Tesla Stock Closing Prices forecasting project

Lindokuhle Tshongolo

2023-07-06

Introduction

In this analysis, we will be running the forecasting of future closing stock prices of the tesla automobile company. We will aggregate these closing prices into weekly and monthly averaged prices.

This analysis will first run exploratory analysis of the data, to find any salient patterns, such as seasonality, trend or heteroscedasticy. Such findings, will inform which models we will build to attempt the forecasting task. Each model we pick, we will train it on a training set from 2010 till 2021, and then subsequently test it on unseen data from 2022 till 2023. We will use the accuracy function to test the forecast errors of our model on the unseen data relative to the forecasts. In this analysis we will also assess their distributional appropriateness against the benchmark model and use the Continuous Ranked Probability Score which is an extention of the Winker Score to assess how often the prediction intervals miss the actual future values in the test data, and thus judge the distributional appropriateness of each of these models.

```
library(fpp3)
```

```
## ── Attaching packages ──────────────────────────────────────────────── fpp3
0.5 ──

## ✓ tibble      3.2.1     ✓ tsibble     1.1.3
## ✓ dplyr       1.1.2     ✓ tsibbledata 0.4.1
## ✓ tidyr       1.3.0     ✓ feasts      0.3.1
## ✓ lubridate   1.9.2     ✓ fable       0.3.3
## ✓ ggplot2     3.4.2     ✓ fabletools  0.3.3

## Warning: package 'tibble' was built under R version 4.2.3

## Warning: package 'dplyr' was built under R version 4.2.3

## Warning: package 'tidyr' was built under R version 4.2.3

## Warning: package 'lubridate' was built under R version 4.2.3

## Warning: package 'ggplot2' was built under R version 4.2.3

## Warning: package 'feasts' was built under R version 4.2.3

## Warning: package 'fabletools' was built under R version 4.2.3

## Warning: package 'fable' was built under R version 4.2.3
## ── Conflicts ─────────────────────────────────────────────────
fpp3_conflicts ──
## ✖ lubridate::date()    masks base::date()
## ✖ dplyr::filter()      masks stats::filter()
## ✖ tsibble::intersect() masks base::intersect()
## ✖ tsibble::interval()  masks lubridate::interval()
## ✖ dplyr::lag()         masks stats::lag()
## ✖ tsibble::setdiff()   masks base::setdiff()
## ✖ tsibble::union()     masks base::union()

library(lubridate)
library(pdfetch)

## Warning: package 'pdfetch' was built under R version 4.2.3

library(broom)

## Warning: package 'broom' was built under R version 4.2.3

library(xts)

## Warning: package 'xts' was built under R version 4.2.3

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 4.2.3

##
## 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

library(tidyquant)

## Warning: package 'tidyquant' was built under R version 4.2.3

## Loading required package: PerformanceAnalytics

## Warning: package 'PerformanceAnalytics' was built under R version 4.2.3

##
## Attaching package: 'PerformanceAnalytics'

## The following object is masked from 'package:fpp3':
##
##     prices

## The following object is masked from 'package:graphics':
##
##     legend

## Loading required package: quantmod

## Warning: package 'quantmod' was built under R version 4.2.3

## Loading required package: TTR

## Warning: package 'TTR' was built under R version 4.2.3
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo

##
## Attaching package: 'tidyquant'

## The following object is masked from 'package:fable':
##
##     VAR

library(pdfetch)
library(ggplot2)
```

We initially get the data into our working environment by reading the csv file and assigning it to the variable tsla.

```
library(readr)

## Warning: package 'readr' was built under R version 4.2.3

tsla <- read_csv("TSLA (1).csv")

## Rows: 3309 Columns: 7
## ── Column specification
────────────────────────────────────────────────────────
## Delimiter: ","
## dbl  (6): Open, High, Low, Close, Adj Close, Volume
## date (1): Date
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this
message.
```

Now lets have a look at the data for TESLA to see if there are any irregularities. We will look at both the head and the tail of the data.

```
head(tsla)

## # A tibble: 6 × 7
##   Date        Open  High   Low Close `Adj Close`    Volume
##   <date>     <dbl> <dbl> <dbl> <dbl>       <dbl>     <dbl>
## 1 2010-06-30  1.72  2.03 1.55   1.59        1.59 257806500
## 2 2010-07-01  1.67  1.73 1.35   1.46        1.46 123282000
## 3 2010-07-02  1.53  1.54 1.25   1.28        1.28  77097000
## 4 2010-07-06  1.33  1.33 1.06   1.07        1.07 103003500
## 5 2010-07-07  1.09  1.11 0.999  1.05        1.05 103825500
## 6 2010-07-08  1.08  1.17 1.04   1.16        1.16 115671000

tail(tsla)

## # A tibble: 6 × 7
##   Date        Open  High   Low Close `Adj Close`    Volume
##   <date>     <dbl> <dbl> <dbl> <dbl>       <dbl>     <dbl>
## 1 2023-08-15  239.  240.  233.  233.        233.  88197600
## 2 2023-08-16  228.  234.  225.  226.        226. 112484500
## 3 2023-08-17  226.  227.  219.  219.        219. 120718400
## 4 2023-08-18  214.  218.  212.  215.        215. 135813700
## 5 2023-08-21  222.  232.  221.  231.        231. 135702700
## 6 2023-08-22  240.  241.  230.  233.        233. 130442800
```

From the data above, it is as we expected it to be, also the data start from 2010 from the time tesla was listed in the stock exchange till the most recent time which is 23 of August 2023.

Below, we look at the most salient features of the data, the summary statistics for tesla's closing stock prices over the years.

```
summary(tsla$Close)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.053   9.957  16.692  65.815  54.086 409.970
```

For our analysis, we will be mostly interested in the closing prices of the stock and thus we will discard the other features such as the opening values, high point,the adjusted closing price and the volume.

From the onset, its quite clear that tesla stock prices have varied wildly over the years as the spread between the lowest price to the highest point is about 408.917 which is relatively large. This tell us that over the years, the stock price of tesla has tended to trend upwards. Also, the interquartile range, which measures the difference between the value in the 25 percentile of the data and the 75 percentile, it is also relatively very large at around 44.129, thus, even in the most dense part of the data we have a huge variation. These values all point out to the fact that the data is spread out significantly thus tesla stock prices have a relatively high variance.

Below, we look at the class to which the data is from, and the structure of the data.

```
class(tsla)

## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"

str(tsla)

## spc_tbl_ [3,309 × 7] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Date     : Date[1:3309], format: "2010-06-30" "2010-07-01" ...
##  $ Open     : num [1:3309] 1.72 1.67 1.53 1.33 1.09 ...
##  $ High     : num [1:3309] 2.03 1.73 1.54 1.33 1.11 ...
##  $ Low      : num [1:3309] 1.553 1.351 1.247 1.055 0.999 ...
##  $ Close    : num [1:3309] 1.59 1.46 1.28 1.07 1.05 ...
##  $ Adj Close: num [1:3309] 1.59 1.46 1.28 1.07 1.05 ...
##  $ Volume   : num [1:3309] 2.58e+08 1.23e+08 7.71e+07 1.03e+08 1.04e+08
...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Date = col_date(format = ""),
##   ..   Open = col_double(),
##   ..   High = col_double(),
##   ..   Low = col_double(),
##   ..   Close = col_double(),
##   ..   `Adj Close` = col_double(),
##   ..   Volume = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

We see tha the data is a "spec_tbl_df" and we see the all the features contained in this data and the various data types each of the features are.

Now, we will change it into a tibble dataframe which is nicer to work with. Then later we will change the data into a tsibble data type which will allow us to apply the tools contained in the fpp3 package.

```
#turn the data from the default format into a tibble format
tesla_tbl <- tsla %>%
  fortify.zoo() %>%
  as_tibble()
#Check if our goal have been achieved.
head(tesla_tbl)

## # A tibble: 6 × 8
##   Index Date        Open  High   Low Close `Adj Close`    Volume
##   <int> <date>     <dbl> <dbl> <dbl> <dbl>       <dbl>     <dbl>
## 1     1 2010-06-30  1.72  2.03 1.55   1.59        1.59 257806500
## 2     2 2010-07-01  1.67  1.73 1.35   1.46        1.46 123282000
## 3     3 2010-07-02  1.53  1.54 1.25   1.28        1.28  77097000
## 4     4 2010-07-06  1.33  1.33 1.06   1.07        1.07 103003500
## 5     5 2010-07-07  1.09  1.11 0.999  1.05        1.05 103825500
## 6     6 2010-07-08  1.08  1.17 1.04   1.16        1.16 115671000

str(tesla_tbl)

## tibble [3,309 × 8] (S3: tbl_df/tbl/data.frame)
##  $ Index    : int [1:3309] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Date     : Date[1:3309], format: "2010-06-30" "2010-07-01" ...
##  $ Open     : num [1:3309] 1.72 1.67 1.53 1.33 1.09 ...
##  $ High     : num [1:3309] 2.03 1.73 1.54 1.33 1.11 ...
##  $ Low      : num [1:3309] 1.553 1.351 1.247 1.055 0.999 ...
##  $ Close    : num [1:3309] 1.59 1.46 1.28 1.07 1.05 ...
##  $ Adj Close: num [1:3309] 1.59 1.46 1.28 1.07 1.05 ...
##  $ Volume   : num [1:3309] 2.58e+08 1.23e+08 7.71e+07 1.03e+08 1.04e+08
...

view(tesla_tbl)
```

Now the data has been converted into a tibble, which will allow us to apply the various augmentations that will allow us to carry out our analysis.

Getting the data into weekly format

To get the data into a weekly format, we will make use of the floor_date function to floor the data to weekly intervals, meaning that the dates after a certain date will be floored to the date of the beginning of that particular week. Thus every five days after the first day of the week will be floored to the beginning day of that week.

```
#floor the data into the preceding first day of that particular week
tesla_tbl$date <- floor_date(tesla_tbl$Date, "week")
head(tesla_tbl)

## # A tibble: 6 × 9
##   Index Date        Open  High   Low Close `Adj Close`    Volume date
##   <int> <date>     <dbl> <dbl> <dbl> <dbl>       <dbl>     <dbl> <date>
## 1     1 2010-06-30  1.72  2.03 1.55   1.59        1.59 257806500 2010-06-
27
## 2     2 2010-07-01  1.67  1.73 1.35   1.46        1.46 123282000 2010-06-
27
## 3     3 2010-07-02  1.53  1.54 1.25   1.28        1.28  77097000 2010-06-
27
## 4     4 2010-07-06  1.33  1.33 1.06   1.07        1.07 103003500 2010-07-
04
## 5     5 2010-07-07  1.09  1.11 0.999  1.05        1.05 103825500 2010-07-
04
## 6     6 2010-07-08  1.08  1.17 1.04   1.16        1.16 115671000 2010-07-
04
```

From the plot data above, it seems we have achieved that end. This analysis will then move towards another aspect of our analysis, where we average the stock prices for each day onto weekly basis.

Now, lets calculate the mean values for each of our variables, averaging each day over a week.

```
tesla_tbl2 <- tesla_tbl %>%
  group_by(date) %>%
  summarise(TSLA.open.avg = mean(Open), TSLA.high.avg = mean(High),
TSLA.low.avg = mean(Low), TSLA.close.avg = mean(Close), TSLA.adjclose.avg =
mean(`Adj Close`), TSLA.volume.avg= mean(Volume))

view(tesla_tbl2)
```

Now that we have our data floored to weekly format, and we have also averaged the data into weekly data, this analysis will turn the tibble into a tsibble format, which is much more convenient for dealing with time series data using the fpp3 package.

```
tesla_tsibble  <- tesla_tbl2 %>%
  mutate(Week = yearweek(date))%>%
  as_tsibble(index = Week) %>%
  subset(select = -date) %>%
  relocate(Week)

head(tesla_tsibble)

## # A tsibble: 6 x 7 [1W]
##       Week TSLA.open.avg TSLA.high.avg TSLA.low.avg TSLA.close.avg
##     <week>         <dbl>         <dbl>        <dbl>          <dbl>
## 1 2010 W25          1.64          1.77         1.38           1.44
## 2 2010 W26          1.17          1.20         1.05           1.11
## 3 2010 W27          1.25          1.33         1.21           1.27
## 4 2010 W28          1.41          1.44         1.36           1.40
## 5 2010 W29          1.39          1.40         1.34           1.37
## 6 2010 W30          1.40          1.42         1.35           1.39
## # ℹ 2 more variables: TSLA.adjclose.avg <dbl>, TSLA.volume.avg <dbl>
```

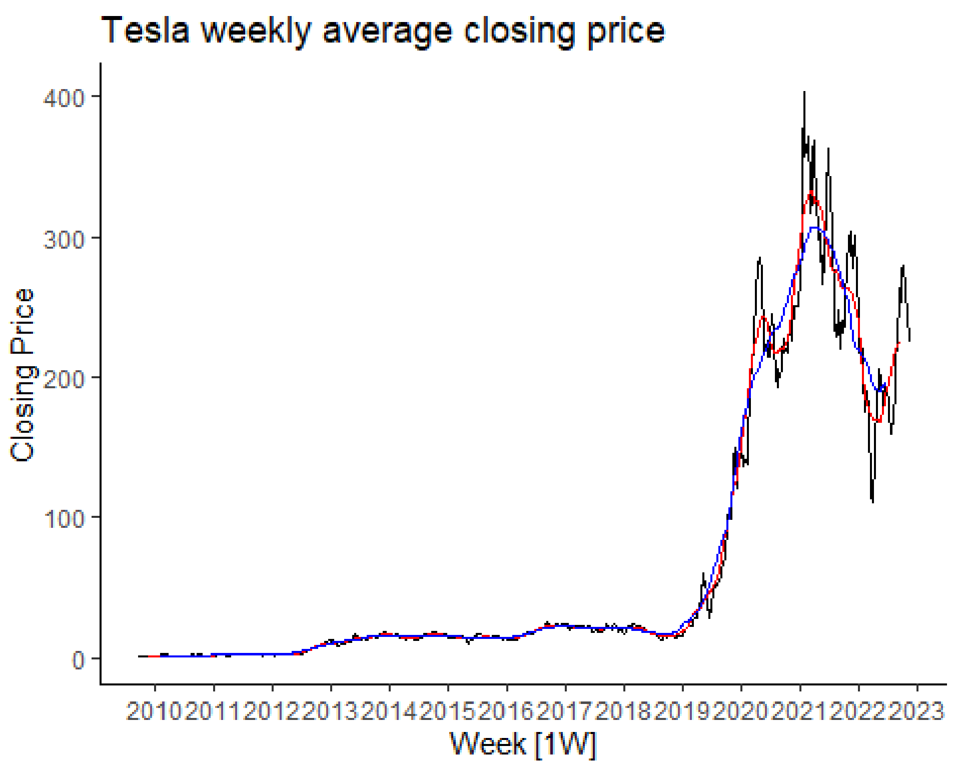
Our data is in terms of a tsibble and we have it in weekly intervals. Now, the next phase of our analysis is to look at the prominent patterns observed in the data. We will do this by plotting the data.

Exploratory Data Analysis

Lets plot our data and see how it looks like and see if there are any patterns that we can discern and thus we can use to build a model that is appropriate to the data.

First, we fit the data, then on top of the data, plot a smoothed version of the time series data to follow any trends or seasonality in the data without the random flactuations which cannot be predetermined.

```
tesla_tsibble$TSLA.close.avg.sma <- rollmean(tesla_tsibble$TSLA.close.avg,
20, fill = NA, align = "center")
tesla_tsibble$TSLA.close.avg.sma2<- rollmean(tesla_tsibble$TSLA.close.avg,
40, fill = NA, align = "center")


tesla_tsibble%>%autoplot(TSLA.close.avg) +
  labs(y = "Closing Price",
       title = "Tesla weekly average closing price") +
  geom_line(aes(y = TSLA.close.avg.sma), col = "red") +
    geom_line(aes(y = TSLA.close.avg.sma2), col = "blue") +

  theme_classic() +
scale_x_yearweek(date_labels = "%Y", date_breaks = "12 months")

## Warning: Removed 19 rows containing missing values (`geom_line()`).

## Warning: Removed 39 rows containing missing values (`geom_line()`).
```



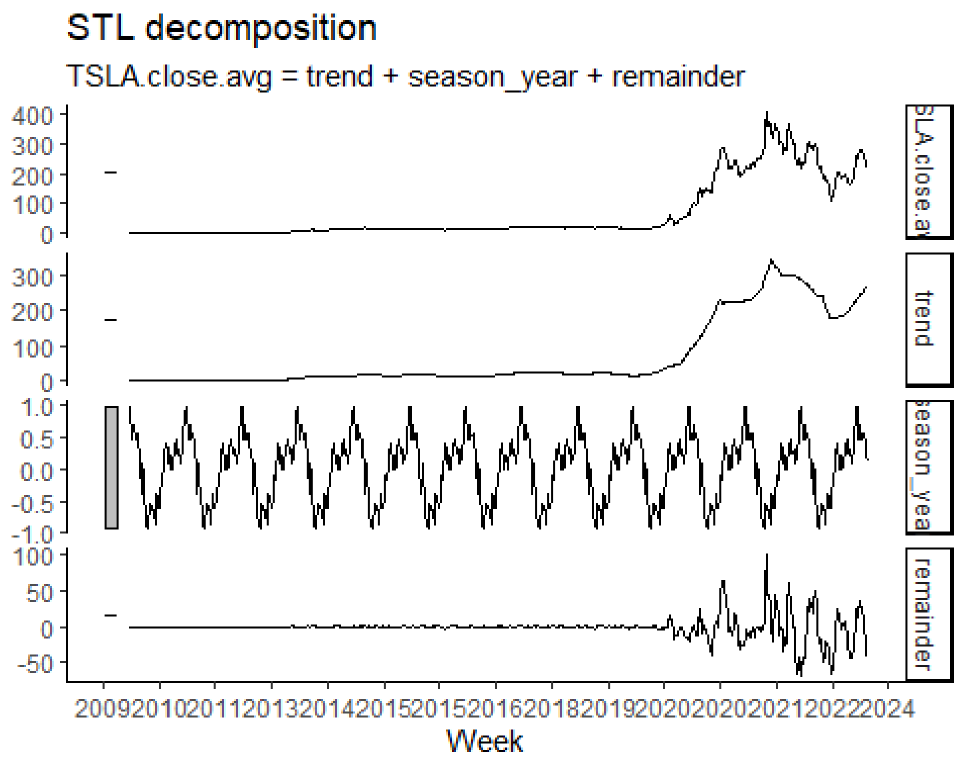
From the plot above, the spread in the data from around 2010 till about 2019 is pretty stable and no strong trend can be seen in the data. However, this can be misleading since there's a huge jump in the model from 2019 which is relatively higher sloped compared to the rest of the data. Thus, this jump shields the other jumps that could have occurred prior to this point. For instance, there's a change in the trend of the data between 2012 and 2013 which is not very clear. From the plot, there's some seasonality, but its not very pronounced and thus is not very visible.

All these subtle patterns can be isolated and be clear by running an STL decomposition of the data into its components.

Below, we run an STL decomposition to validate the patterns in the data and see the various patterns in the data isolated from the onset/listing of TESLA stock prices.

As alluded, its clear that over the long term, there is a clear upward trend in the data, though in the initial years it seems to be flat, this could be due to the much more steeper slope in the later years which shield the true steepness in the initial years. Thus, zooming in into the earlier years might be necessary to get a picture of the patterns in the initial years of the time series. However, the seasonal pattern is also present, and seems to be repeating itself on yearly intervals, with two peaks each year, thus its likely even before zooming in that it'll be there.

```
tesla_tsibble %>%
  model(
    STL(TSLA.close.avg ~ trend(window = 21)+
        season(window = "periodic"),
    robust = TRUE)
  ) %>%
  components() %>%
  autoplot() +
  theme_classic() +
  scale_x_yearweek(date_labels = "%Y", date_breaks = "1 year")
```



Zooming in on the data to discern any patterns in short intervals of time before the big surge in 2019.

First, we need to create a column, called the *year* column, which takes the year part of the data string, then use it to select the years of interest.

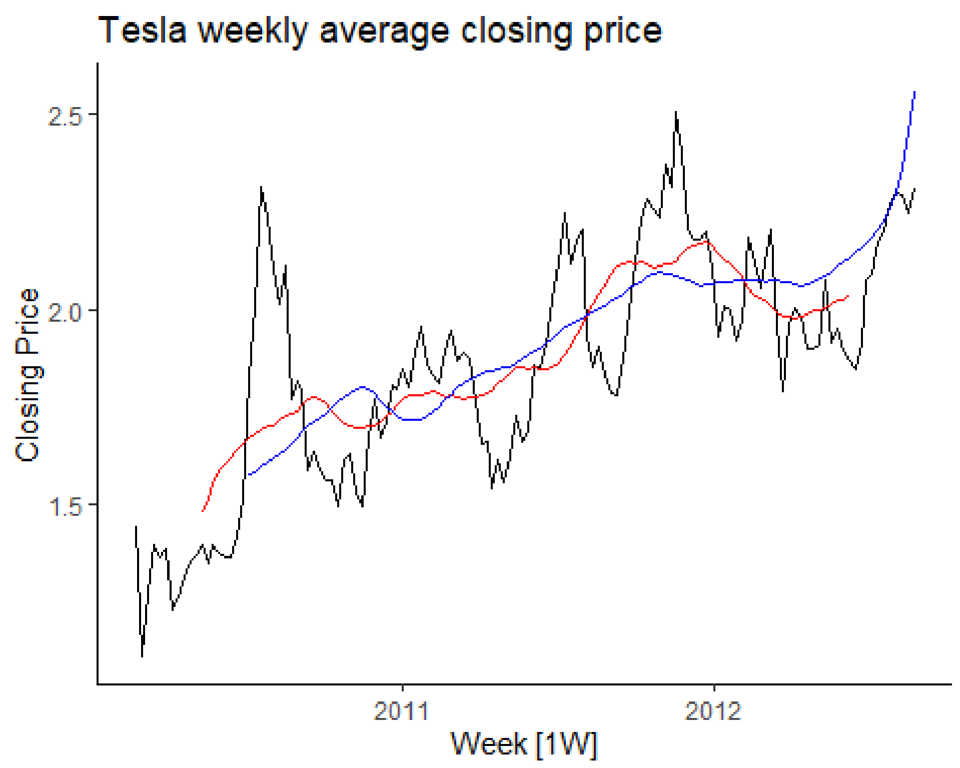
```
tesla_tsibble$year <- as.numeric(substr(tesla_tsibble$Week,1,4))
class(tesla_tsibble$year)

## [1] "numeric"

head(tesla_tsibble)

## # A tsibble: 6 x 10 [1W]
##        Week TSLA.open.avg TSLA.high.avg TSLA.low.avg TSLA.close.avg
##      <week>         <dbl>         <dbl>        <dbl>          <dbl>
## 1 2010 W25           1.64          1.77         1.38           1.44
## 2 2010 W26           1.17          1.20         1.05           1.11
## 3 2010 W27           1.25          1.33         1.21           1.27
## 4 2010 W28           1.41          1.44         1.36           1.40
## 5 2010 W29           1.39          1.40         1.34           1.37
## 6 2010 W30           1.40          1.42         1.35            1.39
## # ℹ 5 more variables: TSLA.adjclose.avg <dbl>, TSLA.volume.avg <dbl>,
## #   TSLA.close.avg.sma <dbl>, TSLA.close.avg.sma2 <dbl>, year <dbl>
```

Nice, it seems we have successfully created the column of interest, and thus we can use then to subset the data for the years we are interested to visualize and decompose.

First, lets see the data from the onset that Tesla was listed on the stock exchange, till about 2012 when the we first see a small jump in the data.

```r
tesla.tsibble.2012 <- tesla_tsibble[which(tesla_tsibble$year <= 2012),]



tesla.tsibble.2012$TSLA.close.avg.sma <-
rollmean(tesla.tsibble.2012$TSLA.close.avg, 23, fill = NA, align = "center")
tesla_tsibble$TSLA.close.avg.sma2 <- rollmean(tesla_tsibble$TSLA.close.avg,
5, fill = NA, align = "center")



tesla.tsibble.2012%>%autoplot(TSLA.close.avg) +
  labs(y = "Closing Price",
       title = "Tesla weekly average closing price") +
  geom_line(aes(y = TSLA.close.avg.sma), col = "red") +
        geom_line(aes(y = TSLA.close.avg.sma2), col = "blue") +
  theme_classic() +
scale_x_yearweek(date_labels = "%Y", date_breaks = "12 months")
```

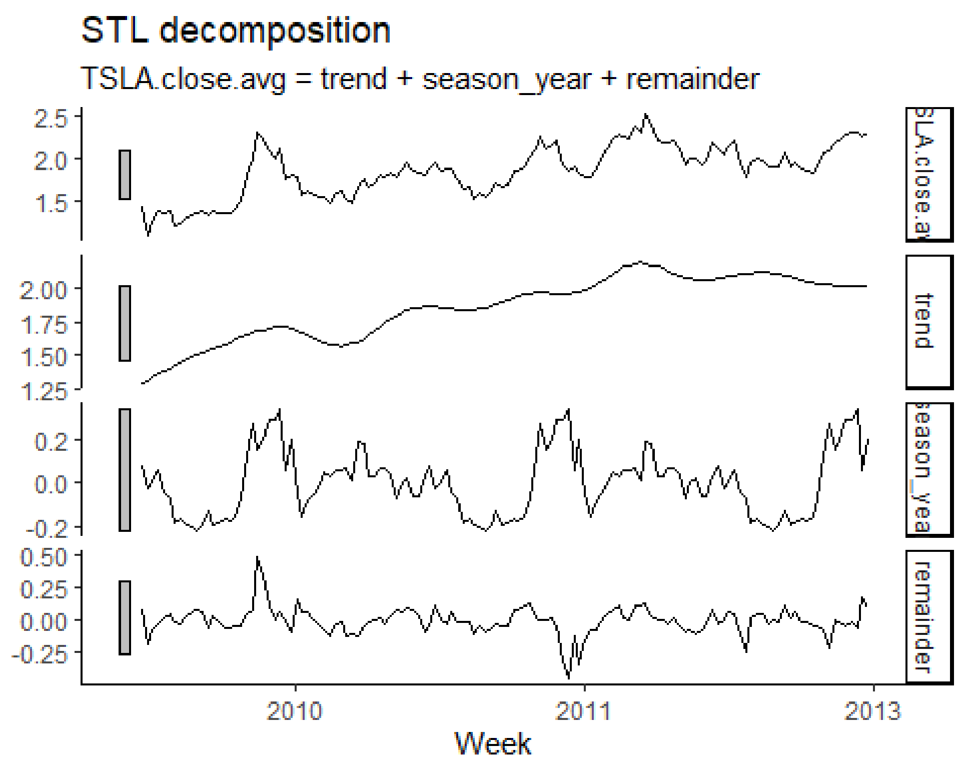
```
## Warning: Removed 22 rows containing missing values (`geom_line()`).

## Warning: Removed 19 rows containing missing values (`geom_line()`).
```



Its clear from the plot above that there is a an upward trend in the data, which is also shown by the red smoothed trendline. Thus, from 2010 till around 2012, the data is upward trending, though this pattern was shielded in the plot over the entire time series. Also, there are some instances of seasonality in the data as shown by the red smoothing line which shows a cyclical repetitive patterns.

Below, we run a STL decomposition of the data from 2010 till 2012 and see the various patterns individually. We have set the toggle inside of the decomposition to be robust to outliers, meaning that it will be able to deal with outliers and they will not deviate the data.

```
tesla.tsibble.2012 %>%
  model(
    STL(TSLA.close.avg ~ trend(window = 21)+
        season(window = "periodic"),
    robust = TRUE)
  ) %>%
  components() %>%
  autoplot() +
  theme_classic() +
  scale_x_yearweek(date_labels = "%Y", date_breaks = "1 year")
```

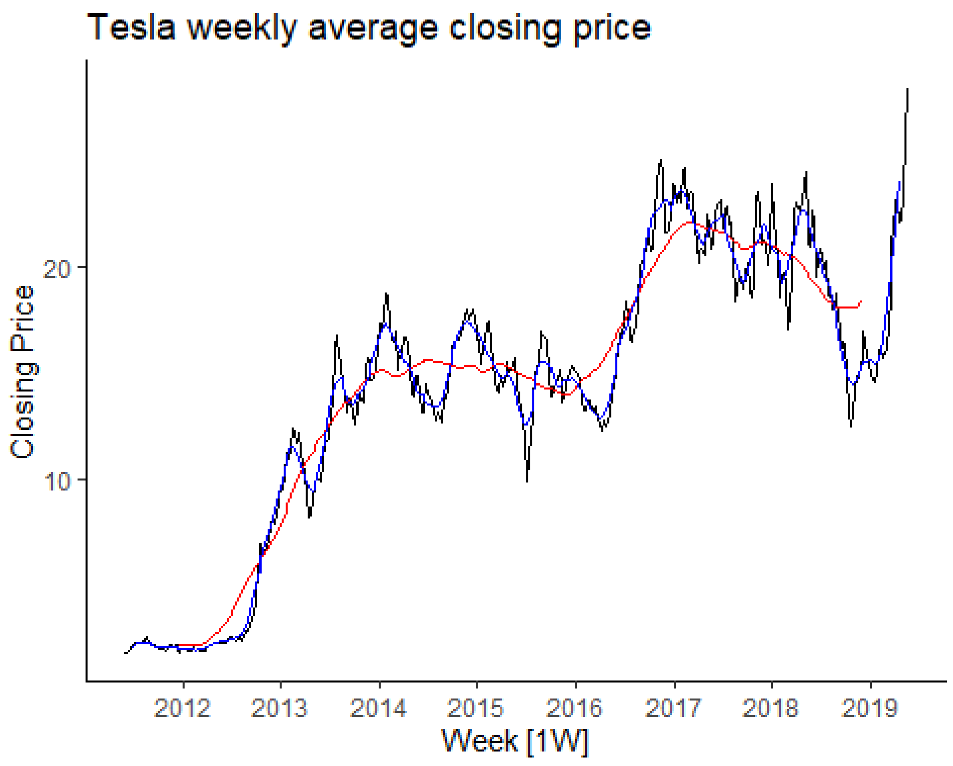


As we have outlined above, there is indeed a clear upward trend in the data. Also, there is some seasonality in the data, though the seasonal nature of the data is not very clear and discernable. The random fluctuation does not really show any pattern and thus the trend and the seasonality component have squeezed out all the predictability.

2012 TO 2019

Another period we want to isolate in the data is the period from 2012 till 2019 as this period was directly before the huge jump seen in 2019 in the data.

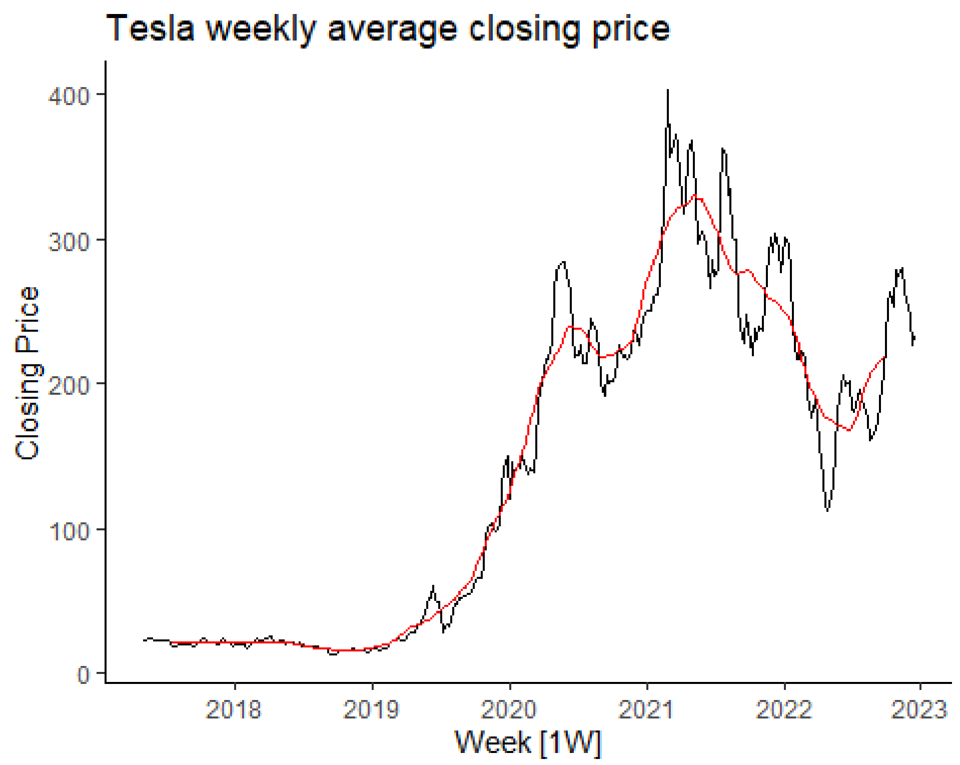
```
tesla.tsibble.2019 <- tesla_tsibble[which(tesla_tsibble$year <= 2019),]
tesla.tsibble.2019 <- tesla_tsibble[which(tesla.tsibble.2019$year >= 2012),]


tesla.tsibble.2019$TSLA.close.avg.sma <-
rollmean(tesla.tsibble.2019$TSLA.close.avg, 53, fill = NA, align = "center")
tesla.tsibble.2019$TSLA.close.avg.sma2 <-
rollmean(tesla.tsibble.2019$TSLA.close.avg, 10, fill = NA, align = "center")


tesla.tsibble.2019%>%autoplot(TSLA.close.avg) +
  labs(y = "Closing Price",
       title = "Tesla weekly average closing price") +
  geom_line(aes(y = TSLA.close.avg.sma), col = "red") +
  geom_line(aes(y = TSLA.close.avg.sma2), col = "blue") +
  theme_classic() +
scale_x_yearweek(date_labels = "%Y", date_breaks = "12 months")

## Warning: Removed 52 rows containing missing values (`geom_line()`).

## Warning: Removed 9 rows containing missing values (`geom_line()`).
```



This pattern again resembles the pattern we saw above. There's a clear trend in the data and the trend is upwardly positive which is also shown by the red smoothed line which validates that there's an upward trend, though around 2014 till around 2016 it sort of stalled, but overall the data is still trending upwards. Also, from the blue curve, we can still see the seasonal pattern, through the repetitive patterns. Thus, overall the data has from this plot retained the general patterns from the earlier years, though there was a huge jump from around 2013 from the relatively smooth data prior from around 2010.

```
tesla.tsibble.2023 <- tesla_tsibble[which(tesla_tsibble$year > 2017),]

tesla.tsibble.2023$TSLA.close.avg.sma <-
rollmean(tesla.tsibble.2023$TSLA.close.avg, 23, fill = NA, align = "center")


tesla.tsibble.2023%>%autoplot(TSLA.close.avg) +
  labs(y = "Closing Price",
       title = "Tesla weekly average closing price") +
  geom_line(aes(y = TSLA.close.avg.sma), col = "red") +
  theme_classic() +
scale_x_yearweek(date_labels = "%Y", date_breaks = "12 months")
```

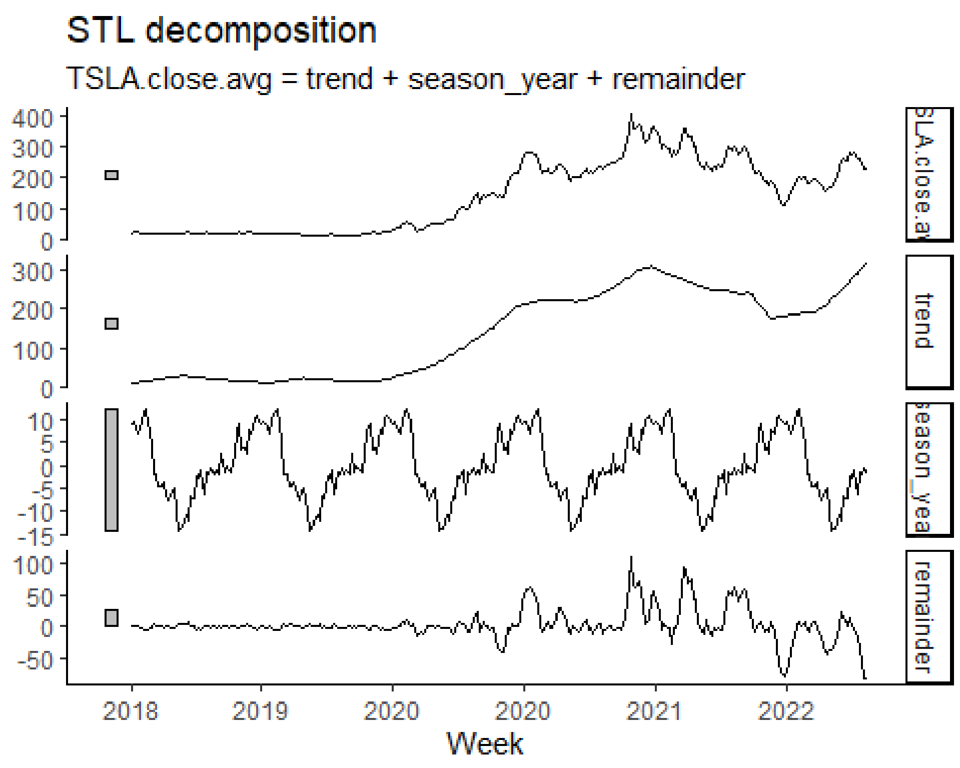
```
## Warning: Removed 22 rows containing missing values (`geom_line()`).
```



Again, from the plot above, it seems the closing stock prices of tesla keep on having this pattern, where the data is relatively calm, then boom there's a surge which rallies on and on then becomes calm once more.

Even from the plot below, the STL decomposition, the pattern has repeated once more, being relatively calm then boom there's a surge.

```
tesla.tsibble.2023 %>%
  model(
    STL(TSLA.close.avg ~ trend(window = 21)+
        season(window = "periodic"),
    robust = TRUE)
  ) %>%
  components() %>%
  autoplot() +
  theme_classic() +
  scale_x_yearweek(date_labels = "%Y", date_breaks = "1 year")
```



Now that we have identified the repetitive and predictable patterns in the data, we can now move on and use them to fit models which we can use to forecast future values.

Model fitting with the various benchmark methods

First, this analysis will attempt to find a benchmark model, which we will compare the performance of the more advanced models against with. Since we have seen from the data that there is some seasonality in the data, we will incoporate a seasonal component, by using a naive seasonality model which simply assumes that the seasonality in the past year will continue going forward.

Furthermore, owing to the fact that this is a stock data, the general assumption is that stock data follow a random walk or naive and that the future stock value are a just the current value plus a random element whose value can only be pre-determined with a random chance.

Further, we also saw that the data has a trend component, thus in our analysis, we will also incoporate the drift component, which fits a line that extrapolates from the beginning of the data, to the end of the data and to the future. Thus, drift is the average seen over the historical data.

Training the benchmark model

Lets get the data we will be training on by subsetting on the whole dataset.

```
#Get the training data by subsetting all the data pre-2022
tesla_tsibble_2022 <- tesla_tsibble[which(tesla_tsibble$year < 2022),]
```

```
tail(tesla_tsibble_2022)

## # A tsibble: 6 x 10 [1W]
##       Week TSLA.open.avg TSLA.high.avg TSLA.low.avg TSLA.close.avg
##     <week>         <dbl>         <dbl>        <dbl>          <dbl>
## 1 2021 W47          373.          379.         358.           365.
## 2 2021 W48          345.          349.         333.           343.
## 3 2021 W49          321.          327.         309.           317.
## 4 2021 W50          317.          329.         311.           326.
## 5 2021 W51          361.          368.         355.           360.
## 6 2021 W52          376.          383.         358.           369.
## # ℹ 5 more variables: TSLA.adjclose.avg <dbl>, TSLA.volume.avg <dbl>,
## #   TSLA.close.avg.sma <dbl>, TSLA.close.avg.sma2 <dbl>, year <dbl>
```

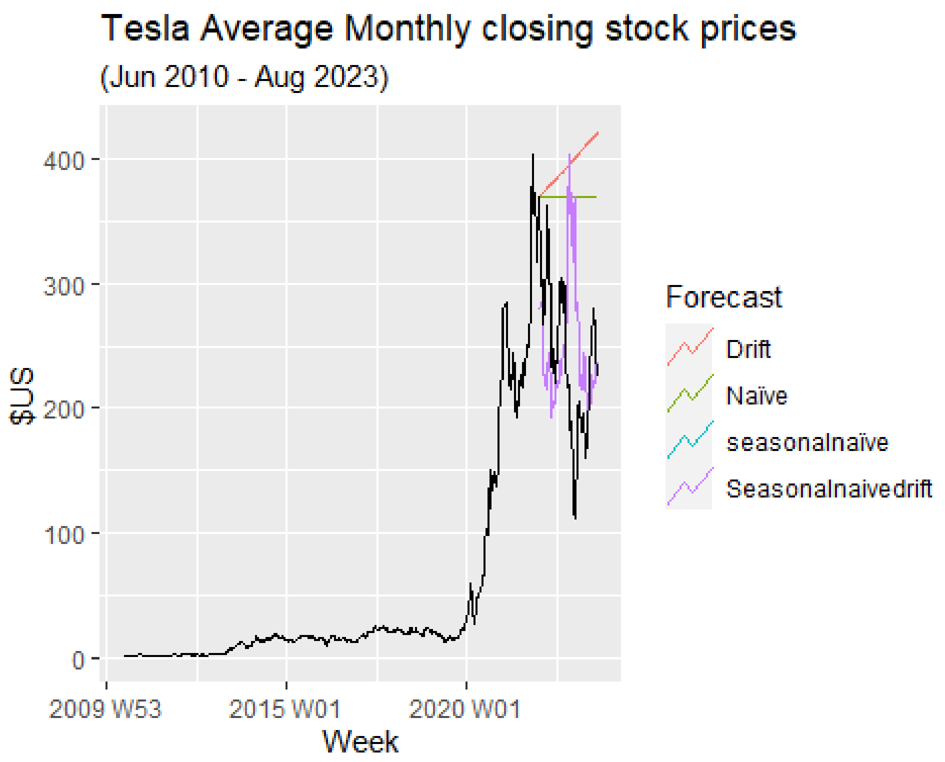
Now lets fit the benchmark models and see which one performs best, we will fit a couple of benchmark model to ensure that we don't leave out anything or make any assumptions though we are guided by the patterns observed in the data.

We fit the naive model, we just assume the forecasted values are equivalent to the last observed value. Then, we fit the drift model which just fit the historical average trend and assume that is what will continue going forward. Then, we fit the seasonal naive and seasonal naive model with drift. These two models are the one's whose properties are consistent with the patterns observed in the data and thus we expect them to perform better.

```
tesla_fit <- tesla_tsibble_2022 %>%
  model(
    Naïve = NAIVE(TSLA.close.avg),
     Drift = NAIVE(TSLA.close.avg ~ drift()),
     seasonalnaïve = SNAIVE(TSLA.close.avg),
    Seasonalnaivedrift =SNAIVE(TSLA.close.avg, drift = TRUE)
  )
```

Lets produce some forecasts for 2022 and 2023 which were not included in the training data to assess visually the perfomance of these benchmark models.

```
tesla_tsb_2023 <-  tesla_tsibble[which(tesla_tsibble$year >= 2022),]
#Lets run the forecast for 2023

tesla_fc <- tesla_fit %>%
  forecast(new_data = tesla_tsb_2023)

#Lets plot the forecasts
tesla_fc %>%
  autoplot(tesla_tsibble, level = NULL) +
  autolayer(tesla_tsb_2023, TSLA.close.avg, colour = "black") +
  labs(y = "$US",
       title = "Tesla Average Monthly closing stock prices",
       subtitle = "(Jun 2010 - Aug 2023)") +
  guides(colour = guide_legend(title = "Forecast"))
```



It seems like the seasonal naive model with drift is the most appropriate model, as the other other models seem to over the long term over estimatie the future fluctuations in the future closing average stock prices. Now, lets make use of some statistical tests to robustly assess which of these simple benchmark models will perform better at forecasting the future fluctuations in the weekly average closing prices in the tesla stock prices.

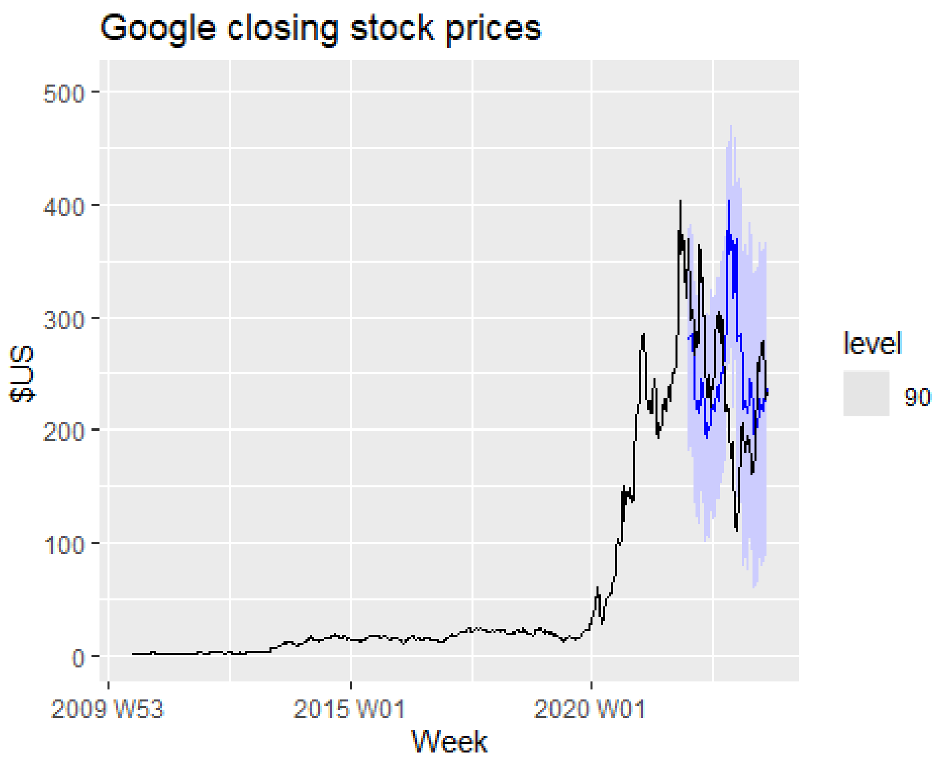
```
accuracy(tesla_fc, tesla_tsibble)

## # A tibble: 4 × 10
##   .model             .type     ME  RMSE   MAE   MPE  MAPE  MASE RMSSE
ACF1
##   <chr>              <chr>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
<dbl>
## 1 Drift              Test  -157.  171.  157.  -79.1  79.1  6.05  2.85
0.940
## 2 Naïve              Test  -131.  143.  131.  -66.3  66.3  5.04  2.39
0.932
## 3 Seasonalnaivedrift Test   -12.5  88.8  68.1 -15.6  34.7  2.62  1.48
0.954
## 4 seasonalnaïve      Test   -12.5  88.8  68.1 -15.6  34.7  2.62  1.48
0.954
```

From the two scale dependent forecast error measurements, the RMSE (Root Mean Squared Error) and MAE (Mean Absolute Errors), it seems that they both are in agreement that the model that minimizes the forecast errors is the Seasonal naive method with drift as it has the lowest error among all the terms. Though the different time series we are measuring have the same units and from the same data, thus there is no need to make use of scale independent forecast accuracy measures, it seems again the MASE(Mean Absolute Scaled Error) and the RMSSE(Root Mean Squared Scaled Error), both have their forecast minimized when we use the Seasonal naive method with drift.

Thus, across all the major accuracy measures, it seems that the seasonally naive method is the best performing one and thus will be used as the benchmark method when we experiment with much more Advanced forecasting methods.

Now, below, we see that the forecasts using the seasonal naive method with drift seems to be following the actual data very well, meaning the distributional form of the our model seems to be resembling the actual distribution of the data. This is important to assess if the actual values tend to fall outside the prediction intervals how many times or withing our prediction intervals.

```r
tesla_fc |>
  filter(.model == "Seasonalnaivedrift") |>
  autoplot(bind_rows(tesla_tsibble_2022, tesla_tsb_2023), level=90)+
  labs(y = "$US",
       title = "Google closing stock prices")
```



Though from the plot above it seems like the actual points in the test are within the prediction intervals most of the time, we will make use of the Continuous Ranked Probabilities Scores to get a sense of the times the score fall within the prediction intervals and behaves well. The CRPS is a average of the Winker score over the entire test set. The winker score is simply a length of the prediction interval associated with a particular forecast point plus a penalty value if such a point fall outside the prediction intervals, with low score associated with narrower intervals. For times the actual points falls outside the intervals, we apply a penalty, which increases the scores each model receives if its forecasts falls outside many times.

```
tesla_fc %>%
  accuracy(tesla_tsibble, list(crps= CRPS))
```

```
## # A tibble: 4 × 3
##   .model             .type  crps
##   <chr>              <chr> <dbl>
## 1 Drift              Test  135.
## 2 Naïve              Test  110.
## 3 Seasonalnaivedrift Test   50.3
## 4 seasonalnaïve      Test   50.3
```

Again, the CRPS score points out that the distributional form of the seasonal and seasonal naive with drift methods to be lowest, meaning they are the least likely to have on average over the entire prediction space have points that fall outside the prediction intervals, thus it has the most appropriate distributional form from the actual data.

Now, we will apply another method, that is a bit more complicated than the simple benchmark model we used, we will apply the regression model.

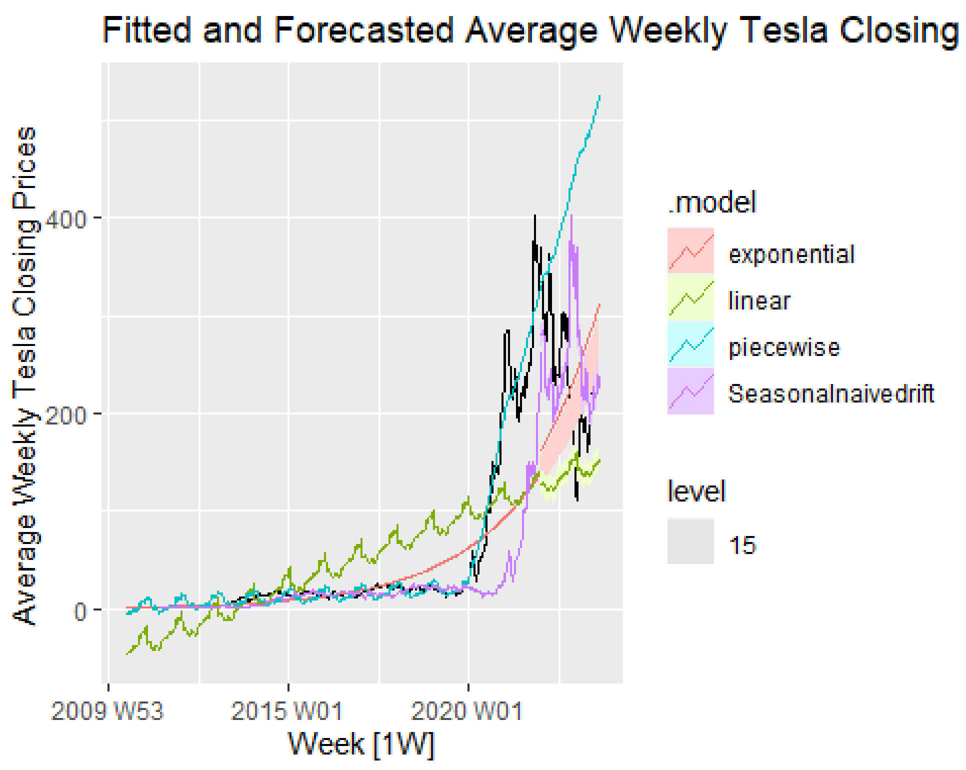
Regression Model for TESLA stock..

A linear model simply tries to fit a line of best fit to the data, with the explanatory variables being linearly mapped to the target variable, which is the future values of the closing stock prices for tesla stocks. The model we will use will try to find a bunch of parameters such that the errors between the actual values and the fitted values is the least or is minimised.

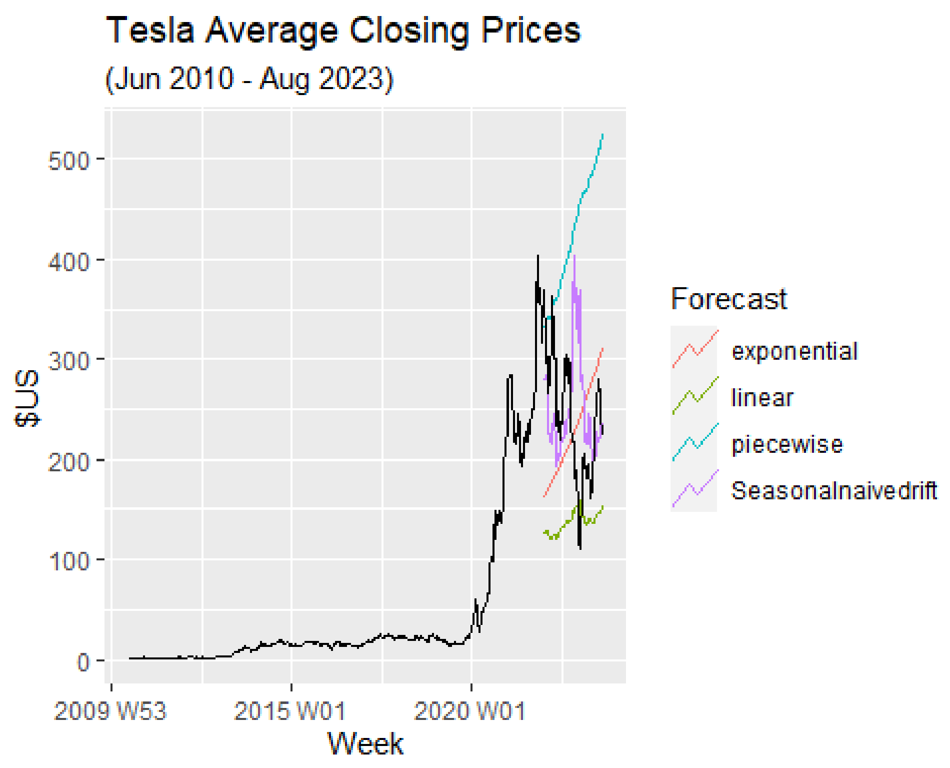
Since the model has a trend and a seasonal component, we will make use of these two features as the explanatory variables in our model, and use some ad hoc analysis to see if our model meets the assumptions we make for our model.

We fit several variations of the linear regression model so that we can get a sense of which of the models are the most appropriate and give robustly better predictions.

```
knots_dates <- yearweek (c("2012 W21", "2020 W01","2021 W05"))
knots_indices <- as.numeric (knots_dates)

fit_tesla <- tesla_tsibble_2022 %>%
  model(
    Seasonalnaivedrift =SNAIVE(TSLA.close.avg, drift = TRUE),
    linear=  TSLM(TSLA.close.avg ~ trend() + season()),
    exponential = TSLM(log(TSLA.close.avg) ~ trend()),
    piecewise = TSLM(TSLA.close.avg ~ trend(knots = knots_indices) +season()
)
)



tesla_fc_regr <- fit_tesla %>%
  forecast(new_data = tesla_tsb_2023)

tesla_tsibble %>%
  autoplot(TSLA.close.avg) +
  geom_line(data = fitted(fit_tesla),
            aes(y = .fitted, colour= .model)) +
    autolayer(tesla_fc_regr,level = 15) +

    labs(y = "Average Weekly Tesla Closing Prices",
       title = "Fitted and Forecasted Average Weekly Tesla Closing Prices
across different models")

## Warning: Removed 52 rows containing missing values (`geom_line()`).
```



```
tesla_fc_regr %>%
  autoplot(tesla_tsibble, level = NULL) +
  autolayer(tesla_tsb_2023, TSLA.close.avg, colour = "black") +
  labs(y = "$US",
       title = "Tesla Average Closing Prices",
       subtitle = "(Jun 2010 - Aug 2023)") +
  guides(colour = guide_legend(title = "Forecast"))
```



From the plot above, we have plotted a couple of models, of which three are a variant of the linear regression model but with non-linear transformations. The first model we fit to the data is the linear regression model, with two variables, namely a trend and a seasonal component. This model does a poor job at really following the data as in the beginning it underfit, then overfit and then underfit again, and thus its distributional form is not appropriate for the data. The next model is the logged model, which is an exponential transformation of the linear regression model, to allow for a non-linear dynamic into the model. As a result, the model follows the data very well initially. Then, however, it takes some time to pick up the surge around 2019, and when it does it assumes that it will continue onwards and thus does not damp down as the surge on the on the original data dies down. Another model we introduce is the linear regression model with knots at all the turning points. This model fit the data relatively well, is able to pick up the trend and at the changing knots, it was also able to pick up the patterns in the data. However, the unfortunate thing with this model, the actual data is the forecast period changed dramatically, and started trending downwards all of a sudden, of which the model could have never picked it up as this downward trend happened only in the test data. Again, our benchmark model the seasonal naive model with drift did reasonably well again, as it was not overconfident as the piescewise function was in its forecast, it was relatively onnocous and conservative in its forecast. The seasonal naive method is slow to increase in confidence which helps it not to be overconfident in its forecasts.

```
accuracy(tesla_fc_regr, tesla_tsibble)

## # A tibble: 4 × 10
##   .model                .type      ME   RMSE    MAE     MPE   MAPE   MASE  RMSSE
```

```
ACF1
##   <chr>              <chr>   <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>
<dbl>
## 1 Seasonalnaivedrift Test   -12.5   88.8  68.1 -15.6   34.7  2.62  1.48
0.954
## 2 exponential        Test     8.00  89.5  78.5  -6.00  35.1  3.02  1.49
0.950
## 3 linear             Test   100.   120.  104.   37.3   40.1  4.00  1.99
0.940
## 4 piecewise          Test  -187.   214.  188.  -95.4   95.7  7.25  3.56
0.956
```

Again, from all the major forecast error measures, the seasonal naive method with drift wins as it has the lowesr forecast across all the measures, both scale and scale independent measures. However, the exponential measure was pretty close, however its assumption that the data will continue trending was its downfall.

We see from the models above that the see that the logged model or the exponential model is the best perfoming out of the models we have above which are the variations of the regression model. Thus to assess the distributional forecasts accuracy relative to the actual test data, we use the Continuous Ranked Probability to penalise for inaccurate prediction intervals.

```
tesla_fc_regr |>
  accuracy(tesla_tsibble, list(crps = CRPS))

## # A tibble: 4 × 3
##   .model             .type  crps
##   <chr>              <chr> <dbl>
## 1 Seasonalnaivedrift Test   50.3
## 2 exponential        Test   54.7
## 3 linear             Test   77.0
## 4 piecewise          Test  178.
```

As seen in the plots with the forecasts, the seasonal naive model with drift has the lowest score and thus has the better distributional forecasts than the other models we built.

Exponential Smoothing Model..

Next we fit an exponential model to the data as means for forecasting. An exponential model is a model that is somewhat a combination of the mean forecasting and naive forecasting. This model will combine the best of these two methods. For example, the most recent value will have the most influence on the forecasted value into the future. However, this model will incoporate the mean forecasting method by weighting the past values, i,e averaging past values. However, the past values will not be equally weighted, with the most recent values getting the most weight compared to the older values.

Since our model has a trend component, or a slope, we will also incoporate this into our model. The trend, or slope is simply the rate of change from one point to the next. Again, the model will smooth the trend component by systematically weighting the past trend components, with the most recent components given the most weight.

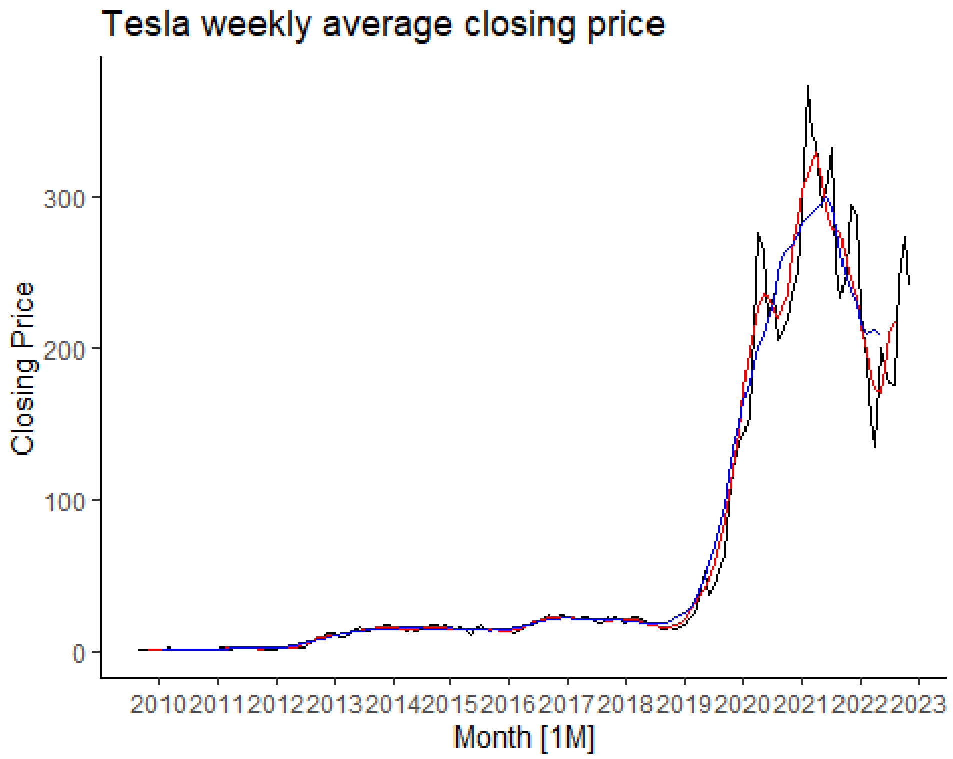
Note, this data also has a seasonal component, and thus a seasonal component will also be included, and we will also use exponentially smoothed past seasonal values to estimate the future seasonal portions. We expect that the model that will properly capture the seasonality is the multiplicative form of the seasonality as we expect that the seasonality in the data will change with the time or as the data increases as we have heteroscedasticity.

All of this will then be added together to find the forecast equation, to forecast the future values.

Turning the data into monthly intervals

The package we are using does not allow when we are fitting either the ARIMA or the exponential model to have data that has a seasonal component larger than 12 months, thus we will need to floor the data to month instead of a weekly basis.

```
#Floor the data to monthly and store this floored date to date2
tesla_tbl$date2 <- floor_date(tesla_tbl$Date, "month")
view(tesla_tbl)

#Average the data on the monthly basis after grouping the data on date2
tesla_tbl2 <- tesla_tbl %>%
  group_by(date2) %>%
  summarise( TSLA.close.avg = mean(Close))

view(tesla_tbl2)

#Now turn the data into a tsibble which is easier to work with.
tesla_tsibble2  <- tesla_tbl2 %>%
  mutate(Month = yearmonth(date2))%>%
  as_tsibble(index = Month) %>%
  subset(select = -date2) %>%
  relocate(Month)

head(tesla_tsibble2)

## # A tsibble: 6 x 2 [1M]
##      Month TSLA.close.avg
##      <mth>          <dbl>
## 1 2010 Jun           1.59
## 2 2010 Jul           1.30
## 3 2010 Aug           1.30
## 4 2010 Sep           1.38
## 5 2010 Oct           1.38
## 6 2010 Nov           1.94

tesla_tsibble2$TSLA.close.avg.sma <- rollmean(tesla_tsibble2$TSLA.close.avg,
6, fill = NA, align = "center")
tesla_tsibble2$TSLA.close.avg.sma2<- rollmean(tesla_tsibble2$TSLA.close.avg,
12, fill = NA, align = "center")

tesla_tsibble2%>%autoplot(TSLA.close.avg) +
  labs(y = "Closing Price",
       title = "Tesla weekly average closing price") +
  geom_line(aes(y = TSLA.close.avg.sma), col = "red") +

    geom_line(aes(y = TSLA.close.avg.sma2), col = "blue") +

  theme_classic() +
scale_x_yearmonth(date_labels = "%Y", date_breaks = "12 months")

## Warning: Removed 5 rows containing missing values (`geom_line()`).

## Warning: Removed 11 rows containing missing values (`geom_line()`).
```



From the plot above, the data has retained the patterns when we had weekly intervals for data, albeit the fact that the data is now smoother compared to the weekly one. However, now we have our frequency period equal to twelve, we will now be able to use the fable package and run exponential smoothing algorithm, and thus be able to compare with the benchmark method.

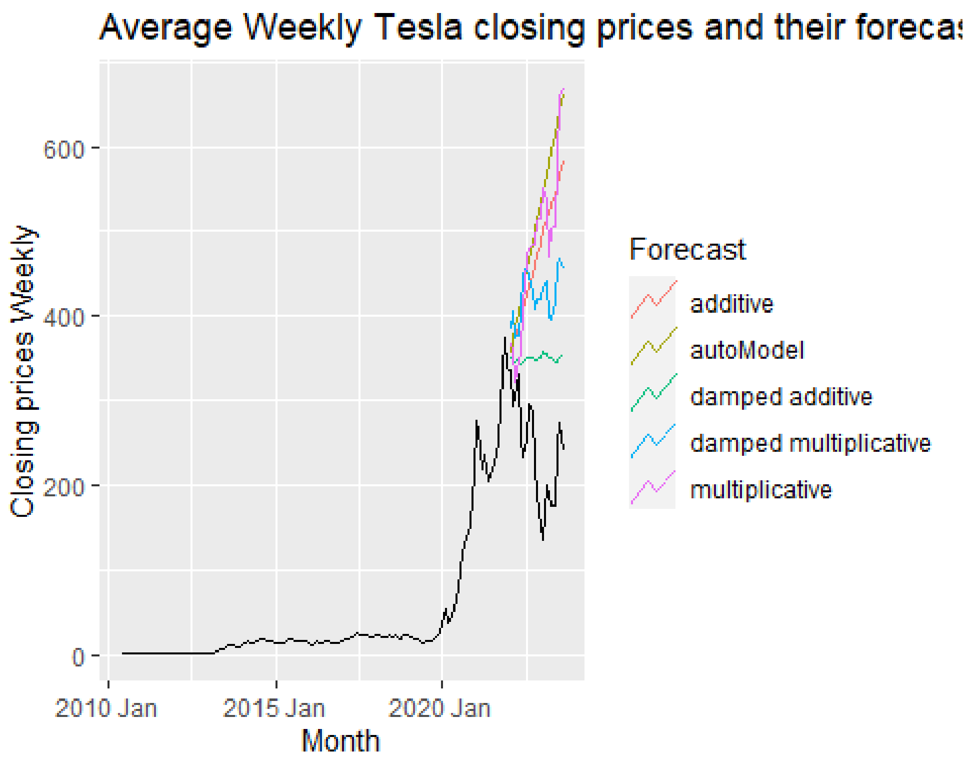
Now, lets create the training and the test set by subsetting our data, we will use as before all the years as the training sets from 2010 till 2021 and the last two years as the test set, namely 2021 and 2022.

```
tesla_tsibble2$year <- as.numeric(substr(tesla_tsibble2$Month,1,4))
class(tesla_tsibble2$year)

## [1] "numeric"

head(tesla_tsibble2)

## # A tsibble: 6 x 5 [1M]
##      Month TSLA.close.avg TSLA.close.avg.sma TSLA.close.avg.sma2  year
##      <mth>          <dbl>              <dbl>               <dbl> <dbl>
## 1 2010 Jun           1.59              NA                  NA     2010
## 2 2010 Jul           1.30              NA                  NA     2010
## 3 2010 Aug           1.30               1.48               NA     2010
## 4 2010 Sep           1.38               1.55               NA     2010
## 5 2010 Oct           1.38               1.62               NA     2010
## 6 2010 Nov           1.94               1.66                1.61  2010

tesla_tsibble2_training <- tesla_tsibble2[which(tesla_tsibble2$year < 2022),]
tesla_tsibble2_test <- tesla_tsibble2[which(tesla_tsibble2$year >= 2022),]

tail(tesla_tsibble2_training)

## # A tsibble: 6 x 5 [1M]
##      Month TSLA.close.avg TSLA.close.avg.sma TSLA.close.avg.sma2  year
##      <mth>          <dbl>              <dbl>               <dbl> <dbl>
## 1 2021 Jul           220.               236.                265.  2021
## 2 2021 Aug           235.               264.                268.  2021
## 3 2021 Sep           251.               285.                275.  2021
## 4 2021 Oct           293.               305.                283.  2021
## 5 2021 Nov           374.               314.                287.  2021
## 6 2021 Dec           340.               323.                289.  2021

tail(tesla_tsibble2_test)

## # A tsibble: 6 x 5 [1M]
##      Month TSLA.close.avg TSLA.close.avg.sma TSLA.close.avg.sma2  year
##      <mth>          <dbl>              <dbl>               <dbl> <dbl>
## 1 2023 Mar           189.               187.                  NA  2023
## 2 2023 Apr           177.               210.                  NA  2023
## 3 2023 May           176.               217.                  NA  2023
## 4 2023 Jun           246.                NA                   NA  2023
## 5 2023 Jul           274.                NA                   NA  2023
## 6 2023 Aug           241.                NA                   NA  2023
```

lets now run the exponential smoothing procedure and its various variations and see how it fits into the data. We will run a variety of models which are variants of the exponential model. One model which we run is the automatic model, where we pass in the target variable and let the model automatically pick the parameters and variables. The second model we run is the model with additive components both in terms of the trend, random and seasonal components. Then, we also experiment with a multiplicative model in the random component and the seasonal component.

The last two models are a variant of the second last two models but we add a dampening/discounting variable into the mix. This is informed by the fact that exponential models tend to be over optimistic or over-forecast, especially for longer time periods. Models with high values for the damping/discounting parameter will tend to be very similar to non-damped forecasts, and if the damping/discounting parameter being 1 being akin to a trend that has not been damped. The consesus for the dampening/discounting parameter is to set it to values between 0.8 and 0.98 since smaller values will tend to approach zero fater and thus damp higher than larger values, however we will set it to a lower value since we do want to be too optimistic from the onset as stock data is highly volatile and upredictable and thus we want our model to resemble a random walk/naive for as much as possible rather than the past values. We mix between two variations of our model, the additive and multiplicative model.

```
fit_tesla_exp <- tesla_tsibble2_training %>%
  model(
     additive = ETS(TSLA.close.avg ~ error("A") + trend("A") + season("A") ),
     multiplicative = ETS(TSLA.close.avg ~ error("M")+ trend("A") +
season("M")),
    `damped additive`  = ETS(TSLA.close.avg ~ error("A") + trend("Ad", phi =
0.70) + season("A") ),
    `damped multiplicative`  = ETS(TSLA.close.avg ~ error("M")+ trend("Ad",
phi = 0.70) + season("M")),
    autoModel = ETS(TSLA.close.avg)
  )

tesla_fc_exp <- fit_tesla_exp %>%
  forecast(new_data = tesla_tsibble2_test)

tesla_fc_exp |>
  autoplot(tesla_tsibble2, level = NULL) +
  labs(title="Average Weekly Tesla closing prices and their forecasts across
Exponentially smoothed models",
       y="Closing prices Weekly") +
  guides(colour = guide_legend(title = "Forecast"))
```

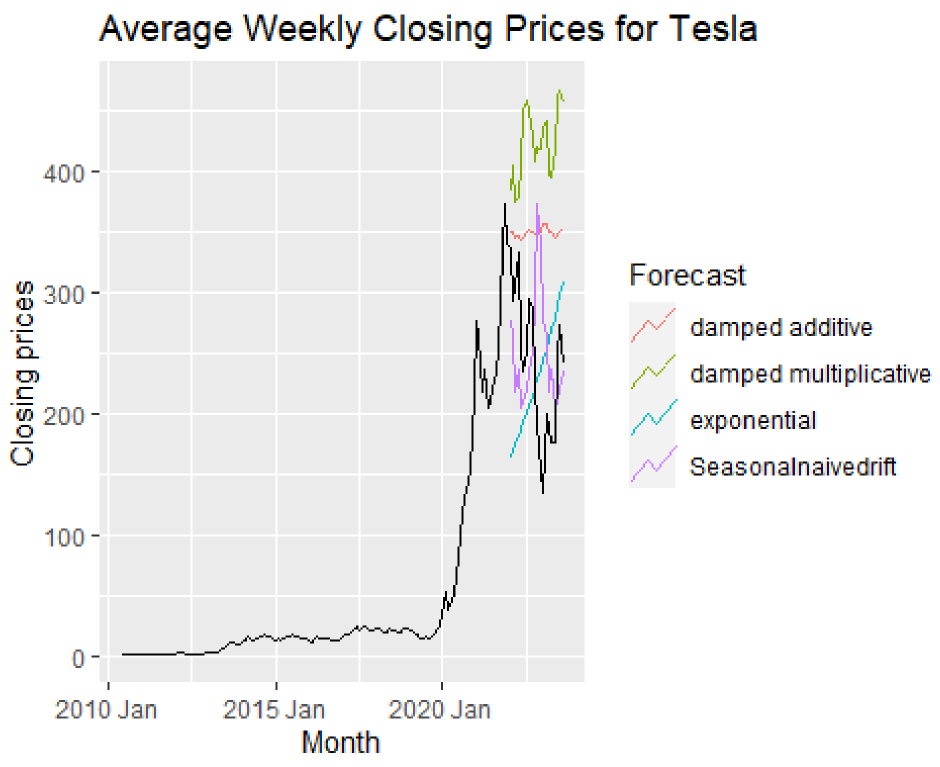


From the models above and looking at their forecasts, it seems like the automatic model is over forecasting drastically, in fact it has the highest forecasted values among all the models. Both the additive and multiplicative models that are not damped also seem to have relatively overforecasted compared to the damped version, however a positive sign is that the multiplicative model seem to have a realistic replication of the seasonal fluctuations. Then, we have the two damped models, with the dampening parameter set to 0.6, which is relatively low compared to industry standards, but to avoid overforecasting, we dampen the trend component to be relatively stronger. The multiplicative damped model looks very promising and it also seems to replicate the past patterns. However, upon closer inspection, it seems to also overforecast the future values. Unfortunately, the tesla stock took a rapid change just when we started forecasted and did not continue in the trend it had ralied on before, thus like knotted regression model we experimented with before, the most promising models in training fail to pick this up and tend to overforecast in the test data.

Now, lets plot our benchmark model against the three most promising model from the exponential setup to see which model can visually perform better.

```
fit_tesla_exp2 <- tesla_tsibble2_training %>%
  model(
    Seasonalnaivedrift =SNAIVE(TSLA.close.avg, drift = TRUE),
    exponential = TSLM(log(TSLA.close.avg) ~ trend()),
    `damped additive` = ETS(TSLA.close.avg ~ error("A") + trend("Ad", phi =
0.70) + season("A") ),
    `damped multiplicative` = ETS(TSLA.close.avg ~ error("M")+ trend("Ad",
phi = 0.70) + season("M"))
  )
```

```
tesla_fc_exp <- fit_tesla_exp2 %>%
  forecast(new_data = tesla_tsibble2_test)

tesla_fc_exp |>
  autoplot(tesla_tsibble2, level = NULL) +
  labs(title="Average Weekly Closing Prices for Tesla",
       y="Closing prices") +
  guides(colour = guide_legend(title = "Forecast"))
```



Visually once more, among the most promising models, the exponential and seasonal naive model with drift seems to be still producing the champion forecasts. The additive model with damped trend also seem to be within reasonable bounds but it still doesn't beat our benchmark model.

Lets run our testing metrics and see which model minimizes the forecast error among the model we have choosen.

```
fit_tesla_exp3 <- tesla_tsibble2_training %>%
  model(
    additive = ETS(TSLA.close.avg ~ error("A") + trend("A") + season("A") ),
    multiplicative = ETS(TSLA.close.avg ~ error("M")+ trend("A") +
season("M")),
    Seasonalnaivedrift =SNAIVE(TSLA.close.avg, drift = TRUE),
    exponential = TSLM(log(TSLA.close.avg) ~ trend()),
    `damped additive` = ETS(TSLA.close.avg ~ error("A") + trend("Ad", phi =
0.70) + season("A") ),
    `damped multiplicative`  = ETS(TSLA.close.avg ~ error("M")+ trend("Ad",
phi = 0.70) + season("M")),
    autoModel = ETS(TSLA.close.avg)

  )

tesla_fc_exp2 <- fit_tesla_exp3 %>%
  forecast(new_data = tesla_tsibble2_test)



accuracy(tesla_fc_exp2, tesla_tsibble2)

## # A tibble: 7 × 10
##   .model                   .type      ME  RMSE   MAE     MPE  MAPE  MASE RMSSE
ACF1
##   <chr>                    <chr>   <dbl> <dbl> <dbl>   <dbl> <dbl> <dbl> <dbl>
<dbl>
## 1 Seasonalnaivedrift   Test    -9.69  82.8  66.8  -12.8   32.6  2.60  1.38
0.794
## 2 additive             Test  -227.   253.  227.  -112.   112.   8.83  4.24
0.825
## 3 autoModel            Test  -269.   301.  269.  -132.   132.  10.4   5.03
0.831
## 4 damped additive      Test  -109.   124.  109.   -55.4   55.4  4.26  2.08
0.717
## 5 damped multiplicati… Test  -183.   195.  183.   -88.1   88.1  7.11  3.27
0.696
## 6 exponential          Test    11.1   90.1  79.6   -4.05  34.5  3.10  1.51
0.799
## 7 multiplicative       Test  -239.   273.  239.  -118.   118.   9.32  4.57
0.817
```

Its clear once more that our benchmark model is still performing way better than the other models we have fitted and across all the forecast error methods we have fitted, its doing well.

Also, visually, the model also seems to have the better distributional form among the models we have fitted. Now, lets run a test to confirming this using the Continuous Ranked Probability Score to see if the distributional form of the damped Additive model compared to our benchmark model and see which one perfoms better

```
tesla_fc_exp2 |>
  accuracy(tesla_tsibble2, list(crps = CRPS))

## # A tibble: 7 × 3
##   .model                    .type  crps
##   <chr>                     <chr> <dbl>
## 1 Seasonalnaivedrift        Test   47.4
## 2 additive                  Test  194.
## 3 autoModel             Test  194.
## 4 damped additive       Test   83.3
## 5 damped multiplicative Test  120.
## 6 exponential           Test   55.6
## 7 multiplicative        Test  154.
```

Here, the Seasonal naive model with drift method is still giving better distributional forecasts than all the exponential models and their variations

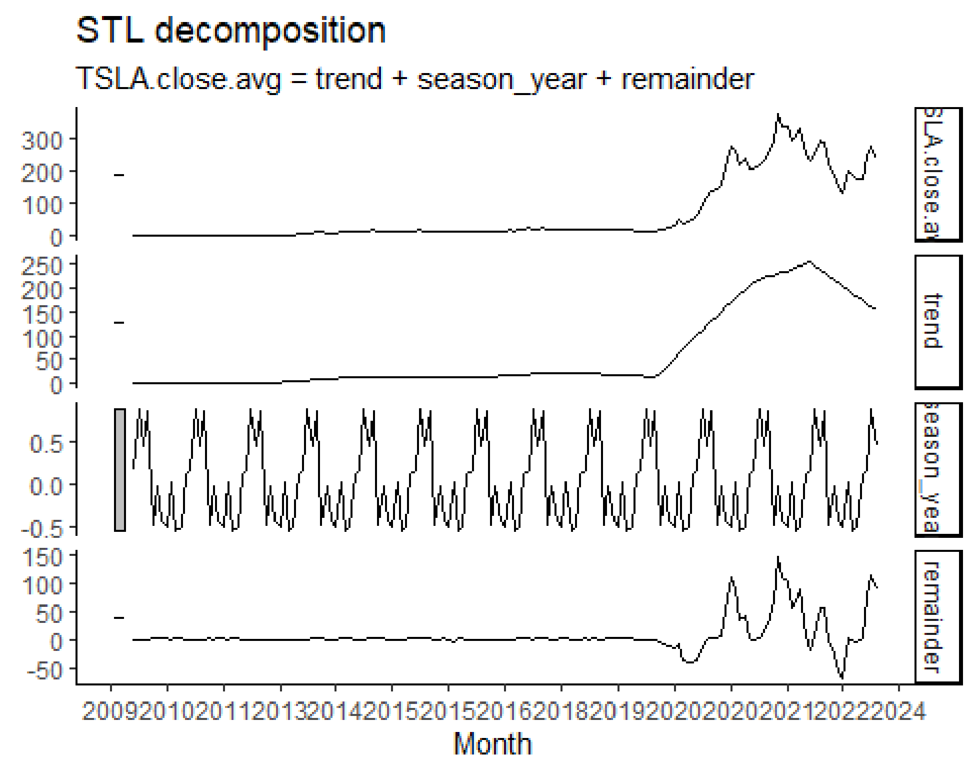
ARIMA Modelling

Now, we shift our attention to the last model we will be fitting to our data, the ARIMA model. ARIMA stands for AutoRegressive Integrated Moving Averages, and is made up of three parts, the AR part, the I part and the MA part. The AutoRegressive part means that we will be regressing the model on itself, that is, we will linearly weight the past values of the data to model the current value of yt, the target variable. Then comes the MA part, this part uses the past errors of the model instead of the past values of the model to forecast future values. Thus, yt can be thought of as the weighted sum of the past estimated error terms.

Then, the intergrated part pertains to the differencing procedure done before fitting the ARIMA model. We do the differencing so that the data is stationary, that is, its statistical properties do not change with time. Note, data that is trended, has seasonality and its variance changes with time (heteroscedasticity) is not stationary and thus breaks the assumptions that need to be met to fit an ARIMA model. The intergration term is the reverse of this process of differencing.

Lets review the patterns observed for this data and see if the statistical properties we need are met or not.

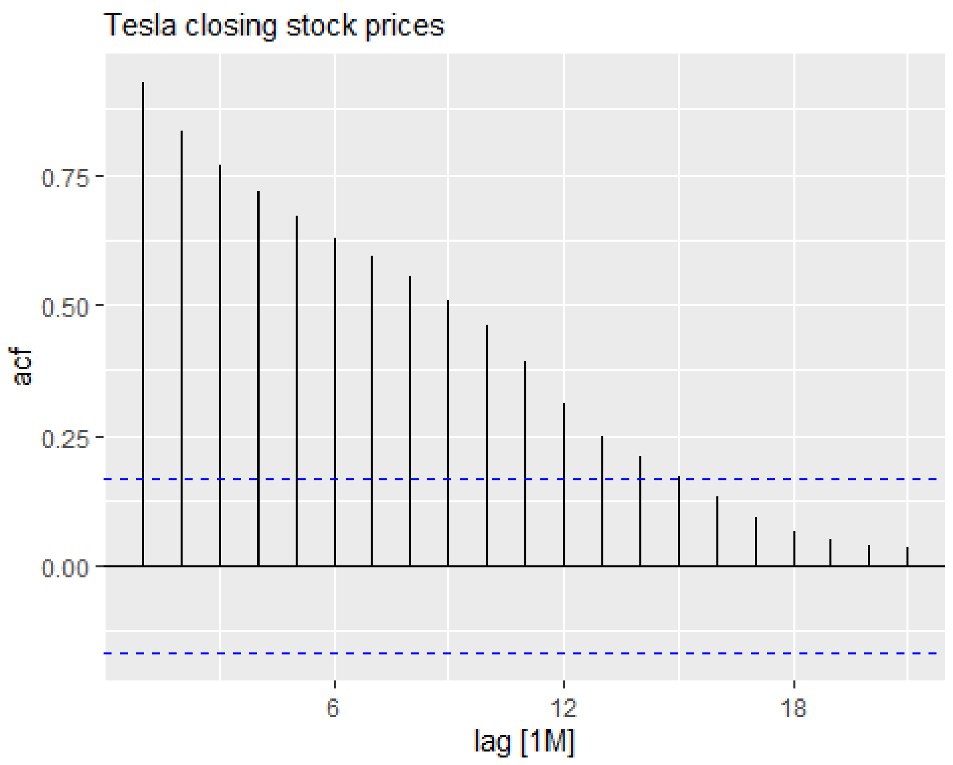
```
tesla_tsibble2 %>%
  model(
    STL(TSLA.close.avg ~ trend(window = 21)+
        season(window = "periodic"),
    robust = TRUE)
  ) %>%
  components() %>%
  autoplot() +
  theme_classic() +
  scale_x_yearweek(date_labels = "%Y", date_breaks = "1 year")
```



From the plot above, its clear that the data has a clear upward trend, and it has a knot point at around 2020 which makes the upward trend even more pronounced. Further, there are clear repetetive patterns that are predictable from when they occur, meaning that we have hints of seasonality in our data. The fact that our seasonality patterns are not very pronounced from around 2010 till about 2020, and then they suddenly becomes even more clear means that we have some sort of heteroscedasticity in model too.

Below, we will run more tests to see for a fact that indeed we do have non-stationarity in our model.

```
tesla_tsibble2_training %>%
  ACF(TSLA.close.avg) %>%
  autoplot() + labs(subtitle = "Tesla closing stock prices")
```

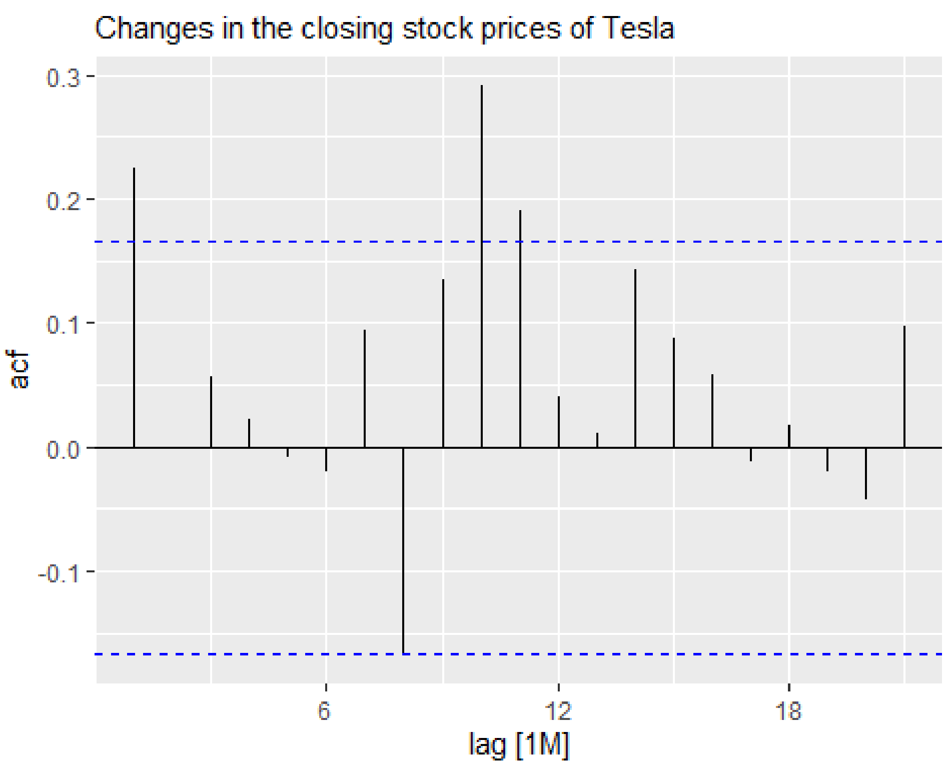


Above, we have plotted the Autocorrelation Function for tesla's closing stock prices, which measures the correlation at different lags between the current value and the various values as different lags in the past. Note, the more spikes are outside of the dotted blue line, the the less stationary the time series is. An ideal situation is to have few spikes being above the dashed blue line, about less than 5% of the spikes, and the ones that are outside should be out of chance. In the plot above, we have many spikes outside the these lines, meaning that our data is not stationary at all.

What we are seeing here is that the closer the values/lags are, the higher the correlation, and the correlation values decreases over time as the lags increases. This means that there is a trend component in our data. Also, there are slight wavelike/sinuid movemevent at regular intervals, which tell us that there is also a seasonal component in the data. Thus, our data is not stationary, from the visual inspections we have made.

Now, lets difference the data, and see if this stationarise the data or not.

```
tesla_tsibble2_training %>%
  ACF(difference(TSLA.close.avg)) %>%
  autoplot() +
  labs(subtitle = "Changes in the closing stock prices of Tesla")
```



Unfortunately, from the plot above, we still have spikes that are outside the boundaries we have set. It seems they are not outside out of chance, but have a repetitive pattern occuring, and this could be attributed to the seasonal pattern we observed above, and thus the differencing we have carried out is not sufficient and thus we also need to carry out seasonal differencing or even more, double differencing for the seasonal and consecutive values.

But, lets first validate these observations with a unit root test, which is used to determine whether differencing is required or not. We use the KPSS version, and we have the null hypothesis that the data is stationary and thus we are looking for evidence that suggest otherwise

```
tesla_tsibble2_training %>%
  features(TSLA.close.avg,unitroot_kpss)
```

```
## # A tibble: 1 × 2
##   kpss_stat kpss_pvalue
##       <dbl>       <dbl>
## 1      1.37        0.01
```

Nice, from the p-value above it is 0.01, and for kpss test, if the p-value is less than 0.01, it will be reported as 0.01 and thus the p-value may actually be smaller than this value of 0.01. This indicates that we should reject the null hypothesis at multiple significance levels such as 0.05,0.25 and 0.01. Thus, we conclude that a differencing is required. But how much differencing do we need to do, as we have seen above that a single round of differencing is not enough to stationarise the data. Thus, we will be running a tests on the data into the number of differences we need to makes.

```
tesla_tsibble2_training |>
  features(TSLA.close.avg, unitroot_ndiffs)
```

```
## # A tibble: 1 × 1
##   ndiffs
##    <int>
## 1      2
```

Nice, from the test above, indeed we need to make not 1 but two rounds of differencing. We also assume that we will also do seasonal differencing since our data has a seasonal component, and this seasonal difference will happen at 12 months round so that its consistent with the data. Further, it is usually advised that we start with seasonal differencing, since it can at times take care of the trend component and thus stop the need to do a second differencing.

```
tesla_tsibble2_training |>
  mutate(diff_close = difference(TSLA.close.avg), 12) |>
  features(diff_close, unitroot_kpss)
```

```
## # A tibble: 1 × 2
##   kpss_stat kpss_pvalue
##       <dbl>       <dbl>
## 1     0.693      0.0142
```

Nice, we have done a seasona differencing, and it seems it has not solved the issue of stationarity as we once more reject the null hypothesis and conclude that we still need to run differencing as the data is not yet stationrary. We will run differencing once more.

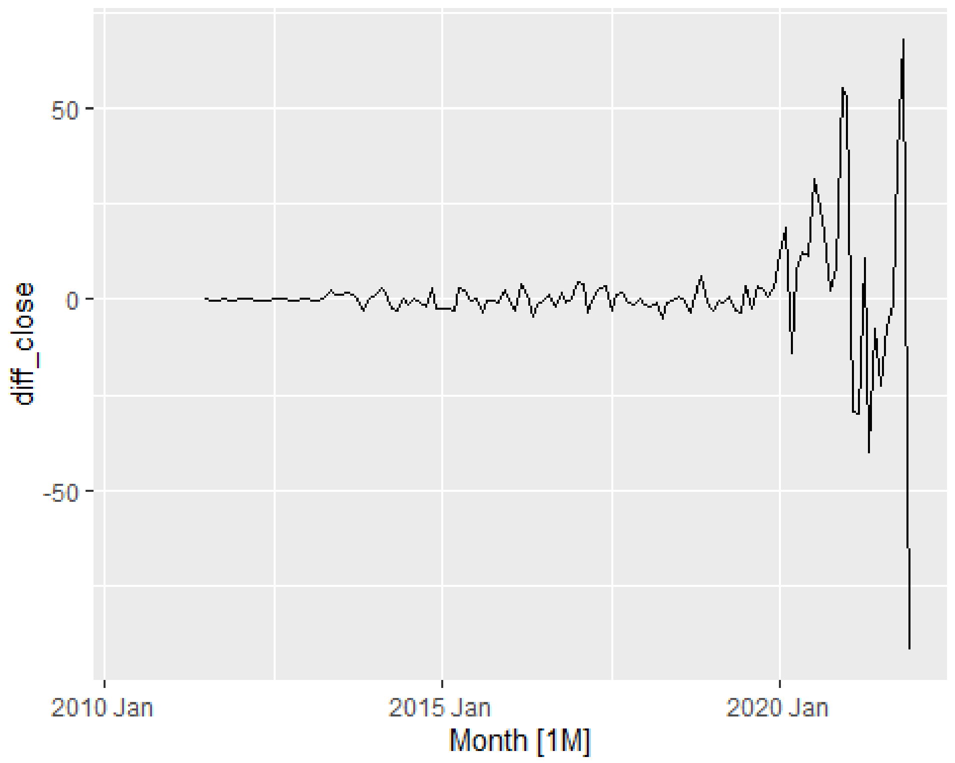
```
tesla_tsibble2_training |>
  mutate(diff_close = difference(difference(TSLA.close.avg), 12),1) |>
  features(diff_close, unitroot_kpss)
```

```
## # A tibble: 1 × 2
##   kpss_stat kpss_pvalue
##       <dbl>       <dbl>
## 1     0.140         0.1
```

Nice, it seems we have solved our problem by doing double differencing. Note, when kpss value is 0.1, that the actual p-value is actually bigger than 0.1, thus after doing double differencing, we fail to reject the null hypothesis and conclude that there is no need for further differencing. However, to conclude that our data is stationary, the data needs to be a white noise process with no signal or pattern dircenable. Thus, we need to plot and run a statistical test to ensure that our data is indeed a white noise process and that all the statistical properties of this data are stationary and do not change as time goes by.

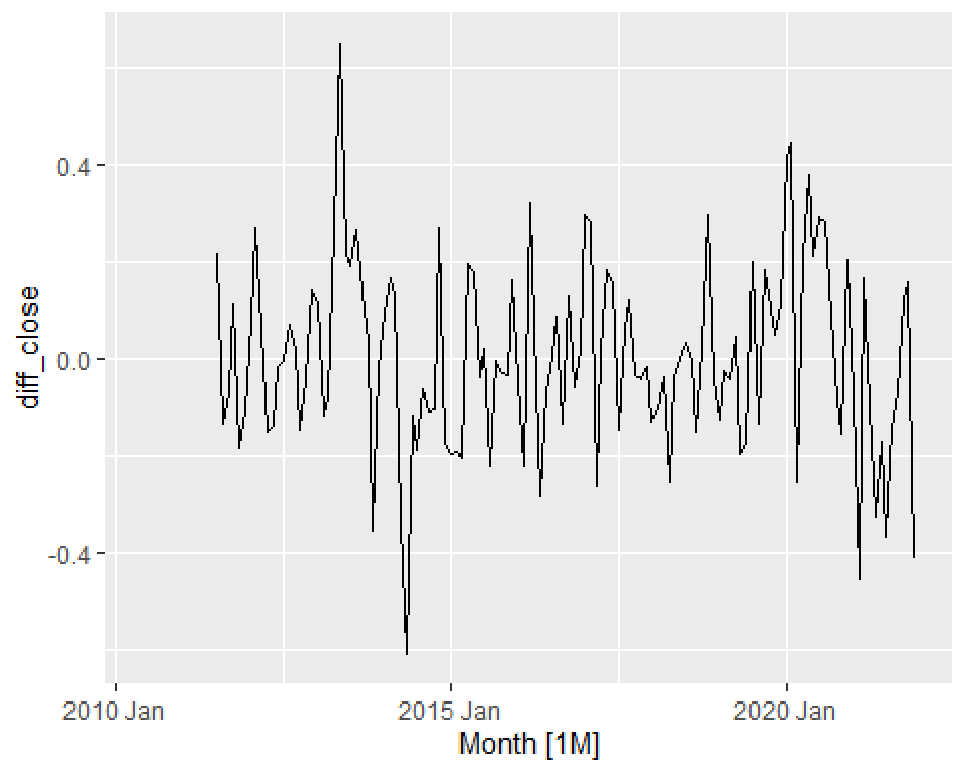
```
tesla_tsibble2_training |>
  mutate(diff_close = difference(difference(TSLA.close.avg), 12),1) |>
  autoplot(diff_close)
```

```
## Warning: Removed 13 rows containing missing values (`geom_line()`).
```



The plot above does not seem to have any trend or seasonal pattern observable. However, the variance in the data seems to be increasing as the data is increasing, meaning there is some sort of heteroscedasticity in the data, that is the variance changes with time. We need to apply a log transformation to solve this issue.

```
tesla_tsibble2_training |>
  mutate(diff_close = difference(difference(log(TSLA.close.avg)), 12),1) |>
  autoplot(diff_close)

## Warning: Removed 13 rows containing missing values (`geom_line()`).
```



Nice, after logging and then differencing, it seems we have solve the issue of heteroscedasticity in the data and now our data seems to be homoscedastic and also no other statistical properties seem to be contingent on the time at which the data is observed.

Below, lets run a test to assess and ascertain this and conclude that indeed there is stationarity in the data using the Ljung-Box test for stationarity. The null hypothesis for the test is that the data is stationary, that it came from a white noise process. The alternative hypothesis states that the data did not come from a white noise process and thus is not stationary.

```
 tesla_tsibble2_training$diff_close <- tesla_tsibble2_training |>
  mutate(diff_close = difference(difference(log(TSLA.close.avg)), 12),1)

tesla_tsibble2_training %>%
features(diff_close, ljung_box, lag = 10)

## # A tibble: 1 × 2
##   lb_stat lb_pvalue
##     <dbl>     <dbl>
## 1   1143.         0
```

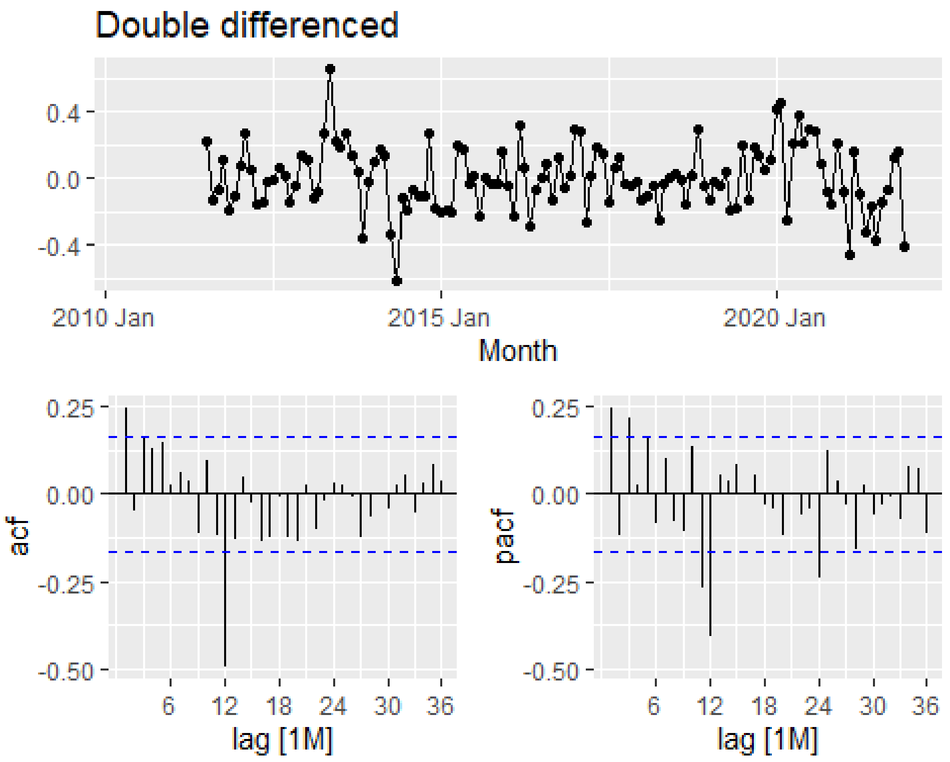
The p-value is very low, indicating that at any significance level, we will fail to reject the null hypothesis in favour of the alternative and conclude that indeed the data came from a white noise process and thus its stationary as alluded by the plot we made above. Thus, to stationarise our data, we needed to deal with the issue of the trend component, the seasonality and then the subtle issue of heteroscedasticity through logging. Only then we could have stationary data.

Now that we know how to get our data to be stationary, we move towards the actual modelling process, and we will apply an AR model, then an MA model, then apply the full Seasonal ARIMA model and then decide which model performs the best using the AICc measure, which whose results are equivalent to the k-fold cross validation.

Model fitting

To fit our models, we first need to run a bunch of plots on the differenced data to have a sense of the parameters we will be using to fit the simplest models which are of the form ARIMA(p,d,0)(P,D,0) or ARIMA(0,d,q)(0,P,Q). These two variations of our models can be simple estimated by using the PACF and ACF plots. However, if we have a model of the form ARIMA(p,d,q)(P,D,Q).

```
view(tesla_tsibble2_training)

tesla_tsibble2_training |>
  gg_tsdisplay(difference(log(TSLA.close.avg), 12) |> difference(),
               plot_type='partial', lag=36) +
  labs(title = "Double differenced", y="")

## Warning: Removed 13 rows containing missing values (`geom_line()`).

## Warning: Removed 13 rows containing missing values (`geom_point()`).
```



Lets first look at the Partial Autocorrelation plot which can help us fit the ARIMA(p,d,0)(P,D,0), which is equivalent to the acf but if controlled for spurious regression. We see that there are two spikes outside the significance region and thus we will set p=2. For the seasonal part, there are also two spikes outside at 12 and 24, thus we will have P=2 for the seasonal part of the ARIMA model. We set d = 1 and D =1 since we have run double differencing, for the seasonal and regular intervals.

When we look at the Auto Correlation Plot, which can help us find parameters for the ARIMA(0,d,q)(0,P,Q). Again, we set d = 1 and D=1 since we did double differencing. For the q and Q parameters, we use the AFC plot. From the plot above, we see we have one significant spike at lag one and thus we will set q = 1. For the seasonal portion, we also set it to 1 since there is one significant spike at the 12th lag.

We will also mix up our model to find other variations that can work, by mixing the PACF and ACF parameter guided selections. Further, we will use an exhaustive search for the most appropriate parameters by not limiting it by doing approximation. We also set the stepwise to false to also encourage the algorithmn to run an exhaustive search.

```
fit_arima <- tesla_tsibble2_training |>
  model(
    arima012011 = ARIMA(log(TSLA.close.avg) ~ pdq(0,1,2) + PDQ(0,1,1)),
    arima210011 = ARIMA(log(TSLA.close.avg) ~ pdq(2,1,0) + PDQ(2,1,0)),
    arima210011 = ARIMA(log(TSLA.close.avg) ~ pdq(2,1,0) + PDQ(0,1,1)),
    auto = ARIMA(log(TSLA.close.avg), stepwise = FALSE, approx = FALSE)
#Robust and exhaustive...
  )
```

To figure out which of these models are most robust, we make use of the AICc measure, whose results for a model choice are similar from running a k-fold cross validation procedure. Thus, AICc measure solves the issue of overfitting in picking the most appropriate model.

```
fit_arima  %>%
  pivot_longer(everything(), names_to = "Model name", values_to = "Orders")

## # A mable: 3 x 2
## # Key:     Model name [3]
##   `Model name`                       Orders
##   <chr>                             <model>
## 1 arima012011  <ARIMA(0,1,2)(0,1,1)[12]>
## 2 arima210011  <ARIMA(2,1,0)(0,1,1)[12]>
## 3 auto           <ARIMA(1,1,1) w/ drift>
```

Nice, we have organize the data into a longer format, now lets run the tests to figure which model has the lowest AICc and thus fit the data better.

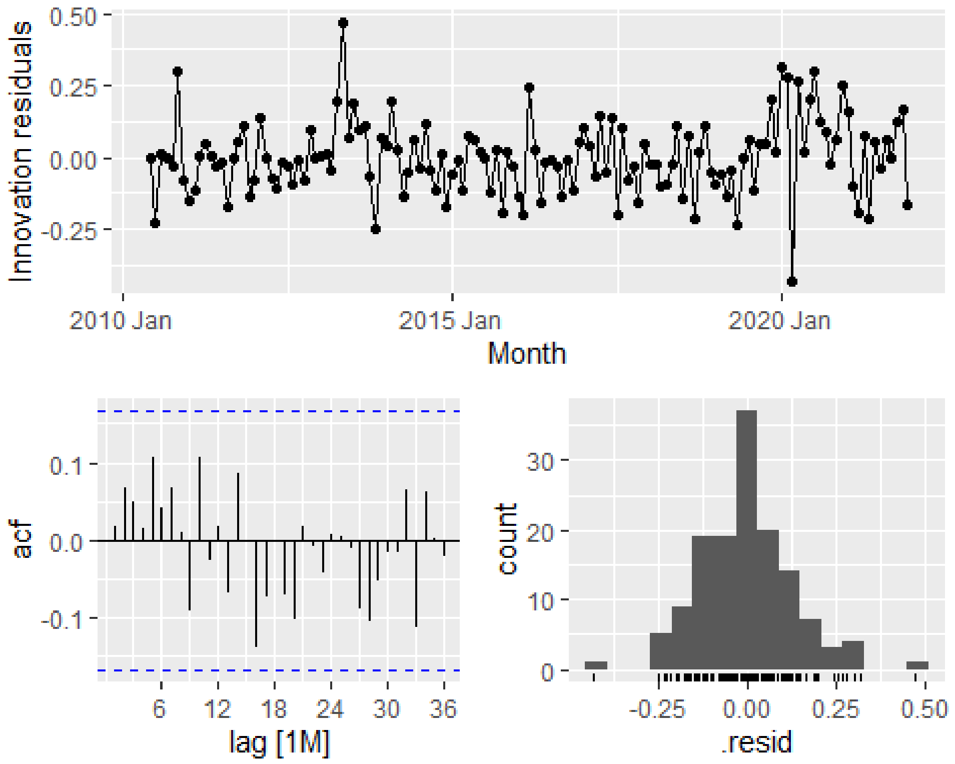
```
glance(fit_arima) %>% #uSE GLANCE TO FIND THE BETTER MODEL/ THE METRICS
  arrange(AICc) %>% #Arrange them by their AICc values
  select(.model:BIC) #select the mmetrics we are most interested in

## # A tibble: 3 × 6
##   .model      sigma2 log_lik   AIC  AICc   BIC
##   <chr>        <dbl>   <dbl> <dbl> <dbl> <dbl>
## 1 auto        0.0175    84.7 -161. -161. -150.
## 2 arima012011 0.0205    61.5 -115. -115. -104.
## 3 arima210011 0.0208    60.1 -112. -112. -101.
```

The model that minises AICc by a huge margin is the automatically selected model, and the other models don't even come close to it. Thus, that is the model we will select to do our forecasting with.

However, one final stretch is to figure out if the residuals from fitting the arima model are a white noise process or not, meaning that they are completetly at random and there's not pattern or signal detectable in them.

Lets see these residuals.

```
fit_arima %>%
  select(auto) %>%
  gg_tsresiduals(lag = 36)
```



The error terms do seem to follow a white noise process as there is not pattern that can be found on the data. Further, the error terms seem to be normally distributed from the histogram plot. Looking at the ACF plot there are no spikes that are outside of the signficance barriers out of the 32 lags we have drawn out, even out chance.

Lets run the ljung_box test which has the null hypothesis that the data is stationary.

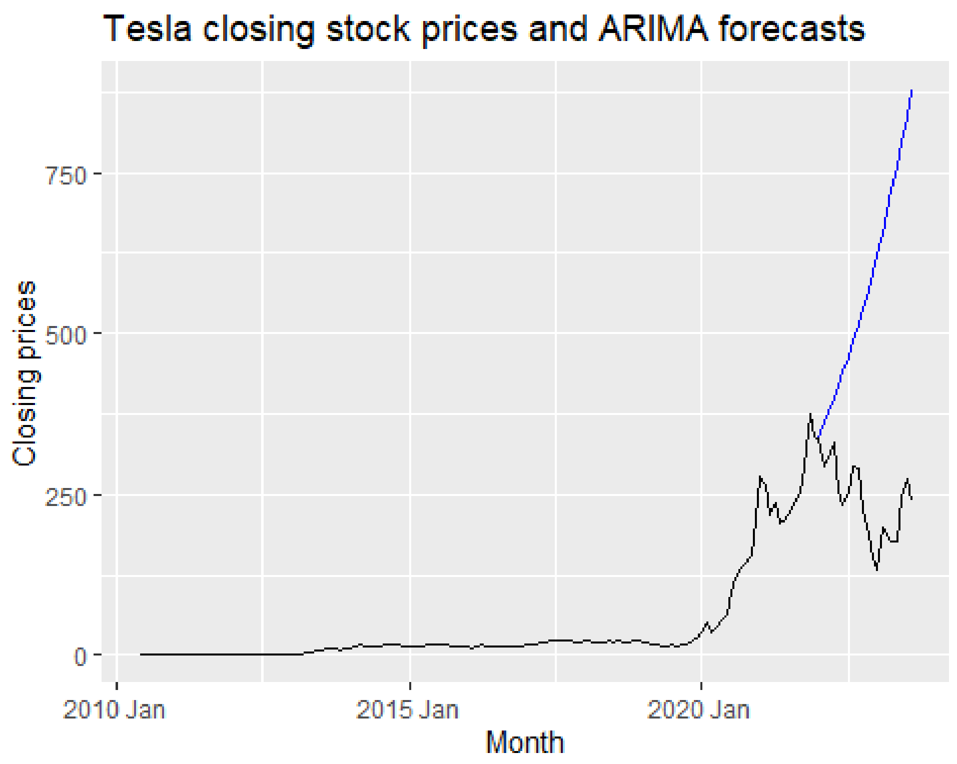
```
augment(fit_arima)%>%
  filter(.model == "auto") %>%
  features(.innov, ljung_box, lag = 36, dof = 4)
```

```
## # A tibble: 1 × 3
##   .model lb_stat lb_pvalue
##   <chr>    <dbl>     <dbl>
## 1 auto      23.3     0.867
```

Under the null hypothesis that the data is stations and is from a white noise process, we fail to reject the null hypothesis at any of the significance levels and conclude that indeed that the error terms are stationary.

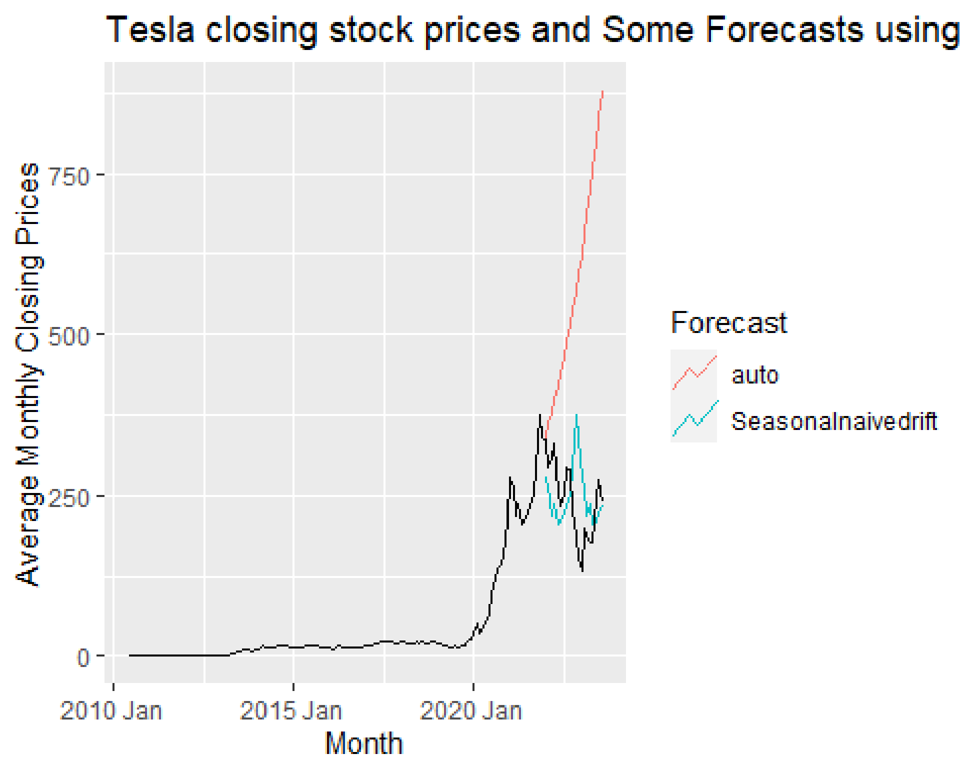
Now, lets forecast

```
tesla_fc_arima <- fit_arima %>%
  forecast(new_data = tesla_tsibble2_test)

tesla_fc_arima |>
  filter(.model == "auto") |>

  autoplot(tesla_tsibble2, level = NULL) +
  labs(title="Tesla closing stock prices and ARIMA forecasts",
       y="Closing prices") +
  guides(colour = guide_legend(title = "Forecast"))
```



Now that we have the best model, its time we compare the best perfoming model between it and the benchmark model we had. We first plot them together below to see how their forecast looks or rather differs if it does.

```r
fit_arima2 <- tesla_tsibble2_training |>
  model(
    Seasonalnaivedrift =SNAIVE(TSLA.close.avg, drift = TRUE),
    auto = ARIMA(log(TSLA.close.avg), stepwise = FALSE, approx = FALSE)
#Robust and exhaustive...
  )



tesla_fc_arima2 <- fit_arima2 %>%
  forecast(new_data = tesla_tsibble2_test)

tesla_fc_arima2 |>
  autoplot(tesla_tsibble2, level = NULL) +
  labs(title="Tesla closing stock prices and Some Forecasts using ARIMA model
and Benchmark model",
       y="Average Monthly Closing Prices") +
  guides(colour = guide_legend(title = "Forecast"))
```



From the plot above, it seems once more, the benchmark method beat this relatively complex ARIMA model by a huge margin. It seems the fact that tesla refused to grow in line with the past patterns affected it and the model was not able to predict the sudden change in its trajectory.

Lets confirm this with one of the tests we used prior, we test which models have the lowest forecast error rate between the two model, the benchmark and the ARIMA model.

```
accuracy(tesla_fc_arima2, tesla_tsibble2)

## # A tibble: 2 × 10
##   .model             .type     ME   RMSE   MAE    MPE  MAPE  MASE RMSSE
ACF1
##   <chr>              <chr>   <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>
<dbl>
## 1 Seasonalnaivedrift Test    -9.69  82.8  66.8  -12.8  32.6  2.60  1.38
0.794
## 2 auto               Test  -333.   388.  333.  -163.  163.  13.0   6.48
0.840
```

Looking at all the forecast error measurements with all the modifications, the benchmark method seems to have done a good job nonetherless. Thus, the seasonal naive model with drift judging from the accuracy measures is the most robust model for forecasting future stock prices of tesla.

Lets now assess the distributional form of the ARIMA model and compare it with the benchmark model which is the ARIMA model. We use the Continous Ranked Probability Score which is an extension of the Winker score.

```
tesla_fc_arima2 |>
  accuracy(tesla_tsibble2, list(crps = CRPS))

## # A tibble: 2 × 3
##   .model             .type  crps
##   <chr>              <chr> <dbl>
## 1 Seasonalnaivedrift Test   47.4
## 2 auto               Test  184.
```

From the test above, its clear that the seasonally naive model with drift if giving better distributional forecasts compared to the ARIMA model.

Conclusion

From the explaratory analysis part, we found that Tesla stock prices have a seasonal and a long term upward trend. There was also heteroscedasticity in the data, with variance in the seasonality of the data increases sharply as time goes. Thus the data was not stationary. Further, there was a major surge in the closing stock prices of tesla at around 2019, which then sustained a long upward trend which started to fizzle around late 2021.

From all the tests we have run above, its uncanny how the benchmark model has beat all the seemingly advanced models in predicting tesla stock prices, the *seasonal naive model with drift*. This may be a testament of the fact that the stock markets are efficient and that there are no descernable patterns that can be exploited and if they exist, they tend to quickly die out as the market participants take advantage of them. Also, the seasonal naive model also have the best distributional accuracy out of the models we have assessed which tend to over forecast using the expanded Winker score, the Continous Probability Score. Thus, indeed movements in the stock markets tend to be a random walk, especially in the short term.