Statistic Quality & Engineering
Universitas Indonesia

Module 2 :

Data Visualization

Using ggplot2

SQE - 2020

Statistic Quality & Engineering
Universitas Indonesia

Content of this module :

1. Introduction to ggplot2
2. Summarizing with dplyr
3. Example of Data Visualization using Gapminder Data
4. Data Visualization Principles

This is what you will learn in this section...

Section 1 - overview

Section	What you will learn...
1.1	• Introduction • Ggplot Objects
1.2	• Aesthetic Mappings • Layers • Global versus local aesthetic mappings • Scales, labels and Colors
1.3	• Add on packages • Other examples

Introduction

Section 1.1 - Introduction

1. install packages dplyr, ggplot and tidyverse

```
> install.packages("dplyr")
> install.packages("ggplot2")
> install.packages("tidyverse")
```

2. Load those packages, we'll heavily use ggplot2 in this module

```
> library(dplyr)
> library(ggplot2)
> library(tidyverse)
```

3. We will start by loading the dataset

```
> library(dslabs)
> data("murders")
```

Introduction

Section 1.1 - Introduction

1. In ggplot we create graphs by adding layers. Layers can define geometrics, compute summary statistics, define what scales to use or even change styles. To add layers, we use the symbol + . in general. a line of code will look like this:

```
> data %>% ggplot() + LAYER 1 + LAYER 2 + LAYER 3 + …. + LAYER N
```

2. Aesthetic mappings describe how properties of the data connect with features of the graph,such as distance along an axis,size or color. Define aesthetic mappings with this:

```
> aes()
```

aes( ) uses variable names from the object component

3. To create a scatterplot and requires x and y aesthetic mappings we can use this:

```
> geom_point()
```

4. To add text to scatter plot and require x and y and label aesthetic mapping we can use this:

```
> geom_text()
```

&

```
> geom_label()
```

Ggplot Objects

Section 1.1 - Ggplot Objects

The first argument is used to specify what data is associated with this object...

```
> ggplot(data=murders)
```

...Or we can also pipe the data is as the first argument

```
> murders %>% ggplot()
```

...It renders a plot, in this case a blank slate since no geometry has been defined. The only style choice we see is a grey background. But we can assign out plot to an object, for example like this

```
> p <- ggplot (data = murders)
> class (p)
[1] “gg” “ggplot”
```

Aesthetic Mappings

Section 1.2 - Aesthetic Mappings

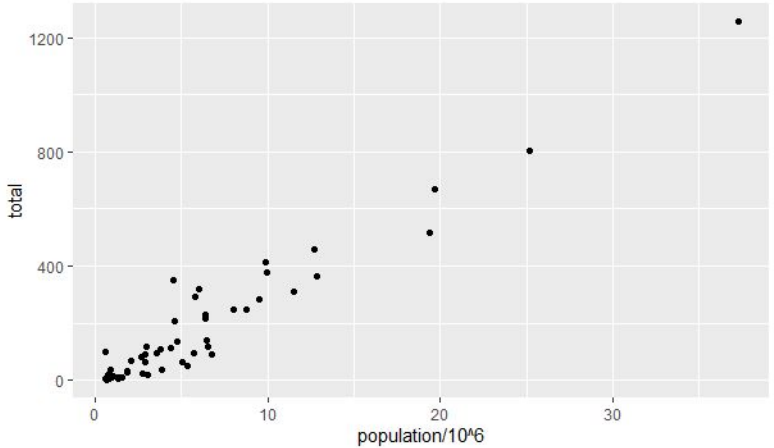
The outcome of the aes function is often used as the argument of a geometry function. This example produces a scatterplot of total murders versus population in millions:

```
> murders %>% ggplot()
 + geom_point(aes(x = population/10^6, y = total))
```

Instead of defining out plot from scratch, we can also add a layer to the p object that was defined above as

```
> p <- ggplot(data = murders)
> p + geom_point(aes(population/10^6,total))
```

Scatter plot Output



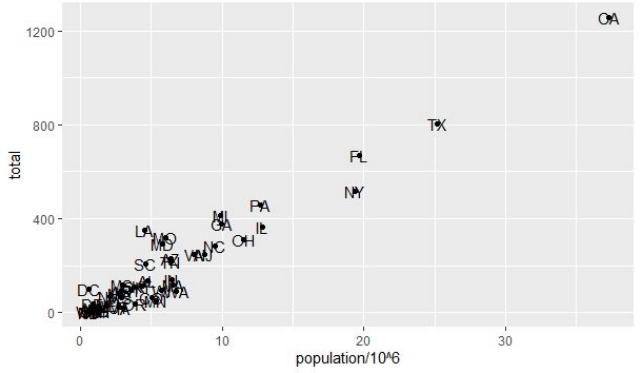
Layers

Section 1.2 - Layers

A second layer in the plot we wish to make involves adding a label to each point to identify the state. The **geom_label** and **geom_text** functions permit us to add text to the plot with and without a rectangle behind the text, respectively. Because each point (each state in this case) has a label, we need a aesthetic mapping to make the connection between points and labels

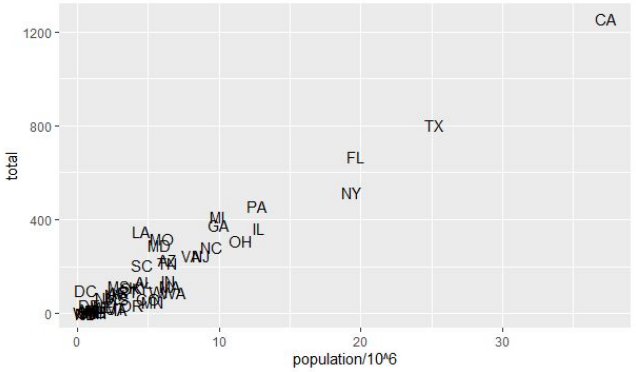
```
> p <- ggplot(data = murders)
> p + geom_point(aes(population/10^6,total))
+ geom_text(aes(population/10^6,total,label = abb))
```

Scatter plot Output by adding labels



As an example of the unique behavior of aes mentioned above, note that this call is **not an error**

```
> p
+ geom_text(aes(population/10^6,total,label=abb))
```



And this call **will be an error** because “abb” is not a globally defined variable and cannot be found outside of aes

```
> p
+ geom_text(aes(population/10^6,total),label=abb)
```

Layers

Section 1.2 - Tinkering with arguments

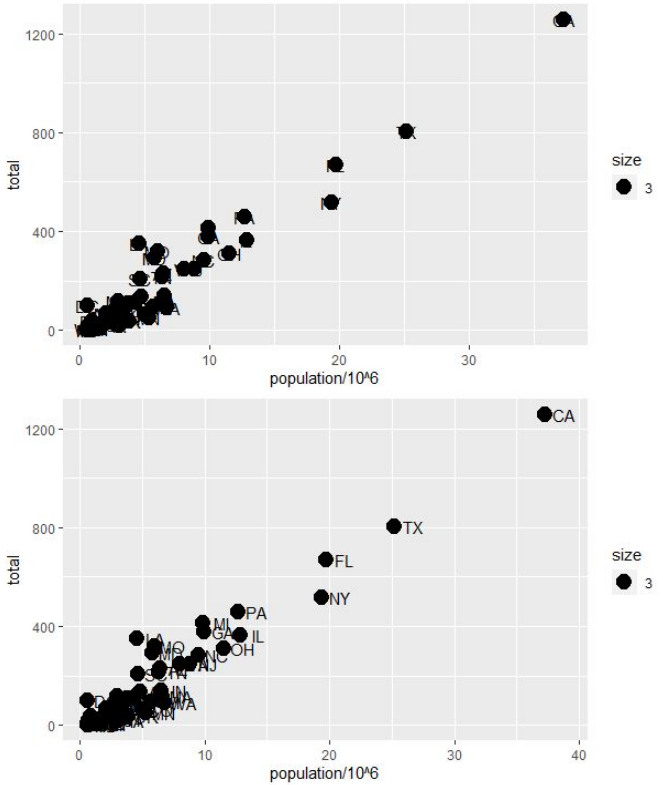
If -- *in the plot we wish to make* -- the points are larger than the default size and that size is an aesthetic, we can change it like this:

```
#change the size of the points
> p <- ggplot(data = murders)
> p + geom_point(aes(population/10^6,total),size = 3)
+ geom_text(aes(population/10^6,total,label = abb))
```

Because the points are larger, it is hard to see the labels. We can move the text slightly to the right or to the left

```
#move text labels slightly to the right
> p + geom_point(aes(population/10^6,total),size = 3)
+ geom_text(aes(population/10^6,total,label=abb),
nudge_x = 1.5)
```

Scatter plot Output



Global versus local aesthetic mappings

Section 1.2 - Global versus local aesthetic mappings

In the previous line of code, we define the mapping aes(population/10^6, total) twice, once in each geometry. We can avoid this by using a **global aesthetic mapping**. If we define a mapping in ggplot, all the geometries that are added as layers will default to this mapping. We redefine p :

```
> p <- murders %>% ggplot(aes(population/10^6,total,
Label = abb))
```

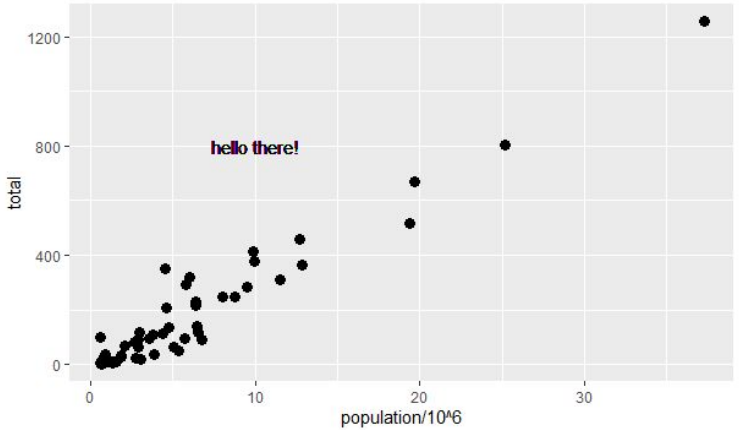
Also note that the geom_point function does not need a label argument and therefore ignores that aesthetic. and then we can simply write the following code to produce the previous plot:

```
#simplify code by adding global aesthetic
> p + geom_point(size = 3) + geom_text(nudge_x = 1.5)
```

Here is an example:

```
#local aesthetics override global aesthetics
> p + geom_point(size = 3) + geom_text(aes(x = 10, y = 800,
label = "hello there!"))
```

Scatter plot Output



Scales, Labels and Colors

Section 1.2 - Scales

First, our desired scales are in log-scale. This is not the default, so this change needs to be **added through a scales layer.** We use them like this:

```
#log base 10 scale the x axis and y axis
> p <- murders %>% ggplot(aes(population/10^6,total,
label = abb))
> p +  geom_point(size = 3) + geom_text(nudge_x = 0.05) +
scale_x_continuous(trans = "log10") +
scale_y_continuous(trans = "log10")
```

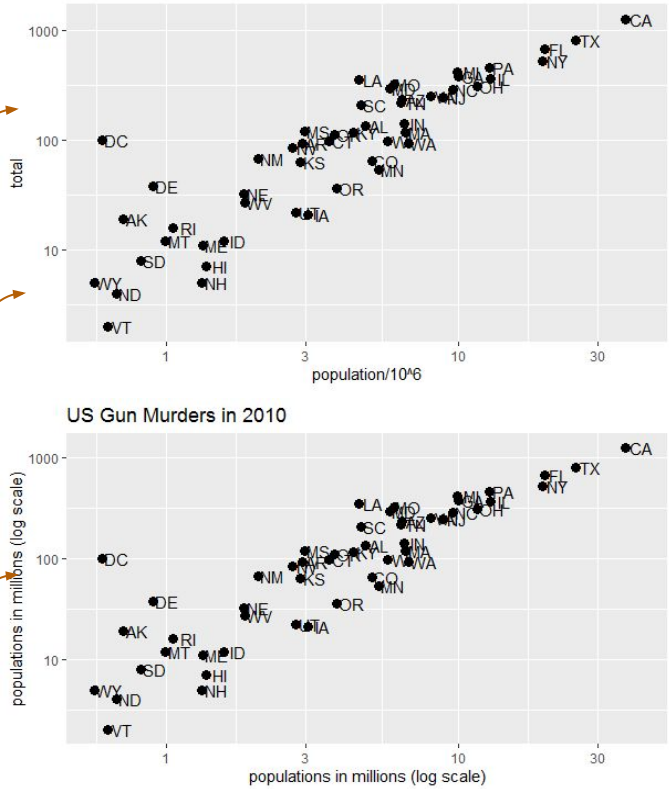
We can rewrite the code to an **efficient axed log scale** like this :

```
#efficient log scaling of the axes
>  p  +  geom_point(size=3)  +  geom_text(nudge_x  =  0.05)  +
scale_x_log10() + scale_y_log10()
```

To change **labels** and **add a title**, we use the following functions:

```
#add labels and title
>  p  +  geom_point(size=3)  +  geom_text(nudge_x  =  0.05)  +
scale_x_log10() + scale_y_log10() +
xlab("Populations in millions (log scale)") +
ylab("Total number of murders (log scale)") +
ggtitle("US Gun Murders in 2010")
```

Scatter plot Output

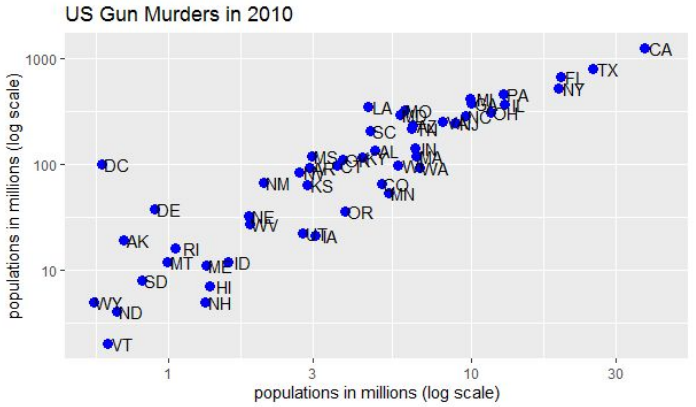


Scales, Labels and Colors

Section 1.2 - Categories as colors

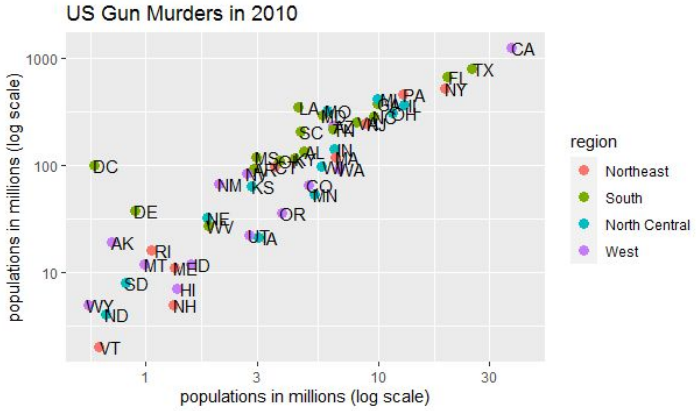
We can change the color of the points using the col argument in the **geom_point** function with make all the **points blue** by adding the color arguments.

```
#make all points blue
p + geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010") +
  geom_point(size = 3, color ="blue")
```



If want to make all the points **colored by region** we need to use aes. And we use the following code:

```
#Color points by region
p + geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010") +
  geom_point(aes(col = region), size = 3)
```



Scales, Labels and Colors

Section 1.2 - Annotation, shapes, and adjustments

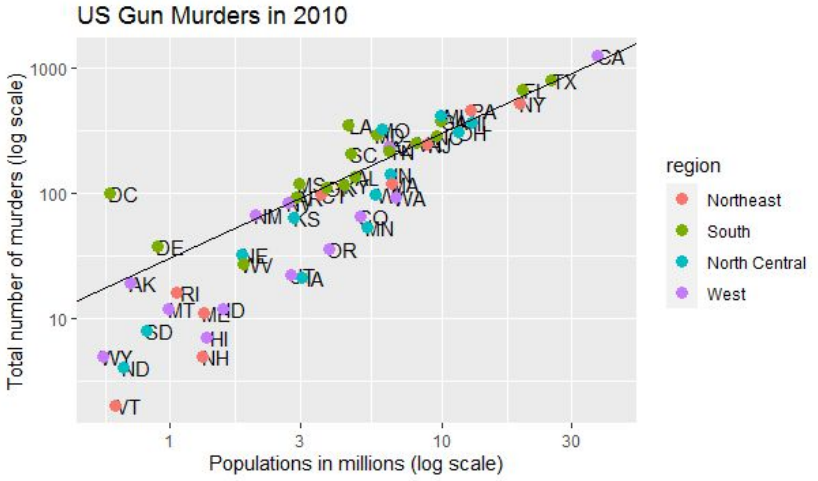
To compute the average rate for the entire country we can use some of the dplyr skills. We have to add up all the totals, add up all the populations and then take the ratio of these two. We use the summarize function to do this.

```
> r <- murders %>%
+   summarize(rate = sum(total) / sum(population) * 10^6) %>%
+   pull(rate)
> r
[1] 30.34555
```

The function pull selects a column in data frame and transforms it into a vector. This is useful to use it in combination with **magritts's pipe operator and dplyr's verbs**

To add a line we use the **geom_abline** function. ggplot2 uses ab in the name to remind us we are supplying the intercept (a) and slope (b). The default line has slope 1 and intercept 0 so we only have to define the intercept:

```
#Basic line with average murder rate for the country
p + geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010") +
  geom_point(aes(col = region), size = 3) +
  geom_abline(intercept = log10(r))
```



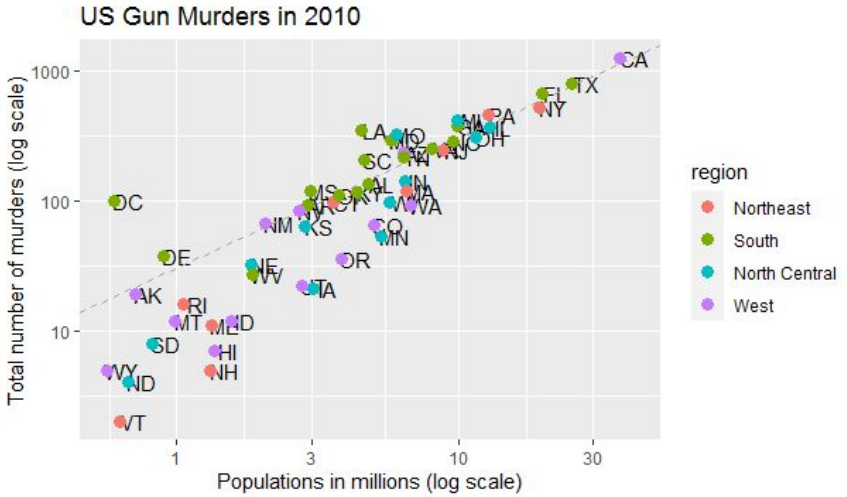
Scales, Labels and Colors

Section 1.2 - Annotation, shapes, and adjustments

We can change the line type and color of the lines using arguments. Also, we draw it first so it doesn't go over our points:

```
#change line to dashed and dark grey,line under points
p + geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010") +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col = region), size = 3)
```

The argument lty can be used to specify the line type. Line type (lty) can be specified using either text ("blank", "solid", "dashed", "dotted", "dotdash" , "twodash") or number (0,1,2,3,4,5,6). **Note that lty = dashed is identical to lty = 2**

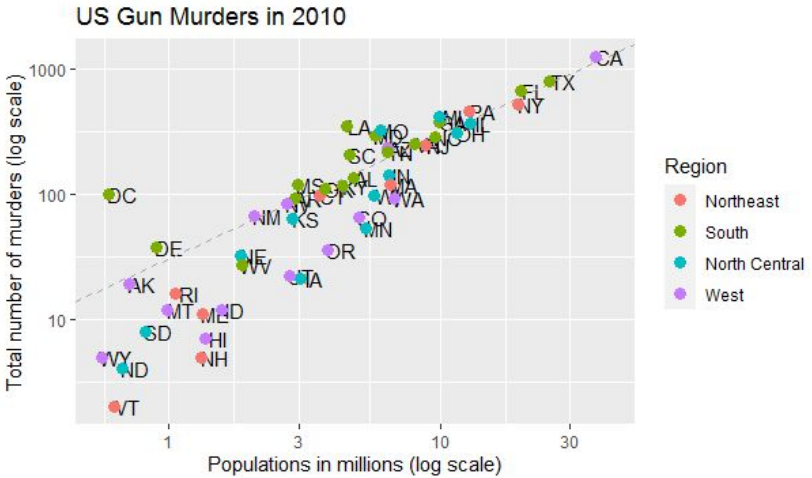


Scales, Labels and Colors

Section 1.2 - Annotation, shapes, and adjustments

We can make changes to the legend via the **scale_color_discrete** function. If we want in our plot the word **region is capitalized**, we can change it like this :

```
#changes legend title
p + geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010") +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col = region), size = 3) +
  scale_color_discrete(name = "Region")
```

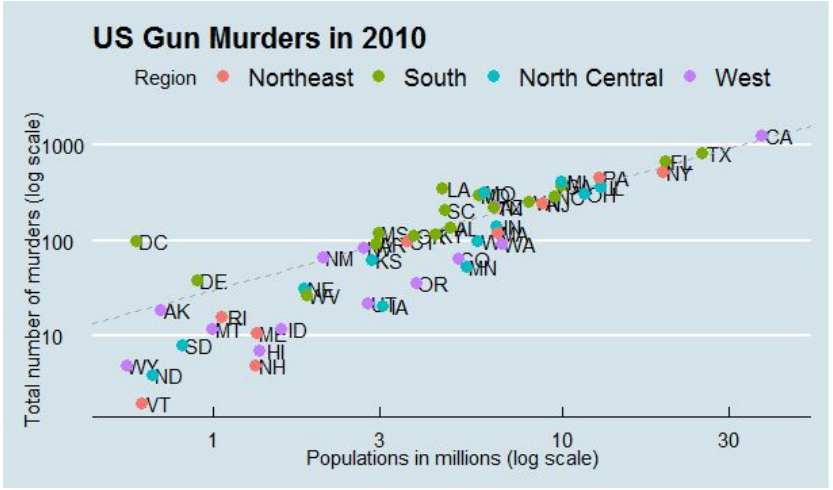


Add on packages

Section 1.3 - Adding themes

The power of ggplot2 is augmented further due to the availability of add-on packages. The remaining changes needed to put the finishing touches on our plot require the ggthemes and ggrepel packages. Many other themes are added by the package **ggthemes**. Among those are the theme_economist theme that we used. After installing the package, you can change the style by adding a layer like this:

```
#themes from ggthemes
library(ggthemes)

#Create the plot
p + geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010") +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col = region), size = 3) +
  scale_color_discrete(name = "Region") +
  theme_economist()
```

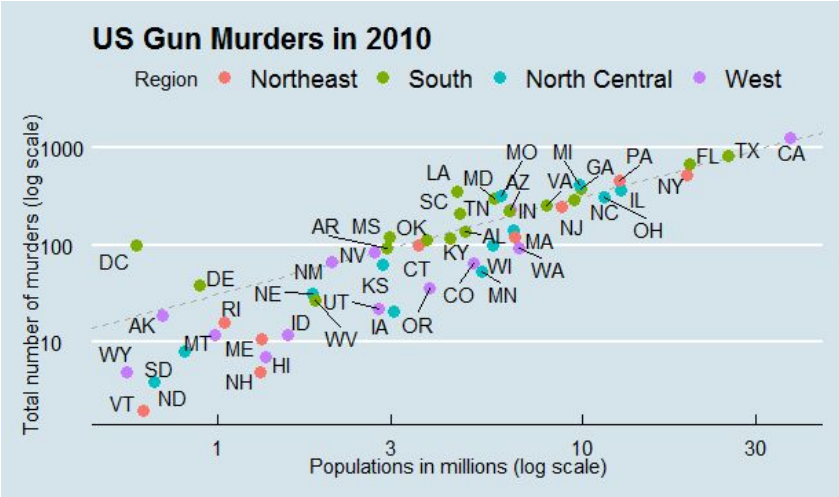


Add on packages

Section 1.3 - Putting it all together

Now that we are done testing, we can write one piece of code that produces our desired plot from scratch.

```
#load libraries
library (tidyverse)
library(ggthemes)
library(ggrepel)
library (dslabs)
data(murders)

#define the intercept
r <- murders %>%
  summarize(rate = sum(total) / sum(population) * 10^6) %>%
  pull(rate)

#make the plot,combining all elements
murders %>% ggplot(aes(population/10^6, total, label = abb)) +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3) +
  geom_text_repel() +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010") +
  scale_color_discrete(name = "Region") +
  theme_economist()
```

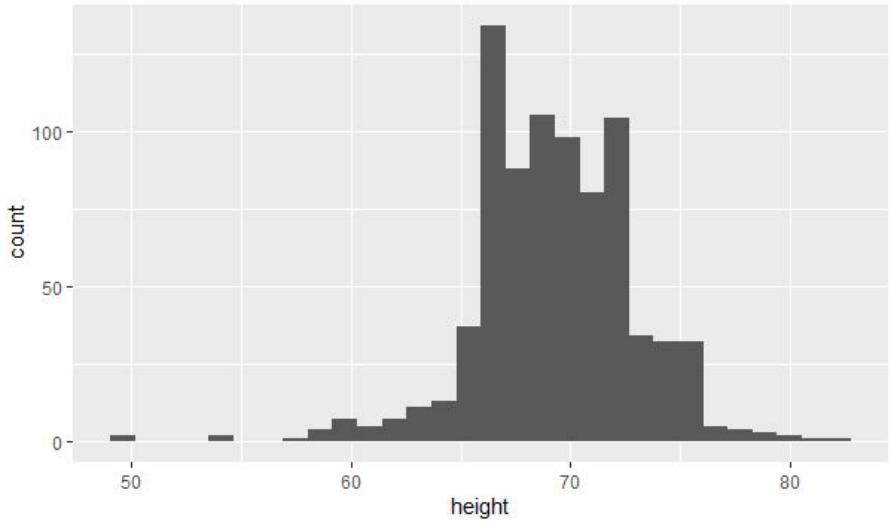


Other Examples

Section 1.3 - Histogram in ggplot2

To generate histogram we use **geom_histogram**. Use the binwidth argument to change the width of bins, the fill argument to change the bar fill color and the col argument to change bar outline color: We will make a histogram for the male's height. The required argument that we will construct in histogram is x. But first we need to filter the heights to only include the males and use the geom_histogram geometry.

```
#load heights data
> library(tidyverse)
> library(dslabs)
> data(heights)

#define p
> p <- heights %>%
  filter(sex == "Male") %>%
  ggplot(aes(x = height))
> p + geom_histogram()
```

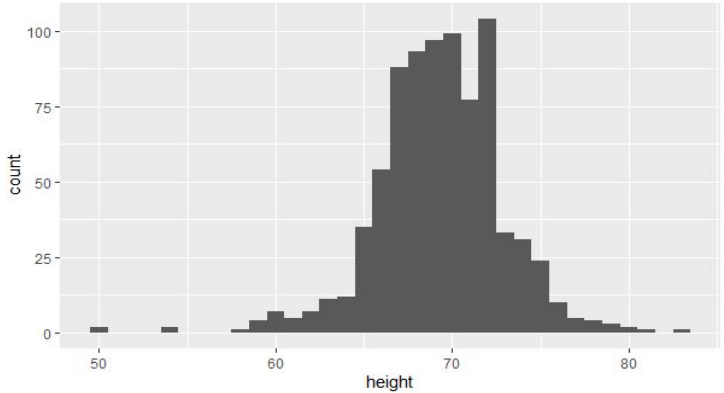


Other Examples

Section 1.3 - Histogram in ggplot2

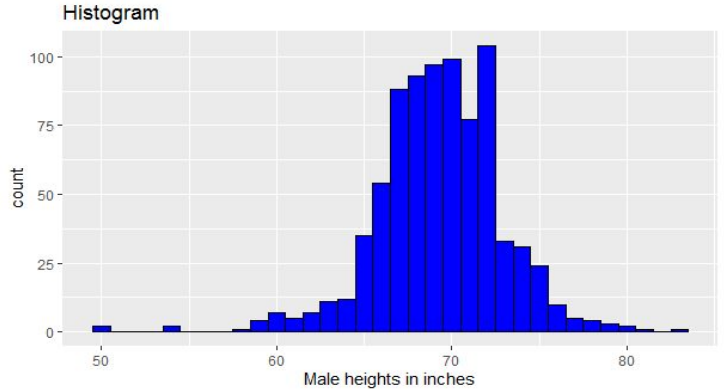
We notice that we get a warning or a message saying that the bin width was not picked. So what we can do now is add the bin width that we want.

```
#define p
> p <- heights %>%
  filter(sex == “Male”) %>%
  ggplot(aes(x = height))
> p + geom_histogram(binwidth = 1)
```



We’ll change the color and add a title

```
#define p
> heights %>%
  filter(sex == "Male") %>%
  ggplot(aes(height)) +
  geom_histogram(binwidth = 1, fill = "blue", col = "black") +
  xlab("Male heights in inches") +
  ggtitle("Histogram")
```

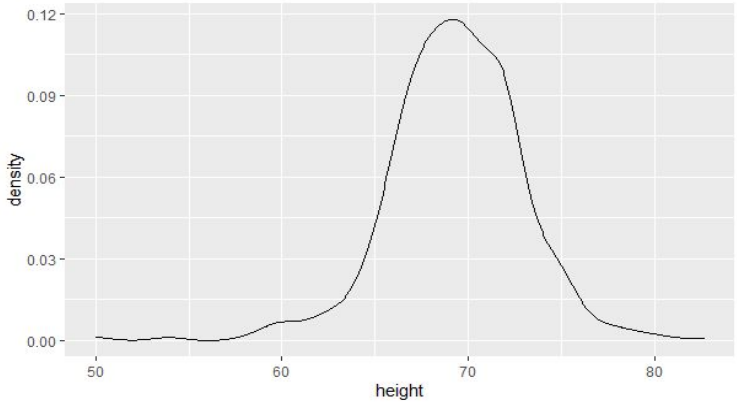


Other Examples

Section 1.3 - Histogram in ggplot2

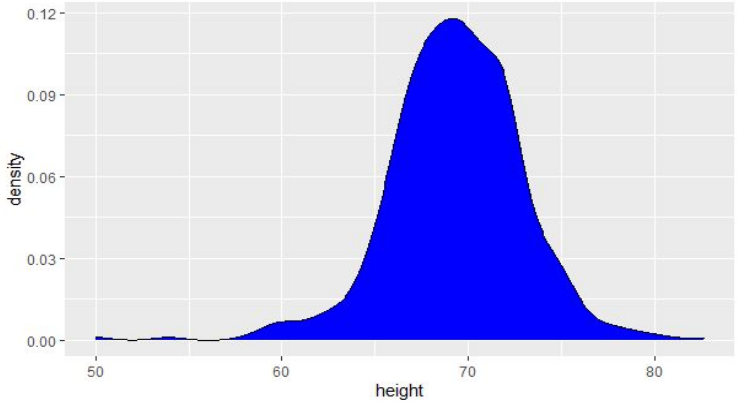
To **create a smooth density**, we use the **geom_density**. To make a smooth density plot with the data previously shown as a histogram we can use this code

```
#smooth density
> heights %>%
  filter(sex == "Male") %>%
  ggplot(aes(height)) +
  geom_density()
```



To **fill in color** we can use the fill argument

```
#smooth density
> heights %>%
  filter(sex == "Male") %>%
  ggplot(aes(height)) +
  geom_density(fill="blue")
```

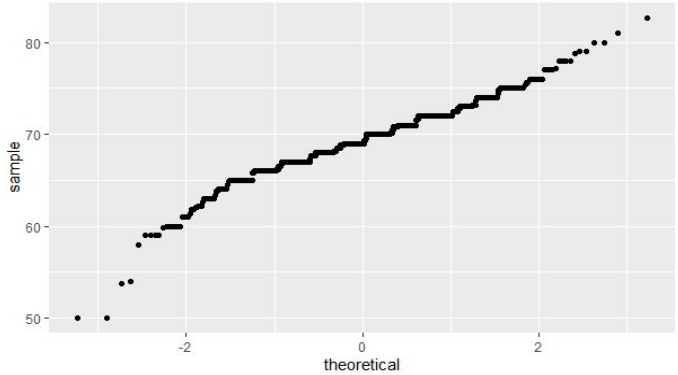


Other Examples

Section 1.3 - Quantile Quantile plot in ggplot2

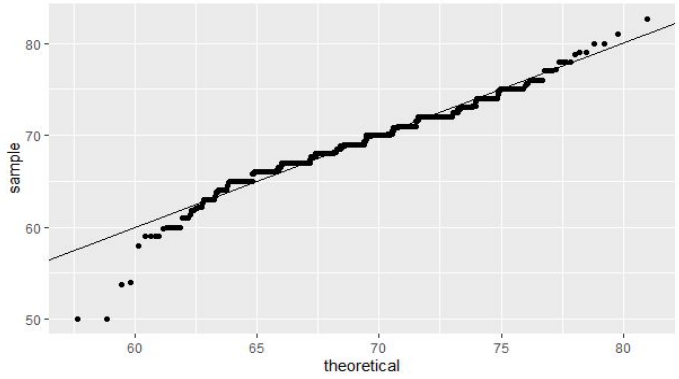
geom_qq() created a quantile-quantile plot. This geometry requires the sample argument. By default, the data are compared to a standard normal distribution with a mean of 0 and standard deviation of 1. Here is the qqplot for men heights

```
#basic QQ-plot
> heights %>% filter(sex=="Male") %>%
  ggplot(aes(sample = height)) +
  geom_qq()
```



By default, the sample variable is compared to a normal distribution with average 0 and standard deviation 1. To change this, we use the **dparams** arguments based on the help file. Adding an identity line is as simple as assigning another layer. For straight lines, we use the **geom_abline** function. The default is the identity line (slope = 1, intercept = 0)

```
#QQ-plot against a normal distribution with same mean/sd as data
> params <- heights %>%
  filter(sex == "Male") %>%
  summarize(mean = mean(height), sd = sd(height))
> heights %>% filter(sex=="Male") %>%
  ggplot(aes(sample = height)) +
  geom_qq(dparams = params) +
  geom_abline()
```

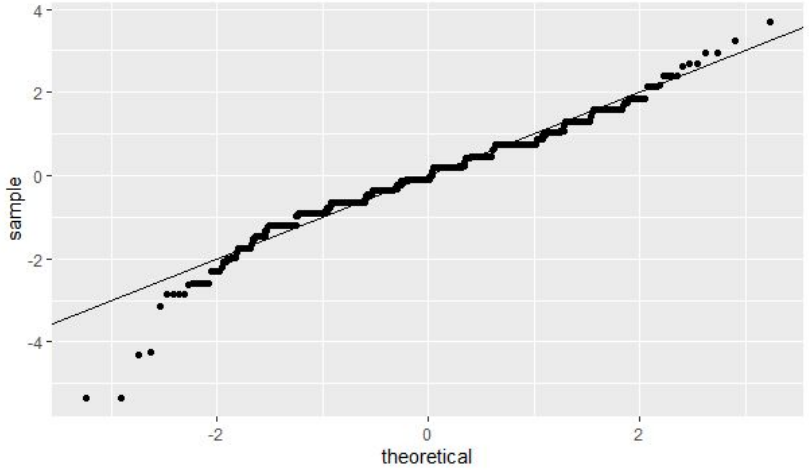


Other Examples

Section 1.3 - Quantile Quantile plot in ggplot2

Another option here is to first scale the data so that we have them in standard units and plot it against the standard normal distribution. This saves us the step of having to compute the mean and standard deviation. The code is a little cleaner and it looks like this

```
#QQ-plot of scaled data against the standard normal distribution
> heights %>% filter(sex=="Male") %>%
  ggplot(aes(sample = height)) +
  geom_qq()
```

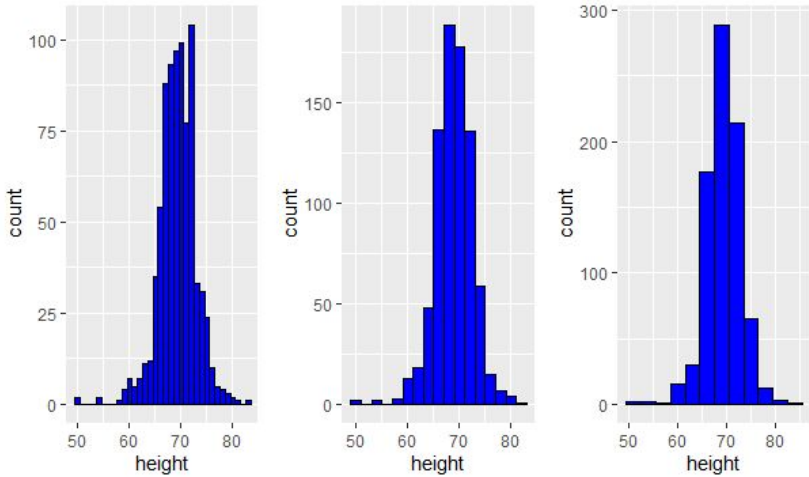


Other Examples

Section 1.3 - Quantile Quantile plot in ggplot2

Plots can be arranged adjacent to each other using grid.arrange() function from the gridextra package. First, create the plots and save them to objects (p1,p2,..). Then pass the plot objects to grid.arrange(). The first step is to define the plot and assign it to the objects.

```
#define plots p1,p2,p3
>p <- heights %>% filter(sex == "Male") %>% ggplot(aes(x = height))
p1 <- p + geom_histogram(binwidth = 1, fill = "blue", col = "black")
p2 <- p + geom_histogram(binwidth = 2, fill = "blue", col = "black")
p3 <- p + geom_histogram(binwidth = 3, fill = "blue", col = "black")

#arrange plots next to each other in 1 row,3 columns
> install.packages("gridExtra")
> library(gridExtra)
> grid.arrange(p1, p2, p3, ncol = 3)
```



Statistic Quality & Engineering
Universitas Indonesia

Content of this module :

1. Introduction to ggplot2
2. Summarizing with dplyr
3. Example of Data Visualization using Gapminder Data
4. Data Visualization Principles

This is what you will learn in this section...

Section 2 - overview

Section	What you will learn...
2.1	• Introduction to dplyr
2.2	• Summarizing data • Extracting data • Grouping data • Examining data after sorting

Introduction

Introducing the package “dplyr”

dplyr is a package mainly used as a grammar of data manipulation. As it was stated before, to install and load the package we could use the steps below.

```
> install.packages(“dplyr”)
> library(dplyr)
```

Several functions that are available from the package **dplyr** are:

1. mutate( )
2. select( )
3. filter( )
4. summarize( )
5. group_by( )
6. arrange( )

We have discussed the use of the first three in the previous module, now we will learn about **summarize( ), group_by( ),** and **arrange( )**

Summarize

Compute summary statistics

Summarize is a useful function in R, especially to obtain a certain information of a data

Summarize( ) is used to return the result of a function in a form of data frame.

The general function summarize( ) is used as:

```
>   summarize(dataset,   col1_name   =   function(   ),
col2_name = function( ), …, coln_name = function( ))
```

We can create as many columns containing summary of a data as we like. col1_name, col2_name, etc. are our preferred column names for the new data frame containing summary statistics.

Examples of function( ) that could be used are mean( ), sd( ), median( ), min( ), max( ), and other functions that return a single value.

For example, we would like to know the average and standard deviation of male height in a dataset called heights. Note that we use the pipe, so calling the dataset in the function won't be necessary.

```
>     heights   %>%   filter(sex   ==   "Male")   %>%
   summarize(average          =          mean(height),
   standard_deviation = sd(height))
```

```
    average standard_deviation
1 69.31475           3.611024
```

You can define your own function as long as it returns a single value!

Summarize

Compute summary statistics

Summarize is a useful function in R, especially to obtain a certain information of a data

We treat the result as a data frame

Because it is returned as a data frame, we can call the value of each column using the separator **$** after saving the dataset result. Suppose we would like to know the average value. Here, we define the result as a data frame called s.

```
>     s  <-  heights  %>%  filter(sex  ==  “Male”)  %>%
   summarize(average          =          mean(height),
   standard_deviation = sd(height))
>     s$average
>     s$standard_deviation
```

```
> s <- heights %>% filter(sex == "Male") %>% summarize(average =
mean(height), standard_deviation = sd(height))
> s$average
[1] 69.31475
> s$standard_deviation
[1] 3.611024
```

Rules of summarize( )

1. The result of function summarize( ) is a **data frame** which consists of the summary values with column names defined prior in the function call.
2. The function summarize( ) can return the summary yielded by any function (even a manually defined function) as long as the function **returns a single value**.

Dot Placeholder

Extracting data

Dot operator returns a single value or vectors instead of a data frame

Accessing a value in a piped data

Dot placeholder or dot operator allows you to access values stored in data that is being piped (%>%). This allows the dplyr package to return a single vectors instead of data frame.

All we need to do is call the column name with **.$col_name**

For example, we want the final result of s being a value of average male height instead of a data frame.

```
>     avg <- heights %>% filter(sex == "Male") %>%
   summarize(average = mean(height),
   standard_deviation = sd(height)) %>% .$average
>     avg
```

This is what we get when we call 'avg'

```
> avg <- heights %>% filter(sex == "Male") %>% summarize(average
= mean(height), standard_deviation = sd(height)) %>% .$average
> avg
[1] 69.31475
```

Thus, the dot placeholder is useful when we would like to access a value of a summarized statistics from a piped data frame.

Note that the result of .$average assigned as 'avg' is equivalent to s$average from before.

Grouping data

Compute summary statistics

Group_by function is used to create a grouped data frame

Grouping data frame by variables

By creating a grouped data frame, the data frame would be grouped based on the function called.
The basic function of group_by is group_by(variables).
The data frame then would be grouped based on the levels of the corresponding variable.

For example, we would like to group the dataset *heights* by female and male.

```
>      heights %>% group_by(sex)
```

However, this function itself would not result in any changes in a data frame. This function affects to results of the next function called, as the next function will instead operate on a grouped data frame.

Combining group_by and summarize

By using group_by before summarize, the summarize function will be operated on each level of the variable used in group_by.
For instance, we calculate the average and standard deviation of height after grouping the data frame by sex.

```
>      heights %>% group_by(sex) %>% summarize(average
   = mean(height), standard_deviation = sd(height))
```

```
# A tibble: 2 x 3
  sex    average standard_deviation
  <fct>    <dbl>              <dbl>
1 Female    64.9               3.76
2 Male      69.3               3.61
```

dplyr will automatically calculate each summarize function for female and male separately.

Arrange

Arranging data frame by a column

Arrange function is used to sort data by a single column in a data frame

Arranging a data frame

Sometimes we would like to arrange a data in a descending or ascending order to obtain information about the data.
For a data frame, we can arrange it by sorting the values of a variable.
The basic function of arrange is **arrange(*variable*)**

For instance, we want to arrange the dataset height from the shortest to the tallest.

```
>      heights %>% arrange(height) %>% head()
```

```
> heights %>% arrange(height) %>% head()
     sex height
1   Male     50
2   Male     50
3 Female     51
4 Female     52
5 Female     52
6 Female     53
```

The default of arrange( ) is to sort data in an ascending order. If we would like to arrange it in a descending order, an additional argument is required:
arrange(desc(*variable*))

```
>      heights %>% arrange(desc(height)) %>% head()
```

```
> heights %>% arrange(desc(height)) %>% head()
   sex   height
1 Male 82.67717
2 Male 81.00000
3 Male 80.00000
4 Male 80.00000
5 Male 79.05000
6 Male 79.00000
```

Arrange

Arranging data frame by a column

Top_n function can be used to find out the top *n* of a data

Obtaining information about top *n* of a data

If we use the function head( ) after arranging data, only six rows of data are presented. What if we'd like to know the 10 tallest height from dataset heights?

```
>      heights %>% top_n(10, height)
```

```
> heights %>% top_n(10, height)
      sex   height
1  Female 78.74016
2    Male 78.00000
3    Male 80.00000
4    Male 80.00000
5    Male 79.05000
6    Male 78.00000
7    Male 79.00000
8    Male 78.00000
9    Male 78.00000
10   Male 78.74000
11 Female 79.00000
12   Male 81.00000
13   Male 82.67717
```

As we can see, the function returns the data of top ten tallest heights from the dataset, but they are not sorted or arranged in an order. We can **combine the function top_n with arrange** to obtain such results.

```
>      heights %>% arrange(desc(height)) %>% top_n(10,
    height)
```

```
> heights %>% arrange(desc(height)) %>% top_n(10, height)
      sex   height
1    Male 82.67717
2    Male 81.00000
3    Male 80.00000
4    Male 80.00000
5    Male 79.05000
6    Male 79.00000
7  Female 79.00000
8  Female 78.74016
9    Male 78.74000
10   Male 78.00000
11   Male 78.00000
12   Male 78.00000
13   Male 78.00000
```

Content of this module :

1. Introduction to ggplot2
2. Summarizing with dplyr
3. Example of Data Visualization using Gapminder Data
4. Data Visualization Principles

This is what you will learn in this section...

Section 3 - overview

Section	What you will learn...
3.1	• Introduction to Gapminder Dataset • Data Examination
3.2	• Faceting • Using Time Series Plot • Transformations • Stratify and Boxplots • Comparing Distributions • Density Plots

Introduction to Gapminder

Problem Statement and Future Analysis

What is Gapminder?

- A prevalent worldview is that the world is divided into two groups of countries:
  - Western world: high life expectancy, low fertility rate
  - Developing world: lower life expectancy, higher fertility rate
- Gapminder data can be used to evaluate the validity of this view.
- A scatterplot of life expectancy versus fertility rate in 1960 suggests that this viewpoint was grounded in reality 50 years ago. **Is it still the case today?**

	country	year	infant_mortality	life_expectancy	fertility	population	gdp	continent	region
1	Albania	1960	115.40	62.87	6.19	1636054	NA	Europe	Southern Europe
2	Algeria	1960	148.20	47.50	7.65	11124892	13828152297	Africa	Northern Africa
3	Angola	1960	208.00	35.98	7.32	5270844	NA	Africa	Middle Africa
4	Antigua and Barbuda	1960	NA	62.97	4.43	54681	NA	Americas	Caribbean
5	Argentina	1960	59.87	65.39	3.11	20619075	108322326649	Americas	South America
6	Armenia	1960	NA	66.86	4.55	1867396	NA	Asia	Western Asia
7	Aruba	1960	NA	65.66	4.82	54208	NA	Americas	Caribbean
8	Australia	1960	20.30	70.87	3.45	10292328	96677859364	Oceania	Australia and New Zealand
9	Austria	1960	37.30	68.75	2.70	7065525	52392699681	Europe	Western Europe
10	Azerbaijan	1960	NA	61.33	5.57	3897889	NA	Asia	Western Asia
11	Bahamas	1960	51.00	62.00	4.50	109526	1306269490	Americas	Caribbean
12	Bahrain	1960	134.50	51.64	7.09	162501	NA	Asia	Western Asia
13	Bangladesh	1960	176.30	46.20	6.73	48200702	12767231590	Asia	Southern Asia
14	Barbados	1960	69.50	61.80	4.33	230934	784120376	Americas	Caribbean
15	Belgium	1960	NA	71.59	2.54	8100037	NA	Europe	Western Europe

Get to know more about “Gapminder”

Introduction and Data Examination

Let’s call out the Gapminder Data

```
#load Packages
library(ggplot2)
library(dslabs)
library(dplyr)
library(tidyverse)

#Load Gapminderr
data(gapminder)
head(gapminder)
str(gapminder)
```

Gapminder is contains longitudinal data (1960-2016) on life expectancy, GDP per capita, and population for 142 countries.

Data Overview

```
> str(gapminder)
'data.frame':   10545 obs. of  9 variables:
 $ country         : Factor w/ 185 levels "Albania","Algeria",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ year            : int  1960 1960 1960 1960 1960 1960 1960 1960 1960 1960 ...
 $ infant_mortality: num  115.4 148.2 208 NA 59.9 ...
 $ life_expectancy : num  62.9 47.5 36 63 65.4 ...
 $ fertility       : num  6.19 7.65 7.32 4.43 3.11 4.55 4.82 3.45 2.7 5.57 ...
 $ population      : num  1636054 11124892 5270844 54681 20619075 ...
 $ gdp             : num  NA 1.38e+10 NA NA 1.08e+11 ...
 $ continent       : Factor w/ 5 levels "Africa","Americas",..: 4 1 1 2 2 3 2 5 4 3 ...
 $ region          : Factor w/ 22 levels "Australia and New Zealand",..: 19 11 10 2 15 21 2 1 22 21 ...
```

Get to know more about “Gapminder”

Introduction and Data Examination

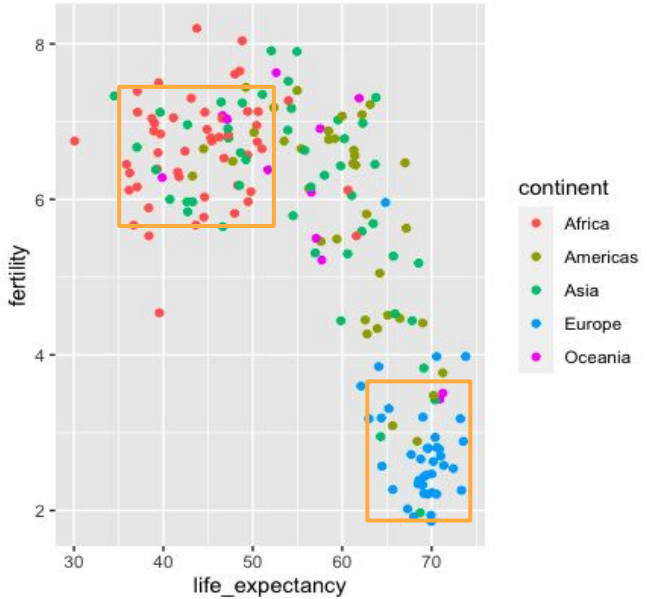
Let’s make simple analysis…

```
#Load Gapminder
data(gapminder)
head(gapminder)
str(gapminder)

#compare infant mortality in United States and Indonesia
gapminder %>%
  filter(year == 1962 & country %in% c("United States","Indonesia")) %>%
  select(country, infant_mortality, life_expectancy)
```

	country	infant_mortality	life_expectancy
1	Indonesia	141.2	46.65
2	United States	24.9	70.21

Based on the table, we can conclude that United States have **better infant_mortatiliy rate rather than Indonesia and infant mortality does affect negatively to life expectancy.**

```
#Scatterplot life expectancy VS fertility in 1962
gapminder %>%
  filter(year == 1962) %>%
  ggplot(aes(life_expectancy, fertility, color = continent)) +
  geom_point()
```



In 1962, world were divided into cluster, Western countries and Developing countries, **is it still the case in 2012?**

Facet in Nutshell

Simple Explanation and Function

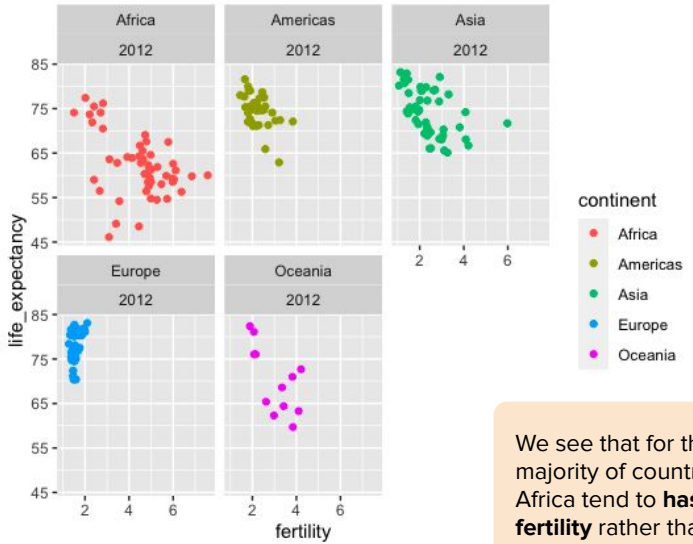
Grouping allows you to plot multiple variables in a single graph, using visual characteristics such as color, shape, and size. **In faceting, a graph consists of several separate plots or *small multiples*, one for each level of a third variable, or combination of variables.**

Facet Function Consist of 2 Formula:

- The **facet_wrap()** function creates a separate graph for each level of variable. The **ncol** option controls the number of columns.
- The **facet_grid()** function allows faceting by up to two variables, with rows faceted by one variable and columns faceted by the other variable. To facet by only one variable **facet_grid(.~variable)**, use the dot operator as the other variable

```
#facet by continent and year for 2012
gapminder %>%
  filter(year == 2012) %>%
  ggplot(aes(fertility,life_expectancy, color = continent)) +
  geom_point() +
  facet_wrap(continent ~ year)
```

Example 1. Based on Dataset on Gapminder, we would like to know more about life expectancy and fertility each continents on 2012.



We see that for the majority of countries in Africa tend to **has higher fertility** rather than countries in Europe.

Facet: Visualizing Simple Comparison 2 Variables

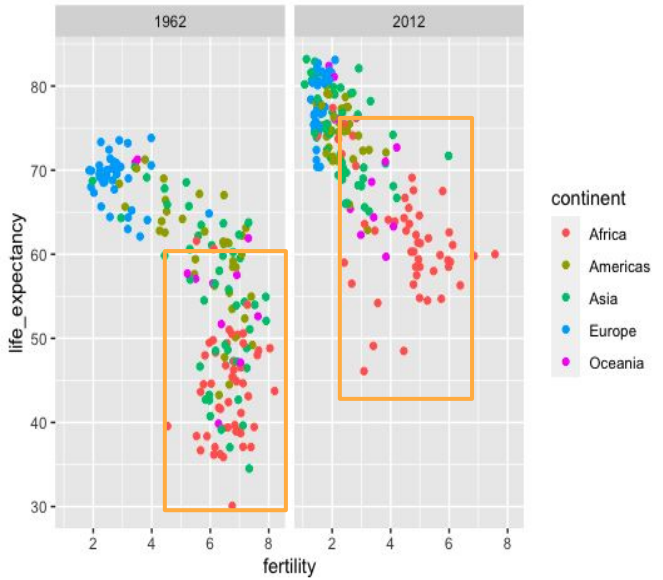
Using facet to comparing life expectancy & fertility in 1962 and 2012

We want to know **how life expectancy and fertility in every continents, after 50 year does the world still facing the same problems?** In the gapminder data table includes a column with the countries' Life Expectancy and Fertility in various years. In 1962, there were stigma that Western Countries has higher life expectancy rather than the developed countries, does it still implies after 50 years?

```
#facet by continent and year for 1962 VS 2012
gapminder %>%
  filter(year %in% c(1962,2012)) %>%
  ggplot(aes(life_expectancy, fertility, col =
continent)) +
  geom_point() +
  facet_grid(.~ year)
```

Using Facet_Grid to define (Row ~ Col, ncol =...)

Faceting keeps the axes fixed across all plots, easing comparisons between plots. **Grid is more preferable if the current data only consist 2 variable that will be compared.**



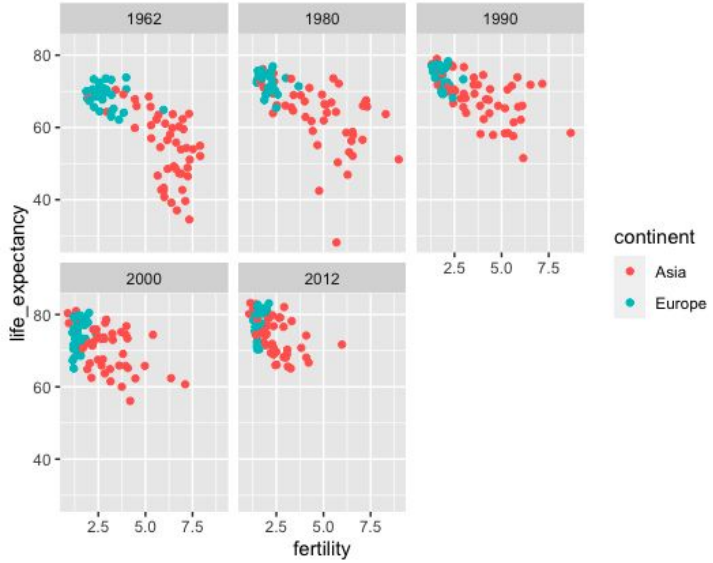
We see that comparison between Western Countries and Developed Countries **no longer make sense in 2012.**

Facet: Visualizing Simple Comparison Multiple Rows

Using facet to comparing life expectancy & fertility European and Asia in 1962, 1980, 1990, 2000, 2012

Based on the analysis before, We know that after 50 years developed countries has improved their life expectancy so the stigma no longer make senses. We want to know **how life expectancy and fertility in European and Asia Countries perform in the last 50 years.**

```
#facet by year, plot wrapped into multiple rows
years <- c(1962, 1980, 1990, 2000, 2012)
continents <- c("Europe", "Asia")
gapminder %>%
  filter(year %in% years & continent %in% continents) %>%
  ggplot(aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_wrap(~year)
```



Using Facet_Wrap to define (Row ~ Col, ncol =...)

Facet Wrap used because the quantity of row is more than 2, thus it would be more useful to use Facet Wrap instead

From the graph we know that Europe and Asia has **great improvement on their life expectancy**

Time Series Plot in Nutshell

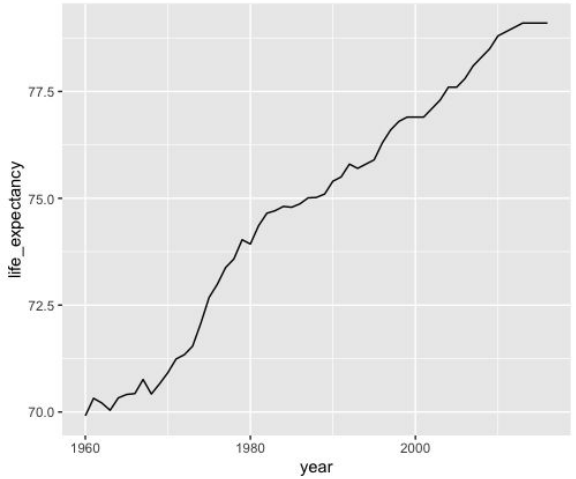
Simple Explanation and Function

A graph can be a powerful vehicle for displaying change over time. The most common time-dependent graph is the time series line graph. When one of the two variables represents time, a line plot can be an effective method of displaying relationship.

A time series is a set of quantitative values obtained at successive time points. The intervals between time points (e.g., hours, days, weeks, months, or years) are usually equal. For example, the code below displays the relationship between time (*year*) and life expectancy (*lifeExp*) in the United States between 1960 and 2016. The data comes from the gapminder dataset.

Example 1. Based on Dataset on Gapminder, we would like to know more about trend of life expectancy in US from 1960-2016

```
#Time Series over the past few years Life Expectancy in
US (1960-2016)
#Filter Data only for US only
library(dplyr)
df <- gapminder %>%
  filter(country == "United States")

#Plotting the time series
ggplot(df, aes(year,life_expectancy)) +
    geom_line()
```



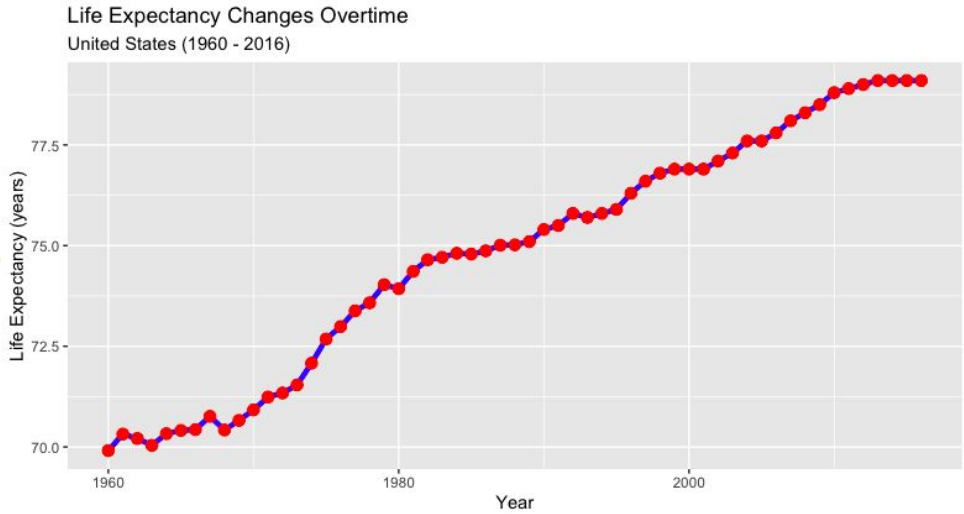
From the graph we know that the life expectancy of United State **Improved from 1960 to 2016**

Time Series Plot: Visualizing 1 Variable

Using line plot to visualize life expectancy changes over times on United States in 1960 - 2016

It's hard to read from the previous line graph, we need to add some adjustment to attract and help your reader to understand more on your data.

```
#Filter Data only for US only
library(dplyr)
df <- gapminder %>%
  filter(country == "United States")

#Plotting the time series
ggplot(df, aes(year,life_expectancy)) +
    geom_line(size = 1.5,
              color = "Blue") +
    geom_point(size = 3,
               color = "red") +
    labs(x = "Year",
         y = "Life Expectancy (years)",
         title = "Life Expectancy Changes Overtime",
         subtitle = "United States (1960 - 2016)")
```



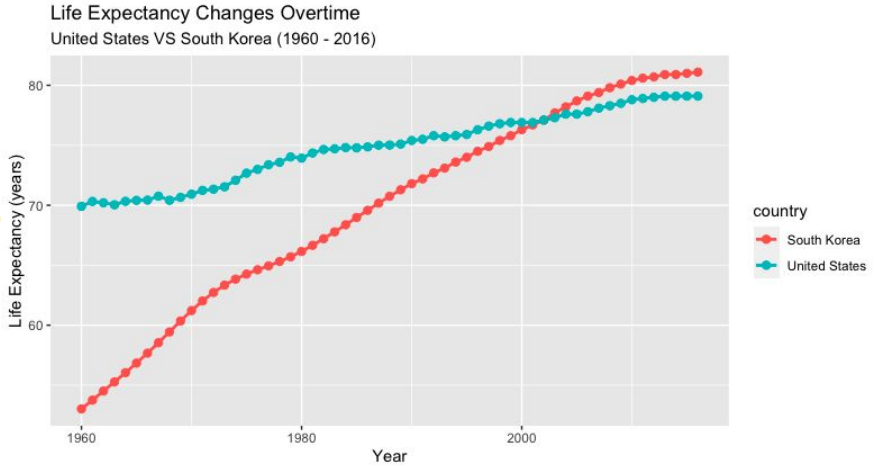
Looks much better than before

Time Series Plot: Visualizing two or more variable

Using line plot to visualize life expectancy changes over times on South Korea and United States in 1960 - 2016

We want to know if we going to compare South Korea and US, does the increment of life expectancy US is better than the South Korea?

```
#Plotting the time series for US and South Korea 1960 - 2016
countries <- c("United States","South Korea")
gapminder %>% filter(country %in% countries) %>%
    ggplot(aes(year, life_expectancy, col = country)) +
    geom_line(size = 1) +
    geom_point(size = 2.25) +
    labs(x = "Year",
         y = "Life Expectancy (years)",
         title = "Life Expectancy Changes Overtime",
         subtitle = "United States VS South Korea (1960 - 2016)")
```



You can plot multiple lines on the same graph. **Remember to group or color by a variable so that the lines are plotted independently.**

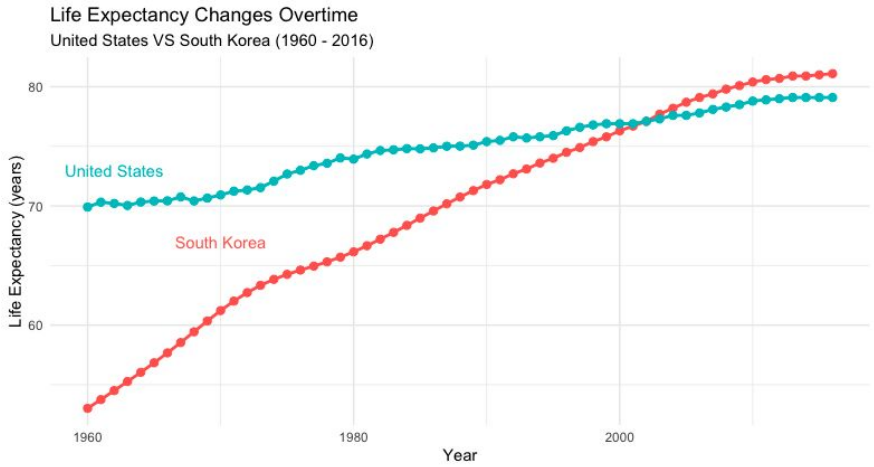
As we can see, the graph shown that the increment of life expectancy South Korea is better than the US

Time Series Plot: Visualizing two or more variable and adding labels

Using line plot to visualize life expectancy changes over times on South Korea and United States in 1960 - 2016

We want to know if we going to compare South Korea and US, does the increment of life expectancy US is better than the South Korea?

```
#adding labels and removing legend
countries <- c("United States","South Korea")
label <- data_frame(country = countries, x = c(1962,1970), y
= c(73,67))
gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year, life_expectancy, col = country)) +
  geom_line(size = 1) +
  geom_point(size = 2.25) +
  geom_text(data = label, aes(x, y, label = country)) +
  labs(x = "Year",
       y = "Life Expectancy (years)",
       title = "Life Expectancy Changes Overtime",
       subtitle = "United States VS South Korea (1960 -
2016)") +
  theme_minimal() +
  theme(legend.position = "none")
```



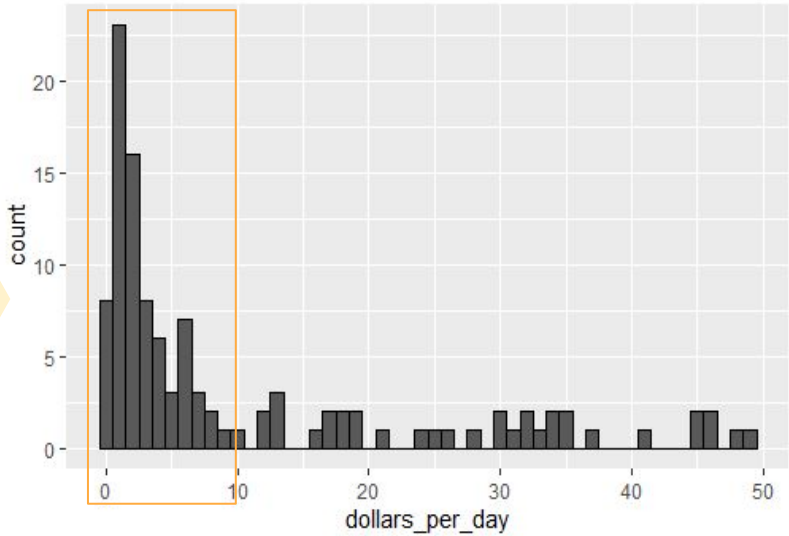
As we can see, the graph shown that the increment of life expectancy South Korea is better than the US

Transformation

Add a Column and Make a Histogram

We want to know **how wealth distribution across the world**. In the gapminder data table includes a column with the countries' gross domestic product (GDP). The GDP per person is often used as a rough summary of a country's wealth.

```
# add column of GDP per day per person
> gapminder <- gapminder %>%
    mutate(dollars_per_day = gdp/population/365)


# histogram of dollars per day
gapminder %>%
    filter(year == 1970 & !is.na(gdp)) %>%
    ggplot(aes(dollars_per_day)) +
    geom_histogram(binwidth = 1, color = "black")
```



We see that for the majority of countries, averages are below $10 a day. It will be more informative to transform the data with the **log transformation**.

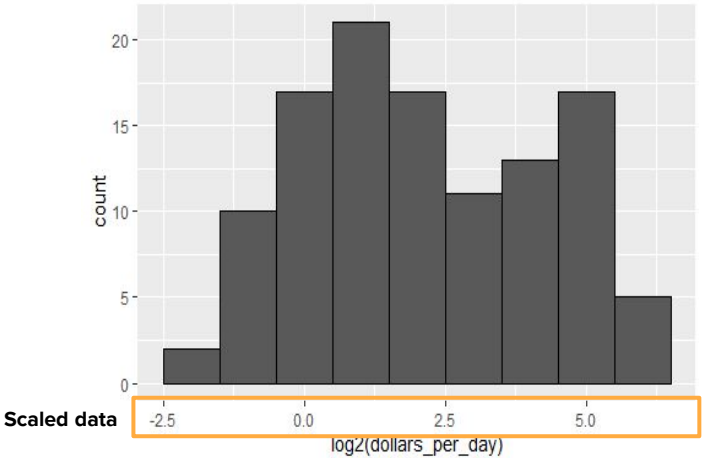
Transformation

How to use log transformation in plots: scales data vs scales axis

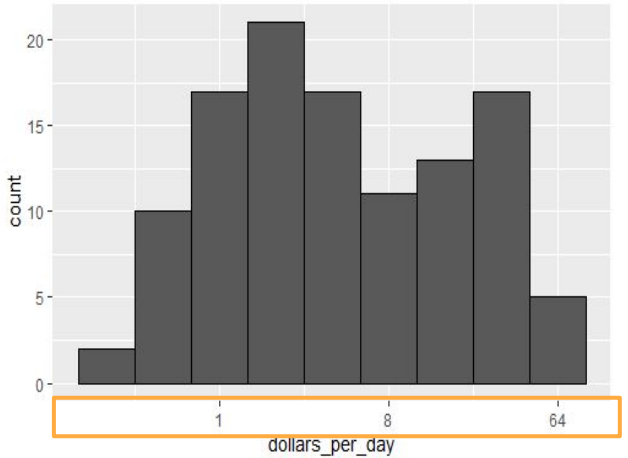
There are two ways we can use log transformation in plots: by **scaling the data** or **scaling the axis**. (1) Scaling the data means we log the values before plotting them. If we log the data, we can more easily interpret intermediate values. (2) Scaling the axis has the advantage that we see the original values on the axis.

```
#log2 scaled data
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(log2(dollars_per_day))) +
  geom_histogram(binwidth = 1, color = "black")
```

```
#log2 scaled x-axis
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, color = "black") +
  scale_x_continuous(trans = "log2")
```



Scaled data



Original value

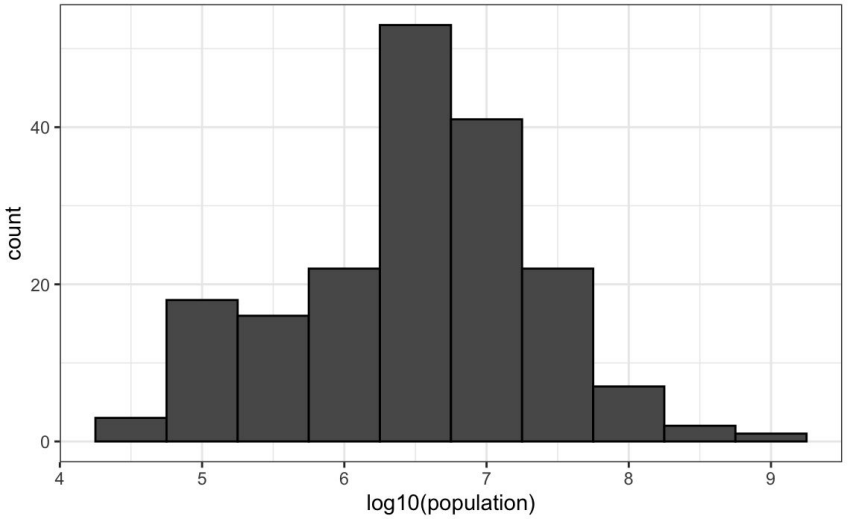
Transformation

How to Choose a Base in Log Transformation

The common choices are the natural log in **base 10, base 2, and base e (the natural log)**. In general, it is **not recommended to use the natural log** for data exploration and visualization because it's not easy to compute and so the scale is not intuitive or easy to interpret.

An example in which base 10 makes more sense than base 2, is consider population size. Using log base 10 makes more sense here since the range for these data goes from 45,000 to about 800 million.

```
filter(gapminder, year == past_year) %>%
  summarize(min = min(population), max =
  max(population))
#>      min      max
#> 1 46075 8.09e+08

gapminder %>%
  filter(year == past_year) %>%
  ggplot(aes(log10(population))) +
  geom_histogram(binwidth = 0.5, color = "black")
```

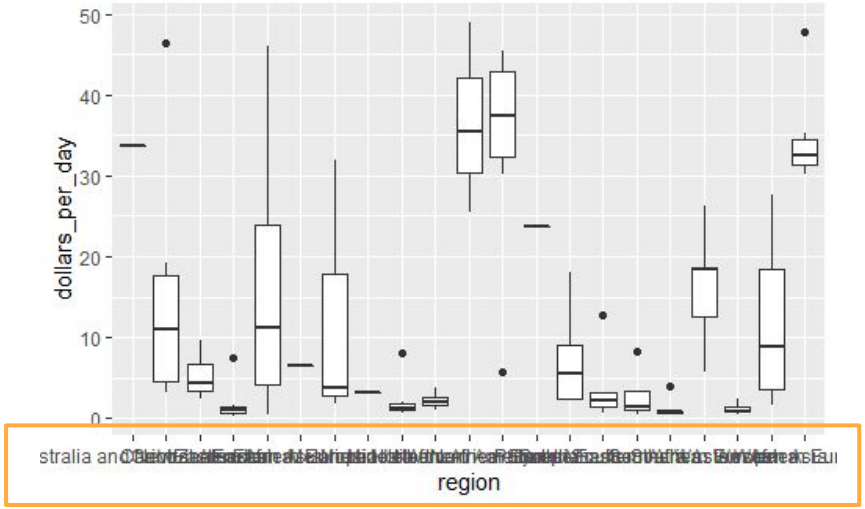


Stratify and Boxplots

Creating a Boxplot that Distributed by Regions

To see distributions by geographical region, we first stratify the data into regions, then examine the distribution for each. Since there are numerous regions, comparing one by one histogram or smooth densities will take a lot of effort. Instead, we can stack **box plots** next to each other.

```
past_year <- 1970
p <- gapminder %>%
    filter(year == past_year & !is.na(gdp)) %>%
    ggplot(aes(region, dollars_per_day))
p + geom_boxplot()
```

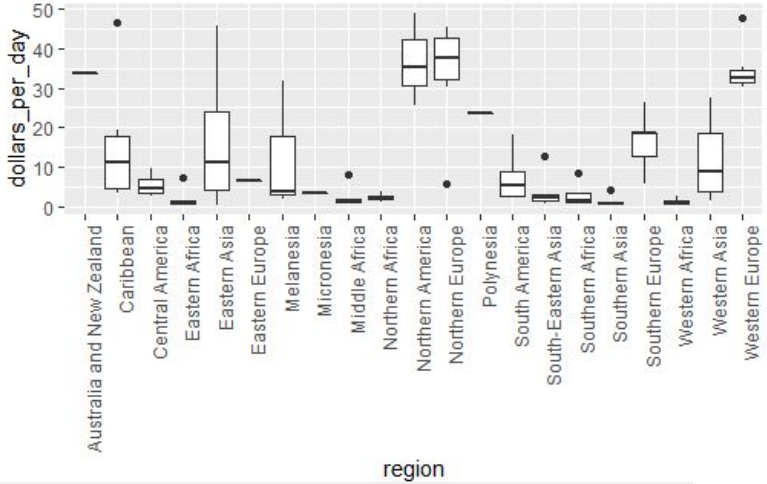


Oops, we can't read the region names :(because the default ggplot behavior is to write the labels horizontally and here we run out of room :((

Stratify and Boxplots

Let me fix you~~

Don’t worry, we can easily fix the problem by rotating the labels by changing the theme through element_text. (WOW)




```
# rotate names on x-axis
p + geom_boxplot() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

justifies this text so that it's next to the axis.

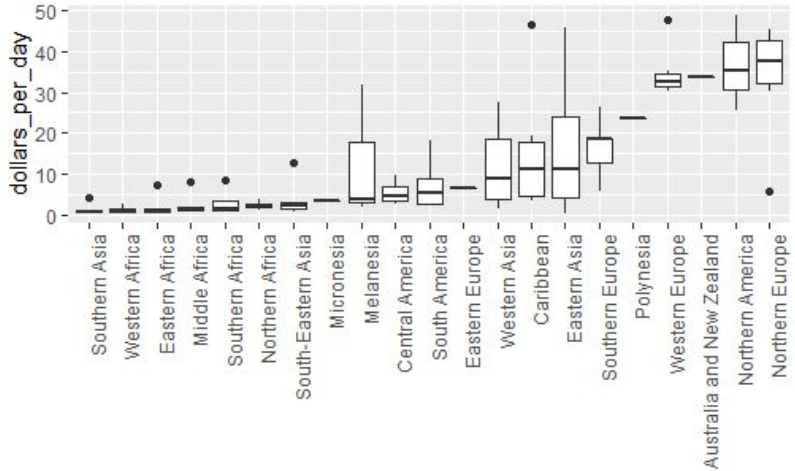
to rotate the text at 90° angle

Stratify and Boxplots

Reorder Function

Next, we want to **reorder the regions by their median income level.** The function that's going to help us achieve this is the **reorder function**. This function lets us change the order of the levels of a factor variable based on a summary computed on a numeric vector. To do this, we we add to mutate that changes region to a new factor where the levels are reordered.

```
p <- gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  mutate(region = reorder(region, dollars_per_day, FUN = median)) %>%    # reorder
  ggplot(aes(region, dollars_per_day)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("")
```

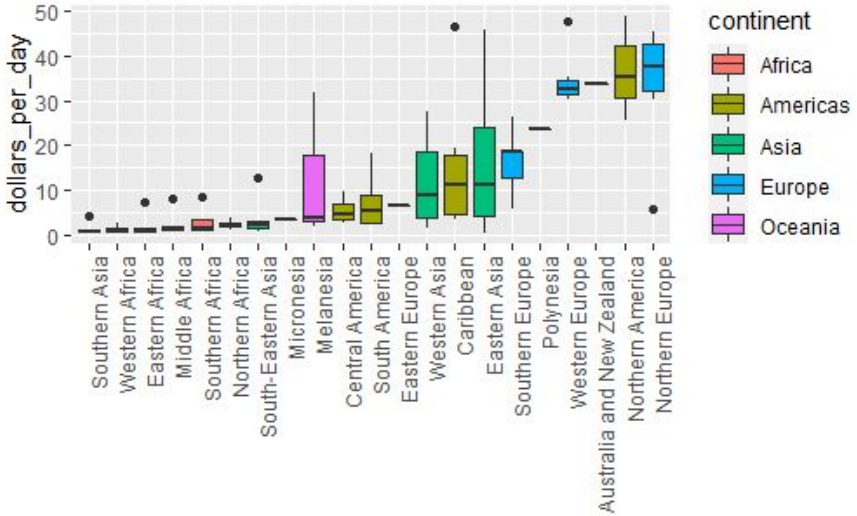


Stratify and Boxplots

Fill Argument

Color is a power which directly influences the soul." ~Wassily Kandinsky. We want to use color to show contingent by using the **fill argument** in the aesthetic mappings of ggplot.

```
p <- gapminder %>%
    filter(year == past_year & !is.na(gdp)) %>%
    mutate(region = reorder(region, dollars_per_day, FUN = median)) %>%    # reorder
    ggplot(aes(region, dollars_per_day, fill = continent)) +    # color by continent
    geom_boxplot() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) + xlab("")
```



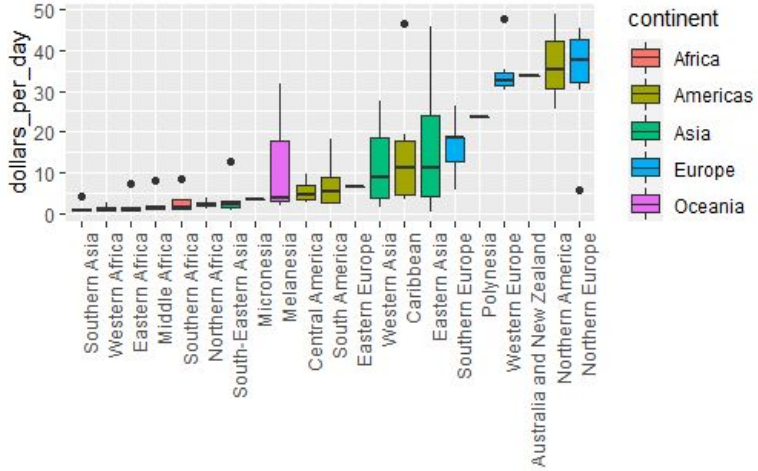
Stratify and Boxplots

Change to the Log Scale

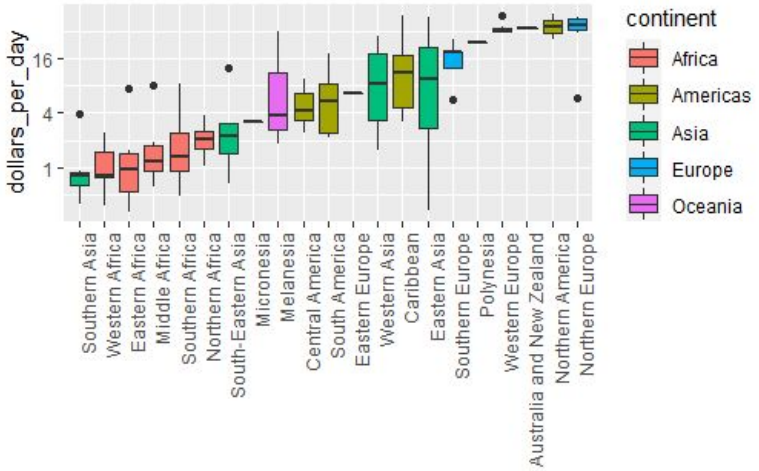
Notice that in the previous picture we hardly see the boxplot in left hand side of the picture, due to the difference range. Changing to the log scale helps us see the differences between the countries with the lower income.

```
p + scale_y_continuous(trans = "log2")
```

Before Log Transformation



After Log Transformation



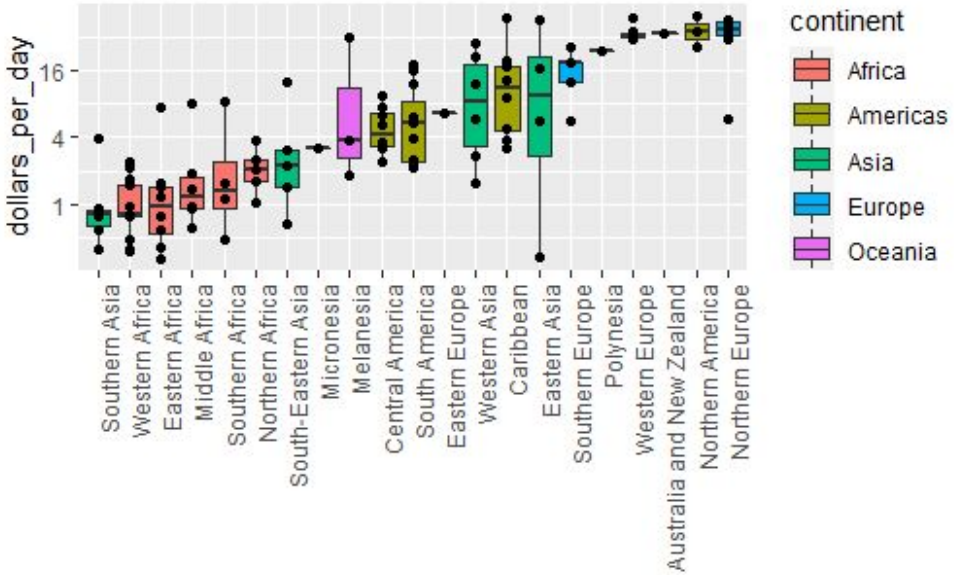
We can see a difference now between the African continent, which is in red, and Asia, which is in green.

Stratify and Boxplots

Show Data Points

To give us even more information, we want to show the data points. In many cases, we don't show the actual individual points, because it adds too much clutter to the plot and it obfuscates the message. But in some cases it is useful to map the actual data.

```
p + scale_y_continuous(trans = "log2")+
    geom_point(show.legend = FALSE)
```



Comparing distributions

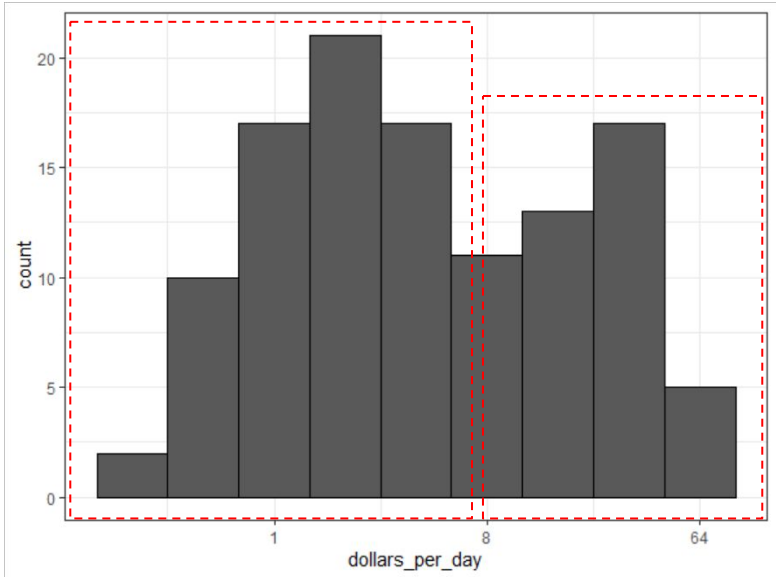
Bimodal distribution in histogram

In the last example, we find out that the spending per-capita (expressed by variable “dollars_per_day”) is **bimodal, meaning it might have two different normal distribution**

```
gapminder %>% filter(year==1970 & !is.na(gdp)) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1,color="black") +
  scale_x_continuous(trans='log2')
```

You’d like to find out why this happened and you hypothesized that *the bimodality happened because there are two different distributions both from the developing and developed countries*. To prove your hypothesis, you have to split the distribution using faceting and then compare the distributions

Spending per-capita is shown by this figure



Comparing distributions

Two different histograms

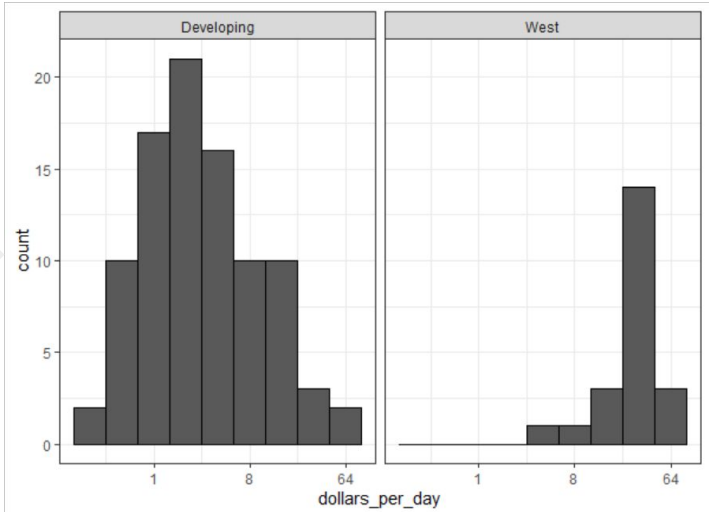
Deriving from the hypothesis, we need to split the distribution by using this function :

```
#1. Define the developed region by grouping these regions
west <- c("Western Europe", "Northern Europe", "Southern Europe",
          "Northern America", "Australia and New Zealand")

#2. Define the year we'd like to compare
past_year <- 1970

#3. Facet the histogram
gapminder %>%
  filter(year==past_year & !is.na(gdp)) %>%
  mutate(group = ifelse(region%in%west,"West","Developing")) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1,color="black") +
  scale_x_continuous(trans="log2")+
  facet_grid(.~group)
```

These region is assumed to be the most developed region in the year of 1970

We use **ifelse** to give ease on defining the rest of developing and developed countries. This function allows us to automatically detect the developing one by just define the list of developed region (re-read about ifelse function in module 1)

The plot should look like this



As we can see, the bimodality is caused by two different distribution from developing and developed country

Comparing distributions

Two different histograms

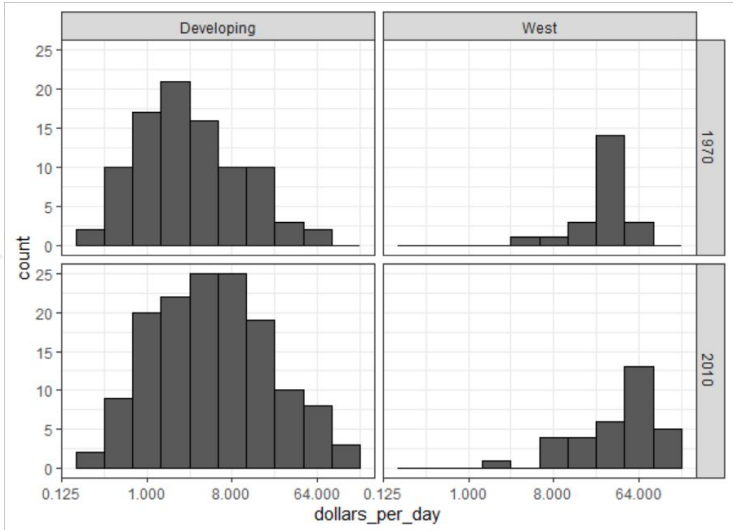
Now suppose you want to compare the condition in 1970 and 2010. You can just define the additional year and facet the row as “year”

```
#1. Define the developed region by grouping these regions
west <- c("Western Europe", "Northern Europe", "Southern Europe",
          "Northern America", "Australia and New Zealand")

#2. Define the year we'd like to compare
past_year <- 1970
present_year <- 2010
years <- c(past_year,present_year)

#3. Facet the histogram
gapminder %>%
  filter(year%in%years & !is.na(gdp)) %>%
  mutate(group = ifelse(region%in%west,"West","Developing")) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1,color="black") +
  scale_x_continuous(trans="log2")+
  facet_grid(year~group)
```

Define the year in a vector

Facet the year as row facet

The plot should look like this



There are shifting of distributions from the developing countries from having a spending mean of $2/person/day in 1970 into having a spending mean somewhere like $7/person/day. It shows us that the economic condition in most developing countries has improved a lot

Comparing distributions

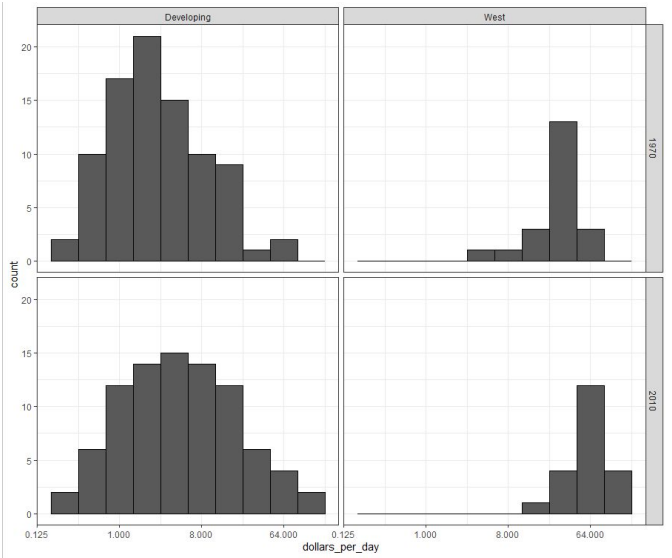
Two different histograms

Note that there are many countries found after 1970, meaning that the distribution in 2010 contains more countries than in 1970, thus makes the comparison to be not “apple to apple”. Hence, we need to compare only the country that has appeared before 1970 and last until 2010

```
#1. Redefine the country list
country_list_1 <- gapminder %>%
  filter(year==1970 & !is.na(gdp)) %>% .$country
country_list_2 <- gapminder %>%
  filter(year==2010 &!is.na(gdp)) %>% .$country
country_list <- intersect(country_list_1,country_list_2)

#2. Define the year we'd like to compare
past_year <- 1970
present_year <- 2010
years <- c(past_year,present_year)

#3. Facet the histogram
gapminder %>%
  filter(year%in%years & country%in%country_list) %>%
  mutate(group = ifelse(region%in%west,"West","Developing")) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1,color="black") +
  scale_x_continuous(trans="log2")+
  facet_grid(year~group)
```

These line of code is used to redefine the country list., **intersect** function is used to find the value that is available in both variable

The plot should look like this



Now we can be pretty sure that the comparison is “apple to apple”.

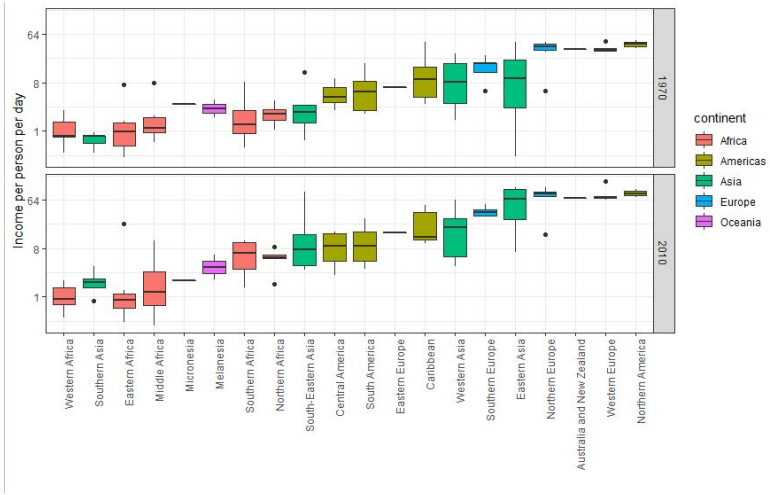
Comparing distributions

Two different boxplots

Suppose we want to see a clearer view about the distribution in each region and compare how it looks like in 1970 and 2010, then we have to show the data using faceted **box plot**

```
#compare boxplot with faceting
gapminder %>% filter(year%in%years & country%in%country_list) %>%
  mutate(region=reorder(region,gdp_person_perday,FUN=median)) %>%
  ggplot(aes(region,gdp_person_perday,fill=continent)) +
  geom_boxplot() +
  theme(axis.text.x=element_text(angle=90,hjust=1)) +
  xlab(" ") +
  ylab("Income per person per day") +
  scale_y_continuous(trans="log2") +
  facet_grid(year~.)
```

The plot should look like this



Comparing distributions

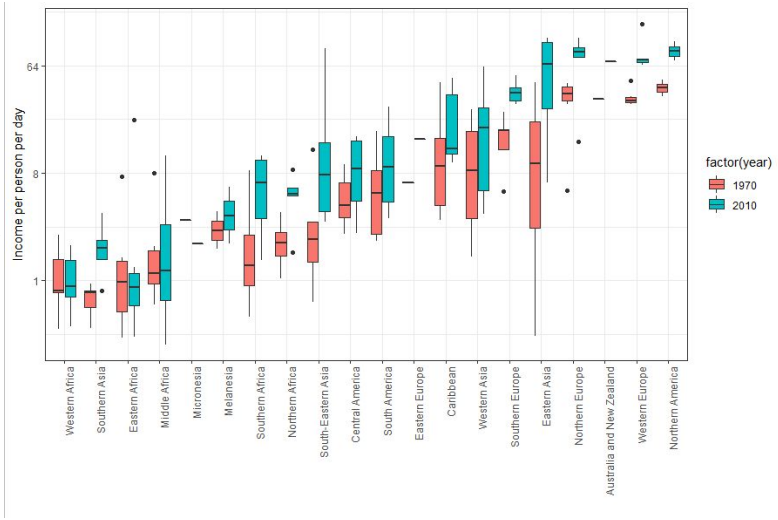
Boxplot - ease comparison

Facet isn't giving us a good comparison since we don't really know which region improved the most. To give ease of comparison, we need to plot the box plot next to each other,

```
#compare boxplot without faceting
gapminder %>% filter(year%in%years & country%in%country_list) %>%
  mutate(region=reorder(region,gdp_person_perday,FUN=median)) %>%
  ggplot(aes(region,gdp_person_perday,fill=factor(year))) +
  geom_boxplot() +
  theme(axis.text.x=element_text(angle=90,hjust=1)) +
  xlab(" ") +
  ylab("Income per person per day") +
  scale_y_continuous(trans="log2")
```

We need to give color to the plot based on the year to easily recognised the different type of year. *Ggplot will automatically assign a color to each level of factor*. However, the year is actually a number, so we need to turn it as a factor by using **factor(year).**

The plot should look like this



As shown on the plot, we can see that most of the region (from 1970 to 2010) has economically improved. We also see that eastern asia has been drastically improved.

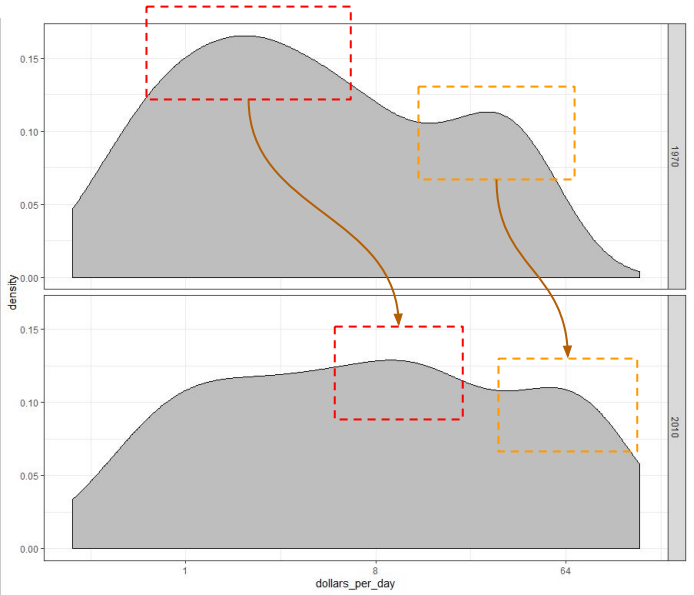
Density Plots

Two different density plots

Suppose we'd like to examine the spending/person/day in 1970 and 2010 by using density plot. Here's the code :

```
#1. Redefine the country list
country_list_1 <- gapminder %>%
  filter(year==1970 & !is.na(gdp)) %>% .$country
country_list_2 <- gapminder %>%
  filter(year==2010 &!is.na(gdp)) %>% .$country
country_list <- intersect(country_list_1,country_list_2)

#2. Define the year we'd like to compare
past_year <- 1970
present_year <- 2010
years <- c(past_year,present_year)

#3 conduct the faceted histogram
gapminder %>% filter(year%in%years & country%in%country_list) %>%
  ggplot(aes(dollars_per_day)) +
  geom_density(fill="grey") +
  scale_x_continuous(trans="log2") +
  facet_grid(year~.)
```

The plot should look like this



As shown on the plot, we'd also see the shift of spending/person/day from 1970 to 2010 both in developed and developing regions. However, we need to delve deeper to see variability within these region by splitting the developed and developing region

Density Plots

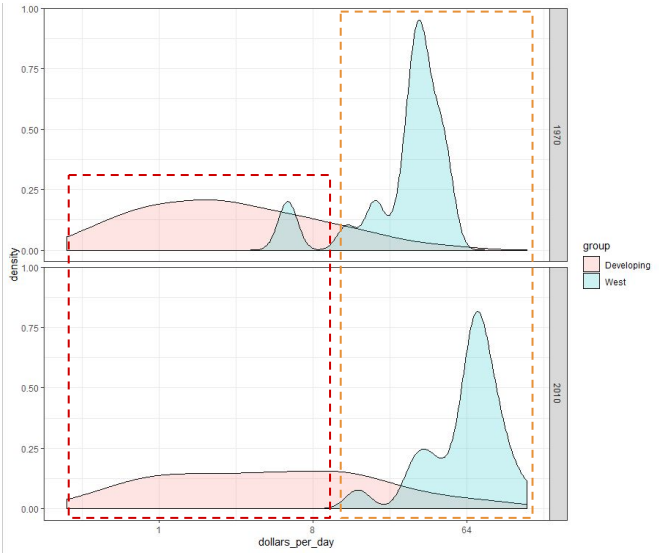
Two different density plots

1 To examine the distribution for developed and developing region separately, here's the code :

```
# conduct the faceted and layered density plot
gapminder %>% filter(year%in%years & country%in%country_list) %>%
  mutate(group = ifelse(region%in%west,"West","Developing")) %>%
  ggplot(aes(x=dollars_per_day,fill=group)) +
  geom_density(alpha=0.2) +
  scale_x_continuous(trans="log2") +
  facet_grid(year~.)
```

fill=group is used to differentiate both distribution. No need to separate group into a column facet because we need to examine them in a single column

The plot should look like this



2 There's a problem, the number of developed and developing regions are different (*see fig. 1*) ...

```
gapminder %>%
  filter(year == past_year & country %in% country_list) %>%
  mutate(group=ifelse(region %in% west, "West", "Developing")) %>%
  group_by(group) %>%
  summarize(n = n()) %>% knitr::kable()
```

Fig. 1

group	n
Developing	87
West	21

3 ... The different number of country makes the distribution to be a bit unequal in shape because the nature of density plot to sum up the area to 1

How to address this issue ? **See the next slide**

Density Plots

Two different density plots

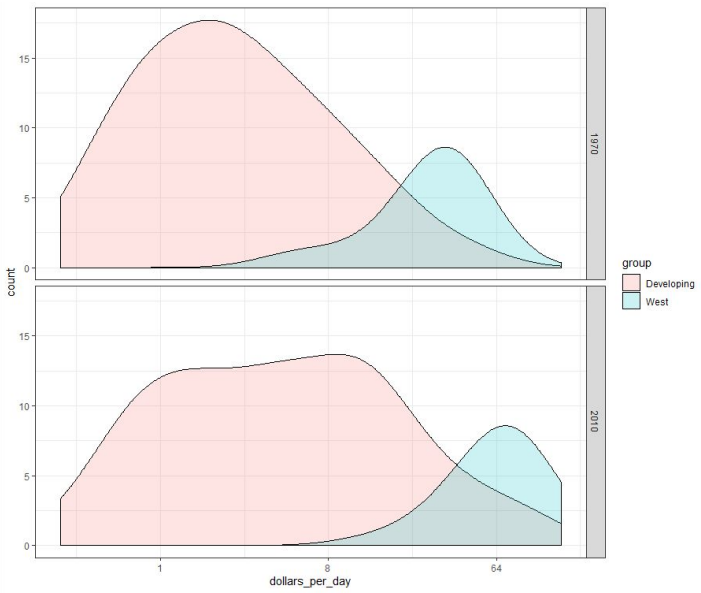
To address the issue that previously explained, then we used **..count..**
Here's the code :

```
# using the ..count.. and bw argument
gapminder %>% filter(year%in%years & country%in%country_list) %>%
  mutate(group = ifelse(region%in%west,"West","Developing")) %>%
  ggplot(aes(x=dollars_per_day,y=..count..,fill=group)) +
  geom_density(alpha=0.2, bw=0.75) +
  scale_x_continuous(trans="log2") +
  facet_grid(year~.)
```

bw=0.75 Is used to make the curve looks a bit more smooth

..count.. Is used to grouped both the calculation of density so the density calculation of developed + developing region can be up to 1

The plot should look like this



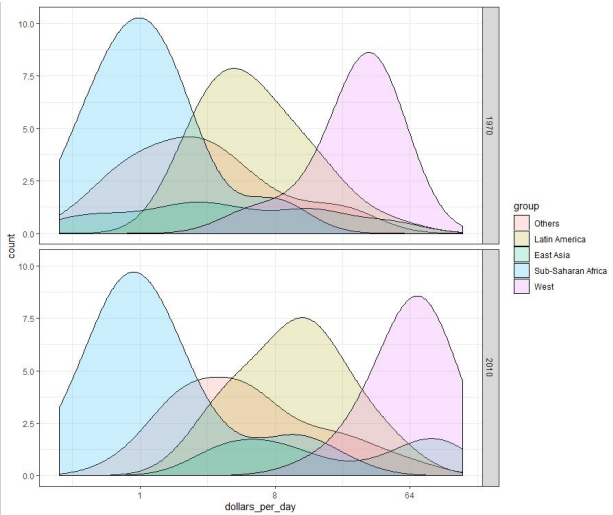
Comparing distributions

Two different histograms

Suppose we'd like to split the distribution into several regions to see the shift in each region better. Here's the code

```
# add group as a factor, grouping regions
gapminder <- gapminder %>%
  mutate(group = case_when(
    .$region %in% west ~ "West",
    .$region %in% c("Eastern Asia", "South-Eastern Asia") ~ "East Asia",
    .$region %in% c("Caribbean", "Central America",
                    "South America") ~ "Latin America",
    .$continent == "Africa" & .$region != "Northern Africa" ~ "Sub-Saharan Africa",
    TRUE ~ "Others"))

# plot the data
gapminder %>% filter(year%in%years & country%in%country_list) %>%
  mutate(group = factor(group, levels = c("Others", "Latin America",
                                          "East Asia", "Sub-Saharan Africa",
                                          "West"))) %>%
  ggplot(aes(x=dollars_per_day,y=..count..,fill=group)) +
  geom_density(alpha=0.2, bw=0.75) +
  scale_x_continuous(trans="log2") +
  facet_grid(year~.)
```

The plot should look like this



We define the group as the region that we defined above, and called them as **factor.** You can see more explanation by typing "**?factor**"

case_when is used to define several region into our definition. I.e we define the eastern asia & southern asia as "east asia", etc

Density Plots

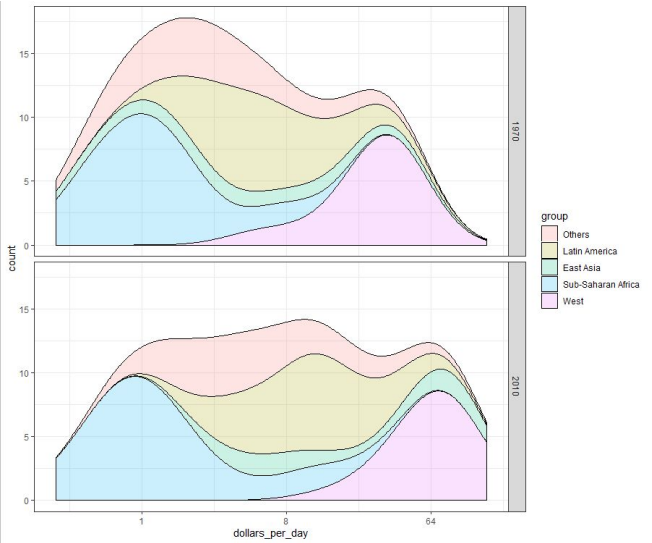
Two different density plots

The plot we previously made was a bit hard to see, thus we need to stacked them with the help of "**position="stack"**"

```
gapminder %>% filter(year%in%years & country%in%country_list) %>%
  mutate(group = factor(group, levels = c("Others", "Latin America",
                                          "East Asia", "Sub-Saharan Africa", "West")))
%>%
  ggplot(aes(x=dollars_per_day,y=..count..,fill=group)) +
  geom_density(alpha=0.2, bw=0.75, position="stack") +
  scale_x_continuous(trans="log2") +
  facet_grid(year~.)
```

position="stack" is used to make the density plot to be above each other, thus make them to be easily examined

The plot should look like this



Statistic Quality & Engineering
Universitas Indonesia

Content of this module :

1. Introduction to ggplot2
2. Summarizing with dplyr
3. Example of Data Visualization using Gapminder Data
4. Data Visualization Principles

This is what you will learn in this section...

Section 4 - overview

Section	What you will learn...
4.1	• Encoding data using visual cues • Know when to include 0 • Do not distort quantities • Order categories by a meaningful value
4.2	• Show the data • Ease comparisons
4.3	• Slope charts • Avoid pseudo 3D plots • Avoid too many significant digits

Data Visualization Principles

Introduction

Before we get started, note that

- The principles discussed in this section mostly based on research related to how humans detect patterns and make visual comparisons.
- The preferred approaches are those that best fit the way our brains process visual information.
- When deciding on a visualization approach, it is also important to keep our goal in mind.

In this section, we will use these libraries:

```
> library(tidyverse)
> library(dslabs)
> library(girdExtra)
```

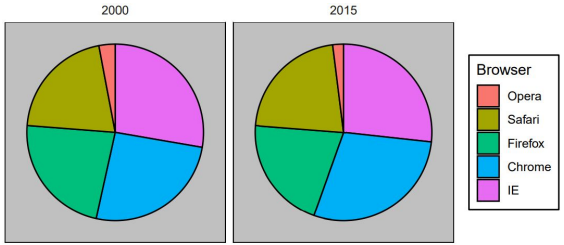
There are several approaches that can be considered in visualizing data, including:

1. **Position**
2. **Angles**
3. **Area**
4. **Brightness**
5. **Color line**

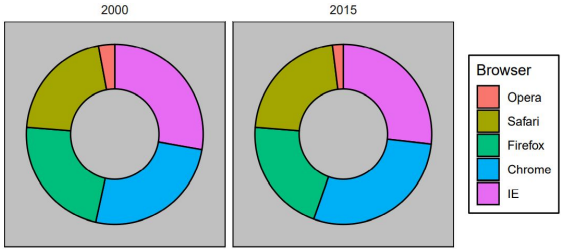
Data Visualization Principles

Section 4.1 - *Encoding data using visual cues*

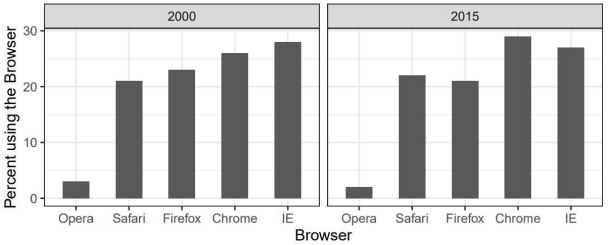
We have discussed in the previous section that several approaches that can be considered in this section are: **position**, **angles**, **area**, **brightness**, and **color line**. For example, suppose we want to report the results from two polls regarding browser preference taken in 2000 and then 2015. For each year, **we are simply comparing four quantities** – percentages. There are 3 charts that are commonly used to visualize this kind of data:



A widely used graphical representation of percentages, popularized by Ms Excel, is the **pie chart**. Here both the angle and area of each pie slice are proportional to the quantity the slice represents. This turns out to be a sub-optimal choice since humans are **not good** at precisely quantifying angles and areas.



Humans are **even worse** when area is the only available visual cue. **Donut chart** is an example of a plot that uses only areas. To see how hard it is to quantify area, note that the rankings and all the percentages in the plots above changed from 2000 to 2015 but you can't know exactly how big the changes are.



The **preferred** way to plot this kind of data is to use length and position as visual cues, since humans are much better at judging linear measures. The **barplot** uses this approach by using bars of length proportional to the quantities of interest. Compare the **barplot** with two previous charts, can you see the differences?

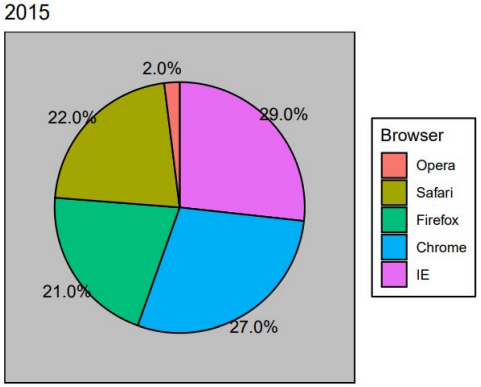
Data Visualization Principles

Section 4.1 - *Encoding data using visual cues*

Actually in this case, you have two more alternatives to visualize the data:

Browser	2000	2015
Opera	3	2
Safari	21	22
Firefox	23	21
Chrome	26	29
IE	28	27

You can simply show the numbers but it isn't visually appealing and would be difficult if we have a large amount of data



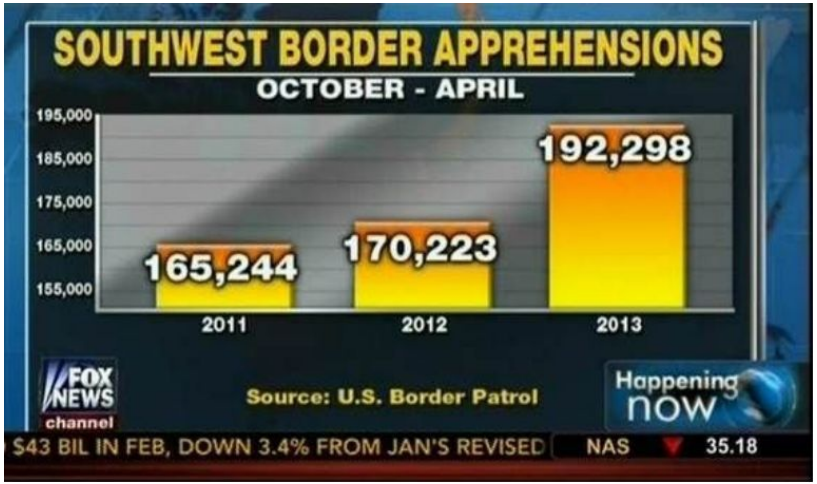
Or w/ a pie chart but label each slice with its respective percentage so viewers don't have to infer them from the angles or area

In general, when displaying quantities, position and length are preferred over angles and area. Brightness and color are even harder to quantify than angles. However, they are sometimes useful when more than two dimensions must be displayed at once

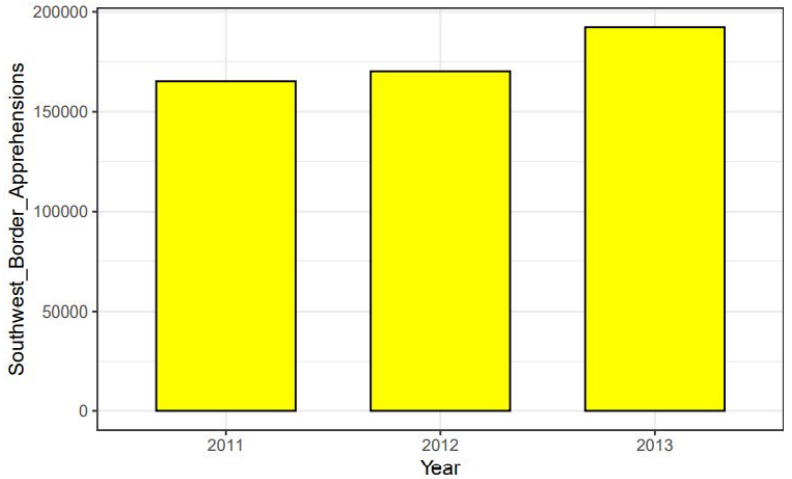
Data Visualization Principles

Section 4.1 - *Know when to include 0*

When using barplots, it is **dishonest** not to start the bars at 0. This is because, by using a barplot, we are implying the length is proportional to the quantities being displayed.



Some people avoid using 0 to make relatively small differences look much bigger than they actually are. This approach is often used by politicians or media organizations to lead an opinion.

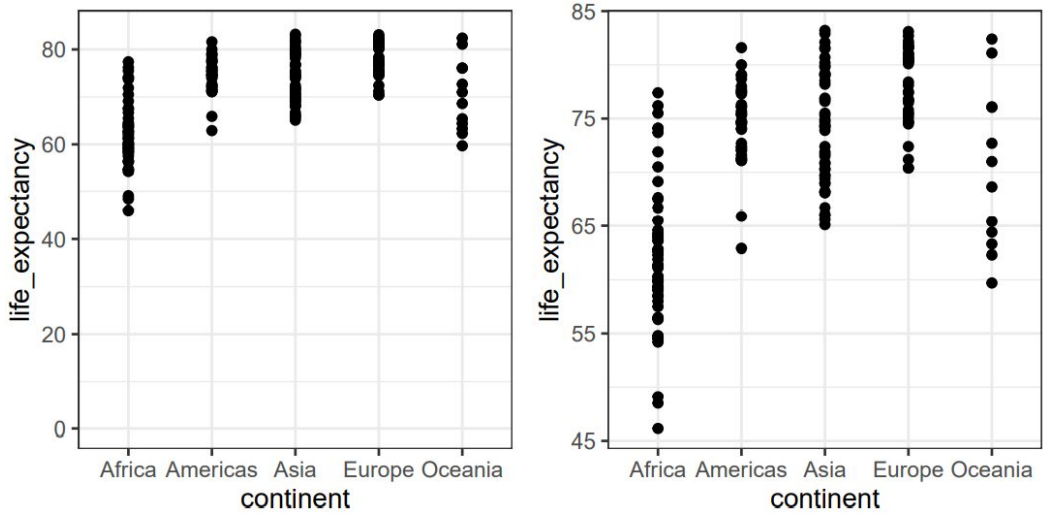


Can you see the differences? From the plot before, it appears that “apprehensions” have almost tripled, while, in fact, they have only increase about 16% as shown in chart above,

Data Visualization Principles

Section 4.1 - *Know when to include 0*

Otherwise, when using position rather than length, it is then **not necessary** to include 0. This is particularly the case when we want to compare differences between groups relative to the within group variability. Here is an illustrative example showing country average life expectancy stratified across continents in 2012



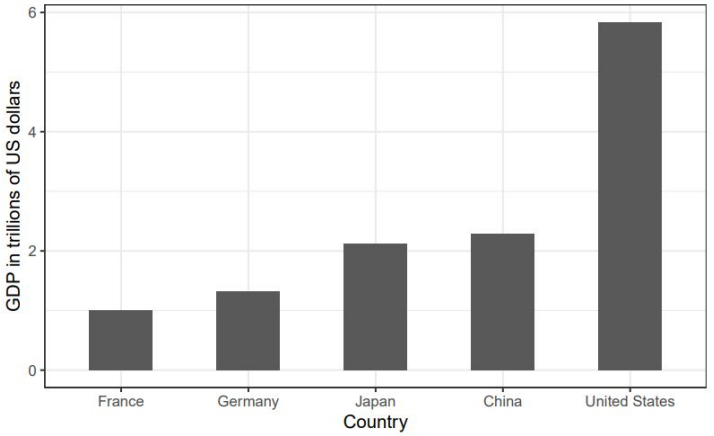
Note that in the plot on the left, which includes 0, the space between 0 and 43 adds no information and makes it harder to compare the between and within group variability

Data Visualization Principles

Section 4.1 - *Do not distort quantities*

To understand what it means, consider the following example

Judging by the area of the circles, the US seems to have an economy over five times larger than China's and over 30 times larger than France's. However, if we look at the actual numbers, we see that the actual ratios are 2.6 and 5.8 times bigger respectively. The reason for this distortion is that the radius, rather than the area, was made to be proportional to the quantity, which implies that the proportion between the areas is squared: 2.6 turns into 6.5 and 5.8 turns into 34.1

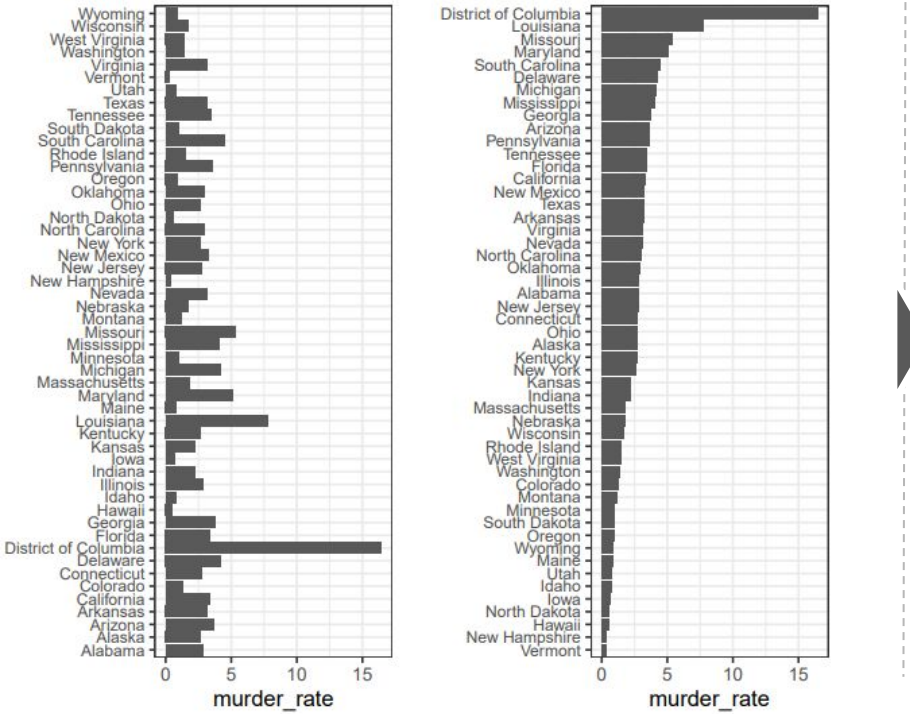


Based on previous example, **ggplot2** defaults to using area rather than radius. But of course, in this case, we really should not be using area at all since we can use position and length. You can see that this **barplot** is more preferred way to visualize that kind of data, as we discussed in previous section.

Data Visualization Principles

Section 4.1 - *Order categories by a meaningful value*

ggplot2 behavior is to order the categories alphabetically. However, we rarely want to use alphabetical order. Instead, we should order by meaningful value.



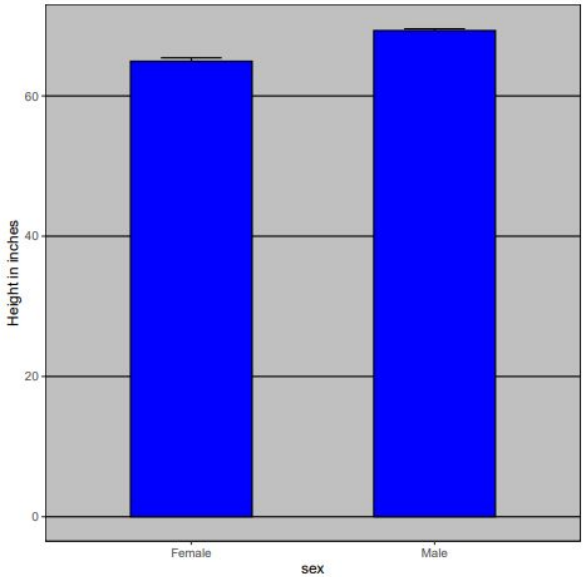
To appreciate how the right order can help convey a message, look at both charts on the left. We are particularly interested in the most dangerous and safest states. Note the difference when we order alphabetically (the default) versus when we order by the actual rate. Can you see the differences? To make the second plot:

```
data(murders)
murders %>% mutate(murder_rate = total / population * 100000) %>%
  mutate(state = reorder(state, murder_rate)) %>%
  ggplot(aes(state, murder_rate)) +
  geom_bar(stat="identity") +
  coord_flip() +
  theme(axis.text.y = element_text(size = 6)) +
  xlab("")
```

Data Visualization Principles

Section 4.2 - *Show the data*

We have focused on displaying single quantities across categories. We now shift our attention to displaying data, with a focus on comparing groups.



This time let's assume we interested in the difference in heights between males and females from "heights" dataset. A commonly seen plot used for comparisons between groups, popularized by software such as Microsoft Excel, is the **dynamite plot**, which shows the average and standard errors. The average of each group is represented by the top of each bar and the antennae extend out from the average to the average plus two standard errors. However, it is difficult to interpret.

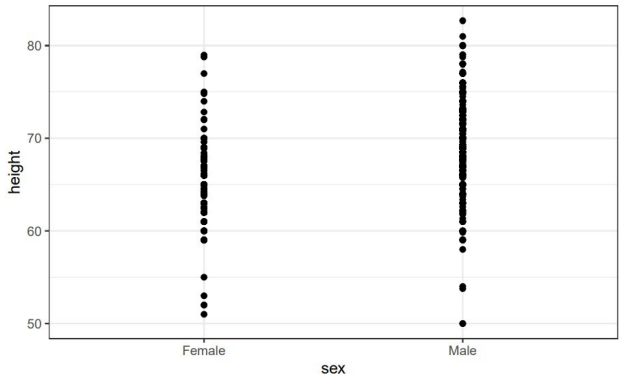
Data Visualization Principles

Section 4.2 - *Show the data*

Fortunately, **ggplot2** code already generates a more informative plot than the barplot by simply showing all the data points. Simply by writing this code:

```
heights %>%
  ggplot(aes(sex, height)) +
  geom_point()
```

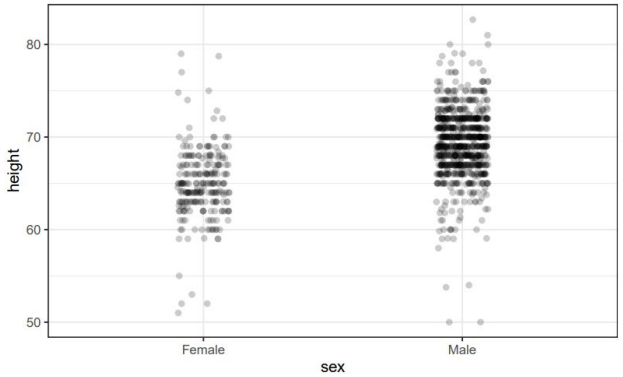
It gives us plot looks like this:



That plot gives us an idea of the range of the data. However, this plot has limitations as well, since we can't really see all the 238 and 812 points plotted for females and males, respectively, and many points are plotted on top of each other. Fortunately, we point out two ways we can improve a plot showing all the points.

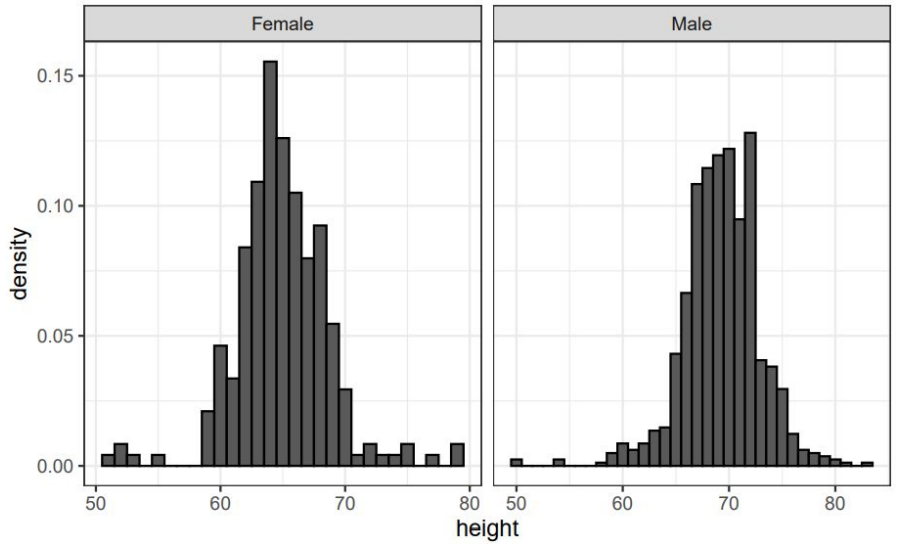
1. Jitter: adds small random shift to each point
2. Alpha Blending: making the points somewhat transparent

```
heights %>%
  ggplot(aes(sex, height)) +
  geom_jitter(width = 0.1, alpha = 0.2)
```



Data Visualization Principles

Section 4.2 - *Ease comparisons: use common axes*

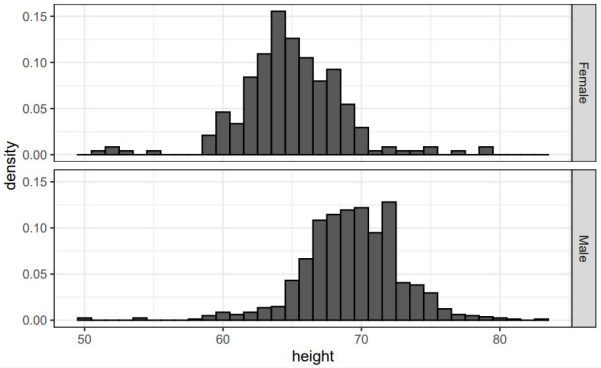


An important principle here is to keep the axes the same when comparing data across two plots. Here we see how the comparison becomes easier to compare since both have the same scale on each axes

Data Visualization Principles

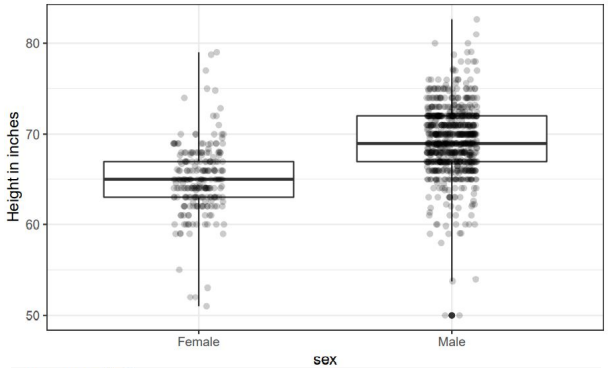
Section 4.2 - *Ease comparisons: align plots*

In this section, the only rule we care about is "**Align plots vertically to see horizontal changes and horizontally to see vertical changes**"




```
heights %>%
  ggplot(aes(height, ..density..)) +
  geom_histogram(binwidth = 1, color="black") +
  facet_grid(sex~.)
```

In these histograms, the visual cue related to decreases or increases in height are shifts to the left or right, respectively: horizontal changes. Aligning the plots vertically helps us see this change when the axes are fixed

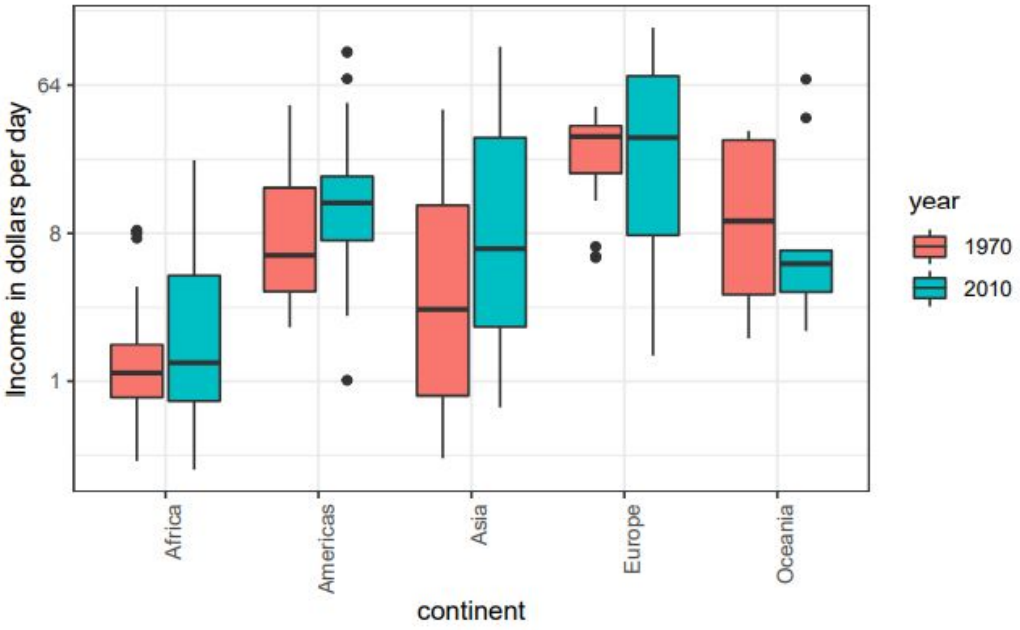



```
heights %>%
 ggplot(aes(sex, height)) +
 geom_boxplot(coef=3) +
 geom_jitter(width = 0.1, alpha = 0.2) +
 ylab("Height in inches")
```

If we want the more compact summary provided by boxplots, we then align them horizontally since, by default, boxplots move up and down with changes in height. Following our show the data principle, we then overlay all the data points

Data Visualization Principles

Section 4.2 - *Ease comparisons: use color*



The comparison becomes easier to make if we use color to denote the two things we want to compare. Can you sense it?

However, remember that about 10% of population is color blind. Fortunately, **ggplot2** does make it easy to change the color palette used in the plot.

```
color_blind_friendly_cols <-
  c("#999999", "#E69F00", "#56B4E9", "#009E73",
    "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
```

Here is the color:

There are some resources that can help us select colors:

- http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/#a-colorblindfriendly-palette
- http://bconnelly.net/2013/10/creating-colorblind-friendly-figures/

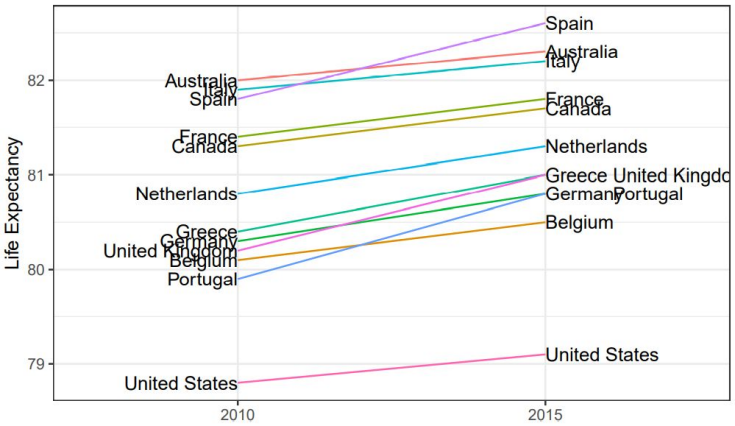
Data Visualization Principles

Section 4.3 - *Use slope chart*

In general, we should use **scatterplots** to visualize the relationship between two variables. However, there are some exceptions and we have two alternative plots here: **the slope chart** and the **Bland-Altman plot**. Note that we will discuss only **the slope chart** here.

The slope chart may be more informative when we are comparing variables of the same type, but at different time points and for a relatively small number of comparisons. There is no geometry for slope charts in **ggplot2**, but we can construct it using `geom_line`.

```
west <- c("Western Europe","Northern Europe","Southern Europe",
+         "Northern America","Australia and New Zealand")
dat <- gapminder %>%
+  filter(year%in% c(2010, 2015) & region %in% west &
+           !is.na(life_expectancy) & population > 10^7)
dat %>%
+  mutate(location = ifelse(year == 2010, 1, 2),
+         location = ifelse(year == 2015 &
+            country %in% c("United Kingdom", "Portugal"),
+                           location+0.22, location),
+         hjust = ifelse(year == 2010, 1, 0)) %>%
mutate(year = as.factor(year)) %>%
ggplot(aes(year, life_expectancy, group = country)) +
geom_line(aes(color = country), show.legend = FALSE) +
geom_text(aes(x = location, label = country, hjust = hjust),
show.legend = FALSE) +
xlab("") + ylab("Life Expectancy")
```

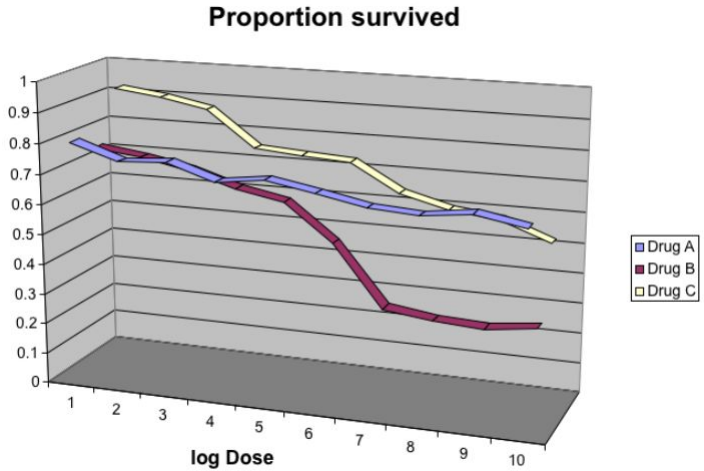


An advantage of the **slope chart** is that it permits us to quickly get an idea of changes based on the slope of the lines. Although we are using angle as the visual cue, we also have position to determine the exact values. Can you see how it is more appropriate than scatterplots?

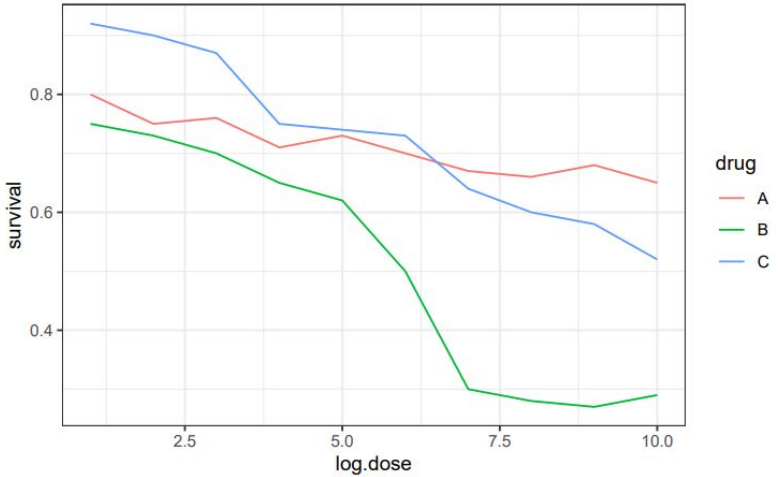
Data Visualization Principles

Section 4.3 - *Avoid pseudo 3D plots*

Humans are not good at seeing in three dimensions and our limitation is even worse with regard to pseudo-three-dimensions. Consider this following example:



To see how hard it is, try to determine the values of the survival variable in the plot above. Can you tell when the purple ribbon intersects the red one?



This is an example in which we can easily use color to represent the categorical variable instead of using a pseudo-3D. Notice how much easier it is to determine the survival values

Data Visualization Principles

Section 4.3 - *Avoid too many significant digits*

By default, statistical software like R returns many significant digits. The default behavior in R is to show 7 significant digits. That many digits often adds no information and the added visual clutter can make it hard for the viewer to understand the message.

state	year	Measles	Pertussis	Polio
California	1940	37.8826320	18.3397861	18.3397861
California	1950	13.9124205	4.7467350	4.7467350
California	1960	14.1386471	0.0000000	0.0000000
California	1970	0.9767889	0.0000000	0.0000000
California	1980	0.3743467	0.0515466	0.0515466

We are reporting precision up to 0.00001 cases per 10,000, a very small value in the context of the changes that are occurring across the dates. In this case, two significant figures is more than enough and clearly makes the point that rates are decreasing, see the following table to help you understand this.

state	year	Measles	Pertussis	Polio
California	1940	37.9	18.3	18.3
California	1950	13.9	4.7	4.7
California	1960	14.1	0.0	0.0
California	1970	1.0	0.0	0.0
California	1980	0.4	0.1	0.1

It contains almost exactly same insights right? Useful ways to change the number of significant digits or to round numbers are signif and round. You can define the number of significant digits globally by setting options like this: `options(digits = 3)`.

Statistic Quality & Engineering
Universitas Indonesia

Assignment

Due date on Oct 18th, 2020 at 23.59 WIB

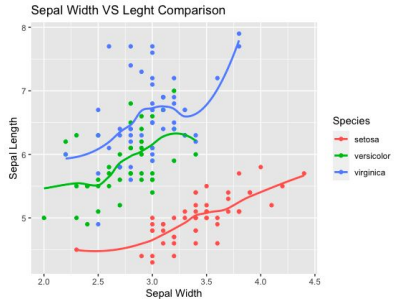
1. Load the dataset **iris**.
   Create a single line code to produce a data frame called **df_iris** containing the average and standard deviation of sepal length for each species. Use the column names ‘average’ and ‘standard_deviation’ for df_iris

2. From the dataset **iris**, display the top 15 widest petal in a descending order. How many of each species made it to the top 15 widest petal list?

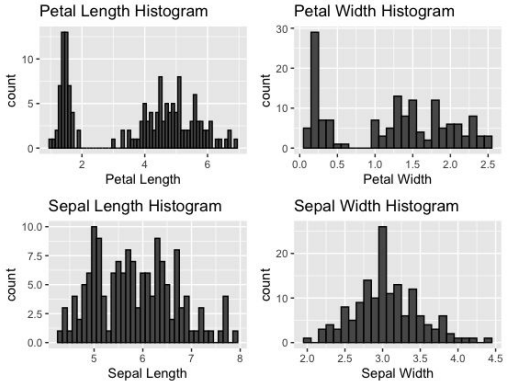
Assignment

Due date on Oct 18th, 2020 at 23.59 WIB

3 From the dataset **iris**, please assess several comparison such as Sepal Width VS Sepal Length



4 From the dataset **iris**, try to call out library(grid) to ease you make these graph. Make histogram for each Sepal Length, Sepal Width, Petal Length, and Petal Width



Assignment

Due date on Oct 18th, 2020 at 23.59 WIB

Answer the questions and explain your answers in a **R Markdown**, please include the question number.

Your R Markdown should be like the following:

```
1    # Question 1
2    # Explain your answer
3    Function(x)
```

Send your assignment with the format (for both email subject and file name): **Tugas 2_Statdas 01/02/03/KKI_Group x.zip** (including both the **Html file** and **RMD file**, grouped in a zipped folder) to **sqe.lab.ie.ui@gmail.com**

Please submit your own work without **plagiarizing others'**, If we detect any plagiarism, we will cut up your score

“In god we trust, all others must bring data” - W. Edward Demings

Thank You

SQE - 2020