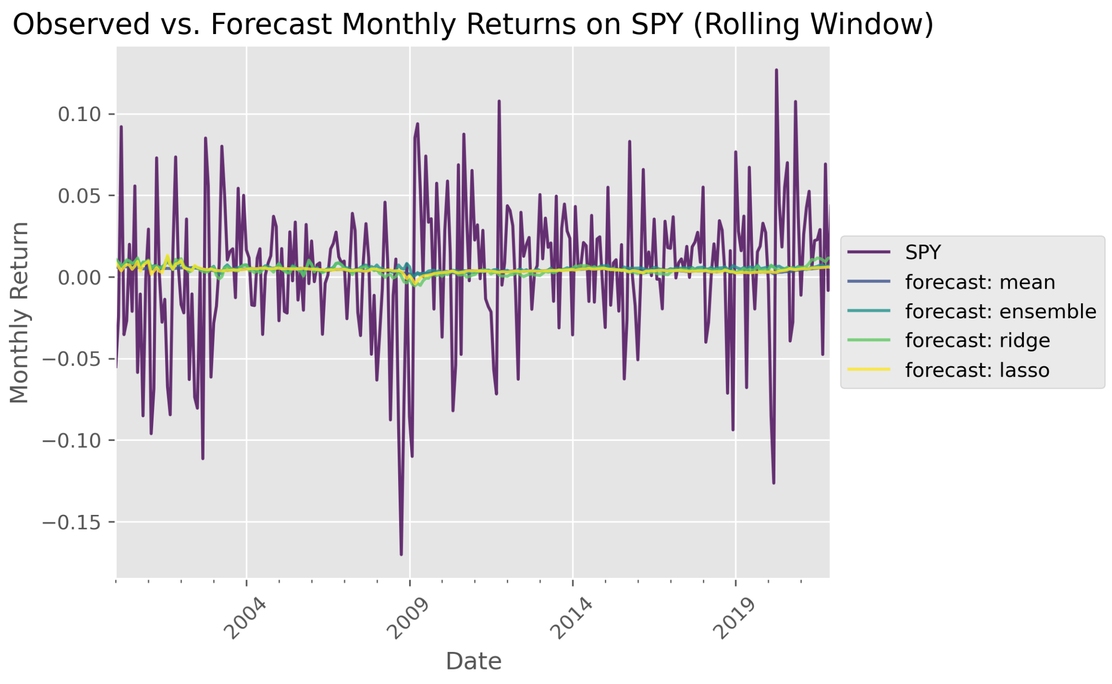
Task 6 - Rolling Window Estimation Approach

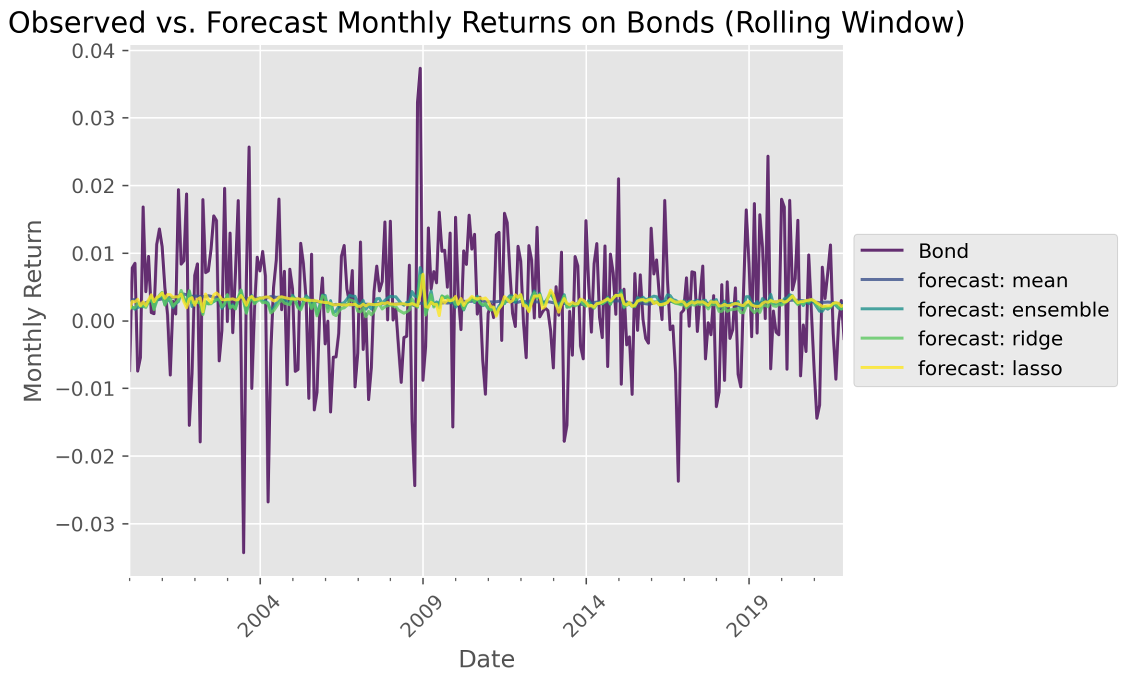
Comparisons using Rolling Window Estimation Approach

For this part, the above tasks were repeated, using a rolling window instead of a recursive estimation approach. All of the code for this section can be found in Appendix A.6.

Predictive Models Rolling Forecasts & Comparisons

As we can see, the performance of the regression models on a rolling window is extremely similar to the recursive one, with the forecasted returns being slightly more volatile on the rolling window.





The Diebold-Mariano Test was run again using a rolling window. Setting $\alpha = 0.05$ (*i.e.* our significance level equal to 5%) we can evaluate the differences in predictive ability of each model by calculating the test statistics and p-values. This was done firstly for the SPY asset class with the results below:

Table 5: Diebold-Mariano Summary Test Statistics for SPY Asset Class.

Model (SPY)	**MSFE Ratio**	**Test Statistic**	**p-value**
forecast: tbl	0.9861	-0.0003	0.5001
forecast: infl	0.9700	-0.0008	0.5003
forecast: ltr	0.9944	-0.0004	0.5002
forecast: E12	0.9797	-0.0006	0.5002
forecast: ntis	0.9835	-0.0004	0.5002
forecast: ensemble	0.9869	-0.0004	0.5002
forecast: ridge	0.9853	-0.0005	0.5002
forecast: lasso	0.9871	-0.0005	0.5002

It is apparent that each of these models have a lower MSFE than the mean benchmark. However, this is essentially irrelevant since we fail to reject the Null Hypothesis for each model. This is almost identical to the results of the Diebold-Mariano Test using the recursive estimation approach.

The same process was repeated for the Bond asset class with the results below:

Table 6: Diebold-Mariano Summary Test Statistics for Bond Asset Class.

Model (Bond)	**MSFE Ratio**	**Test Statistic**	**p-value**
forecast: tbl	0.9784	-0.0001	0.5000
forecast: infl	0.9948	-0.0000	0.5000
forecast: ltr	0.9699	-0.0001	0.5000
forecast: E12	0.9759	-0.0000	0.5000
forecast: ntis	0.9338	-0.0001	0.5000
forecast: ensemble	0.9947	-0.0000	0.5000
forecast: ridge	0.9909	-0.0000	0.5000
forecast: lasso	0.9817	-0.0000	0.5000

Interestingly using the rolling window estimation approach resulted in the MSFE for all of the models being lower than the mean benchmark whereas using the recursive estimation approach resulted in some models performing worse. However, since we fail to reject the Null Hypothesis for each model again this is irrelevant as these models are not significantly different from the mean benchmark.

Construction & Analysis of Rolling Optimal Tangency Portfolios

Table 7: Summary Statistics for Optimal Tangency Portfolios.

Model	Mean Excess Return	Std. Dev of Excess Return	Sharpe Ratio
forecast: mean	0.0349	0.0604	0.5774
forecast: tbl	0.0316	0.0624	0.5060
forecast: infl	0.0259	0.0681	0.3810
forecast: ltr	0.0639	0.0812	0.7865
forecast: E12	0.0823	0.1450	0.5674
forecast: ntis	0.0303	0.0643	0.4719
forecast: ensemble	0.0382	0.0605	0.6314
forecast: ridge	0.0401	0.0604	0.6627
forecast: lasso	0.0370	0.0607	0.6091

The results from Table 7 can be interpreted as follows:

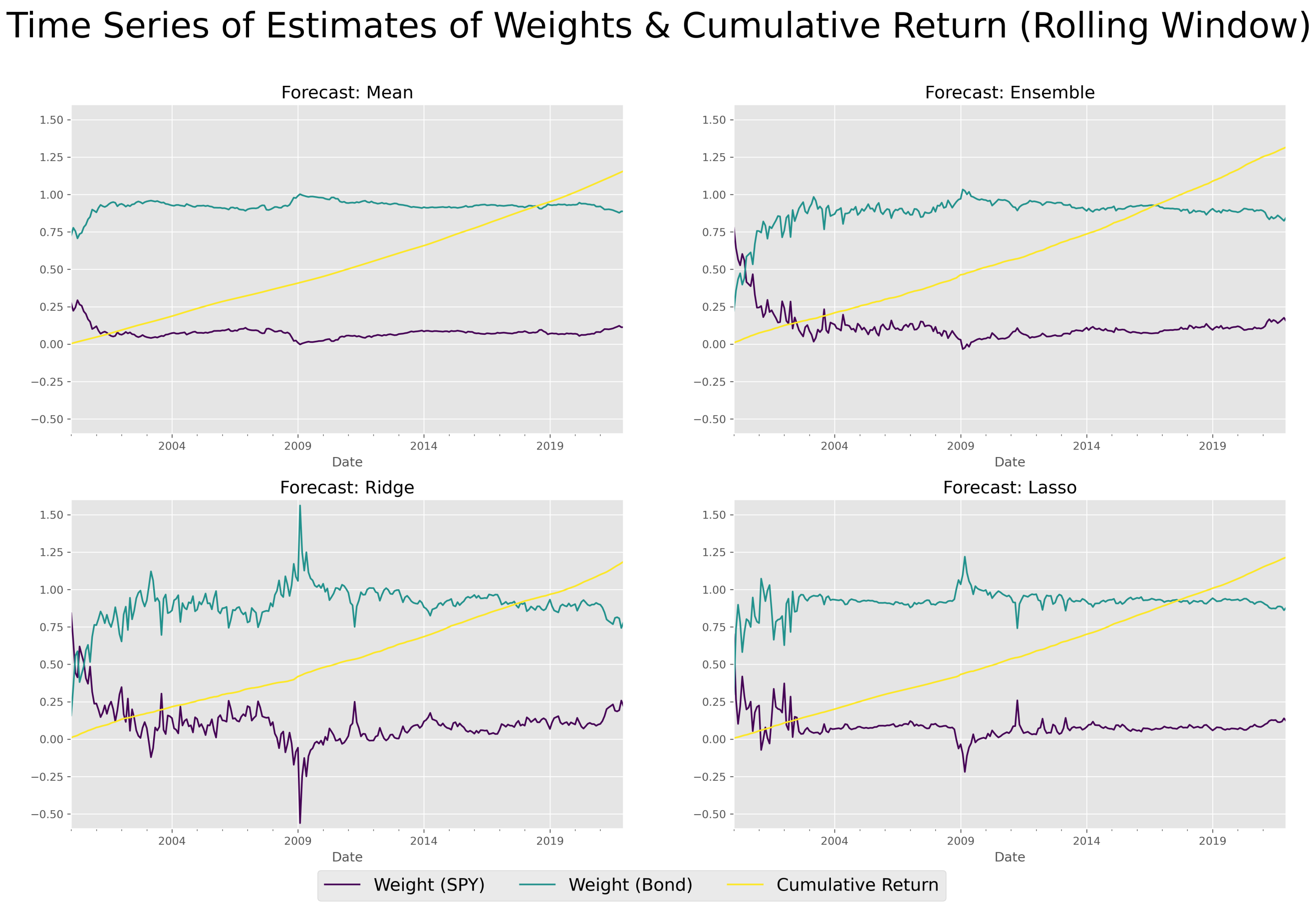
1. **Mean Excess Return:** *forecast: E12* demonstrated the highest mean excess return of 8.2% as was the case in the recursive approach.

2. **Std. Dev of Excess Return:** *forecast: E12* method exhibited both the highest mean and also the highest calculated volatility measured at 14.5%. Despite these high returns the Sharpe Ratio of this portfolio is likely to be relatively low as the high volatility offsets the benefit of the high returns. This is consistent with the results obtained using the recursive estimation approach.

3. **Sharpe Ratio:** The notable differences between the Sharpe Ratios obtained using the rolling window estimation approach versus the recursive estimation approach are seen in the key portfolios that perform better than the mean benchmark. The portfolios that perform better than the mean benchmark are the same for both estimation approaches. However, the best Sharpe Ratio was obtained using the recursive estimation approach (*forecast: ltr*) whereas this decreased after using the rolling window estimation approach. Interestingly, the ensemble and ridge regression portfolios had an improvement in their sharpe ratios but this improvement was not significant enough to favour them. The last notable difference was that the Lasso regression had a decreased sharpe ratio. Overall, the highest performing portfolio was obtained using the recursive estimation approach with only marginal improvements for certain models resulting from the adoption of the rolling window estimation approach.

When looking at the time series of the tangency portfolio cumulative returns, we can see that the outliers we saw for the rolling window are even more extreme. The reason being the smaller sample size of the rolling window, meaning extreme values have proportionally more influence over the parameterization of the model.

Another interesting thing to notice is that, despite nominally having a higher mean excess return, the rolling Ridge model obtains a lower cumulative returns. Relatively poor performance in the period prior to 2009 is balanced out by much stronger performance in later years (as compared to the recursive model), leading to a higher mean excess return, but the timing of these returns in the later periods leads to lower compounding over the entire period and thus lower cumulative excess returns overall.



Conclusion

Using a rolling window allows the model to learn more quickly from changing market conditions, but also leaves it more vulnerable to over-fitting. In terms of optimum tangency portfolio construction, the highest performing portfolio in terms of Sharpe ratio was estimated using the recursive estimation approach. The adoption of the rolling window estimation approach did improve the performance of certain portfolios but not to a degree that would result in the adoption of these portfolios. Interestingly, the MSFE of the models decreased when using the rolling window estimation approach but since every model was not significantly different than the mean benchmark forecast this was irrelevant.

A Appendices - Code

```
# Packages Used
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime as dt
import matplotlib as mpl
import seaborn as sns
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
from scipy.stats import norm


plt.style.use('ggplot')
mpl.rcParams['figure.dpi'] = 300
```

A.1 Question 1

```
# Q1 - SPY Index
spy = pd.read_excel(r'S&P_500_stock_index.xlsx')
spy.rename(columns={'Dates':'Date'}, inplace=True)
spy['SP500 index'] = spy['SP500 index'].pct_change()
spy.dropna(inplace=True)
spy.reset_index(drop=True, inplace=True)

# Q1 - US Aggregate Bond Index
bond = pd.read_excel(r'US_Aggregate_Bond_Index.xlsx')
bond.rename(columns={'LBUSTRUU Index ':'Agg Bond Index', 'Dates': 'Date'}, inplace=True)
bond['Agg Bond Index'] = bond['Agg Bond Index'].pct_change()
bond.dropna(inplace=True)
bond.reset_index(drop=True, inplace=True)

# Q1 - Risk Free Rate
rf = pd.read_excel(r'Risk-free_rate_of_return.xlsx')
rf.rename(columns={'Risk free rate of return ': 'i_rf'}, inplace=True)

# Q1 - Set up Merged DataFrame
df_ind = pd.merge(spy, bond, on='Date', how='inner')
df_final = pd.merge(df_ind, rf, on='Date', how='inner')

# Q1 - Calculate Excess Returns (SPY)
df_final['Excess Returns (SPY)'] = df_final['SP500 index'] - df_final['i_rf']

# Q1 - Calculate Excess Returns (Bond)
df_final['Excess Returns (Bond)'] = df_final['Agg Bond Index'] - df_final['i_rf']
df_final.drop(columns=['SP500 index', 'Agg Bond Index', 'i_rf'], inplace=True)

# Q1 - Function
def sum_stats(df):

    # Annualised Mean from Monthly Excess Returns
    mean_spy = (1 + df['Excess Returns (SPY)'].mean())**12 - 1
    mean_bond = (1 + df['Excess Returns (Bond)'].mean())**12 - 1
    # Annualised Standard Deviation from Monthly Excess Returns
    standard_deviation_spy = df['Excess Returns (SPY)'].std() * np.sqrt(12)
    standard_deviation_bond = df['Excess Returns (Bond)'].std() * np.sqrt(12)

    # Annualised Sharpe Ratio
    sharpe_ratio_spy = mean_spy / standard_deviation_spy
    sharpe_ratio_bond = mean_bond / standard_deviation_bond

    # Skewness from Monthly Returns
    skewness_spy = df['Excess Returns (SPY)'].skew()
    skewness_bond = df['Excess Returns (Bond)'].skew()

    # Kurtosis from Monthly Returns
    kurtosis_spy = df['Excess Returns (SPY)'].kurtosis()
    kurtosis_bond = df['Excess Returns (Bond)'].kurtosis()

    summary_spy = pd.Series([mean_spy, standard_deviation_spy,
    sharpe_ratio_spy, skewness_spy, kurtosis_spy], name='SPY')
    summary_bond = pd.Series([mean_bond, standard_deviation_bond,
    sharpe_ratio_bond, skewness_bond, kurtosis_bond], name='Bond')
    df_summary = pd.concat([summary_spy, summary_bond], axis=1)
    df_summary.index = ['Annualized Mean', 'Standard Deviation',
    'Sharpe Ratio', 'Skewness', 'Kurtosis']

    return df_summary

sum_stats(df_final)
```

Return to Report

A.2 Question 2

```
# Q2 - Divide Sample into 2 Sub-Samples
# Convert to Datetime
df_final['Date'] = pd.to_datetime(df_final['Date'])

# In-sample
startDate_insample = dt.datetime(1980, 1, 31)
endDate_insample = dt.datetime(1999, 12, 31)

# Out-of-sample
startDate_outsample = dt.datetime(2000, 1, 31)
endDate_outsample = dt.datetime(2021, 12, 31)

# Define function to split data into in sample & out of sample periods
def sample_split(df, startDate_insample, endDate_insample, startDate_outsample, endDate_outsample):

    df_insample = df[(df['Date'] >= startDate_insample) &
                     (df['Date'] <= endDate_insample)].reset_index(drop=True)

    df_outsample = df[(df['Date'] >= startDate_outsample) &
                      (df['Date'] <= endDate_outsample)].reset_index(drop=True)

    return df_insample, df_outsample
```

```python
# Function
def recurs_mean_forecast(df, startDate_insample, endDate_insample,
                         startDate_outsample, endDate_outsample):
    # Divide into in-sample & out of sample periods
    df_insample, df_outsample = sample_split(df, startDate_insample, endDate_insample,
                                             startDate_outsample, endDate_outsample)
    # Create DataFrame
    forecast_df = pd.DataFrame(columns=df.columns)
    for i in range(len(df_insample), len(df)):
        forecast_df.loc[i-len(df_insample)] = df.iloc[:i].mean(numeric_only=True)
    forecast_df['Date'] = df_outsample['Date'].reset_index(drop=True)

    return forecast_df

df_mean_forecast = recurs_mean_forecast(df_final, startDate_insample,
                                        endDate_insample, startDate_outsample, endDate_outsample)
```

Return to Report

A.3 Question 3

```python
# Set up DataFrame
df_var = pd.read_excel(r'PredictorData2022.xlsx')
df_var['yyyymm'] = pd.to_datetime(df_var['yyyymm'], format='%Y%m')
df_var.rename(columns={'yyyymm': 'Date'}, inplace=True)

df_var = df_var[(df_var['Date'] > startDate_insample - pd.DateOffset(months=2)) &
                (df_var['Date'] <= endDate_outsample)]

df_var.drop(columns=['Index', 'Date'], inplace=True)
df_var = df_var.dropna(axis=1).reset_index(drop=True)

df_var.insert(0, 'Date', df_final['Date'].reset_index(drop=True))

df_var = df_var.merge(df_final, on='Date', how='inner')

# Q3 - Test Correlation of Predictor Variables
pred_vars = ['tbl', 'infl', 'ltr',  'E12', 'ntis', 'Excess Returns (SPY)', 'Excess Returns (Bond)']
corr_matrix = df_var[pred_vars].rename(columns={
    'Excess Returns (SPY)': 'SPY', 'Excess Returns (Bond)': 'Bonds'}).corr()
cols = corr_matrix.columns
hm = sns.heatmap(corr_matrix, cbar=True, annot=True, square=True, fmt='.2f',
                 annot_kws={'size': 6}, yticklabels=cols, xticklabels=cols, cmap='rocket')
hm.set_title("Predictor Variables Correlation Heatmap")
plt.xticks(rotation='90')
plt.yticks(rotation='0')
plt.show()


def RecursiveLS_Ordinary(df_x, df_y, startDate_insample, endDate_insample,
                         startDate_outsample, endDate_outsample):

    # call sample_split() to split dfs into in & out of sample periods
    df_x_insample, df_x_outsample = sample_split(df_x, startDate_insample, endDate_insample,
                                               startDate_outsample, endDate_outsample)

    df_y_insample, df_y_outsample = sample_split(df_y, startDate_insample, endDate_insample,
                                               startDate_outsample, endDate_outsample)

    df_y_insample.drop(columns='Date', inplace=True)
    out_date = df_y_outsample.pop('Date')

    df_forecast_ols = pd.DataFrame()

    for variable in df_x_insample.drop(columns=['Date']).columns:

        forecast = pd.DataFrame()
        col_name = 'forecast: ' + variable

        # set up data for OLS
        df_x_insample_var = sm.add_constant(df_x_insample[variable])
        df_x_outsample_var = sm.add_constant(df_x_outsample[variable])
        df_y_insample_copy = df_y_insample.copy()

        for i, exog_i in enumerate(df_y_outsample.values):

            # extract endogenous variables of ith observation
            endog_i = df_x_outsample_var.loc[df_x_outsample_var.index==i]

            # Fit our OLS model on in-sample period
            model = sm.OLS(df_y_insample_copy, df_x_insample_var)
            res = model.fit()

            # append prediction to forecast
            forecast = pd.concat([forecast, res.predict(endog_i)], axis=0, ignore_index=True)

            # Include previous observation in next insample
            df_x_insample_var = pd.concat([df_x_insample_var, endog_i], axis=0, ignore_index=True)
            df_y_insample_copy = pd.concat([df_y_insample_copy, pd.DataFrame(
                exog_i, columns=df_y_insample_copy.columns)], axis=0, ignore_index=True)

        df_forecast_ols[col_name] = forecast

    df_forecast_ols.insert(0, 'Date', out_date)

    return df_forecast_ols

df_var_sp = df_var[pred_vars]
df_var_sp.insert(0, 'Date', df_var['Date'])

df_y = df_var[['Date', 'Excess Returns (SPY)']]

sp_forecast_ols = RecursiveLS_Ordinary(df_var_sp, df_y, startDate_insample, endDate_insample,
                                       startDate_outsample, endDate_outsample)
sp_forecast_ols['forecast: ensemble'] = sp_forecast_ols.mean(axis=1, numeric_only=True)

df_var_bond = df_var[pred_vars]
df_var_bond.insert(0, 'Date', df_var['Date'])

df_y = df_var[['Date', 'Excess Returns (Bond)']]

bond_forecast_ols = RecursiveLS_Ordinary(df_var_bond, df_y, startDate_insample, endDate_insample,
                                         startDate_outsample, endDate_outsample)
```

```python
bond_forecast_ols['forecast: ensemble'] = bond_forecast_ols.mean(axis=1, numeric_only=True)


# Define function to standardize a dataset
def Standardizer(df):
    scaler = StandardScaler()
    scaler.fit(df)
    df_standard = pd.DataFrame(scaler.transform(df), columns=df.columns)

    return df_standard, scaler

def RecursiveLS_Penalized(df_x, df_y, startDate_insample, endDate_insample,
                          startDate_outsample, endDate_outsample):

    # call sample_split() to split dfs into in & out of sample periods
    df_x_insample, df_x_outsample = sample_split(df_x, startDate_insample, endDate_insample,
                                                 startDate_outsample, endDate_outsample)

    df_y_insample, df_y_outsample = sample_split(df_y, startDate_insample, endDate_insample,
                                                  startDate_outsample, endDate_outsample)

    df_x_insample.drop(columns='Date', inplace=True)
    df_y_insample.drop(columns='Date', inplace=True)
    df_x_outsample.drop(columns='Date', inplace=True)
    out_date = df_y_outsample.pop('Date')

    df_forecast_pen = pd.DataFrame()

    for L1_wt in [0,1]: # 0 corresponds to ridge regression, 1 to lasso

        forecast = pd.DataFrame()
        if L1_wt == 0:
            col_name = 'forecast: ridge'
            alpha = 2
        else:
            col_name = 'forecast: lasso'
            alpha = 0.1

        for i, exog_i in enumerate(df_y_outsample.values):

            # Standardize exogenous variables
            exog_i = pd.DataFrame(exog_i, columns=df_y_insample.columns)
            df_y_insample_standard, scaler_y = Standardizer(df_y.iloc[:len(df_y_insample)+i,1:])
            exog_i = scaler_y.transform(exog_i)

            # extract variables of ith observation
            endog_i = df_x_outsample.iloc[i,:]

            # Standardize endogenous variables
            df_x_insample_standard, scaler_x = Standardizer(df_x.iloc[:len(df_y_insample)+i,1:])
            endog_i = scaler_x.transform(pd.DataFrame(endog_i.values.reshape(1, -1),
                                                      columns=df_x_insample.columns))

            # Fit our OLS model on in-sample period
            model = sm.OLS(df_y_insample_standard, df_x_insample_standard)
            res = model.fit_regularized(method='elastic_net', L1_wt=L1_wt, alpha=alpha)

            # Transform forecast back to raw returns & append to forecast df
            raw_forecast = scaler_y.mean_ + scaler_y.scale_ * pd.Series(res.predict(endog_i))
            forecast = pd.concat([forecast, raw_forecast], axis=0, ignore_index=True)

        df_forecast_pen[col_name] = forecast
    df_forecast_pen.insert(0, 'Date', out_date)

    return df_forecast_pen

df_y = df_var[['Date', 'Excess Returns (SPY)']]

sp_forecast_pen = RecursiveLS_Penalized(df_var_sp, df_y, startDate_insample, endDate_insample,
                                        startDate_outsample, endDate_outsample)

df_y = df_var[['Date', 'Excess Returns (Bond)']]

bond_forecast_pen = RecursiveLS_Penalized(df_var_bond, df_y, startDate_insample, endDate_insample,
                                          startDate_outsample, endDate_outsample)


# Q3 - DM-Test
# Define function to calculate DM-Test
def DM_test(df, actual, forecast1, forecast2):

    df_test=df.copy()
    # Calculate squared errors
    df_test['error1'] = (df_test[actual] - df_test[forecast1]) ** 2
    df_test['error2'] = (df_test[actual] - df_test[forecast2]) ** 2

    # Calculate MSFE ratio
    msfe_ratio = df_test['error1'].mean() / df_test['error2'].mean()

    # Autocorrelation Function
    h = len(df_test)**(1/3)
    h_lags = int(h)

    acf = sm.tsa.stattools.acf(df_test['error1'] - df_test['error2'], nlags=h_lags, fft=False)

    gamma = (acf[1:].sum() * 2) + 1

    # Calculate test statistic
    d_bar = (df_test['error1'] - df_test['error2']).mean()
    test_stat = np.divide(d_bar, np.sqrt(gamma / len(df_test)))

    # Calculate p-value
    p_value = 1 - norm.cdf(test_stat)

    return test_stat, p_value, msfe_ratio

# Q3 - Prepare DataFrame for DM-Test (SPY)
df_y = df_var[['Date', 'Excess Returns (SPY)']].copy()
sp_forecast = df_y.merge(df_mean_forecast[['Date', 'Excess Returns (SPY)']], how='inner',
                         on='Date').merge(sp_forecast_ols, how='inner',
                            on='Date').merge(sp_forecast_pen, how='inner', on='Date')

sp_forecast.rename(columns={'Excess Returns (SPY)_x': 'SPY',
        'Excess Returns (SPY)_y': 'forecast: mean'}, inplace=True)

# Prepare DataFrames for DM-Test Storage
dm_test_stats_spy = pd.DataFrame(columns=['Model (SPY)',  'MSFE Ratio', 'Test Statistic', 'p-value'])
```

```
# Run DM-Test for each model
for column in sp_forecast.columns[3:]:

    test_stat, p_value, msfe_ratio = DM_test(sp_forecast, 'SPY', 'forecast: mean', column)
    dm_test_stats_spy = pd.concat([dm_test_stats_spy, pd.Series([column, msfe_ratio,
        test_stat, p_value], index=dm_test_stats_spy.columns).to_frame().T], ignore_index=True)

# Q3 - Prepare DataFrame for DM-Test (Bond)
df_y = df_var[['Date', 'Excess Returns (Bond)']].copy()
bond_forecast = df_y.merge(df_mean_forecast[['Date', 'Excess Returns (Bond)']], how='inner',
                            on='Date').merge(bond_forecast_ols, how='inner',
                                             on='Date').merge(bond_forecast_pen, how='inner', on='Date')

bond_forecast.rename(columns={'Excess Returns (Bond)_x': 'Bond',
                              'Excess Returns (Bond)_y': 'forecast: mean'}, inplace=True)

# Prepare DataFrames for DM-Test Storage
dm_test_stats_bond = pd.DataFrame(columns=['Model (Bond)',  'MSFE Ratio', 'Test Statistic', 'p-value'])

# Run DM-Test for each model
for column in bond_forecast.columns[3:]:

    test_stat, p_value, msfe_ratio = DM_test(bond_forecast, 'Bond', 'forecast: mean', column)
    dm_test_stats_bond = pd.concat([dm_test_stats_bond, pd.Series([column, msfe_ratio,
        test_stat, p_value], index=dm_test_stats_bond.columns).to_frame().T], ignore_index=True)


# Plotting forecasts vs. actual Returns
df_y = df_var[['Date', 'Excess Returns (SPY)']].copy()
sp_forecast = df_y.merge(df_mean_forecast[['Date', 'Excess Returns (SPY)']], how='inner',
                         on='Date').merge(sp_forecast_ols[['Date', 'forecast: ensemble']], how='inner',
                                          on='Date').merge(sp_forecast_pen, how='inner', on='Date')

sp_forecast.rename(columns={'Excess Returns (SPY)_x': 'SPY',
                            'Excess Returns (SPY)_y': 'forecast: mean'}, inplace=True)

sp_forecast.plot(kind='line', x='Date', colormap='viridis', alpha=0.8)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.xticks(rotation=45)
plt.ylabel('Monthly Return')
plt.title('Observed vs. Forecast Monthly Returns on SPY (Recursive Window)')
plt.show()

df_y = df_var[['Date', 'Excess Returns (Bond)']].copy()
bond_forecast = df_y.merge(df_mean_forecast[['Date', 'Excess Returns (Bond)']],
                    how='inner', on='Date').merge(bond_forecast_ols[['Date', 'forecast: ensemble']],
                        how='inner', on='Date').merge(bond_forecast_pen, how='inner', on='Date')

bond_forecast.rename(columns={'Excess Returns (Bond)_x': 'Bond',
                               'Excess Returns (Bond)_y': 'forecast: mean'}, inplace=True)

bond_forecast.plot(kind='line', x='Date', colormap='viridis', alpha=0.8)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.xticks(rotation=45)
plt.ylabel('Monthly Return')
plt.title('Observed vs. Forecast Monthly Returns on Bonds (Recursive Window)')
plt.show()
```

Return to Report

A.4 Question 4

```
# Function
def recurs_covar_forecast(df, startDate_insample, endDate_insample,
                          startDate_outsample, endDate_outsample):
    # Divide into in-sample & out of sample periods
    df_insample, df_outsample = sample_split(df, startDate_insample,
                                             endDate_insample, startDate_outsample, endDate_outsample)

    # Create DataFrame
    forecast_arr = np.empty(shape=(1,4))
    for i in range(len(df_insample), len(df)):
        forecast_arr = np.concatenate((forecast_arr, df.drop(
            columns='Date').iloc[:i].cov().values.reshape(1,4)), axis=0)

    forecast_arr = forecast_arr[1:,:]
    forecast_df = pd.DataFrame(forecast_arr)
    forecast_df.columns = ['var_x', 'cov_xy', 'cov_yx', 'var_y']
    forecast_df.insert(0, 'Date', df_outsample['Date'])

    return forecast_df

df_covar_forecast = recurs_covar_forecast(df_final, startDate_insample, endDate_insample,
                                          startDate_outsample, endDate_outsample)
```

Return to Report

A.5 Question 5

```
# Q5 - Prepare DataFrames
sp_forecast = df_mean_forecast[['Date', 'Excess Returns (SPY)']].merge(
    sp_forecast_ols, how='inner', on='Date').merge(sp_forecast_pen, how='inner', on='Date')
sp_forecast.rename(columns={'Excess Returns (SPY)': 'forecast: mean'}, inplace=True)

bond_forecast = df_mean_forecast[['Date', 'Excess Returns (Bond)']].merge(
    bond_forecast_ols, how='inner', on='Date').merge(bond_forecast_pen, how='inner', on='Date')
bond_forecast.rename(columns={'Excess Returns (Bond)': 'forecast: mean'}, inplace=True)

# Q5
def tangency_portfolio_weights(df_mean, df_covar):

    # Drop Date & Convert to numpy arrays
    arr_mu = df_mean.drop(columns='Date').to_numpy()
    arr_cov = df_covar.drop(columns='Date').to_numpy()

    opt_t_weights = pd.DataFrame(columns=['Date', 'w1', 'w2'])
    opt_t_weights['Date'] = df_covar['Date'].copy()

    # Reshape arrays and Calculate weights
    for i in range(0, len(df_covar)):
```

```python
        mu = arr_mu[i].reshape(2,1)
        cov = arr_cov[i].reshape(2,2)

        num = np.linalg.inv(cov).dot(mu)
        denom = np.ones(2).T @ np.linalg.inv(cov).dot(mu)
        w_tp = np.divide(num, denom)
        w_tp = w_tp.reshape(2,)

        opt_t_weights.loc[i,['w1', 'w2']] = w_tp.T
    return opt_t_weights

opt_t_weights = tangency_portfolio_weights(df_mean_forecast, df_covar_forecast)

# Q5 - Annualised Summary Stats
# Q5 - Compute Annualised Summary Stats of Optimal Portfolio's Excess Returns
def annualised_portfolio_stats(weights, excess_returns):

    df_returns = pd.DataFrame()
    df_returns['Portfolio Excess Returns'] = (excess_returns.iloc[:,1]*weights.iloc[:,1] +
                                             excess_returns.iloc[:,2]*weights.iloc[:,2])

    # Annualised Mean from Monthly Excess Returns
    port_mean = (1 + df_returns['Portfolio Excess Returns'].mean())**12 - 1

    # Annualised Standard Deviation from Monthly Excess Returns
    port_std_dev = df_returns['Portfolio Excess Returns'].std() * np.sqrt(12)

    # Annualised Sharpe Ratio from Monthly Excess Returns
    port_sharpe_ratio = port_mean / port_std_dev

    return port_mean, port_std_dev, port_sharpe_ratio

excess_returns = df_var[['Date', 'Excess Returns (SPY)', 'Excess Returns (Bond)']]

stats_lst = []
for column in sp_forecast.columns[1:]:

    df_mean_forecast = sp_forecast[['Date', column]].merge(bond_forecast[['Date', column]],
                                           on='Date', how='inner', suffixes=('_SPY', '_Bond'))
    weights = tangency_portfolio_weights(df_mean_forecast, df_covar_forecast)
    stats_lst.append(annualised_portfolio_stats(weights, excess_returns))

df_tangency = pd.DataFrame(stats_lst, index=sp_forecast.columns[1:],
                           columns=['Mean Excess Return', 'Std. Dev of Excess Return', 'Sharpe Ratio'])

# sp_forecast
def cumulative_excess_return(weights, excess_returns):

    df_returns = pd.DataFrame()
    df_returns['Date'] = weights.iloc[:,0]
    df_returns['Weight (SPY)'] = weights.iloc[:,1]
    df_returns['Weight (Bond)'] = weights.iloc[:,2]
    df_returns['Cumulative Excess Returns'] = (excess_returns.iloc[:,1]*weights.iloc[:,1] +
                            excess_returns.iloc[:,2]*weights.iloc[:,2]+1).cumprod()-1

    return df_returns

fig, axs = plt.subplots(2, 2, figsize=(20,12))
fig.suptitle('Time Series of Estimates of Weights & Cumulative Return (Recursive Window)', fontsize=32)
```

```
forecasts = ['forecast: mean', 'forecast: ensemble', 'forecast: ridge', 'forecast: lasso']
for forecast, ax in zip(forecasts, axs.ravel()):
    df_mean_forecast = sp_forecast[['Date', forecast]].merge(bond_forecast[['Date', forecast]],
                                          on='Date', how='inner', suffixes=('_SPY', '_Bond'))
    weights = tangency_portfolio_weights(df_mean_forecast, df_covar_forecast)

    cumulative_excess_return(weights, df_mean_forecast).plot(kind='line', x='Date',
                                                            cmap='viridis', ax=ax, legend=False)
    ax.set_ylim((-0.6,1.6))
    ax.set_title(forecast.title(), fontsize=16)
    # ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.legend(['Weight (SPY)','Weight (Bond)','Cumulative Return'], bbox_to_anchor=(0.4, -0.1),
                                                                       ncol=3, fontsize=16)
plt.show()
```

Return to Report

A.6 Question 6

```
# Function
def rolling_mean_forecast(df, startDate_insample, endDate_insample,
                                startDate_outsample, endDate_outsample):

    # Divide into in-sample & out of sample periods
    df_insample, df_outsample = sample_split(df, startDate_insample,
                endDate_insample, startDate_outsample, endDate_outsample)
    window = len(df_insample)

    # Create DataFrame
    forecast_df = pd.DataFrame(columns=df.columns)
    forecast_df[df.columns[1:]] = df[df.columns[1:]].rolling(window=window).mean(
        numeric_only=True)[window:].reset_index(drop=True)
    forecast_df['Date'] = df_outsample['Date'].reset_index(drop=True)

    return forecast_df

df_mean_forecast_roll = rolling_mean_forecast(df_final, startDate_insample, endDate_insample,
                                              startDate_outsample, endDate_outsample)

def RollingLS_Ordinary(df_x, df_y, startDate_insample, endDate_insample,
                       startDate_outsample, endDate_outsample):

    # call sample_split() to split dfs into in & out of sample periods
    df_x_insample, df_x_outsample = sample_split(df_x, startDate_insample, endDate_insample,
                                                 startDate_outsample, endDate_outsample)
    df_y_insample, df_y_outsample = sample_split(df_y, startDate_insample, endDate_insample,
                                                 startDate_outsample, endDate_outsample)

    df_y_insample.drop(columns='Date', inplace=True)
    out_date = df_y_outsample.pop('Date')

    df_forecast_ols = pd.DataFrame()
    for variable in df_x_insample.drop(columns=['Date']).columns:

        forecast = pd.DataFrame()
        col_name = 'forecast: ' + variable

        # set up data for OLS
        df_x_insample_var = sm.add_constant(df_x_insample[variable])
        df_x_outsample_var = sm.add_constant(df_x_outsample[variable])
        df_y_insample_copy = df_y_insample.copy()

        for i, exog_i in enumerate(df_y_outsample.values):

            # extract endogenous variables of ith observation
            endog_i = df_x_outsample_var.loc[df_x_outsample_var.index==i]

            # Fit our OLS model on in-sample period
            model = sm.OLS(df_y_insample_copy, df_x_insample_var)
            res = model.fit()

            # append prediction to forecast
            forecast = pd.concat([forecast, res.predict(endog_i)], axis=0, ignore_index=True)

            # Include previous observation in next insample
            df_x_insample_var = pd.concat([df_x_insample_var, endog_i], axis=0, ignore_index=True)
            df_y_insample_copy = pd.concat([df_y_insample_copy,
            pd.DataFrame(exog_i, columns=df_y_insample_copy.columns)], axis=0, ignore_index=True)

        df_forecast_ols[col_name] = forecast

    df_forecast_ols.insert(0, 'Date', out_date)

    return df_forecast_ols

df_y = df_var[['Date', 'Excess Returns (SPY)']]

sp_forecast_ols_roll = RollingLS_Ordinary(df_var_sp, df_y, startDate_insample, endDate_insample,
                                          startDate_outsample, endDate_outsample)
sp_forecast_ols_roll['forecast: ensemble'] = sp_forecast_ols_roll.mean(axis=1, numeric_only=True)

df_y = df_var[['Date', 'Excess Returns (Bond)']]

bond_forecast_ols_roll = RollingLS_Ordinary(df_var_bond, df_y, startDate_insample, endDate_insample,
                                            startDate_outsample, endDate_outsample)
bond_forecast_ols_roll['forecast: ensemble'] = bond_forecast_ols_roll.mean(axis=1, numeric_only=True)

def RollingLS_Penalized(df_x, df_y, startDate_insample, endDate_insample,
                        startDate_outsample, endDate_outsample):

    # call sample_split() to split dfs into in & out of sample periods
    df_x_insample, df_x_outsample = sample_split(df_x, startDate_insample, endDate_insample,
                                                 startDate_outsample, endDate_outsample)
    df_y_insample, df_y_outsample = sample_split(df_y, startDate_insample, endDate_insample,
                                                 startDate_outsample, endDate_outsample)

    df_x_insample.drop(columns='Date', inplace=True)
    df_y_insample.drop(columns='Date', inplace=True)
    df_x_outsample.drop(columns='Date', inplace=True)
    out_date = df_y_outsample.pop('Date')
    df_forecast_pen = pd.DataFrame()

    for L1_wt in [0,1]: # 0 corresponds to ridge regression, 1 to lasso

        forecast = pd.DataFrame()
        if L1_wt == 0:
            col_name = 'forecast: ridge'
            alpha = 2
        else:
            col_name = 'forecast: lasso'
            alpha = 0.12

        for i, exog_i in enumerate(df_y_outsample.values):

            # Standardize exogenous variables
            exog_i = pd.DataFrame(exog_i, columns=df_y_insample.columns)
            df_y_insample_standard, scaler_y = Standardizer(df_y.iloc[i:len(df_y_insample)+i,1:])
            exog_i = scaler_y.transform(exog_i)

            # extract variables of ith observation
            endog_i = df_x_outsample.iloc[i,:]

            # Standardize endogenous variables
            df_x_insample_standard, scaler_x = Standardizer(df_x.iloc[i:len(df_y_insample)+i,1:])
            endog_i = scaler_x.transform(pd.DataFrame(endog_i.values.reshape(1, -1),
                                                      columns=df_x_insample.columns))

            # Fit our OLS model on in-sample period
            model = sm.OLS(df_y_insample_standard, df_x_insample_standard)
            res = model.fit_regularized(method='elastic_net', L1_wt=L1_wt, alpha=alpha)

            # Transform forecast back to raw returns & append to forecast df
            raw_forecast = scaler_y.mean_ + scaler_y.scale_ * pd.Series(res.predict(endog_i))
            forecast = pd.concat([forecast, raw_forecast], axis=0, ignore_index=True)

        df_forecast_pen[col_name] = forecast
    df_forecast_pen.insert(0, 'Date', out_date)

    return df_forecast_pen

df_y = df_var[['Date', 'Excess Returns (SPY)']]

sp_forecast_pen_roll = RollingLS_Penalized(df_var_sp, df_y, startDate_insample, endDate_insample,
                                           startDate_outsample, endDate_outsample)

df_y = df_var[['Date', 'Excess Returns (Bond)']]

bond_forecast_pen_roll = RollingLS_Penalized(df_var_bond, df_y, startDate_insample, endDate_insample,
                                             startDate_outsample, endDate_outsample)

df_y = df_var[['Date', 'Excess Returns (SPY)']].copy()
sp_forecast_roll = df_y.merge(df_mean_forecast_roll[['Date', 'Excess Returns (SPY)']],
how='inner', on='Date').merge(
    sp_forecast_ols_roll[['Date', 'forecast: ensemble']], how='inner',
                        on='Date').merge(sp_forecast_pen_roll, how='inner', on='Date')

sp_forecast_roll.rename(columns={'Excess Returns (SPY)_x': 'SPY',
                                 'Excess Returns (SPY)_y': 'forecast: mean'}, inplace=True)
```

```python
sp_forecast_roll.plot(kind='line', x='Date', colormap='viridis', alpha=0.8)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.xticks(rotation=45)
plt.ylabel('Monthly Return')
plt.title('Observed vs. Forecast Monthly Returns on SPY (Rolling Window)')
plt.show()

df_y = df_var[['Date', 'Excess Returns (Bond)']].copy()
bond_forecast_roll = df_y.merge(df_mean_forecast_roll[['Date', 'Excess Returns (Bond)']],
how='inner', on='Date').merge(
    bond_forecast_ols_roll[['Date', 'forecast: ensemble']], how='inner',
                            on='Date').merge(bond_forecast_pen_roll, how='inner', on='Date')

bond_forecast_roll.rename(columns={'Excess Returns (Bond)_x': 'Bond',
                                   'Excess Returns (Bond)_y': 'forecast: mean'}, inplace=True)

bond_forecast_roll.plot(kind='line', x='Date', colormap='viridis', alpha=0.8)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.xticks(rotation=45)
plt.ylabel('Monthly Return')
plt.title('Observed vs. Forecast Monthly Returns on Bonds (Rolling Window)')
plt.show()

# Function
def rolling_covar_forecast(df, startDate_insample, endDate_insample, startDate_outsample,
                           endDate_outsample):
    # Divide into in-sample & out of sample periods
    df_insample, df_outsample = sample_split(df, startDate_insample, endDate_insample,
                                             startDate_outsample, endDate_outsample)

    forecast_df = df.drop(columns='Date').rolling(window=len(df_insample),
                                          closed='left').cov().unstack().dropna()

    forecast_df.columns = ['cov_xy', 'var_x', 'var_y', 'cov_yx']
    # reorder columns
    cols = ['var_x', 'cov_xy', 'cov_yx', 'var_y']
    forecast_df = forecast_df[cols].reset_index(drop=True)
    forecast_df.insert(0, 'Date', df_outsample['Date'])

    return forecast_df

df_covar_forecast_roll = rolling_covar_forecast(df_final, startDate_insample, endDate_insample,
                                                startDate_outsample, endDate_outsample)

# Q6 - Prepare DataFrames
sp_forecast_roll = df_mean_forecast_roll[['Date', 'Excess Returns (SPY)']].merge(
    sp_forecast_ols_roll, how='inner', on='Date').merge(sp_forecast_pen_roll, how='inner', on='Date')
sp_forecast_roll.rename(columns={'Excess Returns (SPY)': 'forecast: mean'}, inplace=True)

bond_forecast_roll = df_mean_forecast_roll[['Date', 'Excess Returns (Bond)']].merge(
    bond_forecast_ols_roll, how='inner', on='Date').merge(bond_forecast_pen_roll, how='inner', on='Date')
bond_forecast_roll.rename(columns={'Excess Returns (Bond)': 'forecast: mean'}, inplace=True)

opt_t_weights_roll = tangency_portfolio_weights(df_mean_forecast_roll, df_covar_forecast_roll)

# Q6 - Compute Annualised Summary Stats of Optimal Portfolio's Excess Returns
def annualised_portfolio_stats(weights, excess_returns):

    df_returns = pd.DataFrame()
    df_returns['Portfolio Excess Returns'] = (excess_returns.iloc[:,1]*weights.iloc[:,1] +
                                              excess_returns.iloc[:,2]*weights.iloc[:,2])

    # Annualised Mean from Monthly Excess Returns
    port_mean = (1 + df_returns['Portfolio Excess Returns'].mean())**12 - 1

    # Annualised Standard Deviation from Monthly Excess Returns
    port_std_dev = df_returns['Portfolio Excess Returns'].std() * np.sqrt(12)

    # Annualised Sharpe Ratio from Monthly Excess Returns
    port_sharpe_ratio = port_mean / port_std_dev

    return port_mean, port_std_dev, port_sharpe_ratio

excess_returns = df_var[['Date', 'Excess Returns (SPY)', 'Excess Returns (Bond)']]

stats_lst = []
for column in sp_forecast_roll.columns[1:]:

   df_mean_forecast_roll = sp_forecast_roll[['Date', column]].merge(bond_forecast_roll[['Date', column]],
                               on='Date', how='inner', suffixes=('_SPY', '_Bond'))
   weights_roll = tangency_portfolio_weights(df_mean_forecast_roll, df_covar_forecast_roll)
   stats_lst.append(annualised_portfolio_stats(weights_roll, excess_returns))

df_tangency_roll = pd.DataFrame(stats_lst, index=sp_forecast_roll.columns[1:],
                               columns=['Mean Excess Return', 'Std. Dev of Excess Return', 'Sharpe Ratio'])


fig, axs = plt.subplots(2, 2, figsize=(20,12))
fig.suptitle('Time Series of Estimates of Weights & Cumulative Return (Rolling Window)', fontsize=32)

forecasts = ['forecast: mean', 'forecast: ensemble', 'forecast: ridge', 'forecast: lasso']
for forecast, ax in zip(forecasts, axs.ravel()):
    df_mean_forecast_roll = sp_forecast_roll[['Date', forecast]].merge(bond_forecast_roll[['Date',
                               forecast]], on='Date', how='inner', suffixes=('_SPY', '_Bond'))
    weights_roll = tangency_portfolio_weights(df_mean_forecast_roll, df_covar_forecast_roll)

    cumulative_excess_return(weights_roll, df_mean_forecast_roll).plot(kind='line', x='Date',
                               cmap='viridis', ax=ax, legend=False)
    ax.set_ylim((-0.6,1.6))
    ax.set_title(forecast.title(), fontsize=16)

plt.legend(['Weight (SPY)','Weight (Bond)','Cumulative Return'], bbox_to_anchor=(0.4, -0.1),
            ncol=3, fontsize=16)
plt.show()
```

Return to Report