```r
# Description -----------------------------------------------------

# Machine learning is based on repeating small "learning steps"
# over and over again. We will need a programming structure that
# allows us to do this in an efficient way.

# You have already seen the function sapply that does something similar.
# But for machine learning, we often want a more transparent way to do
# iterative steps. In fact we use "loops".
# Specifically, "for-loops" and "while-loops". In this script, we
# consider how they work


# Header ---------------------------------------------------------------

rm(list = ls())

library(tidyverse)


# A for-loop -------------------------------------------------------

# We want to check whether the mean of repeated simulated draws
# from a normal distribution with a given mean really converges
# to that mean. (In other words, we check the law of large numbers.)


# Here is how you can simulate data
(x = rnorm(1, mean = 10, sd = 1))
# Google to learn what this does exactly!


# Some parameters that we need
theoryMean = 10
iterRange = 1:100


# We need an object that collects the single random draws.

# An empty vector that we are going to fill up
# with randomly drawn numbers (in fact, it's not
# empty but contains zeros).
sampleList = vector("double", length(iterRange))

# And another vector that we are going to fill up
# with the resulting means:
sampleMeanList = vector("double", length(iterRange))
# In the jargon, this means preallocating an output object.

# Creating some empty (zero) "containers" that are going to
# be filled up in a loop is called "pre-allocation".
# Make sure to always use this when performing heavy
# calculations. It is much more efficient than
# making an existing vector longer by one element
# at a time!

# So here is how we may perform the loop
for (i in iterRange){
  sampleList[i] = rnorm(1, mean = theoryMean, sd = 10)
  sampleMeanList[i] = mean(sampleList[1:i])
}

X = tibble(iterRange, sampleList, sampleMeanList)
X


# Let's do a quick-and-dirty plot

ggplot(X) +
  geom_point(aes(iterRange, sampleList), color = "blue") +
  geom_line(aes(iterRange, sampleList), color = "blue", size = 0.5) +
  geom_line(aes(iterRange, sampleMeanList), color = "red3", size = 1.2)


# As you see, plotting helps a lot for getting "a picture" what
# you are actually doing...



# A highly inefficient for-loop -------------------------------------------

# Never do this:

sampleList = 0; sampleMeanList = 0

for (i in iterRange){
  x = rnorm(1, mean = 10, sd = 10)
  sampleList = c(sampleList,x)
  sampleMeanList = c(sampleMeanList, mean(sampleList))
}

sampleList; sampleMeanList

# This works fine as long as you run easy stuff. But
# it is extremely inefficient in terms of the inner
# "physical logistics" of the computing process!






# A while loop ------------------------------------------------------------

# Now suppose that we want something different.
# Not a fixed iterRange, but repeat until
# a certain condition is met, say the red curve from before
# (the sample mean) does not change any more. In mathematical
# terms, we put a "convergence condition".
# This needs an additional step. We need an
# initial situation and check whether the condition
# already holds initially.

# Let's set a maximum number of "iterations", to prevent
# our program for running forever if
iMax = 100000

# We do not know how many iterations we will actually run,
# so we set our empty containers to a length of iMax
sampleList = vector("double", iMax)
sampleMeanList = vector("double", iMax)
critList = vector("double", iMax)
theoryMeanList = rep(theoryMean, iMax)


i = 1
crit = 1000  # critical value for passing test whether
             # more iterations should be run.
             # Initialize crit at a high number that
             # certainly does not meet the condition

tolX = 1e-6  # if crit meets this value, the loop stops



while (crit > tolX & i<=iMax){
  sampleList[i] = rnorm(1, mean = theoryMean, sd = 10)
  sampleMeanList[i] = mean(sampleList[1:i])

  # For the first draw (i=1), there is no "previous" draw.
  # So keep crit at the initial value
  if (i>1) {
    crit = abs(sampleMeanList[i] - sampleMeanList[i-1])/
      abs(sampleMeanList[i-1])
  }

  # What does crit actually measure?
  # Does this remind you of some mathematics?

  critList[i] = crit

  i = i+ 1
}

# In mathemaical terms, the idea is (somewhat loosely speaking):
# we hope to have a Cauchy sequence, which implies that it converges
# Ask google if you are interested :-)

# Collect the entire output of interest in a dataframe.
# Note that name that is provided to the first column.

Y = tibble(id = 1:iMax, theoryMeanList, sampleList, sampleMeanList, critList)
```

```r
# Y still contains many zeros. In fact, from the row
# corresponding to the current
# value of i on, nothing has been filled.
# Let's cut Y at that row
Y = Y[1:(i-1), ]

# You could have used filter() here. How?

# inspect the last few rows of Y.
# What can you infer?
tail(Y)





# Plotting the results (template for basic machine learning applications) --------------------


xlab = "Draw(s)"; ylab = "Value"

ggplot(Y) +
  geom_point(aes(id, sampleList), color = "blue", alpha = 0.3) +
  geom_line(aes(id, sampleList), color = "blue", size = 0.5, alpha = 0.3) +
  geom_line(aes(id, sampleMeanList), color = "red3", size = 1.2) +
  labs(x = xlab, y = ylab)

# What does this graph show?
# What does the alpha do?



ggplot(Y) +
  geom_line(aes(id, sampleMeanList), color = "red3", size = 1.2) +
  geom_line(aes(id, theoryMeanList), color = "black", linetype = 2) +
  labs(x = xlab, y = ylab, title = "The mean approaches its theoretical value")





# Only the last N observations to study convergence in detail
N = 100

ggplot(tail(Y, N)) +
  geom_line(aes(id, sampleMeanList), color = "red3", size = 1.2) +
  geom_line(aes(id, theoryMeanList), color = "black", linetype = 2)


# The types of graphs we have here are useful for machine learning.
# There is one catch. So far, it is not easy to create nice legends
# that tell you which line represents what. In the next section
# we look at an elegant way to produce high-quality graphs.
# Sophisticated graphs with informative legends -------------------------------


# Often, in ggplot, the best way to proceed is to bring the data into
# the long format (yes, in the untidy long format...)

# This allows for creating labels for legends etc. You can achieve almost
# everything in this way.
# Below, we just look at a few examples.

# To get from wide to long, you need the opposite of spread(),
# which is gather(). Since everything is a bit complex,
# we first use toy data to work with.


toy = Y[1:5, 1:4]
# What did we do here?
# You can read about subsetting in the book on pp. 300-302.


names(toy)

# The wide-to-long process is somewhat tricky.
# The first argument specifies the columns that
# are going to be stacked on top of each other.
# The respective column names go into the new "key" column.
# (You notice that it is really the reverse of spread()!)
# We call the key column "series". The values in the new long format
# will be in a column with name "Value".


toPlot <- toy %>%
  gather(names(toy)[2:4], key = "series", value = "value")


# The simplest thing we can plot is this:
ggplot(toPlot) +  geom_line(aes(id, value, color = series))

# This looks very ugly, but there is an informative legend,
# something we did not have before!

# Of course, we want the legend to show telling names.
# As we have done before, we create nice telling labels with sapply:

# Inspect the values in the series column...
unique(toPlot$series)


# By now, you know how this works!
toPlot <- toPlot %>%
  mutate(Legend =
           sapply(series, switch,
                  sampleList = "Single draw",
                  sampleMeanList = "Sample mean",
                  theoryMeanList = "Theor. mean"))


# Now, we can use the color asthetic for creating an appropriate legend
ggplot(toPlot) +  geom_line(aes(id, value, color = Legend))


# Or without legend title...
ggplot(toPlot) +  geom_line(aes(id, value, color = Legend)) +
  theme(legend.title=element_blank())


# We still want to make this look nicer, by pickig colors manually
# and selecting specific linetypes.
# Let's look at this with only two series

unique(toPlot$series)

# Select only the columns that we are interested in
toPlot2 <- toPlot %>%
  filter(series %in% c("sampleMeanList", "theoryMeanList"))

ggplot(toPlot2) +  geom_line(aes(id, value, color = Legend))

# Line type added
ggplot(toPlot2) +  geom_line(aes(id, value, color = Legend,
                                 linetype = Legend))


# More customized; note that we put the aes inside ggplot().
# This saves us to repeat them as arguments in geom_line().
ggplot(toPlot2, aes(id, value)) +
  geom_line(aes(color = Legend))+

  # Add a manually chosen color scheme
  scale_colour_manual(values = c("red", "black"))


# Manual color and linetype scheme
ggplot(toPlot2, aes(id, value)) +
  geom_line(aes(color = Legend, linetype = Legend))+
  scale_colour_manual(values = c("black", "red")) +

  # Here goes the linetype scheme
  scale_linetype_manual(values=c("dotted", "solid"))


# And with a "size" scheme...
ggplot(toPlot2, aes(id, value)) +
  geom_line(aes(color = Legend, linetype = Legend, size = Legend))+
  scale_colour_manual(values = c("red", "black")) +
  scale_linetype_manual(values=c("solid", "dotted")) +

  # Defines thickness of the lines
  scale_size_manual(values=c(1, 0.5))


# Useful links:
# http://www.sthda.com/english/wiki/ggplot2-line-types-how-to-change-line-types-of-a-graph-in-r-softwar
# http://www.sthda.com/english/wiki/ggplot2-themes-and-background-colors-the-3-elements


# Now that we have played around with the toy data of just the first five
# rows of Y, we can proceed to plot the entire sample.


# A fully-fledge graph ----------------------------------------------------


# In complete analogy to the toy version above

toPlot <- Y %>%
  gather(names(Y)[2:4], key = "series", value = "value")


toPlot <- toPlot %>%
  mutate(Legend =
           sapply(series, switch,
                  sampleList = "Single draw",
                  sampleMeanList = "Sample mean",
                  theoryMeanList = "Theor. mean"))


toPlot2 <- toPlot %>%
  filter(series %in% c("sampleMeanList", "theoryMeanList"))




# Sample mean and theoretical mean

ggplot(toPlot2, aes(id, value)) +
  geom_line(aes(color = Legend, linetype = Legend, size = Legend))+
  scale_colour_manual(values = c("red", "black")) +
  scale_linetype_manual(values=c("solid", "dotted")) +
  scale_size_manual(values=c(1, 0.5)) +
  labs(x = "Draw(s)", y = "Value",
       title = "The mean of a sample approaches its theoretical value") +
  theme_bw() +
  theme(legend.title=element_blank())



# With only a limited range in the x and y direction
# (for closer inspection)
```

```r
ggplot(toPlot2, aes(id, value)) +
  geom_line(aes(color = Legend, linetype = Legend, size = Legend))+
  scale_colour_manual(values = c("red", "black")) +
  scale_linetype_manual(values=c("solid", "dotted")) +
  scale_size_manual(values=c(1, 0.5)) +

  # Here goes the x range
  xlim(500, max(toPlot$id)) +
  ylim(9,11) +
  labs(x = "Draw(s)", y = "Value",
       title = "The mean of a sample approaches its theoretical value") +
  theme_bw() +
  theme(legend.title=element_blank())




# Including the sample draws...


# You don't see much here...
ggplot(toPlot, aes(id, value)) +
  geom_line(aes(color = Legend, linetype = Legend, size = Legend))+
  scale_colour_manual(values = c("red", "gray", "black")) +
  scale_linetype_manual(values=c("solid", "solid", "dotted")) +
  scale_size_manual(values=c(1, 0.1, 0.1)) +
  labs(x = "Draw(s)", y = "Value",
       title = "The mean of a sample approaches its theoretical value") +
  theme_bw() +
  theme(legend.title=element_blank())


# So include all the alpha stuff
ggplot(toPlot, aes(id, value)) +
  geom_line(aes(color = Legend, linetype = Legend, size = Legend, alpha = Legend))+
  scale_colour_manual(values = c("red", "gray", "black")) +
  scale_linetype_manual(values=c("solid", "solid", "dotted")) +
  scale_size_manual(values=c(1, 0.5, 1)) +

  # Note the guide = F argument, without it, the legend looks weird
  scale_alpha_manual(values = c(1, 0.5, 1), guide = F) +
  labs(x = "Draw(s)", y = "Value",
       title = "The mean of a sample approaches its theoretical value") +
  theme_bw() +
  theme(legend.title=element_blank())
```