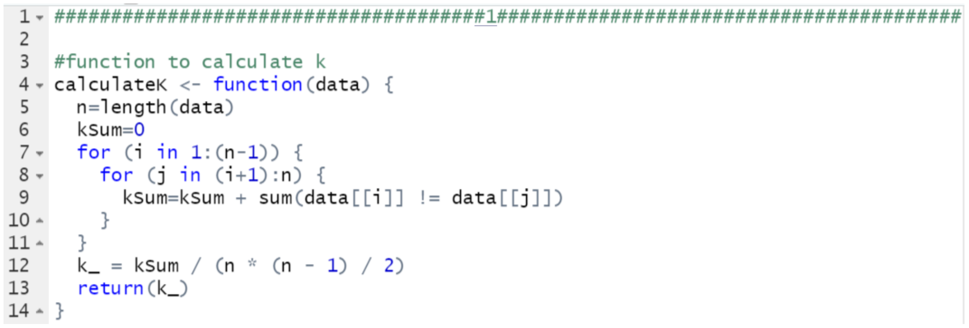
Question 1

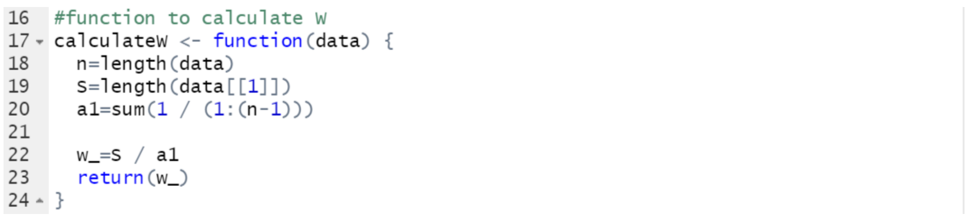
The methods we used for each function are represented below.

- k: Iterate over all pairs of sequences in the dataset and count the number of differences between them. Sum up the differences for all pairs and calculate the average.



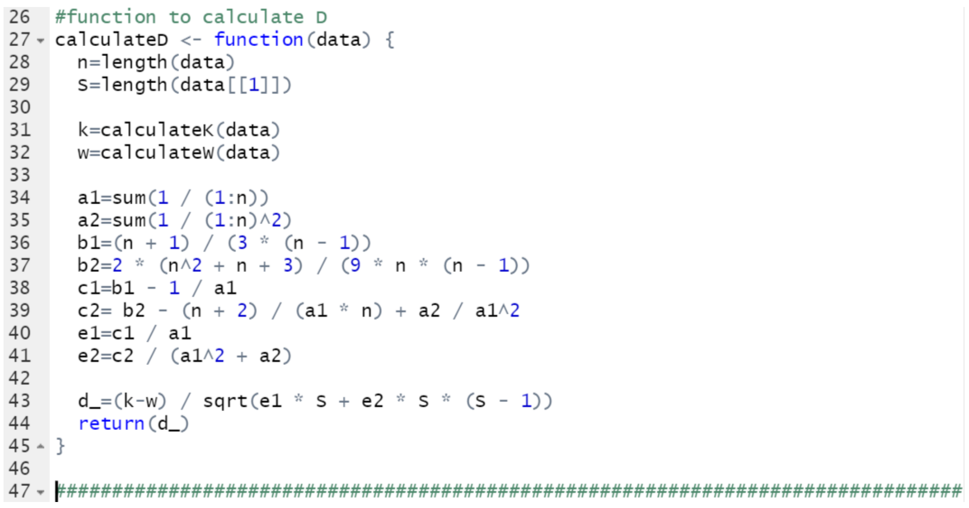
```
######################################1##########################################

#function to calculate k
calculateK <- function(data) {
  n=length(data)
  kSum=0
  for (i in 1:(n-1)) {
    for (j in (i+1):n) {
      kSum=kSum + sum(data[[i]] != data[[j]])
    }
  }
  k_ = kSum / (n * (n - 1) / 2)
  return(k_)
}
```

- w: Calculate the number of polymorphic positions (S) in the dataset. Divide S by the value of a1, where a1 is the sum of the reciprocal of integers from 1 to n-1.



```
#function to calculate W
calculateW <- function(data) {
  n=length(data)
  S=length(data[[1]])
  a1=sum(1 / (1:(n-1)))

  W_=S / a1
  return(W_)
}
```

- Tajima's D: Calculate the difference between the average number of pairwise differences (k) and the number of segregating sites (S), divided by the square root of the variance of k.



```
#function to calculate D
calculateD <- function(data) {
  n=length(data)
  S=length(data[[1]])

  k=calculateK(data)
  W=calculateW(data)

  a1=sum(1 / (1:n))
  a2=sum(1 / (1:n)^2)
  b1=(n + 1) / (3 * (n - 1))
  b2=2 * (n^2 + n + 3) / (9 * n * (n - 1))
  c1=b1 - 1 / a1
  c2= b2 - (n + 2) / (a1 * n) + a2 / a1^2
  e1=c1 / a1
  e2=c2 / (a1^2 + a2)

  d_=(k-W) / sqrt(e1 * S + e2 * S * (S - 1))
  return(d_)
}

################################################################################
```

Question 2

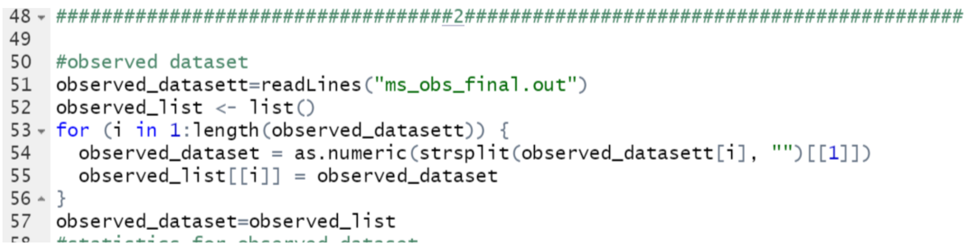
Firstly, we need to read the files. We used the following methods:

- For the observed dataset:

  Read the file line by line.

  Split each line into individual values and convert them to numeric format.

  Store the values as a list, where each element represents a genome.



```r
###################################2#########################################

#observed dataset
observed_datasett=readLines("ms_obs_final.out")
observed_list <- list()
for (i in 1:length(observed_datasett)) {
  observed_dataset = as.numeric(strsplit(observed_datasett[i], "")[[1]])
  observed_list[[i]] = observed_dataset
}
observed_dataset=observed_list
```

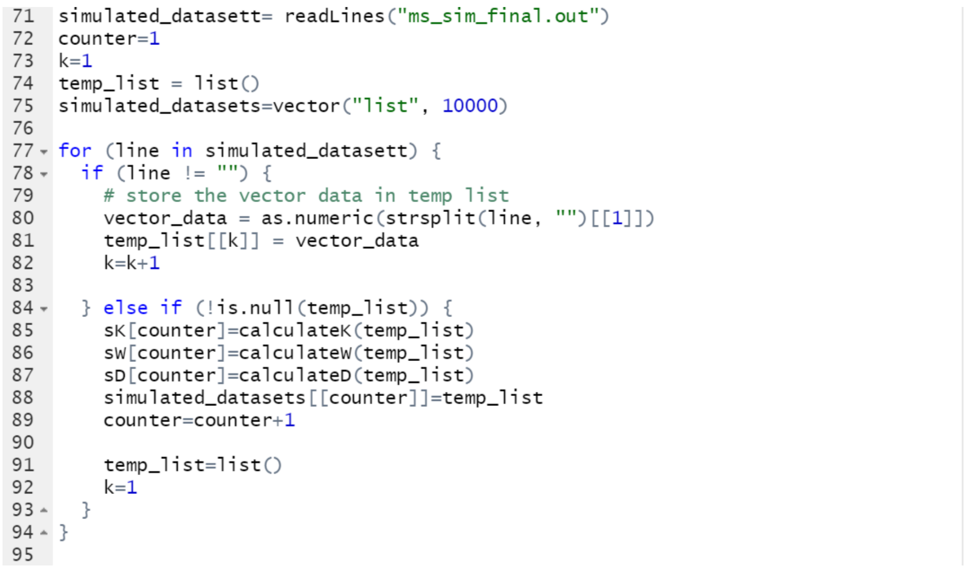
- For the simulated dataset:

  Read the file line by line.

  Each block of lines separated by two new lines represents a simulated dataset.

  Within each block, split the lines into individual values and convert them to numeric format.

  Store the values as a list of lists, where each element represents a simulated dataset.



```r
simulated_datasett= readLines("ms_sim_final.out")
counter=1
k=1
temp_list = list()
simulated_datasets=vector("list", 10000)

for (line in simulated_datasett) {
  if (line != "") {
    # store the vector data in temp list
    vector_data = as.numeric(strsplit(line, "")[[1]])
    temp_list[[k]] = vector_data
    k=k+1

  } else if (!is.null(temp_list)) {
    sK[counter]=calculateK(temp_list)
    sW[counter]=calculateW(temp_list)
    sD[counter]=calculateD(temp_list)
    simulated_datasets[[counter]]=temp_list
    counter=counter+1

    temp_list=list()
    k=1
  }
}

```

The results we got are :

- For the observed dataset:

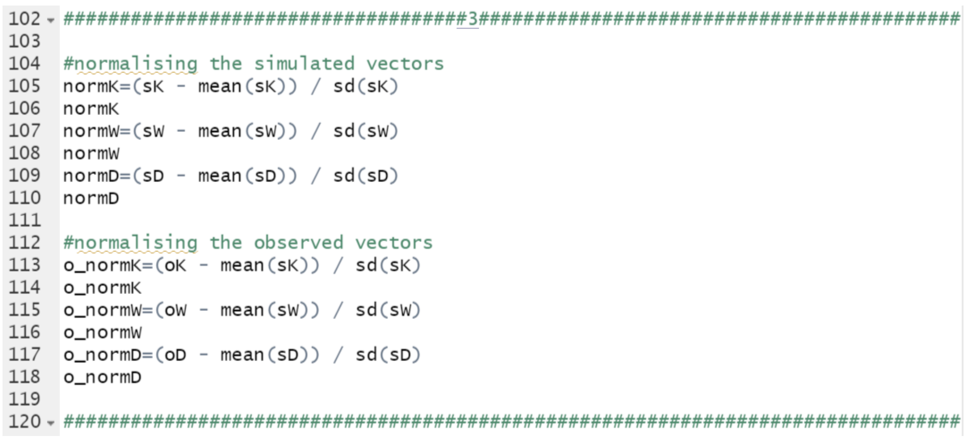
```
> oK=calculateK(observed_list)
> oK
[1] 14.28245
> oW=calculateW(observed_list)
> oW
[1] 29.46951
> oD=calculateD(observed_list)
> oD
[1] -1.848967
```

- For the simulated dataset:(indicatively)

```
> sK
  [1] 12.800000 14.977143  5.693061  6.582041  5.543673  8.227755  5.534694 12.904490  7.165714 13.493878
 [11]  7.586122 21.173878  7.986122  7.024490 18.196735  6.284082  4.517551 11.005714  7.707755 10.057143
 [21] 20.137143 16.893878 30.199184 23.528163  9.000816  7.299592  6.117551  7.768980  5.248163  9.746122
 [31]  4.994286  4.858776  4.160000  9.017959  6.852245  7.230204 16.894694 19.110204  5.706122 17.096327
 [41]  7.809796 12.561633 14.508571 10.933061  6.762449  7.511020  6.524082  6.248980 10.308571  5.792653
 [51] 20.431020 14.464490  6.020408  6.408980  4.711020  6.468571 24.201633  6.475918 14.875918 15.817959
> sW
  [1] 26.567212 28.129990 16.520788 17.860311 14.958010 19.646342 13.618487 28.353244 18.976580 28.129990
 [11] 17.860311 45.990301 18.976580 18.753326 39.962446 18.083565 14.064995 23.218404 21.655627 26.120705
 [21] 39.292684 38.622922 55.366964 48.892601 21.655627 21.209119 16.520788 20.316104 16.297534 20.985865
 [31] 17.190549 14.288249 14.511503 26.120705 16.297534 21.655627 35.943876 40.185700 15.404518 33.041575
 [41] 23.218404 30.809036 27.013720 24.557928 22.325389 18.976580 14.958010 12.055710 28.353244 21.878881
 [51] 45.767047 35.943876 19.646342 17.190549 16.074280 14.511503 56.259979 20.316104 33.264829 31.478798
> sD
  [1] -1.854456 -1.675688 -2.309298 -2.231761 -2.208323 -2.061178 -2.073719 -1.953070 -2.204557 -1.864658
 [11] -2.033072 -1.952534 -2.051424 -2.214371 -1.966281 -2.307146 -2.375072 -1.875327 -2.291470 -2.199788
 [21] -1.759453 -2.029813 -1.649109 -1.878897 -2.079035 -2.331736 -2.218764 -2.192715 -2.387529 -1.903580
 [31] -2.503777 -2.310765 -2.499464 -2.342097 -2.040922 -2.369926 -1.909527 -1.893535 -2.211855 -1.735821
 [41] -2.366078 -2.127137 -1.657323 -1.981241 -2.482445 -2.140104 -1.978349 -1.672265 -2.281252 -2.616649
 [51] -2.002990 -2.153136 -2.459627 -2.213354 -2.488042 -1.942039 -2.067687 -2.418688 -1.988688 -1.787630
```

Question 3

In order to calculate the mean and standard deviation of each vector we used the mean() and sd() functions and normalized each value in the vectors by subtracting the mean and dividing by the standard deviation.

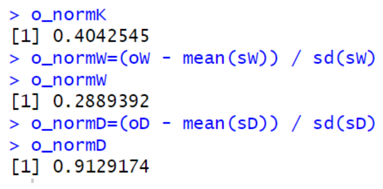


```
####################################3#########################################

#normalising the simulated vectors
normK=(sK - mean(sK)) / sd(sK)
normK
normW=(sW - mean(sW)) / sd(sW)
normW
normD=(sD - mean(sD)) / sd(sD)
normD

#normalising the observed vectors
o_normK=(oK - mean(sK)) / sd(sK)
o_normK
o_normW=(oW - mean(sW)) / sd(sW)
o_normW
o_normD=(oD - mean(sD)) / sd(sD)
o_normD

##############################################################################
```

:)

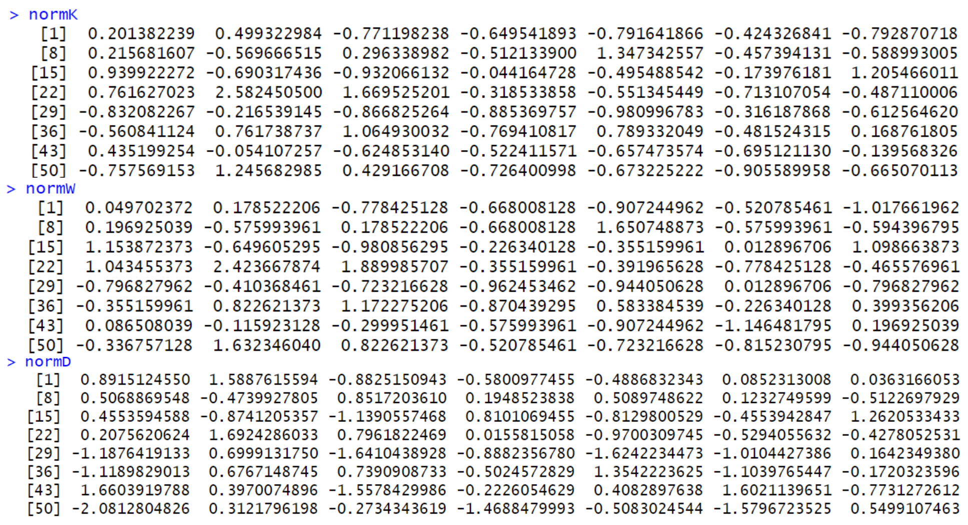
The results we got are :

- For the observed dataset:



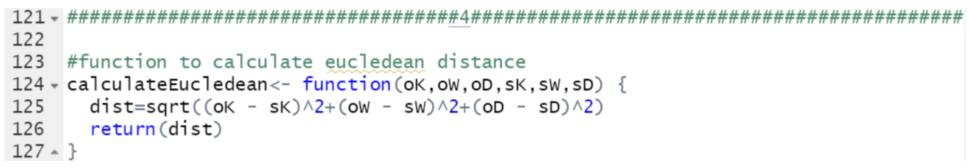
```
> o_normK
[1] 0.4042545
> o_normW=(oW - mean(sW)) / sd(sW)
> o_normW
[1] 0.2889392
> o_normD=(oD - mean(sD)) / sd(sD)
> o_normD
[1] 0.9129174
```

- For the simulated dataset:(indicatively)



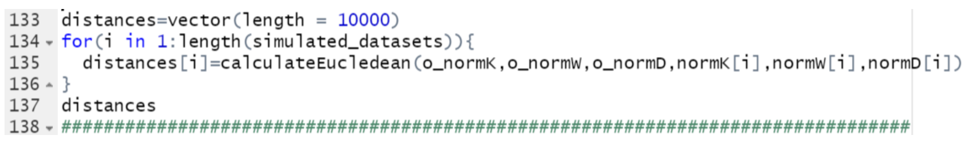
```
> normK
  [1]  0.201382239  0.499322984 -0.771198238 -0.649541893 -0.791641866 -0.424326841 -0.792870718
  [8]  0.215681607 -0.569666515  0.296338982 -0.512133900  1.347342557 -0.457394131 -0.588993005
 [15]  0.939922272 -0.690317436 -0.932066132 -0.044164728 -0.495488542 -0.173976181  1.205466011
 [22]  0.761627023  2.582450500  1.669525201 -0.318533858 -0.551345449 -0.713107054 -0.487110006
 [29] -0.832082267 -0.216539145 -0.866825264 -0.885369757 -0.980996783 -0.316187868 -0.612564620
 [36] -0.560841124  0.761738737  1.064930032 -0.769410817  0.789332049 -0.481524315  0.168761805
 [43]  0.435199254 -0.054107257 -0.624853140 -0.522411571 -0.657473574 -0.695121130 -0.139568326
 [50] -0.757569153  1.245682985  0.429166708 -0.726400998 -0.673225222 -0.905589958 -0.665070113
> normW
  [1]  0.049702372  0.178522206 -0.778425128 -0.668008128 -0.907244962 -0.520785461 -1.017661962
  [8]  0.196925039 -0.575993961  0.178522206 -0.668008128  1.650748873 -0.575993961 -0.594396795
 [15]  1.153872373 -0.649605295 -0.980856295 -0.226340128 -0.355159961  0.012896706  1.098663873
 [22]  1.043455373  2.423667874  1.889985707 -0.355159961 -0.391965628 -0.778425128 -0.465576961
 [29] -0.796827962 -0.410368461 -0.723216628 -0.962453462 -0.944050628  0.012896706 -0.796827962
 [36] -0.355159961  0.822621373  1.172275206 -0.870439295  0.583384539 -0.226340128  0.399356206
 [43]  0.086508039 -0.115923128 -0.299951461 -0.575993961 -0.907244962 -1.146481795  0.196925039
 [50] -0.336757128  1.632346040  0.822621373 -0.520785461 -0.723216628 -0.815230795 -0.944050628
> normD
  [1]  0.8915124550  1.5887615594 -0.8825150943 -0.5800977455 -0.4886832343  0.0852313008  0.0363166053
  [8]  0.5068869548 -0.4739927805  0.8517203610  0.1948523838  0.5089748622  0.1232749599 -0.5122697929
 [15]  0.4553594588 -0.8741205357 -1.1390557468  0.8101069455 -0.8129800529 -0.4553942847  1.2620533433
 [22]  0.2075620624  1.6924286033  0.7961822469  0.0155815058 -0.9700309745 -0.5294055632 -0.4278052531
 [29] -1.1876419133  0.6999131750 -1.6410438928 -0.8882356780 -1.6242234473 -1.0104427386  0.1642349380
 [36] -1.1189829013  0.6767148745  0.7390908733 -0.5024572829  1.3542223625 -1.1039765447 -0.1720323596
 [43]  1.6603919788  0.3970074896 -1.5578429986 -0.2226054629  0.4082897638  1.6021139651 -0.7731272612
 [50] -2.0812804826  0.3121796198 -0.2734343619 -1.4688479993 -0.5083024544 -1.5796723525  0.5499107463
```

Question 4



```
####################################4######################################################

#function to calculate eucledean distance
calculateEucledean<- function(oK,oW,oD,sK,sW,sD) {
  dist=sqrt((oK - sK)^2+(oW - sW)^2+(oD - sD)^2)
  return(dist)
}
```

In this query, we calculated the Euclidean distances between the observed dataset and each simulated dataset and store d everything in distances vector.

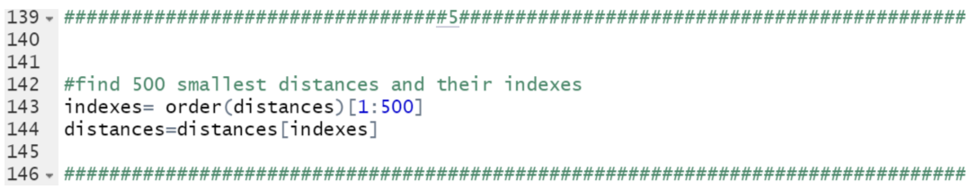


```
distances=vector(length = 10000)
for(i in 1:length(simulated_datasets)){
  distances[i]=calculateEucledean(o_normK,o_normW,o_normD,normK[i],normW[i],normD[i])
}
distances
#################################################################################
```

Question 5

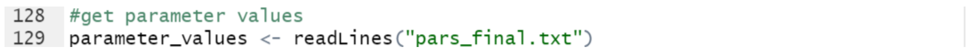
Here we needed to sort the distance_values vector in ascending order and extract the first 500 distances and record their indexes.

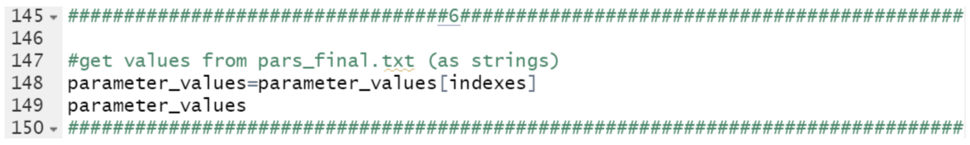
:)


```
##################################5############################################

#find 500 smallest distances and their indexes
indexes= order(distances)[1:500]
distances=distances[indexes]

###############################################################################
```

Question 6

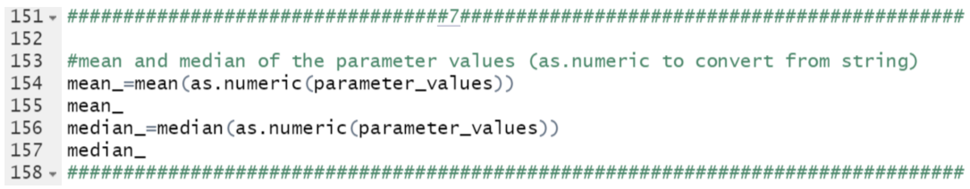
By using the indexes from Q5, we read the corresponding parameter values from "pars_final.txt" .

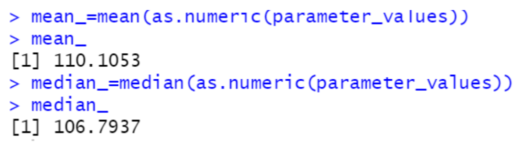

```
#get parameter values
parameter_values <- readLines("pars_final.txt")
```


```
##################################6############################################

#get values from pars_final.txt (as strings)
parameter_values=parameter_values[indexes]
parameter_values
###############################################################################
```

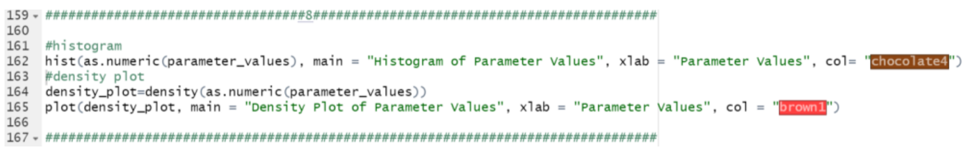
Question 7

We used simple R functins to calculate the mean and median of the 500 parameter values.


```
##################################7############################################

#mean and median of the parameter values (as.numeric to convert from string)
mean_=mean(as.numeric(parameter_values))
mean_
median_=median(as.numeric(parameter_values))
median_
###############################################################################
```

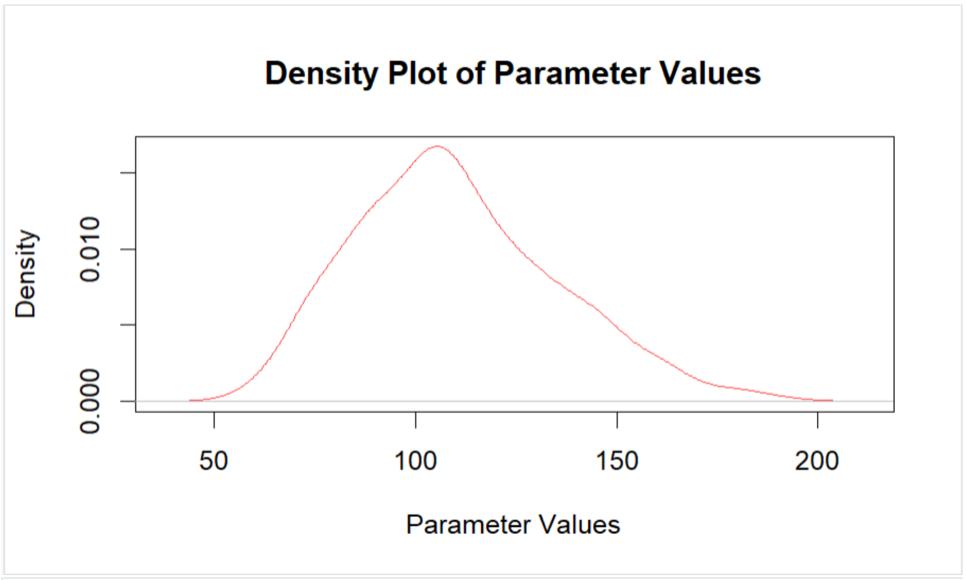

```
> mean_=mean(as.numeric(parameter_values))
> mean_
[1] 110.1053
> median_=median(as.numeric(parameter_values))
> median_
[1] 106.7937
```

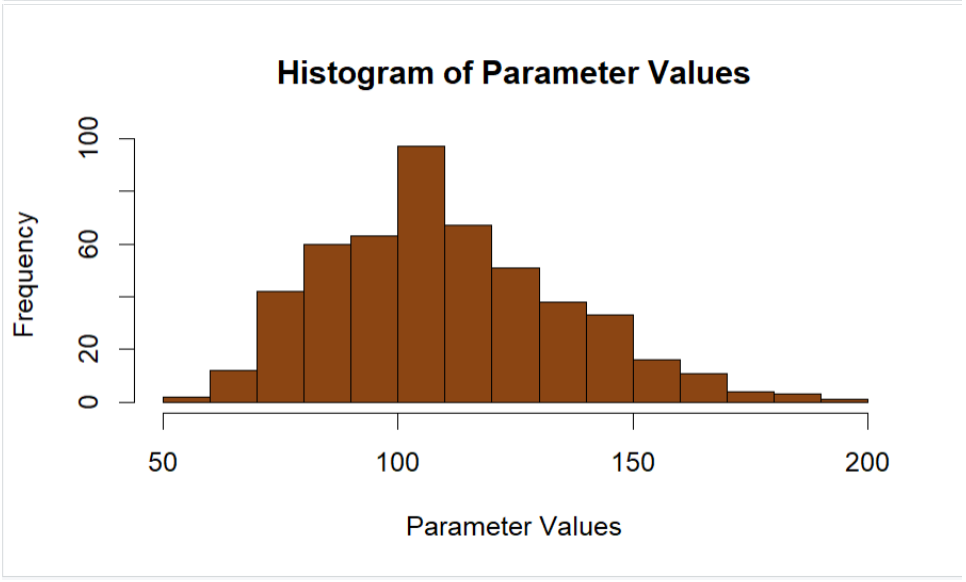
Question 8


```
##################################8############################################

#histogram
hist(as.numeric(parameter_values), main = "Histogram of Parameter Values", xlab = "Parameter Values", col= "chocolate4")
#density plot
density_plot=density(as.numeric(parameter_values))
plot(density_plot, main = "Density Plot of Parameter Values", xlab = "Parameter Values", col = "brown1")

###############################################################################
```

:)

Running the code above, we get :





Question 9

Looking at the results, we observe a bell shaped histogram. A bell-shaped distribution suggests that the distances follow a normal distribution, with a concentration of values around the mean. Most of the values are concetrated around 110. A mean value of 110 implies a significant increase in the number of infections over a year. It indicates that the number of new infections is significantly larger than the number of recoveries or deaths, leading to a rapid expansion of the SARS-CoV-2 population. Moreover, it should be noted that since the mean value is close to the median value of 106, it indicates that the majority of the simulated datasets are within a reasonable distance from the observed dataset. By comparing the observed growth rate in the simulated datasets with historical data and established knowledge about viral dynamics, we can see a consistent pattern. The high mean growth rate in the simulated datasets aligns with what we have observed in the real world during the COVID-19 pandemic.

:)