There are 2 packages you will need to install for today's practical: `install.packages(c("h2o", "eegkit", "forecast", "tseries")` apart from that everything else should already be available on your system. However, 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).

```
knitr::opts_chunk$set(echo = TRUE)

# experimenting with this ML library on my quest to find something pleasant to use in R
library(h2o)
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit https://docs.h2o.ai
##
## ----------------------------------------------------------------------

##
## Attaching package: 'h2o'

## The following objects are masked from 'package:stats':
##
##     cor, sd, var

## The following objects are masked from 'package:base':
##
##     &&, %*%, %in%, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```
h2o::h2o.init(nthreads = 1)
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 hours 57 minutes
##     H2O cluster timezone:       -03:00
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.36.1.2
##     H2O cluster version age:    24 days
##     H2O cluster name:           H2O_started_from_R_f_alh181
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   1.53 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  1
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
```

```
##     R Version:                  R version 4.2.0 (2022-04-22)
```

```
# EEG manipulation library in R (although very limited compared to signal processing libraries availabl
library(eegkit)
```

```
## 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
```

```
# some time series functions (that we only skim the depths of)
library(forecast)
```

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

```
library(tseries)

# just tidyverse libraries that should already be installed
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following object is masked from 'package:signal':
##
##     filter
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(reshape2)
library(purrr)
library(ggplot2)
```

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 `h2o`'s test data S3 bucket:

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

```
##   |                                                                      |
```

```
# add timestamp
Fs <- 117 / dim(eeg_data)[1]
eeg_data <- eeg_data %>% dplyr::mutate(ds=seq(0, 116.99999, by=Fs), eyeDetection=as.factor(eyeDetection))
print(eeg_data %>% dplyr::group_by(eyeDetection) %>% dplyr::count())
```

```
## # A tibble: 2 x 2
## # Groups:   eyeDetection [2]
##   eyeDetection     n
##   <fct>        <int>
## 1 0             8257
## 2 1             6723
```

```
# split dataset into train, validate, test
eeg_train <- eeg_data %>% dplyr::filter(split=='train') %>% dplyr::select(-split)
print(eeg_train %>% dplyr::group_by(eyeDetection) %>% dplyr::count())
```

```
## # A tibble: 2 x 2
## # Groups:   eyeDetection [2]
##   eyeDetection     n
##   <fct>        <int>
## 1 0             4916
## 2 1             4072
```

```
eeg_validate <- h2o::as.h2o(eeg_data %>% dplyr::filter(split=='valid') %>% dplyr::select(-split))
```

```
##   |                                                                      |
```

```
eeg_test <- h2o::as.h2o(eeg_data %>% dplyr::filter(split=='test') %>% dplyr::select(-split))
```

```
##   |                                                                      |
```

0 Knowing the `eeg_data` contains 117 seconds of data, inspect the `eeg_data` dataframe and work out approximately how many samples per second were taken?

A: The sample rate was approximately 128 Hz.

1 How many EEG electrodes/sensors were used?

A: 14 sensors were collected from for this data, with those being the first 14 columns of the dataset.

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!

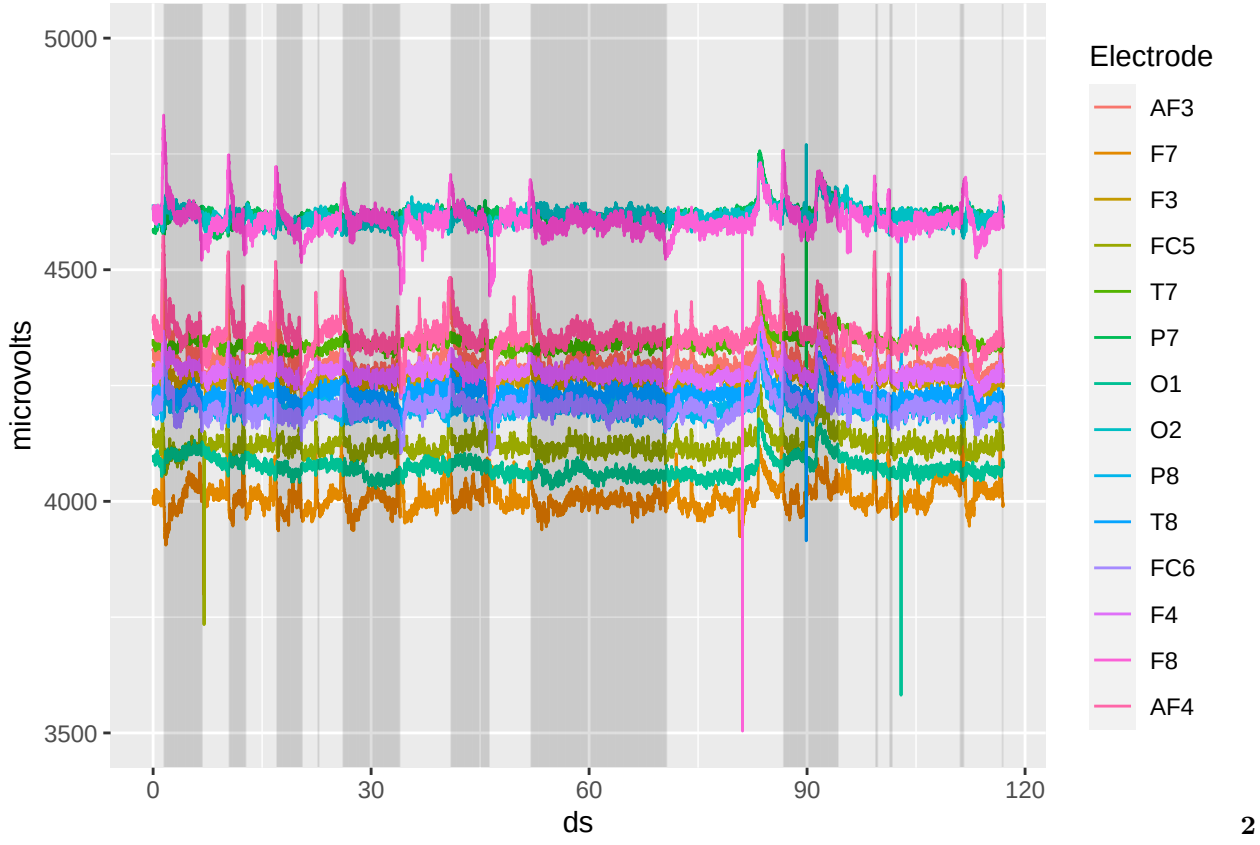
```
h2o::h2o.nacnt(h2o::as.h2o(eeg_data))
```

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

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

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


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?

A: Whenever the eyes are initially opened, there's a spike upwards amongst many of the variables. This is especially apparent in F8 and AF4. F7 also seems to coincide with with this pattern, but with lower spikes and a tendency for the spikes to be slightly earlier than the opening of the eyes, with a downwards spike or decline right after.

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

between these states?

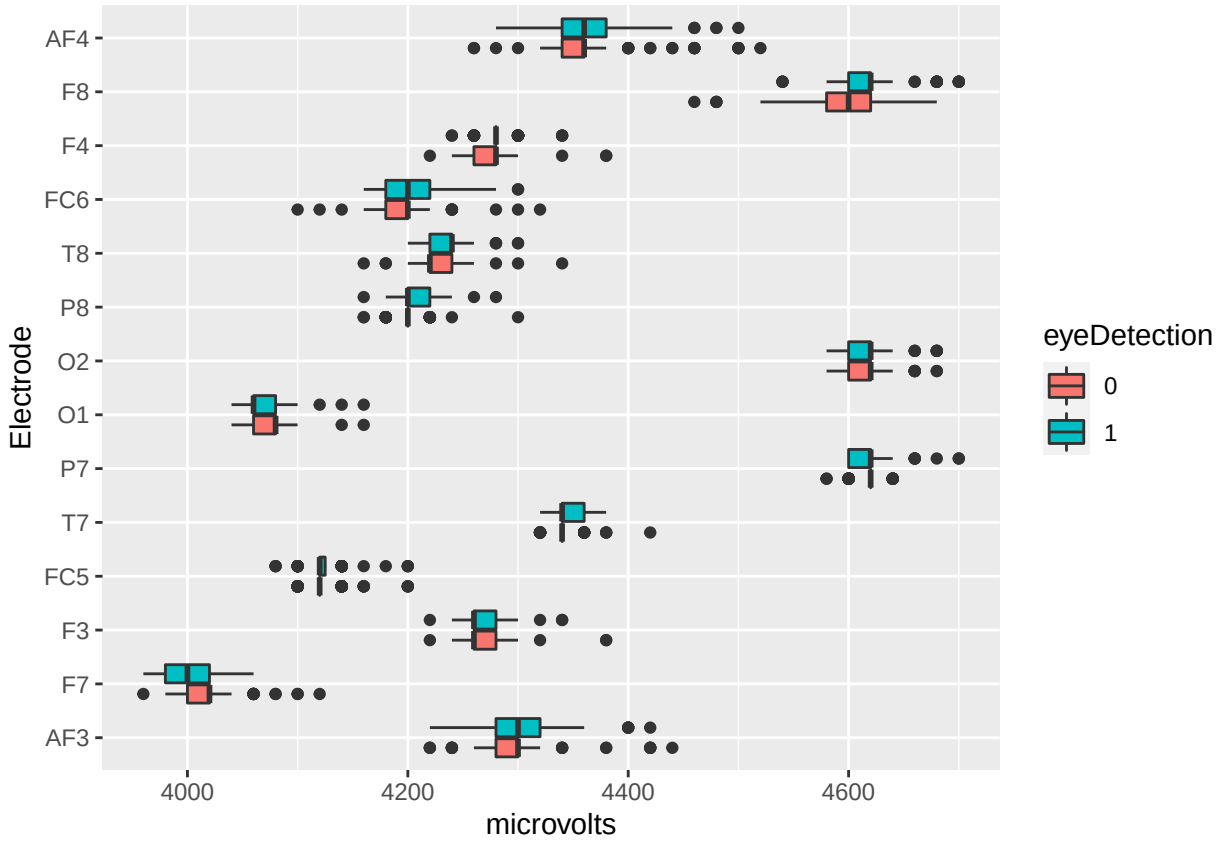
A: Yes, I would expect that

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

```
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:

```
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
## # ... with 18 more rows
```

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

A: It would seem that some average differences occur when looking at the variables with eyes opened and closed, but generally these differences are small and may not reach statistical significance. F7 is slightly higher in activation with eyes closed, but the median and mean differences are within each other's standard deviations.The O1 electrode sees a similar effect, with a median difference of 20 in favour of eyes closed closed. Interestingly, the medians are quite close together despite the medians being roughly a standard deviation apart.

In the T8 electrode a similar effect is seen but in favour of eyes being open, with a median difference of 20, which is roughly a standard deviation apart as well. The means are quite close in this instance, however.

Finally, the F8 electrode had a median and mean difference in favour of eyes open, with both being within the standard deviations. This is also the largest reported mean difference, being almost 20 uV.

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)
```

```
## $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
##
##
```

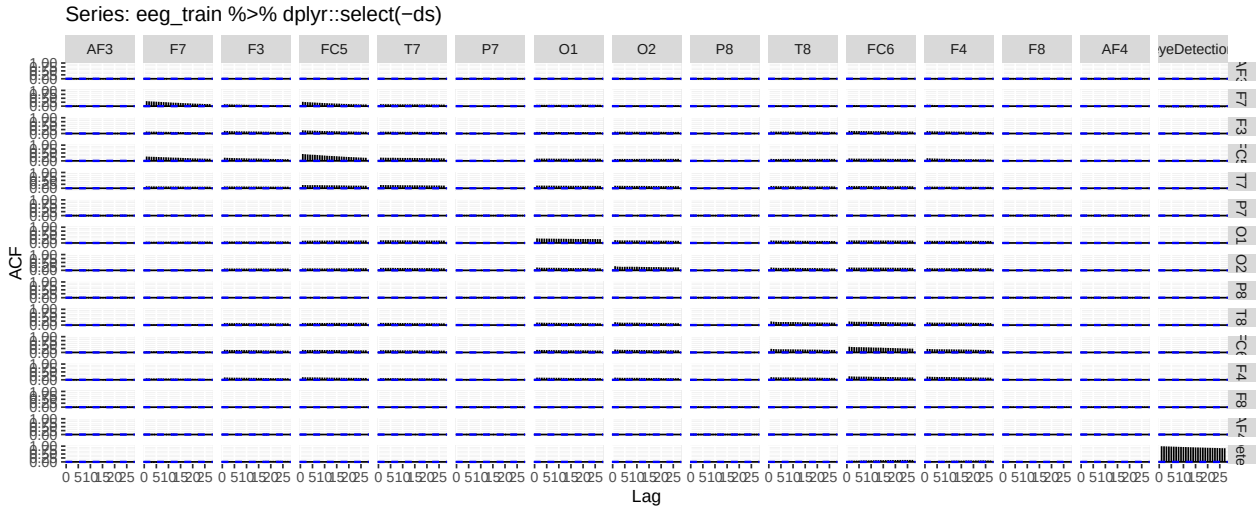
```
## $ds
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -2.4104, Lag order = 20, p-value = 0.4045
## alternative hypothesis: stationary
```

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

A: We are interested in stationarity because, a general trend in the data over time, if it exists, would have to be accouted for when considering the potential effects of different components e.g. eye open/close state. It may be beneficial to remove or otherwise assess the general trend as its own component so that we can more closely look at which changes in activation are attributable or otherwise correlated with eye state. For these test results, the null hypothesis is that a "unit root" exists for the data, and the more negative the score the higher the rejection of this null. With a p-value of 0.01, these results indicate that the odds of observing an ADF value this negative with the null being true in actuality is at or less than 1%. Given this, the null is rejected at the 99% confidence level for all variables except for ds, which is the measure of the time at which the reading was taken. What this means is that there is support for the hypothesis of stationarity in all of the sensor variables.

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?

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

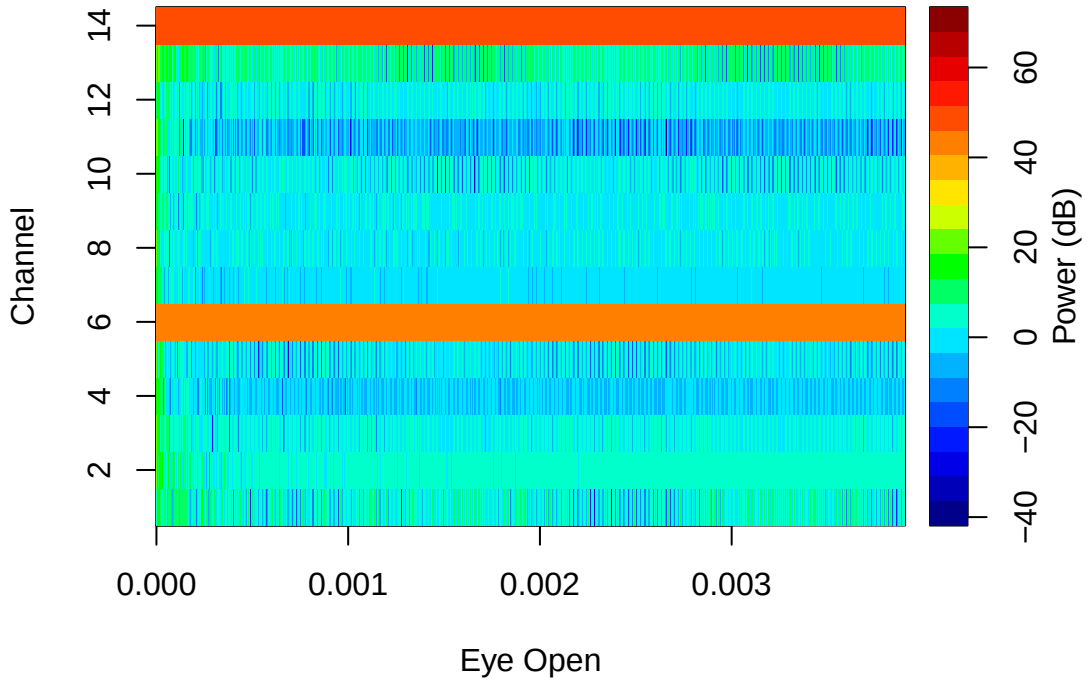


7 Do any fields show signs of strong autocorrelation (diagonal plots)? Do any pairs of fields show signs of cross-correlation? Provide examples.

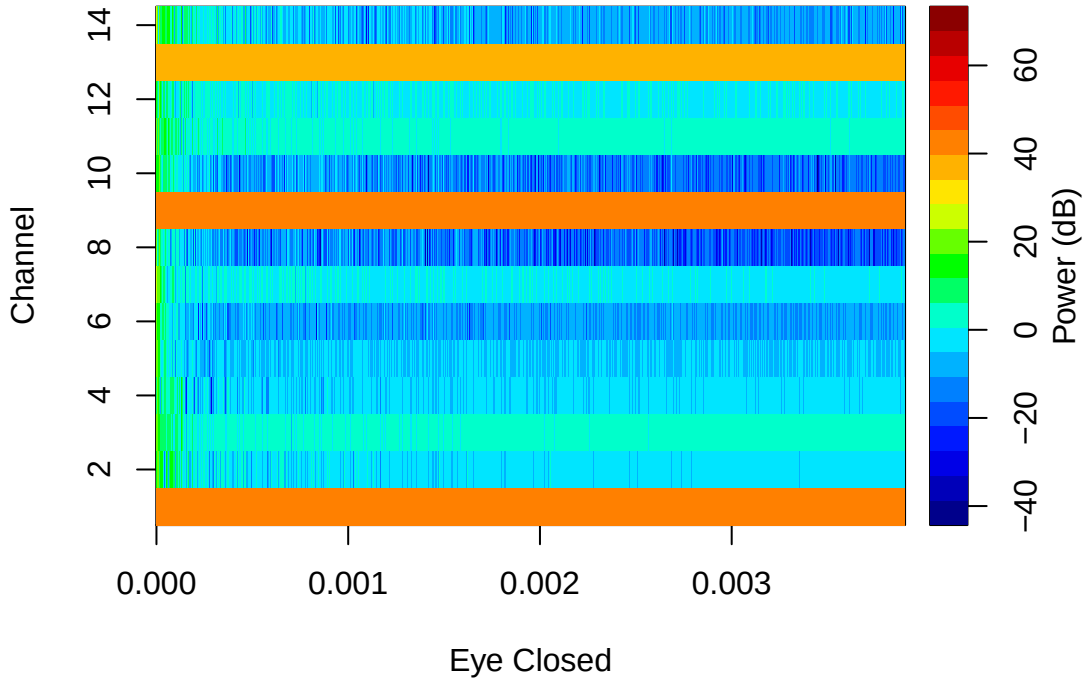
A: F7, FC5, T7, O1, O2, T8, FC6, and F4 all show varying degrees of autocorrelation. Unsurprisingly, autocorrelation is also found in the EyeDetection variable. Out of the sensor variables, of particular note are the correlations for F7, FC5, O1, and FC6. Which are some of the stronger correlations present in the chart overall.

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.

A: One of the most striking differences in spike pattern is the much more negative activation in channels 8 and 10 when the eyes are open. The power level repeatedly reaches around -40 dB in the second graph, while staying around 0 dB in the first. Similarly, channels 1, 9, and 13 have a moderately high and continuous positive activation of 40 dB when the eyes are open despite staying around 0 when the eyes are closed. Finally, the reverse is true of channels 6 and 14, with large positive activation when closed and moderate negative activation when 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)
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

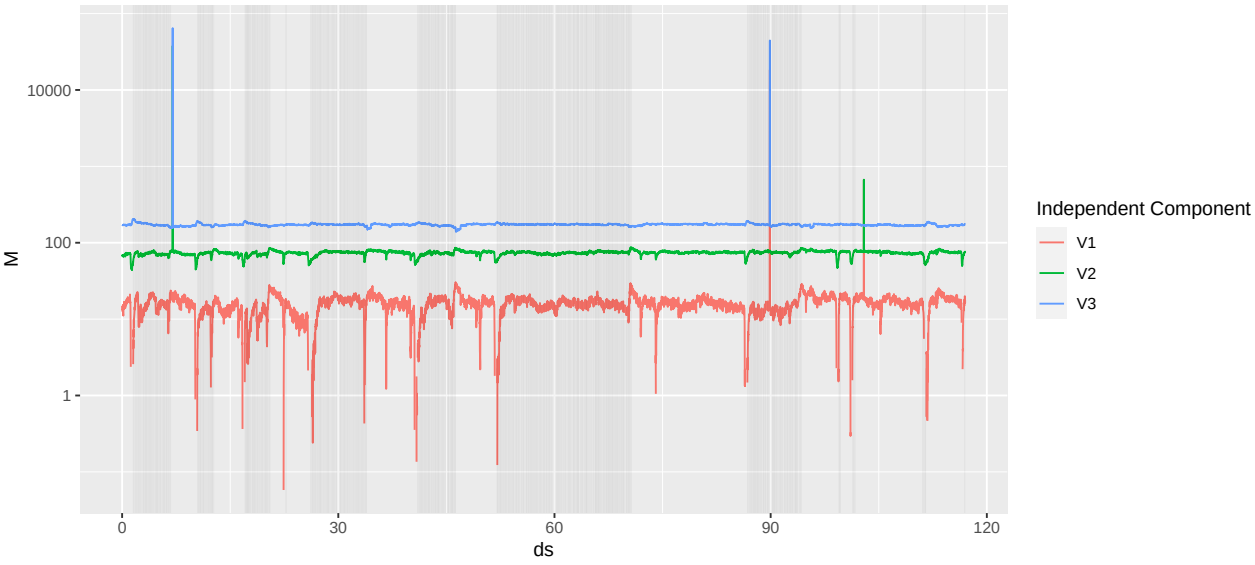
```
mix$eyeDetection <- eeg_train$eyeDetection
mix$ds <- eeg_train$ds
```

```
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()
```

```
## Warning in self$trans$transform(x): NaNs produced
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```



9 Does this suggest eye activate forms an independent component of activity across the electrodes?

A: This plot suggests that the eye activity forms two components that are largely independent, in V1 and V2, which have very different looking spikes but which both follow a pattern of spikes with a clear relation to the opening and closing of the eyes. V2 very consistently shows activation in the form of a downward spike at the beginning of each shaded section (eyes opening). Although much more erratic, V1 also shows this same pattern, and has pronounced spikes that end slightly upward at the time of eyes opening. Upon closer inspection, these upwards spike endpoints are also present in V2, meaning that both components have the same relationship with the eye open/closed state.

This would in my opinion mean that although we can identify components that clearly seem to relate to eye state, eye activity does not necessarily form one independent component, or at the very least that component is neither V1 nor V2, but instead a commonality between the two.

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:

```
model <- h2o::h2o.gbm(x = colnames(dplyr::select(eeg_train, -eyeDetection, -ds)),
                      y = colnames(dplyr::select(eeg_train, eyeDetection)),
                      training_frame = h2o::as.h2o(eeg_train),
                      validation_frame = eeg_validate,
                      distribution = "bernoulli",
                      ntrees = 300,
                      max_depth = 6,
                      learn_rate = 0.31)
```

```
##   |                                                                              |
##   |                                                                              |
```

```
print(model)
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: gbm
## Model ID:  GBM_model_R_1655657002126_8328
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1             300                      300              203508         6
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1         6    6.00000         24         64    49.29333
##
##
## H2OBinomialMetrics: gbm
## ** Reported on training data. **
##
## MSE:  3.410343e-06
## RMSE:  0.001846711
## LogLoss:  0.0009287416
## Mean Per-Class Error:  0
## AUC:  1
## AUCPR:  1
## Gini:  1
## R^2:  0.9999862
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0    1    Error     Rate
## 0      4916    0 0.000000  =0/4916
## 1         0 4072 0.000000  =0/4072
## Totals 4916 4072 0.000000  =0/8988
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
## 1                       max f1  0.981203    1.000000 201
## 2                       max f2  0.981203    1.000000 201
## 3                 max f0point5  0.981203    1.000000 201
## 4                 max accuracy  0.981203    1.000000 201
## 5                max precision  0.999999    1.000000   0
```

```
## 6                   max recall  0.981203    1.000000 201
## 7              max specificity  0.999999    1.000000   0
## 8             max absolute_mcc  0.981203    1.000000 201
## 9   max min_per_class_accuracy  0.981203    1.000000 201
## 10 max mean_per_class_accuracy  0.981203    1.000000 201
## 11                     max tns  0.999999 4916.000000   0
## 12                     max fns  0.999999 3799.000000   0
## 13                     max fps  0.000002 4916.000000 399
## 14                     max tps  0.981203 4072.000000 201
## 15                     max tnr  0.999999    1.000000   0
## 16                     max fnr  0.999999    0.932957   0
## 17                     max fpr  0.000002    1.000000 399
## 18                     max tpr  0.981203    1.000000 201
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## H2OBinomialMetrics: gbm
## ** Reported on validation data. **
##
## MSE:  0.04615386
## RMSE:  0.2148345
## LogLoss:  0.1695443
## Mean Per-Class Error:  0.05844214
## AUC:  0.9857844
## AUCPR:  0.9839267
## Gini:  0.9715688
## R^2:  0.8138274
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0    1    Error       Rate
## 0      1540   95 0.058104   =95/1635
## 1        80 1281 0.058780   =80/1361
## Totals 1620 1376 0.058411  =175/2996
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
## 1                        max f1  0.443251    0.936061 210
## 2                        max f2  0.097827    0.952791 306
## 3                  max f0point5  0.854838    0.948702 107
## 4                  max accuracy  0.498098    0.941589 195
## 5                 max precision  0.999981    1.000000   0
## 6                    max recall  0.000118    1.000000 398
## 7              max specificity  0.999981    1.000000   0
## 8             max absolute_mcc  0.443251    0.882346 210
## 9   max min_per_class_accuracy  0.443251    0.941220 210
## 10 max mean_per_class_accuracy  0.443251    0.941558 210
## 11                     max tns  0.999981 1635.000000   0
## 12                     max fns  0.999981  936.000000   0
## 13                     max fps  0.000019 1635.000000 399
## 14                     max tps  0.000118 1361.000000 398
## 15                     max tnr  0.999981    1.000000   0
## 16                     max fnr  0.999981    0.687730   0
## 17                     max fpr  0.000019    1.000000 399
## 18                     max tpr  0.000118    1.000000 398
##
```

```
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

10 What validation performance can you get with `h2o::h2o.xgboost` instead?

A: Hyperparameters were tested by slowly adjusting ntrees, max_depth, and learn_rate until performance stopped increasing. It was found that a learn rate of about 0.3 was ideal for both models, and performance/running time considerations determined the stopping point for the other two parameters, where performance continued to slightly increase for higher ntrees and 300 was determined to be an adequate stopping point. max_depth showed differing results depending on the value of ntrees but ultimately a depth of 6 was settled on. Strangely, gbm generally outperformed xgboost on AUC and AUCPR metrics, which were the main metrics that were focused on.

```
model2 <- h2o::h2o.xgboost(x = colnames(dplyr::select(eeg_train, -eyeDetection, -ds)),
                      y = colnames(dplyr::select(eeg_train, eyeDetection)),
                      training_frame = h2o::as.h2o(eeg_train),
                      validation_frame = eeg_validate,
                      distribution = "bernoulli",
                      ntrees = 300,
                      max_depth = 6,
                      learn_rate = 0.3)
```

```
##   |                                                                              |
##   |                                                                              |
```

```
print(model2)
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: xgboost
## Model ID:  XGBoost_model_R_1655657002126_8505
## Model Summary:
##   number_of_trees
## 1             300
##
##
## H2OBinomialMetrics: xgboost
## ** Reported on training data. **
##
## MSE:  0.0002065531
## RMSE:  0.01437195
## LogLoss:  0.007713019
## Mean Per-Class Error:  0
## AUC:  1
## AUCPR:  1
## Gini:  1
## R^2:  0.9991664
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0    1    Error     Rate
## 0      4916    0 0.000000  =0/4916
## 1         0 4072 0.000000  =0/4072
## Totals 4916 4072 0.000000  =0/8988
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
```

```
## 1                       max f1  0.888952    1.000000 198
## 2                       max f2  0.888952    1.000000 198
## 3                 max f0point5  0.888952    1.000000 198
## 4                 max accuracy  0.888952    1.000000 198
## 5                max precision  0.999987    1.000000   0
## 6                   max recall  0.888952    1.000000 198
## 7              max specificity  0.999987    1.000000   0
## 8             max absolute_mcc  0.888952    1.000000 198
## 9   max min_per_class_accuracy  0.888952    1.000000 198
## 10 max mean_per_class_accuracy  0.888952    1.000000 198
## 11                      max tns  0.999987 4916.000000   0
## 12                      max fns  0.999987 3827.000000   0
## 13                      max fps  0.000025 4916.000000 399
## 14                      max tps  0.888952 4072.000000 198
## 15                      max tnr  0.999987    1.000000   0
## 16                      max fnr  0.999987    0.939833   0
## 17                      max fpr  0.000025    1.000000 399
## 18                      max tpr  0.888952    1.000000 198
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## H2OBinomialMetrics: xgboost
## ** Reported on validation data. **
##
## MSE:  0.045594
## RMSE:  0.2135275
## LogLoss:  0.15623
## Mean Per-Class Error:  0.06095019
## AUC:  0.9852036
## AUCPR:  0.9827814
## Gini:  0.9704072
## R^2:  0.8160857
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0    1    Error       Rate
## 0      1533  102 0.062385  =102/1635
## 1        81 1280 0.059515   =81/1361
## Totals 1614 1382 0.061081  =183/2996
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
## 1                       max f1  0.436968    0.933285 217
## 2                       max f2  0.136579    0.953299 302
## 3                 max f0point5  0.767994    0.947060 130
## 4                 max accuracy  0.516364    0.939586 196
## 5                max precision  0.999950    1.000000   0
## 6                   max recall  0.003806    1.000000 388
## 7              max specificity  0.999950    1.000000   0
## 8             max absolute_mcc  0.516364    0.878089 196
## 9   max min_per_class_accuracy  0.446211    0.937615 215
## 10 max mean_per_class_accuracy  0.436968    0.939050 217
## 11                      max tns  0.999950 1635.000000   0
## 12                      max fns  0.999950 1171.000000   0
## 13                      max fps  0.000051 1635.000000 399
## 14                      max tps  0.003806 1361.000000 388
```

```
## 15                     max tnr  0.999950    1.000000   0
## 16                     max fnr  0.999950    0.860397   0
## 17                     max fpr  0.000051    1.000000 399
## 18                     max tpr  0.003806    1.000000 388
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

11 Using the best performing of the two models calculate the test performance

A: For some hyperparameter values, xgboost did outperform gbm, but these effects generally became less common for higher values of ntrees. The final comparison between the two models was made with the values of highest observed performance, which happened to be almost the same for the two models, but would probably differ if optimal values for ntrees and max_depth were to be reached. At the comparison point, gbm had slightly higher performance, and so its results are shown below:

```
perf <- h2o::h2o.performance(model = model, newdata = h2o::as.h2o(eeg_test))
print(perf)
```

```
## H2OBinomialMetrics: gbm
##
## MSE:  0.04109554
## RMSE:  0.2027204
## LogLoss:  0.1578308
## Mean Per-Class Error:  0.05147678
## AUC:  0.9871293
## AUCPR:  0.9831307
## Gini:  0.9742587
## R^2:  0.8323863
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0    1    Error       Rate
## 0      1615   91 0.053341   =91/1706
## 1        64 1226 0.049612   =64/1290
## Totals 1679 1317 0.051736  =155/2996
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
## 1                        max f1  0.327880    0.940545 222
## 2                        max f2  0.134817    0.954193 286
## 3                  max f0point5  0.829546    0.950317 115
## 4                  max accuracy  0.327880    0.948264 222
## 5                 max precision  0.999462    0.998403   4
## 6                    max recall  0.000091    1.000000 398
## 7                max specificity  0.999981    0.999414   0
## 8              max absolute_mcc  0.327880    0.894914 222
## 9   max min_per_class_accuracy  0.349626    0.946659 219
## 10 max mean_per_class_accuracy  0.327880    0.948523 222
## 11                     max tns  0.999981 1705.000000   0
## 12                     max fns  0.999981  895.000000   0
## 13                     max fps  0.000016 1706.000000 399
## 14                     max tps  0.000091 1290.000000 398
## 15                     max tnr  0.999981    0.999414   0
## 16                     max fnr  0.999981    0.693798   0
## 17                     max fpr  0.000016    1.000000 399
## 18                     max tpr  0.000091    1.000000 398
##
```

```
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

12 Describe 2 possible alternative modelling approaches we discussed in class but haven't explored in this notebook.

One alternative approach to modelling would be guassian process modelling, which, in comparison to an independent component analysis, involves the generation of many hypothetical components which could model the observed data, and then evaluating the many prospective components or functions on a number of points in order to generate a probable range of potential functions which could generate the observed data, giving something similar in effect to a confidence interval to the distributions it produces that is consistent with the observed data.

Another possible approach would be a hidden markov model, which is a type of state-space model. A hidden markov model could be used here to explore trends within the data, such as associations between individual sensors and eye open/close state, and these trends can be observed using a system of timesteps. With this type of model, each of the timesteps would be considered a new state, and the predictions for a new state are made only based on the current state. Because there is a general regularity to the states, given a fairly consistent sample rate of 128 Hz, this model would find trends in the immediate case, rather than long term, which may or may not provide valuable insight into the data. The model probably wouldn't be ideal to use on its own, but through comparison to other methods some useful trend data may be found. For instance, it might be possible to predict when the eye state will change based on current sensor data using this method, if there are reliable predictors in the data just before a transition from eyes closed to open or vice versa.