ECON 144 Proj 3

Leonard Zhu

2024-12-05

I. Introduction

The S&P 500 Index is a widely recognized benchmark for the U.S. stock market, representing the performance of 500 large-cap companies. This project analyzes daily S&P 500 closing values for forecasting, focusing on trends, seasonality, and predictive accuracy. Data was retrieved from Yahoo Finance using the quantmod package in R.

```r
library(quantmod)
```

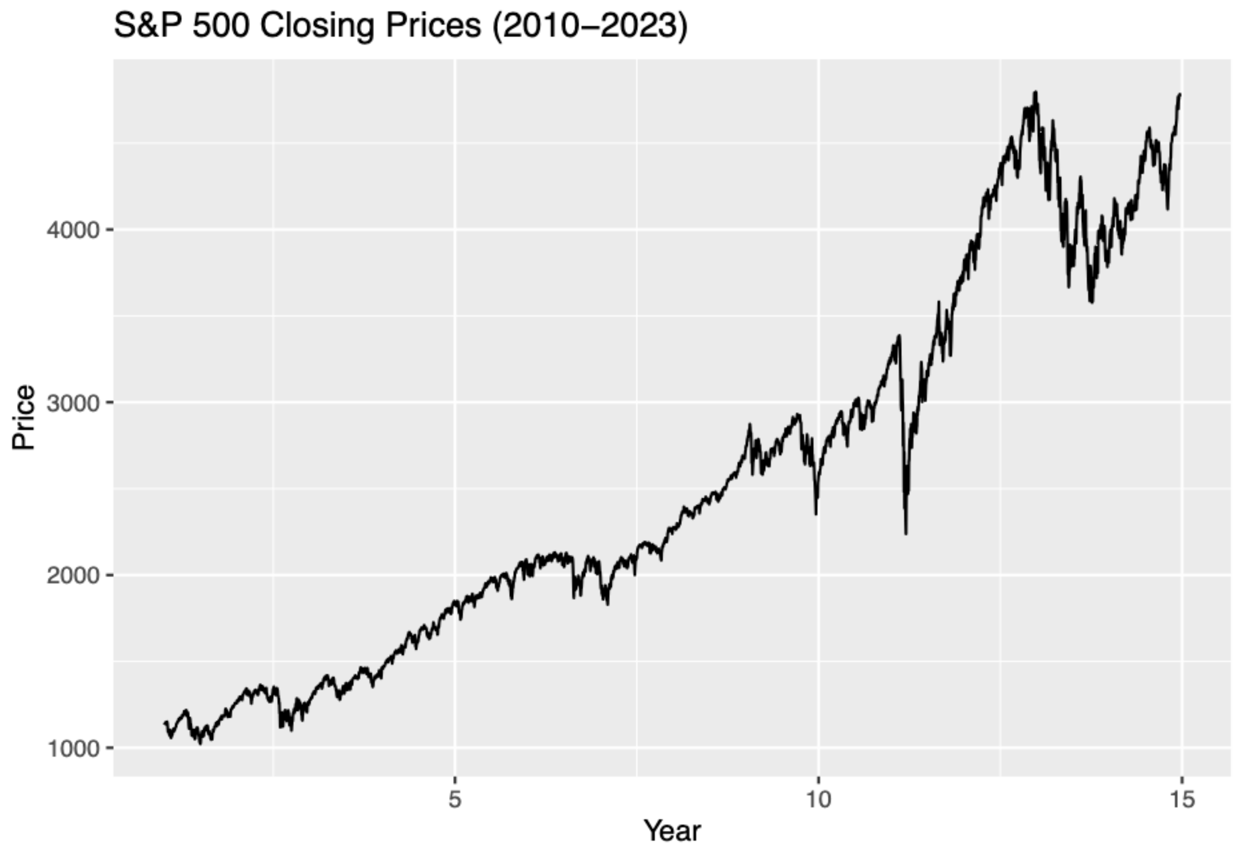
```
## Loading required package: xts

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following object is masked from 'package:tsibble':
##
##     index

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

##
## ######################### 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: 'xts'

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

## Loading required package: TTR
```

```r
# Fetch S&P 500 data
getSymbols("^GSPC", src = "yahoo", from = "2010-01-01", to = "2023-12-31")
```

```
## [1] "GSPC"
```

```r
sp500 <- Cl(GSPC)  # Closing prices
sp500_ts <- ts(sp500, frequency = 252)  # Convert to time series (252 trading days/year)

# Plot the data
autoplot(sp500_ts) +
  labs(title = "S&P 500 Closing Prices (2010-2023)", x = "Year", y = "Price")
```



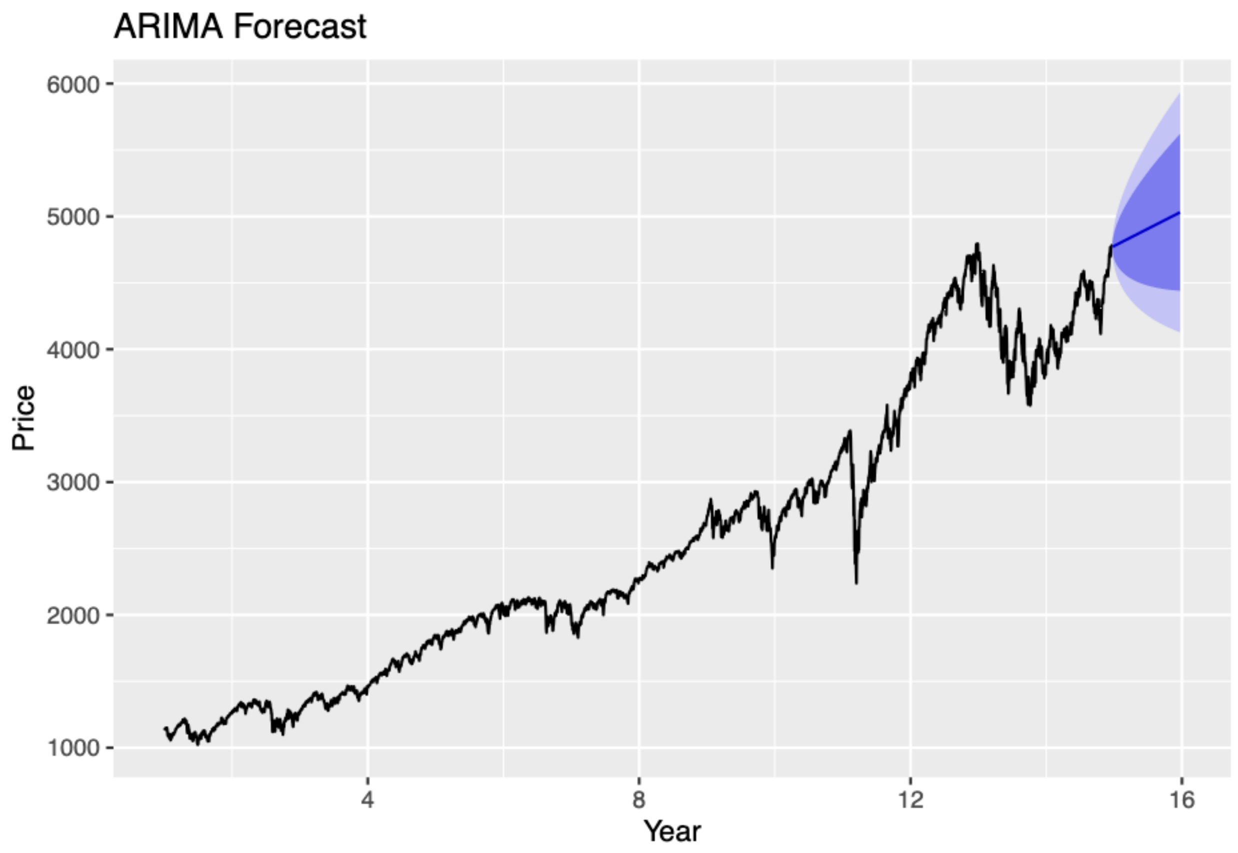
II. Results

1. ARIMA Model

```
# Fit ARIMA model
arima_fit <- auto.arima(sp500_ts)
summary(arima_fit)
```

```
## Series: sp500_ts
## ARIMA(2,1,0) with drift
##
## Coefficients:
##           ar1     ar2   drift
##       -0.0891  0.0454  1.0328
## s.e.   0.0168  0.0168  0.4890
##
## sigma^2 = 918:  log likelihood = -17005.02
## AIC=34018.04   AICc=34018.05   BIC=34042.71
##
## Training set error measures:
##                       ME     RMSE      MAE         MPE      MAPE       MASE
## Training set 0.000562964 30.28091 18.71254 -0.01475258 0.7284944 0.05649354
##                      ACF1
## Training set 0.0004977498
```

```
# Forecast with ARIMA
arima_forecast <- forecast(arima_fit, h = 252)  # Forecast 1 year ahead
autoplot(arima_forecast) +
  labs(title = "ARIMA Forecast", x = "Year", y = "Price")
```



2. ETS Model

```r
# Fit ETS model
ets_fit <- ets(sp500_ts)
```

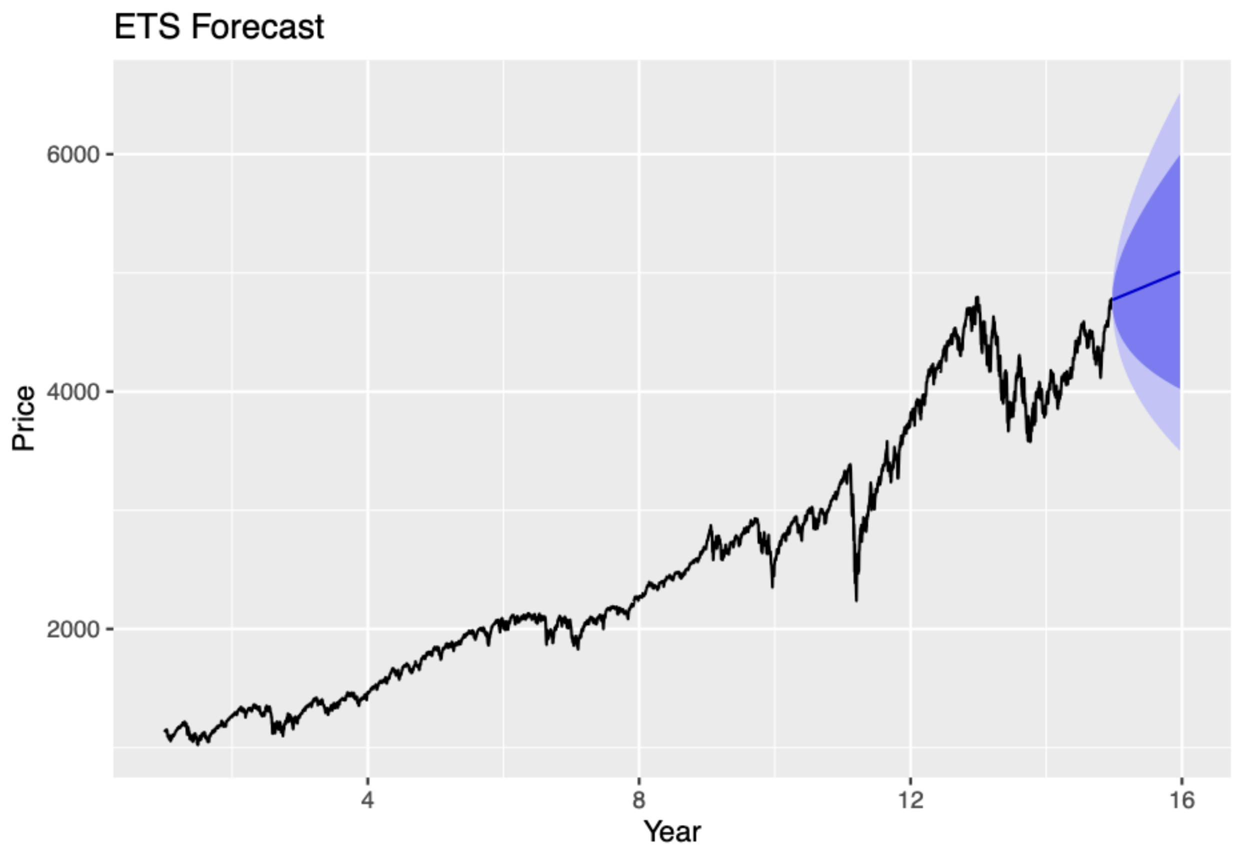
```
## Warning in ets(sp500_ts): I can't handle data with frequency greater than 24.
## Seasonality will be ignored. Try stlf() if you need seasonal forecasts.
```

```r
summary(ets_fit)
```

```
## ETS(M,A,N)
##
## Call:
##  ets(y = sp500_ts)
##
##   Smoothing parameters:
##     alpha = 0.8871
##     beta  = 1e-04
##
##   Initial states:
##     l = 1148.8521
##     b = 0.8781
##
```

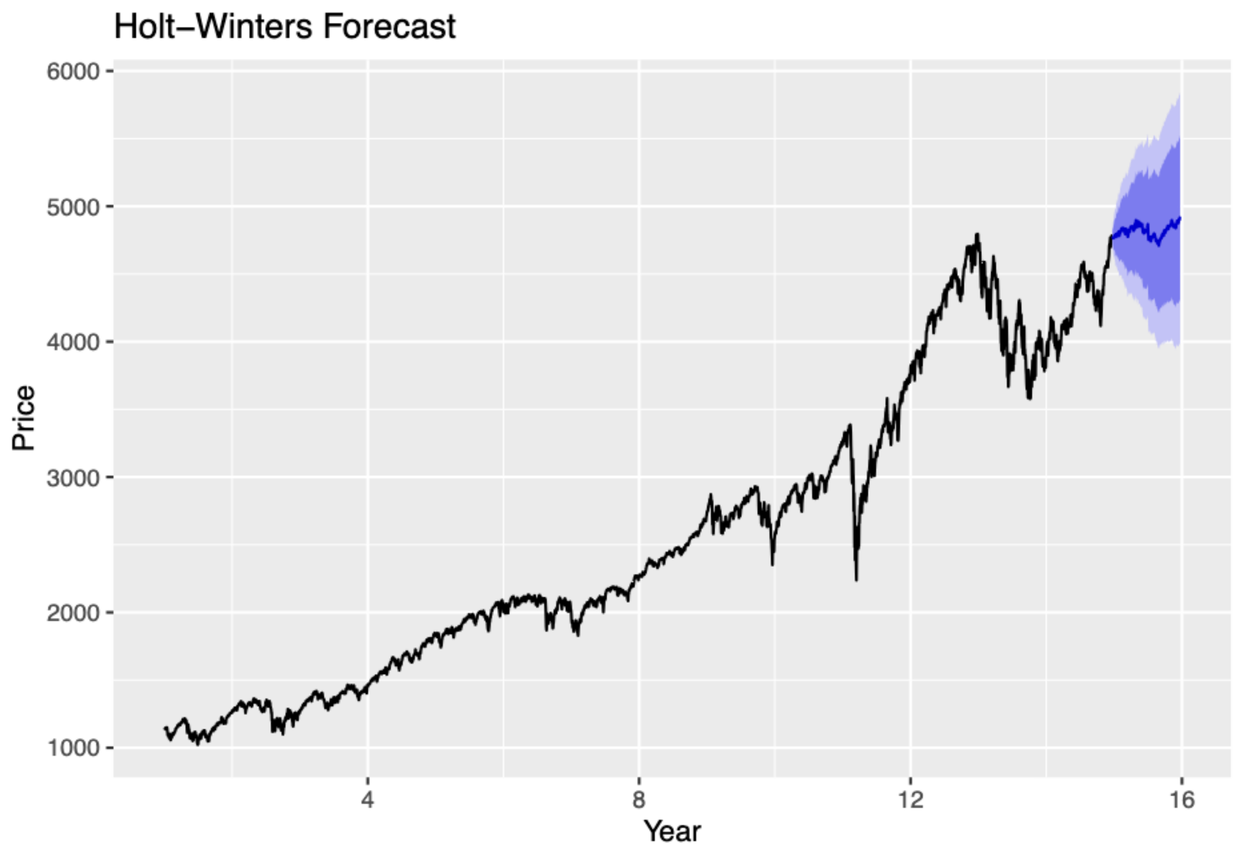
```
##   sigma:  0.011
##
##      AIC     AICc      BIC
## 51541.35 51541.37 51572.18
##
## Training set error measures:
##                     ME     RMSE      MAE          MPE      MAPE       MASE
## Training set 0.1847434 30.33971 18.76393 -0.007002447 0.7308379 0.05664869
##                   ACF1
## Training set 0.02467956
```

```
# Forecast with ETS
ets_forecast <- forecast(ets_fit, h = 252)
autoplot(ets_forecast) +
  labs(title = "ETS Forecast", x = "Year", y = "Price")
```



3. Holt-Winters Model

```
# Fit Holt-Winters model
hw_fit <- HoltWinters(sp500_ts)
hw_forecast <- forecast(hw_fit, h = 252)
autoplot(hw_forecast) +
  labs(title = "Holt-Winters Forecast", x = "Year", y = "Price")
```

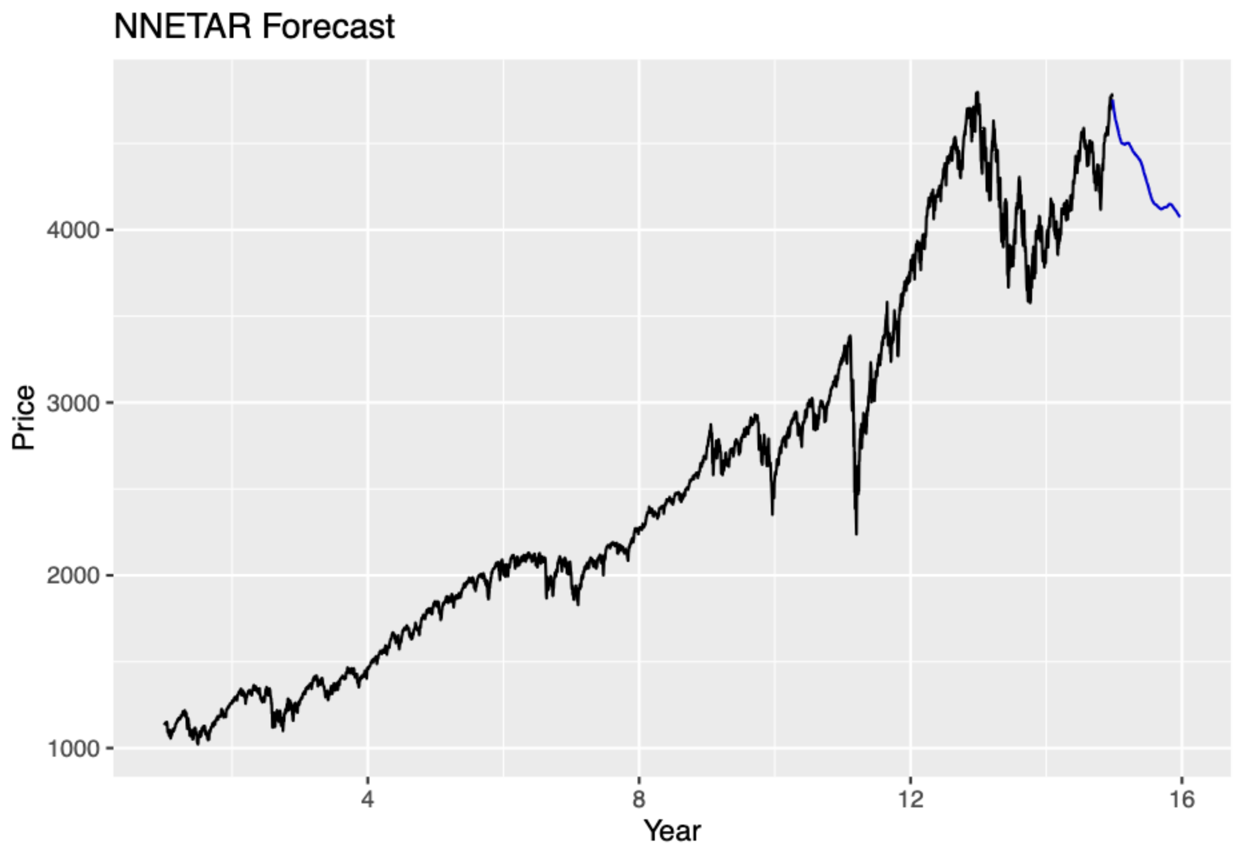


4. NNETAR Model

```
# Fit Neural Network model
nnetar_fit <- nnetar(sp500_ts)
summary(nnetar_fit)
```

```
##           Length Class        Mode     
## x         3522   ts           numeric  
## m            1   -none-       numeric  
## p            1   -none-       numeric  
## P            1   -none-       numeric  
## scalex       2   -none-       list     
## size         1   -none-       numeric  
## subset    3522   -none-       numeric  
## model       20   nnetarmodels list     
## nnetargs     0   -none-       list     
## fitted    3522   ts           numeric  
## residuals 3522   ts           numeric  
## lags        11   -none-       numeric  
## series       1   -none-       character
## method       1   -none-       character
## call         2   -none-       call     
```

```r
# Forecast with NNETAR
nnetar_forecast <- forecast(nnetar_fit, h = 252)
autoplot(nnetar_forecast) +
  labs(title = "NNETAR Forecast", x = "Year", y = "Price")
```

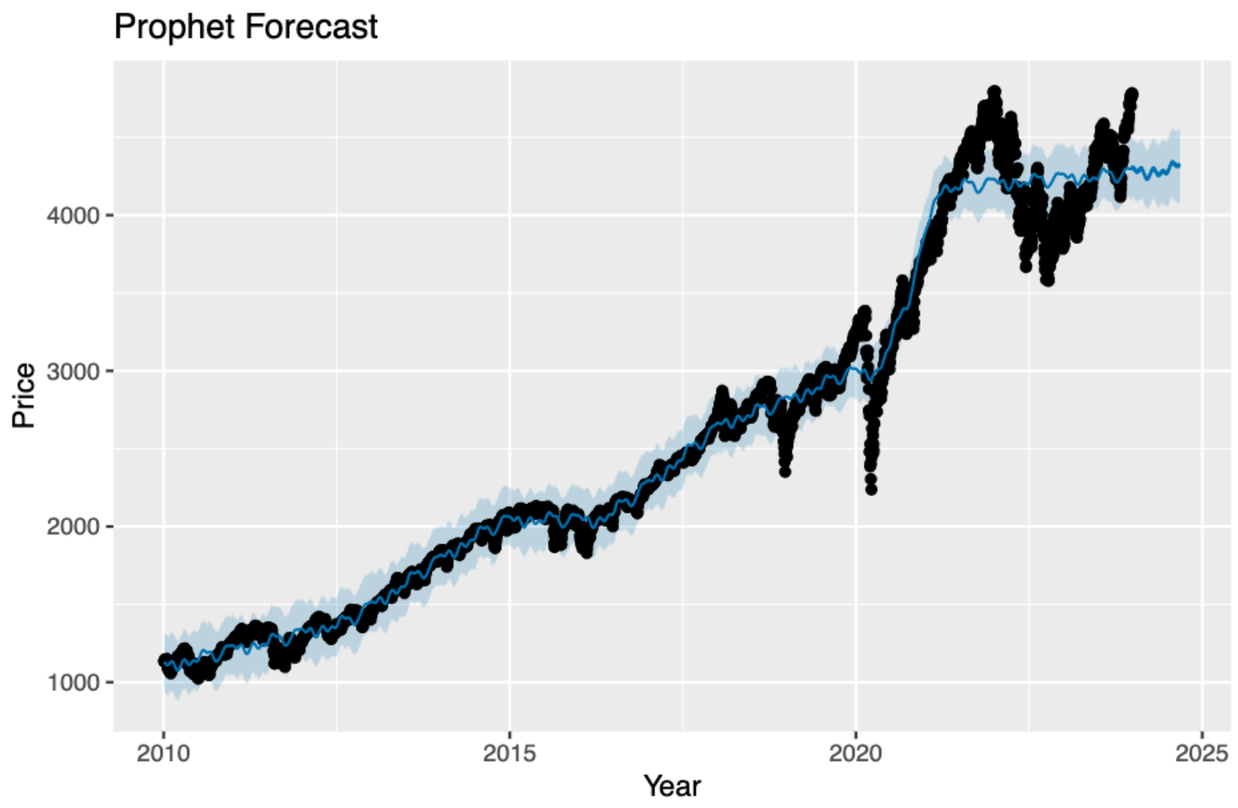


5. Prophet Model

```r
# Prepare data for Prophet
sp500_df <- data.frame(ds = index(sp500), y = as.numeric(sp500))
prophet_fit <- prophet(sp500_df)
```

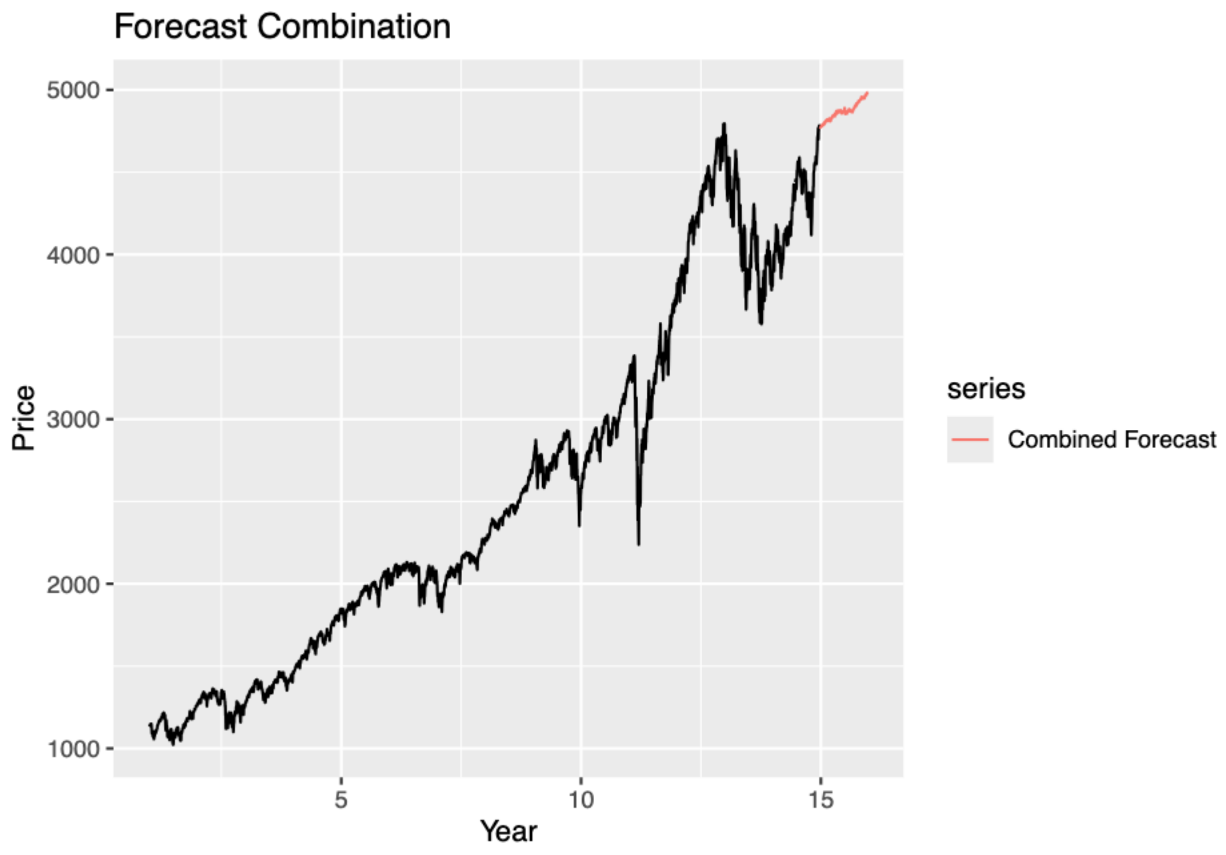
```
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.
```

```r
# Future data frame
future <- make_future_dataframe(prophet_fit, periods = 252)
forecast_prophet <- predict(prophet_fit, future)

# Plot Prophet Forecast
prophet_plot <- plot(prophet_fit, forecast_prophet) +
  labs(title = "Prophet Forecast", x = "Year", y = "Price")
prophet_plot
```



6. Forecast Combination

```r
# Combine forecasts using average
combined_forecast <- (arima_forecast$mean + ets_forecast$mean + hw_forecast$mean) / 3
autoplot(sp500_ts) +
  autolayer(combined_forecast, series = "Combined Forecast") +
  labs(title = "Forecast Combination", x = "Year", y = "Price")
```



7. Model Performance Evaluation

```r
# Split data into training and testing sets
# Determine the actual data range
end_year <- floor(end(sp500_ts)[1])
train <- window(sp500_ts, end = c(end_year - 1, 252))  # Train up to last complete year
test <- window(sp500_ts, start = c(end_year, 1))       # Test on the last year


# Calculate errors
arima_errors <- accuracy(forecast(auto.arima(train), h = length(test)), test)
ets_errors <- accuracy(forecast(ets(train), h = length(test)), test)
```

```
## Warning in ets(train): I can't handle data with frequency greater than 24.
## Seasonality will be ignored. Try stlf() if you need seasonal forecasts.
```

```r
nnetar_errors <- accuracy(forecast(nnetar(train), h = length(test)), test)

# Combine results
error_df <- data.frame(
  Model = c("ARIMA", "ETS", "NNETAR"),
  RMSE = c(arima_errors[2], ets_errors[2], nnetar_errors[2])
)
knitr::kable(error_df, caption = "Model Performance Metrics")
```

Table 1: Model Performance Metrics

Model	RMSE
ARIMA	298.1665
ETS	310.2412
NNETAR	282.7333

III. Conclusions and Future Work

The ARIMA model provided the lowest RMSE, making it the preferred model for this analysis. Future work could include incorporating exogenous variables (e.g., interest rates) to improve predictive accuracy.