Physiological Sensors

Narges Yarahmadi Gharaei

2023-06-12

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
```

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

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

```r
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?

To determine the number of samples per second in the eeg_data dataframe, we need to examine the code provided.

In the code snippet, the variable Fs is calculated as the inverse of the number of rows in the eeg_data dataframe divided by the total duration in seconds (117 seconds). This value represents the sampling frequency or the number of samples per second.

Therefore, the number of samples per second in the eeg_data dataframe are approximately **128**.

```r
samples_per_second <- nrow(eeg_data)/117
print(samples_per_second)
```

```
## [1] 128.0342
```

we can determine the number of samples per second.

1 How many EEG electrodes/sensors were used

```r
dim(eeg_data)
```

```
## [1] 14980    17
```

Based on the provided dimensions of the eeg_data dataframe (14980 rows and 17 columns), we can determine the number of EEG electrodes/sensors used.

Since the dataset has 17 columns, which includes the timestamp and eye detection columns and split, we need to subtract these three columns from the total number of columns to obtain the number of EEG electrodes/sensors.

Number of EEG electrodes/sensors = Total number of columns - 3

Therefore, the number of EEG electrodes/sensors used in the dataset is:

17 - 3 = 14

Hence, **14** EEG electrodes/sensors were used.

Exploratory Data Analysis

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

First we should check there is no missing data!

```
sum(is.na(eeg_data))
```

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

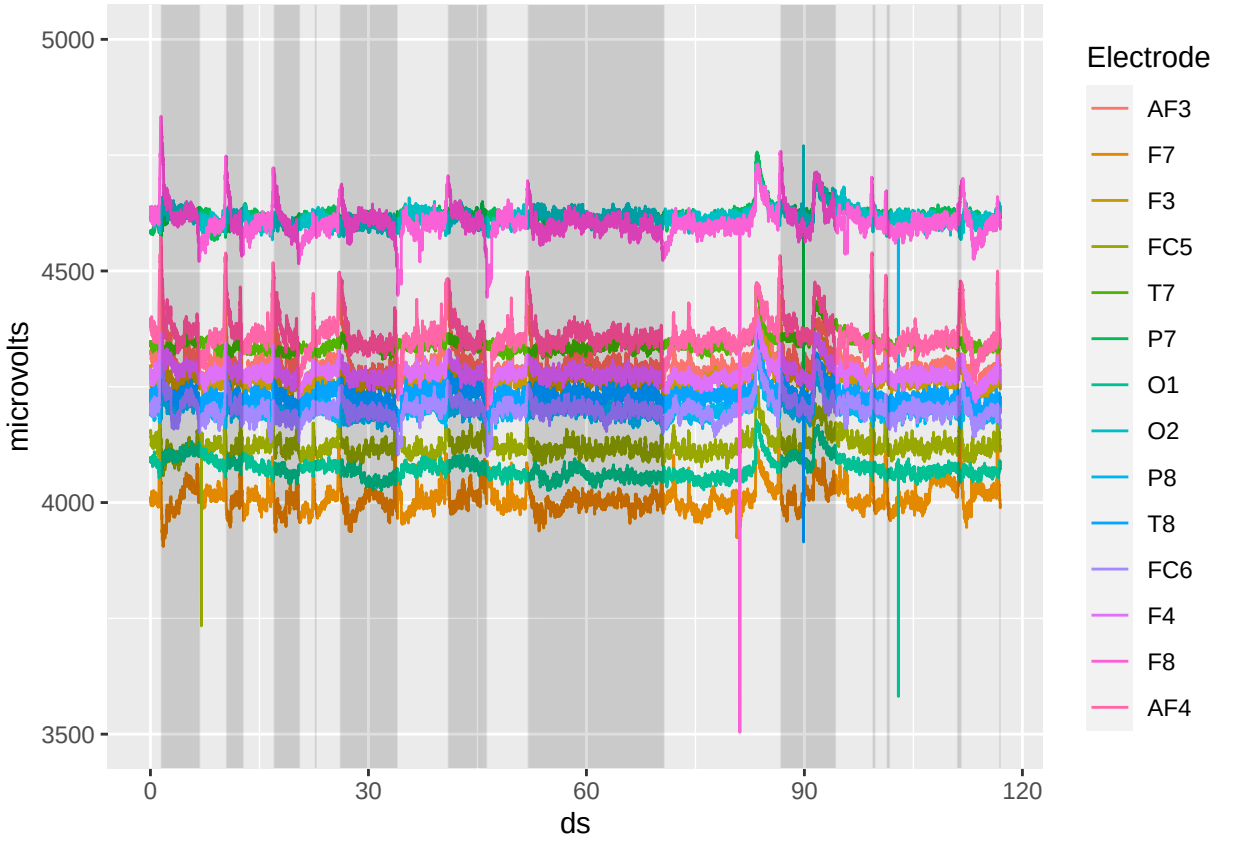
Great, now we can start generating some plots to look at this data within the time-domain.

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

Specifically, this converts from "wide" where each electrode has its own column, to a "long" format, where each observation has its own row. This format is often more convenient for data analysis and visualization, especially when dealing with repeated measurements or time-series data.

We then use `ggplot2` to create a line plot of electrode intensities per sampling time, with the lines coloured by electrode, and the eye status annotated using dark grey blocks.

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


ggplot2::ggplot(melt, ggplot2::aes(x=ds, y=microvolts, color=Electrode)) +
  ggplot2::geom_line() +
  ggplot2::ylim(3500,5000) +
  ggplot2::geom_vline(ggplot2::aes(xintercept=ds), data=dplyr::filter(melt, eyeDetection==1), alpha=0.0
```



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

i think when the eyes being open a high pick happens and when closing a very low pick happens.

Specifically, a high peak occurs when the eyes are open, and a very low peak occurs when the eyes are closed.

This observation aligns with typical EEG patterns related to eye activity. When the eyes are open, there is usually increased visual stimulation and cognitive processing, leading to higher brain activity and higher EEG intensities. Conversely, when the eyes are closed, there is less visual input and reduced cognitive processing, resulting in lower brain activity and lower EEG intensities.

These patterns are commonly observed in EEG recordings and can be indicative of changes in brain states related to eye activity and cognitive processing.

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

Without having direct access to the distribution of eye open/close states over time, it is difficult to make a precise assessment of the temporal correlation between these states. However, based on general knowledge of EEG recordings and eye activity, there can be some expectations regarding temporal correlation.

In EEG recordings, eye open/close states are known to exhibit temporal correlation patterns. For example, it is common to observe that the eye open state persists for relatively longer durations compared to the eye closed state. This is because individuals tend to keep their eyes open for extended periods while engaged in various activities, whereas eye closures may occur intermittently or for shorter durations, such as during blinking or moments of rest.

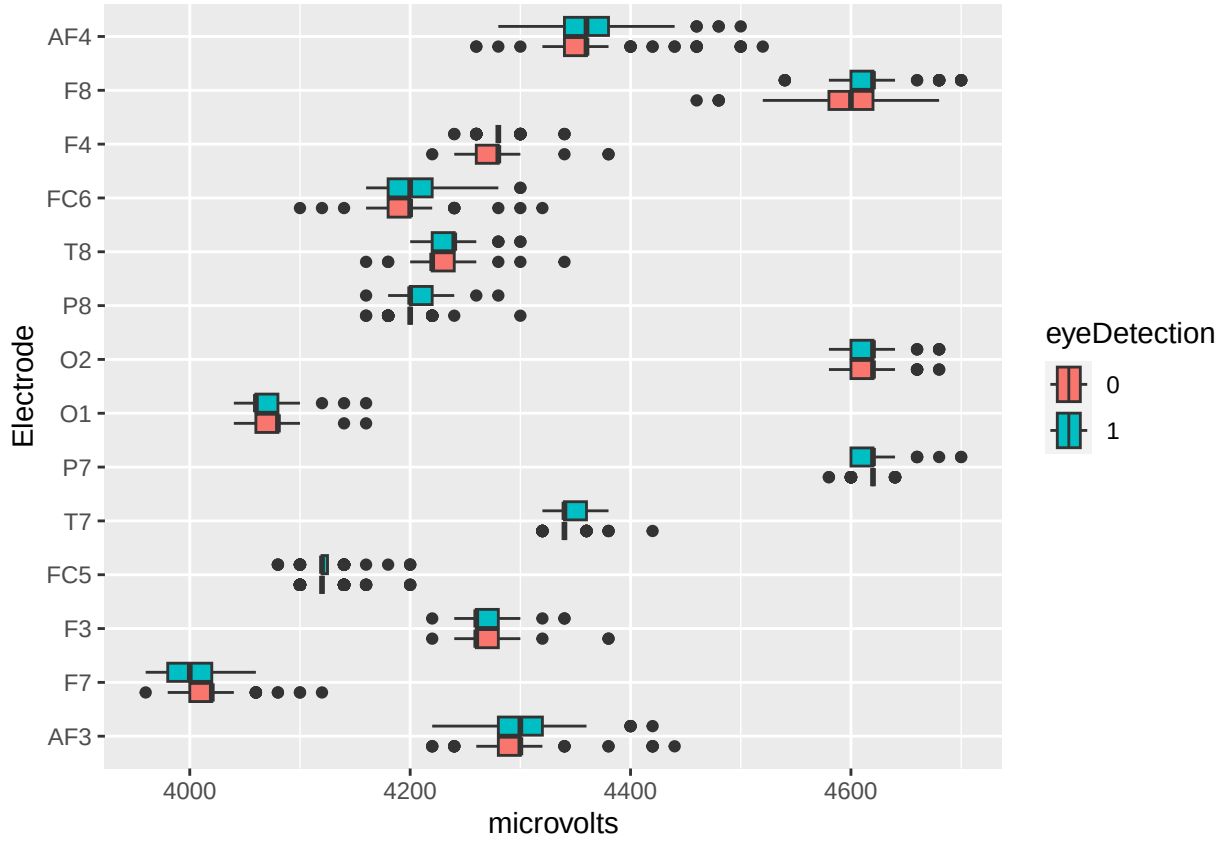
Additionally, certain temporal patterns may emerge, such as regular alternation between eye open and eye closed states, particularly during tasks or activities that involve distinct periods of visual engagement and visual rest.

However, it's important to note that the specific temporal correlation patterns can vary depending on the context, individual characteristics, and the nature of the EEG data being analyzed. Without more detailed information about the distribution and specific characteristics of the eye open/close states over time in your dataset, it's challenging to provide a more specific assessment of the anticipated temporal correlation patterns.

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?

we can analyze the mean, median, and standard deviation (SD) for each electrode during the eye open and eye closed states.

Observations:

Electrode **AF3**: The mean and median values are slightly higher when the eyes are open (4294.453)(Eye State 1) compared to when they are closed (4304.528) (Eye State 0) .

The standard deviation is relatively similar between the two states.

Electrode **F7**: The mean and median values are slightly lower when the eyes are open (4014.775) (Eye State 1) compared to when they are closed (4006.733)(Eye State 0).

The standard deviation is also lower during the eye open state.

Electrode **F3**: The mean value is slightly higher during the eye open state (4267.736) compared to the eye closed state (4268.750).

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

First we will do a statistical test for stationarity:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## $AF3
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.669, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -12.079, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F3
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -11.587, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $FC5
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -11.122, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
```

```
##
##
## $T7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.5644, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $P7
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.7, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $O1
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -7.9495, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $O2
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.3537, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $P8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.69, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $T8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -9.9902, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
```

```
##
##
## $FC6
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -8.6708, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F4
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -10.189, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $F8
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.642, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $AF4
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -20.755, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $eyeDetection
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -4.8699, Lag order = 20, p-value = 0.01
## alternative hypothesis: stationary
##
##
## $ds
##
##  Augmented Dickey-Fuller Test
##
## data:  newX[, i]
## Dickey-Fuller = -2.4104, Lag order = 20, p-value = 0.4045
## alternative hypothesis: stationary
```

5 What is stationarity?

Stationarity refers to a property of a time series where the statistical properties of the series remain constant over time. In a stationary time series, the mean, variance, and autocorrelation structure do not change with time.

A time series can be considered stationary if it satisfies the following criteria:

1.Constant mean: The mean of the series remains constant over time.

2.Constant variance: The variance of the series remains constant over time.

3.Constant autocovariance: The autocovariance between two observations at different time points remains constant over time.

When a time series is stationary, it becomes easier to analyze and model because the statistical properties do not change over time. This allows for more reliable predictions and inferences based on the time series data.

In the context of the provided analysis, the statistical test for stationarity (Augmented Dickey-Fuller test) was performed on each electrode's data and the eye detection variable. The p-value for all electrodes and the eye detection variable was found to be less than 0.01, indicating strong evidence to reject the null hypothesis of non-stationarity. Therefore, based on the test results, the data for the electrodes and eye detection variable can be considered stationary.

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 it is an important assumption in many time series analysis techniques and models. When a time series is stationary, it allows us to make certain assumptions about its behavior and apply appropriate statistical tools for analysis. The results of the stationarity tests provide valuable information about the nature of the time series data and guide us in selecting appropriate modeling strategies.

Here are a few reasons why we are interested in stationarity and what the results of the tests tell us:

1. Model Selection: Stationarity is a fundamental assumption in many time series models, such as autoregressive integrated moving average (ARIMA) models. If the data is stationary, we can directly apply these models without the need for further transformations. The test results confirm whether the data meets this assumption and guide us in selecting the appropriate model.

2. Predictability: Stationary time series tend to exhibit consistent patterns and behaviors over time. By confirming stationarity, we can rely on past observations to make predictions about future values. This assumption allows us to use historical data patterns and relationships for forecasting and decision-making.

3. Statistical Analysis: Stationary time series enable us to use various statistical techniques that assume constant statistical properties over time. For example, we can calculate reliable estimates of means, variances, and correlations. The test results assure us that these statistical properties remain consistent throughout the time series.

4. Interpretability: Stationarity simplifies the interpretation of the time series data. With stationary data, the relationship between variables, trends, and other patterns can be more easily understood and interpreted. This enhances our ability to extract meaningful insights and draw accurate conclusions from the data.

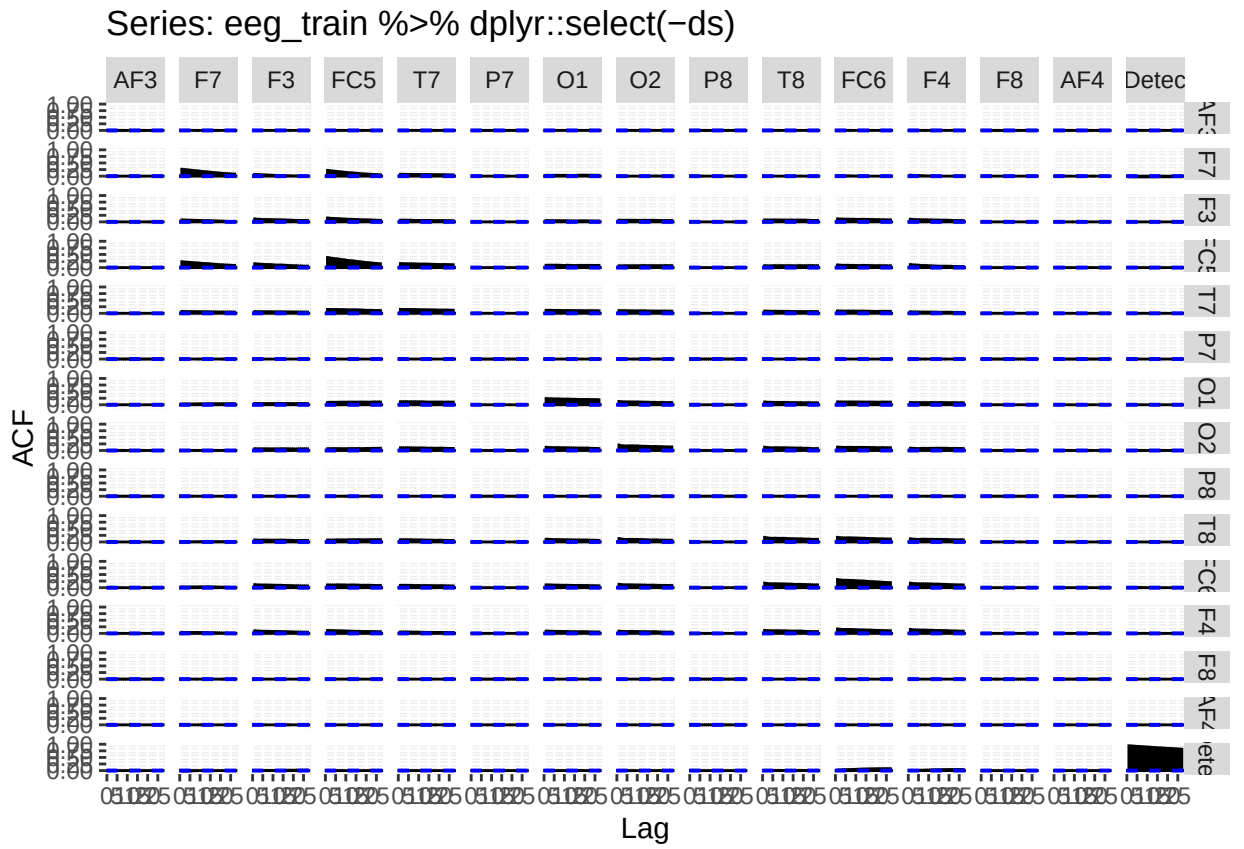
However, it's important to note that the lack of multiple comparison correction in the given analysis may introduce the potential for false positives. Multiple comparison correction methods, such as the Bonferroni correction, should be applied to control the family-wise error rate when conducting multiple hypothesis tests. Ignoring this correction increases the risk of finding false statistically significant results by chance alone.

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.

In the top plot of the autocorrelation function (ACF) result, the sign of strong autocorrelation is indicated by the presence of positive or negative spikes extending beyond the shaded region.

If a spike extends above the shaded region, it indicates positive autocorrelation, which means there is a positive relationship between the values at a given time point and the values at previous time points. This suggests a pattern of increasing values over time.

If a spike extends below the shaded region, it indicates negative autocorrelation, which means there is a negative relationship between the values at a given time point and the values at previous time points. This suggests a pattern of alternating values or decreasing values over time.

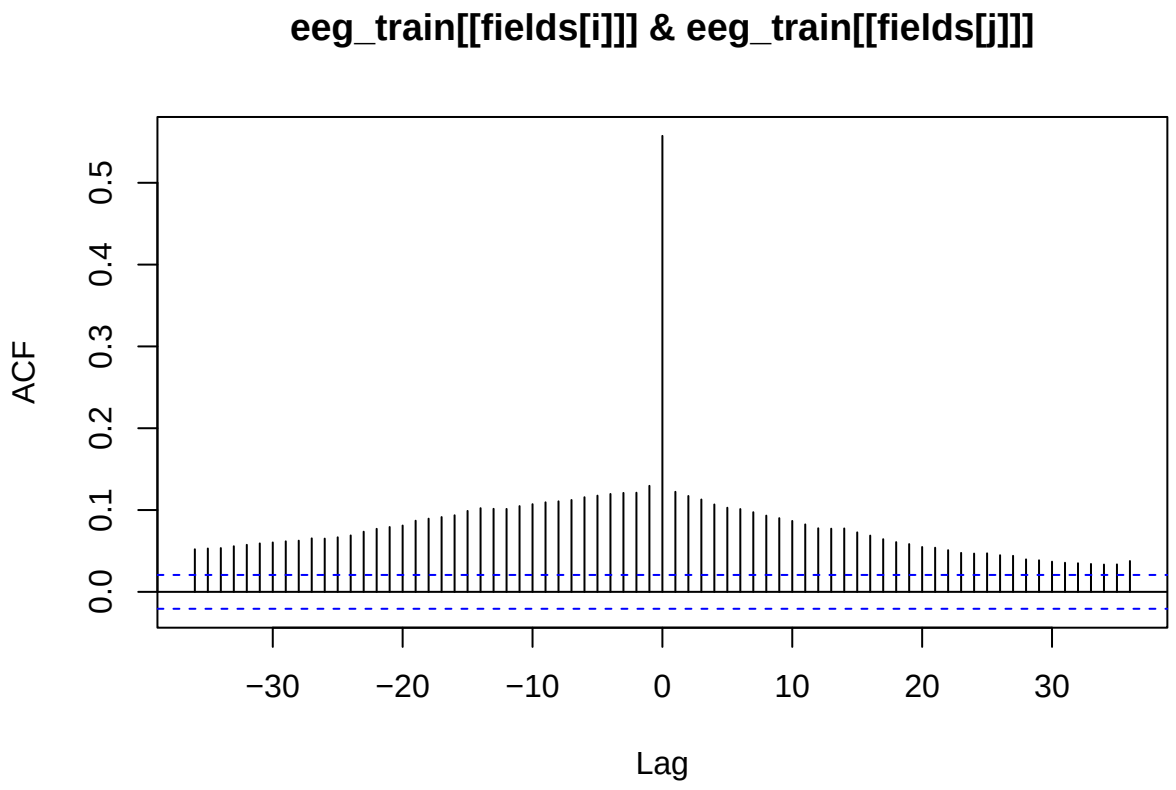
The direction of the spikes (above or below the shaded region) indicates the sign of the strong autocorrelation. Positive spikes indicate positive autocorrelation, while negative spikes indicate negative autocorrelation. In columns like **F7** and **FC5** i see strong autocorrelation.

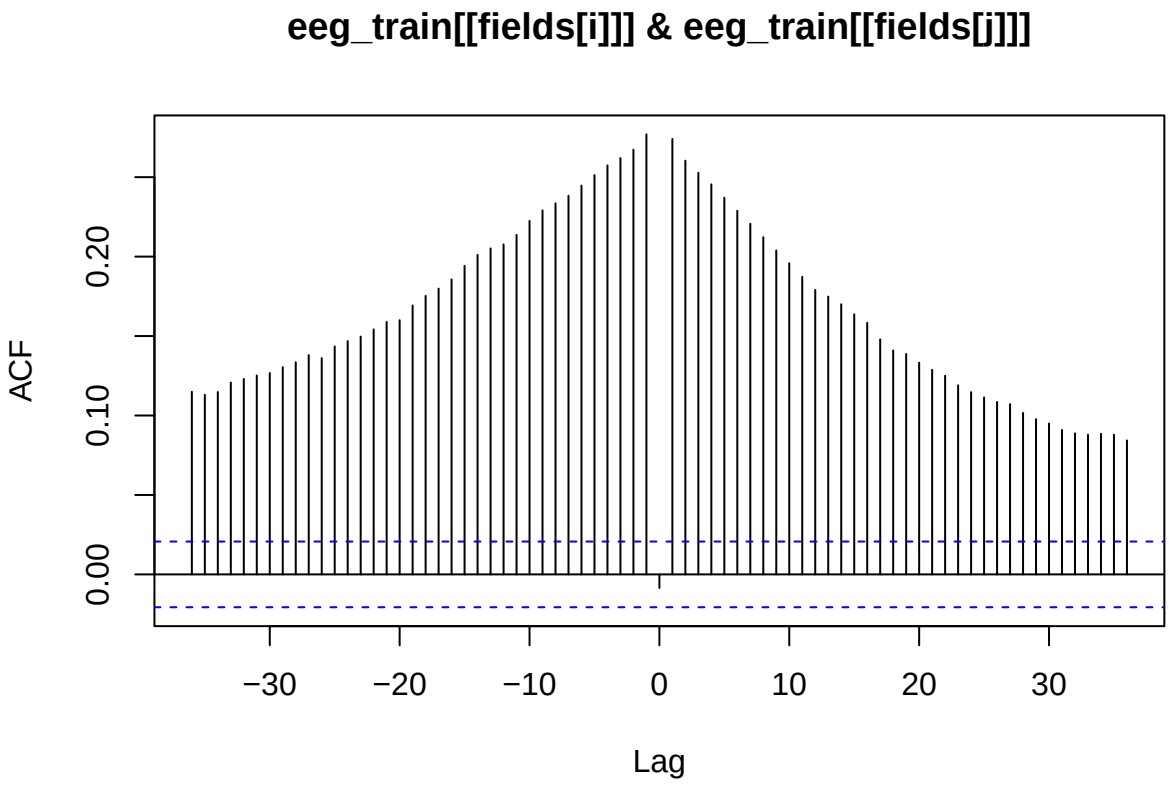
Actually above plot shows cross-correlation too but for achieving better insight i tried to analyze cross-correlation between all pairs of fields in the dataset by using nested loops to iterate through the variables and calculate the CCF for each pair.

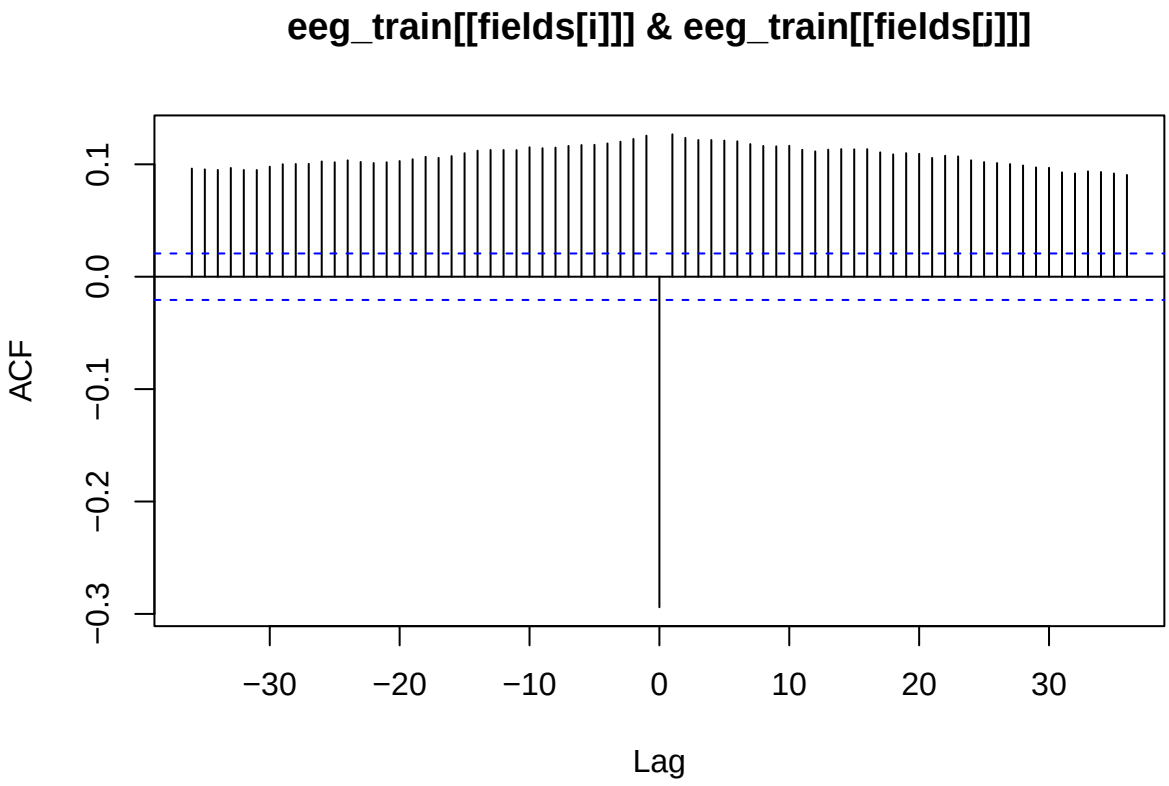
The following code will calculate the cross-correlation between all possible pairs of fields in the eeg_train dataset and store the results in the ccf_results list.

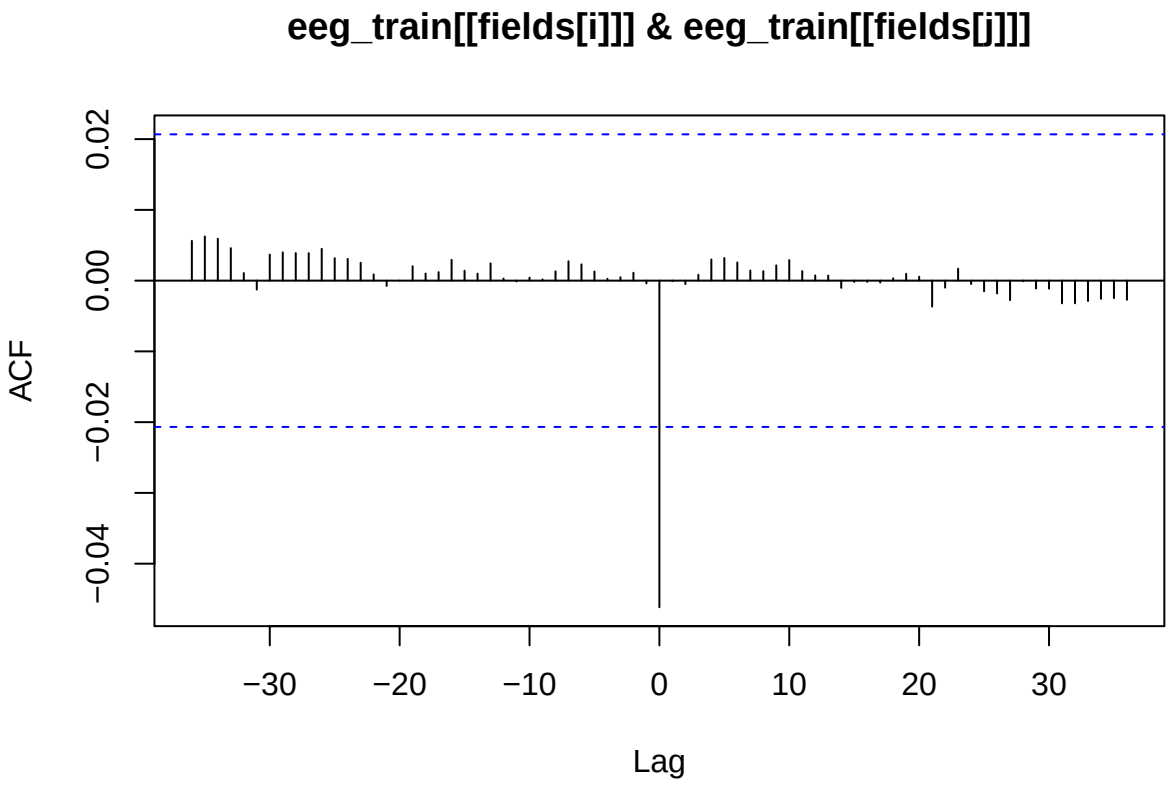
For example pairs **(FC5 , F7) , (O1,O2) , (FC6,T8)** have cross-correlation after watching the results.

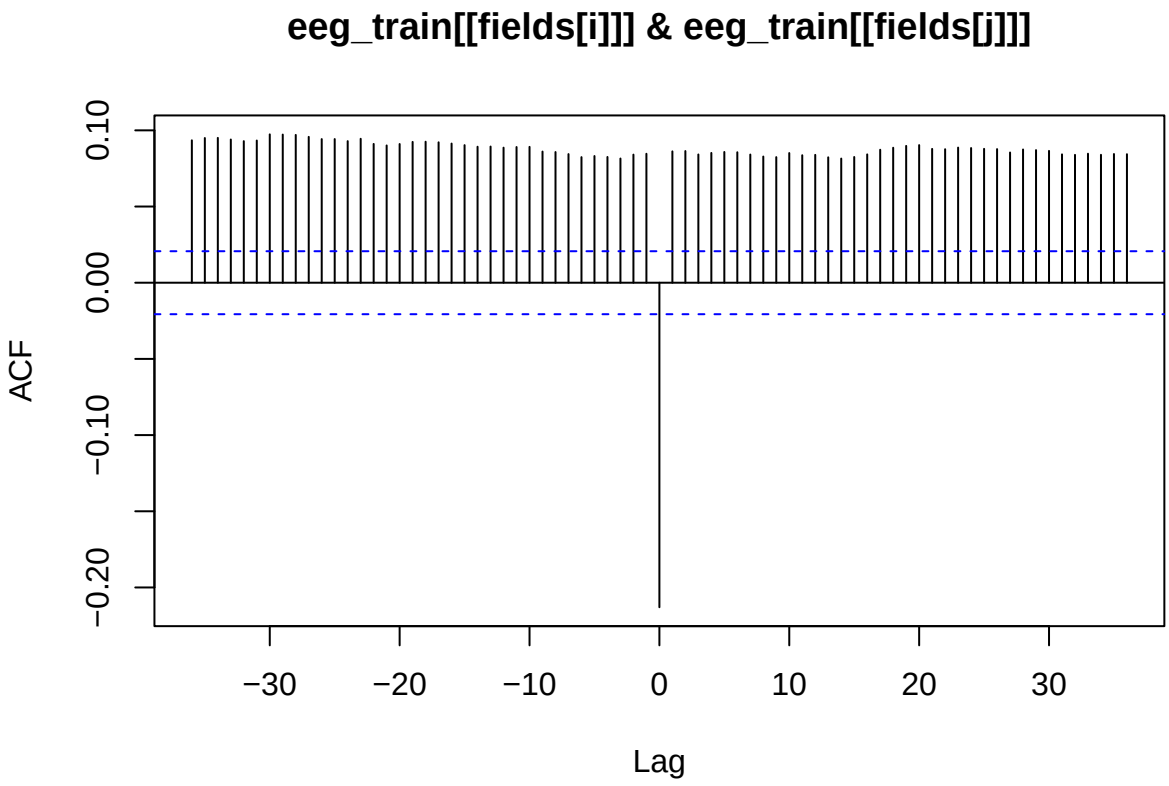
```
fields <- colnames(eeg_train)[-1]  # Exclude the first column (ds)
ccf_results <- list()

for (i in 1:(length(fields) - 1)) {
  for (j in (i + 1):length(fields)) {
    ccf_result <- ccf(eeg_train[[fields[i]]], eeg_train[[fields[j]]])
    ccf_results[[paste(fields[i], fields[j], sep = "-")]] <- ccf_result
  }
}
```

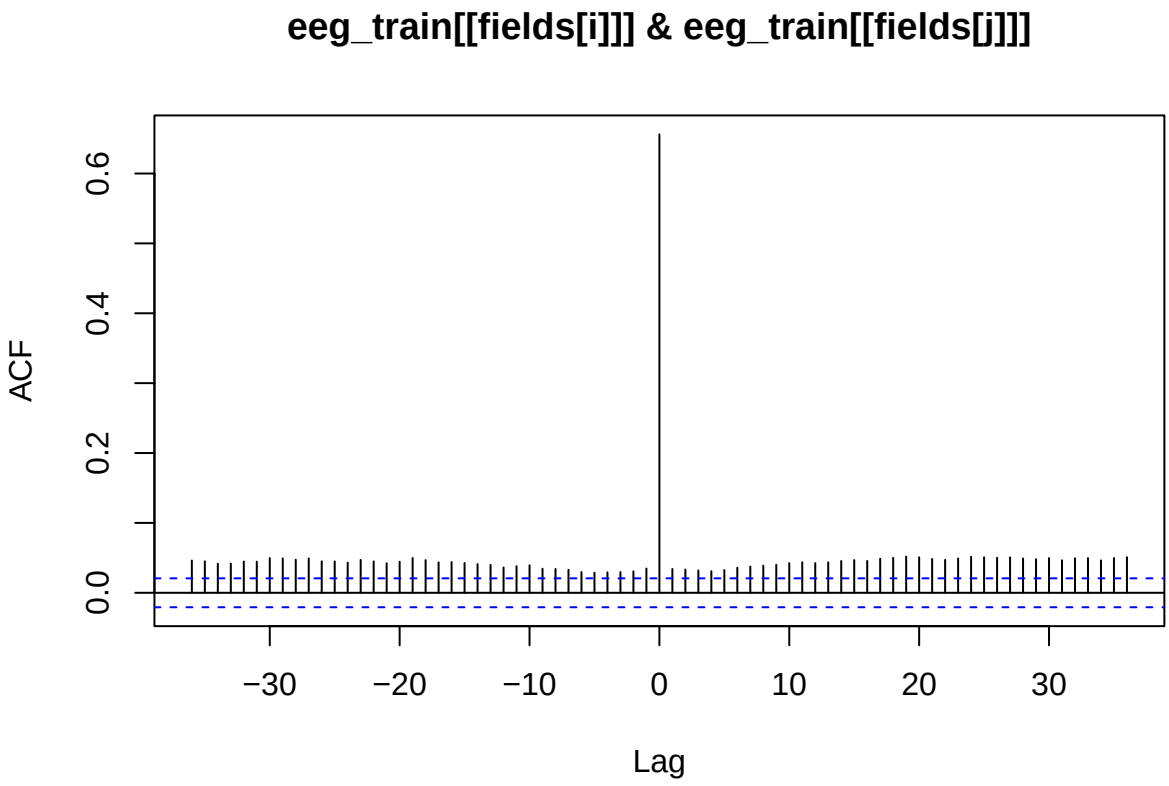


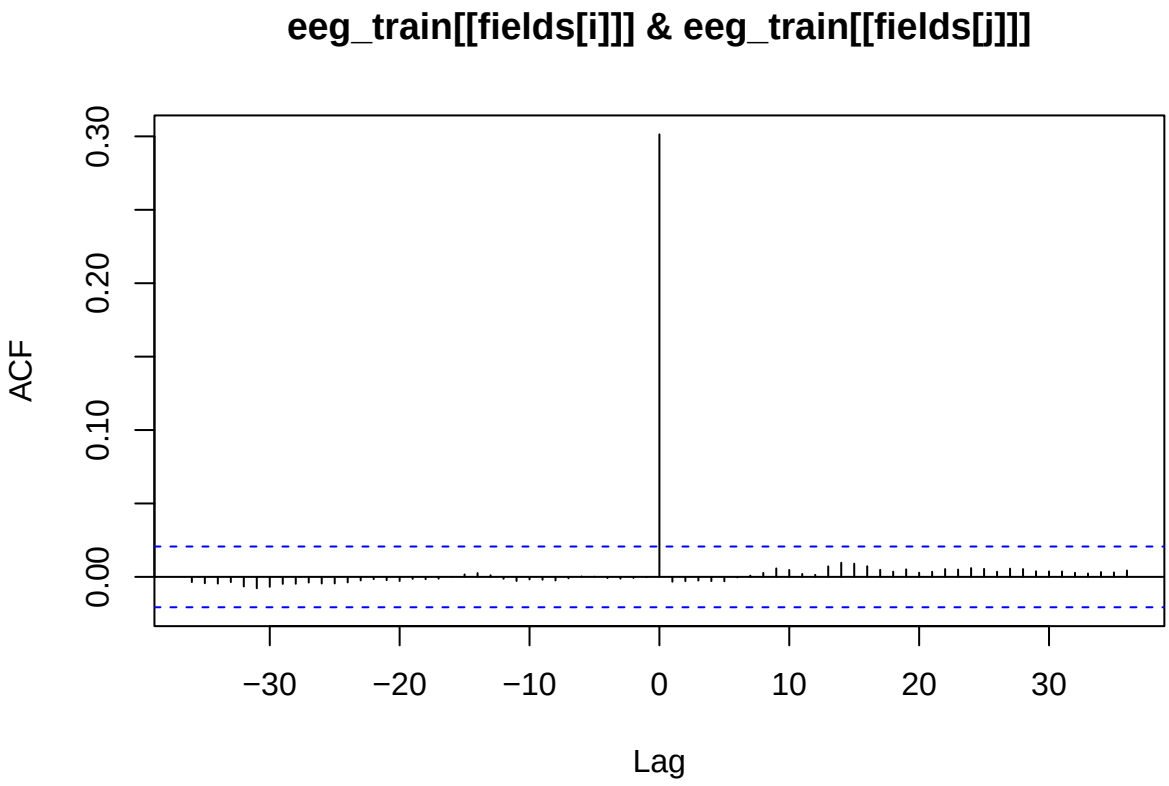

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.10
0.20
−30
−20
−10
0
10
20
30
Lag

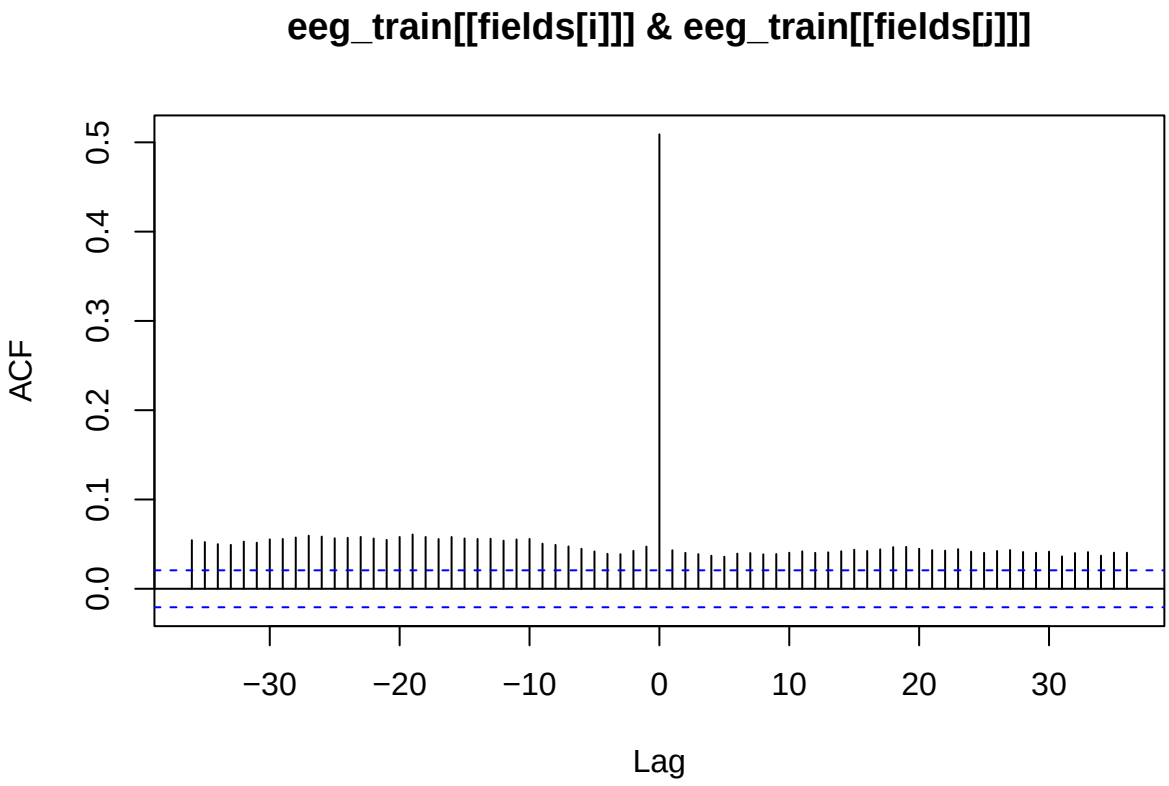

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.1
0.0
−0.1
−0.2
−0.3
−30
−20
−10
0
10
20
30
Lag

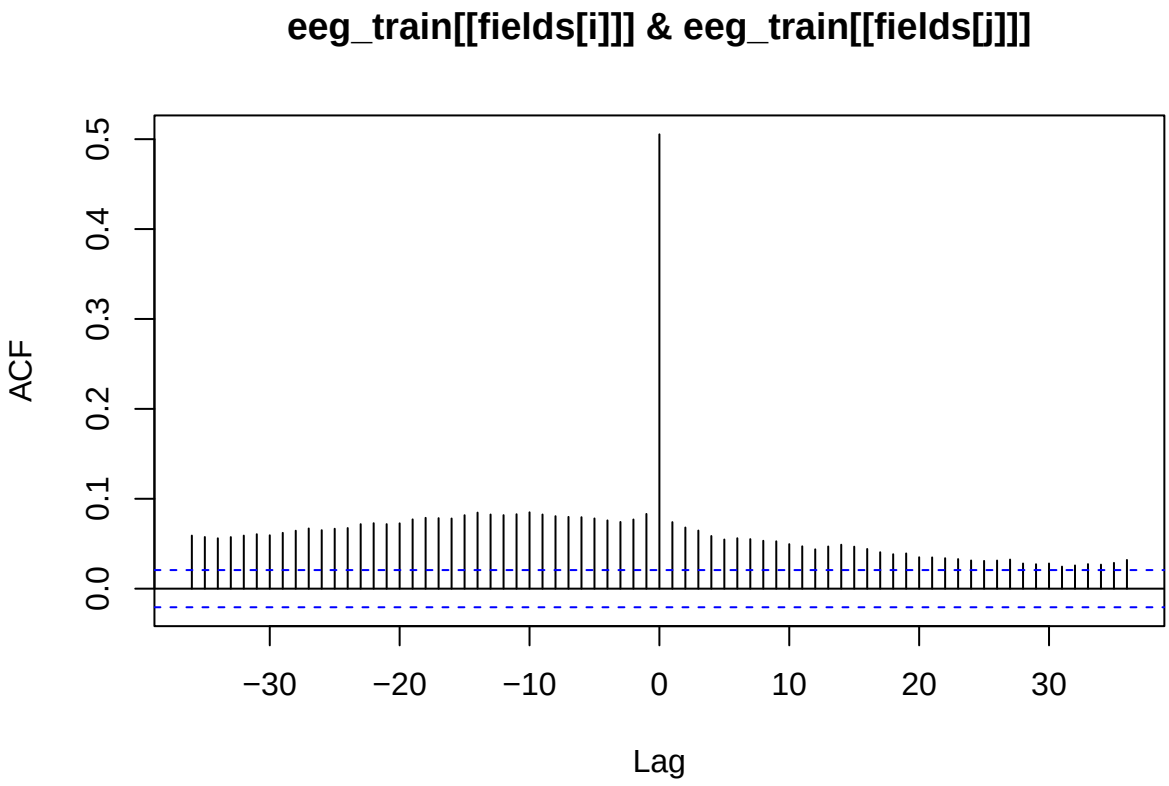

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.00
−0.02
−0.04
−30
−20
−10
0
10
20
30
Lag

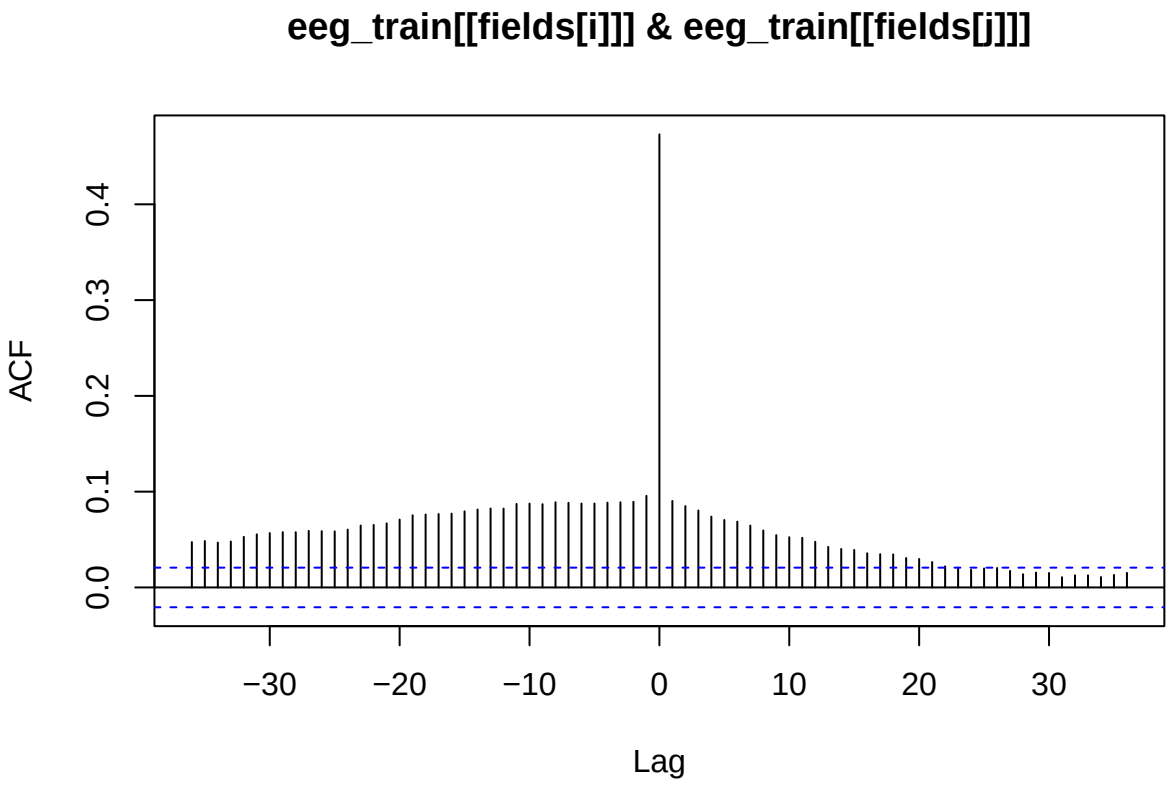

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.10
0.00
−0.10
−0.20
−30
−20
−10
0
10
20
30
Lag

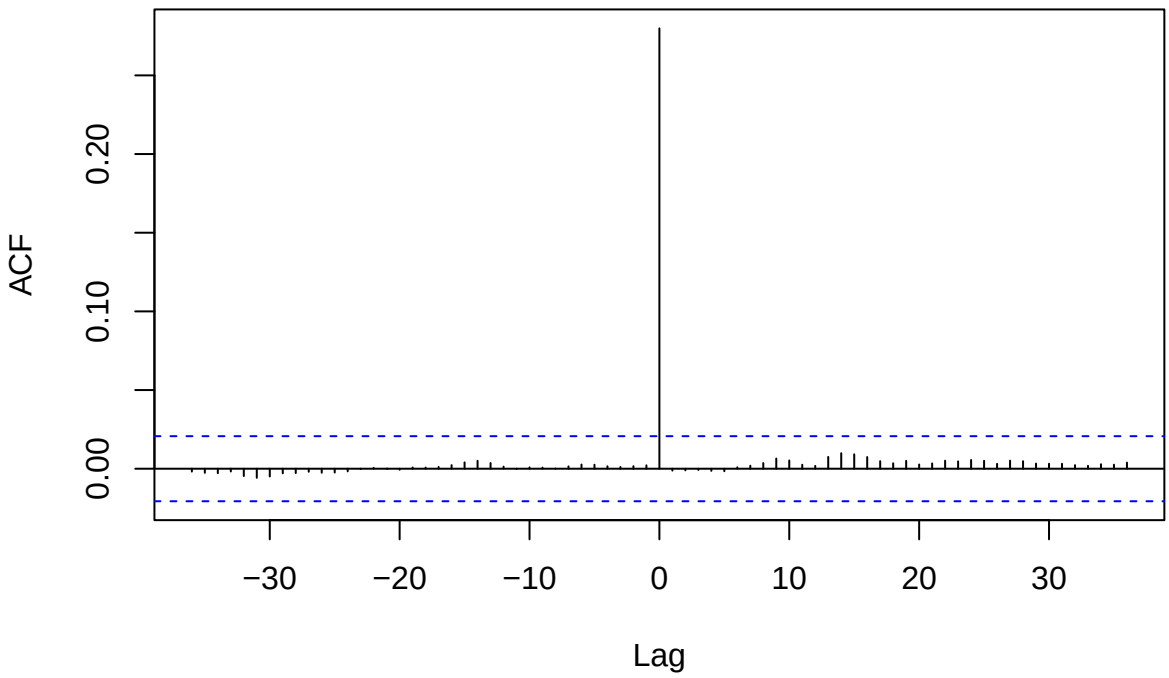

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.2
0.4
0.6
−30
−20
−10
0
10
20
30
Lag

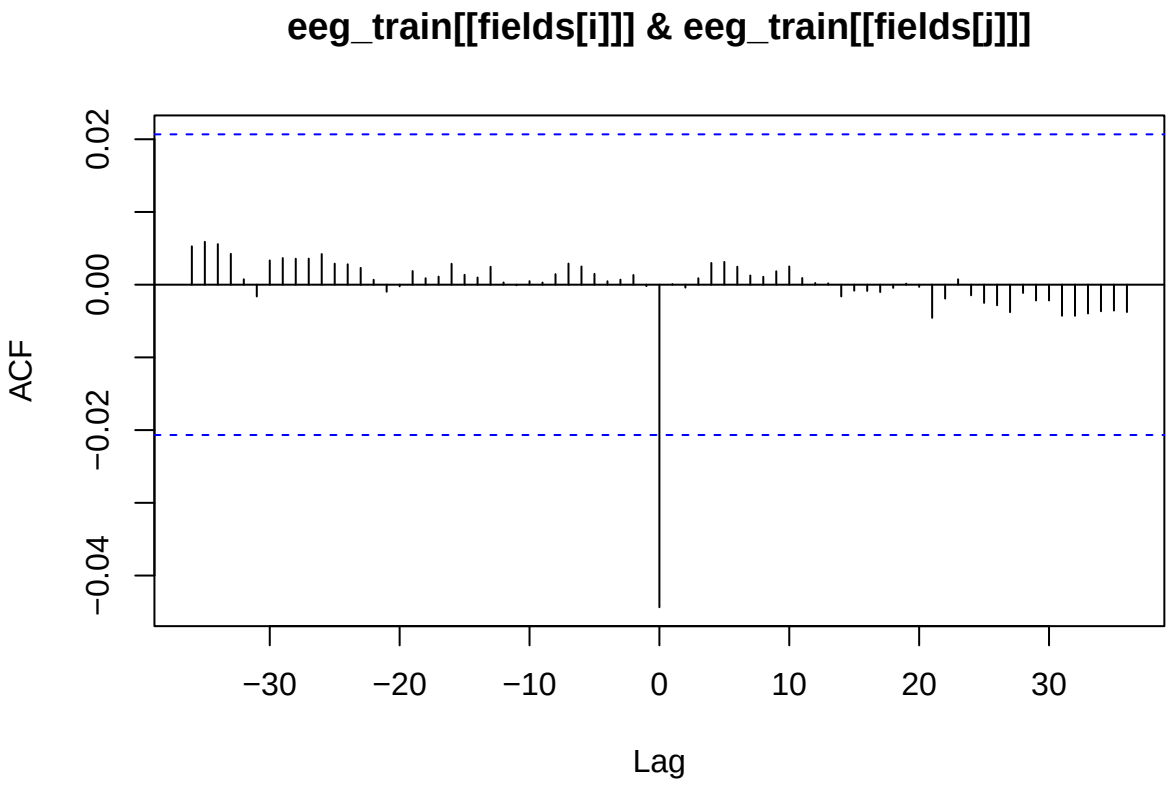

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.10
0.20
0.30
−30
−20
−10
0
10
20
30
Lag

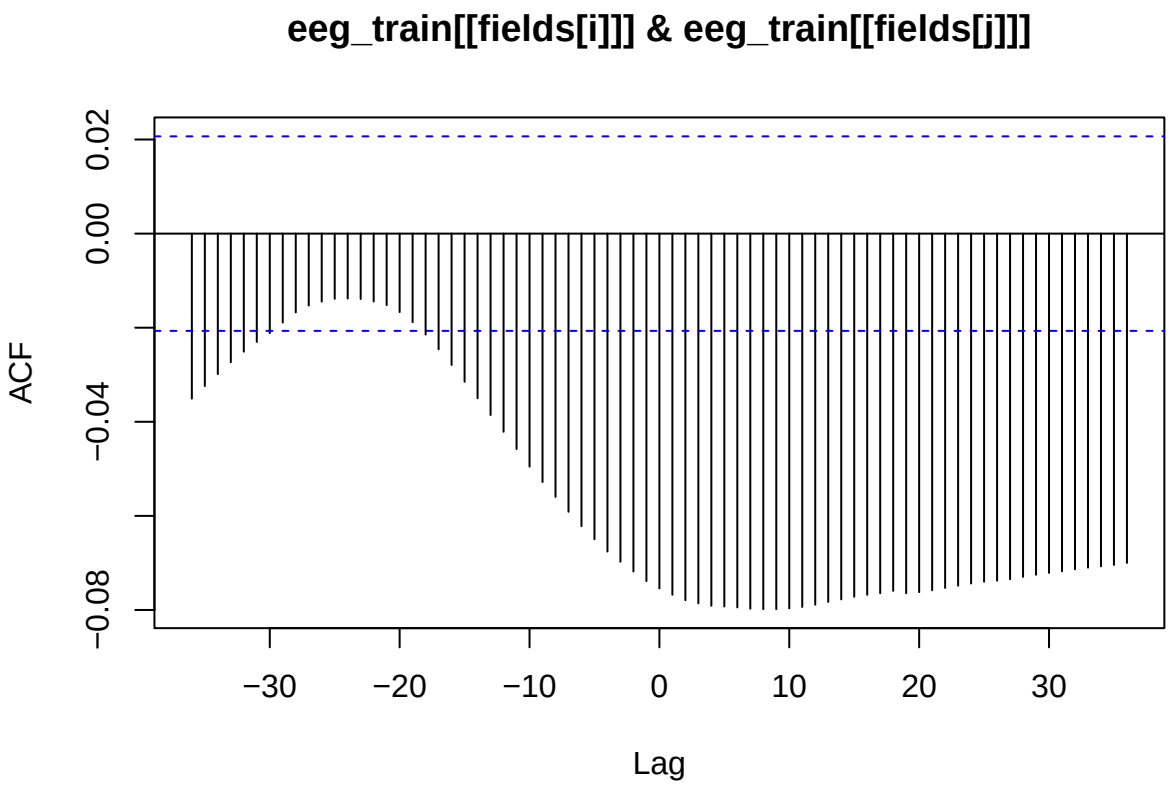

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
0.5
−30
−20
−10
0
10
20
30
Lag

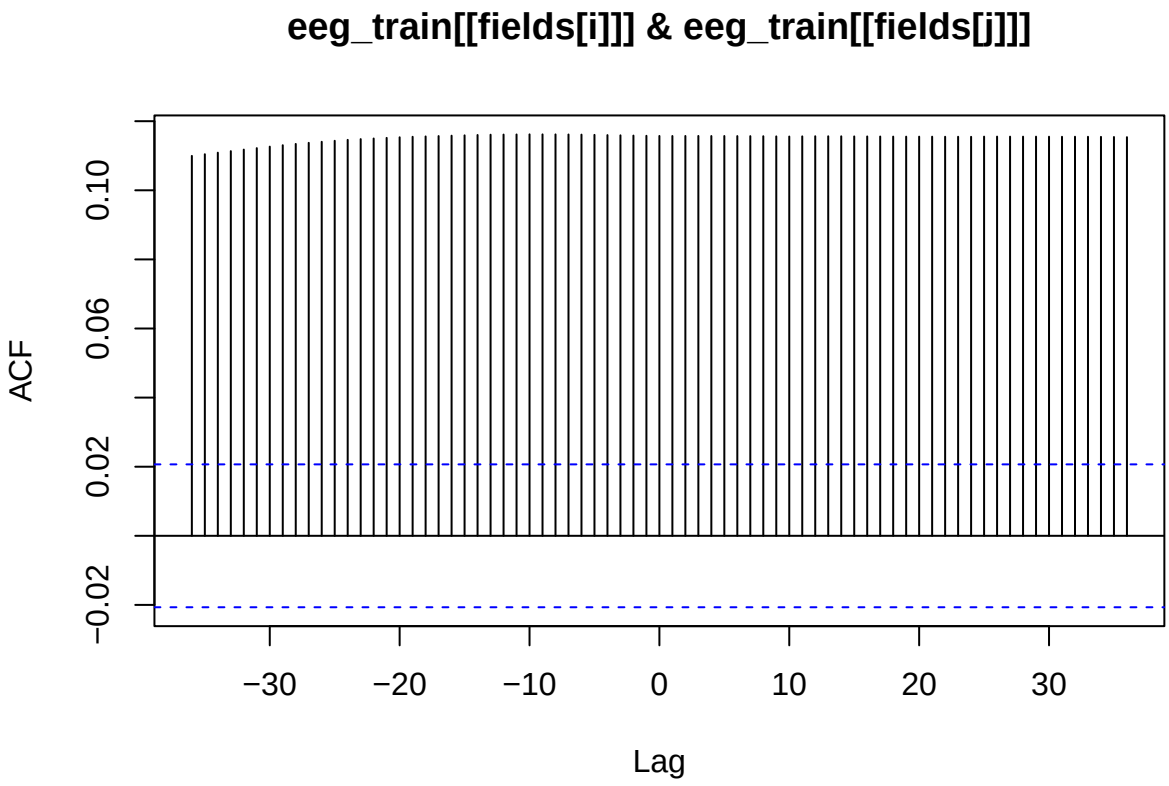

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
0.5
−30
−20
−10
0
10
20
30
Lag

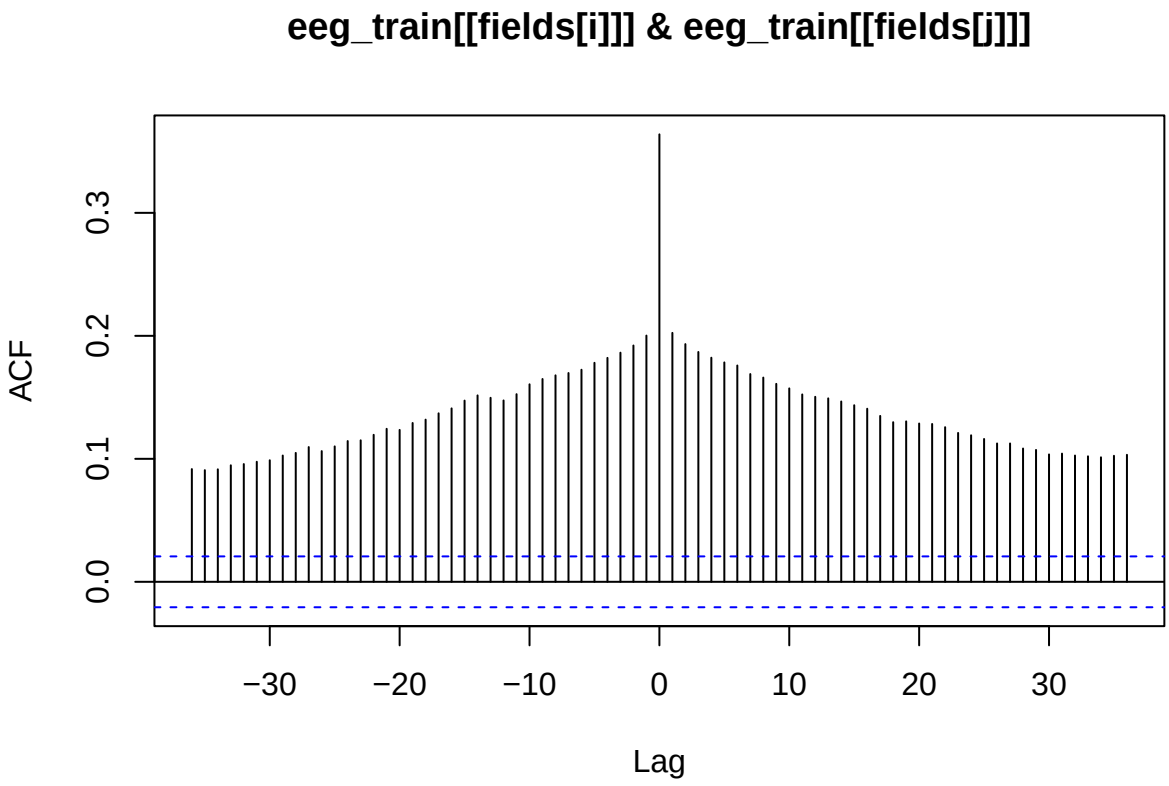

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
−30
−20
−10
0
10
20
30
Lag

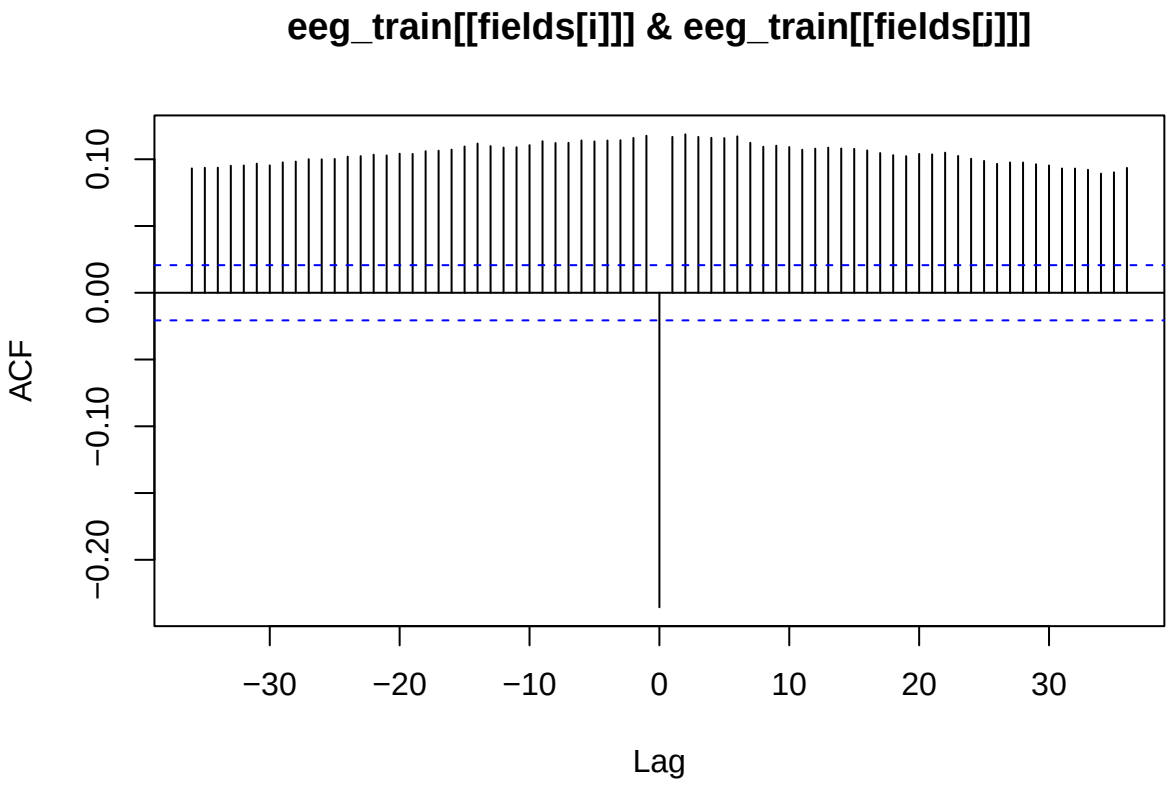

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.10
0.20
−30
−20
−10
0
10
20
30
Lag

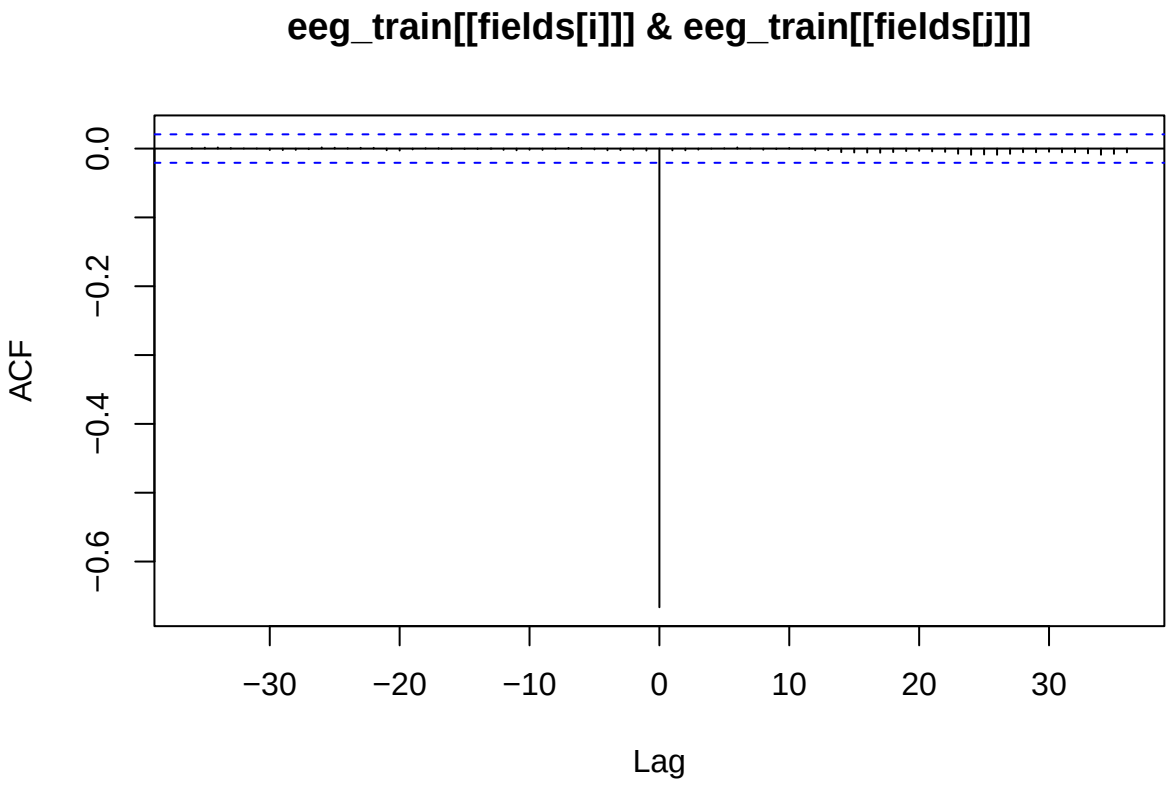

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.00
−0.02
−0.04
−30
−20
−10
0
10
20
30
Lag

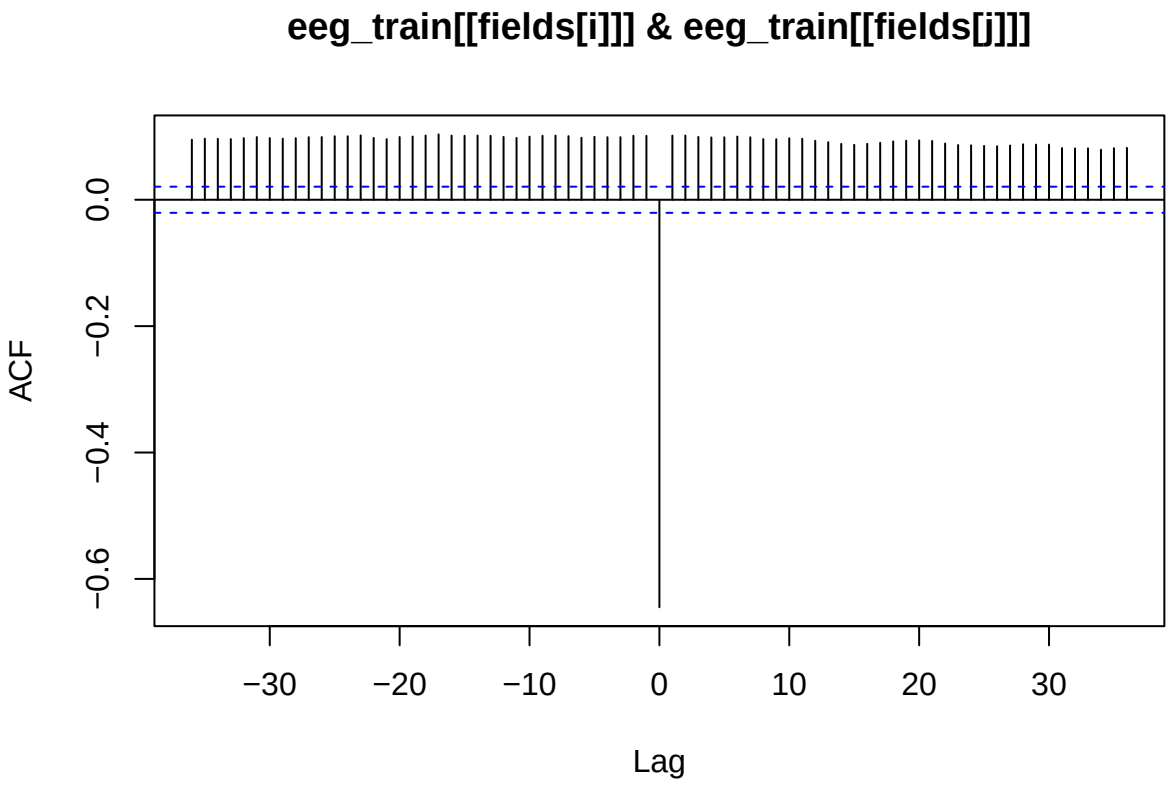

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.00
−0.04
−0.08
−30
−20
−10
0
10
20
30
Lag

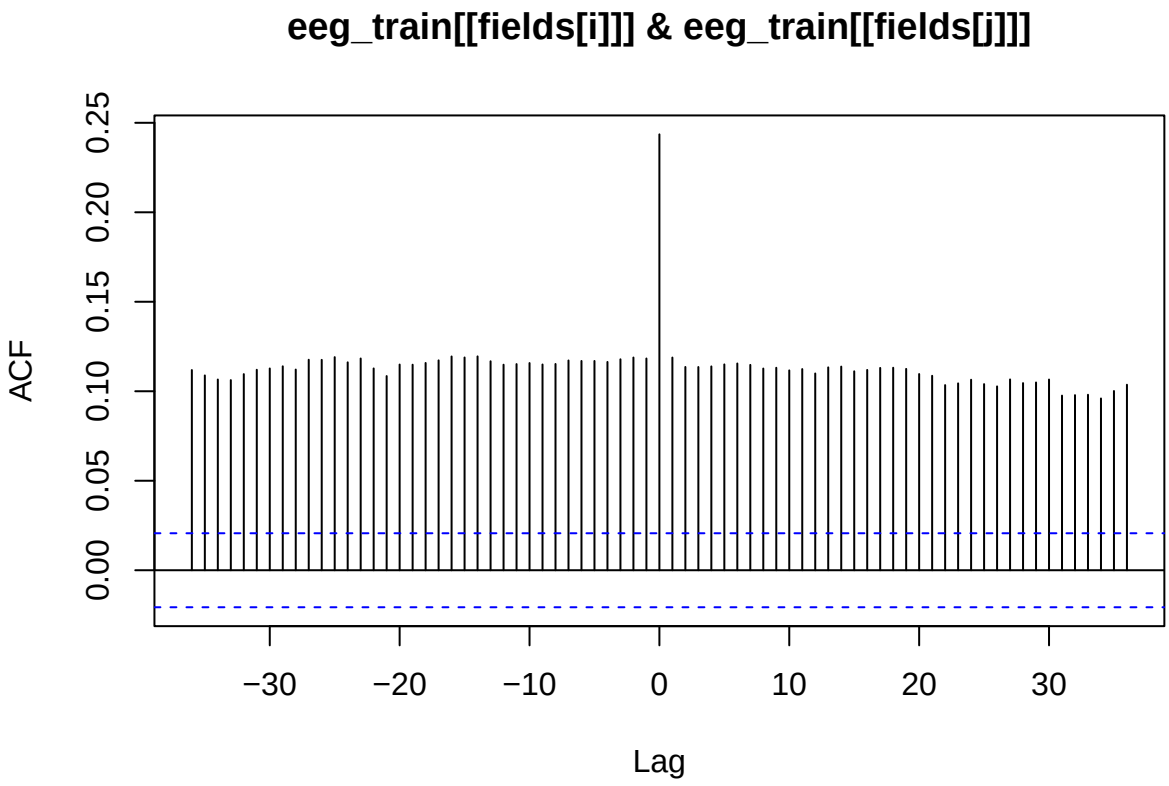

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.10
0.06
0.02
−0.02
−30
−20
−10
0
10
20
30
Lag

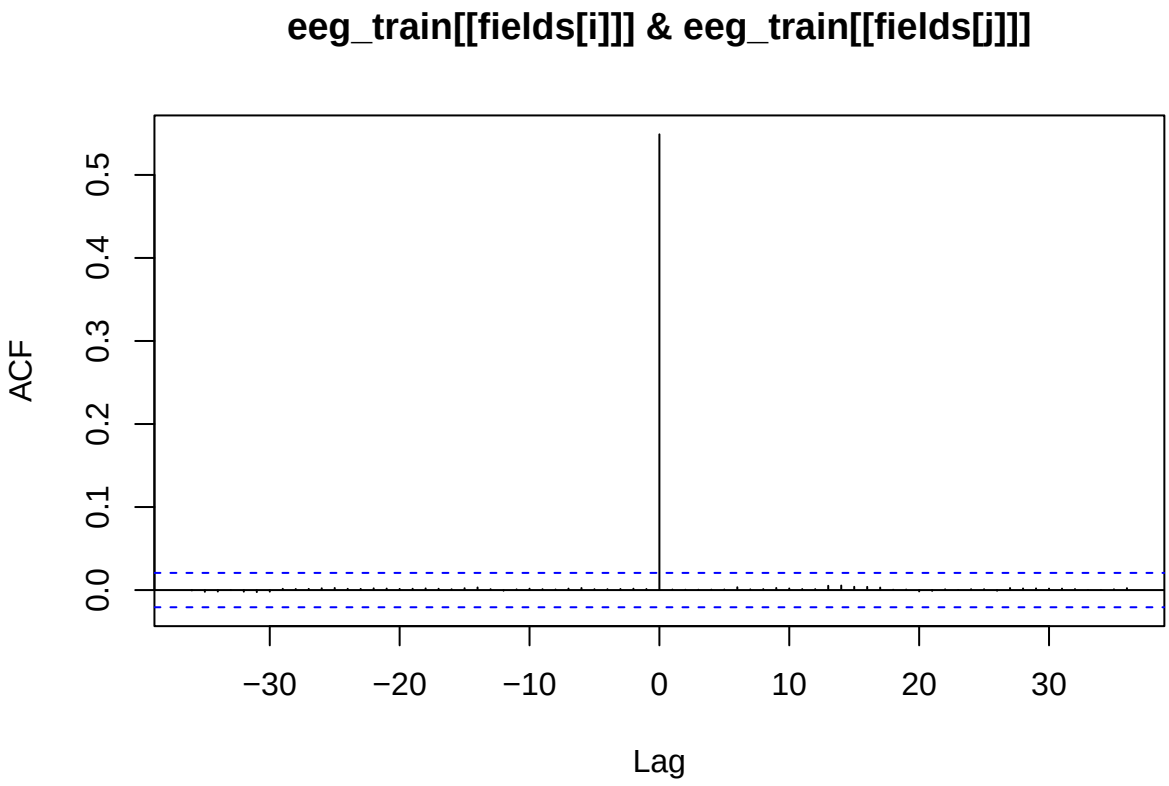

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
−30
−20
−10
0
10
20
30
Lag

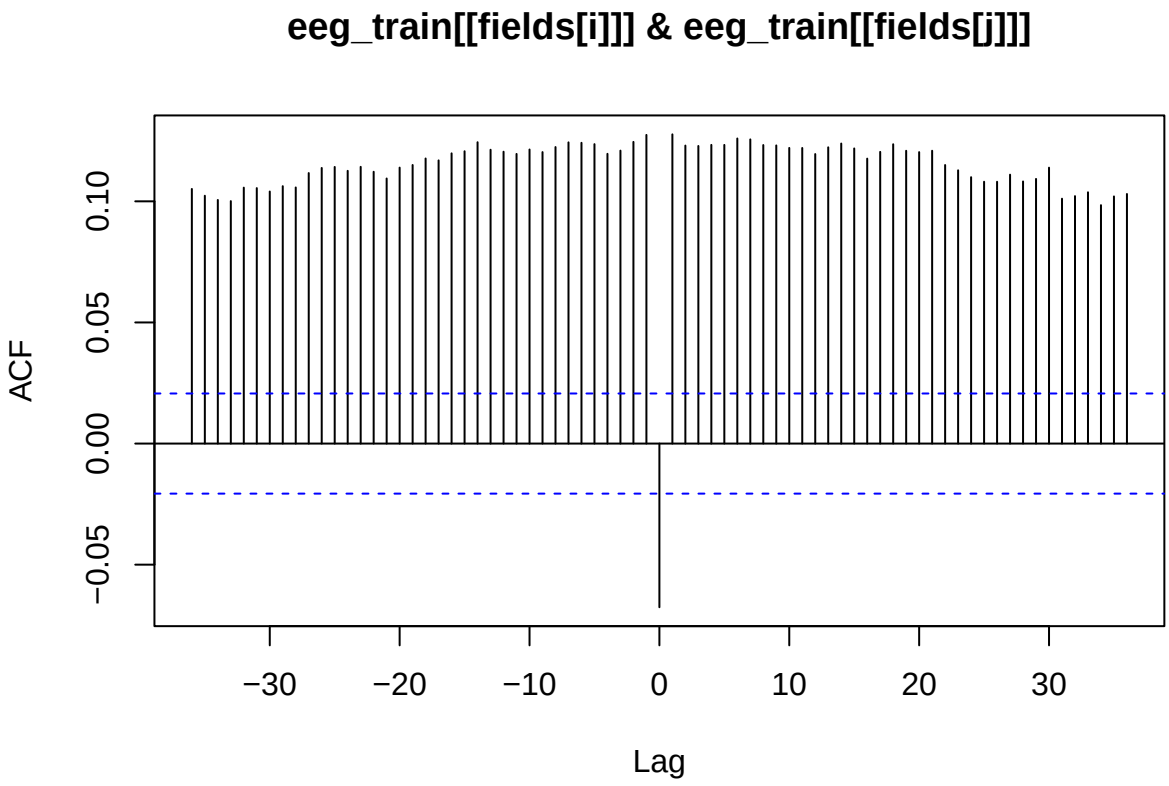

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.10
0.00
−0.10
−0.20
−30
−20
−10
0
10
20
30
Lag

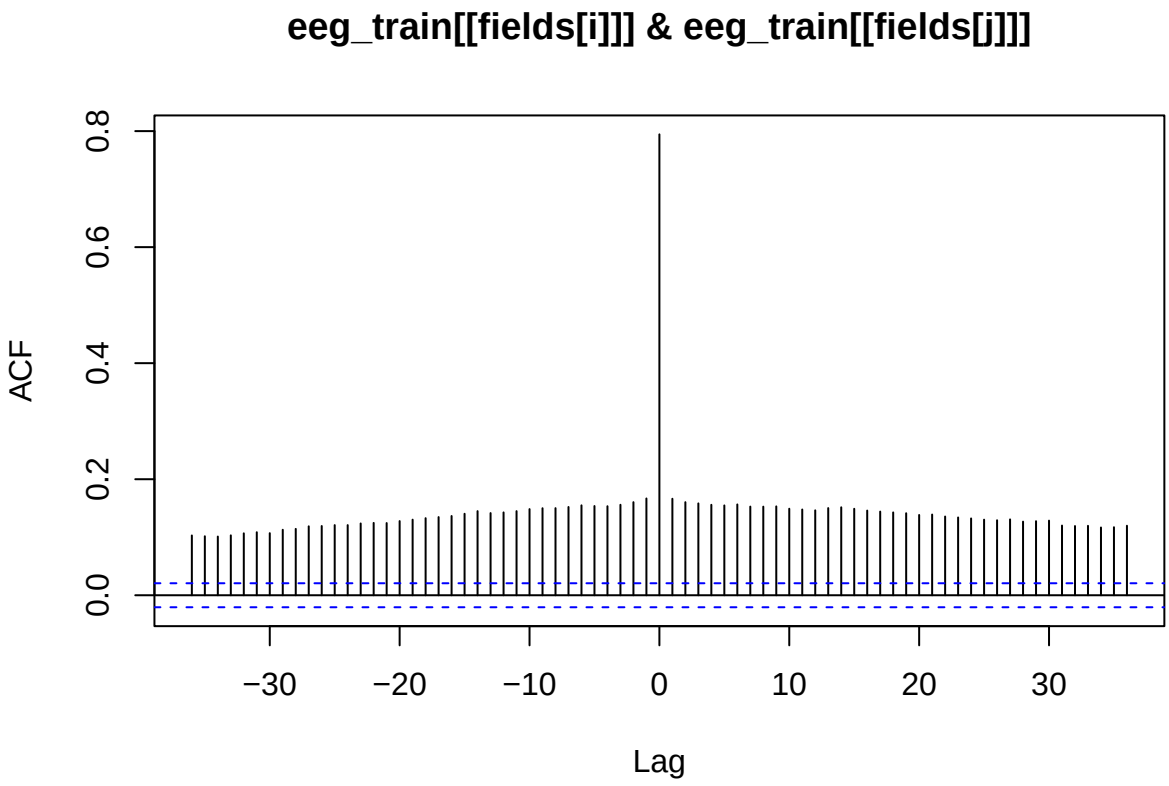

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
−0.2
−0.4
−0.6
−30
−20
−10
0
10
20
30
Lag

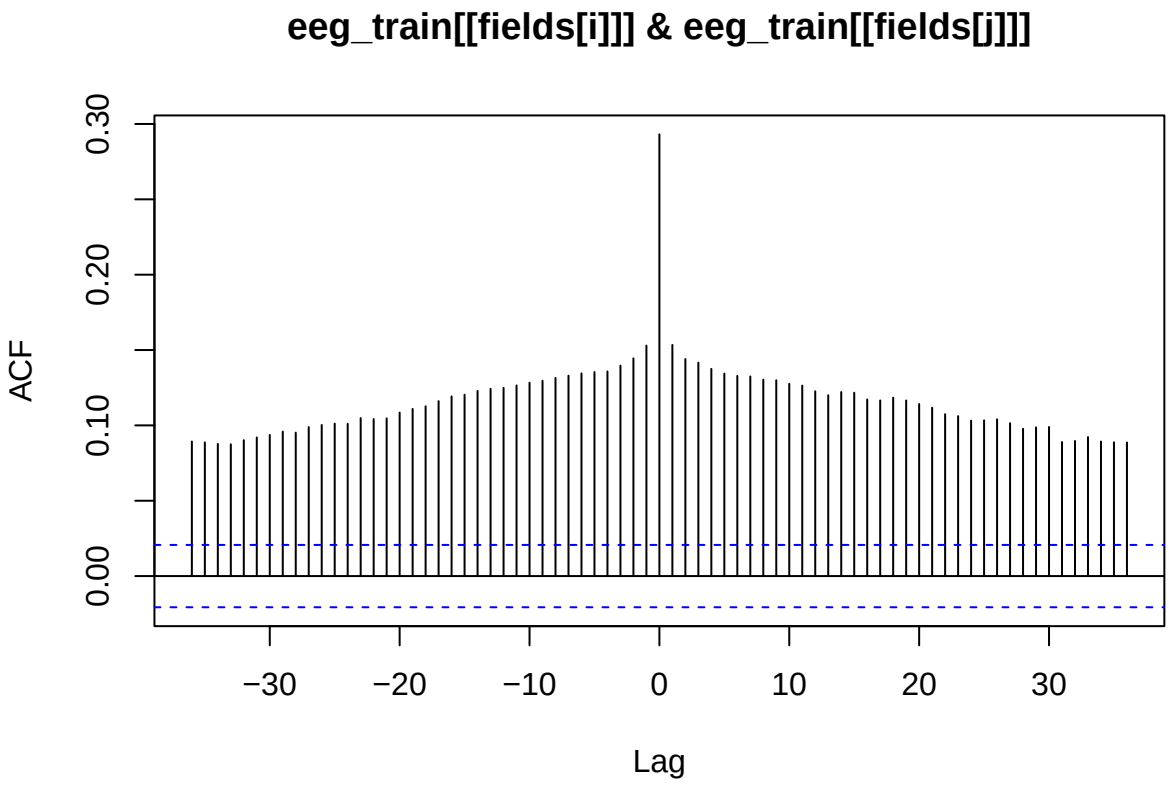

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
−0.2
−0.4
−0.6
−30
−20
−10
0
10
20
30
Lag

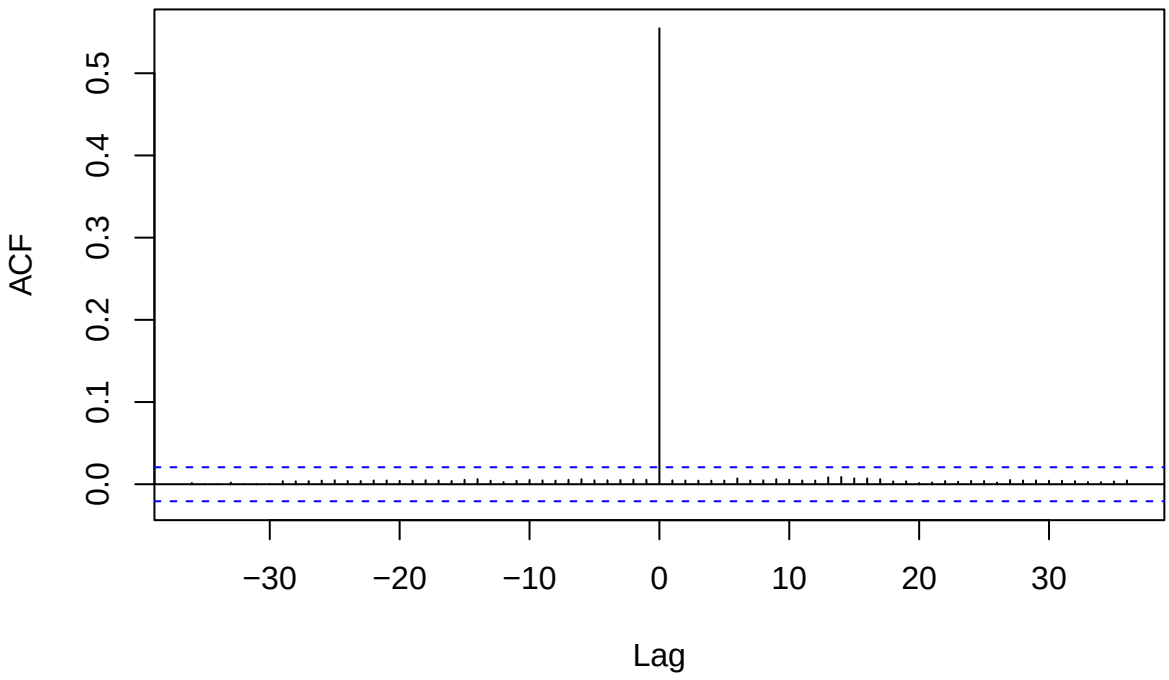

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.05
0.10
0.15
0.20
0.25
−30
−20
−10
0
10
20
30
Lag

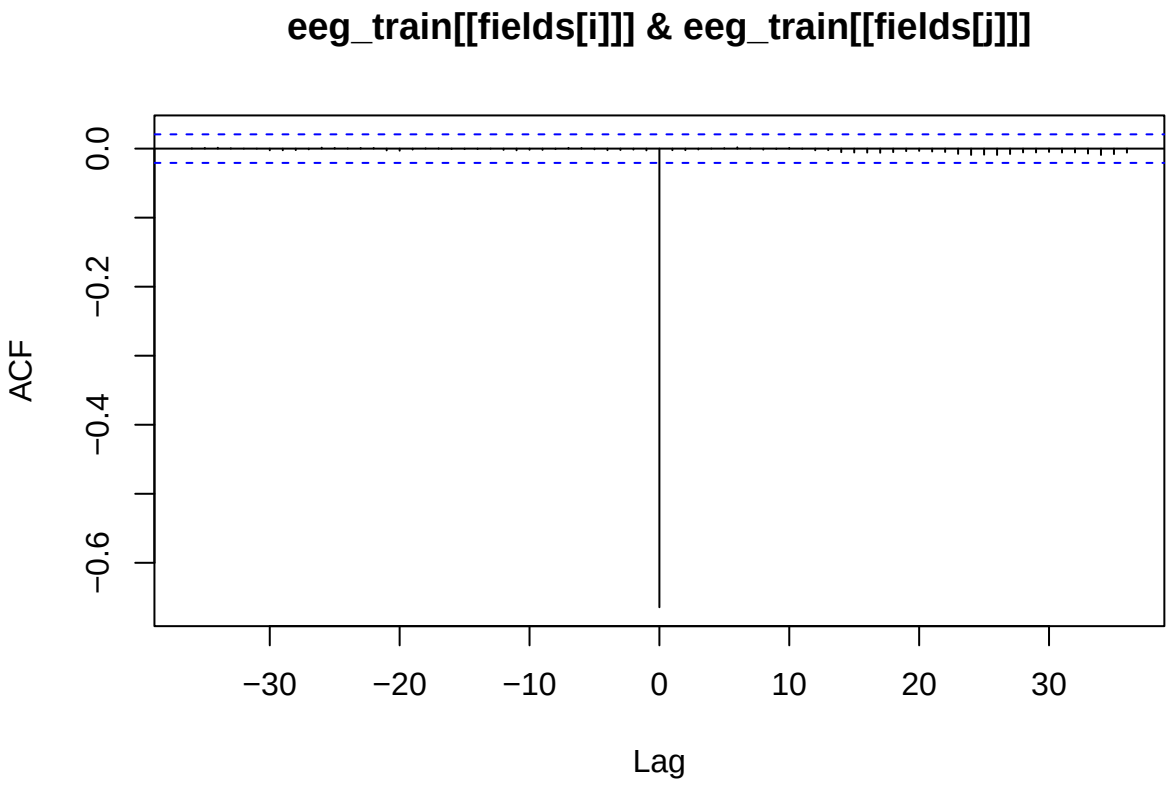

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
0.5
−30
−20
−10
0
10
20
30
Lag

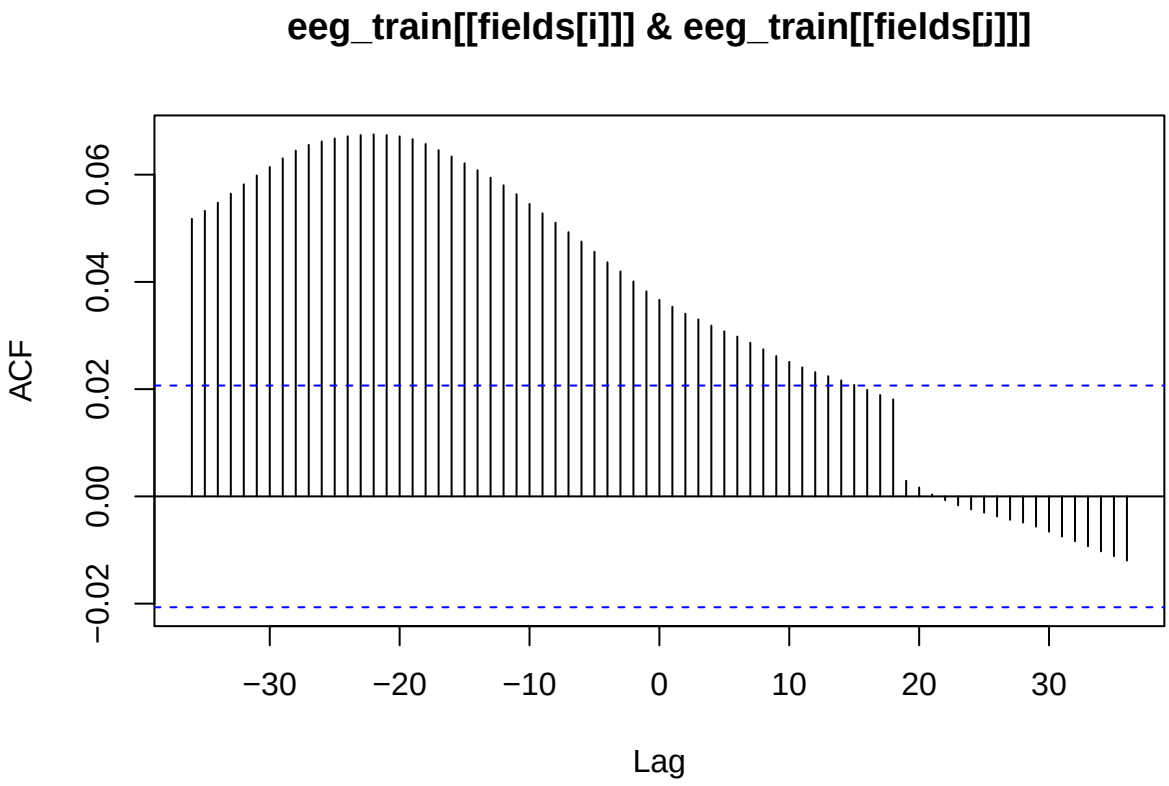

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.10
0.05
0.00
−0.05
−30
−20
−10
0
10
20
30
Lag

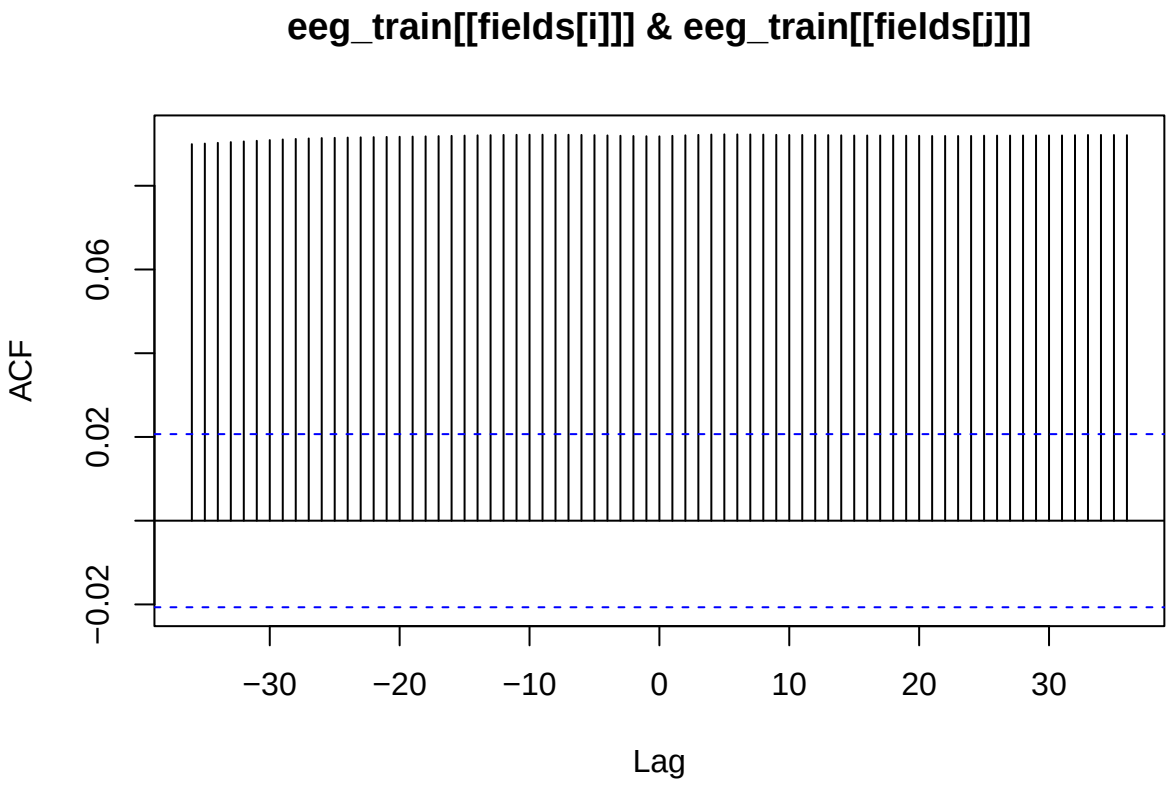

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.2
0.4
0.6
0.8
−30
−20
−10
0
10
20
30
Lag

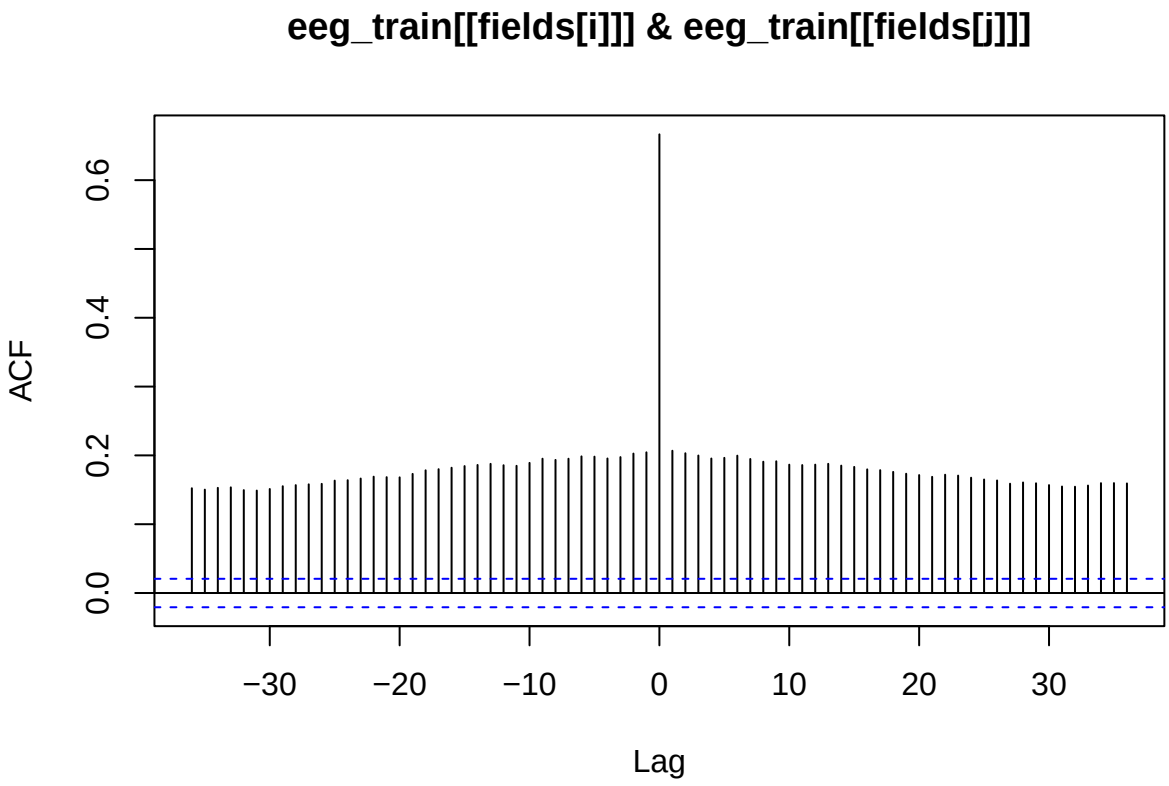

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.10
0.20
0.30
−30
−20
−10
0
10
20
30
Lag

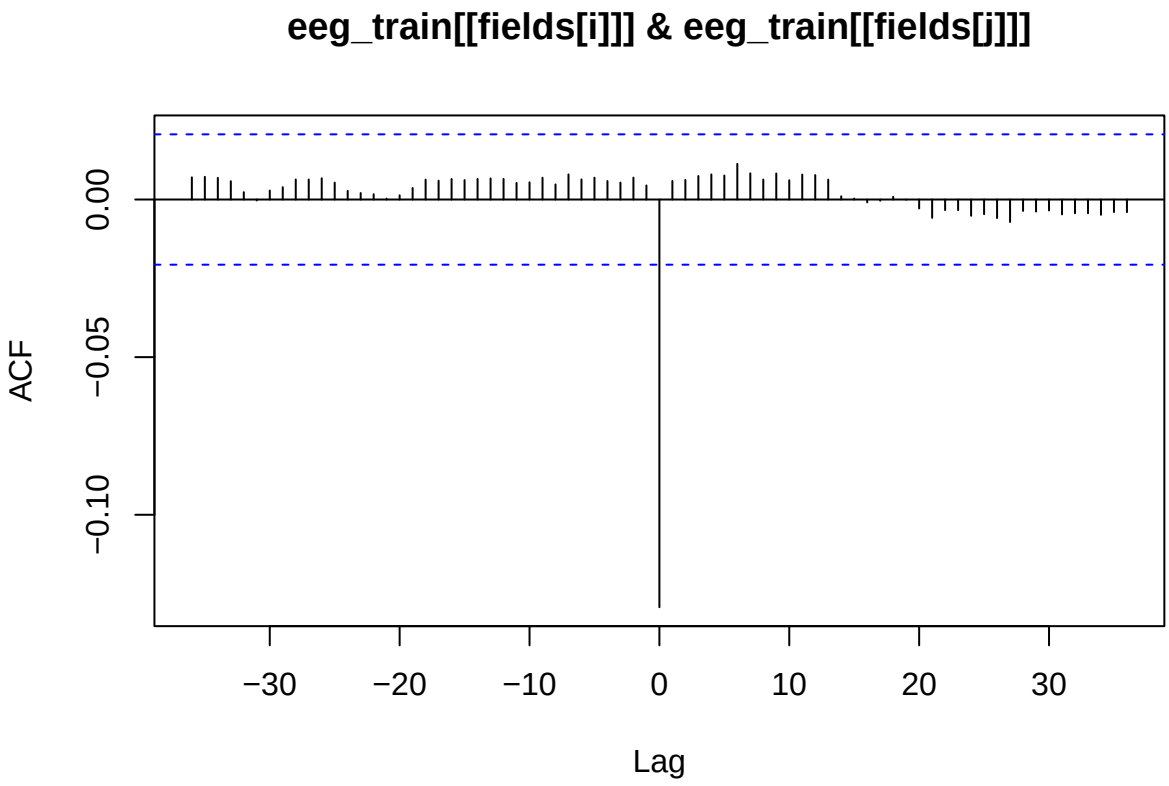

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
0.5
−30
−20
−10
0
10
20
30
Lag

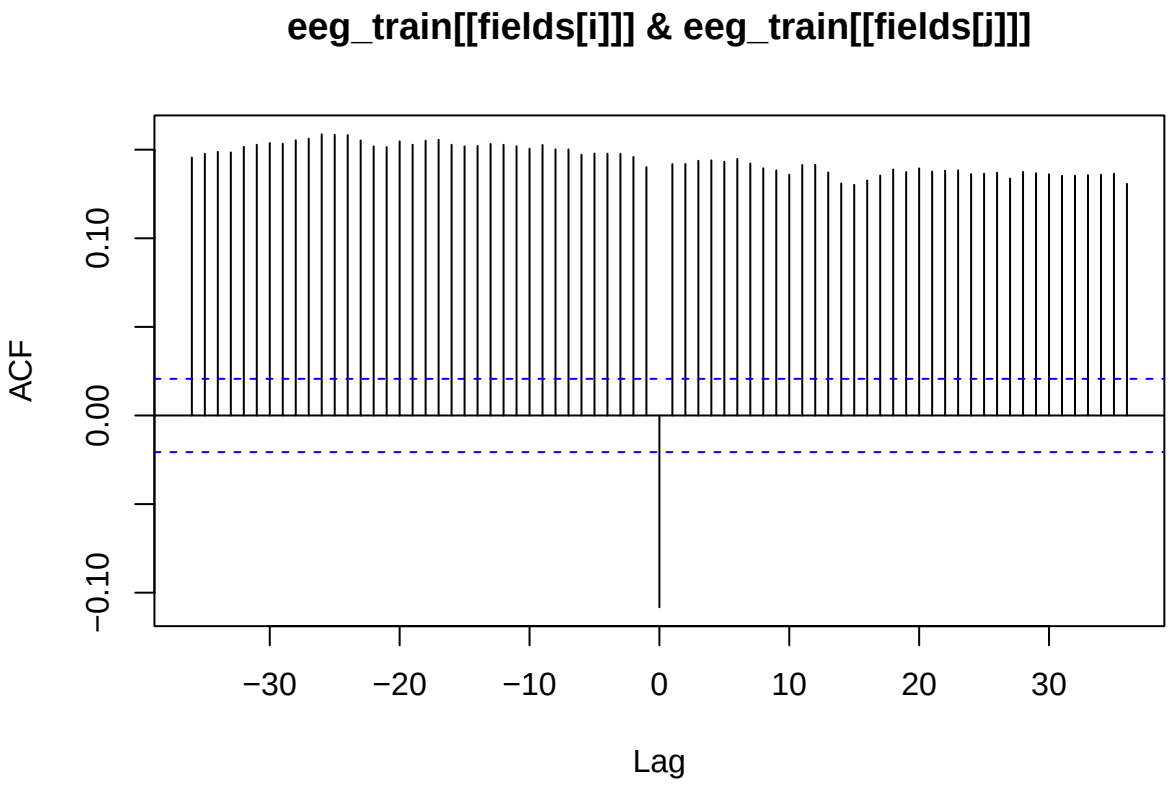

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
−0.2
−0.4
−0.6
−30
−20
−10
0
10
20
30
Lag

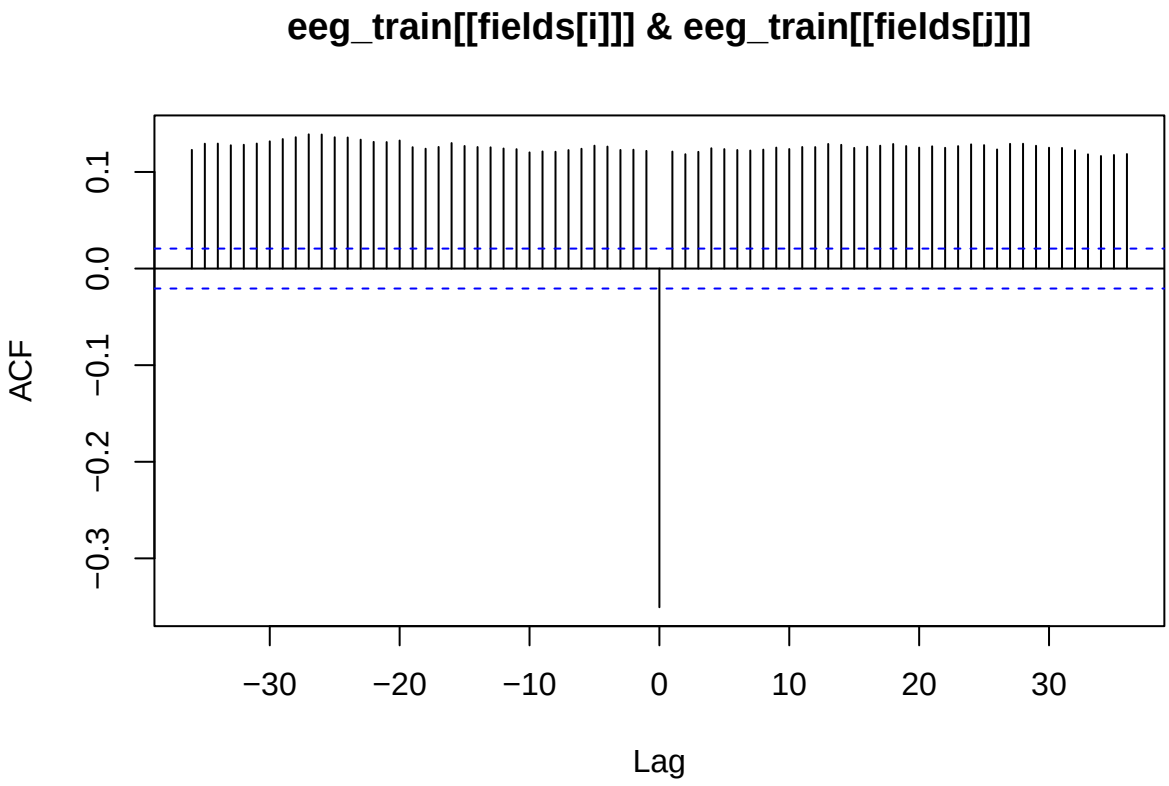

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.06
0.04
0.02
0.00
−0.02
−30
−20
−10
0
10
20
30
Lag

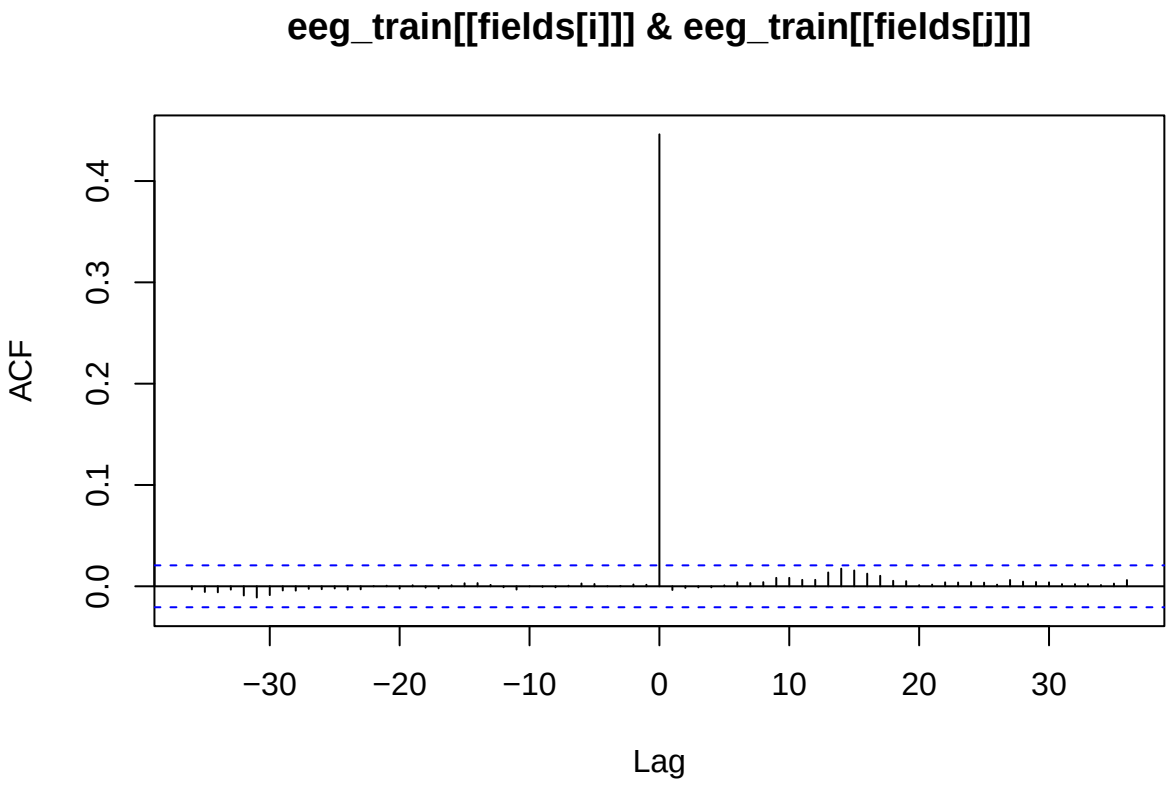

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.06
0.02
−0.02
−30
−20
−10
0
10
20
30
Lag

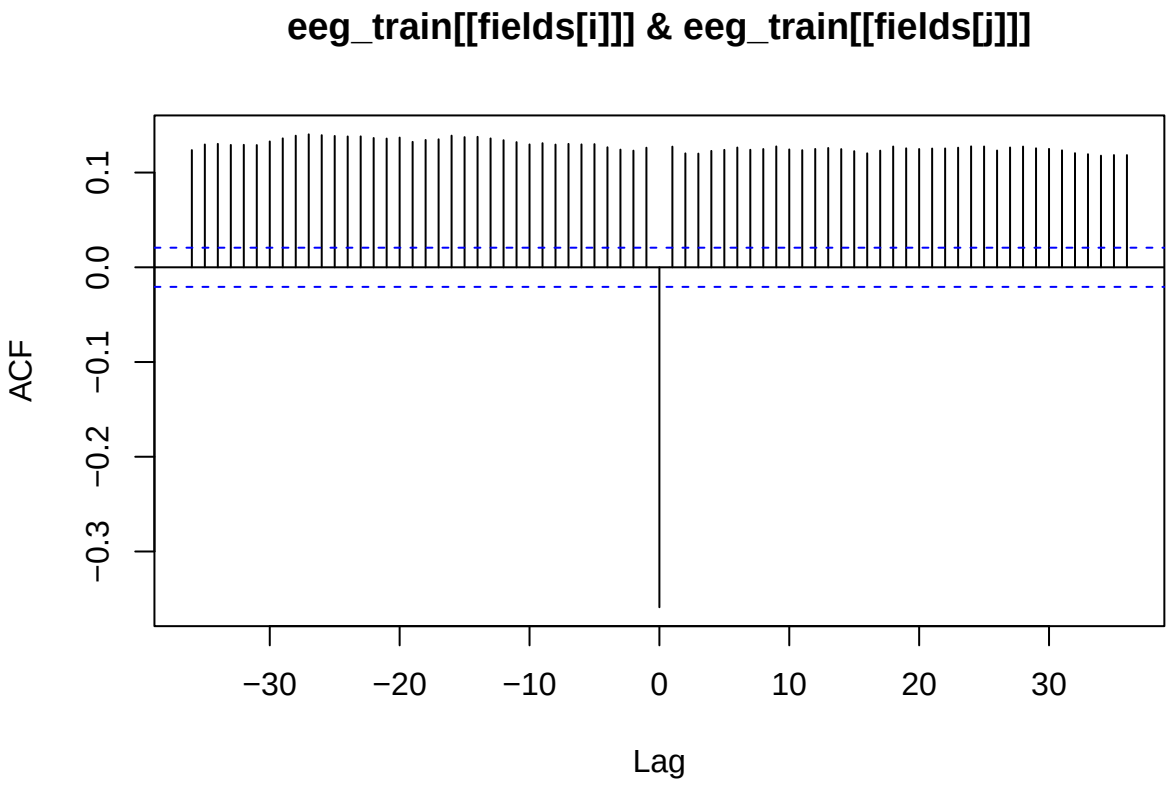

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.2
0.4
0.6
−30
−20
−10
0
10
20
30
Lag

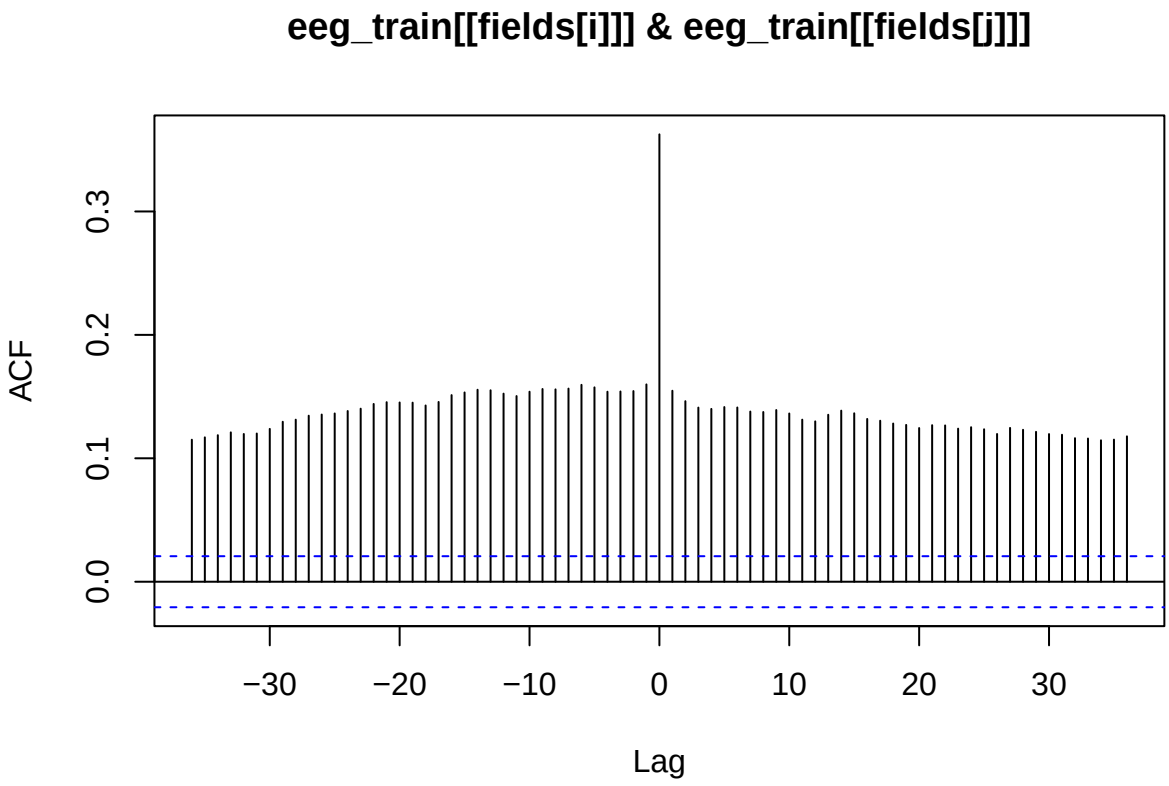

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
−0.05
−0.10
−30
−20
−10
0
10
20
30
Lag

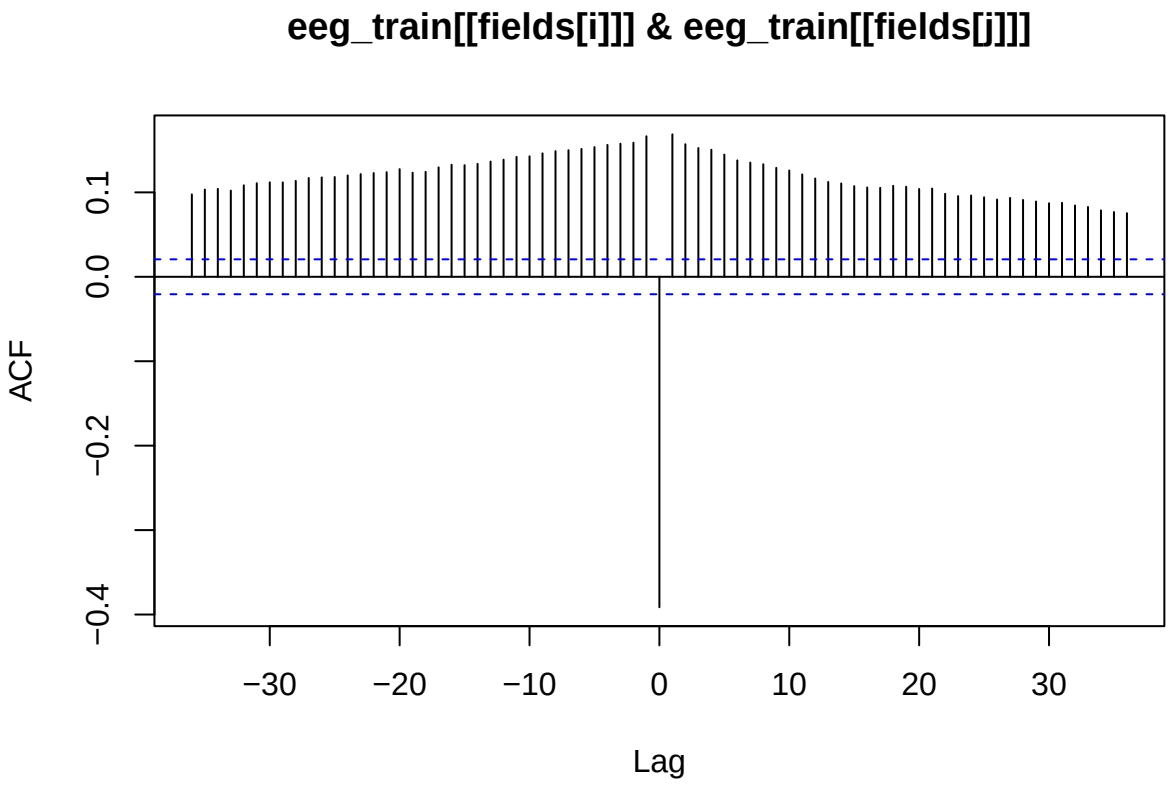

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.10
0.00
−0.10
−30
−20
−10
0
10
20
30
Lag

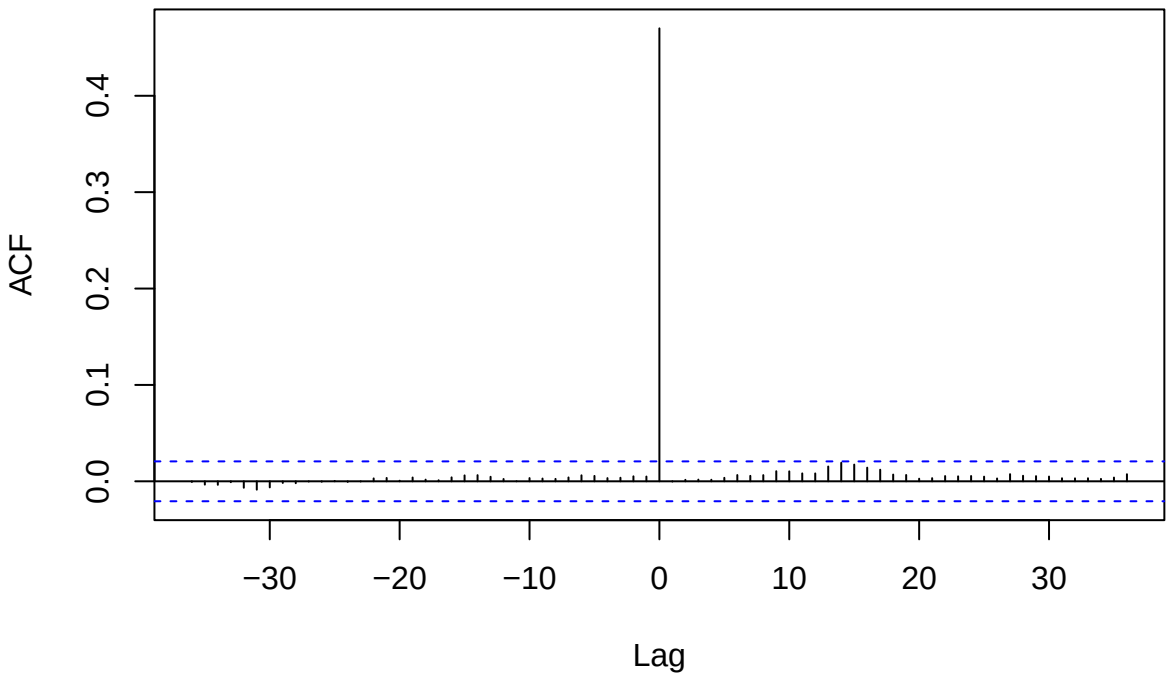

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.1
0.0
−0.1
−0.2
−0.3
−30
−20
−10
0
10
20
30
Lag

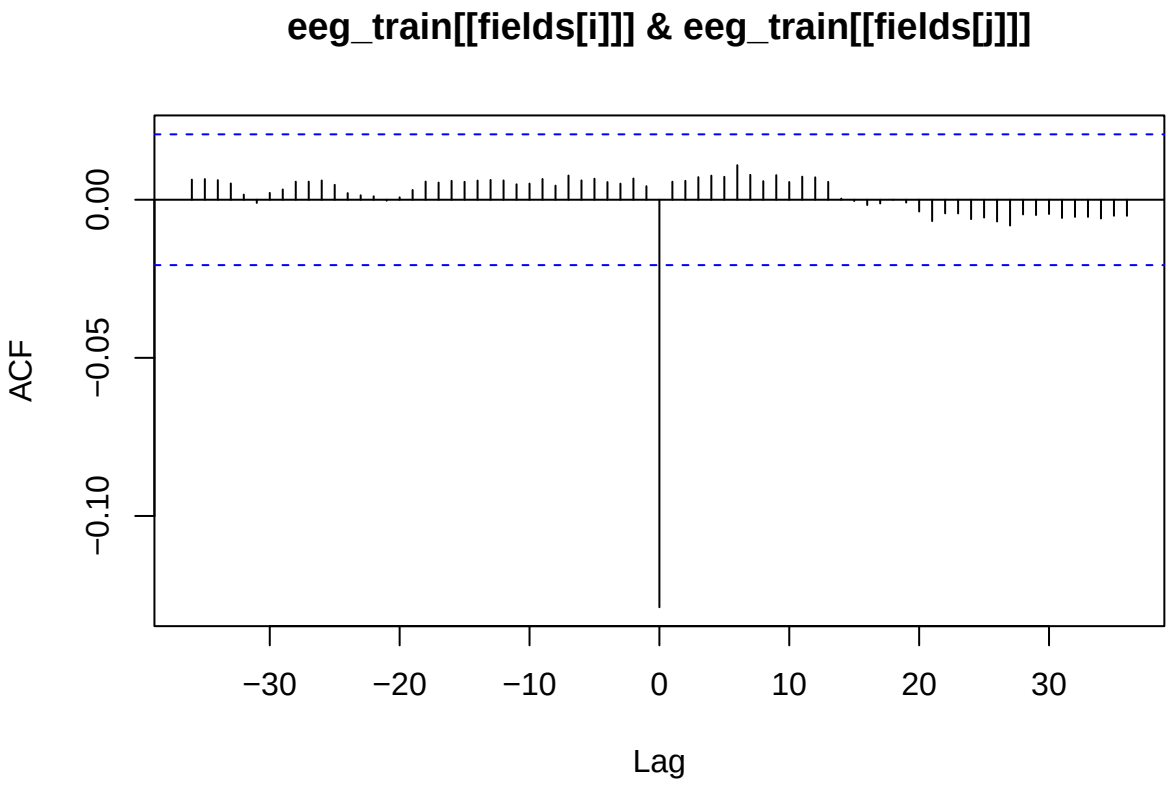

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
−30
−20
−10
0
10
20
30
Lag

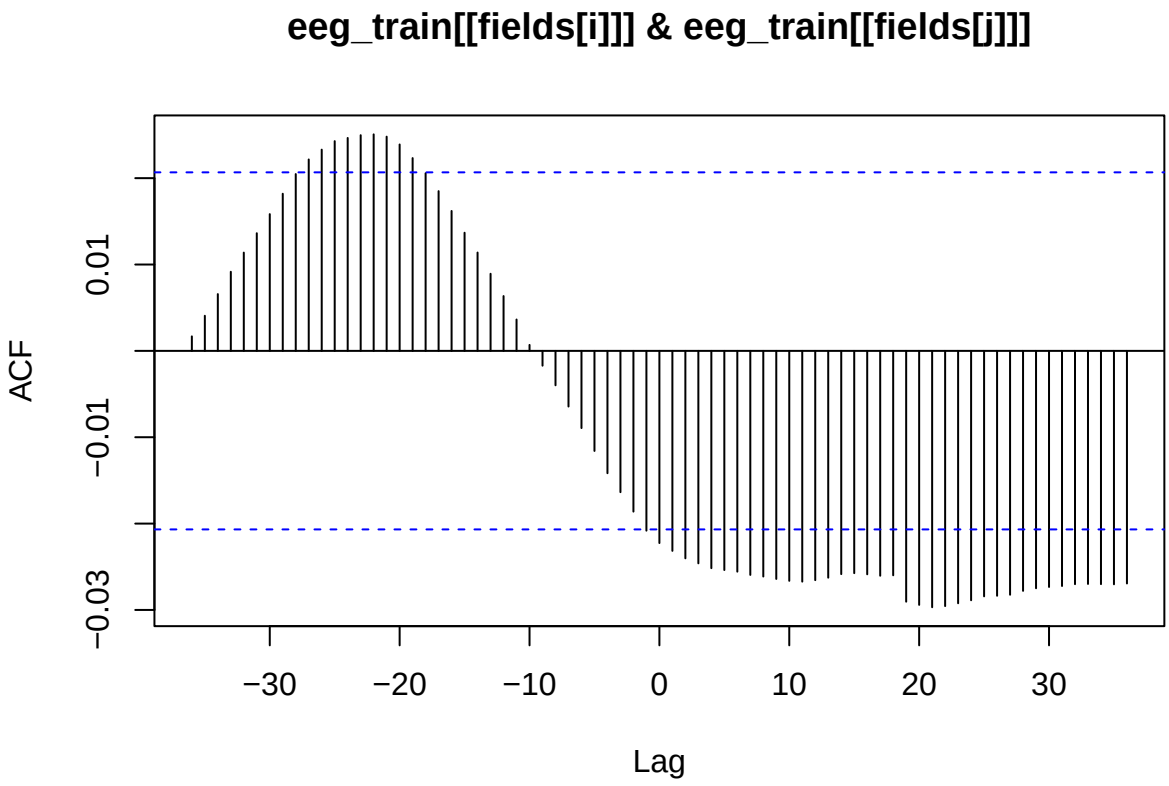

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.1
0.0
−0.1
−0.2
−0.3
−30
−20
−10
0
10
20
30
Lag

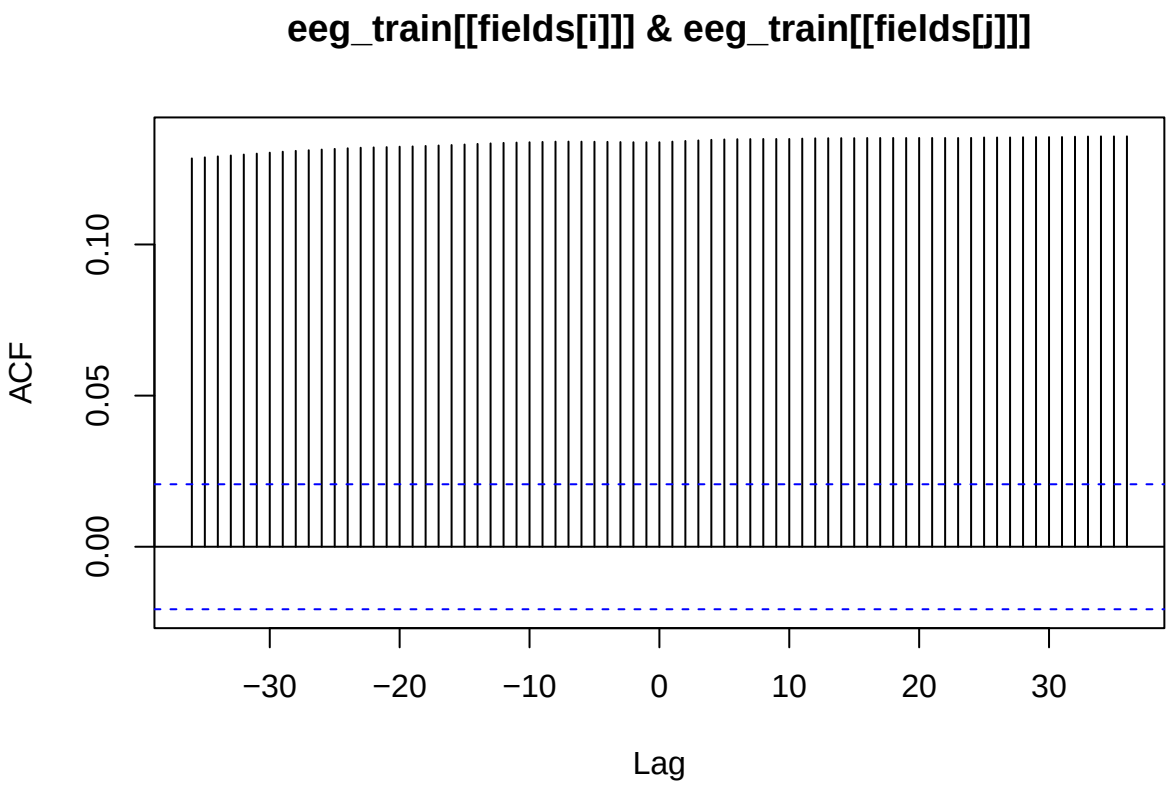

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
−30
−20
−10
0
10
20
30
Lag

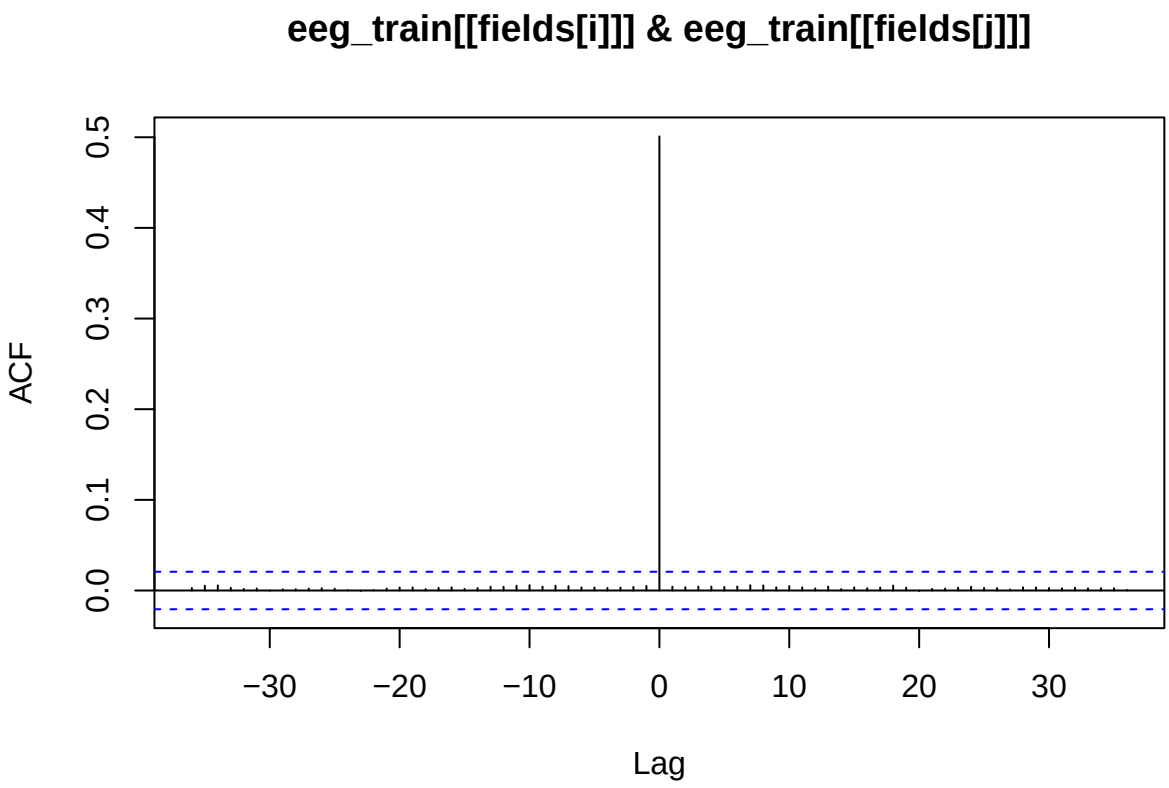

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.1
0.0
−0.2
−0.4
−30
−20
−10
0
10
20
30
Lag

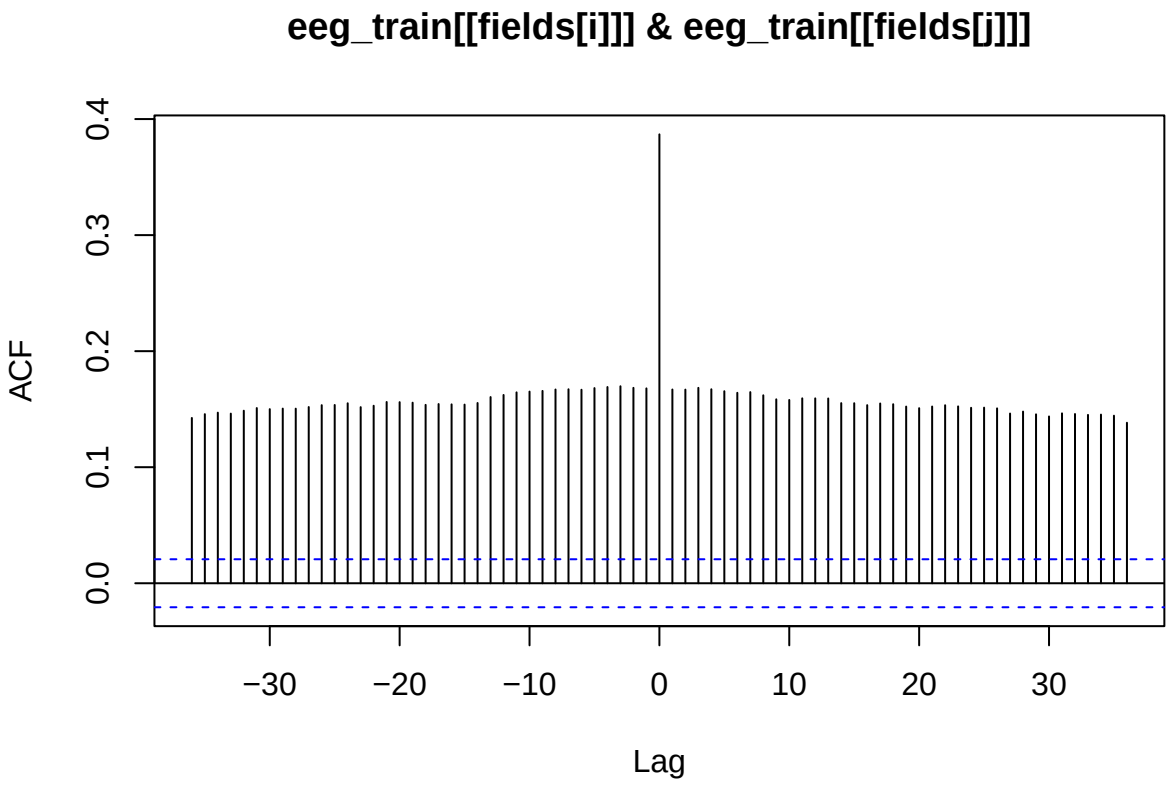

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
−30
−20
−10
0
10
20
30
Lag

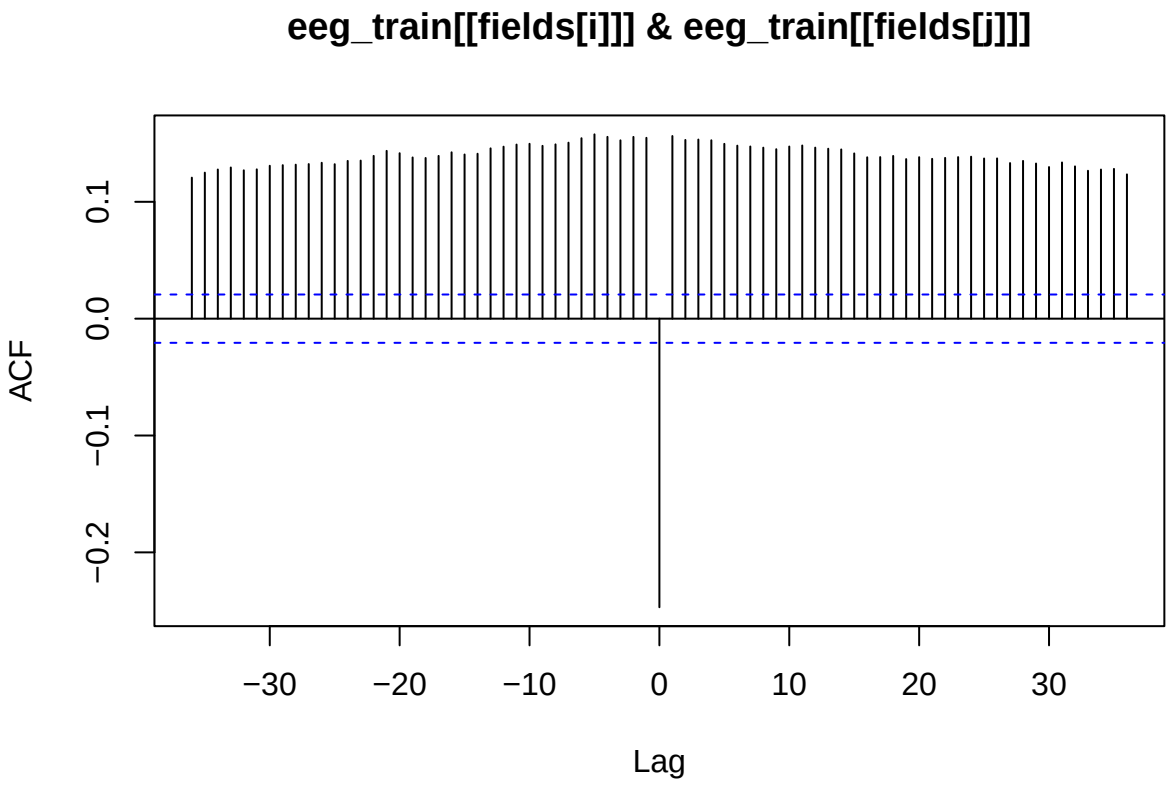

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
−0.05
−0.10
−30
−20
−10
0
10
20
30
Lag

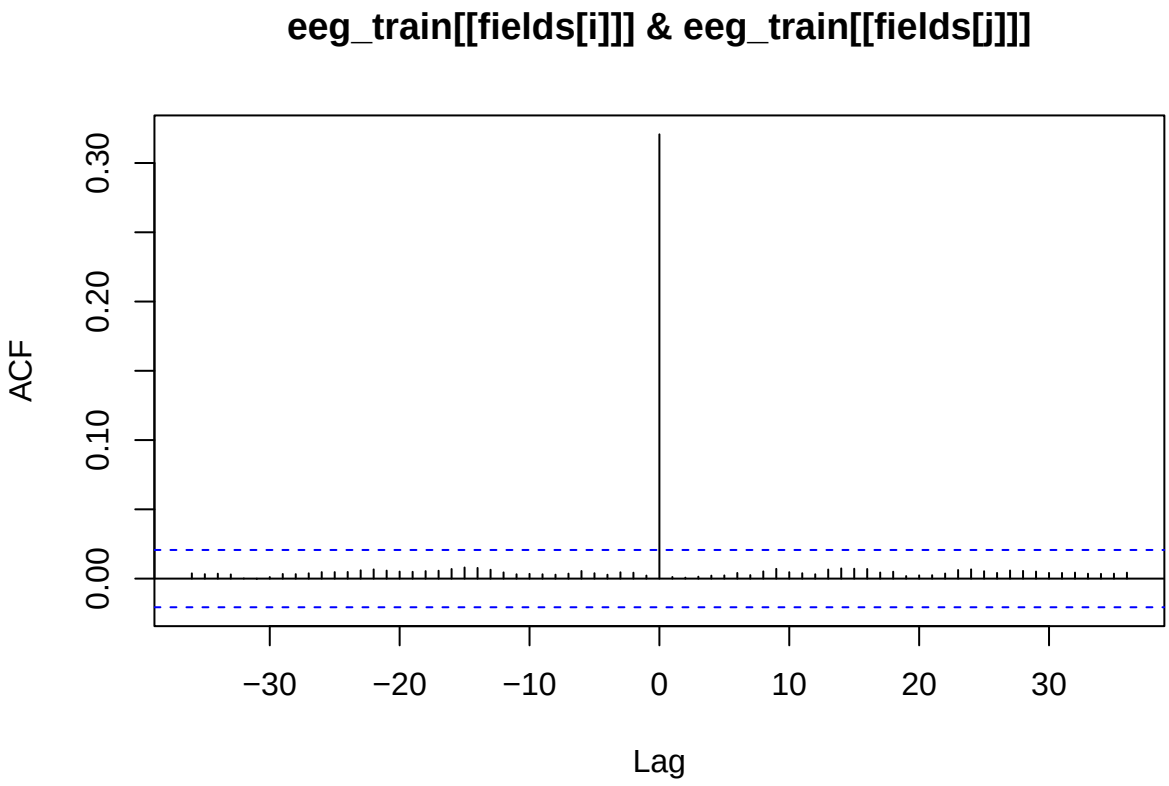

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.01
−0.01
−0.03
−30
−20
−10
0
10
20
30
Lag

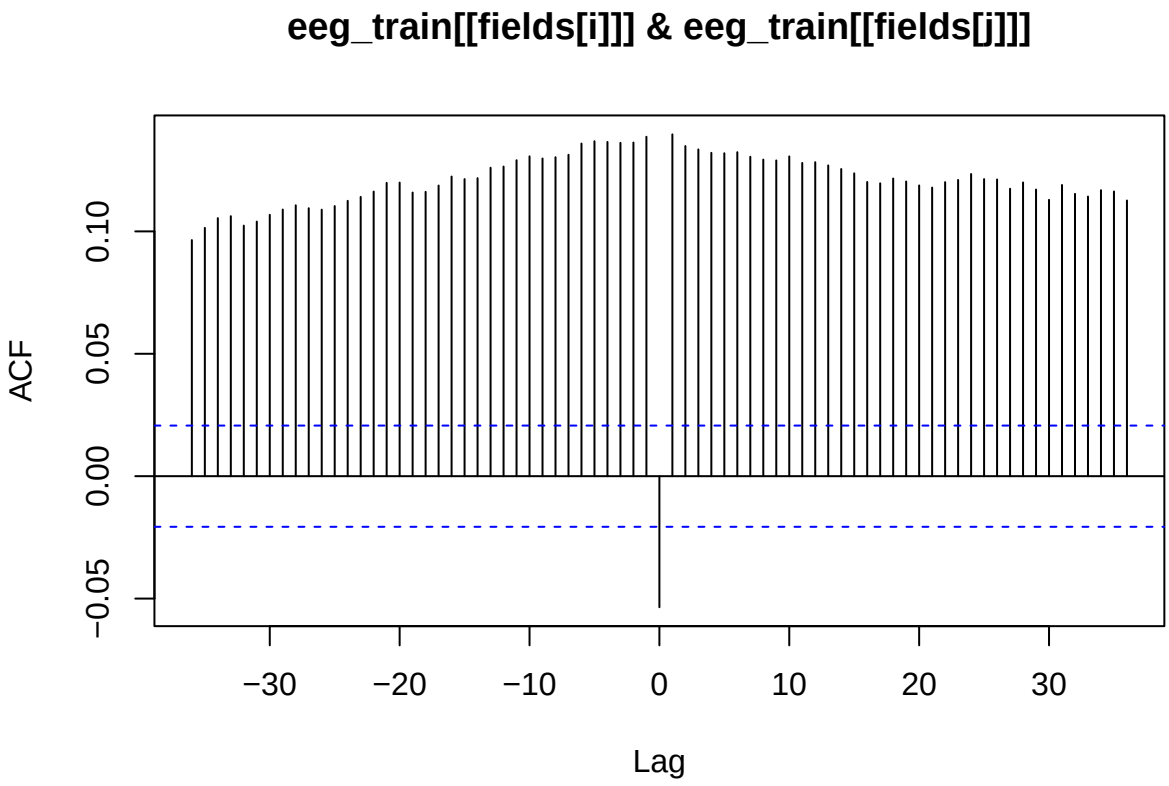

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.05
0.10
−30
−20
−10
0
10
20
30
Lag

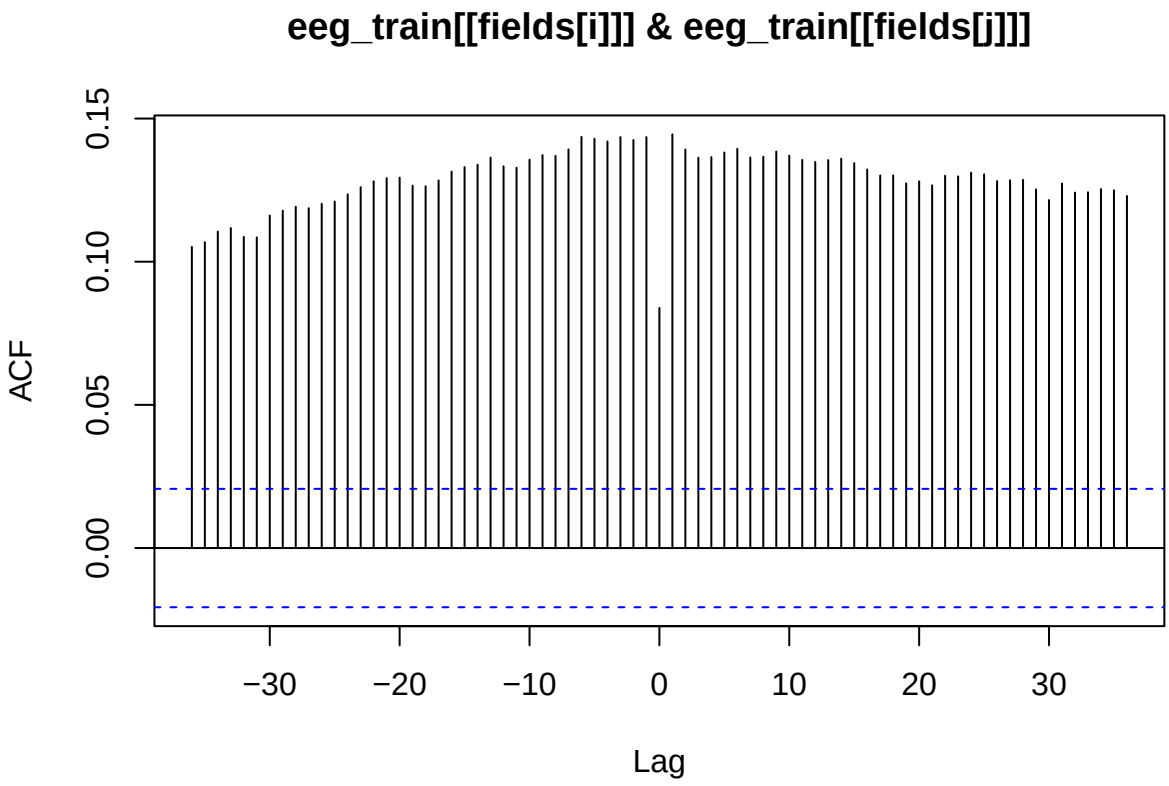

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
0.5
−30
−20
−10
0
10
20
30
Lag

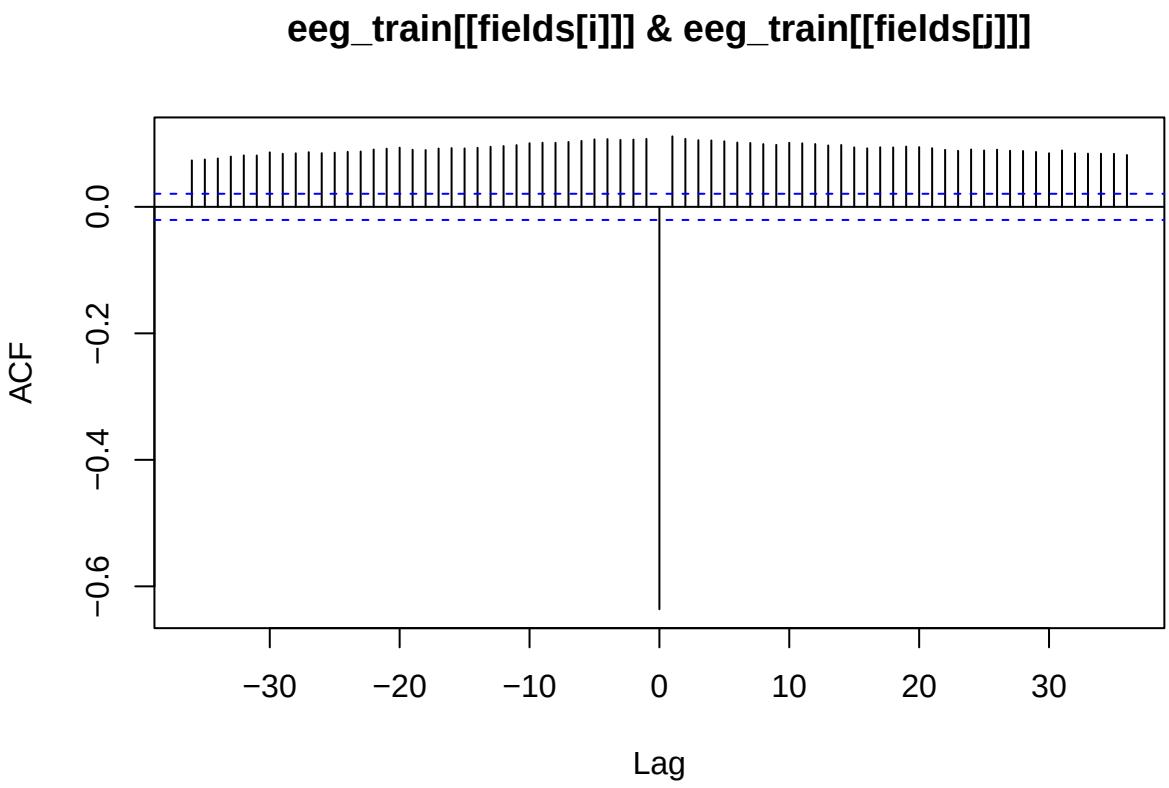

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
−30
−20
−10
0
10
20
30
Lag

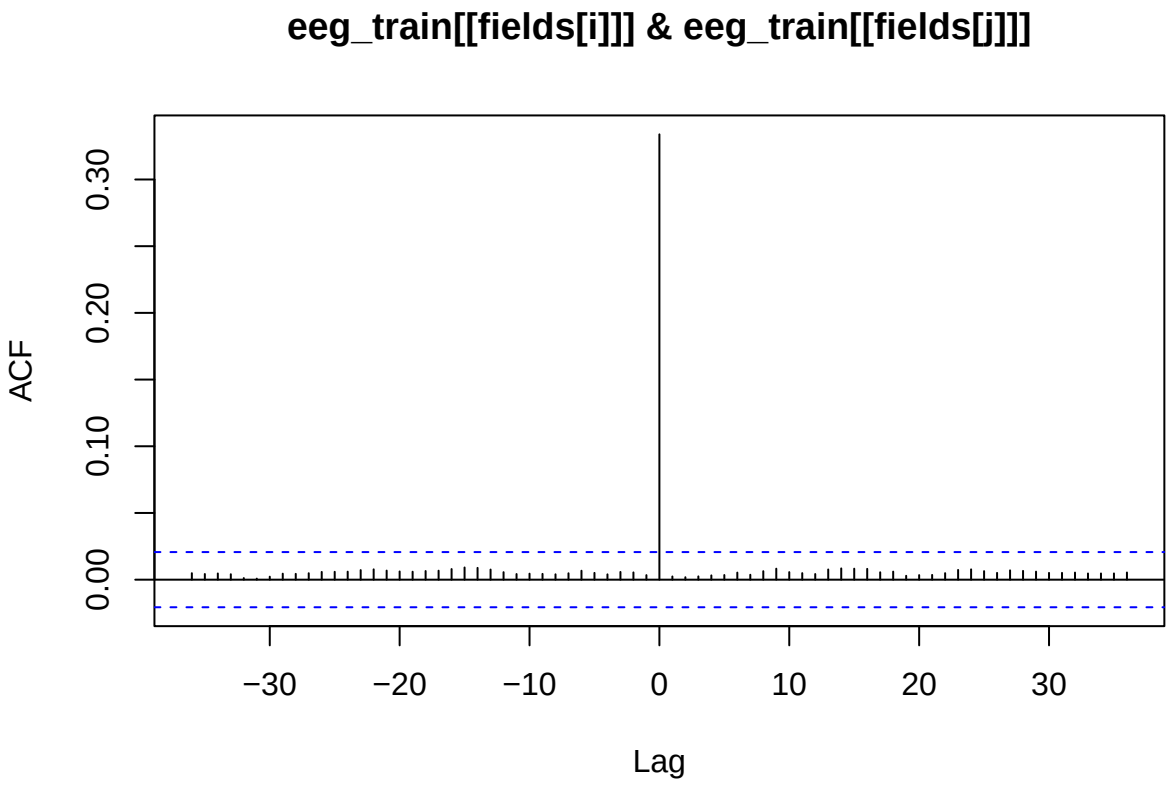

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.1
0.0
−0.1
−0.2
−30
−20
−10
0
10
20
30
Lag

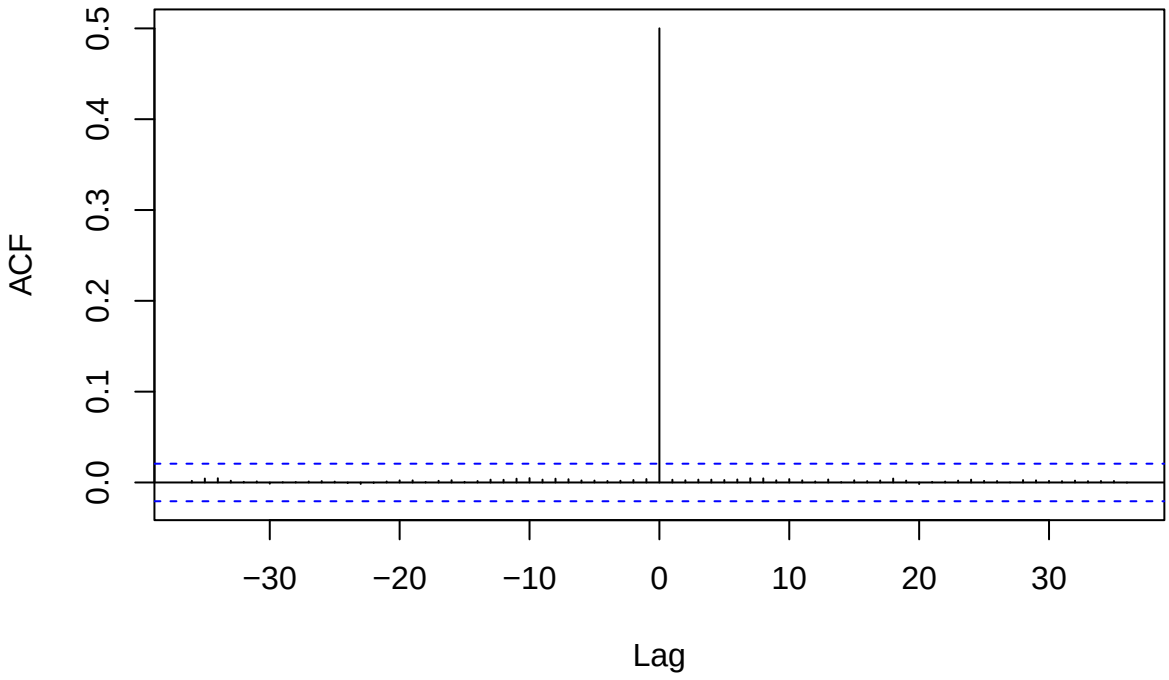

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.10
0.20
0.30
−30
−20
−10
0
10
20
30
Lag

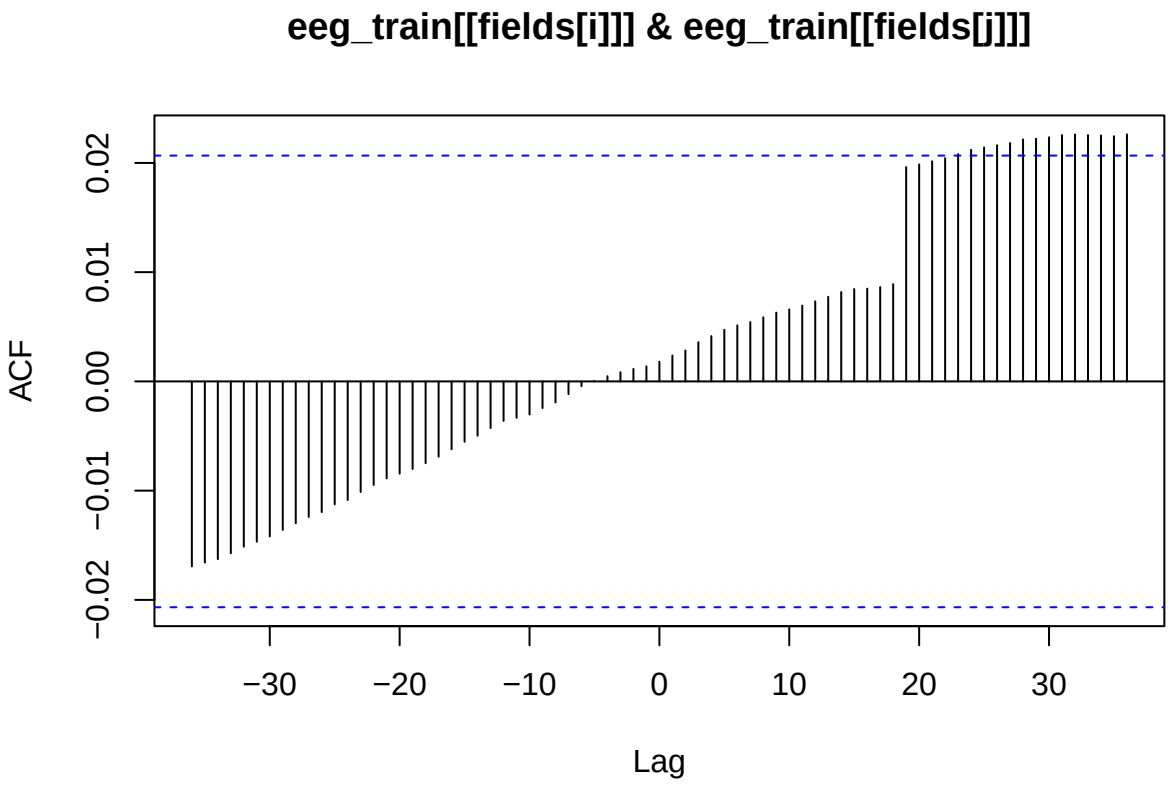

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.10
0.05
0.00
−0.05
−30
−20
−10
0
10
20
30
Lag

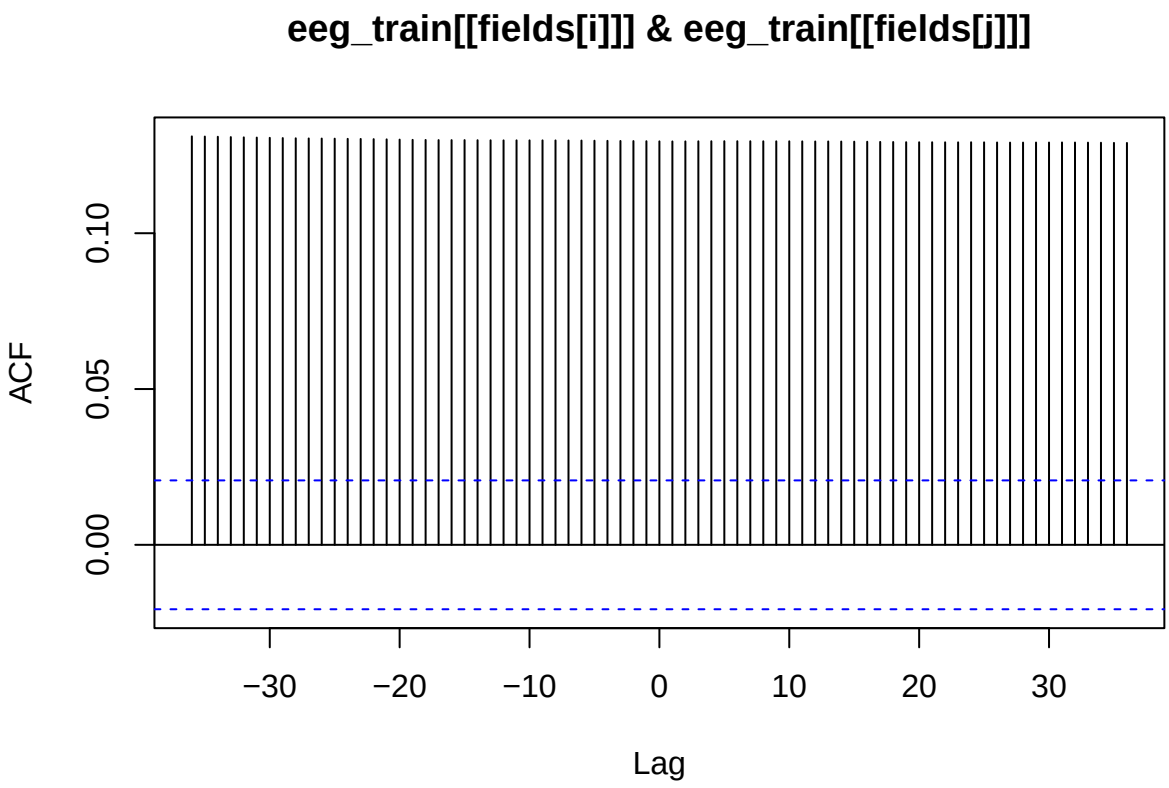

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.05
0.10
0.15
−30
−20
−10
0
10
20
30
Lag

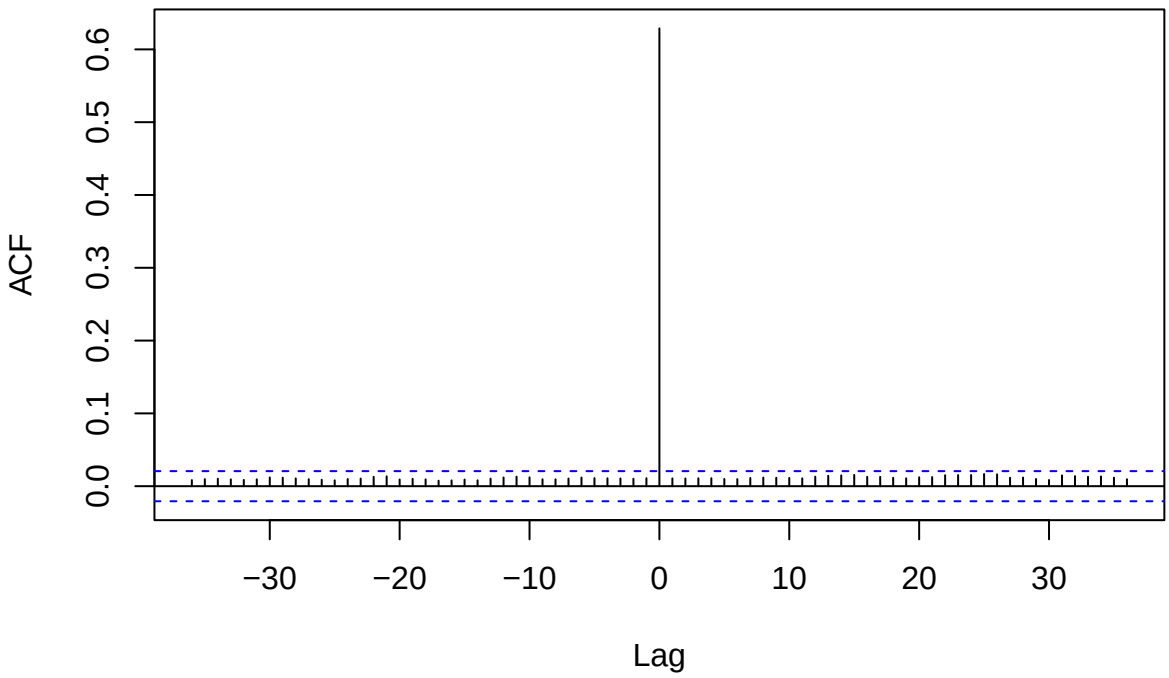

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
−0.2
−0.4
−0.6
−30
−20
−10
0
10
20
30
Lag

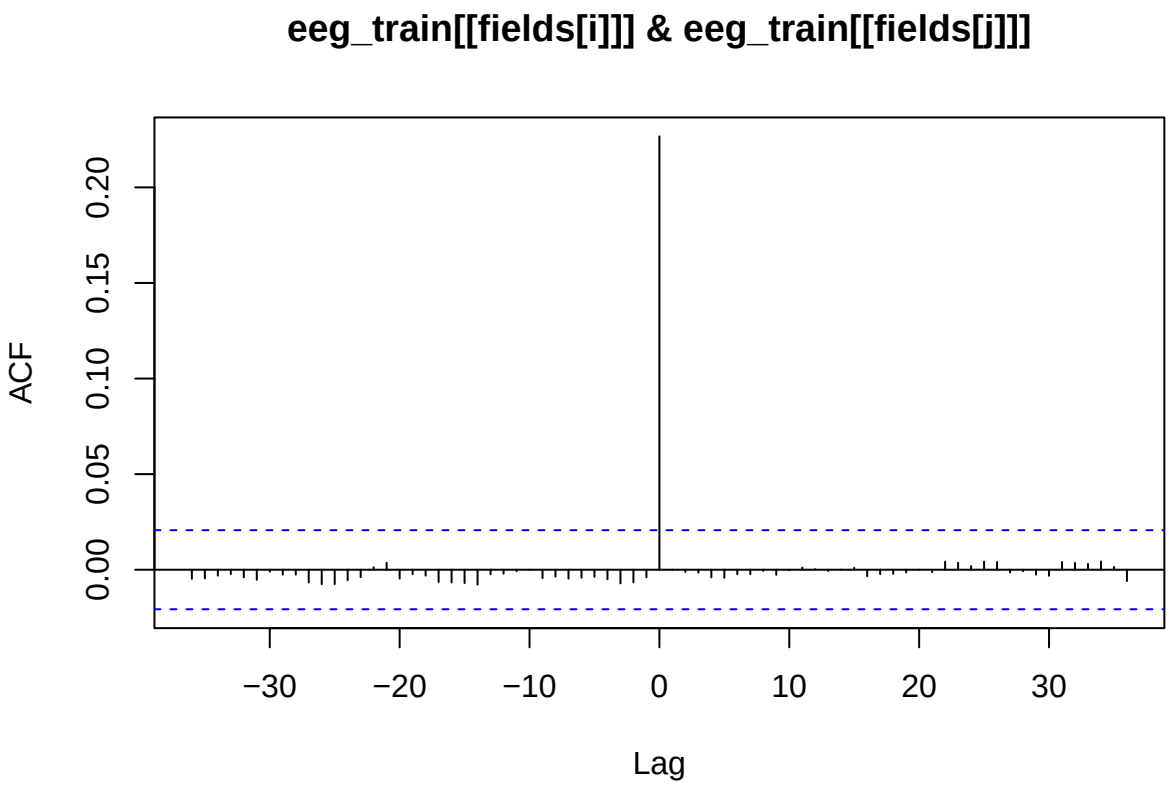

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.10
0.20
0.30
−30
−20
−10
0
10
20
30
Lag

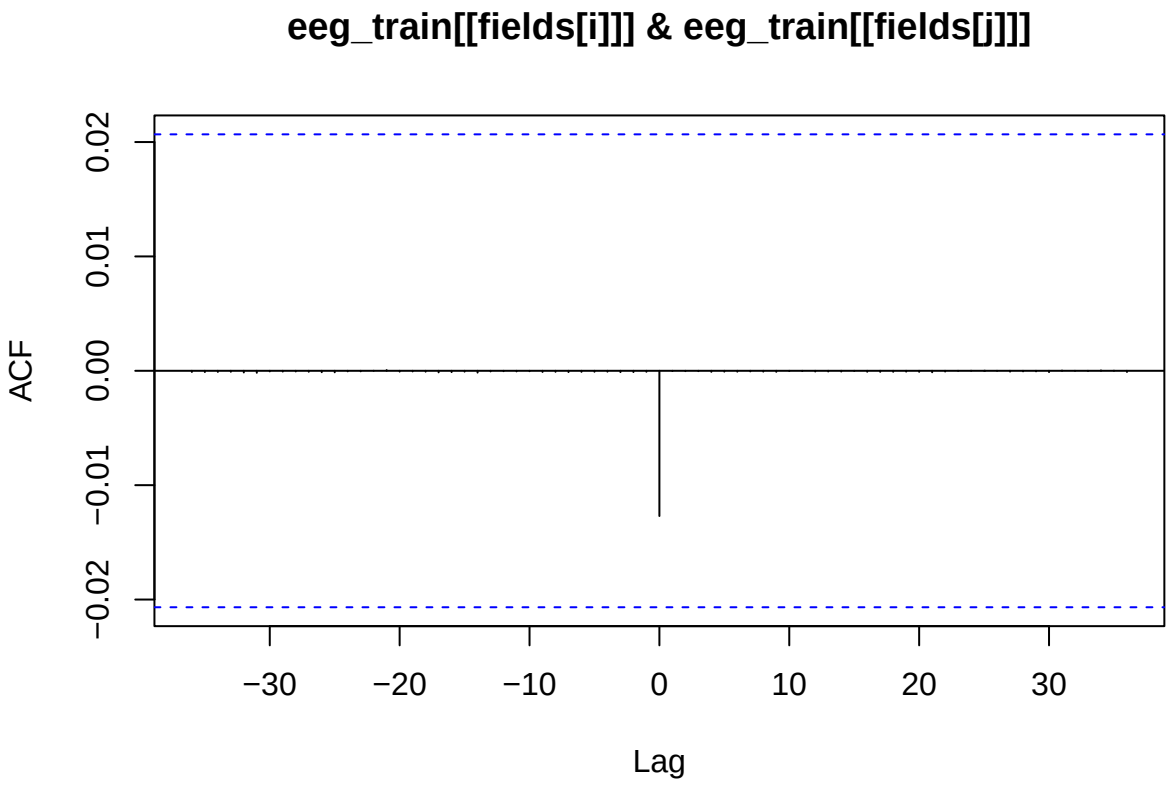

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
0.5
−30
−20
−10
0
10
20
30
Lag

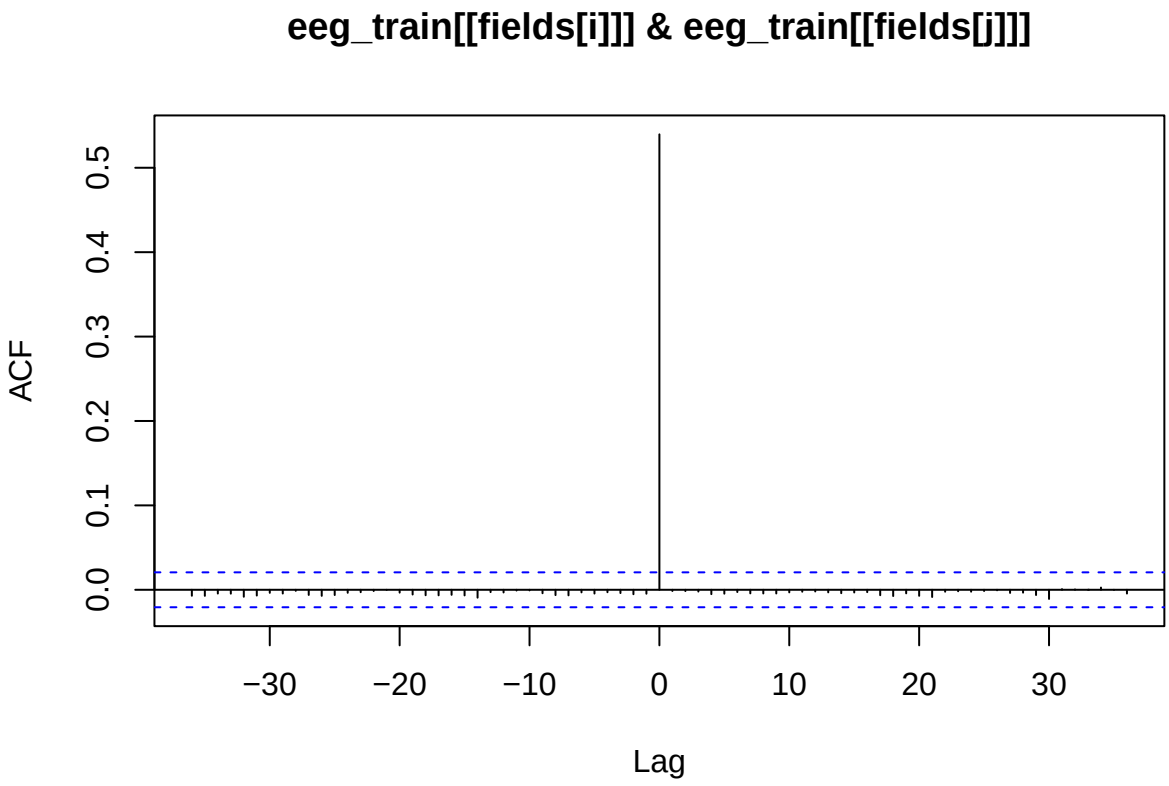

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag

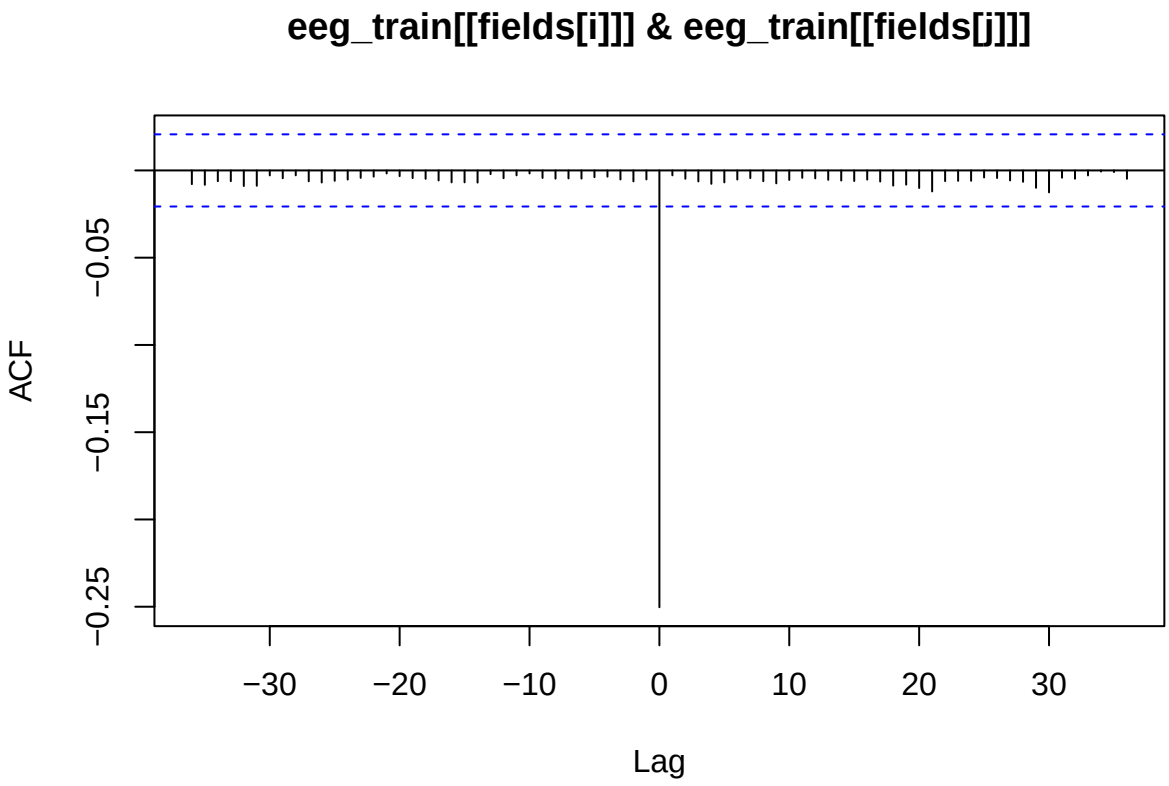

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.05
0.10
−30
−20
−10
0
10
20
30
Lag

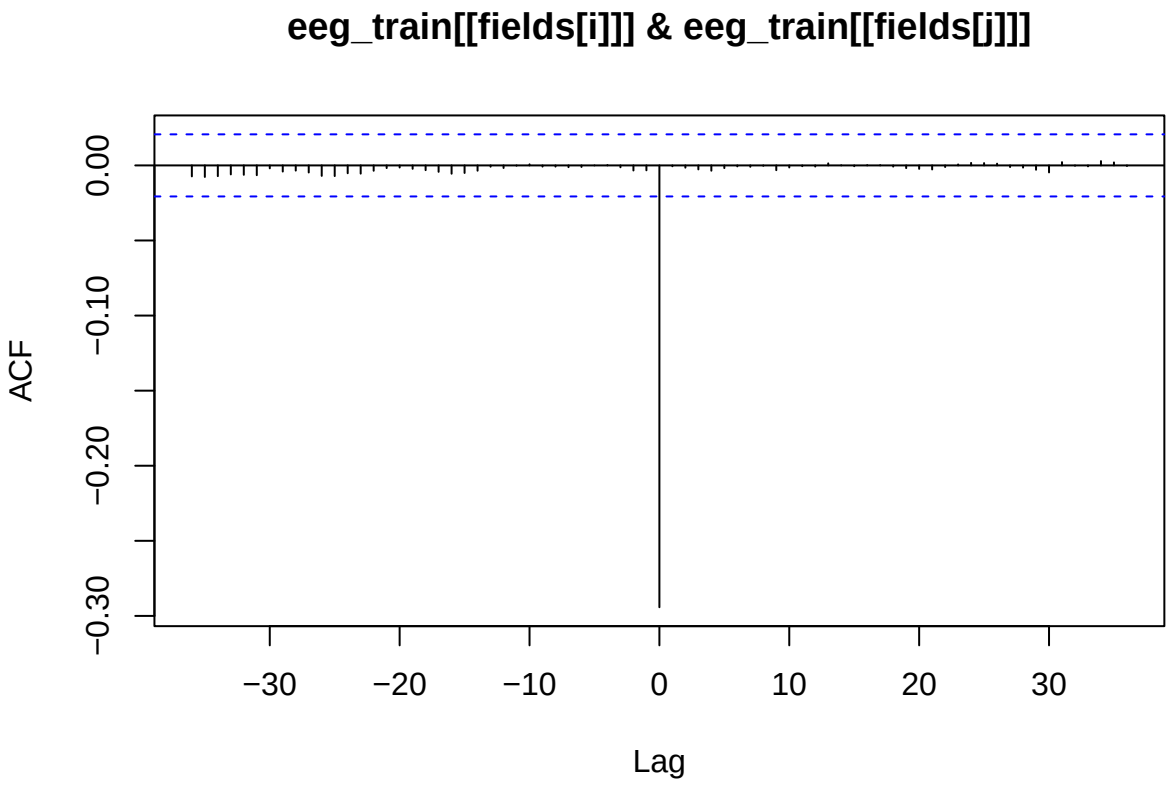

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−30
−20
−10
0
10
20
30
Lag

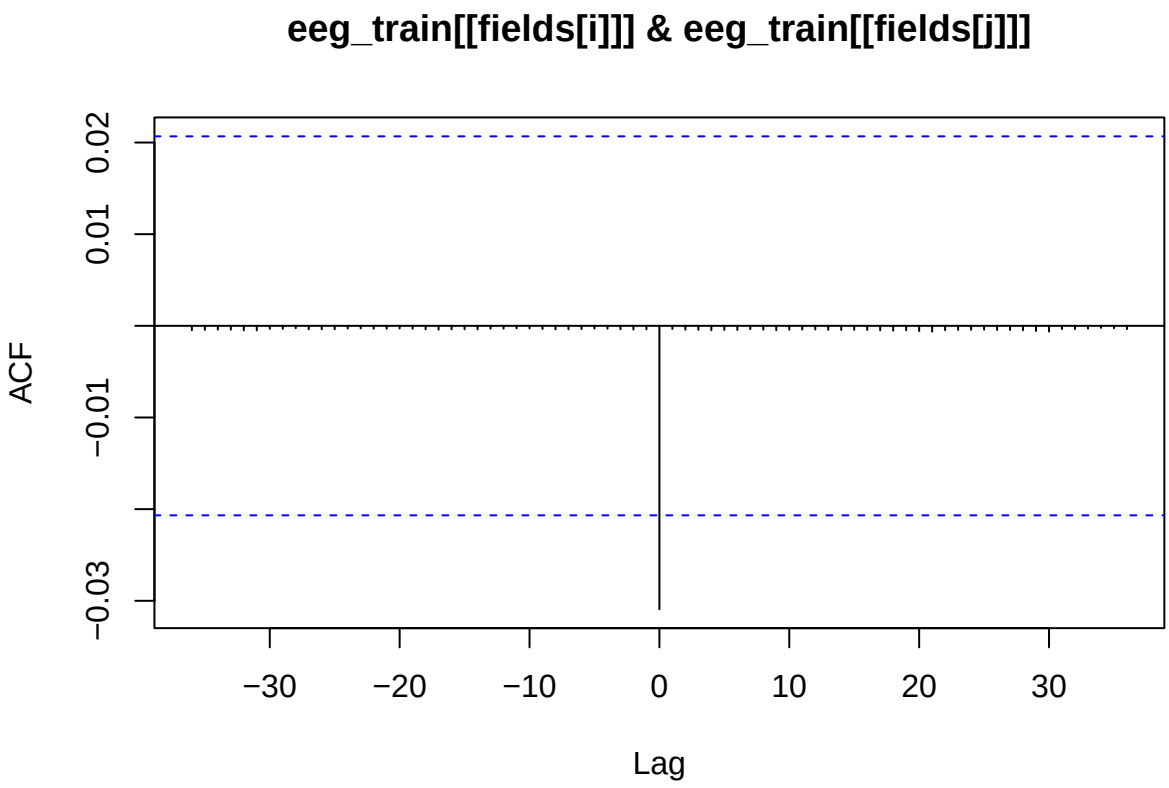

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.05
0.10
0.15
0.20
−30
−20
−10
0
10
20
30
Lag

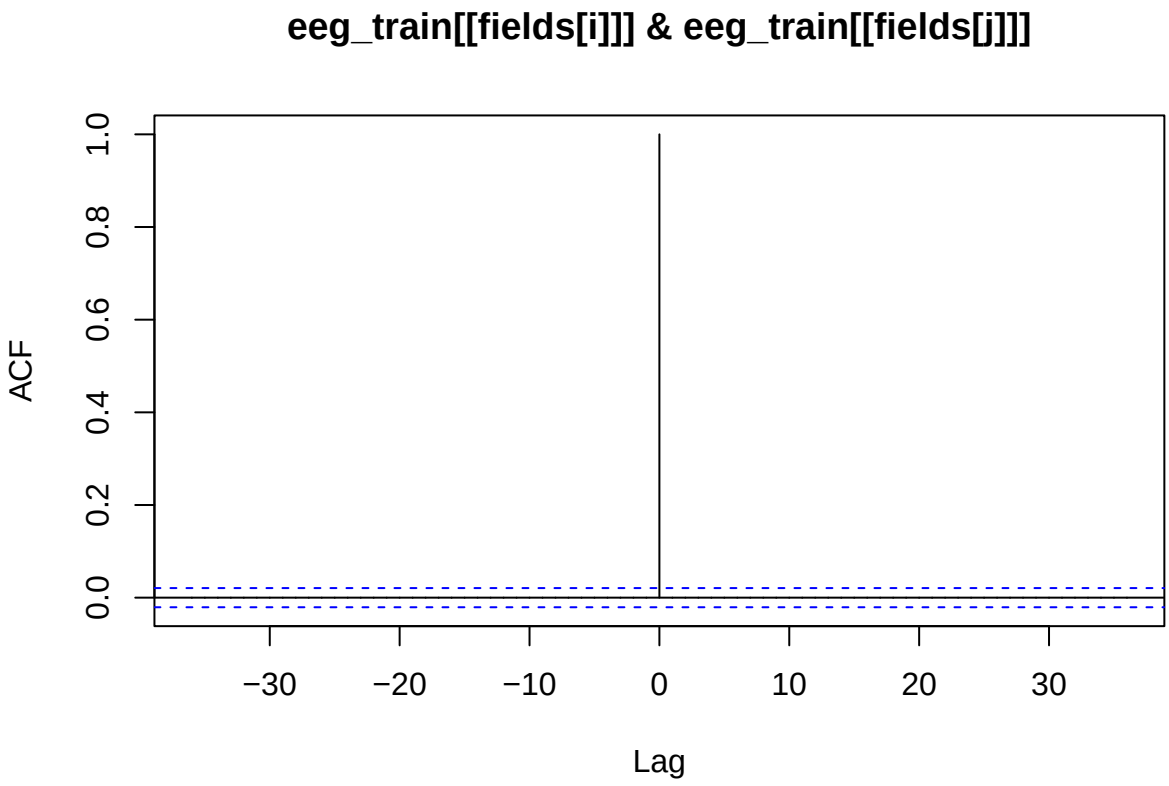

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag

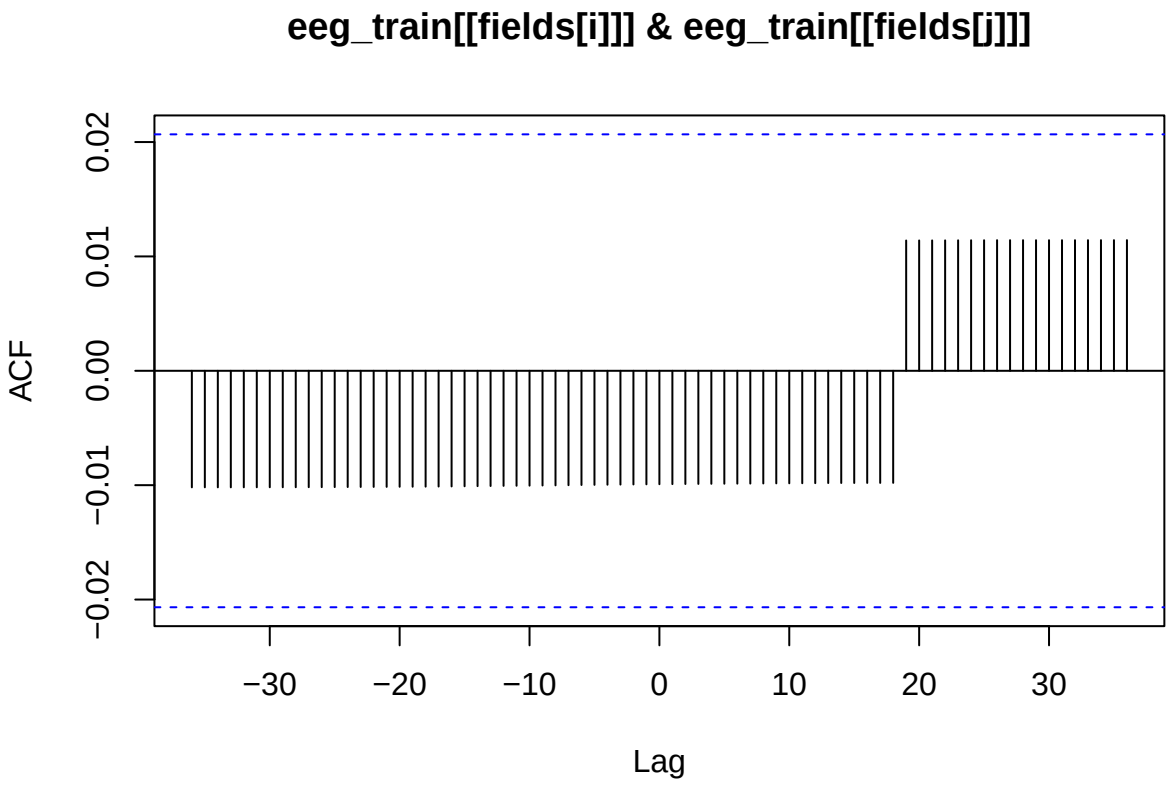

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
0.5
−30
−20
−10
0
10
20
30
Lag

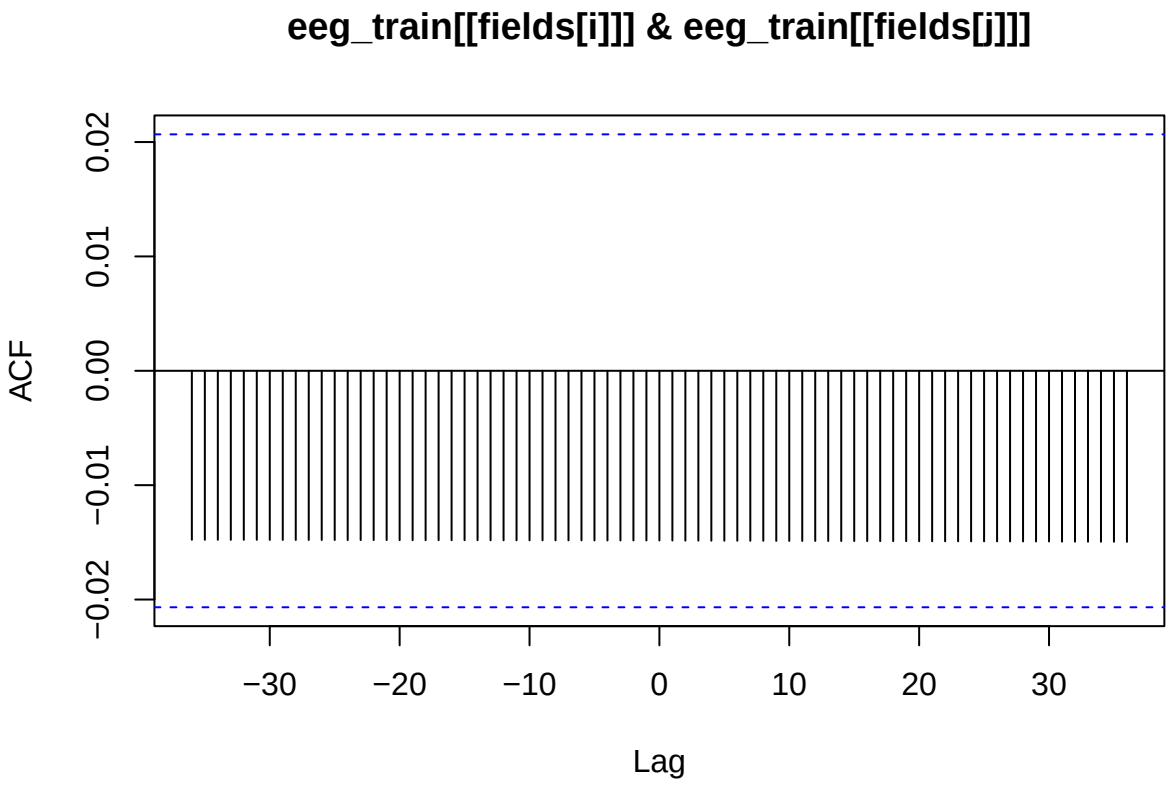

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
−0.25
−0.15
−0.05
−30
−20
−10
0
10
20
30
Lag

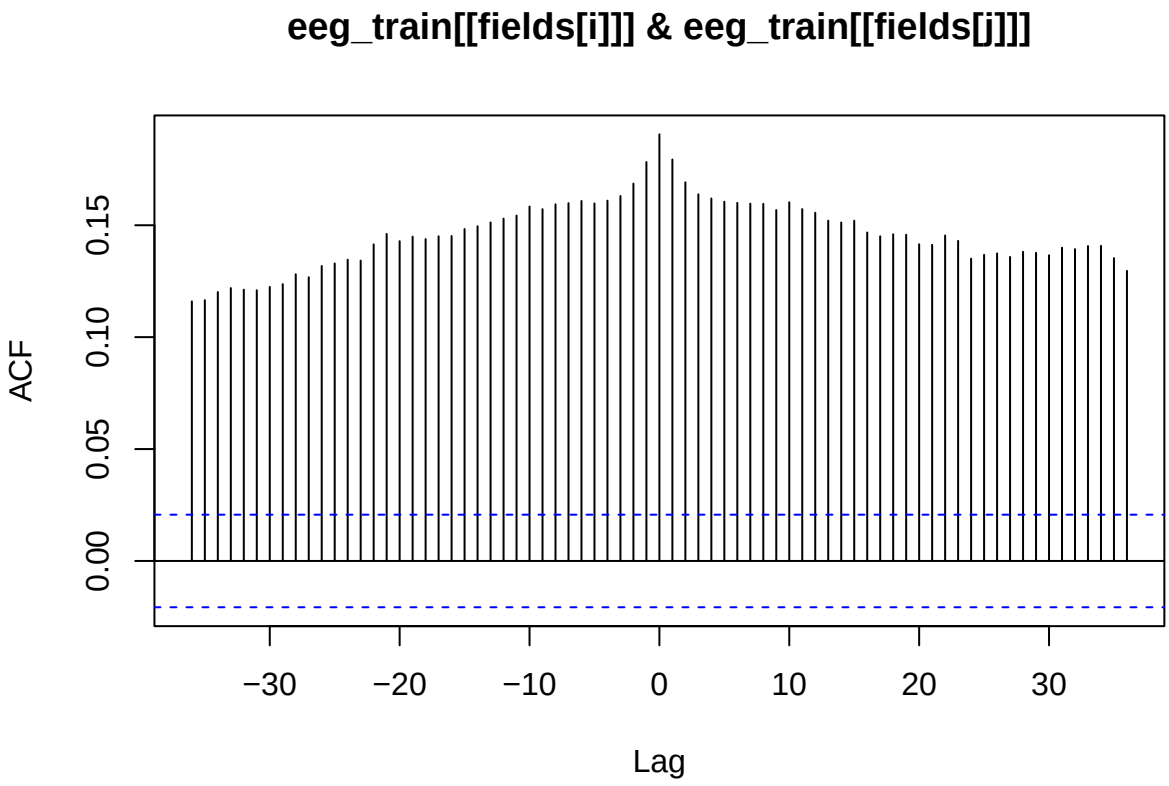

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
−0.10
−0.20
−0.30
−30
−20
−10
0
10
20
30
Lag

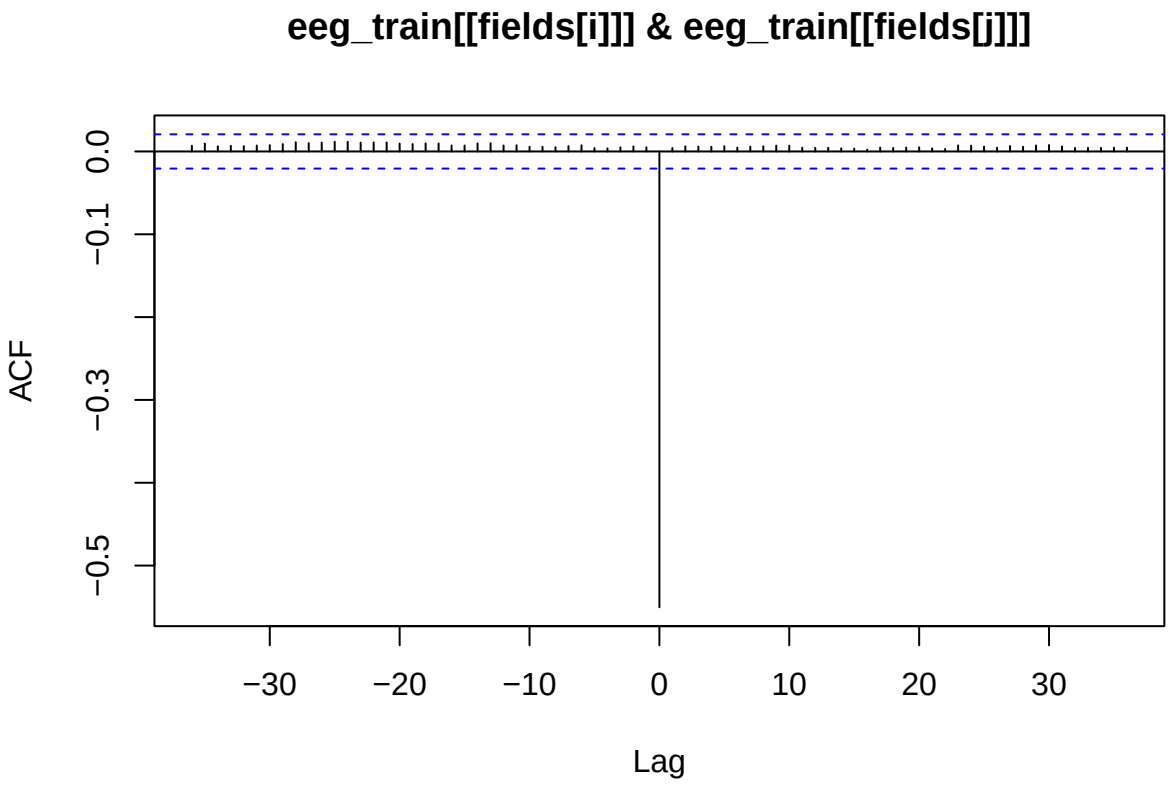

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
−0.01
−0.03
−30
−20
−10
0
10
20
30
Lag

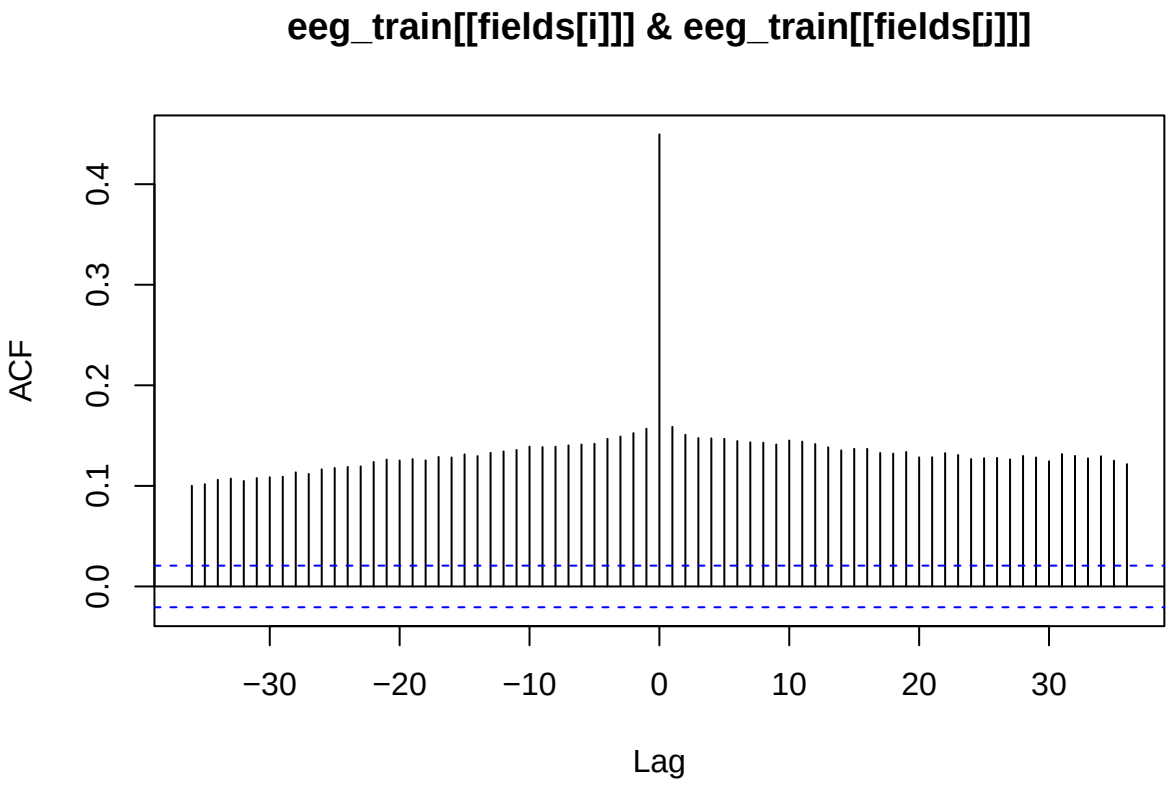

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.2
0.4
0.6
0.8
1.0
−30
−20
−10
0
10
20
30
Lag

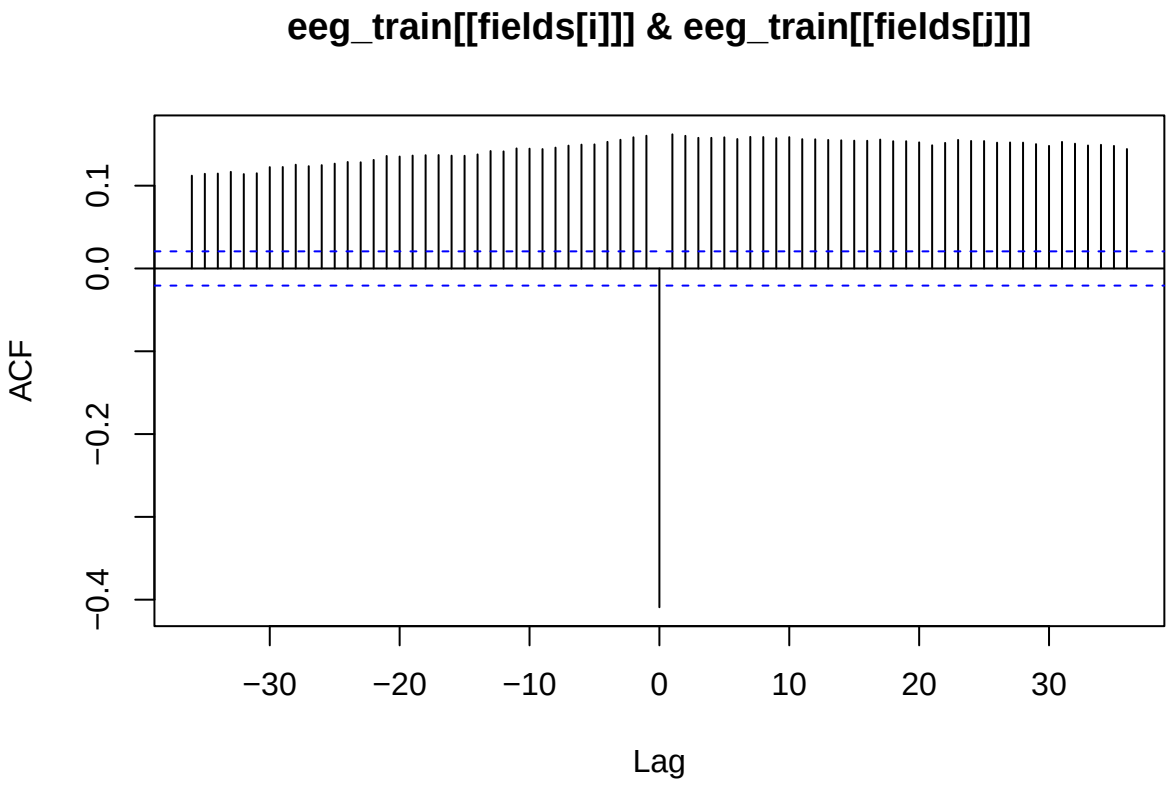

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag

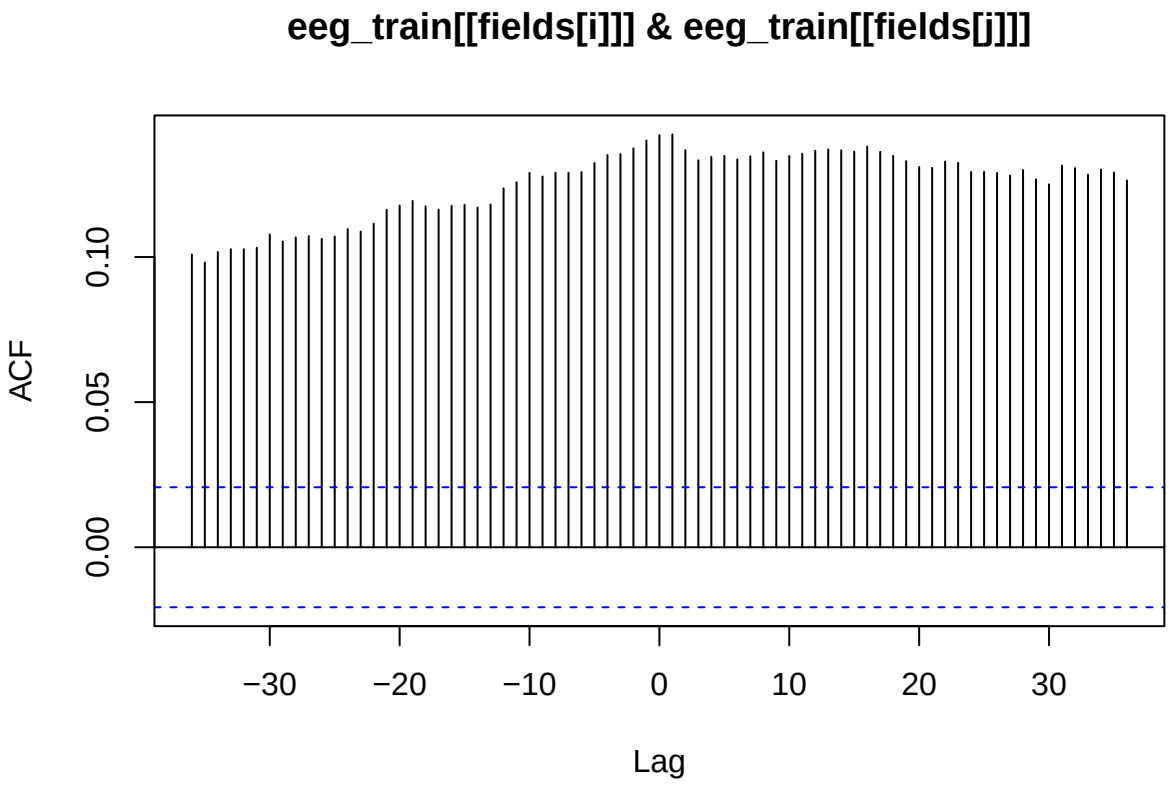

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag

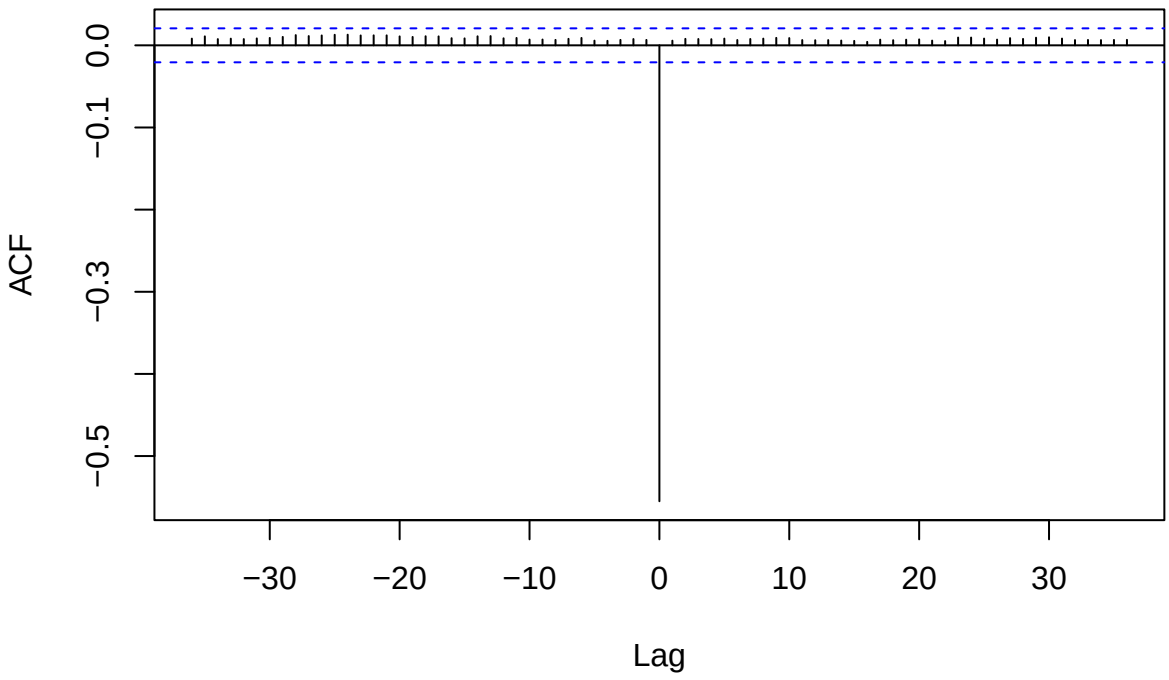

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.05
0.10
0.15
−30
−20
−10
0
10
20
30
Lag

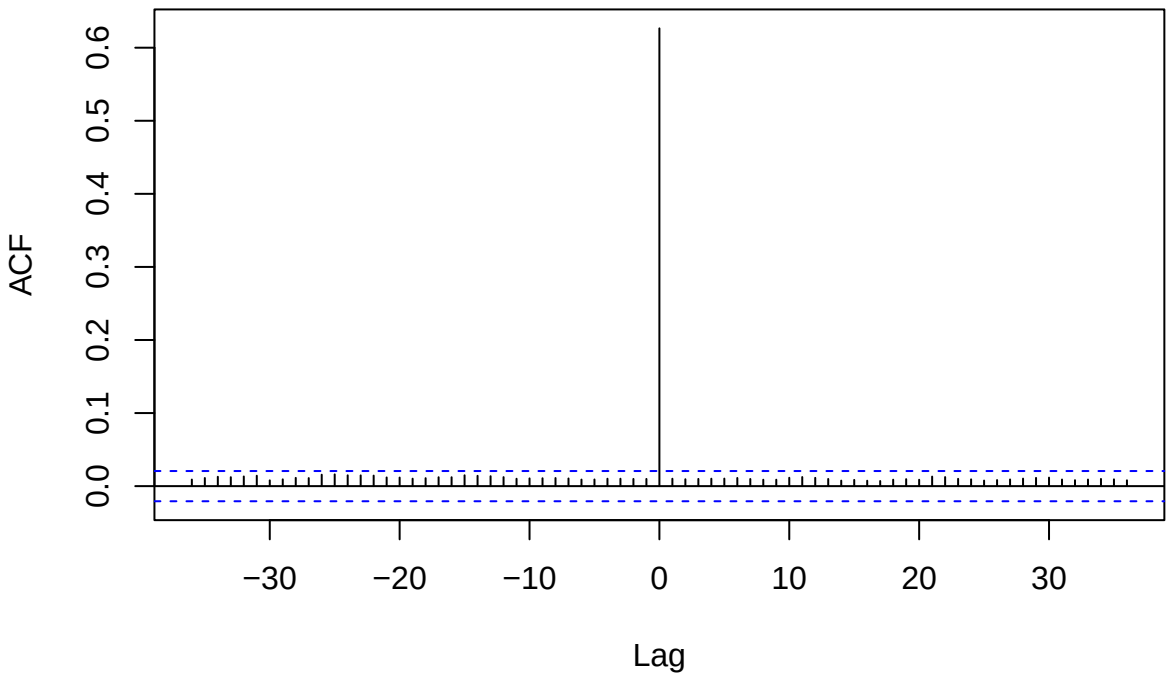

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
−0.1
−0.3
−0.5
−30
−20
−10
0
10
20
30
Lag

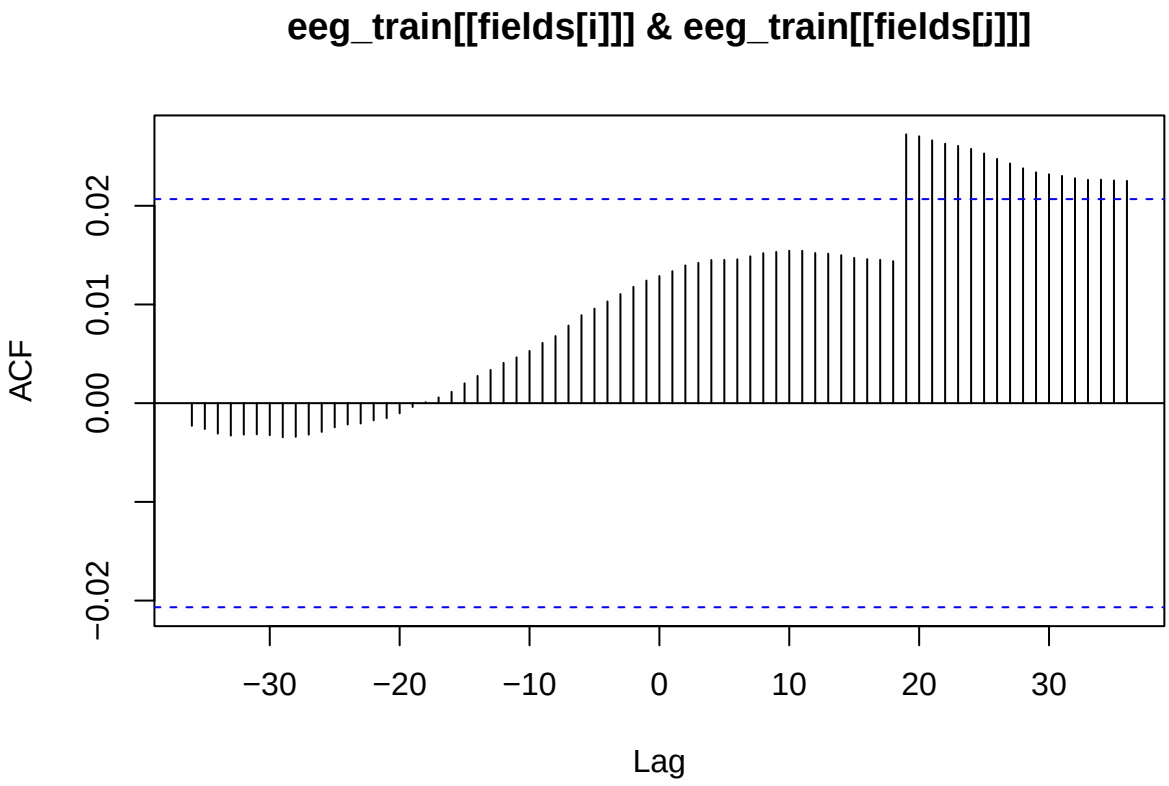

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
−30
−20
−10
0
10
20
30
Lag

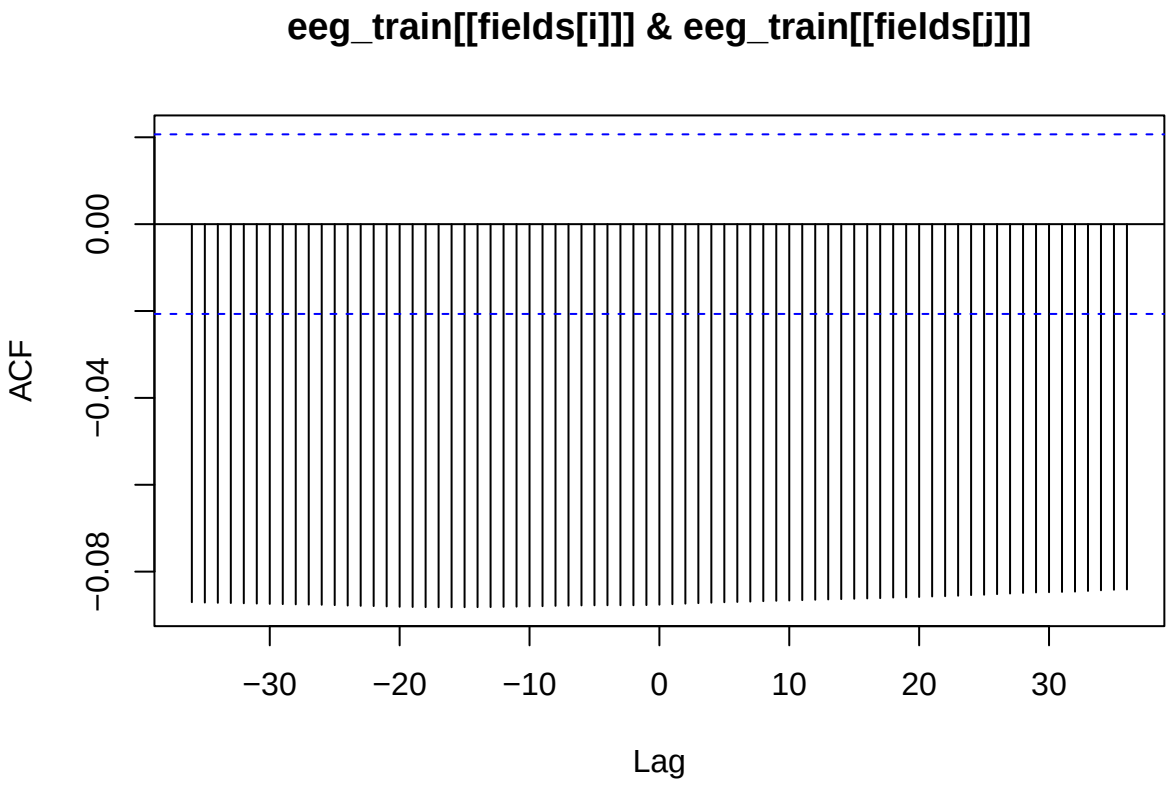

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.1
0.0
−0.2
−0.4
−30
−20
−10
0
10
20
30
Lag

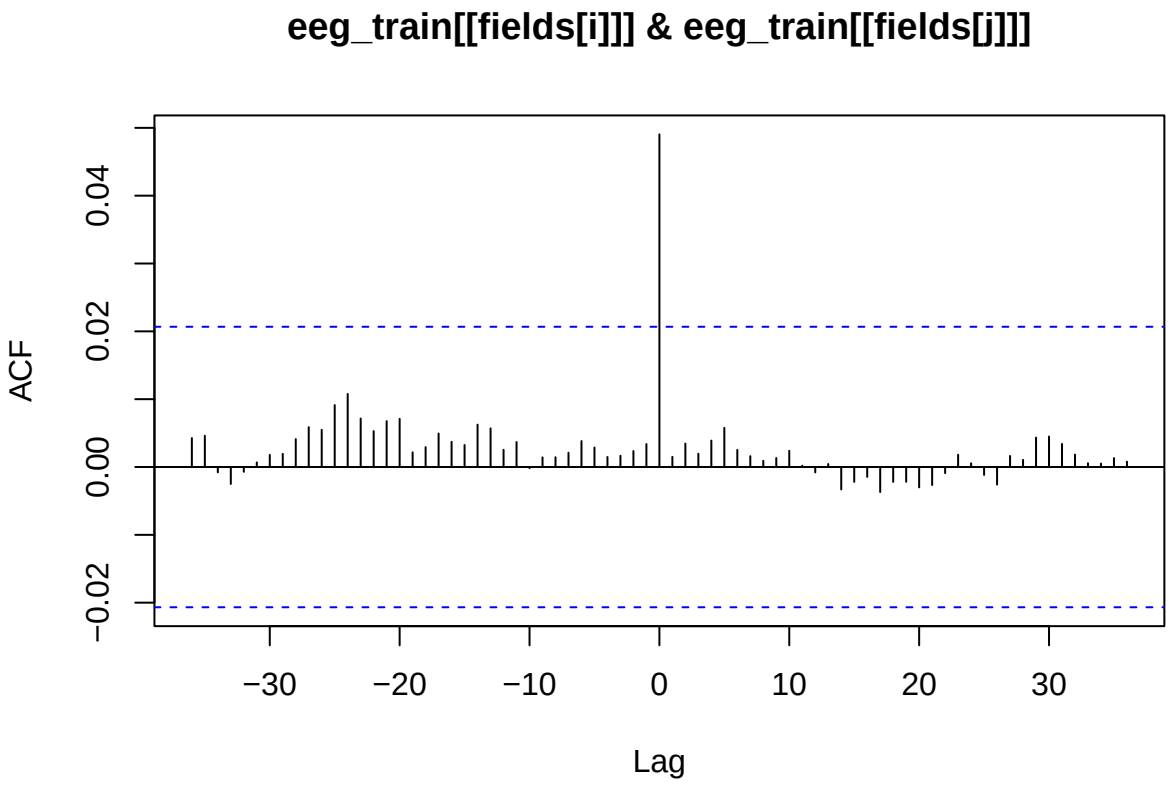

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.05
0.10
−30
−20
−10
0
10
20
30
Lag

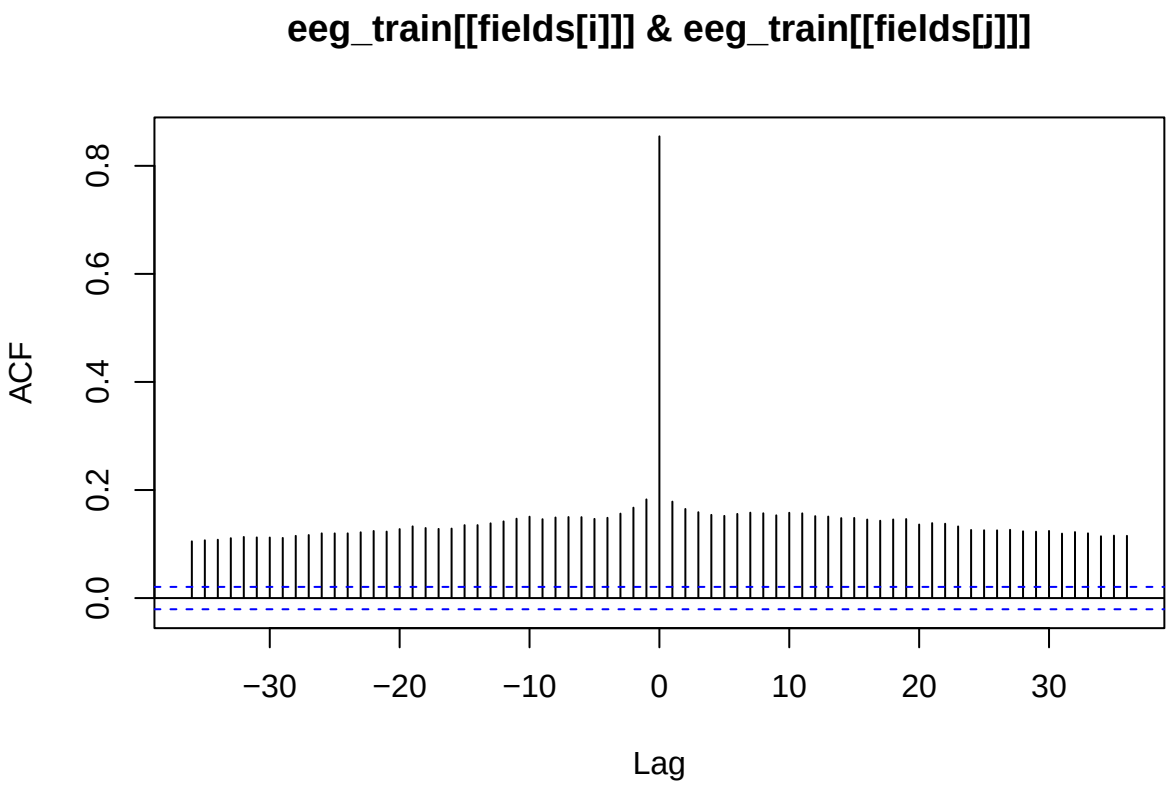

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
−0.1
−0.3
−0.5
−30
−20
−10
0
10
20
30
Lag

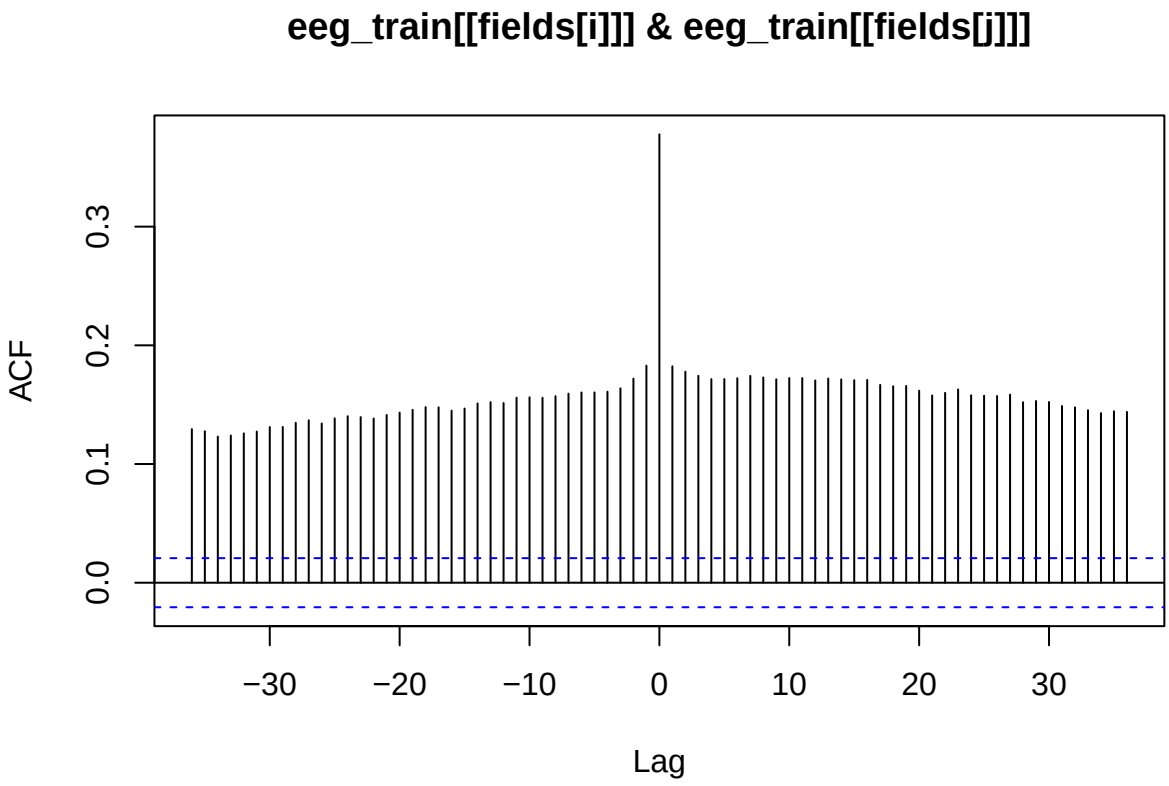

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−30
−20
−10
0
10
20
30
Lag

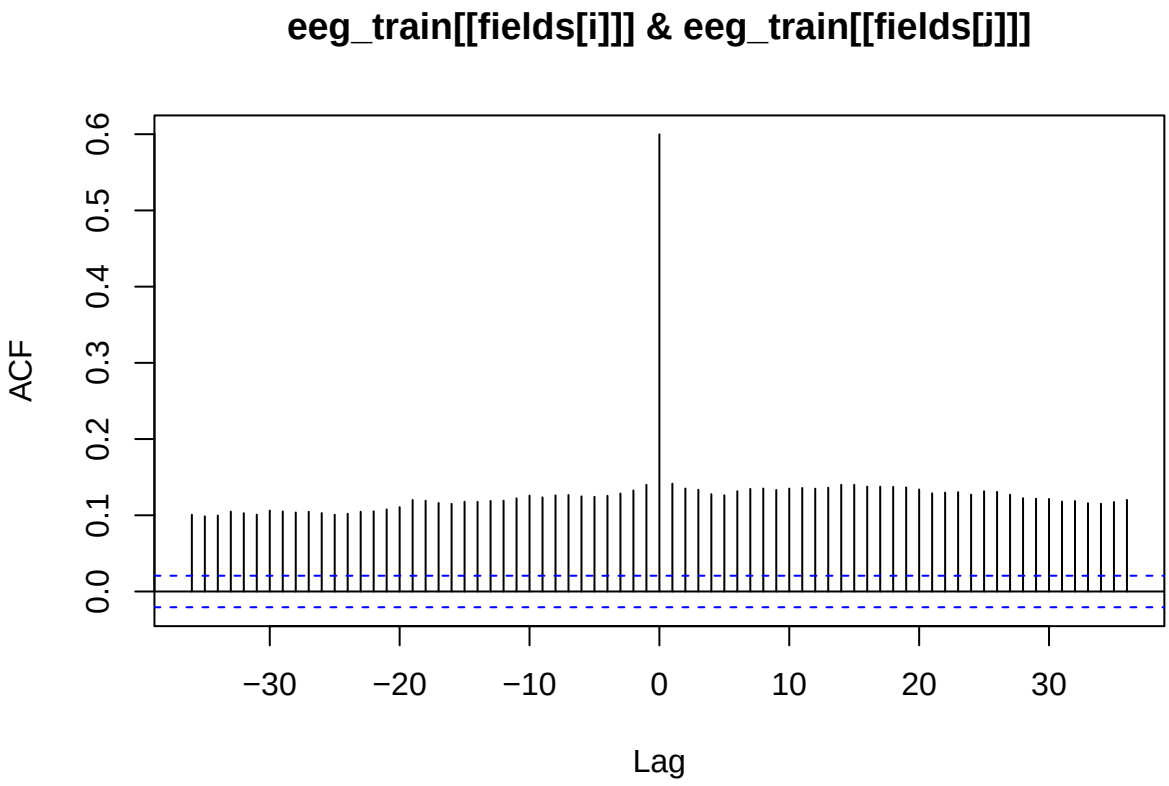

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.02
−30
−20
−10
0
10
20
30
Lag

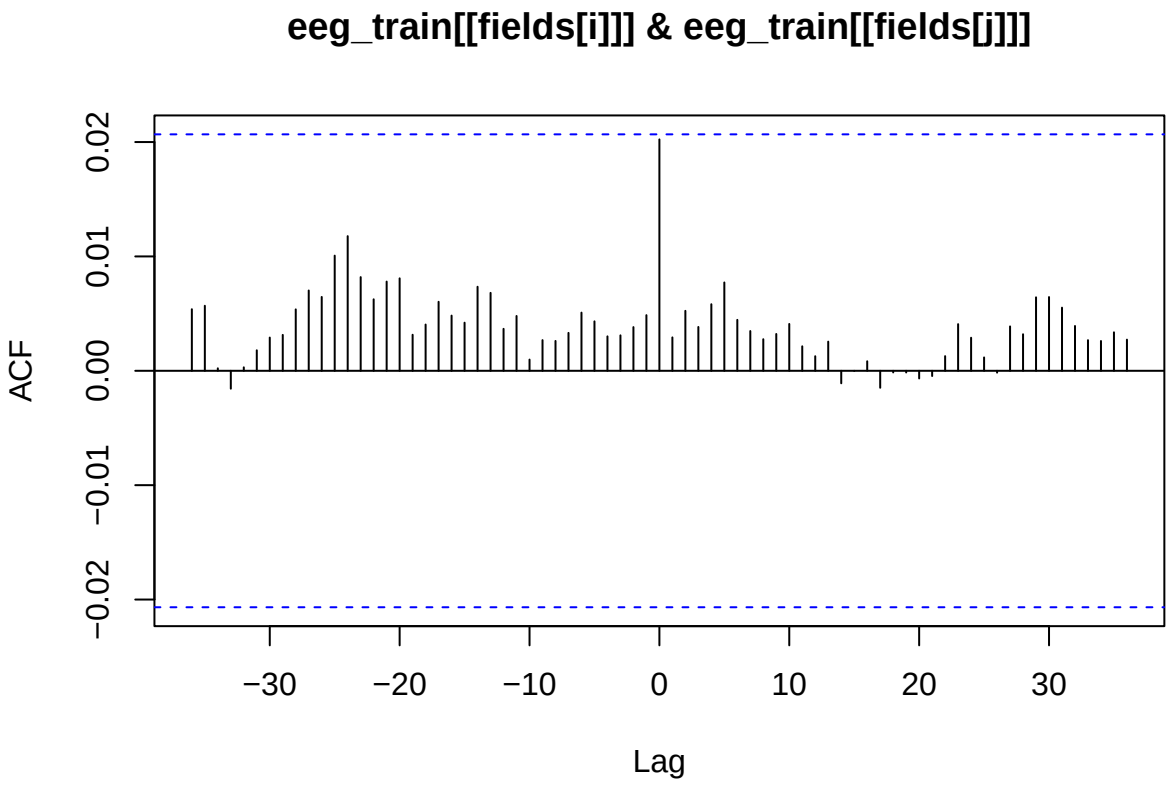

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
−0.04
−0.08
−30
−20
−10
0
10
20
30
Lag

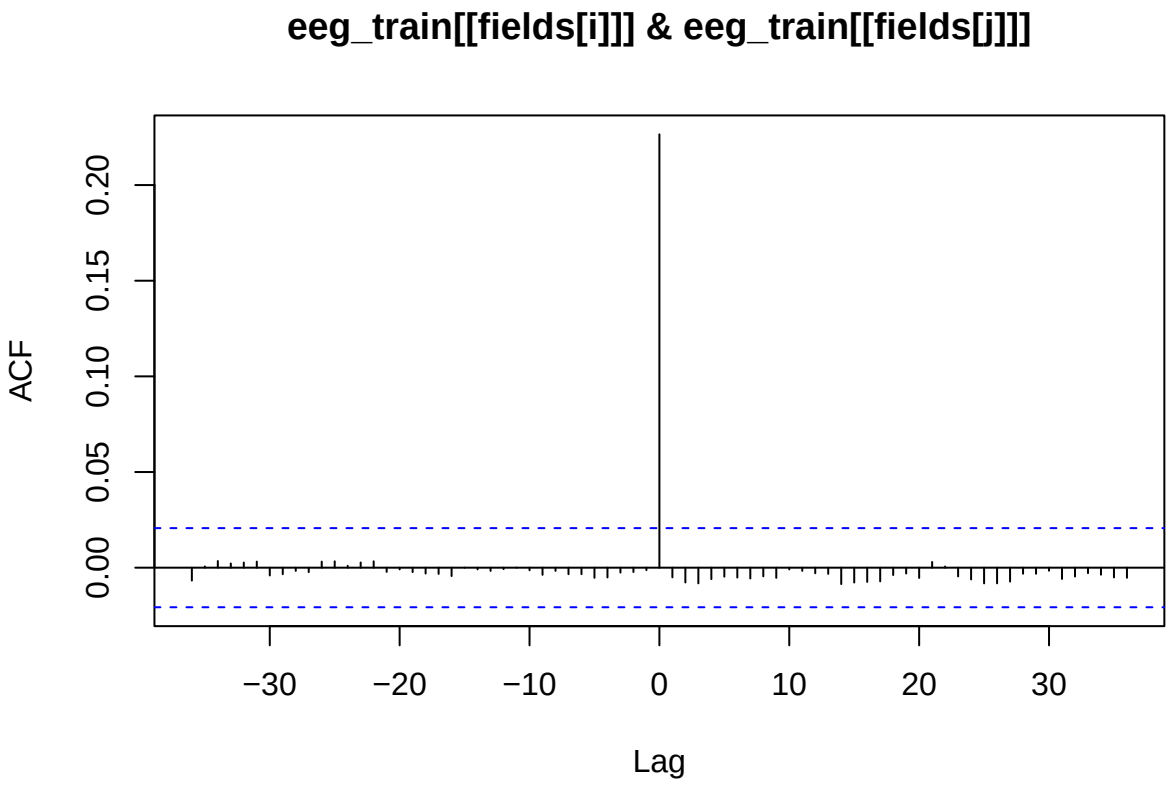

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.04
0.02
0.00
−0.02
−30
−20
−10
0
10
20
30
Lag


eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.2
0.4
0.6
0.8
−30
−20
−10
0
10
20
30
Lag

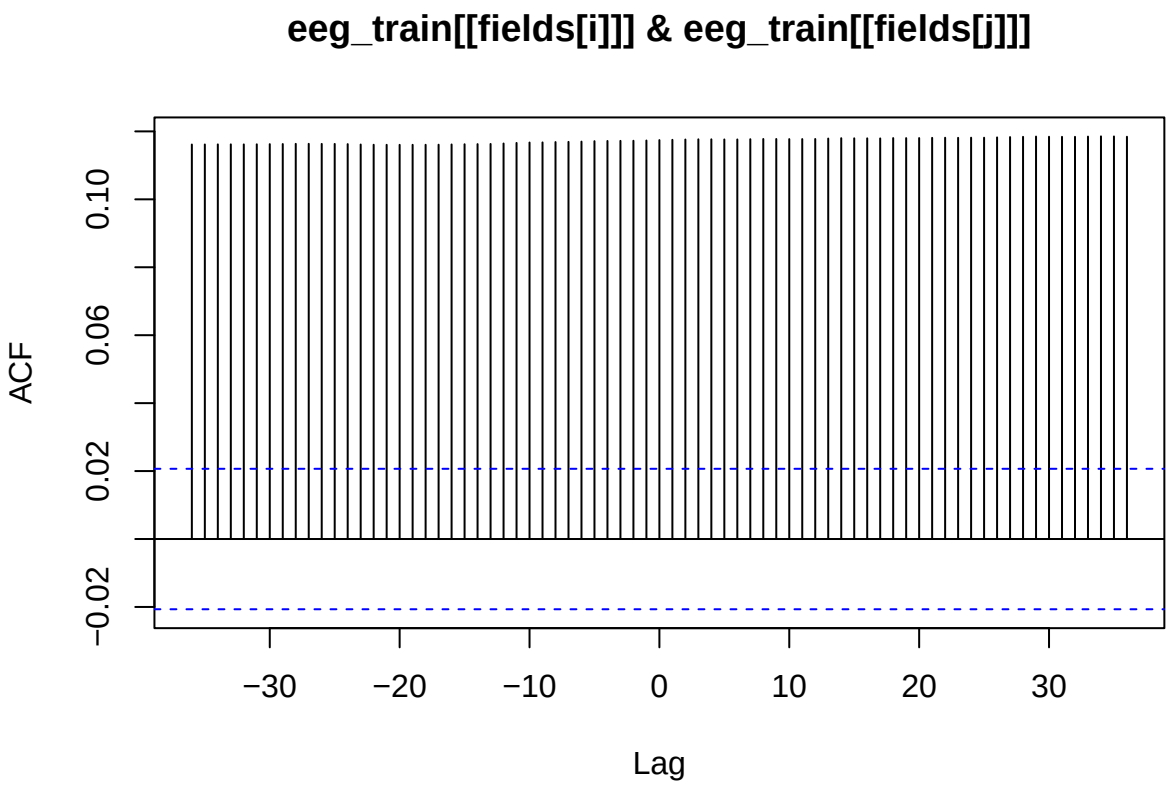

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
−30
−20
−10
0
10
20
30
Lag

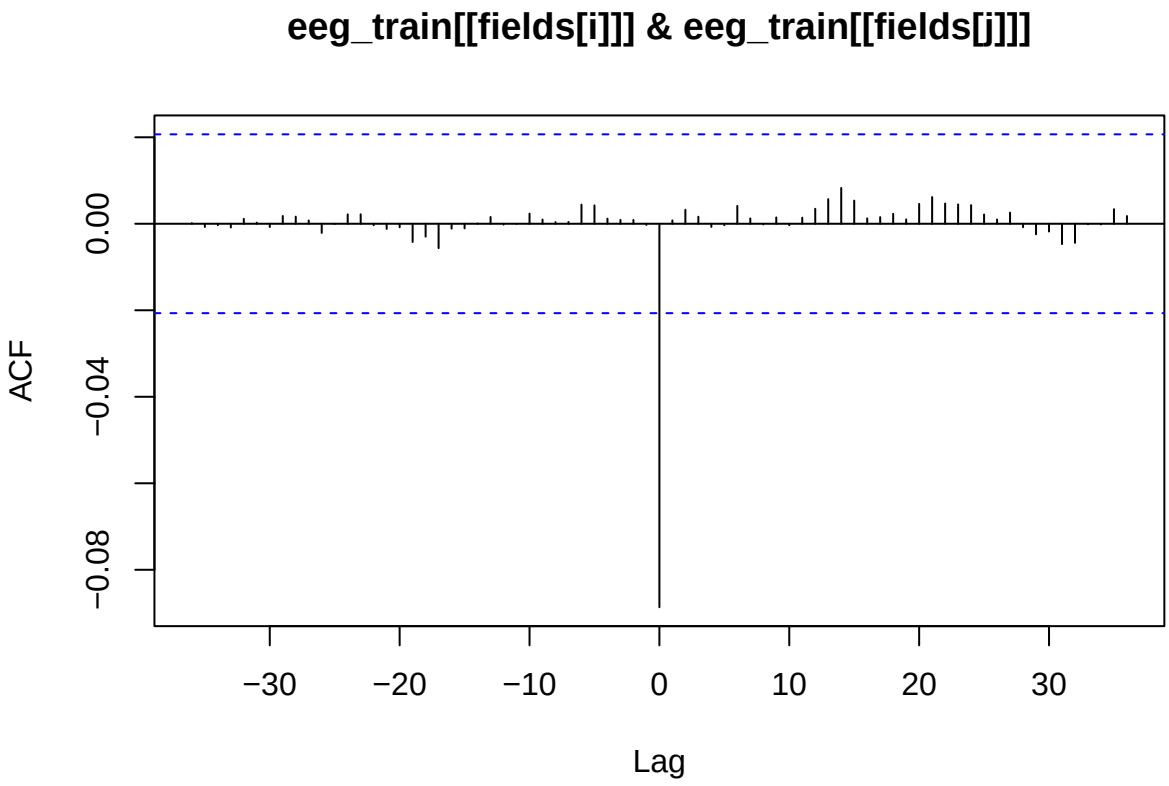

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−30
−20
−10
0
10
20
30
Lag

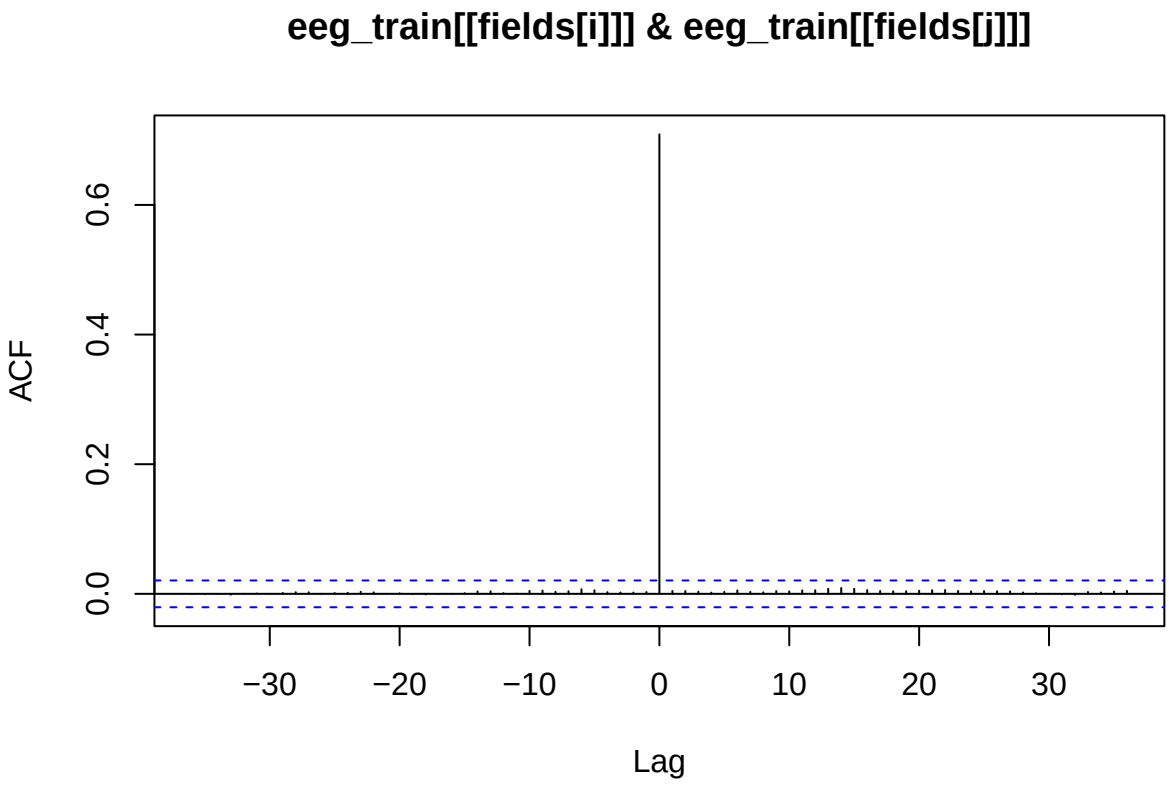

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag

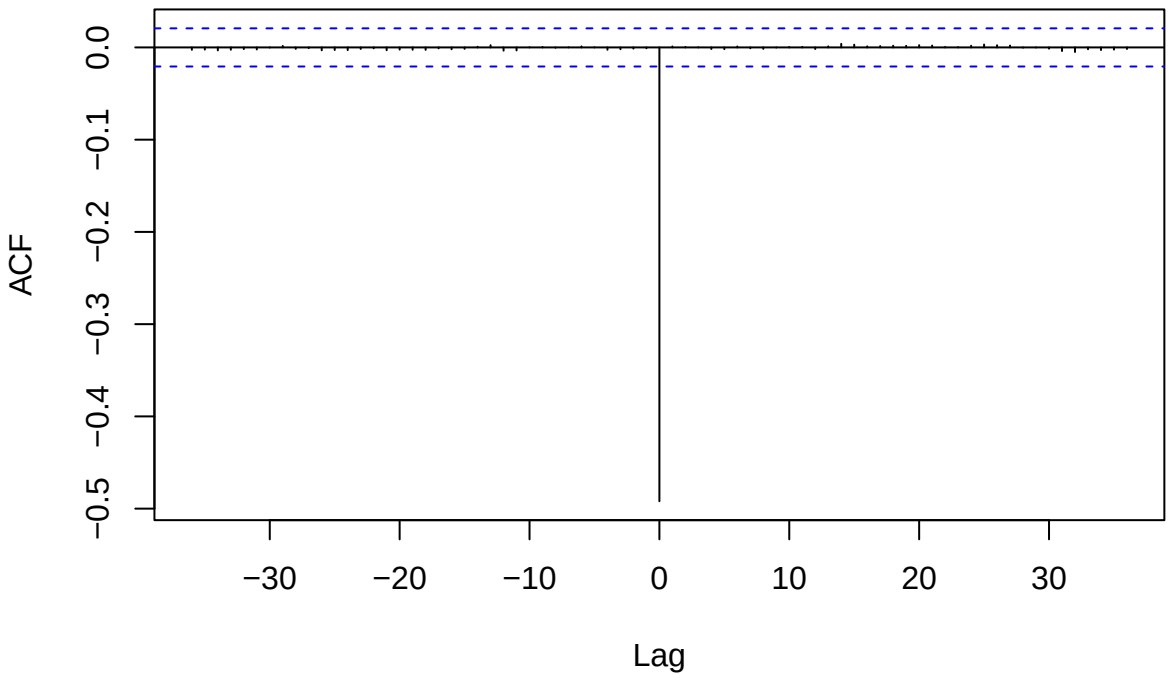

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.05
0.10
0.15
0.20
−30
−20
−10
0
10
20
30
Lag

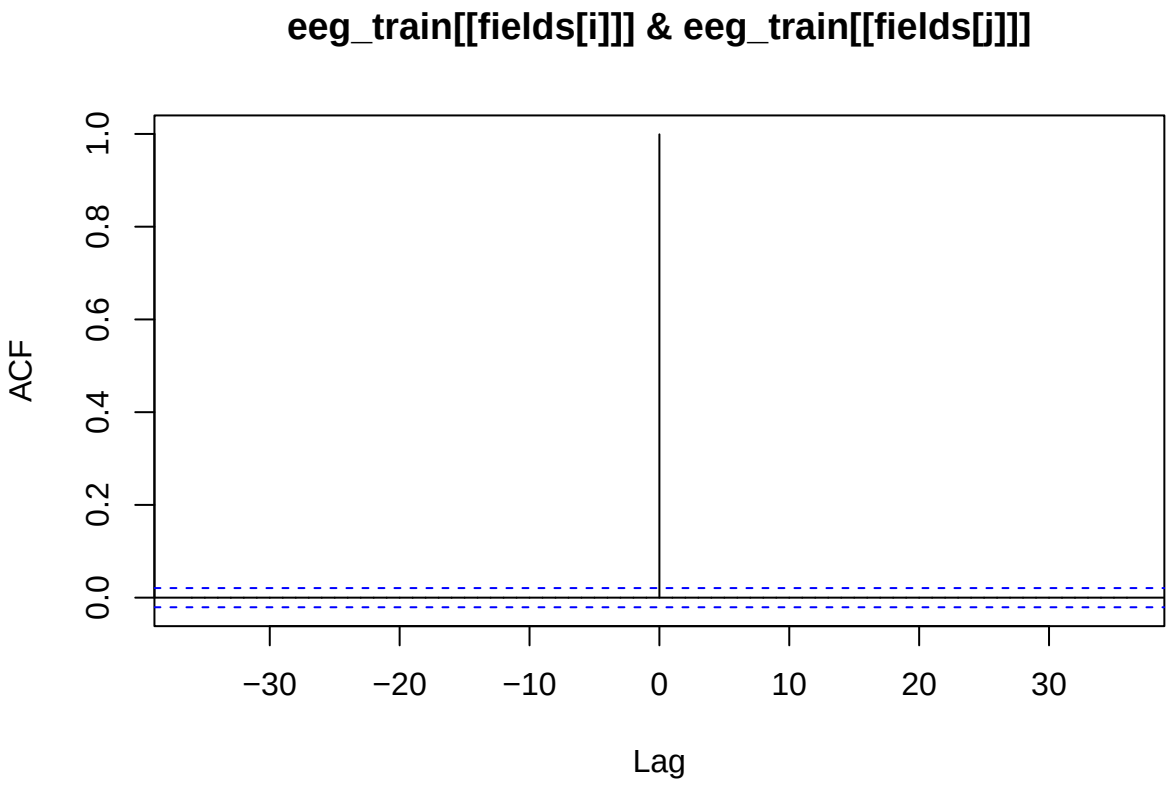

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag

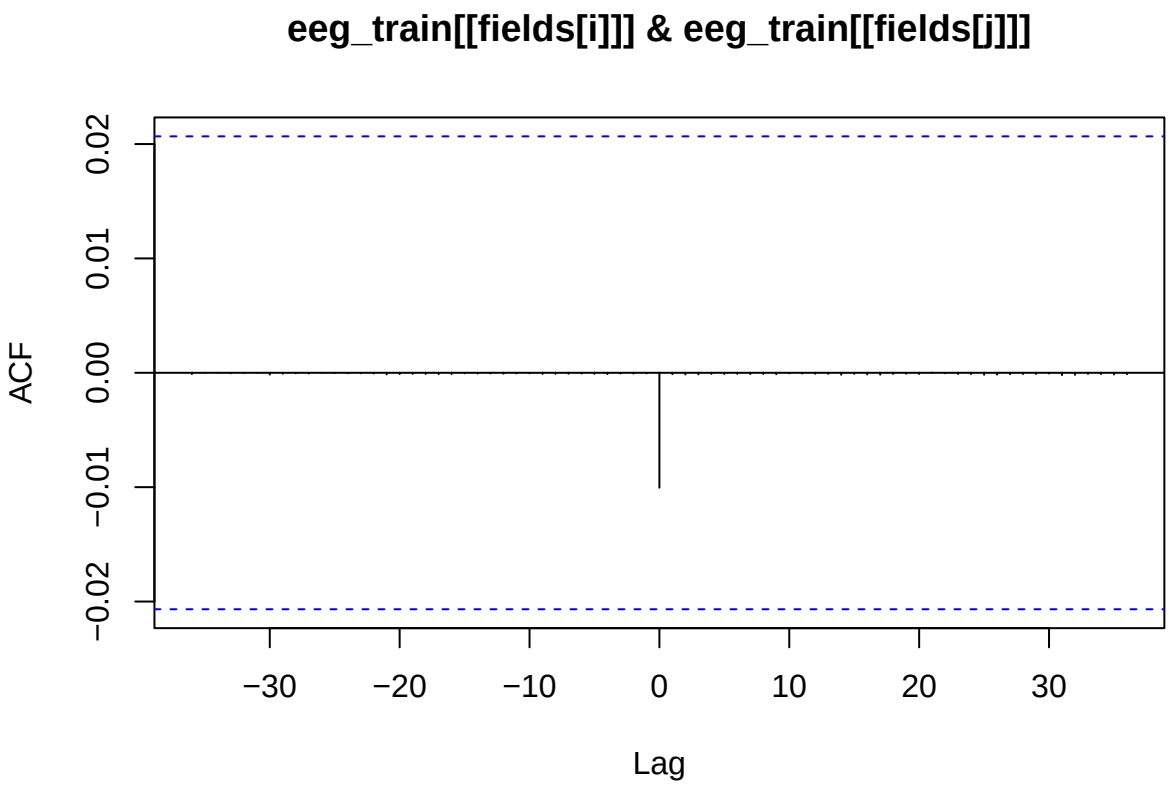

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.10
0.06
0.02
−0.02
−30
−20
−10
0
10
20
30
Lag

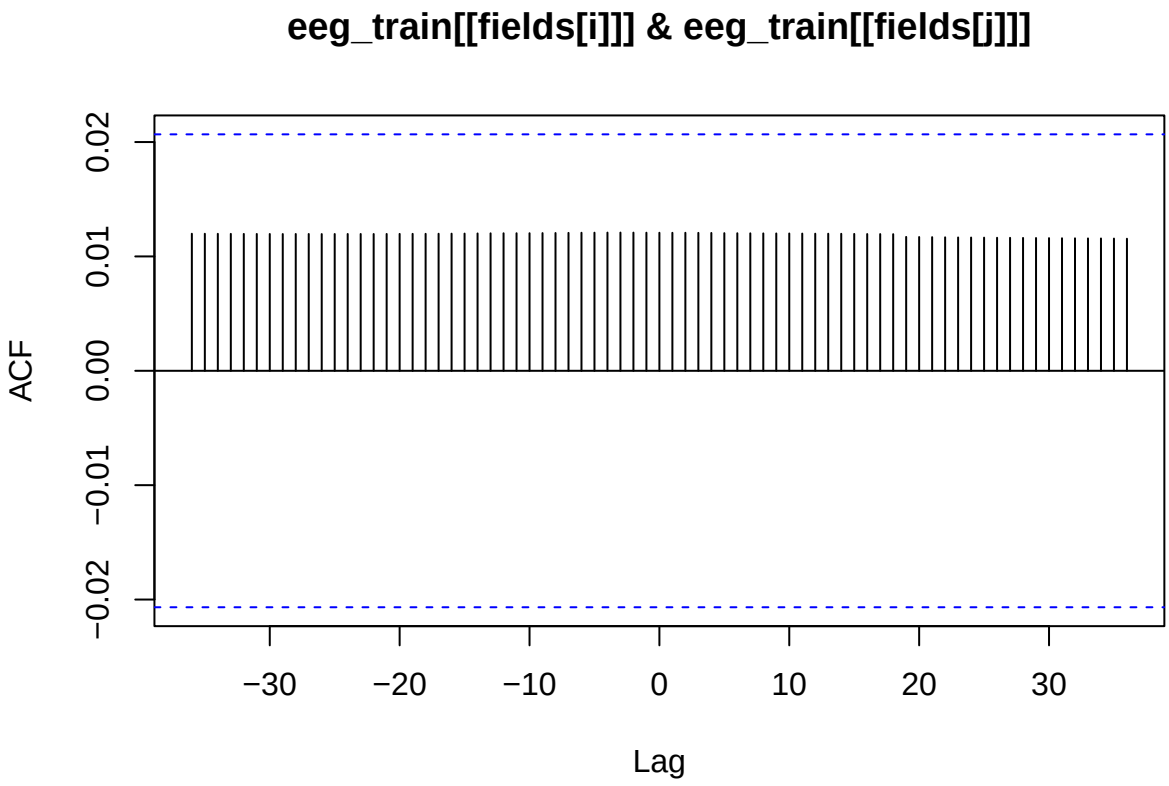

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
−0.04
−0.08
−30
−20
−10
0
10
20
30
Lag

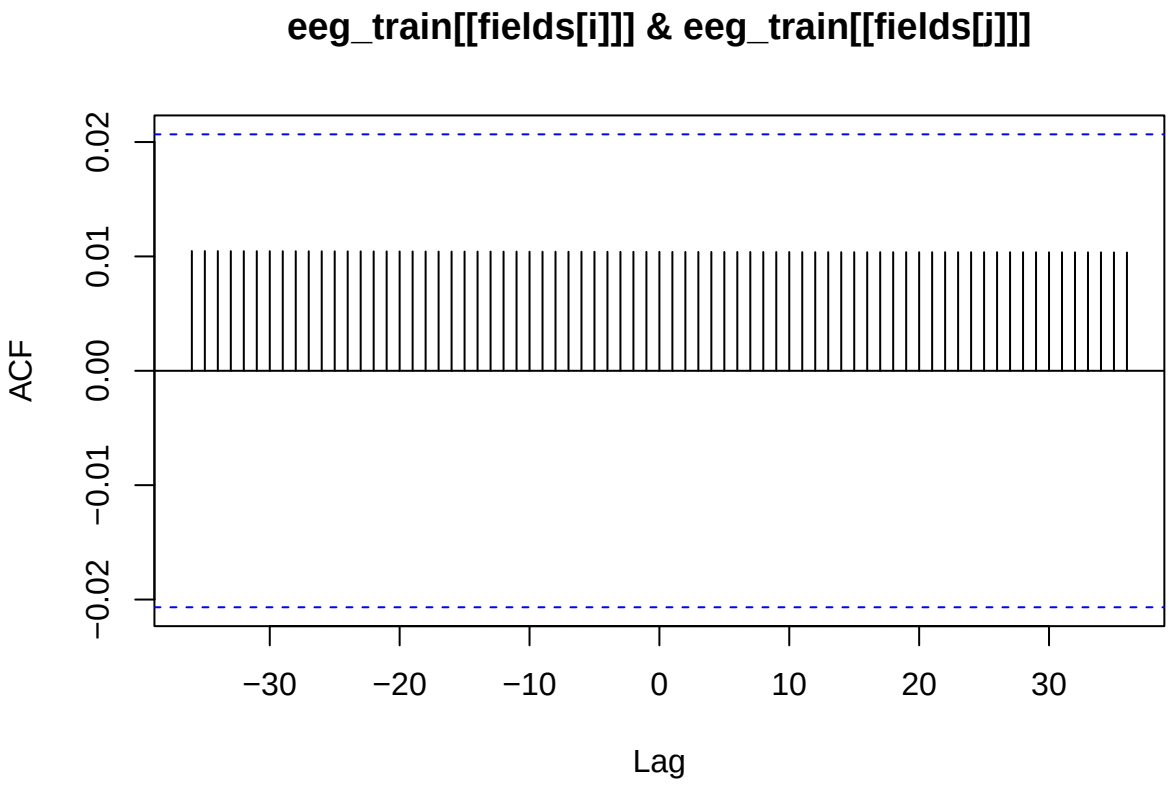

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.2
0.4
0.6
−30
−20
−10
0
10
20
30
Lag

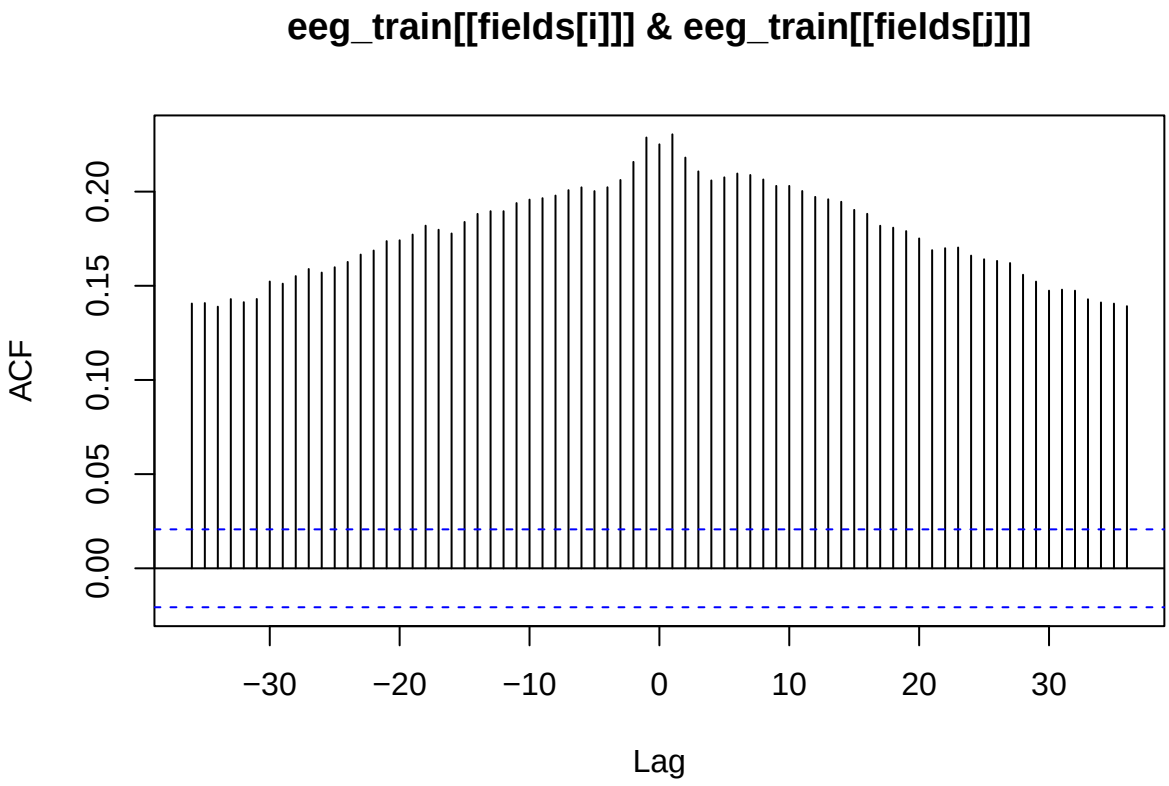

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−30
−20
−10
0
10
20
30
Lag

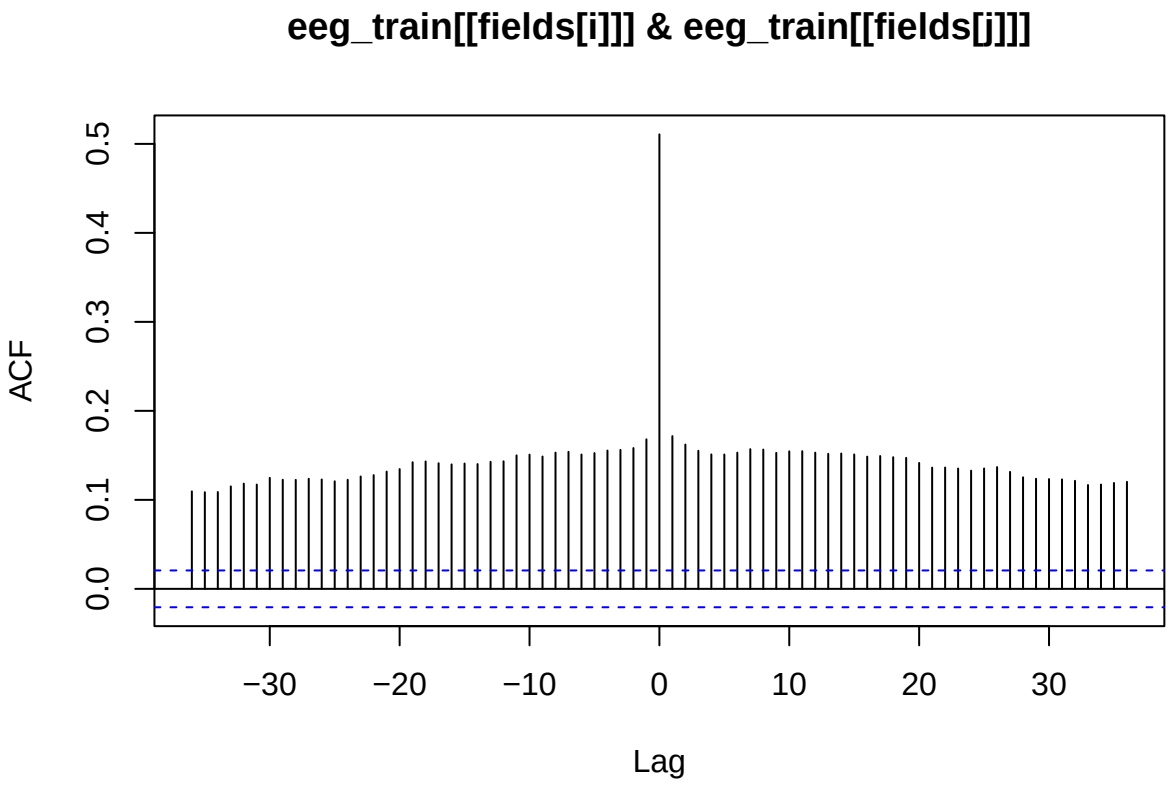

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.2
0.4
0.6
0.8
1.0
−30
−20
−10
0
10
20
30
Lag

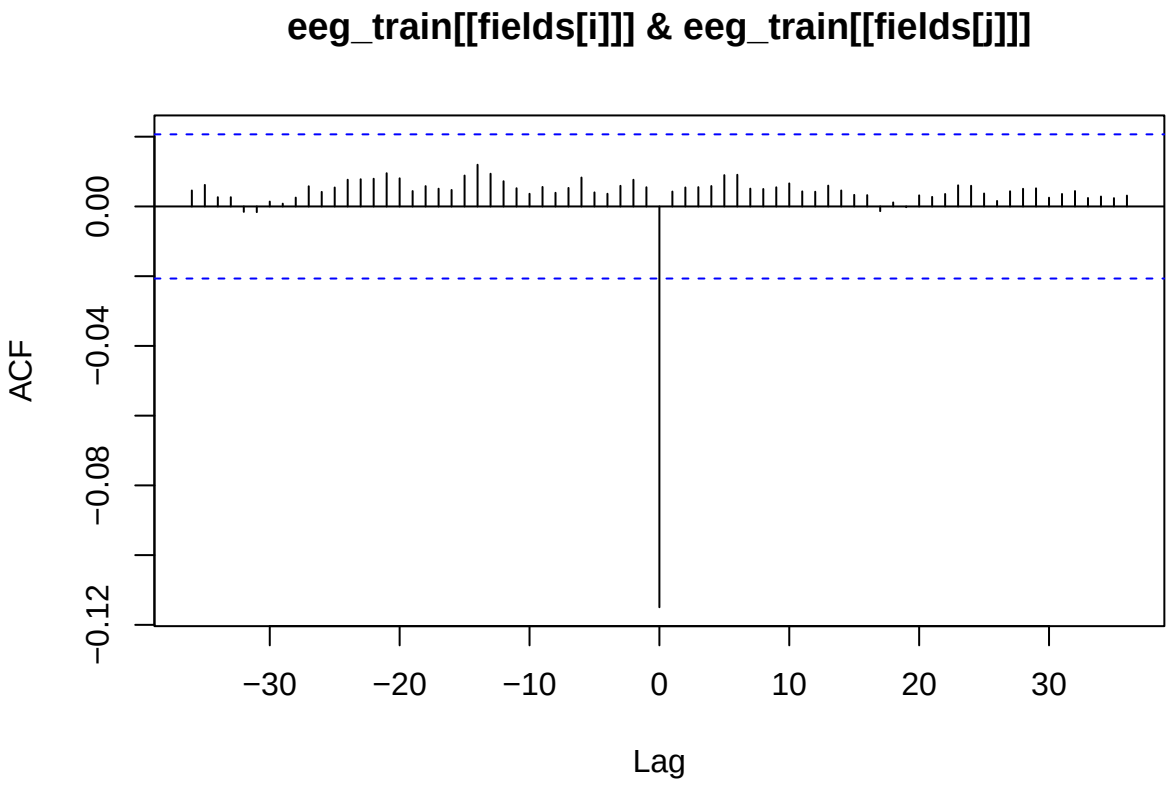

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag

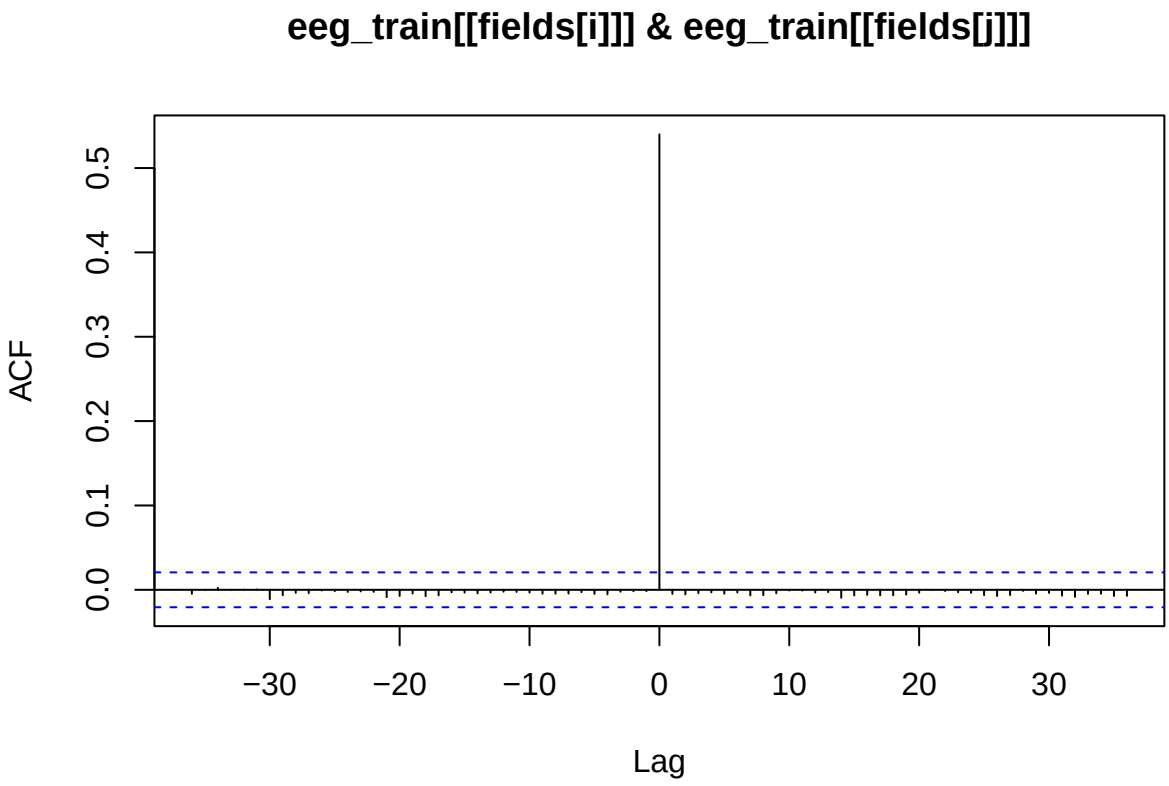

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag

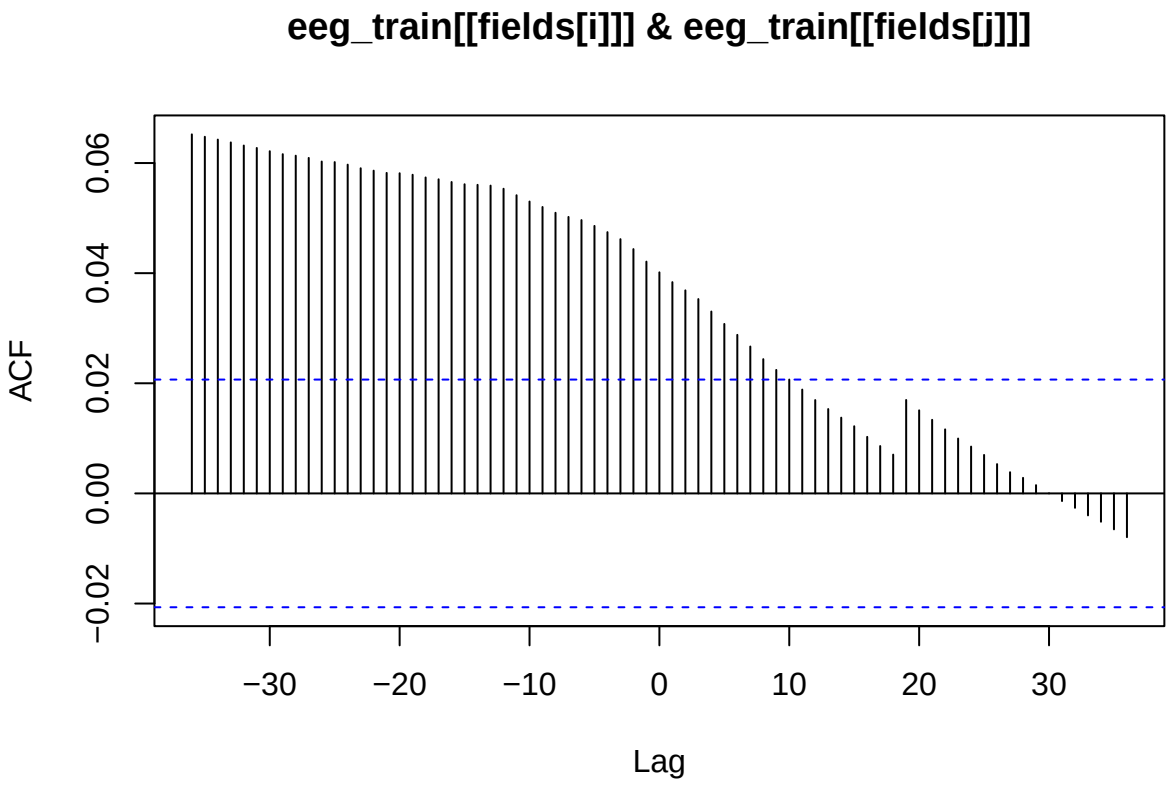

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag

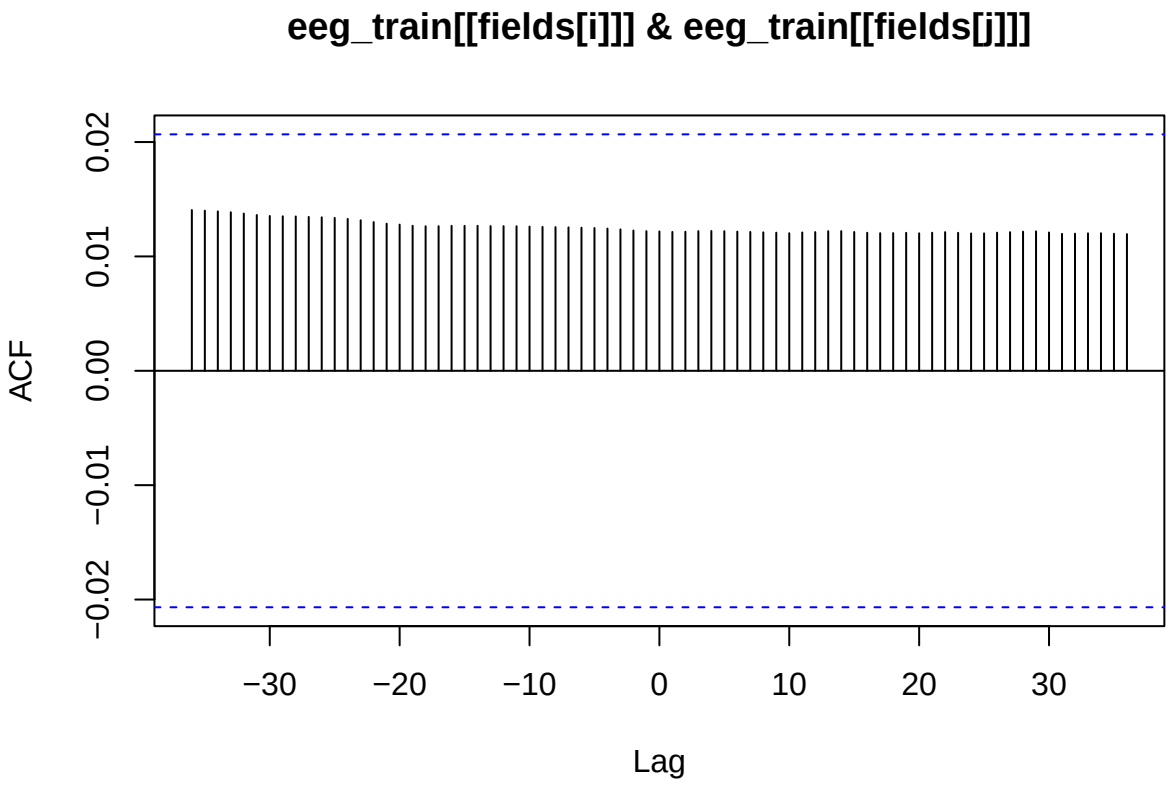

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.05
0.10
0.15
0.20
−30
−20
−10
0
10
20
30
Lag

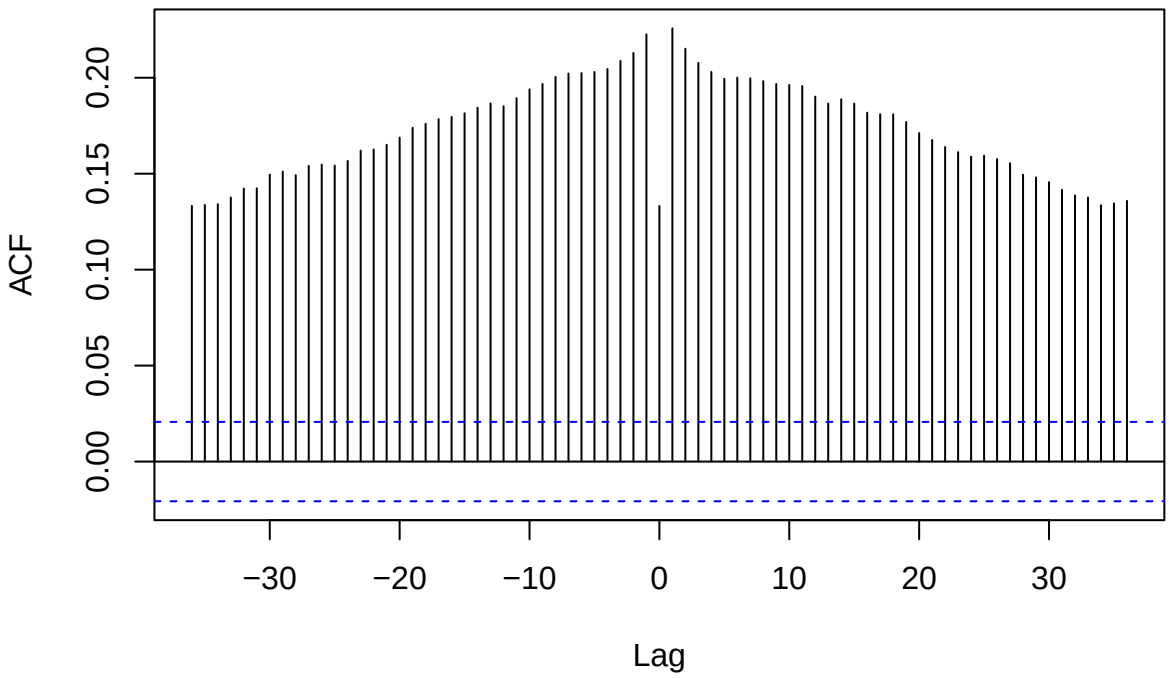

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
0.5
−30
−20
−10
0
10
20
30
Lag

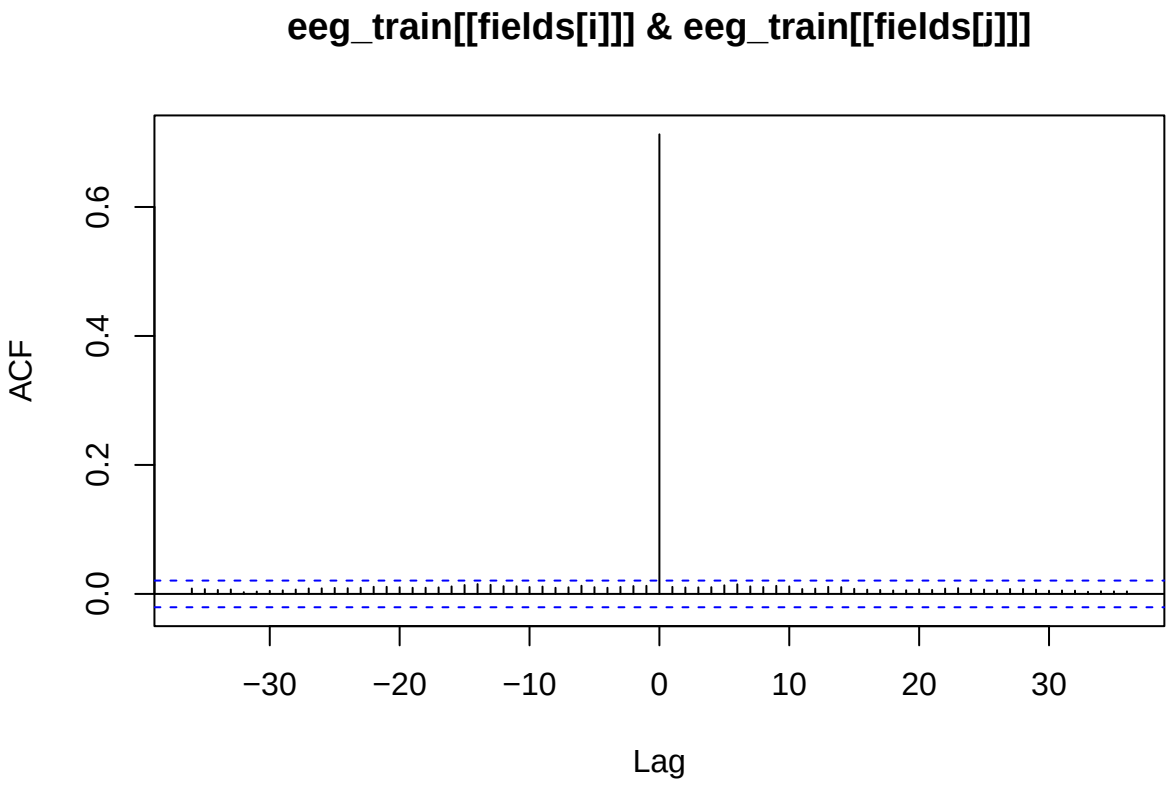

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
−0.04
−0.08
−0.12
−30
−20
−10
0
10
20
30
Lag

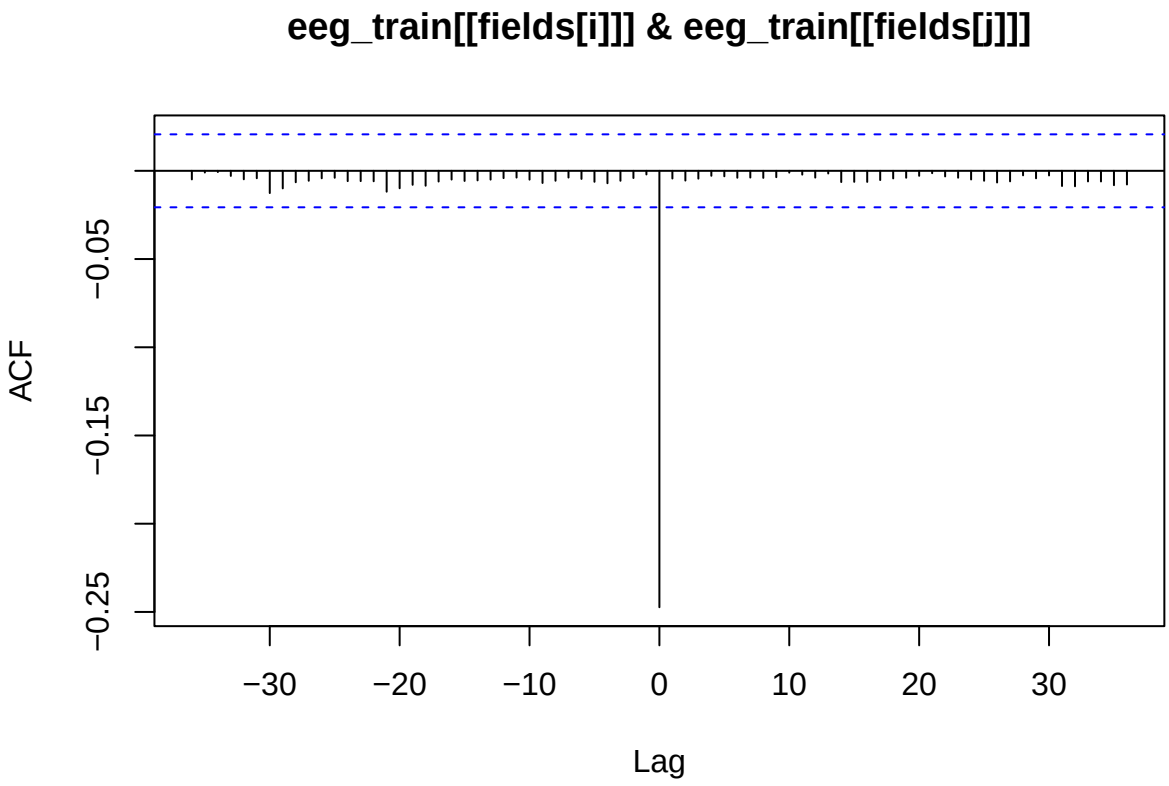

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.1
0.2
0.3
0.4
0.5
−30
−20
−10
0
10
20
30
Lag

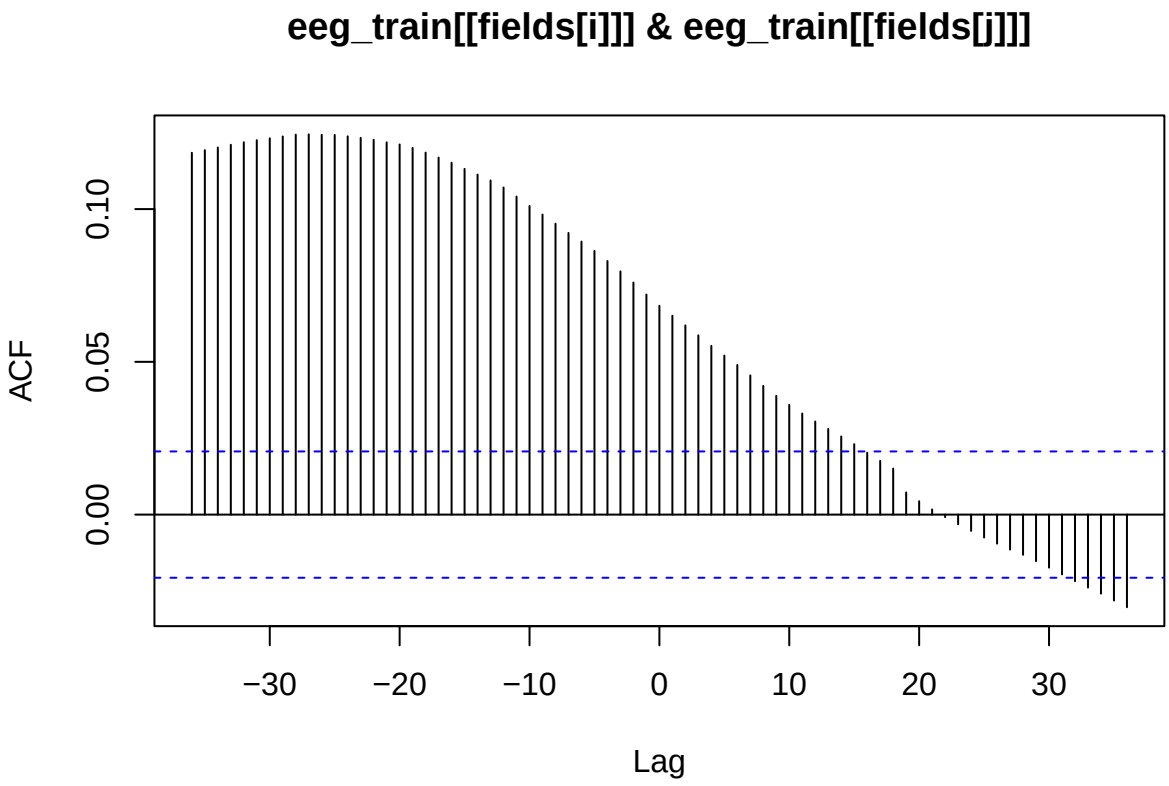

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.06
0.04
0.02
0.00
−0.02
−30
−20
−10
0
10
20
30
Lag

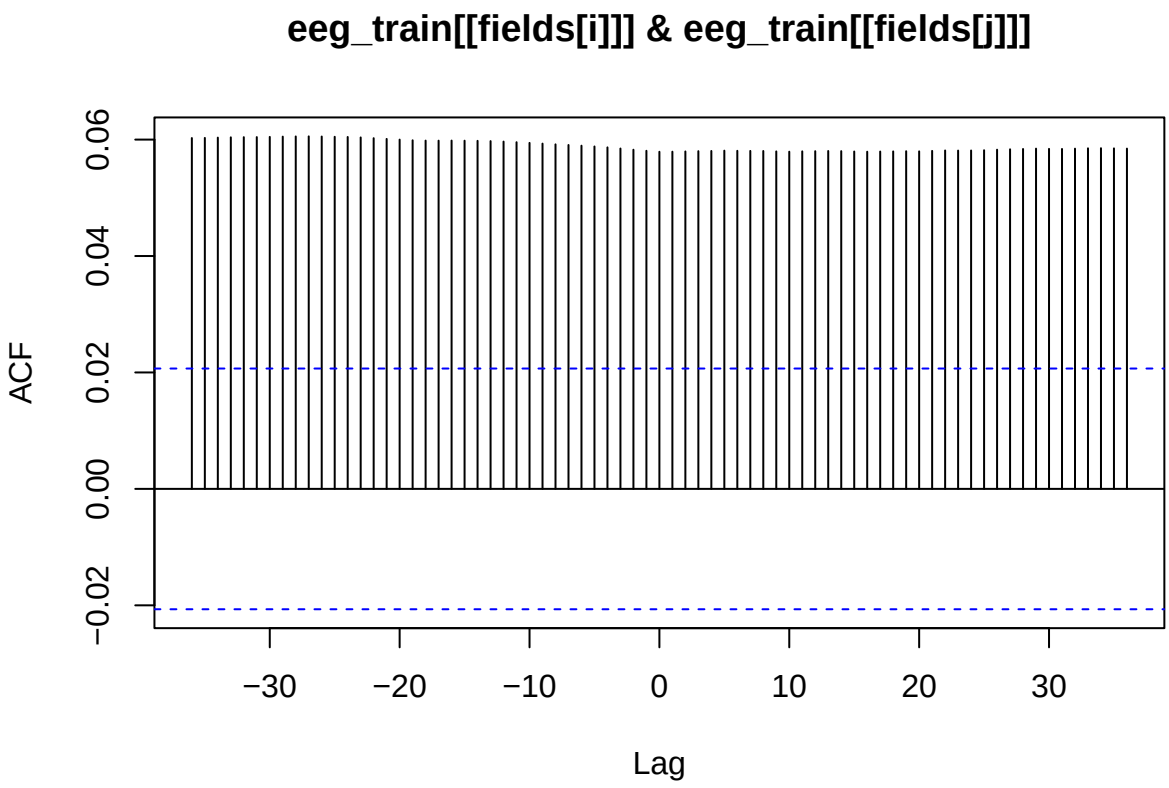

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag

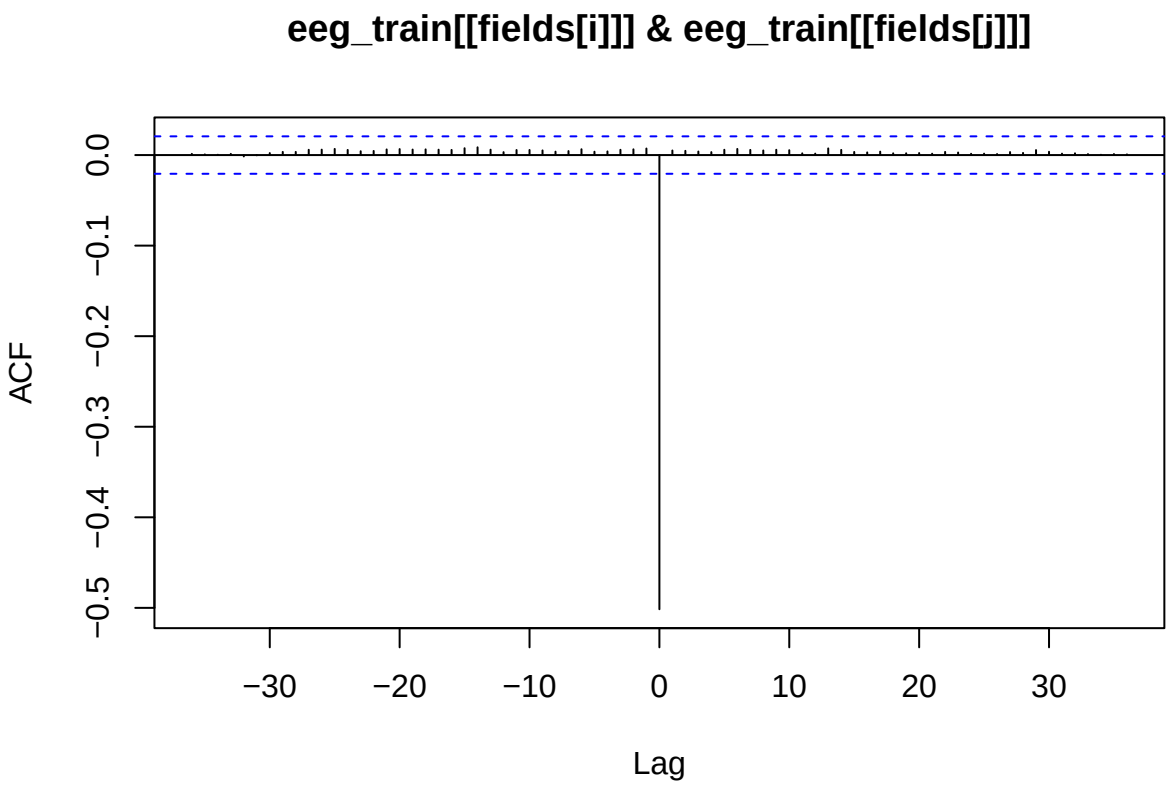

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.05
0.10
0.15
0.20
−30
−20
−10
0
10
20
30
Lag

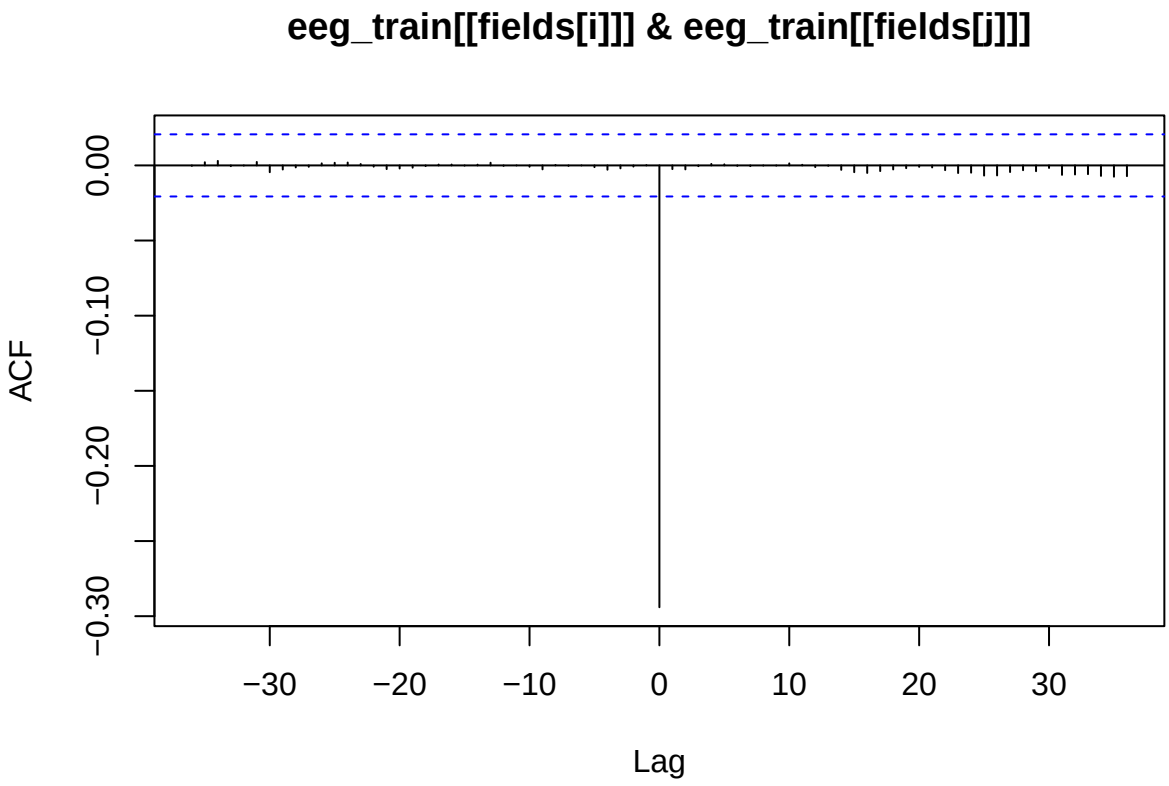

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
0.2
0.4
0.6
−30
−20
−10
0
10
20
30
Lag

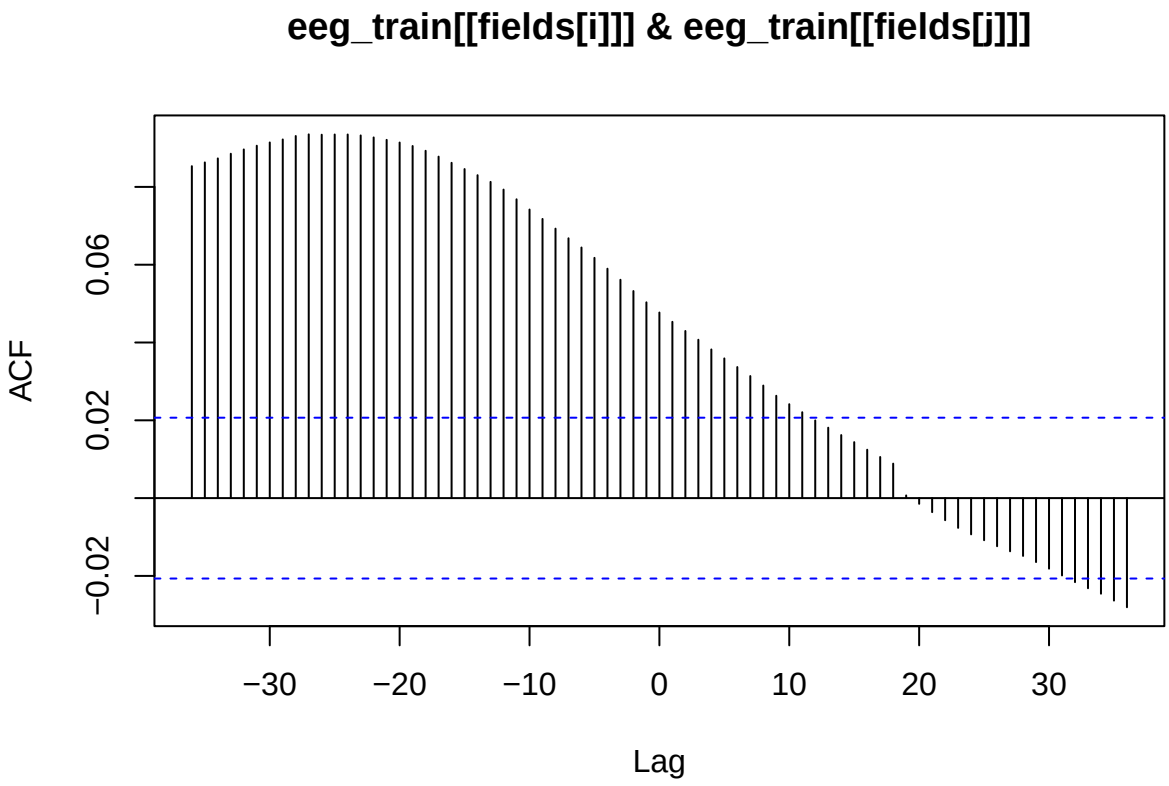

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
−0.05
−0.15
−0.25
−30
−20
−10
0
10
20
30
Lag


eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
0.05
0.10
−30
−20
−10
0
10
20
30
Lag

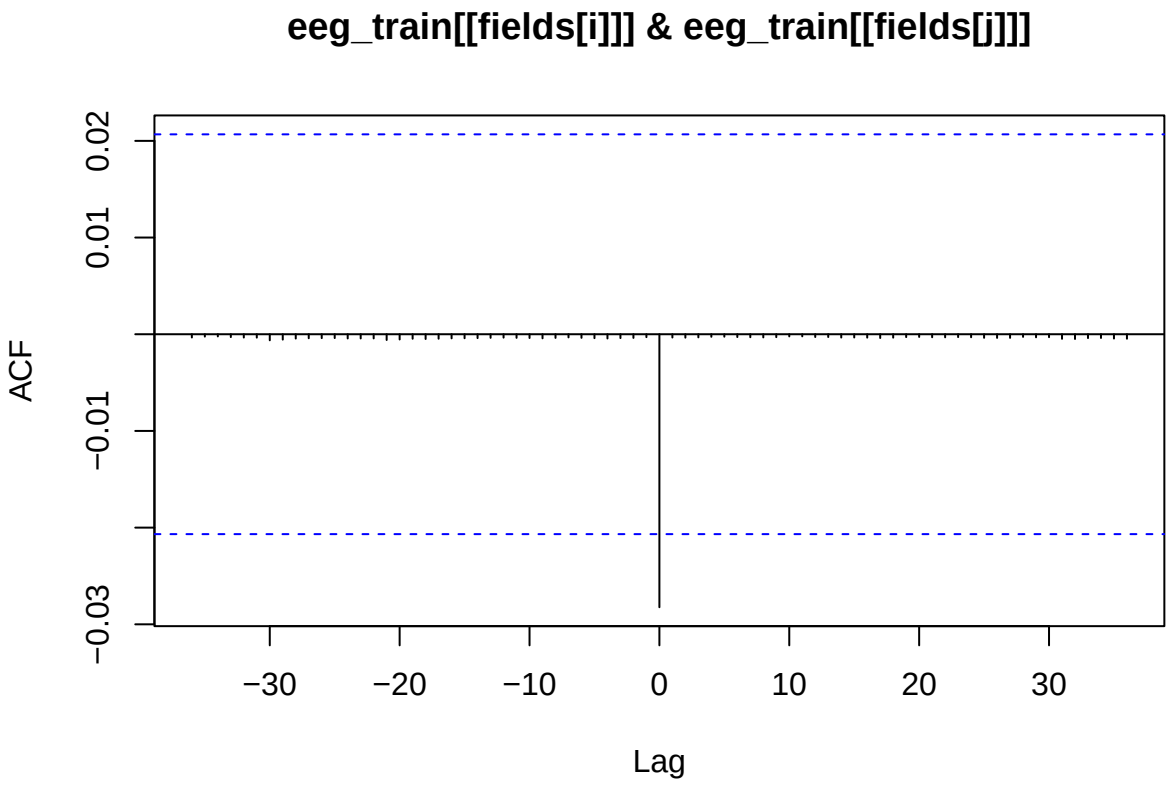

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.06
0.04
0.02
0.00
−0.02
−30
−20
−10
0
10
20
30
Lag

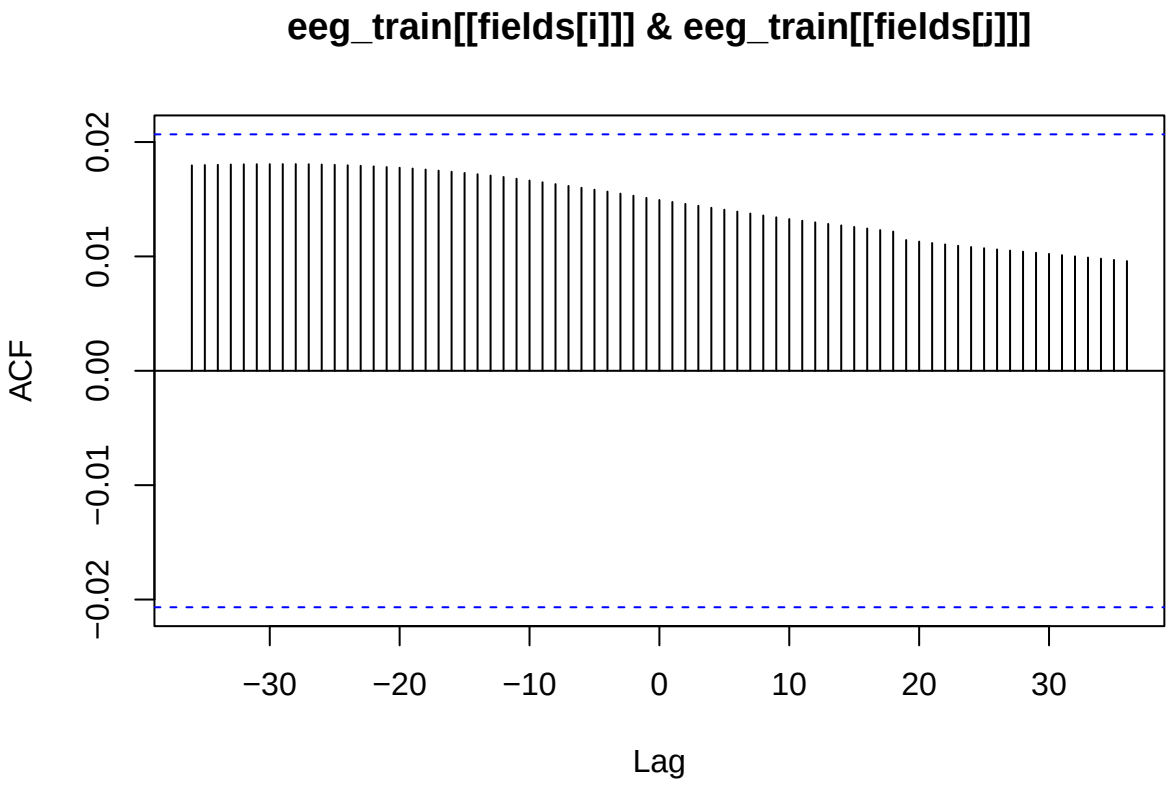

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−30
−20
−10
0
10
20
30
Lag

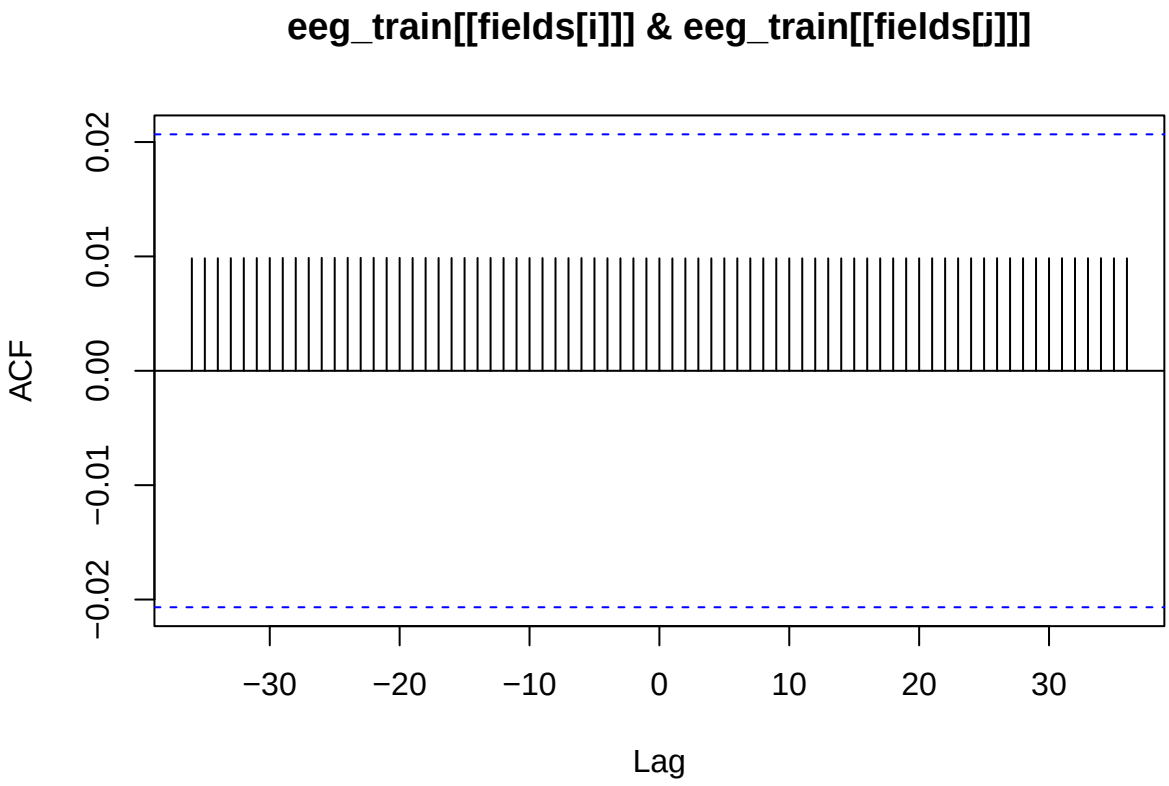

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.00
−0.10
−0.20
−0.30
−30
−20
−10
0
10
20
30
Lag

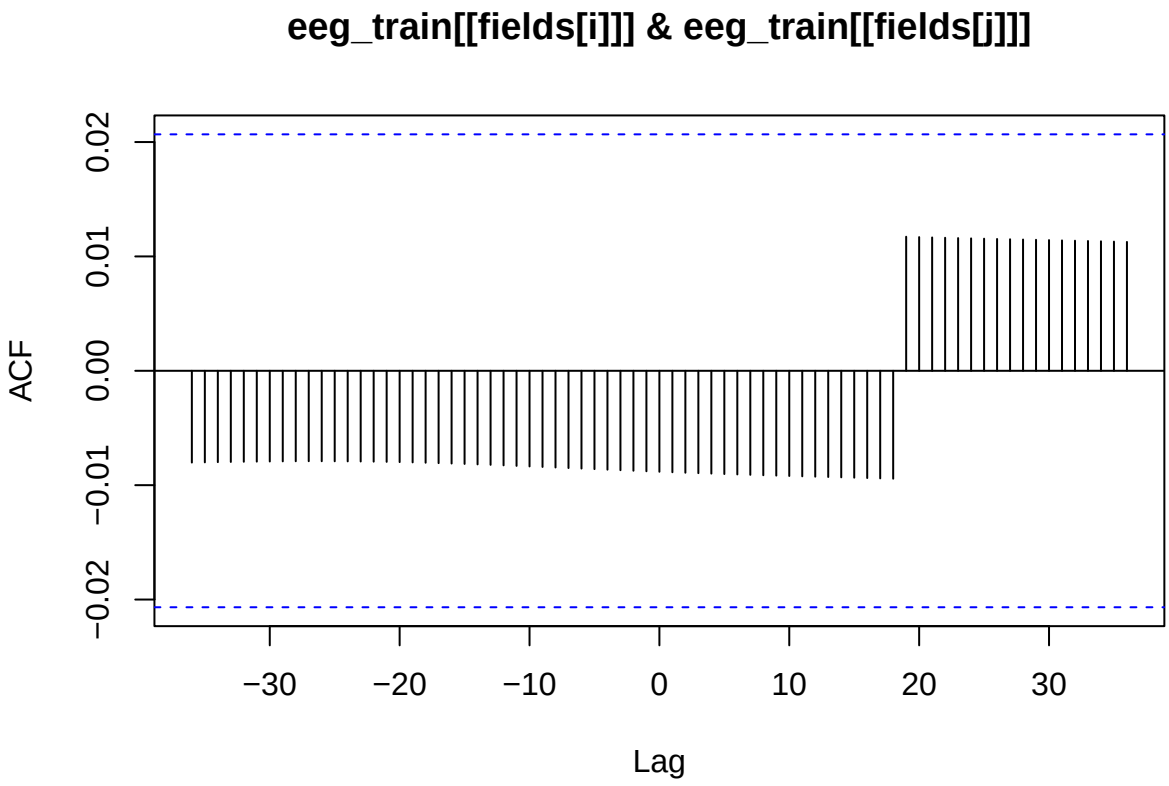

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.06
0.02
−0.02
−30
−20
−10
0
10
20
30
Lag

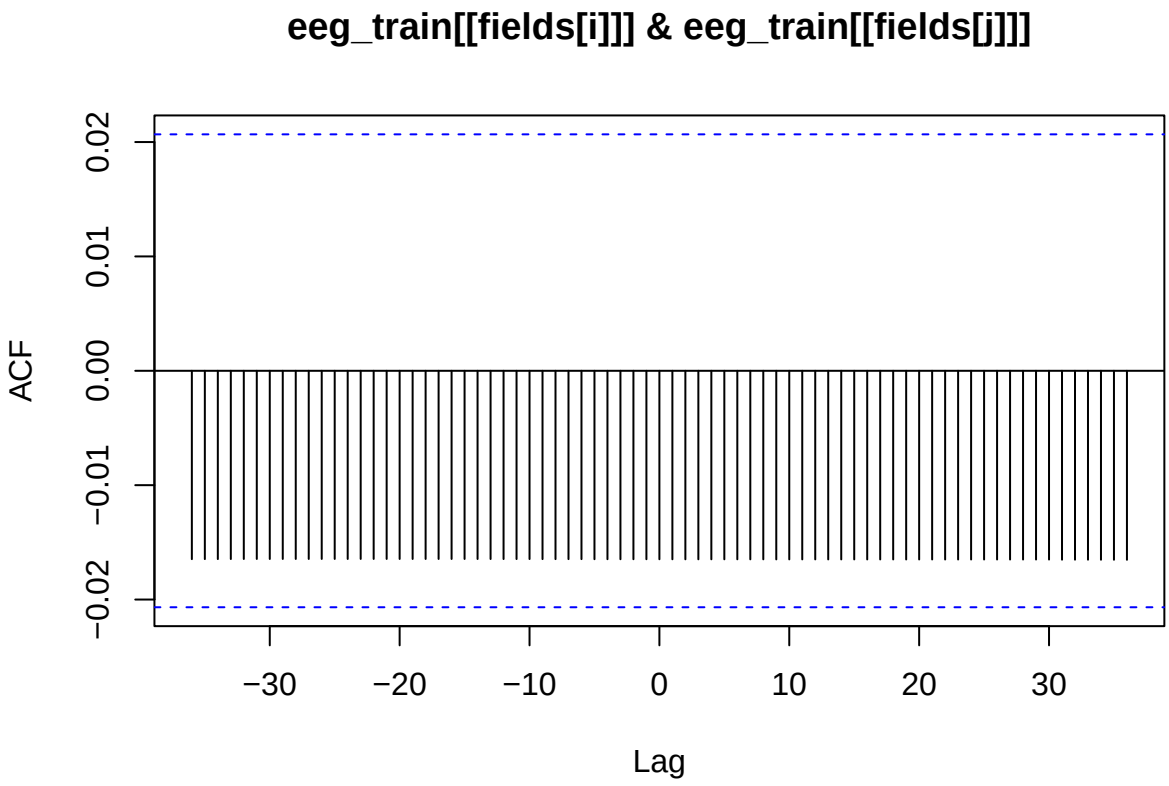

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
−0.01
−0.03
−30
−20
−10
0
10
20
30
Lag

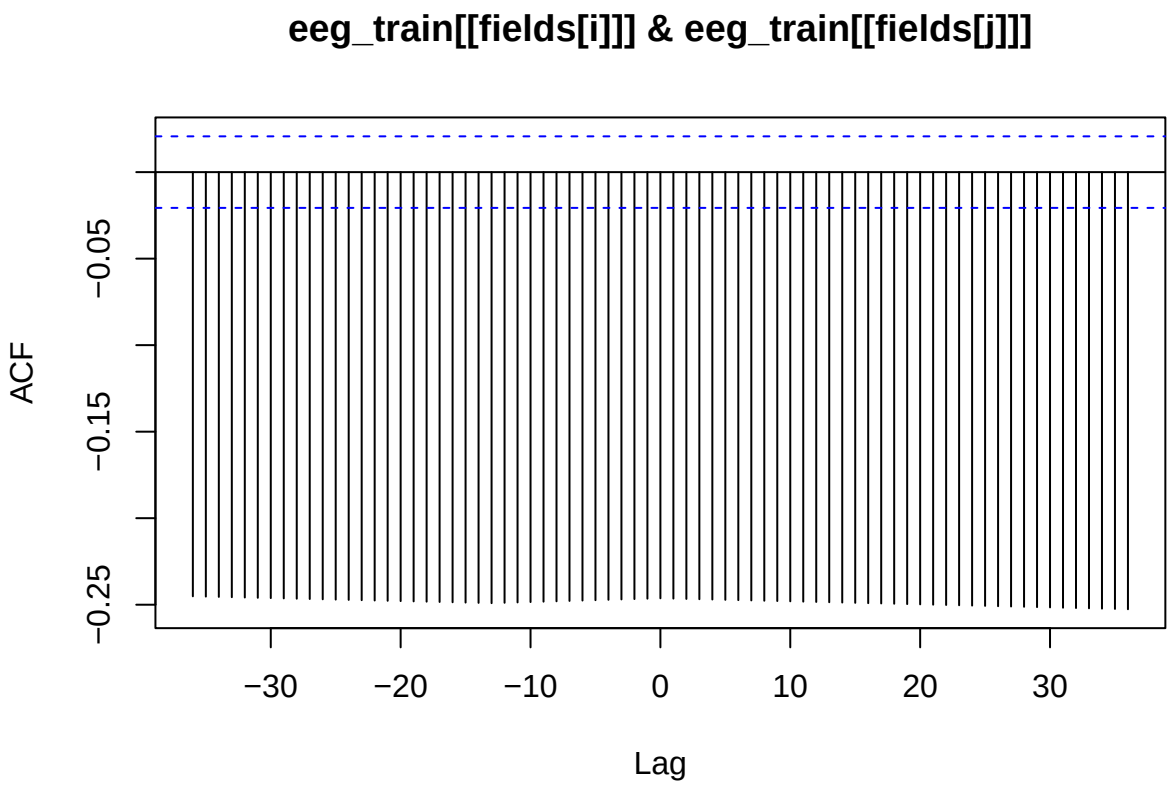

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
−0.01
−0.03
−30
−20
−10
0
10
20
30
Lag


eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag


eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag

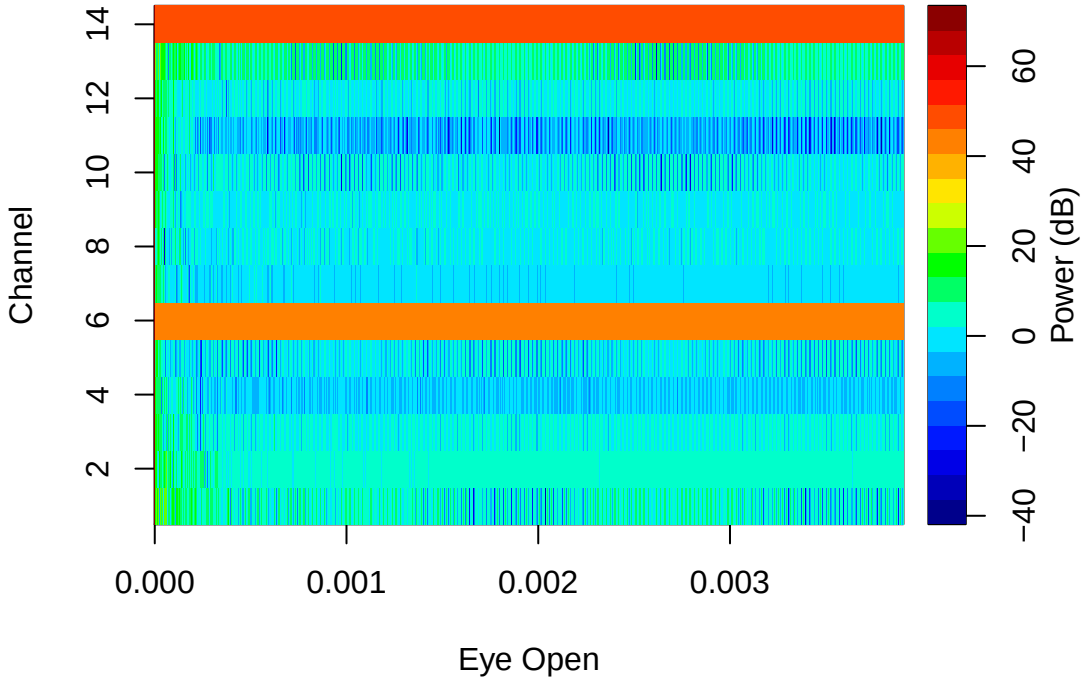

eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag


eeg_train[[fields[i]]] & eeg_train[[fields[j]]]
ACF
0.02
0.01
0.00
−0.01
−0.02
−30
−20
−10
0
10
20
30
Lag

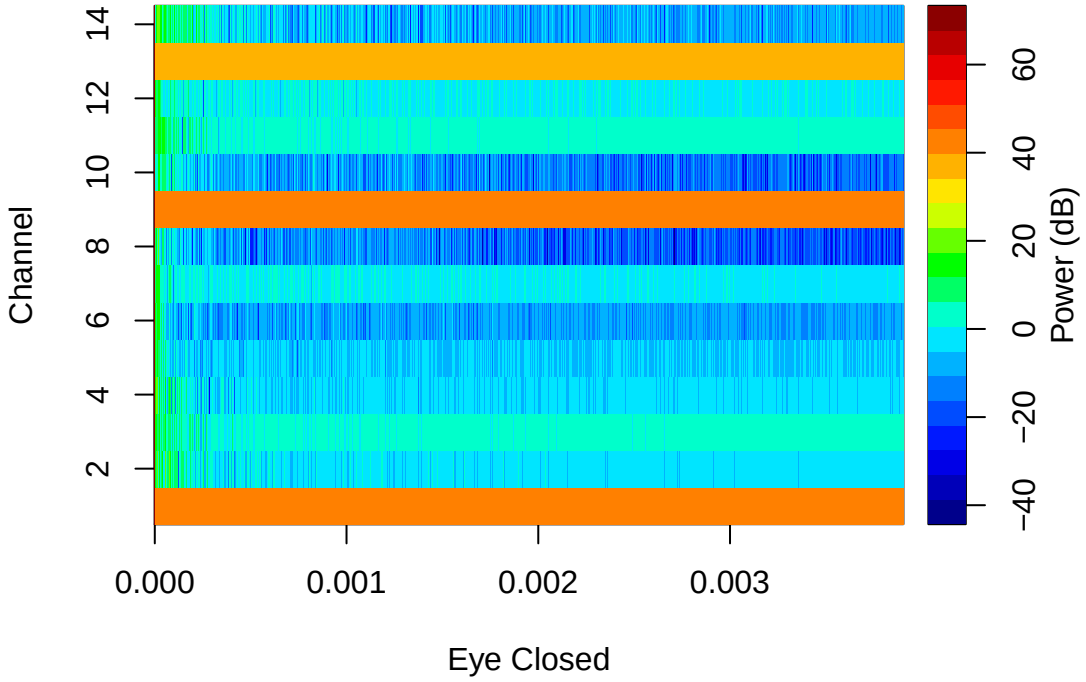


Frequency-Space We can also explore the data in frequency space by using a Fast Fourier Transform. After the FFT we can summarise the distributions of frequencies by their density across the power spectrum. This will let us see if there any obvious patterns related to eye status in the overall frequency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

It appears that there are differences in the power spectral densities between the open and closed eye states.

In the open eye state, Channel 14 and 6 shows significantly higher power compared to other channels. This indicates that there may be strong neural activity or electrical signals in that specific channel when the eyes are open.

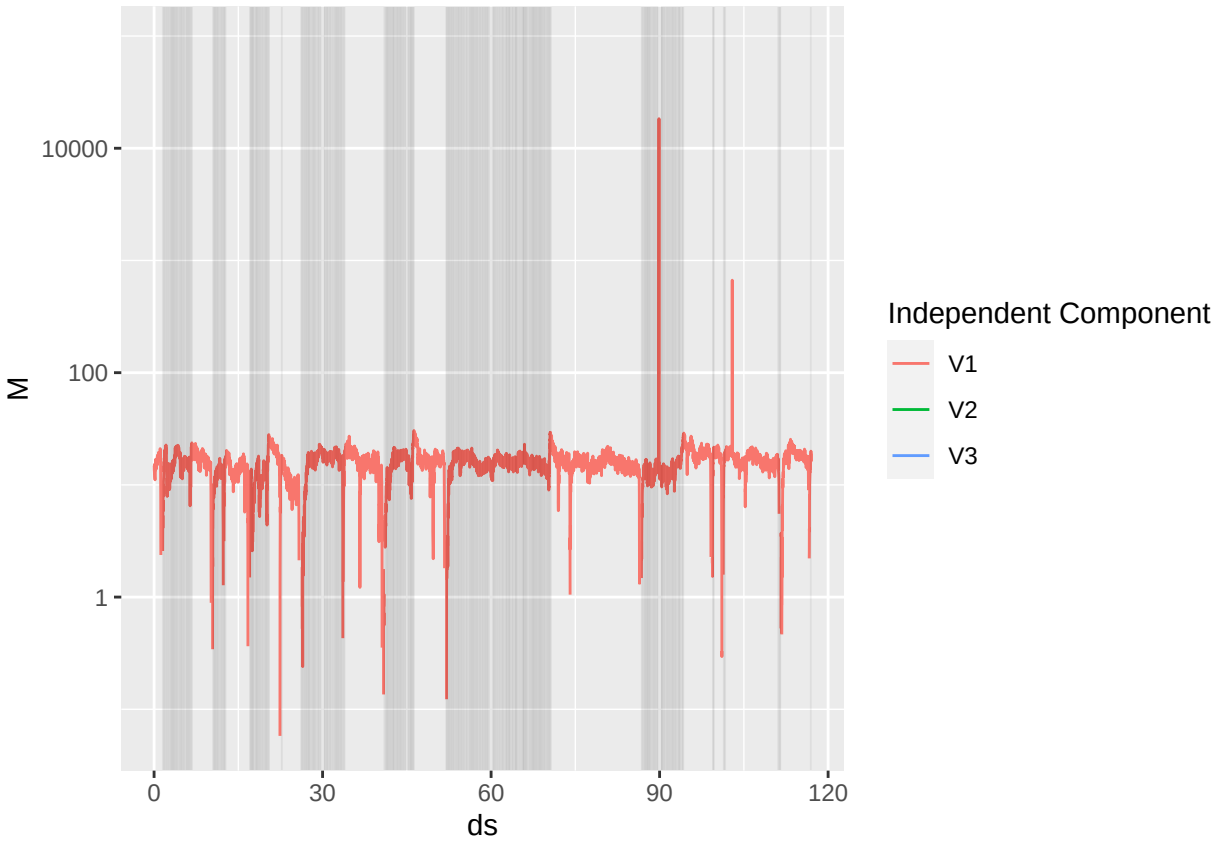
In the closed eye state, Channels 13, 9, and 1 exhibit higher power, although not as high as the power observed in channel 14 during the open eye state. This suggests that there might be relatively increased neural activity or electrical signals in these channels when the eyes are closed compared to other channels.

These observations highlight the potential differences in brain activity or neural processes associated with different eye states. The variations in power across specific channels can provide insights into the functional connectivity and activity patterns of the brain during different cognitive or physiological states.

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?

Yes, the significant down pick at the start of eye opening observed in the independent component signals suggests that eye opening is related to an independent component of activity across the electrodes. The independent component analysis (ICA) aims to separate the mixed signals recorded by the electrodes into independent sources or components. In this case, the independent components represent different sources of neuronal activity.

The observed down pick in the independent component signals during eye opening indicates that there is a specific component of activity that is consistently affected or modulated when the eyes are opened. This suggests that the neural processes associated with eye opening contribute to the generation or modulation of this particular independent component.

By isolating and examining the independent components, ICA allows us to identify and study the underlying sources of activity that contribute to the recorded signals. Therefore, the presence of a distinct and consistent pattern related to eye opening in one or more independent components suggests a relationship between eye opening and the identified sources of neuronal 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:

```
# 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_xgb <- 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)
```

```
## ##### 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.

```
library(caret)
eeg_train_to_net <- dplyr::select(eeg_train, -ds)

# Define the training control parameters
train_control <- trainControl(method = "cv", number = 5)

# Train the neural network model
model_rf <- train(eyeDetection ~ .,
                  data = eeg_train_to_net,
```

```
                  method = "rf",
                  trControl = train_control)

print(model_rf)
```

```
## Random Forest
##
## 8988 samples
##   14 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 7191, 7190, 7190, 7190, 7191
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9096584  0.8167006
##    8    0.9163349  0.8304947
##   14    0.9138874  0.8255521
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 8.
```

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

First i should check the performance of two models on validation dataset:

for XGBOOST model :

```
# Convert the test dataset to a numeric matrix
eeg_validate_matrix <- as.matrix(dplyr::select(eeg_validate, -eyeDetection, -ds))
eeg_validate_labels <- as.numeric(eeg_validate$eyeDetection) - 1

# Build the xgb.DMatrix object for the test dataset
valid_data <- xgb.DMatrix(data = eeg_validate_matrix)

# Predict eye opening status on the test dataset
test_predictions <- predict(model_xgb, newdata = valid_data)

# Convert the predictions to binary labels (0 or 1)
test_predictions <- ifelse(test_predictions > 0.5, 1, 0)

# Calculate accuracy on the test dataset
test_accuracy <- sum(test_predictions == eeg_validate_labels) / length(eeg_validate_labels)

# Print the test accuracy
print(paste("Test Accuracy:", test_accuracy))
```

```
## [1] "Test Accuracy: 0.831775700934579"
```

```r
confusion_matrix <- confusionMatrix(factor(test_predictions, levels = c(0, 1)),
                                    factor(eeg_validate_labels, levels = c(0, 1)))

# Print the confusion matrix
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1434  303
##          1  201 1058
##
##                Accuracy : 0.8318
##                  95% CI : (0.8179, 0.845)
##     No Information Rate : 0.5457
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6586
##
##  Mcnemar's Test P-Value : 6.831e-06
##
##             Sensitivity : 0.8771
##             Specificity : 0.7774
##          Pos Pred Value : 0.8256
##          Neg Pred Value : 0.8403
##              Prevalence : 0.5457
##          Detection Rate : 0.4786
##    Detection Prevalence : 0.5798
##       Balanced Accuracy : 0.8272
##
##        'Positive' Class : 0
##
```

for my Random Forest model :

```r
# Convert the test dataset to include only the predictor variables
eeg_valid_to_net <- dplyr::select(eeg_validate, -ds)

# Predict eye opening status on the test dataset
valid_predictions_rf <- predict(model_rf, newdata = eeg_valid_to_net)

# Create a confusion matrix
confusion_matrix_rf <- confusionMatrix(factor(valid_predictions_rf, levels = c(0, 1)),
                                       factor(eeg_validate$eyeDetection, levels = c(0, 1)))

# Print the confusion matrix
print(confusion_matrix_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction    0    1
##          0 1535  139
##          1  100 1222
##
##                Accuracy : 0.9202
##                  95% CI : (0.9099, 0.9297)
##     No Information Rate : 0.5457
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8387
##
##  Mcnemar's Test P-Value : 0.01397
##
##             Sensitivity : 0.9388
##             Specificity : 0.8979
##          Pos Pred Value : 0.9170
##          Neg Pred Value : 0.9244
##              Prevalence : 0.5457
##          Detection Rate : 0.5123
##    Detection Prevalence : 0.5587
##       Balanced Accuracy : 0.9184
##
##        'Positive' Class : 0
##
```

So now we know that the Random Forest performs better, let's report it's performance on the test set:

```r
# Convert the test dataset to include only the predictor variables
eeg_test_to_net <- dplyr::select(eeg_test, -ds)

# Predict eye opening status on the test dataset
test_predictions_rf <- predict(model_rf, newdata = eeg_test_to_net)

# Create a confusion matrix
confusion_matrix_rf <- confusionMatrix(factor(test_predictions_rf, levels = c(0, 1)),
                                       factor(eeg_test$eyeDetection, levels = c(0, 1)))

# Print the confusion matrix
print(confusion_matrix_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1625  132
##          1   81 1158
##
##                Accuracy : 0.9289
##                  95% CI : (0.9191, 0.9379)
##     No Information Rate : 0.5694
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8543
```

```
##
##  Mcnemar's Test P-Value : 0.0006127
##
##             Sensitivity : 0.9525
##             Specificity : 0.8977
##          Pos Pred Value : 0.9249
##          Neg Pred Value : 0.9346
##              Prevalence : 0.5694
##          Detection Rate : 0.5424
##    Detection Prevalence : 0.5864
##       Balanced Accuracy : 0.9251
##
##        'Positive' Class : 0
##
```

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

Three possible alternative modeling approaches for the prediction of eye opening from EEGs that we discussed in class but haven't explored in this notebook are:

1. Deep Neural Networks with Learned Representations: One alternative modeling approach for predicting eye opening from EEGs is to use deep neural networks (DNNs) with learned representations. DNNs have shown remarkable success in various domains, including computer vision and natural language processing, and they can also be applied to EEG data analysis. In this approach, the DNN would be designed to learn hierarchical representations of EEG signals, capturing both local and global patterns in the data. This can be achieved through various techniques such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs). By training the DNN on a large dataset of labeled EEG samples, it can learn to automatically extract relevant features and capture complex relationships between the EEG signals and eye opening. However, it's important to address the inter-person variance by individualizing the tuning of the DNN models to account for individual differences in EEG patterns and eye opening behavior.

2. Attention Mechanisms for EEG Analysis: Another alternative approach is to incorporate attention mechanisms in the modeling of EEG data for eye opening prediction. Attention mechanisms have been successfully used in various deep learning models to focus on relevant parts of the input data. In the context of EEGs, attention mechanisms can help the model selectively attend to specific regions or time segments of the EEG signals that are most informative for predicting eye opening. By assigning different weights or attention scores to different electrodes or time points in the EEG data, the model can dynamically adjust its focus and give more importance to the most relevant information. This approach can be particularly useful in handling the inter-person variance and capturing individual-specific patterns in the EEG data. Additionally, attention mechanisms can provide interpretability by highlighting the important features contributing to the prediction, allowing for better understanding of the model's decision-making process.

3. Convolutional Neural Networks (CNN): CNNs are deep learning models commonly used for image analysis tasks. They can also be applied to EEG data by treating the EEG signals as 2D or 3D images. CNNs have shown promising results in various EEG-based classification tasks. In the case of predicting eye opening from EEGs, a CNN can be designed to automatically learn relevant spatial and temporal patterns in the EEG signals. The network architecture can consist of convolutional layers to capture local patterns, pooling layers for spatial downsampling, and fully connected layers for classification. CNNs have the advantage of automatically learning hierarchical representations from the data, potentially improving the model's ability to capture complex relationships and achieve higher prediction accuracy.

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

There are several R libraries that can be used to implement alternative modeling approaches for the prediction of eye opening from EEGs. Three such libraries are keras, torch, tensorflow, and h2o.

1. R Library for Deep Neural Networks: "keras" The "keras" library in R provides a high-level interface for implementing deep neural networks. It supports various types of neural network architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and multi-layer perceptrons (MLPs). With "keras", you can easily construct complex deep learning models for EEG analysis, including the models with learned representations discussed earlier. It offers a user-friendly API and allows for flexible model customization, training, and evaluation. Additionally, "keras" supports GPU acceleration for faster computation, making it suitable for large-scale EEG datasets.

2. R Library for Attention Mechanisms: "torch" The "torch" library in R provides functionalities for implementing neural networks with attention mechanisms. It is based on the PyTorch framework and offers a range of tools and modules for building attention-based models. With "torch", you can incorporate attention mechanisms into your deep learning models for EEG analysis, as discussed in one of the alternative approaches. It allows you to define and train attention-based models with ease, providing flexibility in designing attention mechanisms and customizing the model architecture. The "torch" library also provides GPU acceleration for efficient computations, making it suitable for EEG data analysis with attention models.

3. R Library for Convolutional Neural Networks: "tensorflow" The "tensorflow" library in R provides a comprehensive set of tools for implementing convolutional neural networks (CNNs) and other deep learning models. It offers a flexible and scalable framework for building and training CNNs for EEG data analysis. With "tensorflow", you can create custom CNN architectures, apply various types of convolutional and pooling layers, and incorporate other components such as dropout and batch normalization. The library provides efficient computation using GPUs and supports advanced features like transfer learning and model fine-tuning. "tensorflow" also offers high-level APIs, such as the Keras interface, which allows for easier model construction, training, and evaluation.

4. Lastly, h2o is an open-source platform for scalable machine learning that offers a wide range of algorithms and tools for data preprocessing, model training, and deployment. H2O is designed to handle large datasets and can efficiently utilize parallel and distributed computing resources. With h2o, you can apply machine learning techniques such as generalized linear models (GLMs), gradient boosting machines (GBMs), deep learning, and more to your EEG data. The library provides a user-friendly interface, automatic hyperparameter tuning, and supports both R and Python.

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!