Data 624 Homework 3 Chapter 5.11

Enid Roman

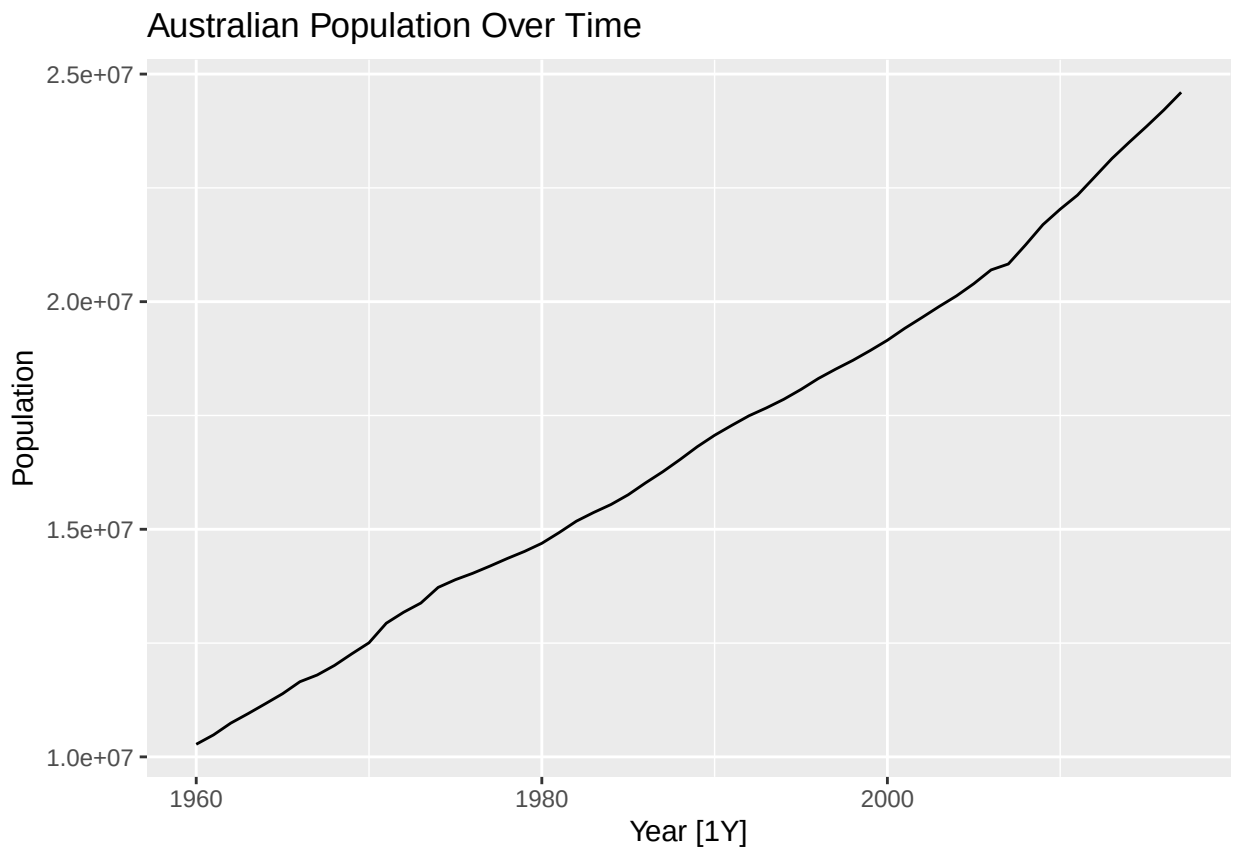
2024-09-21

```
# Load required libraries

library(fpp3)
library(tsibble)
library(ggplot2)
```

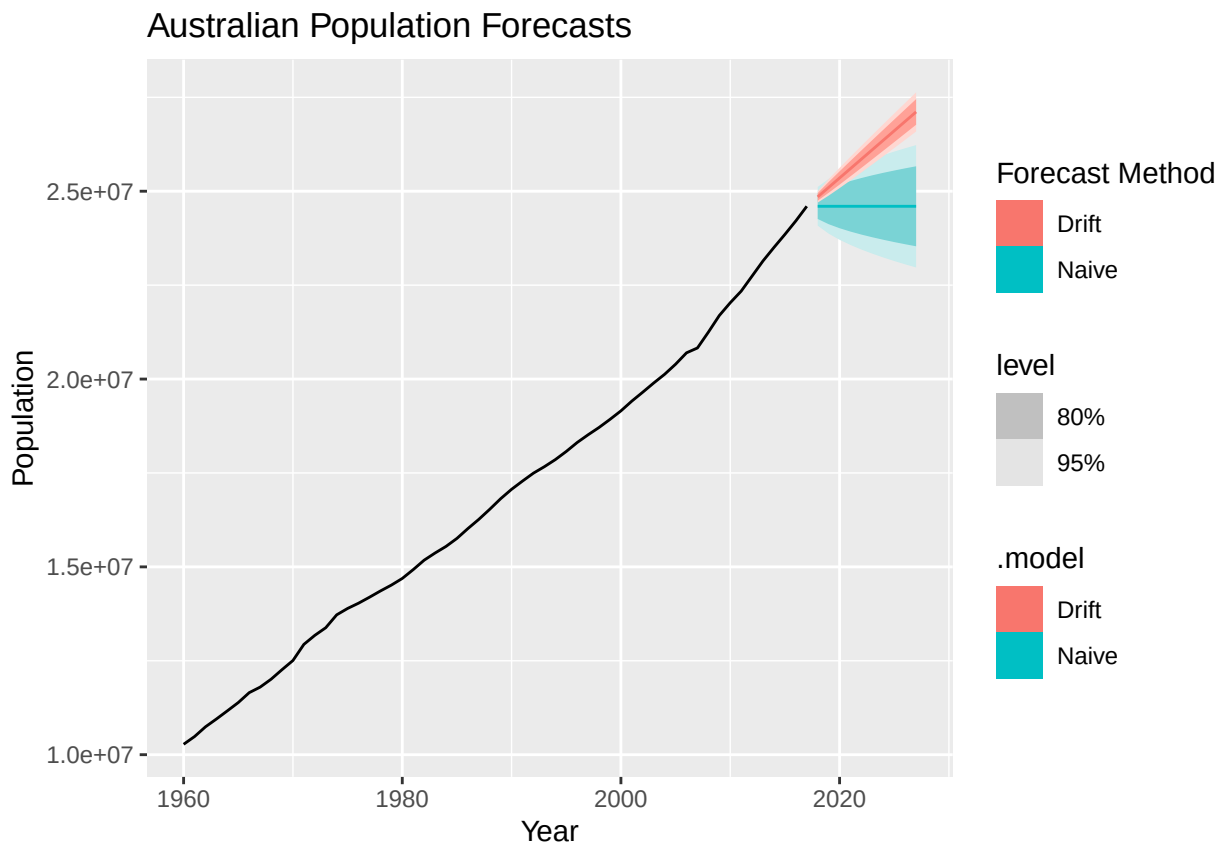
1. Produce forecasts for the following series using whichever of NAIVE(y), SNAIVE(y) or RW(y ~ drift()) is more appropriate in each case:

Australian Population (global_economy)

```
# Filter the data for Australia
aus_data <- global_economy |>
  filter(Country == "Australia")

# Visualize the data to assess trend and seasonality (optional)
aus_data |>
  autoplot(Population) +
  labs(title = "Australian Population Over Time", y = "Population")
```



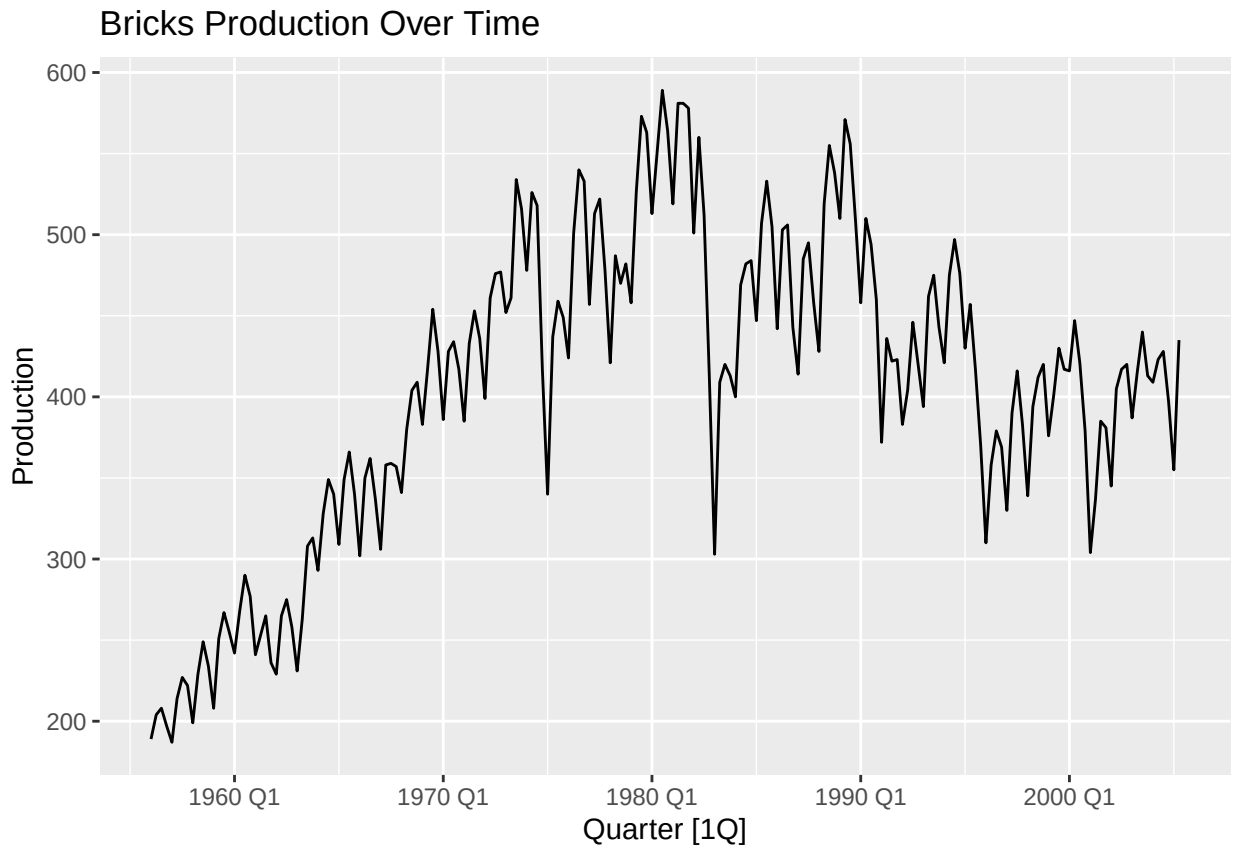

```
# Model the data using appropriate forecasting models
aus_models <- aus_data |>
  model(
    Naive = NAIVE(Population),          # Naive model (no trend or seasonality)
    Drift = RW(Population ~ drift())    # Random walk with drift (appropriate for trending data)
  )

# Produce forecasts for a 10-year horizon
aus_forecasts <- aus_models |>
  forecast(h = "10 years")

# Plot the forecasts
aus_forecasts |>
  autoplot(aus_data) +
  labs(title = "Australian Population Forecasts", y = "Population") +
  guides(colour = guide_legend(title = "Forecast Method"))
```



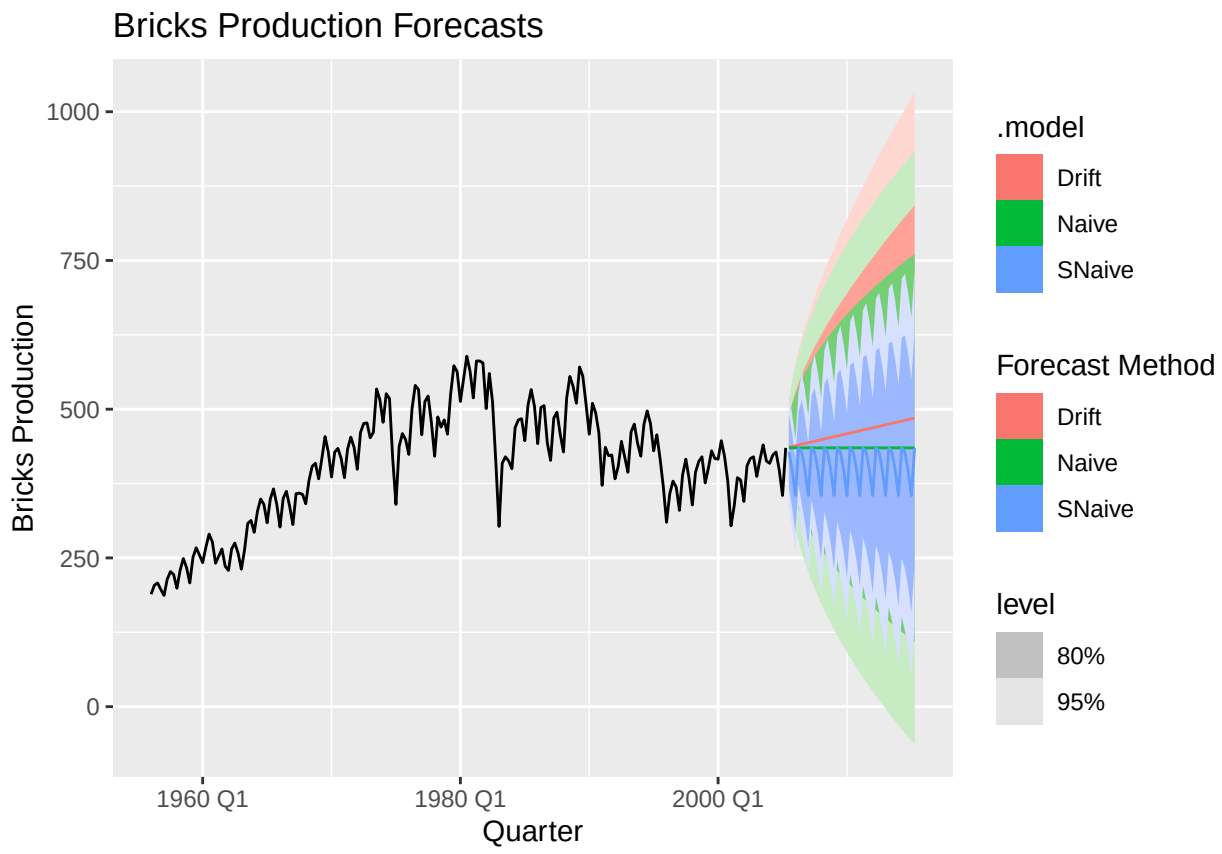
The random walk with drift model (RW(Population ~ drift())) is likely the best option for forecasting the Australian population, as population data usually exhibits a steady upward trend over time.

##**Bricks (aus_production)**

```
# Filter the data for Bricks production and remove missing values
bricks_data <- aus_production |>
  filter(!is.na(Bricks)) |>
  select(Bricks)

# Visualize the Bricks data to check for seasonality and trends (optional)
bricks_data |>
  autoplot(Bricks) +
  labs(title = "Bricks Production Over Time", y = "Production")
```



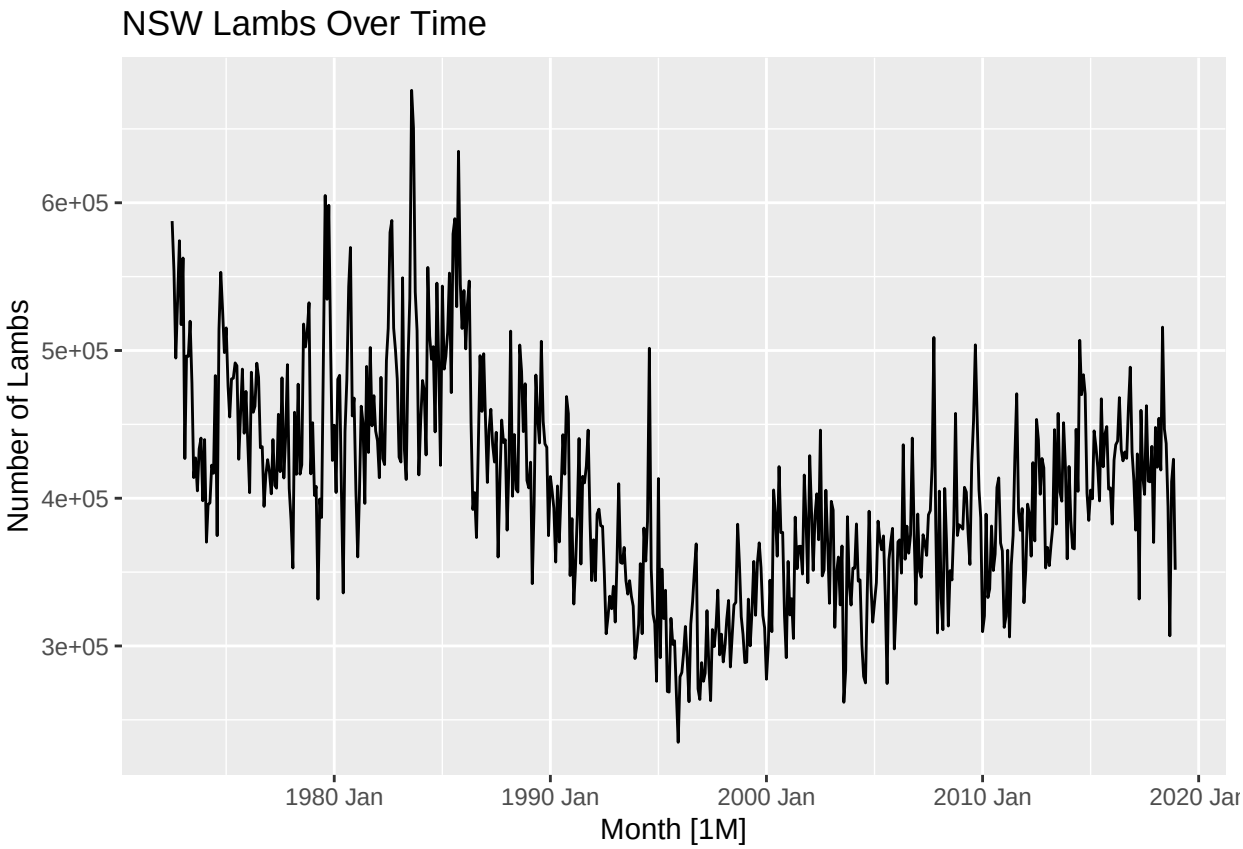

```
# Apply forecasting models
bricks_models <- bricks_data |>
  model(
    Naive = NAIVE(Bricks),              # Naive method (no trend or seasonality)
    SNaive = SNAIVE(Bricks),            # Seasonal naive method (appropriate for seasonal data)
    Drift = RW(Bricks ~ drift())        # Random walk with drift (for trending data)
  )

# Produce forecasts for a given horizon, say 10 periods
bricks_forecasts <- bricks_models |>
  forecast(h = "10 years")

# Plot the forecasts
bricks_forecasts |>
  autoplot(bricks_data) +
  labs(title = "Bricks Production Forecasts", y = "Bricks Production") +
  guides(colour = guide_legend(title = "Forecast Method"))
```



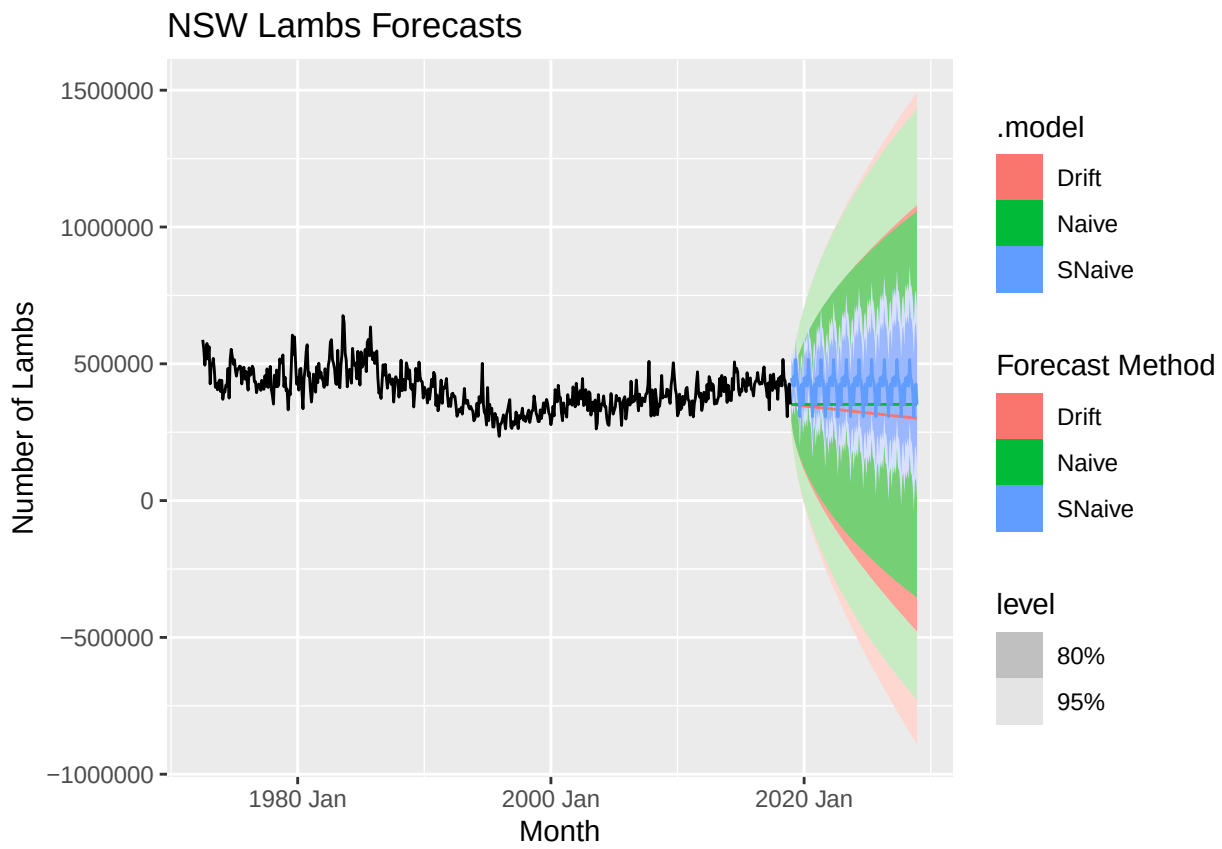
The SNaive model will likely be the best option since the data shows seasonality.

NSW Lambs (aus_livestock)

```
# Filter the data for NSW Lambs
nsw_lambs_data <- aus_livestock |>
  filter(State == "New South Wales", Animal == "Lambs") |>  # Filter for NSW and Lambs
  filter(!is.na(Count))                                      # Remove missing values in the Count colum

# Visualize the data to check for trends and seasonality (optional)
nsw_lambs_data |>
  autoplot(Count) +
  labs(title = "NSW Lambs Over Time", y = "Number of Lambs")
```



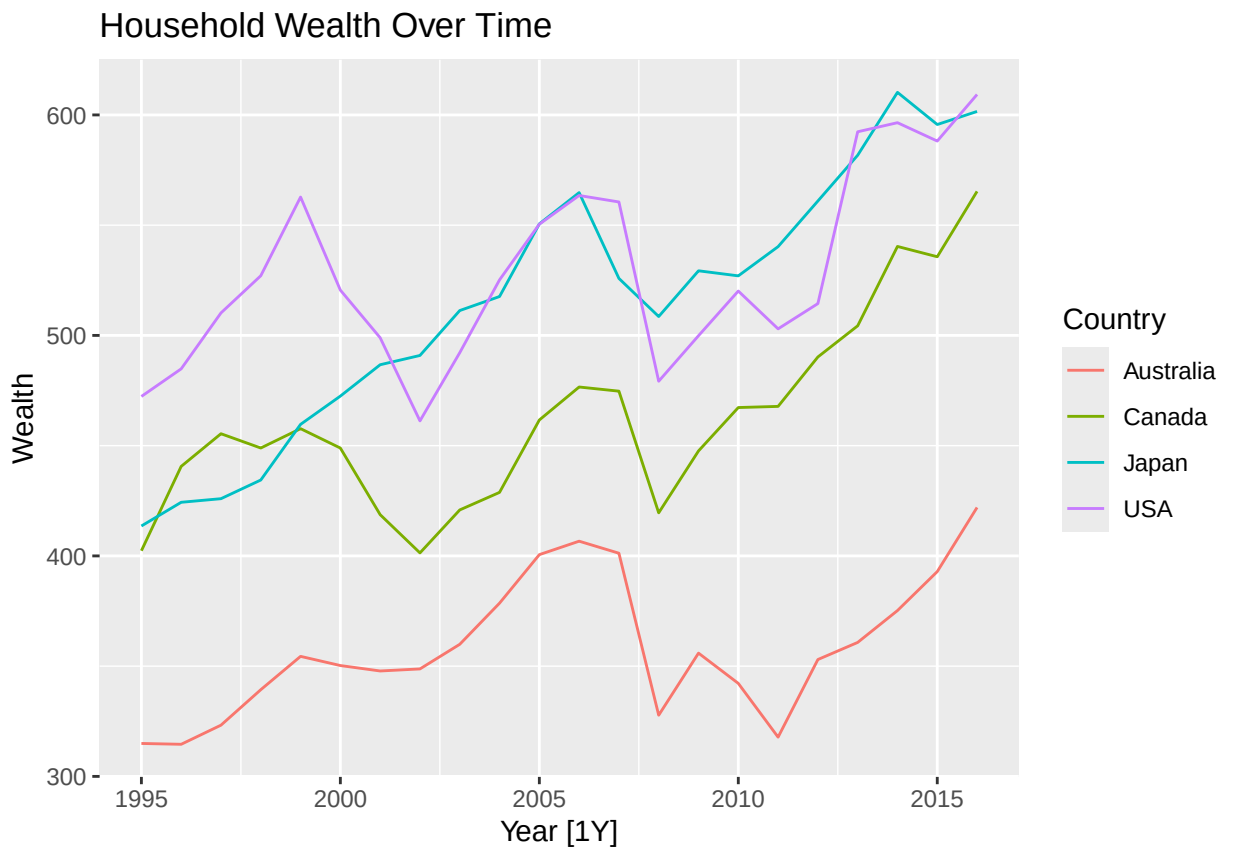

```
# Apply forecasting models
nsw_lambs_models <- nsw_lambs_data |>
  model(
    Naive = NAIVE(Count),                # Naive model (no trend or seasonality)
    SNaive = SNAIVE(Count),              # Seasonal naive model (appropriate for seasonal data)
    Drift = RW(Count ~ drift())          # Random walk with drift (for trending data)
  )

# Produce forecasts for a given horizon, say 10 periods
nsw_lambs_forecasts <- nsw_lambs_models |>
  forecast(h = "10 years")

# Plot the forecasts
nsw_lambs_forecasts |>
  autoplot(nsw_lambs_data) +
  labs(title = "NSW Lambs Forecasts", y = "Number of Lambs") +
  guides(colour = guide_legend(title = "Forecast Method"))
```



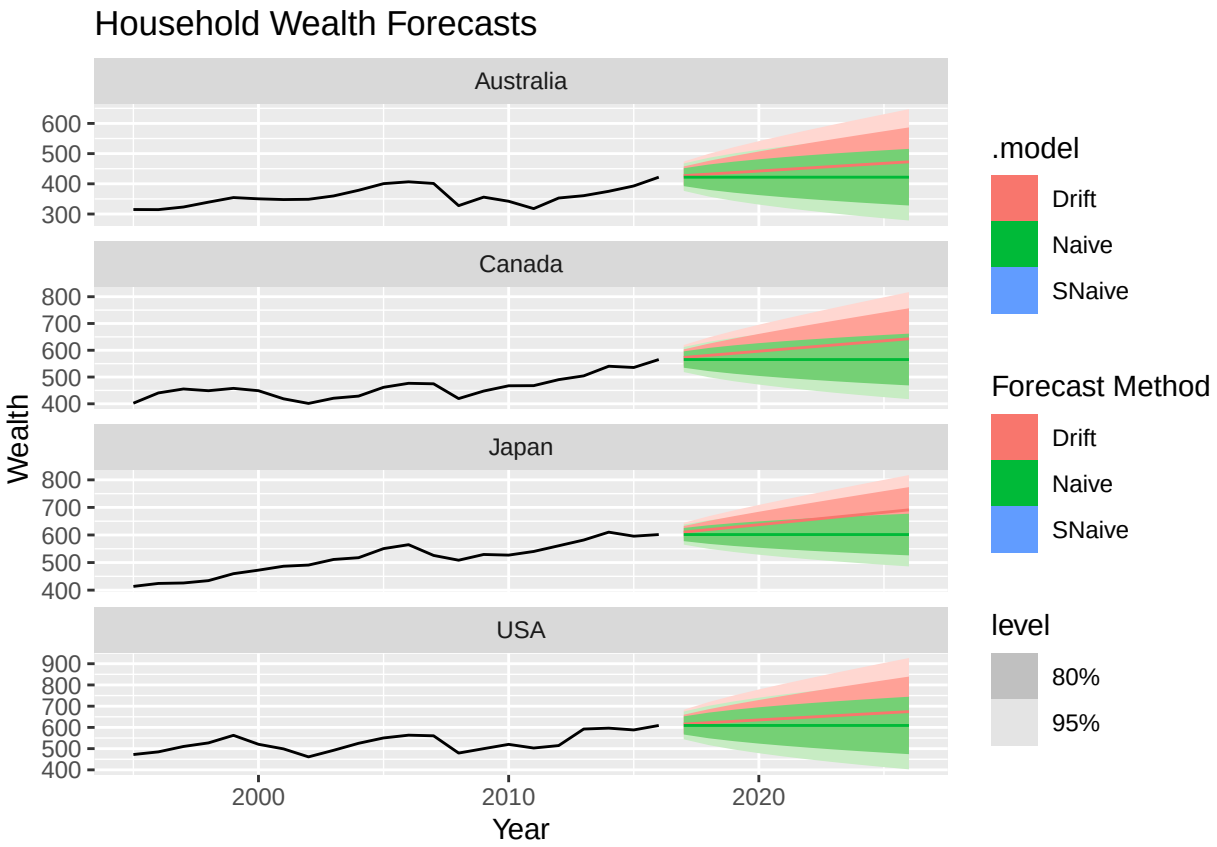
The Seasonal Naive (SNaive) model will probably be the best choice since seasonality is present, which is common in livestock data.

Household wealth (hh_budget)

```
# Filter the Household Wealth data
hh_wealth_data <- hh_budget |>
  select(Wealth) |>                    # Select the Wealth column
  filter(!is.na(Wealth))                # Remove missing values

# Visualize the Household Wealth data to check for trend and seasonality (optional)
hh_wealth_data |>
  autoplot(Wealth) +
  labs(title = "Household Wealth Over Time", y = "Wealth")
```




```
# Apply forecasting models
hh_wealth_models <- hh_wealth_data |>
  model(
    Naive = NAIVE(Wealth),              # Naive model (no trend or seasonality)
    SNaive = SNAIVE(Wealth),            # Seasonal naive model (appropriate for seasonal data)
    Drift = RW(Wealth ~ drift())        # Random walk with drift (for trending data)
  )

# Produce forecasts for a given horizon, say 10 periods
hh_wealth_forecasts <- hh_wealth_models |>
  forecast(h = "10 years")

# Plot the forecasts
hh_wealth_forecasts |>
  autoplot(hh_wealth_data) +
  labs(title = "Household Wealth Forecasts", y = "Wealth") +
  guides(colour = guide_legend(title = "Forecast Method"))
```

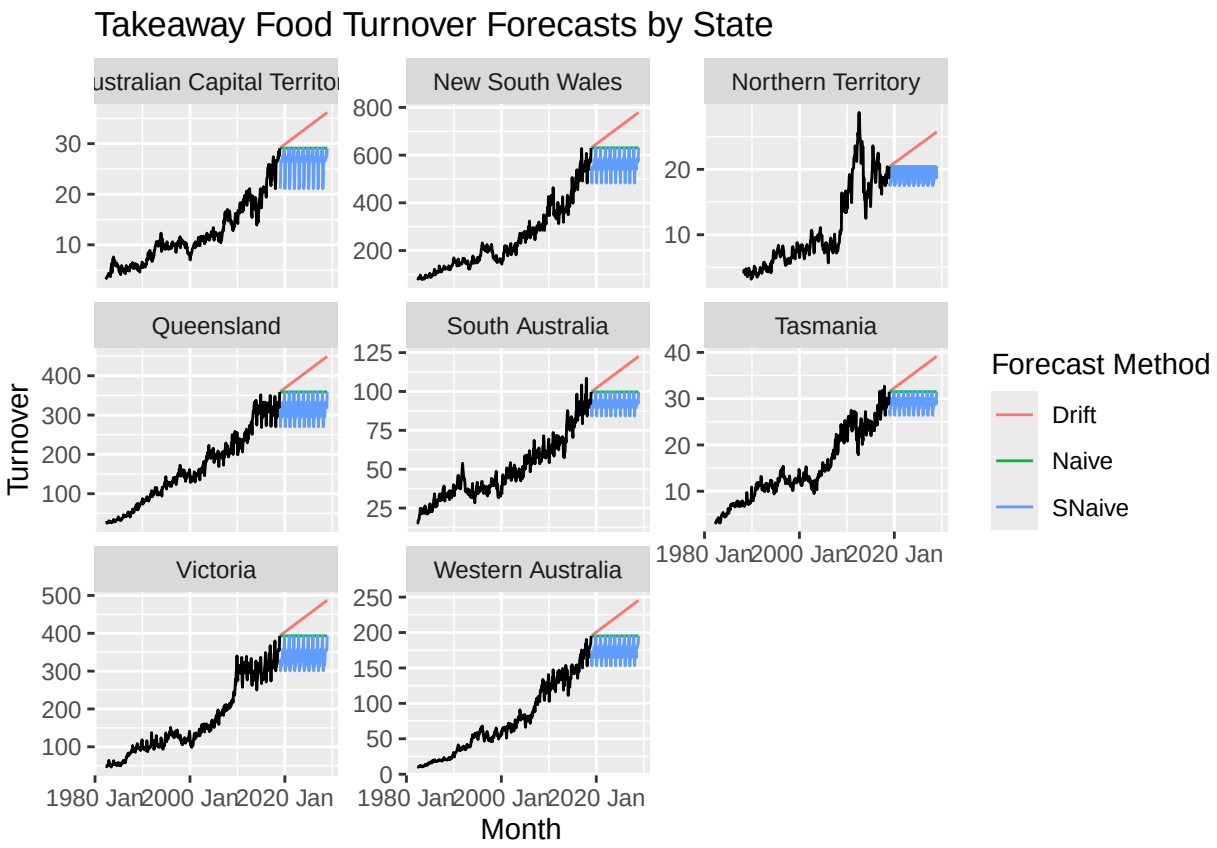


Drift model is the most suitable for forecasting Household Wealth due to its ability to capture the consistent upward or downward trend over time, making it ideal for this series.

Australian takeaway food turnover (aus_retail)

```
# Filter the data for takeaway food services
takeaway_data <- aus_retail |>
  filter(Industry == "Takeaway food services") |>
  filter(!is.na(Turnover))                  # Remove missing values

# Group by State and apply forecasting models for each state
takeaway_forecasts <- takeaway_data |>
  group_by(State) |>
  model(
    Naive = NAIVE(Turnover),                # Naive model (no trend or seasonality)
    SNaive = SNAIVE(Turnover),              # Seasonal naive model (appropriate for seasonal data)
    Drift = RW(Turnover ~ drift())          # Random walk with drift (for trending data)
  ) |>
  forecast(h = "10 years")                  # Forecast for 10 years

# Plot the forecasts for each state
takeaway_forecasts |>
  autoplot(takeaway_data, level = NULL) +  # Plot forecasts
  labs(title = "Takeaway Food Turnover Forecasts by State", y = "Turnover") +
```

```
  facet_wrap(~ State, scales = "free_y") +  # Separate plot for each state
  guides(colour = guide_legend(title = "Forecast Method"))
```



For the Australian takeaway food turnover across different states, the Seasonal Naive (SNaive) model is the most appropriate if the data shows clear seasonal patterns like most of the states, which is common in retail industries. However, in states where only a trend is observed, for example Northern Territory, the Random Walk with Drift model will be more suitable. By applying these models to each state, we can capture both seasonal effects and trends, providing accurate forecasts for each state's takeaway food turnover over the next 10 years.

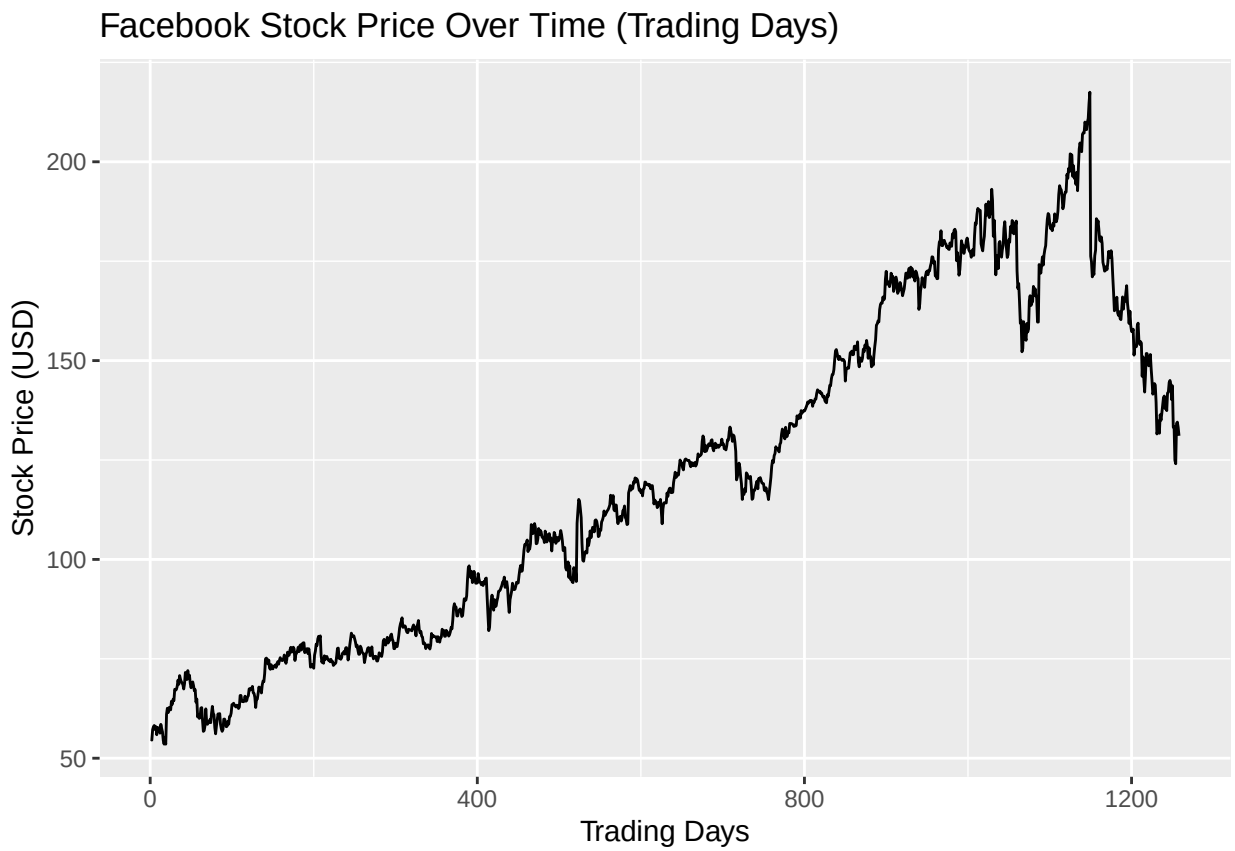
2. Use the Facebook stock price (data set gafa_stock) to do the following:

a. Produce a time plot of the series.

Because stock prices are not observed every day, we first set up a new time index based on the trading days rather than calendar days. This is more appropriate for analyzing stock prices since trading doesn't happen every day of the year.

```
# Filter the data for Facebook (Meta) stock
fb_stock <- gafa_stock |>
  filter(Symbol == "FB") |>   # Filter for Facebook stock
  mutate(day = row_number()) |>
  update_tsibble(index = day, regular = TRUE)
```

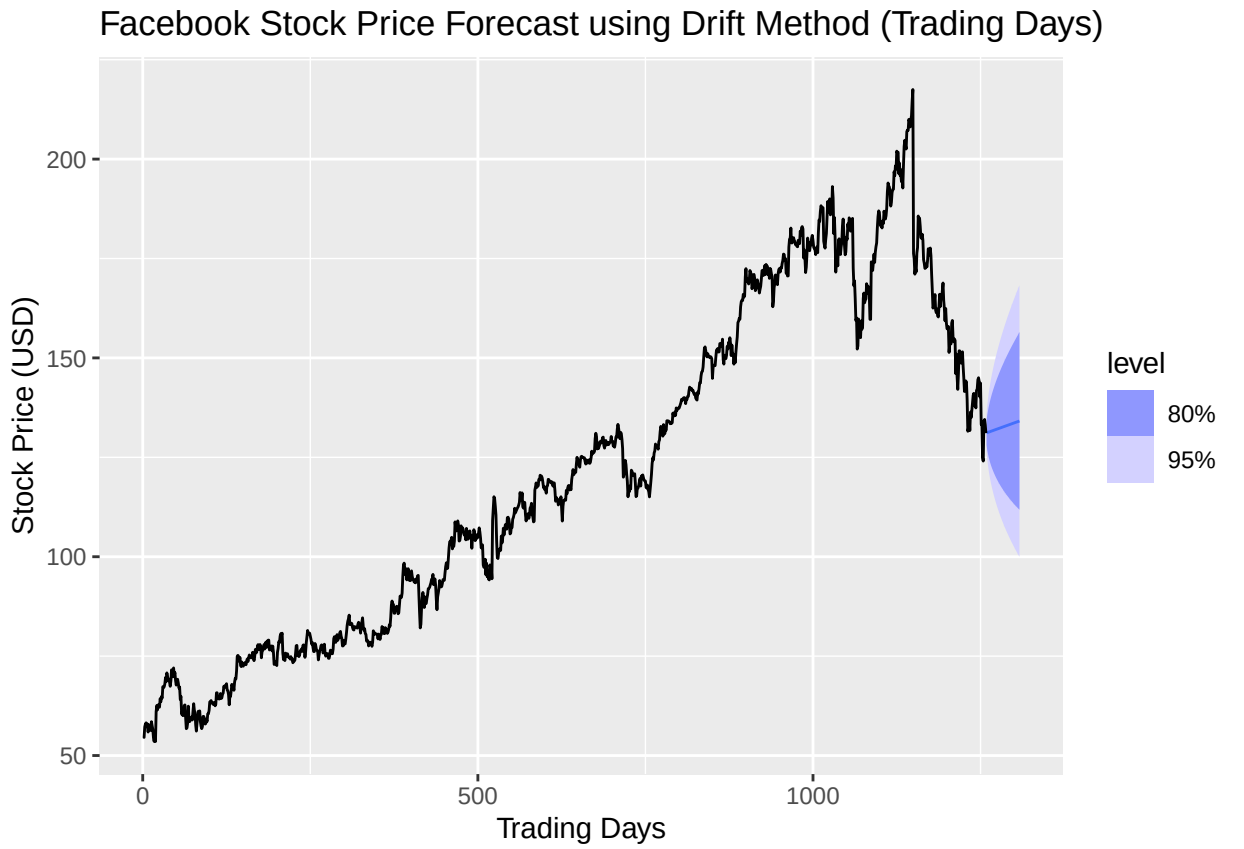
```r
# Produce a time plot of the stock price
fb_stock |>
  autoplot(Close) +                          # Plot the closing price over time
  labs(title = "Facebook Stock Price Over Time (Trading Days)", y = "Stock Price (USD)", x = "Trading Da
```



b. Produce forecasts using the drift method and plot them.

```r
# Filter the data for Facebook (Meta) stock and set a new time index based on trading days
fb_stock <- gafa_stock |>
  filter(Symbol == "FB") |>                  # Filter for Facebook stock
  mutate(day = row_number()) |>              # Create a new trading day index
  update_tsibble(index = day, regular = TRUE)  # Update the tsibble to use trading days as index

# Model the data using the Drift method
fb_drift_model <- fb_stock |>
  model(Drift = RW(Close ~ drift()))         # Apply the drift model on the closing price

# Produce forecasts for a specified horizon (e.g., 50 trading days)
fb_forecasts <- fb_drift_model |>
  forecast(h = 50)                           # Forecast for 50 trading days into the future

# Plot the forecasts alongside the historical data
fb_forecasts |>
```

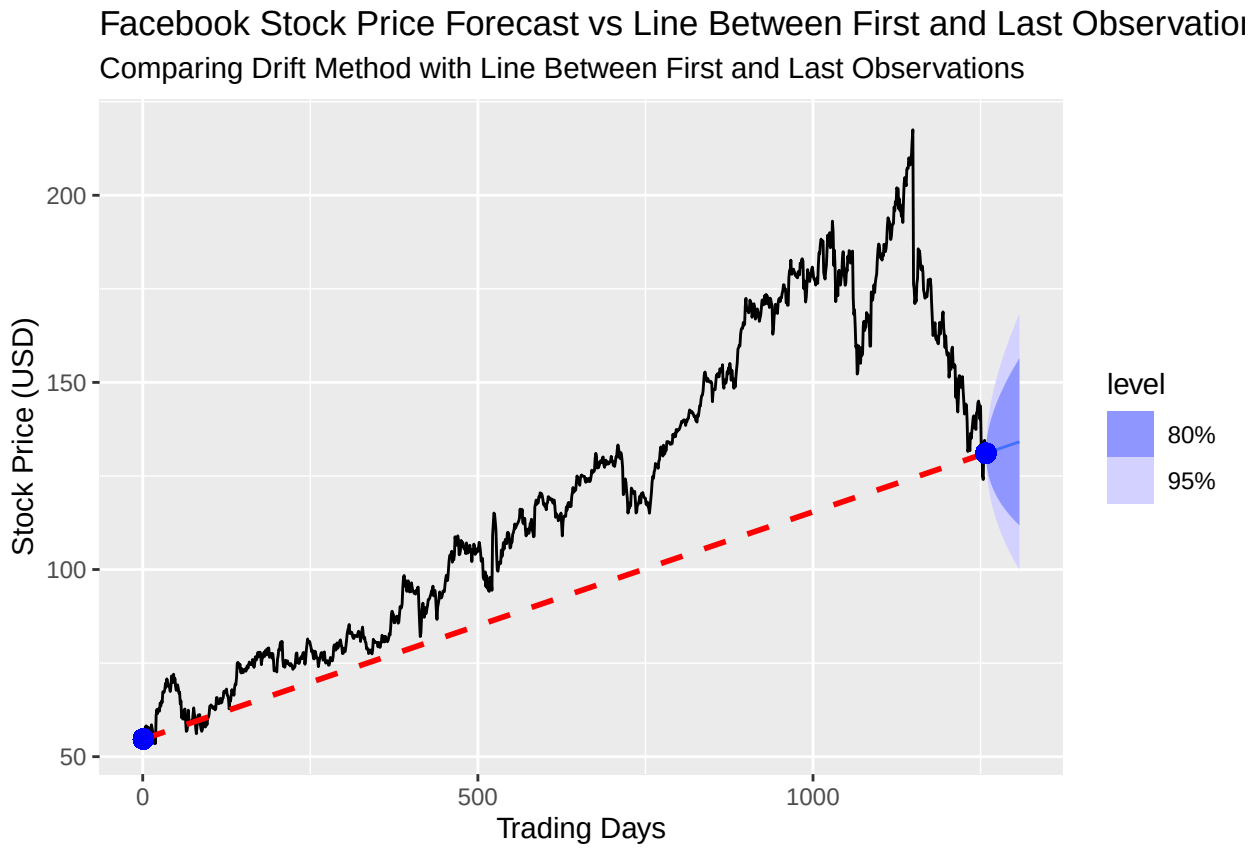
```
  autoplot(fb_stock) +                          # Plot the forecast with historical data
  labs(title = "Facebook Stock Price Forecast using Drift Method (Trading Days)",
       y = "Stock Price (USD)", x = "Trading Days") +
  guides(colour = guide_legend(title = "Forecast Method"))
```



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

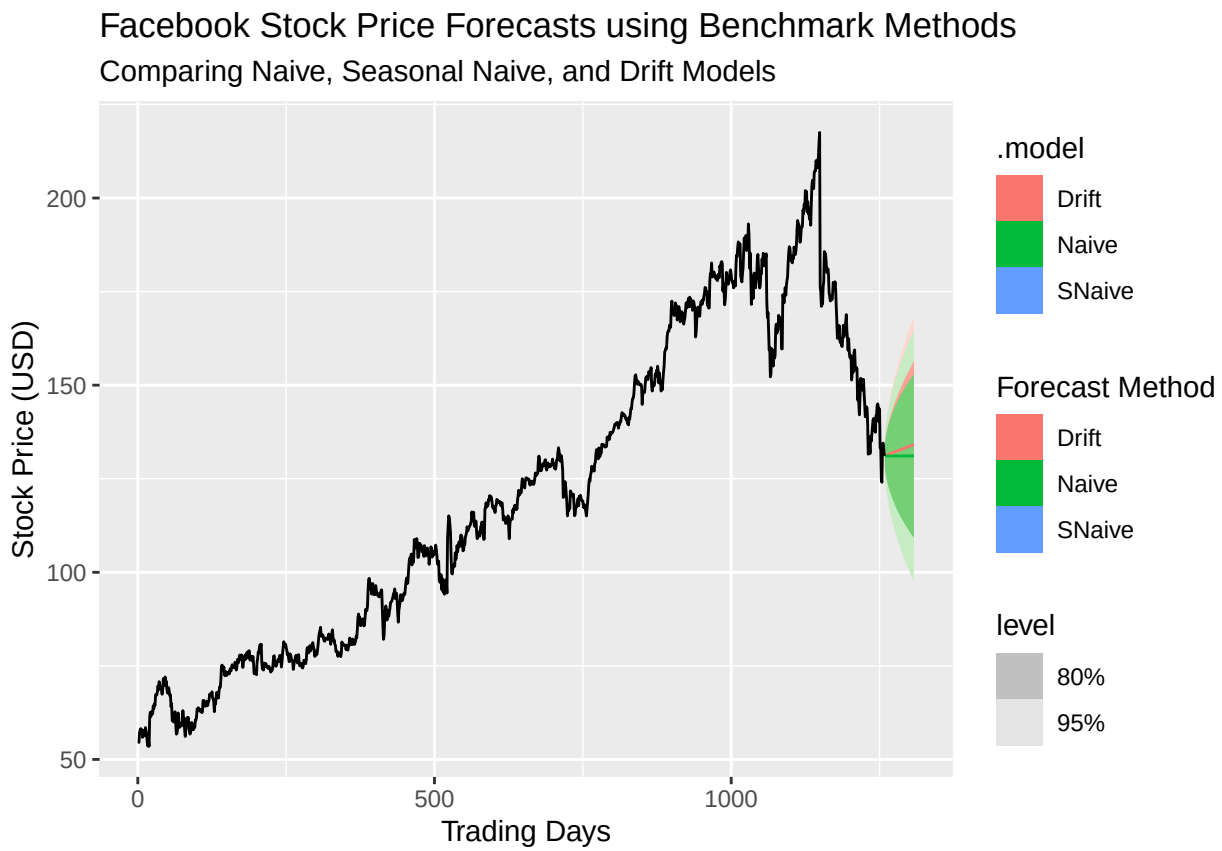
```
# Filter the data for Facebook (Meta) stock and set a new time index based on trading days
fb_stock <- gafa_stock |>
  filter(Symbol == "FB") |>                 # Filter for Facebook stock
  mutate(day = row_number()) |>             # Create a new trading day index
  update_tsibble(index = day, regular = TRUE)  # Update the tsibble to use trading days as index

# Model the data using the Drift method
fb_drift_model <- fb_stock |>
  model(Drift = RW(Close ~ drift()))        # Apply the drift model on the closing price

# Produce forecasts for a specified horizon (e.g., 50 trading days)
fb_forecasts <- fb_drift_model |>
  forecast(h = 50)                          # Forecast for 50 trading days into the future

# Extract the first and last observations
```

```r
first_day <- min(fb_stock$day)
last_day <- max(fb_stock$day)
first_price <- fb_stock %>% filter(day == first_day) %>% pull(Close)
last_price <- fb_stock %>% filter(day == last_day) %>% pull(Close)

# Create a tibble to represent the line between the first and last points
line_data <- tibble(
  day = c(first_day, last_day),
  Close = c(first_price, last_price)
)

# Plot the forecasts and the line connecting the first and last points
fb_forecasts |>
  autoplot(fb_stock) +                        # Plot the forecast with historical data
  geom_line(aes(x = day, y = Close), data = line_data, color = "red", linetype = "dashed", size = 1) +
  labs(title = "Facebook Stock Price Forecast vs Line Between First and Last Observations",
       subtitle = "Comparing Drift Method with Line Between First and Last Observations",
       y = "Stock Price (USD)", x = "Trading Days") +
  geom_point(aes(x = first_day, y = first_price), color = "blue", size = 3) +    # Mark first point
  geom_point(aes(x = last_day, y = last_price), color = "blue", size = 3) +     # Mark last point
  guides(colour = guide_legend(title = "Forecast Method"))
```



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

```
# Filter the data for Facebook (Meta) stock and set a new time index based on trading days
fb_stock <- gafa_stock |>
  filter(Symbol == "FB") |>                # Filter for Facebook stock
  mutate(day = row_number()) |>            # Create a new trading day index
  update_tsibble(index = day, regular = TRUE)  # Update the tsibble to use trading days as index

# Apply different benchmark models (Naive, Seasonal Naive, and Drift)
fb_models <- fb_stock |>
  model(
    Naive = NAIVE(Close),                  # Naive method
    SNaive = SNAIVE(Close),                # Seasonal Naive method
    Drift = RW(Close ~ drift())            # Drift method
  )

# Produce forecasts for a specified horizon (e.g., 50 trading days)
fb_forecasts <- fb_models |>
  forecast(h = 50)

# Plot the forecasts
fb_forecasts |>
  autoplot(fb_stock) +                     # Plot the forecast with historical data
  labs(title = "Facebook Stock Price Forecasts using Benchmark Methods",
       subtitle = "Comparing Naive, Seasonal Naive, and Drift Models",
       y = "Stock Price (USD)", x = "Trading Days") +
  guides(colour = guide_legend(title = "Forecast Method"))
```



Naive Method (NAIVE(Close)): This method assumes that future stock prices will be the same as the last observed price. This is often used as a simple benchmark to see if more complex models outperform simply assuming the status quo will continue.

Seasonal Naive Method (SNAIVE(Close)): This method repeats past seasonal patterns into the future. Since stock prices typically don't exhibit seasonality (they follow more stochastic patterns), this method might not be appropriate for the Facebook stock prices.

Drift Method (RW(Close ~ drift())): This method assumes that the stock price follows a random walk but includes a drift, which is the trend between the first and last observations. Stock prices tend to have trends (upward or downward), so this model is often a good fit for financial data.

Drift Method: This model is typically the best for stock prices, as it captures the overall trend and projects it into the future. Stock prices tend to follow a random walk with a drift, reflecting the general increase or decrease in value over time due to various factors (e.g., company performance, economic conditions).

Naive Method: While simple, the Naive method does not capture trends or other dynamics. It assumes the stock price will remain the same as the last observation, which is unlikely for financial data.

Seasonal Naive Method: This method assumes that the stock price follows a seasonal pattern, which is not characteristic of financial markets. Hence, this is not suitable for stock prices unless specific known seasonal effects exist (which typically don't apply to stock prices).

Which Model is Best?

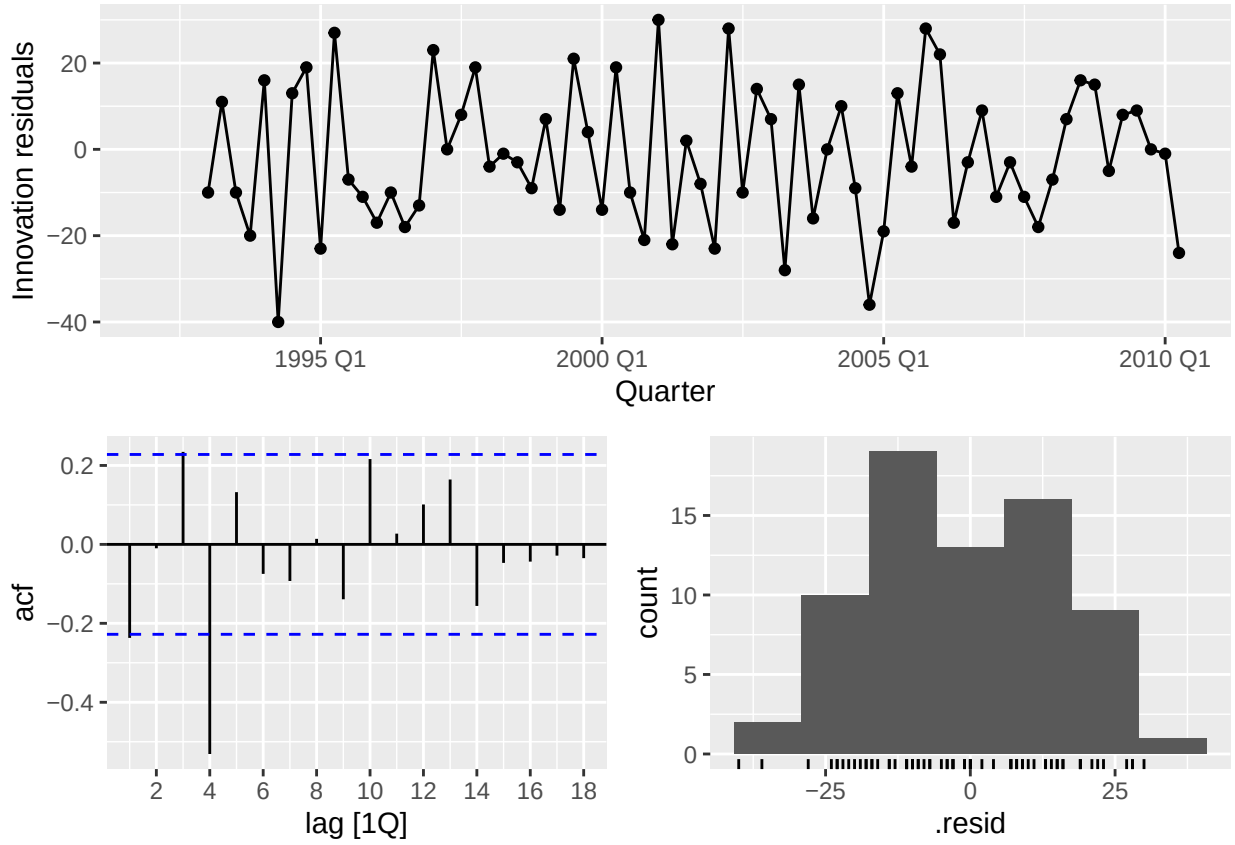
Drift Method is likely the best choice because it incorporates trends, which are a fundamental feature of stock prices.

The Naive method can serve as a baseline for comparison but is less informative than the Drift model.

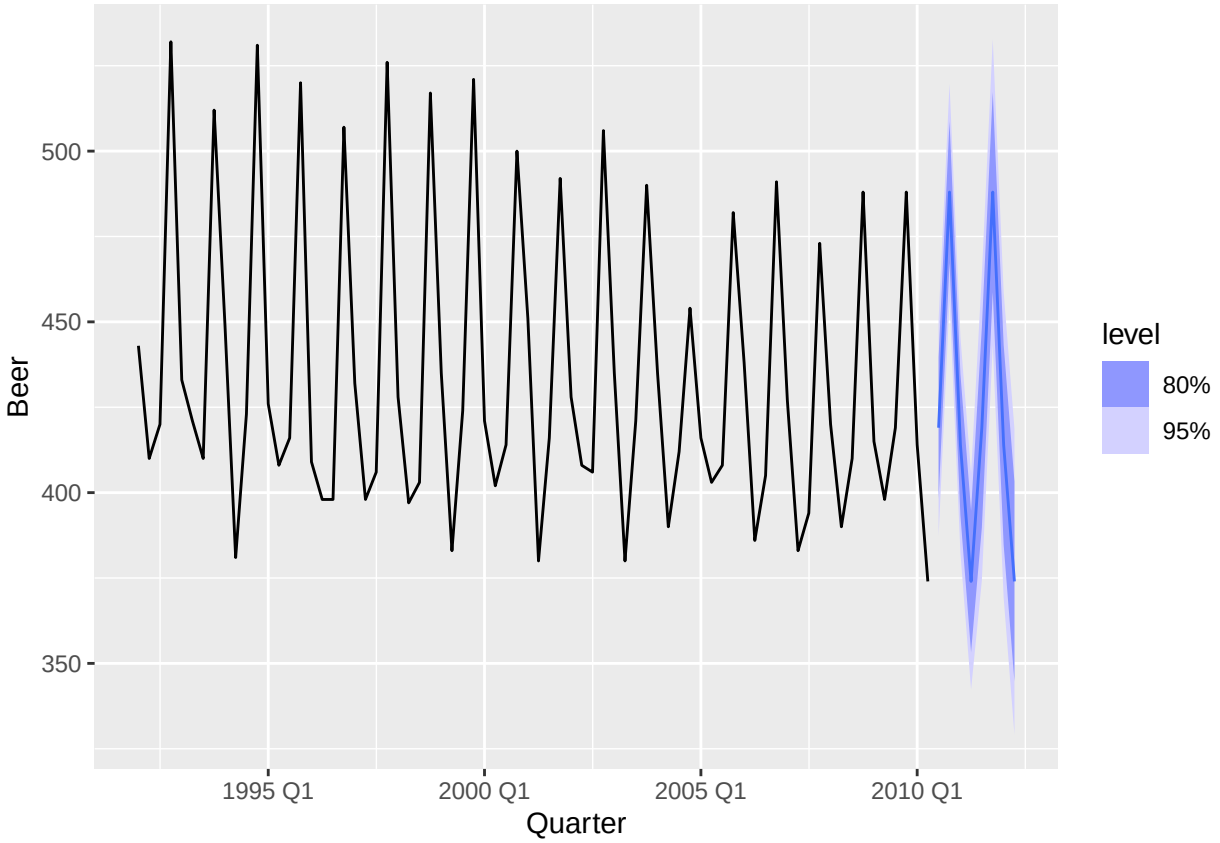
The Seasonal Naive method is not appropriate for stock price data as financial data doesn't typically exhibit seasonality.

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. 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?

Innovation residuals: The residuals plot shows fluctuations that do not appear random. There is a noticeable pattern in the residuals, indicating that the model has not fully captured all the underlying structure in the data.

Autocorrelation Function - ACF: The ACF plot shows significant autocorrelation at several lags, which suggests that the residuals are not white noise. For a good model fit, residuals should show no significant autocorrelation.

Histogram: The histogram of residuals shows some symmetry around zero, which is a good sign, but the wide spread and potential outliers indicate that the model has struggled to capture all the dynamics of the data. The residuals don't follow a normal distribution, and the presence of large residuals suggests some predictions are off, likely due to patterns not captured by the Seasonal Naive method. Overall, while the model captures seasonality, a more sophisticated model (e.g., SARIMA) may better account for trend and other variations.

The forecast plot extends the seasonal pattern into the future, as expected from the Seasonal Naive method. The forecast produces a regular cyclical pattern with confidence intervals (80% and 95%) showing that the future predictions are likely to follow the same seasonal pattern but with increasing uncertainty.

Residual Analysis: The residuals do not resemble white noise because there are visible patterns and significant autocorrelations. This suggests that the Seasonal Naive model does not capture all the structure in the data, possibly missing a trend or some other dynamic component.

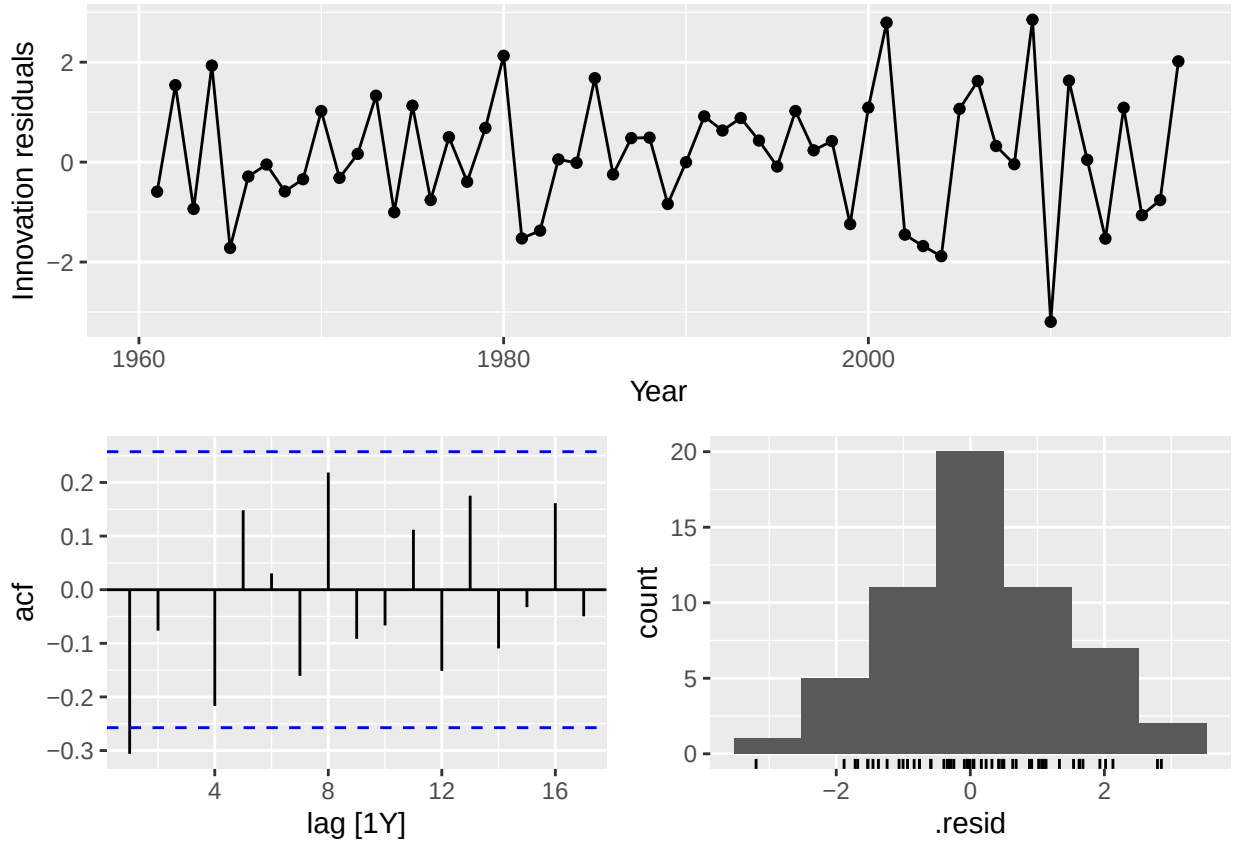
Forecast Accuracy: The Seasonal Naive method does reasonably well at capturing the strong seasonal pattern in beer production. However, given that the residuals are not white noise, a more sophisticated model might be needed to capture any additional trends or autocorrelations.

We might consider using more advanced models, such as an ARIMA model with seasonal components, which might better account for both trend and seasonality, leading to improved residual behavior.

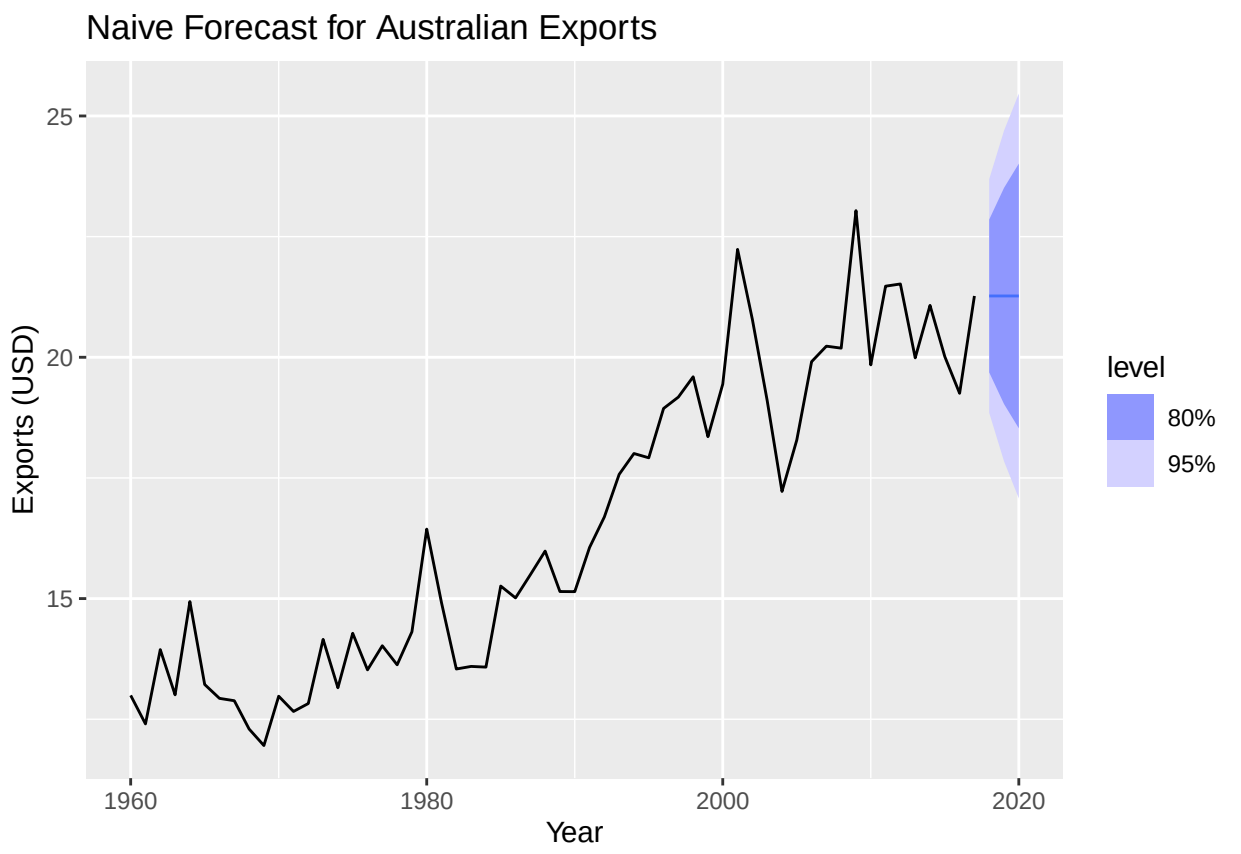
4. Repeat the previous exercise using the Australian Exports series from global_economy and the Bricks series from aus_production. Use whichever of NAIVE() or SNAIVE() is more appropriate in each case.

##Australian Exports series from global_economy

```
# Extract the Australian Exports data
aus_exports <- global_economy |>
  filter(Country == "Australia") |>     # Filter for Australian data
  select(Year, Exports) |>              # Select Year and Exports columns
  filter(!is.na(Exports))               # Remove rows with missing export values

# Apply a Naive model (suitable for yearly data)
fit_naive <- aus_exports |>
  model(NAIVE(Exports))  # Naive model

# Apply a Drift model (suitable for yearly data with a trend)
fit_drift <- aus_exports |>
  model(RW(Exports ~ drift()))  # Drift model

# Check the residuals for the Naive model
fit_naive |> gg_tsresiduals()
```



```r
# Plot forecasts for the Naive model (forecasting 3 years into the future)
fit_naive |>
  forecast(h = "3 years") |>
  autoplot(aus_exports) +
  labs(title = "Naive Forecast for Australian Exports",
       y = "Exports (USD)", x = "Year")
```



Innovation Residuals Plot: The residuals appear to fluctuate randomly, with no clear trend over time. However, there is some volatility in certain periods (e.g., spikes around the 1970s and late 2000s). This suggests that while the Naive model captures some aspects of the data, there may be periods where the model doesn't perform well.

ACF Plot: There are small spikes in autocorrelation at various lags, but most of them stay within the blue confidence bounds. This indicates that while there is some autocorrelation, it's not significant enough to suggest strong remaining patterns in the residuals. Ideally, the ACF should show no significant autocorrelation if the residuals are white noise.

Residuals Histogram: The histogram of residuals is approximately symmetric, centered around zero, which is a good sign. However, the spread of the residuals is not very narrow, indicating the model has some errors in both directions. This may suggest that the model could be improved, but it's not drastically off.

The Naive forecast simply assumes that future values will stay the same as the most recent observation (around 2020). The forecast shows a flat line with a constant value, and the confidence intervals (shaded in blue) show increasing uncertainty as we forecast further into the future. This is expected behavior for the Naive model since it does not incorporate any trend or seasonality.

Residual Analysis: The residuals are mostly random, with no significant autocorrelation, which suggests that the Naive model captures some of the data dynamics. However, the residuals' volatility in certain periods

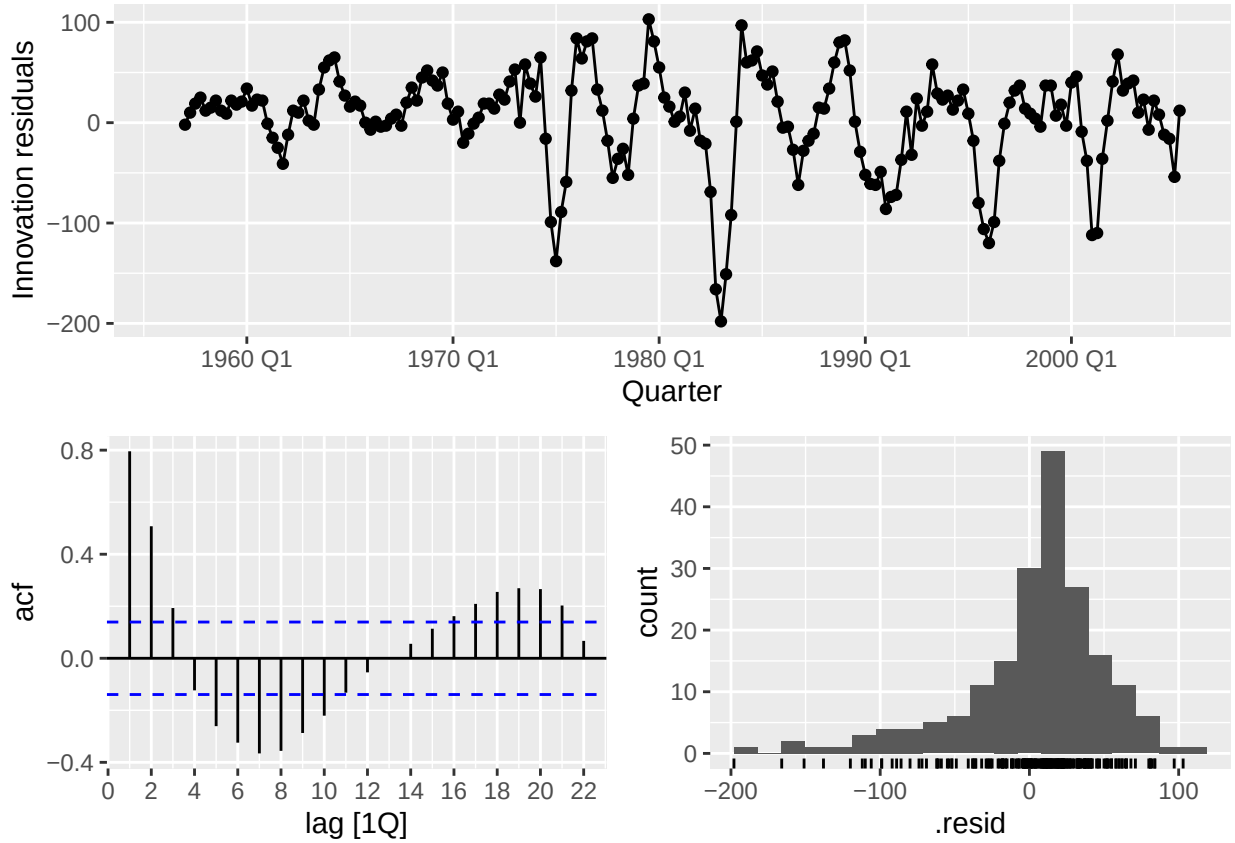
(especially the spikes) suggests that the model could be refined to better capture these periods.

Forecast: The Naive model is very simplistic—it assumes no change in future export values, which might not be realistic for a dataset that shows an increasing trend, like the Australian exports data.

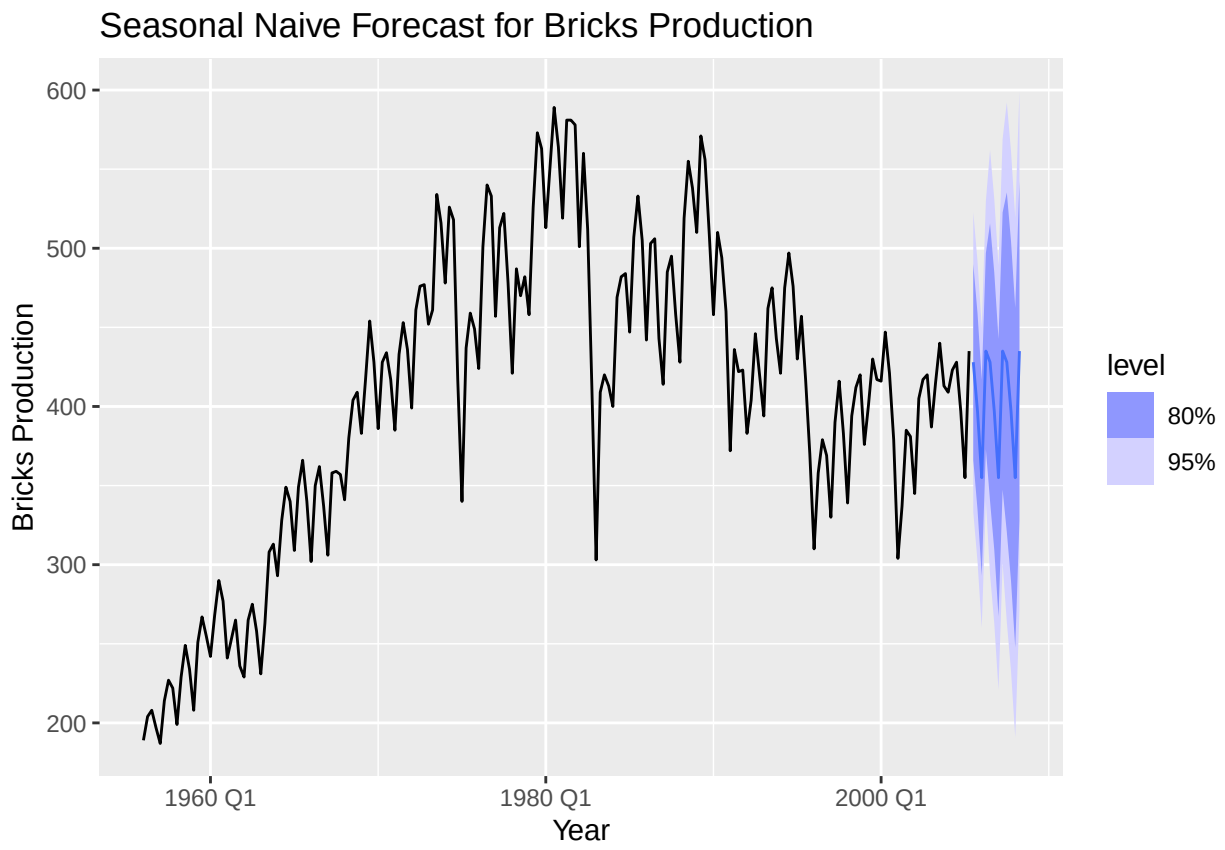
Given that the exports data seems to have a clear upward trend, applying a Drift model might produce more accurate forecasts by accounting for the trend. We might using models that capture both trend and seasonality, such as ARIMA or ETS, to improve forecasting accuracy.

Bricks series from aus_production

```
# Extract the Bricks series from the aus_production dataset
bricks_data <- aus_production |>
  select(Quarter, Bricks) |>              # Select the Quarter and Bricks data
  filter(!is.na(Bricks))                  # Remove rows with missing values

# Choose the appropriate model based on seasonality (use SNAIVE if seasonality is present)
# Applying SNAIVE model (assuming seasonality)
fit_bricks <- bricks_data |>
  model(SNAIVE(Bricks))

# Check the residuals to see if they resemble white noise
fit_bricks |> gg_tsresiduals()
```



```r
# Produce and plot forecasts (e.g., forecast for 3 years)
fit_bricks |>
  forecast(h = "3 years") |>
  autoplot(bricks_data) +
  labs(title = "Seasonal Naive Forecast for Bricks Production",
       y = "Bricks Production", x = "Year")
```



Innovation Residuals: The residuals show clear patterns over time. There's a noticeable cyclical behavior, indicating that the model hasn't fully captured the data's structure. The residuals do not appear to be random, suggesting that the Seasonal Naive model may not have captured all of the underlying seasonality or trend in the data.

ACF Plot: There is significant autocorrelation in the residuals at multiple lags, especially up to lag 8. This indicates that the residuals are not white noise, meaning the model has left some structure in the data unexplained. Ideally, we would expect the ACF values to fall within the blue confidence bounds, but this isn't the case here, indicating room for model improvement.

Residuals Histogram: The histogram is somewhat symmetric, but the wide spread of residuals (with some large residuals) suggests that the model is missing some key patterns in the data. There is a clustering of values around 0, but the tails of the distribution extend out, indicating that the model has larger errors for some periods.

Forecast Plot: The Seasonal Naive forecast shows repeated seasonal patterns for future values, as expected. The confidence intervals (80% and 95%) widen significantly over time, reflecting the uncertainty in the forecast. The forecast shows a relatively flat pattern compared to historical data, which had higher variability in production levels. This might indicate that the Seasonal Naive model is not well-suited to capture changes in trend.

Residuals: The presence of clear patterns and significant autocorrelation in the residuals suggests that the Seasonal Naive model isn't fully capturing the dynamics of the bricks production data. The model may need to account for more complex trend components or other cyclical patterns beyond seasonality.

Forecast: While the Seasonal Naive model produces a reasonable forecast by repeating past seasonal patterns, it doesn't capture the variability and trends in the historical data. The wide confidence intervals also suggest that the model isn't confident in its predictions over time.

Would consider using a SARIMA (Seasonal ARIMA) model or ETS (Error, Trend, Seasonality) model, which could better account for both the trend and seasonal components, as well as address the issues observed in the residuals. If seasonality is strong, it might help to explore more sophisticated seasonal adjustment techniques that can better handle variability within the series.

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

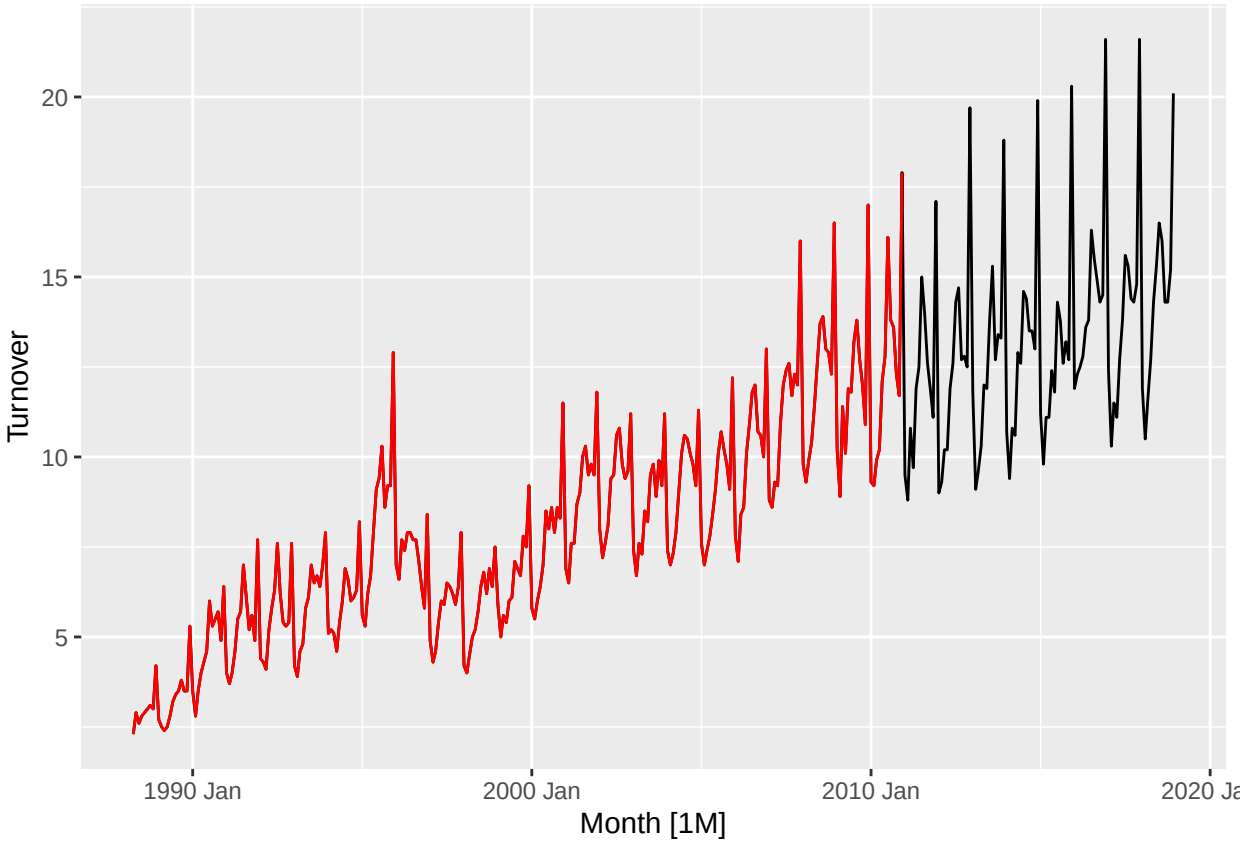
```
set.seed(12345678)
myseries <- aus_retail |>
  filter(`Series ID` == sample(aus_retail$`Series ID`,1))
```

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

```
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")
```

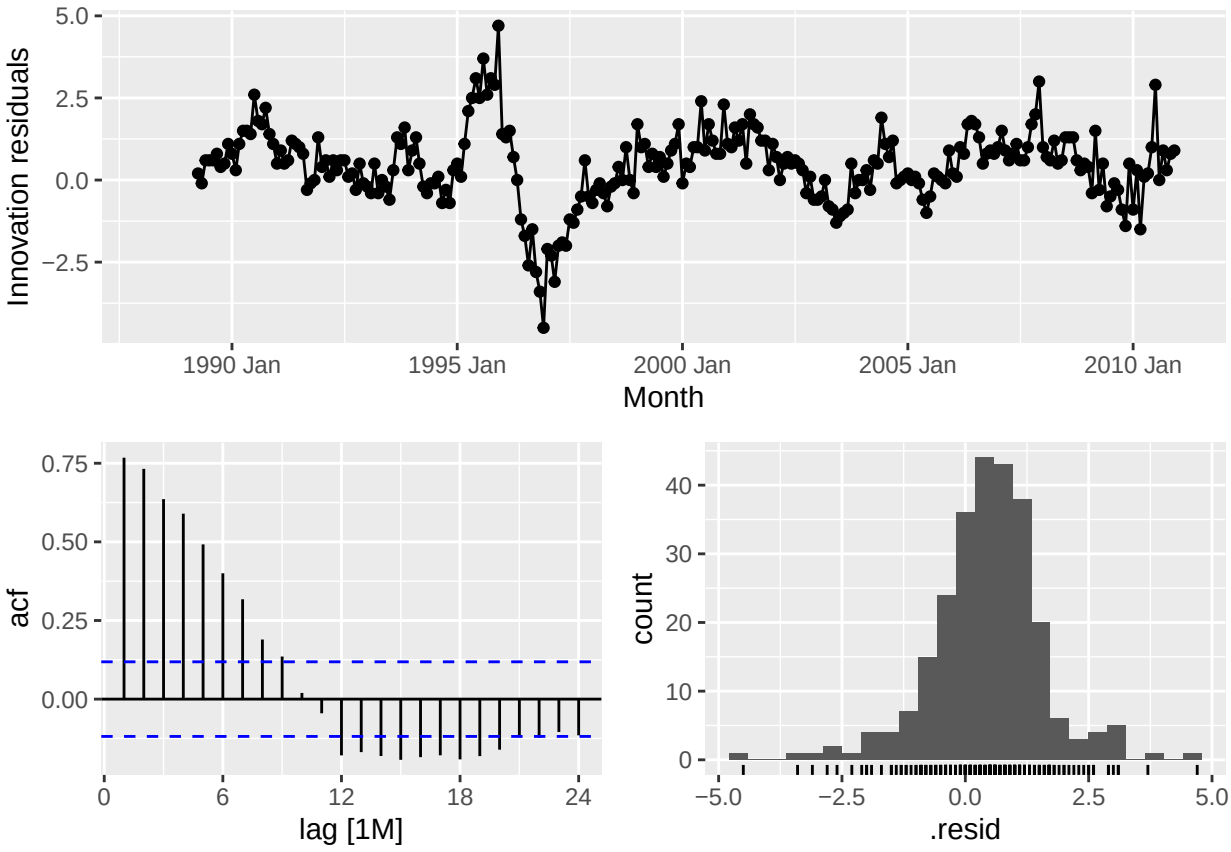


c. Fit a seasonal naïve model using SNAIVE() applied to your training data (myseries_train).

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

d. Check the residuals.

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



Do the residuals appear to be uncorrelated and normally distributed?

The ACF plot shows significant autocorrelation at early lags (up to lag 6), which indicates that the residuals are not fully uncorrelated. Ideally, uncorrelated residuals should show no significant autocorrelation (the bars in the ACF plot should fall within the blue bounds). Since they don't, this suggests that the model has left some patterns in the data unaccounted for.

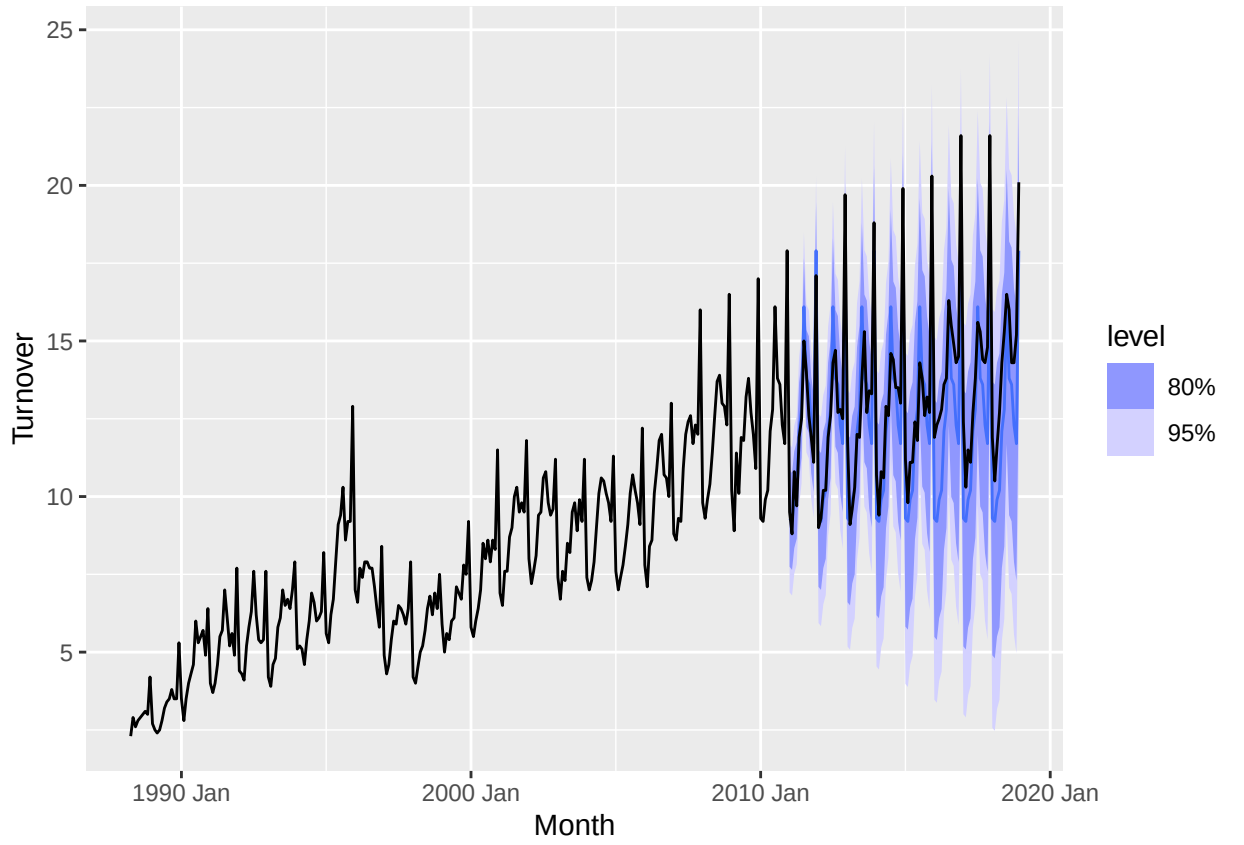
The histogram of residuals is approximately symmetric and somewhat bell-shaped, which is a good sign of normality. However, this alone doesn't fully confirm normality—it just gives an indication. There may still be some issues, as the residuals are tightly clustered around zero with some large outliers, suggesting that the model struggles to capture all the variations in the data.

Residuals are not fully uncorrelated due to the significant autocorrelation at early lags.

The residuals are roughly normally distributed, but the presence of outliers suggests that the model may not be fully adequate in capturing all patterns in the data.

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()
```

```
## # A tibble: 1 x 12
##   State    Industry .model .type    ME  RMSE   MAE   MPE  MAPE  MASE RMSSE  ACF1
##   <chr>    <chr>    <chr>  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Norther~ Clothin~ SNAIV~ Trai~ 0.439  1.21 0.915  5.23  12.4     1     1 0.768
```

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

```
## # A tibble: 1 x 12
##   .model    State Industry .type    ME  RMSE   MAE   MPE  MAPE  MASE RMSSE  ACF1
##   <chr>     <chr> <chr>    <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 SNAIVE(T~ Nort~ Clothin~ 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?

Mean Error (ME): Training Set: 0.4387 Test Set: 0.8365 The ME increased in the test set, indicating that the model overestimated the values more during testing than training.

RMSE (Root Mean Squared Error): Training Set: 1.2137 Test Set: 1.5520 The RMSE increased from training to test, meaning that the average size of forecast errors grew in the test period, indicating the model's performance worsened when predicting unseen data.

MAE (Mean Absolute Error): Training Set: 0.9146 Test Set: 1.2406 Similar to RMSE, the MAE is higher in the test set, meaning that the average magnitude of errors increased with the unseen data, suggesting that the model is somewhat sensitive to unseen data.

MPE and MAPE (Mean Percentage Errors): Training Set MPE: 5.2292%, MAPE: 12.4018% Test Set MPE: 5.9401%, MAPE: 9.0641% Interestingly, while the MPE increased slightly in the test set (higher overestimation), the MAPE (absolute percentage error) actually decreased. This indicates that the relative percentage accuracy improved, even though the error magnitudes increased. The model was able to predict reasonably well in relative terms.

MASE and RMSSE: Training Set MASE: 1.0, RMSSE: 1.0 Test Set MASE: 1.3565, RMSSE: 1.2787 Both MASE and RMSSE are higher in the test set, showing that the model's performance worsened when compared to a simple Naive model. The errors were about 35.65% and 27.87% larger than the Naive benchmark in the test period.

ACF1 (Autocorrelation at Lag 1): Training Set: 0.7676 Test Set: 0.6012 The residual autocorrelation decreased in the test set, suggesting that the remaining temporal structure in the errors was slightly lower. However, both values indicate significant autocorrelation, meaning the model still leaves some structure unmodeled.

Increased Error Magnitude: The RMSE, MAE, and ME all increased in the test set, suggesting that the model is sensitive to changes in the training data and performs worse on unseen data. The error metrics in the test set are higher, implying that the model doesn't generalize as well outside the training data.

MAPE and Relative Error: Interestingly, MAPE decreased in the test set, which suggests that even though the magnitude of the errors increased, the model was able to maintain better relative accuracy in percentage terms.

Autocorrelation: The significant autocorrelation in both the training and test residuals suggests that the model isn't fully capturing all the temporal dependencies, though the autocorrelation did decrease slightly in the test set.

Model Sensitivity: The accuracy measures, particularly RMSE and MAE, increased when moving from the training data to the test data, indicating that the model's performance degraded when applied to unseen data. This shows that the model is somewhat sensitive to the amount of training data used, as performance worsened when forecasting for the future period.

Generalization: While the SNAIVE model did relatively well on the training data, the increase in error magnitudes and high autocorrelation in both training and test residuals suggest that it may not generalize well to unseen data, and a more sophisticated model might be needed for better forecasting.