REASONING SYSTEMS
DAY 4 HANDS-ON

Fan Zhenzhen
Institute of Systems Science
National University of Singapore
E-mail: zhenzhen@nus.edu.sg

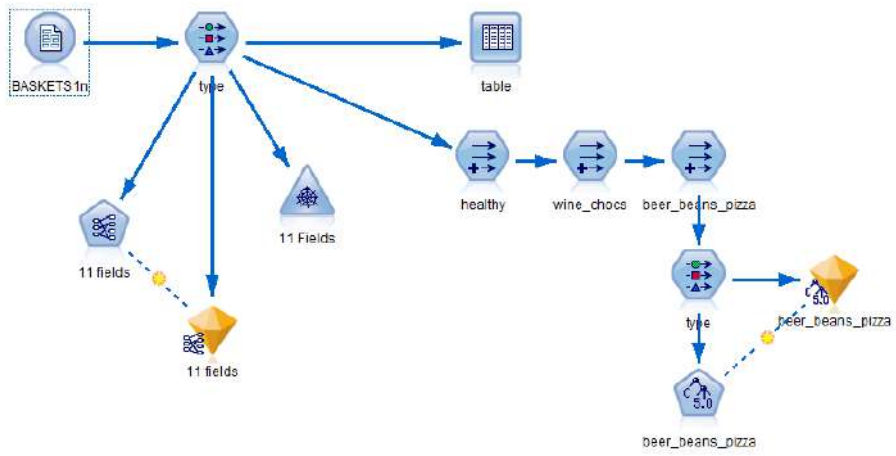
Association Analysis & Profiling with Induction:

A Case Study using SPSS Modeler

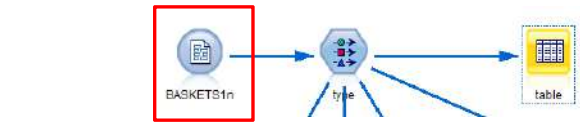
Context

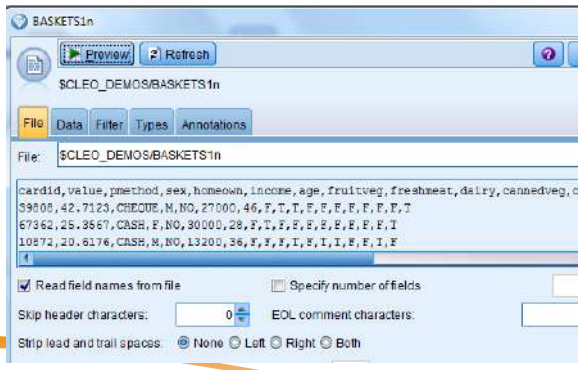
- Supermarket transaction data
- With personal data of the purchaser, collected through the usage of loyalty cards
- Objective:
  - To discover groups of customers who buy similar products
  - To characterize (or profile) such groups demographically
- Using link analysis and association rule modeling to find products frequently bought together
- Using C5.0 rule induction to for group profiling

The Overall Stream

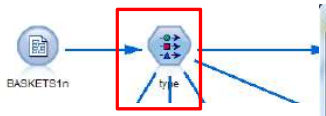


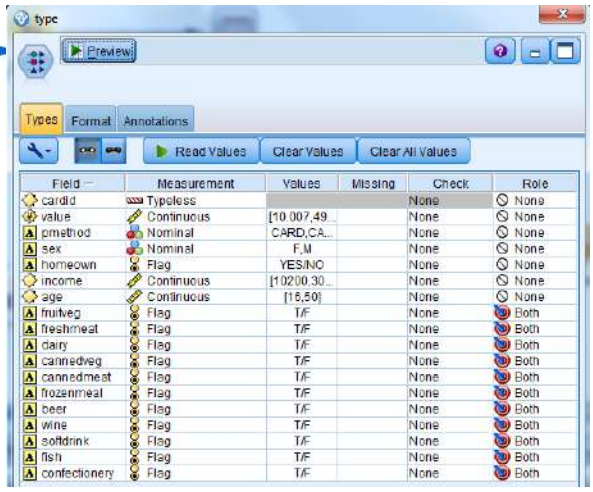
The Data File





Typing the Data





The Variables

- **Basket summary:**
  - *cardid*. Loyalty card identifier for customer purchasing this basket.
  - *value*. Total purchase price of basket.
  - *pmethod*. Method of payment for basket.
- **Personal details of cardholder:**
  - *sex*
  - *homeown*. Whether or not cardholder is a homeowner.
  - *income*
  - *age*
- **Basket contents—flags for presence of product categories:**
  - *fruitveg*
  - *freshmeat*
  - *dairy*
  - *cannedveg*
  - *cannedmeat*
  - *frozenmeal*
  - *beer*
  - *wine*
  - *softdrink*
  - *fish*
  - *confectionery*

The Data Table

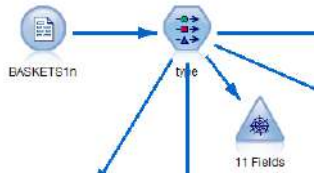
table (18 fields, 1,000 records) #1

File Edit Generate

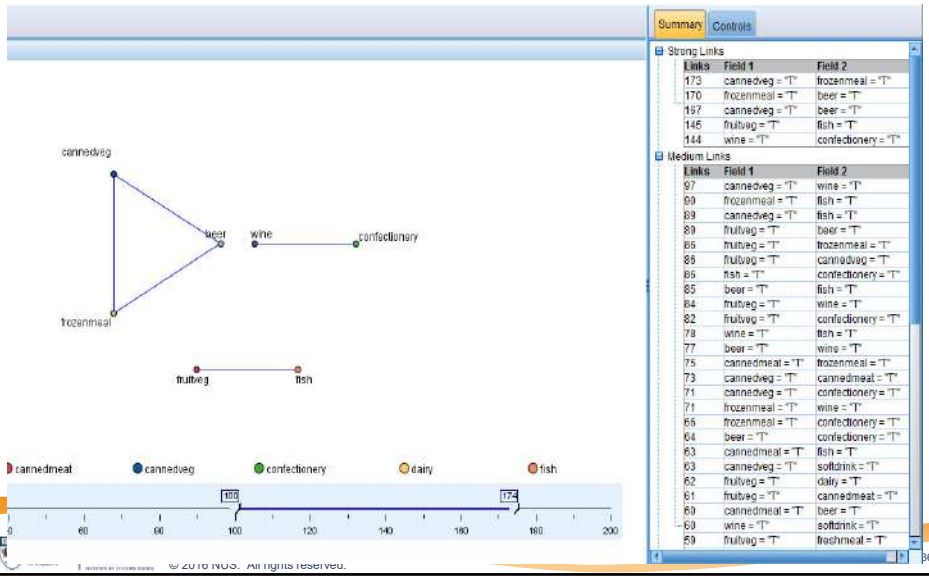
Table Annotations

	cardid	value	pmethod	sex	homeown	income	age	fruitveg	freshmeat	dairy	cannedveg	cannedmeat	frozenmeal
1	39808	42.712	CHEQUE	M	NO	27000	46	F	T	T	F	F	F
2	67362	25.357	CASH	F	NO	30000	28	F	T	F	F	F	F
3	10872	20.618	CASH	M	NO	13200	36	F	F	F	T	F	T
4	26748	23.088	CARD	F	NO	12200	26	F	F	T	F	F	F
5	91609	18.813	CARD	M	YES	11000	24	F	F	F	F	F	F
6	26630	46.487	CARD	F	NO	15000	35	F	T	F	F	F	F
7	62995	14.047	CASH	F	YES	20800	30	T	F	F	F	F	F
8	38765	22.203	CASH	M	YES	24400	22	F	F	F	F	F	F
9	28935	22.975	CHEQUE	F	NO	29500	46	T	F	F	F	F	T
10	41792	14.569	CASH	M	NO	29600	22	T	F	F	F	F	F
11	59480	10.328	CASH	F	NO	27100	18	T	T	T	T	F	F
12	80755	13.780	CASH	F	YES	20000	48	T	F	F	F	F	F
13	70998	36.509	CARD	M	YES	27300	43	F	F	T	F	T	T
14	80617	10.201	CHEQUE	F	YES	28000	43	F	F	F	F	F	F
15	61144	10.374	CASH	F	NO	27400	24	T	F	T	F	F	F
16	36405	34.822	CHEQUE	F	YES	18400	19	F	F	F	F	F	T
17	76567	42.248	CARD	M	YES	23100	31	T	F	F	T	F	F
18	85699	18.169	CASH	F	YES	27000	29	F	F	F	F	F	F
19	11357	10.753	CASH	F	YES	23100	26	F	F	F	F	F	F
20	97761	32.318	CARD	F	YES	25800	38	T	F	F	T	F	F
21	20362	31.720	CASH	M	YES	25100	38	F	F	F	F	F	T

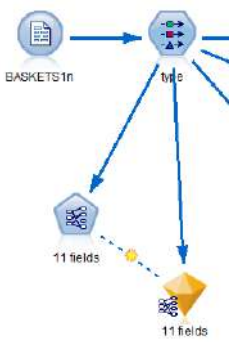
Link Analysis

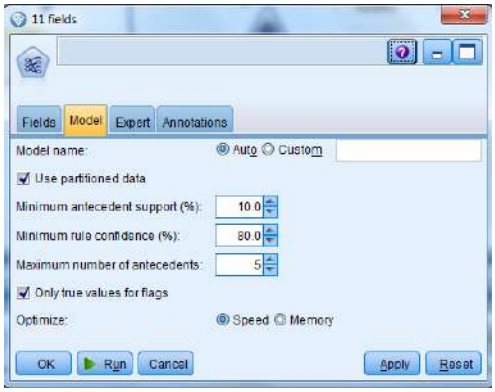


Revealing 3 Groups



Association Rules Modelling





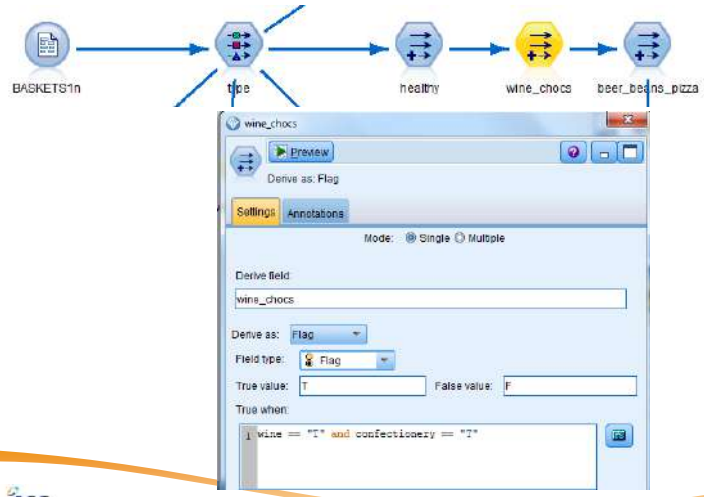
Rules Found

- Based on the given minimum antecedent support and minimum confidence

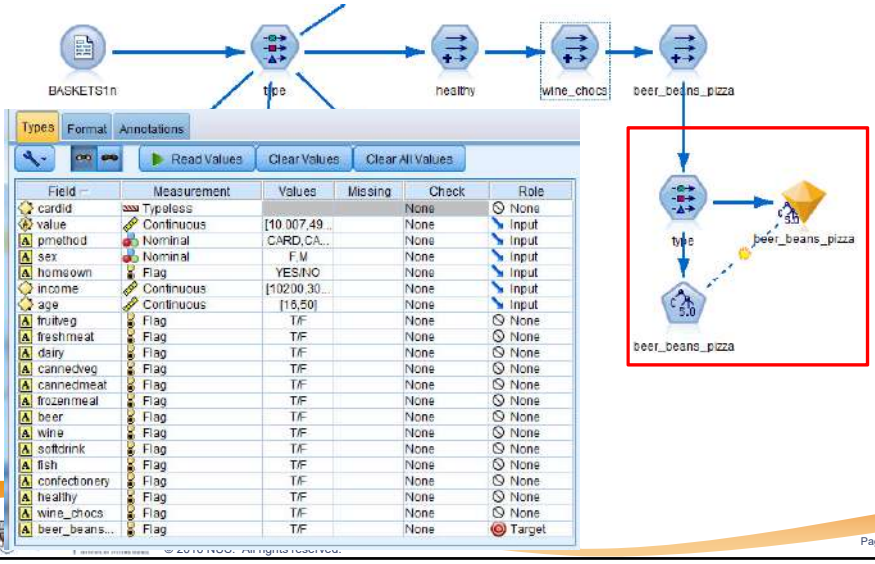
Consequent	Antecedent	Rule ID	Instances	Support %	Confidenc..	Rule Supp..	Lift	Deployability
frozenmeal	beer cannedveg	2	167	16.7	87.425	14.6	2.895	2.1
cannedveg	beer frozenmeal	1	170	17.0	85.882	14.6	2.834	2.4
beer	frozenmeal cannedveg	3	173	17.3	84.393	14.6	2.88	2.7

Deriving Group Variables

- Derive one variable for each group

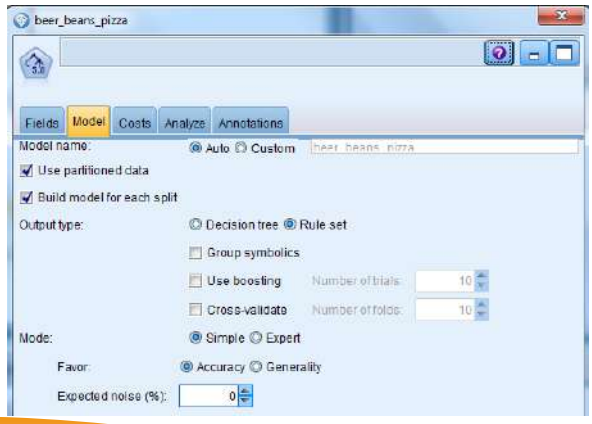


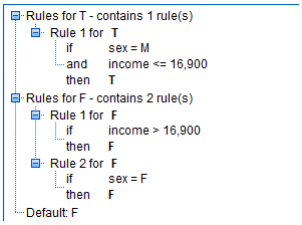
Rule Induction for Profiling



Rule Induction

- Information about this group of customers!





Introduction to R

A Tutorial Helps

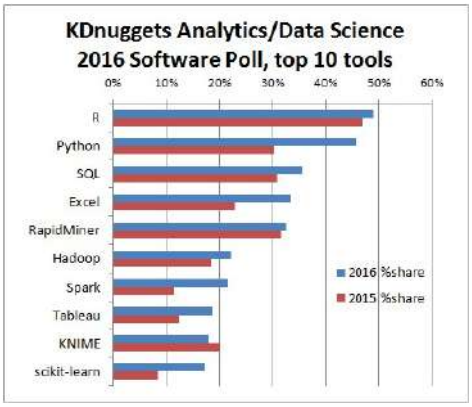
If you have never used R before, please take some time to get familiar with the language and its environment following this free online tutorial, Introduction to R (https://www.datacamp.com/courses/free-introduction-to-r).

You can download R from here: https://www.r-project.org/ . RStudio is a great open source editor for R. You can download it from here: https://www.rstudio.com/products/rstudio/download2/ .

- R is a language and environment for statistical computing and graphics, providing a wide variety of statistical, graphical, and data mining techniques
- Runs on Mac OS, Windows, Linux and Unix platforms
- Open source, licensed under GPL - http://www.r-project.org/
- Command line based, but highly extensible
- Lots of packages contributed by the open community, available at CRAN repositories
- Get the latest version from http://cran.r-project.org/bin/windows/base/

No R&D budget can compete with nearly ALL the statistics departments of the world and their professors working for free on this project.
A Ohri, author of *R for Business Analytics*

R as a Language for Data Analytics



- The most popular in KDnuggets' poll

http://www.kdnuggets.com/2016/06/r-python-top-analytics-data-mining-data-science-software.html

Main Features

- Extensive capabilities to interact with and pull data from databases (Oracle, MySQL, PostGresSQL, Hadoop-based data, etc)
- Advanced data visualisation through packages like *ggplot2*
- A vast array of statistical and data mining packages covering standard regression, decision trees, association rules, cluster analysis, machine learning, neural networks, etc
- GUI (Rattle) available for data miners using R
- Interfaces from almost all other analytical software including SAS, SPSS, JMP, Oracle Data Mining, RapidMiner, Excel, etc. (vendors viewing R as a complementary language)
- Flexible option for enterprise users from commercial vendors like Revolution Analytics
- Lots and lots of tutorials, codes, books available on web

GUIs

- Primarily command-line based
- For Windows users of Basic R, there's a simple GUI (to load package, install package, and set CRAN mirror for downloading packages)
- Other GUIs are available to make the use of R more convenient, such as
  - R Commander: more for statistics, plotting, time series
  - Sciviews-K: flexible GUI, can be used to create other GUIS
  - PKWard: comprehensive GUI with lots of details
  - Red-R: workflow style
  - R Analytic Flow: workflow style
  - Rattle: for data mining
  - PMG: simple interface
  - JGR/Deducer: more for data visualization
  - Grapher: simple graphing

IDEs

- Code editors or Integrated Development Environment (IDE) can make writing R codes easier for developers
  - Enhanced readability with syntax coloring
  - Automatic syntax error checking,
  - Auto code completion
  - Debugging facilities
- Examples
  - RStudio: most popular IDE for R, with code completion, syntax coloring, support for Latex
  - Notepad++: enhanced code editor for a variety of languages
  - TinnR: basic and easy-to-use code editor
  - Eclipse with StatET: R plugin for Eclipse, with support for Latex
  - Other code editors: Gvim, Highlight, etc.

Poll on R Interfaces

Which R interfaces do you use frequently?	
built-in R console (225)	40%
RStudio (135)	24%
Eclipse with StatET (90)	16%
RapidMiner R extension (80)	14.2%
Tinn-R (62)	11%
ESS (Emacs Speaks Statistics) (59)	10.5%
Rattle GUI (53)	9.4%
R Commander (43)	7.7%
Revolution Analytics (31)	5.5%
RKWard (22)	3.9%
JGR (Java Gui for R) (21)	3.7%
RExcel (18)	3.2%
R via a data mining tool plugin (12)	2.1%
Red-R (8)	1.4%
SciViews-R (6)	1.1%
Other (44)	7.8%

Google's R Style Guide

Summary: R Style Rules

1. File Names: end in `.R`
2. Identifiers: `variable.name` (or `variableName`), `FunctionName`, `kConstantName`
3. Line Length: maximum 80 characters
4. Indentation: two spaces, no tabs
5. Spacing
6. Curly Braces: first on same line, last on own line
7. else: Surround else with braces
8. Assignment: use `<-`, not `=`
9. Semicolons: don't use them
10. General Layout and Ordering
11. Commenting Guidelines: all comments begin with `#` followed by a space; inline comments need two spaces before the `#`
12. Function Definitions and Calls
13. Function Documentation
14. Example Function
15. TODO Style: `TODO(username)`

https://google.github.io/styleguide/Rguide.xml

Packages

- Currently >9000 packages from CRAN package repository, almost all free
- CRAN task views by subject areas

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Computational Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
Finance	Empirical Finance
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
MetaAnalysis	Meta-Analysis
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
NumericalMathematics	Numerical Mathematics
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods
Psychometrics	Psychometric Models and Methods
ReproducibleResearch	Reproducible Research
Robust	Robust Statistical Methods
SocialSciences	Statistics for the Social Sciences
Spatial	Analysis of Spatial Data
SpatioTemporal	Handling and Analyzing Spatio-Temporal Data
Survival	Survival Analysis
TimeSeries	Time Series Analysis
WebTechnologies	Web Technologies and Services
gR	gRaphical Models in R

Getting Started

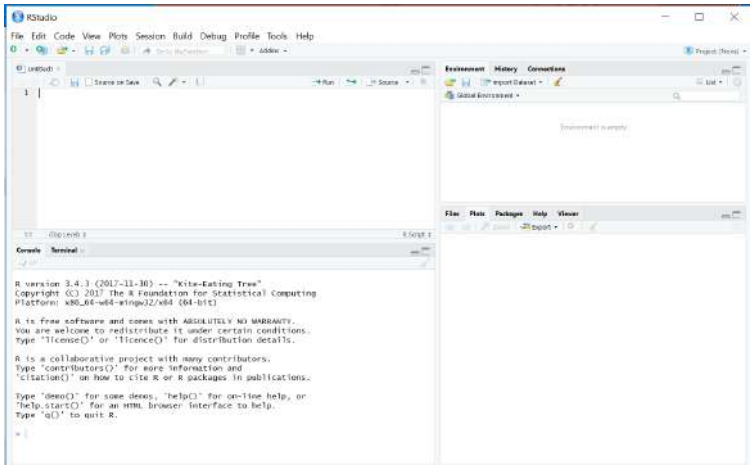
- The latest R basic can be downloaded from CRAN. Current version is 3.4.x
- Installation
  - Windows
    - Double click on the downloaded installer file, and follow the standard installation steps
    - Choose 32-bit or 64-bit based on your system's OS
  - Linux

    sudo apt-get update

    sudo apt-get install r-base

RStudio Gui

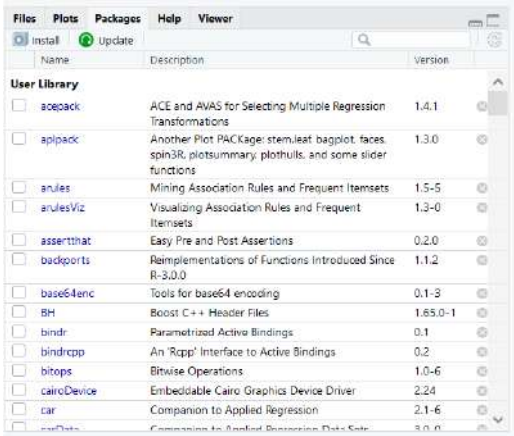
- Console, Script Editor, Graphics windows



Install a package

- Use command

  > install.packages("tm")



Load a package

- Use command

  > library(rattle)
- *Every time R is started, you need to reload the packages you want to use*.
- To update installed packages later on

  > update.packages()

Documentations on Packages

- Many packages have vignettes, prepared documentations (often in pdf)
- To see available vignettes

  vignette()
- To bring up the vignette about a package (it's fine to use single or double quote)

  vignette('tm') or *library(help="tm")*

  The pdf document will be displayed in a viewer like Adobe Acrobat Reader.

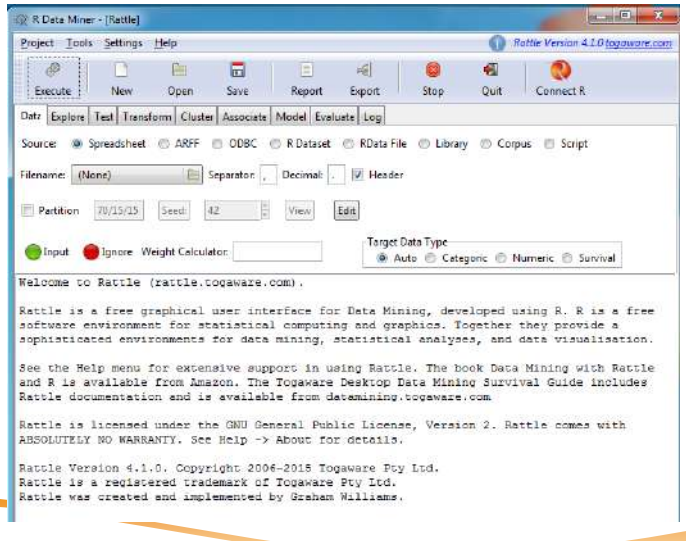
Rattle GUI

- **R A**nalytical **T**ool **T**o **L**earn **E**asily
- Popular GUI for data mining using R, good for R beginners
- Separate tabs for data import, summary, visualization, model building, clustering, association, and evaluation
- Log for R code is auto-generated → good for learning the codes!
- Can fall back to R when functions in Rattle are not sufficient
- An R package itself, therefore, to start Rattle:

  library(rattle)

  rattle()
- You'll be prompted to install many other packages that *rattle* is dependent on. Just agree (click 'yes') to install them all.

Rattle Interface



Rattle DM Tabs

- Tab-based interface
- Order of the tabs from left to right mimicking the typical data mining process
  - **Data** : to load/import data
    - supporting spreadsheets, ARFF, ODBC, R Dataset, R data file, datasets from R libraries, corpus, etc.
  - **Explore**: for data exploration
    - statistical summary, distribution, correlation analysis, principal components, interactive graphs
  - **Test**: for statistical testing
    - Two-sample tests (T-test, F-test, etc.), paired two-sample tests
  - **Transform**: data transformation
    - Scaling, imputation, recoding, cleanup

Rattle DM Tabs

- **Data mining tabs** (continued)
  - **Cluster**
    - K Means, Entropy-weighted K Means, hierarchical clustering, etc.
  - **Associate**
    - Apriori
  - **Model**
    - Decision tree, random forest, boosting, SVM, linear regression, neural network, survival regression
  - **Evaluate**
    - Confusion matrix, lift chart, ROC curve, cost curve, precision/recall, etc.
- **Log** tab – auto generated R codes

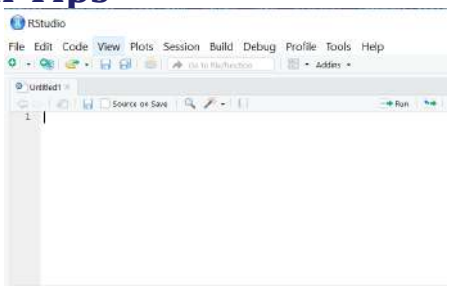
Some Useful Sites

- **Quick R** - lots of samples and short explanations
  - http://www.statmethods.net/
- **An Introduction to R -** accurate, up-to-date information from the R core Team
  - http://cran.r-project.org/doc/manuals/R-intro.pdf
- **Cookbook for R –** solutions to common tasks and problems in data analysis
  - http://www.cookbook-r.com/
- **R-bloggers** - for interesting and latest posts about R
  - http://www.r-bloggers.com
- **OnePageR** – A Survival Guide to Data Science with R
  - http://togaware.com/onepager/
- Find answers to common questions at
  - http://stackoverflow.com
  - http://stats.stackexchange.com/

And a lot more!

Basic R Programming

Useful Tips



- R is **case sensitive**
- **Use "/" in file pathname**

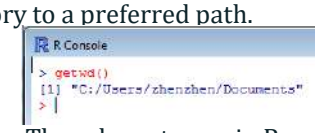
  E.g. *"C:/Users/zhenzhen/Documents"*
- Function calls need **( )**

  E.g. *rattle()*
- Often used interactively at command line. To repeat your last command, press the up arrow button "↑" on your keyboard
- You can also type the codes in a script window (R Editor).
  - Select the codes you want to run
  - Press "**Ctrl+Enter**" to run the line or selection
- The codes in R Editor can be saved as a script file (e.g. "mycode.R")
  - To run the script file: *source("mycode.R")*

R: Basic Stuff

- Useful commands for path checking and setting to make life easier later. Usually you want to set working directory to a preferred path.

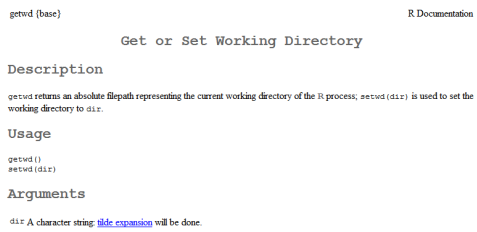
  getwd(), setwd()



- Use '**?**' before command name to get help. The relevant page in R Documentation will be opened in your browser

  ?setwd
- To quit R:

  q()


getwd {base} R Documentation

Get or Set Working Directory

Description

getwd returns an absolute filepath representing the current working directory of the R process; setwd(dir) is used to set the working directory to dir.

Usage

getwd()
setwd(dir)

Arguments

dir A character string; tilde expansion will be done.

Objects in R

- Data structures in R environment are objects created and stored by name.
- Variables are created by assigning (using "<-")some value to a variable name
- The collection of objects is called the *workspace*.
- To list objects in workspace

  ls() or *objects()*
- To remove an object: *rm(x)*
- To remove all objects: *rm(list=ls())*
- Objects in an R session can be stored permanently in a file (e.g. *workspace.RData*) for future use

  File -> Save Workspace… or *save.image('workspace.RData')*

  File -> Load Workspace… or *load ('workspace.Rdata')*

```
> x <- 3
> x
[1] 3
>
```


Common Operators & Functions

- Arithmetic operators

  *+, -, *, /, ^*
- Logical operators

  <, <=, >, >=, ==, !=, !, &, |
- Common functions

  log, exp, sin, cos, tan, sqrt

  max, min, range, sum, length, mean, var

```
> 3 + 4
[1] 7
> 3 < 4
[1] TRUE
```

Data Structures

- Use class() to find out the type of an object
- The class for simple objects and vectors is a *mode*, which could be ***integer***, ***numeric***, ***character***, ***logical***, ***list***.
- The classes *"**matrix**"*, *"**array**"*, *"**factor**"* and *"**data.frame**"* are composites of simpler objects.
- **Vector** – an ordered collection of items, a one-dimensional array

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> class(x)
[1] "integer"
```

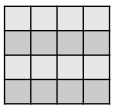
```
> y <- c('a', 'b', 'c', 'd', 'e')
> y
[1] "a" "b" "c" "d" "e"
> class(y)
[1] "character"
```

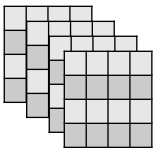
- Use **[ ]** to get subset of a vector

```
> y[3]
[1] "c"
> y[1:3]
[1] "a" "b" "c"
```


Matrix and Array

- **Matrix** – two-dimensional array **Array** – many number of dimensions





- Use *matrix()* to create matrices

```
> m <- matrix(1:24, 4, 6)
> m
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
```

- [] works with matrix too

```
> m[2,5]
[1] 18
```

```
> m[1:3, 2:4]
     [,1] [,2] [,3]
[1,]    5    9   13
[2,]    6   10   14
[3,]    7   11   15
```

- Negative index for removal of column/row

```
> m[-1, -6]
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    6   10   14   18
[2,]    3    7   11   15   19
[3,]    4    8   12   16   20
```

More about Matrix

- Most used in R
- To find out number of rows and columns
  dim() , or *nrow(), ncol()*

```
> dim(m)
[1] 4 6
> ncol(m)
[1] 6
> nrow(m)
[1] 4
```

- Columns and rows can be named, e.g.

```
> dimnames(m) = list(c('r1', 'r2', 'r3', 'r4'),
+ c('col1', 'col2', 'col3', 'col4', 'col5', 'col6'))
> m
   col1 col2 col3 col4 col5 col6
r1    1    5    9   13   17   21
r2    2    6   10   14   18   22
r3    3    7   11   15   19   23
r4    4    8   12   16   20   24
```

- Column/row names can be used to access the respective column/row

```
> m['r3',]
col1 col2 col3 col4 col5 col6
   3    7   11   15   19   23
```


Lists

- For vector, matrix, array, their values must all be of the same mode (type).
- For groupings of **different** R objects. Use *summary*() to get list information

```
> mylist <- list(x, y, m)
> mylist
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10

[[2]]
[1] "a" "b" "c" "d" "e"

[[3]]
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
```

```
> summary(mylist)
     Length Class  Mode
[1,] 10     -none- numeric
[2,]  5     -none- character
[3,] 24     -none- numeric
```

- [ ] bracketing is used for subsetting
- And [[ ]] for extracting a list element

```
> mylist[1]
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
> mylist[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10
```

Data Frames

- Matrix-like structures in which the columns can be of different types.
- Columns and rows can be named

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

- And accessed using "$"

```
> mtcars$gear
 [1] 4 4 4 3 3 3 3 4 4 4 4 3 3 3 3 3 3 4 4 4 3 3 3 3 3 4 5 5 5 5 5 4
```

- Use *attach()* to make the column/row names temporarily visible as variables without using *"mtcars$"*, till *detach()* is called. [Warning: R Style Guide discourages the use of attach().]

```
> attach(mtcars)
> gear
 [1] 4 4 4 3 3 3 3 4 4 4 4 3 3 3 3 3 3 4 4 4 3 3 3 3 3 4 5 5 5 5 5 4
> detach(mtcars)
> gear
Error: object 'gear' not found
```


Factors

- For categorical variables
- Use *as.factor()* when categories are represented as numbers

```
> as.factor(mtcars$cyl)
 [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
Levels: 4 6 8
> mtcars$cyl_f <- as.factor(mtcars$cyl)
> class(mtcars$cyl)
[1] "numeric"
> class(mtcars$cyl_f)
[1] "factor"
```

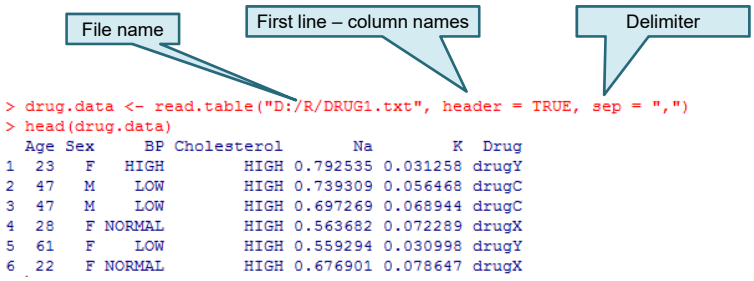
Creating a new column!

- The function *tapply()* can be used to apply a function *mean()* to each group of components' *mpg*, defined by the levels of *cyl_f*

```
> levels(mtcars$cyl_f)
[1] "4" "6" "8"
> mpg_means <- tapply(mtcars$mpg, mtcars$cyl_f, mean)
> mpg_means
       4        6        8
26.66364 19.74286 15.10000
```

Reading in Data

- *scan()* is the low level, more efficient function for reading data into R environment
- Function read.table() is more commonly used, returning a data frame




```
> drug.data <- read.table("D:/R/DRUG1.txt", header = TRUE, sep = ",")
> head(drug.data)
  Age Sex      BP Cholesterol       Na         K  Drug
1  23   F    HIGH        HIGH 0.792535 0.031258 drugY
2  47   M     LOW        HIGH 0.739309 0.056468 drugC
3  47   M     LOW        HIGH 0.697269 0.068944 drugC
4  28   F NORMAL        HIGH 0.563682 0.072289 drugX
5  61   F     LOW        HIGH 0.559294 0.030998 drugY
6  22   F NORMAL        HIGH 0.676901 0.078647 drugX
```


Read Data from Files

- Use *read.csv()* for comma-delimited files
- Use *read.delim()* for tab-delimited files
- Use *read.csv2()* or *read.delim2()* if comma is used to indicate decimal point in numbers (like in Europe)
- Use *read.fwf()* for files with fixed-width columns
- For MS Excel spreadsheets
  - Export them as .csv, then use *read.csv()*
  - Use packages like xlsReadWrite, XLConnect, RODBC, with their respective limitations

  See http://www.r-bloggers.com/read-excel-files-from-r/ for more details
- More functions available to read in data in various formats like ARFF, SAS, SPSS, etc.

To Get Data from Web

- Pass a URL to a suitable function which can handle this type of data
- For example, to get data from Singapore government's data site (install and load package "***jsonlite***" first)

```
> url <- "https://data.gov.sg/api/action/datastore_search?resource_id=76a8852a-093
a-4e8a-ad24-598f5e82c213"
> sgdata <- jsonlite::fromJSON(url)
> sgdata$result$records
    deaths ethnic_group live_births natural_increase year _id
1     8787      Chinese       35608            26821 1971   1
2     1464       Malays        7246             5782 1971   2
3      859      Indians        3090             2231 1971   3
4      219       Others        1144              925 1971   4
5     8905      Chinese       37797            28892 1972   5
6     1478       Malays        7594             6116 1972   6
7      895      Indians        3107             2212 1972   7
8      244       Others        1180              936 1972   8
```

- *Caution!: don't request for data too frequently when you get data from web, or you might be detected and blocked as suspected attack.*

To Get Data from Web

- For financial data, you can use package ***quantmod***, which make stock data scraping really a breeze

  install.packages('quantmod')

  library(quantmod)

  getSymbols("AAPL")

  getSymbols("GOOG")

- The returned is an XTS (Extensible Time Series) object, which can be converted into a data frame.

  GOOG

  class(GOOG)

```
> GOOG.df <- as.data.frame(coredata(GOOG))
> head(GOOG.df)
  GOOG.Open GOOG.High GOOG.Low GOOG.Close GOOG.Volume GOOG.Adjusted
1  231.4944  236.7899 229.0652   232.2842    15513200      232.2842
2  232.9847  240.4114 232.6618   240.0686    15877700      240.0686
3  239.6910  242.1749 237.5102   242.0209    13833500      242.0209
4  242.2693  243.3522 239.5420   240.2276     9570600      240.2276
5  241.1565  242.5475 239.0452   241.1814    10832700      241.1814
6  240.6498  245.1803 239.4625   243.1486    12014600      243.1486
> GOOG.df$Date <- index(GOOG)
> head(GOOG.df)
  GOOG.Open GOOG.High GOOG.Low GOOG.Close GOOG.Volume GOOG.Adjusted       Date
1  231.4944  236.7899 229.0652   232.2842    15513200      232.2842 2007-01-03
2  232.9847  240.4114 232.6618   240.0686    15877700      240.0686 2007-01-04
3  239.6910  242.1749 237.5102   242.0209    13833500      242.0209 2007-01-05
```

Write Data to Files

- Save the data as R object file (.Rdata)
  - E.g. *save(GOOG.df, file="GOOG.RData")*
  - Read the object back using *load()*
- Save the data to files in ASCII format using function *write.table()*

```
write.table(GOOG.df, file = "GOOG.csv", sep = ",", quote = FALSE,
row.names = FALSE, col.names = TRUE)
```

```
GOOG.Open,GOOG.High,GOOG.Low,GOOG.Close,GOOG.Volume,GOOG.Adjusted,Date
231.494354,236.789917,229.065155,232.28421,15513200,232.28421,2007-01-03
232.984665,240.411362,232.661758,240.068588,15877700,240.068588,2007-01-04
239.69104,242.174881,237.510223,242.020889,13833500,242.020889,2007-01-05
242.269272,243.352234,239.542007,240.227554,9570600,240.227554,2007-01-08
241.156509,242.54747,239.045242,241.181351,10832700,241.181351,2007-01-09
240.649811,245.180344,239.462524,243.14856,12014600,243.14856,2007-01-10
246.993546,249.253845,246.486847,248.245407,14510100,248.245407,2007-01-11
```

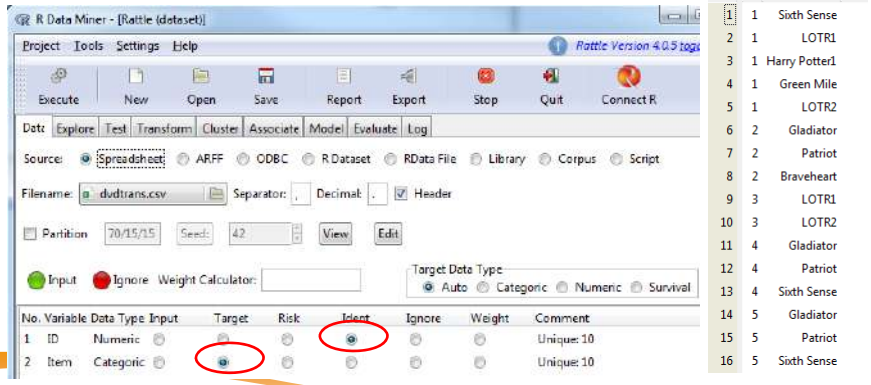

Summary

- We've learned the basics of R
- There's of course a lot more about R. References and resources are plenty on the web.
- The following modules will contain workshops in which we'll use R with Rstudio to perform text mining.

Workshop: Association Analysis with R & Rattle

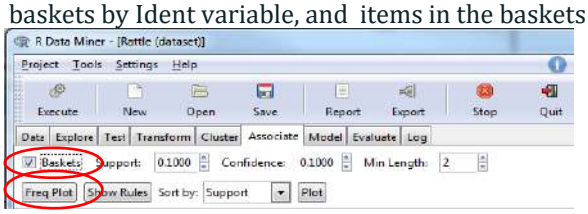
Association Analysis in Rattle

- It works with categorical variables, or "**Basket**" type of data, like "dvdtrans.csv" shown here
  - One variable for "Ident", and one for "Target"



Associate Tab

- On the Associate tab, check "Baskets". This will allow Rattle to identify the baskets by Ident variable, and items in the baskets by Target variable.



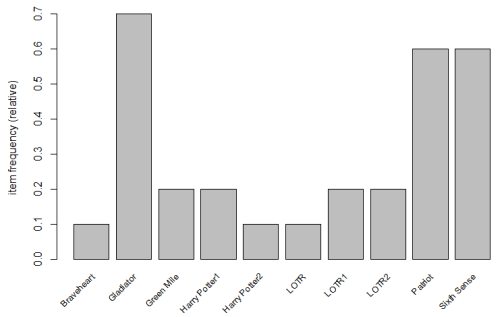
- clicking on button "Freq Plot"

```
# Generate a transactions dataset.

rs$transactions <- as(split(crs$dataset$Item, crs$dataset$ID), "transactions")

# Plot the relative frequencies.

itemFrequencyPlot(crs$transactions, support=0.1, cex=0.8)
```


Relative Item Frequency

- View the relative frequency of items (single)

```
itemFrequencyPlot(crs$transactions, support=0.1, cex=0.8)
```



Class "Transactions"

```
> class(crs$transactions)
[1] "transactions"
attr(,"package")
[1] "arules"
```

```
> inspect(crs$transactions)
     items                                                transactionID
[1]  {Green Mile,Harry Potter1,LOTR1,LOTR2,Sixth Sense}   1
[2]  {Braveheart,Gladiator,Patriot}                       2
[3]  {LOTR1,LOTR2}                                        3
[4]  {Gladiator,Patriot,Sixth Sense}                      4
[5]  {Gladiator,Patriot,Sixth Sense}                      5
[6]  {Gladiator,Patriot,Sixth Sense}                      6
[7]  {Harry Potter1,Harry Potter2}                        7
[8]  {Gladiator,Patriot}                                  8
[9]  {Gladiator,Patriot,Sixth Sense}                      9
[10] {Gladiator,Green Mile,LOTR,Sixth Sense}              10
```


Summary of data

- More information about the data set.

```
> summary(crs$transactions)
transactions as itemMatrix in sparse format with
 10 rows (elements/itemsets/transactions) and
 10 columns (items) and a density of 0.3

most frequent items:
    Gladiator       Patriot   Sixth Sense    Green Mile Harry Potter1       (Other)
            7             6             6             2             2             7

element (itemset/transaction) length distribution:
sizes
2 3 4 5
3 5 1 1
```

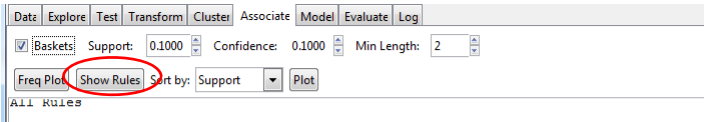
Association Results

- Click "Execute" at the Associate Tab to generate association rules

```
Summary of the Transactions:

      Length         Class          Mode
          10 transactions           S4

Summary of the Apriori Association Rules:

Number of Rules: 117

Summary of the Measures of Interestingness:

    support         confidence          lift
 Min.   :0.1000   Min.   :0.1429   Min.   : 0.7143
 1st Qu.:0.1000   1st Qu.:0.5000   1st Qu.: 1.6667
 Median :0.1000   Median :1.0000   Median : 2.5000
 Mean   :0.1316   Mean   :0.7980   Mean   : 3.1872
 3rd Qu.:0.1000   3rd Qu.:1.0000   3rd Qu.: 5.0000
 Max.   :0.6000   Max.   :1.0000   Max.   :10.0000
```


Association Rules

- Click on "Show Rules" button to show the generated association rules, sorted by confidence




```
All Rules

    lhs                                 rhs                 support confidence lift
37  {Patriot}                        => {Gladiator}         0.6     1.0000000  1.4285714
38  {Gladiator}                      => {Patriot}           0.6     0.8571429  1.4285714
39  {Sixth Sense}                    => {Gladiator}         0.5     0.8333333  1.1904762
40  {Gladiator}                      => {Sixth Sense}       0.5     0.7142857  1.1904762
35  {Patriot}                        => {Sixth Sense}       0.4     0.6666667  1.1111111
36  {Sixth Sense}                    => {Patriot}           0.4     0.6666667  1.1111111
86  {Patriot,Sixth Sense}            => {Gladiator}         0.4     1.0000000  1.4285714
87  {Gladiator,Patriot}              => {Sixth Sense}       0.4     0.6666667  1.1111111
88  {Gladiator,Sixth Sense}          => {Patriot}           0.4     0.8000000  1.3333333
21  {LOTR1}                          => {LOTR2}             0.2     1.0000000  5.0000000
22  {LOTR2}                          => {LOTR1}             0.2     1.0000000  5.0000000
31  {Green Mile}                     => {Sixth Sense}       0.2     1.0000000  1.6666667
32  {Sixth Sense}                    => {Green Mile}        0.2     0.3333333  1.6666667
1   {Harry Potter2}                  => {Harry Potter1}     0.1     1.0000000  5.0000000
2   {Harry Potter1}                  => {Harry Potter2}     0.1     0.5000000  5.0000000
3   {Braveheart}                     => {Patriot}           0.1     1.0000000  1.6666667
```

```
# List rules.

inspect(sort(crs$apriori, by="support"))
```

Sort Using Lift

- We can sort the rules using other criteria, like Lift, and view the top 10 rules

  inspect(sort(crs$apriori, by="lift")[1:10])

```
> inspect(sort(crs$apriori, by="lift")[1:10])
   lhs                 rhs             support confidence lift
1  {Gladiator,
    Green Mile}     => {LOTR}             0.1        1.0   10
2  {Gladiator,
    Green Mile,
    Sixth Sense}    => {LOTR}             0.1        1.0   10
3  {Harry Potter2} => {Harry Potter1}    0.1        1.0    5
4  {Harry Potter1} => {Harry Potter2}    0.1        0.5    5
5  {LOTR}          => {Green Mile}       0.1        1.0    5
6  {Green Mile}    => {LOTR}             0.1        0.5    5
7  {LOTR1}         => {LOTR2}            0.2        1.0    5
8  {LOTR2}         => {LOTR1}            0.2        1.0    5
9  {LOTR,
    Sixth Sense}    => {Green Mile}       0.1        1.0    5
10 {Green Mile,
    Sixth Sense}    => {LOTR}             0.1        0.5    5
```


Another Try

- Let's try with a larger dataset "Groceries", in "**transaction**" format
- Use inspect() to see the content. It's not a data frame!

```
> inspect(Groceries)
```

install.packages("arules")
library(arules)
data(Groceries)

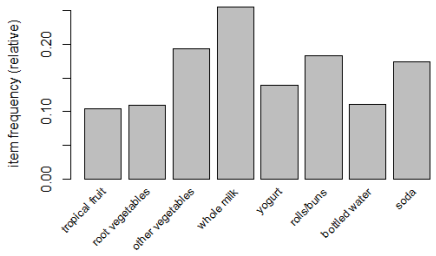
- Get a summary of the data

```
> summary(Groceries)
transactions as itemMatrix in sparse format with
 9835 rows (elements/itemsets/transactions) and
 169 columns (items) and a density of 0.02609146

most frequent items:
      whole milk other vegetables       rolls/buns             soda
            2513             1903             1809             1715
          yogurt          (Other)
            1372            34055

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55   46
  17   18   19   20   21   22   23   24   26   27   28   29   32
  29   14   14    9   11    4    6    1    1    1    1    3    1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   3.000   4.409   6.000  32.000
```

```
9832 {cooking chocolate}
9833 {chicken,
      citrus fruit,
      other vegetables,
      butter,
      yogurt,
      frozen dessert,
      domestic eggs,
      rolls/buns,
      rum,
      cling film/bags}
9834 {semi-finished bread,
      bottled water,
      soda,
      bottled beer}
9835 {chicken,
      tropical fruit,
      other vegetables,
      vinegar,
      shopping bags}
```

Item Frequency Plot

itemFrequencyPlot(Groceries, support = 0.1, cex.names=0.8)



Frequent Item Sets

- Use ECLAT to get the frequent item sets

  fsets = eclat(Groceries, parameter = list(support = 0.05), control = list(verbose=FALSE))

- Then get a summry

```
> summary(fsets)
set of 31 itemsets

most frequent items:
      whole milk other vegetables           yogurt       rolls/buns
               4                2                2                2
     frankfurter          (Other)
               1               23

element (itemset/transaction) length distribution:sizes
 1  2
28  3

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.000   1.000   1.097   1.000   2.000
```

Inspect Item Sets

- Inspect single item sets, multi item sets

```
> singleItems = fsets[size(items(fsets)) == 1];
> inspect(singleItems)
   items                    support
1  {whole milk}          0.25551601
2  {other vegetables}    0.19349263
3  {rolls/buns}          0.18393493
4  {yogurt}              0.13950178
5  {soda}                0.17437722
6  {root vegetables}     0.10899847
7  {tropical fruit}      0.10493137
8  {bottled water}       0.11052364
9  {sausage}             0.09395018
10 {shopping bags}       0.09852567
11 {citrus fruit}        0.08276563
12 {pastry}              0.08896797
13 {pip fruit}           0.07564820
```

```
> multiItems = fsets[size(items(fsets)) > 1]
> inspect(multiItems)
  items                    support
1 {whole milk,
   yogurt}           0.05602440
2 {whole milk,
   rolls/buns}       0.05663447
3 {other vegetables,
   whole milk}       0.07483477
```


Run Apriori

- Call apriori() (you'll need packa "*arules*", "*arulesViz*")

```
> Grules = apriori(Groceries, parameter=list(support=0.01, confidence=0.5))

parameter specification:
 confidence minval smax arem  aval originalSupport support minlen maxlen target
        0.5    0.1    1 none FALSE            TRUE    0.01      1     10  rules
   ext
 FALSE

algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE
```

See the rules

- Inspect the rules

```
> inspect(sort(Grules, by="lift"))
   lhs                      rhs                    support confidence     lift
1  {citrus fruit,
    root vegetables}     => {other vegetables} 0.01037112  0.5862069 3.029608
2  {tropical fruit,
    root vegetables}     => {other vegetables} 0.01230300  0.5845411 3.020999
3  {root vegetables,
    rolls/buns}          => {other vegetables} 0.01220132  0.5020921 2.594890
4  {root vegetables,
    yogurt}              => {other vegetables} 0.01291307  0.5000000 2.584078
5  {curd,
    yogurt}              => {whole milk}       0.01006609  0.5823529 2.279125
```


Access Rule Quality Measures

- The support, confidence, lift measures can be obtained using quality()

```
> quality(Grules)
      support confidence     lift
1  0.01006609  0.5823529 2.279125
2  0.01148958  0.5736041 2.244885
3  0.01230300  0.5525114 2.162336
4  0.01087951  0.5245098 2.052747
5  0.01464159  0.5070423 1.984385
6  0.01352313  0.5175097 2.025351
```

Association Rules Manipulation

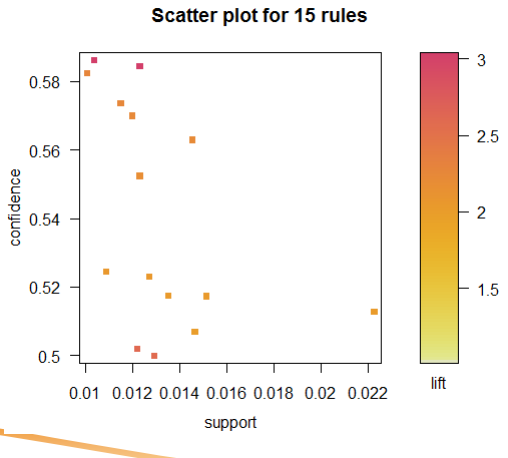
- Select rules based on conditions

```
> subrules = Grules[quality(Grules)$confidence > 0.55]
> inspect(subrules)
  lhs                     rhs                    support    confidence     lift
1 {curd,
   yogurt}             => {whole milk}       0.01006609  0.5823529 2.279125
2 {other vegetables,
   butter}             => {whole milk}       0.01148958  0.5736041 2.244885
3 {other vegetables,
   domestic eggs}      => {whole milk}       0.01230300  0.5525114 2.162336
4 {citrus fruit,
   root vegetables}    => {other vegetables} 0.01037112  0.5862069 3.029608
5 {tropical fruit,
   root vegetables}    => {other vegetables} 0.01230300  0.5845411 3.020999
6 {tropical fruit,
   root vegetables}    => {whole milk}       0.01199797  0.5700483 2.230969
7 {root vegetables,
   yogurt}             => {whole milk}       0.01453991  0.5629921 2.203354
```


Plotting Rules

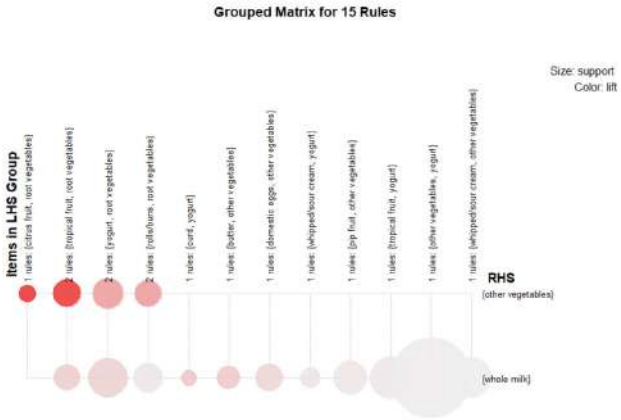
- Limited plotting supported, mainly about rules quality measures

plot(Grules)



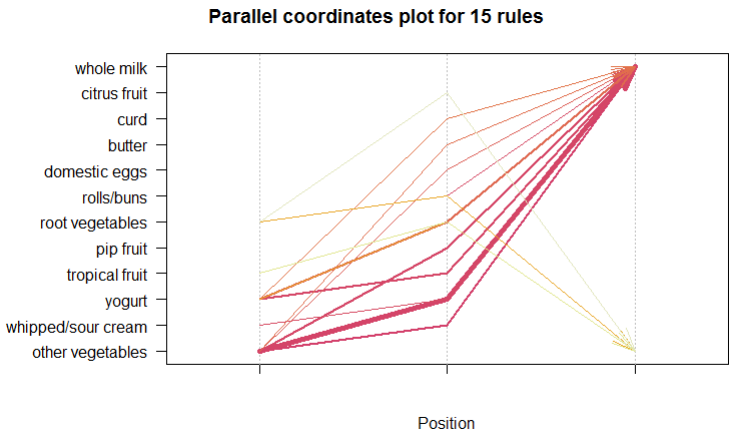
Grouped Plot

plot(Grules, method="grouped")



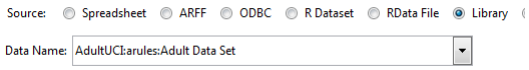
Parallel Coordinates Plot

plot(Grules, method="paracoord")



Exercise

- Load dataset "AdultUCI" into Rattle



- Run association analysis over the dataset.

- A fun article to read:

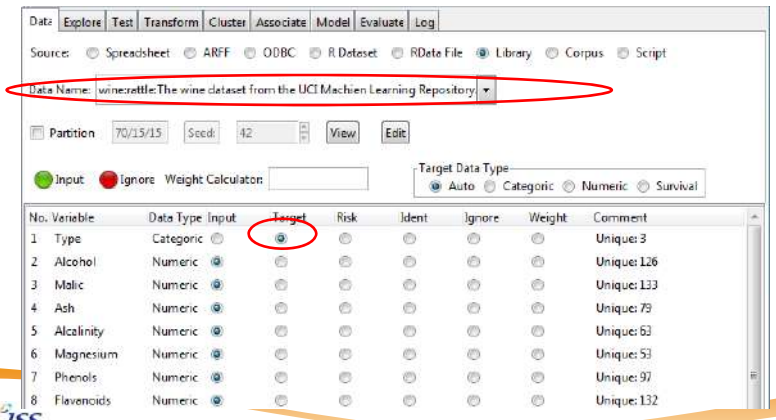
  How Companies Learn Your Secrets

  (http://www.nytimes.com/2012/02/19/magazine/shopping-habits.html?pagewanted=all&_moc.semityn.www&_r=0)

Workshop: Cluster Analysis

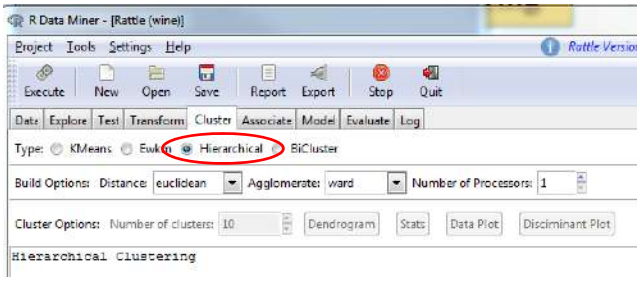
Data

- Use "wine" dataset from rattle package, with no partition, keep "Type" as Target, others as Input.
- Explore your data first for understanding.



Hierarchical Clustering

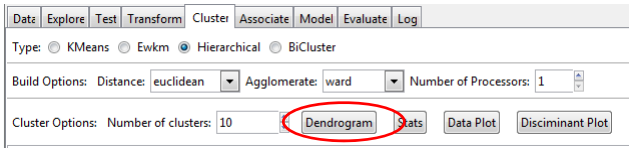
- All the input variables are numerical variables. So they will be all used for clustering. Let's try hierarchical clustering with all inputs in original scales.

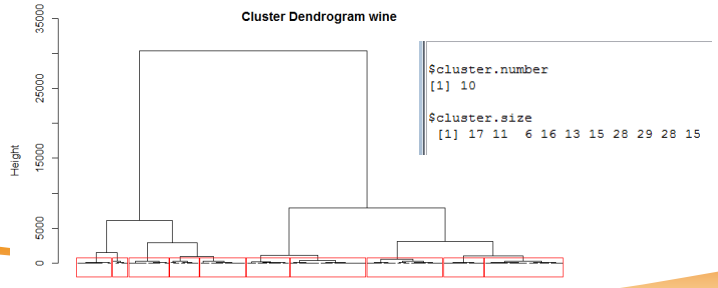


- click "Execute"

Dendrogram

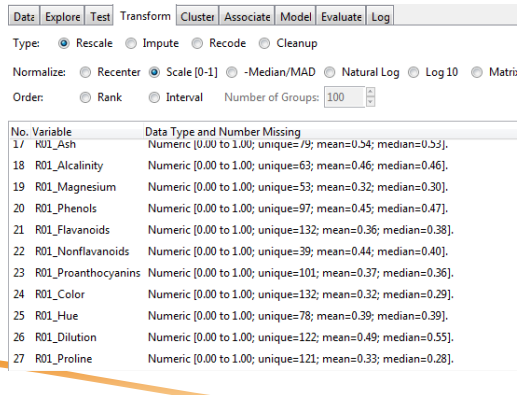
- Plot dendrogram. What would be a good number of clusters in the data?





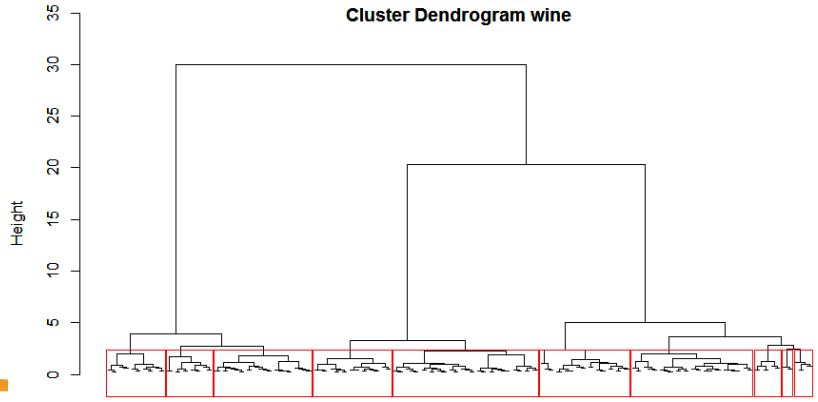
Oops!

- What have we left out? Scaling!
- Clustering works better when data is scaled to a common range.
- Let's rescale our data first to [0, 1]



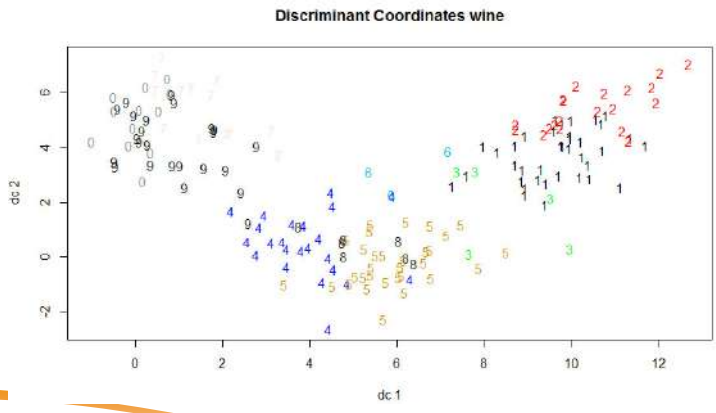
Cluster With Scaled Data

- Let's do hierarchical clustering again, and plot the dendrogram.
- Is there any difference in the dendrogram? How many clusters would you say now?



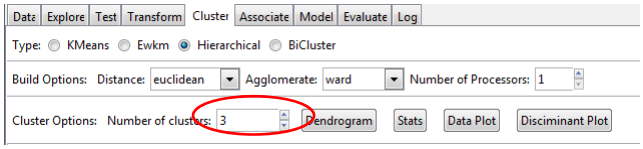
Discriminant Plot

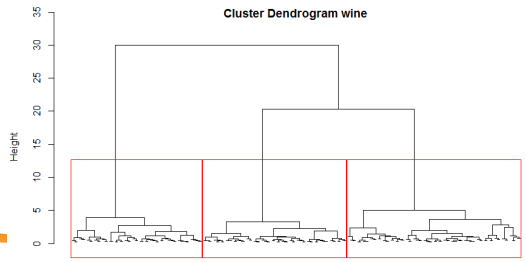
- Click on "Discriminant Plot" button to display how the clusters are distributed across the data. Do you see really 10 clusters?



Cluster Again

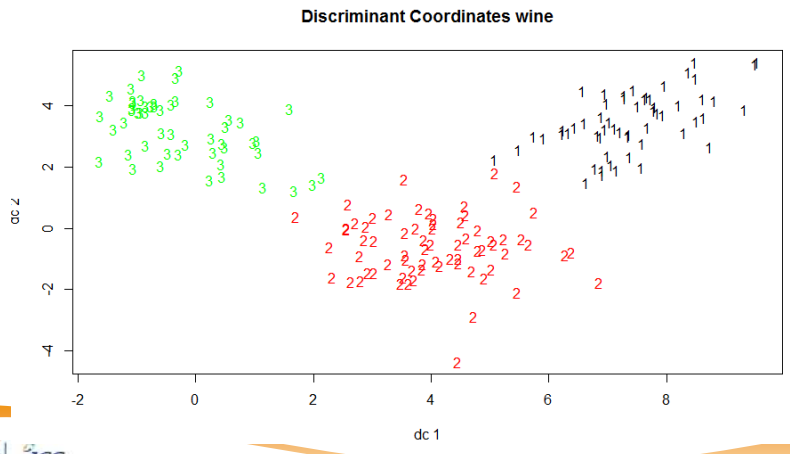
- Change the "Number of clusters" in Cluster Options to be 3.
- And Execute again





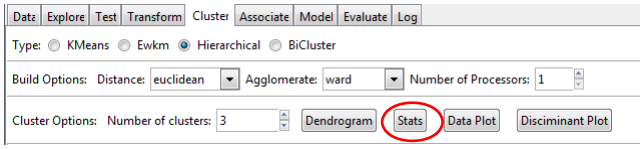
Discriminant Plot

- Bring out the "Discriminant Plot" again. Does it make more sense now?



See More Stats

- Click on "Stats" button to see more details of the clustering results.




```
Cluster means:

     R01_Alcohol R01_Malic   R01_Ash R01_Alcalinity R01_Magnesium R01_Phenols
[1,]   0.7199446 0.2489772 0.5748194      0.3210345     0.3893974   0.6429522
[2,]   0.3321510 0.2285043 0.4834535      0.5013447     0.2714241   0.4608196
[3,]   0.5452429 0.5038005 0.5708556      0.5523394     0.3196070   0.2360743
     R01_Flavanoids R01_Nonflavanoids R01_Proanthocyanins R01_Color   R01_Hue
[1,]      0.5596639         0.2926183           0.4701422 0.3672086 0.4725431
[2,]      0.3815814         0.4418923           0.3919444 0.1545482 0.4789678
[3,]      0.1001298         0.5903483           0.2397476 0.4958815 0.1752033
     R01_Dilution R01_Proline
[1,]    0.6926933   0.6058513
[2,]    0.5736051   0.1745539
[3,]    0.1618766   0.2491084

$cluster.number
[1] 3

$cluster.size
[1] 57 69 52
```


Is the result good?

- Click on "Stats" button to see more details of the clustering results.
- **Sum of Square of within cluster distance** (the distance between each observation and the cluster center), the smaller the better
- **Silhouette**: a method of interpretation and validation of clusters, measuring how tightly grouped the data in a cluster are.

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

  - *a(i)*: the average dissimilarity of *i* with all other data within the same cluster, the smaller the better
  - *b(i)*: the lowest average dissimilarity of *i* to any other cluster, the higher the better
  - -1 ≤ s(i) ≤ 1, the higher the better

Compare the Stats

- From the previous clustering (10 clusters)

```
$within.cluster.ss
[1] 33.43583

$clus.avg.silwidths
         1          2          3          4          5          6          7          8
0.13765918 0.15594389 0.07983620 0.09183144 0.20408111 0.39567200 0.12686515 0.15483978
         9         10
0.19992567 0.11440522

$avg.silwidth
[1] 0.1550279
```

- From this clustering (3 clusters)

```
$within.cluster.ss
[1] 49.85874

$clus.avg.silwidths
         1          2          3
0.4120450 0.1441298 0.3601732

$avg.silwidth
[1] 0.2930367
```


K-Means

- Choose KMeans and set the number of clusters as 3. Execute and check the cluster sizes and cluster centers.


```
Cluster sizes:

[1] "62 51 65"
```

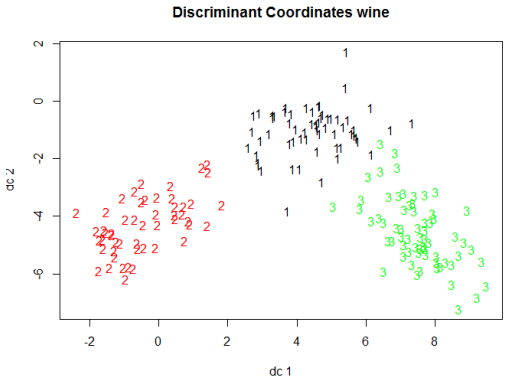
```
Cluster centers:

  R01_Alcohol R01_Malic   R01_Ash R01_Alcalinity R01_Magnesium R01_Phenols
1   0.3086163 0.2384929 0.4758496      0.4954273     0.2549088   0.4209677
2   0.5537152 0.5073626 0.5655867      0.5485143     0.3115942   0.2427316
3   0.6912955 0.2383703 0.5763060      0.3526566     0.3976589   0.6498674
  R01_Flavanoids R01_Nonflavanoids R01_Proanthocyanins R01_Color   R01_Hue R01_Dilution
1      0.3583776         0.4510043           0.3778875 0.1424364 0.4686074    0.5608531
2      0.1010176         0.6074732           0.2321395 0.5080807 0.1723258    0.1562882
3      0.5548523         0.2911466           0.4775540 0.3482673 0.4808005    0.6882502
  R01_Proline
1   0.1602779
2   0.2432659
3   0.5721168
```

K-Means Stats

- How good is the result?
- Click on "Stats" to find out!

```
Within cluster sum of squares:

[1] 20.79043 13.19359 15.00140
```

```
$within.cluster.ss
[1] 48.98541

$clus.avg.silwidths
        1         2         3
0.1912019 0.3613687 0.3548266

$avg.silwidth
[1] 0.2997082
```



Nice clusters, but…?

- In this dataset, we have a variable Type. Let's see how well the clusters tally with Type categories.

require(descr, quietly=TRUE)

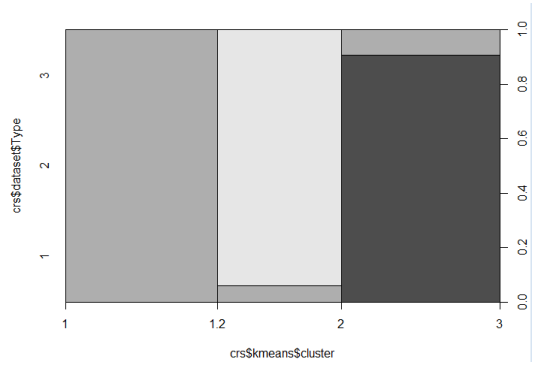
CrossTable(crs$dataset$Type, crs$kmeans$cluster)

crs$dataset$Type	crs$kmeans$cluster 1	2	3	Total
1	0	0	59	59
	20.551	16.904	65.114	
	0.000	0.000	1.000	0.331
	0.000	0.000	0.908	
	0.000	0.000	0.331	
2	62	3	6	71
	56.167	14.785	15.315	
	0.873	0.042	0.085	0.399
	1.000	0.059	0.092	
	0.348	0.017	0.034	
3	0	48	0	48
	16.719	85.282	17.528	
	0.000	1.000	0.000	0.270
	0.000	0.941	0.000	
	0.000	0.270	0.000	
Total	62	51	65	178
	0.348	0.287	0.365	

Plot Type and Clusters

- Visualise the information instead…

  plot(crs$dataset$Type ~ crs$kmeans$cluster)



Visualize Cluster Info

- Well, there's no directly available button, but…
- Remember our visualizing exercises? And don't forget we can learn from the codes in Log!
- For example: