DEFENCE INSTITUTE OF ADVANCED TECHNOLOGY

DEEMED UNIVERSITY

शास्त्रेण रक्षाम्

शस्त्रं प्रकरोति

Machine Learning for Cyber Security (CS-602)

L#02

Introduction – R Programming

By

Dr Sunita Dhavale

Syllabus

- Data Analytics Foundations: R programming, Python Basics -Expressions and Variables, String Operations, Lists and Tuples, Sets, Dictionaries Conditions and Branching, Loops, Functions, Objects and Classes, Reading/Writing files, Hand ling data with Pandas, Scikit Library, Numpy Library, Matplotlib, scikit programming for data analysis, setting up lab environment, study of standard datasets. Introduction to Machine Learning- Applications of Machine Learning, Supervised, unsupervised classification and regression analysis
- Python libraries suitable for Machine Learning Feature Extraction. Data pre-processing, feature analysis etc., Dimensionality Reduction & Feature Selection Methods, Linear Discriminant Analysis and Principal Component Analysis, tackle data class imbalance problem

Syllabus

- Supervised and regression analysis, Regression, Linear Regression, Non-linear Regression, Model evaluation methods, Classification, K-Nearest Neighbor, Naïve Bayes, Decision Trees, Logistic Regression, Support Vector Machines, Artificial Neural Networks, Model Evaluation. Ensemble Learning, Convolutional Neural Networks, Spectral Embedding, Manifold detection and Anomaly Detection
- Unsupervised classification K-Means Clustering, Hierarchical Clustering, Density-Based Clustering, Recommender Systems-Content-based recommender systems, Collaborative Filtering, machine learning techniques for standard dataset, ML applications, Case studies on Cyber Security problems that can be solved using Machine learning like Malware Analysis, Intrusion Detection, Spam detection, Phishing detection, Financial Fraud detection, Denial of Service Detection.

Text/Reference Books

1. Building Machine Learning Systems with Python – Willi Richert, Luis Pedro Coelho
2. Alessandro Parisi, Hands-On Artificial Intelligence for Cybersecurity: Implement smart AI systems for preventing cyber attacks and detecting threats and network anomalies Publication date :Aug 2, 2019, Packt, ISBN-13, 9781789804027
3. Machine Learning: An Algorithmic Perspective – Stephen Marsland
4. Sunita Vikrant Dhavale, "Advanced Image-based Spam Detection and Filtering Techniques", IGI Global, 2017
5. Soma Halder , Sinan Ozdemir, Hands-On Machine Learning for Cybersecurity: Safeguard your system by making your machines intelligent using the Python ecosystem, By Publication date : Dec 31, 2018, Packt, ISBN-13 :9781788992282

1. Stuart Russell, Peter Norvig (2009), "Artificial Intelligence – A Modern Approach", Pearson Elaine Rich & Kevin Knight (1999), "Artificial Intelligence", TMH, 2nd Edition
2. NP Padhy (2010), "Artificial Intelligence & Intelligent System", Oxford
3. ZM Zurada (1992), "Introduction to Artificial Neural Systems", West Publishing Company
4. Research paper for study (if any) – White papers on multimedia from IEEE/ACM/Elsevier/Spinger/ Nvidia sources.

Lab assignments

1	**Python Programming part-1**
2	Python Programming part-2
3	Study and Implement Linear Regression Algorithm for any standard dataset like in cyber security domain
4	Study and Implement KMeans Algorithm for any standard dataset in cyber security domain
5	Study and Implement KNN for any standard dataset in cyber security domain
6	Study and Implement ANN for any standard dataset in cyber security domain
7	Study and Implement PCA for any standard dataset in cyber security domain
8	Case Study: Use of ML along with Fuzzy Logic/GA to solve real world Problem in cyber security domain
9	Mini assignment: Apply ML along with PSO/ACO to solve any real world problem in cyber security domain
10	ML Practice Test – 1 Quiz

Defence Institute of Advanced Technology

School of Computer Engineering & Mathematical Sciences

SEMESTER-I TIME TABLE (AUTUMN 2024)$

PROGRAMMES: (I) CS [M.TECH IN CYBER SECURITY] (II) AI [M.TECH CSE (ARTIFICIAL INTELLIGENCE)] BATCH: 2024-2026

Lecture / Day	L1 0900-1000	L2 1000-1100	L3 1100-1200	L4 1200-1300	Lunch Break 1300-1400	L4 1400-1500	L4 1500-1600	L4 1600-1700	L4 1700-1800
Monday	CE-602 (AI) CS-602 (CS)	CE-604 (AI) CS-603 (CS)	CE-601 (AI) CS-604 (CS)	CE-601 (AI)		LAB CE-601 (AI) LAB CS-602 (CS)		AM607	
Tuesday	CE-603 (AI) LAB CS-603 (CS)	CE-602 (AI) CS-602 (CS)	CE-601 (AI) CS-605 (CS)	CE-604 (AI) CS-604 (CS)		PGC 601		AM607	LAB CS-603 (CS)
Wednesday	CE-604 (AI) CS-605 (CS)	CE-603 (AI) CS-602 (CS)	CE-602 (AI) CS-603 (CS)	LAB CE-604 (AI) CS-604 (CS)		CE-605(AI) LAB CS-605 (CS)	LAB CS-605 (CS)	AM607	LAB CE-604 (AI)
Thursday	CE-604 (AI) CS-603 (CS)	CS-605 (CS)	LAB CE-602 (AI) CS-601 (CS)	CE-603 (AI) CS-601 (CS)		PGC 601		AM607	
Friday	LAB CE-603 (AI) LAB CS-601 (CS)		LAB CE-602 (AI) CS-601 (CS)	LAB CS-604 (CS)		CE-605(AI) LAB CS-604 (CS)	CE-605(AI)	LAB CE-605(AI)	

COURSE CODE & COURSE NAME		FACULTY
Programme: CS [M.Tech in Cyber Security] Classroom: Arjun	Programme: AI [M.Tech CSE (Artificial Intelligence)] Classroom: Kaveri	
CS-601 Data Security & Privacy	CE-601 Responsible Artificial Intelligence;	MJN: Dr. Manisha J. Nene
CS-602 ML for Cyber Security	CE-604 Practical Machine Learning;	SVD: Dr. Sunita V. Dhavale
CS-605 Network and Cloud Security	CE-602 Intelligent Algorithms	CRS: Prof. CRS Kumar
CS-604 Advanced System Security	---------	DVV: Dr. Deepti V. Vidyarthi
CS-603 Applied Cryptography	---------	AM: Dr. Arun Mishra
---------	CE-603 Deep Neural Network;	US: Dr. Upasna Singh
---------	CE-605 Mathematics for ML;	Unit-2: Dr Upasna, Unit 4: Dr Sunita, Unit3:MJN, Unit 1: Faculty To be Nominated
AM-607 Mathematics for Engineers	AM-607 Mathematics for Engineers	OO/DS/DP: Dr Odellu O., Dr Dasari S., Dr. Debasis P.
PGC-601 Research Methodology	PGC-601 Research Methodology	Common Subject for All

$ TENTATIVE T.T. SUBJECT TO CHANGE

Program Coordinator,
M.Tech (CS & AI), Batch 2024-26

Director, SoCE&MS

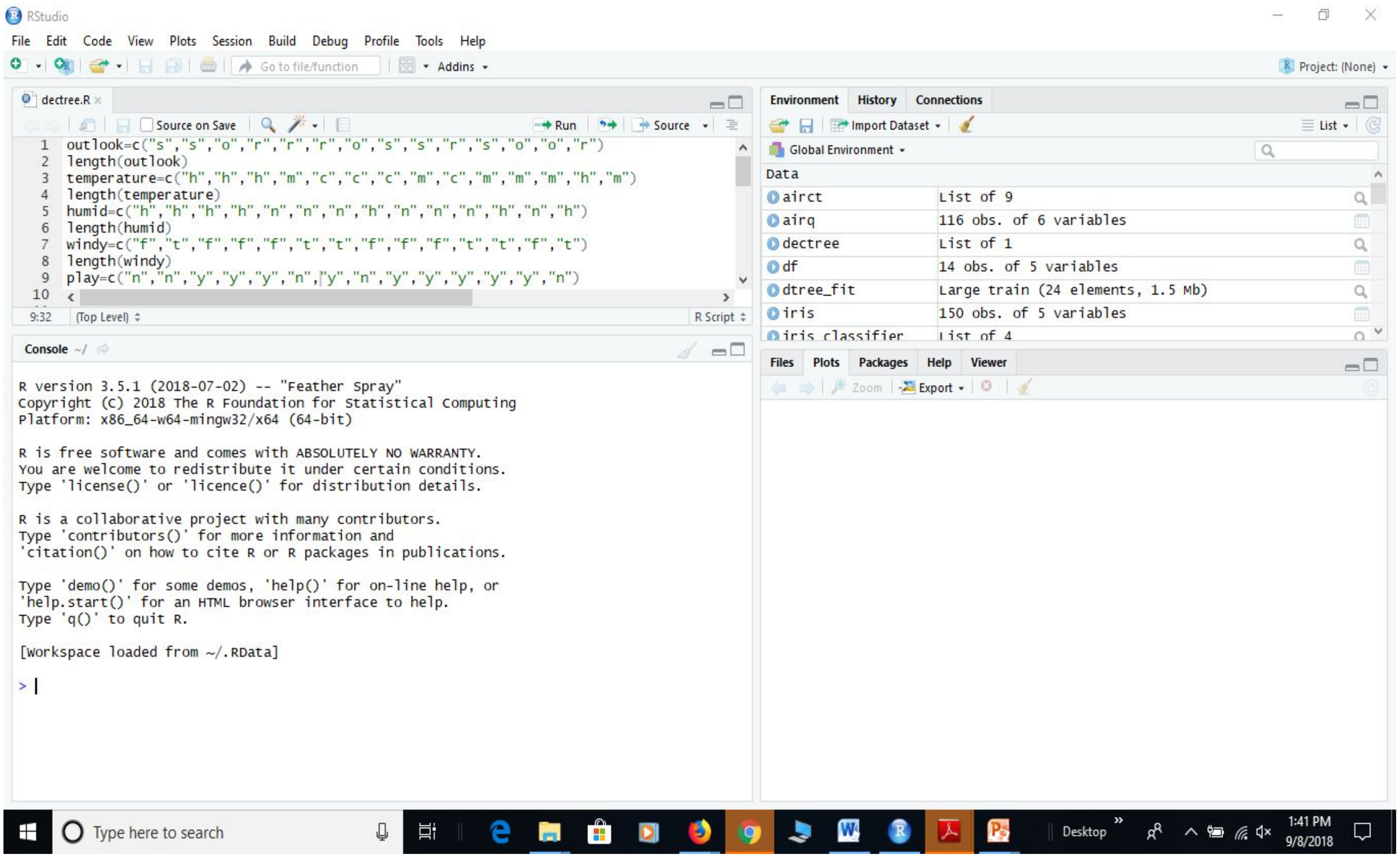
Basic R Programming

R

- R, as a dialect of GNU-S, is a powerful statistical language that can be used to manipulate and analyze data.
- R provides many machine learning packages and visualization functions
- R is open source and free.
- Using R greatly simplifies machine learning.
- R is an interpreted language also means that you can execute R commands directly and see an immediate result.
- All you need to know is how each algorithm can solve your problem, and then you can simply use a written package to quickly generate prediction models on data with a few command lines.
- Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. Journal of Computational and Graphical Statistics, 5(3):299–314, 1996

R with R-Studio

- set up the R environment and integrated development environment, RStudio.
- Official website (http://www.rproject.org/).
- You may select the mirror location closest to you.
- *https://www.rstudio.com/*
- RStudio provides comprehensive facilities for software development. Built-in features such as syntax highlighting, code completion, and smart indentation help maximize productivity.



```
outlook=c("s","s","o","r","r","r","o","s","s","r","s","o","o","r")
length(outlook)
temperature=c("h","h","h","m","c","c","c","m","c","m","m","m","h","m")
length(temperature)
humid=c("h","h","h","h","n","n","n","h","n","n","n","h","n","h")
length(humid)
windy=c("f","t","f","f","f","t","t","f","f","f","t","t","f","t")
length(windy)
play=c("n","n","y","y","y","n","y","n","y","y","y","y","y","n")
```

```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
```

Data	
airct	List of 9
airq	116 obs. of 6 variables
dectree	List of 1
df	14 obs. of 5 variables
dtree_fit	Large train (24 elements, 1.5 Mb)
iris	150 obs. of 5 variables
iris_classifier	List of 4

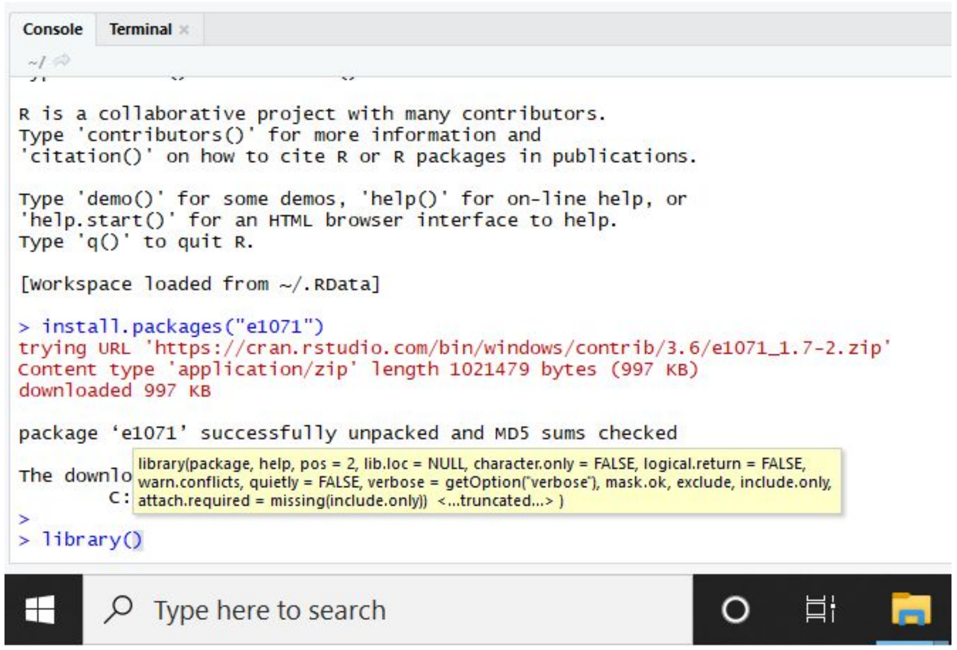
Console Pane/Script Pane/History Pane/Environment Pane/Plots Pane
Run section of script/Source tab to run entire script

Just Start

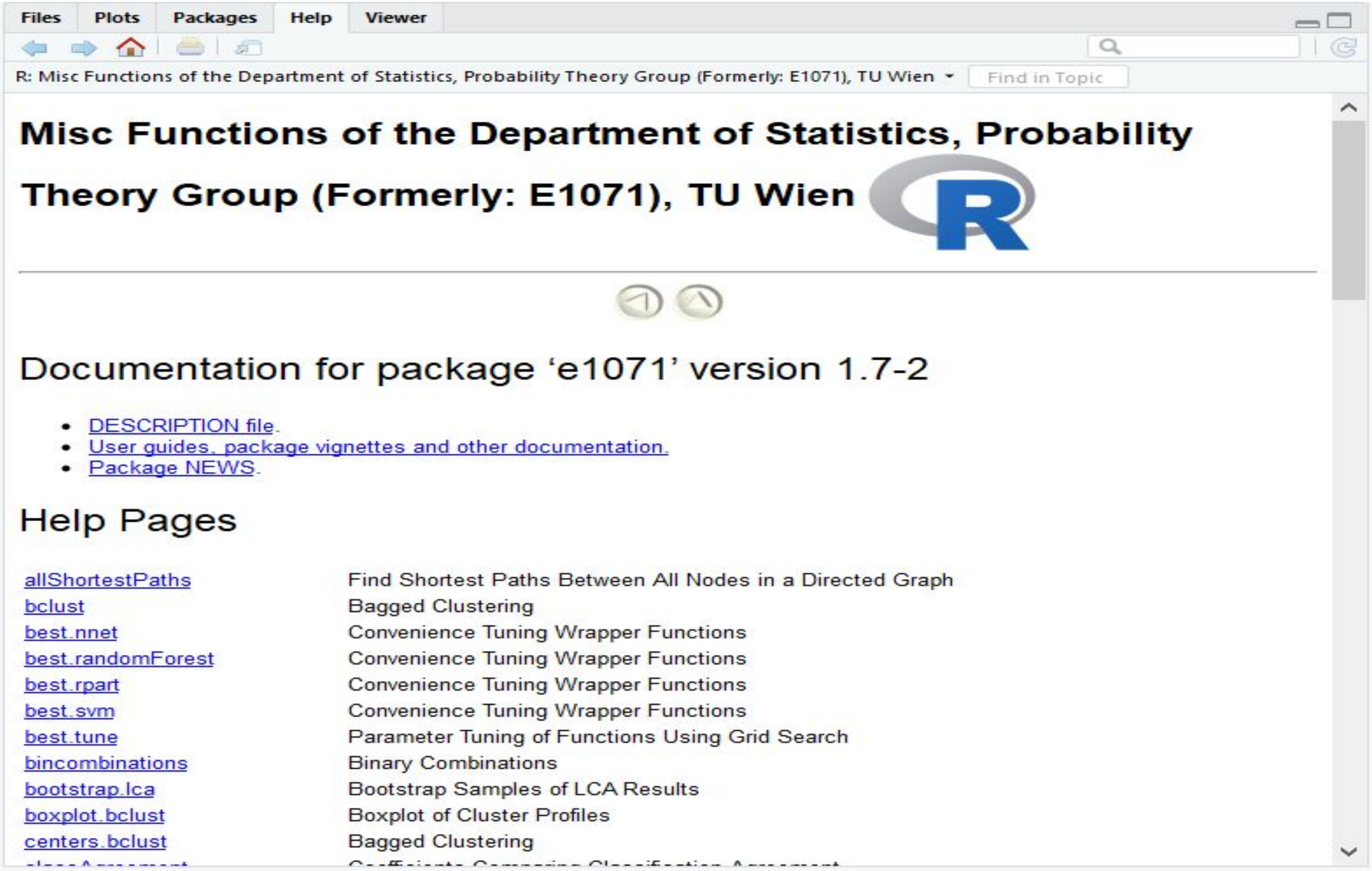
- To load a list of installed packages:
- **> library()**
- 2. Setting the default CRAN (Comprehensive R Archive Network) mirror:
- **> chooseCRANmirror()**
- R will return a list of CRAN mirrors, and then ask the user to either type a mirror ID to select it, or enter zero to exit:
- Install a package from CRAN; take package “e1071” as an example:
- **> install.packages("e1071")**
- **e1071 -**Misc Functions of the Department of Statistics, Probability Theory

Load library

- Update a package from CRAN; take package e1071 as an example:
- **> update.packages("e1071")**
- Load the package the package:
- **> library(e1071)**
- To view the documentation of the package,
- **> help(package="e1071")**



```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> install.packages("e1071")
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.6/e1071_1.7-2.zip'
Content type 'application/zip' length 1021479 bytes (997 KB)
downloaded 997 KB

package ‘e1071’ successfully unpacked and MD5 sums checked

The downlo library(package, help, pos = 2, lib.loc = NULL, character.only = FALSE, logical.return = FALSE,
        C: warn.conflicts, quietly = FALSE, verbose = getOption("verbose"), mask.ok, exclude, include.only,
           attach.required = missing(include.only))  <...truncated...> )
>
> library()
```


Files
Plots
Packages
Help
Viewer
R: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien
Find in Topic

Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien

Documentation for package ‘e1071’ version 1.7-2

- DESCRIPTION file.
- User guides, package vignettes and other documentation.
- Package NEWS.

Help Pages

allShortestPaths	Find Shortest Paths Between All Nodes in a Directed Graph
bclust	Bagged Clustering
best.nnet	Convenience Tuning Wrapper Functions
best.randomForest	Convenience Tuning Wrapper Functions
best.rpart	Convenience Tuning Wrapper Functions
best.svm	Convenience Tuning Wrapper Functions
best.tune	Parameter Tuning of Functions Using Grid Search
bincombinations	Binary Combinations
bootstrap.lca	Bootstrap Samples of LCA Results
boxplot.bclust	Boxplot of Cluster Profiles
centers.bclust	Bagged Clustering

Help

- To view the documentation of the function, you can use the help function:
- **> help(svm, e1071)**
- **> ?e1071::svm**
- **> ??svm (in some versions)**
- to know the argument taken for the lm function:
- **> args(lm)**
- To view an example or demo.
- **> example(lm)**
- **> demo(graphics)**
- To view all the available demos
- library(help = "graphics")
- **> demo()**

Set directory path

- Create your own directory say “PML_Lab_Assignments”
- type getwd() in the R session to obtain the current working directory location.
- change the current working directory,
- setwd("<path>"), where <path> can be replaced as your desired path, to specify the working directory.
- getwd()
- To run script file: source('R_basics.R')

DATA TYPES IN R

- The *logical* data type is a simple binary variable that may have only two values: TRUE or FALSE. also commonly referred to as *flags*. E.g. Married Variable.
- The *numeric* data type stores decimal numbers, while the *integer* data type stores integers.
- If you create a variable containing a number without specifying a data type, R will store it as numeric by default. However, R can usually automatically convert between the numeric and integer data types as needed.
- R also calls the numeric data type double, which is short for a double-precision floating-point number. The terms numeric and *double* are interchangeable.
- The *character* data type is used to store text strings of up to 65,535 characters each.

DATA TYPES IN R

- The *factor* data type is used to store categorical values. Each possible value of a factor is known as a *level*. For example indian states
- The *ordered factor* data type is a special case of the factor data type where the order of the levels is significant. For example, risk ratings of Low, Medium, and High
- There is also a special number Inf which represents infinity. This allows us to represent entities like 1/0. This way, Inf can be used in ordinary calculations; e.g. 1/Inf is 0.
- The value NaN represents an undefined value ("not a number"); e.g. 0/0; NaN can also be thought of as a missing value.

Try some commands

```
• > x <- 1
• > print(x)
• [1] 1
• > x
• [1] 1
• > class(x)
• [1] "numeric“
• > x='hello'
• > class(x)
• [1] "character“
• > x=FALSE
• > class(x)
• [1] "logical“
```

Commands

- > x=integer(3)
- > class(x)
- [1] "integer”
- > print(x)
- [1] 0 0 0
- x <- 10:30
- Print(x)
- rm(list = ls()) //clears all workspace
- rm(objectname)

Add/Sub/Mul/Div

- 2+3
- 2*3
- 2**3
- 10/3
- 7^3
- 10 %% 2
- 10 %% 3
- 10 %/% 3
- 10 %/% 2
- floor(10/3)
- ceiling(10/3)
- Sqrt(4)
- Sqrt(-1)
- Exp(1)
- log2(8)
- Pi

Vector

- The most basic type of R object is a vector. Empty vectors can be created with the vector() function.
- **A vector can only contain objects of the same class**.
- The c() function can be used to create vectors of objects by concatenating things together.
- > x <- c(0.5, 0.6) *## numeric*
- > x <- c(**TRUE**, **FALSE**) *## logical*
- > x <- c(**T**, **F**) *## logical*
- > x <- c("a", "b", "c") *## character*
- x <- vector("numeric", length = 10)
- > x
- [1] 0 0 0 0 0 0 0 0 0 0
- If you explicitly want an integer, you need to specify the L suffix. So entering 1 in R gives you a numeric object; entering 1L explicitly gives you an integer object.
- seq(from = 2, by = -0.1, length.out = 4) //sequence of numbers
- Seq(1,10,0.5)

List

- A list is represented as a vector but can contain objects of different classes.
- > y <- c(1.7, "a") *## character*
- > y <- c(**TRUE**, 2) *## numeric*
- > y <- c("a", **TRUE**) *## character*
- cars <-list(name =c("Honda","BMW","Ferrari"),color =c("Black","Blue","Red"), cc =c(2000,3400,4000))
- cars$name
- # character indicates a comment. Anything to the right of the # (including the # itself) is ignored.
- This is the only comment character in R. Unlike some other languages, R does not support multi-line comments or comment blocks

Indexing

- x=seq(from = 2, by = -0.1, length.out = 4)
- x[1]
- x[2:3]
- x[c(2,1)]
- y=x>=1.9
- y
- x[y]
- x=c(x,19) #append
- Vectors are basic objects in R and they can be subsetted using the [operator.
- x <- c("a", "b", "c", "c", "d", "a")
- The sequence does not have to be in order; you can specify any arbitrary integer vector.
- > x[c(1, 3, 4)]
- [1] "a" "c" "c“

Typecast

- Objects can be explicitly coerced from one class to another using the as.* functions, if available.
- > x <- 0:6
- > class(x)
- [1] "integer"
- > y=as.numeric(x)
- [1] 0 1 2 3 4 5 6
- is.numeric(x)
- class(y)
- > as.logical(x)
- [1] **FALSE TRUE TRUE TRUE TRUE TRUE TRUE**
- > as.character(x)
- [1] "0" "1" "2" "3" "4" "5" "6"

Matrix

- m <- matrix(nrow = 2, ncol = 3)
- dim(m)
- [1] 2 3
- attributes(m)
- $dim
- [1] 2 3
- m <- matrix(1:6, nrow = 2, ncol = 3)
- m
- [,1] [,2] [,3]
- [1,] 1 3 5
- [2,] 2 4 6

- > m <- 1:10
- > m
- [1] 1 2 3 4 5 6 7 8 9 10
- > dim(m) <- c(2, 5)
- > m
- [,1] [,2] [,3] [,4] [,5]
- [1,] 1 3 5 7 9
- [2,] 2 4 6 8 10
- C(m) #convert to 1D vector

Matrix

- Matrix to vector: as.vector(m)
- m <- matrix(1:6, nrow = 2, ncol = 3)
- n <- matrix(10:15, nrow = 2, ncol = 3)
- M+n
- M-n
- M*n (dot product)
- t(n)
- m %*% t(n)
- crossprod(m,n) #same as t(m)%*%n
- rowMeans(m)
- rowSums(m)
- colSums(m)
- colMeans(m)

- n1=c(1,2,3)
- n2=c(10,20,30)
- n1*n2
- n1 %*% n2 (matrix multiplication)/ crossprod(n1, n2)
- n1%o%n2 (outer product)

$$A \cdot B = A^T B = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \end{bmatrix}.$$

The outer product is $A \otimes B = AB^T$, where

$$AB^T = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_1 b_2 & \cdots & a_1 b_n \\ a_2 b_1 & a_2 b_2 & \cdots & a_2 b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_n b_1 & a_n b_2 & \cdots & a_n b_n \end{bmatrix}.$$

Cbind/rbind

- x = 1:3
- y = 10:12
- cbind(x, y)
- x y
  - [1,] 1 10
  - [2,] 2 11
  - [3,] 3 12
- rbind(x, y)
  - [,1] [,2] [,3]
  - x 1 2 3
  - y 10 11 12

Factors

- Factors are used to represent categorical data.
- Having a variable that has values “Male” and “Female” is better than a variable that has values 1 and 2.
- Factor objects can be created with the factor() function.
- x = factor(c("yes", "yes", "no", "yes", "no"))
- x
  - [1] yes yes no yes no
  - Levels: no yes
- table(x)
  - no yes
  - 2   3
- *## See the underlying representation of factor*
- unclass(x)
  - [1] 2 2 1 2 1

NA and NaN

- Missing values are denoted by NA (Not Available, declared in advance) or NaN (Not a Number i.e. 0/0).
- is.na() is used to test objects if they are NA
- is.nan() is used to test for NaN
- NA values have a class also, so there are integer NA, character NA, etc.
- A NaN value is also NA but the converse is not true
- *## Create a vector with NAs in it*
- x = c(1, 2, **NA**, 10, 3)
- complete.cases(x)
- y=x[complete.cases(x)]
- is.na(x)
- x = c(1, 2, **NaN**, **NA**, 4)
- is.na(x)
  - [1] **FALSE FALSE TRUE TRUE FALSE**
- is.nan(x)
  - [1] **FALSE FALSE TRUE FALSE FALSE**

Dataframes

- used to store tabular data in R.
- Unlike matrices, data frames can store different classes of objects in each column. Matrices must have every element be the same class (e.g. all integers or all numeric).
- In addition to column names, indicating the names of the variables or predictors, data frames have a special attribute called row.names which indicate information about each row of the data frame.
- Data frames are usually created by reading in a dataset using the read.table() or read.csv().
- x = data.frame(foo = 1:4, bar = c(T, T, F, F))
- nrow(x)
- ncol(x)
- class(x$foo)
- names <- c('Mike', 'Renee', 'Richard', 'Matthew', 'Christopher')
- scores <- c(85, 92, 95, 97, 96)
- x <- data.frame(names, scores)

Reading data

- There are a few principal functions reading data into R.
- read.table, read.csv, for reading tabular data
- x = read.table("test1.csv",header=T)
- is.data.frame(x)
- readLines, for reading lines of a text file source, for reading in R code files (inverse of dump)
- dget, for reading in R code files (inverse of dput)
- load, for reading in saved workspaces
- unserialize, for reading single R objects in binary form
- y=read.csv('E:/NTROcourse_2019/R_ppts/test1.csv')
- # https://github.com/defcom17/NSL_KDD
- #nsl kdd dataset
- X=read.csv('Small Training Set.csv')
- X=read.csv(‘test2.csv')
- head(X)

Writing data

- write.table, for writing tabular data to text files (i.e. CSV) or connections
- writeLines, for writing character data line-by-line to a file or connection
- dump, for dumping a textual representation of multiple R objects
- dput, for outputting a textual representation of an R object
- save, for saving an arbitrary number of R objects in binary format (possibly compressed) to a file.
- serialize, for converting an R object into a binary format for outputting to a connection (or file).
- save(m,n,file="ab.Rdata")
- Ls()
- Rm(x,y)
- #cntr-L and rm(list=ls())
- load(“ab.RData")
- write.table(iris,file="test2.csv")
- m=read.table("test2.csv")
- Head(m)

Examples

- initial <- read.table(“test2.csv")
- classes <- sapply(initial, class)
- Classes
- when using R with larger datasets, it’s also useful to know a few things about your system.
- How much memory is available on your system?
- What other applications are in use? Can you close any of them?
- Are there other users logged into the same system?
- What operating system are you using? Some operating systems can limit the amount of memory a single process can access.

Examples

- x <- "foo"
- y <- data.frame(a = 1L, b = "a")
- We can dump() R objects to a file by passing a character vector of their names.
- dump(c("x", "y"), file = "data.R")
- rm(x, y)
- The inverse of dump() is source().
- source("data.R")
- str(y)
  - 'data.frame': 1 obs. of 2 variables:
  - $ a: int 1
  - $ b: Factor w/ 1 level "a": 1
- x
  - [1] "foo"

Examples

- key functions for converting R objects into a binary format are save(), save.image(), and
- serialize(). Individual R objects can be saved to a file using the save() function.
- a <- data.frame(x = rnorm(100), y = runif(100))
- b <- c(3, 4.4, 1 / 3)
- *## Save 'a' and 'b' to a file*
- save(a, b, file = "mydata.rda")
- *## Load 'a' and 'b' into your workspace*
- load("mydata.rda")
- If you have a lot of objects that you want to save to a file, you can save all objects in your workspace using the save.image() function.
- save.image(file = "mydata.RData")
- *## load all objects in this file*
- load("mydata.RData")

Reading special data

- con <- gzfile("words.gz")
- x <- readLines(con, 10)
- x
  - [1] "1080" "10-point" "10th" "11-point" "12-point" "16-point"
  - [7] "18-point" "1st" "2" "20-point"
- For more structured text data like CSV files or tab-delimited files, there are other functions like read.csv() or read.table(). > *## Open a URL connection for reading*
- con <- url("http://www.vit.ac.in", "r")
- *## Read the web page*
- x <- readLines(con)
- *## Print out the first few lines*
- head(x)
  - [1] "<!DOCTYPE html>"
  - [2] "<html lang=\"en\">"
  - [3] ""
  - [4] "<head>"
  - [5] "<meta charset=\"utf-8\" />"
  - [6] "<title>Johns Hopkins Bloomberg School of Public Health</title>"

Example

- x <- matrix(1:6, 2, 3)
- x
  - [,1] [,2] [,3]
  - [1,] 1 3 5
  - [2,] 2 4 6
- We can access the $(1, 2)$ or the $(2, 1)$ element of this matrix using the appropriate indices.
- x[1, 2]
  - [1] 3
- x[2, 1]
  - [1] 2
- Indices can also be missing. This behavior is used to access entire rows or columns of a matrix.
- x[1,] *## Extract the first row*
  - [1] 1 3 5
- x[, 2] *## Extract the second column*
  - [1] 3 4

Access items

- > x <- list(foo = 1:4, bar = 0.6)
- > x
- $foo
- [1] 1 2 3 4
- $bar
- [1] 0.6
- The [[operator can be used to extract *single* elements from a list. Here we extract the first element of the list.
- > x[[1]]
- [1] 1 2 3 4
- The [[operator can also use named indices so that you don’t have to remember the exact ordering of every element of the list. You can also use the $ operator to extract elements by name.
- > x[["bar"]]
- [1] 0.6
- > x$bar
- [1] 0.6

removing missing values

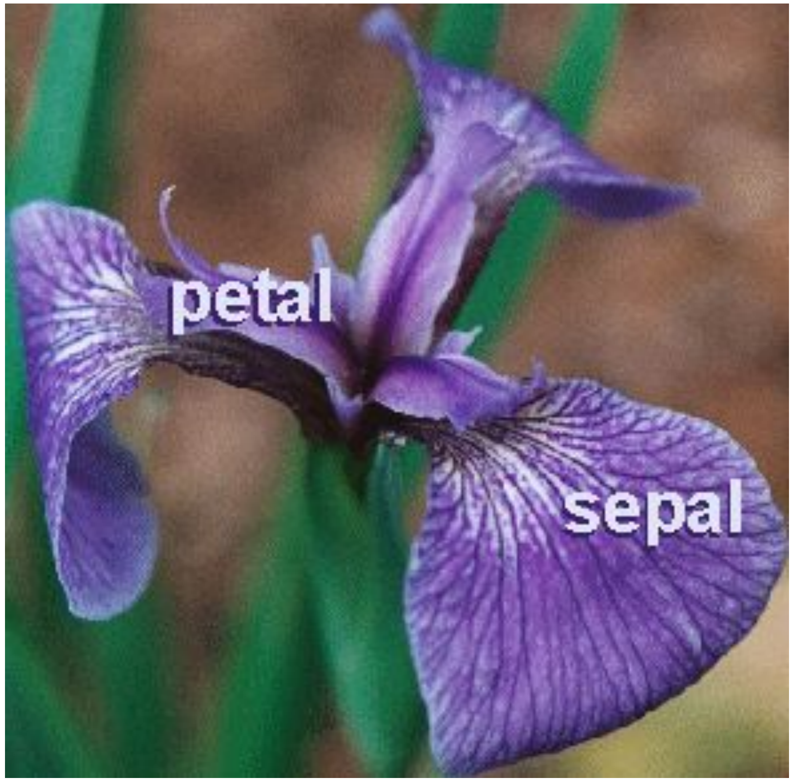
- > x <- c(1, 2, **NA**, 4, **NA**, 5)
- > bad <- is.na(x)
- > print(bad)
- [1] **FALSE FALSE TRUE FALSE TRUE FALSE**
- > x[!bad]
- [1] 1 2 4 5
- > x <- c(1, 2, **NA**, 4, **NA**, 5)
- > y <- c("a", "b", **NA**, "d", **NA**, "f")
- > good <- complete.cases(x, y)
- > good
- [1] **TRUE TRUE FALSE TRUE FALSE TRUE**
- > x[good]
- [1] 1 2 4 5
- > y[good]
- [1] "a" "b" "d" "f"
- You can use complete.cases on data frames too.

Accessing inbuilt datasets

- > head(airquality)
  - Ozone Solar.R Wind Temp Month Day
  - 1 41 190 7.4 67 5 1
  - 2 36 118 8.0 72 5 2
  - 3 12 149 12.6 74 5 3
  - 4 18 313 11.5 62 5 4
  - 5 **NA NA** 14.3 56 5 5
  - 6 28 **NA** 14.9 66 5 6
- > good <- complete.cases(airquality)
- > head(airquality[good,])
  - Ozone Solar.R Wind Temp Month Day
  - 1 41 190 7.4 67 5 1
  - 2 36 118 8.0 72 5 2
  - 3 12 149 12.6 74 5 3
  - 4 18 313 11.5 62 5 4
  - 7 23 299 8.6 65 5 7
  - 8 19 99 13.8 59 5 8

Iris Dataset

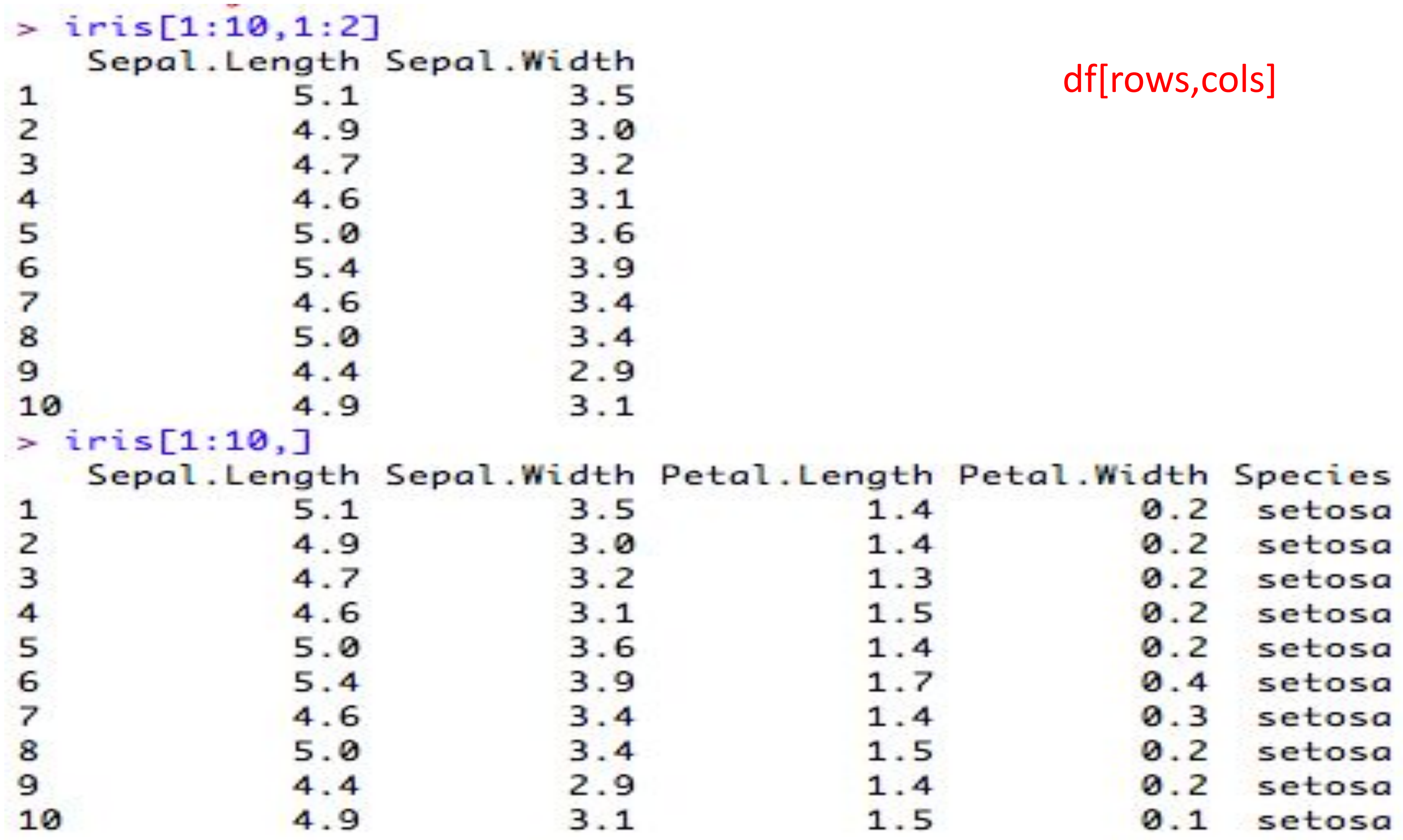
virginica



setosa

`data()` yields many built-in data files.

Iris data



```
> iris[1:10,1:2]
   Sepal.Length Sepal.Width
1           5.1         3.5
2           4.9         3.0
3           4.7         3.2
4           4.6         3.1
5           5.0         3.6
6           5.4         3.9
7           4.6         3.4
8           5.0         3.4
9           4.4         2.9
10          4.9         3.1
> iris[1:10,]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1         3.5          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
3           4.7         3.2          1.3         0.2  setosa
4           4.6         3.1          1.5         0.2  setosa
5           5.0         3.6          1.4         0.2  setosa
6           5.4         3.9          1.7         0.4  setosa
7           4.6         3.4          1.4         0.3  setosa
8           5.0         3.4          1.5         0.2  setosa
9           4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa
```

df[rows,cols]

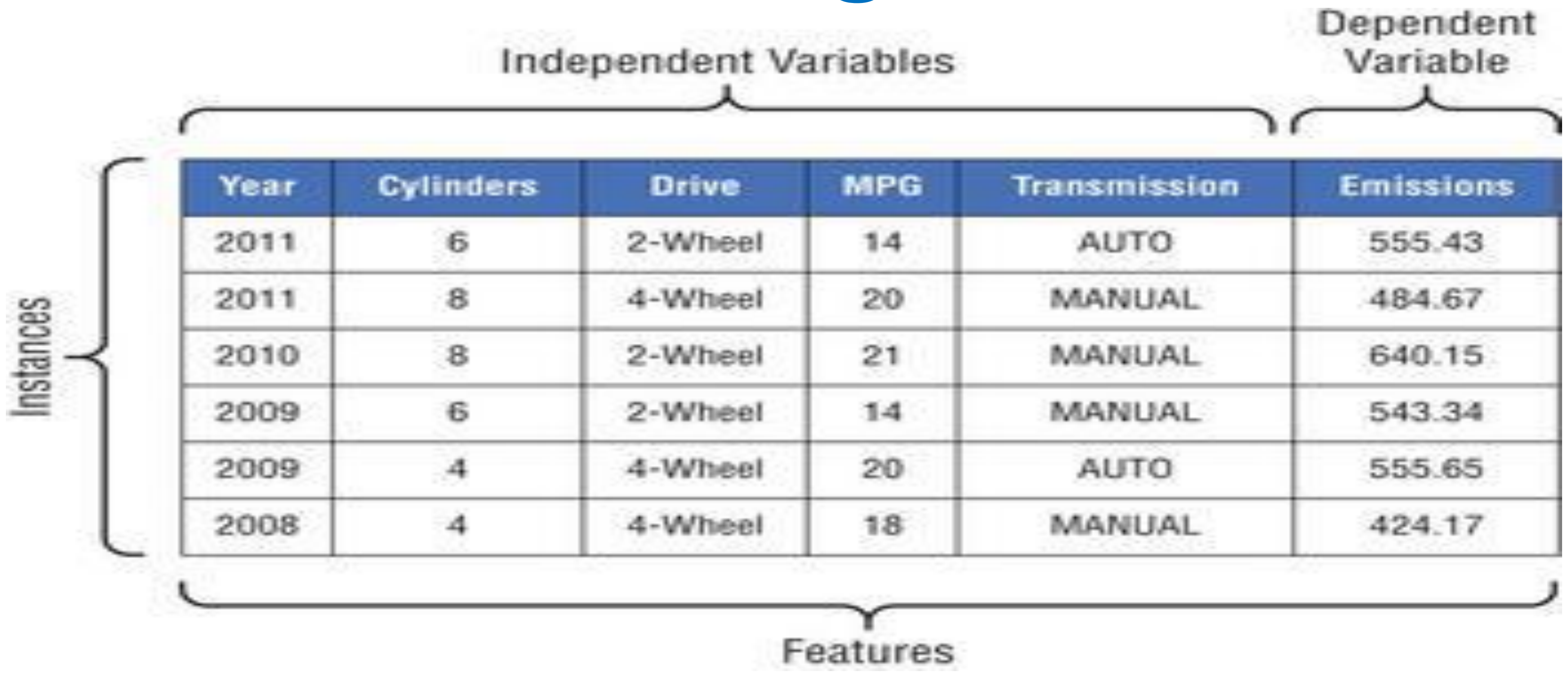
As with vectors, you can "subset" data frames.

If a feature represents a characteristic measured in numbers, it is unsurprisingly called **numeric**. Alternatively, if a feature is an attribute that consists of a set of categories, the feature is called **categorical** or **nominal**.

A special case of categorical variables is called **ordinal**, which designates a nominal variable with categories falling in an ordered list.

Ordinal variables include such as small, medium, and large

Describing the Data


Independent Variables
Dependent Variable
Instances
Features

Year	Cylinders	Drive	MPG	Transmission	Emissions
2011	6	2-Wheel	14	AUTO	555.43
2011	8	4-Wheel	20	MANUAL	484.67
2010	8	2-Wheel	21	MANUAL	640.15
2009	6	2-Wheel	14	MANUAL	543.34
2009	4	4-Wheel	20	AUTO	555.65
2008	4	4-Wheel	18	MANUAL	424.17

- **An instance is a row of data. It is described by a set of attributes or features. A dataset consists of several instances/records/examples/observations.**
- **A feature is a column of data. It is the property or characteristic of an instance. Each instance consists of several features. sometimes refer to features as columns or variables and can be described as either a discrete feature or a continuous feature.**
- **The *dimensionality* of a dataset represents the number of features in the dataset.**

Descriptive/summary statistics for data exploration/understanding

- frequency of a feature value tells us how often the value occurs, and the mode of the feature tells us which value occurs the most for that feature.
- Frequency and mode are typically used to describe categorical data.
- For continuous data, measures such as mean and median are often used to describe the properties of the data.

Tidyverse- collection of packages

- install.packages("tidyverse").
- library(tidyverse)
- includes the packages that you're likely to use in everyday data analyses
- *dplyr* is a package in the tidyverse that is used for data exploration and manipulation
- ggplot2 package for declaratively creating graphics, based on The Grammar of Graphics
- tidyr provides a set of functions that help you get to tidy data. Tidy data is data with a consistent form: in brief, every variable goes in a column, and every column is a variable
- readr provides a fast and friendly way to read rectangular data (like csv)
- purrr - set of tools for working with functions and vectors. allows you to replace many for loops with code that is easier to write and more expressive.
- Other packages are tibble , stringr, forcats

Example – ggplot2

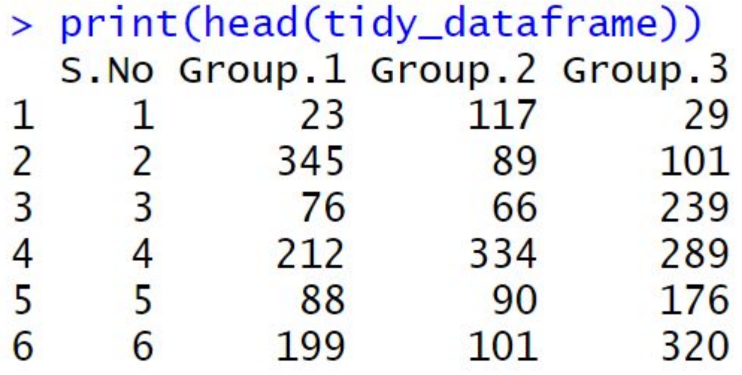
- # load the library
- library("ggplot2")
- # create the dataframe with letters and numbers
- gfg =data.frame(x=c('Amit', ‘Aditi', ‘ganesh', 'Dina', ‘rina', 'Farina'),y=c(80, 80, 100, 75, 70, 50))
- # display the bar
- ggplot(gfg, aes(x, y, fill=x)) + geom_bar(stat="identity")

Examples – dpylr for data manipulation

- group_by() /mutate() /select() /filter() /summarise() /arrange() function
- library(dplyr)
- head(starwars)
- Pipes are written as %>%. They are provided by the *magrittr* package, which is loaded as part of the tidyverse.
- X=starwars %>% select(height)
- table(X)
- starwars %>% select(name:mass) %>% table() %>% prop.table()
- print(starwars %>% filter(species == "Droid"))
- Tibble-> like dataframe but shows only 1st 10 rows/all columns that fit on screen. Easy to work with large data.

Examples – tidyr for data manipulation

- library(tidyr)
- n = 10
- # creating a data frame
- tidy_dataframe = data.frame(S.No = c(1:n),Group.1 = c(23, 345, 76, 212, 88,199, 72, 35, 90, 265),Group.2 = c(117, 89, 66, 334, 90,101, 178, 233, 45, 200), Group.3 = c(29, 101, 239, 289, 176,320, 89, 109, 199, 56))
- # print the elements of the data frame
- print(head(tidy_dataframe))



```
> print(head(tidy_dataframe))
  S.No Group.1 Group.2 Group.3
1    1      23     117      29
2    2     345      89     101
3    3      76      66     239
4    4     212     334     289
5    5      88      90     176
6    6     199     101     320
```

Examples – stringr - data cleaning and data preparation tasks

- # R program for finding length of string
- # Importing package
- library(stringr)
- # Calculating length of string
- str_length('deva shree ganesha')

If/for/while

- x <- runif(1, 0, 10)
- #random number using uniform distribution
- **if**(x > 3) {
- y <- 10
- } **else** {
- y <- 0
- }
- **for**(i **in** 1:10) {
- print(i)
- }
- x <- matrix(1:6, 2, 3)
- **for**(i **in** seq_len(nrow(x))) {
- **for**(j **in** seq_len(ncol(x))) {
- print(x[i, j])
- }
- }

Explore:

While loop

Repeat loop

Next

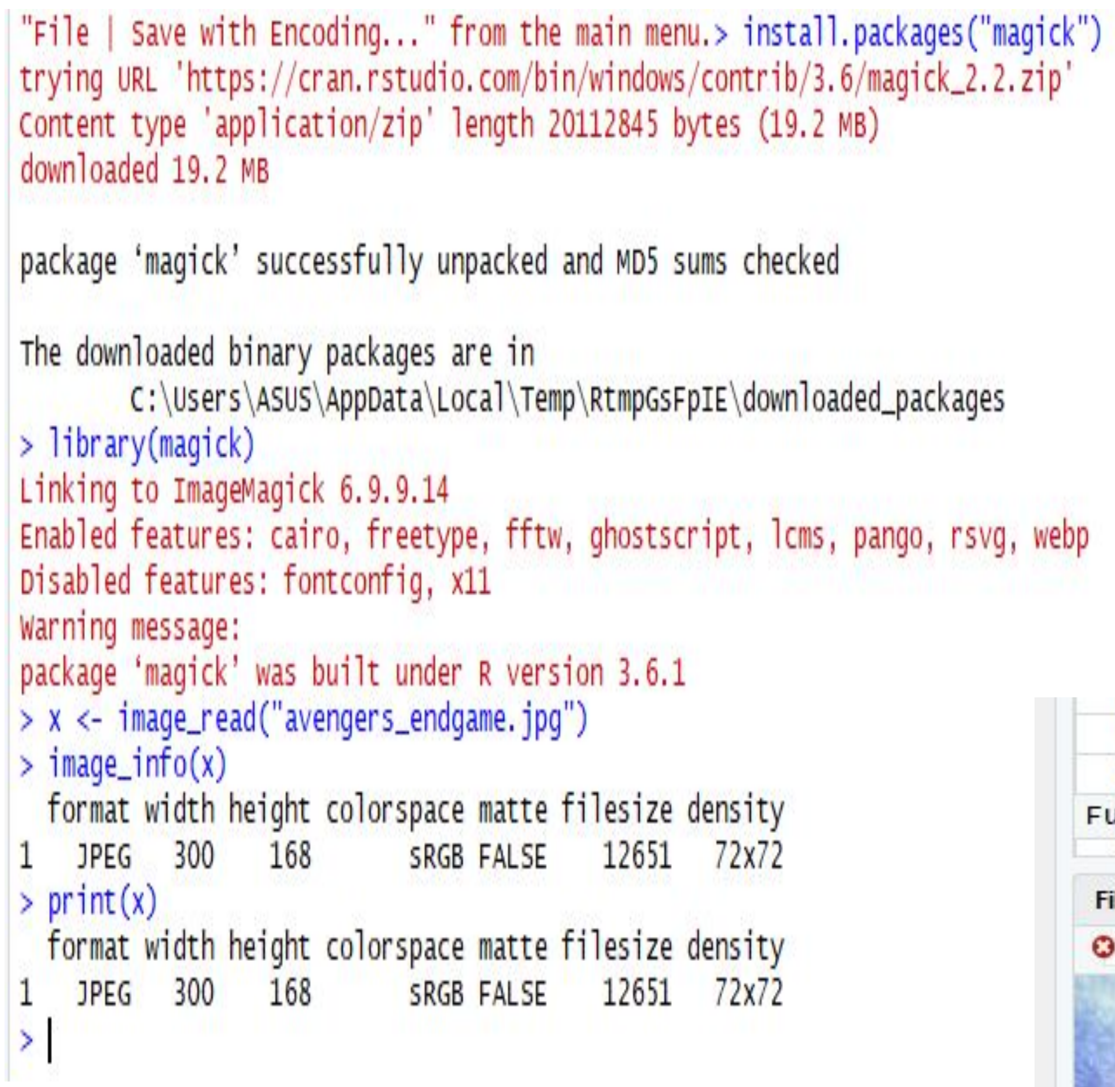
break

Functions

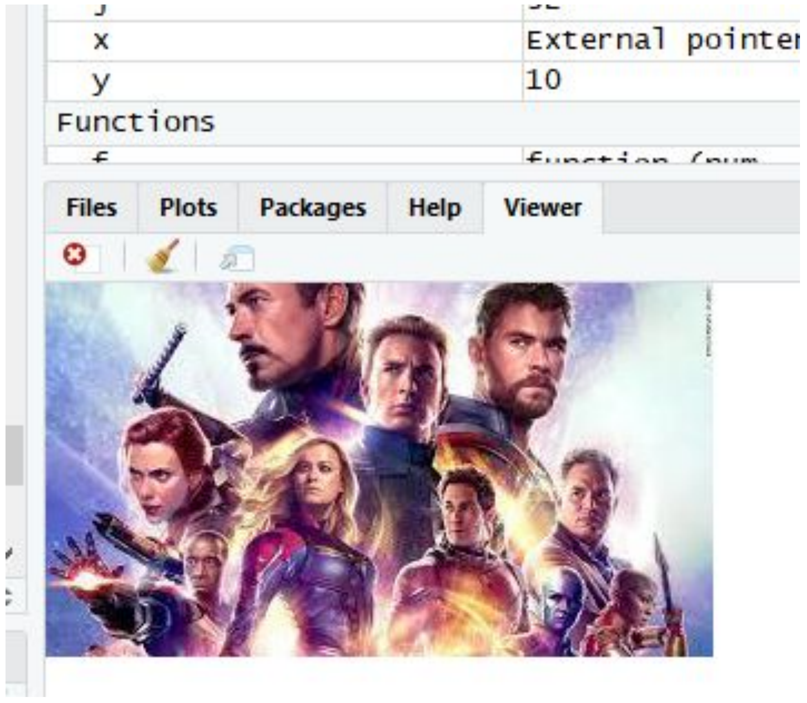
- f <- function(num=1) {
- hello <- "Hello, world!\n"
- for (i in seq_len(num)) {
- print(hello)}
- Return(num+1)
- }
- f(2)
- f(4)
- L=f(5)
- //use list to return multiple values
- x=c(10,9,3,5)
- scaleval=function(x){y=(x-min(x))/(max(x)-min(x))+return(y)}
- scaleval(x)
- x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
- # compute the list mean for each list element
- lapply(x, mean)

Image processing

- install.packages("magick")
- library(magick)
- x <- image_read("test1.jpg")
- image_info(x)
- Print(x)
- y=image_charcoal(x)
- Print(y)
- image_write(y, path = ‘test2.png', format = ‘png')



```
"File | Save with Encoding..." from the main menu.> install.packages("magick")
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.6/magick_2.2.zip'
Content type 'application/zip' length 20112845 bytes (19.2 MB)
downloaded 19.2 MB

package ‘magick’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\ASUS\AppData\Local\Temp\RtmpGsFpIE\downloaded_packages
> library(magick)
Linking to ImageMagick 6.9.9.14
Enabled features: cairo, freetype, fftw, ghostscript, lcms, pango, rsvg, webp
Disabled features: fontconfig, x11
Warning message:
package ‘magick’ was built under R version 3.6.1
> x <- image_read("avengers_endgame.jpg")
> image_info(x)
  format width height colorspace matte filesize density
1   JPEG   300    168       sRGB FALSE    12651   72x72
> print(x)
  format width height colorspace matte filesize density
1   JPEG   300    168       sRGB FALSE    12651   72x72
> |
```



```
x          External pointer
y          10
Functions
```

Thank you

- Submit Assignments.

References

- https://courses.washington.edu/css490/2012.Winter/lecture_slides/02_math_essentials.pdf
- Christopher Bishop: ”Pattern Recognition and Machine Learning” , 2006
- Kevin Murphy: ”Machine Learning: a Probabilistic Perspective”
- David Mackay: ”Information Theory, Inference, and Learning Algorithms”
- Ethem Alpaydin: ”Introduction to Machine Learning” , 2nd edition, 2010.
- R. Duda, P. Hart & D. Stork, ***Pattern Classification*** (2nd ed.), Wiley T. Mitchell, ***Machine Learning,*** McGraw-Hill