Econ 144 Project 2 (Google and S&P 500)

Alison Hui and Agastya Rao

2025-02-23

Introduction

In project 2, we examined the historical price movements of Google (GOOG) and the S&P 500 ETF (SPY) to develop predictive models for financial markets. Google represents a high-growth technology stock, while SPY, which tracks the broader S&P 500 index, serves as a benchmark for overall market performance. This combination allows us to compare individual stock behavior against broader market trends, helping us assess forecasting reliability across different asset types.

For this analysis, we use monthly closing prices sourced from Yahoo Finance, a widely used and reliable financial data provider. The training dataset spans January 2010 to December 2023, covering significant economic events such as the post-2008 financial crisis recovery, the rise of the technology sector, the 2020 COVID-19 market crash, and subsequent economic fluctuations. The testing period, January 2024 to December 2024, allows us to evaluate model performance in real-time market conditions, considering macroeconomic factors like interest rate changes, inflation, and global economic uncertainties. The choice of monthly data helps smooth out short-term volatility while capturing long-term trends relevant to investors and policymakers.

Results

Let's load the libraries we will be using:

```r
library(quantmod)
```

```
## Loading required package: xts

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: TTR

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```r
library(vars)
```

```
## Loading required package: MASS

## Loading required package: strucchange

## Loading required package: sandwich

## Loading required package: urca
## Loading required package: lmtest
```

```r
library(tseries)
library(dplyr)
```

```
##
## ######################### Warning from 'xts' package ##########################
## #                                                                             #
## # The dplyr lag() function breaks how base R's lag() function is supposed to  #
## # work, which breaks lag(my_xts). Calls to lag(my_xts) that you type or       #
## # source() into this session won't work correctly.                            #
## #                                                                             #
## # Use stats::lag() to make sure you're not using dplyr::lag(), or you can add #
## # conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop           #
## # dplyr from breaking base R's lag() function.                                #
## #                                                                             #
## # Code in packages is not affected. It's protected by R's namespace mechanism #
## # Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this warning.  #
## #                                                                             #
## ###############################################################################
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:MASS':
##
##     select
```

```
## The following objects are masked from 'package:xts':
##
##     first, last
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(forecast)
library(ggplot2)
library(lmtest)
library(Metrics)
```

```
##
## Attaching package: 'Metrics'
```

```
## The following object is masked from 'package:forecast':
##
##     accuracy
```

Let's now extract the training (Jan 2010 - December 2023) and testing data (Jan 2024 - December 2024):

```r
# Extract TRAINING DATA (ending in 2023)
getSymbols(c("GOOG", "SPY"), src = "yahoo", from = "2010-01-01", to = "2023-12-31")
```

```
## [1] "GOOG" "SPY"
```

```r
Goog_prices <- GOOG[, "GOOG.Adjusted"]
SPY_prices <- SPY[, "SPY.Adjusted"]

Goog_monthly <- to.monthly(Goog_prices, indexAt = "lastof", OHLC = FALSE)
SPY_monthly <- to.monthly(SPY_prices, indexAt = "lastof", OHLC = FALSE)

Goog_ts <- ts(as.numeric(Goog_monthly), start = c(2010, 1), frequency = 12)
SPY_ts <- ts(as.numeric(SPY_monthly), start = c(2010, 1), frequency = 12)

# Extract ACTUAL DATA (for 2024)
getSymbols(c("GOOG", "SPY"), src = "yahoo", from = "2024-01-01", to = "2024-12-31")
```

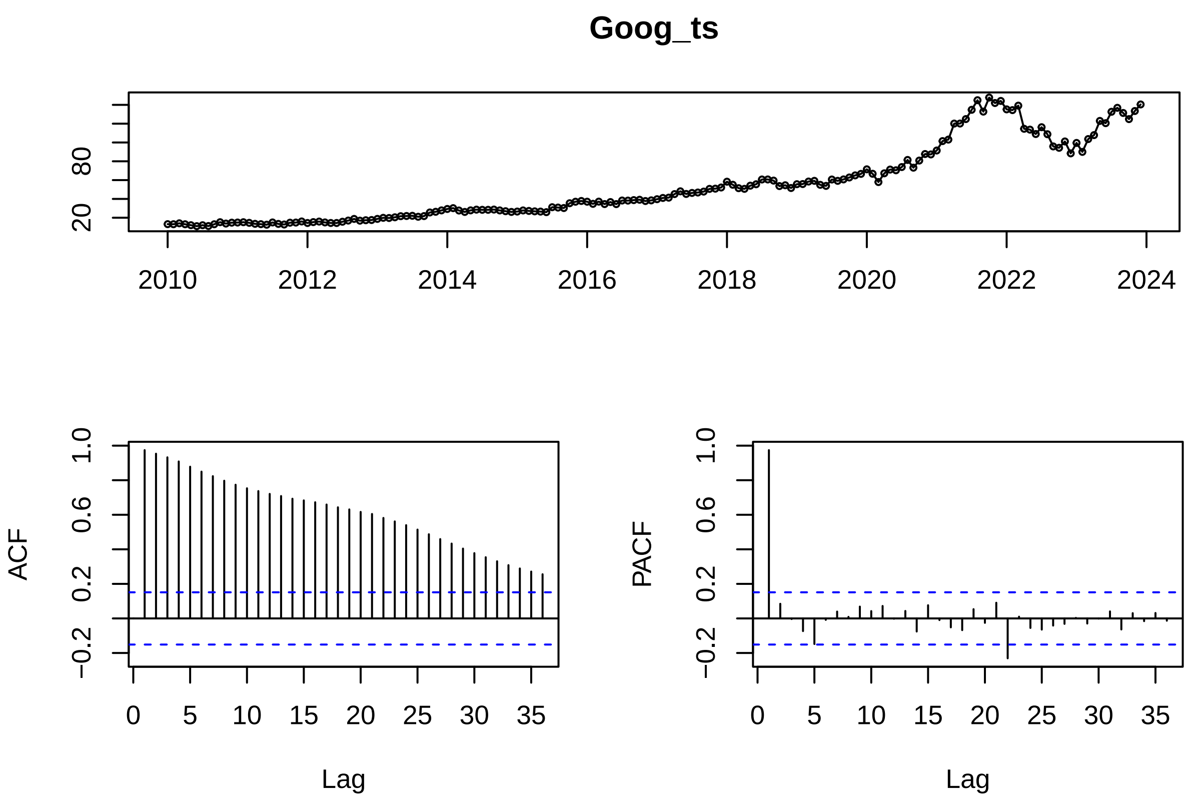
```
## [1] "GOOG" "SPY"
```

```r
Goog_actual_prices <- GOOG[, "GOOG.Adjusted"]
SPY_actual_prices <- SPY[, "SPY.Adjusted"]

Goog_actual_monthly <- to.monthly(Goog_actual_prices, indexAt = "lastof", OHLC = FALSE)
SPY_actual_monthly <- to.monthly(SPY_actual_prices, indexAt = "lastof", OHLC = FALSE)

actual_Goog <- ts(as.numeric(Goog_actual_monthly), start = end(Goog_ts) + c(0, 1), frequency = 12)
actual_SPY <- ts(as.numeric(SPY_actual_monthly), start = end(SPY_ts) + c(0, 1), frequency = 12)
```

Part a

Let's see what the Google time series data looks like, along with the ACF and PACF plots:

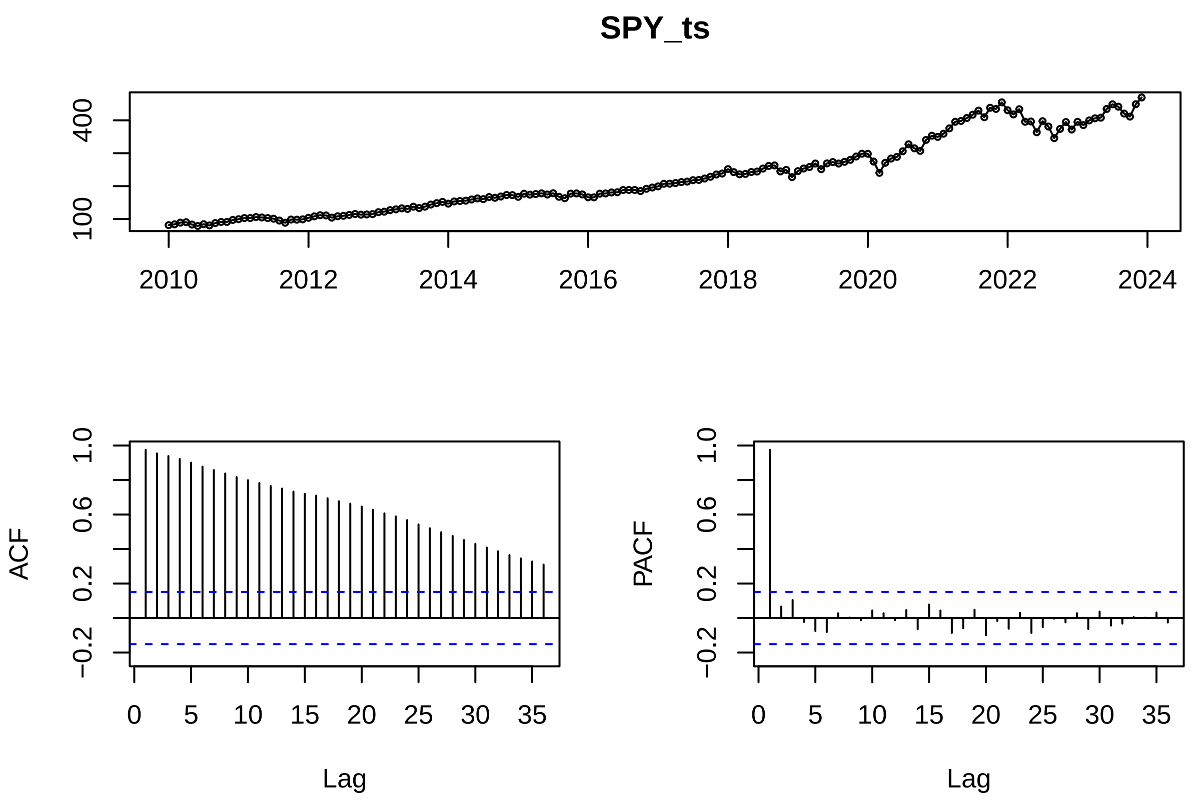
```r
tsdisplay(Goog_ts)
```



The GOOG time series shows a strong upward trend, indicating non-stationarity. The ACF exhibits a slow decay, confirming that first-order differencing is needed. The PACF has a significant spike at lag 1, suggesting an AR(1) process. Based on this, an ARIMA(1,1,0) model is a good starting point, but further checks on stationarity and residuals will be done later on to refine the model.

Let's now see what the S&P 500 time series data looks like, along with the ACF and PACF plots:

```
tsdisplay(SPY_ts)
```

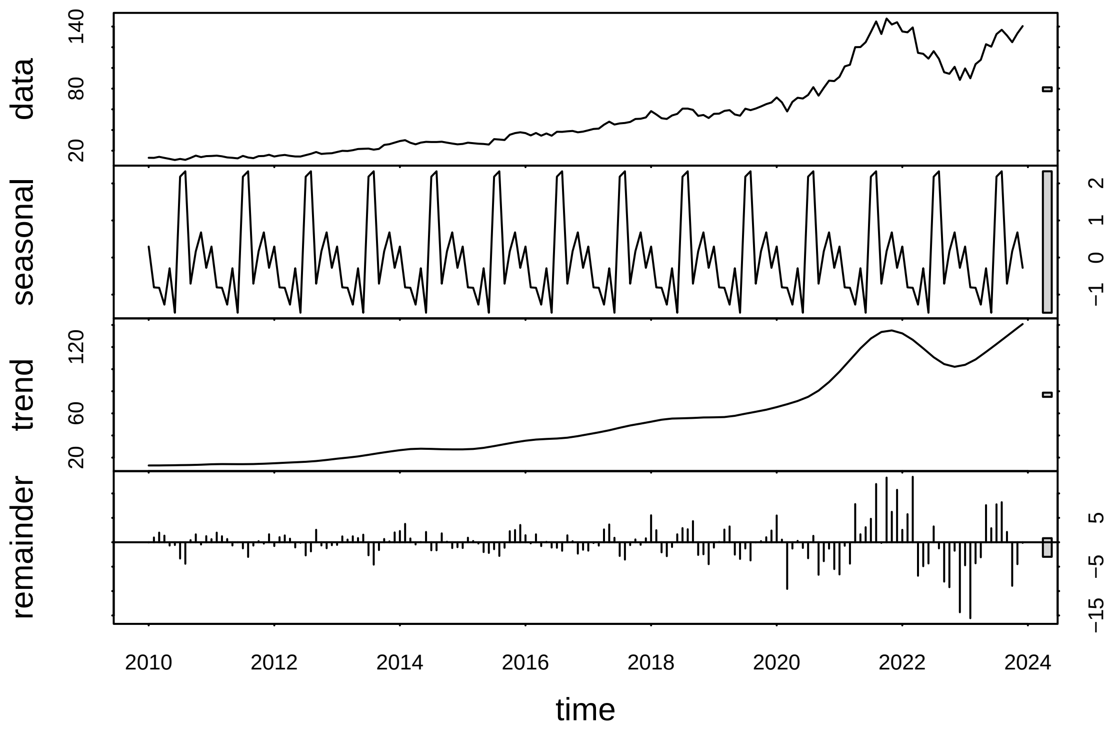


The S&P 500 time series also exhibits a strong upward trend, indicating non-stationarity. The ACF shows a slow decay, confirming the need for first-order differencing (d=1). The PACF has a significant spike at lag 1, suggesting an AR(1) process. Based on this, an ARIMA(1,1,0) model is a reasonable starting point. However, further differencing checks and residual analysis will be performed to assess the model.

Part b

Let's produce an STL decomposition for Google:

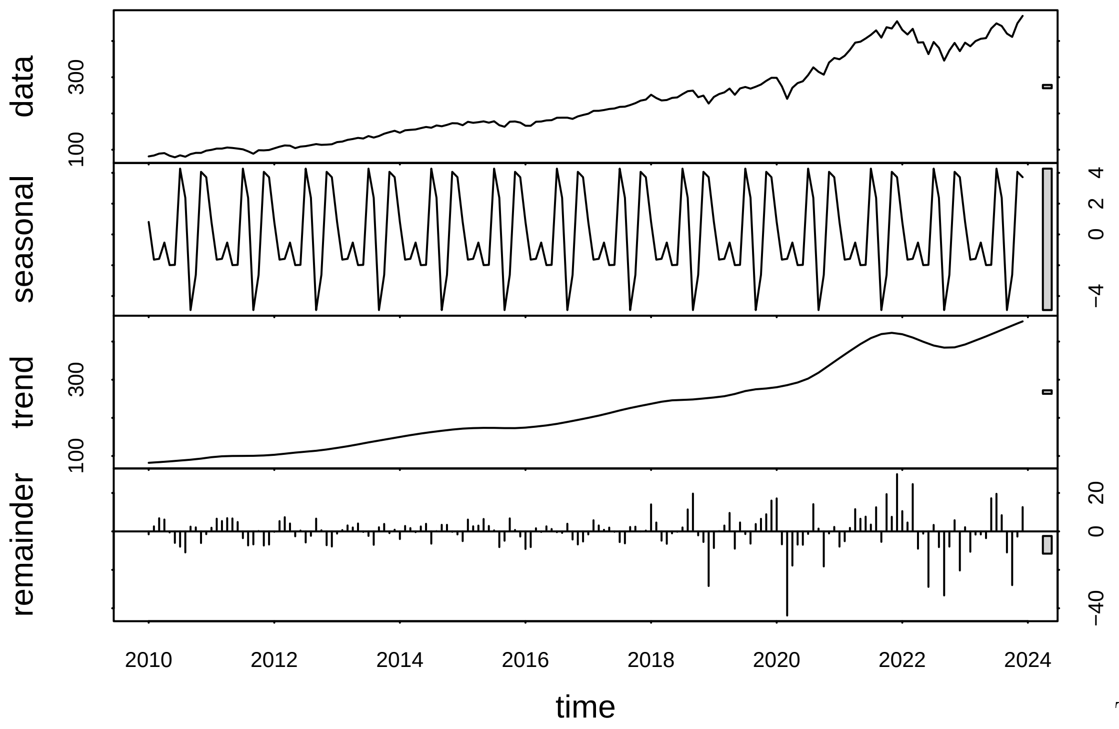
```
stl_Goog <- stl(Goog_ts, s.window = "periodic")
plot(stl_Goog)
```



The time series decomposition highlights a pronounced upward trend, suggesting that the data is non-stationary and requires first-order differencing (d=1) for accurate modeling. The seasonal component exhibits a distinct annual pattern, reinforcing the necessity of seasonal differencing (D=1) in a SARIMA model. Additionally, the remainder (residual) component displays some cyclic fluctuations, indicating potential unmodeled seasonality or external factors influencing the data.

Let's now produce an STL decomposition for S&P 500:

```r
stl_SPY <- stl(SPY_ts, s.window = "periodic")
plot(stl_SPY)
```



The time series decomposition reveals a strong upward trend, indicating that the data is non-stationary and requires first-order differencing (d=1) for proper modeling. Additionally, the seasonal component shows a clear annual pattern, confirming the need for seasonal differencing (D=1) in a SARIMA model. The remainder (residual) component shows some cyclic behavior, indicating that there may still be some unmodeled seasonality or external influences.

Part c

Based on the time series plot, ACF/PACF analysis, and decomposition, a reasonable model for the Google time series would SARIMA(1,1,0)(0,1,0). The strong upward trend in the data indicates non-stationarity, requiring first-order differencing (d=1) to stabilize the mean. The seasonal decomposition shows a clear annual pattern, confirming the need for seasonal differencing (D=1) to remove periodic fluctuations. The PACF plot exhibits a significant lag-1 spike, suggesting an AR(1) process, while the ACF does not show a sharp cutoff, indicating that an MA component is unnecessary (q=0). Therefore, SARIMA(1,1,0)(0,1,0)[12] seems to be a suitable model for capturing both trend and seasonality in the data:

```r
arima_Goog <- Arima(Goog_ts, order = c(1,1,0), seasonal = list(order = c(0,1,0), period = 12))
arima_Goog
```

```
## Series: Goog_ts
## ARIMA(1,1,0)(0,1,0)[12]
##
## Coefficients:
##          ar1
##       0.0139
## s.e.  0.0822
##
## sigma^2 = 48.34:  log likelihood = -520
## AIC=1044   AICc=1044.07   BIC=1050.08
```

As discussed earlier, the S&P 500 time series exhibits a clear upward trend, indicating non-stationarity and the need for first-order differencing (d=1). The ACF plot shows a slow decay, reinforcing the necessity of differencing, while the PACF plot has a significant spike at lag 1, suggesting an AR(1) component. The seasonal decomposition reveals a strong annual pattern, confirming the need for seasonal differencing (D=1) to capture repeating cycles. Given these characteristics, an appropriate model would be SARIMA(1,1,0)(0,1,0)[12], which accounts for both trend and seasonality.

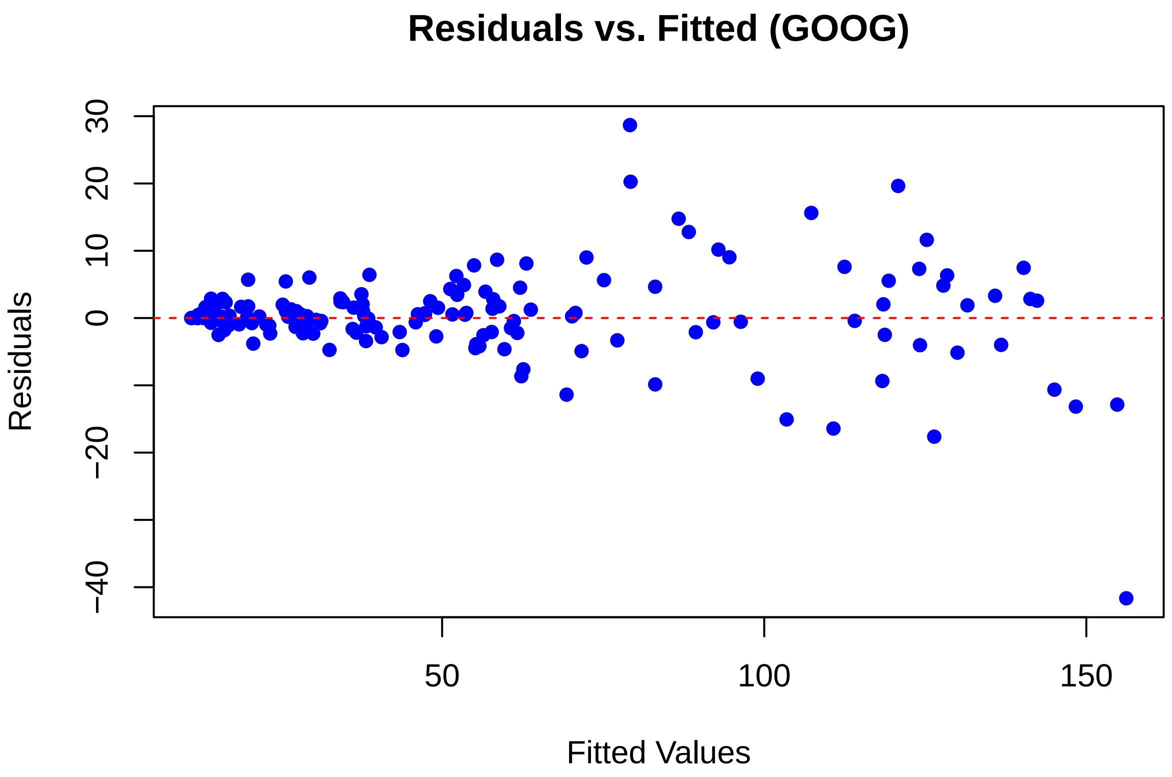
```r
arima_SPY <-Arima(SPY_ts, order = c(1,1,0), seasonal = list(order = c(0,1,0), period = 12))
arima_SPY
```

```
## Series: SPY_ts
## ARIMA(1,1,0)(0,1,0)[12]
##
## Coefficients:
##           ar1
##       -0.1442
## s.e.   0.0810
##
## sigma^2 = 274.6:  log likelihood = -654.64
## AIC=1313.27   AICc=1313.35   BIC=1319.36
```

Part e

Let's plot the residuals vs. fitted Values for the Google SARIMA Model:

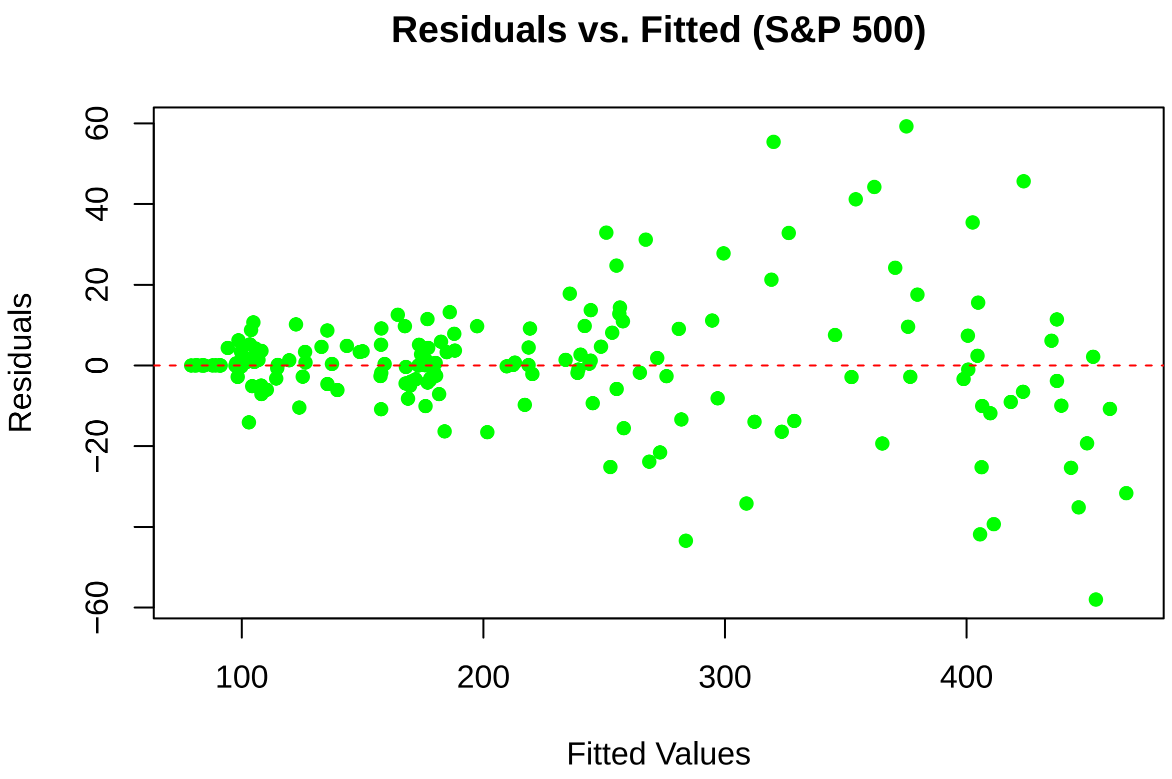
```r
plot(fitted(arima_Goog), residuals(arima_Goog),
     main = "Residuals vs. Fitted (GOOG)",
     xlab = "Fitted Values", ylab = "Residuals",
     col = "blue", pch = 16)
abline(h = 0, col = "red", lty = 2)
```



The residuals vs. fitted plot for the GOOG model shows that residuals are randomly scattered around zero, indicating that the model does not exhibit strong bias. However, there is heteroscedasticity, as residual variance increases with higher fitted values, suggesting that the model may not fully capture all patterns in the data.

Let's now plot the residuals vs. fitted Values for the S&P 500 SARIMA Model:

```r
plot(fitted(arima_SPY), residuals(arima_SPY),
     main = "Residuals vs. Fitted (S&P 500)",
     xlab = "Fitted Values", ylab = "Residuals",
     col = "green", pch = 16)
abline(h = 0, col = "red", lty = 2)  # Add horizontal line at zero
```

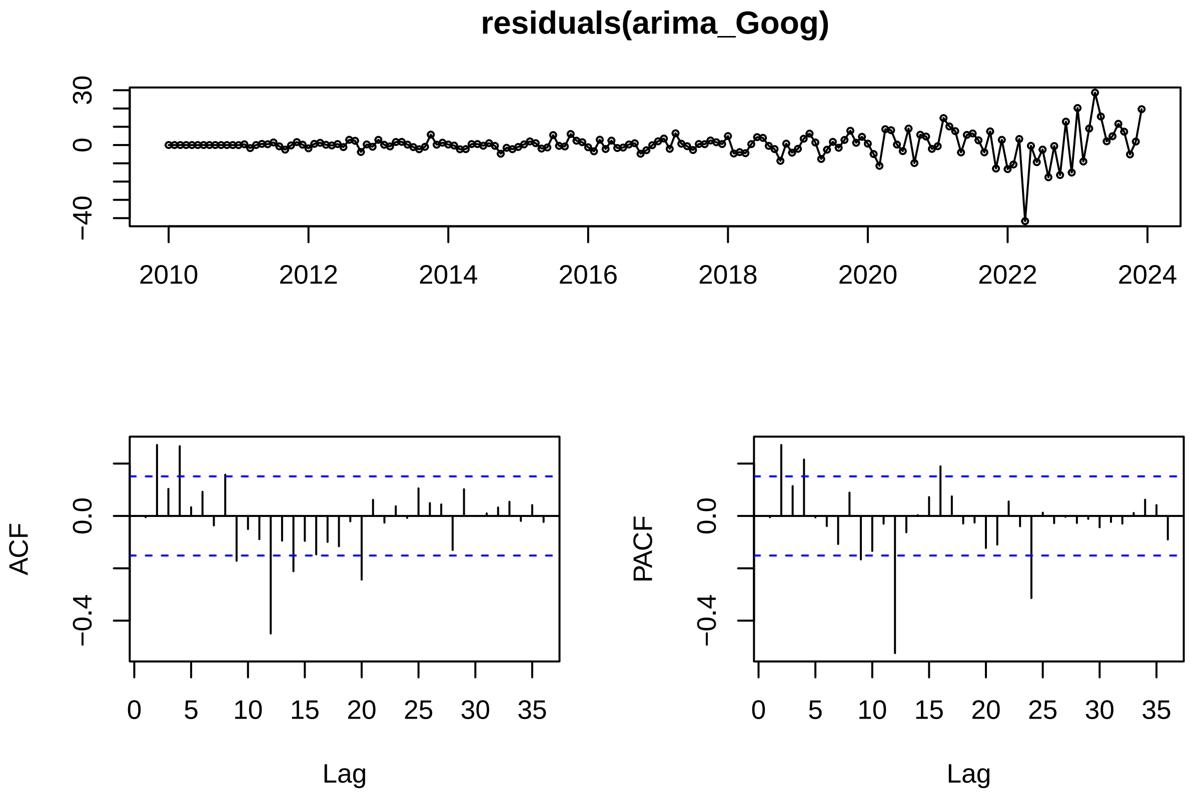


The residuals vs. fitted plot for the S&P 500 model shows that residuals are mostly centered around zero, suggesting that the model does not have significant bias. However, there is an increase in residual variance as fitted values grow, indicating potential heteroscedasticity, which may suggest that the model does not fully capture volatility patterns. Additionally, some large residuals suggest potential outliers or structural breaks in the data.

Part f

Let's plot the ACF and PACF of the residuals from the model fit to the Google time series data:

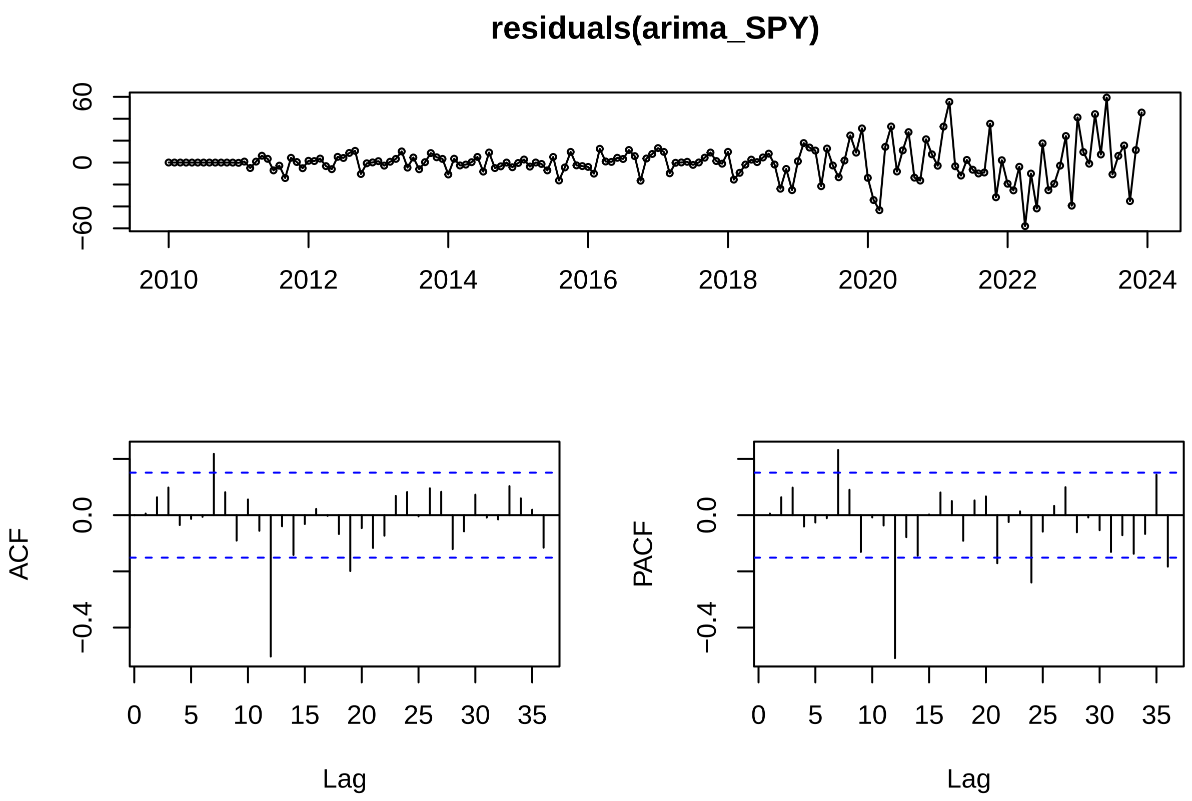
```r
tsdisplay(residuals(arima_Goog))
```



The residual plot for the Google SARIMA model shows that residuals are generally centered around zero, but there is noticeable heteroscedasticity, with increasing volatility after 2020. The ACF plot of the residuals suggests that most autocorrelations are within the confidence bounds, but a few significant lags indicate possible remaining patterns, meaning the model may not fully capture all dependencies. The PACF plot also shows some spikes at lower lags, which might suggest the need for an additional AR or MA term.

Let's now plot the ACF and PACF of the residuals from the model fit to the S&P 500 time series data:

```
tsdisplay(residuals(arima_SPY))
```

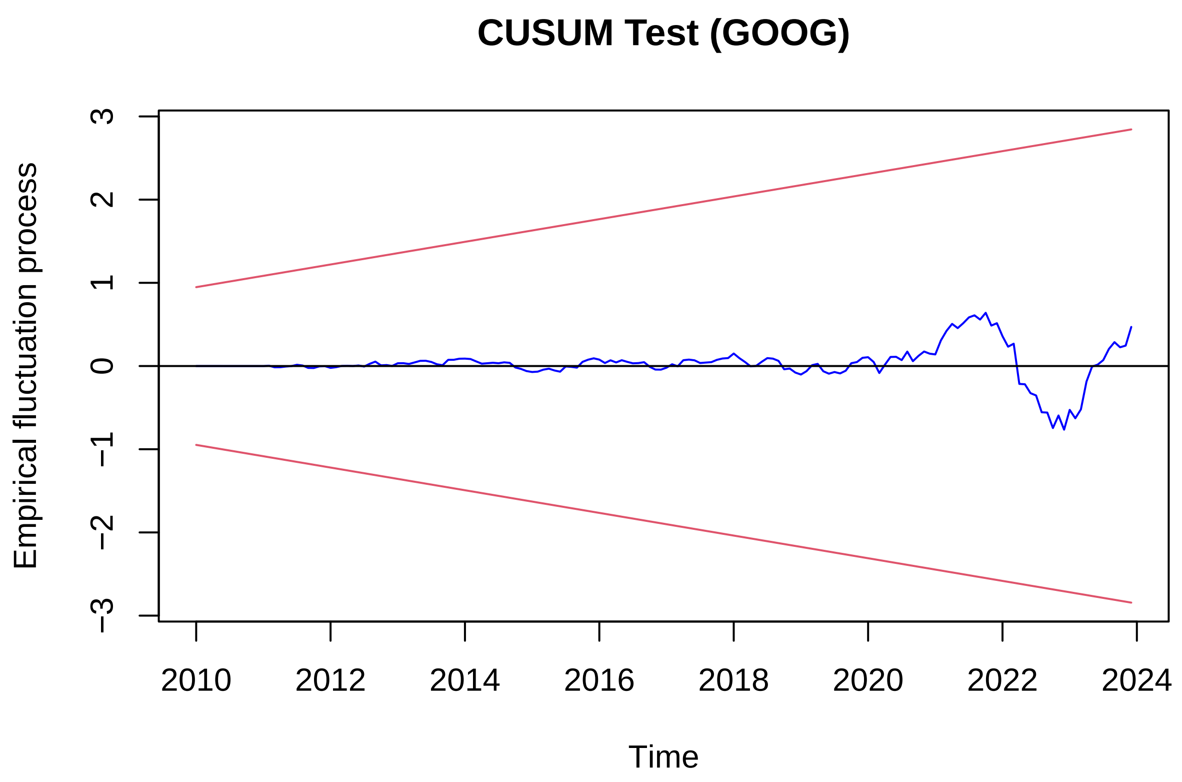


The residual plot for the S&P 500 ARIMA model shows that residuals are generally centered around zero, but with noticeable increasing volatility after 2020, indicating potential heteroscedasticity or external market shocks. The ACF of residuals shows most autocorrelations within the confidence bounds, suggesting that residuals are approximately white noise, though a few significant spikes indicate possible remaining structure in the data. The PACF plot also has some small spikes, which may imply that an additional AR or MA term could improve the model.

Part g

Let's plot the CUMSUM of the residuals from the model applied to the Google time series data:

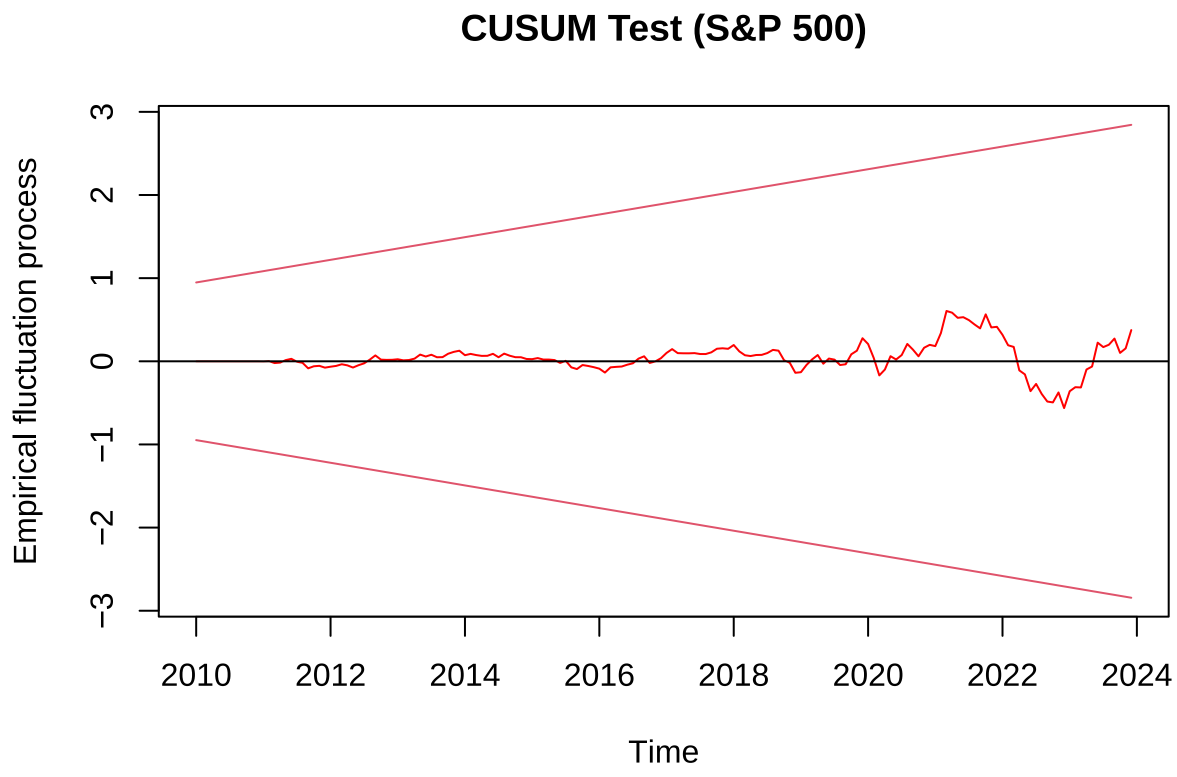
```r
cusum_Goog <- efp(residuals(arima_Goog) ~ 1, type = "Rec-CUSUM")

plot(cusum_Goog, main = "CUSUM Test (GOOG)", col = "blue")
```



Overall, the cumulative sum of residuals of Google's model remains within the confidence bounds for the entire period. However, around 2020, the increasing fluctuations suggested structural changes or instability in the model after that point.

Let's now plot the CUMSUM of the residuals from the model applied to the S&P 500 time series data:

```r
cusum_SPY <- efp(residuals(arima_SPY) ~ 1, type = "Rec-CUSUM")
plot(cusum_SPY, main = "CUSUM Test (S&P 500)", col = "red")
```



Overall, the cumulative sum of residuals in Google's model stays within the confidence bounds for the entire period. However, around 2020, the growing fluctuations suggest potential structural changes or instability in the model from that point onward.

Part h

Let's now look at the associated diagnostic statistics of the model applied to the Google time series data:

```
summary(arima_Goog)
```

```
## Series: Goog_ts
## ARIMA(1,1,0)(0,1,0)[12]
##
## Coefficients:
##          ar1
##       0.0139
## s.e.  0.0822
##
## sigma^2 = 48.34:  log likelihood = -520
## AIC=1044   AICc=1044.07   BIC=1050.08
##
## Training set error measures:
##                     ME     RMSE      MAE       MPE     MAPE      MASE
## Training set 0.2965324 6.656645 3.840541 0.2227106 6.367618 0.2838663
##                      ACF1
## Training set -0.004929264
```

The training set error measures indicate that the model performs reasonably well, with a MAPE of 6.36%, suggesting a moderate forecasting accuracy. The RMSE (6.66) and MAE (3.84) show that the average prediction error is relatively small. The low ACF1 value (-0.005) indicates minimal autocorrelation in residuals, implying that most of the patterns in the data have been captured.

Let's now look at the associated diagnostic statistics of the model applied to the S&P 500 time series data:

```r
summary(arima_SPY)
```

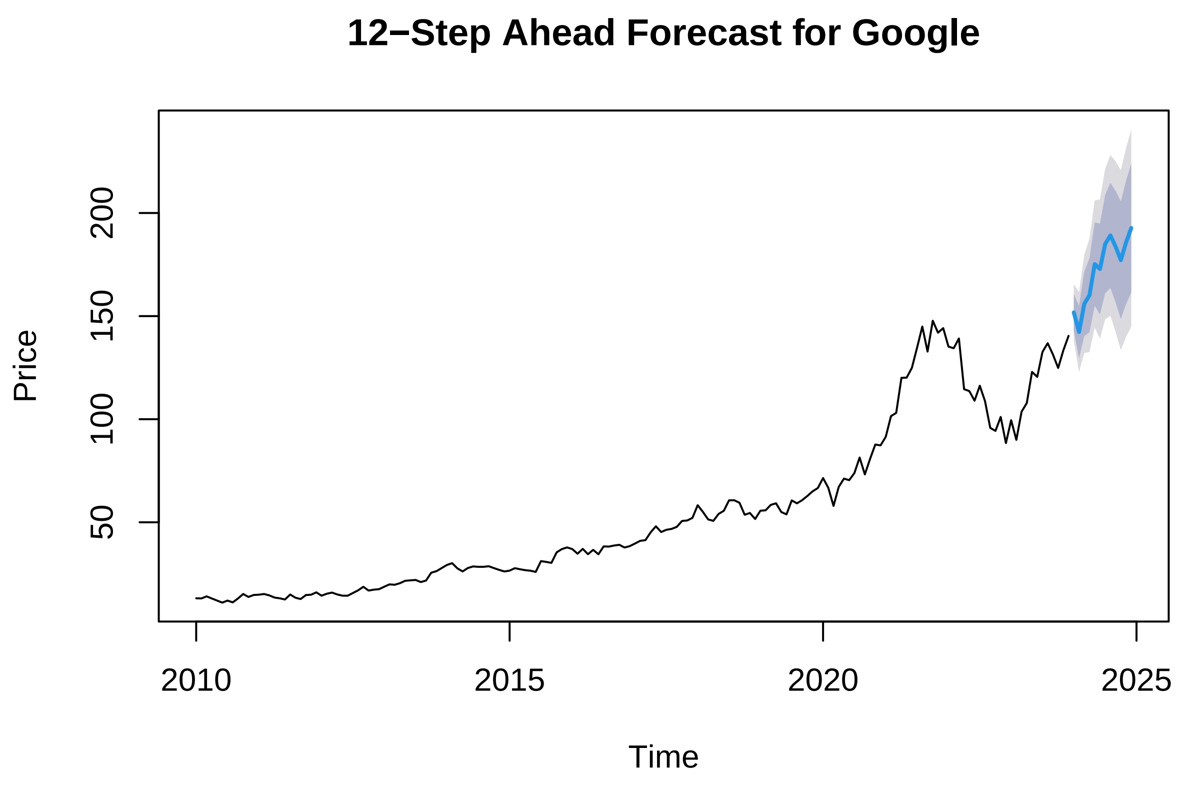
```
## Series: SPY_ts
## ARIMA(1,1,0)(0,1,0)[12]
##
## Coefficients:
##           ar1
##       -0.1442
## s.e.   0.0810
##
## sigma^2 = 274.6:  log likelihood = -654.64
## AIC=1313.27   AICc=1313.35   BIC=1319.36
##
## Training set error measures:
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 0.5020565 15.86607 10.08819 0.05873652 3.921169 0.3113606
##                     ACF1
## Training set 0.005647991
```

The training set error measures indicate a relatively low MAPE (3.92%), suggesting good predictive accuracy. However, the RMSE (15.87) and MAE (10.09) are relatively high, implying larger errors in absolute terms. The ME (0.50) is close to zero, indicating minimal bias, while the ACF1 (0.00) suggests almost no autocorrelation in residuals, which is a positive sign. Despite decent performance, the fairly high RMSE and MAE suggest that extreme values or volatility might not be well captured.

Part i

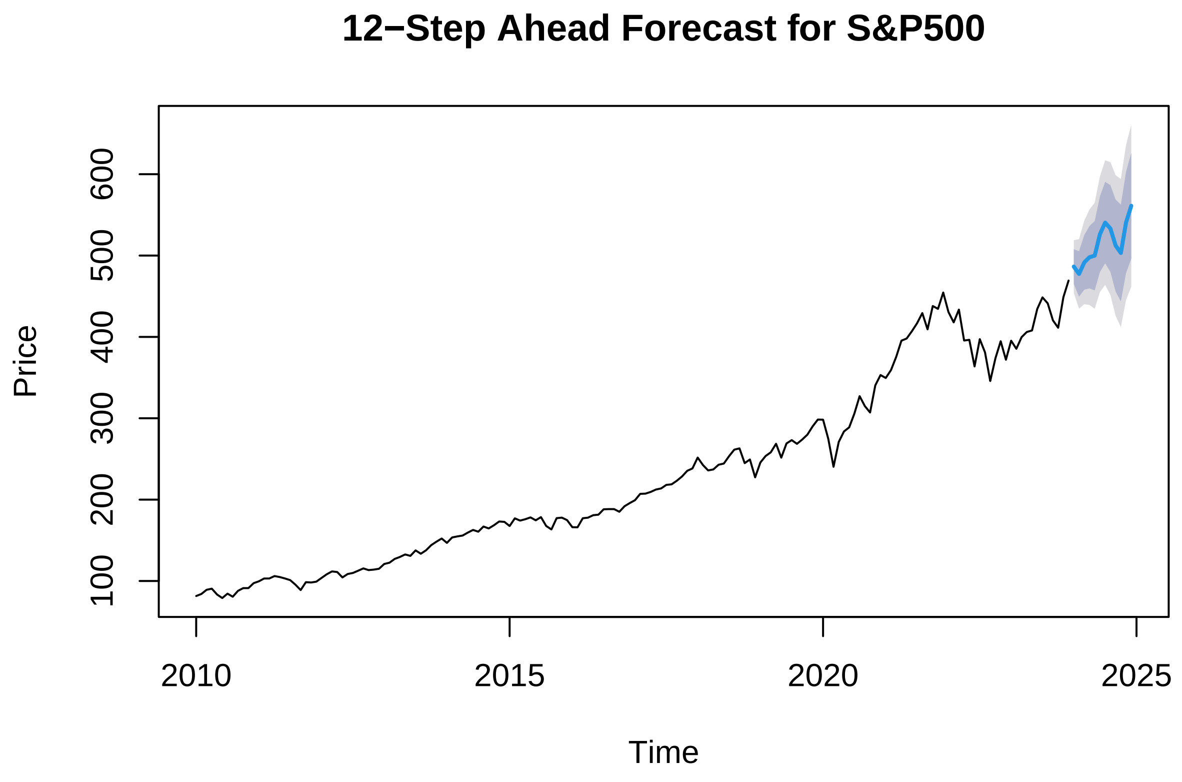
Let's use our model applied to the Google time series to forecast 12 steps ahead:

```r
forecast_Goog <- forecast(arima_Goog, h = 12)
plot(forecast_Goog, main = "12-Step Ahead Forecast for Google",
     ylab = "Price", xlab = "Time")
```



Let's now use our model applied to the S&P 500 time series to forecast 12 steps ahead:

```
forecast_SPY <- forecast(arima_SPY, h = 12)
plot(forecast_SPY, main = "12-Step Ahead Forecast for S&P500",
     ylab = "Price", xlab = "Time")
```



Part j

Next, let's compare our forecasts with the forecasts we would get using a model obtained from applying auto.arima.

```r
auto_arima_Goog <- auto.arima(Goog_ts)
auto_arima_SPY <- auto.arima(SPY_ts)

forecast_auto_Goog <- forecast(auto_arima_Goog, h = 12)
forecast_auto_SPY <- forecast(auto_arima_SPY, h = 12)
```

```r
# Function to calculate MAPE
mape <- function(actual, forecast) {
  mean(abs((actual - forecast) / actual)) * 100
}

# Compute MAPE for SARIMA and Auto.ARIMA models
mape_Goog_sarima <- mape(as.numeric(actual_Goog), as.numeric(forecast_Goog$mean))
mape_Goog_auto <- mape(as.numeric(actual_Goog), as.numeric(forecast_auto_Goog$mean))

mape_SPY_sarima <- mape(as.numeric(actual_SPY), as.numeric(forecast_SPY$mean))
mape_SPY_auto <- mape(as.numeric(actual_SPY), as.numeric(forecast_auto_SPY$mean))

# Print results
print(paste("MAPE for SARIMA Google:", round(mape_Goog_sarima, 2), "%"))
```

```
## [1] "MAPE for SARIMA Google: 5.42 %"
```

```r
print(paste("MAPE for Auto.ARIMA Google:", round(mape_Goog_auto, 2), "%"))
```

```
## [1] "MAPE for Auto.ARIMA Google: 13.54 %"
```

```r
print(paste("MAPE for SARIMA SPY:", round(mape_SPY_sarima, 2), "%"))
```

```
## [1] "MAPE for SARIMA SPY: 5.11 %"
```

```r
print(paste("MAPE for Auto.ARIMA SPY:", round(mape_SPY_auto, 2), "%"))
```

```
## [1] "MAPE for Auto.ARIMA SPY: 11.08 %"
```

We see that the MAPE obtained from the SARIMA models are lower than when using auto.arima.

Part k

Let's now combine the SARIMA models and the auto.arima models.

```r
# Function to calculate MAPE
mape <- function(actual, forecast) {
  mean(abs((actual - forecast) / actual)) * 100
}

# Compute MAPE for Combined Forecast
combined_forecast_Goog <- (forecast_auto_Goog$mean + forecast_Goog$mean) / 2
combined_forecast_SPY <- (forecast_auto_SPY$mean + forecast_SPY$mean) / 2

mape_Goog_combined <- mape(actual_Goog, combined_forecast_Goog)
mape_SPY_combined <- mape(actual_SPY, combined_forecast_SPY)
```

```r
# Print results
print(paste("MAPE for Combined Google Forecast:", round(mape_Goog_combined, 2), "%"))
```

```
## [1] "MAPE for Combined Google Forecast: 5.51 %"
```

```r
print(paste("MAPE for Combined SPY Forecast:", round(mape_SPY_combined, 2), "%"))
```

```
## [1] "MAPE for Combined SPY Forecast: 7.91 %"
```

After combining the forecasts from both approaches, the combined MAPE values for Google (5.51%) and SPY (7.91%) are higher than the MAPEs for the SARIMA models but lower than the MAPES for the auto.arima models. This suggests that the averaging approach of combining forecasts may not have been the best approach to use, and a better approach might be to assign weights based on individual model performance rather than using a simple average.

Part L

Now, let's fit an appropriate VAR model.

```r
# Create a data frame for VAR model
data_var <- data.frame(GOOG = as.numeric(Goog_ts), SPY = as.numeric(SPY_ts))

# Convert to time series object
ts_data_training <- ts(data_var, start = c(2010, 1), frequency = 12)
```

The data must be stationary in order to apply a VAR model. Let's check for stationarity using the ADF test:

```r
# ADF test for stationarity
adf.test(ts_data_training[, "GOOG"])
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_data_training[, "GOOG"]
## Dickey-Fuller = -2.9558, Lag order = 5, p-value = 0.1774
## alternative hypothesis: stationary
```

```r
adf.test(ts_data_training[, "SPY"])
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_data_training[, "SPY"]
## Dickey-Fuller = -2.2179, Lag order = 5, p-value = 0.4853
## alternative hypothesis: stationary
```

The results from the ADF test tell us that there is not enough evidence to say that either of the time series is stationary. So, let's difference the data:

```r
# First difference to make the data stationary
ts_data_training_diff <- diff(ts_data_training)
```

Let's now check for stationarity using the differenced data:

```r
# ADF test after differencing
adf.test(ts_data_training_diff[, "GOOG"])
```

```
##
##  Augmented Dickey-Fuller Test
```

```
##
## data:  ts_data_training_diff[, "GOOG"]
## Dickey-Fuller = -3.8409, Lag order = 5, p-value = 0.0187
## alternative hypothesis: stationary
```

```r
adf.test(ts_data_training_diff[, "SPY"])
```

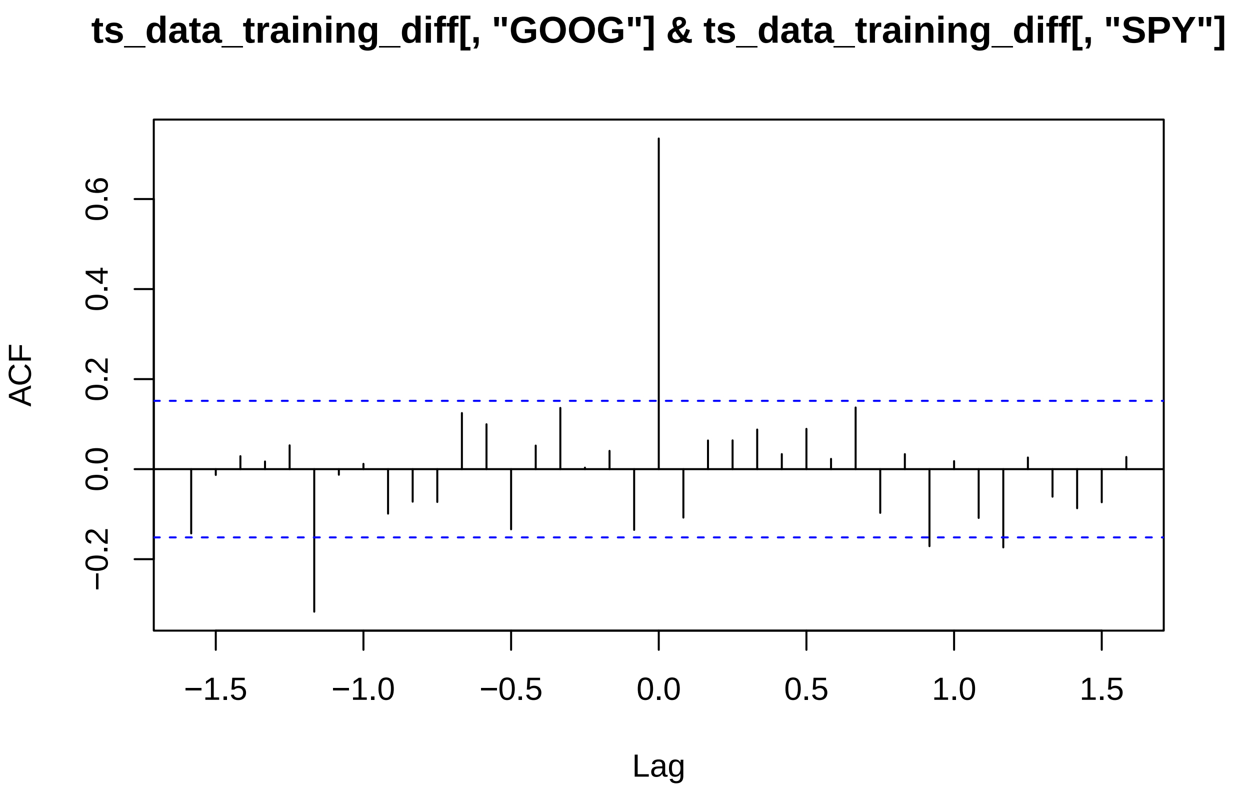
```
## Warning in adf.test(ts_data_training_diff[, "SPY"]): p-value smaller than
## printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_data_training_diff[, "SPY"]
## Dickey-Fuller = -5.3636, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

We have enough evidence to show that both of the differenced time series are stationary.

Before we create the VAR model, let's see what the cross correlation function of the two differenced time series looks like:

```r
ccf(ts_data_training_diff[, "GOOG"], ts_data_training_diff[, "SPY"])
```



Based on the ccf plot, the strongest correlation appears at lag = 0, suggesting that GOOG and SPY move together in the same time period. There are, however, other statistically significant lags which we will try to capture using a VAR model.

Let's now create the VAR model on the differenced data:

```r
VARselect(ts_data_training_diff, lag.max = 12)
```

```
## $selection
## AIC(n)  HQ(n)  SC(n) FPE(n)
##      8      2      1      8
##
## $criteria
##                 1           2           3           4           5           6
## AIC(n)    7.532127    7.489299    7.509800    7.461896    7.486407    7.411057
## HQ(n)     7.579979    7.569052    7.621454    7.605451    7.661864    7.618414
## SC(n)     7.649937    7.685649    7.784690    7.815326    7.918377    7.921567
## FPE(n) 1867.090511 1788.878147 1826.072041 1740.900272 1784.484822 1655.480484
##                 7           8           9          10          11          12
## AIC(n)    7.406314    7.384513    7.419944    7.401902    7.410279    7.408561
## HQ(n)     7.645573    7.655673    7.723006    7.736865    7.777142    7.807326
## SC(n)     7.995364    8.052103    8.166074    8.226572    8.313489    8.390311
## FPE(n) 1648.348307 1613.699235 1673.081975 1644.604463 1660.200204 1659.455177
```

From majority vote, we see that lag 8 would be the most appropriate to use (both AIC and FPE suggested using lag 8).

```r
# Fit the VAR model using the selected lag
var_model_diff <- VAR(ts_data_training_diff, p = 8)
```

```r
# Display summary
summary(var_model_diff)
```

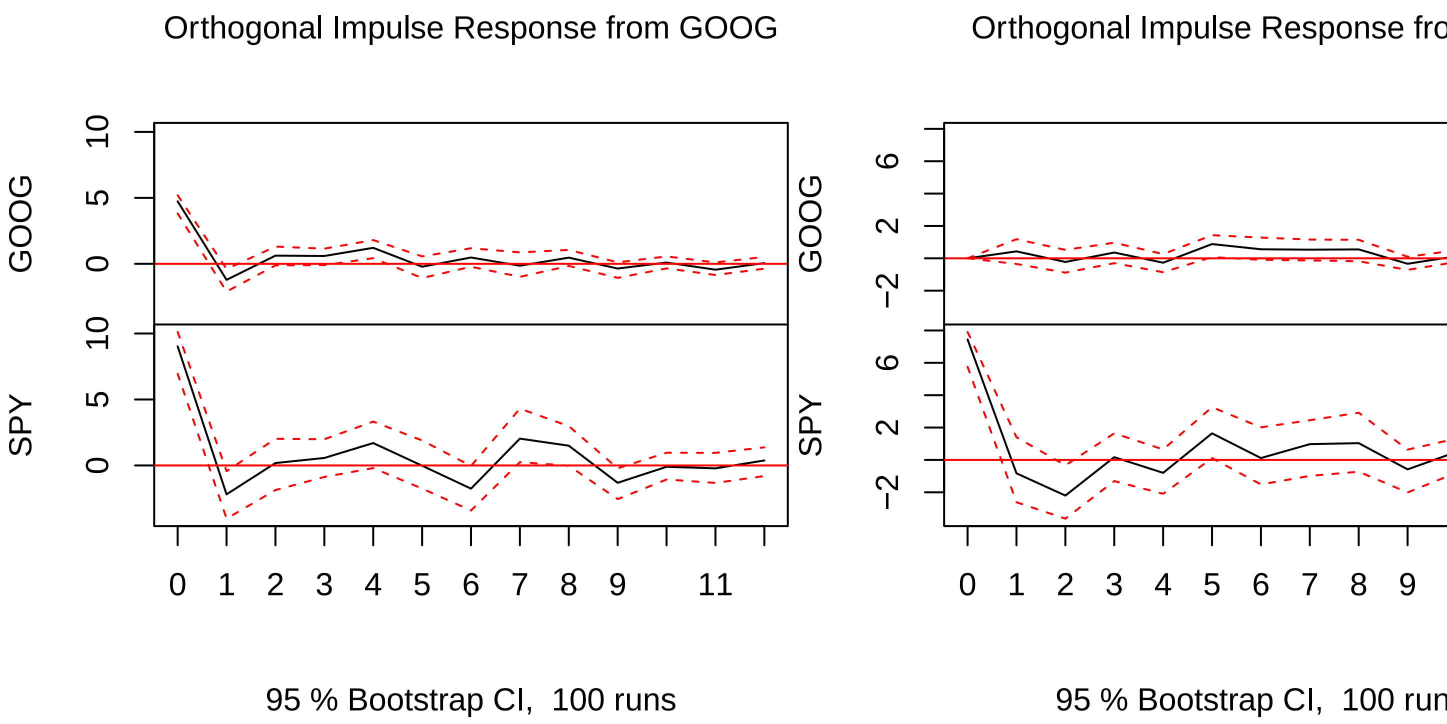
```
##
## VAR Estimation Results:
## =========================
## Endogenous variables: GOOG, SPY
## Deterministic variables: const
## Sample size: 159
## Log Likelihood: -999.767
## Roots of the characteristic polynomial:
## 0.8704 0.8704 0.8549 0.8549 0.8511 0.8511 0.8504 0.8504 0.8493 0.8493 0.8138 0.8138 0.7999 0.7999 0.
## Call:
## VAR(y = ts_data_training_diff, p = 8)
##
##
## Estimation results for equation GOOG:
## =====================================
## GOOG = GOOG.l1 + SPY.l1 + GOOG.l2 + SPY.l2 + GOOG.l3 + SPY.l3 + GOOG.l4 + SPY.l4 + GOOG.l5 + SPY.l5 +
##
##          Estimate Std. Error t value Pr(>|t|)
## GOOG.l1 -0.366000   0.128320  -2.852  0.00499 **
## SPY.l1   0.057603   0.052703   1.093  0.27626
## GOOG.l2  0.068963   0.129940   0.531  0.59643
## SPY.l2  -0.002751   0.055006  -0.050  0.96018
## GOOG.l3  0.093161   0.127902   0.728  0.46758
## SPY.l3   0.049603   0.054484   0.910  0.36414
## GOOG.l4  0.369138   0.128181   2.880  0.00460 **
## SPY.l4  -0.018942   0.054590  -0.347  0.72911
## GOOG.l5 -0.103760   0.128127  -0.810  0.41940
## SPY.l5   0.101893   0.054633   1.865  0.06424 .
## GOOG.l6 -0.215271   0.128562  -1.674  0.09624 .
## SPY.l6   0.125159   0.055154   2.269  0.02476 *
## GOOG.l7 -0.299827   0.129547  -2.314  0.02208 *
## SPY.l7   0.135663   0.056082   2.419  0.01683 *
## GOOG.l8 -0.289323   0.140136  -2.065  0.04078 *
## SPY.l8   0.142631   0.056721   2.515  0.01303 *
## const    0.014813   0.473863   0.031  0.97511
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 4.739 on 142 degrees of freedom
## Multiple R-Squared: 0.2252,  Adjusted R-squared: 0.1379
## F-statistic: 2.579 on 16 and 142 DF,  p-value: 0.001535
##
##
## Estimation results for equation SPY:
## ====================================
## SPY = GOOG.l1 + SPY.l1 + GOOG.l2 + SPY.l2 + GOOG.l3 + SPY.l3 + GOOG.l4 + SPY.l4 + GOOG.l5 + SPY.l5 +
##
##         Estimate Std. Error t value Pr(>|t|)
## GOOG.l1 -0.25019    0.31681  -0.790   0.4310
## SPY.l1  -0.11129    0.13012  -0.855   0.3938
## GOOG.l2  0.48288    0.32081   1.505   0.1345
## SPY.l2  -0.29389    0.13581  -2.164   0.0321 *
## GOOG.l3  0.29337    0.31578   0.929   0.3545
## SPY.l3  -0.07821    0.13452  -0.581   0.5619
## GOOG.l4  0.75317    0.31647   2.380   0.0186 *
## SPY.l4  -0.19076    0.13478  -1.415   0.1592
## GOOG.l5 -0.05487    0.31633  -0.173   0.8625
## SPY.l5   0.10416    0.13488   0.772   0.4413
## GOOG.l6 -0.53356    0.31741  -1.681   0.0950 .
## SPY.l6   0.02470    0.13617   0.181   0.8563
## GOOG.l7 -0.17370    0.31984  -0.543   0.5879
## SPY.l7   0.19240    0.13846   1.390   0.1668
## GOOG.l8 -0.09665    0.34598  -0.279   0.7804
## SPY.l8   0.15275    0.14004   1.091   0.2772
## const    2.47865    1.16992   2.119   0.0359 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 11.7 on 142 degrees of freedom
## Multiple R-Squared: 0.1752,  Adjusted R-squared: 0.08232
## F-statistic: 1.886 on 16 and 142 DF,  p-value: 0.02624
##
##
##
## Covariance matrix of residuals:
##       GOOG    SPY
## GOOG 22.46  42.78
## SPY  42.78 136.88
##
## Correlation matrix of residuals:
##        GOOG    SPY
## GOOG 1.0000 0.7716
## SPY  0.7716 1.0000
```

Part m

```r
irf(var_model_diff)
```

```
##
## Impulse response coefficients
## $GOOG
##              GOOG         SPY
##  [1,]  4.73879518  9.02687969
##  [2,] -1.21442250 -2.19022812
##  [3,]  0.62028474  0.18297881
##  [4,]  0.59502552  0.56591259
##  [5,]  1.21359320  1.69613274
##  [6,] -0.21885802 -0.02038309
##  [7,]  0.47946930 -1.75984205
##  [8,] -0.14038861  2.03235610
##  [9,]  0.46956692  1.49583513
## [10,] -0.35655047 -1.31484757
## [11,]  0.09775554 -0.10285442
##
## $SPY
##             GOOG        SPY
##  [1,]  0.0000000  7.4428863
##  [2,]  0.4287323 -0.8283371
##  [3,] -0.2251052 -2.2024649
##  [4,]  0.3565578  0.1697978
##  [5,] -0.2723168 -0.7987403
##  [6,]  0.8798870  1.6416969
##  [7,]  0.5588646  0.1163162
##  [8,]  0.5344349  0.9741290
##  [9,]  0.5505176  1.0374604
## [10,] -0.3381408 -0.5839419
## [11,]  0.1291873  0.4942092
##
##
## Lower Band, CI= 0.95
## $GOOG
##             GOOG        SPY
##  [1,]  3.8030855  6.5633537
##  [2,] -2.0350576 -4.6463351
##  [3,] -0.1176900 -1.4420024
##  [4,] -0.1074079 -0.8898197
##  [5,]  0.4811628 -0.1970065
##  [6,] -1.0343743 -1.6349919
##  [7,] -0.4017742 -3.9703225
##  [8,] -1.0636303 -0.2069442
##  [9,] -0.3152612 -0.5872260
## [10,] -0.9956741 -2.4869784
## [11,] -0.5232557 -1.3819263
##
## $SPY
##             GOOG        SPY
##  [1,]  0.0000000  5.8113880
##  [2,] -0.3020922 -2.2916561
##  [3,] -1.0676205 -3.8476845
##  [4,] -0.3333934 -1.5769631
##  [5,] -0.9407523 -2.1610675
##  [6,]  0.1549354 -0.2799082
##  [7,] -0.3782902 -1.7995053
##  [8,] -0.2152347 -0.5434819
##  [9,] -0.1156665 -0.6229331
## [10,] -0.9869289 -1.9623515
## [11,] -0.2091899 -0.6495330
##
##
## Upper Band, CI= 0.95
## $GOOG
##             GOOG        SPY
##  [1,]  5.2716830 10.5582486
##  [2,] -0.5409928 -0.4855406
##  [3,]  1.2469461  2.2791008
##  [4,]  1.2094461  2.7208721
##  [5,]  1.8520440  3.6006552
##  [6,]  0.5831706  1.3761075
##  [7,]  1.0937969 -0.2149552
##  [8,]  0.5363803  3.6298122
##  [9,]  1.0332374  2.6622664
## [10,]  0.3454369  0.1138535
## [11,]  0.6640950  1.0507706
##
## $SPY
##            GOOG        SPY
##  [1,] 0.0000000  7.7964101
##  [2,] 1.0459184  0.4952431
##  [3,] 0.4353956 -0.3715452
##  [4,] 1.0404960  2.0266495
##  [5,] 0.4278112  0.8373423
##  [6,] 1.3749852  2.8439944
##  [7,] 1.3338058  1.7553830
##  [8,] 1.1793672  2.6684750
##  [9,] 1.1541379  2.6679882
## [10,] 0.1466093  0.5577396
## [11,] 0.4848826  1.4825369
```

```r
plot(irf(var_model_diff, n.ahead=12))
```



Analysis of the Orthogonal Impulse Response from GOOG: A shock to GOOG has an immediate large effect on itself, which then declines over time, stabilizing near zero. A shock to GOOG initially has a positive impact on SPY, but the effect fluctuates and stabilizes close to zero over time.

Analysis of the Orthogonal Impulse Response from SPY: A shock to SPY has a small initial effect on GOOG, which slightly oscillates but remains close to zero. A shock to SPY has an immediate large effect on itself, which then decays over time.

Part n

Let's now perform a Granger Causality test, using order 8:

```
grangertest(Goog_ts ~ SPY_ts, order = 8)
```

```
## Granger causality test
##
## Model 1: Goog_ts ~ Lags(Goog_ts, 1:8) + Lags(SPY_ts, 1:8)
## Model 2: Goog_ts ~ Lags(Goog_ts, 1:8)
##   Res.Df Df      F  Pr(>F)
## 1    143
## 2    151 -8 2.5666 0.01204 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
grangertest(SPY_ts ~ Goog_ts, order = 8)
```

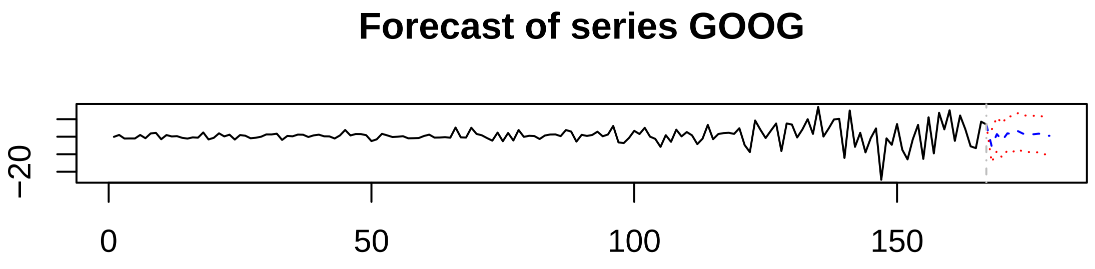
```
## Granger causality test
##
## Model 1: SPY_ts ~ Lags(SPY_ts, 1:8) + Lags(Goog_ts, 1:8)
## Model 2: SPY_ts ~ Lags(SPY_ts, 1:8)
##   Res.Df Df      F  Pr(>F)
## 1    143
## 2    151 -8 2.1891 0.03158 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

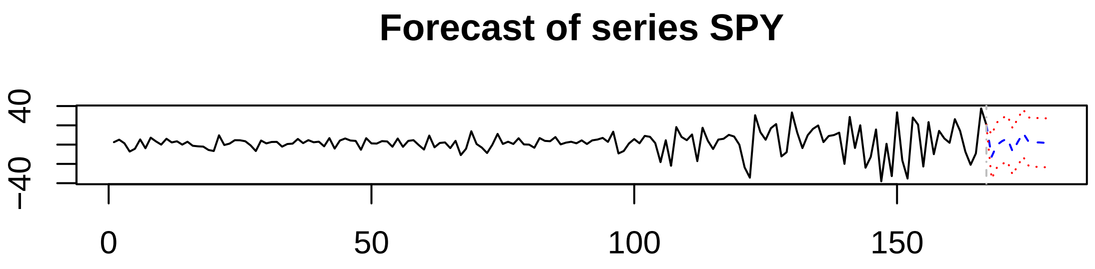
Since both tests yield statistically significant results (p-values < 0.05), this indicates bidirectional Granger causality between SPY_ts and Goog_ts. This means that historical values of SPY_ts help predict Goog_ts, and vice versa. In practical terms, there is evidence of a dynamic relationship between these two time series, suggesting interdependence in their movements over time.

Part o

Let's use our VAR model to forecast 12 steps ahead. Keep in mind that the below forecast is for the differences in prices, not the actual stock prices.

```r
var.predict = predict(object = var_model_diff, n.ahead=12)
plot(var.predict)
```





Now, let's prepare our data to calculate the MAPE from our VAR model.

```r
#Extracting the forecasted values of the differences

GOOG_forecasted_values_diff <- var.predict$fcst$GOOG[, 1]
SPY_forecasted_values_diff <- var.predict$fcst$SPY[, 1]
```

Now, let's use the forecasted differences to forecast the actual stock prices for GOOG and the S&P 500:

```r
last_observed_value_goog <- tail(Goog_ts, 1)
forecasted_goog_prices <- as.numeric(last_observed_value_goog) + cumsum(GOOG_forecasted_values_diff)
```

```r
last_observed_value_spy <- tail(SPY_ts, 1)
forecasted_spy_prices <- as.numeric(last_observed_value_spy) + cumsum(SPY_forecasted_values_diff)
```

Let's now calculate the MAPE:

```r
# Compute MAPE for SARIMA and Auto.ARIMA models
mape_Goog_VAR <- mape(as.numeric(actual_Goog), forecasted_goog_prices)
```

```r
mape_spy_VAR <- mape(as.numeric(actual_SPY), forecasted_spy_prices)

# Print results
print(paste("MAPE for Google Forecatss using VAR:", round(mape_Goog_VAR, 2), "%"))
```

```
## [1] "MAPE for Google Forecatss using VAR: 14.28 %"
```

```r
print(paste("MAPE for S&P 500 Forecasts using VAR", round(mape_spy_VAR, 2), "%"))
```

```
## [1] "MAPE for S&P 500 Forecasts using VAR 12.79 %"
```

The MAPEs using VAR appear to be higher than the MAPEs obtained when using our custom SARIMA model and using auto.arima.

Conclusion

The first part of our analysis demonstrates that ARIMA models effectively capture the historical price movements of both Google and S&P 500, indicating that past values hold significant predictive power. The strong performance of these models suggests that time series forecasting can be a valuable tool for understanding stock price trends, particularly when historical patterns exhibit consistent behavior over time.

Additionally, our VAR model reveals a bidirectional Granger-causal relationship between S&P 500 and Google, with statistically significant p-values. This means that historical values of S&P 500 help predict Google and vice versa, highlighting a dynamic interdependence between the two assets. This relationship makes sense, as Google is a component of the S&P 500, meaning its performance directly influences the index, while at the same time, SPY reflects broader market trends that impact Google. However, while VAR successfully captures this interdependence, its MAPE was higher than those of SARIMA and auto.arima, indicating that ARIMA-based models provide better forecasting accuracy. This suggests that VAR is useful for understanding relationships between assets but may not be the best model for precise price predictions.

Given the unpredictable nature of financial markets, external shocks, and structural changes must be considered when applying these models in real-world decision-making. Future research could expand on this by exploring the interdependencies among multiple large-cap technology stocks, such as Apple, Microsoft, Amazon, and Meta, to better understand how movements in the broader tech sector influence individual stock performance.

Works Cited

Yahoo Finance. (2024). Google (GOOG) and S&P 500 ETF (SPY) Historical Market Data. Retrieved from https://finance.yahoo.com