FOUNDATIONS OF DATA SCIENCE

MCA III SEMESTER

BY

K.ISSACK BABU MCA

ASSISTANT PROFESSOR

FOUNDATIONS OF DATA SCIENCE (ELECTIVE-III)

UNIT I

INTRODUCTION TO DATA SCIENCE: Data science process – roles, stages in data science project, setting expectations, loading data into R – working with data from files, working with relational databases. Exploring data – Using summary statistics to spot problems, spotting problems using graphics and visualization. Managing data – cleaning and sampling formodelling and validation.

UNIT II

MODELING METHODS: Choosing and evaluating models – mapping problems to machine learning tasks, evaluating models, validating models – cluster analysis – Kmeans algorithm, Naïve Bayes, Memorization Methods – KDD and KDD Cup 2009, building single variable models, building models using multi variable, Linear and logistic regression, unsupervised methods – cluster analysis, association rules.

UNIT III

INTRODUCTION TO R Language: Reading and getting data into R, viewing named objects, Types of Data items, the structure of data items, examining data structure, workingwith history commands, saving your work in R.

PROBABILITY DISTRIBUTIONS in R - Binomial, Poisson, Normal distributions. Manipulating objects - data distribution.

UNIT IV

DELIVERING RESULTS: Documentation and deployment–producing effective presentations–Introduction to graphical analysis – plot()function – displaying multivariate data– matrix plots– multiple plots in one window - exporting graph – using graphics parameters in R Language.

UNIT I

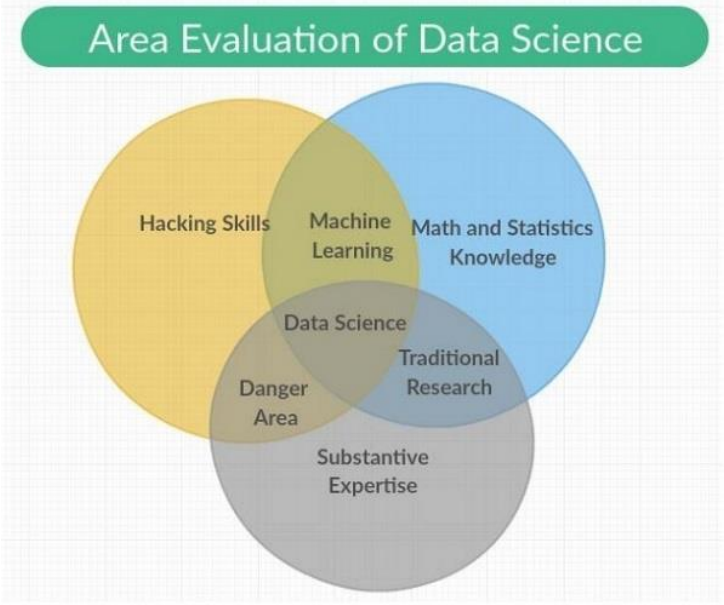
INTRODUCTION TO DATA SCIENCE: Data science process – roles, stages in data science project, setting expectations, loading data into R – working with data from files, working with relational databases. Exploring data – Using summary statistics to spot problems, spotting problems using graphics and visualization. Managing data – cleaning and sampling for modelling and validation.

INTRODUCTION TO DATA SCIENCE

Data science process – roles:

Data science process:-

"Data science is the blend of data interface, algorithm development and technology in order to solve analytical complex problems".



Data science is an interdisciplinary field encompassing scientific methods, processes and systems with categories included in it as Machine learning, math and statistics knowledge with traditional research. It also includes a combination of hacking skills with substantive expertise. Data science draws principles from mathematics, statistics, information science, and computer science, data mining and predictive analysis.

The different roles that form part of the data science team are mentioned below

Customers:- Customers are the people who use the product. Their interest determines the success of project and their feedback is very valuable in data science.

Business Development:- This team of data science signs in early customers, either firsthand or through creation of landing pages and promotions. Business development team delivers the value of product.

Product Managers:- Product managers take in the importance to create best product, which is valuable in market.

Interaction designers:- They focus on design interactions around data models so that users find appropriate value.

Data scientists:- Data scientists explore and transform the data in new ways to create and publish new features. These scientists also combine data from diverse sources to create a new value. They play an important role in creating visualizations with researchers, engineers and web developers.

Researchers:- As the name specifies researchers are involved in research activities. They solve complicated problems, which data scientists cannot do. These problems involve intense focus and time of machine learning and statistics module.

Adapting to Change:- All the team members of data science are required to adapt to new changes and work on the basis of requirements. Several changes should be made for adopting agile methodology with data science, which are mentioned as follows −

- Choosing generalists over specialists.
- Preference of small teams over large teams.
- Using high-level tools and platforms.
- Continuous and iterative sharing of intermediate work.

Roles:-

The role of a data scientist is normally associated with tasks such as predictive modeling, developing segmentation algorithms, recommender systems, A/B testing frameworks and often working with raw unstructured data.

The nature of their work demands a deep understanding of mathematics, applied statistics and programming. There are a few skills common between a

data analyst and a data scientist, for example, the ability to query databases. Both analyze data, but the decision of a data scientist can have a greater impact in an organization.

Here is a set of skills a data scientist normally need to have −

- Programming in a statistical package such as: R, Python, SAS, SPSS, or Julia
- Able to clean, extract, and explore data from different sources
- Research, design, and implementation of statistical models
- Deep statistical, mathematical, and computer science knowledge

In big data analytics, people normally confuse the role of a data scientist with that of a data architect. In reality, the difference is quite simple. A data architect defines the tools and the architecture the data would be stored at, whereas a data scientist uses this architecture. Of course, a data scientist should be able to set up new tools if needed for ad-hoc projects, but the infrastructure definition and design should not be a part of his task.

*****

Stages in data science project:

Life cycle of data science is recursive. After completing the all phases, the data scientist can back to top. The data Science life cycle is like a cross industry process for data mining as data science is an interdisciplinary field of data collection, data analysis, feature engineering, data prediction, data visualization and is involved in both structured and unstructured data.

The phases of Data Science are –

- Business Understanding
- Data Mining
- Data Cleaning
- Exploration
- Feature Engineering
- Prediction Modeling
- Data Visualization

Business Understanding:-

At first, the data scientist identifies the problem, a group of people analyzes the problem and discuss their solutions. They also learn the previous records to

identify whether such problem happened earlier or not. Every decision has to be in favour of the organization.

Data Mining:-

Data mining is the process of identifying what type of data is available to them?, is data sufficient according the requirement?, or is there any need to buy the data from a third party?, if yes, would the data secure or private? This process is time consuming, as in it data gathered from different sources. The main perspective of data mining is to gathering all the needful data.

Data Cleaning:-

The collected data in the data mining process may contain lots of unnecessary data or may be inconsistent way. It may also happen that some pieces of the data are in different sources, the date format may be incomplete. So, the next task of data scientist is to clean all the unwanted data or make data consolidation. This process may be time consuming, as all depends on the quality of gathered data. At last of the process the data scientist has cleaning and manipulated data.

Exploration:-

Data exploration is in actually the starting stage of data analysis. In this process, the data scientist summarizes the data with main characteristics and analyze and explore each data set very carefully. They can use the different graphical representation technique like histogram, scatter plots and so on.

Feature Engineering:-

This process is basically the applied machine learning. In this process, domain knowledge and deep learning of data is required to make the machine learning algorithm to work. This is very difficult and expensive. This process requires brainstorming to improve the features. The features in your data is important for the data prediction.

Prediction Modeling:-

Here, the data scientist predicts the project. There are so many predictive analytics questions in front of the finally built data science project. They are also predicting the future events and actions.

Data Visualization:-

Data visualization is to show the information in the pictorial or graphical configuration. It empowers leaders to see examination displayed outwardly, so they can get a handle on troublesome ideas or recognize new examples. With intelligent perception, you can make the idea a stride facilitate by utilizing innovation to penetrate down into diagrams and charts for more detail, intuitively changing what information you see and how it's prepared.

*****

Setting expectations:

Having the title of Data Scientist can come with a lot of assumptions and expectations. Different companies have their definitions for what it means to be a data scientist there. And each company comes with its expectations and assumptions about what they want you to do for them. With that, I picked out three of the most tiring assumptions and expectations I often face while working or interviewing as a data scientist.

We can do EVERYTHING:-

Often, data scientists are expected to look at the data and make an analysis work for the person requesting it. Some companies will hire and assume that the data scientist can perform multiple roles: data scientist, front-end developer, backend developer, data analyst, data engineer, ML Ops, and more. Unfortunately, this is not the case. A data scientist shouldn't be expected to perform every aspect of the pipeline.

Each of the roles in the data space has a subset of skills that they do and do well. Each one is a cog in the machine that keeps the process moving end to end. Data scientists shouldn't be expected to know and understand every aspect of the pipeline. Instead, we should be working with a team of software developers, engineers, SME's, and more to build a long-lasting product.

One role I held expected me to recreate all datasets in a separate location. While at the same time, we had a data engineering team who housed our data in the cloud in an easy-to-access format. We were not a small team or a startup that needed people to generalize their skills to get things working.

I wrote a post about this a while back because this request bothered me. If we have a data engineering team that will work closely with us to provide the data in

the format we need, why are we recreating all of that? The reason — my manager wanted a self-sufficient team that did not rely on others. This is not fair. Teams need to work effectively together, not battle against unreasonable requests.

You are required to use ML/AI for all projects:-

As much as some don't want to hear it, not every problem requires the most extensive ML/AI algorithm or tool. You can solve some issues with solutions like physics-based analytics, descriptive statistics, or dashboarding.

When you are evaluating your business objective, you need to determine the best solution. When I started as a data scientist, the best advice I was given was to find the most straightforward solution first. Focus on what is simple, and then build your solution out from there. Don't over-complicate a problem to flex your skills. You may be able to solve someone's problem without the need for extensive ML/AI. Make sure you are evaluating the use case first and using ML/AI when appropriate.

Final Thoughts:-

The title of data scientist can come with a lot of assumptions and expectations. Don't let this overwhelm you. Instead, focus on what skills and knowledge are essential to know for your role and objectives.

- Don't focus on mastering everything. Determine where you want to apply your skills in the data space and target that, whether it be data science, data engineering, ML Ops, or something else. Find your area and develop skills there.
- You are not required to use ML/AI for every project. Understand your business objectives, and learn your use cases. ML/AI should be applied where applicable.
- Along with ML/AI, tools do not work right out of the box. You will need to evaluate if a tool is the right one for the job or not. The newest tool or the next trend is not always the right fit for your use case.

*****

Loading data into R – working with data from files:

Loading data into R:

R is a programming language designed for data analysis. Therefore loading data is one of the core features of R.

R contains a set of functions that can be used to load data sets into memory. You can also load data into memory using R Studio - via the menu items and toolbars. In this tutorial I will cover both methods.

Which method of loading data in R you should use depends on what you are doing. If you are just playing around with some data, using the R Studio menu items might be fine. But if you are writing an R program that needs be repeated for many different data sets, it might be better to write the loading of data as R program statements.

R has three different functions which can import data. These are:

- read.table()
- read.csv()
- read.delim()

These functions are very similar to each other, so if you master one of them you will soon master the others. In fact, you can probably just use the read.table() function for all of your data imports. These 3 functions will be covered in the following sections.

read.table()

The R function read.table() function loads data from a file into a tabular data set (table) in memory. A tabular data set consists of rows and columns, just like a spreadsheet. Sometimes rows are also referred to as "records" and columns referred to as "fields" or "properties".

The read.table() function takes three parameters:

- The file name of the file to load
- A flag telling if the file contains a header line
- The separator character used inside the file to separate the values of each row.

The parameters to read.table() are listed between the parentheses, separated with commas. Here is an example of loading a CSV file using read.table() in R:

```
read.table("data.csv", header=T, sep=";")
```

The first parameter is the path to the file to read. In the example above that is the "data.csv" part. This parameter should contain a path to the file to read. In the above example only the file name itself is shown. Then R expects to find the file in the same directory R is running from. If you want to specify the full path to the file, you can do so too. Here is an example of how that looks on Windows:

```
"d:\\data\\projects\\tutorial-projects\\r-programming\\data.csv"
```

Normally, Windows only uses a single backslash (the \ character) between directory names, but in programming languages it is normal to use the \ character as an escape character in strings (text variables). When a programming language sees a \ in a string it will normally look at the next character after the \ to determine what character to insert into the string. To actually insert a \ you will therefore often need two \ (\\) as shown above.

The same file path on a Mac or Linux machine could look like this:

```
"/data/projects/tutorial-projects/r-programming/data.csv"
```

Notice the use of / between directories instead of \, and notice that you only need a single / between the directories, because / is not an escape character.

The second parameter of read.table() is the header=T part. This tells the read.table() function whether the first line in the data file is a header line or not. A value of header=T or header=TRUE means that the first line is a header line. A value of header=F or header=FALSE means that the first line is not a header line.

By "header line" is meant whether the first line contains the column names, or if the first line already contains data. Look at this CSV file:

```
name;id;salary
John Doe;1;99999
Joe Blocks;2;120000
Cindy Loo;3;150000
```

Notice how the first row contains the column names for the data on the following rows.

The third parameter specifies what character inside the data file that is used to separate the different column values on each row. If you look at the CSV file contents above you can see that a semicolon (;) is used as separator. That is why the third parameter to the read.table() function call is sep=";" meaning that the separator character used in the data file is a semicolon.

To execute read.table() you type the commands shown in this section into the console part of R Studio and press the "Enter" key.

read.csv()

The read.csv() function reads a CSV file into the memory. The read.csv() function takes 3 parameters, just like the read.table() function. Here is an example call to the read.csv() function:

```
data = read.csv("D:\\data\\data.csv", header=T, sep=";")
```

This example loads the CSV file located at D:\\data\\data.csv and assign it to the variable named data. The first line is a header line containing the names of the columns in the CSV file. This is specified by the second parameter header=T. The third parameter specifies that the separator character used inside the CSV file is ; (a semicolon).

read.delim()

The read.delim() function reads a CSV file into the memory, just like the read.csv() function. The read.delim() function takes 3 parameters, just like the read.table() function. Here is an example call to the read.delim() function:

```
data = read.delim("D:\\data\\data.csv", header=T, sep=";")
```

This example loads the CSV file located at D:\\data\\data.csv and assign it to the variable named data. The first line is a header line containing the names of the columns in the CSV file. This is specified by the second parameter header=T. The third parameter specifies that the separator character used inside the CSV file is ; (a semicolon).

Working with data from files:

File formats are designed to store specific types of information, such as **CSV, XLSX** etc. The file format also tells the computer how to display or process its content. Common file formats, such as **CSV, XLSX, ZIP, TXT** etc.
If you see your future as a data scientist so you must understand the different types of file format. Because data science is all about the data and it's processing and if you don't understand the file format so may be it's quite complicated for you. Thus, it is mandatory for you to be aware of different file formats.

Different type of file formats:-

CSV: the CSV is stand for Comma-separated values. as-well-as this name CSV file is use comma to separated values. In CSV file each line is a data record and Each record consists of one or more then one data fields, the field is separated by commas.

```
import pandas as pd

df = pd.read_csv("file_path / file_name.csv")

print(df)
```

XLSX: The XLSX file is Microsoft Excel Open XML Format Spreadsheet file. This is used to store any type of data but it's mainly used to store financial data and to create mathematical models etc.

```
import pandas as pd

df = pd.read_excel (r'file_path\\name.xlsx')

print (df)
```

Note:
install xlrd before reading excel file in python for avoid the error. You can install xlrd using following command.

```
pip install xlrd
```

ZIP: ZIP files are used an data containers, they store one or more then one files in the compressed form. it widely used in internet After you downloaded ZIP file, you need to unpack its contents in order to use it.

```
import pandas as pd

df = pd.read_csv(' File_Path \\ File_Name .zip')

print(df)
```

TXT: TXT files are useful for storing information in plain text with no special formatting beyond basic fonts and font styles. It is recognized by any text editing and other software programs.

```
import pandas as pd

df = pd.read_csv('File_Path \\ File_Name .txt')

print(df)
```

JSON: JSON is stand for JavaScript Object Notation. JSON is a standard text-based format for representing structured data based on JavaScript object syntax

```
import pandas as pd

df = pd.read_json('File_path \\ File_Name .json')

print(df)
```

HTML: HTML is stand for stands for Hyper Text Markup Language is use for creating web pages. we can read html table in python pandas using read_html() function.

```
import pandas as pd

df = pd.read_html('File_Path \\File_Name.html')

print(df)
```

PDF: pdf stands for Portable Document Format (PDF) this file format is use when we need to save files that cannot be modified but still need to be easily available.

```
pip install tabula-py
pip install pandas
df = tabula.read_pdf(file_path \\ file_name .pdf)
print(df)
```

Working with relational databases:

In the age of big data, data scientists should leverage relational databases in their workflow. Doing so, analysts can integrate the power of a database engine for munging and calculating and streamlined workflow for data summarization, visualization, modeling, and high end computing, all while rendering a reproducible, efficient process.

However, many data analysts today continue to work with small to large flat files including delimited text (.csv, .tab, .txt) files; Excel (.xls, .xlsx, .xlsb, .xlsm) files; other software binary types (.sas7bdat, .dta, .sav); nested XML/JSON files; and other formats that can easily be changed, moved, deleted, and corrupted to interrupt workflows and version controls set in place. Additionally these formats can maintain redundant, repetitive indicator information for inefficient disk storage use.

As a solution, relational databases provide a sound solution in most workflows as they provide:

- Relational model of primary/foreign keys between related tables to avoid repetitive, redundant information and orphaned records;
- Powerful engine that adheres to query optimization with indexes and execution plans;
- Expressive declarative, universal SQL language for easy set-based operations (SELECT, JOIN, UNION, GROUP BY) compared to counterpart operations in programming languages (i.e., Java, C++, Python, R) or software (SAS, Stata, SPSS, Matlab);
- Ensure reproducibility in data analytics and research process with stable data access and sourcing and constraints for data typology and value mismatches;
- Secure, reliable ACID-based platform with user access controls and backup recoveries in place.

Today, practically all programming languages and software maintain database APIs, including popular tools in data science:.

- **Python** with its PEP 249 specification: generalized (pyodbc, JayDeBeApi) and specific (cx_Oracle, pymssql, pymysql, ibmdb, psycopg2, and sqlite3);
- **R** with its DBI standard including: generalized (odbc, RJDBC) and specific APIs (RPostgreSQL, RMySQL, RSQLite, ROracle, and others);
- **Julia** databases including: general interfaces (ODBC.jl, JDBC.jl, DBI.jl), and specific (MySQL.jl, SQLite.jl, PostgreSQL.jl);
- **Excel** ODBC/OLEDB connections via ADO or DAO modules;
- **SAS** drivers for JDBC and ODBC driver via `libname` and `proc sql` module;
- **Stata** ODBC drivers and DSN connections via `odbc` command;
- **SPSS** ODBC/OLEDB connection via `GET DATA /TYPE` command;
- **Matlab** ODBC/JDBC connection via `database` command.

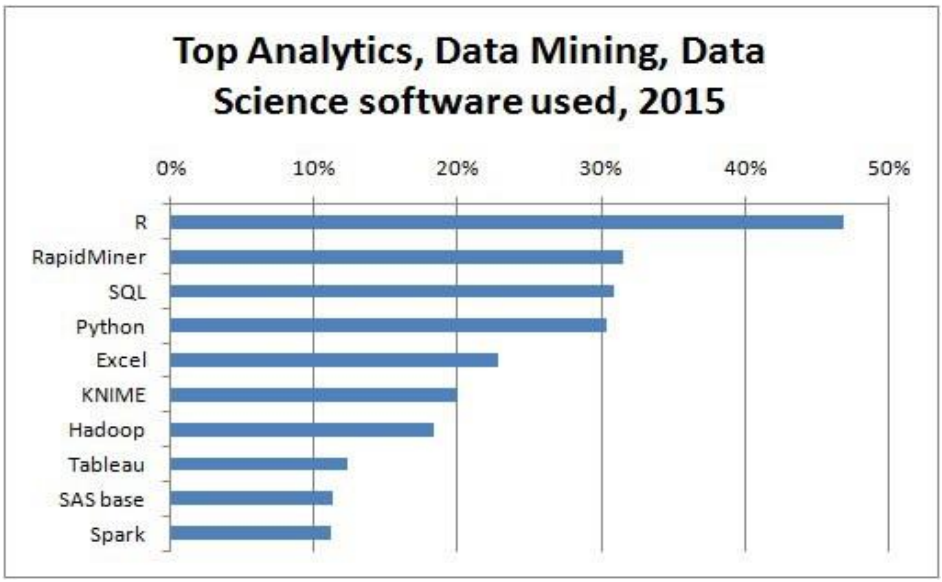
*****

Exploring data – Using summary statistics to spot problems:

Exploratory data analysis is a concept developed by John Tuckey (1977) that consists on a new perspective of statistics. Tuckey's idea was that in traditional statistics, the data was not being explored graphically, is was just being used to test hypotheses. The first attempt to develop a tool was done in Stanford, the project was called prim9. The tool was able to visualize data in nine dimensions, therefore it was able to provide a multivariate perspective of the data.

In recent days, exploratory data analysis is a must and has been included in the big data analytics life cycle. The ability to find insight and be able to communicate it effectively in an organization is fueled with strong EDA capabilities.

Based on Tuckey's ideas, Bell Labs developed the **S programming language** in order to provide an interactive interface for doing statistics. The idea of S was to provide extensive graphical capabilities with an easy-to-use language. In today's world, in the context of Big Data, **R** that is based on the **S** programming language is the most popular software for analytics.



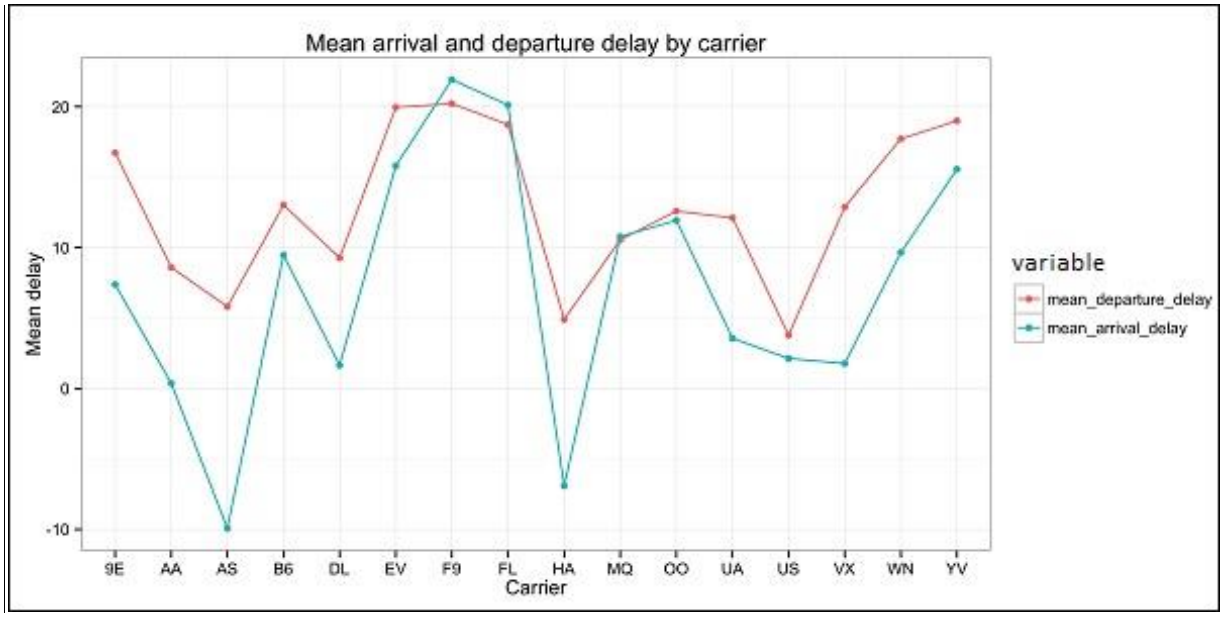
The following program demonstrates the use of exploratory data analysis.

The following is an example of exploratory data analysis. This code is also available in **part1/eda/exploratory_data_analysis.R** file.

```
library(nycflights13)
library(ggplot2)
library(data.table)
library(reshape2)

# Using the code from the previous section
# This computes the mean arrival and departure delays by carrier.
DT <- as.data.table(flights)
mean2 = DT[, list(mean_departure_delay = mean(dep_delay, na.rm = TRUE),
  mean_arrival_delay = mean(arr_delay, na.rm = TRUE)),
  by = carrier]

# In order to plot data in R usign ggplot, it is normally needed to reshape the data
# We want to have the data in long format for plotting with ggplot
dt = melt(mean2, id.vars = 'carrier')

# Take a look at the first rows
print(head(dt))

# Take a look at the help for ?geom_point and geom_line to find similar examples
# Here we take the carrier code as the x axis
# the value from the dt data.table goes in the y axis
```

```
# The variable column represents the color
p = ggplot(dt, aes(x = carrier, y = value, color = variable, group = variable)) +
  geom_point() + # Plots points
  geom_line() + # Plots lines
  theme_bw() + # Uses a white background
  labs(list(title = 'Mean arrival and departure delay by carrier',
    x = 'Carrier', y = 'Mean delay'))
print(p)

# Save the plot to disk
ggsave('mean_delay_by_carrier.png', p,
  width = 10.4, height = 5.07)
```

The code should produce an image such as the following −



Spotting problems using graphics and visualization:

Spotting problems using graphics:-

Occasionally, distortions or even errors result when plotting:

1) "Error in plot.new() : figure margins too large"

This error indicates that the margins of the particular plot are very large while the region allocated for the plot is too small. You can solve this problem by increasing the size of the plots pane.

2) Graphic with missing or distorted components

When legends, lines, text, or points are missing or "incorrectly" placed, this is often the result of R condensing the plot to fit the region. You can generally solve this by increasing or decreasing the plotting region.

3) Reset your graphics device

Resetting your graphics device will remove any leftover options or settings from previous plots. These might be causing undesired behavior or errors with your current plotting environment. See ?par and ?options for more details. For example:

```
> plot(cars)
> par(mfrow=c(2,2))
> plot(cars)
```

To fix this behavior, sometimes it is best to reset your graphics device and then try your plot again. Subsequent plots will use the default graphics settings. To reset your graphics device, call the following code from the console:

```
> dev.off()
```

UNIT II

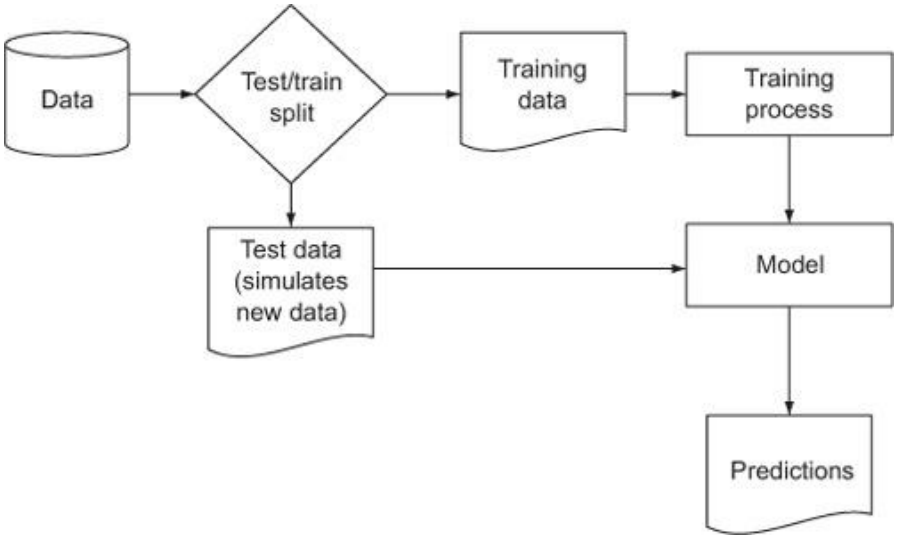
MODELING METHODS

MODELING METHODS: Choosing and evaluating models – mapping problems to machine learning tasks, evaluating models, validating models – cluster analysis – Kmeans algorithm, Naïve Bayes, Memorization Methods – KDD and KDD Cup 2009, building single variable models, building models using multi variable, Linear and logistic regression, unsupervised methods – cluster analysis, association rules.

Choosing and evaluating models:

As a data scientist, your ultimate goal is to solve a concrete business problem: increase look-to-buy ratio, identify fraudulent transactions, predict and manage the losses of a loan portfolio, and so on. Many different statistical modelling methods can be used to solve any given problem. Each statistical method will have its advantages and disadvantages for a given business goal and business constraints. This chapter presents an outline of the most common machine learning and statistical methods used in data science.

To make progress, you must be able to measure model quality during training and also ensure that your model will work as well in the production environment as it did on your training data. In general, we'll call these two tasks model *evaluation* and model *validation*. To prepare for these statistical tests, we always split our data into training data and test data,



We define model evaluation as quantifying the performance of a model. To do this we must find a measure of model performance that's appropriate to both the original business goal and the chosen modeling technique. For example, if we're predicting who would default on loans, we have a classification task, and measures like precision and recall are appropriate. If we instead are predicting revenue lost to defaulting loans, we have a scoring task, and measures like root mean square error (RMSE) are appropriate. The point is this: there are a number of measures the data scientist should be familiar with.

Mapping problems to machine learning tasks

Your task is to map a business problem to a good machine learning method. To use a real-world situation, let's suppose that you're a data scientist at an online retail company. There are a number of business problems that your team might be called on to address:

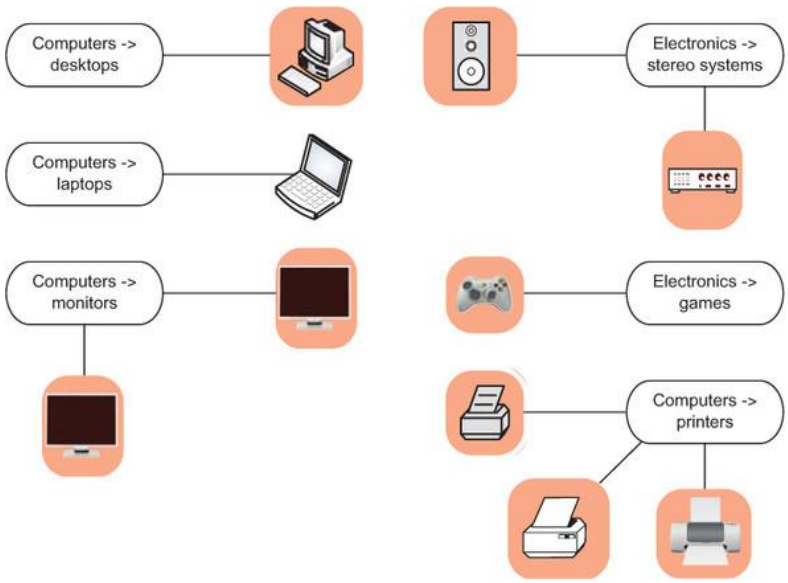
- Lginetcdir cwrq srsecmtuo tmigh ugb, adbse vn rcsb acnsiontarts
- Jfgidinynet aentldrufu ttnrcansioas
- Uigtireenmn icerp tyiaitescl (drk stkr rc hwchi s crepi nicaeres ffjw serdeace asels, nhs osoj aersv) lx avriuso upocstrd et urdctop assescl
- Ktgnermieni rxg rpxz hws kr pretnse rdtucpo isgnstli nwog costmruec hesaescr txl nc rjmx
- Youetmrs aeisngmeotnt: npirgguo rmssucteo jdrw miiarsl ncaghrpisu rbaihevo
- YyMtyv nlaivouat: wyx bzbm rpk omncpya slhudo espnd rk qud tcnraei vn raeshc nenegis
- Livlgtauna gmtniakre mnsagpaci
- Dngaigznri wno pcdrusto jrnv s ucodrpt laaogct

Your intended uses of the model have a big influence on what methods you should use. If you want to know how small variations in input variables affect outcome, then you likely want to use a regression method. If you want to know what single variable drives most of a categorization, then decision trees might be a good choice. Also, each business problem suggests a statistical approach to try. If you're trying to predict scores, some sort of regression is likely a good choice; if you're trying to predict categories, then something like random forests is probably a good choice.

Solving classification problems

Suppose your task is to automate the assignment of new products to your company's product categories, as shown in figure . This can be more complicated than it sounds. Products that come from different sources may have their own product classification that doesn't coincide with the one that you use on your retail site, or they may come without any classification at all. Many large online retailers use teams of human taggers to hand-categorize their products. This is not only labor-intensive, but inconsistent and error-prone. Automation is an attractive option; it's lab or-saving, and can improve the quality of the retail site.

. Assigning products to product categories



Evaluating Models :

After training a model, AutoML Tables uses the test dataset to evaluate the quality and accuracy of the new model, and provides an aggregate set of evaluation metrics indicating how well the model performed on the test dataset.

Using the evaluations metrics to determine the quality of your model depends on your business need and the problem you model is trained to solve. For example, there might be a higher cost to false positives than for false negatives, or vice versa. For regression models, does the delta between the prediction and the correct answer matter or not? These kinds of questions affect how you will look at your model evaluation metrics.

If you included a weight column in your training data, it does not affect evaluation metrics. Weights are considered only during the training phase.

Evaluation metrics for classification models

Classification models provide the following metrics:

- **AUC PR**: The area under the precision-recall (PR) curve. This value ranges from zero to one, where a higher value indicates a higher-quality model.
- **AUC ROC**: The area under the receiver operating characteristic (ROC) curve. This ranges from zero to one, where a higher value indicates a higher-quality model.
- **Accuracy**: The fraction of classification predictions produced by the model that were correct.
- **Log loss**: The cross-entropy between the model predictions and the target values. This ranges from zero to infinity, where a lower value indicates a higher-quality model.
- **F1 score**: The harmonic mean of precision and recall. F1 is a useful metric if you're looking for a balance between precision and recall and there's an uneven class distribution.
- **Precision**: The fraction of positive predictions produced by the model that were correct. (Positive predictions are the false positives and the true positives combined.)
- **Recall**: The fraction of rows with this label that the model correctly predicted. Also called "True positive rate".
- **False positive rate**: The fraction of rows predicted by the model to be the target label but aren't (false positive).

These metrics are returned for every distinct value of the target column. For multi-class classification models, these metrics are micro-averaged and returned as the summary metrics. For binary classification models, the metrics for the minority class are used as the summary metrics. The micro-averaged metrics are the expected value of each metric on a random sample from your dataset.

In addition to the above metrics, AutoML Tables provides two other ways to understand your classification model, the confusion matrix and a feature importance graph.

- **Confusion matrix**: The confusion matrix helps you understand where misclassifications occur (which classes get "confused" with each other). Each row represents ground truth for a specific label, and each column shows the labels predicted by the model.

  Confusion matrices are provided only for classification models with 10 or fewer values for the target column.

  Feature importance: AutoML Tables tells you how much each feature impacts this model. It is shown in the **Feature importance** graph. The values are provided as a percentage for each feature: the higher the percentage, the more strongly that feature impacted model training.

  You should review this information to ensure that all of the most important features make

  sense for your data and business problem.

How micro-averaged precision is calculated

The micro-averaged precision is calculated by adding together the number of true positives (TP) for each potential value of the target column and dividing it by the number of true positives (TP) and true negatives (TN) for each potential value.

$$precision_{micro}=\frac{TP_1+\ldots+TP_n}{TP_1+\ldots+TP_n+FP_1+\ldots+FP_n}$$

where

- TP1+…+TPn is the sum of the true positives for each of n classes
- FP1+…+FPn is the sum of false positives for each of n classes

Score threshold

The score threshold is a number that ranges from 0 to 1. It provides a way to specify the minimum confidence level where a given prediction value should be taken as true. For example, if you have a class that is quite unlikely to be the actual value, then you would want to lower the threshold for that class; using a threshold of .5 or higher would result in that class being predicted extremely rarely (or never).

A higher threshold decreases false positives, at the expense of more false negatives. A lower threshold decreases false negatives at the expense of more false positives.

Put another way, the score threshold affects precision and recall. A higher threshold results in an increase in precision (because the model never makes a prediction unless it is extremely sure) but the recall (the percentage of positive examples that the model gets right) decreases.

Evaluation metrics for regression models

Regression models provide the following metrics:

- **MAE**: The mean absolute error (MAE) is the average absolute difference between the target values and the predicted values. This metric ranges from zero to infinity; a lower value indicates a higher quality model.
- **RMSE**: The root-mean-square error metric is a frequently used measure of the differences between the values predicted by a model or an estimator and the values observed. This metric ranges from zero to infinity; a lower value indicates a higher quality model.
- **RMSLE**: The root-mean-squared logarithmic error metric is similar to RMSE, except that it uses the natural logarithm of the predicted and actual values plus 1. RMSLE penalizes under-prediction more heavily than over-prediction. It can also be a good metric when you don't want to penalize differences for large prediction values more heavily than for small prediction values. This metric ranges from zero to infinity; a lower value indicates a higher quality model. The RMSLE evaluation metric is returned only if all label and predicted values are non-negative.
- **r^2**: r squared (r^2) is the square of the Pearson correlation coefficient between the labels and predicted values. This metric ranges between zero and one; a higher value indicates a higher quality model.
- **MAPE**: Mean absolute percentage error (MAPE) is the average absolute percentage difference between the labels and the predicted values. This metric ranges between zero and infinity; a lower value indicates a higher quality model.

  MAPE is not shown if the target column contains any 0 values. In this case, MAPE is undefined.
- **Feature importance**: AutoML Tables tells you how much each feature impacts this model. It is shown in the **Feature importance** graph. The values are provided as a percentage for each feature: the higher the percentage, the more strongly that feature impacted model training.

You should review this information to ensure that all of the most important features make sense for your data and business problem. Learn more about explainability.

Getting the evaluation metrics for your model

To evaluate how well your model did on the test dataset, you inspect the evaluation metrics for your model.

ConsoleREST & CMD LINEJavaNode.jsPython

To see your model's evaluation metrics using the Google Cloud Console:

1. Go to the AutoML Tables page in the Google Cloud Console.

   Go to the AutoML Tables page
2. Select the **Models** tab in the left navigation pane, and select the model you want to get the evaluation metrics for.
3. Open the **Evaluate** tab.

The summary evaluation metrics are displayed across the top of the screen. For binary classification models, the summary metrics are the metrics of the minority class. For multi-class classification models, the summary metrics are the micro-averaged metrics.

For classification metrics, you can click on individual target values to see the metrics for that value.

validating models:

model validation is the process of verifying that models are providing satisfactory outcomes to their input data, in line with both qualitative and quantitative objectives. While partially consisting of a set of tried-and-true processes, model validation is a heterogeneous process that cannot easily be pinned down or characterized in general and applied to all models, creating opportunities for creativity and ingenuity. Model validation and verification ensures the effectiveness and accuracy of a trained model in preparation for use. Without model validation, the model may perform poorly and the training time will be irrecoverable. A model that is not properly validated will not be robust enough to adapt to new stress scenarios or may be too overfitted to receive and properly use new inputs. Different than model monitoring, model validation will take place before the model is put into place with the full dataset. Monitoring will regularly occur alongside a running model.

How to validate a model

There are two straightforward ways to statistically validate a model: one can evaluate the model on the data the model was trained on, or one can evaluate it on an external test set. The first method introduces the problem of overfitting: one can fit any dataset arbitrarily well at

the cost of creating a model brittle to extra data. If the model is perfected to one dataset, the model may not be able to use and identify correct outputs with new data and thus will not validate. One could take the case of trying to fit a curve through (x,y) pairs when given 100 of them as a training set.

A high-degree polynomial could fit the data in the training set exactly while being very brittle to data outside the training set. It is common practice instead to validate models using a test set. When originally given a data set, one can construct the test set by randomly extracting 10 to 20 percent of the data. In the case of the 100 (x,y) pairs, one could discover that the high-degree polynomial was overfitting easily by separating out a test set and evaluating the model against it. One might then choose to use a simpler model, such as a linear regression, which has a higher chance of passing model validation.

Many different statistical evaluation metrics can be used for model validation in general, including mean average error, mean squared error, and the ROC curve.

Model validation pitfalls

It is a mistake to believe that model validation is a purely quantitative or statistical process. For instance, a key part of model validation is ensuring that you have picked the right high-level statistical model. One could consider the example of training a system to predict the price of an item given an image of it. One could obtain reasonably good results by simply applying a logistic regression to the set of images. But this would ignore much better results that could potentially be obtained by applying a multiple layer convolutional neural network to the images.

It is thus important to perform thorough research of the machine learning literature as a part of model validation. The results of endless hours of work on a model that is a poor or mediocre choice for a given dataset can be surpassed by a simple glance at the right areas of

the arXiv. On the other hand, a model that is not exactly the right choice for a given data set, but still close to the optimum, can still be considered to pass model validation.

It is generally mistaken to take the perspective prevalent in Kaggle competitions that the goal is to squeeze every last drop of performance out of your model. Redoing a machine learning problem with a different model carries the problems of being expensive, time-consuming, and error-prone. It is often true that there is either one model that is "right" for the dataset, as is the case with large image datasets and neural networks. Or that there is no one "right" model, and several will be close to the optimum, as is the case in most non-image-based Kaggle competitions.

What is data validation?

Data validation is another key component of model validation. Data values can be corrupted or contain errors in ways that impair the results of model training. The integrity of data values can be verified by manually delving into sections of the data, programmatically searching through it, or by creating graphs. The integrity can also be checked qualitatively by ensuring that the data was drawn from a reliable, trustworthy, well-maintained and up-to-date source.

Data for the training and test sets should be drawn from the same probability distribution or as close as possible to achieve adequate results. In addition, there is a risk that models may be vulnerable to errors on specific input data values because they are poorly represented in the training set. If it happens to be the case that such a class of errors is possible, it is important to verify that the training set adequately covers all data inputs on which the model will need to be evaluated, or you may lose model validation. There are many methods of guaranteeing that the training set is adequate, including manual searching through the data or creating visual plots of it.

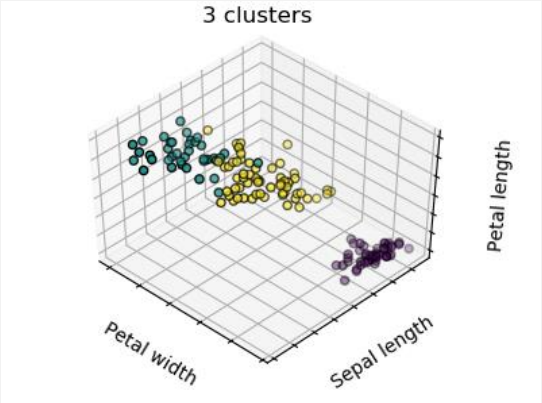
It is in general critical to have made correct assumptions about the similarities between the training set and the data the model will ultimately be evaluated on, which is again a qualitative process.

cluster analysis:

Cluster analysis is a technique whose purpose is to divide into ***groups*** (***clusters***) a collection of objects in such a way that:

1. The objects of the same group are the most similar possible.

2. The objects of the same group are the most similar possible (internal cohesion of the group). And the objects of different groups are as different as possible.



CLUSTER ANALYSIS

Cluster analysis is the grouping of objects based on their characteristics such that there is high intra-cluster similarity and low inter-cluster similarity.

WHAT IS CLUSTERING?

Cluster analysis is the grouping of objects such that objects in the same cluster are more similar to each other than they are to objects in another cluster. The classification into clusters is done using criteria such as smallest distances, density of data points, graphs, or various statistical distributions. Cluster analysis has wide applicability, including in unsupervised machine learning, data mining, statistics, Graph Analytics, image processing, and numerous physical and social science applications.

WHY CLUSTER ANALYSIS?

Data scientists and others use clustering to gain important insights from data by observing what groups (or clusters) the data points fall into when they apply a clustering algorithm to the data. By definition, unsupervised learning is a type of machine learning that searches for patterns in a data set with no pre-existing labels and a minimum of human intervention. Clustering can also be used for anomaly detection to find data points that are not part of any cluster, or outliers.

Clustering is used to identify groups of similar objects in datasets with two or more variable quantities. In practice, this data may be collected from marketing, biomedical, or geospatial databases, among many other places.

HOW IS CLUSTER ANALYSIS DONE?

It's important to note that analysis of clusters is not the job of a single algorithm. Rather, various algorithms usually undertake the broader task of analysis, each often being significantly different from others. Ideally, a clustering algorithm creates clusters where intra-cluster similarity is very high, meaning the data inside the cluster is very similar to one another. Also, the algorithm should create clusters where the inter-cluster similarity is much less, meaning each cluster contains information that's as dissimilar to other clusters as possible.

There are many clustering algorithms, simply because there are many notions of what a cluster should be or how it should be defined. In fact, there are more than 100 clustering algorithms that have been published to date. They represent a powerful technique for machine learning on unsupervised data. An algorithm built and designed for a specific type of cluster model will usually fail when set to work on a data set containing a very different kind of cluster model.

The common thread in all clustering algorithms is a group of data objects. But data scientists and programmers use differing cluster models, with each model requiring a different algorithm. Clusterings or sets of clusters are often distinguished as either hard clustering where each object belongs to a cluster or not, or soft clustering where each object belongs to each cluster to some degree.

This is all apart from so-called server clustering, which generally refers to a group of servers working together to provide users with higher availability and to reduce downtime as one server takes over when another fails temporarily.

Clustering analysis methods include:

- K-Means finds clusters by minimizing the mean distance between geometric points.
- DBSCAN uses density-based spatial clustering.

- Spectral clustering is a similarity graph-based algorithm that models the nearest-neighbor relationships between data points as an undirected graph.
- Hierarchical clustering groups data into a multilevel hierarchy tree of related graphs starting from a finest level (original) and proceeding to a coarsest level.

Clustering use cases

With the growing number of clustering algorithms available, it isn't surprising that clustering has become a staple methodology across a range of business and organizational types, with varying use cases. Clustering use cases include biological sequence analysis, human genetic clustering, medical image tissue clustering, market or customer segmentation, social network or search result grouping for recommendations, computer network anomaly detection, natural language processing for text grouping, crime cluster analysis, and climate cluster analysis. Below is a description of some examples.

- *Network traffic classification*. Organizations seek various ways of understanding the different types of traffic entering their websites, particularly what is spam and what traffic is coming from bots. Clustering is used to group together common characteristics of traffic sources, then create clusters to classify and differentiate the traffic types. This allows more reliable traffic blocking while enabling better insights into driving traffic growth from desired sources.
- *Marketing and sales*. Marketing success means targeting the right people or prospects in the right way. Clustering algorithms group together people with similar traits, perhaps based on their likelihood to purchase. With these groups or clusters defined, test marketing across them becomes more effective, helping to refine messaging to reach them.
- *Document analysis*. Any organization dealing with high volumes of documents will benefit by being able to organize them effectively and quickly as they're generated. That means being able to understand underlying themes in the documents, and then being able to compare that to other documents. Clustering algorithms examine text in documents, then group them into clusters of different themes. That way they can be speedily organized according to actual content.

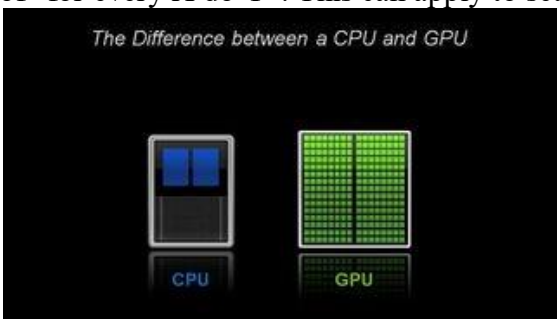
Data scientists and clustering

As noted, clustering is a method of unsupervised machine learning. Machine learning can process huge data volumes, allowing data scientists to spend their time analyzing the processed data and models to gain actionable insights. Data scientists use clustering analysis to gain some valuable insights from our data by seeing what groups the data points fall into when they apply a clustering algorithm.

ACCELERATING CLUSTER AND GRAPH ANALYTICS WITH GPUS

Cluster analysis plays a critical role in a wide variety of applications, but it's now facing the computational challenge due to the continuously increasing data volume. Parallel computing with GPUs is one of the most promising solutions to overcoming the computational challenge.

GPUs provide a great way to accelerate data-intensive analytics and graph analytics in particular, because of the massive degree of parallelism and the memory access-bandwidth advantages. A GPU's massively parallel architecture, consisting of thousands of small cores designed for handling multiple tasks simultaneously, is well suited for the computational task of "for every X do Y". This can apply to sets of vertices or edges within a large graph.



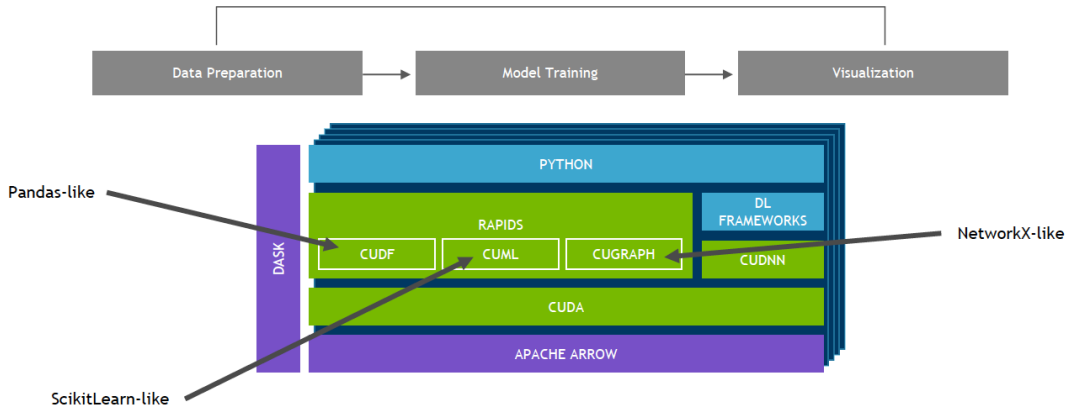
Cluster analysis is a problem with significant parallelism and can be accelerated by using GPUs. The NVIDIA Graph Analytics library (nvGRAPH) will provide both spectral and hierarchical clustering/partitioning techniques based on the minimum balanced cut metric in the future. The nvGRAPH library is freely available as part of the NVIDIA® CUDA® Toolkit. For more information about graphs, please refer to the Graph Analytics page.

GPU-ACCELERATED, END-TO-END DATA SCIENCE

The NVIDIA RAPIDS™ suite of open-source software libraries, built on CUDA-X AI™, provides the ability to execute end-to-end data science and analytics pipelines entirely on GPUs. It relies on NVIDIA CUDA® primitives for low-level compute optimization, but exposes that GPU parallelism and high-bandwidth memory speed through user-friendly Python interfaces.

RAPIDS's cuML machine learning algorithms and mathematical primitives follow the familiar scikit-learn-like API. Popular algorithms like K-means, XGBoost, and many others

are supported for both single-GPU and large data center deployments. For large datasets, these GPU-based implementations can complete 10-50X faster than their CPU equivalents.



K means algorithm:

K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm

, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

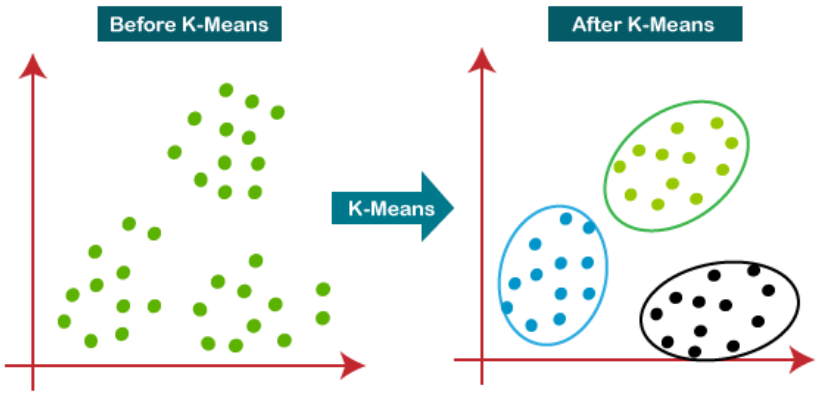
The k-means clustering

algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

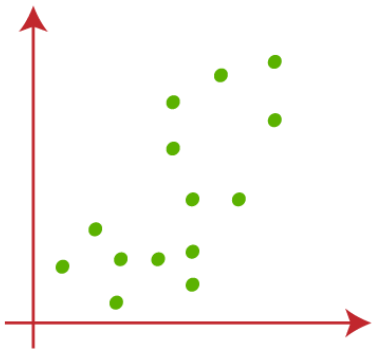
Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

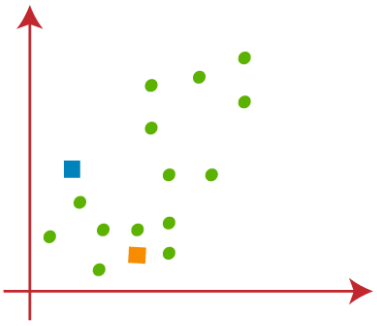
Step-7: The model is ready.

Let's understand the above steps by considering the visual plots:

Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:

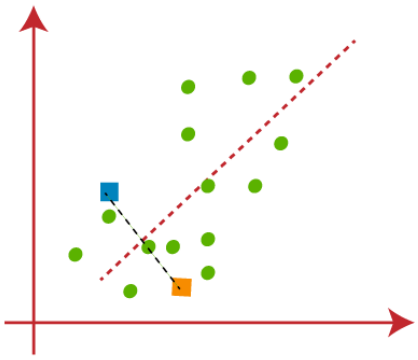


- Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the part of our dataset. Consider the below image:

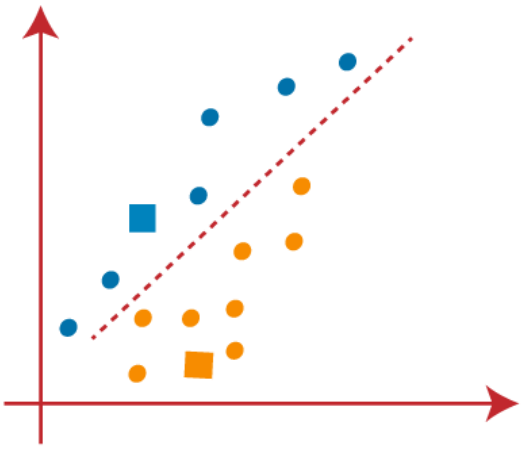


- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a

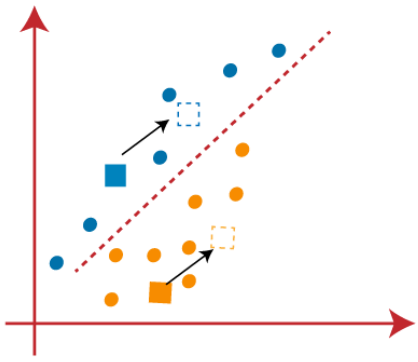
median between both the centroids. Consider the below image:



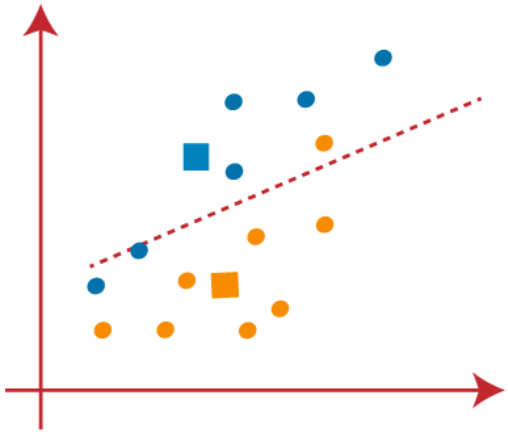
From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



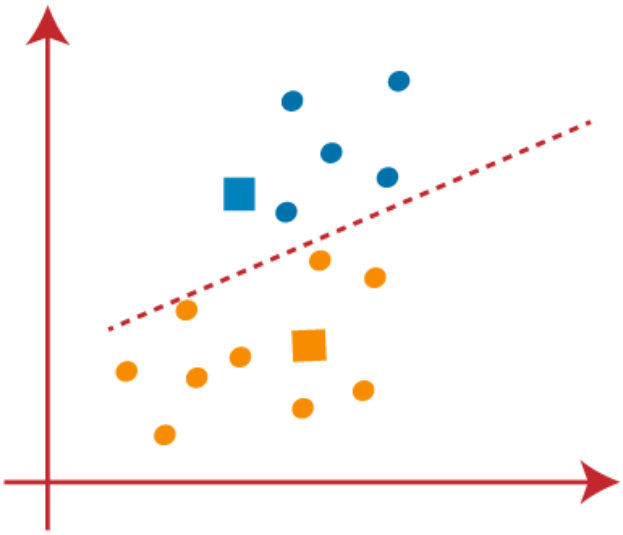
- As we need to find the closest cluster, so we will repeat the process by choosing **a new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:



- Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below
- image:



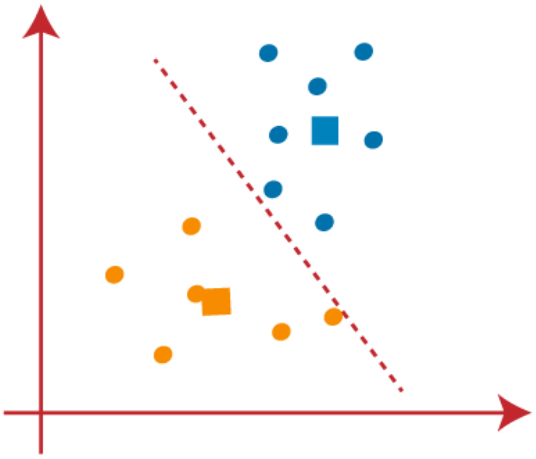
From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.



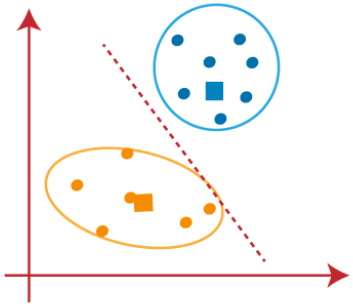
As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

- We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:

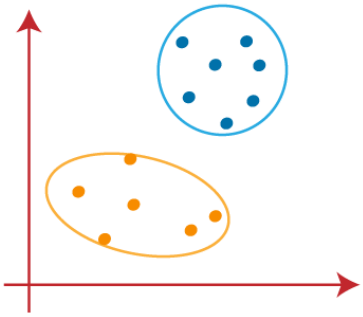
- As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:



- We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:



As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:



Introduction

In this article, we will discuss the mathematical intuition behind Naive Bayes Classifiers, and we'll also see how to implement this on Python.

This model is easy to build and is mostly used for large datasets. It is a probabilistic machine learning model that is used for classification problems. The core of the classifier depends on the Bayes theorem with an assumption of independence among predictors. That means changing the value of a feature doesn't change the value of another feature.

Why is it called Naive?

It is called Naive because of the assumption that 2 variables are independent when they may not be. In a real-world scenario, there is hardly any situation where the features are independent.

Naive Bayes does seem to be a simple yet powerful algorithm. But why is it so popular?

Since it is a probabilistic approach, the predictions can be made real quick. It can be used for both binary and multi-class classification problems.

Before we dive deeper into this topic we need to understand what is "*Conditional probability*", what is "*Bayes' theorem*" and how conditional probability help's us in Bayes' theorem.

Table of Contents

1. Conditional Probability for Naive Bayes
2. Bayes Rule
3. The Naive Bayes
4. Assumptions of Naive Bayes
5. Gaussian Naive Bayes
6. End Notes

Conditional Probability for Naive Bayes

Conditional probability is defined as the likelihood of an event or outcome occurring, based on the occurrence of a previous event or outcome. Conditional probability is calculated by multiplying the probability of the preceding event by the updated probability of the succeeding, or conditional, event.

Let's start understanding this definition with examples.

Suppose I ask you to pick a card from the deck and find the probability of getting a king given the card is clubs.

Observe carefully that here I have mentioned a **condition** that the card is clubs.

Now while calculating the probability my denominator will not be 52, instead, it will be 13 because the total number of cards in clubs is 13.

Since we have only one king in clubs the probability of getting a KING given the card is clubs will be 1/13 = 0.077.

Let's take one more example,

Consider a random experiment of tossing 2 coins. The sample space here will be:

S = {HH, HT, TH, TT}

If a person is asked to find the probability of getting a tail his answer would be 3/4 = 0.75

Now suppose this same experiment is performed by another person but now we give him the *condition* that **both the coins should have heads.** This means if event A: '*Both the coins should have heads'*, has happened then the elementary outcomes {HT, TH, TT} could not have happened. Hence in this situation, the probability of getting heads on both the coins will be 1/4 = 0.25

From the above examples, we observe that the probability may change if some additional information is given to us. This is exactly the case while building any machine learning model, we need to find the output given some features.

Mathematically, the conditional probability of event A given event B has already happened is given by:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Probability of event A given B has occured

Probability of event A occured and event B occured

Probability of event B

Bayes' Rule

Now we are prepared to state one of the most useful results in conditional probability: Bayes' Rule.

Bayes' theorem which was given by Thomas Bayes, a British Mathematician, in 1763 provides a means for calculating the probability of an event given some information.

Mathematically Bayes' theorem can be stated as:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}.$$

Basically, we are trying to find the probability of event A, given event B is true.

Here P(B) is called prior probability which means it is the probability of an event before the evidence

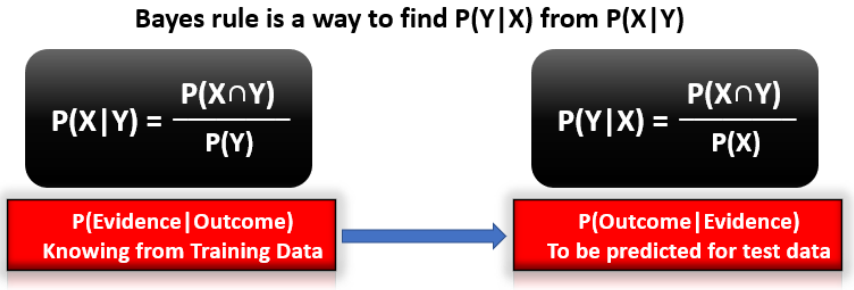
P(B|A) is called the posterior probability i.e., Probability of an event after the evidence is seen.

With regards to our dataset, this formula can be re-written as:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

Y: class of the variable

X: dependent feature vector (of size *n*)



What is Naive Bayes?

Bayes' rule provides us with the formula for the probability of Y given some feature X. In real-world problems, we hardly find any case where there is only one feature.

When the features are independent, we can extend Bayes' rule to what is called Naive Bakes which assumes that the features are independent that means changing the value of one feature doesn't influence the values of other variables and this is why we call this algorithm "*NAIVE*"

Naive Bayes can be used for various things like face recognition, weather prediction, Medical Diagnosis, News classification, Sentiment Analysis, and a lot more.

When there are multiple X variables, we simplify it by assuming that X's are independent, so

$$P(Y=k|X) = \frac{P(X|Y=k)*P(Y=k)}{P(X)}$$

For n number of X, the formula becomes **Naive Bayes**:

$$P(Y=k|X1,X2....Xn) = \frac{P(X1|Y=k)*P(X2|Y=k).....*P(Xn|Y=k)*P(Y=k)}{P(X1)*P(X2)....*P(Xn)}$$

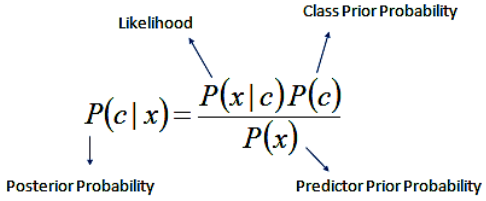
Which can be expressed as:

$$P(Y=k|X1,X2....Xn) = \frac{\mathrm{P(Y)}\prod_{\mathrm{i=1}}^{\mathrm{n}}\mathrm{P(Xi|Y)}}{P(X1)*P(X2)....*P(Xn)}$$

Since the denominator is constant here so we can remove it. It's purely your choice if you want to remove it or not. Removing the denominator will help you save time and calculations.

$$P(Y=k|X1,X2....Xn) \propto \mathrm{P(Y)}\prod_{\mathrm{i=1}}^{\mathrm{n}}\mathrm{P(Xi|Y)}$$

This formula can also be understood as:



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

There are a whole lot of formulas mentioned here but worry not we will try to understand all this with the help of an example.

Naive Bayes Example

Let's take a dataset to predict whether we can *pet an animal or not*.

	Animals	Size of Animal	Body Color	Can we Pet them
0	Dog	Medium	Black	Yes
1	Dog	Big	White	No
2	Rat	Small	White	Yes
3	Cow	Big	White	Yes
4	Cow	Small	Brown	No
5	Cow	Big	Black	Yes
6	Rat	Big	Brown	No
7	Dog	Small	Brown	Yes
8	Dog	Medium	Brown	Yes
9	Cow	Medium	White	No
10	Dog	Small	Black	Yes
11	Rat	Medium	Black	No
12	Rat	Small	Brown	No
13	Cow	Big	White	Yes

Assumptions of Naive Bayes

· All the variables are independent. That is if the animal is Dog that doesn't mean that Size will be Medium

· All the predictors have an equal effect on the outcome. That is, the animal being dog does not have more importance in deciding If we can pet him or not. All the features have equal importance.

We should try to apply the Naive Bayes formula on the above dataset however before that, we need to do some precomputations on our dataset.

We need to find $P(x_i|y_j)$ for each x_i in X and each y_j in Y. All these calculations have been demonstrated below:

Animals

	Yes	No	P(Yes)	P(No)
Dog	4	1	4/8	1/6
Rat	1	3	1/8	3/6
Cow	3	2	3/8	2/6
Total	8	6	100%	100%

Size of Animal

	Yes	No	P(Yes)	P(No)
Medium	2	2	2/8	2/6
Big	3	2	3/8	2/6
Small	3	2	3/8	2/6
Total	8	6	100%	100%

Body Color

	Yes	No	P(Yes)	P(No)
Black	3	1	3/8	1/6
White	3	2	3/8	2/6
Brown	2	3	2/8	3/6
Total	8	6	100%	100%

We also need the probabilities (P(y)), which are calculated in the table below. For example, P(Pet Animal = NO) = 6/14.

Play		P(yes)/P(no)
Yes	8	8/14
No	6	6/14
Total	14	100%

Now if we send our test data, suppose **test = (Cow, Medium, Black)**

Probability of petting an animal :

$$P(Yes|Test) = \frac{P(Animal = Cow|Yes) * P(Size = Medium|Yes) * P(Color = Black|Yes) * P(Yes)}{P(Test)}$$

$$P(Yes|Test) = \frac{3}{8} * \frac{2}{8} * \frac{3}{8} * \frac{8}{14} = 0.0200$$

And the probability of not petting an animal:

$$P(No|Test) = \frac{P(Animal = Cow|No) * P(Size = Medium|No) * P(Color = Black|No) * P(No)}{P(Test)}$$

$$P(No|Test) = \frac{2}{6} * \frac{2}{6} * \frac{1}{6} * \frac{6}{14} = 0.0079$$

We know P(Yes|Test)+P(No|test) = 1

So, we will normalize the result:

$$\mathrm{P(Yes|Test)} = \frac{0.0200}{0.0200 + 0.0079} = 0.7168$$

$$\mathrm{P(No|Test)} = \frac{0.0079}{0.0079 + 0.0200} = 0.2831$$

We see here that P(Yes|Test) > P(No|Test), so the prediction that we can pet this animal is **"Yes"**.

Gaussian Naive Bayes

So far, we have discussed how to predict probabilities if the predictors take up discrete values. But what if they are continuous? For this, we need to make some more assumptions regarding the distribution of each feature. The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(xi | y). Here we'll discuss Gaussian Naïve Bayes.

Gaussian Naïve Bayes is used when we assume all the continuous variables associated with each feature to be distributed according to **Gaussian Distribution.** Gaussian Distribution is also called Normal distribution.

The conditional probability changes here since we have different values now. Also, the (PDF) probability density function of a normal distribution is given by:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

We can use this formula to compute the probability of likelihoods if our data is continuous.

Endnotes

Naive Bayes algorithms are mostly used in face recognition, weather prediction, Medical Diagnosis, News classification, Sentiment Analysis, etc. In this article, we learned the mathematical intuition behind this algorithm. You have already taken your first step to master this algorithm and from here all you need is practice.

Memorization Methods – KDD and KDD Cup 2009:

Overview of the KDD Process

Reference: Fayyad, Piatetsky-Shapiro, Smyth, "From Data Mining to Knowledge Discovery: An Overview", in Fayyad, Piatetsky-Shapiro, Smyth, Uthurusamy, *Advances in Knowledge Discovery and Data Mining,* AAAI Press / The MIT Press, Menlo Park, CA, 1996, pp.1-34

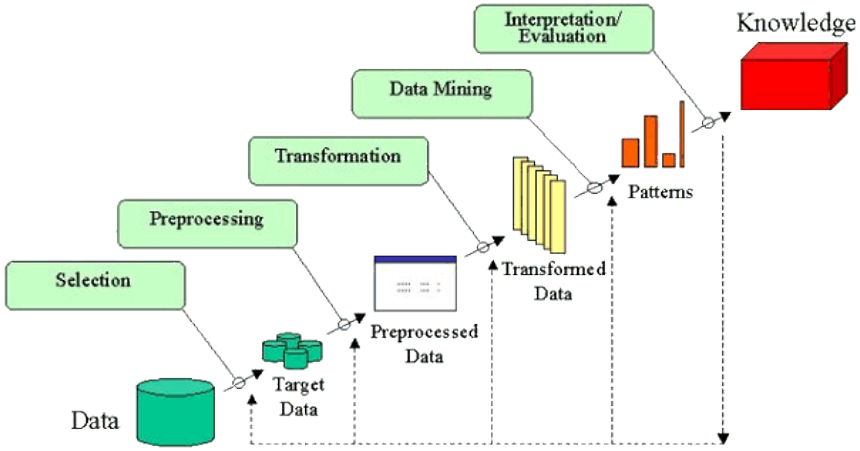
What is the KDD Process?

The term *Knowledge Discovery in Databases*, or KDD for short, refers to the broad process of finding knowledge in data, and emphasizes the "high-level" application of particular data mining methods. It is of interest to researchers in machine learning, pattern recognition, databases, statistics, artificial intelligence, knowledge acquisition for expert systems, and data visualization.

The unifying goal of the KDD process is to extract knowledge from data in the context of large databases.

It does this by using data mining methods (algorithms) to extract (identify) what is deemed knowledge, according to the specifications of measures and thresholds, using a database along with any required preprocessing, subsampling, and transformations of that database.

An Outline of the Steps of the KDD Process



The overall process of finding and interpreting patterns from data involves the repeated application of the following steps:

1. Developing an understanding of
    - the application domain
    - the relevant prior knowledge
    - the goals of the end-user
2. Creating a target data set: selecting a data set, or focusing on a subset of variables, or data samples, on which discovery is to be performed.
3. Data cleaning and preprocessing.
    - Removal of noise or outliers.
    - Collecting necessary information to model or account for noise.
    - Strategies for handling missing data fields.
    - Accounting for time sequence information and known changes.
4. Data reduction and projection.
    - Finding useful features to represent the data depending on the goal of the task.
    - Using dimensionality reduction or transformation methods to reduce the effective number of variables under consideration or to find invariant representations for the data.
5. Choosing the data mining task.
    - Deciding whether the goal of the KDD process is classification, regression, clustering, etc.
6. Choosing the data mining algorithm(s).
    - Selecting method(s) to be used for searching for patterns in the data.

- Deciding which models and parameters may be appropriate.
- Matching a particular data mining method with the overall criteria of the KDD process.

7. Data mining.
   - Searching for patterns of interest in a particular representational form or a set of such representations as classification rules or trees, regression, clustering, and so forth.
8. Interpreting mined patterns.
9. Consolidating discovered knowledge.

The terms *knowledge discovery* and *data mining* are distinct.

KDD refers to the overall process of discovering useful knowledge from data. It involves the evaluation and possibly interpretation of the patterns to make the decision of what qualifies as knowledge. It also includes the choice of encoding schemes, preprocessing, sampling, and projections of the data prior to the data mining step.
Data mining refers to the application of algorithms for extracting patterns from data without the additional steps of the KDD process.

Definitions Related to the KDD Process

Knowledge discovery in databases is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.

Data	A set of facts, F.
Pattern	An expression E in a language L describing facts in a subset F_E of F.
Process	KDD is a *multi-step process* involving data preparation, pattern searching, knowledge evaluation, and refinement with iteration after modification.
Valid	Discovered patterns should be true on new data with some degree of certainty. Generalize to the future (other data).
Novel	Patterns must be novel (should not be previously known).
Useful	Actionable; patterns should potentially lead to some useful actions.
Understandable	The process should lead to human insight. Patterns must be made understandable in order to facilitate a better understanding of the underlying data.

KDD Cup 2009

Customer Relationship Management (CRM) is a key element of modern marketing strategies. The KDD Cup 2009 offers the opportunity to work on large marketing databases from the French Telecom company Orange to predict the propensity of customers to switch provider (churn), buy new products or services (appetency), or buy upgrades or add-ons proposed to them to make the sale more profitable (up-selling).

The most practical way, in a CRM system, to build knowledge on customer is to produce scores. A score (the output of a model) is an evaluation for all instances of a target variable to explain (i.e. churn, appetency or up-selling). Tools which produce scores allow to project, on a given population, quantifiable information. The score is computed using input variables which describe instances. Scores are then used by the information system (IS), for example, to personalize the customer relationship. An industrial customer analysis platform able to build prediction models with a very large number of input variables has been developed by Orange Labs. This platform implements several processing methods for instances and variables selection, prediction and indexation based on an efficient model combined with variable selection regularization and model averaging method. The main characteristic of this platform is its ability to scale on very large datasets with hundreds of thousands of instances and thousands of variables. The rapid and robust detection of the variables that have most contributed to the output prediction can be a key factor in a marketing application.

The challenge is to beat the in-house system developed by Orange Labs. It is an opportunity to prove that you can deal with a very large database, including heterogeneous noisy data (numerical and categorical variables), and unbalanced class distributions. Time efficiency is often a crucial point. Therefore part of the competition will be time-constrained to test the ability of the participants to deliver solutions quickly.

building single variable models

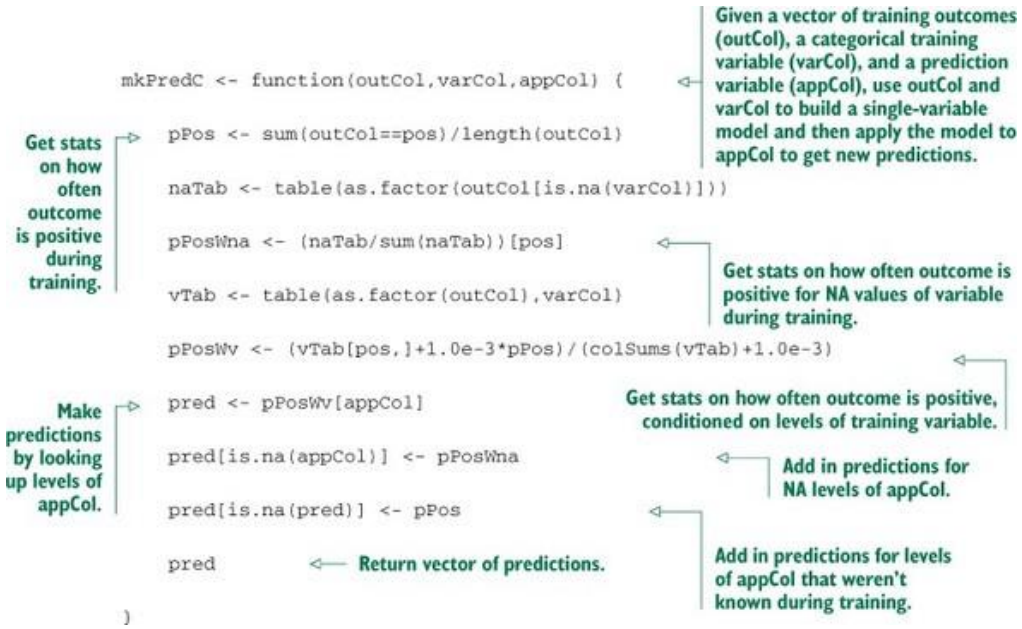
Single-variable models are simply models built using only one variable at a time. Single-variable models can be powerful tools, so it's worth learning how to work well with them before jumping into general modeling (which almost always means multiple variable models). We'll show how to build single-variable models from both categorical and numeric variables. By the end of this section, you should be able to build, evaluate, and cross-validate single-variable models with confidence.

Single-variable models can be thought of as being simple memorizations or summaries of the training data. This is especially true for categorical variables where the model is essentially a contingency table or pivot table, where for every level of the variable we record the distribution of training outcomes). Some sophisticated ideas (like smoothing, regularization, or shrinkage) may be required to avoid overfitting and to build good single-variable models. But in the end, single-variable models essentially organize the training data into a number of

subsets indexed by the predictive variable and then store a summary of the distribution of outcome as their future prediction. These models are atoms or sub-assemblies that we sum in different ways to get the rest of the models of this chapter.

From this, we see variable 218 takes on two values plus NA, and we see the joint distribution of these values against the churn outcome. At this point it's easy to write down a single-variable model based on variable 218.

Function to build single-variable models for categorical variables



Building models using multi variable:

When building a model, the first thing to check is if the model even works on the data it was trained from. In this section, we do this by introducing quantitative measures of model performance. From an evaluation point of view, we group model types this way:

Classification

For most model evaluations, we just want to compute one or two summary scores that tell us if the model is effective. To decide if a given score is high or low, we have to appeal to a few ideal models: a null model (which tells us what low performance looks like), a Bayes rate model (which tells us what high performance looks like), and the best single-variable model (which tells us what a simple model can achieve). We outline the concepts in

In this section, we'll present the standard measures of model quality, which are useful in model construction. In all cases, we suggest that in addition to the standard model quality assessments you try to design your own custom "business-oriented loss function" with your project sponsor or client. Usually this is as simple as assigning a notional dollar value to each outcome and then seeing how your model performs under that criterion. Let's start with how to evaluate classification models and then continue from there.

Ideal models to calibrate against

Ideal model	Purpose

Ideal model	Purpose
Null model	A null model is the best model of a very simple form you're trying to outperform. The two most typical null model choices are a model that is a single constant (returns the same answer for all situations) or a model that is independent (doesn't record any important relation or interaction between inputs and outputs). We use null models to lower-bound desired performance, so we usually compare to a best null model. For example, in a categorical problem, the null model would always return the most popular category (as this is the easy guess that is least often wrong); for a score model, the null model is often the average of all the outcomes (as this has the least square deviation from all of the outcomes); and so on. The idea is this: if you're not out-performing the null model, you're not delivering value. Note that it can be hard to do as good as the best null model, because even though the null model is simple, it's privileged to know the overall distribution of the items it will be quizzed on. We always assume the null model we're comparing to is the best of all possible null models.
Bayes rate model	A Bayes rate model (also sometimes called a *saturated model*) is a best possible model given the data at hand. The Bayes rate model is the perfect model and it only makes mistakes when there are multiple examples with the exact same set of known facts (same *x*s) but different outcomes (different *y*s). It isn't always practical to construct the Bayes rate model, but we invoke it as an upper bound on a model evaluation score. If we feel our model is performing significantly above the null model rate and is approaching the Bayes rate, then we can stop tuning. When we have a lot of data and very few modeling features, we can estimate the Bayes error rate. Another way to estimate the Bayes rate is to ask several different people to score the same small sample of your data; the found inconsistency rate can be an estimate of the Bayes rate.[a]
Single-variable models	We also suggest comparing any complicated model against the best single-variable model you have available for how to convert single variables into single-variable models). A complicated model can't be justified if it doesn't outperform the best single-variable model available from your training data. Also, business analysts have many tools for building effective single-variable models (such as pivot tables), so if your client is an analyst, they're likely looking for performance above this level.

The first part of the `summary()` is how the `lm()` model was constructed:

```
Call:
lm(formula = log(PINCP, base = 10) ~ AGEP + SEX + COW + SCHL,
    data = dtrain)
```

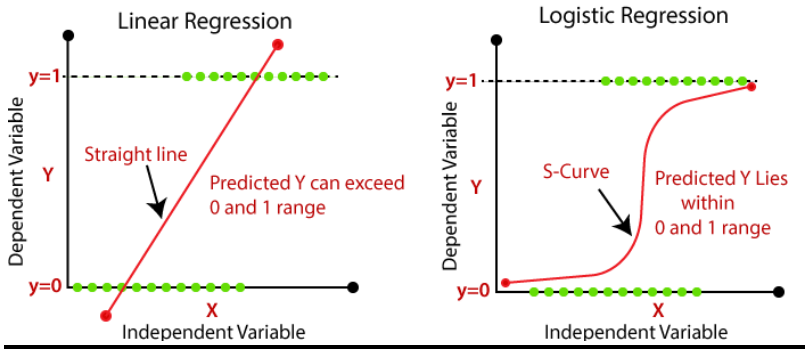
we looked at how to use linear regression to model and predict quantitative output, and how to use logistic regression to predict class probabilities. Linear and logistic regression models are powerful tools, especially when you want to understand the relationship between the input variables and the output. They're robust to correlated variables (when regularized), and logistic regression preserves the marginal probabilities of the data. The primary shortcoming of both these models is that they assume that the relationship between the inputs and the output is monotone. That is, if more is good, than much more is always better.

You want R-squared to be fairly large (1.0 is the largest you can achieve) and R-squareds that are similar on test and training. A significantly lower R-squared on test data is a symptom of an overfit model that looks good in training and won't work in production. In our case, our R-squareds were 0.338 on training and 0.261 on test. We'd like to see R-squares higher than this (say, 0.7–1.0). So the model is of low quality, but not substantially overfit.
Once the model is fit, scoring is fast.

While the sponsor is the role that represents the business interest, the client is the role that represents the model's end users' interests. Sometimes the sponsor and client roles may be filled by the same person. Again, the data scientist may fill the client role if they can weight business trade-offs, but this isn't ideal.
As with the sponsor, you should keep the client informed and involved. Ideally you'd like to have regular meetings with them to keep your efforts aligned with the needs of the end users. Generally the client belongs to a different group in the organization and has other responsibilities beyond your project. Keep meetings focused, present results and progress in terms they can understand, and take their critiques to heart. If the end users can't or won't use your model, then the project isn't a success, in the long run.

Linear and logistic regression:

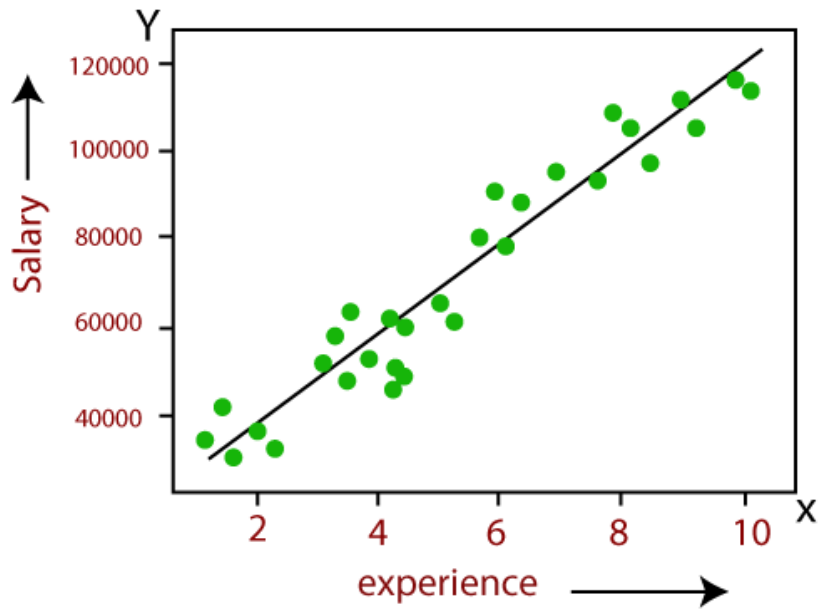
Linear Regression and Logistic Regression are the two famous Machine Learning Algorithms which come under supervised learning technique. Since both the algorithms are of supervised in nature hence these algorithms use labeled dataset to make the predictions. But the main difference between them is how they are being used. The Linear Regression is used for solving Regression problems whereas Logistic Regression is used for solving the Classification problems. The description of both the algorithms is given below along with difference table.



Linear Regression:

- Linear Regression is one of the most simple Machine learning algorithm that comes under Supervised Learning technique and used for solving regression problems.

- It is used for predicting the continuous dependent variable with the help of independent variables.
- The goal of the Linear regression is to find the best fit line that can accurately predict the output for the continuous dependent variable.
- If single independent variable is used for prediction then it is called Simple Linear Regression and if there are more than two independent variables then such regression is called as Multiple Linear Regression.
- By finding the best fit line, algorithm establish the relationship between dependent variable and independent variable. And the relationship should be of linear nature.
- The output for Linear regression should only be the continuous values such as price, age, salary, etc. The relationship between the dependent variable and independent variable can be shown in below image:



In above image the dependent variable is on Y-axis (salary) and independent variable is on x-axis(experience). The regression line can be written as:

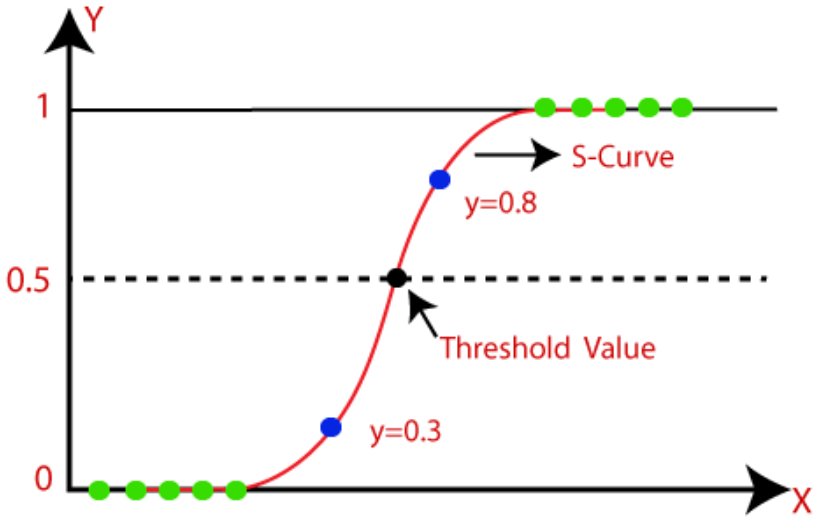
$$y= a_0+a_1x+ \varepsilon$$

Where, a_0 and a_1 are the coefficients and ε is the error term.

Logistic Regression:

- Logistic regression is one of the most popular Machine learning algorithm that comes under Supervised Learning techniques.

- It can be used for Classification as well as for Regression problems, but mainly used for Classification problems.
- Logistic regression is used to predict the categorical dependent variable with the help of independent variables.
- The output of Logistic Regression problem can be only between the 0 and 1.
- Logistic regression can be used where the probabilities between two classes is required. Such as whether it will rain today or not, either 0 or 1, true or false etc.
- Logistic regression is based on the concept of Maximum Likelihood estimation. According to this estimation, the observed data should be most probable.
- In logistic regression, we pass the weighted sum of inputs through an activation function that can map values in between 0 and 1. Such activation function is known as **sigmoid function** and the curve obtained is called as sigmoid curve or S-curve. Consider the below image:



- The equation for logistic regression is:

$$\log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

Difference between Linear Regression and Logistic Regression:

Linear Regression	Logistic Regression
Linear regression is used to predict the continuous dependent variable using a given set of independent variables.	Logistic Regression is used to predict the categorical dependent variable using a given set of independent variables.

Linear Regression is used for solving Regression problem.	Logistic regression is used for solving Classification problems.
In Linear regression, we predict the value of continuous variables.	In logistic Regression, we predict the values of categorical variables.
In linear regression, we find the best fit line, by which we can easily predict the output.	In Logistic Regression, we find the S-curve by which we can classify the samples.
Least square estimation method is used for estimation of accuracy.	Maximum likelihood estimation method is used for estimation of accuracy.
The output for Linear Regression must be a continuous value, such as price, age, etc.	The output of Logistic Regression must be a Categorical value such as 0 or 1, Yes or No, etc.
In Linear regression, it is required that relationship between dependent variable and independent variable must be linear.	In Logistic regression, it is not required to have the linear relationship between the dependent and independent variable.
In linear regression, there may be collinearity between the independent variables.	In logistic regression, there should not be collinearity between the independent variable.

Unsupervised methods – cluster analysis:

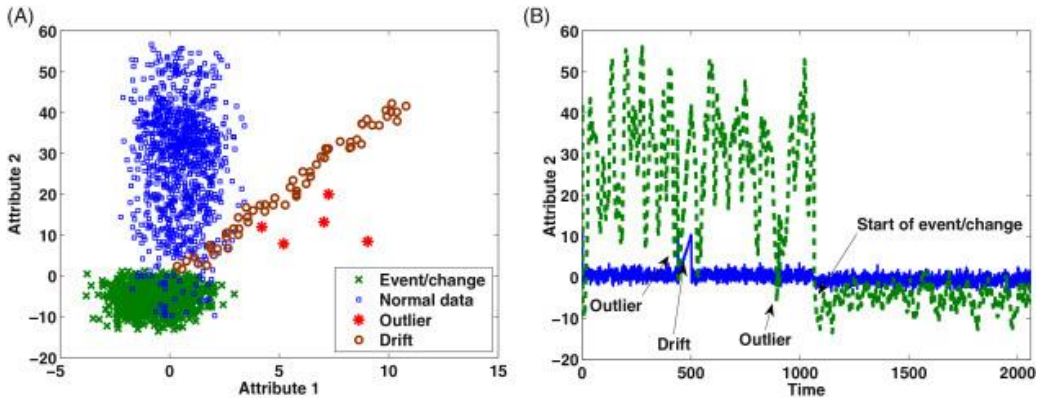
Clustering is an unsupervised data science technique where the records in a dataset are organized into different logical groupings. The data are grouped in such a way that records inside the same group are more similar than records outside the group. Clustering has a wide variety of applications ranging from market segmentation to customer segmentation, electoral grouping, web analytics, and outlier detection. Clustering is also used as a data compression technique and data preprocessing technique for supervised tasks. Many different data science approaches are available to cluster the data and are developed based on proximity between the records, density in the dataset, or novel application of neural networks. *k*-Means clustering, density clustering, and self-organizing map techniques are reviewed in the chapter along with implementations using RapidMiner.

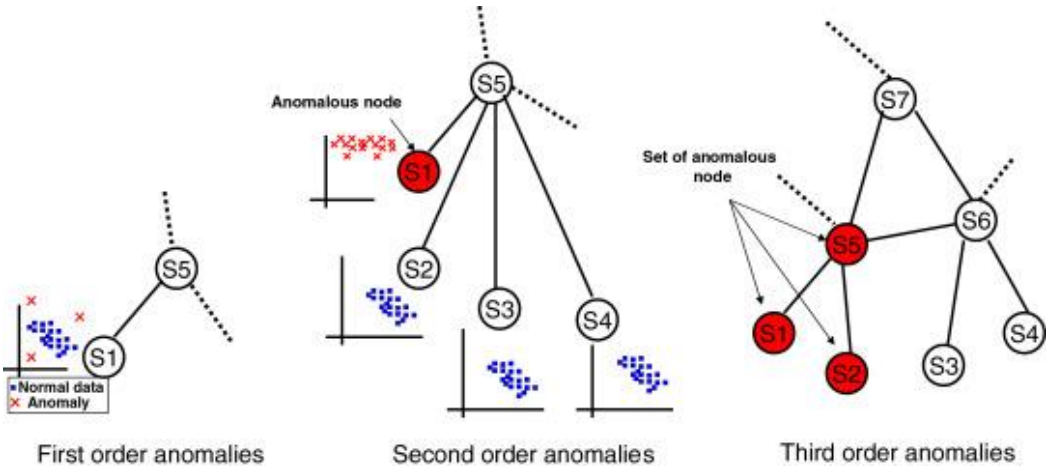
A wide array of data based process monitoring techniques have been developed for the online classification of process data into normal and faulty classes (Ge 2013), however many of these methods are “supervised”, or require that the training data for the models be organized

into labelled groups. In real plants this is rarely true, and unsupervised data mining algorithms are needed to find meaningful clusters corresponding to fault data.

With any fault diagnosis system, a major obstacle to implementation is that process data are often uncategorized. Algorithms need to (1) separate fault data from normal data, (2) train a model based on statistics or a supervised learning technique for fault detection, and (3) assist with the identification and management of new faults. It is important for the larger acceptance of these methods that those tasks are all performed in a way that is simple to understand for non-experts in data science and easy to deploy on multiple units around a plant with low overhead.

This research studies the potential for data clustering and unsupervised learning to automatically separate data into groups significant to abnormal event detection. Vekatasubramanian (2009) calls for a "tool box" based approach in which a data modeller is comfortable with using a diverse array of modelling techniques to solve a given problem. In that spirit, this research evaluates a set of knowledge discovery techniques for mining databases to solve process monitoring problems. Sensor data from an industrial separations tower, reactor, and the Tennessee Eastman simulation are studied and used to compare different dimensionality reduction and clustering techniques in terms of their effectiveness in extracting knowledge from process databases.





Association rules.:

Association Rule Learning

Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable. It tries to find some interesting relations or associations among the variables of dataset. It is based on different rules to discover the interesting relations between variables in the database.

The association rule learning is one of the very important concepts of machine learning, and it is employed in **Market Basket analysis, Web usage mining, continuous production, etc.** Here market basket analysis is a technique used by the various big retailer to discover the associations between items. We can understand it by taking an example of a supermarket, as in a supermarket, all products that are purchased together are put together.

For example, if a customer buys bread, he most likely can also buy butter, eggs, or milk, so these products are stored within a shelf or mostly nearby. Consider the below diagram:

1. **Apriori**
2. **Eclat**

3. **F-P Growth Algorithm**

We will understand these algorithms in later chapters.

How does Association Rule Learning work?

Association rule learning works on the concept of If and Else Statement, such as if A then B.

Here the If element is called **antecedent**, and then statement is called as **Consequent**. These types of relationships where we can find out some association or relation between two items is known *as single cardinality*. It is all about creating rules, and if the number of items increases, then cardinality also increases accordingly. So, to measure the associations between thousands of data items, there are several metrics. These metrics are given below:

- **Support**
- **Confidence**
- **Lift**

Let's understand each of them:

Support

Support is the frequency of A or how frequently an item appears in the dataset. It is defined as the fraction of the transaction T that contains the itemset X. If there are X datasets, then for transactions T, it can be written as:

$$\text{Supp(X)} = \frac{Freq(X)}{T}$$

Confidence

Confidence indicates how often the rule has been found to be true. Or how often the items X and Y occur together in the dataset when the occurrence of X is already given. It is the ratio of the transaction that contains X and Y to the number of records that contain X.

$$\text{Confidence}=\frac{Freq(X,Y)}{Freq(X)}$$

Lift

It is the strength of any rule, which can be defined as below formula:

$$\text{Lift}=\frac{Supp(X,Y)}{Supp(X)\times Supp(Y)}$$

It is the ratio of the observed support measure and expected support if X and Y are independent of each other. It has three possible values:

- If **Lift= 1**: The probability of occurrence of antecedent and consequent is independent of each other.
- **Lift>1**: It determines the degree to which the two itemsets are dependent to each other.
- **Lift<1**: It tells us that one item is a substitute for other items, which means one item has a negative effect on another.

Types of Association Rule Learning

Association rule learning can be divided into three algorithms:

Apriori Algorithm

This algorithm uses frequent datasets to generate association rules. It is designed to work on the databases that contain transactions. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset efficiently.

It is mainly used for market basket analysis and helps to understand the products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

Eclat Algorithm

Eclat algorithm stands for **Equivalence Class Transformation**. This algorithm uses a depth-first search technique to find frequent itemsets in a transaction database. It performs faster execution than Apriori Algorithm.

F-P Growth Algorithm

The F-P growth algorithm stands for **Frequent Pattern**, and it is the improved version of the Apriori Algorithm. It represents the database in the form of a tree structure that is known as a frequent pattern or tree. The purpose of this frequent tree is to extract the most frequent patterns.

Applications of Association Rule Learning

It has various applications in machine learning and data mining. Below are some popular applications of association rule learning:

- **Market Basket Analysis:** It is one of the popular examples and applications of association rule mining. This technique is commonly used by big retailers to determine the association between items.
- **Medical Diagnosis:** With the help of association rules, patients can be cured easily, as it helps in identifying the probability of illness for a particular disease.
- **Protein Sequence:** The association rules help in determining the synthesis of artificial Proteins.
- It is also used for the **Catalog Design** and **Loss-leader Analysis** and many more other applications.

UNIT III

INTRODUCTION TO R Language: Reading and getting data into R, viewing named objects, Types of Data items, the structure of data items, examining data structure, workingwith history commands, saving your work in R.

PROBABILITY DISTRIBUTIONS in R - Binomial, Poisson, Normal distributions. Manipulating objects - data distribution.

INTRODUCTION TO R LANGUAGE:

Reading files into R

Usually we will be using data already in a file that we need to read into R in order to work on it. R can read data from a variety of file formats—for example, files created as text, or in Excel, SPSS or Stata. We will mainly be reading files in text format .txt or .csv (comma-separated, usually created in Excel).

To read an entire data frame directly, the external file will normally have a special form

- The first line of the file should have a *name* for each variable in the data frame.
- Each additional line of the file has as its first item a *row label* and the values for each variable.

Here we use the example dataset called airquality.csv and airquality.txt

Inputs file form with names and row labels:

```
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
```

...

By default numeric items (except row labels) are read as numeric variables. This can be changed if necessary.

The function **`read.table()`** can then be used to read the data frame directly

```
> airqual <-
read.table("C:/Desktop/airquality.txt")
```

Similarly, to read .csv files the **read.csv()** function can be used to read in the data frame directly

[Note: I have noticed that occasionally you'll need to do a double slash in your path //. This seems to depend on the machine.]

```
> airqual <- read.csv("C:/Desktop/airquality.csv")
```

In addition, you can read in files using the file.choose() function in R. After typing in this command in R, you can manually select the directory and file where your dataset is located.

1. Read the airquality.csv file into R using the read.csv command.
2. Read the airquality.txt file into R using the file.choose() command

Occasionally, you will need to read in data that does not already have column name information. For example, the dataset BOD.txt looks like this:

```
1     8.3

2    10.3

3    19.0

4    16.0

5    15.6

7    19.8
```

Initially, there are no column names associated with the dataset. We can use the **colnames()** command to assign column names to the dataset. Suppose that we want to assign columns, "Time" and "demand" to the BOD.txt dataset. To do so we do the following

```
> bod <- read.table("BOD.txt", header=F)

> colnames(bod) <- c("Time","demand")

> colnames(bod)

[1] "Time"    "demand"
```

The first command reads in the dataset, the command "header=F" specifies that there are no column names associated with the dataset.

Read in the **cars.txt** dataset and call it car1. Make sure you use the **"header=F"** option to specify that there are no column names associated with the dataset. Next, assign "speed" and "dist" to be the first and second column names to the car1 dataset.

The two videos below provide a nice explanations of different methods to read data from a spreadsheet into an R dataset.

Getting data into R:

Getting data into R

One of the biggest hurdles that students and novice **R** users have is importing data into **R**. A number of recent packages have made this process easier and, importantly in the age of big data, faster. Most of the packages outlined below are part of Hadley Wickham's tidyverse and owe their speed to calling C or C++ libraries from **R**.

The readr package

The **readr** package provides new functions for importing tabular data into **R**. Specifically, the functions `read_table()`, `read_csv()`, `read_delim()` are intended as fast (around 10 times faster) replacements for the base **R** `read.table()`, `read.csv()`, `read.delim()`.
The **readr** functions do not convert strings to factors by default, are able to parse dates and times and can automatically determine the data types in each column (it does this by doing an initial check of the first 1000 rows). You can even import compressed files and they will be automatically decompressed and read into **R**. There is also file writing functionality, with `write_csv()`, `write_tsv()` and `write_delim()`. If you want even more speediness in your data importing, you might also consider the `fread()` function from the **data.table** package.

The readxl package

It used to be that the most reliable way to get data from Excel into **R** was to first save it as a tab (or comma) delimited text file. The easiest way to import tabular data from `xls` and `xlsx` formats files is to use the **readxl** package. Importantly it has no external dependencies, so is very straightforward to install and use on all platforms. The syntax is very similar to the **readr** package functions, for example you specify the file name and the sheet of interest, `read_excel("spreadsheet.xlsx", sheet = "data")`. You can also write a data frame to an Excel file using the **writexl** package.

The haven package

The **haven** package provides functions for importing from SAS, SPSS and Stata file formats, `read_sas()`, `read_sav()` and `read_dta()`. This functionality is similar to that available in the base **R** foreign package but is often faster, can read SAS7BDAT files and formats, works with Stata 14 and 14 files. Following is the code to read SAS7BDAT.

```
install.packages("haven")
library(haven)
dat = read_sas("path to file", "path to formats catalog")
```

The returned object will be a data frame where SAS variable labels are attached as an attribute to each variable. When a variable is attached to a format in SAS and the formats are stored in a library, its path also needs to be supplied. Missing values in numeric variables should be seamlessly converted. Missing values in character variables are converted to the empty string. To convert empty strings to missing values, use `zap_empty()`, for example,

```
dat$x1 = zap_empty(dat$x1)
```

SAS, Stata and SPSS all have the notion of a "labelled"" variable. These are similar to categorical factor variables in R, but integer, numeric and character vectors can be labelled and not every value must be associated with a label. To turn a labelled variable into a standard factor **R** variable use the `as_factor()` function,

```
dat$facvar = as_factor(dat$facvar)
```

The **haven** package is under active development and becoming increasingly robust. If you have difficulties loading a file, try using the development version on GitHub:

```
devtools::install_github("hadley/haven")
```

For example, consider the National Youth Tobacco Survey (NYTS) from the CDC website. After downloading the files (and installing the development version from GitHub) the data can be imported into R using

```
x =
haven::read_sas("nyts2014_dataset.sas7bdat","nyts2014_formats.sas7bcat")
# convert qn1 to a factor:
x$qn1 = as_factor(x$qn1)
```

The rio package

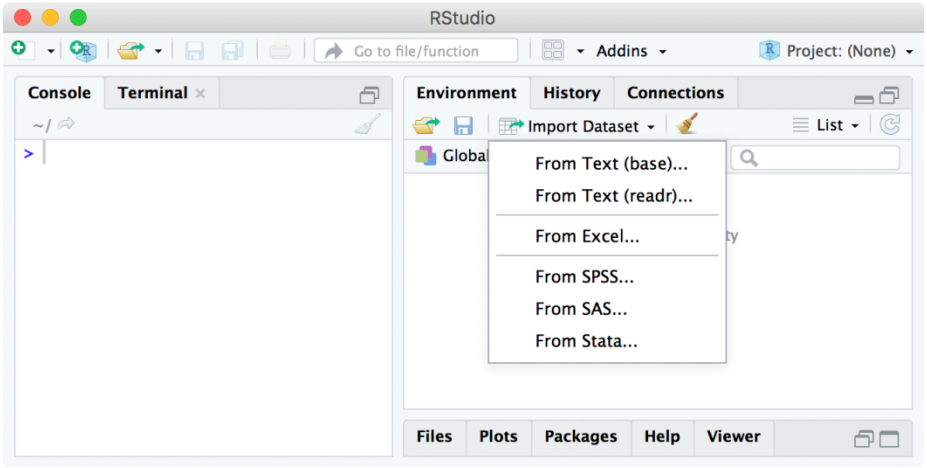
The **rio** package describes itself as "a Swiss army knife for data I/O". It unifies many of the above methods by providing the wrapper function `import()` that takes as an input the path to a data file. It then uses the file extension to determine the file type and imports the data into **R**. The

one function can be used to import standard text files, RData, JSON, Stata, SPSS, Excel, SAS, XML, Minitab and many more. There is an analogous `export()` function that allows users to similarly easily export data to various file types.

Using RStudio to import data

RStudio allows users to import various file types via a graphical user interface, which is perfect for novice users and experts alike as they get used to the new functions and customisation options. Once you've clicked through the various options, it will output the required code at the console so that you can see exactly what was done to get the data in and edit the code as necessary for next time.

In the image below, text files, such as CSV files, can be read in using either the **readr** package (e.g. `read_csv()`) or using **base** R functions such as `read.csv()`. Note you don't need to load any additional packages to use **base** R functions, they're available to use whenever R is running.



viewing named objects,

The team at RStudio have put together a webinar on getting data into **R** which is well worth watching.

Once you've got your data into **R**, you'll probably need to restructure it in some way prior to analysis. To help with this, you may want to take a look at the **tidyr** package provides a suite of functions to get your data set in a standardised format, such that each observation is a row, each variable is a column and there are no data in the labels.

A list is an object in R Language which consists of heterogeneous elements. A list can even contain matrices, data frames, or functions as its elements. The list can be created using **list()** function in R. Named list is also created with the same function by specifying the names of the elements to access them. Named list can also be created using **names()** function to specify the names of elements after defining the list. In this article, we'll learn to create named list in R using two different methods and different operations that can be performed on named lists.

Syntax: *names(x) <- value*

Parameters:
x: *represents an R object*
value: *represents names that has to be given to elements of* ***x*** *object*

Creating a Named List

A Named list can be created by two methods. The first one is by allocating the names to the elements while defining the list and another method is by using **names()** function.

Example 1:

In this example, we are going to create a named list without using **names()** function.

```
# Defining a list with names
x <- list(mt = matrix(1:6, nrow = 2),
                  lt = letters[1:8],
                  n = c(1:10))

# Print whole list
cat("Whole List:\n")
print(x)
```

Output:

```
Whole List:

$mt

     [,1] [,2] [,3]

[1,]    1    3    5

[2,]    2    4    6


$lt

[1] "a" "b" "c" "d" "e" "f" "g" "h"


$n

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

Example 2:

In this example, we are going to define the names of elements of the list using **names()** function after defining the list.

```
# Defining list

x <- list(matrix(1:6, nrow = 2),

          letters[1:8],
```

```
        c(1:10))

  # Print whole list

cat("Whole list:\n")

print(x)
```

Output:

```
Whole list:
[[1]]
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6


[[2]]
[1] "a" "b" "c" "d" "e" "f" "g" "h"


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

Accessing components of Named List

Components of a named list can be easily accessed by **$** operator.

Example:

```
# Defining a list with names

x <- list(mt = matrix(1:6, nrow = 2),

        lt = letters[1:8],

        n = c(1:10))

  # Print list elements using the names given
```

```
# Prints element of the list named "mt"

cat("Element named 'mt':\n")

print(x$mt)

cat("\n")

  # Print element of the list named "n"

cat("Element named 'n':\n")

print(x$n)
```

Output:

```
Element named 'mt':
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6


Element named 'n':
 [1]  1  2  3  4  5  6  7  8  9 10
```

Modifying components of Named List

Components of named list can be modified by assigning new values to them.

Example:

```
# Defining a named list

lt <- list(a = 1,

          let = letters[1:8],

          mt = matrix(1:6, nrow = 2))

  cat("List before modifying:\n")
```

```
print(lt)

  # Modifying element named 'a'

lt$a <- 5

  cat("List after modifying:\n")

print(lt)
```

Output:

```
List before modifying:
$a
[1] 1

$let
[1] "a" "b" "c" "d" "e" "f" "g" "h"

$mt
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

List after modifying:
$a
[1] 5

$let
[1] "a" "b" "c" "d" "e" "f" "g" "h"

$mt
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Deleting components from Named List

To delete elements from named list, we'll use **within()** function and the result will be assigned to the named list itself.

Example:

```
# Defining a named list
lt <- list(a = 1,
           let = letters[1:8],
           mt = matrix(1:6, nrow = 2))
  cat("List before deleting:\n")
print(lt)
  # Modifying element named 'a'
lt <- within(lt, rm(a))
  cat("List after deleting:\n")
print(lt)
```

Output:

```
List before deleting:
$a
[1] 1


$let
[1] "a" "b" "c" "d" "e" "f" "g" "h"


$mt
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6


List after deleting:
$let
[1] "a" "b" "c" "d" "e" "f" "g" "h"


$mt
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Types of Data items:

Generally, while doing programming in any programming language, you need to use various variables to store various information. Variables are nothing but reserved memory locations to store values. This means that, when you create a variable you reserve some space in memory.

You may like to store information of various data types like character, wide character, integer, floating point, double floating point, Boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

In contrast to other programming languages like C and java in R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. There are many types of R-objects. The frequently used ones are −

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

The simplest of these objects is the **vector object** and there are six data types of these atomic vectors, also termed as six classes of vectors. The other R-Objects are built upon the atomic vectors.

Data Type	Example	Verify

Logical	TRUE, FALSE	`v <- TRUE`
		`print(class(v))`
		it produces the following result −
		`[1] "logical"`

Numeric	12.3, 5, 999	`v <- 23.5`
		`print(class(v))`
		it produces the following result −
		`[1] "numeric"`

Integer	2L, 34L, 0L	`v <- 2L`
		`print(class(v))`
		it produces the following result −
		`[1] "integer"`

Complex	3 + 2i	`v <- 2+5i`

		`print(class(v))`
		it produces the following result −
		`[1] "complex"`

Character	'a' , '"good", "TRUE", '23.4'	

Raw	"Hello" is stored as 48 65 6c 6c 6f	`v <- charToRaw("Hello")`
		`print(class(v))`
		it produces the following result −
		`[1] "raw"`

In R programming, the very basic data types are the R-objects called **vectors** which hold elements of different classes as shown above. Please note in R the number of classes is not confined to only the above six types. For example, we can use many atomic vectors and create an array whose class will become array.

Vectors

When you want to create vector with more than one element, you should use **c()** function which means to combine the elements into a vector.

```
# Create a vector.
apple <- c('red','green',"yellow")
print(apple)

# Get the class of the vector.
print(class(apple))
```

When we execute the above code, it produces the following result −

```
[1] "red"    "green"  "yellow"
[1] "character"
```

Lists

A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

```
# Create a list.
list1 <- list(c(2,5,3),21.3,sin)

# Print the list.
print(list1)
```

When we execute the above code, it produces the following result −

```
[[1]]
[1] 2 5 3

[[2]]
[1] 21.3
```

```
[[3]]
function (x)  .Primitive("sin")
```

Matrices

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```
# Create a matrix.
M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow
= TRUE)
print(M)
```

When we execute the above code, it produces the following result −

```
     [,1] [,2] [,3]
[1,] "a"  "a"  "b"
[2,] "c"  "b"  "a"
```

Arrays

While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension. In the below example we create an array with two elements which are 3x3 matrices each.

```
# Create an array.
a <- array(c('green','yellow'),dim = c(3,3,2))
print(a)
```

When we execute the above code, it produces the following result −

```
, , 1

     [,1]     [,2]     [,3]
[1,] "green"  "yellow" "green"
[2,] "yellow" "green"  "yellow"
[3,] "green"  "yellow" "green"

2

     [,1]     [,2]     [,3]
[1,] "yellow" "green"  "yellow"
[2,] "green"  "yellow" "green"
[3,] "yellow" "green"  "yellow"
```

Factors

Factors are the r-objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. They are useful in statistical modeling.

Factors are created using the **factor()** function. The **nlevels** functions gives the count of levels.

```
# Create a vector.
apple_colors <-
c('green','green','yellow','red','red','red','green')

# Create a factor object.
factor_apple <- factor(apple_colors)

# Print the factor.
print(factor_apple)
print(nlevels(factor_apple))
```

When we execute the above code, it produces the following result −

```
[1] green  green  yellow red    red    red    green
Levels: green red yellow
[1] 3
```

Data Frames

Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.

Data Frames are created using the **data.frame()** function.

```
# Create the data frame.
BMI <-        data.frame(
   gender = c("Male", "Male","Female"),
   height = c(152, 171.5, 165),
   weight = c(81,93, 78),
   Age = c(42,38,26)
)
print(BMI)
```

When we execute the above code, it produces the following result −

```
  gender height weight Age
1   Male  152.0     81  42
2   Male  171.5     93  38
3 Female  165.0     78  26
```

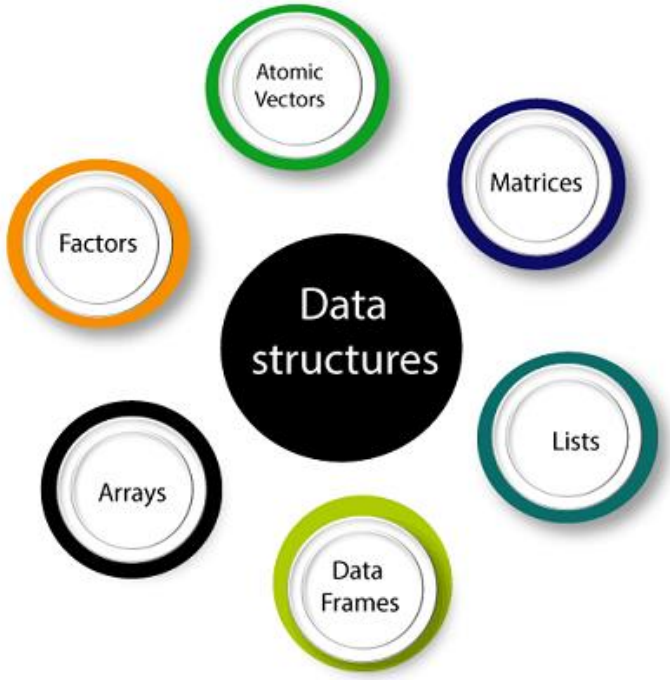
The structure of data items

Data Structures in R Programming

Data structures are very important to understand. Data structure are the objects which we will manipulate in our day-to-day basis in R. Dealing with object

conversions is the most common sources of despairs for beginners. We can say that everything in R is an object.

R has many data structures, which include:



1. Atomic vector
2. List
3. Array
4. Matrices
5. Data Frame
6. Factors

Vectors

A vector is the basic data structure in R, or we can say vectors are the most basic R data objects. There are six types of atomic vectors such as logical, integer, character, double, and raw. **"A vector is a collection of elements which is most commonly of mode character, integer, logical or numeric"** A vector can be one of the following two types:

1. Atomic vector
2. Lists

List

In R, **the list** is the container. Unlike an atomic vector, the list is not restricted to be a single mode. A list contains a mixture of data types. The list is also known as generic vectors because the element of the list can be of any type of R object. **"A list is a special type of vector in which each element can be a different type."**

We can create a list with the help of list() or as.list(). We can use vector() to create a required length empty list.

Arrays

There is another type of data objects which can store data in more than two dimensions known as arrays. **"An array is a collection of a similar data type with contiguous memory allocation."** Suppose, if we create an array of dimension (2, 3, 4) then it creates four rectangular matrices of two rows and three columns.

In R, an array is created with the help of **array()** function. This function takes a vector as an input and uses the value in the dim parameter to create an array.

Matrices

A matrix is an R object in which the elements are arranged in a two-dimensional rectangular layout. In the matrix, elements of the same atomic types are contained. For mathematical calculation, this can use a matrix containing the numeric element. A matrix is created with the help of the matrix() function in R.

Syntax

The basic syntax of creating a matrix is as follows:

1. matrix(data, no_row, no_col, by_row, dim_name)

Data Frames

A **data frame** is a two-dimensional array-like structure, or we can say it is a table in which each column contains the value of one variable, and row contains the set of value from each column.

There are the following characteristics of a data frame:

1. The column name will be non-empty.
2. The row names will be unique.
3. A data frame stored numeric, factor or character type data.

4. Each column will contain same number of data items.

Factors

Factors are also data objects that are used to categorize the data and store it as levels. Factors can store both strings and integers. Columns have a limited number of unique values so that factors are very useful in columns. It is very useful in data analysis for statistical modeling.

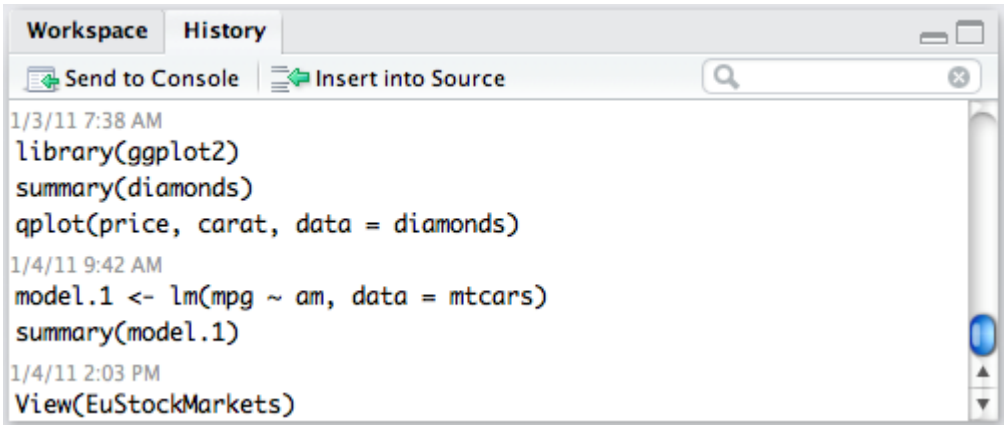
Factors are created with the help of **factor()** function by taking a vector as an input parameter.

Using Command History

The RStudio IDE maintains a database of all commands which you have ever entered into the Console. You can browse and search this database using the History pane.

Browsing History

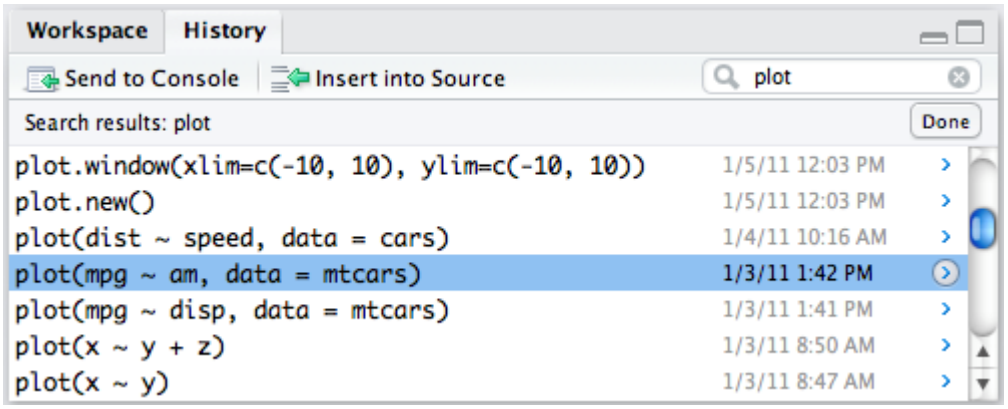
Commands you have previously entered in the RStudio console can be browsed from the History tab. The commands are displayed in order (most recent at the bottom) and grouped by block of time:



Searching History

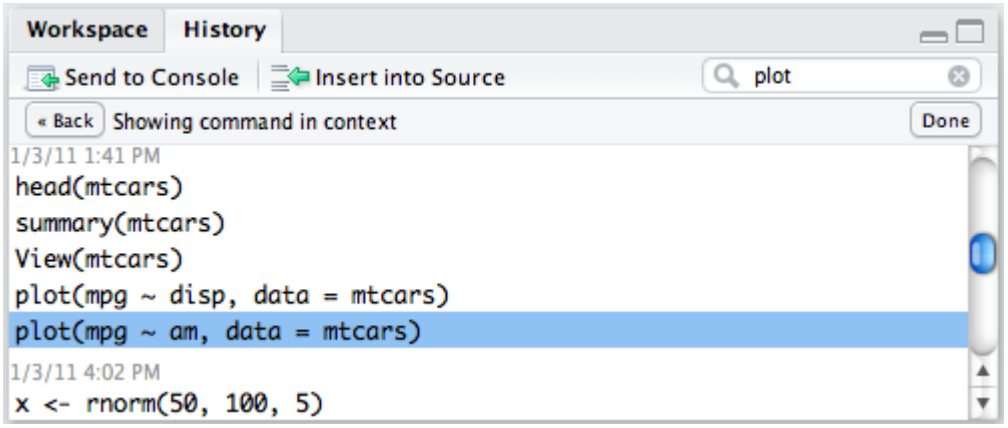
Executing a Search

You can use the search box at the top right of the history tab to search for all instances of a previous command (e.g. `plot`). The search can be further refined by adding additional words separated by spaces (e.g. the name of particular dataset):



Showing Command Context

After searching for a command within your history you may wish to view the other commands that were executed in proximity to it. By clicking the arrow in the right margin of the search results you can view the command within its context:

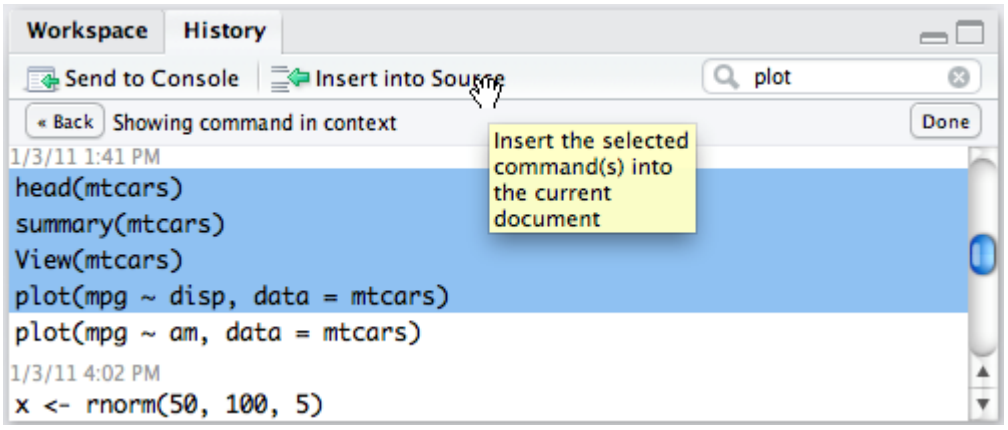


Using Commands

Commands selected within the History pane can be used in two fashions (corresponding to the two buttons on the left side of the History toolbar):

- **Send to Console**— Sends the selected command(s) to the Console. Note that the commands are inserted into the Console however they are not executed until you press **Enter**.
- **Insert into Source**— Inserts the selected command(s) into the currently active Source document. If there isn't currently a Source document available then a new untitled one will be created.

Within the history list you can select a single command or multiple commands:



Saving an R data file

As you work with your data in R you will eventually want to save it to disk. This will allow you to work with the data later and still retain the original dataset. It can also allow you to share your dataset with other analysts.

Before learning how to save a dataset in R, it is a good idea to create an example dataset. The following R script creates an R data frame [explained in another topic of this learning infrastructure] for you to practice saving.

```
x <- c(1:10) # create a numeric vector

y <- c(11:20) # create a numeric vector

z <- c(21:30) # create a numeric vector

m <- cbind(x, y, z) # create a matrix

d <- as.data.frame(m) # create a data frame

# create a text vector

t <- c("red", "blue", "red", "white", "blue", "white",
"red","blue", "white", "white")

df <- cbind(d, t) # add the text vector to the data
frame
```

Your R session now has a data frame object named **df** that you can use for the exercises below.

R dataset files

One of the simplest ways to save your data is by saving it into an RData file with the function **save( )**. R saves your data to the working folder on your computer disk in a binary file. This storage method is efficient and the only

drawback is that, because it is stored in an R binary format, you can only open it in R [there are some exceptions that will not be discussed here].

You can save the data frame **df** [from the above example] using this command:

```
save(df, file = "df.RData")
```

While the **save( )** command can have several arguments, this example uses only two. The first argument is the name of your R data object, **df** in this example. The second argument assigns a name to the RData file, **df.RData** in this example. You can use any text as your file name as long as it does not contain any embedded spaces. While you do not have to use the **.RData** extension, this is a recommended practice because the **.RData** extension will help RStudio to identify your R datasets. Notice that the file name is enclosed in quotation marks.

Try to save your data frame using the **save( )** command. Another topic in this learning infrastructure addressed how to load a R dataset into R so that will not be covered here.

Text files

There are other options for saving your data from your R session. You can save your data as text file. One advantage of saving your data into a text file is that you can open it in another application, such as a text editor or Excel, and work with it there.

The simplest way to save your data into a text file is by using the **write.csv( )** command. You may recall from the learning infrastructure topic about reading data files that a csv file is a text file that uses commas to separate each item of data form the other items of data. You can experiment saving the data frame **df** using the command:

```
write.csv(df, file = "df.csv")
```

While the **write.csv( )** command can have several arguments, this example uses only two. The first argument is the name of your R data object, **df** in this example. The second argument assigns a name to the csv file, **df.csv** in this example. You can use any text as your file name as long as it does not contain any embedded spaces. While you do not have to use

the .csv extension, this is a recommended practice. Notice that the file name is enclosed in quotation marks.

If you open **df.csv** in a text editor, you will see

"","x","y","z","t"

"1",1,11,21,"red"

"2",2,12,22,"blue"

"3",3,13,23,"red"

"4",4,14,24,"white"

"5",5,15,25,"blue"

"6",6,16,26,"white"

"7",7,17,27,"red"

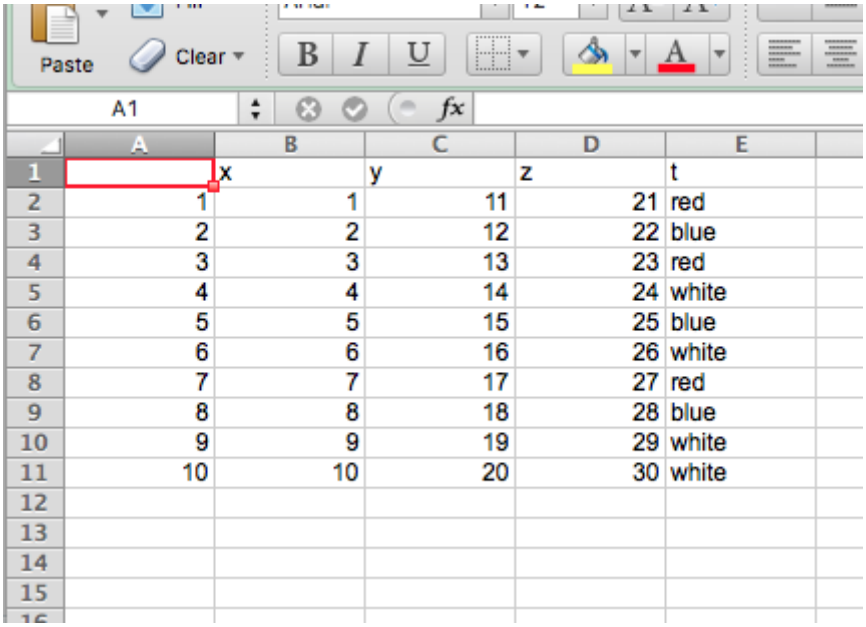
"8",8,18,28,"blue"

"9",9,19,29,"white"

"10",10,20,30,"white"

Notice that each item of data is separated from the other items of data with a comma and the header row of column titles is included. Another thing you may notice are the numbers enclosed in quotes in front of every line. This will be discussed below.

If you open **df.csv** in Excel, you will see



	A	B	C	D	E
1		x	y	z	t
2	1	1	11	21	red
3	2	2	12	22	blue
4	3	3	13	23	red
5	4	4	14	24	white
6	5	5	15	25	blue
7	6	6	16	26	white
8	7	7	17	27	red
9	8	8	18	28	blue
10	9	9	19	29	white
11	10	10	20	30	white
12					
13					
14					
15					

In both cases, your data is available for you to work with as text. The one issue is the fact that your export of **df** included the line numbers. This can be corrected by adding a third argument to your **write.csv( )** command. If you save your data object using this command

```
write.csv(df, file = "df2.csv", row.names = FALSE)
```

It will save **df** without the line numbers. Notice that the data object is saved as **df2.csv** this time. A different name was used so you can compare the two csv files later.

If you open **df2.csv** in a text editor, you will see

"x","y","z","t"

1,11,21,"red"

2,12,22,"blue"

3,13,23,"red"

4,14,24,"white"

5,15,25,"blue"

6,16,26,"white"

7,17,27,"red"

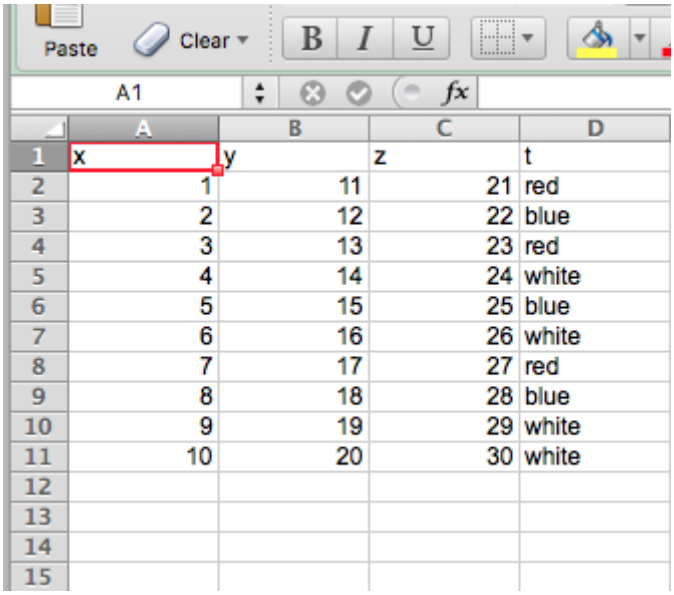
8,18,28,"blue"

9,19,29,"white"

10,20,30,"white"

The first column of line numbers is not in **df2.csv**. Everything else looks like **df.csv**.

If you open **df2.csv** in Excel, you will see



	A	B	C	D
1	x	y	z	t
2	1	11	21	red
3	2	12	22	blue
4	3	13	23	red
5	4	14	24	white
6	5	15	25	blue
7	6	16	26	white
8	7	17	27	red
9	8	18	28	blue
10	9	19	29	white
11	10	20	30	white
12				
13				
14				
15				

Again, this looks like the df2.csv Excel worksheet without the line numbers.

You can export your R data object using other R functions. One example of this is the function **write.table( )**. These functions will not be discussed here, but references to them are easily found on the Internet.

Working with Excel files in R

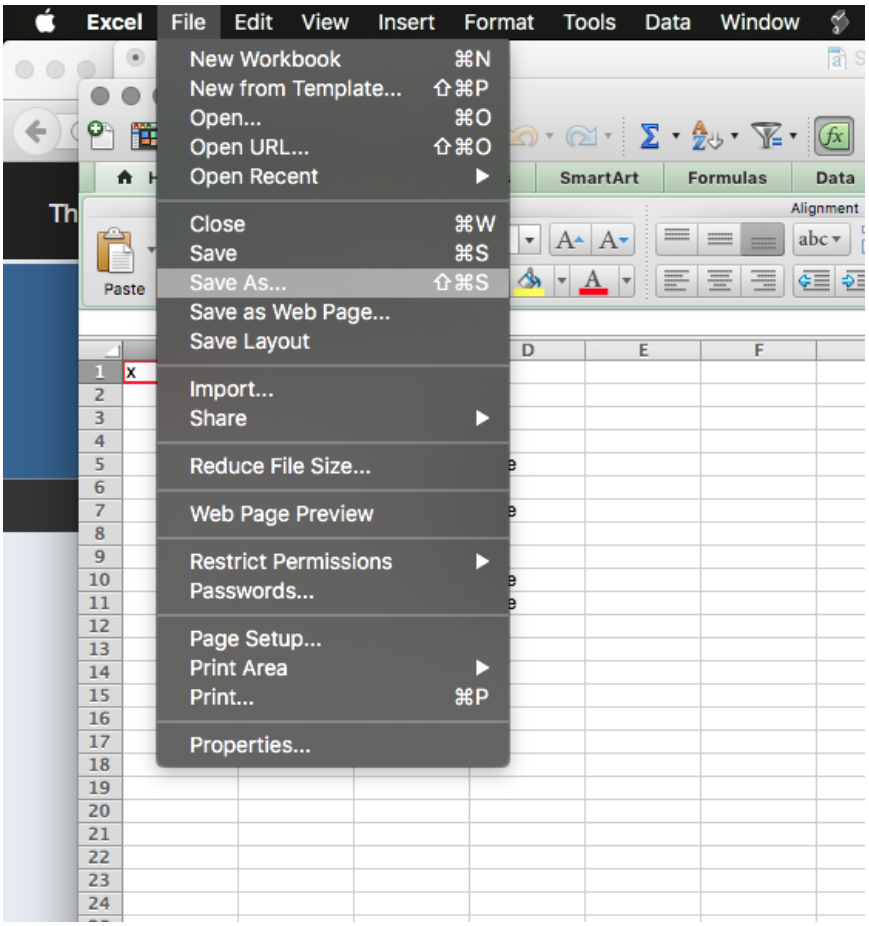
You can export your R data object as an Excel spreadsheet using functions in the **xlsx** R package. You will need to manually install this package because the RStudio package manager will not do it. To install the package, enter this command in the command console

```
install.packages("xlsx")
```

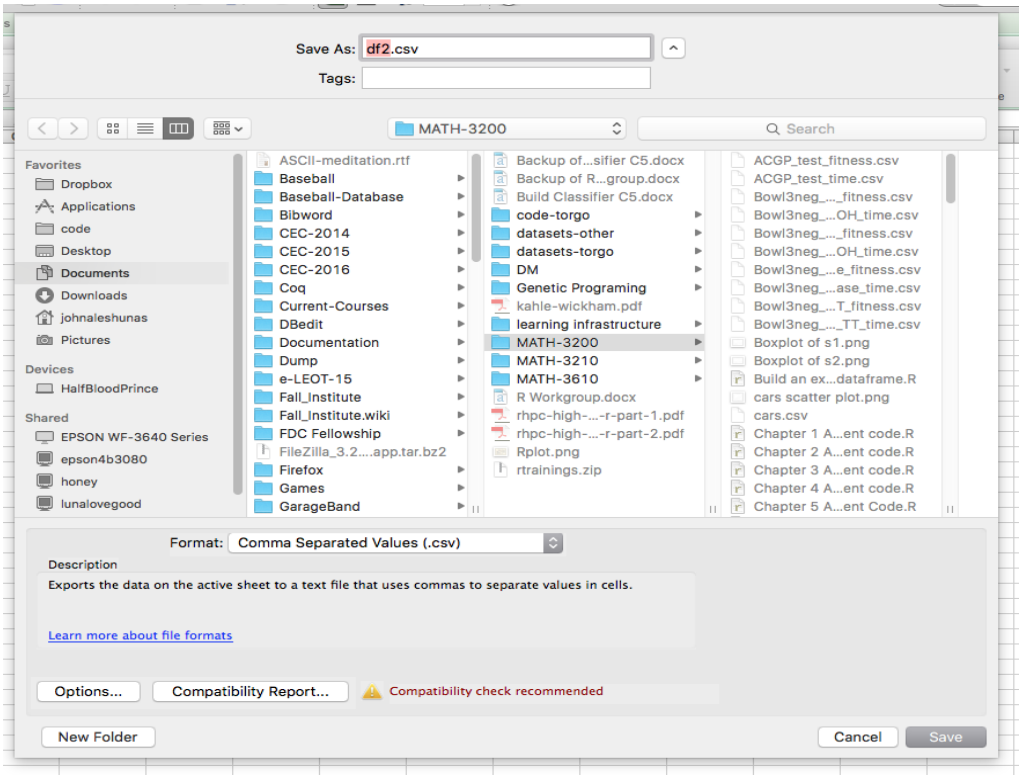
This will install the packages and its dependencies. You will find the package in the Packages panel of RStudio. Check the box next to the package to load it for use in your R session. This package will enable you to read and write directly into and out of Excel files from your R session. A good reference for this package can be found at

If you work with your text data file in Excel, you can export it as a csv file and easily import it into your R session as discussed in another learning infrastructure topic.

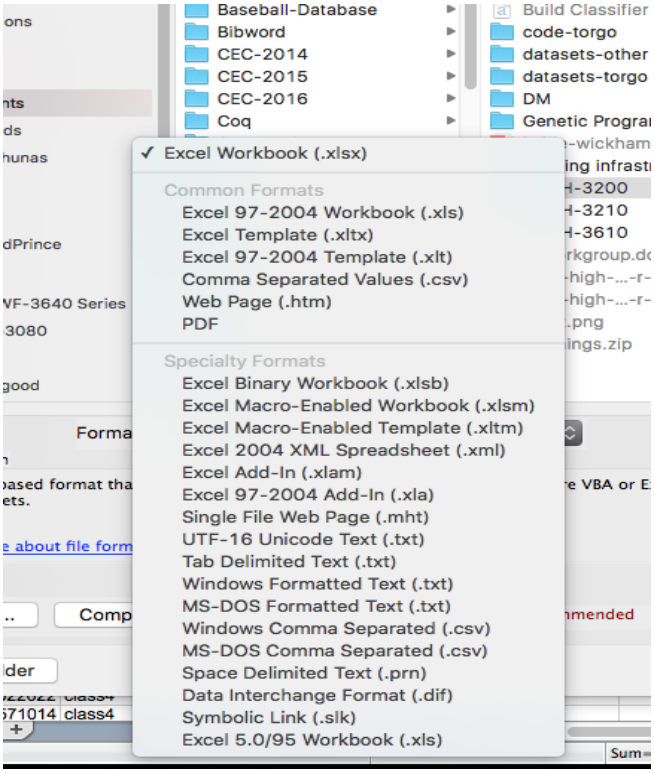
You can easily save an Excel worksheet as a csv file. In Excel, open the **File** menu and click **Save As**. [note: this example uses Mac Excel screen shots, Windows Excel will act similarly]



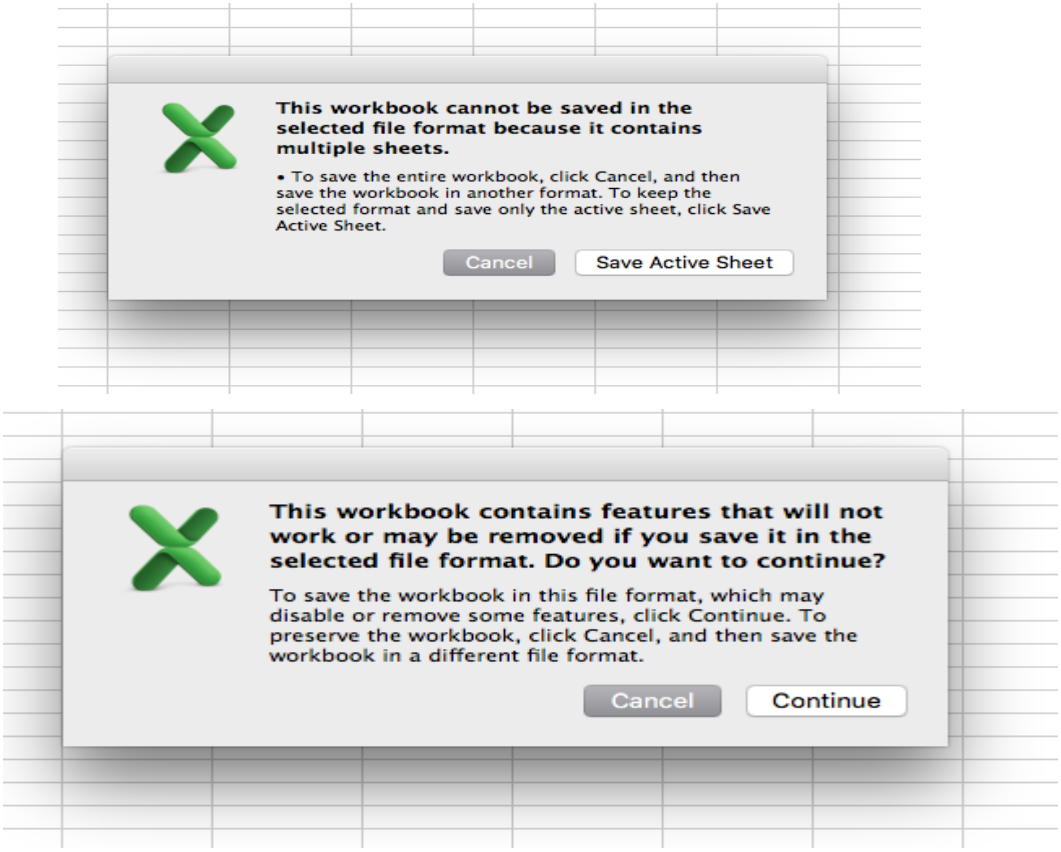
The File Save As dialog will open



Enter the name that you wish to use for your file in the file name box at the top of the dialog. Next, go to the File Format box below the folder directory and open the list. You can now choose the MS-DOS Comma Separated (.csv) format.



Click the **Save** button. If you are exporting an Excel spreadsheet, you will encounter two warning dialogs. They will look like this



In the first warning dialog, click **Save Active Sheet**. In the second warning dialog click **Continue**. Excel will now save your data into a csv file.

PROBABILITY DISTRIBUTIONS in R:

The binomial distribution model deals with finding the probability of success of an event which has only two possible outcomes in a series of experiments. For example, tossing of a coin always gives a head or a tail. The probability of finding exactly 3 heads in tossing a coin repeatedly for 10 times is estimated during the binomial distribution.

R has four in-built functions to generate binomial distribution. They are described below.

```
dbinom(x, size, prob)
pbinom(x, size, prob)
qbinom(p, size, prob)
rbinom(n, size, prob)
```

Following is the description of the parameters used −

- **x** is a vector of numbers.

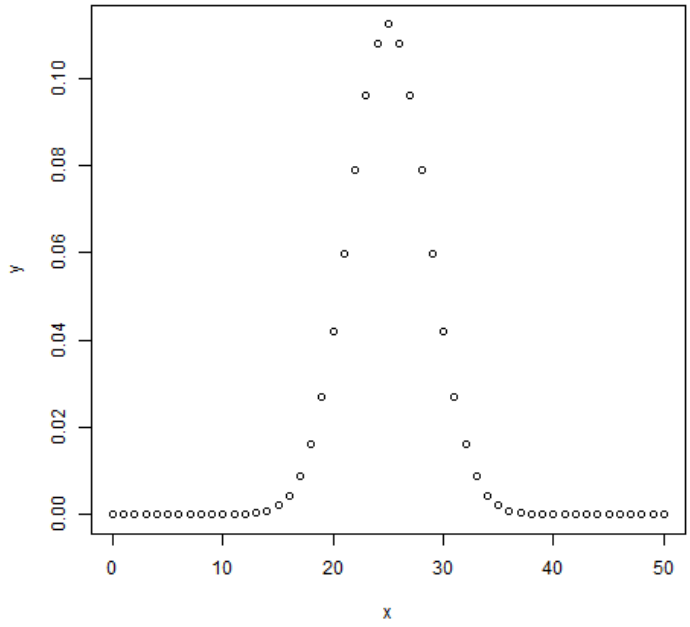
- **p** is a vector of probabilities.
- **n** is number of observations.
- **size** is the number of trials.
- **prob** is the probability of success of each trial.

dbinom()

This function gives the probability density distribution at each point.

```
# Create a sample of 50 numbers which are incremented by 1.
x <- seq(0,50,by = 1)

# Create the binomial distribution.
y <- dbinom(x,50,0.5)

# Give the chart file a name.
png(file = "dbinom.png")

# Plot the graph for this sample.
plot(x,y)

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −



pbinom()

This function gives the cumulative probability of an event. It is a single value representing the probability.

```
# Probability of getting 26 or less heads from a 51 tosses of a
coin.
x <- pbinom(26,51,0.5)

print(x)
```

When we execute the above code, it produces the following result −

```
[1] 0.610116
```

qbinom()

This function takes the probability value and gives a number whose cumulative value matches the probability value.

```
# How many heads will have a probability of 0.25 will come out
when a coin
# is tossed 51 times.
x <- qbinom(0.25,51,1/2)

print(x)
```

When we execute the above code, it produces the following result −

```
[1] 23
```

rbinom()

This function generates required number of random values of given probability from a given sample.

```
# Find 8 random values from a sample of 150 with probability of
0.4.
x <- rbinom(8,150,.4)

print(x)
```

When we execute the above code, it produces the following result −

```
[1] 58 61 59 66 55 60 61 67
```

Poisson Distribution

The **Poisson distribution** is the probability distribution of independent event occurrences in an interval. If *λ* is the mean occurrence per interval, then the probability of having *x* occurrences within a given interval is:

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!} \quad where\ x = 0, 1, 2, 3, ...$$

Problem

If there are twelve cars crossing a bridge per minute on average, find the probability of having seventeen or more cars crossing the bridge in a particular minute.

Solution

The probability of having *sixteen or less* cars crossing the bridge in a particular minute is given by the function ppois.

```
> ppois(16, lambda=12)   # lower tail
[1] 0.89871
```

Hence the probability of having seventeen or more cars crossing the bridge in a minute is in the *upper tail* of the probability density function.

```
> ppois(16, lambda=12, lower=FALSE)   # upper tail
[1] 0.10129
```

Answer

If there are twelve cars crossing a bridge per minute on average, the probability of having seventeen or more cars crossing the bridge in a particular minute is 10.1%.

R - Normal Distribution:

In a random collection of data from independent sources, it is generally observed that the distribution of data is normal. Which means, on plotting a graph with the value of the variable in the horizontal axis and the count of the values in the vertical axis we get a bell shape curve. The center of the curve represents the mean of the data set. In the graph, fifty percent of values lie to the left of the mean and the other fifty percent lie to the right of the graph. This is referred as normal distribution in statistics.

R has four in built functions to generate normal distribution. They are described below.

```
dnorm(x, mean, sd)
pnorm(x, mean, sd)
qnorm(p, mean, sd)
rnorm(n, mean, sd)
```

Following is the description of the parameters used in above functions −

- **x** is a vector of numbers.
- **p** is a vector of probabilities.
- **n** is number of observations(sample size).
- **mean** is the mean value of the sample data. It's default value is zero.
- **sd** is the standard deviation. It's default value is 1.

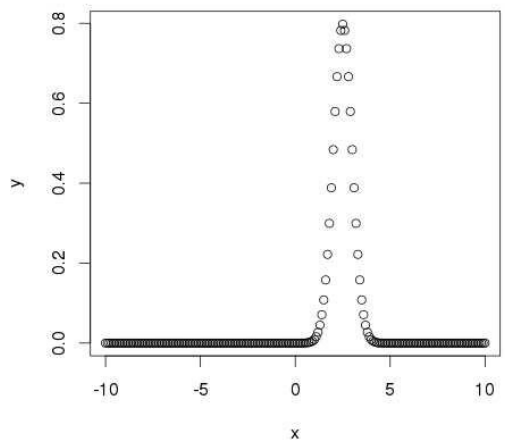
dnorm()

This function gives height of the probability distribution at each point for a given mean and standard deviation.

```
# Create a sequence of numbers between -10 and 10 incrementing by
0.1.
x <- seq(-10, 10, by = .1)

# Choose the mean as 2.5 and standard deviation as 0.5.
y <- dnorm(x, mean = 2.5, sd = 0.5)

# Give the chart file a name.
```

```
png(file = "dnorm.png")

plot(x,y)

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −

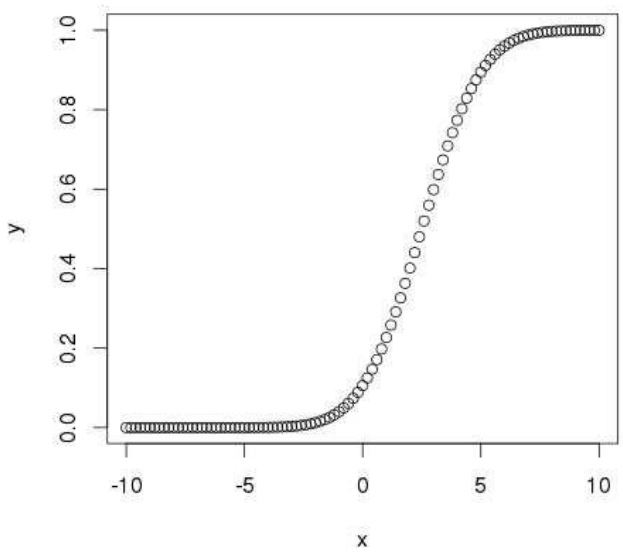


pnorm()

This function gives the probability of a normally distributed random number to be less that the value of a given number. It is also called "Cumulative Distribution Function".

```
# Create a sequence of numbers between -10 and 10 incrementing by
0.2.
x <- seq(-10,10,by = .2)

# Choose the mean as 2.5 and standard deviation as 2.
y <- pnorm(x, mean = 2.5, sd = 2)

# Give the chart file a name.
png(file = "pnorm.png")

# Plot the graph.
plot(x,y)

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −

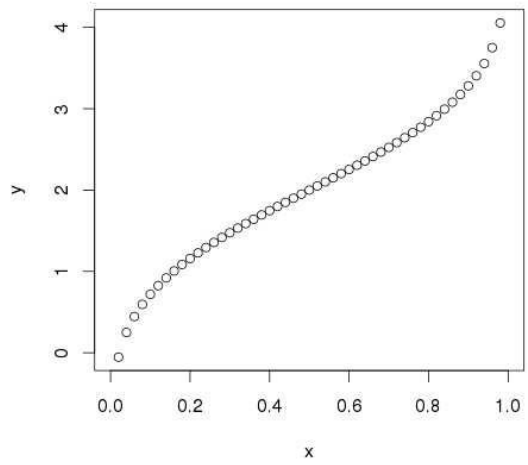


qnorm()

This function takes the probability value and gives a number whose cumulative value matches the probability value.

```
# Create a sequence of probability values incrementing by 0.02.
x <- seq(0, 1, by = 0.02)

# Choose the mean as 2 and standard deviation as 3.
y <- qnorm(x, mean = 2, sd = 1)

# Give the chart file a name.
png(file = "qnorm.png")

# Plot the graph.
plot(x,y)

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result −

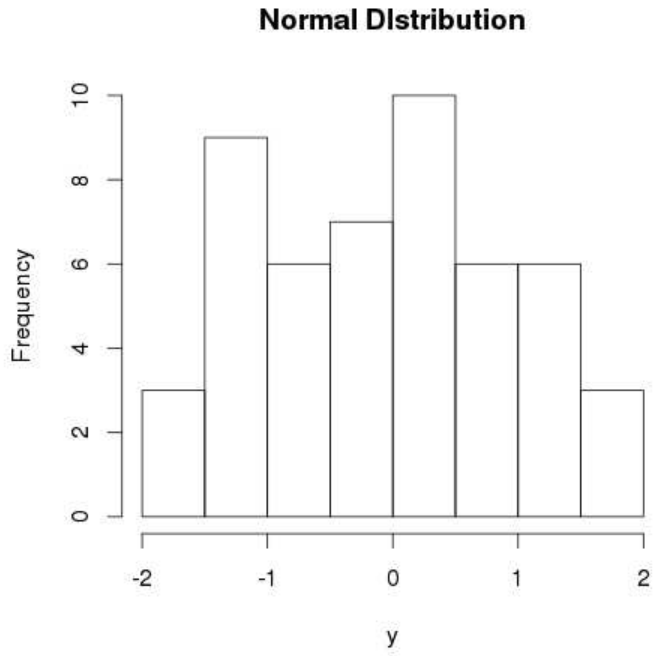


rnorm()

This function is used to generate random numbers whose distribution is normal. It takes the sample size as input and generates that many random numbers. We draw a histogram to show the distribution of the generated numbers.

```
# Create a sample of 50 numbers which are normally distributed.
y <- rnorm(50)

# Give the chart file a name.
png(file = "rnorm.png")

# Plot the histogram for this sample.
hist(y, main = "Normal DIstribution")

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result –

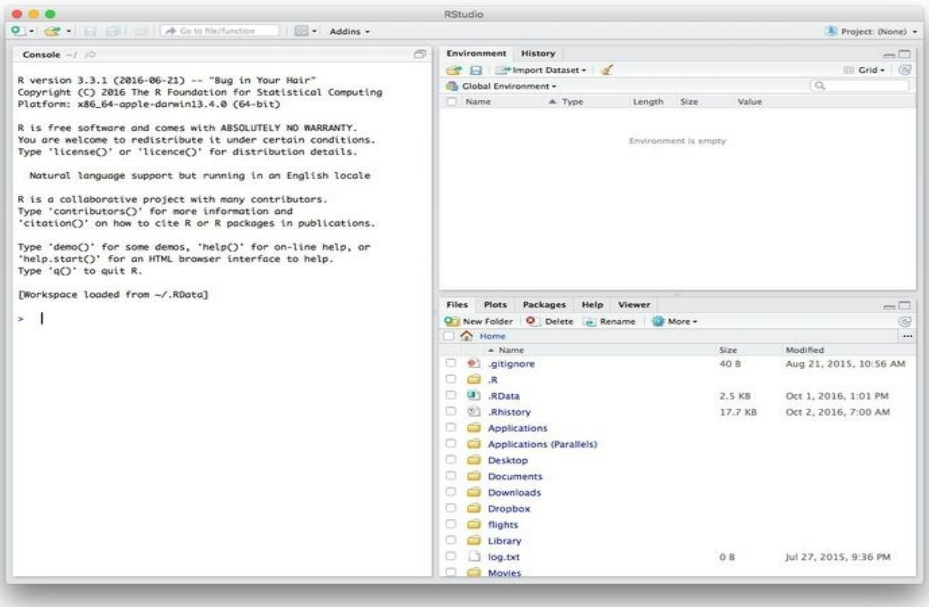


we will process data using **R**, which is a very powerful tool, designed by statisticians for data analysis. Described on its website as “free software environment for statistical computing and graphics,” R is a programming language that opens a world of possibilities for making graphics and analyzing and processing data. Indeed, just about anything you may want to do with data can be done with R, from web scraping to making interactive graphics.

Next week we will make static graphics with R. We will explore’s its potential for making interactive charts and maps in week 13, and use it to make animations in week 14. Our goal for this week’s class is to get used to working with data in R.

RStudio is an “integrated development environment,” or IDE, for R that provides a user-friendly interface.

Launch RStudio, and the screen should look like this:



The main panel to the left is the R Console. Type valid R code into here, hit `return`, and it will be run. See what happens if you run:

```
print("Hello World!")
```

The data we will use today

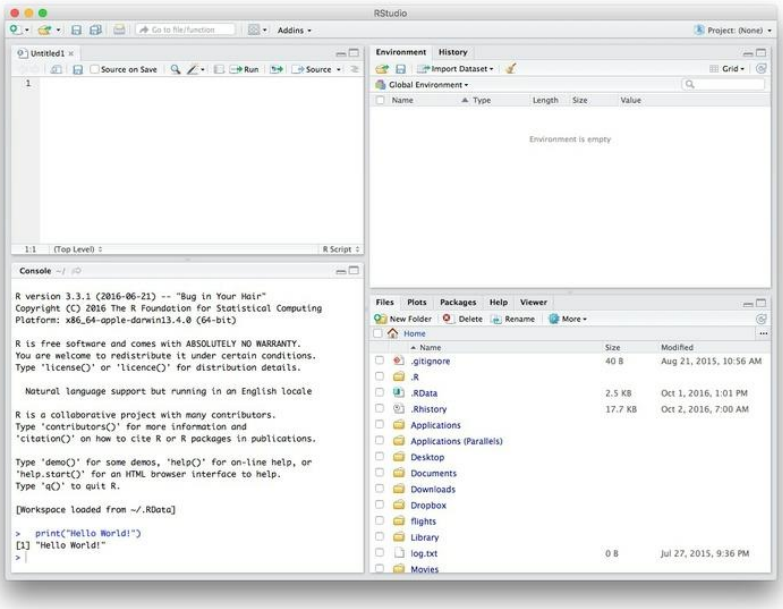
Download the data for this session from here, unzip the folder and place it on your desktop. It contains the following files, used in reporting this story, which revealed that some of the doctors paid as “experts” by the drug company Pfizer had troubling disciplinary records:

- `pfizer.csv` Payments made by Pfizer to doctors across the United States in the second half on 2009. Contains the following variables:
  - `org_indiv` Full name of the doctor, or their organization.
  - `first_plus` Doctor’s first and middle names.
  - `first_name last_name`. First and last names.
  - `city state` City and state.
  - `category of payment` Type of payment, which include `Expert-led Forums`, in which doctors lecture their peers on using Pfizer’s drugs, and `Professional Advising.
  - `cash` Value of payments made in cash.
  - `other` Value of payments made in-kind, for example puschase of meals.
  - `total` value of payment, whether cash or in-kind.
- `fda.csv` Data on warning letters sent to doctors by the U.S. Food and Drug Administration, because of problems in the way in which they ran clinical trials testing experimental treatments. Contains the following variables:
  - `name_last name_first name_middle` Doctor’s last, first, and middle names.

- `issued` Date letter was sent.
- `office` Office within the FDA that sent the letter.

Reproducibility: Save your scripts

Data journalism should ideally be fully documented and reproducible. R makes this easy, as every operation performed can be saved in a script, and repeated by running that script. Click on the  icon at top left and select `R Script`. A new panel should now open:



Any code we type in here can be run in the console. Hitting `Run` will run the line of code on which the cursor is sitting. To run multiple lines of code, highlight them and click `Run`. Click on the save/disk icon in the script panel and save the blank script to the file on your desktop with the data for this week, calling it `week7.R`.

Set your working directory

Now we can set the working directory to this folder by selecting from the top menu `Session>Set Working Directory>To Source File Location`. (Doing so means we can load the files in this directory without having to refer to the full path for their location, and anything we save will be written to this folder.)

Notice how this code appears in the console:

```
setwd("~/Desktop/week7")
```

Save your data

The panel at top right has two tabs, the first showing the `Environment`, or all of the "objects" loaded into memory for this R session. We can save this as well, so we don't have to load and process data again if we return to return to a project later.
(The second tab shows the `History` of the operations you have performed in RStudio.)
Click on the save/disk icon in the `Environment` panel to save and call the file `week7.RData`.
You should see the following code appear in the Console:

```
save.image("~/Desktop/week7/week7.RData")
```

Copy this code into your script, placing it at the end, with a comment, explaining what it does:

```
# save session data
save.image("~/Desktop/week7/week7.RData")
```

Comment your code

Anything that appears on a line after `#` will be treated as a comment, and will be ignored when the code is run. Get into the habit of commenting your code: Don't trust yourself to remember what it does!

Some R code basics

- `<-` is known as an "assignment operator." It means: "Make the object named to the left equal to the output of the code to the right."
- `&` means AND, in Boolean logic, which we discussed in week 5 when working with web search forms.
- `|` means OR, in Boolean logic.
- `!` means NOT, in Boolean logic.
- When referring to values entered as text, or to dates, put them in quote marks, like this: `"United States"`, or `"2016-07-26"`. Numbers are not quoted.
- When entering two or more values as a list, combine them using the function `c`, with the values separated by commas, for example: `c("2016-07-26","2016-08-04")`
- As in a spreadsheet, you can specify a range of values with a colon, for example: `c(1:10)` creates a list of integers (whole numbers) from one to ten.

- Some common operators:

    - `+ -` add, subtract.
    - `* /` multiply, divide.
    - `> <` greater than, less than.
    - `>= <=` greater than or equal to, less than or equal to.
    - `!=` not equal to.

- Equals signs can be a little confusing, but see how they are used in the code we use today:
    - `==` test whether an object is equal to a value. This is often used when filtering data, as we will see.
    - `=` make an object equal to a value; works like `<-`, but used within the brackets of a function.

We encountered **functions** in week 1 in the context of spreadsheet formulas. They are followed by brackets, and act on the code in the brackets.

Important: Object and variable names in R should not contain spaces.

Install and load R packages

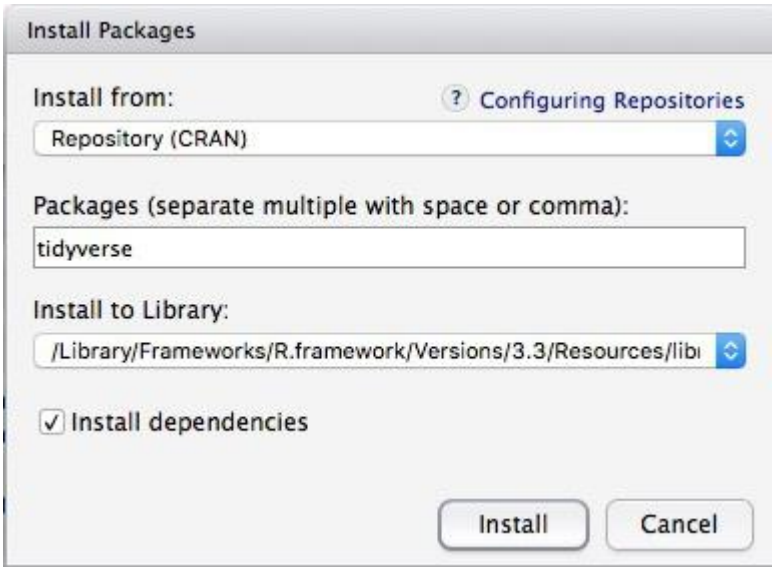
Much of the power of R comes from the thousands of "packages" written by its community of open source contributors. These are optimized for specific statistical, graphical or data-processing tasks. To see what packages are available in the basic distribution of R, select the `Packages` tab in the panel at bottom right. To find packages for particular tasks, try searching Google using appropriate keywords and the phrase "R package."

In this class, we will work with two incredibly useful packages developed by Hadley Wickham, chief scientist at RStudio:

- **readr** For reading and writes CSV and other text files.
- **dplyr** For processing and manipulating data.

These and several other useful packages have been combined into a super-package called **tidyverse**.

To install a package, click on the `Install` icon in the `Packages` tab, type its name into the dialog box, and make sure that `Install dependencies` is checked, as some packages will only run correctly if other packages are also installed. Click `Install` and all of the required packages should install:



Notice that the following code appears in the console:

```
install.packages("tidyverse")
```

So you can also install packages with cod in this format, without using the point-and-click interface.

Each time you start R, it's a good idea to click on `Update` in the `Packages` panel to update all your installed packages to the latest versions.
Installing a package makes it available to you, but to use it in any R session you need to load it. You can do this by checking its box in the `Packages` tab. However, we will enter the following code into our script, then highlight these lines of code and run them:

```
# load packages to read, write and manipulate data
library(readr)
library(dplyr)
```

At this point, and at regular intervals, save your script, by clicking the save/disk icon in the script panel, or using the `⌘-S` keyboard shortcut.

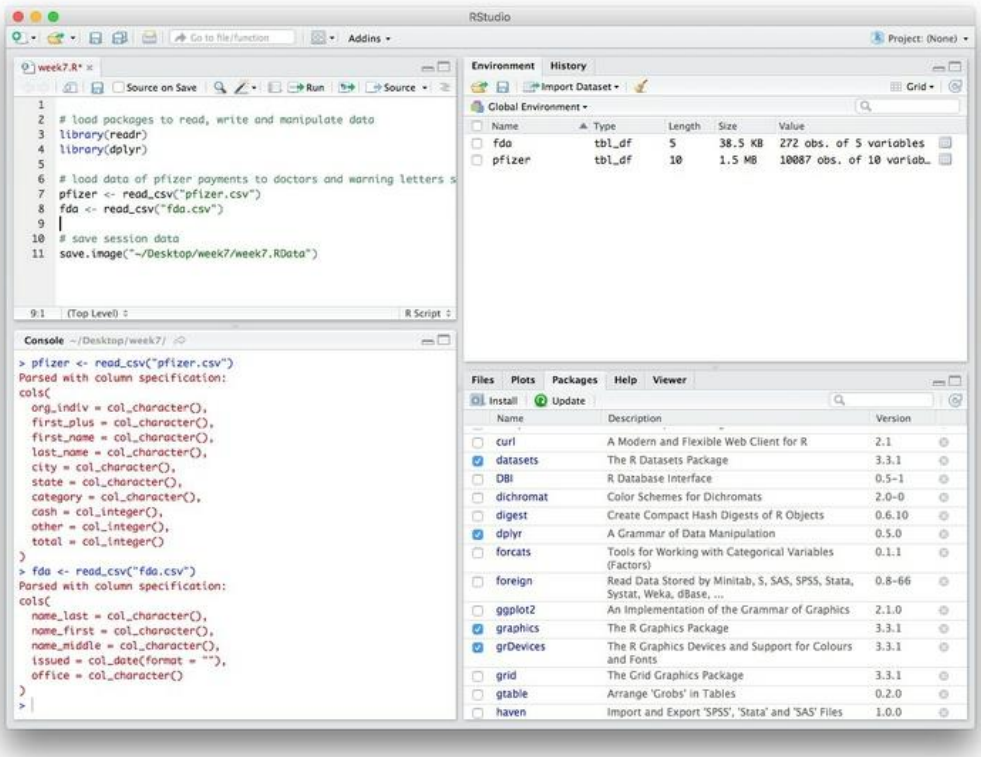
Load and view data

Load data

You can load data into the current R session by selecting `Import Dataset>From Text File...` in the `Environment` tab.
However, we will use the `read_csv` function from the **readr** package. Copy the following code into your script and `Run`:

```
# load data of pfizer payments to doctors and warning letters sent by food and drug adminstration
pfizer <- read_csv("pfizer.csv")
fda <- read_csv("fda.csv")
```

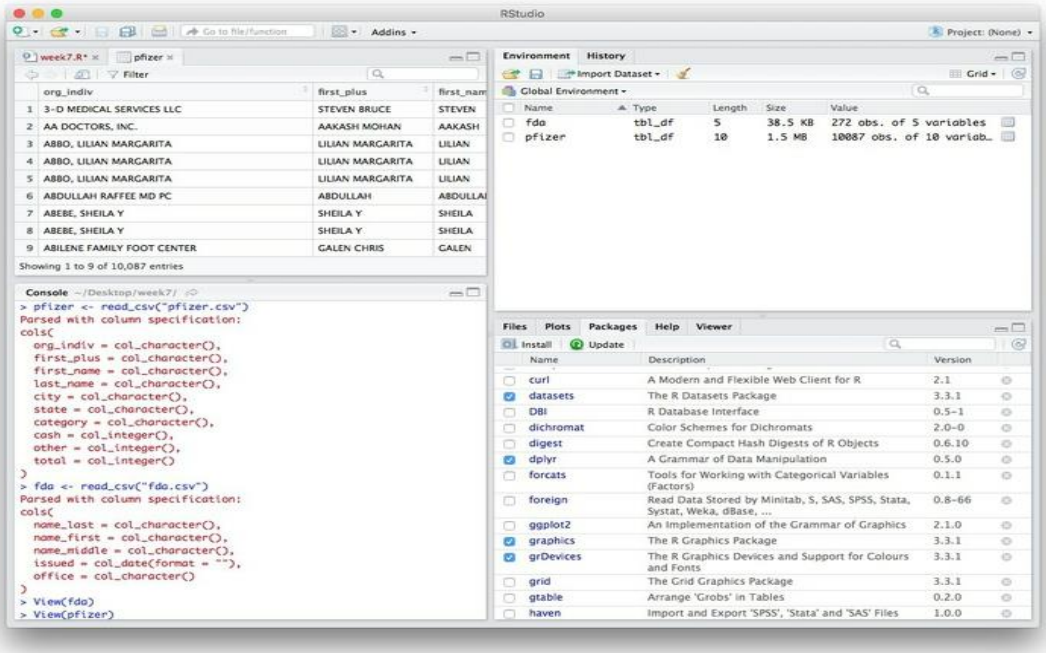
Notice that the `Environment` now contains two objects, of the type `tbl_df`, a variety of the standard R object for holding tables of data, known as a **data frame**:



The `Value` for each data frame details the number of columns, and the number of rows, or observations, in the data.
You can remove any object from your environment by checking it in the `Grid` view and clicking the broom icon.

Examine the data

We can `View` data at any time by clicking on its table icon in the `Environment` tab in the `Grid` view.
Here, for example, I am looking at the `pfizer` view:



The `str` function will tell you more about the columns in your data, including their data type. Copy this code into your script and `Run`:

```
# view structure of data
str(pfizer)
```

This should give the following output in the R Console:

```
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':    10087 obs. of  10 variables:
 $ org_indiv : chr  "3-D MEDICAL SERVICES LLC" "AA DOCTORS, INC." "ABBO,
LILIAN MARGARITA" "ABBO, LILIAN MARGARITA" ...
 $ first_plus: chr  "STEVEN BRUCE" "AAKASH MOHAN" "LILIAN MARGARITA" "LILIAN
MARGARITA" ...
 $ first_name: chr  "STEVEN" "AAKASH" "LILIAN" "LILIAN" ...
 $ last_name : chr  "DEITELZWEIG" "AHUJA" "ABBO" "ABBO" ...
 $ city      : chr  "NEW ORLEANS" "PASO ROBLES" "MIAMI" "MIAMI" ...
 $ state     : chr  "LA" "CA" "FL" "FL" ...
 $ category  : chr  "Professional Advising" "Expert-Led Forums" "Business
Related Travel" "Meals" ...
 $ cash      : int  2625 1000 0 0 1800 750 0 825 3000 0 ...
 $ other     : int  0 0 448 119 0 0 47 0 0 396 ...
 $ total     : int  2625 1000 448 119 1800 750 47 825 3000 396 ...
```

`chr` means “character,” or a string of text (which can be treated as a categorical variable); `int` means an integer, or whole number.
Also examine the structure of the `fda` data frame using the following code:

```
str(fda)
```

This should be the console output:

```
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':    272 obs. of  5 variables:
```

```
 $ name_last  : chr  "ADELGLASS" "ADKINSON" "ALLEN" "AMSTERDAM" ...
 $ name_first : chr  "JEFFREY" "N." "MARK" "DANIEL" ...
 $ name_middle: chr  "M." "FRANKLIN" "S." NA ...
 $ issued     : Date, format: "1999-05-25" ...
 $ office     : chr  "Center for Drug Evaluation and Research" "Center for
Biologics Evaluation and Research" "Center for Devices and Radiological
Health" "Center for Biologics Evaluation and Research" ...
```

Notice that `issued` has been recognized as a `Date` variable. Other common data types include `num`, for numbers that may contain decimals and `POSIXct` for full date and time.

If you run into any trouble importing data with **readr**, you may need to specify the data types for some columns — in particular for date and time. This link explains how to set data types for individual variables when importing data with **readr**.

To specify an individual column use the name of the data frame and the column name, separated by `$`. Type this into your script and run:

```
# print values for total in pfizer data
pfizer$total
```

The output will be the first 10,000 values for that column.

If you need to change the data type for any column, use the following functions:

- `as.character` converts to a text string.
- `as.numeric` converts to a number.
- `as.factor` converts to a categorical variable.
- `as.integer` converts to an integer
- `as.Date` converts to a date
- `as.POSIXct` convets to a full date and time.

(Conversions to full dates and times can get complicated, because of timezones. Contact me for advice if you need to work with full dates and times for your project!)

Now add the following code to your script to convert the convert `total` in the `pfizer` data to a numeric variable (which would allow it to hold decimal values, if we had any).

```
# convert total to numeric variable
pfizer$total <- as.numeric(pfizer$total)
str(pfizer)
```

Notice that the data type for `total` has now changed:

```
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':    10087 obs. of  10 variables:
 $ org_indiv : chr  "3-D MEDICAL SERVICES LLC" "AA DOCTORS, INC." "ABBO,
LILIAN MARGARITA" "ABBO, LILIAN MARGARITA" ...
 $ first_plus: chr  "STEVEN BRUCE" "AAKASH MOHAN" "LILIAN MARGARITA" "LILIAN
MARGARITA" ...
 $ first_name: chr  "STEVEN" "AAKASH" "LILIAN" "LILIAN" ...
 $ last_name : chr  "DEITELZWEIG" "AHUJA" "ABBO" "ABBO" ...
 $ city      : chr  "NEW ORLEANS" "PASO ROBLES" "MIAMI" "MIAMI" ...
 $ state     : chr  "LA" "CA" "FL" "FL" ...
```

```
 $ category  : chr  "Professional Advising" "Expert-Led Forums" "Business
Related Travel" "Meals" ...
 $ cash      : int  2625 1000 0 0 1800 750 0 825 3000 0 ...
 $ other     : int  0 0 448 119 0 0 47 0 0 396 ...
 $ total     : num  2625 1000 448 119 1800 ...
```

The `summary` function will run a quick statistical summary of a data frame, calculating mean, median and quartile values for continuous variables:

```
# summary of pfizer data
summary(pfizer)
```

Here is the last part of the console output:

```
     total
 Min.   :      0
 1st Qu.:    191
 Median :    750
 Mean   :   3507
 3rd Qu.:   2000
 Max.   :1185466
```

Manipulate and analyze data

Now we will use **dplyr** to manipulate the data, using the basic operations we discussed in week 1:

- **Sort:** Largest to smallest, oldest to newest, alphabetical etc.
- **Filter:** Select a defined subset of the data.
- **Summarize/Aggregate:** Deriving one value from a series of other values to produce a summary statistic. Examples include: count, sum, mean, median, maximum, minimum etc. Often you'll **group** data into categories first, and then aggregate by group.
- **Join:** Merging entries from two or more datasets based on common field(s), e.g. unique ID number, last name and first name.

Here are some of the most useful functions in **dplyr**:

- `select` Choose which columns to include.
- `filter` **Filter** the data.
- `arrange` **Sort** the data, by size for continuous variables, by date, or alphabetically.
- `group_by` **Group** the data by a categorical variable.
- `summarize` **Summarize**, or aggregate (for each group if following `group_by`). Often used in conjunction with functions including:
    - `mean` Calculate the mean, or average.

- `median` Calculate the median.
- `max` Find the maximum value.
- `min` Find the minimum value
- `sum` Add all the values together.
- `n` Count the number of records.

- `mutate` Create new column(s) in the data, or change existing column(s).
- `rename` Rename column(s).
- `bind_rows` Merge two data frames into one, combining data from columns with the same name.

There are also various functions to **join** data, which we will explore below.

These functions can be chained together using the operator `%>%` which makes the output of one line of code the input for the next. This allows you to run through a series of operations in logical order. I find it helpful to think of `%>%` as "then."

Filter and sort data

Now we will **filter** and **sort** the data in specific ways. For each of the following examples, copy the code that follows into your script, and view the results. Notice how we create a new objects to hold the processed data.

Find doctors in California paid $10,000 or more by Pfizer to run "Expert-Led Forums."

```
# doctors in California who were paid $10,000 or more by Pfizer to run
"Expert-Led Forums."
ca_expert_10000 <- pfizer %>%
  filter(state == "CA" & total >= 10000 & category == "Expert-Led Forums")
```

Notice the use of `==` to find values that match the specified text, `>=` for greater than or equal to, and the Boolean operator `&`.

Now add a **sort** to the end of the code to list the doctors in descending order by the payments received:

```
# doctors in California who were paid $10,000 or more by Pfizer to run
"Expert-Led Forums."
ca_expert_10000 <- pfizer %>%
  filter(state == "CA" & total >= 10000 & category == "Expert-Led Forums") %>%
  arrange(desc(total))
```

If you `arrange` without the `desc` function, the **sort** will be from smallest to largest.

Find doctors in California *or* New York who were paid $10,000 or more by Pfizer to run "Expert-Led Forums."

```
ca_ny_expert_10000 <- pfizer %>%
  filter((state == "CA" | state == "NY") & total >= 10000 & category ==
"Expert-Led Forums") %>%
```

```
  arrange(desc(total))
```

Notice the use of the | Boolean operator, and the brackets around that part of the query. This ensures that this part of the query is run first. See what happens if you exclude them.

Find doctors in states *other than* California who were paid $10,000 or more by Pfizer to run "Expert-Led Forums."

```
not_ca_expert_10000 <- pfizer %>%
  filter(state != "CA" & total >= 10000 & category=="Expert-Led Forums")) %>%
  arrange(desc(total))
```

Notice the use of the != operator to exclude doctors in California.

Find the 20 doctors across the four largest states (CA, TX, FL, NY) who were paid the most for professional advice.

```
ca_ny_tx_fl_prof_top20 <- pfizer %>%
  filter((state=="CA" | state == "NY" | state == "TX" | state == "FL") &
category == "Professional Advising") %>%
  arrange(desc(total)) %>%
  head(20)
```

Notice the use of `head`, which grabs a defined number of rows from the start of a data frame. Here, it is crucial to run the sort first! See what happens if you change the order of the last two lines.

Filter the data for all payments for running Expert-Led Forums or for Professional Advising, and arrange alphabetically by doctor (last name, then first name)

```
# Filter the data for all payments for running Expert-Led Forums or for
Professional Advising, and arrange alphabetically by doctor (last name, then
first name)
expert_advice <- pfizer %>%
  filter(category == "Expert-Led Forums" | category == "Professional
Advising") %>%
  arrange(last_name, first_name)
```

Notice that you can **sort** by multiple variables, separated by commas.

Use pattern matching to filter text

The following code uses the `grepl` function to find values containing a particular string of text. This can simplify the code used to **filter** based on text.

```
# use pattern matching to filter text
expert_advice <- pfizer %>%
  filter(grepl("Expert|Professional", category)) %>%
  arrange(last_name, first_name)

not_expert_advice <- pfizer %>%
  filter(!grepl("Expert|Professional", category)) %>%
```

```
  arrange(last_name, first_name)
```

This code differs only by the `!` Boolean operator. Notice that it has split the data into two, based on categories of payment.

Append one data frame to another

The following code uses the `bind_rows` function to append one data frame to another, here recreating the unfiltered data from the two data frames above.

```
# merge/append data frames
pfizer2 <- bind_rows(expert_advice, not_expert_advice)
```

Write data to a CSV file

readr can write data to CSV and other text files.

```
# write expert_advice data to a csv file
write_csv(expert_advice, "expert_advice.csv", na="")
```

When you run this code, a CSV file with the data should be saved in your `week7` folder. `na=""` ensures that any empty cells in the data frame are saved as blanks — R represents null values as `NA`, so if you don't include this, any null values will appear as `NA` in the saved file.

Group and summarize data

Calculate the total payments, by state

```
# calculate total payments by state
state_sum <- pfizer %>%
  group_by(state) %>%
  summarize(sum = sum(total)) %>%
  arrange(desc(sum))
```

Notice the use of `group_by` followed by `summarize` to **group** and **summarize** data, here using the function `sum`.

Calculate some additional summary statistics, by state

```
# As above, but for each state also calculate the median payment, and the
number of payments
state_summary <- pfizer %>%
  group_by(state) %>%
  summarize(sum = sum(total), median = median(total), count = n()) %>%
  arrange(desc(sum))
```

Notice the use of multiple summary functions, `sum`, `median`, and `n`. (You don't specify a variable for `n` because it is simply counting the number of rows in the data.)

Group and summarize for multiple categories

```
# as above, but group by state and category
```

```
state_category_summary <- pfizer %>%
  group_by(state, category) %>%
  summarize(sum = sum(total), median = median(total), count = n()) %>%
  arrange(state, category)
```

As for `arrange`, you can `group_by` by multiple variables, separated by commas.

Working with dates

Now let's run see how to work with dates, using the FDA warning letters data.

Filter the data for letters sent from the start of 2005 onwards

```
# FDA warning letters sent from the start of 2005 onwards
post2005 <- fda %>%
  filter(issued >= "2005-01-01") %>%
  arrange(issued)
```

Notice that operators like `>=` can be used for dates, as well as for numbers.

Count the number of letters issued by year

```
# count the letters by year
letters_year <- fda %>%
  mutate(year = format(issued, "%Y")) %>%
  group_by(year) %>%
  summarize(letters=n())
```

This code introduces **dplyr's** `mutate` function to create a new column in the data. The new variable `year` is the four-digit year `"%Y` (see here for more on time and date formats in R), extracted from the `issued` dates using the `format` function. Then the code groups by year and counts the number of letters for each one.

Add columns giving the number of days and weeks that have elapsed since each letter was sent

```
# add new columns showing many days and weeks elapsed since each letter was
sent
fda <- fda %>%
  mutate(days_elapsed = Sys.Date() - issued,
         weeks_elapsed = difftime(Sys.Date(), issued, units = "weeks"))
```

Notice in the first line that this code changes the `fda` data frame, rather than creating a new object. The function `Sys.Date` returns the current date, and if you subtract another date, it will calculate the difference in days. To calculate date and time differences using other units, use the `difftime` function.
Notice also that you can `mutate` multiple columns at one go, separated by commas.

Join data from two data frames

There are also a number of **join** functions in **dplyr** to combine data from two data frames. Here are the most useful:

- `inner_join()` returns values from both tables only where there is a match.
- `left_join()` returns all the values from the first-mentioned table, plus those from the second table that match.
- `semi_join()` filters the first-mentioned table to include only values that have matches in the second table.
- `anti_join()` filters the first-mentioned table to include only values that have no matches in the second table.

To illustrate, these **joins** will find doctors paid by Pfizer to run expert led forums who had also received a warning letter from the FDA:

```
# join to identify doctors paid to run Expert-led forums who also received a
warning letter
expert_warned_inner <- inner_join(pfizer, fda, by=c("first_name" =
"name_first", "last_name" = "name_last")) %>%
  filter(category=="Expert-Led Forums")

expert_warned_semi <- semi_join(pfizer, fda, by=c("first_name" = "name_first",
"last_name" = "name_last")) %>%
  filter(category=="Expert-Led Forums")
```

The code in `by=c()` defines how the **join** should be made. If instructions on how to join the tables are not supplied, **dplyr** will look for columns with matching names, and perform the **join** based on those.
The difference between the two **joins** above is that the first contains all of the columns from both data frames, while the second gives only columns from the `pfizer` data frame. In practice, you may wish to `inner_join` and then use **dplyr's** `select` function to select the columns that you want to retain, for example:

```
# as above, but select desired columns from data
expert_warned <- inner_join(pfizer, fda, by=c("first_name" = "name_first",
"last_name" = "name_last")) %>%
  filter(category=="Expert-Led Forums") %>%
  select(first_plus, last_name, city, state, total, issued)

expert_warned <- inner_join(pfizer, fda, by=c("first_name" = "name_first",
"last_name" = "name_last")) %>%
  filter(category=="Expert-Led Forums") %>%
  select(2:5,10,12)
```

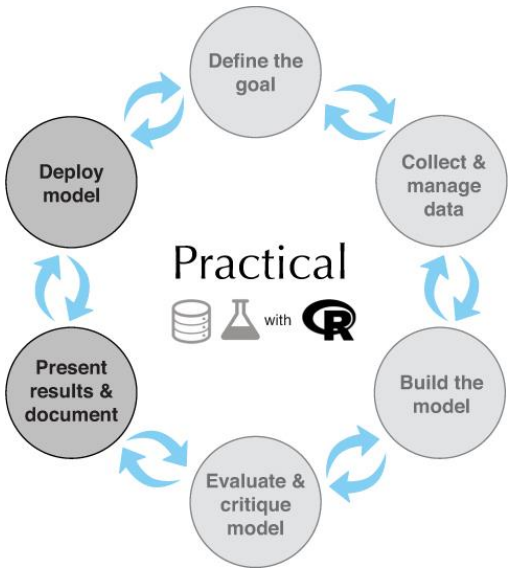
UNIT-IV

DELIVERING RESULTS: Documentation and deployment–producing effective presentations–Introduction to graphical analysis – plot()function – displaying multivariate data– matrix plots– multiple plots in one window - exporting graph – using graphics parameters in R Language.

DELIVERING RESULTS:

Documentation and deployment:

- Producing effective milestone documentation
- Managing project history using source control
- Deploying results and making demonstrations
- we'll survey techniques for documenting and deploying your work. We will work specific scenarios, and point to resources for further study if you want to master the techniques being discussed. The theme is this: now that you can build machine learning models, you should explore tools and procedures to become proficient at saving, sharing, and repeating successes. Our mental model) for this chapter emphasizes that this chapter is all about sharing what you model. Let's use to get some more-specific goals in this direction.



Goal Description

Goal	Description
Produce effective milestone documentation	A readable summary of project goals, data provenance, steps taken, and technical results (numbers and graphs). Milestone documentation is usually read by collaborators and peers, so it can be concise and can often include actual code. We'll demonstrate a great tool for producing excellent milestone documentation: the R *knitr* and *rmarkdown* packages, which we will refer to generically as *R markdown*. R markdown is a product of the "reproducible research" movement (see Christopher Gandrud's *Reproducible Research with R and RStudio*, Second Edition, Chapman and Hall, 2015) and is an excellent way to produce a reliable snapshot that not only shows the state of a project, but allows others to confirm the project works.
Manage a complete project history	It makes little sense to have exquisite milestone or checkpoint documentation of how your project worked last February if you can't get a copy of February's code and data. This is why you need good version control discipline to protect code, and good data discipline to preserve data.
Deploy demonstrations	True production deployments are best done by experienced engineers. These engineers know the tools and environment they will be deploying to. A good way to jump-start production deployment is to have a reference application. This allows engineers to experiment with your work, test corner cases, and build acceptance tests.

This chapter explains how to share your work—even sharing it with your future self. We'll discuss how to use R markdown to create substantial project milestone documentation and automate reproduction of graphs and other results. You'll learn about using effective

comments in code, and using Git for version management and for collaboration. We'll also discuss deploying models as HTTP services and applications.

For some of the examples, we will use RStudio, which is an integrated development environment (IDE) that is a product of RStudio, Inc. (and not part of R/CRAN itself). Everything we show can be done without RStudio, but RStudio supplies a basic editor and some single-button-press alternatives to some scripting tasks.

or our example scenario, we want to use metrics collected about the first few days of article views to predict the long-term popularity of an article. This can be important for selling advertising and predicting and managing revenue. To be specific: we will use measurements taken during the first eight days of an article's publication to predict if the article will remain popular in the long term.

Our tasks for this chapter are to save and share our Buzz model, document the model, test the model, and deploy the model into production.

To simulate our example scenario of predicting long term article popularity or buzz we will use the *Buzz dataset* from http://ama.liglab.fr/datasets/buzz/. We'll work with the data found in the file TomsHardware-Relative-Sigma-500.data.txt.[1] The original supplied documentation (TomsHardware-Relative-Sigma-500.names.txt and BuzzDataSetDoc.pdf) tells us the Buzz data is structured as shown in

Attribute	Description
Rows	Each row represents many different measurements of the popularity of a technical personal computer discussion topic.
Topics	Topics include technical issues about personal computers such as brand names, memory, overclocking, and so on.
Measurement types	For each topic, measurement types are quantities such as the number of discussions started, number of posts, number of authors, number of readers,

Attribute	Description
	and so on. Each measurement is taken at eight different times.
Times	The eight relative times are named 0 through 7 and are likely days (the original variable documentation is not completely clear and the matching paper has not yet been released). For each measurement type, all eight relative times are stored in different columns in the same data row.
Buzz	The quantity to be predicted is called *buzz* and is defined as being true or 1 if the ongoing rate of additional discussion activity is at least 500 events per day averaged over a number of days after the observed days. Likely buzz is a future average of the seven variables labeled *NAC* (the original documentation is unclear on this).

Using R markdown to produce milestone documentation

The first audience you'll have to prepare documentation for is yourself and your peers. You may need to return to previous work months later, and it may be in an urgent situation like an important bug fix, presentation, or feature improvement. For self/peer documentation, you want to concentrate on facts: what the stated goals were, where the data came from, and what techniques were tried. You assume that as long as you use standard terminology or references, the reader can figure out anything else they need to know. You want to emphasize any surprises or exceptional issues, as they're exactly what's expensive to relearn. You can't expect to share this sort of documentation with clients, but you can later use it as a basis for building wider documentation and presentations.

The first sort of documentation we recommend is project milestone or checkpoint documentation. At major steps of the project, you should take some time out to repeat your work in a clean environment (proving you know what's in intermediate files and you can in fact recreate them). An important, and often neglected, milestone is the start of a project. In this section, we'll use the knitr and rmarkdown R packages to document starting work with the Buzz data.

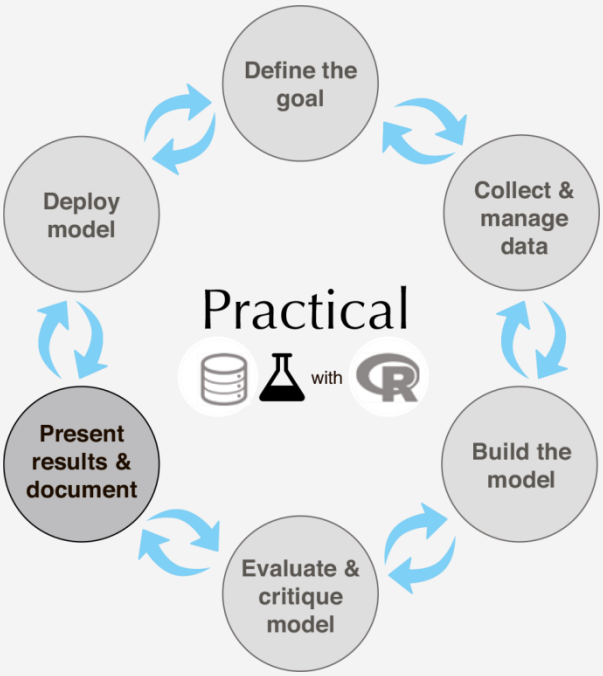
Documentation scenario: Share the ROC curve for the Buzz model

Our first task is to build a document that contains the ROC curve for the example model. We want to be able to rebuild this document automatically if we change model or evaluation data, so we will use R markdown to produce the document.

What is R markdown?

R markdown is a variation of the Markdown document specification[2] that allows the inclusion of R code and results inside documents. The concept of processing a combination of code and text should be credited to the R Sweave package[3] and from Knuth's formative ideas of literate programming.[4] In practice, you maintain a master file that contains both user-readable documentation and chunks of program source code. The document types supported by R markdown include Markdown, HTML, LaTeX, and Word. LaTeX format is a good choice for detailed, typeset, technical documents. Markdown format is a good choice for online documentation and wikis.

producing effective presentations:



We'll continue with the example from last chapter: our company (let's call it WVCorp) makes and sells home electronic devices and associated software and apps. WVCorp wants to monitor topics on the company's product forums and discussion board to identify "about-to-

buzz" issues: topics that are posed to generate a lot of interest and active discussion. This information can be used by product and marketing teams to proactively identify desired product features for future releases, and to quickly discover issues with existing product features. Once we've successfully built a model for identifying about-to-buzz topics on the forum, we'll want to explain the work to the project sponsor, and also to the product managers, marketing managers, and support engineering managers who will be using the results of our model.

Entity	Description
WVCorp	The company you work for
eRead	WVCorp's e-book reader
TimeWrangler	WVCorp's time-management app
BookBits	A competitor's e-book reader
GCal	A third-party cloud-based calendar service that TimeWrangler can integrate with

A disclaimer about the data and the example project

The dataset that we used for the buzz model was collected from Tom's Hardware (tomshardware.com), an actual forum for discussing electronics and electronic devices. Tom's Hardware is not associated with any specific product vendor, and the dataset doesn't specify the topics that were recorded. The example scenario we're using in this chapter was chosen to present a situation that would produce data similar to the data in the Tom's Hardware dataset. All product names and forum topics in our example are fictitious.

We provide the PDF versions (with notes) of our example presentations at *https://github.com/WinVector/zmPDSwR/tree/master/Buzz* as 1.ProjectSponsorPresentation.pdf, UserPresentation.pdf, and PeerPresentation.pdf.

We provide the PDF versions (with notes) of our example presentations at *https://github.com/WinVector/zmPDSwR/tree/master/Buzz* as ProjectSponsorPresentation.pdf, UserPresentation.pdf, and PeerPresentation.pdf.

Introduction to graphical analysis:

Objective: Introduction to Graphical Analysis To use Graphical Analysis program to graph sets of data and find the line of best fit and to identify and interpret the slope and y-intercept of this line.

Slope and y-intercept:

The equation of a linear line is: y = mx+b. "m" is the slope of the line, and "b" is the ySlope is a measure of how much a line rises or falls from one point on the line to another point on that line. The slope will have units of the y axis divided by the

$$\text{Slope} = \frac{Y_2 - Y_1}{X_2 - X_1} = \frac{\text{Rise}}{\text{Run}} = \frac{\Delta Y}{\Delta X}$$

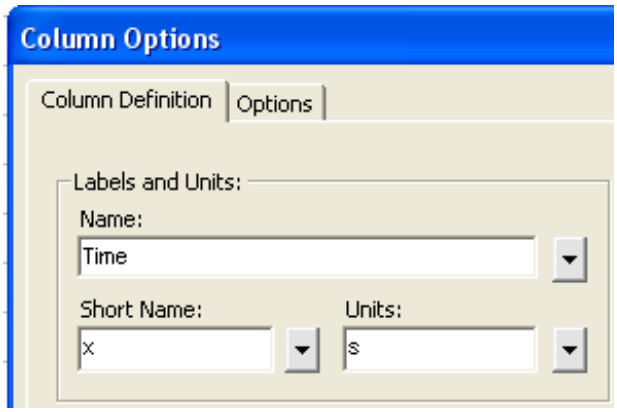
units of x-axis. To find the slope of a line using the math method:

The y-intercept is where the line crosses the y-axis. The y-intercept will have units of the y-axis.

Graphing y vs. x When graphing a ***y vs. x*** graph put the y values on the y-axis and the x values on the x-axis. Forexample when making a "distance vs. time " graph distance (m) goes on the y-axis and time (s) goes on the x-axis.

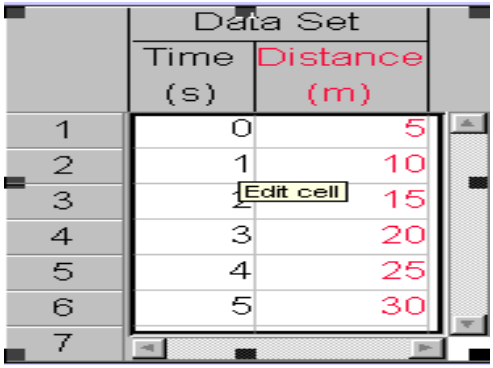
Using Graphical Analysis to plot a line

Using the computers in the Physics Lab (SM252) double click on the physics folder that is located on the desktop. Inside that folder is program called Graphical Analysis. Double click on that icon to startthe program. Double click on x column on the table and name the x-axis **Time** and give it units of seconds or s.



Do the same for the y column name it **Distance** and give it units of meters or m. Right click on the graph and select graph options andtitle the graph Distance vs Time and uncheck connect points.

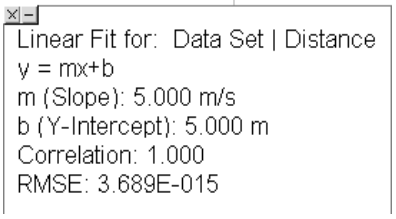
Now we need some data. Input these values in the table and graphical analysis will automatically plot the graph from the values that were entered. The graph will be linear. Instead of finding the slope of this line using the math we can have Graphical Analysis find it for us. In Graphical Analysis this is called a linear fit. Preforming alinear fit will also give us



the y intercept of the line. Using

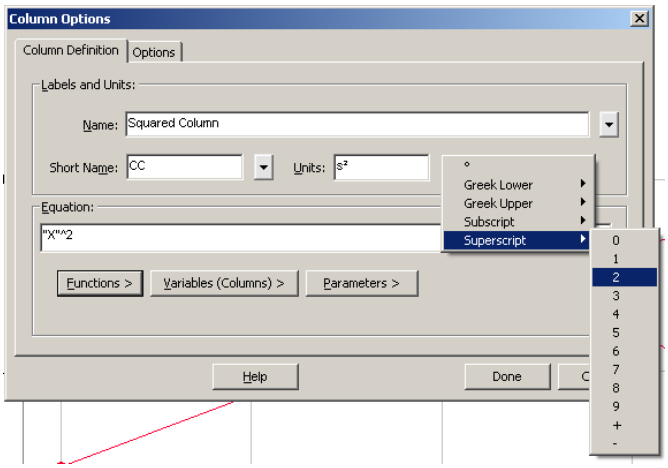
Using Graphical Analysis to find the slope and y-intercept

Next, we must tell the computer to which points we want to "fit" the line. Do this by clicking and holding on an area ofthe graph near the first plotted point. Then drag the cursor to form a gray box, which encloses all the data points. Let go of the mouse and click on the Analyze menu and select Linear Fit. Notice
the linear fit box gives you the values for the slope and y-intercept.



Using Graphical Analysis to square data

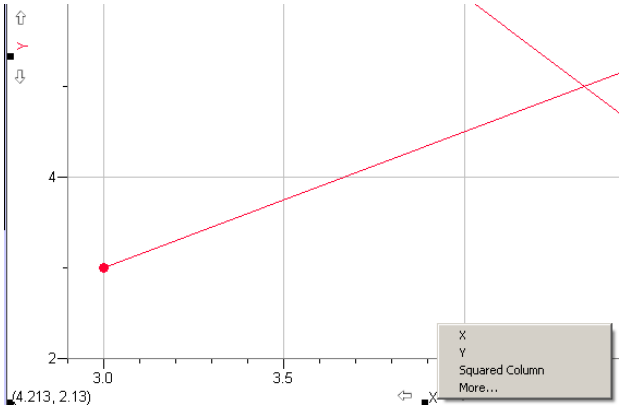
1. Click on the Data menu and select New Calculated Column. Label the new column and enter units in the box provided. To labelthe units correctly click the down arrow button next to the units box and click superscript and select 2.



2. Click the Variables button and select the column name which needs to be squared. The column name selected will then appearin the equation

box. In the equation box enter ^2 this will square your data in that column. Click done.

3. Left click on the axis that needs to be changedand select the new column made with the squared data.



Plot

The `plot()` function is used to draw points (markers) in a diagram.

The function takes parameters for specifying points in the diagram.

Parameter 1 specifies points on the **x-axis**.

Parameter 2 specifies points on the **y-axis**.

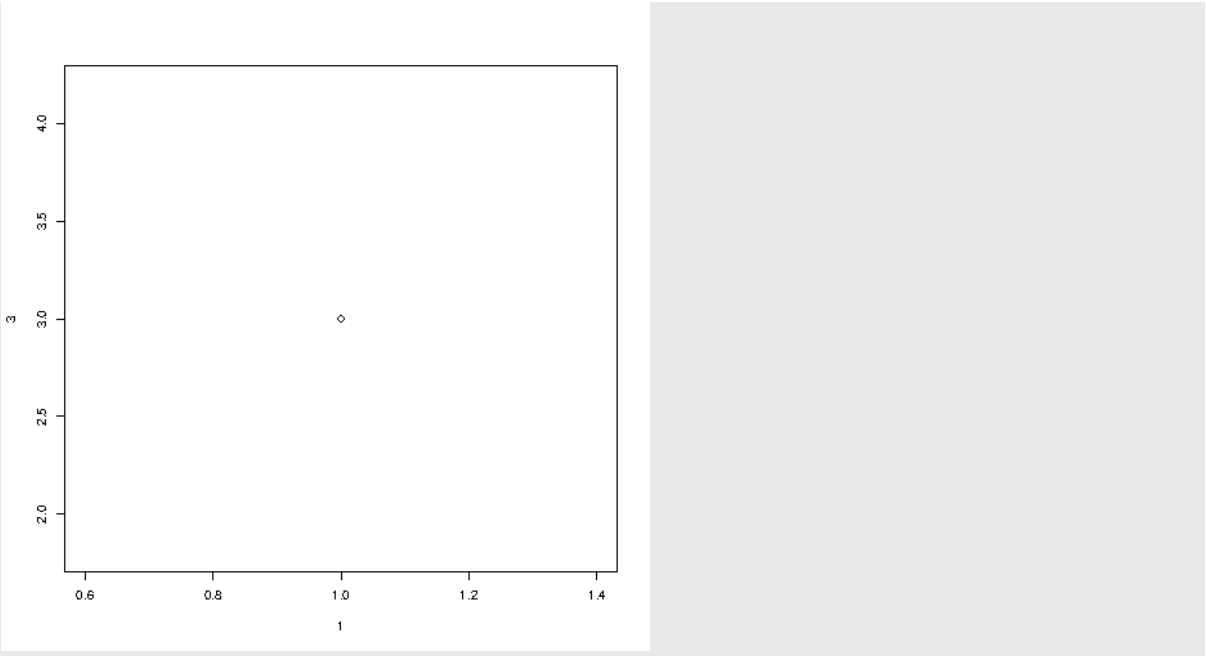
At its simplest, you can use the `plot()` function to plot two numbers against each other:

Example

Draw one point in the diagram, at position (1) and position (3):

```
plot(1, 3)
```

Result:



Displaying multivariate data: Multivariate Analysis

This booklet tells you how to use the R statistical software to carry out some simple multivariate analyses, with a focus on principal components analysis (PCA) and linear discriminant analysis (LDA).

This booklet assumes that the reader has some basic knowledge of multivariate analyses, and the principal focus of the booklet is not to explain multivariate analyses, but rather to explain how to carry out these analyses using R.

If you are new to multivariate analysis, and want to learn more about any of the concepts presented here, I would highly recommend the Open University book "Multivariate Analysis" (product code M249/03), available from from the Open University Shop.

In the examples in this booklet, I will be using data sets from the UCI Machine Learning Repository, http://archive.ics.uci.edu/ml.

Reading Multivariate Analysis Data into R

The first thing that you will want to do to analyse your multivariate data will be to read it into R, and to plot the data. You can read data into R using the read.table() function.

For example, the file http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data contains data on concentrations of 13 different chemicals in wines grown in the same region in Italy that are derived from three different cultivars.

The data set looks like this:

```
1,14.23,1.71,2.43,15.6,127,2.8,3.06,.28,2.29,5.64,1.04,3.92,1065
1,13.2,1.78,2.14,11.2,100,2.65,2.76,.26,1.28,4.38,1.05,3.4,1050
1,13.16,2.36,2.67,18.6,101,2.8,3.24,.3,2.81,5.68,1.03,3.17,1185
1,14.37,1.95,2.5,16.8,113,3.85,3.49,.24,2.18,7.8,.86,3.45,1480
1,13.24,2.59,2.87,21,118,2.8,2.69,.39,1.82,4.32,1.04,2.93,735
...
```

There is one row per wine sample. The first column contains the cultivar of a wine sample (labelled 1, 2 or 3), and the following thirteen columns contain the concentrations of the 13 different chemicals in that sample. The columns are separated by commas.

When we read the file into R using the read.table() function, we need to use the "sep=" argument in read.table() to tell it that the columns are separated by commas. That is, we can read in the file using the read.table() function as follows:

```
> wine <- read.table("http://archive.ics.uci.edu/ml/machine-learning-
databases/wine/wine.data",
          sep=",")
> wine
     V1    V2   V3   V4   V5  V6   V7   V8   V9  V10       V11   V12  V13  V14
 1    1 14.23 1.71 2.43 15.6 127 2.80 3.06 0.28 2.29  5.640000 1.040 3.92 1065
 2    1 13.20 1.78 2.14 11.2 100 2.65 2.76 0.26 1.28  4.380000 1.050 3.40 1050
 3    1 13.16 2.36 2.67 18.6 101 2.80 3.24 0.30 2.81  5.680000 1.030 3.17 1185
 4    1 14.37 1.95 2.50 16.8 113 3.85 3.49 0.24 2.18  7.800000 0.860 3.45 1480
 5    1 13.24 2.59 2.87 21.0 118 2.80 2.69 0.39 1.82  4.320000 1.040 2.93  735
 ...
 176  3 13.27 4.28 2.26 20.0 120 1.59 0.69 0.43 1.35 10.200000 0.590 1.56  835
 177  3 13.17 2.59 2.37 20.0 120 1.65 0.68 0.53 1.46  9.300000 0.600 1.62  840
 178  3 14.13 4.10 2.74 24.5  96 2.05 0.76 0.56 1.35  9.200000 0.610 1.60  560
```

In this case the data on 178 samples of wine has been read into the variable 'wine'.

Plotting Multivariate Data

Once you have read a multivariate data set into R, the next step is usually to make a plot of the data.

A Matrix Scatterplot

One common way of plotting multivariate data is to make a "matrix scatterplot", showing each pair of variables plotted against each other. We can use the "scatterplotMatrix()" function from the "car" R package to do this. To use this function, we first need to install the "car" R package (for instructions on how to install an R package, see How to install an R package).

Once you have installed the "car" R package, you can load the "car" R package by typing:

```
> library("car")
```

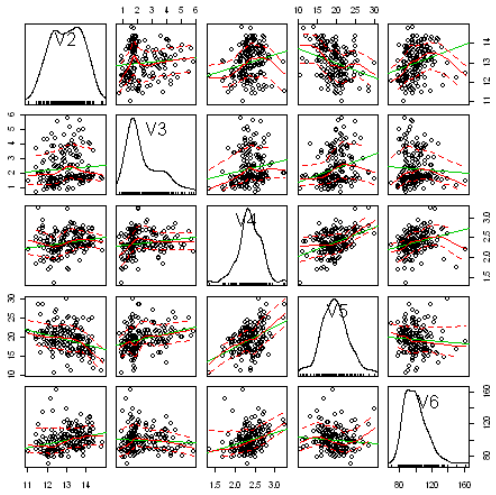
You can then use the "scatterplotMatrix()" function to plot the multivariate data.

To use the scatterplotMatrix() function, you need to give it as its input the variables that you want included in the plot. Say for example, that we just want to include the variables corresponding to the concentrations of the first five chemicals. These are stored in columns 2-6 of the variable "wine". We can extract just these columns from the variable "wine" by typing:

```
> wine[2:6]
        V2   V3   V4   V5  V6
1   14.23 1.71 2.43 15.6 127
2   13.20 1.78 2.14 11.2 100
3   13.16 2.36 2.67 18.6 101
4   14.37 1.95 2.50 16.8 113
5   13.24 2.59 2.87 21.0 118
...
```

To make a matrix scatterplot of just these 13 variables using the scatterplotMatrix() function we type:

```
> scatterplotMatrix(wine[2:6])
```



In this matrix scatterplot, the diagonal cells show histograms of each of the variables, in this case the concentrations of the first five chemicals (variables V2, V3, V4, V5, V6).

Each of the off-diagonal cells is a scatterplot of two of the five chemicals, for example, the second cell in the first row is a scatterplot of V2 (y-axis) against V3 (x-axis).

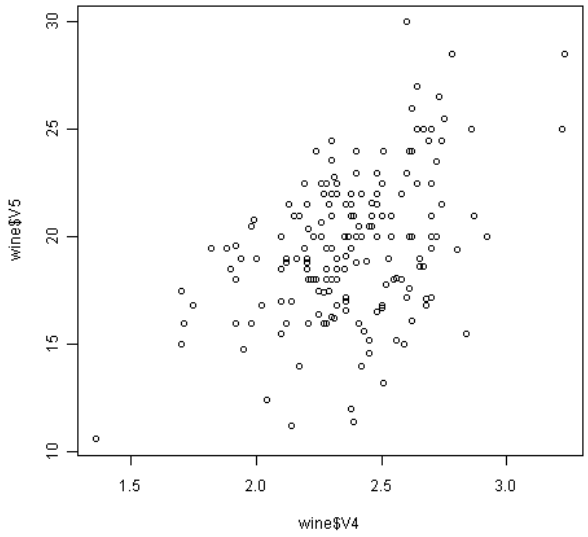
A Scatterplot with the Data Points Labelled by their Group

If you see an interesting scatterplot for two variables in the matrix scatterplot, you may want to plot that scatterplot in more detail, with the data points labelled by their group (their cultivar in this case).

For example, in the matrix scatterplot above, the cell in the third column of the fourth row down is a scatterplot of V5 (x-axis) against V4 (y-axis). If you look at this scatterplot, it appears that there may be a positive relationship between V5 and V4.

We may therefore decide to examine the relationship between V5 and V4 more closely, by plotting a scatterplot of these two variable, with the data points labelled by their group (their cultivar). To plot a scatterplot of two variables, we can use the "plot" R function. The V4 and V5 variables are stored in the columns V4 and V5 of the variable "wine", so can be accessed by typing wine$V4 or wine$V5. Therefore, to plot the scatterplot, we type:

```
> plot(wine$V4, wine$V5)
```



If we want to label the data points by their group (the cultivar of wine here), we can use the "text" function in R to plot some text beside every data point. In this case, the cultivar of wine is stored in the column V1 of the variable "wine", so we type:

```
> text(wine$V4, wine$V5, wine$V1, cex=0.7, pos=4, col="red")
```

If you look at the help page for the "text" function, you will see that "pos=4" will plot the text just to the right of the symbol for a data point. The "cex=0.5" option will plot the text at half the default size, and the "col=red" option will plot the text in red. This gives us the following plot:

We can see from the scatterplot of V4 versus V5 that the wines from cultivar 2 seem to have lower values of V4 compared to the wines of cultivar 1.

Matrix plots:

he aim of the package `plot.matrix` is to visualize a matrix as is with a heatmap. Automatic reordering of rows and columns is only done if necessary. This is different as in similar function like `heatmap`. Additionally it should be user-friendly and give access to a lot of options if necessary.

Currently the package implements the S3 functions below such that you can use the generic `plot` function to plot matrices as heatmaps:

- `plot.matrix` for a heatmap for a plain matrix,
- `plot.loadings` for a heatmap for a loadings matrix from factor analysis or principal component analysis (reordering of rows!).

The plot itself is composed by a heatmap (usually left) where colors represent matrix entries and a key (usually right) which links the colors to the values.

First examples

```
library('plot.matrix')
# numeric matrix
x <- matrix(runif(35), ncol=5) # create a numeric matrix object
class(x)
#> [1] "matrix" "array"
par(mar=c(5.1, 4.1, 4.1, 4.1)) # adapt margins
plot(x)
```

```
# logical matrix
m <- matrix(runif(35)<0.5, ncol=7)
plot(m)
```

```
# text matrix
s <- matrix(sample(letters[1:10], 35, replace=TRUE), ncol=5)
plot(s)
library('plot.matrix')
library('psych')
data <- na.omit(bfi[,1:25])
fa <- fa(data, 5, rotate="varimax")
par(mar=c(5.1, 4.1, 4.1, 4.1)) # adapt margins
plot(loadings(fa), cex=0.5)
```

Assigning colors and breaks

`plot.matrix` uses the command `assignColors`, also part of `plot.matrix`, assigns to each value in `x` a color based on the parameters `breaks`, `col` and `na.col` given.

In case of a numeric matrix `breaks` can be

- a number, giving the number of intervals covering the range of `x`,
- a vector of two numbers, given the range to cover with 10 intervals, or
- a vector with more than two numbers, specify the interval borders

In case of a non-numeric vector `breaks` must contain all values which are will get a color. If `breaks` is not given then a sensible default is chosen: in case of a numeric vector derived from `pretty` and otherwise all unique values/levels are used.

`col` can be either be a vector of colors or a function which generates via `col(n)` a set of `n` colors. The default is to use `heat.colors`.

Choosing color palettes/functions

In case that you want to provide your own color palettes/functions for plotting there are several good choices within R packages:

Source: Datanovia - Top R Color Palettes to Know for Great Data Visualization

- `viridis` or `viridisLite`,
- `RColorBrewer`,
- `ggplot2`,
- `ggsci`,

- wesanderson,
- cetcolor,
- colormap,
- ColorPalette,
- colorr,
- colorRamps,
- dichromat,
- jcolors,
- morgenstemning,
- painter,
- paletteer,
- pals,
- Polychrome,
- qualpalr,
- randomcoloR, or
- Redmonder.

Structure of the plot

The plot is created in several steps

1. a call to the `plot` command to create the basic plot
2. draw colored polygons for each matrix entry with the `polygon` command
3. if necessary add the value of each matrix entry with the `text` command in a polygon
4. if necessary draw x- and y-axis with the `axis` command into the plot
5. if necessary draw the key with the `axis` and the `polygon` command

Formal parameters

plot.matrix(	x,= NULL		
	y	=	NULL,
	breaks	=	NULL,
	col	=	heat.colors,

```
                na.col          =  “white”,
                na.cell         =  TRUE,
                na.print        =  TRUE,
                digits          =  NA,
                fmt.cell        =  NULL,
                fmt.key         =  NULL,
                spacing.key     =  c(1, 0.5, 0),
                polygon.cell    =  NULL,
                polygon.key     =  NULL,
                text.cell       =  NULL,
                key             =  list(side = 4, las = 1),
                axis.col        =  list(side = 1),
                axis.row        =  list(side = 2),
                axis.key        =  NULL,
                max.col         =  70,
                …)
```

You may influence the appearance by setting your own parameters:

1. `. . .` all parameters given here will be given to the `plot` command, e.g. `xlab`, `ylab`, ….
2. `polygon.cell` list of parameters for drawing polygons for matrix entries
3. `text.cell` list of parameters for putting for matrix entries as texts

4. `axis.col` and `axis.row` list of parameters for drawing for row and column axes
5. `key`, `axis.key`, `spacing.key` and `polygon.key` to draw the key
6. `max.col` to determine when text color and background color to near

Set global parameters

You may set global parameters for all subsequent calls of `axis`, `polygon` and `text` via the . . . . The following parameters are supported

function	parameter(s)
axis	cex.axis, col.axis, col.ticks, font, font.axis, hadj, las, lwd.ticks, line , outer, padj, tck, tcl, tick
polygon	angle, border, density
text	cex, font, vfont

```
par(mar=c(5.1, 4.1, 4.1, 4.1)) # adapt margins
# omit all borders
plot(x, border=NA)
```

```
x <- matrix(runif(35), ncol=5) # create a numeric matrix object
par(mar=c(5.1, 4.1, 4.1, 4.1)) # adapt margins such that all labels are visible
plot(x, axis.col=list(side=1, las=2), axis.row = list(side=2, las=1))
```

You need to access the position of the cell text used by accessing the invisible return of `plot.matrix`. The `cell.text` contains the parameters used to draw the text.

Note the double braces: `[[i,j]]`

```
param_est <- matrix(runif(25), nrow=5)
par(mar=c(5.1, 4.1, 4.1, 4.1)) # adapt margins such that all labels are visible
res <- plot(param_est, digits=2, text.cell=list(pos=3, cex=0.75))
sderr_est <- matrix(runif(25)/10, nrow=5)
for (i in 1:nrow(param_est)) {
  for (j in 1:ncol(param_est)) {
    args <- res$cell.text[[i,j]]
```

```
    args$labels <- paste0('(', fmt(sderr_est[i,j], 3), ')')
    args$cex    <- 0.5
    args$pos    <- 1
    do.call(text, args)
  }
}
```

Or alternatively

```
param_est <- matrix(runif(25), nrow=5)
par(mar=c(5.1, 4.1, 4.1, 4.1)) # adapt margins such that all labels are visible
res <- plot(param_est, digits=2, text.cell=list(cex=0.75))
sderr_est <- matrix(runif(25)/10, nrow=5)
for (i in 1:nrow(param_est)) {
  for (j in 1:ncol(param_est)) {
    args <- res$cell.text[[i,j]]
    args$labels <- paste0('(', round(sderr_est[i,j], 3), ')')
    args$cex    <- 0.6
    args$y      <- args$y-0.3
    do.call(text, args)
  }
}
```

Modifying a plot

Defaults

The default plot always draws a heatmap and a key where the colors and breaks are determined by the entries of `x`. In case of a numeric matrix ten colors from `heat.colors` are chosen and eleven breaks with cover the range of entries with an equidistant grid. In case of a non-numeric matrix each unique element gets a color determined from `heat.colors`.

Modifying the breaks

In case of a numeric matrix the `breaks` give the interval borders for a color otherwise for each unique matrix entry `breaks` should contain a value. If `breaks` are not given then they will be determined from the matrix object by using the `pretty` function of base R.

```
par(mar=c(5.1, 4.1, 4.1, 4.1))    # adapt margins
# we only want the range of x
```

```
plot(x, breaks=range(x))
```

Multiple plots in one window:

R par() function

We can put multiple graphs in a single plot by setting some graphical parameters with the help of `par()` function. R programming has a lot of graphical parameters which control the way our graphs are displayed.

The `par()` function helps us in setting or inquiring about these parameters. For example, you can look at all the parameters and their value by calling the function without any argument.

```
>par()

$xlog

[1] FALSE

...

$yaxt

[1] "s"

$ylbias

[1] 0.2
```

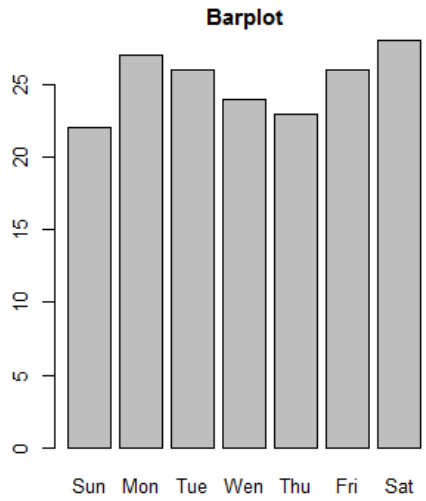
You will see a long list of parameters and to know what each does you can check the help section `?par`. Here we will focus on those which help us in creating subplots.

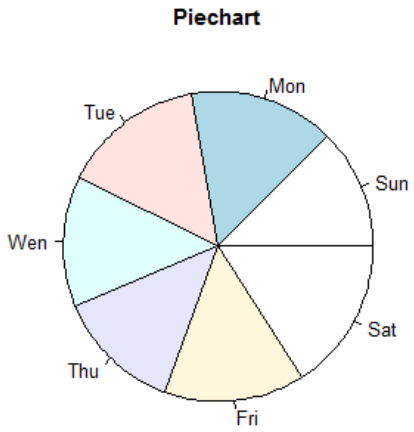
Graphical parameter `mfrow` can be used to specify the number of subplot we need.

It takes in a vector of form `c(m, n)` which divides the given plot into m*n array of subplots. For example, if we need to plot two graphs side by side, we would have `m=1` and `n=2`. Following example illustrates this.

```
>max.temp    # a vector used for plotting
```

```
Sun Mon Tue Wen Thu Fri Sat

22  27  26  24  23  26  28

par(mfrow=c(1,2))    # set the plotting area into a 1*2 array

barplot(max.temp, main="Barplot")

pie(max.temp, main="Piechart", radius=1)
```



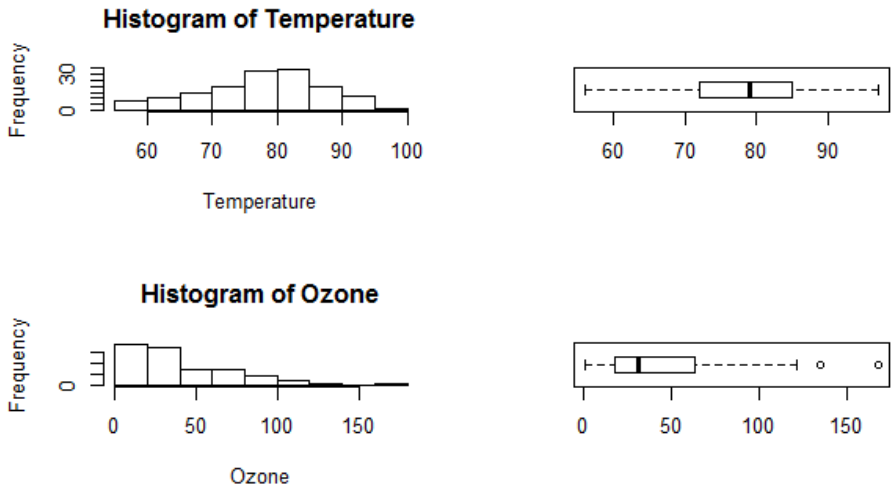


This same phenomenon can be achieved with the graphical parameter `mfcol`.

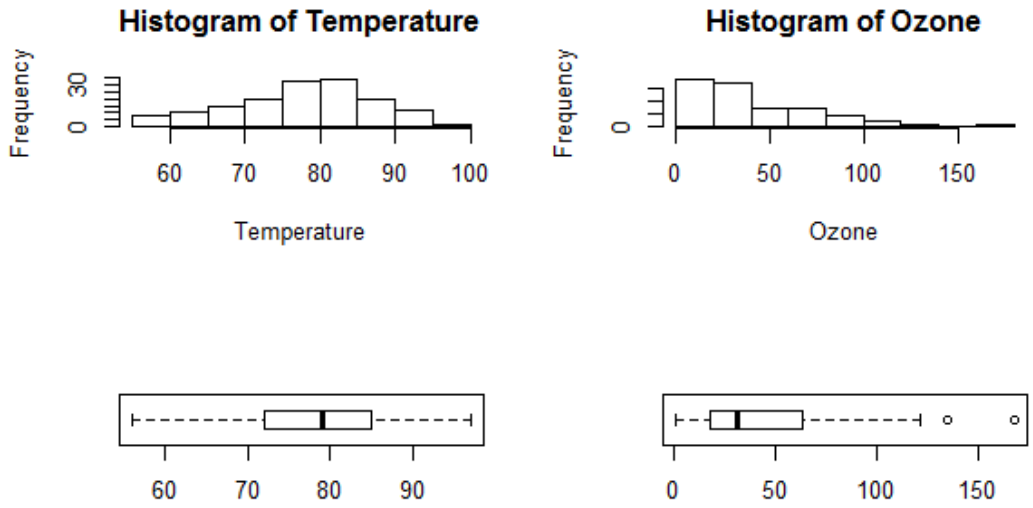
The only difference between the two is that, `mfrow` fills in the subplot region row wise while `mfcol` fills it column wise.

```
Temperature <- airquality$Temp

Ozone <- airquality$Ozone

par(mfrow=c(2,2))

hist(Temperature)

boxplot(Temperature, horizontal=TRUE)
```

```
hist(Ozone)

boxplot(Ozone, horizontal=TRUE)
```



Same plot with the change `par(mfcol = c(2, 2))` would look as follows. Note that only the ordering of the subplot is different.



More Precise Control

The graphical parameter `fig` lets us control the location of a figure precisely in a plot.

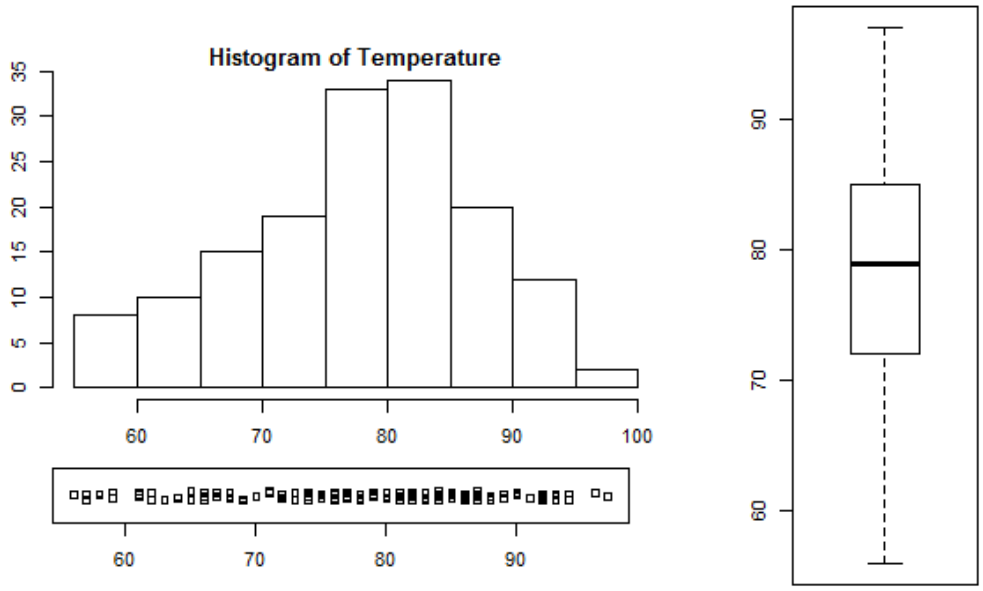
We need to provide the coordinates in a normalized form as `c(x1, x2, y1, y2)`. For example, the whole plot area would be `c(0, 1, 0, 1)` with `(x1, y1) =`

`(0, 0)` being the lower-left corner and `(x2, y2) = (1, 1)` being the upper-right corner.

Note: we have used parameters `cex` to decrease the size of labels and `mai` to define margins.

```
# make labels and margins smaller

par(cex=0.7, mai=c(0.1,0.1,0.2,0.1))

Temperature <- airquality$Temp

# define area for the histogram

par(fig=c(0.1,0.7,0.3,0.9))

hist(Temperature)

# define area for the boxplot

par(fig=c(0.8,1,0,1), new=TRUE)

boxplot(Temperature)

# define area for the stripchart

par(fig=c(0.1,0.67,0.1,0.25), new=TRUE)

stripchart(Temperature, method="jitter")
```

The numbers assigned to `fig` were arrived at with a hit-and-trial method to achieve the best looking plot.



Exporting graph:

Creating plots in R is all well and good but what if you want to use these plots in your thesis, report or publication? One option is to click on the 'Export' button in the 'Plots' tab in RStudio as we described previously. You can also export your plots from R to an external file by writing some code in your R script. The advantage of this approach is that you have a little more control over the output format and it also allows you to generate (or update) plots automatically whenever you run your script. You can export your plots in many different formats but the most common are, pdf, png, jpeg and tiff.

By default, R (and therefore RStudio) will direct any plot you create to the plot window. To save your plot to an external file you first need to redirect your plot to a different graphics device. You do this by using one of the many graphics device functions to start a new graphic device. For example, to save a plot in pdf format we will use the pdf() function. The first argument in the pdf() function is the filepath and filename of the file we want to save (don't forget to include the .pdf extension). Once we've used the pdf() function we can then write all of the code we used to create our plot including any graphical parameters such as setting the margins and splitting up the plotting device. Once the code has run we need to close the pdf plotting device using the dev.off() function.

```
pdf(file = 'output/my_plot.pdf')
par(mar = c(4.1, 4.4, 4.1, 1.9), xaxs="i", yaxs="i")
plot(flowers$weight, flowers$shootarea,
     xlab = "weight (g)",
     ylab = expression(paste("shoot area (cm"^"2",")")),
     xlim = c(0, 30), ylim = c(0, 200), bty = "l",
     las = 1, cex.axis = 0.8, tcl = -0.2,
```

```
      pch = 16, col = "dodgerblue1", cex = 0.9)
text(x = 28, y = 190, label = "A", cex = 2)
dev.off()
```

If we want to save this plot in png format we simply use the png() function in more or less the same way we used the pdf() function.

```
png('output/my_plot.png')
par(mar = c(4.1, 4.4, 4.1, 1.9), xaxs="i", yaxs="i")
plot(flowers$weight, flowers$shootarea,
      xlab = "weight (g)",
      ylab = expression(paste("shoot area (cm"^"2",")")),
      xlim = c(0, 30), ylim = c(0, 200), bty = "l",
      las = 1, cex.axis = 0.8, tcl = -0.2,
      pch = 16, col = "dodgerblue1", cex = 0.9)
text(x = 28, y = 190, label = "A", cex = 2)
dev.off()
```

using graphics parameters in R Language:

You can customize many features of your graphs (fonts, colors, axes, titles) through graphic options.

One way is to specify these options in through the **par( )** function. If you set parameter values here, the changes will be in effect for the rest of the session or until you change them again. The format is **par(*optionname*=*value*, *optionname*=*value*, ...)**

```
# Set a graphical parameter using par()


par()               # view current settings
opar <- par()       # make a copy of current settings
par(col.lab="red") # red x and y labels
hist(mtcars$mpg)    # create a plot with these new settings
par(opar)           # restore original settings
```

A second way to specify graphical parameters is by providing the ***optionname***=*value* pairs directly to a high level plotting function. In this case, the options are only in effect for that specific graph.

```
# Set a graphical parameter within the plotting function
hist(mtcars$mpg, col.lab="red")
```

See the help for a specific high level plotting function (e.g. plot, hist, boxplot) to determine which graphical parameters can be set this way.

The remainder of this section describes some of the more important graphical parameters that you can set.

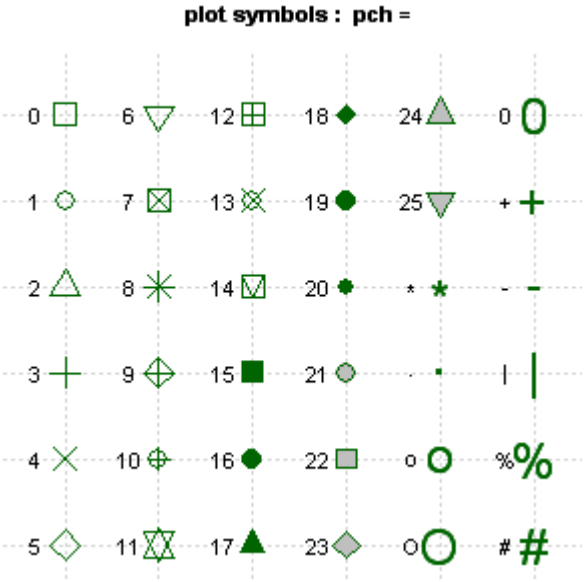
Text and Symbol Size

The following options can be used to control text and symbol size in graphs.

option	description
cex	number indicating the amount by which plotting text and symbols should be scaled relative to the default. 1=default, 1.5 is 50% larger, 0.5 is 50% smaller, etc.
cex.axis	magnification of axis annotation relative to cex
cex.lab	magnification of x and y labels relative to cex
cex.main	magnification of titles relative to cex
cex.sub	magnification of subtitles relative to cex

Plotting Symbols

Use the **pch=** option to specify symbols to use when plotting points. For symbols 21 through 25, specify border color (col=) and fill color (bg=).

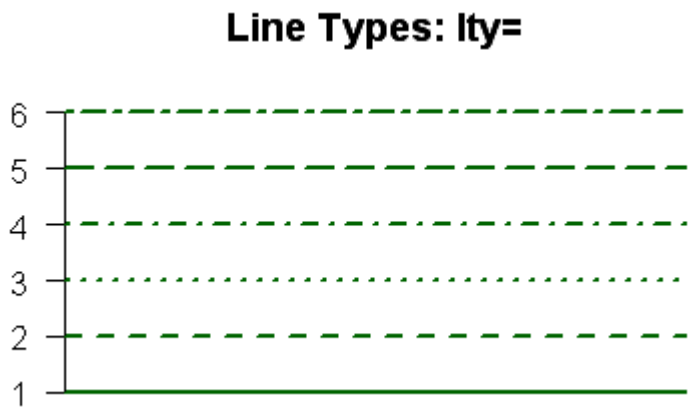


Lines

You can change lines using the following options. This is particularly useful for reference lines, axes, and fit lines.

option	description
lty	line type. see the chart below.

lwd	line width relative to the default (default=1). 2 is twice as wide.

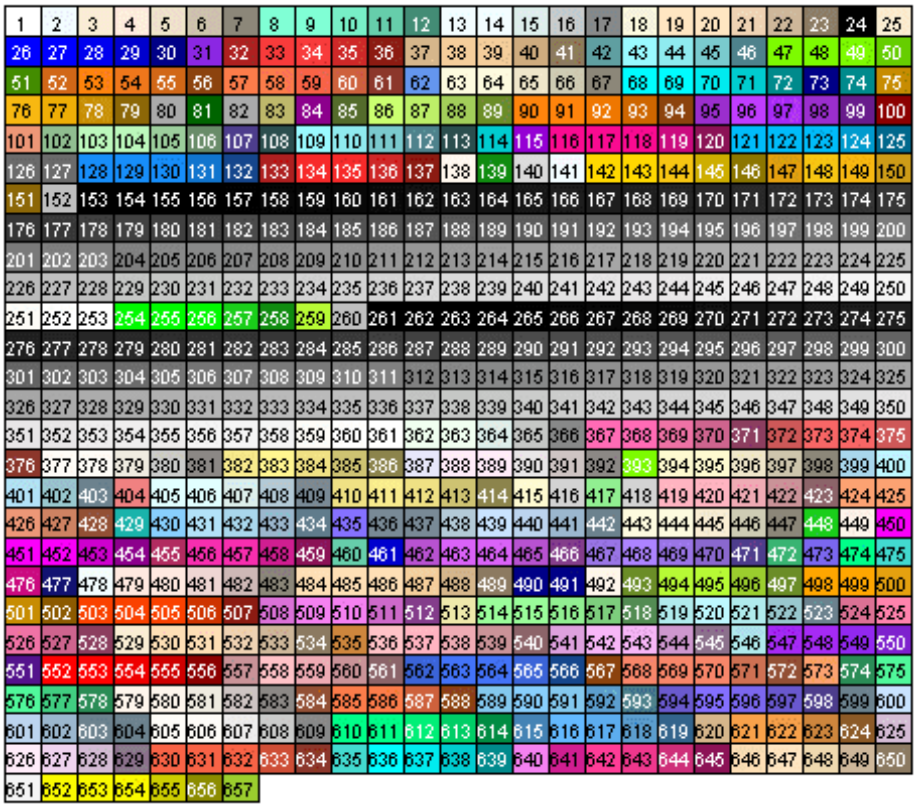


Colors

Options that specify colors include the following.

option	description
col	Default plotting color. Some functions (e.g. lines) accept a vector of values that are recycled.
col.axis	color for axis annotation
col.lab	color for x and y labels
col.main	color for titles
col.sub	color for subtitles
fg	plot foreground color (axes, boxes - also sets col= to same)
bg	plot background color

You can specify colors in R by index, name, hexadecimal, or RGB.
For example **col=1**, **col="white"**, and **col="#FFFFFF"** are equivalent.
The following chart was produced with code developed by Earl F. Glynn. See his Color Chart for all the details you would ever need about using colors in R.



You can also create a vector of *n* contiguous colors using the functions **rainbow(*n*)**, **heat.colors(*n*)**, **terrain.colors(*n*)**, **topo.colors(*n*)**, and **cm.colors(*n*)**.
colors() returns all available color names.

Fonts

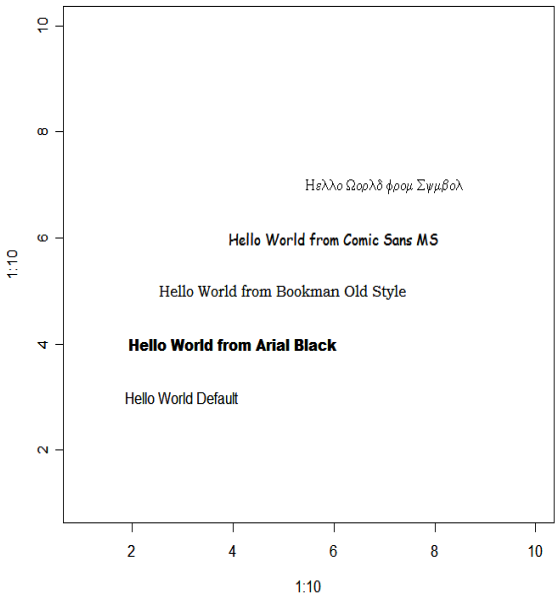
You can easily set font size and style, but font family is a bit more complicated.

option	description
font	Integer specifying font to use for text. 1=plain, 2=bold, 3=italic, 4=bold italic, 5=symbol
font.axis	font for axis annotation
font.lab	font for x and y labels
font.main	font for titles
font.sub	font for subtitles

ps	font point size (roughly 1/72 inch) text size=ps*cex
family	font family for drawing text. Standard values are "serif", "sans", "mono", "symbol". Mapping is device dependent.

In windows, mono is mapped to "TT Courier New", serif is mapped to"TT Times New Roman", sans is mapped to "TT Arial", mono is mapped to "TT Courier New", and symbol is mapped to "TT Symbol" (TT=True Type). You can add your own mappings.

```
# Type family examples - creating new mappings
plot(1:10,1:10,type="n")
windowsFonts(
  A=windowsFont("Arial Black"),
  B=windowsFont("Bookman Old Style"),
  C=windowsFont("Comic Sans MS"),
  D=windowsFont("Symbol")
)
text(3,3,"Hello World Default")
text(4,4,family="A","Hello World from Arial Black")
text(5,5,family="B","Hello World from Bookman Old Style")
text(6,6,family="C","Hello World from Comic Sans MS")
text(7,7,family="D", "Hello World from Symbol")
```



Margins and Graph Size

You can control the margin size using the following parameters.

option	description
mar	numerical vector indicating margin size c(bottom, left, top, right) in lines. default = c(5, 4, 4, 2) + 0.1
mai	numerical vector indicating margin size c(bottom, left, top, right) in inches
pin	plot dimensions (width, height) in inches

For complete information on margins, see Earl F. Glynn's **margin tutorial.**

Going Further

See **help(par)** for more information on graphical parameters. The customization of plotting axes and text annotations are covered **next section.**

****