Compulsory 3

Group 1

2023-11-28

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

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

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

```r
library(ggplot2)
library(tseries)
```

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

```r
library(forecast)
library(stats)
library(depmixS4)
```

```
## Loading required package: nnet

## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

## Loading required package: Rsolnp

## Loading required package: nlme
```

```
##
## Attaching package: 'nlme'

## The following object is masked from 'package:forecast':
##
##     getResponse

## The following object is masked from 'package:dplyr':
##
##     collapse
```

```r
library(tidyr)
library(zoo)
```

```
##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v readr     2.1.4
## v lubridate 1.9.3     v stringr   1.5.0
## v purrr     1.0.2     v tibble    3.2.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x nlme::collapse() masks dplyr::collapse()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x MASS::select()   masks dplyr::select()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(readr)
library(dtw)
```

```
## Loading required package: proxy
##
## Attaching package: 'proxy'
##
## The following objects are masked from 'package:stats':
##
##     as.dist, dist
##
## The following object is masked from 'package:base':
##
##     as.matrix
##
## Loaded dtw v1.23-1. See ?dtw for help, citation("dtw") for use in publication.
```

```r
library(glmnet)
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-8
```

```r
library(tsfeatures)
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(randomForest)
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:ggplot2':
##
##     margin
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
library(TSclust)
```

```
## Loading required package: pdc
## Loading required package: cluster
```

#Exercise 1)

##Task a)

```r
rom_electricity = read.csv('romanian_electricity.csv')
head(rom_electricity)
```

```
##              DateTime Consumption Production Nuclear Wind Hydroelectric
## 1 2019-01-01 00:00:00        6352       6527    1395   79          1383
## 2 2019-01-01 01:00:00        6116       5701    1393   96          1112
## 3 2019-01-01 02:00:00        5873       5676    1393  142          1030
## 4 2019-01-01 03:00:00        5682       5603    1397  191           972
## 5 2019-01-01 04:00:00        5557       5454    1393  159           960
## 6 2019-01-01 05:00:00        5525       5385    1395   91           958
##   Oil.and.Gas Coal Solar Biomass
## 1        1896 1744     0      30
## 2        1429 1641     0      30
## 3        1465 1616     0      30
## 4        1455 1558     0      30
## 5        1454 1458     0      30
## 6        1455 1456     0      30
```

```
# Checking for missing values in the dataset
missing_values <- sapply(rom_electricity, function(x) sum(is.na(x)))
missing_values
```

```
##      DateTime   Consumption    Production       Nuclear          Wind
##             0             0             0             0             0
## Hydroelectric   Oil.and.Gas          Coal         Solar       Biomass
##             0             0             0             0             0
```

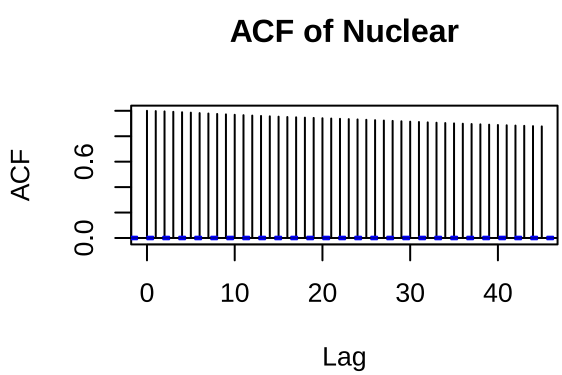
```
rom_electricity <- rom_electricity %>%
  mutate(Import_noNC = Consumption - Production + Nuclear)
```

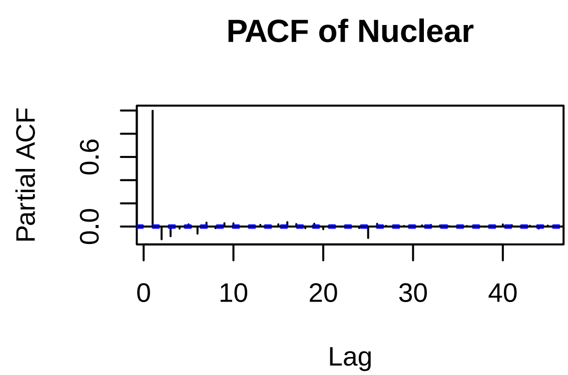
```
variables <- c("Nuclear", "Import_noNC")

par(mfrow=c(2, 2))

for (var in variables) {
  cat("Analysis for:", var, "\n")

  # KPSS Test for trend stationarity
  cat("KPSS Test for Trend Stationarity\n")
  print(kpss.test(rom_electricity[[var]]))

  # Checking for number of differences required
  cat("Number of non-seasonal differences (ndiffs) needed:")
  print(ndiffs(rom_electricity[[var]]))

  # Autocorrelation
  acf(rom_electricity[[var]], main=paste("ACF of", var))

  # Partial Autocorrelation
  pacf(rom_electricity[[var]], main=paste("PACF of", var))
}
```

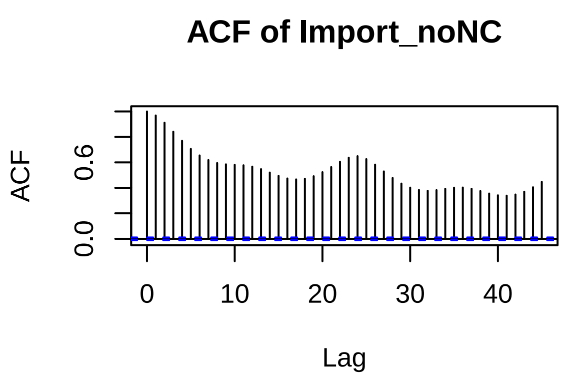
```
## Analysis for: Nuclear
## KPSS Test for Trend Stationarity
```

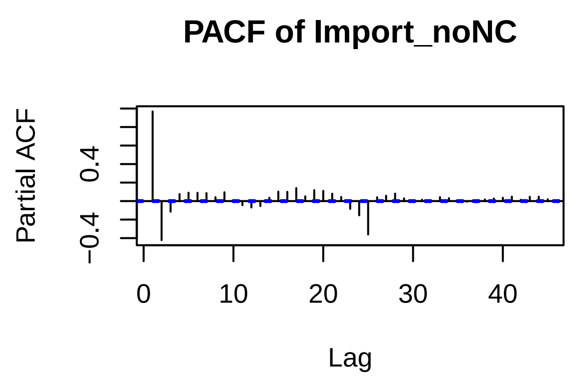
```
## Warning in kpss.test(rom_electricity[[var]]): p-value smaller than printed
## p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  rom_electricity[[var]]
## KPSS Level = 1.1688, Truncation lag parameter = 17, p-value = 0.01
##
## Number of non-seasonal differences (ndiffs) needed:[1] 1

## Analysis for: Import_noNC
## KPSS Test for Trend Stationarity

## Warning in kpss.test(rom_electricity[[var]]): p-value smaller than printed
## p-value

##
##  KPSS Test for Level Stationarity
##
## data:  rom_electricity[[var]]
## KPSS Level = 5.0478, Truncation lag parameter = 17, p-value = 0.01
##
## Number of non-seasonal differences (ndiffs) needed:[1] 1
```









KPSS-tests yielded a p-value of 0.01 for all variables. This means that the null hypothesis can be rejected, which suggests there's no trend stationarity present for these features. The ndiffs function returns 1 for both variables, indicating the at least 1 differencin term is needed to make the series stationary. The ACF plot for Nuclear shows significant autocorrelation that persists across many lags without a sharp cut-off, which would suggest non-stationarity. The ACF for Import_noNC gradually decays but does not drop off sharply, which can indicate a non-stationary series or a series with long-term dependence.

All off these factors indicates that both time series are non-stationary.

##Task b)

1. What would the 2 states represent?

The two states in an HMM applied to "Nuclear" might represent two distinct phases of nuclear power production. For instance, State 2 could represent a standard operation level, characterized by stable and predictable production levels. On the other hand, State 1 could be a reduced operation level where production levels are lower.

In the context of "Import_noNC", the two states could represent different levels of energy importation. State 1 might be periods with increased imports of energy importation, possibly due to increased demand or reduced domestic production. State 2 could be phases of low import, indicating lesser dependency on external energy sources.

2. Which types of stochastic processes are used to model the observable and the latent sequence, respectively?

The observable sequence is typically modeled by a probability distribution that depends on the current state. For example, if the data is continuous, a Gaussian distribution is a common choice.

The latent sequence is modeled using a Markov process. This is a stochastic process that satisfies the Markov property, meaning the next state depends only on the current state and not on the sequence of events that preceded it.

3. Are the model assumptions fulfilled?

Markov property: This assumption holds if the future state of the series depends only on the current state. For Nuclear and Import_noNC, if the processes are mainly governed by the current state of the system, without the influence of previous states, then this assumption is satisfied.

Output Independence: If the observable outputs at each time point depend solely on the current hidden state and are independent of the outputs at other times, the assumption of output independence is met.

Stationarity: The assumption is not met if we consider our results from previous tasks. However, the degree of non-stationarity might be acceptable for practical purposes.

Task c)

```r
# Fitting HMM to "Nuclear"
nuclear_hmm <- depmix(Nuclear ~ 1, family = gaussian(), data = rom_electricity,
                      nstates = 2)
nuclear_fit <- fit(nuclear_hmm)
```

```
## converged at iteration 16 with logLik: -184600.9
```

```r
# Model parameters for "Nuclear"
summary(nuclear_fit)
```

```
## Initial state probabilities model
## pr1 pr2
##   1   0
##
## Transition matrix
##         toS1  toS2
## fromS1 0.999 0.001
## fromS2 0.004 0.996
##
## Response parameters
## Resp 1 : gaussian
##     Re1.(Intercept) Re1.sd
## St1        1386.287 32.174
## St2         705.395 79.188
```

```r
posterior_nuc = posterior(nuclear_fit, type = 'viterbi')

# Fitting HMM to "Import_noNC"
import_noNC_hmm <- depmix(Import_noNC ~ 1, family = gaussian(),
                          data = rom_electricity, nstates = 2)
import_noNC_fit <- fit(import_noNC_hmm)
```

```
## converged at iteration 36 with logLik: -282712.7
```

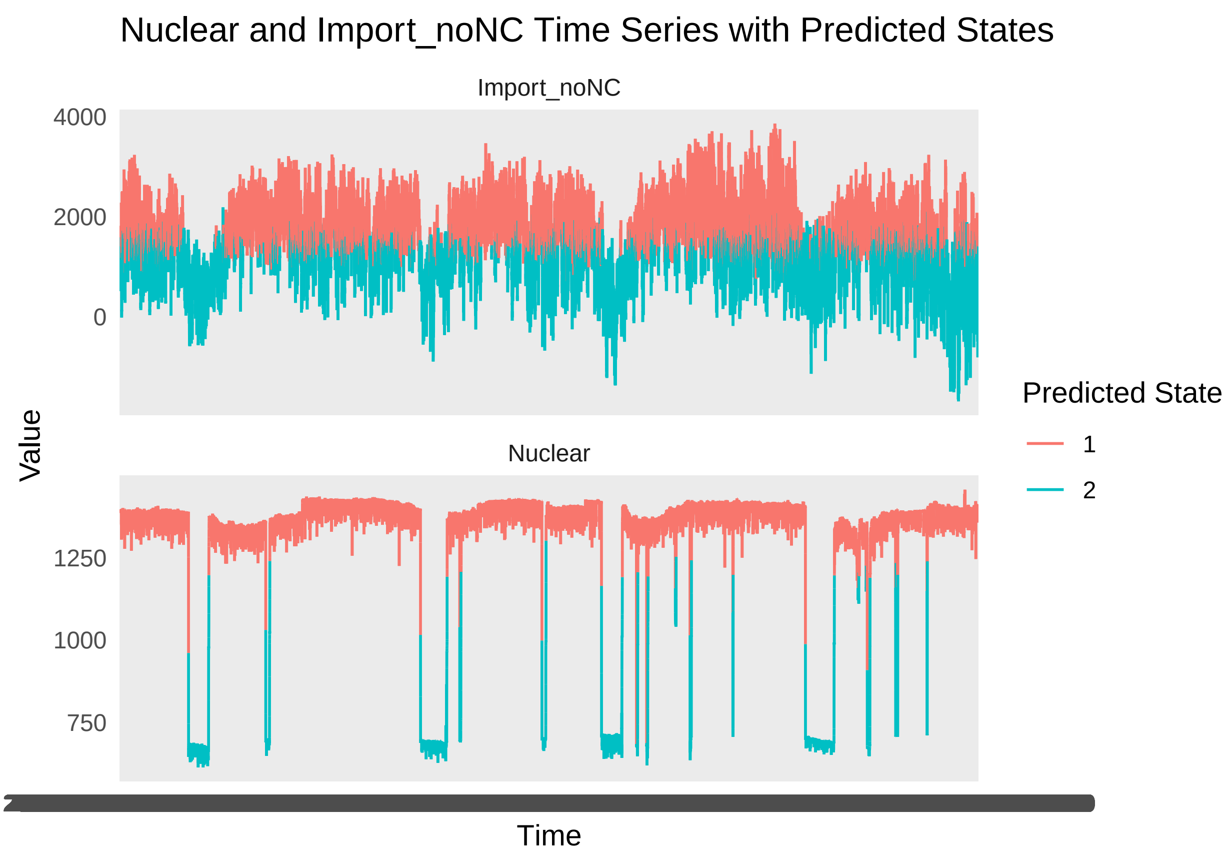
```r
# Model parameters for "Nuclear"
summary(import_noNC_fit)
```

```
## Initial state probabilities model
## pr1 pr2
##   1   0
##
## Transition matrix
##         toS1  toS2
## fromS1 0.971 0.029
## fromS2 0.037 0.963
##
## Response parameters
## Resp 1 : gaussian
##     Re1.(Intercept)  Re1.sd
## St1        2005.773 446.767
## St2         747.654 531.122
```

```r
posterior_noNC = posterior(import_noNC_fit, type = 'viterbi')

rom_electricity$nuclear_state <- posterior_nuc$state
rom_electricity$import_noNC_state <- posterior_noNC$state
```

```r
# Reshaping the data
long_data <- rom_electricity %>%
  gather(key = "variable", value = "value", Nuclear, Import_noNC) %>%
  mutate(state = ifelse(variable == "Nuclear", nuclear_state,
                        import_noNC_state))
```

```r
ggplot(long_data, aes(x = DateTime, y = value, color = factor(state),
                      group = variable)) +
  geom_line() +
  facet_wrap(~variable, scales = "free_y", nrow = 2) +
  labs(title = "Nuclear and Import_noNC Time Series with Predicted States",
       x = "Time",
       y = "Value",
       color = "Predicted State") +
  theme_minimal()
```



We can see from the plots that the two time series often is subject to changes in their predicted states at roughly similar time points. There seems to be a type of trigger which changes the predicted state for both time series. This can for instance be external triggers which determines how much energy importation and nuclear production is needed at certain times. A important point to mention here is that the variable import_noNC seems to be a little nit noisy, since the predicted states change frequently regardless of the current value.

Task d)

```r
rom_electricity$Import_noNC_smoothed <- zoo::rollmean(rom_electricity$Import_noNC,
                                                      k = 501, fill = NA,
                                                      align = "center")

#Re-fitting the HMM to the Smoothed Series
```

```r
import_noNC_smoothed_hmm <- depmix(Import_noNC_smoothed ~ 1,family = gaussian(),
                                   data = rom_electricity, nstates = 2)
import_noNC_smoothed_fit <- fit(import_noNC_smoothed_hmm)
```

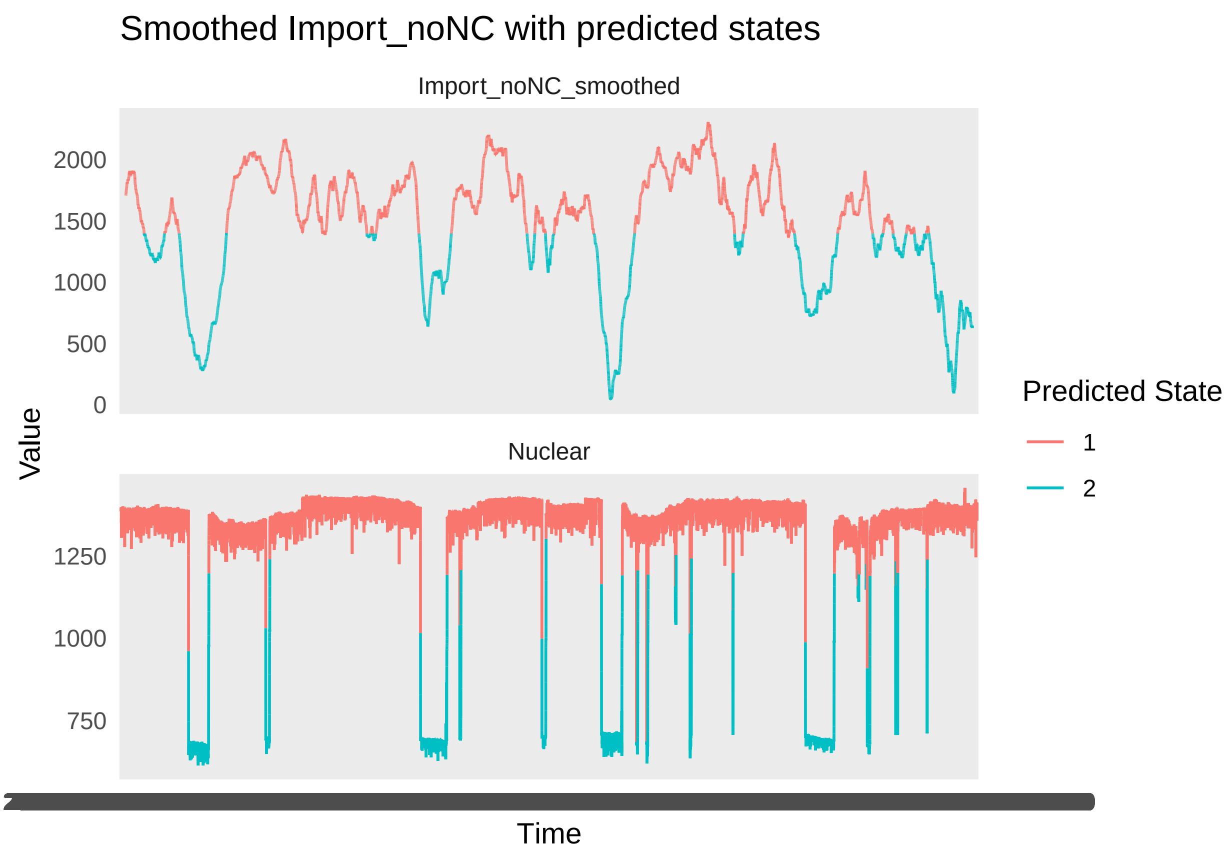
```
## converged at iteration 22 with logLik: -252217.1
```

```r
# Printing model parameters for smoothed "Import_noNC"
summary(import_noNC_smoothed_fit)
```

```
## Initial state probabilities model
## pr1 pr2
##   1   0
##
## Transition matrix
##         toS1  toS2
## fromS1 0.999 0.001
## fromS2 0.001 0.999
##
## Response parameters
## Resp 1 : gaussian
##     Re1.(Intercept)  Re1.sd
## St1        1741.981 210.812
## St2         941.307 352.695
```

```r
posterior_noNC_smoothed = posterior(import_noNC_smoothed_fit, type = 'viterbi')

rom_electricity$import_noNC_smoothed_state <- posterior_noNC_smoothed$state

#Results
long_data_smoothed <- rom_electricity %>%
  gather(key = "variable", value = "value", Nuclear, Import_noNC_smoothed) %>%
  mutate(state = ifelse(variable == "Nuclear", nuclear_state,
                        import_noNC_smoothed_state))

ggplot(long_data_smoothed, aes(x = DateTime, y = value, color = factor(state),
                               group = variable)) +
  geom_line() +
  facet_wrap(~variable, scales = "free_y", nrow = 2) +
  labs(title = "Smoothed Import_noNC with predicted states",
       x = "Time",
       y = "Value",
       color = "Predicted State") +
  theme_minimal()
```

```
## Warning: Removed 500 rows containing missing values (`geom_line()`).
```



(Also included Nuclear so that the plots from c) and d) look similar visually). In the plot from c), the changes in state are more frequent. This suggests that the HMM is detecting more frequent shifts between states due to the higher volatility of the unsmoothed data. In the plot from d), the color changes are less frequent, indicating that smoothing has led to a more stable prediction of states over time. The differences in state predictions before and after smoothing suggest that the model's perception of the system's dynamics can be affected by preprocessing steps. We can also notice that the colors of the are reversed for the two versions of import_noNC. This could be due to the changes in the data distribution that is caused by the smoothing.

Exercise 2

Task a

Load in the dataset and perform an exploratory data analysis. Convert the calendar week column to appropriate date object.

```
tipburn = read.csv("tipburn.csv")

# changing the date
df = data_frame(Date=as.Date(paste(tipburn$calendar_week, "1",sep ="_"),
                              format=("%Y_%U_%u")))
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## i Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
```

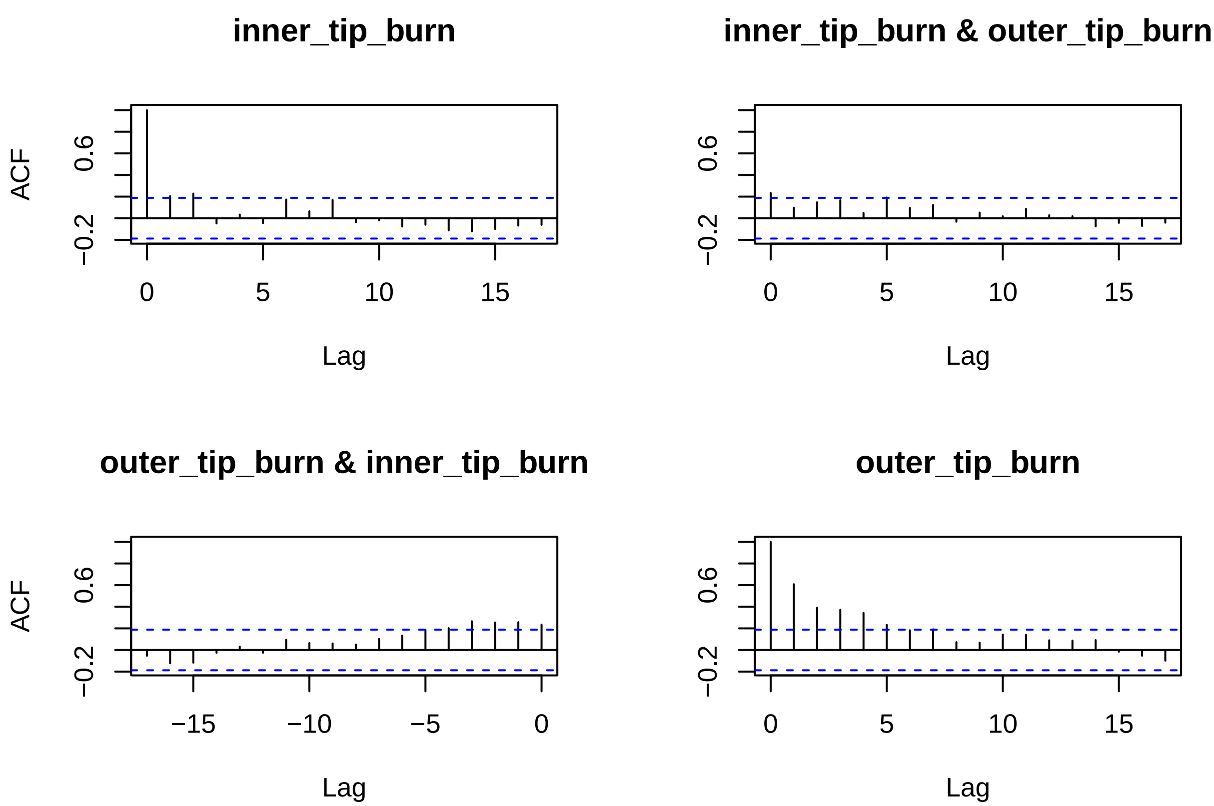
```
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.
```

```
tipburn = cbind(df, tipburn[,2-3])
head(tipburn, 3)
```

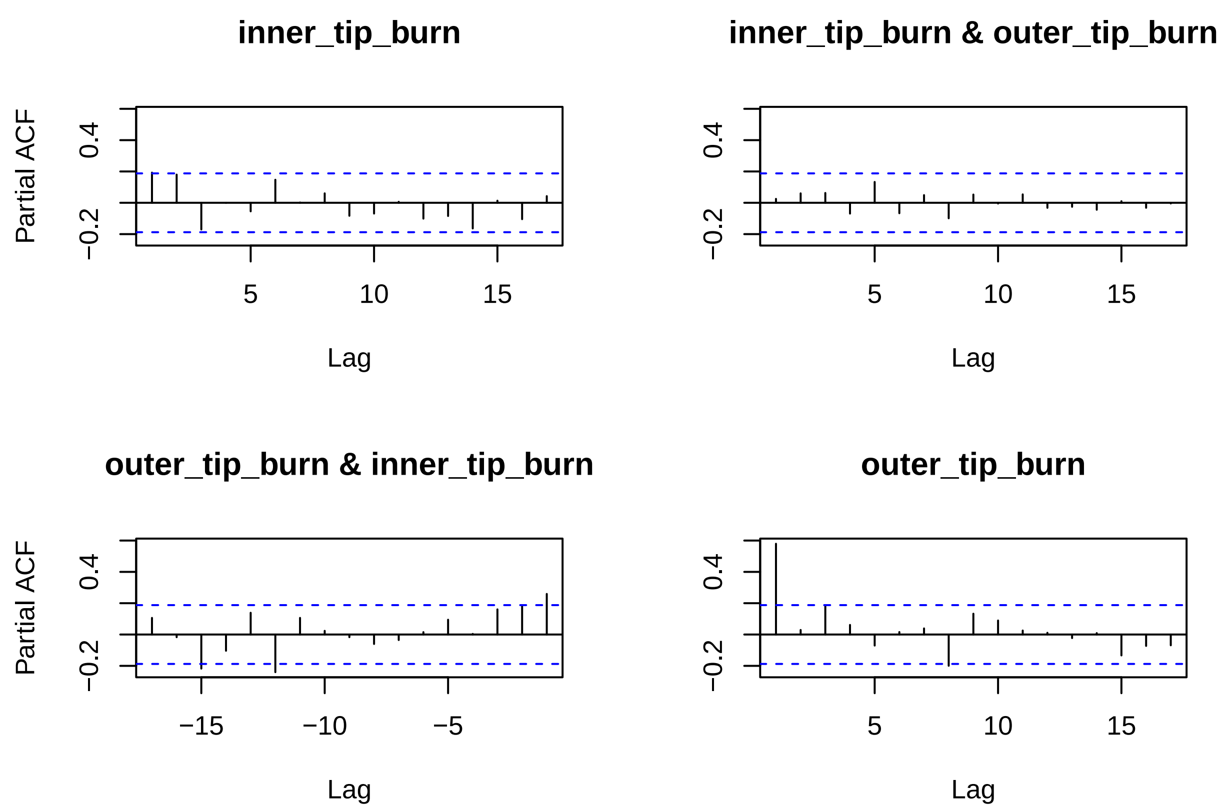
```
##         Date inner_tip_burn outer_tip_burn
## 1 2018-08-13            0.0              0
## 2 2018-08-20            0.0              0
## 3 2018-08-27            0.6              0
```

Exploratory data analysis.

```
acf(tipburn[,2:3])
```



```r
pacf(tipburn[,2:3])
```




```r
summary(tipburn)
```

```
##       Date            inner_tip_burn    outer_tip_burn
##  Min.   :2018-08-13   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:2019-02-25   1st Qu.:0.00000   1st Qu.:0.00000
##  Median :2019-10-28   Median :0.00000   Median :0.06667
##  Mean   :2019-10-16   Mean   :0.05901   Mean   :0.17164
##  3rd Qu.:2020-05-18   3rd Qu.:0.07500   3rd Qu.:0.28000
##  Max.   :2021-01-18   Max.   :0.60000   Max.   :0.86667
```

```r
# Add 'group_inner' column
tipburn <- tipburn %>%
  mutate(group_inner = ifelse(tipburn$inner_tip_burn == 0, "None",
  ifelse(tipburn$inner_tip_burn > 0 & tipburn$inner_tip_burn <= 0.25, "Weak", "Strong")))

# Add 'group_outer' column
tipburn <- tipburn %>%
  mutate(group_outer = ifelse(tipburn$outer_tip_burn == 0, "None",
  ifelse(tipburn$outer_tip_burn > 0 & tipburn$outer_tip_burn <= 0.25, "Weak", "Strong")))

head(tipburn, 5)
```

```
##          Date inner_tip_burn outer_tip_burn group_inner group_outer
## 1 2018-08-13      0.0000000     0.00000000        None        None
## 2 2018-08-20      0.0000000     0.00000000        None        None
## 3 2018-08-27      0.6000000     0.00000000      Strong        None
## 4 2018-09-03      0.2333333     0.10000000        Weak        Weak
## 5 2018-09-10      0.1666667     0.03333333        Weak        Weak
```

Task b)

Experts assume that tip burn is caused by some unobserved climate conditions. Explain why it makes sense to apply a discrete Hidden Markov Model with two latent states, which can be interpreted as "risk" and "non-risk" climate conditions.

The term "hidden" in Hidden Markov Model reflects the idea that the underlying states (in this case, climate conditions) are not directly observed but can be inferred from the observable outcomes (severity of tip burn).

Climate conditions can vary over time and exhibit different patterns or regimes. For example, there may be periods of favorable conditions ("non-risk") and periods with conditions that contribute to tip burn ("risk"). The temporal aspect of HMMs allows for modeling the transitions between different climate states over time.

The HMM's ability to model transitions between latent states is crucial. In the context of tip burn, it's plausible that plants may experience shifts in climate conditions that influence the likelihood and severity of tip burn.

The two latent states, "risk" and "non-risk," provide a simplified representation of the underlying dynamics.

The three levels of the observable variable are: "none", "weak" and "strong" tip burn. You may treat inner and outer tip-burn as two separate models first. Which parameters would you suggest to use for initialization ($\pi 0$, A and B)? Which dimensions do these parameters have for each of the tip-burn types?

$\pi 0$**(Initial State Probabilities)**: It represents the probabilities of starting in each latent state. Since we have two latent states ("risk" and "non-risk"), you can initialize $\pi 0$ as a vector with two elements, representing the initial probability of being in each state.

Since we don't have prior knowledge about the initial state probabilities, we may choose to initialize them uniformly. This means setting $\pi 0$ to indicating an equal probability of starting in either the "risk" or "non-risk" state.

$$\pi 0 = [0.5, 0.5]$$

A (State Transition Probability Matrix): It represents the probabilities of transitioning from one latent state to another. Again, since we have two latent states, A is a 2x2 matrix. The element A[i, j] represents the probability of transitioning from state i to state j.

For the state transition matrix A, consider the expected persistence or stability of climate conditions. If you expect climate conditions to remain relatively stable over time then we may choose to initialize them uniformly.

$$A = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$

B (Emission Probability Matrix): It represents the probabilities of observing each level of the observable variable given the latent state. As we have three levels ("none," "weak," and "strong"), B is a 2x3 matrix for each tip-burn type (inner and outer). The element B[i, k] represents the probability of observing level k given the latent state i.

For the emission probability matrix B, you need to consider the probabilities of observing each level of the observable variable given the latent state. Since we don't have prior knowledge, we might start with a shifted uniform distribution.

$$B = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.3 & 0.3 & 0.4 \end{bmatrix}$$

Task c)

```
# Model for group_inner
tipburn$group_inner <- as.factor(tipburn$group_inner)

set.seed(4)
hmm_model_inner <- depmix(group_inner ~ 1, nstates = 2,
                          family = multinomial(), data = tipburn)
hmm_model_inner <- depmixS4::fit(hmm_model_inner)
```

```
## converged at iteration 251 with logLik: -85.04021
```

```
summary(hmm_model_inner)
```

```
## Initial state probabilities model
## pr1 pr2
##   1   0
##
## Transition matrix
##         toS1  toS2
## fromS1 0.748 0.252
## fromS2 0.334 0.666
##
## Response parameters
## Resp 1 : multinomial
##     Re1.(Intercept).None Re1.(Intercept).Strong Re1.(Intercept).Weak
## St1                    0                 -3.831               -2.577
## St2                    0                  5.425                7.853
```

```
# Posterior states
posterior_inner <- posterior(hmm_model_inner, type="viterbi")

# latent state
ls_inner <- viterbi(hmm_model_inner)$state
```

```
# Model for group_outer
tipburn$group_outer <- as.factor(tipburn$group_outer)

set.seed(4)
hmm_model_outer <- depmix(group_outer ~ 1, nstates = 2,
                          family = multinomial(), data = tipburn)
hmm_model_outer <- depmixS4::fit(hmm_model_outer)
```

```
## converged at iteration 63 with logLik: -102.7431
```

```r
summary(hmm_model_outer)
```

```
## Initial state probabilities model
## pr1 pr2
##   0   1
##
## Transition matrix
##         toS1  toS2
## fromS1 0.890 0.110
## fromS2 0.087 0.913
##
## Response parameters
## Resp 1 : multinomial
##     Re1.(Intercept).None Re1.(Intercept).Strong Re1.(Intercept).Weak
## St1                    0                  3.021                2.287
## St2                    0                 -3.120               -0.103
```

```r
# Posterior states
posterior_inner <- posterior(hmm_model_outer, type="viterbi")

# latent state
ls_outer <- viterbi(hmm_model_outer)$state
```

```r
# Model for both group_inner and group_outer simultaneously
hmm_model_both <- depmix(list(group_inner~ 1, group_outer~ 1), nstates = 2,
                     family = list(multinomial(),multinomial()), data = tipburn)
hmm_model_both <- depmixS4::fit(hmm_model_both)
```

```
## converged at iteration 86 with logLik: -189.7092
```

```r
summary(hmm_model_both)
```

```
## Initial state probabilities model
## pr1 pr2
##   1   0
##
## Transition matrix
##         toS1  toS2
## fromS1 0.900 0.100
## fromS2 0.092 0.908
##
## Response parameters
## Resp 1 : multinomial
## Resp 2 : multinomial
##     Re1.(Intercept).None Re1.(Intercept).Strong Re1.(Intercept).Weak
## St1                    0                 -4.057               -0.864
## St2                    0                 -1.393                0.556
##     Re2.(Intercept).None Re2.(Intercept).Strong Re2.(Intercept).Weak
## St1                    0                 -4.247               -0.227
## St2                    0                  2.126                1.680
```

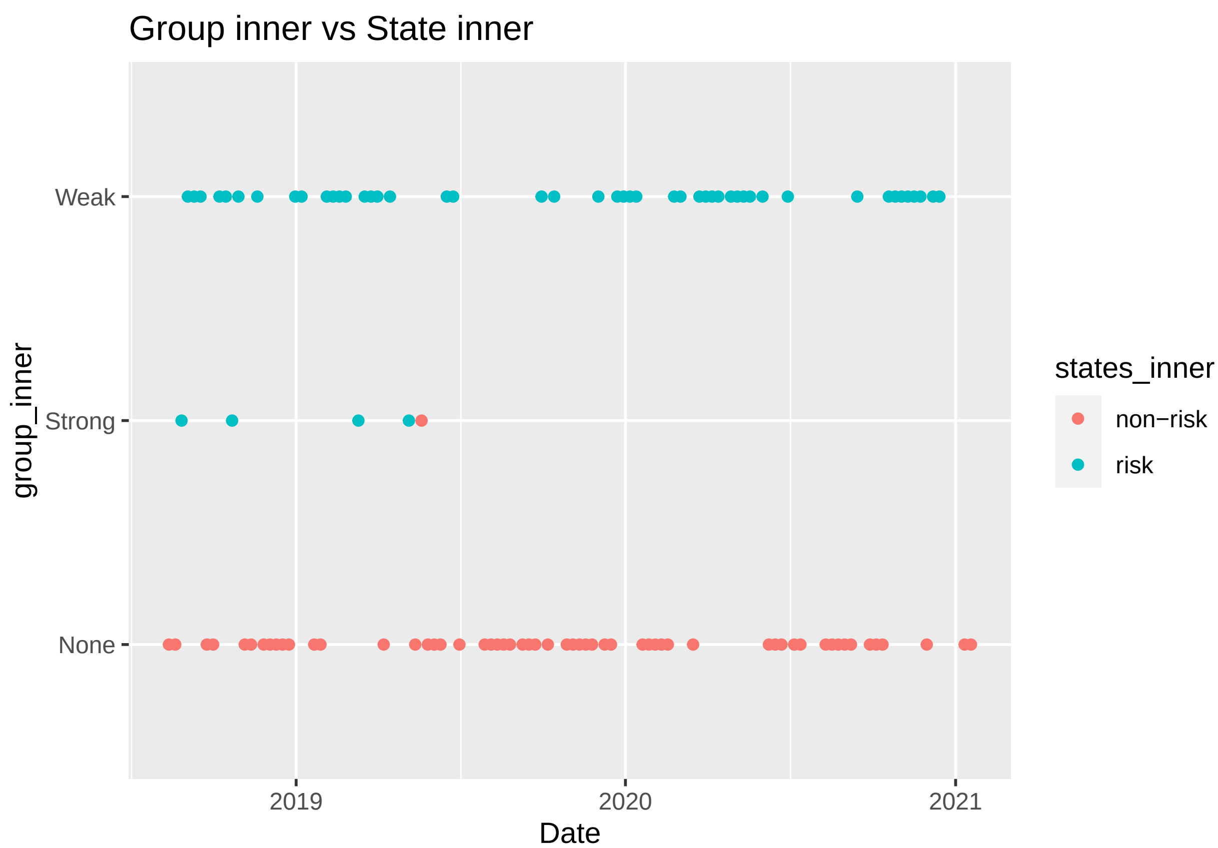
```r
# Posterior states
posterior_inner <- posterior(hmm_model_both, type="viterbi")

# latent sequence
#ls_both <- ifelse(ls_inner == 1, "Risk", "Non-Risk")
ls_both <- viterbi(hmm_model_both)$state
```

Task d)

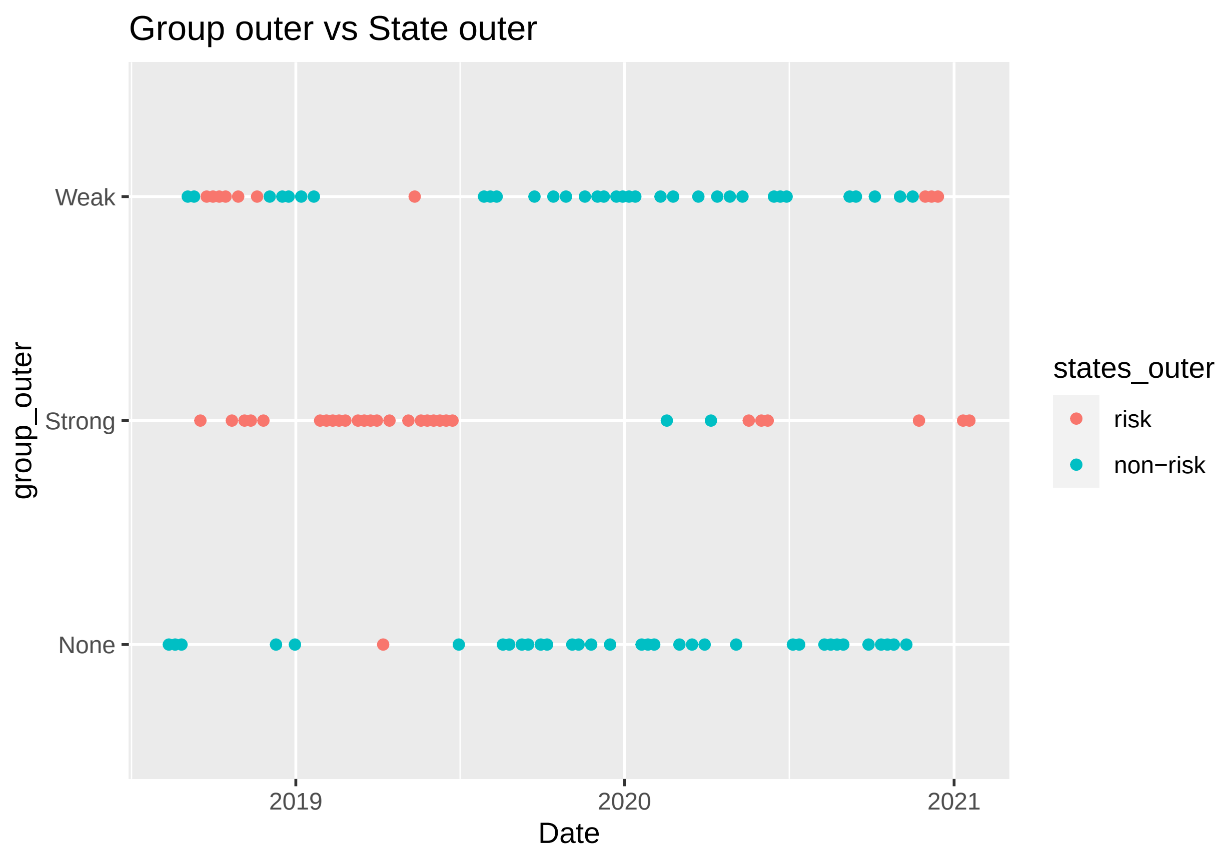
```r
# three new columns states_inner, states_outer and states_joint.
tipburn$states_inner <- ls_inner
tipburn$states_outer <- ls_outer
tipburn$states_both <- ls_both
head(tipburn,3)
```

```
##         Date inner_tip_burn outer_tip_burn group_inner group_outer states_inner
## 1 2018-08-13            0.0              0        None        None            1
## 2 2018-08-20            0.0              0        None        None            1
## 3 2018-08-27            0.6              0      Strong        None            2
##   states_outer states_both
## 1            2           1
## 2            2           1
## 3            2           1
```

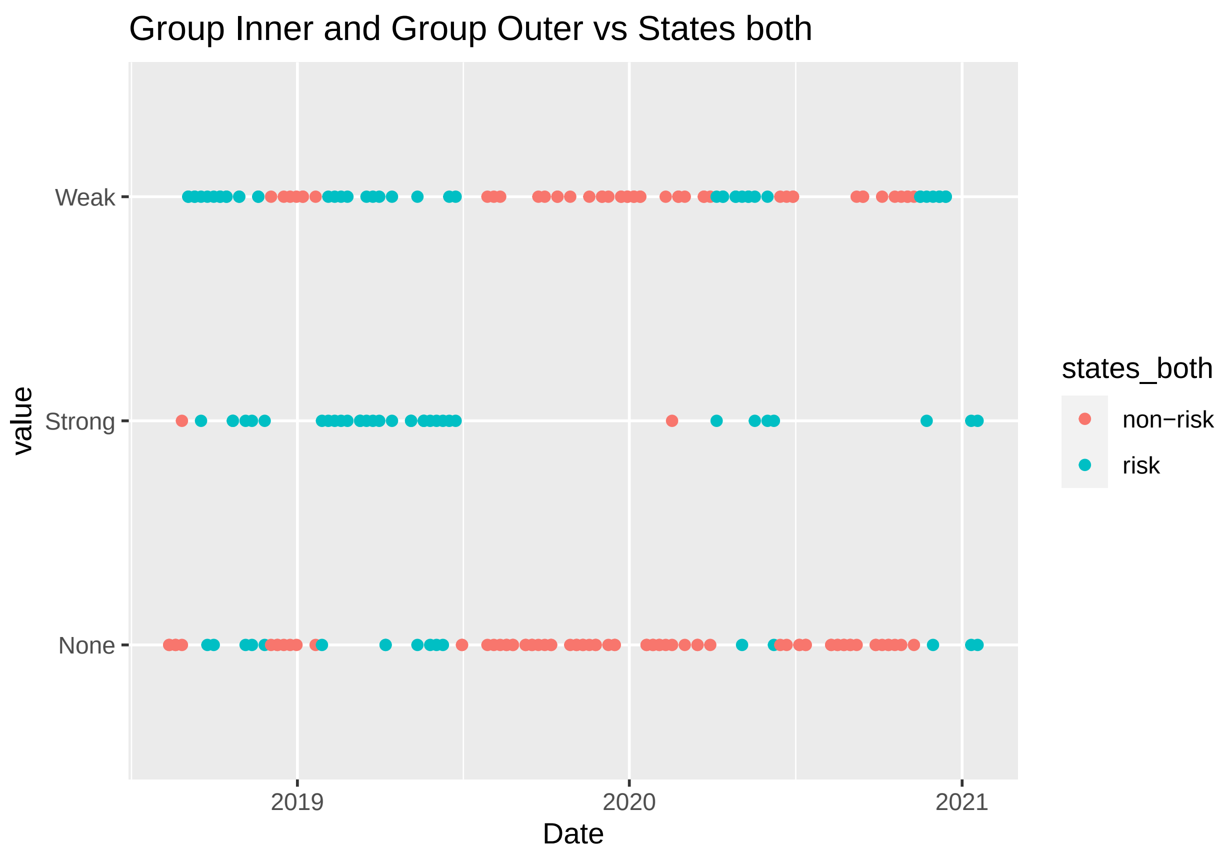
```r
# group inner vs state inner
tipburn$states_inner <- as.factor(tipburn$states_inner)
ggplot(tipburn, aes(Date, group_inner, col= states_inner)) +
  scale_color_discrete(labels = c("non-risk","risk")) +
  geom_point() + ggtitle("Group inner vs State inner")
```




```r
# group outer vs state outer
tipburn$states_outer <- as.factor(tipburn$states_outer)
ggplot(tipburn, aes(Date, group_outer, col= states_outer)) +
  scale_color_discrete(labels = c("risk", "non-risk")) +
  geom_point() + ggtitle("Group outer vs State outer")
```




```r
# group inner and group outer vs state both
tipburn$states_both <- as.factor(tipburn$states_both)
tipburn_long <- gather(tipburn, key = "group_type", value = "value", group_inner, group_outer)

ggplot(tipburn_long, aes(x = Date, y = value, col = states_both, linetype = group_type)) +
  ggtitle("Group Inner and Group Outer vs States both") +
  scale_color_discrete(labels = c("non-risk","risk")) +
  geom_point()
```



Group inner vs State inner:

We can assume that the risk is mostly on the weak and some in strong, nothing in the None.

Group outer vs State outer:

We can assume that the risk is more in the strong part, while the non-risk is mainly in the weak- and none region.

Group Inner and Group Outer vs States both:

The risk and non-risk is in all the three values in the y-axis, for the weak we have slightly more non-risk that risk. In the case for strong we have almost only risk. None has a majority of non-risk.

```
# Function to compute pairwise accuracy
compute_pairwise_accuracy <- function(seq1, seq2) {
  if (length(seq1) != length(seq2)) {
    stop("Sequences must have the same length.")
  }
  matching_states <- sum(seq1 == seq2)
  accuracy <- matching_states / length(seq1)
  return(accuracy)
}

# Compute pairwise accuracy
accuracy_inner_outer <- compute_pairwise_accuracy(ls_inner, ls_outer)
accuracy_inner_both <- compute_pairwise_accuracy(ls_inner, ls_both)
accuracy_outer_both <- compute_pairwise_accuracy(ls_outer, ls_both)
```

```r
# Display the results
cat("Pairwise Accuracy (Inner vs. Outer):", accuracy_inner_outer, "\n")
```

```
## Pairwise Accuracy (Inner vs. Outer): 0.4036697
```

```r
cat("Pairwise Accuracy (Inner vs. Inner and Outer):", accuracy_inner_both, "\n")
```

```
## Pairwise Accuracy (Inner vs. Inner and Outer): 0.6697248
```

```r
cat("Pairwise Accuracy (Outer vs. Inner and Outer):", accuracy_outer_both, "\n")
```

```
## Pairwise Accuracy (Outer vs. Inner and Outer): 0.0733945
```

Exercise 3

Task a)

```r
# Loading the dataset
pedestrian_data <- read.csv("pedestrian.csv")
```

```r
# Exploratory Data Analysis
print(head(pedestrian_data, 3))
```

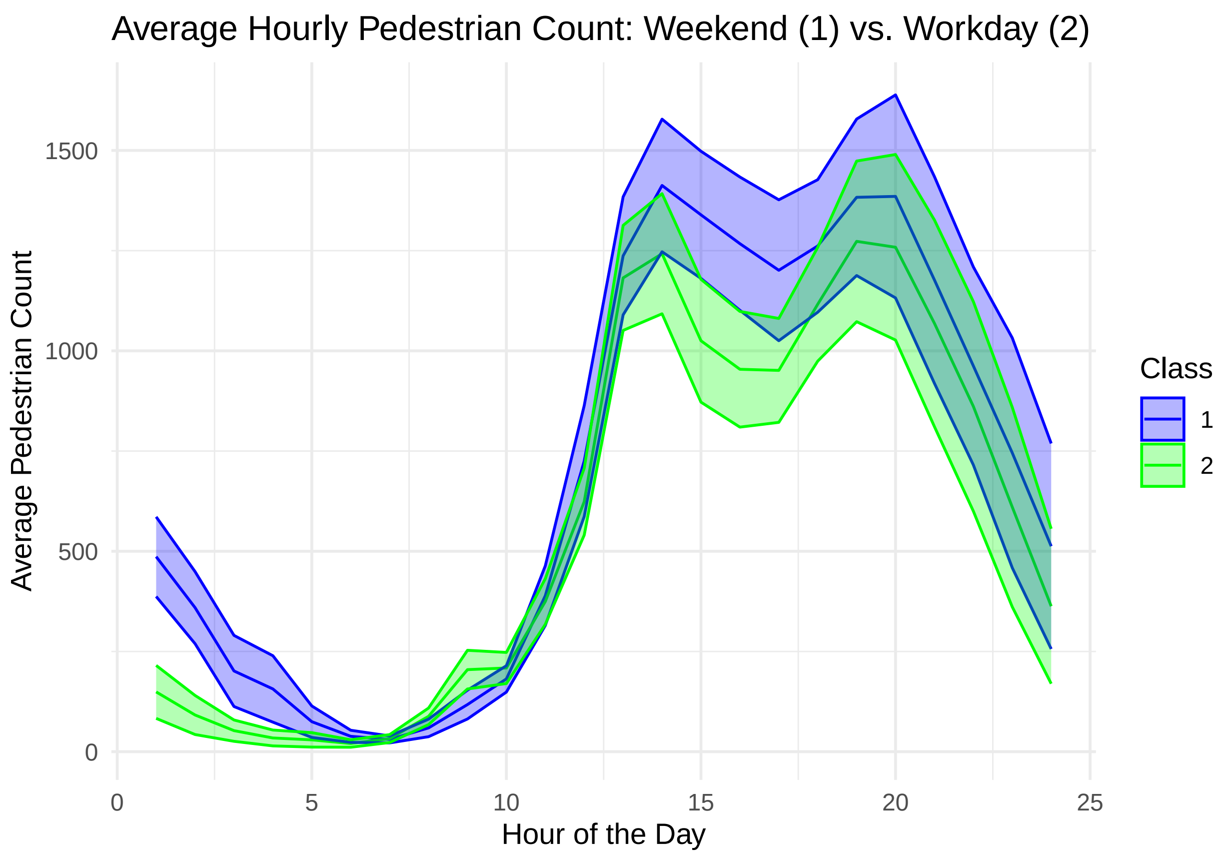
```
##    H1  H2  H3  H4  H5  H6 H7  H8  H9 H10 H11 H12  H13  H14  H15  H16  H17  H18
## 1 501 328 195 218  67  17 28  72 132 215 406 765 1207 1427 1234 1238 1107 1190
## 2 880 752 913 863 402 112 60 112 119 186 365 596  990 1193 1040 1063 1009 1025
## 3 493 389 174 121  82  36 27  64 127 203 415 747 1164 1414 1520 1295 1265 1430
##    H19  H20  H21  H22  H23 H24 target
## 1 1255 1144  905  690  386 192      1
## 2 1089  979  706  585  356 187      1
## 3 1637 1697 1456 1319 1179 848      1
```

```r
summary(pedestrian_data)
```

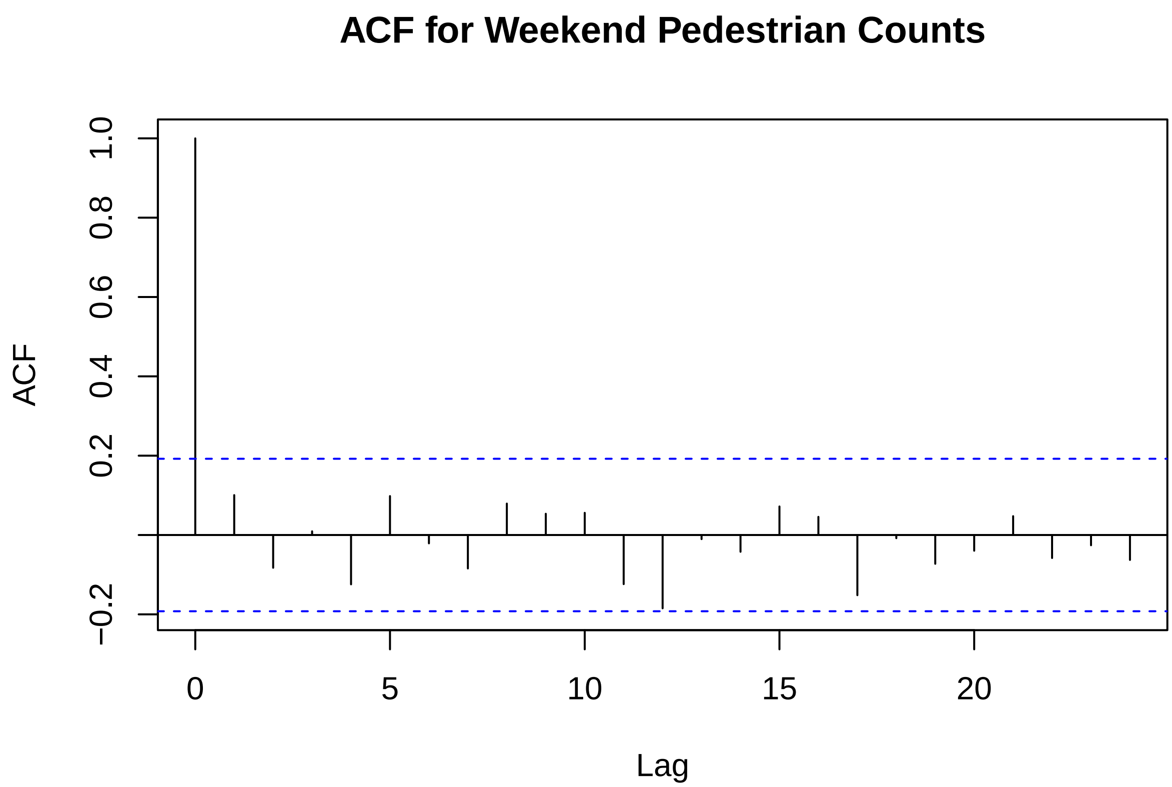
```
##        H1               H2               H3                H4
##  Min.   : 51.0   Min.   : 24.0   Min.   : 11.00   Min.   :  6.00
##  1st Qu.:115.0   1st Qu.: 69.0   1st Qu.: 41.00   1st Qu.: 24.00
##  Median :167.0   Median : 99.0   Median : 59.00   Median : 40.00
##  Mean   :245.8   Mean   :168.2   Mean   : 95.12   Mean   : 69.21
##  3rd Qu.:391.0   3rd Qu.:287.5   3rd Qu.:138.00   3rd Qu.:102.00
##  Max.   :880.0   Max.   :813.0   Max.   :913.00   Max.   :863.00
##        H5                H6                H7               H8
##  Min.   :  3.00   Min.   :  3.00   Min.   : 9.00   Min.   : 22.0
##  1st Qu.: 20.00   1st Qu.: 16.00   1st Qu.:26.00   1st Qu.: 64.0
##  Median : 34.00   Median : 22.00   Median :31.00   Median : 80.0
##  Mean   : 42.45   Mean   : 25.54   Mean   :32.08   Mean   : 79.9
##  3rd Qu.: 57.50   3rd Qu.: 31.00   3rd Qu.:37.00   3rd Qu.: 95.0
##  Max.   :402.00   Max.   :112.00   Max.   :86.00   Max.   :185.0
```

```
##        H9              H10            H11             H12              H13
##  Min.   : 48.0   Min.   :104   Min.   :113.0   Min.   : 292.0   Min.   : 850
##  1st Qu.:141.0   1st Qu.:177   1st Qu.:343.0   1st Qu.: 589.5   1st Qu.:1110
##  Median :179.0   Median :199   Median :375.0   Median : 642.0   Median :1195
##  Mean   :179.7   Mean   :201   Mean   :378.8   Mean   : 652.8   Mean   :1198
##  3rd Qu.:217.0   3rd Qu.:222   3rd Qu.:407.5   3rd Qu.: 703.0   3rd Qu.:1280
##  Max.   :485.0   Max.   :510   Max.   :815.0   Max.   :1611.0   Max.   :1823
##       H14             H15             H16             H17              H18
##  Min.   : 883   Min.   : 691   Min.   : 605   Min.   : 623.0   Min.   : 827
##  1st Qu.:1168   1st Qu.: 957   1st Qu.: 889   1st Qu.: 891.5   1st Qu.:1032
##  Median :1282   Median :1081   Median :1000   Median : 997.0   Median :1136
##  Mean   :1291   Mean   :1115   Mean   :1044   Mean   :1022.8   Mean   :1158
##  3rd Qu.:1394   3rd Qu.:1249   3rd Qu.:1198   3rd Qu.:1125.0   3rd Qu.:1279
##  Max.   :2052   Max.   :2014   Max.   :1969   Max.   :1733.0   Max.   :1623
##       H19             H20             H21              H22
##  Min.   : 682   Min.   : 776   Min.   : 567.0   Min.   : 385.0
##  1st Qu.:1153   1st Qu.:1118   1st Qu.: 896.5   1st Qu.: 682.0
##  Median :1260   Median :1227   Median :1047.0   Median : 827.0
##  Mean   :1305   Mean   :1295   Mean   :1099.6   Mean   : 890.3
##  3rd Qu.:1458   3rd Qu.:1492   3rd Qu.:1296.5   3rd Qu.:1093.0
##  Max.   :1891   Max.   :1927   Max.   :1847.0   Max.   :1731.0
##       H23              H24             target
##  Min.   : 253.0   Min.   : 134.0   Min.   :1.000
##  1st Qu.: 425.5   1st Qu.: 235.5   1st Qu.:1.000
##  Median : 565.0   Median : 309.0   Median :2.000
##  Mean   : 648.9   Mean   : 405.7   Mean   :1.713
##  3rd Qu.: 891.5   3rd Qu.: 600.5   3rd Qu.:2.000
##  Max.   :1375.0   Max.   :1188.0   Max.   :2.000
```

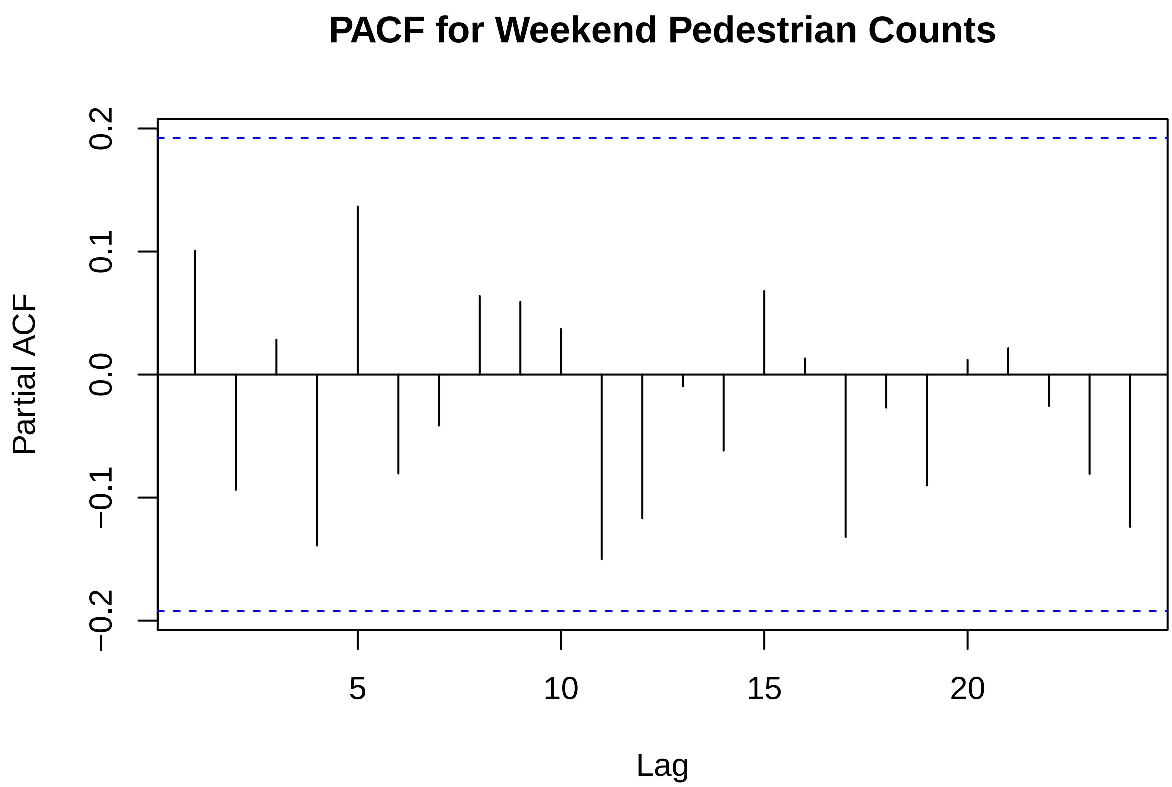
```r
# Create a separate hour column
pedestrian_data_long <- pivot_longer(pedestrian_data, cols = starts_with("H"),
                                     names_to = "hour", values_to = "count")
pedestrian_data_long$hour <- as.numeric(sub("H", "", pedestrian_data_long$hour))

# Calculate mean and standard deviation
agg_data <- pedestrian_data_long %>%
  group_by(target, hour) %>%
  summarise(mean = mean(count), std = sd(count), .groups = 'drop')

# Plotting
ggplot(agg_data, aes(x = hour, y = mean, group = target,
                     color = factor(target))) +
  geom_line() +
  geom_ribbon(aes(ymin = mean - std, ymax = mean + std,
                  fill = factor(target)), alpha = 0.3) +
  scale_color_manual(values = c("blue", "green")) +
  scale_fill_manual(values = c("blue", "green")) +
  labs(x = "Hour of the Day", y = "Average Pedestrian Count",
       title = "Average Hourly Pedestrian Count: Weekend (1) vs. Workday (2)",
       color = "Class", fill = "Class") +
  theme_minimal()
```



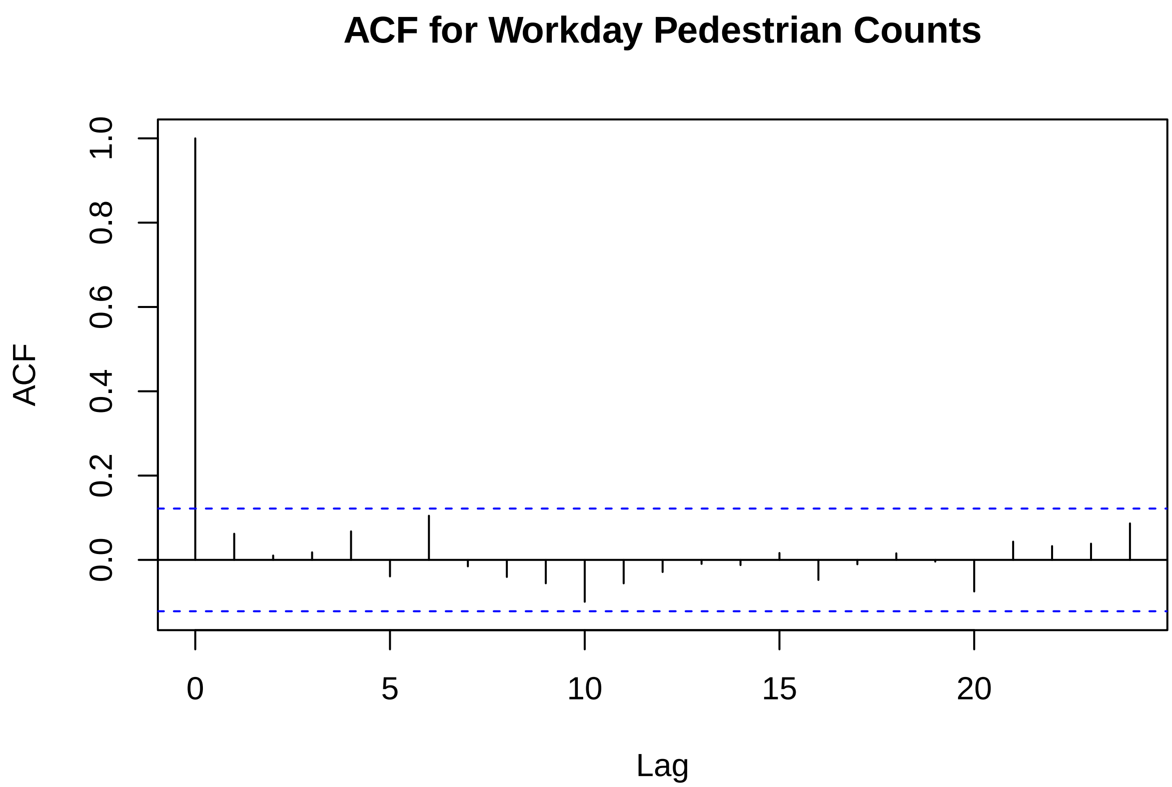

```r
# Separating the dataset into weekend and workday
weekend_data <- pedestrian_data[pedestrian_data$target == 1, 1:24]
workday_data <- pedestrian_data[pedestrian_data$target == 2, 1:24]

# Aggregate data by summing across all hours for each day
daily_counts_weekend <- rowSums(weekend_data)
daily_counts_workday <- rowSums(workday_data)

# Convert to time series objects
ts_weekend <- ts(daily_counts_weekend)
ts_workday <- ts(daily_counts_workday)

# ACF for weekend
acf(ts_weekend, main="ACF for Weekend Pedestrian Counts", lag.max = 24)
```



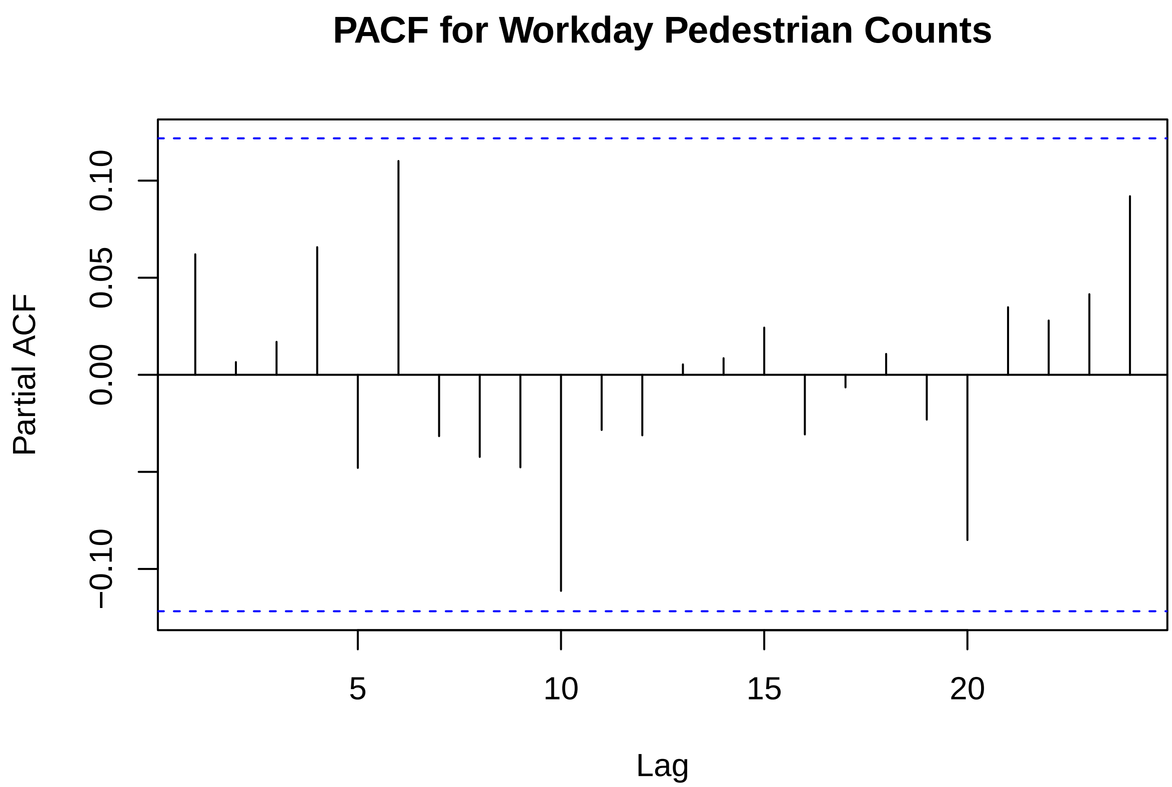

```
# PACF for weekend
pacf(ts_weekend, main="PACF for Weekend Pedestrian Counts", lag.max = 24)
```




```r
# ACF for workday
acf(ts_workday, main="ACF for Workday Pedestrian Counts", lag.max = 24)
```




```r
# PACF for workday
pacf(ts_workday, main="PACF for Workday Pedestrian Counts", lag.max = 24)
```



There are several differences between the two classes:

Peak Times:

For workdays, there's a noticeable peak during typical office hours, particularly in the early evening.

Morning Activity:

On workdays, the pedestrian count starts increasing earlier in the morning, likely due to people commuting to work, while on weekends, the increase in pedestrian traffic starts later.

Evening Activity:

The pedestrian count remains higher in the evening hours on weekends compared to workdays, suggesting more recreational or social activities during weekend evenings.

Variability:

The standard deviation (indicated by the shaded area) is generally higher on weekends, especially during the evening hours, indicating more variability in pedestrian counts.

Task b)

```
# Split the samples into a 70%-30% train-test split
set.seed(123)

index <- createDataPartition(y=pedestrian_data$target, p = 0.7, list = FALSE)
train_data <- pedestrian_data[index,]
```

```r
test_data <- pedestrian_data[-index,]

# fjerne konstant kolloner
train_features <- tsfeatures(ts(t(unname(as.matrix(train_data[,1:24])))))
test_features <- tsfeatures(ts(t(unname(as.matrix(test_data[,1:24])))))

head(train_features, 3)
```

```
## # A tibble: 3 x 16
##   frequency nperiods seasonal_period trend     spike linearity curvature e_acf1
##       <dbl>    <dbl>           <dbl> <dbl>     <dbl>     <dbl>     <dbl>  <dbl>
## 1         1        0               1 0.936 0.0000198     2.17     -1.61   0.613
## 2         1        0               1 0.873 0.0000431     0.399    -0.358  0.533
## 3         1        0               1 0.934 0.0000206     2.04     -2.03   0.642
## # i 8 more variables: e_acf10 <dbl>, entropy <dbl>, x_acf1 <dbl>,
## #   x_acf10 <dbl>, diff1_acf1 <dbl>, diff1_acf10 <dbl>, diff2_acf1 <dbl>,
## #   diff2_acf10 <dbl>
```

```r
head(test_features, 3)
```

```
## # A tibble: 3 x 16
##   frequency nperiods seasonal_period trend      spike linearity curvature e_acf1
##       <dbl>    <dbl>           <dbl> <dbl>      <dbl>     <dbl>     <dbl>  <dbl>
## 1         1        0               1 0.954 0.00000540      3.49    -0.348  0.596
## 2         1        0               1 0.939 0.0000119       3.34    -0.338  0.525
## 3         1        0               1 0.950 0.0000109       3.07    -0.903  0.604
## # i 8 more variables: e_acf10 <dbl>, entropy <dbl>, x_acf1 <dbl>,
## #   x_acf10 <dbl>, diff1_acf1 <dbl>, diff1_acf10 <dbl>, diff2_acf1 <dbl>,
## #   diff2_acf10 <dbl>
```

```r
# Logistic Regression
mod_logreg <- caret::train(
  x = train_features,
  y = as.factor(train_data$target),
  method = "glmnet",
  family = "binomial",
  trControl = trainControl(method = "cv"),
  tuneLength = 3
)
```

```
## Warning: Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
```

```
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
```

```
# Make predictions on the test set
logistic_pred <- predict(mod_logreg, newdata = test_features)

# confusion matrix
logistic_metrics <- confusionMatrix(logistic_pred, as.factor(test_data$target))

summary(mod_logreg)
```

```
##              Length Class      Mode
## a0            100   -none-     numeric
## beta         1600   dgCMatrix  S4
## df            100   -none-     numeric
## dim             2   -none-     numeric
## lambda        100   -none-     numeric
## dev.ratio     100   -none-     numeric
## nulldev         1   -none-     numeric
## npasses         1   -none-     numeric
## jerr            1   -none-     numeric
## offset          1   -none-     logical
## classnames      2   -none-     character
## call            5   -none-     call
## nobs            1   -none-     numeric
## lambdaOpt       1   -none-     numeric
## xNames         16   -none-     character
## problemType     1   -none-     character
## tuneValue       2   data.frame list
## obsLevels       2   -none-     character
## param           1   -none-     list
```

```
print(logistic_metrics)
```

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

```
##           Reference
## Prediction  1  2
##          1 26  1
##          2  2 79
##
##                Accuracy : 0.9722
##                  95% CI : (0.921, 0.9942)
##     No Information Rate : 0.7407
##     P-Value [Acc > NIR] : 7.974e-11
##
##                   Kappa : 0.9268
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9286
##             Specificity : 0.9875
##          Pos Pred Value : 0.9630
##          Neg Pred Value : 0.9753
##              Prevalence : 0.2593
##          Detection Rate : 0.2407
##    Detection Prevalence : 0.2500
##       Balanced Accuracy : 0.9580
##
##        'Positive' Class : 1
##
```

```
# Train a classifier (Random Forest)
mod_rf <- caret::train(
  x = train_features,
  y = as.factor(train_data$target),
  method = "rf",
  family = "binomial",
  trControl = trainControl(method = "cv"),
  tuneLength = 3)
```

```
## Warning: Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
```

```
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
```

```
# Make predictions on the test set
rf_pred <- predict(mod_rf, newdata = test_features)

# confusion matrix
rf_metrics <- confusionMatrix(rf_pred, as.factor(test_data$target))

summary(mod_rf)
```

```
##                 Length Class      Mode
## call               5   -none-     call
## type               1   -none-     character
## predicted        255   factor     numeric
## err.rate        1500   -none-     numeric
## confusion          6   -none-     numeric
## votes            510   matrix     numeric
## oob.times        255   -none-     numeric
## classes            2   -none-     character
## importance        16   -none-     numeric
## importanceSD       0   -none-     NULL
## localImportance    0   -none-     NULL
## proximity          0   -none-     NULL
## ntree              1   -none-     numeric
## mtry               1   -none-     numeric
## forest            14   -none-     list
## y                255   factor     numeric
## test               0   -none-     NULL
## inbag              0   -none-     NULL
## xNames            16   -none-     character
## problemType        1   -none-     character
## tuneValue          1   data.frame list
## obsLevels          2   -none-     character
## param              1   -none-     list
```

```
print(rf_metrics)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 26  2
```

```
##          2  2 78
##
##                Accuracy : 0.963
##                  95% CI : (0.9079, 0.9898)
##     No Information Rate : 0.7407
##     P-Value [Acc > NIR] : 7.548e-10
##
##                   Kappa : 0.9036
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9286
##             Specificity : 0.9750
##          Pos Pred Value : 0.9286
##          Neg Pred Value : 0.9750
##              Prevalence : 0.2593
##          Detection Rate : 0.2407
##    Detection Prevalence : 0.2593
##       Balanced Accuracy : 0.9518
##
##        'Positive' Class : 1
##
```

```r
# Extract accuracy and F1 scores
logistic_accuracy <- logistic_metrics$overall["Accuracy"]
logistic_f1 <- logistic_metrics$byClass["F1"]
rf_accuracy <- rf_metrics$overall["Accuracy"]
rf_f1 <- rf_metrics$byClass["F1"]

# Compare and interpret the results
cat("Logistic Regression Model:\n")
```

```
## Logistic Regression Model:
```

```r
cat("Accuracy:", logistic_accuracy, "\n")
```

```
## Accuracy: 0.9722222
```

```r
cat("F1 Score:", logistic_f1, "\n\n")
```

```
## F1 Score: 0.9454545
```

```r
cat("Random Forest Model:\n")
```

```
## Random Forest Model:
```

```r
cat("Accuracy:", rf_accuracy, "\n")
```

```
## Accuracy: 0.962963
```

```r
cat("F1 Score:", rf_f1, "\n")
```

```
## F1 Score: 0.9285714
```

```r
# Calculate mean and standard deviation using mutate and rowwise
train_data <- train_data %>%
  rowwise() %>%
  mutate(
    mean = mean(c_across(starts_with("H"))),
    sd = sd(c_across(starts_with("H")))
  )

test_data <- test_data %>%
  rowwise() %>%
  mutate(
    mean = mean(c_across(starts_with("H"))),
    sd = sd(c_across(starts_with("H")))
  )

# creating new dataframe
train_mean_sd <- train_data[,25:27]
test_mean_sd <- test_data[,25:27]

head(train_mean_sd,3)
```

```
## # A tibble: 3 x 3
## # Rowwise:
##   target  mean    sd
##    <int> <dbl> <dbl>
## 1      1  622.  491.
## 2      1  649.  382.
## 3      1  834.  696.
```

```r
head(test_mean_sd,3)
```

```
## # A tibble: 3 x 3
## # Rowwise:
##   target  mean    sd
##    <int> <dbl> <dbl>
## 1      1  796.  610.
## 2      1  794.  608.
## 3      1  704.  564.
```

```r
# Train logistic regression on baseline features
mod_logreg_baseline <- caret::train(
  x = train_mean_sd[, 2:3],
  y = as.factor(train_mean_sd$target),
  method = "glmnet",
  family = "binomial",
  trControl = trainControl(method = "cv"),
  tuneLength = 3
)
```

```
## Warning: Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
```

```r
# Make predictions on the test set
logistic_pred_baseline <- predict(mod_logreg_baseline, newdata = test_mean_sd)

# Evaluate baseline model
acc_logreg_baseline <- confusionMatrix(logistic_pred_baseline, as.factor(test_mean_sd$target))

# Extract accuracy and F1 scores
logistic_accuracy_mean_sd <- acc_logreg_baseline$overall["Accuracy"]
logistic_f1_mean_sd <- acc_logreg_baseline$byClass["F1"]


# Compare the performance of models
cat("\nComparison:\n")
```

```
##
## Comparison:
```

```r
print(acc_logreg_baseline)
```

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

```
##           Reference
## Prediction  1  2
##          1 21  1
##          2  7 79
##
##                Accuracy : 0.9259
##                  95% CI : (0.8593, 0.9675)
##     No Information Rate : 0.7407
##     P-Value [Acc > NIR] : 8.512e-07
##
##                   Kappa : 0.7927
##
##  Mcnemar's Test P-Value : 0.0771
##
##             Sensitivity : 0.7500
##             Specificity : 0.9875
##          Pos Pred Value : 0.9545
##          Neg Pred Value : 0.9186
##              Prevalence : 0.2593
##          Detection Rate : 0.1944
##    Detection Prevalence : 0.2037
##       Balanced Accuracy : 0.8688
##
##        'Positive' Class : 1
##
```

```r
# Train logistic regression on baseline features
mod_rf_baseline <- caret::train(
  x = train_mean_sd[,2:3],
  y = as.factor(train_mean_sd$target),
  method = "rf",
  family = "binomial",
  trControl = trainControl(method = "cv"),
  tuneLength = 3
)
```

```
## note: only 1 unique complexity parameters in default grid. Truncating the grid to 1 .
```

```
## Warning: Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
```

```r
# Make predictions on the test set
rf_pred_baseline <- predict(mod_rf_baseline, newdata = test_mean_sd)
```

```r
# Evaluate baseline model
acc_rf_baseline <- confusionMatrix(rf_pred_baseline, as.factor(test_mean_sd$target))

# Extract accuracy and F1 scores
rf_accuracy_mean_sd <- acc_rf_baseline$overall["Accuracy"]
rf_f1_mean_sd <- acc_rf_baseline$byClass["F1"]

# Compare the performance of models
cat("\nComparison:\n")
```

```
##
## Comparison:
```

```r
print(acc_rf_baseline)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 19  6
##          2  9 74
##
##                Accuracy : 0.8611
##                  95% CI : (0.7813, 0.9201)
##     No Information Rate : 0.7407
##     P-Value [Acc > NIR] : 0.001874
##
##                   Kappa : 0.6253
##
##  Mcnemar's Test P-Value : 0.605577
##
##             Sensitivity : 0.6786
##             Specificity : 0.9250
##          Pos Pred Value : 0.7600
##          Neg Pred Value : 0.8916
##              Prevalence : 0.2593
##          Detection Rate : 0.1759
##    Detection Prevalence : 0.2315
##       Balanced Accuracy : 0.8018
##
##        'Positive' Class : 1
##
```

```r
# Comparing all the model results from both task c and d

# Logistic regression

cat("\nLogistic Regression Model with tsfeatures:\n")
```

```
##
## Logistic Regression Model with tsfeatures:
```

```r
cat("Accuracy:", logistic_accuracy, "\n")
```

```
## Accuracy: 0.9722222
```

```r
cat("F1 Score:", logistic_f1, "\n\n")
```

```
## F1 Score: 0.9454545
```

```r
cat("Logistic Regression Baseline:\n")
```

```
## Logistic Regression Baseline:
```

```r
cat("Accuracy:", logistic_accuracy_mean_sd, "\n")
```

```
## Accuracy: 0.9259259
```

```r
cat("F1 Score:", logistic_f1_mean_sd, "\n\n")
```

```
## F1 Score: 0.84
```

```r
# Random forest

cat("Random Forest Model with tsfeatures:\n")
```

```
## Random Forest Model with tsfeatures:
```

```r
cat("Accuracy:", rf_accuracy, "\n")
```

```
## Accuracy: 0.962963
```

```r
cat("F1 Score:", rf_f1, "\n\n")
```

```
## F1 Score: 0.9285714
```

```r
cat("random forest Regression Baseline :\n")
```

```
## random forest Regression Baseline :
```

```r
cat("Accuracy:", rf_accuracy_mean_sd, "\n")
```

```
## Accuracy: 0.8611111
```

```r
cat("F1 Score:", rf_f1_mean_sd, "\n\n")
```

```
## F1 Score: 0.7169811
```

Exercise 4

Task a)

```r
coffee_data <- read.csv("coffee_train.csv")

test_coffee <- read.csv("coffee_test.csv")
```

```r
# Exploratory Data Analysis
print(head(coffee_data,1))
```

```
##   index target        V2         V3         V4         V5         V6        V7
## 1     1      0 -0.518419 -0.4858836 -0.5050075 -0.5601829 -0.6362994 -0.753229
##           V8         V9        V10        V11        V12        V13        V14
## 1 -0.8272291 -0.8597647 -0.9063207 -0.9237965 -0.9332122 -0.9344293 -0.9207872
##          V15        V16       V17       V18        V19        V20        V21
## 1 -0.9365719 -0.9559969 -0.959341 -0.969406 -0.9824055 -0.9765995 -0.9625258
##          V22        V23        V24      V25       V26       V27       V28
## 1 -0.9703189 -0.9819924 -0.9874616 -1.01522 -1.048061 -1.059242 -1.049885
##         V29       V30       V31       V32       V33       V34       V35
## 1 -1.046732 -1.064399 -1.087794 -1.099335 -1.085081 -1.064191 -1.054523
##         V36       V37        V38        V39        V40       V41        V42
## 1 -1.044503 -1.023637 -0.9942171 -0.9637198 -0.9223722 -0.866169 -0.8135157
##          V43        V44        V45       V46        V47        V48        V49
## 1 -0.7726676 -0.7234398 -0.6642433 -0.634246 -0.6280451 -0.6046936 -0.5682978
##          V50        V51        V52        V53        V54        V55         V56
## 1 -0.5377832 -0.5056997 -0.4612259 -0.3889274 -0.2929667 -0.2005633 -0.09099993
##           V57        V58       V59       V60       V61       V62      V63
## 1 0.008419047 0.08265814 0.2202938 0.3421723 0.4288343 0.5548792 0.549875
##         V64     V65       V66       V67       V68      V69       V70       V71
## 1 0.5104567 0.60485 0.6628724 0.6490801 0.7082333 0.790188 0.8614067 0.9066245
##         V72      V73      V74      V75      V76       V77       V78       V79
## 1 0.9917938 1.073976 1.047915 1.061288 1.035781 0.9600131 0.8966958 0.7829689
##         V80       V81       V82       V83       V84       V85       V86
## 1 0.7389871 0.7380739 0.7861767 0.7996705 0.7592426 0.7087103 0.6322128
##        V87       V88       V89       V90       V91      V92       V93       V94
## 1 0.636782 0.6333657 0.6359178 0.6603797 0.6844996 0.679707 0.6353288 0.6073989
##         V95       V96       V97       V98       V99      V100      V101
## 1 0.6031809 0.5153471 0.4463631 0.4497672 0.4276543 0.3612676 0.3186955
##        V102      V103      V104      V105      V106      V107      V108
## 1 0.2946546 0.2304915 0.2159363 0.2023713 0.2116452 0.2375017 0.2030858
##        V109      V110     V111      V112      V113      V114      V115    V116
## 1 0.1827435 0.2048752 0.236739 0.3138302 0.3832218 0.4050667 0.4396105 0.48577
##        V117      V118      V119      V120      V121      V122      V123
## 1 0.5335979 0.5615982 0.6031303 0.6474101 0.6407156 0.6786326 0.7238296
##        V124     V125      V126     V127      V128      V129      V130      V131
## 1 0.7136778 0.735464 0.6906632 0.632419 0.5672294 0.5124406 0.4748466 0.4324775
##        V132      V133      V134      V135      V136      V137      V138
## 1 0.4044378 0.3390347 0.2620487 0.2230598 0.2414072 0.2403556 0.2072977
##        V139      V140      V141      V142      V143      V144      V145
## 1 0.2128129 0.2494869 0.2562097 0.2665836 0.3389987 0.4433204 0.5402046
##        V146      V147      V148      V149      V150      V151      V152
## 1 0.6352104 0.7082664 0.7658252 0.8743454 0.8832562 0.8463181 0.9807682
##       V153     V154     V155     V156     V157     V158     V159      V160
## 1 1.046898 1.040239 1.037378 1.032466 1.032877 1.043206 1.025169 0.9572421
##        V161      V162      V163      V164      V165      V166      V167
## 1 0.9823372 0.9352685 0.8430386 0.7601121 0.7210253 0.6747341 0.6162969
```

```
##        V168      V169      V170     V171      V172      V173        V174
## 1 0.6064854 0.5967017 0.4825903 0.324634 0.2011121 0.1009692 -0.03416513
##         V175       V176      V177     V178       V179       V180       V181
## 1 -0.1594421 -0.2496523 -0.319281 -0.34077 -0.3573074 -0.3549524 -0.3420819
##        V182       V183       V184       V185       V186        V187        V188
## 1 -0.342095 -0.2866997 -0.2209032 -0.1878498 -0.1403672 -0.08561849 -0.09316192
##        V189       V190       V191        V192       V193      V194      V195
## 1 -0.118843 -0.1480344 -0.1347337 -0.03841305 0.03704008 0.1375908 0.2645301
##        V196      V197      V198      V199     V200      V201     V202     V203
## 1 0.3918031 0.5022774 0.5545248 0.6248466 0.719324 0.9026986 1.030115 1.114387
##       V204     V205    V206     V207     V208     V209     V210     V211
## 1 1.258064 1.398263 1.49467 1.586657 1.639197 1.701021 1.772069 1.802299
##       V212     V213     V214     V215     V216     V217     V218     V219
## 1 1.663724 1.548187 1.505758 1.408008 1.412457 1.467514 1.518852 1.533485
##       V220     V221     V222     V223     V224     V225     V226      V227
## 1 1.469025 1.663035 1.679521 1.524594 1.350075 1.175358 1.008876 0.9080989
##        V228      V229      V230    V231     V232     V233     V234     V235
## 1 0.9052138 0.9461304 0.9643524 1.02457 1.228416 1.315338 1.373425 1.354578
##       V236      V237      V238      V239        V240       V241       V242
## 1 1.069437 0.7506417 0.4867503 0.2382077 -0.01533208 -0.2477473 -0.4584269
##         V243       V244       V245      V246      V247      V248      V249
## 1 -0.6072507 -0.7793242 -0.9083501 -1.018756 -1.084125 -1.103827 -1.100227
##        V250      V251      V252      V253      V254      V255      V256
## 1 -1.096213 -1.092783 -1.082716 -1.093841 -1.148036 -1.237549 -1.347114
##        V257      V258      V259      V260      V261      V262      V263
## 1 -1.481561 -1.593583 -1.687037 -1.751625 -1.794304 -1.824178 -1.843286
##        V264      V265      V266     V267      V268      V269      V270    V271
## 1 -1.856656 -1.866502 -1.875271 -1.88185 -1.887185 -1.892067 -1.896899 -1.9006
##       V272      V273      V274      V275      V276      V277      V278
## 1 -1.90435 -1.907744 -1.909911 -1.912719 -1.916328 -1.918193 -1.920125
##        V279      V280      V281      V282      V283      V284      V285
## 1 -1.922313 -1.924212 -1.926997 -1.928721 -1.930026 -1.932301 -1.933631
##        V286      V287
## 1 -1.934964 -1.936007
```

```r
# Removing the 'target' column before checking for NaN values
coffee_data_no_target <- dplyr::select(coffee_data, -target)
coffee_test_no_target <- dplyr::select(test_coffee, -target)

# Checking for NaN values in both datasets (excluding 'target' column)
sum_nan_coffee_data <- sapply(coffee_data_no_target, function(x) sum(is.nan(x)))
sum_nan_coffee_test <- sapply(coffee_test_no_target, function(x) sum(is.nan(x)))

print("NaN values in coffee_data (excluding 'target'):")
```

```
## [1] "NaN values in coffee_data (excluding 'target'):"
```

```r
print(sum(sum_nan_coffee_data))
```

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

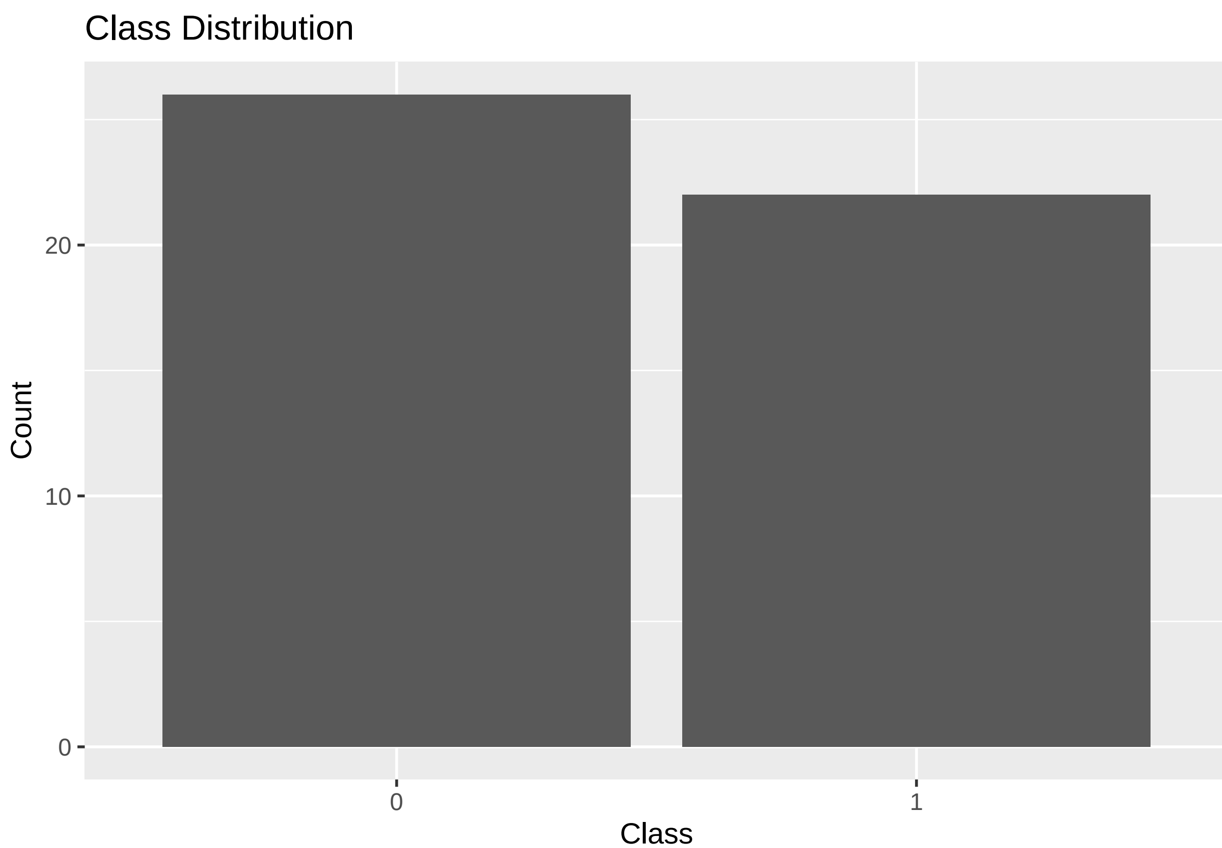
```r
print("NaN values in test_coffee (excluding 'target'):")
```

```
## [1] "NaN values in test_coffee (excluding 'target'):"
```

```r
print(sum(sum_nan_coffee_test))
```

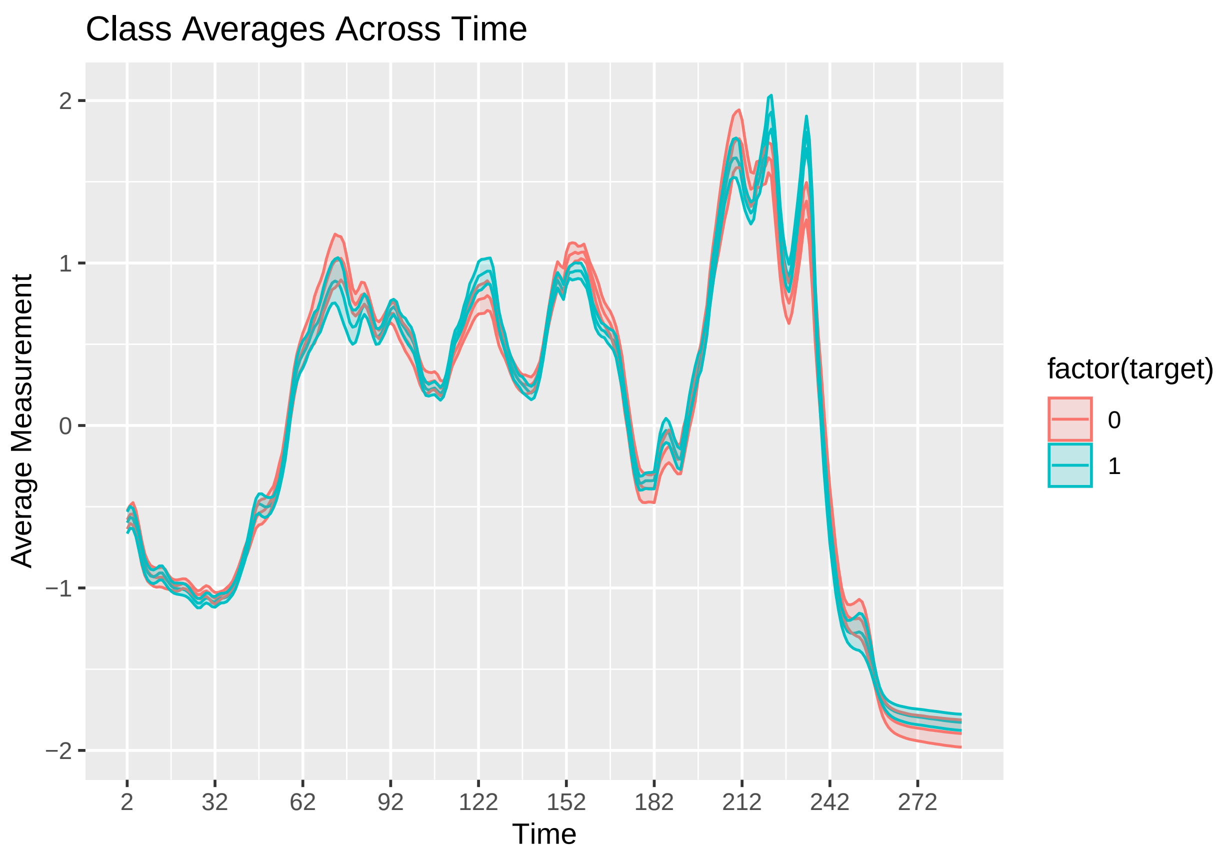
```
## [1] 0
```

```r
# Plotting Class Distributions
ggplot(coffee_data, aes(x = factor(target))) +
    geom_bar() +
    labs(title = "Class Distribution", x = "Class", y = "Count")
```



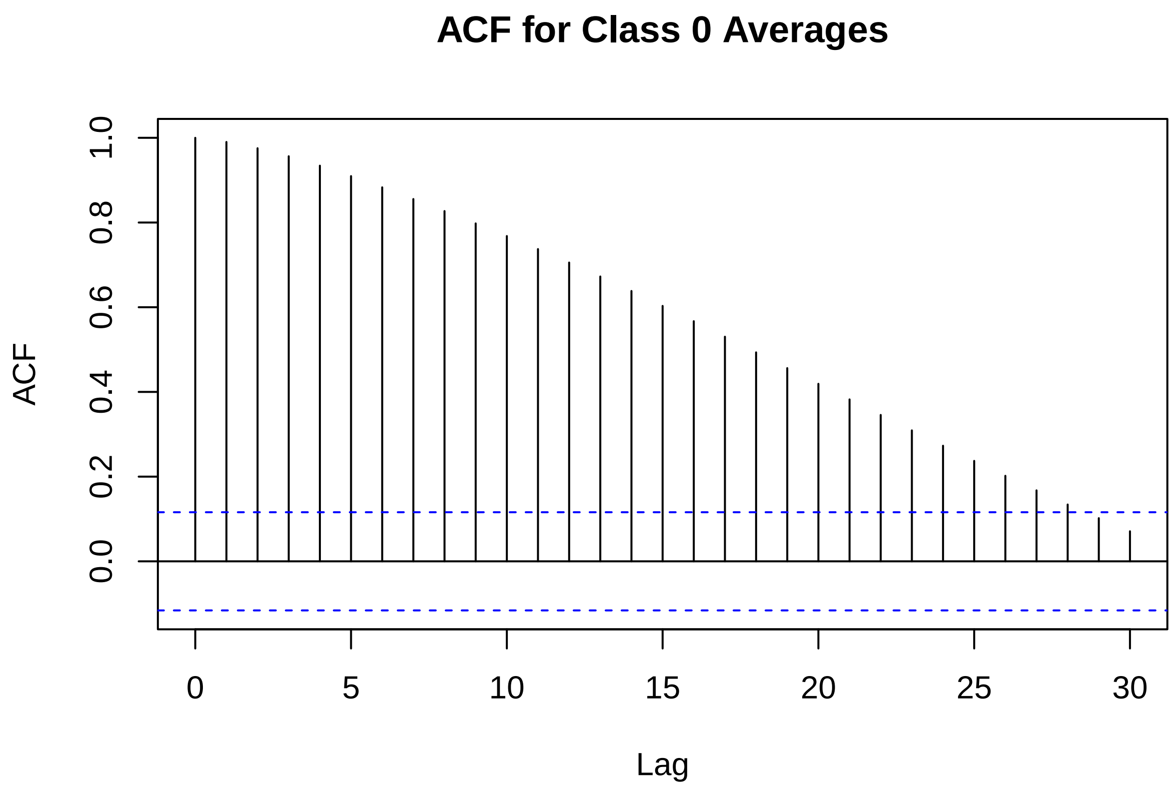

```r
# Preparing data for time series plot
# Melting the data frame from wide to long format
long_coffee_data <- coffee_data %>%
    pivot_longer(cols = starts_with("V"), names_to = "time",
                 values_to = "value") %>%
    mutate(time = as.numeric(gsub("V", "", time)))

# Calculating mean and standard deviation for each class and time
class_stats <- long_coffee_data %>%
    group_by(target, time) %>%
```

```r
    summarize(mean = mean(value), sd = sd(value), .groups = 'drop')

# Plotting Class Averages Across Time with Shading
ggplot(class_stats, aes(x = time, y = mean, color = factor(target))) +
    geom_line() +
    geom_ribbon(aes(ymin = mean - sd, ymax = mean + sd, fill = factor(target)),
                alpha = 0.2) +
    labs(title = "Class Averages Across Time", x = "Time",
         y = "Average Measurement") +
    scale_x_continuous(breaks = seq(min(class_stats$time),
                                    max(class_stats$time), by = 30))
```

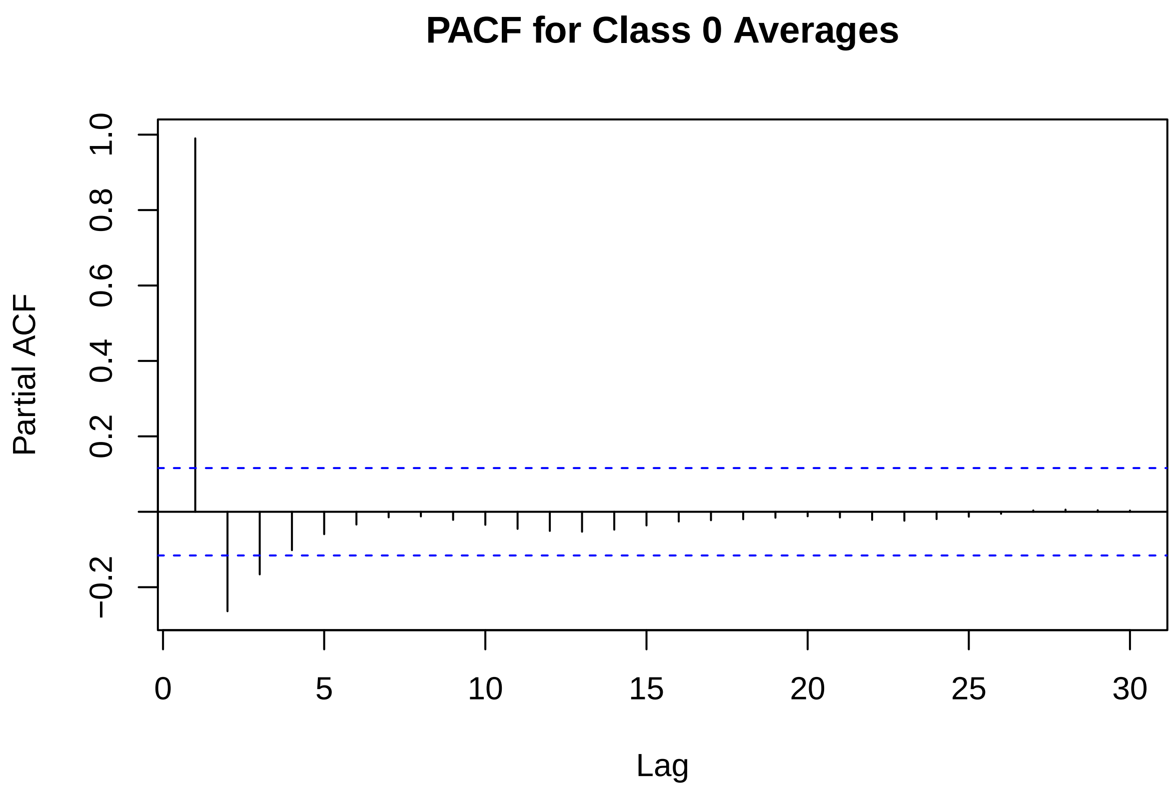



```r
# Melting the data frame from wide to long format
long_coffee_data <- coffee_data %>%
    pivot_longer(cols = starts_with("V"), names_to = "time",
                 values_to = "value") %>%
    mutate(time = as.numeric(gsub("V", "", time)))

# Calculating mean for each class and time
class_averages <- long_coffee_data %>%
    group_by(target, time) %>%
    summarize(mean = mean(value), .groups = 'drop')

# Separate the data by class
class_0_averages <- filter(class_averages,
```

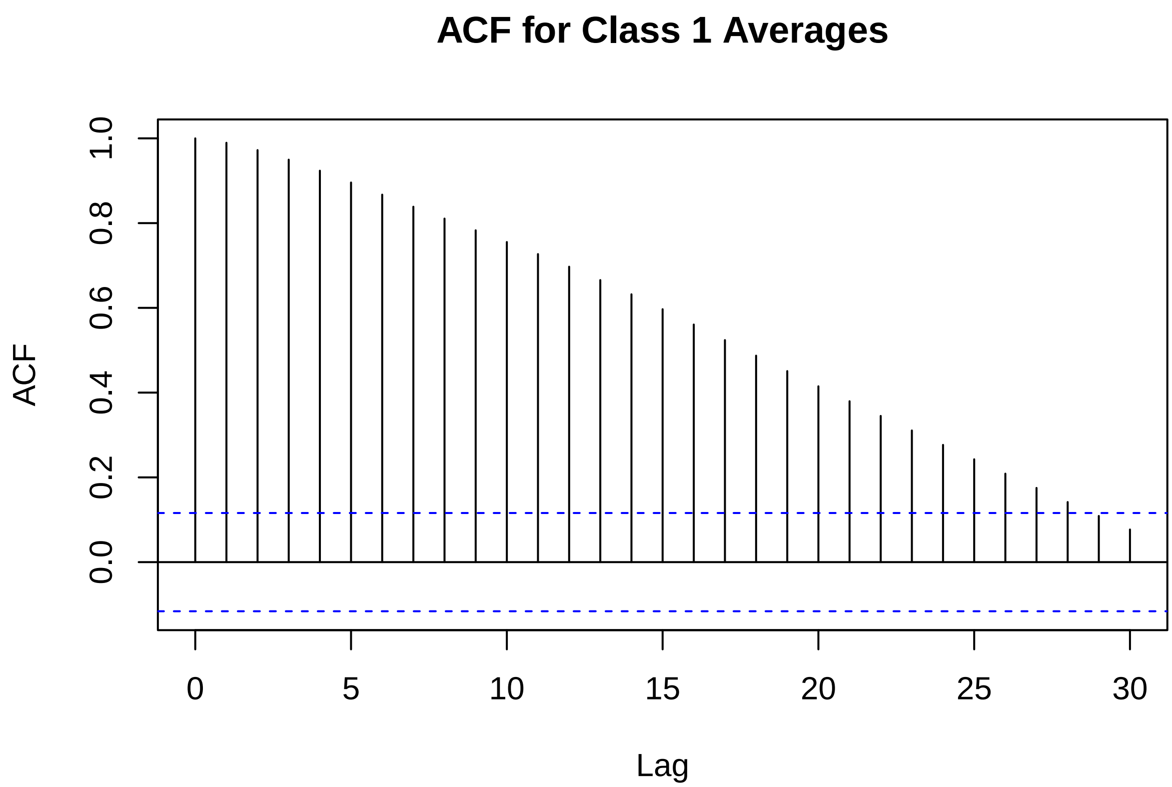
```
                              target == 0) %>% arrange(time) %>% dplyr::select(mean)
class_1_averages <- filter(class_averages,
                              target == 1) %>% arrange(time) %>% dplyr::select(mean)

# ACF and PACF for Class 0
acf(as.numeric(class_0_averages$mean), main="ACF for Class 0 Averages",
    lag.max = 30)
```



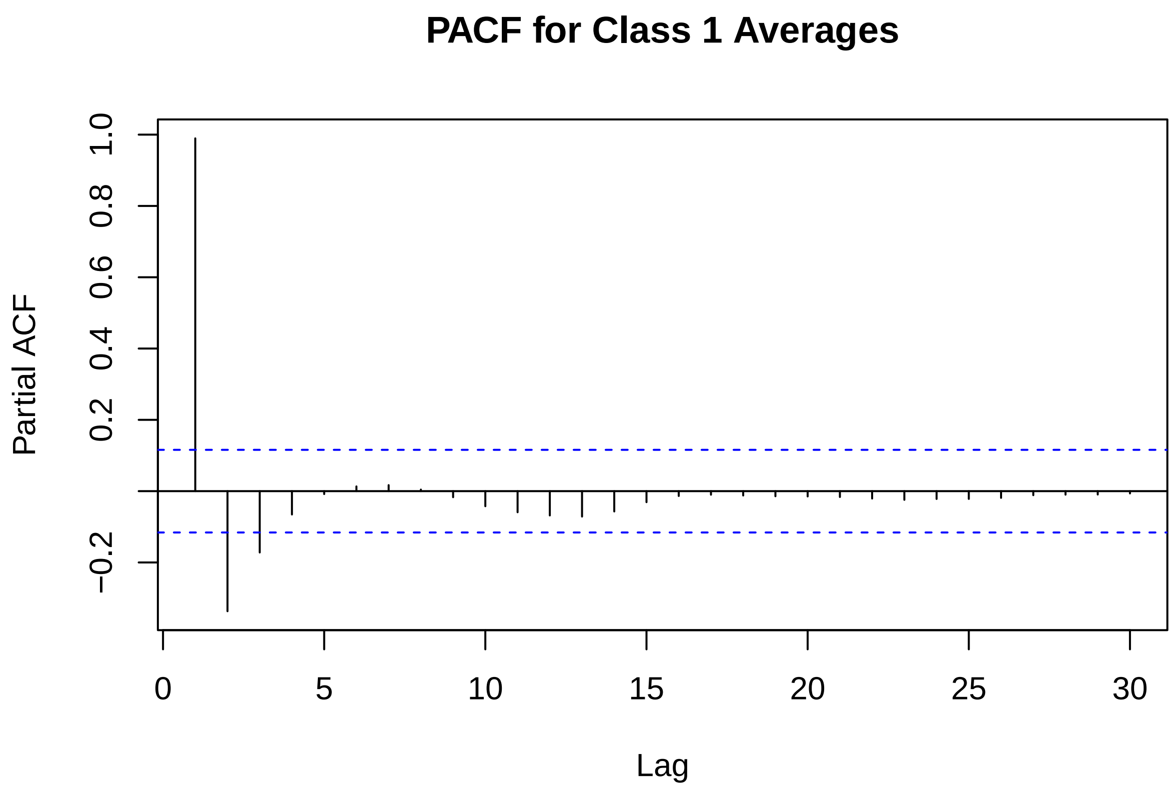

```
pacf(as.numeric(class_0_averages$mean), main="PACF for Class 0 Averages",
     lag.max = 30)
```



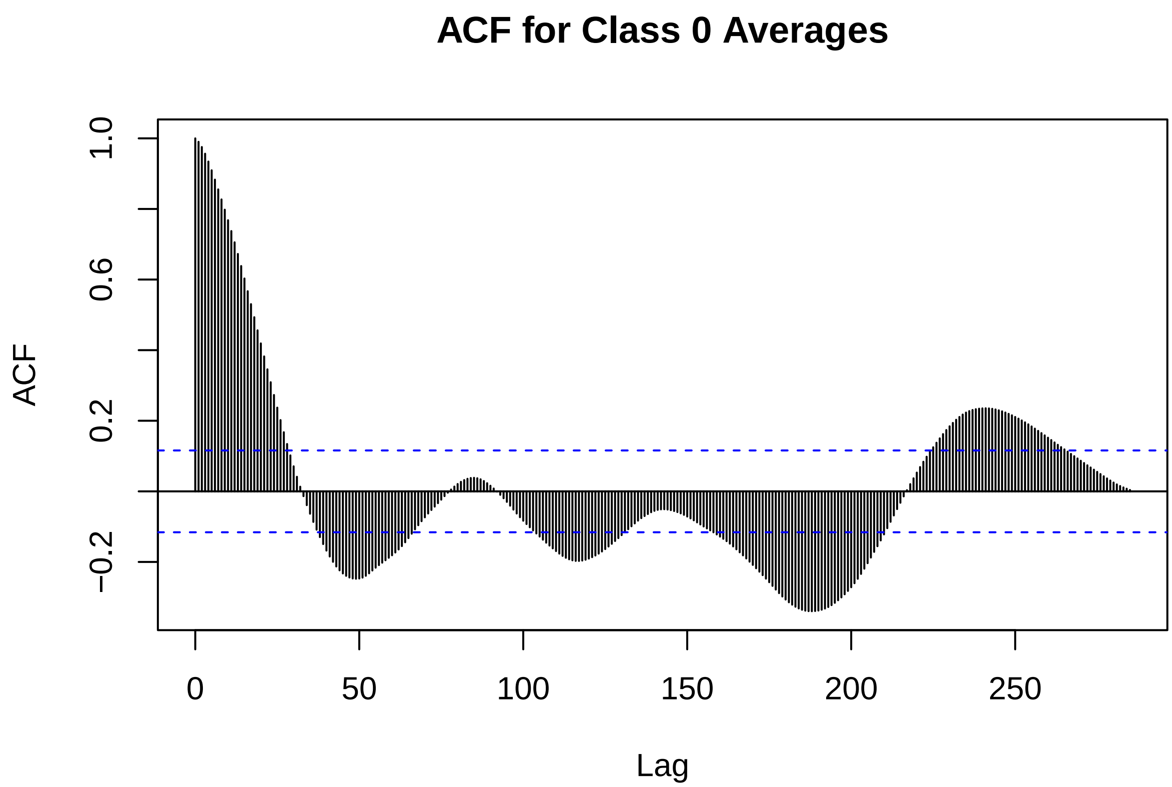

```
# ACF and PACF for Class 1
acf(as.numeric(class_1_averages$mean), main="ACF for Class 1 Averages",
    lag.max = 30)
```



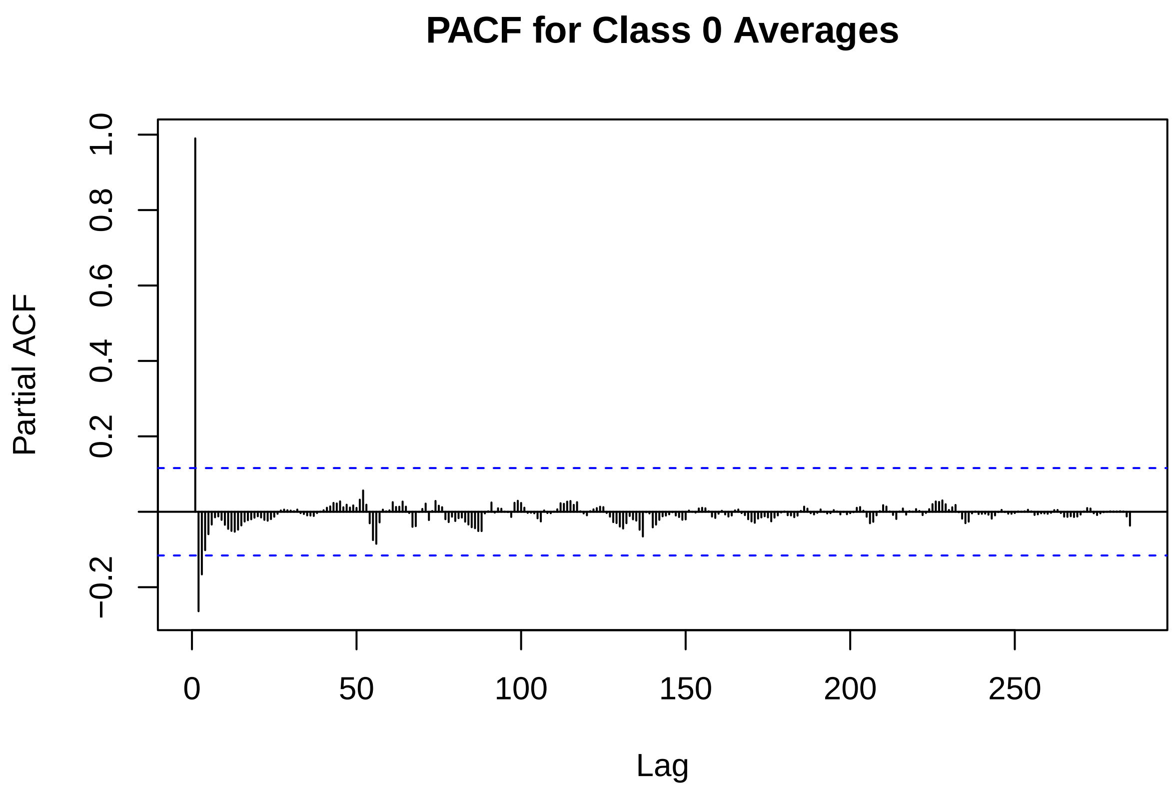

```
pacf(as.numeric(class_1_averages$mean), main="PACF for Class 1 Averages",
     lag.max = 30)
```



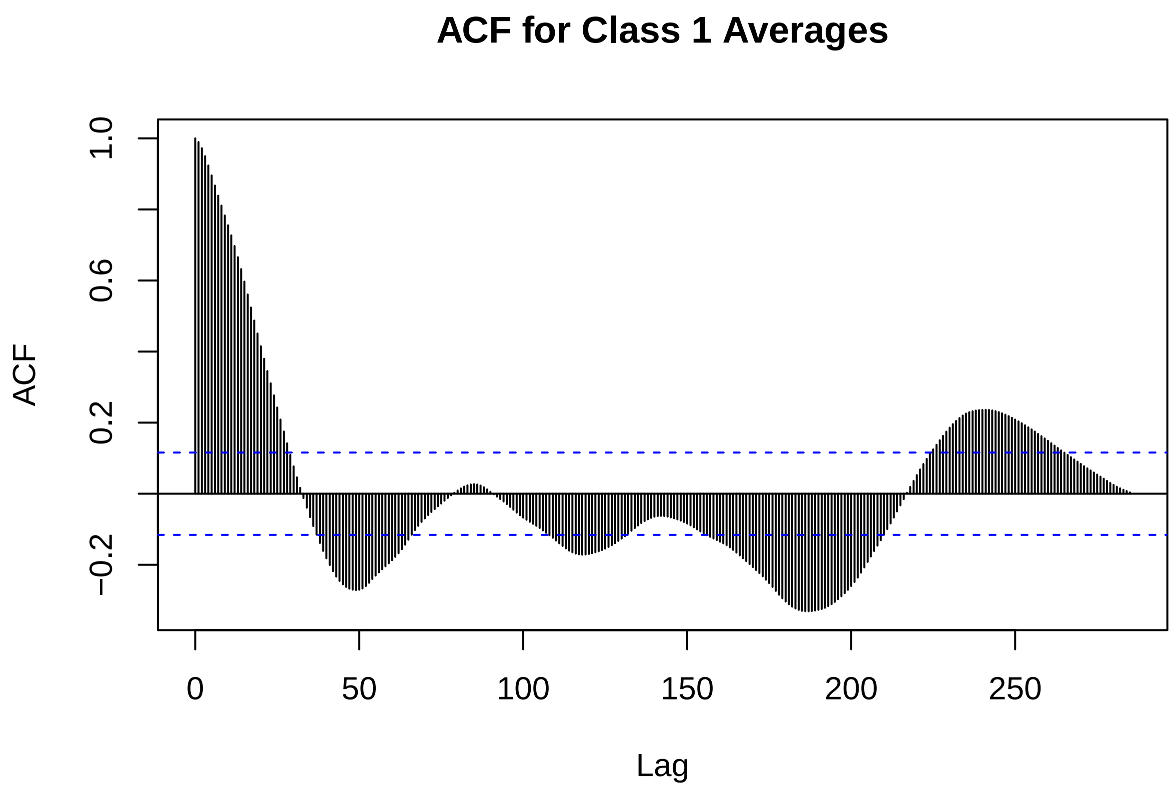

```
# ACF and PACF for Class 0
acf(as.numeric(class_0_averages$mean), main="ACF for Class 0 Averages",
    lag.max = 365)
```



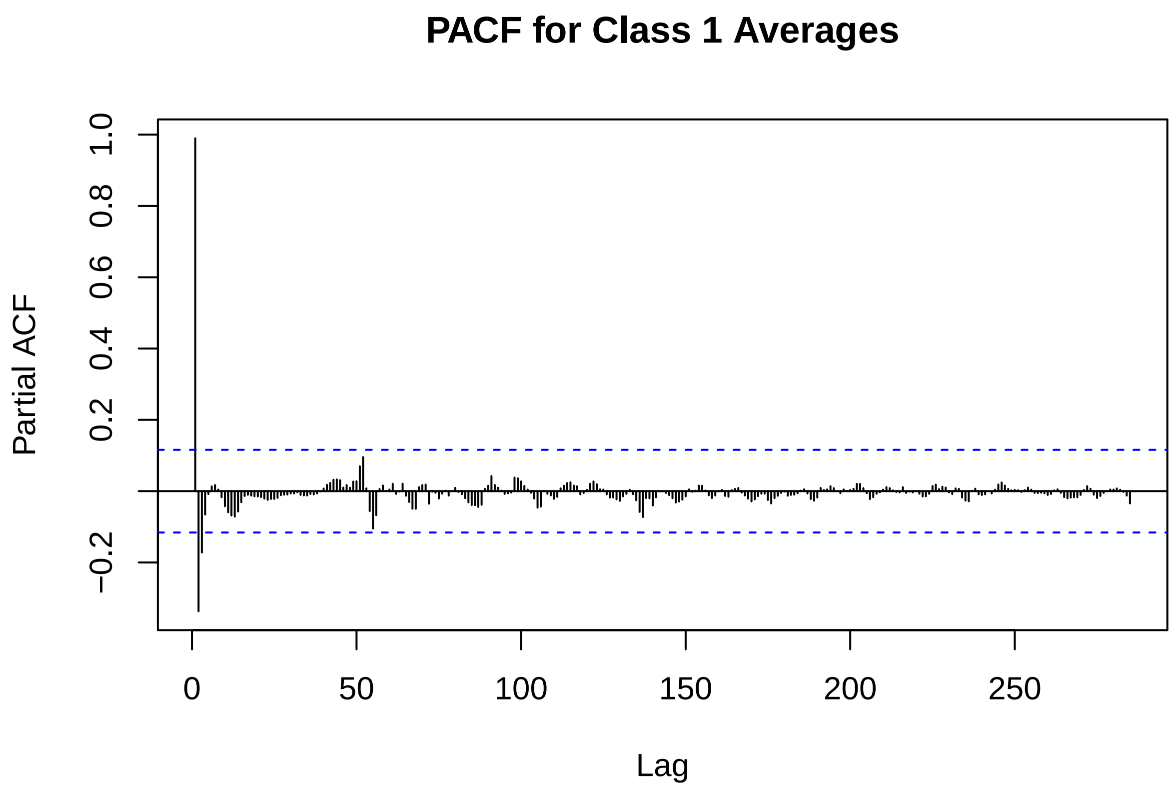

```
pacf(as.numeric(class_0_averages$mean), main="PACF for Class 0 Averages",
     lag.max = 365)
```




```r
# ACF and PACF for Class 1
acf(as.numeric(class_1_averages$mean), main="ACF for Class 1 Averages",
    lag.max = 365)
```



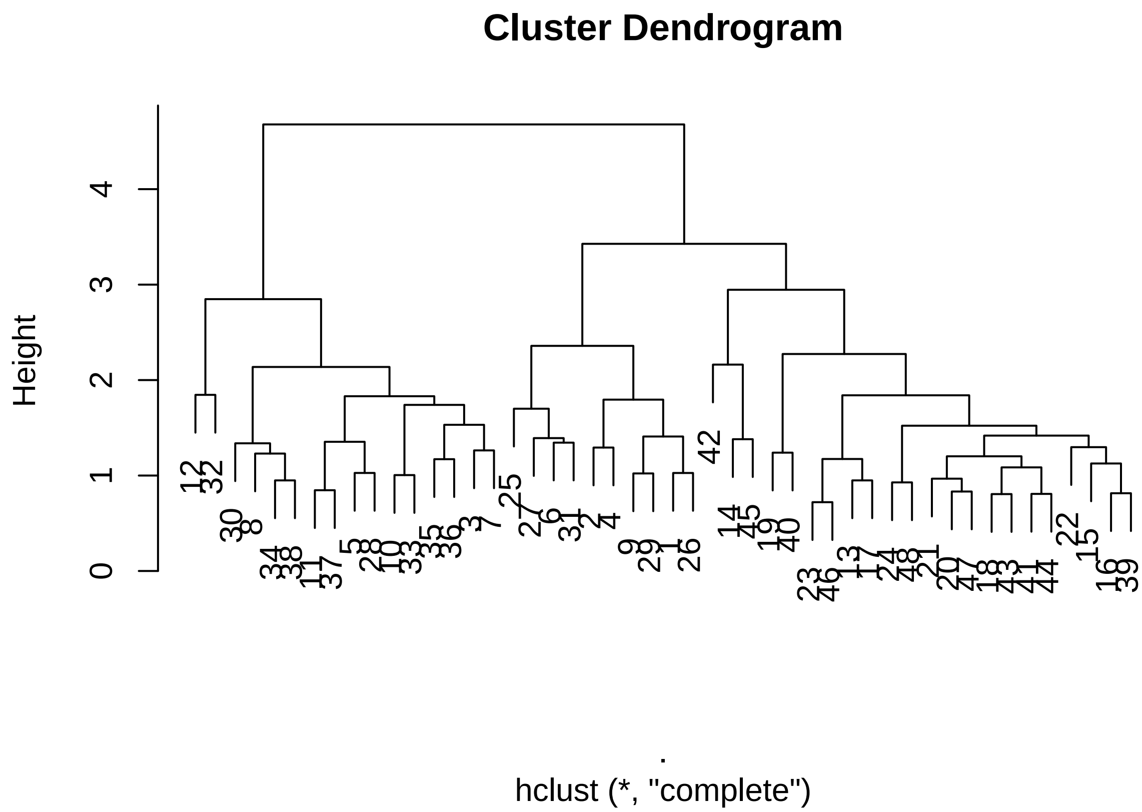

```r
pacf(as.numeric(class_1_averages$mean), main="PACF for Class 1 Averages",
     lag.max = 365)
```



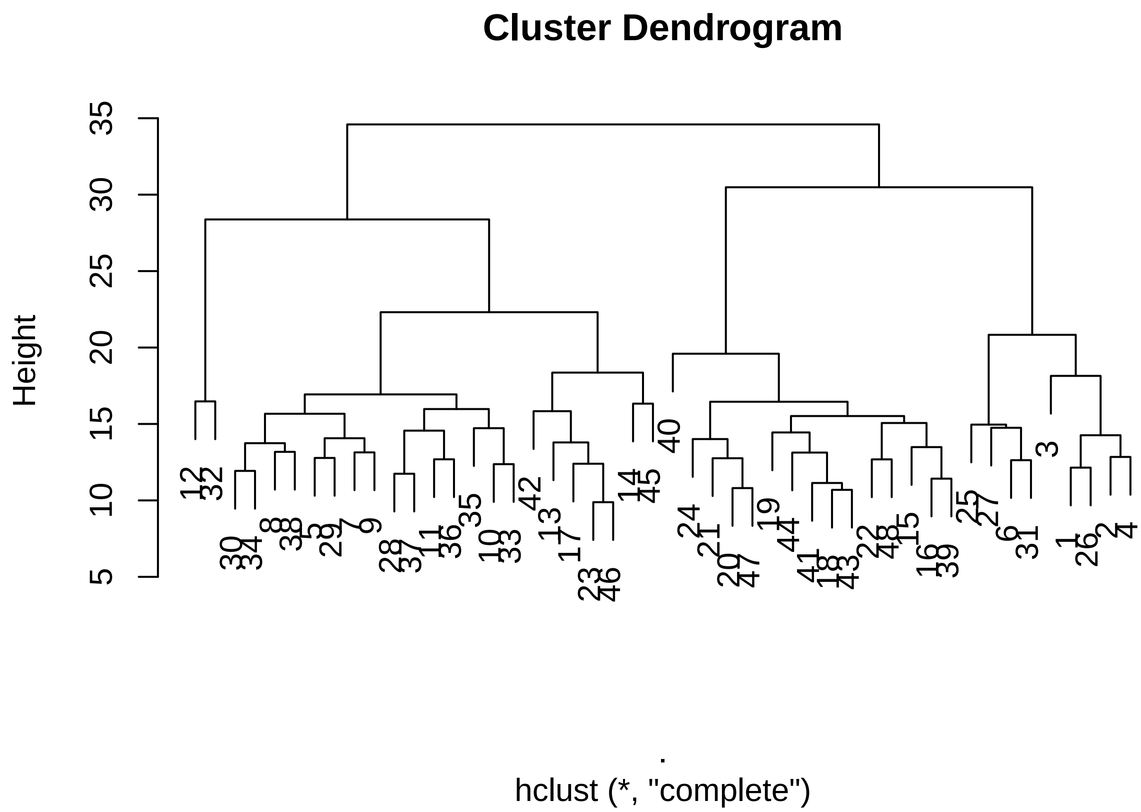
From the information gathered from the exploratory analysis, one can see that the two classes are almost identical. This can be seen from the graph of the class averages over time, where the standard deviation of the two classes overlap several times. This is also evident from the ACF plots, where the differences between the classes are minuscule.

Task b)

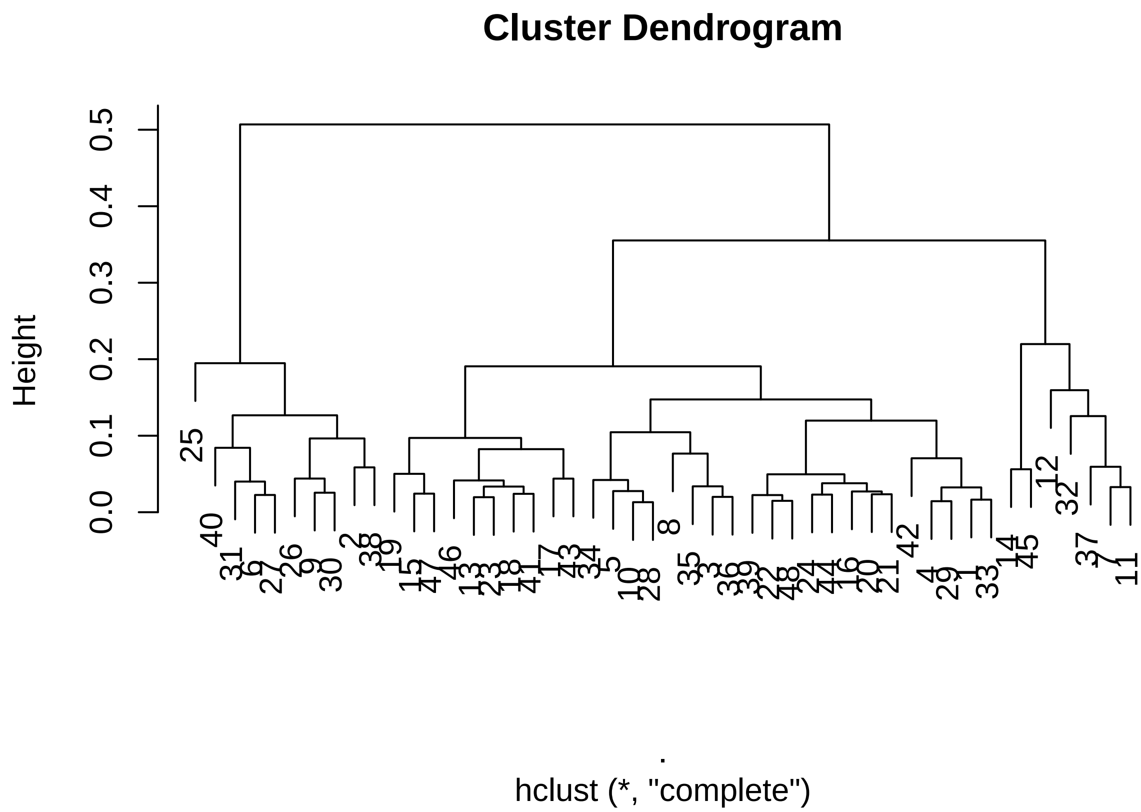
```
# Cluster for euclidian distance

euclidean_dist <- coffee_data %>%
  dplyr::select(-target, -index) %>%
  t() %>%
  as.data.frame() %>%
  TSclust::diss(METHOD = "EUCL")

names(euclidean_dist) <- coffee_data$index


euclidean_dist %>%
  hclust() %>%
  plot()
```



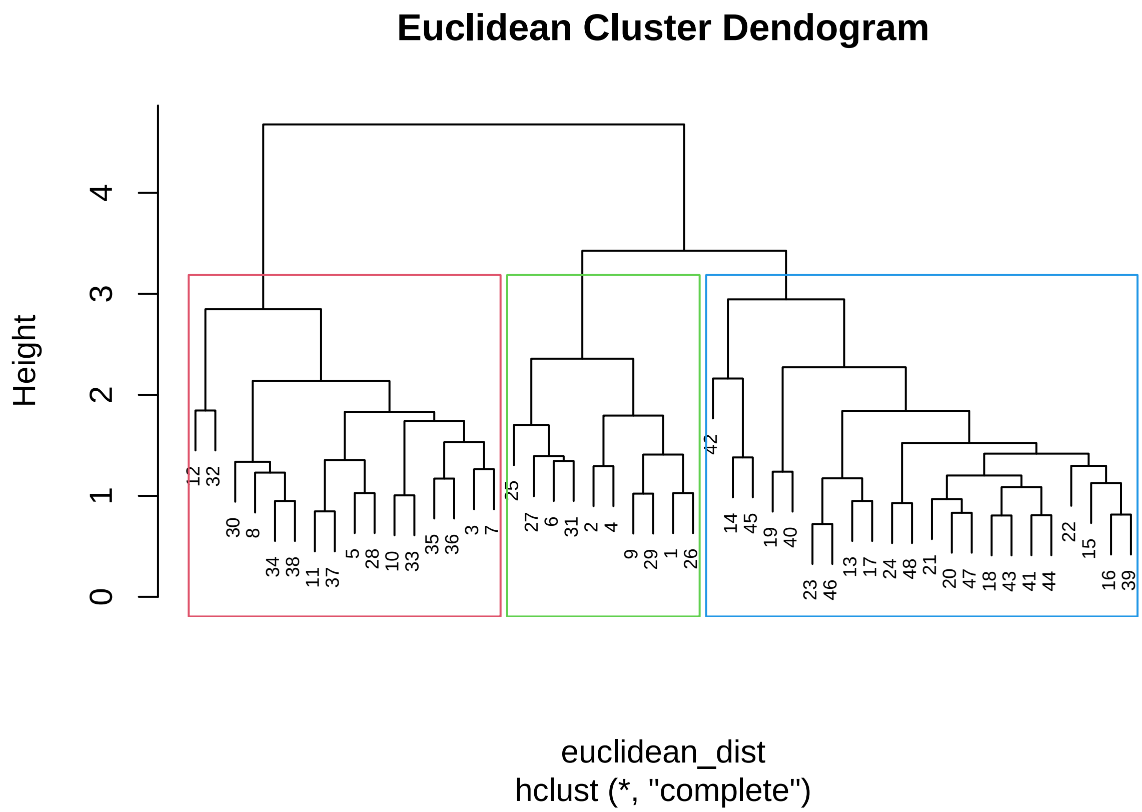

```r
# Cluster for DTW distance
dtw_dist <- coffee_data %>%
  dplyr::select(-target, -index) %>%
  t() %>%
  as.data.frame() %>%
  TSclust::diss(METHOD = "DTWARP")

names(dtw_dist) <- coffee_data$index


dtw_dist %>%
  hclust() %>%
  plot()
```



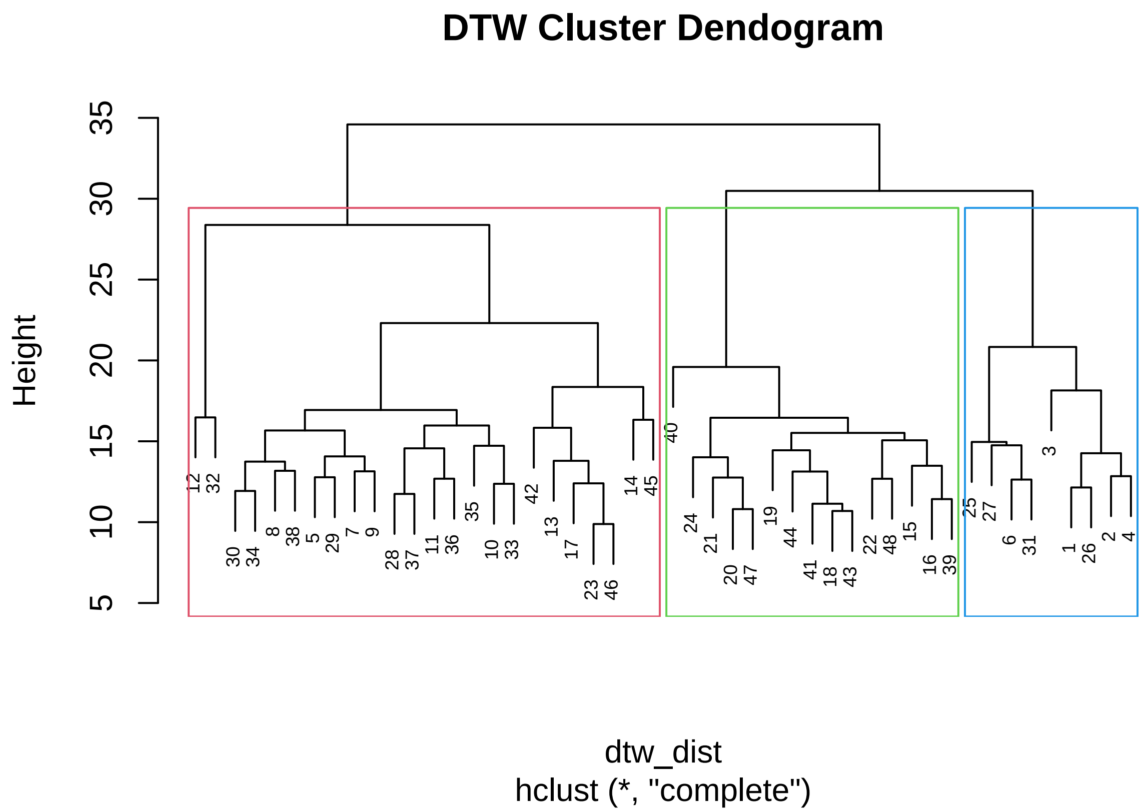

```
# Cluster for ACF distance

acf_dist <- coffee_data %>%
  dplyr::select(-target, -index) %>%
  t() %>%
  as.data.frame() %>%
  TSclust::diss(METHOD = "ACF")

names(acf_dist) <- coffee_data$index


acf_dist %>%
  hclust() %>%
  plot()
```



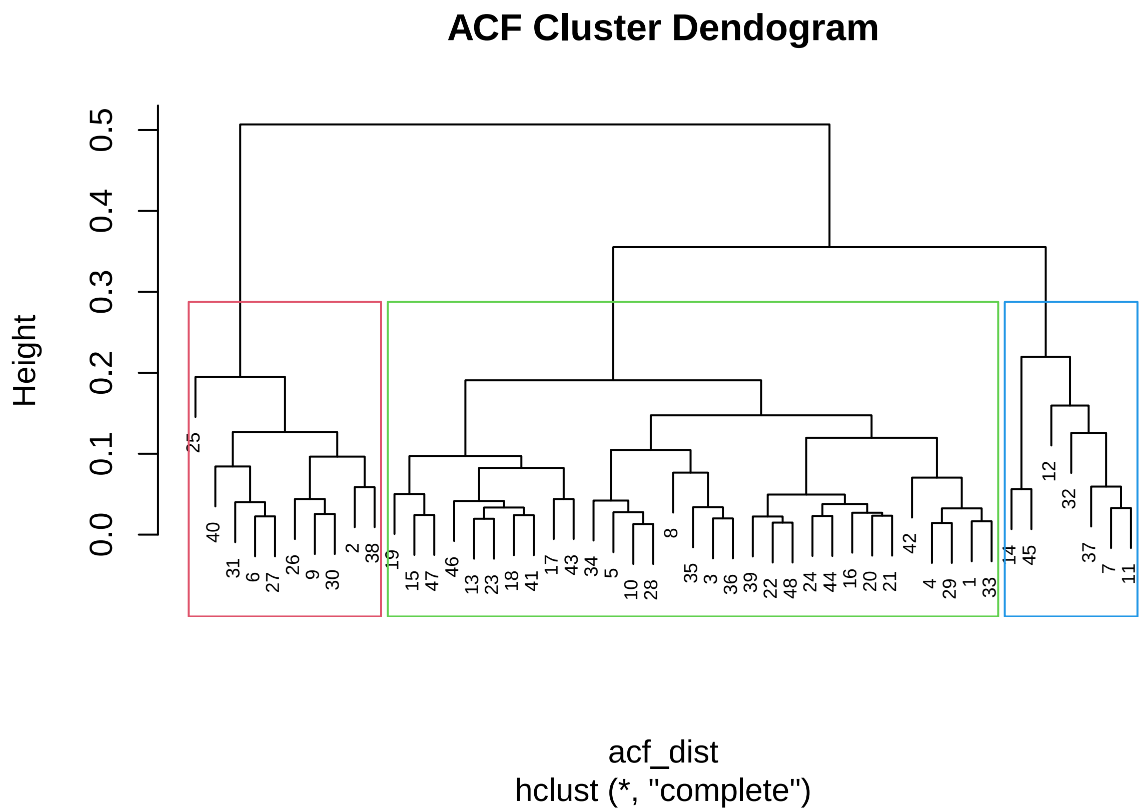

```
# Perform hierarchical clustering
hc_euclidean <- hclust(euclidean_dist)

# Cut the dendrogram into 3 clusters
clusters_euclidean <- cutree(hc_euclidean, k = 3)

# Plot the dendrogram with color-coded clusters
plot(hc_euclidean, labels = coffee_data$index, cex = 0.6,
     main = "Euclidean Cluster Dendogram")
rect.hclust(hc_euclidean, k = 3, border = 2:4)
```




```r
# Perform hierarchical clustering
hc_dtw <- hclust(dtw_dist)

# Cut the dendrogram into 3 clusters
clusters_dtw <- cutree(hc_dtw, k = 3)

# Plot the dendrogram with color-coded clusters
plot(hc_dtw, labels = coffee_data$index, cex = 0.6,
     main = "DTW Cluster Dendogram")
rect.hclust(hc_dtw, k = 3, border = 2:4)
```




```r
# Perform hierarchical clustering
hc_acf <- hclust(acf_dist)

# Cut the dendrogram into 3 clusters
clusters_acf <- cutree(hc_acf, k = 3)

# Plot the dendrogram with color-coded clusters
plot(hc_acf, labels = coffee_data$index, cex = 0.6,
     main = "ACF Cluster Dendogram")
rect.hclust(hc_acf, k = 3, border = 2:4)
```



Task c)

```
coffee_data$cluster_euclidean <- clusters_euclidean
coffee_data$cluster_dtw <- clusters_dtw
coffee_data$cluster_acf <- clusters_acf
```

```
# Function to calculate purity
calculate_purity <- function(data, cluster_col) {
  data %>%
    group_by(!!sym(cluster_col), target) %>%
    summarise(count = n(), .groups = 'drop') %>%
    arrange(!!sym(cluster_col), desc(count)) %>%
    group_by(!!sym(cluster_col)) %>%
    summarise(purity = first(count) / sum(count), .groups = 'drop')
}

# Calculate purity for each clustering method
purity_euclidean <- calculate_purity(coffee_data, "cluster_euclidean")
purity_dtw <- calculate_purity(coffee_data, "cluster_dtw")
purity_acf <- calculate_purity(coffee_data, "cluster_acf")

# Calculate average purity
avg_purity_euclidean <- mean(purity_euclidean$purity)
avg_purity_dtw <- mean(purity_dtw$purity)
```

```r
avg_purity_acf <- mean(purity_acf$purity)

# Print average purities
print(avg_purity_euclidean)
```

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

```r
print(avg_purity_dtw)
```

```
## [1] 0.9027778
```

```r
print(avg_purity_acf)
```

```
## [1] 0.7423963
```

```r
best_method <- which.max(c(avg_purity_euclidean, avg_purity_dtw, avg_purity_acf))
method_names <- c("Euclidean", "DTW", "ACF")
print(paste("Best method:", method_names[best_method]))
```

```
## [1] "Best method: Euclidean"
```

The best method is the Euclidean method, with a purity of 1. The number does seem a little bit too good to be true, because it does not seem realistic.

Task d)

```r
distances  <- euclidean_dist %>%
  hclust() %>%
  stats::cutree(k=3)
```

```r
# Add cluster assignments to the coffee_data
coffee_data$cluster <- cutree(hclust(euclidean_dist), k = 3)

# Function to calculate centroids
calculate_centroids <- function(data, cluster_col) {
  data %>%
    dplyr::select(starts_with("V"), all_of(cluster_col)) %>%
    group_by(!!sym(cluster_col)) %>%
    summarise(across(starts_with("V"), mean, na.rm = TRUE), .groups = 'drop')
}

# Calculate centroids
centroids <- calculate_centroids(coffee_data, "cluster")
```

```
## Warning: There was 1 warning in ‘summarise()‘.
## i In argument: ‘across(starts_with("V"), mean, na.rm = TRUE)‘.
## i In group 1: ‘cluster = 1‘.
## Caused by warning:
```

```
## ! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
## Supply arguments directly to `.fns` through an anonymous function instead.
##
##   # Previously
##   across(a:b, mean, na.rm = TRUE)
##
##   # Now
##   across(a:b, \(x) mean(x, na.rm = TRUE))
```

```r
# Function to assign test data to the nearest centroid
assign_to_nearest_centroid <- function(test_data, centroids) {
  v_cols <- colnames(centroids)[colnames(centroids) != "cluster"]

  sapply(1:nrow(test_data), function(i) {
    distances <- sapply(1:nrow(centroids), function(j) {
      dist(rbind(test_data[i, v_cols], centroids[j, v_cols]))
    })
    which.min(distances)
  })
}

# Assign clusters to the test data
test_coffee$assigned_cluster <- assign_to_nearest_centroid(test_coffee,
                                                           centroids)
```

```r
# Make dataframe from the target and cluster assignments

clust.table <- coffee_data %>%
  dplyr::select(index, target) %>%
  mutate(clust.ecl = distances)
clust.table %>% group_by(clust.ecl) %>% count(target)
```

```
## # A tibble: 3 x 3
## # Groups:   clust.ecl [3]
##   clust.ecl target     n
##       <int>  <int> <int>
## 1         1      0    10
## 2         2      0    16
## 3         3      1    22
```

```r
# After determining the majority class for each cluster
# we can assign the cluster to the target
cluster_to_target <- c("1" = 0, "2" = 0, "3" = 1)

test_coffee$predicted_target <- sapply(test_coffee$assigned_cluster, function(cluster) cluster_to_target


confusion_matrix <- table(Predicted = test_coffee$predicted_target
                          , Actual = test_coffee$target)

print(confusion_matrix)
```

```
##          Actual
```

```
## Predicted 0 1
##         0 3 0
##         1 0 2
```