Week_2_lab

Dan Crownover

January 27, 2024

This lab has a lot of information in it. You can refer to these sections to help you with your problem set (and future problem sets)

In this lab we will work on the following:

1. A bit more about R Notebooks
2. More on installing Packages and Loading libraries
3. Working directories
4. Importing data
5. Summary statistics
6. Inline R code
7. Piping in R
8. Making a histogram (you will upload to canvas)
9. Bonus: A scatter plot
10. Bonus: A boxplot
11. Bonus: Piping with ggplot 12: T tests in R 13: Non-parametric tests in R 14: Log example

To turn in today (5 points): a .png file of the histogram on pm2.5 data. The histogram should be uploaded by 12pm Friday (noon), EDT for full credit.

#1 A bit about R Markdown and Notebooks

R markdown is a simple and easy to use plain text language used to combine your R code, results from your data analysis (including plots and tables) and written commentary into a single nicely formatted and reproducible document (like a report, publication, thesis chapter or a web page like this one).

Technically, R markdown is a variant of another language (yet another language!) called Markdown and both are a type of 'markup' language. A markup language simply provides a way of creating an easy to read plain text file which can incorporate formatted text, images, headers and links to other documents.

At the top of the R Markdown document between the dashed lines, you see what is called the YAML header and you can change for the markdown in the header. The first line of the YAML header is the title of the document. Please go ahead and change this as well as the author name. You can select what kind of output you want when you knit the document. I would like you to submit your reports as a PDF, so you will want to set that as an option.

If you go to the gear pull down menu above, you can play around with the options. At the bottom of the pull down menu options is Output Options. Make sure that PDF is selected. You can also decide if you want to see the output from the chunks inline (below the chunk) or in the console. Please select Chunk Output Inline. This will allow us to see the output in the document we knit.

An R Notebook is an R Markdown document with chunks that can be executed independently and interactively, with output visible immediately beneath the input.

#2 Install and load the libraries.

If you haven't installed the packages, you will need to do that first.Remember to install a packages, you'll need to type install.packages("NameOfPackage") in the console. I typically do this in the console because once you do it, you don't need to repeat it, so therefore I don't need that command stored in my rmarkdown. However, you will need to load the packages each time, so I always start my Rmd with an R chunk that i call library. See how I named the chunk below library? That is the name following r in the curly brackets. You can name your chunks whatever (or not name them at all).

```r
library(coin)
```

```
## Loading required package: survival
```

```r
library(lawstat)
library(ggpubr)
```

```
## Loading required package: ggplot2
```

```r
library(rstatix)
```

```
##
## Attaching package: 'rstatix'
```

```
## The following objects are masked from 'package:coin':
##
##     chisq_test, friedman_test, kruskal_test, sign_test, wilcox_test
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```r
library(ggplot2) #a data visualiztion package
library(ggthemes) #a package of themes for visualizations. themes are settings to make are visualizatio
library(RColorBrewer) # I like color, RColorBrewer is a package of colors and color combinations.
library(moments) #allows us to calculate skewness and kurtosis

library(kableExtra) #this is for making data tables in R
```

#3 The working directory

Remember, to figure out the current working directory (where R will read and store files), you should use the function getwd(). You don't need to put anything in the brackets (no arguments). The working directory is where R is storing and reading files. It makes things smoother if all files are stored in one place. Please download the worldbank.csv file and place it in a suitable folder (remember, mine is called Data).

If you would like to change the working directory, use the function setwd(). You'll need to change the pathname to the appropriate path on your machine.

```r
getwd() # tells you the current working directory
```

```
## [1] "/Users/daniel/Downloads/BIO 591 Coding Files/Labs/Lab 2"
```

```
#setwd("~/Documents/Classes/BIO562_2023/Labs/BIOL653_2023")
# if you want to change the directory, use setwd(). You need to remove the # in the line above and chan
```

#4 Importing data

Now we are ready to load the data. First, please open the data in excel. I always do this to make sure I'm happy with how it looks. Make sure there aren't any missing values, columns are aligned, and overall looks clean. Once you do that, make sure the file is saved in your working directory as a .csv. You can now read in the .csv data file. Name the dataframe a name that makes sense to you. Once I read in the data, i always like to check out the first few lines to make sure it looks good. The function for this is head(). You can also look at the structure of the data frame with the str() function. The structure tells us about the variables and number of observations.

```
## Reading in worldbank data
worldbank.df <- read.csv("~/Downloads/BIO 591 Coding Files/Labs/Lab 2/Lab 2 Data/worldbank-2.csv")

## displaying the first head?
## str displays string of data
head(worldbank.df)
```

```
##               Country  Region  urban urban_growth      pm25
## 1 C\xf4te d\x92Ivoire African 54.869     3.774025 23.534438
## 2             Lesotho African 27.840     3.247536 24.701170
## 3             Liberia African 50.100     3.305928  7.851815
## 4             Algeria African 71.304     2.637964 35.564533
## 5              Angola African 44.819     5.098251 36.395428
## 6               Benin African 44.395     3.770961 35.162373
```

```
str(worldbank.df)
```

```
## 'data.frame':    160 obs. of  5 variables:
##  $ Country     : chr  "C\xf4te d\x92Ivoire" "Lesotho" "Liberia" "Algeria" ...
##  $ Region      : chr  "African" "African" "African" "African" ...
##  $ urban       : num  54.9 27.8 50.1 71.3 44.8 ...
##  $ urban_growth: num  3.77 3.25 3.31 2.64 5.1 ...
##  $ pm25        : num  23.53 24.7 7.85 35.56 36.4 ...
```

In the dataframe you should see five variables: (1) country; (2) UN regional group; (3) Urban; (4) Urban growth and (5) PM 2.5. The three numeric variables were all downloaded from World Bank websites. It's always good to check out the source of the data and make sure you understand the units of measurement. We call the data about the data, metadata.

Metadata urban: % of total population that lives in urban area, source World Bank (2015) urban_growth: Annual % change in urban population (%), source World Bank (2015) pm25: Particulate Matter 2.5,PM2.5 air pollution, mean annual exposure (micrograms per cubic meter), source World Bank

You should see that the data frame has 162 observations with five variables. Country and Region are factor variables. urban, urban growth and pm2.5 are numeric.

#5 Summary statistics

We can run summary statistics on several measures of central tendency and spread. Ill do it for the first (urban) and you could run the same for urban_growth and pm25. This time I'm going to assign objects to each of the summary statistics so i can then insert them in text. You can see that I used the argument na.rm=TRUE. This tells R to remove any missing values (NA) from the analysis (rm=remove).

```
## running summary statistics for only URBAN column --> the specified comlumn comes after $
length.urban<-length(worldbank.df$urban)
mean.urban<-mean(worldbank.df$urban, na.rm=TRUE)
median.urban<-median(worldbank.df$urban, na.rm=TRUE)
sd.urban<-sd(worldbank.df$urban, na.rm=TRUE)
IQR.urban<-IQR(worldbank.df$urban, na.rm=TRUE)
range.urban<-range(worldbank.df$urban, na.rm=TRUE)
skewness.urban<-skewness(worldbank.df$urban, na.rm=TRUE) #skewness is in the packages 'moments'.
## na.rm = true is what is telling them to perform statistical calclulations on the data
#If we want to see the output, we can call those objects:
length.urban
```

```
## [1] 160
```

```
mean.urban
```

```
## [1] 57.92549
```

```
median.urban
```

```
## [1] 59.1955
```

```
sd.urban
```

```
## [1] 22.08894
```

```
IQR.urban
```

```
## [1] 36.464
```

```
range.urban
```

```
## [1]  8.352 98.358
```

```
skewness.urban
```

```
## [1] -0.2107753
```

#6 Inline Coding We can discuss the summary statistics in a paragraph using inline R

R Notebook allows us to use inline R code as we report our results. To use R inline, you need to put a mark () `followed by the letter r, then the code you want, then close with a mark` (). For example:

Across 162 nations, the average percent of the population that lives in urban areas is 57.9254875%.

Go ahead and knit it! Do you see the output of this sentence in PDF?

As you look over the PDF doc, you will see that there are way too many significant digits that are reported. Please report the appropriate number of digits–in this case I would feel comfortable reporting to one or two decimal places. So how can we do this? There is a round function in R round(). So let's try this 57.93%. Go ahead and knit and see what happens. The mean should be reported to two decimal places.

Now, you could have calculated the mean using just inline R, by typing 58% instead of calculating the mean in the chunk above and assigning it to an object. Either way, you should get the same thing. I like running my summary stats in one chunk so i can see them together and easily run the chunk, but this is more of a personal preference.

It's good practice not to type in the numeric results directly in the .Rmd (such as "the mean was found to be 57.77%"). If we find a mistake in the data or have to rerun it, we then have to go back and make changes. This is a pain in the neck and is often an error-prone process. We want to avoid this if at all possible. Inline code is one of the advantages of running R in a notebook with Rmd.

#7 Introduction to PIPING in R The pipe operator, written as %>%, has been a longstanding feature of the magrittr package for R. It takes the output of one function and passes it into another function as an argument. This allows us to link a sequence of analysis steps.

To visualize this process, imagine a factory with different machines placed along a conveyor belt. Each machine is a function that performs a stage of our analysis, like filtering or transforming data. The pipe therefore works like a conveyor belt, transporting the output of one machine to another for further processing.

We can see exactly how this works in a real example using the mtcars dataset. This dataset comes with base R, and contains data about the specs and fuel efficiency of various cars. The code below groups the data by the number of cylinders in each car, and then returns the mean miles-per-gallon of each group. Make sure to install the tidyverse suite of packages before running this code, since it includes both the pipe and the group_by and summarise functions.

```
## the %> is the pipe operator, the pipe operator acts as a conveyorbelt which transposes infornation t
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v lubridate 1.9.2     v tibble    3.2.1
## v purrr     1.0.1     v tidyr     1.3.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter()     masks rstatix::filter(), stats::filter()
## x dplyr::group_rows() masks kableExtra::group_rows()
## x dplyr::lag()        masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
result <- mtcars %>%
    group_by(cyl) %>%
    summarise(meanMPG = mean(mpg))
```

The pipe operator feeds the mtcars dataframe into the group_by function, and then the output of group_by into summarise. The outcome of this process is stored in the tibble result

```
result
```

```
## # A tibble: 3 x 2
##     cyl meanMPG
##   <dbl>   <dbl>
## 1     4    26.7
## 2     6    19.7
## 3     8    15.1
```

Although this example is very simple, it demonstrates the basic pipe workflow. To go even further, I'd encourage playing around with this. Perhaps swap and add new functions to the 'pipeline' to gain more insight into the data. Doing this is the best way to understand how to work with the pipe. But why should we use it in the first place?

The pipe has a huge advantage over any other method of processing data in R: it makes processes easy to read. If we read %>% as "then", the code from the previous section is very easy to digest as a set of instructions in plain English:

Load tidyverse packages To get our result, take the mtcars dataframe, THEN Group its entries by number of cylinders, THEN Compute the mean miles-per-gallon of each group

This is far more readable than if we were to express this process in another way. The two options below are different ways of expressing the previous code, but both are worse for a few reasons.

```
# Option 1: Store each step in the process sequentially
result <- group_by(mtcars, cyl)
result <- summarise(result, meanMPG = mean(mpg))
# Option 2: chain the functions together
result <- summarise(
                group_by(mtcars, cyl),
                meanMPG = mean(mpg))
```

Option 1 gets the job done, but overwriting our output dataframe result in every line is problematic. For one, doing this for a procedure with lots of steps isn't efficient and creates unnecessary repetition in the code. This repetition also makes it harder to identify exactly what is changing on each line in some cases.

Option 2 is even less practical. Nesting each function we want to use gets ugly fast, especially for long procedures. It's hard to read, and harder to debug. This approach also makes it tough to see the order of steps in the analysis, which is bad news if you want to add new functionality later.

It's easy to see how using the pipe can substantially improve most R scripts. It makes analyses more readable, removes repetition, and simplifies the process of adding and modifying code. Is there anything it can't do?

Read more about piping here: https://towardsdatascience.com/an-introduction-to-the-pipe-in-r-823090760d64

##7.1 displaying summary statistics in a table Now that I have introduced you to piping, we will use the functions kableExtra and piping notation to summarize into a nice data table.

the r chunk below will summarize the numeric variables in the worldbank.df.

```
worldbank.df %>%
  summarize(`Mean PM 2.5` = mean(pm25), `Median PM 2.5` = median(pm25)) %>%
  ungroup
```

Mean PM 2.5 Median PM 2.5 1 29.24154 22.98965

That is kind of ugly, lets use Kable to make it prettier

```
worldbank.df %>%
  summarize(`Mean PM 2.5` = round(mean(pm25), 2), `Median PM 2.5` = round(median(pm25), 2)) %>%
  ungroup %>%
  kbl() %>% kable_styling()
```

We can also separate out summary statistics by region

Mean PM 2.5	Median PM 2.5
29.24	22.99

Region	Mean PM 2.5	Median PM 2.5
African	37.38	35.16
Asia Pacific	42.88	37.60
Eastern European	24.66	20.55
Latin American and Caribbean	19.87	17.99
Western Europe and Others (WEOG)	11.65	11.02

```r
worldbank.df %>%
  group_by(Region) %>%
  summarize(`Mean PM 2.5` = round(mean(pm25), 2), `Median PM 2.5` = round(median(pm25), 2)) %>%
  ungroup %>%
  kbl() %>% kable_styling()
```

#8 Making a histogram

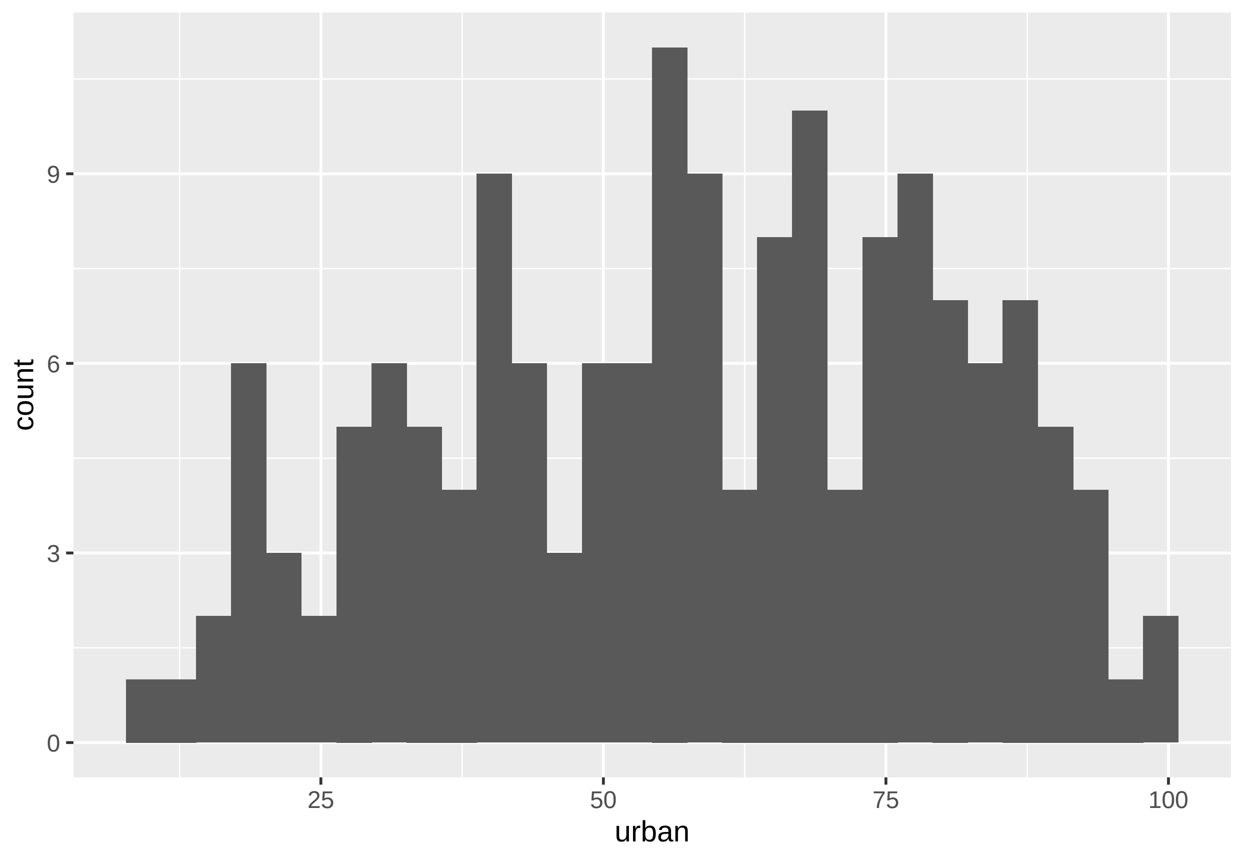
Let's make a histogram of the variable urban and let's make it pretty. We will build it in phases, so you can see how it is done.

We will start off with the ugly, basic histogram. The first argument is the data frame. The second argument is the aesthetic. We will set this to urban. You do not need to tell R that it's in the dataframe worldbank.df, since that is already set. The plus sign at the end of the line tells R to carry on to the next line. geom_ tells R which geometry to use. There are lots (historgram, boxplot, scatter, etc. etc.). We will start with the histogram.

Okay, please run the chunk below!

```r
ggplot(worldbank.df, aes(x=urban))+
  geom_histogram()
```

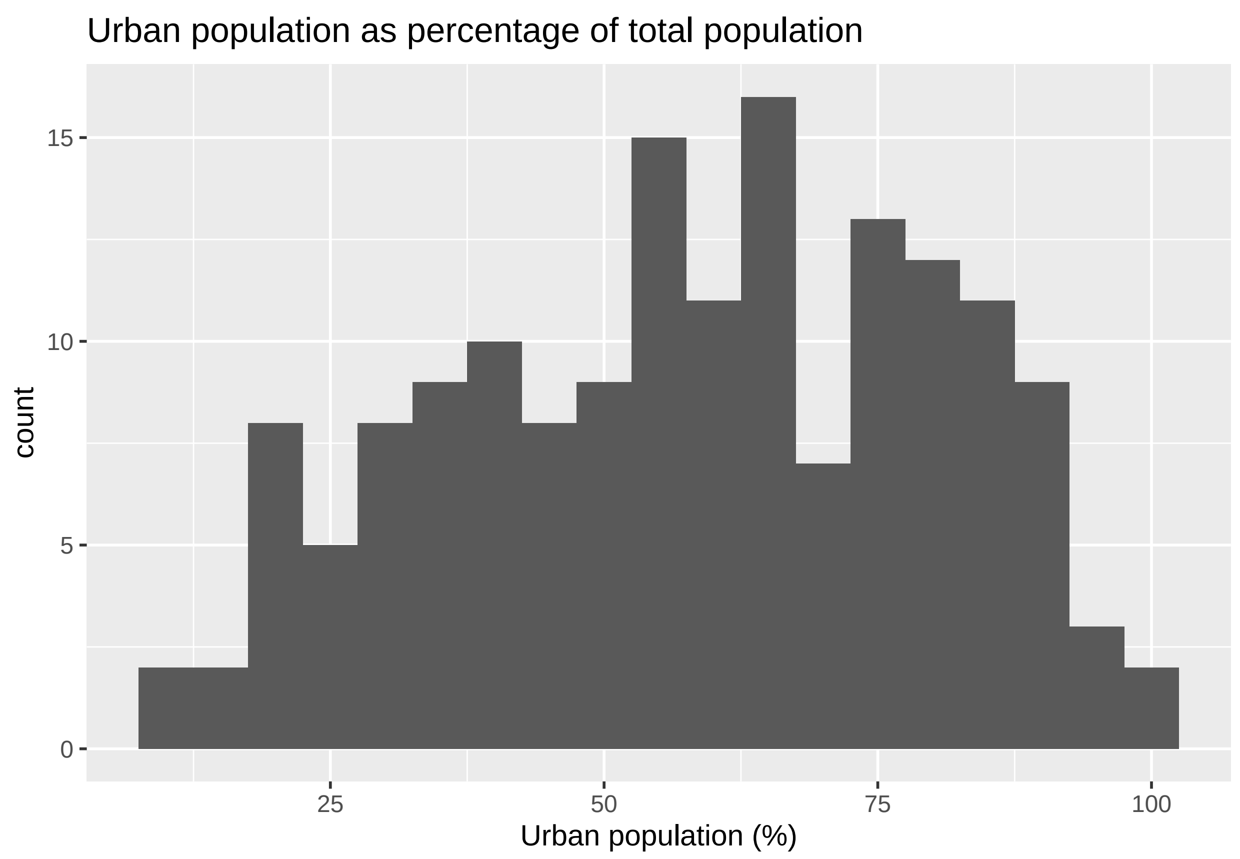
```
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```



Woah that's ugly! Be we can make it better.

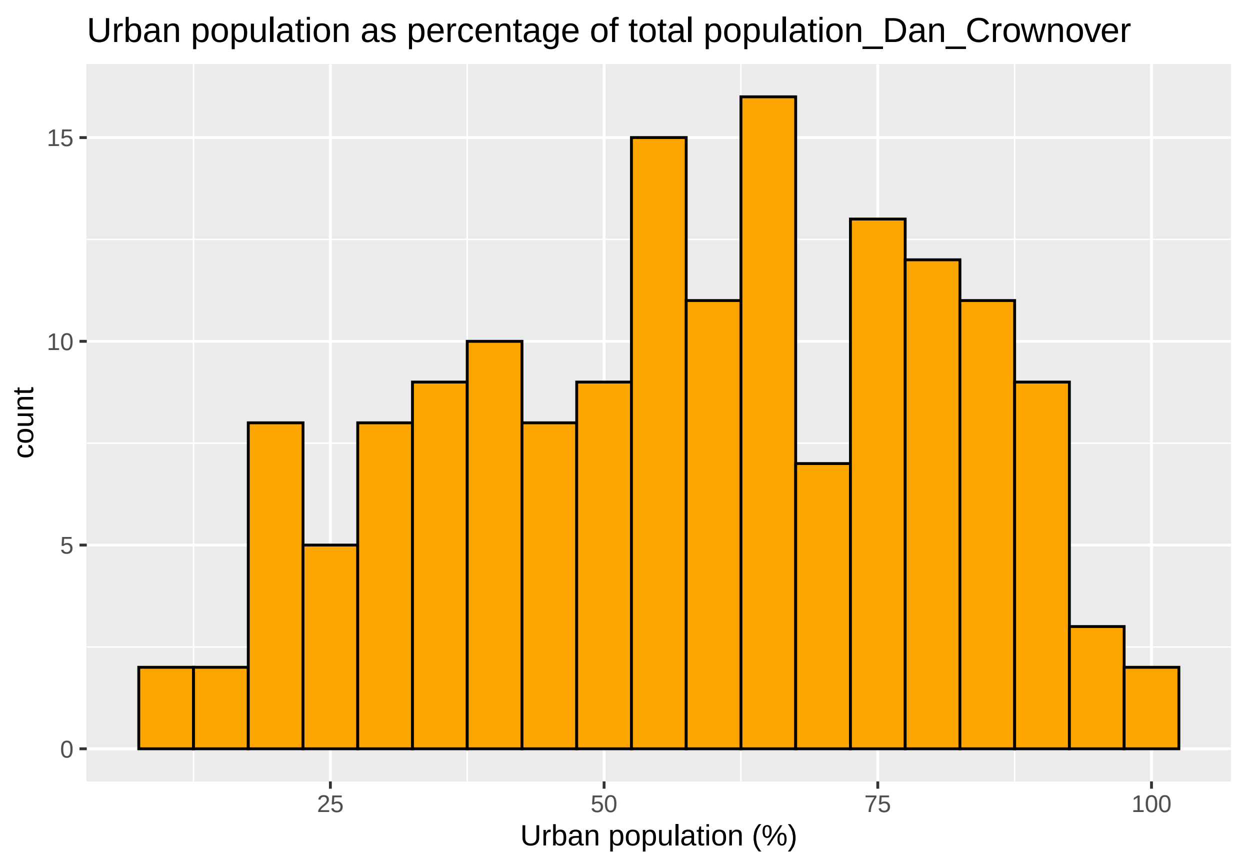
Let's next add labels and add a binwidth of 5.

```
ggplot(worldbank.df, aes(x=urban))+
  geom_histogram(binwidth=5) +
  labs(title="Urban population as percentage of total population", x="Urban population (%)")
```



Okay, it's a little better. But let's change its color. We can do this in the geom_histogram().

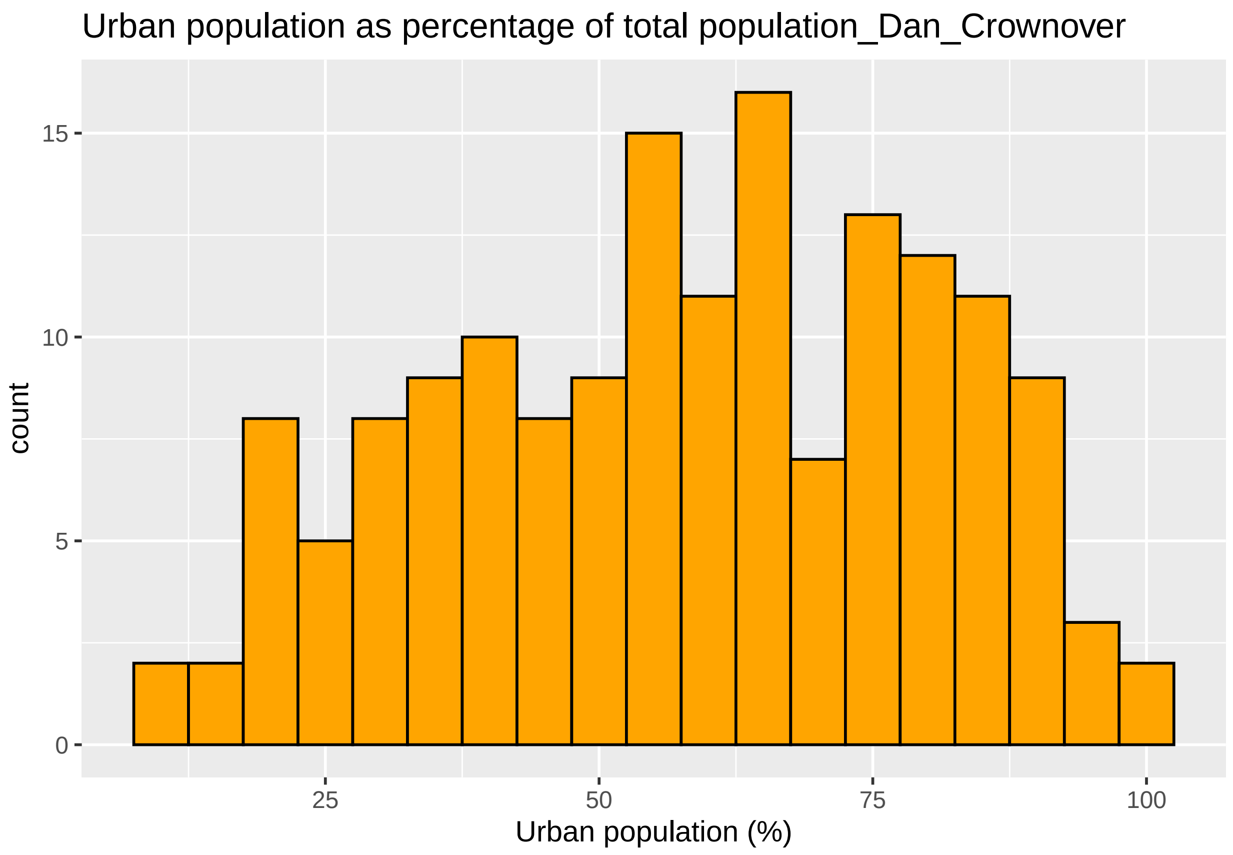
```
## changing the color
ggplot(worldbank.df, aes(x=urban))+
  geom_histogram(binwidth=5, color="black", fill="orange") +
  labs(title="Urban population as percentage of total population_Dan_Crownover", x="Urban population (%
```



##8.1 Saving an image as a .png in order to save the above plot we should first give it a name.

```
plot_to_save <- ggplot(worldbank.df, aes(x=urban))+
  geom_histogram(binwidth=5, color="black", fill="orange") +
  labs(title="Urban population as percentage of total population_Dan_Crownover", x="Urban population (%
```

```
#and we can look at it by calling it in the r chunk
plot_to_save
```



Now we can use ggsave() to save it as a .png in our working directory.

```r
##saving the plot with ggplot
ggsave("my_histogram.png", plot = plot_to_save)
```

```
## Saving 6.5 x 4.5 in image
```

#Assignment: Now it is your turn! Please make a histogram of the PM 2.5. Please be sure to label it appropriately and color it in with your favorite colors. Check out the STHDA website and play around with the figure. http://www.sthda.com/english/wiki/ggplot2-histogram-plot-quick-start-guide-r-software-and-data-visualization Upload the .png of your histogram to Canvas

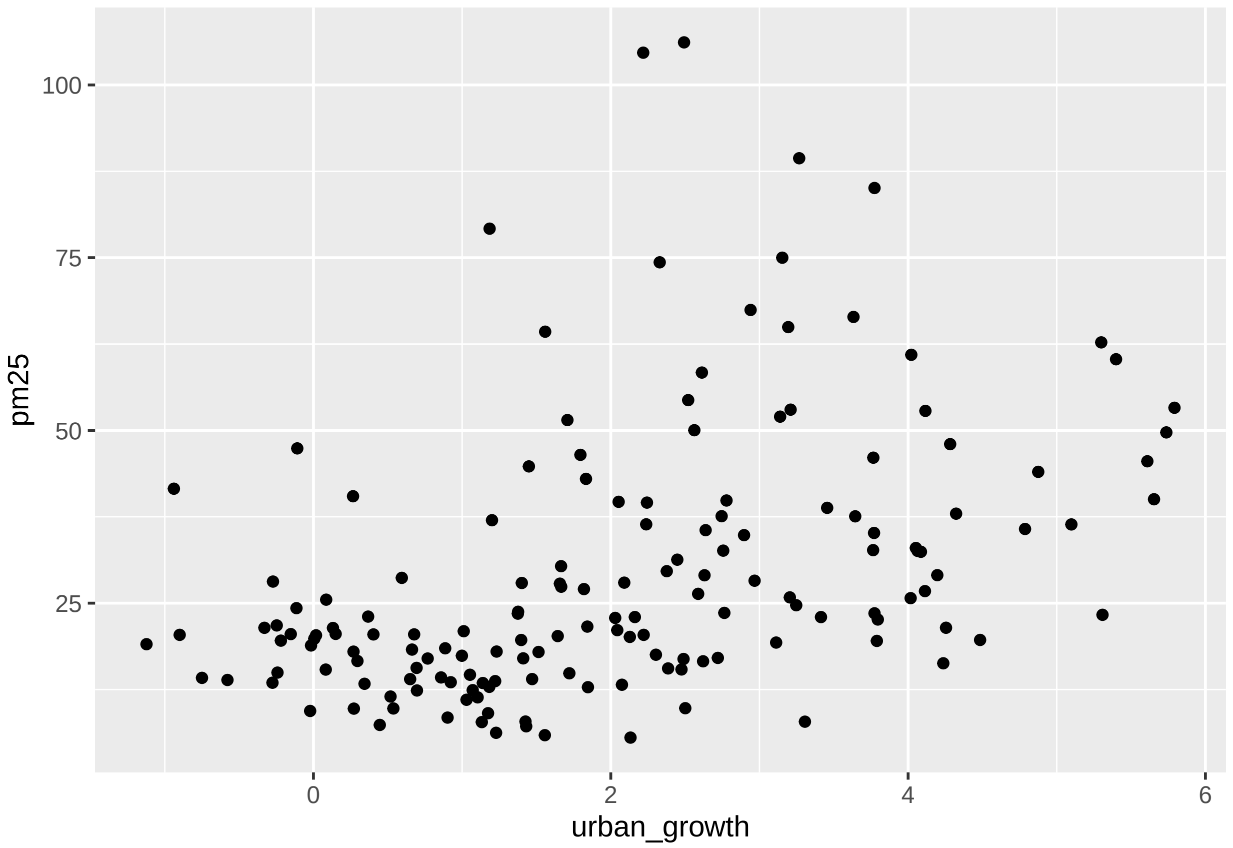
#9 Bonus: Drawing a scatter plot!

We are interested in plotting the relationship between urban growth rates and pm2.5 (particulate matter). We are also interested in if this relationship varies by UN Regional Group and relative size of urban population.

We will start off simply in the chunk below (called 'simplyugly').

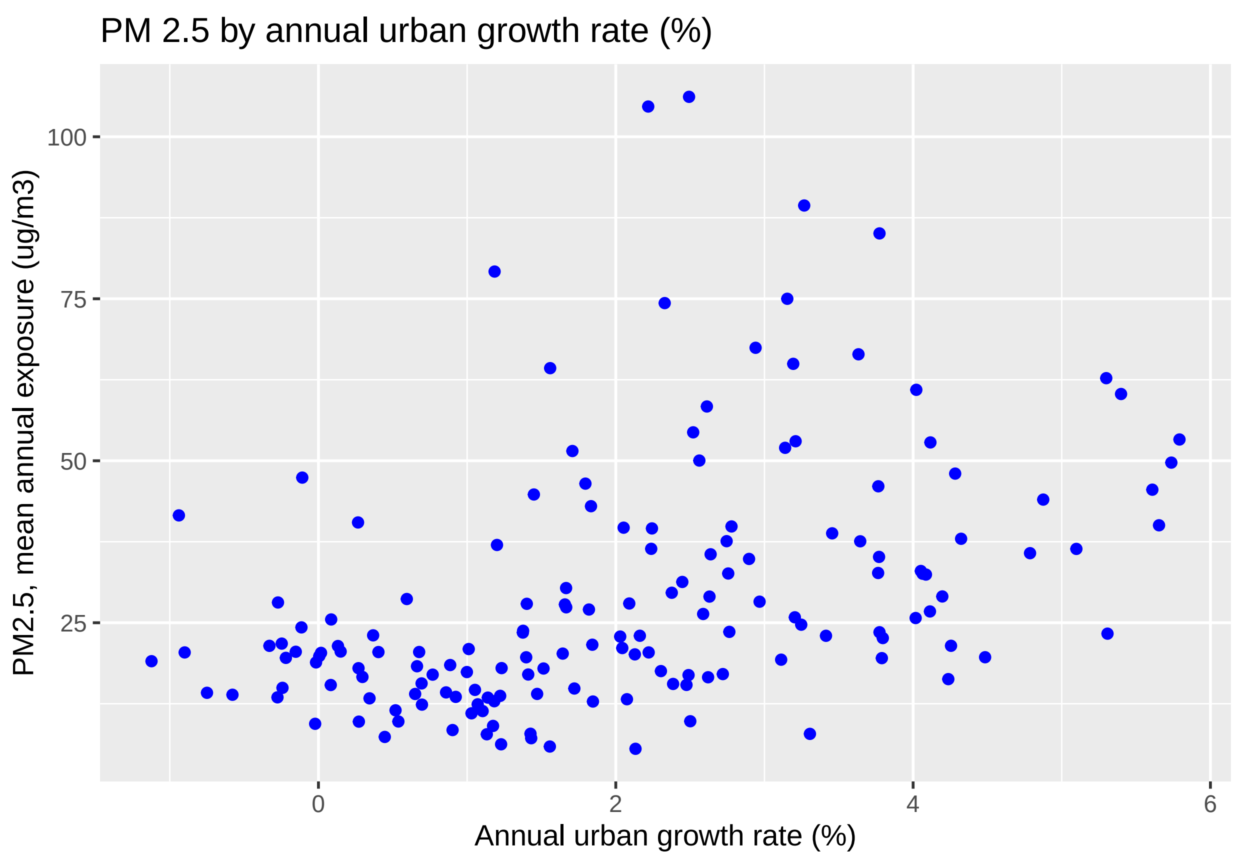
The first line sets the data frame (worldbank.df) and the aesthetics (x=urban_growth, y=pm25). geom_point() tells R to make a scatter plot.

```r
## this is the function--> using ggplot to create a scatter plot
ggplot(worldbank.df, aes(x=urban_growth, y=pm25))+
  geom_point()
```



Okay, that's pretty boring. Let's next add labels and make our points blue. Please feel free to change the color if you'd like.

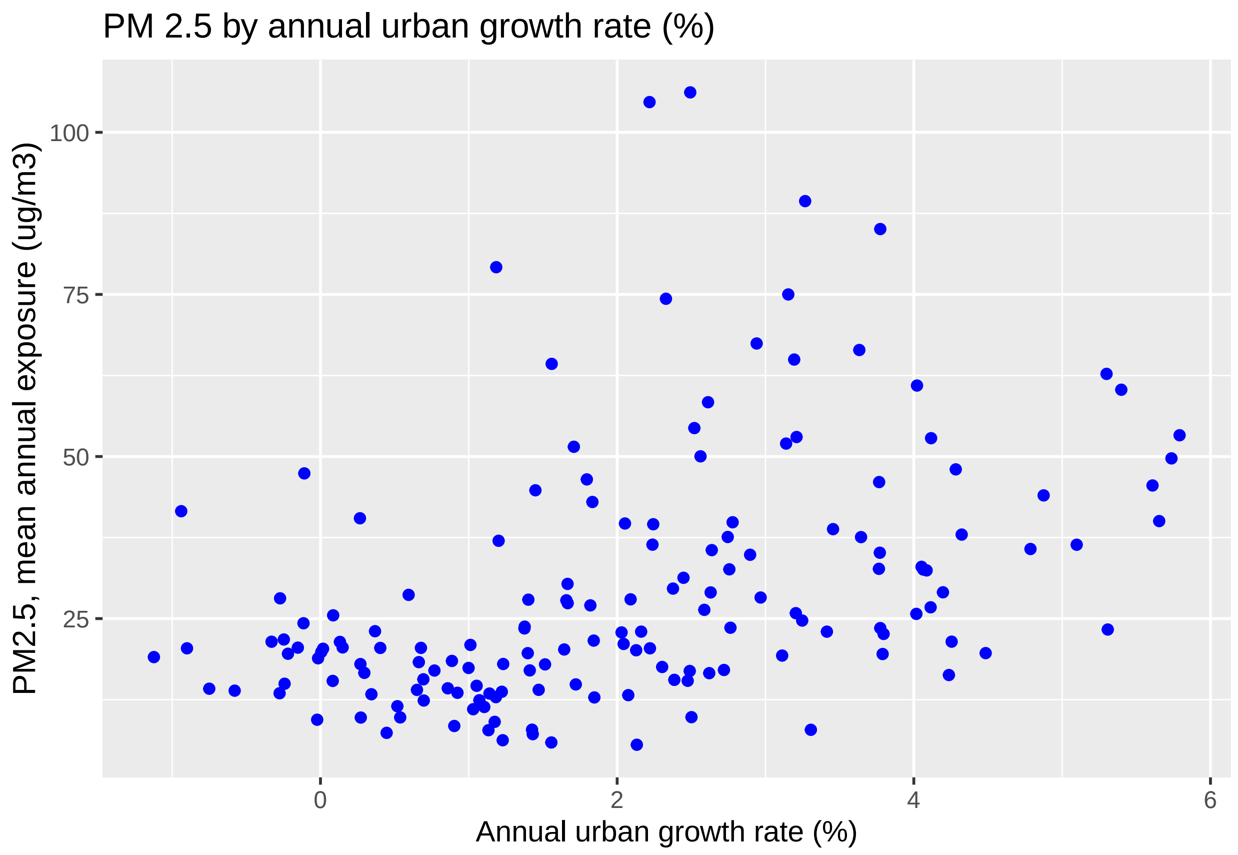
```
## changing color and adding the x and y axis
ggplot(worldbank.df, aes(x=urban_growth, y=pm25))+
  labs(title="PM 2.5 by annual urban growth rate (%)",
       x="Annual urban growth rate (%)", y = "PM2.5, mean annual exposure (ug/m3)")+
  geom_point(color="blue")
```



In the plot above, I noticed that the y-label was a little long. Let's see if we can make that look better (or we might want to shorten the title). I added a line about the theme where I set the axis.title.y to size equals ten. The theme sets the overall look of the figure.

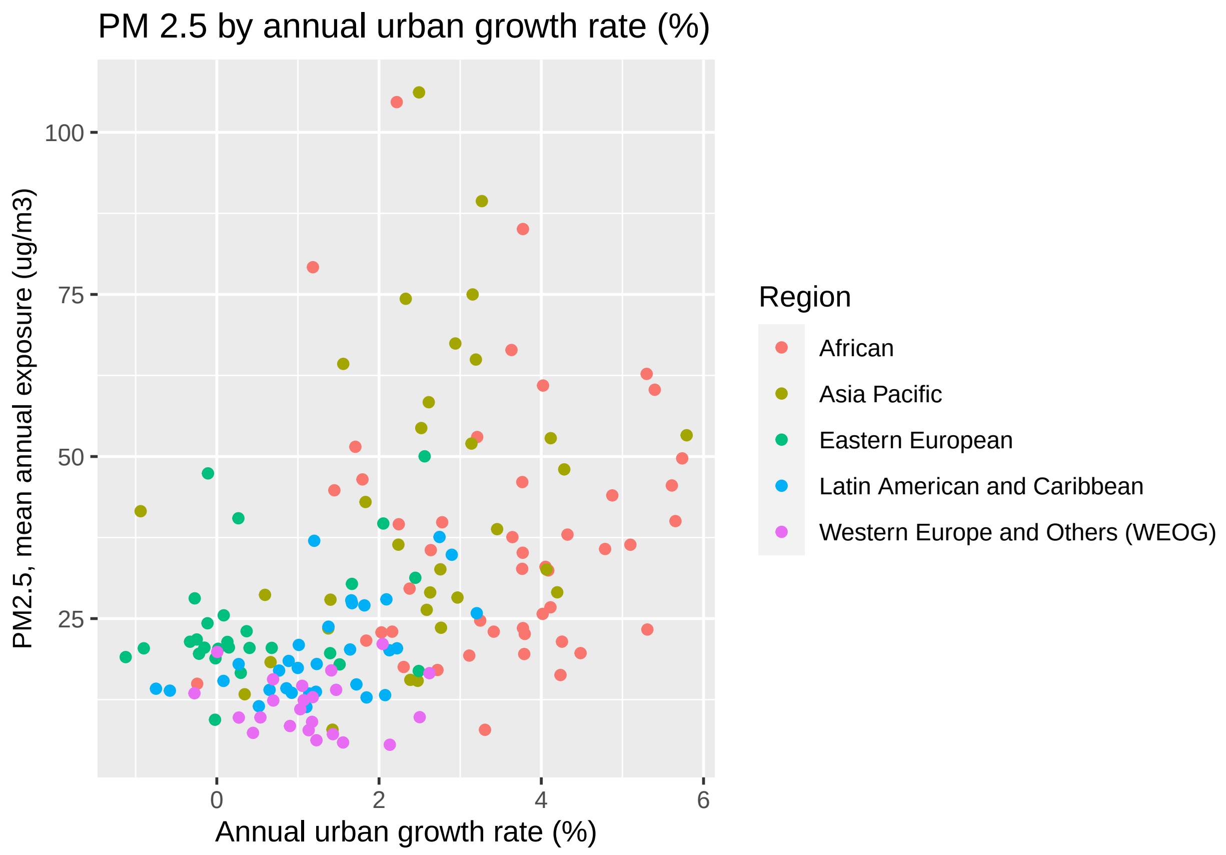
Play around with the size until it fits on the figure.

```
ggplot(worldbank.df, aes(x=urban_growth, y=pm25))+
  labs(title="PM 2.5 by annual urban growth rate (%)",
       x="Annual urban growth rate (%)", y = "PM2.5, mean annual exposure (ug/m3)")+

theme(
  axis.title.y = element_text(color="black", size= 12
                          ))+
  geom_point(color="blue")
```



So now let's get a little fancier and set the color of the points to represent the different UN regional groups. This is pretty easy to do. In aesthetics, you need to set col=Region. Region is the name of the variable, and col is short for color.We also need to delete the col="blue" in geom_point (I already did this for you).

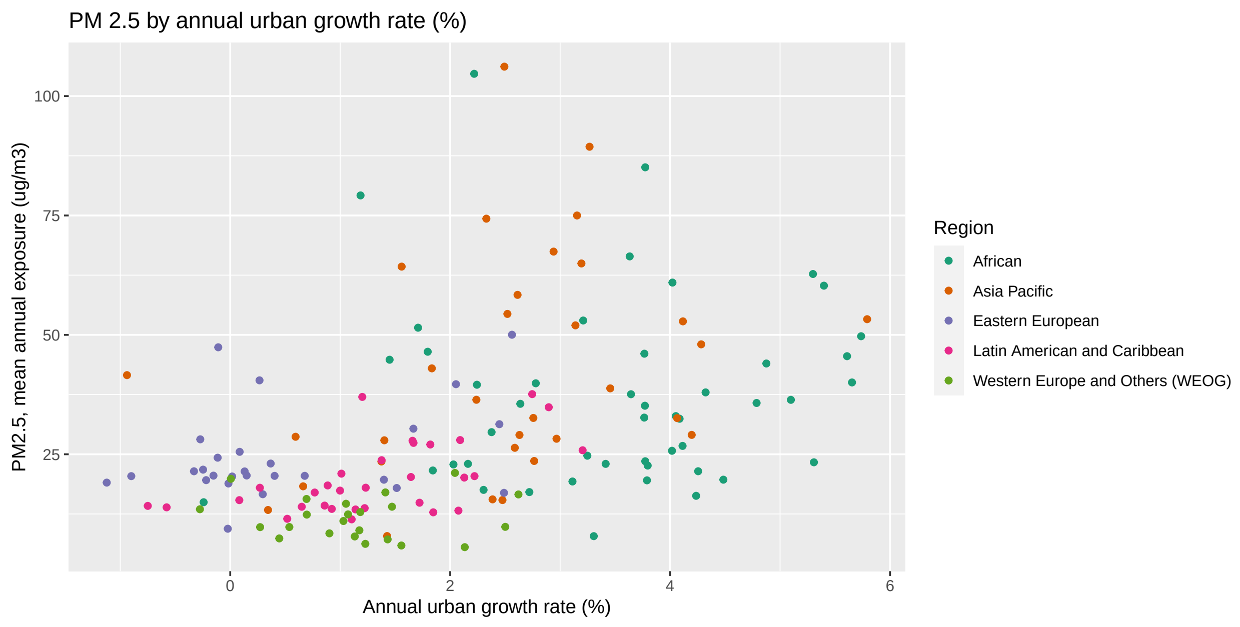
```
ggplot(worldbank.df, aes(x=urban_growth, y=pm25, col=Region))+
  labs(title="PM 2.5 by annual urban growth rate (%)",
       x="Annual urban growth rate (%)", y = "PM2.5, mean annual exposure (ug/m3)")+

theme(
  axis.title.y = element_text(color="black", size=10))+
  geom_point()
```



Okay, it is looking better! But I don't love the colors. There are lots of palettes out there that we could use. I frequently use palettes from the package RColorBrewer. Google RColorBrewer to check out the palettes. I'm going to switch the palette to Dark2. You can do this by adding a line to the plot: scale_color_brewer(palette="Dark2") + . Remember to put a + at the end of the line so R continues on to the next line. There is a lot to learn about use of colors and palettes. Be sure to check out the resources on Canvas to help you.

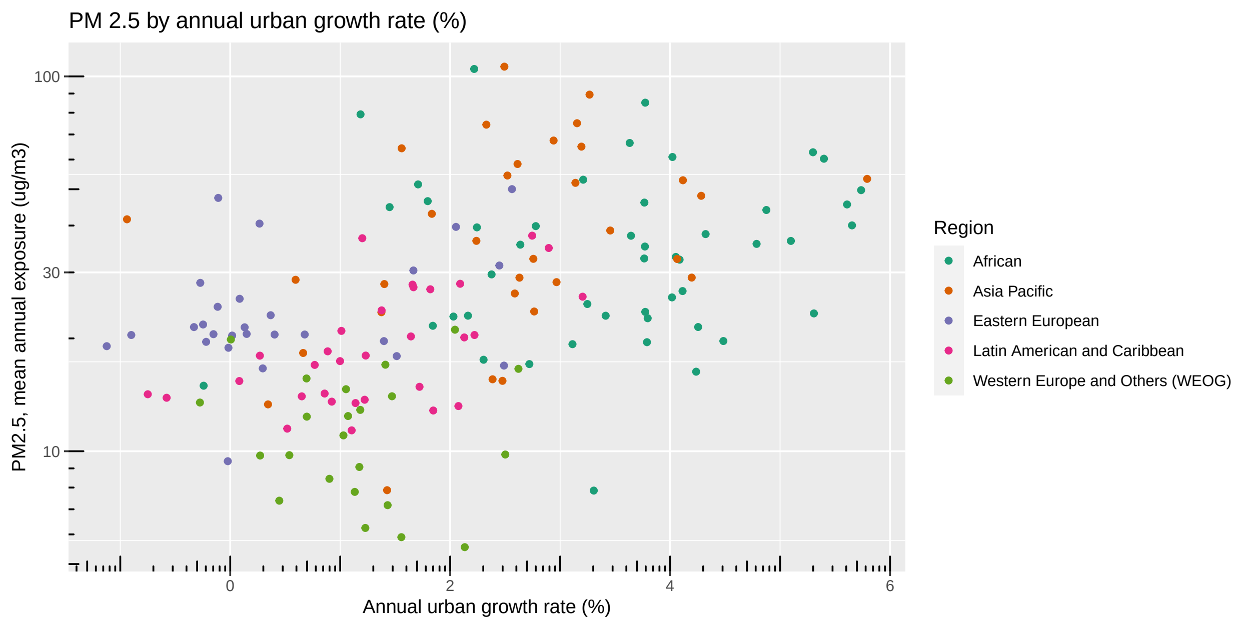
When we ran the plot above, it got a little squished because of the legend. Let's set the figure size. We can do that in the curly brackets. I am setting it to five by ten inches.

```r
ggplot(worldbank.df, aes(x=urban_growth, y=pm25, col=Region))+
  labs(title="PM 2.5 by annual urban growth rate (%)",
       x="Annual urban growth rate (%)", y = "PM2.5, mean annual exposure (ug/m3)")+
 scale_color_brewer(palette="Dark2") +
  geom_point()
```



Looking even better! One thing I noticed about the spread of the PM2.5 data is that the observations are more dense in number in the lower range and more sparse in the upper range. This suggests a skewed distribution. Because of this, we might want to consider putting the y-axis on a log scale. To do this, I am adding the line: scale_y_log10(). I also added annotation_logticks() to add tick marks.

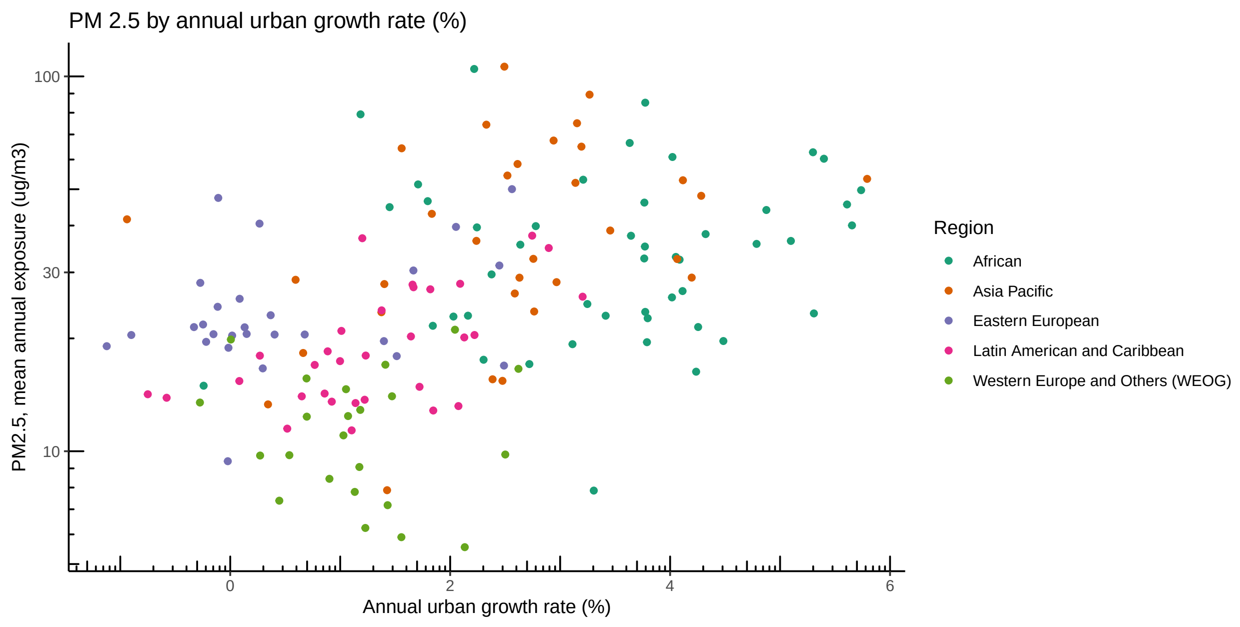
```r
ggplot(worldbank.df, aes(x=urban_growth, y=pm25, col=Region))+
  labs(title="PM 2.5 by annual urban growth rate (%)",
       x="Annual urban growth rate (%)", y = "PM2.5, mean annual exposure (ug/m3)")+
 scale_color_brewer(palette="Dark2") +
  ## addscale_y_log10()+ to make it on a log scale

  scale_y_log10()+
  ## don't forget this in the probelm set
  annotation_logticks()+ #This function makes the ticks on the axes in logscale.
  geom_point()
```



See how the y-axis changed? There are positives and negatives to changing the axis to a log scale–and it really depends on the purpose of the figure and the audience. If I'm interested in modeling a relationship between x and y, it's helpful for me to see if a log might render a more linear relationship. We could also do in essence the same thing if we logged the PM2.5 data. But we are getting way ahead of ourselves here. We will get to all of that when we get to simple linear regression.

I'm not a huge fan of the gray background. We can change this by changing the theme. I often use theme_classic() or theme_minimal(), but there are many more themes that you should check out. I'll run it first in theme_classic, then you switch it to theme_minimal to try it out.

```
ggplot(worldbank.df, aes(x=urban_growth, y=pm25, col=Region))+
  labs(title="PM 2.5 by annual urban growth rate (%)",
       x="Annual urban growth rate (%)", y = "PM2.5, mean annual exposure (ug/m3)")+
 scale_color_brewer(palette="Dark2") +
  scale_y_log10()+
  annotation_logticks()+
  theme_classic()+
  ## what is the geom point doing here?
  geom_point()
```



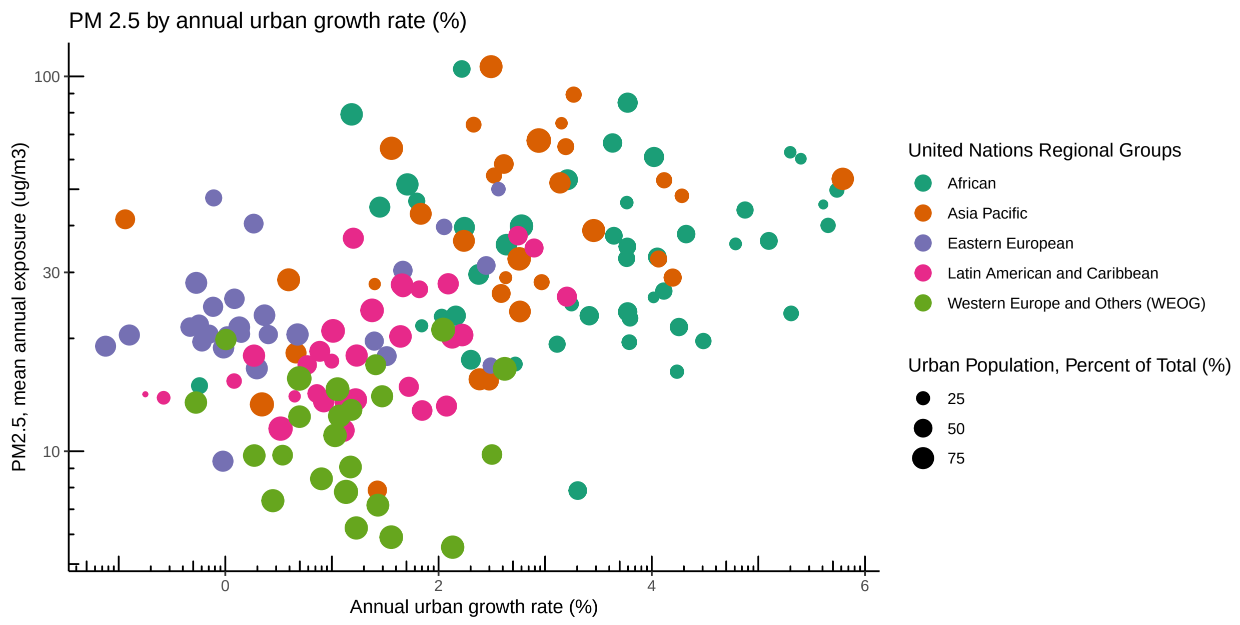
Okay, let's do one more thing. We can set the size of the points to the relative size of the urban population. We can do this in the aesthetic with size=urban.

The guides() function tells the plot the order of legends (which one you want on top), and the size of the points for color (size=4) in the legend.

In the chunk below I am naming the plot myplot. I do this so i can easily call it later or add to it. When you do this though, you will not see the plot until you call the name of the plot. You will do this is in the next code chunk. Run both code chunks.

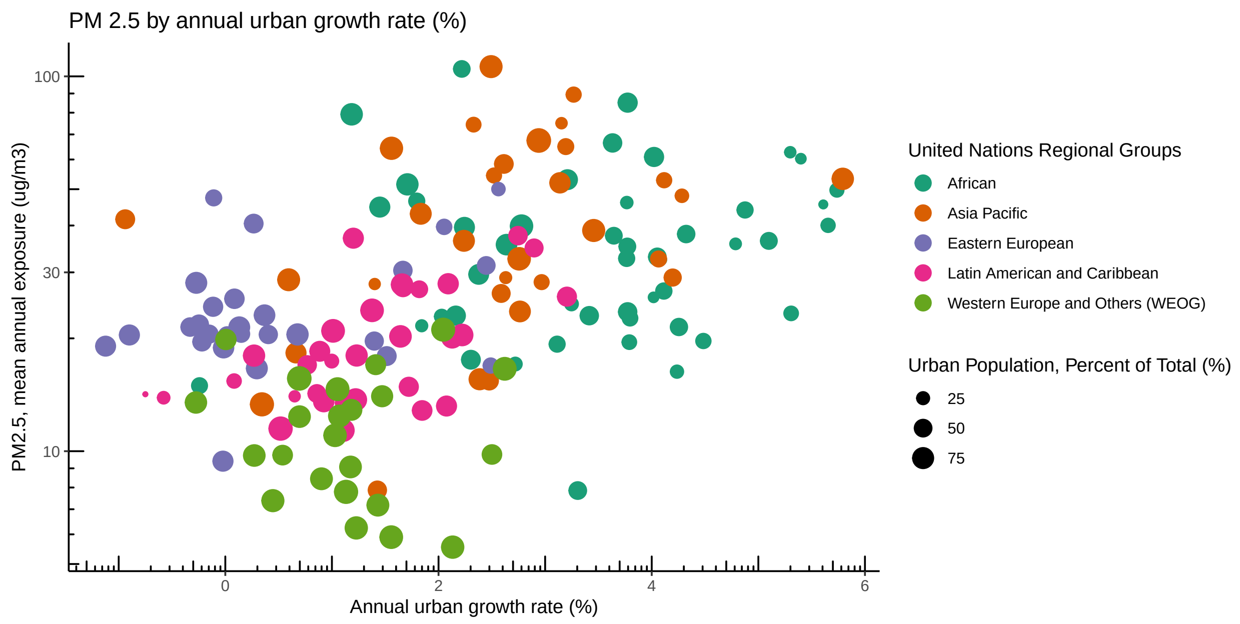
```
myplot<-ggplot(worldbank.df, aes(x=urban_growth, y=pm25, col=Region, size=urban))+
geom_point()+
scale_color_brewer(palette="Dark2") +
labs(title="PM 2.5 by annual urban growth rate (%)",
       x="Annual urban growth rate (%)", y = "PM2.5, mean annual exposure (ug/m3)")+
labs(col = "United Nations Regional Groups")+
  ##  below we are settiing size of the points to the relative size of the urban population(but what do
labs(size = "Urban Population, Percent of Total (%)")+
scale_y_log10()+
theme_classic()+
guides(color = guide_legend(order = 1, override.aes = list(size = 4)),
         size = guide_legend(order = 2))+
annotation_logticks()
```

```
myplot
```



Now we have a final plot. You can make adjustments in the size of the figure in the curly brackets to adjust the size. Feel free to adjust the fig.height and fig.width

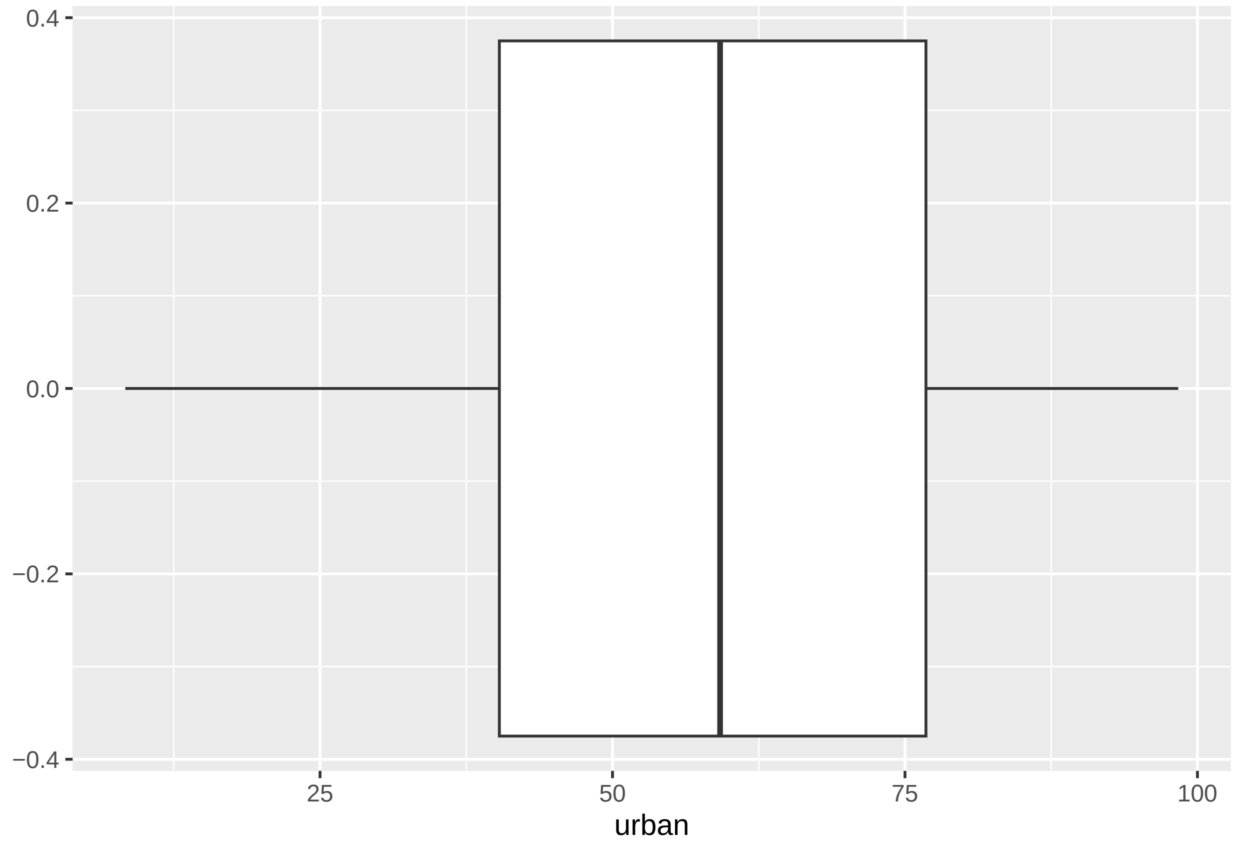
```
## here we are making adjustments to the height and width of the figure in the grey area above
myplot
```



Congratulations! You have finished the bonus scatterplot. It may be helpful to knit your document to see what it looks like. Also, please remember to upload the .png of the histogram.

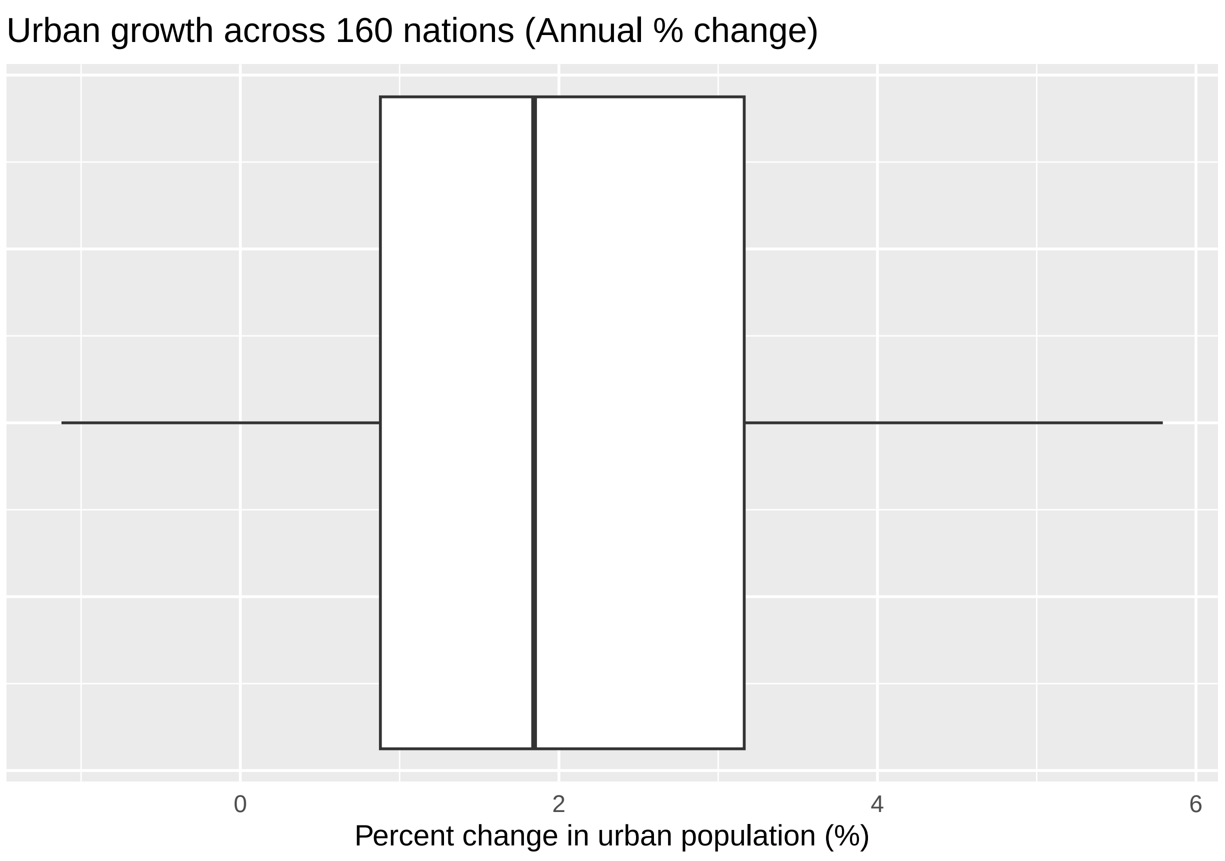
#10 Bonus: Drawing a boxplot Let's make a box plot of our urban population growth (urban_growth) variable. First Ill make a boxplot of the variable without grouping and then I will make a side-by-side box plot by Region. I encourage you to check out STHDA website on boxplots.

```r
ggplot(worldbank.df, aes(urban))+
geom_boxplot()
```



Okay, That's not great. We should change the label on the x-axis and remove the scale on the y-axis. We could change the color.

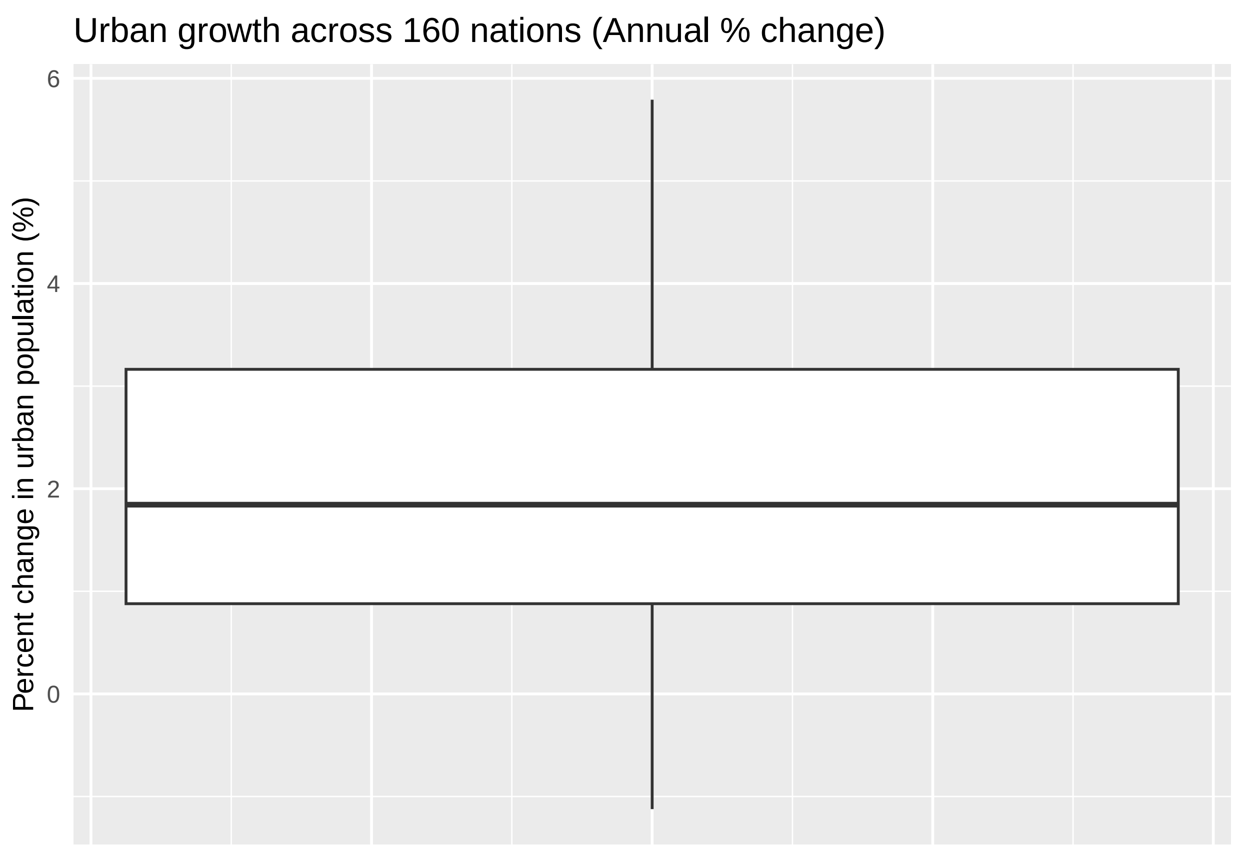
```r
ggplot(worldbank.df, aes(urban_growth))+
geom_boxplot()+
  labs(title="Urban growth across 160 nations (Annual % change)", x="Percent change in urban population

  theme(axis.text.y = element_blank(), axis.ticks = element_blank())
```



```
        #this line above removes ticks and text on y axis
```

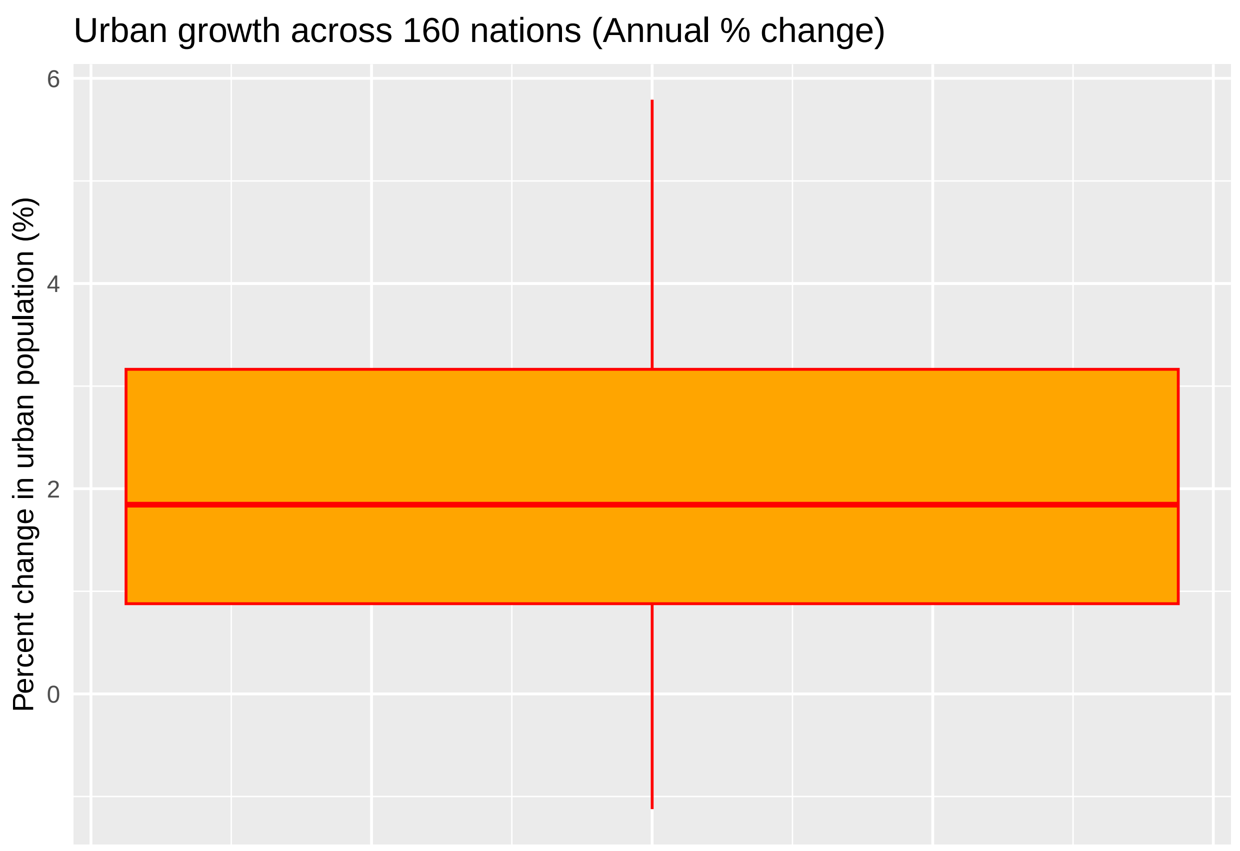
Okay, you have a simple box plot. You could also turn it on its side with coord_flip().I will need to change the axis.text.y to axis.text.x.

```
ggplot(worldbank.df, aes(urban_growth))+
geom_boxplot()+
  labs(title="Urban growth across 160 nations (Annual % change)", x="Percent change in urban population

  theme(axis.text.x = element_blank(), axis.ticks = element_blank()) +
         #this line above removes ticks and text on y axis
  coord_flip()
```



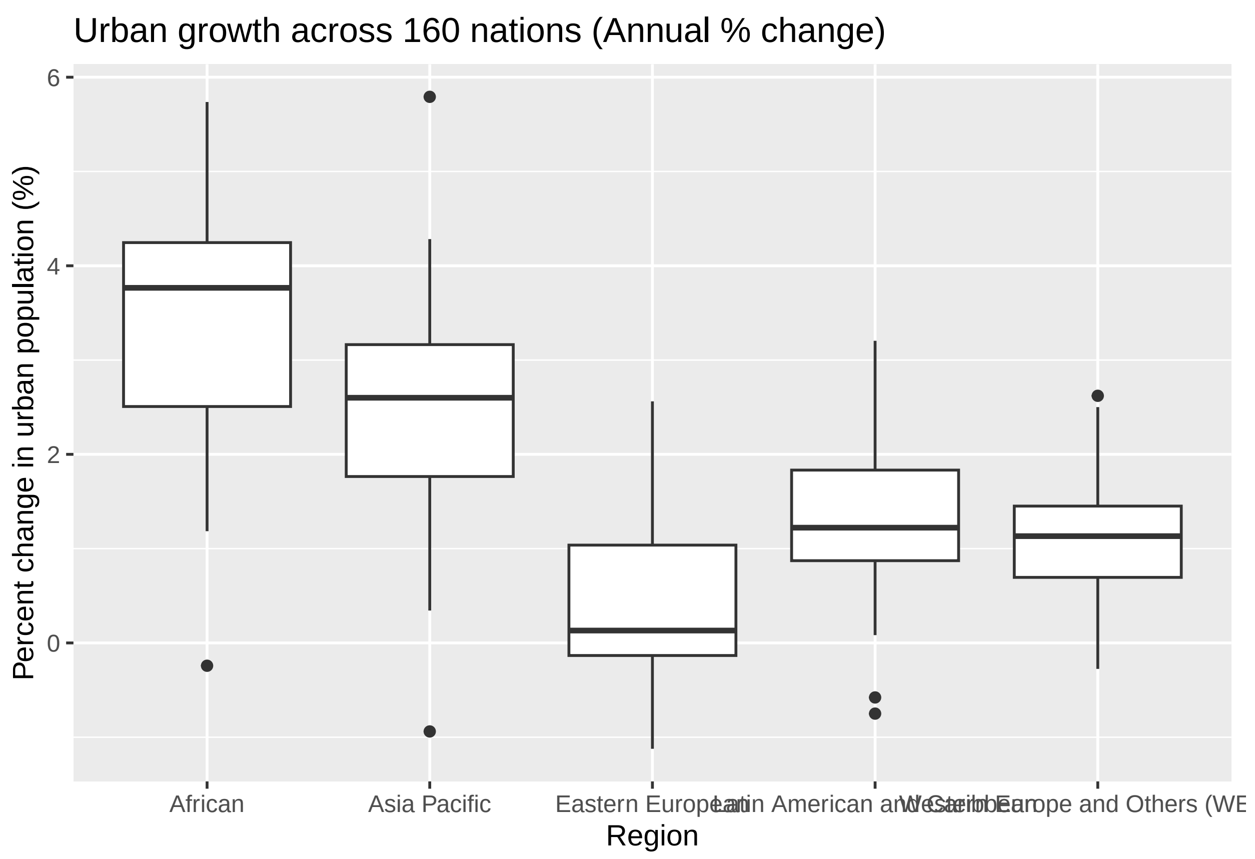
Let’s add some color!

```
ggplot(worldbank.df, aes(urban_growth))+
geom_boxplot(fill="orange", col="red")+
  labs(title="Urban growth across 160 nations (Annual % change)", x="Percent change in urban population

  theme(axis.text.x = element_blank(), axis.ticks = element_blank()) +
         #this line above removes ticks and text on y axis
  coord_flip()
```



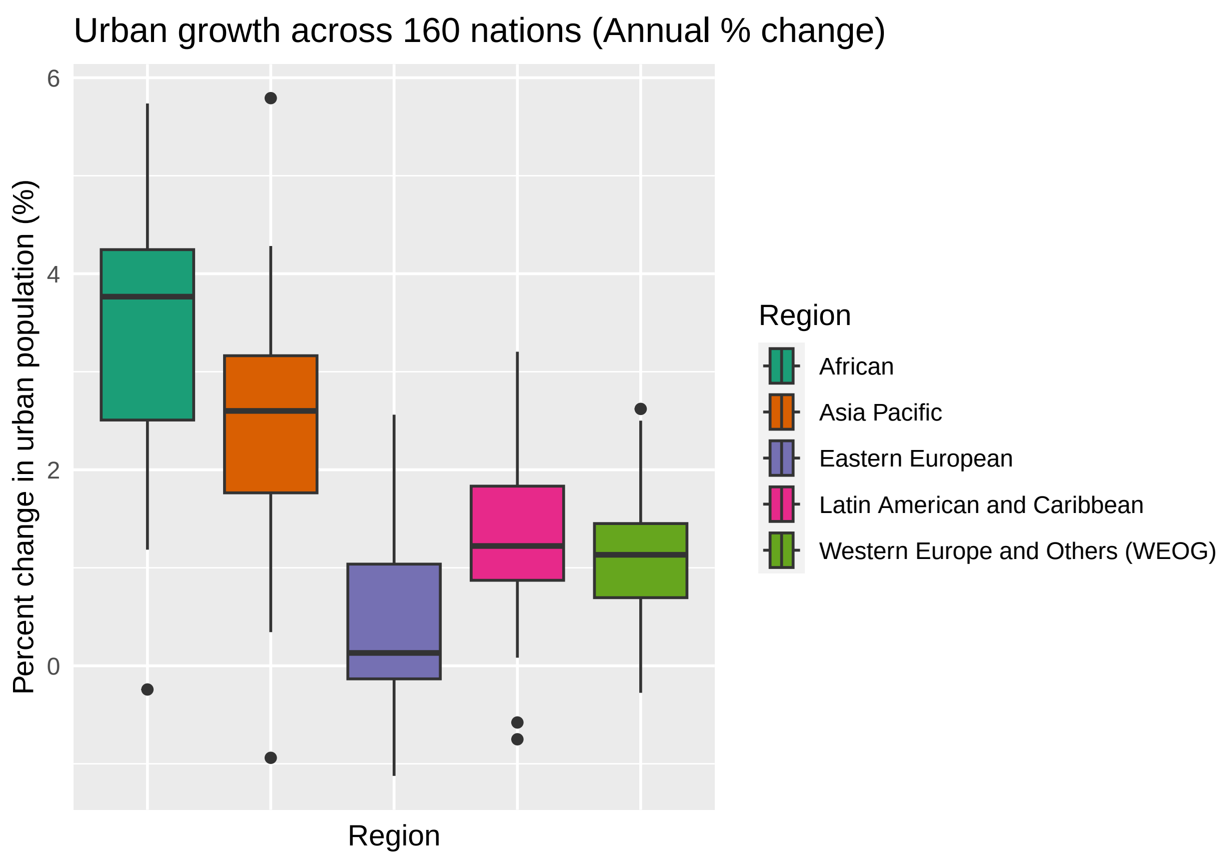
And finally, let's make boxplots by grouping by Region. I have eliminate the theme(axis.text.x) from above, because we want the regions labeled.

```
ggplot(worldbank.df, aes(x=urban_growth, y=Region))+
geom_boxplot()+
  labs(title="Urban growth across 160 nations (Annual % change)", x="Percent change in urban population

  coord_flip()
```



And last, but not least, let's add color. I'm using the Dark2 palette from RColorBrewer here in using the scale_fill_brewer(palette="Dark2").I have also set fill=Region in the aesthetic. We will use ggsave() to save the plot to our working directory.

```r
ggplot(worldbank.df, aes(x=urban_growth, y=Region, fill=Region))+
geom_boxplot()+
  labs(title="Urban growth across 160 nations (Annual % change)", x="Percent change in urban population

  theme(axis.text.x = element_blank(), axis.ticks = element_blank()) +
         #this line above removes ticks and text on y axis
  scale_fill_brewer(palette="Dark2")+
  coord_flip()
```



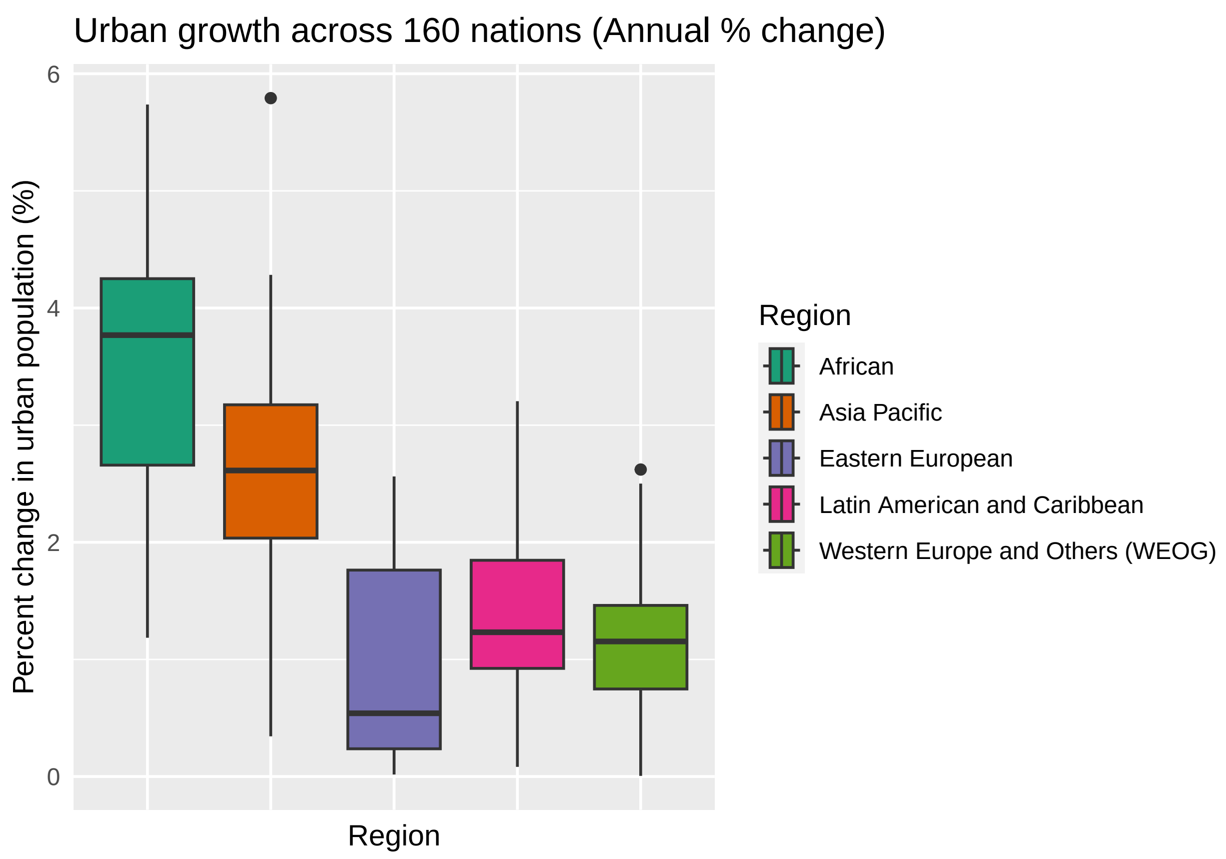
```r
ggsave("finalplot.png") # you can set the arguments to tell it the size you want, with width=, height=,
```

```
## Saving 6.5 x 4.5 in image
```

#11 Bonus: Piping and ggplot

Just like we did above, we can use piping combined with ggplot. Take the boxplot above. What if you only wanted to use data where urban_growth >0 . A simple way to do this is through piping and the filter command from the dplyr package (hint load that package into r if you don't already have it)!

```r
## this is whats happening here, we are piping -- we establish data set, move to filter "urban_growth"

worldbank.df %>% filter(urban_growth > 0) %>%
ggplot(aes(x=urban_growth, y=Region, fill=Region))+
geom_boxplot()+
  labs(title="Urban growth across 160 nations (Annual % change)", x="Percent change in urban population

  theme(axis.text.x = element_blank(), axis.ticks = element_blank()) +
         #this line above removes ticks and text on y axis
  scale_fill_brewer(palette="Dark2")+
  coord_flip()
```



#12 T Tests in R

the idea of a t statistic is simple. it explains the probibility of getting an extreame value in the distrobution Okay, we have now practiced wrangling data and made summary statistics tables and plots. Let's answer the questions we posed in lecture. I am going to do all of this with piping, but note there are also ways to do this in base r.

##12.1 One sample t test The one-sample t-test, also known as the single-parameter t test or single-sample t-test, is used to compare the mean of one sample to a known standard (or theoretical / hypothetical) mean. Lets ask does the mean weight of our sampled bsb differ from the historical average in 1950?

Pretend we sample a bunch of black sea bass weights and got these values - first make a dataframe of the values

```
## establishing the data frame
bsb_df <- data.frame(name = c("B_1","B_2","B_3","B_4","B_5","B_6","B_7","B_8","B_9","B_10"),
                 weight = c(18900,19500,23100,20100,20300,23400,20900,17500,18600,19100)
                 )
```

###12.1.1 Summary stats Compute some summary statistics: count (number of subjects), mean and sd (standard deviation)

```
library(rstatix)
bsb_df %>% get_summary_stats(weight, type = "mean_sd")
```

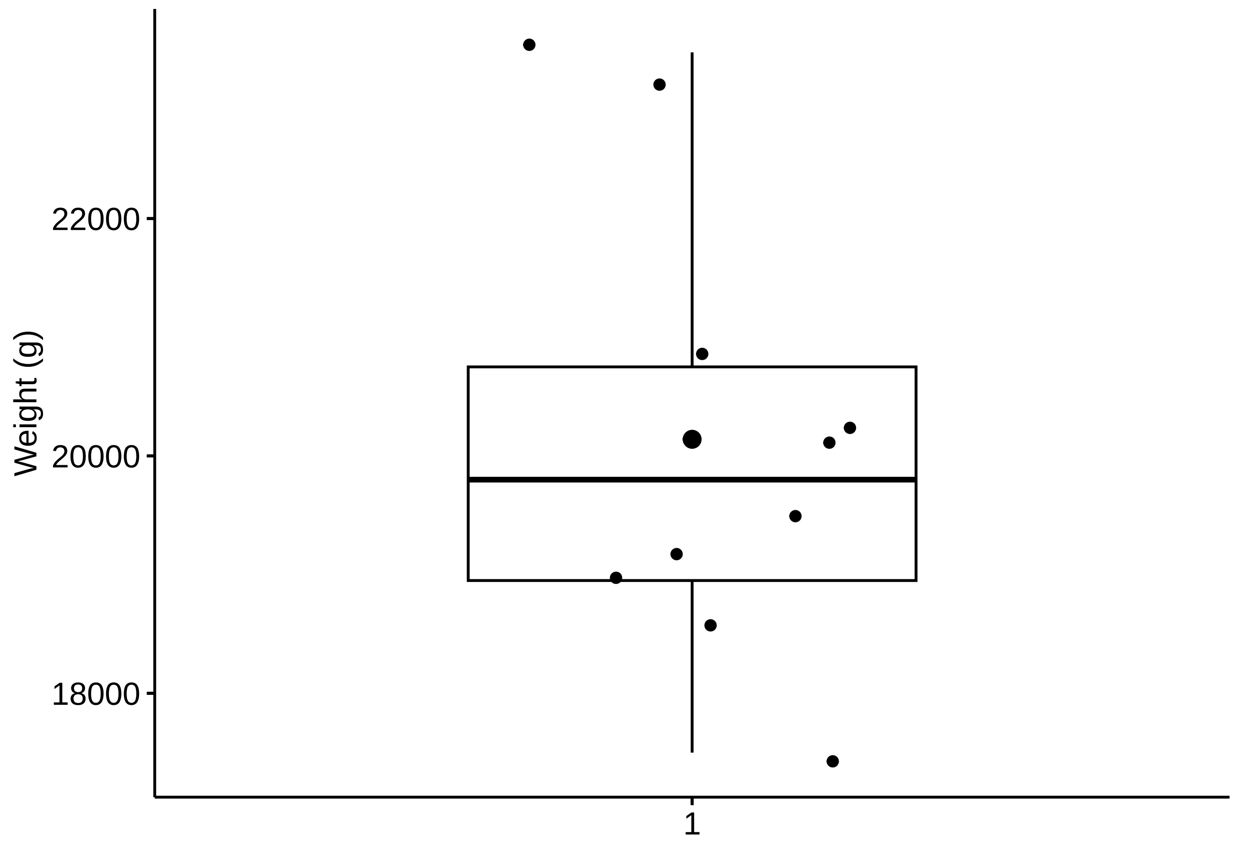
```
## # A tibble: 1 x 4
##   variable     n  mean    sd
##   <fct>    <dbl> <dbl> <dbl>
## 1 weight      10 20140 1896.
```

Create a boxplot to visualize the distribution of bsb weights. Add also jittered points to show individual observations. The big dot represents the mean point. I will show you how to do it using ggpubr which works like ggplot and geom_boxplot (from ggplot2 above)

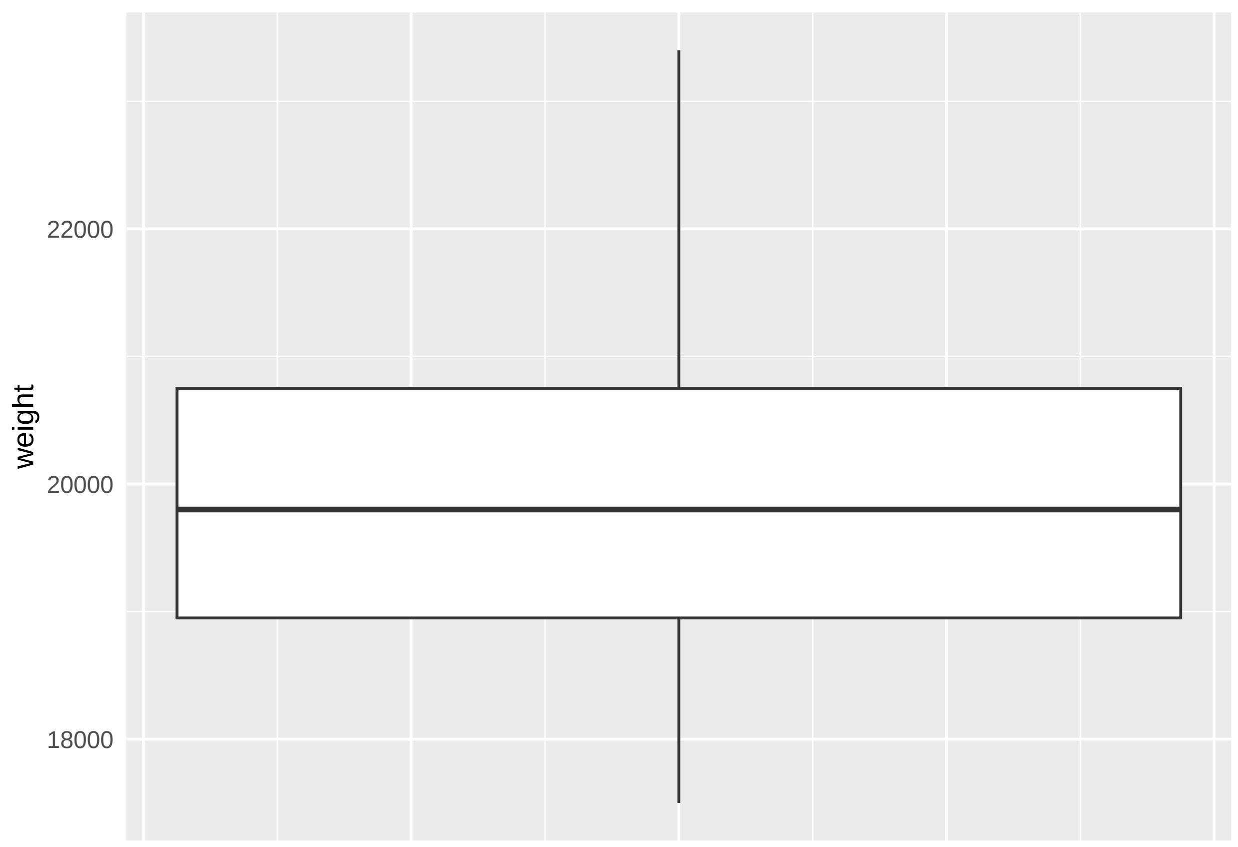
```
## creating a box plot
library(ggpubr)
bxp <- ggboxplot(
  bsb_df$weight, width = 0.5, add = c("mean", "jitter"),
  ylab = "Weight (g)", xlab = FALSE
  )
```

```
## Warning: The `fun.y` argument of `stat_summary()` is deprecated as of ggplot2 3.3.0.
## i Please use the `fun` argument instead.
## i The deprecated feature was likely used in the ggpubr package.
##   Please report the issue at <https://github.com/kassambara/ggpubr/issues>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

## Warning: The `fun.ymin` argument of `stat_summary()` is deprecated as of ggplot2 3.3.0.
## i Please use the `fun.min` argument instead.
## i The deprecated feature was likely used in the ggpubr package.
##   Please report the issue at <https://github.com/kassambara/ggpubr/issues>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

## Warning: The `fun.ymax` argument of `stat_summary()` is deprecated as of ggplot2 3.3.0.
## i Please use the `fun.max` argument instead.
## i The deprecated feature was likely used in the ggpubr package.
##   Please report the issue at <https://github.com/kassambara/ggpubr/issues>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
bxp
```



```
#OR ggplot (like above)
bsb_df %>% ggplot(aes(x = weight))+
geom_boxplot()+ theme(axis.text.x = element_blank(), axis.ticks = element_blank()) +
         #this line above removes ticks and text on y axis
  coord_flip()
```



###12.1.2 Assumptions and preliminary tests The one-sample t-test assumes the following characteristics about the data:

####a) No significant outliers in the data Normality. the data should be approximately normally distributed In this section, we'll perform some preliminary tests to check whether these assumptions are met.

Identify outliers: Outliers can be easily identified using boxplot methods, implemented in the R function identify_outliers() [rstatix package].

```
bsb_df %>% identify_outliers(weight)
```

```
## [1] name       weight     is.outlier is.extreme
## <0 rows> (or 0-length row.names)
```

```
#its blank so no outliers
```

####b) Check normality The normality assumption can be checked by computing the Shapiro-Wilk test. If the data is normally distributed, the p-value should be greater than 0.05.

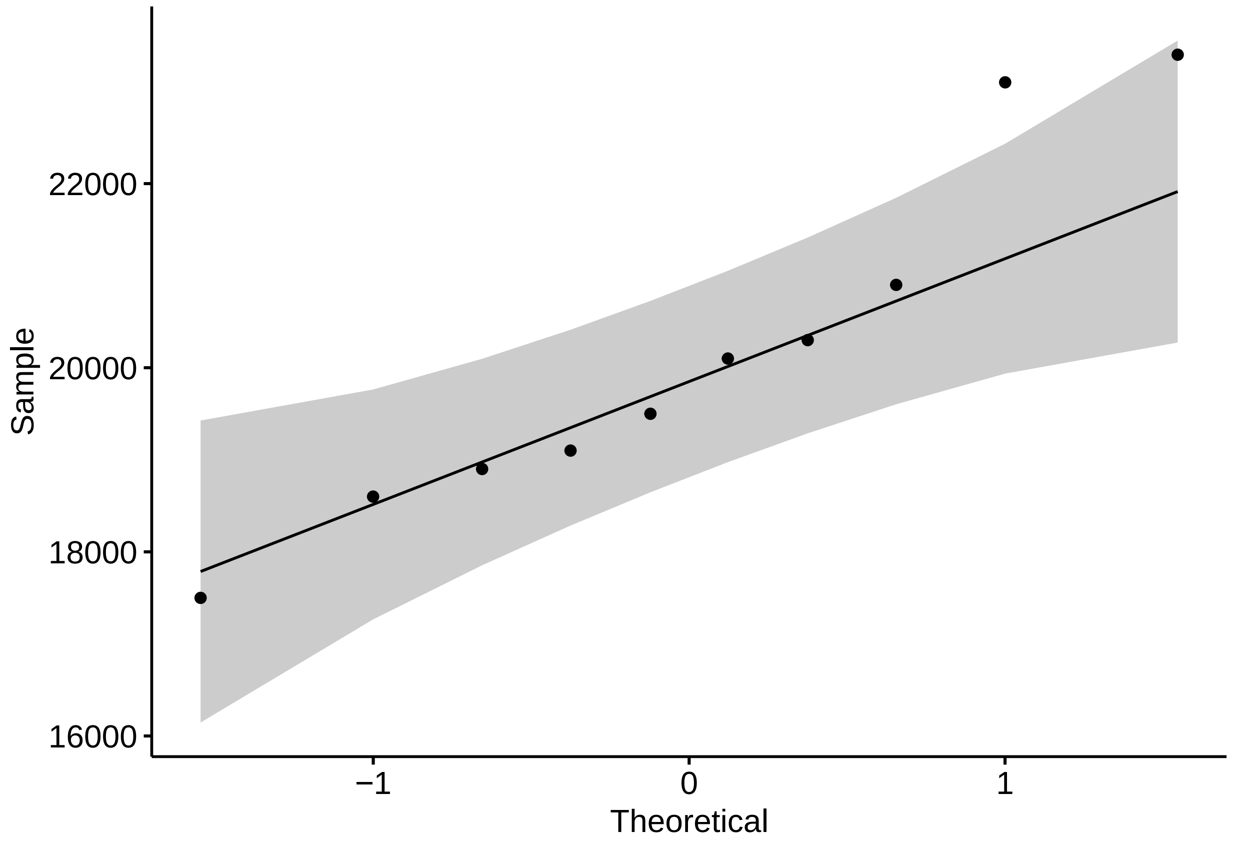
```
## if your data follows a normal distribution and your calculated p-value is greater than 0.05, it mean
## if the P is greater than 5 percent you fail to reject the null.

bsb_df %>% shapiro_test(weight)
```

```
## # A tibble: 1 x 3
##   variable statistic     p
##   <chr>          <dbl> <dbl>
## 1 weight         0.923 0.382
```

From the output, the p-value is greater than the significance level 0.05 indicating that the distribution of the data are not significantly different from the normal distribution. In other words, we can assume the normality. you fail to reject the null

You can also create a QQ plot of the weight data. QQ plot draws the correlation between a given data and the normal distribution.

```
## a qq plot draws the correlation between a given data and the normal distribution.
ggqqplot(bsb_df, x = "weight")
```



All the points fall approximately along the (45-degree) reference line, for each group. So we can assume normality of the data.

Note that, if your sample size is greater than 50, the normal QQ plot is preferred because at larger sample sizes the Shapiro-Wilk test becomes very sensitive even to a minor deviation from normality.

If the data are not normally distributed, it's recommended to use a non-parametric test.

###12.1.3 Conduct the t test We want to know, whether the average weight of the bsb differs from the historical average of 2000g (two-tailed test)?

```
##here we are comparing the 2 means to measure extremes of the data.
stat.test <- bsb_df %>% t_test(weight ~ 1, mu = 2000, detailed = T)
stat.test
```

```
## # A tibble: 1 x 12
##   estimate .y.   group1 group2     n statistic        p    df conf.low conf.high
## *    <dbl> <chr> <chr>  <chr>  <int>     <dbl>    <dbl> <dbl>    <dbl>     <dbl>
## 1    20140 weig~ 1      null ~    10      30.3 2.31e-10     9   18783.    21497.
## # i 2 more variables: method <chr>, alternative <chr>
```

###12.1.4 Effect Size To calculate an effect size, called Cohen's d, for the one-sample t-test you need to divide the mean difference by the standard deviation of the difference, as shown below. Note that, here: sd(x-mu) = sd(x).

Cohen's d formula:

d = abs(mean(x) - mu)/sd(x), where:

x is a numeric vector containing the data. mu is the mean against which the mean of x is compared (default value is mu = 0).

```
##

bsb_df %>% cohens_d(weight ~ 1, mu = 2000)
```

```
## # A tibble: 1 x 6
##   .y.    group1 group2     effsize     n magnitude
## * <chr>  <chr>  <chr>        <dbl> <int> <ord>
## 1 weight 1      null model    9.57    10 large
```

Recall that, t-test conventional effect sizes, proposed by Cohen J. (1998), are: 0.2 (small effect), 0.5 (moderate effect) and 0.8 (large effect) (Cohen 1998). As the effect size, d, is 9.565 you can conclude that there is a large effect.

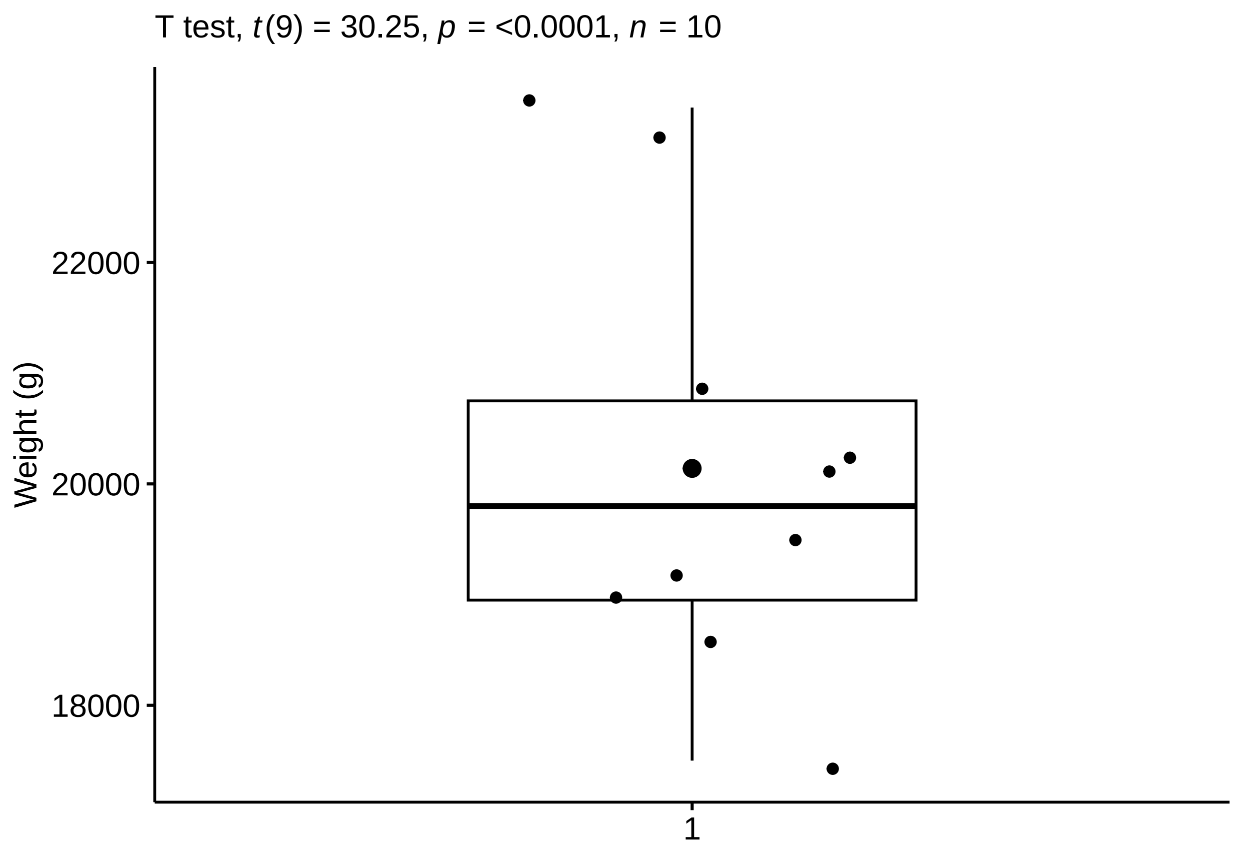
###12.1.5 Report results ####a) writing A one-sample t-test was computed to determine whether the caught bsb weight was different to the historical 1950s mean weight (2000g).

The bsb weight value were normally distributed, as assessed by Shapiro-Wilk's test ($p > 0.05$) and there were no extreme outliers in the data, as assessed by boxplot method.

The measured bsb mean weight (20140 +/- 1.94) was statistically significantly lower than the historical population weight of 2000(t(9) = 30.3, $p < 0.0001$, d = 9.57);

####b) boxplot

```
bxp + labs(
  subtitle = get_test_label(stat.test, detailed = TRUE)
  )
```



##12.2 Two sample (independent T-test) The independent samples t-test (or unpaired samples t-test) is used to compare the mean of two independent groups.

For example, you might want to compare the average weights of black sea bass caught off of North Carolina and Freshwater bass caught in California: two unrelated/independent groups.

The independent samples t-test comes in two different forms:

the standard Student's t-test, which assumes that the variance of the two groups are equal. the Welch's t-test, which is less restrictive compared to the original Student's test. This is the test where you do not assume that the variance is the same in the two groups, which results in the fractional degrees of freedom.

Here is our dataset

```
## Welch's t-test, which is less restrictive compared to the original Student's test.
bsb_df2 <- data.frame(id = 1:40,
                     group = rep(c("BSB", "Fresh"), each = 20),
                     weight = c(1970.748,2065.755,2117.404,1897.875,2075.464,2080.388,2011.217,2013.016
```

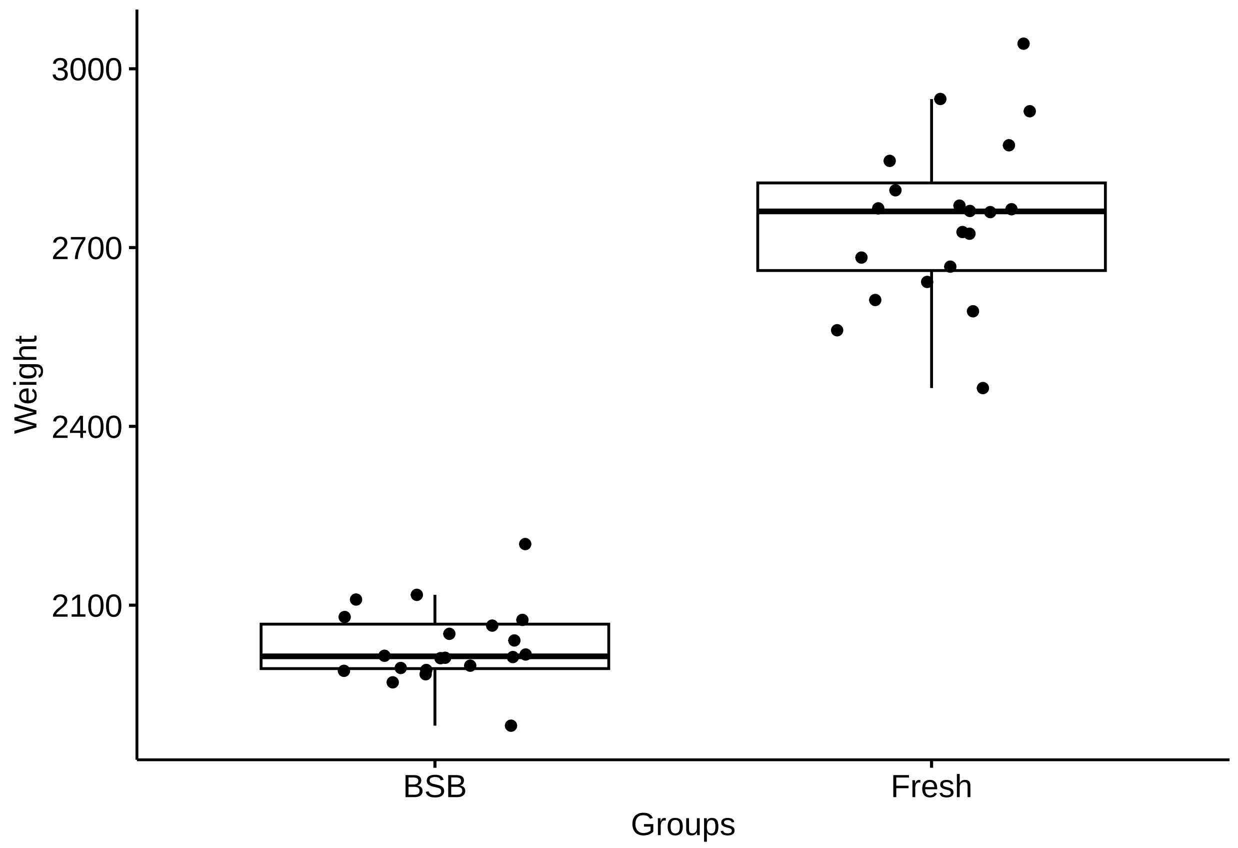
###12.2.1 Summary stats Compute some summary statistics: count (number of subjects), mean and sd (standard deviation)

```
bsb_df2 %>%
  group_by(group) %>%
  get_summary_stats(weight, type = "mean_sd")
```

```
## # A tibble: 2 x 5
##   group variable     n  mean    sd
##   <chr> <fct>    <dbl> <dbl> <dbl>
## 1 BSB   weight      20 2032.  64.9
## 2 Fresh weight      20 2746. 139.
```

Visualize the data using box plots. Plot weight by groups.

```
bxp <- ggboxplot(
  bsb_df2, x = "group", y = "weight",
  ylab = "Weight", xlab = "Groups", add = "jitter"
  )
bxp
```



###12.2.2 Assumptions and preliminary tests The two-samples independent t-test assume the following characteristics about the data:

Independence of the observations. Each subject should belong to only one group. There is no relationship between the observations in each group. No significant outliers in the two groups Normality. the data for each group should be approximately normally distributed. Homogeneity of variances. the variance of the outcome variable should be equal in each group. In this section, we'll perform some preliminary tests to check whether these assumptions are met.

####a) Identify outliers by groups

```
bsb_df2 %>%
  group_by(group) %>%
  identify_outliers(weight)
```

```
## # A tibble: 2 x 5
##   group    id weight is.outlier is.extreme
##   <chr> <int>  <dbl> <lgl>      <lgl>
## 1 BSB      20  2203. TRUE       FALSE
## 2 Fresh    31  3042. TRUE       FALSE
```

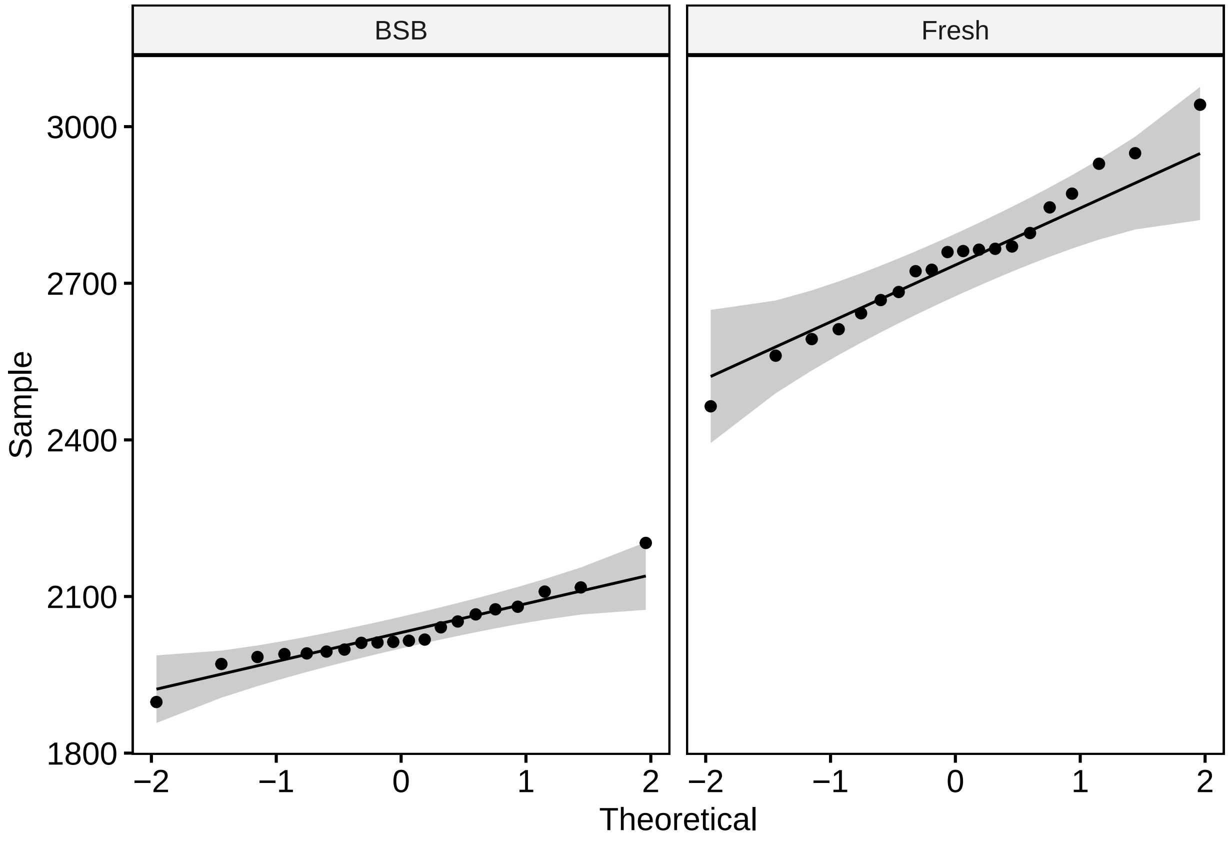
There were no extreme outliers.

####b) check normality by groups

```
## pick 123 of what --> bsb_df2 variables -->grouping ---> shapiro test
set.seed(123)
bsb_df2 %>%
  group_by(group) %>%
  shapiro_test(weight)
```

```
## # A tibble: 2 x 4
##   group variable statistic     p
##   <chr> <chr>        <dbl> <dbl>
## 1 BSB   weight       0.938 0.224
## 2 Fresh weight       0.986 0.989
```

```
# Draw a qq plot by group
ggqqplot(bsb_df2, x = "weight", facet.by = "group")
```



From the output above, we can conclude that the data of the two groups are normally distributed.

####c) Check the equality of variances This can be done using the Levene's test. If the variances of groups are equal, the p-value should be greater than 0.05.

```
bsb_df2 %>% levene_test(weight ~ group)
```

```
## Warning in leveneTest.default(y = y, group = group, ...): group coerced to
## factor.
```

```
## # A tibble: 1 x 4
##     df1   df2 statistic      p
##   <int> <int>     <dbl>  <dbl>
## 1     1    38      6.12 0.0180
```

The p-value of the Levene's test is significant, suggesting that there is a significant difference between the variances of the two groups. Therefore, we'll use the Weltch t-test, which doesn't assume the equality of the two variances.

###12.2.3 Compute the t test We want to know, whether the average weights are different between groups.

Recall that, by default, R computes the Weltch t-test, which is the safer one:

```
stat.test <- bsb_df2 %>%
  t_test(weight ~ group, detailed = T) %>%
  add_significance()
stat.test
```

```
## # A tibble: 1 x 16
##   estimate estimate1 estimate2 .y.    group1 group2    n1    n2 statistic
##      <dbl>     <dbl>     <dbl> <chr>  <chr>  <chr>  <int> <int>     <dbl>
## 1    -714.     2032.     2746. weight BSB    Fresh     20    20     -20.8
## # i 7 more variables: p <dbl>, df <dbl>, conf.low <dbl>, conf.high <dbl>,
## #   method <chr>, alternative <chr>, p.signif <chr>
```

If you want to assume the equality of variances (Student t-test), specify the option var.equal = TRUE:

###12.2.4 Effect size Cohen's d for Student t-test This effect size is calculated by dividing the mean difference between the groups by the pooled standard deviation.

Cohen's d formula:

d = (mean1 - mean2)/pooled.sd, where:

pooled.sd is the common standard deviation of the two groups. pooled.sd = sqrt([var1*(n1-1) + var2*(n2-1)]/[n1 + n2 -2]); var1 and var2 are the variances (squared standard deviation) of group1 and 2, respectively. n1 and n2 are the sample counts for group 1 and 2, respectively. mean1 and mean2 are the means of each group, respectively.

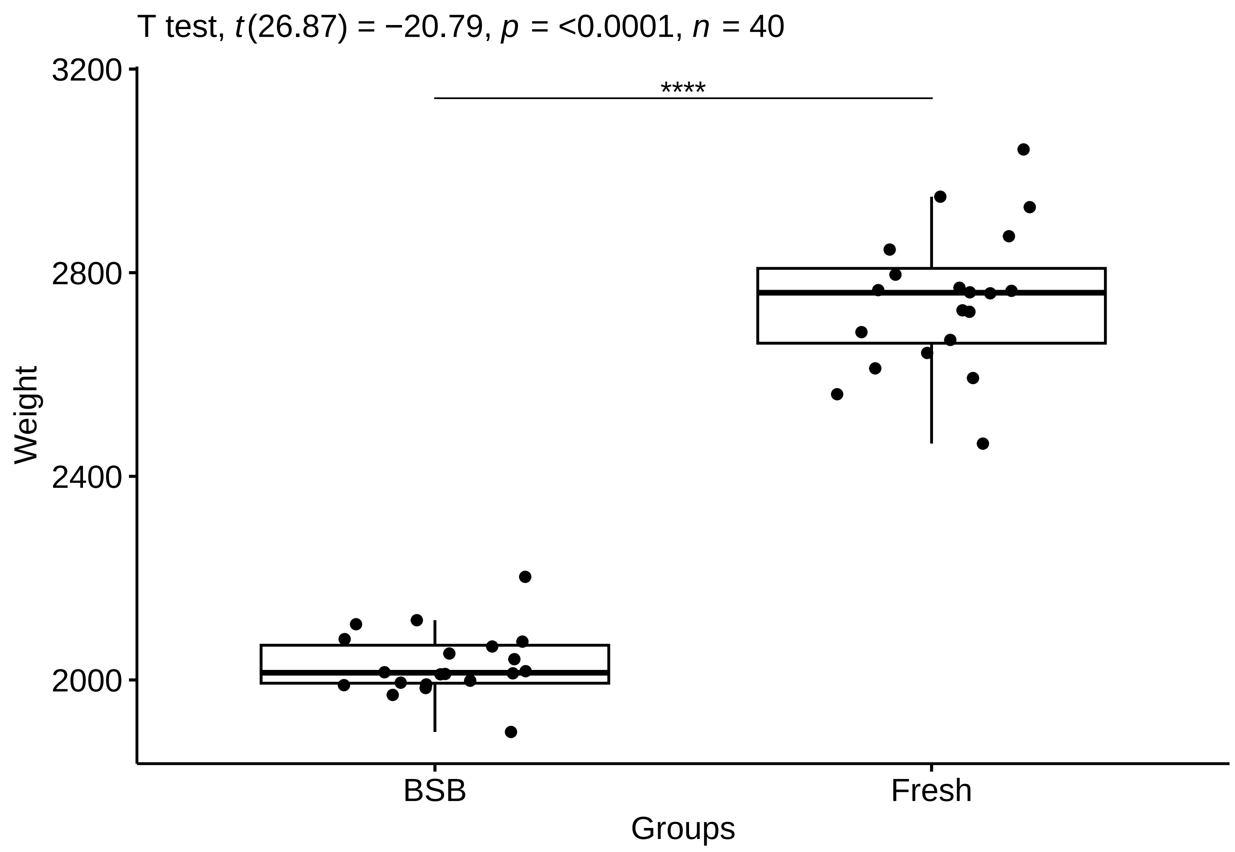
```
bsb_df2 %>%  cohens_d(weight ~ group, var.equal = FALSE)
```

```
## # A tibble: 1 x 7
##   .y.    group1 group2 effsize    n1    n2 magnitude
## * <chr>  <chr>  <chr>    <dbl> <int> <int> <ord>
## 1 weight BSB    Fresh    -6.57    20    20 large
```

There is a large effect size, d = 6.57.

###12.2.5 Report results

```
stat.test <- stat.test %>% add_xy_position(x = "group")
bxp +
  stat_pvalue_manual(stat.test, tip.length = 0) +
  labs(subtitle = get_test_label(stat.test, detailed = TRUE))
```



##12.3 Paired samples t-test The paired sample t-test is used to compare the means of two related groups of samples. Put into another words, it's used in a situation where you have two pairs of values measured for the same samples.

Lets say we did an expirament on black sea bass in tanks and we want to compare the average weight of bsb before and after treatment

```
bsb_df3 <- data.frame(before = c(187.2,194.2,231.7,200.5,201.7,235.0,208.7,172.4,184.6,189.6), after =
```

Transform into long data: gather the before and after values in the same column

```
bsb_df3.long <- bsb_df3 %>%
  gather(key = "group", value = "weight", before, after)
head(bsb_df3.long, 3)
```

```
##   id  group weight
## 1  1 before  187.2
## 2  2 before  194.2
## 3  3 before  231.7
```

###12.3.1 Summary stats

```
bsb_df3.long %>%
  group_by(group) %>%
  get_summary_stats(weight, type = "mean_sd")
```

```
## # A tibble: 2 x 5
##   group  variable     n  mean    sd
##   <chr>  <fct>    <dbl> <dbl> <dbl>
## 1 after  weight      10  400.  30.1
## 2 before weight      10  201.  20.0
```

###12.3.2 Assumptions and preliminary tests The paired samples t-test assume the following characteristics about the data:

the two groups are paired. In our example, this is the case since the data have been collected from measuring twice the weight of the same bsb No significant outliers in the difference between the two related groups Normality. the difference of pairs follow a normal distribution.

First, start by computing the difference between groups:

```
bsb_diff <- bsb_df3 %>% mutate(differences = before - after)
head(bsb_diff, 3)
```

```
##   before after id differences
## 1  187.2 429.5  1      -242.3
## 2  194.2 404.4  2      -210.2
## 3  231.7 405.6  3      -173.9
```

identify outliers

```
bsb_diff %>% identify_outliers(differences)
```

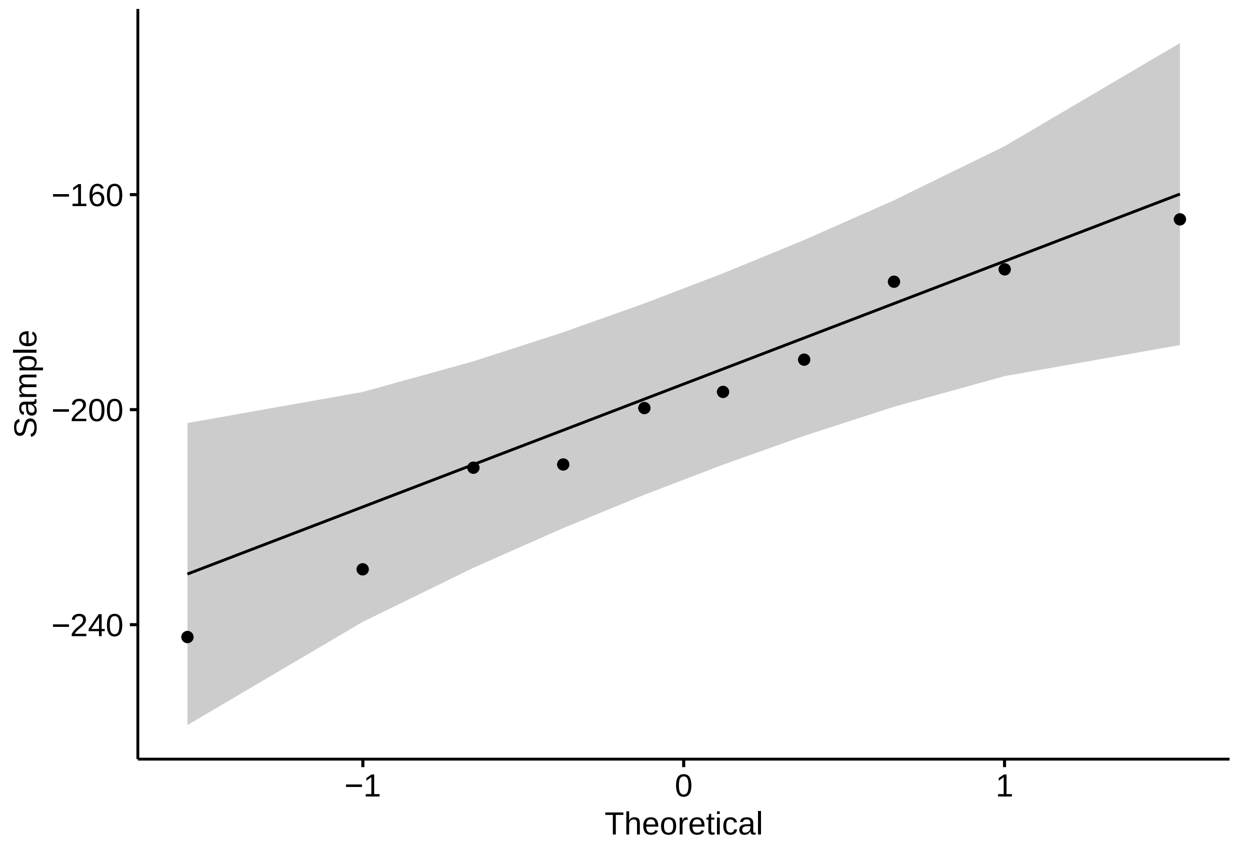
```
## [1] before      after       id          differences is.outlier  is.extreme
## <0 rows> (or 0-length row.names)
```

Check normality assumption

```
bsb_diff %>% shapiro_test(differences)
```

```
## # A tibble: 1 x 3
##   variable    statistic     p
##   <chr>           <dbl> <dbl>
## 1 differences     0.968 0.867
```

```
# QQ plot for the difference
ggqqplot(bsb_diff, "differences")
```



###12.3.3 Compute the t test

```r
stat.test <- bsb_df3.long  %>%
  t_test(weight ~ group, paired = TRUE, detailed = T) %>%
  add_significance()
stat.test
```

```
## # A tibble: 1 x 14
##   estimate .y.    group1 group2    n1    n2 statistic          p    df conf.low
##      <dbl> <chr>  <chr>  <chr>  <int> <int>     <dbl>      <dbl> <dbl>    <dbl>
## 1     199. weight after  before    10    10      25.5    1.04e-9     9     182.
## # i 4 more variables: conf.high <dbl>, method <chr>, alternative <chr>,
## #   p.signif <chr>
```

###12.3.4 Effect size The effect size for a paired-samples t-test can be calculated by dividing the mean difference by the standard deviation of the difference, as shown below.

Cohen's formula:

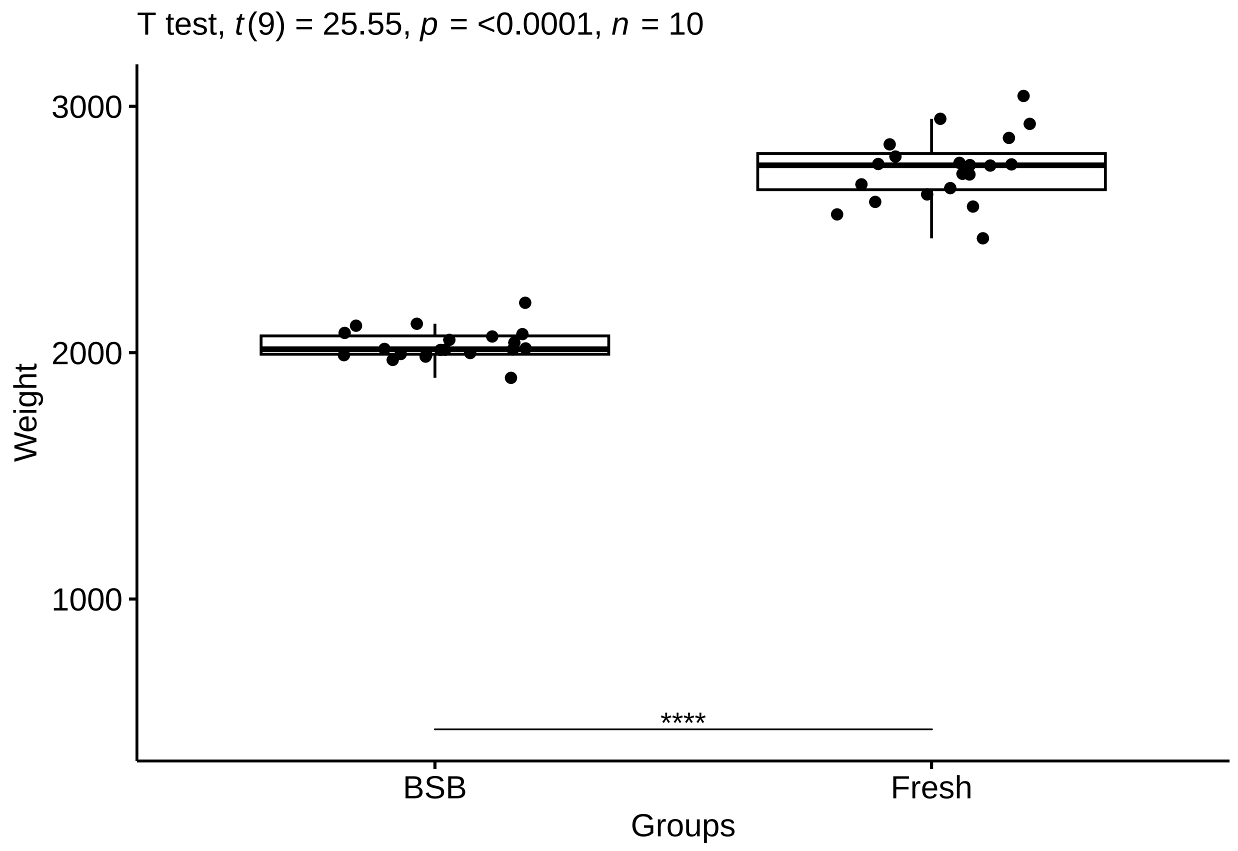
d = mean(D)/sd(D), where D is the differences of the paired samples values.

```r
bsb_df3.long  %>% cohens_d(weight ~ group, paired = TRUE)
```

```
## # A tibble: 1 x 7
##   .y.    group1 group2 effsize    n1    n2 magnitude
## * <chr>  <chr>  <chr>    <dbl> <int> <int> <ord>
## 1 weight after  before    8.08    10    10 large
```

###12.3.5 Report We could report the result as follow: The average weight of bsb was significantly increased after treatment, t(9) = 25.5, p < 0.0001, d = 8.07.

```
stat.test <- stat.test %>% add_xy_position(x = "group")
bxp +
  stat_pvalue_manual(stat.test, tip.length = 0) +
  labs(subtitle = get_test_label(stat.test, detailed= TRUE))
```



#13 NonParametric Tests in R The Wilcoxon test is a non-parametric alternative to the t-test for comparing two means. It's particularly recommended in a situation where the data are not normally distributed.

Like the t-test, the Wilcoxon test comes in two forms, one-sample and two-samples. They are used in more or less the exact same situations as the corresponding t-tests.

Note that, the sample size should be at least 6. Otherwise, the Wilcoxon test cannot become significant.

In this section, you will learn how to compute the different types of Wilcoxon tests in R, including:

One-sample Wilcoxon signed rank test Wilcoxon rank sum test and Wilcoxon signed rank test on paired samples Check Wilcoxon test assumptions Calculate and report Wilcoxon test effect size (r value). The effect size r is calculated as Z statistic divided by the square root of the sample size (N) (Z/sqrt(N)). The Z value is extracted from either coin::wilcoxsign_test() (case of one- or paired-samples test) or coin::wilcox_test() (case of independent two-samples test).

Note that N corresponds to the total sample size for independent-samples test and to the total number of pairs for paired samples test. The r value varies from 0 to close to 1. The interpretation values for r commonly in published literature are: 0.10 - < 0.3 (small effect), 0.30 - < 0.5 (moderate effect) and >= 0.5 (large effect).

We'll use the pipe-friendly function wilcox_test() [rstatix package].

##13.1 One-Sample Wilcoxon signed rank test Lets go back to our original quesiton and ask does the mean weight of our sampled bsb differ from the historical average in 1950?

Pretend we sample a bunch of black sea bass weights and got these values - that are now not normally distributed

```
bsb_df <- data.frame(name = c("B_1","B_2","B_3","B_4","B_5","B_6","B_7","B_8","B_9","B_10"),
                 weight = c(16900,16800,20100,17600,16300,20400,17100,21500,17600,17200)
                 )

bsb_df %>% shapiro_test(weight) #just to check if they are normally distributed
```

```
## # A tibble: 1 x 3
##   variable statistic      p
##   <chr>        <dbl>  <dbl>
## 1 weight       0.824 0.0284
```

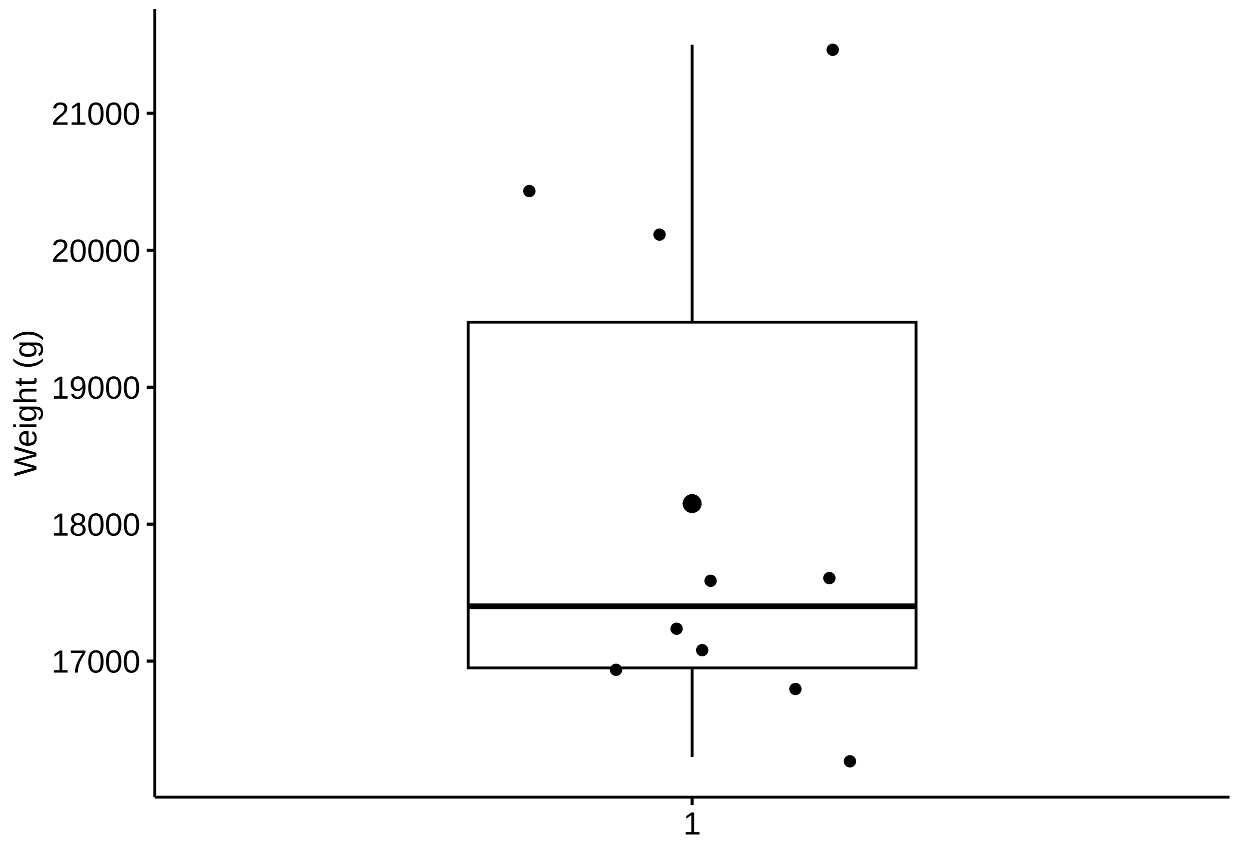
###13.1.1 Summary statistics Compute the median and the interquartile range (IQR):

```
bsb_df %>% get_summary_stats(weight, type = "median_iqr")
```

```
## # A tibble: 1 x 4
##   variable     n median   iqr
##   <fct>    <dbl>  <dbl> <dbl>
## 1 weight      10  17400  2525
```

Create a box plot to visualize the distribution of bsb weights. Add also jittered points to show individual observations. The big dot represents the mean point.

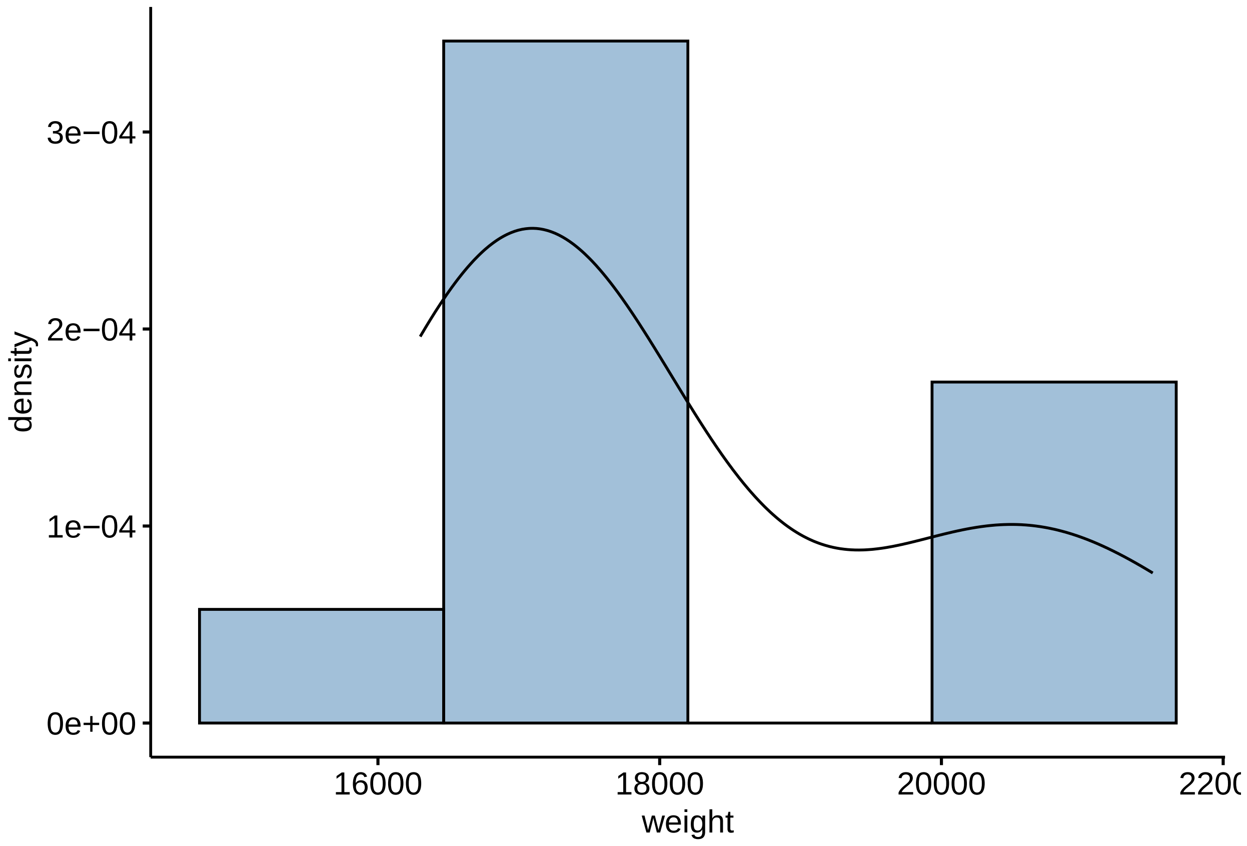
```
bxp <- ggboxplot(
  bsb_df$weight, width = 0.5, add = c("mean", "jitter"),
  ylab = "Weight (g)", xlab = FALSE
  )
bxp
```



###13.1.2 Assumptions and preliminary tests The Wilcoxon signed-rank test assumes that the data are distributed symmetrically around the median. This can be checked by visual inspection using histogram and density distribution.

Create a histogram: As we have only 10 individuals in our data, we specify the option bins = 4 instead of 30 (default).

```
gghistogram(bsb_df, x = "weight", y = "..density..",
            fill = "steelblue",bins = 4, add_density = TRUE)
```



From the plot above, it can be seen that the weight data are approximately symmetrical (you should not expect them to be perfect, particularly when you have smaller numbers of samples in your study). Therefore, we can use the Wilcoxon signed-rank test to analyse our data.

Note that, in the situation where your data is not symmetrically distributed, you could consider performing a sign test, instead of running the Wilcoxon signed-rank test.

The sign test does not make the assumption of a symmetrically-shaped distribution. However, it will most likely be less powerful compared to the Wilcoxon test.

We can also check this assumption using symmetry.test function from the lawstat package. This function replaces the mean and standard deviation in the classic measure of asymmetry by corresponding robust estimators; the median and mean deviation from the median.

Statistical hypothesis: – Null hypothesis: the data is symmetric – Alternative hypothesis: the data is not symmetric

```r
library(lawstat)
bsb.weight <- bsb_df$weight
symmetry.test(bsb.weight, boot = FALSE)
```

```
##
##  Symmetry test by Miao, Gel, and Gastwirth (2006)
##
## data:  bsb.weight
## Test statistic = 1.9417, p-value = 0.05218
## alternative hypothesis: the distribution is asymmetric.
```

the p-value is not significant (p = 0.052), so we accept the null hypothesis: Our data is symmetrically distributed around the median.

###13.1.3 Computation We want to know, whether the average weight of the bsb differs from the historical average of 20000g (two-tailed test)?

```
stat.test <- bsb_df %>% wilcox_test(weight ~ 1, mu = 20000)
stat.test
```

```
## # A tibble: 1 x 6
##   .y.    group1 group2         n statistic      p
## * <chr>  <chr>  <chr>      <int>     <dbl>  <dbl>
## 1 weight 1      null model    10         6 0.0322
```

Note that, to compute one-sided wilcoxon test, you can specify the option alternative, which possible values can be "greater", "less" or "two.sided".

####Effect Size We'll use the R function wilcox_effsize() [rstatix]. It requires the coin package for computing the Z statistic.

```
bsb_df %>%  wilcox_effsize(weight ~ 1, mu = 2000)
```

```
## # A tibble: 1 x 6
##   .y.    group1 group2     effsize     n magnitude
## * <chr>  <chr>  <chr>        <dbl> <int> <ord>
## 1 weight 1      null model   0.887    10 large
```

A large effect size is detected, r = 0.89.

###13.1.4 Report results We could report the result as follow:

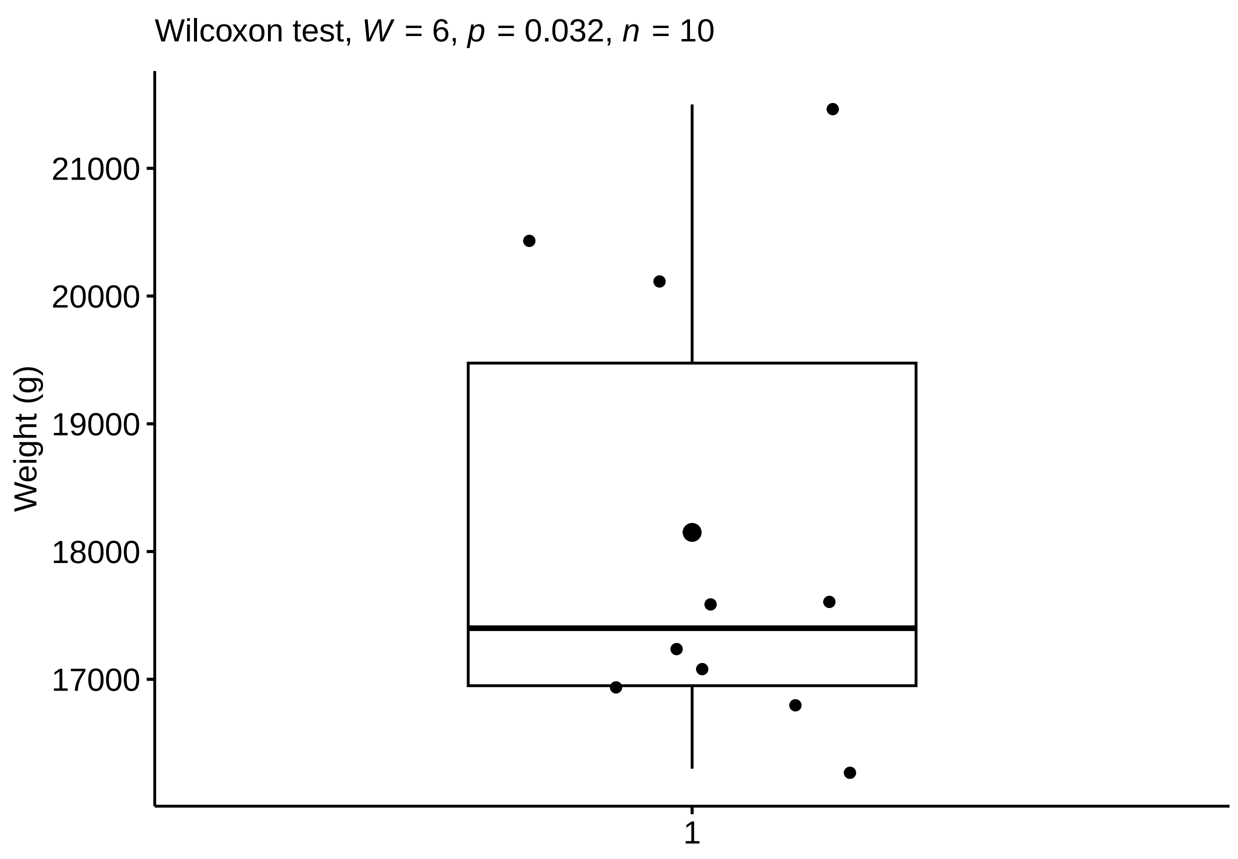
A Wilcoxon signed-rank test was computed to assess whether the caught bcb median weight was different to the historical median weight (2000g).

The bsb weight value were approximately symmetrically distributed, as assessed by a histogram with superimposed density curve.

The measured bsb median weight (17400) was statistically significantly lower than the historical median weight 2000g (p = 0.03, effect size r = 0.89).

Create a box plot with p-value:

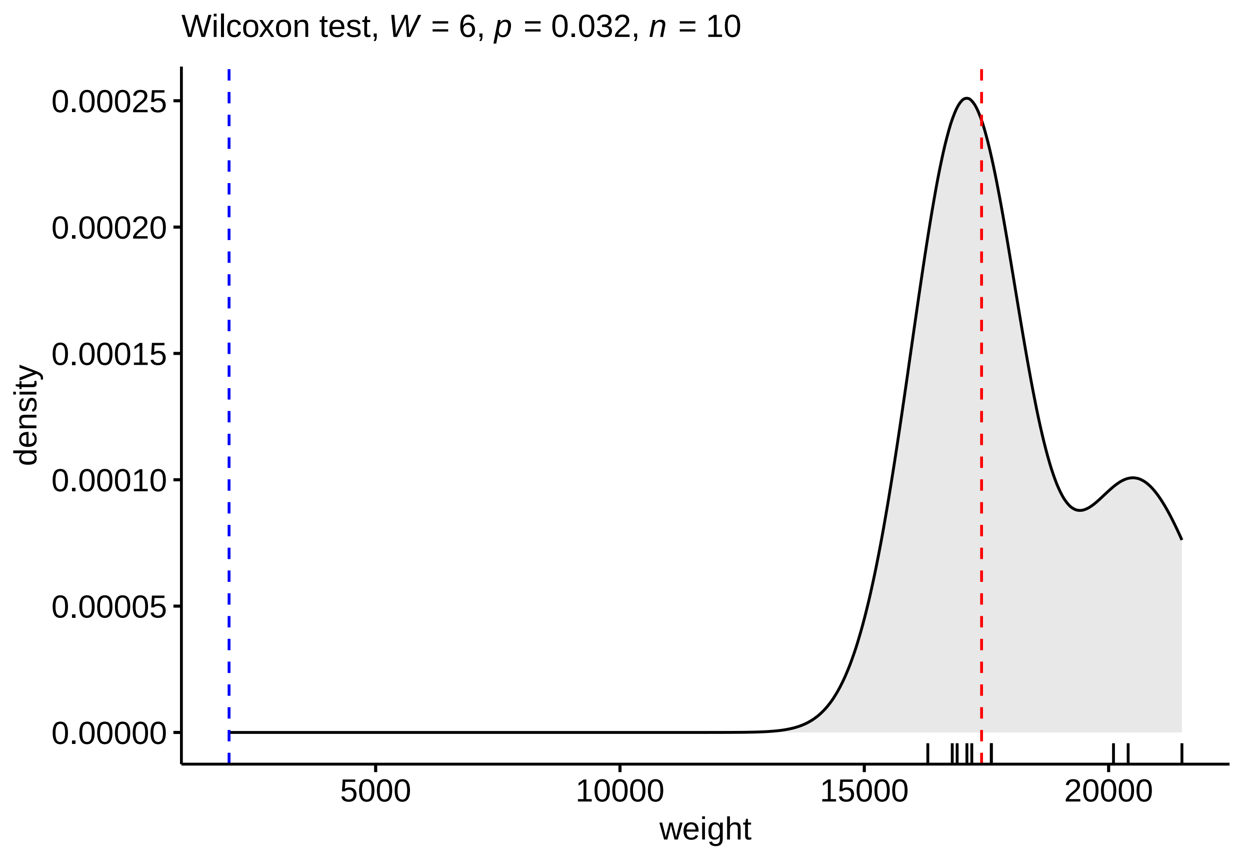
```
bxp +
  labs(subtitle = get_test_label(stat.test, detailed = TRUE))
```



Create a density plot with p-value:

Red line corresponds to the observed median Blue line corresponds to the theoretical median

```
ggdensity(bsb_df, x = "weight", rug = TRUE, fill = "lightgray") +
  stat_central_tendency(type = "median", color = "red", linetype = "dashed") +
  geom_vline(xintercept = 2000, color = "blue", linetype = "dashed") +
  labs(subtitle = get_test_label(stat.test, detailed = TRUE))
```



##13.2 Wilcoxon rank sum test The Wilcoxon rank sum test is a non-parametric alternative to the independent two samples t-test for comparing two independent groups of samples, in the situation where the data are not normally distributed.

Synonymous: Mann-Whitney test, Mann-Whitney U test, Wilcoxon-Mann-Whitney test and two-sample Wilcoxon test.

If we were to look at our sample from earlier of bsb vs freshwater bass

```
bsb_df2 <- data.frame(id = 1:40,
                      group = rep(c("BSB", "Fresh"), each = 20),
                      weight = c(1970.748,2065.755,2117.404,1897.875,2075.464,2080.388,2011.217,2013.016
```

get summary statistics

```
bsb_df2 %>%
  group_by(group) %>%
  get_summary_stats(weight, type = "median_iqr")
```

```
## # A tibble: 2 x 5
##   group variable     n median   iqr
##   <chr> <fct>    <dbl>  <dbl> <dbl>
## 1 BSB   weight      20  2014.  74.6
## 2 Fresh weight      20  2761. 147.
```

compute the test

```r
stat.test <- bsb_df2 %>%
  wilcox_test(weight ~ group) %>%
  add_significance()
stat.test
```

```
## # A tibble: 1 x 8
##   .y.    group1 group2    n1    n2 statistic        p p.signif
##   <chr>  <chr>  <chr>  <int> <int>     <dbl>    <dbl> <chr>
## 1 weight BSB    Fresh     20    20         0 1.45e-11 ****
```

calculate effect size

```r
bsb_df2 %>% wilcox_effsize(weight ~ group)
```

```
## # A tibble: 1 x 7
##   .y.    group1 group2 effsize    n1    n2 magnitude
## * <chr>  <chr>  <chr>    <dbl> <int> <int> <ord>
## 1 weight BSB    Fresh    0.855    20    20 large
```

##13.3 Wilcoxon Ranked Sign test on Paired Samples The Wilcoxon signed rank test on paired sample is a non-parametric alternative to the paired samples t-test for comparing paired data. It's used when the data are not normally distributed.

The test assumes that differences between paired samples should be distributed symmetrically around the median.

Note that, in the situation where your data is not symmetrically distributed, you could consider performing a sign test, instead of running the Wilcoxon signed-rank test.

The sign test does not make the assumption of a symmetrically-shaped distribution. However, it will most likely be less powerful compared to the Wilcoxon test.

computing the test is similar to the above

```r
stat.test <- bsb_df3.long  %>%
  wilcox_test(weight ~ group, paired = TRUE) %>%
  add_significance()
stat.test
```

```
## # A tibble: 1 x 8
##   .y.    group1 group2    n1    n2 statistic       p p.signif
##   <chr>  <chr>  <chr>  <int> <int>     <dbl>   <dbl> <chr>
## 1 weight after  before    10    10        55 0.00195 **
```

```r
bsb_df3.long %>%
  wilcox_effsize(weight ~ group, paired = TRUE)
```

```
## # A tibble: 1 x 7
##   .y.    group1 group2 effsize    n1    n2 magnitude
## * <chr>  <chr>  <chr>    <dbl> <int> <int> <ord>
## 1 weight after  before   0.886    10    10 large
```

#14 Log ## Testing for normality

One of the assumptions underlying the t-test are normally distributed populations. As we saw above, our samples did not look normally distributed (especially California). We could test for normality with the Shapiro-Wilk test. The null hypothesis of the Shapiro Wilk Test is that the sample was pulled from a normal population. We can run the Shapiro Wilk test [function: shapiro.test()] for each of the set of observations of Arizona and California. To do so, it may be easiest to make a new data frame for each state. The first three lines makes the new data frame that i have called arizona.df. arizona.df will include all Arizona observations on July 1st. Then the shapiro.test() runs the test on the arizona.df.

```
airquality.df<-read.csv("~/Downloads/BIO 591 Coding Files/Labs/Lab 2/Lab 2 Data/airquality-1.csv")
airquality.df <- airquality.df %>% dplyr::filter(State.Name =="Arizona" | State.Name =="California" & Da


arizona.df<-airquality.df %>%
  filter(State.Name =="Arizona") %>%
  filter(Date.Local == "7/1/2019")


shapiro.test(arizona.df$ozone)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  arizona.df$ozone
## W = 0.96567, p-value = 0.2112
```

The Arizona data passes the Shapiro-Wilk test. We do NOT reject the null hypothesis (that samples were pulled from normal distributions). Let's see about the California data. The first three lines develops the data frame for California observations on july 1 and names it california.df. I then run shapiro.test().

```
california.df<-airquality.df %>%
  filter(State.Name =="California") %>%
  filter(Date.Local == "7/1/2019")


shapiro.test(california.df$ozone)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  california.df$ozone
## W = 0.96953, p-value = 0.0008277
```

The California ozone levels are clearly not pulled from a normal distribution (based on the p-value being P<0.01, have strong evidence against the null) –and therefore the normality assumption is likely not met.

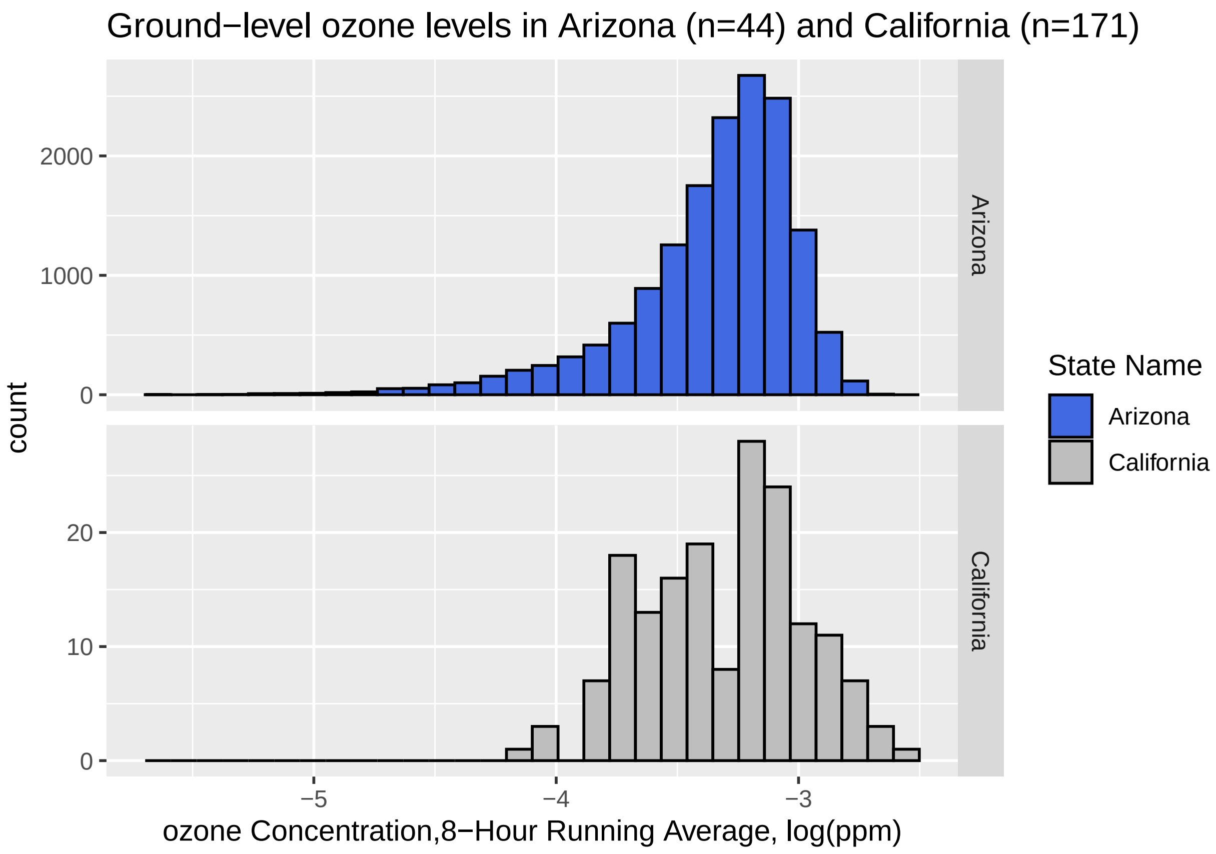
We need to make a new variable in our data frame–this will be the log of ozone. This is easy to do. We will call this new variable log.ozone.

```
airquality.df$log.ozone<-log(airquality.df$ozone)
```

Let's make histograms of the logged data!

```r
p<-ggplot(airquality.df, aes(x=log.ozone, fill=State.Name))+
geom_histogram(col="black")+
scale_fill_manual(values=c("royalblue", "gray"))

p + facet_grid(State.Name ~ ., scales = "free")+
labs(x="ozone Concentration,8-Hour Running Average, log(ppm)", title="Ground-level ozone levels in Arizo
     fill="State Name")
```

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



Looks much better. To double check, we could run the Shapiro Wilk test on the logged data to see. We now make our assumptions about the populations of the logged ozone.

Finally, we can run the t-test on the logged data.

```r
t.test(log.ozone ~ State.Name, airquality.df)
```

```
##
##  Welch Two Sample t-test
##
## data:  log.ozone by State.Name
## t = -1.9376, df = 174.3, p-value = 0.05428
## alternative hypothesis: true difference in means between group Arizona and group California is not e
## 95 percent confidence interval:
##  -0.0983806541  0.0009063317
## sample estimates:
##    mean in group Arizona mean in group California
##                -3.354796                -3.306059
```

Great. In our original data, the p-value was quite small, $p<0.02$, and now the p-value is around 0.054. Which test do you think is more valid and why?

Confidence Interval on Logged Data

When we have logged data, difference of means becomes the multiplicative factor of the medians. This is because of the log rules: log(A) - log(B) = log(A/B). So we need to back transform the difference in means. So to interpret our difference in means and confidence intervals, we need to first back transform with exp(). We then talk about multiplicative factor of the medians.

```r
exp.mean.diff<- exp(-3.354796 - -3.306059) #I'm using a short cut here with the output of the Welch's t
exp.mean.diff
```

```
## [1] 0.9524316
```

So our back transformed difference in means is 0.95. What does this mean? It means that the MEDIAN ozone levels in Arizona is estimated to be 95% of the median ozone level in California (on July 1st).

Now, we also need to back transform the confidence interval and that becomes the confidence interval for the multiplicative factor of the medians!

```r
lower.exp<-exp(-0.0983806541)
upper.exp<-exp(0.0009063317)

lower.exp
```

```
## [1] 0.9063038
```

```r
upper.exp
```

```
## [1] 1.000907
```

So, finally! the 95% confidence interval for the multiplicative factor of the medians is (0.91, 1.03). We are 95% confidence that this interval includes the true multiplicative factor of the medians! BUT THIS TEST WAS NOT SIGNIFICANT