Econometrics II - Problem 4

William Radaic Peron

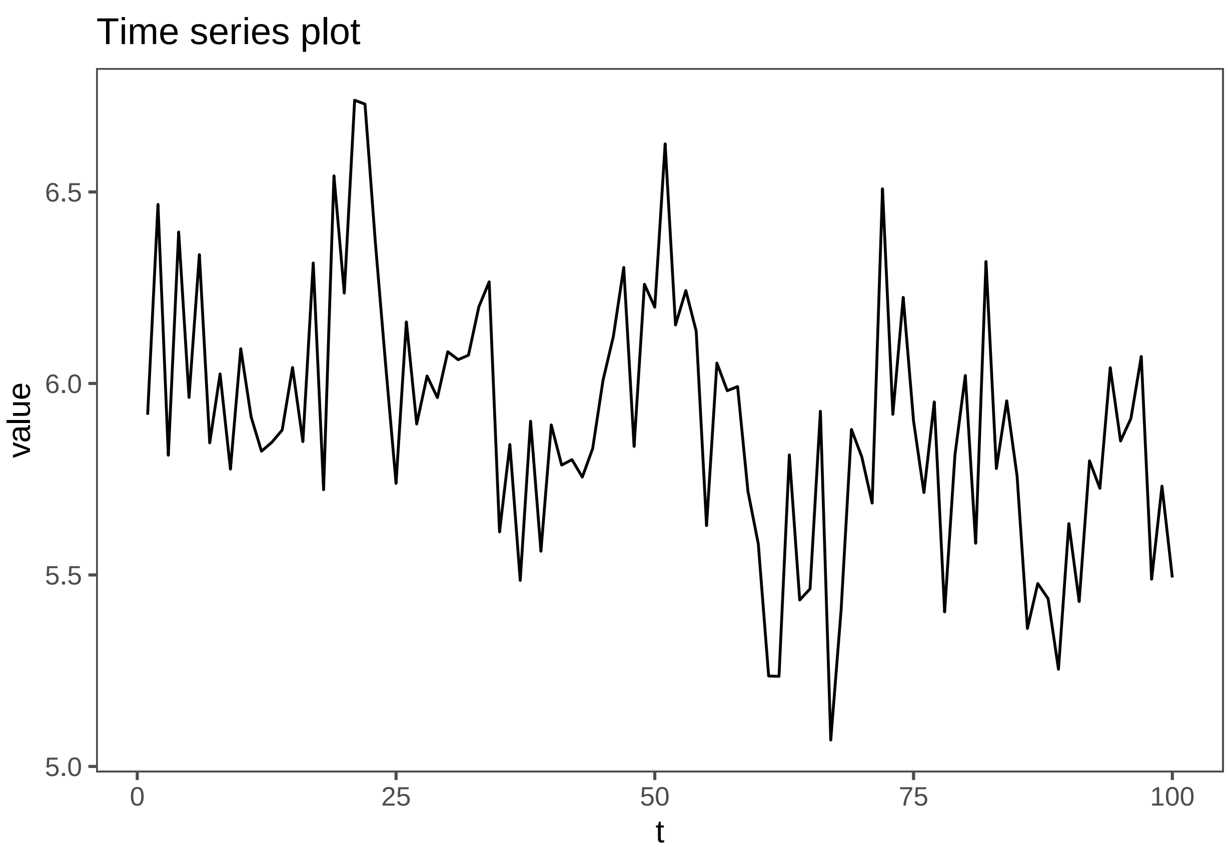
September 20, 2020

In this problem, we'll be tackling the issue of *forecasting* of an ARMA model. The problem is split in two parts: (i) *cross-validation*; and (ii) *bootstrapping*.

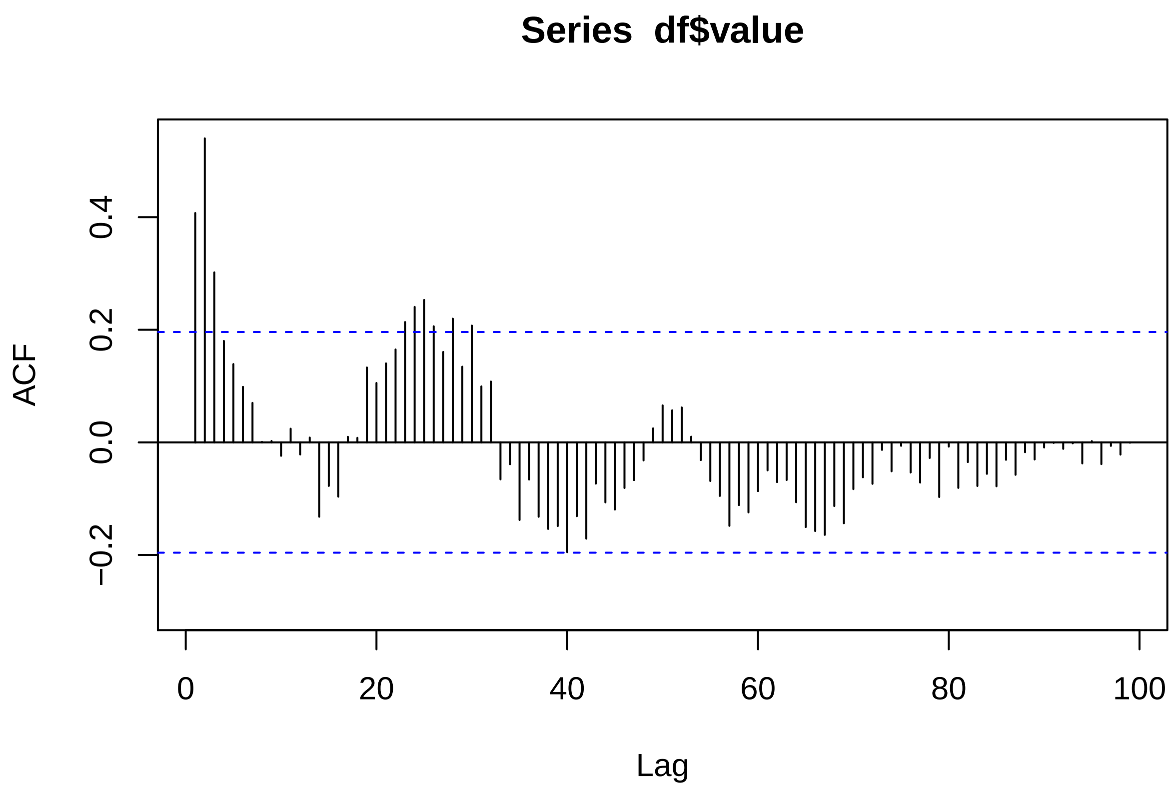
Identification and estimation

First, let's identify the best model for our time series.

```
df <- data.frame(df)

pplot <- ggplot(data = df, aes(x = t, y = value)) + geom_line() +
    ggtitle("Time series plot") + theme_few()
pplot
```




```
acf_ts <- Acf(df$value, lag.max = 5000)
```




```r
acf_test_values <- acf_ts$acf/sd(acf_ts$acf)

head(data.frame(acf_test_values))
```

```
##   acf_test_values
## 1        6.176432
## 2        2.515951
## 3        3.335438
## 4        1.864909
## 5        1.112884
## 6        0.858639
```

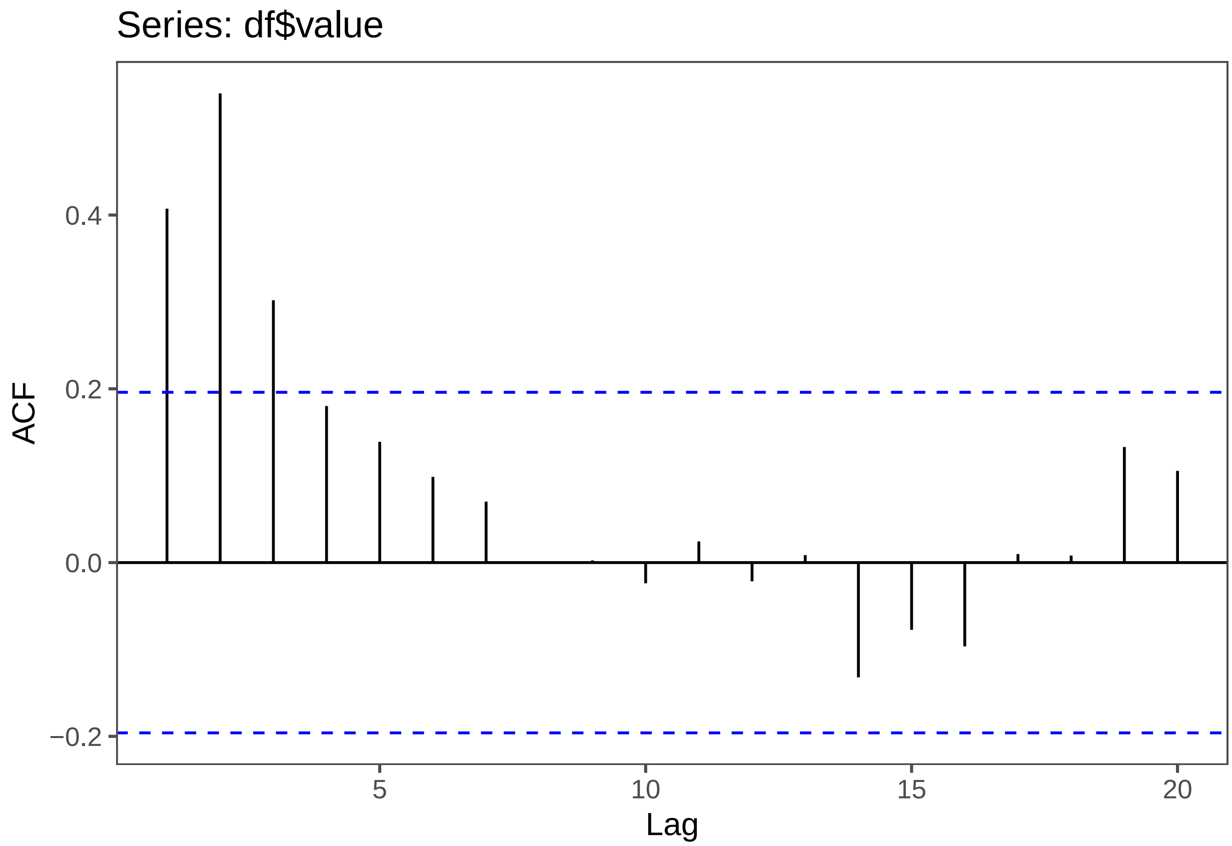
```r
facst <- ggAcf(df$value, type = "correlation", lag.max = 20,
    plot = T) + theme_few()
faclt <- ggAcf(df$value, type = "correlation", lag.max = 5000,
    plot = T) + theme_few()

facpst <- ggPacf(df$value, type = "correlation", lag.max = 100,
    plot = T) + theme_few()
```

```
## Warning: Ignoring unknown parameters: type
```

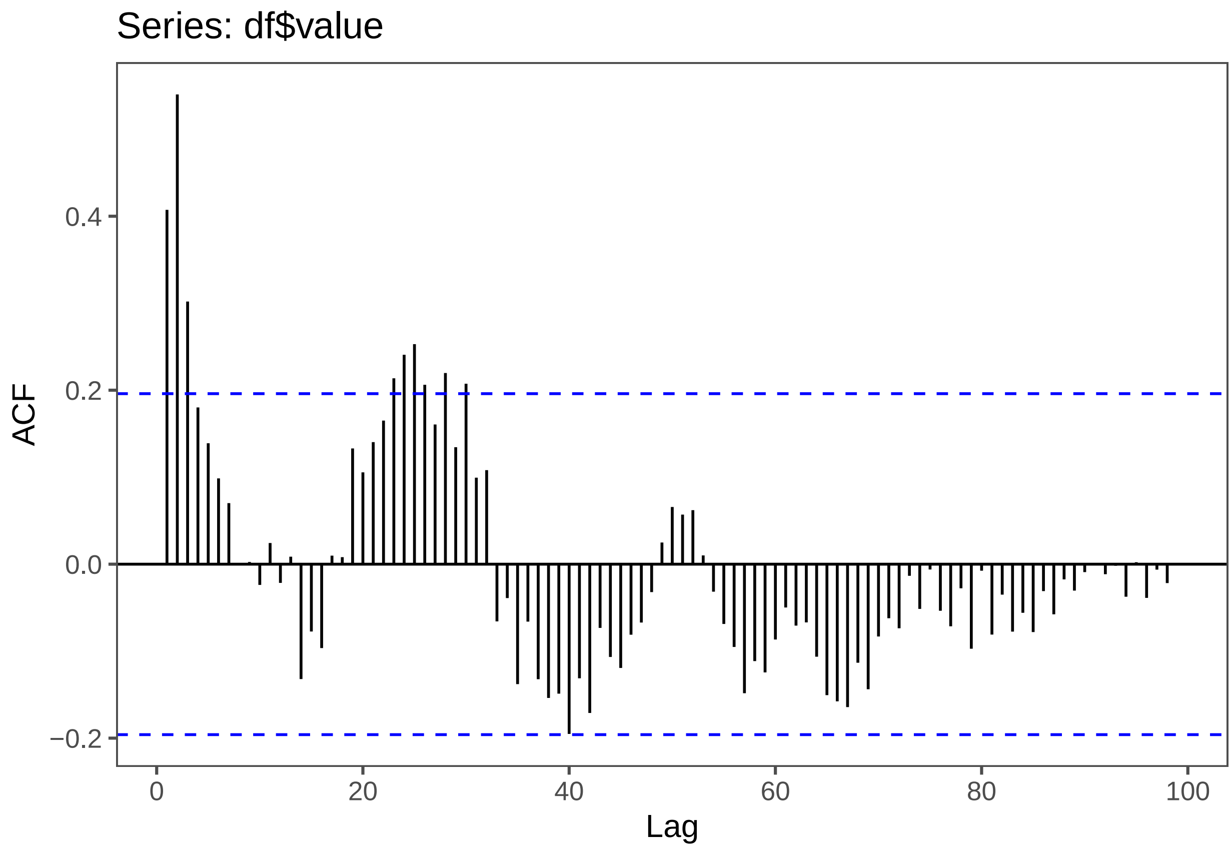
```r
facplt <- ggPacf(df$value, type = "correlation", lag.max = 5000,
    plot = T) + theme_few()
```

```
## Warning: Ignoring unknown parameters: type
```

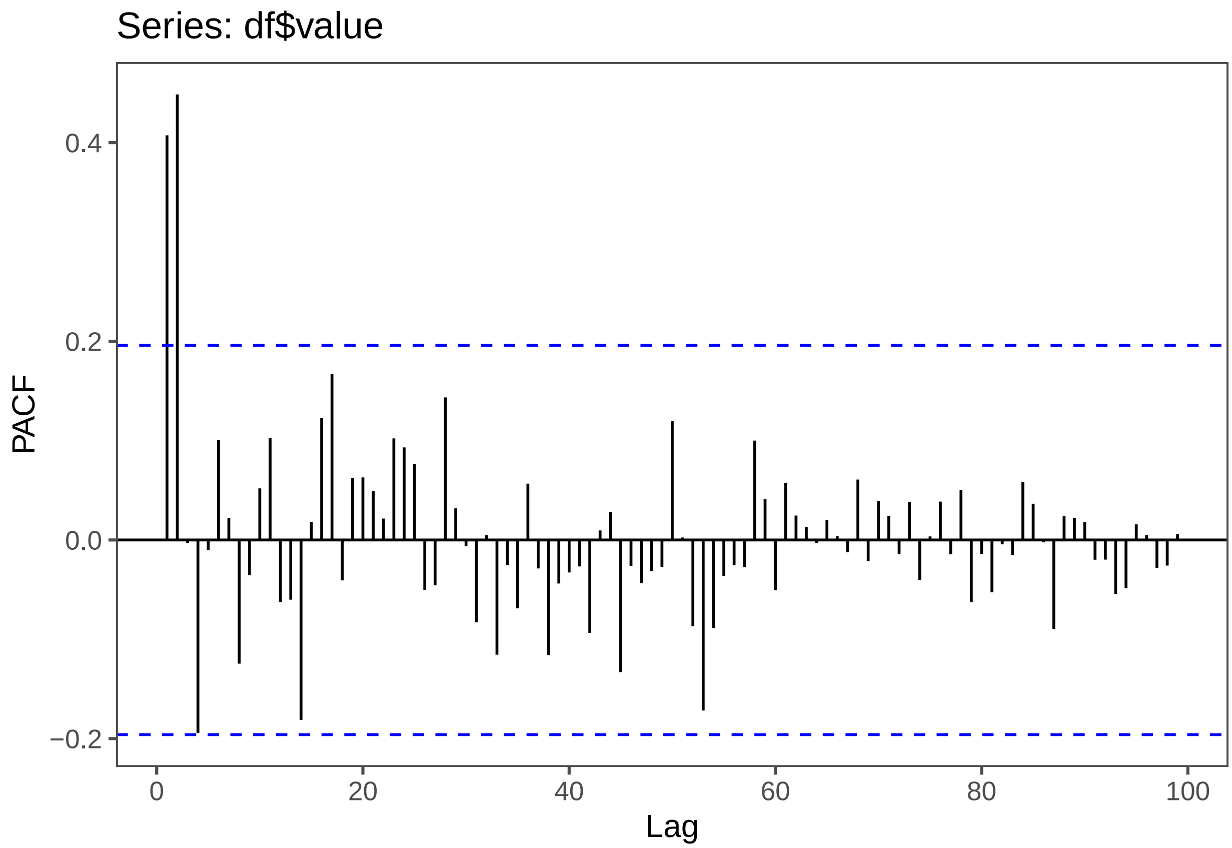
```
facst
```



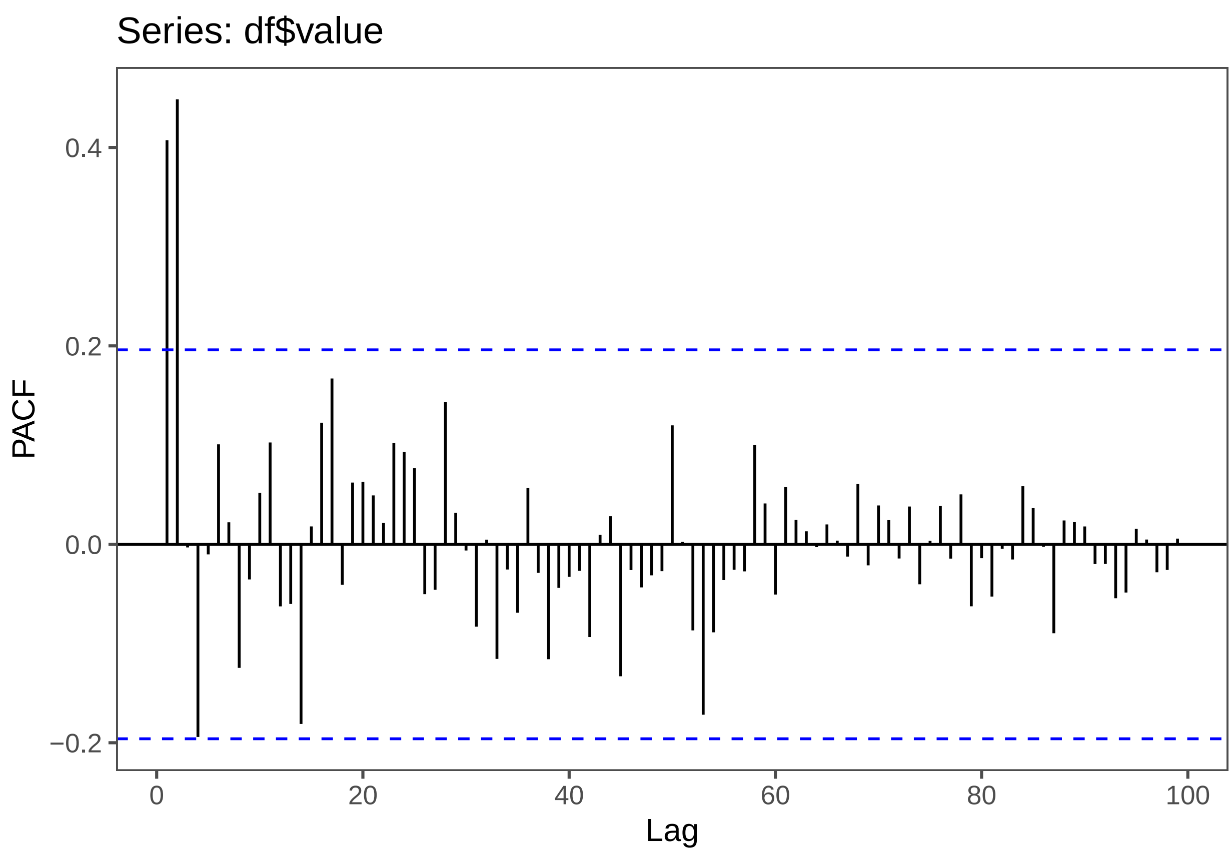

```
faclt
```




```
facpst
```




```
facplt
```



We'll now use the function *auto.arima* from the package *forecast* to identify and estimate the model.

```
aa_model <- auto.arima(df$value, num.cores = 24, max.d = 0, stepwise = F)

summary(aa_model)
```

```
## Series: df$value
## ARIMA(0,0,3) with non-zero mean
##
## Coefficients:
##          ma1     ma2     ma3    mean
##       0.1814  0.6647  0.4001  5.8982
## s.e.  0.0852  0.0750  0.0949  0.0562
##
## sigma^2 estimated as 0.0667:  log likelihood=-5.42
## AIC=20.85   AICc=21.49   BIC=33.88
##
## Training set error measures:
##                        ME      RMSE       MAE        MPE     MAPE      MASE
## Training set -0.002315954 0.2530428 0.2131067 -0.2268814 3.612855 0.7314965
##                    ACF1
## Training set 0.03868106
```

```
print("t-values: ")
```

```
## [1] "t-values: "
```

```r
aa_t <- matrix(NA, nrow = aa_model$arma[1] + aa_model$arma[2])

for (i in c(1:4)) {

    aa_t[i] <- aa_model$coef[i]/sqrt(aa_model$var.coef[i, i])

}

aa_t <- data.frame(aa_t)

aa_t
```

```
##         aa_t
## 1   2.128691
## 2   8.861580
## 3   4.216481
## 4 105.004537
```

```r
aa_q <- Box.test(aa_model$residuals, lag = aa_model$arma[1] +
    aa_model$arma[2])
aa_q
```

```
##
##  Box-Pierce test
##
## data:  aa_model$residuals
## X-squared = 0.35002, df = 3, p-value = 0.9504
```

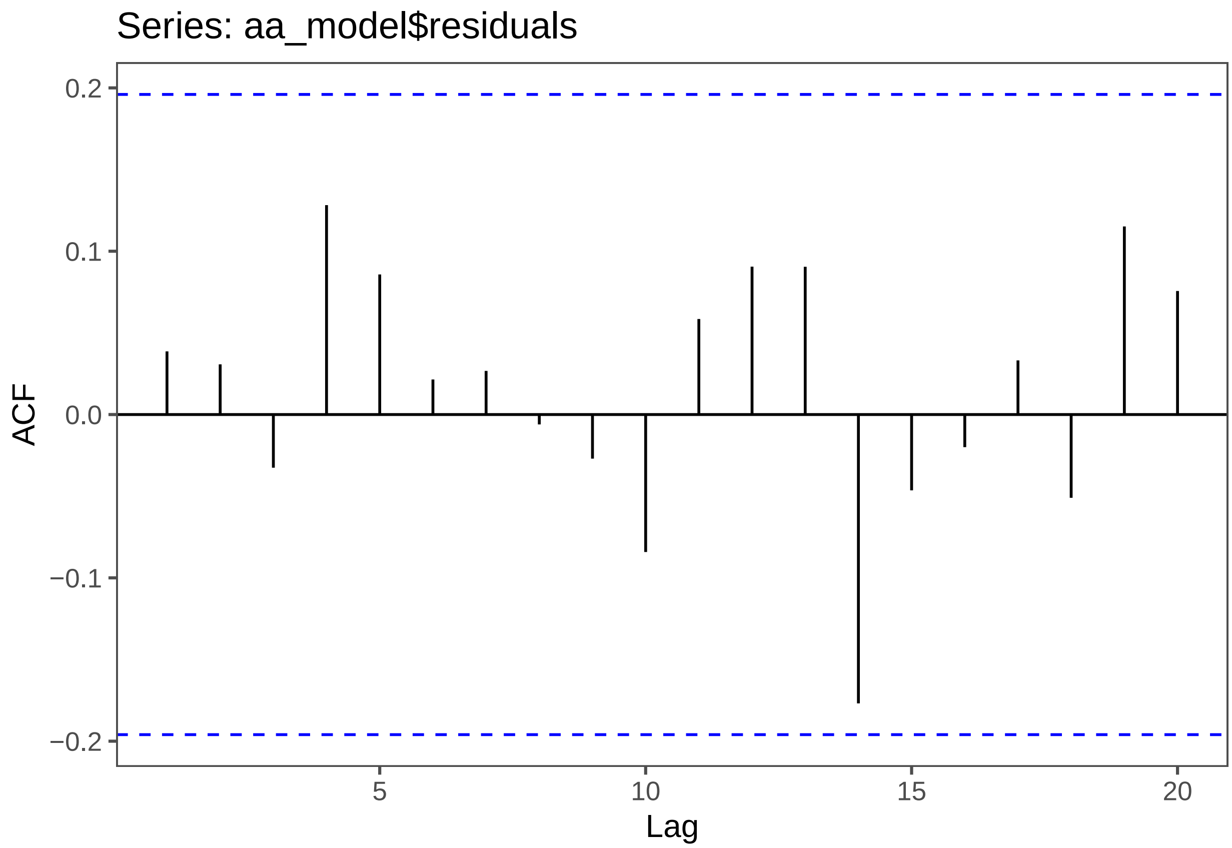
```r
criteria <- matrix(NA, nrow = 1, ncol = 3)

aa_criteria <- data.frame("MA(3)*", aa_model$aic, aa_model$bic)

names(aa_criteria) <- c("Model", "AIC", "BIC")

aa_criteria
```

```
##    Model      AIC      BIC
## 1 MA(3)* 20.84963 33.87549
```

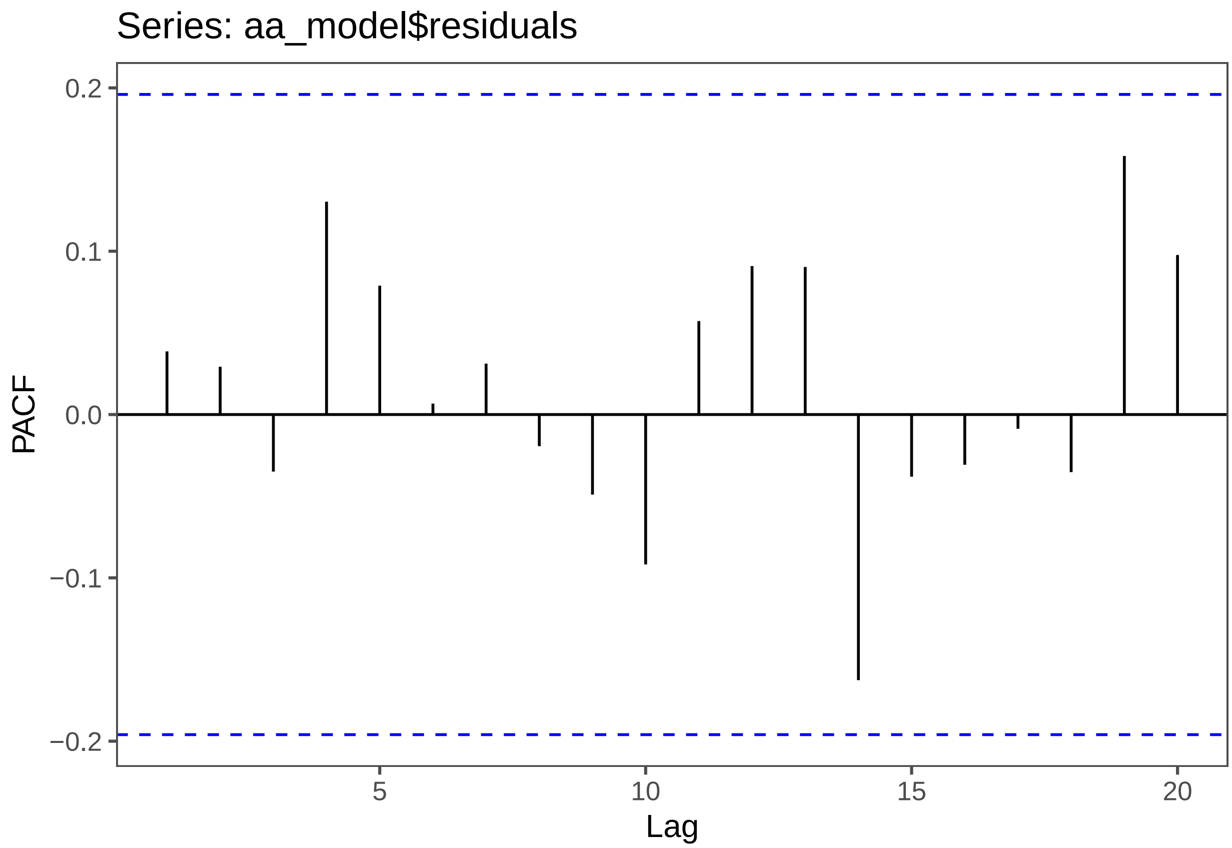
```r
fac_e <- ggAcf(aa_model$residuals, type = "correlation", lag.max = 20,
    plot = T) + theme_few()

facp_e <- ggPacf(aa_model$residuals, type = "correlation", lag.max = 20,
    plot = T) + theme_few()
```

```
## Warning: Ignoring unknown parameters: type
```

```r
fac_e
```




```
facp_e
```




```r
mean(aa_model$residuals)
```

```
## [1] -0.002315954
```

The results of *auto.arima* imply that the best model is an ARMA(0,3) – i.e., a MA(3):

$$y_t = c + +\theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \theta_3 \varepsilon_{t-3} + \varepsilon_t, \quad \varepsilon_t \sim wn(0, \sigma^2)$$

Furthermore, the Q-statistic *(Box.test)* seems to indicate that ε_t is truly white noise.

Cross-validation

Let's now cross-validate or model. This will now be done manually; afterwards, an automatized version from *fpp* shall be presented.

Let $h := 5$; $frac = 0.2$. T is the size of our sample; k is the *training* database. The remainder shall be used for testing purposes.

As we have discovered previously, *auto.arima* yields a **MA(3)** model. It will now be used.

```r
h <- 5

frac <- 0.2

T <- length(df$value)

k <- floor((1 - frac) * T)
```

```
# Estimating MA(3) with k = 80
fit <- Arima(df$value[1:k], order = c(0, 0, 3))

# Generating predictions from the model
pred <- predict(fit, n.ahead = h)

# Calculating errors between the predicted values of the
# model and the actual values of the testing database

e <- df$value[(k + h)] - pred$pred[h]

e
```

```
## [1] -0.1951299
```

Let's now update our training database iteratively with a for loop.

```
e <- matrix(NA, nrow = 100)

# Updating the model

for (i in k:(T - h)) {

    fit <- Arima(df$value[1:i], order = c(0, 0, 3))

    pred <- predict(fit, n.ahead = h)

    e[i, 1] <- df$value[(i + h)] - pred$pred[h]

}
```

With the matrix e in hands, we can now calculate MSE:

```
mse <- mean(e^2, na.rm = T)
```

This procedure can now be used to compare other models against the model from *auto.arima*.

```
max_p <- 5

max_q <- 5

e <- matrix(NA, nrow = 100, ncol = (max_p + 1) * (max_q + 1))

pred <- vector("list", (max_p + 1) * (max_q + 1))

fit <- vector("list", (max_p + 1) * (max_q + 1))

# Updating the model
for (u in 0:max_q) {

    for (j in 0:max_p) {


        for (i in k:(T - h)) {

            fit[[(((max_p + 1) * j) + u + 1)]] <- Arima(df$value[1:i],
```

```r
                order = c(j, 0, u))

            # fit <- append(fit, Arima(df$value[1:i], order = c(j,0,u)))

            # pred <- append(pred, predict(fit[[(j+u)]], n.ahead = h))

            pred[[(((max_p + 1) * j) + u + 1)]] <- predict(fit[[(((max_p +
                1) * j) + u + 1)]], n.ahead = h)

            e[i, (((max_p + 1) * j) + u + 1)] <- df$value[(i +
                h)] - pred[[(((max_p + 1) * j) + u + 1)]]$pred[h]

        }

    }

}

mse <- matrix(NA, nrow = ((max_p + 1) * (max_q + 1)), ncol = 1)




mse <- colMeans(e^2, na.rm = T)

mse
```

```
##  [1] 0.1357466 0.1354001 0.1368083 0.1374243 0.1376508 0.1441940 0.1347115
##  [8] 0.1269779 0.1347789 0.1373465 0.1398588 0.1436175 0.1313779 0.1315448
## [15] 0.1435805 0.1355649 0.1421277 0.1335153 0.1316765 0.1333955 0.1400838
## [22] 0.1427467 0.1473227 0.1347447 0.1320856 0.1333025 0.1354734 0.1341742
## [29] 0.1380676 0.1357880 0.1346228 0.1382810 0.1319484 0.1308446 0.1382417
## [36] 0.1327046
```

```r
optimal_index <- which.min(mse)

cv_model <- fit[[optimal_index]]

summary(cv_model)
```

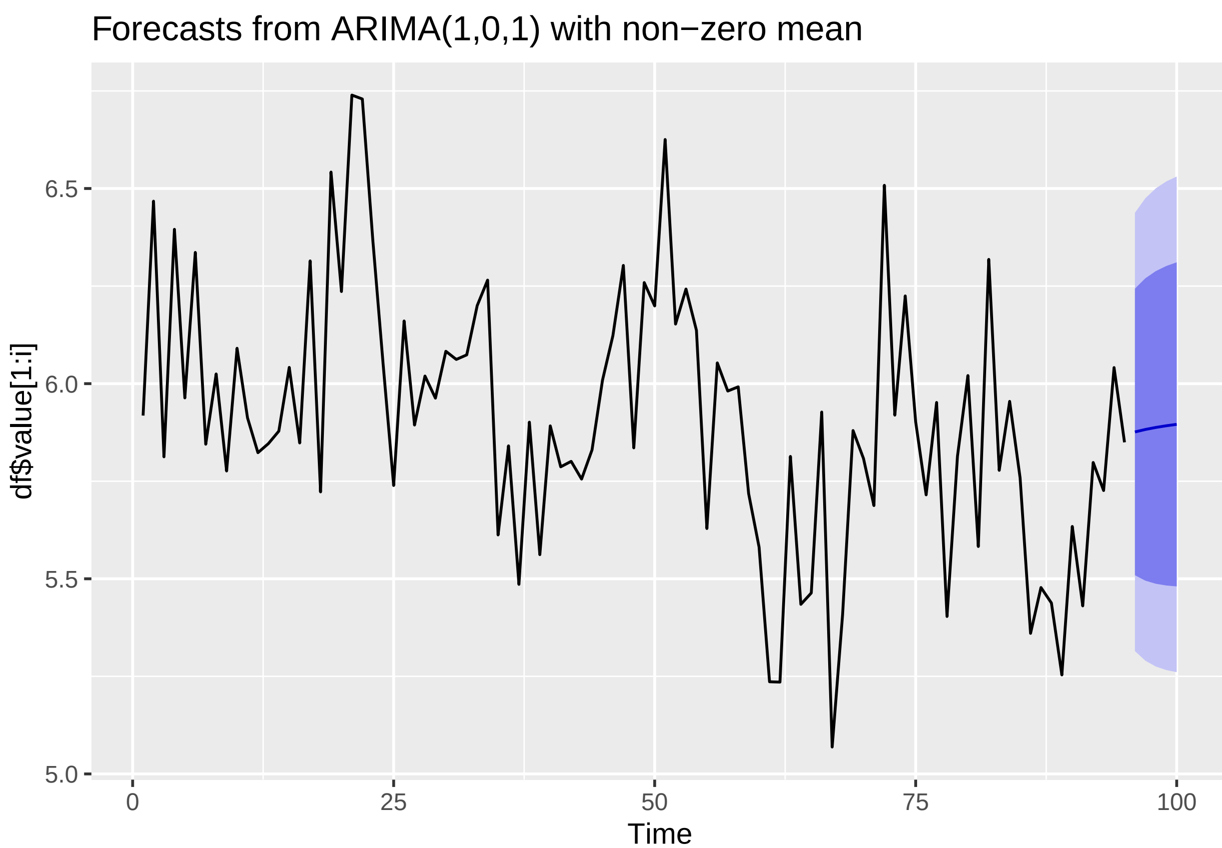
```
## Series: df$value[1:i]
## ARIMA(1,0,1) with non-zero mean
##
## Coefficients:
##          ar1      ma1    mean
##       0.8253  -0.4888  5.9125
## s.e.  0.0814   0.1118  0.0815
##
## sigma^2 estimated as 0.08209:  log likelihood=-14.72
## AIC=37.43   AICc=37.88   BIC=47.65
##
## Training set error measures:
##                        ME      RMSE       MAE        MPE     MAPE      MASE
## Training set -0.002725722 0.2819456 0.2228183 -0.2756862 3.787114 0.7600473
##                     ACF1
```

```
## Training set -0.1279409
```

The cross-validation method constructed above yielded an ARMA(1,1):

$$y_t = c + \phi_1 y_{t-1} + +\theta_1 \varepsilon_{t-1} + \varepsilon_t, \quad \varepsilon_t \sim wn(0, \sigma^2)$$

```r
cv_fc <- forecast(cv_model, h = h)

autoplot(cv_fc)
```



Bootstrapping

Now, let's proceed to *bootstrapping*. It envolves the following steps:

1. - Estimate ARMA(p,q)

$$Y_t = c + \sum_{j=1}^{p} \phi_j Y_{t-j} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

   - Calculate the residuals of the regression:

$$\hat{\varepsilon}_t := Y_t - (\hat{c} + \sum_{j=1}^{p} \hat{\phi}_j Y_{t-j} + \sum_{j=1}^{q} \hat{\theta}_j \varepsilon_{t-j})$$

   - If the residuals do not have mean 0, create the centered residuals:

$$\tilde{\varepsilon}_t = \hat{\varepsilon}_t - \frac{1}{t} \sum_{t=1}^{T} \hat{\varepsilon}_t$$

2. - Select at random, with restocking, a sample with $T + m$ elements, $m >> 0$:

$$\{\varepsilon_1^*, ..., \varepsilon_{T+m}^*\}$$

   - Create a series $\{Y_t^*\}_{t=1}^{T+m}$:

$$Y_t^* = Y_t, 1 \leq t \leq max(p, q)$$

$$Y_t^* = \hat{c} + \sum_{j=1}^{p} \hat{\phi}_j Y_{t-j} + \sum_{j=1}^{q} \hat{\theta}_j \varepsilon_{t-j}^* + \varepsilon_t^*, max(p, q) < t \leq T + m$$

3. - Using the simulated sample $\{Y_t^*\}_{t=1}^{T+m}$, create a forecast for $h > 0$ periods using the estimated coefficients *obtained with the real sample*.
   - This yields a vector of dimension h containing the forecasts in the form:

$$(\hat{Y}_{T+1}^*, ..., \hat{Y}_{T+h}^*)$$

   - Repeat steps 2 and 3 for S times. Create a matrix with the results.
   - This yields a S x h matrix where each row is equal to the aforementioned vector.

We'll use, again, the optimal model from *auto.arima*, MA(3):

$$y_t = c + +\theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \theta_3 \varepsilon_{t-3} + \varepsilon_t, \quad \varepsilon_t \sim wn(0, \sigma^2)$$

```
S <- 1000

m <- 100



optimal_p <- aa_model$arma[1]

optimal_q <- aa_model$arma[2]

e_sample <- data.frame(matrix(NA, nrow = S, ncol = (length(df$value) +
    m)))

y_star <- data.frame(matrix(NA, nrow = S, ncol = (length(df$value) +
    m + max(aa_model$arma[1], aa_model$arma[2]))))

arima_star <- data.frame(matrix(NA, nrow = S, ncol = (length(df$value) +
    m + max(aa_model$arma[1], aa_model$arma[2]))))


for (i in 1:S) {

    e_sample[i] <- sample(aa_model$residuals, replace = T, size = (length(df$value) +
        m))

}

for (i in 1:S) {

    for (j in ((aa_model$arma[1] + aa_model$arma[2] + 1):(length(df$value) +
```

```
        m))) {

        arima_star[i, j] <- (aa_model$coef[4] + (aa_model$coef[1] *
            e_sample[i, j - 1]) + (aa_model$coef[2] * e_sample[i,
            j - 2]) + (aa_model$coef[3] * e_sample[i, j - 3]) +
            e_sample[i, j])

    }

}


y_fixed <- data.frame(matrix(NA, nrow = S, ncol = (aa_model$arma[1] +
    aa_model$arma[2])))

for (i in 1:S) {
    y_fixed[i, 1] <- data.frame(df$value[1])
    y_fixed[i, 2] <- data.frame(df$value[2])
    y_fixed[i, 3] <- data.frame(df$value[3])
}


y_star <- data.frame(y_fixed, arima_star[, -(1:3)])

y_m <- y_star[, -(1:100)]

y_m <- y_m[, -(101:103)]

y_mt <- t(y_m)

y_matrix <- as.matrix(y_m)
```

```
fc_list <- vector("list", S)

for (i in 1:S) {

    fc_list[[i]] <- forecast(ts(y_matrix[i, ]), model = aa_model,
        h = 5)

}

fc_list[[1]]
```

```
##     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 101       5.880900 5.549926 6.211875 5.374719 6.387082
## 102       5.869038 5.532663 6.205413 5.354597 6.383479
## 103       5.841494 5.439567 6.243421 5.226800 6.456189
## 104       5.898184 5.475000 6.321368 5.250980 6.545387
## 105       5.898184 5.475000 6.321368 5.250980 6.545387
```

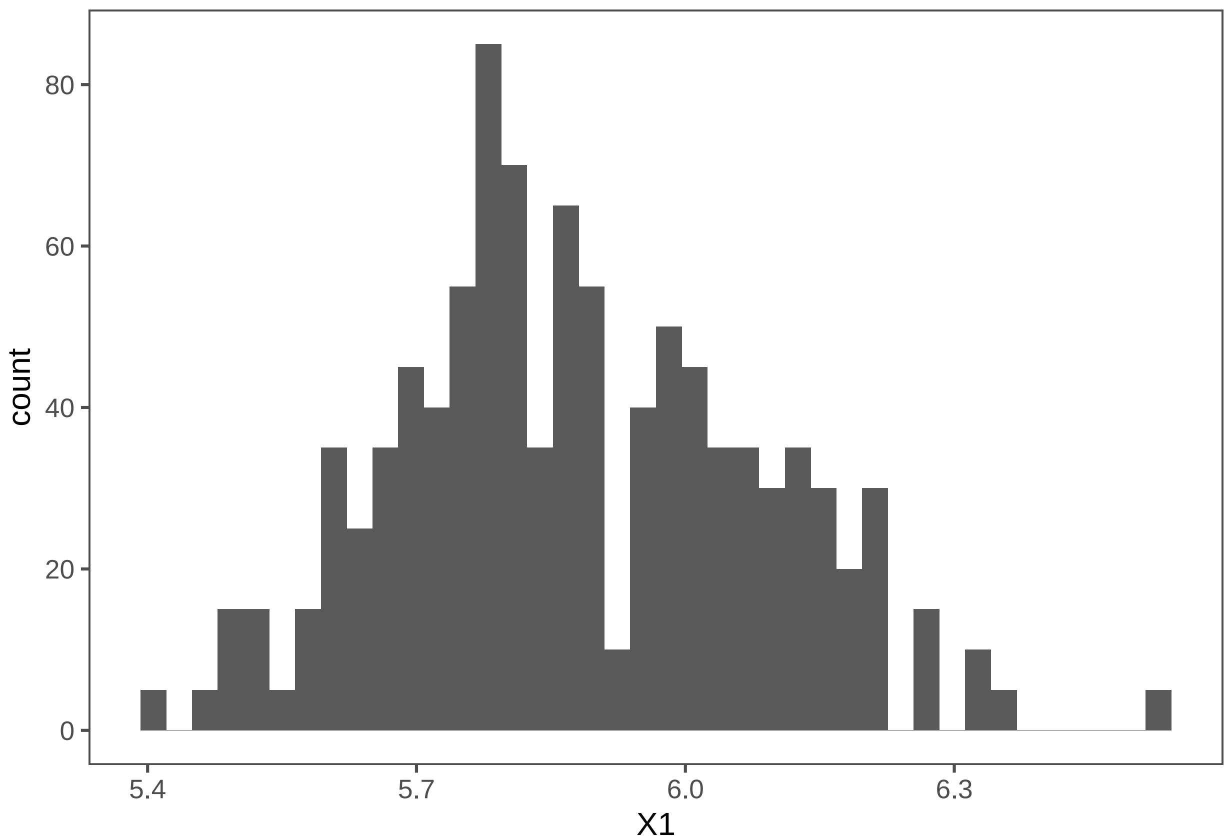
```
fc_mean <- data.frame(matrix(NA, nrow = S, ncol = 5))

for (i in 1:S) {
```

```r
    fc_mean[i, ] <- fc_list[[i]]$mean

}
```

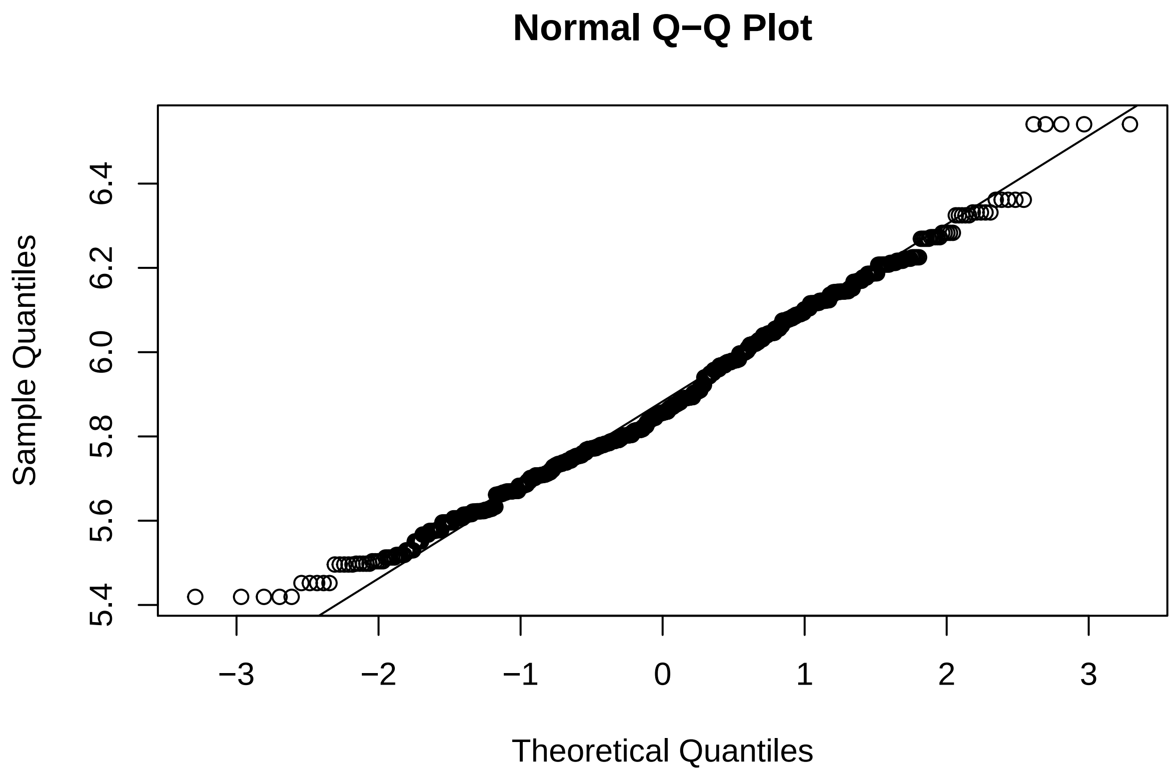
```r
head(fc_mean)
```

```
##         X1       X2       X3       X4       X5
## 1 5.880900 5.869038 5.841494 5.898184 5.898184
## 2 5.734428 5.543250 5.786803 5.898184 5.898184
## 3 5.728049 5.722659 5.832225 5.898184 5.898184
## 4 5.844103 5.943213 5.958997 5.898184 5.898184
## 5 5.550780 5.518844 5.732659 5.898184 5.898184
## 6 5.742853 5.863462 5.848949 5.898184 5.898184
```

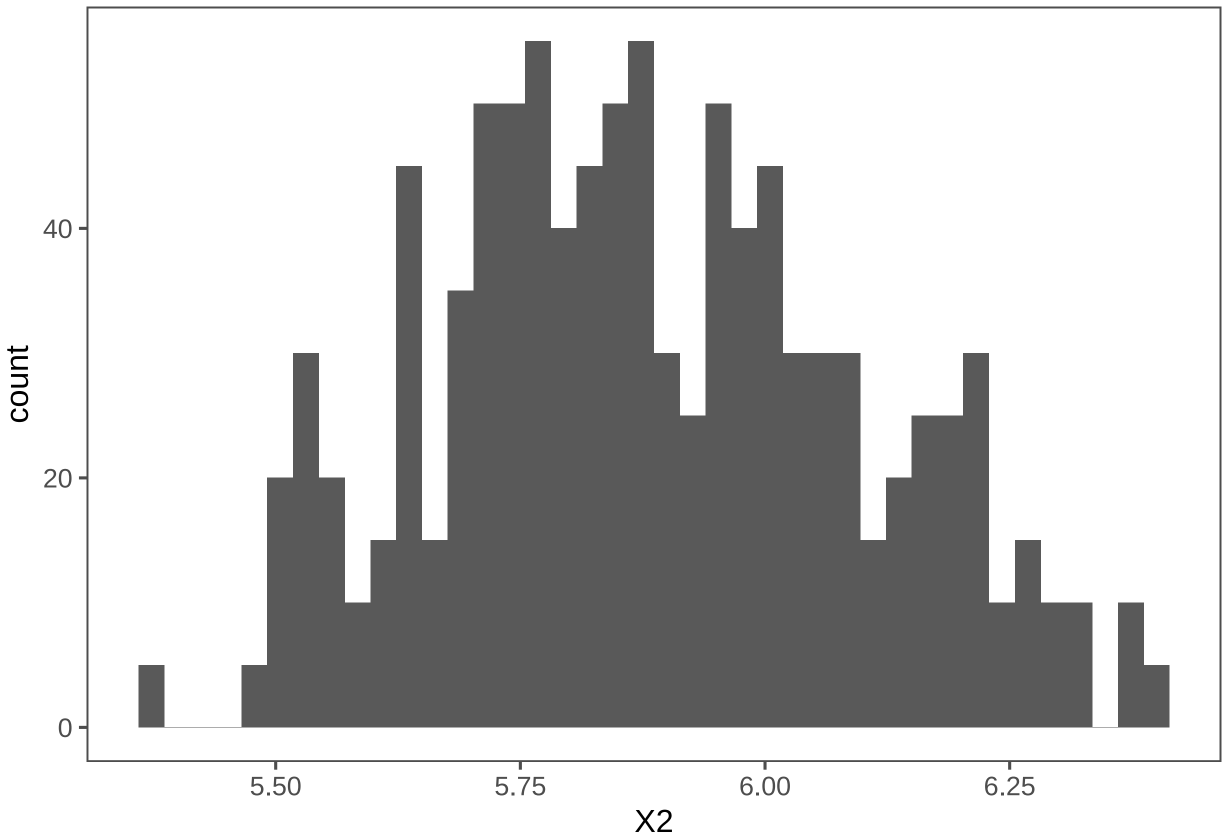
```r
hist_x1 <- ggplot(data = fc_mean, aes(x = X1)) + geom_histogram(bins = 40) +
    theme_few()
hist_x1
```



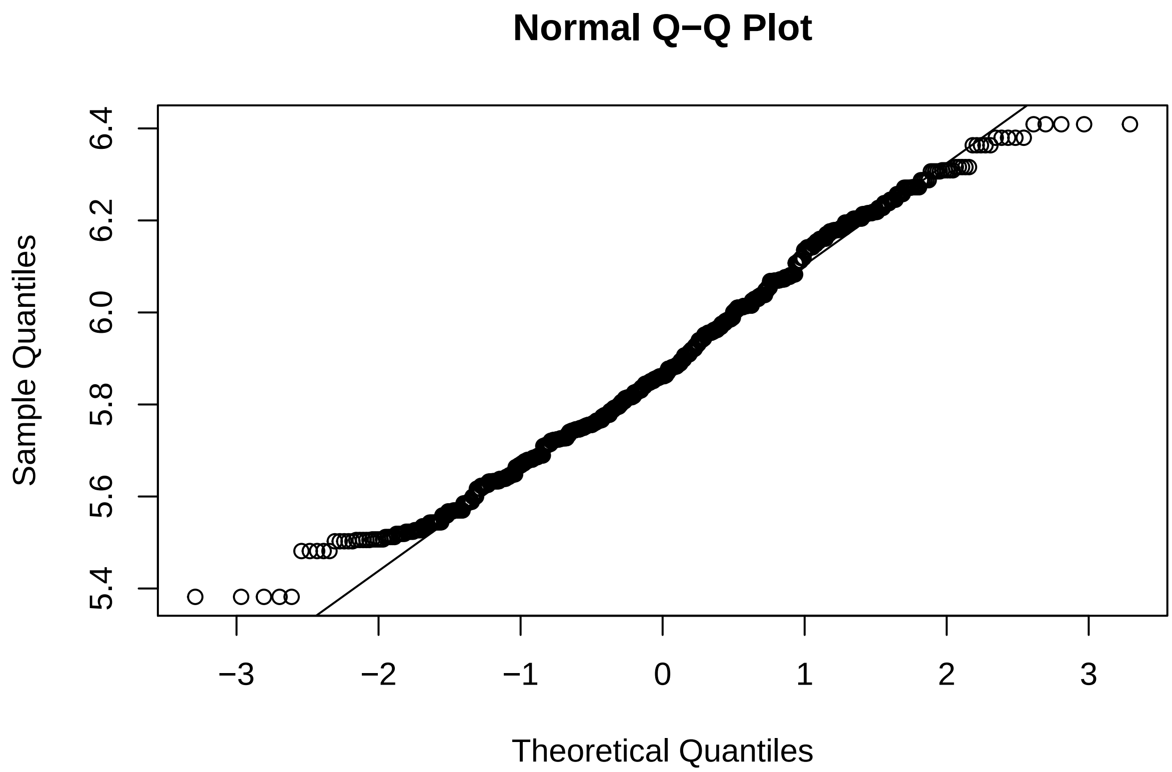

```r
qq_x1 <- qqnorm(fc_mean$X1)
qqline(fc_mean$X1)
```



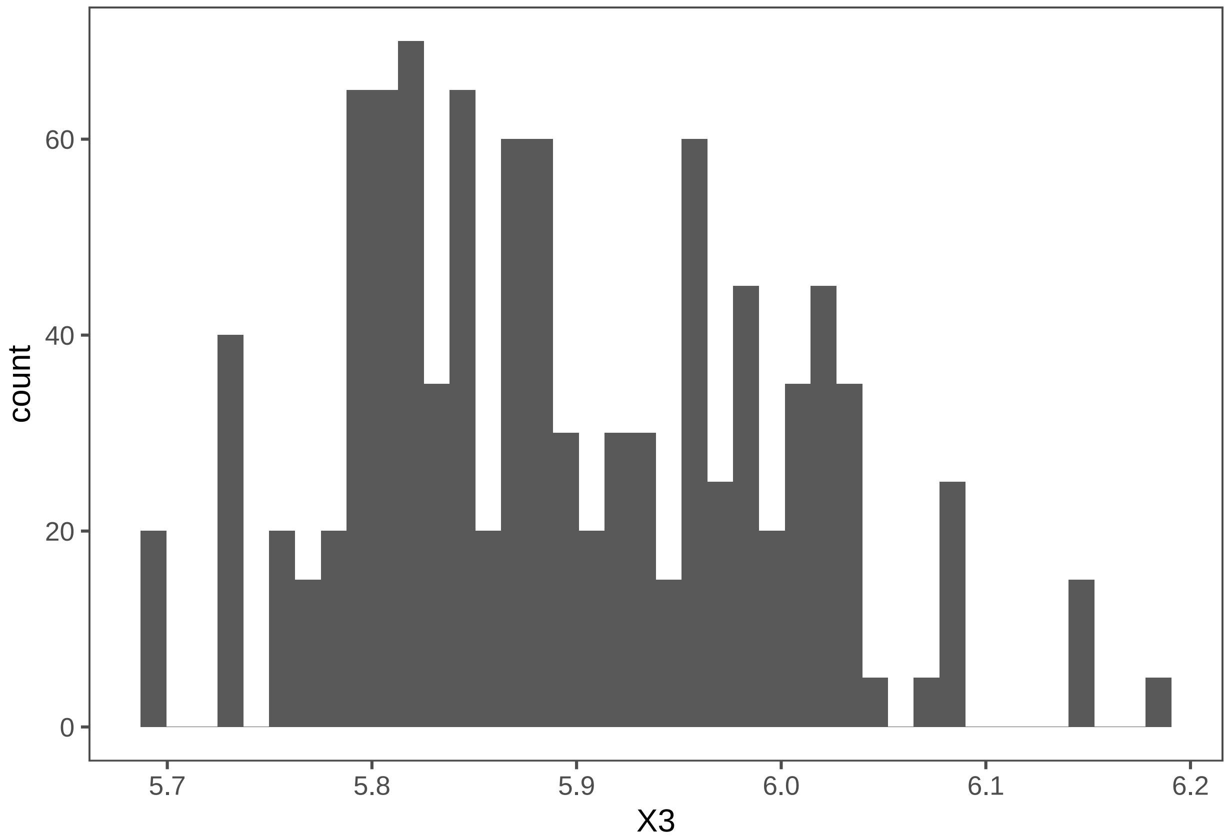

```r
hist_x2 <- ggplot(data = fc_mean, aes(x = X2)) + geom_histogram(bins = 40) +
    theme_few()
hist_x2
```



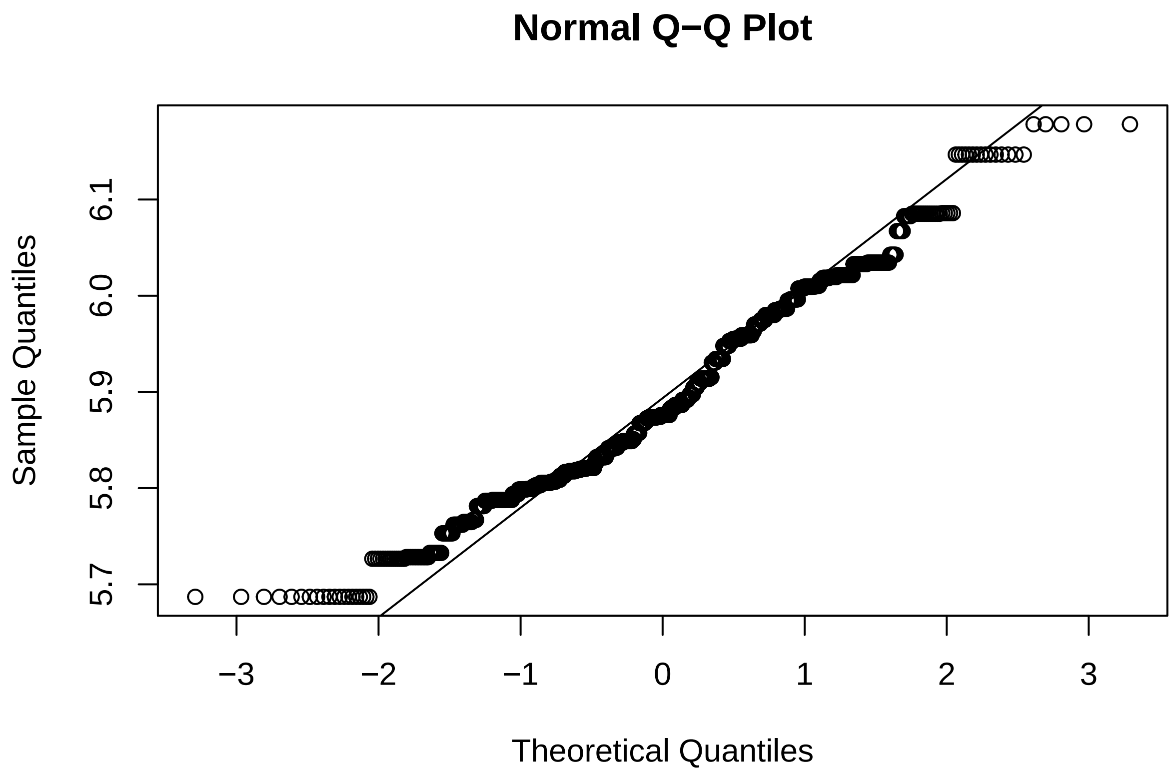

```
qq_x2 <- qqnorm(fc_mean$X2)
qqline(fc_mean$X2)
```



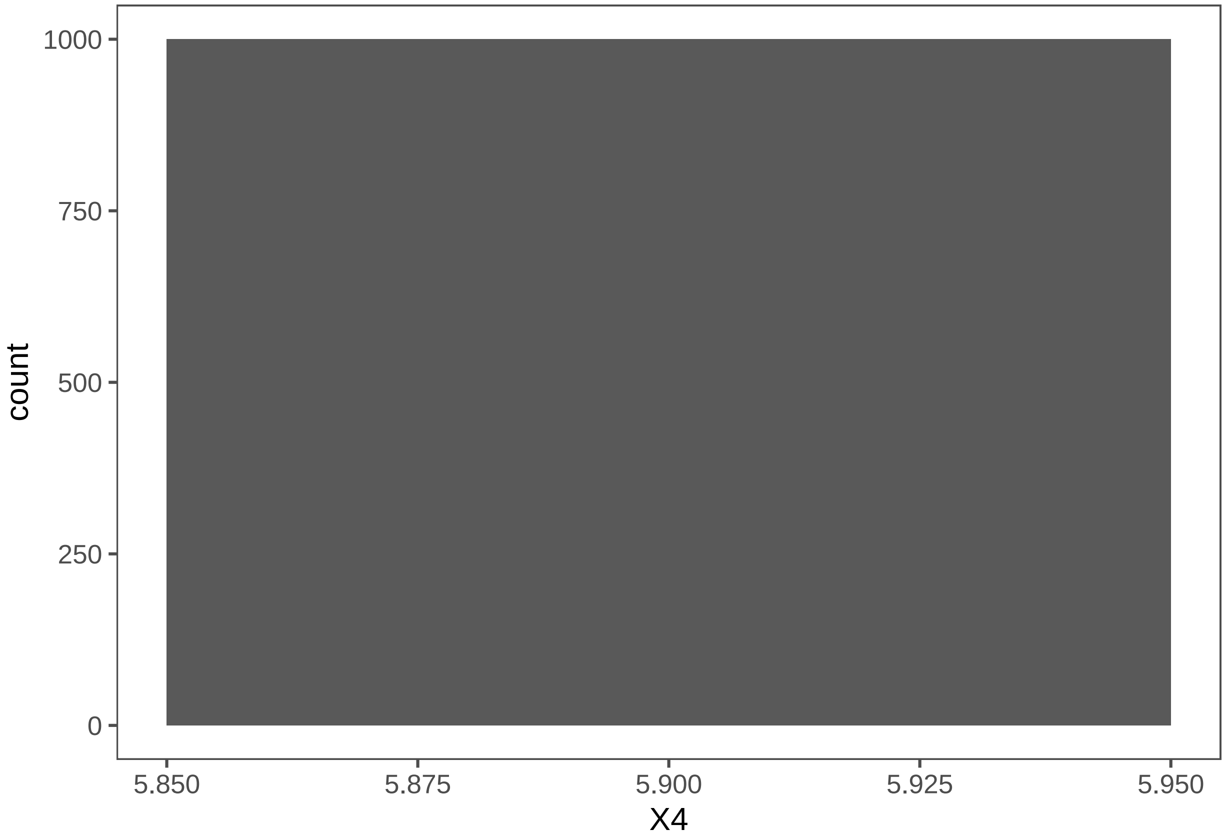

```
hist_x3 <- ggplot(data = fc_mean, aes(x = X3)) + geom_histogram(bins = 40) +
    theme_few()
hist_x3
```



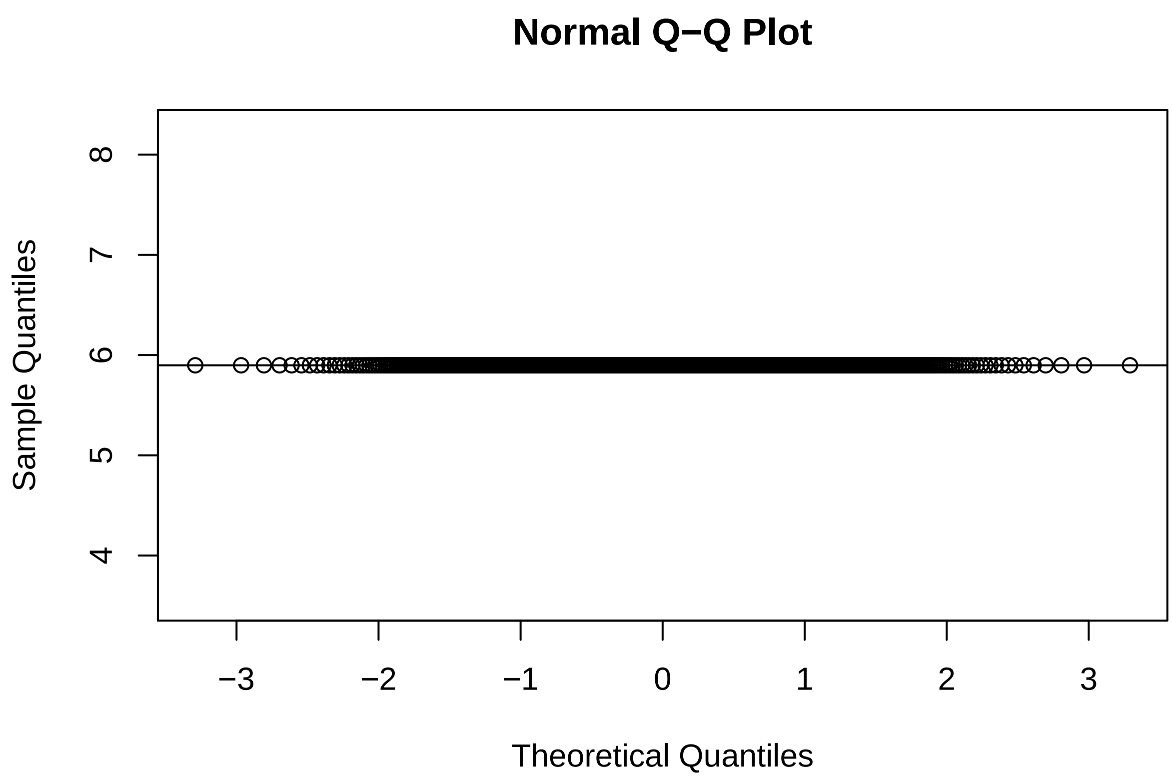

```
qq_x3 <- qqnorm(fc_mean$X3)
qqline(fc_mean$X3)
```



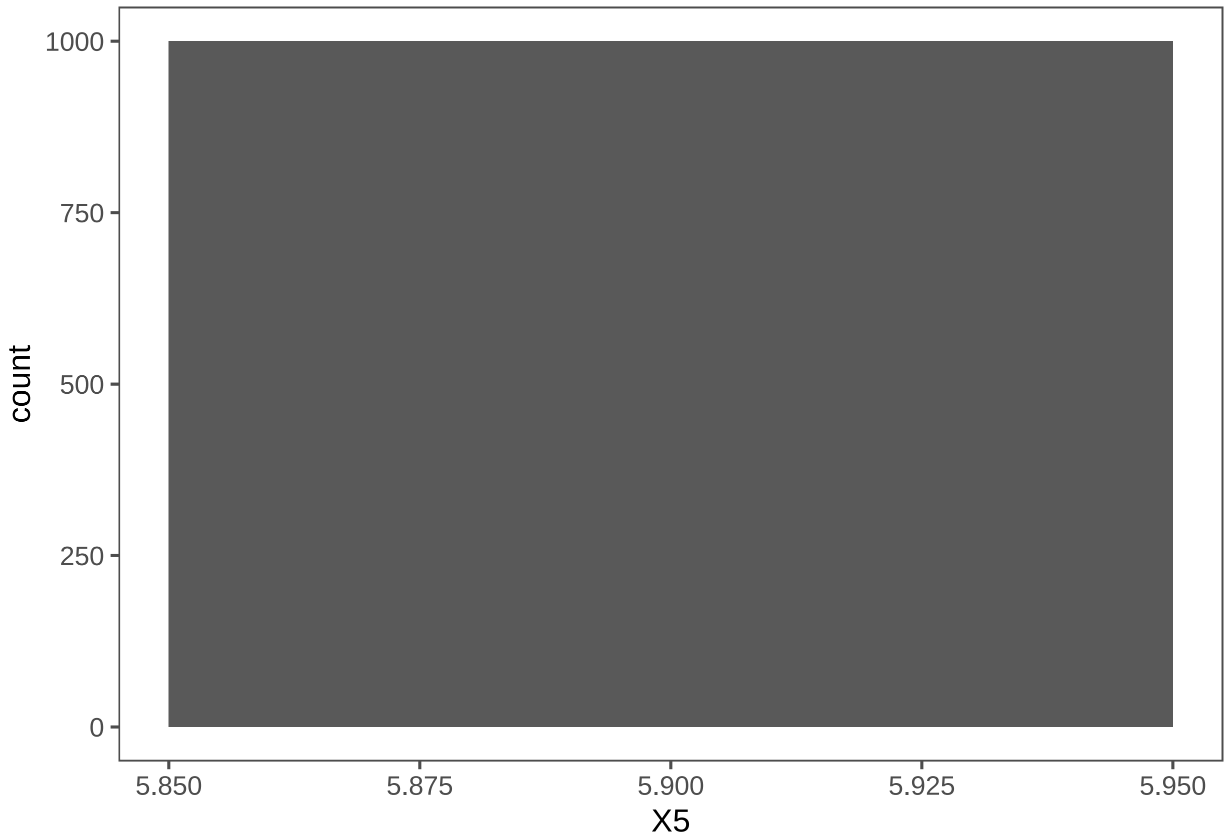

```r
hist_x4 <- ggplot(data = fc_mean, aes(x = X4)) + geom_histogram(bins = 40) +
    theme_few()
hist_x4
```



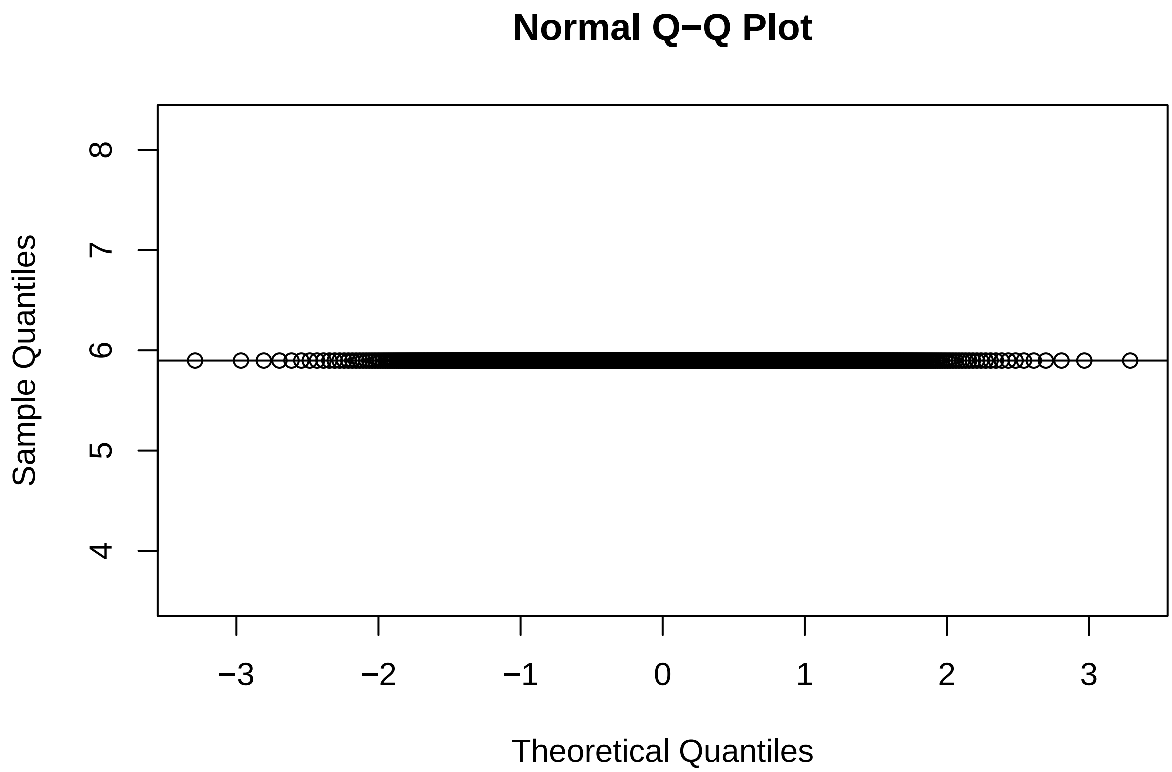

```
qq_x4 <- qqnorm(fc_mean$X4)
qqline(fc_mean$X4)
```




```
hist_x5 <- ggplot(data = fc_mean, aes(x = X5)) + geom_histogram(bins = 100) +
    theme_few()
hist_x5
```



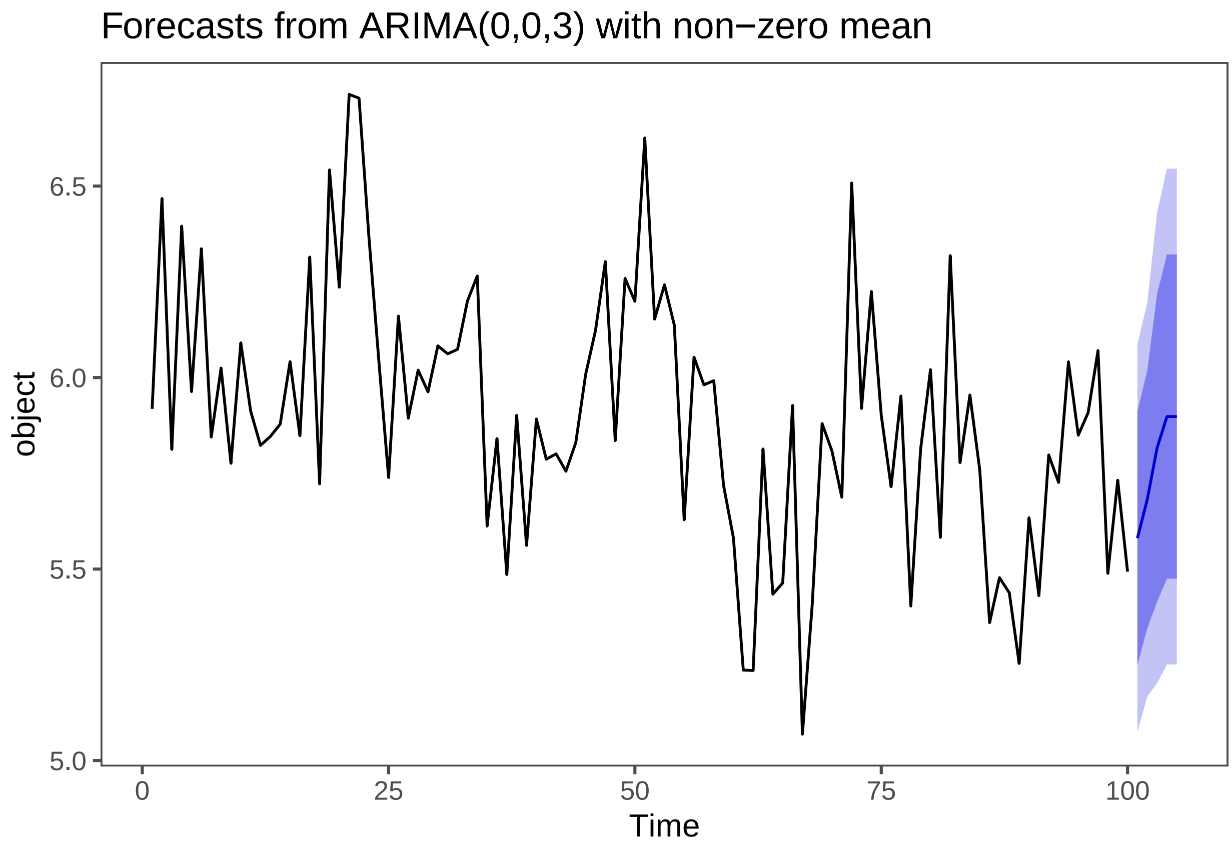

```
qq_x5 <- qqnorm(fc_mean$X5)
qqline(fc_mean$X5)
```



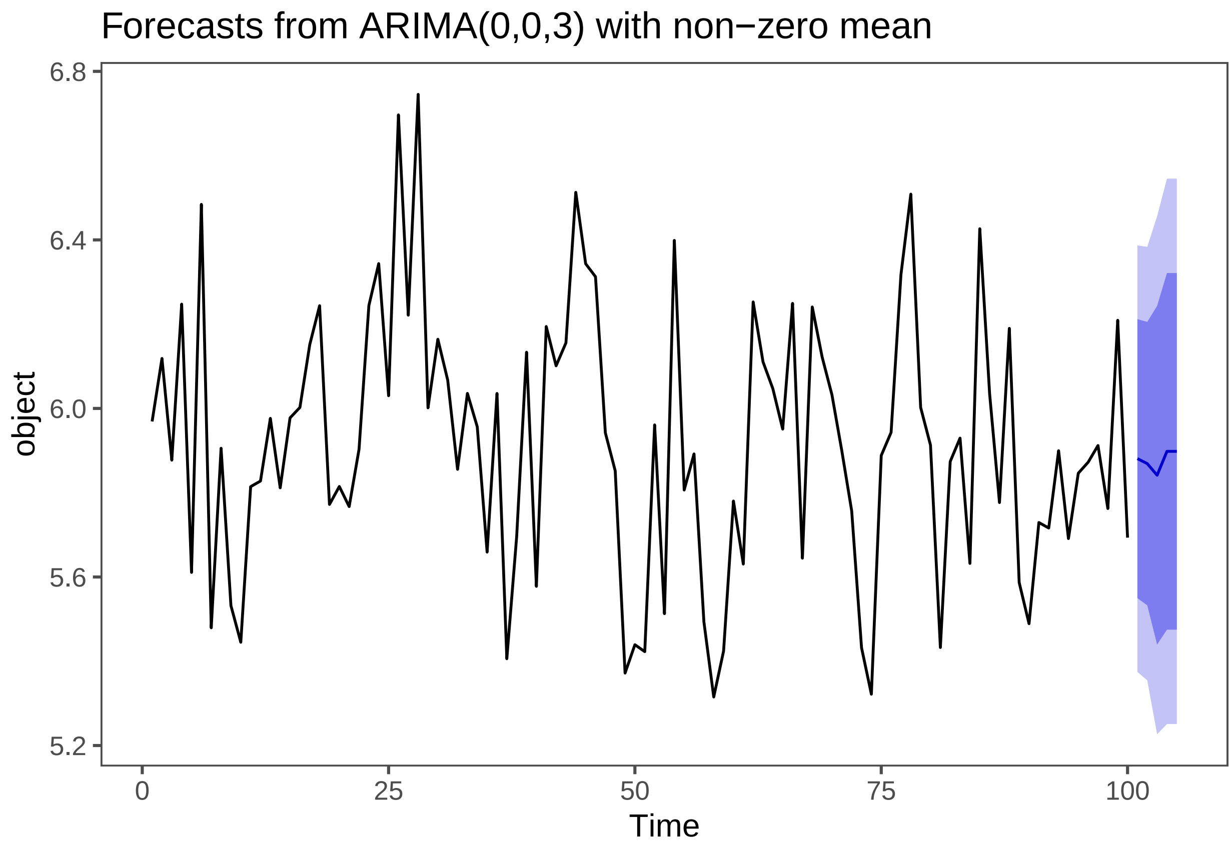
The results show that, from $h \geq 4$, the predicted value is the mean of the series.

Now, some forecasting plots:

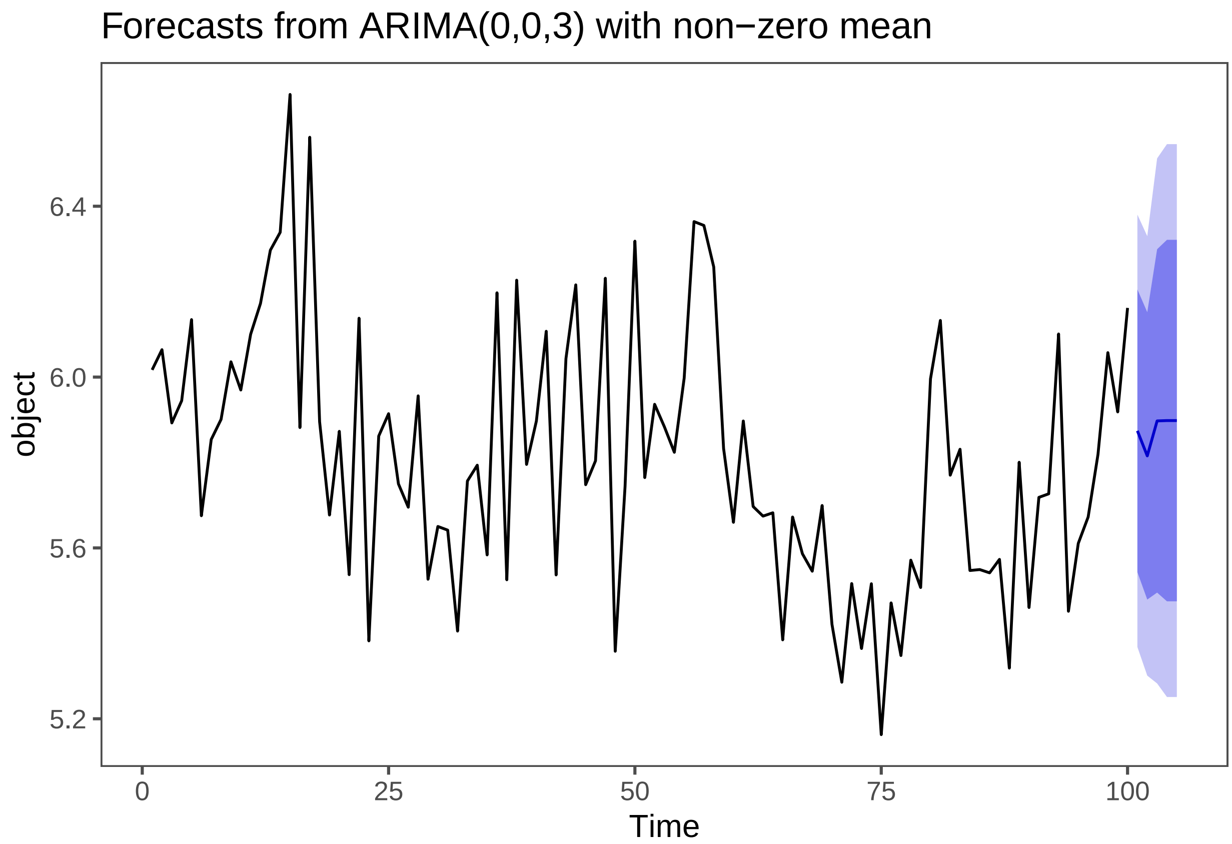
```
fc <- forecast(df$value, model = aa_model, h = h)

autoplot(fc) + theme_few()
```




```
autoplot(fc_list[[1]]) + theme_few()
```




```
autoplot(fc_list[[66]]) + theme_few()
```




```
autoplot(fc_list[[796]]) + theme_few()
```

Forecasts from ARIMA(0,0,3) with non−zero mean