# Assignment 2

EMATM0061: Statistical Computing and Empirical Methods, TB1, 2022

Dr. Rihuan Ke

## Introduction

### Create an R Markdown for assignment

First, it is recommended that you create a single R Markdown document to include your solutions, with headings created by heading codes such as "## 1.1 (Q1)", "## 3 (Q1)", etc.

You are not required to hand in this R Markdown document, but it is a good practice to use R Markdown to organize your code and results.

### Load packages

Then we need to load two packages, namely Stat2Data and tidyverse, before answering the questions. If they haven't been installed in your computer, please use `install.packages()` to install them first.

1. Load the tidyverse package:

```
library(tidyverse)
```

2. Load the Stat2Data package and then the dataset Hawks:

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

# 1. Data Wrangling

This part is mainly about data wrangling. Basic concepts of data wrangling can be found in lecture 4.

## 1.1 Select and filter

**(Q1)**. Use a combination of the **select()** and **filter()** functions to generate a data frame called "hSF" which is a sub-table of the original Hawks data frame, such that

1. Your data frame should include the columns:
   a) "Wing"
   b) "Weight"
   c) "Tail"
2. Your data frame should contain a row for every hawk such that:
   a) They belong to the species of Red-Tailed hawks
   b) They have weight at least 1kg.
3. Use the pipe operator to simplify your code.

The data frame should look like this:

```
##   Wing Weight Tail
## 1  412   1090  230
## 2  412   1210  210
```

```
## 3  405   1120  238
## 4  393   1010  222
## 5  371   1010  217
```

**(Q2)** How many variables does the data frame hSF have? What would you say to communicate this information to a Machine Learning practitioner?

How many examples does the data frame hSF have? How many observations? How many cases?

## 1.2 The arrange function

**(Q1)** Use the arrange() function to sort the hSF data frame created in the previous section so that the rows appear in order of increasing wing span.

Then use the head command to print out the top five rows of your sorted data frame. Your results should look something like this:

```
##    Wing Weight Tail
## 1  37.2   1180  210
## 2 111.0   1340  226
## 3 199.0   1290  222
## 4 241.0   1320  235
## 5 262.0   1020  200
```

## 1.3 Join and rename functions

The species of Hawks within the data frame have been indicated via a two-letter code (e.g., RT, CH, SS). The correspondence between these codes and the full names is given by the following data frame:

```
##   species_code species_name_full
## 1           CH          Cooper's
## 2           RT        Red-tailed
## 3           SS     Sharp-shinned
```

**(Q1)**. Use **data.frame()** to create a data frame that is called **hawkSpeciesNameCodes** and is the same as the above data frame (i.e., containing the correspondence between codes and the full species names).

**(Q2)**. Use a combination of the functions **left_join()**, the **rename()** and the **select()** functions to create a new data frame called "hawksFullName" which is the same as the "Hawks" data frame except that the Species column contains the full names rather than the two-letter codes.

**(Q3)**. Use a combination of the **head()** and **select()** functions to print out the top seven rows of the columns "Species", "Wing" and "Weight" of the data frame called **"hawksFullName"**. Do this without modifying the data frame you just created. Your result should something like this:

```
##         Species Wing Weight
## 1    Red-tailed  385    920
## 2    Red-tailed  376    930
## 3    Red-tailed  381    990
## 4      Cooper's  265    470
## 5 Sharp-shinned  205    170
## 6    Red-tailed  412   1090
## 7    Red-tailed  370    960
```

Does it matter what type of join function you use here?

In what situations would it make a difference?

## 1.4 The mutate function

Suppose that the fictitious "Healthy Hawks Society"[1] has proposed a new measure called the "bird BMI" which attempts to measure the mass of a hawk standardized by their wing span. The "bird BMI" is equal to the weight of the hawk (in grams) divided by their wing span (in millimeters) squared. That is,

$$\text{Bird-BMI} := 1000 \times \text{Weight}/\text{Wing-pan}^2.$$

**(Q1)**. Use the **mutate()**, **select()** and **arrange()** functions to create a new data frame called "hawksWithBMI" which has the same number of rows as the original Hawks data frame but only two columns - one with their Species and one with their "bird BMI". Also, arrange the rows in descending order of "bird BMI". The top 8 rows of your data frame should look something like this:

```
##   Species  bird_BMI
## 1      RT 852.69973
## 2      RT 108.75741
## 3      RT  32.57493
## 4      RT  22.72688
## 5      CH  22.40818
## 6      RT  19.54932
## 7      CH  15.21998
## 8      RT  14.85927
```

## 1.5 Summarize and group-by functions

Using the data frame "hawksFullName", from problem 3 above, to do the following tasks:

**(Q1)**. In combination with the summarize() and the group_by functions, create a summary table, broken down by Hawk species, which contains the following summary quantities:

1. The number of rows (num_rows);
2. The average wing span in centimeters (mn_wing);
3. The median wing span in centimeters (nd_wing);
4. The trimmed average wing span in centimeters with trim=0.1, i.e., the mean of the numbers after the 10% largest and the 10% smallest values being removed (t_mn_wing);
5. The biggest ratio between wing span and tail length (b_wt_ratio).

Type `?summarize` to see a list of useful functions (mean, sum, etc) that can be used to compute the summary quantities. Your final result should look something like this:

```
## # A tibble: 3 x 6
##   Species       num_rows mn_wing md_wing t_mn_wing b_wt_ratio
##   <chr>            <int>   <dbl>   <dbl>     <dbl>      <dbl>
## 1 Cooper's            70     NA      240        NA       1.67
## 2 Red-tailed         577    383.     384       385.      3.16
## 3 Sharp-shinned      261    185.     191       184.      1.67
```

**(Q2)**. Next create a summary table of the following form: Your summary table will show the number of missing values, broken down by species, for the columns `Wing, Weight, Culmen, Hallux, Tail, StandardTail, Tarsus, and Crop`. You can complete this task by combining the **select()**, **group_by()**, **summarize()**, **across()**, **everything()**, **sum()** and **is.na()** functions. You should end with a summary table of the following form:

```
## # A tibble: 3 x 9
##   Species        Wing Weight Culmen Hallux  Tail StandardTail Tarsus  Crop
##   <chr>         <int>  <int>  <int>  <int> <int>        <int>  <int> <int>
## 1 Cooper's          1      0      0      0     0           19     62    21
```

[1] Both the "Healthy Hawks Society" and the concept of "bird BMI" were made up purely for this assignment.

```
## 2 Red-tailed        0      5      4      3      0          250    538   254
## 3 Sharp-shinned     0      5      3      3      0           68    233    68
```

# 2. Tidy data and iteration

Tidy data and iteration has been introduced in Lecture 5.

## 2.1. Missing data and iteration

In this task we investigate the effect of missing data and writing iterations in R.

**(Q1)** The following function performs imputation by mean. What library do we need to load to run this function?

```
impute_by_mean<-function(x){
  mu<-mean(x,na.rm=1) # first compute the mean of x
  impute_f<-function(z){ # coordinate-wise imputation
    if(is.na(z)){
      return(mu) # if z is na replace with mean
      }else{
        return(z) # otherwise leave in place
      }
    }
return(map_dbl(x,impute_f)) # apply the map function to impute across vector
}
```

**(Q2)** Create a function called `impute_by_median` which imputes missing values based on the median of the sample, rather than the mean.

You can test your function on the following sample vector:

```
v<-c(1,2,NA,4)
impute_by_median(v)
```

```
## [1] 1 2 2 4
```

**(Q3)** Next generate a data frame with two variables $x$ and $y$. For our first variable $x$ we have a sequence $(x1, x2, \cdots, x_n)$ where $x_1 = 0$, $x_n = 10$ and for each $i = 1, \cdots, n-1$, $x_{i+1} = x_i + 0.1$. For our second variable $y$ we set $y_i = 5x_i + 1$ for $i = 1, \cdots, n$. Generate data of this form and place within a data frame called `df_xy`.

```
df_xy %>% head(5)
```

```
##     x   y
## 1 0.0 1.0
## 2 0.1 1.5
## 3 0.2 2.0
## 4 0.3 2.5
## 5 0.4 3.0
```

The `map2()` function is similar to the `map()` function but iterates over two variables in parallel rather than one. You can learn more here https://purrr.tidyverse.org/reference/map2.html. The following simple example shows you how `map2_dbl()` can be combined with the `mutate()` function.

```
##     x   y   z
## 1 0.0 1.0 1.0
## 2 0.1 1.5 1.6
## 3 0.2 2.0 2.2
## 4 0.3 2.5 2.8
```

```
## 5 0.4 3.0 3.4
```

**(Q4)** We will now use **map2_dbl()** to generate a new data frame with missing data.

First create a function `sometimes_missing` with two arguments: `index` and `value`. The `function` should return `NA` if index is divisible by 5 and returns value otherwise.

Your function should produce the following outputs:

```
sometimes_missing(14,25)
```

```
## [1] 25
```

```
sometimes_missing(15,25)
```

```
## [1] NA
```

Next generate a new data frame called `df_xy_missing` with two variables $x$ and $y$, but some missing data. For the first variable x we have a sequence $(x_1, \cdots, x_n)$, which is precisely the same as with `df_xy`. For the second variable y we have a sequence $(y_1, \cdots, y_n)$ where $yi$ = NA if $i$ is divisible by 5 and otherwise $y_i = 5x_i + 1$. To generate the data frame `d_xy_missing` you may want to make use of the functions `row_number(), map2_dbl(), mutate()` as well as `sometimes_missing()`.

Check that the first ten rows of your data frame are as follows:

```
df_xy_missing %>% head(10)
```

```
##      x   y
## 1  0.0 1.0
## 2  0.1 1.5
## 3  0.2 2.0
## 4  0.3 2.5
## 5  0.4  NA
## 6  0.5 3.5
## 7  0.6 4.0
## 8  0.7 4.5
## 9  0.8 5.0
## 10 0.9  NA
```

**(Q5)** Create a new data frame `df_xy_imputed` with two variables $x$ and $y$. For the first variable $x$ we have a sequence $(x_1, \cdots, x_n)$, which is precisely the same as with `df_xy`. For the second variable $y$ we have a sequence $(y'_1, \cdots, y'_n)$ which is formed from $(y_1, \cdots, y_n)$ by imputing any missing values with the median. To generate `df_xy_imputed` from `df_xy_missing` by applying a combination of the functions `mutate()` and `impute_by_median()`.

The first part of the data frame should look like this:

```
##     x    y
## 1 0.0  1.0
## 2 0.1  1.5
## 3 0.2  2.0
## 4 0.3  2.5
## 5 0.4 26.0
## 6 0.5  3.5
```

## 2.2 Tidying data with pivot functions

In this task you will read in data from a spreadsheet and apply some data wrangling tasks to tidy that data.

First download the excel spreadsheet entitled `"HockeyLeague.xlsx"`. The excel file contains two spread-sheets - one with the wins for each team and one with the losses for each team. To read this spreadsheet into R we

shall make use of the `readxl` library. You may need to install the library:

install.packages("readxl")

The following code shows how to read in a sheet within an excel file as a data frame. You will need to edit the `folder_path` variable to be the directory which contains your copy of the spreadsheet

```
library(readxl) # load the readxl library
folder_path<-"C:/Users/" # set this to the name of the
# directory containing "HockeyLeague.xlsx"
file_name<-"HockeyLeague.xlsx" # set the file name
file_path<-paste(folder_path,file_name,sep="") # create the file_path
wins_data_frame<-read_excel(file_path,sheet="Wins") # read of a sheet from an xl file
```

Inspect the first 3 rows of the first five columns:

```
wins_data_frame %>%
  select(1:5)%>%
  head(3)
```

```
## # A tibble: 3 x 5
##   ...1   `1990`   `1991`   `1992`   `1993`
##   <chr>  <chr>    <chr>    <chr>    <chr>
## 1 Ducks  30 of 50 11 of 50 30 of 50 12 of 50
## 2 Eagles 24 of 50 12 of 50 37 of 50 14 of 50
## 3 Hawks  20 of 50 22 of 50 33 of 50 11 of 50
```

A cell value of the form "a of b" means that games were won out of a total of b for that season. For example, the element for the "Ducks" row of the "1990" column is "30 of 50" meaning that 30 out of 50 games were won that season.

Is this tidy data?

**(Q1)** Now apply your data wrangling skills to transform the `"wins_data_frame"` data frame object into a data frame called `"wins_tidy"` which contains the same information but has just four columns entitled `"Team"`, `"Year"`, `"Wins"`, `"Total"`. The `"Team"` column should contain the team name, the `"Year"` column should contain the year, the `"Wins"` column should contain the number of wins for that season and the `"Total"` column the total number of games for that season. The first column should be of character type and the remaining columns should be of integer type. You can do this by combining the following functions: `rename()`, `pivot_longer()`, `mutate()` and `separate()`.

You can check the shape of your data frame and the first five rows as follows:

```
wins_tidy %>% dim() # check the dimensions
```

```
## [1] 248   4
```

```
wins_tidy%>%head(5) # inspect the top 5 rows
```

```
## # A tibble: 5 x 4
##   Team   Year  Wins Total
##   <chr> <int> <int> <int>
## 1 Ducks  1990    30    50
## 2 Ducks  1991    11    50
## 3 Ducks  1992    30    50
## 4 Ducks  1993    12    50
## 5 Ducks  1994    24    50
```

**(Q2)** The `"HockeyLeague.xlsx"` also contains a sheet with the losses for each team by season. Apply a similar procedure to read the data from this sheet and transform that data into a data frame called

"losses_tidy" with four columns: "Team", "Year", "Losses", "Total" which are similar to those in the "wins_tidy" data frame except for the "Losses" column gives the number of losses for a given season and team, rather than the number of losses.

Your results should look like this:

```
losses_tidy %>% head(5)
```

```
## # A tibble: 5 x 4
##   Team   Year Losses Total
##   <chr> <int>  <int> <int>
## 1 Ducks  1990     20    50
## 2 Ducks  1991     37    50
## 3 Ducks  1992      1    50
## 4 Ducks  1993     30    50
## 5 Ducks  1994      7    50
```

You may notice that the number of wins plus the number of losses for a given team, in a given year does not add up to the total. This is because some of the games are neither wins nor losses but draws. That is, for a given year the number of draws is equal to the total number of games minus the sum of the wins and losses.

**(Q3)** Now combine your two data frames, "wins_tidy" and "losses_tidy", into a single data frame entitled "hockey_df" which has 248 rows and 9 columns: A "Team" column which gives the name of the team as a character, the "Year" column which gives the season year, the "Wins" column which gives the number of wins for that team in the given year, the "Losses" column which gives the number of losses for that team in the given year and the "Draws" column which gives the number of draws for that team in the given year, the "Wins_rt" which gives the wins as a proportion of the total number of games (ie. Wins/Total) and similarly the "Losses_rt" and the "Draws_rt" which gives the losses and draws as a proportion of the total, respectively. To do this you can make use of the mutate() function. You may also want to utilise the across() function for a slightly neater solution.

The top five rows of your data frame should look as follows:

```
hockey_df %>% head(5)
```

```
## # A tibble: 5 x 9
##   Team   Year  Wins Total Losses Draws Wins_rt Losses_rt Draws_rt
##   <chr> <int> <int> <int>  <int> <int>   <dbl>     <dbl>    <dbl>
## 1 Ducks  1990    30    50     20     0    0.6       0.4      0
## 2 Ducks  1991    11    50     37     2    0.22      0.74     0.04
## 3 Ducks  1992    30    50      1    19    0.6       0.02     0.38
## 4 Ducks  1993    12    50     30     8    0.24      0.6      0.16
## 5 Ducks  1994    24    50      7    19    0.48      0.14     0.38
```

**(Q4)** To conclude this task generate a summary data frame which displays, for each team, the median win rate, the mean win rate, the median loss rate, the mean loss rate, the median draw rate and the mean draw rate. The number of rows in your summary should equal the number of teams. These should be sorted in descending order or median win rate. You may want to make use of the following functions: select(), group_by(), across(), arrange().

```
## # A tibble: 8 x 7
##   Team         W_md  W_mn  L_md  L_mn  D_md  D_mn
##   <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Eagles      0.45  0.437 0.25  0.279 0.317 0.284
## 2 Penguins    0.45  0.457 0.3   0.310 0.133 0.232
## 3 Hawks       0.417 0.388 0.233 0.246 0.32  0.366
## 4 Ducks       0.383 0.362 0.34  0.333 0.25  0.305
## 5 Owls        0.32  0.333 0.3   0.33  0.383 0.337
```

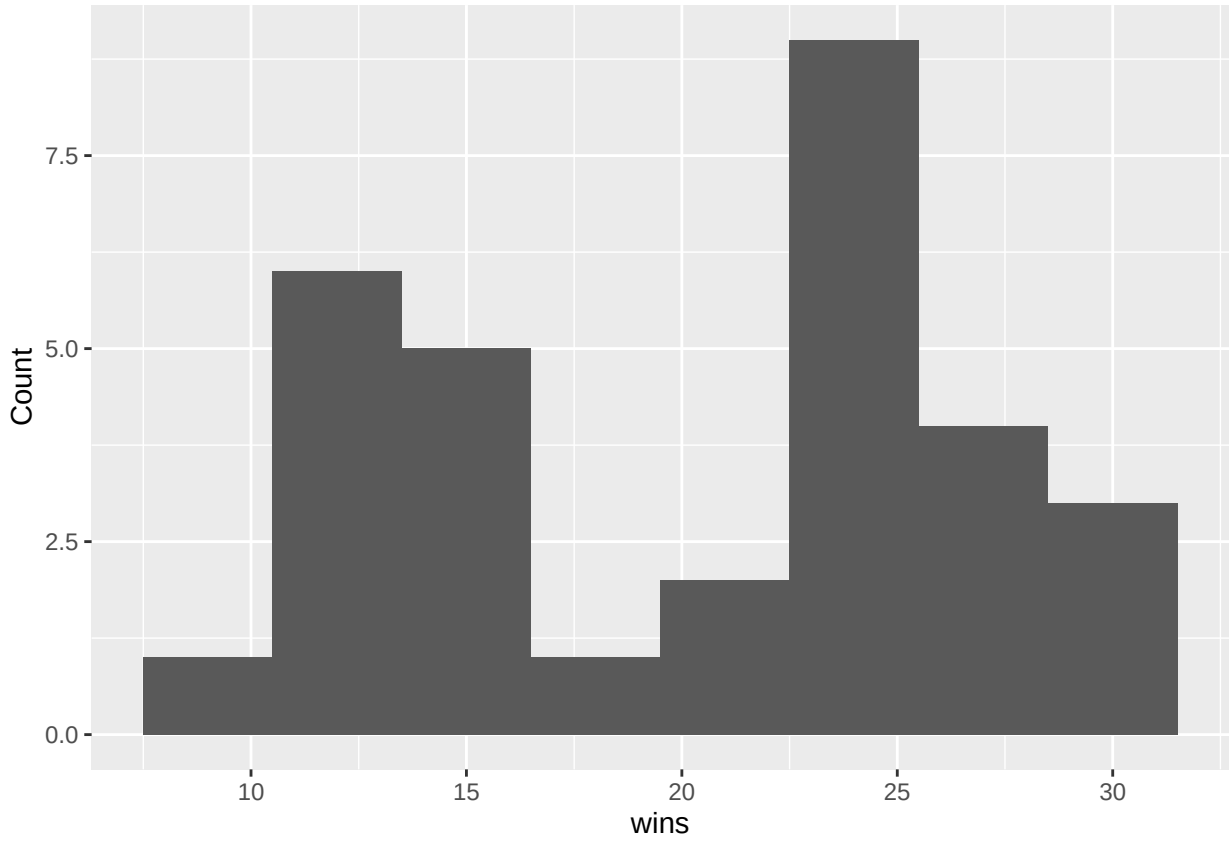
```
## 6 Ostriches   0.3   0.309 0.4   0.395 0.267 0.296
## 7 Storks      0.3   0.284 0.22  0.283 0.48  0.433
## 8 Kingfishers 0.233 0.245 0.34  0.360 0.4   0.395
```

# 3. Visualisation

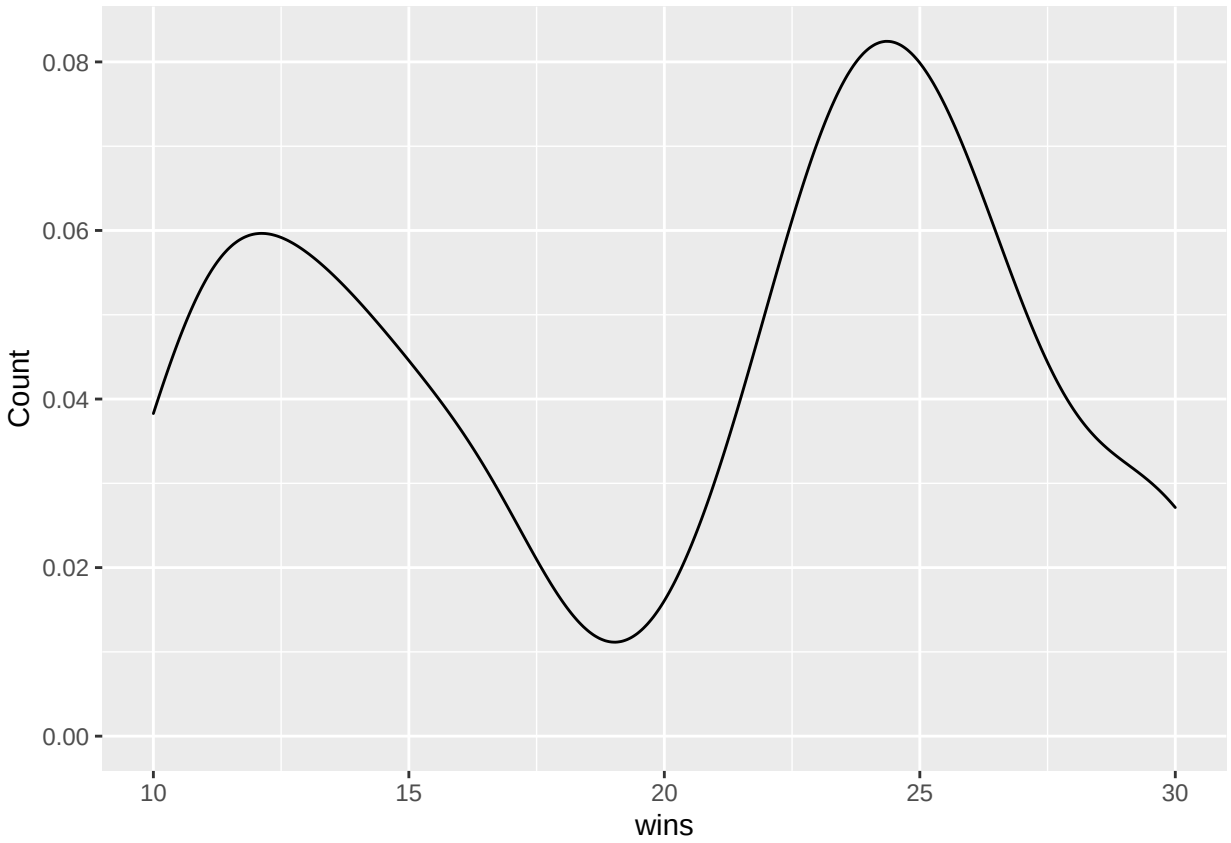
This part is mainly about visualisation using ggplot2. It covers a part of Lecture 6.

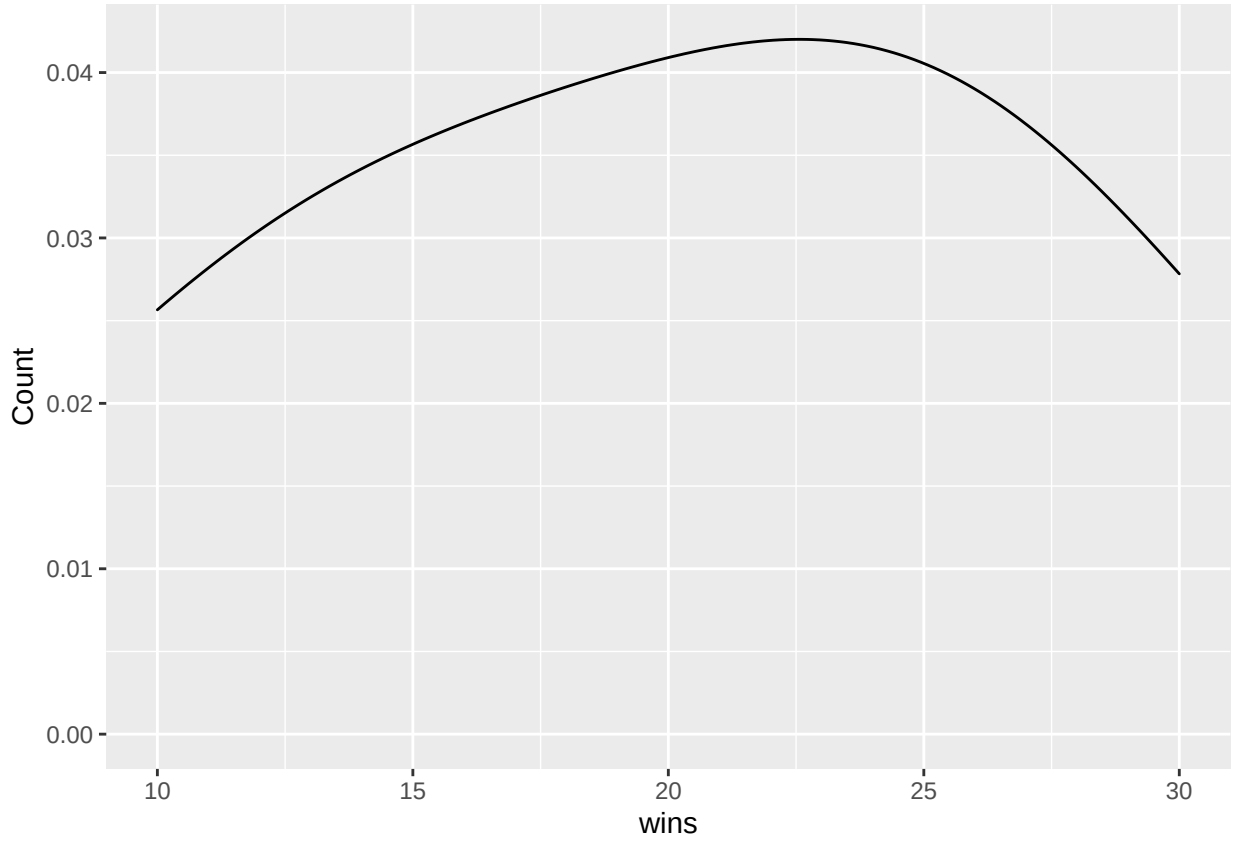
We will reuse the data `wins_tidy` obtained above to do visualisation.

**(Q1)** Use a combination of the functions `filter()`, `ggplot()` and `geom_histogram` to create a histogram plot of the `Wins` of `Ducks` within data frame `wins_tidy` with bin widths of 3. Your result should look something like this:




**(Q2)** Similar to (Q1), use the `geom_density()` function to create two density plots, with parameters `adjust=0.5` and `adjust=2`, respectively. Your results should look like this:


0.08
0.06
0.04
0.02
0.00
Count
10
15
20
25
30
wins




Can you explain the difference between the two plots?

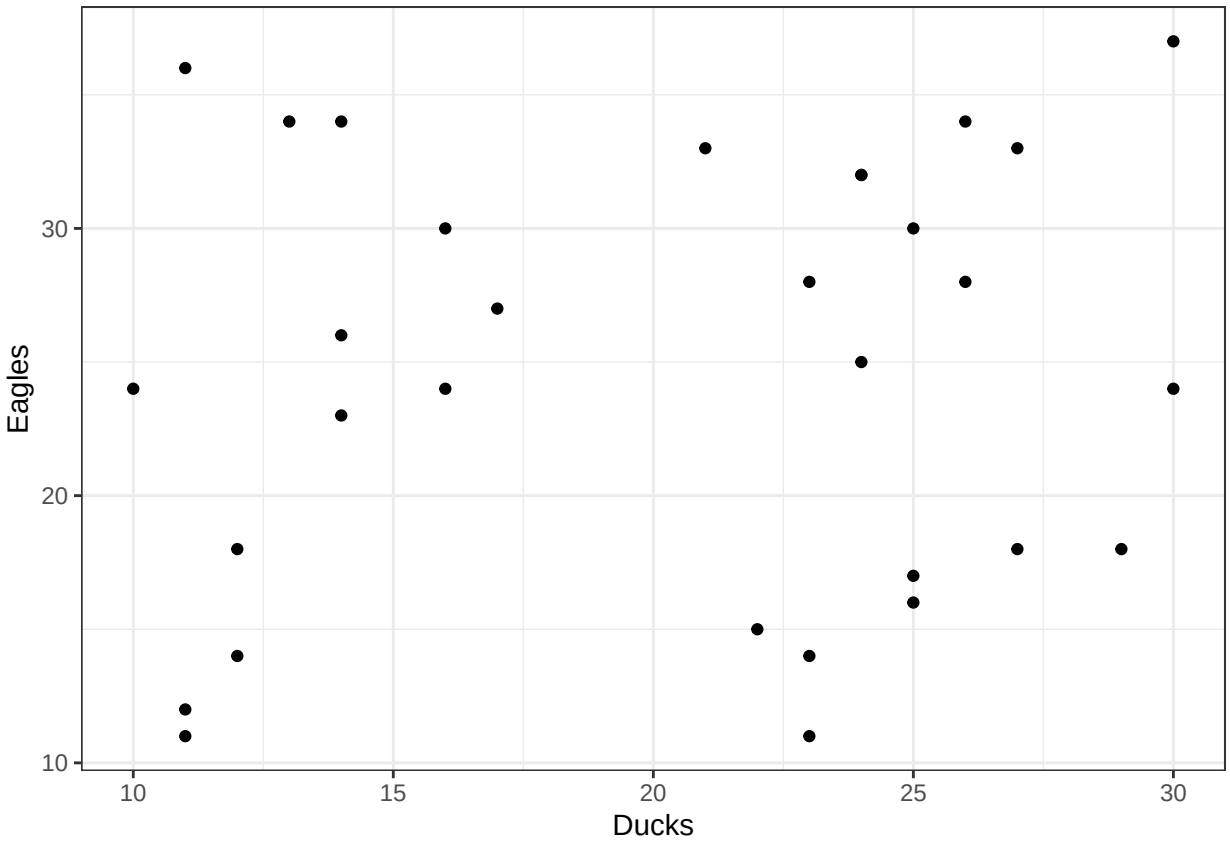
**(Q3)** Next, let's create a bivariate plot. First, from `wins_tidy`, create a data frame called `wins_teams` with columns: `Year`, `Ducks`, `Eagles`, as well as the other teams. For example, the column `Ducks` represent the `Wins` of the team `Ducks` for different years. You can use a combination of `select` and `pivot_wider` to do so. The data frame should look like:

```
wins_team
```

```
## # A tibble: 31 x 9
##     Year Ducks Eagles Hawks Kingfishers Ostriches  Owls Penguins Storks
##    <int> <int>  <int> <int>       <int>     <int> <int>    <int>  <int>
##  1  1990    30     24    20          16        13    19       23     20
##  2  1991    11     12    22          19        13    13       29     13
##  3  1992    30     37    33          12        10    18       30     18
##  4  1993    12     14    11          10        25    16       32     22
##  5  1994    24     32    20          17        10    13       33     19
##  6  1995    13     34    18          11        21    24       36     11
##  7  1996    25     17    21          11        13    24       12     15
##  8  1997    24     25    23          12        18    10       16     15
##  9  1998    27     33    18          11        24    20       34     14
## 10  1999    23     28    28          19        18    21       20     13
## # ... with 21 more rows
```

Then from `wins_team`, use `geom_point()` to create a scatter plot, the x-axis is the Wins of `Ducks`, and the y-axis is the `Wins` of `Eagles`.

Check if you figure is similar to:


Eagles
30
20
10
10
15
20
25
30
Ducks