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:         2 hours 15 minutes
##     H2O cluster timezone:       America/Halifax
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.36.1.2
##     H2O cluster version age:    18 days
##     H2O cluster name:           H2O_started_from_R_joel-densitas_xbu938
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   7.99 GB
##     H2O cluster total cores:    10
##     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?

There are approximately 128 samples per second.

1 How many EEG electrodes/sensors were used?

Looks like there are 14 sensors in the dataframe.

Exploratory Data Analysis

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

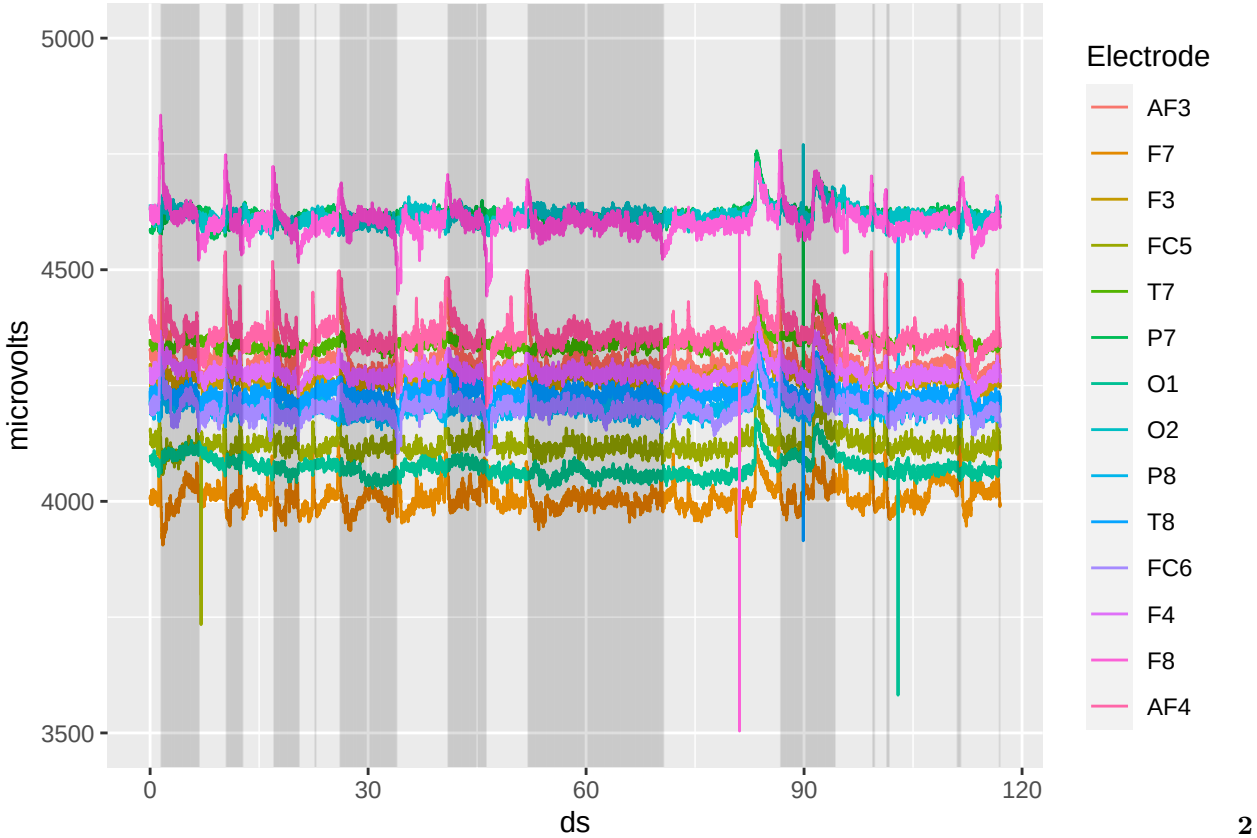
First we should check there is no missing data!

```
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?

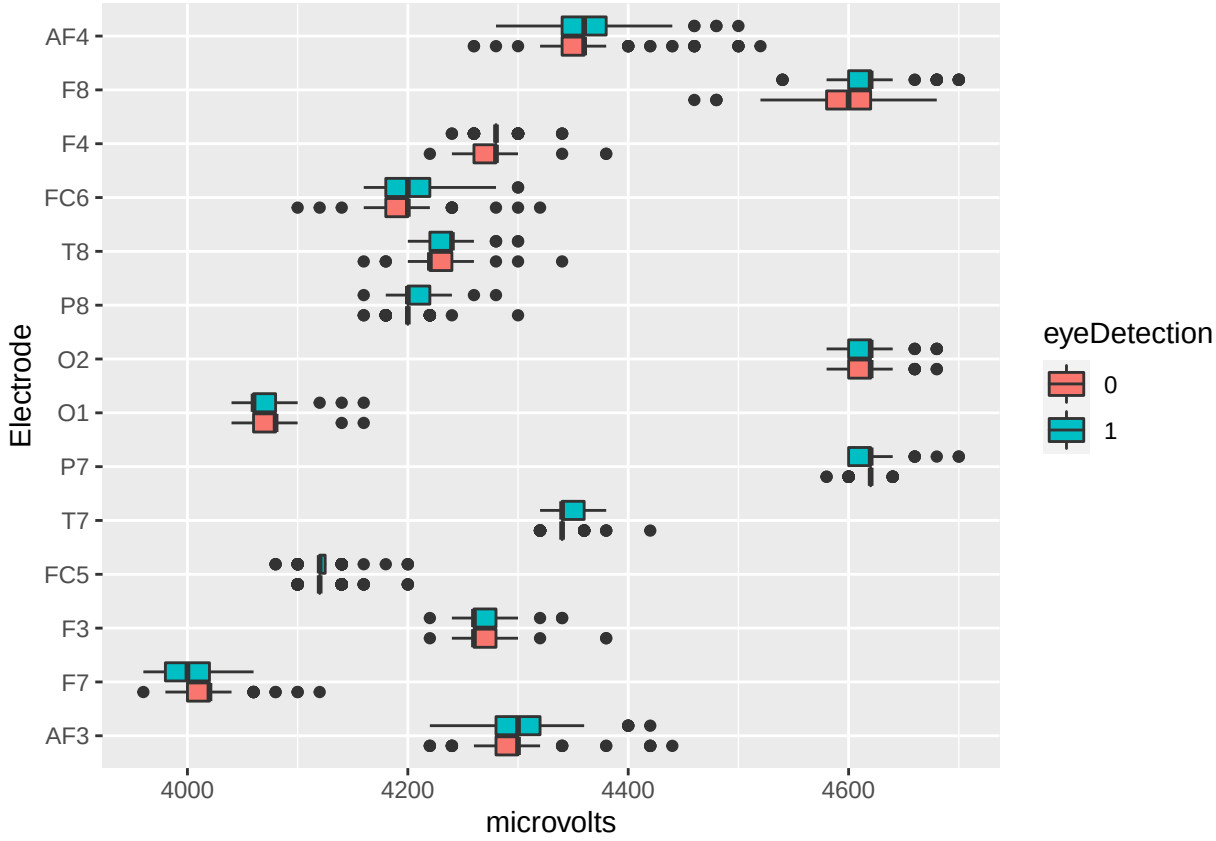
There doesn't seem to be any patterns once the state transitions. However, when the eye opens, there is a sharp positive voltage spike, and a sharp negative one when the eye closes.

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

The plot makes it seem like the eyes are open about as much as they are closed. This changes toward the end, when they're closed much more than open.

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?

More intense: F7, O1, O2, T8

More varied: F7, P7, O1, O2

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
##
```

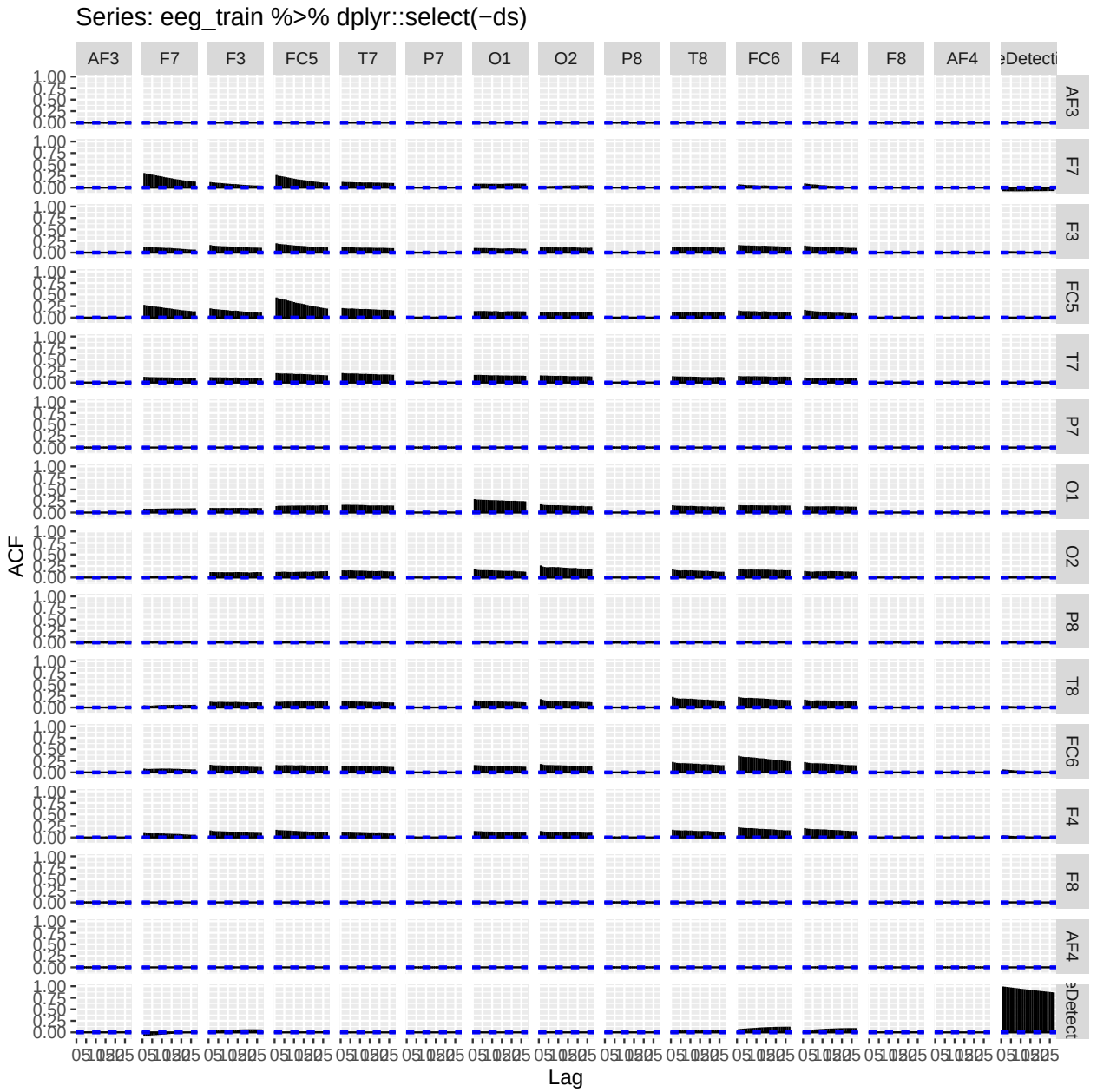
```
##
## $FC6
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -8.6708, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F4
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -10.189, Lag order = 20, p-value = 0.01
## 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. . .)

Stationarity tells us whether the data has any significant changes in magnitude or variance over time. With p-vals of 0.01, we can reject the null and say that the data is stationary.

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))
```



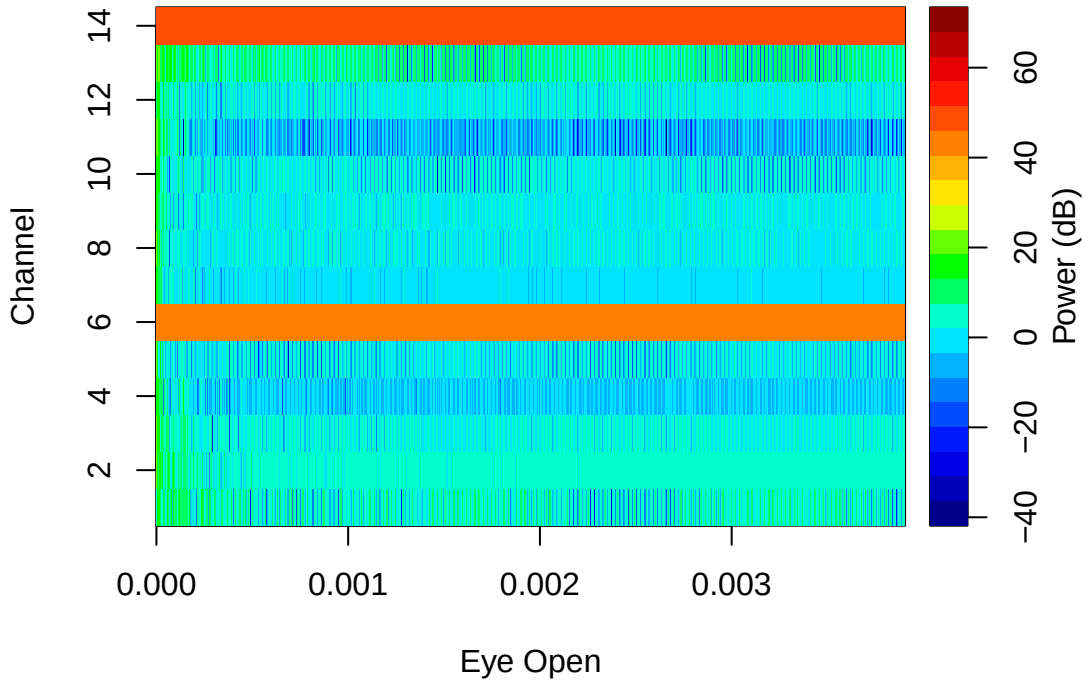
7 Do any fields show signs of strong autocorrelation (diagonal plots)? Do any pairs of fields show signs of cross-correlation? Provide examples.

The fields most strongly autocorrelated are F7, FC5, FC6 and eyeDetection.

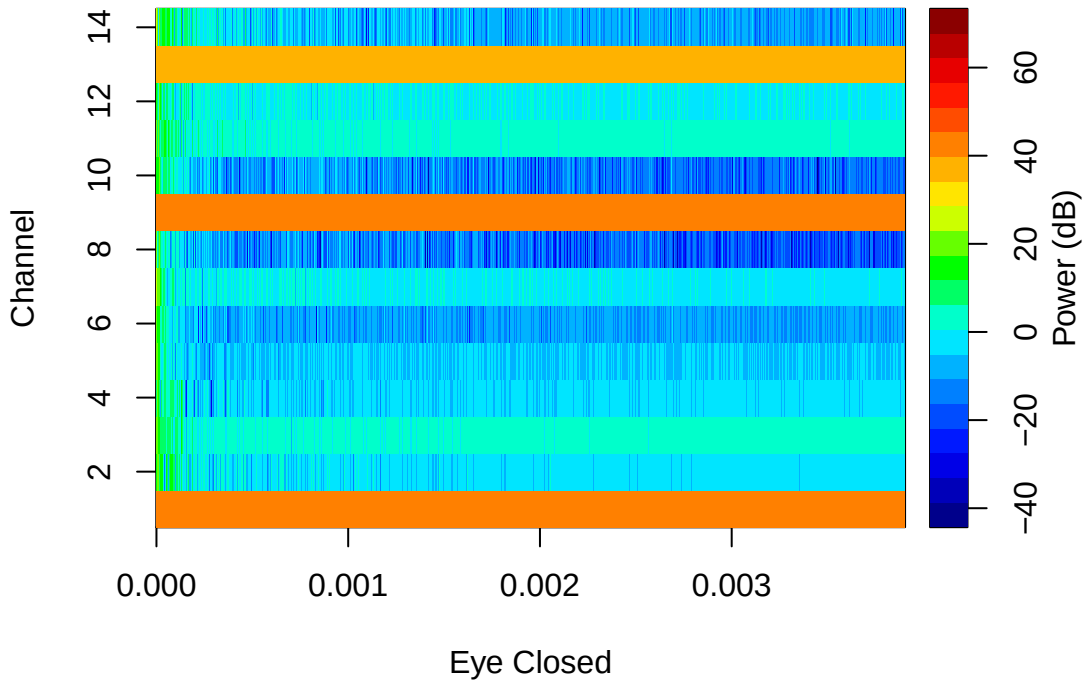
It looks like F7-FC5, T8-FC6, and F4-FC6 are cross-correlated.

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.

For open eyes, channels 6 and 14 are the strongest. This is different than closed eyes, where channels 1, 9

and 13 are strongest. The lower-power channels also appear to be more similar to each other when the eyes are 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)
```

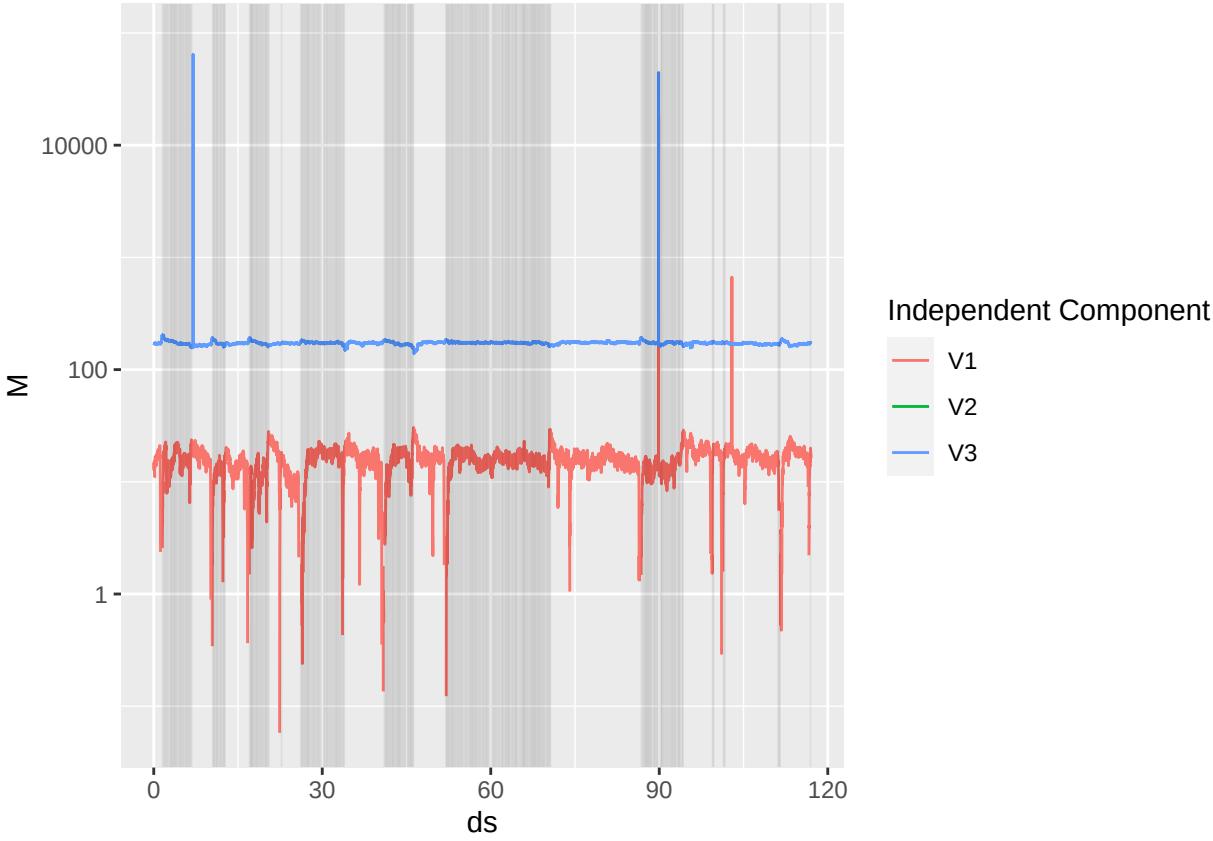
```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

```
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
```

```
## Warning: Removed 8987 row(s) containing missing values (geom_path).
```



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

The ICA seems to suggest that eye activity is an independent component. The majority of the activity in the plot is in the V1 channel, with just a little bit in the V3 channel. I don't see the V2 channel anywhere.

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.gbm <- 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 = 100,
                          max_depth = 4,
                          learn_rate = 0.1)
```

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

```
print(model.gbm)
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: gbm
```

```
## Model ID:  GBM_model_R_1655136128131_808
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1             100                     100               24844         4
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1         4    4.00000         12         16    15.17000
##
##
## H2OBinomialMetrics: gbm
## ** Reported on training data. **
##
## MSE:  0.1076065
## RMSE:  0.3280343
## LogLoss:  0.3600893
## Mean Per-Class Error:  0.1304077
## AUC:  0.9464442
## AUCPR:  0.940608
## Gini:  0.8928883
## R^2:  0.5657448
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0    1    Error        Rate
## 0      4293  623 0.126729   =623/4916
## 1       546 3526 0.134086   =546/4072
## Totals 4839 4149 0.130062  =1169/8988
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
## 1                        max f1  0.454458    0.857803 202
## 2                        max f2  0.306210    0.899876 263
## 3                  max f0point5  0.582581    0.882564 153
## 4                  max accuracy  0.464202    0.870828 198
## 5                 max precision  0.990029    1.000000   0
## 6                    max recall  0.062162    1.000000 380
## 7                max specificity  0.990029    1.000000   0
## 8               max absolute_mcc  0.464202    0.739374 198
## 9   max min_per_class_accuracy  0.449294    0.868861 204
## 10 max mean_per_class_accuracy  0.462152    0.869714 199
## 11                      max tns  0.990029 4916.000000   0
## 12                      max fns  0.990029 4071.000000   0
## 13                      max fps  0.014820 4916.000000 399
## 14                      max tps  0.062162 4072.000000 380
## 15                      max tnr  0.990029    1.000000   0
## 16                      max fnr  0.990029    0.999754   0
## 17                      max fpr  0.014820    1.000000 399
## 18                      max tpr  0.062162    1.000000 380
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## H2OBinomialMetrics: gbm
## ** Reported on validation data. **
##
## MSE:  0.1200593
## RMSE:  0.3464957
## LogLoss:  0.3894168
```

```
## Mean Per-Class Error:  0.1585421
## AUC:  0.9239359
## AUCPR:  0.9173075
## Gini:  0.8478718
## R^2:  0.5157124
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0    1    Error       Rate
## 0      1328  307 0.187768  =307/1635
## 1       176 1185 0.129317  =176/1361
## Totals 1504 1492 0.161215  =483/2996
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
## 1                       max f1  0.425963    0.830705 225
## 2                       max f2  0.317548    0.887594 268
## 3                 max f0point5  0.606576    0.850985 152
## 4                 max accuracy  0.516521    0.846462 189
## 5                max precision  0.978424    1.000000   0
## 6                   max recall  0.084742    1.000000 373
## 7              max specificity  0.978424    1.000000   0
## 8             max absolute_mcc  0.479757    0.690191 204
## 9   max min_per_class_accuracy  0.458183    0.839089 213
## 10 max mean_per_class_accuracy  0.479757    0.844921 204
## 11                     max tns  0.978424 1635.000000   0
## 12                     max fns  0.978424 1358.000000   0
## 13                     max fps  0.012874 1635.000000 399
## 14                     max tps  0.084742 1361.000000 373
## 15                     max tnr  0.978424    1.000000   0
## 16                     max fnr  0.978424    0.997796   0
## 17                     max fpr  0.012874    1.000000 399
## 18                     max tpr  0.084742    1.000000 373
##
## 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?

We get similar performance using xgboost as with gbm. Scores are slightly lower with xgboost for the most part.

```
model.xgb <- 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 = 100,
                      max_depth = 4,
                      learn_rate = 0.1)
```

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

```
print(model.xgb)
```

```
## Model Details:
## ==============
```

```
##
## H2OBinomialModel: xgboost
## Model ID:  XGBoost_model_R_1655136128131_975
## Model Summary:
##   number_of_trees
## 1             100
##
##
## H2OBinomialMetrics: xgboost
## ** Reported on training data. **
##
## MSE:  0.1041154
## RMSE:  0.3226691
## LogLoss:  0.3504083
## Mean Per-Class Error:  0.1241307
## AUC:  0.9511955
## AUCPR:  0.9466089
## Gini:  0.902391
## R^2:  0.5798336
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0    1    Error        Rate
## 0      4269  647 0.131611   =647/4916
## 1       475 3597 0.116650   =475/4072
## Totals 4744 4244 0.124833  =1122/8988
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
## 1                        max f1  0.441854    0.865079 209
## 2                        max f2  0.303433    0.903291 266
## 3                  max f0point5  0.594676    0.891128 149
## 4                  max accuracy  0.497477    0.878727 187
## 5                 max precision  0.988084    1.000000   0
## 6                    max recall  0.086796    1.000000 363
## 7                max specificity  0.988084    1.000000   0
## 8               max absolute_mcc  0.497477    0.755212 187
## 9   max min_per_class_accuracy  0.447978    0.874695 206
## 10 max mean_per_class_accuracy  0.460171    0.876025 201
## 11                      max tns  0.988084 4916.000000   0
## 12                      max fns  0.988084 4061.000000   0
## 13                      max fps  0.005874 4916.000000 399
## 14                      max tps  0.086796 4072.000000 363
## 15                      max tnr  0.988084    1.000000   0
## 16                      max fnr  0.988084    0.997299   0
## 17                      max fpr  0.005874    1.000000 399
## 18                      max tpr  0.086796    1.000000 363
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## H2OBinomialMetrics: xgboost
## ** Reported on validation data. **
##
## MSE:  0.1240409
## RMSE:  0.3521945
## LogLoss:  0.3968676
```

```
## Mean Per-Class Error:  0.166577
## AUC:  0.9156777
## AUCPR:  0.909787
## Gini:  0.8313553
## R^2:  0.4996513
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0    1    Error       Rate
## 0      1369  266 0.162691  =266/1635
## 1       232 1129 0.170463  =232/1361
## Totals 1601 1395 0.166222  =498/2996
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
## 1                       max f1  0.453107    0.819303 210
## 2                       max f2  0.310093    0.874145 270
## 3                 max f0point5  0.618014    0.846704 147
## 4                 max accuracy  0.509092    0.834446 188
## 5                max precision  0.989231    1.000000   0
## 6                   max recall  0.068065    1.000000 377
## 7              max specificity  0.989231    1.000000   0
## 8             max absolute_mcc  0.509092    0.666042 188
## 9   max min_per_class_accuracy  0.448525    0.831804 212
## 10 max mean_per_class_accuracy  0.453107    0.833423 210
## 11                      max tns  0.989231 1635.000000   0
## 12                      max fns  0.989231 1359.000000   0
## 13                      max fps  0.005926 1635.000000 399
## 14                      max tps  0.068065 1361.000000 377
## 15                      max tnr  0.989231    1.000000   0
## 16                      max fnr  0.989231    0.998530   0
## 17                      max fpr  0.005926    1.000000 399
## 18                      max tpr  0.068065    1.000000 377
##
## 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

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

```
## H2OBinomialMetrics: gbm
##
## MSE:  0.1169783
## RMSE:  0.3420209
## LogLoss:  0.3829199
## Mean Per-Class Error:  0.1482088
## AUC:  0.9285579
## AUCPR:  0.9166558
## Gini:  0.8571158
## R^2:  0.5228882
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0    1    Error       Rate
## 0      1482  224 0.131301  =224/1706
## 1       213 1077 0.165116  =213/1290
```

```
## Totals 1695 1301 0.145861  =437/2996
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold       value idx
## 1                        max f1  0.450096    0.831339 204
## 2                        max f2  0.294631    0.884164 272
## 3                  max f0point5  0.594812    0.855658 146
## 4                  max accuracy  0.456380    0.854473 202
## 5                 max precision  0.983987    1.000000   0
## 6                    max recall  0.065326    1.000000 381
## 7                max specificity  0.983987    1.000000   0
## 8               max absolute_mcc  0.456380    0.703061 202
## 9    max min_per_class_accuracy  0.430993    0.846424 212
## 10 max mean_per_class_accuracy  0.450096    0.851791 204
## 11                      max tns  0.983987 1706.000000   0
## 12                      max fns  0.983987 1287.000000   0
## 13                      max fps  0.014244 1706.000000 399
## 14                      max tps  0.065326 1290.000000 381
## 15                      max tnr  0.983987    1.000000   0
## 16                      max fnr  0.983987    0.997674   0
## 17                      max fpr  0.014244    1.000000 399
## 18                      max tpr  0.065326    1.000000 381
##
## 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.

1. We could have used a hidden Markov model to model the transitions between eye-open/eye-closed states, and potentially some hidden ones.
2. We also could have used an ARIMA model to forecast the eye opening or closing. I'm not sure of the usefulness of such a model, but it's something we could consider doing.