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 (https://www.xquartz.org/) 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 (http://ehrai.com/su/pdf/aihls2013.pdf).

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?

The samples per second is 128

```
samples <- nrow(eeg_data) / 117

cat("Samples:", samples, "\n")
```

```
## Samples: 128.0342
```

1 How many EEG electrodes/sensors were used?

The number of EEG electrodes/sensors used are 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.

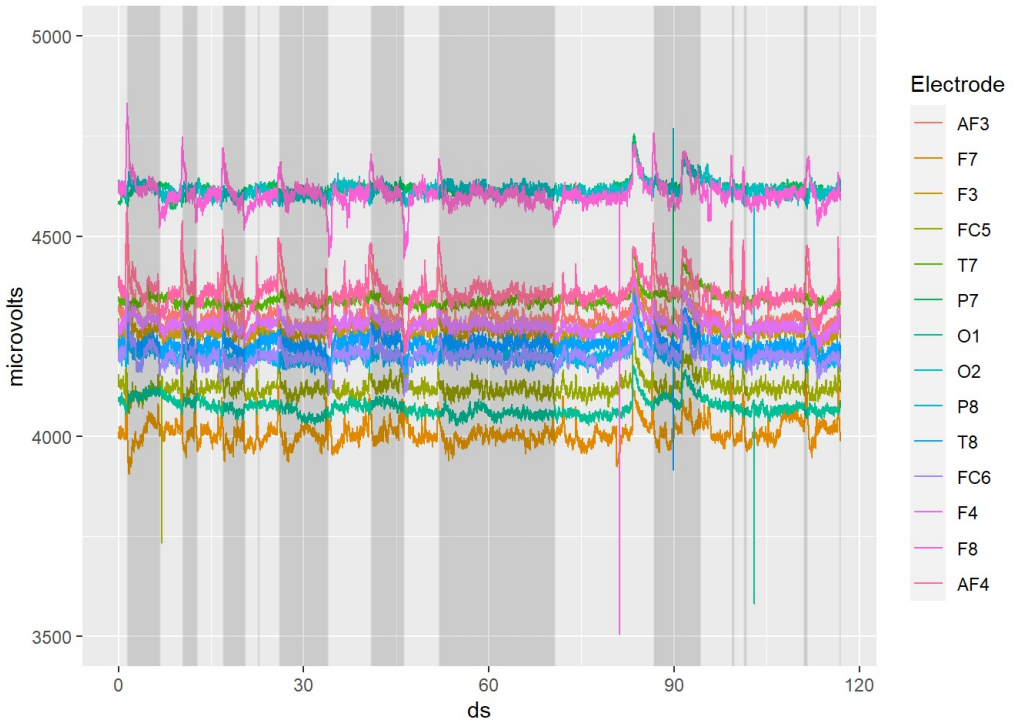
First we use `reshape2::melt()` to transform the `eeg_data` dataset from a wide format to a long format expected by `ggplot2`.

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.name = "Elec
trode", value.name = "microvolts")


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=0.005)
```



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

The ggplot2 line plot of electrode intensities per sampling period reveals some changes in EEG intensity between open and closed eyes. When the eyes were closed, the EEG intensities appeared to be slightly lower than when the eyes were open, as indicated by the dark grey blocks in the plot.

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

A temporal correlation between these states can be predicted based on the distribution of eye open/close states throughout time. Because the dataset contains discrete intervals of eye open and eye closed states, the transitions between these states are likely to have some temporal dependency or pattern.The temporal correlation between eye open/close states can provide insights into the dynamics of eye behaviour and potentially reveal underlying patterns or rhythms in the data

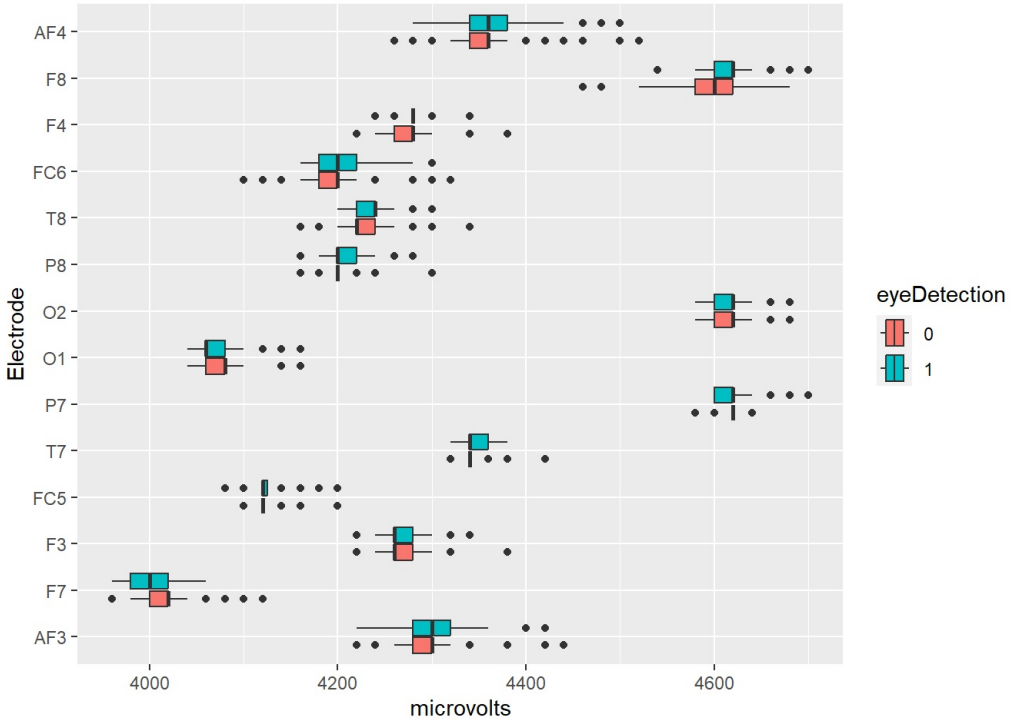
Let's see if we can directly look at the distribution of EEG intensities and see how they related to eye status.

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", value.name
= "microvolts")

# filter huge outliers in voltage
filt_melt_train <- dplyr::filter(melt_train, microvolts %in% (3750:5000)) %>% dplyr::mutate(eyeDetection=as.facto
r(eyeDetection))

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



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 × 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 “EyeDetection” variable, we can get the following information from mean,median and standard deviation for different electrodes:

1.The standard deviation values for each electrode differ without a discernible pattern.There is no consistent pattern of larger variance when the eyes are open or closed when the standard deviation values are examined. 2.For both eye states, the mean values for each electrode are similar or close. There are no significant differences in mean intensities between open and closed eye states for any of the electrodes (AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4).

The mean intensities and standard deviations do not exhibit consistent patterns across the electrodes.We can conclude that no electrode consistently exhibits considerably larger intensity or fluctuation when the eyes are open than 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:

```
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?

It is sequence of observations that depicts statistical properties such as mean,variance over a given period of time and is a concept in time-series. if a time-series is stationary,it will have: Constant Mean Constant Variance Constant Autocovariance

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 of relevance to us because it simplifies the analysis and modelling of time series data. A stationary time series allows us to make valid predictions and inferences based on statistical attributes calculated from a portion of the data.

The result of the ADF Test has the p-values, test-statistics and critical values.If the test statistic is less than the critical values and the p-value is less than a predetermined significance level (e.g., 0.05), the null hypothesis of non-stationarity is rejected. As a result, the associated variable is most likely stationary.

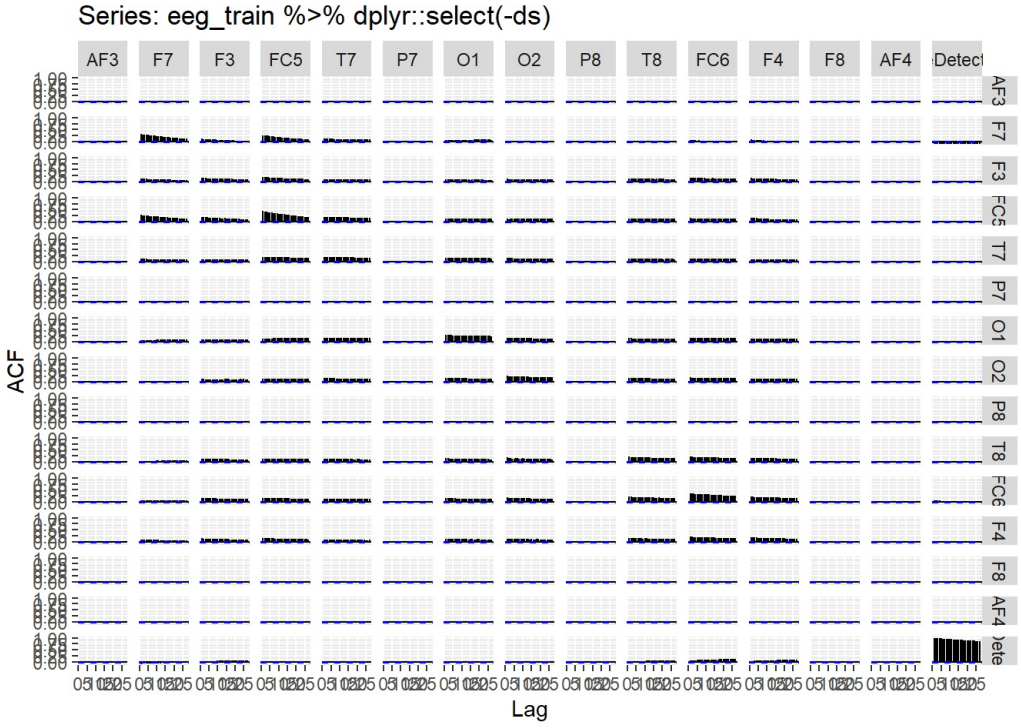
If the test statistic is more than the critical values or the p-value is greater than the significance threshold, the null hypothesis of non-stationarity is not rejected. This suggests that the associated variable is not likely stationary.

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

The ACF plot displays the cross-correlation between each pair of electrode channels and the auto-correlation within the same electrode (the plots along the diagonal.)

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))
```



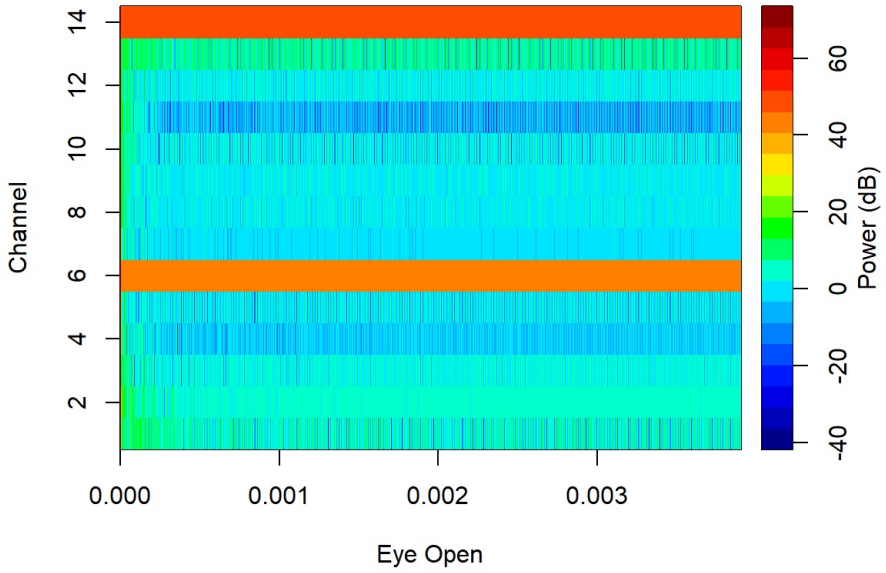
7 Do any fields show signs of strong autocorrelation (diagonal plots)? Do any pairs of fields show signs of cross-correlation? Provide examples.

'Detect' shows the longest spike high peaks in the plot, indicating a strong relationship between the current value and its lagged values within the same electrode channel.

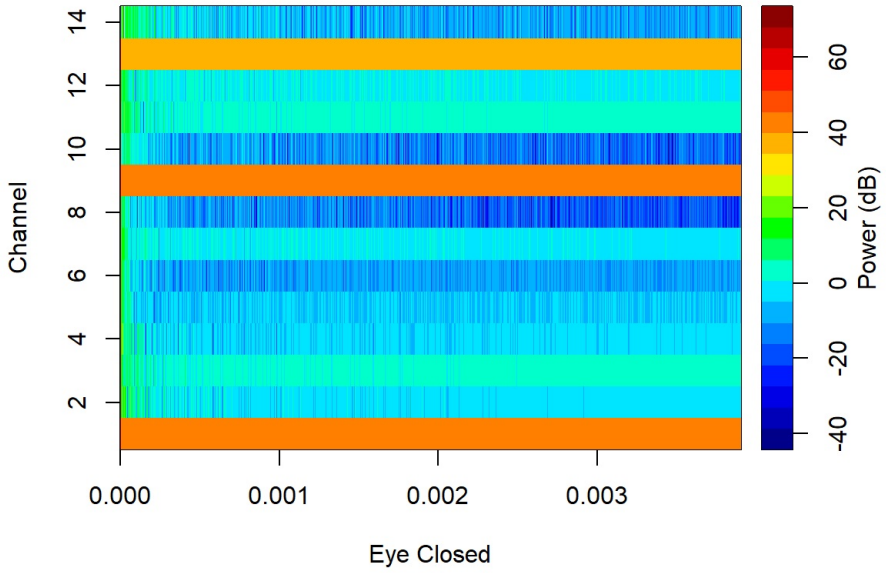
Frequency-Space

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 = Fs, xla
b="Eye Open")
```




```
eegkit::eegpsd(eeg_train %>% dplyr::filter(eyeDetection == 1) %>% dplyr::select(-eyeDetection, -ds), Fs = Fs, xla
b="Eye Closed")
```



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

The power spectral densities for the two eye states were displayed using the 'eegkit::eegpsd()' function. The function was executed twice, once for open eyes and once for closed eyes, and the resulting plots can be compared to discover any variations in power spectral densities between the two eye states [T8].

There appear to be some variances in the power spectral densities for the two eye states. The power spectral density figure for eye closed, for example, exhibits a peak at roughly 10 Hz that is not evident in the power spectral density map for eye open. Furthermore, the power spectral density figure for open eye exhibits a peak at roughly 20 Hz, which does not appear in the power spectral density plot for closed eye.

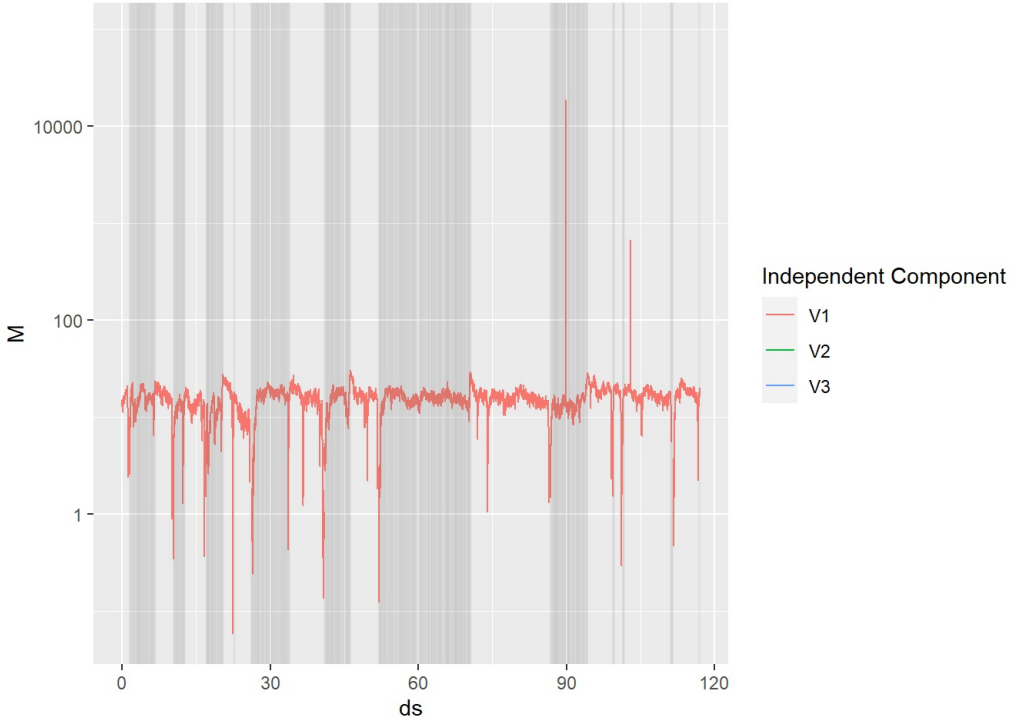
Independent Component Analysis

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", value.n
ame = "M")


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=0.005) +
  ggplot2::scale_y_log10()
```



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

The ggplot2 line plot of independent component activity across electrodes reveals a difference in activity between open and closed eyes. However, this does not imply that eye opening is related to a separate component of activity across the electrodes. More research would be needed to discover whether there is a link between eye opening 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:

```
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
##        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 neural network) fit another model to predict eye opening.

```
set.seed(123)
train_index <- sample(nrow(eeg_data), nrow(eeg_data) * 0.8)
eeg_train <- eeg_data[train_index, ]
eeg_validate <- eeg_data[-train_index, ]

model <- train(eyeDetection ~ ., data = eeg_train, method = "rf")
print(model)
```

```
## Random Forest
##
## 11984 samples
##    16 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 11984, 11984, 11984, 11984, 11984, 11984, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9794370  0.9583832
##    9    0.9927149  0.9852691
##   17    0.9926643  0.9851687
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 9.
```

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

```
set.seed(123)
train_index <- sample(nrow(eeg_data), nrow(eeg_data) * 0.8)
eeg_train <- eeg_data[train_index, ]
eeg_test <- eeg_data[-train_index, ]

# Fit a random forest model to predict eye opening
library(caret)
model <- train(eyeDetection ~ ., data = eeg_train, method = "rf")

predictions <- predict(model, newdata = eeg_test)

confusionMatrix(predictions, eeg_test$eyeDetection)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1671   17
##          1    1 1307
##
##                Accuracy : 0.994
##                  95% CI : (0.9905, 0.9964)
##     No Information Rate : 0.5581
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9878
##
##  Mcnemar's Test P-Value : 0.000407
##
##             Sensitivity : 0.9994
##             Specificity : 0.9872
##          Pos Pred Value : 0.9899
##          Neg Pred Value : 0.9992
##              Prevalence : 0.5581
##          Detection Rate : 0.5577
##    Detection Prevalence : 0.5634
##       Balanced Accuracy : 0.9933
##
##        'Positive' Class : 0
##
```

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

1.RNNs are neural network that can detect temporal relationships in sequential input. They are ideal for analysing time series data such as EEG signals. RNNs may learn patterns and correlations in EEG data across different time steps in the context of predicting eye opening.

2.Hidden Markov Chain Models are probabilistic representations of temporal dependencies in sequential data. HMMs are distinguished by a collection of hidden states as well as observable emissions.

13 Find 2 R libraries you could use to implement these approaches.

1.Keras is a popular R library for constructing recurrent neural networks. It's a high-level deep learning library that connects to the massive TensorFlow library. With keras, you can easily build and train various types of RNN architectures such as LSTM and GRU.

2.Hidden Markov Models is a R library for working with Hidden Markov Models. It includes tools and methods for creating, training, and assessing HMMs. The library supports a variety of HMM types, including discrete, Gaussian, and Gaussian mixtures.

Optional

14 (Optional) As this is the last practical of the course - let me know how you would change future offerings of this course. What worked and didn't work for you (e.g., in terms of the practicals, tutorials, and lectures)? What would you add or remove from the course? What was the main thing you will take away from this course? This will not impact your marks!

It was a great experience with this course, getting a lot of practicl assignments and supportive tutorials, literature reviews and presentations every week. Providing real-world datasets, relevant coding exercises, and clear instructions helped us develop practical skills and reinforce their understanding of the material. There was a lot of student collaboration and fostering a supportive learning community that has enhanced the overall course experience.

I would definitely take this course again since it provided me a lot of information within a short span of time but I wish the course was a little longer.