Input file is taken from: [goog](https://finance.yahoo.com/quote/GOOG/history?p=GOOG)

# Load libraries

library(readr)

library(lubridate)

library(tidyverse)

library(forecast)

library(prophet)

# Read in data

data <- read\_csv("GOOG.csv")

data$Date <- dmy(data$Date)

data$Date <- ymd(data$Date)

library(ggplot2)

#data <- read.csv("GOOG.csv")

data\_ts <- ts(data$Close, frequency = 12, start = c(2004, 8))

data\_ts\_log <- log(data\_ts)

#Load required libraries

#Read in data

#data <- read.csv("GOOG.csv")

#data\_ts <- ts(data$Close, frequency = 12, start = c(2004, 8))

#data\_ts\_log <- log(data\_ts)

#Plot time series

ggplot() + geom\_line(aes(x = time(data\_ts\_log), y = data\_ts\_log), col='blue') +

labs(title = "Google Stock Price (Log Scale)", x = "Year", y = "Closing Price (Log Scale)")

#Check stationarity with ADF test

library(tseries)

adf\_test <- adf.test(data\_ts\_log)

adf\_test

cat("ADF Statistic:", adf\_test$statistic, "\n")

cat("p-value:", adf\_test$p.value, "\n")

cat("Critical Values:", adf\_test$statistic, "\n") ##Correction made here: critical values->statistic

# Convert to time series

#data\_ts <- ts(data$Close, frequency = 252, start = c(2004, 8))

decomp <- decompose(data\_ts\_log, type="multiplicative")

decomp

# Plot decomposition

par(mfrow=c(4,1))

plot(data\_ts\_log, main="Original", col='blue')

plot(decomp$trend, main="Trend", col='blue')

plot(decomp$seasonal, main="Seasonality", col='blue')

plot(decomp$random, main="Residuals", col='blue')

train\_data <- data\_ts[1:(length(data\_ts)-30)]

test\_data <- data\_ts[(length(data\_ts)-29):length(data\_ts)]

#Naive forecast

naive\_forecast <- rep(tail(train\_data, 1), length(test\_data))

naive\_forecast

#ARMA model

arma\_model <- arima(train\_data, order=c(1, 1, 1))

arma\_forecast <- predict(arma\_model, n.ahead = length(test\_data))$pred

arma\_forecast

#SARIMA model

sarima\_model <- arima(train\_data, order=c(1, 1, 1), seasonal=list(order=c(1, 1, 1), period=12))

sarima\_forecast <- predict(sarima\_model, n.ahead = length(test\_data))$pred

sarima\_forecast

#Plot actual and forecasted values

par(mfrow=c(1,1))

plot(data$Date, data\_ts, type="l", xlab="Date", ylab="Closing Price", main="Google Stock Price",col='blue')

lines(data$Date[(length(data\_ts)-29):length(data\_ts)],test\_data,type='l', col="orange")

lines(data$Date[(length(data\_ts)-29):length(data\_ts)],naive\_forecast, col="green")

lines(data$Date[(length(data\_ts)-29):length(data\_ts)],arma\_forecast, col="red")

lines(data$Date[(length(data\_ts)-29):length(data\_ts)],sarima\_forecast, col="purple")

legend("topright", legend=c("Training Data", "Test Data", "Naive Forecast", "ARIMA Forecast", "SARIMA Forecast"), col=c("blue", "orange", "green", "red", "purple"), lty=1)

##Removed, repetetive code

# Split the data into training and testing sets

#train\_ts <- window(data\_ts, end = c(2018,12))

#test\_ts <- window(data\_ts, start = c(2019,1))

train\_ts <- train\_data

test\_ts <- test\_data

#Defining a function for calculating SMAPE(symmetric mean absolute percentage error ##Addition done here

find\_smape <- function(a, f) {

return (0.01/length(a) \* sum(2\*abs(f-a) / (abs(a)+abs(f))\*100))

}

# Naive forecast

naive\_rmse <- sqrt(mean((test\_ts - naive\_forecast)^2))

naive\_rmse

naive\_mse <- mean((test\_ts - naive\_forecast)^2)

naive\_mse

naive\_mae <- sum(abs(test\_ts-naive\_forecast))/length(test\_ts)

naive\_mae

naive\_smape <- find\_smape(test\_ts,naive\_forecast)

naive\_smape

# ARIMA forecast

#arima\_model <- auto.arima(train\_ts)

#arima\_forecast <- forecast(arima\_model, h = length(test\_ts))

arima\_rmse <- sqrt(mean((test\_ts - arma\_forecast)^2))

arima\_rmse

arima\_mse <- mean((test\_ts - arma\_forecast)^2)

arima\_mse

arima\_mae <- sum(abs(test\_ts-arma\_forecast))/length(test\_ts)

arima\_mae

arima\_smape <- find\_smape(test\_ts,arma\_forecast)

arima\_smape

# SARIMA forecast

sarima\_rmse <- sqrt(mean((test\_ts - sarima\_forecast)^2))

sarima\_rmse

sarima\_mse <- mean((test\_ts - sarima\_forecast)^2)

sarima\_mse

sarima\_mae <- sum(abs(test\_ts-sarima\_forecast))/length(test\_ts)

sarima\_mae

sarima\_smape <- find\_smape(test\_ts,sarima\_forecast)

sarima\_smape

# ETS forecast

ets\_model <- ets(train\_ts)

ets\_forecast <- forecast(ets\_model, h = length(test\_ts))

ets\_rmse <- sqrt(mean((test\_ts - ets\_forecast$mean)^2))

ets\_rmse

ets\_mse <- mean((test\_ts - ets\_forecast$mean)^2)

ets\_mse

ets\_mae <- sum(abs(test\_ts-ets\_forecast$mean))/length(test\_ts)

ets\_mae

ets\_smape <- find\_smape(test\_ts,ets\_forecast$mean)

ets\_smape

# Prophet forecast

prophet\_data <- data.frame(ds = data$Date, y = data$Close)

prophet\_model <- prophet(prophet\_data,daily.seasonality=TRUE, yearly.seasonality=TRUE)

prophet\_future <- make\_future\_dataframe(prophet\_model, periods = length(test\_ts), freq = "days", include\_history = FALSE)

prophet\_forecast <- predict(prophet\_model, prophet\_future)$yhat

prophet\_rmse <- sqrt(mean((test\_ts - prophet\_forecast)^2))

prophet\_rmse

prophet\_mse <- mean((test\_ts - prophet\_forecast)^2)

prophet\_mse

prophet\_mae <- sum(abs(test\_ts-prophet\_forecast))/length(test\_ts)

prophet\_mae

#Function called with parameters to find SMAPE

prophet\_smape <- find\_smape(test\_ts, prophet\_forecast)

prophet\_smape

# Print RMSE values

cat(paste0("Naive RMSE: ", round(naive\_rmse, 2), "\n"))

cat(paste0("ARIMA RMSE: ", round(arima\_rmse, 2), "\n"))

cat(paste0("ETS RMSE: ", round(ets\_rmse, 2), "\n"))

cat(paste0("Prophet RMSE: ", round(prophet\_rmse, 2)))

# Print MSE values

cat(paste0("Naive MSE: ", round(naive\_mse, 2), "\n"))

cat(paste0("ARIMA MSE: ", round(arima\_mse, 2), "\n"))

cat(paste0("ETS MSE: ", round(ets\_mse, 2), "\n"))

cat(paste0("Prophet MSE: ", round(prophet\_mse, 2)))