



R - Part II/II

ggplot2 function
predict.lm
break
aov
for
if
anova
ls
stopifnot
t.test
kruskal.test
binom.test
shapiro.test
wilcox.test
next
lm
chisq.test
repeat
FALSE
dplyr
while
TRUE
NaN
ks.test
else

Alexander Hirner
Aleksandra Tyjan

Version 6 (Juni 2024)

R - Part II

Version 6.0 (June 2024)

Alexander Hirner

Table of contents

1	Selected Functions from the <code>dplyr</code> Package	2
2	Selected Functions from the <code>tidyr</code> Package	10
3	Statistics	14
4	The <code>apply</code> -Family of Functions	21
5	Advanced Visualisation with <code>ggplot2</code> and <code>cowplot</code>	26
6	Loops and Simulation	43
7	Functions for the <code>matrix</code> (and <code>array</code>) Object	46
8	Recommended Reading	47

1 Selected Functions from the dplyr Package

1.1 Required Packages

Some very useful packages for data wrangling are

- dplyr
- tibble
- tidyr

I recommend installing the package `tidyverse` using `install.packages("tidyverse")`, which contains these packages and some more (e.g. `ggplot2`):

```
> library(tidyverse)
— Attaching packages — tidyverse 1.3.2 —
✓ ggplot2 3.3.6    ✓ purrr 0.3.4
✓ tibble 3.1.8     ✓ dplyr 1.0.9
✓ tidyr 1.2.0      ✓ stringr 1.4.0
✓ readr 2.1.2      ✓ forcats 0.5.2
— Conflicts — tidyverse_conflicts() —
✗ dplyr::filter() masks stats::filter()
✗ dplyr::lag()     masks stats::lag()
```

1.2 Sample Data

We use the following `data.frame`:

```
rm(list=ls())
set.seed(123)
var1 <- round(c(rnorm(n=30, mean=30, sd=2), rnorm(n=30, mean=35, sd=2)),3)
var2 <- var1 + 2
set.seed(456)
var3 <- round(rchisq(n=60, df=3),3)
var4 <- rep(c("A","B"), 30)
var5 <- sample(c("X","Y","Z"), 60, replace=TRUE)
mydata <- data.frame(var1, var2, var3, var4, var5)
print(head(mydata,3))
```

	var1	var2	var3	var4	var5
1	28.879	30.879	0.215	A	X
2	29.540	31.540	4.638	B	Y
3	33.117	35.117	1.225	A	Z

1.3 Advanced DM-Operations

Calculating New Variables

Function	Description
<code>dplyr::mutate(X, newvar1=formula1, newvar2=formula2, ...)</code>	Adds new variables to a <code>data.frame X</code> and preserves existing ones.
<code>dplyr::transmute(X, newvar1=formula1, newvar2=formula2, ...)</code>	Computes new columns in <code>X</code> and drops existing columns.

💡 What's wrong with `transform`?

Nothing. A similar base-R function that resembles `mutate` is `transform`. The advantage of `dplyr::mutate` is the ability that a newly created variable is “immediately available” and can be used in the formula for the next variable, which you want to create.

Example: Calculating New Variables Using `dplyr::transmute`

Note that every variable you do not “mention” in the function call, is deleted:

```
X <- dplyr::transmute(mydata, var1, var6=exp(var1), var7=sin(var2))
print(head(X,1))
```

```
      var1      var6      var7
1 28.879 3.483296e+12 -0.5114974
```

Selecting Variables

Function	Description
<code>dplyr::select(X, ...)</code>	Selects variables in a <code>data.frame X</code>

- Variables can be addressed by using their (unquoted) names (not their positions).
- The colon (`:`) selects a range of consecutive variables.
- The exclamation mark (`!`) takes the complement of a set of variables.
- Useful *selection helpers* are `starts_with`, `ends_with` and `contains`.
- The function `select` can also be used for rearranging variables (without dropping any of them).

Using the Pipe Operator (%>%)

The Pipe Operator %>%

The pipe operator (originally from the `magrittr`-package) pipes an object forward into a function or call expression, i.e. you can rewrite a function call `f(x)` as `x %>% f`, e.g.:

```
library(magrittr)
x = 7
x %>% sin
```

```
[1] 0.6569866
```

I personally recommend **to avoid** using this notation.

Sorting Values

Function	Description
<code>dplyr::arrange(X, var1, var2,...)</code>	Returns a sorted <code>data.frame</code> X . Use <code>desc(var)</code> to sort by a variable var in descending order.

Finding the “Top n” or “Bottom n” Observations

Function	Description
<code>dplyr::top_n(x, n, wt)</code>	Selects top n (or bottom n with negative value for n) rows by value.
<code>dplyr::slice_min(X, order_by, n)</code>	Rows with the smallest values of a variable.
<code>dplyr::slice_max(X, order_by, n)</code>	Rows with the largest values of a variable.

Selecting Observations

Function	Description
<code>dplyr::slice_head(X, n)</code>	Selects the first rows of your data X (use with an integer n or a fraction prop).
<code>dplyr::slice_tail(X, n)</code>	Selects the last rows of your data X (use with an integer n or a fraction prop).
<code>dplyr::slice_sample(X, n)</code>	Randomly selects observations from X (use with an integer n or a fraction prop).

Function	Description
<code>dplyr::filter(X, condition)</code>	Selects observations from X based on a logical condition.

Useful operators for `dplyr::filter` are:

- Classical comparison operators (`>`, `==`, etc.)
- `&` (and), `|` (or), `!` (not) and `xor` (exclusive or)
- `is.na` (for missing values) `dplyr::between()` and `dplyr::near()`

Creating Summaries

Function	Description
<code>tibble::as_tibble(X)</code>	Converts (usually a <code>data.frame</code> X) to a <i>tibble</i> .
<code>tibble::tibble(col1, col2, ...)</code>	Builds a <i>tibble</i> from individual columns.
<code>dplyr::group_by(X, groupvar)</code>	Groups data in X , returns a “grouped table”, i.e. all operations will be performed groupwise.
<code>dplyr::ungroup(X)</code>	Undo the grouping.
<code>dplyr::summarize(X, ...)</code>	Summarizes your data (see example).

Example: Creating (Grouped) Summaries

```
X <- mydata
head(X, 4)
```

	var1	var2	var3	var4	var5
1	28.879	30.879	0.215	A	X
2	29.540	31.540	4.638	B	Y
3	33.117	35.117	1.225	A	Z
4	30.141	32.141	0.826	B	X

```
X <- mydata
X <- dplyr::group_by(X, var5)
result1 <- dplyr::summarize(X, MEANV1 = mean(var1), SDV1=sd(var1), COUNT=dplyr::n())
print(result1)
```

```
# A tibble: 3 x 4
  var5 MEANV1 SDV1 COUNT
<chr> <dbl> <dbl> <int>
1 X      32.2  3.05     25
2 Y      32.6  3.34     19
3 Z      33.5  3.63     16
```

Now undo the grouping and calculate the summary again:

```
X <- dplyr::ungroup(X)
result2 <- dplyr::summarize(X, MEANV1 = mean(var1), SDV1=sd(var1), COUNT=dplyr::n())
print(result2)
```

```
# A tibble: 1 x 3
  MEANV1 SDV1 COUNT
<dbl> <dbl> <int>
1  32.6  3.29     60
```

Joins in R

Joins - Mutating Joins

The following *joins* are **mutating joins**. They combine variables from two tables:

Function	Description
<code>dplyr::inner_join(x, y, by)</code>	Inner Join
<code>dplyr::left_join(x, y, by)</code>	Left Join
<code>dplyr::right_join(x, y, by)</code>	Right Join
<code>dplyr::full_join(x, y, by)</code>	Full Join
<code>dplyr::cross_join(x, y)</code>	Cross Join

Joins - Filtering Joins

The following *joins* are **filtering joins**. They keep cases from the “left” table:

Function	Description
<code>dplyr::semi_join(x, y, by)</code>	Returns all rows from the left table, where there are matching variables in the right table, keeping just columns from the left table (without duplicates).
<code>dplyr::anti_join(x, y, by)</code>	Returns all rows from the left table where there are not matching values in the right table, keeping just columns from the left table.

Sample Data for the Joins

```
t1v1 <- c(1,2,2,3)
t1v2 <- c("a","b","c","d")
t1 <- data.frame(t1v1, t1v2)

t2v1 <- c(1,2,4)
t2v2 <- c("A","B","C")
t2 <- data.frame(t2v1, t2v2)
```

```
print(t1)
```

	t1v1	t1v2
1	1	a
2	2	b
3	2	c
4	3	d

```
print(t2)
```

	t2v1	t2v2
1	1	A
2	2	B
3	4	C

Example: Left Inner Join

```
# left (inner) join
X1 <- dplyr::left_join(x=t1, y=t2, by=c("t1v1"="t2v1"))
X1
```

	t1v1	t1v2	t2v2
1	1	a	A
2	2	b	B
3	2	c	B
4	3	d	<NA>

Example: Inner Join

```
# inner join
X2 <- dplyr::inner_join(x=t1, y=t2, by=c("t1v1"="t2v1"))
X2
```

	t1v1	t1v2	t2v2
1	1	a	A
2	2	b	B
3	2	c	B

Example: Right Join

```
# right join
X2 <- dplyr::right_join(x=t1, y=t2, by=c("t1v1"="t2v1"))
X2
```

	t1v1	t1v2	t2v2
1	1	a	A
2	2	b	B
3	2	c	B
4	4	<NA>	C

Example: Full Join

```
# full join
X2 <- dplyr::full_join(x=t1, y=t2, by=c("t1v1"="t2v1"))
X2
```

	t1v1	t1v2	t2v2
1	1	a	A
2	2	b	B
3	2	c	B
4	3	d	<NA>
5	4	<NA>	C

Example: Cross Join

```
# cross join
X2 <- dplyr::cross_join(x=t1, y=t2)
head(X2, 8)
```

	t1v1	t1v2	t2v1	t2v2
1	1	a	1	A
2	1	a	2	B
3	1	a	4	C
4	2	b	1	A
5	2	b	2	B
6	2	b	4	C
7	2	c	1	A
8	2	c	2	B

2 Selected Functions from the tidyr Package

2.1 Create Sample Data (a tibble):

```
id <- c(1,2,3,4,5)
vname <-c("Alfred", "Beate", "Christian", "Doris", "Eva")
height <- 171:175
weight <- 72:68
mydata <- tibble::tibble(id, vname, height, weight)
print(mydata)
```

A tibble: 5 x 4

	id	vname	height	weight
	<dbl>	<chr>	<int>	<int>
1	1	Alfred	171	72
2	2	Beate	172	71
3	3	Christian	173	70
4	4	Doris	174	69
5	5	Eva	175	68

2.2 Using tidyr::gather

With `gather` we can change a table to *long format*:

```
X1 <- tidyr::gather(mydata, key="Attribute", value="Value", height, weight)
head(X1,7)
```

A tibble: 7 x 4

	id	vname	Attribute	Value
	<dbl>	<chr>	<chr>	<int>
1	1	Alfred	height	171
2	2	Beate	height	172
3	3	Christian	height	173
4	4	Doris	height	174
5	5	Eva	height	175
6	1	Alfred	weight	72
7	2	Beate	weight	71

2.3 Using tidyr::spread

The function `spread` is the opposite of `gather` - we can convert a table back to *wide format*:

```
X2 <- tidyr::spread(X1, key="Attribute", value="Value")
head(X2, 2)
```

```
# A tibble: 2 x 4
      id vname  height weight
  <dbl> <chr>   <int>   <int>
1     1  Alfred    171     72
2     2   Beate    172     71
```

2.4 Using tidyr::unite

The function `unite` unites multiple columns into one column:

```
X3 <- mydata
X3 <- tidyr::unite(mydata, col="alltogether", sep=";")
head(X3, 2)
```

```
# A tibble: 2 x 1
  alltogether
  <chr>
1 1;Alfred;171;72
2 2;Beate;172;71
```

2.5 Using tidyr::separate

The opposite of the function `unite` is `separate`:

```
X4 <- tidyr::separate(data=X3, col=alltogether, into=c("id", "name", "height", "weight"))
head(X4, 2)
```

```
# A tibble: 2 x 4
      id  name  height weight
  <chr> <chr>   <chr>   <chr>
1 1    Alfred 171     72
2 2    Beate 172     71
```

2.6 Handling Missing Values

The following functions from the `tidyr` package will be useful if you deal with missing values:

Useful Functions

Function	Description
<code>drop_na(X)</code>	Drops rows where any column has missing values.
<code>fill(X, variable, .direction)</code>	Fills missing values in selected columns with next or previous entry.
<code>replace_na(X, list(var=value, ...))</code>	Replaces NAs with specified values.

Examples

```
x <- c(1, 2, NA, 4, 5)
y <- c(1, NA, 3, 4, 5)
z <- c("group1", NA, NA, "group2", NA)
Z <- data.frame(x, y, z)
Z
```

```
  x y      z
1  1 1 group1
2  2 NA <NA>
3 NA 3 <NA>
4  4 4 group2
5  5 5 <NA>
```

Example: fill

```
Z1 <- tidyr::fill(Z, z, .direction="down")
Z1
```

```
  x y      z
1  1 1 group1
2  2 NA group1
3 NA 3 group1
4  4 4 group2
5  5 5 group2
```

Example: drop_na

```
Z2 <- tidyr::drop_na(Z)
Z2
```

```
  x y      z
1 1 1 group1
2 4 4 group2
```

Example: replace_na

```
Z3 <- tidyr::replace_na(Z, list(x=0, y=0, z="no data"))
Z3
```

```
  x y      z
1 1 1 group1
2 2 0 no data
3 0 3 no data
4 4 4 group2
5 5 5 no data
```

3 Statistics

3.1 Required Packages

- `stats` (no need to install)
- `cluster` - for cluster analysis

3.2 Sample Data

We will create a sample `data.frame`:

```
set.seed(123)
var1 <- round(c(rnorm(n=30, mean=30, sd=2), rnorm(n=30, mean=35, sd=2)),3)
var2 <- var1 + 2
set.seed(456)
var3 <- round(rchisq(n=60, df=3),3)
var4 <- rep(c("A","B"), 30)
var5 <- sample(c("X","Y","Z"), 60, replace=TRUE)
mydata <- data.frame(var1, var2, var3, var4, var5)
print(head(mydata,5))
```

	var1	var2	var3	var4	var5
1	28.879	30.879	0.215	A	X
2	29.540	31.540	4.638	B	Y
3	33.117	35.117	1.225	A	Z
4	30.141	32.141	0.826	B	X
5	30.259	32.259	0.745	A	X

We have 60 rows and five variables:

```
print(dim(mydata))
```

```
[1] 60 5
```

3.3 Statistical Tests

The table below lists the most important statistical tests.

- To find out more about their details, use the help function, e.g. `help(binom.test)` or the abbreviated version, e.g. `?binom.test`.
- The keyword `formula` appears in many functions (e.g. for the T-Test) - it is usually an expression similar to `var1 ~ var2` where `var1` is the “outcome” and `var2` is the “independent” (or sometimes grouping) variable.
- It is a good idea to **save** your output from a test in a variable (which is usually a list or a list-like object).

3.4 Statistical Tests - Overview

T-Tests

Test	Description
<code>t.test(x, mu, alternative)</code>	Performs a one sample T-Test
<code>t.test(formula, data)</code>	Performs a Welch-Test
<code>t.test(formula, data, var.equal=TRUE)</code>	Performs a T-Test for independent samples .
<code>t.test(x, y, paired=TRUE)</code>	Performs a T-Test for dependent samples (i.e. repeated measurements).
<code>pairwise.t.test(x, g)</code>	Calculate pairwise comparisons between group levels with corrections for multiple testing.

Linear Modelling, ANOVA and Related Functions

Test	Description
<code>lm(formula, data)</code>	Fits linear models.
<code>predict.lm(object, newdata)</code>	Returns predicted values based on linear model object. The linear model object is obtained from the <code>lm</code> -function.
<code>anova(object)</code>	Computes analysis of variance tables. The argument <code>object</code> is a linear model (obtained from the <code>lm</code> -function).
<code>TukeyHSD(x)</code>	Computes Tukey Honest Significant Differences. The argument <code>x</code> is an ANOVA-Table.

Tests for Checking the Model Assumptions

Some statistical tests require a normal distribution in each group and equal variances among the groups.

Test	Description
<code>shapiro.test(x)</code>	Performs the Shapiro-Wilk Test of normality.
<code>bartlett.test(formula, data)</code>	Performs Bartlett's Test of the null that the variances in each group (samples) are the same.
<code>ks.test(x, y)</code>	Performs a one sample Kolmogorov-Smirnov Test . Here, <code>x</code> is the data and <code>y</code> a distribution (such as <code>pnorm</code> to test against the normal distribution)
<code>ks.test(x, y)</code>	Performs a two-sample Smirnov Test that <code>x</code> and <code>y</code> were drawn from the same <i>continuous</i> distribution.
<code>fligner.test(formula, data)</code>	Performs a Fligner-Killeen (median) Test of the null that the variances in each of the groups (samples) are the same.

Nonparametric Tests

Test	Description
<code>binom.test(x, n, p, alternative, conf.level)</code>	Performs an exact test (binomial test) of a simple null hypothesis about the probability of success in a Bernoulli Experiment .
<code>wilcox.test(formula,data)</code> <code>oneway.test(formula,data)</code>	Wilcoxon or Mann-Whitney-Test Test for Equal Means in a One-Way Layout (equal variances not necessarily assumed).
<code>kruskal.test(formula, data)</code>	Performs a Kruskal-Wallis Rank Sum Test .
<code>chisq.test(x)</code> <code>chisq.test(x, p, rescale.p)</code>	Test for contingency in a frequency table. Compares an empirical with a theoretical distribution. The values in <code>p</code> must be probabilities (unless you specify <code>rescale.p=TRUE</code>) and of the same length as your data.
<code>friedman.test(formula, data)</code>	Performs a Friedman Rank Sum Test with unreplicated blocked data.
<code>fisher.test(x)</code>	Performs Fisher's Exact Test for testing the null of independence of rows and columns in a contingency table with fixed marginals.
<code>mood.test(formula, data)</code>	Performs Mood's Two-Sample Test for a difference in scale parameters.

Test	Description
<code>ansari.test(formula, data)</code>	Performs the Ansari-Bradley Two-Sample Test for a difference in scale parameters.

Advanced Statistics

Function	Description
<code>clara(x,k,metric,stand)</code>	Clustering Large Applications. You need the cluster package. Here, x is your data, k the number of clusters, metric the metric used to calculate the distances between the objects, stand a logical value (i.e. TRUE or FALSE) that will indicate if your variables shall be standardized first.
<code>glm(formula, data, ...)</code>	Generalized Linear Models , e.g. logistic regression or poisson regression.
<code>predict.glm(object, newdata,...)</code>	Forecasting with a generalized linear model.
<code>princomp(x,...)</code>	Principal Components Analysis.

3.5 Examples

Example 1 - Binomial Test

We use the `data.frame` *mydata*. The outcome “X” should be counted as a success. We want to test if the probability for success is greater than 1/3:

```
binom.test(sum(X$var5=="X"), nrow(X),
p=1/3, alternative="greater")
```

Exact binomial test

```
data:  sum(X$var5 == "X") and nrow(X)
number of successes = 25, number of trials = 60, p-value = 0.1101
alternative hypothesis: true probability of success is greater than 0.3333333
95 percent confidence interval:
 0.3087114 1.0000000
sample estimates:
probability of success
 0.4166667
```

Example 2 - Comparing an Empirical with a Theoretical Distribution

We want to know if the variable *var4* from the `data.frame` *mydata* follows a discrete uniform distribution, i.e. if the values “X”, “Y” and “Z” have the same probability:

```
X <- mydata
table(mydata$var5)
```

```
 X  Y  Z
25 19 16
```

```
chisq.test(x = c(25, 19, 16), p=c(1,1,1), rescale.p=TRUE)
```

Chi-squared test for given probabilities

```
data:  c(25, 19, 16)
X-squared = 2.1, df = 2, p-value = 0.3499
```

Example 3 - χ^2 for Contingency Tables

```
tab <- table(mydata$var4, mydata$var5)
tab
```

```
  X  Y  Z
A 11 11  8
B 14  8  8
```

```
chisq.test(tab)
```

Pearson's Chi-squared test

```
data:  tab
X-squared = 0.83368, df = 2, p-value = 0.6591
```

Example 4 - ANOVA

We will make use of our sample data again to test if the means of *var1* differ among the groups defined in the variable *var5* using ANOVA (if it is possible). This is our data:

```
X <- mydata
print(head(X,2))
```

	var1	var2	var3	var4	var5
1	28.879	30.879	0.215	A	X
2	29.540	31.540	4.638	B	Y

Now check the assumptions:

```
# 1) assumption: equal variances
bartlett.test(var1 ~ var5, data=X)
```

Bartlett test of homogeneity of variances

data: var1 by var5

Bartlett's K-squared = 0.57779, df = 2, p-value = 0.7491

```
# 2) assumption: normality
doBy::summaryBy(var1 ~ var5, data=X, FUN=shapiro.test)[ , 1:4]
```

	var5	var1.statistic	var1.p.value	var1.method
1	X	0.9570081	0.3581135	Shapiro-Wilk normality test
2	Y	0.9465947	0.34527	Shapiro-Wilk normality test
3	Z	0.9398446	0.3471144	Shapiro-Wilk normality test

We calculate ANOVA using `lm()` and output the result in a well-known ANOVA-table:

```
anova(lm(var1 ~ var5, data=X))
```

Analysis of Variance Table

Response: var1

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
var5	2	16.82	8.4102	0.7715	0.4671
Residuals	57	621.39	10.9016		

Example 5 - Logistic Regression

```
X$outcome <- ifelse(X$var4=="A",1,0)
print(head(X,2))
```

```
      var1   var2  var3 var4 var5 outcome
1 28.879 30.879 0.215    A    X        1
2 29.540 31.540 4.638    B    Y        0
```

```
logisticmodel <- glm(outcome ~ var3, data=X, family=binomial(link="logit"))
print(confint(logisticmodel))
```

Waiting for profiling to be done...

```
              2.5 %    97.5 %
(Intercept) -0.7009819 0.8991228
var3         -0.2319846 0.1653252
```

```
summary(logisticmodel)
```

Call:

```
glm(formula = outcome ~ var3, family = binomial(link = "logit"),
    data = X)
```

Coefficients:

```
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.09357    0.40378   0.232   0.817
var3         -0.02958    0.09823  -0.301   0.763
```

(Dispersion parameter for binomial family taken to be 1)

```
Null deviance: 83.178  on 59  degrees of freedom
Residual deviance: 83.087  on 58  degrees of freedom
AIC: 87.087
```

Number of Fisher Scoring iterations: 3

```
# attach probabilities to raw data:
X$prob <- predict.glm(logisticmodel, X, type="response")
```

4 The apply-Family of Functions

4.1 Required Packages

No special packages required. All the `apply`-functions are included in the `base` package.

4.2 Create Sample Data

```
set.seed(123)
x <- matrix(rchisq(1000, df=3), ncol=4)
X <- as.data.frame(x)
X$V5 <- rep(c("A","B","C","D","E"), 50)
head(X)
```

	V1	V2	V3	V4	V5
1	1.03611518	1.0692228	1.167547	3.1752303	A
2	5.08870916	4.0810045	1.480796	0.7986091	B
3	0.04818784	1.1842015	1.141494	0.9105581	C
4	2.26693313	0.4574213	2.005387	0.7168796	D
5	6.90085393	1.1796676	11.123718	3.8133264	E
6	3.02805429	0.1332375	9.735566	0.2529623	A

```
dim(X)
```

```
[1] 250  5
```

4.3 The apply Function

Syntax

- **Description:** Returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.
- **Usage:** `apply(X, MARGIN, FUN)`.
 - X is your “rectangular” data.
 - MARGIN 1 row-wise operation, 2: column-wise operation.
 - FUN - function to apply.

Example

```
x1 <- apply(X[, -5], 1, sum)
length(x1)
```

```
[1] 250
```

```
# vector with 250 elements (i.e. number of rows in X)
```

4.4 The lapply Function

Syntax

- **Description:** Apply a function over a list or vector and returns a list.
- **Usage:** `lapply(X, FUN)`.
 - `X` is your input data.
 - `FUN` - function to apply.

Example

```
byV5 <- split(X, X$V5)
#byV5 is a list of data.frame objects.
x <- lapply(byV5, dim)
```

```
class(x)
```

```
[1] "list"
```

```
print(x)
```

```
$A
```

```
[1] 50  5
```

```
$B
```

```
[1] 50  5
```

```
$C
```

```
[1] 50  5
```

```
$D
```

```
[1] 50  5
```

```
$E
```

```
[1] 50  5
```

4.5 The sapply Function

Syntax

- **Description:** Applies a function over a list or vector and returns a vector or an array.
- **Usage:** `sapply(X, FUN)`.
 - `X` is your input data.
 - `FUN` - function to apply.

Example

```
byV5 <- split(X, X$V5)
x1 <- sapply(byV5, dim)
x1
```

```
      A B C D E
[1,] 50 50 50 50 50
[2,]  5  5  5  5  5
```

```
class(x1)
```

```
[1] "matrix" "array"
```

```
print(x1)
```

```
      A B C D E
[1,] 50 50 50 50 50
[2,]  5  5  5  5  5
```

4.6 The `tapply` Function

Syntax

- **Description:** Applies a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.
- **Usage:** `tapply(X$var1, X$var2, FUN)`.
 - `X` is a `data.frame` (or similar structure, e.g. a `tibble`)
 - `X$var1` is the variable to analyze.
 - `X$var2` is the grouping variable.
 - `FUN` - function to apply, usually something related to statistics, e.g. the mean or the median.

Examples

The average (arithmetic mean) for each group:

```
tapply(X$V1, X$V5, mean)
```

	A	B	C	D	E
	2.733125	2.949721	2.776092	2.456608	2.882719

The maximum value for each group:

```
tapply(X$V1, X$V5, max)
```

	A	B	C	D	E
	8.815004	8.963654	10.252916	11.083424	14.252381

5 Advanced Visualisation with ggplot2 and cowplot

5.1 Required packages

- ggplot2 - for the plots
- dplyr - for data management
- cowplot - to combine multiple plots on one page

5.2 Create Sample Data

We use the following `data.frame`:

```
set.seed(123)
var1 <- round(c(rnorm(n=100, mean=30, sd=2), rnorm(n=100, mean=35, sd=2)),3)
var2 <- var1 + rnorm(200, 0, 5)
set.seed(456)
var3 <- round(rchisq(n=200, df=3),3)
var4 <- rep(c("A","B"), 100)
var5 <- sample(c("X","Y","Z"), 200, replace=TRUE)
mydata <- data.frame(var1, var2, var3, var4, var5)
rm(var1, var2, var3, var4, var5)
head(mydata, 5)
```

	var1	var2	var3	var4	var5
1	28.879	39.87305	0.215	A	Z
2	29.540	36.10206	4.638	B	X
3	33.117	31.79127	1.225	A	Z
4	30.141	32.85697	0.826	B	Y
5	30.259	28.18730	0.745	A	Y

5.3 Plots Using ggplot2 by Example

We create 12 representative statistical plots (some scatterplots, a piechart, a histogram, etc) and later combine them into three graphs using the function `plot_grid` in the `cowplot`-package. It is always possible to create graphics files with the following code:

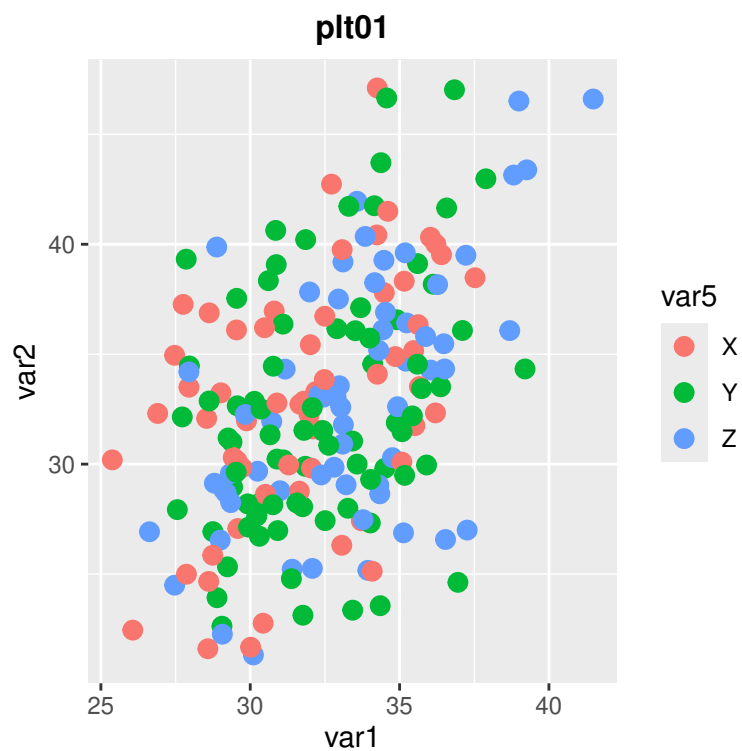
```
pdf("nameoffile.pdf") # or png(...), jpeg(...), ...
# your graphics commands here
dev.off()
```

Plot 01 - A Simple Scatterplot

```
head(mydata, 3)
```

	var1	var2	var3	var4	var5
1	28.879	39.87305	0.215	A	Z
2	29.540	36.10206	4.638	B	X
3	33.117	31.79127	1.225	A	Z

```
library(ggplot2)
plt01 <- ggplot(mydata, aes(x=var1, y=var2, color=var5))
plt01 <- plt01 + geom_point(size=3)
plt01 <- plt01 + ggtitle("plt01")
plt01 <- plt01 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))
plt01
```

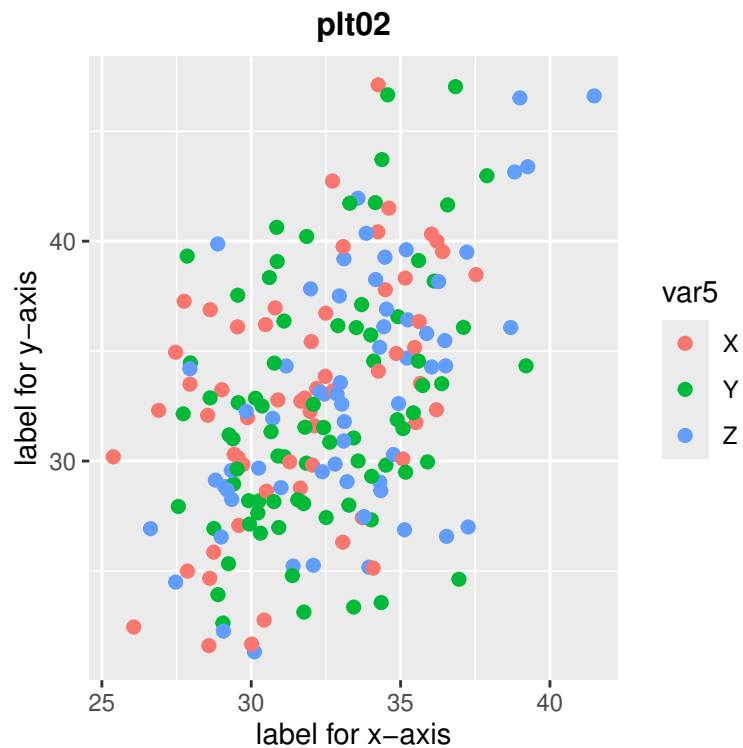


Plot 02 - Adding More Text

```
head(mydata, 3)
```

	var1	var2	var3	var4	var5
1	28.879	39.87305	0.215	A	Z
2	29.540	36.10206	4.638	B	X
3	33.117	31.79127	1.225	A	Z

```
plt02 <- ggplot(mydata, aes(x=var1, y=var2, color=var5))
plt02 <- plt02 + geom_point(size=2)
plt02 <- plt02 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))
plt02 <- plt02 + xlab("label for x-axis")
plt02 <- plt02 + ylab("label for y-axis")
plt02 <- plt02 + ggtitle("plt02")
plt02
```



Plot 03 - Formatting the Title and Annotations

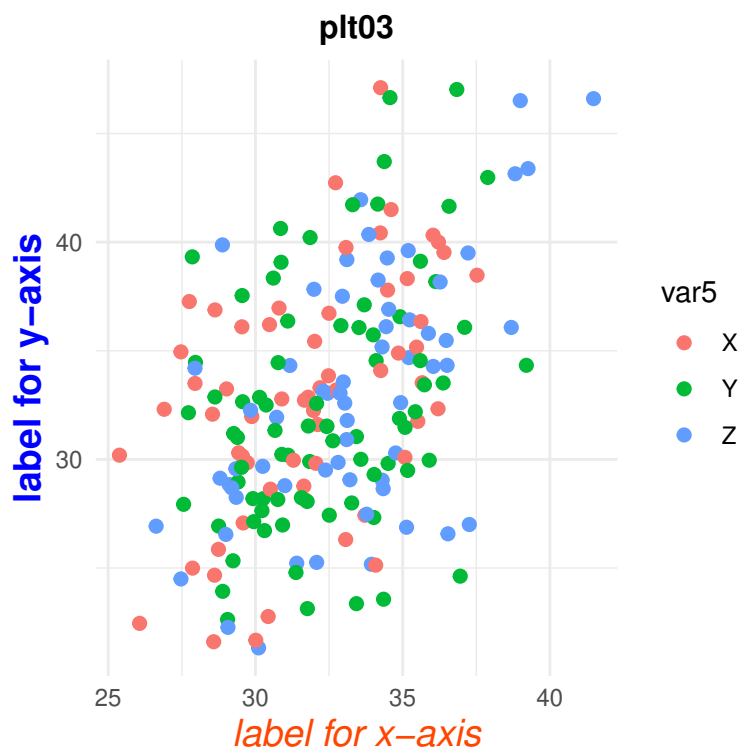
```
head(mydata, 3)
```

	var1	var2	var3	var4	var5
1	28.879	39.87305	0.215	A	Z
2	29.540	36.10206	4.638	B	X
3	33.117	31.79127	1.225	A	Z

```
plt03 <- ggplot(mydata, aes(x=var1, y=var2, color=var5))
plt03 <- plt03 + geom_point(size=2)
plt03 <- plt03 + theme_minimal()
plt03 <- plt03 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))
plt03 <- plt03 + xlab("label for x-axis") + ylab("label for y-axis")
plt03 <- plt03 + theme(axis.title.x = element_text(color="orangered",
                                                    size=14, face="italic"))

plt03 <- plt03 + theme(axis.title.y = element_text(color="blue",
                                                    size=14, face="bold"))

plt03<- plt03 + ggtitle("plt03")
plt03
```



Explanation: Formatting Text

With `element_text(...)` we change the appearance of texts, i.e. the title and axes annotations:

- `family` - font family.
- `face` - font face, e.g. “plain”, “italic” or “bold”.
- `colour` - (or `color`) - text color.
- `size` - text size in pts.
- `hjust` - horizontal justification (in $[0, 1]$).
- `vjust` - vertical justification (in $[0, 1]$).
- `lineheight` - line height (for multiline text only).

Explanation: Themes

It is possible to use predefined templates for the plots (in the previous example I used `theme_minimal()`). Here are some of them:

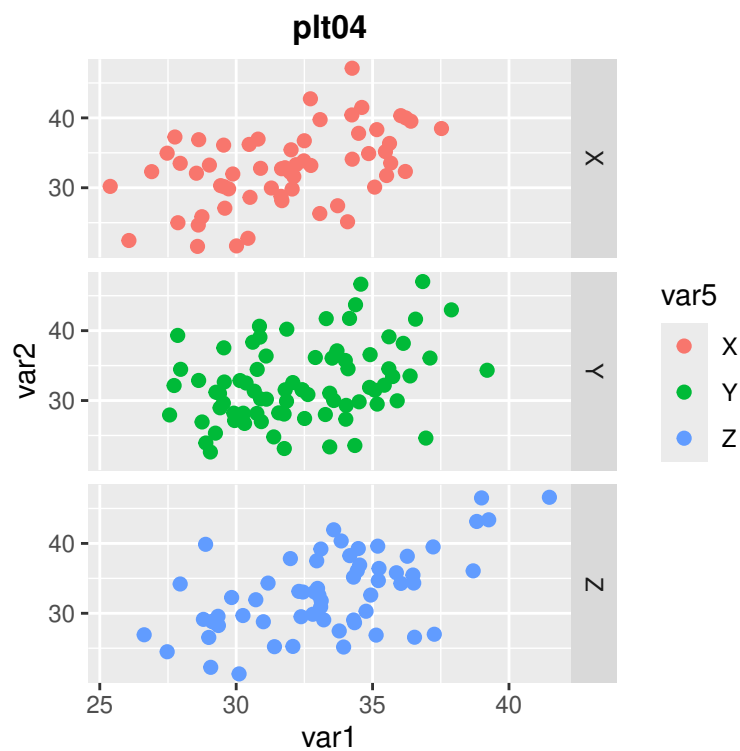
- `theme_grey()`
- `theme_bw()`
- `theme_light()`
- `theme_dark()`
- `theme_classic()`
- `theme_void()`

Plot 04 - Using facet_grid (Version 1)

```
head(mydata, 3)
```

```
  var1    var2  var3 var4 var5
1 28.879 39.87305 0.215   A   Z
2 29.540 36.10206 4.638   B   X
3 33.117 31.79127 1.225   A   Z
```

```
plt04 <- ggplot(mydata, aes(x=var1, y=var2, color=var5))
plt04 <- plt04 + geom_point(size=2)
plt04 <- plt04 + facet_grid(var5~.)
plt04 <- plt04 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))
plt04<- plt04 + ggtitle("plt04")
plt04
```

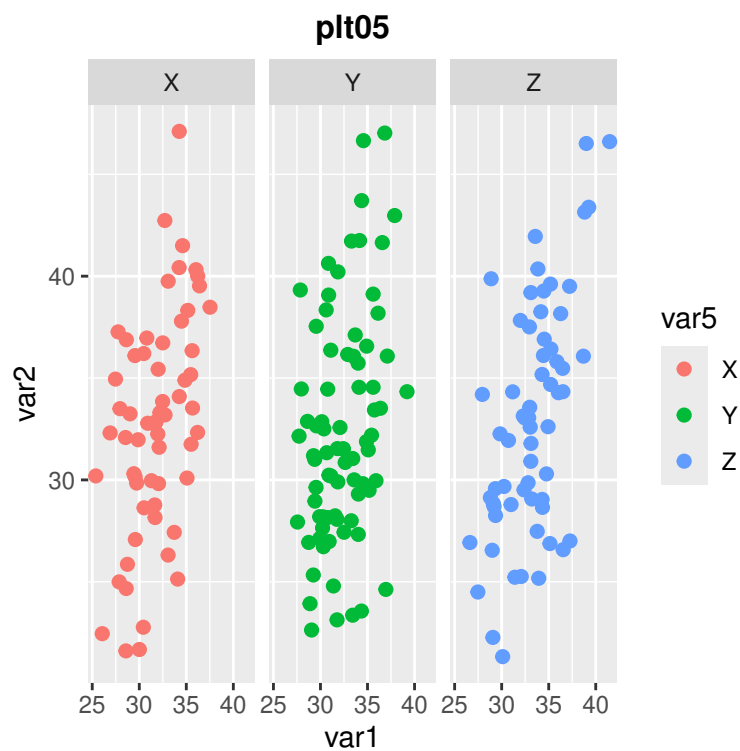


Plot 05 - Using facet_grid (Version 2)

```
head(mydata, 3)
```

```
  var1    var2  var3 var4 var5
1 28.879 39.87305 0.215   A   Z
2 29.540 36.10206 4.638   B   X
3 33.117 31.79127 1.225   A   Z
```

```
plt05 <- ggplot(mydata, aes(x=var1, y=var2, color=var5))
plt05 <- plt05 + geom_point(size=2)
plt05 <- plt05 + facet_grid(.~var5)
plt05 <- plt05 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))
plt05<- plt05 + ggtitle("plt05")
plt05
```

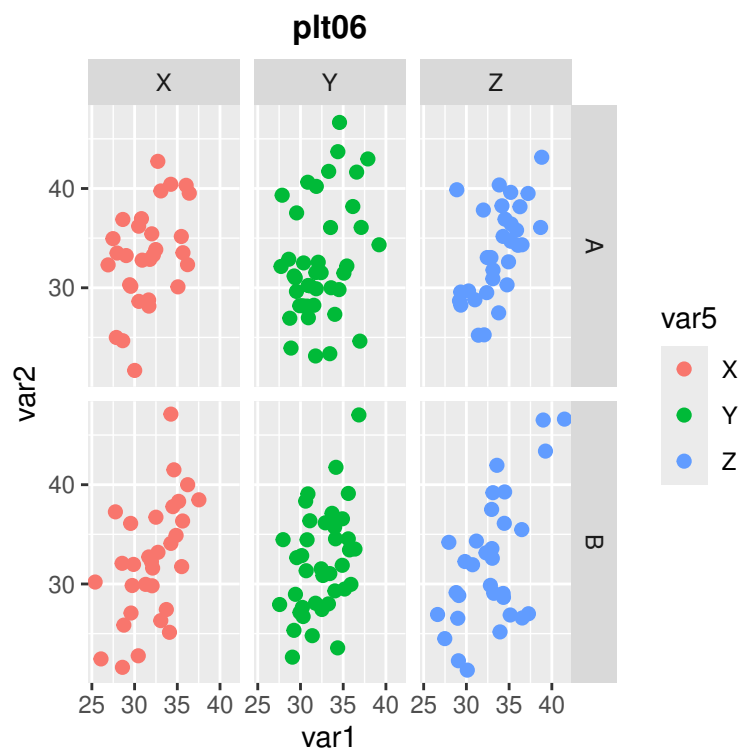


Plot 06 - Using facet_grid (Version 3)

```
head(mydata, 3)
```

```
   var1    var2  var3 var4 var5
1 28.879 39.87305 0.215   A    Z
2 29.540 36.10206 4.638   B    X
3 33.117 31.79127 1.225   A    Z
```

```
plt06 <- ggplot(mydata, aes(x=var1, y=var2, color=var5))
plt06 <- plt06 + geom_point(size=2)
plt06 <- plt06 + facet_grid(var4~var5)
plt06 <- plt06 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))
plt06<- plt06 + ggtitle("plt06")
plt06
```

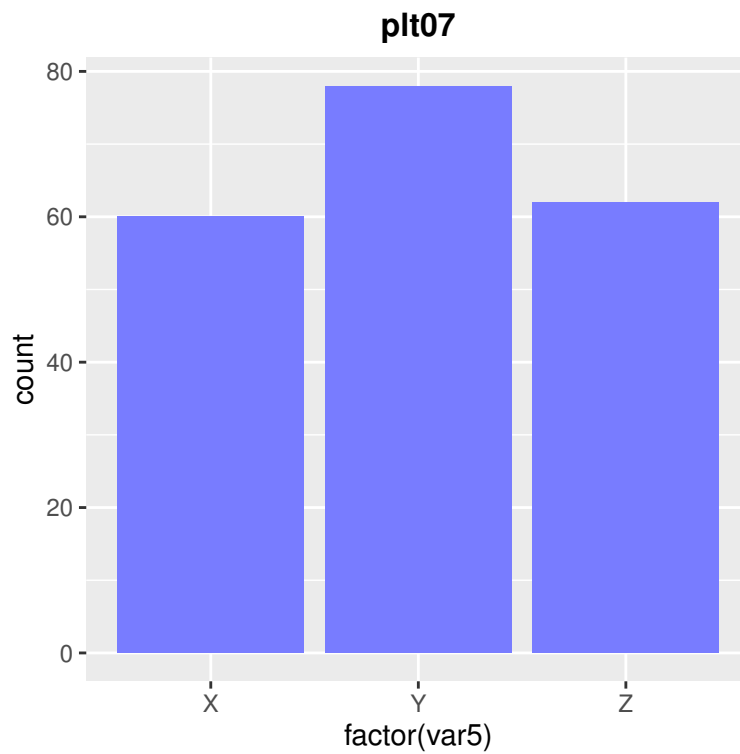


Plot 07 - A Barplot

```
head(mydata, 3)
```

	var1	var2	var3	var4	var5
1	28.879	39.87305	0.215	A	Z
2	29.540	36.10206	4.638	B	X
3	33.117	31.79127	1.225	A	Z

```
plt07 <- ggplot(mydata, aes(x=factor(var5)))  
plt07 <- plt07 + geom_bar(stat="count", fill="#787cff")  
  
plt07 <- plt07 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))  
plt07<- plt07 + ggtitle("plt07")  
plt07
```

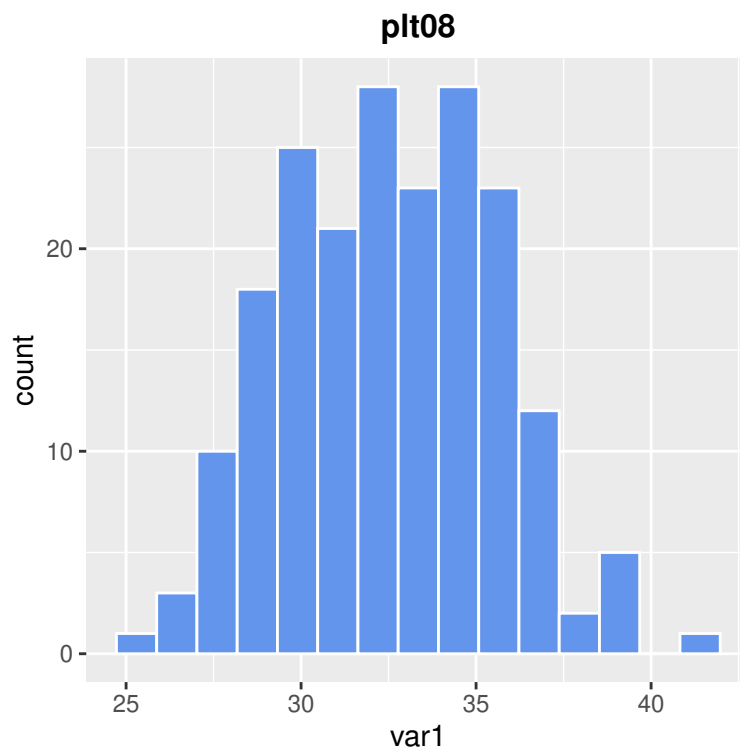


Plot 08 - A Histogram

```
head(mydata, 3)
```

	var1	var2	var3	var4	var5
1	28.879	39.87305	0.215	A	Z
2	29.540	36.10206	4.638	B	X
3	33.117	31.79127	1.225	A	Z

```
plt08 <- ggplot(mydata, aes(x=var1))  
plt08 <- plt08 + geom_histogram(color="white", fill="cornflowerblue", bins=15)  
plt08 <- plt08 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))  
plt08<- plt08 + ggtitle("plt08")  
plt08
```

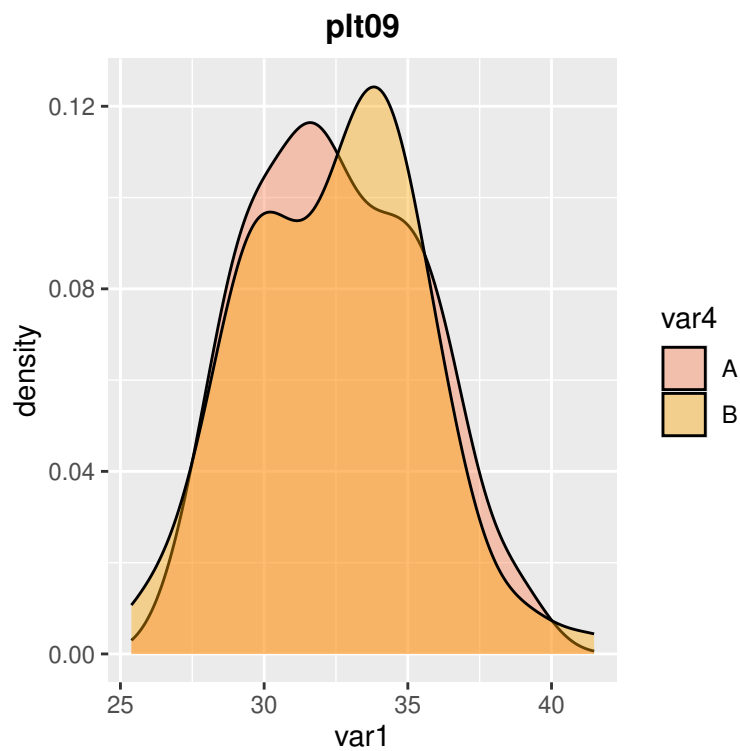


Plot 09 - Densities

```
head(mydata, 3)
```

	var1	var2	var3	var4	var5
1	28.879	39.87305	0.215	A	Z
2	29.540	36.10206	4.638	B	X
3	33.117	31.79127	1.225	A	Z

```
plt09 <- ggplot(mydata, aes(x=var1, fill=var4))  
plt09 <- plt09 + geom_density(alpha=0.4)  
plt09 <- plt09 + scale_fill_manual(values=c("coral", "orange"))  
plt09 <- plt09 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))  
plt09<- plt09 + ggtitle("plt09")  
plt09
```

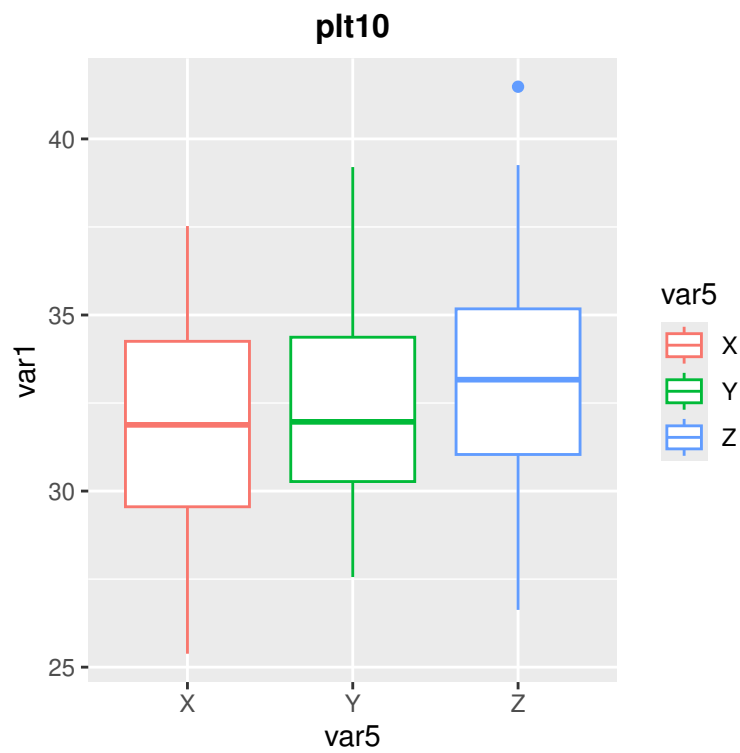


Plot 10 - Boxplot by Group

```
head(mydata, 3)
```

	var1	var2	var3	var4	var5
1	28.879	39.87305	0.215	A	Z
2	29.540	36.10206	4.638	B	X
3	33.117	31.79127	1.225	A	Z

```
plt10 <- ggplot(mydata, aes(x=var5, y=var1, color=var5))  
plt10 <- plt10 + geom_boxplot()  
plt10 <- plt10 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))  
plt10<- plt10 + ggtitle("plt10")  
plt10
```



Plot 11 - Another Barplot

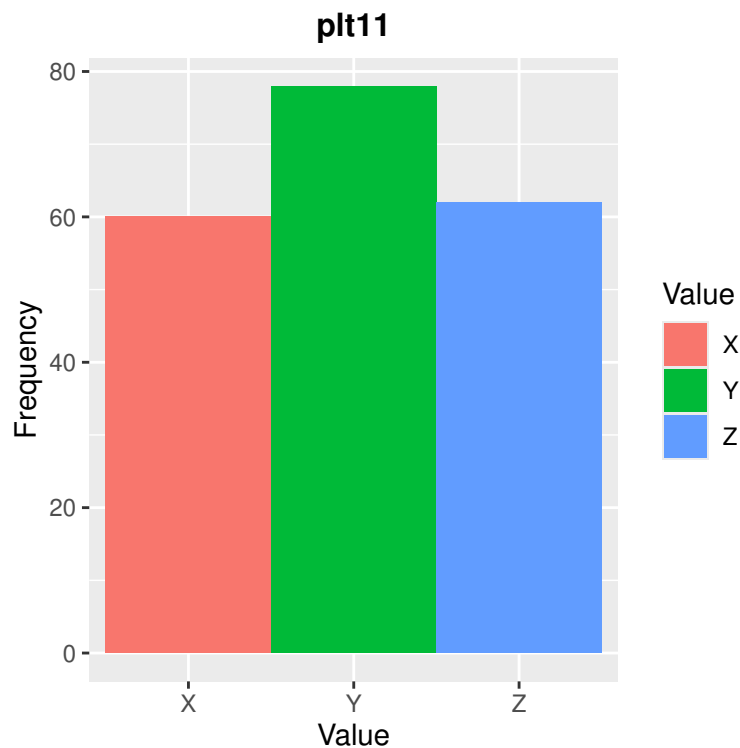
```
head(mydata, 1)
```

```
      var1      var2  var3 var4 var5  
1 28.879 39.87305 0.215   A    Z
```

```
mytab = table(mydata$var5); mytab <- as.data.frame(mytab)  
names(mytab) <- c("Value", "Frequency")  
mytab
```

```
Value Frequency  
1     X         60  
2     Y         78  
3     Z         62
```

```
plt11 <- ggplot(mytab, aes( x=Value, y=Frequency, fill=Value))  
plt11 <- plt11 + geom_bar(width=1, stat="identity")  
plt11 <- plt11 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))  
plt11<- plt11 + ggtitle("plt11")  
plt11
```

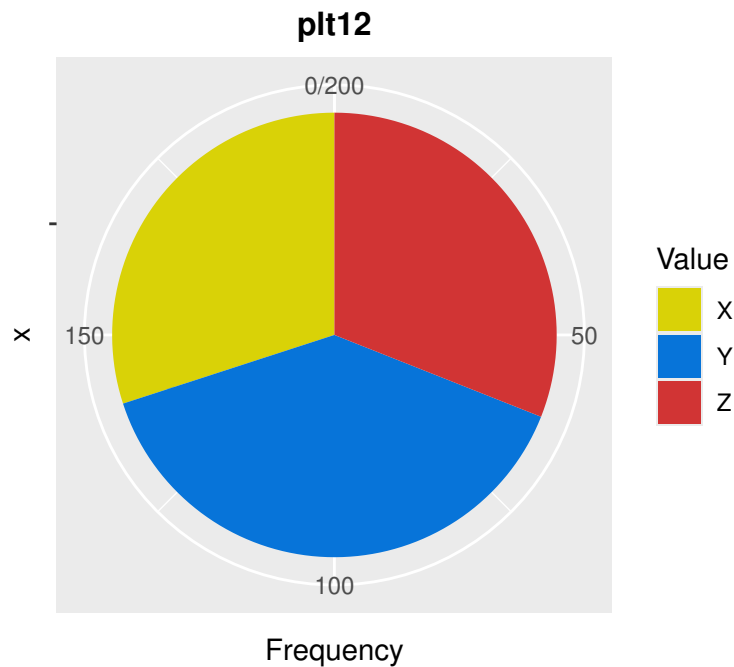


Plot 12 - A Piechart

```
head(mydata, 3)
```

	var1	var2	var3	var4	var5
1	28.879	39.87305	0.215	A	Z
2	29.540	36.10206	4.638	B	X
3	33.117	31.79127	1.225	A	Z

```
mytab = table(mydata$var5)
mytab <- as.data.frame(mytab)
names(mytab) <- c("Value", "Frequency")
plt12 <- ggplot(mytab, aes(x="", y=Frequency, fill=Value))+
  geom_bar(width = 1, stat = "identity")
plt12 <- plt12 + coord_polar("y", start=0)
plt12 <- plt12 + scale_fill_manual(values=c("#d9d207", "#0774d9", "#d13434"))
plt12 <- plt12 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))
plt12<- plt12 + ggtitle("plt12")
plt12
```

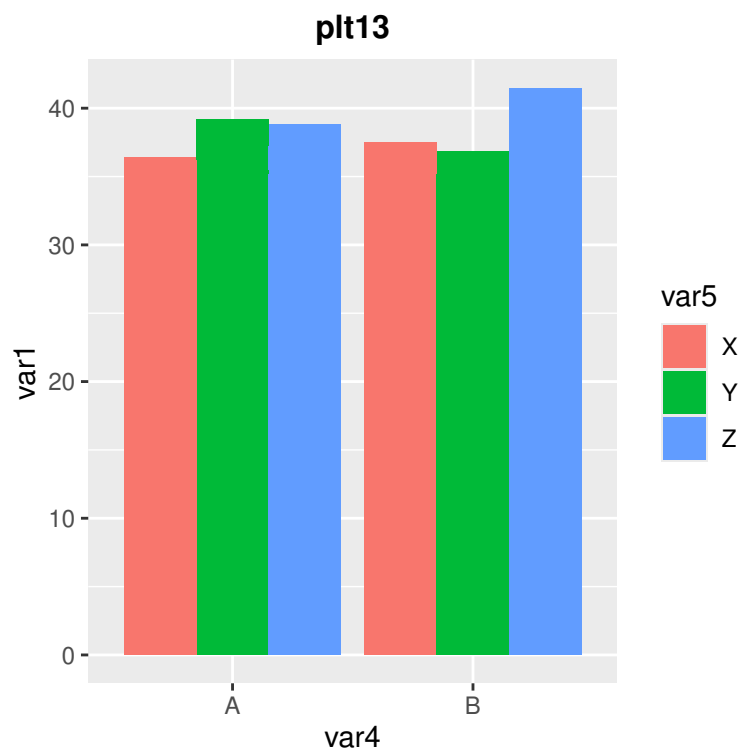


Plot 13 - A Stacked Barplot

```
head(mydata, 3)
```

	var1	var2	var3	var4	var5
1	28.879	39.87305	0.215	A	Z
2	29.540	36.10206	4.638	B	X
3	33.117	31.79127	1.225	A	Z

```
plt13 <- ggplot(mydata, aes(fill=var5, y=var1, x=var4))
plt13 <- plt13 + geom_bar(position="dodge", stat="identity")
plt13 <- plt13 + ggtitle("plt13")
plt13 <- plt13 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))
plt13<- plt13 + ggtitle("plt13")
plt13
```



Plot 14 - A Percent Stacked Barplot

```
head(mydata, 3)
```

	var1	var2	var3	var4	var5
1	28.879	39.87305	0.215	A	Z
2	29.540	36.10206	4.638	B	X
3	33.117	31.79127	1.225	A	Z

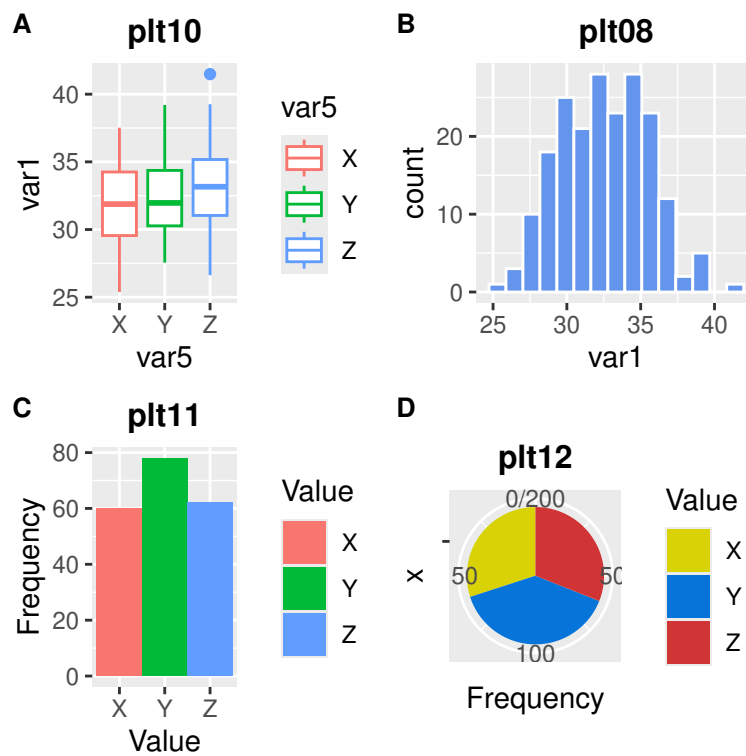
```
plt14 <- ggplot(mydata, aes(fill=var5, y=var1, x=var4))
plt14 <- plt14 + geom_bar(position="fill", stat="identity")
plt14 <- plt14 + ggtitle("plt14")
plt14 <- plt14 + theme(plot.title=element_text(size=12, face="bold", hjust=0.5 ))
plt14<- plt14 + ggtitle("plt14")
plt14
```



5.4 Combining the Plots

```
library(ggplot2)
library(cowplot)

plot_grid(plotlist = list(plt10, plt08, plt11, plt12),
  labels = c("A", "B", "C", "D"), label_size = 10, nrow=2)
```



6 Loops and Simulation

Example: Calculate the test statistic for a contingency table of two variables. One variable has four possible values, the other one has two possible values. If we calculate the measure of discrepancy, say, 5000 times and save these numbers, the distribution of these measures should closely resemble a χ^2 distribution with $(4 - 1) \times (2 - 1) = 3$ degrees of freedom.

Please install the `tidyverse` package first.

Code

```
library(ggplot2)

simul <- function(n){
  values = rep(NA, n)

  for (i in 1:n)
  {
    x <- sample(c("A","B","C","D"), 250, replace=TRUE)
    y <- sample(c("X", "Y"), 250, replace=TRUE)

    u <- table(x,y)

    result <- chisq.test(u)
    names(result)
    values[i] <- result$statistic
  }
  return(values)
}

n <- 5000
simresults <- simul(n)
# create a dataframe
mydata <- data.frame(simresults, theoretical = rchisq(n,3))
print(head(mydata,2))
```

	simresults	theoretical
1	2.302526	4.935269
2	3.056930	2.433447

```
# randomly select data for Smirnov Test
mydata_part <- dplyr::slice_sample(mydata, prop=0.1)
# Smirnov Test
print(ks.test(mydata_part$simresults, mydata_part$theoretical))
```

Asymptotic two-sample Kolmogorov-Smirnov test

data: mydata_part\$simresults and mydata_part\$theoretical

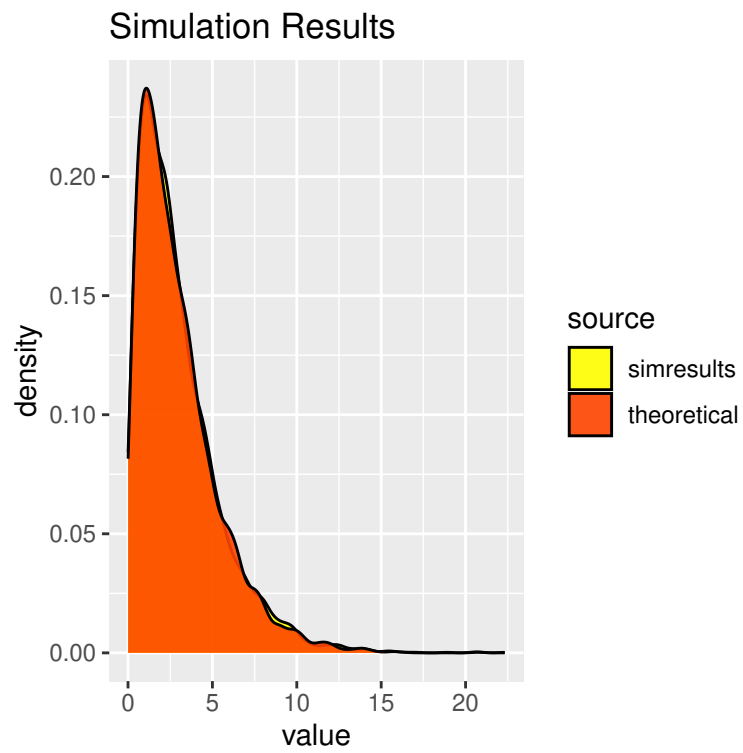
D = 0.058, p-value = 0.3696

alternative hypothesis: two-sided

```
X <- tidyr::gather(mydata, key="source", value="value", simresults, theoretical)
print(head(X, 2))
```

	source	value
1	simresults	2.302526
2	simresults	3.056930

```
plt <- ggplot(X, aes(x=value, fill=source))
plt <- plt + geom_density(alpha=.9)
plt <- plt + scale_fill_manual(values=c("yellow", "orangered"))
plt <- plt + ggtitle("Simulation Results")
plt
```



As expected from theoretical considerations we get an empirical distribution that closely resembles a χ^2 -distribution with $df = 3$.

7 Functions for the matrix (and array) Object

7.1 Functions

Function	Description
<code>matrix(data, nrow, byrow)</code>	Creates a matrix. For vectors use <code>c()</code> .
<code>array(data, dim)</code>	Creates an array.
<code>+, -, *, /</code>	Basic arithmetic operators.
<code>%*%</code>	Matrix multiplication.
<code>t(A)</code>	Transposes a matrix A.
<code>solve(A)</code>	Creates the inverse matrix A^{-1} .
<code>solve(A, b)</code>	Solves the linear system $Ax = b$.
<code>rank(A)</code>	Rank of a matrix A.
<code>det(A)</code>	Determinant of a Matrix A.
<code>eigen(A)</code>	Calculates eigenvalues and eigenvectors.
<code>qr(A)</code>	Computes the QR decomposition of a matrix.
<code>crossprod(A, B)</code>	Cross product.
<code>tcrossprod(A, B)</code>	Transposed cross product.

7.2 Examples

```
A <- matrix(c(1,2,3,4), nrow=2, byrow=TRUE)
Ainv <- solve(A)
print(A)
```

```
      [,1] [,2]
[1,]     1     2
[2,]     3     4
```

```
print(Ainv)
```

```
      [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
```

```
B <- A %*% Ainv
print(B)
```

```

      [,1]      [,2]
[1,]      1 1.110223e-16
[2,]      0 1.000000e+00

```

```

Z <- array(1:30, dim=c(2,5,3))
print(Z)

```

```

, , 1

```

```

      [,1] [,2] [,3] [,4] [,5]
[1,]      1      3      5      7      9
[2,]      2      4      6      8     10

```

```

, , 2

```

```

      [,1] [,2] [,3] [,4] [,5]
[1,]     11     13     15     17     19
[2,]     12     14     16     18     20

```

```

, , 3

```

```

      [,1] [,2] [,3] [,4] [,5]
[1,]     21     23     25     27     29
[2,]     22     24     26     28     30

```

```

print(Z[1,2,3])

```

```

[1] 23

```

8 Recommended Reading

- Everitt, B., Hothorn, T. (2011). *An Introduction to Applied Multivariate Analysis with R*. Berlin: Springer.
- Kabacoff, R. (2011). *R in Action. Data Analysis and Graphics with R*. Shelter Island: Manning Publications.
- Ligges, U. (2008). *Programmieren mit R*. (3. Auflage). Berlin/Heidelberg: Springer.
- Sauer, S. (2019). *Moderne Datenanalyse mit R: Daten einlesen, aufbereiten, visualisieren, modellieren und kommunizieren*. Wiesbaden: Springer.
- Wickham, H., Grommund, G. (2016). *R for Data Science. Import, tidy, transform, visualize and model data*. Sebastopol: O'Reilly Media.
- Zeltermann, D. (2015). *Applied Multivariate Statistics with R*. Berlin: Springer.