Untitled

Group 3: Shaya Engelman, Julia Ferris, Amanda Fox, Jean Jimenez

2024-06-23

setwd("C:/Users/shaya/OneDrive/Documents/repos/Data624")

library(tidyverse)

## Warning: package 'ggplot2' was built under R version 4.3.3

## Warning: package 'stringr' was built under R version 4.3.2

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.3 ✔ readr 2.1.4  
## ✔ forcats 1.0.0 ✔ stringr 1.5.1  
## ✔ ggplot2 3.5.1 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.2 ✔ tidyr 1.3.0  
## ✔ purrr 1.0.2   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(readxl)  
library(here)

## Warning: package 'here' was built under R version 4.3.3

## here() starts at C:/Users/shaya/OneDrive/Documents/repos/Data624

library(tsibble)

## Warning: package 'tsibble' was built under R version 4.3.3

##   
## Attaching package: 'tsibble'  
##   
## The following object is masked from 'package:lubridate':  
##   
## interval  
##   
## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, union

library(fable)

## Warning: package 'fable' was built under R version 4.3.3

## Loading required package: fabletools

## Warning: package 'fabletools' was built under R version 4.3.3

library(ggcorrplot)  
library(forecast)

## Warning: package 'forecast' was built under R version 4.3.3

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

library(tseries)

## Warning: package 'tseries' was built under R version 4.3.3

library(zoo)

## Warning: package 'zoo' was built under R version 4.3.2

##   
## 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

library(writexl)

## Warning: package 'writexl' was built under R version 4.3.3

library(prophet)

## Warning: package 'prophet' was built under R version 4.3.3

## Loading required package: Rcpp  
## Loading required package: rlang  
##   
## Attaching package: 'rlang'  
##   
## The following objects are masked from 'package:purrr':  
##   
## %@%, flatten, flatten\_chr, flatten\_dbl, flatten\_int, flatten\_lgl,  
## flatten\_raw, invoke, splice

# Load data

complete\_data <- read\_excel(here("Project\_1", "Prompt", "Data Set for Class.xls"))  
  
# Split the data into 6 different data frames by the column "category"  
data\_list <- split(complete\_data, complete\_data$category)  
  
# Extract category three  
category\_five <- data\_list$`S05` |>  
 select(-category) # drop the category column  
  
summary(category\_five)

## SeriesInd Var01 Var02 Var03   
## Min. :40669 Min. : 56.99 Min. : 4156600 Min. : 55.94   
## 1st Qu.:41304 1st Qu.: 78.86 1st Qu.: 11201200 1st Qu.: 77.21   
## Median :41946 Median : 85.95 Median : 14578800 Median : 84.87   
## Mean :41945 Mean : 84.21 Mean : 16791350 Mean : 82.97   
## 3rd Qu.:42586 3rd Qu.: 91.00 3rd Qu.: 20021200 3rd Qu.: 89.74   
## Max. :43221 Max. :104.76 Max. :118023500 Max. :103.95   
## NA's :143 NA's :141 NA's :145   
## Var05 Var07   
## Min. : 56.85 Min. : 56.57   
## 1st Qu.: 77.94 1st Qu.: 78.10   
## Median : 85.32 Median : 85.43   
## Mean : 83.58 Mean : 83.63   
## 3rd Qu.: 90.37 3rd Qu.: 90.46   
## Max. :104.42 Max. :104.38   
## NA's :145 NA's :145

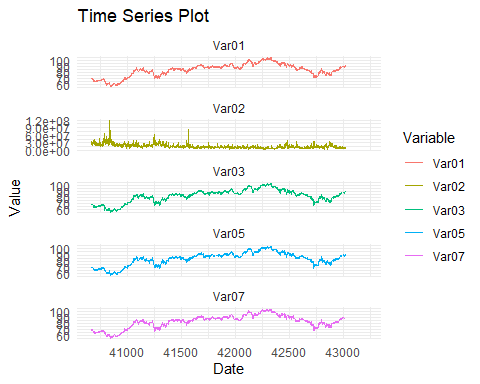
# Data Exploration

## Exploring the relationship between all variables

For this project, only variables ‘Var02’ and ‘Var03’ will be forecasted. However, it can still be useful to explore the other variables to see if there are any insights.

# Plot the five variables as faceted plots to allow for different scales on the y-axis  
category\_five |>  
 pivot\_longer(cols = -SeriesInd, names\_to = "variable", values\_to = "value") |>  
 ggplot(aes(x = SeriesInd, y = value, color = variable)) +  
 geom\_line() +  
 labs(title = "Time Series Plot", x = "Date", y = "Value", color = "Variable") +  
 theme\_minimal() +  
 facet\_wrap(~ variable, scales = "free\_y", ncol = 1)

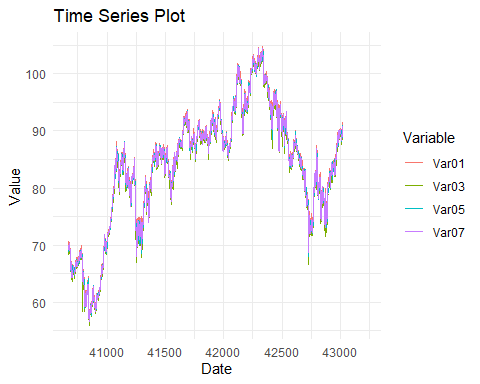
## Warning: Removed 700 rows containing missing values or values outside the scale range  
## (`geom\_line()`).



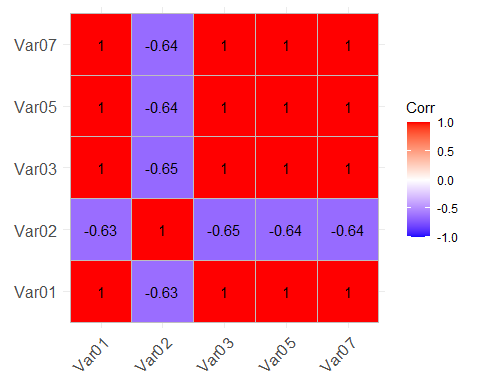
The above plots reveal an almost perfect correlation between variables Var01, Var03, Var05, and Var07. Var02 seems to be something entirely different and on a much larger scale. To confirm the relationship between the variables, I will recreate this plot with those variables stacked on the same set of axes, additionally, I will calculate the correlation between them.

category\_five |>  
 pivot\_longer(cols = -SeriesInd, names\_to = "variable", values\_to = "value") |>  
 filter(variable %in% c("Var01", "Var03", "Var05", "Var07")) |>  
 ggplot(aes(x = SeriesInd, y = value, color = variable)) +  
 geom\_line() +  
 labs(title = "Time Series Plot", x = "Date", y = "Value", color = "Variable") +  
 theme\_minimal()

## Warning: Removed 560 rows containing missing values or values outside the scale range  
## (`geom\_line()`).



# Compute correlation matrix  
category\_five |>  
 select(Var01, Var02, Var03, Var05, Var07) |>  
 na.omit() |> # cor() does not handle missing values  
 cor() |>  
 ggcorrplot(lab = TRUE)



The above plots confirm the practically perfect correlation between Var01, Var03, Var05, and Var07 and also reveal a pretty significant negative correlation between Var02 and the other variables. Moreover, the stacked lineplots reveal that variables Var01, Var03, Var05, and Var07 are not just extremely correlated, but also have almost the same values. This suggests that they are essentially the same variable, just with different names. This allows me to not worry about them being confounding variables when trying to forecast individual variables and also allows me to use other variables to fill in missing values.

## Summary statistics

The first step in analyzing the data is to generate summary statistics for the variable to be forecasted, ‘Var03’. First, I will convert the data to a tsibble object to have the data in a format that is easier to work with for time series analysis. Var02 will be worked on in a different section.

cat5\_var3 <- category\_five |>  
 select(SeriesInd, Var03)  
  
# Convert to tsibble  
cat5\_var3 <- as\_tsibble(cat5\_var3, index = SeriesInd)   
  
# Check the tsibble object  
length(cat5\_var3$SeriesInd)

## [1] 1762

# Summary statistics  
summary(cat5\_var3)

## SeriesInd Var03   
## Min. :40669 Min. : 55.94   
## 1st Qu.:41304 1st Qu.: 77.21   
## Median :41946 Median : 84.87   
## Mean :41945 Mean : 82.97   
## 3rd Qu.:42586 3rd Qu.: 89.74   
## Max. :43221 Max. :103.95   
## NA's :145

Since the data is completely deidentified, I do not know exactly what the ‘SeriesInd’ column represents. The actual values appear to be Excel date serials (the number of days since December 30, 1899), but since this is not confirmed, I will leave them as an index unless there is a reason to change them. An interesting observation is that while the SeriesInd column does not have any missing values, it isn’t completely continuous. The data ranges from 40669 to 43221, which is a range of 2552 observations, however the length of the data is only 1762. Since this isn’t showing up as missing values, the data for those rows must be entirely missing from the dataset. A quick glance through the data seems to suggest a pattern of five consecutive indices, followed by a jump of 2 indices. This is something to keep in mind when working with the data and further suggests that the index is a date column only measuring business days.

We see 145 missing values in ‘Var03’ The majority of the missing values are at the end of the data set, due to there being rows in the data to be forecast. Thus they aren’t actually missing values. That leaves us with 5 missing values in the middle of the data set. Since the data is a time series, it is possible to impute these missing values using a method such as linear interpolation. Since there are only 5 missing values, the method of imputation is not as important as it would be if there were more missing values.

# Remove last 140 rows  
n <- nrow(cat5\_var3)  
cat5\_train <- cat5\_var3[-c((n-139):n), ]  
  
# Print the rows of missing values of the entire data set  
cat5\_train |>  
 filter(is.na(Var03))

## # A tsibble: 5 x 2 [1]  
## SeriesInd Var03  
## <dbl> <dbl>  
## 1 41821 NA  
## 2 42897 NA  
## 3 42898 NA  
## 4 42997 NA  
## 5 43000 NA

# Impute missing values  
cat5\_train$Var03 <- na.approx(cat5\_train$Var03)  
  
# Check for missing values to make sure they were filled in  
cat5\_train |>  
 filter(is.na(Var03))

## # A tsibble: 0 x 2 [?]  
## # ℹ 2 variables: SeriesInd <dbl>, Var03 <dbl>

After handling the missing values with a more robust method, I will now add the implicitly missing rows into the data and fill them in with the data from the last row. I specifically am not using the same method as before because for these missing values, there likely was 0 of whatever the data is measuring (sales or stock price or whatever). Using an average of the two closest values would not be appropriate in this case.

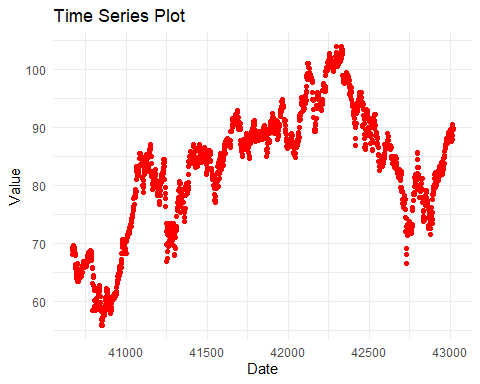
# Add the implicitly missing rows  
cat5\_train <- cat5\_train |>   
 fill\_gaps()  
  
# Fill NAs with the last value  
cat5\_train$Var03 <- na.locf(cat5\_train$Var03)  
  
# Check for missing values  
cat5\_train |>  
 filter(is.na(Var03))

## # A tsibble: 0 x 2 [?]  
## # ℹ 2 variables: SeriesInd <dbl>, Var03 <dbl>

## Visualizing the Time Series

We now move onto basic exploration of the variable to be forecasted, ‘Var03’.

# Plot the time series  
cat5\_train |>  
 ggplot(aes(x = SeriesInd, y = Var03)) +  
 geom\_point(color = "red") +  
 labs(title = "Time Series Plot", x = "Date", y = "Value") +  
 theme\_minimal()



There aren’t any major outliers but there is some minor outliers at just under index 52750. Additionally, the data seems to have an overall upwards trend but has a lot of noise and that might just be an illusion. If there is no overall trend, then those points mentiponed earlier are not outliers. Theyt are well within the range of the overall data, merely far from the majority of the data points at that point in time.

There is also no obvious seasonality or cyclical patterns in the data. There might be a seasonal pattern in the middle of forming but the data is not long enough to capture that.

## Seasonality

To determine the seasonality of the data, I will use the ‘findfrequency()’ function from the forecast package. The ‘findfrequency()’ function will return the frequency of the data, which will allow me to determine the seasonality of the data. The frequency of the data is the number of observations per season or cycle.

# Find the frequency of the data  
findfrequency(cat5\_train$Var03)

## [1] 1

The frequency of the data is 1, which means that there is no seasonality in the data. This is consistent with the visual inspection of the data, which did not show any obvious seasonality.

## Stationarity

The next step is to check for stationarity in the data. This is important because many time series models assume that the data is stationary. If the data is not stationary, then it will need to be transformed in some way to make it stationary. The most common way to check for stationarity is to use the Augmented Dickey-Fuller test. This test has a null hypothesis that the data is not stationary. If the p-value is less than 0.05, then the null hypothesis is rejected and the data is considered stationary.

# Augmented Dickey-Fuller test  
adf.test(cat5\_train$Var03)

##   
## Augmented Dickey-Fuller Test  
##   
## data: cat5\_train$Var03  
## Dickey-Fuller = -1.976, Lag order = 13, p-value = 0.5885  
## alternative hypothesis: stationary

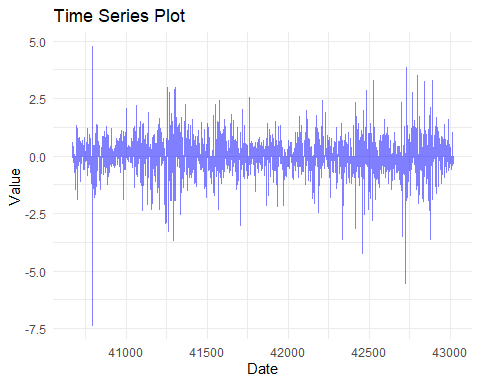
The p-value of the ADF test is extremely high, which means that we fail to reject the null hypothesis that the data is not stationary. This suggests that the data is not stationary and will need to be transformed in some way to make it stationary.

I will use the ndiffs() function from the forecast package to determine the number of differences needed to make the data stationary. I expect a value of 1, as the data seemingly has an upward trend with no seasonality.

# Determine the number of differences needed to make the data stationary  
ndiffs(cat5\_train$Var03)

## [1] 1

# Take the first difference  
var3\_diff <- diff(cat5\_train$Var03)  
  
# Plot the differenced data  
ggplot() +  
 geom\_line(aes(x = cat5\_train$SeriesInd[-1], y = var3\_diff), color = "blue", alpha = 0.5) +  
 labs(title = "Time Series Plot", x = "Date", y = "Value", color = "Variable") +  
 theme\_minimal()



The first difference of the data appears to be stationary. I will now run the ADF test on the differenced data to confirm that it is stationary.

# Augmented Dickey-Fuller test on differenced data  
adf.test(var3\_diff)

## Warning in adf.test(var3\_diff): p-value smaller than printed p-value

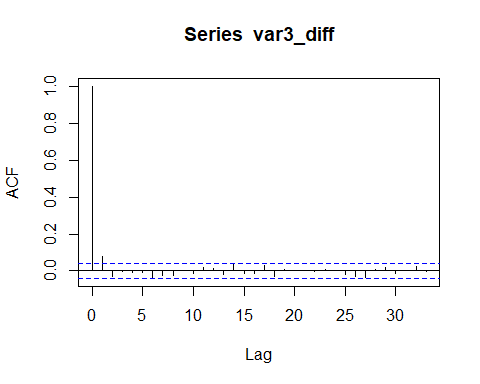
##   
## Augmented Dickey-Fuller Test  
##   
## data: var3\_diff  
## Dickey-Fuller = -13.018, Lag order = 13, p-value = 0.01  
## alternative hypothesis: stationary

We now see that the p-value is extremely low, implying that the data is now stationary. We can also view this in the above plot, where we see the mean of the data is now constant. We do note two periods of high variability in the data, but the mean is constant.

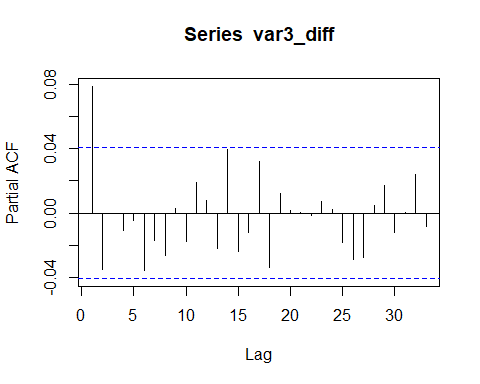
## ACF and PACF

The next step in the analysis is to make sure there is no longer any autocorrelation in the data. We can do this by plotting the autocorrelation function (ACF) and partial autocorrelation function (PACF) of the data. The ACF and PACF plots will show the correlation between the data and its lagged values. If the data is stationary, then the ACF and PACF plots should show no correlation between the data and its lagged values. Here, we got slightly different results for the two variables so I will discuss them separately.

# ACF and PACF plots  
acf(var3\_diff, plot = T)



pacf(var3\_diff, plot = T)



The ACF and PACF plots show autocorrelation only at lag 1, which is expected for stationary data. All the other lags are within the confidence interval (the area between the blue lines), which means they are not statistically significant. This implies that the data is stationary and there is no autocorrelation in the data.

# Modeling

## ARIMA Model

The next step is to fit an ARIMA model to the data. ARIMA stands for AutoRegressive Integrated Moving Average. An ARIMA model is a type of time series model that combines autoregressive (AR) and moving average (MA) components with differencing to make the data stationary. The ARIMA model is defined by three parameters: p, d, and q. The p parameter is the number of autoregressive terms, the d parameter is the number of differences needed to make the data stationary, and the q parameter is the number of moving average terms.

To determine the parameters of the ARIMA model, I will use the auto.arima() function from the forecast package. The auto.arima() function will automatically select the best ARIMA model based on the AIC (Akaike Information Criterion) value. The AIC value is a measure of the goodness of fit of the model, with lower values indicating a better fit.

# Fit ARIMA model  
arima\_model\_diff <- auto.arima(var3\_diff)  
  
# Summary of ARIMA model  
summary(arima\_model\_diff)

## Series: var3\_diff   
## ARIMA(0,0,1) with zero mean   
##   
## Coefficients:  
## ma1  
## 0.0839  
## s.e. 0.0211  
##   
## sigma^2 = 0.5569: log likelihood = -2648.52  
## AIC=5301.04 AICc=5301.04 BIC=5312.56  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE ACF1  
## Training set 0.008434248 0.7461196 0.4548361 NaN Inf 0.6829121 -0.002458467

The auto.arima() function selected an ARIMA(0,0,1) model with a constant term. The model has a MA(1) term, which means that the model uses the moving average of the previous observation to predict the next observation.

Since we differenced the data by 1, we can use an ARIMA(0,1,1) model to forecast the original data. This is identical to the ARIMA(0,0,1) model on the differenced data but will allow much easier forecasting of untransformed data.

# Fit ARIMA model to original data  
arima\_model <- Arima(cat5\_train$Var03, order = c(0,1,1))  
  
# Summary of ARIMA model  
summary(arima\_model)

## Series: cat5\_train$Var03   
## ARIMA(0,1,1)   
##   
## Coefficients:  
## ma1  
## 0.0839  
## s.e. 0.0211  
##   
## sigma^2 = 0.5569: log likelihood = -2648.52  
## AIC=5301.04 AICc=5301.04 BIC=5312.56  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 0.008459641 0.7459624 0.4546718 0.006599418 0.5630772 1.014359  
## ACF1  
## Training set -0.00243113

## ARIMA with Drift

The ARIMA model with drift is a variation of the ARIMA model that includes a linear trend term. The ARIMA model with drift is defined by four parameters: p, d, q, and k. The p parameter is the number of autoregressive terms, the d parameter is the number of differences needed to make the data stationary, the q parameter is the number of moving average terms, and the k parameter is the drift term.

# Fit ARIMA model with drift  
arima\_drift\_model <- Arima(cat5\_train$Var03, order = c(0,1,1), include.drift = T)  
  
# Summary of ARIMA model with drift  
summary(arima\_drift\_model)

## Series: cat5\_train$Var03   
## ARIMA(0,1,1) with drift   
##   
## Coefficients:  
## ma1 drift  
## 0.0838 0.0091  
## s.e. 0.0211 0.0167  
##   
## sigma^2 = 0.5571: log likelihood = -2648.37  
## AIC=5302.73 AICc=5302.74 BIC=5320.02  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 2.357975e-05 0.7459145 0.4552601 -0.003747695 0.5638011 1.015672  
## ACF1  
## Training set -0.002309169

## ETS Model

Another type of time series model is the ETS (Error, Trend, Seasonality) model. The ETS model is a type of exponential smoothing model that decomposes the data into error, trend, and seasonality components. The ETS model is defined by three parameters: error, trend, and seasonality. The error parameter is the type of error model (additive or multiplicative), the trend parameter is the type of trend model (additive or multiplicative), and the seasonality parameter is the type of seasonality model (additive, multiplicative, or none).

To determine the parameters of the ETS model, I will use the ets() function from the forecast package. The ets() function will automatically select the best ETS model based on the AIC value.

# Fit ETS model  
ets\_model <- ets(cat5\_train$Var03)  
  
# Summary of ETS model  
summary(ets\_model)

## ETS(A,N,N)   
##   
## Call:  
## ets(y = cat5\_train$Var03)   
##   
## Smoothing parameters:  
## alpha = 0.9999   
##   
## Initial states:  
## l = 68.2059   
##   
## sigma: 0.7488  
##   
## AIC AICc BIC   
## 16909.77 16909.78 16927.06   
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 0.009135698 0.7484419 0.4480661 0.00705294 0.5551549 0.9996219  
## ACF1  
## Training set 0.07860425

The ets() function selected an ETS(A,N,N) model. The model has an additive error term and no trend or seasonality components. This means that the model uses the additive error term to predict the next observation.

## Prophet Model

The Prophet model is a type of time series model developed by Facebook that is designed to handle time series data with seasonality and holidays. The Prophet model is defined by several parameters, including the seasonality mode, the seasonality prior scale, and the holidays prior scale.

This is the first model where I am making assumptions about the time series. As noted in the beginning, the data seems to be daily values with missing data for weekends. Until now, I left the data anonymized to avoid making assumptions. For this model however, I need to make that assumption. To fit the Prophet model to the data, I will use the prophet() function from the prophet package. The prophet() function will automatically select the best Prophet model based on the AIC value.

The prophet model requires the data to be in a specific format, with the date column named ‘ds’ and the value column named ‘y’. I will convert the data to this format before fitting the Prophet model.

# Convert data to Prophet format  
cat5\_train\_prophet <- cat5\_train |>  
 rename(ds = SeriesInd, y = Var03) |>  
 mutate(ds = as.Date(ds, origin = "1899-12-30"))  
  
head(cat5\_train\_prophet)

## # A tsibble: 6 x 2 [1D]  
## ds y  
## <date> <dbl>  
## 1 2011-05-06 68.2  
## 2 2011-05-07 68.8  
## 3 2011-05-08 69.3  
## 4 2011-05-09 69.4  
## 5 2011-05-10 69.2  
## 6 2011-05-11 69.2

# Fit Prophet model  
prophet\_model <- prophet(cat5\_train\_prophet, daily.seasonality = F, weekly.seasonality = F, yearly.seasonality = F)  
  
# Summary of Prophet model  
summary(prophet\_model)

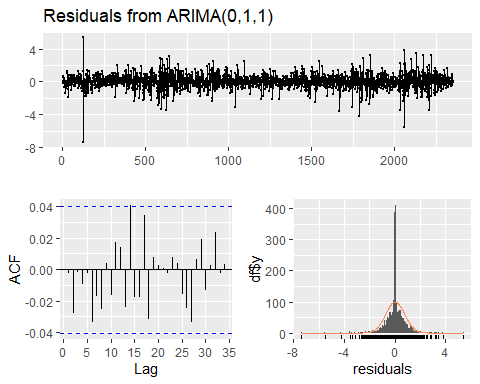
## Length Class Mode   
## growth 1 -none- character  
## changepoints 25 POSIXct numeric   
## n.changepoints 1 -none- numeric   
## changepoint.range 1 -none- numeric   
## yearly.seasonality 1 -none- logical   
## weekly.seasonality 1 -none- logical   
## daily.seasonality 1 -none- logical   
## holidays 0 -none- NULL   
## seasonality.mode 1 -none- character  
## seasonality.prior.scale 1 -none- numeric   
## changepoint.prior.scale 1 -none- numeric   
## holidays.prior.scale 1 -none- numeric   
## mcmc.samples 1 -none- numeric   
## interval.width 1 -none- numeric   
## uncertainty.samples 1 -none- numeric   
## specified.changepoints 1 -none- logical   
## start 1 POSIXct numeric   
## y.scale 1 -none- numeric   
## logistic.floor 1 -none- logical   
## t.scale 1 -none- numeric   
## changepoints.t 25 -none- numeric   
## seasonalities 0 -none- list   
## extra\_regressors 0 -none- list   
## country\_holidays 0 -none- NULL   
## stan.fit 4 -none- list   
## params 6 -none- list   
## history 5 tbl\_df list   
## history.dates 2353 POSIXct numeric   
## train.holiday.names 0 -none- NULL   
## train.component.cols 3 data.frame list   
## component.modes 2 -none- list   
## fit.kwargs 0 -none- list

## Comparing models

For each model, I will check the residuals to ensure they are normally distributed and have constant variance. I will also generate forecasts and plot to see which looks the best.

### ARIMA Model

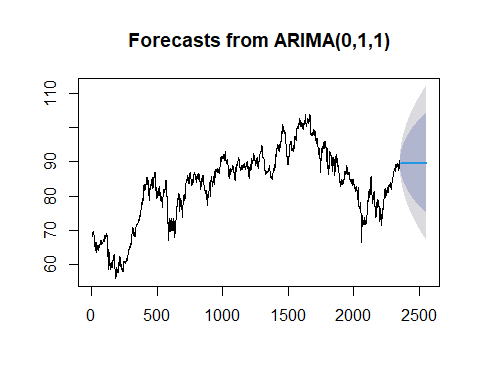
# Check residuals  
checkresiduals(arima\_model)



##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(0,1,1)  
## Q\* = 7.7234, df = 9, p-value = 0.5622  
##   
## Model df: 1. Total lags used: 10

The residuals of the ARIMA model appear to be normally distributed and have constant variance. The ACF plot of the residuals shows no autocorrelation, which indicates that the model is a good fit for the data.

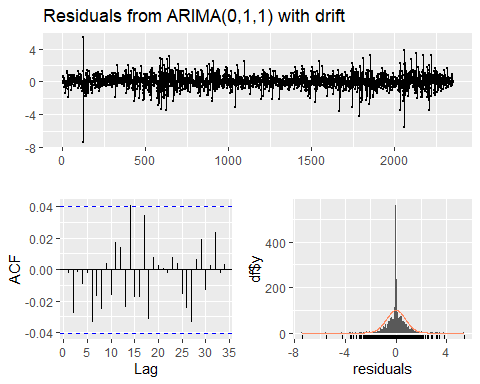
# Generate forecasts  
arima\_forecast <- forecast(arima\_model, h = 200)  
  
# Plot forecasts  
plot(arima\_forecast)



The ARIMA model appears to provide very wide confidence intervals, which is likely due to the high variability in the data. The forecasted values are the means of the data but their CIs are very wide to be useful. Additionally, the successive forecasts are all the same. Since the data does seem to have some trend, this model isn’t very useful.

### ARIMA with Drift Model

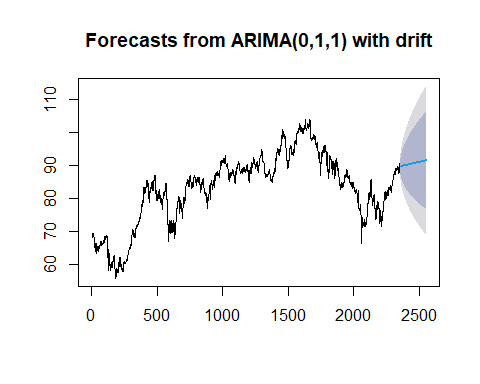
# Check residuals  
checkresiduals(arima\_drift\_model)



##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(0,1,1) with drift  
## Q\* = 7.7238, df = 9, p-value = 0.5622  
##   
## Model df: 1. Total lags used: 10

The residuals of the ARIMA model with drift appear to be normally distributed and have constant variance. The ACF plot of the residuals shows no autocorrelation, which indicates that the model is a good fit for the data.

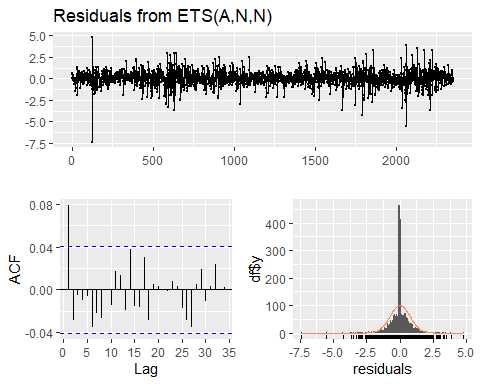
# Generate forecasts  
arima\_drift\_forecast <- forecast(arima\_drift\_model, h = 200)  
  
# Plot forecasts  
plot(arima\_drift\_forecast)



The ARIMA model with drift provides similar results to the ARIMA model. The confidence intervals are very wide with limited utility. However, the successive forecasts are not all the same, which **might** be an improvement ove the previous model.

### ETS Model

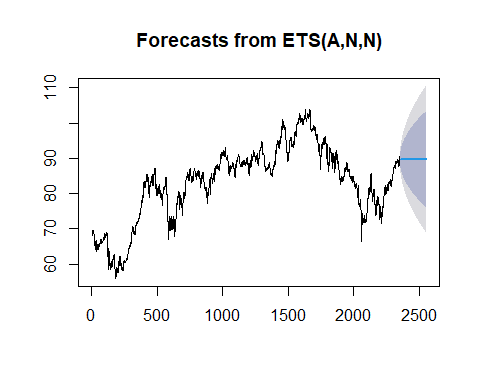
# Check residuals  
checkresiduals(ets\_model)



##   
## Ljung-Box test  
##   
## data: Residuals from ETS(A,N,N)  
## Q\* = 23.172, df = 10, p-value = 0.01013  
##   
## Model df: 0. Total lags used: 10

The residuals of the ETS model appear to be normally distributed and have constant variance. The ACF plot of the residuals shows no autocorrelation, which indicates that the model is a good fit for the data.

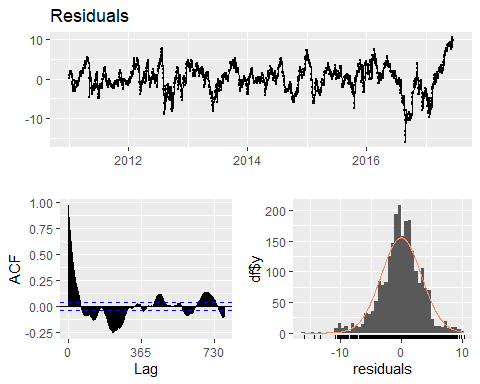
# Generate forecasts  
ets\_forecast <- forecast(ets\_model, h = 200)  
  
# Plot forecasts  
plot(ets\_forecast)



The ETS model provides slightly better results than the arima models. The confidence intervals are still wide but slightly smaller than the ARIMA models. The successive forecasts are also all the same but there does appear to be an extremely large amount of variablity in the data that is likely causing this.

### Prophet Model

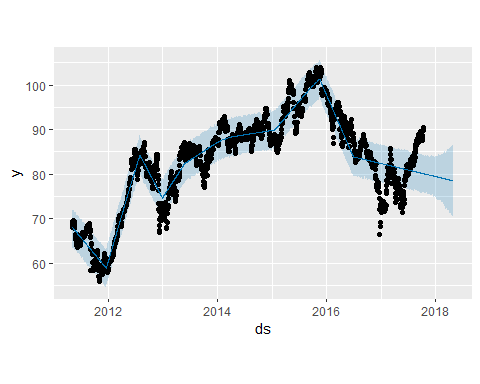
# Predict on the training data  
df\_train\_pred <- predict(prophet\_model, cat5\_train\_prophet)  
  
# Calculate residuals  
residuals <- cat5\_train\_prophet$y - df\_train\_pred$yhat  
  
# Create a time series object  
residuals\_ts <- ts(residuals, start = c(year(min(cat5\_train\_prophet$ds)), month(min(cat5\_train\_prophet$ds))), frequency = 365)  
  
# Use checkresiduals() function from the forecast package  
checkresiduals(residuals\_ts)



##   
## Ljung-Box test  
##   
## data: Residuals  
## Q\* = 47947, df = 471, p-value < 2.2e-16  
##   
## Model df: 0. Total lags used: 471

These residuals are much more interesting. They do appear to be normally distributed but the ACF plot seems to show significant autocorrelation.

# Create a dataframe for future dates (e.g., forecasting the next 200 days)  
future <- make\_future\_dataframe(prophet\_model, periods = 200)  
  
# Generate forecasts  
prophet\_forecast <- predict(prophet\_model, future)  
  
# Plot forecasts  
plot(prophet\_model, prophet\_forecast)



The forecasts for the Prophet model are significantly different than the other models. The confidence intervals are much smaller and the forecasts have a clear negative trend. If these forecasts are correct, then this model would be the best since it gives the most detailed forecasts.

## Model selection

I will use the Prophet model for the final forecast. The Prophet model provides the best results with the smallest confidence intervals and the most detailed forecasts. The ARIMA model with drift is at risk of overfitting for the drift and also has an extremely wide confidence interval. The ETS model provides slightly better results than the ARIMA model while still being supported by the overall shape of the ARIMA model’s similar forecasts, however, the forecasted values are all the same (basically just an average of the data due to all the noise and lack of trend or seasonality). The Prophet model seems to be the best model for this data.I am hesitant to use the Prophet model because it is the only model that requires assumptions about the data, the residuals are not as clean as the other models, and the forecasts are significantly different than the other models. Additionally, I do not have any experience with the Prophet model and am worried I might have made a mistake in the implementation. Nevertheless, I will use the Prophet model for the final forecast due to it providing the only meaningful forecasts. I would like to come back to this when I have the actual data we are forecasting to see if the Prophet model is the best model or the most inaccurate one.

# Forecasting

I will now add the forecasts to the data and save them to an Excel file.

# Extract the mean forecasted values  
forecasted\_values <- prophet\_forecast$yhat  
  
# Combine the actual values with the forecasted values  
actual\_values <- cat5\_train$Var03  
  
# Create the combined vector  
var\_3\_complete <- c(actual\_values, forecasted\_values[(length(actual\_values) + 1):length(forecasted\_values)])

# Select last 200 entries from prophet\_forecast$yhat  
future\_forecast <- tail(prophet\_forecast$yhat, 200)  
  
# Combine actual data points and future forecasts  
var\_3\_complete <- c(cat5\_train\_prophet$y, future\_forecast)

# convert to a dataframe and add the SeriesInd column with the correct dates  
var\_3\_complete\_df <- data.frame(  
 SeriesInd = prophet\_forecast$ds,  
 Var03 = as.numeric(var\_3\_complete)  
)  
  
# View  
head(var\_3\_complete\_df)

## SeriesInd Var03  
## 1 2011-05-06 68.19  
## 2 2011-05-07 68.80  
## 3 2011-05-08 69.34  
## 4 2011-05-09 69.42  
## 5 2011-05-10 69.22  
## 6 2011-05-11 69.22

# Convert the SeriesInd column to a numeric value  
var\_3\_complete\_df$SeriesInd <- time(var\_3\_complete\_df$SeriesInd)  
  
# Add 40668 to the SeriesInd column to convert it back to Excel date serials  
var\_3\_complete\_df$SeriesInd <- as.numeric(var\_3\_complete\_df$SeriesInd) + 40668  
  
# View  
head(var\_3\_complete\_df)

## SeriesInd Var03  
## 1 40669 68.19  
## 2 40670 68.80  
## 3 40671 69.34  
## 4 40672 69.42  
## 5 40673 69.22  
## 6 40674 69.22

# Replace Var03 in category\_five with Var03 from var\_3\_complete  
category\_five\_test <- category\_five |>  
 left\_join(var\_3\_complete\_df, by = "SeriesInd") |>  
 select(-Var03.x) |>  
 rename(Var03 = Var03.y)  
  
# Check the updated category\_five  
head(category\_five\_test)

## # A tibble: 6 × 6  
## SeriesInd Var01 Var02 Var05 Var07 Var03  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 40669 69.3 27809100 68.7 69.2 68.2  
## 2 40670 69.4 30174700 69.2 69.4 68.8  
## 3 40671 70.6 35044700 69.4 70.0 69.3  
## 4 40672 70.1 27192100 69.9 69.8 69.4  
## 5 40673 69.8 24891800 69.7 69.5 69.2  
## 6 40676 70.5 30685000 69.9 70.3 69.7

tail(category\_five\_test)

## # A tibble: 6 × 6  
## SeriesInd Var01 Var02 Var05 Var07 Var03  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 43213 NA NA NA NA 78.6  
## 2 43214 NA NA NA NA 78.6  
## 3 43218 NA NA NA NA 78.5  
## 4 43219 NA NA NA NA 78.5  
## 5 43220 NA NA NA NA 78.5  
## 6 43221 NA NA NA NA 78.5

# Save the updated data  
write\_xlsx(category\_five\_test, here("Project\_1", "Prompt", "category\_five\_forecasted.xlsx"))