Introduction to Data Science (11372 & G 11516)

Semester 1 2021

UNIVERSITY OF CANBERRA

INTRODUCTION TO DATA SCIENCE

Lecture 6

Dr. Ibrahim Radwan

DISTINCTIVE BY DESIGN

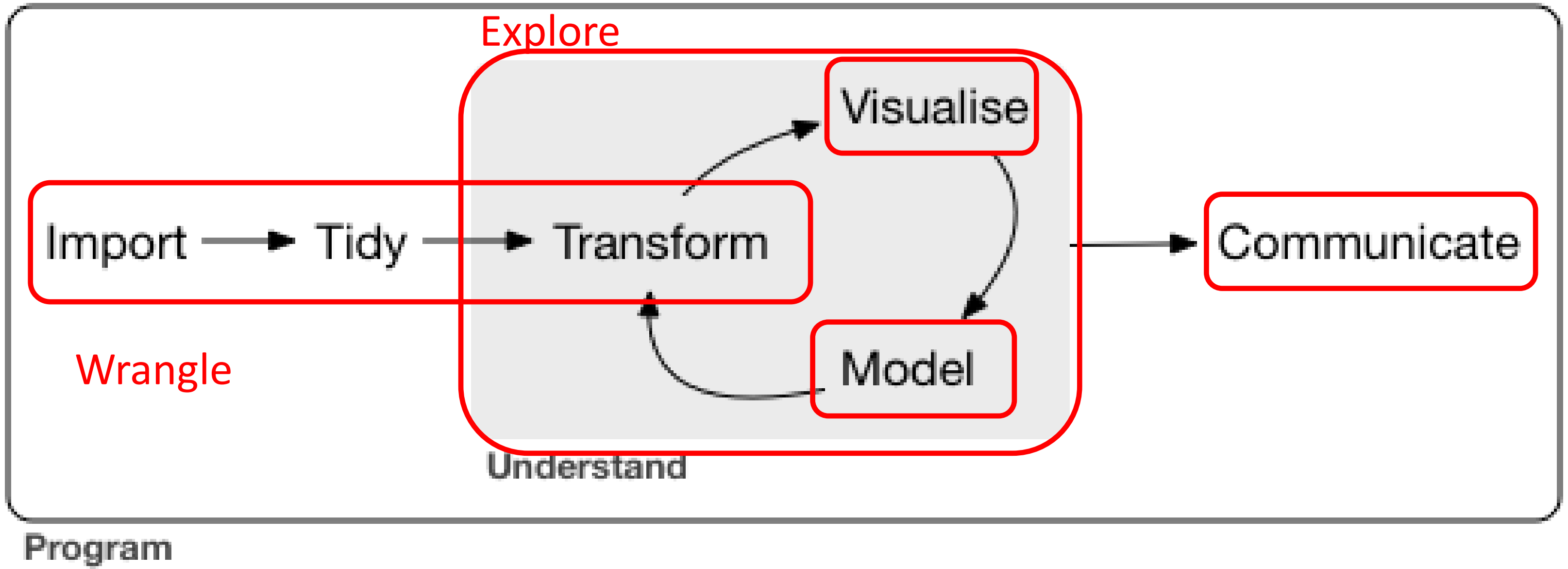
OUTLINE

UNIVERSITY OF CANBERRA

- Data Science, a Practical View
- Data Wrangling
- Data Manipulation

DATA SCIENCE; A PRACTICAL VIEW

UNIVERSITY OF CANBERRA



R for Data Science, by *Garrett Grolemund and Hadley Wickham*

Program Steps

1- Reading Data

2- Data Wrangling

3- Data Exploratory

4- Modelling

5- Result Communication

DATA WRANGLING

UNIVERSITY OF CANBERRA

- Data wrangling is the process of transforming the data into a suitable format to conduct the modelling or analysis processes.
- Data wrangling is important: without it, the data may not be ready to be analysed.
- Data wrangling involves the following five steps:

import

1. Gather your data from different sources

Transform

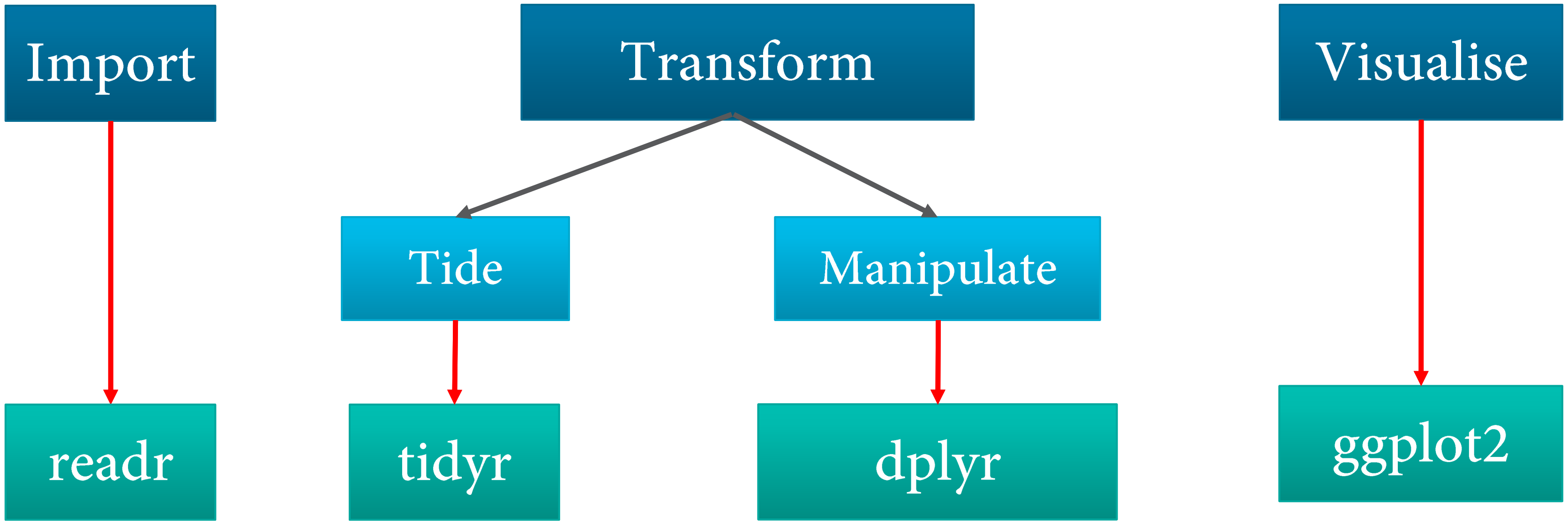
2. Clean up the duplicates, blanks, and other logical errors
3. Join all your data into a single table (e.g. data frame)
4. Adding new variables/records by calculating new fields and re-ordering

Visualise

5. View or visualize the data to remove outliers and illogical results

DATA WRANGLING (2)

- Practically, we have three main processes to wrangle the data



- These four packages provide the *grammar* of the data importing, tidying, manipulation and data visualisation.

DATA IMPORT

UNIVERSITY OF CANBERRA

- Reading flat files such as CSV or text files
- Suppose we have the following CSV file:
  - We will use the `***readr***` library to read the csv files
  - Then, we can use the read_csv(“filename.csv”) function to read the contents.
  - The returned object is a tibble.
- It reads the data with 10x faster than the base R functions.

```
ID, Name, Age
23424, Ana, 45
11234, Charles, 23
77654, Susanne, 76
```

```
> read_csv("data_sample.csv")
Parsed with column specification:
cols(
  ID = col_double(),
  Name = col_character(),
  Age = col_double()
)
# A tibble: 3 x 3
     ID Name       Age
  <dbl> <chr>    <dbl>
1 23424 Ana         45
2 11234 Charles     23
3 77654 Susanne     76
>
```

DATA IMPORT (2)

UNIVERSITY OF CANBERRA

```
read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"),
  quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,
  n_max), progress = interactive())
```

```
a,b,c
1,2,3
4,5,NA
```

A	B	C
1	2	3
4	5	NA

Comma Delimited Files

read_csv("file.csv"**)**

```
a b c
1 2 3
4 5 NA
```

A	B	C
1	2	3
4	5	NA

Tab Delimited Files

read_tsv("file.tsv"**)**

Also **read_table()**

```
a;b;c
1;2;3
4;5;NA
```

A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files

read_csv2("file2.csv"**)**

```
a|b|c
1|2|3
4|5|NA
```

A	B	C
1	2	3
4	5	NA

Files with Any Delimiter

read_delim("file.txt", delim = "|"**)**

To save data into csv or txt file

Comma delimited file

write_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**

File with arbitrary delimiter

write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append**)**

DATA IMPORT (3)

UNIVERSITY OF CANBERRA

```r
# Import data into R

# It is part of the `tidyverse` package
if (!("tidyverse" %in% rownames(installed.packages()))){
  install.packages("tidyverse")
}

library("tidyverse")

heights <- read_csv("data/heights.csv")

# supply inline csv file
read_csv("a,b,c
1,2,3
4,5,6")

# skip first two lines, i.e. meta data the beginning of a file
read_csv("The first line of metadata
  The second line of metadata
  x,y,z
  1,2,3", skip = 2)
# or skip when comment
read_csv("# A comment I want to skip
  x,y,z
  1,2,3", comment = "#")

# by default the first row is for column, we can skip this rule
read_csv("1,2,3\n4,5,6", col_names = FALSE) # \n is convenient shortcut for new line
# pass column names
read_csv("1,2,3\n4,5,6", col_names = c("x", "y", "z"))

# deal with na values
read_csv("a,b,c\n1,2,.", na = ".")
```

DATA IMPORT – COL SPECIFICATIONS

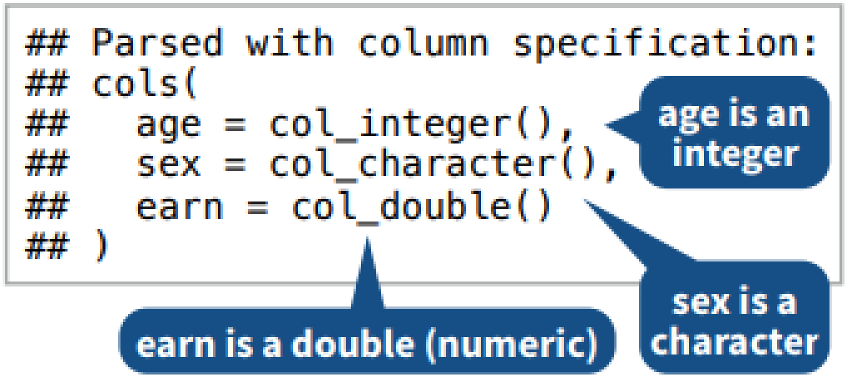
- The functions of `readr` package <u>**guess**</u> the types of each column and convert them as appropriate (but will NOT convert strings to factors automatically). A message shows the type of each column will appear in the result.

For each col_* function, there is a parse_* function

col_guess function uses the first 1000 rows from each column to guess the column type.

What about if:

- *The first 1000 rows are just special case and the rest are different?*
- *The first 1000 rows are `na` ?*



The solution is to use problems() function

DATA IMPORT – COL SPECIFICATIONS (2)

```r
# col specifications
# logicals
lgl <- parse_logical(c("TRUE", "FALSE", "NA"))
str(lgl)
# integers
intgr <- parse_integer(c("1", "2", "3"))
str(intgr)
# date
dt <- parse_date(c("2010-01-01", "1979-10-14"))
str(dt)
# which strings should be specified as missing
intgr_missing <- parse_integer(c("1", "231", ".", "456"), na = ".")
str(intgr_missing)
# sometimes the parsing fails, NAs will be used when failure is occuring
x <- parse_integer(c("123", "345", "abc", "123.45", "221"))
x
# here you can use problems
problems(x)

# parse numbers with . or , or $ or %
dbl <- parse_double("1.24")
str(dbl)
num <- parse_number("$123,456,789")
num
num <- parse_number("$123,456,789")
num
num <- parse_number("$123.456.789", locale=locale(grouping_mark = '.'))
num
num <- parse_number("123'456'789", locale = locale(grouping_mark = "'"))
num
```

DATA IMPORT – COL SPECIFICATIONS (3)

UNIVERSITY OF CANBERRA

- Using the *problems()* function with the read_* functions from the `readr` package to check the problems in parsing and guessing the column types.

```
problems(tibble_object)
```

DATA IMPORT – COL SPECIFICATIONS (4)

UNIVERSITY OF CANBERRA

```r
# base function to read data.frame
df <- read.csv(readr_example("challenge.csv"))
df
view(df)
df[1001,]
str(df)
problems(df) # no problems as everything are chars
##################################
# readr functions to read csv
tbl <- read_csv(readr_example("challenge.csv"))
tbl
view(tbl)
tbl[1001,]
str(tbl)
##################################
# use problems to check the problems in the tbl parsing
problems(tbl)

# how to fix the problems
# first let us fix the problem by hard-specifying the column types
tbl2 <- read_csv(
  readr_example("challenge.csv"),
  col_types = cols(
    x = col_double(),
    y = col_character()
  )
)
tbl2
tail(tbl2) # there are dates stored with this column
# change the character to date
tbl3 <- read_csv(
  readr_example("challenge.csv"),
  col_types = cols(
    x = col_double(),
    y = col_date()
  )
)
```

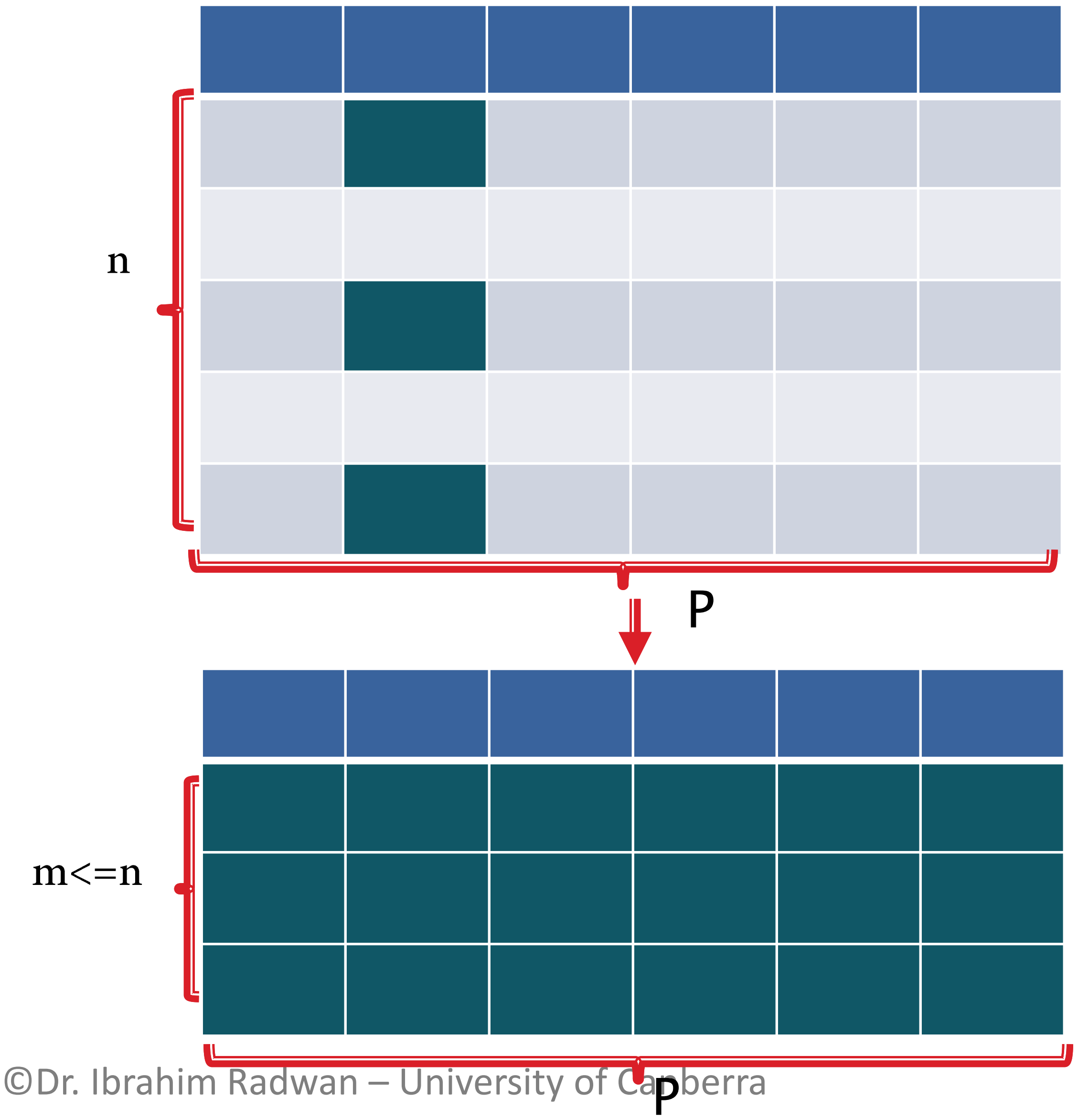
DATA MANIPULATION

- The `*dplyr*` package in `*tidyverse*` library presents five *verbs* for manipulating the data in data frames:

    1. filter() extracts a subset of the rows (i.e., observations) based on some criteria
    2. select() extracts a subset of the columns (i.e., features, variables) based on some criteria
    3. mutate() adds or modifies existing columns
    4. arrange() sorts the rows
    5. summarise() aggregates the data across rows (e.g., group them according to some criteria)

- Each of these functions takes a data frame as its first argument and returns a data frame.

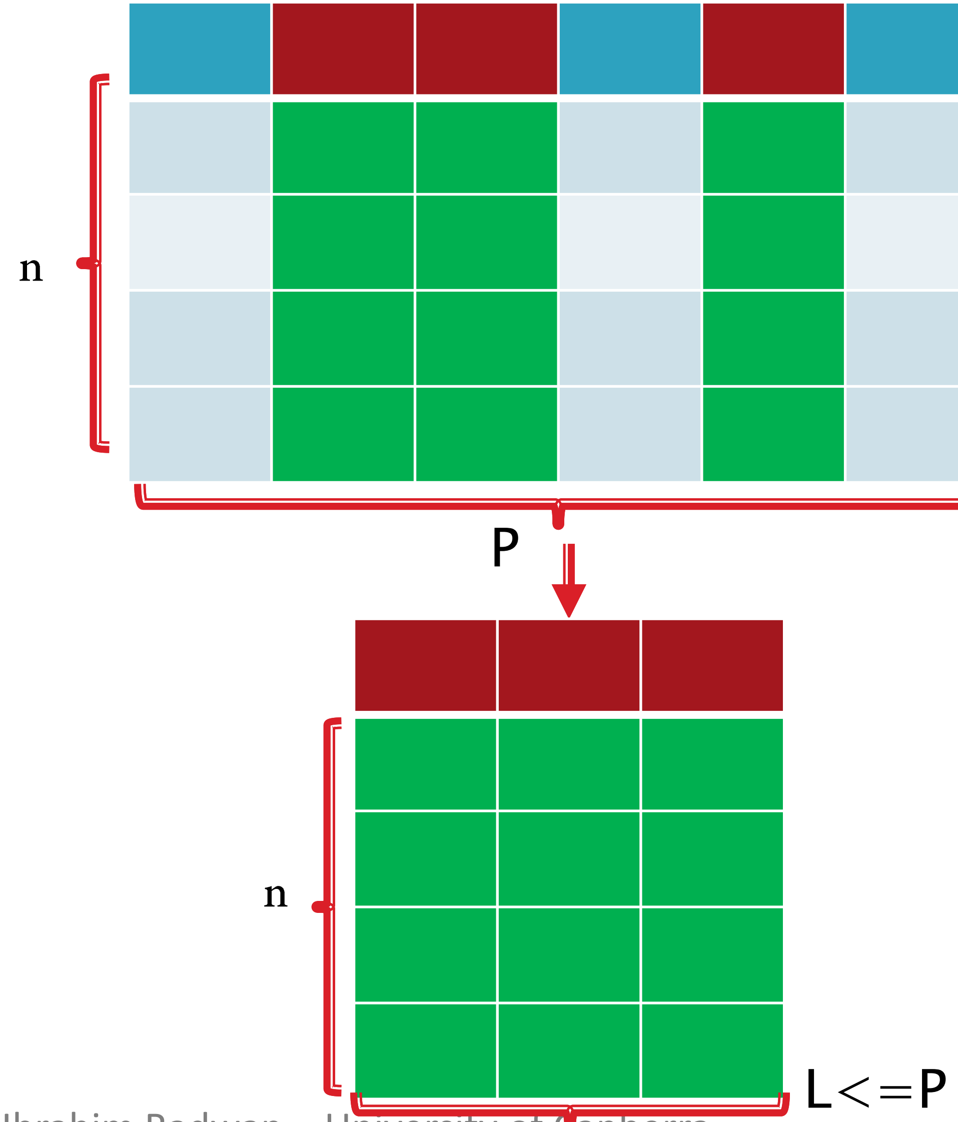
SUBSET BY ROW (FILTER)




```
filter(dataframe, criteria)
```

```
# call the required libraries
library(tidyverse)
library(nycflights13)
df <- flights
# filter based on conditions
filter(df, month == 1, day == 1)
filter(df, month == 12, day == 25)
# you may combine conditions using logical
operators
filter(df, month == 1 | month == 12)
# or by combining variables in vector
filter(df, month %in% c(11, 12))
# comma means and (&)
# for example, extract all records for flights,
that were not delayed (arr and dep) more than 2 hrs
filter(flights, arr_delay <= 120, dep_delay <= 120)
```

SUBSET BY COLUMN (SELECT)

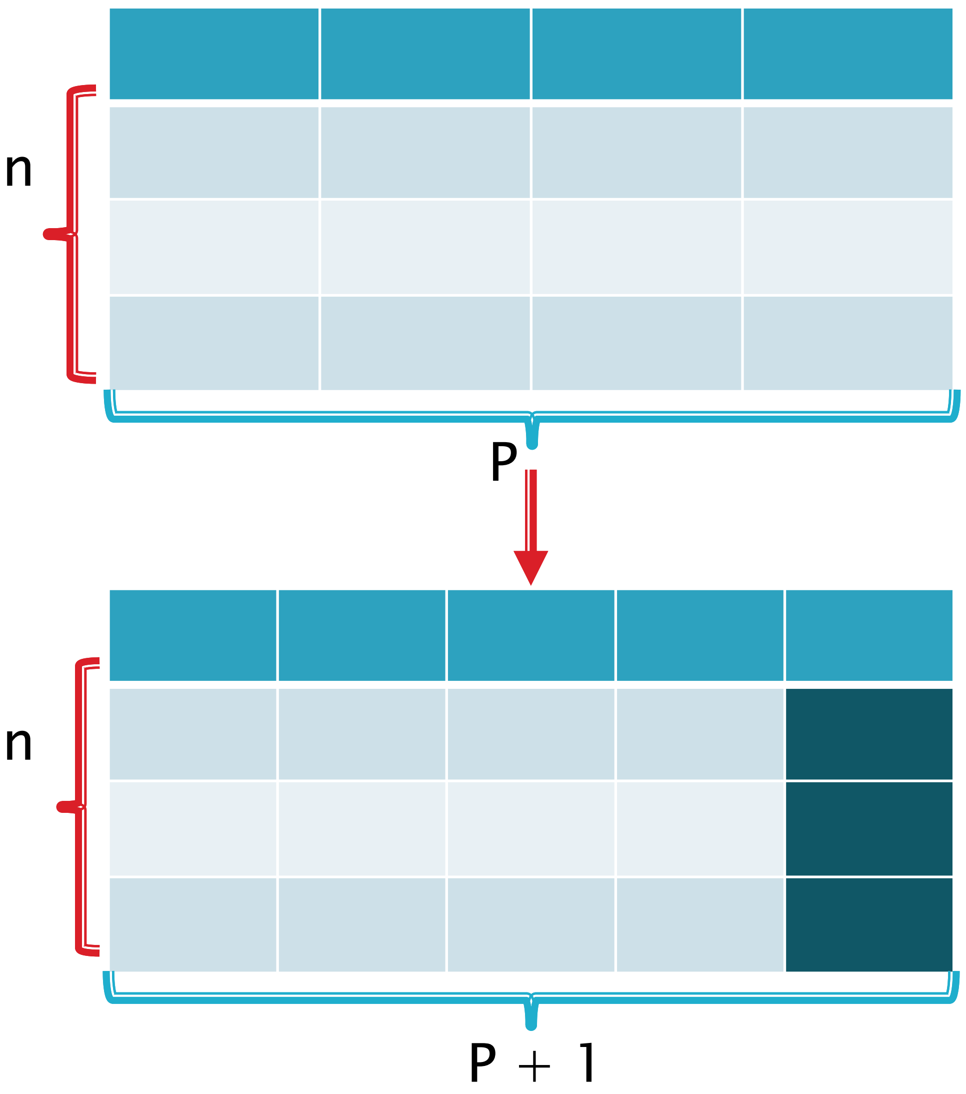



```
select(dataframe, col_names or criteria)
```

```
# call the required libraries
library(tidyverse)
library(nycflights13)
df <- flights
# Select columns by name
select(flights, year, month, day)
# Select all columns between year and day
(inclusive)
select(flights, year:day)
# Select all columns except those from year to day
(inclusive)
select(flights, -(year:day))
# rename() is a variant of select() that keeps all
the variables that aren't explicitly mentioned:
rename(flights, tail_num = tailnum)
# Move a variable to the start of the data frame.
select(flights, time_hour, air_time, everything())
```

ADD - MODIFY COLUMN (MUTATE)

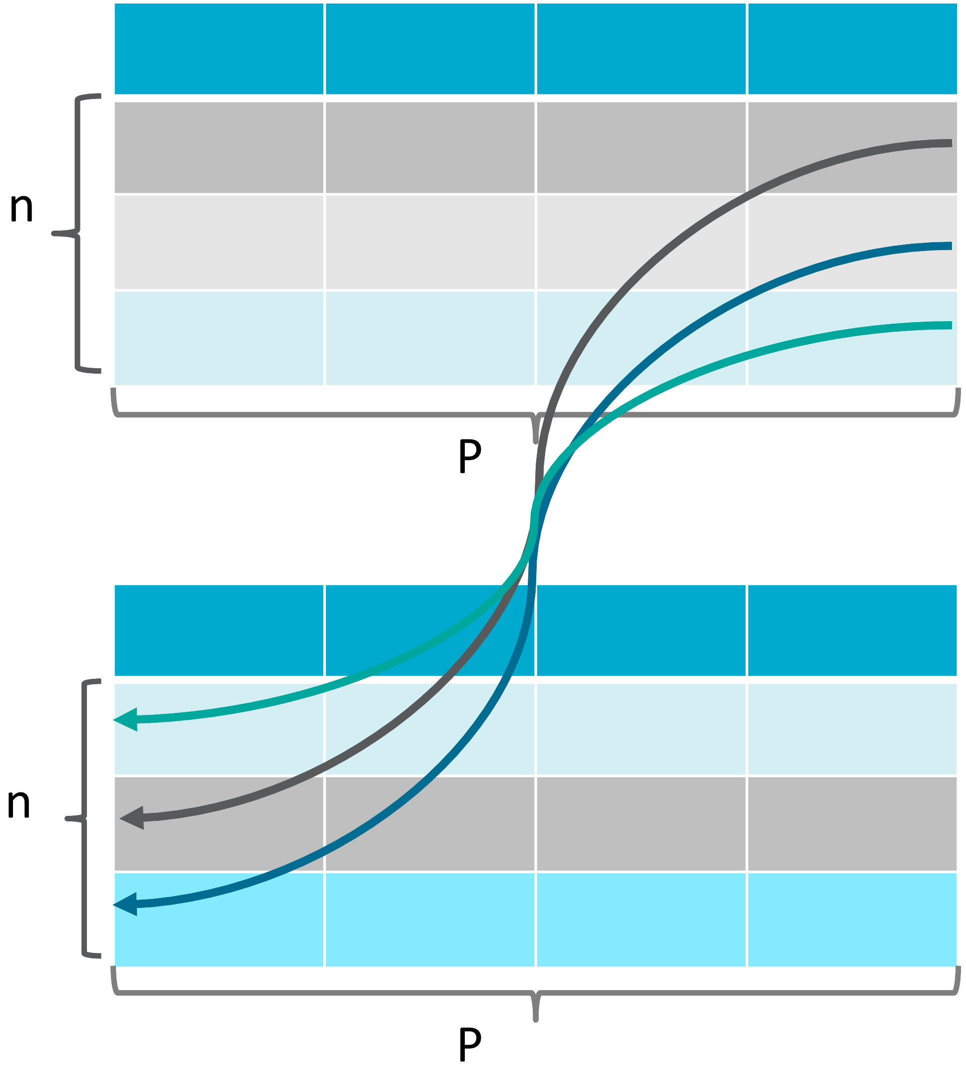
UNIVERSITY OF CANBERRA




```
mutate(dataframe, new_cols or modified_cols)
```

```
# call the required libraries
library(tidyverse)
library(nycflights13)
df <- flights
# Create a new dataset
flights_sml <- select(df, year:day, ends_with("delay"),
distance, air_time )
# add new columns to the data frame
mutate(flights_sml, gain = dep_delay - arr_delay, speed =
distance / air_time * 60 )
# Note that you can refer to columns that you've just
created:
mutate(flights_sml, gain = dep_delay - arr_delay, hours =
air_time / 60, gain_per_hour = gain / hours )
# If you only want to keep the new variables, use
transmute():
transmute(flights, gain = dep_delay - arr_delay, hours =
air_time / 60, gain_per_hour = gain / hours )
```

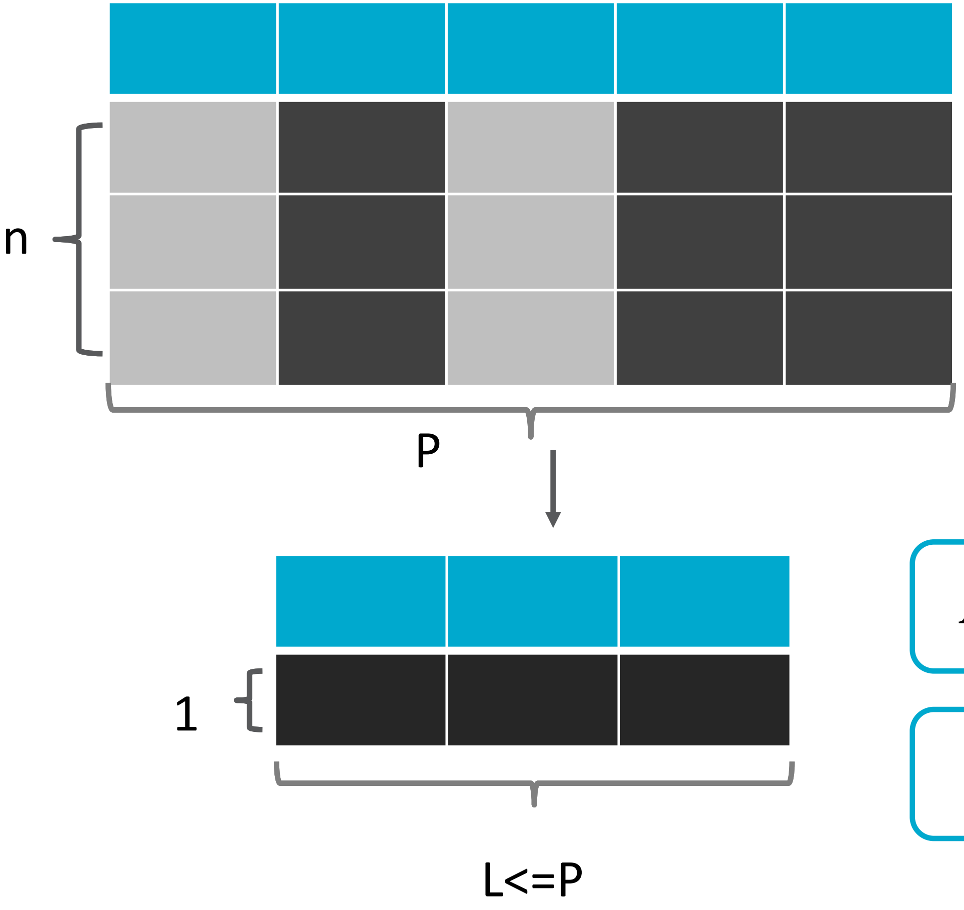
RE-ORDER ROWS (ARRANGE)




```
arrange(dataframe, col_name)
```

```
# call the required libraries
library(tidyverse)
library(nycflights13)
df <- flights
# sort data by distance
arrange(df, distance)
# sort data by distance descendingly
arrange(df, desc(distance))
# Sort Data by Multiple Variables
arrange(df, dep_time, arr_time)
```

AGGREGATE (SUMMARISE)




```
summarise(dataframe, agg_func(col_name))
```

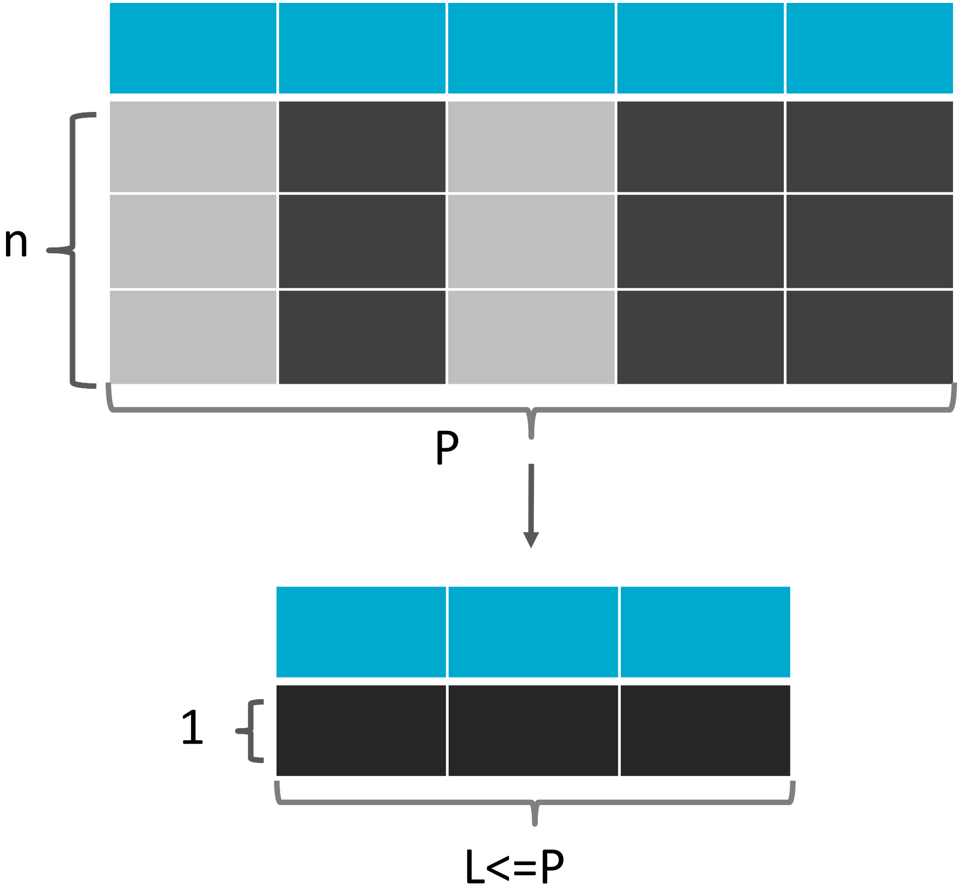
```
# call the required libraries
library(tidyverse)
library(nycflights13)
df <- flights
# extract a statistical metric from variable /
variables of the data
summarise(df, delay = mean(dep_delay, na.rm = TRUE))
```

Aggregation functions such as mean, sd, var, median, min and max

There is also summarise_each(dataframe, funs(aggregation_func))

summarise() is not terribly useful unless we pair it with group_by()

AGGREGATE WITH GROUPING




```
group_by(dataframe, col_name)
```

```
# group the data of the flights by the date
by_day <- group_by(flights, year, month, day)
# get the average delay per date/day
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
# Imagine that we want to explore the relationship
between the distance and average delay for each
location.
by_dest <- group_by(flights, dest)

# extract the number of flights, average distance and
average delay for each destination
delay <- summarise(by_dest, count= n(), dist=
mean(distance, na.rm = TRUE), delay= mean(arr_delay,
na.rm = TRUE) )

# visualise to understand the relationship
ggplot(data= delay, mapping=aes(x= dist, y= delay)) +
geom_point(aes(size= count), alpha= 1/ 3) +
geom_smooth()
```

THE PIPE OPERATOR %>%

- In data wrangling, most likely, you need to perform series of operations (i.e. *verbs*) on the same data.
- This will need you to create intermediate tables temporarily to save the results to be processed with the next operations.
- R provides an elegant way to perform series of operations on the same data in one go via using the *pipe operator* %>%

original data ⟶ select ⟶ filter

```
F(x) is the same as
x %>% F
```

```
16 %>% sqrt() %>% log2()
[1] 2
```

THE PIPE OPERATOR %>% (2)

- In the previous `*nycflights*` example, there were three steps to extract the relationship between the distance and the delay of the flights per destination.
    1. Group flights by destination.
    2. Summarise to compute distance, average delay, and number of flights.
    3. Filter to remove noisy points

This can be achieved by using the pipe operator:

```
delay <- df %>%
group_by(dest) %>%
summarise(count= n(), dist= mean(distance, na.rm = TRUE),
delay= mean(arr_delay, na.rm = TRUE)) %>%
filter(count > 20, dest != 'HNL')
```

RECOMMENDED READING

UNIVERSITY OF CANBERRA

- You are recommended to read chapters 5 & 12 from the *"R for Data Science"* book:
  - https://r4ds.had.co.nz/transform.html