Assignment 7

JUFENG YANG

2022-11-16

Load packages

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

1. Maximum likelihood estimates

1.1 Maximum likelihood estimates for Red tailed hawks

In this question we will fit a Gaussian model to a Red-Tailed hawk data set. First load the Hawks data set as follows:

```
library(Stat2Data)
data("Hawks")
#Hawks
```

1.1 (Q.1)

```
RedTailedDf <- Hawks %>% filter(Species == "RT") %>% select(c("Weight", "Tail", "Wing"))
```

1.1 (Q.2)

We model the tail follow the N distribution. And the The maximum likelihood estimates for μ_0 is given by $\mu_{MLE} = \frac{1}{n}\sum_1^n X_i$ and the maximum likelihood estimate for σ_0^2 is given by $\sigma_{MLE}^2 = \frac{1}{n}\sum_0^n (Xi - \mu_M LE)^2$.

```
mu_MLE <- RedTailedDf $ Tail %>% mean()
Tails <- RedTailedDf $ Tail
sigma_2_MLE <- RedTailedDf $ Tail %>% var() * ((length(Tails)) - 1) / (length(Tails))

mu_MLE
```

```
## [1] 222.149
```

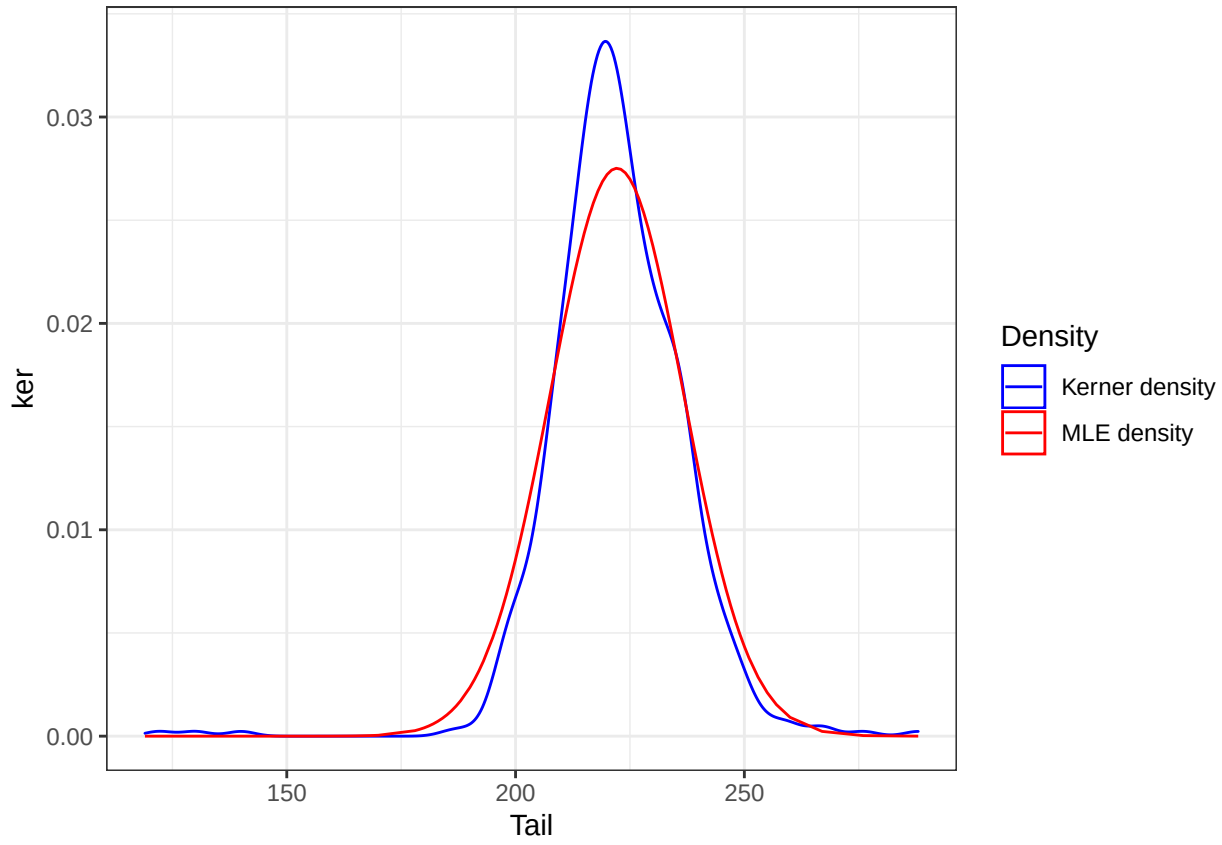
```
sigma_2_MLE
```

```
## [1] 210.2031
```

1.1 (Q.3)

Next generate a plot which compares the probability density function for your fitted Gaussian model for the tail length of the Red-Tailed hawks with a kernel density plot.

```
x = Hawks$Tail
ker_density <- data.frame(x = x, ker = dnorm(x, mean = mu_MLE, sd = sqrt(sigma_2_MLE)))
density_comparise <- ggplot() + theme_bw() +
  geom_density(data = RedTailedDf, aes(x = Tail, colour = "Kerner density")) +
  geom_line(data = ker_density, aes(x = x, y = ker, colour = "MLE density")) +
  scale_colour_manual(name = "Density",
                      values = c("Kerner density" = "blue", "MLE density" = "red") )
density_comparise
```



1.2 Unbiased estimation of the population variance

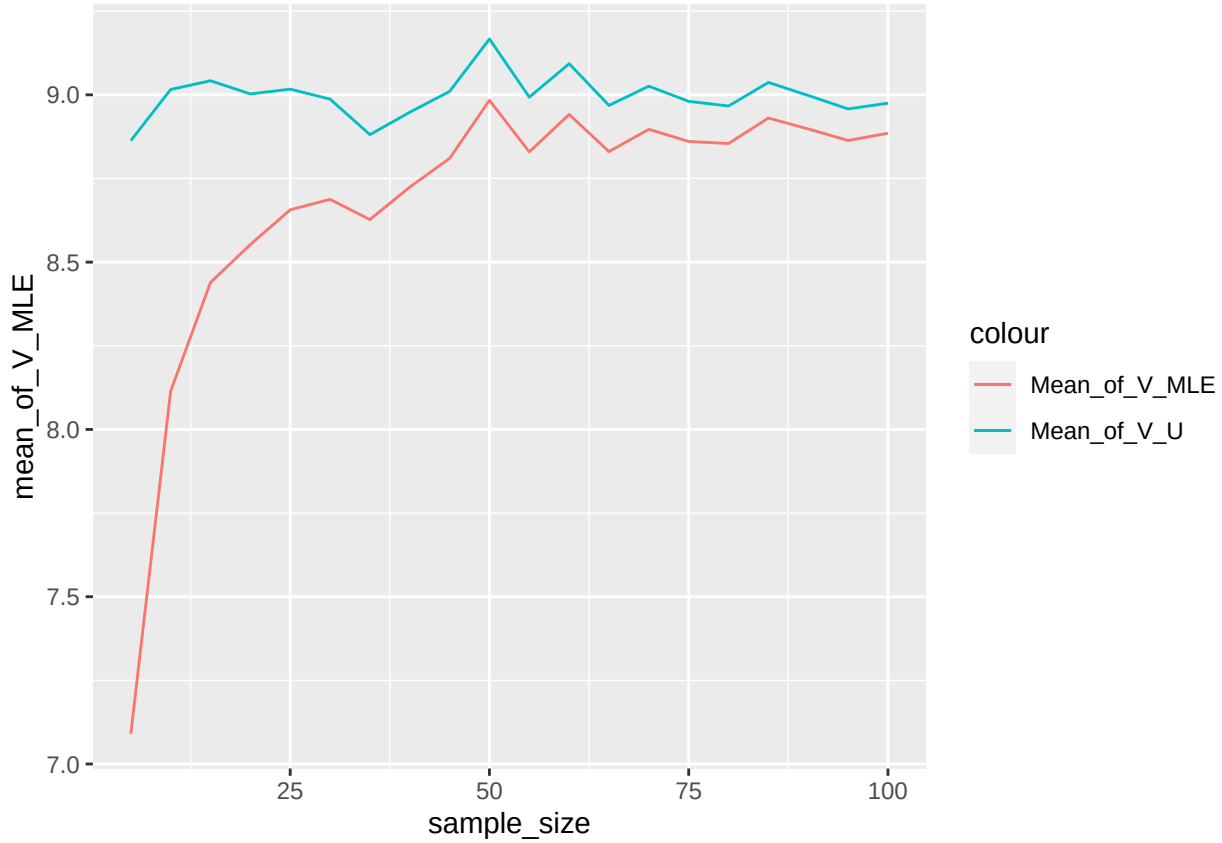
1.2 (Q.1)

```
Trails <- 1000
sample_size <- seq(5, 100, by = 5)

get_V <- function(sample, flag){
  mn = mean(sample)
  if(flag == "V_U"){
    variance = sum(map_dbl(.x = sample, .f = ~(.x - mn)^2)) / (length(sample) - 1)
  }else if(flag == "V_MLE"){
    variance = sum(map_dbl(.x = sample, .f = ~(.x - mn)^2)) / (length(sample))
  }

  #variance = var(sample) * length(sample) / (length(sample)-1)
  return(variance)
}

MLE_U_Compares_df <- crossing(Trails = seq(0,Trails), Sample_size = sample_size) %>%
  mutate(Sample = map(.x = Sample_size, .f = ~rnorm(.x, mean = 1, sd = 3))) %>%
  #mutate(V_MLE = map_dbl(.x = Sample, .f = ~get_V(.x, flag = "V_MLE"))) %>%
  #mutate(V_U = map_dbl(.x = Sample, .f = ~get_V(.x, flag = "V_U"))) %>%
  mutate(V_U = map_dbl(.x = Sample, .f = var)) %>%
  mutate(V_MLE = map2_dbl(.x = Sample, .y = Sample_size, .f = ~(var(.x) * (.y - 1) / .y)))

MLE_U_Compares_df
```

```
## # A tibble: 20,020 x 5
##    Trails Sample_size Sample       V_U V_MLE
##     <int>       <dbl> <list>     <dbl> <dbl>
##  1      0           5 <dbl [5]>   5.61  4.49
##  2      0          10 <dbl [10]>  7.71  6.94
##  3      0          15 <dbl [15]>  4.92  4.59
##  4      0          20 <dbl [20]> 11.5  10.9
##  5      0          25 <dbl [25]>  8.85  8.50
##  6      0          30 <dbl [30]>  8.87  8.57
##  7      0          35 <dbl [35]> 10.4  10.1
##  8      0          40 <dbl [40]>  9.03  8.80
##  9      0          45 <dbl [45]>  8.54  8.35
## 10      0          50 <dbl [50]> 11.0  10.8
## # ... with 20,010 more rows
```

```
mean_MLE_U_Compares_df <- MLE_U_Compares_df %>%
  group_by(Sample_size) %>%
  summarise(mean_of_V_MLE = mean(V_MLE), mean_V_U = mean(V_U))

#ggplot() + geom_line(data = MLE_U_Compares_df, aes(x = sample_size, y = mean_of_V_MLE, color = "V_MLE"
plot_MLE_U <- ggplot() +
  geom_line(data = mean_MLE_U_Compares_df,
            aes(x = sample_size, y = mean_of_V_MLE, color = "Mean_of_V_MLE")) +
  geom_line(data = mean_MLE_U_Compares_df,
```

```
            aes(x = sample_size, y = mean_V_U, color = "Mean_of_V_U"))
plot_MLE_U
```



1.2 (Q.2)

```
mean_MLE_U_Compares_df <- MLE_U_Compares_df %>%
  group_by(Sample_size) %>%
  summarise(mean_of_V_MLE = mean(V_MLE), mean_V_U = mean(V_U))
mean_MLE_U_Compares_df
```

```
## # A tibble: 20 x 3
##    Sample_size mean_of_V_MLE mean_V_U
##          <dbl>         <dbl>    <dbl>
##  1           5          7.09     8.86
##  2          10          8.11     9.02
##  3          15          8.44     9.04
##  4          20          8.55     9.00
##  5          25          8.66     9.02
##  6          30          8.69     8.99
##  7          35          8.63     8.88
##  8          40          8.72     8.95
##  9          45          8.81     9.01
## 10          50          8.98     9.17
```

```
## 11          55          8.83     8.99
## 12          60          8.94     9.09
## 13          65          8.83     8.97
## 14          70          8.90     9.03
## 15          75          8.86     8.98
## 16          80          8.85     8.97
## 17          85          8.93     9.04
## 18          90          8.90     9.00
## 19          95          8.86     8.96
## 20         100          8.89     8.98
```

From the content of data frame, we can know the mean_V_U always around 9, so the V_U can unbiased estimator for σ_0

1.3 Maximum likelihood estimation with the Poisson distribution

1.3 (Q.1)

for a sample $X_1.X_2, ..., X_n$, the likelihood function $l : (0, \infty) \to (0, \infty)$is given by:

$$l(\lambda) = e^{-n\lambda} \cdot \lambda^{n\overline{X}} \cdot \left(\prod_{i=1}^{n} \frac{1}{X_i!} \right)$$

The derivative of the log-likelihood: $\frac{\partial}{\partial \lambda} = \log l(\lambda)$

$$\begin{aligned}
\frac{\partial}{\partial \lambda} \log l(\lambda) &= \frac{\partial}{\partial \lambda} \log \left[e^{-n\lambda} \cdot \lambda^{n\overline{X}} \cdot \left(\prod_{i=1}^{n} \frac{1}{X_i!} \right) \right] \\
&= \frac{\partial}{\partial \lambda} \left[-n\lambda \ + \ n\overline{X} \log \lambda + \log \left(\prod_{i=1}^{n} \frac{1}{X_i!} \right) \right] \\
&= -n \ + \ n\overline{X} \cdot \frac{1}{\lambda} \\
&= n \cdot (\frac{\overline{X}}{\lambda} - 1)
\end{aligned}$$

1.3 (Q.2)

From last question, we get $\frac{\partial}{\partial \lambda} \log l(\lambda) = n \cdot (\frac{\overline{X}}{\lambda} - 1)$ When $\frac{\partial}{\partial \lambda} \log l(\lambda) = 0$, we can make $\log l(\lambda)$ reaches its maximum. So let $\frac{\overline{X}}{\lambda} = 1$, so $\lambda = \overline{X}$, Hence, the maximum likelihood estimate for the true parameter λ^0 is $\hat{\lambda}_{MLE} = \overline{X}$.

1.3 (Q.3)

```
lambda <- 0.5
sample_size <- 1000
r_V_pos <- rpois(sample_size, lambda = lambda)
mn_r_v <- mean(r_V_pos)
mn_r_v
```

```
## [1] 0.485
```

Run the code above, we set $\lambda_0 = 0.5$, and generate a random variables followed by Poisson distribution, we compute the mean of those variables, and the mean is 0.501, so it prove $\overline{X} = \lambda_0$

1.3 (Q.4)

```
VonBortkiewicz <- read.csv("D:/Master in UoB/TB1 of UoB/SCEM/R project/Week8-Lab/VonBortkiewicz.csv")
fatalities <- VonBortkiewicz $ fatalities
lambda <- mean(fatalities)
density_poisson <- dpois(fatalities, lambda = lambda)
density_prob <- data.frame(fatalities, prob = density_poisson) %>% unique()
density_prob
```

```
##    fatalities        prob
## 1           0 0.496585304
## 8           1 0.347609713
## 15          2 0.121663399
## 72          3 0.028388127
## 82          4 0.004967922
```

```
# density_poisson
```

Run code above, we can know the fatalities is 0, the prob = 0.496585304, the fatalities = 1, the prob = 0.347609713, so when fatalities = 0, the probability is 0.4966.

1.4 Maximum likelihood estimation for the exponential distribution

1.4 (Q.1)

Recall from our last assignment that given a positive real number $\lambda > 0$, an exponential random variable X with parameter λ is a continuous random variable with density $p_\lambda : R \to (0, \infty)$ define by

$$p_\lambda(x) \begin{cases} 0 & if \quad x < 0 \\ \\ \lambda e^{-\lambda x} & if \quad x \geq 0 \end{cases}$$

We can get the maximum likelihood function is $l(\lambda) = \prod_{i=1}^{i=n} p_\lambda(x \geq 0)$ So as bellow shows:

$$l(\lambda) = \prod_{i=1}^{i=n} p_\lambda(x \geq 0)$$

Using log in two sides:

$$\log l(\lambda) = \log \prod_{i=1}^{i=n} p_\lambda(x \geq 0)$$

$$= \log \prod_{i=1}^{i=n} \lambda e^{-\lambda x}$$

Using derivate in two sides:

$$\frac{\partial}{\partial \lambda} \log l(\lambda) = \frac{\partial}{\partial \lambda} \log \prod_{i=1}^{i=n} \lambda e^{-\lambda x}$$

$$= \frac{\partial}{\partial \lambda} \left(\log \lambda^n + \log e^{-\lambda \cdot \sum_{i=1}^{n} x_i} \right)$$

$$= \frac{\partial}{\partial \lambda} \left(\log \lambda^n + -\lambda \cdot \sum_{i=1}^{n} x_i \right)$$

$$= \frac{n}{\lambda} - \lambda \cdot \sum_{i=1}^{n} x_i$$

$$= \frac{n}{\lambda} - n\overline{x}$$

Let equation above equal to 0:

$$0 = \frac{n}{\lambda} - n\overline{x}$$

$$\Rightarrow \lambda = \frac{1}{\overline{X}}$$

1.4 (Q.2)

```
CustomerPurchase <- read.csv("D:/Master in UoB/TB1 of UoB/SCEM/R project/Week8-Lab/CustomerPurchase.csv
CustomerPurchase %>% head(30)
```

```
##    Time Purchase
## 1   564     3.25
## 2   571   504.85
## 3   578     7.60
## 4   600    43.45
## 5   745     9.30
## 6   806   352.80
```

```
## 7   833   182.05
## 8   881     8.55
## 9   888    65.35
## 10  958   211.00
## 11  996   471.30
## 12 1058    76.30
## 13 1279     0.05
## 14 1332     0.00
## 15 1384   406.50
## 16 1478    51.55
## 17 1511   740.20
## 18 1528    24.55
## 19 1557    13.35
## 20 1675    60.25
## 21 1707   168.20
## 22 1722    76.35
## 23 1750    41.10
## 24 1755    82.40
## 25 1758    94.65
## 26 1787   123.90
## 27 1985     0.00
## 28 2044    84.55
## 29 2094    20.50
## 30 2166     0.10
```

```
Time <- CustomerPurchase $ Time
Time %>% head(30)
```

```
##  [1]  564  571  578  600  745  806  833  881  888  958  996 1058 1279 1332 1384
## [16] 1478 1511 1528 1557 1675 1707 1722 1750 1755 1758 1787 1985 2044 2094 2166
```

```
time_diffs <- array(data = NA, dim = length(Time))
for(i in seq(1, length(Time) - 1, by = 1)){
  time_diffs[i] <- Time[i+1] - Time[i]
}
time_diffs %>% head(30)
```

```
##  [1]   7   7  22 145  61  27  48   7  70  38  62 221  53  52  94  33  17  29 118
## [20]  32  15  28   5   3  29 198  59  50  72   2
```

```
CustomerPurchase <- CustomerPurchase %>% mutate(time_diffs)
CustomerPurchase %>% head(30)
```

```
##    Time Purchase time_diffs
## 1   564     3.25          7
## 2   571   504.85          7
## 3   578     7.60         22
## 4   600    43.45        145
## 5   745     9.30         61
## 6   806   352.80         27
## 7   833   182.05         48
## 8   881     8.55          7
```

```
## 9   888    65.35         70
## 10  958   211.00         38
## 11  996   471.30         62
## 12 1058    76.30        221
## 13 1279     0.05         53
## 14 1332     0.00         52
## 15 1384   406.50         94
## 16 1478    51.55         33
## 17 1511   740.20         17
## 18 1528    24.55         29
## 19 1557    13.35        118
## 20 1675    60.25         32
## 21 1707   168.20         15
## 22 1722    76.35         28
## 23 1750    41.10          5
## 24 1755    82.40          3
## 25 1758    94.65         29
## 26 1787   123.90        198
## 27 1985     0.00         59
## 28 2044    84.55         50
## 29 2094    20.50         72
## 30 2166     0.10          2
```

```
#lambda <- mean(fatalities)
#density_poisson <- dpois(fatalities, lambda = lambda)
#density_prob <- data.frame(fatalities, prob = density_poisson) %>% unique()
#density_prob
```

1.4 (Q.3)

Model the sequence of differences in purchase times X1, · · · , Xn as independent and identically distributed exponential random variables. Compute the maximum likelihood estimate of the rate parameter λ_{MLE}.

```
lambda_MLE <- 1 / (mean(time_diffs[1:length(time_diffs)-1]))
lambda_MLE
```

```
## [1] 0.02007792
```

1.4 (Q.4)

Use your fitted exponential model to give an estimate of the probability of an arrival time in excess of one minute. You may wish to make use of the pexp() function.

```
pexp(q = 60, rate = lambda_MLE)
```

```
## [1] 0.7002107
```

Run the above code, the probability more than 60s is 0.7002107

2. Confidence intervals

2.1 Student's t-confidence intervals

2.1 (Q.1)

Sample mean do not change the width of confident interval, it just let the confident interval move to left or right. if sample standard become bigger, the confident interval will become wider. if sample size become bigger, the width of confident interval will become bigger.

2.1 (Q.2)

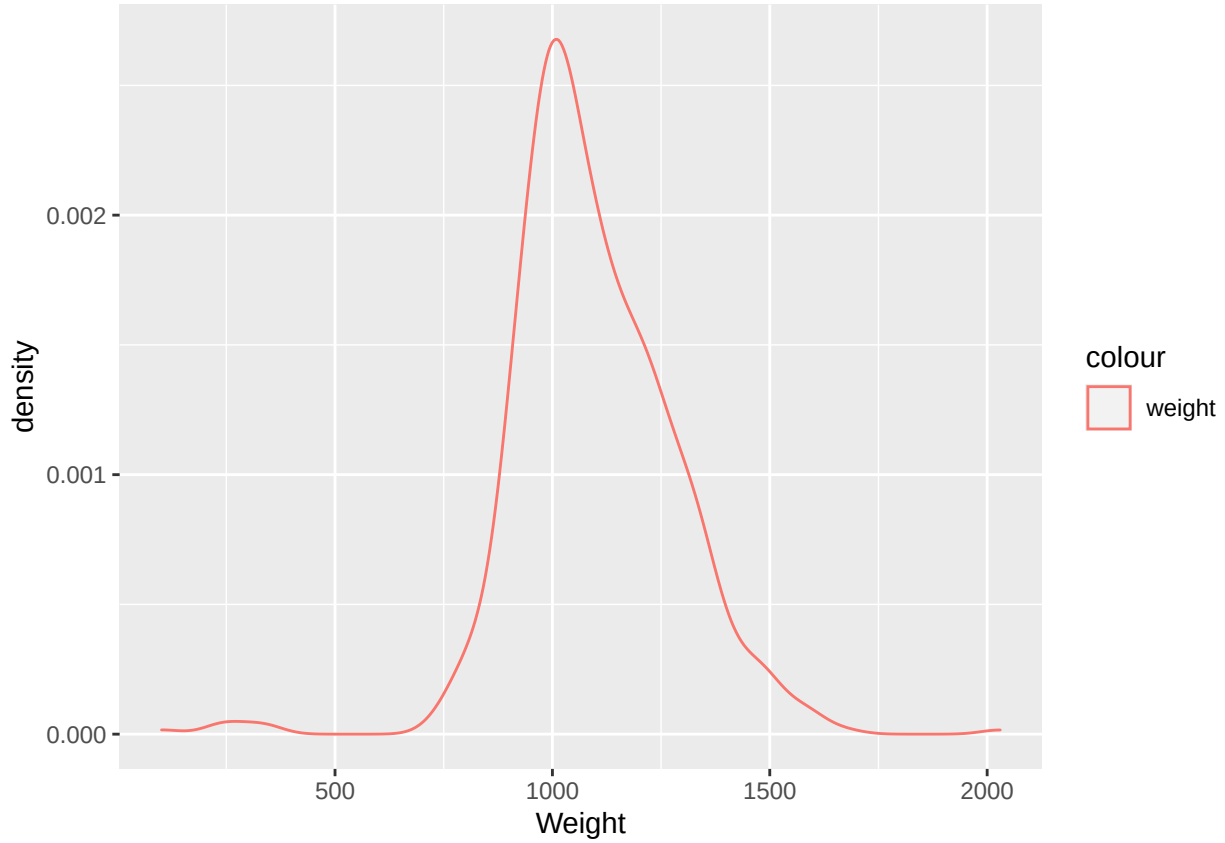
```
weight <- RedTailedDf $ Weight
weight <- weight[!is.na(weight)]

alpha <- 0.01
sample_size <- length(weight) # Weight is a given vector
sample_mean <- mean(weight)
sample_sd <- sd(weight)
t <- qt(1-alpha/2,df=sample_size-1)

# confidence interval
confidence_interval_l <- sample_mean-t*sample_sd/sqrt(sample_size)
confidence_interval_u <- sample_mean+t*sample_sd/sqrt(sample_size)
confidence_interval <- c(confidence_interval_l,confidence_interval_u)
confidence_interval
```

```
## [1] 1073.984 1114.877
```

2.1 (Q.3)

```
ggplot() +  geom_density(data = RedTailedDf, aes(x = Weight, colour = "weight"))
```

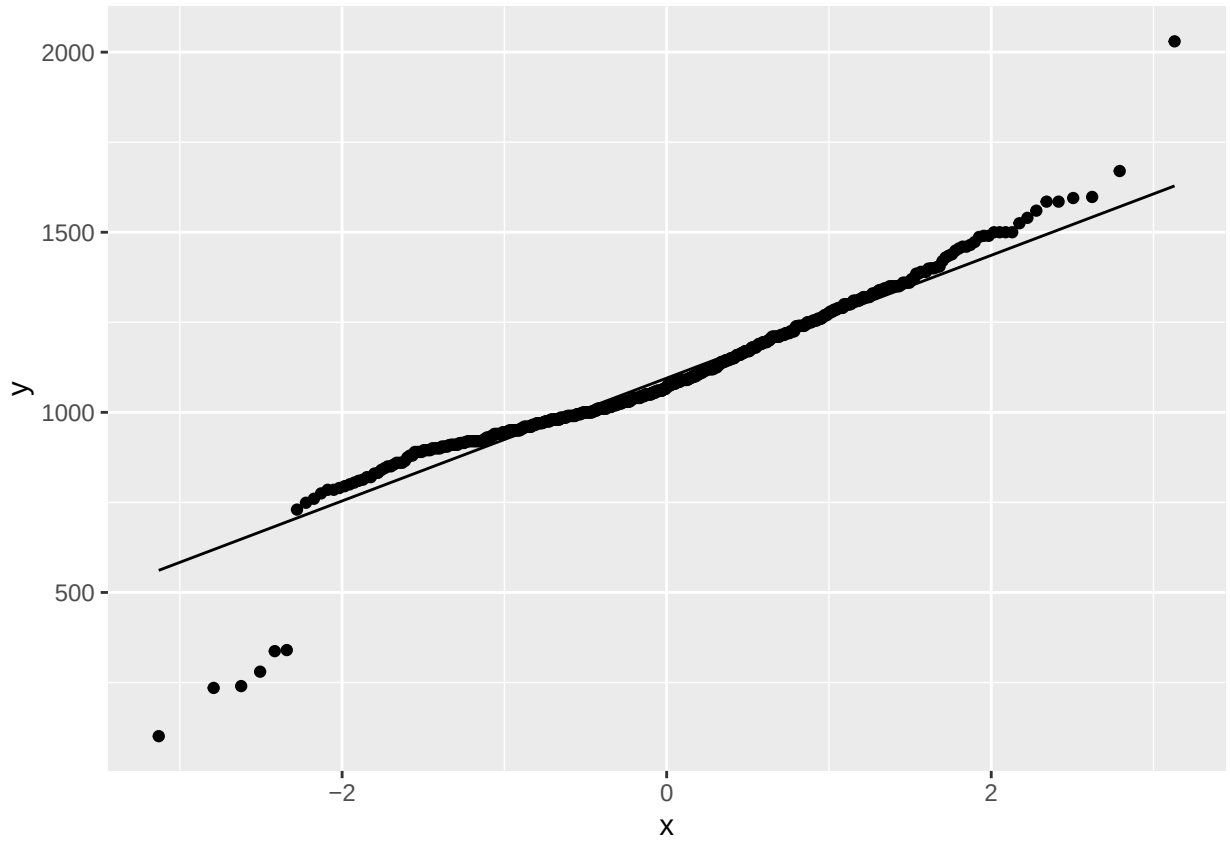
```
## Warning: Removed 5 rows containing non-finite values (stat_density).
```




```
ggplot(data = RedTailedDf, aes(sample = Weight)) + stat_qq() + stat_qq_line()
```

```
## Warning: Removed 5 rows containing non-finite values (stat_qq).
```

```
## Warning: Removed 5 rows containing non-finite values (stat_qq_line).
```




```
#+geom_line(data = ker_den, aes(x = weight, y = kern, colour = "weight")) +
```

2.2 Investigating coverage for Student's t intervals

```
student_t_confidence_interval <- function(sample, confidence_level){

sample <- sample[!is.na(sample)] # remove any missing values
n <- length(sample) # compute sample size
mu_est <- mean(sample) # compute sample mean
sig_est <- sd(sample) # compute sample sd
alpha = 1 - confidence_level # alpha from gamma
t <- qt(1 - alpha / 2, df = n-1) # get student t quantile
l = mu_est - (t / sqrt(n)) * sig_est # lower
u = mu_est + ( t / sqrt(n)) * sig_est # upper
return(c(l,u))

}
```

2.2 (Q.1)

```
num_trials <- 100000
sample_size <- 30
```

```r
mu_0 <- 1
sigma_0 <- 3
alpha <- seq(0.01, 0.1, by = 0.01)
set.seed(0) # set random seed for reproducibility
single_alpha_coverage_simulation_df <- function(num_trials,
                                                sample_size,
                                                mu_0, sigma_0,
                                                alpha){

  midian_df <- data.frame(trial=seq(num_trials)) %>%
  # generate random Gaussian samples:
  mutate(sample=map(.x=trial, .f = ~rnorm(n = sample_size,
                                          mean = mu_0,
                                          sd = sigma_0))) %>%
  # generate confidence intervals:
  mutate(ci_interval = map(.x = sample,
                           .f = ~student_t_confidence_interval(.x, 1 - alpha)))%>%
  # check if interval covers mu_0:
  mutate(cover = map_lgl(.x = ci_interval,
                         .f = ~((min(.x) <= mu_0) & (max(.x) >= mu_0))))%>%
  # compute interval length:
  mutate(ci_length = map_dbl(.x = ci_interval,
                             .f = ~(max(.x) - min(.x))))
  # estimate of coverage probability:
  covers <- midian_df $ cover
  prob <- covers %>% mean()
  return(prob)
}

compares_alpha <- data.frame(confident_level = 1 - alpha) %>%
  mutate(prob_confident = map_dbl(.x = alpha,
                                  .f = ~single_alpha_coverage_simulation_df(num_trials = 100000,
                                                                            sample_size = 30,
                                                                            mu_0 = 1,
                                                                            sigma_0 = 3,
                                                                            alpha = .x)))

compares_alpha
```

```
##    confident_level prob_confident
## 1             0.99        0.99020
## 2             0.98        0.98043
## 3             0.97        0.97010
## 4             0.96        0.95968
## 5             0.95        0.94927
## 6             0.94        0.94020
## 7             0.93        0.93070
## 8             0.92        0.91975
## 9             0.91        0.91000
## 10            0.90        0.89970
```

```r
# prob_confodent <- array(data = NA, dim =  length(alpha))
# i <- 1
#
# for(alpha_i in alpha){
#    probs <- single_alpha_coverage_simulation_df(num_trials = 100000, sample_size = 30, mu_0 = 1, sigm
#    prob_confodent[i] <- probs
#    i = i + 1
# }
#
#
# compares_alpha <- data.frame(confident_level = 1 - alpha, prob_confodent)
```

It is easy to know the $P\{L_\alpha(X_1, \dots, X_n) \leq \mu_0 \leq R_\alpha(X_1, \dots, X_n)\}$ has the same trend varies with 1 - α

2.2 (Q.2)

```r
num_trials <- 100000
sample_size <- 30
mu_0 <- 1
sigma_0 <- 3
alpha <- seq(0.01, 0.1, by = 0.01)
set.seed(0) # set random seed for reproducibility
single_alpha_coverage_simulation_df <- function(num_trials,
                                                sample_size,
                                                mu_0, sigma_0,
                                                alpha){

  midian_df <- data.frame(trial=seq(num_trials)) %>%
  # generate random Gaussian samples:
  mutate(sample=map(.x=trial, .f = ~rnorm(n = sample_size,
                                          mean = mu_0,
                                          sd = sigma_0))) %>%
  # generate confidence intervals:
  mutate(ci_interval = map(.x = sample,
                           .f = ~student_t_confidence_interval(.x, 1 - alpha)))%>%
  # check if interval covers mu_0:
  mutate(cover = map_lgl(.x = ci_interval,
                         .f = ~((min(.x) <= mu_0) & (max(.x) >= mu_0))))%>%
  # compute interval length:
  mutate(ci_length = map_dbl(.x = ci_interval, .f = ~(max(.x) - min(.x))))
  # estimate of coverage probability:
  ci_lengths <- midian_df $ ci_length
  ave_len <- ci_lengths %>% mean()
  return(ave_len)
}

ave_ci_length <- data.frame(confident_level = 1 - alpha) %>%
  mutate(ave_ci_length = map_dbl(.x = alpha,
                                 .f = ~single_alpha_coverage_simulation_df(num_trials = 100000,
                                                                           sample_size = 30,
                                                                           mu_0 = 1,
```

```
                                                    sigma_0 = 3,
                                                    alpha = .x)))

ave_ci_length
```

```
##    confident_level ave_ci_length
## 1             0.99      2.992652
## 2             0.98      2.674456
## 3             0.97      2.479760
## 4             0.96      2.335615
## 5             0.95      2.221197
## 6             0.94      2.124452
## 7             0.93      2.043288
## 8             0.92      1.970996
## 9             0.91      1.905974
## 10            0.90      1.845336
```

3. One sample hypothesis testing

3.1 One sample t-test on penguins data

```
library(palmerpenguins)
```

```
## Warning:   'palmerpenguins' R 4.2.2
```

```
data("penguins")

bill_Adelie <- penguins %>% filter(species == "Adelie") %>% pull(bill_length_mm)
bill_Adelie <- bill_Adelie[!is.na(bill_Adelie)]

t.test(bill_Adelie, mu = 40)
```

```
##
##  One Sample t-test
##
## data:  bill_Adelie
## t = -5.5762, df = 150, p-value = 1.114e-07
## alternative hypothesis: true mean is not equal to 40
## 95 percent confidence interval:
##  38.36312 39.21966
## sample estimates:
## mean of x
##  38.79139
```

The p-value is 1.114e-07 is less than 0.01, so we reject this hypothesis and conclude $\mu_0 \neq 40$

3.2 Implementing a one-sample t-test

Create a t test:

```
my_t_test <- function(x, mu){
  T <- (mean(x) - mu) / (sd(x) / sqrt(length(x)))
  p_value <- 2 * (1 - pt(abs(T), df = length(x) - 1))
  #mn_p <- mean(p_value)
  return(p_value)
}

my_t_test(x = bill_Adelie, mu = 40)
```

```
## [1] 1.114322e-07
```

The function above is the t-test function I created, when the $\mu = 40$ is tested, the p-value is 1.14322e-07. The p-value is same as the in-built function t-test.