Assignment 6

Exercise 1

First we need to write down the Metropolis-Hasting Algorithm:

```
# Parameters:
# func : a function whose first argument is a real vector of parameters
# func returns a log10 of the likelihood function
# theta.init : the initial value of the Markov Chain (and of func)
# n.sample: number of required samples
# sigma : standar deviation of the gaussian MCMC sampling pdf

metropolis.Hasting <- function(func , theta.init , n.sample) {
theta.cur <- theta.init
func.Cur <- func(theta.cur)
func.Samp <- matrix(data=NA, nrow=n.sample , ncol=2+1)
n.accept <- 0
rate.accept <- 0.0

for (n in 1:n.sample) {
theta.prop <- rnorm(n=1, mean = 0, 1)
func.Prop <- func(theta.prop)
logMR <- func.Prop - func.Cur # Log10 of the Metropolis ratio
if ( logMR >=0 || logMR >log10(runif(1)) ) {
theta.cur <- theta.prop
func.Cur <- func.Prop
n.accept <- n.accept + 1
}
func.Samp[n, 1] <- func.Cur
func.Samp[n, 2] <- theta.cur
func.Samp[n, 3] <- n
}
return(func.Samp)
}
```

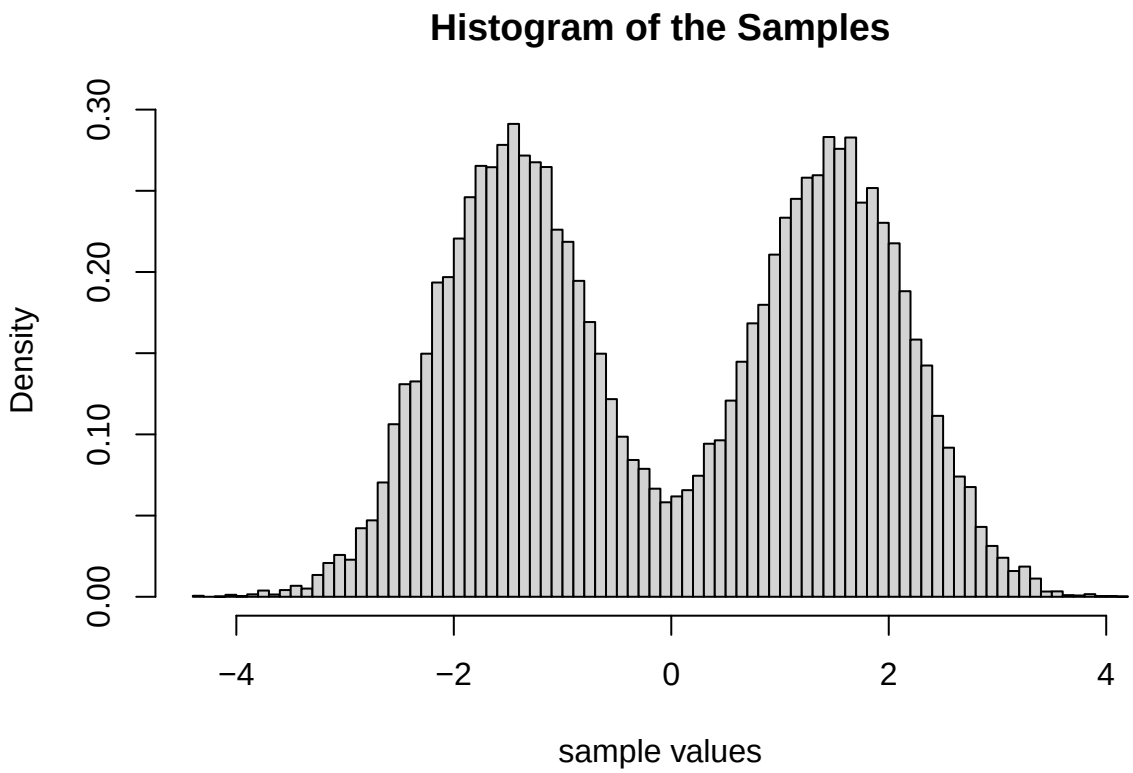
Now we need to define the posterior function we are going to sample from:

```
testfunc <- function(theta) {
return(0.5 * exp(-(theta + 3)^2/2) + 0.5 * exp(-(theta - 3)^2/2) )}

testfunc.metropolis <- function(lambda) {
return(log10(testfunc(lambda)))}
```

Now we are ready to sample from the desired posterior:

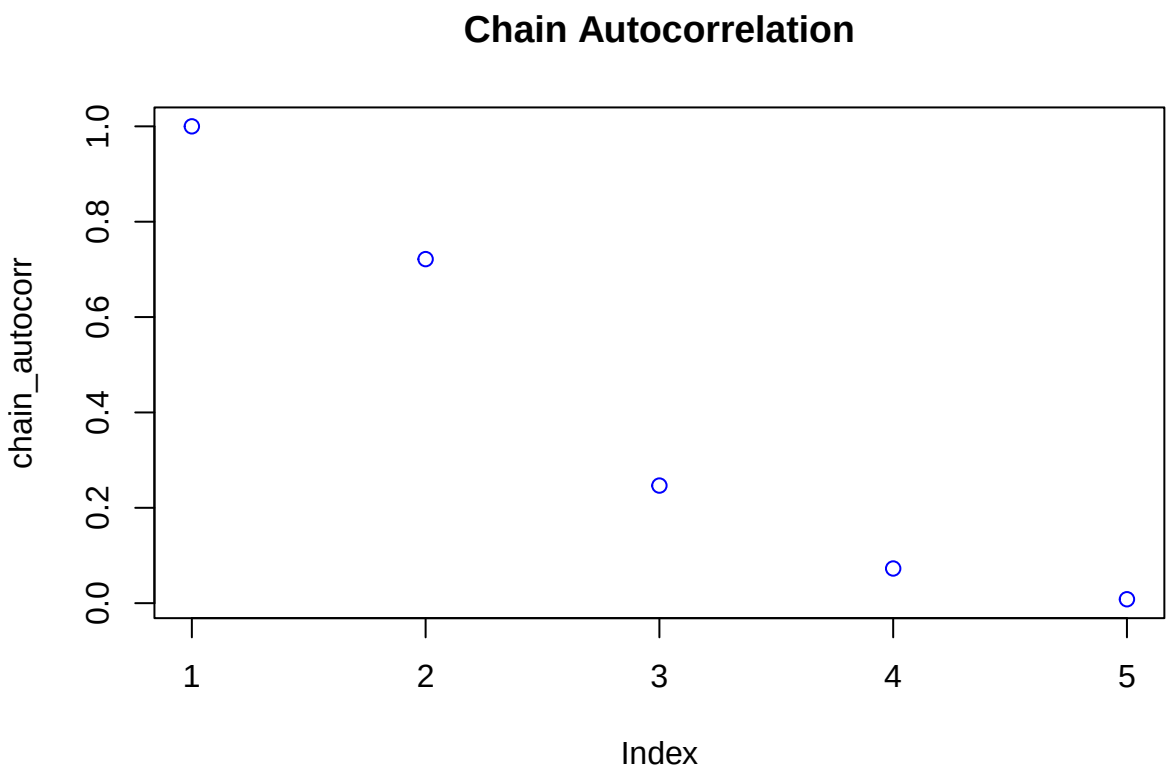
```
theta.init <- 0.1
n.sample <- 10^5

set.seed(20190513)
chain <- metropolis.Hasting(func=testfunc.metropolis ,
theta.init = theta.init ,
n.sample = n.sample )
```

```r
mcmc.data = chain[1000:nrow(chain) , 2]

hist(mcmc.data , breaks = 100 , freq = F , main = 'Histogram of the Samples'  , xlab = 'sample values')
```



Above I have used a 1000 burn-in interval.

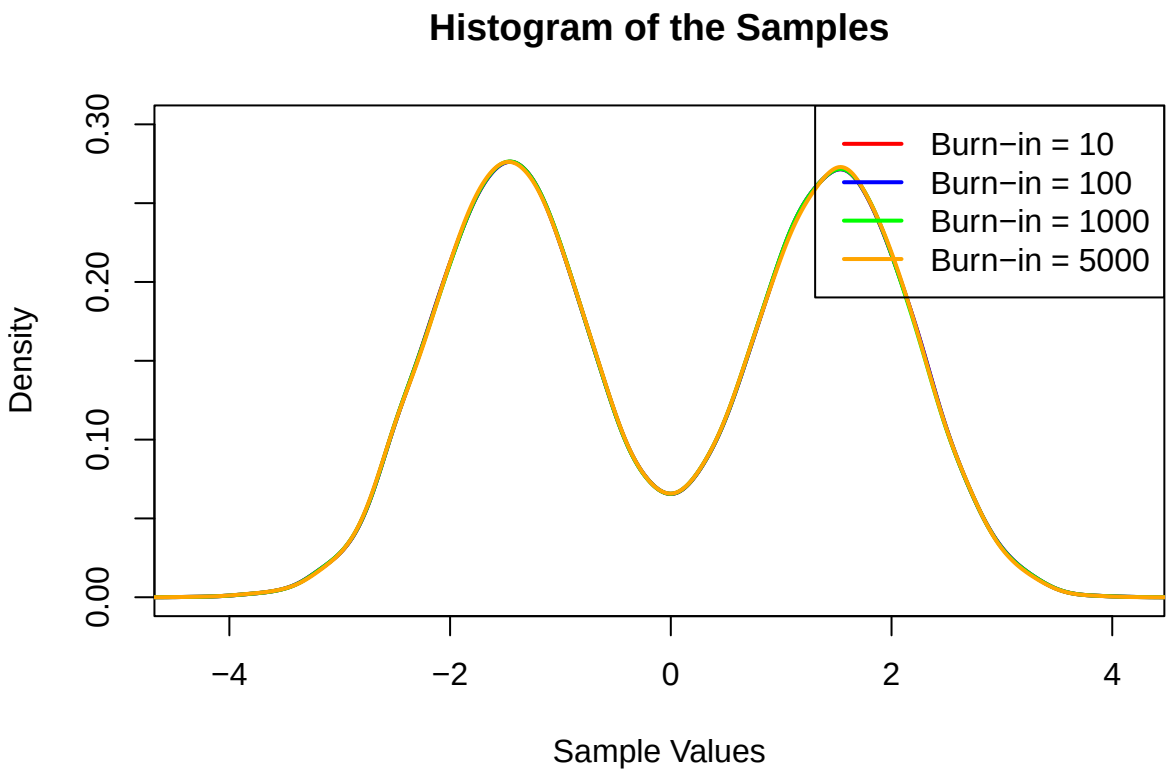
```r
library(coda)

# Convert your chain data to a CODA object
mcmc.data <- as.mcmc(mcmc.data)

# Compute the autocorrelation
chain_autocorr <- autocorr(mcmc.data)

# Plot the chain autocorrelation
plot(chain_autocorr, main = "Chain Autocorrelation" , col = 'blue')
```



We see that the auto-correlation drops quickly thanks to the Metropolis-Hasting algorithm.

Now lets try different burn-ins and plot the results together:

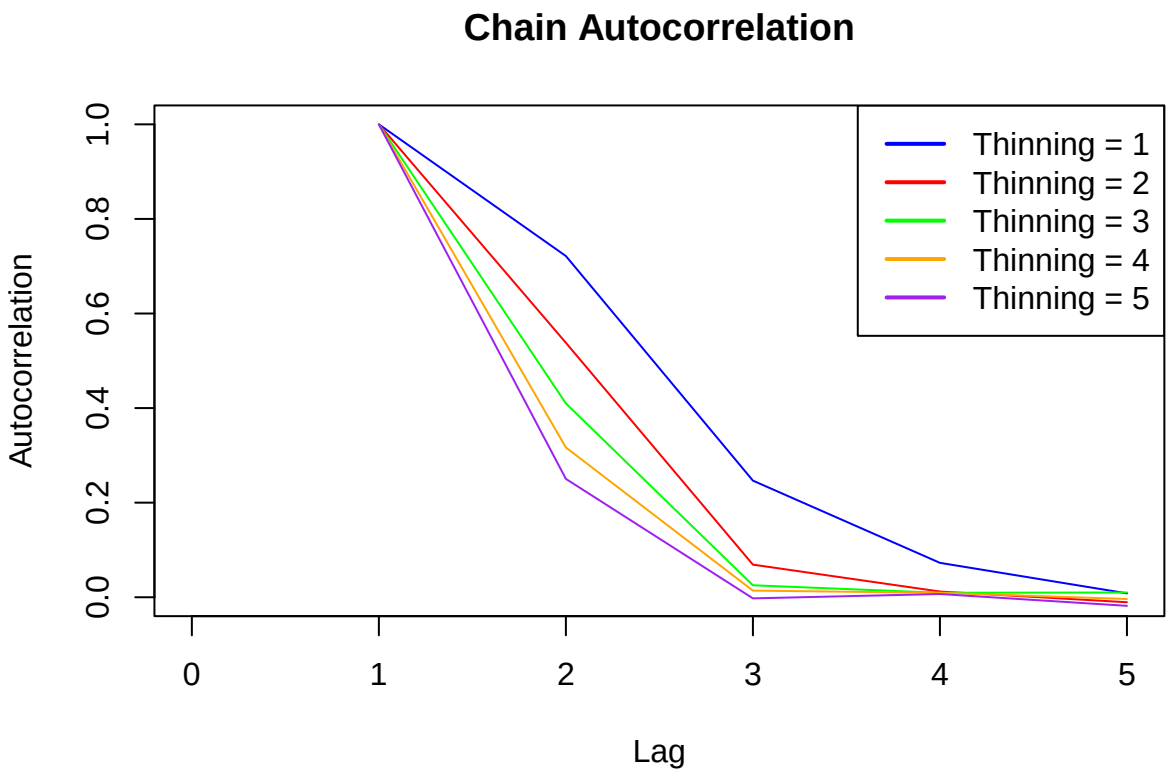
```
burn_ins <- c(10, 100, 1000, 5000)
legend_labels <- paste("Burn-in =", burn_ins)
colors <- c("red", "blue", "green", "orange")

# Create an empty plot
plot(NULL, xlim = range(chain[, 2]), ylim = c(0, 0.3), type = "n", main = 'Histogram of the Samples', x

# Superimpose histograms for different burn-ins
for (i in 1:length(burn_ins)){
  bi <- burn_ins[i]
  mcmc.data <- chain[(bi+1):nrow(chain), 2]  # Exclude burn-in samples

  # Compute density estimate
  dens <- density(mcmc.data)

  # Plot density as a line
  lines(dens$x, dens$y, col = colors[i], lwd = 2)
}

# Add legend
legend("topright", legend = legend_labels, col = colors, lwd = 2)
```



We see that changing the burn-in values have a negligible effect on the posterior.

To see how the auto-correlation function changes with different thinning intervals:

```
library(coda)  # Load the CODA package if not already loaded

thinnings <- c(1, 2, 3, 4, 5)
colors <- c("blue", "red", "green", "orange", "purple")
legend_labels <- paste("Thinning =", thinnings)

# Create an empty plot
plot(NULL, xlim = c(0, length(chain_autocorr)), ylim = c(0, 1), type = "n", main = "Chain Autocorrelati

# Superimpose autocorrelation plots for different thinning intervals
for (i in 1:length(thinnings)){
  thin <- thinnings[i]
  mcmc.data <- chain[seq(from = 1000, to = nrow(chain), by = thin), 2]
  mcmc.data <- as.mcmc(mcmc.data)
  chain_autocorr <- autocorr(mcmc.data)

  # Plot autocorrelation with different colors
  lines(chain_autocorr, col = colors[i])
}

# Add legend
legend("topright", legend = legend_labels, col = colors, lwd = 2)
```



We see that by increasing the thinning interval, the auto-correlation function drops faster.

Exercise 2

We start by defining a model and sampling p, the probability of a member of a group to be diagnosed with covid, from its derived posterior. Then the efficacy can be calculated accordingly:

$$\text{efficacy} = \frac{p_{\text{placebo}} - p_{\text{vaccine}}}{p_{\text{placebo}}}$$

Let's begin with Jcovden:

```
#Jcovden

library(rjags)
```

```
## Linked to JAGS 4.3.1
```

```
## Loaded modules: basemod,bugs
```

```
# Prepare the data
vaccine_cases <- 116
vaccine_population <- 19630
placebo_cases <- 348
```

```r
placebo_population <- 19691

data <- list(vaccine_cases = vaccine_cases,
             vaccine_population = vaccine_population,
             placebo_cases = placebo_cases,
             placebo_population = placebo_population)


# Compile the model
model <- jags.model("model.txt", data = data)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2
##    Unobserved stochastic nodes: 2
##    Total graph size: 11
##
## Initializing model
```

```r
# Define parameters to monitor
parameters <- c("p_vaccine", "p_placebo", "efficacy")

# Run the MCMC
iterations <- 10000
burn_in <- 5000
thin <- 1
chains <- 2

samples <- coda.samples(model, variable.names = parameters,
                        n.iter = iterations, thin = thin, n.chains = chains)

# Summarize the results
summary(samples)
```

```
##
## Iterations = 1001:11000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                Mean        SD  Naive SE Time-series SE
## efficacy  66.353372 3.5754999 3.575e-02      4.510e-02
## p_placebo  0.017719 0.0009433 9.433e-06      1.192e-05
## p_vaccine  0.005945 0.0005455 5.455e-06      6.831e-06
##
## 2. Quantiles for each variable:
##
##                2.5%       25%       50%       75%     97.5%
```

```
## efficacy  59.005672 64.032103 66.506296 68.828650 72.881883
## p_placebo  0.015923  0.017076  0.017702  0.018348  0.019617
## p_vaccine  0.004928  0.005566  0.005931  0.006302  0.007038
```

As we see the 95% credibility interval for Jcovden is 59.01 to 72.96

```
#Moderna

library(rjags)

# Prepare the data
vaccine_cases <- 11
vaccine_population <- 14134
placebo_cases <- 185
placebo_population <- 14073

data <- list(vaccine_cases = vaccine_cases,
             vaccine_population = vaccine_population,
             placebo_cases = placebo_cases,
             placebo_population = placebo_population)


# Compile the model
model <- jags.model("model.txt", data = data)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2
##    Unobserved stochastic nodes: 2
##    Total graph size: 11
##
## Initializing model
```

```
# Define parameters to monitor
parameters <- c("p_vaccine", "p_placebo", "efficacy")

# Run the MCMC
iterations <- 10000
burn_in <- 5000
thin <- 1
chains <- 2

samples <- coda.samples(model, variable.names = parameters,
                        n.iter = iterations, thin = thin, n.chains = chains)

# Summarize the results
summary(samples)
```

```
##
## Iterations = 1001:11000
## Thinning interval = 1
```

```
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                Mean        SD  Naive SE Time-series SE
## efficacy  9.354e+01 1.9185897 1.919e-02      2.673e-02
## p_placebo 1.323e-02 0.0009707 9.707e-06      1.248e-05
## p_vaccine 8.495e-04 0.0002436 2.436e-06      3.439e-06
##
## 2. Quantiles for each variable:
##
##                2.5%       25%       50%       75%     97.5%
## efficacy  8.919e+01 92.422713 9.372e+01 94.941039 96.717037
## p_placebo 1.139e-02  0.012560 1.320e-02  0.013870  0.015188
## p_vaccine 4.414e-04  0.000675 8.274e-04  0.001001  0.001381
```

We see that for Moderna, the credibility interval is from 89.38 to 96.62.

(The link for Astrazenca was expired)

Exercise 3

We start by loading the data:

```
# Load the data from the URL into a dataframe
url <- "https://covid.ourworldindata.org/data/owid-covid-data.csv"
data <- read.csv(url)
```

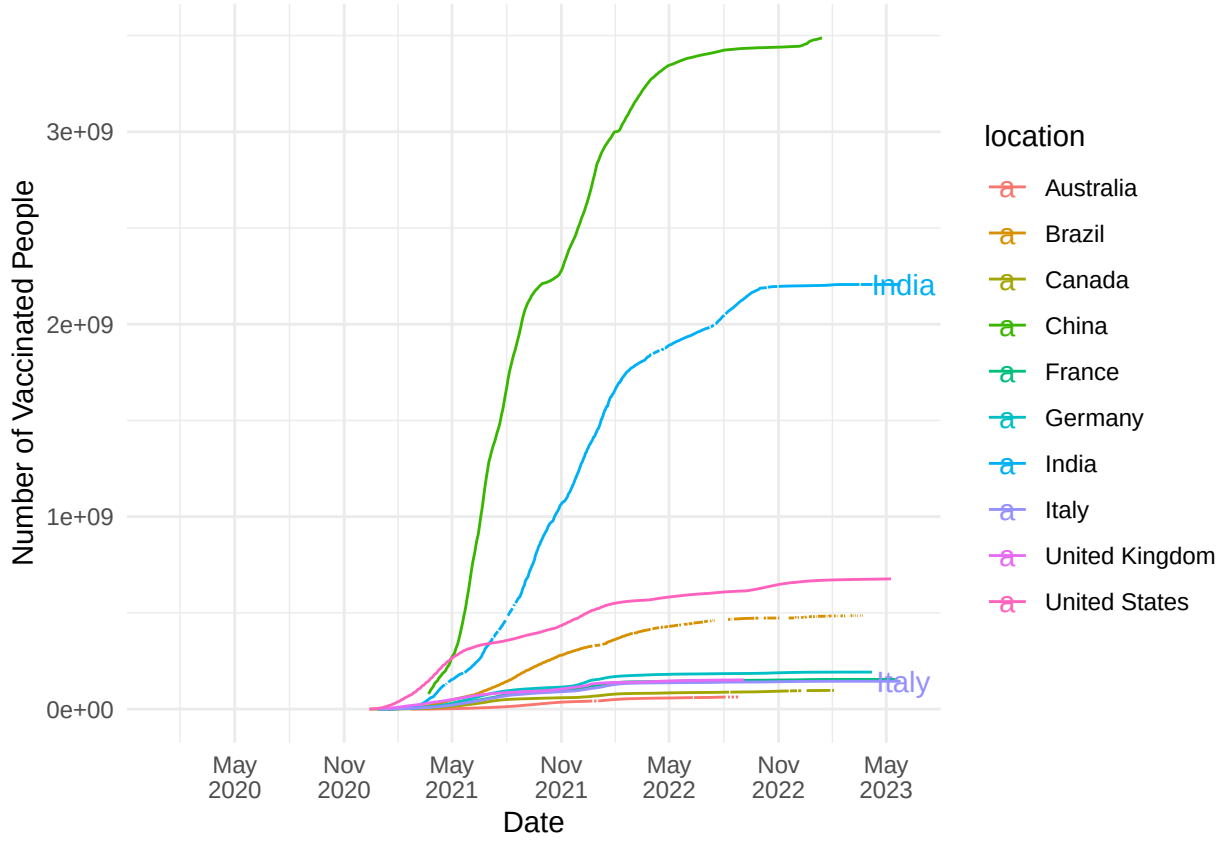
Since the data is quite large, we just do the plots for 10 specific countries:

```
library(ggplot2)
# Select 10 countries for plotting
selected_countries <- c("United States", "United Kingdom", "Germany", "France", "Italy",
                        "Canada", "Brazil", "India", "China", "Australia")

# Filter the data to include only the selected countries
filtered_data <- subset(data, location %in% selected_countries,
                        select = c("location", "date", "total_vaccinations"))

# Convert the date column to proper date format
filtered_data$date <- as.Date(filtered_data$date)

# Plot the number of vaccinated people over time for the selected countries
plot <- ggplot(filtered_data, aes(x = date, y = total_vaccinations, color = location)) +
  geom_line() +
  geom_text(data = subset(filtered_data, date == max(date)), aes(label = location),
            nudge_x = 5, nudge_y = 10000, check_overlap = TRUE) +
  labs(x = "Date", y = "Number of Vaccinated People") +
  scale_x_date(date_labels = "%b\n%Y", date_breaks = "6 month") +
  theme_minimal()
```

```
# Display the plot
print(plot)
```

```
## Warning: Removed 4334 rows containing missing values (‘geom_line()‘).
```

```
## Warning: Removed 2 rows containing missing values (‘geom_text()‘).
```



For the world:

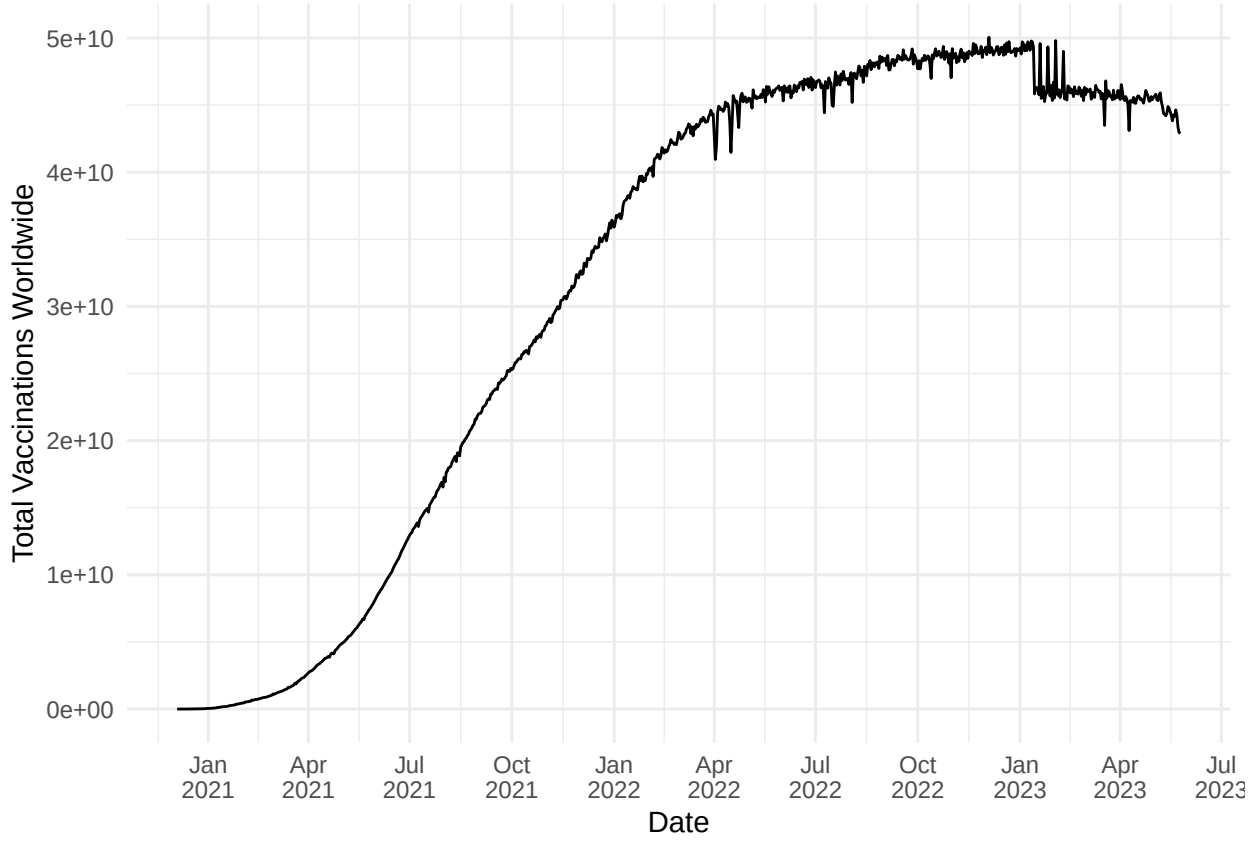
```
library(dplyr)
```

```
##
## 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
```

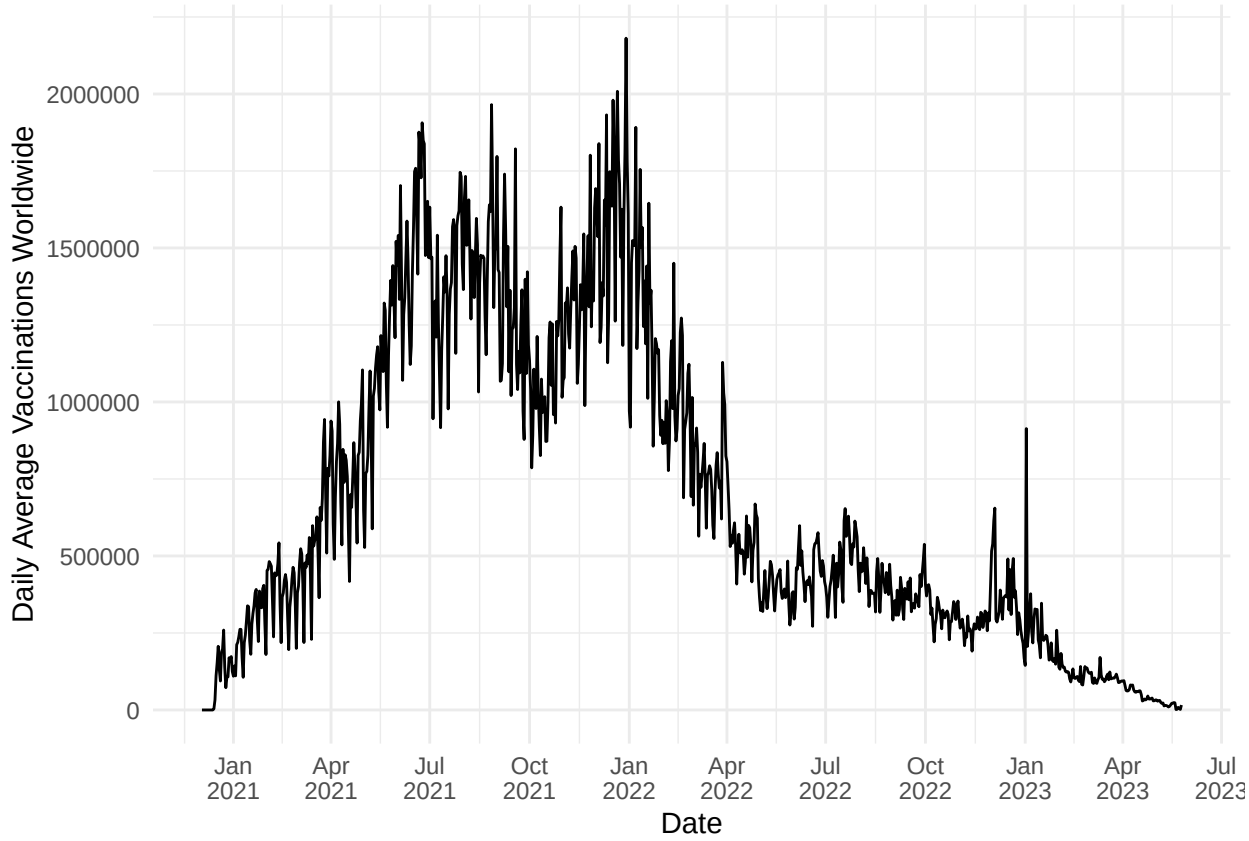
```r
# Filter the data to include only the relevant columns
filtered_data <- subset(data, select = c("date", "total_vaccinations"))



# Filter out rows with missing or zero total_vaccinations
filtered_data <- filtered_data[!is.na(filtered_data$total_vaccinations) & filtered_data$total_vaccinati

# Calculate the cumulative vaccinations worldwide
worldwide_data <- filtered_data %>%
  group_by(date) %>%
  summarise(total_vaccinations_worldwide = sum(total_vaccinations))

worldwide_data$date <- as.Date(worldwide_data$date)

# Plot the total number of vaccinated people worldwide over time
plot <- ggplot(worldwide_data, aes(x = date, y = total_vaccinations_worldwide)) +
  geom_line() +
  labs(x = "Date", y = "Total Vaccinations Worldwide") +
  scale_x_date(date_labels = "%b\n%Y", date_breaks = "3 month") +
  theme_minimal()

# Display the plot
print(plot)
```



We see that for some reason the cumulative number of people vaccinated worldwide has decreased in 2023,

which shows there is some error in the report. A possible explanation is that the data for some countries is missing for this period and therefore while aggregating the data, the number has dropped.

```
# Filter the data to include only the relevant columns
filtered_data <- subset(data, select = c("date", "new_vaccinations"))

# Convert the date column to Date class format
filtered_data$date <- as.Date(filtered_data$date, format = "%Y-%m-%d")

# Filter out rows with missing or zero new_vaccinations
filtered_data <- filtered_data[!is.na(filtered_data$new_vaccinations) & filtered_data$new_vaccinations

# Calculate the daily average vaccinations worldwide
worldwide_data <- filtered_data %>%
  group_by(date) %>%
  summarise(average_vaccinations_worldwide = mean(new_vaccinations))

# Convert the date column to Date format
worldwide_data$date <- as.Date(worldwide_data$date)

# Plot the daily average vaccinations worldwide over time
plot <- ggplot(worldwide_data, aes(x = date, y = average_vaccinations_worldwide)) +
  geom_line() +
  labs(x = "Date", y = "Daily Average Vaccinations Worldwide") +
  scale_x_date(date_labels = "%b\n%Y", date_breaks = "3 month") +
  theme_minimal()

# Display the plot
print(plot)
```




```
# Filter the data to include only the relevant columns
filtered_data <- subset(data, select = c("date", "new_vaccinations"))

# Convert the date column to Date class format
filtered_data$date <- as.Date(filtered_data$date, format = "%Y-%m-%d")

# Filter out rows with missing or zero new_vaccinations
filtered_data <- filtered_data[!is.na(filtered_data$new_vaccinations) & filtered_data$new_vaccinations

# Extract the year and week from the date
filtered_data$year <- lubridate::year(filtered_data$date)
filtered_data$week <- lubridate::week(filtered_data$date)

# Calculate the weekly average vaccinations worldwide
worldwide_data <- filtered_data %>%
  group_by(year, week) %>%
  summarise(average_vaccinations_worldwide = mean(new_vaccinations))
```

```
## `summarise()` has grouped output by 'year'. You can override using the
## `.groups` argument.
```

```
# Create a new date column representing the start of each week
worldwide_data$date <- as.Date(paste(worldwide_data$year, worldwide_data$week, "1", sep = "-"), format =
```

```
## Warning in strptime(x, format, tz = "GMT"): (0-based) yday 369 in year 2020 is
```

```
## invalid
```

```
## Warning in strptime(x, format, tz = "GMT"): (0-based) yday 367 in year 2021 is
## invalid
```

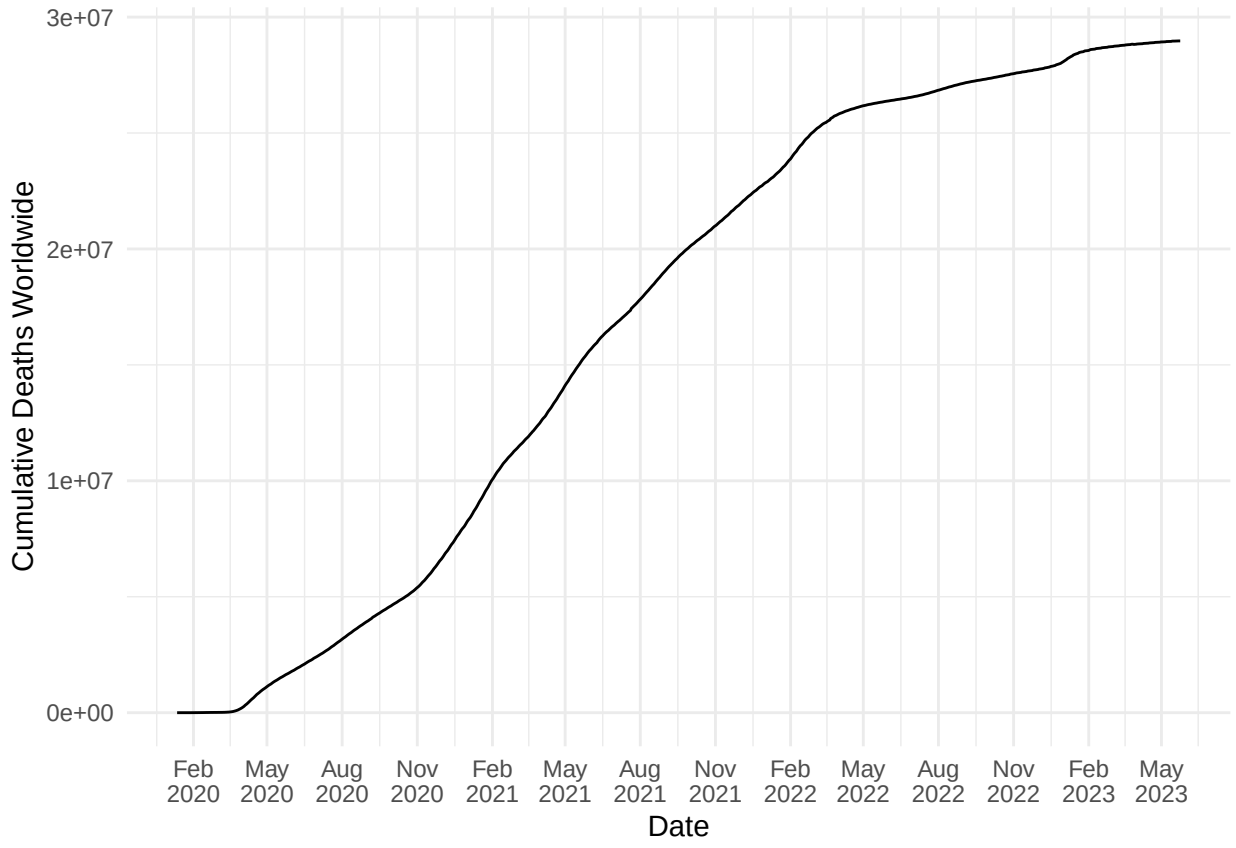
```
## Warning in strptime(x, format, tz = "GMT"): (0-based) yday 366 in year 2022 is
## invalid
```

```
# Plot the weekly average vaccinations worldwide over time
plot <- ggplot(worldwide_data, aes(x = date, y = average_vaccinations_worldwide)) +
  geom_line() +
  labs(x = "Date", y = "Weekly Average Vaccinations Worldwide") +
  scale_x_date(date_labels = "%b\n%Y", date_breaks = "3 months") +
  theme_minimal()

# Display the plot
print(plot)
```

```
## Warning: Removed 3 rows containing missing values (`geom_line()`).
```


```
# Filter the data to include only the relevant columns
filtered_data <- subset(data, select = c("date", "total_deaths"))

# Convert the date column to Date class format
```

```r
filtered_data$date <- as.Date(filtered_data$date, format = "%Y-%m-%d")

# Filter out rows with missing or zero total_deaths
filtered_data <- filtered_data[!is.na(filtered_data$total_deaths) & filtered_data$total_deaths != 0, ]

# Calculate the cumulative number of deaths worldwide
worldwide_data <- filtered_data %>%
  group_by(date) %>%
  summarise(cumulative_deaths_worldwide = sum(total_deaths))

# Plot the cumulative number of deaths worldwide over time
plot <- ggplot(worldwide_data, aes(x = date, y = cumulative_deaths_worldwide)) +
  geom_line() +
  labs(x = "Date", y = "Cumulative Deaths Worldwide") +
  scale_x_date(date_labels = "%b\n%Y", date_breaks = "3 months") +
  theme_minimal()

# Display the plot
print(plot)
```




```r
# Filter the data to include only the relevant columns
filtered_data <- subset(data, select = c("date", "new_deaths"))

# Convert the date column to Date class format
filtered_data$date <- as.Date(filtered_data$date, format = "%Y-%m-%d")
```

```r
# Filter out rows with missing or zero new_deaths
filtered_data <- filtered_data[!is.na(filtered_data$new_deaths) & filtered_data$new_deaths != 0, ]

# Calculate the weekly average number of deaths worldwide
worldwide_data <- filtered_data %>%
  group_by(date = as.Date(lubridate::floor_date(date, "week"))) %>%
  summarise(average_deaths_worldwide = mean(new_deaths))

# Plot the weekly average number of deaths worldwide over time
plot <- ggplot(worldwide_data, aes(x = date, y = average_deaths_worldwide)) +
  geom_line() +
  labs(x = "Date", y = "Weekly Average Deaths Worldwide") +
  scale_x_date(date_labels = "%b\n%Y", date_breaks = "3 months") +
  theme_minimal()

# Display the plot
print(plot)
```