Assignment 2

Rachel Montgomery

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`.

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

head(gas) ##looking at data
```

```
# A tsibble: 6 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
```

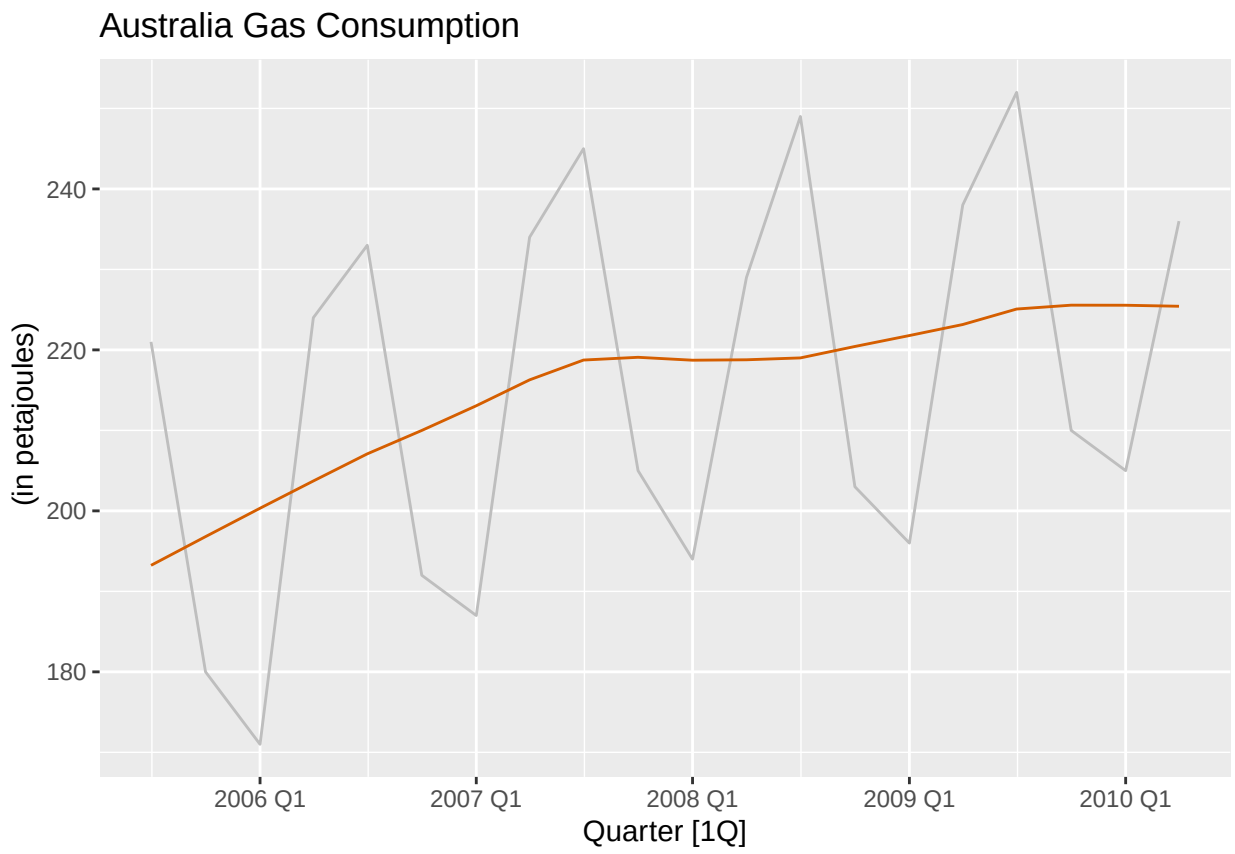
a. Plot the time series. Can you identify seasonal fluctuations and/or a trend-cycle?

```
autoplot(gas)
```

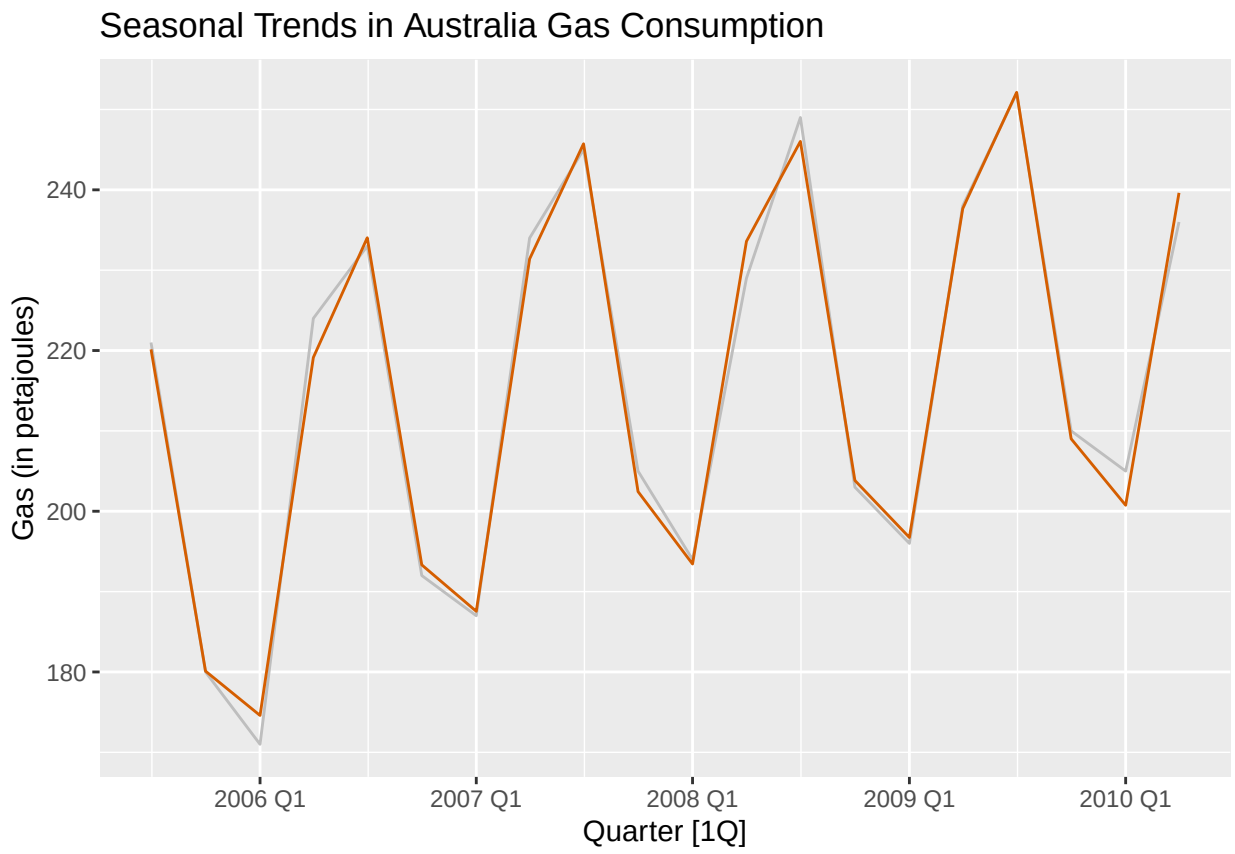
There is an upward trend with a quarterly seasonality.

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

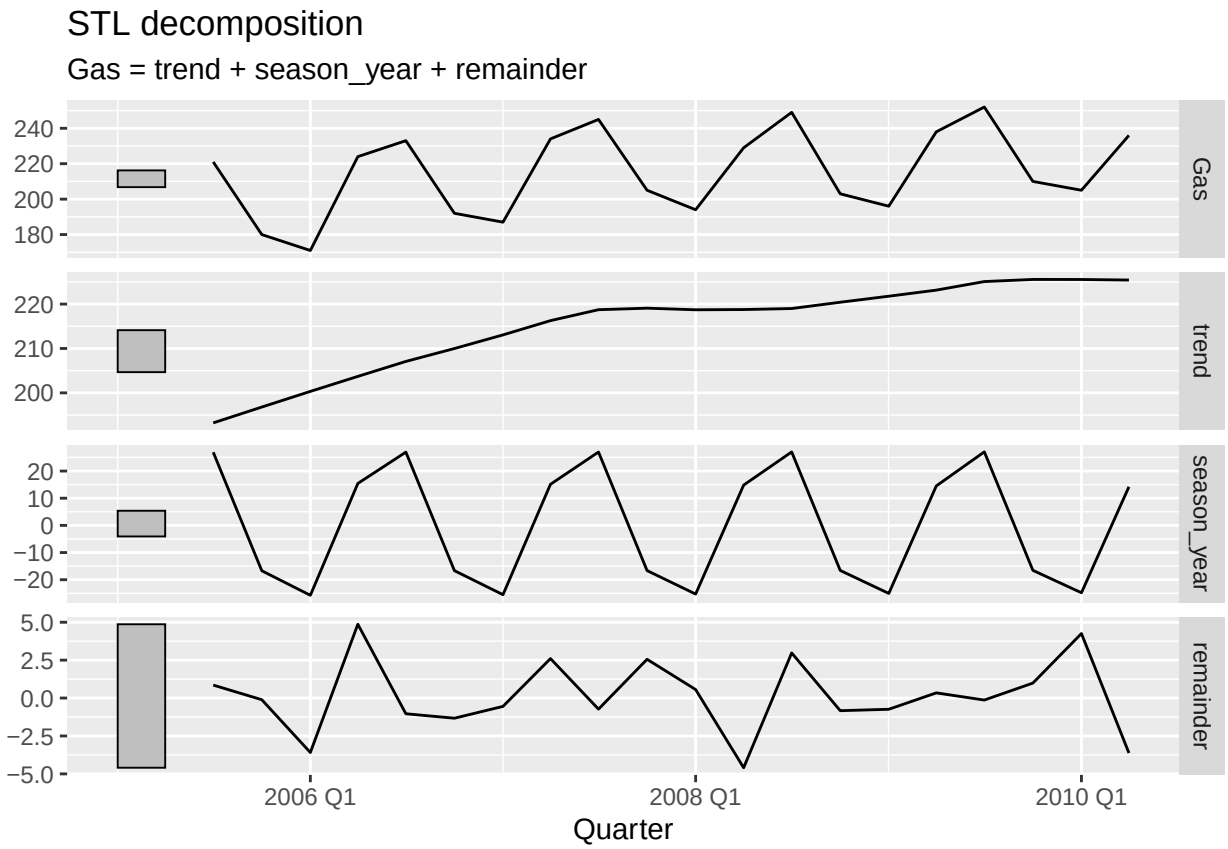
```
# Store components
gas_components <- gas %>%
  model(stl = STL(Gas))

# STL Trend
gas %>%
  autoplot(Gas, color = 'gray') +
  autolayer(
    components(gas_components),
    trend, # plot trend
    color = '#D55E00'
  ) +
  labs(
    y = " (in petajoules)",
    title = "Australia Gas Consumption"
  )
```




```
gas %>%
  autoplot(Gas, color = 'gray') +
  autolayer(
    components(gas_components),
    trend + season_year, # plot trend and seasonality
    color = '#D55E00'
  ) +
  labs(
    y = "Gas (in petajoules)",
    title = "Seasonal Trends in Australia Gas Consumption"
  )
```




```
# STL decomposition
gas %>% # dataset
  model(stl = STL(Gas)) %>% # model (STL)
  components() %>% # components of decomposition
  autoplot() # plot
```

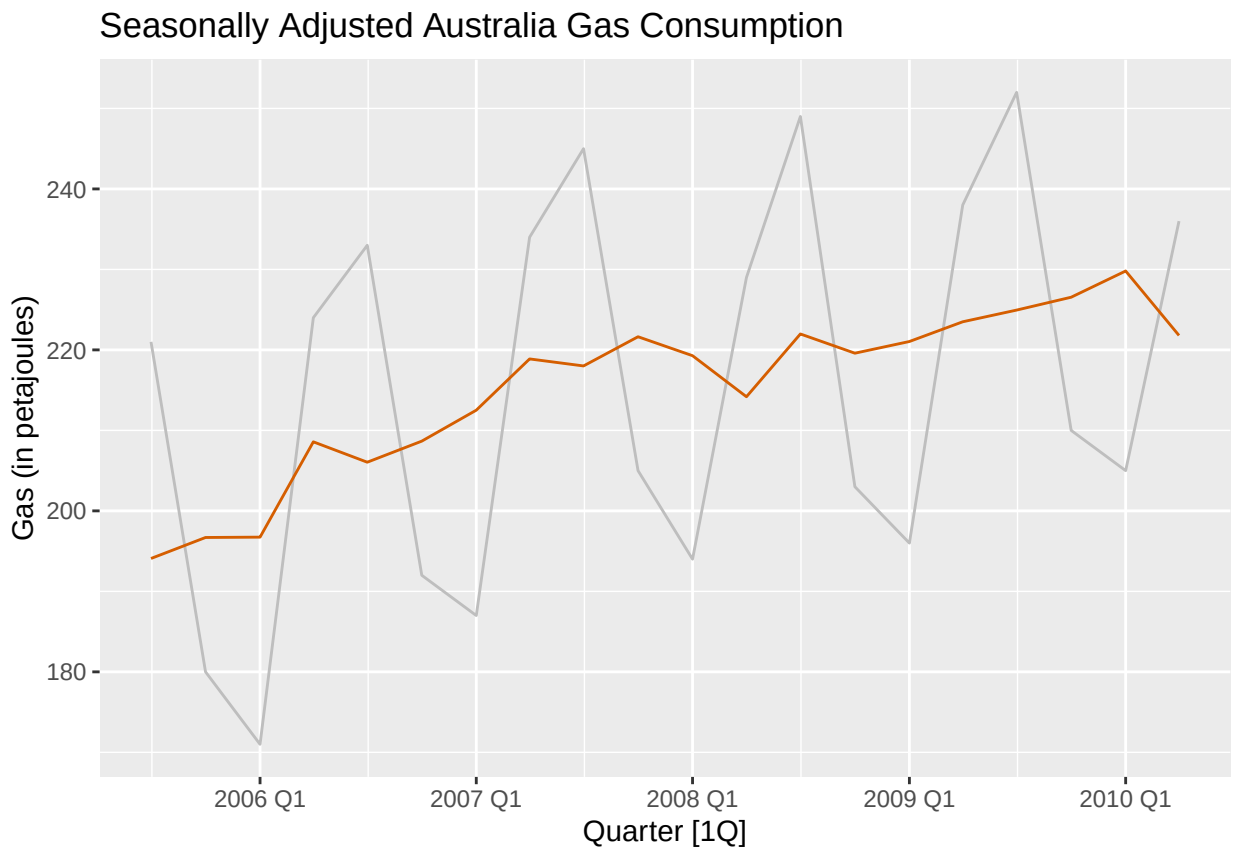


c. Do the results support the graphical interpretation from part a?

Yes, after the STL decomposition, we can se the upward trend and the quarterly in the decompositon components.

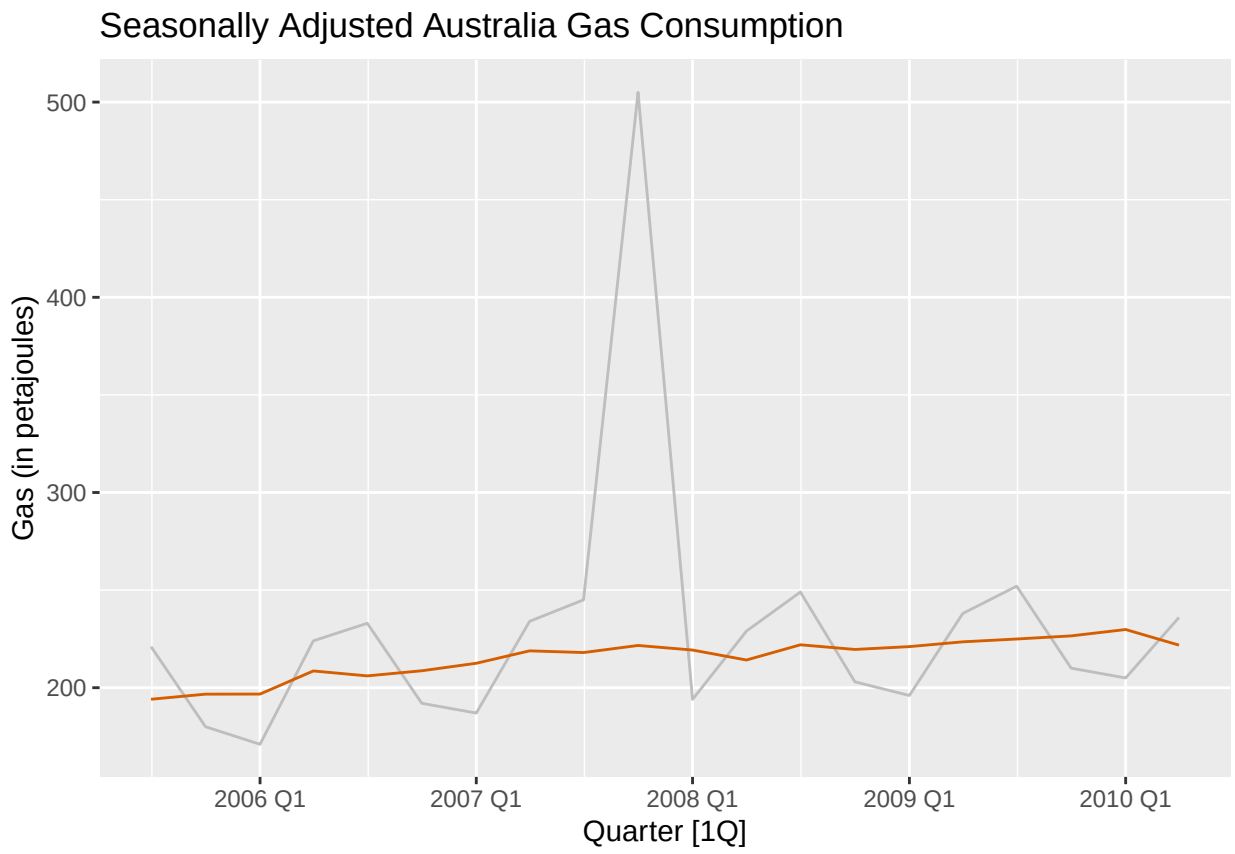
d. Compute and plot the seasonally adjusted data.

```
# Hint: use `season_adjust` in STL plot

# STL Season Adjustment
gas %>%
  autoplot(Gas, color = 'gray') +
  autolayer(
    components(gas_components),
    season_adjust, # plot season adjustment
    color = '#D55E00'
  ) +
  labs(
    y = "Gas (in petajoules)",
    title = "Seasonally Adjusted Australia Gas Consumption"
  )
```



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?

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

# Replot seasonally adjusted data using STL
gas_outlier %>%
  autoplot(Gas, color = 'gray') +
  autolayer(
    components(gas_components),
    season_adjust, # plot season adjustment
    color = '#D55E00'
  ) +
  labs(
    y = "Gas (in petajoules)",
    title = "Seasonally Adjusted Australia Gas Consumption"
  )
```



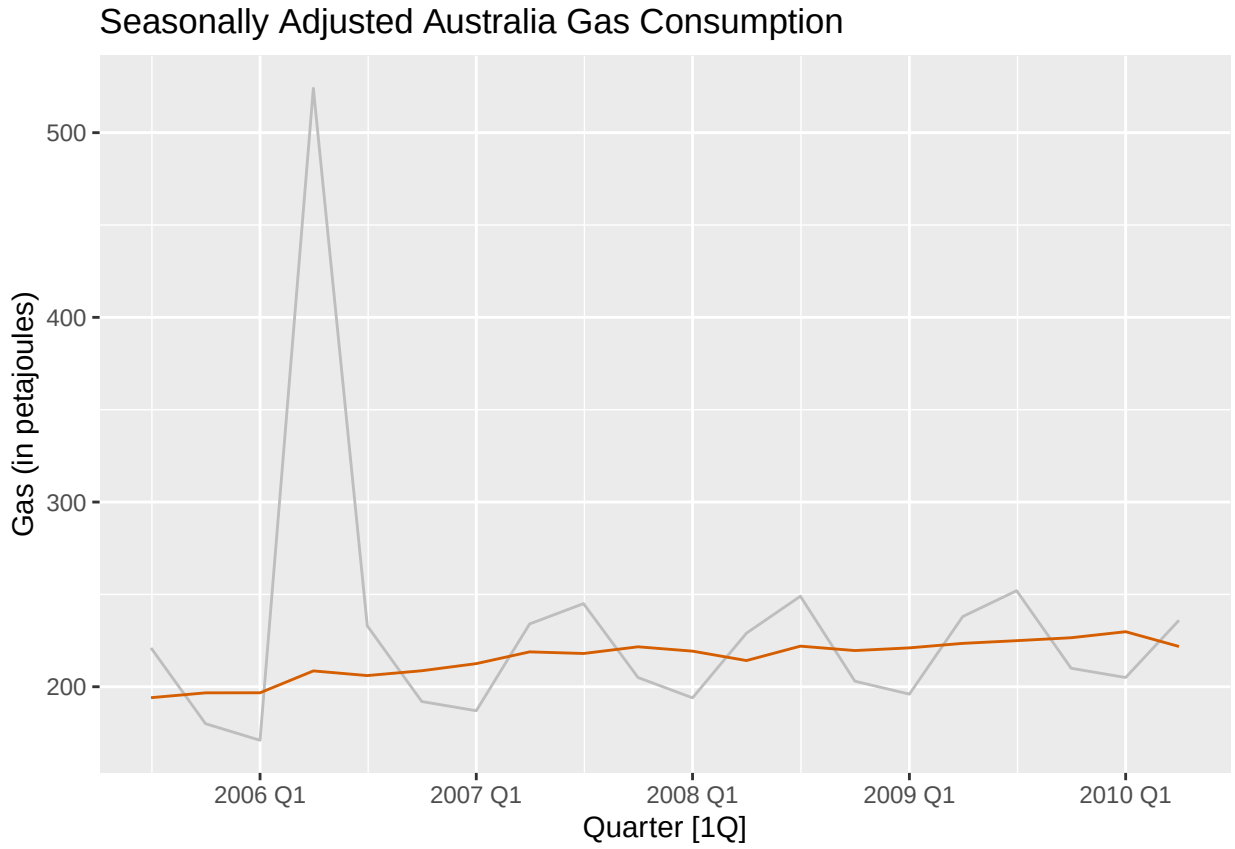
Comment on the effect relative to the original seasonally adjusted plot

Adding the outlier of 300 created a laege spike in the data from 2007 to 2008. It also lowered the seasonal adjustment to 200, and made the trend decrease drastically, so the adjustment almost appears constant. The effect is similar to that of a pool - once a large dent was created, the rest of the plot had to "balance itself out" by lowering the values around the spike.

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

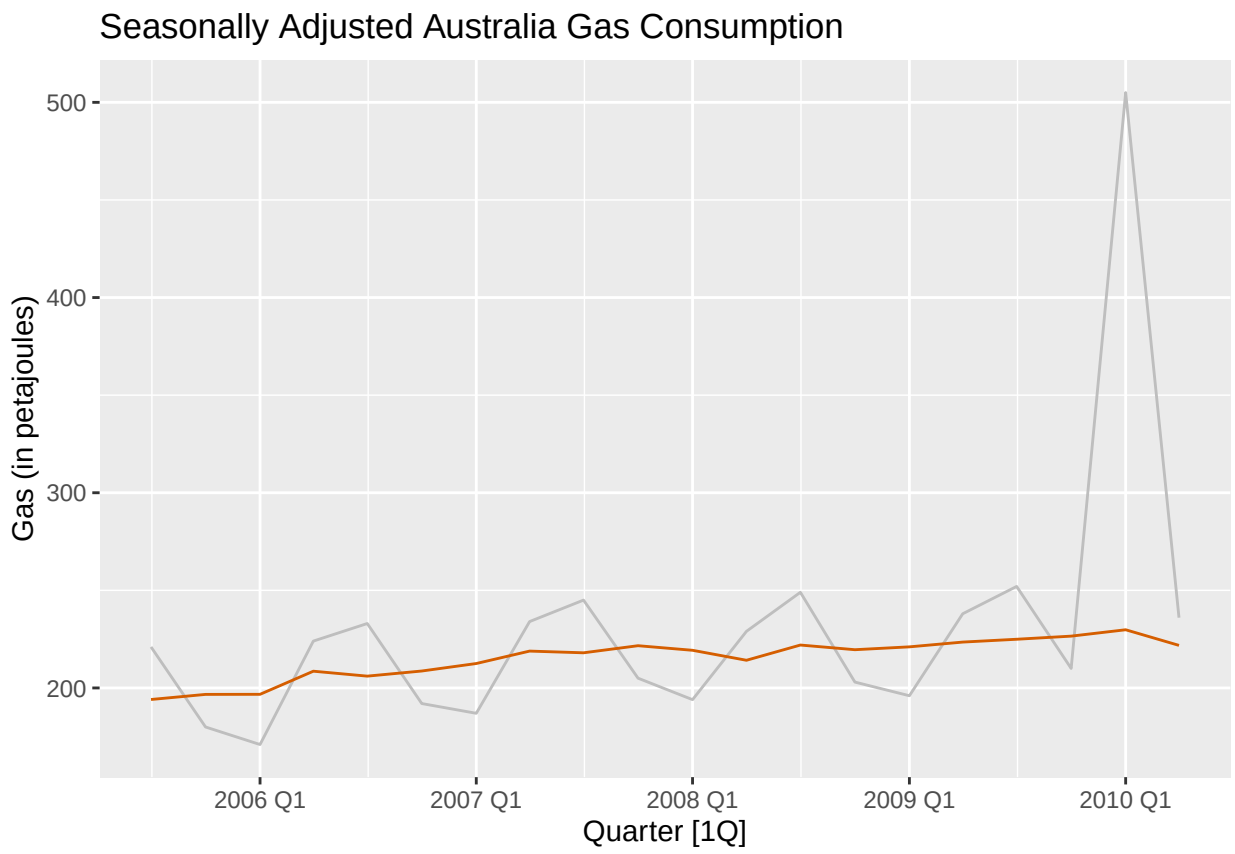
```
# Hint: Use the code provided above but
# adjusted the year and quarter
# Try a few different years and quarters
# to get a better understanding
# (plot at least three different combinations)


# Add outlier at the beginning
gas_outlier <- gas %>%
  mutate(Gas = if_else(Quarter == yearquarter("2006Q2"), Gas + 300, Gas))

# Replot seasonally adjusted data using STL
gas_outlier %>%
```

```r
  autoplot(Gas, color = 'gray') +
  autolayer(
    components(gas_components),
    season_adjust, # plot season adjustment
    color = '#D55E00'
  ) +
  labs(
    y = "Gas (in petajoules)",
    title = "Seasonally Adjusted Australia Gas Consumption"
  )
```



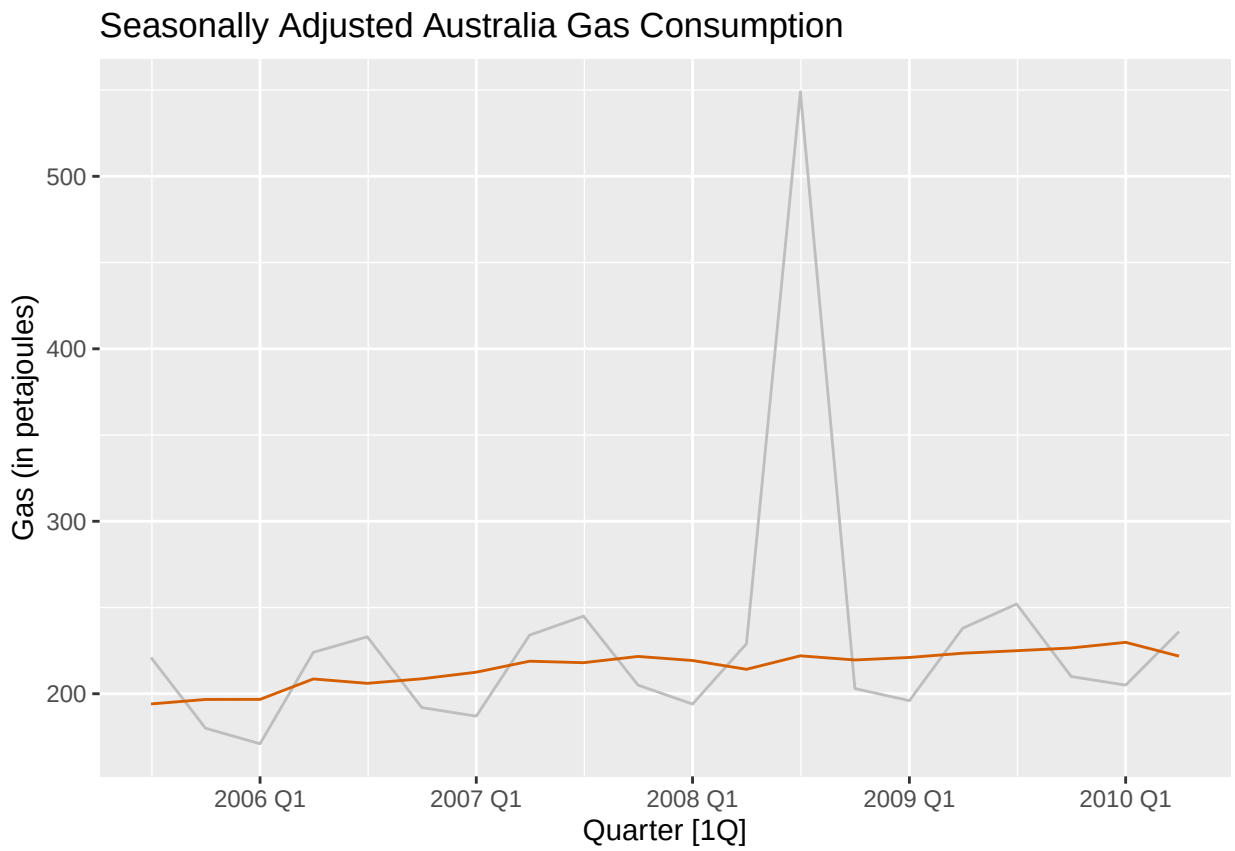

```r
# Hint: Use the code provided above but
# adjusted the year and quarter
# Try a few different years and quarters
# to get a better understanding
# (plot at least three different combinations)

# Add outlier at the end
gas_outlier <- gas %>%
  mutate(Gas = if_else(Quarter == yearquarter("2010Q1"), Gas + 300, Gas))

# Replot seasonally adjusted data using STL
gas_outlier %>%
  autoplot(Gas, color = 'gray') +
  autolayer(
```

```
    components(gas_components),
    season_adjust, # plot season adjustment
    color = '#D55E00'
  ) +
  labs(
    y = "Gas (in petajoules)",
    title = "Seasonally Adjusted Australia Gas Consumption"
  )
```

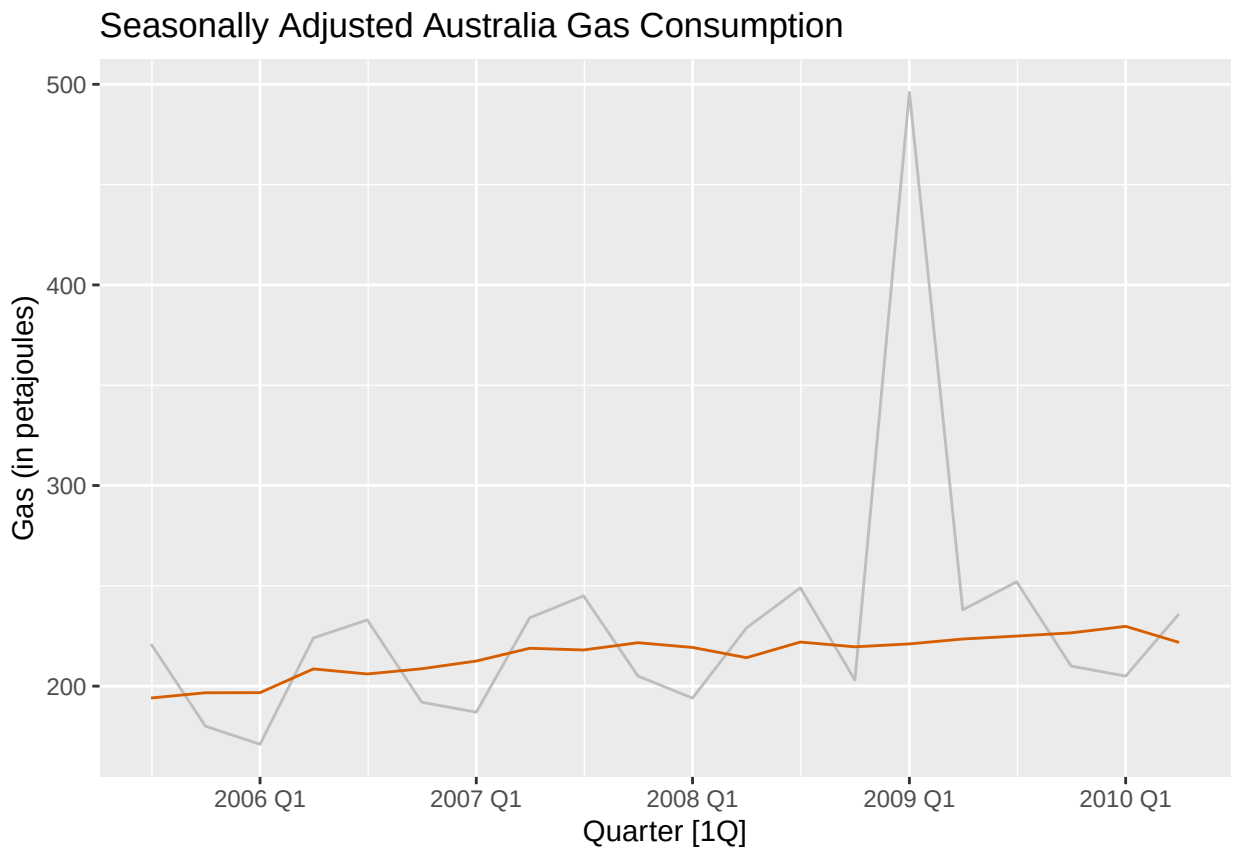



```
# Hint: Use the code provided above but
# adjusted the year and quarter
# Try a few different years and quarters
# to get a better understanding
# (plot at least three different combinations)

# Add outlier in the first 1/3 of the data
gas_outlier <- gas %>%
  mutate(Gas = if_else(Quarter == yearquarter("2008Q3"), Gas + 300, Gas))

# Replot seasonally adjusted data using STL
gas_outlier %>%
  autoplot(Gas, color = 'gray') +
  autolayer(
    components(gas_components),
    season_adjust, # plot season adjustment
```

```
    color = '#D55E00'
  ) +
  labs(
    y = "Gas (in petajoules)",
    title = "Seasonally Adjusted Australia Gas Consumption"
  )
```




```
# Add outlier in last 3/4 of data
gas_outlier <- gas %>%
  mutate(Gas = if_else(Quarter == yearquarter("2009Q1"), Gas + 300, Gas))

# Replot seasonally adjusted data using STL
gas_outlier %>%
  autoplot(Gas, color = 'gray') +
  autolayer(
    components(gas_components),
    season_adjust, # plot season adjustment
    color = '#D55E00'
  ) +
  labs(
    y = "Gas (in petajoules)",
    title = "Seasonally Adjusted Australia Gas Consumption"
  )
```



Comment on general patterns you see (e.g., what happens to the trend? what happens to seasonlity?) No matter where the outlier is in the data, we can see that the rest of the time series still "flattens" out everywhere else. The seasonality is still present, but the trend is not.

Participant 18

Prepare Data

About the data: Emotions over two weeks, queried 4 times per day, between March 11-April 4, 2020. Our goal: Forecast a person's level of feeling worried.

```
# Load data
emotions <- read_csv("clean_ema.csv")

##view data
head(emotions)
```

```
# A tibble: 6 x 26
  ID        Scheduled           Issued Response Duration    Q1    Q2    Q3    Q4
  <chr>     <dttm>              <chr>  <chr>    <chr>    <dbl> <dbl> <dbl> <dbl>
1 User #25~ 2020-03-16 12:00:00 2020-~ 2020-03~ 285.734      1     1     2     1
2 User #25~ 2020-03-16 15:00:00 2020-~ 2020-03~ 1822.742     2     1     2     1
3 User #25~ 2020-03-16 18:00:00 2020-~ 2020-03~ 238.774      1     1     3     1
4 User #25~ 2020-03-16 21:00:00 2020-~ 2020-03~ 967.132      1     1     3     2
```

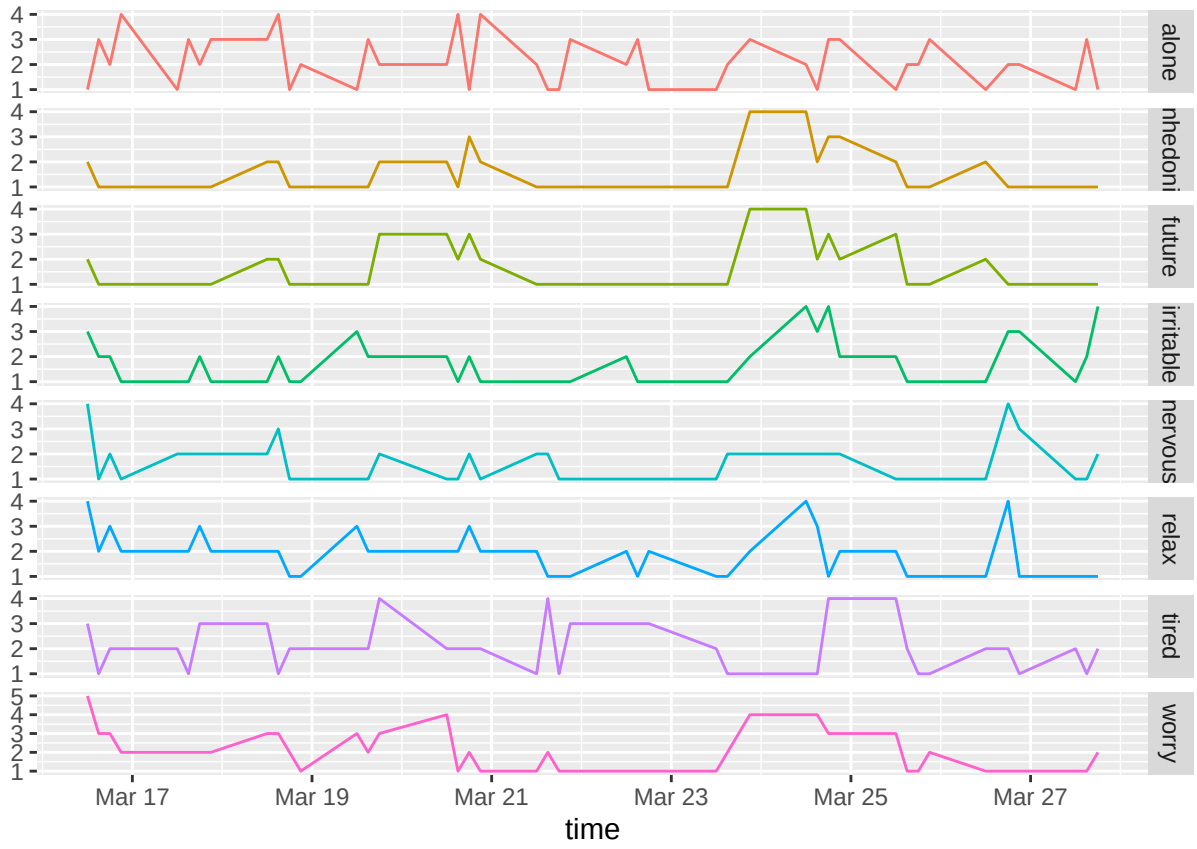
```
5 User #25~ 2020-03-17 12:00:00 2020-~ 2020-03~ 1995.54      1     1     2     1
6 User #25~ 2020-03-17 15:00:00 2020-~ 2020-03~ 721.237      3     2     2     1
# i 17 more variables: Q5 <dbl>, Q6 <dbl>, Q7 <dbl>, Q8 <dbl>, Q9 <dbl>,
#   Q10 <dbl>, Q11 <dbl>, Q12 <dbl>, Q13 <dbl>, Q14 <dbl>, Q15 <dbl>,
#   Q16 <dbl>, Q17 <dbl>, Q18 <dbl>, time <dttm>, Day <date>, beepvar <dbl>
```

```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`
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
]
```

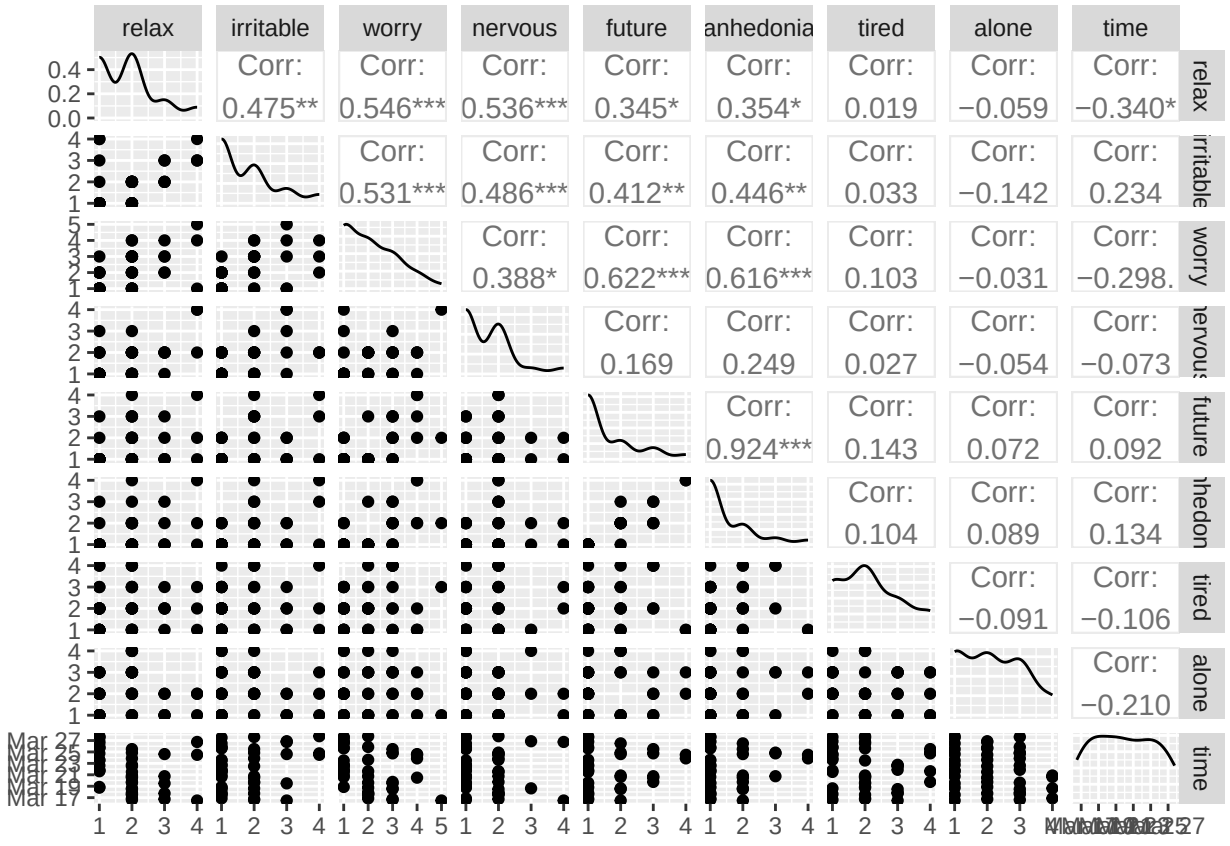
```r
# For modeling residuals
prediction_fill <- ts_fill[
  -c((ts_fill_length - 7):ts_fill_length), # remove last 4 points
]

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

### use this model for the ladt Q
#use ts model to generate data for future values to make a prediction about worry
## say something like : based on these variables, make random, future values
```

```
# Report fit
report(emotion_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
```

```
glance(emotion_fit) %>%
  select(adj_r_squared, CV, AIC, AICc, BIC, log_lik)
```

```
# A tibble: 1 x 6
  adj_r_squared    CV   AIC  AICc   BIC log_lik
          <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
1         0.433 0.838 -10.8 -5.93  5.85   -52.3
```

Report on predictors (were any significant?): Only anhedonia is predicted through thought (beta = 0.92, $p < 0.001$)

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

```
#chat gpt prompt: Can you please generate a dataset made of 12 points() with the values time, relax, ir
                    # "2020-03-30 18:00:00", "2020-03-30 21:00:00",
                    # "2020-03-31 12:00:00", "2020-03-31 15:00:00",
                    # "2020-03-31 18:00:00", "2020-03-31 21:00:00",
                    # "2020-04-01 12:00:00", "2020-04-01 15:00:00",
                    # "2020-04-11 18:00:00", "2020-04-01 21:00:00"

library(lubridate)

gpt_data <- read_csv("gptdata3.csv",
    col_types = cols(relax = col_integer(),
        irritable = col_integer(), worry = col_integer(),
        nervous = col_integer(), future = col_integer(),
        anhedonia = col_integer(), tired = col_integer(),
        alone = col_integer()))

gpt_data$time <- ymd_hms(gpt_data$time) #Make into time format it will like

gpt_data1 <- as_tsibble(gpt_data,index=time) #making into tibble so can plots

# Forecast new scenarios
fc_gpt<- emotion_fit %>%
  forecast(new_data = gpt_data1)
```

Forecast: Method with Random Data

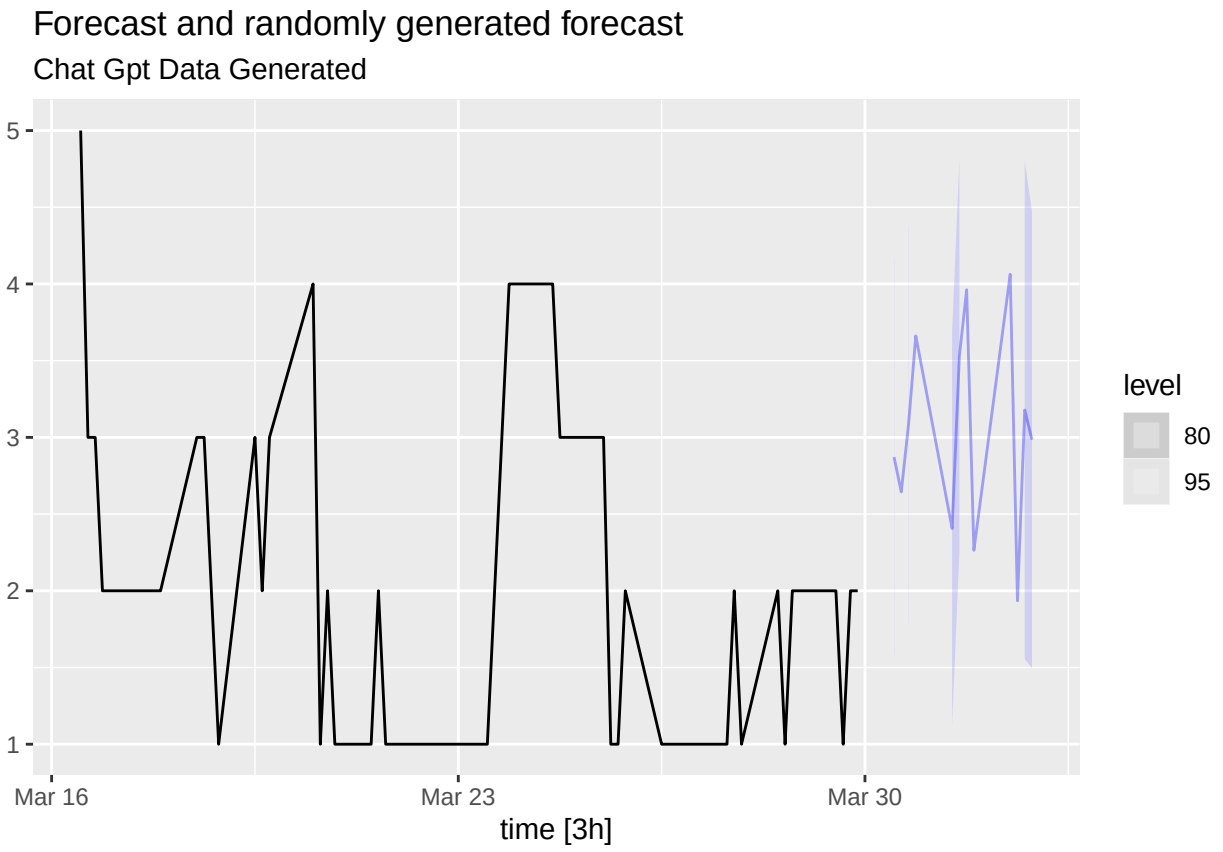
```
# Load the necessary library for working with dates
library(lubridate)

# Set the number of observations
num_observations <- 12

# Generate random times within a specific date range
# start_date <- as.POSIXct("2020-03-30 00:00:00")
# end_date <- as.POSIXct("2023-04-01 18:00:00")
# times <- seq(start_date, end_date, by = "3 hours")

#need to make the days march 30th noon -> april 1st at 3 pm
#4 points per day, 3 days
times_for_swag <- c("2020-03-30 12:00:00", "2020-03-30 15:00:00",
                    "2020-03-30 18:00:00", "2020-03-30 21:00:00",
                    "2020-03-31 12:00:00", "2020-03-31 15:00:00",
                    "2020-03-31 18:00:00", "2020-03-31 21:00:00",
                    "2020-04-01 12:00:00", "2020-04-01 15:00:00",
                    "2020-04-01 18:00:00", "2020-04-01 21:00:00")

# Create a random dataset
set.seed(16) # for reproducibility

random_data <- data.frame(
  time = times_for_swag,
```

```
  relax = sample(1:4, num_observations, replace = TRUE),
  irritable = sample(1:4, num_observations, replace = TRUE),
  worry = sample(1:4, num_observations, replace = TRUE),
  nervous = sample(1:4, num_observations, replace = TRUE),
  future = sample(1:4, num_observations, replace = TRUE),
  anhedonia = sample(1:4, num_observations, replace = TRUE),
  tired = sample(1:4, num_observations, replace = TRUE),
  alone = sample(1:4, num_observations, replace = TRUE) )

#random_data$time <- as.POSIXct(random_data$time) #makes time var

random_data$time <- ymd_hms(random_data$time) #Make into time format it will like

random_data1 <- as_tsibble(random_data, index=time) #making into tibble so can plot

# Make future possibilities
#random_scenarios <- scenarios(random_data)

# Forecast new scenarios
fc_random<- emotion_fit %>%
  forecast(new_data = random_data1)
```

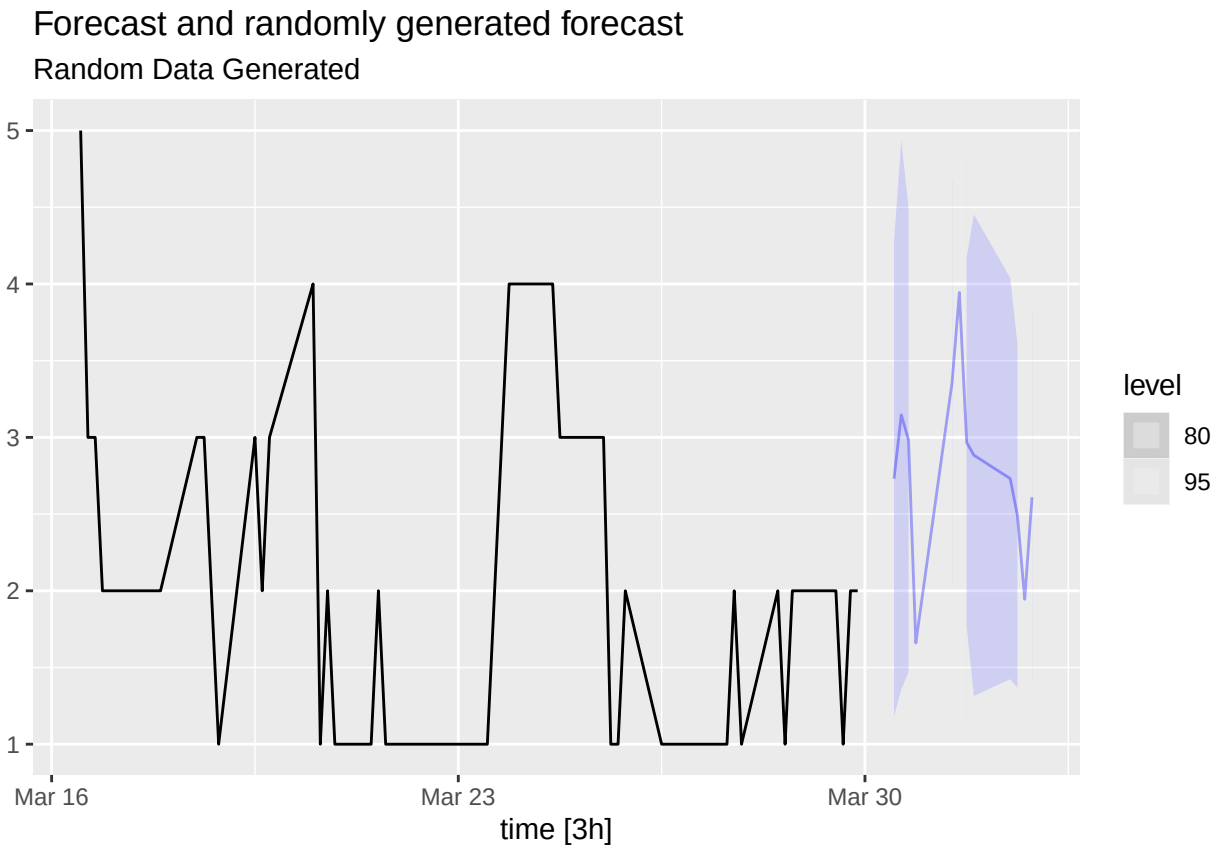
Forecast: Plot Forecast

```
data <- as_tsibble(data,index=time) #making into tibble so can plots
```

```
# Plot forecasts simultaneously
data %>%
  autoplot(worry) +
  autolayer(fc_gpt, alpha = 0.333) +
  labs(
    # No y-axis label
    y = NULL,
    # Change title
    title = "Forecast and randomly generated forecast",
    subtitle = "Chat Gpt Data Generated"
  ) +
  scale_y_continuous(
    limits = c(1, 5), # minimum and maximum of y-axis
    breaks = seq(1, 5, 1) # breaks on y-axis
  )
```




```
# Plot forecasts simultaneously
data %>%
  autoplot(worry) +
  autolayer(fc_random, alpha = 0.333) +
  labs(
    # No y-axis label
    y = NULL,
    # Change title
    title = "Forecast and randomly generated forecast",
    subtitle = "Random Data Generated"
  ) +
  scale_y_continuous(
    limits = c(1, 5), # minimum and maximum of y-axis
    breaks = seq(1, 5, 1) # breaks on y-axis
  )
```



Comment on each of the predictions. Do any seem to work well? Which do you think performed best and why?

It seems like the Chat GPT data is performing better. The error bars are smaller for that plot than the randomly generated data.

Forecast: Evaluate Forecast

```
# Merge or join the datasets based on the common timestamp or time column
merged_data <- inner_join(fc_gpt, data, by = "time")

# Calculate accuracy measures (e.g., RMSE, MAE, etc.) based on the merged data
# Replace "forecasted_values" and "actual_values" with your specific column names
rmse <- sqrt(mean((merged_data$fc_gpt - merged_data$data)^2))
mae <- mean(abs(merged_data$fc_gpt - merged_data$data))

# Print the accuracy measures
cat("RMSE:", rmse, "\n")
cat("MAE:", mae, "\n")
```

```
# Merge or join the datasets based on the common timestamp or time column
merged_data <- inner_join(fc_random, data, by = "time")
```

```r
# Calculate accuracy measures (e.g., RMSE, MAE, etc.) based on the merged data
# Replace "forecasted_values" and "actual_values" with your specific column names
rmse <- sqrt(mean((merged_data$fc_gpt - merged_data$data)^2))
mae <- mean(abs(merged_data$fc_gpt - merged_data$data))

# Print the accuracy measures
cat("RMSE:", rmse, "\n")
```

```
## RMSE: NaN
```

```r
cat("MAE:", mae, "\n")
```

```
## MAE: NaN
```

> Which method performed best based on these evaluation measures? Justify which method you think performed best

Because of how I coded the new values, the typical metrics won't work. But, we can imply from the charts that in general, random forecasts are not the best way to go. From just looking at the plots, it seems like the Chat GPT data is performing better. The error bars are smaller for that plot than the randomly generated data.