Practical 4 - Physiological Sensors

Alexandra Del Favero-Campbell

2024-06-10

There are 3-4 packages you will need to install for today's practical: `install.packages(c("xgboost", "eegkit", "forecast", "tseries", "caret"))` apart from that everything else should already be available on your system.

If you are using a newer Mac you may have to also install quartz to have everything work (do this if you see errors about `X11` during install/execution).

I will endeavour to use explicit imports to make it clear where functions are coming from (functions without `library_name::` are part of base R or a function we've defined in this notebook).

```
## Loading required package: eegkitdata

## Loading required package: bigsplines

## Loading required package: quadprog

## Loading required package: ica

## Loading required package: rgl

## Loading required package: signal

##
## Attaching package: 'signal'

## The following objects are masked from 'package:stats':
##
##     filter, poly

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

## Loading required package: ggplot2

## Loading required package: lattice

##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:signal':
##
##     filter

## The following object is masked from 'package:xgboost':
##
##     slice

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

##
## Attaching package: 'purrr'

## The following object is masked from 'package:caret':
##
##     lift
```

EEG Eye Detection Data

One of the most common types of medical sensor data (and one that we talked about during the lecture) are Electroencephalograms (EEGs).
These measure mesoscale electrical signals (measured in microvolts) within the brain, which are indicative of a region of neuronal activity. Typically, EEGs involve an array of sensors (aka channels) placed on the scalp with a high degree of covariance between sensors.

As EEG data can be very large and unwieldy, we are going to use a relatively small/simple dataset today from this paper.

This dataset is a 117 second continuous EEG measurement collected from a single person with a device called a "Emotiv EEG Neuroheadset". In combination with the EEG data collection, a camera was used to record whether person being recorded had their eyes open or closed. This was eye status was then manually annotated onto the EEG data with `1` indicated the eyes being closed and `0` the eyes being open. Measures microvoltages are listed in chronological order with the first measured value at the top of the dataframe.

Let's parse the data directly from the `h2o` library's (which we aren't actually using directly) test data S3 bucket:

```
eeg_url <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/eeg/eeg_eyestate_splits.csv"
eeg_data <- read.csv(eeg_url)

# add timestamp
Fs <- 117 / nrow(eeg_data)
eeg_data <- base::transform(eeg_data, ds = seq(0, 116.99999, by = Fs), eyeDetection = as.factor(eyeDete
print(table(eeg_data$eyeDetection))
```

```
##
##    0    1
## 8257 6723
```

```r
# split dataset into train, validate, test
eeg_train <- subset(eeg_data, split == 'train', select = -split)
print(table(eeg_train$eyeDetection))
```

```
##
##    0    1
## 4916 4072
```

```r
eeg_validate <- subset(eeg_data, split == 'valid', select = -split)
eeg_test <- subset(eeg_data, split == 'test', select = -split)
```

0 Knowing the `eeg_data` contains 117 seconds of data, inspect the `eeg_data` dataframe and the code above too and determine how many samples per second were taken?
Based on the code output from above, we know that there are a total of 14980 samples. The total duration of the EEG data is 117 seconds. Based on our calculations (shown below), there were approximately 128 samples per second taken.

```r
#Determining the Sampling Rate (samples per second)
# Calculate the number of rows
num_samples <- nrow(eeg_data)
num_samples #Total number of samples
```

```
## [1] 14980
```

```r
total_duration <- 117  # seconds
sampling_rate <- num_samples / total_duration
sampling_rate
```

```
## [1] 128.0342
```

1 How many EEG electrodes/sensors were used?

There were 14 electrodes/sensors used. This is further confirmed when reviewing the paper where this data came from.

```r
colnames(eeg_data)
```

```
##  [1] "AF3"          "F7"           "F3"           "FC5"          "T7"
##  [6] "P7"           "O1"           "O2"           "P8"           "T8"
## [11] "FC6"          "F4"           "F8"           "AF4"          "eyeDetection"
## [16] "split"        "ds"
```

Exploratory Data Analysis

Now that we have the dataset and some basic parameters let's begin with the ever important/relevant exploratory data analysis.

First we should check there is no missing data!

```
sum(is.na(eeg_data))
```

```
## [1] 0
```

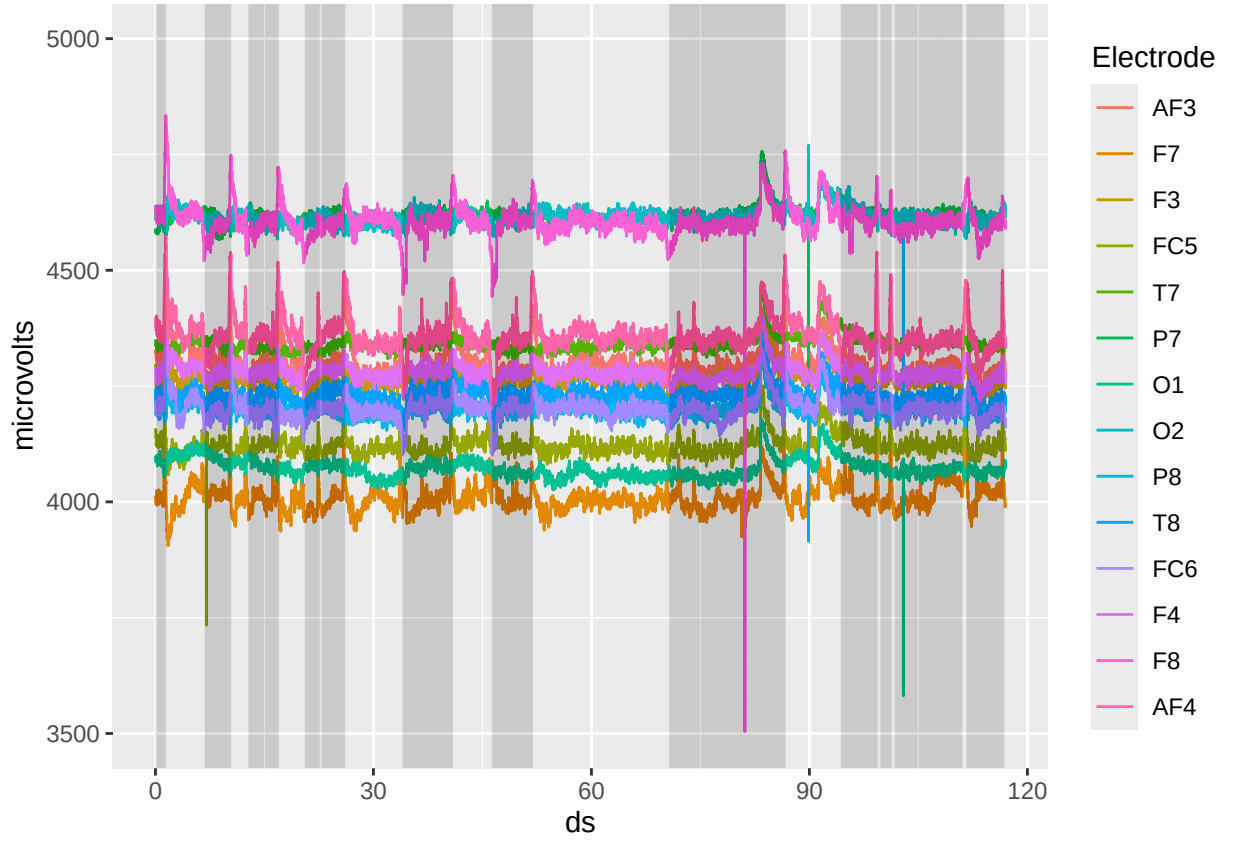
Great, now we can start generating some plots to look at this data within the time-domain.

First we use `reshape2::melt()` to transform the `eeg_data` dataset from a wide format to a long format expected by `ggplot2`.

Specifically, this converts from "wide" where each electrode has its own column, to a "long" format, where each observation has its own row. This format is often more convenient for data analysis and visualization, especially when dealing with repeated measurements or time-series data.

We then use `ggplot2` to create a line plot of electrode intensities per sampling time, with the lines coloured by electrode, and the eye status annotated using dark grey blocks.

```
melt <- reshape2::melt(eeg_data %>% dplyr::select(-split), id.vars=c("eyeDetection", "ds"), variable.nam


ggplot2::ggplot(melt, ggplot2::aes(x=ds, y=microvolts, color=Electrode)) +
  ggplot2::geom_line() +
  ggplot2::ylim(3500,5000) +
  ggplot2::geom_vline(ggplot2::aes(xintercept=ds), data=dplyr::filter(melt, eyeDetection==0), alpha=0.0
```

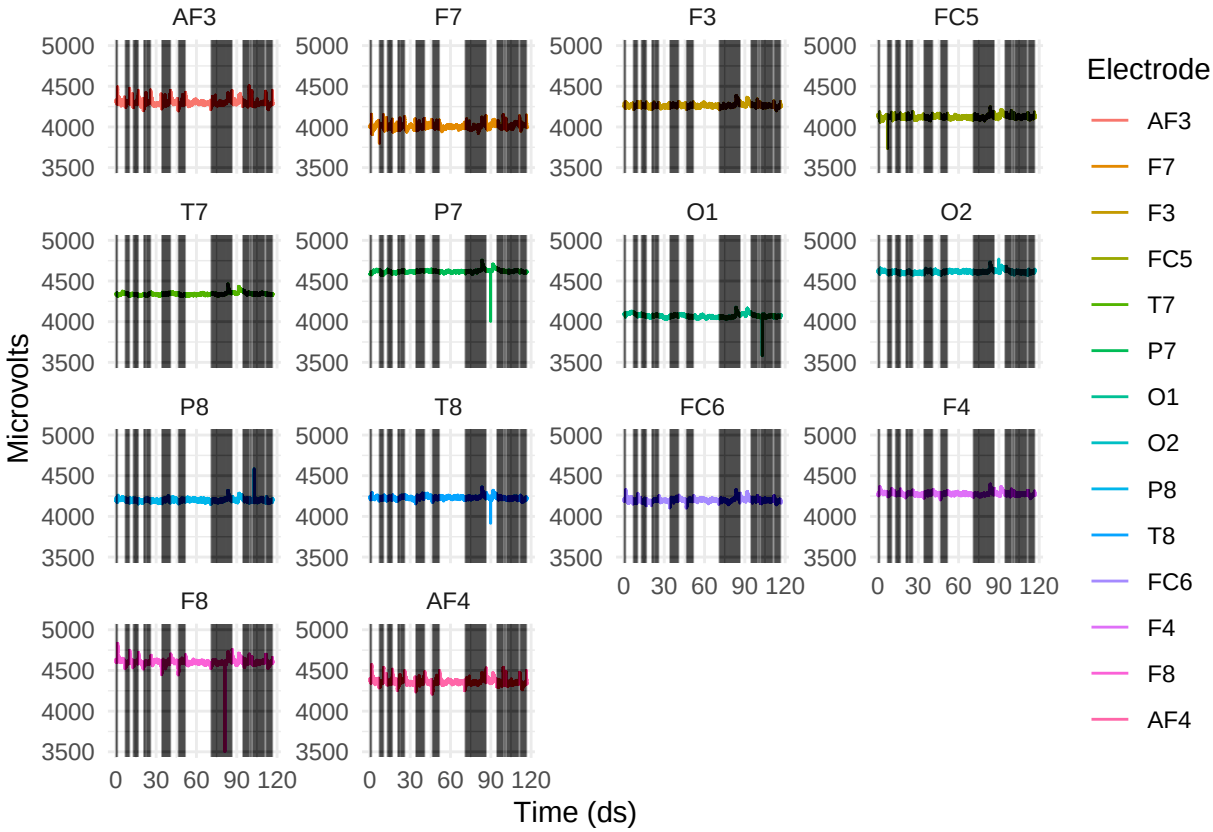


2 Do you see any obvious patterns between eyes being open (dark grey blocks in the plot) and the EEG intensities?

There are noticeable fluctuations in EEG intensities overtime, and the fluctuations appear to be more prominent in certain electrodes, especially as compared to others. If we take a further look at the patterns of each electrode in separate plots (see plots below), we confirm that different electrodes show varying levels of activity, with some having higher intensity and amounts of activity than others do. For example, some electrodes, such as F8 and AF3, show more pronounced changes when the eyes are closed. In other words, we see somewhat distinct increases in EEG intensity during periods when the eyes are closed; this could potentially suggest a correlation between the eyes closed state and increased EEG intensity in this particular electrode region. Moreover, similar to F8, certain electrodes, like AF3 and F3, show somewhat of an increase in EEG intensities when the eyes are closed too. However, the changes are less pronounced in comparison to F8. On the other hand, other electrodes, such as T7, P7, O1, and O2, do not show extremely significant differences between the eyes closed and the eyes opened states. They show more stable EEG intensity activity levels and display less distinctions between the closed eye state and the opened eye state. Furthermore, there seems to be a spike in EEG intensity just before and after the dark grey blocks.

In conclusion, there is a recognizable pattern suggesting that specific electrodes, such as F8 and AF3, display increased EEG activity when the eyes are in a closed state, potentially indicating that these corresponding specific regions of the brain become more active when visual inputs are reduced (i.e., when the eyes are closed). Furthermore, many other electrodes show way less variability, which suggests that they may be less sensitive to the state of the eye. It could also suggest that these less variable electrodes are recording from specific regions of the brain that are less affected by whether the eyes are open or not.

```r
# Load required libraries if not already loaded
library(dplyr)
library(ggplot2)
library(reshape2)  # Make sure reshape2 package is loaded for melt function

# Assuming eeg_data is your dataset containing EEG data

# Reshape data to long format
melt <- reshape2::melt(eeg_data %>% select(-split), id.vars = c("eyeDetection", "ds"), variable.name =

# Plot data separately for each electrode
ggplot(melt, aes(x = ds, y = microvolts, color = Electrode)) +
  geom_line() +
  ylim(3500, 5000) +  # Adjust ylim as needed
  geom_vline(aes(xintercept = ds), data = filter(melt, eyeDetection == 0), alpha = 0.005) +
  facet_wrap(~ Electrode, scales = "free_y") +  # Facet by Electrode
  labs(x = "Time (ds)", y = "Microvolts", color = "Electrode") +  # Adjust axis labels
  theme_minimal()
```



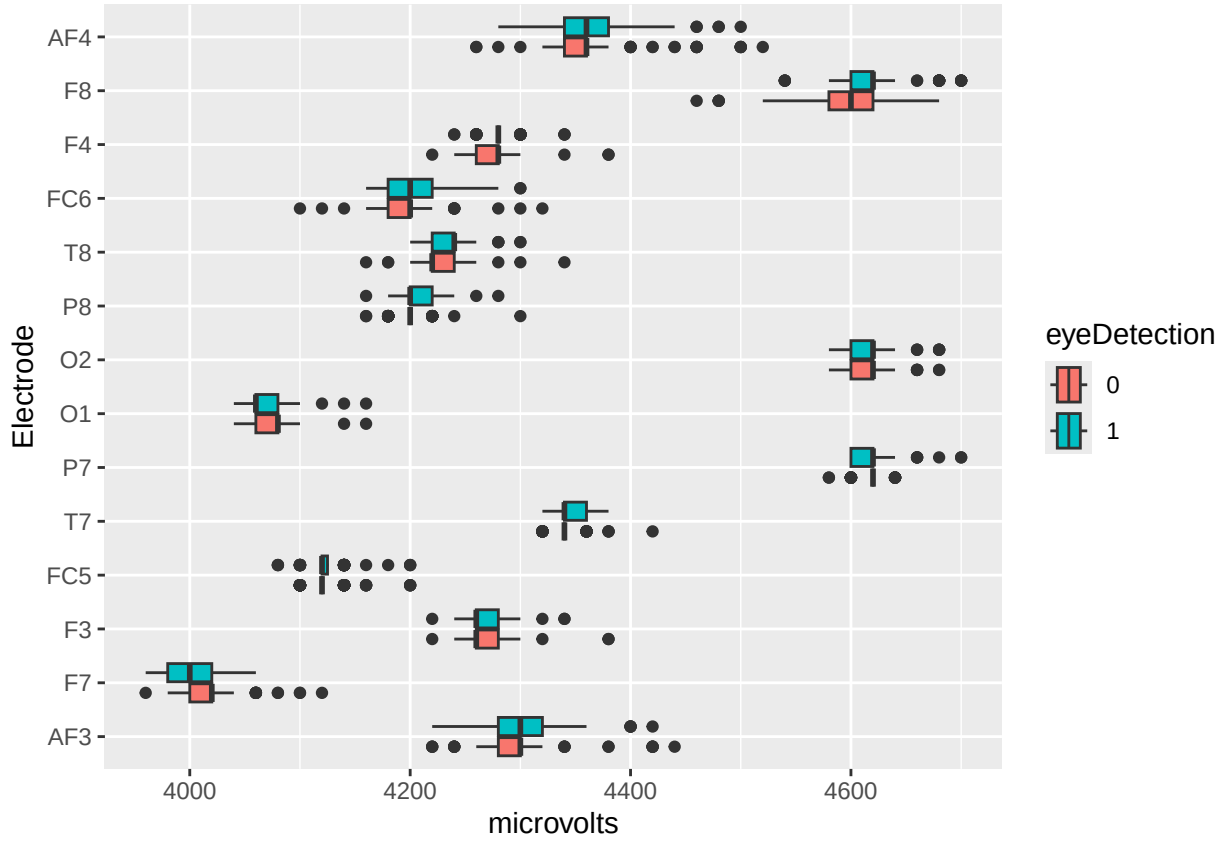
3 Similarly, based on the distribution of eye open/close state over time to anticipate any temporal correlation between these states?

Overall, there does seem to maybe be a temporal correlation between these states. There seems to be a regular and periodic nature to the transitions that we see. The periods when the eyes are open occur in clusters for varying amounts of time, and the plot shows us that there are alternating periods between when the eyes are open and when they are closed. However, we do see that the clustering patterns of the eyes closed periods seem to be set in brief periods in which they tend to remain closed for brief periods of time before the eyes open again. Moreover, there seems to be more of a consistent alteration between the eyes being closed and then open throughout the 117 second duration period. Thus, temporally, it seems like there is a certain regularity to the transitions between when the eyes are open and when they close, which seems indicative of a possible cyclical or periodic behavioral pattern (e.g., due to potential natural blinking patterns or deliberate eye opening and closing in response to someone's instructions or resting periods).

Let's see if we can directly look at the distribution of EEG intensities and see how they related to eye status.

As there are a few extreme outliers in voltage we will use the `dplyr::filter` function to remove values outwith of 3750 to 50003. The function uses the `%in%` operator to check if each value of microvolts is within that range. The function also uses the `dplyr::mutate()` to change the type of the variable eyeDetection from numeric to a factor (R's categorical variable type).

```
melt_train <- reshape2::melt(eeg_train, id.vars=c("eyeDetection", "ds"), variable.name = "Electrode", va

# filter huge outliers in voltage
filt_melt_train <- dplyr::filter(melt_train, microvolts %in% (3750:5000)) %>% dplyr::mutate(eyeDetectio

ggplot2::ggplot(filt_melt_train, ggplot2::aes(y=Electrode, x=microvolts, fill=eyeDetection)) + ggplot2:
```



Plots are great but sometimes so it is also useful to directly look at the summary statistics and how they related to eye status. We will do this by grouping the data based on eye status and electrode before calculating the statistics using the convenient `dplyr::summarise` function.

```
filt_melt_train %>% dplyr::group_by(eyeDetection, Electrode) %>%
    dplyr::summarise(mean = mean(microvolts), median=median(microvolts), sd=sd(microvolts)) %>%
    dplyr::arrange(Electrode)
```

```
## `summarise()` has grouped output by 'eyeDetection'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 28 x 5
## # Groups:   eyeDetection [2]
##    eyeDetection Electrode  mean median    sd
##    <fct>        <fct>     <dbl>  <dbl> <dbl>
##  1 0            AF3       4294.   4300  35.4
##  2 1            AF3       4305.   4300  34.4
##  3 0            F7        4015.   4020  28.4
##  4 1            F7        4007.   4000  24.9
##  5 0            F3        4268.   4260  20.9
##  6 1            F3        4269.   4260  17.4
##  7 0            FC5       4124.   4120  17.3
##  8 1            FC5       4124.   4120  19.2
##  9 0            T7        4341.   4340  13.9
## 10 1            T7        4342.   4340  15.5
## # i 18 more rows
```

4 Based on these analyses are any electrodes consistently more intense or varied when eyes are open?

Most of the electrodes seem consistently the same in EEG intensity between eyes open status and eyes closed status, respective of the electrode. Although some electrodes may show slight differences in mean EEG intensities and variability (e.g., standard deviation), many (e.g., electrodes AF3, F3, FC5, T7, P8, T8, FC6, and F4) do not show what I would consider very significant differences or any difference at all.

However, there are a few that vary when it comes to the eyes being open versus the eyes being shut. For example, electrode F7 shows that the eyes being open has a slightly higher electrode intensity at around 4014.8 microvolts, whereas having the eyes closed showed an electrode intensity of around 4006.7 microvolts. It also shows less variability with the eyes closed (SD=24.86 microvolts) versus when the eyes were open (SD=28.38 microvolts); this indicates that the eyes being open may have slightly higher variability and EEG intensity. Electrodes O1 and P7 showed a similar mean between the eyes being open (around 4072.3 microvolts and 4617.86 microvolts, respectively) and the eyes being closed (around 4071.8 microvolts and 4617.61 microvolts, respectively), but more variability and wider spread when the eyes are closed (SD=24.29 and 19.13, respectively) versus when they were open (SD=16.93 and 11.18, respectively). Electrode F8 showed a slightly different trend with a slightly higher electrode intensity when the eyes were closed (mean of 4615.2 microvolts) than when the eyes were open (mean of 4595.7 microvolts). Electrode AF4 showed a slightly greater mean EEG intensity and less variability when the eyes were closed (4369.455±36.01 microvolts) than when they were open (4363.894±46.59 microvolts). Lastly, Electrode O2 showed slightly higher mean EEG intensity and more variability when the eyes were closed (4619.813±20.28 microvolts) as compared to when the eyes were opened (4615.71±17.46 microvolts).

Therefore, some varied in their patterns, but overall these variations were quite minimal. In terms of intensity, overall, most electrodes did not show too much of a significant difference in mean EEG intensities in terms of when the eyes were opened versus when the eyes were closed. However, some electrodes, such as electrode F8 and electrode AF4 did show somewhat higher EEG intensities when the eyes were closed. In terms of variability, there were quite a few electodes that showed more variability when the eyes were closed versus when they were open, including electrodes F8, O1, P7, and O2). There does not, however, seem to be a very consistent pattern across all of the electrodes that indicates that EEG readings are higher in intensity when the eyes are open versus when they are closed. However, we do see that potentially variability tends to maybe be higher when the eyes are closed, but only for certain electrodes.

Time-Related Trends As it looks like there may be a temporal pattern in the data we should investigate how it changes over time.

First we will do a statistical test for stationarity:

```
apply(eeg_train, 2, tseries::adf.test)
```

```
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
```

```
## $AF3
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.669, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -12.079, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F3
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -11.587, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $FC5
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -11.122, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $T7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.5644, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $P7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.7, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
```

```
## $O1
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -7.9495, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $O2
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.3537, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $P8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.69, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $T8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.9902, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $FC6
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -8.6708, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F4
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -10.189, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
```

```
## $F8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.642, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $AF4
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.755, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $eyeDetection
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -4.8699, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $ds
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -2.4104, Lag order = 20, p-value = 0.4045
## alternative hypothesis: stationary
```

5 What is stationarity?

The idea of "stationarity" asks the quesiton of whether the given time series that you are working with actually shows change overtime in a meaningful way. Short-term fluctuations (e.g., seasonality) can happen, but if what you are seeing is consistent, for example every year or every breath exhalation, and you see so really long-term trends or irregular cycling component, then you are probably dealing with a stationary signal as time does not really seem to affect the underlying probability distribution of that given signal.

Put simply, stationarity means that the statistical properties of a process that is generating a time series does not change over time. Technically, a signal is defined as "stationary" if that signal's statistical characteristics do *not* vary over time. This means, consequently, that things such as means and variance also do not change over time for that signal. A potential example to try to explain this is if we imagine a reservoir where inflow and outflow are balanced, which allows it to maintain a constant average water level over time. If we are measuring the water level of that reservoir overtime, we may see it fluctuate over time slightly due to things like waves and minor variations in flow; however, the overall statistical properties, such as mean and variance, are very likely to remain stable, making this signal a good example of at least weak stationarity. Conversely, nonstationarity basically happens when the statistical characteristics of a signal *do* change with time. To clarify, stationarity does not mean that, that given series does not change over time; it just means that the *way* in which it changes does not itself change over time (Affek, 2019). If we think about it kind of like an algebraic linear function, the linear function changes as the variable, x, grows; however, the *way*

in which it grows and changes remains constant in the idea that it has a constant slope or one value that captures that rate of change over time (Affek, 2019).

It is important to check a time series for stationarity, particularly before applying statistical and machine learning models because there are many models that have the assumption that the input data is stationary. Thus, if you were to input non-stationary data, it could lead to some misleading results and poor modeling performance.

6 Why are we interested in stationarity? What do the results of these tests tell us? (ignoring the lack of multiple comparison correction...)

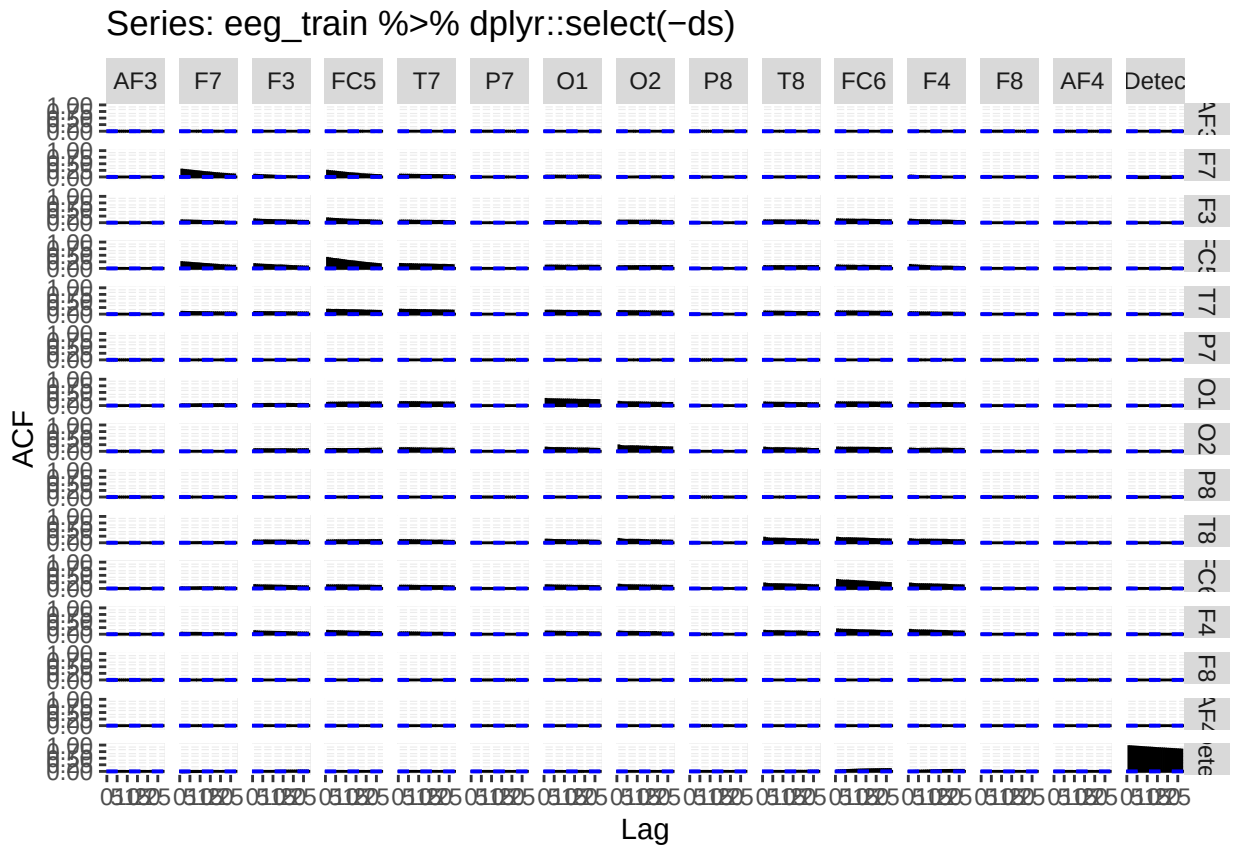
As mentioned previously, we are interested in stationarity within a time series context because its implications in a time series analysis are critical when it comes to accurate modeling and predicting. Many time series models (e.g., AutoRegressive Integrated Moving Average (ARIMA)) assume that the data you are using is stationary. This also true when it comes to thinking about predicting and forecasting. Accurate forecasting usually relies on the assumption of stationarity, as non-stationay data often leads to inconsistent and biased predictions. Since stationarity means statistical properties are constant overtime, past data is able to be used to predict future data. This allows for more reliable statistical inferencing. Thus, if the data you are using in these sorts of models is actually not stationary, then you are using an inappropriate model for your data or you may need to transform you data before using such a model to achieve some semblance of stationarity first.

With regard to the second portion of this question, the results of our tests are basically helping us check for stationarity in a time series. The Augmented Dickey-Fuller (ADF) test that we performed basically tells us that the electrodes (i.e., AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, and AF4) and the eyeDetection variable all yielded a p-value of 0.01, which indicates that these time series are stationary. This essentially means that their statistical properties, such as mean and variance, do not change over time, which is perfect for time series modeling and forecasting. However, the ADF test also found that the ds variable has a p-value of 0.4045, which is indicative of nonstationarity. This basically means that the statistical properties of the ds variable change over time, potentially suggesting the presence of seasonality and trends. This variable, thus, may require some additional preprocessing and transforming, using something like differencing, to make it more stationary before it is able to be used for time series modeling.

Then we may want to visually explore patterns of autocorrelation (previous values predict future ones) and cross-correlation (correlation across channels over time) using `forecast::ggAcf` function.

The ACF plot displays the cross- and auto-correlation values for different lags (i.e., time delayed versions of each electrode's voltage timeseries) in the dataset. It helps identify any significant correlations between channels and observations at different time points. Positive autocorrelation indicates that the increase in voltage observed in a given time-interval leads to a proportionate increase in the lagged time interval as well. Negative autocorrelation indicates the opposite!

```
forecast::ggAcf(eeg_train %>% dplyr::select(-ds))
```



7 Do any fields show signs of strong autocorrelation (diagonal plots)? Do any pairs of fields show signs of cross-correlation? Provide examples.

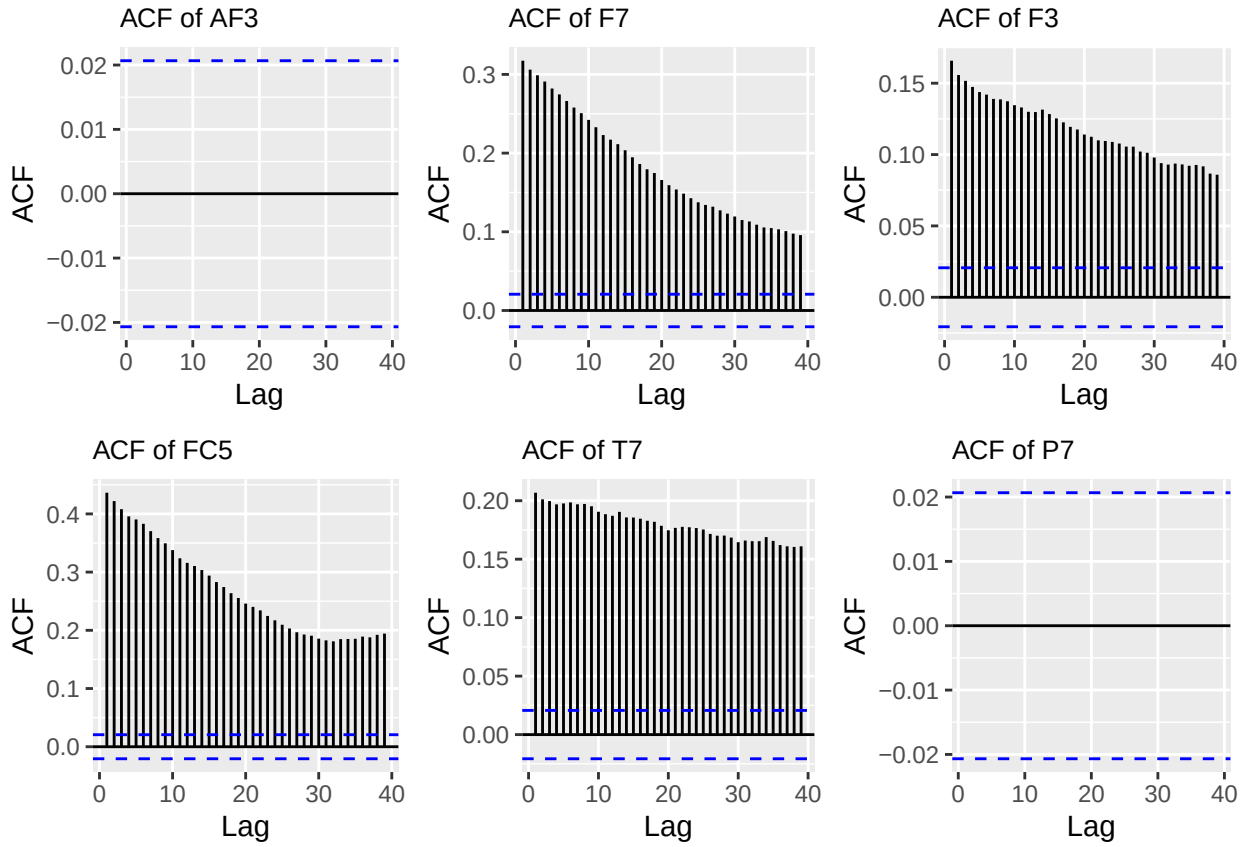
```
# Load required libraries
library(forecast)
library(ggplot2)
library(dplyr)
library(gridExtra)
```

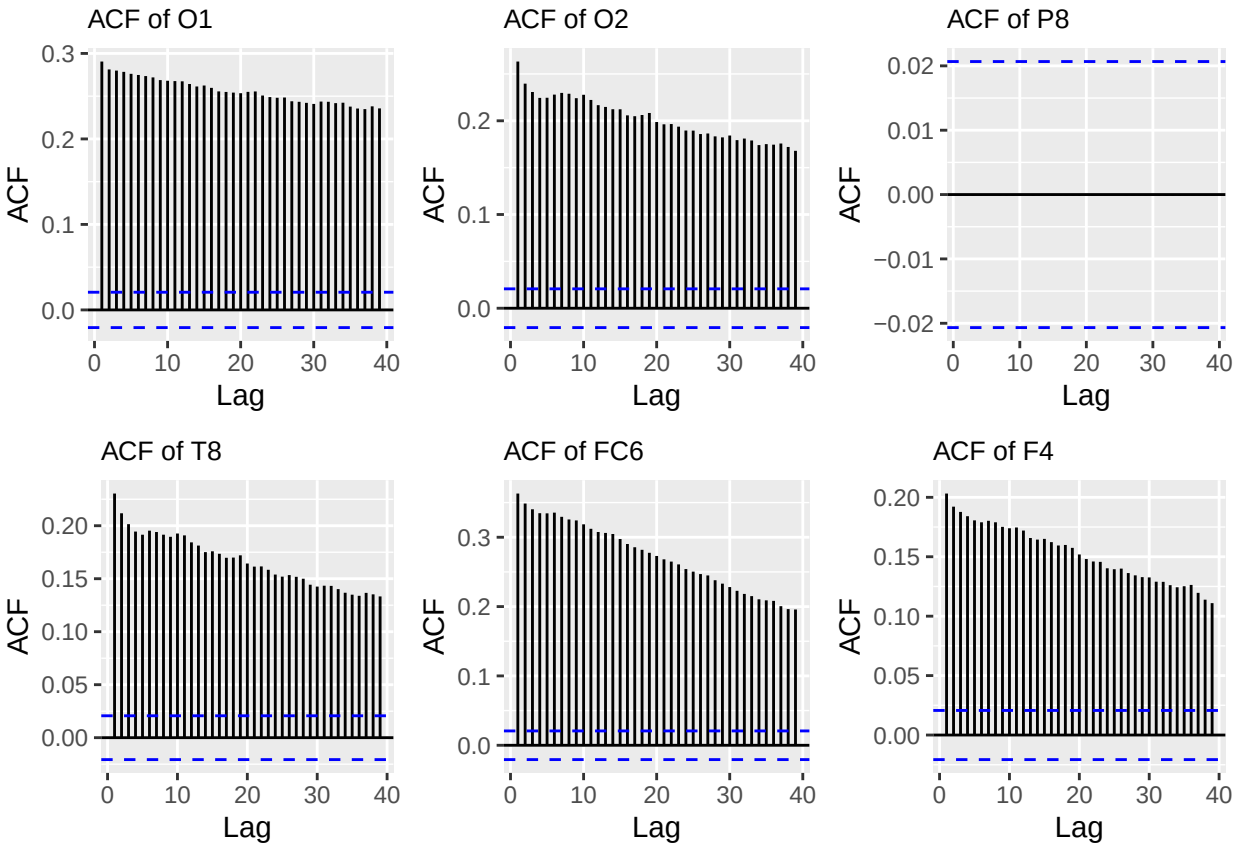
```
##
## Attaching package: 'gridExtra'
```

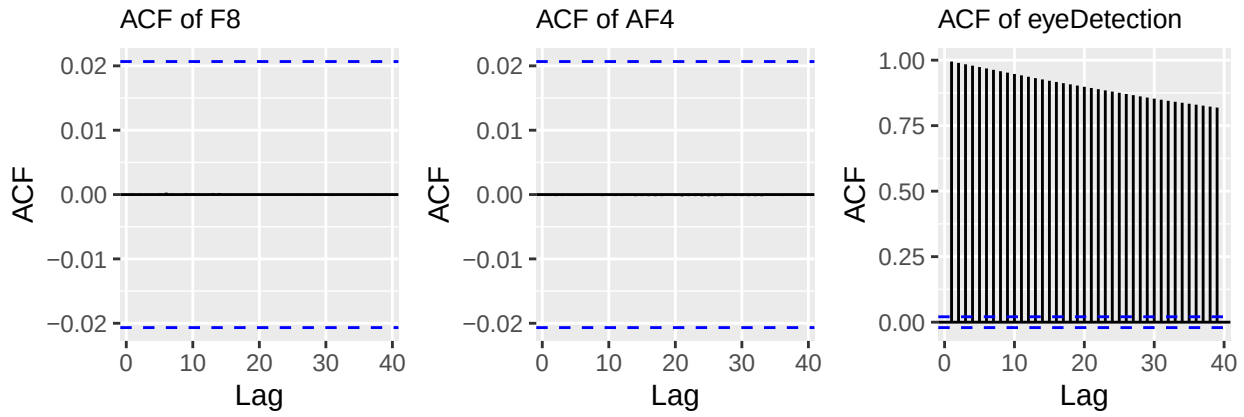
```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
# Select columns excluding 'ds'
eeg_train_selected <- eeg_train %>% select(-ds)

# Create a list to store the plots
acf_plots <- list()

# Loop over each column to create individual ACF plots
for (column_name in colnames(eeg_train_selected)) {
  # Extract the column data
```

```r
  column_data <- eeg_train_selected[[column_name]]

  # Create the ACF plot
  acf_plot <- ggAcf(column_data) +
    ggtitle(paste("ACF of", column_name)) +
    theme(plot.title = element_text(size = 10))  # Adjust title size here

  # Add the plot to the list
  acf_plots[[length(acf_plots) + 1]] <- acf_plot
}

# Print plots in pages of 6
plot_pages <- split(acf_plots, ceiling(seq_along(acf_plots) / 6))

for (page in plot_pages) {
  do.call("grid.arrange", c(page, ncol = 3, nrow = 2))
}
```




ACF of O1
ACF of O2
ACF of P8
ACF of T8
ACF of FC6
ACF of F4
ACF
Lag

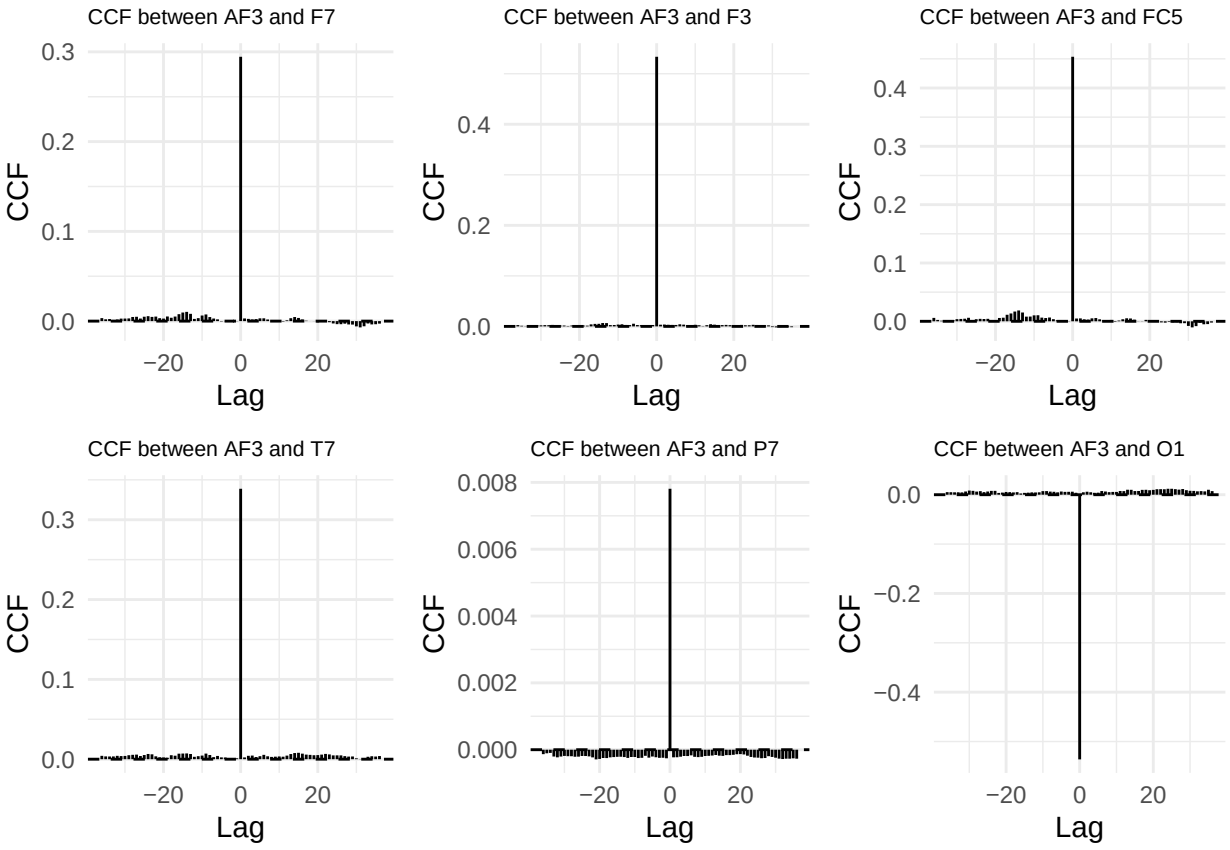


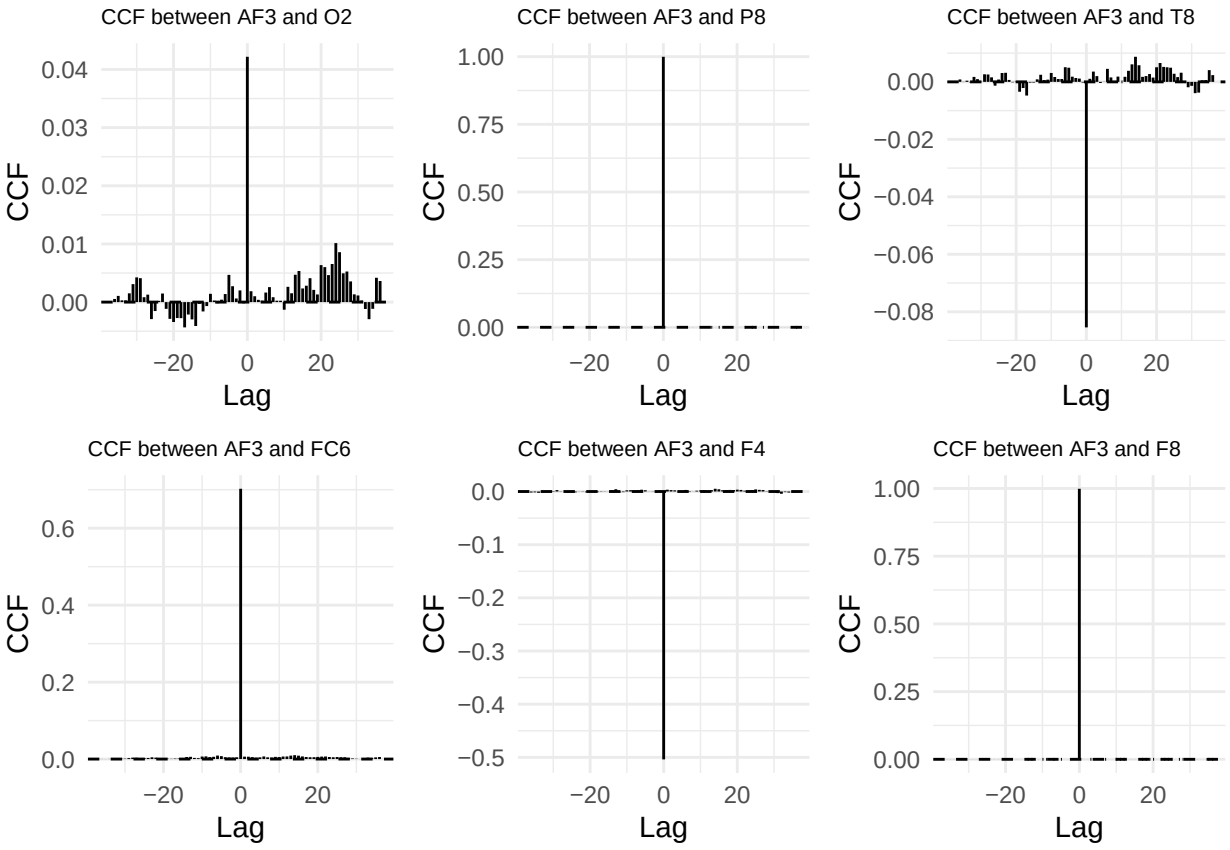
The ACF plot for AF3 shows that all autocorrelations are pretty close to 0 and show no significant spikes, all falling well within the 95% confidence bounds, indicating no significant autocorrelation. The same can be said of the ACF plots for electrodes AF4, P7, P8, and F8. The ACF plots for eyeDetection and F3 show significant autocorrelations at several lags, staying above the confidence bounds for multiple lags. This indicates strong autocorrelation. The same can be said for electrodes F4, F7, FC5, O2, O1, FC6, T7, and T8. Thus, the variables that seem to show strong autocorrelation are eyeDetection and electrodes F3, F4, F7, FC5, O2, O1, FC6, T7, and T8.

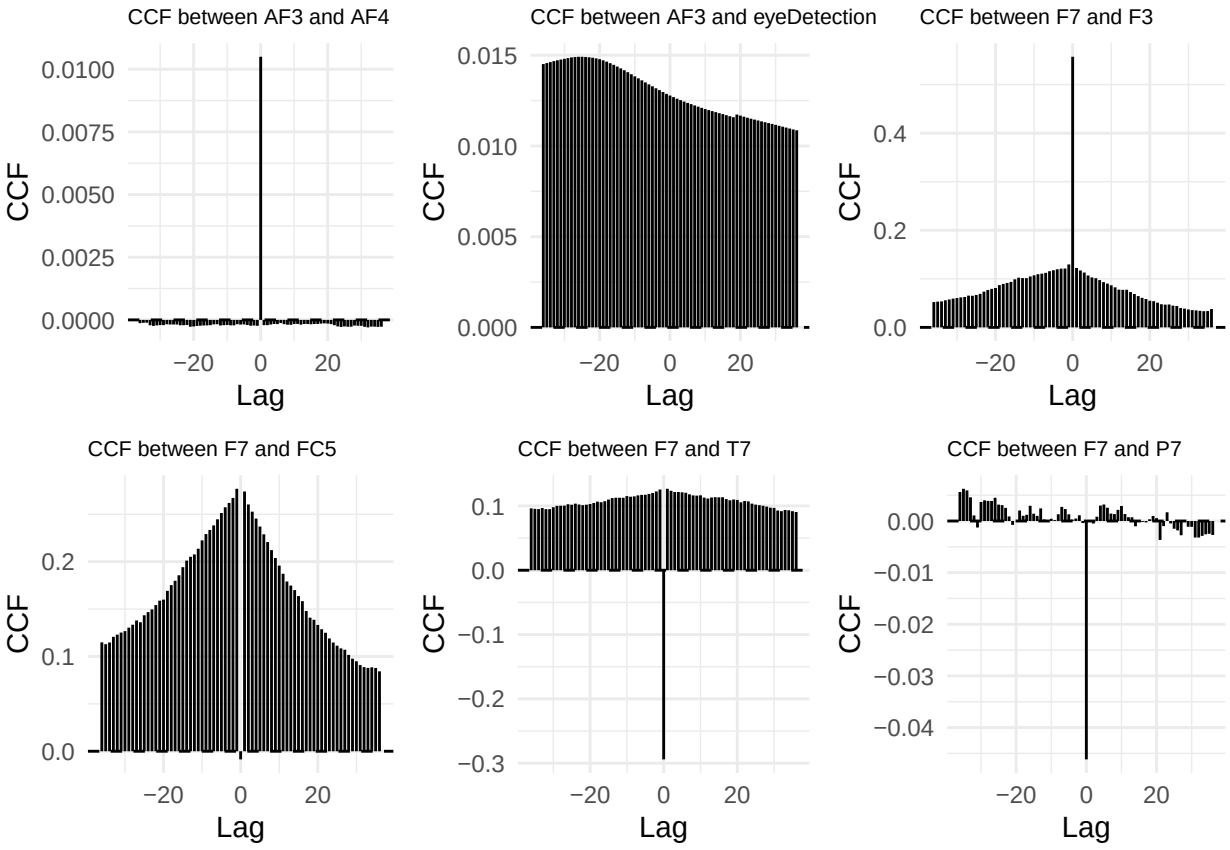
As far as cross-correlation goes (see plots below), cross-correlation can be identified by seeing significant peaks at certain lags in CCF plots. The following are just a few examples, but are definitely not all examples identified from this data. The CCF plot between AF3 and eyeDetection shows significant peaks across a range of lags, which indicates a strong cross-correlation between the eyeDetection signal and electrode AF3. Another example can be seen in the CCF plot between T7 and FC6, which shows significant peaks across a range of lags, further indicating significant cross-correlation between those two variables. A third example is the CCF plot between F7 and FC5, which, again, shows significant spikes across many lags and further indicates cross-correlation. A fourth example that shows significant spikes across lags is the CCF plot between O1 and O2, indicating signs of cross-correlation.

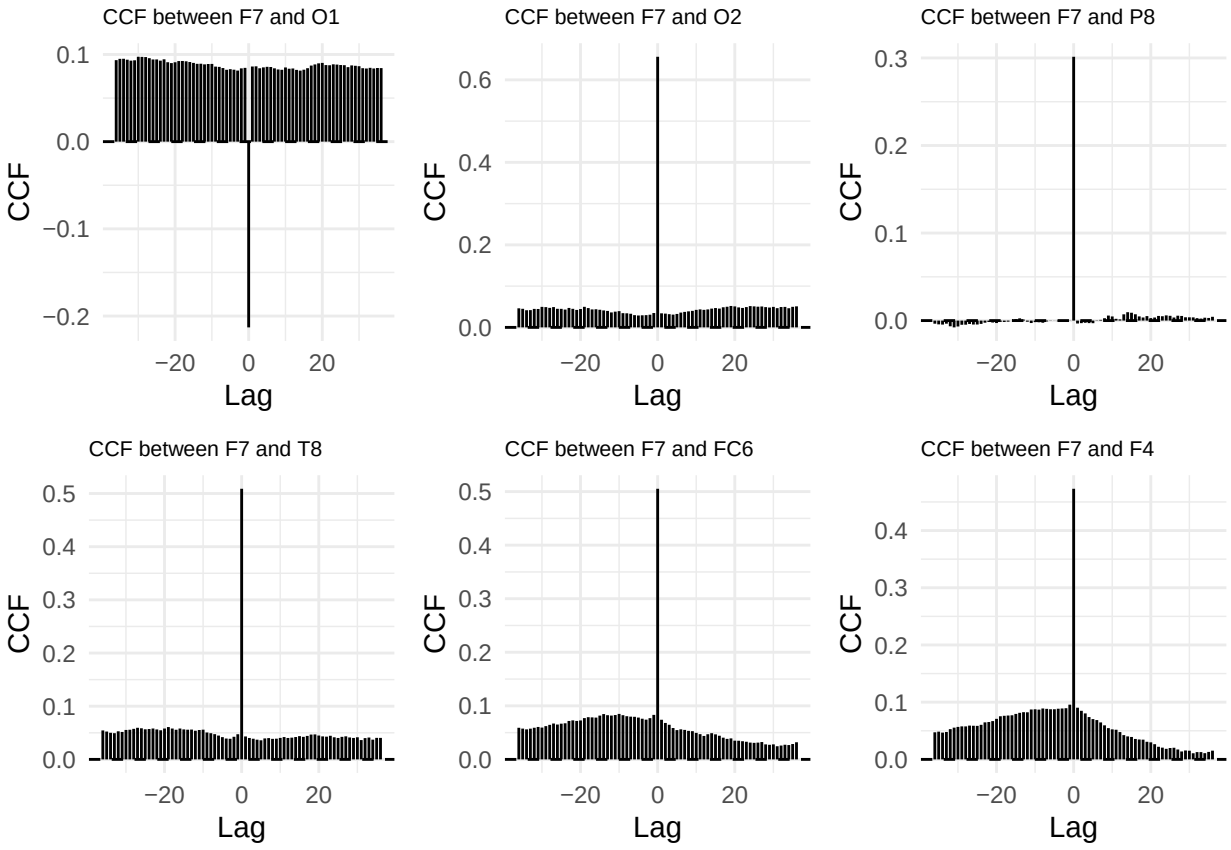
```
library(forecast)
library(ggplot2)
library(dplyr)
library(gridExtra)

# Function to plot cross-correlation
plot_ccf <- function(series1, series2, series1_name, series2_name) {
  ccf_result <- ccf(series1, series2, plot = FALSE)
  ccf_data <- data.frame(
```

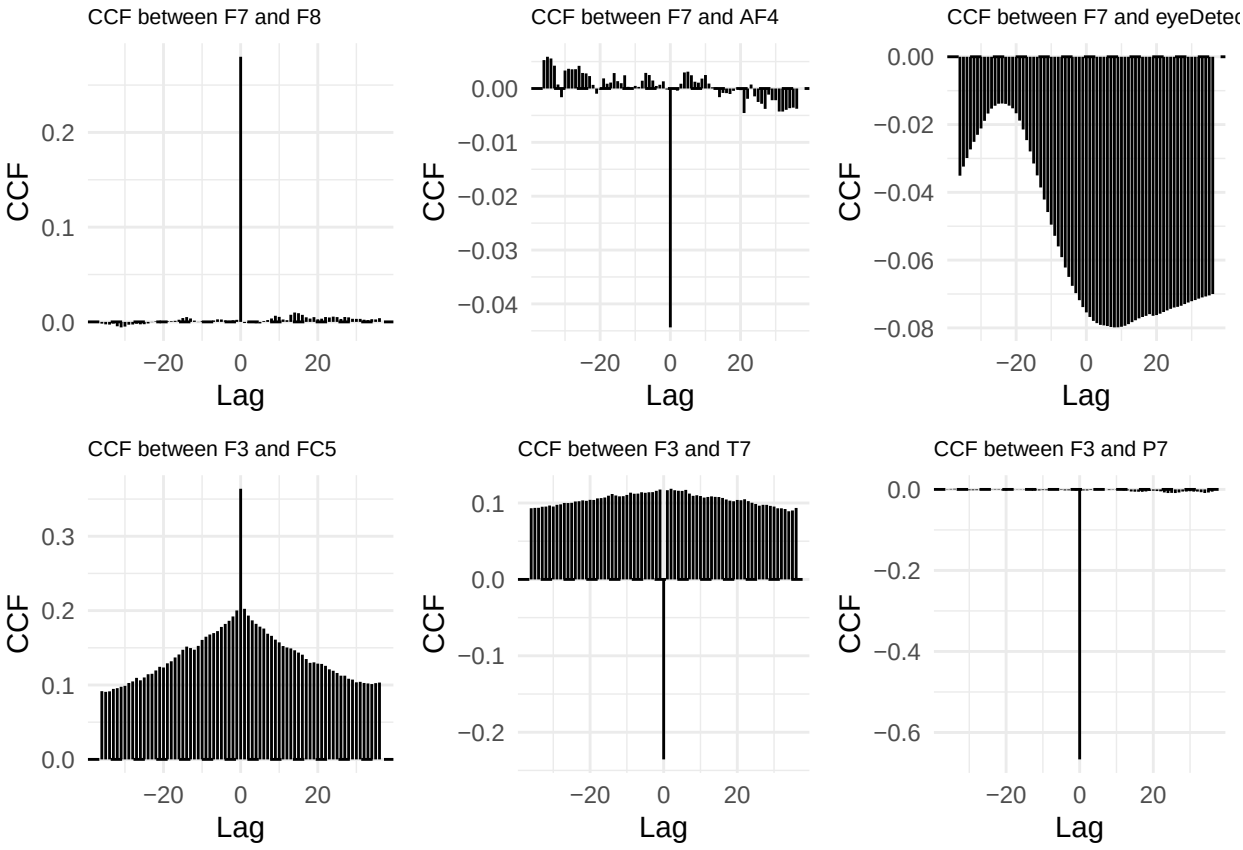
```
    lag = ccf_result$lag,
    acf = ccf_result$acf
  )
  ggplot(ccf_data, aes(x = lag, y = acf)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    geom_segment(aes(xend = lag, yend = 0)) +
    labs(
      title = paste("CCF between", series1_name, "and", series2_name),
      x = "Lag",
      y = "CCF"
    ) +
    theme_minimal() +
    theme(plot.title = element_text(size = 8))  # Adjust title size here
}

# Select columns and remove 'ds'
eeg_data <- eeg_train %>% select(-ds)

# Create pairs of columns
colnames <- colnames(eeg_data)
pairs <- combn(colnames, 2, simplify = FALSE)

# Plot cross-correlation for each pair
plots <- lapply(pairs, function(pair) {
  series1_name <- pair[1]
  series2_name <- pair[2]
  series1 <- eeg_data[[series1_name]]
  series2 <- eeg_data[[series2_name]]
  plot_ccf(series1, series2, series1_name, series2_name)
})

# Print plots in pages of 6
plot_pages <- split(plots, ceiling(seq_along(plots)/6))

for (page in plot_pages) {
  do.call("grid.arrange", c(page, ncol = 3, nrow = 2))
}
```

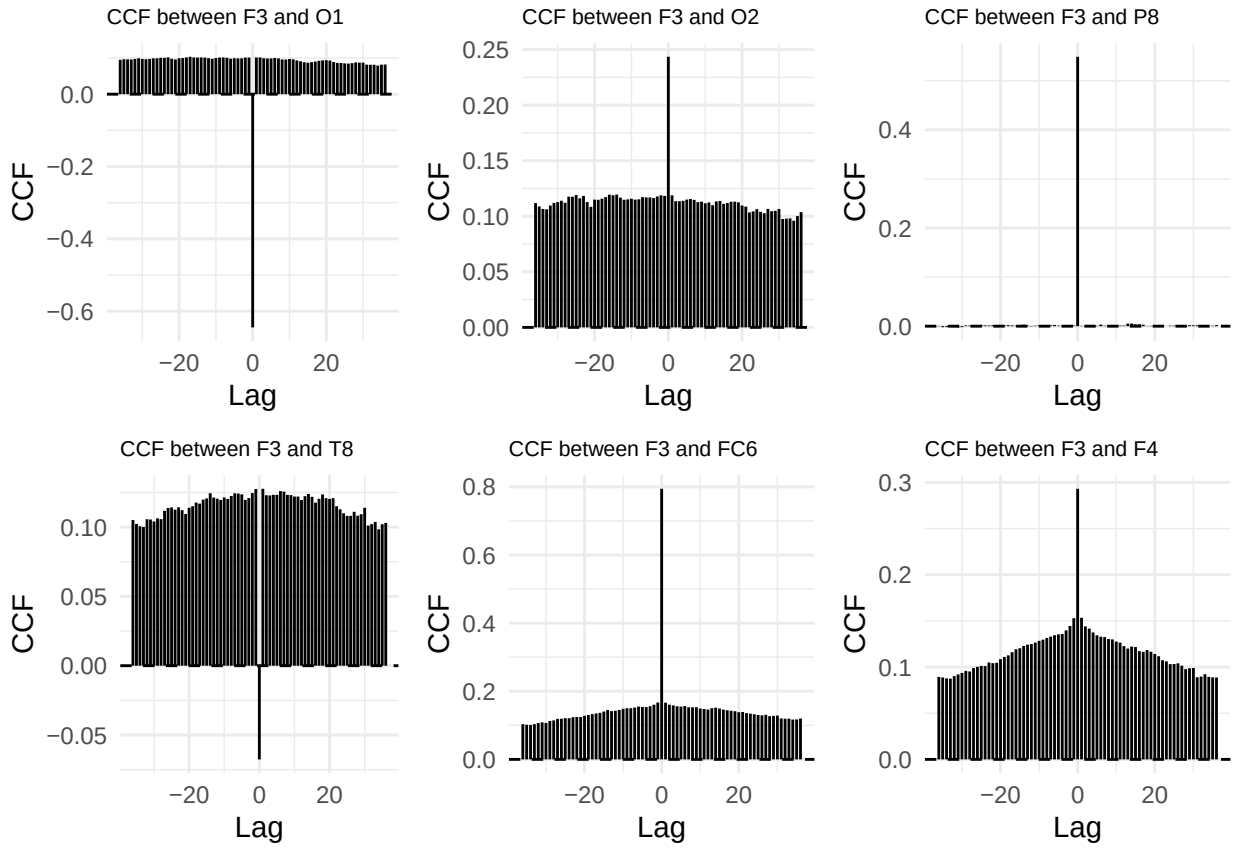

CCF between AF3 and F7
CCF between AF3 and F3
CCF between AF3 and FC5
CCF between AF3 and T7
CCF between AF3 and P7
CCF between AF3 and O1
CCF
Lag

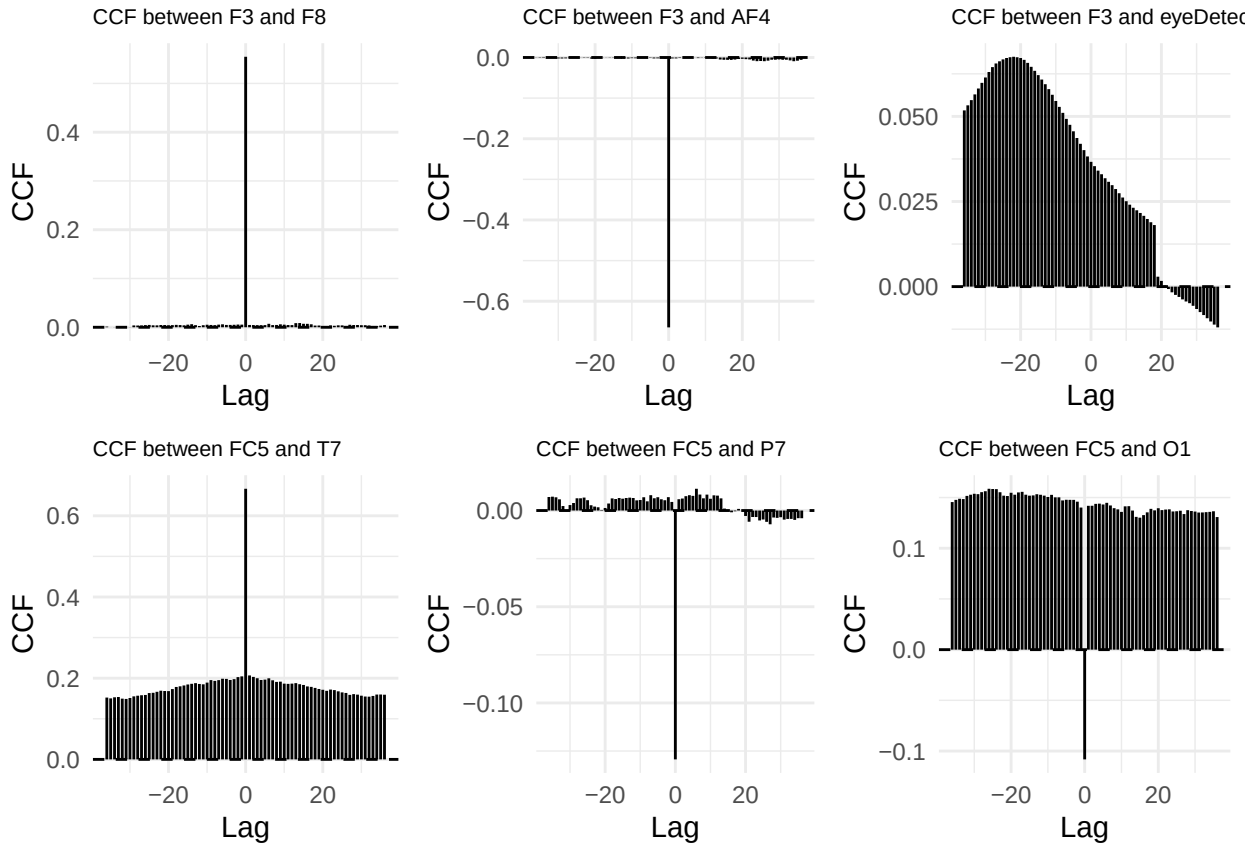

CCF between AF3 and O2
CCF between AF3 and P8
CCF between AF3 and T8
CCF between AF3 and FC6
CCF between AF3 and F4
CCF between AF3 and F8
CCF
Lag
0.04
0.03
0.02
0.01
0.00
1.00
0.75
0.50
0.25
−0.02
−0.04
−0.06
−0.08
0.6
0.4
0.2
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−20
0
20

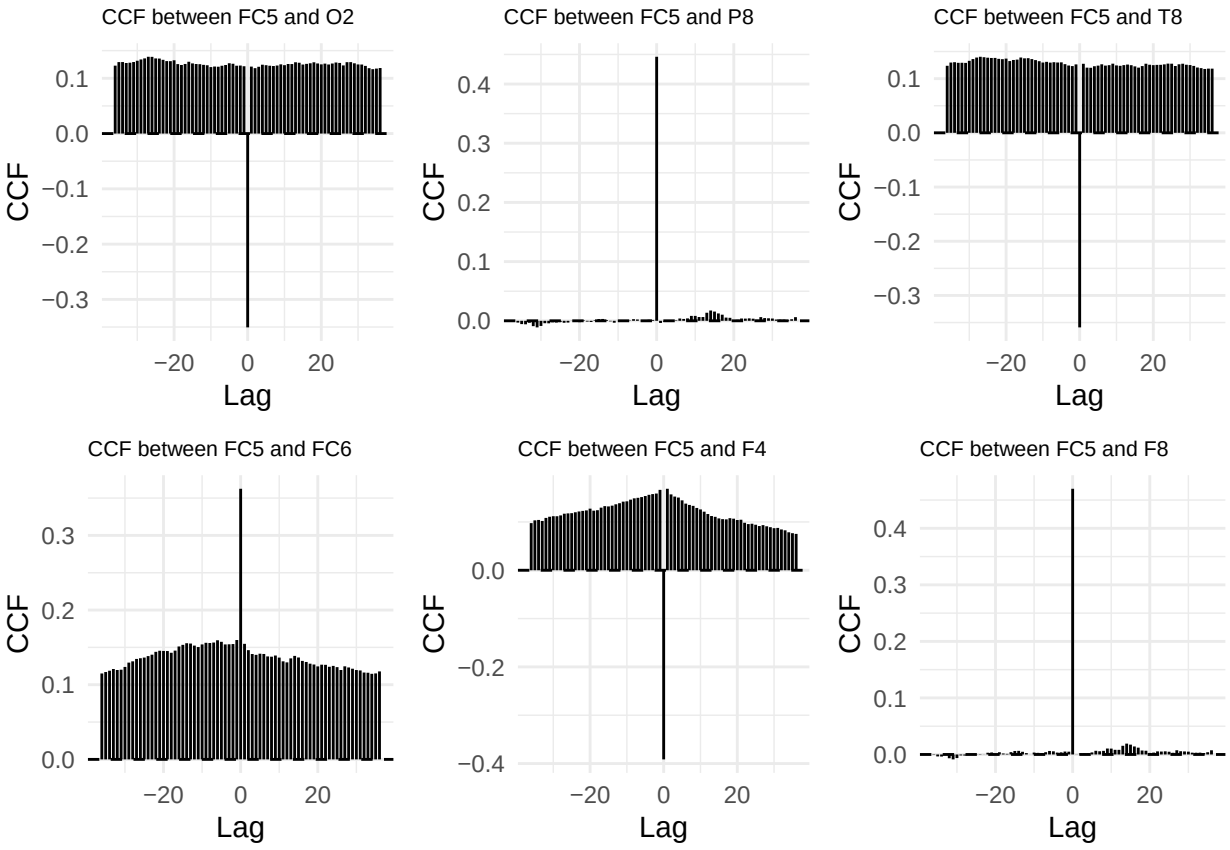

CCF between AF3 and AF4
CCF between AF3 and eyeDetection
CCF between F7 and F3
CCF between F7 and FC5
CCF between F7 and T7
CCF between F7 and P7
CCF
Lag

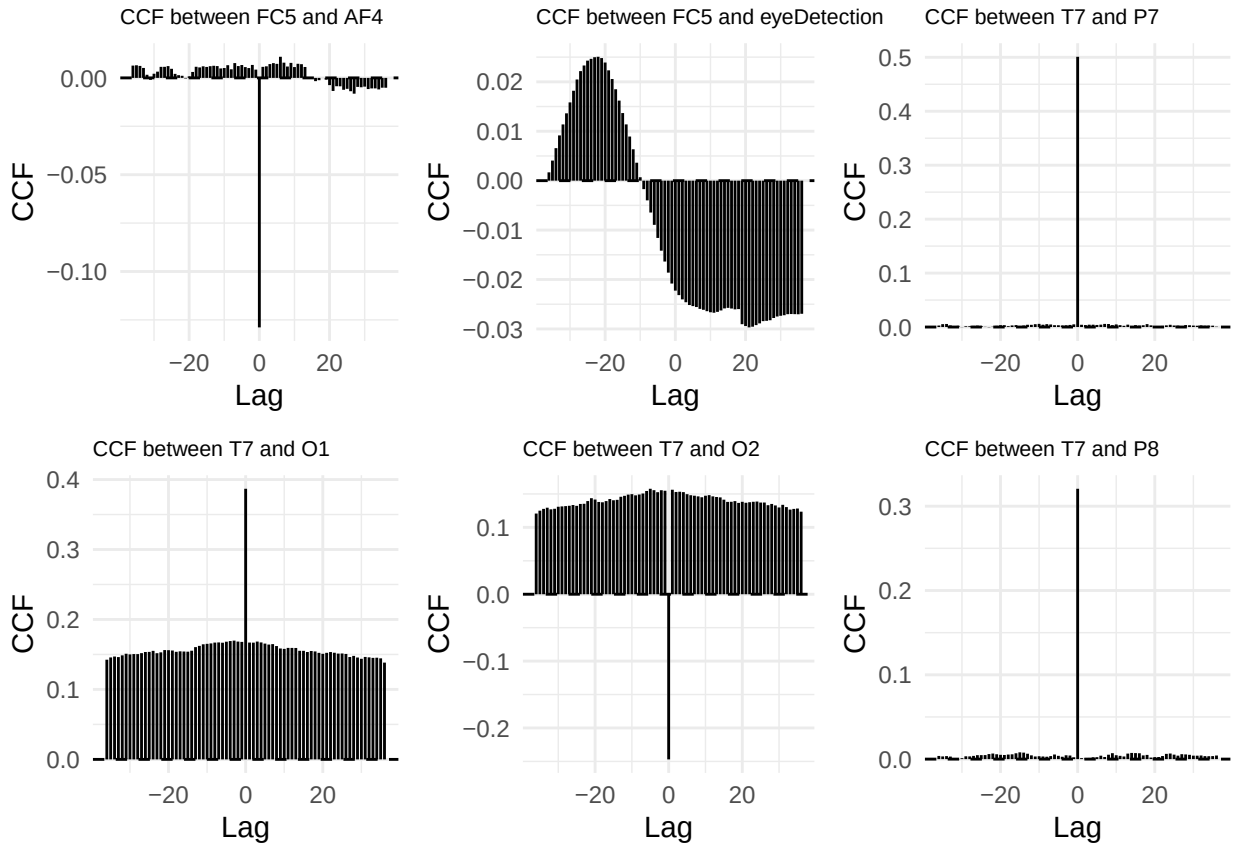

CCF between F7 and O1
CCF between F7 and O2
CCF between F7 and P8
CCF between F7 and T8
CCF between F7 and FC6
CCF between F7 and F4
CCF
Lag
0.1
0.0
−0.1
−0.2
0.6
0.4
0.2
0.3
0.5
−20
0
20

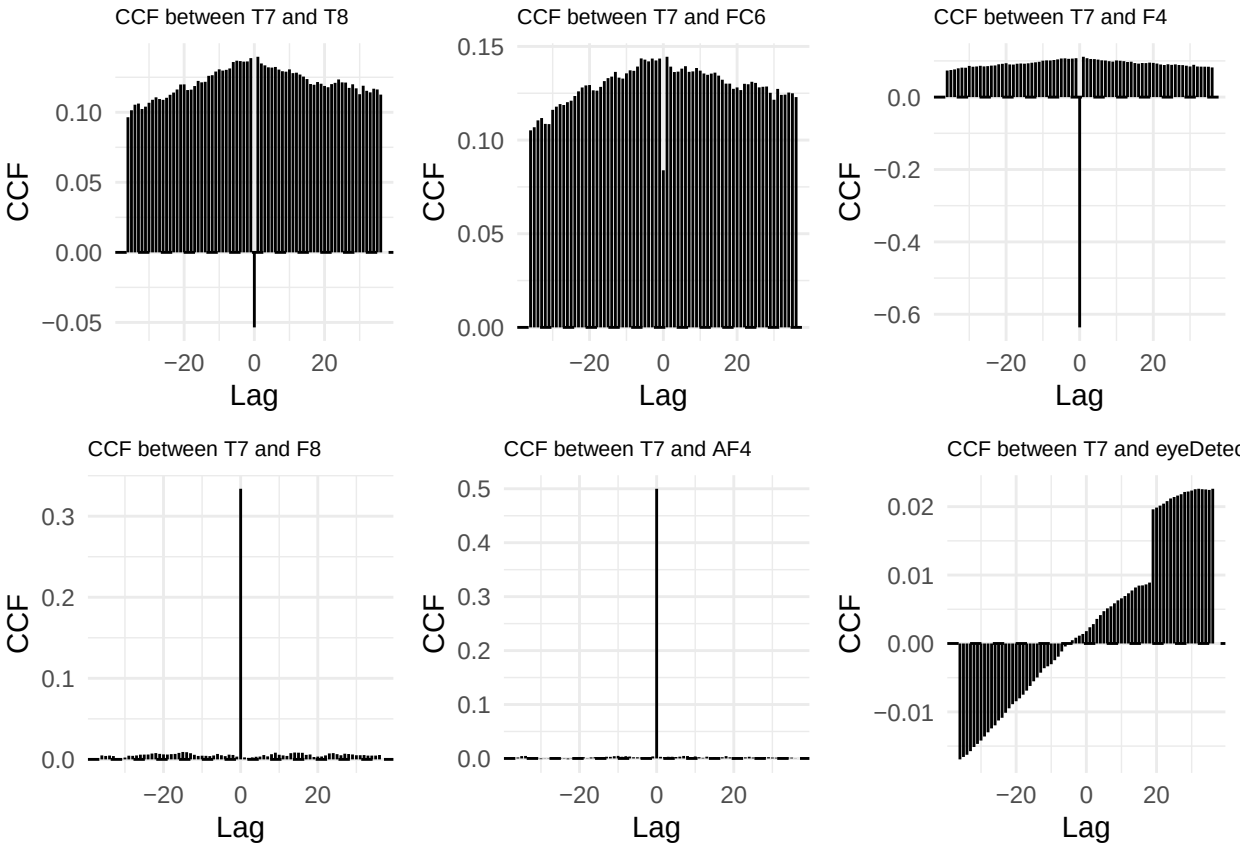

CCF between F7 and F8
CCF between F7 and AF4
CCF between F7 and eyeDetec
CCF between F3 and FC5
CCF between F3 and T7
CCF between F3 and P7
CCF
Lag

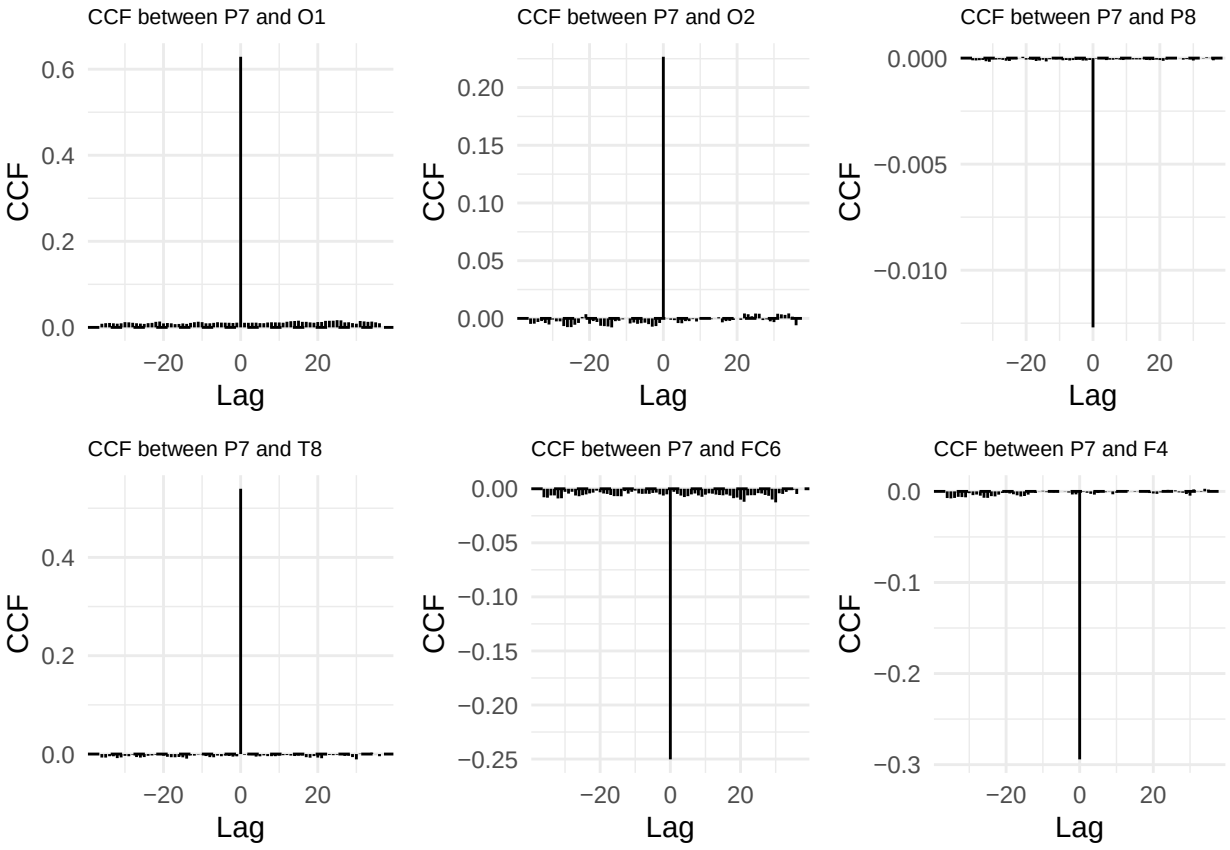

CCF between F3 and O1
CCF between F3 and O2
CCF between F3 and P8
CCF between F3 and T8
CCF between F3 and FC6
CCF between F3 and F4
CCF
Lag

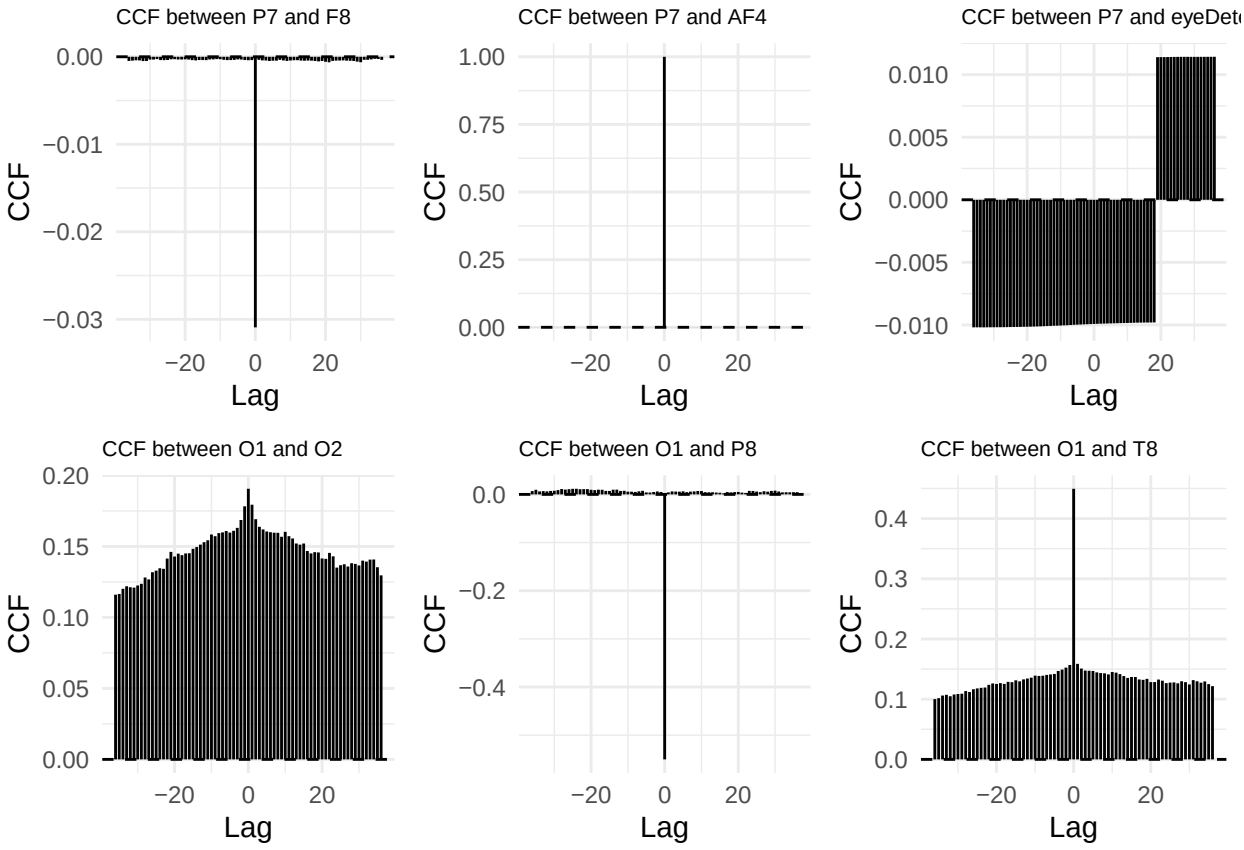

CCF between F3 and F8
CCF between F3 and AF4
CCF between F3 and eyeDetec
CCF between FC5 and T7
CCF between FC5 and P7
CCF between FC5 and O1
CCF
Lag
0.4
0.2
0.0
−20
0
20
−0.2
−0.4
−0.6
0.050
0.025
0.000
0.6
0.00
−0.05
−0.10
0.1
−0.1

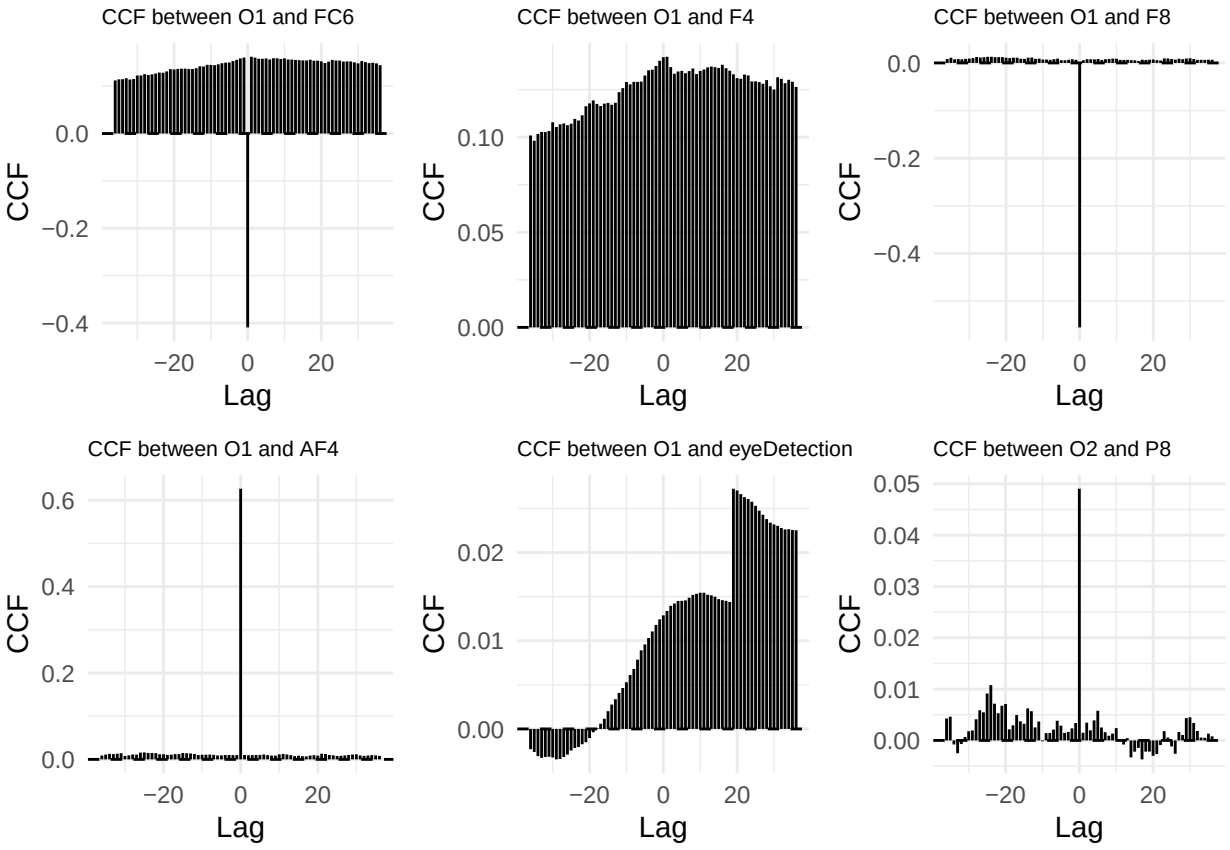

CCF between FC5 and O2
CCF between FC5 and P8
CCF between FC5 and T8
CCF between FC5 and FC6
CCF between FC5 and F4
CCF between FC5 and F8
CCF
Lag
0.1
0.0
−0.1
−0.2
−0.3
0.4
0.3
0.2
−0.4
−20
0
20

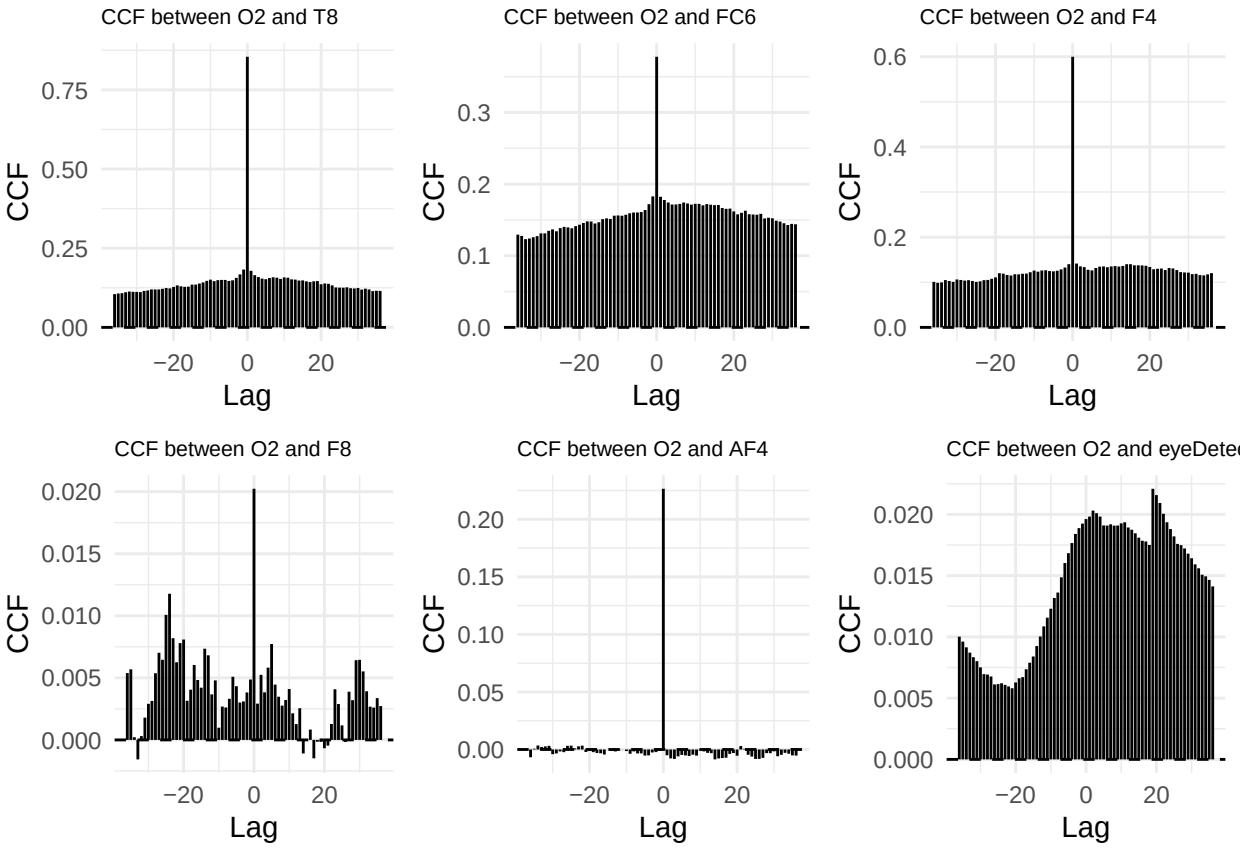

CCF between FC5 and AF4
CCF between FC5 and eyeDetection
CCF between T7 and P7
CCF between T7 and O1
CCF between T7 and O2
CCF between T7 and P8
CCF
Lag

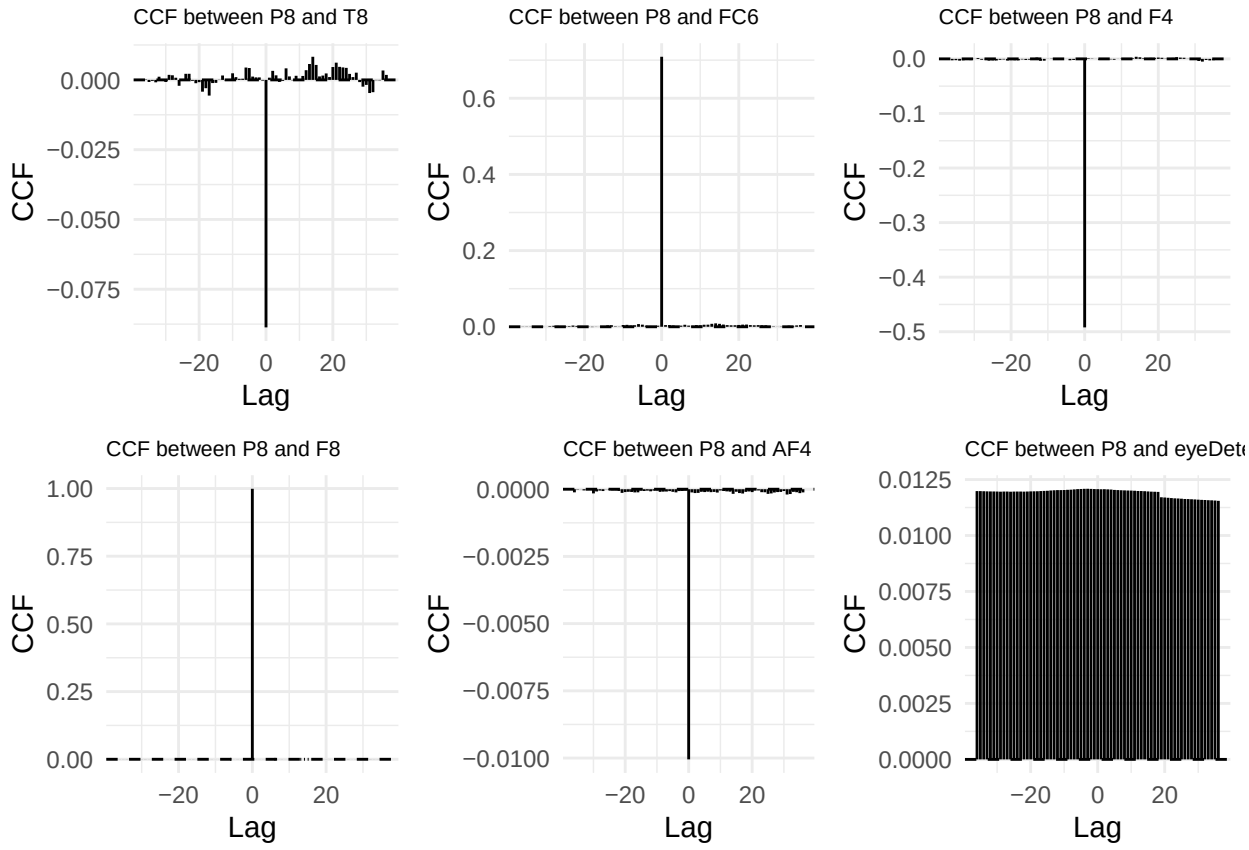

CCF between T7 and T8
CCF between T7 and FC6
CCF between T7 and F4
CCF between T7 and F8
CCF between T7 and AF4
CCF between T7 and eyeDetec
CCF
Lag

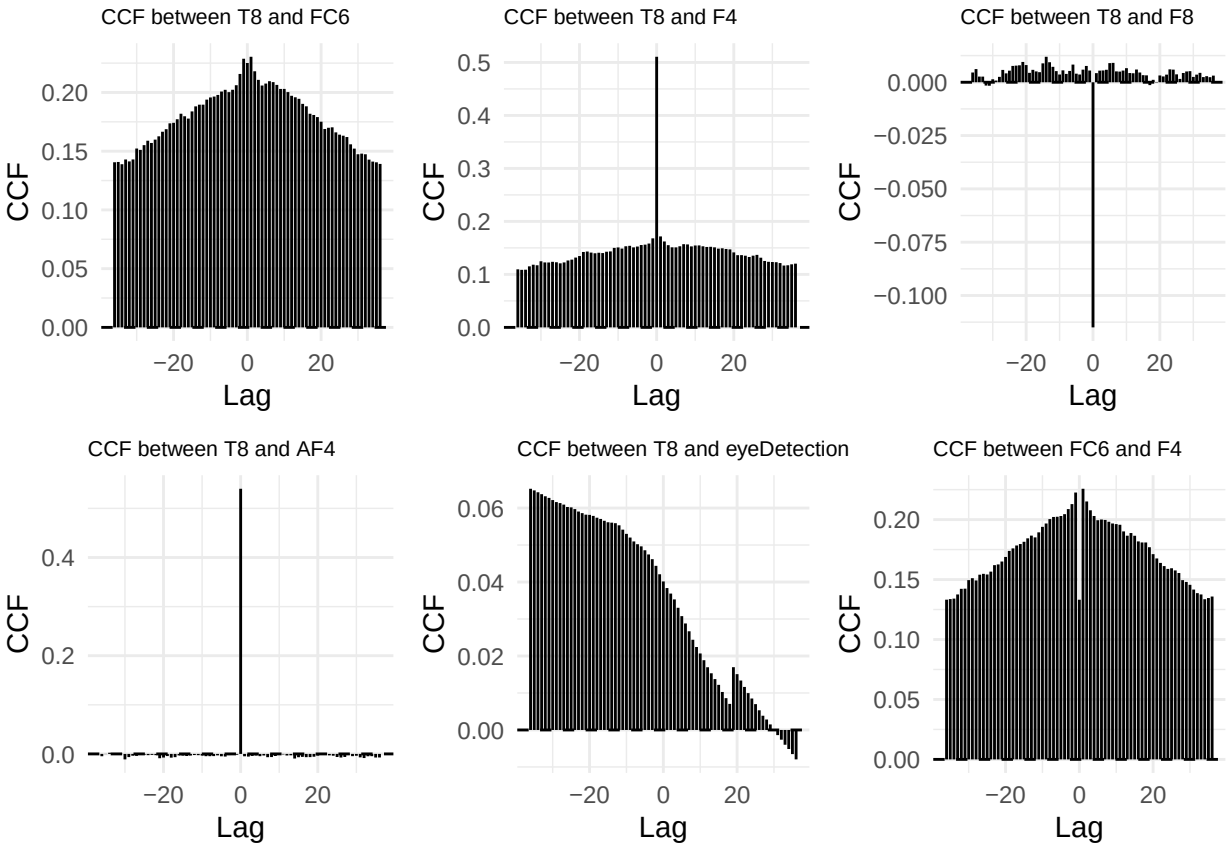

CCF between P7 and O1
CCF between P7 and O2
CCF between P7 and P8
CCF between P7 and T8
CCF between P7 and FC6
CCF between P7 and F4
CCF
Lag
0.6
0.4
0.2
0.0
0.20
0.15
0.10
0.05
0.00
0.000
−0.005
−0.010
−0.05
−0.10
−0.15
−0.20
−0.25
−0.1
−0.2
−0.3
−20
0
20

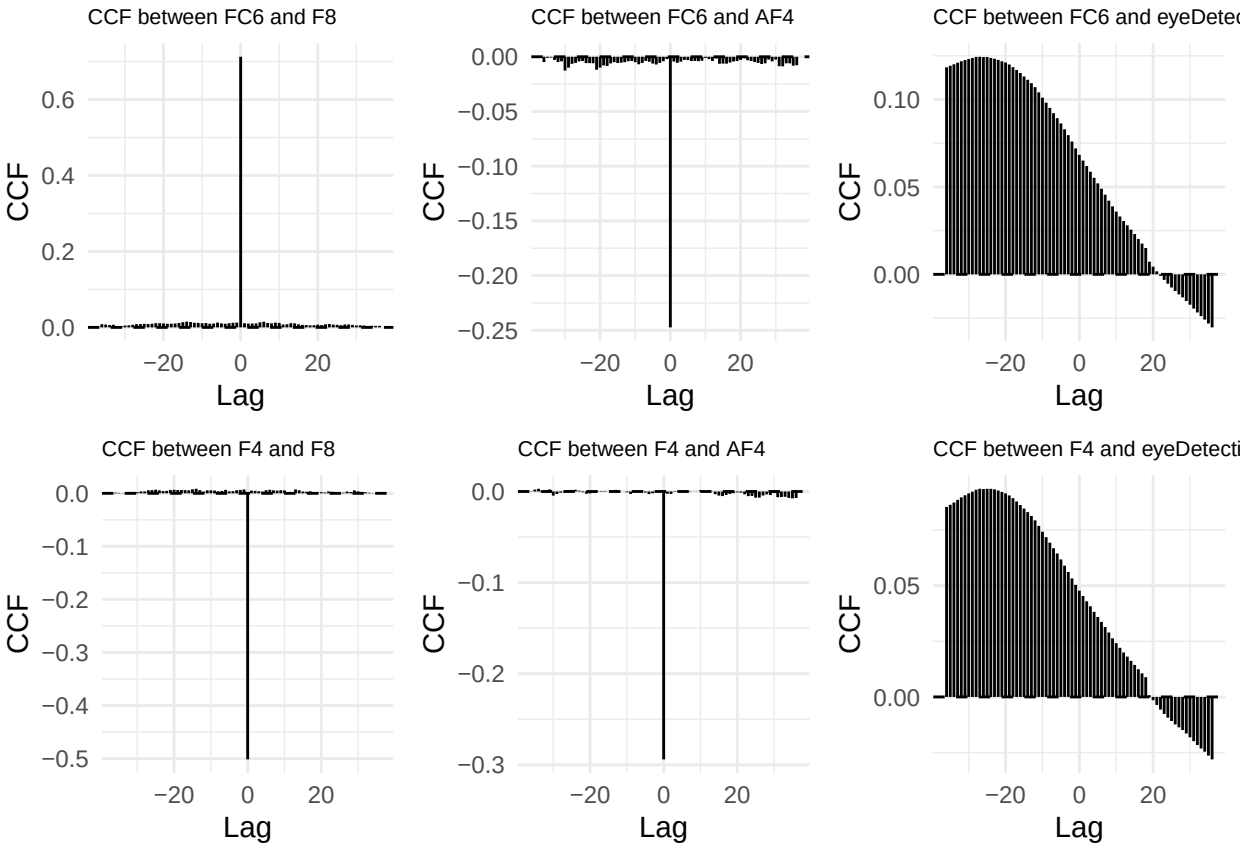

CCF between P7 and F8
CCF between P7 and AF4
CCF between P7 and eyeDet
CCF between O1 and O2
CCF between O1 and P8
CCF between O1 and T8
CCF
Lag

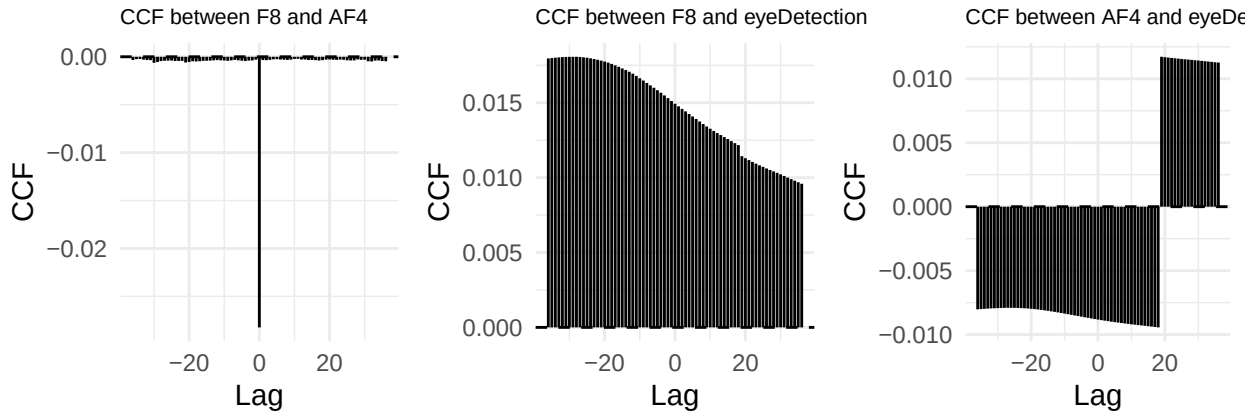

CCF between O1 and FC6
CCF between O1 and F4
CCF between O1 and F8
CCF between O1 and AF4
CCF between O1 and eyeDetection
CCF between O2 and P8
CCF
Lag
0.0
−0.2
−0.4
0.10
0.05
0.00
0.6
0.4
0.2
0.02
0.01
0.04
0.03
−20
0
20


CCF between O2 and T8
CCF between O2 and FC6
CCF between O2 and F4
CCF between O2 and F8
CCF between O2 and AF4
CCF between O2 and eyeDete
CCF
Lag
0.75
0.50
0.25
0.00
0.3
0.2
0.1
0.0
0.6
0.4
0.020
0.015
0.010
0.005
0.000
0.20
0.15
0.10
0.05
−20
0
20


CCF between P8 and T8
CCF between P8 and FC6
CCF between P8 and F4
CCF between P8 and F8
CCF between P8 and AF4
CCF between P8 and eyeDet
CCF
Lag
0.000
−0.025
−0.050
−0.075
0.6
0.4
0.2
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
1.00
0.75
0.50
0.25
0.00
0.0000
−0.0025
−0.0050
−0.0075
−0.0100
0.0125
0.0100
0.0075
0.0050
0.0025
−20
0
20

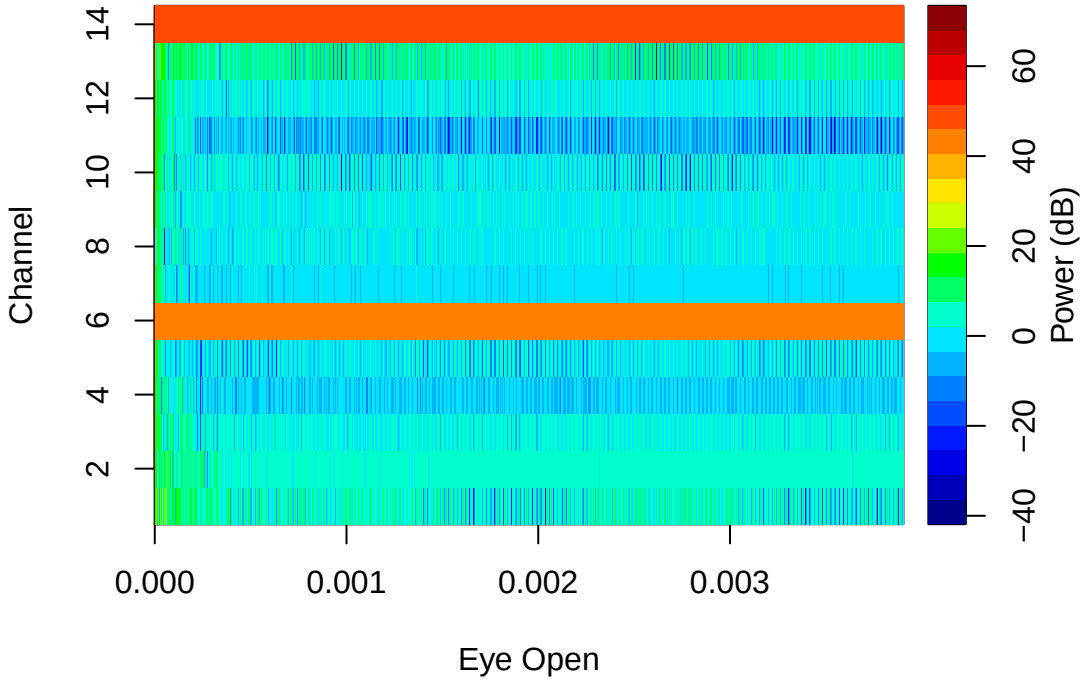

CCF between T8 and FC6
CCF between T8 and F4
CCF between T8 and F8
CCF between T8 and AF4
CCF between T8 and eyeDetection
CCF between FC6 and F4
CCF
Lag


CCF between FC6 and F8
CCF between FC6 and AF4
CCF between FC6 and eyeDetec
CCF between F4 and F8
CCF between F4 and AF4
CCF between F4 and eyeDetecti
CCF
Lag
0.6
0.4
0.2
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
0.10
0.05
−20
0
20



Frequency-Space We can also explore the data in frequency space by using a Fast Fourier Transform. After the FFT we can summarise the distributions of frequencies by their density across the power spectrum. This will let us see if there any obvious patterns related to eye status in the overall frequency distributions.

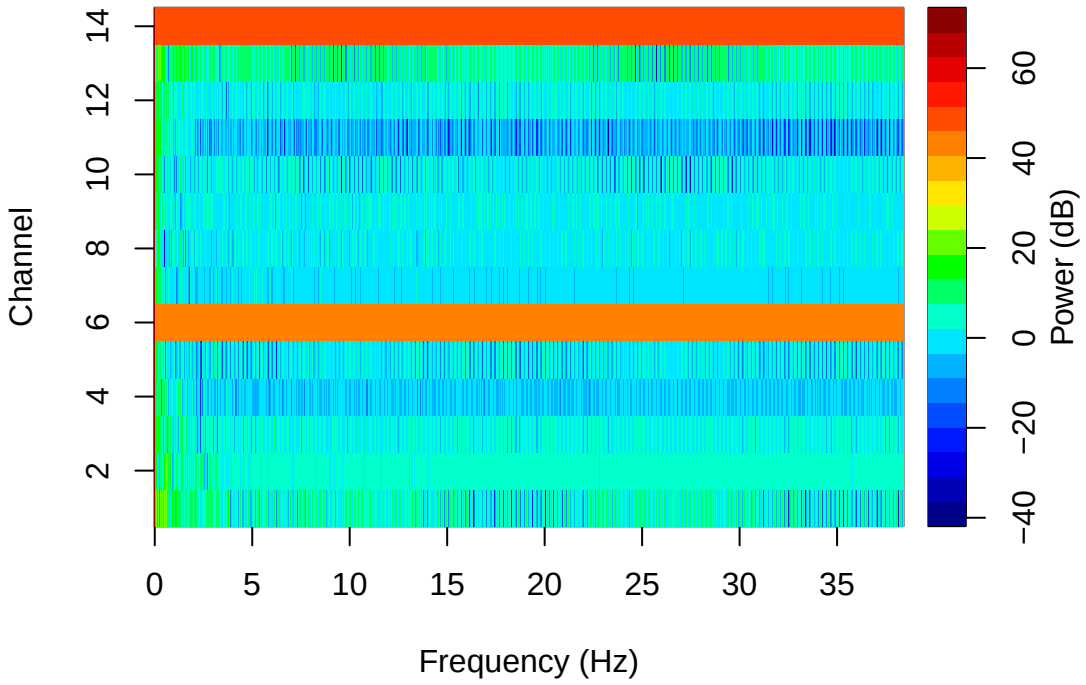
```
eegkit::eegpsd(eeg_train %>% dplyr::filter(eyeDetection == 0) %>% dplyr::select(-eyeDetection, -ds), Fs
```



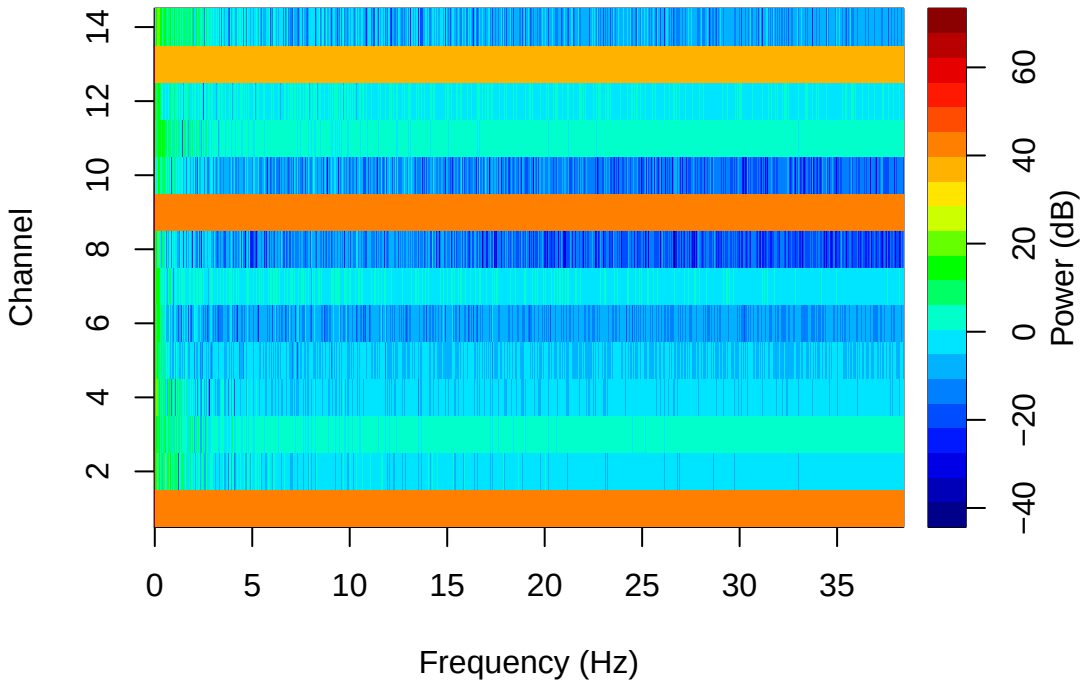
Note: Just wondering if the coding may be slightly wrong for this? Isn't the x-axis supposed to potentially demonstrate frequency in something like Hertz? Maybe something like below?

```
# Load required libraries
library(eegkit)
library(dplyr)
library(ggplot2)
library(gridExtra)

# Assuming eeg_train is already loaded
# Calculate sampling rate Fs (samples per second)
num_samples <- nrow(eeg_train)
total_duration <- 117  # seconds
Fs <- num_samples / total_duration

# Function to plot EEG PSD with ggplot2
plot_eeg_psd <- function(eeg_data, eye_status, Fs) {
  # Calculate the power spectral density
  psd_result <- eegkit::eegpsd(eeg_data, Fs = Fs)

  # Create a data frame from the psd_result
  psd_data <- data.frame(
    Frequency = psd_result$freq,
    Power = psd_result$spec
  )
```

```r
  # Generate the plot using ggplot2
  p <- ggplot(psd_data, aes(x = Frequency, y = Power)) +
    geom_line() +
    labs(
      x = "Frequency (Hz)",
      y = "Power"
    ) +
    ggtitle(ifelse(eye_status == 0, "Eyes Open", "Eyes Closed")) +  # Title based on eye_status
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5, size = 10))  # Center and resize the title

  return(p)
}

# Filter data for eyes open and closed
eeg_open <- eeg_train %>% filter(eyeDetection == 0) %>% select(-eyeDetection, -ds)
eeg_closed <- eeg_train %>% filter(eyeDetection == 1) %>% select(-eyeDetection, -ds)

# Plot EEG Power Spectral Density (PSD) for eyes open and closed
psd_open <- plot_eeg_psd(eeg_open, 0, Fs)
```

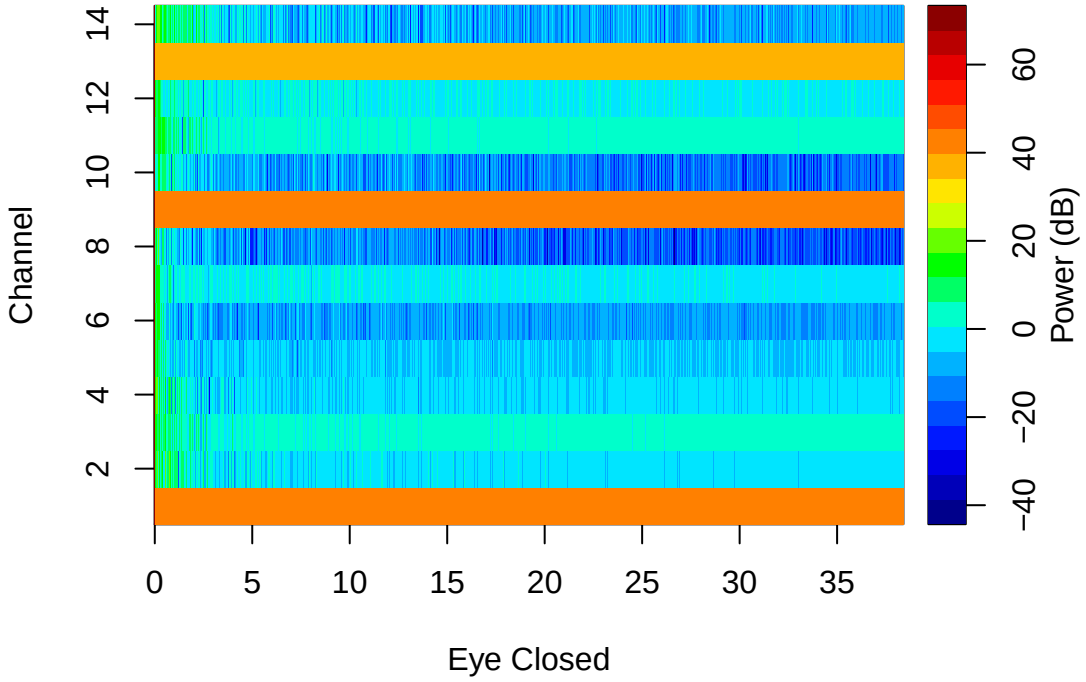


```
psd_closed <- plot_eeg_psd(eeg_closed, 1, Fs)
```




```
# Add titles directly in each plot
psd_open <- psd_open +
  labs(title = "Eyes Open")

psd_closed <- psd_closed +
  labs(title = "Eyes Closed")
```

```
eegkit::eegpsd(eeg_train %>% dplyr::filter(eyeDetection == 1) %>% dplyr::select(-eyeDetection, -ds), Fs
```



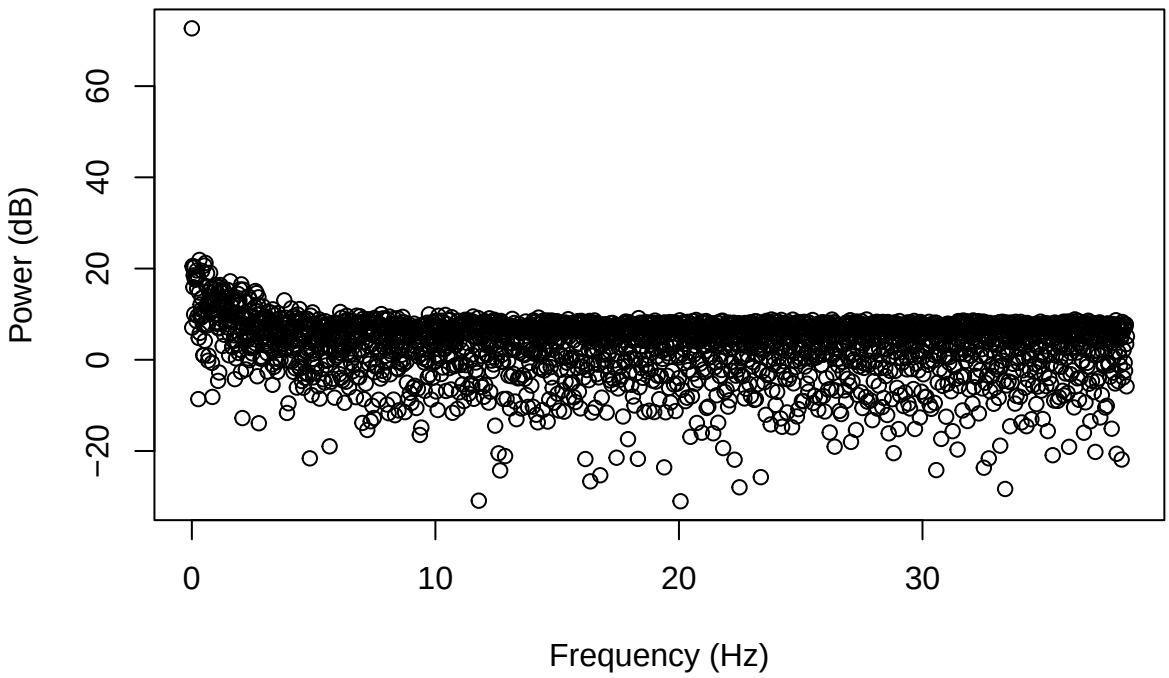
8 Do you see any differences between the power spectral densities for the two eye states? If so, describe them.

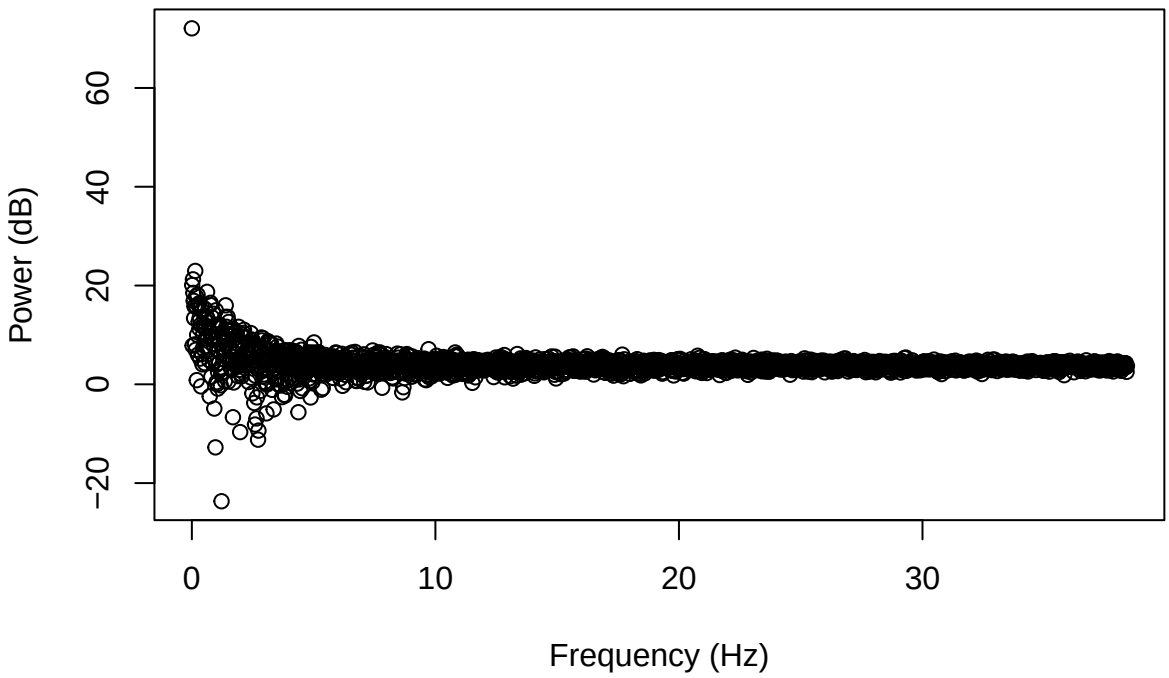
For the eyes open state, channels 14 and 6 show the highest power spectral densities (between 50-60 dB), while in the eyes closed state, channels 13, 9, and 1 show the highest power spectral densities (between 40-50 dB). All other channels for both eye states seem to show low to medium power spectral densities (-40 to 20dB). When we look at these channels in separate plots, we see some patterns arise.

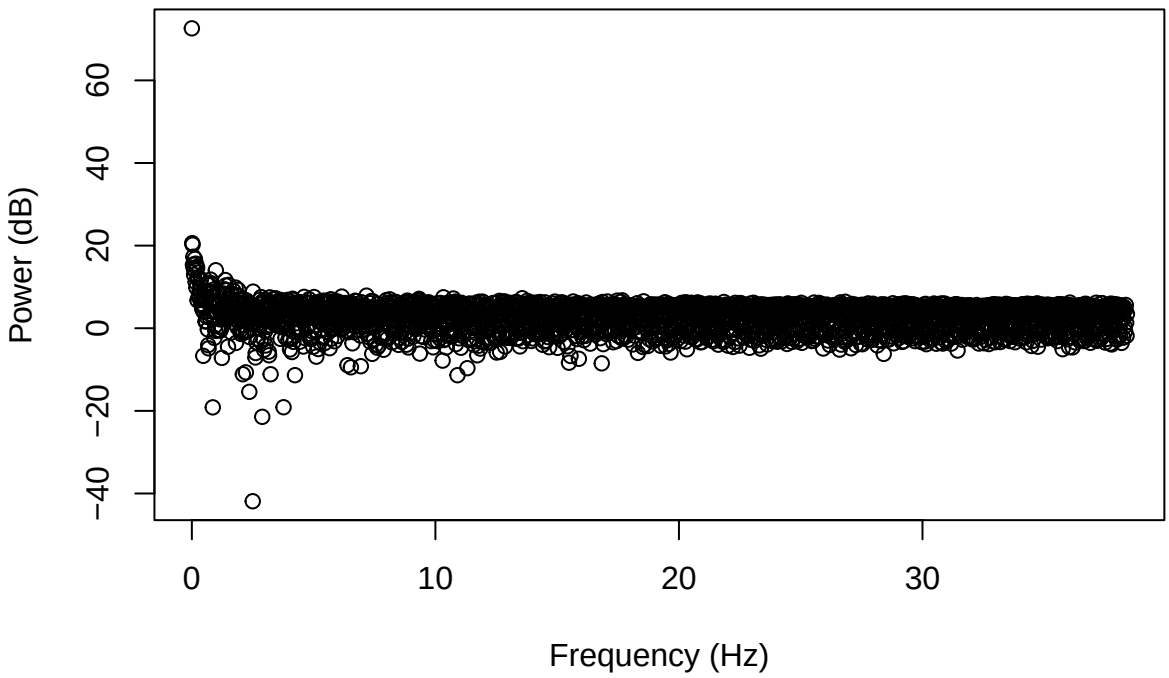
When we look at the individual plots of each channel/electrode (see below, found in the same order of electrodes (top to bottom) pictured in the legend in the very first plot of this practical), we see that, when the eyes are opened, the power spectral densities seem much more evenly distributed across frequencies and have less pronounced peaks in the lower frequency ranges. In addition, overall, it seems like the power density levels across the majority of the electrodes seem lower than they are, overall, compared to when the eyes are closed. Furthermore, electrodes, such as P7 and AF4, show extremely low power density levels across all of the frequencies and exhibit a flat line throughout the frequencies. This potentially indicates a significant reduction in EEG activity, especially as compared to when the eyes are closed.

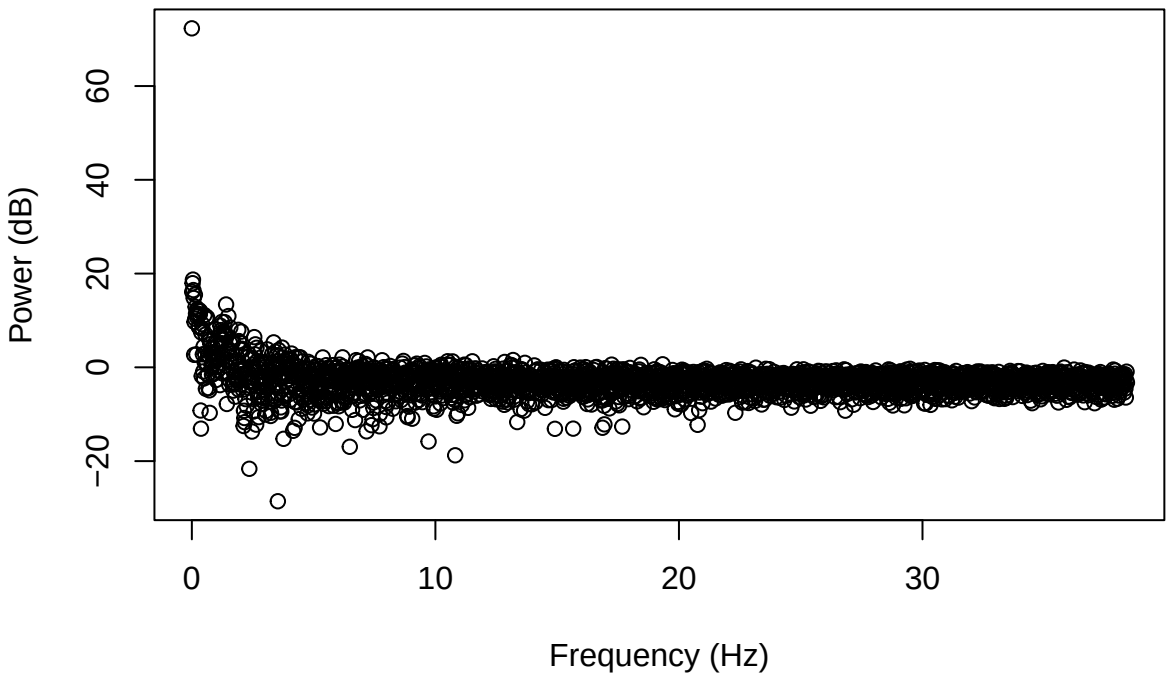
When we look at the individual plots of each channel/electrode (see below, found in the same order of electrodes (top to bottom) pictured in the legend in the very first plot of this practical), we see that, when the eyes are closed, some electrodes, such as AF3, F7, P8, and F8, have very low power levels or do not really show many prominent peaks. On the other hand, many electrodes (e.g., AF4, T7, P7, T8, O1, and O2) show a prominent peak in the power spectral density at lower frequencies. For those electrodes that show a prominent peak at lower frequencies then show a power decrease that gradually lowers as frequency increases after about 10 Hertz. Therefore, there does seem to be a difference between power spectral densities between the two eye states.

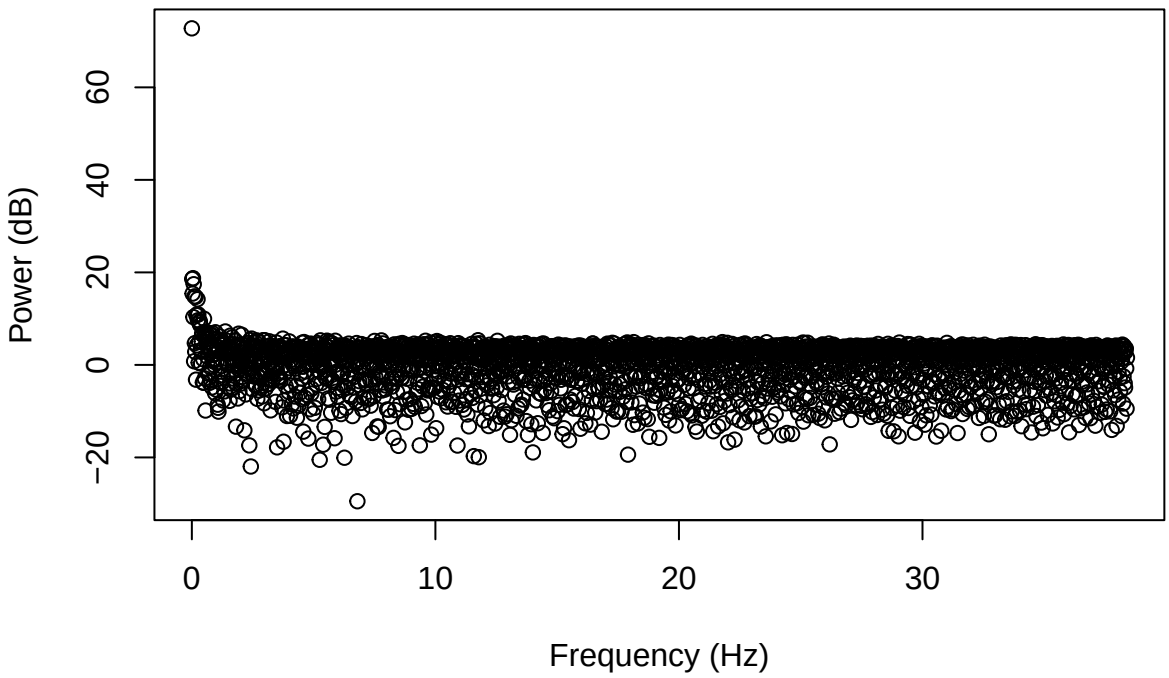
```r
##Code for Eyes Opened looking at each channel individually on a plot
# Load required libraries
library(eegkit)
library(dplyr)
library(ggplot2)
library(gridExtra)

# Assuming eeg_train is already loaded
# Calculate sampling rate Fs (samples per second)
num_samples <- nrow(eeg_train)
total_duration <- 117  # seconds
Fs <- num_samples / total_duration

# Function to plot EEG PSD with ggplot2 for each channel
plot_eeg_psd_individual <- function(eeg_data, Fs) {
  num_channels <- ncol(eeg_data)
  plots <- list()

  for (ch in 1:num_channels) {
    # Extract EEG data for the current channel
    eeg_channel <- eeg_data[, ch]

    # Calculate the power spectral density
    psd_result <- eegkit::eegpsd(eeg_channel, Fs = Fs)

    # Create a data frame from the psd_result
    psd_data <- data.frame(
      Frequency = psd_result$freq,
      Power = psd_result$spec
    )

    # Generate the plot using ggplot2
    p <- ggplot(psd_data, aes(x = Frequency, y = Power)) +
      geom_line() +
      labs(
        x = "Frequency (Hz)",
        y = "Power"
      ) +
      ggtitle(paste("Eyes Open - Channel", ch)) +  # Title for eyes open and channel
      theme_minimal() +
      theme(plot.title = element_text(hjust = 0.5, size = 10))  # Center and resize the title

    plots[[ch]] <- p
  }

  return(plots)
}

# Filter data for eyes open
eeg_open <- eeg_train %>% filter(eyeDetection == 0) %>% select(-eyeDetection, -ds)

# Plot EEG Power Spectral Density (PSD) for each channel separately for eyes open
psd_open_plots <- plot_eeg_psd_individual(eeg_open, Fs)
```

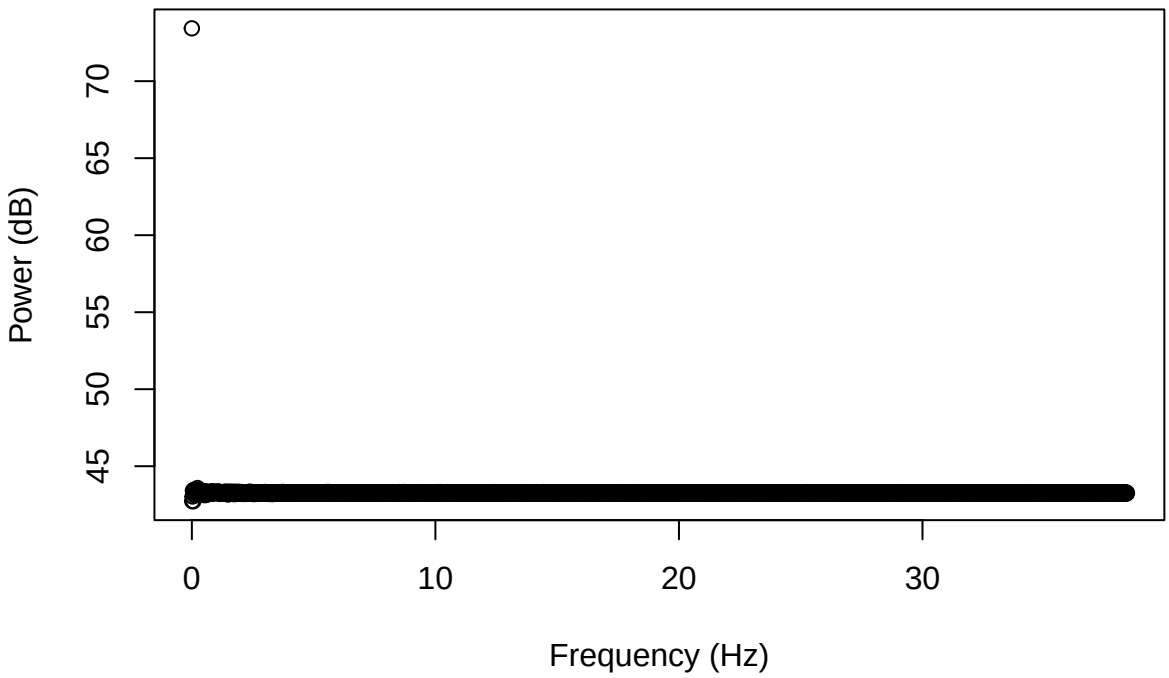

Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)

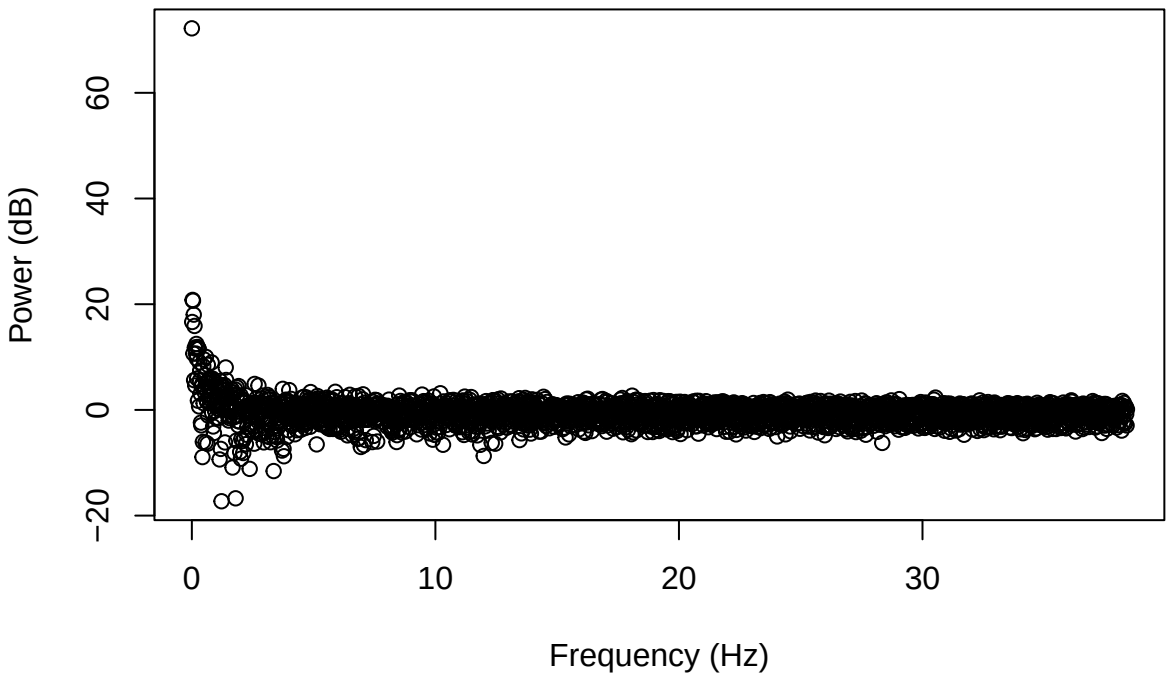

Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)

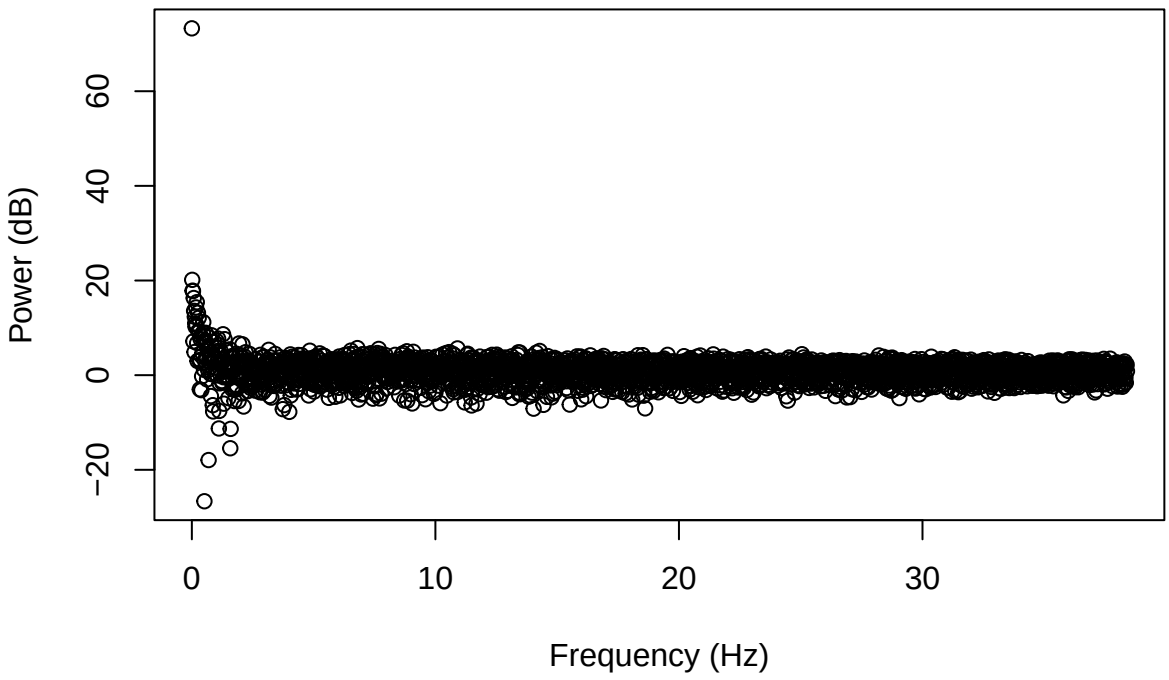

Power (dB)
60
40
20
0
−20
−40
0
10
20
30
Frequency (Hz)

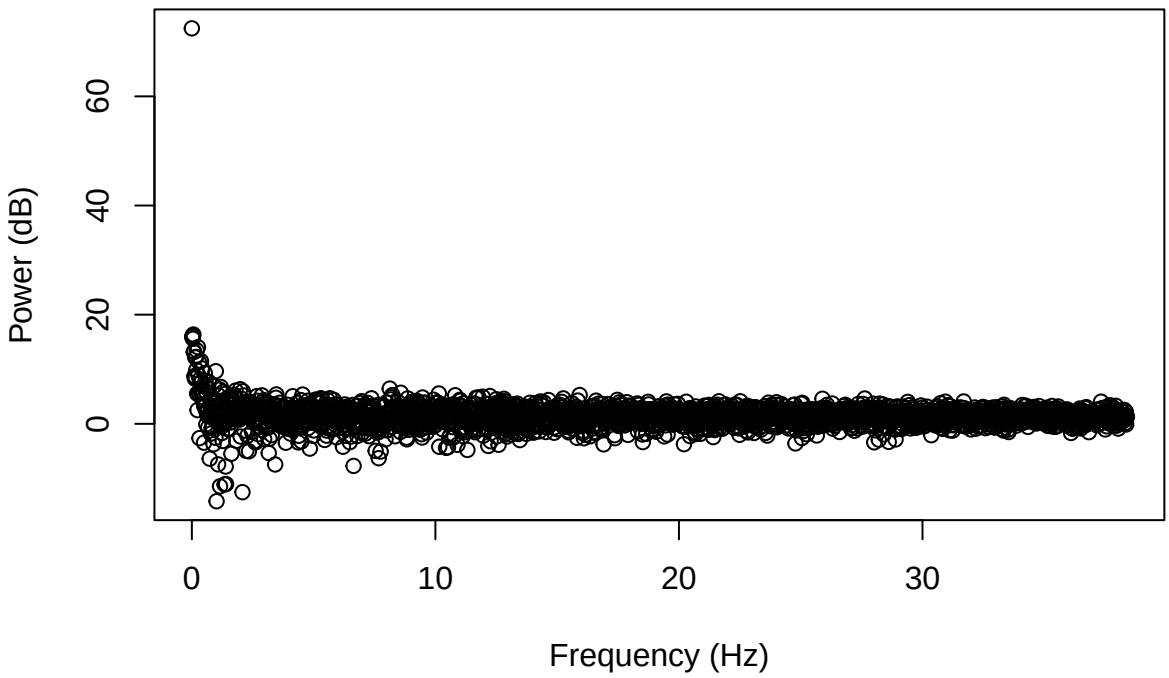

Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)

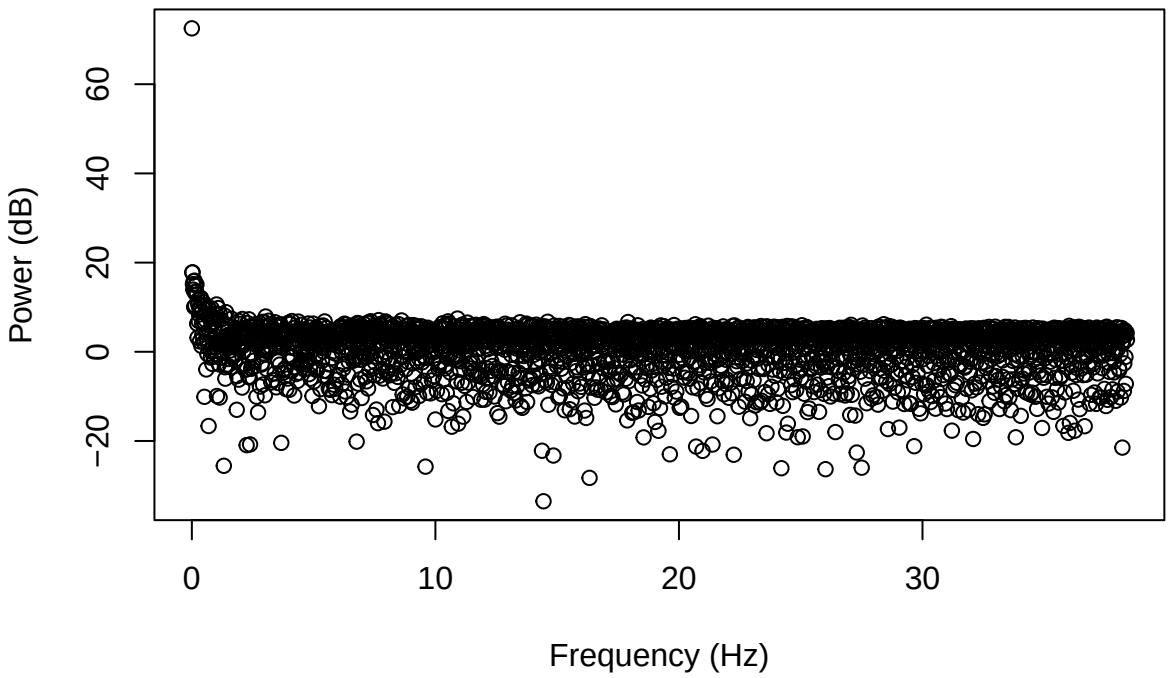

Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)

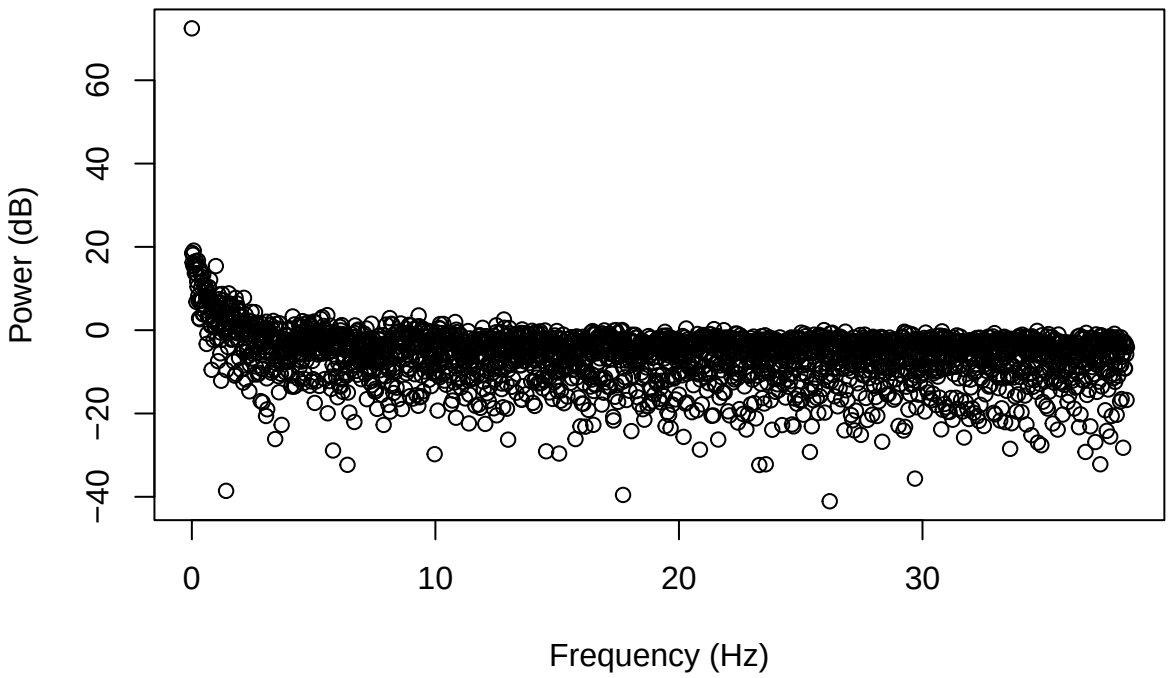

Power (dB)
45
50
55
60
65
70
0
10
20
30
Frequency (Hz)

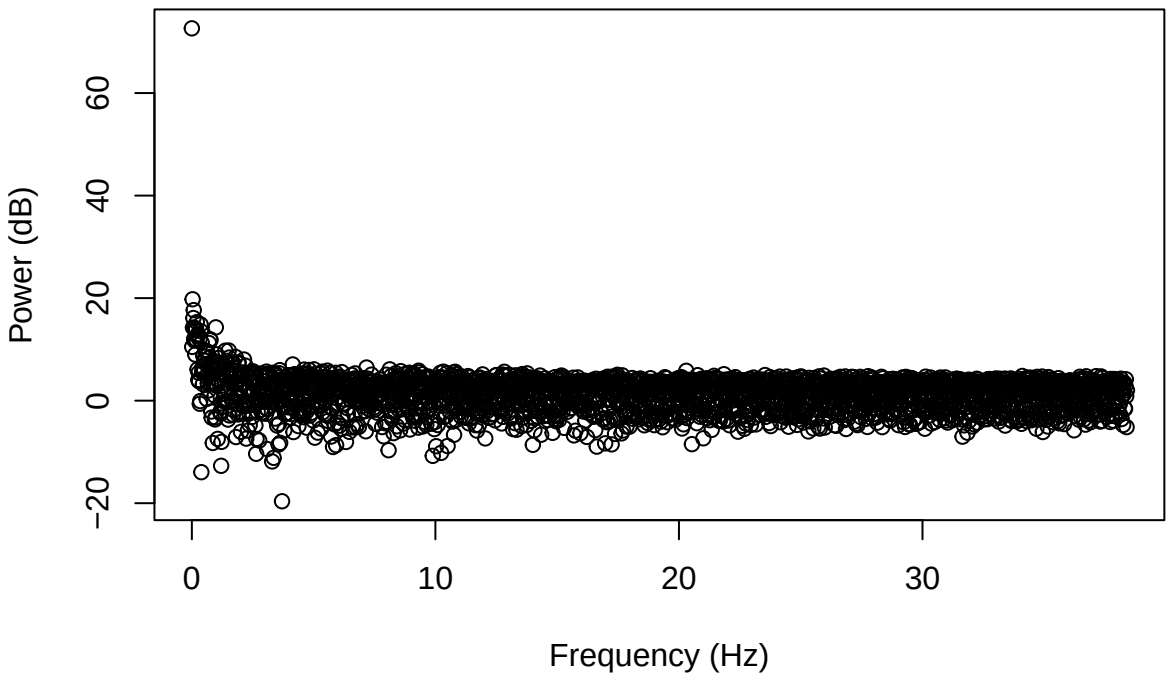

Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)

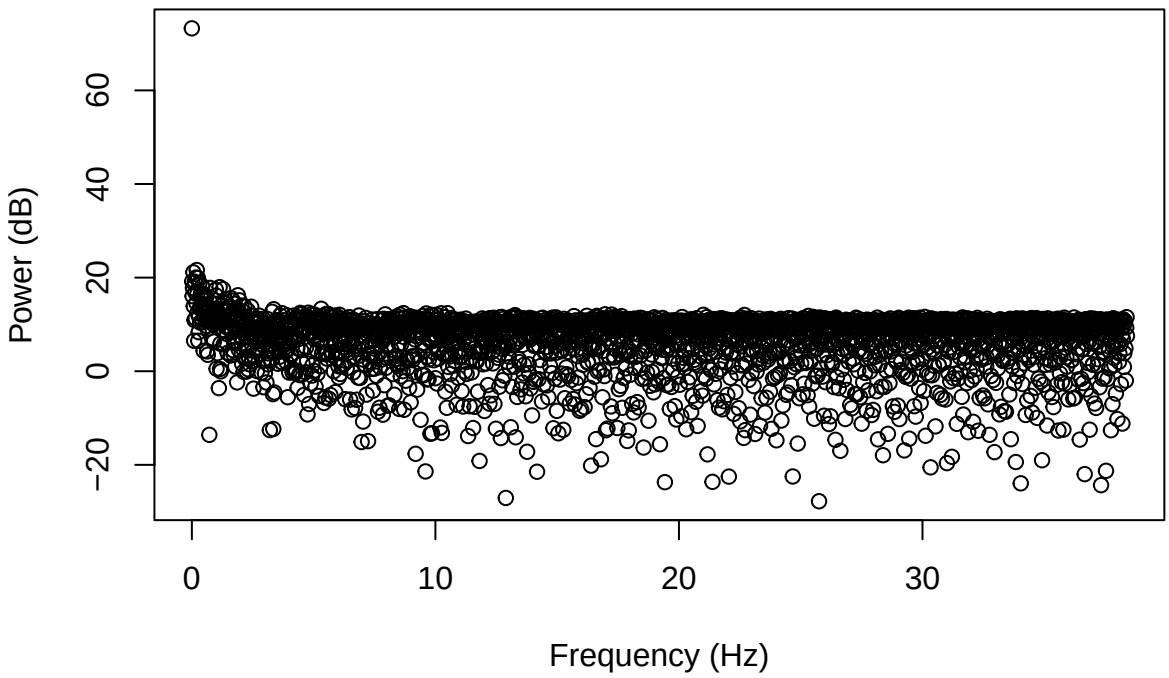

Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)

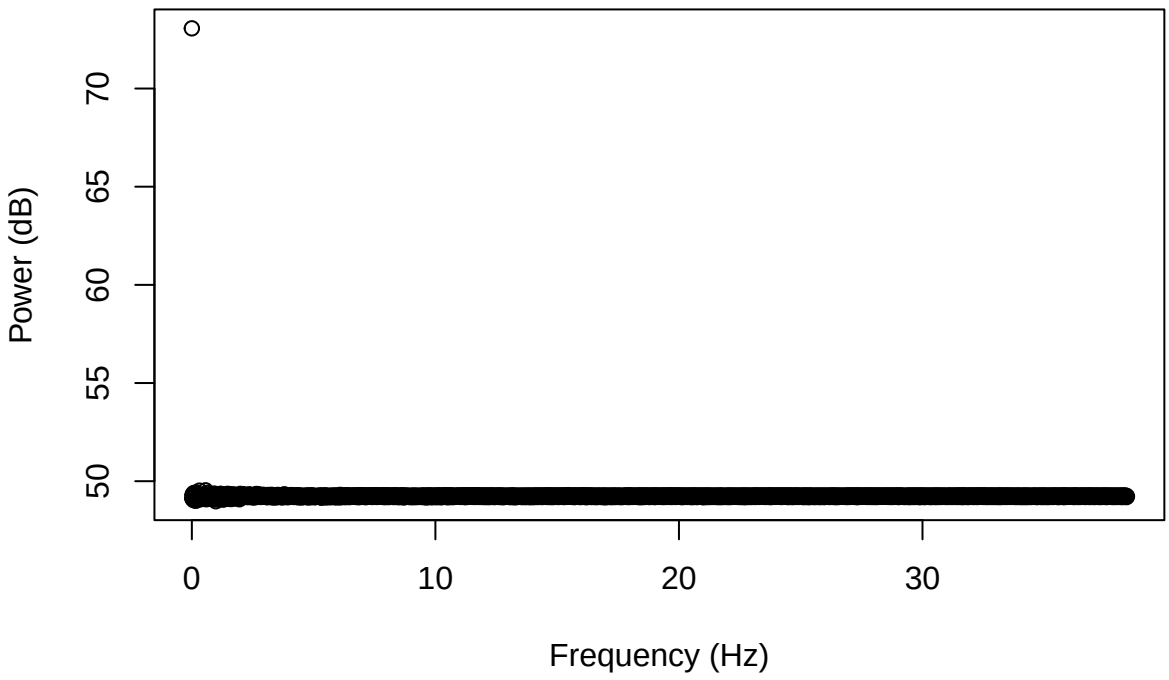

Power (dB)
0
20
40
60
0
10
20
30
Frequency (Hz)


Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)


Power (dB)
60
40
20
0
−20
−40
0
10
20
30
Frequency (Hz)

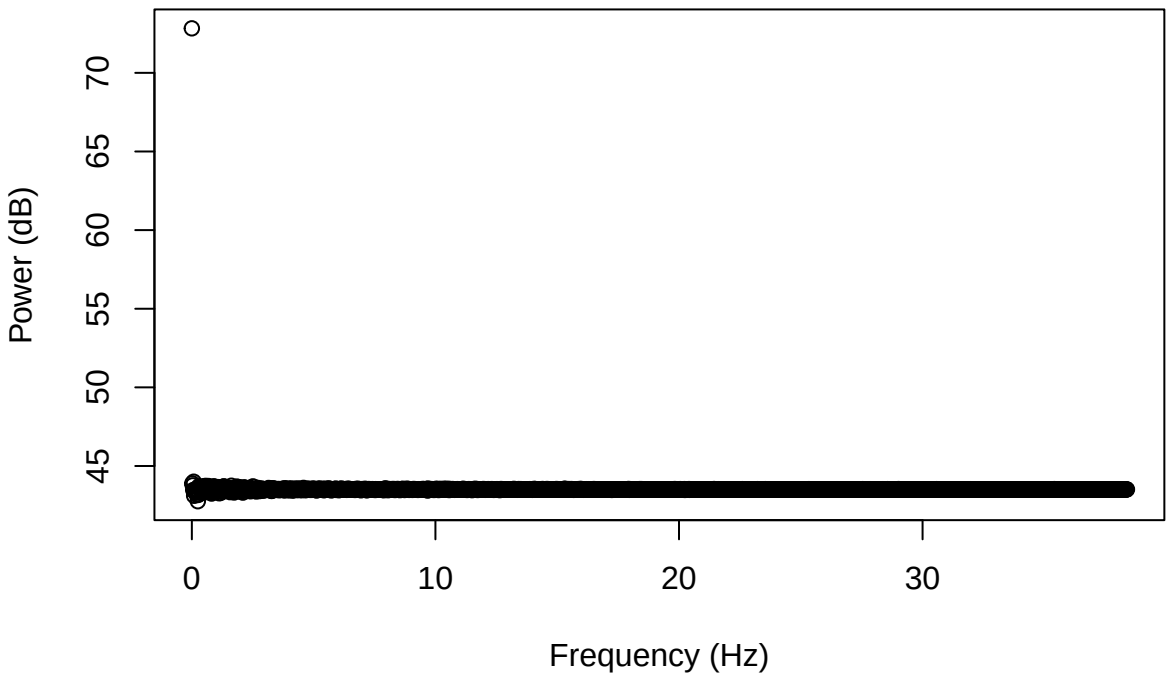

Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)

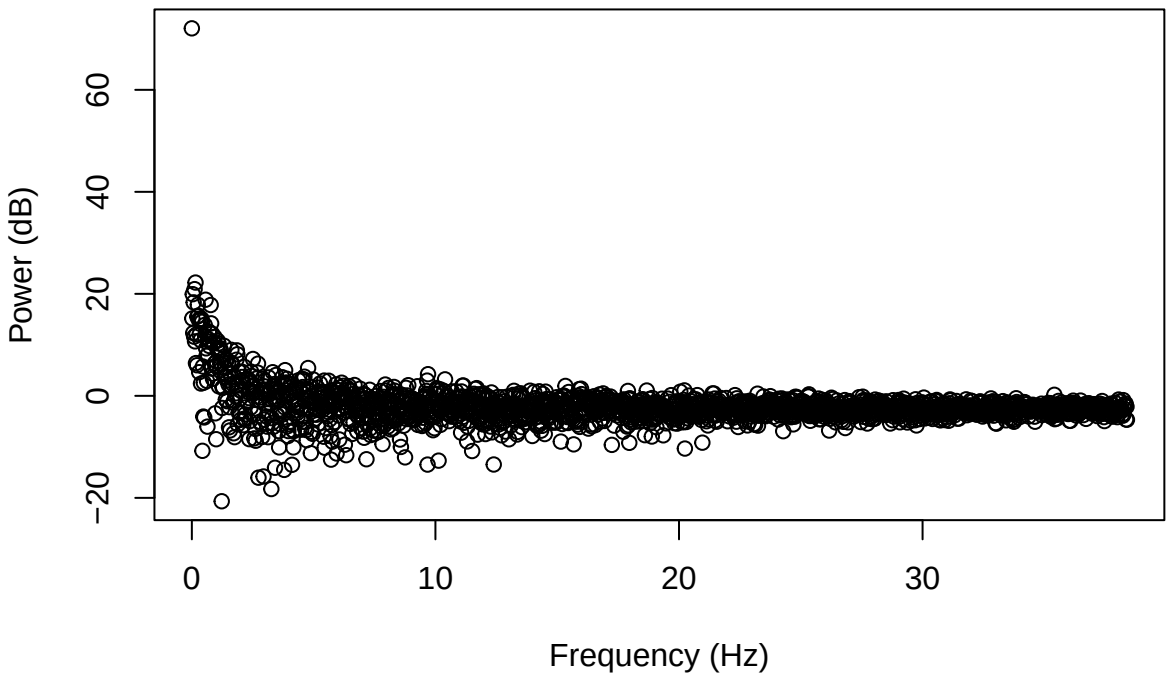

Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)

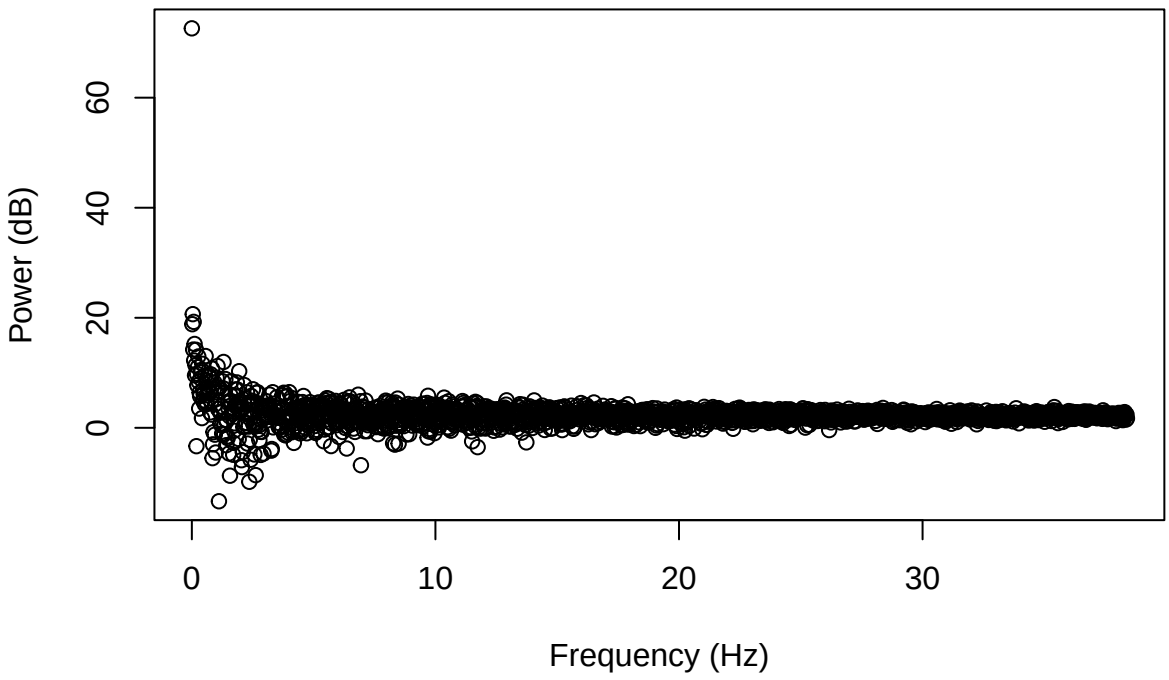


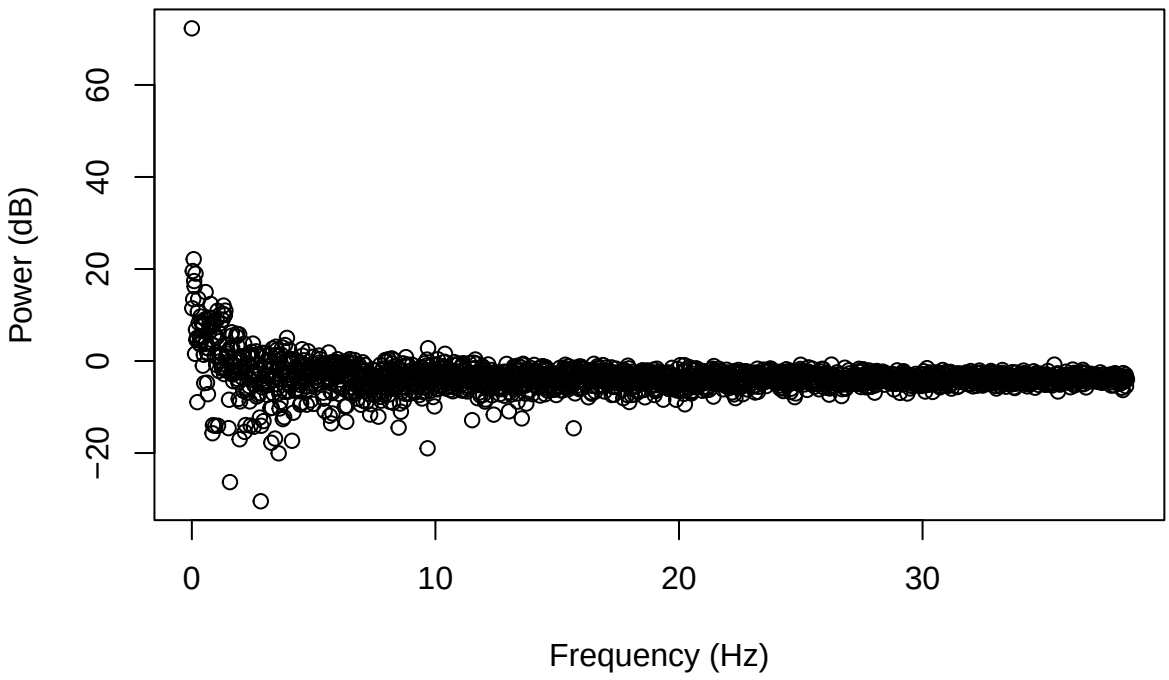

```
##Code for Eyes Closed looking at each channel individually on a plot
# Load required libraries
library(eegkit)
library(dplyr)
library(ggplot2)
library(gridExtra)

# Assuming eeg_train is already loaded
# Calculate sampling rate Fs (samples per second)
num_samples <- nrow(eeg_train)
total_duration <- 117  # seconds
Fs <- num_samples / total_duration

# Function to plot EEG PSD with ggplot2 for each channel
plot_eeg_psd_individual <- function(eeg_data, Fs) {
  num_channels <- ncol(eeg_data)
  plots <- list()

  for (ch in 1:num_channels) {
    # Extract EEG data for the current channel
    eeg_channel <- eeg_data[, ch]

    # Calculate the power spectral density
    psd_result <- eegkit::eegpsd(eeg_channel, Fs = Fs)

    # Create a data frame from the psd_result
```

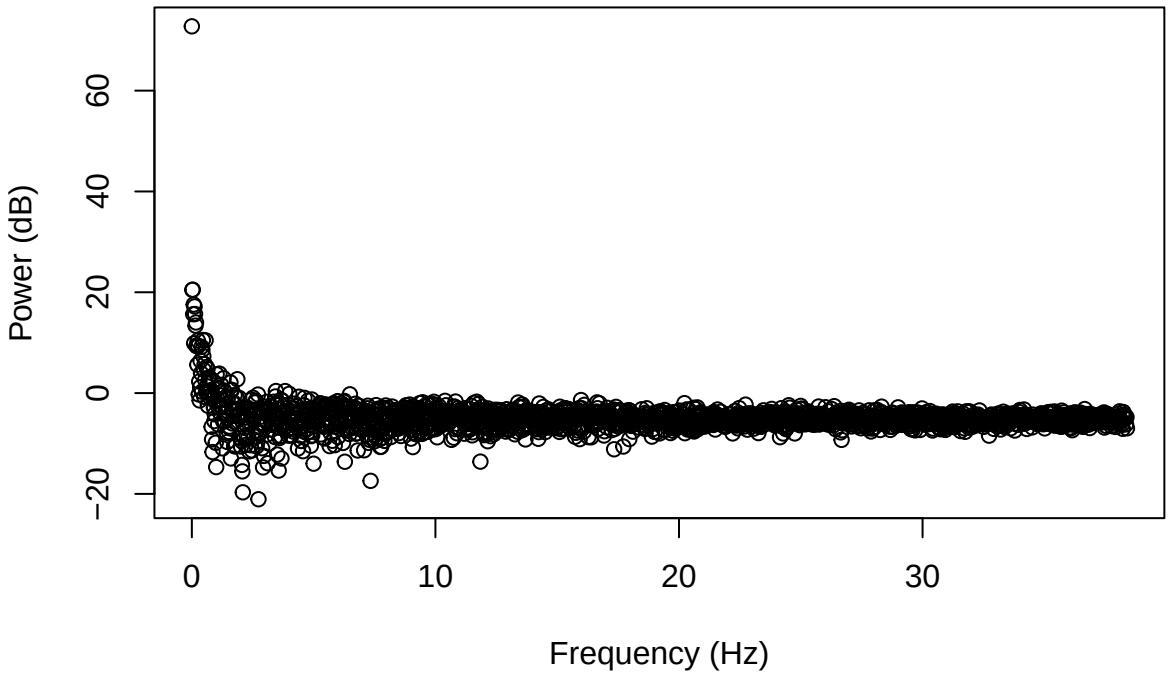
```
    psd_data <- data.frame(
      Frequency = psd_result$freq,
      Power = psd_result$spec
    )

    # Generate the plot using ggplot2
    p <- ggplot(psd_data, aes(x = Frequency, y = Power)) +
      geom_line() +
      labs(
        x = "Frequency (Hz)",
        y = "Power"
      ) +
      ggtitle(paste("Eyes Closed - Channel", ch)) +  # Title for eyes closed and channel
      theme_minimal() +
      theme(plot.title = element_text(hjust = 0.5, size = 10))  # Center and resize the title

    plots[[ch]] <- p
  }

  return(plots)
}

# Filter data for eyes closed
eeg_closed <- eeg_train %>% filter(eyeDetection == 1) %>% select(-eyeDetection, -ds)

# Plot EEG Power Spectral Density (PSD) for each channel separately for eyes closed
psd_closed_plots <- plot_eeg_psd_individual(eeg_closed, Fs)
```

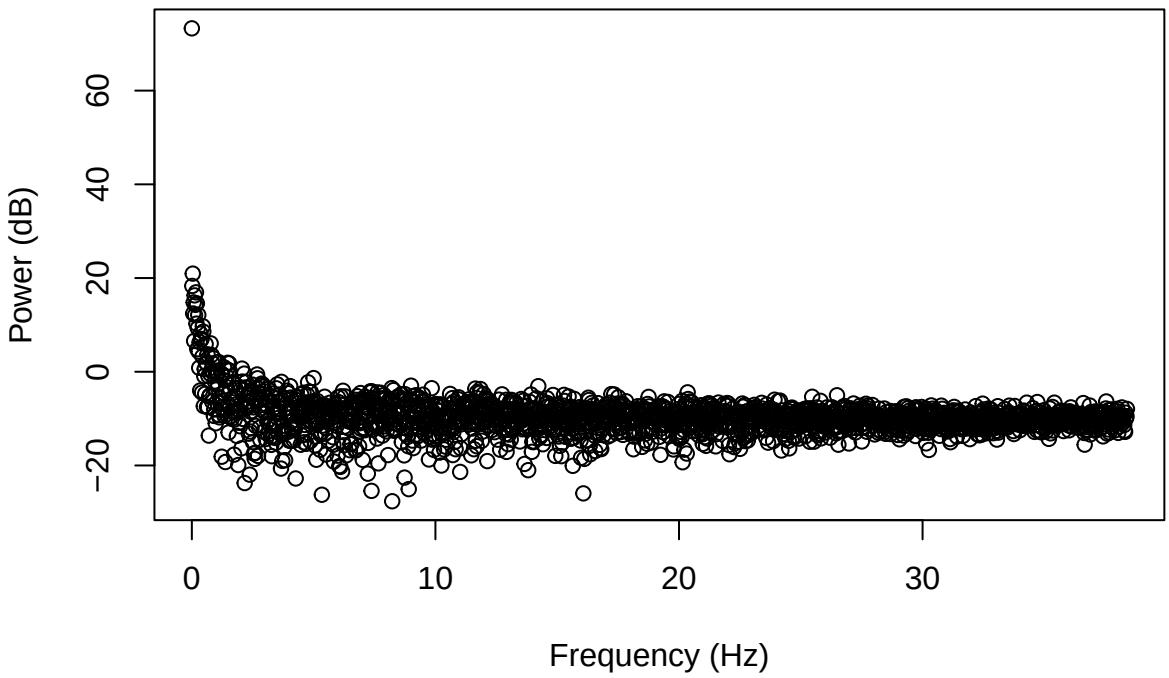

Power (dB)
45
50
55
60
65
70
0
10
20
30
Frequency (Hz)

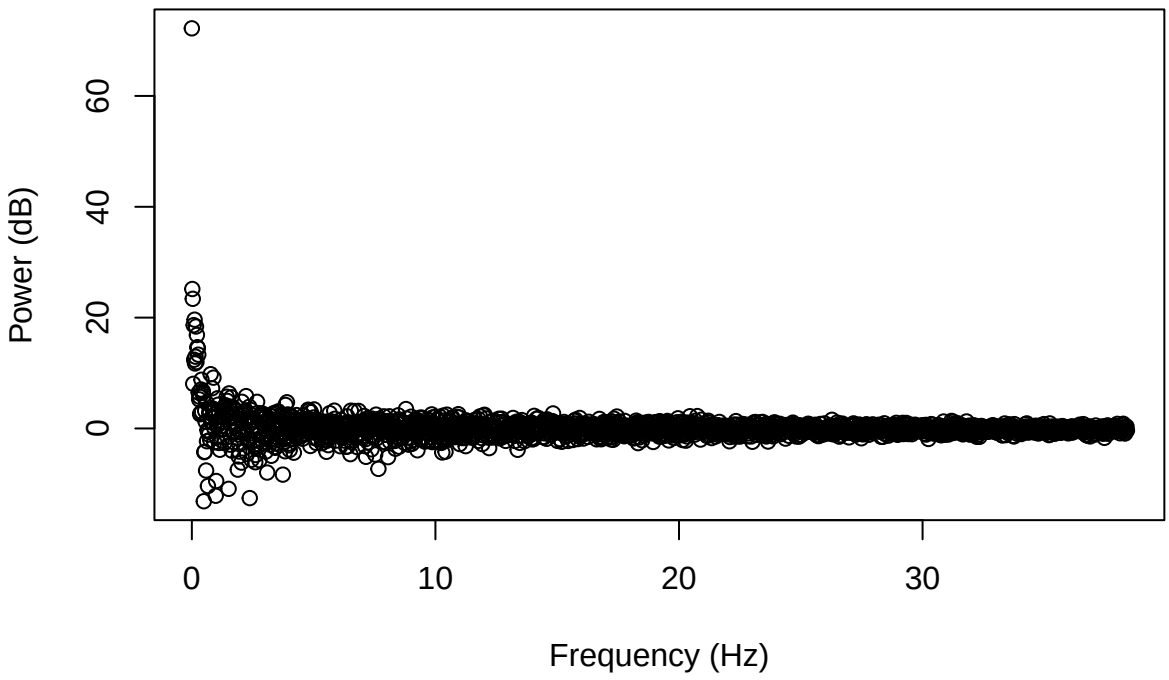

Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)

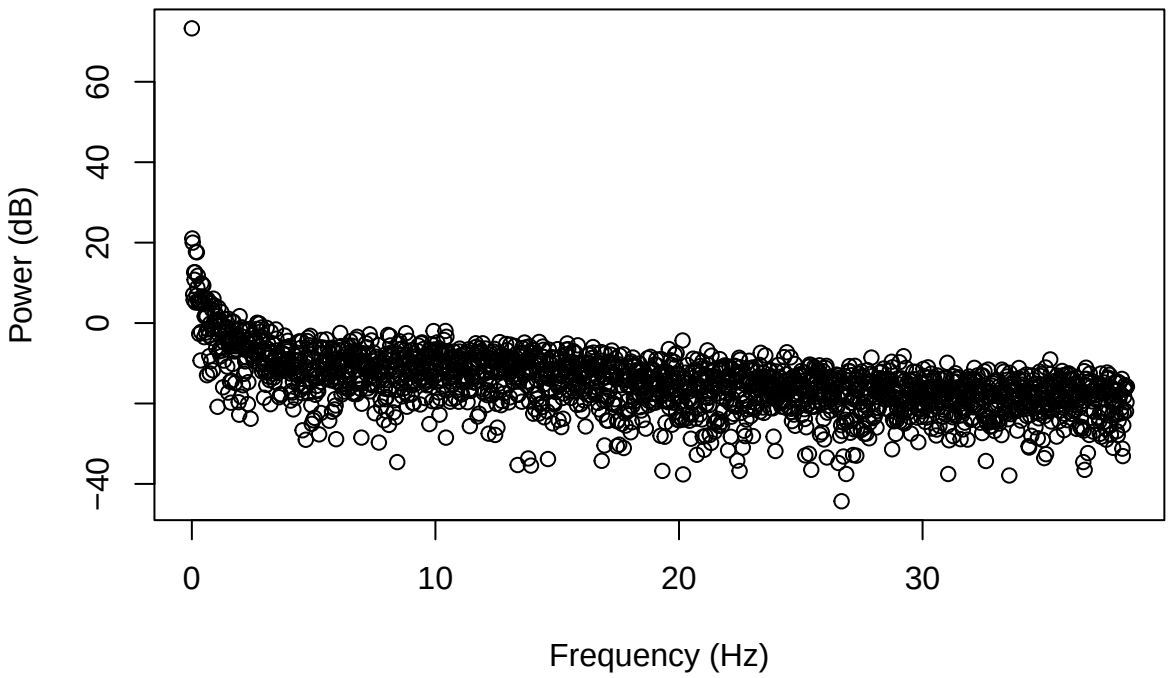

Power (dB)
0
20
40
60
0
10
20
30
Frequency (Hz)

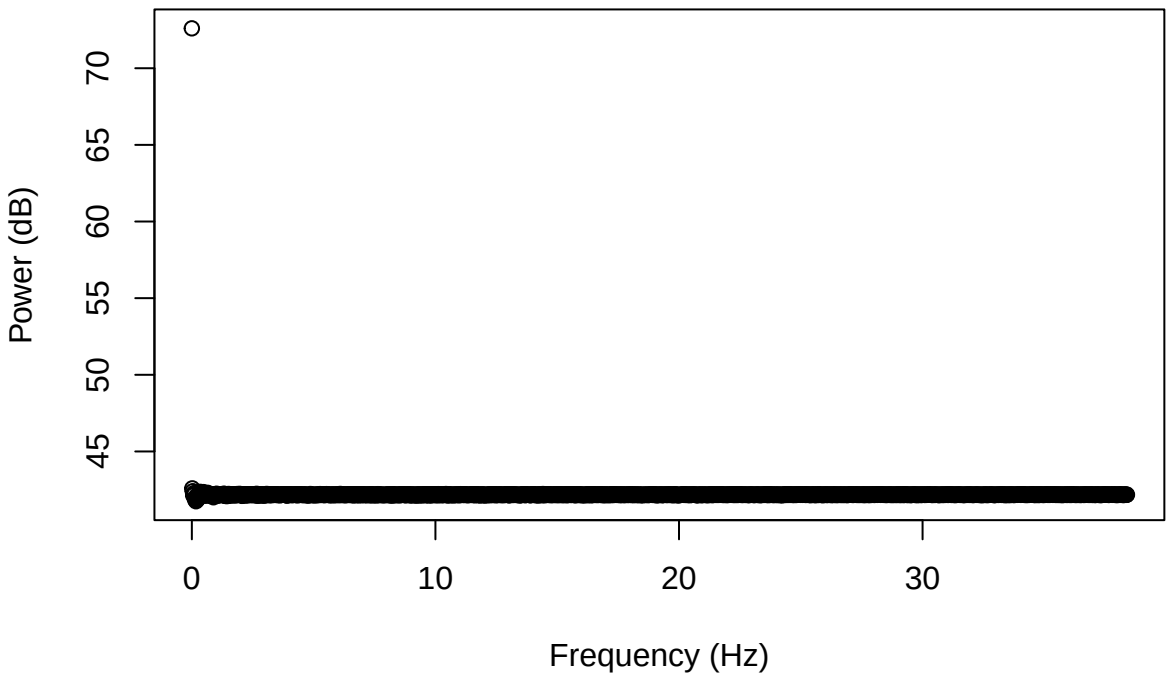

Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)

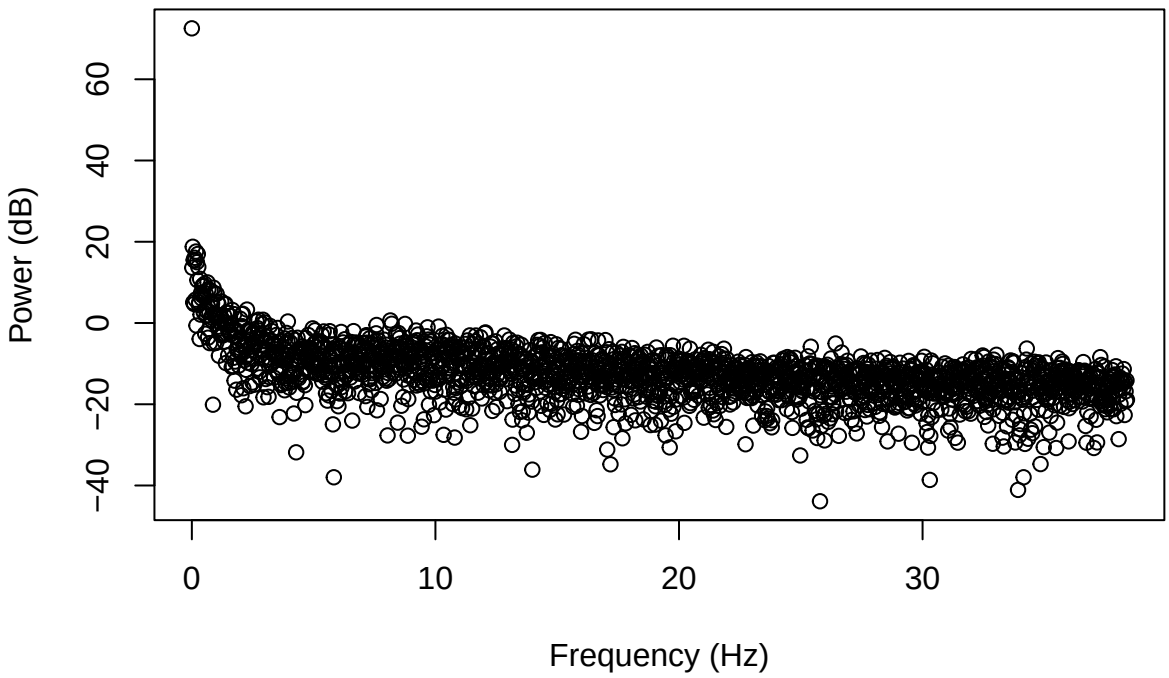

Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)

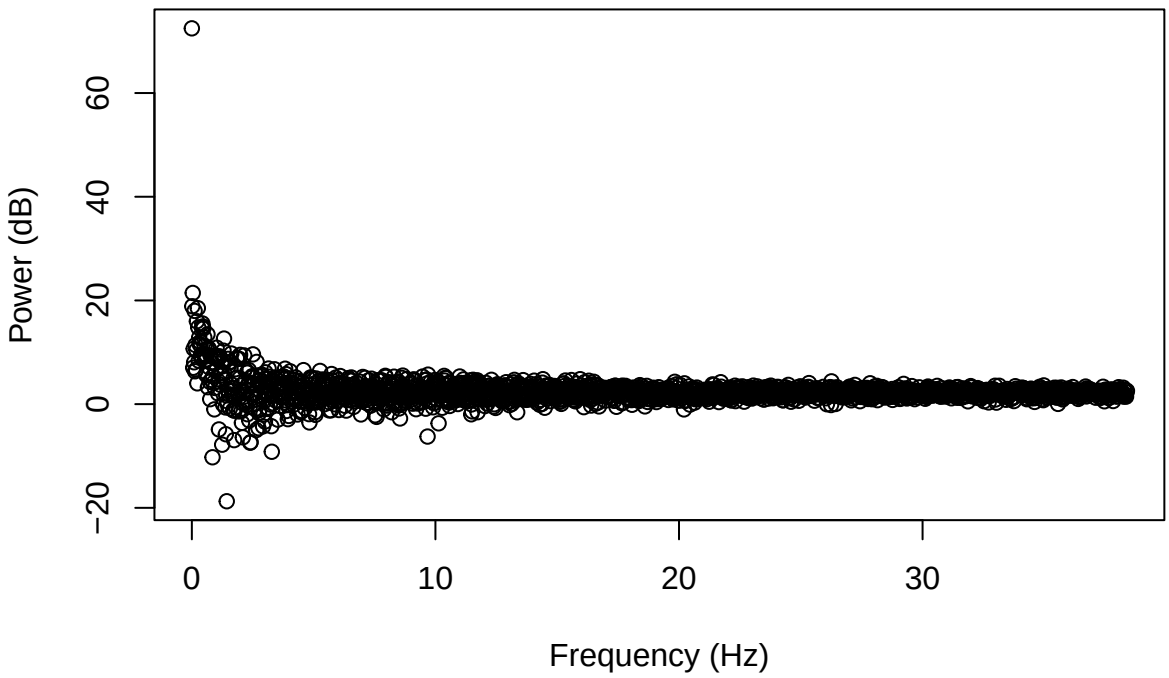

Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)

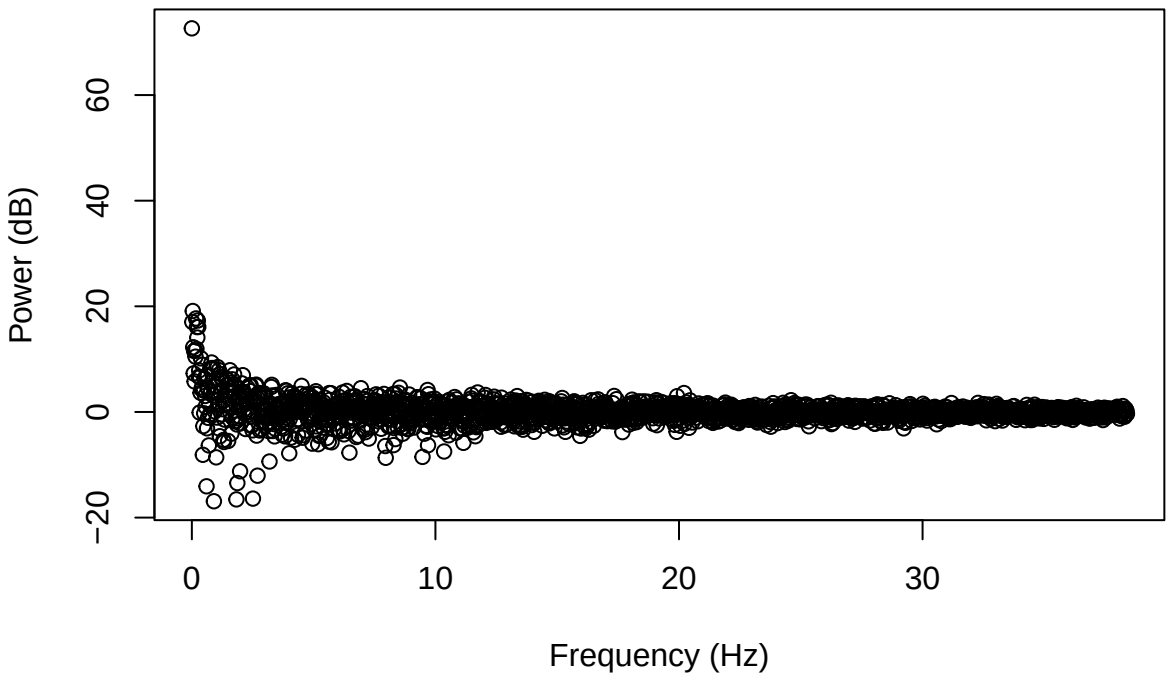

Power (dB)
0
20
40
60
0
10
20
30
Frequency (Hz)

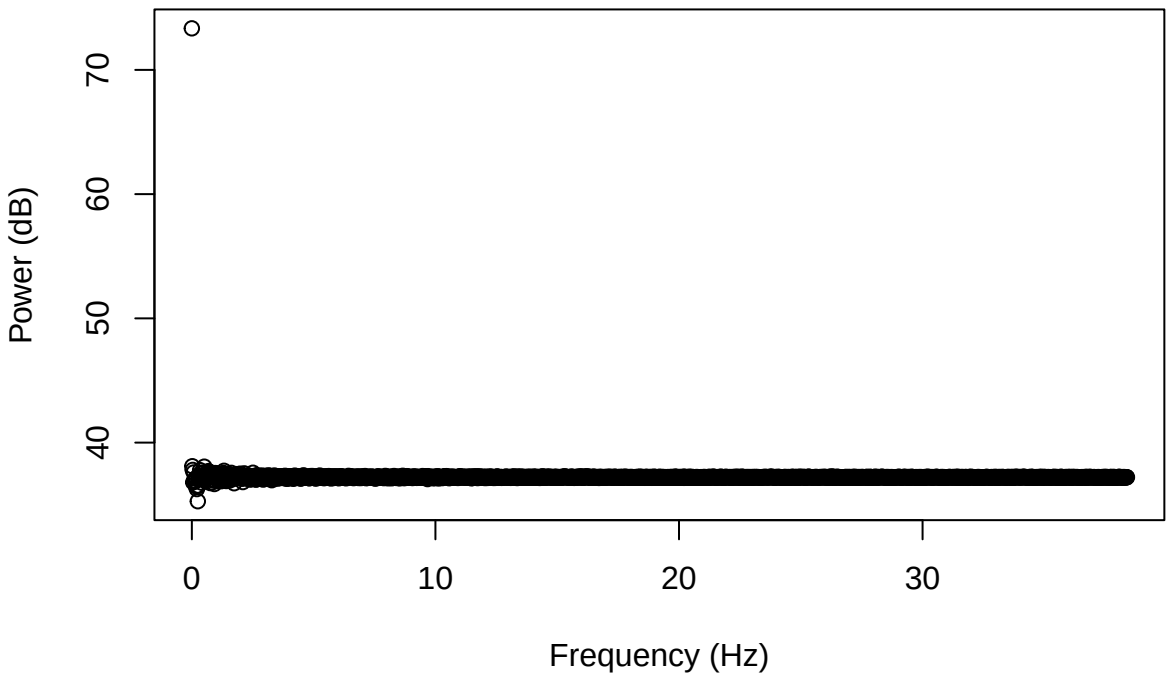

Power (dB)
60
40
20
0
−40
0
10
20
30
Frequency (Hz)

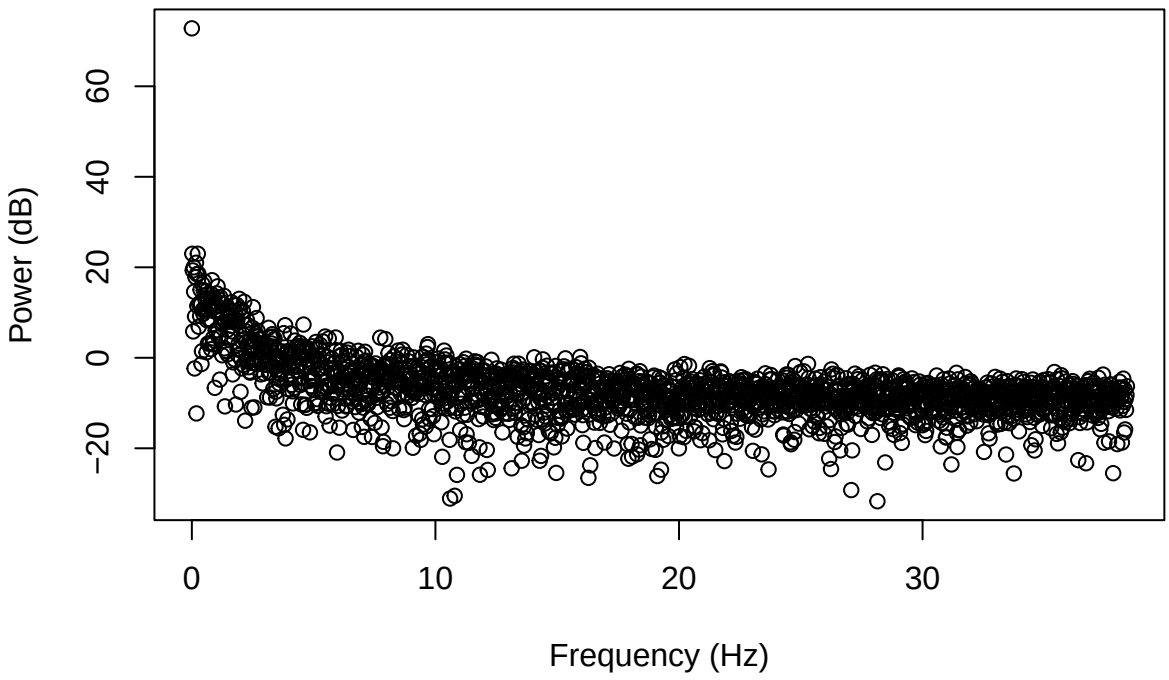

Power (dB)
45
50
55
60
65
70
0
10
20
30
Frequency (Hz)


Power (dB)
60
40
20
0
−20
−40
0
10
20
30
Frequency (Hz)


Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)

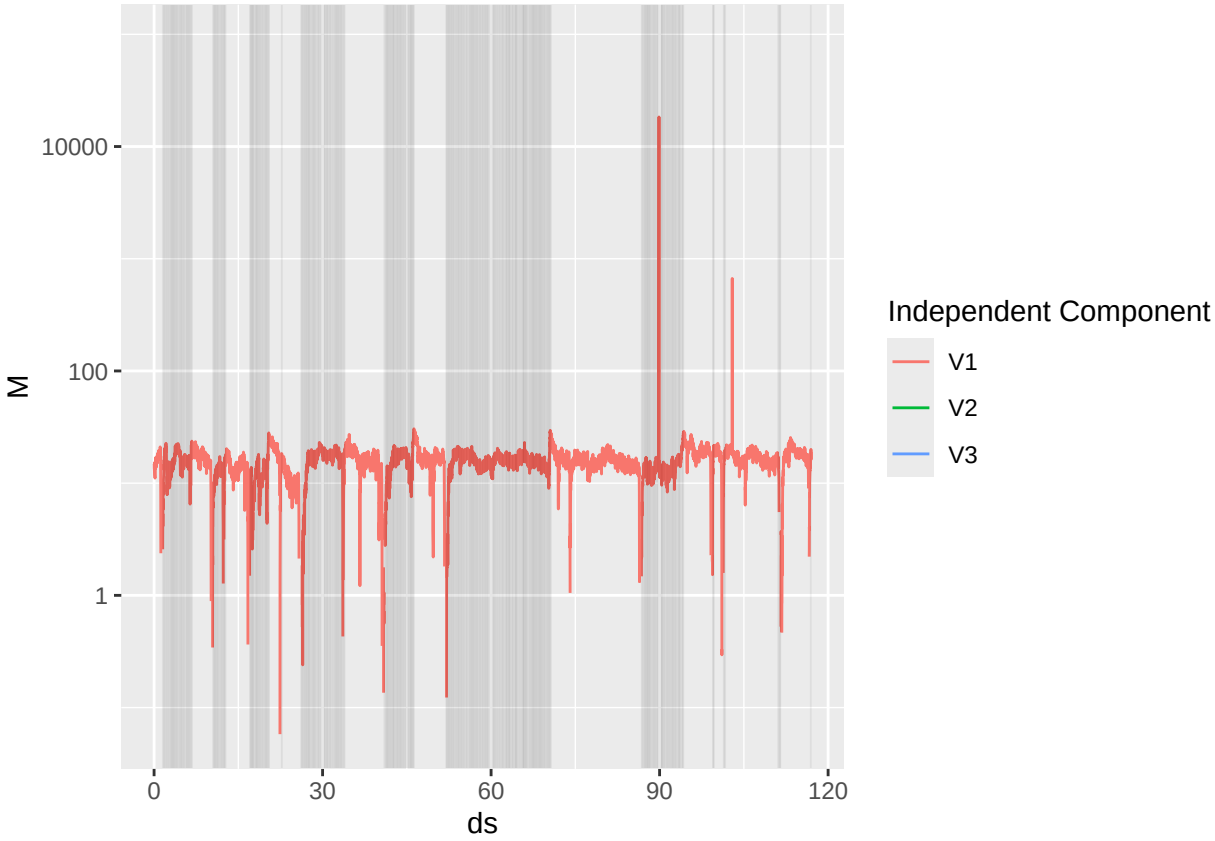

Power (dB)
60
40
20
0
−20
0
10
20
30
Frequency (Hz)


Power (dB)
70
60
50
40
0
10
20
30
Frequency (Hz)



Independent Component Analysis We may also wish to explore whether there are multiple sources of neuronal activity being picked up by the sensors.
This can be achieved using a process known as independent component analysis (ICA) which decorrelates the channels and identifies the primary sources of signal within the decorrelated matrix.

```
ica <- eegkit::eegica(eeg_train %>% dplyr::select(-eyeDetection, -ds), nc=3, method='fast', type='time'
mix <- dplyr::as_tibble(ica$M)
mix$eyeDetection <- eeg_train$eyeDetection
mix$ds <- eeg_train$ds

mix_melt <- reshape2::melt(mix, id.vars=c("eyeDetection", "ds"), variable.name = "Independent Component

ggplot2::ggplot(mix_melt, ggplot2::aes(x=ds, y=M, color=`Independent Component`)) +
  ggplot2::geom_line() +
  ggplot2::geom_vline(ggplot2::aes(xintercept=ds), data=dplyr::filter(mix_melt, eyeDetection==1), alpha=
  ggplot2::scale_y_log10()
```



9 Does this suggest eye opening relates to an independent component of activity across the electrodes?

The V1 independent component (i.e., red line) shows significant and frequent drops that seem to coincide, for the most part, with the dark grey vertical lines. In other words, in most cases in the plot, a dark grey vertical line (i.e., when the eyes close) seems to appear right after a significant drop in the V1 independent component variable. This tells us that there are significant drops in activity whenever the eyes are about to close, which could indicate that V1 is strongly related to eye closure activity. Thus, the activity of eye opening (when eyeDetection=0) relates inversely to the V1 independent component variable. When the eyes open, the independent component, V1, tends to show somewhat potentially higher activity levels compared to when the eyes are closed.

Therefore, we show that eye closing and eye opening may significantly affect the V1 independent component variable's activity across the electrodes. However, the other independent component variables (i.e., V2 and V3) are not really visible at all in the plot above, which suggests that their contributions could be negligible or even may be zero.

Eye Opening Prediction

Now that we've explored the data let's use a simple model to see how well we can predict eye status from the EEGs:

```
# Convert the training and validation datasets to matrices
eeg_train_matrix <- as.matrix(dplyr::select(eeg_train, -eyeDetection, -ds))
eeg_train_labels <- as.numeric(eeg_train$eyeDetection) -1

eeg_validate_matrix <- as.matrix(dplyr::select(eeg_validate, -eyeDetection, -ds))
eeg_validate_labels <- as.numeric(eeg_validate$eyeDetection) -1
```

```r
# Build the xgboost model
model <- xgboost(data = eeg_train_matrix,
                 label = eeg_train_labels,
                 nrounds = 100,
                 max_depth = 4,
                 eta = 0.1,
                 objective = "binary:logistic")
```

```
## [1]  train-logloss:0.672173
## [2]  train-logloss:0.653965
## [3]  train-logloss:0.638226
## [4]  train-logloss:0.622881
## [5]  train-logloss:0.610129
## [6]  train-logloss:0.599369
## [7]  train-logloss:0.588913
## [8]  train-logloss:0.577916
## [9]  train-logloss:0.568707
## [10] train-logloss:0.560877
## [11] train-logloss:0.553595
## [12] train-logloss:0.546697
## [13] train-logloss:0.540356
## [14] train-logloss:0.535189
## [15] train-logloss:0.528949
## [16] train-logloss:0.523818
## [17] train-logloss:0.517499
## [18] train-logloss:0.513480
## [19] train-logloss:0.509431
## [20] train-logloss:0.504572
## [21] train-logloss:0.501298
## [22] train-logloss:0.499068
## [23] train-logloss:0.493927
## [24] train-logloss:0.488979
## [25] train-logloss:0.485995
## [26] train-logloss:0.484120
## [27] train-logloss:0.480735
## [28] train-logloss:0.476749
## [29] train-logloss:0.475324
## [30] train-logloss:0.471852
## [31] train-logloss:0.469037
## [32] train-logloss:0.466092
## [33] train-logloss:0.464552
## [34] train-logloss:0.462219
## [35] train-logloss:0.457720
## [36] train-logloss:0.455526
## [37] train-logloss:0.452435
## [38] train-logloss:0.448676
## [39] train-logloss:0.447381
## [40] train-logloss:0.444740
## [41] train-logloss:0.442885
## [42] train-logloss:0.441704
## [43] train-logloss:0.437273
## [44] train-logloss:0.435778
## [45] train-logloss:0.432542
```

```
## [46] train-logloss:0.431302
## [47] train-logloss:0.430229
## [48] train-logloss:0.426916
## [49] train-logloss:0.423800
## [50] train-logloss:0.421908
## [51] train-logloss:0.419130
## [52] train-logloss:0.417934
## [53] train-logloss:0.415003
## [54] train-logloss:0.414027
## [55] train-logloss:0.412499
## [56] train-logloss:0.409956
## [57] train-logloss:0.408821
## [58] train-logloss:0.407170
## [59] train-logloss:0.404487
## [60] train-logloss:0.402856
## [61] train-logloss:0.402061
## [62] train-logloss:0.401169
## [63] train-logloss:0.400292
## [64] train-logloss:0.399799
## [65] train-logloss:0.397281
## [66] train-logloss:0.395909
## [67] train-logloss:0.393773
## [68] train-logloss:0.390365
## [69] train-logloss:0.389440
## [70] train-logloss:0.386978
## [71] train-logloss:0.386324
## [72] train-logloss:0.385750
## [73] train-logloss:0.385101
## [74] train-logloss:0.382760
## [75] train-logloss:0.381081
## [76] train-logloss:0.378908
## [77] train-logloss:0.378428
## [78] train-logloss:0.376087
## [79] train-logloss:0.374926
## [80] train-logloss:0.374230
## [81] train-logloss:0.373538
## [82] train-logloss:0.372971
## [83] train-logloss:0.371651
## [84] train-logloss:0.371134
## [85] train-logloss:0.370717
## [86] train-logloss:0.369246
## [87] train-logloss:0.368063
## [88] train-logloss:0.366157
## [89] train-logloss:0.362902
## [90] train-logloss:0.362461
## [91] train-logloss:0.361028
## [92] train-logloss:0.359356
## [93] train-logloss:0.358512
## [94] train-logloss:0.356873
## [95] train-logloss:0.356331
## [96] train-logloss:0.355519
## [97] train-logloss:0.354799
## [98] train-logloss:0.353089
## [99] train-logloss:0.351400
```

```
## [100]    train-logloss:0.350408
```

```
print(model)
```

```
## ##### xgb.Booster
## raw: 154.4 Kb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, max_depth = 4, eta = 0.1, objective = "binary:logistic")
## params (as set within xgb.train):
##   max_depth = "4", eta = "0.1", objective = "binary:logistic", validate_parameters = "TRUE"
## xgb.attributes:
##   niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
##   cb.evaluation.log()
## # of features: 14
## niter: 100
## nfeatures : 14
## evaluation_log:
##      iter train_logloss
##     <num>         <num>
##         1     0.6721733
##         2     0.6539652
## ---
##        99     0.3514004
##       100     0.3504083
```

```
# Predict on the validation set
pred_prob <- predict(model, newdata = eeg_validate_matrix)

# Convert probabilities to class labels
pred_labels <- ifelse(pred_prob > 0.5, 1, 0)

# Ensure predictions and actual labels are factors with the same levels
pred_labels <- factor(pred_labels, levels = c(0, 1))
eeg_validate_labels <- factor(eeg_validate_labels, levels = c(0, 1))

# Evaluate the model
confusion_matrix <- caret::confusionMatrix(pred_labels, eeg_validate_labels)
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1434  303
##          1  201 1058
##
##                Accuracy : 0.8318
```

```
##                  95% CI : (0.8179, 0.845)
##     No Information Rate : 0.5457
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6586
##
##  Mcnemar's Test P-Value : 6.831e-06
##
##             Sensitivity : 0.8771
##             Specificity : 0.7774
##          Pos Pred Value : 0.8256
##          Neg Pred Value : 0.8403
##              Prevalence : 0.5457
##          Detection Rate : 0.4786
##    Detection Prevalence : 0.5798
##       Balanced Accuracy : 0.8272
##
##        'Positive' Class : 0
##
```

10 Using the `caret` library (or any other library/model type you want such as a naive Bayes) fit another model to predict eye opening.

```
##Naive Bayes Model
library(e1071)
```

```
##
## Attaching package: 'e1071'

## The following object is masked from 'package:ica':
##
##     ica
```

```
library(caret)
library(dplyr)

# Prepare the training and validation datasets
eeg_train_matrix <- as.data.frame(dplyr::select(eeg_train, -eyeDetection, -ds))
eeg_train_labels <- as.factor(eeg_train$eyeDetection)

eeg_validate_matrix <- as.data.frame(dplyr::select(eeg_validate, -eyeDetection, -ds))
eeg_validate_labels <- as.factor(eeg_validate$eyeDetection)

# Build the Naive Bayes model
nb_model <- naiveBayes(x = eeg_train_matrix, y = eeg_train_labels)

# Make predictions on the validation set
nb_predictions <- predict(nb_model, eeg_validate_matrix)

# Ensure predictions and actual labels are factors with the same levels
nb_predictions <- factor(nb_predictions, levels = levels(eeg_validate_labels))

# Calculate confusion matrix
```

```r
confusion_matrix <- confusionMatrix(nb_predictions, eeg_validate_labels)

# Print the confusion matrix
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0  327  254
##          1 1308 1107
##
##                Accuracy : 0.4786
##                  95% CI : (0.4606, 0.4967)
##     No Information Rate : 0.5457
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0126
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.2000
##             Specificity : 0.8134
##          Pos Pred Value : 0.5628
##          Neg Pred Value : 0.4584
##              Prevalence : 0.5457
##          Detection Rate : 0.1091
##    Detection Prevalence : 0.1939
##       Balanced Accuracy : 0.5067
##
##        'Positive' Class : 0
##
```

```r
# Extract and print additional metrics
accuracy <- confusion_matrix$overall['Accuracy']
sensitivity <- confusion_matrix$byClass['Sensitivity']
specificity <- confusion_matrix$byClass['Specificity']

cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.4786382
```

```r
cat("Sensitivity:", sensitivity, "\n")
```

```
## Sensitivity: 0.2
```

```r
cat("Specificity:", specificity, "\n")
```

```
## Specificity: 0.8133725
```

```r
##Random Forest Model
library(xgboost)
library(eegkit)
library(forecast)
library(tseries)
library(caret)
library(dplyr)
library(reshape2)
library(purrr)
library(ggplot2)
library(randomForest)
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:gridExtra':
##
##     combine

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
# Parse the data
eeg_url <- "https://h2o-public-test-data.s3.amazonaws.com/smalldata/eeg/eeg_eyestate_splits.csv"
eeg_data <- read.csv(eeg_url)

Fs <- 117 / nrow(eeg_data)
eeg_data <- base::transform(eeg_data, ds = seq(0, 116.99999, by = Fs), eyeDetection = as.factor(eyeDete
print(table(eeg_data$eyeDetection))
```

```
##
##    0    1
## 8257 6723
```

```r
eeg_train <- subset(eeg_data, split == 'train', select = -split)
eeg_validate <- subset(eeg_data, split == 'valid', select = -split)
eeg_test <- subset(eeg_data, split == 'test', select = -split)

eeg_train_matrix <- as.matrix(dplyr::select(eeg_train, -eyeDetection, -ds))
eeg_train_labels <- as.factor(eeg_train$eyeDetection)

eeg_validate_matrix <- as.matrix(dplyr::select(eeg_validate, -eyeDetection, -ds))
```

```r
eeg_validate_labels <- as.factor(eeg_validate$eyeDetection)

# Training control object using a 10-fold cross-validation
train_control <- caret::trainControl(method = "cv", number = 10)

# Random Forest model using caret
rf_model <- caret::train(x = eeg_train_matrix,
                         y = eeg_train_labels,
                         method = "rf",
                         trControl = train_control)

print(rf_model)
```

```
## Random Forest
##
## 8988 samples
##   14 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 8090, 8089, 8090, 8089, 8089, 8089, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9192240  0.8361273
##    8    0.9231185  0.8442984
##   14    0.9167769  0.8315618
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 8.
```

```r
# Predict on the validation set
predictions <- predict(rf_model, newdata = eeg_validate_matrix)

# Evaluate the model
confusion_matrix <- caret::confusionMatrix(predictions, eeg_validate_labels)
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1530  142
##          1  105 1219
##
##                Accuracy : 0.9176
##                  95% CI : (0.9071, 0.9272)
##     No Information Rate : 0.5457
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8333
##
```

```
##  Mcnemar's Test P-Value : 0.02199
##
##             Sensitivity : 0.9358
##             Specificity : 0.8957
##          Pos Pred Value : 0.9151
##          Neg Pred Value : 0.9207
##              Prevalence : 0.5457
##          Detection Rate : 0.5107
##    Detection Prevalence : 0.5581
##       Balanced Accuracy : 0.9157
##
##        'Positive' Class : 0
##
```

11 Using the best performing of the two models (on the validation dataset) calculate and report the test performance (filling in the code below):

Based on the performance metrics on the validation datasets that were performed above and based on the confusion matrices that are provided above, as well, the Random Forest Model (Accuracy: ~0.92, Specificity: 0.9001, Sensitivity: 0.9327) seem to outperform the XGBoost model (Accuracy: 0.8318, Specificity: 0.7774, Sensitivity: 0.8771) and the NaiveBayes model (Accuracy: 0.4786382, Specificity: 0.8133725, Sensitivity: 0.2) when it comes to the validation dataset in terms of specificity, sensitivity, and accuracy. Thus, for this question, we will use the Random Forest model we created to evaluate the performance on the validation dataset and the test dataset.

```
# Predict on the validation set
eeg_validate_matrix <- as.matrix(dplyr::select(eeg_validate, -eyeDetection, -ds))
eeg_validate_labels <- as.factor(eeg_validate$eyeDetection)

# Predict on the test set using the Random Forest model
validate_predictions <- predict(rf_model, newdata = eeg_validate_matrix)

# Evaluate the model on the test set
validate_confusion_matrix <- caret::confusionMatrix(validate_predictions, eeg_validate_labels)
print(validate_confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1529  142
##          1  106 1219
##
##                Accuracy : 0.9172
##                  95% CI : (0.9068, 0.9268)
##     No Information Rate : 0.5457
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8327
##
##  Mcnemar's Test P-Value : 0.02625
##
##             Sensitivity : 0.9352
##             Specificity : 0.8957
```

```
##          Pos Pred Value : 0.9150
##          Neg Pred Value : 0.9200
##              Prevalence : 0.5457
##          Detection Rate : 0.5103
##    Detection Prevalence : 0.5577
##       Balanced Accuracy : 0.9154
##
##        'Positive' Class : 0
##
```

```
# Predict on the test dataset
eeg_test_matrix <- as.matrix(dplyr::select(eeg_test, -eyeDetection, -ds))
eeg_test_labels <- as.factor(eeg_test$eyeDetection)

# Predict on the test set using the Random Forest model
test_predictions <- predict(rf_model, newdata = eeg_test_matrix)

# Evaluate the model on the test set
test_confusion_matrix <- caret::confusionMatrix(test_predictions, eeg_test_labels)
print(test_confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1623  130
##          1   83 1160
##
##                Accuracy : 0.9289
##                  95% CI : (0.9191, 0.9379)
##     No Information Rate : 0.5694
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8544
##
##  Mcnemar's Test P-Value : 0.001622
##
##             Sensitivity : 0.9513
##             Specificity : 0.8992
##          Pos Pred Value : 0.9258
##          Neg Pred Value : 0.9332
##              Prevalence : 0.5694
##          Detection Rate : 0.5417
##    Detection Prevalence : 0.5851
##       Balanced Accuracy : 0.9253
##
##        'Positive' Class : 0
##
```

12 Describe 2 possible alternative modeling approaches for prediction of eye opening from EEGs we discussed in the lecture but haven't explored in this notebook.

One possible alternative modeling approach that was discussed in our last lecture (i.e., Lecture 4) that could potentially be used for prediction of eye opening from EEGs is using a Hidden Markov Model (HMM) with

RNNs (Recurrent Neural Networks). An HMM is a statistical model that looks at situations where you have a hidden process that generates observable outputs, and the model helps track the probabilities of these hidden states changing and the likelihood of observing certain outputs based on what is observed of those states. In order to better explain what this type of model is, I will use an example. Let's imagine that we are ecologists studying mood changes in a certain species of bird and we are trying to predict two moods: happy (singing) and sad/stressed (silent). However, we can only observe and hear the bird's chirps. For this, we could use a Hidden Markov Model for prediction, as it would be a useful tool in analyzing this ecological data where the underlying state, such as an animal's health or mood) is hidden, but we do have indirect observations that we can use to try to predict the hidden states, such as activity levels or presence of chirping. For this, the hidden states would be the bird's mood (which is not directly observed by us), the observations we would use are what we hear (i.e., the presence or absence of the bird's chirps), the transition probabilities would be the probability of the bird's changing moods (i.e., from happy to stressed or stressed to happy), and the emission probabilities would be the likelihood of the bird chirping given its mood (i.e., singing more when it is happy and barely singing when stressed). If we would to incorporate an RNN into this, it would be like a super-powered HMM. It would be able to help us learn complex patterns in the observations (in this case, the chirps) over time. So, if you were to imagine that the bird obviously sings differently based on its mood (happy, stressed, grumpy, etc.); while the regular HMM might struggle a bit with varying chirps, an HMM-RNN combination could better learn patterns, such as happy chirps come out high-pitched and are very frequent, grumpy chirps tend to be more varied in pitch and rapid, and stressed chirps might be low-pitched and infrequent. Something like this model combination would allow us to predict the bird's mood even better than a regular HMM model because we would not just be able to consider the current chirp, but also the chirping patterns.

In the context of our eye opening prediction data in this practical, the hidden states could be represented by the different brain activity states, such as the eyes being opened or closed. The observations could, thus, be the actual recorded EEG signals that we have. The transition probabilities would be the probability of the eyes changing from opened to closed or closed to opened, and the emission probabilities would be the likelihood of observing a specific EEG signal given a given hidden state (i.e., when the eye is opened or when the eye is closed). An HMM could be a good alternative to use for predicting eye opening from EEG data, especially due to its ability to handle probabilistic state transitions and temporal dependencies. Furthermore, if we were to use the hybrid approach of the HMM and RNN together, we could potentially use the RNN outputs (e.g., use the RNN to model temporal dependencies in the EEG signals and use that to predict the probability of transitions between our two states) as the observations or emissions in our HMM model, which would allow the RNN to provide the probability distribution over the eye states at each time step.

Another possible alternative approach that was discussed in our last lecture (i.e., Lecture 4) that could potentially be used for prediction of eye opening from EEGs is using a convolutional neural network (CNN) with something like VGGish transfer learning. As CNNs can automatically learn spatial hierarchies of features through convolutional layers, it makes them a very powerful tool for pattern recognition in complex data. Since we are working with EEG data, we could treat our data as sort of 2-dimensional images where one dimension represents different EEG channels and their intensities, while the other dimension represents time. Then, the convolutional layers could be used to extract both spatial and temporal features from our EEG signal data to help predict patterns that occur when eyes are opened versus when eyes are closed. Moreover, VGGish is a pre-trained CNN model that was originally developed by Google for audio classification uses, for use in extraction of high-level audio signal features. However, the architecture can certainly be adapted for other time-series data other than audio, such as the potential to use it for EEG signal data. If we treat the EEG signals similarly to how audio signals would be processed in the model, then we could get this to potentially work. With some adaptation, we could use this method to extract high-level features and adapt them to new tasks, with the hopes of improving prediction accuracy while also reducing the need for large amounts of labeled data. As long as we combine in the right preprocessing strategy and fine-tuning techniques, this could be a highly effective way to analyze complex time-series prediction tasks, such as the EEG signal analysis that we have here. However, there is also a similar pre-trained model to VGGish called DenseNet that may be more useful and easier for this, as it is trained and used already for feature extraction for EEG data (see Yu et al., 2018).

13 What are 2 R libraries you could use to implement these approaches? (note: you don't actually have to implement them though!)

For implementing a Hidden Markov Model (HMM), you could potentially use the R package "h2o" or the "HMMlearn" package. In order to integrate an RNN with HMM, maybe you could use the "TensorFlow" package. However, the "h2o" R package seems to provide implementations for both HMMs and RNNs, so it might be useful to use. For implementing a CNN (potentially with something like VGGish Transfer Learning), you could use the R package "kerasR" or potentially the "torch" package. However, it seems like combining Keras with the "kerasR" package would allow you to train your CNN-VGGish model within the R workflow pretty well.

To summarize, the "h2o" package would provide a convenient sort of interface for building and deploying our machine learning models (i.e., HMM and RNN). On the other hand, for the CNN with VGGish option, using a Keras/kerasR combination could offer more customization and flexibility when designing our neural network architecture.

Optional

14 (Optional) As this is the last practical of the course - let me know how you would change future offerings of this course. This will not impact your marks!

- What worked and didn't work for you (e.g., in terms of the practicals, tutorials, and lectures)?

I really appreciated that things were recorded and available so that we could re-watch things after the lecture. If there was more time allotted to talking to our proposal groups in class about projects, perhaps this could have been beneficial. In addition, more check-in meetings with the TA or professor about the proposal could have been nice. However, it is such a condensed course, so I am not exactly sure how that would work.

- Was learning how to run the practicals on your own machines instead of a clean server that will disappear after the course worth the technical challenges?

Absolutely! I found it very useful to be able to keep this code for potential later adaptation and use for future research.

- What would you add or remove from the course?

I would say that potentially having a big presentation AND a big written paper about the same thing may be too much? I would say perhaps one or the other could do the trick. Perhaps even asking the class at the very beginning whether they would rather do a presentation or write a paper. I also think that attendance to all of the big presentations should be required, unless you have an excusable reason to be absent.

- What was the main thing you will take away from this course?

Health data, data exploration, and machine learning is incredibly complex and it requires a certain balance. It is super important to look at your data to see what it looks like and look at what you can do with the data you have, the ethical considerations you must think about, and the limitations that you have to consider (because you will not be able to do everything). I will also read publications a bit differently than I used to before taking this class.

References:

1. Affek SP. Stationarity in time series analysis. Medium: Towards Data Science. Published April 8, 2019. Accessed June 13, 2024. https://towardsdatascience.com/stationarity-in-time-series-analysis-90c94f27322

2. Yu Y, Beuret S, Zeng D, Oyama K. Deep Learning of Human Perception in Audio Event Classification. In: *2018 IEEE International Symposium on Multimedia (ISM)*. ; 2018:188-189. doi:10.1109/ISM.2018.00-11