Data 624 - HW 3 - Benchmark Forecasting Methods

William Jasmine

2024-09-22

All exercises can be found in "Forecasting: Principles and Practice" written by Rob J Hyndman and George Athanasopoulos.

Setup

The following packages are required to rerun this `.rmd` file:

```r
library(tidyverse)
library(fpp3)
```

Exercise 5.1

Description

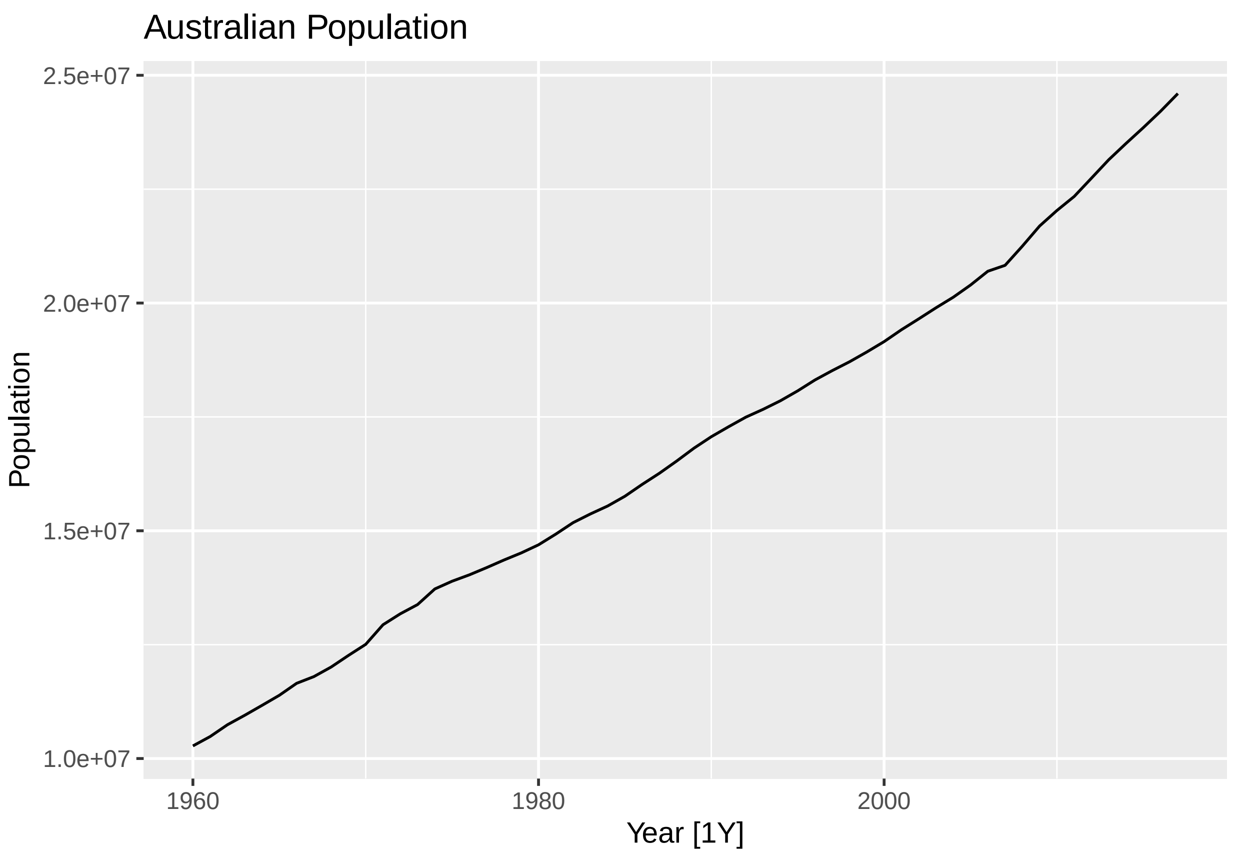
Produce forecasts for the following series using whichever of `NAIVE(y)`, `SNAIVE(y)` or `RW(y ~ drift())` is more appropriate in each case:

- Australian Population (`global_economy`)
- Bricks (`aus_production`)
- NSW Lambs (`aus_livestock`)
- Household wealth (`hh_budget`).
- Australian takeaway food turnover (`aus_retail`).

Solution

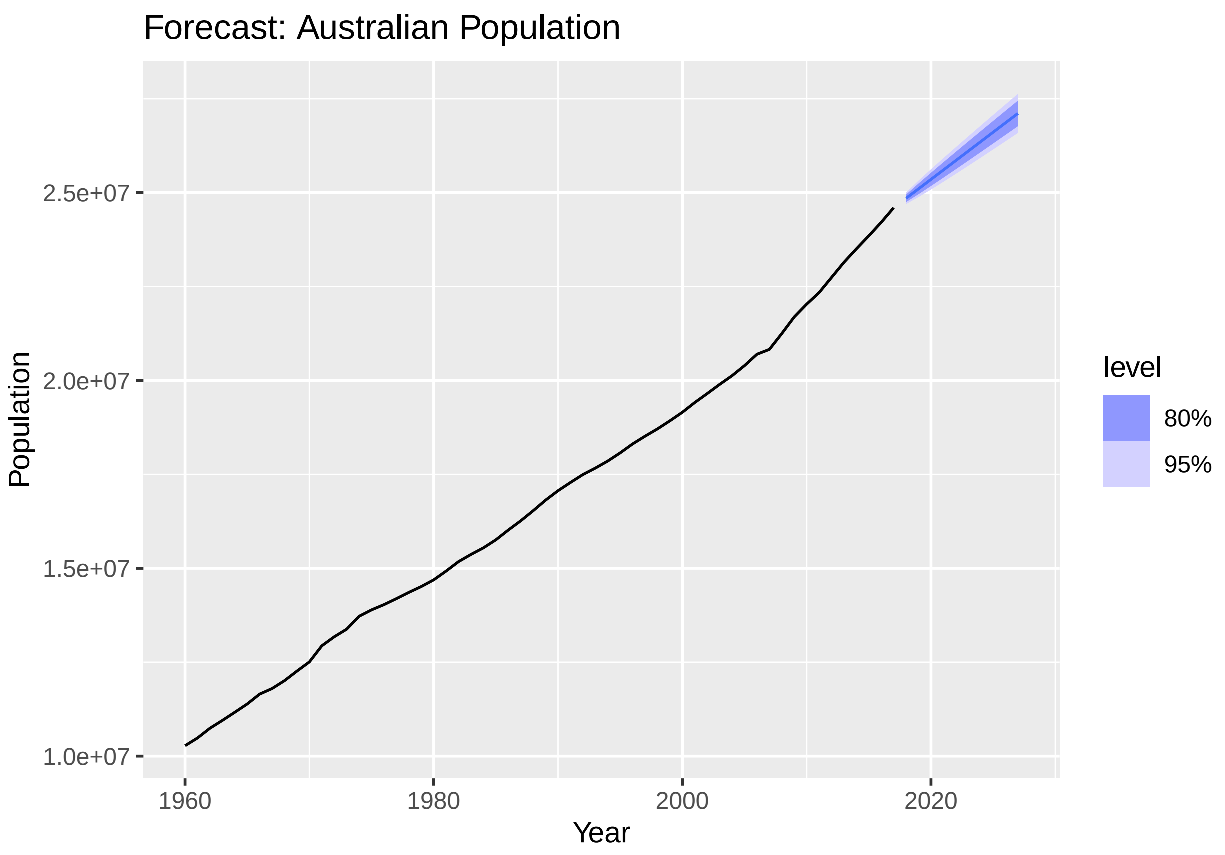
The cell below uses the `global_economy` dataset to plot a time series of the Australian population:

```r
global_economy %>%
  filter(Code == 'AUS') %>%
    autoplot(Population) +
      labs(
        title = 'Australian Population'
      )
```



The plot above exhibits a clear upward trend, but no apparent seasonality. As such, the `drift` forecasting methodology is most appropriate:

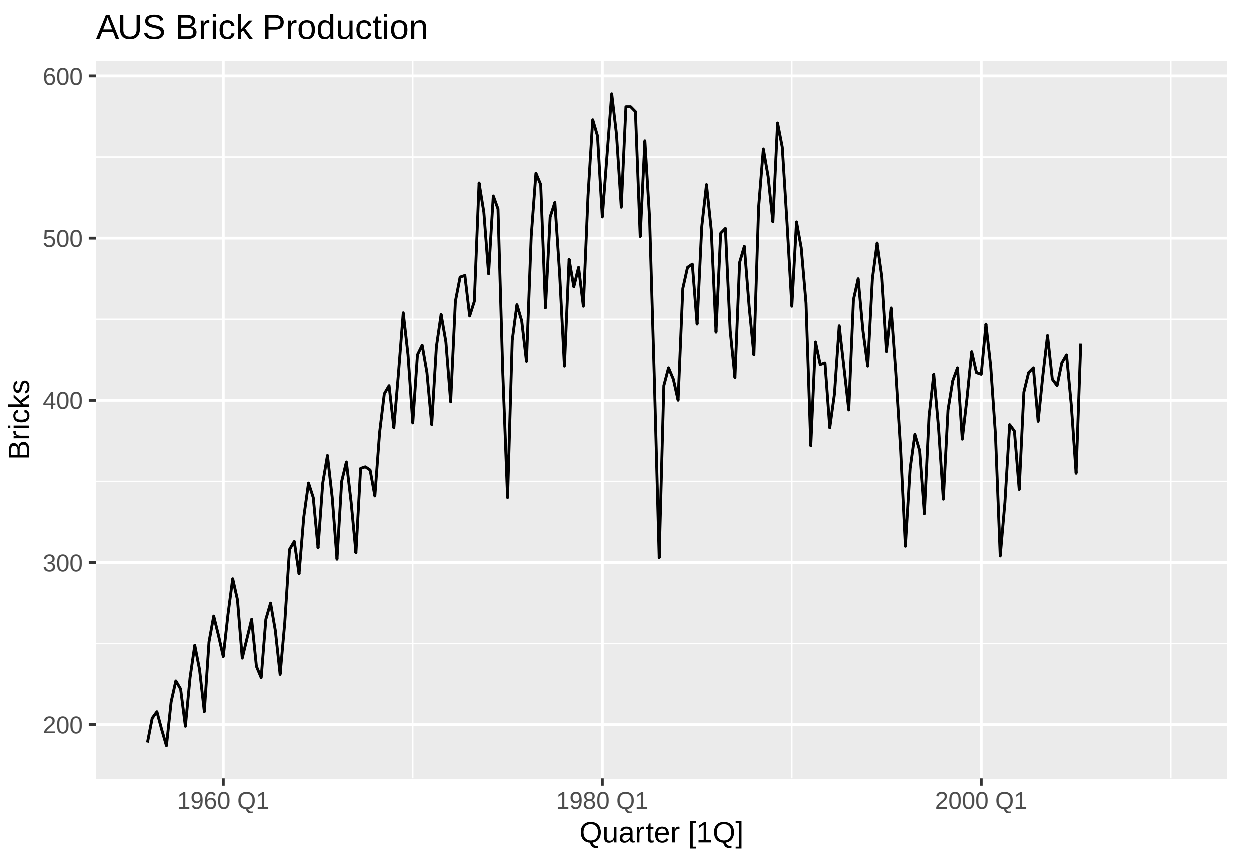
```
global_economy %>%
  filter(Code == 'AUS') %>%
    model(RW(Population ~ drift())) %>%
      forecast(h = '10 years') %>%
        autoplot(global_economy) +
          labs(
            title = 'Forecast: Australian Population'
          )
```



The cell below plots a time series of Australian brick production over time using the `aus_production` dataset:

```r
aus_production %>%
  autoplot(Bricks) +
    labs(
      title = 'AUS Brick Production'
    )
```

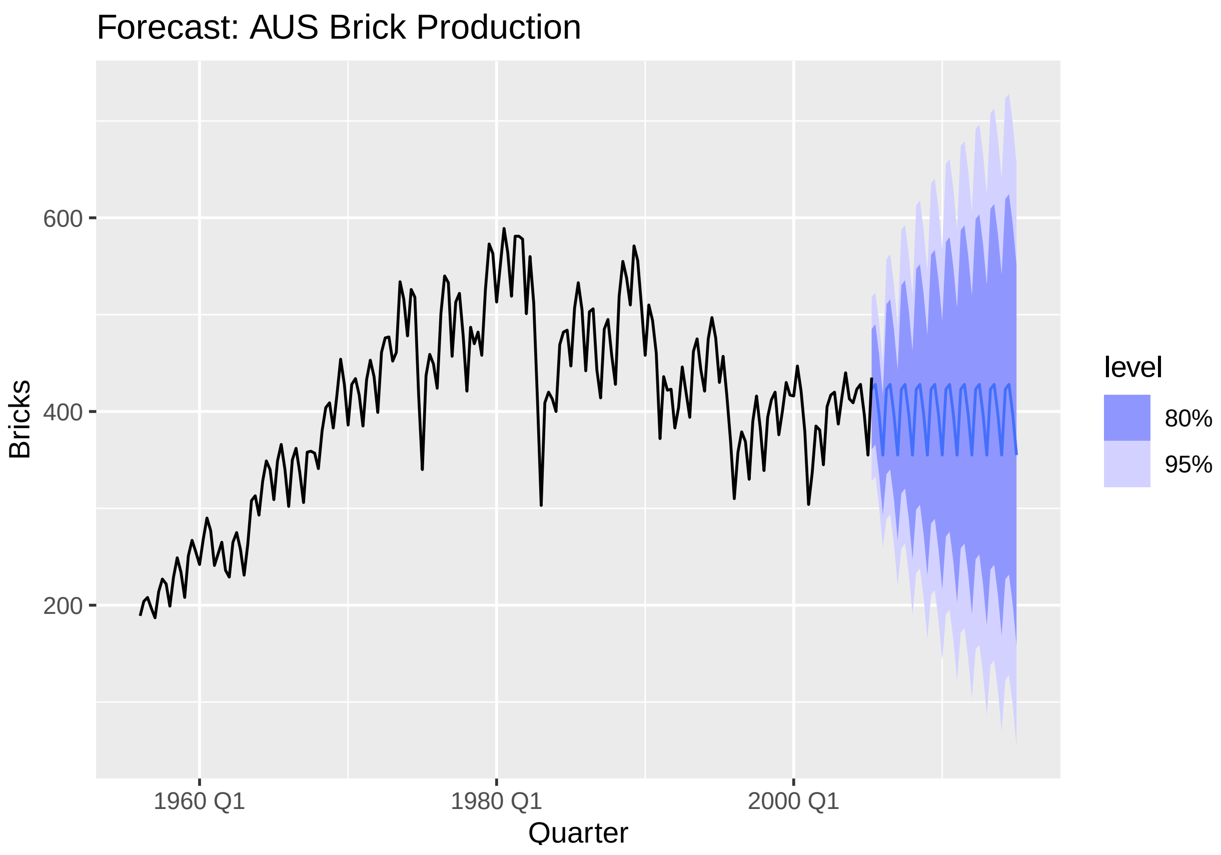
```
## Warning: Removed 20 rows containing missing values or values outside the scale range
## (`geom_line()`).
```



The plot above exhibits seasonality, but no clear trend. As such, the `SNAIVE` forecasting methodology is most appropriate:

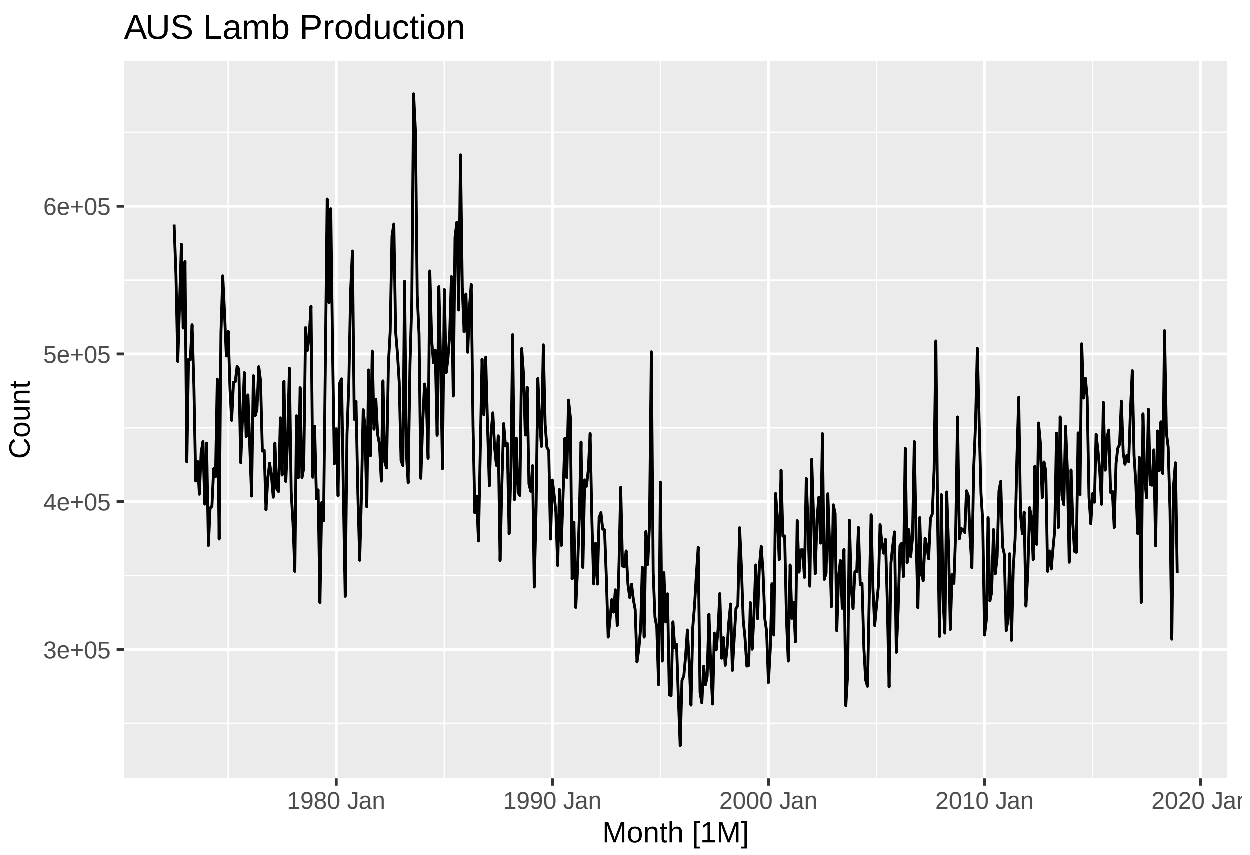
```r
aus_production %>%
    filter(Quarter < yearquarter('2005 Q2')) %>%
    model(SNAIVE(Bricks)) %>%
      forecast(h = '10 years') %>%
        autoplot(aus_production) +
          labs(
            title = 'Forecast: AUS Brick Production'
          )
```

```
## Warning: Removed 20 rows containing missing values or values outside the scale range
## (`geom_line()`).
```



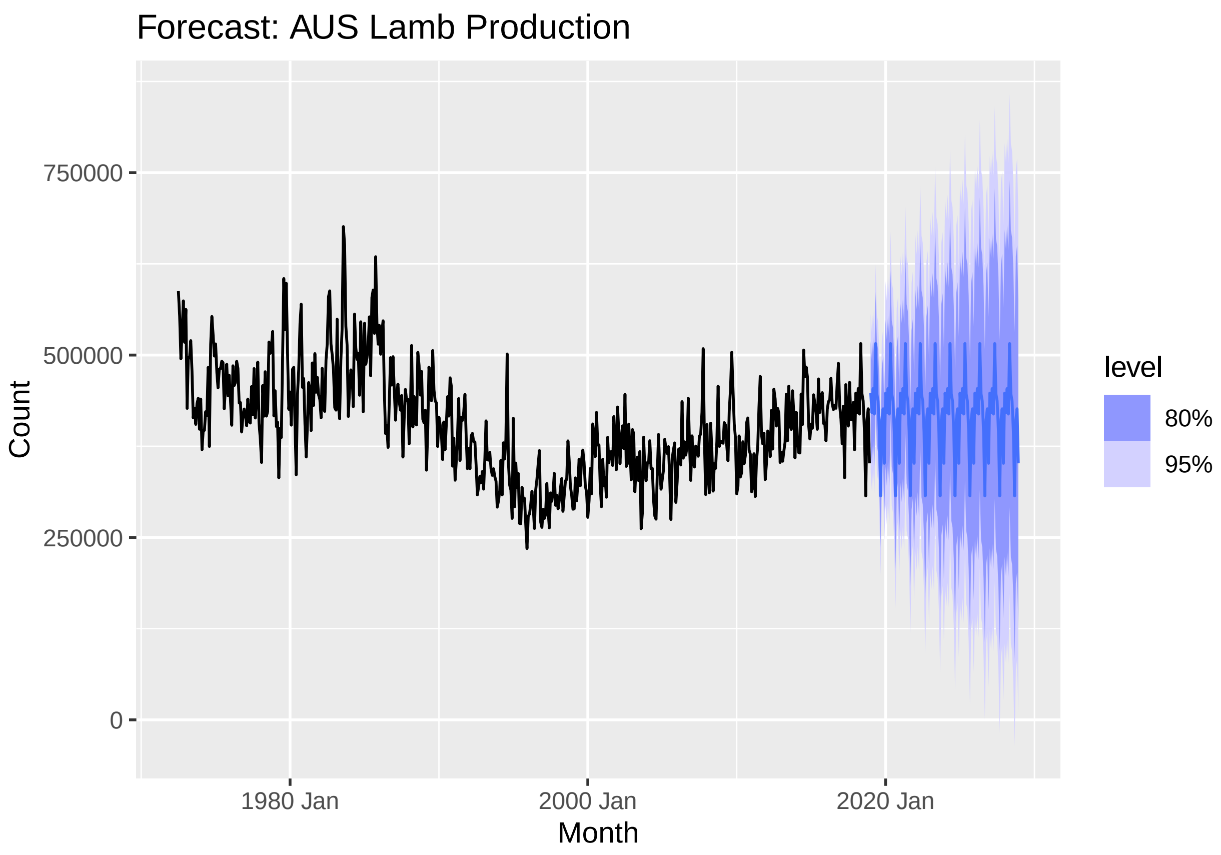
Next, the cell below plots a time series of the number of Lambs produced in New South Wales using the `aus_livestock` dataset:

```r
aus_livestock %>%
  filter(State == 'New South Wales' & Animal == 'Lambs') %>%
    autoplot(Count) +
      labs(
        title = 'AUS Lamb Production'
      )
```



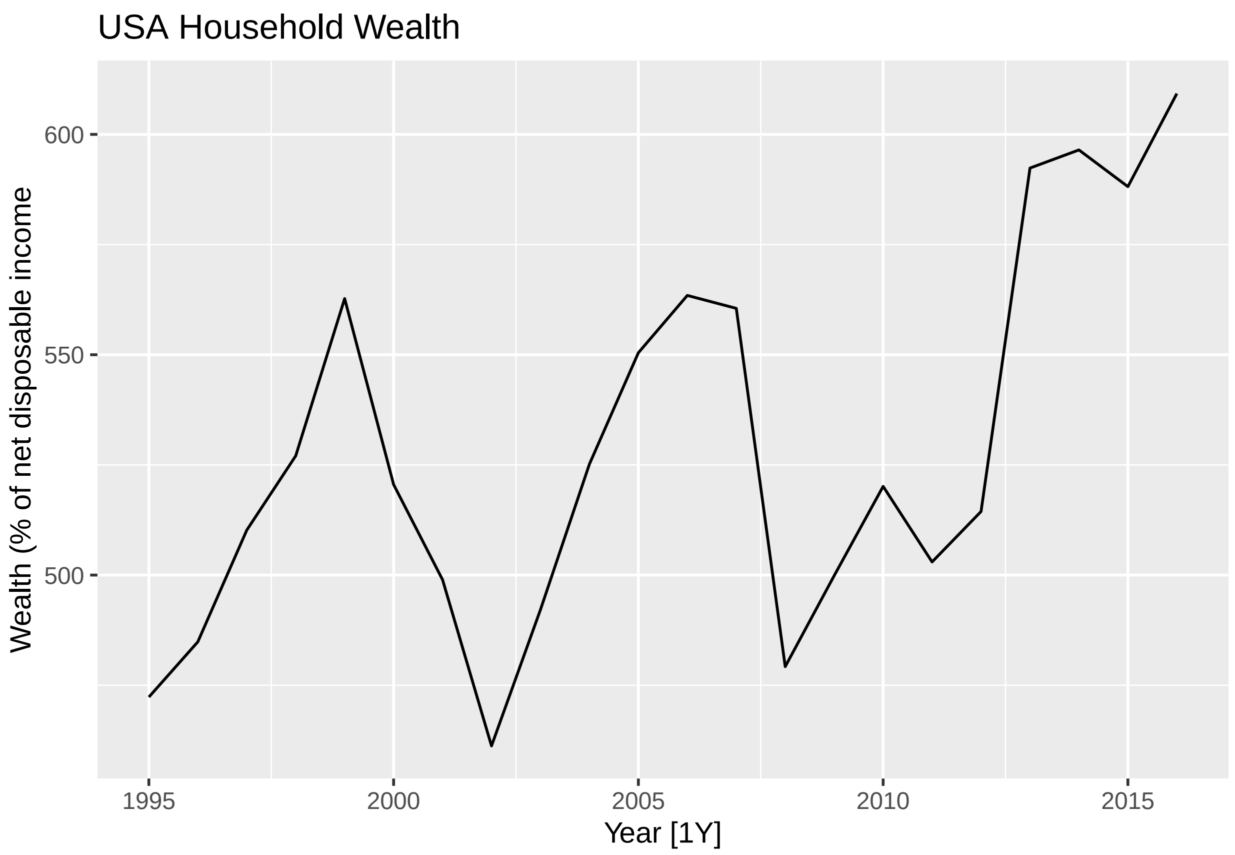
Similarly to the previous example, we see apparent seasonality without a clear trend, meaning the `SNAIVE` methodology should be implemented:

```
aus_livestock %>%
  filter(State == 'New South Wales' & Animal == 'Lambs') %>%
    model(SNAIVE(Count)) %>%
      forecast(h = '10 years') %>%
        autoplot(aus_livestock) +
          labs(
            title = 'Forecast: AUS Lamb Production'
          )
```



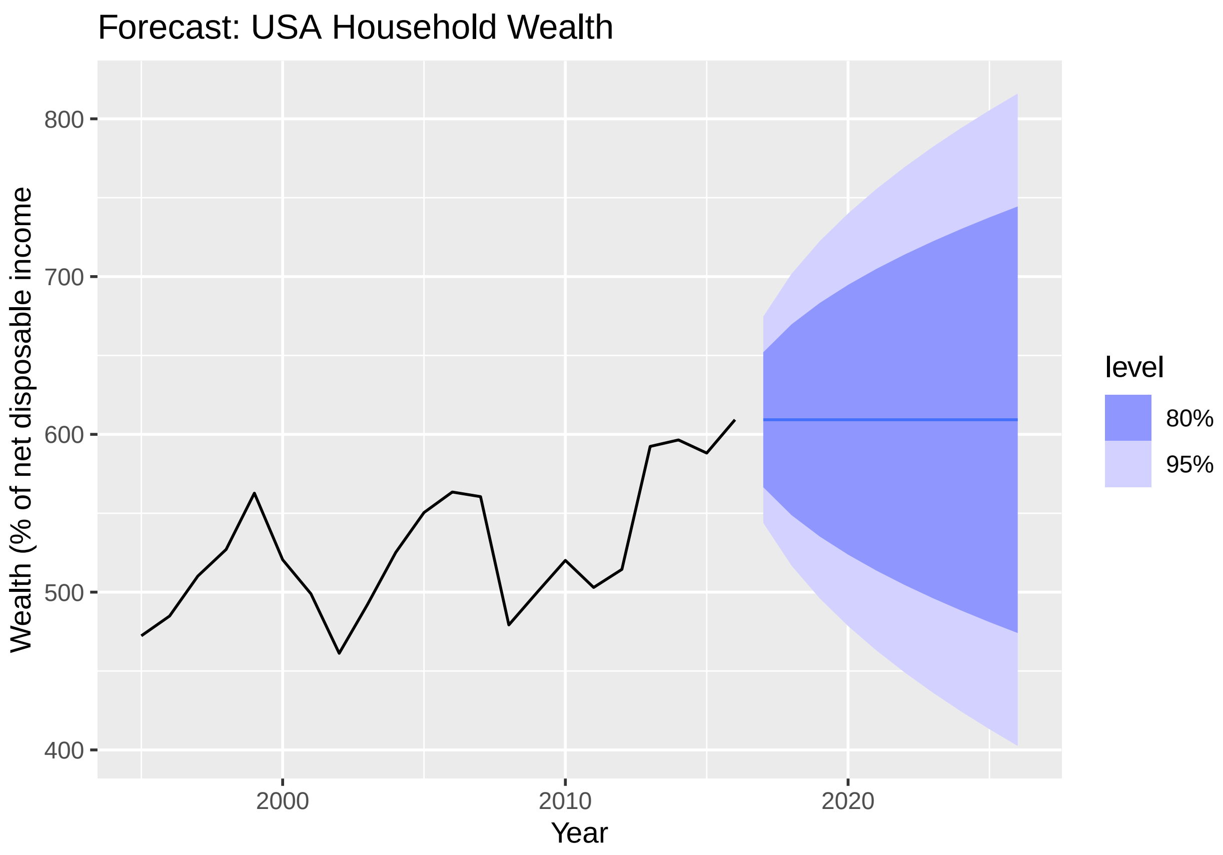
The cell below plots a time series of US household wealth using the `hh_budget` dataset:

```
hh_budget %>%
  filter(Country == 'USA') %>%
    autoplot(Wealth) +
      labs(
        title = 'USA Household Wealth',
        y = 'Wealth (% of net disposable income'
      )
```



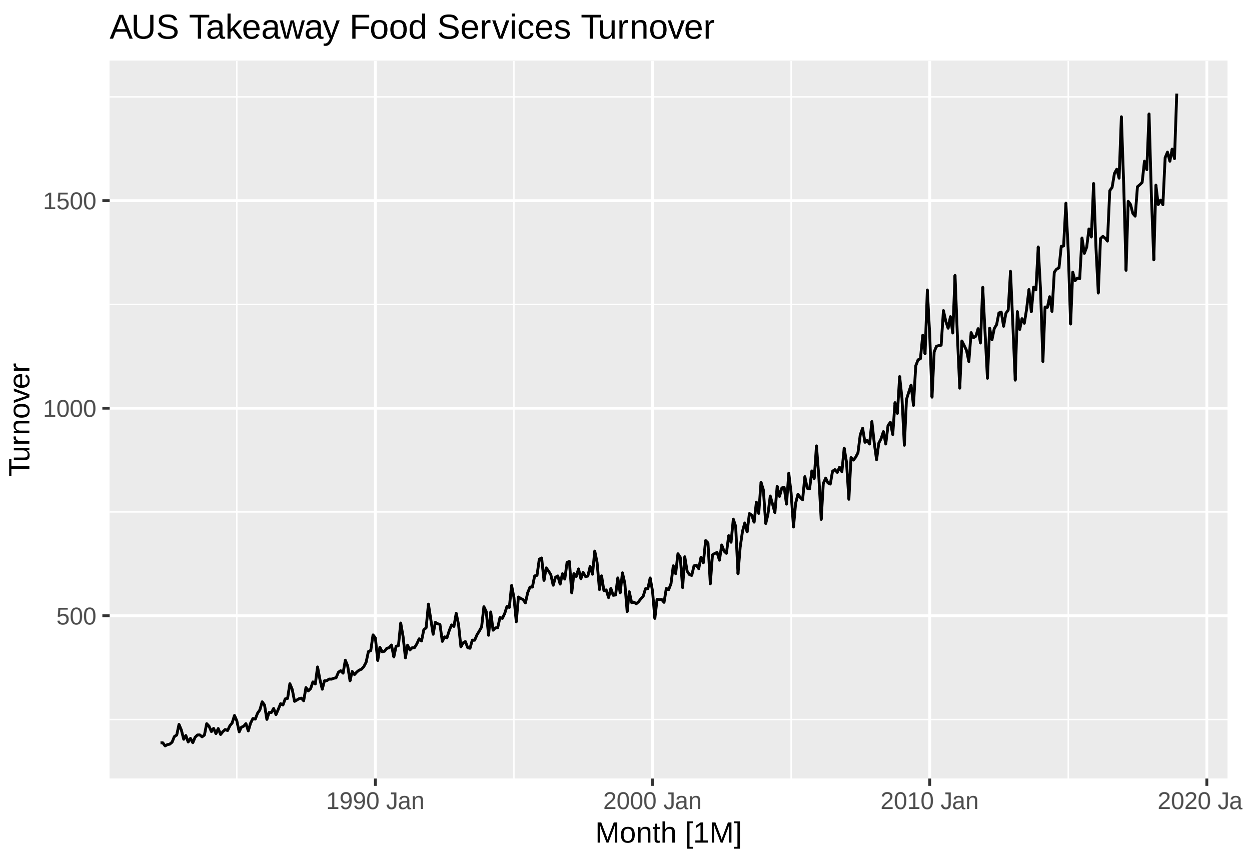
Here, there appears to be neither a clear trend nor seasonality. As such, we can make a forecast using the `NAIVE` methodology:

```
hh_budget %>%
  filter(Country == 'USA' & Year < 2017) %>%
    model(NAIVE(Wealth)) %>%
      forecast(h = '10 years') %>%
        autoplot(hh_budget) +
          labs(
            title = 'Forecast: USA Household Wealth',
            y = 'Wealth (% of net disposable income'
      )
```



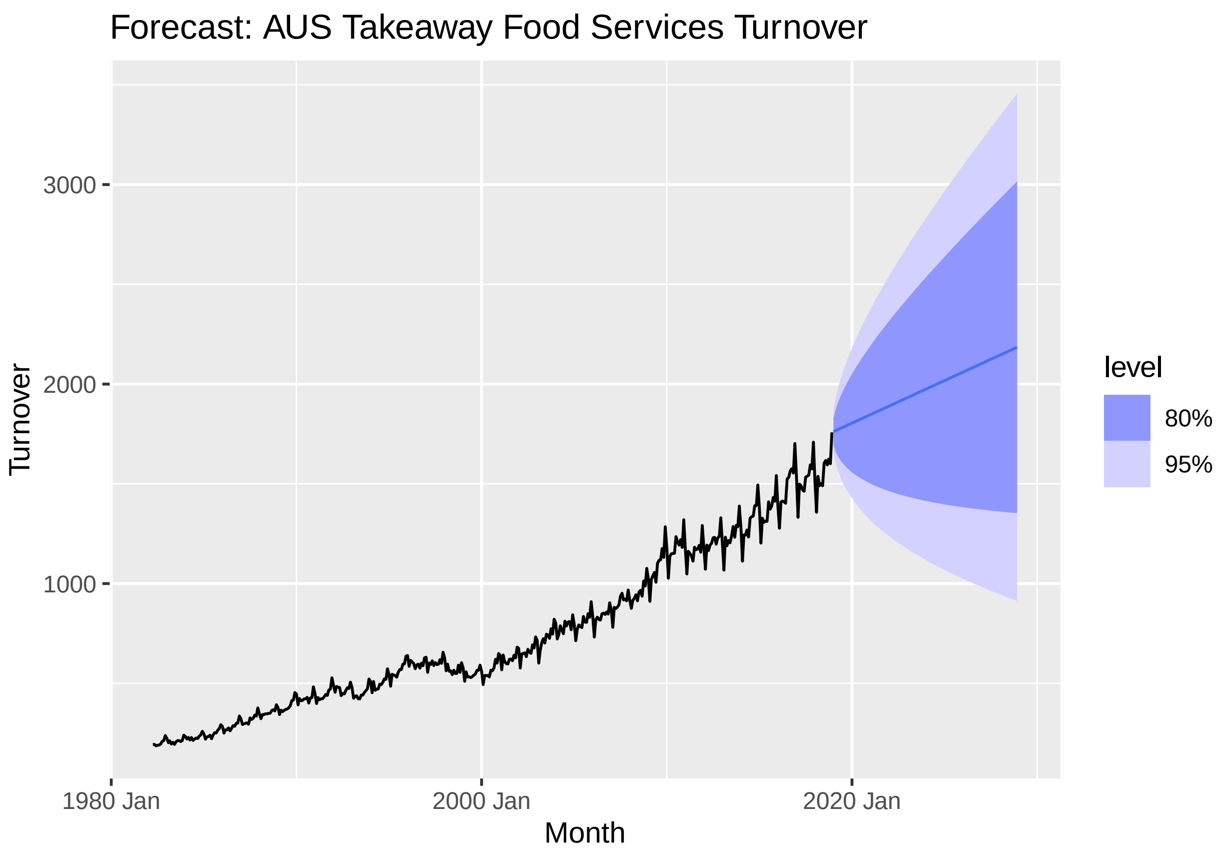
Lastly, the cell below creates a plot of turnover in the Australian takeaway food services industry:

```
aus_retail %>%
  filter(Industry == 'Takeaway food services') %>%
    summarise(Turnover = sum(Turnover)) %>%
      autoplot(Turnover) +
        labs(
          title = 'AUS Takeaway Food Services Turnover'
        )
```



This plot exhibits both seasonality and trend, meaning that a forecast made using any of the `NAIVE`, `SNAIVE`, or `drift` methodologies will not fully capture the behavior of the time series. However, given that the trend is much more prominent than the seasonality we will use the `drift` methodology to make a forecast:

```
aus_takeaway <- aus_retail %>%
  filter(Industry == 'Takeaway food services') %>%
    summarise(Turnover = sum(Turnover))

aus_takeaway %>%
  model(RW(Turnover ~ drift())) %>%
    forecast(h = '10 years') %>%
      autoplot(aus_takeaway) +
        labs(
          title = 'Forecast: AUS Takeaway Food Services Turnover'
        )
```



Exercise 5.2

Description

Use the Facebook stock price (data set `gafa_stock`) to do the following:

a. Produce a time plot of the series.
b. Produce forecasts using the drift method and plot them.
c. Show that the forecasts are identical to extending the line drawn between the first and last observations.
d. Try using some of the other benchmark functions to forecast the same data set. Which do you think is best? Why?

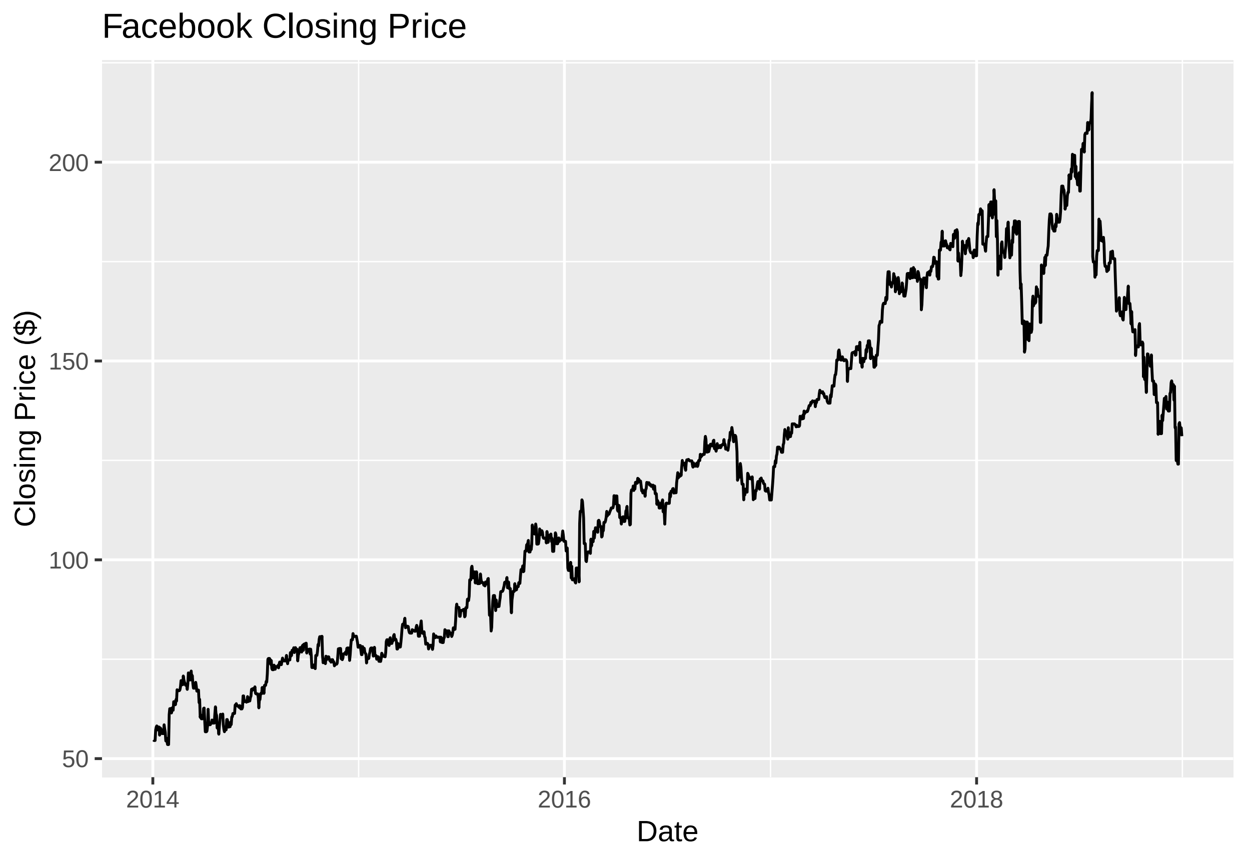
Solution

The cell below creates a `tsibble` named `fb_close`, which represents a time series of Facebook's closing stock price over time. To ensure accurate forecasting, it's essential to perform several data cleaning steps, especially focusing on addressing the gaps in the `gafa_stock` dataset, which correspond to days when the stock market was closed. To account for this, rows for each missing day are added and imputed with the most recent existing closing price.

```r
fb_close <- gafa_stock %>%
  filter(Symbol == 'FB') %>%
    select(Date, Close) %>%
      tidyr::complete(Date = seq(min(Date), max(Date), by = "day")) %>%
        fill(Close, .direction = "down") %>%
          as_tsibble(index = Date)
```

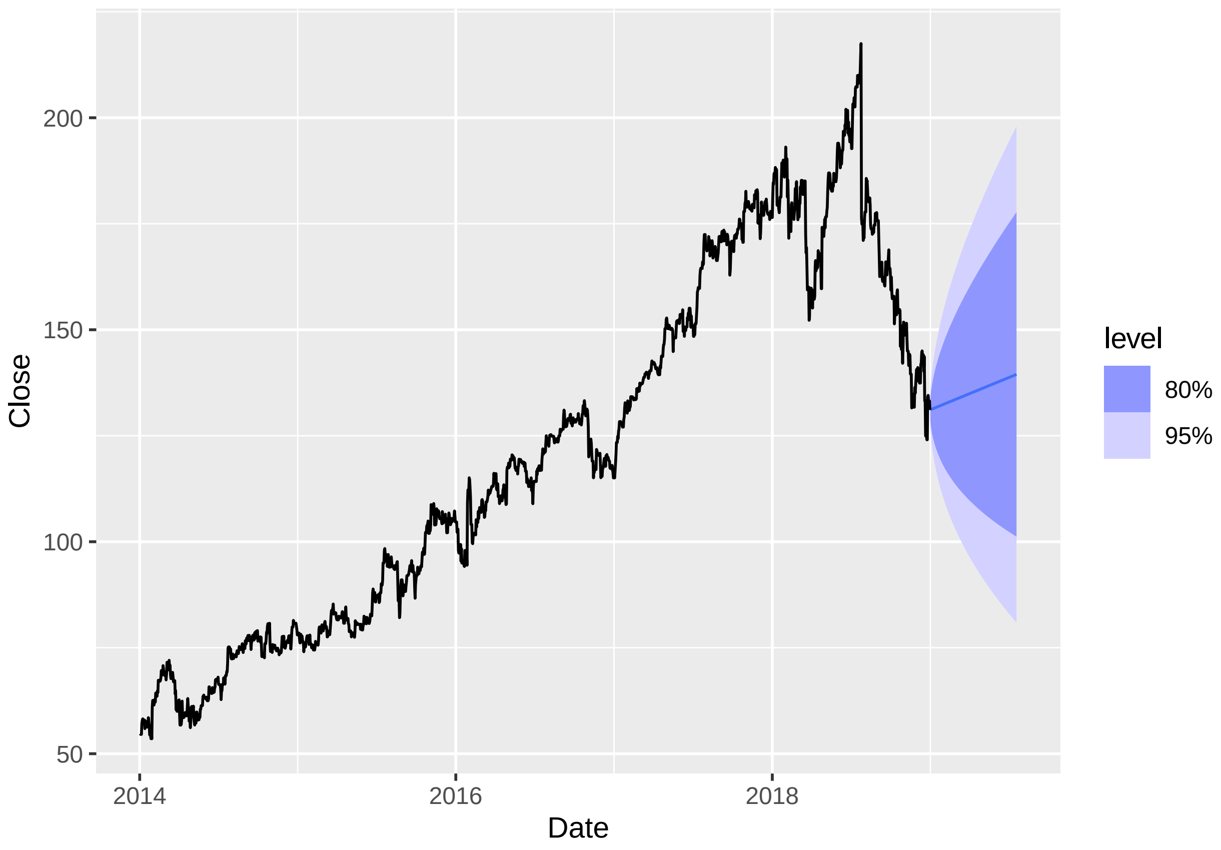
Now that the data has been cleaned, we can plot the time series:

```
fb_close %>%
  autoplot(Close) +
    labs(
      title = "Facebook Closing Price",
      x = 'Date',
      y = 'Closing Price ($)'
    )
```



Using the `drift()` methodology, the cell below creates a 200 day forecast of Facebook closing stock price:

```
mod <- fb_close |> model(RW(Close ~ drift()))
mod %>%
  forecast(h = '200 days') %>%
    autoplot(fb_close)
```



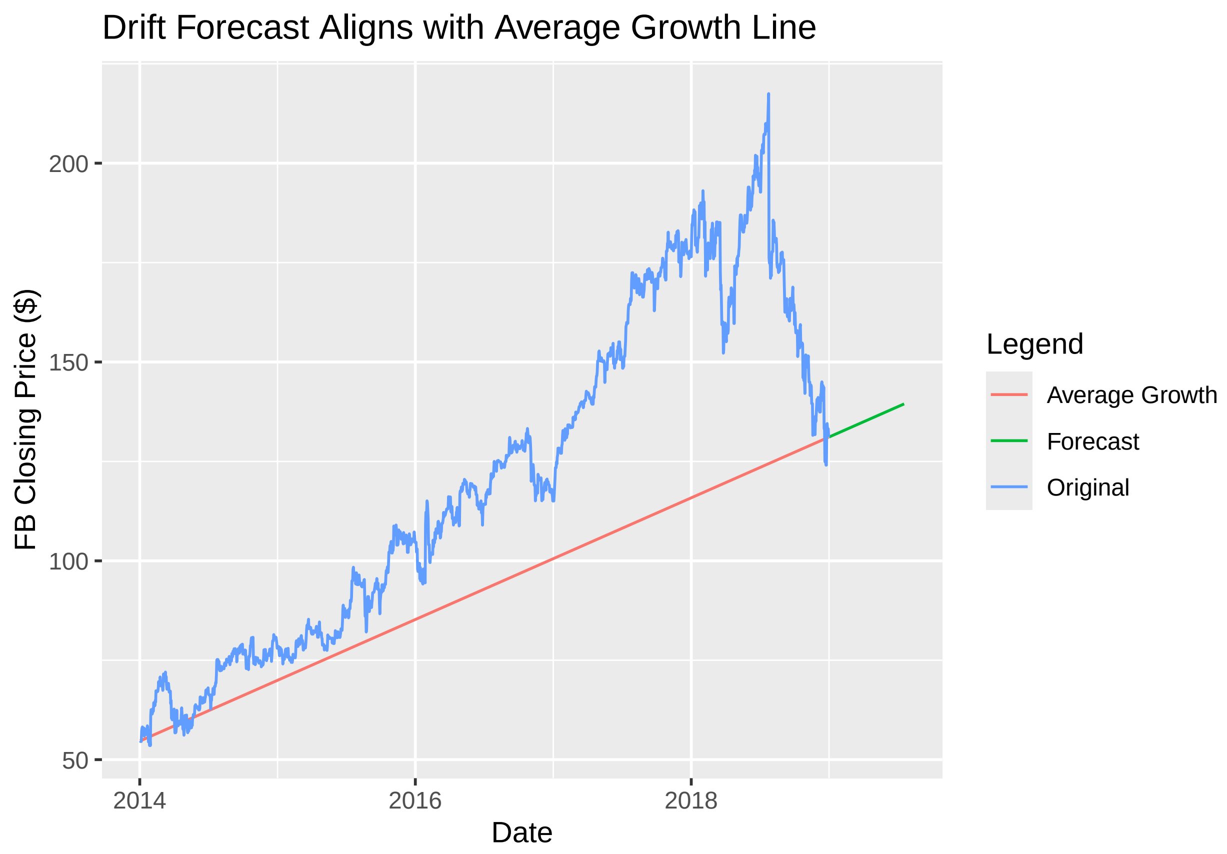
The `drift()` forecasting method calculates the average rate of growth across all data points and generates a forecast by subsequently adding this rate to the last observed value. In other words, this methodology is equivalent to extending a line that connects the first and last observed value. We can confirm this by first calculating the equation for that line:

```
# get the x and y values of the first and last observation
x_1 <- pull(fb_close[1,1])[1]
y_1 <- pull(fb_close[1,2])[1]
x_2 <- pull(fb_close[dim(fb_close[1]), 1])[1]
y_2 <- pull(fb_close[dim(fb_close[2]), 2])[1]

# calculate the slope of the line between the first and last observation
m = (y_2 - y_1) / as.numeric(x_2 - x_1)

# use the slope to generate a list of values for the average growth line
xs = seq(0, dim(fb_close)[1]-1)
ys = (m * xs) + y_1

# add the values to the original data
fb_close$ys <- ys
```

and then plotting it alongside the forecast:

```
# turn forecast into a tsibble that can be appended to fb_close
forecast <- mod %>%
  forecast(h = '200 days') %>%
    as_tsibble(index=Date) %>%
      select(Date, .mean)
```

```r
# append fb_close and the forecasted data
plt_data <- fb_close %>%
  bind_rows(forecast) %>%
    rename(
      Original = Close,
      Forecast = .mean,
      "Average Growth" = ys
    ) %>%
      # pivot the data so it can be shown on a single plot
      pivot_longer(c('Original', 'Forecast', 'Average Growth'),
                   names_to = "Legend")

# plot the original, average growth, and forecast data all in one plot
ggplot(plt_data, aes(x=Date, y=value, color=Legend)) + geom_line() +
  labs(
    title = "Drift Forecast Aligns with Average Growth Line",
    x = 'Date',
    y = 'FB Closing Price ($)'
  )
```



As you can see in the above plot, the forecast is simply a continuation of the average growth line, or the line that connects the first and last data point.

Exercise 5.3

Description

Apply a seasonal naïve method to the quarterly Australian beer production data from 1992. Check if the residuals look like white noise, and plot the forecasts. The following code will help.

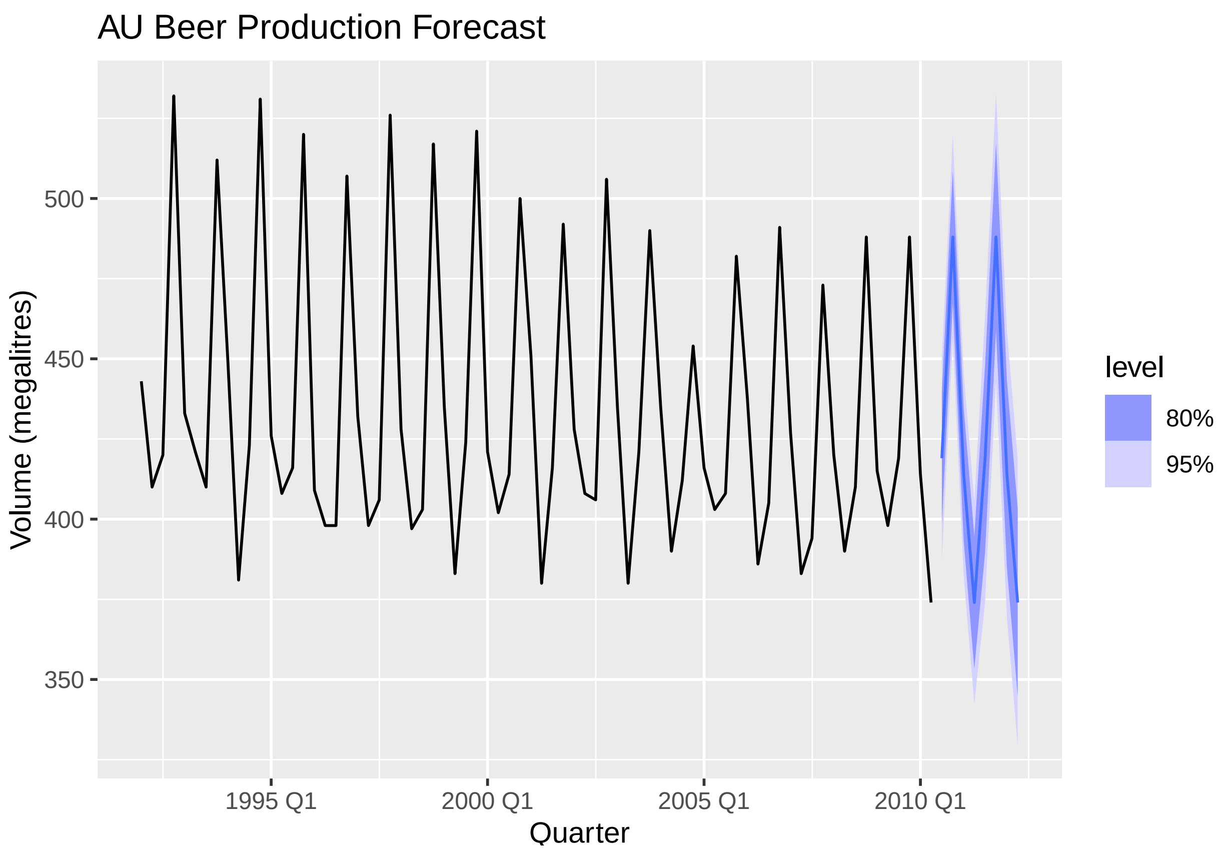
```
# Extract data of interest
recent_production <- aus_production |>
  filter(year(Quarter) >= 1992)
# Define and estimate a model
fit <- recent_production |> model(SNAIVE(Beer))
# Look at the residuals
fit |> gg_tsresiduals()
# Look a some forecasts
fit |> forecast() |> autoplot(recent_production)
```

What do you conclude?

Solution

The cell below extracts data from `aus_production` into a `tsibble` (`recent_production`), applies the SNAIVE() method, and generates a forecast plot::

```
recent_production <- aus_production |>
  filter(year(Quarter) >= 1992)

fit <- recent_production |> model(SNAIVE(Beer))

fit %>% forecast() %>%
  autoplot(recent_production) +
    labs(
      y = 'Volume (megalitres)',
      title = 'AU Beer Production Forecast'
    )
```



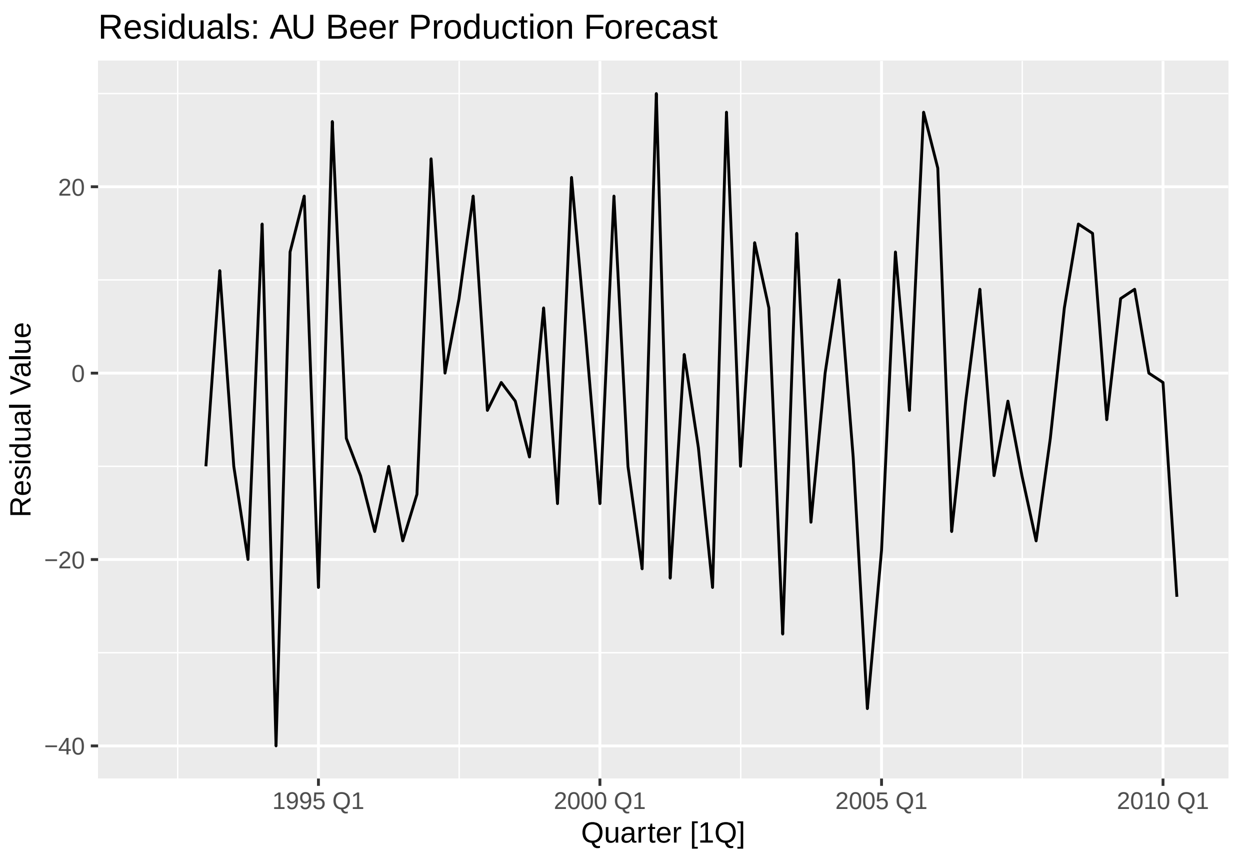
It is evident from the plot that the forecast made by `SNAIVE()` methodology has taken into account the apparent seasonality of the observed data. To assess the performance of the `SNAIVE()` method, we can evaluate the innovation residuals (since the original data was not transformed in any way, the innovation residuals in this case are equivalent to the residuals). More specifically, forecasting methods should produce innovation residuals with the following characteristics:

1. The innovation residuals are uncorrelated. If there are correlations between innovation residuals, then there is information left in the residuals which should be used in computing forecasts.
2. The innovation residuals have zero mean. If they have a mean other than zero, then the forecasts are biased.
3. The innovation residuals have constant variance. This is known as "homoscedasticity".
4. The innovation residuals are normally distributed.

First, the cell below plots the residuals:

```r
aug <- fit %>% augment()
autoplot(aug, .innov) +
  labs(
    y = "Residual Value",
    title = "Residuals: AU Beer Production Forecast"
  )
```

```
## Warning: Removed 4 rows containing missing values or values outside the scale range
## (`geom_line()`).
```



We can see that the mean of the residuals appears to be close to 0, which we can confirm via direct calculation:

```
mean(aug$.innov[!is.na(aug$.innov)])
```

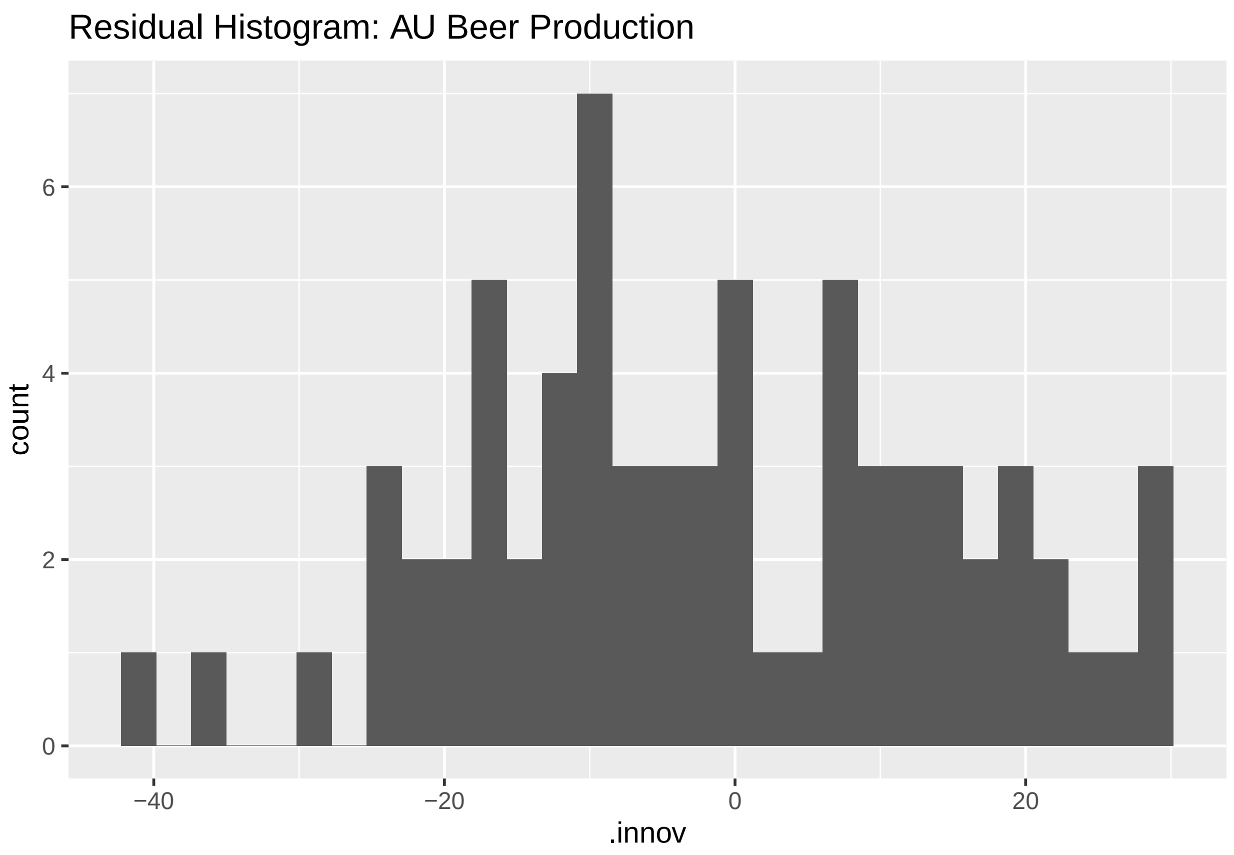
```
## [1] -1.571429
```

A histogram of the residuals confirms that they do appear to be close to normally distributed:

```
aug %>%
  ggplot(aes(x = .innov)) +
  geom_histogram() +
  labs(
    title = "Residual Histogram: AU Beer Production"
  )
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

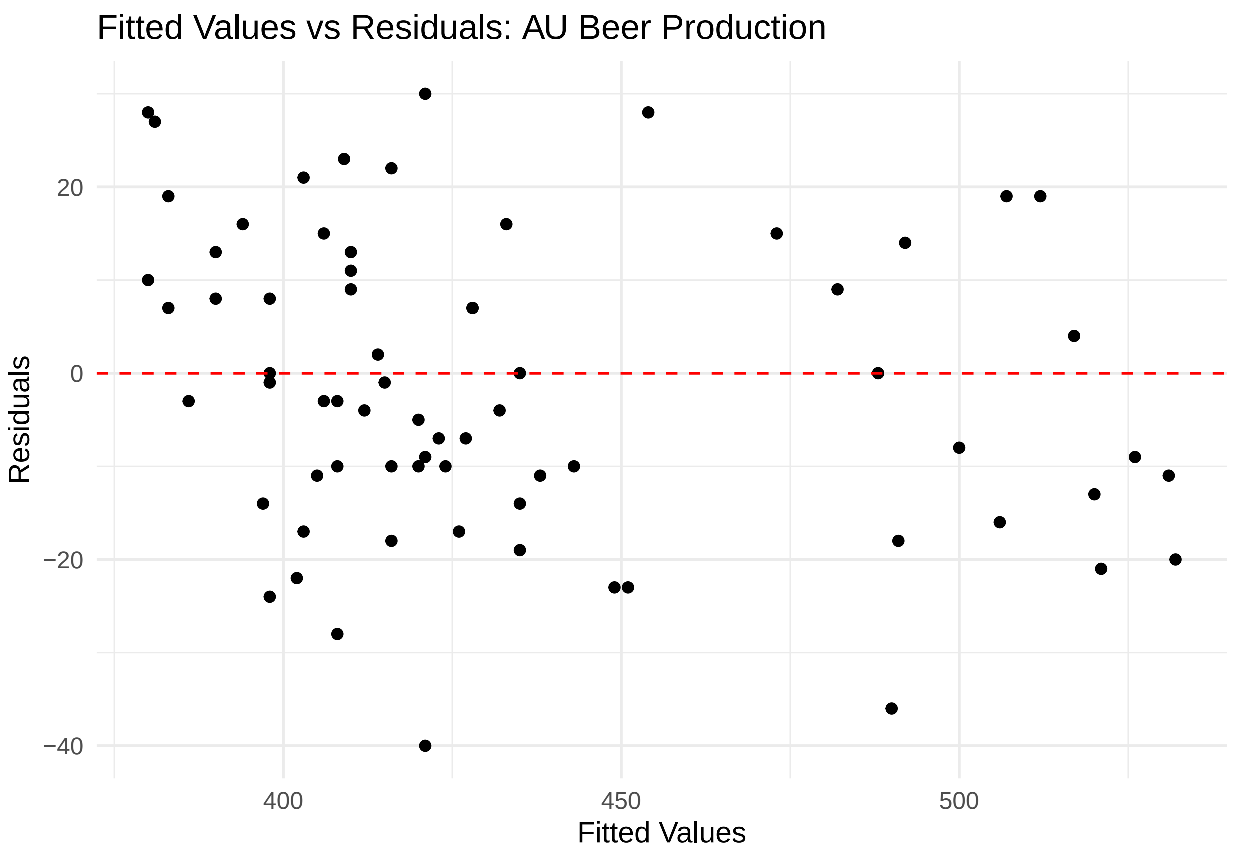
```
## Warning: Removed 4 rows containing non-finite outside the scale range
## (`stat_bin()`).
```



Plotting the fitted values of the model against the residuals confirms that they are homoscedastic:

```r
ggplot(aug, aes(x = .fitted, y = .resid)) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
  labs(title = "Fitted Values vs Residuals: AU Beer Production",
       x = "Fitted Values",
       y = "Residuals") +
  theme_minimal()
```

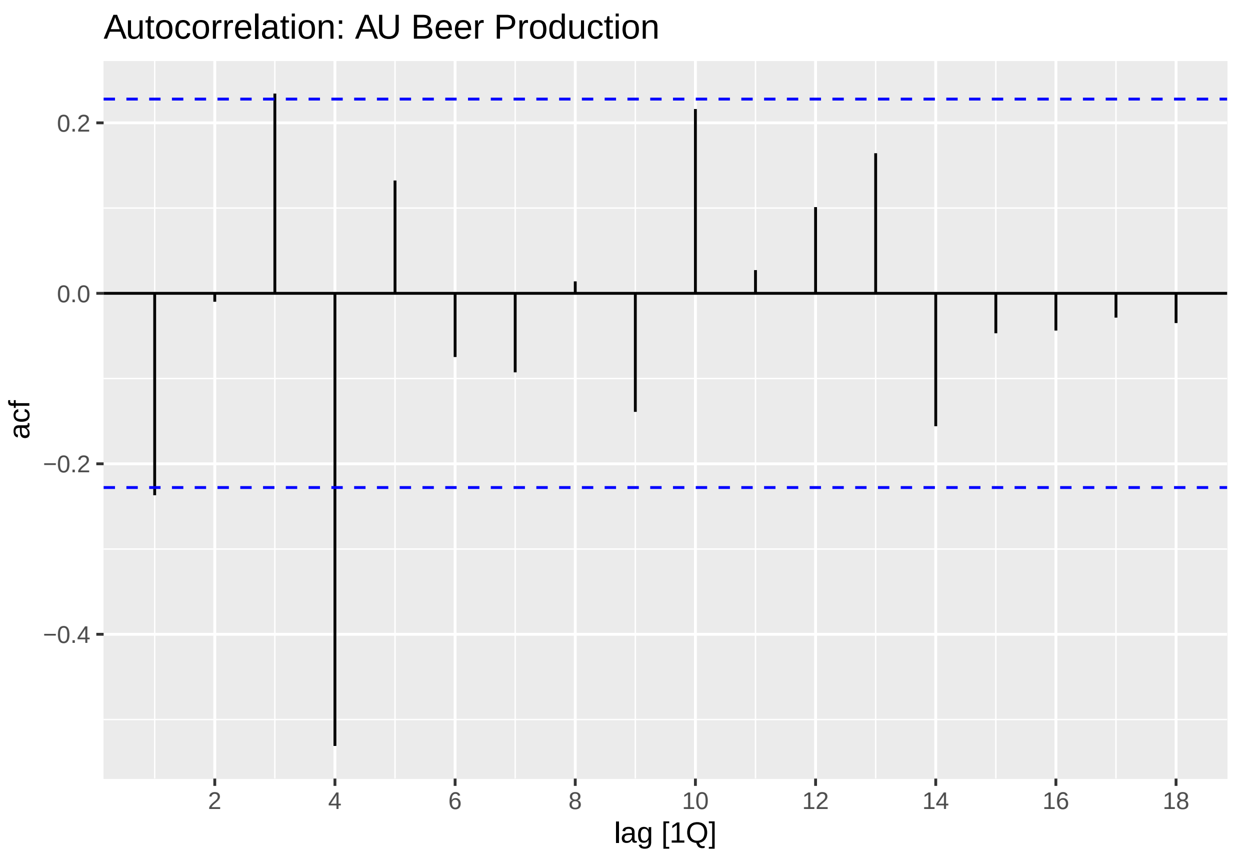
```
## Warning: Removed 4 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



In other words, the spread of the residuals appears consistent across the range of fitted values.

While the above plot suggests the residuals are uncorrelated, this is more clearly confirmed by an ACF plot:

```r
aug |>
  ACF(.innov) |>
  autoplot() +
  labs(title = "Autocorrelation: AU Beer Production")
```



An ACF or auto-correlation function plot shows if the time series correlates with any lagged version of itself. At a 4-quarter lag, the autocorrelation slightly exceeds the significance bounds, but it is only notably problematic for one value, which is minor enough that it can probably be disregarded.

All in all, the residual analysis indicate that the `SNAIVE()` forecasting methodology performed quite well with the data provided.

Exercise 5.4

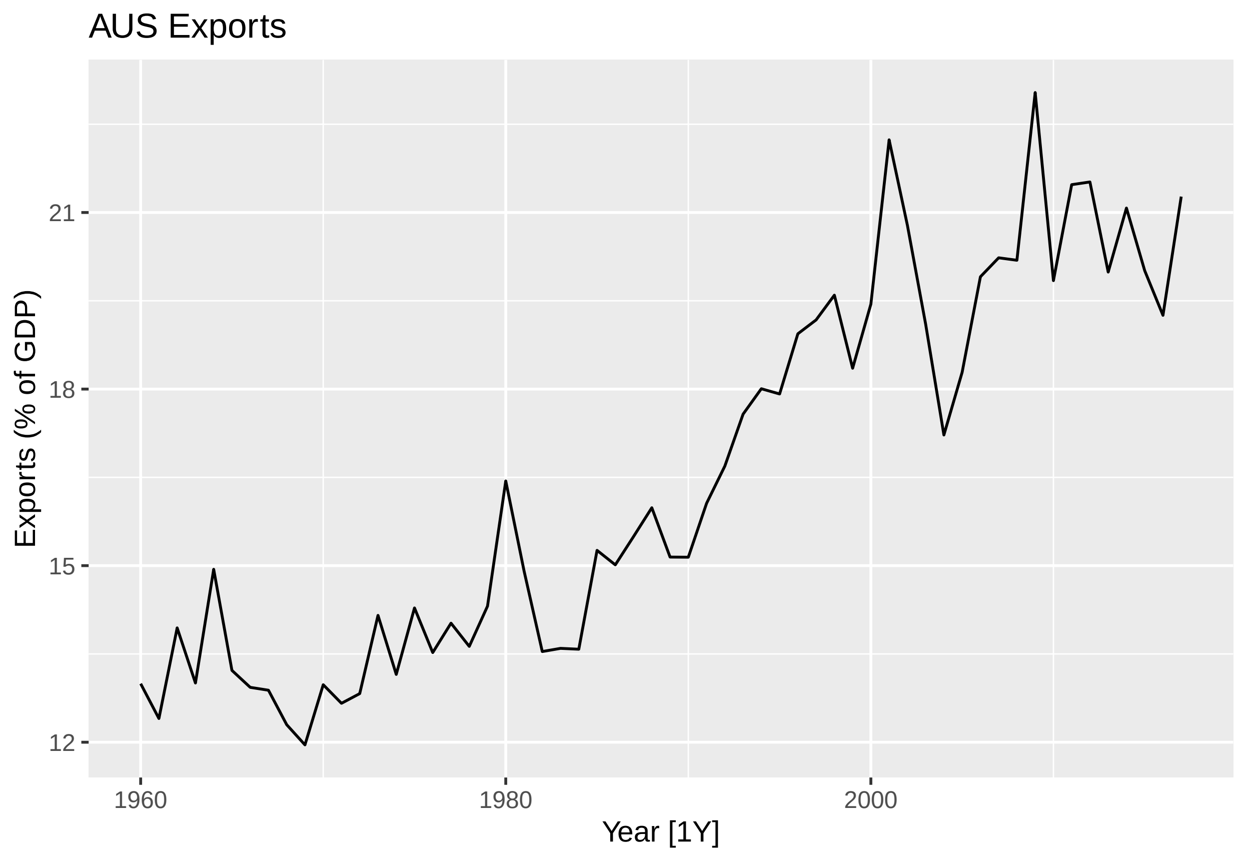
Description

Repeat the previous exercise using the Australian Exports series from `global_economy` and the Bricks series from `aus_production`. Use whichever of `NAIVE()` or `SNAIVE()` is more appropriate in each case.

Solution

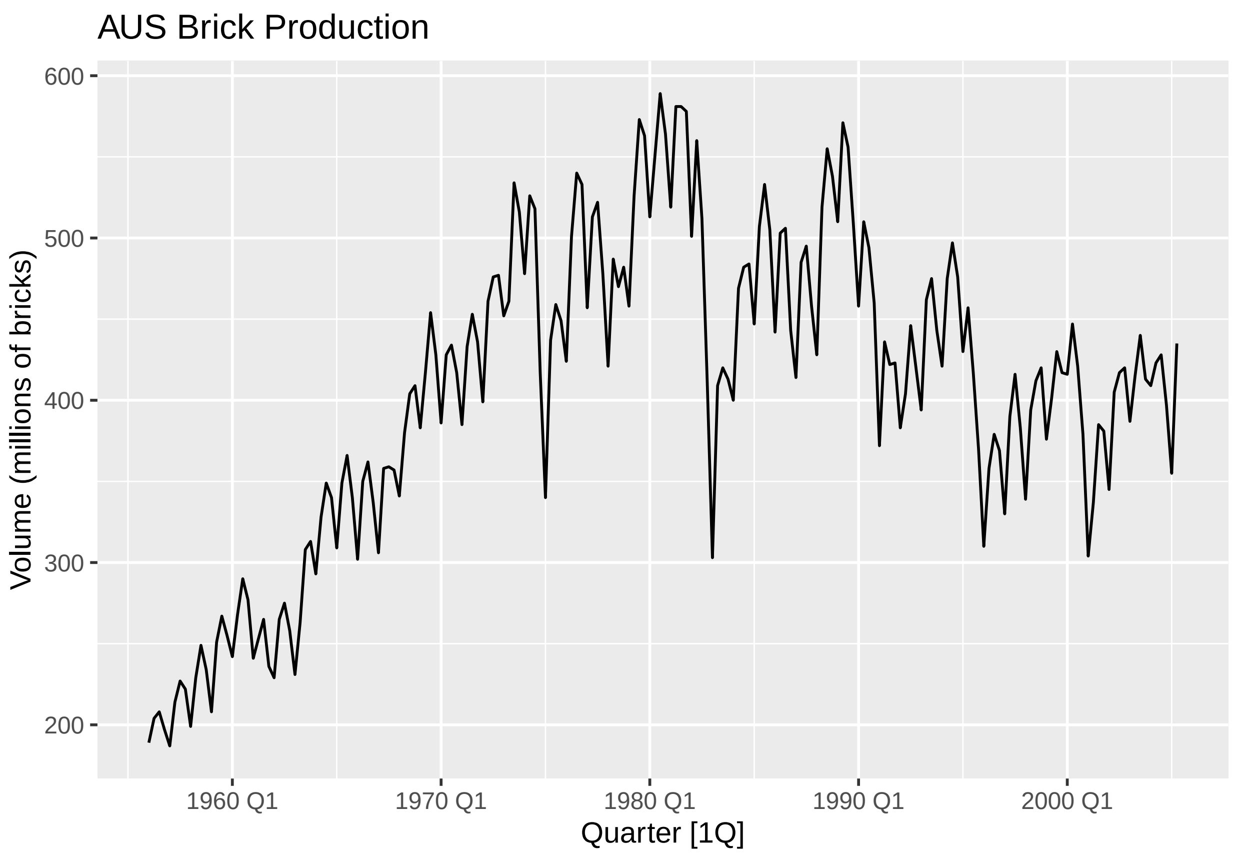
First, the cells below create time series plots of each series to see whether the `NAIVE` or `SNAIVE` forecasting methodology is more appropriate. First, AUS annual exports:

```
aus_exports <- global_economy %>%
  filter(Code == 'AUS') %>%
    select(Year, Exports)

aus_exports %>%
  autoplot(Exports) +
    labs(
      y = 'Exports (% of GDP)',
      title = 'AUS Exports'
    )
```



The plot above shows no apparent seasonality, meaning that a `NAIVE()` forecast model is more appropriate. However, when plotting annual AUS brick production:

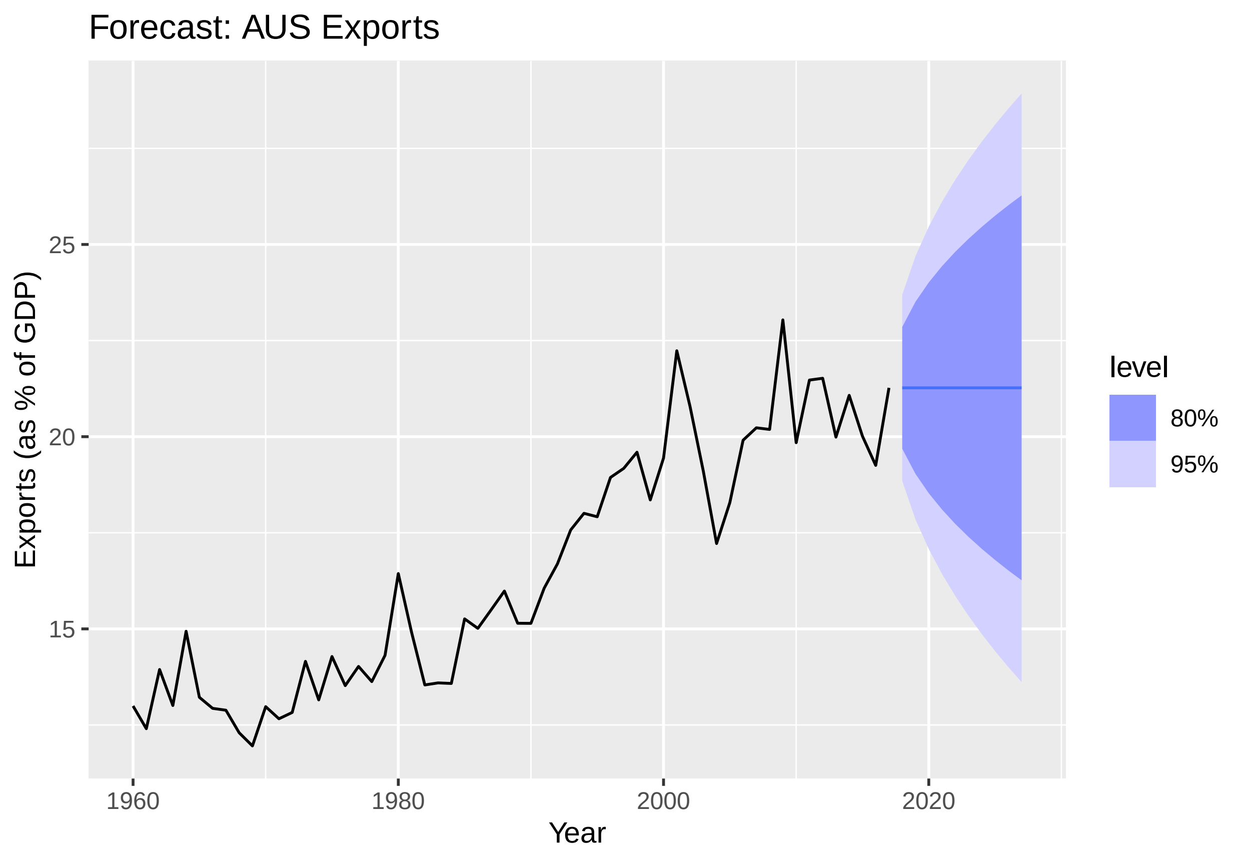
```
aus_bricks <- aus_production %>%
    # NA value after 2005Q2 that need to be removed
    filter(Quarter < yearquarter('2005 Q3')) %>%
      select(Quarter, Bricks)

aus_bricks %>%
  autoplot(Bricks) +
    labs(
      y = 'Volume (millions of bricks)',
      title = 'AUS Brick Production'
    )
```



we see that there is a repeating pattern over time. As such, forecasts the `SNAIVE()` forecasting model is more appropriate in this case.

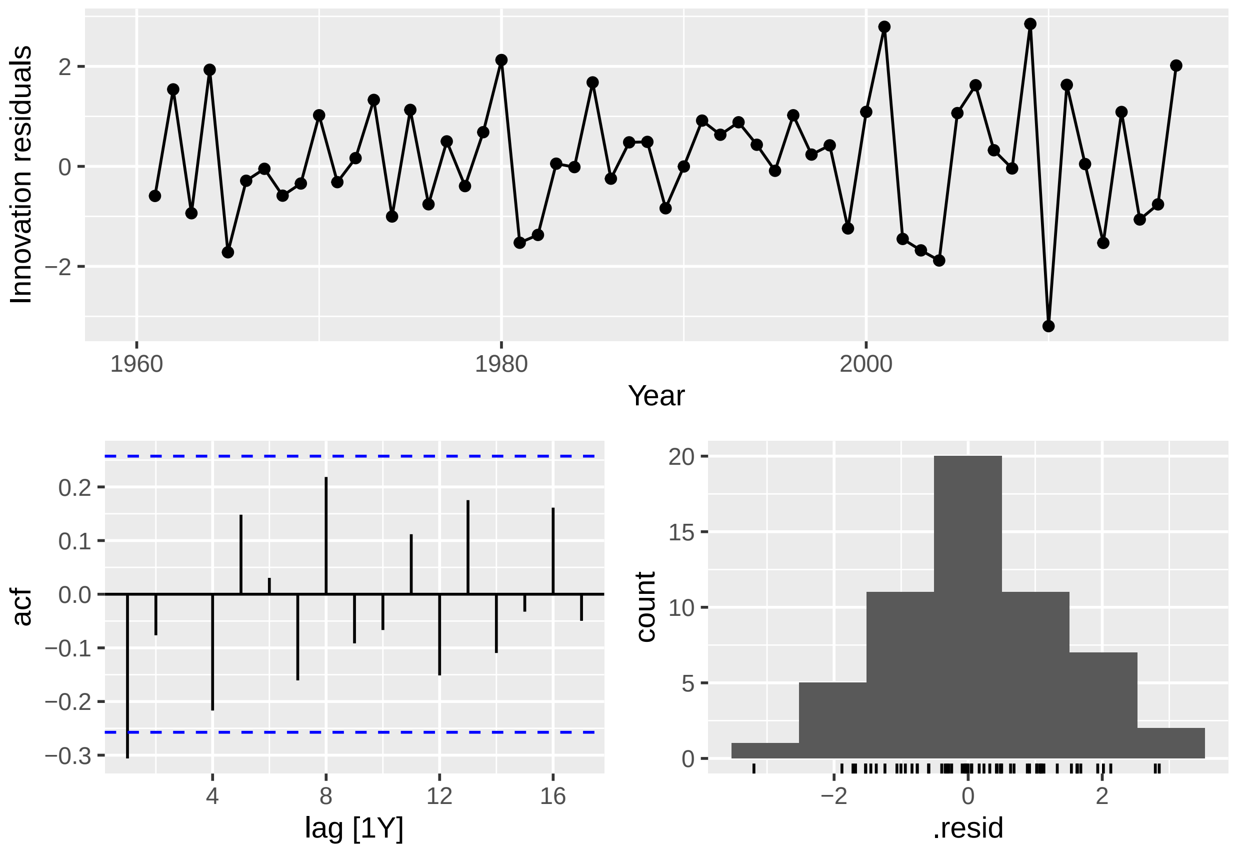
The `NAIVE()` forecast of the `aus_exports` data is shown below:

```r
fit <- aus_exports |> model(NAIVE(Exports))

fit %>% forecast(h='10 years') %>%
  autoplot(aus_exports) +
    labs(
      y = 'Exports (as % of GDP)',
      title = 'Forecast: AUS Exports'
    )
```



This time, we use the `gg_tsresiduals()` functions to produce all the residual figures at once:

```
fit |> gg_tsresiduals()
```



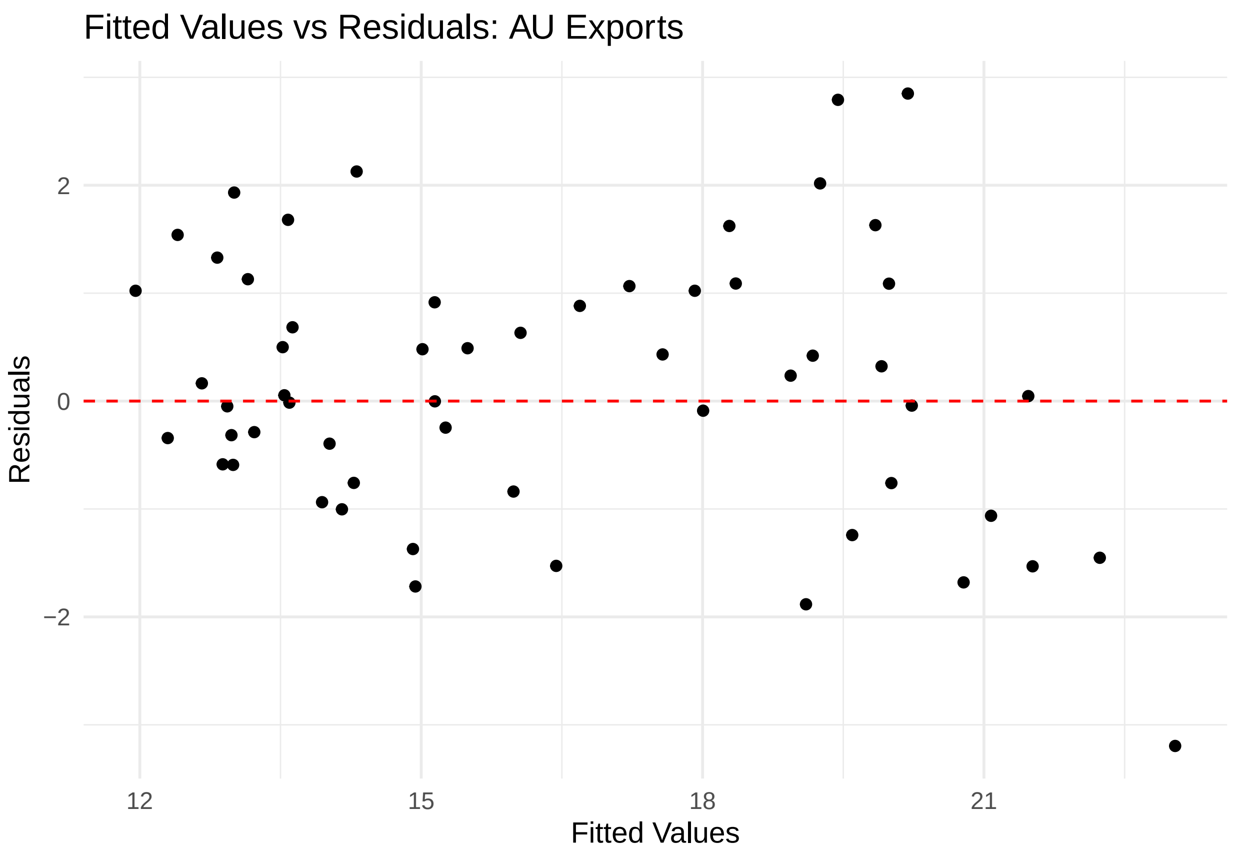
The plots above provide evidence that our residuals meet three of the four essential criteria required to validate the choice of an appropriate forecasting method:

1. The residual values appear equally distributed above and below 0, indicating that it is close to their mean value.
2. The histogram is a near perfect picture of a normal distribution.
3. The residuals do not correlate with themselves for any lag value.

To prove that the residuals exhibit homoscedasticity, we can plot the residuals against their fitted values:

```
aug <- fit %>% augment()
ggplot(aug, aes(x = .fitted, y = .resid)) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
  labs(title = "Fitted Values vs Residuals: AU Exports",
       x = "Fitted Values",
       y = "Residuals") +
  theme_minimal()
```

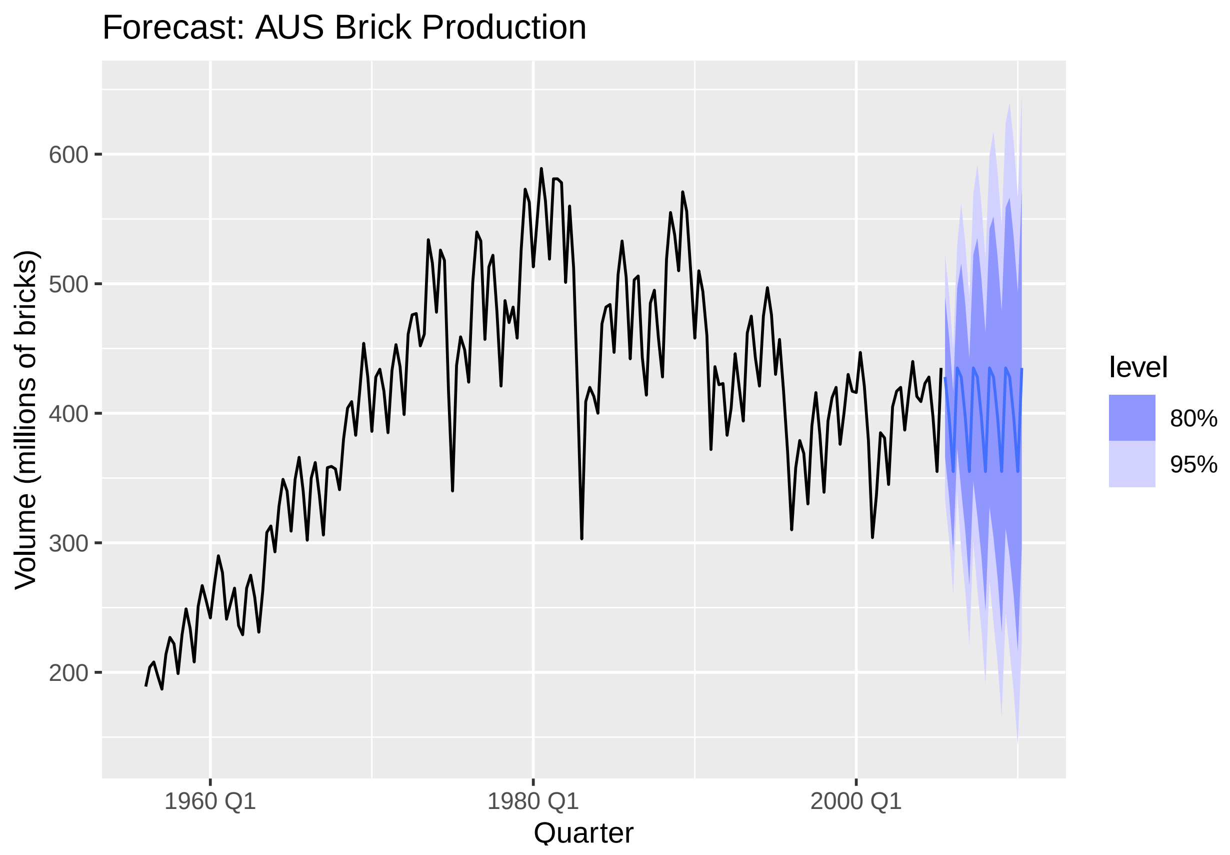
```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_point()`).
```



The distribution of the residuals does not vary much across the range of fitted values, indicating that we have met our fourth requirement.

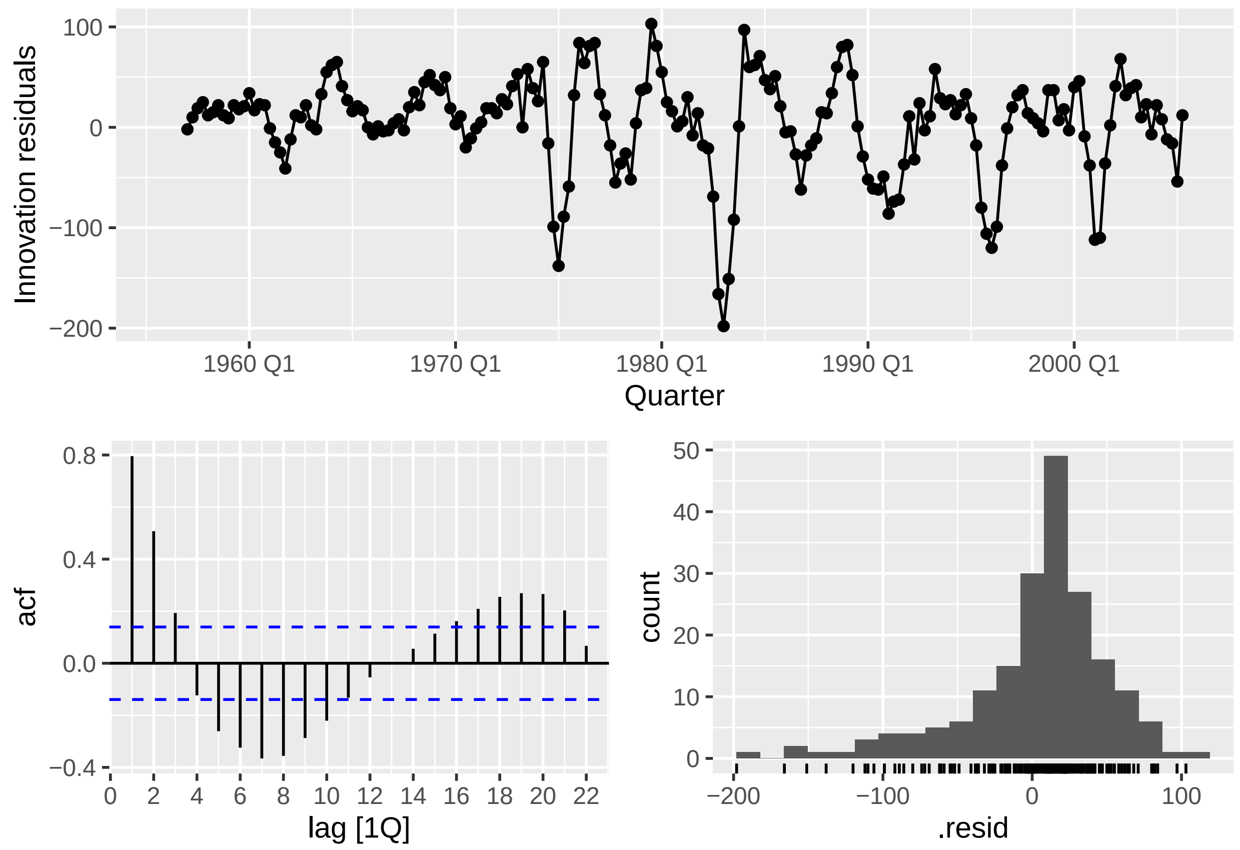
Next, the cell below creates the same set of figures for the `aus_bricks` dataset, this time using the `SNAIVE()` forecasting method:

```
fit <- aus_bricks |> model(SNAIVE(Bricks))

fit %>% forecast(h='5 years') %>%
  autoplot(aus_bricks) +
    labs(
      y = 'Volume (millions of bricks)',
      title = 'Forecast: AUS Brick Production'
    )
```



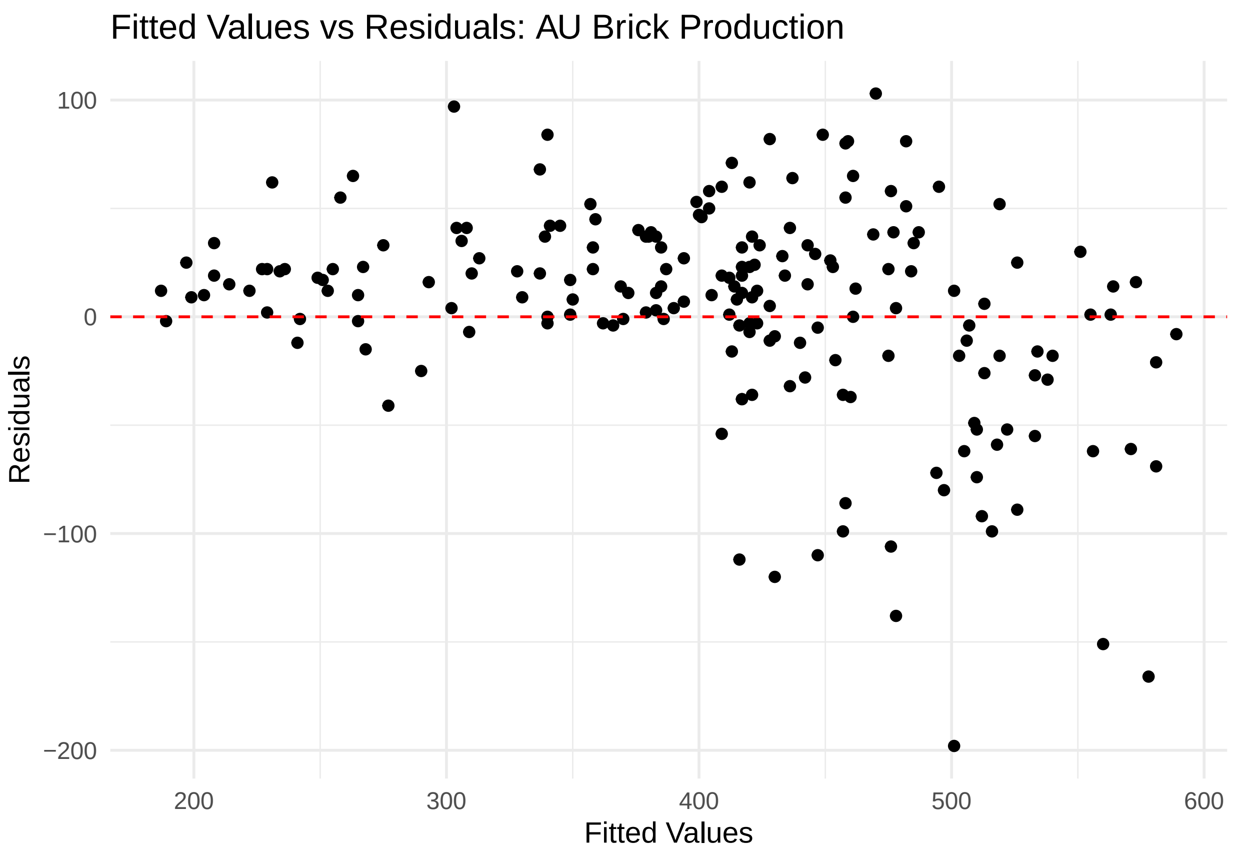
```
fit |> gg_tsresiduals()
```

```
## Warning: Removed 4 rows containing missing values or values outside the scale range
## (`geom_line()`).

## Warning: Removed 4 rows containing missing values or values outside the scale range
## (`geom_point()`).

## Warning: Removed 4 rows containing non-finite outside the scale range
## (`stat_bin()`).
```



```r
aug <- fit %>% augment()
ggplot(aug, aes(x = .fitted, y = .resid)) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
  labs(title = "Fitted Values vs Residuals: AU Brick Production",
       x = "Fitted Values",
       y = "Residuals") +
  theme_minimal()
```

```
## Warning: Removed 4 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



The residual plots here present a stark contrast to the patterns revealed when applying the `NAIVE()` method to the `aus_exports` dataset. All four of the requirements are not met:

1. Many low residual values indicate that their mean is significantly below 0.
2. The distribution of residuals is greatly shifted to the left with negative skew (does not appear normal).
3. The series correlates with itself for many different lag periods.
4. The distribution of the residuals when plotted against their fitted values does not exhibit a constant variance throughout.

The reason for the failure of the forecast model in this case is likely due to the varying overall trend of the observations shifting multiple times over the time period.

Exercise 5.7

Description

For your retail time series (from Exercise 7 in Section 2.10):

a. Create a training dataset consisting of observations before 2011 using

```
myseries_train <- myseries |>
  filter(year(Month) < 2011)
```

b. Check that your data have been split appropriately by producing the following plot.

```
autoplot(myseries, Turnover) +
  autolayer(myseries_train, Turnover, colour = "red")
```

c.Fit a seasonal naïve model using `SNAIVE()` applied to your training data (`myseries_train`).

```
fit <- myseries_train |>
  model(SNAIVE())
```

d. Check the residuals.

```
fit |> gg_tsresiduals()
```

Do the residuals appear to be uncorrelated and normally distributed?

e. Produce forecasts for the test data

```
fc <- fit |>
  forecast(new_data = anti_join(myseries, myseries_train))
fc |> autoplot(myseries)
```

f. Compare the accuracy of your forecasts against the actual values.

```
fit |> accuracy()
fc |> accuracy(myseries)
```

g. How sensitive are the accuracy measures to the amount of training data used?

Solution

The cell below pulls the data referenced in the exercise description:

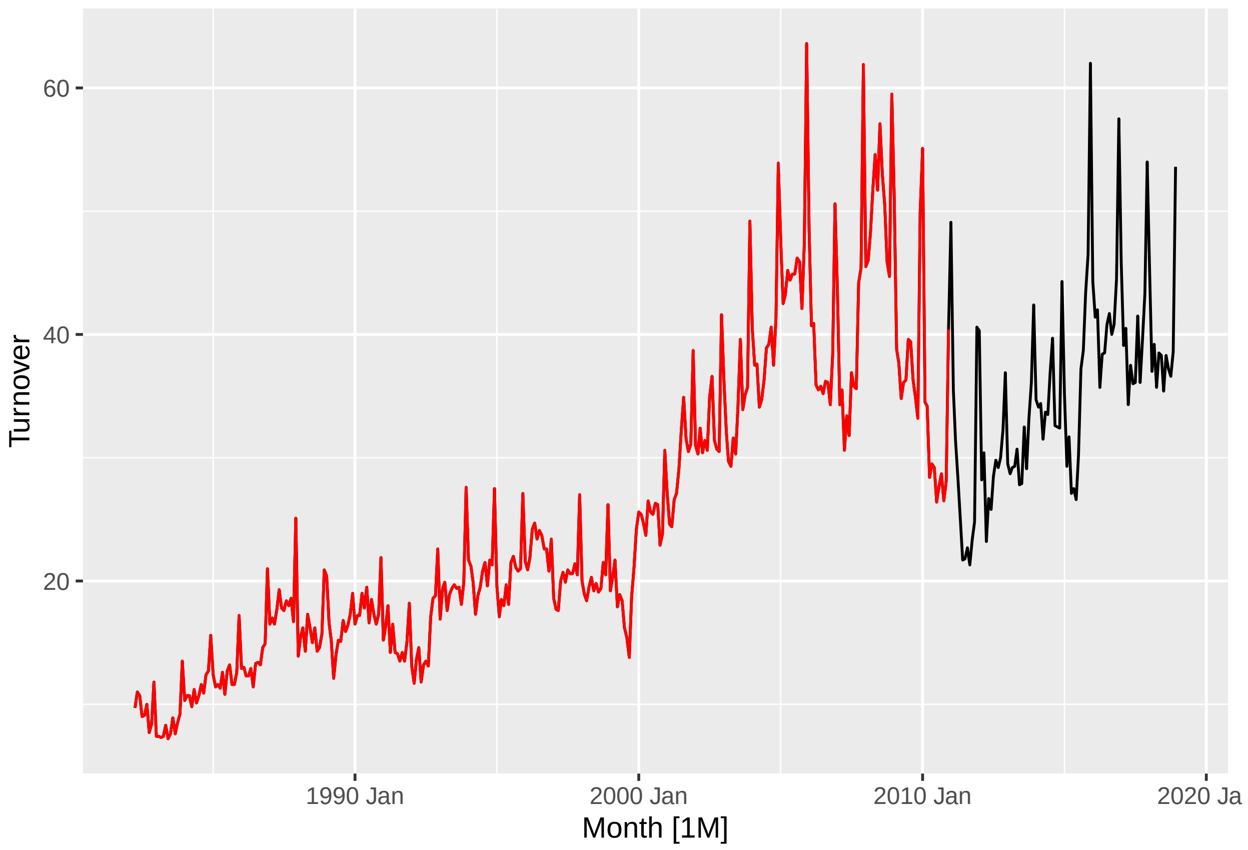
```
set.seed(42)
myseries <- aus_retail |>
  filter(`Series ID` == sample(aus_retail$`Series ID`,1))
```

Next, a training set produced that only includes those observations made before 2011:

```
myseries_train <- myseries |>
  filter(year(Month) < 2011)
```

The plot below reveals that the data has been properly split, with the training data only being visible up until 2011.

```
autoplot(myseries, Turnover) +
  autolayer(myseries_train, Turnover, colour = "red")
```



We can now train and produce `SNAIVE()` model using the training data:

```
fit <- myseries_train |>
  model(SNAIVE(Turnover))
```

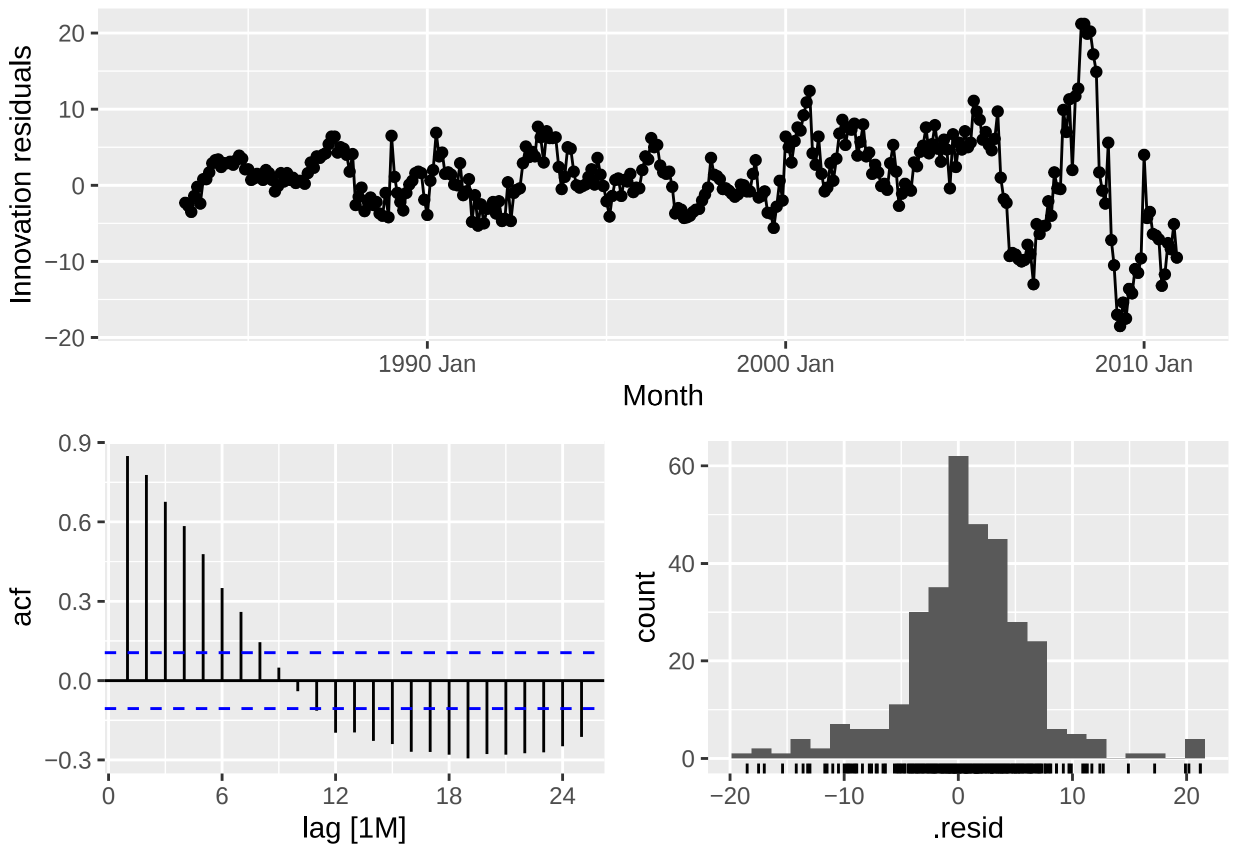
The cell below produces the residual plots needed to evaluate the quality of the forecasting model selection:

```
fit |> gg_tsresiduals()
```

```
## Warning: Removed 12 rows containing missing values or values outside the scale range
## (`geom_line()`).
```

```
## Warning: Removed 12 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

```
## Warning: Removed 12 rows containing non-finite outside the scale range
## (`stat_bin()`).
```



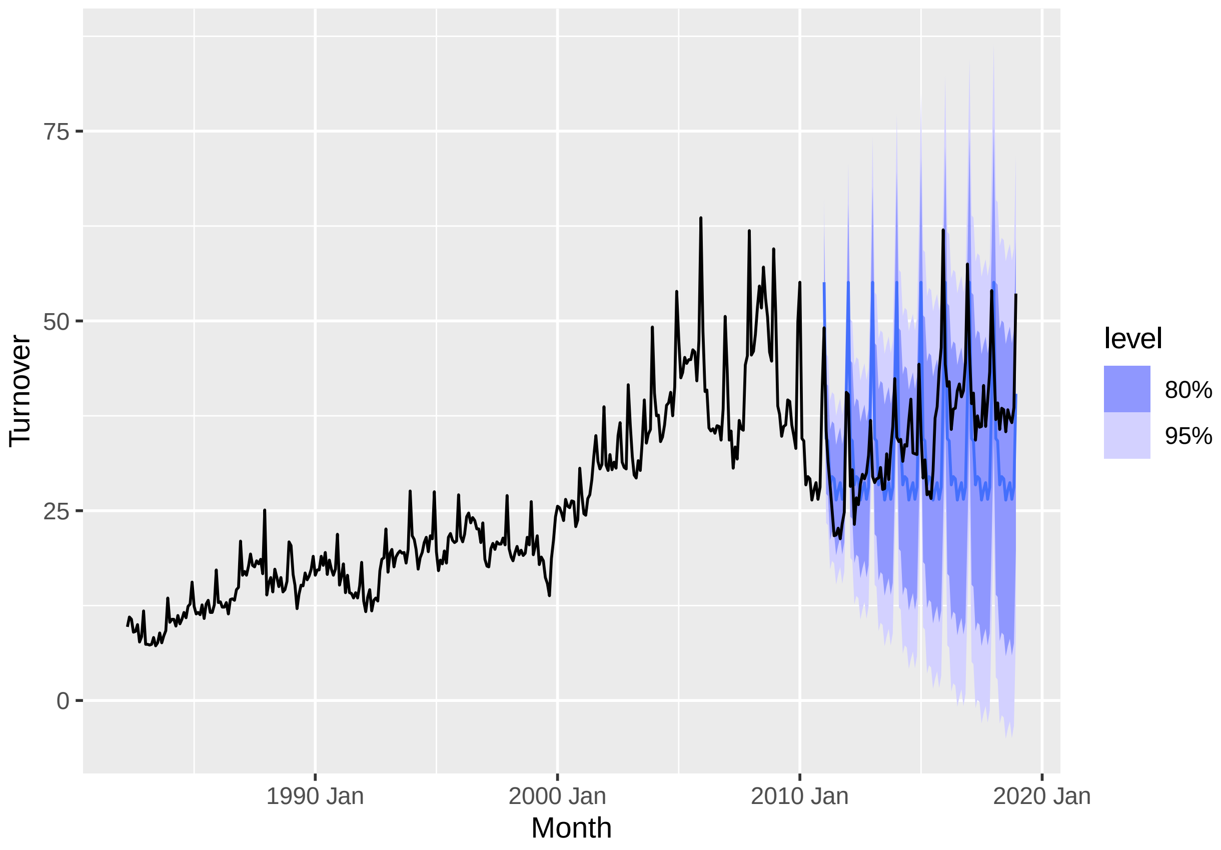
In this case, while the residuals appear close to having a normal distribution, but are most definitely correlated with themselves (the ACF plot is past the bounds of significance for almost all lag values).

Next, the cell below plots the forecast made using the training data against the actual observed values:

```r
fc <- fit |>
  forecast(new_data = anti_join(myseries, myseries_train))
```

```
## Joining with `by = join_by(State, Industry, `Series ID`, Month, Turnover)`
```

```r
fc |> autoplot(myseries)
```



The accuracy score (using the RMSE, or root mean squared error) of the model on both the training and test (forecast) data is outputted below:

```
# train set accuracy
accuracy(fc, myseries)$RMSE
```

```
## [1] 9.108427
```

```
# test set accuracy
accuracy(fit)$RMSE
```

```
## [1] 5.671408
```

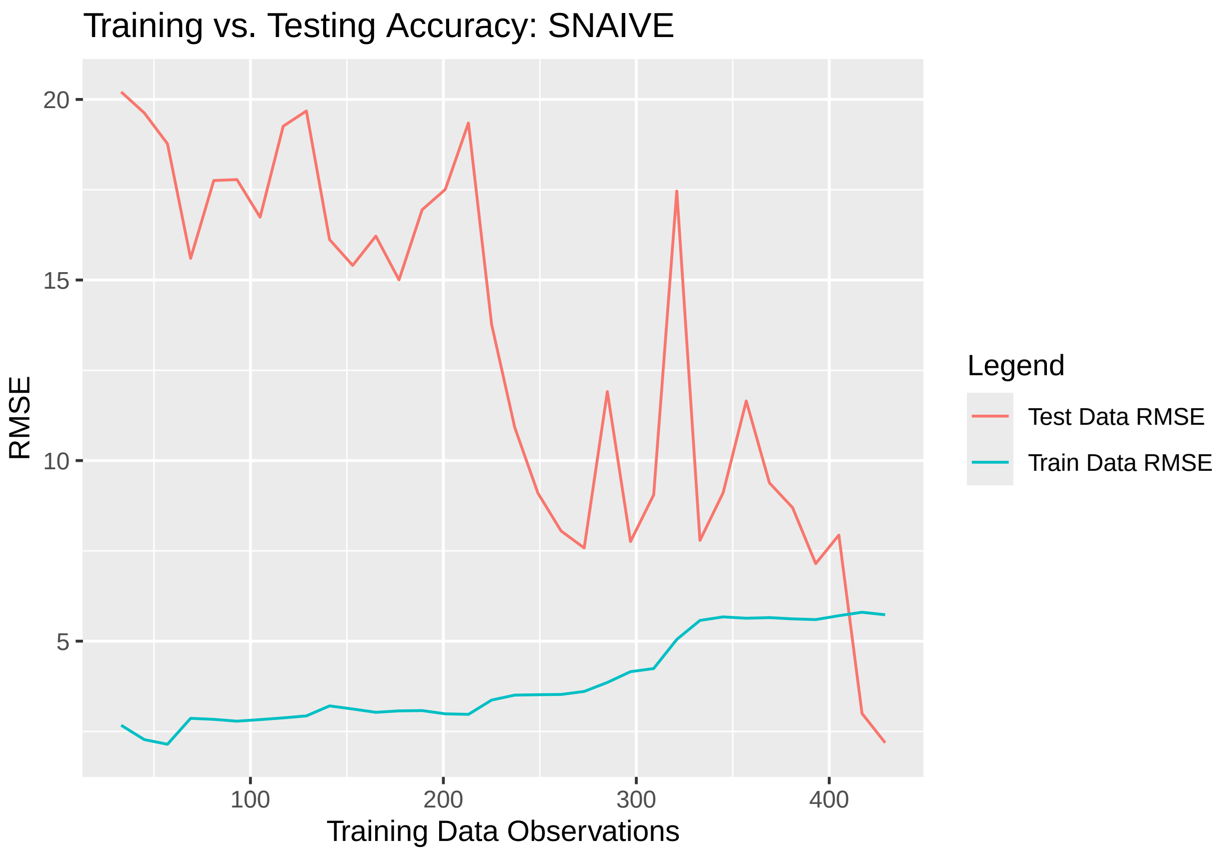
The error is clearly smaller on the training data compared to the test data. This makes sense given the above plot: the forecast did not accuractely capture the upwards trend of the actual test data.

To see how the accuracy scores change with the amount of training data used, the cell below determines the test set and training set accuracy using different training set sizes:

```
acc_scores <- matrix(nrow = 0, ncol = 3)  # Initially no rows, 3 columns


for(i in 1985:2018){
  myseries_train <- myseries |>
    filter(year(Month) < i)
  train_data_size <- dim(myseries_train)[1]
  fit <- myseries_train |>
    model(SNAIVE(Turnover))
  fc <- fit |>
    forecast(new_data = anti_join(myseries, myseries_train))
  test_acc <- accuracy(fc, myseries)$RMSE
  train_acc <- accuracy(fit)$RMSE
  new_row <- c(train_data_size, train_acc, test_acc)
  acc_scores <- rbind(acc_scores, new_row)
}

acc_scores <- as.data.frame(acc_scores)
colnames(acc_scores) <- c('size', 'Train Data RMSE', 'Test Data RMSE')
rownames(acc_scores) <- NULL
```

The results are plotted using the cell below

```
acc_scores %>%
  pivot_longer(c('Train Data RMSE', 'Test Data RMSE'), names_to = 'Legend') %>%
    ggplot(aes(x=size, y=value, color=Legend)) + geom_line() +
  labs(
    title = "Training vs. Testing Accuracy: SNAIVE",
    x = 'Training Data Observations',
    y = 'RMSE'
  )
```



We see that in general the test set accuracy (or the accuracy of the forecast) decreases as more training data is provided. This is due to the fact that the forecast period is shorter, and likely more representative of the period immediately following the end of the test set observations. The opposite is true for train set accuracy, in which the error increases as the train set size increases. This is because the training set contains more data overall, increasing the likelihood that the fitted values will eventually deviate from the actual observations.