Lecture 9 Code Examples

1 Time Series Packages

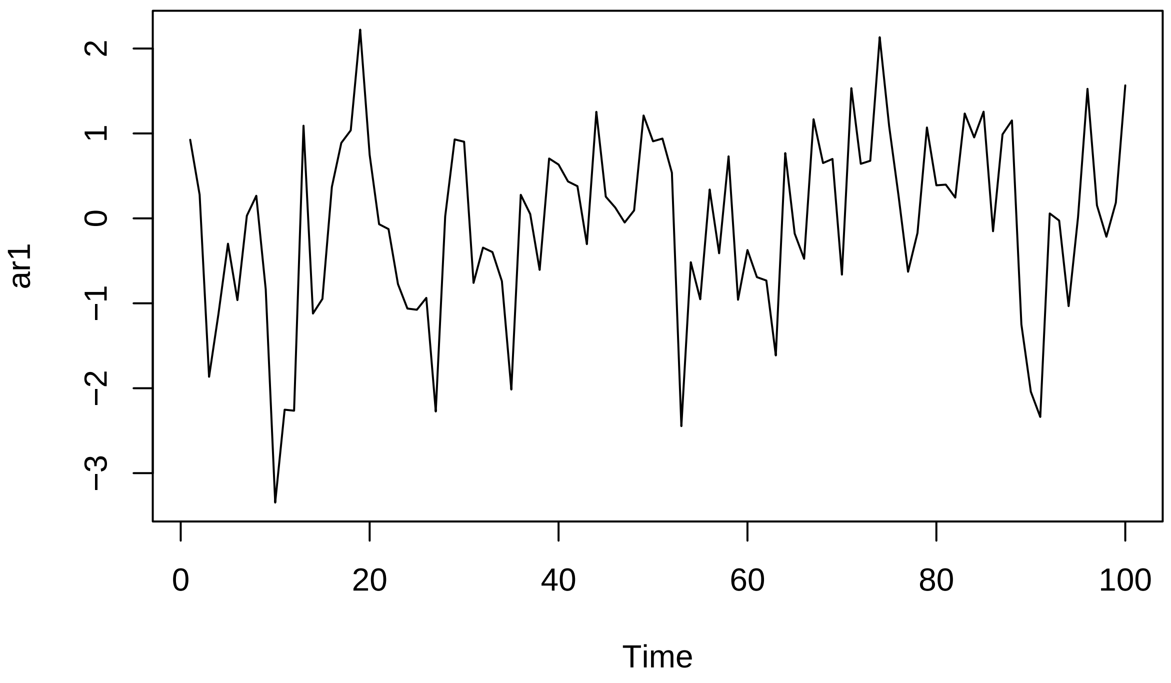
There is a vibrant open source community for data science in R, and for any data task, there will usually be many packages available. Time series analysis is no different. There are many packages available for computing ARIMA or Holt-Winters models, data wrangling, time series decomposition, etc. Be careful! Some of these packages may contain bugs! In order to find the most reputable packages, you should read the Time Series Analysis entry on CRAN task views.

2 ARIMA modeling on simulated data

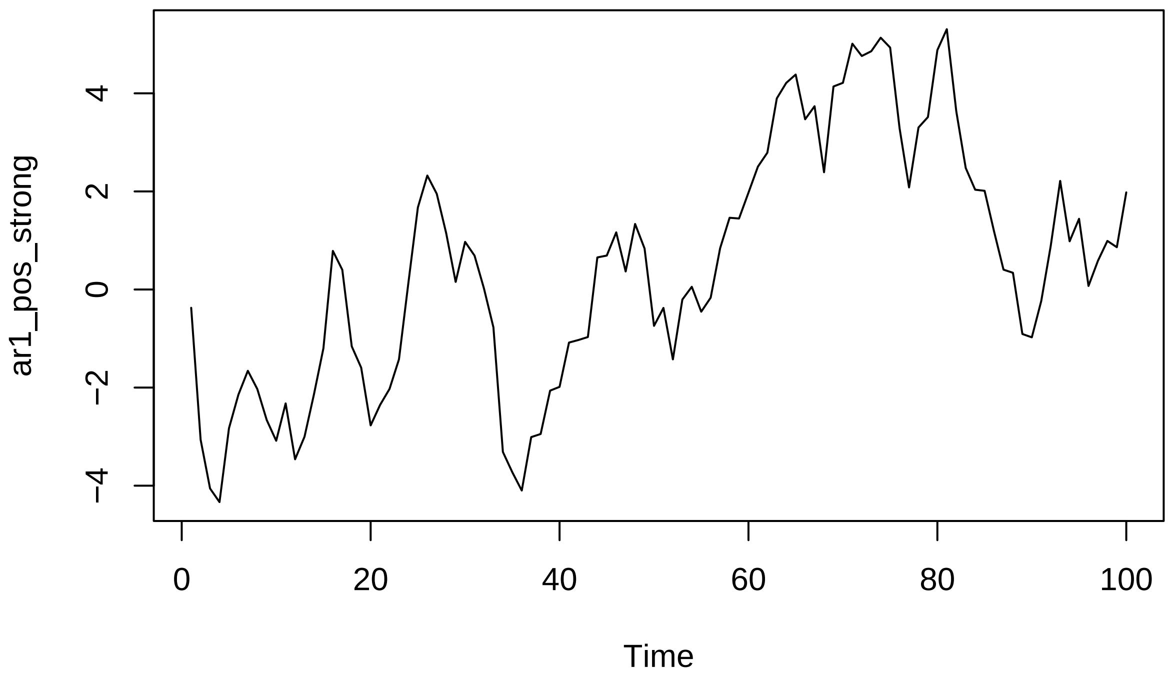
2.1 Generating the data

We use `arima.sim()` to generate data from ARIMA models.

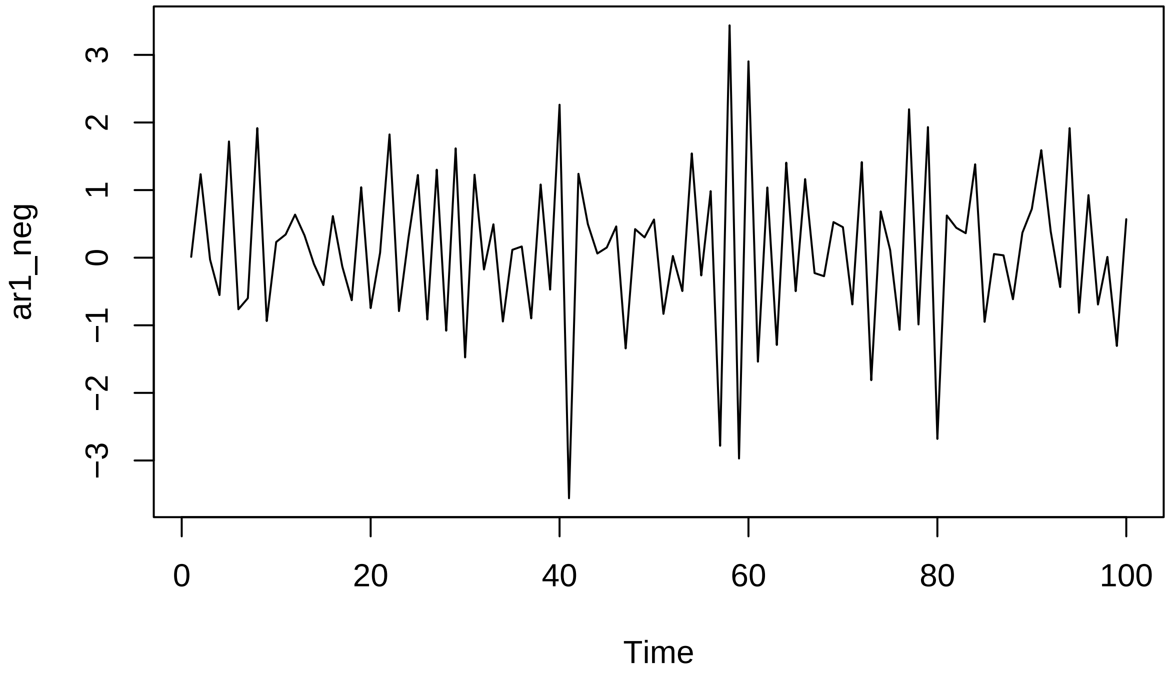
```r
set.seed(5209)
ar1 <- arima.sim(model = list(ar = c(0.5)), n = 100)
ar1_neg <- arima.sim(model = list(ar = c(-0.5)), n = 100)
ar1_pos_strong <- arima.sim(model = list(ar = c(0.9)), n = 100)

# par(mfrow = c(3, 1)) # Uncomment if you want subplots
ar1 |> plot()
```



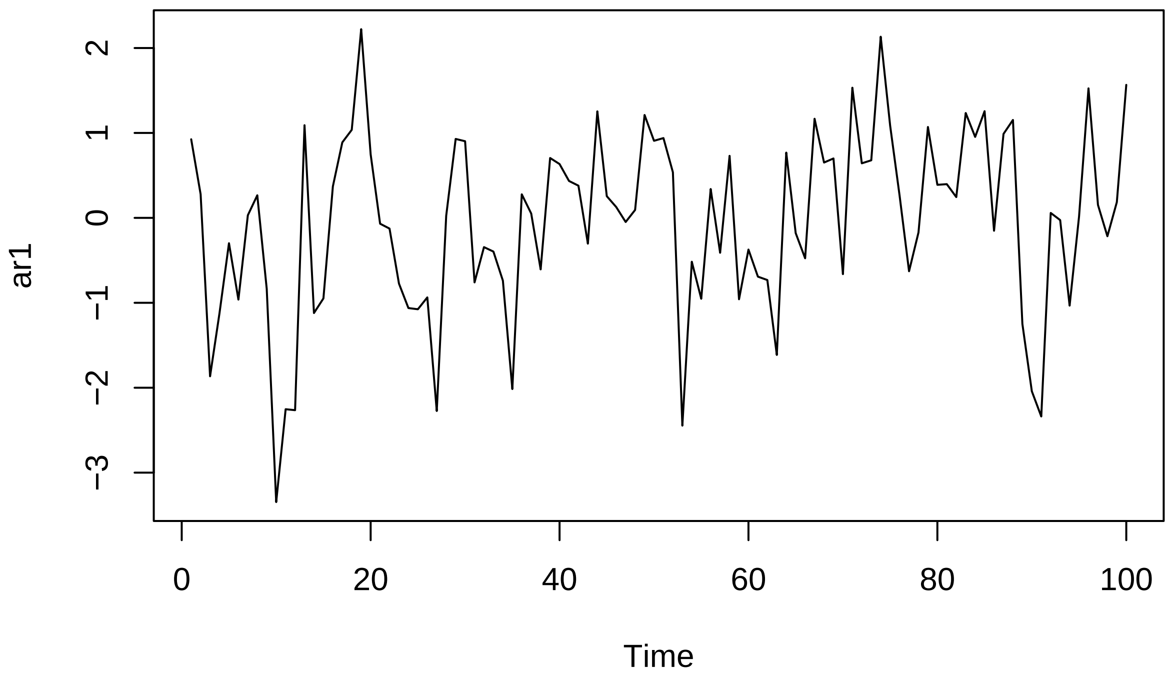
```r
ar1_pos_strong |> plot() # Larger coefficient gives a smoother series
```



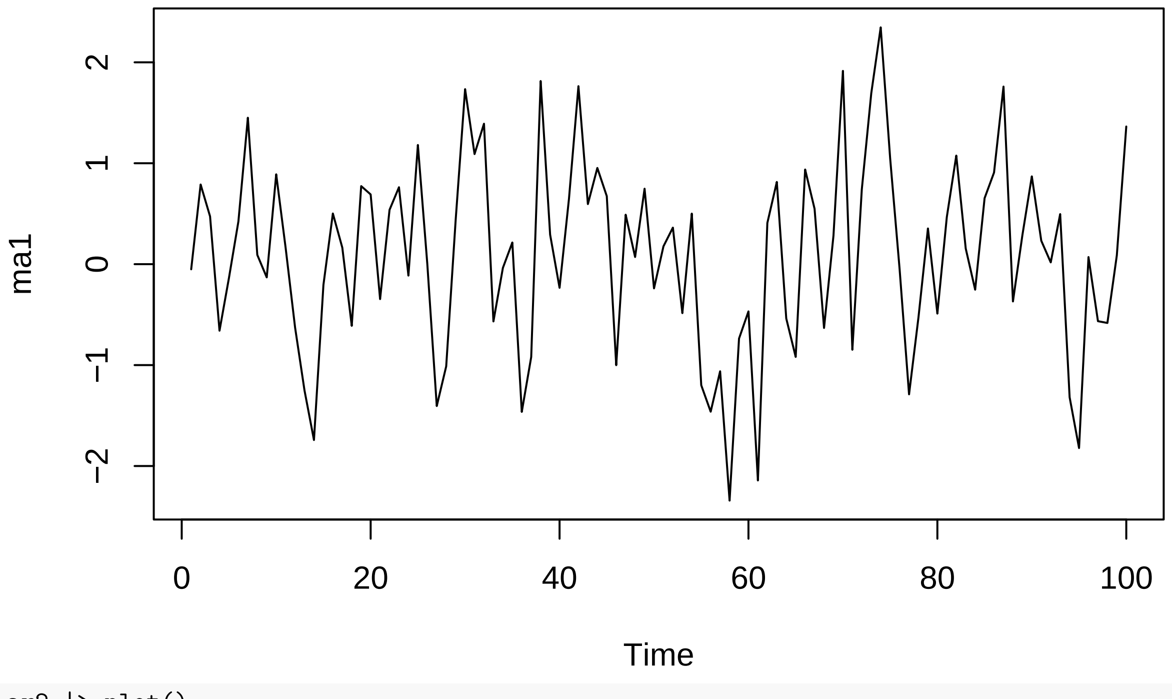
```
ar1_neg |> plot() # Negative coefficient leads to more deterministic oscillations
```



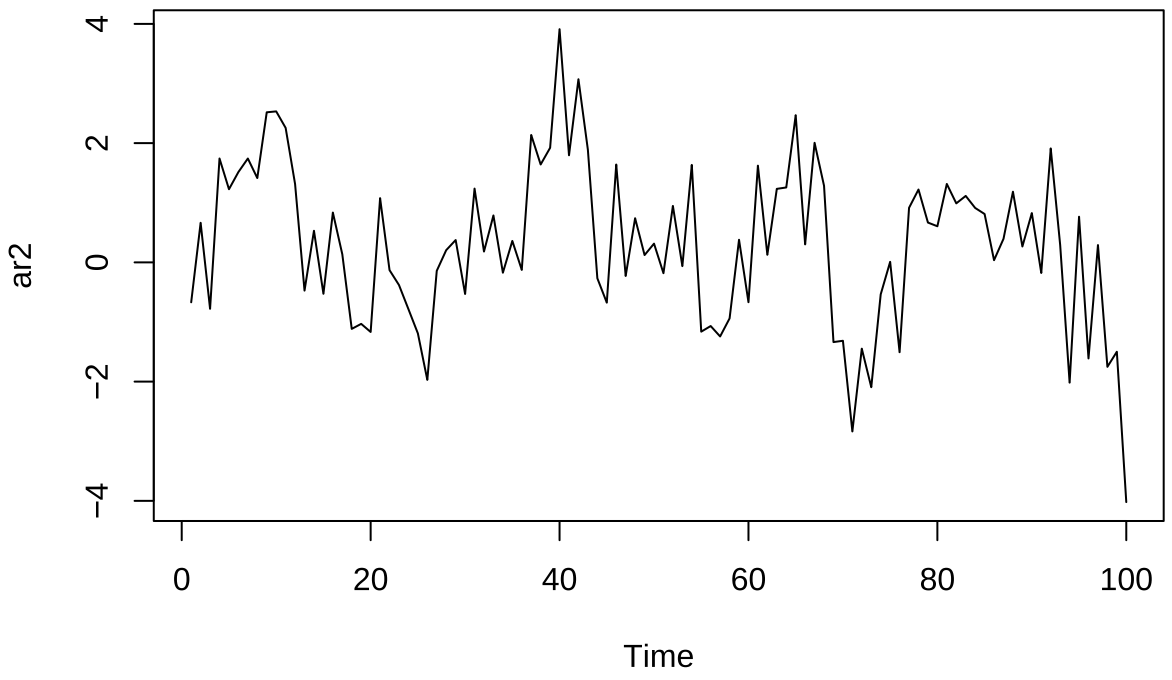
```
ma1 <- arima.sim(model = list(ma = c(0.5)), n = 100)
arma11 <- arima.sim(model = list(ar = c(0.5), ma = c((0.5))), n = 100)
ar2 <- arima.sim(model = list(ar = c(0.4, 0.2)), n = 100)

# par(mfrow = c(2, 2)) # Uncomment if you want subplots
ar1 |> plot()
```



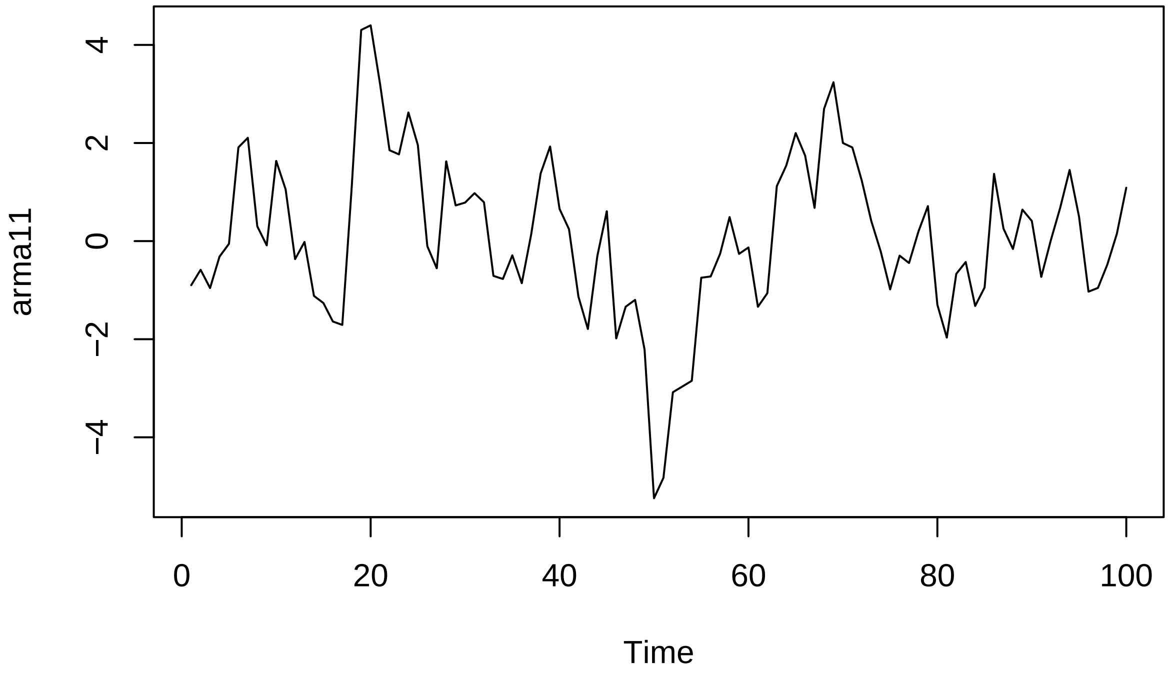
```r
ma1 |> plot()
```



```r
ar2 |> plot()
```



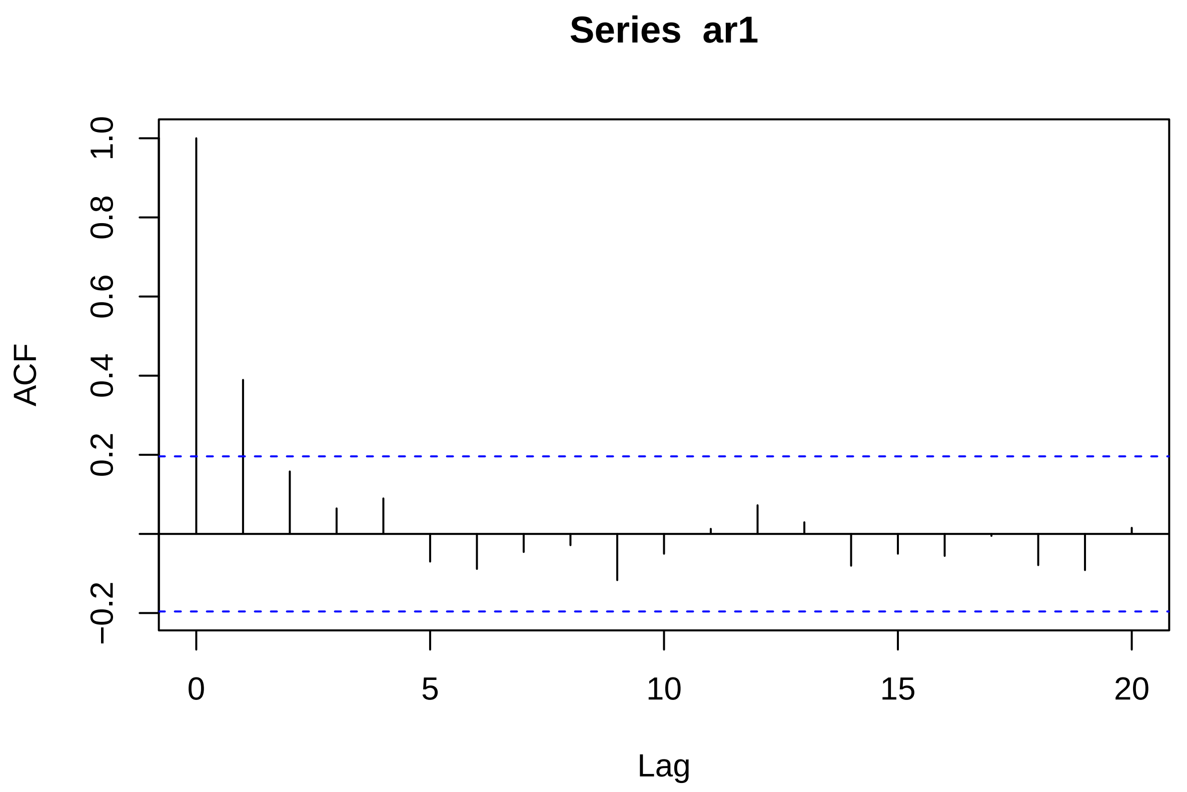
```
arma11 |> plot()
```



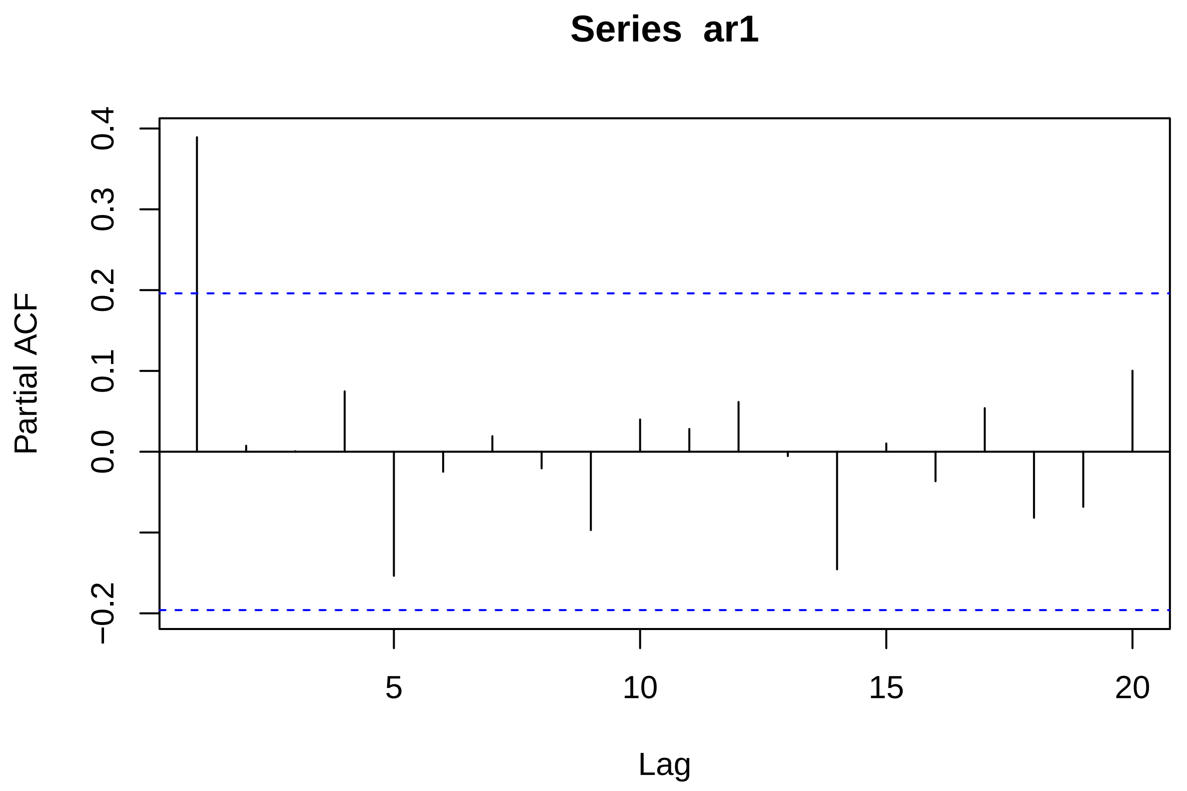
```
# Hard to distinguish the different series from the time plots
```

2.2 ACF and PACF plots

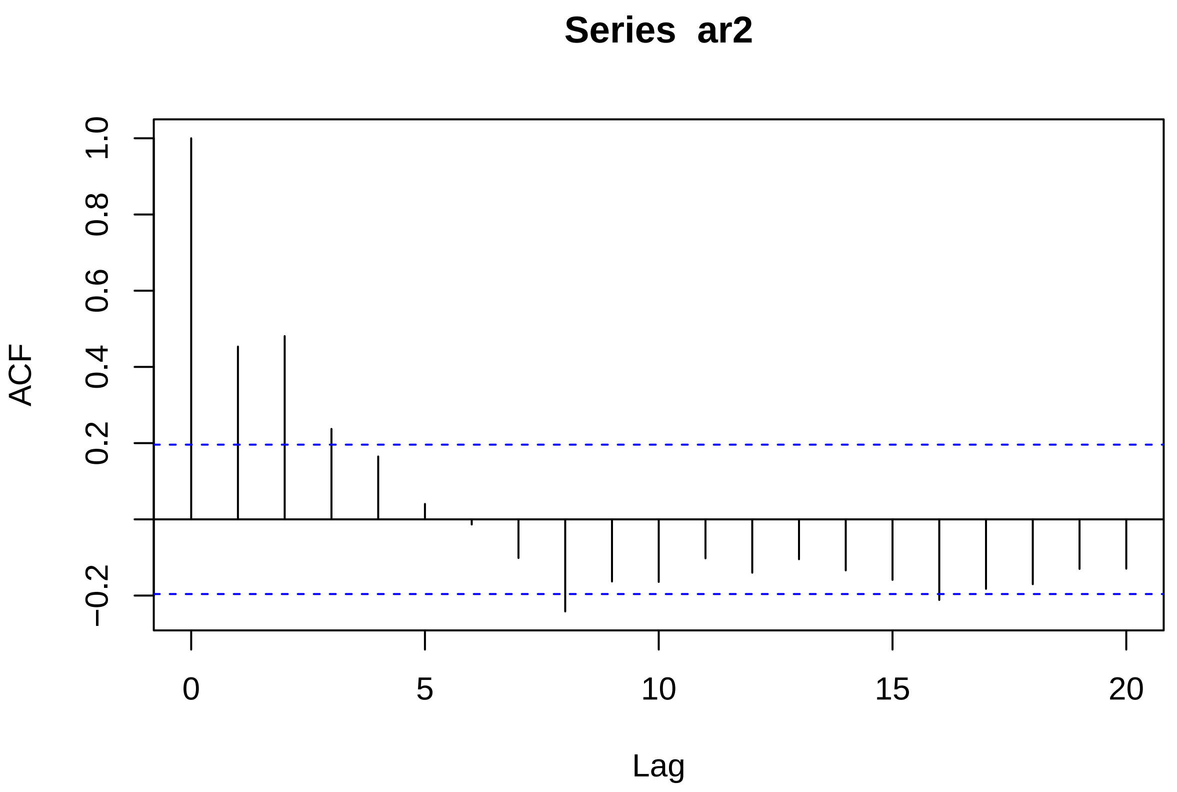
```
# par(mfrow = c(1, 2)) # Uncomment if you want subplots
ar1 |> acf()
```



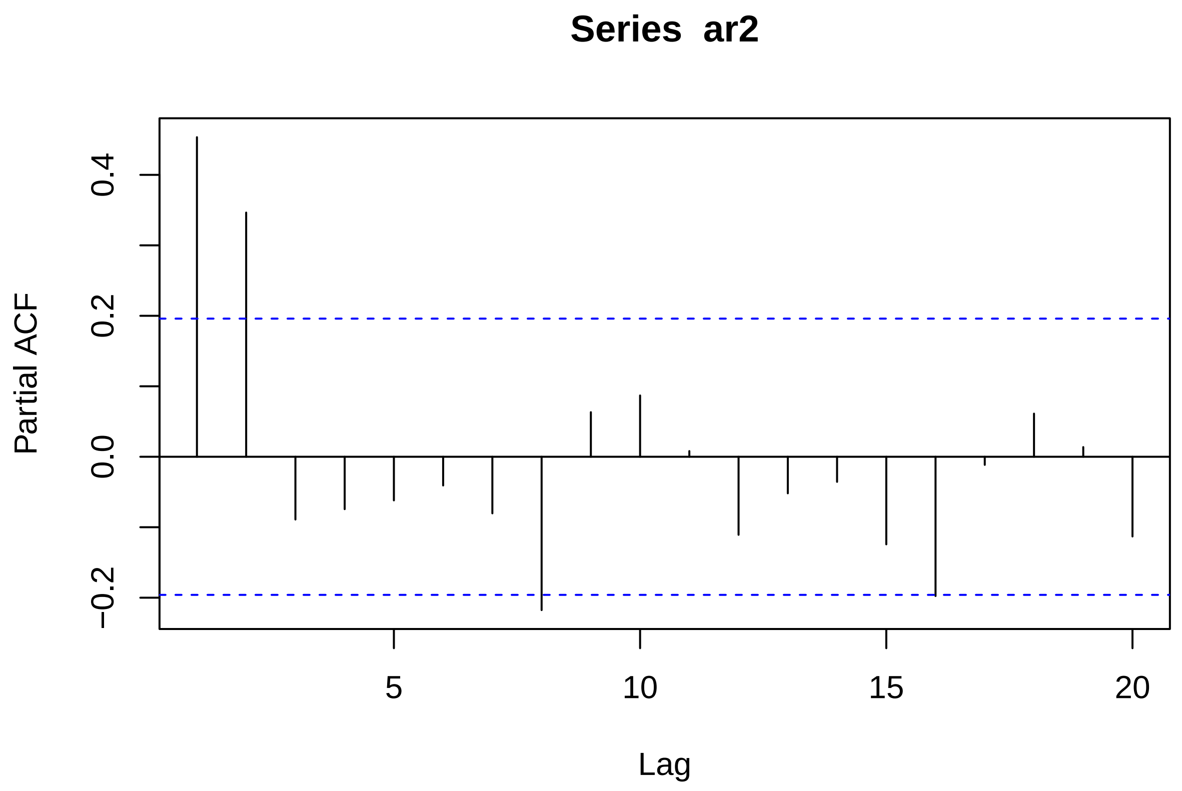
```
ar1 |> pacf()
```



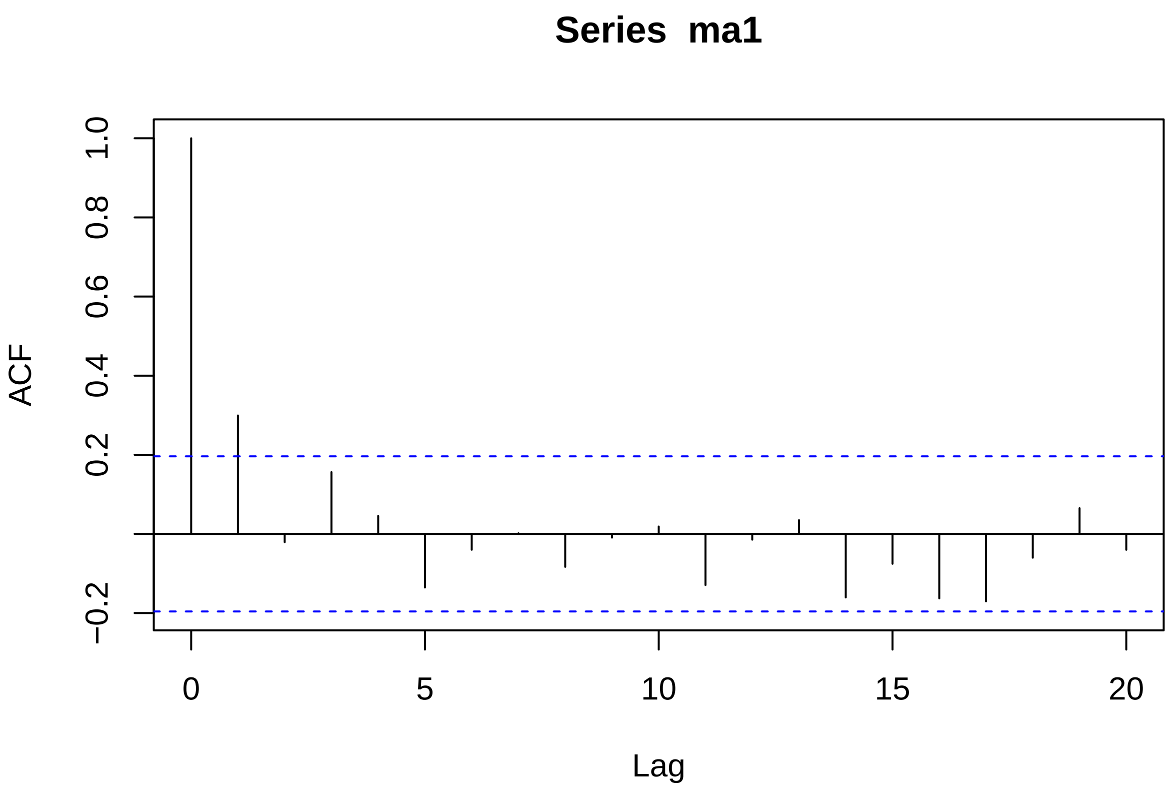
```
# par(mfrow = c(2, 2)) # Uncomment if you want subplots
ar2 |> acf()
```



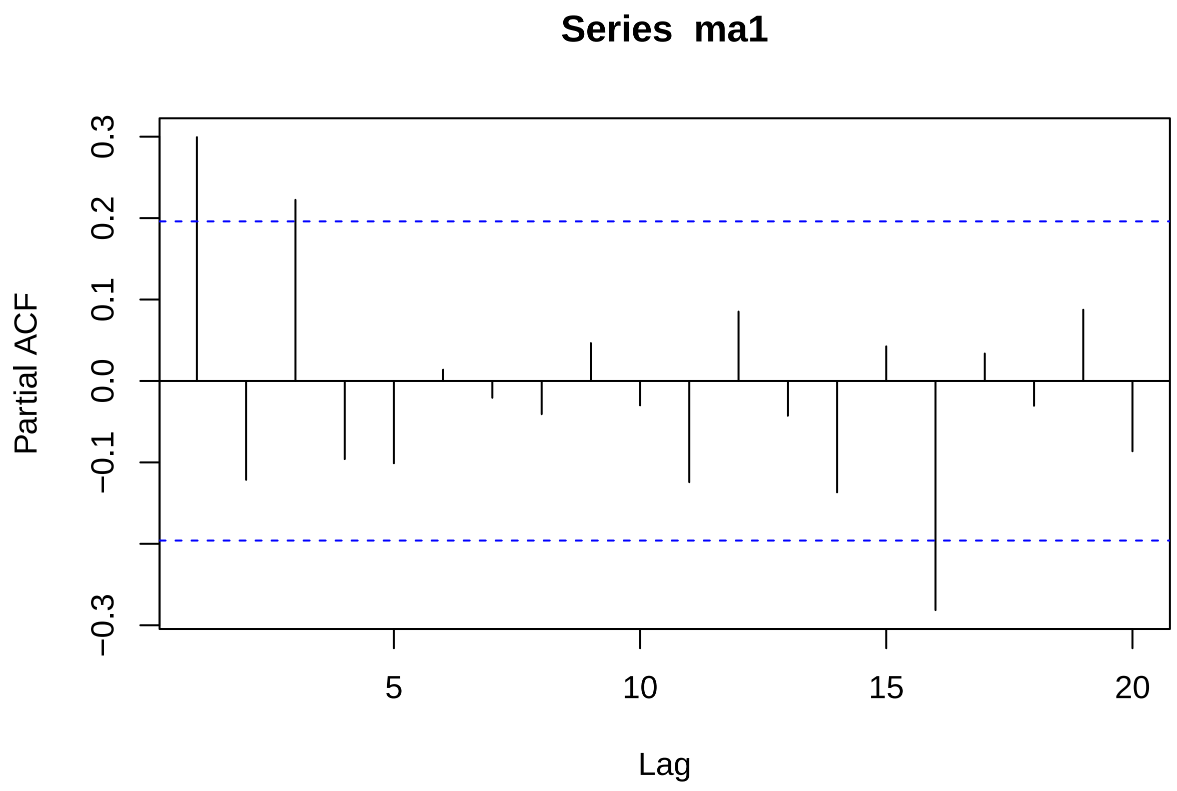
```r
ar2 |> pacf()
```



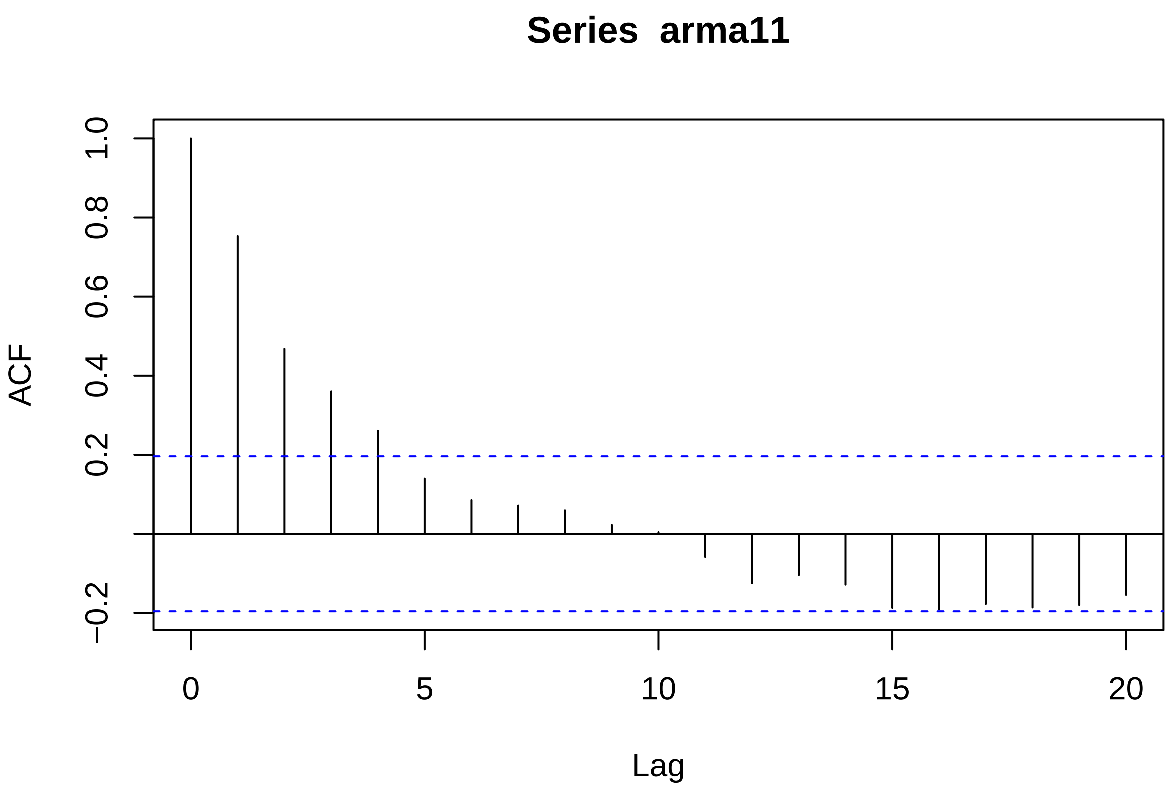
```r
ma1 |> acf()
```



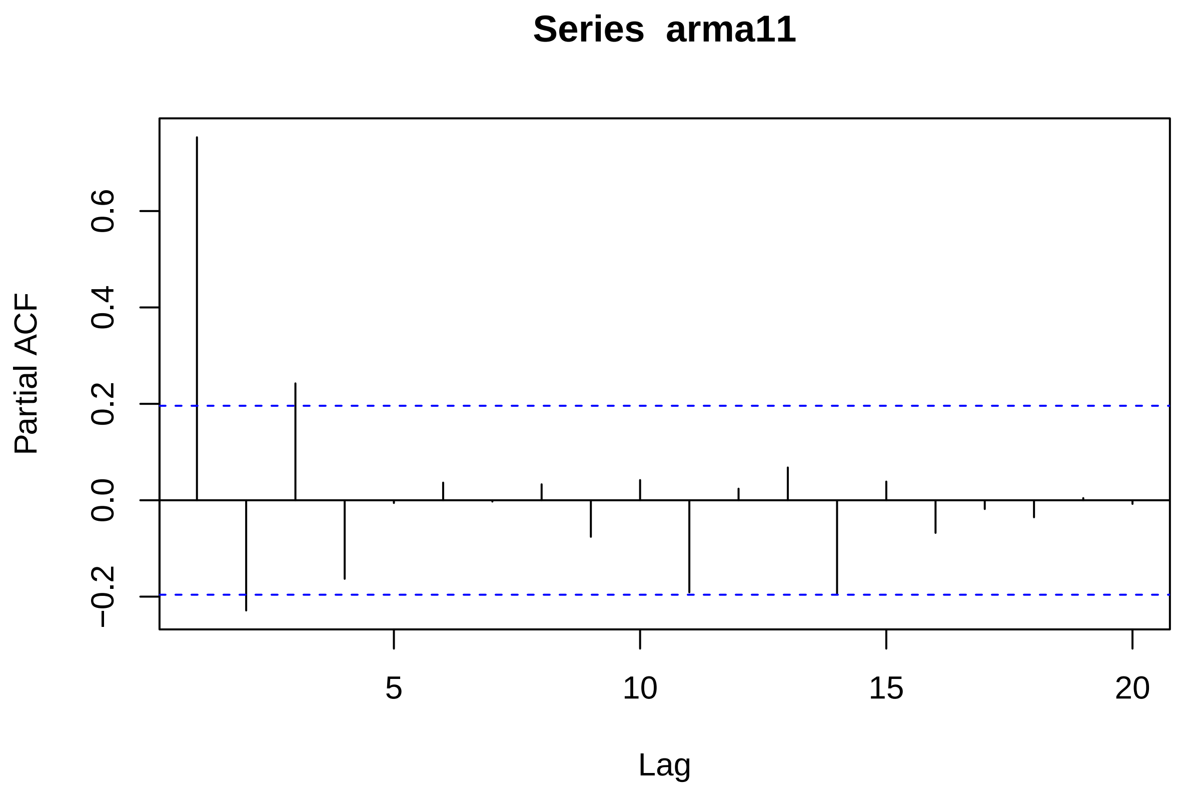
```
ma1 |> pacf()
```



```
# par(mfrow = c(1, 2)) # Uncomment if you want subplots
arma11 |> acf()
```



```
arma11 |> pacf()
```



##

Estimation and forecasting with ARMA

Base R, via the `stats` package, has implementations of both the Holt-Winters method as well as ARIMA method for forecasting. We will later use a more sophisticated package called `fable` that integrates better with `tidyverse`, but for now, we will illustrate how to use the base functions for some quick and dirty analysis.

For fitting an ARIMA model, we may use the `arima()` function. Inspecting the documentation, we see that it fits a model using maximum likelihood, although one can select the option of using conditional sum of squares instead. The `ar()` function fits an AR model, but on top of that, does model selection, i.e. it chooses the order of the AR model using AIC criterion (we will cover this during the next lecture.)

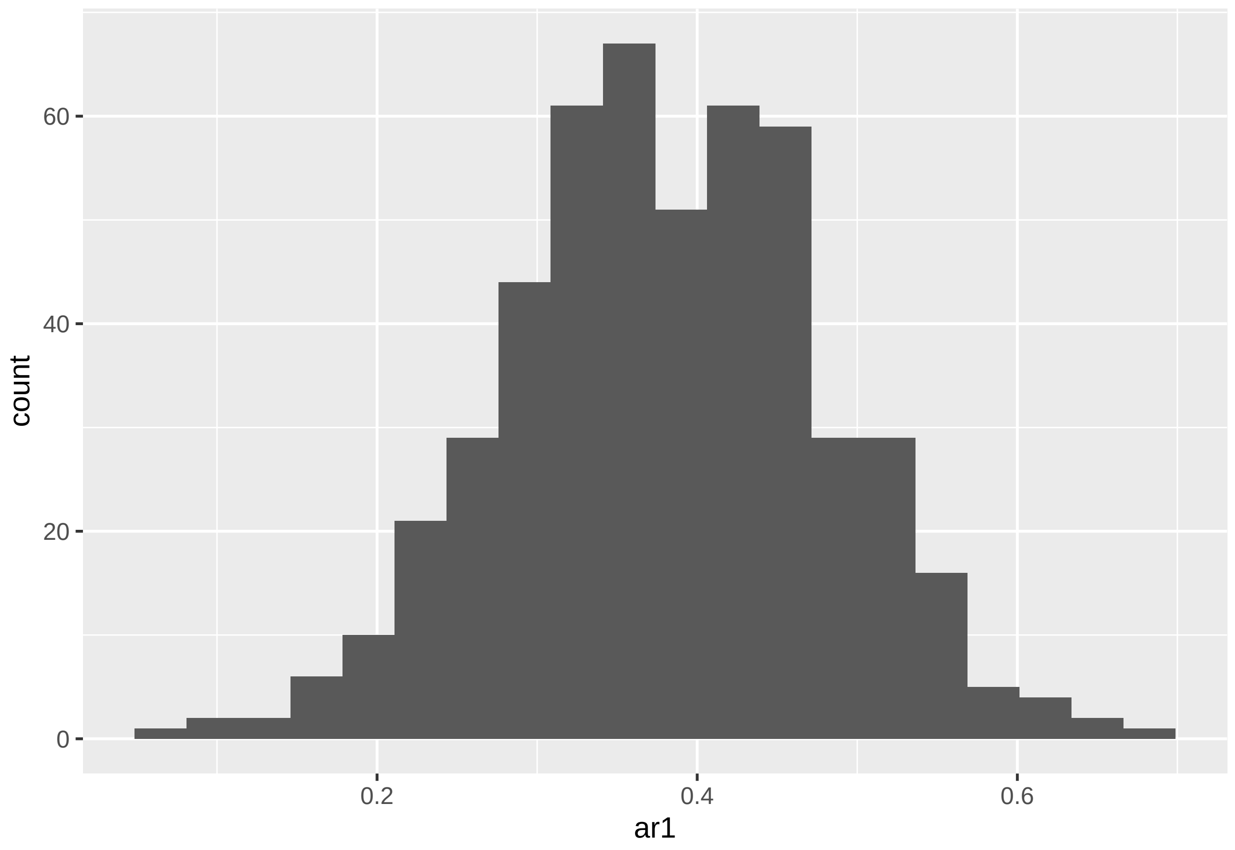
```
ar_fit <- arima(ar2, order = c(2, 0, 0)) # (0.4, 0.2)
ar_fit
```

```
##
## Call:
## arima(x = ar2, order = c(2, 0, 0))
##
## Coefficients:
##          ar1     ar2  intercept
##       0.3216  0.3904     0.1584
## s.e.  0.0961  0.0969     0.3664
##
## sigma^2 estimated as 1.165:  log likelihood = -149.84,  aic = 307.68
```

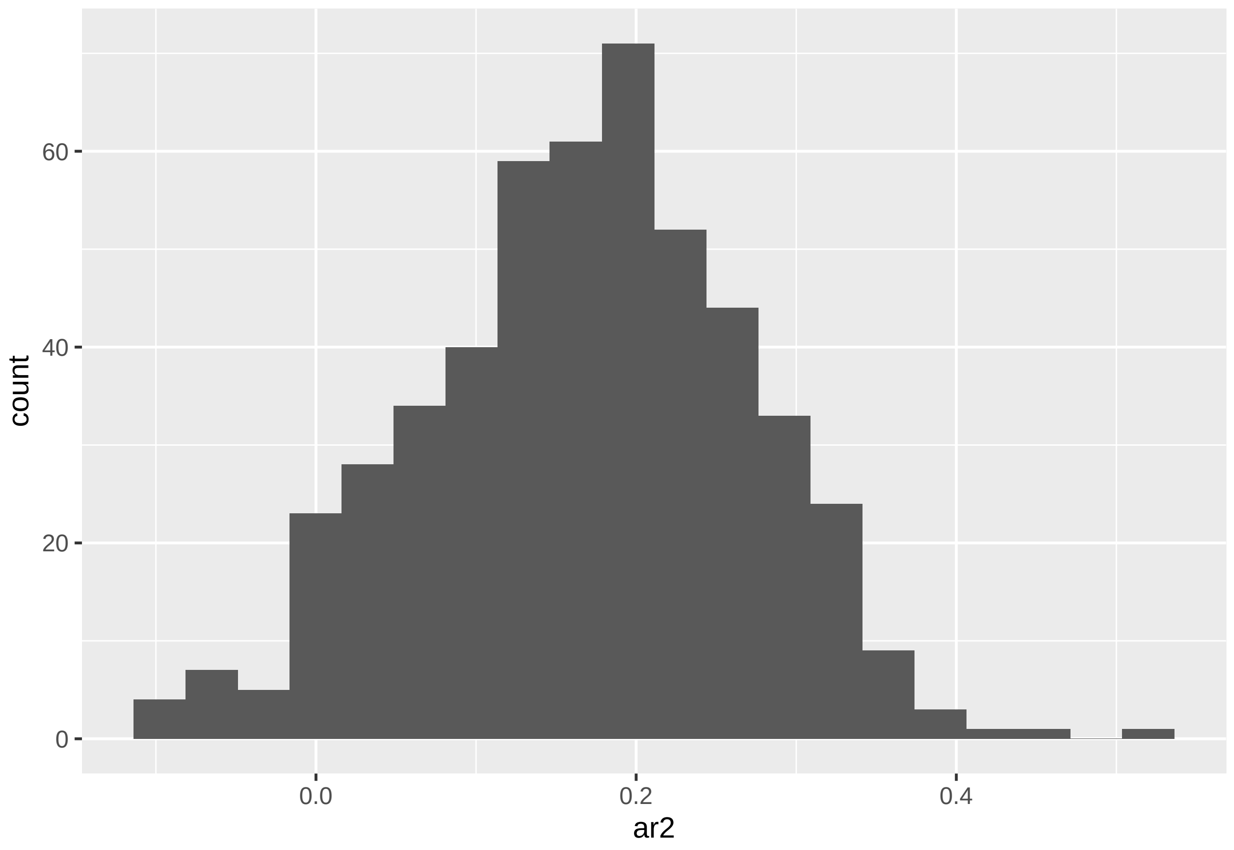
```
B <- 500
ar_coefs_ <- map(1:B, ~ arima(arima.sim(model = list(ar = c(0.4, 0.2)), n = 100),
                              order = c(2, 0, 0))$coef) |>
  transpose() |>
  map(unlist) |>
  as.tibble()
```

```
## Warning: `as.tibble()` was deprecated in tibble 2.0.0.
## i Please use `as_tibble()` instead.
## i The signature and semantics have changed, see `?as_tibble`.
```

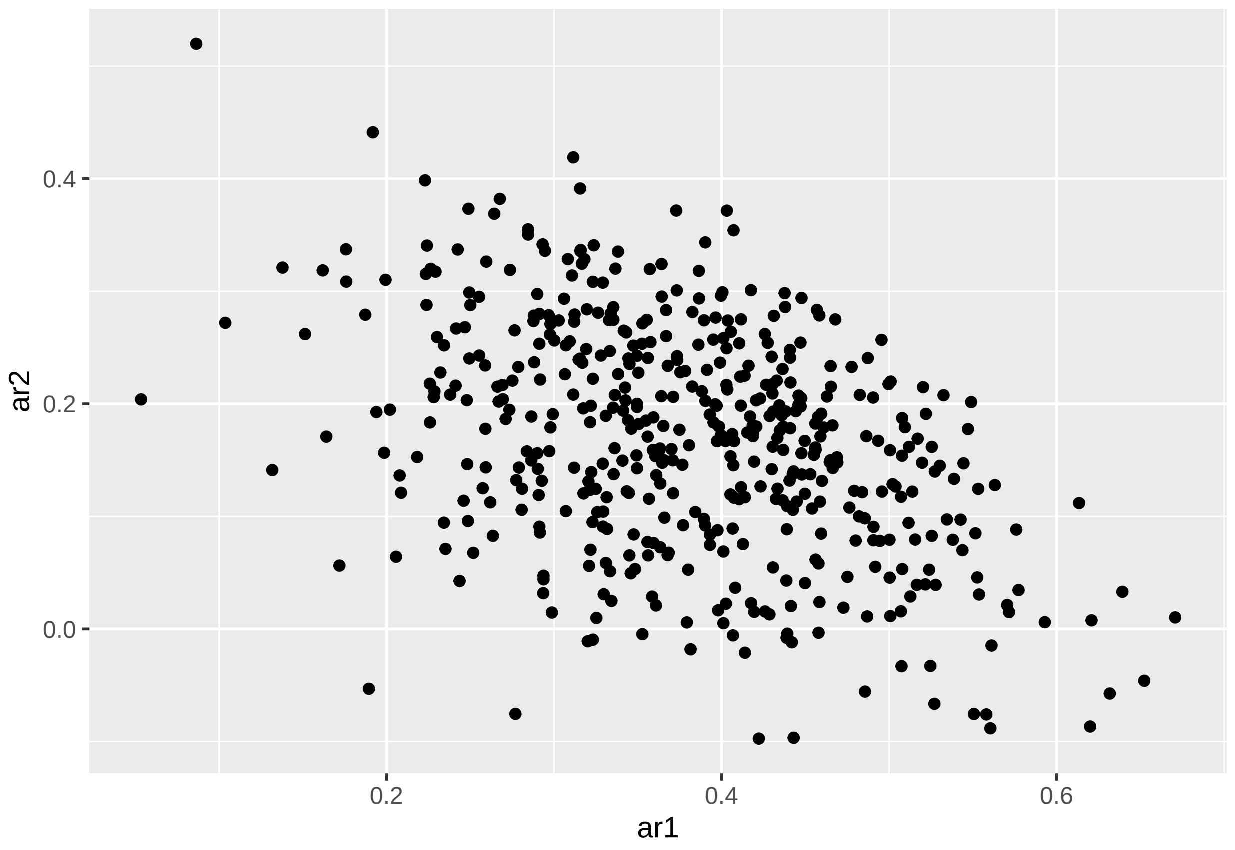
```
ggplot(ar_coefs_) + geom_histogram(aes(x = ar1), bins = 20)
```



```
ggplot(ar_coefs_) + geom_histogram(aes(x = ar2), bins = 20)
```



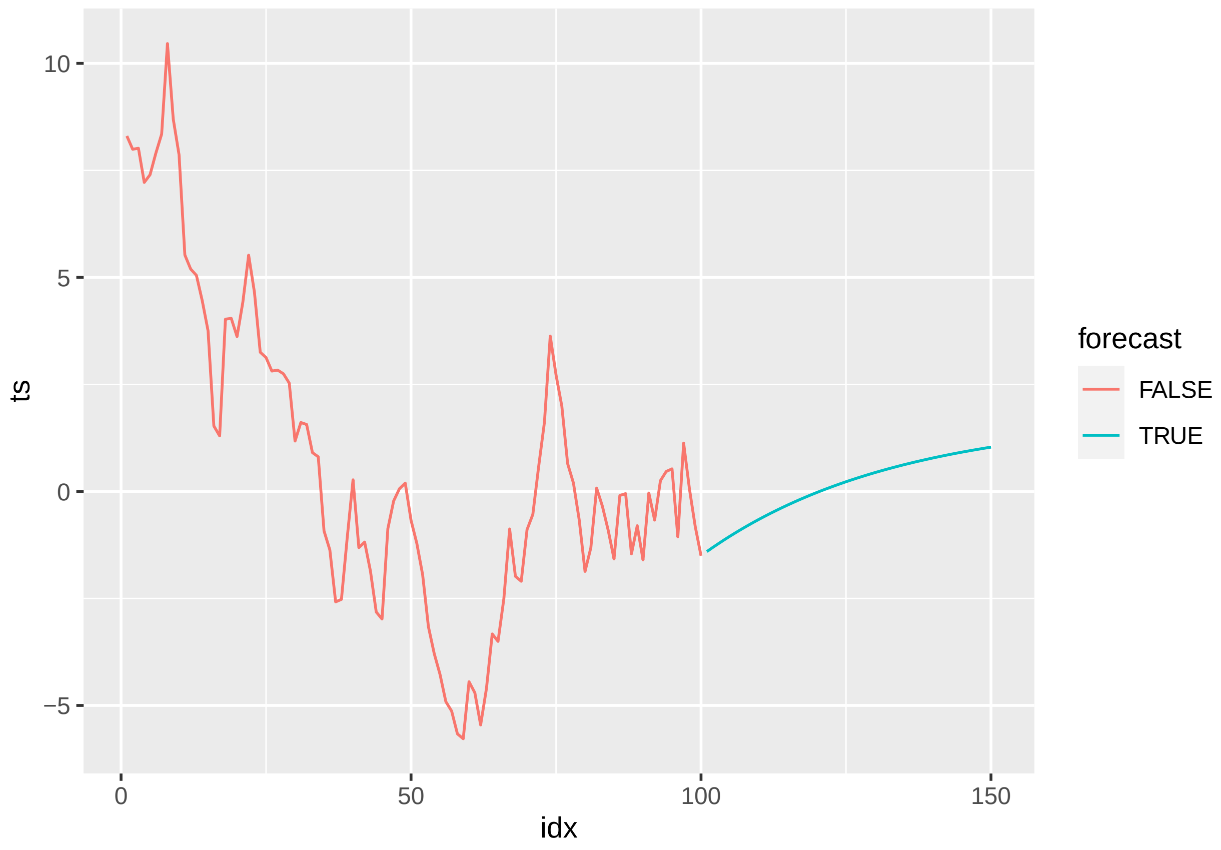
```
ggplot(ar_coefs_) + geom_point(aes(x = ar1, y = ar2))
```



```r
# set.seed(5209)
ar_data <- arima.sim(model = list(ar = c(0.99)), n = 100)
ar_fit <- arima(ar_data, order = c(1, 0, 0))
ar_fit
```

```
##
## Call:
## arima(x = ar_data, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
##       0.9700     1.7436
## s.e.  0.0234     2.6845
##
## sigma^2 estimated as 1.025:  log likelihood = -144.56,  aic = 295.12
```

```r
ar_forecast <- ar_fit %>% predict(n.ahead = 50)
ts_df <- tibble(idx = 1:150, ts = c(ar_data, ar_forecast$pred), forecast = idx > 100)
ts_df %>% ggplot() + geom_line(aes(x = idx, y = ts, color = forecast))
```



3 Real-world time series analysis

We work with a United States Energy Consumption dataset that can be found on Kaggle. The original dataset measures the hourly energy consumption, measured in Megawatts, by customers of the American Electric Power Company between 2004-10-01 and 2018-08-03. This gives more than 12,000 measurements, so we first compress the data by summing over the measurements for each day. We also convert the data into a `tsibble` object. This is a convenient data structure that is able to contain multiple time series, which makes fitting multiple models and cross-validation much more convenient.

```
all_energy <- read_csv("AEP_hourly.csv") |>
  group_by(Datetime) |>
  summarise(AEP_MW = mean(AEP_MW))

daily_energy <- all_energy |>
  mutate(date = date(Datetime)) |>
  group_by(date) |>
  summarise(energy_use = 24*mean(AEP_MW)) |>
  as_tsibble()
```

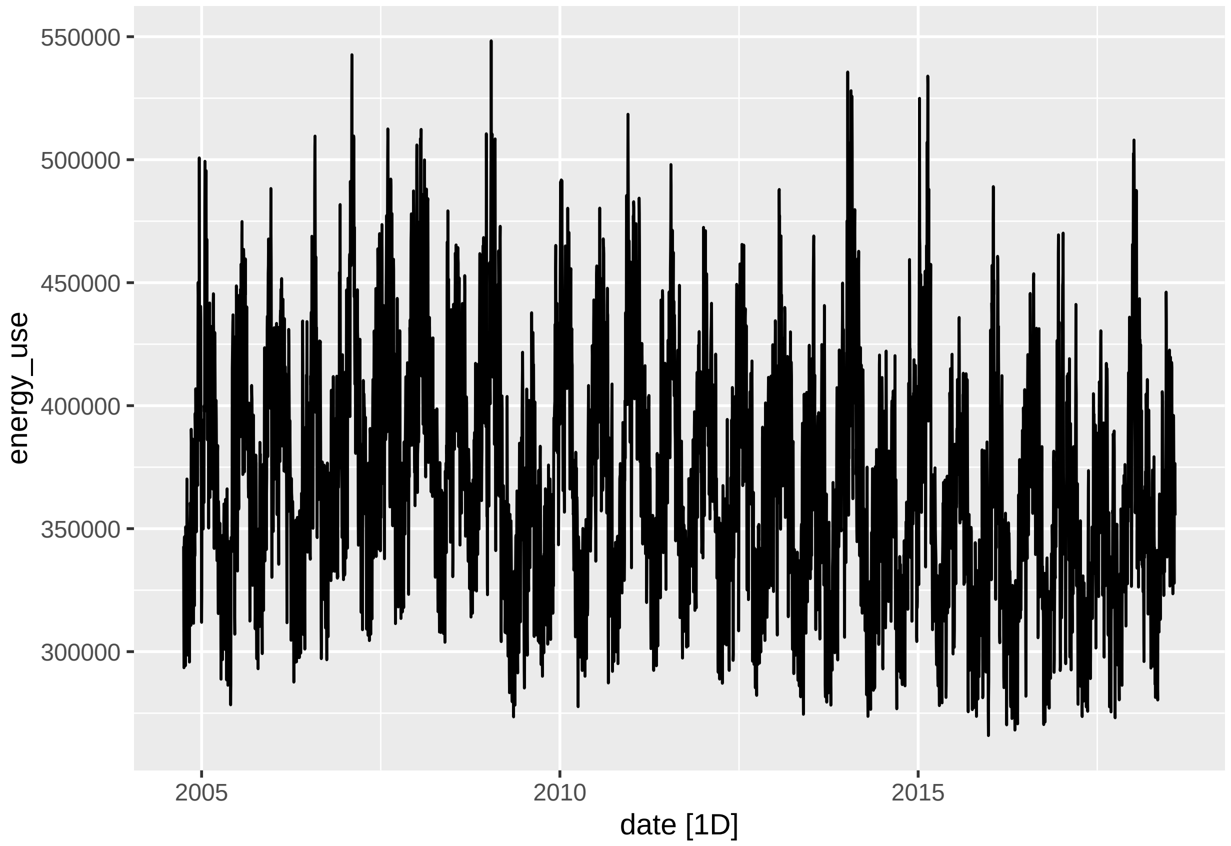
3.1 EDA

3.1.1 Time plot

The time series is too long, so we can filter to get a 2 year window before doing a time plot. From this plot, we observe that there is a yearly seasonal component.

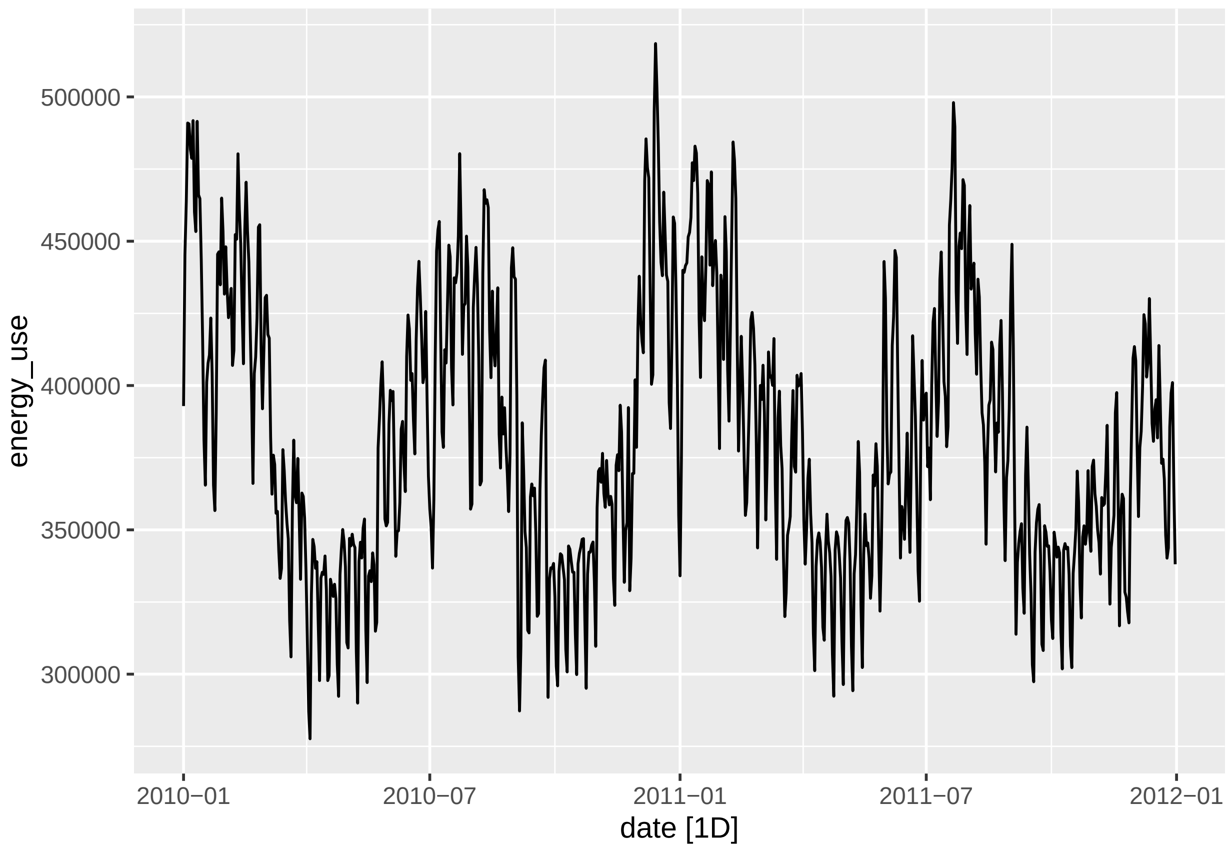
```
daily_energy |> autoplot()
```

```
## Plot variable not specified, automatically selected `.vars = energy_use`
```



```
daily_energy |>
  filter_index("2010" ~ "2011") |>
  autoplot()
```

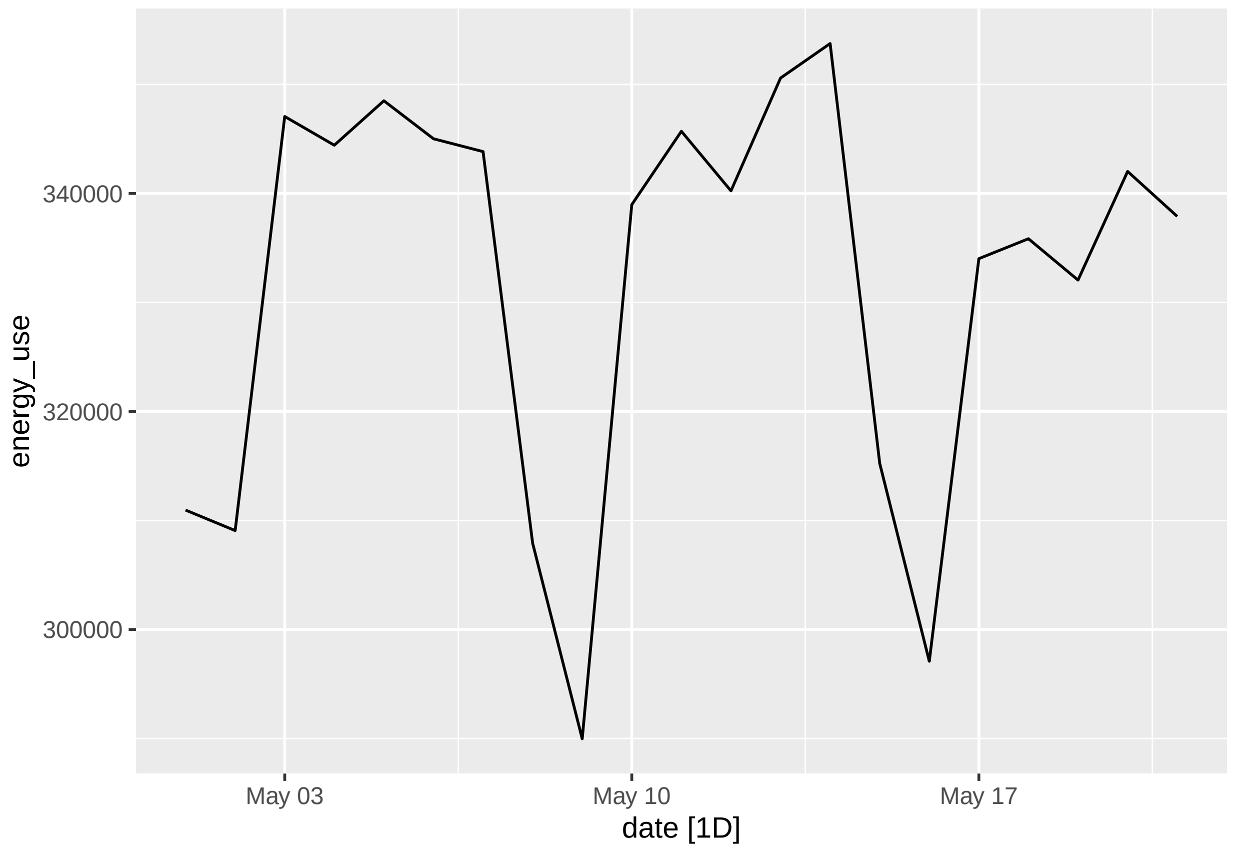
```
## Plot variable not specified, automatically selected `.vars = energy_use`
```



There also seems to be some weekly seasonality, but this is not easy to see from just a few periods.

```
# daily_energy |> autoplot()

daily_energy |>
  filter_index("2010-05-01" ~ "2010-05-21") |>
  autoplot()
```

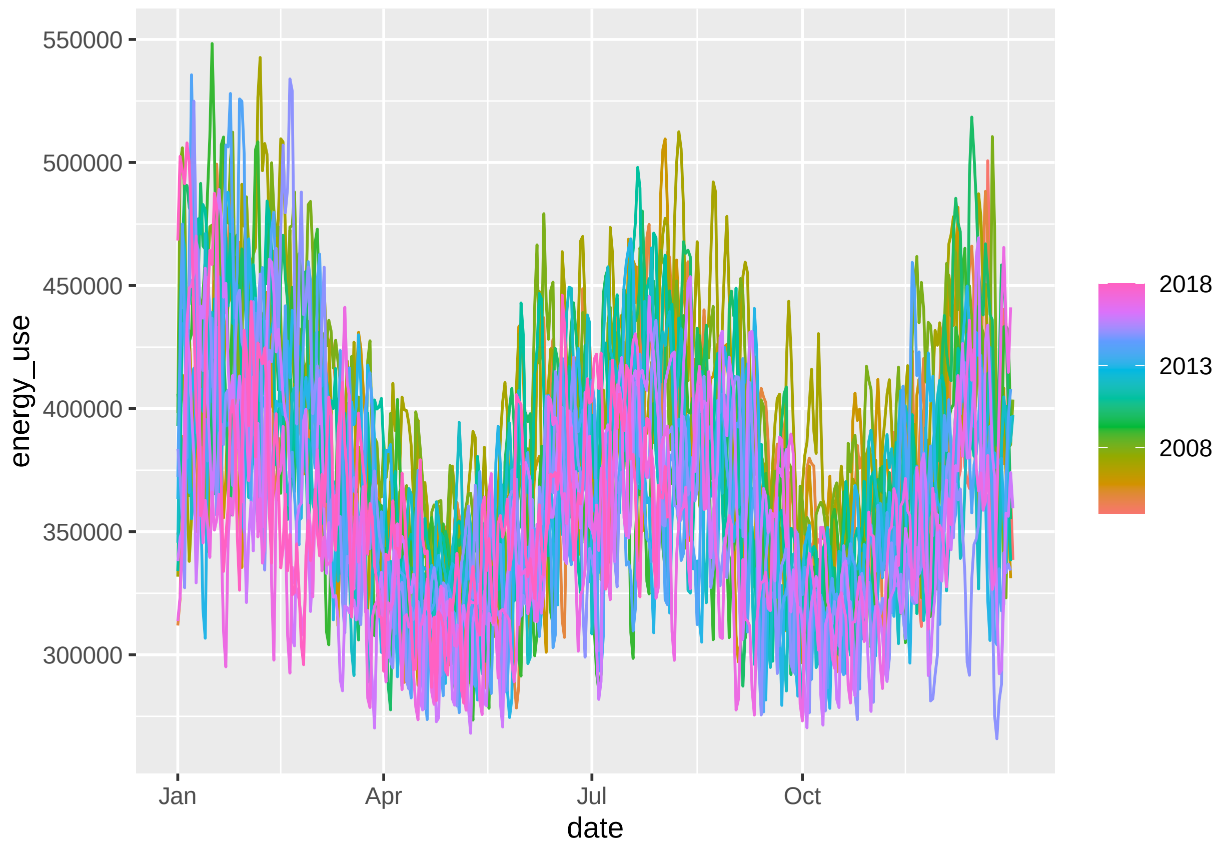
```
## Plot variable not specified, automatically selected `.vars = energy_use`
```



3.1.2 Seasonal plots

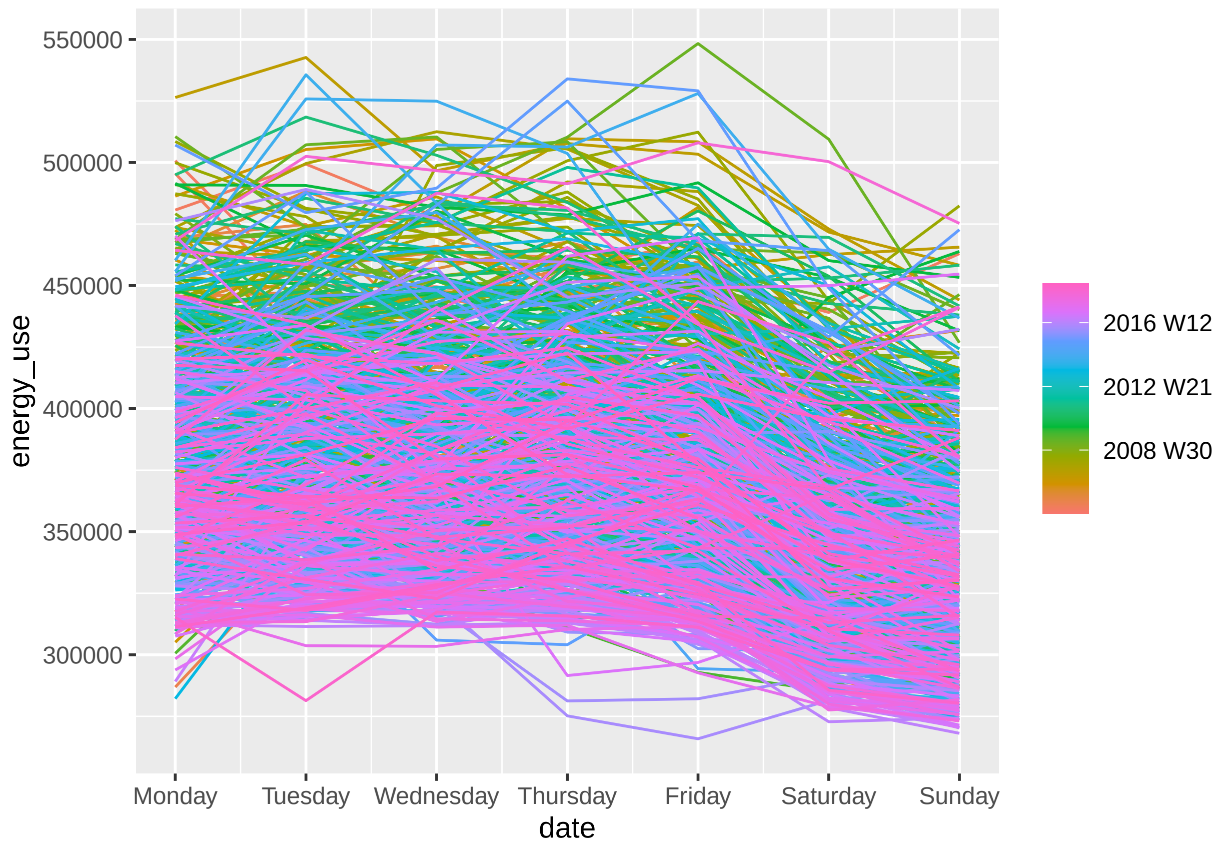
```
# daily_energy |> autoplot()

daily_energy |>
  gg_season(period = "year")
```

```
## Plot variable not specified, automatically selected `y = energy_use`
```



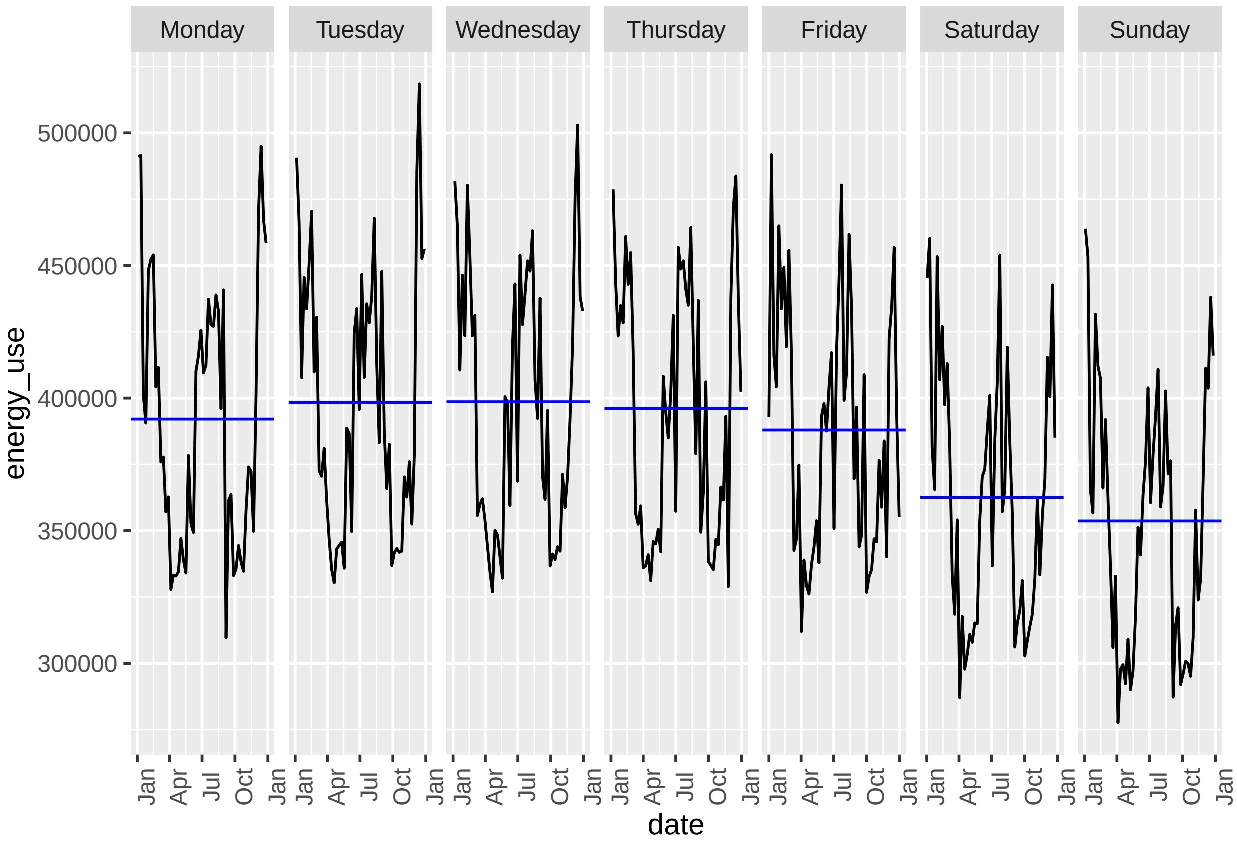
```
daily_energy |>
  gg_season(period = "week")
```

```
## Plot variable not specified, automatically selected `y = energy_use`
```



```r
daily_energy |>
  filter_index("2010") |>
  gg_subseries(period = "week")
```

```
## Plot variable not specified, automatically selected `y = energy_use`
```

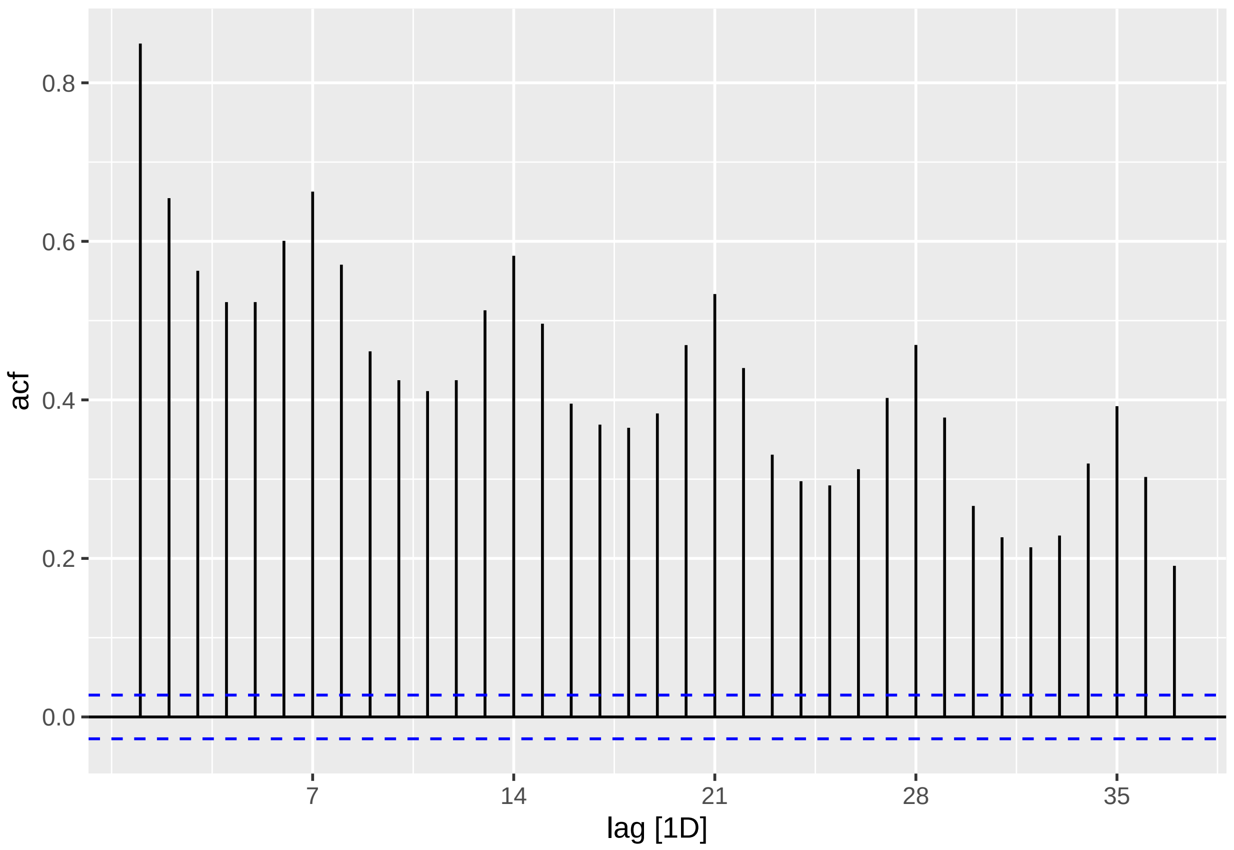


### ACF, PACF and lag plots

The ACF and PACF plot show significant values for a large number of lags. This is symptomatic of a non-stationary time series.

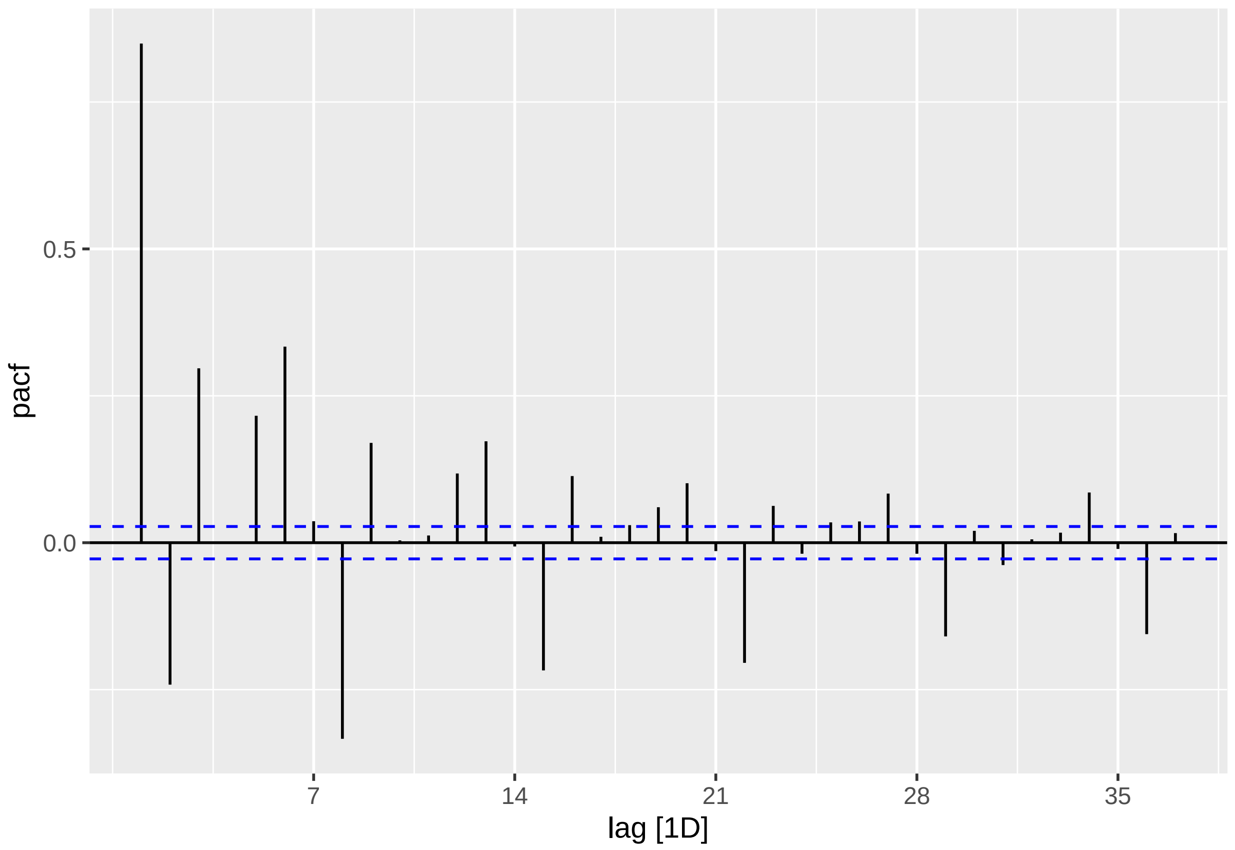
```
daily_energy |>
  ACF() |>
  autoplot()
```

```
## Response variable not specified, automatically selected `var = energy_use`
```



```
daily_energy |>
  PACF() |>
  autoplot()
```

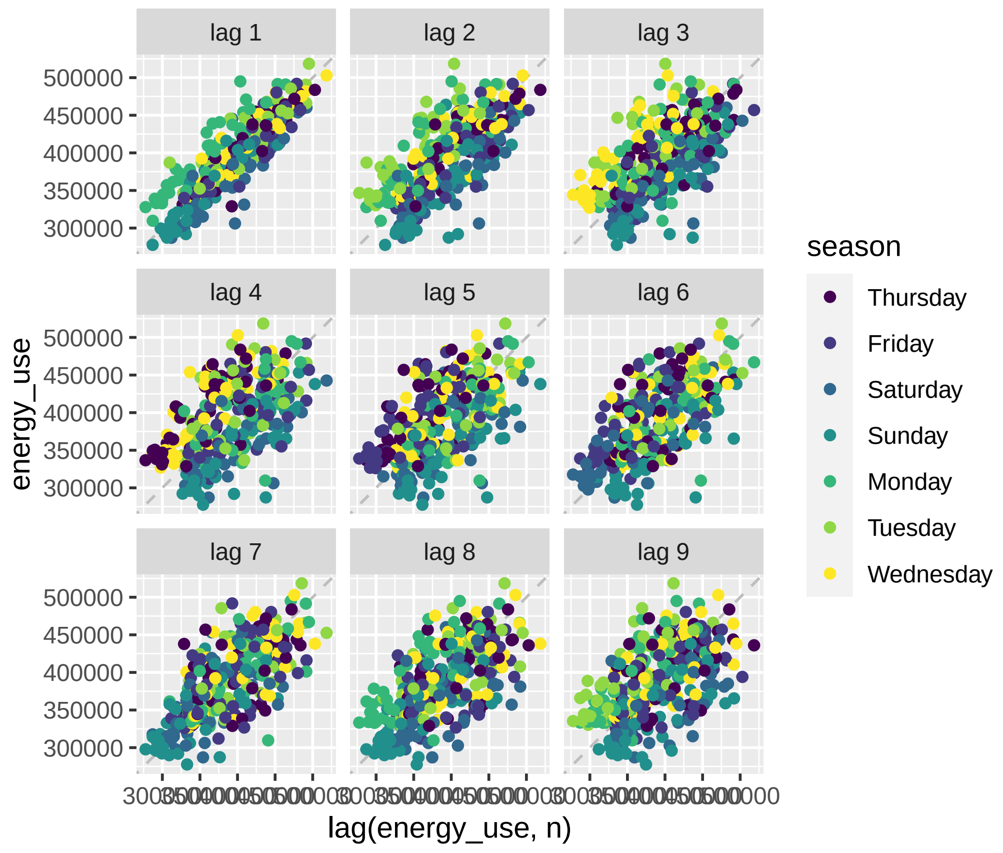
```
## Response variable not specified, automatically selected `var = energy_use`
```



This can be further validated using a lag plot. The lag plot allows us to detect nonlinear dependencies with lagged regressors.

```
daily_energy |>
  filter_index("2010") |>
  gg_lag(geom = "point")
```

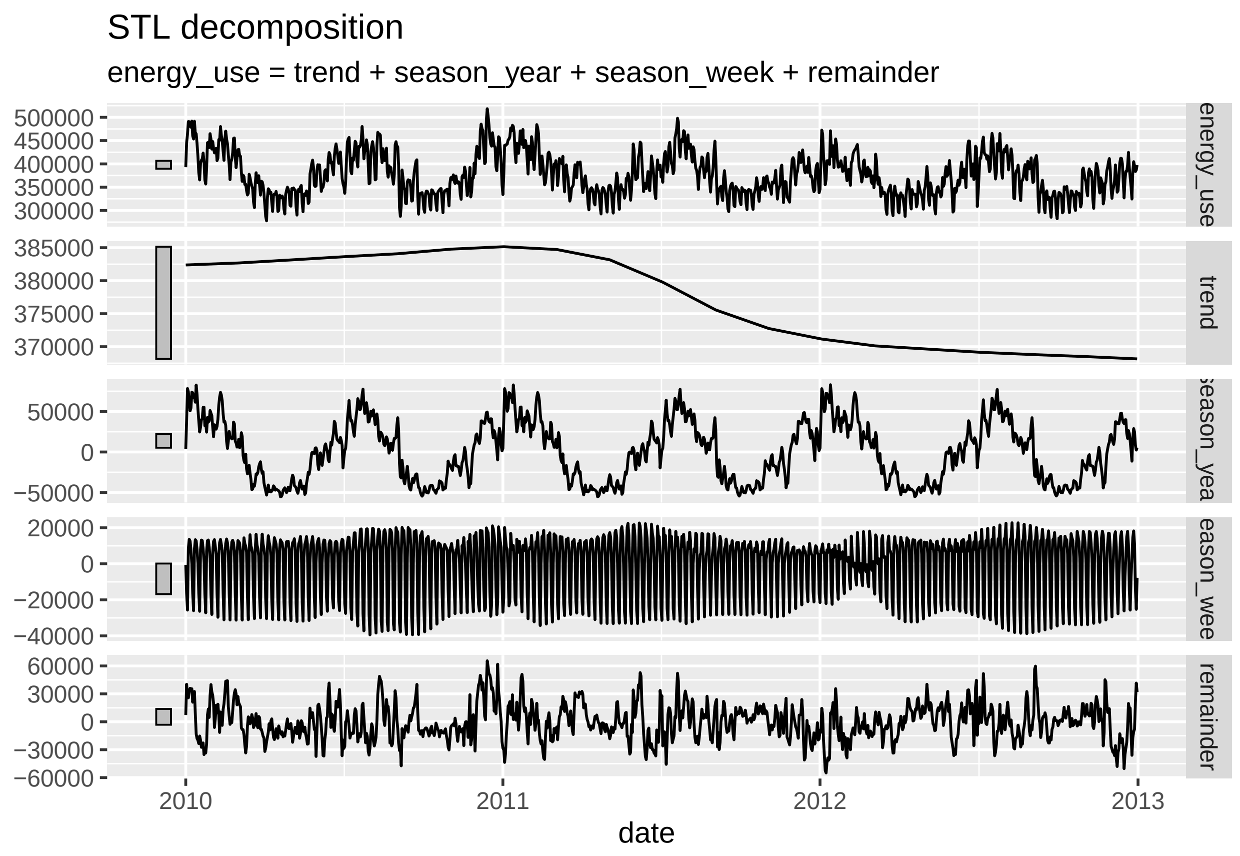
```
## Plot variable not specified, automatically selected `y = energy_use`
```



## Decomposition

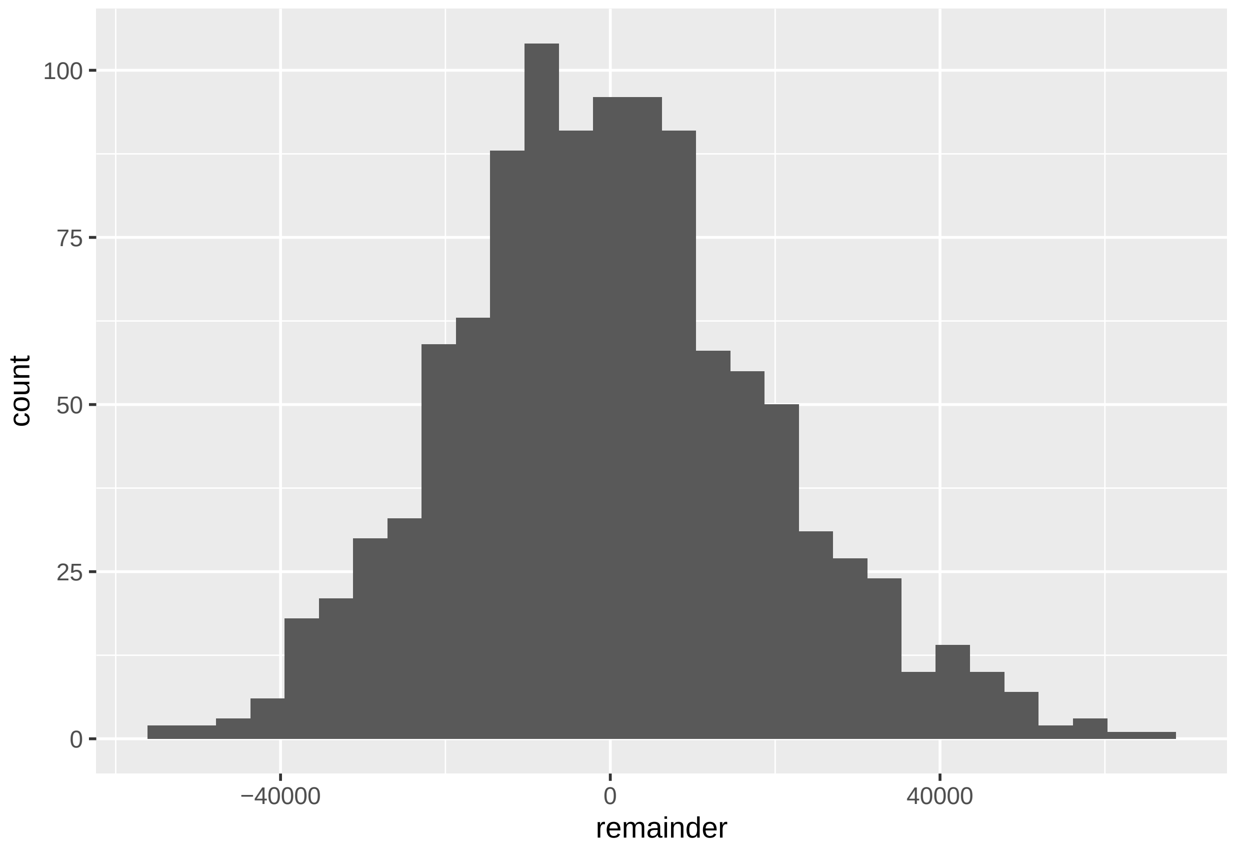
If we would like to model the time series dataset with ARMA, then we should remove the trend and seasonality to get a stationary time series. One can do this using temporal differencing, which is the I in ARIMA. Another way is to directly estimate the trend and seasonality components. In this course, we have learnt the classical decomposition. There is another decomposition algorithm called STL decomposition that is able to automatically handle the multi-periodic seasonal behavior we have in this time series.

```
energy_decomp <- daily_energy |>
  filter_index("2010" ~ "2012") |>
  model(STL(energy_use)) |>
  components()

energy_decomp |> autoplot()
```

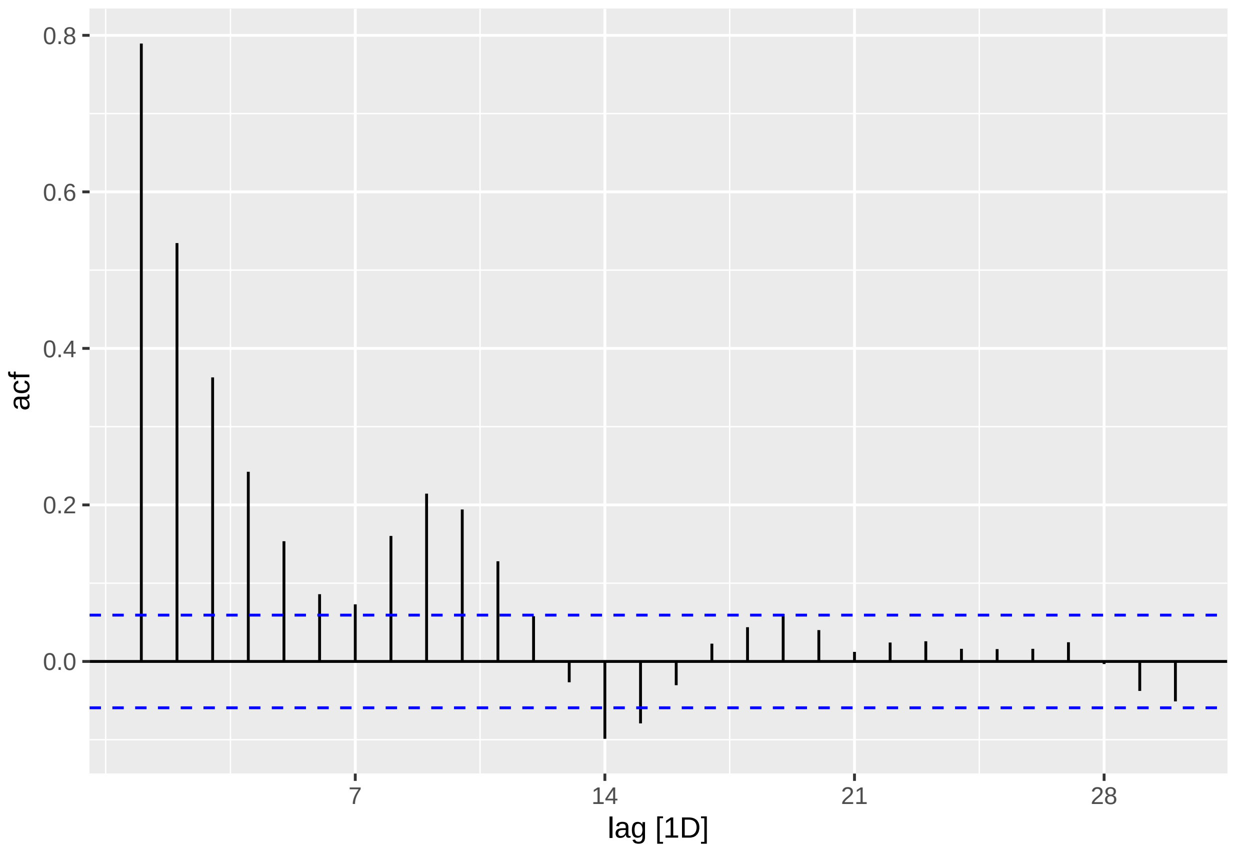


```
ggplot(energy_decomp) + geom_histogram(aes(x=remainder)) # Remainder distribution looks Gaussian
```

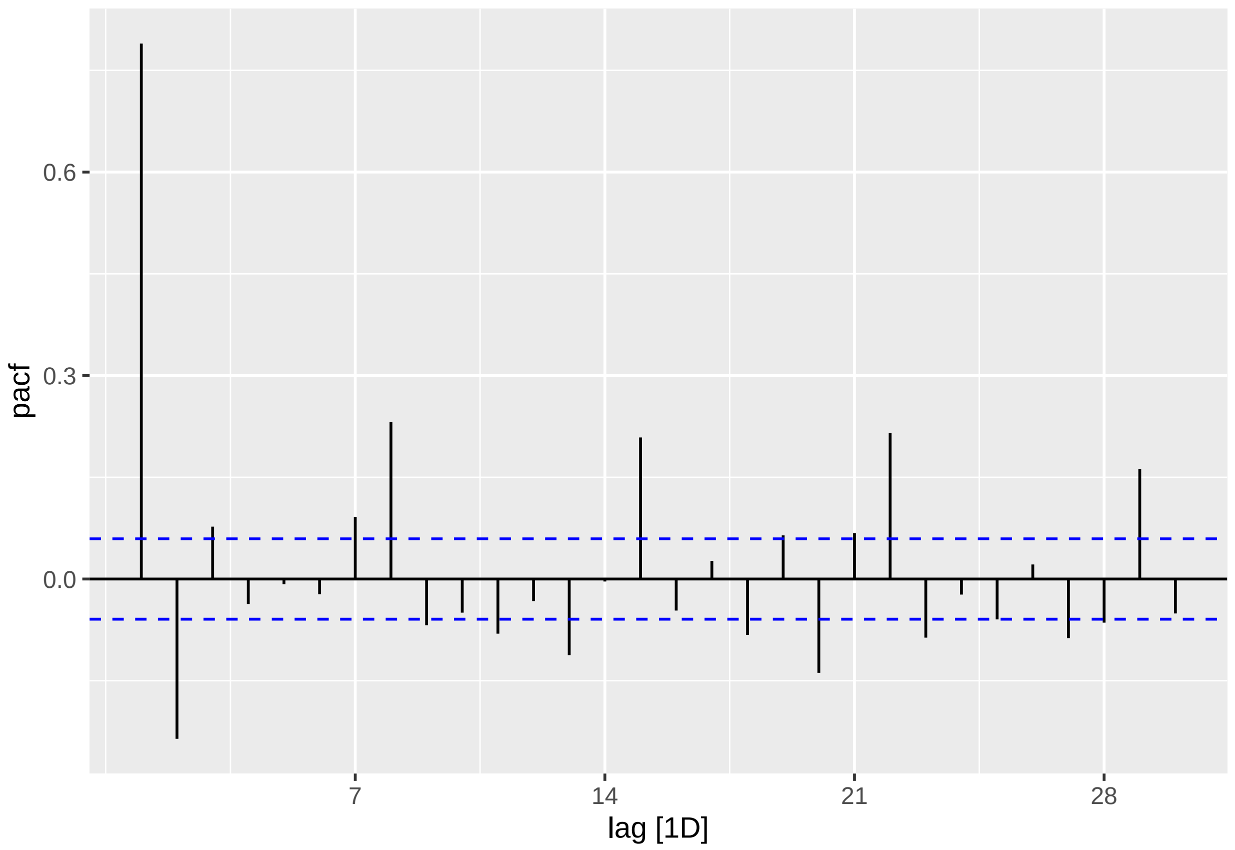
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
energy_decomp |> ACF(remainder) |> autoplot()
```



```
energy_decomp |> PACF(remainder) |> autoplot()
```


pacf
0.6
0.3
0.0
7
14
21
28
lag [1D]