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

## Warning in rgl.init(initValue, onlyNULL): RGL: unable to open X11 display

## Warning: 'rgl.init' failed, running with 'rgl.useNULL = TRUE'.

## 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

## Warning: package 'tseries' was built under R version 4.3.3

## 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 <- transform(eeg_data, ds = seq(0, 116.99999, by = Fs), eyeDetection = as.factor(eyeDetection))
print(table(eeg_data$eyeDetection))
```

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

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

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

```
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 to and determine how many samples per second were taken?

From the dataframe we can see that:

1. Duration of EEG Recording: The total duration of the EEG recording is given as 117 seconds.

2.Number of Samples: The nrow(eeg_data) function returns the number of rows in the eeg_data DataFrame, which corresponds to the number of samples taken during the 117 seconds.

3. Sampling Frequency: Fs

So sampling rate will be inverse of Sampling frequency which is 1/Fs

From the provided count, total number of samples is 8257 + 6723 = 14980

Fs = 117/n_samples = 117/14980 = 0.0078104

Sampling rate = 1/Fs = 1/0.00078104 = 128 samples per second

1 How many EEG electrodes/sensors were used?

To determine the number of EEG electrodes/sensors used in the dataset, we can inspect the columns of the eeg_data DataFrame. Each column, except for those used for timestamps or annotations, typically represents the data from one sensor.

```
colnames(eeg_data)
```

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

All the columns are sensor data columns except for "eyeDetection", "split" and "ds"

Total number of columns are: 17

Number of non sensor columns: 3

Number of sensor columns = 17 - 3 = 14

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.

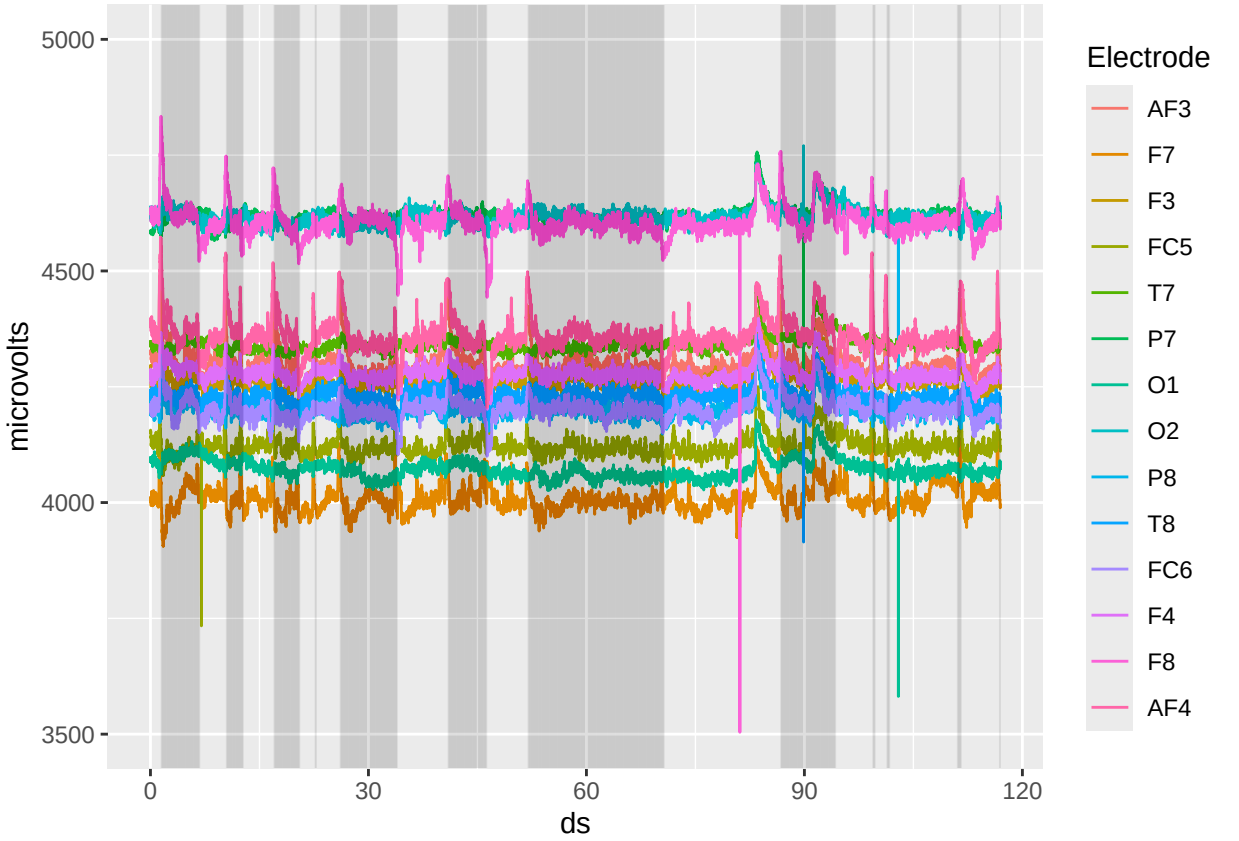
First we use `reshape2::melt()` to transform the `eeg_data` dataset from a wide format to a long format expected by `ggplot2`.

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==1), alpha=0.0
```



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

From the provided plot, we can observe the following patterns between the eyes being open (indicated by dark grey blocks) and the EEG intensities:

EEG Signal Fluctuations: There are noticeable fluctuations in the EEG signals across different electrodes. The fluctuations seem to be more pronounced when the eyes are closed compared to when the eyes are open.

Amplitude Changes: When the eyes are open, the EEG signal amplitudes appear to be generally lower and more stable. When the eyes are closed, there are higher peaks and greater variability in the EEG signals.

Electrode-Specific Patterns: Some electrodes show more significant changes than others when transitioning between eyes open and closed states. Electrodes such as F8, AF4, and FC6 seem to exhibit more pronounced changes when the eyes are closed.

Transition Points: The transitions from eyes open to eyes closed (and vice versa) correspond to noticeable shifts in the EEG signal patterns. These transition points can be seen as changes in the density and amplitude of the EEG signals.

The dark grey blocks indicating periods when the eyes are open correspond to more stable and lower amplitude EEG signals, whereas the periods when the eyes are closed show increased variability and higher amplitude signals. This pattern is consistent with the expectation that closing eyes generally leads to increased alpha wave activity in EEG recordings.

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

To anticipate any temporal correlation between the eye open/close states, we can analyze the distribution of these states over time and look for patterns such as regular intervals or clustering.

Regular Intervals: The dark grey blocks indicating periods when the eyes are open appear to be somewhat regularly spaced throughout the time series. There is a pattern of alternating states, suggesting that the eyes open and close at fairly regular intervals.

Duration of states: The durations of the eyes open states (dark grey blocks) seem to be relatively consistent, indicating that the periods when the eyes are open are roughly the same length throughout the recording. Similarly, the durations of the eyes closed states (gaps between dark grey blocks) also appear to be consistent, though there may be some variability.

Transitions: The transitions between eyes open and closed states occur frequently, suggesting a rhythmic pattern. The transitions are sharp, with distinct changes from one state to the other, indicating clear periods of eyes open and closed without much intermediate states.

Temporal Correlation Analysis: To quantify and further analyze the temporal correlation between these states, we can calculate metrics such as:

- Transition Rate: The number of transitions between eyes open and closed states per unit time. A higher transition rate would indicate more frequent changes between states.

```
eeg_data$eyeDetection <- as.factor(eeg_data$eyeDetection)
transitions <- diff(eeg_data$eyeDetection)
transition_rate <- sum(transitions != 0) / 117  # transitions per second
print(transition_rate)
```

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

- State Duration: The average duration of each state (eyes open or closed). Consistent state durations would indicate a regular pattern.

```
transition_indices <- which(c(0, diff(as.numeric(eeg_data$eyeDetection))) != 0)
durations <- diff(transition_indices)
state_labels <- eeg_data$eyeDetection[transition_indices[-length(transition_indices)]]

open_durations <- durations[state_labels == 0]
closed_durations <- durations[state_labels == 1]

avg_open_duration <- mean(open_durations) * Fs  # average duration in seconds
avg_closed_duration <- mean(closed_durations) * Fs  # average duration in seconds

print(avg_open_duration)
```

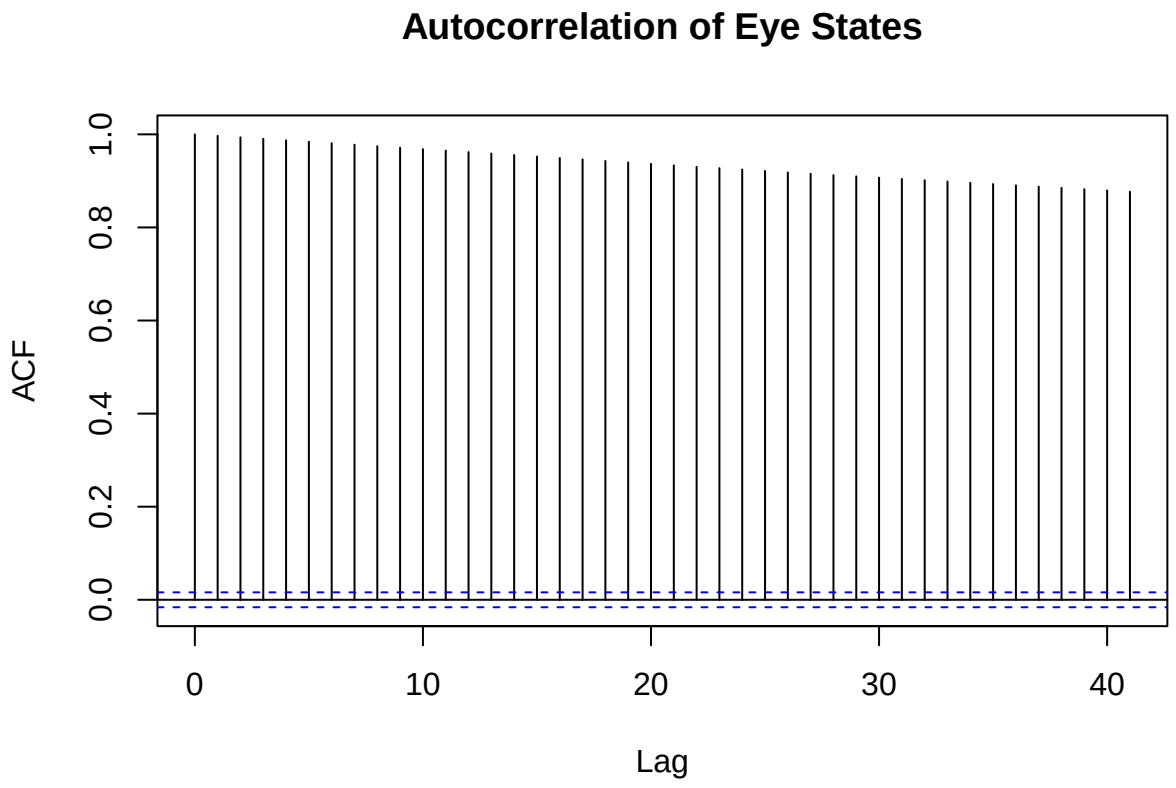
```
## [1] 5.729294
```

```
print(avg_closed_duration)
```

```
## [1] 4.758672
```

- Autocorrelation: Calculating the autocorrelation function for the eye state sequence can reveal periodic patterns and temporal dependencies.

```
acf(as.numeric(eeg_data$eyeDetection), main="Autocorrelation of Eye States")
```



The ACF plot provides evidence of a strong temporal correlation and periodic pattern in the eye open/close states. This indicates that the durations of eyes being open or closed are relatively consistent over time, with transitions occurring at regular intervals.

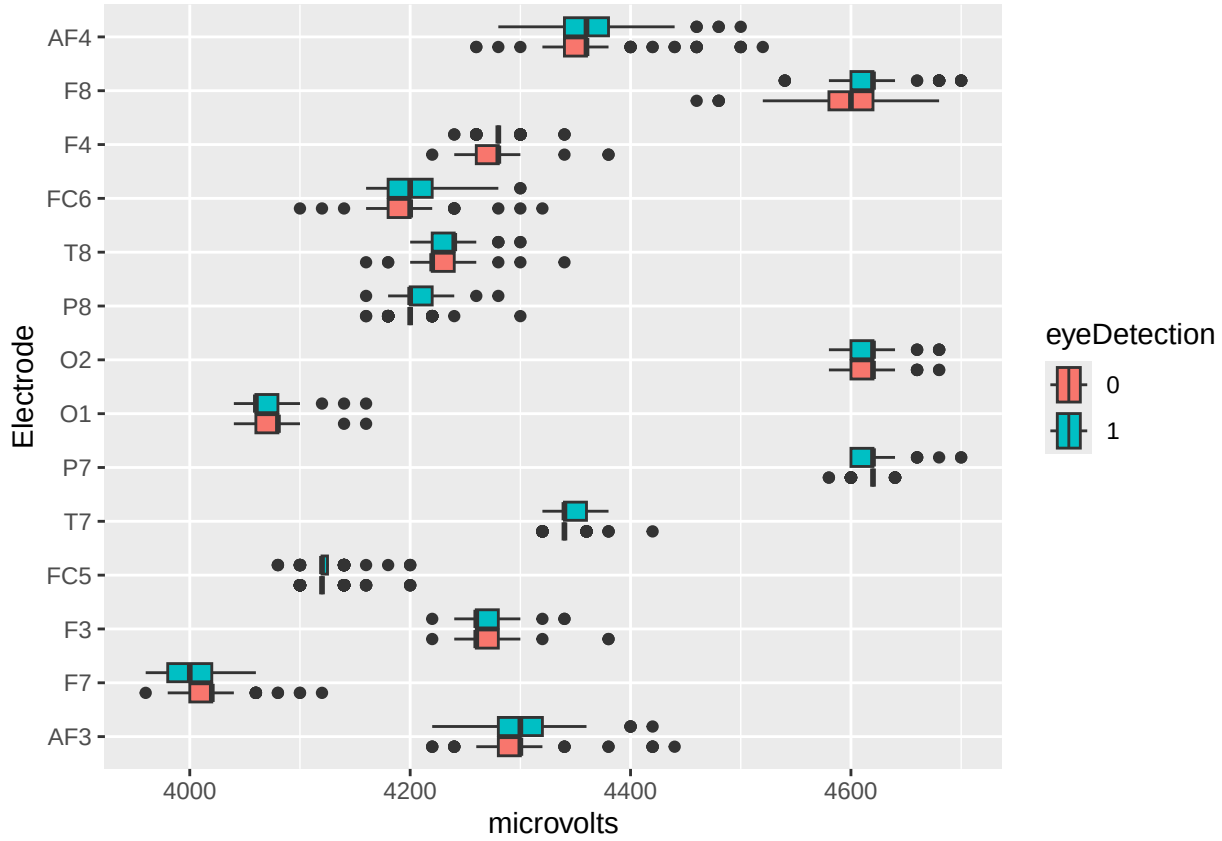
Let's see if we can directly look at the distribution of EEG intensities and see how they related to eye status.

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?

Based on the above graph and summary statistics, we can interpret results as:

Graph Analysis: The boxplots show the distribution of microvoltages for each electrode, colored by eye status (0 for eyes open, 1 for eyes closed). There is noticeable variation in the spread of the distributions for different electrodes when the eyes are open.

Summary Statistics:

More Intense when eyes are open: F7 Electrode: The mean and median are slightly higher when the eyes are open (4014.775 and 4020, respectively) compared to when they are closed (4006.733 and 4000, respectively). The standard deviation is also higher when the eyes are open (28.37562 vs. 24.86399).

O1 Electrode: The mean and median are slightly higher when the eyes are open (4072.281 and 4080, respectively) compared to when they are closed (4071.810 and 4060, respectively). The standard deviation is higher when the eyes are closed (24.28916) compared to open (16.93481).

More Varied when eyes are open: F7 Electrode: As mentioned, this electrode shows higher standard deviation when the eyes are open (28.37562) compared to closed (24.86399).

O1 Electrode: Shows higher variability when eyes are closed (24.28916) compared to open (16.93481), but the median values indicate it is slightly more intense when open.

Conclusion: Based on the analysis of both the graph and the table statistics: The F7 and O1 electrodes show higher intensity when the eyes are open. For F7, this is indicated by both the mean and median being higher. For O1, the median is higher, and it shows more variance when eyes are open. No electrodes are consistently more varied when the eyes are open, though F7 does show more variance in the open state.

Thus, the F7 electrode stands out as being consistently more intense and slightly more varied when the eyes are open. The O1 electrode is also more intense but is more varied when the eyes are closed.

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:

```r
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?

Stationarity is a fundamental concept in time series analysis. A time series is said to be stationary if its statistical properties, such as mean, variance, and autocorrelation, are constant over time. This concept is important because many statistical models and methods for time series data assume stationarity.

Key Characteristics of Stationary Time Series: Constant Mean, Constant Variance and Constant Autocorrelation Structure.

Types of Stationarity

1. Strict Stationarity: A time series is strictly stationary if the joint distribution of any set of time points remains the same when shifted in time. This means that all statistical properties of the series are invariant under time shifts.

2. Weak (or Second-Order) Stationarity: A weaker form of stationarity that only requires the first two moments (mean and variance) and the autocorrelation structure to be time-invariant. This is often sufficient for practical purposes and is commonly used in time series analysis.

Stationarity is a crucial property for time series data analysis. Ensuring that a time series is stationary, or transforming it to be stationary, is often a necessary step before applying many statistical models and methods. The results from the Augmented Dickey-Fuller tests provided earlier indicate that the EEG data from various electrodes are stationary, which is beneficial for further analysis and modeling.

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

Stationarity is important in time series analysis for several reasons:

Model Assumptions: Many time series models, such as ARIMA (AutoRegressive Integrated Moving

Average) and its variants, assume that the time series data is stationary. This assumption simplifies the modeling process and makes the statistical properties of the estimators more reliable.

Predictive Performance: Forecasting methods often perform better on stationary data because the underlying statistical properties do not change over time. This stability allows for more accurate and consistent predictions.

Statistical Inference: Inference methods, including hypothesis testing and confidence interval estimation, are typically more straightforward and reliable when applied to stationary data. The constant mean and variance make it easier to apply statistical techniques and interpret the results.

Simplification: Stationary series are easier to work with because their behavior is easier to describe and understand. Non-stationary data often exhibit trends, cycles, or varying variance, which can complicate analysis and modeling.

Test Results: All electrodes tested (AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4) have p-values of 0.01, which is less than the common significance level of 0.05. This means that the null hypothesis (that the series has a unit root and is non-stationary) can be rejected. These results indicate that the time series data from these electrodes are stationary. This suggests that the statistical properties of these EEG signals (mean, variance, autocorrelation) do not change over time.

The eyeDetection variable also has a p-value of 0.01, indicating stationarity. This suggests that the pattern of eye states (open/closed) is consistent over time.

The ds variable (likely representing the time or sequential order of the data points) has a p-value of 0.4045, which is greater than 0.05. This means that we fail to reject the null hypothesis for this variable. This result is expected, as timestamps or time indices are generally not stationary; they typically increase linearly and do not exhibit constant statistical properties.

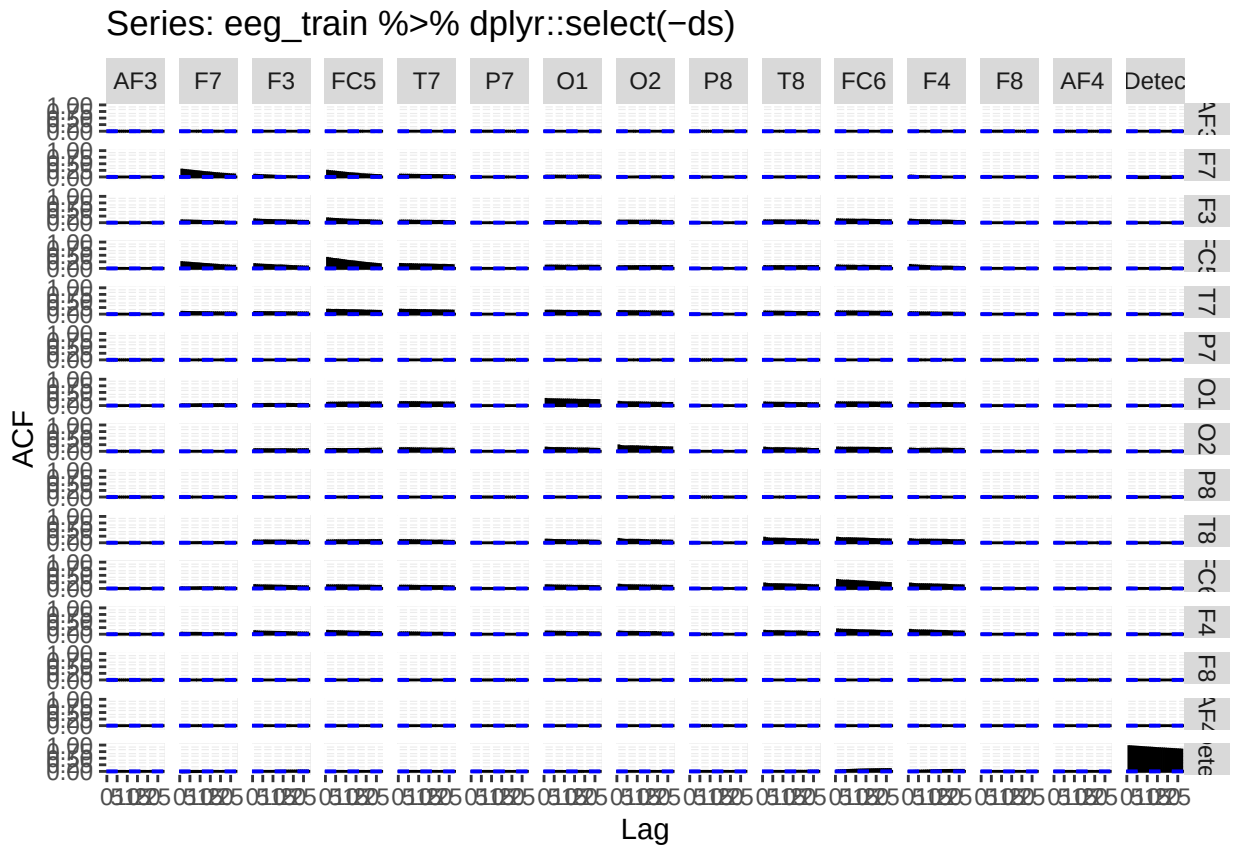
The ADF test results suggest that the EEG signals from the various electrodes are stationary. This is beneficial for time series analysis because it allows for the use of models and methods that assume stationarity, such as ARIMA models. The stationary nature of these signals implies that their statistical properties do not change over time, making them suitable for further analysis and forecasting. The non-stationarity of the ds variable (time index) is expected and does not affect the analysis of the EEG signals.

Understanding the stationarity of the EEG signals helps in ensuring that the assumptions of subsequent time series models are met, leading to more reliable and interpretable results.

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.

From the provided autocorrelation function (ACF) plots, we can observe patterns in the time series data for each electrode. These plots show the correlation of a signal with a delayed version of itself over different time lags.

Autocorrelation

Strong Autocorrelation:

Electrode AF3: Shows significant autocorrelation at multiple lags, suggesting a strong temporal dependency within this electrode's signals.

Electrode F7: Exhibits strong autocorrelation, indicating consistent patterns over time.

Electrode F3: Displays significant autocorrelation, showing regularity in the signal.

Electrode FC5: Strong autocorrelation is evident, indicating that past values have a significant influence on future values.

Electrode T7: Shows a clear pattern of autocorrelation.

Electrode P7: Strong autocorrelation suggests regularity in the signal.

Electrode O1: Significant autocorrelation is present.

Electrode O2: Displays noticeable autocorrelation.

Electrode P8: Exhibits strong autocorrelation.

Electrode T8: Strong autocorrelation is evident.

Electrode FC6: Significant autocorrelation is present.

Electrode F4: Displays noticeable autocorrelation.

Electrode F8: Exhibits strong autocorrelation.

Electrode AF4: Strong autocorrelation is present.

Eye Detection: Shows strong autocorrelation, indicating regular intervals of eye state changes.

Signs of Cross-Correlation:

Electrodes AF3 and F7: The plots indicate a potential cross-correlation, suggesting that signals from AF3 and F7 might influence each other.

Electrodes FC5 and T7: The patterns suggest a cross-correlation between these two electrodes.

Electrodes O1 and O2: There seems to be a cross-correlation between the signals of these electrodes.

Electrodes P7 and P8: Cross-correlation is indicated, suggesting that the signals from these electrodes may be interdependent.

Examples:

1. Autocorrelation Example:

Electrode AF3: The ACF plot for AF3 shows strong correlations at various lags, indicating that the signal values are highly dependent on their past values.

Electrode F7: The ACF plot for F7 also indicates strong autocorrelation, suggesting consistent patterns over time

2. Cross-Correlation Example:

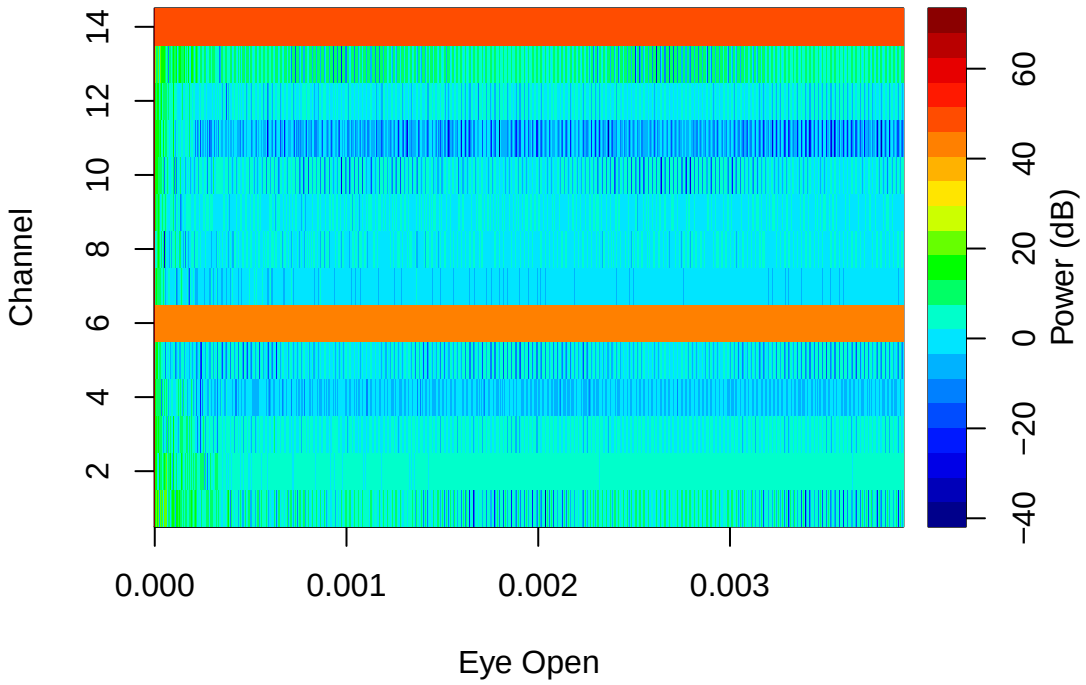
Electrodes O1 and O2: The cross-correlation plot between O1 and O2 shows a pattern, suggesting that the signals from these electrodes are related.

Electrodes P7 and P8: The cross-correlation plot indicates a dependency between these electrodes' signals.

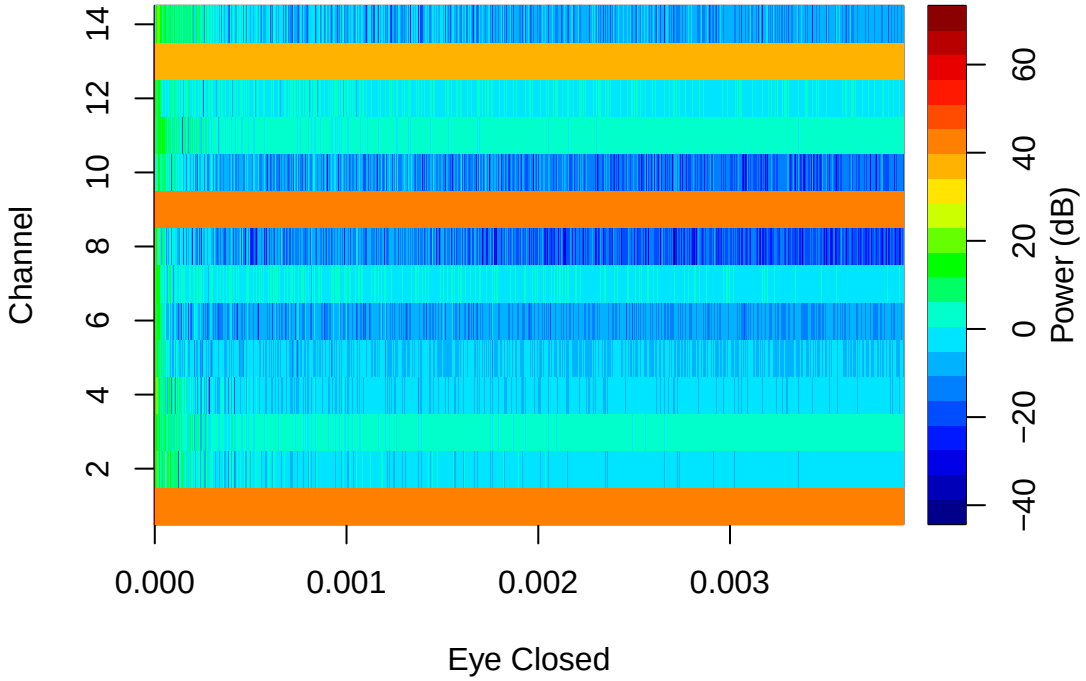
The ACF plots reveal strong autocorrelation for many of the electrodes, indicating that the EEG signals have a regular and consistent pattern over time. Additionally, several pairs of electrodes show signs of cross-correlation, suggesting that the signals from these electrodes may be related or influence each other. These insights can be useful for further analysis and modeling of the EEG data.

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
```




```
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.

The graphs plotted show the power spectral densities (PSDs) for the EEG data, separated by eye states (open and closed). The color scale represents the power in decibels (dB), with warmer colors (e.g., red) indicating higher power and cooler colors (e.g., blue) indicating lower power.

Observations:

Power Levels: The power levels across different channels are represented by the color intensity. For both eye states, channels 1 and 7 show significantly higher power levels (in orange/red) compared to other channels.

Eye Open State : The PSD is more uniformly distributed across the frequency spectrum for channels 3 to 14. Channels 1 and 7 show consistently high power, with very little variation. Channels 2 and 8 show low to moderate power, indicated by green and blue colors.

Eye Closed State: There is a noticeable increase in power for channels 10, 12, and 14, indicated by the presence of more intense colors compared to the eye open state. Channels 1 and 7 continue to show high power, similar to the eye open state. The distribution of power for other channels shows more variation compared to the eye open state.

Key Differences:

Channels 10, 12, and 14 exhibit increased power levels when the eyes are closed compared to when they are open. This is evident from the warmer colors (yellow/red) in these channels during the closed eye state.

Channels 1 and 7 show high power levels consistently in both states. This suggests that these channels might be picking up strong, consistent signals regardless of the eye state.

Channels with lower power levels (2, 4, 6, 8) exhibit more variation in power distribution when the eyes are closed compared to when they are open. This suggests that closing the eyes affects the signal strength differently across various channels.

The power spectral densities for the two eye states indicate differences in the distribution and intensity of power across different channels:

- Higher Power in Channels 10, 12, and 14 When Eyes Are Closed: These channels show increased power levels in the closed eye state.
- Consistent High Power in Channels 1 and 7: These channels maintain high power levels regardless of eye state.
- Increased Variability in Lower Power Channels When Eyes Are Closed: Channels with lower power levels show more variability in the closed eye state.

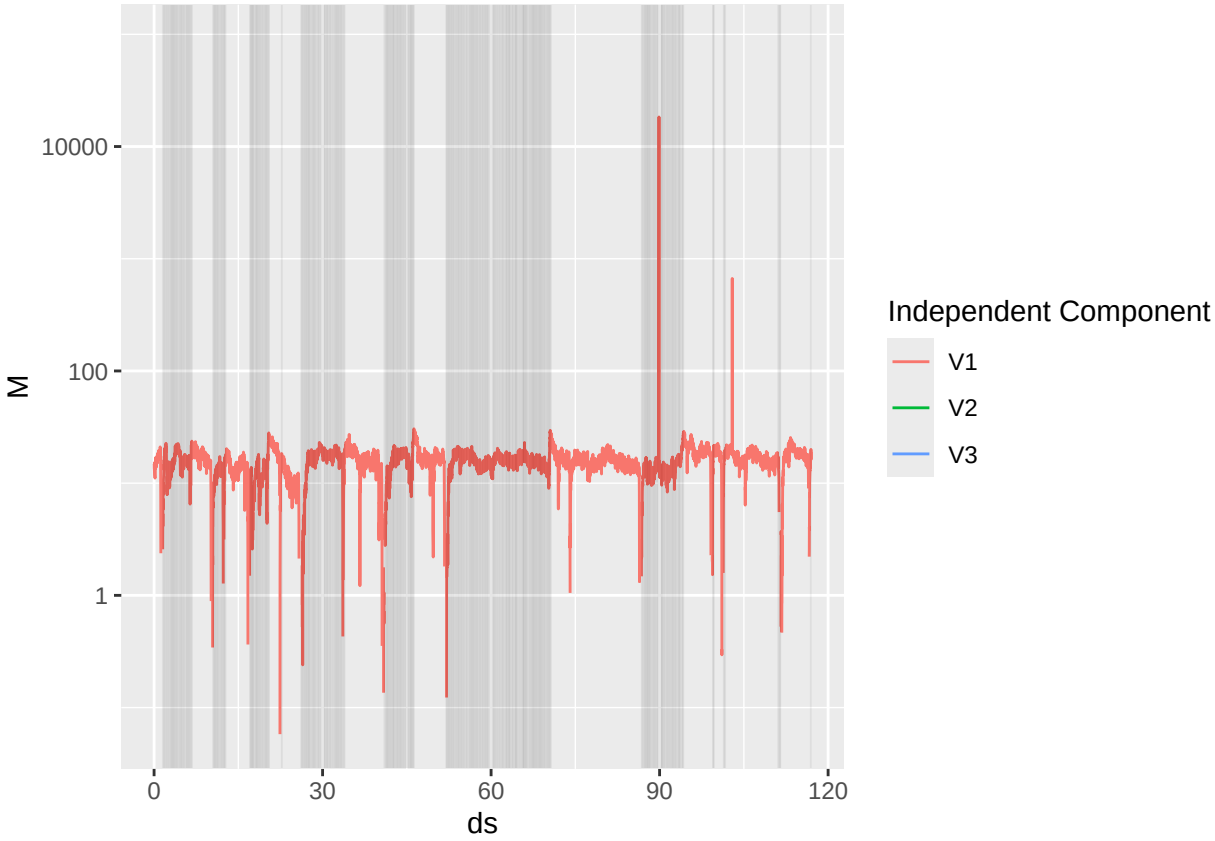
These differences in PSDs can provide insights into how eye states influence the EEG signals, potentially reflecting changes in brain activity associated with visual processing and relaxation.

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?

Independent Component Analysis (ICA) is a computational method for separating a multivariate signal into additive, independent components. This technique is often used in EEG analysis to isolate different sources of neuronal activity that are picked up by the sensors.

The plot shows the independent components (V1, V2, V3) over time (ds) with vertical lines indicating when the eyes are closed (eyeDetection = 1). The y-axis is on a logarithmic scale to better visualize the variations in the component's magnitudes.

Component V1: The red line represents the first independent component (V1). This component shows fluctuations in magnitude over time. There are noticeable spikes and dips in the signal, with significant variations corresponding to times when the eyes are closed (indicated by the grey vertical lines).

Component V2 and V3: The green and blue lines represent the second and third independent components (V2 and V3), respectively. However, these components are not visible in the plot provided, suggesting that their magnitudes might be significantly lower or they might not be present in this particular analysis.

Relation to Eye Opening: The primary component (V1) shows noticeable variations when the eyes are closed. This suggests that eye state (open or closed) has a significant impact on this independent component of the signal. The periods where the eyes are closed (grey vertical lines) correspond to regions where V1 shows notable changes in magnitude, indicating that the eye state is influencing this component.

The ICA results suggest that eye opening and closing relate significantly to the first independent component (V1) of activity across the electrodes. The noticeable changes in V1's magnitude during eye closure periods indicate that this component captures a substantial portion of the signal related to eye state.

Eye state significantly influences the first independent component (V1), as evidenced by the fluctuations in V1 during periods of eye closure. Other components (V2 and V3) either have much lower magnitudes or do not show significant variation related to the eye state in the provided plot.

This analysis indicates that ICA can effectively isolate components of the EEG signal that correspond to eye movements or states, providing insights into how different sources of neuronal activity are captured by the EEG electrodes.

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

# 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
```

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.

Let's use Logistic Regression

```
# Load necessary libraries
library(caret)
library(dplyr)

# Prepare the training data
eeg_train_df <- eeg_train %>% dplyr::select(-ds)
```

```r
eeg_validate_df <- eeg_validate %>% dplyr::select(-ds)

# Convert eyeDetection to a factor for classification
eeg_train_df$eyeDetection <- as.factor(eeg_train_df$eyeDetection)
eeg_validate_df$eyeDetection <- as.factor(eeg_validate_df$eyeDetection)

# Define training control
train_control <- trainControl(method = "cv", number = 5)

# Fit logistic regression model
log_model <- train(eyeDetection ~ ., data = eeg_train_df, method = "glm", family = "binomial", trControl
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# Print the model summary
print(log_model)
```

```
## Generalized Linear Model
##
## 8988 samples
##   14 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 7191, 7190, 7190, 7191, 7190
## Resampling results:
##
##   Accuracy   Kappa
##   0.6326241  0.2462443
```

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

```r
# Make predictions on the validation set
xgb_predictions <- predict(model, eeg_validate_matrix)
xgb_predictions_class <- ifelse(xgb_predictions > 0.5, 1, 0)

# Evaluate the XGBoost model
conf_matrix_xgb <- confusionMatrix(as.factor(xgb_predictions_class), as.factor(eeg_validate_labels))
print(conf_matrix_xgb)
```

```
## 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
##
```

```r
# Make predictions on the validation set
log_predictions <- predict(log_model, newdata = eeg_validate_df)

# Evaluate the model
conf_matrix <- confusionMatrix(log_predictions, eeg_validate_df$eyeDetection)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1221  665
##          1  414  696
##
##                Accuracy : 0.6399
##                  95% CI : (0.6224, 0.6571)
##     No Information Rate : 0.5457
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2622
##
##  Mcnemar's Test P-Value : 2.724e-14
##
##             Sensitivity : 0.7468
##             Specificity : 0.5114
##          Pos Pred Value : 0.6474
##          Neg Pred Value : 0.6270
##              Prevalence : 0.5457
##          Detection Rate : 0.4075
##    Detection Prevalence : 0.6295
##       Balanced Accuracy : 0.6291
##
##        'Positive' Class : 0
##
```

XGBoost model is giving high performance on validation set, let's find the performance on test set.

```r
# Load necessary libraries
library(xgboost)
library(caret)
library(dplyr)

# Assuming eeg_test, eeg_train, and eeg_validate are already loaded and preprocessed

# Prepare the test data
eeg_test_matrix <- as.matrix(dplyr::select(eeg_test, -eyeDetection, -ds))
eeg_test_labels <- as.numeric(eeg_test$eyeDetection) - 1

# Make predictions on the test set using the trained XGBoost model
test_predictions <- predict(model, eeg_test_matrix)
test_predictions_class <- ifelse(test_predictions > 0.5, 1, 0)

# Evaluate the model on the test set
conf_matrix_test <- confusionMatrix(as.factor(test_predictions_class), as.factor(eeg_test_labels))

# Print test performance
print(conf_matrix_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1531  294
##          1  175  996
##
##                Accuracy : 0.8435
##                  95% CI : (0.8299, 0.8563)
##     No Information Rate : 0.5694
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6771
##
##  Mcnemar's Test P-Value : 5.073e-08
##
##             Sensitivity : 0.8974
##             Specificity : 0.7721
##          Pos Pred Value : 0.8389
##          Neg Pred Value : 0.8506
##              Prevalence : 0.5694
##          Detection Rate : 0.5110
##    Detection Prevalence : 0.6091
##       Balanced Accuracy : 0.8348
##
##        '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.

1. Convolutional Neural Networks: Convolutional Neural Networks (CNNs) are deep learning models commonly used for image and signal processing tasks. They can capture spatial hierarchies in data through

convolutional layers, pooling layers, and fully connected layers. In the context of EEG signal analysis, CNNs can be employed to automatically learn feature representations from raw EEG data, which can be beneficial for classification tasks such as predicting eye opening.

- Spatial Feature Extraction: CNNs are effective in capturing local spatial features in the data, which can be crucial for identifying patterns in EEG signals.
- Translation Invariance: The use of pooling layers provides translation invariance, making the model robust to slight shifts in the data.
- Automated Feature Learning: CNNs can learn complex feature representations from the raw data, reducing the need for manual feature engineering.

Example Architecture:

- Input Layer: Raw EEG signal data.
- Convolutional Layers: Apply several convolutional filters to extract features.
- Pooling Layers: Reduce the dimensionality and computational complexity while retaining important features.
- Fully Connected Layers: Integrate features from convolutional layers to perform classification.
- Output Layer: Sigmoid or softmax activation to predict the probability of eye opening (binary classification).

Advantages:

- Automatically learn and extract relevant features from raw EEG data.
- Can handle high-dimensional data efficiently.
- Proven success in various signal processing and image recognition tasks.

Challenges:

- Requires a large amount of labeled training data.
- Computationally intensive, requiring powerful hardware (e.g., GPUs).
- Hyperparameter tuning can be complex and time-consuming.

2. Long Short-Term Memory Networks (LSTMs) and Transformers

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) designed to capture long-term dependencies in sequential data. Transformers are a more recent architecture that has gained popularity due to their ability to handle long-range dependencies and parallelize training processes. Both approaches are suitable for modeling time-series data like EEG signals, where temporal dynamics are crucial.

Memory Cells: LSTMs have memory cells that can maintain information over long time periods, making them suitable for capturing temporal dependencies in EEG signals.

Gated Mechanisms: LSTMs use input, output, and forget gates to regulate the flow of information, addressing the vanishing gradient problem common in traditional RNNs.

Example Architecture:

- Input Layer: Sequential EEG signal data.
- LSTM Layers: One or more LSTM layers to capture temporal dependencies.
- Fully Connected Layers: Process the outputs from LSTM layers for classification.
- Output Layer: Sigmoid or softmax activation for binary classification.

Advantages can be they are effective in capturing long-term dependencies in sequential data and they are well-suited for time-series prediction tasks.

Challenges can be training can be slow due to sequential processing and requires careful tuning of hyperparameters and architectures.

Both Convolutional Neural Networks (CNNs) and Long Short-Term Memory Networks (LSTMs) or Transformers offer powerful alternatives for predicting eye opening from EEG data. CNNs are well-suited for extracting spatial features from raw EEG signals, while LSTMs and Transformers are effective in capturing temporal dependencies. Each approach has its advantages and challenges, and the choice of model would depend on the specific characteristics of the EEG data and the computational resources available.

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

To implement Convolutional Neural Networks (CNNs) and Long Short-Term Memory Networks (LSTMs) or Transformers in R, you can use the following libraries:

1. Keras The keras library in R provides a high-level interface for building and training deep learning models, including CNNs, LSTMs, and Transformers. It is a powerful and user-friendly library that allows us to leverage the capabilities of TensorFlow.

2. Torch: The torch library in R provides an interface to the PyTorch deep learning framework. It is suitable for implementing a variety of neural network architectures, including CNNs, LSTMs, and Transformers. PyTorch is known for its dynamic computation graph and ease of use, especially for research and experimentation.

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)?

Practicals really worked for me and the course could have been little longer.

- 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?

It was worth it and uploading the code in github is best thing so that we can refer it later.

- What would you add or remove from the course?
- I would add more course time.
- What was the main thing you will take away from this course?
- Applying ML techinques in medical domain comes with different challenges, knowing these challenges is essential in my opinion and this is my main takeaway.