# 心理统计 II 课程完整笔记：1–13 章 R 实操（期中 + 期末整合）

周睿
北京大学心理与认知科学学院

2023 春

## 目录

# 1 引言

本文整合了北大心理与认知科学学院《心理统计 II》2023 春学期所有 13 章 R 实操，是一份**考前一站式参考手册**。每节都配完整的 R 代码 + 示例数据 + 运行结果（表格、回归输出、模型诊断图、KM 曲线、聚类树状图等）。

**13 章一句话导览**

**期中（第 1–7 章）**：

- **Ch.1–3 Non-Parametric Tests**：当数据不满足正态、连续、独立等参数假设时换用基于秩次的检验（Wilcoxon / Mann–Whitney U / Kruskal–Wallis / McNemar / Kolmogorov–Smirnov / Friedman）。
- **Ch.4 ANCOVA + Latin Square**：方差分析的两个扩展——ANCOVA 在 ANOVA 之外引入连续协变量控制混杂；Latin Square 是控制两个 nuisance 因子的实验设计。
- **Ch.5 Nested ANOVA + MANOVA**：处理层级结构数据（嵌套），以及多个因变量同时考察组间差异（MANOVA）。
- **Ch.6 Logistic Regression**：因变量是二分类时的回归（连续 OLS 不适用），用对数几率（log-odds）建模。
- **Ch.7 Multi-Linear Regression**：基础但最常用——用多个解释变量预测连续因变量，注意多重共线性、残差诊断、影响点。

**期末（第 8–13 章）**：

- **Ch.8 Moderation & Mediation**：调节是「$X \to Y$ 的强度受 $Z$ 影响」（交互项），中介是「$X \to M \to Y$」（间接效应 + bootstrap CI）。
- **Ch.9 Correlation & Distance**：区分零阶 / 偏 / 半偏相关，以及聚类前要懂的距离度量（欧氏 / 马氏 / 曼哈顿）。
- **Ch.10 PCA / EFA**：两种降维。PCA 是纯几何变换；EFA 假设潜在心理结构驱动观测变量。
- **Ch.11 Reliability Analysis**：量表信度，Cronbach $\alpha$、$\omega$、test-retest 等。
- **Ch.12 Survival Analysis**：Kaplan–Meier 估计 + Cox 比例风险模型，处理含右删失的时间-事件数据。

- **Ch.13 Clustering**：无监督学习入门，层次聚类 vs $k$-means，加上 $k$ 选择的轮廓系数 / elbow / 平行分析。

**这门课的核心脉络**

心理统计 II 的中心问题是：**当 ANOVA / OLS 的基本假设不满足时，统计推断要怎么变形？**

- 数据是**秩次而非真值**（小样本 / 离散）⇒ Ch.1–3 非参数
- 因子有**层级结构**或**多个因变量**并行 ⇒ Ch.5 嵌套 / MANOVA
- 因变量**不是连续**（二分 / 计数 / 含删失）⇒ Ch.6 Logistic、Ch.12 Survival
- 想理解**中间机制**而不只是关联 ⇒ Ch.8 中介
- 变量**太多需要降维**（高维 / 多重共线）⇒ Ch.10 PCA / EFA
- 想发现**隐藏分组**而不是验证已知组 ⇒ Ch.13 聚类

每一种方法都对应一类「假设违反」的修补——这是把整门课记牢的最稳骨架。

**阅读建议**：每个 r-tutorial 单章页（如 `/research/r-tutorials/r-pca`）有更详细的**核心概念框**（concept）和**易踩的坑**（pitfall），想精读某个方法的同学可以单独跳读对应单章。本文是「全景代码菜谱」。

# 2 第一部分：期中范围（1–7 章）

# 3 Non-Parametric Tests

Parametric statistics assumes data coming from a type of probability distribution (i.e., the parametric form of the distribution) and makes inferences about the parameters of the distribution. Estimation is about estimating the parameters; Hypothesis testing is about deriving the sampling distribution of a test statistic (e.g., t, F).

Non-parametric statistics uses **distribution-free** methods which do not rely on assumptions that the data are drawn from a given probability distribution (don't rely on the estimation of parameters).

The disadvantage of non-parametric tests is that they're less powerful! The probability of rejecting the null hypothesis is lower, and also a higher probability to make type II error when a true effect or difference exists.

Non-parametric tests are used when:

- Non-normality or skewness
- Outliers
- Small sample sizes Thus don't trust normality assumptions
- **Ordinal** or **Nominal** data

## 3.1 One Sample

### 3.1.1 Chi-Square Test

The chi-square test is a statistical hypothesis test that is used to determine **if there is a significant association between two categorical variables**. It is a non-parametric test, which means that it does not assume any particular distribution of the data.

The test involves comparing the *observed frequencies* of different categories with the *expected frequencies*, assuming that there is no association between the two variables. The expected frequencies are calculated based on the assumption of **independence** between the two variables.

The test statistic is calculated as

$$\chi^2 = \sum^{C} \frac{(f_o - f_e)^2}{f_e}$$

$$C = \#\text{Categories},\ f_o = \text{observed freq.},\ f_e = \text{expected freq.}$$

Note that the degree of freedom for chi-square statistic does not depend on the sample size, but only on number of categories.

The resulting value is then compared to the chi-square distribution. If the calculated chi-square value is greater than the critical value from the chi-square distribution, then the null hypothesis of independence is rejected, and it can be concluded that there is a significant association between the two variables.

Note that for chi-square test, under each category (bin), the sample size should be no less than 5, otherwise you have to merge some adjacent categories or use less bins.

**3.1.1.1 Goodness of Fit** The function takes one or more contingency tables as input and produces a chi-square test result object that contains the test statistic, degrees of freedom, p-value, and expected frequencies.

Here is the general syntax for using the chisq.test() function:

```
chisq.test(x, y = NULL,
           correct = TRUE,
           p = NULL, rescale.p = FALSE)
```

The `x` argument is the contingency table or data that is used to perform the chi-square test. If `y` is specified, a test for independence is performed, and

if `y` is not specified, a goodness of fit test is performed.

The `correct` argument is a logical value that indicates whether to apply a continuity correction to the test statistic. The `p` argument specifies the expected probabilities for the goodness of fit test. The `rescale.p` argument is a logical value that indicates whether to rescale the expected probabilities to sum to 1 (if necessary).

The most useful example for the test of goodness of fit is in linear models. Suppose with the well-known dataset `mtcars`, we're to check the fitting effect between predicted value and the real value, regressing `mpg` on `wt`.

```
chisq.test(fitted(lm(mpg~wt,data = mtcars)),
           mtcars$mpg)
```

```
##
##  Pearson's Chi-squared test
##
## data:  fitted(lm(mpg ~ wt, data = mtcars)) and mtcars$mpg
## X-squared = 682.67, df = 672, p-value = 0.3792
```

#### 3.1.1.2 Independece test

It's beautiful for the logic under the independence test, and its relationship with test of goodness of fit.

Here, we've successfully **transformed a problem of correlation into a problem of the similarity of distributions**. And such transformation is interesting and not twisted at all.

Suppose we have data on 500 individuals regarding their smoking habits and whether they have lung cancer. The data is summarized in the following contingency table:

```
smoking <- c("Never", "Former", "Current")
cancer <- c("Yes", "No")
observed <- matrix(c(50, 200, 100, 100, 50, 0),
                   nrow = 3, byrow = TRUE,
```

```
                  dimnames = list(smoking, cancer))
observed
```

```
##         Yes  No
## Never    50 200
## Former  100 100
## Current  50   0
```

In this table, the rows represent the smoking status of the individuals (`Never`, `Former`, `Current`), and the columns represent whether they have lung cancer (`Yes`, `No`).

Pass a contingency table into the `chisq.test()` directly and see the result, testing whether smoking is associated with lung cancer.

```
smoke = chisq.test(observed)
smoke
```

```
##
##  Pearson's Chi-squared test
##
## data:  observed
## X-squared = 125, df = 2, p-value < 2.2e-16
```

Moreover, we can also inspect the *expected* frequencies of the contingency table using the `result$expected` command. This will give us a matrix of the expected frequencies under the assumption of independence between smoking and lung cancer.

This will undoubtedly save great efficiency, in case that it's tested by the teacher, otherwise there's gonna be a huge efficiency loss.

```
smoke$expected
```

```
##         Yes  No
## Never   100 150
## Former   80 120
```

```
## Current  20  30
```

**3.1.1.3 Effect Size** `rstatix::cramer_v()` is a function in R that is used to compute the Cramer's V statistic, which is a measure of the association between two categorical variables. Cramer's V is a normalized version of the chi-square statistic and is often used as a measure of effect size in chi-square tests.

The parameters of this function is much the same as `chisq.test()`.

```
rstatix::cramer_v(observed)
```

```
## [1] 0.5
```

| | | |
|---|---|---|
| For $df^* = 1$ | $0.10 < V < 0.30$ | Small effect |
| | $0.30 < V < 0.50$ | Medium effect |
| | $V > 0.50$ | Large effect |
| For $df^* = 2$ | $0.07 < V < 0.21$ | Small effect |
| | $0.21 < V < 0.35$ | Medium effect |
| | $V > 0.35$ | Large effect |
| For $df^* = 3$ | $0.06 < V < 0.17$ | Small effect |
| | $0.17 < V < 0.29$ | Medium effect |
| | $V > 0.29$ | Large effect |

### 3.1.2 Binomial Test

The binomial test is a statistical hypothesis test that is used to determine **if the proportion of successes in a binary experiment is significantly different from a hypothesized value**. The test assumes that the outcome of each trial is either a success or a failure and that the trials are independent.

In the binomial test, the null hypothesis is that the proportion of successes is equal to a hypothesized value, and the alternative hypothesis is that the proportion of successes is different from the hypothesized value.

In R, the `binom.test()` function can be used to perform the binomial test. Here is an example of how to use the function:

```r
# Generate a vector of binary data
toss <- c(1, 0, 1, 1, 1, 0, 0, 1, 1, 0)

# Perform the binomial test
binom.test(sum(toss),length(toss),p = 0.5)
```

```
##
##  Exact binomial test
##
## data:  sum(toss) and length(toss)
## number of successes = 6, number of trials = 10, p-value = 0.7539
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.2623781 0.8784477
## sample estimates:
## probability of success
##                    0.6
```

In this example, we first generate a vector of binary data representing the outcomes of a binary experiment. We then use the `binom.test()` function to perform the binomial test on this data, specifying the number of successes (`sum(data)`), the number of trials (`length(data)`), and the hypothesized proportion of successes (`p = 0.5`).

The output of the test will include the test statistic, the p-value, and the confidence interval for the proportion of successes.

### 3.1.3 Runs Test

The runs test is a non-parametric test that can be used to **assess the randomness of a set of data**. The test is based on the number of runs in the data, where a run is defined as a sequence of consecutive observations with the same sign.

In statistics, a "run" is a sequence of consecutive data points that have the same value or the same sign. A run can be defined for any type of data, whether it's categorical or continuous.

For example, consider the following sequence of binary data:

1 1 0 1 0 0 0 1 1 1

In this sequence, there are 4 runs: two runs of 1's, one run of 0's, and one run of 1's.

Another example is a sequence of temperature readings over a period of time:

20.1 20.3 20.4 20.4 20.5 20.4 20.2 20.0

In this sequence, there is one run of increasing temperatures (from 20.1 to 20.5) followed by one run of decreasing temperatures (from 20.5 to 20.0).

The null hypothesis of the runs test is that the data is random, and the alternative hypothesis is that the data is not random. The test statistic is based on the number of runs in the data and is compared to the expected number of runs under the assumption of randomness.

The expected number of runs can be calculated using the following formula:

$$E(R) = \frac{2n_1n_2}{n_1 + n_2} + 1$$

where $n_1$ is the number of positive signs in the data, $n_2$ is the number of negative signs in the data.

The test statistic is given by:

$$z = \frac{R - E(R)}{Var(R)}$$

where $R$ is the observed number of runs, $E(R)$ is the expected number of runs, and $Var(R)$ is the variance of the number of runs. Under the null hypothesis of randomness, the test statistic follows a standard normal distribution.

Use `DescTools::RunsTest(data)` in R to determine whether a coin is randomly tossed or not:

```
# Generate a vector of coin toss outcomes (0 = tails, 1 = heads)
coin_tosses <- c(1, 0, 1, 1, 0, 1, 1, 0,
                 0, 1, 0, 0, 1, 0, 1, 1)

# Perform the runs test
DescTools::RunsTest(coin_tosses)
```

```
##
##  Runs Test for Randomness
##
## data:  coin_tosses
## runs = 11, m = 7, n = 9, p-value = 0.3024
## alternative hypothesis: true number of runs is not equal the expected number
```

### 3.1.4 KS Test

The Kolmogorov-Smirnov (KS) test is a nonparametric statistical test that is used to compare the distributions of two samples or to test whether a sample follows a specific distribution. The KS test is based on the maximum difference between the empirical cumulative distribution functions (ECDFs) of the two samples.

The KS test can be used to test the null hypothesis that the two samples are drawn from the same distribution or that a single sample is drawn from a specific distribution. The test is based on the KS statistic, which is defined as the maximum vertical difference between the ECDFs of the two samples.

To perform the KS test in R, you can use the `ks.test()` function, which is part of the base R package. Here's the syntax for the `ks.test()` function:

```
ks.test(x, y, alternative,
        exact = NULL,
        conf.level = 0.95, ...)
```

where `x` and `y` are the two samples to be compared, `alternative` is a character string specifying the alternative hypothesis (either "two.sided", "less", or "greater"), `exact` is a logical value indicating whether to use the exact distribution of the KS statistic (for small sample sizes), and `conf.level` is the confidence level for the test.

Here's an example of how to perform the KS test in R using the `ks.test()` function:

```
# Generate two samples of data from normal distributions
set.seed(123)
x <- rnorm(100, mean = 0, sd = 1)
y <- rnorm(100, mean = 0.8, sd = 1)

# Perform the KS test
ks.test(x, y)
```

```
##
##  Asymptotic two-sample Kolmogorov-Smirnov test
##
## data:  x and y
## D = 0.27, p-value = 0.001365
## alternative hypothesis: two-sided
```

In this example, we first generate two samples of data from normal distribu-

tions with different means. We then use the `ks.test()` function to perform the KS test on the two samples. The function returns a list containing the test statistic, the p-value, and the alternative hypothesis.

Moreover, when comparing your data to a specific distribution, you need to specify the distribution you want to compare your data to using the `y` argument, and also provide the parameters of the distribution using additional arguments `....` For example, if you want to compare your data to a normal distribution, you can provide the mean and standard deviation of the normal distribution using the mean and sd arguments, respectively. If you want to compare your data to a uniform distribution, you can provide the minimum and maximum values of the uniform distribution using the min and max arguments, respectively.

```r
# Generate a sample of data
set.seed(123)
x <- rnorm(100, mean = 0, sd = 1)

# Perform the KS test against a normal distribution
ks.test(x, "pnorm", mean = mean(x), sd = sd(x))
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.058097, p-value = 0.8884
## alternative hypothesis: two-sided
```

### 3.1.5 Kendall's Tau

Kendall's tau is a non-parametric measure of association for **ordinal** data that quantifies the strength and **direction** of the relationship between two variables. Kendall's tau is particularly useful when the variables being studied are ordinal or ranked.

Kendall's tau is based on the difference between the number of concordant and discordant pairs of observations. A concordant pair is a pair in which the ranks of both variables are in the same direction (i.e., both are either increasing or decreasing). A discordant pair is a pair in which the ranks of the variables are in the opposite direction (i.e., one is increasing while the other is decreasing).

Kendall's tau coefficient ranges from -1 to +1, with values of -1 indicating a perfect negative association, 0 indicating no association, and +1 indicating a perfect positive association. The formula for Kendall's tau is:

$$\tau = \frac{(\text{number of concordant pairs}) - (\text{number of discordant pairs})}{(\text{number of pairs})} = 1 - \frac{2(\text{number of discordant pairs})}{\binom{n}{2}}.$$

In R, you can calculate Kendall's tau using the `cor.test()` functionwith the argument `method = "kendall"`. The output of the test provides the estimated Kendall's tau coefficient, along with the p-value and confidence interval for the test.

In this example, we have ten companies in the tech industry and their ranks of revenue for two different years.

| Company | Year 1 Rank | Year 2 Rank |
|---|---|---|
| Apple | 1 | 2 |
| Microsoft | 2 | 1 |
| Amazon | 3 | 3 |
| Alphabet | 4 | 4 |
| Facebook | 5 | 5 |
| Intel | 6 | 6 |
| IBM | 7 | 7 |
| Samsung | 8 | 9 |
| Cisco | 9 | 8 |
| Qualcomm | 10 | 10 |

```r
rank_data <- data.frame(
  Company = c("Apple", "Microsoft", "Amazon",
              "Alphabet", "Facebook", "Intel",
              "IBM", "Samsung", "Cisco", "Qualcomm"),
  Year_1_Rank = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
  Year_2_Rank = c(2, 1, 3, 4, 5, 6, 7, 9, 8, 10)
)
cor.test(rank_data$Year_1_Rank,
         rank_data$Year_2_Rank,
         method = "kendall")
```

```
##
##  Kendall's rank correlation tau
##
## data:  rank_data$Year_1_Rank and rank_data$Year_2_Rank
## T = 43, p-value = 2.976e-05
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##       tau
## 0.9111111
```

The high tau and highly significant result indicate that there's a innegligible great relationship.

### 3.1.6 Kendall's W

Kendall's W is a measure of agreement among multiple raters or judges who are assessing the same set of items or objects. Kendall's W is based on the number of concordant and discordant triples of ratings among the raters.

Kendall's W ranges from 0 to 1, with values of 0 indicating no agreement among the raters and 1 indicating perfect agreement. The formula for Kendall's W is:

$$W = (n_c - n_d)/(n_c + n_d + n_t)$$

where $n_c$ is the number of concordant triples, $n_d$ is the number of discordant triples, and $n_t$ is the number of tied triples.

In another form,

$$W = \frac{\sum_i R_i^2 - \frac{(\sum_i R_i)^2}{N}}{\frac{1}{12}M^2(N^3 - N)}$$

where $R_i$ is the total ranking of $i$th subject.

Kendall's W can be converted into Spearman correlation.

$$r_{spearman} = \frac{MW - 1}{M - 1}$$

In R, you can calculate Kendall's W using the `irr::kendall(ratings)` function, where rating is a $n * m$ matrix, $n$ subjects and $m$ raters.

Here's an example of how to use the `irr::kendall()` function to calculate Kendall's W coefficient:

```
# Create a data frame with ratings from three judges for four items
ratings <- data.frame(
  Item = c("A", "B", "C", "D"),
  Judge_1 = c(3, 4, 2, 1),
  Judge_2 = c(2, 4, 1, 3),
  Judge_3 = c(3, 2, 1, 4))

irr::kendall(ratings = ratings)
```

```
##  Kendall's coefficient of concordance W
##
##  Subjects = 4
##    Raters = 4
```

```
##         W = 0.225
##
##  Chisq(3) = 2.7
##   p-value = 0.44
```

Meanwhile, a chi-square test is carried for each subject according to their sum of rankings. If significant, there exists a difference between their performance.

Note that there's no signal about the significance about W, only that about $\chi^2$.

## 3.2 Two Samples

### 3.2.1 Related Samples

**3.2.1.1 Wilcoxon Signed Ranks Test** The Wilcoxon Signed Ranks test is a nonparametric statistical test used to determine whether the **median** of a paired difference between two related samples is statistically different from zero. It is used when the data does not meet the assumptions of normality required for a parametric test like the paired t-test.

The test is performed by first calculating the differences between the paired observations, then ranking the absolute values of these differences. The sum of the ranks of the positive differences is then compared to the sum of the ranks of the negative differences, and take the smaller one as the statistic called $T$. The null hypothesis is that the median difference is zero, while the alternative hypothesis is that it is not zero.

In R, you can perform the Wilcoxon Signed Ranks test using the `wilcox.test()` function. `wilcox.test()` is a function in R that performs the **Wilcoxon rank test**, or the **Mann-Whitney U test**. When conducting the former, `paired = T`; the latter, `paired = F`. Just specify the two vectors as the first two parameters.

Here's an example of how to use the `wilcox.test()` function to perform

the test on two related samples. Suppose we are interested in comparing the effectiveness of two different weight loss programs. We have a sample of 10 individuals who have completed both programs. We measure each individual's weight before and after each program, and we want to determine whether there is a significant difference in weight loss between the two programs.

```
# Create two related samples
weightloss <- data.frame(
  individual = 1:10,
  program_A = c(185, 190, 200, 175, 195, 180, 170, 185, 195, 200),
  program_B = c(180, 185, 195, 170, 190, 175, 165, 180, 190, 195)
)

# Perform the Wilcoxon Signed Ranks test
wilcox.test(weightloss$program_A,weightloss$program_B, paired = TRUE)
```

```
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  weightloss$program_A and weightloss$program_B
## V = 55, p-value = 0.001904
## alternative hypothesis: true location shift is not equal to 0
```

Note:

- Faced with tied values, typically use the averge rank.
- Faced with zero difference, typically keep the observation. Deleting data is always a concern! Zero difference observation can participate in computing $T$ and influence the degree of freedom.

**3.2.1.2 McNemar Test** In the context of the McNemar Test, the off-diagonal terms refer to the counts of the two categories that are inconsistent between the two paired measurements. Specifically, they refer to the counts of the following two categories:

- False positives (FP): The number of cases where the first measurement is negative, but the second measurement is positive.
- False negatives (FN): The number of cases where the first measurement is positive, but the second measurement is negative.

For example, consider a medical study that measures the presence or absence of a certain disease in a group of patients before and after treatment. The 2x2 contingency table for this study might look like this:

```
            After Treatment
            Positive Negative
Before Treatment
Positive       a        b
Negative       c        d
```

In this table, `a` represents the number of patients who tested positive for the disease both before and after treatment (true positives), `d` represents the number of patients who tested negative for the disease both before and after treatment (true negatives), `b` represents the number of patients who tested negative for the disease before treatment but positive after treatment (false positives), and `c` represents the number of patients who tested positive for the disease before treatment but negative after treatment (false negatives).

The off-diagonal terms in the McNemar Test are used because they represent the counts of the inconsistent categories between two paired measurements. These inconsistent categories are the false positives (FP) and false negatives (FN), which represent the cases where the two measurements disagree.

In the context of the McNemar Test, the goal is to determine whether there is a significant difference between the proportions of the inconsistent categories, namely FP and FN. This is done by comparing the observed counts of FP and FN to the expected counts under the null hypothesis of no difference. The expected counts are calculated assuming that the marginal totals of the contingency table are fixed, which is a characteristic of paired data.

So the key difference between the Chi-Square Test on a Contingency Table and the McNemar Test is the question that each test addresses. The former tests for independence between two categorical variables, while the latter tests for a significant difference between two correlated proportions. The two tests have different assumptions and are used in different contexts, but both use the chi-squared statistic to calculate the test statistic.

Here's an example of the McNemar Test being used to analyze a medical experiment in R. In this example, we will examine the effectiveness of a new treatment for a certain disease compared to a standard treatment.

Suppose we have a sample of 50 patients who have been diagnosed with the disease. We randomly assign each patient to receive either the new treatment (Treatment A) or the standard treatment (Treatment B). After the treatment, we measure whether the patient's symptoms have improved or not. The results are summarized in the following 2x2 contingency table:

```
            After Treatment
            Improved Not Improved
Before Treatment
Improved         15         10
Not Improved     5         20
```

In this table, the rows represent the before-treatment condition and the columns represent the after-treatment condition. The cells represent the number of patients in each category. For example, 15 patients improved after receiving Treatment A, while 10 did not improve.

To analyze these data using the McNemar Test in R, we can use the `mcnemar.test()` function. Here's the R code:

```r
# Create a 2x2 contingency table of the data
medicine <- matrix(c(15, 5, 10, 20),
                   nrow = 2,
                   dimnames = list(c("Improved", "Not Improved"),
                                   c("Improved", "Not Improved")))
```

```r
# Perform the McNemar Test
mcnemar.test(medicine)
```

```
##
##  McNemar's Chi-squared test with continuity correction
##
## data:  medicine
## McNemar's chi-squared = 1.0667, df = 1, p-value = 0.3017
```

In this code, we first create a 2x2 contingency table of the data using the `matrix()` function. We then use the `mcnemar.test()` function with the contingency table as the input. The function returns a list of information about the test, including the chi-squared statistic, the degrees of freedom, and the p-value.

In the example above, we cannot reject the null hypothesis that there is no significant difference between the proportions of patients who improved after receiving Treatment A compared to Treatment B. McNemar test is used to compare the proportions of patients who improved after receiving a new treatment compared to a standard treatment. The output of the test provides information about the significance of the difference between the proportions, which can inform decisions about the effectiveness of the treatments.

**3.2.1.3 Sign Test** The Sign Test is a non-parametric statistical test that is used to compare two related samples or matched pairs in which the data are measured on an ordinal or continuous scale. The test is based on the distribution of the signs of the differences between the paired observations.

The Sign Test is used when the data do not meet the assumptions of a parametric test, such as the paired t-test, which assumes that the data are normally distributed. The Sign Test is more robust to outliers and non-normality, and it does not require knowledge of the distribution of the data.

The Sign Test is a one-sample test, meaning that it compares the differences between the paired observations to a specified value, such as zero. The null hypothesis is that the median difference between the paired observations is equal to zero, while the alternative hypothesis is that the median difference is not equal to zero.

To perform the Sign Test, we first calculate the differences between the paired observations. We then assign a sign (+ or -) to each difference, depending on whether the observation in the first sample is greater or less than the observation in the second sample. We then count the number of positive and negative signs, and calculate the test statistic as the minimum of the two counts. The test statistic follows a binomial distribution with the number of pairs as the sample size and a probability of 0.5 under the null hypothesis.

The p-value of the test can be calculated using the **binomial** distribution, or it can be *approximated* using a *normal* approximation. If the p-value is less than the chosen significance level, the null hypothesis is rejected, and the conclusion is that there is a significant difference between the two related samples.

The Sign Test can be useful in a variety of applications, including clinical trials, psychology experiments, and ecological studies. It is a simple and robust method for comparing two related samples when the data do not meet the assumptions of a parametric test.

There's no `sign.test()` in R, but that can be accomplished "manually" through `binomial.test()`, since the statistic follows such distribution. We only have to assign the signs of positive or negative according to the differences.

Same with the "weightloss" example in the Wilcoxon test, we'll continue that in the sign test.

```
binom.test(x = sum(weightloss$program_A > weightloss$program_B),
               n = nrow(weightloss))
```

```
##
##  Exact binomial test
##
## data:  sum(weightloss$program_A > weightloss$program_B) and nrow(weightloss)
## number of successes = 10, number of trials = 10, p-value = 0.001953
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.6915029 1.0000000
## sample estimates:
## probability of success
##                      1
```

Note that the tests for signed ranks of related samples may vary a lot in terms of their significance level, which largely indicates that the effectiveness of nonparametric tests may be limited. However, the **Wilcoxon test** is generally more recognized.

### 3.2.2 Independent Samples

**3.2.2.1 Mann-Whitney U Test** The Mann-Whitney U Test, is a non-parametric statistical test that is used to compare the **medians** of two **independent** samples.

The Mann-Whitney U Test is based on the rank sums of the two samples. To perform the Mann-Whitney U Test, we first combine the two samples and rank the observations from lowest to highest, assigning the same rank to ties. We then calculate the sum of the ranks for each sample. The test statistic is then calculated as the smaller of the two rank sums, $U$. The test statistic follows a normal distribution with a mean and variance that can be calculated from the sample sizes and the $U$ statistic. Below is an example.

| 秩 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 点数和 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 分数 | 4 | 6 | 9 | 10 | 12 | 17 | 22 | 28 | 40 | 62 | 90 | 95 | 97 | |
| 样本 | A | A | A | B | B | B | B | A | A | B | A | B | B | |
| 样本 A 的点数 | | | | | | | | 4 | 4 | | 5 | | | 13 |
| 样本 B 的点数 | | | | 3 | 3 | 3 | 3 | | | 5 | | 6 | 6 | 29 |

The p-value of the test can be calculated using the normal distribution, or it can be approximated using tables or software. If the p-value is **less** than the chosen significance level, the null hypothesis is rejected, and the conclusion is that there is a significant difference between the two independent samples.

It is worth noting that the Mann-Whitney U Test is different from the Wilcoxon Signed-Rank Test. **The Mann-Whitney U Test is used for independent samples, while the Wilcoxon Signed-Rank Test is used for paired samples.**

Here, we assume that in the example provided in the "weightloss" part, the two data in two programs correspond to two groups of people, and thus independent.

```
wilcox.test(weightloss$program_A,
            weightloss$program_B,
            paired = F)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  weightloss$program_A and weightloss$program_B
## W = 64, p-value = 0.3029
## alternative hypothesis: true location shift is not equal to 0
```

**3.2.2.2 KS Z Test** Compared to KS test before, this is a more "general" version. KS Z test compares two distributions against each other, while KS test, strictly speaking, compares a distribution against a theoretical distribution.

Use the same function `ks.test()` as before, all others the same, so *no* need to cover it again here.

Moreover, it's interesting to compare KS test and Mann-Whitney U test.

- The Mann-Whitney test first ranks all the values from low to high, and then computes a P value that depends on the discrepancy between the mean ranks of the two groups.
- The Kolmogorov-Smirnov test compares the cumulative distribution of the two data sets, and computes a $p$ value that depends on the largest.

Guidelines for choosing between the two tests.

1. The KS test is sensitive to any differences in the two distributions. Substantial differences in shape, spread or median will result in a small P value. In contrast, the **MW test is mostly sensitive to changes in the median**.
2. The MW test is used more often and is recognized by more people, so choose it if you have no idea which to choose.
3. The MW test has been extended to handle tied values. The KS test does not handle ties so well. If your data are categorical, so has many ties, don't choose the KS test.
4. Some fields of science tend to prefer the KS test over the MW test. It makes sense to follow the traditions of your field.

#### 3.2.2.3 Moses Extreme Reaction

- Check the span of the rank of one group when two groups are mixed and ranked (up and low 5% deleted).
- **Good for distributions with mass on both ends**.

  But less powerful and robust as KS test.

In the case of MER, the null hypothesis is that there is no difference in gene expression levels between individuals with extreme reactions and those with

normal reactions. The alternative hypothesis is that there is a significant difference in gene expression levels between the two groups.

The **range** is tested, so the MER has different *sensitivity* for different distributions.

Use `DescTools::MosesTest(x,y,extreme)` to carry the MER. If with a grouping variable, parameters can be passed through a formula-pattern. `extreme` allows you to rule out some extreme values.

In the example provided below, the data record the effect of two kinds of pills. `ycss` is the effect, and `zb` is the grouping label.

```
hypnotic <- haven::read_sav("Data12-06.sav")
```

The MER is especially useful for medical experiments.

```
DescTools::MosesTest(ycss~zb,data = hypnotic)
```

```
##
##  Moses Test of Extreme Reactions
##
## data:  ycss by zb
## S = 15, p-value = 0.6858
## alternative hypothesis: extreme values are more likely in x than in y
```

#### 3.2.2.4 Wald-Wolfowitz Runs

- Put two groups together and rank them; the group label should mix if they are similar.
- **Good for distributions that differ more on variance than on location**.

Wald-Wolfowitz Runs is kind of Runs test, but the two-independent-sample version.

Use the same function as before, `DescTools::RunsTest(x,y)` to carry the test!

```
DescTools::RunsTest(ycss~zb,data = hypnotic)
```

```
##
##  Wald-Wolfowitz Runs Test
##
## data:  ycss by zb
## runs = 6, m = 10, n = 10, p-value = 0.03704
## alternative hypothesis: true number of runs is not equal the expected number
```

## 3.3 K Samples

### 3.3.1 Dependent Samples

The "dependent" here means, a person or an item is tested for $K$ times.

**3.3.1.1 Friedman Test** The Friedman test is a non-parametric statistical test used to determine whether there are **differences** among two or more groups based on **repeated** measures data. It is often used as an alternative to *repeated measures ANOVA* when the data do not meet the assumptions of normality or homogeneity of variance.

The Friedman test involves *ranking* the data for each group and then computing the *sum of the ranks* for each condition or treatment. The test statistic is calculated as the ratio of the sum of squares of the ranks to the total sum of squares, and is compared to the *chi-squared* distribution with degrees of freedom equal to the number of groups minus one. The Friedman test is appropriate when the data are measured on an ordinal or continuous scale, but not when the data are nominal.

If the test statistic is significant, it indicates that there are differences among the groups. However, the Friedman test does not identify which groups are different from each other. *Post-hoc tests*, such as the Wilcoxon signed-rank test or the Dunn's test, can be used to identify the pairwise differences between the groups.

One of the advantages of the Friedman test is that it is robust, meaning that it is not affected by outliers or non-normality in the data. It is also useful for analyzing repeated measures data, where each participant or object is tested multiple times under different conditions or treatments.

The statistic is what we calculated as $\chi^2$ in Kendall's W. No need to cover here!

Note that no need to pass the pre-ranked data into `kendall()`, the raw data is fine!

**3.3.1.2 Cochran's Q** Cochran's Q test is a non-parametric statistical test used to analyze the differences between *three or more* **related** categorical variables. It is similar to the McNemar test, which is used to compare two related categorical variables.

The Cochran's Q test is used when we have a set of subjects measured on a **binary** (dichotomous) variable at three or more time points or under three or more conditions. The test determines if there is a statistically significant difference in the proportion of positive responses across the different time points or conditions.

The null hypothesis of the Cochran's Q test is that the proportion of positive responses is the same across all time points or conditions, while the alternative hypothesis is that at least one time point or condition has a different proportion of positive responses.

If the Cochran's Q test indicates that there are significant differences between the time points or conditions, post-hoc tests such as McNemar's test or Bonferroni correction can be used to determine which time points or conditions are significantly different from each other.

Suppose we have a courses and a group of 30 students enrolled. We measure the pass/fail status of each student at three different time points (midterm, final exam, and overall course grade) to determine if there is a significant difference in the proportion of passing students across the different time

points.

```
midterm <- c(rep("Pass", 5), rep("Fail", 5), rep("Pass", 6), rep("Fail", 4), rep("Pass"
final <- c(rep("Pass", 8), rep("Fail", 2), rep("Pass", 4), rep("Fail", 6), rep("Pass",
overall <- c(rep("Pass", 9), rep("Fail", 1), rep("Pass", 6), rep("Fail", 4), rep("Pass"

df <- data.frame(midterm, final, overall) |>
  gather(key = 'exam',value = 'result') |>
  mutate(stu = rep(rep(1:30),3))
```

In this dataset, we have three categorical variables (midterm, final, and overall) measured at three time points for each student. To perform Cochran's Q test on this dataset, we can use the `cochran_qtest()` function from the `rstatix` package in R:

```
rstatix::cochran_qtest(data = df,formula = result~exam|stu)
```

```
## # A tibble: 1 x 6
##   .y.        n statistic    df     p method
## * <chr>  <int>     <dbl> <dbl> <dbl> <chr>
## 1 result    30      4.57     2 0.102 Cochran's Q test
```

The non-significant result here means that the three exams does not have differences in difficulty.

#### 3.3.1.3 Kendall's W

In essence, it's a Friedman Test, no need to cover here.

### 3.3.2 Independent Samples

#### 3.3.2.1 Kruskal–Wallis Test

The Kruskal-Wallis test is a non-parametric statistical test used to compare *three or more* **independent** samples to determine if they come from the same population or not. It is often used as an alternative to *one-way ANOVA* when the data do not meet the assumptions of normality or homogeneity of variance.

The Kruskal-Wallis test ranks the data within each group and calculates the sum of ranks for each group. It then uses these sums to calculate a test statistic that measures the differences between groups. The null hypothesis of the Kruskal-Wallis test is that the samples come from the same population, while the alternative hypothesis is that at least one sample comes from a different population.

The test is appropriate when the data are measured on an ordinal or continuous scale, but not when the data are nominal. If the Kruskal-Wallis test indicates that there are significant differences between the groups, post-hoc tests such as the Mann-Whitney U test or Dunn's test can be used to determine which groups are significantly different from each other.

To perform the Kruskal-Wallis test in R, we can use the `kruskal.test()` function. Here's an example:

```
group1 <- c(4, 8, 7, 6, 5)
group2 <- c(3, 7, 6, 5, 4)
group3 <- c(2, 6, 5, 4, 3)

kruskal.test(list(group1, group2, group3))
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  list(group1, group2, group3)
## Kruskal-Wallis chi-squared = 3.2051, df = 2, p-value = 0.2014
```

In this example, we have three independent groups with five observations each. We use the `list()` function to group the data together, and then pass the list to the `kruskal.test()` function. The output of this code will provide the Kruskal-Wallis test statistic, degrees of freedom, and p-value.

If the data is consisted of one value column and one group column, then using the formula-pattern would be more convenient!

```r
df = data.frame(
  value = c(group1,group2,group3),
  group = rep(1:3,each = 5)
)

kruskal.test(value~group,data = df)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  value by group
## Kruskal-Wallis chi-squared = 3.2051, df = 2, p-value = 0.2014
```

**3.3.2.2 Jonckheere-Terpstra (Trend) test** The Jonckheere-Terpstra test, also known as the trend test, is a non-parametric statistical test used to determine **if there is a trend in the ordered levels of a variable across multiple groups**. It is often used to analyze data where there is an **ordered** categorical variable and the groups are ordered in a natural way.

The Jonckheere-Terpstra test ranks the data within each group and calculates the sum of ranks for each group. It then uses these sums to calculate a test statistic that measures the trend in the ordered levels of the variable. The null hypothesis of the Jonckheere-Terpstra test is that there is no trend in the ordered levels of the variable across the groups, while the alternative hypothesis is that there is a trend.

The test is appropriate when the data are measured on an ordinal or continuous scale, but not when the data are nominal. It is commonly used in fields such as medicine, psychology, and social sciences to analyze trends in disease severity, performance, or attitudes across different groups.

If the Jonckheere-Terpstra test indicates that there is a trend in the ordered levels of the variable, post-hoc tests such as the Wilcoxon rank-sum test or Dunn's test can be used to determine which groups are significantly different

from each other.

- Only used when the groups are ordered.
- It actually tests about the median between groups.
- it has **more power** than Kruskal-Wallis.

Suppose we want to analyze the trend in intelligence scores across different age groups. We have four age groups (20-30, 31-40, 41-50, and 51-60) and 10 participants in each group. We measure the intelligence scores of each participant and want to determine if there is a significant trend in intelligence scores across the age groups.

Use `DescTools::JonckheereTerpstraTest()` to carry the test. `nperm` is the number of permutations for the reference distribution. The default is NULL in which case the permutation p-value is not computed. It's recommended to set `nperm` to 1000 or higher if permutation p-value is desired.

```
age <- c(rep("20-30", 10), rep("31-40", 10),
         rep("41-50", 10), rep("51-60", 10))
age = as.ordered(age)
intelligence <- c(80, 75, 85, 90, 70, 75, 80, 85, 65, 70,
                  75, 80, 85, 90, 70, 75, 80, 85, 65, 70,
                  70, 75, 80, 85, 65, 70, 75, 80, 85, 90,
                  60, 65, 70, 75, 80, 85, 90, 70, 75, 80)

df <- data.frame(age, intelligence)

DescTools::JonckheereTerpstraTest(intelligence~age,
                                  data = df)
```

```
## Warning in JonckheereTerpstraTest.default(c(80, 75, 85, 90, 70, 75, 80, : Sample siz
##  p-value based on normal approximation. Specify nperm for permutation p-value

##
##  Jonckheere-Terpstra test
##
```

```
## data:  intelligence by age
## JT = 276, p-value = 0.5624
## alternative hypothesis: two.sided
```

The result is slightly calibrated.

```
DescTools::JonckheereTerpstraTest(intelligence~age,
                                  data = df,
                                  nperm = 5000)
```

```
## Warning in JonckheereTerpstraTest.default(nperm = nperm, c(80, 75, 85, 90, : Sample
##  p-value based on normal approximation. Specify nperm for permutation p-value

##
##  Jonckheere-Terpstra test
##
## data:  intelligence by age
## JT = 276, p-value = 0.5636
## alternative hypothesis: two.sided
```

#### 3.3.2.3 Median Test

Very simple.

- Count the # of data $>$ or $<$ median;
- A Chi-square test on binomial distribution.
- Need a large sample; not preferred.

Now that you've ranked your data, why not try other more powerful non-parametric tests?

```
library(agricolae)
library(irr)
Median.test(intelligence,age)
```

## 3.4 Non-Parametric Correlation Analysis

There are several types of correlation coefficients that are commonly used in statistics to measure the strength and direction of the relationship between two variables. The most commonly used correlation coefficients are Pearson correlation, Spearman correlation, point-biserial correlation, and phi correlation.

Spearman correlation, point-biserial correlation, and phi correlation are non-parametric ones.

1. Pearson correlation: Pearson correlation coefficient measures the linear relationship between two continuous variables. It ranges from -1 to 1, where -1 indicates a perfect negative relationship, 0 indicates no relationship, and 1 indicates a perfect positive relationship. Pearson correlation assumes that the data are normally distributed and that there is a linear relationship between the variables.

2. Spearman correlation: Spearman correlation coefficient measures the strength and direction of the monotonic relationship between two variables. It ranges from -1 to 1, where -1 indicates a perfect negative monotonic relationship, 0 indicates no monotonic relationship, and 1 indicates a perfect positive monotonic relationship. Spearman correlation does not assume that the data are normally distributed and can be used for ordinal or continuous variables.

3. Point-biserial correlation: Point-biserial correlation coefficient measures the relationship between a dichotomous variable and a continuous variable. It ranges from -1 to 1, where -1 indicates a perfect negative relationship between the dichotomous variable and the continuous variable, 0 indicates no relationship, and 1 indicates a perfect positive relationship. Point-biserial correlation is used when one of the variables is dichotomous and the other is continuous.

4. Phi correlation: Phi correlation coefficient measures the relationship between two dichotomous variables. It ranges from -1 to 1, where -1

indicates a perfect negative relationship between the two variables, 0 indicates no relationship, and 1 indicates a perfect positive relationship. Phi correlation is used when both variables are dichotomous.

In general, Pearson and Spearman correlations are used for continuous or ordinal variables, while point-biserial and phi correlations are used for dichotomous variables. Pearson correlation is most appropriate when the data are normally distributed and there is a linear relationship between the variables, while Spearman correlation is more appropriate when the data are non-normally distributed or the relationship is not linear. Point-biserial and phi correlations are appropriate when one or both of the variables are dichotomous.

Moreover, the chi-square independence test is used to determine if there is a significant association between two categorical variables, while the phi correlation coefficient is used to measure the strength and direction of the relationship between two dichotomous variables. The two tests are different perspectives to view the same problem, either say correlation or say independence.

All can be dealt with by `cor.test()`.

What I want to address here is the Phi coefficient. Phi coefficient can be obtained by `cor.test()` using the traditional Pearson-way, and also fine with `psych::phi(matrix)`, but the latter won't give you the significance information.

In the example, we want to test the correlation between parents' political stance and their parenting way. We have the following contingency table.

| | | Political Orientation | | |
|---|---|---|---|---|
| | | Liberals | Conservatives | Total |
| Permissive | | 15 (12.5) | 10 (12.5) | 25 |
| Not Permissive | | 5 (7.5) | 10 (7.5) | 15 |
| | | | | |
| Total | | 20 | 20 | 40 |

I'd like to illustrate first with the "traditional" way.

```
phidt = data.frame(
  parenting = c(rep(1,25),rep(0,15)),
  politics = c(rep(1,15),rep(0,10),rep(1,5),rep(0,10))
)
cor.test(phidt$parenting,phidt$politics)
```

```
##
##  Pearson's product-moment correlation
##
## data:  phidt$parenting and phidt$politics
## t = 1.6475, df = 38, p-value = 0.1077
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.05797333  0.52729702
## sample estimates:
##       cor
## 0.2581989
```

The `psych::phi(matrix)` is indeed not recommend! But it's last advantage is that the data is expressed in the easiest way! If just to get the correlation value, probably go with it.

```
phi_dataset = matrix(c(15,10,5,10),nrow=2,ncol=2)
psych::phi(phi_dataset)
```

```
## [1] 0.26
```

However, if we carry a chi-square test, that differs a little in its significance.

```
chisq.test(phi_dataset)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  phi_dataset
## X-squared = 1.7067, df = 1, p-value = 0.1914
```

# 4 ANCOVA

The data `df` contains information about miners' vital capacity as `vitalcp`, an indicator for their health of lungs; their age as `age`, probability related to `vitalcp`; a dummy `time` telling whether or not they have worked as a miner for 10 years, 1 if so, 0 otherwise.

```
miner = readxl::read_excel("data09-06.xlsx", sheet = 1) |>
  rstatix::convert_as_factor(time)
```

## 4.1 Pre-Inspection

### 4.1.1 Equal Variance

```
miner |>
  rstatix::levene_test(vitalcp~time)
```

```
## # A tibble: 1 x 4
##     df1   df2 statistic     p
##   <int> <int>     <dbl> <dbl>
## 1     1    26      2.44 0.130
```

### 4.1.2 Independence of Covariate and Treatment

```
anova(aov(age~time,data = miner))
```

```
## Analysis of Variance Table
##
## Response: age
##           Df  Sum Sq Mean Sq F value  Pr(>F)
## time       1  202.74 202.741  3.6665 0.06658 .
## Residuals 26 1437.69  55.296
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 4.1.3 Homogeniety of Regression Slopes

```
anova(aov(vitalcp~time*age,data = miner))
```

```
## Analysis of Variance Table
##
## Response: vitalcp
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## time       1  0.2045  0.2045  0.3595 0.5543807
## age        1 10.8806 10.8806 19.1264 0.0002044 ***
## time:age   1  0.1023  0.1023  0.1799 0.6752461
## Residuals 24 13.6532  0.5689
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

## 4.2 ANCOVA

### 4.2.1 Fit the Model

It's noteworthy that in R, the more essential the effect variable is, the earlier it should be put.

The common "rank" is as follows.

- covariate
- main effect
- interaction

The order that IVs are plugged matters! Since we have the algorithm of SS related to that.

In the formula, connect the main effect and covariate with sign `+`.

```
cfit = aov(vitalcp~age+time,data = miner)
```

### 4.2.2 Summary the Model

**4.2.2.1 ANCOVA View** As for the fitted ANCOVA model, say `model`, plug in `model` into `summary()` or `anova()` will return the same and necessary results.

Since `summary()` is more generally used, I'd like to recommend `summary()`!

```
# anova(cfit)
summary(cfit)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## age          1 10.543  10.543  19.162 0.000187 ***
## time         1  0.542   0.542   0.985 0.330439
## Residuals   25 13.755   0.550
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

**4.2.2.2 Linear-Model View** View the ANOVA model as a special case of linear model, with covariate and main effect as its independent variables as is used above, and intercept is included.

`car::Anova(model,type = 'III')` can return the SS, miner, $F$ and $p$ for each term, including the intercept. The coefficients, together with the same significance, can be reported through `summary(model)`.

```
car::Anova(cfit,type = '3')
```

```
## Anova Table (Type III tests)
##
## Response: vitalcp
##             Sum Sq Df F value    Pr(>F)
## (Intercept) 37.959  1 68.9886 1.177e-08 ***
## age         10.881  1 19.7751 0.0001563 ***
## time         0.542  1  0.9852 0.3304389
## Residuals   13.755 25
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(lm(vitalcp~age+time,data = miner))
```

```
##
## Call:
## lm(formula = vitalcp ~ age + time, data = miner)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.68343 -0.37405  0.03523  0.44561  1.12263
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.27718    0.99654   8.306 1.18e-08 ***
## age         -0.08700    0.01956  -4.447 0.000156 ***
## time2       -0.30033    0.30258  -0.993 0.330439
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7418 on 25 degrees of freedom
## Multiple R-squared:  0.4463, Adjusted R-squared:  0.402
## F-statistic: 10.07 on 2 and 25 DF,  p-value: 0.0006186
```

### 4.2.3 Effect Size

```
rstatix::partial_eta_squared(cfit)
```

```
##        age       time
## 0.43389844 0.03791233
```

### 4.2.4 Post-Hoc Test

Library the package `multcomp` and use the `glht(model, linfct = mcp(time = 'Tukey'))` within to get the post-hoc model. Then, use `summary()` to get its result.

```
library(multcomp)
postHoc = glht(cfit,linfct = mcp(time = 'Tukey'))
summary(postHoc)
```

Well, the significant intercept term means that the grand mean does account for some of the variation.

### 4.2.5 Visualize the Model

Library package `HH`, and then use the function `ancova()` to fit the model. Note that all parameters passed into `ancova()` is the same as those into `aov()`.

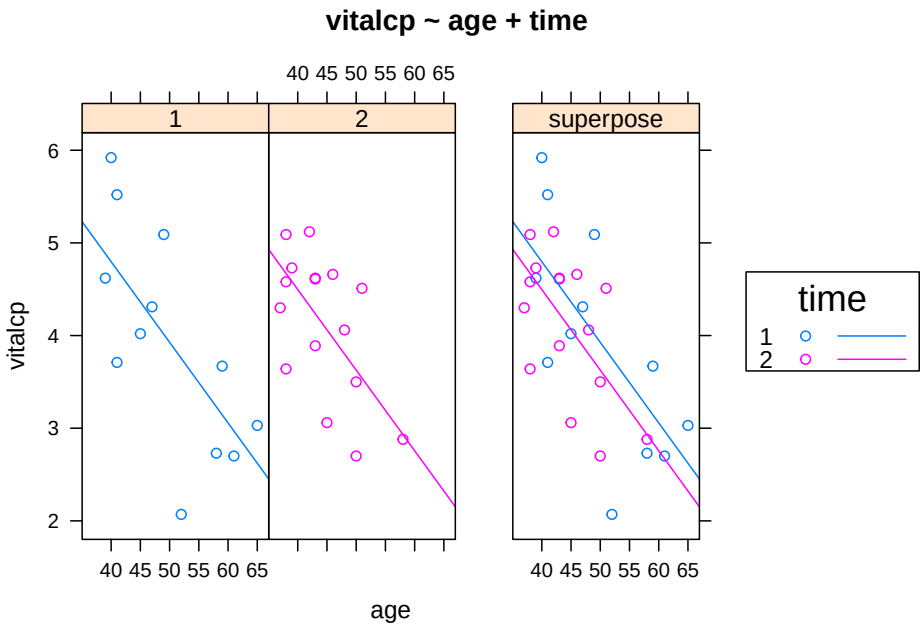
Mind the order of independent variables, since that does matter!

`ancova(...)` will not only return the completely same result as we've seen in `summary(aov(...))`, but also a graph illustrating the comprehensive correlation between the dependent variable and covariate & main effect.

Note that within `ancova()`, more underlying functions from that package `HH` are loaded, which means you can not get away with just specifying the package like `HH::ancova`, but have to library the package specifically.

```
library(HH)
ancova(vitalcp~age+time , data = miner)
```

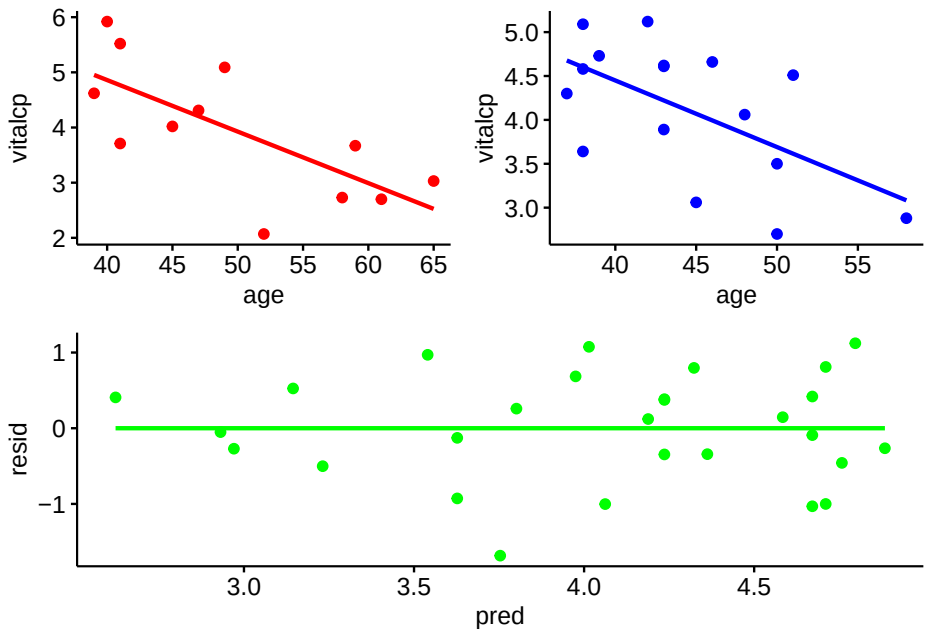
```
## Analysis of Variance Table
##
## Response: vitalcp
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## age        1 10.5431 10.5431 19.1617 0.0001871 ***
## time       1  0.5421  0.5421  0.9852 0.3304389
## Residuals 25 13.7555  0.5502
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



### 4.2.6 Visual Check

```
library(ggpubr)
library(patchwork)
p1 = ggscatter(miner[miner$time==1,],
          x="age",y="vitalcp",
          color="red",add="reg.line")
p2 = ggscatter(miner[miner$time==2,],
          x="age",y="vitalcp",
          color="blue",add="reg.line")
p3 = ggscatter(data = data.frame(
  pred = fitted(cfit),
  resid = residuals(cfit)
),x="pred",y="resid",color="green",add="reg.line")

(p1+p2)/p3
```



### 4.3 Report the Data

ANCOVA was conducted on `vitalcp` with `time` as the main effect and `age` as the covariate.

The covariate, age of the workers, was significantly related to the vital capacity, $F(1, 25) = 19.775, p < 0.001, r = 0.665, \eta^2 p = 0.43389844$. There was no significant effect of time on the vital capacity, after controlling the effect of age, $F(1, 25) = 0.965, p = 0.33, \eta^2 p = 0.03791233$.

## 5 Latin Square

The following data `yie` has the information about the yields of vegetables, say `yield`, and the corresponding conditions of growing, such as the variety of vegetables, say `variety` here; and the location information for `rep` and `col`. In the three treatments, varieties of vegetables are the primary one that we're interested in, and the other two are secondary variables to be controlled.

```
knitr::include_graphics('Latin Square.png')
```

```
yie = readxl::read_xlsx("veget.xlsx") |>
  rstatix::convert_as_factor(rep,col,variety)
```

### 5.1 Fit the Model

The model is fitted the same as before for ANOVA. Note that Latin Square is a special design, where each letter (level) appears only once in each row and each column.

Here, though the factors differ in importance, the order that they're plugged in does not matter!

```
latinfit = aov(yield~rep+col+variety,
               data = yie)
```

## 5.2 Summary the Model

### 5.2.1 ANOVA View

Basically, `summary(model)` and `anova(model)` will return the same result.

```
# summary(latinfit)
anova(latinfit)
```

```
## Analysis of Variance Table
##
## Response: yield
##           Df Sum Sq Mean Sq F value  Pr(>F)
## rep        5  4.460  0.8919  0.6461 0.66553
## col        5  1.695  0.3389  0.2455 0.94034
## variety    5 21.563  4.3126  3.1242 0.01477 *
## Residuals 56 77.302  1.3804
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 5.2.2 Linear-Model View

ANOVA (even Latin Square design) is a special case of linear model. Similarly, use `car::Anova(model,type = 'III')` to see the intercept term, with other terms same with the results from the functions mentioned above.

However, if one grouping variable has multiple levels, the linear model may seems not so simple, because each level will sit at a term. In this case, traditional view of ANOVA seems "better".

```r
car::Anova(latinfit, type = 'III')
```

```
## Anova Table (Type III tests)
##
## Response: yield
##              Sum Sq Df  F value  Pr(>F)
## (Intercept) 1342.92  1 972.8543 < 2e-16 ***
## rep            4.46  5   0.6461 0.66553
## col            1.69  5   0.2455 0.94034
## variety       21.56  5   3.1242 0.01477 *
## Residuals     77.30 56
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
summary(lm(yield~rep+col+variety,
           data = yie))
```

```
##
## Call:
## lm(formula = yield ~ rep + col + variety, data = yie)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.4000 -0.9833 -0.0250  0.9167  2.2000
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.728e+01  5.539e-01  31.191  < 2e-16 ***
## rep2        -1.917e-01  4.797e-01  -0.400  0.69097
## rep3         1.667e-01  4.797e-01   0.347  0.72954
## rep4         8.333e-02  4.797e-01   0.174  0.86270
## rep5        -3.333e-01  4.797e-01  -0.695  0.48996
## rep6        -5.500e-01  4.797e-01  -1.147  0.25639
## col2         1.667e-01  4.797e-01   0.347  0.72954
```

```
## col3         1.583e-01  4.797e-01   0.330  0.74255
## col4         4.583e-01  4.797e-01   0.956  0.34340
## col5         3.917e-01  4.797e-01   0.817  0.41764
## col6         2.000e-01  4.797e-01   0.417  0.67829
## variety2     4.500e-01  4.797e-01   0.938  0.35218
## variety3     1.083e-01  4.797e-01   0.226  0.82213
## variety4    -7.401e-17  4.797e-01   0.000  1.00000
## variety5     1.517e+00  4.797e-01   3.162  0.00253 **
## variety6    -2.266e-17  4.797e-01   0.000  1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.175 on 56 degrees of freedom
## Multiple R-squared:  0.2639, Adjusted R-squared:  0.06676
## F-statistic: 1.339 on 15 and 56 DF,  p-value: 0.2112
```

## 5.3 Post-Hoc Test

```
TukeyHSD(latinfit)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = yield ~ rep + col + variety, data = yie)
##
## $rep
##            diff       lwr       upr     p adj
## 2-1 -0.19166667 -1.606952 1.2236185 0.9986162
## 3-1  0.16666667 -1.248619 1.5819518 0.9992958
## 4-1  0.08333333 -1.331952 1.4986185 0.9999768
## 5-1 -0.33333333 -1.748619 1.0819518 0.9817860
## 6-1 -0.55000000 -1.965285 0.8652852 0.8595512
## 3-2  0.35833333 -1.056952 1.7736185 0.9749576
```

```
## 4-2  0.27500000 -1.140285 1.6902852 0.9923856
## 5-2 -0.14166667 -1.556952 1.2736185 0.9996812
## 6-2 -0.35833333 -1.773619 1.0569518 0.9749576
## 4-3 -0.08333333 -1.498619 1.3319518 0.9999768
## 5-3 -0.50000000 -1.915285 0.9152852 0.9013962
## 6-3 -0.71666667 -2.131952 0.6986185 0.6693358
## 5-4 -0.41666667 -1.831952 0.9986185 0.9523845
## 6-4 -0.63333333 -2.048619 0.7819518 0.7725796
## 6-5 -0.21666667 -1.631952 1.1986185 0.9975120
##
## $col
##             diff        lwr      upr     p adj
## 2-1  0.166666667 -1.2486185 1.581952 0.9992958
## 3-1  0.158333333 -1.2569518 1.573619 0.9994513
## 4-1  0.458333333 -0.9569518 1.873619 0.9298025
## 5-1  0.391666667 -1.0236185 1.806952 0.9632783
## 6-1  0.200000000 -1.2152852 1.615285 0.9983024
## 3-2 -0.008333333 -1.4236185 1.406952 1.0000000
## 4-2  0.291666667 -1.1236185 1.706952 0.9900234
## 5-2  0.225000000 -1.1902852 1.640285 0.9970236
## 6-2  0.033333333 -1.3819518 1.448619 0.9999998
## 4-3  0.300000000 -1.1152852 1.715285 0.9886570
## 5-3  0.233333333 -1.1819518 1.648619 0.9964650
## 6-3  0.041666667 -1.3736185 1.456952 0.9999993
## 5-4 -0.066666667 -1.4819518 1.348619 0.9999923
## 6-4 -0.258333333 -1.6736185 1.156952 0.9943019
## 6-5 -0.191666667 -1.6069518 1.223619 0.9986162
##
## $variety
##           diff          lwr        upr     p adj
## 2-1  0.4500000 -0.965285169  1.8652852 0.9347773
## 3-1  0.1083333 -1.306951836  1.5236185 0.9999148
## 4-1  0.0000000 -1.415285169  1.4152852 1.0000000
```

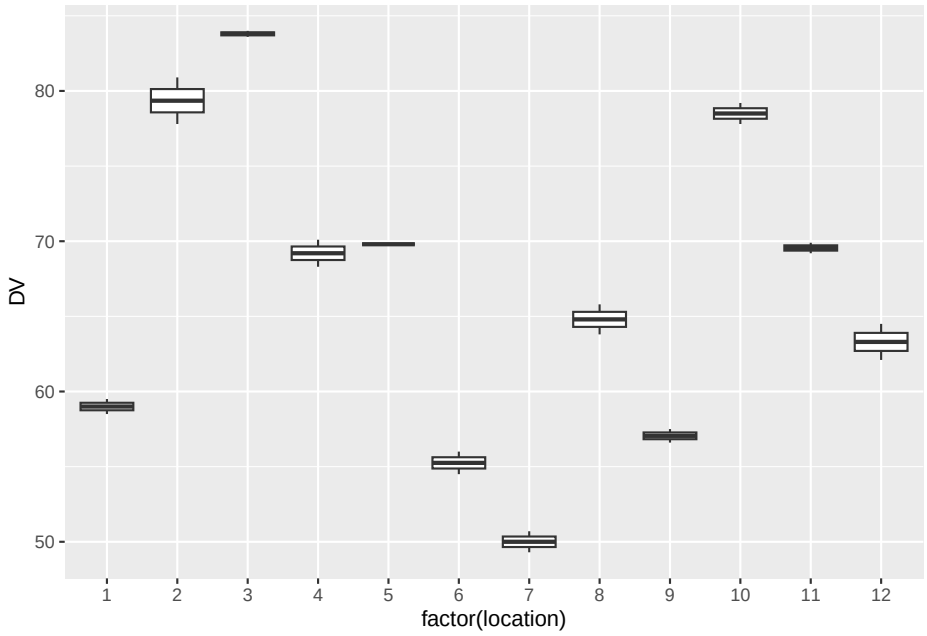
```
## 5-1  1.5166667  0.101381497  2.9319518 0.0289344
## 6-1  0.0000000 -1.415285169  1.4152852 1.0000000
## 3-2 -0.3416667 -1.756951836  1.0736185 0.9796809
## 4-2 -0.4500000 -1.865285169  0.9652852 0.9347773
## 5-2  1.0666667 -0.348618503  2.4819518 0.2434519
## 6-2 -0.4500000 -1.865285169  0.9652852 0.9347773
## 4-3 -0.1083333 -1.523618503  1.3069518 0.9999148
## 5-3  1.4083333 -0.006951836  2.8236185 0.0518445
## 6-3 -0.1083333 -1.523618503  1.3069518 0.9999148
## 5-4  1.5166667  0.101381497  2.9319518 0.0289344
## 6-4  0.0000000 -1.415285169  1.4152852 1.0000000
## 6-5 -1.5166667 -2.931951836 -0.1013815 0.0289344
```

# 6 Nested ANOVA

```
nestedData = readxl::read_excel("nested anova.xlsx")
```

## 6.1 visualization

```
# we should eyeball the data first to make sense of the test.
# create boxplots to visualize the data
ggplot(nestedData, aes(x=factor(location), y=DV)) +
  geom_boxplot()
```




## 6.2 nested ANOVA

Theoretically, the order of computing the variance sum of squares matters, thus appointing the `summation = 'I'`.

However, here the summation type does not matter.

Note that the result of the main effect of the group is wrong. We can manually make it straight from the SS of the group and the subgroup.

```
# note that nested anova can use error type I and III.
# aov() has a default type I to process factors in an sequential order
# here type I and III don't differ, since effectively we have no orders
aov_model = aov(DV ~ factor(method)/factor(location),
                data = nestedData,summation="I")
```

```
## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'summation' will be disregarded
```

```
summary(aov_model)
```

```
##                                 Df Sum Sq Mean Sq F value   Pr(>F)
## factor(method)                   2  665.7   332.8   255.7 1.45e-10 ***
## factor(method):factor(location)  9 1720.7   191.2   146.9 6.98e-11 ***
## Residuals                       12   15.6     1.3
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 6.2.1 error adjustment

An alternative refinement is to adjust the error term.

```
aov.result2 = aov(DV ~ factor(method) +
                    Error(factor(method):factor(location)),
                  data = nestedData)
```

```
## Warning in aov(DV ~ factor(method) + Error(factor(method):factor(location)), :
## Error() model is singular
```

```
summary(aov.result2)
```

```
##
## Error: factor(method):factor(location)
##                Df Sum Sq Mean Sq F value Pr(>F)
## factor(method)  2  665.7   332.8   1.741   0.23
## Residuals       9 1720.7   191.2
##
## Error: Within
##           Df Sum Sq Mean Sq F value Pr(>F)
## Residuals 12  15.62   1.302
```

### 6.2.2 comparison with one-way

If we wrongly ignore the nested factor and use one-way ANOVA, the results would be different

Note that there's only one group factor, therefore, the type of summation does not matter.

```
aov_model = aov(DV ~ factor(method),
                data = nestedData,summation="III")
```

```
## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'summation' will be disregarded
```

```
summary(aov_model)
```

```
##                Df Sum Sq Mean Sq F value Pr(>F)
## factor(method)  2  665.7   332.8   4.026 0.0331 *
## Residuals      21 1736.3    82.7
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
aov_model = aov(DV ~ factor(method),
                data = nestedData)
summary(aov_model)
```

```
##                Df Sum Sq Mean Sq F value Pr(>F)
## factor(method)  2  665.7   332.8   4.026 0.0331 *
## Residuals      21 1736.3    82.7
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# 7 MANOVA

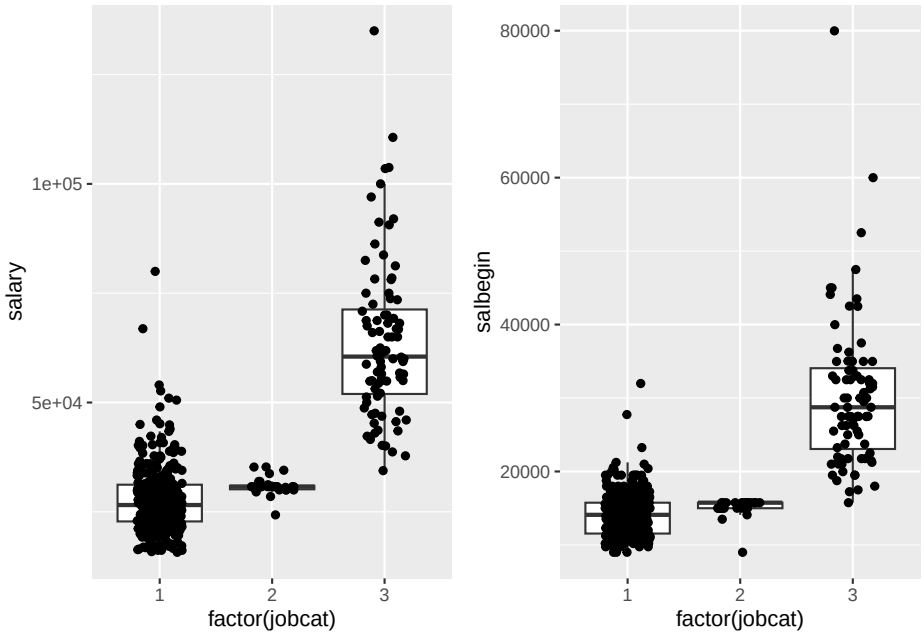
```
# salary data example
salaryData = readxl::read_xlsx("example.xlsx")
DV = cbind(salaryData$salary, salaryData$salbegin)
```

## 7.1 Visualization

Note that in order to generate the boxplot according to the "categorical" data, "factor" type should be designated in the mapping parameter.

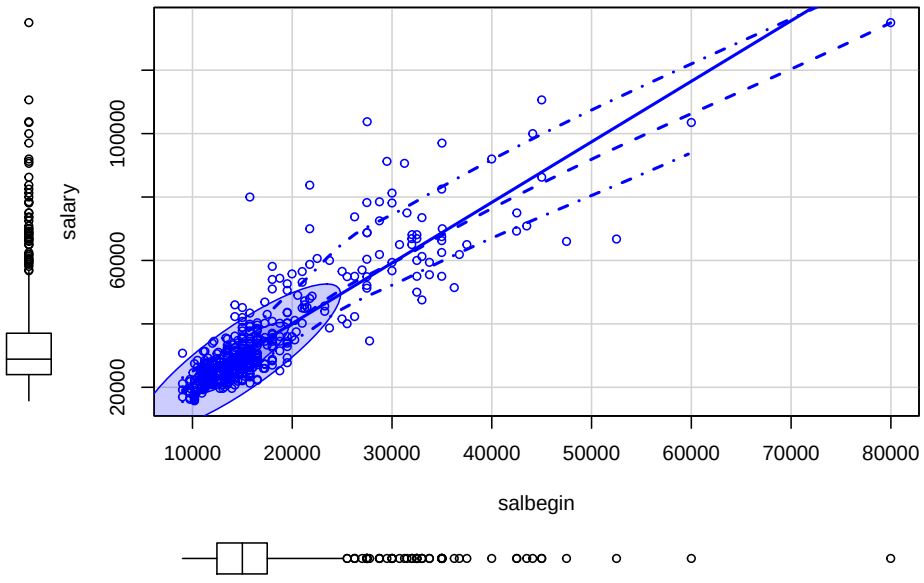
If not "factored", the scatter plot works fine with grouped effect, but the boxplot fails.

```
# eyeball the data
library(gridExtra)
p1 = ggplot(salaryData, aes(x = factor(jobcat), y = salary)) +
  geom_boxplot(outlier.shape = NA) + geom_jitter(width = 0.2) +
  theme(legend.position="top")
p2 = ggplot(salaryData, aes(x = factor(jobcat), y = salbegin)) +
  geom_boxplot(outlier.shape = NA) + geom_jitter(width = 0.2) +
  theme(legend.position="top")
grid.arrange(p1, p2, ncol=2)
```




We can also eyeball the DVs.

```
car::scatterplot(salary ~ salbegin, data=salaryData, ellipse=TRUE,
  smooth=list(style="lines"))
```



## 7.2 Asumptions

### 7.2.1 Normality

**7.2.1.1 pipe-friendly (Preferred)** Note that `mshapiro_test()` is from the package `rstatix`, one step further from `tidyverse`.

```
salaryData |> dplyr::select(salary,salbegin) |>
  rstatix::mshapiro_test()
```

```
## # A tibble: 1 x 2
##   statistic  p.value
##       <dbl>    <dbl>
## 1     0.707 6.13e-28
```

```
salaryData |> group_by(jobcat) |>
  shapiro_test(salary,salbegin)
```

```
## # A tibble: 6 x 4
##   jobcat variable statistic        p
##    <dbl> <chr>        <dbl>    <dbl>
## 1      1 salary       0.882 4.57e-16
## 2      1 salbegin     0.931 6.10e-12
## 3      2 salary       0.818 2.86e- 4
## 4      2 salbegin     0.499 1.64e- 8
## 5      3 salary       0.929 1.72e- 4
## 6      3 salbegin     0.860 2.14e- 7
```

**7.2.1.2 classical (Not Preferred)** Not preferred. It's not efficient to do this way.

The specific function `mshapiro.test()` requires to library the package `mvnormtest`, which surprisingly has a single function! Moreover, it requires the data to be transposed before being processed.

The most annoying part is that the data has to be packed purposefully.

```
mvnormtest::mshapiro.test(t(DV))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Z
## W = 0.70689, p-value < 2.2e-16
```

### 7.2.2 Homogeniety of Variance & Covariance

Using the Box's M-test and Levene's Test.

**7.2.2.1 pipe-friendly (Preferred)** Using `box_m()` from the package `rstatix` to test the homogeniety of covariance matrices, in which the first parameter is the DVs needed to be analyzed (No group variable! This can be achieved through `select()`), and the second is the `group` parameter specifying the group variable.

Moreover, the function `heplots::boxM()` is helpful. Instead of taking the formula to specify the parameters, using the "traditional" way to write `group = ...` works pretty well combined with pipe-friendly environment!

The drawback of the functions is that they **cannot search column variables automatically**, and that should be specified **manually**. And more annoying one is that, when encountering with NAs, an error will be raised by `rstatix::box_m()`, while that's fine with `heplots::boxM()`.

```
salaryData |> dplyr::select(salary,salbegin) |>
  rstatix::box_m(group = salaryData$jobcat)
```

```
## # A tibble: 1 x 4
##   statistic   p.value parameter method
##       <dbl>     <dbl>     <dbl> <chr>
## 1      509. 1.14e-106         6 Box's M-test for Homogeneity of Covariance Matr~
```

```
salaryData |> dplyr::select(salary,salbegin) |>
  heplots::boxM(group = salaryData$jobcat)
```

```
##
##  Box's M-test for Homogeneity of Covariance Matrices
##
## data:  dplyr::select(salaryData, salary, salbegin)
## Chi-Sq (approx.) = 508.68, df = 6, p-value < 2.2e-16
```

Do not forget that the assumption of homogeniety of variance for each DV is needed either! However, the traditional `levene_test()` (or its variants) can deal with one DV at a time, which is tedious when we have many DVs to check.

Hopefully, we have `heplots::leveneTests(data,group)` to deal with that issue. Combined with pipe-friendly procedures, that's fabulous!

One noteworthy flaw is that the `heplots::leveneTests(data,group)` cannot search column variables like that in the "tidyverse" environment.

```
salaryData |>  dplyr::select(salbegin,salary) |>
  heplots::leveneTests(group = salaryData$jobcat)
```

```
## Warning in leveneTest.default(x, group = group, center = center, ...): group
## coerced to factor.

## Warning in leveneTest.default(x, group = group, center = center, ...): group
## coerced to factor.

## Levene's Tests for Homogeneity of Variance (center = median)
##
##          df1 df2 F value    Pr(>F)
## salbegin   2 471  68.862 < 2.2e-16 ***
## salary     2 471  51.189 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

**7.2.2.2 classical** The most annoying part is that the data has to be packed purposefully.

An alternative way to use `heplots::boxM()` has been mentioned above. We can renovate the efficiency to carry this test by replacing the "formula" specifying-pattern with the "group" one.

```
heplots::boxM(cbind(salary, salbegin) ~ factor(jobcat),
              data = salaryData)
```

```
##
##  Box's M-test for Homogeneity of Covariance Matrices
##
## data:  Y
## Chi-Sq (approx.) = 508.68, df = 6, p-value < 2.2e-16
```

## 7.3 MANOVA

Use the function `manova()`, which is from the baseR, no special need to library packages.

Similarly, the most annoying part is that the **DVs and IVs has to be packed** previously.

```
jobcat = salaryData$jobcat
fit = manova(DV ~ jobcat)
```

As before, simply use `summary()` of the fitted model to see the results.

```
summary(fit)  #for MANOVA
```

```
##            Df  Pillai approx F num Df den Df    Pr(>F)
## jobcat      1 0.62914    399.5      2    471 < 2.2e-16 ***
## Residuals 472
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 7.3.1 Effect Size

Checking the effect size is also indispensable.

Use `eta_squared()` from the package `effectsize`, which contains plenty of useful functions computing effect size under various scenarios.

Note that there's also an `eta_squared()` from the package `rstatix()`, but that can only deal with univariate ANOVA, it would fail when encountering MANOVA models.

Also note that the package `effectsize` is incorporated in `bruceR`!

```
effectsize::eta_squared(fit)
```

```
## # Effect Size for ANOVA (Type I)
##
## Parameter | Eta2 (partial) |       95% CI
## -----------------------------------------
## jobcat    |           0.63 | [0.59, 1.00]
##
## - One-sided CIs: upper bound fixed at [1.00].
```

### 7.3.2 Univariate ANOVA

Needed if MANOVA reports significant results.

`summary.aov()` will report the univariate ANOVA respectively.

```
summary.aov(fit)  #for indDVidual anovas
```

```
##  Response 1 :
##              Df     Sum Sq    Mean Sq F value    Pr(>F)
## jobcat        1 8.3933e+10 8.3933e+10  733.86 < 2.2e-16 ***
## Residuals   472 5.3983e+10 1.1437e+08
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
##  Response 2 :
##              Df     Sum Sq    Mean Sq F value    Pr(>F)
## jobcat        1 1.6687e+10 1.6687e+10  624.44 < 2.2e-16 ***
## Residuals   472 1.2614e+10 2.6724e+07
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 7.3.3 Post-Hoc Tests

Note that the post-hoc tests here are essentially for the univariate ANOVA.

Note that the `games_howell_test()` is different from `tukey_hsd()`.

Note that for many DVs to take the post-hoc tests simultaneously, first `gather()` the targeted columns, and then take the **grouped** data for the test.

```
salaryData %>% gather(key = "var",
                      value = "value",
                      salary, salbegin) |>
  group_by(var) |>
  games_howell_test(value~jobcat)
```

```
## # A tibble: 6 x 9
##   var      .y.   group1 group2 estimate conf.low conf.high    p.adj p.adj.signif
## * <chr>    <chr> <chr>  <chr>     <dbl>    <dbl>     <dbl>    <dbl> <chr>
## 1 salary   value 1      2         3100.    1746.     4455. 1.22e- 6 ****
## 2 salary   value 1      3        36139.   31302.    40977. 4.25e-10 ****
## 3 salary   value 2      3        33039.   28196.    37881. 4.21e-10 ****
## 4 salbegin value 1      2          982.     256.     1707. 6   e- 3 **
## 5 salbegin value 1      3        16162.   13539.    18784. 2.74e-10 ****
## 6 salbegin value 2      3        15180.   12514.    17846. 4.77e-10 ****
```

# 8 Logistic Regression

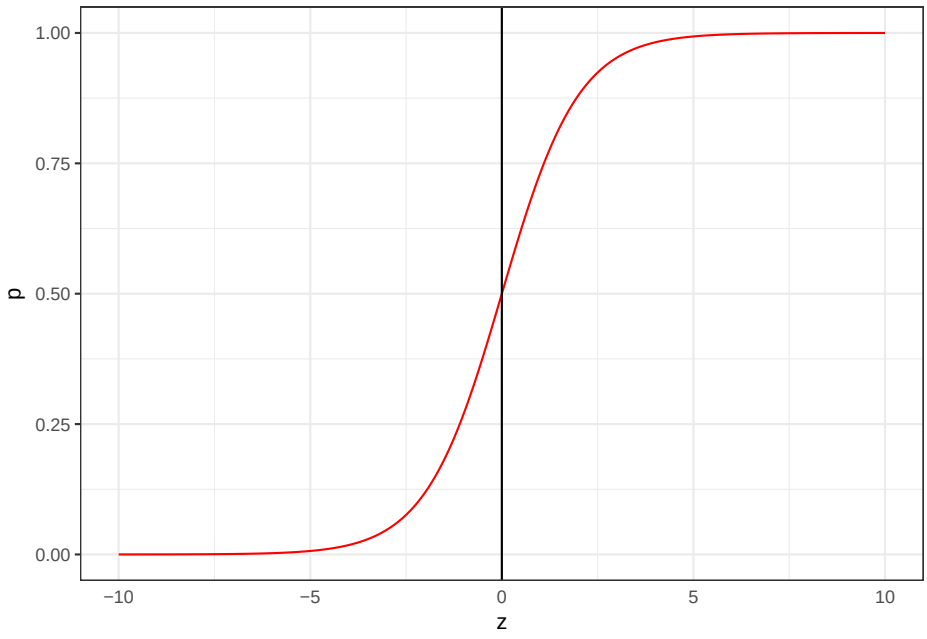
Logistic Regression is for fitting a regression **curve** to data where the DV is **dichotomous**. Meanwhile, the IV(s) can take on any form, such as binary, categorical and continuous.

The goal of logistic regression is to model the probability of a certain event (or say, a specific input). However, probability $p$ is not used directly here. Use **Odds** instead.

$$Odds = \frac{p}{1-p}$$

The logistic function used as such a predictor is as follows.

$$p(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z}$$




The logistic function performs well that it's the C.D.F of a normal distribution! Or take derivative of the logistic function, beautiful and unique traits

of the normal distribution can be well applied.

In the logistic function, $z = \alpha + \beta x$, which can also be extended to multiple IVs, i.e. $z = \beta_0 + \beta_1 + ... + \beta_n x$.

Obviously, if the coefficient of IV is positive, then the probability will increase with IV. If not significantly different from zero, then probability won't significantly be affected by it.

Take a step further, the coefficient $\beta$ is interpreted through the term "**Odds Ratio**", where

$$Odds\ Ratio = \frac{adds(x_i + 1)}{odds(x_i)} = e^{\beta_i}$$

The odds ratio is the effect size of corresponding IV $x_i$, which means one unit increase in x would induce the odds to change to $e^{\beta_i}$ times of itself. The direction given by it coincides with previous discussion of the sign of coefficients and direction of odds' change. Then, if $\beta_i$ is statistically differently from 0 (through $z$ test or Wald test, equivalent), then the odds ratio is significantly different from 1, and 1 doesn't fall in its CI.

Furthermore, given an observation, the CI of probability can be obtained.

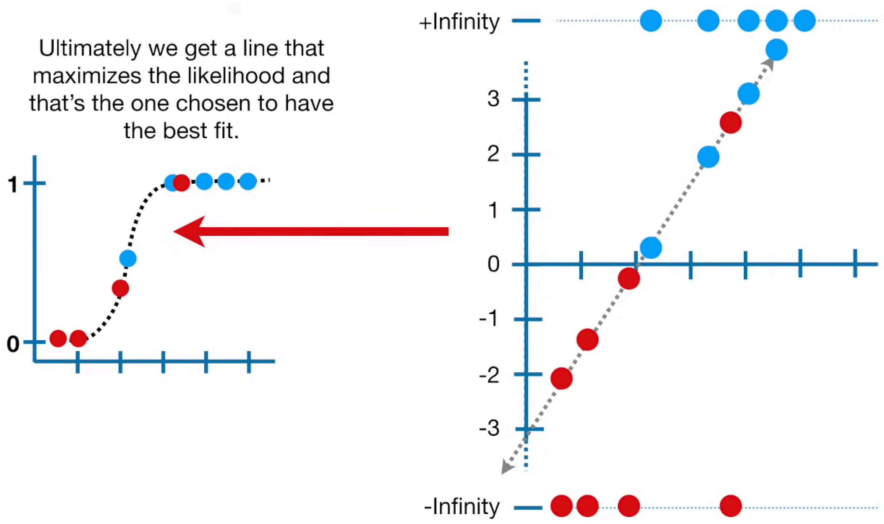
All above seems strange at first glance. However, the focus on odds isn't odd. From the logistic function, doing some transformation, we can get the logit function.

$$Logit(p) = \ln(\frac{p}{1-p}) = \alpha + \beta x$$

The logit function is linear and the response variable has much to do with odds!

However, the linear regression line here isn't obtained "traditionally" as we did. Maximum Likelihood Estimation (MLE) is used, instead of Ordinary Least Squares (OLS).

The following picture can help understand why MLE.




For real data, the observed outcome can be either 0 or 1. However, that won't make any sense considering linear transforming that into $Logit(p) = \ln(\frac{p}{1-p})$, either $-\infty$ or $+\infty$. Thus, OLS doesn't work here.

Alternatively, MLE is used here to estimate a "good" fit. Given an arbitrary regression line, each point is projected onto the line according to its $x$ value(s). Then the goodness of such fit is rated by its "likelihood" with the reality, which means better model should explain and predict the data better, thus depicting the world better.

The likelihood used here is as follows.

$$\mathrm{L}(data) = \Pi_{i=1}^{n}[p(x_i)^{y_i}(1-p(x_i))^{1-y_i}]$$
$$\mathrm{LL}(data) = \sum_{i=1}^{n}[y_i \times p(x_i) + 1 - y_i \times (1-p(x_i))]$$

Starting from a seemingly "arbitrary" line (the initial parameter should be specified or guessed by human, and that matters when it comes to efficiency.), the algorithm is smart enough that it'll change the parameters repeatedly until for a direction with higher likelihood, and ultimately find the maximized one and the corresponding parameters as the solution.

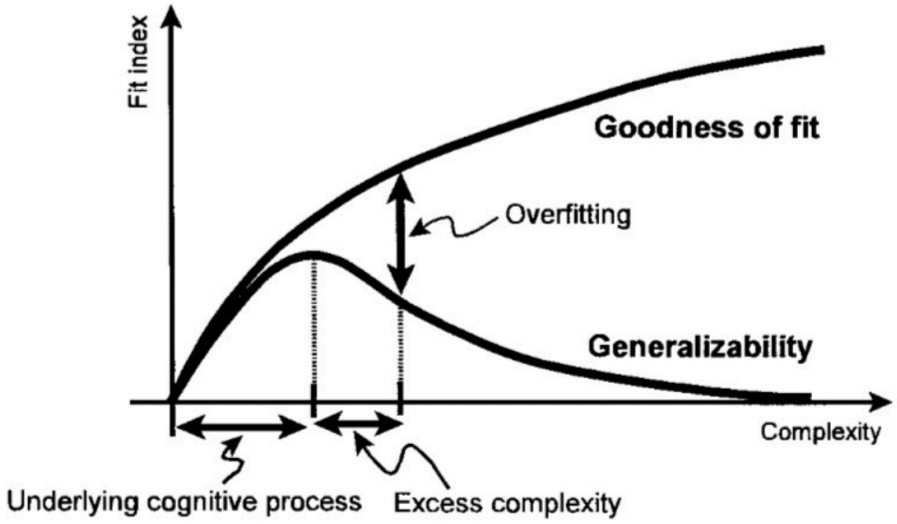
Finally, the fitted result can be evaluated through four aspects.

| Predicted \ Actual | 1 | 0 |
|---|---|---|
| 1 | True positive | False positive |
| 0 | False negative | True negative |

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$
$$Precision = \frac{TP}{TP + FP}$$
$$Sensitivity = \frac{TP}{TP + FN}$$
$$Specificity = \frac{TN}{TN + FP}$$

- Accuracy is the rate that the model can predict true results, **either positive or negative**.
- Precision is the rate that the model can predict true positive results, **within reported positive ones**.
- Sensitivity is the rate that the model can predict true positive results, **within actual positive ones**.
- Specificity is the rate that the model can predict the true **negative** results, **within actual negative ones**.

How many parameters the mdoel should have in order to have an effective and efficient prediction? More parameters added into a certain model will increase accuracy for the present data (sometimes even won't), however, such complicated model may lose generalizability and lose its ability to predict. Well, on the other hand, simple model may not have high ability to explain or predict the data. We should find a compromise.




Generally speaking, for models evaluated by MLE, the comparison of MLEs under different hypothesis can be applied to model comparison.

$$
\begin{aligned}
H_0 &: \theta = \theta_0 \\
H_1 &: \theta = \theta_1 \quad If\ \Lambda(y) > c,\ reject\ H_0 \\
\Lambda(y) &= \frac{L(\theta_1|y)}{L(\theta_2|y)}
\end{aligned}
$$

Based on above, here to introduce the nested hypothesis test. Apparently, the nested model is simpler, we're to check necessity of the addition of other IV(s).

Suppose Model 0 ($M_0$) is nested in Model 1 ($M_1$). $M_0$ has $m$ parameters, and $M_1$ has $n$. Define

$$\Delta(y) = 2\ln(L(\widehat{\theta_1}|y) - 2\ln(L(\widehat{\theta_0}|y))$$

Here factor "2" is to scale the statistic to make it that

$$\Delta(y) \sim \chi^2_{n-m}$$

Hence, if $\Delta(y)$ exceeds the critical value, reject $M_0$ and recognize that $M_1$ is simpler but not worse (i.e. general and powerful).

Often, a model with intercept and predictors (B) is compared to an intercept-only model to test whether the predictors contribute something over and above the intercept-only.

Note that considering sample size when evaluating goodness of fit, $\chi^2$ statistics are heavily influenced by sample size so that with a very large sample, even a tiny difference will be significant. If the sample size is large and the $\chi^2$ is significant, this may not be important.

Coming back, there're three ways for model fitting.

- Enter: all theoretically-based variables are added at once.
- Forward Selection: one parameter at a time, significant ones kept. Used for exploratory purposes.
- Backward Elimination: full model first, eliminate one by one. Used for finding a concise model.

All three way should be based on the significance of $\chi^2$ test.

Meanwhile, there're more concepts or evaluations for model fitting, indicating goodness of fit.

- Nagelkerke $r^2$: between 0 and 1, similar to $r^2$, preferred.
- Hosmer and Lemeshow (HL) test: compare the observed frequency and the predicted frequency, an alternative to model $\chi^2$ test. If NOT significant, good!

## 8.1 Fit the Model

Using the data `logisticData`, we're to predict whether or not to have cancer through logistic regression model, with `age`, `pathsize` and `pathscat` (3 types) as predictors.

```
logisticData <- readxl::read_xlsx("Data11-02.xlsx")
```

Observations with `pathscat == 99` are invalid and should be eliminated.

```
logisticData = logisticData |> filter(pathscat != 99)
```

Then, set dummy variables.

```
logisticData = data.frame(model.matrix(
    ~factor(pathscat),logisticData)) |>
  dplyr::select(2:3) |>
  rename(
    pathscat2 = factor.pathscat.2,
    pathscat3 = factor.pathscat.3
  ) |>
  cbind(logisticData)
```

Setting dummies is quite important when having categorical data, and for more details, see appendix.

Use `glm(formula, data, family = binomial('logit'))` to get the logistic model. Here `glm()` stands for "Generalized Linear Model".

As for the model, others are the same with what we've learned.

```
fit = glm(ln_yesno ~ age + pathsize + pathscat2 + pathscat3,
          data = logisticData,
          family = binomial('logit'))
```

Note that `'logit'` is the default for `binomial('logit')`).

Not that in `lm()`, categorical variables can be thrown in directly, without setting dummies tediously. But the categorical variable should be "factor" or "character", not "numeric".

```
summary(fit)
```

```
##
```

```
## Call:
## glm(formula = ln_yesno ~ age + pathsize + pathscat2 + pathscat3,
##     family = binomial("logit"), data = logisticData)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9676  -0.7400  -0.6134  -0.4497   2.1834
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.583306   0.391314  -1.491   0.1361
## age         -0.024646   0.005764  -4.276  1.9e-05 ***
## pathsize     0.423857   0.130885   3.238   0.0012 **
## pathscat2   -0.122002   0.266498  -0.458   0.6471
## pathscat3    0.184969   0.845896   0.219   0.8269
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1216.7  on 1120  degrees of freedom
## Residual deviance: 1151.8  on 1116  degrees of freedom
## AIC: 1161.8
##
## Number of Fisher Scoring iterations: 4
```

The logistic regression model is different from the traditional OLS model; there's no F value and multiple R squared. Instead, deviance of the designated model and the null model are reported at bottom, no test represented here. In fact, the **deviance** is 2 times the **Log-Likelihood** taking the absolute value. and the larger the AIC, the prettier the model comparison.

The effect size for each $x_i$ is $e^{\beta_i}$, which can be obtained through a matrix. Interestingly, the `exp()` can be exerted to a data frame and all values within.

```
exp(cbind(OR = coef(fit), confint(fit)))
```

```
## Waiting for profiling to be done...
```

```
##                    OR     2.5 %    97.5 %
## (Intercept) 0.5580501 0.2582678 1.1992024
## age         0.9756550 0.9645972 0.9866601
## pathsize    1.5278427 1.1832781 1.9780602
## pathscat2   0.8851466 0.5234879 1.4893971
## pathscat3   1.2031815 0.2348608 6.7095766
```

## 8.2 Goodness of Fit

### 8.2.1 Each Predictor

**8.2.1.1 Separately** Use `aod::wald.test(Sigma, b, Terms)` to carry the Wald test for each coefficient.

- `Sigma`: a var-cov matrix of the model, extracted from `vcov(model)`.
- `b`: a vector of coefficients with var-cov matrix `Sigma`, extracted from `coef(model)`.
- `Terms`: an optional integer vector specifying which coefficients should be jointly tested, like choosing the variable from the var-cov matrix.

The result of significance of the Wald test is the same as is reported in `summary(model)`.

```
aod::wald.test(
  Sigma = vcov(fit),
  b = coef(fit),
  Terms = 2:2)
```

```
## Wald test:
## ----------
```

```
##
## Chi-squared test:
## X2 = 18.3, df = 1, P(> X2) = 1.9e-05
```

**8.2.1.2 Forward Selection** Each predictor is tested sequentially, corresponding to "forward selection".

Note that if you didn't set dummies encountering with categorical variables, fine with the linear model, but here goes wrong!

Note that `test = 'Chisq'` with capital 'C', otherwise an error will be raised.

```
anova(fit,test = 'Chisq')
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: ln_yesno
##
## Terms added sequentially (first to last)
##
##
##           Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                       1120     1216.7
## age        1   32.315      1119     1184.3 1.311e-08 ***
## pathsize   1   32.024      1118     1152.3 1.523e-08 ***
## pathscat2  1    0.510      1117     1151.8    0.4750
## pathscat3  1    0.048      1116     1151.8    0.8265
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 8.2.2 Overall Model

The deviance is the absolute value of 2 times the log-likelihood, the difference of the deviance of the null model and the designated model is the statistic used for the model comparison, which follows a $\chi^2$ distribution with its degree of freedom as the difference of the number of parameter in two models.

$$\Delta(y) = deviance_{null} - deviance \sim \chi^2_n, n = \#IV$$

Therefore, we can even carry the model test manually.

```
with(fit, null.deviance - deviance)
with(fit, df.null - df.residual)
with(fit, pchisq(null.deviance - deviance,
                 df.null - df.residual,
                 lower.tail = FALSE))
```

The result of model comparison is highly significant, meaning that the designated model can explain far more variation of the data than the null one.

But better to use `anova(model1,null-model,test = 'Chisq')`. Same result.

```
nullfit = glm(ln_yesno ~ 1,
          data = logisticData,
          family = binomial('logit'))
anova(fit,nullfit,test = 'Chisq')
```

```
## Analysis of Deviance Table
##
## Model 1: ln_yesno ~ age + pathsize + pathscat2 + pathscat3
## Model 2: ln_yesno ~ 1
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1      1116     1151.8
## 2      1120     1216.7 -4  -64.897 2.705e-13 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 8.2.3 Model Fitting

The designated model wins, but that unnecessarily means it can explains the data well. Then we're to take the test of goodness of fit, checking the observed and predicted frequencies. No significance is wanted here.

Methods for such model fitting varies. Here we use HL-test. Use `ResourceSelection::hoslem.test(x = observed_values,y = expected_values,g = 10)`, where `g` is the number of bins to calculate quantiles and `x` & `y` cannot be reversed.

```
ResourceSelection::hoslem.test(x = logisticData$ln_yesno,
                               y = fitted(fit),
                               g = 10)
```

```
##
##  Hosmer and Lemeshow goodness of fit (GOF) test
##
## data:  logisticData$ln_yesno, fitted(fit)
## X-squared = 8.9727, df = 8, p-value = 0.3446
```

*Note that the bins set should not be so dense that the sample size within each bin will fall short.*

### 8.2.4 Fitting Quality

**8.2.4.1 Log-Likelihood** Log-likelihood can be obtained using `logLik(model)` directly, with `df` telling parameters in the model (including the intercept).

```
logLik(fit)
```

```
## 'log Lik.' -575.8852 (df=5)
```

Log-likelihood can be obtained through *deviance*, which is 2 times the log-likelihood, and you can get that using `model$deviance`.

```
fit$deviance
```

```
## [1] 1151.77
```

#### 8.2.4.2 R Squares

Obtain all 5 R Squares by `pscl::pR2(model)`.

Hint:

- `r2CU` is the Nagelkerke R2;
- `r2ML` is the Cox&Snell R2;
- `llh` is the log-likelihood of the designated model;
- `llhNull` is the log-likelihood of the null model.

```
pscl::pR2(fit)
```

```
## fitting null model for pseudo-r2
```

```
##           llh       llhNull            G2      McFadden          r2ML
## -575.88522800 -608.33388920   64.89732239    0.05334022    0.05624846
##          r2CU
##    0.08494006
```

### 8.2.5 Model Comparison

You can compare one model against another using `anova(model1,model2,test = 'Chisq')`.

For example, it's natural to think of an alternative model from what we've fitted originally. Just keep all significant IV(s), and we get a new model to be compared against.

```
fit2 = glm(ln_yesno ~ age + pathsize,
           data = logisticData,
           family = binomial("logit"))
```

```
anova(fit,fit2,test = 'Chisq')
```

```
## Analysis of Deviance Table
##
## Model 1: ln_yesno ~ age + pathsize + pathscat2 + pathscat3
## Model 2: ln_yesno ~ age + pathsize
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1      1116     1151.8
## 2      1118     1152.3 -2 -0.55846   0.7564
```

The significance is shown under `Pr(>Chi)`. And `Df` shows the degree of freedom of the certain $\chi^2$ distribution. `Deviance` shows the $\chi^2$ statistic value.

In this case, a simpler model does not perform statistically worse and is considered better.

### 8.2.6 Classification Table

It's the last step of logistic regression. Just see how well the model works when putting that into practice!

```
pred_y = round(predict(fit,type='response'),0)

classification_df = data.frame(
  observed_y = logisticData$ln_yesno,
  predicted_y = round(pred_y,0))

xtabs(~predicted_y+observed_y,data=classification_df)
```

```
##            observed_y
## predicted_y   0   1
##           0 846 246
##           1  14  15
```

You can also use the `caret::confusionMatrix()` to make a classification table, and get all the performance metrics conveniently.

```
library(caret)
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:survival':
##
##     cluster
```

```
classtab = confusionMatrix(factor(pred_y),factor(logisticData$ln_yesno))
classtab
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 846 246
##          1  14  15
##
##                Accuracy : 0.7681
##                  95% CI : (0.7422, 0.7925)
##     No Information Rate : 0.7672
##     P-Value [Acc > NIR] : 0.4884
##
##                   Kappa : 0.0597
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.98372
##             Specificity : 0.05747
##          Pos Pred Value : 0.77473
##          Neg Pred Value : 0.51724
##              Prevalence : 0.76717
```

```
##          Detection Rate : 0.75468
##    Detection Prevalence : 0.97413
##       Balanced Accuracy : 0.52060
##
##        'Positive' Class : 0
##
```

### 8.3 Report Results

A logistic regression analysis was conducted to predict the occurrence of cancer using cancer category, age and tumor size as predictors. A test of the full model against a constant-only model was statistically significant, indicating that the predictors as a set reliably distinguished between cancer and non-cancer ($\chi^2(4) = 64.897, p < .001$).

Nagelkerke's R2 of .085 indicated a weak relationship between prediction and grouping. Prediction success overall was 76.8% (98.4% for detecting a non-cancer and 5.7% for cancer). The Wald criterion demonstrated that only age and tumor size made a significant contribution to prediction (p $<$ .001 and p =.001, respectively). Cancer category was not a significant predictor. EXP(B) value indicates that when tumor size is raised by one unit the odds ratio is 1.5 times as large and therefore the tumor are 1.5 more times likely to be cancer.

## 9 Multi-Linear Regression

The data `df` presented in MLR contains information about salary, beginning salary, education year, job category, previous working experience and so on. The relationship between salary and beginning salary, education, to name a few, is of interest.

```
df = readxl::read_excel("data02-01.xlsx", sheet = 1)
attach(df)
```

```
## The following objects are masked _by_ .GlobalEnv:
##
##     age, jobcat
```

## 9.1 Pre-Inspection

### 9.1.1 linear relationship

Use `cor()` to carry the correlation test, the default method of which is `method = 'pearson'`. The result will be presented as a correlation matrix.

However, obviously the correlation matrix won't give you information about p-value (or say, significance).

Good thing is that the check of linear relationship is the first and rough step for further regression, maybe other more information (about p-value or CIs) is unnecessary. Another good thing is that the `cor()` is from base R, no extra requirement to library packages.

```
df |> dplyr::select(salary,educ,salbegin,jobtime,prevexp) |> cor()
```

```
##               salary        educ    salbegin      jobtime      prevexp
## salary    1.00000000  0.66055891  0.88011747  0.084092267 -0.097466926
## educ      0.66055891  1.00000000  0.63319565  0.047378777 -0.252352521
## salbegin  0.88011747  0.63319565  1.00000000 -0.019753475  0.045135627
## jobtime   0.08409227  0.04737878 -0.01975347  1.000000000  0.002978134
## prevexp  -0.09746693 -0.25235252  0.04513563  0.002978134  1.000000000
```

Well, it's clumsy to compress the target data (variables) into a matrix to take the test and get the correlation matrix. Just fine to know a bit about that.

```
attach(df)
```

```
## The following objects are masked _by_ .GlobalEnv:
##
```

```
##     age, jobcat

## The following objects are masked from df (pos = 3):
##
##     age, bdate, educ, gender, id, jobcat, jobtime, minority, prevexp,
##     salary, salbegin
```

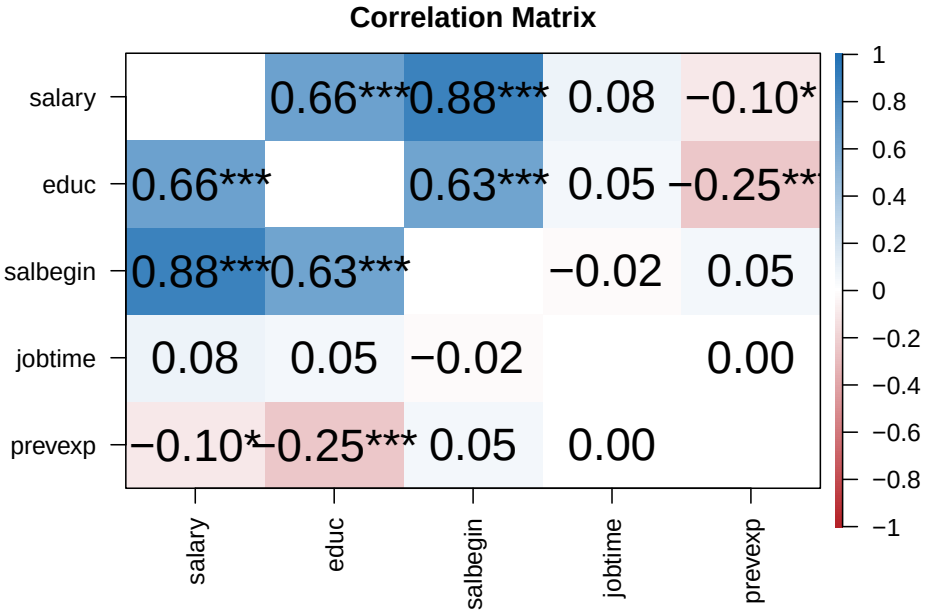
```
# first create an empty matrix, primarily filled with NA
m = matrix(nrow=nrow(salaryData),ncol=5)
m[,1]=salary
m[,2]=educ
m[,3] = salbegin
m[,4]=jobtime
m[,5]=prevexp
#check the correlation matrix
cor(m)
```

```
##             [,1]        [,2]        [,3]         [,4]         [,5]
## [1,]  1.00000000  0.66055891  0.88011747  0.084092267 -0.097466926
## [2,]  0.66055891  1.00000000  0.63319565  0.047378777 -0.252352521
## [3,]  0.88011747  0.63319565  1.00000000 -0.019753475  0.045135627
## [4,]  0.08409227  0.04737878 -0.01975347  1.000000000  0.002978134
## [5,] -0.09746693 -0.25235252  0.04513563  0.002978134  1.000000000
```

Additionally, I'd like to use `bruceR::Corr()` for the correlation matrix, which is visible for both in amount and in significance. Moreover, it's fancy!

For simplicity, the two decimals by default is well enough. If more accuracy is needed, you can set the parameter `digits =`.

```
df |> dplyr::select(salary,educ,salbegin,jobtime,prevexp) |>
  bruceR::Corr(digits = 2)
```




```
## Correlation matrix is displayed in the RStudio `Plots` Pane.
##
## Pearson's r and 95% confidence intervals:
##
##                        r       [95% CI]     p       N
##
## salary-educ         0.66 [ 0.61,  0.71] <.001 *** 474
## salary-salbegin     0.88 [ 0.86,  0.90] <.001 *** 474
## salary-jobtime      0.08 [-0.01,  0.17]  .067 .   474
## salary-prevexp     -0.10 [-0.19, -0.01]  .034 *   474
## educ-salbegin       0.63 [ 0.58,  0.68] <.001 *** 474
## educ-jobtime        0.05 [-0.04,  0.14]  .303     474
## educ-prevexp       -0.25 [-0.33, -0.17] <.001 *** 474
## salbegin-jobtime   -0.02 [-0.11,  0.07]  .668     474
## salbegin-prevexp    0.05 [-0.05,  0.13]  .327     474
## jobtime-prevexp     0.00 [-0.09,  0.09]  .948     474
##
```

### 9.1.2 multicollinearity

Use `car::vif(model)` and get the result directly.

For the convenience of narration, check of multicollinearity is presented here. Most of the time, the check is a "post-hoc" one, after obtaining the fitted model.

VIF less than 2 is ideal, and less than 5 is the bottom line.

```
fit = lm(salary ~ educ + salbegin + jobtime + prevexp,data = df)
car::vif(fit)
```

```
##     educ salbegin  jobtime  prevexp
## 1.937010 1.813582 1.007613 1.155814
```

## 9.2 Fit the Model

### 9.2.1 basic summary

Firstly, use `lm()` to get the linear model.

Subsequently, use `summary()` to get the detailed result.

Period.

```
fit = lm(salary ~ educ + salbegin + jobtime + prevexp,data = df)
summary(fit)
```

```
##
## Call:
## lm(formula = salary ~ educ + salbegin + jobtime + prevexp, data = df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -29600  -4119  -1246   2642  46079
##
```

```
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.615e+04  3.255e+03  -4.961 9.84e-07 ***
## educ         6.699e+02  1.656e+02   4.045 6.11e-05 ***
## salbegin     1.768e+00  5.873e-02  30.111  < 2e-16 ***
## jobtime      1.615e+02  3.425e+01   4.715 3.19e-06 ***
## prevexp     -1.730e+01  3.528e+00  -4.904 1.30e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7465 on 469 degrees of freedom
## Multiple R-squared:  0.8105, Adjusted R-squared:  0.8089
## F-statistic: 501.5 on 4 and 469 DF,  p-value: < 2.2e-16
```

It's noteworthy that `bruceR::print_table(model)` can spring up the regression result as a decent table, and can be exported to a `.doc` file.

However, fine with `summary()`, decency doesn't matter much here, and the `summary()` can give more information about the model itself that is essential, such F value and multiple R squared.

### 9.2.2 coefficients

Use `coefficients(model)` from base R to see the coefficients for each IV.

Furthermore, use `confint(model,level = 0.95)` from base R to see the CIs (95% by default) for each coefficients.

```
coefficients(fit)
```

```
##   (Intercept)          educ      salbegin       jobtime       prevexp
## -16149.671204    669.913570      1.768427    161.485642    -17.303251
```

```
confint(fit, level=0.95)
```

```
##                     2.5 %       97.5 %
## (Intercept) -22546.784635 -9752.557773
```

```
## educ           344.511104   995.316035
## salbegin         1.653019     1.883835
## jobtime         94.190179   228.781104
## prevexp        -24.236663   -10.369839
```

Multiple R squared is reported through `summary()`.

By definition, Multiple R squared is defined as the square of the correlation between estimated (fitted) value and real values.

```
cor(salary,fit$fitted.values)
```

```
## [1] 0.9002721
```

### 9.2.3 model comparison

Compare the model with the "zero" model, formally called the null model. The null model has no IV but with intercept, which is the mean. The null model uses the mean to predict the data or explain the variation of values.

Get the null model by formula `DV~1`.

```
fit2 = lm(salary ~ 1 )
```

Then use `anova(model1,model2)` to compare the two models.

```
anova(fit,fit2)
```

```
## Analysis of Variance Table
##
## Model 1: salary ~ educ + salbegin + jobtime + prevexp
## Model 2: salary ~ 1
##   Res.Df        RSS Df   Sum of Sq      F    Pr(>F)
## 1    469 2.6137e+10
## 2    473 1.3792e+11 -4 -1.1178e+11 501.45 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

If we think further, the F test for the MLR model is to test with null hypothesis that all the coefficients of all the IVs are zero. Therefore, the model comparison by essence coincides with the F test! Obviously, the F value "copies" that in the `summary()` result.

## 9.3 Standard MLR

Transform the formula with each variable embedded into the function `scale()`, no other adjustments!

Under standard MLR, the coefficients are standardized ones.

```
fit_std = lm(scale(salary) ~ scale(educ) + scale(salbegin) +
               scale(jobtime) + scale(prevexp),
             data = df)
```

```
summary(fit_std)
```

```
##
## Call:
## lm(formula = scale(salary) ~ scale(educ) + scale(salbegin) +
##     scale(jobtime) + scale(prevexp), data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.73349 -0.24121 -0.07298  0.15475  2.69850
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -1.129e-16  2.008e-02   0.000        1
## scale(educ)      1.132e-01  2.798e-02   4.045 6.11e-05 ***
## scale(salbegin)  8.151e-01  2.707e-02  30.111  < 2e-16 ***
## scale(jobtime)   9.515e-02  2.018e-02   4.715 3.19e-06 ***
## scale(prevexp)  -1.060e-01  2.161e-02  -4.904 1.30e-06 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4372 on 469 degrees of freedom
## Multiple R-squared:  0.8105, Adjusted R-squared:  0.8089
## F-statistic: 501.5 on 4 and 469 DF,  p-value: < 2.2e-16
```

### 9.3.1 (semi-) partial for x

(Semi-) partial correlation indicates the variance accounted for by the certain IV specifically, which is especially intriguing and important in standard MlR.

Use `pcor()` from the package `ppcor` to compute the partial correlation for IVs.

Use `spcor()` from the package `ppcor` to compute the semi-partial correlation for IVs.

Throw the regressed variables (both IVs & DV) into the functions above, and period.

The results will include estimated value of correlation and its p-value. However, the drawback is that the returned matrix does not include rownames and colnames, making it reading-unfriendly. However, we only care about the (semi-) partial correlation between DV and all IVs, which means only the very one row or column is enough.

Therefore, DV goes first! Only check the first row or column!

```
# partial correlation
df |> dplyr::select(salary,educ,salbegin,jobtime,prevexp) |>
  ppcor::pcor()
```

```
## $estimate
##              salary        educ   salbegin     jobtime     prevexp
## salary    1.0000000  0.18362578  0.8118323  0.21275206 -0.22085427
## educ      0.1836258  1.00000000  0.2347785  0.04216897 -0.30912243
```

```
## salbegin  0.8118323  0.23477855  1.0000000 -0.21340370  0.33602452
## jobtime   0.2127521  0.04216897 -0.2134037  1.00000000  0.07981028
## prevexp  -0.2208543 -0.30912243  0.3360245  0.07981028  1.00000000
##
## $p.value
##                 salary         educ      salbegin      jobtime      prevexp
## salary    0.000000e+00 6.105604e-05 1.163309e-111 3.186348e-06 1.296712e-06
## educ      6.105604e-05 0.000000e+00  2.550384e-07 3.611647e-01 6.903840e-12
## salbegin 1.163309e-111 2.550384e-07  0.000000e+00 2.967847e-06 6.775217e-14
## jobtime   3.186348e-06 3.611647e-01  2.967847e-06 0.000000e+00 8.358673e-02
## prevexp   1.296712e-06 6.903840e-12  6.775217e-14 8.358673e-02 0.000000e+00
##
## $statistic
##             salary       educ  salbegin    jobtime   prevexp
## salary    0.000000  4.0454629 30.110719  4.7153987 -4.904006
## educ      4.045463  0.0000000  5.230663  0.9140415 -7.039248
## salbegin 30.110719  5.2306629  0.000000 -4.7305296  7.726346
## jobtime   4.715399  0.9140415 -4.730530  0.0000000  1.733935
## prevexp  -4.904006 -7.0392480  7.726346  1.7339351  0.000000
##
## $n
## [1] 474
##
## $gp
## [1] 3
##
## $method
## [1] "pearson"
```

```
# part or semi-partial correlation
df |> dplyr::select(salary,educ,salbegin,jobtime,prevexp) |>
  ppcor::spcor()
```

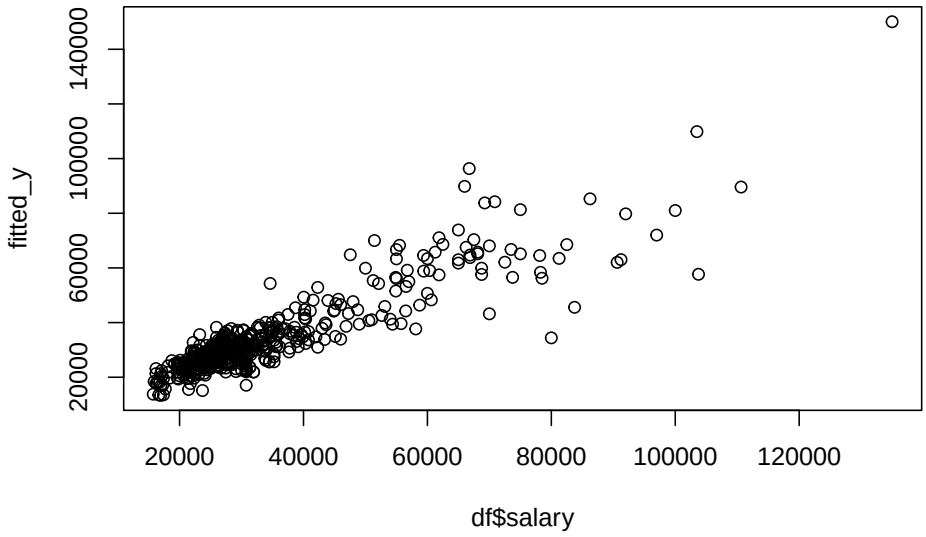
```
## $estimate
```

```
##              salary        educ   salbegin     jobtime     prevexp
## salary    1.0000000  0.08132013  0.6052725  0.09478688 -0.09857818
## educ      0.1319374  1.00000000  0.1705911  0.02981024 -0.22957564
## salbegin  0.6028339  0.10472088  1.0000000 -0.09470792  0.15468589
## jobtime   0.2119468  0.04108416 -0.2126269  1.00000000  0.07793657
## prevexp  -0.2054291 -0.29487448  0.3236570  0.07263463  1.00000000
##
## $p.value
##                salary         educ     salbegin    jobtime      prevexp
## salary   0.000000e+00 7.788628e-02 2.067811e-48 0.03975543 3.244186e-02
## educ     4.126643e-03 0.000000e+00 1.994428e-04 0.51867959 4.739749e-07
## salbegin 6.161045e-48 2.303088e-02 0.000000e+00 0.03992148 7.559174e-04
## jobtime  3.477648e-06 3.736610e-01 3.230041e-06 0.00000000 9.112418e-02
## prevexp  6.973073e-06 6.655125e-11 6.011570e-13 0.11543038 0.000000e+00
##
## $statistic
##             salary       educ  salbegin    jobtime   prevexp
## salary    0.000000  1.7669539 16.466993  2.0620275 -2.145298
## educ      2.882488  0.0000000  3.749348  0.6458697 -5.108220
## salbegin 16.362654  2.2804165  0.000000 -2.0602942  3.390753
## jobtime   4.696710  0.8904871 -4.712493  0.0000000  1.692976
## prevexp  -4.545809 -6.6830784  7.407988  1.5771712  0.000000
##
## $n
## [1] 474
##
## $gp
## [1] 3
##
## $method
## [1] "pearson"
```

## 9.4 Goodness of Fit

### 9.4.1 fitted values & real values

```
fitted_y = fitted(fit) # predicted values
plot(df$salary,fitted_y,'p')
```

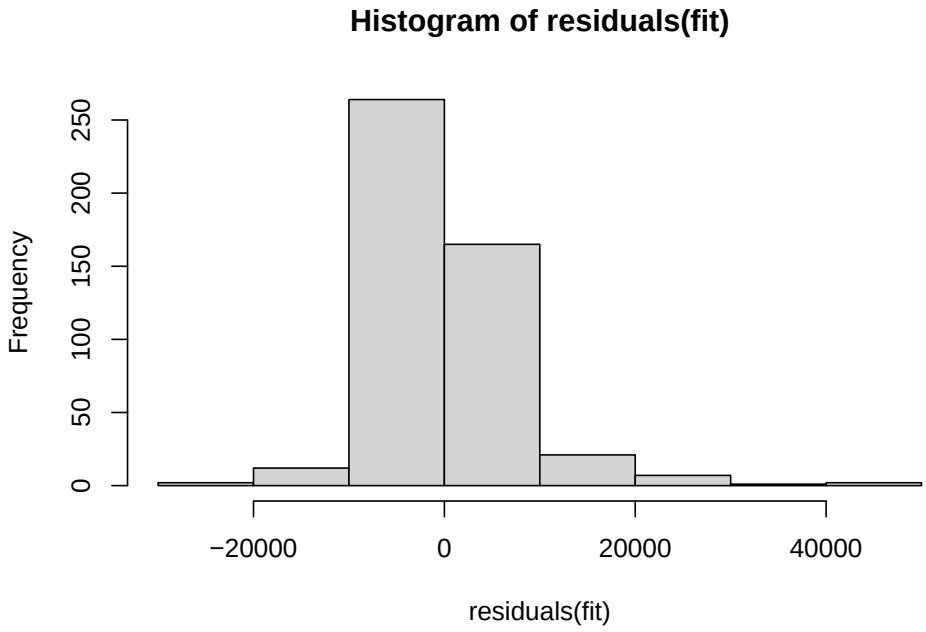



### 9.4.2 residual

If the model fits well, then the residual should follow a normal distribution.

Fitted model's residuals can be obtained through `residuals(model)` or `model$residuals`. This works the same for other attributes of a linear model, such as coefficients.

```
# hist(fit$residuals)
hist(residuals(fit))
```




### 9.4.3 plot more

Use `plot(model)` to get multiple plots reflecting the goodness of fit from many perspectives.

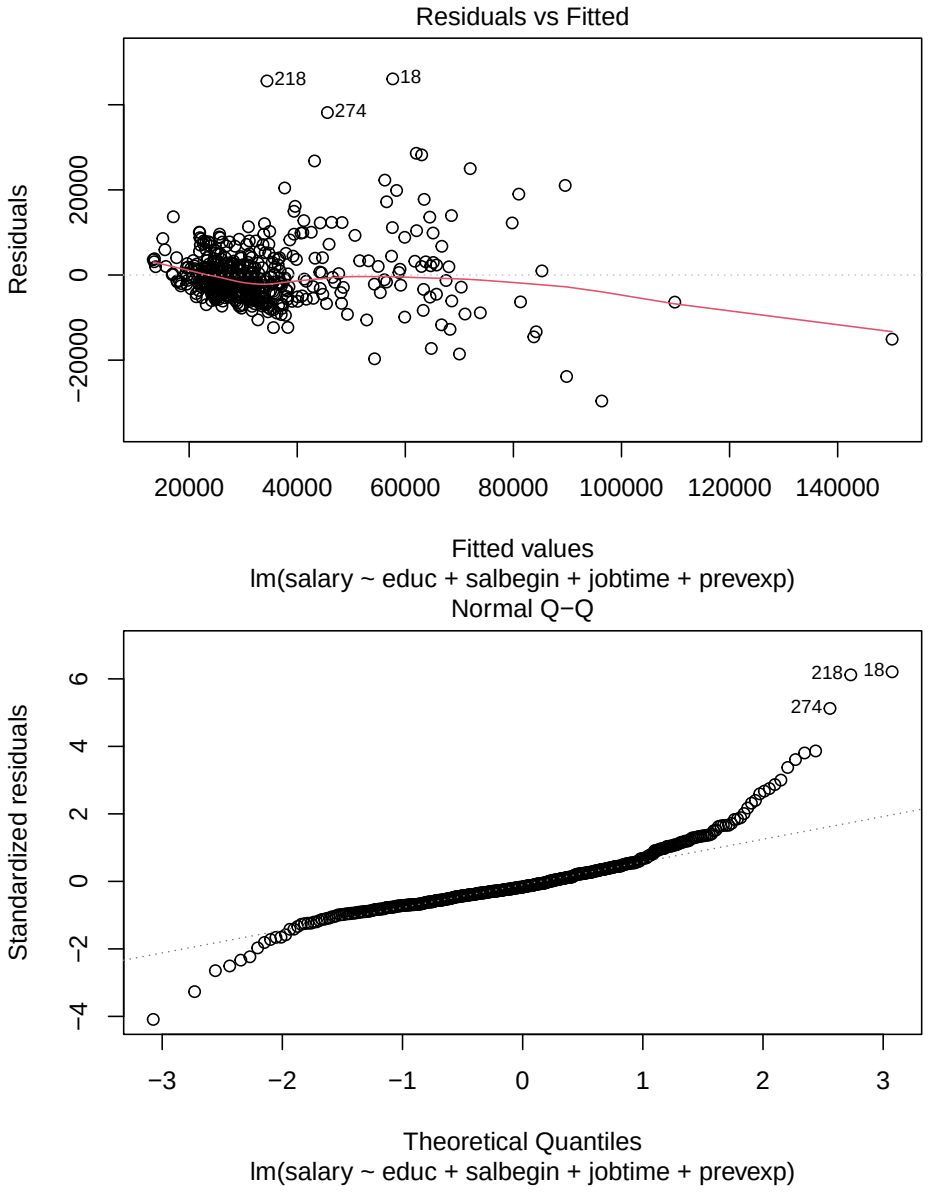
Specify parameter `which = vector_of_range` to designate the plots to show, 6 the most. `which = 1:4` by default.

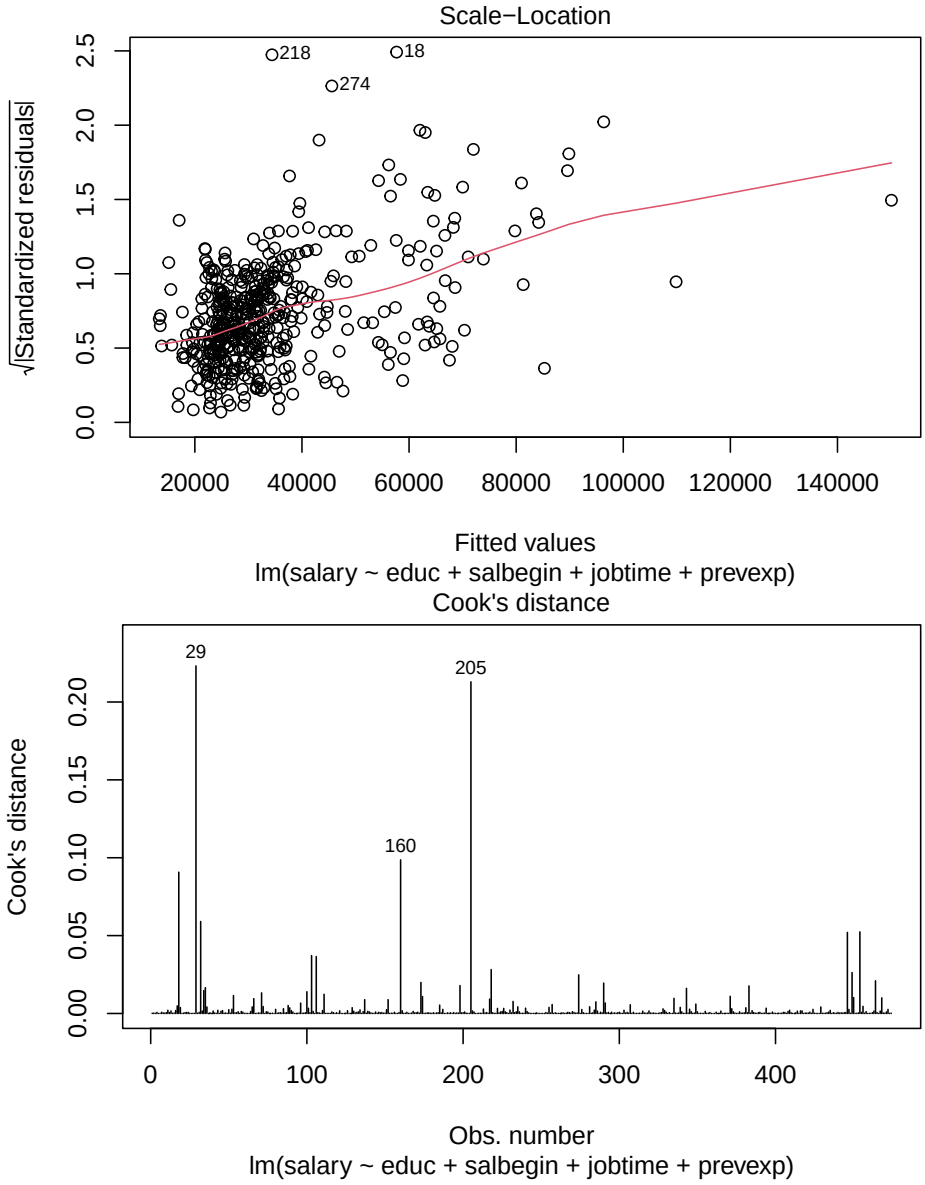
Following the order, the six plots are:

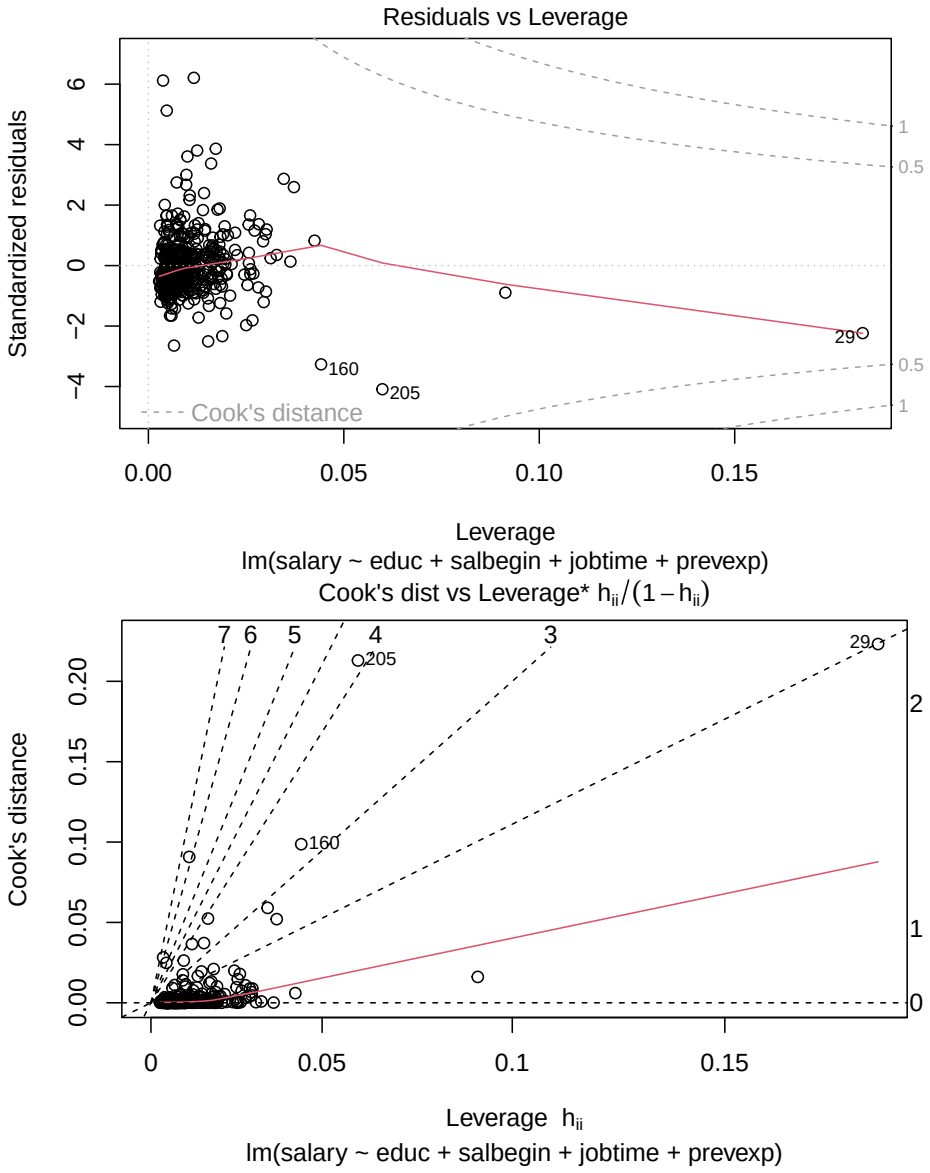
- Residuals & Fitted Values
  - test the homoscedascitiy assumption
- QQ Plot for Residuals
  - test the Gaussian assumption
- Square Root of Standardized Residuals & Fitted Value
  - similar to the first one
- Cook's Distance
  - check outliers
- Standardized Residuals & Leverage

  - See if influential points have huge residual, if so, possibly outlier!
- Cook's Distance & Leverage
  - Large in both, possibly outlier!

```
plot(fit, which=1:6)
```




Scale−Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
2.5
218
18
274
20000
40000
60000
80000
100000
120000
140000
Fitted values
lm(salary ~ educ + salbegin + jobtime + prevexp)
Cook's distance
Cook's distance
0.00
0.05
0.10
0.15
0.20
29
205
160
0
100
200
300
400
Obs. number
lm(salary ~ educ + salbegin + jobtime + prevexp)

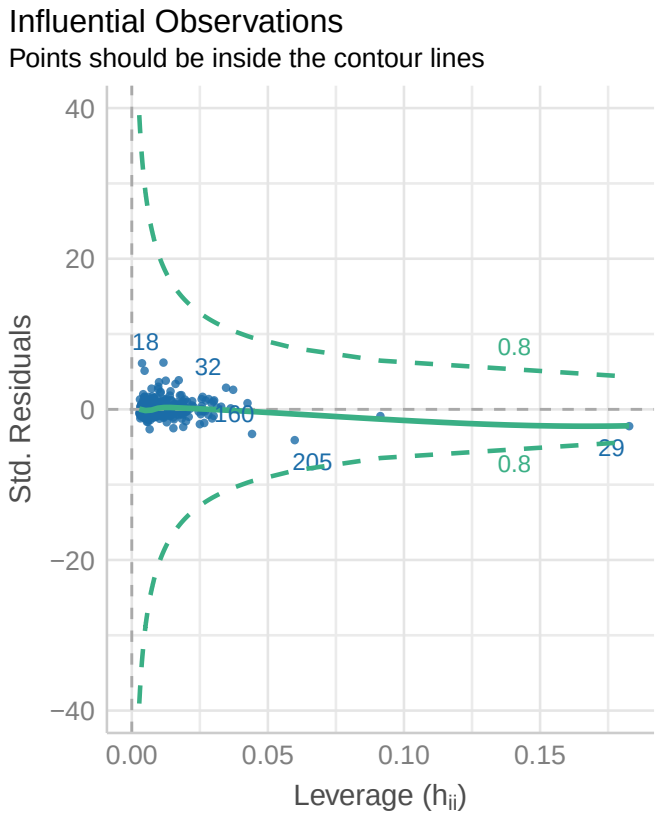



There's another way to do the model checking. Use `performance::check_model(model)`.

The default option is to check all the "vif", "qq", "normality", "linearity", "ncv", "homogeneity", "outliers", "reqq", "pp_check", "binned_residuals" and "overdispersion", which can be specified through `check = specifying_vector`. Here I'll take the check of outliers as an

example.

```
performance::check_model(fit,check = 'outliers')
```




Do not forget to `detach()` the attached data.

```
detach(df)
```

# 10 Appendix

## 10.1 `bruceR::MANOVA()`

`bruceR::MANOVA()` is an old friend. What's new here is that, for covariate(s), just specify the parameter `covariate`. If single, a character is good; if multiple, compress charactors into a factor.

Mind the differences between `covariate` and `between`. The latter should put interaction(s) into consideration.

For the ANCOVA example above,

```
ancfit = bruceR::MANOVA(data = miner,
                        dv = 'vitalcp',
                        between = 'time',
                        covariate = 'age' )
```

```
## Warning: Numerical variables NOT centered on 0 (matters if variable in interaction):
##    age

##
## ====== ANOVA (Between-Subjects Design) ======
##
## Descriptives:
##
##  "time"  Mean    S.D.  n
##
##   time1 3.949 (1.198) 12
##   time2 4.122 (0.768) 16
##
## Total sample size: N = 28
##
## ANOVA Table:
## Dependent variable(s):      vitalcp
## Between-subjects factor(s): time
## Within-subjects factor(s):  -
## Covariate(s):               age
##
##           MS   MSE df1 df2      F     p      ²p [90% CI of  ²p]   ²G
##
## time   0.542 0.550   1  25  0.985  .330       .038 [.000, .218] .038
## age   10.881 0.550   1  25 19.775 <.001 ***   .442 [.196, .617] .442
```

```
##
## MSE = mean square error (the residual variance of the linear model)
##  ²p = partial eta-squared = SS / (SS + SSE) = F * df1 / (F * df1 + df2)
##  ²p = partial omega-squared = (F - 1) * df1 / (F * df1 + df2 + 1)
##  ²G = generalized eta-squared (see Olejnik & Algina, 2003)
## Cohen' s f² =  ²p / (1 -  ²p)
##
## Levene' s Test for Homogeneity of Variance:
##
##              Levene' s F df1 df2     p
##
## DV: vitalcp       2.701   1  26  .112
##
```

For the Latin Square design's example above,

```
latfit = bruceR::MANOVA(data = yie,
                        dv = 'yield',
                        covariate = c('rep','col'),
                        between = 'variety')
```

```
##
## ====== ANOVA (Between-Subjects Design) ======
##
## Descriptives:
##
##  "variety"   Mean    S.D.  n
##
##   variety1 17.367 (1.026) 12
##   variety2 17.817 (1.206) 12
##   variety3 17.475 (1.189) 12
##   variety4 17.367 (1.251) 12
##   variety5 18.883 (1.093) 12
##   variety6 17.367 (0.953) 12
##
```

```
## Total sample size: N = 72
##
## ANOVA Table:
## Dependent variable(s):      yield
## Between-subjects factor(s): variety
## Within-subjects factor(s):  -
## Covariate(s):               rep, col
##
##             MS   MSE df1 df2     F     p      ²p [90% CI of  ²p]   ²G
##
## variety  4.313 1.380   5  56 3.124  .015 *     .218 [.029, .331] .218
## rep      0.892 1.380   5  56 0.646  .666       .055 [.000, .100] .055
## col      0.339 1.380   5  56 0.246  .940       .021 [.000, .008] .021
##
## MSE = mean square error (the residual variance of the linear model)
##  ²p = partial eta-squared = SS / (SS + SSE) = F * df1 / (F * df1 + df2)
##  ²p = partial omega-squared = (F - 1) * df1 / (F * df1 + df2 + 1)
##  ²G = generalized eta-squared (see Olejnik & Algina, 2003)
## Cohen' s f² =  ²p / (1 -  ²p)
##
## Levene' s Test for Homogeneity of Variance:
##
##            Levene' s F df1 df2     p
##
## DV: yield       0.241   5  66  .943
##
```

## 10.2 Set Dummies

### 10.2.1 `model.matrix()`

To set dummies in R, you can use the "factor" function to convert categorical variables into a series of binary variables (i.e., dummies) that can be used

in a regression model.

Here's an example of how to set dummies in R:

```r
# load the mtcars dataset
data(mtcars)

# convert the gear variable to a factor variable
mtcars$gear <- factor(mtcars$gear)

# create dummies for the gear variable
dummies <- model.matrix(~gear - 1, data = mtcars)

# view the dummies
head(dummies)
```

```
##                   gear3 gear4 gear5
## Mazda RX4             0     1     0
## Mazda RX4 Wag         0     1     0
## Datsun 710            0     1     0
## Hornet 4 Drive        1     0     0
## Hornet Sportabout     1     0     0
## Valiant               1     0     0
```

In this example, we first load the mtcars dataset and convert the `gear` variable to a factor variable using the `factor()` function. This converts the `gear` variable **from a numeric variable to a categorical variable, which can then be converted to dummies**.

Next, we use the `model.matrix()` function to create dummies for the `gear` variable. The "~`gear` - 1" formula tells R to create dummies for the `gear` variable, but to **exclude the intercept term** (i.e., the constant term).

Finally, we view the dummies using the `head()` function. The output shows a matrix with three columns, corresponding to the three possible values of the `gear` variable (3, 4, and 5), and a row for each observation in the mtcars

dataset. Each element in the matrix is either 0 or 1, representing whether or not the corresponding observation has a particular value of the `gear` variable.

Note that dummies can also be created for multiple categorical variables simultaneously using the "cbind" function. For example, if you have two categorical variables "`gear`" and "cyl", you can create dummies for both variables using the following code:

```
dummies <- cbind(model.matrix(~gear - 1, data = mtcars),
                 model.matrix(~cyl - 1, data = mtcars))

head(dummies)
```

```
##                   gear3 gear4 gear5 cyl
## Mazda RX4             0     1     0   6
## Mazda RX4 Wag         0     1     0   6
## Datsun 710            0     1     0   4
## Hornet 4 Drive        1     0     0   6
## Hornet Sportabout     1     0     0   8
## Valiant               1     0     0   6
```

This will create a matrix with columns for each possible value of the `gear` variable and each possible value of the `cyl` variable, and a row for each observation in the mtcars dataset.

Note that for a categorical variable with $n$ levels, you can only plug in $n-1$ of the dummies into the regression data, in case of multicollinearity.

### 10.2.2 `fastDummies::dummy_cols()`

Pipe-friendly functions used to create dummies for categorical variables include

- `fastDummies::dummy_cols()`
- `caret::dummyVars()`

Here's an example of how to use the `dummy_cols()` function to create dummies for the `gear` and `cyl` variable in the `mtcars` dataset:

```
mtcars %>%
  mutate(gear = factor(gear),
         cyl = factor(cyl)) %>%
  fastDummies::dummy_cols(
    select_columns = c("gear",'cyl'),
    remove_first_dummy = TRUE) |>
  head() |>
  dplyr::select(contains('gear'),contains('cyl'))
```

```
##   gear gear_4 gear_5 cyl cyl_6 cyl_8
## 1    4      1      0   6     1     0
## 2    4      1      0   6     1     0
## 3    4      1      0   4     0     0
## 4    3      0      0   6     1     0
## 5    3      0      0   8     0     1
## 6    3      0      0   6     1     0
```

In this example, we use the `mutate` function to convert the `gear` and `cyl` variable to a factor variable (it works fine with `dummy_cols()` even skip this step, but wrong with `dummyVars()`). We then use the `dummy_cols()` function to create dummies for the `gear` and `cyl` variable. The "select_columns" parameter specifies the column(s) to create dummies for (packed in a vector), and the `remove_first_dummy` parameter specifies whether or not to remove the first dummy (i.e., the constant term).

The output of the `dummy_cols()` function is a new data frame with the dummies added as new columns. This can be piped into other functions for further analysis or modeling.

### 10.2.3 `caret::dummyVars()`

Usage of this function will be clear after a vivid example.

```
library(caret)
dummyVars( ~ gear + factor(cyl), data = mtcars) %>%
  predict(mtcars) |> head()
```

```
##                   gear.3 gear.4 gear.5 factor(cyl)4 factor(cyl)6 factor(cyl)8
## Mazda RX4              0      1      0            0            1            0
## Mazda RX4 Wag          0      1      0            0            1            0
## Datsun 710             0      1      0            1            0            0
## Hornet 4 Drive         1      0      0            0            1            0
## Hornet Sportabout      1      0      0            0            0            1
## Valiant                1      0      0            0            1            0
```

In summary, the `predict()` function is used to apply the dummies that were created by the `dummyVars()` function to a new dataset. It is a way to use the same set of dummies that were created for one dataset to create dummies for a different dataset that contains the same categorical variable as the original dataset.

Note that the "dummyVars()" function can also be used to create dummies for multiple categorical variables simultaneously using the "+" operator in the formula.

Note that categorical variable should be converted to factor type before.

## 10.3 Prediction

### 10.3.1 Basic Usage

In R, the `predict()` function is used to generate predicted values. It can be used with various types of models (such as linear regression, logistic regression, decision trees, etc.) and can generate corresponding predicted values based on input of new data.

Here are the basic steps to use the `predict()` function:

- First, pass the model object to the `predict()` function.

  If no more parameters are specified, then the `predict(model)` will return the fitted values of such model. You can image that as passing all IV(s) into the parameter `newdata` to make predictions. However, you can get the fitted values through `model$fitted.values` and `fitted(fit)`. The folllowing three lines of command will give you the completely same result.

```
predict(fit)
fit$fitted.values
fitted(fit)
```

- Next, pass the new data wanted to predict to the `predict()` function.

  This new data should be a data frame or matrix, with the number of **columns** matching the number of predictor variables used in the model.

  In our example presented in "Logistic Regression" above, the model has four predictor variables. All the four predictors should be incorporated into the data frame. In case of problems of variables' order, the colnames should correspond with names used in the model.

```
fit = glm(ln_yesno ~ age + pathsize + pathscat2 + pathscat3,
          data = logisticData,
          family = binomial('logit'))
```

For instance, we're to predict a case with `pathscat == 2`, `pathsize == 2.0`, `age == 66` and another case with `pathscat == 1`, `pathsize == 0.5`, `age == 20`. Accordingly, create the data frame containing the conditions of the two cases.

```
conditions = data.frame(
  pathscat2 = c(1,0),
  pathscat3 = c(0,0),
  pathsize = c(2,.5),
```

```
  age = c(66,20)
)

predict(fit,newdata = conditions,type = 'response')
```

```
##         1         2
## 0.1847871 0.2964413
```

`predict(model,newdata)` will return a vector containing the estimated predicted values. If you have multiple predictor variables, the predicted values will be a matrix, with each row representing a new predicted value.

### 10.3.2 `type`

The `type` parameter in the `predict()` function in R is used to specify the type of prediction that you want to generate. The possible values for the `type` parameter depend on the type of model that you are using. The default value of `type` is `response`, which is typically the most appropriate option for most models.

For example, if you are using a linear regression model, the `type` parameter can be set to `response` or `terms`. If you set `type` to `response`, the prediction() function will return the predicted values of the dependent variable. If you set `type` to "terms", the function will return the predicted values of the individual terms in the model (i.e., the predicted values for each predictor variable).

Similarly, if you are using a logistic regression model, the `type` parameter can be set to `response`, `terms`, or `link`. If you set `type` to `response`, the `predict()` function will return the predicted **probabilities** of the binary outcome variable. If you set `type` to `terms`, the function will return the predicted values of the individual terms in the model. If you set `type` to `link`, the function will return the predicted values on the **log-odds** scale.

In generalized linear models (GLMs), such as logistic regression or Poisson

regression, the `link` scale refers to the scale used in modeling the relationship between the predictor variables and the response variable. The link function maps the predicted values to the scale of the response variable. For example, in logistic regression, the link function is typically the logit function, which maps the probability of a binary outcome to the log odds of that outcome.

When you set `type` to `link` in the prediction() function, it returns the predicted values on the link scale, which may be different from the scale of the response variable.

On the other hand, when you set `type` to `response` in the prediction() function, it returns the predicted values on the scale of the response variable. For example, in logistic regression, the predicted values would be the probabilities of the binary outcome.

The choice between `link` and `response` depends on the purpose of the analysis. If you want to compare the predicted values for different values of the predictor variables on the scale of the response variable, then `response` is the appropriate option. If you want to perform further analysis or transformation on the predicted values, such as calculating odds ratios or performing hypothesis tests, then `link` is the appropriate option.

Back to our example, we can see it more clearly.

```
predict(fit,newdata = conditions,type = 'response')
```

```
##         1         2
## 0.1847871 0.2964413
```

```
predict(fit,newdata = conditions,type = 'link')
```

```
##          1          2
## -1.4842449 -0.8643023
```

## 10.4 Classification Table

`confusionMatrix()` is a function from the `caret` package in R that creates a confusion matrix and computes various performance metrics for classification models. The function takes at least two arguments: `data` and `reference`.The `data` argument is a vector of predicted class labels, while the `reference` argument is a vector of actual class labels. The two vectors must have the same length.

If necessary, also specify `positive` and `dnn`. `positive` will tell the factor level that corresponds to a "positive" result. If there are only two factor levels, the first level will be used as the "positive" result. `dnn` reads a character vector of dimnames for the table.

When you call `confusionMatrix()` with these arguments, it produces a confusion matrix that shows the number of true positives, true negatives, false positives, and false negatives for the **classification model**.

Based on this confusion matrix, the function then calculates several performance metrics, including accuracy, precision, sensitivity (also known as recall), specificity, F1 score, and Kappa statistic. In addition to these metrics, the confusionMatrix() function can also compute various other statistics, such as positive predictive value (PPV), negative predictive value (NPV), and the area under the ROC curve (AUC).

More specifically,

- Accuracy: The proportion of correct predictions out of the total number of predictions.
- Precision: The proportion of true positives (i.e., correct positive predictions) out of the total number of positive predictions.
- Sensitivity/Recall: The proportion of true positives out of the total number of actual positives (i.e., the proportion of positive cases that were correctly identified).

- Specificity: The proportion of true negatives (i.e., correct negative predictions) out of the total number of actual negatives (i.e., the proportion of negative cases that were correctly identified).
- F1 score: The harmonic mean of precision and recall, which provides a balanced measure of both metrics. It ranges from 0 to 1, with higher values indicating better performance.
- Kappa statistic: A measure of agreement between the predicted and actual class labels that accounts for the proportion of agreement that is expected by chance alone. It ranges from -1 to 1, with higher values indicating better agreement.
- Positive predictive value (PPV): The proportion of true positives out of the total number of positive predictions.
- Negative predictive value (NPV): The proportion of true negatives out of the total number of negative predictions.
- Area under the ROC curve (AUC): A measure of the model's ability to distinguish between positive and negative cases across all possible thresholds. It ranges from 0.5 (indicating no discrimination) to 1 (indicating perfect discrimination).

Use the example provided above in "*Logistic Regression*" to show the function!

```
fit = glm(ln_yesno ~ age + pathsize + pathscat2 + pathscat3,
          data = logisticData,
          family = binomial('logit'))
```

```
caret::confusionMatrix(
  data = factor(round(fitted(fit),0)),
  reference  = factor(logisticData$ln_yesno),
  positive = '1',
  dnn = c('predicted','actual')
)
```

```
## Confusion Matrix and Statistics
##
##          actual
## predicted   0   1
##         0 846 246
##         1  14  15
##
##                Accuracy : 0.7681
##                  95% CI : (0.7422, 0.7925)
##     No Information Rate : 0.7672
##     P-Value [Acc > NIR] : 0.4884
##
##                   Kappa : 0.0597
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.05747
##             Specificity : 0.98372
##          Pos Pred Value : 0.51724
##          Neg Pred Value : 0.77473
##              Prevalence : 0.23283
##          Detection Rate : 0.01338
##    Detection Prevalence : 0.02587
##       Balanced Accuracy : 0.52060
##
##        'Positive' Class : 1
##
```

## 10.5 Attach and Detach

We can use the function `attach()` to use variables in the database conveniently. After `attach()`, column variables can be used directly without `$`. (R will do the searching work behind for you!)

However, sometimes names of the variables will conflict with each other and cause problems. Not recommended from my perspective, because you'll not know what database the variable is from, especially when you're trying to check the codes. Combined with tube-friendly and formula-pattern genre, this won't be any more trouble.

```
attach(df)
```

```
## The following objects are masked _by_ .GlobalEnv:
##
##     age, jobcat
```

```
## The following objects are masked from df (pos = 3):
##
##     age, bdate, educ, gender, id, jobcat, jobtime, minority, prevexp,
##     salary, salbegin
```

After attaching a database, remember to `detach()` that finally, in case of conflicts with other variables.

## 10.6 Other Examples

### 10.6.1 Logistic Regression

A teacher wants to investigate the factors that affect the performance of a certain course. Among them, he wants to test whether the time spent by students on course learning per week (time, in hours), whether they have taken a prerequisite course (pre, 0-not taken, 1-taken), and whether they attend the course regularly (attendance, 0-not attend, 1-attend) have an impact on whether the final grade is excellent (grade, 0-not excellent, 1-excellent). The data is recorded in "grade.sav".

```
pupils = haven::read_sav('grade.sav')
```

**10.6.1.1 Regression Model** Take `grade` as the response variable, with `pre`, `attendance` and `time` as independent variables to carry logistic regression.

```
lfit = glm(grade~pre+attendance+time,data = pupils,
           family = binomial('logit'))

summary(lfit)
```

```
##
## Call:
## glm(formula = grade ~ pre + attendance + time, family = binomial("logit"),
##     data = pupils)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8026  -0.5564   0.2268   0.6624   2.0829
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -4.9147     2.2019  -2.232 0.025615 *
## pre           2.2420     0.8585   2.611 0.009017 **
## attendance    3.1925     0.9109   3.505 0.000457 ***
## time          0.2606     0.1765   1.476 0.139871
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 81.516  on 62  degrees of freedom
## Residual deviance: 53.424  on 59  degrees of freedom
## AIC: 61.424
##
## Number of Fisher Scoring iterations: 5
```

The logistic model, together with coefficient and significance for each IV is included from the R code output above. Nevertheless, I'd like to report that formally below.

The logistic model is as follows.

$$\text{Logit}(p) = -4.9147 + 2.2420 \times pre + 3.1925 \times attendance + 0.2606 \times time$$

The coefficients of `pre` and `attendance` are significant, while that of `time` is not significant. Related information is presented on the table below.

```
bruceR::print_table(lfit)
```

```
##
##              Estimate    S.E.      z     p
##
## (Intercept)    -4.915 (2.202) -2.232  .026 *
## pre             2.242 (0.859)  2.611  .009 **
## attendance      3.193 (0.911)  3.505 <.001 ***
## time            0.261 (0.177)  1.476  .140
##
```

The significant variables have influence on the odds statistically, with positive ones indicating positively-correlated relationship. On the other hand, the one that isn't significant doesn't prove to be statistically influential on the odds.

In the case given, it's estimated that one unit increment in `pre` will induce 0.299 unit increment in Logit($p$) and the influence is significant ($z = 2.611, p < .01$), and one unit increment in `pre` will induce 0.299 unit increment in Logit($p$) and its influence is highly significant ($z = 3.505, p < .001$). `time` is estimated to entail 0.033 unit increment in Logit($p$) for every unit increment, but that's not significant ($z = 3.505, p = 0.140$).

#### 10.6.1.2 Odds Ratio

Odds Ratio is the effect size of each independent variable, which is $e^{\beta_i}$, where $\beta_i$ is the coefficient of IV $x_i$.

```
exp(cbind(OddsRatio = coef(lfit),confint(lfit)))
```

```
##                 OddsRatio        2.5 %      97.5 %
## (Intercept)  0.007337845 5.941785e-05   0.4256995
## pre          9.412000659 2.074149e+00  69.1689029
## attendance  24.350432282 5.040915e+00 200.7150914
## time         1.297725148 9.338253e-01   1.9055477
```

Odds ratio indicates one unit increase in an IV will cause the odds to change to how much times of itself. Odds ratio that is greater than 1 means the increase in the IV will cause the odds (or the probability) to increase.

In this example, the odds ratios for `pre`, `attendance` and `time` are 9.412000659, 24.350432282 and 1.297725148 respectively. All of them tend to have positive correlation with DV.

Moreover, if we're to check the 95% CI to make a statistical inference, 1 is not included with `pre` and `attendence`, and the fact corresponds with their significance as is mentioned above. However, 1 falls within 95% CI of `time`, and that reminds me of the non-significant result.

```
with(lfit, null.deviance - deviance)
```

#### 10.6.1.3 Against NULL Model

```
## [1] 28.09241
```

```
with(lfit, df.null - df.residual)
```

```
## [1] 3
```

```
with(lfit, pchisq(null.deviance - deviance,
                  df.null - df.residual,
                  lower.tail = FALSE))
```

```
## [1] 3.473385e-06
```

Model comparison between the regressed model and the null one showed a significant result, $\chi^2_3 = 28.09241, p < .001$, indicating that the model performs much better in prediction.

```
pscl::pR2(lfit)
```

#### 10.6.1.4 R Squared

```
## fitting null model for pseudo-r2
```

```
##         llh     llhNull          G2    McFadden        r2ML        r2CU
## -26.7118943 -40.7580991  28.0924094   0.3446236   0.3597594   0.4956710
```

The Nagelkerke $r^2$ is 0.4956710, indicating that the fitted model can account for 48.70% of variation of the data, which is great!

```
ResourceSelection::hoslem.test(x = pupils$grade,
                               y = fitted(lfit),
                               g = 5)
```

#### 10.6.1.5 Goodness of Fit

```
##
##  Hosmer and Lemeshow goodness of fit (GOF) test
##
## data:  pupils$grade, fitted(lfit)
## X-squared = 1.6552, df = 3, p-value = 0.6469
```

The Hosmer and Lemeshow goodness of fit (GOF) test reports that a non-significant result, enumerating all possible numbers of bins to carry such test (from 3 to 9). The number of bins presented here is the one that returns the

lowest $p$ value, but to my happiness it's far away from significant, which is 0.6469 and with $\chi^2_3 = 1.6552$.

*Note that the bins set should not be so dense that the sample size within each bin will fall short.*

The non-significant result here indicates that there's no statistically-significant differences between the fitted values and the observed values, and the model fits well with the real world!

**10.6.1.6 Prediction** Use the logistic model to predict the outcomes of the two cases.

```
stu = data.frame(
  pre = (c(0,1)),
  attendance = c(0,1),
  time = c(8,16)
)

predictions = predict(lfit,
                      newdata = stu,
                      interval = 'confidence',
                      type = 'response')
predictions
```

```
##          1          2
## 0.05573454 0.99089361
```

*Note that the* `predict()` *for logistic regression will return probability* $p$*, instead of Logit*$(p)$ *(DV of the model)!*

Therefore, as for getting excellence on this course, student a has the probability of 0.05573454, while student b has the probability of 0.99089361.

```
pred_y = round(predict(lfit,type='response'),0)

classification_df = data.frame(
  observed_y = pupils$grade,
  predicted_y = round(pred_y,0))

xtabs(~predicted_y+observed_y,data=classification_df)
```

#### 10.6.1.7 Summarize the Result

```
##             observed_y
## predicted_y  0  1
##           0 12  4
##           1 10 37
```

From the classification, we can clearly see that

$$
\begin{aligned}
Accuracy &= \frac{TP+TN}{TP+TN+FP+FN} = \frac{37+12}{37+12+4+10} = \frac{7}{9} \approx 77.8\% \\
Precision &= \frac{TP}{TP+FP} = \frac{37}{37+10} = \frac{37}{47} \approx 78.7\% \\
Sensitivity &= \frac{TP}{TP+FN} = \frac{37}{37+12} = \frac{37}{49} \approx 75.5\% \\
Specificity &= \frac{TN}{TN+FP} = \frac{12}{12+4} = \frac{3}{4} = 75.0\%
\end{aligned}
$$

**10.6.1.8 Report the Model** A logistic regression analysis was carried to predict the excellence using prerequisite's requirement, time spent each week and attendance as predictors. The logistic model is as follows.

$$\text{Logit}(p) = -4.9147 + 2.2420 \times pre + 3.1925 \times attendance + 0.2606 \times time$$

where predictor `pre` and `attendance` are significantly influential ($z_{pre} = 2.611, p < .01; z_{attendance} = 3.505, p < .001$), while predictor `time` wasn't

($z_{time} = 3.505, p = 0.140$). Odds ratio, which is effect size, can predict that when `pre` is from 0 to 1, the odds is 9.412 as large, and when `attendance` is from 0 to 1, the odds becomes 24.350 as large, both significant. The finding firmly tell that to cover the prerequisites and to attend the class are extraordinarily important when it comes to the excellence of the score! However, the odds ratio for `time` is 1.298 and not significant, meaning its influence is not statistically proved. Jointly, prerequisites and attendance is far more important than time invested, if the first two cannot be satisfied, expanding your time budget on this course may result in vain.

As for the model, Nagelkerke's $R^2$ of 0.4956710 indicated a great relationship between predictors and the outcome. Hosmer and Lemeshow goodness of fit (GOF) test was conducted, showing a great goodness of fit, with $p$ no less than 0.6469 and $\chi^2_3 = 1.6552$. Moreover, the regressed model was compared against the intercept-only null model, returning a highly significant result, implying that the model's better performance in prediction. Overall, the model has an accuracy of 77.8%, a precision of 78.7%, a sensitivity of 75.5%, and a specificity of 75.0%.

### 10.6.2 MLR

```
heart = read.csv('heart.csv')
```

```
mfit = lm(scale(heart.disease)~scale(biking)+scale(smoking),
          data = heart)
```

#### 10.6.2.1 Fit the Model

```
summary(mfit)
```

#### 10.6.2.2 Summary the Model

```
##
## Call:
## lm(formula = scale(heart.disease) ~ scale(biking) + scale(smoking),
##     data = heart)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.47658 -0.09762  0.00792  0.09671  0.42283
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -6.119e-17  6.410e-03    0.00        1
## scale(biking)  -9.403e-01  6.418e-03 -146.53   <2e-16 ***
## scale(smoking)  3.234e-01  6.418e-03   50.39   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1431 on 495 degrees of freedom
## Multiple R-squared:  0.9796, Adjusted R-squared:  0.9795
## F-statistic: 1.19e+04 on 2 and 495 DF,  p-value: < 2.2e-16
```

It's predicted that if `biking` changes for one standard deviation, then `heart.disease` is to change for -0.940 standard deviation, and that if `smoking` changes for one standard deviation, then `heart.disease` is to change for 0.323 standard deviation. Meanwhile, coefficients of both `biking` and `smoking` are highly significant with $p < .001$.

**10.6.2.3 Goodness of Fit** After qualitative analysis, now move on to quantitative works.

```
result = summary(mfit)
result$adj.r.squared
```

```
## [1] 0.9795351
```

*Adjusted* $R^2$ of 0.9795 indicates a highly awesome goodness of fit and its ability to explain the variation of the data!

```
cor(x = fitted(mfit), y = heart$heart.disease)
```

```
## [1] 0.9897563
```

The correlation between the fitted values and the observed (real) values is as high as 0.9897563, which is exactly the square root of $R^2$. Jointly speaking, the model has great credibility in prediction.

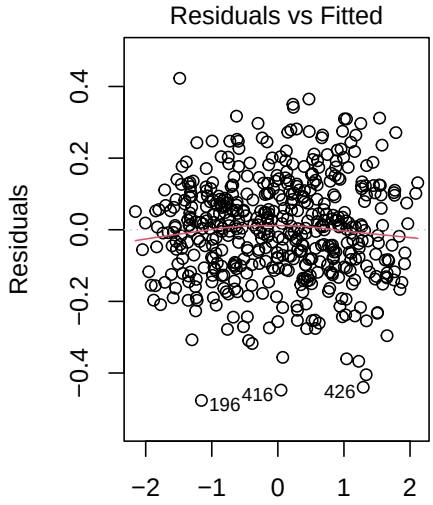
```
car::vif(mfit)
```

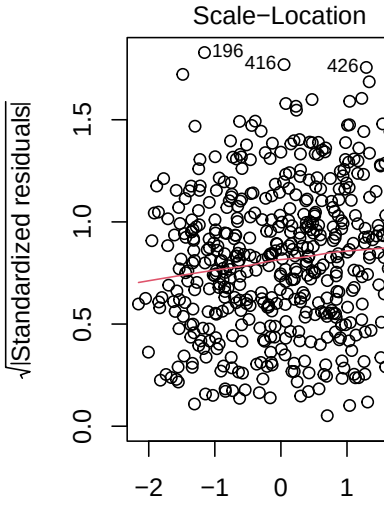
#### 10.6.2.4 VIF

```
##  scale(biking) scale(smoking)
##       1.000229       1.000229
```

VIFs of `biking` and `smoking` are 1.000229 and 1.000229 respectively, far less than 5, showing that there should be no worries about multicollinearity.

```
par(mfrow = c(1,2))
plot(mfit,which = c(1,3))
```

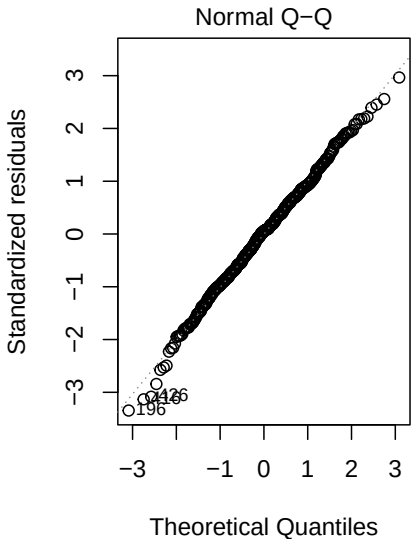


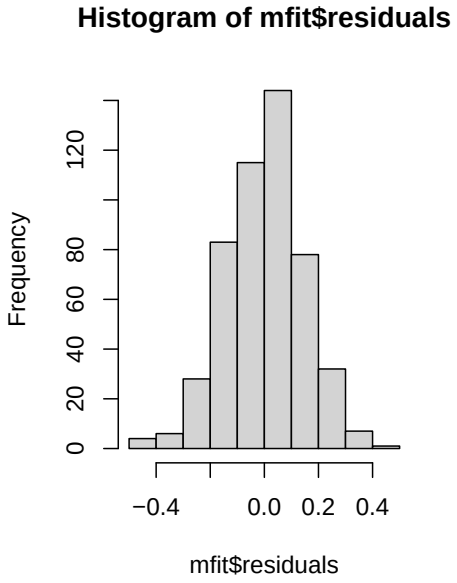




#### 10.6.2.5 Plots of Residuals

From the first plot, it's shown that there's no obvious tendency between the fitted values and the residuals.

```
par(mfrow = c(1,2))
plot(mfit,which = 2)
hist(mfit$residuals)
```



In QQ plot, the scatter points of residuals closely center around the line offered by normal distribution. In the histogram of residuals, a clear prototype of normal distribution is depicted. What's more, move on from qualitative analysis to the quantitative.

```
shapiro.test(mfit$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  mfit$residuals
## W = 0.997, p-value = 0.4935
```

```
mfit$residuals |> mean()
```

```
## [1] 2.305236e-18
```

From the Shapiro test, $W = 0.9969963, p = 0.4935143$, indicating that the residuals from the fitted model follow a normal distribution. Moreover, the mean is computed to be (extremely close to) zero.

Therefore, we can safely say that the residuals of the fitted model do follow a normal distribution.

Based on what we've discussed, we can conclude confidently that the model fits the data well!

## 10.7 `select()`

`select()` is one of the most useful functions in "tidyverse" environment. However, `select()` is incorporated in two packages, "dplyr" (or "tidyverse") and "MASS". When the two are both loaded, then there is going be a clash!

To solve this annoying problem, decorate every `select()` as `dplyr::select`, or change the settings of "library" essentially.

# 11 第二部分：期末范围（8–13 章）

```
library(bruceR)
library(tidyverse)
library(ggplot2)
```

# 12 MLR

The `job` contains information about salary, beginning salary, education year, job category, previous working experience and so on. The relationship between salary and beginning salary, education, to name a few, is of interest.

```
job = import("data02-01.xlsx", sheet = 1)
```

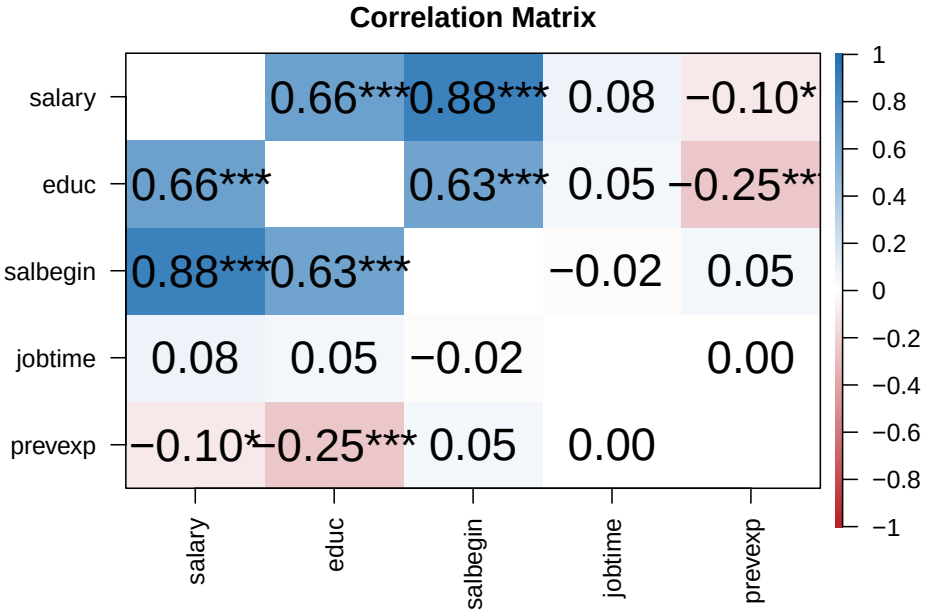
```
##   Successfully imported: 474 obs. of 11 variables
```

## 12.1 Pre-Inspection

### 12.1.1 Correlation

- `bruceR::Corr()`: correlation matrix, the default method of which is `method = 'pearson'`.
- `ppcor::spcor()`: semi-partial correlation for IVs.
- `ppcor::pcor()`: partial correlation.

```
job |> dplyr::select(salary,educ,salbegin,jobtime,prevexp) |> Corr()
```




```
## Correlation matrix is displayed in the RStudio `Plots` Pane.
##
## Pearson's r and 95% confidence intervals:
##
##                        r       [95% CI]     p       N
##
## salary-educ         0.66 [ 0.61,  0.71] <.001 *** 474
## salary-salbegin     0.88 [ 0.86,  0.90] <.001 *** 474
## salary-jobtime      0.08 [-0.01,  0.17]  .067 .   474
## salary-prevexp     -0.10 [-0.19, -0.01]  .034 *   474
## educ-salbegin       0.63 [ 0.58,  0.68] <.001 *** 474
## educ-jobtime        0.05 [-0.04,  0.14]  .303     474
## educ-prevexp       -0.25 [-0.33, -0.17] <.001 *** 474
## salbegin-jobtime   -0.02 [-0.11,  0.07]  .668     474
## salbegin-prevexp    0.05 [-0.05,  0.13]  .327     474
## jobtime-prevexp     0.00 [-0.09,  0.09]  .948     474
##
```

More will be introduced in *Correlation* chapter.

```
# partial correlation
job |> dplyr::select(salary,educ,salbegin,jobtime,prevexp) |>
  ppcor::pcor()

# part or semi-partial correlation
job |> dplyr::select(salary,educ,salbegin,jobtime,prevexp) |>
  ppcor::spcor()
```

### 12.1.2 Multicollinearity

Use `car::vif(model)` and get the result directly.

VIF less than 2 is ideal, and less than 5 is the bottom line.

```
fit = lm(salary ~ educ + salbegin + jobtime + prevexp,data = job)
car::vif(fit)
```

```
    educ salbegin  jobtime  prevexp
1.937010 1.813582 1.007613 1.155814
```

## 12.2 Fit the Model

Firstly, use `lm()` to get the linear model. Subsequently, use `summary()` to get the detailed result. Period.

```
jobfit = lm(salary ~ educ + salbegin + jobtime + prevexp,data = job)
summary(jobfit)
```

```
Call:
lm(formula = salary ~ educ + salbegin + jobtime + prevexp, data = job)

Residuals:
   Min     1Q Median     3Q    Max
-29600  -4119  -1246   2642  46079
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.615e+04  3.255e+03  -4.961 9.84e-07 ***
educ         6.699e+02  1.656e+02   4.045 6.11e-05 ***
salbegin     1.768e+00  5.873e-02  30.111  < 2e-16 ***
jobtime      1.615e+02  3.425e+01   4.715 3.19e-06 ***
prevexp     -1.730e+01  3.528e+00  -4.904 1.30e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7465 on 469 degrees of freedom
Multiple R-squared:  0.8105,    Adjusted R-squared:  0.8089
F-statistic: 501.5 on 4 and 469 DF,  p-value: < 2.2e-16
```

### 12.2.1 Coefficients & CI

Use `coefficients(model)` or `model$coefficients` to see the coefficients for each IV.

Furthermore, use `confint(model,level = 0.95)` to see the CIs (95% by default) for each coefficient.

```
jobfit$coefficients
```

```
coefficients(jobfit)
```

```
##   (Intercept)          educ      salbegin       jobtime       prevexp
## -16149.671204    669.913570      1.768427    161.485642    -17.303251
```

```
confint(jobfit, level=0.95)
```

```
##                     2.5 %       97.5 %
## (Intercept) -22546.784635 -9752.557773
## educ           344.511104   995.316035
## salbegin         1.653019     1.883835
```

```
## jobtime          94.190179   228.781104
## prevexp         -24.236663   -10.369839
```

### 12.2.2 R Squared

Multiple R squared is reported through `summary()`.

By definition, Multiple R squared is defined as the square of the correlation between fitted value and real values.

```
cor(job$salary,jobfit$fitted.values)
```

```
[1] 0.9002721
```

### 12.2.3 Model Comparison

Compare the model with the "zero" model, so-called the null model. The null model has no IV but intercept, which is the mean. The null model uses the mean to predict the data or explain the variation of values. Get the null model by formula `DV~1`.

```
jobfit2 = lm(salary ~ 1,data = job)
```

Then use `anova(model1,model2)` to compare the two models.

F test for the MLR model is to test against the null hypothesis that all coefficients of IVs are zero. Therefore, the model comparison by essence coincides with the F test!

```
anova(jobfit,jobfit2)
```

```
Analysis of Variance Table

Model 1: salary ~ educ + salbegin + jobtime + prevexp
Model 2: salary ~ 1
  Res.Df        RSS Df   Sum of Sq      F    Pr(>F)
1    469 2.6137e+10
```

```
2    473 1.3792e+11 -4 -1.1178e+11 501.45 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

## 12.3 Standard MLR

Transform the formula with each variable embedded into the function `scale()`, no other adjustments!

Under standard MLR, the coefficients are standardized ones.

```
fit_std = lm(scale(salary) ~ scale(educ) + scale(salbegin) + scale(jobtime) + scale(pre
             ,data = job)
```

```
summary(fit_std)
```

```
Call:
lm(formula = scale(salary) ~ scale(educ) + scale(salbegin) +
    scale(jobtime) + scale(prevexp), data = job)

Residuals:
     Min       1Q   Median       3Q      Max
-1.73349 -0.24121 -0.07298  0.15475  2.69850

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)     -1.129e-16  2.008e-02   0.000        1
scale(educ)      1.132e-01  2.798e-02   4.045 6.11e-05 ***
scale(salbegin)  8.151e-01  2.707e-02  30.111  < 2e-16 ***
scale(jobtime)   9.515e-02  2.018e-02   4.715 3.19e-06 ***
scale(prevexp)  -1.060e-01  2.161e-02  -4.904 1.30e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.4372 on 469 degrees of freedom
Multiple R-squared:  0.8105,    Adjusted R-squared:  0.8089
F-statistic: 501.5 on 4 and 469 DF,  p-value: < 2.2e-16
```

## 12.4 Plots for Goodness of Fit

### 12.4.1 Fitted & Real Values

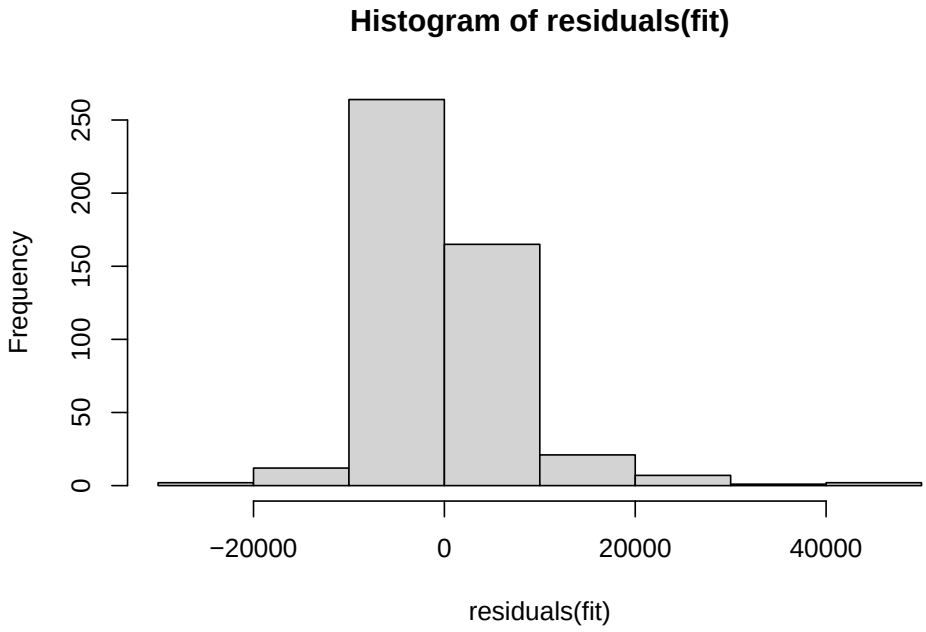
```
fitted_salary = fitted(jobfit) # predicted values
plot(job$salary,fitted_salary,'p')
```


### 12.4.2 Residuals

If the model fits well, the residuals should follow a normal distribution.

Fitted model's residuals can be obtained through `residuals(model)` or `model$residuals`.

```
hist(residuals(fit))
```




```
# hist(fit$residuals)
```

### 12.4.3 More Plots

Use `plot(model)` to get multiple plots reflecting the goodness of fit from many perspectives.

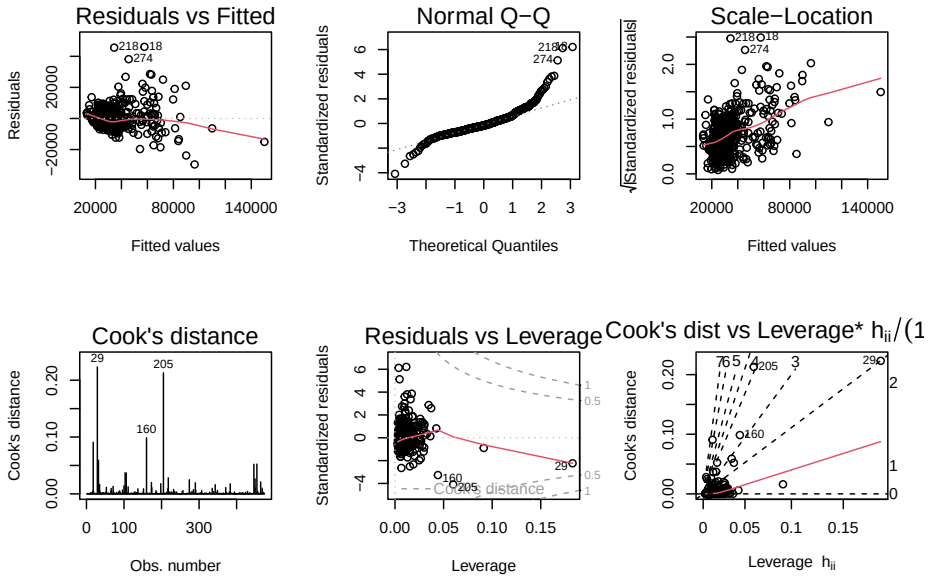
Specify parameter `which = vector_of_range` to designate the plots to show, 6 the most. `which = 1:4` by default.

Following the order, the six plots are:

- Residuals & Fitted Values
  - test the homoscedascitiy assumption
- QQ Plot for Residuals
  - test the Gaussian assumption
- Square Root of Standardized Residuals & Fitted Value
  - similar to the first one
- Cook's Distance

  - check outliers
- Standardized Residuals & Leverage
  - See if influential points have huge residual, if so, possibly outlier!
- Cook's Distance & Leverage
  - Large in both, possibly outlier!

```
par(mfrow = c(2,3))
plot(fit, which=1:6)
```




# 13 Moderation & Mediation

## 13.1 Moderation

The data in "`moderation_data.sav`" contains information about loyalty to marriage, relationship between partners, and their age. We're interested in the relationship between age and loyalty, and how the relationship in marriage moderates it.

```
marriage <- bruceR::import("moderation_data.sav")
```

```
##   Successfully imported: 25 obs. of 8 variables
```

```
attach(marriage)
```

Note that when an interaction term is introduced into the model, better to centralize the variable for convenience of interpretation! Even without centered independent variables, the significance output won't change. However, **VIF** suffers with uncentered data.

```
C_Relationship = Relationship - mean(Relationship)
C_age_con = age_con  - mean(age_con)
```

The NULL model here is without the interaction term, but to inspect `age` and `Relationship` through MLR.

```
model_0 <- lm(Loyalty ~ C_Relationship + C_age_con)
summary(model_0)
```

```
##
## Call:
## lm(formula = Loyalty ~ C_Relationship + C_age_con)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.2008 -0.7192  0.2991  0.8225  3.1450
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    11.63320    0.28539   40.76  < 2e-16 ***
## C_Relationship  0.81162    0.19277    4.21 0.000361 ***
## C_age_con      -0.02447    0.03949   -0.62 0.541850
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 1.427 on 22 degrees of freedom
## Multiple R-squared:  0.4496, Adjusted R-squared:  0.3996
## F-statistic: 8.985 on 2 and 22 DF,  p-value: 0.001404
```

### 13.1.1 Moderation Model

Then, add the interaction of moderator and predictor into the MLR model. (Moderation in essence is a MLR with interaction terms.)

```
model_moderation <- lm(Loyalty ~ C_Relationship*C_age_con)
summary(model_moderation)
```

```
##
## Call:
## lm(formula = Loyalty ~ C_Relationship * C_age_con)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.2137 -0.7480  0.2958  0.8216  2.3732
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)              1.162e+01  2.646e-01  43.900  < 2e-16 ***
## C_Relationship           7.550e-01  1.806e-01   4.181 0.000421 ***
## C_age_con                3.173e-04  3.837e-02   0.008 0.993481
## C_Relationship:C_age_con 5.626e-02  2.617e-02   2.150 0.043378 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.322 on 21 degrees of freedom
## Multiple R-squared:  0.5489, Adjusted R-squared:  0.4844
## F-statistic: 8.517 on 3 and 21 DF,  p-value: 0.0006803
```

After introducing the interaction term into our model, four things are of

great interest.

- Significance output of interaction term.
- Significance output of original terms in null model.
- Adjusted $R^2$ compared to null model.
- VIF, in case of multicollinearity.

  The improvement of $R^2$ indicates that, even after considering the added complexity of model, the explanatory power is greater than the null.

### 13.1.2 Model Comparison

The most natural impulse is to compare the moderation model against the null one. The difference lies in the interaction term. In fact, the output comes down right to the interaction term.

```
anova(model_0,model_moderation)
```

```
## Analysis of Variance Table
##
## Model 1: Loyalty ~ C_Relationship + C_age_con
## Model 2: Loyalty ~ C_Relationship * C_age_con
##   Res.Df    RSS Df Sum of Sq      F  Pr(>F)
## 1     22 44.796
## 2     21 36.715  1    8.0803 4.6217 0.04338 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

## 13.2 After Significant

After seeing the moderation (interaction) is significant, we need to check how it exerts its effect by **simple main effect**, because a significant mod-

eration term will affect your interpretation of main effect. We're to compute marginal means, where the data is divided into $3 \times 3$ categories using $Mean \pm SD$, i.e., with 3 $X$ levels and 3 $M$ levels.

```
library(emmeans)
m_Relationship<- mean(Relationship, na.rm = TRUE)
sd_Relationship<- sd(Relationship, na.rm = TRUE)
m_Age<- mean(age_con, na.rm = TRUE)
sd_Age<- sd(age_con, na.rm = TRUE)

# use un-centered data to see how the raw looks like in slope analysis.
model_moderation <- lm(Loyalty ~ Relationship*age_con)

# compute Estimated Marginal Means
emm <- emmeans(model_moderation,  ~ Relationship*age_con,
               cov.keep = 3,
               at = list(
                 age_con = c(m_Age-sd_Age, m_Age, m_Age+sd_Age),
                 Relationship = c(m_Relationship-sd_Relationship,
                                  m_Relationship,
                                  m_Relationship+sd_Relationship)),
               level = 0.95)
summary(emm)
```

```
##  Relationship age_con emmean    SE df lower.CL upper.CL
##          3.14    24.9  11.10 0.518 21    10.02     12.2
##          4.65    24.9  11.61 0.389 21    10.80     12.4
##          6.16    24.9  12.13 0.633 21    10.81     13.4
##          3.14    32.2  10.47 0.379 21     9.69     11.3
##          4.65    32.2  11.62 0.265 21    11.07     12.2
##          6.16    32.2  12.76 0.381 21    11.96     13.5
##          3.14    39.6   9.85 0.501 21     8.81     10.9
##          4.65    39.6  11.62 0.386 21    10.82     12.4
##          6.16    39.6  13.39 0.565 21    12.21     14.6
```

```
##
## Confidence level used: 0.95
```

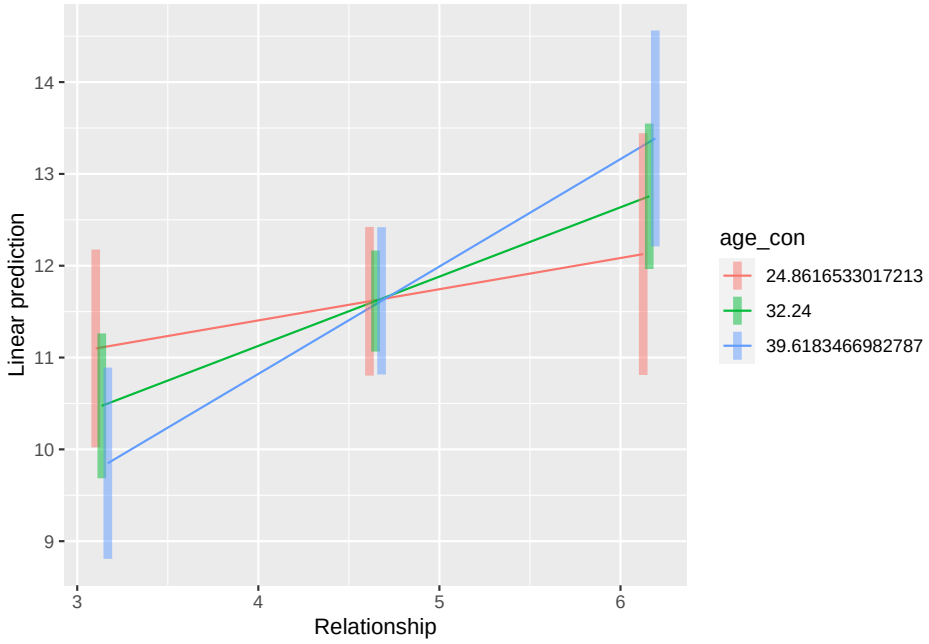
We then do the slope analysis, similar to simple main effect.

```
simpleSlope <- emtrends(model_moderation,
                        pairwise ~ age_con,
                        var = "Relationship",
                        cov.keep = 3,
                        at = list(
                          age_con = c(m_Age-sd_Age, m_Age, m_Age+sd_Age)),
                        level = 0.95)
simpleSlope
```

```
## $emtrends
##  age_con Relationship.trend    SE df lower.CL upper.CL
##     24.9              0.340 0.283 21   -0.249    0.928
##     32.2              0.755 0.181 21    0.379    1.130
##     39.6              1.170 0.244 21    0.662    1.678
##
## Confidence level used: 0.95
##
## $contrasts
##  contrast                                            estimate    SE df t.ratio
##  age_con24.8616533017213 - age_con32.24               -0.415 0.193 21  -2.150
##  age_con24.8616533017213 - age_con39.6183466982787    -0.830 0.386 21  -2.150
##  age_con32.24 - age_con39.6183466982787               -0.415 0.193 21  -2.150
##  p.value
##   0.1039
##   0.1039
##   0.1039
##
## P value adjustment: tukey method for comparing a family of 3 estimates
```

Slopes can be visualized.

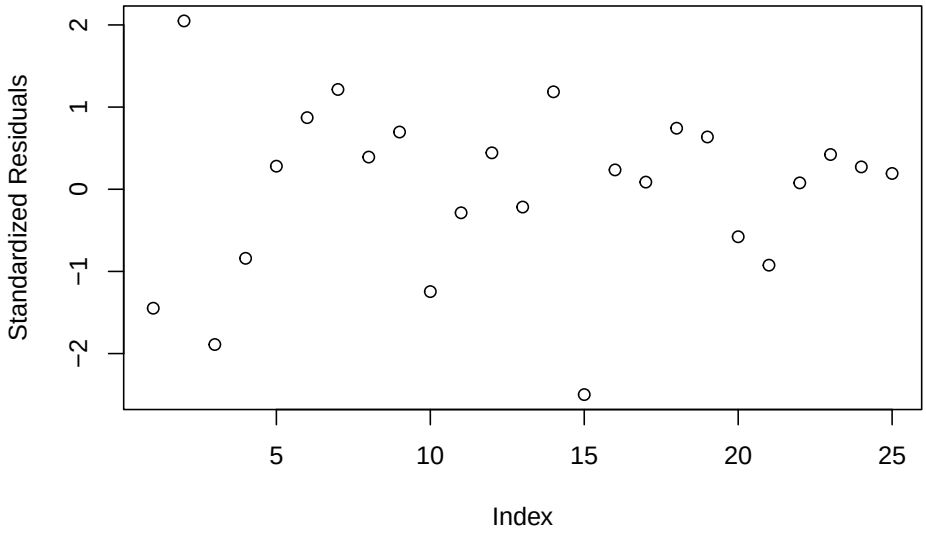
```
emmip(model_moderation,
      age_con ~ Relationship,
      cov.keep = 3,
      at = list(
        Relationship = c(m_Relationship-sd_Relationship,
                         m_Relationship,
                         m_Relationship+sd_Relationship),
        age_con = c(m_Age-sd_Age, m_Age, m_Age+sd_Age)),
      CIs = TRUE, level = 0.95, position = "jitter")
```



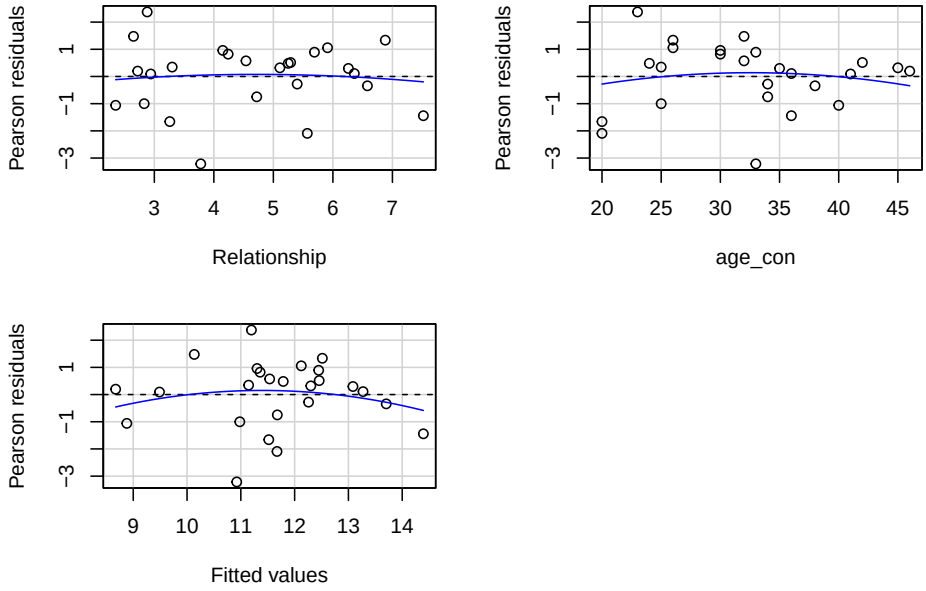

```
# emmip(emm)  # original Rmd had bare emm without formula — skipped
```

## 13.3 Residual Plots

```
res.std <- rstandard(model_moderation)
plot(res.std, ylab="Standardized Residuals")
```

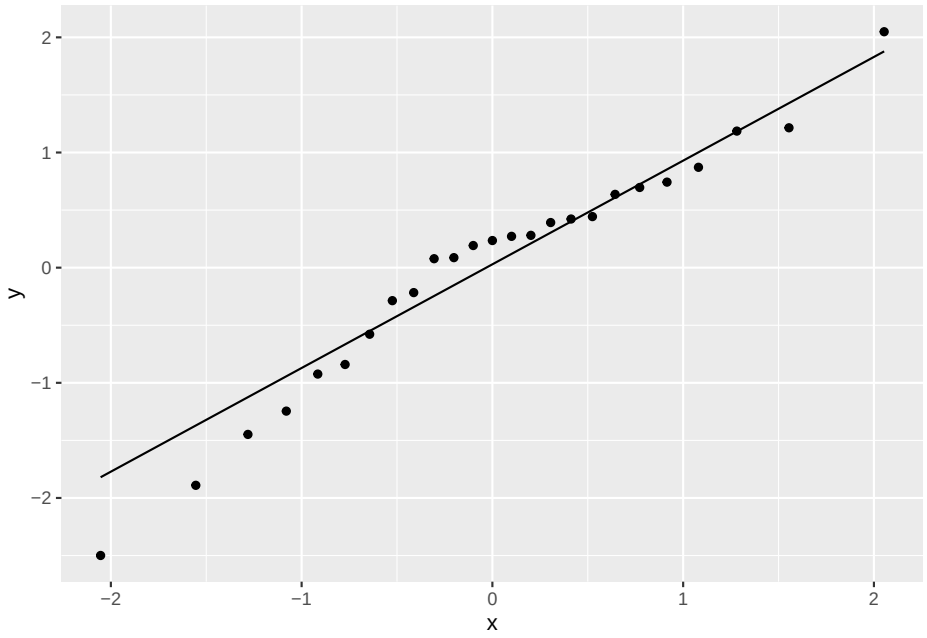



```
car::residualPlots(model_moderation)
```




```
##              Test stat Pr(>|Test stat|)
## Relationship   -0.2597           0.7977
## age_con        -0.5863           0.5642
## Tukey test     -1.0147           0.3102
```

```
library(ggplot2)
ggplot(as.data.frame(res.std), aes(sample = res.std)) +
  geom_qq() +
  geom_qq_line()
```




## 13.4 Mediation

Three steps are:

1. IV could predict DV.
2. IV could predict Mediator.
3. Mediator could predict DV on the presence of IV.

Besides, the predictive power of IV is then reduced,

- If IV is no longer a significant predictor → full mediation.
- If IV is still significant but reduced in magnitude → partial mediation.

Note that the effect of mediator on DV should primarily be based on IV!

The data in "mediation_data.sav" has information about customers' satisfaction, relationship with customers and discounts offered. We are interested in whether discount can mediate the influence of customer relationship on their satisfaction about some consumer goods.

```
consumer <- bruceR::import("mediation_data.sav")
```

```
##   Successfully imported: 40 obs. of 3 variables
```

## 13.5 Mediation Model

### 13.5.1 Direct Effect

Check without any consideration of mediation as a baseline model, which is the relationship between Y and X.

```
# Regress Y only on X
model_0 <- lm(Satisfaction ~ Relationship,data = consumer)
summary(model_0)
```

```
##
## Call:
## lm(formula = Satisfaction ~ Relationship, data = consumer)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.50746  0.02041  0.02041  0.02041  0.29254
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    2.6189     0.3069   8.534 2.30e-10 ***
## Relationship   0.4721     0.0651   7.252 1.12e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.1805 on 38 degrees of freedom
## Multiple R-squared:  0.5806, Adjusted R-squared:  0.5695
## F-statistic:  52.6 on 1 and 38 DF,  p-value: 1.123e-08
```

### 13.5.2 Internal Correlation

```
# M as a function of X
model_M <- lm(Discount ~ Relationship,data = consumer)
summary(model_M)
```

```
##
## Call:
## lm(formula = Discount ~ Relationship, data = consumer)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.56953 -0.00450  0.01718  0.01718  0.43047
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.91634    0.39477   7.388 7.40e-09 ***
## Relationship  0.41330    0.08374   4.935 1.62e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2322 on 38 degrees of freedom
## Multiple R-squared:  0.3906, Adjusted R-squared:  0.3746
## F-statistic: 24.36 on 1 and 38 DF,  p-value: 1.624e-05
```

Significant result here is the necessity for further mediation analysis. If not, no need to move forward.

### 13.5.3 Full Model

```
# Y as a function of both X and M
model_Y <- lm(Satisfaction ~ Relationship + Discount,data = consumer)
summary(model_Y)
```

```
##
## Call:
## lm(formula = Satisfaction ~ Relationship + Discount, data = consumer)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.192183  0.000099  0.007817  0.007817  0.143511
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.48089    0.16148   2.978   0.0051 **
## Relationship  0.16913    0.02811   6.016 5.97e-07 ***
## Discount      0.73313    0.04251  17.244  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06084 on 37 degrees of freedom
## Multiple R-squared:  0.9536, Adjusted R-squared:  0.9511
## F-statistic: 380.1 on 2 and 37 DF,  p-value: < 2.2e-16
```

Here, it's a case of partial mediation. And the adjusted $R^2$ is greatly improved, indicating that the mediation model is far more effective than the baseline model.

## 13.6 Mediation Effect

Mediation analysis is based on both the full model and the internal-correlation model. Important indicators measuring the mediation effect:

- ACME, average causal mediation effect, the indirect effect of X on Y.
- ADE: average direct effect.
- Total Effect: the total effect of X on Y, the sum of direct and indirect effect.
- Proportion Mediated: how much percentage of indirect effect within the total effect.

```
library(mediation)
mediation_results <- mediate(model_M, model_Y,
                             treat='Relationship',
                             mediator='Discount',
                             boot=TRUE, sims=500)
summary(mediation_results)
```

```
##
## Causal Mediation Analysis
##
## Nonparametric Bootstrap Confidence Intervals with the Percentile Method
##
##                Estimate 95% CI Lower 95% CI Upper p-value
## ACME              0.303        0.122         0.50  <2e-16 ***
## ADE               0.169        0.106         0.20   0.004 **
## Total Effect      0.472        0.297         0.65  <2e-16 ***
## Prop. Mediated    0.642        0.400         0.81  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Sample Size Used: 40
##
##
```

```
## Simulations: 500
```

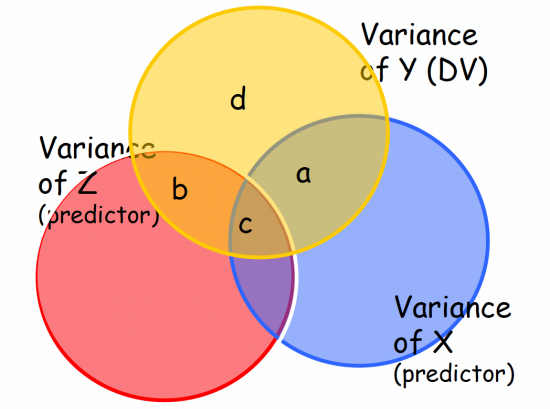
# 14 Correlation & Distance

## 14.1 Partial & Part Correlation

Zero-order correlation refers to the correlation between $X$ and $Y$ without controlling other covariates, such as Pearson and Spearman correlation.

We've discussed partial and part (or, semi-partial) correlation in multilinear regression. Take a step further, we can consider those under the most general cases, i.e., without the setting of MLR but some variables.

Suppose we have three variables, called $X, Y$ and $Z$, where $X$ is the predictor, $Y$ is the dependent variable, and $Z$ is the covariate. Noteworthy that the assumption of the roles of $X, Y, Z$ don't matter much, only for accommodating the setting.




Note that, in the general case, $Y$ is not represented as the whole area in the picture, but just an equal role as a circle.

If the correlation of $X$ and $Y$ is of our great interest,

$$r_x = \frac{a+c}{a+b+c+d}$$
$$pr_x = \frac{a}{a+d}$$
$$spr_x = \frac{a}{a+b+c+d}$$

- Partial: Holding $Z$ from BOTH $X$ and $Y$, i.e., removing the effect of $Z$.

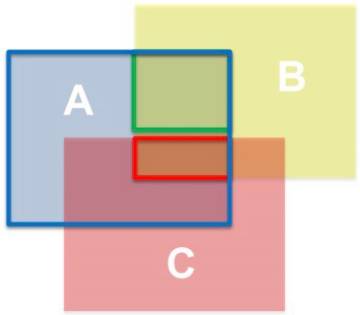
  Good for knowing the **variance uncounted by other control variable(s)**.

- Semi-Partial: Holding $Z$ from $X$ only.

  Good for knowing the **unique contribution of** $X$.

- Zero-Order: No extra constraint.

One more enlightening way to think about partial correlation is through multilinear regression.




If we're interested in the partial correlation between $A$ and $B$, with $C$ as the covariate. Then, what we need to do is partial out the effect of $C$ on BOTH $A$ and $B$, and this goal can be accomplished through step-by-step linear regression.

Specifically, first we conduct a simple linear regression of $A$ on $C$, and the residual part is that of $A$ which doesn't intersect with $C$. Similarly, then followed by a SLR of $B$ on $C$, same with the meaning of residual part. For

the two residual parts, they share it in common that they're clean of $C$. Lastly, compute the correlation between $A$ and $B$ directly, and that's what we want!

### 14.1.1 pcor() & spcor()

```
library(ppcor)
```

A popular radio talk show host has just received the latest government study on public health care funding and has uncovered a startling fact: As health care funding increases, disease rates also increase! Cities that spend more actually seem to be worse off than cities that spend less. Then, shall we just cut the funding and save people's health?

```
health <- import("health_funding.sav") %>% dplyr::select(-4)
```

```
##   Successfully imported: 50 obs. of 4 variables
```

First of all, check overall zero-order correlation.

```
cor(health)
```

```
##           funding   disease    visits
## funding 1.0000000 0.7370609 0.9643374
## disease 0.7370609 1.0000000 0.7619660
## visits  0.9643374 0.7619660 1.0000000
```

It seems astonishing that "`funding`" is greatly correlated with "`disease`"! May it offer the plausible reason to cut the funding? Remember, we still have the covariate "`visits`". One **alternative explanation** might be that, offered with more funding, people may be more likely to pay visits to health care, and thus more diseases are detected, which should have been detected but not due to wealth state.

```
pcor(health)$estimate
```

```
##             funding    disease    visits
```

```
## funding 1.00000000 0.01323488 0.9201361
## disease 0.01323488 1.00000000 0.2861809
## visits  0.92013611 0.28618088 1.0000000
```

It's astonishing that, considering the partial correlation, "`funding`" almost has no correlation with "`disease`" in essence!

Additionally, we can consider the part correlation.

```
spcor(health)$estimate
```

```
##             funding     disease    visits
## funding 1.000000000 0.003502953 0.6218522
## disease 0.008571132 1.000000000 0.1934086
## visits  0.595895776 0.075745172 1.0000000
```

Note that part correlations from rows differ from those from columns.

## 14.2 Distance

We need to distinguish between two kinds of distance. One is distance between **cases** (or observations), the other is distance between **variables**. There are traditionally three ways to measure distance, either from the perspective of dissimilarity or similarity.

- Dissimilarity
  - Euclidean distance

$$d(\vec{p}, \vec{q}) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

  - Chebychev distance

    A vector space where the distance between two vectors is the greatest of their differences along any coordinate dimension.

$$D(\vec{x}, \vec{y}) = \max_i |x_i - y_i|$$

- Similarity

- Cosine similarity
  - A measure of similarity between two non-zero vectors of an inner product space, a judgment of orientation but not magnitude.

```
climate = import("data10-04.sav", sheet = 1) %>%
  dplyr::select(hgrow,temp,rain,hsun,humi)
```

```
##   Successfully imported: 12 obs. of 6 variables
```

### 14.2.1 Correlation

Note that *correlation itself is a measure of distance*. However, correlation only applies for **distance between variables**, not for distance between cases. (If you insist to test the correlation between cases, easy, just transpose your data!)

```
cor(climate)
```

```
##           hgrow      temp      rain        hsun        humi
## hgrow 1.0000000 0.9833871 0.7093700  0.70442944  0.37357335
## temp  0.9833871 1.0000000 0.7148209  0.69049008  0.29198319
## rain  0.7093700 0.7148209 1.0000000  0.70184182  0.38432585
## hsun  0.7044294 0.6904901 0.7018418  1.00000000 -0.05093789
## humi  0.3735734 0.2919832 0.3843259 -0.05093789  1.00000000
```

### 14.2.2 Euclidean Distance

Then, we're interested in Euclidean distance **between cases** by default. Use "`dist(x, method = "euclidean")`". Mind that the distance makes sense only if the data is **scaled**!

```
# climate %>% dist()
climate  %>%
  scale() %>% dist()
```

If the Euclidean distance between **variables** is of interest, then *transpose* your data.

```
climate  %>%
  scale() %>% t() %>% dist()
```

It's noteworthy that, the Euclidean distance result returned by "`dist()`" is a unique type of "`dist`", which is incompatible with some other types. Luckily, use "`as.matrix()`" to transform that into a matrix (with the diagonal and upper right triangle filled automatically).

### 14.2.3 Cosine Similarity

Cosine similarity is for **between cases**. Also, if similarity between variables is of interest, be flexible with the data.

"`lsa::cosine()`" deals with cosine similarity **between variables** (column vectors) by default.

```
climate %>%
  as.matrix.data.frame() %>% t() %>%
  lsa::cosine()
```

# 15 PCA

```
library(psych)
```

The Big-Five scale "bfi" data is used here.

```
bfi = bfi %>% dplyr::select(1:25) %>% drop_na()
```

## 15.1 Prerequisites

### 15.1.1 Bartlett Test of Sphericity for Correlation Matrix

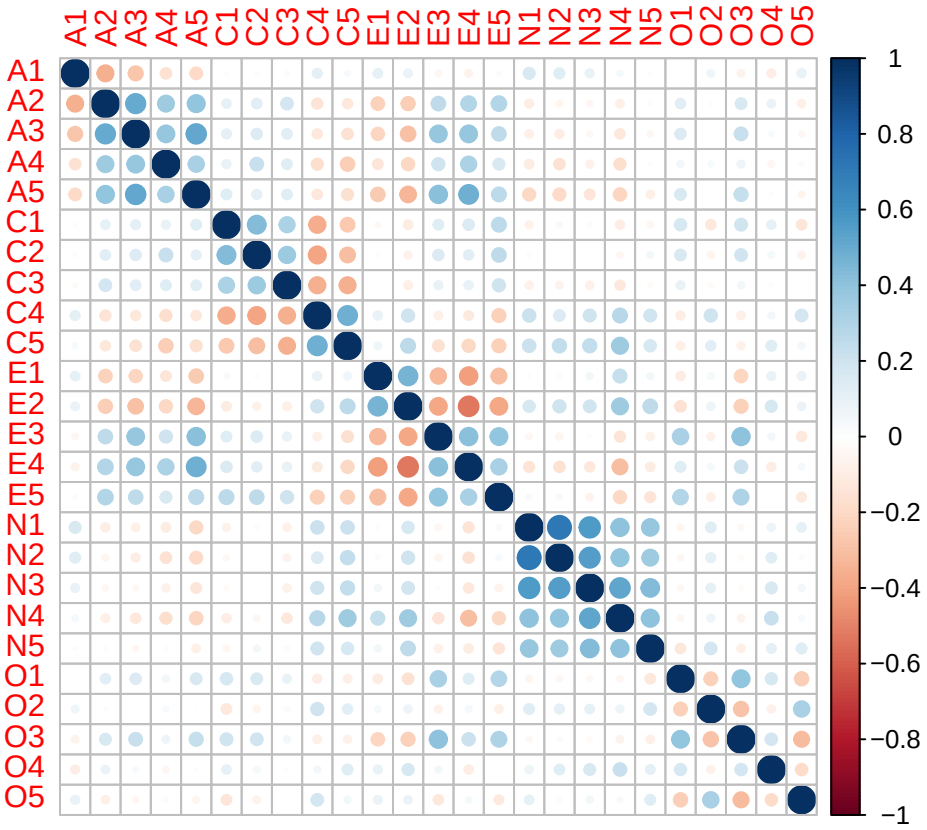
Firstly, generate a correlation matrix. Then, compare it against the identity matrix. Significant result is desired.

```
bfi_cor = cor(bfi)
cortest.bartlett(bfi_cor,n = nrow(bfi_cor))
```

```
## $chisq
## [1] 110.9584
##
## $p.value
## [1] 1
##
## $df
## [1] 300
```

Note that **reversely-coded** items are critical for explanatory factor analysis. If any, make sure they are detected and correctly-coded. To check this, you can view through correlation matrix plot, or through reliability analysis ("`reliability()`" will inform you of reversely-coded items.)

```
corrplot::corrplot(bfi_cor)
```




```
jmv::reliability(data = bfi,vars = colnames(bfi))
```

```
##
##  RELIABILITY ANALYSIS
##
##  Scale Reliability Statistics
##
##              Cronbach's
##
##    scale       0.5246608
##
##    Note. items 'A1',
##    'C4', 'C5', 'E1',
##    'E2', 'N1', 'N2',
```

```
##    'N3', 'N4', 'N5',
##    'O2', 'O4', and 'O5'
##    correlate negatively
##    with the total scale
##    and probably should be
##    reversed
```

### 15.1.2 KMO for Sampling Adequacy

0.8 and higher are great, 0.7 is acceptable, 0.6 is mediocre, less than 0.5 is unacceptable.

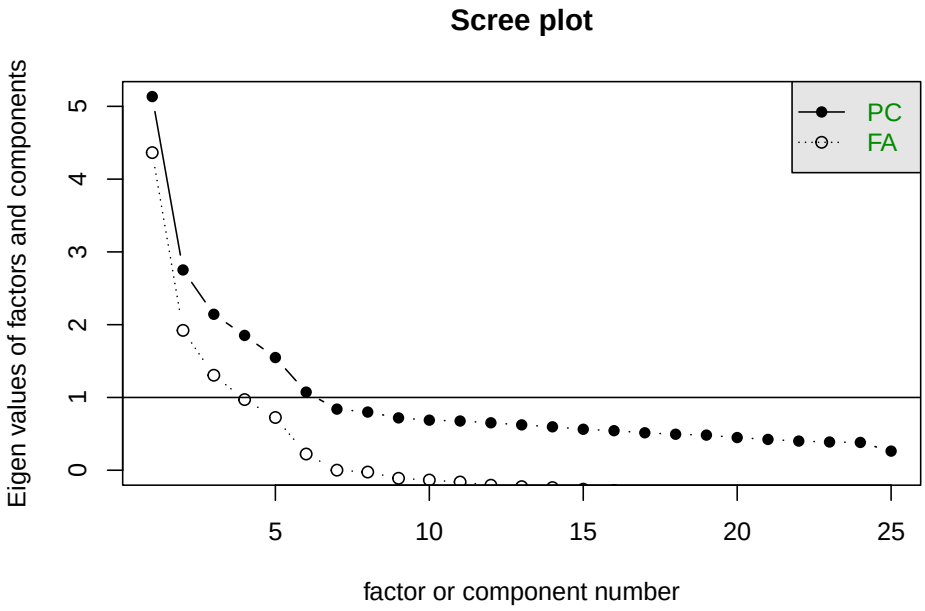
```
KMO(bfi_cor)
```

```
## Kaiser-Meyer-Olkin factor adequacy
## Call: KMO(r = bfi_cor)
## Overall MSA =  0.85
## MSA for each item =
##   A1   A2   A3   A4   A5   C1   C2   C3   C4   C5   E1   E2   E3   E4   E5   N1
## 0.75 0.84 0.87 0.88 0.90 0.84 0.80 0.85 0.83 0.86 0.84 0.88 0.90 0.88 0.89 0.78
##   N2   N3   N4   N5   O1   O2   O3   O4   O5
## 0.78 0.86 0.89 0.86 0.86 0.78 0.84 0.77 0.76
```

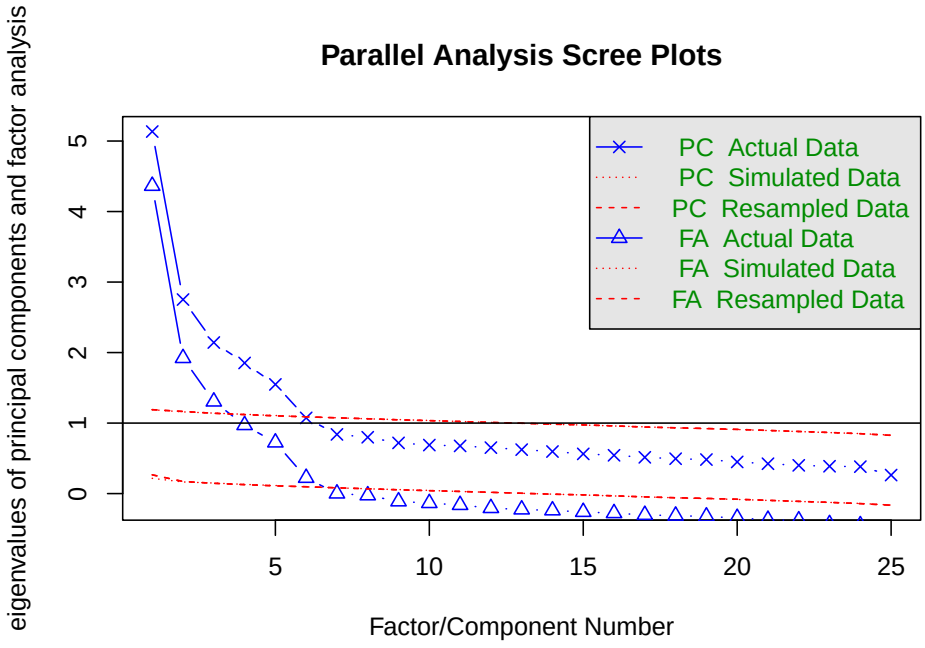
## 15.2 No. of Factors

- To determine the number of factors, scree and parallel are two methods.
- Scree is obsolete; parallel suggests 3 factors here.

```
scree(bfi)
```




```
fa.parallel(bfi)
```




```
## Parallel analysis suggests that the number of factors =  6  and the number of compon
```

## 15.3 Factor Analysis

Use "`psych::fa(r, nfactors, rotate, fm)`".

### 15.3.1 NULL Model

Take a look: without selecting the number of factors, without rotation.

The Proportion Var and Cumulative Var are the variance that can be explained by factors.

"SS loadings" is the sum of squared loadings for each factor; **>1** means the factor is worth keeping.

The model can be printed with the following results:

- Standardized loadings (pattern matrix) based upon correlation matrix
  - SS loadings
    - The eigenvalues, the sum of the squared loadings for each factor.
    - **Better to be >1**.
  - Proportion Var
    - The overall variance (for all variables/items) that can be explained by each factor.
  - Cumulative Var
  - Proportion Explained
    - The relative importance of each factor.
  - Cumulative Proportion
- Data frame with loading matrix, communality, uniqueness, and item complexity.

**Standardized loadings (pattern matrix) based upon correlation matrix** is the most important part!

```
fa_none = fa(bfi,nfactors = ncol(bfi),
             fm='minres',rotate='none')
```

```
fa_none
```

```
## Factor Analysis using method =  minres
## Call: fa(r = bfi, nfactors = ncol(bfi), rotate = "none", fm = "minres")
## Standardized loadings (pattern matrix) based upon correlation matrix
##      MR1   MR2   MR3   MR4   MR5   MR6   MR7   MR8   MR9  MR10  MR11  MR12
## A1 -0.23 -0.02  0.14 -0.01 -0.42  0.31  0.07 -0.06  0.02  0.00  0.00  0.04
## A2  0.48  0.30 -0.19  0.15  0.38 -0.22  0.14  0.15 -0.11  0.03 -0.04  0.06
## A3  0.54  0.31 -0.25  0.12  0.29  0.00  0.11 -0.13  0.10 -0.02 -0.04 -0.13
## A4  0.43  0.12 -0.12  0.30  0.17  0.02  0.00 -0.22 -0.01  0.24  0.00  0.15
## A5  0.59  0.19 -0.27  0.06  0.18  0.14  0.01 -0.05  0.16 -0.14  0.06 -0.10
## C1  0.35  0.14  0.47  0.10 -0.02  0.13 -0.08  0.18  0.18 -0.02 -0.03  0.03
## C2  0.34  0.20  0.52  0.27  0.02  0.19 -0.09  0.05  0.01  0.24 -0.04 -0.10
## C3  0.32  0.06  0.37  0.30  0.02  0.02  0.06  0.12 -0.11 -0.18  0.03 -0.02
## C4 -0.48  0.11 -0.48 -0.20  0.03  0.27  0.09  0.04 -0.09  0.05  0.03  0.04
## C5 -0.51  0.16 -0.32 -0.29  0.13  0.09 -0.01  0.26  0.14  0.15 -0.08 -0.09
## E1 -0.42 -0.19  0.30  0.12  0.30  0.23  0.23 -0.15  0.06 -0.05 -0.10  0.02
## E2 -0.62 -0.04  0.24  0.06  0.33  0.11  0.12 -0.01  0.03  0.03  0.19 -0.03
## E3  0.53  0.33 -0.13 -0.19 -0.09  0.20  0.03 -0.10 -0.10 -0.09  0.02 -0.07
## E4  0.61  0.18 -0.34  0.11 -0.21  0.15 -0.15  0.03  0.15 -0.03  0.02  0.14
## E5  0.52  0.30  0.10 -0.06 -0.24 -0.03  0.23  0.16 -0.12  0.05 -0.07  0.00
## N1 -0.44  0.64  0.04  0.09 -0.27 -0.14  0.13 -0.11  0.02  0.03  0.04 -0.02
## N2 -0.43  0.65  0.10  0.04 -0.22 -0.21  0.16  0.01  0.15  0.00  0.09 -0.01
## N3 -0.41  0.62  0.04  0.06 -0.02 -0.02 -0.15 -0.10  0.01 -0.09 -0.14  0.06
## N4 -0.53  0.42  0.08 -0.04  0.23  0.06 -0.14 -0.02 -0.07 -0.07 -0.23 -0.02
## N5 -0.35  0.44 -0.02  0.24  0.13  0.06 -0.27  0.02 -0.16  0.00  0.18 -0.03
## O1  0.33  0.20  0.19 -0.38  0.02  0.16  0.12 -0.03 -0.09  0.00 -0.02  0.08
## O2 -0.20  0.07 -0.29  0.41 -0.05  0.17  0.09  0.16 -0.02 -0.01  0.03  0.04
## O3  0.41  0.29  0.13 -0.46  0.03  0.12 -0.04 -0.05 -0.06  0.07  0.09 -0.08
## O4 -0.07  0.26  0.18 -0.24  0.32  0.08  0.01  0.12  0.05 -0.08  0.09  0.19
## O5 -0.21 -0.05 -0.26  0.45 -0.13  0.21  0.07  0.05 -0.08 -0.03 -0.04 -0.04
##      MR13  MR14  MR15  MR16  MR17  MR18  MR19  MR20  MR21  MR22  MR23  MR24 MR25
## A1  0.04  0.18  0.06 -0.02 -0.05  0.01 -0.01  0.04  0.01  0.03  0.00 -0.01    0
```

```
## A2  0.08  0.06  0.03 -0.08 -0.01  0.06  0.00  0.02  0.04 -0.01  0.00  0.00    0
## A3  0.01 -0.04  0.01  0.05 -0.09  0.04 -0.06  0.01 -0.03  0.03  0.00 -0.01    0
## A4  0.00  0.06  0.00  0.07 -0.01 -0.04  0.03  0.01 -0.01  0.00  0.02  0.00    0
## A5  0.00  0.10 -0.04 -0.03  0.06 -0.10  0.02 -0.02  0.01  0.00 -0.01  0.01    0
## C1  0.08 -0.01 -0.12  0.01 -0.11  0.03 -0.02  0.00  0.00 -0.02  0.02  0.01    0
## C2 -0.06 -0.05  0.07 -0.03  0.03 -0.01  0.01 -0.03 -0.01  0.00 -0.02  0.00    0
## C3  0.10  0.02  0.15  0.08 -0.01 -0.05  0.03 -0.01 -0.01 -0.01  0.01  0.00    0
## C4  0.12 -0.04  0.00 -0.04 -0.08 -0.03  0.03 -0.04 -0.04 -0.02  0.00  0.00    0
## C5  0.01  0.06  0.05  0.07  0.03 -0.01  0.00 -0.01  0.03  0.00  0.01  0.00    0
## E1  0.02 -0.06 -0.01 -0.13  0.03  0.00  0.00 -0.02  0.03  0.00  0.01  0.00    0
## E2  0.00  0.08 -0.07  0.08  0.04  0.07  0.03  0.03 -0.02 -0.02 -0.02  0.00    0
## E3 -0.15  0.01  0.00  0.03 -0.01  0.06  0.03 -0.01  0.01 -0.04  0.02 -0.01    0
## E4  0.07 -0.04  0.03 -0.05  0.10  0.05 -0.01  0.02 -0.01 -0.02 -0.01 -0.01    0
## E5 -0.04  0.05 -0.16 -0.03  0.04 -0.04  0.03  0.00 -0.02  0.02  0.00 -0.01    0
## N1  0.06 -0.08 -0.02  0.05 -0.01 -0.08 -0.05  0.02  0.04 -0.03 -0.01  0.00    0
## N2 -0.02  0.00  0.07 -0.06  0.05  0.05  0.03 -0.02 -0.03  0.01  0.02  0.01    0
## N3 -0.02  0.04 -0.02  0.04 -0.06  0.05  0.06 -0.04  0.02  0.01 -0.03  0.00    0
## N4 -0.01  0.02  0.01 -0.02  0.06 -0.03 -0.02  0.06 -0.04 -0.01  0.01  0.00    0
## N5  0.03  0.04 -0.06 -0.07  0.00  0.00 -0.05 -0.02  0.01  0.02  0.02  0.00    0
## O1  0.01  0.03  0.02  0.06  0.07  0.03 -0.10 -0.04  0.00  0.00 -0.01  0.01    0
## O2 -0.19 -0.03  0.05 -0.04 -0.06 -0.02 -0.03  0.03  0.00  0.00 -0.01  0.01    0
## O3  0.07 -0.10  0.03 -0.05 -0.02  0.00  0.05  0.06  0.02  0.02 -0.01  0.01    0
## O4 -0.09 -0.09  0.01  0.05  0.00 -0.05  0.01  0.01  0.01  0.03  0.01 -0.01    0
## O5  0.06 -0.12 -0.06  0.10  0.08  0.05  0.02  0.00  0.02  0.03  0.01  0.00    0
##      h2   u2 com
## A1 0.39 0.61 3.6
## A2 0.66 0.34 5.0
## A3 0.62 0.38 3.7
## A4 0.48 0.52 4.6
## A5 0.59 0.41 2.8
## C1 0.50 0.50 3.7
## C2 0.63 0.37 4.0
## C3 0.45 0.55 4.9
```

```
## C4 0.64 0.36 3.6
## C5 0.63 0.37 4.4
## E1 0.58 0.42 6.0
## E2 0.65 0.35 2.5
## E3 0.56 0.44 3.2
## E4 0.69 0.31 3.0
## E5 0.57 0.43 3.6
## N1 0.77 0.23 2.7
## N2 0.78 0.22 2.8
## N3 0.63 0.37 2.3
## N4 0.62 0.38 3.2
## N5 0.54 0.46 4.7
## O1 0.40 0.60 4.6
## O2 0.41 0.59 4.3
## O3 0.55 0.45 3.8
## O4 0.36 0.64 6.1
## O5 0.43 0.57 3.8
##
##                        MR1  MR2  MR3  MR4  MR5  MR6  MR7  MR8  MR9 MR10 MR11
## SS loadings           4.74 2.40 1.70 1.35 1.11 0.62 0.39 0.35 0.24 0.24 0.20
## Proportion Var        0.19 0.10 0.07 0.05 0.04 0.02 0.02 0.01 0.01 0.01 0.01
## Cumulative Var        0.19 0.29 0.35 0.41 0.45 0.48 0.49 0.51 0.52 0.53 0.53
## Proportion Explained  0.34 0.17 0.12 0.10 0.08 0.04 0.03 0.02 0.02 0.02 0.01
## Cumulative Proportion 0.34 0.50 0.63 0.72 0.80 0.84 0.87 0.89 0.91 0.93 0.94
##                       MR12 MR13 MR14 MR15 MR16 MR17 MR18 MR19 MR20 MR21 MR22
## SS loadings           0.16 0.13 0.12 0.10 0.09 0.07 0.05 0.03 0.02 0.01 0.01
## Proportion Var        0.01 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
## Cumulative Var        0.54 0.54 0.55 0.55 0.56 0.56 0.56 0.56 0.56 0.56 0.57
## Proportion Explained  0.01 0.01 0.01 0.01 0.01 0.01 0.00 0.00 0.00 0.00 0.00
## Cumulative Proportion 0.95 0.96 0.97 0.98 0.99 0.99 0.99 1.00 1.00 1.00 1.00
##                       MR23 MR24 MR25
## SS loadings           0.00 0.00 0.00
## Proportion Var        0.00 0.00 0.00
```

```
## Cumulative Var        0.57 0.57 0.57
## Proportion Explained  0.00 0.00 0.00
## Cumulative Proportion 1.00 1.00 1.00
##
## Mean item complexity =  3.9
## Test of the hypothesis that 25 factors are sufficient.
##
## df null model =  300  with the objective function =  7.48 with Chi Square =  18146.0
## df of  the model are -25  and the objective function was  0
##
## The root mean square of the residuals (RMSR) is  0
## The df corrected root mean square of the residuals is  NA
##
## The harmonic n.obs is  2436 with the empirical chi square  0  with prob <  NA
## The total n.obs was  2436  with Likelihood Chi Square =  0  with prob <  NA
##
## Tucker Lewis Index of factoring reliability =  1.017
## Fit based upon off diagonal values = 1
## Measures of factor score adequacy
##                                                  MR1  MR2 MR3  MR4  MR5  MR6
## Correlation of (regression) scores with factors   0.96 0.94 0.9 0.86 0.86 0.77
## Multiple R square of scores with factors          0.93 0.88 0.8 0.73 0.73 0.60
## Minimum correlation of possible factor scores     0.85 0.75 0.6 0.47 0.47 0.20
##                                                  MR7   MR8   MR9  MR10  MR11
## Correlation of (regression) scores with factors   0.71  0.67  0.62  0.59  0.59
## Multiple R square of scores with factors          0.50  0.44  0.38  0.35  0.34
## Minimum correlation of possible factor scores     0.00 -0.11 -0.24 -0.30 -0.31
##                                                  MR12  MR13  MR14  MR15  MR16
## Correlation of (regression) scores with factors   0.51  0.47  0.45  0.43  0.42
## Multiple R square of scores with factors          0.26  0.22  0.20  0.18  0.17
## Minimum correlation of possible factor scores     -0.48 -0.56 -0.60 -0.64 -0.65
##                                                  MR17  MR18  MR19  MR20  MR21
## Correlation of (regression) scores with factors   0.39  0.36  0.27  0.21  0.19
```

```
## Multiple R square of scores with factors          0.15  0.13  0.07  0.05  0.04
## Minimum correlation of possible factor scores    -0.70 -0.74 -0.85 -0.91 -0.93
##                                                   MR22  MR23  MR24 MR25
## Correlation of (regression) scores with factors   0.15  0.10  0.05    0
## Multiple R square of scores with factors          0.02  0.01  0.00    0
## Minimum correlation of possible factor scores    -0.95 -0.98 -1.00   -1
```

For illustration purpose:

- If using 25 factors (here only 25 items), we can see how loading and proportion of variance decrease with factors;
- The cumulative variance explained will approach 1.

Scores, or the coefficient for the model: $Factors = scores * X + \varepsilon$ where $X$ are the original data, $Factors$ are the obtained latent variables, $scores$ are their coefficients.

```
head(fa_none$scores)
```

```
##             MR1        MR2        MR3        MR4        MR5         MR6
## [1,] -0.9626360 -0.9588661 -1.4054098  0.4759941 -0.8015550 -0.43231916
## [2,] -0.1390885  0.1084315 -0.6963273 -0.2378928 -0.5391072  0.05982153
## [3,] -0.3348634  0.4224894  0.3075130 -0.3589328 -0.8904569 -0.09589599
## [4,] -0.6125530 -0.4145437 -1.1996091  0.3418352 -0.2270343  0.98903364
## [5,] -0.1559778 -0.5521422 -0.2218671  0.2608480 -1.0992961  0.16307467
## [6,]  1.1302479  0.7788624  0.7855226  0.2839649 -0.9010212  0.03915730
##             MR7         MR8        MR9       MR10       MR11        MR12
## [1,]  0.4218875  0.33649546 -0.1776481  0.2520133  0.5236582  0.02549219
## [2,] -1.5204589  0.08899538 -0.4270758 -0.8301846 -0.4759909 -1.51574630
## [3,]  0.2469924  0.07177995  0.6214508  0.5948036  0.4869344 -0.58456895
## [4,]  1.2248993 -0.31491131  0.9663665  0.4122710 -0.7407253 -0.99481003
## [5,] -0.9427302 -0.24649398 -0.6914795 -0.7686449 -0.4167629 -0.59352826
## [6,]  0.4974892  0.64567163  0.2967707  0.2815885  0.3121439  0.11768794
##             MR13        MR14         MR15        MR16         MR17        MR18
## [1,] -0.65557522 -0.16413109  0.150318429 -0.51002543  0.100235548 -0.27031025
## [2,] -0.08241697 -0.08283261 -0.001249183 -0.04638787 -0.295174589 -0.03465558
```

```
## [3,] -0.14606000  0.45539975  0.220009474  0.27571184 -0.181852697  0.18798816
## [4,]  0.19190363 -0.09312211  0.257157316 -0.42900268 -0.041377111  0.29575845
## [5,] -0.45669848  0.14139336 -0.278256908 -0.13307298 -0.001875009 -0.30444354
## [6,] -0.37216733  0.73609907  0.701941763 -0.43639326 -0.428401601 -0.06485289
##            MR19         MR20        MR21        MR22         MR23         MR24
## [1,]  0.1362685 -0.082227379 -0.03603757 -0.01820091 -0.008749894  0.009569002
## [2,] -0.3614771 -0.074749964 -0.03486963 -0.14930817  0.164475662  0.002198851
## [3,]  0.1632381  0.057890828  0.09244331  0.20478642 -0.036201997 -0.066984670
## [4,]  0.4032549 -0.004110559 -0.38316119  0.18234705  0.210808208  0.029328607
## [5,]  0.6039341 -0.125366947 -0.21772004 -0.07448912  0.039062175  0.033158942
## [6,]  0.1335250  0.137412524  0.09439241  0.08965242  0.250201940 -0.044897119
##               MR25
## [1,] -4.469031e-15
## [2,] -6.221992e-15
## [3,]  1.227396e-15
## [4,]  4.065705e-15
## [5,] -5.304509e-15
## [6,]  4.235866e-15
```

Communality, the variance that can be explained by factors, which should be high if the model is good.

```
# communality
apply(fa_none$loadings^2,1,sum)
```

```
##        A1        A2        A3        A4        A5        C1        C2        C3
## 0.3943710 0.6632439 0.6199889 0.4787753 0.5939516 0.4954832 0.6324677 0.4484558
##        C4        C5        E1        E2        E3        E4        E5        N1
## 0.6362078 0.6263201 0.5758485 0.6532971 0.5583746 0.6854708 0.5666877 0.7738224
##        N2        N3        N4        N5        O1        O2        O3        O4
## 0.7798513 0.6340109 0.6203578 0.5405542 0.4042214 0.4110534 0.5488778 0.3609073
##        O5
## 0.4338612
```

```
# uniqueness
1 - apply(fa_none$loadings^2,1,sum)
```

```
##        A1        A2        A3        A4        A5        C1        C2        C3
## 0.6056290 0.3367561 0.3800111 0.5212247 0.4060484 0.5045168 0.3675323 0.5515442
##        C4        C5        E1        E2        E3        E4        E5        N1
## 0.3637922 0.3736799 0.4241515 0.3467029 0.4416254 0.3145292 0.4333123 0.2261776
##        N2        N3        N4        N5        O1        O2        O3        O4
## 0.2201487 0.3659891 0.3796422 0.4594458 0.5957786 0.5889466 0.4511222 0.6390927
##        O5
## 0.5661388
```

### 15.3.2 Reduced Model

According to scree plot and SS loadings for factors, 5 is suggested. Note that restrictions on factor number will have practical effect on the model, instead of just trimming the model to the reduced one.

```
fa5_none = fa(bfi,5,rotate='none')
fa5_none
```

```
## Factor Analysis using method =  minres
## Call: fa(r = bfi, nfactors = 5, rotate = "none")
## Standardized loadings (pattern matrix) based upon correlation matrix
##      MR1   MR2   MR3   MR4   MR5   h2   u2 com
## A1 -0.22 -0.02  0.13  0.00 -0.37 0.20 0.80 1.9
## A2  0.47  0.29 -0.18  0.13  0.34 0.46 0.54 3.1
## A3  0.53  0.30 -0.25  0.11  0.30 0.54 0.46 2.9
## A4  0.42  0.11 -0.12  0.27  0.16 0.30 0.70 2.5
## A5  0.58  0.18 -0.26  0.04  0.17 0.47 0.53 1.8
## C1  0.34  0.13  0.45  0.12 -0.02 0.35 0.65 2.2
## C2  0.34  0.20  0.48  0.27  0.02 0.45 0.55 2.9
## C3  0.32  0.05  0.35  0.31  0.03 0.32 0.68 3.0
## C4 -0.47  0.10 -0.45 -0.21  0.03 0.48 0.52 2.5
```

```
## C5 -0.49  0.15 -0.29 -0.28  0.11 0.44 0.56 2.6
## E1 -0.41 -0.18  0.26  0.11  0.26 0.35 0.65 3.1
## E2 -0.62 -0.04  0.23  0.06  0.32 0.55 0.45 1.9
## E3  0.53  0.33 -0.12 -0.18 -0.09 0.44 0.56 2.2
## E4  0.60  0.17 -0.33  0.10 -0.19 0.54 0.46 2.0
## E5  0.51  0.29  0.09 -0.04 -0.23 0.41 0.59 2.1
## N1 -0.44  0.64  0.03  0.10 -0.27 0.68 0.32 2.2
## N2 -0.42  0.62  0.08  0.05 -0.20 0.61 0.39 2.1
## N3 -0.41  0.61  0.03  0.07 -0.02 0.54 0.46 1.8
## N4 -0.53  0.42  0.08 -0.04  0.22 0.51 0.49 2.3
## N5 -0.35  0.41 -0.03  0.21  0.12 0.35 0.65 2.7
## O1  0.33  0.20  0.20 -0.36  0.01 0.32 0.68 3.2
## O2 -0.20  0.06 -0.29  0.37 -0.04 0.27 0.73 2.6
## O3  0.41  0.30  0.15 -0.45  0.01 0.47 0.53 3.0
## O4 -0.06  0.26  0.18 -0.23  0.30 0.25 0.75 3.7
## O5 -0.20 -0.06 -0.27  0.41 -0.11 0.30 0.70 2.5
##
##                        MR1  MR2  MR3  MR4  MR5
## SS loadings           4.60 2.27 1.55 1.22 0.96
## Proportion Var        0.18 0.09 0.06 0.05 0.04
## Cumulative Var        0.18 0.27 0.34 0.39 0.42
## Proportion Explained  0.43 0.21 0.15 0.12 0.09
## Cumulative Proportion 0.43 0.65 0.79 0.91 1.00
##
## Mean item complexity =  2.5
## Test of the hypothesis that 5 factors are sufficient.
##
## df null model =  300  with the objective function =  7.48 with Chi Square =  18146.0
## df of  the model are 185  and the objective function was  0.64
##
## The root mean square of the residuals (RMSR) is  0.03
## The df corrected root mean square of the residuals is  0.04
##
```

```
## The harmonic n.obs is  2436 with the empirical chi square  1131.9  with prob <  1.1e
## The total n.obs was  2436  with Likelihood Chi Square =  1538.52  with prob <  8.6e-
##
## Tucker Lewis Index of factoring reliability =  0.877
## RMSEA index =  0.055  and the 90 % confidence intervals are  0.052 0.057
## BIC =  95.87
## Fit based upon off diagonal values = 0.98
## Measures of factor score adequacy
##                                                  MR1  MR2  MR3  MR4  MR5
## Correlation of (regression) scores with factors   0.95 0.91 0.85 0.81 0.80
## Multiple R square of scores with factors          0.90 0.83 0.73 0.66 0.64
## Minimum correlation of possible factor scores     0.80 0.66 0.46 0.33 0.28
```

### 15.3.3 Factor Rotation

Factor loadings are listed for each variable. The square of it is the variance accounted for. Unrotated factors are typically not very interpretable (most factors are correlated with many variables), though it has largest variance explained. The hope is that all these correlation coefficients are close to (+/-)1 or 0, so we can easily interpret the meanings of these components (factors). We will accomplish this goal through rotation.

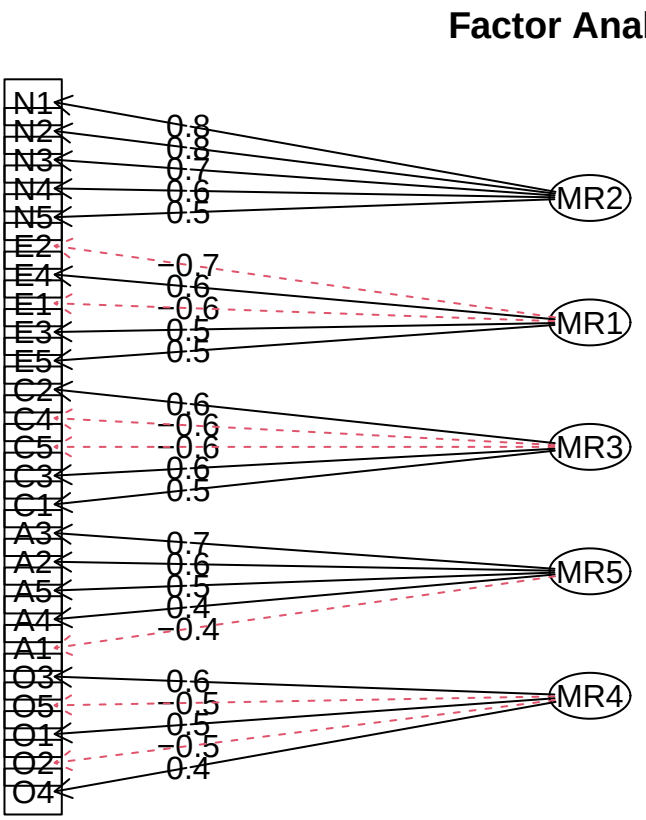
Factors are rotated to make them more meaningful and easier to interpret (each variable is associated with a minimal number of factors). The most popular rotational method is Varimax rotations, which uses orthogonal rotations yielding uncorrelated factors/components. Varimax attempts to maximize the variance that a single factor can account for (across all variables). This enhances the interpretability of the factors. (i.e., kind of simplification of interpretation) However, the loadings and complexity for each variable will change, while communality and uniqueness won't change; SS loadings for each factors must change, while cumulative variance accounted for will not change.

```
fa5_rot = fa(bfi,nfactors = 5,rotate = 'varimax')
```

### 15.4 Factor Diagram

Use "`fa.diagram()`" to show how variables relate to the factors. Note `fa.diagram()` use r = 0.3 as a cut off, you can change it.

```
fa.diagram(fa5_rot)
```




# 16 Reliability Analysis

Use `reliability()` function in package "`jmv`" to conduct reliability analysis. Note that "`psych`" package has a `reliability()` function too. don't

load the wrong package.

```
library("jmv")
```

The following data is from a scale designed for athletes. However, in reliability analysis, the practical meanings of items don't matter that much compared to factor analysis.

```
athletes = import("data15-01.xlsx") %>%
  dplyr::select(c('x1','x2','x4','x10','x13','x39','x41','x45'))
```

```
##   Successfully imported: 312 obs. of 54 variables
```

## 16.1 Reliability

In `reliability()`, the most important parameters are:

- `data`: the data as a data frame
- `vars`: a vector of strings *naming* the variables of interest in data
- `alphaScale`: TRUE (default) or FALSE, provide Cronbach's alpha
- `corPlot`: TRUE or FALSE (default), provide a correlation plot
- `revItems`: a vector containing strings naming the varibales that are reverse scaled
- `alphaItems`: TRUE or FALSE (default), provide what the Cronbach's alpha would be if the item was dropped
- `itemRestCor`: TRUE or FALSE (default), provide item-rest correlations

Other parameters:

- `meanItems`: TRUE or FALSE (default), provide item means
- `sdItems`: TRUE or FALSE (default), provide item standard deviations
- `omegaScale`: TRUE or FALSE (default), provide McDonald's omega
- `omegaItems`: TRUE or FALSE (default), provide what the McDonald's omega would be if the item was dropped
- `meanScale`: TRUE or FALSE (default), provide the mean

- `sdScale`: TRUE or FALSE (default), provide the standard deviation

```
RA_results = reliability(athletes,
                         vars = c('x1','x2','x4','x10','x13','x39','x41','x45'),
                         meanScale=TRUE,
                         sdScale=TRUE,
                         corPlot=TRUE,
                         alphaItems=TRUE,
                         meanItems=TRUE,
                         sdItems=TRUE,
                         itemRestCor=TRUE)
```

For reversely-coded items, the `reliability()` will return with a "Note" to inform you of those, if any. Those reversely-coded items can also be detected through `Corplot`, which will correlate negatively with right-ordered ones.

The `reliability()` function returns an object with

- `items`: Item Reliability Statistics, with mean, sd, item-rest correlation, (if-removed) Cronbach's $\alpha$, etc. Results on item-level.
- `scale`: Scale Reliability Statistics, with mean, sd, Cronbach's $\alpha$, etc. Results on scale-level.

```
RA_results$items
```

```
##
##  Item Reliability Statistics
##
##           mean        sd           item-rest correlation    Cronbach's
##
##    x1     4.019231    0.8819093              0.101491299     0.102859793
##    x2     3.416667    1.0786340              0.009362630     0.155679950
##    x4     3.092949    1.1202671              0.117490345     0.081986674
##    x10    3.548077    1.0837574              0.246701216    -0.008887225
##    x13    3.663462    1.2077769              0.015286062     0.154828393
##    x39    2.759615    1.3761533             -0.057467371     0.219073269
```

```
##    x41    3.028846    1.1658876              0.016813511     0.152730592
##    x45    2.336538    1.4408637              0.030066097     0.148840877
##
```

```
RA_results$scale
```

```
##
##  Scale Reliability Statistics
##
##             mean        sd           Cronbach's
##
##    scale    3.233173    0.4466260       0.1439260
##
##    Note. items 'x39', 'x41', and 'x45'
##    correlate negatively with the total scale
##    and probably should be reversed
```

```
RA_results_rv = reliability(athletes,
                            vars = c('x1','x2','x4','x10','x13','x39','x41','x45'),
                            meanScale=TRUE,sdScale=TRUE,corPlot=TRUE,alphaItems=TRUE,
                            meanItems=TRUE,sdItems=TRUE,itemRestCor=TRUE,
                            revItems=c('x39','x41','x45'))
```

## 16.2 Nonadditivity Test

Nonadditivity test paves the way for reliability test, which checks whether the item factor interact with the subject factor. Nonadditivity test is conducted by `agricolae::nonadditivity()`.

```
library(agricolae)
```

The data for nonadditivity test should be in **long-format**.

```
athletes_long = athletes %>% mutate(x39 = 6-x39,
                        x41 = 6-x41,
                        x45= 6-x45) %>%
```

```
  pivot_longer(cols = everything(),
                    names_prefix = 'x',
                    names_to = 'item',
                    values_to = 'score') %>%
  mutate(subj = rep(1:nrow(athletes),each = ncol(athletes)))
```

For nonadditivity test, first construct a model without interaction term. Then, the `nonadditivity()` receives four parameters.

- `y`: Answer of the experimental unit, which is the score for each item.
- `factor1`: **subject** factor (first treatment applied to each experimental unit).
- `factor2`: **item** factor (second treatment applied to each experimental unit).
- `df`: Degrees of freedom of the experimental error
- `MSerror`: Means square error of the experimental

```
model_nonadd = lm(score ~ subj+as.factor(item),athletes_long)
MSerror = deviance(model_nonadd)/df.residual(model_nonadd)
res =  with(athletes_long, nonadditivity(score,subj,item, df.residual(model_nonadd), MS
```

```
##
## Tukey's test of nonadditivity
## score
##
## P : -55.72362
## Q : 330.7249
##
## Analysis of Variance Table
##
## Response: residual
##                 Df Sum Sq Mean Sq F value   Pr(>F)
## Nonadditivity    1    9.4  9.3888  6.7735 0.009307 **
## Residuals     2486 3445.9  1.3861
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

## 16.3 ICC

Note that package "psych" is used here. ICC is also available in other packages ("psy", "irr"), but the functions would be used quite differently.

```
library(psych)
```

Note that whenever you use the data to carry reliability analysis, the reversely-coded items should be corrected.

In ICC(), specify parameter check.keys = TRUE to reverse those items that do not correlate with total score (with alerting message and reversed items). Note that this is not done by default.

```
ICC_results = ICC(athletes,check.keys = T)
```

```
## Some items were negatively correlated with total scale and were automatically revers
##  This is indicated by a negative sign for the variable name.

##  reversed items
##    x39 x41 x45
```

ICC() can be designated to an object, and the most important returned values are:

- results: A matrix of 6 rows and 8 columns, including the ICCs, $F$ test, $p$ values, and confidence limits
- stats: The anova statistics (converted from lmer if using lmer)

Types of ICC are:

- ICC1: single random factor
- ICC2: two-way random factor
- ICC3: two-way mixed factor

```
ICC_results$results
```

```
##                           type       ICC        F df1  df2           p
## Single_raters_absolute   ICC1 0.1030658 1.919272 311 2184 7.30497e-17
## Single_random_raters     ICC2 0.1115472 2.097898 311 2177 1.26596e-21
## Single_fixed_raters      ICC3 0.1206760 2.097898 311 2177 1.26596e-21
## Average_raters_absolute ICC1k 0.4789692 1.919272 311 2184 7.30497e-17
## Average_random_raters   ICC2k 0.5011019 2.097898 311 2177 1.26596e-21
## Average_fixed_raters    ICC3k 0.5233324 2.097898 311 2177 1.26596e-21
##                          lower bound upper bound
## Single_raters_absolute    0.07308939   0.1382257
## Single_random_raters      0.08104023   0.1471864
## Single_fixed_raters       0.08910136   0.1575172
## Average_raters_absolute   0.38681209   0.5620130
## Average_random_raters     0.41366008   0.5799578
## Average_fixed_raters      0.43900148   0.5993178
```

```
ICC_results$stats  #the ANOVA results
```

```
##           subjects       Judges    Residual
## df    3.110000e+02 7.000000e+00 2177.000000
## SumSq 8.002308e+02 2.578900e+02 2670.108979
## MS    2.573089e+00 3.684144e+01    1.226508
## F     2.097898e+00 3.003765e+01          NA
## p     1.265960e-21 7.229148e-40          NA
```

# 17 Survival Analysis

```
library(survival)
library(survminer)
library(ggfortify)
```

Dataset “lung” in package “survival” is of interest, which contains the in-

formation about survival in patients with advanced lung cancer from the North Central Cancer Treatment Group. Variables include:

- time: Survival time in days
- status: censoring status `1=censored`, `2=dead`
- sex: `Male=1 Female=2`

```
attach(lung)
```

## 17.1 Survival Model

### 17.1.1 Survival Object

`survival::Surv()` creates a survival object, with censored cases marked with "+" following up censoring time and death cases simply showing the death time. The result is usually used as a response variable in a model formula. Two most important parameters:

- `time` :For right censored data, this is the follow up time. For interval data, the first argument is the starting time for the interval.
- `event`: The status indicator, normally `0=alive`, `1=dead`. Other choices are `TRUE/FALSE` (`TRUE = death`) or `1/2` (`2=death`).

```
survobj = Surv(time, status==2)
```

### 17.1.2 Overall Model

Use `survival::survfit(formula,data)` to fit the model. For the *overall* survival analysis, just specify the null hypothesis with `Surv~1`, i.e., NO groups.

```
sfit <- survfit(Surv(time, status==2)~1, data=lung)
```

The fitted survival model itself will tell you the results, with observations, events, median and 95% CI.

```
sfit
```

```
## Call: survfit(formula = Surv(time, status == 2) ~ 1, data = lung)
##
##        n events median 0.95LCL 0.95UCL
## [1,] 228    165    310     285     363
```

If more details are needed, i.e., the instantaneous events and estimates of survival, std.err, 95% CI, just `summary(sfit)`. Moreover, if `time` is designated, then the `summary()` function will return the *estimated* results.

```
summary(sfit,time = 200)
```

```
## Call: survfit(formula = Surv(time, status == 2) ~ 1, data = lung)
##
##  time n.risk n.event survival std.err lower 95% CI upper 95% CI
##   200    144      72     0.68  0.0311        0.622        0.744
```

### 17.1.3 Grouped Model

If grouping is of interest, specify it in the formula with `Surv~group`.

```
sfit_sex = survfit(Surv(time,status==2)~sex)
sfit_sex
```

```
## Call: survfit(formula = Surv(time, status == 2) ~ sex)
##
##         n events median 0.95LCL 0.95UCL
## sex=1 138    112    270     212     310
## sex=2  90     53    426     348     550
```

It's ideal that the 95% CI of groups doesn't intersect, and a significant result can be obtained instantly. However, that's often not the case; take a step further and use Cox regression to see exactly.

## 17.2 Cox Regression

```
coxph(Surv(time, status==2)~sex)
```

```
## Call:
## coxph(formula = Surv(time, status == 2) ~ sex)
##
##        coef exp(coef) se(coef)      z       p
## sex -0.5310    0.5880   0.1672 -3.176 0.00149
##
## Likelihood ratio test=10.63  on 1 df, p=0.001111
## n= 228, number of events= 165
```

where "`exp(coef)`" is the HR (hazard ratio), which measures the relative risk between two groups.

Theoretically, it's better only with binary variables in Cox regression. However, for categorical variables with more than 2 levels, use "`as.factor(cat_var)`" with "`model = TRUE`" to generate corresponding dummy variables. For continuous variables, they'll be included into the regression model as usual, but the model will be plotted with expected value of continuous variables.

The simpler version of such comparison can be accomplished by "`survival::survdiff()`". Both of the functions have the same formula. From "`survdiff()`", we can get the intermediate results when computing the log rank, or the chi-square test. However, this wouldn't return the coefficient. Jointly speaking, the combined results of two functions emphasize the fact that the log rank test (named Mantel-Haenszel test), is a chi-square test in essence.

```
surv_diff = survdiff(Surv(time, status==2)~sex)
surv_diff
```

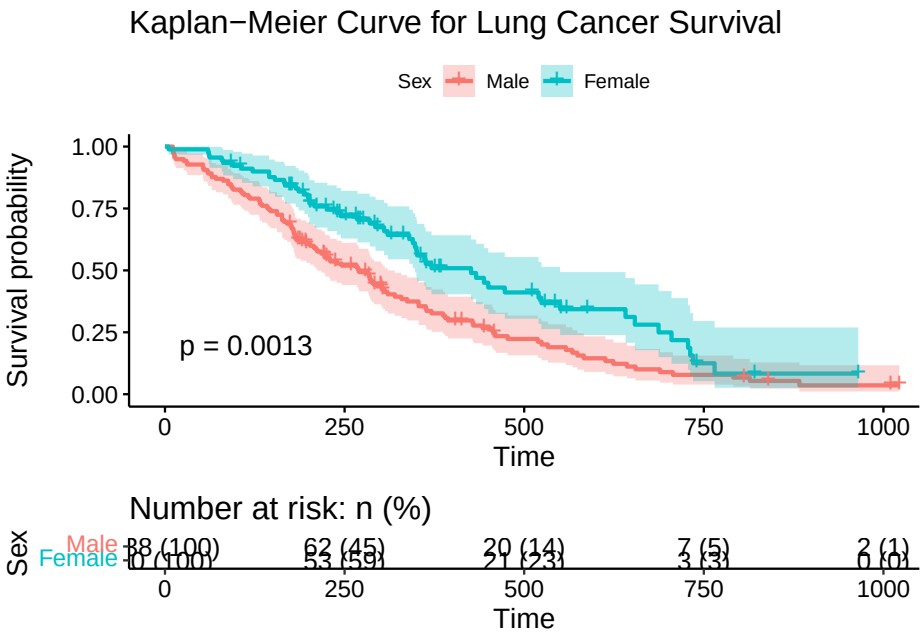
```
## Call:
## survdiff(formula = Surv(time, status == 2) ~ sex)
```

```
##
##         N Observed Expected (O-E)^2/E (O-E)^2/V
## sex=1 138      112     91.6      4.55      10.3
## sex=2  90       53     73.4      5.68      10.3
##
##  Chisq= 10.3  on 1 degrees of freedom, p= 0.001
```

## 17.3 Surv Plots

Use "`ggsurvplot(model,data)`" to visualize the survival analysis. Note that if the "`data`" wasn't passed into `survfit()`, an error will be raised.

```
ggsurvplot(sfit_sex,data = lung,
           conf.int=TRUE, pval=TRUE,
           risk.table='abs_pct', # risk table below the Kaplan-Meier curve(s)
           legend.labs=c("Male", "Female"), legend.title="Sex",
           title="Kaplan-Meier Curve for Lung Cancer Survival")
```

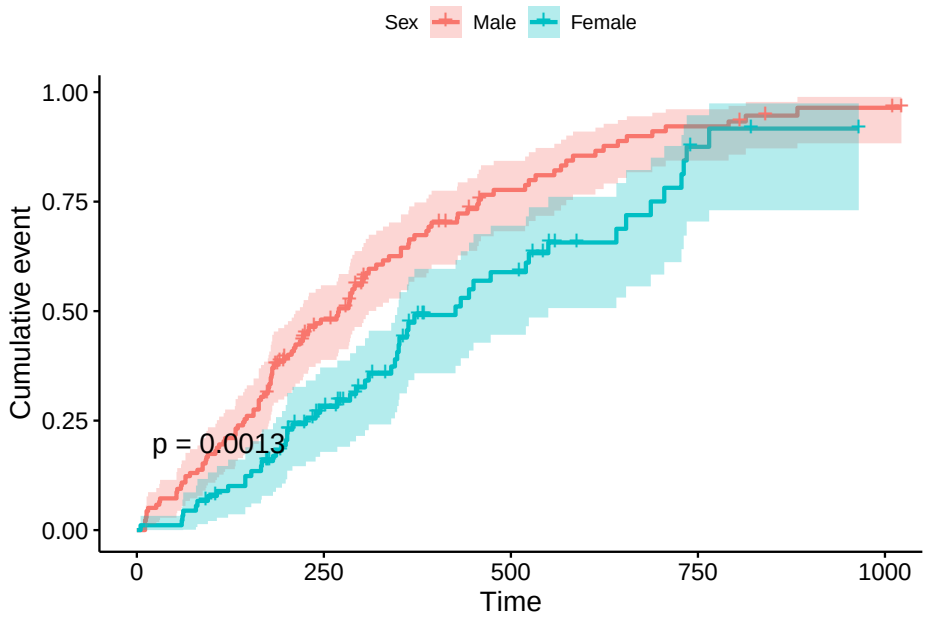



Moreover, the parameter "`fun`" is quite useful, which will define a transfor-

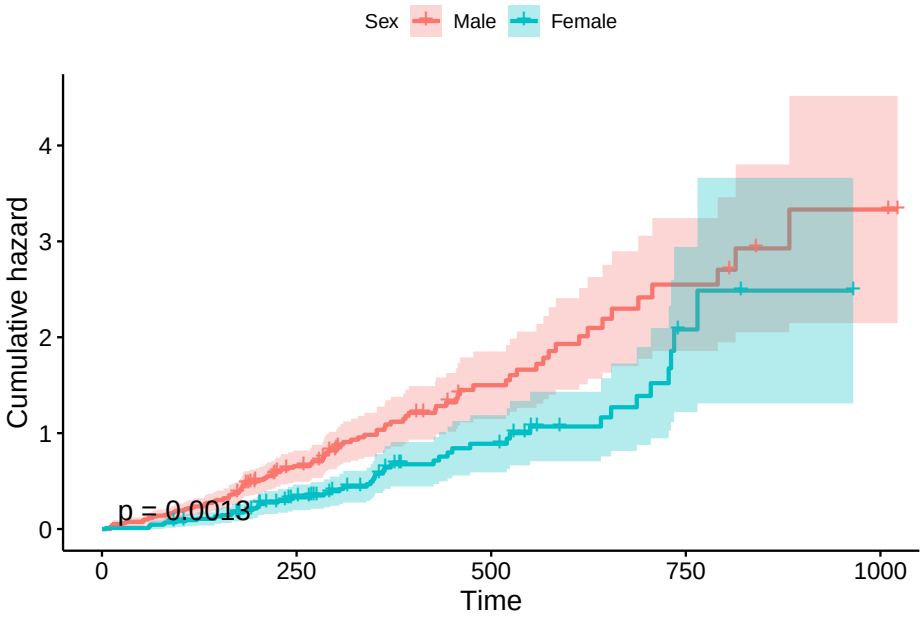
mation of the survival curve.

- "pct": for survival probability in percentage, **by default**.
- "event": plots cumulative events, $f(y) = 1 - y$.
- "cumhaz": plots the cumulative hazard function, $f(y) = -\log(y)$.

```
ggsurvplot(sfit_sex,data = lung,
           conf.int=TRUE, pval=TRUE,
           # risk.table='abs_pct',
           legend.labs=c("Male", "Female"), legend.title="Sex",
           fun = 'event')
```




```
ggsurvplot(sfit_sex,data = lung,
           conf.int=TRUE, pval=TRUE,
           # risk.table='abs_pct',
           legend.labs=c("Male", "Female"), legend.title="Sex",
           fun = 'cumhaz')
```




# 18 Clustering

```
library(parameters)
library("factoextra")
```

## 18.1 Hierarchical Clustering

```
automobile = import("data13-01.xlsx") %>%
  dplyr::select(c('type','price','engine_s','horsepow',
           'wheelbas','width','length','curb_wgt',
           'fuel_cap','mpg')) %>% drop_na() %>% scale()
```

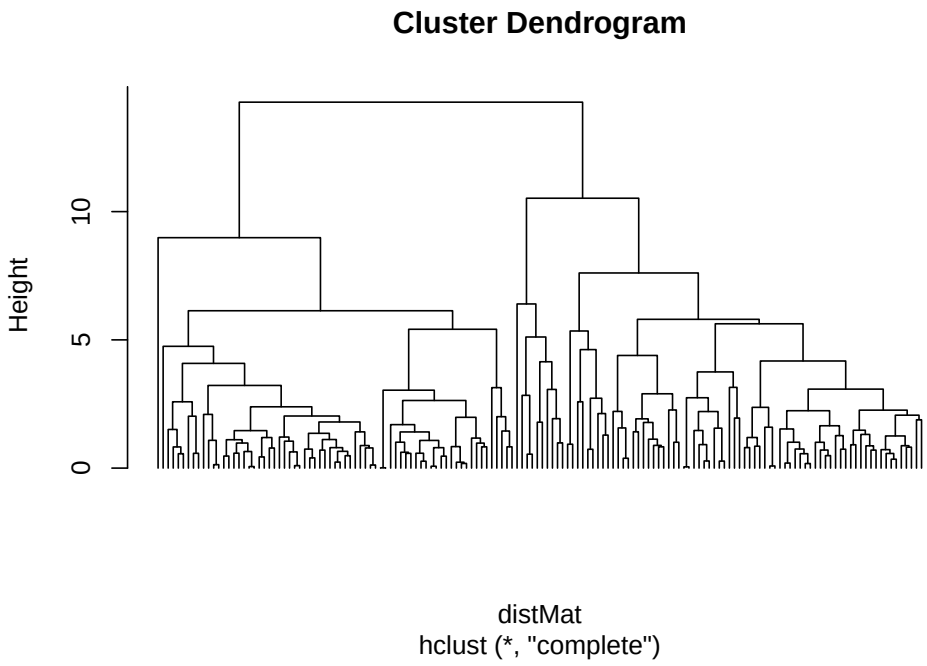
```
##   Successfully imported: 157 obs. of 27 variables
```

Use "`hclust()`" from base R and transmit a **dissimilarity** structure produced by "`dist()`".

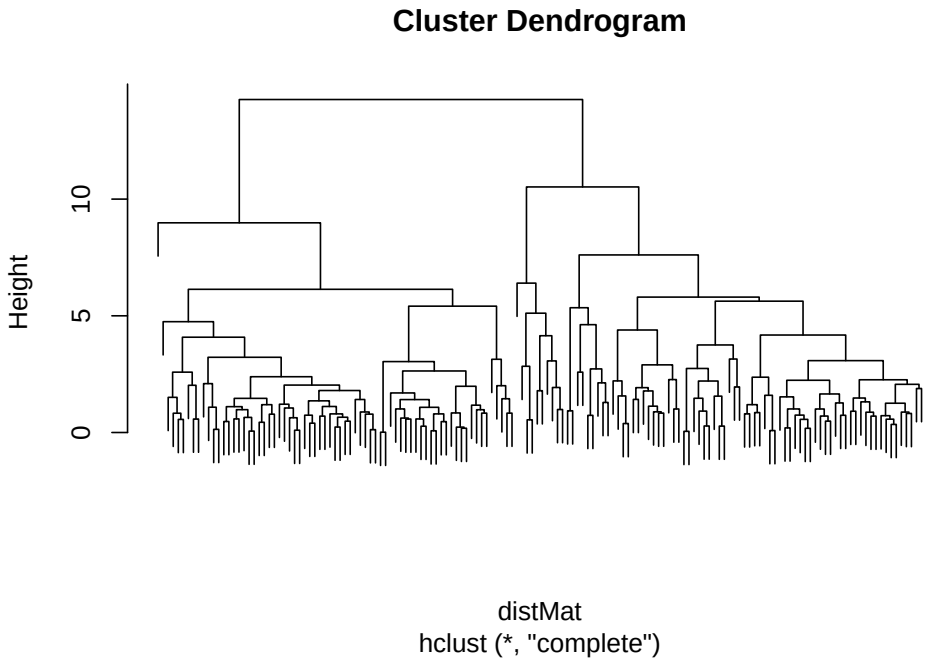
```
distMat = dist(automobile, method = "euclidean") # Euclidean distance matrix
hc = hclust(distMat)  # hierarchical clustering, using the default method
```

Use "`plot(hclust_obj)`" to visualize the hierarchical clustering. Note that you should specify "`hang = -1`" to ensure all nodes touch the same horizontal line and looks decent. If the sample isn't that large, set "`labels = TRUE`" (by default) to see the corresponding observations.

```
plot(hc,labels=FALSE,hang = -1)
```

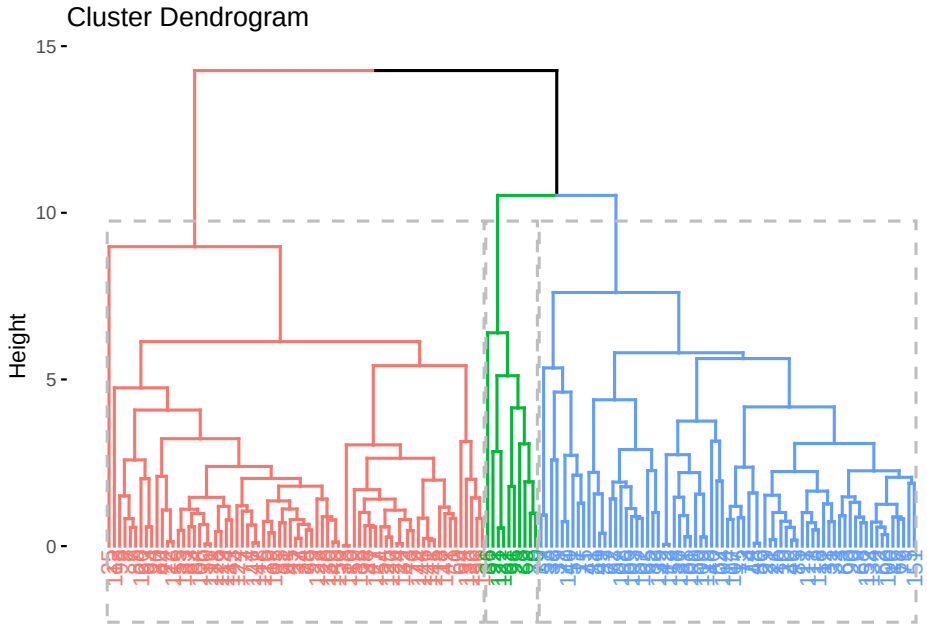



```
plot(hc,labels=FALSE)
```




Moreover, you can draw colored dendrogram with grouping.

```
fviz_dend(hc,k=3, rect = T)
```



Also, you can also check how many members for each cluster and make a frequency table.Note that the grouping might change if we change cluster number.

```
memberLabels = cutree(hc,3)
table(memberLabels)
```

```
## memberLabels
##  1  2  3
## 71 71 10
```

Then, we can get the centers of cluters by "`mean`" summary statistics. Use "`aggregate()`" to achieve that.

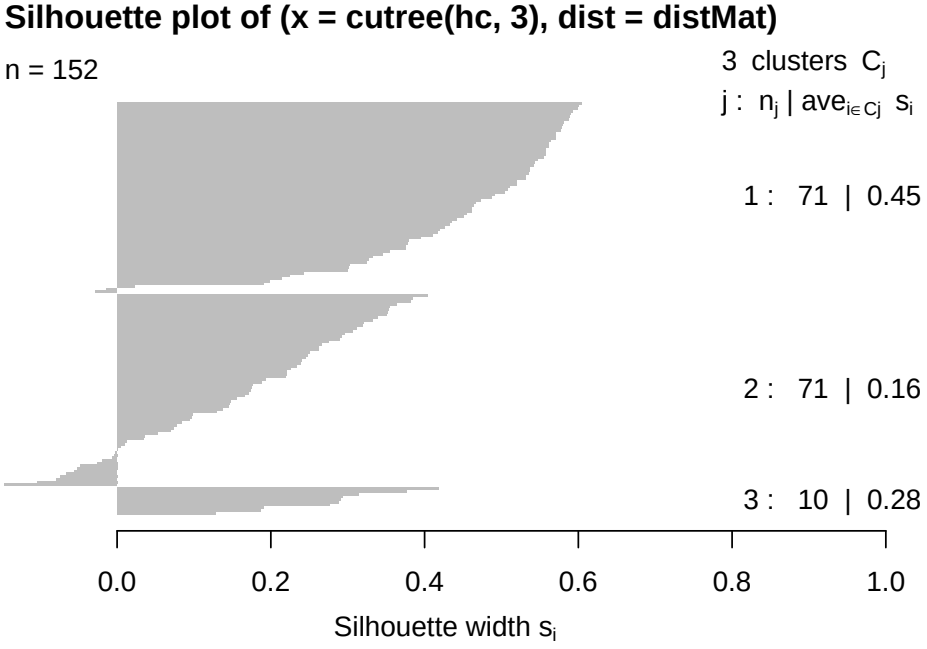
- `x`: usually a data frame.
- `by`: a list of grouping elements, each **as long** as the variables in the data frame x. The elements are coerced to factors before use.
- `FUN`: a function to compute the summary statistics which can be applied to all data subsets.

```
aggregate(automobile,list(memberLabels),mean)
```

```
##   Group.1       type      price   engine_s   horsepow   wheelbas      width
## 1       1 -0.4362472 -0.4985325 -0.7231739 -0.6579479 -0.6259158 -0.6459818
## 2       2  0.5201409  0.1193177  0.5017194  0.3293009  0.6601116  0.5979645
## 3       3 -0.5956452  2.6924245  1.5723270  2.3333938 -0.2427902  0.3409229
##        length   curb_wgt   fuel_cap        mpg
## 1 -0.65105171 -0.7853229 -0.7200528  0.6620751
## 2  0.65553921  0.7236353  0.6270241 -0.5592928
## 3 -0.03186123  0.4379815  0.6605042 -0.7297543
```

Moreover, we can use Silhouette plot to check whether some cases are outliers.

```
library(cluster)
plot(silhouette(cutree(hc,3), distMat))
```




## 18.2 Non-Hierarchical Clustering (K-Means)

### 18.2.1 Computation

```
athlete = import("data13-02.xlsx") %>% dplyr::select(-1)
```

```
##   Successfully imported: 10 obs. of 4 variables
```

- `x`: numeric matrix-like data
- `centers`: either the number of clusters, say kk, or a set of initial (distinct) cluster centres. If a number, a random set of (distinct) rows in x is chosen as the initial centres.
- `iter.max`: the maximum number of iterations allowed.
- `nstart`: if centers is a number, how many random sets should be chosen. Trying several random starts is often recommended.

```
kmean_results = kmeans(athlete, centers = 4,
                       iter.max = 10, nstart = 1)
```

`kmeans()` returns an object of class "`kmeans`", a list with at least the following components:

- `cluster` : A vector of integers (from 1:k) indicating the cluster to which each point is allocated.
- `centers`: A matrix of cluster centres.
- `totss`: The total sum of squares.
- `withinss`: Vector of within-cluster sum of squares, one component per cluster.
- `tot.withinss` :Total within-cluster sum of squares, i.e. sum(withinss).
- `betweenss`: The between-cluster sum of squares, i.e. `totss-tot.withinss`.
- `size`: The number of points in each cluster.

```
kmean_results
```

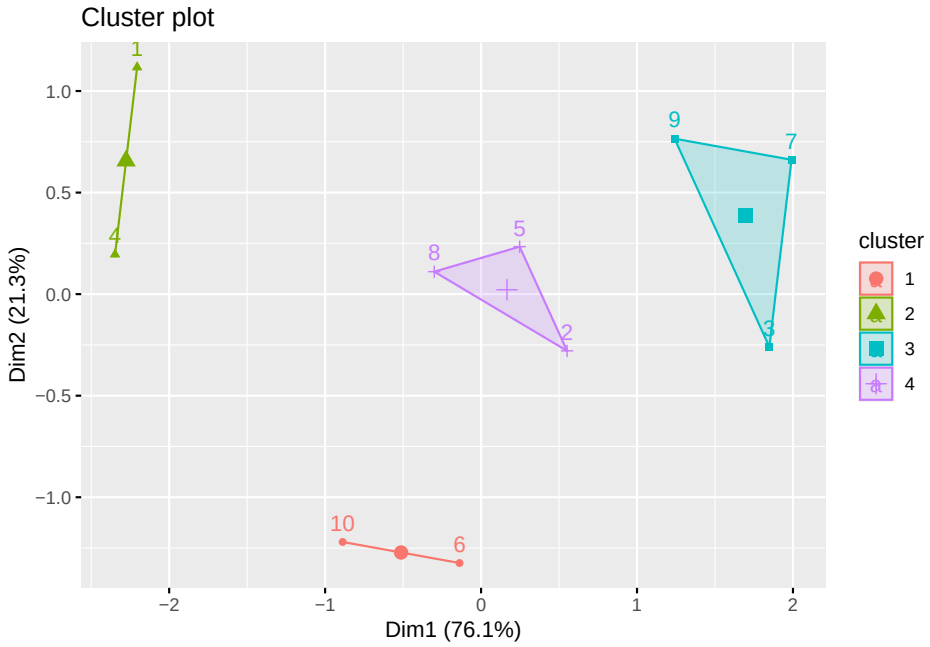
```
## K-means clustering with 4 clusters of sizes 2, 2, 3, 3
##
## Cluster means:
##         x1       x2       x3
## 1 120.5000 19.00000 44.50000
## 2 124.5000 20.00000 44.50000
## 3 121.0000 17.00000 41.66667
## 4 121.6667 18.33333 43.00000
##
## Clustering vector:
##  [1] 2 4 3 2 4 1 3 4 3 1
##
## Within cluster sum of squares by cluster:
## [1] 1.000000 1.000000 2.666667 1.333333
##  (between_SS / total_SS =  88.4 %)
##
```

```
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

## 18.3 Visualization

Use `factoextra::fviz_cluster(object, data)`. For more details about useful parameters, see the help document. Remember that `object` and `data` are two most indispensable parameters.

```
fviz_cluster(kmean_results, data = athlete)
```

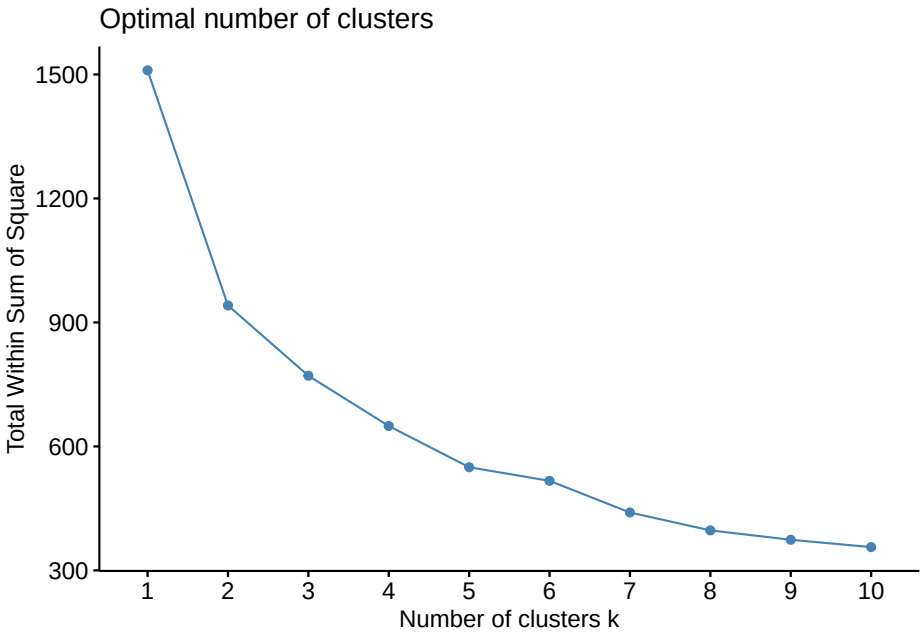


## 18.4 No. of Clusters

### 18.4.1 Within-Cluster SS

Looking for the "elbow" where the SS starts to slow down its drop. Here suggests 2 or 3.

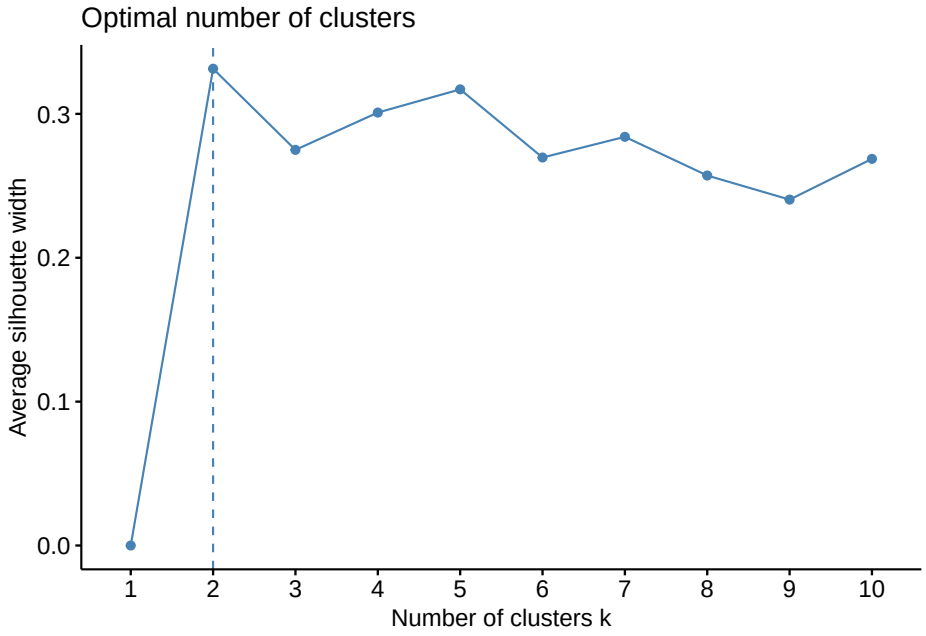
```
fviz_nbclust(automobile, kmeans, method = "wss", print.summary=TRUE)
```




### 18.4.2 Silhouette Method

Looking for the value that sets the highest average silhouette width, which indicates the optimal number of clusters.

```
fviz_nbclust(automobile, kmeans, method = "silhouette",print.summary=TRUE)
```



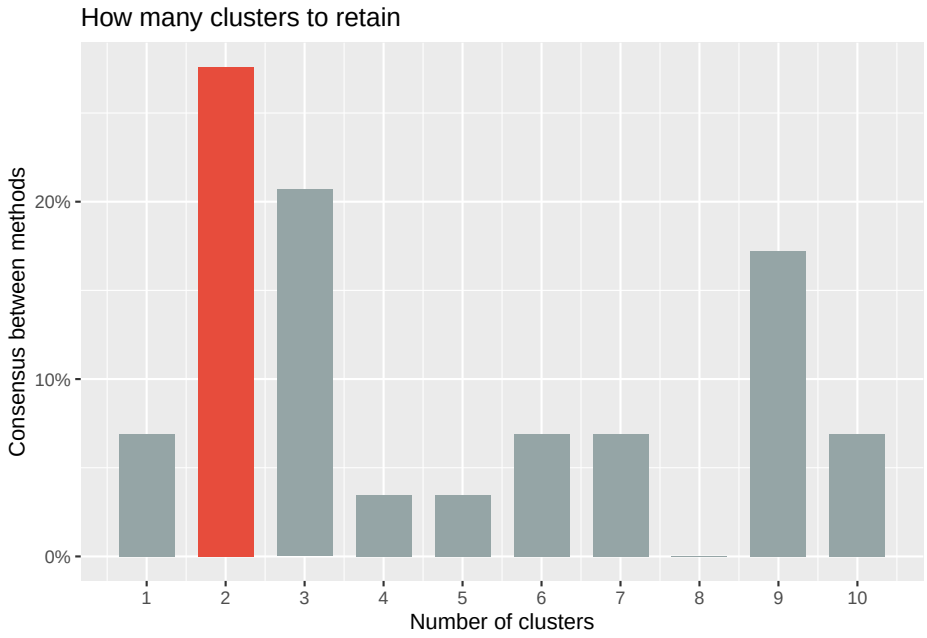

### 18.4.3 Majority Rule

`n_clusters()` is the main function to find out the optimal numbers of clusters present in the data based on the maximum consensus of a large number of methods.Note that the `n_clusters()` function will do the normalization automatically (by default).

```
n_clusters(data.frame(automobile))
```

```
## # Method Agreement Procedure:
##
## The choice of 2 clusters is supported by 8 (27.59%) methods out of 29 (Elbow, Silhou
```

```
plot(n_clusters(data.frame(automobile)))
```




### 18.4.4 Two Step Clustering

Two-Step Clustering refers to using hierarchical methods first to obtain the number of clusters and their centroids. Then use these as inputs to run non-hierarchical analysis.

```
# two step clustering in R, not required
library(prcr)
d <- pisaUSA15
m3 <- create_profiles_cluster(d,
                              broad_interest, enjoyment, instrumental_mot, self_efficac
                              n_profiles = 3)
summary(m3)
```