Mastering R by Examples

Avan Suinesiaputra

Table of contents

About **3**

1 The pipe operator (%>%) **4**

1.1 What is a pipe or chain operator? 4
1.2 Example 1: The Empire Strikes Back 5
1.2.1 Without piping 6
1.2.2 With piping 6
1.3 Example 2: Piping with `gtsummary` 8
1.4 Example 3: `ggplot2` 8
1.5 Differences between `%>%` and `|>` operators 10

2 Reading CSV file **12**

2.1 Without correction 12
2.2 With corrections 13
2.3 Table summary 15

3 Different ways to summarise **17**

3.1 `desc_table` 17
3.2 `skim` 18
3.3 `tableone` 20
3.4 `tbl_summary` 20

4 Bland-Altman analysis **23**

4.1 Loading the data 23
4.1.1 Automated segmentation 23
4.1.2 Manual segmentation 24
4.1.3 Join the tables 25
4.2 Calculate differences and averages 25
4.3 LVEDV 26
4.4 RVEDV 27

References **30**

About

This is a cookbook recipes for R using data directly. There is no structure in this book. All chapters are independent and can be executed directly. Use these recipe to quickly learn to solving data analytics with R.

1 The pipe operator (%>%)

The cryptic operator %>% in this book is called *pipe* or *chain* operator. It's a simple way to reformat your R scripts to be human readable scripts. I'll show you how here.

> Note
>
> The pipe operator %>% is available from the magrittr package, which can be activated by using the tidyverse package. Hence, you need to call this statement
>
> ```
> library(tidyverse)
> ```
>
> before you can use the %>% operator.
> Since ver 4.1.0, R provides the default |> pipe operator. It's basically the same function between the two operators with some advantages of using %>%, which I prefer. I write these differences at the end of this chapter.

1.1 What is a pipe or chain operator?

The idea is to simply perform a set of operations where the output of one operation becomes an input of the next operation.

Let's say you have the following operations:

```
# --- this code is not running
data <- get_input()
data_1 <- shift_data(data, h=10)
data_2 <- separate_data(data_1, sep="_", sort=TRUE)
print(data_2)
# ---
```

You can write the whole operations to become

```
# --- this code is not running
print(separate_data(shift_data(get_input(), h=10), sep="_", sort=TRUE))
# ---
```

but this becomes cumbersome and unreadable.

If you examine the operations further, you notice that the first argument of each operation, except the first one, is the output of the previous operation. With the pipe operator `%>%`, you can rewrite it into

```
# --- this code is not running
get_input() %>%
  shift_data(h=10) %>%
  separate_date(sep="_", sort=TRUE) %>%
  print()
# ---
```

And the block of operation becomes readable: *get the input data*, then *shift the data by 10*, then *separate the data by using the character '_' as a separator and sort them*, and finally *print the result*. You don't even define a variable to capture the results !!.

> Note
>
> By the default, the `%>%` operator will give the input of the left hand side operation into **the first argument** of the right hand side operation. There is a trick to put which argument you want on the right hand side by using the dot operator. See the later examples.

1.2 Example 1: The Empire Strikes Back

There is a built-in data called `starwars` from `tidyr` package, which is part of `tidyverse`. You are interested to see names of characters that appeared in the famous "The Empire Strikes Back" episode of Star Wars.

```
glimpse(starwars)
```

```
Rows: 87
Columns: 14
$ name       <chr> "Luke Skywalker", "C-3PO", "R2-D2", "Darth Vader", "Leia Or~
$ height     <int> 172, 167, 96, 202, 150, 178, 165, 97, 183, 182, 188, 180, 2~
$ mass       <dbl> 77.0, 75.0, 32.0, 136.0, 49.0, 120.0, 75.0, 32.0, 84.0, 77.~
$ hair_color <chr> "blond", NA, NA, "none", "brown", "brown, grey", "brown", N~
$ skin_color <chr> "fair", "gold", "white, blue", "white", "light", "light", "~
$ eye_color  <chr> "blue", "yellow", "red", "yellow", "brown", "blue", "blue",~
$ birth_year <dbl> 19.0, 112.0, 33.0, 41.9, 19.0, 52.0, 47.0, NA, 24.0, 57.0, ~
$ sex        <chr> "male", "none", "none", "male", "female", "male", "female",~
$ gender     <chr> "masculine", "masculine", "masculine", "masculine", "femini~
$ homeworld  <chr> "Tatooine", "Tatooine", "Naboo", "Tatooine", "Alderaan", "T~
$ species    <chr> "Human", "Droid", "Droid", "Human", "Human", "Human", "Huma~
$ films      <list> <"A New Hope", "The Empire Strikes Back", "Return of the J~
$ vehicles   <list> <"Snowspeeder", "Imperial Speeder Bike">, <>, <>, <>, "Imp~
$ starships  <list> <"X-wing", "Imperial shuttle">, <>, <>, "TIE Advanced x1",~
```

1.2.1 Without piping

Let's work out traditionally without using the pipe operators. To simplify the name of variables, I'll use the same name of variable for the input/output processes.

```r
# read the data
dt <- starwars
# filter only characters that appeared in "The Empire Strikes Back" movie
dt <- rowwise(dt)
dt <- filter(dt, "The Empire Strikes Back" %in% films)
dt <- ungroup(dt)
# show the name, sex, gender, homeworld and their species only
dt <- select(dt, c(name, sex, gender, homeworld, species))
# print
dt
```

1.2.2 With piping

```r
starwars %>%
  rowwise() %>%
  filter("The Empire Strikes Back" %in% films) %>%
  ungroup() %>%
  select(c(name, sex, gender, homeworld, species)) %>%
  knitr::kable()
```

name	sex	gender	homeworld	species
Luke Skywalker	male	masculine	Tatooine	Human
C-3PO	none	masculine	Tatooine	Droid
R2-D2	none	masculine	Naboo	Droid
Darth Vader	male	masculine	Tatooine	Human

name	sex	gender	homeworld	species
Leia Organa	female	feminine	Alderaan	Human
Obi-Wan Kenobi	male	masculine	Stewjon	Human
Chewbacca	male	masculine	Kashyyyk	Wookiee
Han Solo	male	masculine	Corellia	Human
Wedge Antilles	male	masculine	Corellia	Human
Yoda	male	masculine	NA	Yoda's species
Palpatine	male	masculine	Naboo	Human
Boba Fett	male	masculine	Kamino	Human
IG-88	none	masculine	NA	Droid
Bossk	male	masculine	Trandosha	Trandoshan
Lando Calrissian	male	masculine	Socorro	Human
Lobot	male	masculine	Bespin	Human

Note

In the example above, there is statement `rowwise()` before filtering the rows. This is needed because by default statistical operations are performed column-wise. For example, `max(A,B,C)` will perform the maximum of columns A, B, and C together.

```
dt <- data.frame(id=paste("id", c(1:5), sep="_"), A=runif(5), B=runif(5), C=runif(5))

mutate(dt, max=max(A,B,C))
```

```
    id         A         B         C       max
1 id_1 0.9335346 0.5877446 0.8786894 0.9797678
2 id_2 0.6730267 0.1993654 0.9601472 0.9797678
3 id_3 0.9797678 0.5094320 0.1706950 0.9797678
4 id_4 0.3117778 0.3979198 0.2999308 0.9797678
5 id_5 0.9211886 0.2073221 0.4544533 0.9797678
```

Compare to this:

```
dt %>% rowwise() %>% mutate(max=max(A,B,C)) %>% ungroup()
```

```
# A tibble: 5 x 5
  id        A     B     C   max
  <chr> <dbl> <dbl> <dbl> <dbl>
1 id_1  0.934 0.588 0.879 0.934
2 id_2  0.673 0.199 0.960 0.960
3 id_3  0.980 0.509 0.171 0.980
```

```
4 id_4  0.312 0.398 0.300 0.398
5 id_5  0.921 0.207 0.454 0.921
```

The `rowwise()` function groups data by rows, and the last `ungroup()` function removes the grouping.
See more about row-wise and column-wise operation in R.

1.3 Example 2: Piping with `gtsummary`

There are some packages that are fully compatible with `%>%` operator. One of them is `gtsummary`, which provides lots of useful summarisation functions for different tables.

Let's try to make a summary of the `iris` table:

```
library(gtsummary)
```

```
iris %>%
  tbl_summary(by=Species) %>%
  add_p() %>%
  modify_header(label = "*Morphology*") %>%
  modify_spanning_header(all_stat_cols() ~ "**Iris Species N = {N}**")
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

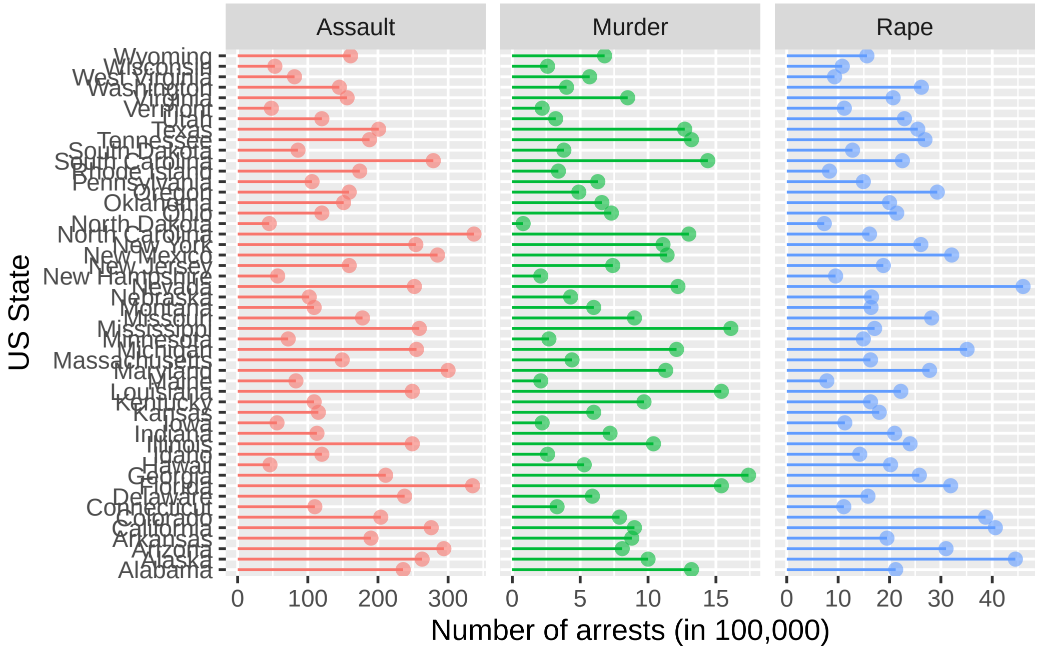
Morphology	**setosa**, N = 50	**versicolor**, N = 50	**virginica**, N = 50	**p-value**
Sepal.Length	5.00 (4.80, 5.20)	5.90 (5.60, 6.30)	6.50 (6.23, 6.90)	<0.001
Sepal.Width	3.40 (3.20, 3.68)	2.80 (2.53, 3.00)	3.00 (2.80, 3.18)	<0.001
Petal.Length	1.50 (1.40, 1.58)	4.35 (4.00, 4.60)	5.55 (5.10, 5.88)	<0.001
Petal.Width	0.20 (0.20, 0.30)	1.30 (1.20, 1.50)	2.00 (1.80, 2.30)	<0.001

1.4 Example 3: `ggplot2`

The package `ggplot2` also provide *chaining* operations by using a different symbol, i.e. `+` the plus character. Usually it's intended to add new objects to the canvas. You can combine both pipe operations to create a nice plot For example,

```r
library(ggplot2)
```

```r
USArrests %>%
  # make an explicit column for the row names
  rownames_to_column("State") %>%
  # ignore the UrbanPop column
  select(-c(UrbanPop)) %>%
  # make a long table
  pivot_longer(-c(State), names_to = "Crime", values_to = "Arrests") %>%
  # sort by US state
  arrange(State) %>%
  # give it to ggplot
  ggplot(aes(x=State, y=Arrests, color=Crime)) +
  # hence the chaining operations in ggplot
  facet_wrap(vars(Crime), scales="free_x") +
  geom_segment(aes(x=State, xend=State, y=0, yend=Arrests)) +
  geom_point(size=2, alpha=0.6) +
  coord_flip() +
  theme(legend.position="none") +
  xlab("US State") +
  ylab("Number of arrests (in 100,000)")
```



1.5 Differences between %>% and |> operators

The statement:

```
mtcars %>% filter(gear==5)
```

is equivalent with

```
mtcars |> filter(gear==5)
```

which is the piping operation for

```
filter(mtcars, gear==5)
```

There are some advantages of using %>% operator:

1. It allows you to change the argument placement of the next operation by using the dot operator:

```r
# default is the first argument
x %>% f(1,2,3)   # equals to f(x,1,2,3)
x %>% f(1,.,2,3) # equals to f(1,x,2,3)
```

For the base operator, you must give a named attribute to change the placement:

```r
# default is the first argument
x |> f(1,2,3)   # equals to f(x,1,2,3)
x |> f(1,y=_)   # equals to f(1,x)
```

2. More flexibility with dot operator:

   ```r
   x %>% f(.$var)      # equals to f(x, x$var)
   x %>% {f(.$a, .$b)} # equals to f(x$a, x$b)
   ```

3. You can use dot to create a lambda function

   ```r
   f <- . %>% filter(x="abc")
   f
   ```

   ```
   Functional sequence with the following components:

    1. filter(., x = "abc")

   Use 'functions' to extract the individual functions.
   ```

2 Reading CSV file

R built-in function to load a CSV file is the `read.csv()` function, but I prefer to use `read_csv()` function from `tidyverse` package. It's more versatile and you have more controls over correctness of the data.

This example demonstrates how to read a CSV file, correcting some column data types, and creating new categorical variables.

The data is taken from echocardiographic exams, but we only select some variables and cases to reduce the dimension for the sake of the explanation of this example.

```r
library(tidyverse)
```

2.1 Without correction

```r
read_csv('sample_data/echo_data.csv', show_col_types = FALSE) %>%
  select(StudyID, PatientID, Sex, Age_At_Echo, Examination_Date, Outcome, SoV, STJ, AscAo)
  sample_n(10000) %>%
  glimpse()
```

```
Rows: 10,000
Columns: 9
$ StudyID          <dbl> 2063479, 1129045, 1386632, 1971327, 2114340, 1360726,~
$ PatientID        <dbl> 772658, 535499, 337076, 721132, 850426, 372974, 69885~
$ Sex              <chr> "Female", "Male", "Female", "Male", "Male", "Female",~
$ Age_At_Echo      <dbl> 66, 83, 31, 79, 77, 48, 55, 18, 49, 78, 86, 93, 57, 5~
$ Examination_Date <date> 2014-06-02, 2015-07-16, 2013-10-01, 2015-02-10, 2012~
$ Outcome          <chr> "Alive", "Dead", "Alive", "Dead", "Dead", "Alive", "A~
$ SoV              <dbl> 3.800, 2.700, 2.930, 3.215, 4.300, 3.620, 3.423, 2.90~
$ STJ              <dbl> NA, NA, NA, NA, 4.200, NA, NA, NA, NA, NA, NA, NA, 3.~
$ AscAo            <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 3.4, NA, ~
```

2.2 With corrections

and some new variables added

```r
dt <- read_csv('sample_data/echo_data.csv', show_col_types = FALSE,
         col_types = cols(
           PatientID = col_character(),
           StudyID = col_character(),
           Examination_Date = col_date("%Y-%m-%d"),
           Sex = col_factor(),
           Outcome = col_factor())) %>%
  # select few variables for the sake of simplicity
  select(StudyID, PatientID, Sex, Age_At_Echo, Examination_Date, Outcome, SoV, STJ, AscAo)
  # just take randomly 10,000 rows for this demonstration
  sample_n(10000) %>%
  mutate(
    # we want to create a new variable Age that consists of range of ages
    Age = case_when(
      Age_At_Echo < 40 ~ "< 40",
      Age_At_Echo < 50 ~ "40-50",
      Age_At_Echo < 60 ~ "50-60",
      Age_At_Echo < 70 ~ "60-70",
      Age_At_Echo >= 70 ~ "  70"
    ) %>% factor(levels=c("< 40", "40-50", "50-60", "60-70", "  70"))
  ) %>%
  # another computation is to create a new variable called Aorta_Size
  # which takes the maximum value between SoV, STJ, and Asc_Ao values
  # for each scan, then categorise it to 4 groups of severities
  rowwise() %>%
  mutate(
    AortaSize = max(SoV, STJ, AscAo, na.rm=TRUE),
    AortaSize_cat = case_when(
      AortaSize <= 4.0 ~ "Normal",
      AortaSize <= 4.5 ~ "Mild",
      AortaSize <= 5.0 ~ "Moderate",
      AortaSize <= 9.0 ~ "Severe",
      .default = NA) %>% factor(levels = c("Normal", "Mild", "Moderate", "Severe"))) %>%
  ungroup()

# show the structure
glimpse(dt)
```

```
Rows: 10,000
Columns: 12
$ StudyID          <chr> "863177", "1015020", "1510051", "1949173", "891322", ~
$ PatientID        <chr> "487745", "468862", "518925", "708559", "118310", "48~
$ Sex              <fct> Male, Female, Male, Male, Female, Female, Male, Femal~
$ Age_At_Echo      <dbl> 64, 78, 85, 55, 67, 77, 64, 87, 54, 74, 87, 62, 60, 5~
$ Examination_Date <date> 2014-05-14, 2011-03-24, 2011-08-02, 2013-04-09, 2014~
$ Outcome          <fct> Alive, Alive, Alive, Alive, Alive, Dead, Alive, Dead,~
$ SoV              <dbl> 3.390, 3.300, 3.700, 3.102, 3.310, 2.582, 4.000, 3.30~
$ STJ              <dbl> NA, NA, NA, NA, NA, 2.51, 3.80, NA, 3.10, NA, NA, NA,~
$ AscAo            <dbl> 3.3, NA, NA, NA, 2.9, NA, NA, NA, NA, 2.9, NA, NA, NA~
$ Age              <fct> 60-70,   70,   70, 50-60, 60-70,   70, 60-70,   70, 5~
$ AortaSize        <dbl> 3.390, 3.300, 3.700, 3.102, 3.310, 2.582, 4.000, 3.30~
$ AortaSize_cat    <fct> Normal, Normal, Normal, Normal, Normal, Normal, Norma~
```

Explanations

1. Correcting data types

The argument of

```
col_types = cols(
  PatientID = col_character(),
  StudyID = col_character(),
  Examination_Date = col_date("%Y-%m-%d"),
  Sex = col_factor(),
  Outcome = col_factor()
)
```

forces `read_csv()` to use specific data types for specific columns (see: `cols` specification).

2. Create a new variable with `mutate`

The statement

```
mutate(
  Age = case_when(
    Age_At_Echo < 40 ~ "< 40",
    Age_At_Echo < 50 ~ "40-50",
    Age_At_Echo < 60 ~ "50-60",
    Age_At_Echo < 70 ~ "60-70",
    Age_At_Echo >= 70 ~ "  70"
    ) %>% factor(levels=c("< 40", "40-50", "50-60", "60-70", "  70"))
)
```

creates a new column `Age` as a factor that shows a range of ages between `<40, 40-50, 50-60, 60-70,  70` years old.

There is also another `mutate` statement to create a new column `AortaSize` and `AortaSize_cat` based on the maximum value between `STJ`, `SoV` and `AscAo` measurements. I separated this creation from the above because we need to specify R to calculate the maximum value row-wise instead of column-wise. Hence the `rowwise()` function preceded.

```
rowwise() %>%
mutate(
  AortaSize = max(SoV, STJ, AscAo, na.rm=TRUE),
  AortaSize_cat = case_when(
    AortaSize <= 4.0 ~ "Normal",
    AortaSize <= 4.5 ~ "Mild",
    AortaSize <= 5.0 ~ "Moderate",
    AortaSize <= 9.0 ~ "Severe",
    .default = NA) %>% factor(levels = c("Normal", "Mild", "Moderate", "Severe"))
)
```

See more about `mutate`, `case_when`, and `rowwise` functions.

2.3 Table summary

Let's summarise our data to compare all patients based on their survival: *dead* or *alive*.

```
library('gtsummary')
```

Note that the data may contain multiple scans for a patient. Thus, we will search the earliest scan first for each patient for the comparison.

```
dt %>%
  # analyse examination date per patient
  group_by(PatientID) %>%
  mutate(
    Earliest_Date = min(Examination_Date)
  ) %>%
  # release the grouping and now filter the earliest date only
  ungroup() %>%
  filter(Examination_Date == Earliest_Date) %>%
  # this should filter out multiple scan
  # we can safely give the data to tbl_summary function
  tbl_summary(
    by = Outcome,
    include = c(Sex, Age, SoV, STJ, AscAo, AortaSize, AortaSize_cat),
    missing = "no"
  ) %>%
  add_p() %>%
  separate_p_footnotes()
```

```
Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.
```

Characteristic	**Alive**, N = 7,063	**Dead**, N = 2,826	**p-value**
Sex			<0.001
Male	3,710 (53%)	1,592 (56%)	
Female	3,353 (47%)	1,234 (44%)	
Age			<0.001
< 40	1,084 (15%)	82 (2.9%)	
40-50	890 (13%)	114 (4.0%)	
50-60	1,374 (19%)	229 (8.1%)	
60-70	1,719 (24%)	499 (18%)	
70	1,996 (28%)	1,902 (67%)	
SoV	3.30 (2.96, 3.60)	3.37 (3.01, 3.70)	<0.001
STJ	3.20 (2.71, 3.50)	3.40 (2.80, 3.70)	<0.001
AscAo	3.30 (3.00, 3.60)	3.40 (3.20, 3.70)	<0.001
AortaSize	3.30 (3.00, 3.60)	3.40 (3.04, 3.70)	<0.001
AortaSize_cat			<0.001
Normal	6,548 (93%)	2,556 (90%)	
Mild	433 (6.1%)	223 (7.9%)	
Moderate	78 (1.1%)	34 (1.2%)	
Severe	4 (<0.1%)	13 (0.5%)	

3 Different ways to summarise

```r
library(tidyverse)
```

Let's use `mtcars` data, but first we need to convert two numeric variables into factor:

```r
# fix some numeric variables into factor
dt <- mutate( mtcars,
  vs = factor(vs, levels=c(0, 1), labels=c("V-shaped", "straight")),
  am = factor(am, levels=c(0, 1), labels=c("automatic", "manual"))
)

glimpse(dt)
```

```
Rows: 32
Columns: 11
$ mpg  <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, 17.8,~
$ cyl  <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4, 4, 4, 4, 8,~
$ disp <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.7, 140.8, 16~
$ hp   <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 180, 180, 180~
$ drat <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, 3.92, 3.92,~
$ wt   <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.190, 3.150, 3.~
$ qsec <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.00, 22.90, 18~
$ vs   <fct> V-shaped, V-shaped, straight, straight, V-shaped, straight, V-sha~
$ am   <fct> manual, manual, manual, automatic, automatic, automatic, automati~
$ gear <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, 4, 4, 3, 3,~
$ carb <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4, 1, 2, 1, 1, 2,~
```

3.1 desc_table

```r
library(desctable)
```

The library `desctable` provides `desc_table()` function to calculate main descriptive statistics. The output is a new dataframe. You can also change the output using `desc_output()` function.

Numeric variables

```
desc_table(dt %>% select(-c(am, vs))) %>% desc_output('pander')
```

	Min	Q1	Med	Mean	Q3	Max	sd	IQR
mpg	10	15	19	20	23	34	6	7.4
cyl	4	4	6	6.2	8	8	1.8	4
disp	71	121	196	231	326	472	124	205
hp	52	96	123	147	180	335	69	84
drat	2.8	3.1	3.7	3.6	3.9	4.9	0.53	0.84
wt	1.5	2.6	3.3	3.2	3.6	5.4	0.98	1
qsec	14	17	18	18	19	23	1.8	2
gear	3	3	4	3.7	4	5	0.74	1
carb	1	2	2	2.8	4	8	1.6	2

Categorical variables

```
desc_table(dt %>% select(c(am, vs))) %>% desc_output("pander")
```

	N	%
am	32	
automatic	19	59
manual	13	41
vs	32	
V-shaped	18	56
straight	14	44

See more: https://cran.r-project.org/web/packages/desctable/vignettes/desctable.html

3.2 skim

```
library(skimr)
```

`skim()` from `skimr` package provides a complete summary separated between numeric and categorical variables. Interestingly, histogram bars are shown in the last column for numeric variables.

```
skim(dt)
```

Table 3.3: Data summary

Name	dt
Number of rows	32
Number of columns	11
Column type frequency:	
factor	2
numeric	9
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
vs	0	1	FALSE	2	V-s: 18, str: 14
am	0	1	FALSE	2	aut: 19, man: 13

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
mpg	0	1	20.09	6.03	10.40	15.43	19.20	22.80	33.90	
cyl	0	1	6.19	1.79	4.00	4.00	6.00	8.00	8.00	
disp	0	1	230.72	123.94	71.10	120.83	196.30	326.00	472.00	
hp	0	1	146.69	68.56	52.00	96.50	123.00	180.00	335.00	
drat	0	1	3.60	0.53	2.76	3.08	3.70	3.92	4.93	
wt	0	1	3.22	0.98	1.51	2.58	3.33	3.61	5.42	
qsec	0	1	17.85	1.79	14.50	16.89	17.71	18.90	22.90	
gear	0	1	3.69	0.74	3.00	3.00	4.00	4.00	5.00	
carb	0	1	2.81	1.62	1.00	2.00	2.00	4.00	8.00	

See more: https://cran.r-project.org/web/packages/skimr/vignettes/skimr.html

3.3 tableone

```
library(tableone)
```

Table 1 is a common name used in biomedical research paper that describes the patient demographics. A package called `tableone` aims to ease the production of this table, and we can use this package to summarise our data.

```
CreateTableOne(data=dt, strata="am")
```

```
                 Stratified by am
                  automatic       manual          p      test
n                     19              13
mpg (mean (SD))    17.15 (3.83)    24.39 (6.17)   <0.001
cyl (mean (SD))     6.95 (1.54)     5.08 (1.55)    0.002
disp (mean (SD))  290.38 (110.17) 143.53 (87.20)  <0.001
hp (mean (SD))    160.26 (53.91)  126.85 (84.06)   0.180
drat (mean (SD))    3.29 (0.39)     4.05 (0.36)   <0.001
wt (mean (SD))      3.77 (0.78)     2.41 (0.62)   <0.001
qsec (mean (SD))   18.18 (1.75)    17.36 (1.79)    0.206
vs = straight (%)      7 (36.8)        7 ( 53.8)   0.556
am = manual (%)        0 ( 0.0)       13 (100.0)  <0.001
gear (mean (SD))    3.21 (0.42)     4.38 (0.51)   <0.001
carb (mean (SD))    2.74 (1.15)     2.92 (2.18)    0.754
```

3.4 tbl_summary

```
library(gtsummary)
```

The `gtsummary` package provides a rich collection of functions for summarising tables and results from different statistical analyses, e.g. regression, survival analysis, etc.. The simple one, `tbl_summary()`, can generate a beautiful summary table, ready for publication.

```
dt %>%
  tbl_summary(
    by = am,
    label = c(
      mpg ~ "Miles/gallon (US)",
      cyl ~ "Number of cylinders",
      disp ~ "Displacement (cu.in)",
      hp ~ "Gross horsepower",
      drat ~ "Rear axle ratio",
      wt ~ "Weight (1,000 lbs)",
      qsec ~"Quarter mile time",
      vs ~ "Engine type",
      gear ~ "Number of forward gears",
      carb ~ "Number of carburators"
    ),
    statistic = c(all_continuous() ~ "{mean} ± {sd}")
  ) %>%
  add_p() %>%
  separate_p_footnotes() %>%
  add_overall(last = TRUE) %>%
  modify_spanning_header(c("stat_1", "stat_2") ~ "**Transmission**")
```

Characteristic	**automatic**, N = 19	**manual**, N = 13	**p-value**	**Overall**, N = 32
Miles/gallon (US)	17.1 ± 3.8	24.4 ± 6.2	0.002	20.1 ± 6.0
Number of cylinders			0.009	
4	3 (16%)	8 (62%)		11 (34%)
6	4 (21%)	3 (23%)		7 (22%)
8	12 (63%)	2 (15%)		14 (44%)
Displacement (cu.in)	290 ± 110	144 ± 87	<0.001	231 ± 124
Gross horsepower	160 ± 54	127 ± 84	0.046	147 ± 69
Rear axle ratio	3.29 ± 0.39	4.05 ± 0.36	<0.001	3.60 ± 0.53
Weight (1,000 lbs)	3.77 ± 0.78	2.41 ± 0.62	<0.001	3.22 ± 0.98
Quarter mile time	18.18 ± 1.75	17.36 ± 1.79	0.3	17.85 ± 1.79
Engine type			0.3	
V-shaped	12 (63%)	6 (46%)		18 (56%)
straight	7 (37%)	7 (54%)		14 (44%)
Number of forward gears			<0.001	
3	15 (79%)	0 (0%)		15 (47%)
4	4 (21%)	8 (62%)		12 (38%)
5	0 (0%)	5 (38%)		5 (16%)
Number of carburators			0.3	
1	3 (16%)	4 (31%)		7 (22%)

Characteristic	**automatic**, N = 19	**manual**, N = 13	**p-value**	**Overall**, N = 32
2	6 (32%)	4 (31%)		10 (31%)
3	3 (16%)	0 (0%)		3 (9.4%)
4	7 (37%)	3 (23%)		10 (31%)
6	0 (0%)	1 (7.7%)		1 (3.1%)
8	0 (0%)	1 (7.7%)		1 (3.1%)

See more: https://www.danieldsjoberg.com/gtsummary/index.html

4 Bland-Altman analysis

The seminal paper written by Douglas G. Altman and J. Martin Bland on Lancet in 1986 (Bland and Altman 1986) has been the most cited paper ever. The method compares values measured by two different observers, devices, or algorithm. If the two measurements are comparable, then we expect that

1. the bias or mean of differences will be around zero,
2. the differences are within the limits of agreement (usually ± 2 standard deviation), and
3. there is no systematic differences or a trend line as the size of the object is getting bigger or smaller.

Let's create a Bland-Altman plot.

4.1 Loading the data

This example uses datasets from a study that segmented human heart from cardiac MR images. Two measurements were performed: by an automatic segmentation method and by manual segmentation. We're going to use only the left ventricle volume at end-diastole (LVEDV) and the right ventricle volume at end-diastole (RVEDV).

```
library(tidyverse)
library(ggplot2)
library(knitr)
```

4.1.1 Automated segmentation

```
dt_auto <- read_csv('sample_data/auto_heart.csv', show_col_types = FALSE,
                    col_types = cols(
                      name = col_character(),
                      Frame = col_character(),
                      lv_vol = col_double(),
                      rv_vol = col_double()
                    )) %>%
  # the ED frame for automated segmentation is at Frame == '000'
  filter(Frame == '000') %>%
  # select only the necessary columns
  select(name, lv_vol, rv_vol) %>%
  # fix the name to match with patient ID's from the manual data
  mutate(
    name = str_replace(str_replace(name, 'HCMR_', ''), '_', '-'),
  ) %>%
  rename(LVEDV = lv_vol, RVEDV = rv_vol, patientID = name)

glimpse(dt_auto)
```

```
Rows: 2,561
Columns: 3
$ patientID <chr> "001-0001", "001-0002", "001-0003", "001-0004", "001-0005", ~
$ LVEDV     <dbl> 253.2800, 204.6309, 226.3057, 176.5268, 192.3013, 176.6057, ~
$ RVEDV     <dbl> 525.5834, 335.4120, 370.6469, 239.0387, 300.2459, 409.7711, ~
```

We have 2561 rows of `patientID`, `LVEDV`, and `RVEDV` for the automated segmentation method.

4.1.2 Manual segmentation

```
dt_man <- read_csv('sample_data/corelab_heart.csv', show_col_types = FALSE,
                   col_types = cols(
                     sitePatID = col_character(),
                     lvedv = col_double(),
                     rvedv = col_double()
                   )) %>%
  select(sitePatID, lvedv, rvedv) %>%
  rename(LVEDV = lvedv, RVEDV = rvedv, patientID = sitePatID)

glimpse(dt_man)
```

```
Rows: 2,752
Columns: 3
$ patientID <chr> "001-0001", "001-0002", "001-0003", "001-0004", "001-0005", ~
$ LVEDV     <dbl> 253.39, 171.29, 210.51, 184.92, 190.18, 166.26, 229.61, 246.~
$ RVEDV     <dbl> 214.25, 161.98, 156.02, 137.83, 174.10, 147.49, 181.40, 202.~
```

We have 2752 rows of `patientID`, `LVEDV`, and `RVEDV` from the manual segmentation method.

4.1.3 Join the tables

We need to join the two tables using `inner_join`. See more: join operations.

```
dt_joined <- inner_join(dt_auto, dt_man,
                        by = c("patientID" = "patientID"),
                        suffix = c("_auto", "_man"))

glimpse(dt_joined)
```

```
Rows: 2,540
Columns: 5
$ patientID  <chr> "001-0001", "001-0002", "001-0003", "001-0004", "001-0005",~
$ LVEDV_auto <dbl> 253.2800, 204.6309, 226.3057, 176.5268, 192.3013, 176.6057,~
$ RVEDV_auto <dbl> 525.5834, 335.4120, 370.6469, 239.0387, 300.2459, 409.7711,~
$ LVEDV_man  <dbl> 253.39, 171.29, 210.51, 184.92, 190.18, 166.26, 229.61, 246~
$ RVEDV_man  <dbl> 214.25, 161.98, 156.02, 137.83, 174.10, 147.49, 181.40, 202~
```

We have now 2540 rows of `patientID`, `LVEDV_auto`, `LVEDV_man`, `RVEDV_auto`, and `RVEDV_man`.

A sanity check: we can check if there are duplicate `patientID`'s in the table by checking if the number of unique patient ID the same with the number of rows

```
nrow(dt_joined) == nrow(distinct(dt_joined, patientID))
```

```
[1] TRUE
```

4.2 Calculate differences and averages

Bland-Altman analysis plots points, where

- x-axis is the average between the two methods, and
- y-axis is the differences between the two methods

Thus, we need to calculate the differences and the averages between the automated and segmentation methods for both LVEDV and RVEDV values.

```
dt_ba <- dt_joined %>%
  mutate(
    # differences between automated and manual segmentations
    LVEDV_diff = LVEDV_auto - LVEDV_man,
    RVEDV_diff = RVEDV_auto - RVEDV_man,
    # average between automated and manual segmentations
    LVEDV_avg = 0.5 * (LVEDV_auto + LVEDV_man),
    RVEDV_avg = 0.5 * (RVEDV_auto + RVEDV_man)
  )
```

which will give you a table like this:

```
knitr::kable(head(dt_ba, n=5))
```

patientID	LVEDV_auto	RVEDV_auto	LVEDV_man	RVEDV_man	LVEDV_diff	RVEDV_diff	LVEDV_avg	RVEDV_avg
001-0001	253.2800	525.5834	253.39	214.25	-0.1099607	311.3334	253.3350	369.9167
001-0002	204.6309	335.4120	171.29	161.98	33.3409099	173.4320	187.9605	248.6960
001-0003	226.3057	370.6469	210.51	156.02	15.7956930	214.6269	218.4078	263.3334
001-0004	176.5268	239.0387	184.92	137.83	-8.3932075	101.2087	180.7234	188.4343
001-0005	192.3013	300.2459	190.18	174.10	2.1212527	126.1459	191.2406	237.1729

4.3 LVEDV

We can calculate the bias and the limits of agreement: ±1.96 standard deviation:

```
(LVEDV_stats <-
   mean(dt_ba$LVEDV_diff, na.rm=TRUE) +
   c(-1.96, 0, 1.96) * sd(dt_ba$LVEDV_diff, na.rm=TRUE))
```

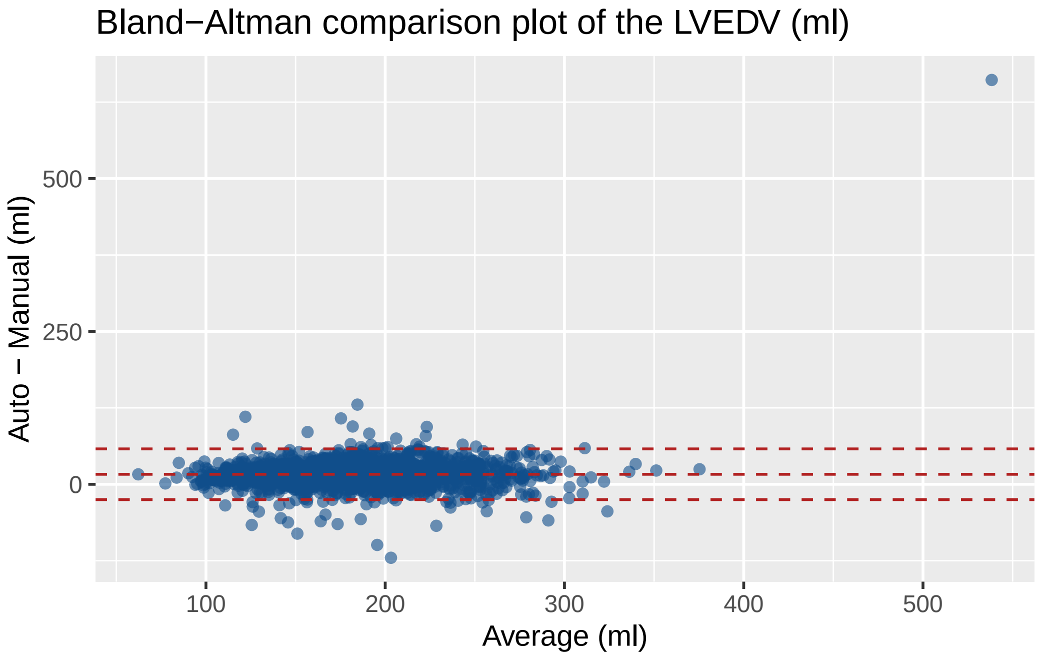
```
[1] -25.01554  16.42433  57.86419
```

This means that for LVEDV:

1. The bias is 16.424326 ml, and
2. The limit of agreements is between [-25.0155425, 57.8641945] ml.

Create the plot

```
dt_ba %>%
  # drop missing values to avoid warning from ggplot2
  drop_na(LVEDV_avg, LVEDV_diff) %>%
  # plot LVEDV_diff vs LVEDV_avg
  ggplot(aes(x=LVEDV_avg, y=LVEDV_diff)) +
  geom_point(col="dodgerblue4", alpha=0.6) +
  ylab("Auto - Manual (ml)") +
  xlab("Average (ml)") +
  ggtitle("Bland-Altman comparison plot of the LVEDV (ml)") +
  geom_hline(yintercept = LVEDV_stats, col="firebrick", lty=2)
```



One particular case needs to be checked for a gross error in the automated segmentation.

4.4 RVEDV

We can calculate the bias and the limits of agreement: ±1.96 standard deviation:

```r
(RVEDV_stats <-
   mean(dt_ba$RVEDV_diff, na.rm=TRUE) +
   c(-1.96, 0, 1.96) * sd(dt_ba$RVEDV_diff, na.rm=TRUE))
```

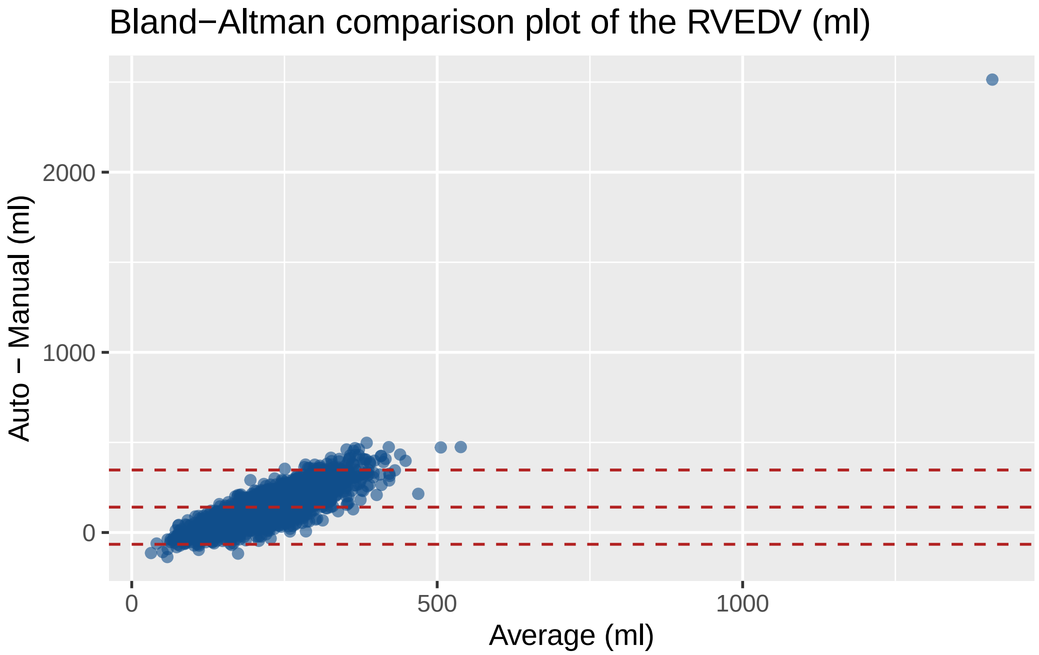
```
[1] -65.41394 140.74064 346.89523
```

This means that for LVEDV:

1. The bias is 140.7406444 ml, and
2. The limit of agreements is between [-65.4139384, 346.8952272] ml.

Create the plot

```r
dt_ba %>%
  # drop missing values to avoid warning from ggplot2
  drop_na(RVEDV_avg, RVEDV_diff) %>%
  # plot RVEDV_diff vs RVEDV_avg
  ggplot(aes(x=RVEDV_avg, y=RVEDV_diff)) +
  geom_point(col="dodgerblue4", alpha=0.6) +
  ylab("Auto - Manual (ml)") +
  xlab("Average (ml)") +
  ggtitle("Bland-Altman comparison plot of the RVEDV (ml)") +
  geom_hline(yintercept = RVEDV_stats, col="firebrick", lty=2)
```



There is clearly a systematic bias here with a strong trend line towards large bias when the size of the heart is increasing.

References

Bland, J Martin, and DouglasG Altman. 1986. “Statistical Methods for Assessing Agreement Between Two Methods of Clinical Measurement.” *The Lancet* 327 (8476): 307–10.