

# Introduction to Statistical Programming

Timothy Wong<sup>1</sup>  
James Gammerman<sup>2</sup>

August 21, 2018

---

<sup>1</sup>`timothy.wong@hotmail.co.uk`

<sup>2</sup>`jgammerman@gmail.com`

# Outline

- R Ecosystem

- Data Transformation

- Regression Models

  - Linear Regression

  - Poisson Regression

  - Logistic Regression

- Tree-based Methods

  - Decision Tree

  - Random Forest

- Neural Networks

  - Multilayer Perceptron

- Time Series Analysis

  - Auto-Correlation Function

  - Decomposition

  - ARIMA Model

- Survival Analysis

  - Kaplan-Meier Estimator

  - Cox Proportional Hazard Model

- Unsupervised Learning

  - K-means Clustering

  - Hierarchical Clustering

# R Ecosystem

# What is the R language?

- ▶ Offers modern and sophisticated statistical algorithms
- ▶ Used by millions of analysts and researchers worldwide
- ▶ Has a thriving open-source community
- ▶ Enables big data analytics

## Easy to Use

- ▶ PROC REG = `lm()` or `glm()`
- ▶ PROC SQL = `%>%`
- ▶ PROC SORT = `arrange()`
- ▶ PROC MEANS = `mean()`, `sd()`
- ▶ PROC GPLOT = `plot()`, `ggplot()`, `autoplot()`

# RStudio Server Pro

- RStudio is your integrated development environment (IDE)

The screenshot displays the RStudio Server Pro interface. The main editor window shows an R script with the following code:

```
722 # Visualize the non-priority appointment roll on a chart
723
724 getHead(model.appt.priority,
725         seq.Date(as.Date("2018-07-01") - days(MAX_DAYS_AHEAD_PRIORITY), as.Date("2018-07-01") + days(20), 1),
726         c("11P11", "11P12", "11P13", "11P14"),
727         14, events) %>%
728 ggplot(aes(x=days_ahead, y=num_appt_Pred, colour=operating_district_code)) + geom_line() + facet_wrap(~created_date, ncol = 7) +
729     scale_y_log10() +
730     xlab("Number of rolled days") +
731     ylab("Number of expected appointments for each initial contact")
732
733 # Calculate the number of appointments landed on each date.
734
735 wmis.appt.all.xdf %>%
736   group_by(operating_district_code, is_priority, appt_date) %>%
737   summarise(total_actual_appt = sum(num_appt)) %>%
738   rxxdfToDataFrame() %>%
739   as.tibble() %>%
740   readr::write_csv("wmis_appt_total.csv")
741
742 wmis.appt.total <- readr::read_csv("wmis_appt_total.csv")
743
744 # Plot a chart
745 fitted.contact %>%
746   filter(date >= seq.Date(from = TRAIN_START_DATE,
747                           to = TEST_END_DATE,
748                           by = 1)) %>%
749   select(opernum_day_of_rollcode, wmo_code, date,
750          contact_non_priority, contact_priority,
751          fitted_priority_fit, fitted_priorroll)
752
753 fitted_non_priority_fit, fitted_non_priority.sp.fit) %>%
```

The console output shows the execution of the script, including the creation of the `wmis.appt.total` tibble and the resulting data structure:

```
appt_date = col_date(format = "%Y-%m-%d"),
total_actual_appt = col_integer()
> wmis.appt.total
# A tibble: 354,604 x 4
  operating_district_code is_priority appt_date total_actual_appt
  <chr> <lgl> <dttm> <int>
1 11P11 FALSE 2015-07-01 0
2 11P42 FALSE 2015-07-01 5
3 65V04 FALSE 2015-07-01 0
4 21Q75 FALSE 2015-07-01 2
5 32R32 FALSE 2015-07-01 3
6 11P42 FALSE 2015-07-01 5
7 21Q74 FALSE 2015-07-01 3
8 65V73 FALSE 2015-07-01 1
9 70463 FALSE 2015-07-01 1
10 70442 FALSE 2015-07-01 5
# ... with 354,594 more rows
```

The Environment pane on the right shows the loaded objects and their sizes:

- Global Environment
- wmis.appt.total: 354684 obs. of 4 variables
- wmis.contact: 348196 obs. of 4 variables
- wmis.ref.geogra\_: 18318 obs. of 17 variables
- wmis.wmo.mapping: 173 obs. of 2 variables
- wx.actual.daily: 49648 obs. of 9 variables
- wx.actual.hourly: 1108932 obs. of 10 variables
- wx.seasonal.non\_: 219688 obs. of 7 variables
- wx.seasonal.non\_: 9158 obs. of 9 variables

The Files pane shows the project structure, including folders like `gitignore`, `Rhistory`, `efods_actual_hourly.csv`, `events.csv`, `gas-repair-demand.Rproj`, `gr.appt.hol`, `gr.contact.hol`, `mapping_l6.csv`, `model.RData`, `model.Rmd`, `model.Lappt.priority.rds`, `patch_map.Rmd`, `Sectors.xml`, `wmis_appt.all.xdf`, `wmis_appt_nonpriority.xdf`, and `wmis_appt_priority.xdf`.

# Packages

- ▶ **CRAN** is the Comprehensive R Archive Network.
- ▶ User-contributed packages: source code, binaries, documentation.

```
# Return all installed packages
installed.packages()
# Install a new package and all its dependencies from CRAN
# This will install to the default library location
install.packages("ggplot2", dependencies = TRUE)
# Load an installed package
# (both lines are identical)
library(ggplot2)
library("ggplot2")
```

# Packages

- ▶ **CRAN Task View** is a curated list of packages.
- ▶ Useful guide to get started with R.
- ▶ <https://cran.r-project.org/web/views/>



# Variable Assignment

- ▶ Assign variables using the <- symbol.
- ▶ Do not use reserved words. This will confuse the interpreter.

```
# Assign variables  
myVarX <- 5  
myVarY <- 20  
# Perform multiplication  
myVarX * myVarY  
# Look at the reserved words  
?Reserved
```

# Vectors

- ▶ R is a vectorised programming language.
- ▶ Vector contains elements of the same data type.

```
myVec1 <- 1:10
# Find out the length of vector
length(myVec1)

## [1] 10

# Reverse the vector
# This does not change the value of myVec1
rev(myVec1)

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

# Create a custom vector of 10, 15, 20, 25, 30
myVec2 <- c(10, 15, 20, 25, 30)
myVec2

## [1] 10 15 20 25 30

# Create a vector of sequential numbers with increment 0.5
myVec3 <- seq(from = -2, to = 2, by = 0.5)
myVec3

## [1] -2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0

# Elements of a vector can be named
myVec4 <- c(`New York` = 8.5,
            `London` = 8.6,
            `Moscow` = 11.9)
myVec4

## New York   London   Moscow
##      8.5      8.6     11.9
```

# Subsetting a Vector

- You can subset elements from a vector.

```
# Select the second element of the vector  
myVec2[2]
```

```
## [1] 15
```

```
# Subset a range from the vector  
myVec2[2:4]
```

```
## [1] 15 20 25
```

```
# Subset specified elements  
myVec2[c(4,2,3)]
```

```
## [1] 25 15 20
```

```
# Subset named element of a vector  
myVec4["New York"]
```

```
## New York
```

```
##      8.5
```

# Vectorised Operations

- Operations in R are vectorised.

```
# Arithmetic operations
```

```
myVec1 + 10
```

```
## [1] 11 12 13 14 15 16 17 18 19 20
```

```
myVec1 - 10
```

```
## [1] -9 -8 -7 -6 -5 -4 -3 -2 -1 0
```

```
myVec1 * 2
```

```
## [1] 2 4 6 8 10 12 14 16 18 20
```

```
myVec1 / 2
```

```
## [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
myVec1 ^ 2
```

```
## [1] 1 4 9 16 25 36 49 64 81 100
```

```
log(myVec1)
```

```
## [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
```

```
## [8] 2.0794415 2.1972246 2.3025851
```

# Character Vectors

- Vector can also contain character objects.

```
# Vector can contain character objects
myVec4 <- c("Bill", "Mark", "Steve", "Jeff", "Larry")
myVec4

## [1] "Bill" "Mark" "Steve" "Jeff" "Larry"

# Constant character vectors in R
LETTERS

## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
## [18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"

letters

## [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"

month.name

## [1] "January" "February" "March" "April" "May"
## [6] "June" "July" "August" "September" "October"
## [11] "November" "December"

month.abb

## [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov"
## [12] "Dec"
```

# Date and Date/Time Vectors

- ▶ Date is a data type in base R.
- ▶ The package lubridate extends date/time functionalities.
- ▶ Enables easier date/time manipulation.

```
# This is a vector of Date objects
myVec5 <- as.Date(c("2017-07-13",
                    "2017-10-11",
                    "2017-11-21",
                    "2018-01-16",
                    "2018-03-27"))

# Load the lubridate package
# Use the function ymd_hms() to parse date/time with timezone
# Returns a vector of POSIXct (date/time) object
library(lubridate)
myVec6 <- ymd_hms(c("2017-07-13 09:30:00",
                    "2017-10-11 08:00:00",
                    "2017-11-21 10:00:00",
                    "2018-01-16 11:30:00",
                    "2018-03-27 12:00:00"),
                  tz = "Europe/London")

# Date/time manipulation applied to a vector
myVec7 <- myVec6 + hours(1) + minutes(30)
# Compute the day of week - returns a vector of characters
weekdays(myVec7)

## [1] "Thursday" "Wednesday" "Tuesday" "Tuesday" "Tuesday"
```

# Logical Operators

- Apply logical operators on vector objects.

```
# Find all values greater than 5 - returns a vector of logical values  
myVec1 > 5
```

```
## [1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE
```

```
# Find all values equal to 7  
myVec1 == 7
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE
```

```
# Find all values matching 2,4,6 and 8  
myVec1 %in% c(2,4,6,8)
```

```
## [1] FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE FALSE
```

```
# Find all values between 2 and 7  
myVec1 >= 2 & myVec1 <= 7
```

```
## [1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE
```

```
# Find all values equal to 7 or equal to 8  
myVec1 == 7 | myVec1 == 8
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE FALSE
```

# Special Numbers in R

```
# Pi is constant 3.14159...
pi

## [1] 3.141593

# One divided by zero is infinity
1/0

## [1] Inf

# Negative number divided by zero is negative infinity
-1/0

## [1] -Inf

# Infinity divided by infinity is Not-a-Number (NaN)
Inf/Inf

## [1] NaN

# Not available (NA) plus one is still NA
NA + 1

## [1] NA

# Effects of different special numbers
c(5, 10, 15, NA, 25, 30, NaN, 35, 40, Inf, 50, -Inf, 60) / 5

## [1] 1 2 3 NA 5 6 NaN 7 8 Inf 10 -Inf 12
```



# List

- List is a generic container for objects of different data types

```
myFavBook <- list(title = "R for Data Science",
  authors = c("Garrett Grolemund", "Hadley Wickham"),
  publishDate = as.Date("2016-12-12"),
  price = 18.17,
  currency = "USD",
  edition = 1,
  isbn = 1491910399)

myFavBook

## $title
## [1] "R for Data Science"
##
## $authors
## [1] "Garrett Grolemund" "Hadley Wickham"
##
## $publishDate
## [1] "2016-12-12"
##
## $price
## [1] 18.17
##
## $currency
## [1] "USD"
##
## $edition
## [1] 1
##
## $isbn
## [1] 1491910399
```

# Subsetting a List

- ▶ You can subset a particular member from a list

```
# Select a named element of a list  
# Use the dollar sign, followed by name without bracket  
myFavBook$title  
  
## [1] "R for Data Science"  
  
# Use double squared brackets with element's name as character  
myFavBook[["authors"]]  
  
## [1] "Garrett Grolemund" "Hadley Wickham"  
  
# Select the fourth element in the list  
myFavBook[[4]]  
  
## [1] 18.17
```

# Data Frame

- ▶ Table with rows (observations) and columns (variables).
- ▶ Analogous to an Excel workbook.

[illegible]

# Data Frame

```
myFavMovies1
```

```
##           title year    box      bondActor
## 1           Dr. No 1962   59.5      Sean Connery
## 2         Goldfinger 1964  125.0      Sean Connery
## 3 Diamonds are Forever 1971 120.0      Sean Connery
## 4           Moonraker 1979  210.3      Roger Moore
## 5 The Living Daylights 1987  191.2 Timothy Dalton
## 6           GoldenEye 1995  355.0 Pierce Brosnan
## 7       Casino Royale 2006  599.0   Daniel Craig
```

# Tibble

- ▶ Similar to traditional data frame.
- ▶ tibble is the modern standard in R.

```
library(tibble)
myFavMovies2 <- tibble(title = c("Dr. No",
                                "Goldfinger",
                                "Diamonds are Forever",
                                "Moonraker",
                                "The Living Daylights",
                                "GoldenEye",
                                "Casino Royale"),
  year = c(1962, 1964, 1971, 1979,
           1987, 1995, 2006),
  box = c(59.5, 125, 120, 210.3,
          191.2, 355, 599),
  bondActor = c("Sean Connery",
                 "Sean Connery",
                 "Sean Connery",
                 "Roger Moore",
                 "Timothy Dalton",
                 "Pierce Brosnan",
                 "Daniel Craig"))

# Append an extra row at the end of the tibble
# Rewrite the original tibble object
myFavMovies2 <- add_row(myFavMovies2,
  title = "Spectre", year = 2015, box = 880.7,
  bondActor = "Daniel Craig")
```

# Tibble

```
myFavMovies2
```

```
## # A tibble: 8 x 4
```

```
##   title                year    box bondActor  
##   <chr>                <dbl> <dbl> <chr>
```

```
## 1 Dr. No                1962  59.5 Sean Connery
```

```
## 2 Goldfinger            1964  125   Sean Connery
```

```
## 3 Diamonds are Forever  1971  120   Sean Connery
```

```
## 4 Moonraker             1979  210.  Roger Moore
```

```
## 5 The Living Daylights  1987  191.  Timothy Dalton
```

```
## 6 GoldenEye             1995  355   Pierce Brosnan
```

```
## 7 Casino Royale         2006  599   Daniel Craig
```

```
## 8 Spectre               2015  881.  Daniel Craig
```

# Subsetting a Tibble

```
# Get one column by name
myFavMovies2[["title"]]
myFavMovies2$title

# Get a range of columns by position ID
myFavMovies2[, 1:2]
myFavMovies2[1:2]

# Get rows 1 to 3
myFavMovies2[1:3, ]

# Get the "year" variable of row 1-3
myFavMovies2[1:3, "year"]

# Get the "title" and "year" variables of row 4-7
myFavMovies2[4:7, c("title", "year")]
```

# Functions

- Functions are vectorised.

```
is.odd <- function(x) {  
  # The modulo operator %% returns the remainder  
  # If a number divide by 2 gives remainder 1, then it is an odd number  
  remainder <- x %% 2  
  equalToOne <- remainder == 1  
  return(equalToOne)  
}  
# Execute the function with one input  
is.odd(5)  
  
## [1] TRUE  
  
# Execute the function with an integer vector  
is.odd(1:10)  
  
## [1] TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE  
  
# Define another function  
is.even <- function(x) {  
  !is.odd(x)  
}  
# Return true for even numbers  
is.even(1:10)  
  
## [1] FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE
```



# If-Else

```
library(lubridate)
# Find out what day is today
myWeekday <- weekdays(today())
# Check whether today is Saturday or Sunday
if (myWeekday %in% c("Saturday","Sunday")) {
  myGreeting <- "Have a nice weekend"
} else {
  myGreeting <- "Go back to work"
}
# Prints the message
myGreeting
```

# If-Else

- Multiple conditions are allowed.

```
library(lubridate)
myWeekday <- weekdays(today())
# Checks multiple conditions
if (myWeekday %in% c("Saturday","Sunday")) {
  myGreeting <- "Have a nice weekend"
} else if(myWeekday == "Friday"){
  myGreeting <- "It's Friday!"
} else if(myWeekday == "Monday"){
  myGreeting <- "Oh no..."
} else {
  myGreeting <- "Go back to work"
}
myGreeting
```

# While

- ▶ Loops as long the condition stays TRUE.

```
myCounter <- 100
while (myCounter > 0) {
  myCounter <- myCounter - 5
  print(myCounter)
}
```

# While

- ▶ Skip an iteration using `next`
- ▶ Early exit using `break`

```
myCounter <- 100
while (myCounter > 0) {
  myCounter <- myCounter - 5
  if (myCounter > 50) {
    # Skips all iterations if the counter value is greater than 50
    next
  }
  if (myCounter == 10) {
    # Early stop if the counter value matches 10
    break
  }
  print(myCounter)
}
```

```
## [1] 50
## [1] 45
## [1] 40
## [1] 35
## [1] 30
## [1] 25
## [1] 20
## [1] 15
```

# For

```
myResult <- 0
for (i in 1:100) {
  myResult <- myResult + i ^ 2
}
myResult

## [1] 338350
```

# Apply

- ▶ `apply()` takes a 'grid' input: data frame, tibble, or matrix
- ▶ Margin 1 = apply over rows
- ▶ Margin 2 = apply over columns

```
# The second argument 1 indicates iterate over rows
myMessages1 <- apply(myFavMovies2, 1, function(row){
  sprintf("%s was released in %s.", row["title"], row["year"])
})
myMessages1

## [1] "Dr. No was released in 1962."
## [2] "Goldfinger was released in 1964."
## [3] "Diamonds are Forever was released in 1971."
## [4] "Moonraker was released in 1979."
## [5] "The Living Daylights was released in 1987."
## [6] "GoldenEye was released in 1995."
## [7] "Casino Royale was released in 2006."
## [8] "Spectre was released in 2015."
```

# lapply

- ▶ `lapply()` always returns a list

```
# The second argument 1 indicates iterate over rows
myMessages2 <- lapply(myFavMovies2$title,
                      function(x){ sprintf("%s is a great movie!", x) })
# Checks the data type of the result
typeof(myMessages2)

## [1] "list"

# Check whether it is a list
is.list(myMessages2)

## [1] TRUE

# Select the 6th element of the list
myMessages2[[6]]

## [1] "GoldenEye is a great movie!"
```

# apply

- ▶ `sapply()` always returns a vector

```
myMessages3 <- sapply(myFavMovies2$title,  
                      function(x){ sprintf("%s is a great movie!", x) })  
# Check whether it is a list  
is.list(myMessages3)
```

```
## [1] FALSE
```

```
myMessages3
```

```
##                               Dr. No  
##      "Dr. No is a great movie!"  
##                               Goldfinger  
##      "Goldfinger is a great movie!"  
##                               Diamonds are Forever  
##      "Diamonds are Forever is a great movie!"  
##                               Moonraker  
##      "Moonraker is a great movie!"  
##                               The Living Daylights  
##      "The Living Daylights is a great movie!"  
##                               GoldenEye  
##      "GoldenEye is a great movie!"  
##                               Casino Royale  
##      "Casino Royale is a great movie!"  
##                               Spectre  
##      "Spectre is a great movie!"
```



# Looping

- ▶ Looping can be slow.
- ▶ Always try to vectorise your code.

```
# Vectorised operation is fast
system.time({
  myResult <- 1:10000 * 2
})

##      user  system elapsed
##         0         0         0

# Looping is quite slow
system.time({
  myResult <- sapply(1:10000, function(x){ x * 2 })
})

##      user  system elapsed
##         0         0         0

# Appending to vector is much slower
system.time({
  myResult <- c()
  for(i in 1:10000){
    myResult <- c(myResult, i * 2)
  }
})

##      user  system elapsed
##    0.17    0.00    0.17
```

# UK User Communities

- ▶ LondonR -  LONDONR
- ▶ ManchesterR -  MANCHESTERR
- ▶ CaRdiff -  CaRdiff
- ▶ SheffieldR
- ▶ EdinbR -  EdinbR
- ▶ CambR -  CambR
- ▶ NottinghamR
- ▶ BirminghamR -  BIRMINGHAMR
- ▶ Oxford RUG -  Oxford
- ▶ Bristol Data Scientists -  BRISTOL  
DATA SCIENTISTS

# International User Communities

- ▶ International R User Conference (useR!)



- ▶ Enterprise Application of the R Language (EARL)



- ▶ European R User Meeting (ERUM)



# Data Transformation

# The Tidyverse

# Regression Models

# Simple Regression

Univariate linear regression model

$$\hat{y}_i = \beta_0 + \beta_1 x_i$$

- ▶ Analogous to a straight line  $y = mx + c$
- ▶ Can be chained with  $M$  dependent variables (Multivariate)

$$\hat{y}_i = \beta_0 + \sum_{m=1}^M \beta_m x_{m,i}$$

- ▶ Residual term  $\epsilon_i = y_i - \hat{y}_i$  assumed to be Gaussian

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

- ▶ Also known as ordinary least squared (OLS) regression

# Linear Regression in R

```
# Build a univariate linear model
# These two lines are equivalent
myModel1 <- lm(mpg ~ wt, mtcars)
myModel1 <- lm(formula = mpg ~ wt, data = mtcars)
# Read the model summary
summary(myModel1)

##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -4.5432 -2.3647 -0.1252  1.4096  6.8727
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   37.2851     1.8776  19.858 < 2e-16 ***
## wt           -5.3445     0.5591  -9.559 1.29e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.046 on 30 degrees of freedom
## Multiple R-squared:  0.7528, Adjusted R-squared:  0.7446
## F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```



# More Linear Regression Models

**Multivariate linear model** Additional independent variables can be chained using the + symbol. Categorical variables can be encoded as dummy on-the-fly using `factor()`.

```
myModel2 <- lm(mpg ~ wt + hp + qsec + factor(am), mtcars)
```

**Polynomial term** Model can become more flexible when an independent variable is converted into polynomial terms. Use the `poly()` function.

```
myModel3 <- lm(mpg ~ wt + qsec + factor(am) +  
               poly(hp, 3), mtcars)
```

**Interaction term** Two variables can be combined to create synergy effect. The \* symbol is used to combine variables together.

```
myModel4 <- lm(mpg ~ wt * hp + qsec + factor(am), mtcars)
```

# Regression Diagnostics

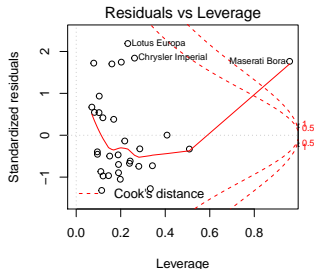
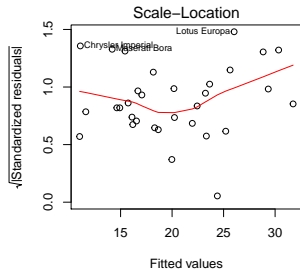
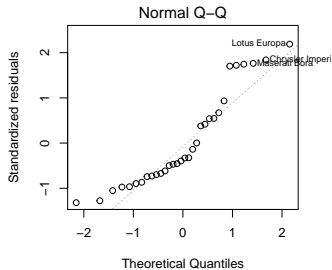
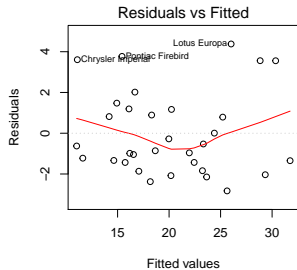
**Residuals vs Fitted** Checks for non-linear relationship. Look for a near horizontal line.

**Normal Quantile-Quantile** It aligns model residuals against a theoretical normal distribution. If the residuals spread along a straight diagonal line on the Q-Q plot, it suggests that the residuals are normally distributed.

**Scale-Location** Checks for homoscedasticity and heteroscedasticity. It is homoscedastic if observations scatter without any observable pattern.

**Residual vs Leverage (Cook's Distance)** Identifies observations having strong influence to the model.

# Diagnostics Plots

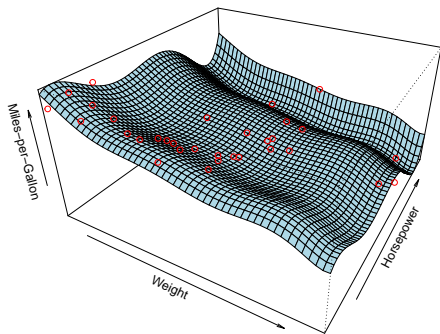


# Overfitting

- ▶ Flexible models are prone to overfitting.
- ▶ Overfitting makes the model less generalisable.
- ▶ Solution
  - ▶ Use less flexible methods.
  - ▶ Impose regularisation.

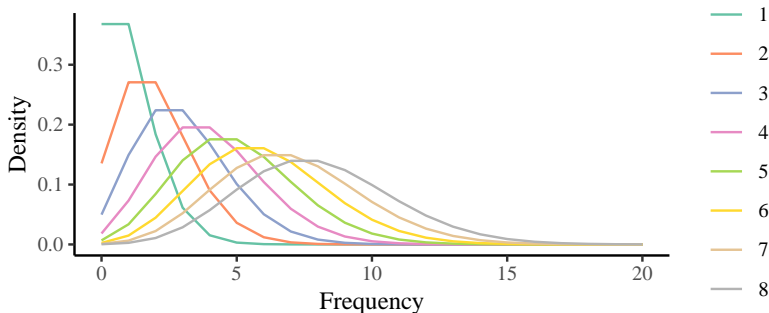
# Overfitting: Visual Explanation

$$\hat{y} = \beta_0 + \sum_{j=1}^8 \beta_{wtj} x_{wt}^j + \sum_{k=1}^5 \beta_{hp_k} x_{hp}^k$$



# Poisson Distribution

- ▶ Count of distinct events are drawn from Poisson distribution.
  - ▶ Always positive.
  - ▶ In most cases they are integers.
- ▶ e.g. Number of people in a room, number of flights delayed per day... etc.



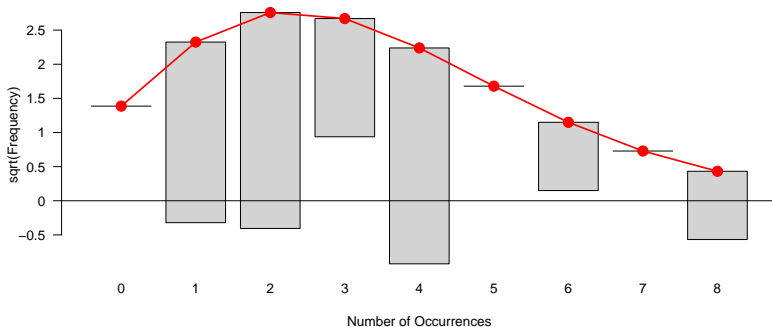
# Testing for Poisson Distribution

- ▶ Chi-square goodness of fit test.
- ▶ Fits the data against theoretical Poisson distribution.
- ▶ Look for statistical significance.

```
# Performs the Chi-squared goodness-of-fit test.  
# It checks whether the variable is drawn from a Poisson distribution.  
library(vcd)  
gf <- goodfit(mtcars$carb, type= "poisson", method= "ML")  
# Checks the statistical p-value of the goodness-of-fit test.  
# If  $p \leq 0.05$  then it is safe to say that the variable is Poisson.  
summary(gf)  
  
##  
## Goodness-of-fit test for poisson distribution  
##  
## X^2 df P(> X^2)  
## Likelihood Ratio 20.53973 4 0.0003906369
```

# Goodness of Fit Plot

*# Plots the observed frequency vs theoretical Poisson distribution.*  
*# The hanging bars should fill the space if it is perfectly Poisson.*  
`plot(gf)`





# Poisson Regression

```
# Build a Poisson model to predict the number of carburetors in a car.
myPoissonModel <- glm(carb ~ hp + wt + factor(am),
                      family="poisson",
                      data=mtcars)

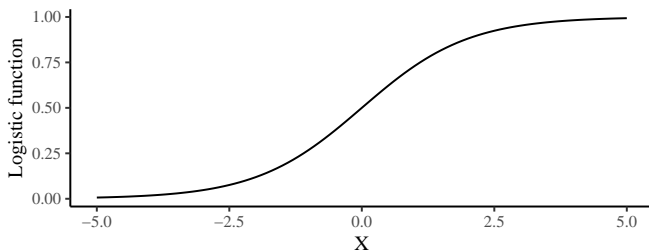
# Read the model summary
summary(myPoissonModel)

##
## Call:
## glm(formula = carb ~ hp + wt + factor(am), family = "poisson",
##      data = mtcars)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -0.91420  -0.48423  -0.07246   0.19252   1.26155
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.418081   0.604211  -0.692   0.4890
## hp           0.004316   0.001880   2.296   0.0217 *
## wt           0.179583   0.191352   0.938   0.3480
## factor(am)1  0.393750   0.324978   1.212   0.2257
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##      Null deviance: 27.043  on 31  degrees of freedom
## Residual deviance: 10.798  on 28  degrees of freedom
## AIC: 108.16
##
## Number of Fisher Scoring iterations: 4
```

# Binomial Distribution

- ▶ There are only two possible outcomes  $\{Y, \neg Y\}$ .
  - ▶ Toss a coin (Head or tail)
  - ▶ Taking an examination (Pass or fail)
  - ▶ Selling a product (Sold or unsold)
- ▶ Likelihood of event  $Y$  and  $\neg Y$  expressed as probability  $P(Y) + P(\neg Y) = 1$ .
- ▶ Logistic function squeezes real value range  $X$  into  $(0, 1)$  to express probability.

$$P(Y) = \frac{1}{1 + e^{-X}}$$



# Logistic Regression

- Equation for logistic regression

$$P(Y) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_M x_M)}}$$

- Coefficients  $\beta_1, \beta_2, \beta_3, \dots, \beta_M$  can be converted into odds ratios  $OR(x_1), OR(x_2), OR(x_3), \dots, OR(x_M)$ .

$$OR(x_1) = \frac{odds(x_1 + 1)}{odds(x_1)} = \frac{e^{\beta_0 + \beta_1(x_1 + 1) + \beta_2 x_2 + \dots + \beta_M x_M}}{e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_M x_M}} = e^{\beta_1}$$

- $OR(x_1)$  Represents the change in probability when  $x_1$  increases by 1 unit.

# Training a Logistic Regression Model

- Build a logistic regression model to predict the dependent variable am. (1=manual; 0=auto)

```
myLogisticModel <- glm(am ~ mpg + hp + disp,  
                        family="binomial",  
                        data=mtcars)  
summary(myLogisticModel)
```

Calculate the odds ratios for this model.

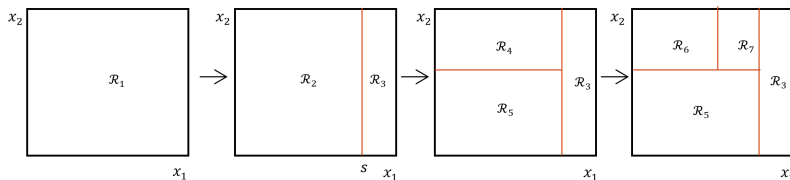
```
# Calculate the odds-ratios by taking the exponential of the coefficients  
# You may also calculate the 95% confidence interval of the odds-ratio  
exp(cbind(oddsratio = myLogisticModel$coefficients,  
          confint(myLogisticModel)))
```

##	oddsratio	2.5 %	97.5 %
## (Intercept)	2.066683e-15	9.104693e-49	0.02539238
## mpg	3.614582e+00	1.198283e+00	61.43800377
## hp	1.161095e+00	1.049568e+00	1.50974362
## disp	9.366496e-01	8.197068e-01	0.98685201

# Tree-based Methods

# Recursive Partitioning

- ▶ Cut off point is denoted as  $s$ .
- ▶ Divides data into regions (leaves)  $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \dots$  recursively.
- ▶ Works with real values as well as categorical variables.
- ▶ Large tree risks overfitting
  - ▶ Removes weaker leaves.
  - ▶ Regularisation.



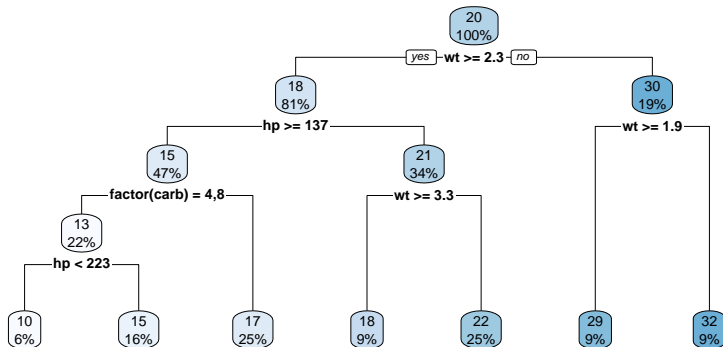
# Decision Tree

- Trees can be trained with a formula and optional control parameters.

```
# Load the rpart package for recursive partitioning  
library(rpart)  
# Build a decision tree to predict mpg  
myTree <- rpart(formula = mpg ~ wt + hp +  
                 factor(carb) +  
                 factor(am),  
                 data = mtcars,  
                 control = rpart.control(minsplit=5))  
# Read the detailed summary of the tree  
summary(myTree)
```

# Decision Tree: Visualisation

```
# Load the rpart.plot package for tree visualisation  
library(rpart.plot)  
rpart.plot(myTree)
```



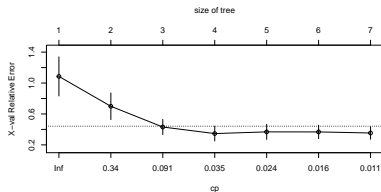


# Tree Pruning

```
printcp(myTree)

##
## Regression tree:
## rpart(formula = mpg ~ wt + hp + factor(carb) + factor(am), data = mtcars,
##       control = rpart.control(minsplit = 5))
##
## Variables actually used in tree construction:
## [1] factor(carb) hp          wt
##
## Root node error: 1126/32 = 35.189
##
## n= 32
##
##      CP nsplit rel error  xerror   xstd
## 1 0.652661    0  1.000000 1.08570 0.252677
## 2 0.178618    1  0.347339 0.69922 0.172943
## 3 0.046269    2  0.168721 0.43098 0.098563
## 4 0.026109    3  0.122453 0.34614 0.095951
## 5 0.022593    4  0.096343 0.36810 0.097621
## 6 0.011989    5  0.073751 0.36754 0.087290
## 7 0.010000    6  0.061762 0.35358 0.081453

plotcp(myTree)
```



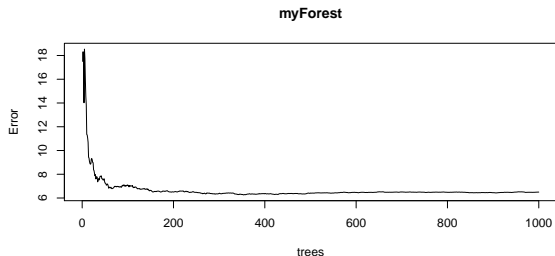
# Random Forest

- ▶ Consists of many decision trees
  - ▶ Randomly selected variables will be used in each split
  - ▶ Usually no need to prune them (all trees are allowed to grow big)
- ▶  $M$  trees in a random forest produces  $M$  predictions
  - ▶ Final prediction is calculated as mean value for regression problem
  - ▶ Classification problem will use most the common label (majority voting)

# Training a Random Forest

```
library(randomForest)
library(dplyr)
# Build a random forest with 1000 trees
# Each tree has 2 randomly selected variables
# You can change the parameters
myForest <- randomForest(mpg ~ wt + hp + carb + am,
                          ntree = 1000,
                          mtry = 2,
                          data = mtcars %>% mutate(carb = factor(carb),
                                                    am = factor(am)))

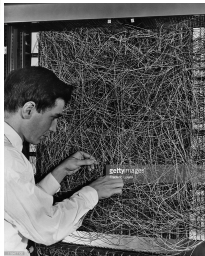
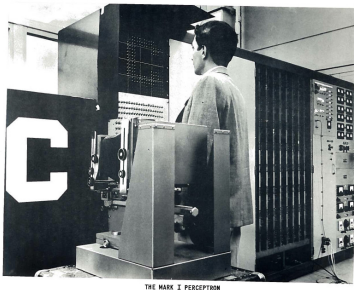
# Plot the error as the forest expands
plot(myForest)
```



# Neural Networks

# Artificial Neurons

- ▶ Inspired by neurons in biological brain.
- ▶ McCulloch and Pitts described neuron as a logical process.
  - ▶ Neuron takes several inputs  $\{x_1, x_2, x_3, \dots, x_M\}$
  - ▶ Fires (activates) if the combined weighted input  $\sum_{m=1}^M w_m x_m$  exceeds threshold.
  - ▶ Output can either be fire 1 or not fire 0.
- ▶ Rosenblatt's Mark I Perceptron



# Modern Neural Networks

- ▶ Neural nets are based on non-linear processing power.
- ▶ Trained via gradient descent optimisers (backpropagation).
  - ▶ Network initialise randomly.
  - ▶ Requires differentiable loss function.
  - ▶ Requires strong gradient in order to improve.
  - ▶ Model weights improve iteratively.
  - ▶ Converge at local minimum.
- ▶ Neurons can be stacked as layers.
  - ▶ Can either be shallow (1 layer) or deep (many layers).
  - ▶ State-of-the art neural networks have highly bespoke topologies.

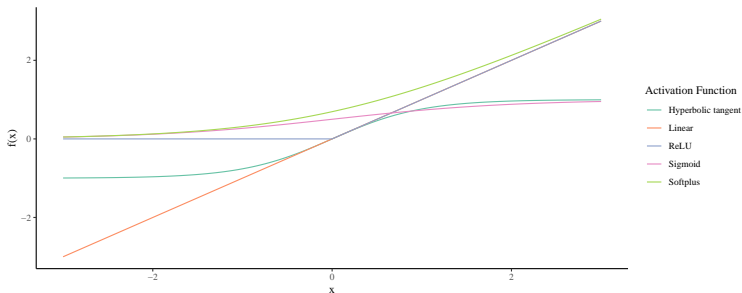
# Activation

- ▶ Weighted inputs are combined linearly.

$$X = \sum_{m=1}^M w_m x_m$$

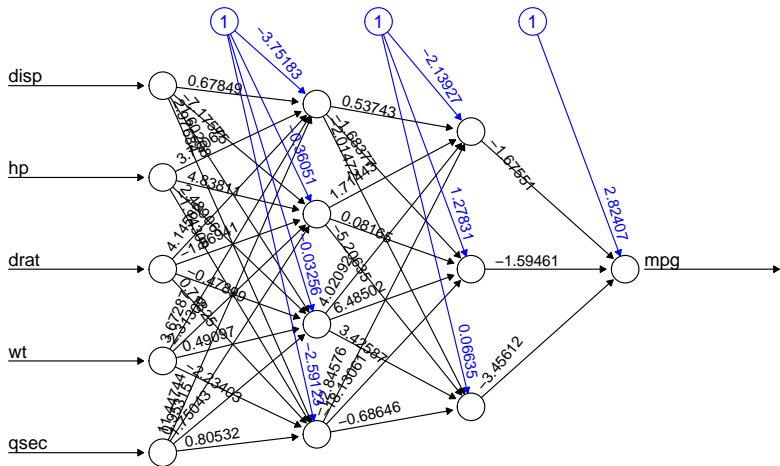
- ▶ Non-linear activation functions

- ▶ Sigmoid
- ▶ Hyperbolic tangent
- ▶ etc...



# Multilayer Perceptron: Topology

- ▶ Layers are fully-interconnected.
- ▶ Usually having two or more layers.



Error: 0.045594 Steps: 2263



# Time Series Analysis

# Time Series Data

- ▶ Observations repeatedly taken at regular interval.
- ▶ Explore variable relationship across temporal space.

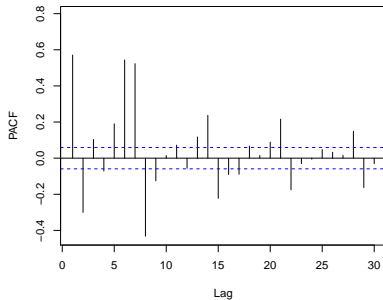
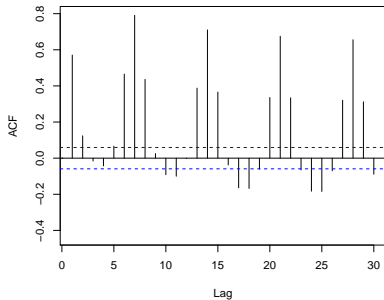
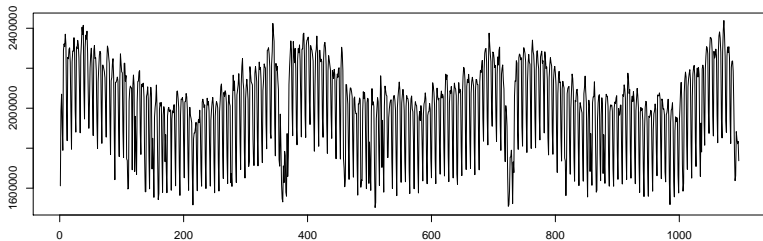
**Auto-correlation Function (ACF)** Measures the correlation of a single variable along the temporal dimension between  $x_t$  and  $x_{t+h}$ . It shows the correlation of the variable over different lag periods. For most time series variables, correlation is usually strong at lag  $h = 1$  and it gradually diminishes as lag period increases. Cyclic pattern in the correlogram suggests possible seasonality which you can analyse further.

**Partial Auto-correlation Function (PACF)** Also measures the correlation between different lag periods, but it controls the correlation across the temporal dimension so that only the contribution of an individual lag is reflected.

**Cross Correlation Function (CCF)** Analyses the temporal correlation between two variables.

# ACF and PACF Correlograms

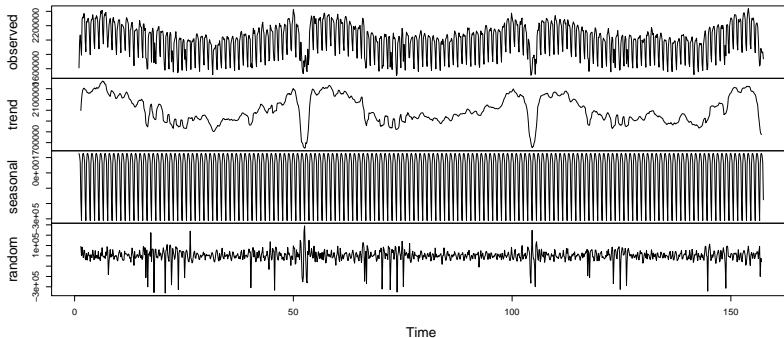
myTrainingSet\$total\_demand



# Decomposition

- ▶ Time series can be decomposed into:
  - ▶ Seasonal component  $S_t$
  - ▶ Trend component  $T_t$
  - ▶ Residual component  $\epsilon_t$
- ▶ Additive time series is expressed as  $X_t = S_t + T_t + \epsilon_t$
- ▶ Multiplicative time series is expressed as  $X_t = S_t \times T_t \times \epsilon_t$

Decomposition of additive time series



# Time Series Linear Regression Model

- ▶ Decomposed components can be used as independent variable in linear regression:

$$X_t = \beta_0 + \beta_{trend} T_t + \beta_{seasonal} S_t + \sum_{m=1}^M (\beta_m x_{mt}) + \epsilon_t$$

- ▶ Components can also form interaction terms with other independent variables.

# Auto-regressive Moving Average Model (ARMA)

- ▶  $ARMA(p, q)$  model

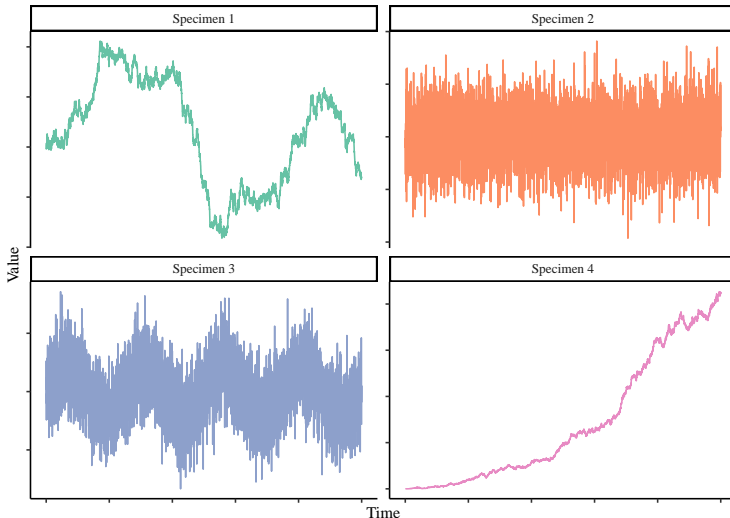
$$\underbrace{X_t}_{\text{Observation}} = \underbrace{\beta_0}_{\text{intercept}} + \underbrace{\sum_{i=1}^p (\phi_i X_{t-i})}_{\text{AR}(p)} + \underbrace{\sum_{i=1}^q (\theta_i \epsilon_{t-i})}_{\text{MA}(q)} + \underbrace{\epsilon_t}_{\text{residual}}$$

- ▶  $ARIMA(p, d, q)$  model

- ▶ Auto-regressive Integrative Moving Average
- ▶  $I(d)$   $d^{\text{th}}$  order integration can be added.
  - ▶ Integration refers to the difference from previous time step
  - ▶ First order differencing  $I(1)$  :  $X'_t = X_t - X_{t-1}$
  - ▶ To satisfy **stationarity** requirement.

# Stationarity

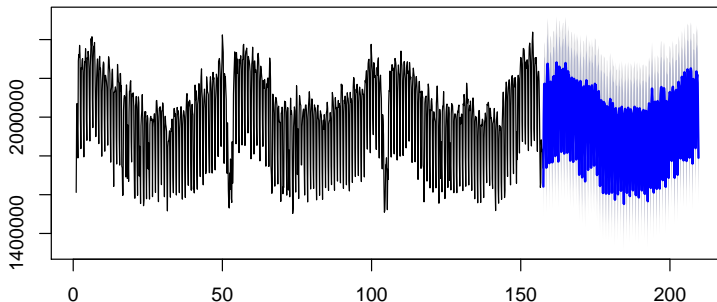
- ▶ Equal properties (mean and variance) across time.
  - ▶ Only one below is a stationary time series.
  - ▶ Which one is it?



# ARIMA with Seasonality (SARIMA)

- ▶  $ARIMA(p, d, q)(P, D, Q)_m$ 
  - ▶ All parameter values can be automatically identified.
  - ▶ Simple models are always preferred
  - ▶ Intend to keep  $p + q + P + Q$  small.

**Forecasts from Regression with  $ARIMA(2,0,0)(1,1,1)[7]$  errors**





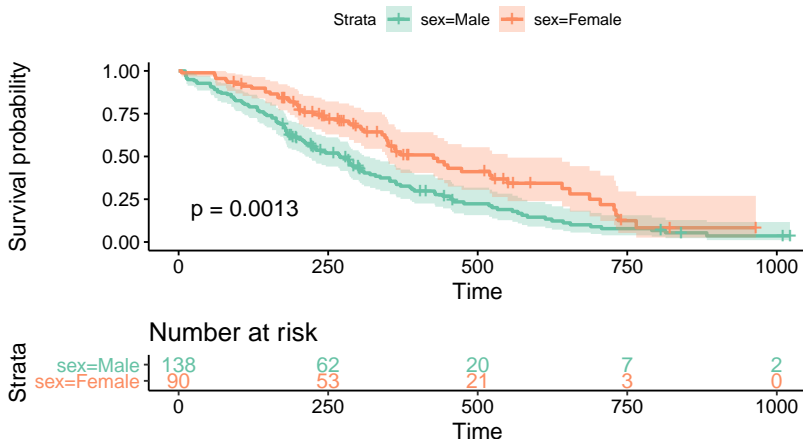
# Survival Analysis

# Survival Analysis

- ▶ Event occurring at irregular intervals.
  - ▶ e.g. Patients gets sick, machine failing... etc
- ▶ Also known as **time-to-event analysis** or **event history analysis**.

# Kaplan-Meier Estimator

- ▶ It is used to measure how many subjects survives in a clinical trial since treatment began.
- ▶ Categorical variables only.



# Cox Proportional Hazard Model

- ▶ It is a regression method which can take into account categorical and numeric variables.
- ▶ Assumes effects are time-independent (proportional hazard assumption).
- ▶ Hazard function  $h_t$  is defined as:

$$h_t = h_{0,t} \times e^{\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots + \beta_M x_M}$$

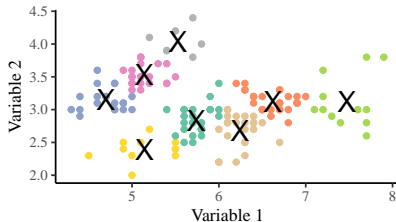
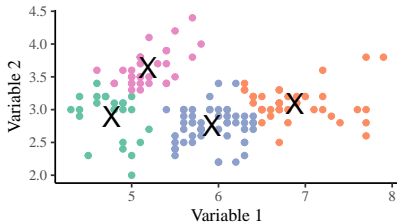
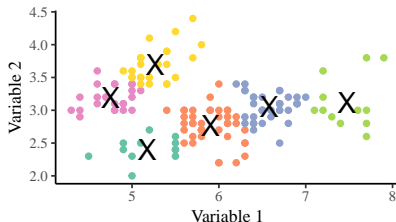
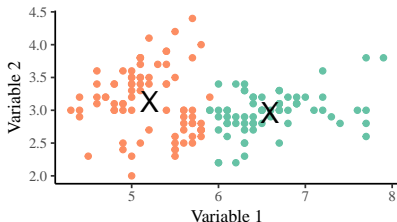
```
# Build a Cox model with predictor variables
myCoxModel1 <- coxph(Surv(time, status==2) ~ factor(sex) + age +
                    ph.ecog + ph.karno +
                    pat.karno +
                    meal.cal + wt.loss, data = lung)

# Read the model summary
summary(myCoxModel1)
```

# Unsupervised Learning

# K-means Clustering

- ▶ Works with unlabelled data.
- ▶ Arrange objects into groups (clusters)
- ▶ Objective interpretation of results as  $K$  is arbitrarily selected.



# Agglomerative Hierarchical Clustering

- ▶  $N$  objects can form maximum  $N$  clusters, each having 1 member object.
- ▶ Identify distance between closest cluster pair.
- ▶ Merge them.
- ▶ Repeat until there are no more clusters left.

# Agglomerative Hierarchical Clustering: Example

