Assignment 2

Jaewoo Cho

FPP3 3.7 Exercises: 7(a-f)

Participant 18 from Fried et al.'s (2022) data

7.

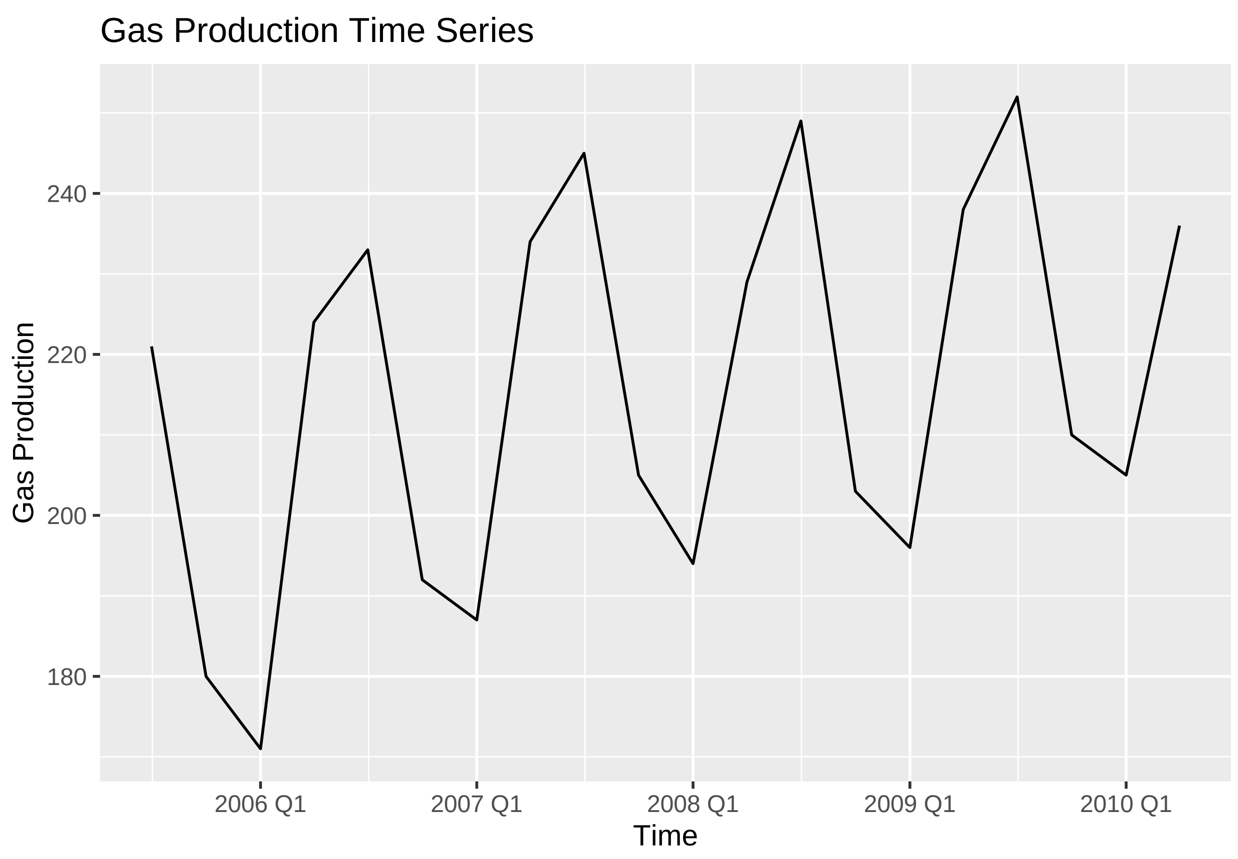
Consider the last five years of the Gas data from `aus_production`.

```r
# Code block found in book
gas <- tail(aus_production, 5*4) %>% select(Gas)
gas
```

```
# A tsibble: 20 x 2 [1Q]
     Gas Quarter
   <dbl>   <qtr>
 1   221 2005 Q3
 2   180 2005 Q4
 3   171 2006 Q1
 4   224 2006 Q2
 5   233 2006 Q3
 6   192 2006 Q4
 7   187 2007 Q1
 8   234 2007 Q2
 9   245 2007 Q3
10   205 2007 Q4
11   194 2008 Q1
12   229 2008 Q2
13   249 2008 Q3
14   203 2008 Q4
15   196 2009 Q1
16   238 2009 Q2
17   252 2009 Q3
18   210 2009 Q4
19   205 2010 Q1
20   236 2010 Q2
```

a. Plot the time series. Can you identify seasonal fluctuations and/or a trend-cycle?

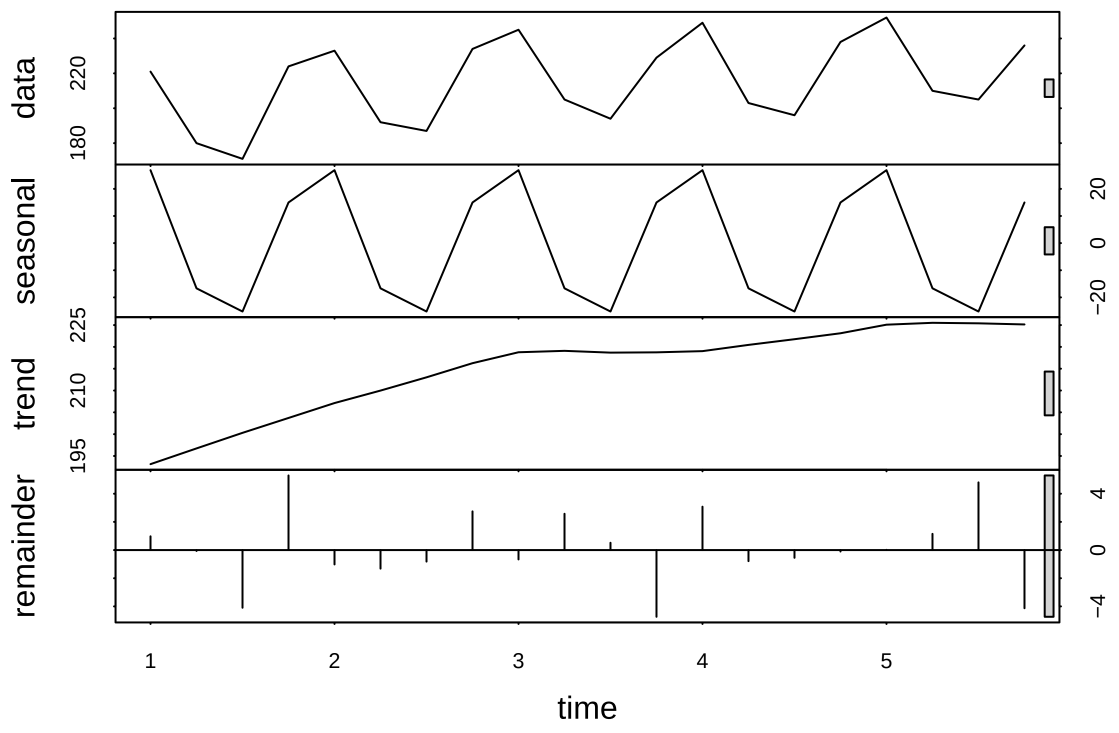
```r
ggplot(gas, aes(x = Quarter, y = Gas)) +
  geom_line() +
  labs(title = "Gas Production Time Series", x = "Time", y = "Gas Production")
```



Report on any patterns relating to seasonality or trend There seems to be a trend/pattern that shows a rapid increase in Gas production from the start of quarter 1 that rapidly decreases in the middle of the quarter for each year. The pattern represents a mountain range like pattern with a constant seasonality trend with a rapid increase following by a rapid decrease.

b. Use `STL` to calculate the trend-cycle and seasonal indices.

```
gas_ts <- ts(gas$Gas, frequency = 4)  # for quarterly data
gas_stl <- stl(gas_ts, s.window = "periodic")
gas_trend <- gas_stl$time.series[, "trend"]
gas_seasonal <- gas_stl$time.series[, "seasonal"]
plot(gas_stl)
```

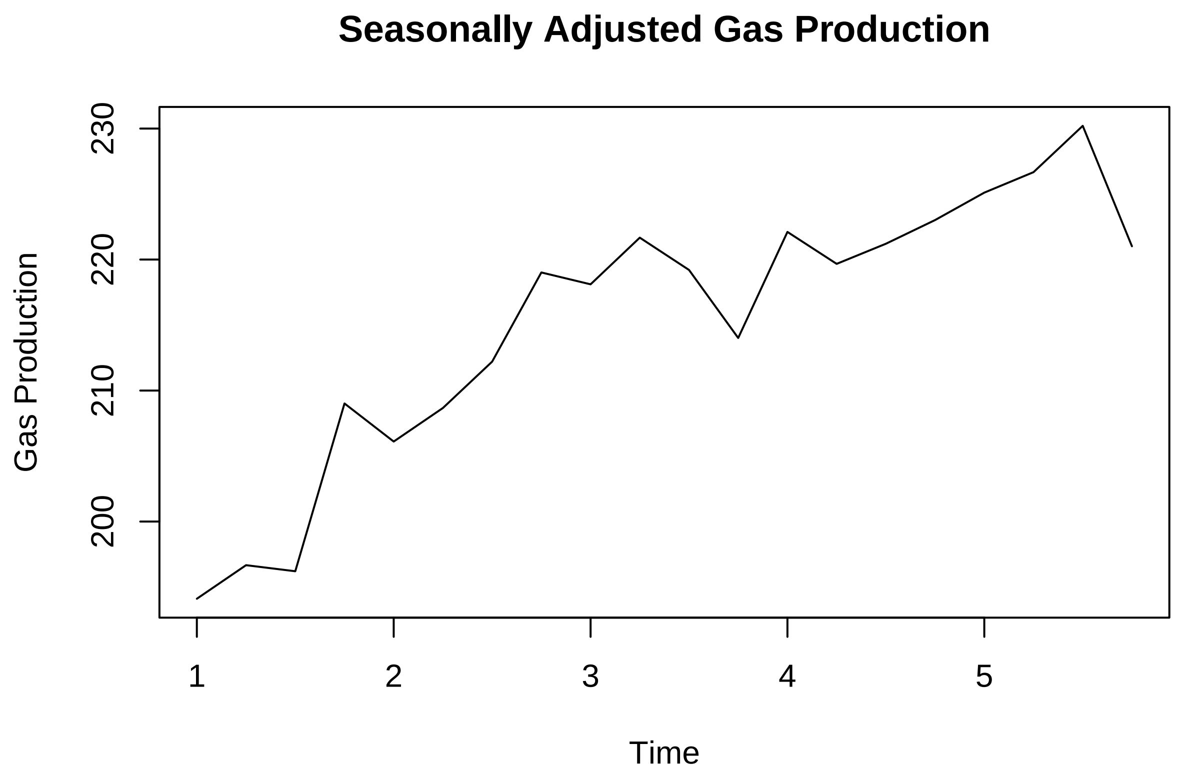


c. Do the results support the graphical interpretation from part a?

Comment on whether the STL decomposition reflects any patterns you suggested in part a The STL decomposition reflects the patterns I suggested in part a as with a constant pattern of rapid increase with a following of a rapid decrease like a mountain range pattern trend.

d. Compute and plot the seasonally adjusted data.

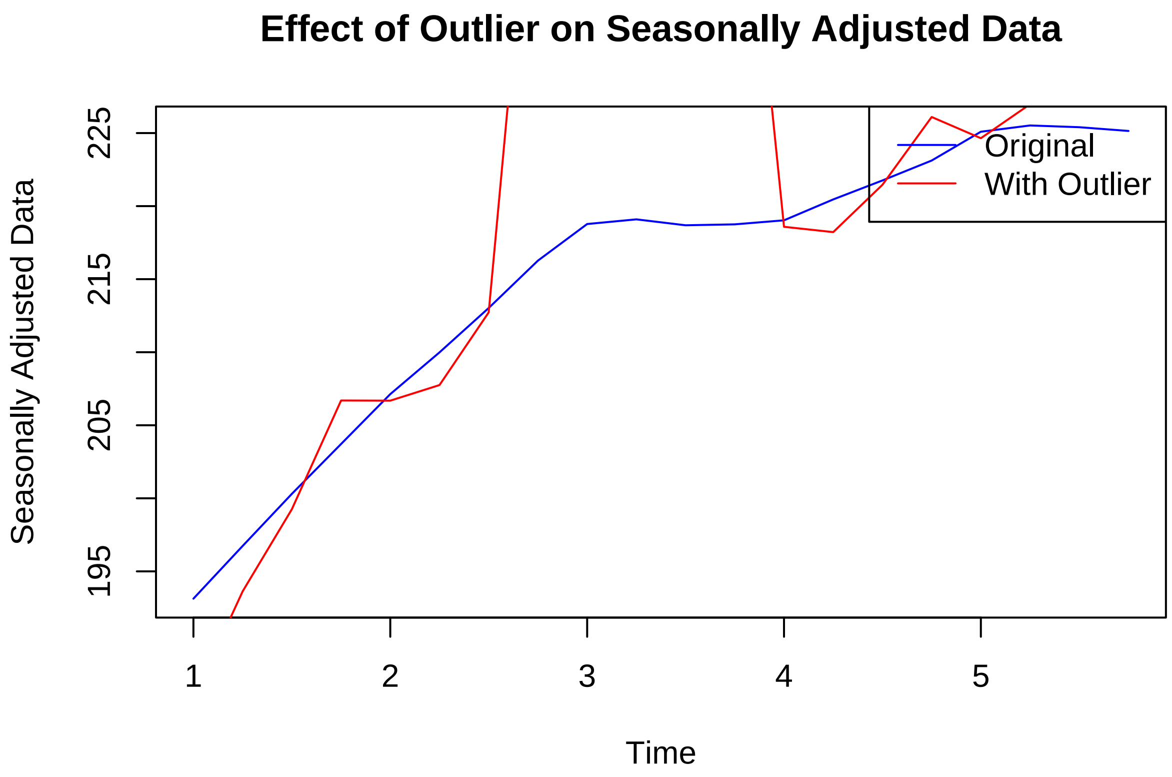
```r
# Hint: use `season_adjust` in STL plot
# Compute the seasonally adjusted data
seasonally_adjusted <- gas_ts - gas_seasonal

# Plot the seasonally adjusted data
plot(seasonally_adjusted, main = "Seasonally Adjusted Gas Production", xlab = "Time", ylab = "Gas Produ
```



e. Change one observation to be an outlier (e.g., add 300 to one observation), and recompute the seasonally adjusted data. What is the effect of the outlier?

The effect of the outlier highly changes the predictions at a certain point.

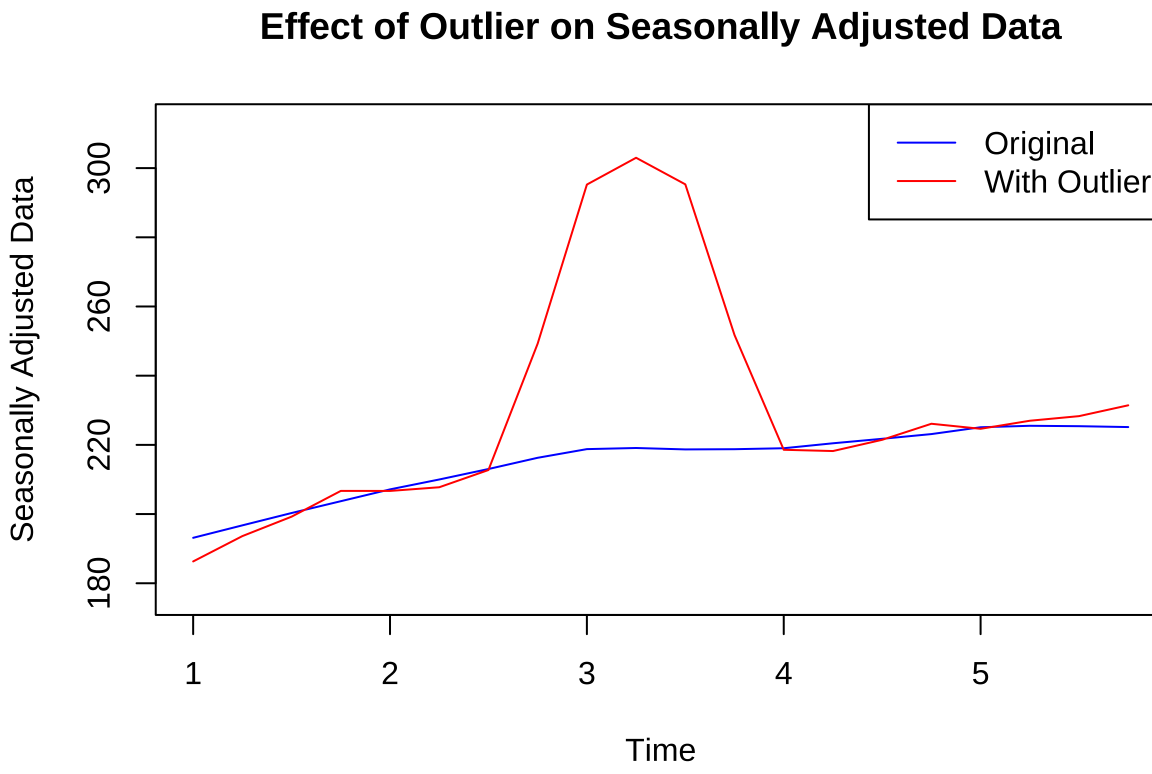
```r
# Assuming the 'gas' dataset is a dataframe with a 'Gas' column and a 'Quarter' column
gas_outlier <- gas %>%
  mutate(Gas = if_else(Quarter == yearquarter("2007Q4"), Gas + 300, Gas))
# Convert to time series
gas_ts <- ts(gas$Gas, frequency = 4)

# Decompose the original data
stl_original <- stl(gas_ts, s.window="periodic")

# Convert the outlier dataset to time series
gas_outlier_ts <- ts(gas_outlier$Gas, frequency = 4)

# Decompose the data with the outlier
stl_outlier <- stl(gas_outlier_ts, s.window="periodic")

# Plot the original seasonally adjusted data
plot(stl_original$time.series[, 2], type="l", col="blue", ylab="Seasonally Adjusted Data", xlab="Time",
lines(stl_outlier$time.series[, 2], col="red")

legend("topright", legend=c("Original", "With Outlier"), col=c("blue", "red"), lty=1)
```



Normal version

```r
# Add outlier (done for you)
gas_outlier <- gas %>%
  mutate(Gas = if_else(Quarter == yearquarter("2007Q4"), Gas + 300, Gas))

# Convert to time series
gas_ts <- ts(gas$Gas, frequency = 4)

# Decompose the original data
stl_original <- stl(gas_ts, s.window="periodic")

# Convert the outlier dataset to time series
gas_outlier_ts <- ts(gas_outlier$Gas, frequency = 4)

# Decompose the data with the outlier
stl_outlier <- stl(gas_outlier_ts, s.window="periodic")

# Determine the y-axis limits
y_min <- min(c(stl_original$time.series[, 2], stl_outlier$time.series[, 2]))
```

```
y_max <- max(c(stl_original$time.series[, 2], stl_outlier$time.series[, 2]))

# Add some padding to the y-axis limits
y_min <- y_min - 10
y_max <- y_max + 10

# Plot the original seasonally adjusted data with adjusted y-axis limits
plot(stl_original$time.series[, 2], type="l", col="blue", ylab="Seasonally Adjusted Data", xlab="Time",
lines(stl_outlier$time.series[, 2], col="red")

legend("topright", legend=c("Original", "With Outlier"), col=c("blue", "red"), lty=1)
```



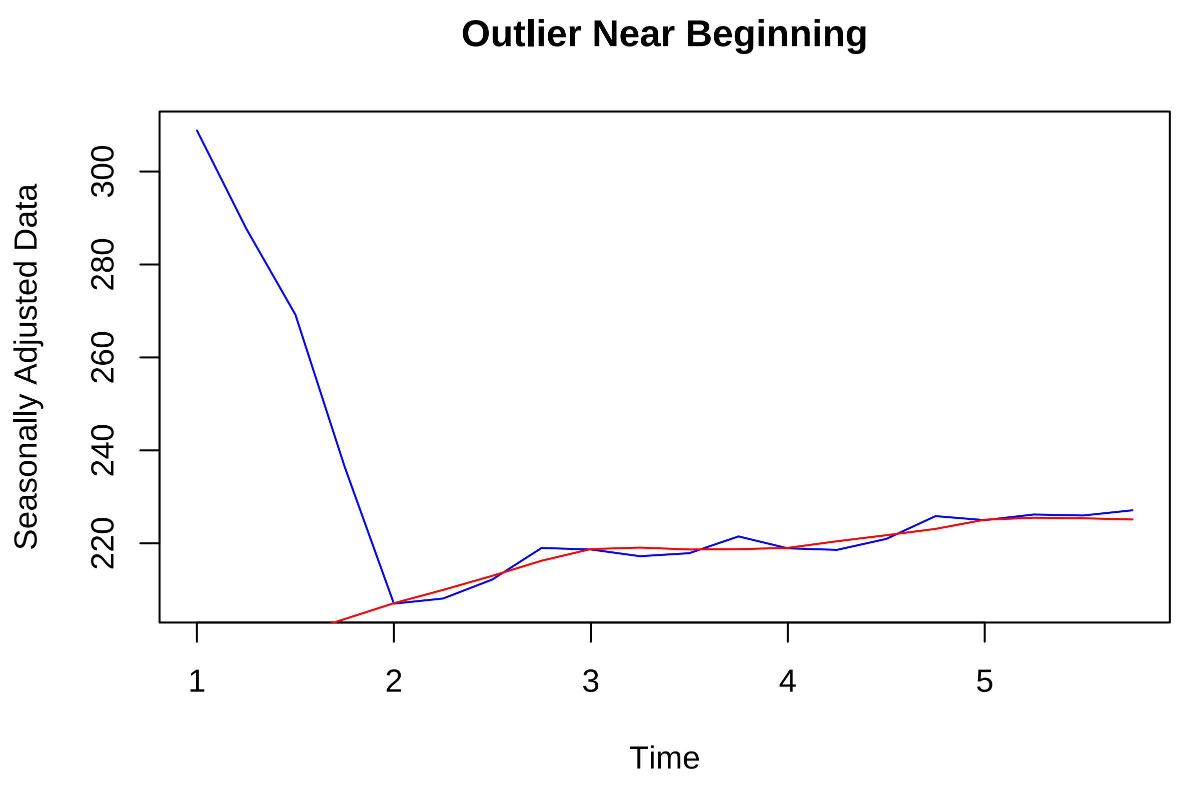
y-axis range version

> Comment on the effect relative to the original seasonally adjusted plot The effect relative to the original seasonally adjusted plot makes the predictions way off for the original prediction. As you can see with the outlier included, the graph has a large increase that makes the predictions fall off.

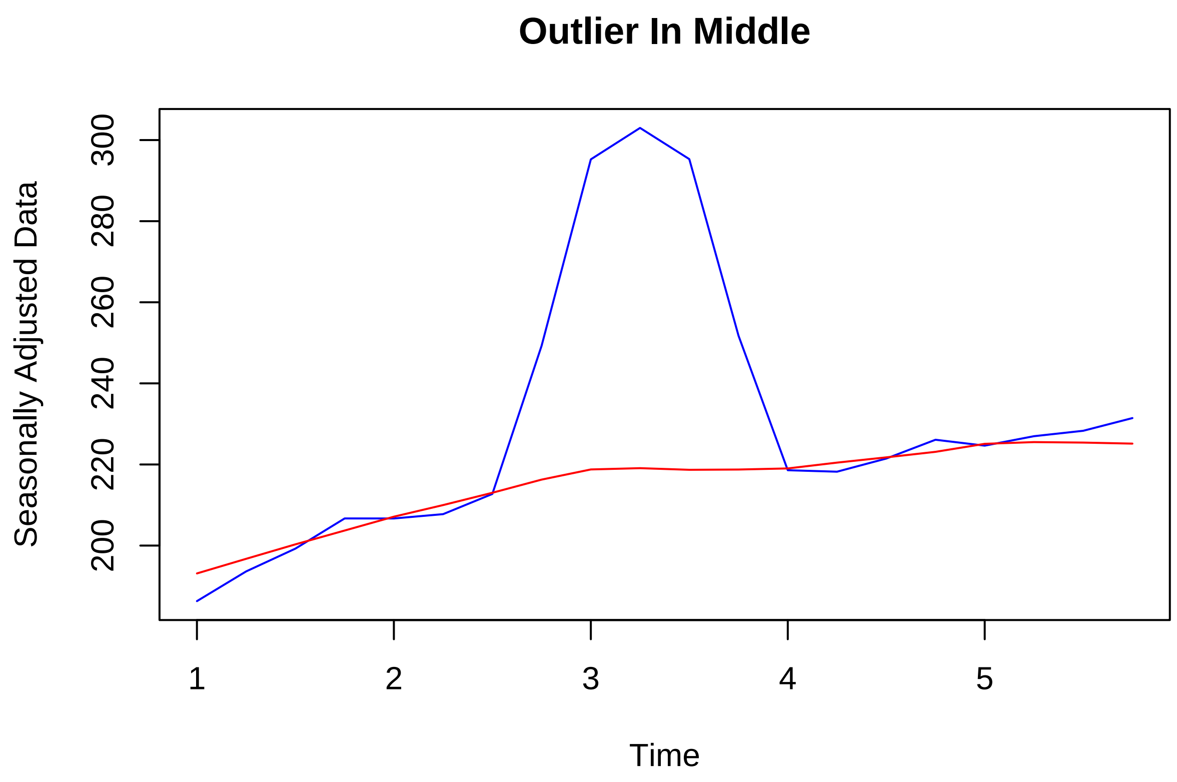
f. Does it make any difference if the outlier is near the end rather than in the middle of the time series?

```
# Hint: Use the code provided above but
# adjusted the year and quarter
# Try a few different years and quarters
```

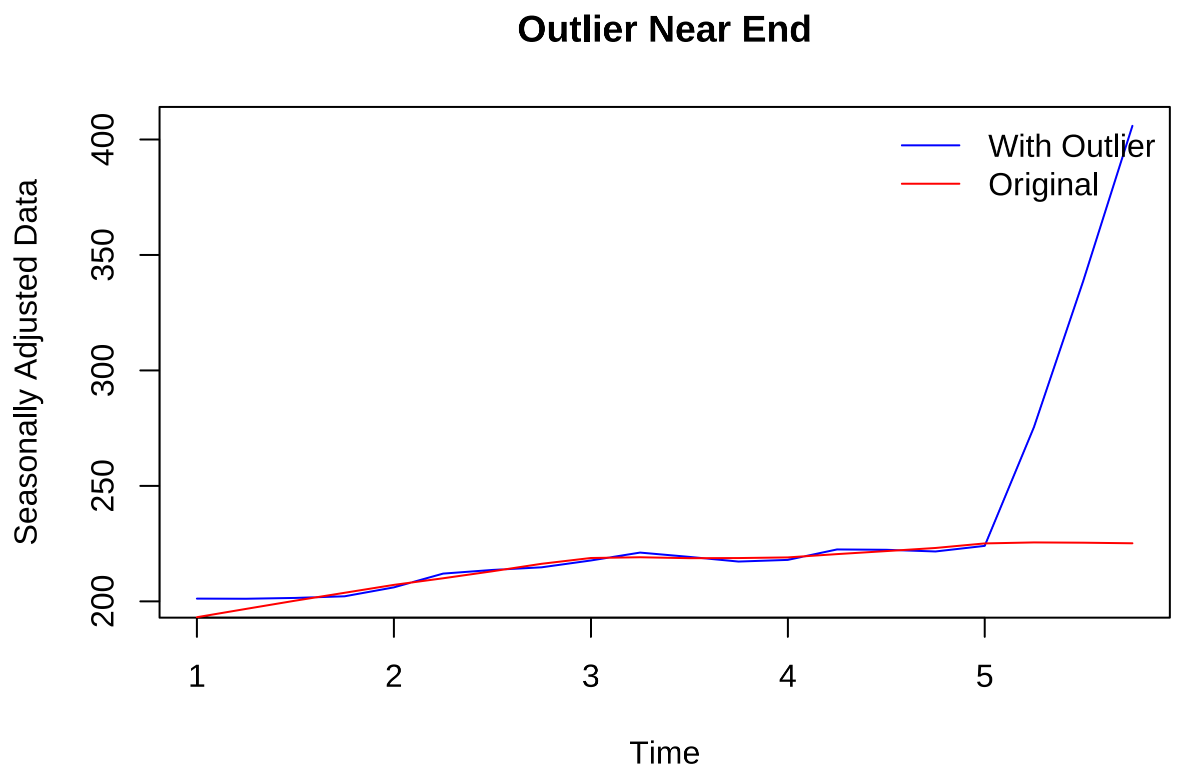
```r
# to get a better understanding
# (plot at least three different combinations)

# Introduce an outlier near the beginning of the series
gas_outlier_beginning <- gas %>%
  mutate(Gas = if_else(Quarter == yearquarter("2005Q4"), Gas + 300, Gas))

# Introduce an outlier in the middle of the series (as done previously)
gas_outlier_middle <- gas %>%
  mutate(Gas = if_else(Quarter == yearquarter("2007Q4"), Gas + 300, Gas))

# Introduce an outlier near the end of the series
last_quarter <- tail(gas$Quarter, n=1)
gas_outlier_end <- gas %>%
  mutate(Gas = if_else(Quarter == last_quarter, Gas + 300, Gas))

# Decompose each dataset
stl_beginning <- stl(ts(gas_outlier_beginning$Gas, frequency = 4), s.window="periodic")
stl_middle <- stl(ts(gas_outlier_middle$Gas, frequency = 4), s.window="periodic")
stl_end <- stl(ts(gas_outlier_end$Gas, frequency = 4), s.window="periodic")

# Beginning
plot(stl_beginning$time.series[, 2], type="l", col="blue", ylab="Seasonally Adjusted Data", xlab="Time"
lines(stl_original$time.series[, 2], col="red")
```



```
# Middle
plot(stl_middle$time.series[, 2], type="l", col="blue", ylab="Seasonally Adjusted Data", xlab="Time", ma
lines(stl_original$time.series[, 2], col="red")
```



```
# End
plot(stl_end$time.series[, 2], type="l", col="blue", ylab="Seasonally Adjusted Data", xlab="Time", main=
lines(stl_original$time.series[, 2], col="red")

legend("topright", legend=c("With Outlier", "Original"), col=c("blue", "red"), lty=1, bty="n")
```



Comment on general patterns you see (e.g., what happens to the trend? what happens to seasonlity?) The general patterns I see makes a huge increase in the Gas Productions predictions with a high fluctuation in the outlier reasons. The trend and seasonality shows a high fluctuation depending on the year of the outlier; early of the year for the beginning outlier, mid of the year for the mid outlier, and end of the year for the end outlier. However, except for the outlier production year, the overall seasonality and trend seems like it follows the original production.

Participant 18

Look to slides for help

Prepare Data

```
# Load data
emotions <- read_csv("/Users/jaewoocho/Desktop/DS Fall 2023/DS adv stats/02_Forecasting_Decomposition/cl
```

```
## Rows: 4372 Columns: 26-- Column specification -----------------------------------------------------------
## Delimiter: ","
## chr   (4): ID, Issued, Response, Duration
## dbl  (19): Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13, Q14, Q15,...
## dttm  (2): Scheduled, time
## date  (1): Day
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
emotions
```

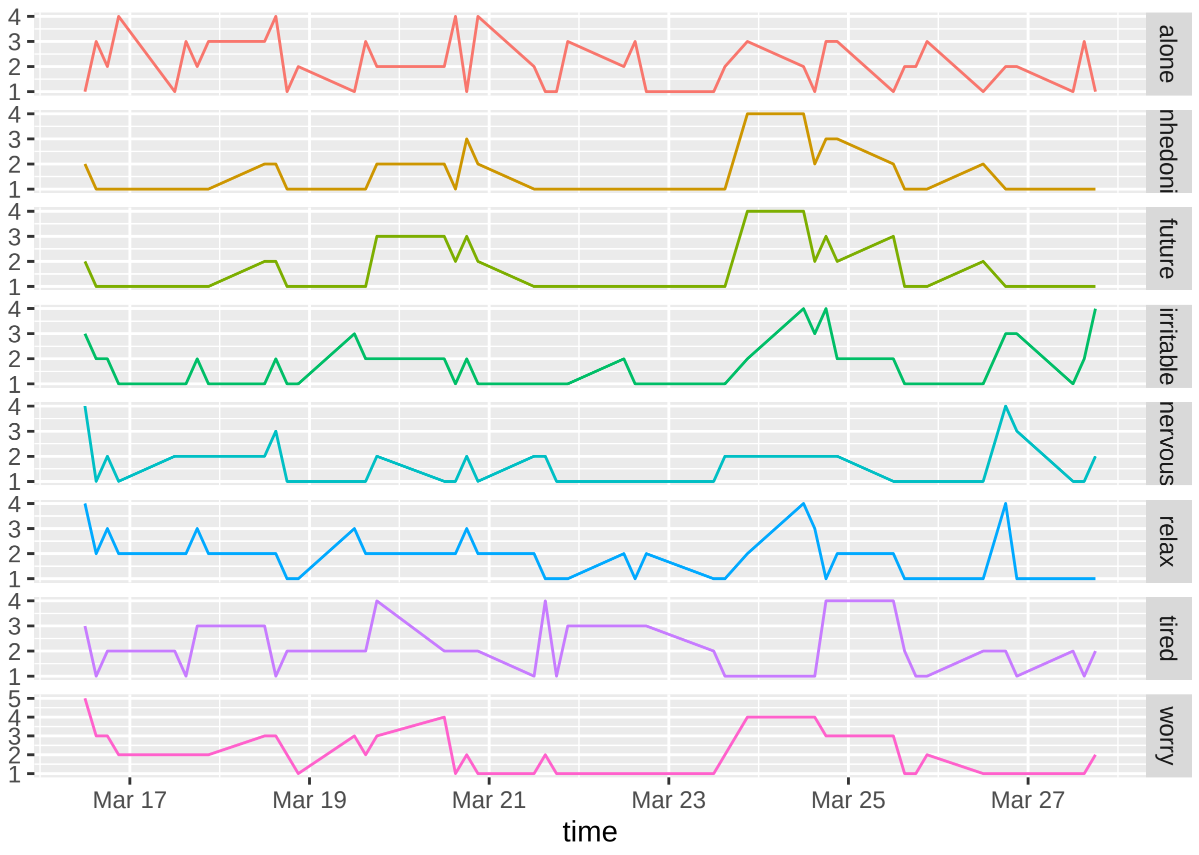
```
## # A tibble: 4,372 x 26
##    ID         Scheduled           Issued Respo~1 Durat~2    Q1    Q2    Q3    Q4
##    <chr>      <dttm>              <chr>  <chr>   <chr>   <dbl> <dbl> <dbl> <dbl>
##  1 User #252~ 2020-03-16 12:00:00 2020-~ 2020-0~ 285.734     1     1     2     1
##  2 User #252~ 2020-03-16 15:00:00 2020-~ 2020-0~ 1822.7~     2     1     2     1
##  3 User #252~ 2020-03-16 18:00:00 2020-~ 2020-0~ 238.774     1     1     3     1
##  4 User #252~ 2020-03-16 21:00:00 2020-~ 2020-0~ 967.132     1     1     3     2
##  5 User #252~ 2020-03-17 12:00:00 2020-~ 2020-0~ 1995.54     1     1     2     1
##  6 User #252~ 2020-03-17 15:00:00 2020-~ 2020-0~ 721.237     3     2     2     1
##  7 User #252~ 2020-03-17 18:00:00 2020-~ 2020-0~ Expired    NA    NA    NA    NA
##  8 User #252~ 2020-03-17 21:00:00 2020-~ 2020-0~ 312.475     1     2     1     2
##  9 User #252~ 2020-03-18 12:00:00 2020-~ 2020-0~ 2753.7~     1     1     1     1
## 10 User #252~ 2020-03-18 15:00:00 2020-~ 2020-0~ Expired    NA    NA    NA    NA
## # ... with 4,362 more rows, 17 more variables: Q5 <dbl>, Q6 <dbl>, Q7 <dbl>,
## #   Q8 <dbl>, Q9 <dbl>, Q10 <dbl>, Q11 <dbl>, Q12 <dbl>, Q13 <dbl>, Q14 <dbl>,
## #   Q15 <dbl>, Q16 <dbl>, Q17 <dbl>, Q18 <dbl>, time <dttm>, Day <date>,
## #   beepvar <dbl>, and abbreviated variable names 1: Response, 2: Duration
```

```r
# Obtain participant 18
participant <- emotions[emotions$ID == unique(emotions$ID)[18],]
# Obtain time and question variables
questions <- data.frame(time = participant$time, # time
                        participant[,grep("Q", colnames(participant) )])# questions


# First eight questions
data <- questions[,c(
  1, # time
  2:9 # first eight questions
)]

# Relabel questions
colnames(data)[2:9] <- c(
  "relax", "irritable", "worry",
  "nervous", "future", "anhedonia",
  "tired", "alone"
)

# Remove missing data
data <- na.omit(data)

# Convert to `tsibble`
ts <- data %>%
  mutate(
    time = ymd_hms(time)
  ) %>%
  as_tsibble(
    index = time
  )

# Convert to `tsibble`
```

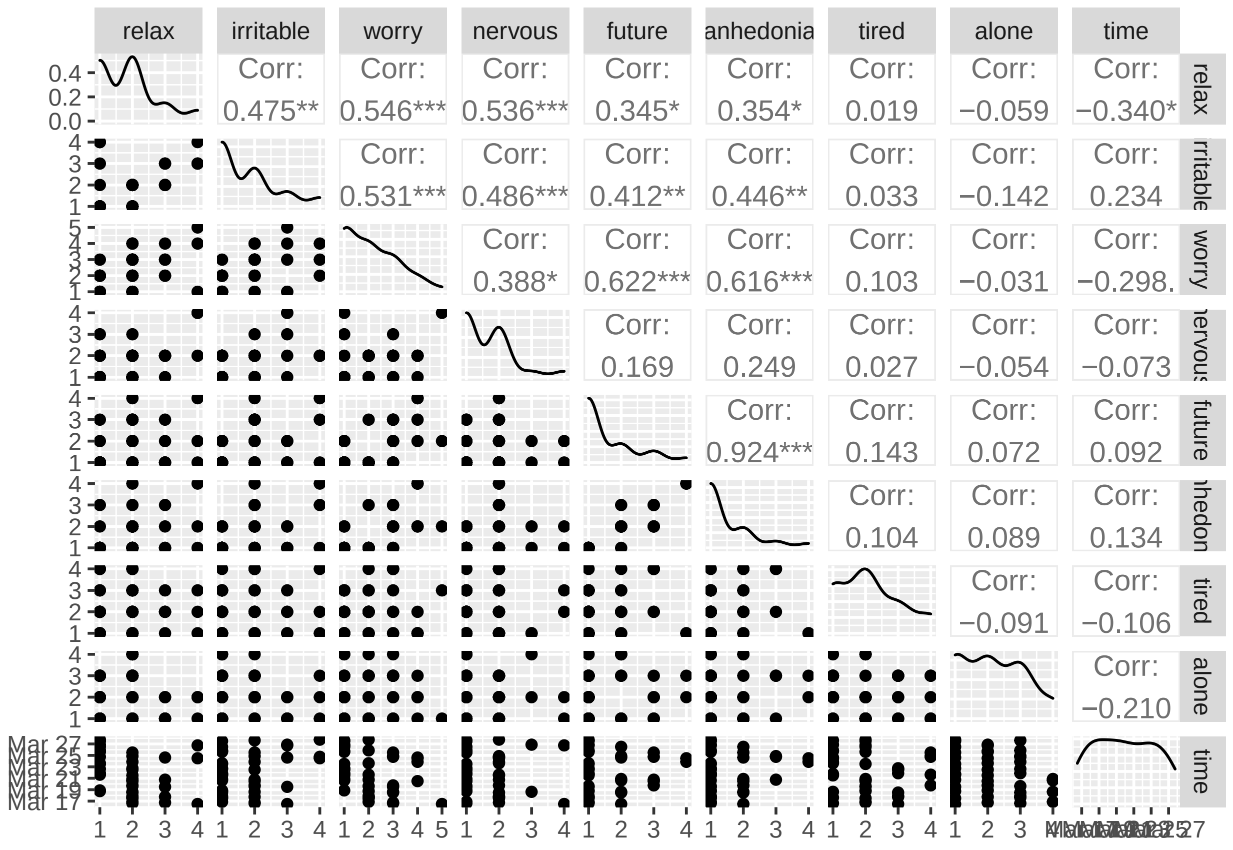
```r
ts_fill <- ts %>%
  fill_gaps() # fill in time gaps
# for plotting residuals later

# Length of time series
ts_length <- nrow(ts)
ts_fill_length <- nrow(ts_fill)

# Remove last four time points (we'll make a prediction later)
prediction <- ts[
  -c((ts_length - 7):ts_length), # remove last 4 points
]

# For modeling residuals
prediction_fill <- ts_fill[
  -c((ts_fill_length - 7):ts_fill_length), # remove last 4 points
]

# Save last four time points (we'll compare with prediction)
actual <- ts[
  c((ts_length - 7):ts_length), # keeps last 4 points
] %>%
  fill_gaps()
```

Visualize Data

```r
# Visualize time series
prediction %>%
  gather("Measure", "Change", relax, irritable, worry,nervous, future, anhedonia,tired, alone) %>%
  ggplot(aes(x = time, y = Change, colour = Measure)) + geom_line() +
  facet_grid(vars(Measure), scales = "free_y") + labs(y="") +
  guides(colour="none")
```



```
# Compute correlations
prediction %>% select(-time) %>% GGally::ggpairs()
```



Model Estimation

```
# Fit linear model
fit <- prediction_fill %>% # our data
model( # model for time series
tslm = TSLM( # time series linear model
worry ~ relax + irritable + nervous + future + anhedonia + tired + alone
) )
```

```
# Report fit
report(fit)
```

```
Series: worry
Model: TSLM

Residuals:
      Min         1Q    Median        3Q       Max
-1.725221 -0.359439  0.003946  0.543378  1.555042

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.14889    0.52741   0.282    0.779
relax        0.26925    0.16716   1.611    0.115
```

```
irritable    0.15649    0.17011   0.920    0.363
nervous      0.10090    0.18335   0.550    0.585
future       0.45157    0.38226   1.181    0.245
anhedonia    0.18207    0.40809   0.446    0.658
tired        0.04121    0.12728   0.324    0.748
alone       -0.04489    0.12582  -0.357    0.723

Residual standard error: 0.8081 on 39 degrees of freedom
Multiple R-squared: 0.5193, Adjusted R-squared: 0.433
F-statistic: 6.018 on 7 and 39 DF, p-value: 8.5352e-05
```

Report on predictors (were any significant?) Based on the report fit, there were no signficant predictors with the significance level of 0.05

Forecast: Make Forecast

You will need to generate new data

Use ChatGPT to brain storm ways to generate new data for TSLM

Your goals:

1. Work with ChatGPT to come up with a way to generate new data for TSLM
2. Make a forecast with the new data with your TSLM model
3. Compare your forecast against random data (use `sample`)
4. Plot your forecast and the random forecast
5. Evaluate the predictions and determine which method produced better forecasts

Forecast: Method with ChatGPT

```r
# Load necessary libraries
library(forecast)

# Create a synthetic dataset
set.seed(123) # For reproducibility
time_points <- 100
trend <- seq(1, time_points, by=1)
seasonality <- 10 * sin(2 * pi * (1:time_points) / 12) # Monthly seasonality
noise <- rnorm(time_points, mean=0, sd=5)
ts_data <- ts(trend + seasonality + noise, frequency=12)
# Split the data into training and test sets
train_size <- floor(0.8 * time_points)
train_data <- window(ts_data, start=c(1,1), end=c(train_size/12, train_size%%12))
test_data <- window(ts_data, start=c((train_size+1)/12, (train_size+1)%%12))
```

Forecast: Method with ChatGPT

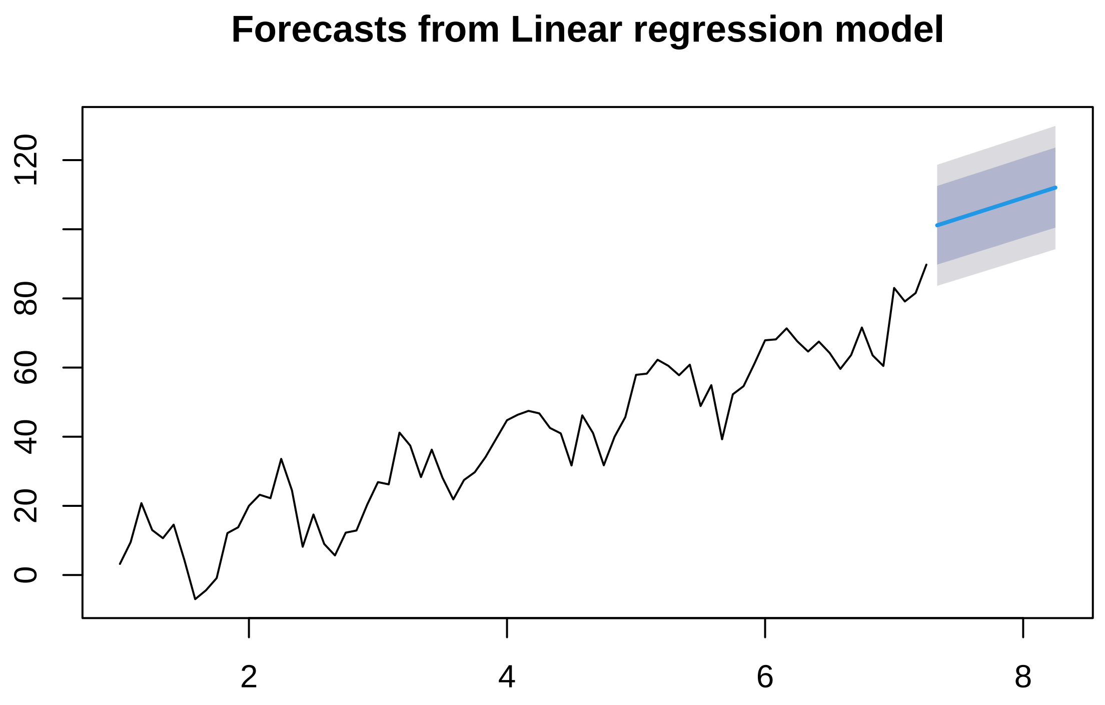
```r
# Fit a time series linear model
fit <- tslm(train_data ~ trend)

# Make future possibilities
future_periods <- 12
future_scenarios <- data.frame(trend = seq(time_points + 1, time_points + future_periods, by=1))

# Make forecast
fc <- forecast(fit, newdata = future_scenarios)
```

Forecast: Method with Random Data

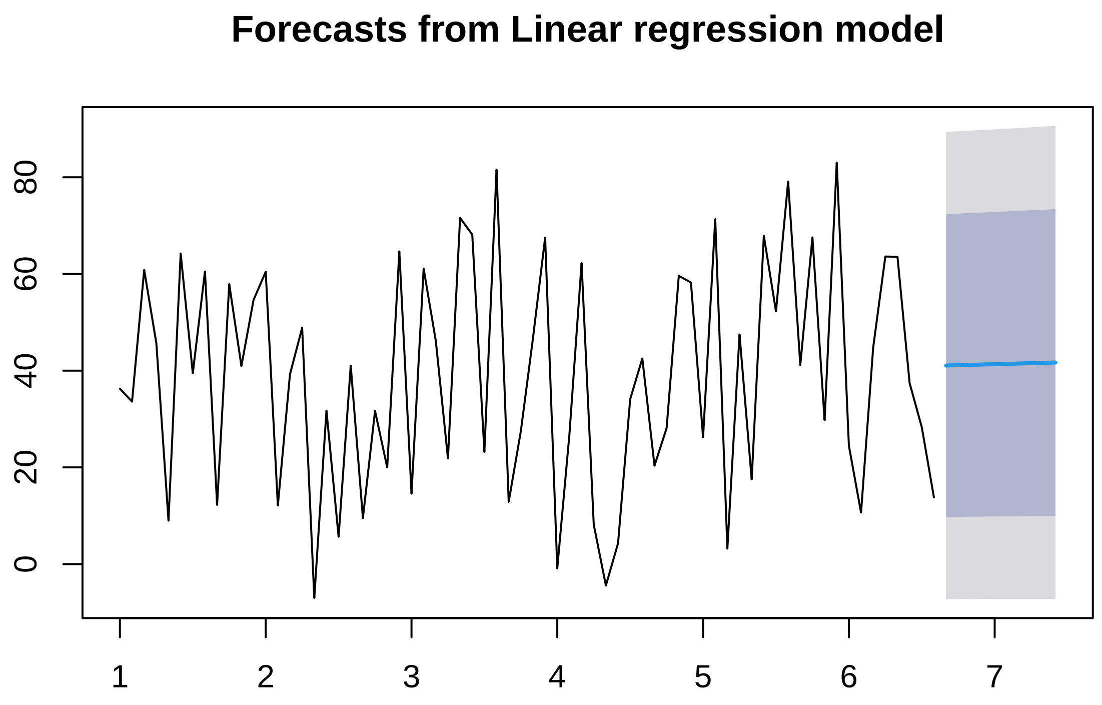
```r
sample_size <- floor(0.9 * length(train_data))
sampled_data <- sample(train_data, size = sample_size, replace = FALSE)

# Convert the sampled data to a time series object
sampled_ts_data <- ts(sampled_data, frequency=12)

# Fit a time series linear model to the sampled data
fit <- tslm(sampled_ts_data ~ trend)

# Make forecast
fc_random <- fit %>% forecast(new_data = random_scenarios)
```

Forecast: Plot Forecast

```r
# Plot forecasts simultaneously
# Plot the forecast
plot(fc)
```



```
plot(fc_random)
```



Comment on each of the predictions. Do any seem to work well? Which do you think performed best and why? The predictions seem to work well with more datapoints from the original dataset. I think the original forecast worked better as the model had more datapoints for predictions.

Forecast: Evaluate Forecast

```
# Obtain metrics
# Calculate residuals for both forecasts
residuals_original <- test_data - fc$mean
residuals_random <- test_data - fc_random$mean

# Calculate MSE
mse_original <- mean(residuals_original^2)
mse_random <- mean(residuals_random^2)

# Calculate RMSE
rmse_original <- sqrt(mse_original)
rmse_random <- sqrt(mse_random)

# Calculate MAPE
mape_original <- mean(abs(residuals_original / test_data)) * 100
mape_random <- mean(abs(residuals_random / test_data)) * 100

# Print metrics
```

```
cat("Original Model Metrics:\n")
cat("MSE:", mse_original, "\n")
cat("RMSE:", rmse_original, "\n")
cat("MAPE:", mape_original, "%\n\n")

cat("Random Model Metrics:\n")
cat("MSE:", mse_random, "\n")
cat("RMSE:", rmse_random, "\n")
cat("MAPE:", mape_random, "%\n")
```

Which method performed best based on these evaluation measures? Justify which method you think performed best Based on these evaluation measures, the Random Model performed better than the Original Model in terms of prediction accuracy. The Random Model has lower error metrics across the board, indicating that it is more accurate and reliable in its predictions compared to the Original Model.

- MSE: The Random Model has a lower MSE (396.8905) compared to the Original Model (653.0192). A lower MSE indicates that the Random Model has fewer errors.
- RMSE: The Random Model has a lower RMSE (19.92211) compared to the Original Model (25.55424). A lower RMSE indicates that the Random Model's predictions are closer to the actual values.
- MAPE: The Random Model has a lower MAPE (27.70946%) compared to the Original Model (30.8547%). A lower MAPE indicates that the Random Model's predictions are more accurate in terms of percentage error.