DATA 624 Homework 3

Fomba Kassoh

2024-21-03

{r setup, include=FALSE} knitr::opts_chunk$set(echo = TRUE)

Load Required Libraries

```r
options(repos = c(CRAN = "https://cloud.r-project.org"))

# Function to install a package if not already installed
install_if_needed <- function(pkg) {
  if (!requireNamespace(pkg, quietly = TRUE)) {
    install.packages(pkg)

  }
}

# List of packages to check and install if necessary
required_packages <- c("fpp3", "dplyr", "ggplot2", "lubridate", "tsibble",
                       "tsibbledata", "feasts", "fable", "fabletools",
                       "curl", "USgas", "readxl", "readr", "tidyr", "forecast",
                       "seasonal", "patchwork", "LaTeX", "tinytex")

# Loop through the list and install packages only if needed
for (pkg in required_packages) {
  install_if_needed(pkg)
}
```

```
## Registered S3 method overwritten by 'tsibble':
##   method               from
##   as_tibble.grouped_df dplyr

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

## Installing package into 'C:/Users/RemoteUser/AppData/Local/R/win-library/4.4'
## (as 'lib' is unspecified)

## Warning: package 'LaTeX' is not available for this version of R
##
## A version of this package for your version of R might be available elsewhere,
## see the ideas at
## https://cran.r-project.org/doc/manuals/r-patched/R-admin.html#Installing-packages
```

```r
# Function to suppress package startup messages
suppressPackageStartupMessages({
  library(fpp3)
  library(dplyr)
  library(ggplot2)
  library(lubridate)
  library(tsibble)
  library(tsibbledata)
  library(feasts)
  library(fable)
  library(fabletools)
  library(readr)
  library(readxl)
  library(tidyr)
  library(forecast)
  library(seasonal)
  library(patchwork)
  library(tinytex)
})
```

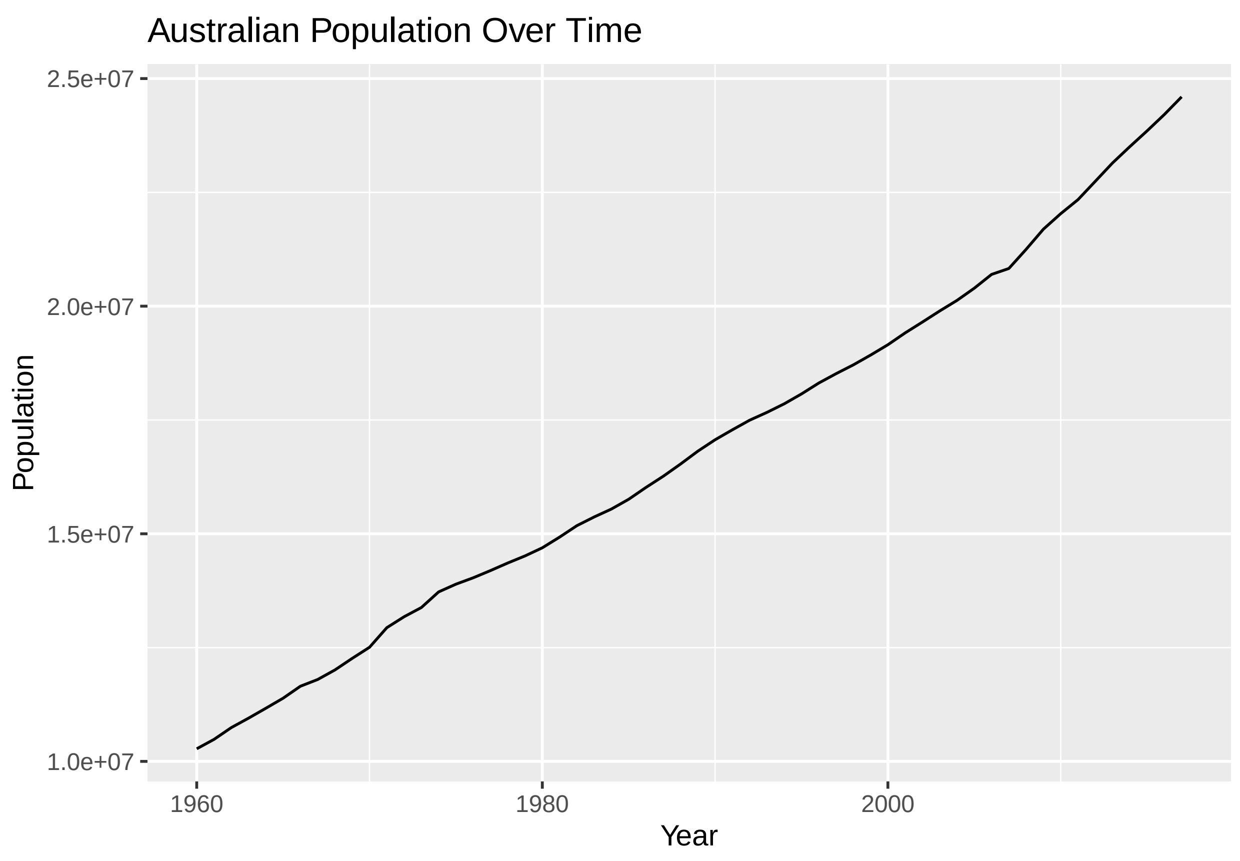
Exercise 5.11.1

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

1. Plot the `Population` for Australia Below is the plot.

```r
# head(global_economy)
aus_population <- global_economy |>
  # 1. Filter rows where Population is available (no missing values) and Country is Australia
  filter(!is.na(Population), Country == 'Australia')

  # 2. Plot the `Population` over time using autoplot
  aus_population |> autoplot(Population) +

  # 3. Customize the plot labels:
  labs(title = "Australian Population Over Time",
       x = "Year",
       y = "Population") +

  # 4. Remove the legend from the plot
  theme(legend.position = "none")
```



2. Analysis of the Chart

- **Trend:** The time series shows a clear upward trend over time.
- **Seasonality:** The chart does not show visible seasonality. The increase appears to be consistent over the years without repeating seasonal patterns.
- **Stationarity:** The data appears non-stationary due to the evident trend.

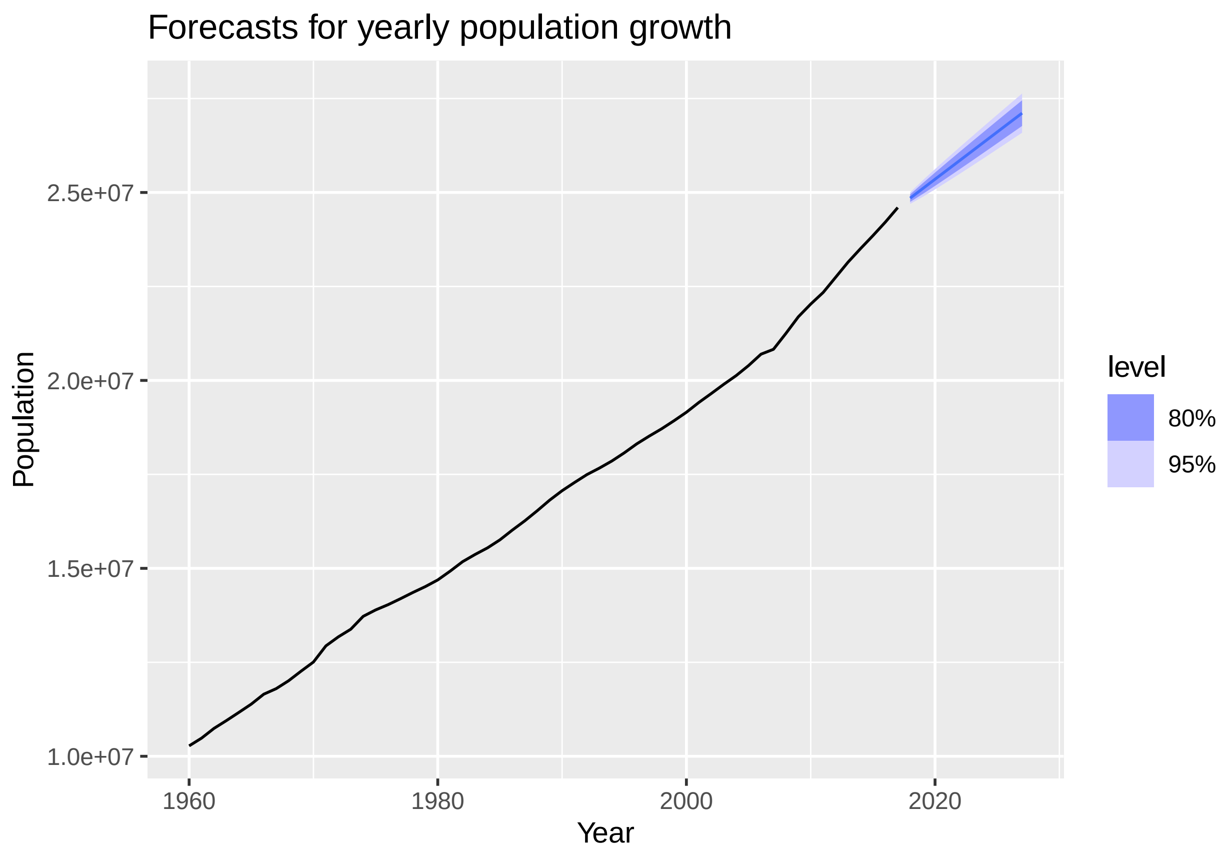
3. Select Model and Forecast - RW(y ~ drift())

- The models NAIVE(y), SNAIVE(y) are more suitation for time series with no trend, and is stationary.
- Given the evident upward trend in the data, RW(y ~ drift()) is the most appropriate method to forecast this time series.

Appropriateness: The RW(y ~ drift()) model appears to be well-suited for this data series as it captures the steady, consistent growth trend in population over time. The forecast is a reasonable continuation of the historical pattern.

```r
aus_population_fit <- aus_population |>
  model(RW(Population ~ drift()))

# Generate forecasts for 10 years
aus_population_fc <- aus_population_fit |> forecast(h = 10)

# Plot forecasts against actual values
```

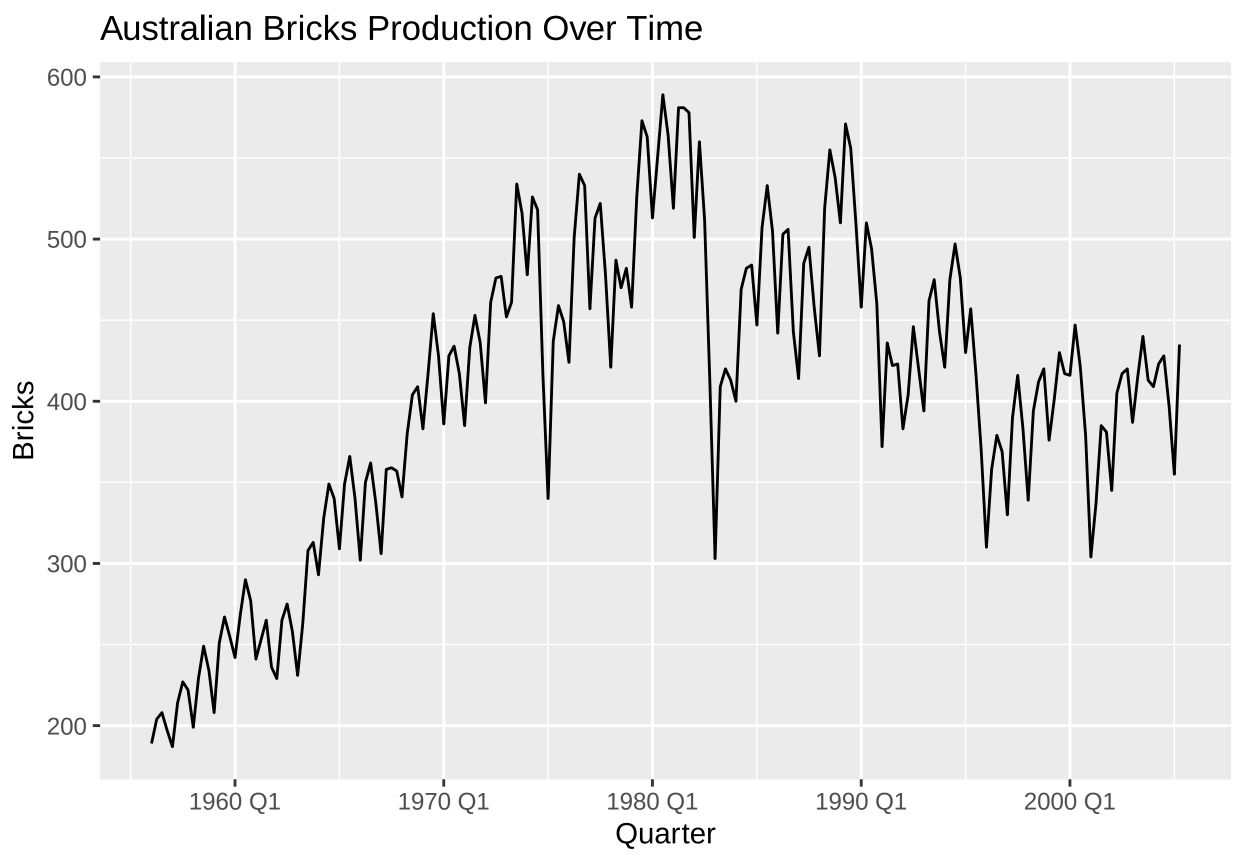
```
aus_population_fc |>
  autoplot(aus_population) +
  labs(
    y = "Population",
    title = "Forecasts for yearly population growth"
  ) +
  guides(colour = guide_legend(title = "Forecast"))
```



Australian Bricks Production

1. Plot the Bricks production for Australia Below is the plot.

```
# head(global_economy)
aus_brick <- aus_production |>
  # 1. Filter rows where Bricks is available (no missing values) and Country is Australia
  filter(!is.na(Bricks)) |>
  select(Quarter, Bricks)

  # 2. Plot the `Bricks` over time using autoplot
   aus_brick |> autoplot(Bricks) +

  # 2. Customize the plot labels:
    labs(title = "Australian Bricks Production Over Time",
        x = "Quarter",
       y = "Bricks") +

  # 4. Remove the legend from the plot
    theme(legend.position = "none")
```



2. Analysis of the Chart

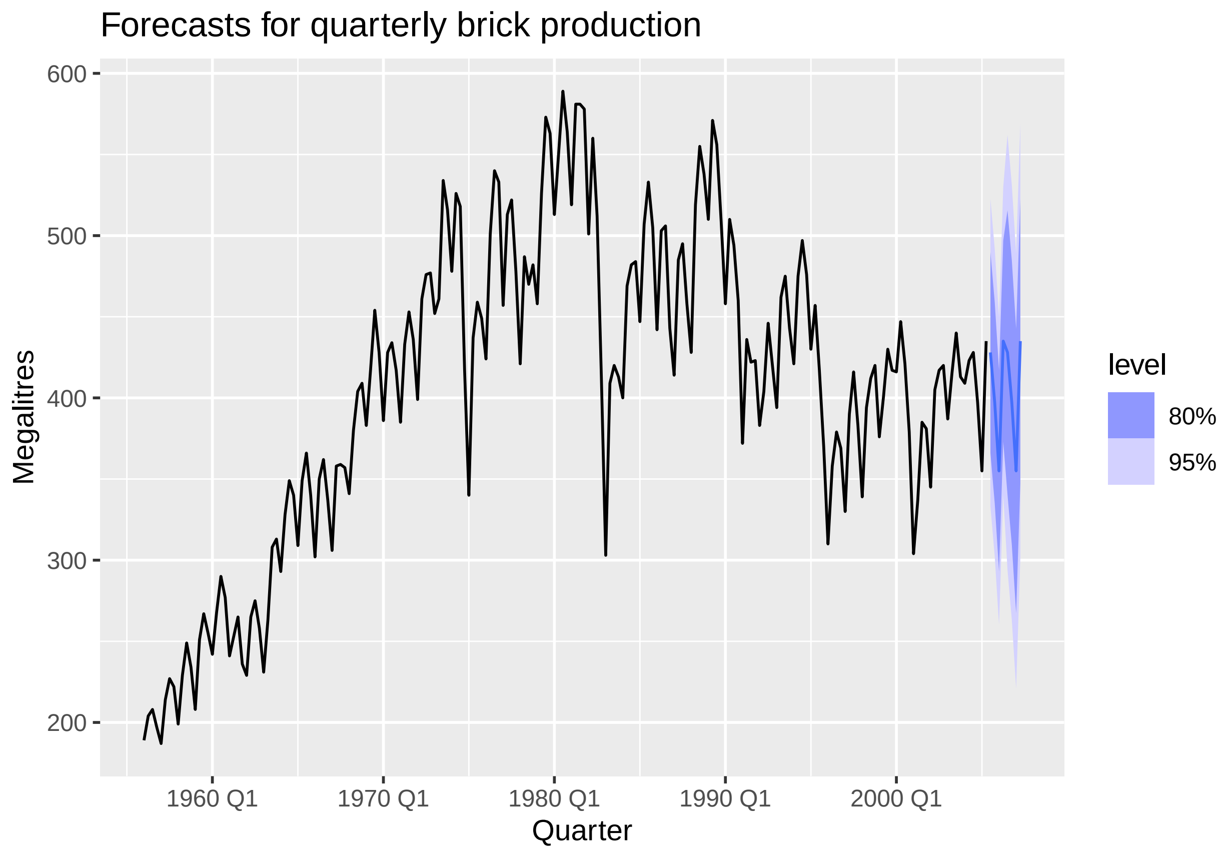
- **Trend:** The time series shows an initial upward trend followed by a decline and then fluctuating cycles over time
- **Seasonality:** The chart shows clear seasonal patterns, with regular fluctuations within each year.
- **Stationarity:** The data appears non-stationary due to the changing levels, trend, and evident seasonality.

3. Select Model and Forecast - SNAIVE(y)

- The models NAIVE(y) and RW(y ~ drift()) are more suitable for time series with no seasonality or those with a consistent trend.
- Given the evident seasonality in the data, SNAIVE(y) is the most appropriate method to forecast this time series.

Appropriateness: The SNAIVE(y) model is well-suited for this data series as it captures the regular seasonal patterns observed over time. The forecast reflects the historical seasonal fluctuations using the last observed value from the same quarter in the previous year.

```r
aus_brick_fit <- aus_brick |>
  model(SNAIVE(Bricks))

# Generate short-term (2 year) forecast focused on seasonality.
aus_brick_fc <- aus_brick_fit |> forecast(h=8)

# Plot forecasts against actual values
aus_brick_fc |>
  autoplot(aus_brick) +
  labs(
    y = "Megalitres",
    title = "Forecasts for quarterly brick production"
  ) +
  guides(colour = guide_legend(title = "Forecast"))
```

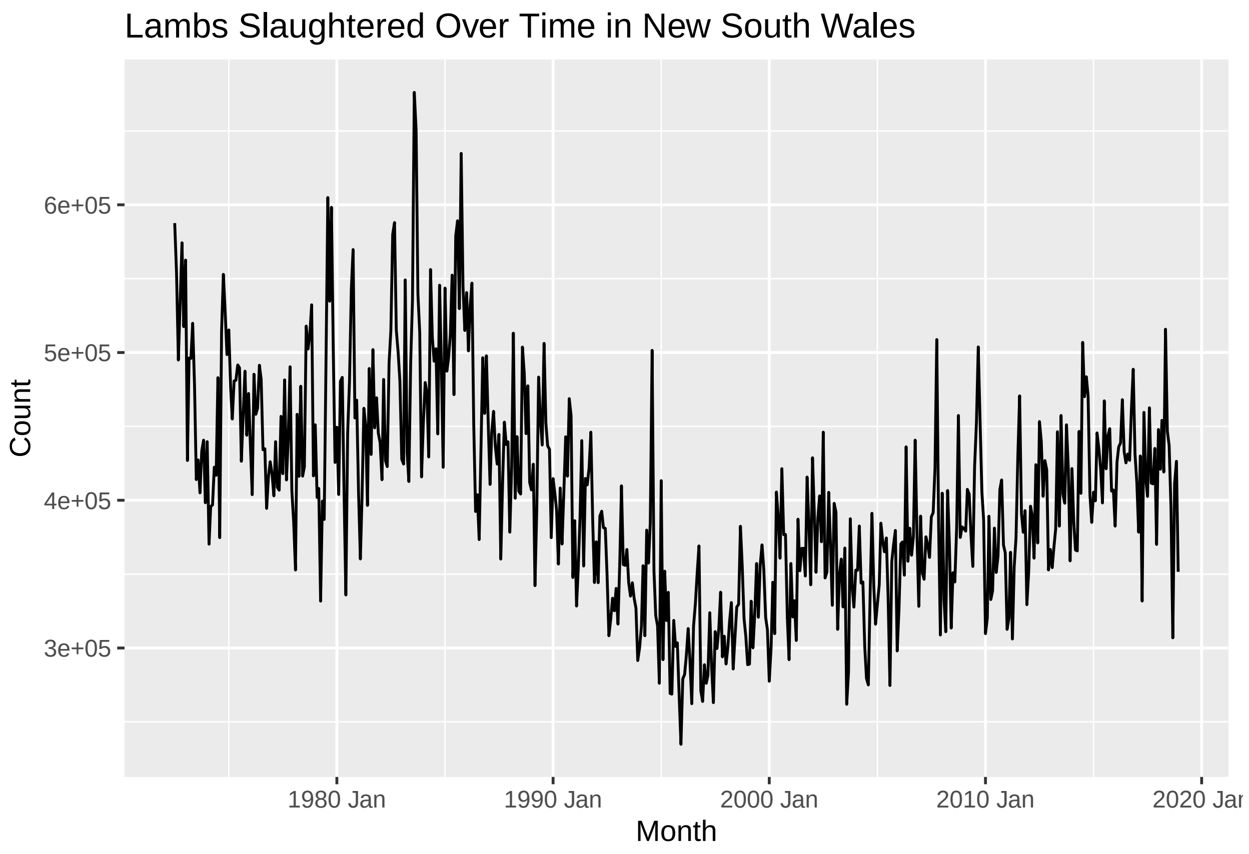


New South Walse Lamb Slaughter

1. Plot the `Lamb` Slaughter for New South Wales Below is the plot.

```r
# Filter data for Lambs in New South Wales
nsw_lamb <- aus_livestock |>
  filter(Animal == "Lambs", State == "New South Wales")

# Plot the data with legend included
```

```r
nsw_lamb |>
  autoplot(Count) +
  labs(title = "Lambs Slaughtered Over Time in New South Wales",
       x = "Month",
       y = "Count")
```



2. **Analysis of the Chart**

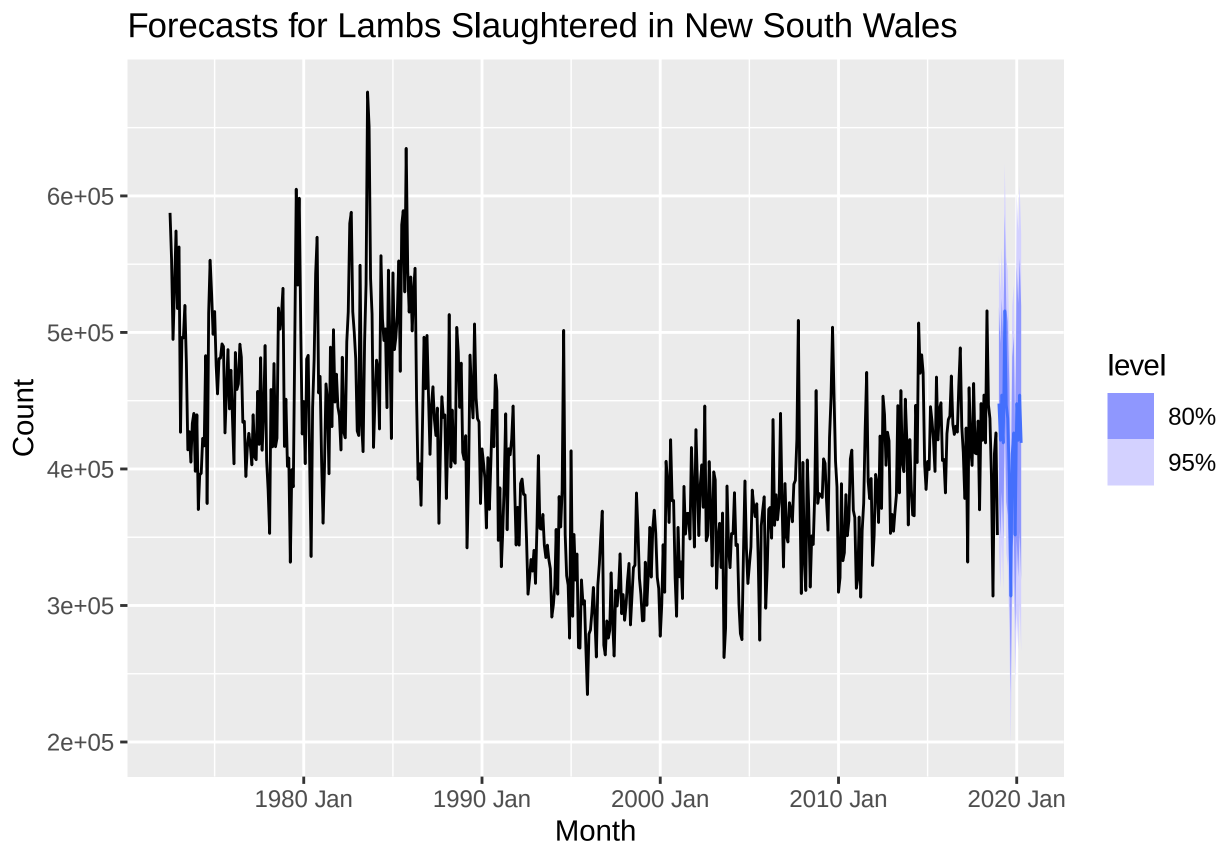
- **Trend:** The time series shows fluctuations with no consistent long-term trend. There are periods of increase and decrease over time.
- **Seasonality:** The chart displays clear seasonal patterns with regular fluctuations within each year.
- **Stationarity:** The data appears non-stationary due to the changing levels and evident seasonality.

3. **Select Model and Forecast - SNAIVE(y)**

- The models NAIVE(y) and RW(y ~ drift()) are more suitable for time series with no seasonality or those with a consistent trend.
- Given the evident seasonality in the data, SNAIVE(y) is the most appropriate method to forecast this time series.

Appropriateness: The SNAIVE(y) model is well-suited for this data series as it captures the regular seasonal patterns observed over time. The forecast reflects the historical seasonal fluctuations using the last observed value from the same season in the previous year.

```r
nsw_lamb_fit <- nsw_lamb |>
  model(SNAIVE(Count))

# Generate short-term (4 year) forecast focused on seasonality
nsw_lamb_fc <- nsw_lamb_fit |>
  forecast(h=16)

# Plot forecasts against actual values
nsw_lamb_fc |>
  autoplot(nsw_lamb) +
  labs(
    y = "Count",
    title = "Forecasts for Lambs Slaughtered in New South Wales"
  ) +
  guides(colour = guide_legend(title = "Forecast"))
```

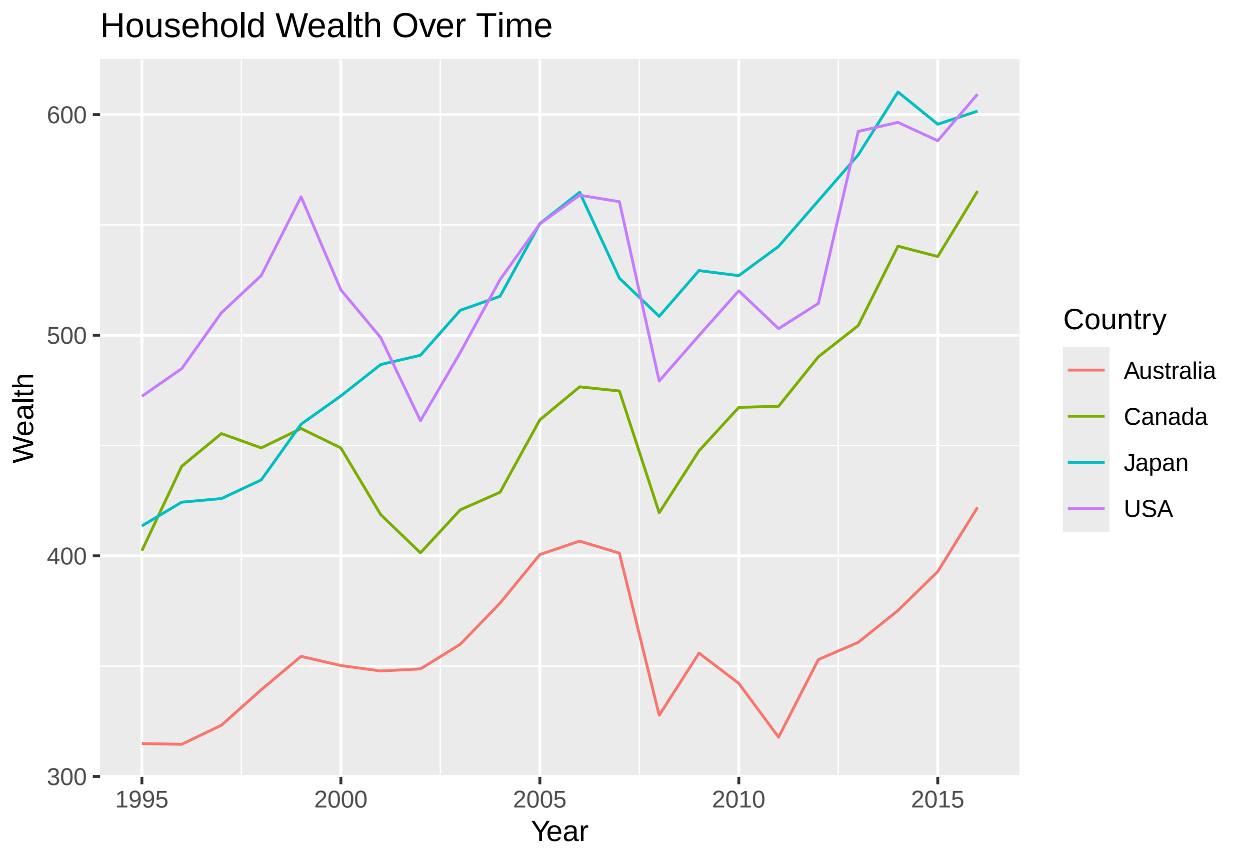


Household Wealth (hh_budget)

1. Plot the Household wealth over time Below is the plot.

```r
# Filter data for Household wealth
hh_wealth <- hh_budget |>
  filter(!is.na(Wealth)) |>
  select(Year, Wealth)
```

```r
# Plot the data with legend included
hh_wealth |>
  autoplot(Wealth) +
  labs(title = "Household Wealth Over Time",
       x = "Year",
       y = "Wealth")
```



2. Analysis of the Chart

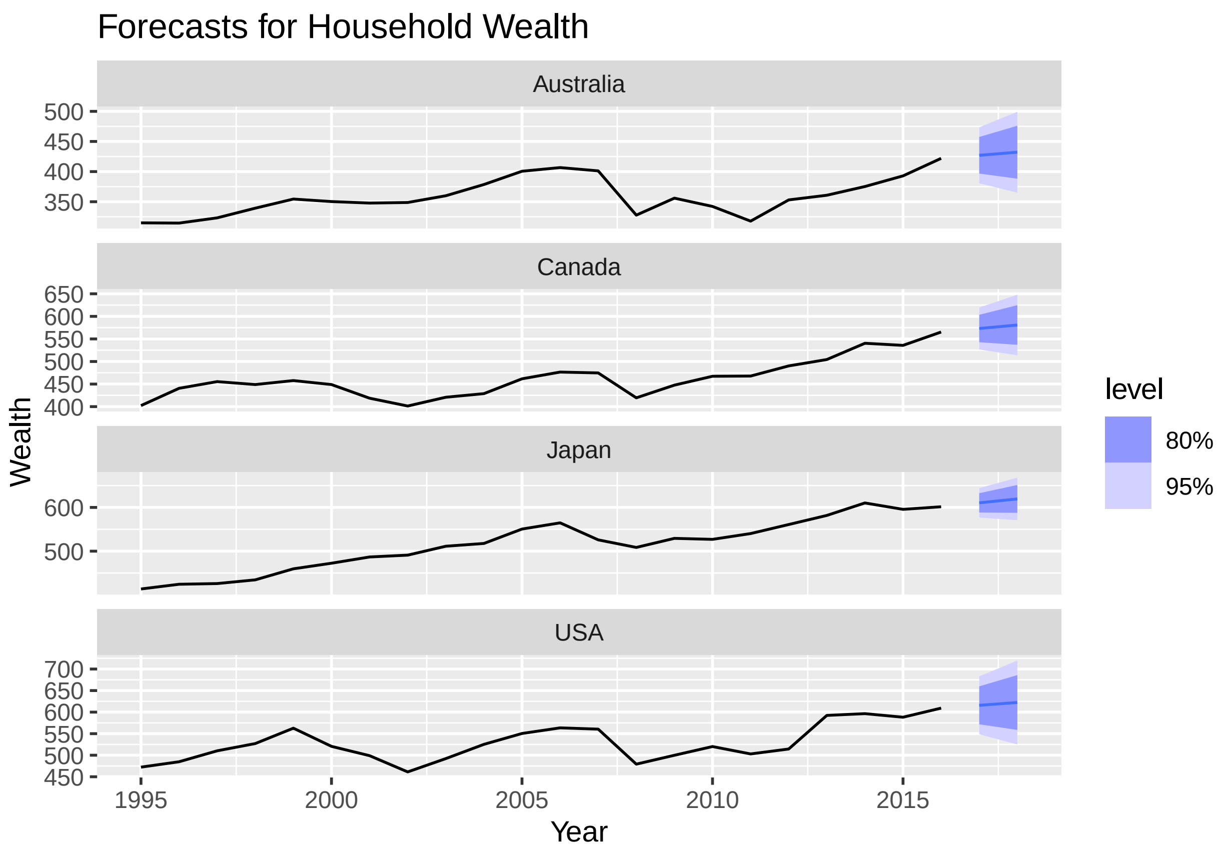
- **Trend:** The time series shows a general upward trend over time, indicating an increase in household wealth.
- **Seasonality:** The chart does not show clear seasonal patterns, as household wealth typically does not fluctuate in a regular seasonal manner.
- **Stationarity:** The data appears non-stationary due to the evident trend.

3. Select Model and Forecast - RW(y ~ drift())

- The models NAIVE(y) and SNAIVE(y) are more suitable for time series with no trend or those with evident seasonality.
- Given the evident upward trend in the data, RW(y ~ drift()) is the most appropriate method to forecast this time series.

Appropriateness: The RW(y ~ drift()) model is well-suited for this data series as it captures the steady, consistent growth trend in household wealth over time. The forecast reflects the historical trend using a linear extension of the recent past.

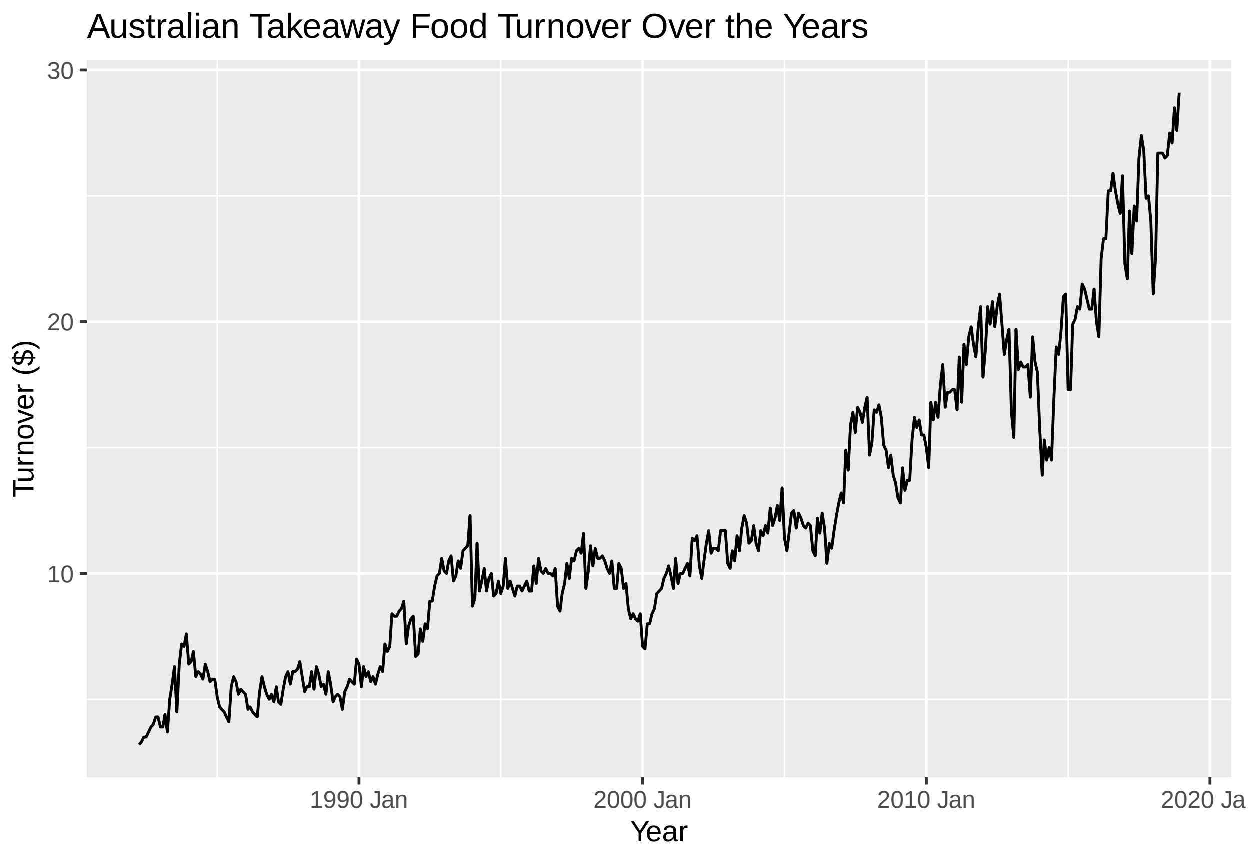
```r
hh_wealth_fit <- hh_wealth %>%
  model(RW(Wealth ~ drift()))

# Generate short-term (2 year) forecast focused on the trend
hh_wealth_fc <- hh_wealth_fit %>%
  forecast(h=2)

# Plot forecasts against actual values
hh_wealth_fc %>%
  autoplot(hh_wealth) +
  labs(
    y = "Wealth",
    title = "Forecasts for Household Wealth"
  ) +
  guides(colour = guide_legend(title = "Forecast"))
```



Australian Takeaway Food Turnover

1. Plot the Takeaway Food Turnover for Australian Capital Territory Below is the plot:

```r
# Filter for Australian Capital Territory and takeaway food turnover data
takeaway_data <- aus_retail |>
  filter(State == 'Australian Capital Territory', Industry == "Takeaway food services") |>
  select(Month, Turnover)

# Plot the data with legend included
takeaway_data |>
  autoplot(Turnover) +
  labs(title = "Australian Takeaway Food Turnover Over the Years",
       x = "Year",
       y = "Turnover ($)")
```



2. **Analysis of the Chart**

- **Trend:** The time series shows a clear and strong upward trend over the years, especially from the mid-2000s onward.
- **Seasonality:** The chart does not show a strong seasonal pattern; instead, the trend is more dominant.
- **Stationarity:** The data appears non-stationary due to the clear upward trend.

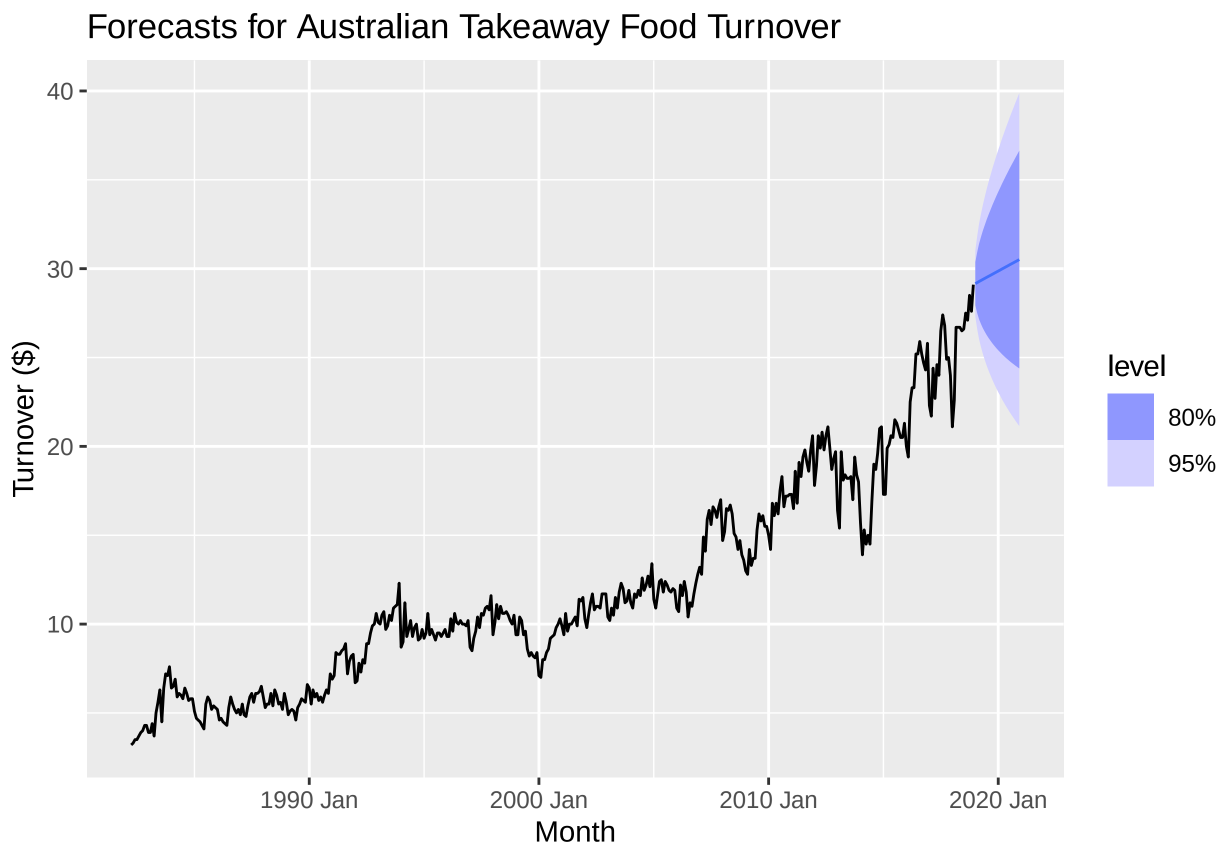
3. **Select Model and Forecast - RW(y ~ drift())**

- The models NAIVE(y) and SNAIVE(y) are more suitable for time series with no trend or those with evident seasonality.

- Given the clear upward trend and the absence of strong seasonality, RW(y ~ drift()) is the most appropriate method to forecast this time series.

Appropriateness: If forecasting the trend is most important, the RW(y ~ drift()) model is suited for this data series as it captures the ongoing upward trend in takeaway food turnover. The forecast reflects the historical trend by projecting an increase in turnover based on the drift observed in the recent past.

```r
# Fit the model using RW with drift
takeaway_fit <- takeaway_data |>
  model(RW(Turnover ~ drift()))

# Generate short-term (2 year) forecast focused on the trend
takeaway_fc <- takeaway_fit |>
  forecast(h = 24)  # Assuming monthly data, h = 24 for 2 years

# Plot forecasts against actual values
takeaway_fc |>
  autoplot(takeaway_data) +
  labs(
    y = "Turnover ($)",
    title = "Forecasts for Australian Takeaway Food Turnover"
  ) +
  guides(colour = guide_legend(title = "Forecast"))
```



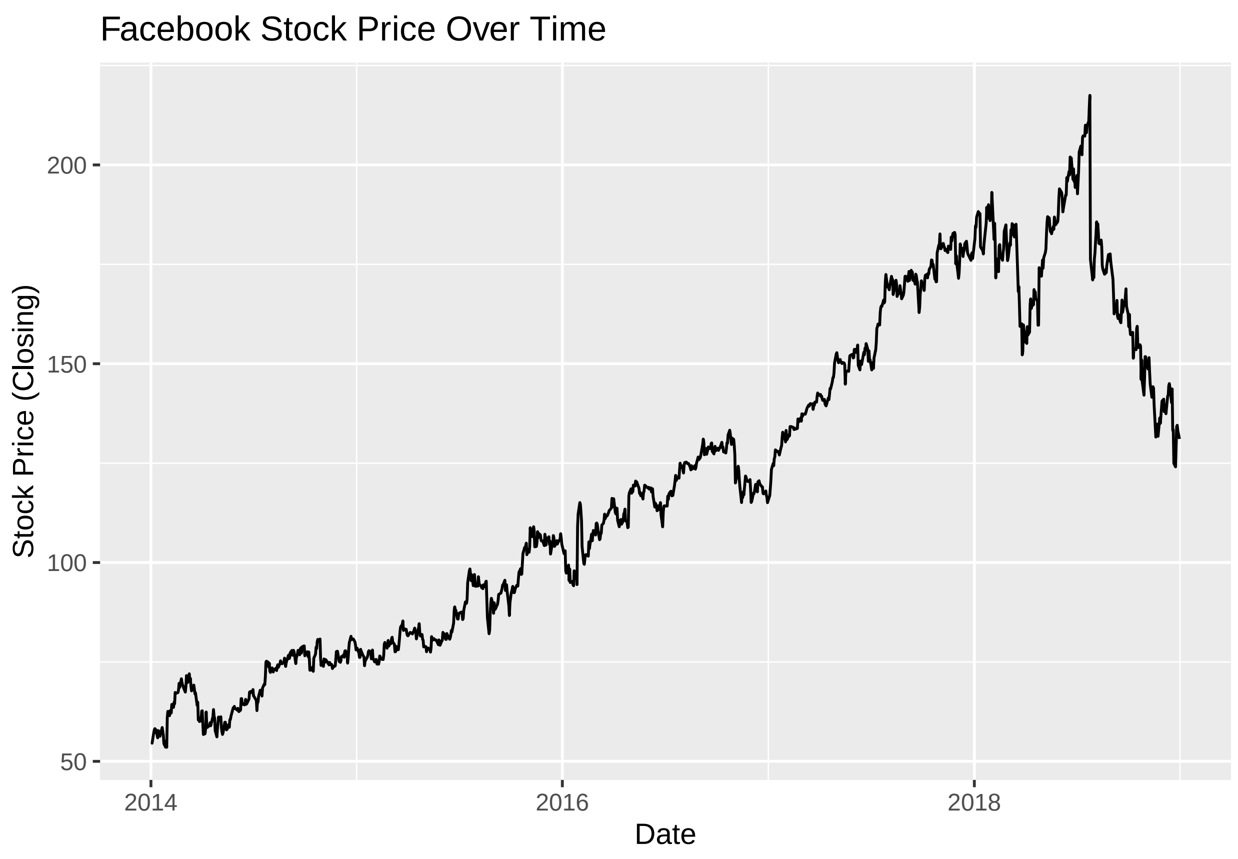
Exercise 5.11.2

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

a. Produce a time plot of the series.

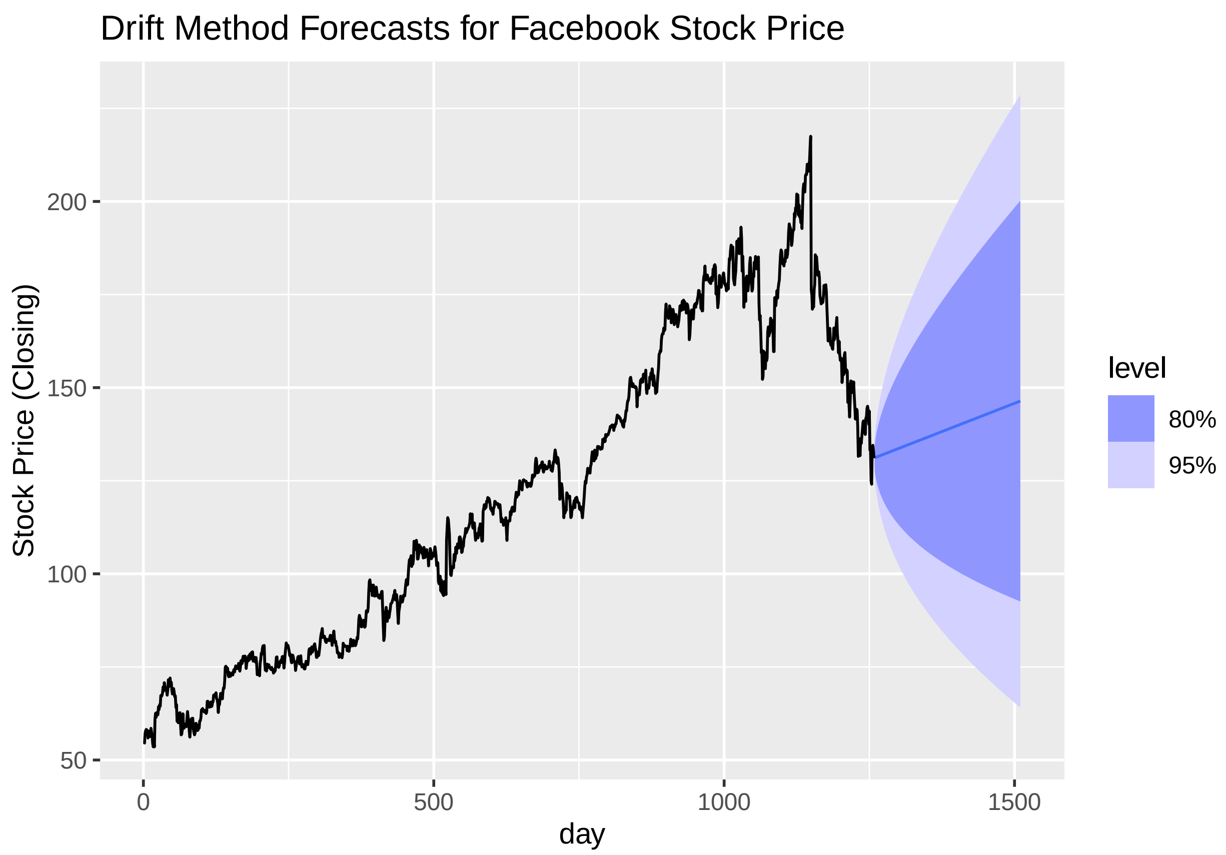
plot the time series

```
# Load necessary libraries
library(fpp3)

# Filter for Facebook stock data
# Ensure the tsibble is correctly defined with Date as the index
facebook_stock <- gafa_stock |>
  filter(Symbol == "FB") |>
  as_tsibble(index = Date, key = Symbol) |>
  fill_gaps()

# Produce a time plot of the Facebook stock price
facebook_stock |>
  autoplot(Close) +
  labs(title = "Facebook Stock Price Over Time",
       x = "Date",
       y = "Stock Price (Closing)")
```



### b. Produce Forecasts Using the Drift Method and Plot Them

```
# Fit the model using RW with drift on the filtered data
fb_drift_fit <- facebook_stock |>
  mutate(Close = as.numeric(Close), day = row_number()) |>
  update_tsibble(index = day, regular = TRUE) |>
  select(Date, Close)

fb_drift_fc <- fb_drift_fit |>
  model(RW(Close ~ drift())) |>
  forecast(h = 252) |>
  autoplot(fb_drift_fit) +
        labs(y = "Stock Price (Closing)", title = "Drift Method Forecasts for Facebook Stock Price")

fb_drift_fc
```

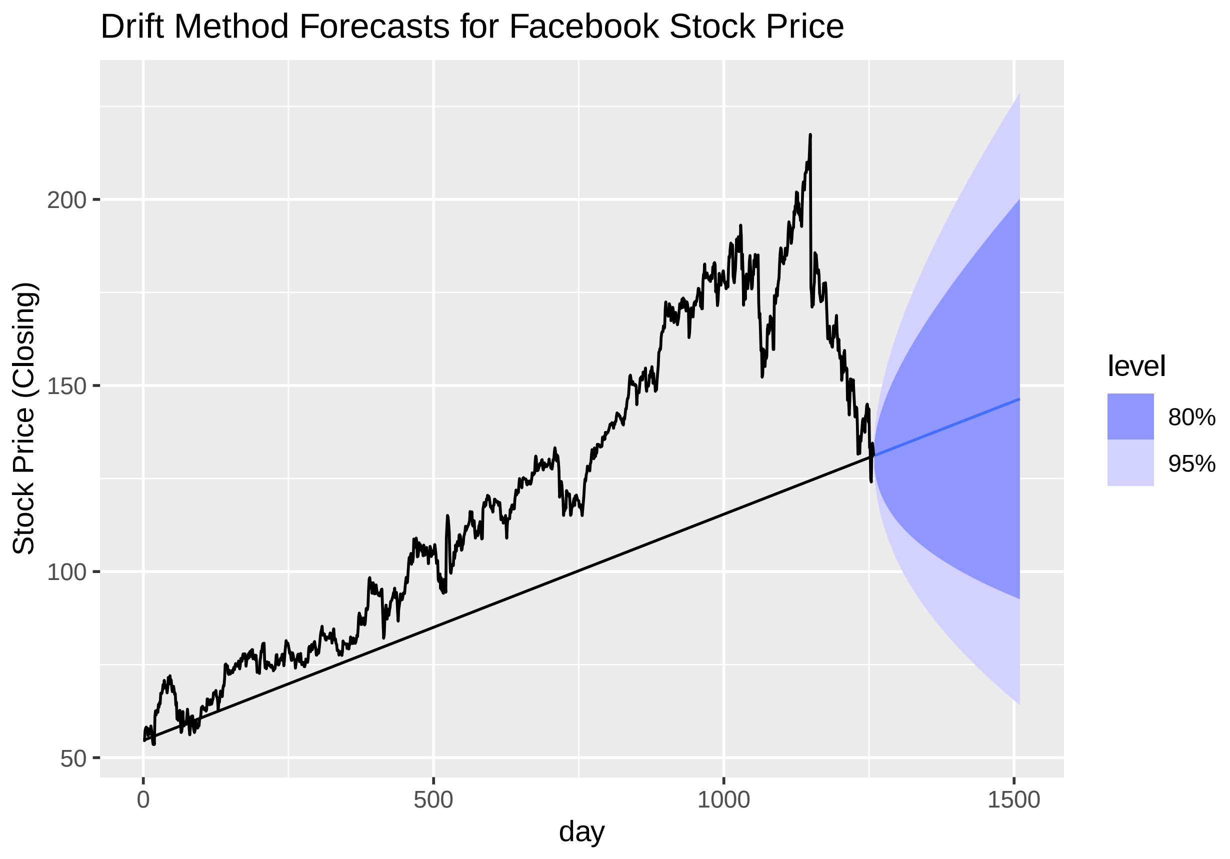


### c. Show that the forecasts are identical to extending the line drawn between the first and last observations.

The fact that the line connecting the first and last observations aligns closely with the forecast produced by the drift method confirms. This demonstrate that the forecasts produced by the drift method are identical to extending the line drawn between the first and last observations.

```
# Fit the model using RW with drift on the filtered data
facebook_stock <- facebook_stock |>
  mutate(day = row_number()) |>
  update_tsibble(index = day, regular = TRUE)
```

```r
# Generate forecast
fb_drift_fc <- facebook_stock |>
  model(RW(Close ~ drift())) |>
  forecast(h = 252)

# Get the first and last observations
first_day <- min(facebook_stock$day)
last_day <- max(facebook_stock$day)
first_close <- facebook_stock$Close[1]
last_close <- facebook_stock$Close[nrow(facebook_stock)]

# Plot the forecasts against actual values with a line connecting first and last points
autoplot(fb_drift_fc, facebook_stock) +
  annotate("segment", x = first_day, y = first_close, xend = last_day, yend = last_close,
           color = "black", linetype = "solid") +
  labs(y = "Stock Price (Closing)", title = "Drift Method Forecasts for Facebook Stock Price")
```

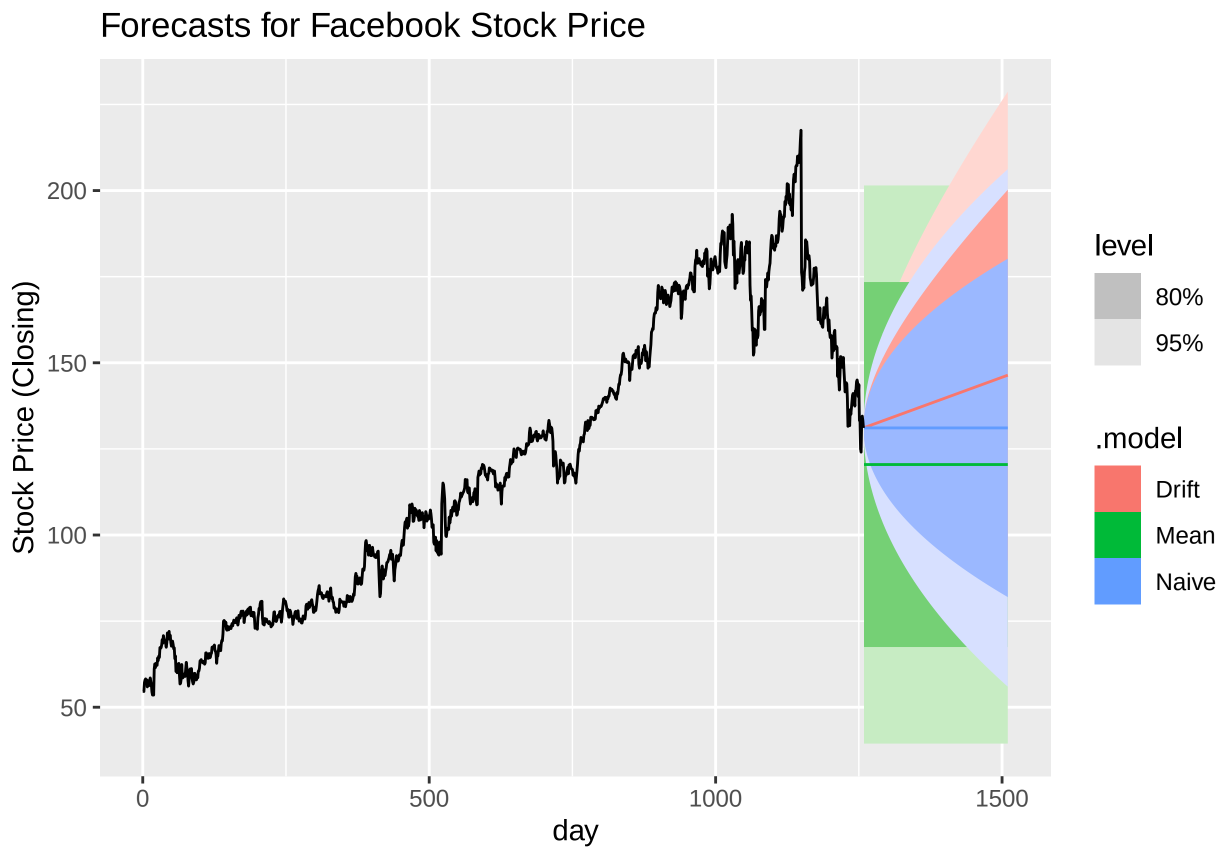


d. Try using some of the other benchmark functions to forecast the same data set. Which do you think is best? Why?

Model to Exclude: Given the nature of daily stock prices and the lack of seasonality, it's best to exclude the SNAIVE model since it's not appropriate for this type of data.

Best Model: The Drift method is the best for forecasting Facebook stock prices because it extends the observed trend into the future. The Naive method simply repeats the last observed value, ignoring the trend, while the Mean method averages past values, failing to capture recent movements. Since the data shows a clear trend, the Drift method provides the most realistic and accurate forecast.

```r
# Generate forecast using appropriate models for daily data
fb_drift_fc2 <- facebook_stock |>
  model(
    Mean = MEAN(Close),
    Naive = NAIVE(Close),
    Drift = RW(Close ~ drift())
  ) |>
  forecast(h = 252)

# Plot the forecasts against actual values
fb_drift_fc2 |>
  autoplot(facebook_stock) +
  labs(y = "Stock Price (Closing)", title = "Forecasts for Facebook Stock Price")
```



Exercise 5.11.3

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.What do you conclude?

Residuals Analysis

- **Innovation Residuals:** The residuals fluctuate around zero, which is typical for white noise. However, there are periods where the residuals deviate significantly, indicating that the model might not have fully captured all underlying patterns.
- **Residual Distribution (Histogram):** The histogram shows that the residuals are centered around zero, which is good, but the presence of outliers suggests that the residuals are not purely random and may contain some unexplained variance.
- **Autocorrelation (ACF Plot):** The ACF plot shows a small but noticeable spike at lag 4, which slightly exceeds the confidence interval. This suggests that there may be some autocorrelation at this lag, indicating that the residuals are not entirely white noise.

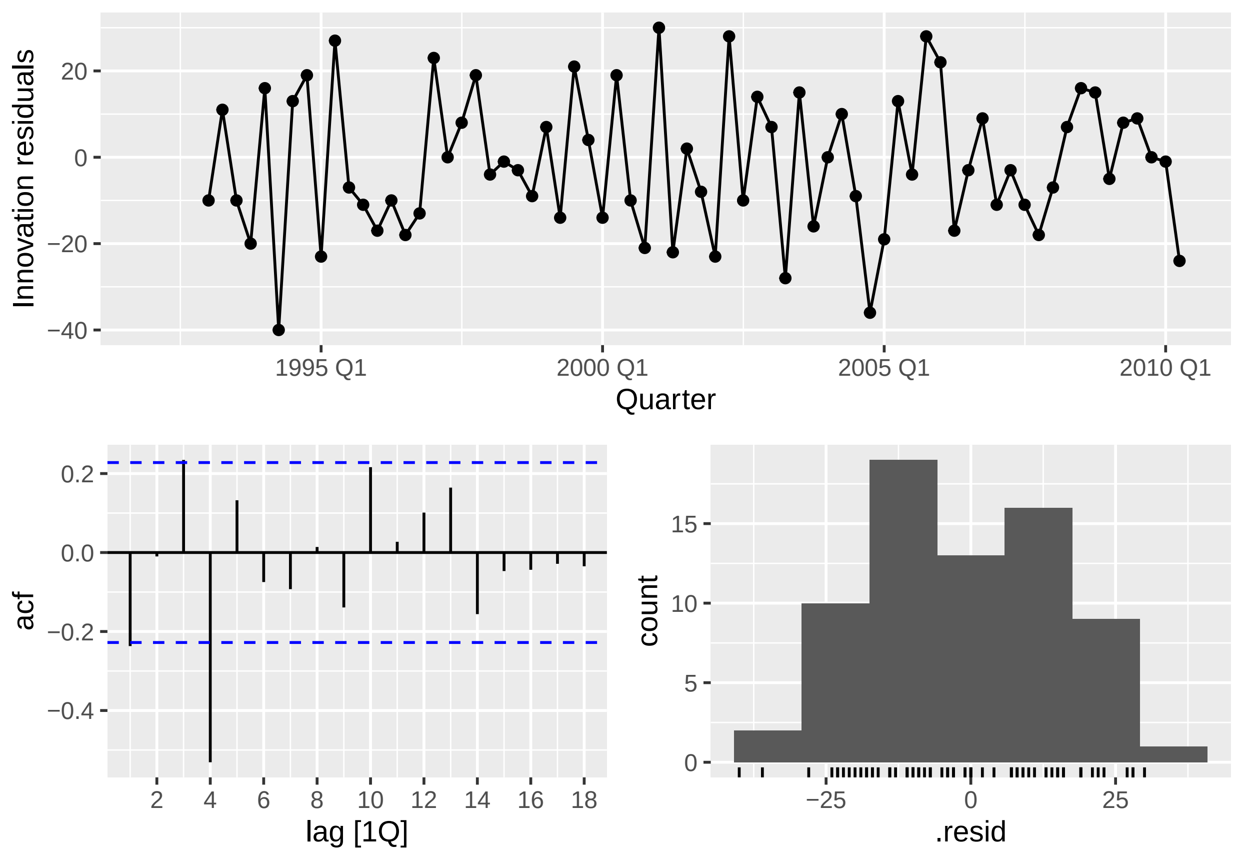
Conclusion Given the small autocorrelation at lag 4 and the presence of outliers, further refinement of the model may be necessary to improve accuracy and better capture the data's underlying patterns.

```
# 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()
```

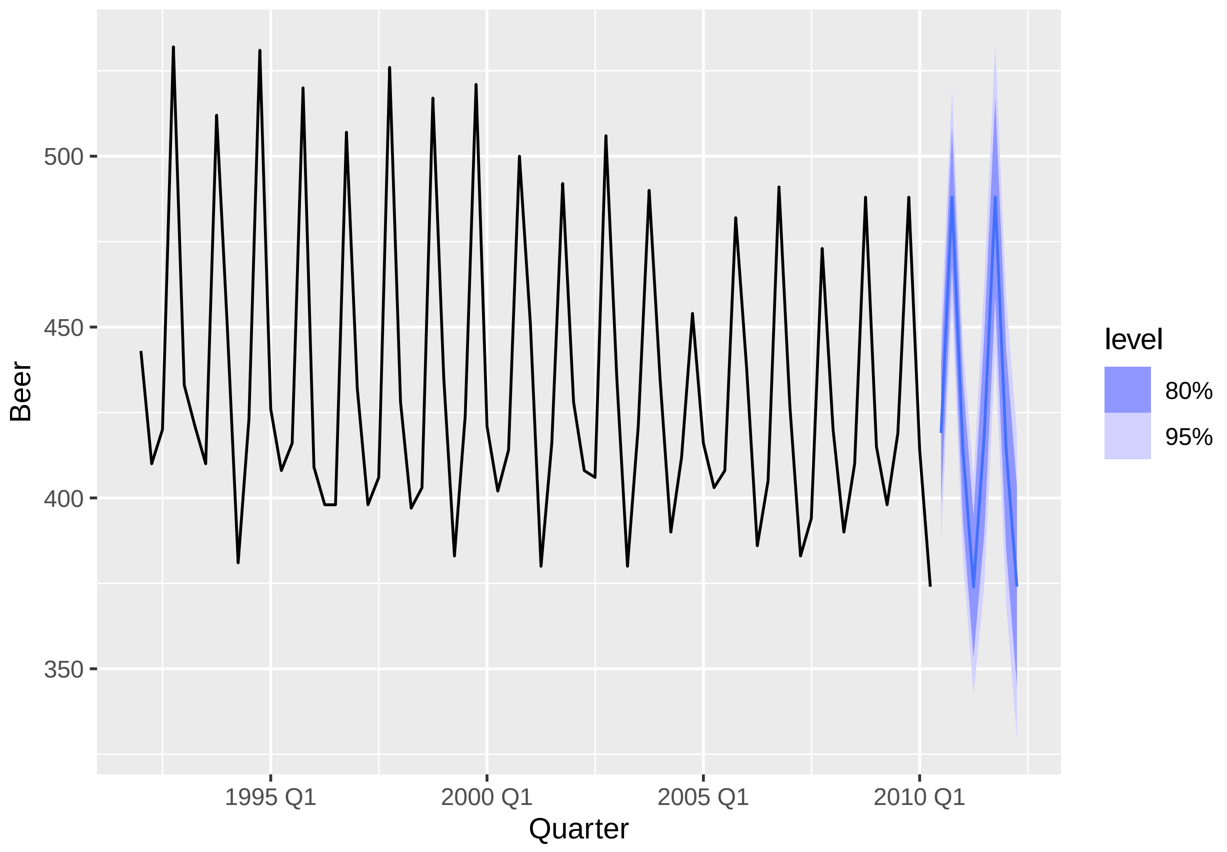
```
## Warning: Removed 4 rows containing missing values or values outside the scale range
## (‘geom_line()‘).
```

```
## 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
# Look a some forecasts
fit |> forecast() |> autoplot(recent_production)
```



Exercise 5.11.4

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.

1. Australian Exports Series from global_economy (NAIVE):

Residuals Analysis Innovation Residuals: The residuals generally fluctuate around zero, which is expected for white noise. However, some significant deviations indicate that the model might not fully capture all underlying patterns in the data.

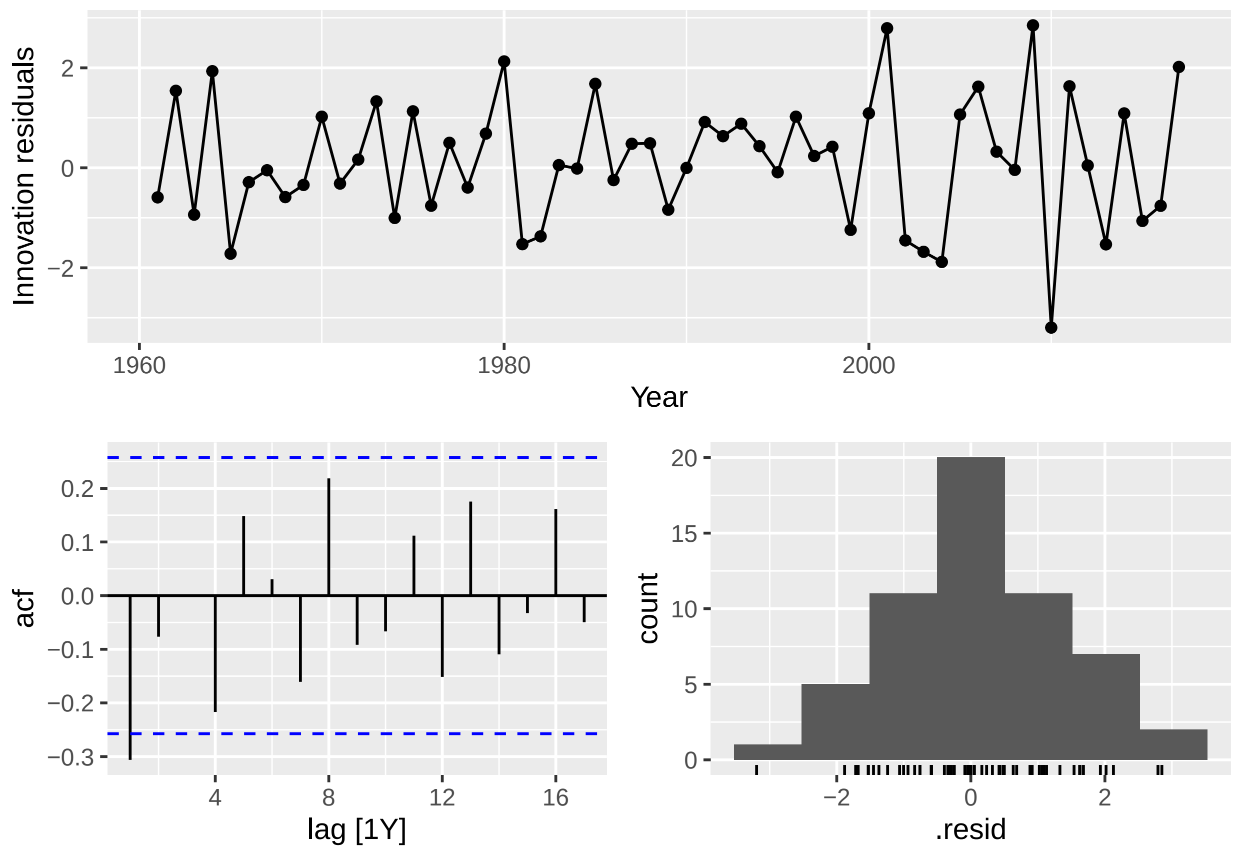
Residual Distribution (Histogram): The histogram shows that residuals are centered around zero, indicating no consistent bias in the model. However, the presence of outliers suggests that some variance remains unexplained by the model.

Autocorrelation (ACF Plot): The ACF plot shows no significant autocorrelation at any specific lag, as the spikes mostly fall within the confidence intervals. This suggests that the residuals are close to white noise.

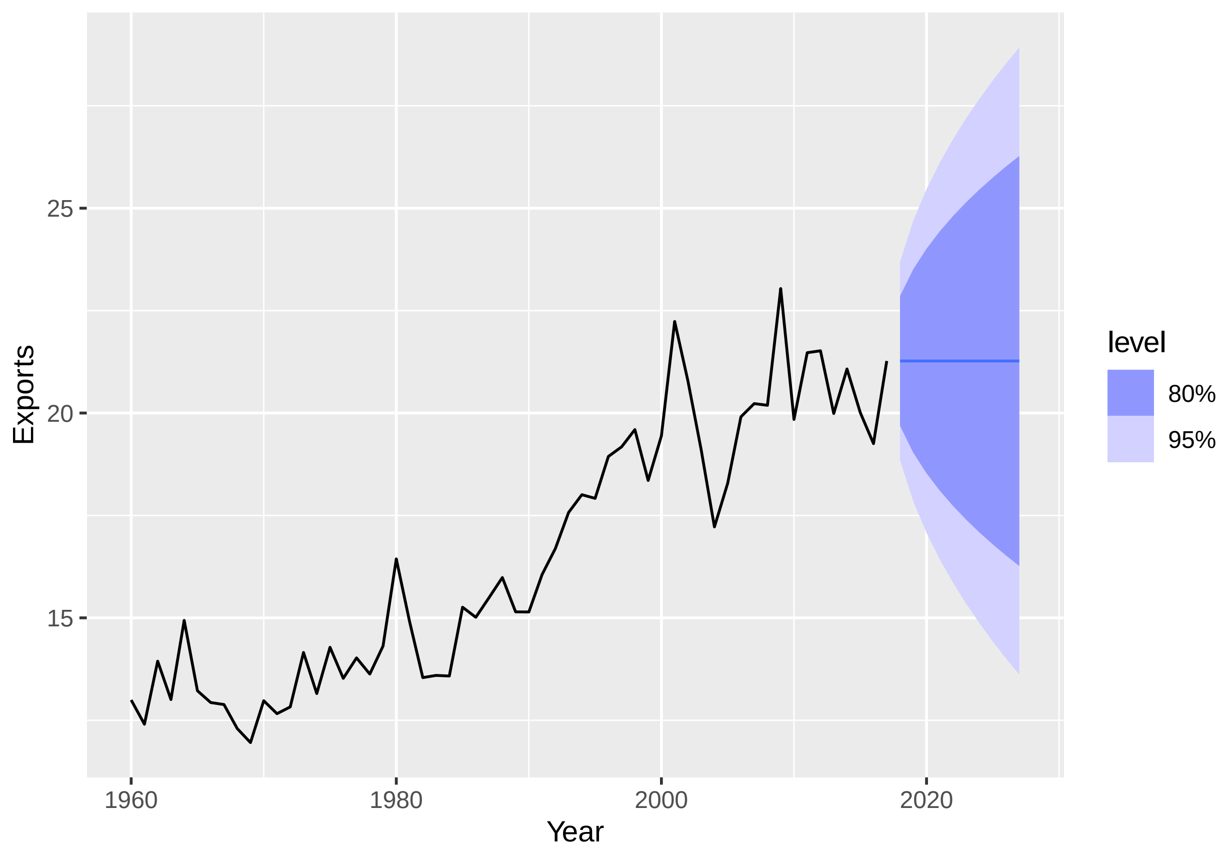
Conclusion The model captures most of the data's structure, as indicated by the lack of significant autocorrelation in the residuals. However, the presence of outliers and some significant deviations in the residuals suggest that further model refinement might be needed to better account for all the variability in the data.

```r
# Extract data of interest
recent_exports <- global_economy |>
  filter(Country == "Australia") |>
  select(Year, Exports)

# Define and estimate a model
# Since exports data might have seasonality based on annual economic cycles, we consider using SNAIVE i
fit_exports <- recent_exports |> model(NAIVE(Exports))

# Look at the residuals
fit_exports |> gg_tsresiduals()
```

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

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

## Warning: Removed 1 row containing non-finite outside the scale range
## (‘stat_bin()‘).
```



```r
# Look at some forecasts
fit_exports |> forecast(h = 10) |> autoplot(recent_exports)
```



2. Bricks Series from aus_production (SNAIVE)

Residuals Analysis **Innovation Residuals:** The residuals generally fluctuate around zero, but there are significant deviations, especially in the 1970s and 1980s. This suggests that the model may not fully capture all underlying patterns during these periods.

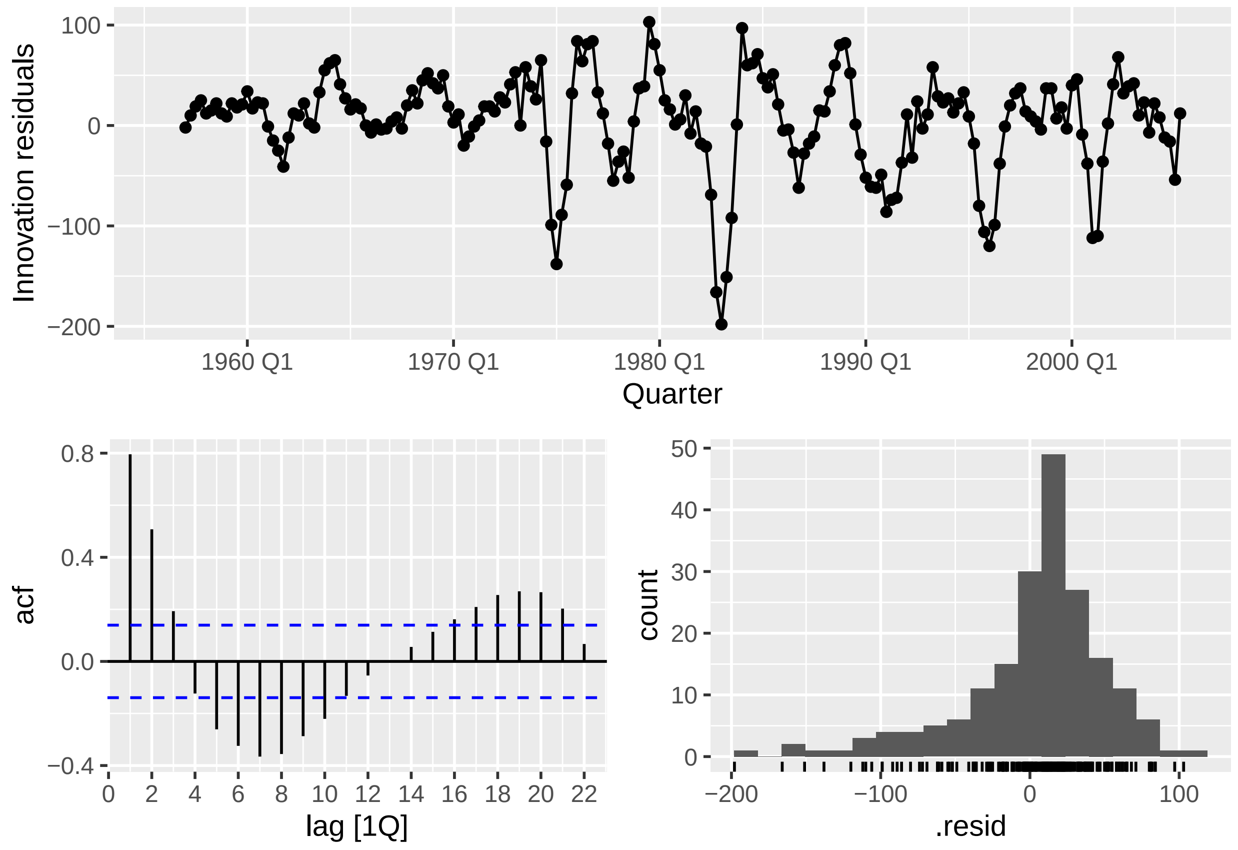
Residual Distribution (Histogram): The histogram shows residuals centered around zero, but with a noticeable skew to the right and some extreme positive values. This indicates the model might not fully explain all the variability in the data.

Autocorrelation (ACF Plot): The ACF plot shows significant autocorrelation at lag 1 and some other lags, suggesting that the residuals are not completely white noise and that some structure in the data remains unexplained.

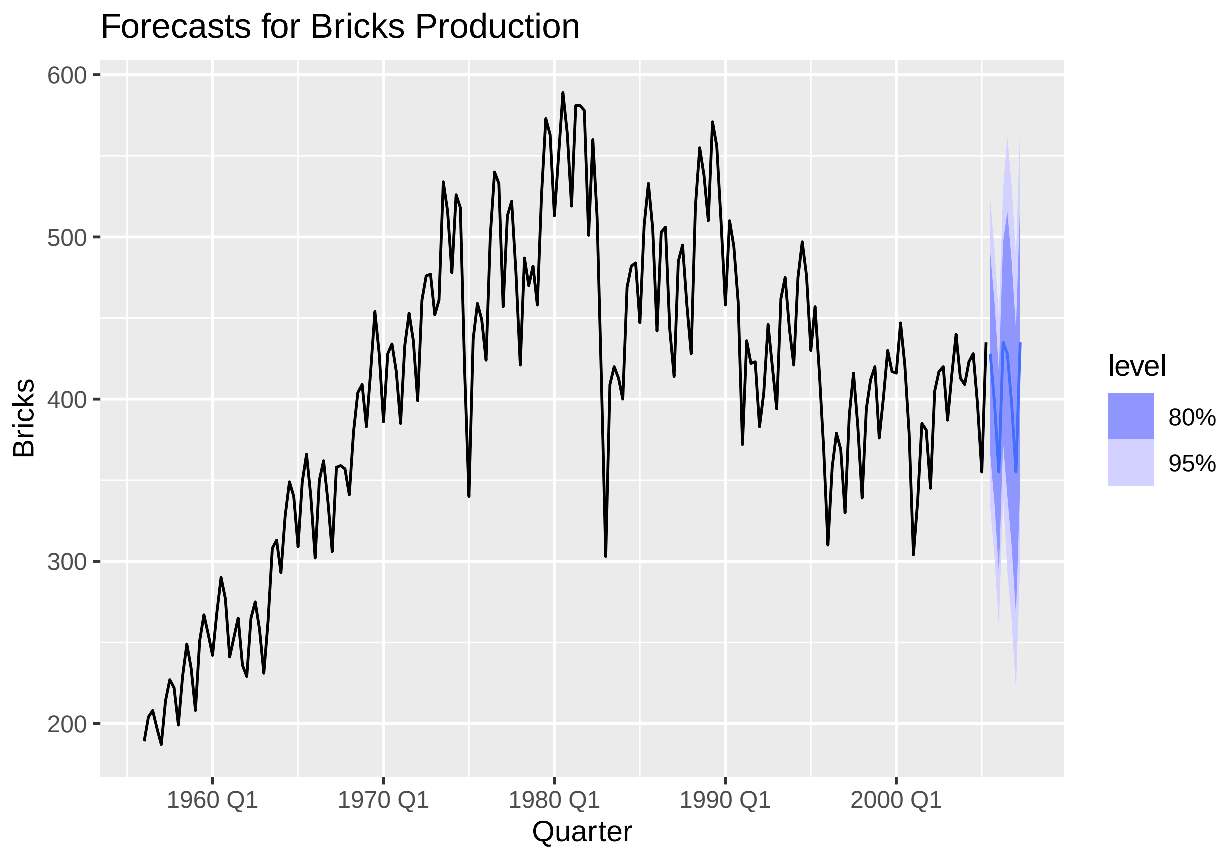
Conclusion The model captures the general structure of the data but misses some patterns, as indicated by significant autocorrelation and skewness in the residuals. Further model refinement may be necessary to better account for these issues.

```r
# Load the necessary libraries
library(fpp3)

# Remove rows with missing or non-finite values
# Extract data of interest
recent_bricks <- aus_production |>
  filter(!is.na(Bricks), is.finite(Bricks)) |>
  select(Quarter, Bricks)

# Define and estimate a model
# The bricks series is likely to have seasonality, so SNAIVE is appropriate.
bricks_fit <- recent_bricks |> model(SNAIVE(Bricks))

# Look at the residuals
bricks_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
# Generate and plot some forecasts
bricks_fc <- bricks_fit |> forecast(h = 8)

autoplot(bricks_fc, recent_bricks) +
  labs(title = "Forecasts for Bricks Production", y = "Bricks")
```



Exercise 5.11.7: For your retail time series (from Exercise 7 in Section 2.10)

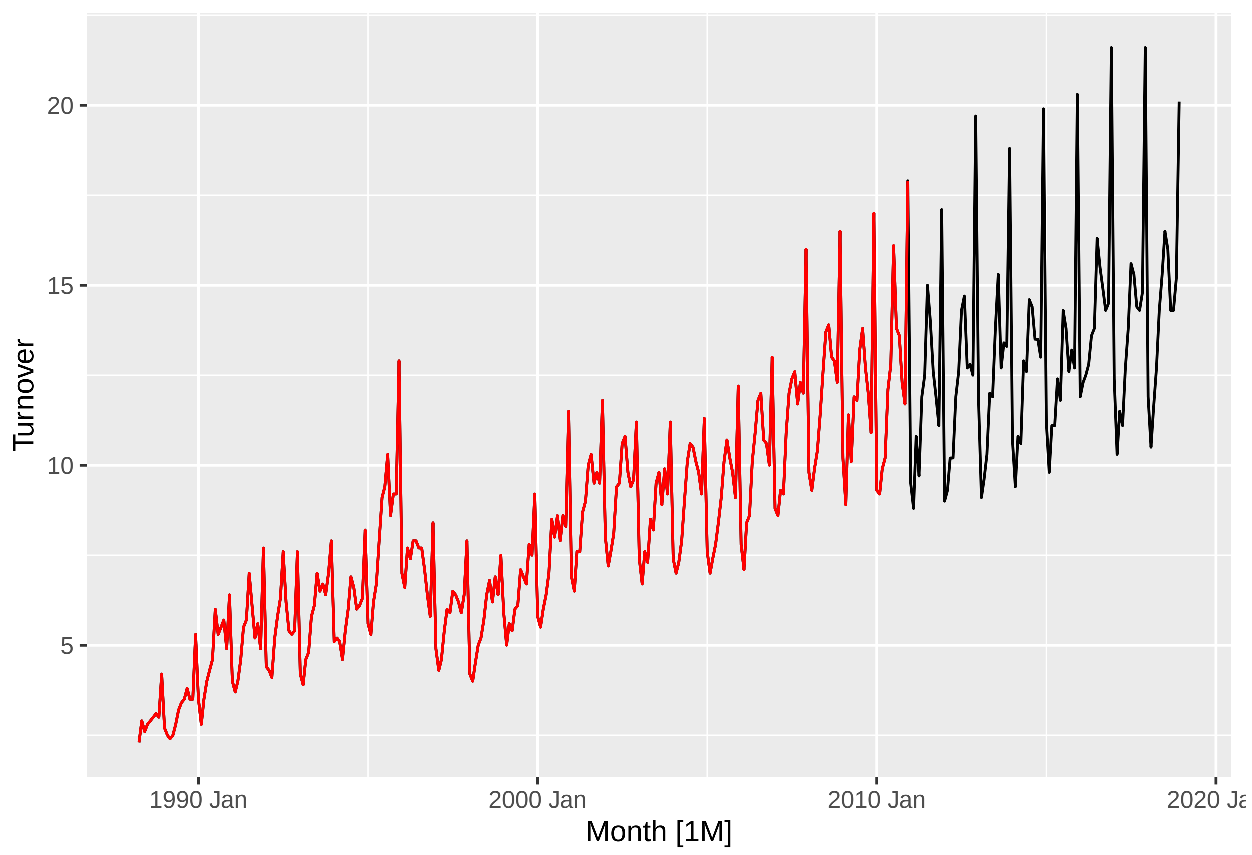
```r
set.seed(12345678)
myseries <- aus_retail |>
  filter(`Series ID` == sample(aus_retail$`Series ID`,1))
```

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

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

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

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



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

```
# Fit the SNAIVE model explicitly specifying the response variable
fit <- myseries_train |>
  model(snaive_model = SNAIVE(Turnover))
```

d. Check the residuals.

Analysis of Residuals

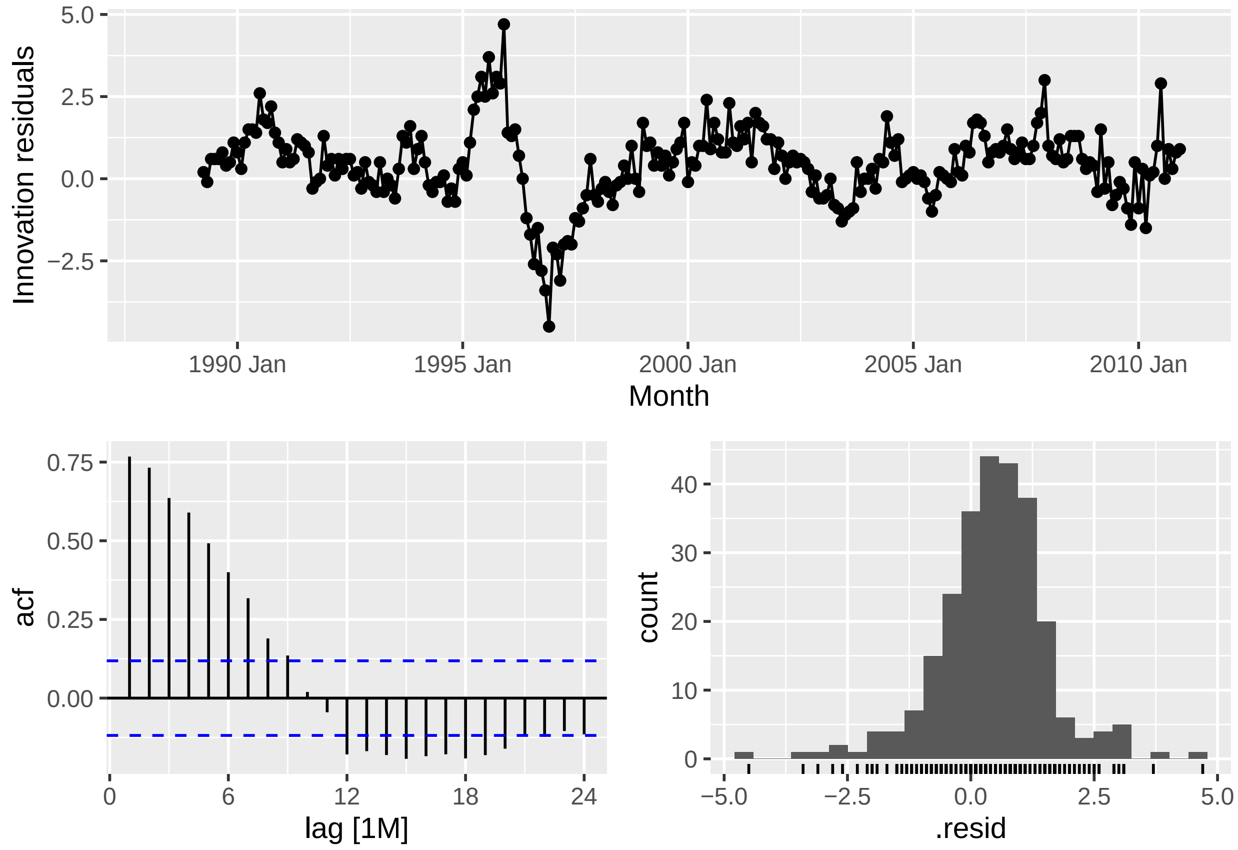
Autocorrelation (ACF Plot): The ACF plot shows significant autocorrelation, particularly at the first few lags, with a gradual decline. This suggests that the residuals are not entirely uncorrelated.

Residual Distribution (Histogram): The histogram shows that the residuals are centered around zero, with a roughly symmetric distribution, which is a good sign. However, the distribution has a slight peak and tails that extend beyond the central bulk of the data. This suggests that while the residuals are approximately normally distributed, there may be some deviation from normality, particularly in the tails.

Conclusion: The residuals do not appear to be entirely uncorrelated, as indicated by the significant autocorrelation at early lags. Additionally, while the residuals are roughly normally distributed, there are slight deviations, particularly in the tails.

```
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()`).
```

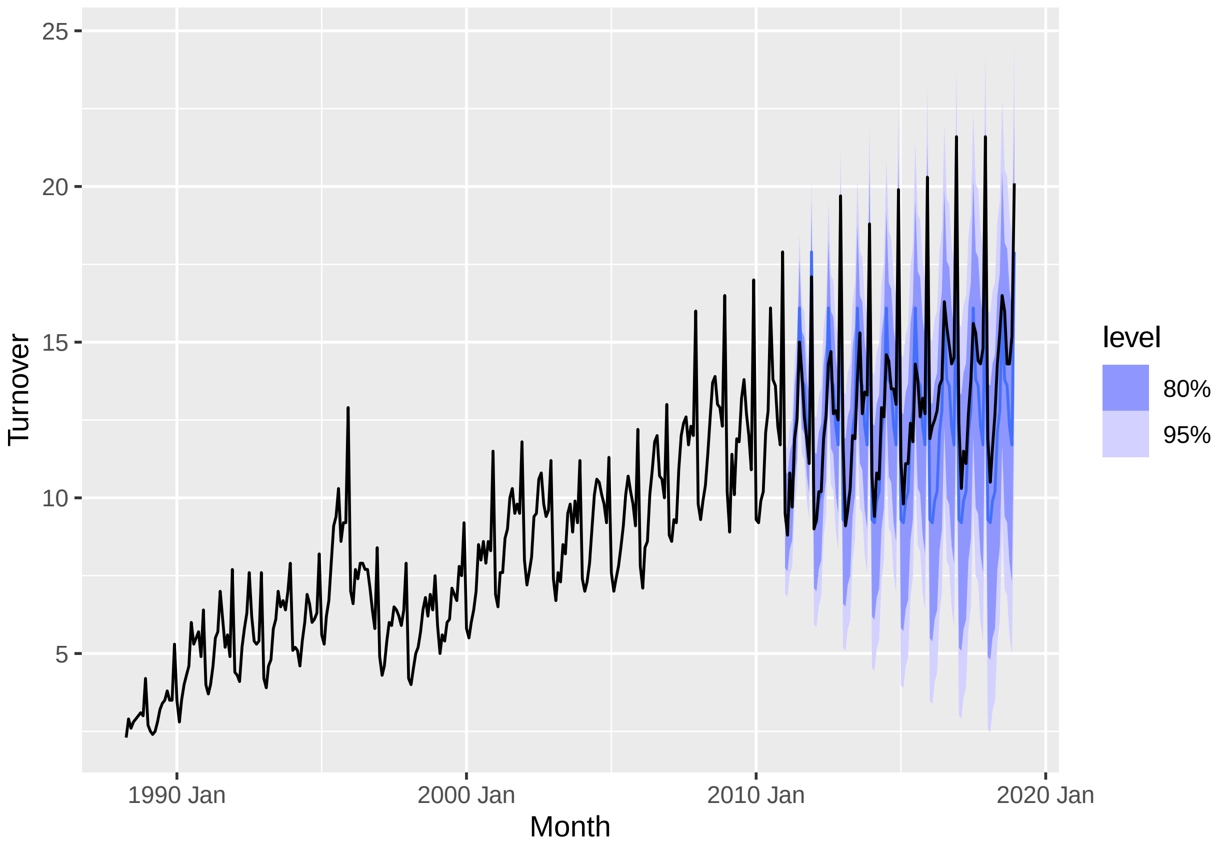


e. Produce forecasts for the test data.

```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)
```



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

Visual comparison The forecast appears to align well with the actual data in terms of both trend and seasonality, indicating that the model has captured the key patterns in the data. The widening of the confidence intervals is appropriate and suggests that the model is appropriately cautious about future uncertainty.

numerical accuracy assessment Based on the ***Training*** and ***Test*** accuracy matrix, here is a comparison between the training and test set metrics:

1. **Mean Error (ME):**
   - **Training:** 0.4387 | **Test:** 0.8365
   - **Interpretation:** The test ME is nearly double the training ME, indicating that the model tends to overestimate more on the test data than on the training data.
2. **Root Mean Squared Error (RMSE):**
   - **Training:** 1.2137 | **Test:** 1.5520
   - **Interpretation:** RMSE is higher on the test set, suggesting that the model's predictions are less accurate on unseen data, with larger errors more prevalent.
3. **Mean Absolute Error (MAE):**
   - **Training:** 0.9146 | **Test:** 1.2406
   - **Interpretation:** The MAE is higher on the test set, indicating that, on average, the model's predictions are further from the actual values when applied to unseen data.

4. **Mean Percentage Error (MPE) and Mean Absolute Percentage Error (MAPE):**
   - **MPE Training:** 5.2292 | **MPE Test:** 5.9401
   - **MAPE Training:** 12.4018 | **MAPE Test:** 9.0641
   - **Interpretation:** MPE is slightly worse in the test set, but interestingly, MAPE is lower in the test set. This suggests that while the absolute percentage errors are higher in the training set, the test set still has some significant deviations, possibly due to overfitting.
5. **Mean Absolute Scaled Error (MASE) and Root Mean Squared Scaled Error (RMSSE):**
   - **MASE Training:** 1.0000 | **MASE Test:** 1.3565
   - **RMSSE Training:** 1.0000 | **RMSSE Test:** 1.2787
   - **Interpretation:** Both MASE and RMSSE are higher in the test set, indicating that the model performs worse on unseen data compared to a naive baseline, suggesting potential overfitting or the need for model improvement.
6. **Autocorrelation of Residuals at Lag 1 (ACF1):**
   - **Training:** 0.7676 | **Test:** 0.6012
   - **Interpretation:** The lower ACF1 in the test set suggests that residuals are less autocorrelated, which is desirable, but the training set's higher ACF1 indicates that the model may not have fully captured all patterns in the training data.

Overall Conclusion: The model performs better on the training data than on the test data, as evidenced by the lower error metrics on the training set.

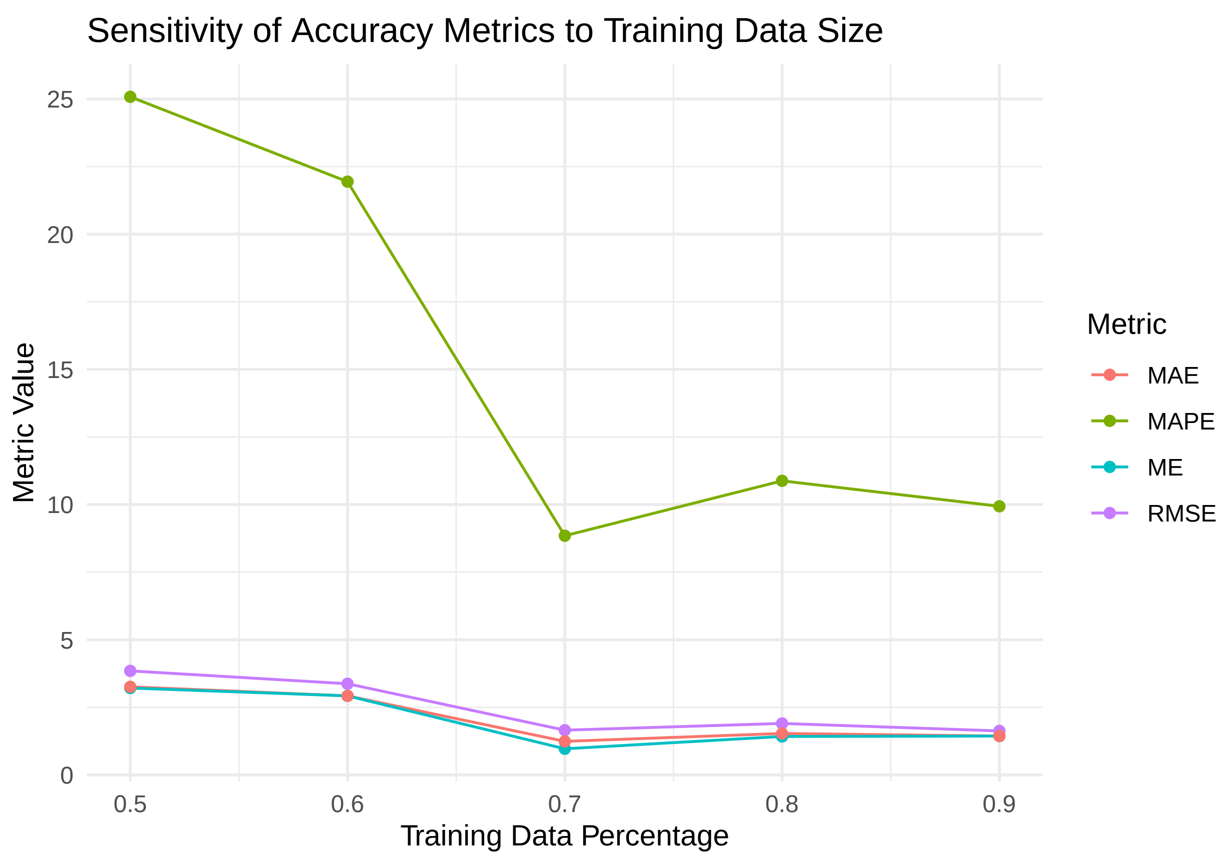
```r
library(dplyr)

# Calculate accuracy metrics for the fit
fit_accuracy <- fit |> accuracy()

# Calculate accuracy metrics for the forecast against the actual series
fc_accuracy <- fc |> accuracy(myseries)

# Combine both accuracy results into one data frame
combined_accuracy <- bind_rows(fit_accuracy, fc_accuracy) |>
  select(-State, -Industry, -.model)

# View the combined data frame
combined_accuracy
```

```
## # A tibble: 2 x 9
##   .type       ME  RMSE   MAE   MPE  MAPE  MASE RMSSE  ACF1
##   <chr>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Training 0.439  1.21 0.915  5.23 12.4   1     1    0.768
## 2 Test     0.836  1.55 1.24   5.94  9.06  1.36  1.28 0.601
```

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

The plot below indicates that, overall, the accuracy measures are not highly sensitive to the amount of training data used, with the exception of MAPE, which shows more variation, particularly at lower percentages of training data. This suggests that increasing the training data generally improves forecast accuracy, but the improvement becomes marginal beyond a threshold of approximately 70% training data, especially for measures like RMSE and MAE.

```r
library(fpp3)
library(tidyr)  # For reshaping data

# Define the percentages of data to use for training
training_percentages <- c(0.5, 0.6, 0.7, 0.8, 0.9)

# Initialize a list to store accuracy results
accuracy_results <- list()

for (pct in training_percentages) {

  # Determine the number of observations to use for training
  n_train <- floor(pct * nrow(myseries))

  # Split the data into myseries_train and myseries_test sets
  myseries_train <- myseries[1:n_train, ]
  myseries_test <- myseries[(n_train + 1):nrow(myseries), ]

  # Train the model on the myseries_train set
  fit <- myseries_train |> model(SNAIVE(Turnover))

  # Generate forecasts
  fc <- fit |> forecast(h = nrow(myseries_test))

  # Calculate accuracy on the myseries_train set and myseries_test set
  train_accuracy <- accuracy(fit)
  test_accuracy <- accuracy(fc, myseries_test)

  # Combine results and store them
  combined_accuracy <- bind_rows(train_accuracy, test_accuracy)
  combined_accuracy <- combined_accuracy |> mutate(Training_Percentage = pct)
  accuracy_results[[length(accuracy_results) + 1]] <- combined_accuracy
}

# Combine all results into a single data frame
all_results <- bind_rows(accuracy_results)

# Reshape the data to a long format for plotting
all_results_long <- all_results |>
  filter(.type == "Test") |>
  select(Training_Percentage, ME, RMSE, MAE, MAPE) |>  # Select the metrics you want to plot
  pivot_longer(cols = c(ME, RMSE, MAE, MAPE),
               names_to = "Metric",
               values_to = "Value")

# Plot multiple metrics on the same plot
ggplot(all_results_long, aes(x = Training_Percentage, y = Value, color = Metric)) +
  geom_line() +
  geom_point() +
  labs(title = "Sensitivity of Accuracy Metrics to Training Data Size",
       x = "Training Data Percentage",
       y = "Metric Value") +
  theme_minimal()
```


Sensitivity of Accuracy Metrics to Training Data Size
Metric Value
Training Data Percentage
0
5
10
15
20
25
0.5
0.6
0.7
0.8
0.9
Metric
MAE
MAPE
ME
RMSE