HW3 Review

Anthony Tricarico

> **Warning**
>
> The following document should not be viewed as a solution to the exercises, but rather as a resource that can help you think about how you can solve similar problems. For issues, always refer back to the original solution provided by your Professor in class.

1 Exercise 1

First, we import the necessary libraries.

```
library(tidyquant)
library(fpp3)
```

You can look up on the internet the ticker symbol for General Electric (GE) to understand what to input as the first argument to the `tq_get()` function.

```
ge_data <- tq_get("GE", from = "2024-06-01", to = "2024-11-10")
head(ge_data)
```

```
# A tibble: 6 x 8
  symbol date        open  high   low close  volume adjusted
  <chr>  <date>     <dbl> <dbl> <dbl> <dbl>   <dbl>    <dbl>
1 GE     2024-06-03  166.  167.  159.  161. 5220500     161.
2 GE     2024-06-04  161   162.  158.  161. 5598600     161.
3 GE     2024-06-05  162.  163.  161.  163. 4279900     162.
4 GE     2024-06-06  162.  163.  160.  161. 3867300     160.
5 GE     2024-06-07  161.  164.  160.  162. 3341800     161.
6 GE     2024-06-10  162   164.  162.  163. 4385900     162.
```

Now we check if the data we have is already in the form of a tsibble.

```
is_tsibble(ge_data)
```

```
[1] FALSE
```

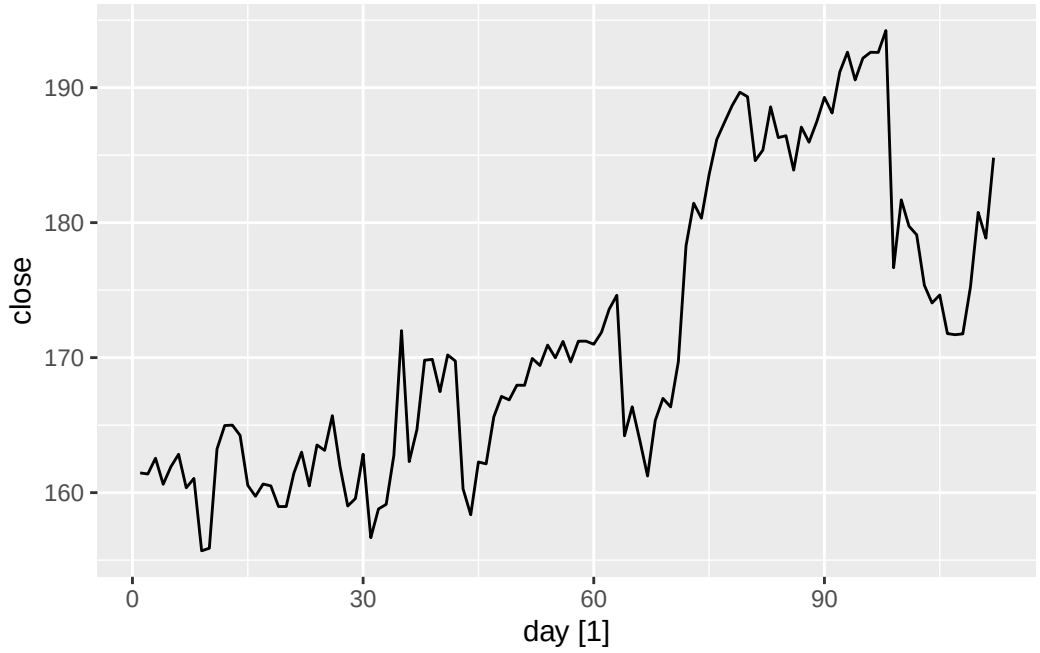
Since it is not a tsibble as it does not include a continuous (without gaps) index, we need to create one for it. We do so by using the `row_number()` function to produce a sequence of numbers going from 1 to the number of the last row of the dataset. Finally, we convert the tibble into a tsibble using the `as_tsibble()` function and specifying `day` (the column we just created) as the index.

```
close_ge <- ge_data %>%
  mutate(day = dplyr::row_number()) %>%
  select(close, day)

ge_tsibble <- as_tsibble(close_ge, index = day)
```

Before we delve deeper into the analysis, it is useful to visualize a time plot of our variable of interest (`close`) over time.

```
autoplot(ge_tsibble, .vars=close)
```



1.1 Point A)

The first step is to set up the cross validation structure using the `stretch_tsibble()` function and specify the `.step = 1` and `.init = 3` arguments to say that we want to start with a first set of observations which will be made of 3 rows in our data and that each time we want to increase the number of observations in this set by 1. These values are arbitrary and you can pick different ones to set up a different cross validation structure.

```
(ge_cr <- stretch_tsibble(ge_tsibble, .step = 1, .init = 3))
```

```
# A tsibble: 6,325 x 3 [1]
# Key:       .id [110]
   close   day   .id
   <dbl> <int> <int>
 1  161.     1     1
 2  161.     2     1
 3  163.     3     1
 4  161.     1     2
 5  161.     2     2
 6  163.     3     2
 7  161.     4     2
 8  161.     1     3
 9  161.     2     3
10  163.     3     3
# i 6,315 more rows
```

Now, we fit the Naive and drift model to our data organized with the proper cross validation structure.

```
fit <- model(ge_cr,
      naive = NAIVE(close),
      drift = RW(close~drift()))
```

And then we produce the 1-step forecast using our models, as asked.

```
fore <- forecast(fit, h=1)
```

We also get some information about which model performs better. Based solely on the RMSE, the Naive method seems to be performing better as it has the lowest value among the two.

```
select(accuracy(fore, ge_tsibble), .model, RMSE)
```

```
# A tibble: 2 x 2
  .model  RMSE
  <chr>  <dbl>
1 drift   3.63
2 naive   3.58
```

1.2 Point B)

Now, we split the data in train and test sets gathering the first 90 rows in the train set and the remaining rows in the

```
train <- ge_tsibble[1:90,]
test <- ge_tsibble[91:nrow(ge_tsibble),]
```

```
nrow(ge_tsibble)
```

```
[1] 112
```

> **Tip**
>
> In the cell above we used slicing to split the dataset in train and test sets. Remember about the possibility of doing this in the future. Also, notice how the `nrow()` function returns the number of the last row (or the total number of rows) present in the dataset that is passed as an argument inside of it.

Then, we follow the same steps as above. We train model on training set and

```
fit_train <- model(train,
      naive = NAIVE(close),
      drift = RW(close ~ drift()))
```

then produce forecast on test data.

```
fore_train <- forecast(fit_train, new_data=test)
select(accuracy(fore_train, test), .model, RMSE)
```

```
# A tibble: 2 x 2
  .model  RMSE
  <chr>  <dbl>
1 drift   14.2
2 naive   10.5
```

The two methods yield similar results as both show that the Naive method is the one with the lowest RMSE. However, I would trust more the results obtained through cross-validation as they are obtained by averaging the entire set of prediction errors obtained by producing a 1-step forecast each time. Also, cross validation is a more advanced technique to test our model on a set of data it did not see before, and therefore it is useful to prevent model overfitting.

2 Exercise 2