Precept 4 Answers

Your Name

2019-10-23

Objectives

- `ggplot2` functions
- `ggpairs()` in the `GGally` package

Demo

We will use the dataset `ChickWeight` for the first part of the demo.

Check the data type for each column of the dataset with str()

`strict.width` restricts the width of the output of `str`

```
# We are going to use the `ChickWeight` dataset Let's take a
# look at help manual of the dataset
`?`(ChickWeight)

str(ChickWeight, strict.width = "cut")
Classes 'nfnGroupedData', 'nfGroupedData', 'groupedData' and 'data.frame':  578 obs. of  4 variab
 $ weight: num  42 51 59 64 76 93 106 125 149 171 ...
 $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...
 $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15 15..
 $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, "formula")=Class 'formula'  language weight ~ Time | Chick
  .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
 - attr(*, "outer")=Class 'formula'  language ~Diet
  .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
 - attr(*, "labels")=List of 2
  ..$ x: chr "Time"
  ..$ y: chr "Body weight"
 - attr(*, "units")=List of 2
  ..$ x: chr "(days)"
  ..$ y: chr "(gm)"
```

ggplot

You will need the packages `ggplot2` and `GGally` for this precept. Make sure that you have installed and loaded the packages.

In previous weeks we learned graphic functions in R's basic `graphics` package. For this precept we will use `ggplot2` which gives us more advanced options in plotting. Creating a plot using this package always follows the following steps:

1. Dataset - use the function `ggplot()` to specify what dataset we will work with to create the plot; the dataset needs to be a data frame.

2. Type of graph to make - use `geom_FunctionTpye()` to specify what kinds of graphs (such as scatterplots, histograms, or boxplots) we want to make.

3. Aesthetic mappings - specify the variables to use for the plot(s) and how they appear on the plot.

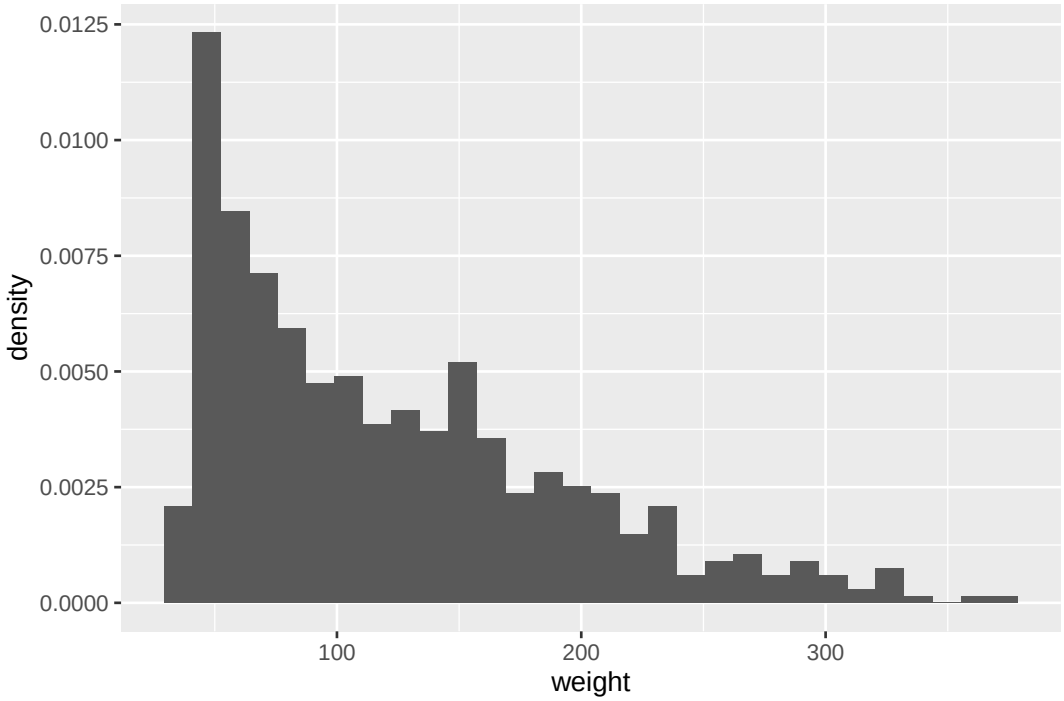
4. Appearance - using the various parameters to change the display of the plot, such as specifying the labels, colors or showing the data by groups.

Let's first get us familiar with the dataset used for the demo.

```
class(ChickWeight) # It's a data frame
[1] "nfnGroupedData" "nfGroupedData"  "groupedData"    "data.frame"
dim(ChickWeight)
[1] 578   4

library(ggplot2)
```

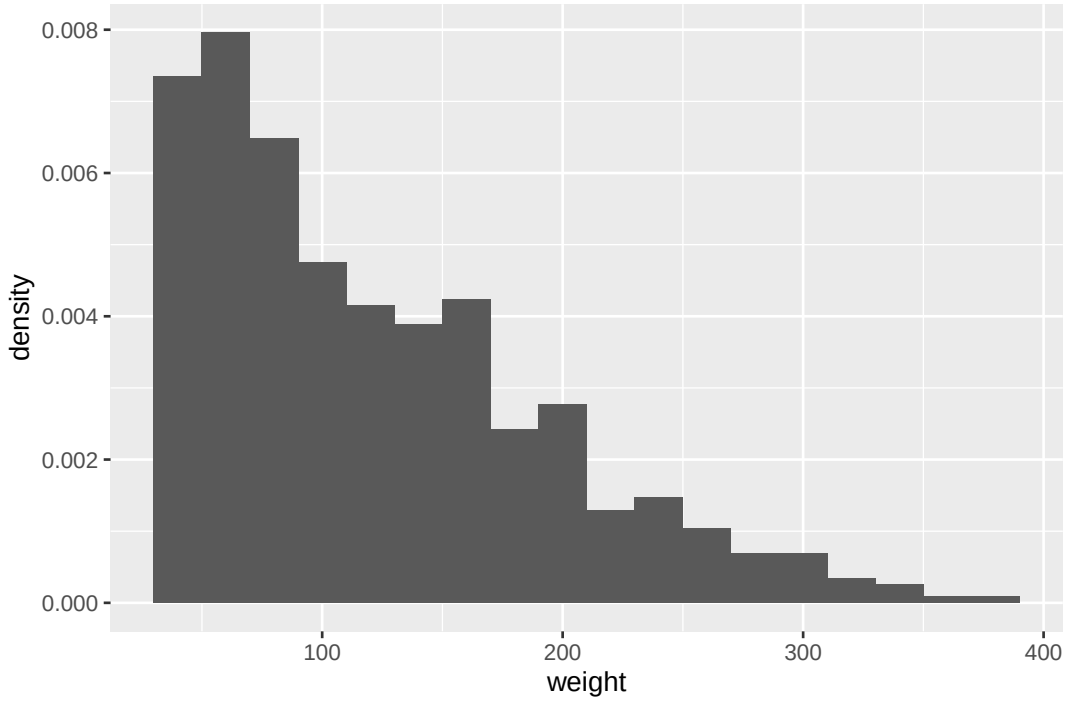
Let's make a histogram for the chicken's weights.

```
# First step - defining the dataset to be used;
# create a ggplot object and we name it `chickPlot`
chickPlot = ggplot(data=ChickWeight)

# at this point R knows we want to use the `ChickWeight` dataset
# Second step - use histogram plot
# we use `geom_histogram(..., binwidth = ...)` to create a histogram with certain bin width
# `y = ..density..` allows us to use "density" for the y-axis, not frequency
chickPlot + geom_histogram(mapping = aes(x=weight, y=..density..))
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

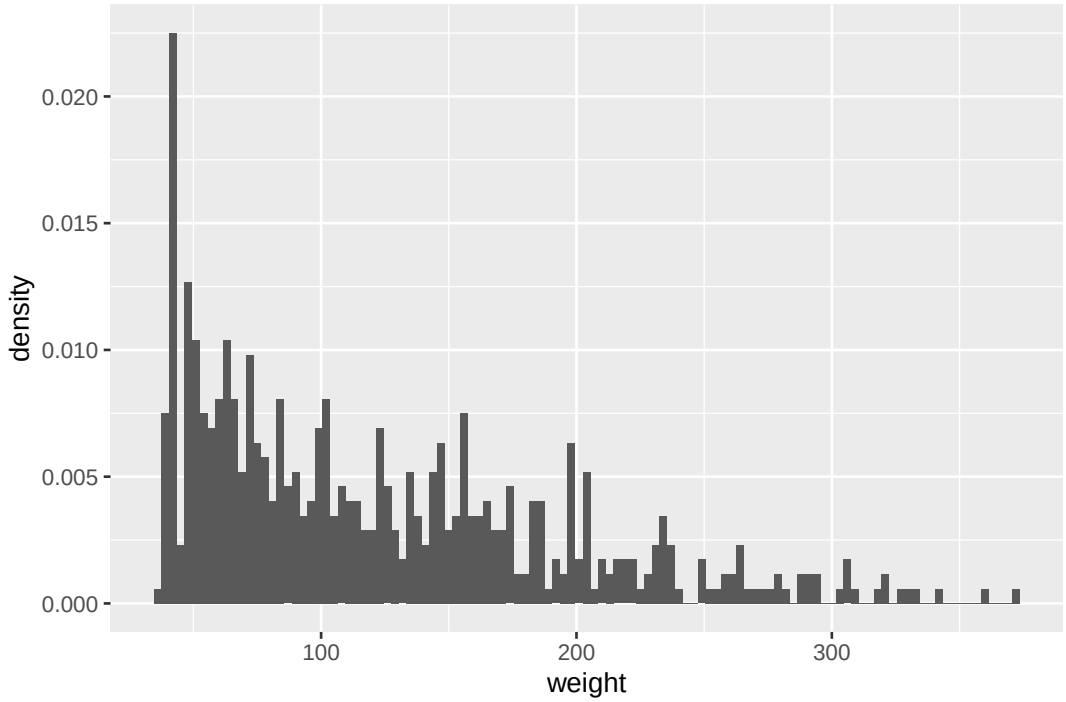


We can see from the histogram above that the bin width is 10. Let's try a few more options for the bin width before deciding on one.

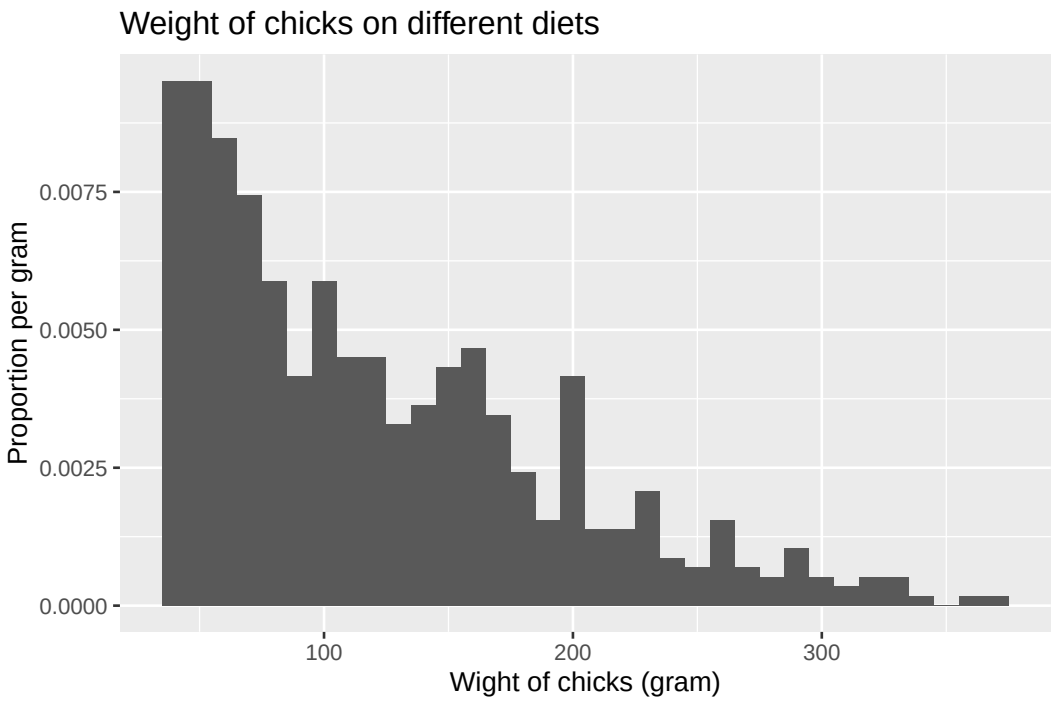
```
chickPlot + geom_histogram(mapping = aes(x = weight, y = ..density..),
    binwidth = 20)  #too wide
```




```
chickPlot + geom_histogram(mapping = aes(x = weight, y = ..density..),
    binwidth = 3)  #too narrow
```



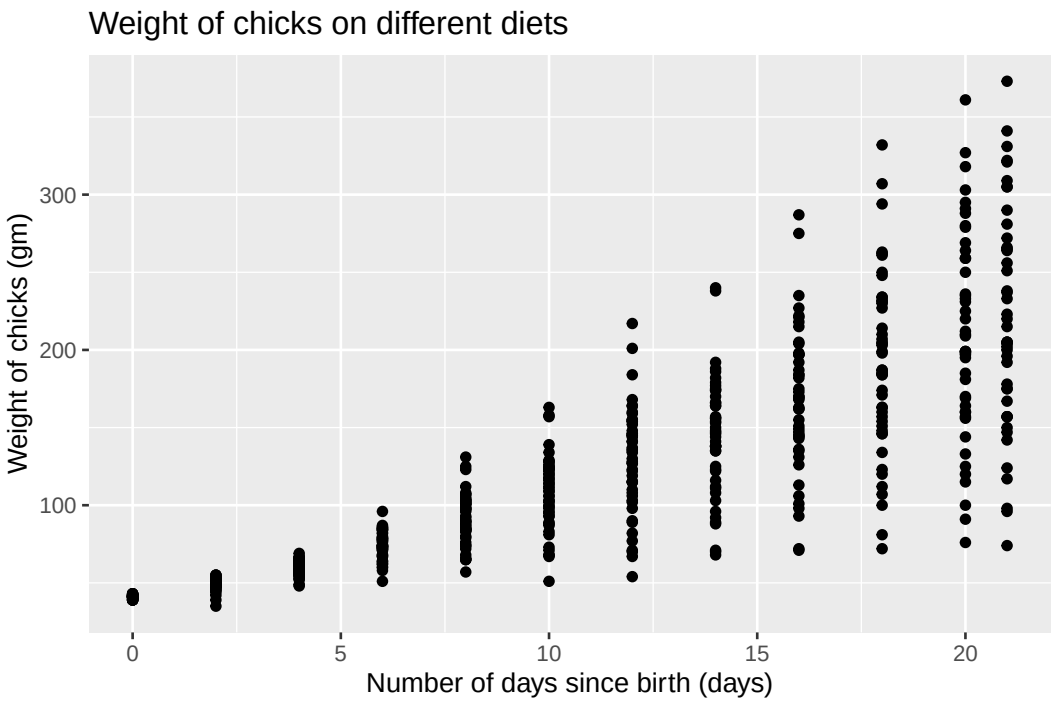
```
# choose the appropriate bin width
chickPlot + geom_histogram(aes(x = weight, y = ..density..), binwidth = 10) +
    labs(x = "Wight of chicks (gram)", y = "Proportion per gram",
        title = "Weight of chicks on different diets")
```



`binwidth =7` seems to be the best choice among the ones that we have tried since the graph shows enough details of the distribution without over-complicating the graph and making it harder to read.

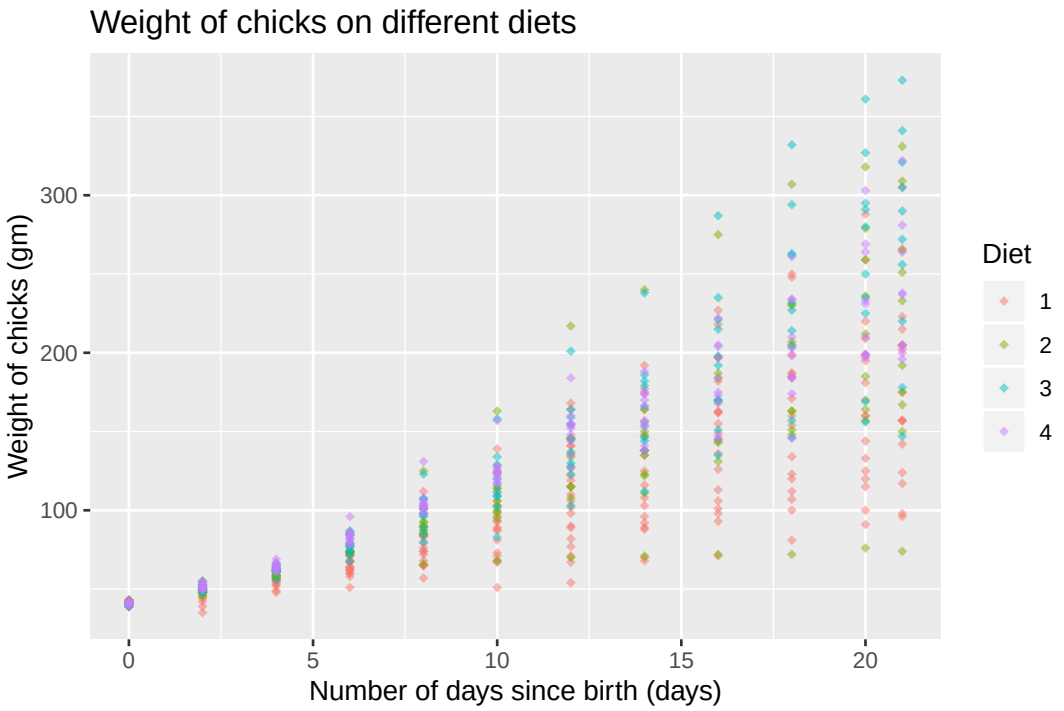
We now make a scatterplot for the weights of the chicken v.s. their ages.

```
# We add labels. All the points will have the same color
chickPlot +
  geom_point(mapping = aes(x=Time, y=weight)) +
  labs(x="Number of days since birth (days)",
       y="Weight of chicks (gm)",
       title = "Weight of chicks on different diets")
```



Let's give different colors to the points according to the chicken's diet.

```
# Add color to the plot by `Diet`
ggplot(ChickWeight) + geom_point(mapping = aes(x = Time, y = weight,
    colour = Diet), shape = 18, alpha = 0.5) + labs(x = "Number of days since birth (days)",
    y = "Weight of chicks (gm)", title = "Weight of chicks on different diets")
```



You can see a list of the available shapes for the `geom_point()` function here (http://sape.inf.usi.ch/quick-reference/ggplot2/shape).

The examples here make histograms and scatterplots, but `ggplot2` has other additional functions for making different types of plots; these functions including:

- geom_boxplot()
- geom_histogram()
- geom_line()
- geom_point()
- geom_smooth()
- geom_hex()

The syntax for these functions following similar reasonings as `geom_histogram()` and `geom_point` do.

`ggpairs()` in `GGally` package

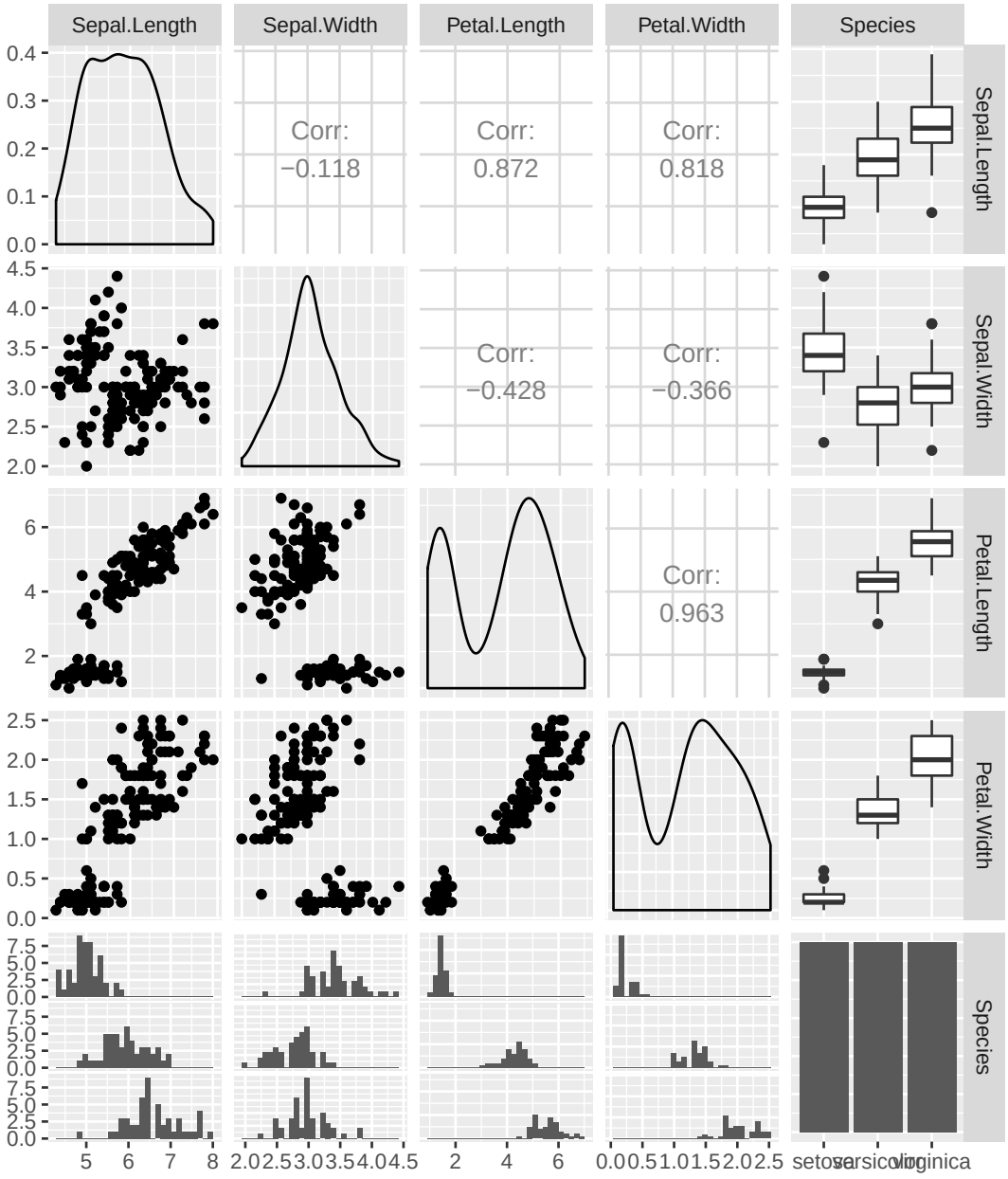
`ggpairs()` in the `GGally` package makes plots for each pair of variables in the input data frame; such matrix plot is very useful when we want to investigate the pairwise relationship of the variables in our dataset. Since `GGally` is an extension of the `ggplot2` package, the syntax for `ggpairs()` is somewhat similar to that for functions in `ggplot2`.

Note that you will need to load both the `ggpairs()` *and* `GGally` *packages in order to use functions in the* `GGally` *package.*

We will use the `iris` dataset for this part of the demo. Let's get ourselves familiar with the dataset first.

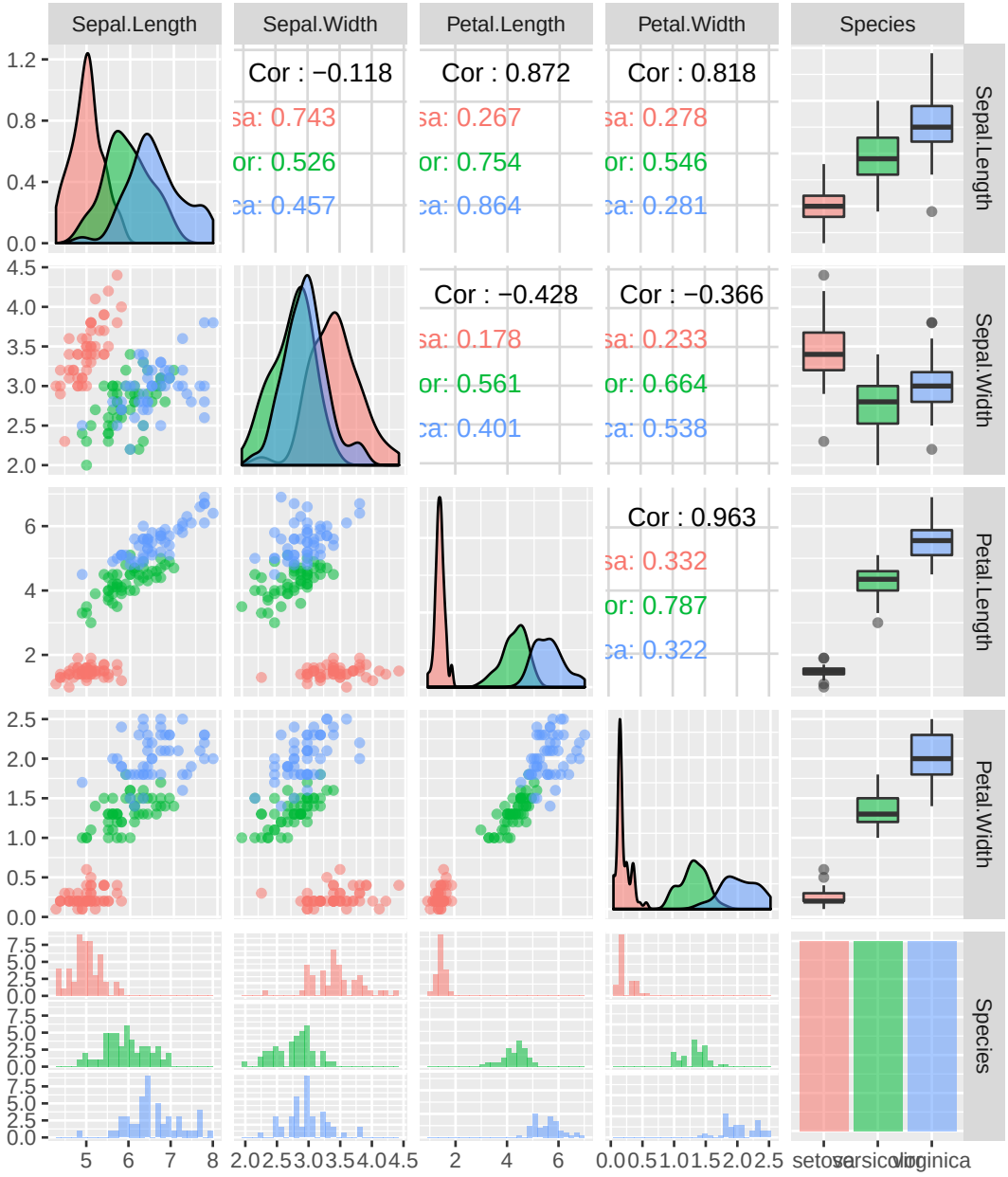
```
`?`(iris)
str(iris, strict.width = "cut")
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1..
class(iris)
[1] "data.frame"
dim(iris)
[1] 150   5
head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

```
library(GGally)
Registered S3 method overwritten by 'GGally':
  method from
  +.gg   ggplot2

ggpairs(iris)
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

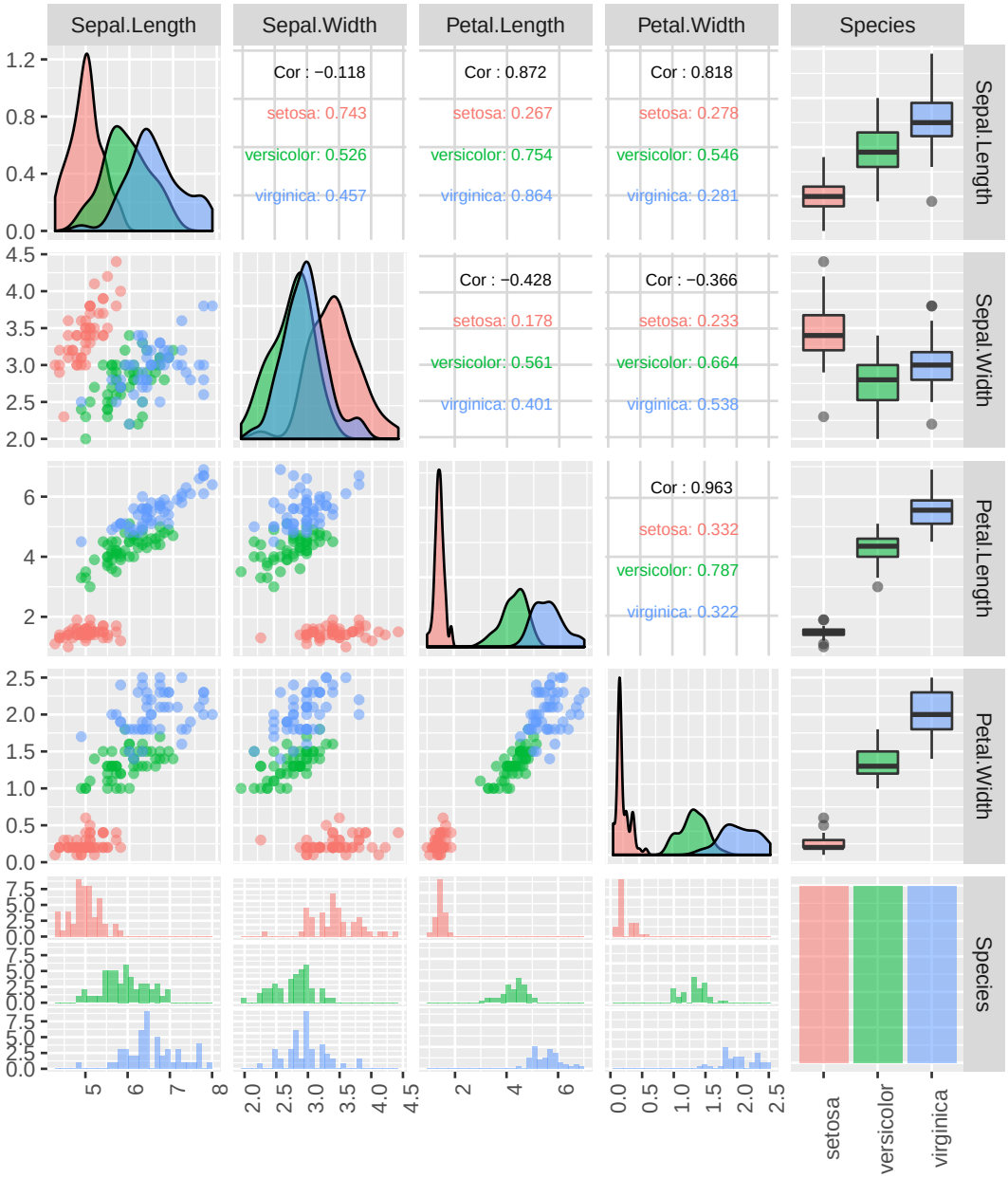



```
# We have boxplots for other_variables v.s. Species
# because `Species` is a factor;
# also note that on the bottom row of the matrix plot
# in each of the first 4 graphs from the left
# there are three histograms rather than one;
# this is also because `Species` is a factor;
```

```
ggpairs(iris, aes(colour = Species, alpha = 0.4))
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
# This shows the distributions by Species and
# calculate the correlations between variables
# by Species

ggpairs(iris, aes(colour = Species, alpha = 0.4),
        upper = list(continuous = wrap("cor", size = 2.5, alignPercent = .8))) +
  # upper: are the variables associated with the upper-triangular
  # of the plot both continues (continues), both discrete (discrete)
  # or a combination of continuous and discrete (combo)?
  # size:   text absolute size in pts; can also use rel(2) for relative size
  # alignPercent:  right align position of numbers for the correlation
# figures. Default is .6  which stands for 60 percent across the horizontal.

  theme(axis.text.x = element_text(angle = 90))
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# rotate x-axis annotations for 90 degree counter-clockwise;
```

Preceptor: Please also remind students what `fig.height` and `fig.width` do.

Working through a question

(Part a is optional; the purpose of this part is to show you the steps for solving a problem correctly. If you are confident about the steps, you can skip part

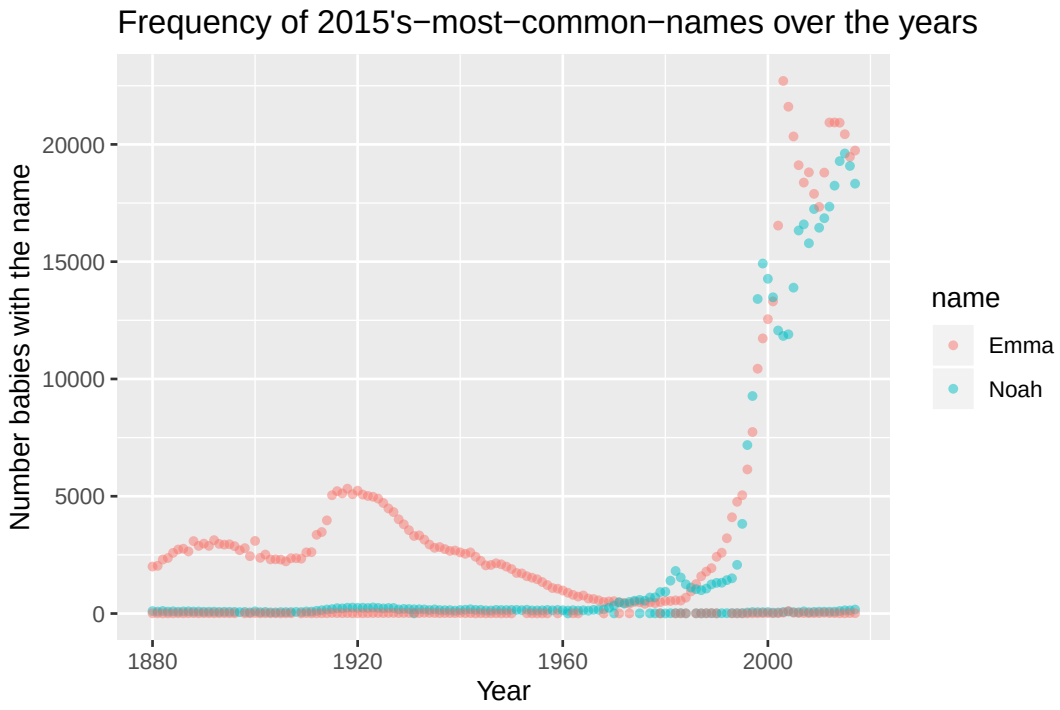
a; otherwise, please go through part a after you finish all the exercises in this precept.)

(a) Use the dataset `babynames`. Find the boy name and girl name that were used most often in 2015. For each of these most popular names in 2015, find out its frequency of use throughout the years in the period 1880 to 2017.

(b) Use `ggplot` to plot the frequency of use throughout the years that you obtain in part a. Provide meaningful labels and use a different color for each name.

```
library(babynames)
head(babynames)
# A tibble: 6 x 5
   year sex   name          n   prop
  <dbl> <chr> <chr>     <int>  <dbl>
1  1880 F     Mary       7065 0.0724
2  1880 F     Anna       2604 0.0267
3  1880 F     Emma       2003 0.0205
4  1880 F     Elizabeth  1939 0.0199
5  1880 F     Minnie     1746 0.0179
6  1880 F     Margaret   1578 0.0162

#To solve this we will break down the question into steps
#We extract the boy names used in 2015
boys.2015.subset = babynames[babynames$year == 2015 & babynames$sex == "M",]
#We extract the girl names used in 2015
girls.2015.subset = babynames[babynames$year == 2015 & babynames$sex == "F",]

#verify that we have one sex in each the table
unique(boys.2015.subset$sex)
[1] "M"
unique(girls.2015.subset$sex)
[1] "F"

#Sort names by the number of usages for each gender
girls.2015 = girls.2015.subset$n
names(girls.2015) = girls.2015.subset$name

boys.2015 = boys.2015.subset$n
names(boys.2015) = boys.2015.subset$name

girls.2015.sorted = sort(girls.2015, decreasing = T)
boys.2015.sorted = sort(boys.2015, decreasing = T)

#Check the vector that we created for the top female name uses
head(girls.2015.sorted)
    Emma   Olivia   Sophia      Ava Isabella      Mia
   20435    19669    17402    16361    15594    14892
```

```r
#Create a data set for the most frequently used names
frequent.names = babynames[babynames$name == names(boys.2015.sorted)[1] |
                           babynames$name == names(girls.2015.sorted)[1], ]

head(frequent.names)
# A tibble: 6 x 5
   year sex   name      n      prop
  <dbl> <chr> <chr> <int>     <dbl>
1  1880 F     Emma   2003 0.0205
2  1880 M     Noah    103 0.000870
3  1880 M     Emma     10 0.0000845
4  1881 F     Emma   2034 0.0206
5  1881 M     Noah     81 0.000748
6  1881 M     Emma      9 0.0000831
ggplot(frequent.names, aes(x=year,y=n,colour=name))+
         geom_point(shape=16, alpha = 0.5) +
 labs( x="Year",
       y="Number babies with the name",
       title = "Frequency of 2015's-most-common-names over the years")
```



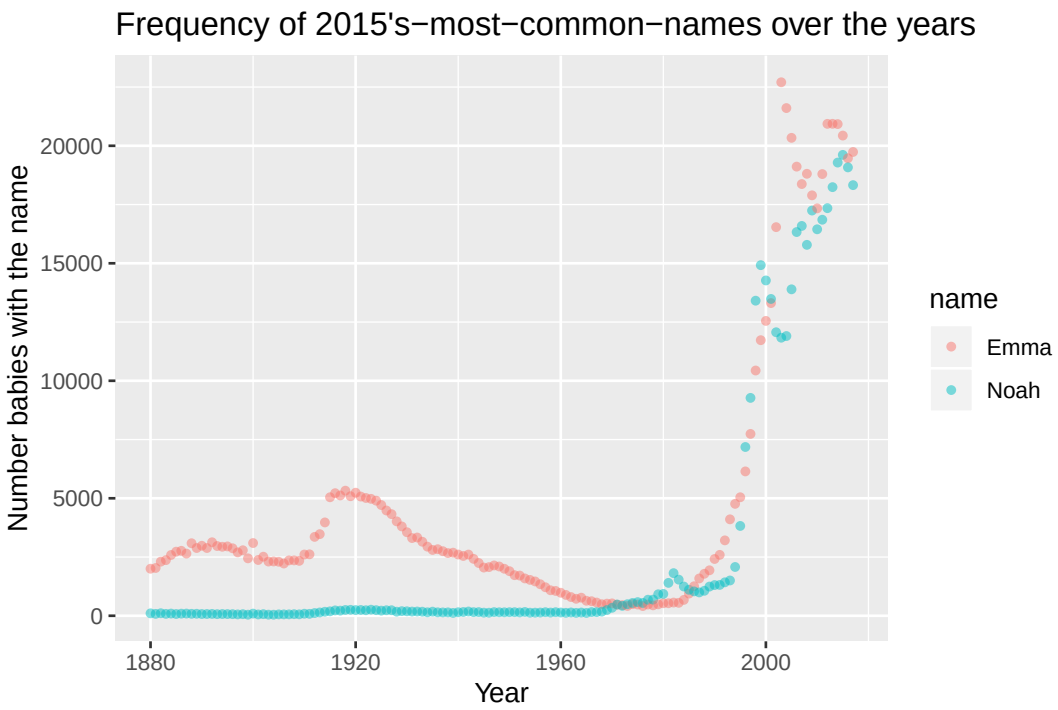
Why are there 2 points for each color per year? How can you check if this is a mistake?

We should have checked the data frame that we created.

```
# Take another look at the data frame that we created
head(frequent.names)
# A tibble: 6 x 5
   year sex   name      n      prop
  <dbl> <chr> <chr> <int>     <dbl>
1  1880 F     Emma   2003 0.0205
2  1880 M     Noah    103 0.000870
3  1880 M     Emma     10 0.0000845
4  1881 F     Emma   2034 0.0206
5  1881 M     Noah     81 0.000748
6  1881 M     Emma      9 0.0000831

dim(frequent.names)
[1] 434   5
frequent.names[frequent.names$year == 1880, ]
# A tibble: 3 x 5
   year sex   name      n      prop
  <dbl> <chr> <chr> <int>     <dbl>
1  1880 F     Emma   2003 0.0205
2  1880 M     Noah    103 0.000870
3  1880 M     Emma     10 0.0000845
# now we realized that we should have extract out the rows that
# correspond to the popular name for each sex.
```

Now we can create the data frame for the plotting in part b.

```
# Create a data set for the most frequently used names
frequent.names = babynames[(babynames$name == names(boys.2015.sorted)[1] &
    babynames$sex == "M") | (babynames$name == names(girls.2015.sorted)[1] &
    babynames$sex == "F"), ]

head(frequent.names)
# A tibble: 6 x 5
   year sex   name      n     prop
  <dbl> <chr> <chr> <int>    <dbl>
1  1880 F     Emma   2003 0.0205
2  1880 M     Noah    103 0.000870
3  1881 F     Emma   2034 0.0206
4  1881 M     Noah     81 0.000748
5  1882 F     Emma   2303 0.0199
6  1882 M     Noah    108 0.000885
ggplot(frequent.names, aes(x = year, y = n, colour = name)) + geom_point(shape = 16,
    alpha = 0.5) + labs(x = "Year", y = "Number babies with the name",
    title = "Frequency of 2015's-most-common-names over the years")
```



How can you create two plots, one for each gender, instead of having just one plot for both genders? Remember what you learned about facet_grid in class.

Saving a plot (optional)

When we use Rmarkdown we generate a full report including text, figures and tables etc. However, in some cases, we might want to save just the plots to a file. For example, we might want to include the plot in another document such as a power point presentation or a poster. In the example below, we will show you how to save the plot to a pdf file. Note that you can also save the graph in other file formats such jpg (by using `jpeg()`) instead of `pdf()`–see the help page for `jpeg()` for other options.

```
getwd()  #check what your working directory is
[1] "/Users/billhaarlow/Desktop/SML201/Precept4"
# We create a pdf file in the working directory
pdf(file = "myIrisPlot.pdf")  #open the device to save pdf
pairs(iris[1:4])  #generate a plot in the pdf
dev.off()  #close the device
pdf
  2
```

After opening the device, each plot will be saved on a new page. Make sure to close the device.

Exercises

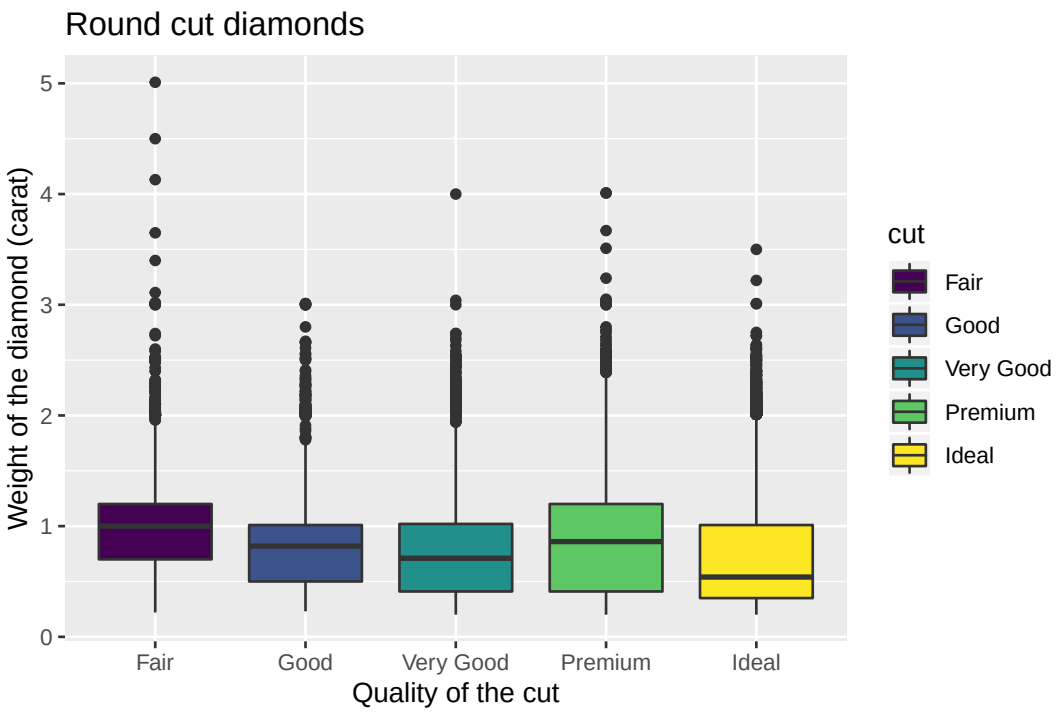
Make sure that your graph has a title and appropriate informative axis labels.

Use "ggplot2" package for the following questions.

1. Load the "ggplot2" package and use the dataset named "diamonds". Make a side by side box plot for "carat" across different categories of "cut". You can use the following labels for your graph:

```
x = "Quality of the cut",
y = "Weight of the diamond (carat)",
title = "Round cut diamonds"
```

```
#We only need to do this once in each Rmarkdown file.
#We add this here for clarity since this is the first time we use ggplot2
#In the answers section
library(ggplot2)

#Let's understand the structure of the `diamonds` dataset first
str(diamonds, strict.width='cut') #restrict the width
Classes 'tbl_df', 'tbl' and 'data.frame':   53940 obs. of  10 variables:
 $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
 $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 ..
 $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 ..
 $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3..
 $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
 $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
 $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
 $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
 $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
 $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...

ggplot(data = diamonds) + #Create a new plot using `diamonds` data
  geom_boxplot(mapping = aes(x = cut, y= carat, fill = cut)) +
  labs(x = "Quality of the cut",
       y = "Weight of the diamond (carat)",
       title = "Round cut diamonds")
```




```
#use boxplot with `cut` on the x
```

2. Use the dataset "diamonds".

a. Subset the diamonds data to include the rows for the diamonds of "Good" and "Fair" cuts only.

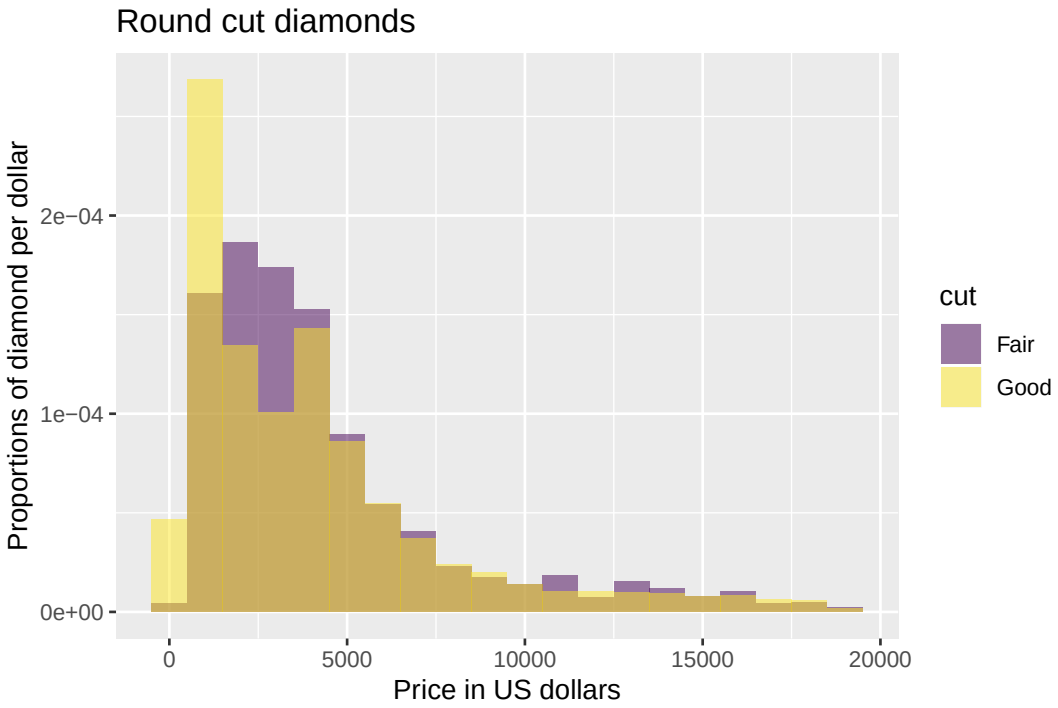
```
my.diamonds = diamonds[diamonds$cut %in% c("Good", "Fair"), ]
```

b. Use the function in the ggplot2 package. Make two histograms, one for the prices for the fair-cut and one for the good-cut diamonds. Allow one histogram to superimpose on the other. Make sure that the bin area represents the proportion of the diamonds (of a certain price range) within a particular cut category. Also, think about how you can show both histograms clearly.

You can use the following labels for your graph:

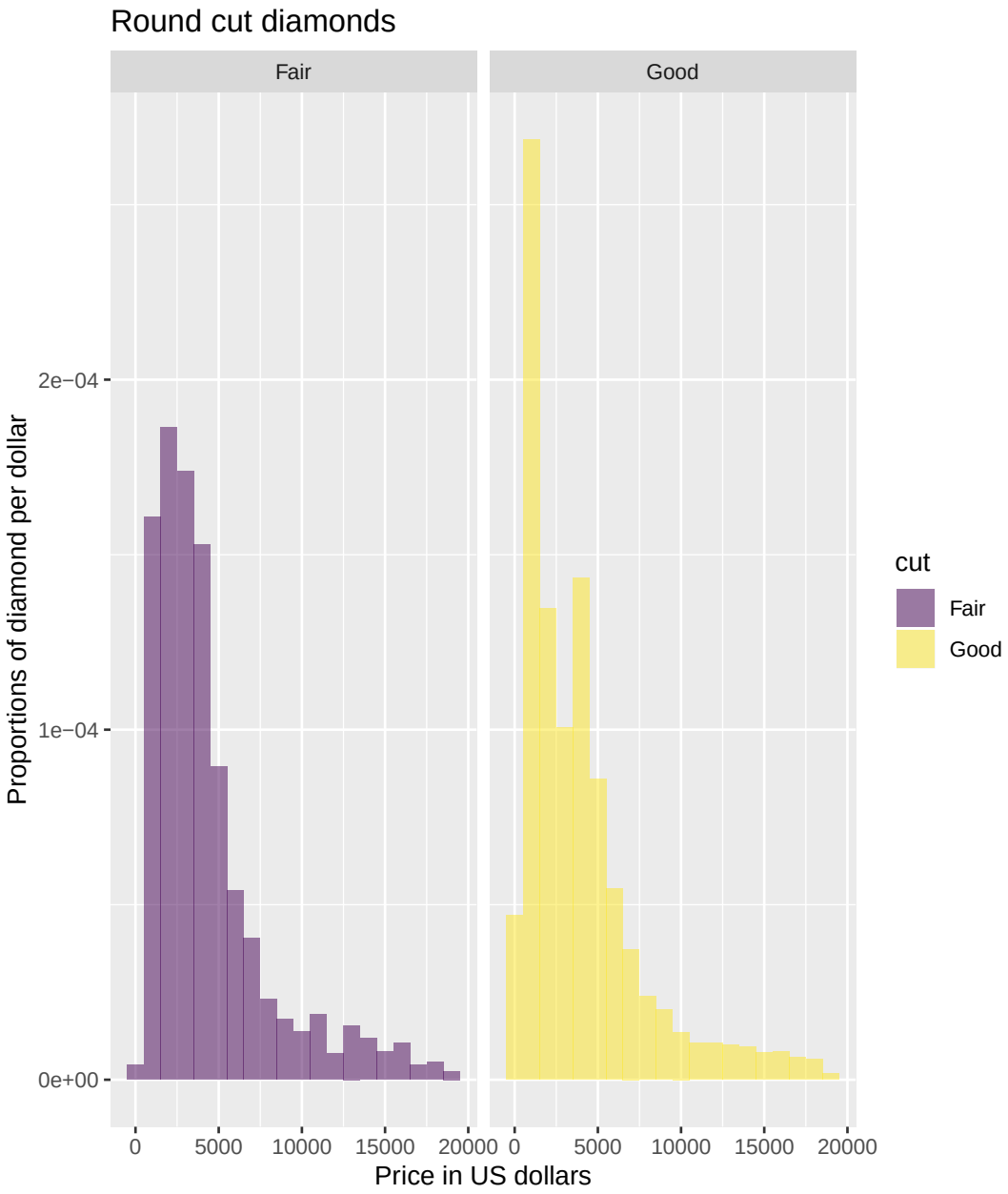
```
x = "Price in US dollars",
y = "Proportions of diamond per dollar",
title = "Round cut diamonds"
```

```
#Superimpose the plots
ggplot(my.diamonds,  aes(x=price, y = ..density..)) +
  geom_histogram(mapping = aes(fill = cut),
                 binwidth = 1000, alpha = .5, position="identity") +
  labs(x = "Price in US dollars",
       y = "Proportions of diamond per dollar",
       title = "Round cut diamonds")
```



c. Use the subset in part a, draw histograms for the prices of diamonds for each different cut; display the two histograms side by side.

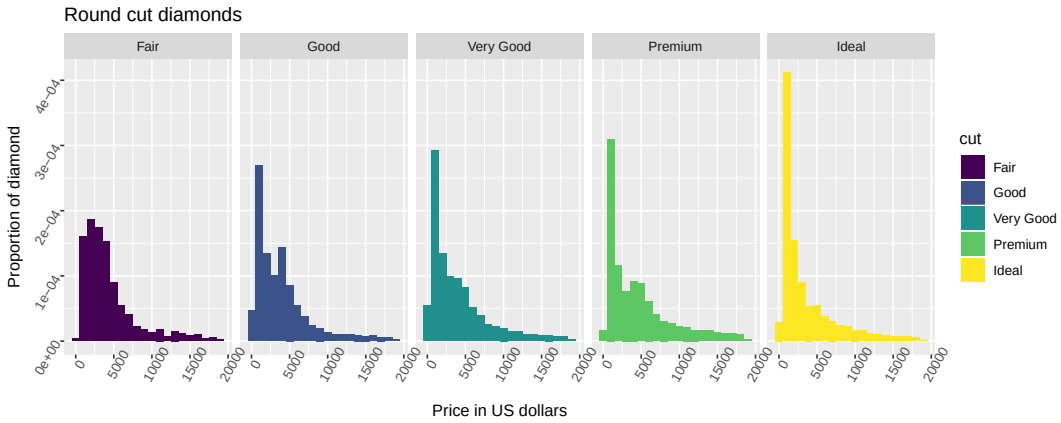
```
#Use facet_grid to create side-by-side histograms based on cut

ggplot(my.diamonds,  aes(x=price, y = ..density.., fill = cut)) +
  geom_histogram(binwidth = 1000, alpha = .5, position="identity") +
  facet_grid(.~cut) +
  labs(x = "Price in US dollars",
       y = "Proportions of diamond per dollar",
       title = "Round cut diamonds")
```



d. Now, repeat part c but for all types of the diamonds (i.e., use the original `diamond` dataset instead of the subset); also, color the histograms according to cut. What trend do you observe?

```
#Use density for the y-axis and change the colour according to the cut
#Notice how using density helps us identify the trend
ggplot(diamonds,  aes(x=price, ..density.., fill = cut)) +
  geom_histogram(binwidth = 1000) + facet_grid(.~cut)+
```

```
  labs(x = "Price in US dollars",
       y = "Proportion of diamond",
       title = "Round cut diamonds") +
  theme(axis.text=element_text(angle=60))
```



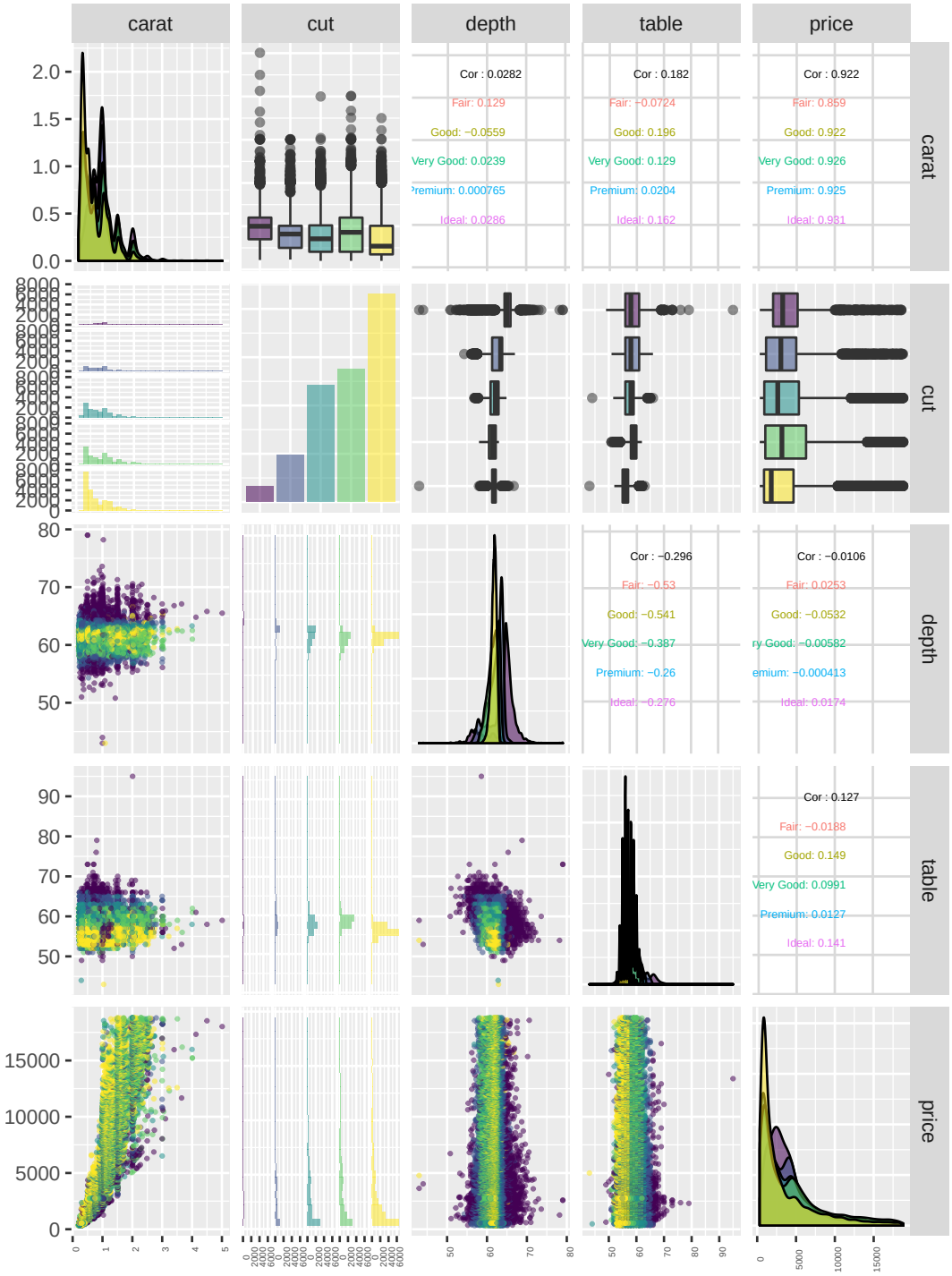
We observe that regardless of the grades of the diamonds, both of their prices have a mode around \$1500 to \$2000. Also, ideal grade diamonds don't seem to be more expensive.

Suggestion: Change binwidth for better interpretation.

e. Compare the pairwise relationship between the variables: `price`, `carat`, `depth` and `table` for different categories in `cut`. Use `ggpairs()` to do this.

```
str(diamonds, restrict.width = 'cut')
Classes 'tbl_df', 'tbl' and 'data.frame':   53940 obs. of  10 variables:
 $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
 $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
 $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
 $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
 $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
 $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
 $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
 $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
 $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
 $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
# the column `cut` is already a factor vector

ggpairs(diamonds[, c(1, 2, 5:7)], aes(colour = cut, alpha = 0.4),
        upper = list(continuous = wrap("cor", size = 1.9)),
        lower = list(continuous = wrap('points', size = .5))
        ) +
  theme(axis.text.x = element_text(angle = 90, size = 4))
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



3. Use the `subset()` function to redo 2.a. If you already used the `subset()` function in 2.a, use extraction with logical row indices to redo 2.a.

`subset()` can be used on vectors, matrices and data frames. The following is an example that demonstrates how `subset()` works.

```
# The two objects below are equivalent.

use.subset = subset(x = diamonds, subset = (diamonds$cut == "Good"))
# Here `x` is the object to be subsetted.  `subset` is the
# logical expression indicating the rows to keep: missing values
# are taken as false.

use.row.index = diamonds[diamonds$cut == "Good", ]

# The following checks if the corresponding elements in each
# vector are equal and then count the number of the element
# pairs that are not equal
sum(use.subset != use.row.index)
[1] 0
# all are equal
```

```
my.diamonds = diamonds[diamonds$cut %in% c("Good", "Fair"), ]
my.diamonds2 = subset(diamonds, cut %in% c("Good", "Fair"))

# check:
sum(my.diamonds != my.diamonds2)
[1] 0
```