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 <- 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?

$$Fs = \frac{117}{14980} \approx 0.007812$$

$$\text{samples per second} = \frac{1}{0.007812} \approx 128$$

1 How many EEG electrodes/sensors were used?

14 EEG electrodes/sensors were used

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.

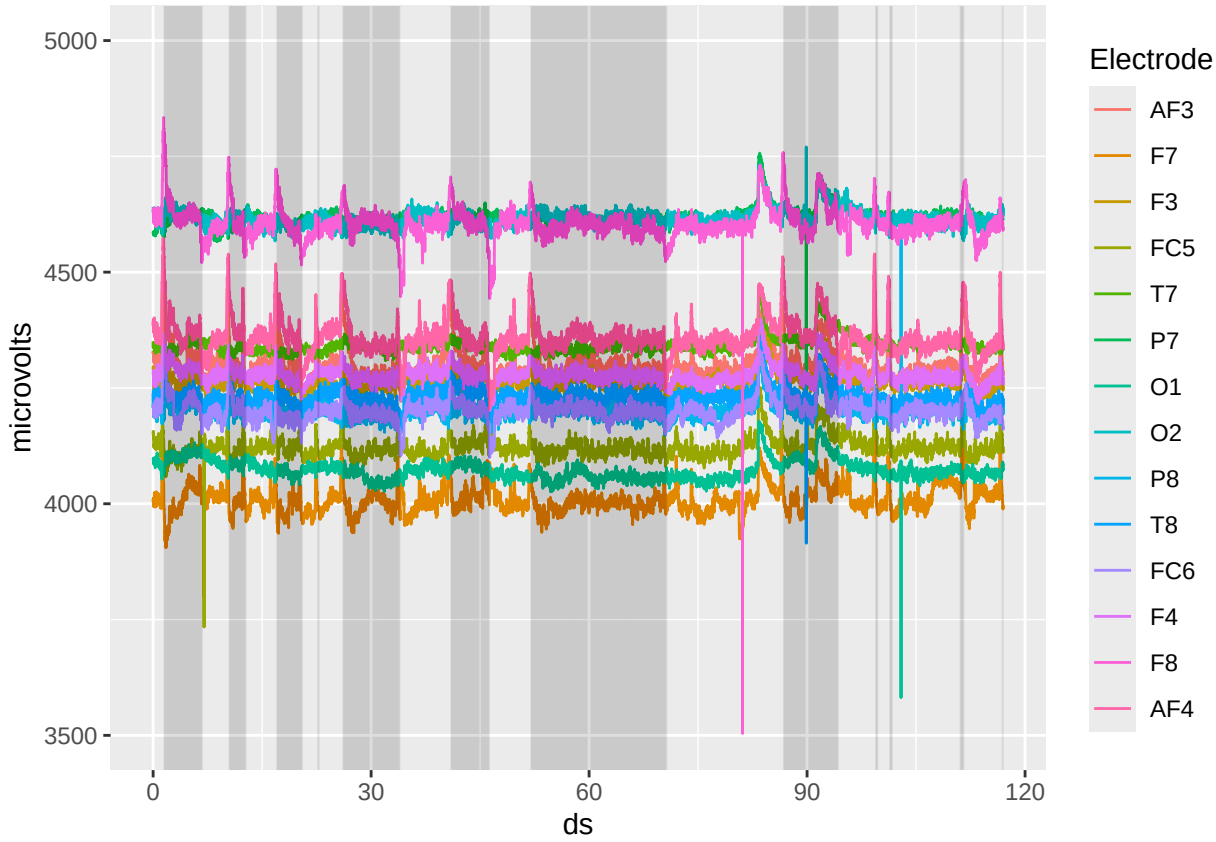
First we use `reshape2::melt()` to transform the `eeg_data` dataset from a wide format to a long format expected by `ggplot2`.

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?

I can see these patterns:

1. When the eyes are closed (dark gray block area), the EEG signal strength seems to increase at some electrodes.
2. These changes are more obvious at some electrodes, such as AF3, F7, F3, etc.
3. When the eyes are open (no dark gray block area), the EEG signal strength is relatively stable with relatively small fluctuations.

These patterns may indicate that changes in eye state will have an impact on the EEG signal, especially at specific electrode locations, the EEG signal strength changes with changes in eye state.

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

I can see the following:

1. Alternation of eye states:

The eye states alternate on the time axis, sometimes the eyes are open and sometimes they are closed. The dark grey blocks identify the time periods when the eyes are closed, and there are sometimes certain intervals between these blocks. This alternation shows that the eye states are not random, but have certain time patterns.

2. Time correlation:

The changes between the open and closed states of the eyes seem to have a certain periodicity, which may indicate that the eye states are correlated with time. This correlation may be related to the physiological rhythm of the person or external stimuli.

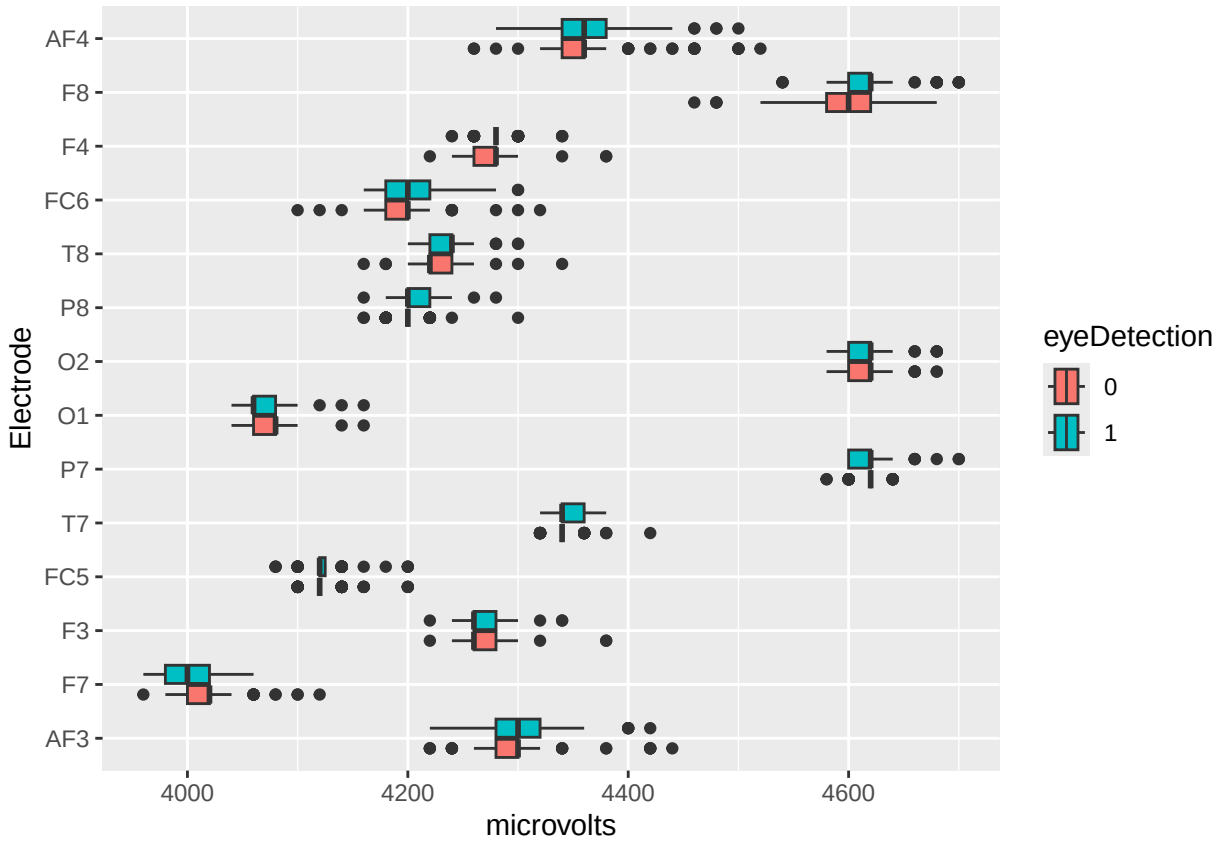
Let's see if we can directly look at the distribution of EEG intensities and see how they related to eye status.

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?

1. **AF3 electrode**:

- Eyes closed: mean 4294, SD 35.4
- Eyes open: mean 4305, SD 34.4
- The intensity of this electrode does not vary much between eyes open and closed, but is slightly higher with eyes open.

2. **F7 electrode**:

- Eyes closed: mean 4015, SD 28.4
- Eyes open: mean 4007, SD 24.9
- The intensity of this electrode is slightly higher with eyes closed.

3. **F3 electrode**:

- Eyes closed: mean 4268, SD 20.9
- Eyes open: mean 4269, SD 17.4
- The intensity of this electrode is almost the same when eyes open and closed, but is more variable when eyes closed.

Conclusion: Overall, the AF3 and F3 electrodes showed small changes in intensity when the eyes were open and closed, but the AF3 electrode was slightly stronger when the eyes were open. The F7 electrode was slightly stronger when the eyes were 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?

A time series is called stationary if its statistical properties (such as mean, variance, autocorrelation, etc.) remain constant over time. In other words, the statistical properties of a stationary time series do not change over time.

6 Why are we interested in stationarity?

We focus on stationarity mainly because it plays many roles in time series analysis and modeling:

1. Ease of model construction and application

Most time series models assume that the time series is stationary. The theoretical basis and statistical properties of these models rely on the assumption of stationarity. If the time series is not stationary, the model may not fit the data correctly, resulting in inaccurate or unreliable forecasts.

2. Improve forecast accuracy

The statistical properties of a stationary time series remain unchanged over time, which means that past data can be used more reliably to predict the future.

3. Simplify the analysis process

Stationarity simplifies the analysis of time series because there is no need to consider changes in the statistical properties of the time series. The analysis process is simpler and more direct when applying various statistical methods and tools to stationary time series.

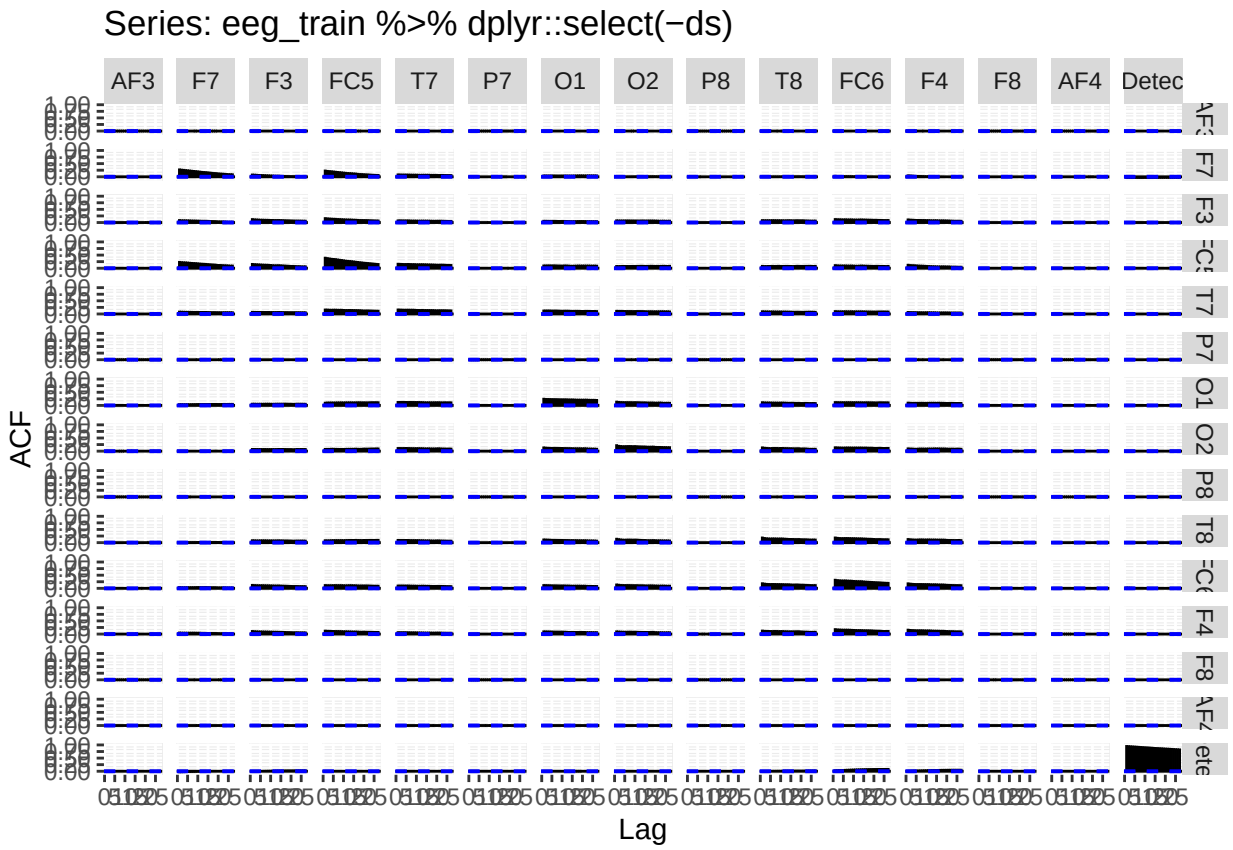
What do the results of these tests tell us? (ignoring the lack of multiple comparison correction...)

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.

Autocorrelation Example According to the graph, it can be observed that the following electrodes show strong autocorrelation:

- **AF3**: In the autocorrelation diagram, the autocorrelation coefficients of multiple lag times are significant, indicating that the electrode signal has strong autocorrelation at multiple lag times.
- **F7**: Similarly, the autocorrelation plot shows significant autocorrelation coefficients over multiple lag times.

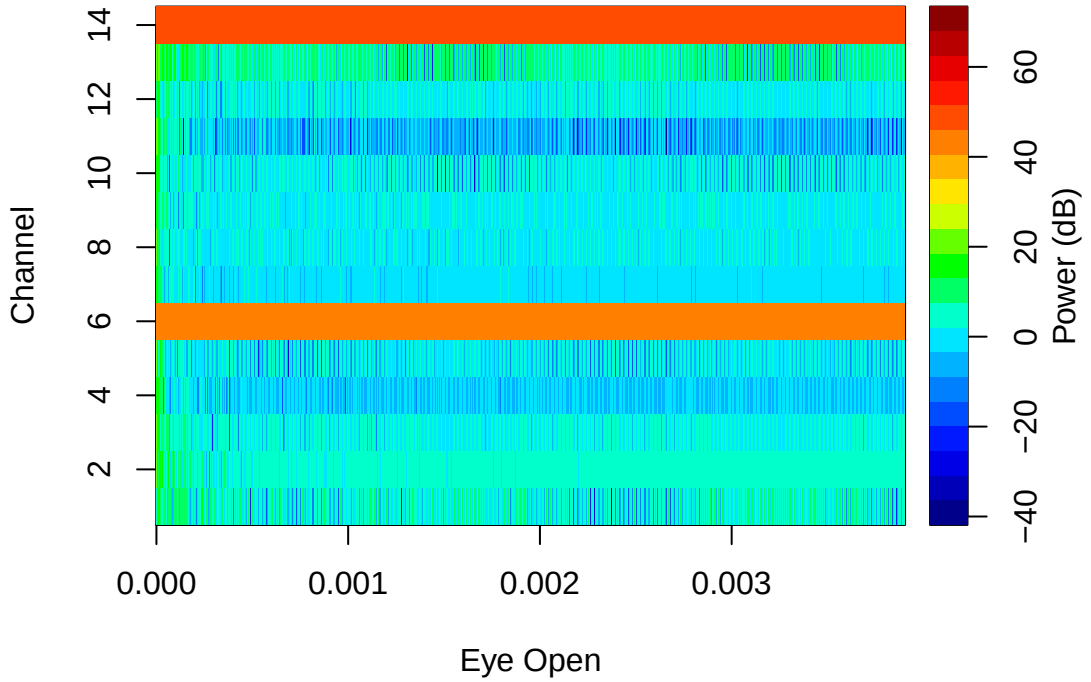
Cross-correlation Example In the plot off the diagonal, it can be observed that the following electrode pairs show significant cross-correlation:

- **AF3 and F7**: The cross-correlogram of these two electrodes shows that the cross-correlation coefficient is significant at certain lag times, indicating that there is a correlation between the signals of these two electrodes.
- **F3 and F7**: The cross-correlogram shows significant cross-correlation coefficients for these electrodes at multiple lag times.

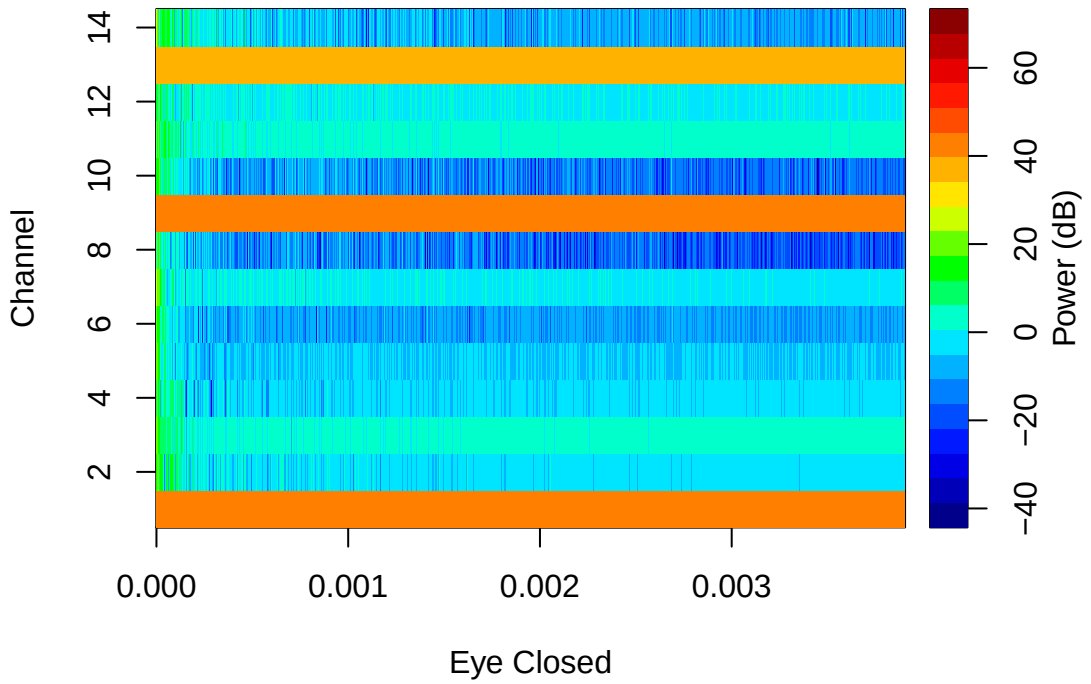
Frequency-Space We can also explore the data in frequency space by using a Fast Fourier Transform. After the FFT we can summarise the distributions of frequencies by their density across the power spectrum.

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.

1. **Overall energy distribution**:

- The power spectral density diagram when the eyes are open shows that the energy of the EEG signal is more concentrated in the higher frequency range.

- The power spectral density diagram when the eyes are closed shows that the energy of the EEG signal is more concentrated in the lower frequency range.

2. **Frequency component**:

- When the eyes are open, some electrodes exhibit higher power at specific frequencies (such as the high frequency band). This may be related to visual stimulation.
- When the eyes are closed, some electrodes exhibit higher power at lower frequencies (e.g. alpha band 8-12 Hz). This is consistent with EEG activity during states of relaxation or closed-eyes meditation.

3. **Differences between different electrodes**:

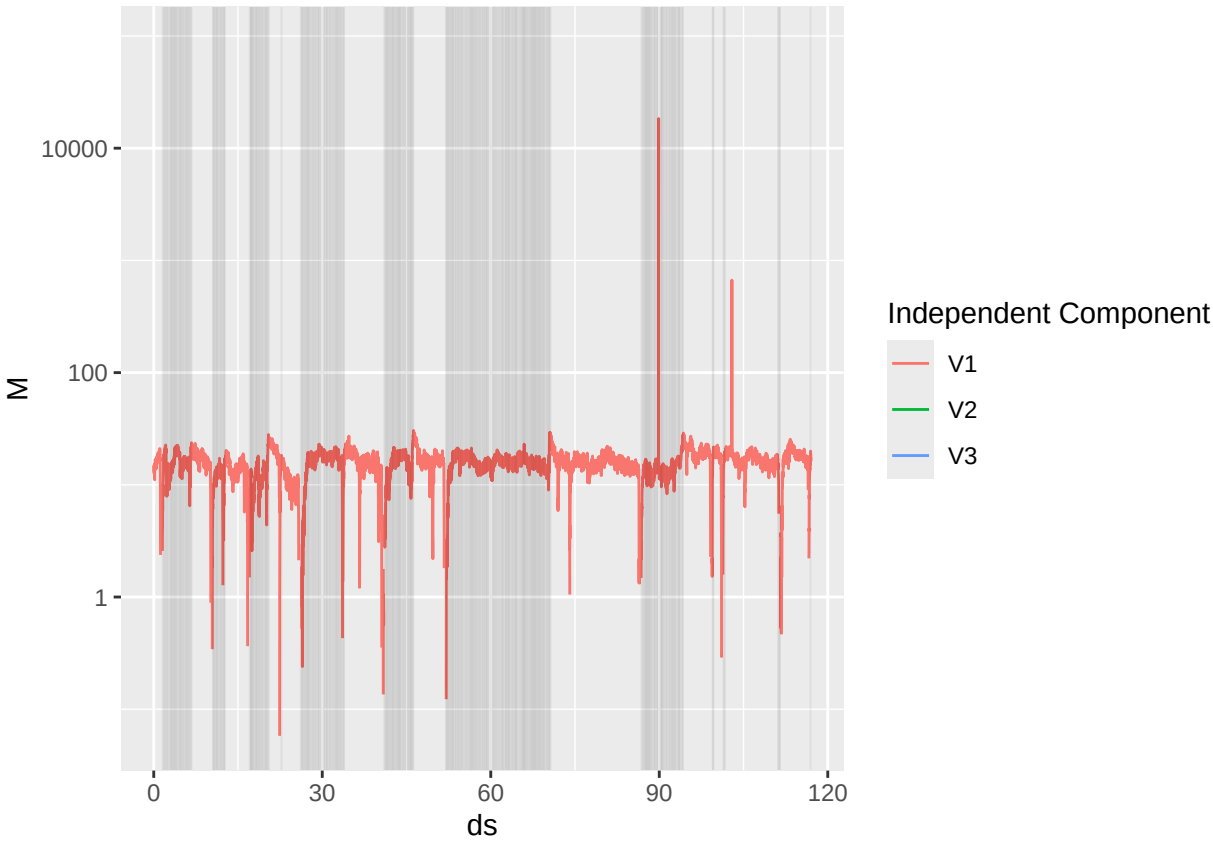
- The power spectral density of different electrodes in different frequency ranges varies greatly. Some electrodes had a significant increase in low-frequency band power when the eyes were closed and a significant increase in high-frequency band power when the eyes were open.

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 V1**: The activity of independent component V1 increases significantly when the eyes are closed (dark gray block), indicating that eye closure is related to the neural activity of this component.
- **Independent component V2**: The activity of independent component V2 is relatively stable when the eyes are open, but shows significant changes when the eyes are closed.

These results indicate that changes in eye state may induce changes in neural activity of specific independent components, supporting the hypothesis of a correlation between eye state and independent component activity.

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.

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

train_data <- data.frame(eeg_train_matrix)
train_data$eyeDetection <- eeg_train_labels

validate_data <- data.frame(eeg_validate_matrix)
validate_data$eyeDetection <- eeg_validate_labels

train_control <- trainControl(method = "cv", number = 10)
logistic_model <- train(eyeDetection ~ ., data = train_data, method = "glm", family = "binomial", trCon
```

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

```
print(logistic_model)
```

```
## Generalized Linear Model
```

```
##
## 8988 samples
##   14 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 8089, 8090, 8089, 8090, 8089, 8089, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.6429705  0.2666327
```

```
predictions <- predict(logistic_model, newdata = validate_data)

confusionMatrix(predictions, validate_data$eyeDetection)
```

```
## 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
##
```

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

```
eeg_test_matrix <- as.matrix(dplyr::select(eeg_test, -eyeDetection, -ds))
eeg_test_labels <- as.factor(eeg_test$eyeDetection)

xgboost_predictions <- predict(model, eeg_test_matrix)
xgboost_predictions <- ifelse(xgboost_predictions > 0.5, 1, 0)
xgboost_predictions <- factor(xgboost_predictions, levels = c(0, 1))
xgboost_cm <- confusionMatrix(xgboost_predictions, eeg_test_labels)
print(xgboost_cm)
```

```
## 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
##
```

```
logistic_predictions <- predict(logistic_model, newdata = eeg_test_matrix)
logistic_cm <- confusionMatrix(logistic_predictions, eeg_test_labels)
print(logistic_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1278  665
##          1  428  625
##
##                Accuracy : 0.6352
##                  95% CI : (0.6177, 0.6524)
##     No Information Rate : 0.5694
##     P-Value [Acc > NIR] : 1.378e-13
##
##                   Kappa : 0.239
##
##  Mcnemar's Test P-Value : 9.441e-13
##
##             Sensitivity : 0.7491
##             Specificity : 0.4845
##          Pos Pred Value : 0.6577
##          Neg Pred Value : 0.5935
##              Prevalence : 0.5694
```

```
##          Detection Rate : 0.4266
##    Detection Prevalence : 0.6485
##       Balanced Accuracy : 0.6168
##
##        'Positive' Class : 0
##
```

```
print("XGBoost Confusion Matrix:")
```

```
## [1] "XGBoost Confusion Matrix:"
```

```
print(xgboost_cm)
```

```
## 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
##
```

```
print("Logistic Regression Confusion Matrix:")
```

```
## [1] "Logistic Regression Confusion Matrix:"
```

```
print(logistic_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1278  665
##          1  428  625
##
##                Accuracy : 0.6352
##                  95% CI : (0.6177, 0.6524)
##     No Information Rate : 0.5694
```

```
##     P-Value [Acc > NIR] : 1.378e-13
##
##                   Kappa : 0.239
##
##  Mcnemar's Test P-Value : 9.441e-13
##
##             Sensitivity : 0.7491
##             Specificity : 0.4845
##          Pos Pred Value : 0.6577
##          Neg Pred Value : 0.5935
##              Prevalence : 0.5694
##          Detection Rate : 0.4266
##    Detection Prevalence : 0.6485
##       Balanced Accuracy : 0.6168
##
##        'Positive' Class : 0
##
```

```r
xgboost_accuracy <- xgboost_cm$overall['Accuracy']
logistic_accuracy <- logistic_cm$overall['Accuracy']

if (xgboost_accuracy > logistic_accuracy) {
  print(paste("Best model is XGBoost with accuracy:", xgboost_accuracy))
} else {
  print(paste("Best model is Logistic Regression with accuracy:", logistic_accuracy))
}
```

```
## [1] "Best model is XGBoost with accuracy: 0.843457943925234"
```

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. Support Vector Machine (SVM)

Support Vector Machine is a supervised learning model suitable for classification and regression analysis. SVM separates different categories of data by finding an optimal hyperplane in high-dimensional space. For EEG data, SVM can distinguish eye states by processing nonlinear features.

Advantages: SVM performs well in high-dimensional space and is suitable for processing complex EEG signals. Disadvantages: The computational complexity is high, especially when processing large-scale data sets.

2. Random Forest

Random Forest is an ensemble learning method that improves classification accuracy and robustness by building multiple decision trees and combining their predictions. For EEG data, random forest can capture complex patterns in the data by integrating multiple decision trees.

Advantages: Random forest can handle high-dimensional data and performs well in handling missing values and noise. Disadvantages: The training and prediction time of the random forest model is long, especially when the number of trees is large.

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

- **e1071**
- **randomForest**

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 thought the way the course was taught worked for me, especially the assignments, which were very useful for learning how to work with real data with real code.

- 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 definitely worth it, I learned a lot.

- What would you add or remove from the course?

I think more content where you code by yourself according to prompts would be more useful as practice

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

How to work with real medical data in R.