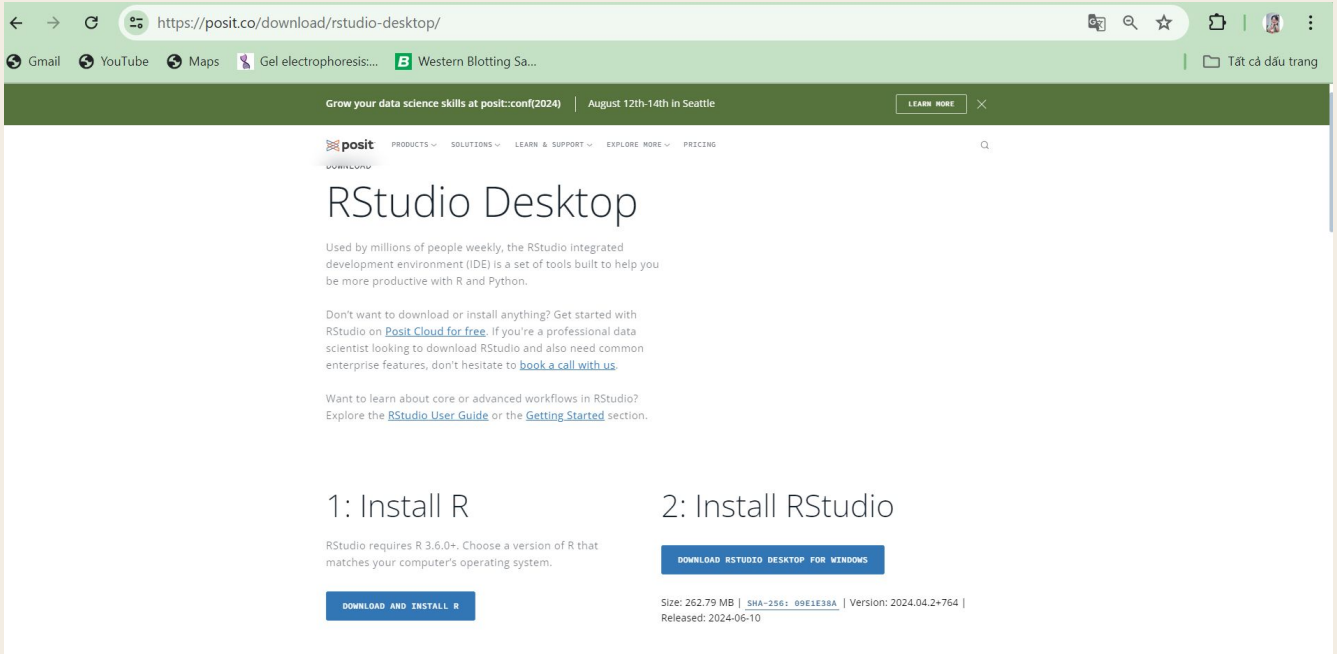
Introduction to R for Microbiome Data (part 1)

Presenter: Van Ho Hoang Kim
30/06/2024

CONTENT



https://posit.co/download/rstudio-desktop/

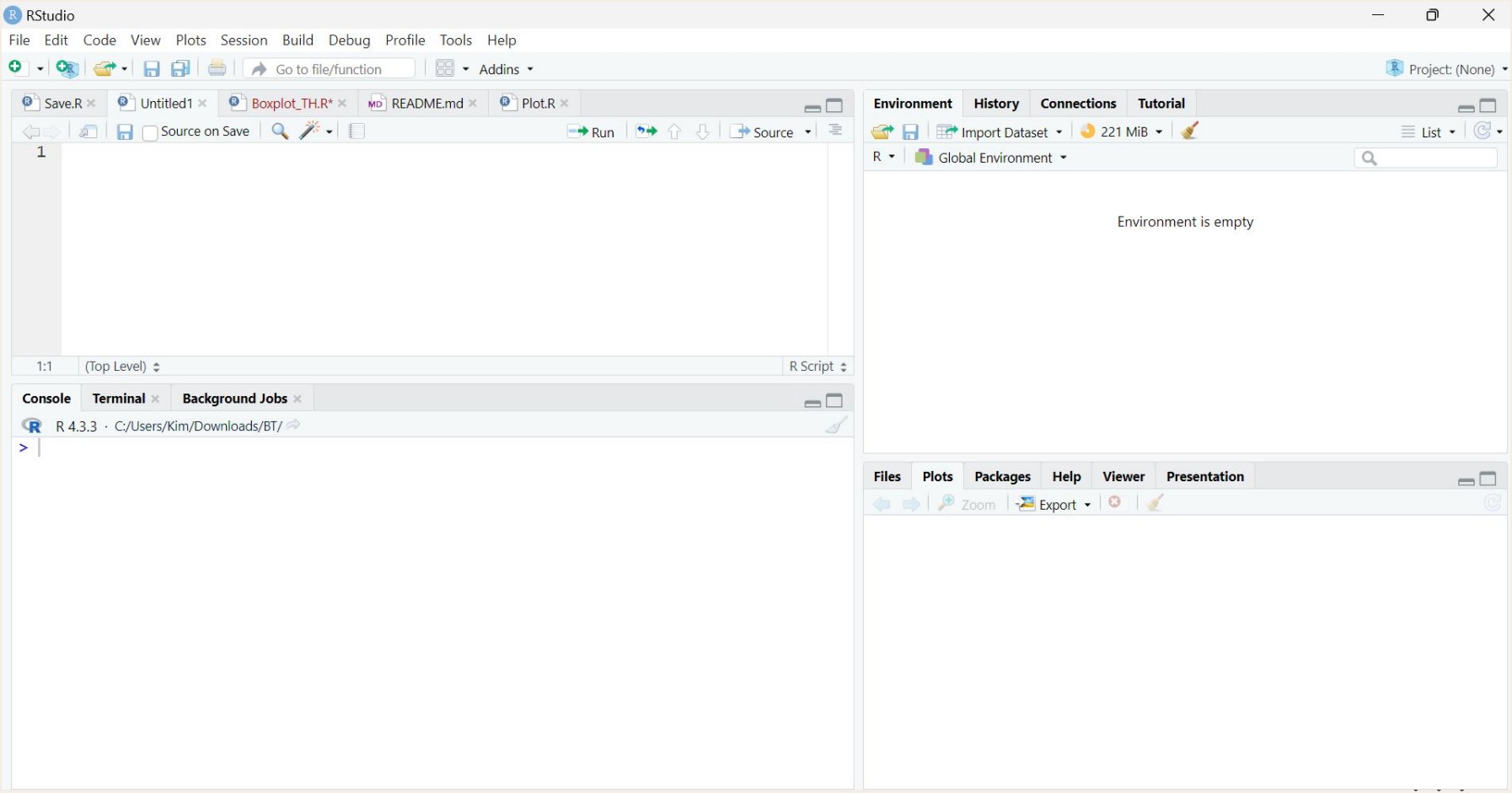
https://www.youtube.com/watch?v=dsjF57fjL40&list=PLMIaO-u3S5-jO2rMt8r8HD5ifiZv_Sd9O
https://www.youtube.com/watch?v=ngpB_00Jiyk&t=825s

RGui (64-bit)

File Edit View Misc Packages Windows Help

R Console

```
R version 4.3.3 (2024-02-29 ucrt) -- "Angel Food Cake"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

>
```


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Save.R
Untitled1
Boxplot_TH.R*
README.md
Plot.R
Source on Save
Run
Source
1
1:1
(Top Level)
R Script
Console
Terminal
Background Jobs
R 4.3.3 · C:/Users/Kim/Downloads/BT/
>
Environment
History
Connections
Tutorial
Import Dataset
221 MiB
List
R
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Presentation
Zoom
Export

1. Some Useful R Functions

1.1. Use SaveRDS() to Save Single R Object

```
# Save an object to a file
saveRDS(object, file = "my_data.rds")

# Restore the object
readRDS(file = "my_data.rds")
```

- object: An R object to save
- file: the name of the file where the R object is saved to or read from

Example:

```
# Saving data as RDS
saveRDS(microbiome_data,"microbiome_data.rds")

# Reading data from RDS file
readRDS("microbiome_data.rds")

# Restore it under a different name
loaded_data <- readRDS("microbiome_data.rds")
head(loaded_data)
```

1.2. Use Save() to Save Multiple R Objects

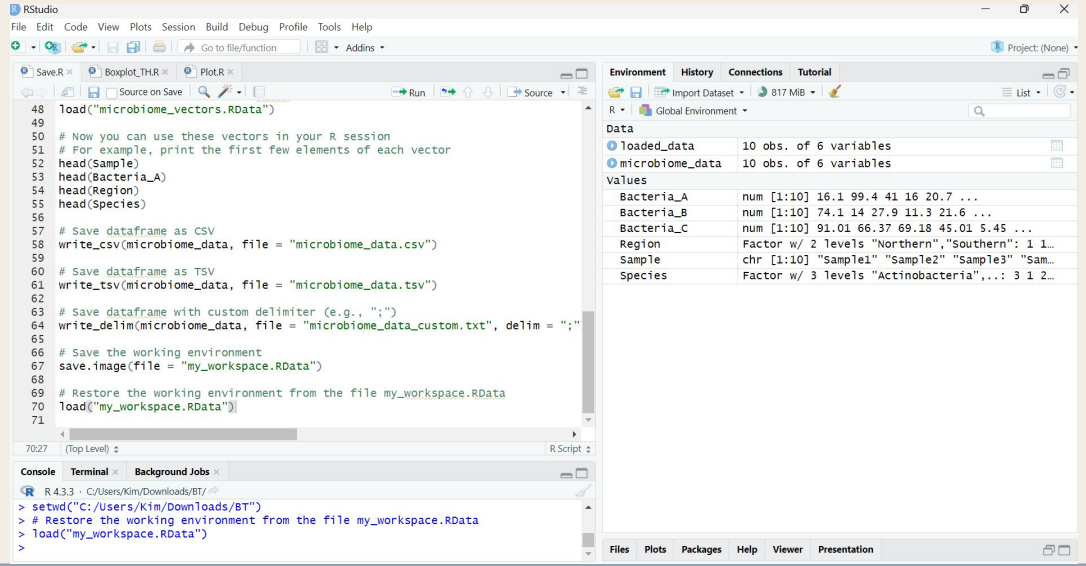
```
# Saving on object in RData format
save(data1, file = "data.RData")
# Save multiple objects
save(data1, data2, file = "data.RData")
# To load the data again
load("data.RData")
```

Example:

```
> print(microbiome_data)
     Sample Bacteria_A Bacteria_B Bacteria_C   Region        Species
1   Sample1      16.14      74.12      91.01 Northern     Firmicutes
2   Sample2      99.43      14.00      66.37 Northern Actinobacteria
3   Sample3      40.97      27.95      69.18 Northern    Bacteroides
4   Sample4      16.02      11.35      45.01 Northern    Bacteroides
5   Sample5      20.66      21.61       5.45 Northern    Bacteroides
6   Sample6      22.11      99.94      35.16 Southern    Bacteroides
7   Sample7      61.11      63.28      30.94 Southern Actinobacteria
8   Sample8      60.13      15.23      48.72 Southern     Firmicutes
9   Sample9      46.08      86.54      54.43 Southern Actinobacteria
10 Sample10      39.31      59.18      49.98 Southern     Firmicutes
```

```
# Saving each column of microbiome data as individual vectors
Sample <- microbiome_data$Sample
Bacteria_A <- microbiome_data$Bacteria_A
Bacteria_B <- microbiome_data$Bacteria_B
Bacteria_C <- microbiome_data$Bacteria_C
Region <- microbiome_data$Region
Species <- microbiome_data$Species
# Checking the data types of each vector
class(Sample)     # "character"
class(Bacteria_A) # "numeric"
class(Bacteria_B) # "numeric"
class(Bacteria_C) # "numeric"
class(Region)     # "factor"
class(Species)    # "factor"
# Save to .RData file
save(Sample, Bacteria_A, Bacteria_B, Bacteria_C, Region, Species, file =
"microbiome_vectors.RData")
# Load the vectors from the .RData file
load("microbiome_vectors.RData")
# For example, print the first few elements of each vector
head(Sample)
head(Bacteria_A)
head(Region)
head(Species)
```

1.3. Use Save.Image() to Save Workspace Image




```
# Save the working environment
save.image(file = "my_workspace.RData")

# Restore the working environment
load("my_workspace.RData")
```

1.4. Use Download.File() to Download File from Website

```
> download.file("https://cran.r-project.org/src/contrib/Archive/ggplot2/ggplot2_3.3.5.tar.gz", "ggplot2_3.3.5.tar.gz")
trying URL 'https://cran.r-project.org/src/contrib/Archive/ggplot2/ggplot2_3.3.5.tar.gz'
Content type 'application/x-gzip' length 3063309 bytes (2.9 MB)
==================================================
downloaded 2.9 MB

>
> download.file("https://www.uniprot.org/uniprot/P12345.fasta", "P12345.fasta")
trying URL 'https://www.uniprot.org/uniprot/P12345.fasta'
Content type 'text/plain;format=fasta' length 553 bytes
==================================================
downloaded 553 bytes
```

2. Some Useful R Packages for Microbiome Data

2.1. Readr

```
install.packages("readr")

library(readr)

# General function of writing data from R
write_delim(df, file, delim = " ")

# Write comma (",") separated value files
write_csv(df, file)

# Write tab ("\t") separated value files
write_tsv(df, file)
```

where **df** is a data frame to be written; **file** is used to specify the file name to be saved;while **delim** is the delimiter that is used to separate values. It must be single character.

2.1. Readr

Example

BT.R* | microbiome_data | a | Untitled1* | data_BT | README.md*

Filter

	Sample	Bacteria_A	Bacteria_B	Bacteria_C	Region	Species
1	Sample1	16.14	74.12	91.01	Northern	Firmicutes
2	Sample2	99.43	14.00	66.37	Northern	Actinobacteria
3	Sample3	40.97	27.95	69.18	Northern	Bacteroides
4	Sample4	16.02	11.35	45.01	Northern	Bacteroides
5	Sample5	20.66	21.61	5.45	Northern	Bacteroides
6	Sample6	22.11	99.94	35.16	Southern	Bacteroides
7	Sample7	61.11	63.28	30.94	Southern	Actinobacteria
8	Sample8	60.13	15.23	48.72	Southern	Firmicutes
9	Sample9	46.08	86.54	54.43	Southern	Actinobacteria
10	Sample10	39.31	59.18	49.98	Southern	Firmicutes

• • •

• • •

• • •

• • •

```
# Save dataframe as CSV
write_csv(microbiome_data, file = "microbiome_data.csv")

# Save dataframe as TSV
write_tsv(microbiome_data, file = "microbiome_data.tsv")

# Save dataframe with custom delimiter (e.g., ";")
write_delim(microbiome_data, file =
"microbiome_data_custom.txt", delim = ";")
```

microbiome_data.csv

```
Sample,Bacteria_A,Bacteria_B,Bacteria_C,Region,Species
Sample1,16.14,74.12,91.01,Northern,Firmicutes
Sample2,99.43,14,66.37,Northern,Actinobacteria
Sample3,40.97,27.95,69.18,Northern,Bacteroides
Sample4,16.02,11.35,45.01,Northern,Bacteroides
Sample5,20.66,21.61,5.45,Northern,Bacteroides
Sample6,22.11,99.94,35.16,Southern,Bacteroides
Sample7,61.11,63.28,30.94,Southern,Actinobacteria
Sample8,60.13,15.23,48.72,Southern,Firmicutes
Sample9,46.08,86.54,54.43,Southern,Actinobacteria
Sample10,39.31,59.18,49.98,Southern,Firmicutes
```

microbiome_data.tsv

```
Sample   Bacteria_A   Bacteria_B   Bacteria_C   Region   Species
Sample1  16.14   74.12   91.01   Northern   Firmicutes
Sample2  99.43   14      66.37   Northern   Actinobacteria
Sample3  40.97   27.95   69.18   Northern   Bacteroides
Sample4  16.02   11.35   45.01   Northern   Bacteroides
Sample5  20.66   21.61   5.45    Northern   Bacteroides
Sample6  22.11   99.94   35.16   Southern   Bacteroides
Sample7  61.11   63.28   30.94   Southern   Actinobacteria
Sample8  60.13   15.23   48.72   Southern   Firmicutes
Sample9  46.08   86.54   54.43   Southern   Actinobacteria
Sample10         39.31   59.18   49.98   Southern   Firmicutes
```

*microbiome_data_custom.txt

```
Sample;Bacteria_A;Bacteria_B;Bacteria_C;Region;Species
Sample1;16.14;74.12;91.01;Northern;Firmicutes
Sample2;99.43;14;66.37;Northern;Actinobacteria
Sample3;40.97;27.95;69.18;Northern;Bacteroides
Sample4;16.02;11.35;45.01;Northern;Bacteroides
Sample5;20.66;21.61;5.45;Northern;Bacteroides
Sample6;22.11;99.94;35.16;Southern;Bacteroides
Sample7;61.11;63.28;30.94;Southern;Actinobacteria
Sample8;60.13;15.23;48.72;Southern;Firmicutes
Sample9;46.08;86.54;54.43;Southern;Actinobacteria
Sample10;39.31;59.18;49.98;Southern;Firmicutes
```

2.2 Ggpubr

Type: Package
Title: 'ggplot2' Based Publication Ready Plots

- Using CRAN (Comprehensive R Archive Network):

```
install.packages("ggpubr")
```

- Using GitHub:

```
install.packages("devtools")
devtools::install_github("kassambara/ggpubr")
```

2.2 Ggpubr

Download dataset:
https://datadryad.org/stash/dataset/doi:10.5061/dryad.x95x69pqf

```
install.packages("readxl")
library(readxl)
data_PT <- read_excel("C:/Users/Kim/Downloads/BT/plateassay_growthrates.xlsx")
View(data_PT)
```

Save.R × | README.md × | data_PT × | Boxplot_TH.R × | Plot.R ×

Filter

	index	replicate	species	temperature	photoperiod	bacteria	mumax
1	1	1	T_rotula	4	24	xenic	0.08829063
2	2	2	T_rotula	4	24	xenic	0.13004346
3	3	3	T_rotula	4	24	xenic	0.11478548
4	4	4	T_rotula	4	24	xenic	0.10251854
5	5	5	T_rotula	4	24	xenic	0.12222029
6	6	6	T_rotula	4	24	xenic	0.09568441
7	7	7	T_rotula	4	24	xenic	0.11578493
8	8	8	T_rotula	4	24	xenic	0.11919964
9	9	9	T_rotula	4	24	xenic	0.08955854
10	10	10	T_rotula	4	24	xenic	0.12102892
11	11	11	T_rotula	4	24	xenic	0.08787527
12	12	12	T_rotula	4	24	xenic	0.11457817
13	13	13	T_rotula	4	24	xenic	0.12561646
14	14	14	T_rotula	4	24	xenic	0.10937829
15	15	15	T_rotula	4	24	xenic	0.10288063
16	16	16	T_rotula	4	24	xenic	0.09750211
17	17	17	T_rotula	4	24	xenic	0.10131356
18	18	18	T_rotula	4	24	xenic	0.06079321
19	19	19	T_rotula	4	24	xenic	0.08617507
20	20	20	T_rotula	4	24	xenic	0.08294450

Showing 1 to 20 of 1,728 entries, 7 total columns

index: A unique identifier for each observation in the dataset, typically used to distinguish and identify individual rows of data.

replicate: The biological replicate number of the respective experimental treatment.

species: Defines the species studied in each observation. In this case, species can be "T. rotula" or "T. gravida", denoting specific marine diatom species under investigation.

temperature: The applied temperature in the experimental condition.

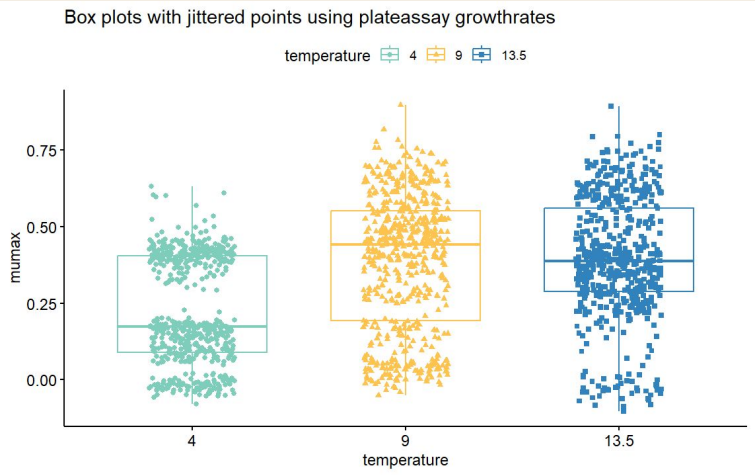
photoperiod: The duration of light applied in the experimental condition.

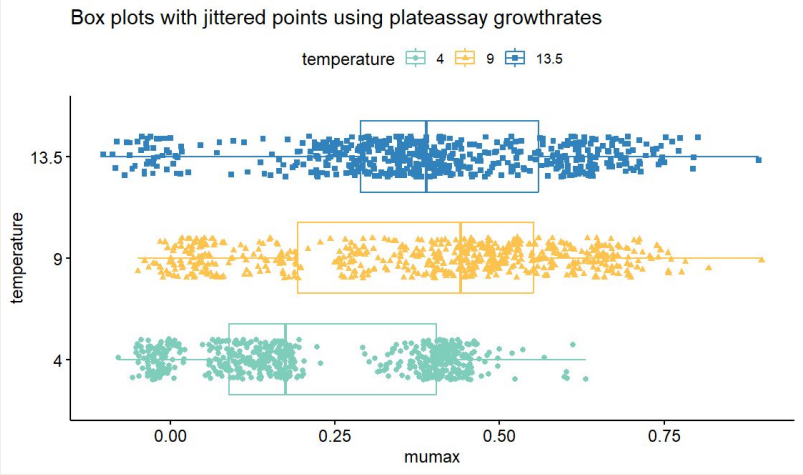
bacteria: Specifies if the native microbiome is present (xenic) or removed (axenic) for the respective observation in the experimental condition.

mumax: Maximum growth rate for each experimental unit.

2.2. Ggpubr

2.2.1. Box Plots

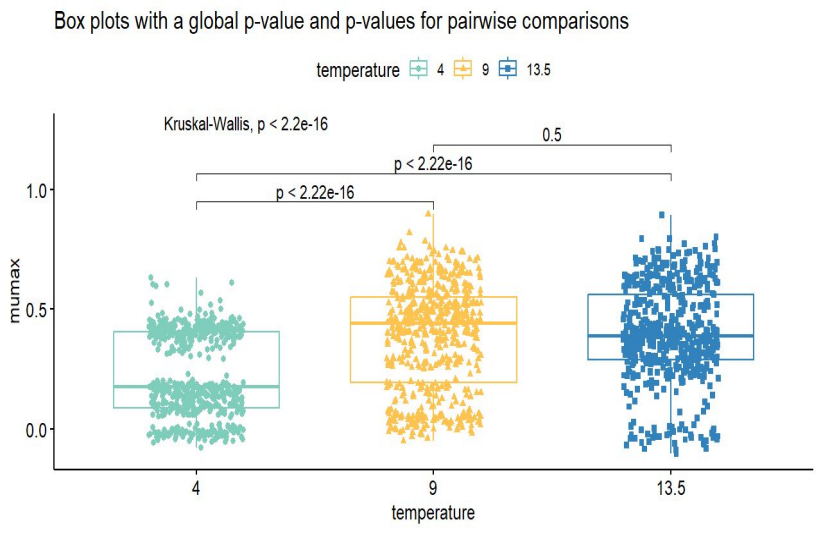


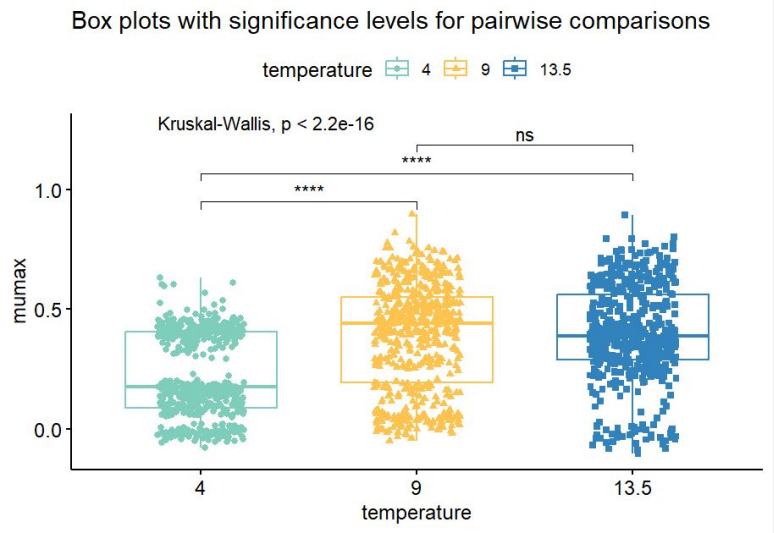



```
data_PT <- read_excel("C:/Users/Kim/Downloads/BT/plateassay_growthrates.xlsx")

p_1 <- ggboxplot(data_PT, x = "temperature", y = "mumax",
                 color = "temperature",
                 palette = c("#7fcdbb", "#fec44f", "#3182bd"),
                 add = "jitter", shape = "temperature") +
  ggtitle("Box plots with jittered points using plateassay growthrates")
```

```
p_1 <- ggboxplot(data_PT, x = "temperature", y = "mumax",
                 color = "temperature",
                 palette = c("#7fcdbb", "#fec44f", "#3182bd"),
                 add = "jitter", shape = "temperature",
                 rotate = TRUE) +
  ggtitle("Box plots with jittered points using plateassay growthrates")
```

2.2.1. Box Plots

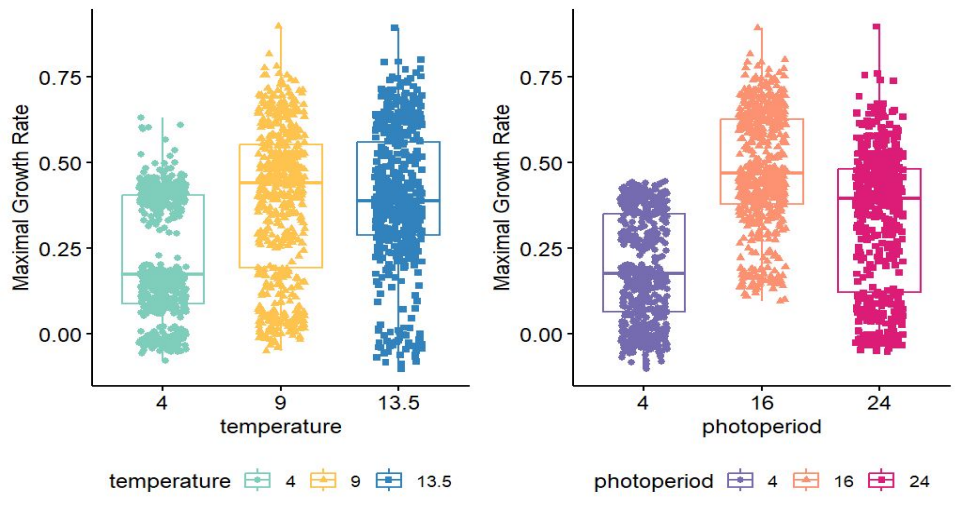





```
# Specify the pairwise group comparisons
comps <- list(c("4", "9"), c("4", "13.5"), c("9", "13.5"))
p_1 <- p_1 +
  stat_compare_means(comparisons = comps) +
  stat_compare_means(label.y = 1.25) +
  labs(title =
  "Box plots with a global p-value and p-values for pairwise comparisons")
```

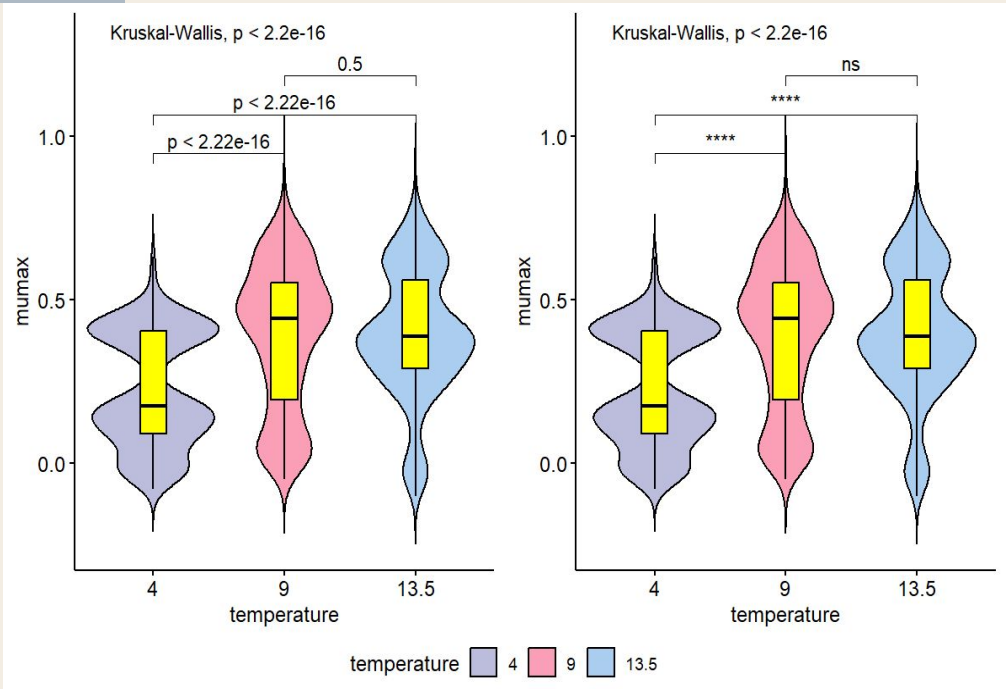
```
# Specify the comparisons of interest
comps <- list(c("4", "9"), c("4", "13.5"), c("9", "13.5"))
p_1 <- p_1 +
  stat_compare_means(comparisons = comps, label = "p.signif") +
  stat_compare_means(label.y = 1.25) +
  labs(title =
  "Box plots with significance levels for pairwise comparisons")
```

2.2.1. Box Plots



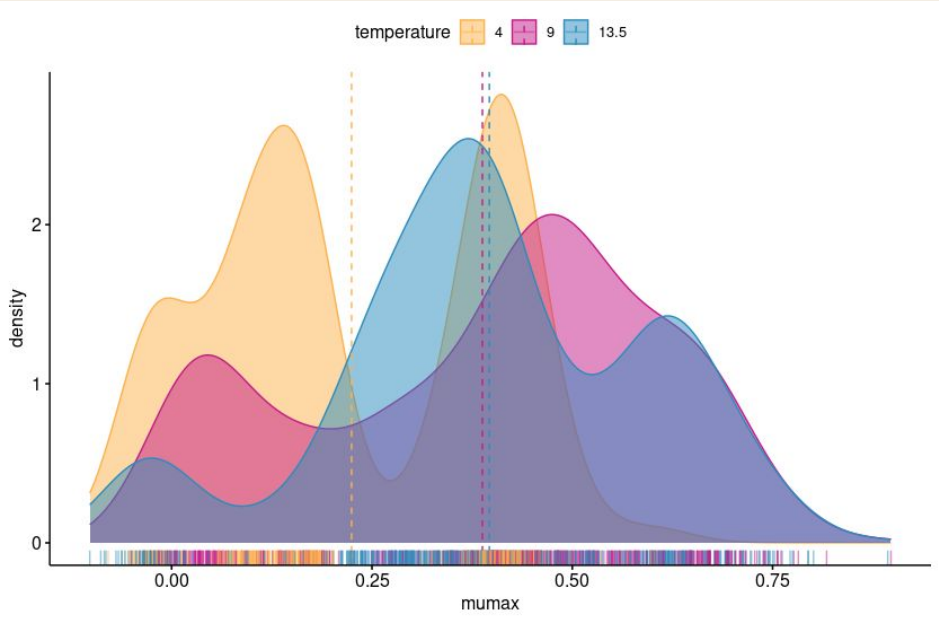

```
p_2 <- ggboxplot(data_PT, x = "temperature", y = "mumax",
                 color = "temperature",
                 palette = c("#7fcdbb", "#fec44f", "#3182bd"),
                 ylab = "Maximal Growth Rate",
                 add = "jitter", shape = "temperature")
p_3 <- ggboxplot(data_PT, x = "photoperiod", y = "mumax",
                 color = "photoperiod",
                 palette = c("#756bb1", "#fc9272", "#dd1c77"),
                 ylab = "Maximal Growth Rate",
                 add = "jitter", shape = "photoperiod")
combined_plot <- ggarrange(p_2, p_3,  ncol = 2, nrow = 1,
                           common.legend = FALSE, legend = "bottom")
```

2.2.2 Violin Plots



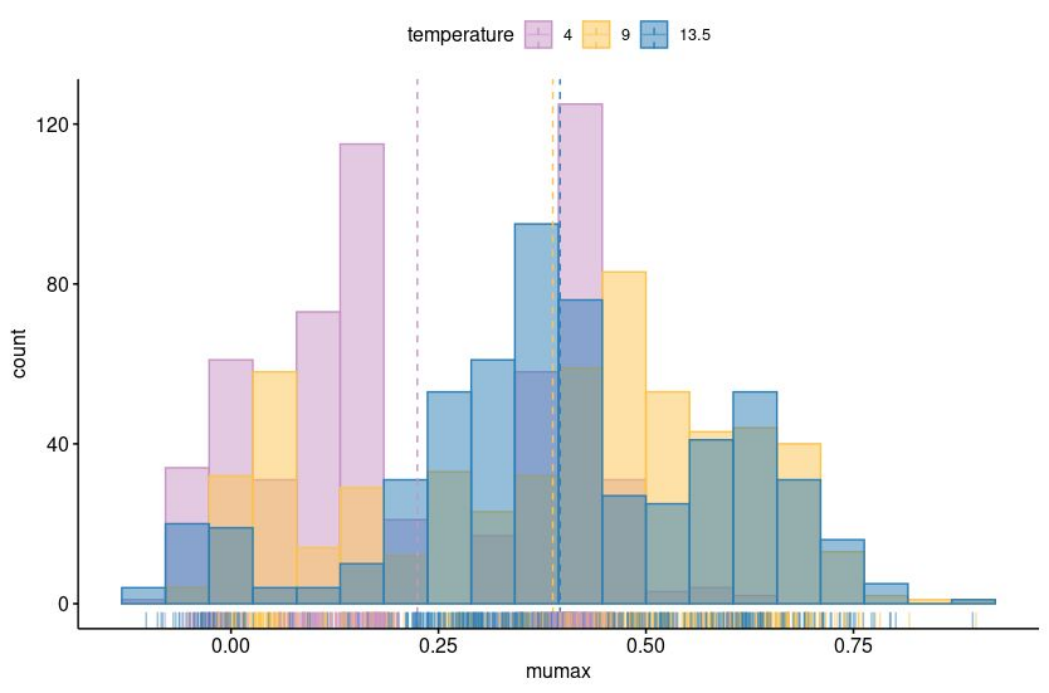

```r
library(ggpubr)

# Specify the pairwise group comparisons
comps <- list(c("4", "9"), c("4", "13.5"), c("9", "13.5"))

# Plot 1: Violin plot with boxplot inside, standard p-value labels
plot1 <- ggviolin(data_PT, x = "temperature", y = "mumax", fill = "temperature",
                  palette = c("#bcbddc", "#fa9fb5", "#abcdef"),
                  add = "boxplot", add.params = list(fill = "yellow")) +
  stat_compare_means(comparisons = comps) +
  stat_compare_means(label.y = 1.30)


# Plot 2: Violin plot with boxplot inside, customized p-value labels
plot2 <- ggviolin(data_PT, x = "temperature", y = "mumax", fill = "temperature",
                  palette = c("#bcbddc", "#fa9fb5", "#abcdef"),
                  add = "boxplot", add.params = list(fill = "yellow")) +
  stat_compare_means(comparisons = comps, label = "p.signif") +
  stat_compare_means(label.y = 1.30)

# Combine the plots into a single frame
ggarrange(plot1, plot2, ncol = 2, nrow = 1,
          common.legend = TRUE, legend = "bottom")
```

2.2.3. Density Plots




```
data_PT$temperature <- factor(data_PT$temperature)
ggdensity(data_PT, x = "mumax",
          add = "mean", rug = TRUE,
          color = "temperature", fill = "temperature",
          palette = c("#feb24c", "#c51b8a", "#2b8cbe"))
```

2.2.4. Histogram Plots




```
gghistogram(data_PT, x = "mumax",
            add = "mean", rug = TRUE,
            color = "temperature", fill = "temperature",
            bins = 20,
            palette = c("#c994c7", "#fec44f", "#2c7fb8"))
```

2.3. Tidyverse

R packages for data science
The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:
install.packages("tidyverse")

https://www.tidyverse.org/

2.3. Tidyverse

- **ggplot2**, for data visualisation.
- **dplyr** for data manipulation.
- **tidyr**, for data tidying.
- **readr**, for data import.

```
> library(tidyverse)
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✓ dplyr     1.1.4     ✓ readr     2.1.5
✓ forcats   1.0.0     ✓ stringr   1.5.1
✓ ggplot2   3.5.0     ✓ tibble    3.2.1
✓ lubridate 1.9.3     ✓ tidyr     1.3.1
✓ purrr     1.0.2
```

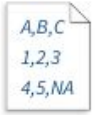
Readr

Read Tabular Data with readr

read_*(file, col_names = TRUE, col_types = NULL, col_select = NULL, id = NULL, locale, n_max = Inf, skip = 0, na = c("", "NA"), guess_max = min(1000, n_max), show_col_types = TRUE) See **?read_delim**

A	B	C
1	2	3
4	5	NA

read_delim("file.txt", delim = "|"**)** Read files with any delimiter. If no delimiter is specified, it will automatically guess.
To make file.txt, run: write_file("A|B|C\n1|2|3\n4|5|NA", file = "file.txt")



A	B	C
1	2	3
4	5	NA

read_csv("file.csv**)** Read a comma delimited file with period decimal marks.
write_file("A,B,C\n1,2,3\n4,5,NA", file = "file.csv")

A	B	C
1.5	2	3
4.5	5	NA

read_csv2("file2.csv**)** Read semicolon delimited files with comma decimal marks.
write_file("A;B;C\n1,5;2;3\n4,5;5;NA", file = "file2.csv")

A	B	C
1	2	3
4	5	NA

read_tsv("file.tsv**)** Read a tab delimited file. Also **read_table()**.
read_fwf("file.tsv", fwf_widths(c(2, 2, NA))**)** Read a fixed width file.
write_file("A\tB\tC\n1\t2\t3\n4\t5\tNA\n", file = "file.tsv")

USEFUL READ ARGUMENTS

A	B	C
1	2	3
4	5	NA

No header
read_csv("file.csv", col_names = FALSE)

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header
read_csv("file.csv",
col_names = c("x", "y", "z"))

Read multiple files into a single table
read_csv(c("f1.csv", "f2.csv", "f3.csv"),
id = "origin_file")

1	2	3
4	5	NA

Skip lines
read_csv("file.csv", skip = 1)

A	B	C
1	2	3

Read a subset of lines
read_csv("file.csv", n_max = 1)

A	B	C
NA	2	3
4	5	NA

Read values as missing
read_csv("file.csv", na = c("1"))

Specify decimal marks
read_delim("file2.csv", locale =
locale(decimal_mark = ","))

tidyr

Data tidying with tidyr : : CHEAT SHEET

Tidy data is a way to organize tabular data in a consistent data structure across packages.
A table is tidy if:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own row

Access **variables** as **vectors** — Preserve **cases** in vectorized operations

Tibbles

AN ENHANCED DATA FRAME

Tibbles are a table format provided by the **tibble** package. They inherit the data frame class, but have improved behaviors:

- **Subset** a new tibble with], a vector with [[and $.
- **No partial matching** when subsetting columns.
- **Display** concise views of the data on one screen.

options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf) Control default display settings.

View() or **glimpse()** View the entire data set.

CONSTRUCT A TIBBLE

tibble(…) Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(…) Construct by rows.
tribble(~x, ~y,
1, "a",
2, "b",
3, "c")

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

as_tibble(x, …) Convert a data frame to a tibble.

enframe(x, name = "name", value = "value") Convert a named vector to a tibble. Also **deframe()**.

is_tibble(x) Test whether x is a tibble.

Reshape Data - Pivot data to reorganize values into a new layout.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

pivot_longer(data, cols, names_to = "name", values_to = "value", values_drop_na = FALSE)

"Lengthen" data by collapsing several columns into two. Column names move to a new names_to column and values to a new values_to column.

pivot_longer(table4a, cols = 2:3, names_to ="year", values_to = "cases")

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

pivot_wider(data, names_from = "name", values_from = "value")

The inverse of pivot_longer(). "Widen" data by expanding two columns into several. One column provides the new column names, the other the values.

pivot_wider(table2, names_from = type, values_from = count)

Split Cells - Use these functions to split or combine cells into individual, isolated values.

table5

country	century	year
A	19	99
A	20	00
B	19	99
B	20	00

→

country	year
A	19[illegible]
A	20[illegible]
B	19[illegible]
B	20[illegible]

unite(data, col, …, sep = "_", remove = TRUE, na.rm = FALSE) Collapse cells across several columns into a single column.

unite(table5, century, year, col = "year", sep = "")

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", …) Separate each cell in a column into several columns. Also **extract()**.

separate(table3, rate, sep = "/", into = c("cases", "pop"))

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M

→

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M

separate_rows(data, …, sep = "[^[:alnum:].]+", convert = FALSE) Separate each cell in a column into several rows.

separate_rows(table3, rate, sep = "/")

Expand Tables

Create new combinations of variables or identify implicit missing values (combinations of variables not present in the data).

x

x1	x2	x3
A	1	3
B	1	4
B	2	3

→

x1	x2
A	1
A	2
B	1
B	2

expand(data, …) Create a new tibble with all possible combinations of the values of the variables listed in … Drop other variables.
expand(mtcars, cyl, gear, carb)

x

x1	x2	x3
A	1	3
B	1	4
B	2	3

→

x1	x2	x3
A	1	3
A	2	NA
B	1	4
B	2	3

complete(data, …, fill = list()) Add missing possible combinations of values of variables listed in … Fill remaining variables with NA.
complete(mtcars, cyl, gear, carb)

Handle Missing Values

Drop or replace explicit missing values (NA).

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
D	3

drop_na(data, …) Drop rows containing NA's in … columns.
drop_na(x, x2)

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	1
C	1
D	3
E	3

fill(data, …, .direction = "down") Fill in NA's in … columns using the next or previous value.
fill(x, x2)

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	2
C	2
D	3
E	2

replace_na(data, replace) Specify a value to replace NA in selected columns.
replace_na(x, list(x2 = 2))

RStudio

dplyr

Data transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(.data, …, wt = NULL, sort = FALSE, name = NULL) Count number of rows in each group defined by the variables in … Also **tally()**.
count(mtcars, cyl)

Group Cases

Use **group_by**(.data, …, .add = FALSE, .drop = TRUE) to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

```
mtcars %>%
  group_by(cyl) %>%
  summarise(avg = mean(mpg))
```

Use **rowwise**(.data, …) to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

```
starwars %>%
  rowwise() %>%
```

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …, .preserve = FALSE) Extract rows that meet logical criteria.
filter(mtcars, mpg > 20)

distinct(.data, …, .keep_all = FALSE) Remove rows with duplicate values.
distinct(mtcars, gear)

slice(.data, …, .preserve = FALSE) Select rows by position.
slice(mtcars, 10:15)

slice_sample(.data, …, n, prop, weight_by = NULL, replace = FALSE) Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
slice_sample(mtcars, n = 5, replace = TRUE)

slice_min(.data, order_by, …, n, prop, with_ties = TRUE) and **slice_max()** Select rows with the lowest and highest values.
slice_min(mtcars, mpg, prop = 0.25)

slice_head(.data, …, n, prop) and **slice_tail()** Select the first or last rows.
slice_head(mtcars, n = 5)

Logical and boolean operators to use with filter()

==	<	<=	is.na()	%in%	|	xor()
!=	>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …, .by_group = FALSE) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1, name = NULL, …) Extract column values as a vector, by name or index.
pull(mtcars, wt)

select(.data, …) Extract columns as a table.
select(mtcars, mpg, wt)

relocate(.data, …, .before = NULL, .after = NULL) Move columns to new position.
relocate(mtcars, mpg, cyl, .after = last_col())

Use these helpers with select() and across()
e.g. select(mtcars, mpg:cyl)

contains(match)	**num_range**(prefix, range)	:, e.g. mpg:cyl
ends_with(match)	**all_of**(x)/**any_of**(x, …, vars)	-, e.g. -gear
starts_with(match)	**matches**(match)	**everything()**

MANIPULATE MULTIPLE VARIABLES AT ONCE

across(.cols, .funs, …, .names = NULL) Summarise or mutate multiple columns in the same way.
summarise(mtcars, across(everything(), mean))

c_across(.cols) Compute across columns in row-wise data.
transmute(rowwise(UKgas), total = sum(c_across(1:2)))

MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …, .keep = "all", .before = NULL, .after = NULL) Compute new column(s). Also **add_column(), add_count(),** and **add_tally().**
mutate(mtcars, gpm = 1 / mpg)

ggplot2

Data visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

```
ggplot (data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
stat = <STAT>, position = <POSITION>) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

required

Not required, sensible defaults supplied

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

last_plot() Returns the last plot.

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Aes

Common aesthetic values.

color and **fill** - string ("red", "#RRGGBB")

linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash")

lineend - string ("round", "butt", or "square")

linejoin - string ("round", "mitre", or "bevel")

size - integer (line width in mm)

shape - integer/shape name or a single character ("a")

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank() and **a + expand_limits()**
Ensure limits include values across all plots.

b + geom_curve(aes(yend = lat + 1, xend = long + 1), curvature = 1) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend = "butt", linejoin = "round", linemitre = 1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(alpha = 50)) - x, y, alpha, color, fill, group, subgroup, linetype, size

b + geom_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept = 0, slope = 1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend = lat + 1, xend = long + 1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly()
x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5)
x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy))
x, y, alpha, color, fill, linetype, size, weight

discrete

d <- ggplot(mpg, aes(fl))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

both continuous

e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1) - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_point()
x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile()
x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl")
x, y, alpha, color, linetype, size

e + geom_smooth(method = lm)
x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1) - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

one discrete, one continuous

f <- ggplot(mpg, aes(class, hwy))

f + geom_col()
x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot()
x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill, group

f + geom_violin(scale = "area")
x, y, alpha, color, fill, group, linetype, size, weight

both discrete

g <- ggplot(diamonds, aes(cut, color))

g + geom_count()
x, y, alpha, color, fill, shape, size, stroke

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

continuous bivariate distribution

h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + geom_density_2d()
x, y, alpha, color, group, linetype, size

h + geom_hex()
x, y, alpha, color, fill, size

continuous function

i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))

j + geom_crossbar(fatten = 2) - x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar() - x, ymax, ymin, alpha, color, group, linetype, size, width
Also **geom_errorbarh()**.

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange() - x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map) **+ expand_limits**(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size

THREE VARIABLES

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z))
x, y, z, alpha, color, group, linetype, size, weight

l + geom_contour_filled(aes(fill = z))
x, y, alpha, color, fill, group, linetype, size, subgroup

l + geom_raster(aes(fill = z), hjust = 0.5, vjust = 0.5, interpolate = FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z))
x, y, alpha, color, fill, linetype, size, width


thank you
ANNALEIGH + BENJAMIN | 06.10.20XX