# Overview of the Data Mining Process
# Part 1: Importing Data into R

Aram Balagyozyan

Department of Operations and Information Management
Kania School of Management
The
University of Scranton

February 13, 2022

# Outline

1. Some Preliminaries
2. Importing Data into R

# Outline

1. Some Preliminaries
2. Importing Data into R

# Introduction

- In this module, we will discuss issues related to data collection, cleaning, and preprocessing.
- We will introduce the notion of data partitioning, where methods are trained on a set of training data and then their performance is evaluated on a separate set of validation data, as well as explain how this practice helps avoid overfitting.
- Finally, we will discuss the steps of model building.

# Core Ideas in Data Mining

- Some of the most frequent tasks in data mining are **classification**, **prediction**, and **association rules**.
- **Classification** involves tasks where the class of any given data unit is unknown (or will become apparent only in future) and the analyst's goal is to predict what classification is or what will be. E.g., a credit card transaction can be normal or fraudulent, a given street parking spot may be available or not, a victim of an illness can be recovered, remain ill, or be deceased.
- **Prediction** is very similar to classification except that we are trying to predict the values of a numerical variable (e.g., amount of purchase) rather than a class (e.g., purchase or non-purchase).
- **Association Rules or Affinity Analysis** is designed to find general associations patterns between items in a large database. E.g. grocery stores can use association rules for product placement, weekly promotional offers, and bundling products.

# Core Ideas in Data Mining

- Online recommender systems, such as those used on Amazon and Netflix use *collaborative filtering*, a method that uses individual users' as well as other users' preferences and tastes given their historic purchases, rating, browsing, or any other measurable behavior indicative of an individual' preference.
- In contrast to *association rules* that generate rules general to an entire population, *collaborative filtering* generates rules specific for an individual user.

# Core Ideas in Data Mining

- **Predictive analytics** is a sub-field of analytics that incorporates classification, prediction, association rules, and collaborative filtering methods. In this course, we will dedicate three units to predictive analytics (multiple-regression, classification-tree, and logistic-regression analyses).
- **Descriptive Analytics** is another branch of analytics that incorporates all the methods for exploring and describing data. These methods include data cleaning, manipulation, aggregation, numerical summarization, dimension reduction, and visualization. Two unit of this course will be dedicated to descriptive analytics (data visualization and dimension reduction).

# Core Ideas in Data Mining

- **Prescriptive Analytics** is yet another branch of analytics that entails the application of mathematical and computational sciences and suggests decision options to take advantage of the results of descriptive and predictive analytics. Prescription analytics may involve methods such as revenue and sales optimization or optimal customer segmentation. We will dedicate one unit of this course to predictive analytics (integer optimization).

# Core Ideas in Data Mining

- A fundamental distinction among data mining techniques is between *supervised* and *unsupervised methods*.
- **Supervised learning** algorithms are those used in classification and prediction (e.g., linear regression)
- In supervised learning, we must have data available in which the value of the outcome of interest (e.g., purchase or no purchase) is known.
- Such data are also called "labeled data," since they contain the label (outcome value) for each record.
- *Training data* are the data from which the classification or prediction algorithm "learns," or is "trained," about the relationship between predictor variables and the outcome variable. Once the algorithm has learned from the training data, it is then applied to another sample of labeled data (the validation and test data) where the outcome is known but initially hidden, to see how well it does compared to other models.

# Core Ideas in Data Mining

- **Unsupervised learning** algorithms are those used where there is no outcome variable to predict or classify. Hence, there is no "learning" from cases where such an outcome variable is known. (e.g., association rules, dimension reduction methods, and clustering techniques).
- Supervised and unsupervised methods are sometimes used in conjunction. For example, unsupervised clustering methods are used to separate loan applicants into several risk-level groups. Then, supervised algorithms are applied separately to each risk-level group for predicting propensity of loan default.

# Organization of Data Sets

- There is a diverse set of data sources that a data analyst may deal with (e.g. sensors, Internet, etc.) and there are also many different types of data (e.g. text, sound, images, etc.). Still, most modeling techniques and packages are a bit restrictive on the type of data that they can handle. This means that a lot of time is spent on pre-processing these data into some data structure that is manageable by these standard techniques.
- Regardless, most data tables (or data sets) are two-dimensional where each row represents an entity (e.g. product, person, etc.) and the columns represent properties of the entities (e.g name, age, temperature).
- Terms *entity* and *property* have many synonyms:
  - *entity* = objects, tuples, records, examples, feature-vectors.
  - *property* = features, attributes, variables, dimensions, or fields.

# Organization of Data sets (continued)

- In terms of rows, data sets can have two main types of setups:
  a. Each row is independent from the others,
  b. There is some dependency among rows. E.g some form of time or spatial order among observations (in time-series and spatial data respectively)
- Regarding columns, we can talk about the type of data values they store. The most frequent distinction is between **quantitative** and **categorical** variables. A finer taxonomy of the data types can cast these into:
  a. numerical: able to assume any real numerical value, usually in a given range.
  b. integer: taking only integer values.
  c. date.
  d. categorical: assuming one of a limited values number of values.
     - Categorical variables can be coded as numerical (1,2,3, etc.) or text (male, female).
     - Categorical variables can be unordered (also called *nominal*, again e.g. male, female) or ordered (aka *ordinal variables*, e.g. freshman, sophomore, etc.)

# Outline

# Importing Text Files

- To demonstrate the data import process, we will rely on the WestRoxbury.csv file that includes information on values and 13 other variables on 5,802 single family, owner-occupied homes in West Roxbury, Boston, MA in 2014.
- Each row represents a home. E.g. the first home's assessed value is $344,200, its tax was $4430, and so on.
- In supervised learning situations, one of these variables will be the outcome variable that we will be interested to predict (e.g. TOTAL VALUE).
- Base R has several functions (such as `read.csv()`) that are able to read different types of text files.
- Although these functions are perfectly suitable for most setups, we will be mainly using the functions provided by package **readr** (Wickham and Francois, 2015b), instead.
- The functions provided by this package are more robust in some setups, are much faster, and have the added advantage of returning a tbl data frame object.

# Importing Text Files (continued)

- The following code will read the content of WestRoxbury.csv into an R data frame table:

```
library(readr)
housing.df<-read_csv("WestRoxbury.csv")
```

- The above code assumes that the csv file is located under the current working directory (if you don't know the current working directory just type `getwd()` in the console and hit enter).
- If the file is not under the current working directory or if its stored on the web, just provide the full path to the file as in the code examples below:

```
housing.df<-read_csv("C:/MyDataFiles
                      /WestRoxbury.csv")
# OR
housing.df<-read_csv("http://bit.ly/2Gqx14i")
```

# Importing Text Files (continued)

- Function `read_csv()` takes the name of the CSV file as the first argument and may include a series of other arguments that allow to fine tune the reading process. The result of the function is a data frame table object from package **dplyr**.
- Some countries use the comma character as decimal separator in real numbers. In these countries the CSV format uses the semi-colon as the values separator character. Function `read_csv2()` works exactly as the above function, but assumes the semi-colon as the values separator instead of the comma, which is used as decimal separator.
- Some files simply use spaces as separators between the values. Suppose we have a file named z.txt with the following content:

| ID | Name | Age | Phone |
|---|---|---|---|
| 23424 | Ana | 40 | ??? |
| 11234 | Charles | 12 | 34567678 |
| 77654 | Susanne | 45 | 23435567 |

# Importing Text Files (continued)

- The content of this file could be read as follows:

```
library(readr)
 d <- read_delim("z.txt", delim=" "
                  col_names = TRUE, na="???")
```

- In addition to the file name, `read_delim()` requires 3 other commonly used parameters:
  - `delim` - allows you to indicate the character used as separator of the column values.
  - `col_names` - a Boolean indicating whether the first line contains the column names.
  - `na` - provides a vector of strings that are to be taken as missing values.
- The result of all these functions is a **dplyr** data frame table.

# Importing Text Files: Precautions and Other Packages

- A word of warning on reading text files. A frequent source of frustrating errors is character encoding. Different countries or event different operating systems may be using different character encoding and reading files created on these other environments may lead to some strange results. The above functions also include facilities to handle such situations through parameter `locale` that you may wish to explore in the future.
- The functions of package **readr** are considerably faster than the equivalent functions of base R. Still, for extremely large files you may consider the alternative provided by function `fread()` from package **data.table** (Dowle et al., 2015), or function `read.csv.raw()` from package **iotools** (Urbanek and Arnold, 2015).

# Importing Spreadsheets

- There are several ways of importing data from spreadsheets, in particular from the common Excel spreadsheets. We are going to illustrate a few of the simplest.
- Most spreadsheets (all?) should easily allow to export some data table into a CSV text file, or even other text formats. This means that we could first export the data from the spreadsheet into a text file and then use one of the procedures described in previous slides.
- When saving Excel files as csv, beware of two things:
  1. Opening a .csv file in Excel strips off leading 0's, which may corrupt data such as zip codes.
  2. Saving a .csv file in Excel saves only the digits that are displayed. If you need precision to certain number of decimals, ensure that they are displayed before saving

# Importing Spreadsheets (continued)

- When the data tables are small there is a very simple form of getting the data from the spreadsheet into an R data frame.
- Open your spreadsheet and select the data range you want to import.
- Copy this range into the clipboard (for instance on Excel in Windows by doing `Ctrl+C`). Now go to R and type the following:

```
d <- read.table("clipboard", header=TRUE)
```

- Function `read.table()` is one of the base R functions for reading text files.
- Option "clipboard" is specific to `read.table()` and can't be used in conjunction with functions of package **readr**
- In the above example we are assuming that we have selected and copied a range that included the column names in the first row and thus the `header=TRUE` argument.

# Importing Spreadsheets (continued)

- For larger data tables stored in Excel the usage of the clipboard may not be so convenient. For such situations the package **readxl** (Wickham, 2015a) contains a function named `read_excel()` that is very handy.

```
library(readxl)
 housing.df<-read_excel("WestRoxbury.xlsx",
                        sheet = 1)
 class(housing.df)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

# Importing Data from Other Formats

- R includes many other packages and functions that may help you in getting your data into an R data frame.
- The manual "R Data Import/Export" that comes with any R installation is a good starting point if you are searching for some specific format.
- Formats used in other statistical and data mining software packages can be read using the functions provided by package **foreign** (R Core Team, 2015a).
- As usual, searching the Web is most probably the fastest way of getting an answer on how to read some strange data format using R.

# RStudio's Data Import Utility

- Sometimes you may be interested in a non-programmatic, manual data import.
- If you are using RStudio, its native data import utility may be very convenient for those instances.
- Go to `File -> Import Dataset`, select the format of the data that you will be importing from, and select the file.
- Upon selection, RStudio will open a interface that allows you to preview the data that you are importing and tweak different presets.
- The `Import Dataset` interface also shows you the R code it executes in order to read the dataset into R. This feature is very handy if you want to fine-tune your data import.

# Initial Look at West Roxbury Data

- The data "dictionary" describing each variable can be found in the WestRoxburyDataDictionary.csv and is self explanatory. Feel free to take a look at it using a spreadsheet software or load it and examine it from within R.

```
library(readr)
datDict<-read_csv("WestRoxburyDataDictionary.csv")
View(datDict)
```

- You can also take a look at the top few rows of the West Roxbury housing file.

```
head(housing.df)
```

- To display the dimensions of the data set run the following code:

```
dim(housing.df)
```

```
## [1] 5802   14
```

- As mentioned before, the Roxbury.csv has records on 5802 homes and each record contain 14 house descriptors.

# Initial Look at West Roxbury Data

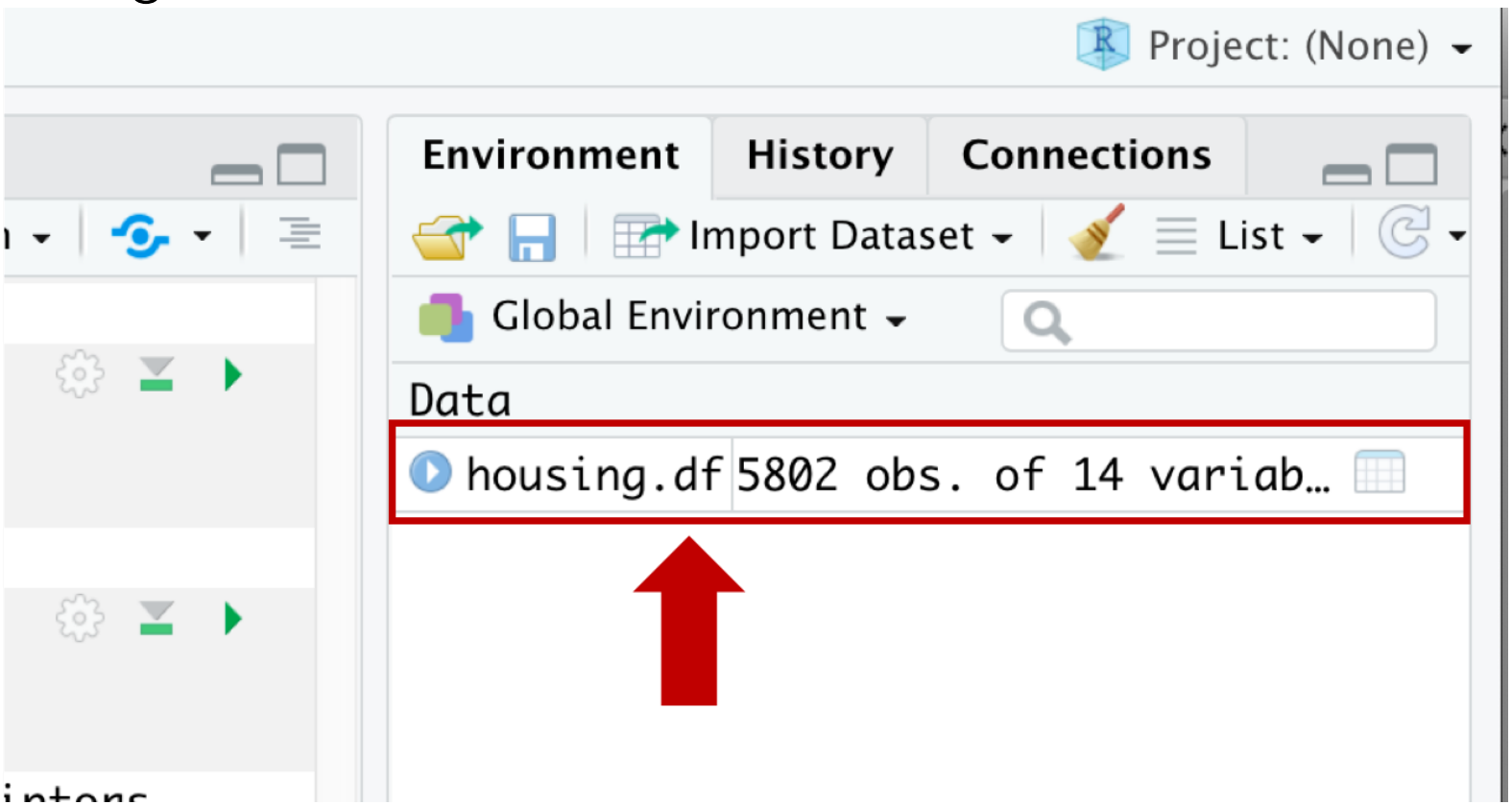
- To check whether those descriptors are the same as the ones listed in the data dictionary, execute the following code:

```
t(t(names(housing.df))) # t(t()) is for nicer display
```

```
##       [,1]
##  [1,] "TOTAL_VALUE"
##  [2,] "TAX"
##  [3,] "LOT_SQFT"
##  [4,] "YR_BUILT"
##  [5,] "GROSS_AREA"
##  [6,] "LIVING_AREA"
##  [7,] "FLOORS"
##  [8,] "ROOMS"
##  [9,] "BEDROOMS"
## [10,] "FULL_BATH"
## [11,] "HALF_BATH"
## [12,] "KITCHEN"
## [13,] "FIREPLACE"
## [14,] "REMODEL"
```

# Initial Look at West Roxbury Data

- Sometimes, you may need to view and browse the whole dataset as you would do in Microsoft Excel.
- Clicking the variable name in the `Environment` pane of RStudio will open up the data browser and allow you to browse through the rows and columns.



# Initial Look at West Roxbury Data

- Alternatively, executing the `View(variable name)` command in the RStudio console will lead to the same outcome:

```
View(housing.df)
```