There are 3-4 packages you will need to install for today's practical: `install.packages(c("xgboost", "eegkit", "forecast", "tseries", "caret"))` apart from that everything else should already be available on your system.

If you are using a newer Mac you may have to also install quartz to have everything work (do this if you see errors about `X11` during install/execution).

I will endeavour to use explicit imports to make it clear where functions are coming from (functions without `library_name::` are part of base R or a function we've defined in this notebook).

```
## Loading required package: eegkitdata

## Loading required package: bigsplines

## Loading required package: quadprog

## Loading required package: ica

## Loading required package: rgl

## Loading required package: signal

##
## Attaching package: 'signal'

## The following objects are masked from 'package:stats':
##
##     filter, poly

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

## Loading required package: ggplot2

## Loading required package: lattice

##
## Attaching package: 'dplyr'

## The following object is masked from 'package:signal':
##
##     filter

## The following object is masked from 'package:xgboost':
##
##     slice

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

##
## Attaching package: 'purrr'

## The following object is masked from 'package:caret':
##
##     lift
```

EEG Eye Detection Data

One of the most common types of medical sensor data (and one that we talked about during the lecture) are Electroencephalograms (EEGs).
These measure mesoscale electrical signals (measured in microvolts) within the brain, which are indicative of a region of neuronal activity. Typically, EEGs involve an array of sensors (aka channels) placed on the scalp with a high degree of covariance between sensors.

As EEG data can be very large and unwieldy, we are going to use a relatively small/simple dataset today from this paper.

This dataset is a 117 second continuous EEG measurement collected from a single person with a device called a "Emotiv EEG Neuroheadset". In combination with the EEG data collection, a camera was used to record whether person being recorded had their eyes open or closed. This was eye status was then manually annotated onto the EEG data with `1` indicated the eyes being closed and `0` the eyes being open. Measures microvoltages are listed in chronological order with the first measured value at the top of the dataframe.

Let's parse the data directly from the `h2o` library's (which we aren't actually using directly) test data S3 bucket:

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

# add timestamp
Fs <- 117 / nrow(eeg_data)
eeg_data <- transform(eeg_data, ds = seq(0, 116.99999, by = Fs), eyeDetection = as.factor(eyeDetection))
print(table(eeg_data$eyeDetection))
```

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

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

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

```
eeg_validate <- subset(eeg_data, split == 'valid', select = -split)
eeg_test <- subset(eeg_data, split == 'test', select = -split)
```

0 Knowing the `eeg_data` contains 117 seconds of data, inspect the `eeg_data` dataframe and the code above to and determine how many samples per second were taken?

$1/\text{Fs} = 1/0.00781041388518024 = 128.0342$

1 How many EEG electrodes/sensors were used?

There were 14 EEG electrodes/sensors used, as there are 14 variables that represent the electrodes.

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.

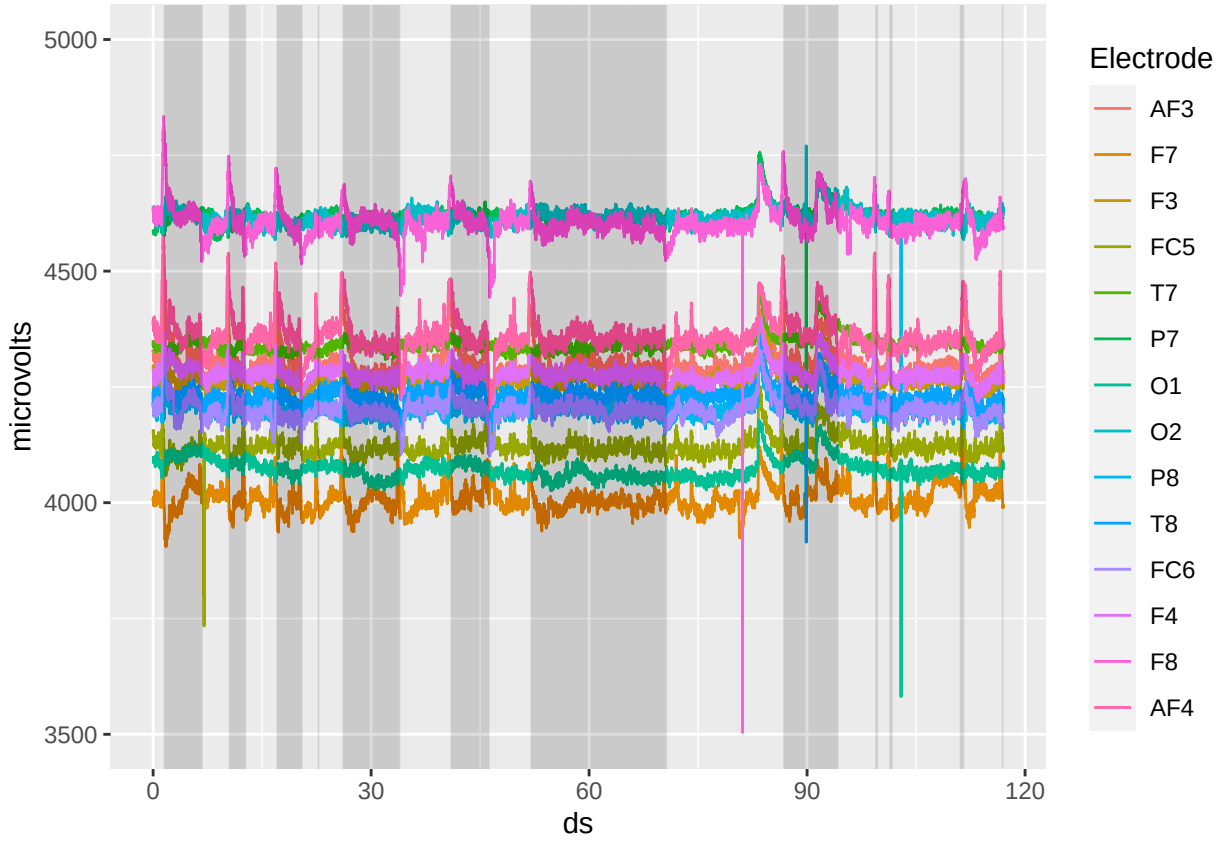
First we use `reshape2::melt()` to transform the `eeg_data` dataset from a wide format to a long format expected by `ggplot2`.

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.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?

When the eyes are open (dark grey blocks) there are usually upward spikes on the EEG intensity at the beginning of each dark grey blocks. Throughout the lighter grey blocks the EEG intensities seem to be more uniform with smaller spikes, when the eyes are closed.

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

Based on the distribution of eye open/closed state over time, I do anticipate a temporal correlation between these states because the change in microvolts are quite similar in each transition from the light grey channels to the dark grey channels.

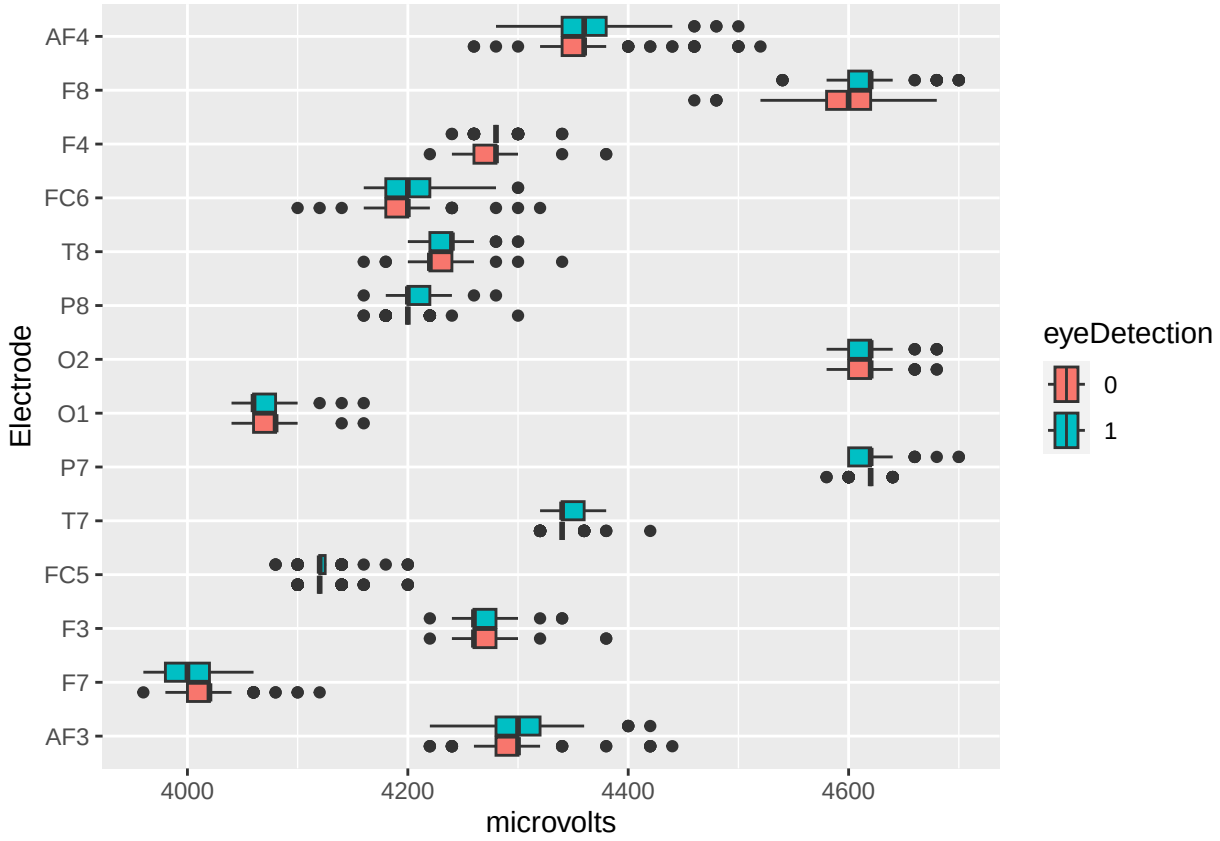
Let's see if we can directly look at the distribution of EEG intensities and see how they related to eye status.

As there are a few extreme outliers in voltage we will use the `dplyr::filter` function to remove values outwith of 3750 to 50003. The function uses the `%in%` operator to check if each value of microvolts is within that range. The function also uses the `dplyr::mutate()` to change the type of the variable eyeDetection from numeric to a factor (R's categorical variable type).

```
melt_train <- reshape2::melt(eeg_train, id.vars=c("eyeDetection", "ds"), variable.name = "Electrode", va

# filter huge outliers in voltage
filt_melt_train <- dplyr::filter(melt_train, microvolts %in% (3750:5000)) %>% dplyr::mutate(eyeDetectio

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



Plots are great but sometimes so it is also useful to directly look at the summary statistics and how they related to eye status. We will do this by grouping the data based on eye status and electrode before calculating the statistics using the convenient `dplyr::summarise` function.

```
filt_melt_train %>% dplyr::group_by(eyeDetection, Electrode) %>%
    dplyr::summarise(mean = mean(microvolts), median=median(microvolts), sd=sd(microvolts)) %>%
    dplyr::arrange(Electrode)
```

```
## `summarise()` has grouped output by 'eyeDetection'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 28 x 5
```

```
## # Groups:   eyeDetection [2]
##    eyeDetection Electrode  mean median    sd
##    <fct>        <fct>     <dbl>  <dbl> <dbl>
##  1 0            AF3       4294.   4300  35.4
##  2 1            AF3       4305.   4300  34.4
##  3 0            F7        4015.   4020  28.4
##  4 1            F7        4007.   4000  24.9
##  5 0            F3        4268.   4260  20.9
##  6 1            F3        4269.   4260  17.4
##  7 0            FC5       4124.   4120  17.3
##  8 1            FC5       4124.   4120  19.2
##  9 0            T7        4341.   4340  13.9
## 10 1            T7        4342.   4340  15.5
## # i 18 more rows
```

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

O1 and F7 are slightly more intense when the eyes are open due to the higher medians when eyeDetection = 0.

AF4, F4, F3 and F7 is slightly more varied when eyes are open due to the higher standard deviation when eyeDetection = 0.

Time-Related Trends As it looks like there may be a temporal pattern in the data we should investigate how it changes over time.

First we will do a statistical test for stationarity:

```r
apply(eeg_train, 2, tseries::adf.test)
```

```
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
```

```
## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value

## Warning in FUN(newX[, i], ...): p-value smaller than printed p-value
## $AF3
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.669, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -12.079, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F3
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -11.587, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $FC5
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -11.122, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $T7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.5644, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $P7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
```

```
## Dickey-Fuller = -20.7, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $O1
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -7.9495, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $O2
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.3537, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $P8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.69, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $T8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.9902, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $FC6
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -8.6708, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F4
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
```

```
## Dickey-Fuller = -10.189, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.642, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $AF4
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.755, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $eyeDetection
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -4.8699, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $ds
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -2.4104, Lag order = 20, p-value = 0.4045
## alternative hypothesis: stationary
```

5 What is stationarity?

Stationarity is used in time series data and shows that means, medians, standard deviations, and autocorrelations do not change over time.

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

We are interested in stationarity because we want to see if the means and variances of the EEG intensities change over time and comparing when the eyes are open versus when the eyes are closed.

All of the p-values are are significant except for the last one (0.4045), therefore the alternative hypotheses were accepted and the data is stationary.

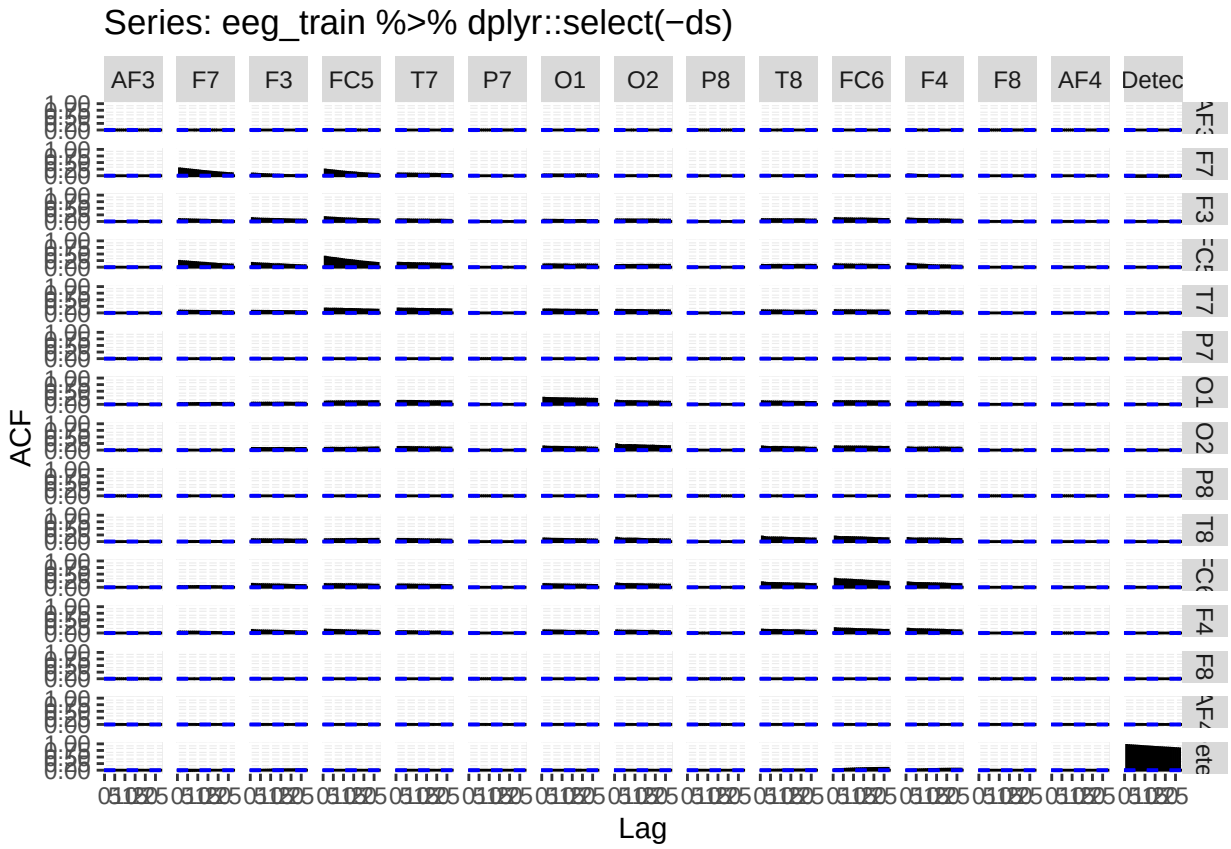
Then we may want to visually explore patterns of autocorrelation (previous values predicting future ones) and cross-correlation (correlation across channels over time) using `forecast::ggAcf` function.

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))
```



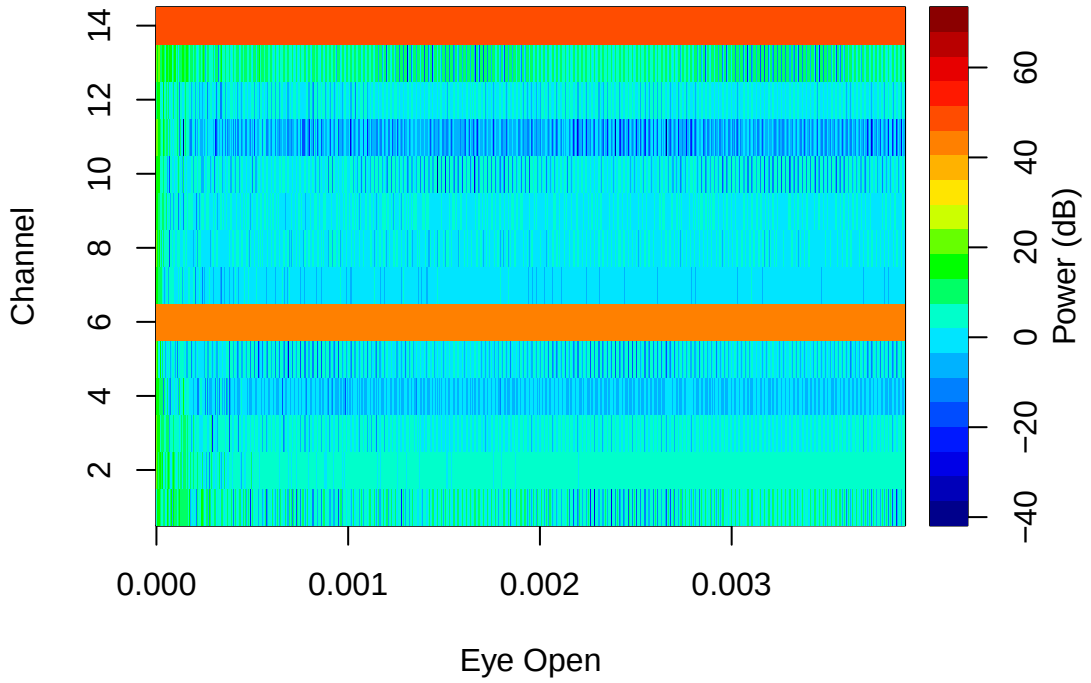
7 Do any fields show signs of strong autocorrelation (diagonal plots)? Do any pairs of fields show signs of cross-correlation? Provide examples.

The fields that show signs of strong autocorrelation are: FC5, F7, FC6, O1

The pairs of fields that show signs of cross-correlation are: FC5 and F7

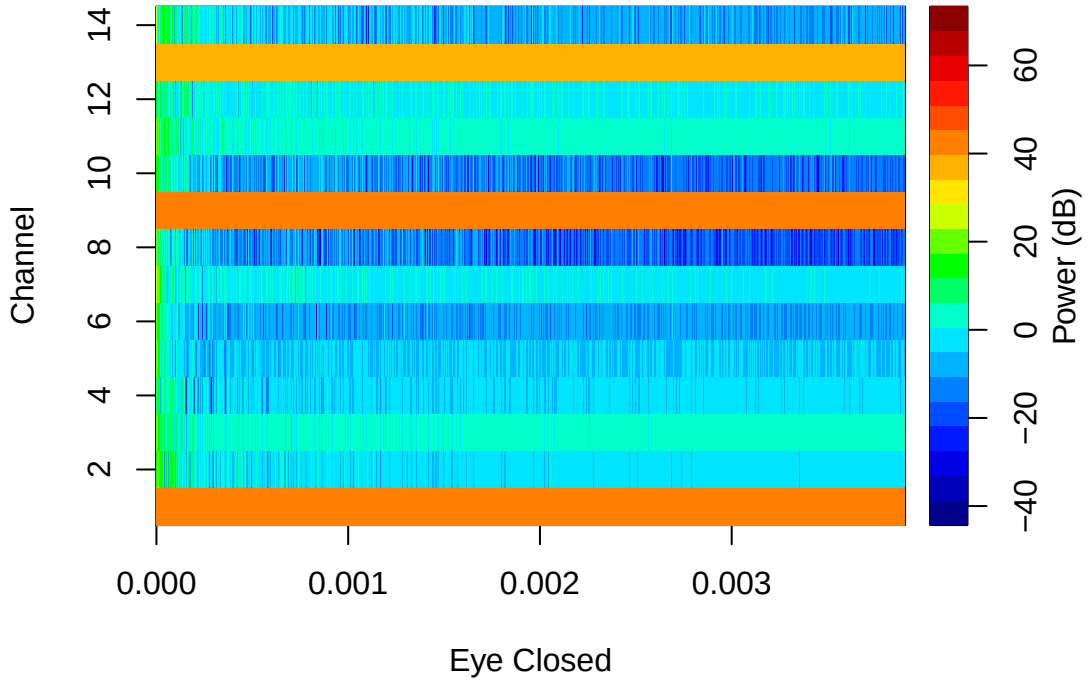
Frequency-Space We can also explore the data in frequency space by using a Fast Fourier Transform. After the FFT we can summarise the distributions of frequencies by their density across the power spectrum. This will let us see if there any obvious patterns related to eye status in the overall frequency distributions.

```
eegkit::eegpsd(eeg_train %>% dplyr::filter(eyeDetection == 0) %>% dplyr::select(-eyeDetection, -ds), Fs
```



Eye Open

```
eegkit::eegpsd(eeg_train %>% dplyr::filter(eyeDetection == 1) %>% dplyr::select(-eyeDetection, -ds), Fs
```



Eye Closed

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

Eyes open: less blue, more red at the top (channel 14), orange in the middle (channel 6).

Eyes closed: more blue, more orange/red at the bottom (channel 1 and 9), dark yellow at top (channel 14).

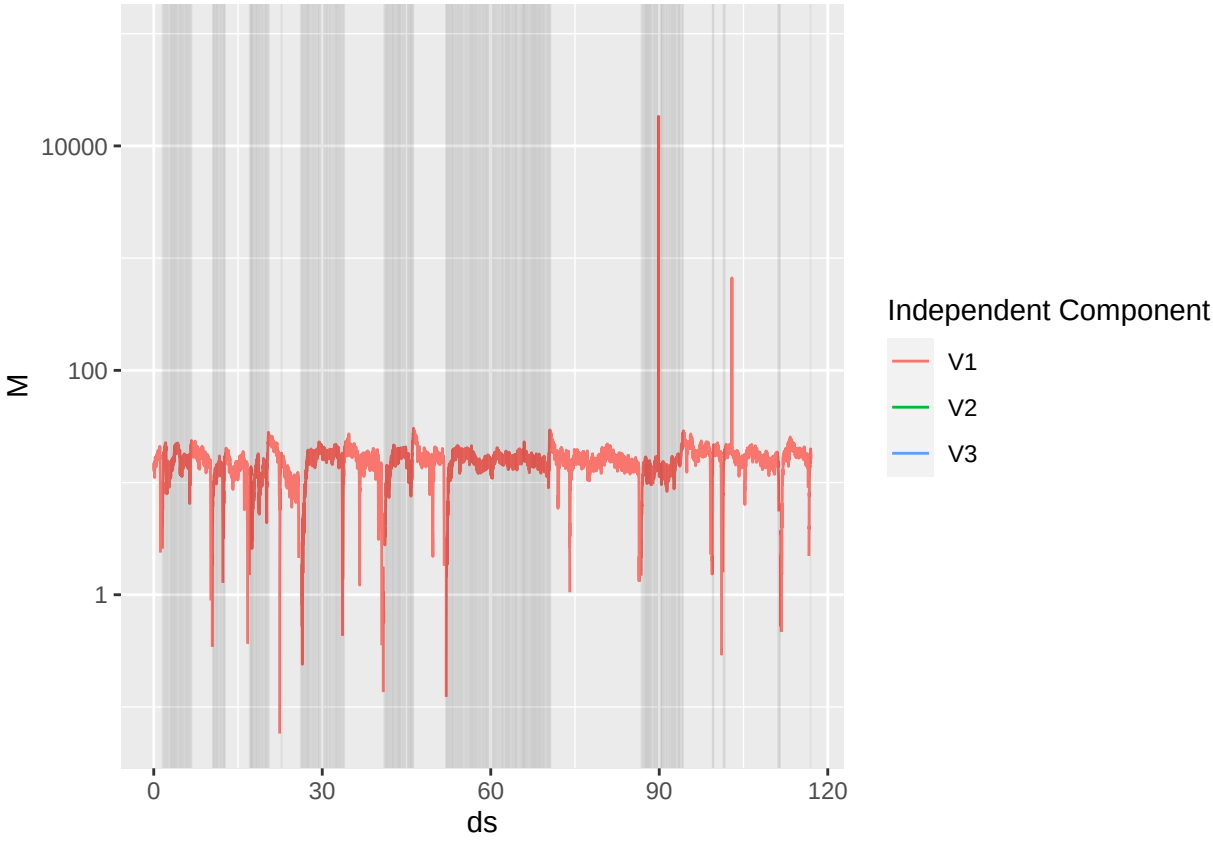
Overall it is more low power (20 to -30 ish), with only 2 channels with high power in eyes open. In the eyes closed graph there is more higher power channels, with most of the power being above -10 in all channels.

Independent Component Analysis We may also wish to explore whether there are multiple sources of neuronal activity being picked up by the sensors.
This can be achieved using a process known as independent component analysis (ICA) which decorrelates the channels and identifies the primary sources of signal within the decorrelated matrix.

```
ica <- eegkit::eegica(eeg_train %>% dplyr::select(-eyeDetection, -ds), nc=3, method='fast', type='time')
mix <- dplyr::as_tibble(ica$M)
mix$eyeDetection <- eeg_train$eyeDetection
mix$ds <- eeg_train$ds

mix_melt <- reshape2::melt(mix, id.vars=c("eyeDetection", "ds"), variable.name = "Independent Component")


ggplot2::ggplot(mix_melt, ggplot2::aes(x=ds, y=M, color=`Independent Component`)) +
  ggplot2::geom_line() +
  ggplot2::geom_vline(ggplot2::aes(xintercept=ds), data=dplyr::filter(mix_melt, eyeDetection==1), alpha=
  ggplot2::scale_y_log10()
```



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

Dark grey indicates the eyes being open open. When the eyes are open there is a big spike downwards. Therefore this does suggest that eye opening relates to an independent component of activity across the electrodes.

Eye Opening Prediction

Now that we've explored the data let's use a simple model to see how well we can predict eye status from the EEGs:

```r
# Convert the training and validation datasets to matrices
eeg_train_matrix <- as.matrix(dplyr::select(eeg_train, -eyeDetection, -ds))
eeg_train_labels <- as.numeric(eeg_train$eyeDetection) -1

eeg_validate_matrix <- as.matrix(dplyr::select(eeg_validate, -eyeDetection, -ds))
eeg_validate_labels <- as.numeric(eeg_validate$eyeDetection) -1

# Build the xgboost model
model <- xgboost(data = eeg_train_matrix,
                 label = eeg_train_labels,
                 nrounds = 100,
                 max_depth = 4,
                 eta = 0.1,
                 objective = "binary:logistic")
```

```
## [1]  train-logloss:0.672173
## [2]  train-logloss:0.653965
## [3]  train-logloss:0.638226
## [4]  train-logloss:0.622881
## [5]  train-logloss:0.610129
## [6]  train-logloss:0.599369
## [7]  train-logloss:0.588913
## [8]  train-logloss:0.577916
## [9]  train-logloss:0.568707
## [10] train-logloss:0.560877
## [11] train-logloss:0.553595
## [12] train-logloss:0.546697
## [13] train-logloss:0.540356
## [14] train-logloss:0.535189
## [15] train-logloss:0.528949
## [16] train-logloss:0.523818
## [17] train-logloss:0.517499
## [18] train-logloss:0.513480
## [19] train-logloss:0.509431
## [20] train-logloss:0.504572
## [21] train-logloss:0.501298
## [22] train-logloss:0.499068
## [23] train-logloss:0.493927
## [24] train-logloss:0.488979
## [25] train-logloss:0.485995
## [26] train-logloss:0.484120
## [27] train-logloss:0.480735
## [28] train-logloss:0.476749
## [29] train-logloss:0.475324
## [30] train-logloss:0.471852
## [31] train-logloss:0.469037
## [32] train-logloss:0.466092
## [33] train-logloss:0.464552
## [34] train-logloss:0.462219
## [35] train-logloss:0.457720
## [36] train-logloss:0.455526
## [37] train-logloss:0.452435
## [38] train-logloss:0.448676
## [39] train-logloss:0.447381
```

```
## [40] train-logloss:0.444740
## [41] train-logloss:0.442885
## [42] train-logloss:0.441704
## [43] train-logloss:0.437273
## [44] train-logloss:0.435778
## [45] train-logloss:0.432542
## [46] train-logloss:0.431302
## [47] train-logloss:0.430229
## [48] train-logloss:0.426916
## [49] train-logloss:0.423800
## [50] train-logloss:0.421908
## [51] train-logloss:0.419130
## [52] train-logloss:0.417934
## [53] train-logloss:0.415003
## [54] train-logloss:0.414027
## [55] train-logloss:0.412499
## [56] train-logloss:0.409956
## [57] train-logloss:0.408821
## [58] train-logloss:0.407170
## [59] train-logloss:0.404487
## [60] train-logloss:0.402856
## [61] train-logloss:0.402061
## [62] train-logloss:0.401169
## [63] train-logloss:0.400292
## [64] train-logloss:0.399799
## [65] train-logloss:0.397281
## [66] train-logloss:0.395909
## [67] train-logloss:0.393773
## [68] train-logloss:0.390365
## [69] train-logloss:0.389440
## [70] train-logloss:0.386978
## [71] train-logloss:0.386324
## [72] train-logloss:0.385750
## [73] train-logloss:0.385101
## [74] train-logloss:0.382760
## [75] train-logloss:0.381081
## [76] train-logloss:0.378908
## [77] train-logloss:0.378428
## [78] train-logloss:0.376087
## [79] train-logloss:0.374926
## [80] train-logloss:0.374230
## [81] train-logloss:0.373538
## [82] train-logloss:0.372971
## [83] train-logloss:0.371651
## [84] train-logloss:0.371134
## [85] train-logloss:0.370717
## [86] train-logloss:0.369246
## [87] train-logloss:0.368063
## [88] train-logloss:0.366157
## [89] train-logloss:0.362902
## [90] train-logloss:0.362461
## [91] train-logloss:0.361028
## [92] train-logloss:0.359356
## [93] train-logloss:0.358512
```

```
## [94] train-logloss:0.356873
## [95] train-logloss:0.356331
## [96] train-logloss:0.355519
## [97] train-logloss:0.354799
## [98] train-logloss:0.353089
## [99] train-logloss:0.351400
## [100]    train-logloss:0.350408
```

```
print(model)
```

```
## ##### xgb.Booster
## raw: 154.4 Kb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, max_depth = 4, eta = 0.1, objective = "binary:logistic")
## params (as set within xgb.train):
##   max_depth = "4", eta = "0.1", objective = "binary:logistic", validate_parameters = "TRUE"
## xgb.attributes:
##   niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
##   cb.evaluation.log()
## # of features: 14
## niter: 100
## nfeatures : 14
## evaluation_log:
##     iter train_logloss
##        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.

```
# Build the caret model
fit.control <- trainControl(method="repeatedcv", number=6,repeats=6)
crt.tree <- train(eyeDetection~., data=eeg_train, method="rpart", trControl=fit.control)
crt.tree
```

```
## CART
##
## 8988 samples
##   15 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (6 fold, repeated 6 times)
## Summary of sample sizes: 7489, 7489, 7490, 7491, 7490, 7491, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
```

```
##   0.05697446  0.7707402  0.5408789
##   0.05832515  0.7295250  0.4656957
##   0.31974460  0.6119714  0.1802322
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.05697446.
```

```
print(crt.tree)
```

```
## CART
##
## 8988 samples
##   15 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (6 fold, repeated 6 times)
## Summary of sample sizes: 7489, 7489, 7490, 7491, 7490, 7491, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.05697446  0.7707402  0.5408789
##   0.05832515  0.7295250  0.4656957
##   0.31974460  0.6119714  0.1802322
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.05697446.
```

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

```
#xgboost model validation matrix
model_2 <- xgboost(data = eeg_validate_matrix,
                   label = eeg_validate_labels,
                   nrounds = 100,
                   max_depth = 4,
                   eta = 0.1,
                   objective = "binary:logistic")
```

```
## [1]  train-logloss:0.670929
## [2]  train-logloss:0.652327
## [3]  train-logloss:0.636649
## [4]  train-logloss:0.621038
## [5]  train-logloss:0.607762
## [6]  train-logloss:0.594711
## [7]  train-logloss:0.583488
## [8]  train-logloss:0.573616
## [9]  train-logloss:0.564726
## [10] train-logloss:0.555171
## [11] train-logloss:0.546140
## [12] train-logloss:0.537264
## [13] train-logloss:0.529169
## [14] train-logloss:0.521566
## [15] train-logloss:0.513323
## [16] train-logloss:0.507607
## [17] train-logloss:0.500895
```

```
## [18] train-logloss:0.494617
## [19] train-logloss:0.490636
## [20] train-logloss:0.486139
## [21] train-logloss:0.481007
## [22] train-logloss:0.476175
## [23] train-logloss:0.472637
## [24] train-logloss:0.468687
## [25] train-logloss:0.464124
## [26] train-logloss:0.460206
## [27] train-logloss:0.458055
## [28] train-logloss:0.454212
## [29] train-logloss:0.449894
## [30] train-logloss:0.446445
## [31] train-logloss:0.442729
## [32] train-logloss:0.441019
## [33] train-logloss:0.437816
## [34] train-logloss:0.434634
## [35] train-logloss:0.431249
## [36] train-logloss:0.427560
## [37] train-logloss:0.424941
## [38] train-logloss:0.421618
## [39] train-logloss:0.418470
## [40] train-logloss:0.417093
## [41] train-logloss:0.414585
## [42] train-logloss:0.411051
## [43] train-logloss:0.408334
## [44] train-logloss:0.406437
## [45] train-logloss:0.403727
## [46] train-logloss:0.401046
## [47] train-logloss:0.399211
## [48] train-logloss:0.396694
## [49] train-logloss:0.393773
## [50] train-logloss:0.390032
## [51] train-logloss:0.388889
## [52] train-logloss:0.387597
## [53] train-logloss:0.385377
## [54] train-logloss:0.383647
## [55] train-logloss:0.382230
## [56] train-logloss:0.378064
## [57] train-logloss:0.376435
## [58] train-logloss:0.373210
## [59] train-logloss:0.371441
## [60] train-logloss:0.369737
## [61] train-logloss:0.366535
## [62] train-logloss:0.365510
## [63] train-logloss:0.363745
## [64] train-logloss:0.361837
## [65] train-logloss:0.359743
## [66] train-logloss:0.358569
## [67] train-logloss:0.357131
## [68] train-logloss:0.356314
## [69] train-logloss:0.353418
## [70] train-logloss:0.351987
## [71] train-logloss:0.348299
```

```
## [72] train-logloss:0.347144
## [73] train-logloss:0.345108
## [74] train-logloss:0.342173
## [75] train-logloss:0.341086
## [76] train-logloss:0.340354
## [77] train-logloss:0.338155
## [78] train-logloss:0.335124
## [79] train-logloss:0.333062
## [80] train-logloss:0.331292
## [81] train-logloss:0.328744
## [82] train-logloss:0.327639
## [83] train-logloss:0.326873
## [84] train-logloss:0.323255
## [85] train-logloss:0.321038
## [86] train-logloss:0.318180
## [87] train-logloss:0.317331
## [88] train-logloss:0.316759
## [89] train-logloss:0.314819
## [90] train-logloss:0.313236
## [91] train-logloss:0.312367
## [92] train-logloss:0.311226
## [93] train-logloss:0.310522
## [94] train-logloss:0.309569
## [95] train-logloss:0.306829
## [96] train-logloss:0.305377
## [97] train-logloss:0.304575
## [98] train-logloss:0.302102
## [99] train-logloss:0.301588
## [100]    train-logloss:0.299242
```

```
print(model_2)
```

```
## ##### xgb.Booster
## raw: 152.7 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     0.6709290
##        2     0.6523275
## ---
```

```
##       99     0.3015877
##      100     0.2992420
```

```
#caret model validation dataset
fit.control <- trainControl(method="repeatedcv", number=6,repeats=6)
crt.tree_2 <- train(eyeDetection~., data=eeg_validate, method="rpart", trControl=fit.control)
crt.tree_2
```

```
## CART
##
## 2996 samples
##   15 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (6 fold, repeated 6 times)
## Summary of sample sizes: 2497, 2498, 2496, 2496, 2497, 2496, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.05694342  0.7938936  0.5925341
##   0.06245408  0.7411980  0.4906761
##   0.30198384  0.6096001  0.1827351
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.05694342.
```

```
print(crt.tree_2)
```

```
## CART
##
## 2996 samples
##   15 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (6 fold, repeated 6 times)
## Summary of sample sizes: 2497, 2498, 2496, 2496, 2497, 2496, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.05694342  0.7938936  0.5925341
##   0.06245408  0.7411980  0.4906761
##   0.30198384  0.6096001  0.1827351
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.05694342.
```

```
#make a confusion matrix to compare models
confusion_matrix <- confusionMatrix(data = crt.tree_2, reference = model_2)
confusion_matrix
```

```
## Cross-Validated (6 fold, repeated 6 times) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
```

```
##            Reference
## Prediction    0    1
##          0 40.7  6.7
##          1 13.9 38.7
##
##  Accuracy (average) : 0.7939
```

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.

We could use wavelet transformations (to allow for retaining of frequency and time resolution) or Gaussian processes (since it can capture time, frequency and state-space models, defined by covariance kernel between functions).

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

rwavelet is a library to implement wavelet transformations and GauProR for Guassian processes.

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!

I think the amount of work/practicals/readings/presentations was very manageable for the six week course.

As an epidemiology student, I would like to have like one class at the beginning for some explanation or an intro to R class that isn't a lab. Maybe this could be an optional class, as computer science students may not need it.

I think that the main thing I will take away from the course is the new coding skills from R. I am doing my thesis in R and I did my undergraduate thesis in R two years ago, so it was good to get back into it before getting into my thesis full time.Another thing I will take away from the course is that health research is so diverse and there is so many different areas, like data science, and throughout my epi courses, we really didn't talk about any topics like machine or deep learning models.