# Compulsory 2

Group 1: Joel Yacob, Artush Mkrtchyan, Vegard Molaug

2023-11-03

```
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
```

```
library(ggplot2)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v stringr   1.5.0
## v lubridate 1.9.3     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.0
## v readr     2.1.4
```

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

```
library(lubridate)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(tseries)
```

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

```
library(zoo)
```

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

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

```
library(imputeTS)
```

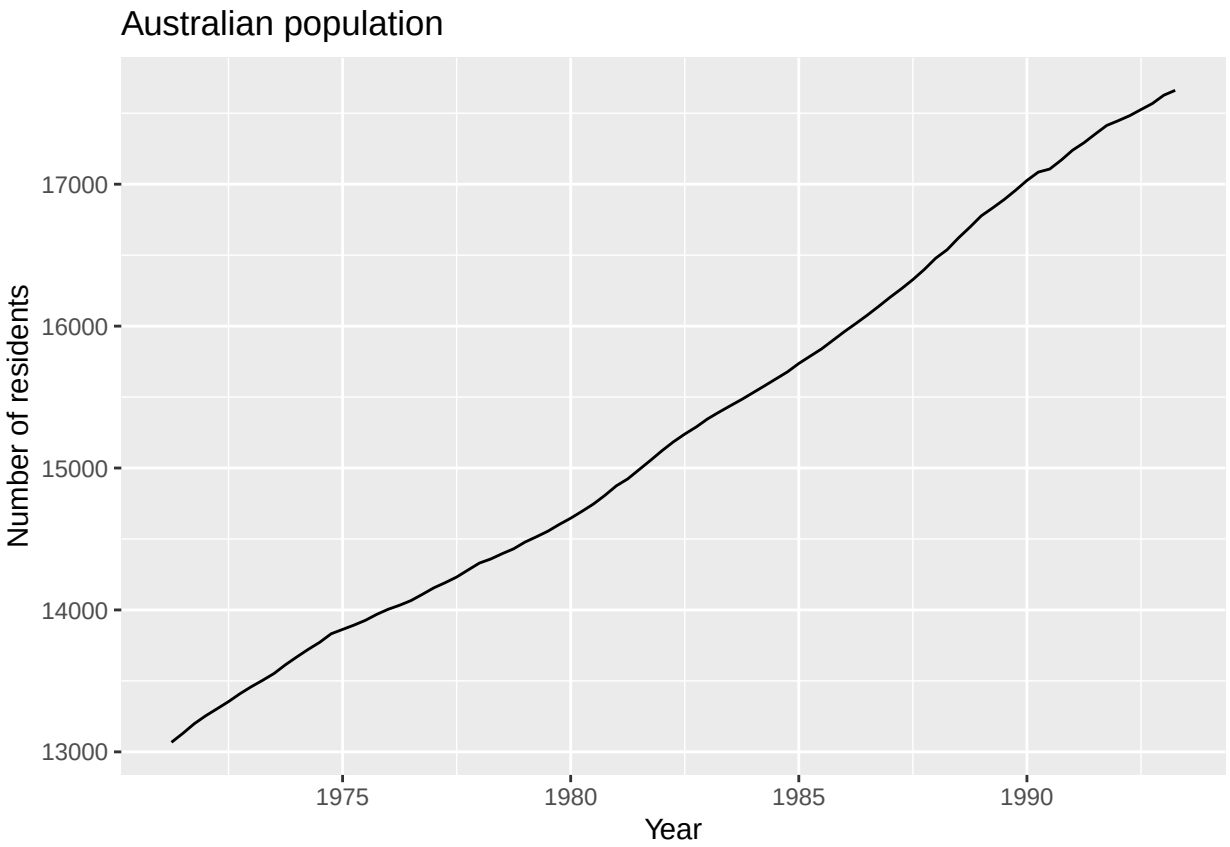
```
##
## Attaching package: 'imputeTS'
##
## The following object is masked from 'package:zoo':
##
##     na.locf
##
## The following object is masked from 'package:tseries':
##
##     na.remove
```

```
library(lmtest)
library(forecast)
library(tseries)
library(datasets)
library(tinytex)
```

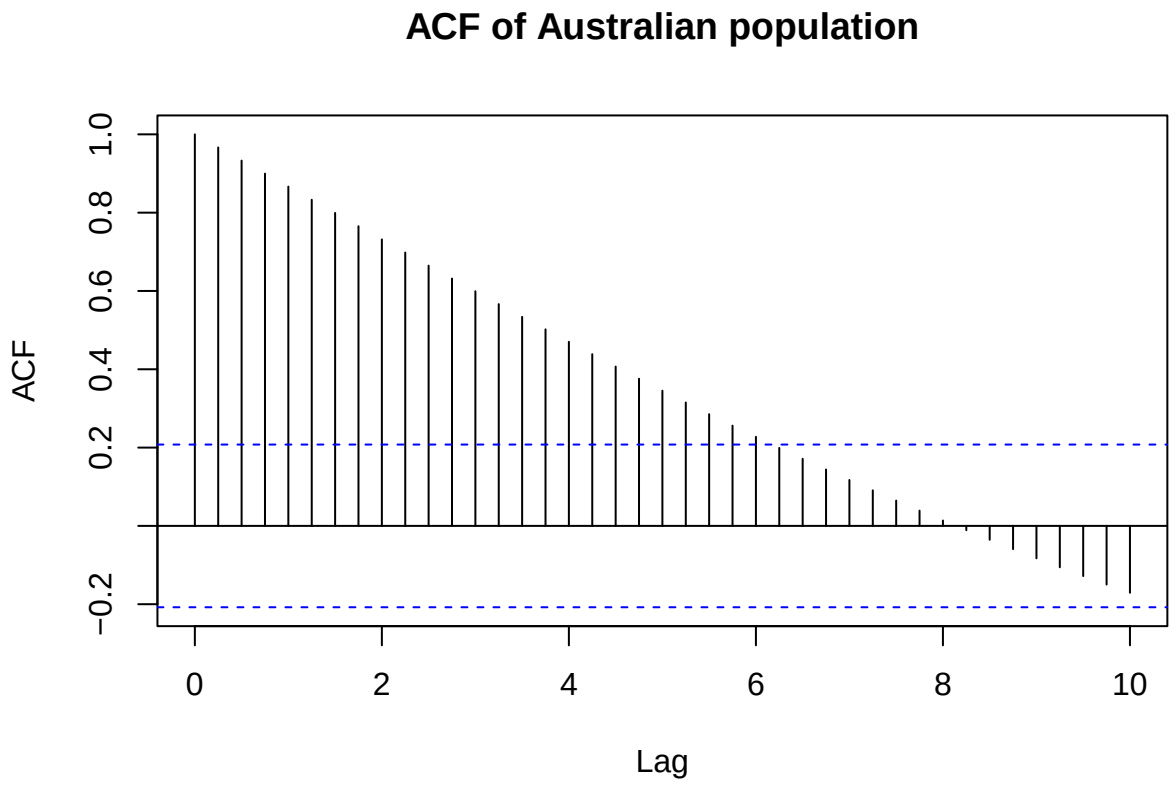
# Exercise 1

## Task a)

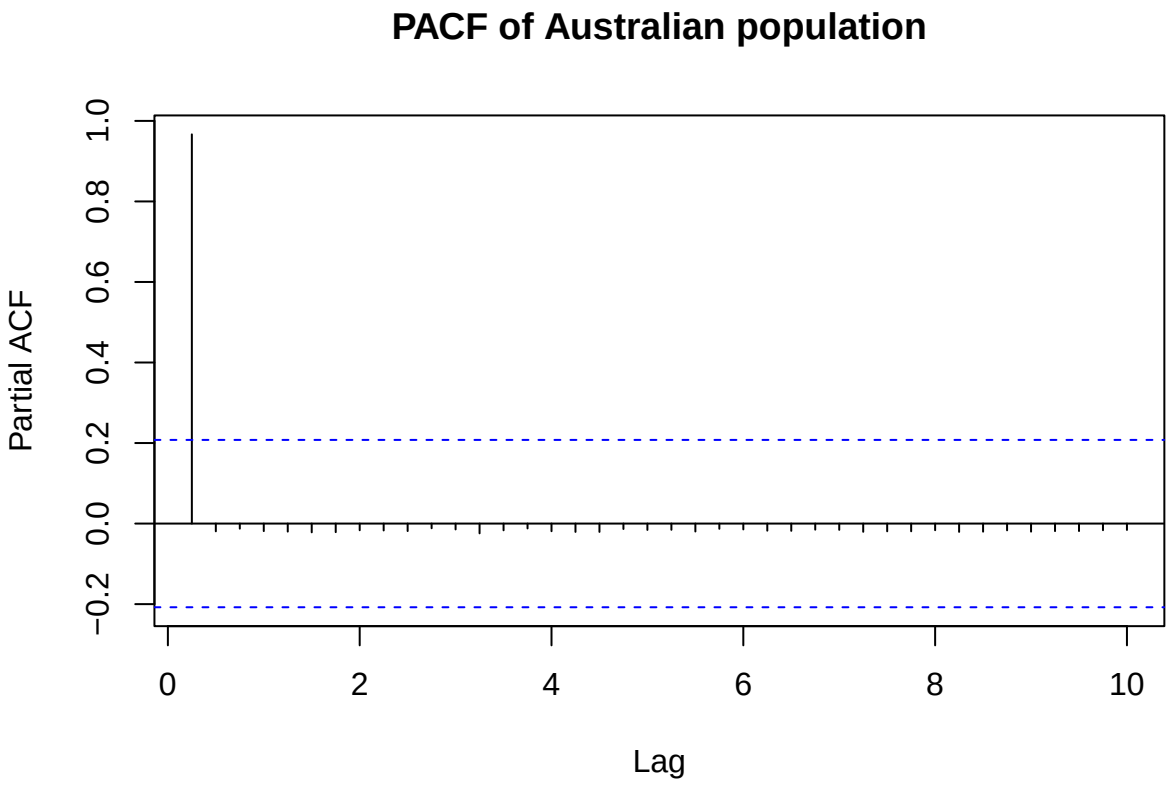
```
data(austres)

autoplot(austres) + ggtitle('Australian population') + ylab('Number of residents') + xlab('Year')
```



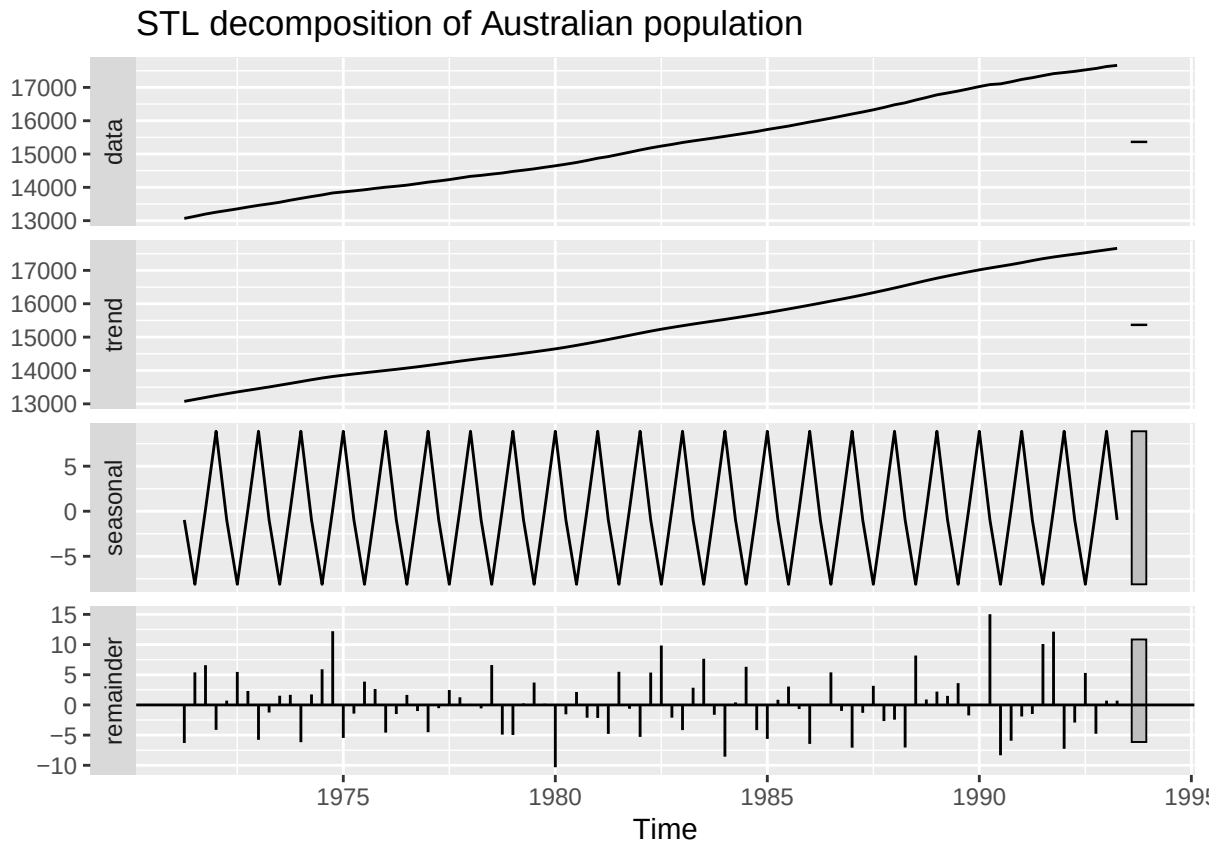

```
acf(austres, main='ACF of Australian population', lag.max = 40)
```




```
pacf(austres, main='PACF of Australian population', lag.max = 40)
```




```
# STL Decomposition
austres_stl <- stl(austres, s.window="periodic")
autoplot(austres_stl) + ggtitle('STL decomposition of Australian population')
```




```
kpss_result <- kpss.test(austres)
```

```
## Warning in kpss.test(austres): p-value smaller than printed p-value
```

```
print(kpss_result)
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  austres
## KPSS Level = 2.3122, Truncation lag parameter = 3, p-value = 0.01
```

There appears to be an upward trend in the number of Australian residents over the years. We can also observe some repeating patterns at regular intervals, indicating seasonality in the data.

The ACF shows a slow decay, which suggests that the series might be non-stationary. This claim is also backed up by the result of the KPSS-test.

The PACF plot shows a significant spike at lag 1 and then tapers off, which is typical for a series with an autoregressive nature. We would therefore suggest an AR term of 1 as a baseline for later analysis. If we also consider the non-stationary nature of the series, one differencing term might be needed.The exact number of MA could be tricky to decide, so we believe a number of zero could be a valid option.

So the arima model we would suggest in this task has a p of 1, d of 1 and q of 0.

## Task b)

```
library(forecast)
```

```r
auto_arima <- function(ts_data, criterion = "AIC") {
  criterion_score <- Inf
  best_p <- best_d <- best_q <- NULL

  for (p in 0:4) {
    for (d in 0:3) {
      for (q in 0:4) {
        # Use tryCatch to handle potential errors
        model_fit <- tryCatch({
          if (d <= 1) {
            Arima(ts_data, order=c(p,d,q), include.mean=TRUE)
          } else {
            Arima(ts_data, order=c(p,d,q), include.mean=FALSE)
          }
        }, error = function(e) NULL)  # Return NULL on error

        # If model fitting was successful, compare AIC/BIC
        if (!is.null(model_fit)) {
          if (criterion == "AIC") {
            current_aic_bic <- model_fit$aic
          } else {
            current_aic_bic <- model_fit$bic
          }

          if (current_aic_bic < criterion_score) {
            criterion_score <- current_aic_bic
            best_p <- p
            best_d <- d
            best_q <- q
          }
        }
      }
    }
  }

  return(list(p=best_p, d=best_d, q=best_q, criterion_score=criterion_score))
}
```

### Task c)

```r
train_data <- window(austres, start=c(1971,2), end=c(1988,3))  # First 70 data points
test_data <- window(austres, start=c(1988,4)) # Remaining 19 data points

results_aic <- auto_arima(train_data, criterion="AIC")
results_bic <- auto_arima(train_data, criterion="BIC")

cat("Model parameters based on AIC:\n")
```

```
## Model parameters based on AIC:
```

```r
print(results_aic)
```

```
## $p
## [1] 3
```

```
##
## $d
## [1] 2
##
## $q
## [1] 2
##
## $criterion_score
## [1] 464.9315
```

```
cat("\nModel parameters based on BIC:\n")
```

```
##
## Model parameters based on BIC:
```

```
print(results_bic)
```

```
## $p
## [1] 3
##
## $d
## [1] 2
##
## $q
## [1] 2
##
## $criterion_score
## [1] 478.2486
```

Interestingly, the results from the hyperparameter search yielded similar results for both BIC and AIC. The parameters here were different than the model we suggested in a), which was a ARIMA(1,1,0).

## Task d)

```
# Model with hyperparameters from AIC
model_aic <- Arima(train_data, order=c(results_aic$p, results_aic$d, results_aic$q))
forecast_aic <- forecast(model_aic, h=19)

# Model with hyperparameters from BIC
model_bic <- Arima(train_data, order=c(results_bic$p, results_bic$d, results_bic$q))
forecast_bic <- forecast(model_bic, h=19)

# Model with suggested hyperparameters from task a)
model_task_a <- Arima(train_data, order=c(1,1,0))
forecast_suggested <- forecast(model_task_a, h=19)

# Average model
model_avg <- meanf(train_data, h=19)

# Drift model
model_drift <- rwf(train_data, h=19, drift=TRUE)

# Naive model
model_naive <- rwf(train_data, h=19)

# Seasonal Naive model
```

```r
model_snaive <- snaive(train_data, h=19)

rmse <- function(predictions, actual) {
  sqrt(mean((predictions - actual)^2))
}

rmse_values <- data.frame(
  Model = c("AIC", "BIC", "Model based on task A", "Average", "Drift", "Naive", "Seasonal Naive"),
  RMSE = c(
    rmse(forecast_aic$mean, test_data),
    rmse(forecast_bic$mean, test_data),
    rmse(forecast_suggested$mean, test_data),
    rmse(model_avg$mean, test_data),
    rmse(model_drift$mean, test_data),
    rmse(model_naive$mean, test_data),
    rmse(model_snaive$mean, test_data)
  )
)

print(rmse_values[order(rmse_values$RMSE), ])
```

```
##                   Model       RMSE
## 5                 Drift   85.41271
## 3 Model based on task A  226.02784
## 1                   AIC  228.27622
## 2                   BIC  228.27622
## 6                 Naive  664.94453
## 7        Seasonal Naive  774.32599
## 4               Average 2490.57216
```

Considering the RMSE score, the drift model performed the best. Surprisingly, the parameters we suggested in a) performed slightly better than the optimized parameters from task b). For this dataset, the average and naive models do not appear to be as effective as the other models in this task.

# Exercise 2

# Exercise 3

## Task a)

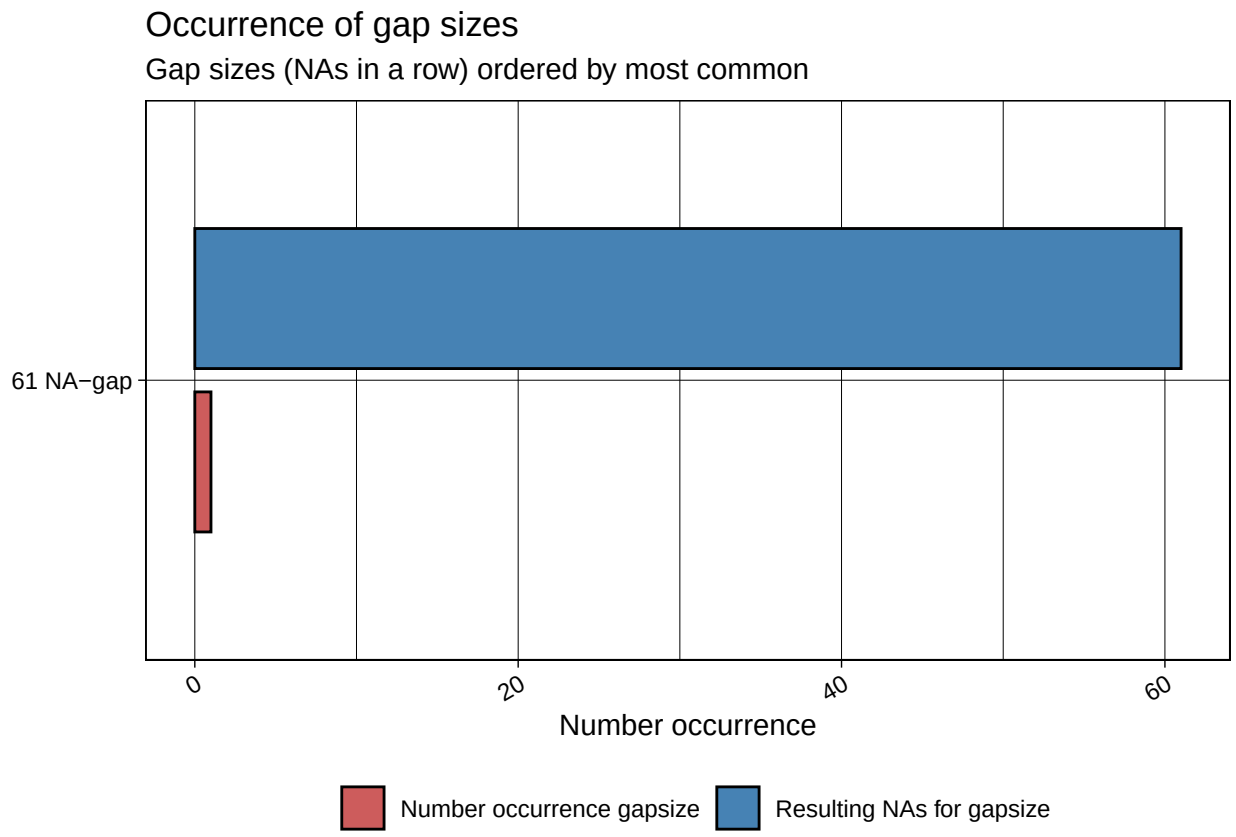
```r
watershed <- read.csv('watershed.csv')
head(watershed)
```

```
##         date hydr_year     gauge        rain snow     temp
## 1 1979-10-01         1 0.3042305 1.362244670    0 12.65813
## 2 1979-10-02         1 0.2866576 0.047807657    0 12.29846
## 3 1979-10-03         1 0.2661559 0.092385567    0 12.92369
## 4 1979-10-04         1 0.2489492 0.034734455    0 12.96524
## 5 1979-10-05         1 0.2383322 0.000127301    0 12.86054
## 6 1979-10-06         1 0.2313763 0.004703795    0 13.08902
```

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

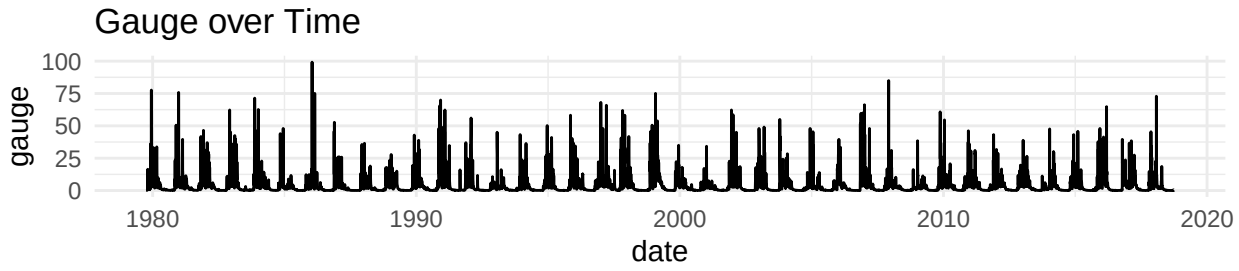
```
##       date hydr_year      gauge       rain       snow       temp
##          0          0         61          0          0          0
```

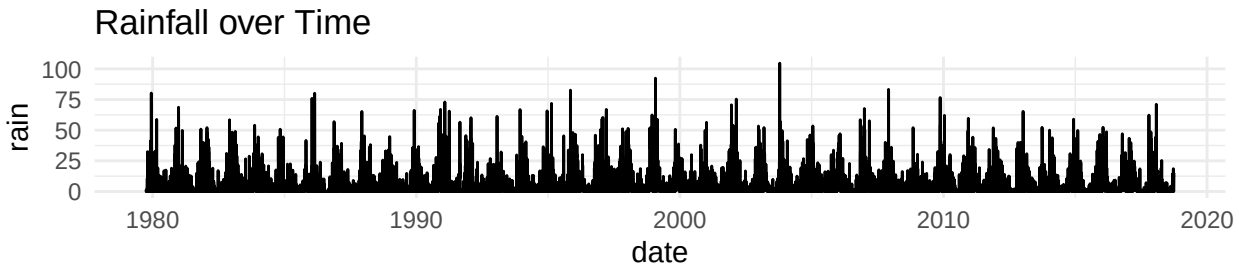
```
ggplot_na_gapsize(watershed$gauge)
```

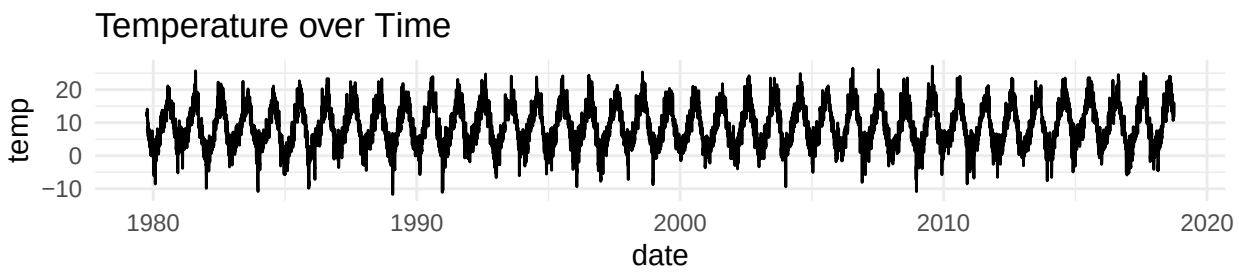



```
# We remove the missing values by weighted moving average imputation
watershed <- watershed %>%
  mutate(gauge = na_ma(gauge))
```

```
watershed$date <- as.Date(watershed$date, format='%Y-%m-%d')

# Plotting the variables gauge, rain, and temp over time
p1 <- ggplot(watershed, aes(x=date, y=gauge)) + geom_line() +
  ggtitle('Gauge over Time') + theme_minimal()
p2 <- ggplot(watershed, aes(x=date, y=rain)) + geom_line() +
  ggtitle('Rainfall over Time') + theme_minimal()
p3 <- ggplot(watershed, aes(x=date, y=temp)) + geom_line() +
  ggtitle('Temperature over Time') + theme_minimal()

grid.arrange(p1, p2, p3)
```




```
kpss_gauge <- kpss.test(watershed$gauge)
```

```
## Warning in kpss.test(watershed$gauge): p-value greater than printed p-value
```

```
kpss_rain <- kpss.test(watershed$rain)
```

```
## Warning in kpss.test(watershed$rain): p-value greater than printed p-value
```

```
kpss_temp <- kpss.test(watershed$temp)

list(kpss_gauge, kpss_rain, kpss_temp)
```

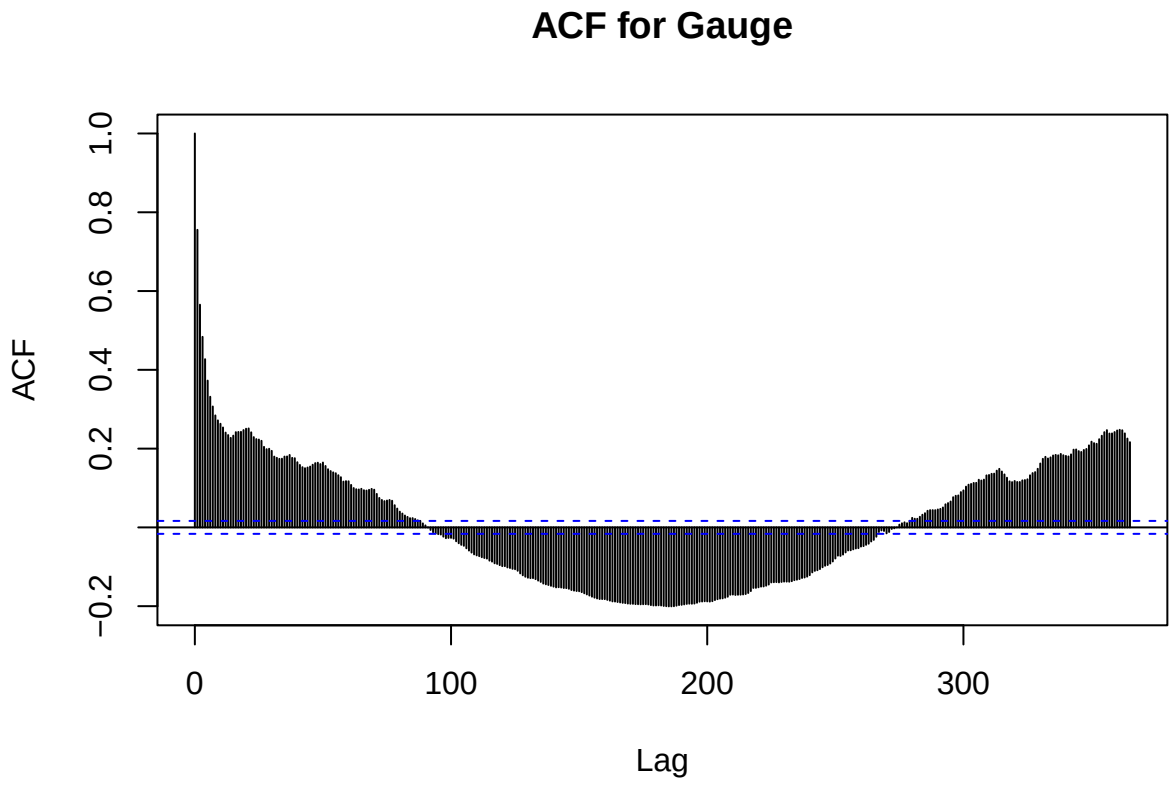
```
## [[1]]
##
##  KPSS Test for Level Stationarity
##
## data:  watershed$gauge
## KPSS Level = 0.078678, Truncation lag parameter = 13, p-value = 0.1
##
##
## [[2]]
##
##  KPSS Test for Level Stationarity
##
## data:  watershed$rain
## KPSS Level = 0.067964, Truncation lag parameter = 13, p-value = 0.1
##
##
```

```
## [[3]]
##
##  KPSS Test for Level Stationarity
##
## data:  watershed$temp
## KPSS Level = 0.42809, Truncation lag parameter = 13, p-value = 0.06505
```

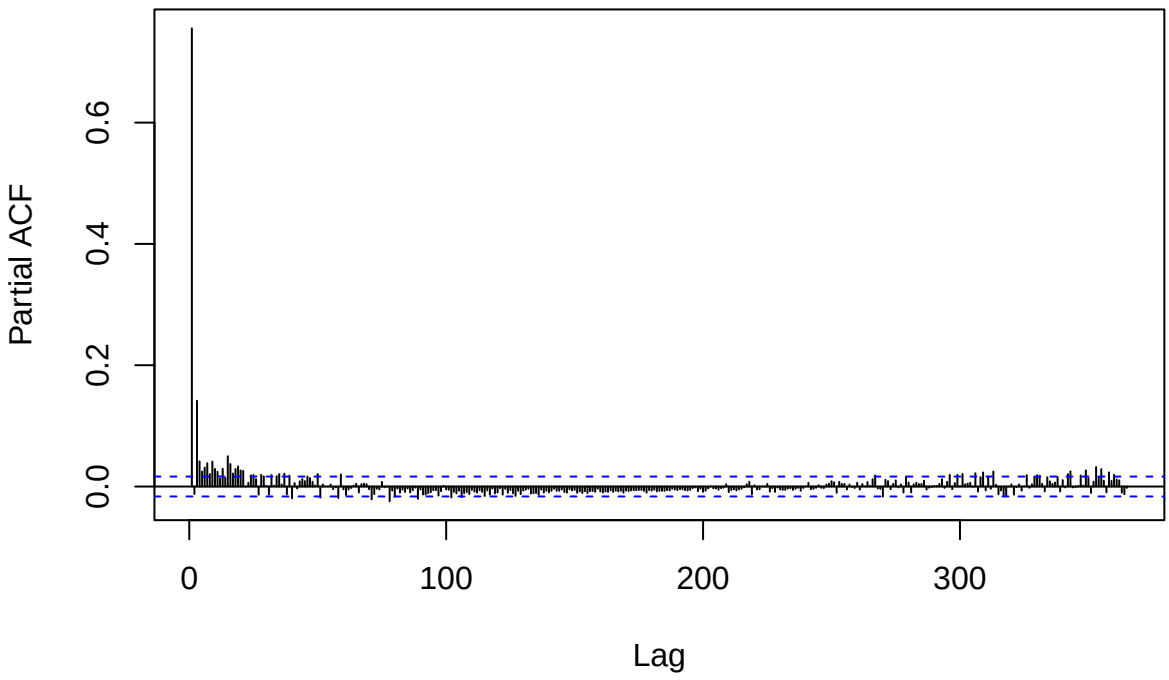
We can observe that the KPSS test return a p-value of 0.1 for the columns rain and gauge. Therefore, we can not reject the null hypothesis for these columns, which means that they appear to be stationary. Temp has a p-value of 0.6505, which is close to the given significance level of 0.05. We can also assume that temp is stationary, but we can not assert this with the same confidence as the other variables in the dataset.

### Task b)

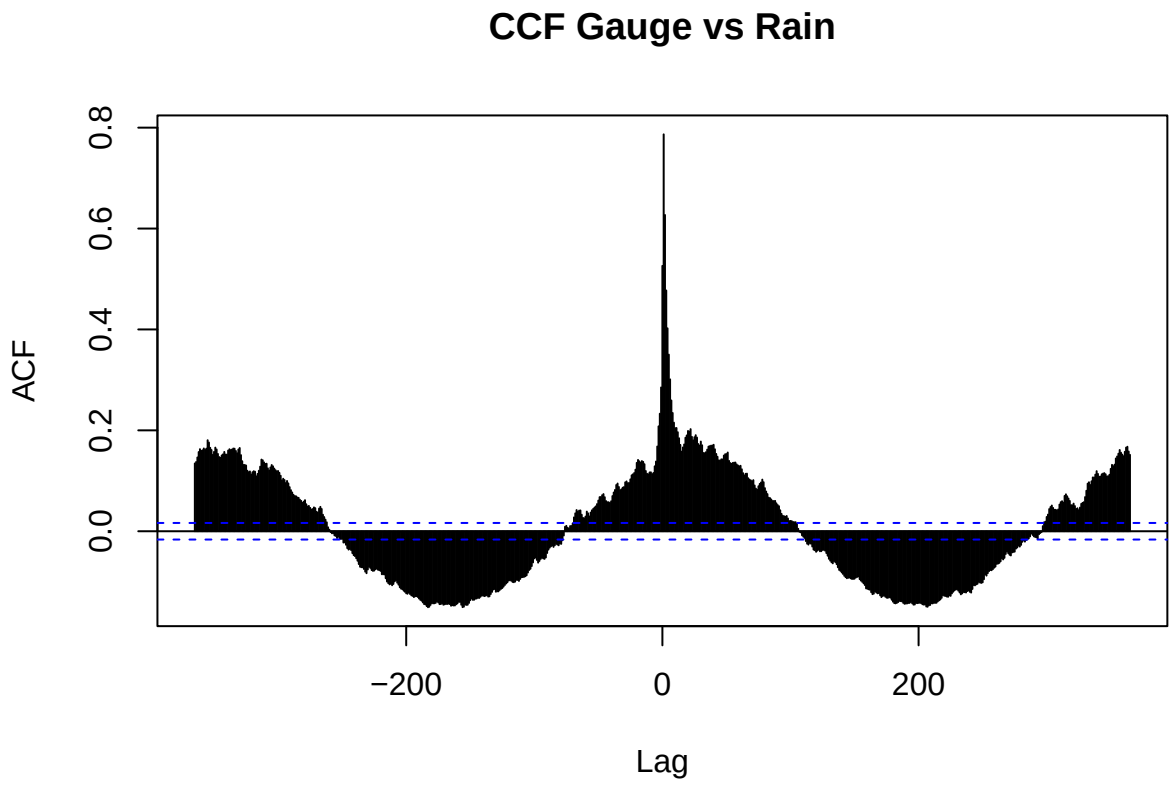
```r
# ACF, PACF and CCF for gauge
acf(watershed$gauge, main='ACF for Gauge', lag.max = 365)
```



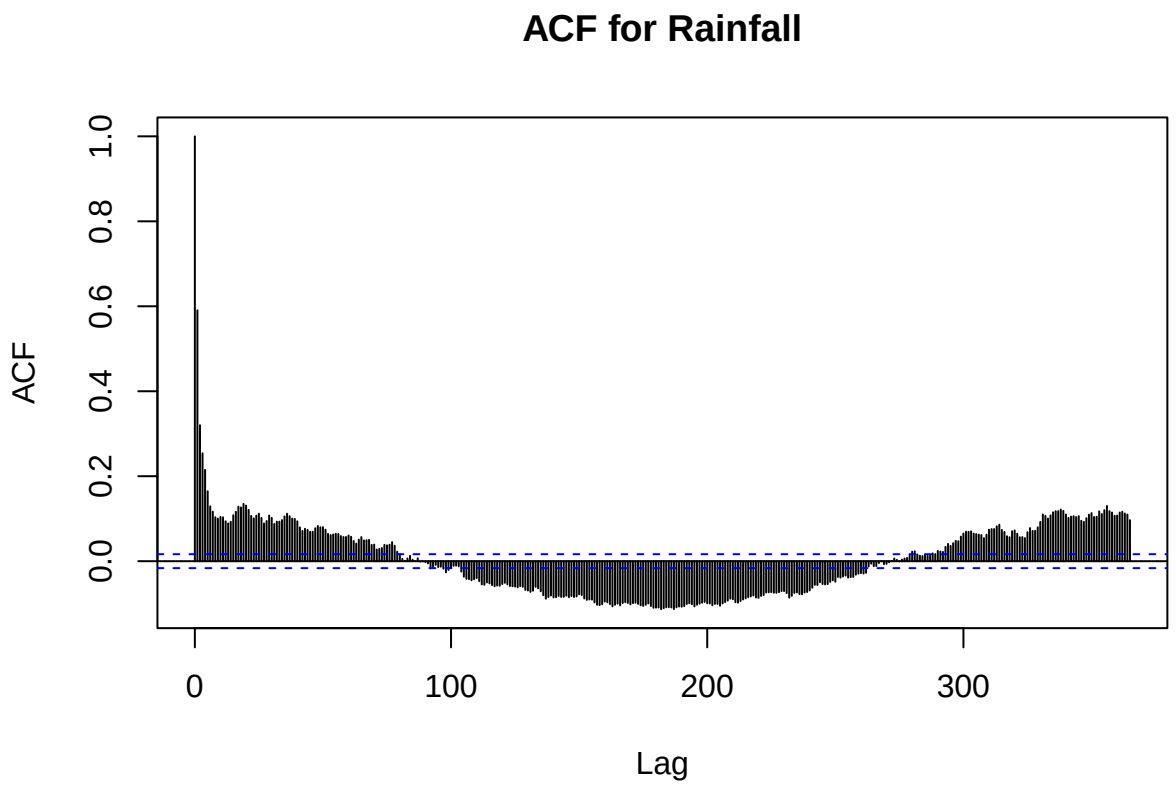

```r
pacf(watershed$gauge, main='PACF for Gauge', lag.max = 365)
```



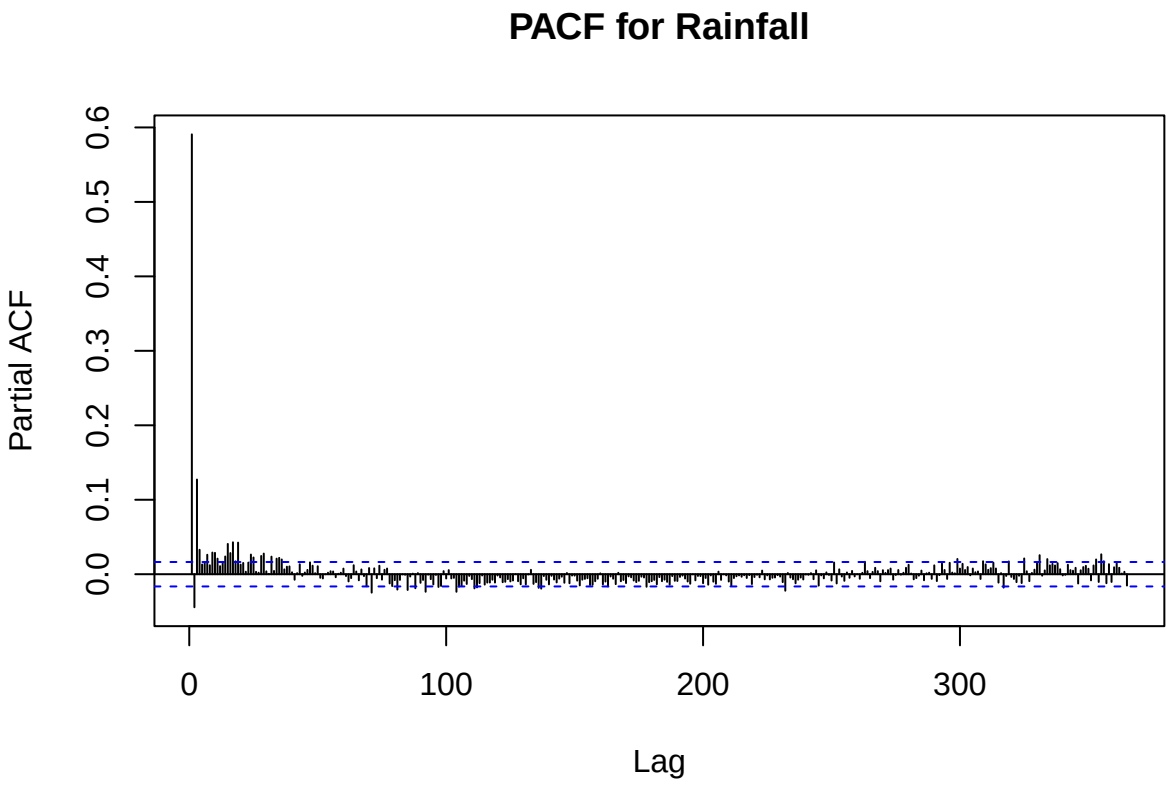

```
ccf(watershed$gauge, watershed$rain, main='CCF Gauge vs Rain', lag.max = 365)
```



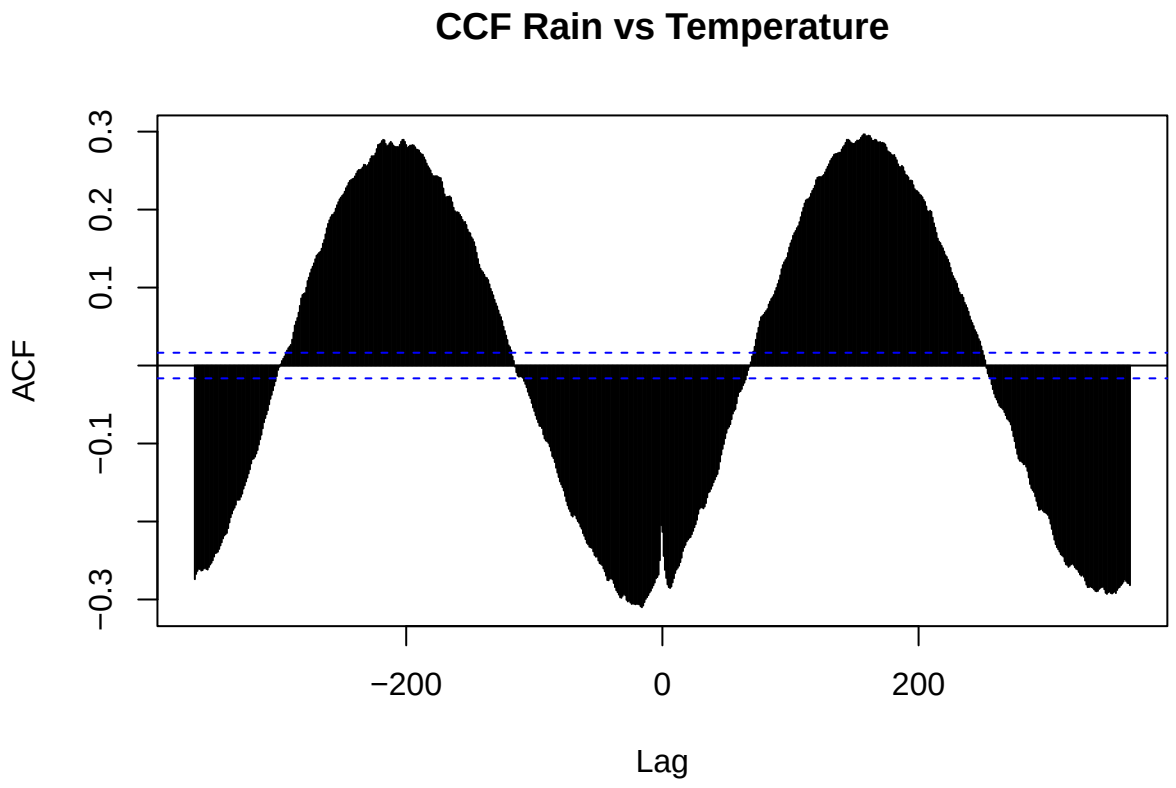

```
# ACF, PACF and CCF for rain
acf(watershed$rain, main='ACF for Rainfall', lag.max = 365)
```



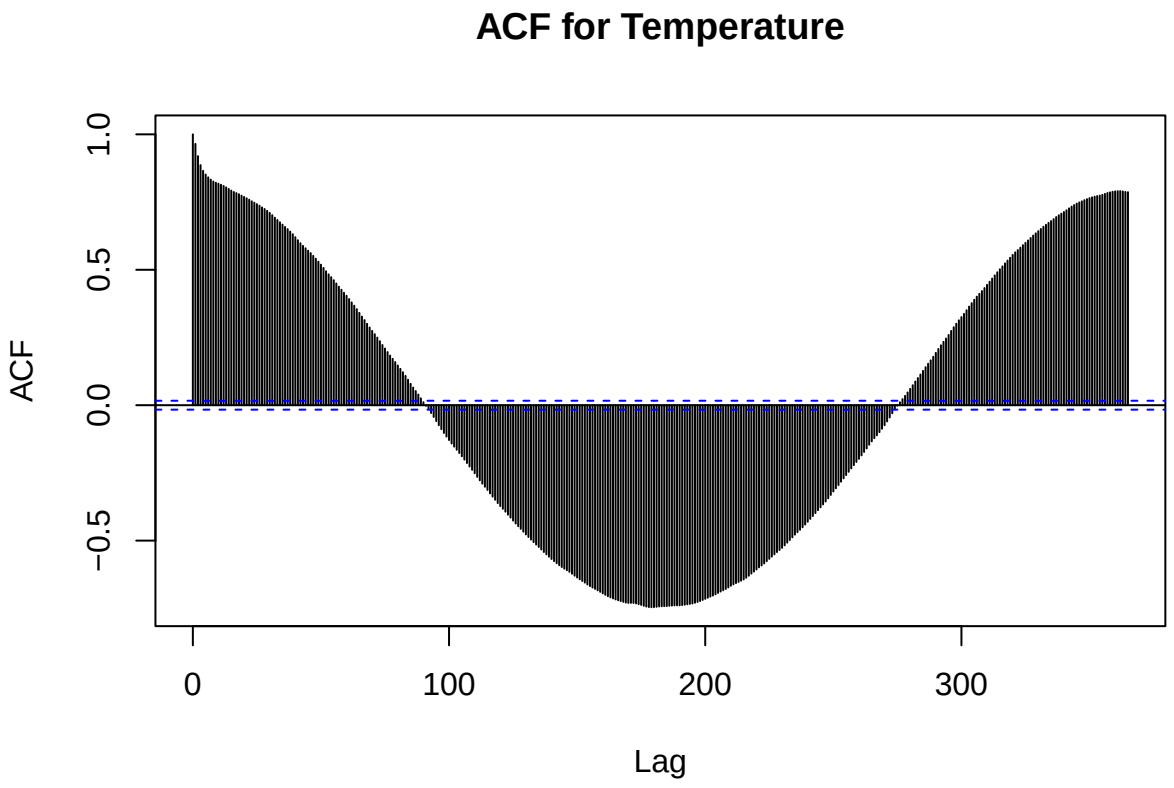

```
pacf(watershed$rain, main='PACF for Rainfall', lag.max = 365)
```



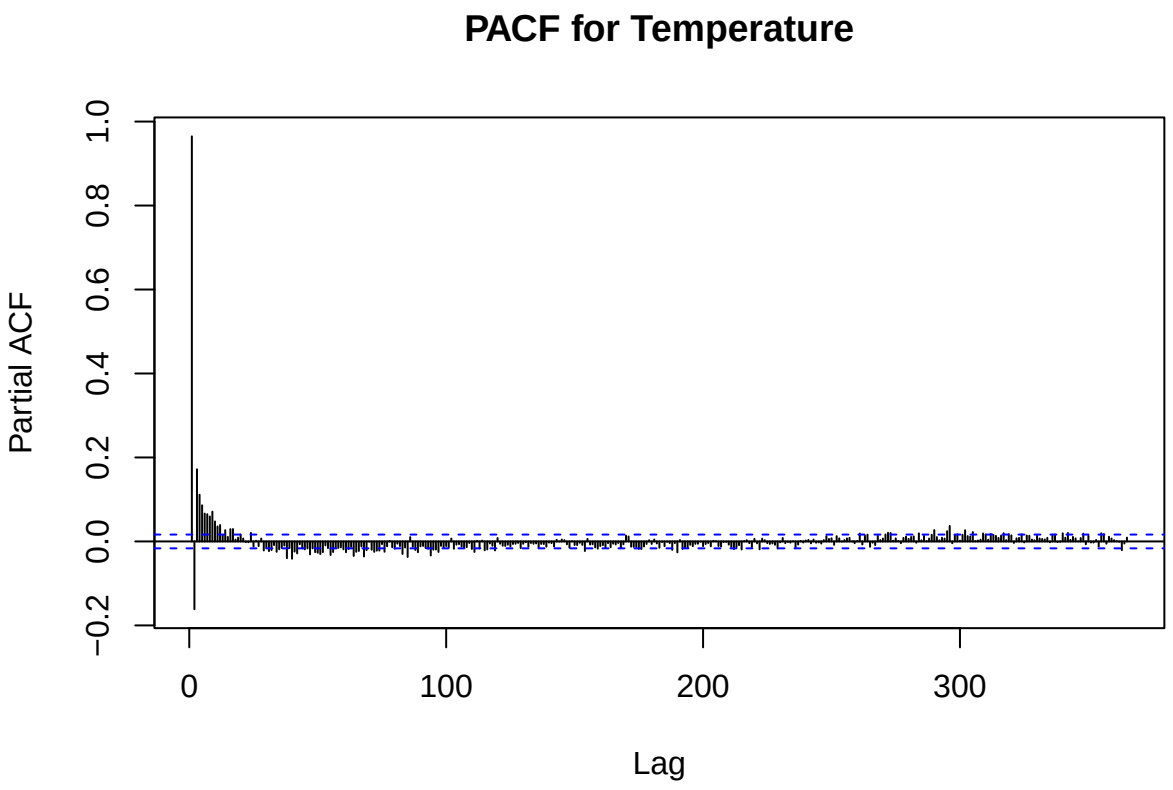

```
ccf(watershed$rain, watershed$temp, main='CCF Rain vs Temperature', lag.max = 365)
```



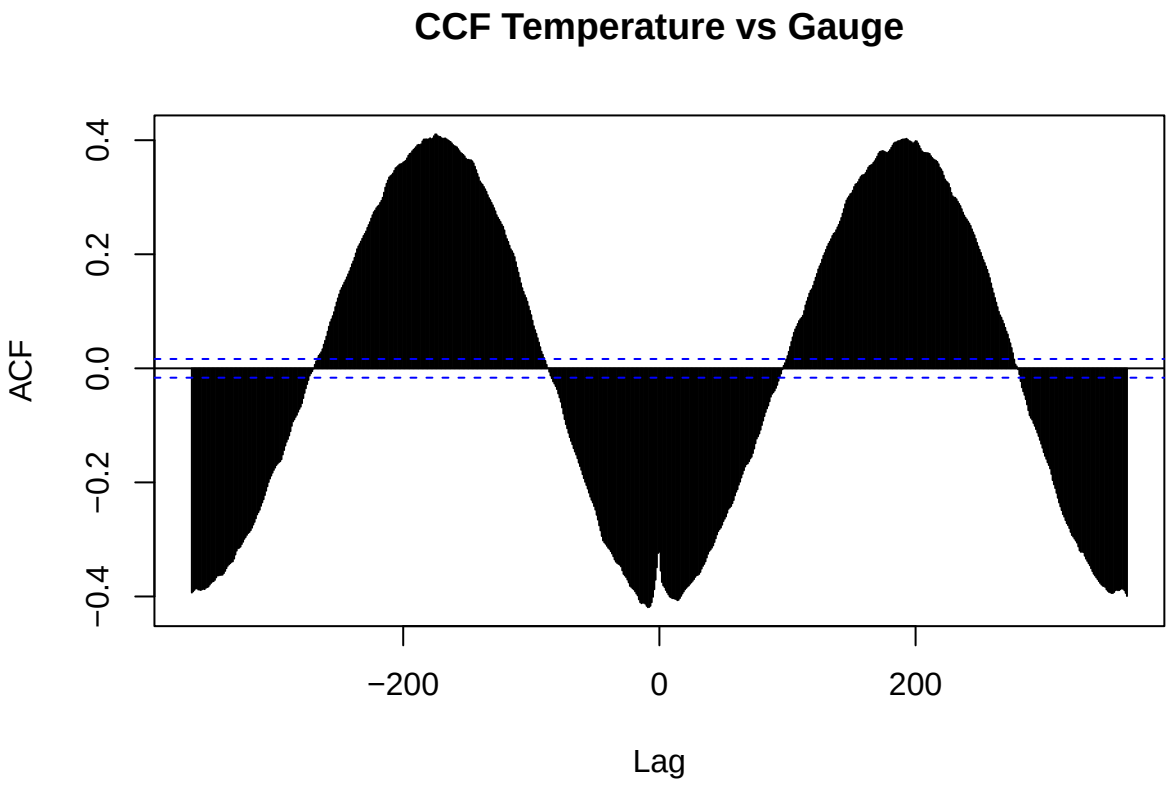

```
# ACF, PACF and CCF for temp
acf(watershed$temp, main='ACF for Temperature', lag.max = 365)
```




```
pacf(watershed$temp, main='PACF for Temperature', lag.max = 365)
```




```
ccf(watershed$temp, watershed$gauge, main='CCF Temperature vs Gauge', lag.max = 365)
```




The season of the year appears to be the common denominator for the variables in the dataset. All variables have roughly similar patterns for the chosen interval of lags.

```
# Granger test with gauge and rain
granger_gauge_rain <- grangertest(gauge ~ rain, data = watershed, order = 1)
print(granger_gauge_rain)
```

```
## Granger causality test
##
## Model 1: gauge ~ Lags(gauge, 1:1) + Lags(rain, 1:1)
## Model 2: gauge ~ Lags(gauge, 1:1)
##   Res.Df Df     F    Pr(>F)
## 1  14241
## 2  14242 -1 13579 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Granger test with gauge and temp
granger_gauge_temp <- grangertest(gauge ~ temp, data = watershed, order = 1)
print(granger_gauge_temp)
```

```
## Granger causality test
##
## Model 1: gauge ~ Lags(gauge, 1:1) + Lags(temp, 1:1)
## Model 2: gauge ~ Lags(gauge, 1:1)
##   Res.Df Df      F    Pr(>F)
## 1  14241
## 2  14242 -1 245.76 < 2.2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

**Results for Gauge and Rain**: Model 1: gauge is predicted by its own past values and past values of rain. Model 2: gauge is only predicted by its own past values.

The F value for the rain variable is 13579 and the p-value is $< 2.2e\text{-}16$. The low p-value indicates that adding the past values of rain improves predictions of gauge compared to using past values of gauge alone.

**Results for Gauge and Temp**: Model 1: gauge is predicted by its own past values and the past values of temp. Model 2: gauge is only predicted by its own past values.

The F value for the temp variable is 245.76 and the p-value is $< 2.2e\text{-}16$.The low p-value suggests that adding the past values of temp significantly improves the prediction of gauge compared to using the past values of gauge alone.

Overall, we can assume that past values of rain and temp can provide additional information in predicting the current value of gauge beyond what the past values of gauge can provide.

## Task c)

```r
train_data <- watershed[1:(30*365),]
train_data <- train_data %>%
  mutate(lagged_rain_1 = lag(rain, 1),
         lagged_rain_2 = lag(rain, 2),
         lagged_rain_3 = lag(rain, 3))

train_data <- na.omit(train_data)

test_data <- watershed[(30*365 + 1):nrow(watershed),]
test_data <- test_data %>%
  mutate(lagged_rain_1 = lag(rain, 1),
         lagged_rain_2 = lag(rain, 2),
         lagged_rain_3 = lag(rain, 3))

test_data <- na.omit(test_data)

# Linear regression model with rain as the predictor
linear_model <- lm(gauge ~ rain, data=train_data)

# Distributed lag model
dlm <- lm(gauge ~ rain + lagged_rain_1 + lagged_rain_2 + lagged_rain_3, data=train_data)

# ARIMA model
arima_model <- auto.arima(train_data$gauge)

# Dynamic regression model with rain as the predictor
dynamic_reg <- auto.arima(train_data$gauge, xreg = train_data$rain)

# Dynamic regression model with lagged rain as predictors
dynamic_reg_lagged <- auto.arima(train_data$gauge, xreg = as.matrix(train_data[, c("lagged_rain_1", "lag
```

```r
# Predictions for linear regression model
linear_pred <- predict(linear_model, newdata=test_data)

# Predictions for distributed lag model
```

```r
dlm_pred <- predict(dlm, newdata = test_data)

# Predictions for ARIMA model
arima_pred <- forecast(arima_model, h = nrow(test_data))$mean

# Predictions for dynamic regression model
dynamic_reg_pred <- forecast(dynamic_reg, xreg = test_data$rain)$mean

# Convert lagged variables to a matrix for dynamic regression with lagged predictors
test_lagged_matrix <- as.matrix(test_data[, c("lagged_rain_1", "lagged_rain_2", "lagged_rain_3")])

# Predictions for dynamic regression model with lagged predictors
test_lagged_matrix <- as.matrix(test_data[, c("lagged_rain_1", "lagged_rain_2", "lagged_rain_3")])
dynamic_reg_lagged_pred <- forecast(dynamic_reg_lagged, xreg = test_lagged_matrix)$mean
```

```r
rmse <- function(actual, predicted) {
  sqrt(mean((actual - predicted)^2))
}

rsquared <- function(actual, predicted) {
  1 - (sum((actual - predicted)^2) / sum((actual - mean(actual))^2))
}

models <- list(linear_pred, dlm_pred, arima_pred, dynamic_reg_pred, dynamic_reg_lagged_pred)
names(models) <- c("Linear regression", "Distributed Lag", "ARIMA", "Dynamic Regression", "Dynamic Regr

rmse_values <- sapply(models, function(pred) rmse(test_data$gauge, pred))
rsquared_values <- sapply(models, function(pred) rsquared(test_data$gauge, pred))

performance_table <- data.frame(Model = names(models), RMSE = rmse_values, R2 = rsquared_values)
print(performance_table)
```

```
##                                                 Model     RMSE           R2
## Linear regression                   Linear regression 5.355572  0.287104425
## Distributed Lag                       Distributed Lag 3.528897  0.690477541
## ARIMA                                           ARIMA 6.370482 -0.008692531
## Dynamic Regression                 Dynamic Regression 6.222818  0.037527267
## Dynamic Regression Lagged Dynamic Regression Lagged 3.580769  0.681311182
```

### Task d)

The lagged versions of dynamic regression and linear regression performed significantly better than the other models for predictions on the test set. This applies to scores for both RMSE and R2, which makes these models better for a practical problem setup. This is likely because the lagged models take into account both the current rainfall and the effect of rainfall from previous days. These factors can be crucial when modeling phenomena like runoff where the effect of rain might be distributed over several days. With a slightly better score for RMSE and R2, the lagged linear regression model appear to be best suited for this problem.

## Exercise 4

### Task a)

The model equations are as follows:

***Linear Regression***: $y_i = x_i\beta + \varepsilon_i$, (1)
***Autoregressive Error***: $\varepsilon_i = \phi_1\varepsilon_{i-1} + \eta_i$, (2)
***Re-estimate the parameters ($\hat{\beta}$)***: $\eta_i \sim_{\text{iid}} \mathrm{N}(0, \sigma^2)$, (3)

**Transformation in step c)**

We want to start from the equation $\tilde{y}_i = y_i - \phi_1 y_{i-1}$ and derive the equation $\tilde{x}_i = x_i - \phi_1 x_{i-1}$, respectively, for $i > 1$.

We start with the transformed data:

$$\tilde{y}_i = y_i - \phi_1 y_{i-1}$$

Now, we want to derive an equation for $\tilde{x}_i$, so we need to express $y_i$ in terms of $x_i$ using the original linear regression equation:

$$y_i = x_i\beta + \epsilon_i$$

Plug this into the transformation for $\tilde{y}_i$:

$$\tilde{y}_i = (x_i\beta + \epsilon_i) - \phi_1(x_{i-1}\beta + \epsilon_{i-1})$$

Now, expand and simplify:

$$\tilde{x}_i\beta = x_i\beta + \epsilon_i - \phi_1 x_{i-1}\beta - \phi_1\epsilon_{i-1}$$

The equation for $\tilde{y}_i$ does not directly involve the error term $\epsilon_i$, as the transformation is applied to the observed values of $y_i$ and $y_{i-1}$ to remove the autoregressive effect. The error term is still present in the original linear regression equation but is not explicitly included in the transformation itself.

The transformation is designed to create a new set of variables $\tilde{y}_i$ and $\tilde{x}_i$ that no longer have the autoregressive structure. The error term $\epsilon_i$ remains in the original equation, but it is not explicitly factored into the transformation because the goal is to eliminate the autoregressive component in the transformed variables.

Now, factor out $\beta$:

$$\tilde{x}_i\beta = \beta(x_i - \phi_1 x_{i-1}) + \epsilon_i - \phi_1\epsilon_{i-1}$$

Divide both sides by $\beta$:

$$\tilde{x}_i = x_i - \phi_1 x_{i-1} + \frac{\epsilon_i - \phi_1\epsilon_{i-1}}{\beta}$$

Now, if we assume that the errors $\epsilon_i$ and $\epsilon_{i-1}$ are independently normally distributed with mean 0, and $\beta$ is a constant, then we can define $\eta_i$ as the term in the denominator:

$$\eta_i = \frac{\epsilon_i - \phi_1\epsilon_{i-1}}{\beta}$$

Substituting this into the equation:

$$\tilde{x}_i = x_i - \phi_1 x_{i-1} + \eta_i$$

### Uncorrelated errors

A regression model on the transformed predictors $\tilde{x}_i$ and target variable $\tilde{y}_i$ has uncorrelated errors due to the removal of the autoregressive structure. Assuming correct estimation of $\phi_1$, the transformation eliminates the autocorrelation, leading to independent errors in the transformed model. This independence allows for reliable OLS parameter estimation.

## Task b)

**Is the parameter vector $\beta$ the same in the linear regression model on the transformed predictors as in the linear regression model trained on the original predictors?**

In a linear regression model, the parameter vector $\beta$ may not be the same in the model on transformed predictors as in the model trained on the original predictors. The transformation introduces new variables and changes the relationship between predictors and the target variable.

The parameter vector $\beta$ in the transformed model reflects the relationship between the transformed predictors ($\tilde{x}_i$) and the transformed target variable ($\tilde{y}_i$), which may differ from the relationship in the original model ($x_i$ and $y_i$).

While the transformation aims to make the errors uncorrelated and enable valid parameter estimation, the estimated $\beta$ coefficients in the transformed model are typically different from those in the original model due to the changes introduced by the transformation.

**Is the same concept applicable for general AR(k) error terms?**

The concept of transforming data to remove autocorrelation and obtaining uncorrelated errors is applicable not only to AR(1) models but also to general autoregressive models with AR(k) error terms, where $k$ represents the order of the autoregressive process. It's crucial to correctly estimate the autoregressive parameters (e.g., $\phi_1, \phi_2, \ldots, \phi_k$) to determine the appropriate transformation.

**Can you think of any practical limitations associated with the described procedure?**

# usikker om rett

(direct estimation via OLS is possible, but may be affected by multicolliearity)

**Assumption of No Intercep**t: The procedure assumes that the model does not have an intercept. In practical applications, an intercept term may be necessary to account for the mean or baseline level of the target variable, and not including it could result in biased parameter estimates.

**Validity of AR(1) Assumption**: The accuracy of the procedure heavily depends on correctly estimating the autoregressive parameter ($\phi_1$. If the true data-generating process doesn't conform to an AR(1) structure, the procedure may lead to inaccurate parameter estimates and, consequently, uncorrelated residuals. Ensuring the appropriateness of the AR(1) assumption is paramount for the model's success.

**Data Stationarity**: AR models assume that the data is stationary, meaning that statistical properties like mean and variance remain constant over time. In practice, achieving stationarity can be challenging, and the model may not perform well with non-stationary data. Data preprocessing techniques may be necessary to make the data stationary.

**Model Order Selection**: Selecting the appropriate order for the AR model (e.g., AR(1), AR(2), etc.) is crucial. Choosing an incorrect order can lead to poor model performance. Determining the optimal order often involves trial and error or more sophisticated model selection techniques.

**Assumption of Linearity**: AR models assume a linear relationship between the past observations and the current observation. If the true relationship is nonlinear, the model may not capture the underlying patterns effectively.

**Lack of Causality Information**: AR models are purely data-driven and do not inherently capture causality. While they can identify temporal dependencies, they may not distinguish between causative relationships and spurious correlations in the data.

**Assumption of Gaussian Errors**: AR models often assume that the errors follow a Gaussian distribution. If this assumption is violated, the model's accuracy may be compromised. Robust modeling approaches may be required for non-Gaussian data.

## Task c)

**Is the same concept applicable if the model contains an intercept? If yes, does the relation between the parameters $\beta$ in the transformed and the original space still hold if we have a model with intercept? Justify based on the model formulas!**

The same concept of applying an autoregressive transformation to remove autocorrelation in the errors can be applied when the model contains an intercept. However, when an intercept is included in the model, the relation between the parameters $\beta$ in the transformed and the original space may be slightly different. We can justify this based on the model formulas.

In a model with an intercept, the original linear regression equation is given by:

$$y_i = \beta_0 + x_i'\beta + \epsilon_i$$

Where:
- $\beta_0$ is the intercept term.
- $x_i'$ is the vector of predictors for the $i$th observation.
- $\beta$ is the vector of coefficients for the predictors.
- $\epsilon_i$ is the error term for the $i$th observation.

Now, if we apply the autoregressive transformation to both the target variable and the predictors, as described earlier, we obtain the transformed equations:

$$\tilde{y}_i = y_i - \phi_1 y_{i-1}$$

$$\tilde{x}_i = x_i - \phi_1 x_{i-1}$$

In the transformed space, there's still a relationship between the parameters $\beta$ and $\tilde{\beta}$ (the transformed parameters). The relationship can be derived from the original and transformed equations as follows:

$$\tilde{y}_i = \beta_0 + \tilde{x}_i'\tilde{\beta} + \epsilon_i - \phi_1(\beta_0 + \tilde{x}_{i-1}'\tilde{\beta} + \epsilon_{i-1})$$

Simplifying this equation, we get:

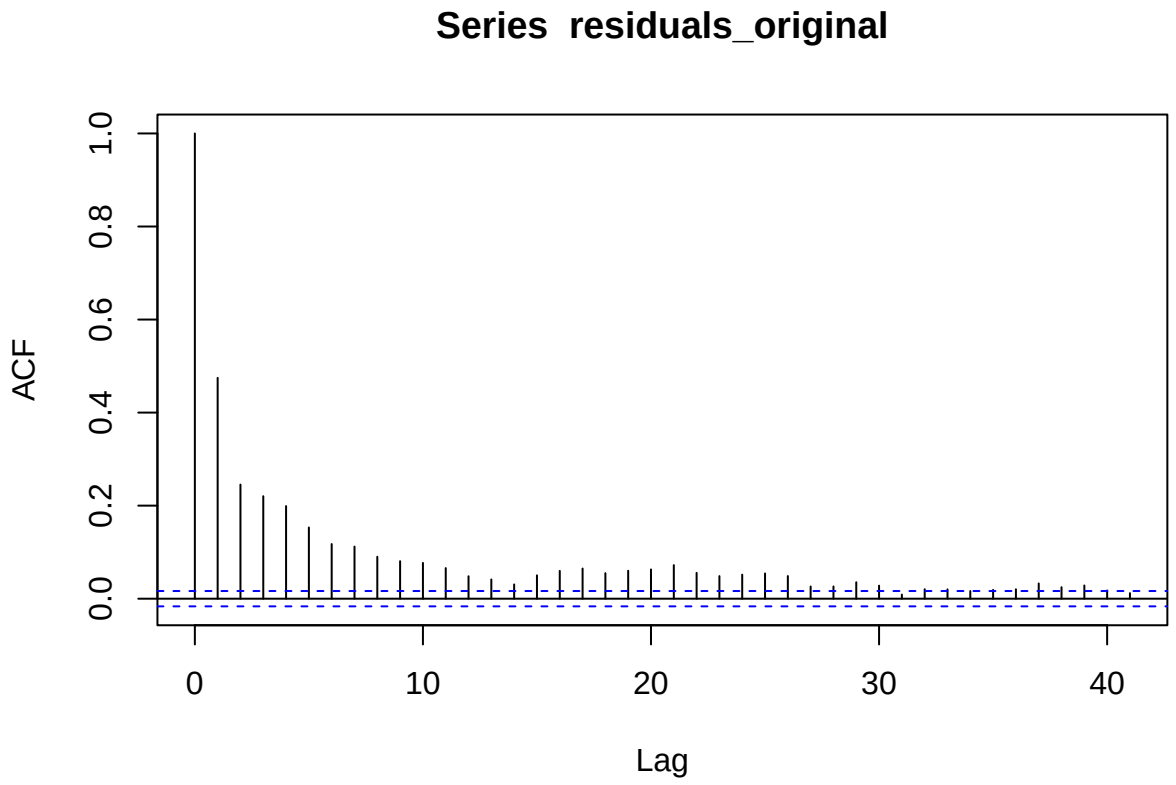
$$\tilde{y}_i = (\beta_0 - \phi_1\beta_0) + \tilde{x}_i'\tilde{\beta} - \phi_1\tilde{x}_{i-1}'\tilde{\beta} + (\epsilon_i - \phi_1\epsilon_{i-1})$$

We can see that the relation between the coefficients of the predictors ($\beta$) and the transformed coefficients ($\tilde{\beta}$) still holds in the presence of an intercept. The transformation does not alter the relationship between the coefficients and their transformed counterparts.

In summary, the concept of removing autocorrelation through an autoregressive transformation can be applied to models with an intercept, and the relationship between the parameters $\beta$ and $\tilde{\beta}$ remains consistent, with the transformation applied to both the intercept and the predictors.

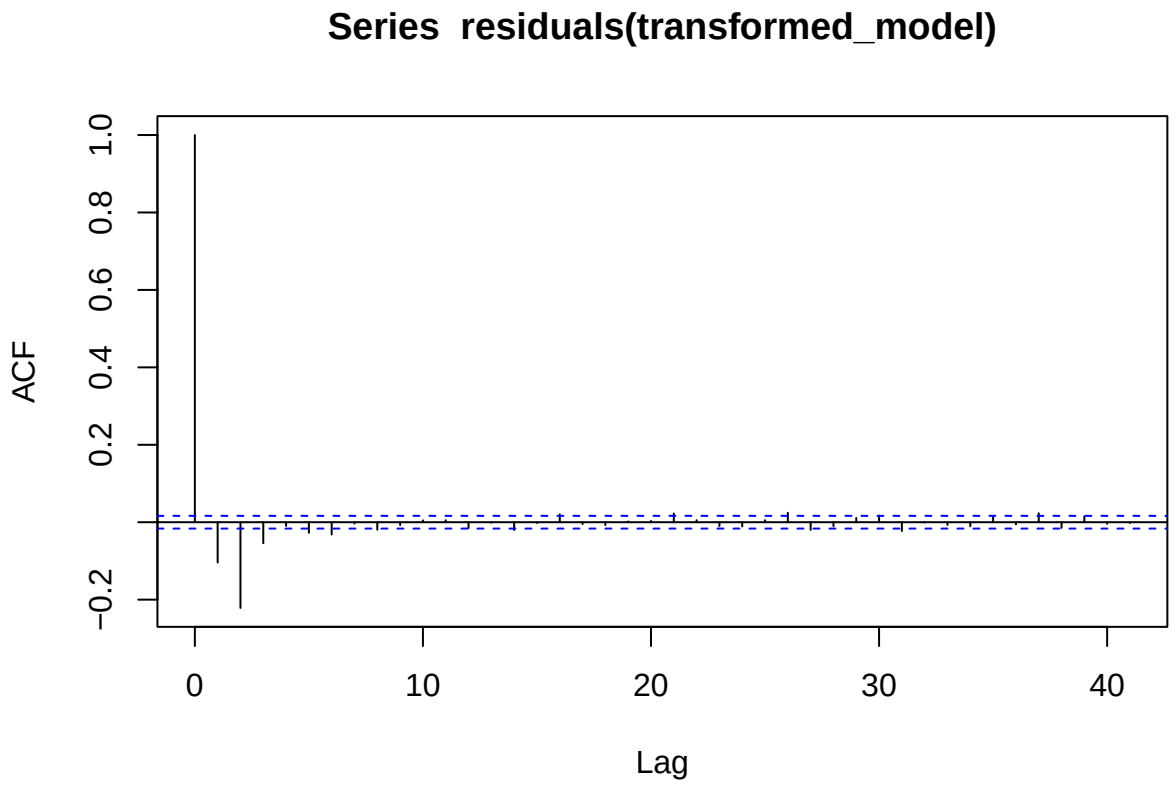
### Task d)

```r
# Make sure to replace 'data.csv' with the actual file path
watershed_data <- read.csv("watershed.csv")

water = watershed_data[, c("gauge", "temp", "snow", "rain")]

# Fit the original linear regression model to obtain initial parameter estimates
original_model <- lm(gauge ~ temp + rain + snow,
                     data = water)

# Calculate the residuals from the original model
residuals_original <- residuals(original_model)

acf(residuals_original)
```




```r
# Estimate the autoregressive parameter phi_1 from the residuals
phi_1 <- arima(residuals_original, order = c(1,0,0))$coef[2]

# Perform the autoregressive transformation on the target variable and climate variables
transformed_data <- diff(is.numeric(water), lag = 1, differences = 1)

# Apply differencing to all numeric columns in the data frame
df<- data.frame(lapply(water, diff))

# Fit the transformed model with the autoregressive parameter phi_1
```

```r
transformed_model <- lm(gauge ~ temp + rain + snow, data = df)

# Check the autocorrelation of residuals in the transformed model
acf(residuals(transformed_model))
```




**Are the model residuals uncorrelated after applying the transformation?**

Yes, they are. In the first plot, the autocorrelation in the errors is not explicitly addressed, and the errors are assumed to be independent.

In the second ACF plot, which represents the autocorrelation of the residuals after applying the transformation, we observe a notable difference. An AR(1) model is employed to address potential autocorrelation by introducing a lagged error term ($\phi_1 \eta_{i-1}$) into the transformed equation.

The ACF values in the transformed data are significantly closer to zero compared to the original data, indicating that the transformation has effectively reduced autocorrelation. This outcome aligns with the goal of AR(1) models, where the residuals should be uncorrelated.