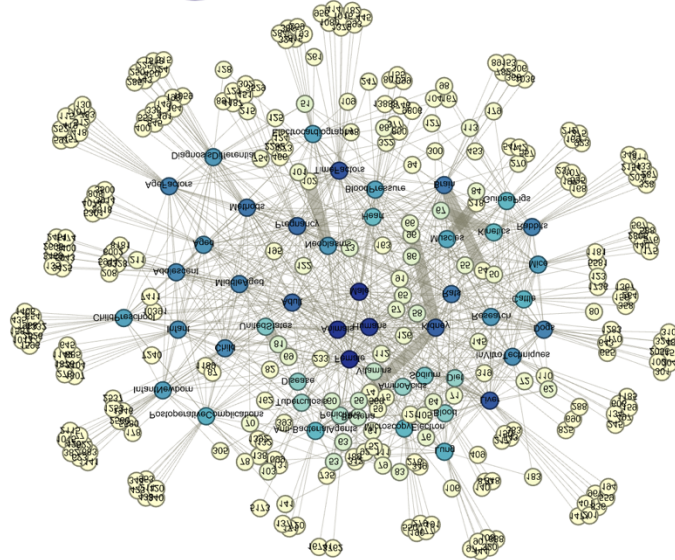

Data Mining With Python and R



Data Mining With Python and R

Wenqiang Feng and Ming Chen

April 27, 2019

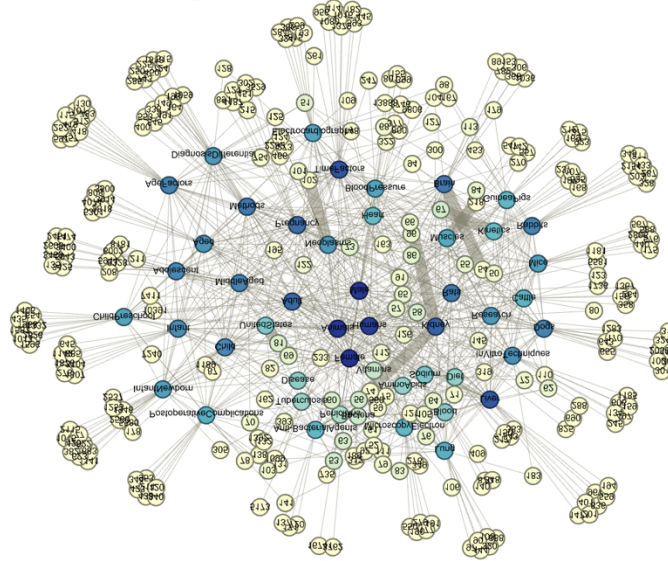
CONTENTS

1	Preface	3
1.1	About this tutorial	3
1.1.1	About the authors	3
1.2	Motivation for this tutorial	4
1.3	Copyright notice and license info	4
1.4	Acknowledgement	4
1.5	Feedback and suggestions	5
2	Python or R for data analysis?	7
2.1	Ponder over questions	7
2.2	Comparison List	7
2.3	My Opinions	8
3	Getting Started	9
3.1	Installing programming language	9
3.2	Installing programming platform	10
3.3	Installing packages	10
4	Data Exploration	15
4.1	Procedures	15
4.2	Datasets in this Tutorial	15
4.3	Loading Datasets	15
4.3.1	Loading table format database	16
4.3.2	Loading data from .csv	17
4.3.3	Loading data from .xlsx	17
4.4	Audit Data	18
4.4.1	Check missing rate	18
4.4.2	Checking zero variance features	19
4.5	Understand Data With Statistics methods	20
4.5.1	Summary of the data	20
4.5.2	The size of the data	22
4.5.3	Data type of the features	23
4.5.4	The column names	24
4.5.5	The first or last parts of the data	24
4.5.6	Correlation Matrix	25

4.5.7	Covariance Matrix	27
4.6	Understand Data With Visualization	29
4.6.1	Summary plot of data in figure	29
4.6.2	Histogram of the quantitative predictors	29
4.6.3	Boxplot of the quantitative predictors	33
4.6.4	Correlation Matrix plot of the quantitative predictors	37
4.7	Source Code for This Section	38
5	Data Manipulation	43
5.1	Combining DataFrame	43
5.1.1	Mutating Joins	43
5.1.2	Filtering Joins	48
5.2	DataFrame Operations	48
6	Pre-processing procedures	49
6.1	Rough Pre-processing	49
6.2	Source Code for This Section	49
7	Summary of Data Mining Algorithms	55
7.1	Diagram of Data Mining Algorithms	55
7.2	Categories of Data Mining Algorithms	55
8	Dimension Reduction Algorithms	59
8.1	What is dimension reduction?	59
8.2	Singular Value Decomposition (SVD)	59
8.3	Principal Component Analysis (PCA)	60
8.4	Independent Component Analysis (ICA)	61
8.5	Nonnegative matrix factorization (NMF)	61
9	Regression Algorithm	63
9.1	Introduction	63
9.2	Ordinary Least Squares Regression (OLSR)	63
9.2.1	How to solve it?	63
9.2.2	Ordinary Least Squares	65
9.3	Linear Regression (LR)	65
10	Classification Algorithms	67
10.1	Logistic Regression (LR)	67
10.2	k-Nearest Neighbour (kNN)	67
10.3	Linear Discriminant Analysis (LDA)	67
10.4	Quadratic Discriminant Analysis (QDA)	67
11	Regularization Algorithms	69
11.1	Subset Selection (SubS)	69
11.2	Ridge Regression (Ridge)	69
11.3	Least Absolute Shrinkage and Selection Operator (LASSO)	69
12	Resampling Algorithms	71

13	Developing Your Own R Packages	73
14	Developing Your Own Python Packages	75
14.1	Hierarchical Structure	75
14.2	Set Up	75
14.3	Requirements	76
14.4	ReadMe	76
	Index	79

Data Mining With Python and R



Welcome to my **Data Mining With Python and R** tutorials! In these tutorials, you will learn a wide array of concepts about Python and R programming in Data Mining. The PDF version can be downloaded from [HERE](#).

PREFACE

1.1 About this tutorial

This document is an enhanced extension of my Data Mining Methods & Application (STAT 577) course in University of Tennessee at Knoxville. **You may download and distribute it. Please be aware, however, that the note contains typos as well as inaccurate or incorrect description. Please give the original author corresponding credit by using thank you email or citations.** If you find your work wasn't cited in this note, please feel free to let me know.

Although I am by no means an data mining programming expert, I decided that it would be useful for me to share what I learned about data mining programming in the form of easy tutorials with detailed example. I hope those tutorials will be a valuable tool for your studies.

The tutorials assume that the reader has a preliminary knowledge of programming and unix. And this document is generated automatically by using [sphinx](#).

1.1.1 About the authors

- **Wenqiang Feng**
 - Data Scientist and PhD in Mathematics
 - University of Tennessee at Knoxville
 - Email: von198@gmail.com
- **Ming Chen**
 - Data Scientist and PhD in Genome Science and Technology
 - University of Tennessee at Knoxville
 - Email: ming.chen0919@gmail.com

- **Biography**

Wenqiang Feng is Data Scientist within DST's Applied Analytics Group. Dr. Feng's responsibilities include providing DST clients with access to cutting-edge skills and technologies, including Big Data analytic solutions, advanced analytic and data enhancement techniques and modeling.

Dr. Feng has deep analytic expertise in data mining, analytic systems, machine learning algorithms, business intelligence, and applying Big Data tools to strategically solve industry problems in a cross-functional business. Before joining DST, Dr. Feng was an IMA Data Science Fellow at The Institute for Mathematics and its Applications (IMA) at the University of Minnesota. While there, he helped startup companies make marketing decisions based on deep predictive analytics.

Dr. Feng graduated from University of Tennessee, Knoxville, with Ph.D. in Computational Mathematics and Master's degree in Statistics. He also holds Master's degree in Computational Mathematics from Missouri University of Science and Technology (MST) and Master's degree in Applied Mathematics from the University of Science and Technology of China (USTC).

- **Declaration**

The work of Wenqiang Feng was supported by the IMA, while working at IMA. However, any opinion, finding, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the IMA, UTK and DST.

1.2 Motivation for this tutorial

Data mining is a relatively new, while the technology is not. Here are the several main motivation for this tutorial:

1. It is no exaggeration to say that data mining has thunderstorms impacted on our real lives. I have great interest in data mining and am eager to learn those technologies.
2. Fortunately, I had a chance to register Dr. Haileab Hilafu's Data Mining Methods & Application class. Dr. Haileab Hilafu and his class inspired me to do a better job.
3. However, I still found that learning data mining programming was a difficult process. I have to Google it and identify which one is true. It was hard to find detailed examples which I can easily learn the full process in one file.
4. Good sources are expensive for a graduate student.

1.3 Copyright notice and license info

This [Data Mining With Python and R](#) PDF file is supposed to be a free and living document, which is why its source is available online at [Data Mining With Python and R at Github](#). But this document is licensed according to both [MIT License](#) and [Creative Commons Attribution-NonCommercial 2.0 Generic \(CC BY-NC 2.0\) License](#).

When you plan to use, copy, modify, merge, publish, distribute or sublicense, Please see the terms of those licenses for more details and give the corresponding credits to the author.

1.4 Acknowledgement

At here, I would like to thank Dr. Haileab Hilafu for providing some of his R code and homework solutions. I also would like to thank Bo Gao, Le Yin, Chen Wen, Jian Sun and Huan Chen for the valuable discussion

and thank the generous anonymous authors for providing the detailed solutions and source code on the Internet. Without those help, those tutorials would not have been possible to be made. In those tutorials, I try to use the detailed demo code to show how to use each functions in R and Python to do data mining.

1.5 Feedback and suggestions

Your comments and suggestions are highly appreciated. I am more than happy to receive corrections, suggestions or feedbacks through email (Wenqiang Feng: von198@gmail.com) for improvements.

PYTHON OR R FOR DATA ANALYSIS?

Note: Sharpening the knife longer can make it easier to hack the firewood – old Chinese proverb

There is an old Chinese proverb that Says ‘sharpening the knife longer can make it easier to hack the firewood’. In other words, take extra time to get it right in the preparation phase and then the work will be easier. So it is worth to take several minites to think about which programming language is better for you.

When you google it, you will get many useful results. Here are some valueable information from [Quora](#):

2.1 Ponder over questions

- Six questions to ponder over from [Vipin Tyagi at Quora](#)
 1. Is your problem is purely data analysis based or mixed one involving mathematics, machine-learning, artificial intelligence based?
 2. What are the commonly used tools in your field?
 3. What is the programming expertise of your human resources?
 4. What level of visualization you require in your presentations?
 5. Are you academic, research-oriented or commercial professional?
 6. Do you have access to number of data analytic softwares for doing your assignment?

2.2 Comparison List

- comparative list from [Yassine Alouini at Quora](#)

	R	Python
advantages	<ul style="list-style-type: none">• great for prototyping• great for statistical analysis• nice IDE	<ul style="list-style-type: none">• great for scripting and automating your different data mining pipelines• integrates easily in a production workflow• can be used across different parts of your software engineering team• scikit-learn library is awesome for machine-learning tasks.• Ipython is also a powerful tool for exploratory analysis and presentations
disadvantages	<ul style="list-style-type: none">• syntax could be obscure• libraries documentation isn't always user friendly• harder to integrate to a production workflow.	<ul style="list-style-type: none">• It isn't as thorough for statistical analysis as R• learning curve is steeper than R, since you can do much more with Python

2.3 My Opinions

In my opinion, if you want to be a decent Data Analyst or Data Scientist, you should learn both – **R** and **Python**. Since they are open-source softwares (open-source is always good in my eyes) and are free to download. If you are a beginner without any programming experience and only want to do some data analysis, I would definitely suggest to use **R**. Otherwise, I would suggest to use both.

GETTING STARTED

Note: Good tools are prerequisite to the successful execution of a job – old Chinese proverb

Let's keep sharpening our tools. A good programming platform can save you lots of troubles and time. Herein I will only present how to install my favorite programming platform for R and Python and only show the easiest way which I know to install them on Linux system. If you want to install on the other operator system, you can Google it. In this section, you may learn how to install R, Python and the corresponding programming platform and package.

3.1 Installing programming language

Python

Go to Ubuntu Software Center and follow the following steps:

1. Open Ubuntu Software Center
2. Search for python
3. And click Install

Or Open your terminal and using the following command:

```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev
libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev
sudo apt-get install python
sudo easy_install pip
sudo pip install ipython
```

R

Go to Ubuntu Software Center and follow the following steps:

1. Open Ubuntu Software Center
2. Search for r-base
3. And click Install

Or Open your terminal and using the following command:

```
sudo apt-get update
sudo apt-get install r-base
```

3.2 Installing programming platform

My favorite programming platform for R is definitely *RStudio* IDE and for Python is *PyCharm*.

Python

- **Installing PyCharm**

Go to Ubuntu Software Center and follow the following steps:

1. Open Ubuntu Software Center
2. Search for Eclipse
3. And click Install

Here is the video tutorial for installing Pydev for Eclipse on Youtube: [Pydev on Youtube](#)

R

- **Installing RStudio**

Go to Ubuntu Software Center and follow the following steps:

1. Open Ubuntu Software Center
2. Search for RStudio
3. And click Install

3.3 Installing packages

Python

- **Installing package for Python**

Install package or modules for Python in Linux can also be quite easy. Here I will only present installation by using pip.

- **Installing pip**

```
sudo easy_install pip
```

- **Installing numpy**

```
pip install numpy
```

- **Installing pandas**


```
pip install pandas
```

- **Installing scikits-learn**

```
pip install -U scikit-learn
```

The following are the best Python modules for data mining from [kdnuggets](#), you may also want to install all of them.

1. Basics

- **numpy** - numerical library, <http://numpy.scipy.org/>
- **scipy** - Advanced math, signal processing, optimization, statistics, <http://www.scipy.org/>
- **matplotlib**, python plotting - Matplotlib, <http://matplotlib.org>

2. Machine Learning and Data Mining

- **MDP**, a collection of supervised and unsupervised learning algorithms, <http://pypi.python.org/pypi/MDP/2.4>
- **mlpy**, Machine Learning Python, <http://mlpy.sourceforge.net>
- **NetworkX**, for graph analysis, <http://networkx.lanl.gov/>
- **Orange**, Data Mining Fruitful & Fun, <http://biolab.si>
- **pandas**, Python Data Analysis Library, <http://pandas.pydata.org>
- **pybrain**, <http://pybrain.org>
- **scikits-learn** - Classic machine learning algorithms - Provide simple an efficient solutions to learning problems, <http://scikit-learn.org/stable/>

3. Natural Language

- **NLTK**, Natural Language Toolkit, <http://nltk.org>

4. For web scraping

- **Scrapy**, An open source web scraping framework for Python, <http://scrapy.org>
- **urllib/urllib2**

Herein I would like to add one more important package **Theano** for deep learning and **textmining** for text mining:

- **Theano**, deep learning, <http://deeplearning.net/tutorial/>
- **textmining**, text mining, <https://pypi.python.org/pypi/textmining/1.0>

R

- **Installing package for R**

Install package for R in RStudio os super easy, I will use tree package as a example:

```
install.packages("tree")
```

The following are the top 20 R machine learning and data science packages from [Bhavya Geethika](#), you may want to install all of them.

- **e1071** Functions for latent class analysis, short time Fourier transform, fuzzy clustering, support vector machines, shortest path computation, bagged clustering, naive Bayes classifier etc (142479 downloads)
- **rpart** Recursive Partitioning and Regression Trees. (135390)
- **igraph** A collection of network analysis tools. (122930)
- **nnet** Feed-forward Neural Networks and Multinomial Log-Linear Models. (108298)
- **randomForest** Breiman and Cutler's random forests for classification and regression. (105375)
- **caret** package (short for Classification And REgression Training) is a set of functions that attempt to streamline the process for creating predictive models. (87151)
- **kernlab** Kernel-based Machine Learning Lab. (62064)
- **glmnet** Lasso and elastic-net regularized generalized linear models. (56948)
- **ROCR** Visualizing the performance of scoring classifiers. (51323)
- **gbm** Generalized Boosted Regression Models. (44760)
- **party** A Laboratory for Recursive Partitioning. (43290)
- **arules** Mining Association Rules and Frequent Itemsets. (39654)
- **tree** Classification and regression trees. (27882)
- **klaR** Classification and visualization. (27828)
- **RWeka** R/Weka interface. (26973)
- **ipred** Improved Predictors. (22358)
- **lars** Least Angle Regression, Lasso and Forward Stagewise. (19691)
- **earth** Multivariate Adaptive Regression Spline Models. (15901)
- **CORElearn** Classification, regression, feature evaluation and ordinal evaluation. (13856)
- **mboost** Model-Based Boosting. (13078)

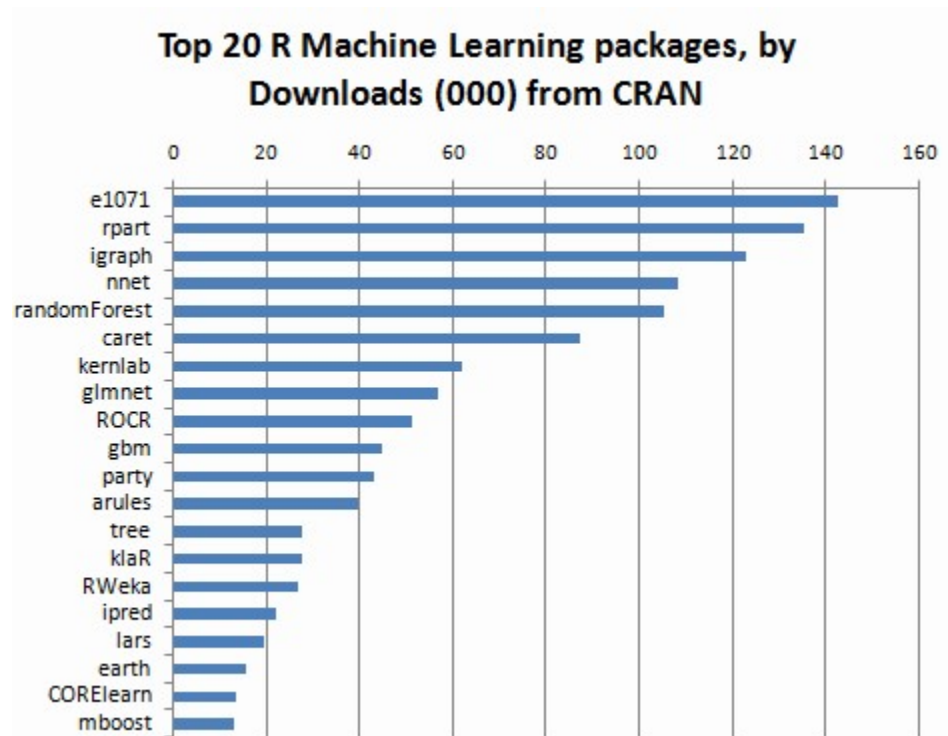


Fig. 1: Top 20 R Machine Learning and Data Science packages. From <http://www.kdnuggets.com/2015/06/top-20-r-machine-learning-packages.html>

DATA EXPLORATION

Note: **Know yourself and know your enemy, and you will never be defeated** – idiom, from Sunzi’s Art of War

4.1 Procedures

Data mining is a complex process that aims to discover patterns in large data sets starting from a collection of existing data. In my opinion, data mining contains four main steps:

1. **Collecting data:** This is a complex step, I will assume we have already gotten the datasets.
2. **Pre-processing:** In this step, we need to try to understand your data, denoise, do dimensionality reduction and select proper predictors etc.
3. **Feeding data mining:** In this step, we need to use your data to feed your model.
4. **Post-processing :** In this step, we need to interpret and evaluate your model.

In this section, we will try to know our enemy – datasets. We will learn how to load data, how to understand data with statistics method and how to understand data with visualization. Next, we will start with Loading Datasets for the Pre-processing.

4.2 Datasets in this Tutorial

The datasets for this tutorial are available to download: [Heart](#), [Energy Efficiency](#). Those data are from my course materials, the copyrights belong to the original authors.

4.3 Loading Datasets

There are three main data source database, *.csv and *.xlsx. We will show how to load those two types of data in **R** and **Python**, respectively.

4.3.1 Loading table format database

User and Database information:

```
user = '*****'  
pw='*****'  
host = '***.***.***.***'  
database = '**'  
table_name = '***'
```

Python

- Loading data from database in **Python**

```
# import library  
import psycopg2  
import pandas as pd  
  
# Create the database connection  
conn = psycopg2.connect(host=host, database=database,  
                        user=user, password=pw)  
cur = conn.cursor()  
  
# Create the SQL query string.  
sql = """  
    SELECT *  
    FROM {table_name}  
    """.format(table_name=table_name)  
df = pd.read_sql(sql, conn)  
  
df.head(4)
```

R

- Loading data from database in **R**

```
# load the library  
library("sqldf")  
library('RODBC')  
library('RPostgreSQL')  
  
# Create a driver  
drv <- DBI::dbDriver( "PostgreSQL" )  
# Create the database connection  
conn <- dbConnect( drv, dbname = database, host = host, port = '5432',  
                  user = user, password = pw )  
  
# Create the SQL query string. Include a semi-colon to terminate  
querystring = sprintf('SELECT * FROM %s;', table_name)  
# Execute the query and return results as a data frame  
df = dbGetQuery(conn, querystring )  
  
head(df)
```

4.3.2 Loading data from .csv

Python

- Loading data from .csv in **Python**

```
import pandas as pd

# set data path
path = '~/Dropbox/MachineLearningAlgorithms/python_code/data/Heart.csv'

# read data set
rawdata = pd.read_csv(path)
```

R

- Loading data from .csv in **R**

```
# set the path or environment
setwd("/home/feng/R-language/sat577/HW#4/data")

# read data set
rawdata = read.csv("spam.csv")
```

4.3.3 Loading data from .xlsx

Python

- Loading data from .xlsx in **Python**

```
import pandas as pd

# set data path
path = ('/home/feng/Dropbox/MachineLearningAlgorithms/python_code/data/'
'energy_efficiency.xlsx')

# read data set from first sheet
rawdata= pd.read_excel(path, sheetname=0)
```

R

- Loading data from .xlsx in **R**

```
# set the path or environment
setwd("~/Dropbox/R-language/sat577/")

#install.packages("readxl") # CRAN version
library(readxl)

# read data set
energy_eff=read_excel("energy_efficiency.xlsx")
```

4.4 Audit Data

In my opinion, data audit is the first step you need to do when you get your dataset. Since you need to know whether the data quality is good enough or not.

4.4.1 Check missing rate

Python

- Checking missing rate in **Python**

```
import pandas as pd

d = {'A': [1, 2, None, 3],
      'B': [None, None, 4, 5],
      'C': [None, 'b', 'c', 'd']}

# create DataFrame
df = pd.DataFrame(d)
print(df)

# define the missing rate function
def missing_rate(df_in):
    # calculate missing rate and transpose the DataFrame
    rate = df_in.isnull().sum() / df_in.shape[0]
    # rename the column
    rate = pd.DataFrame(rate).reset_index() \
        .rename(columns={'index': 'feature', 0: 'missing_
→rate'})
    print(rate)

missing_rate(df)
```

The results:

	A	B	C
0	1.0	NaN	None
1	2.0	NaN	b
2	NaN	4.0	c
3	3.0	5.0	d

	feature	missing_rate
0	A	0.25
1	B	0.50
2	C	0.25

R

- Checking missing rate in **R**


```
# create DataFrame
x = data.frame(A = c(1, 2, NA, 3), B = c(NA, NA, 4, 5), C = c(NA, 'b', 'c', 'd'
→))

# loading library
library('dplyr')
#library('tidyverse')

# define the missing rate function
missing_rate <- function(df){
  # calculate missing rate and transpose the DataFrame
  rate <-t( df %>% summarize_all(funs(sum(is.na(.)) / length(.))))
  # rename the column
  colnames(rate)[1] <- "missing_rate"
  print(rate)
}

x

missing_rate(x)
```

The results:

```
> x
  A B  C
1 1 NA <NA>
2 2 NA  b
3 NA 4  c
4 3 5  d
> missing_rate(x)
missing_rate
A          0.25
B          0.50
C          0.25
```

4.4.2 Checking zero variance features

Python

- Checking zero variance features in **Python**

```
import pandas as pd

d = {'A': [1, 2, 3, 3],
      'B': [1, 1, 1, 1],
      'C': ['a', 'b', 'c', 'd']}

# create DataFrame
df = pd.DataFrame(d)
print(df)
```

(continues on next page)

(continued from previous page)

```
def zero_variance(df_in):  
    counts = df_in.nunique()  
    counts = pd.DataFrame(counts)\  
        .reset_index().rename(columns={'index': 'feature', 0: 'count'})  
    return list(counts[counts['count'] == 1]['feature'])  
  
print(zero_variance(df))
```

```
   A  B  C  
0  1  1  a  
1  2  1  b  
2  3  1  c  
3  3  1  d  
['B']
```

R

- Checking zero variance features in R

```
df = data.frame(A = c(1, 2, 3, 3), B = c(1, 1, 1, 1), C = c('a', 'b', 'c', 'd'  
→'))  
  
zero_variance <- function(df){  
  compData <- data.frame(feature= c(NA), count= c(NA))  
  for(i in 1:ncol(df))  
  {  
    compData[i, ] <- c(colnames(df)[i], length(unique(df[,i])))  
  }  
  return(compData[compData$count==1,]$feature)  
}
```

```
> zero_variance(df)  
[1] "B"
```

4.5 Understand Data With Statistics methods

After we get the data in hand, then we can try to understand them. I will use “Heart.csv” dataset as a example to demonstrate how to use those statistics methods.

4.5.1 Summary of the data

It is always good to have a glance over the summary of the data. Since from the summary you will know some statistics features of your data, and you will also know whether you data contains missing data or not.

Python

- Summary of the data in **Python**

```
print("> data summary")
print rawdata.describe()
```

Then you will get

```
> data summary
      Age      Sex  RestBP      Chol      Fbs  RestECG_
count 303.000000 303.000000 303.000000 303.000000 303.000000 303.000000
mean  54.438944  0.679868 131.689769 246.693069  0.148515  0.990099
std    9.038662  0.467299 17.599748  51.776918  0.356198  0.994971
min   29.000000  0.000000  94.000000 126.000000  0.000000  0.000000
25%   48.000000  0.000000 120.000000 211.000000  0.000000  0.000000
50%   56.000000  1.000000 130.000000 241.000000  0.000000  1.000000
75%   61.000000  1.000000 140.000000 275.000000  0.000000  2.000000
max   77.000000  1.000000 200.000000 564.000000  1.000000  2.000000

      MaxHR      ExAng      Oldpeak      Slope      Ca
count 303.000000 303.000000 303.000000 303.000000 299.000000
mean 149.607261  0.326733  1.039604  1.600660  0.672241
std  22.875003  0.469794  1.161075  0.616226  0.937438
min   71.000000  0.000000  0.000000  1.000000  0.000000
25%  133.500000  0.000000  0.000000  1.000000  0.000000
50%  153.000000  0.000000  0.800000  2.000000  0.000000
75%  166.000000  1.000000  1.600000  2.000000  1.000000
max  202.000000  1.000000  6.200000  3.000000  3.000000
```

R

- Summary of the data in **R**

```
summary(rawdata)
```

Then you will get

```
> summary(rawdata)
      Age      Sex      ChestPain      RestBP
Min.   :29.00  Min.   :0.0000  asymptomatic:144  Min.   : 94.0
1st Qu.:48.00  1st Qu.:0.0000  nonanginal   : 86  1st Qu.:120.0
Median :56.00  Median :1.0000  nontypical   : 50  Median :130.0
Mean   :54.44  Mean   :0.6799  typical      : 23  Mean   :131.7
3rd Qu.:61.00  3rd Qu.:1.0000                      3rd Qu.:140.0
Max.   :77.00  Max.   :1.0000                      Max.   :200.0

      Chol      Fbs      RestECG      MaxHR
Min.   :126.0  Min.   :0.0000  Min.   :0.0000  Min.   : 71.0
1st Qu.:211.0  1st Qu.:0.0000  1st Qu.:0.0000  1st Qu.:133.5
Median :241.0  Median :0.0000  Median :1.0000  Median :153.0
Mean   :246.7  Mean   :0.1485  Mean   :0.9901  Mean   :149.6
3rd Qu.:275.0  3rd Qu.:0.0000  3rd Qu.:2.0000  3rd Qu.:166.0
Max.   :564.0  Max.   :1.0000  Max.   :2.0000  Max.   :202.0
```

(continues on next page)

(continued from previous page)

```

      ExAng      Oldpeak      Slope      Ca
Min.   :0.0000   Min.   :0.00   Min.   :1.000   Min.   :0.0000
1st Qu.:0.0000   1st Qu.:0.00   1st Qu.:1.000   1st Qu.:0.0000
Median :0.0000   Median :0.80   Median :2.000   Median :0.0000
Mean   :0.3267   Mean   :1.04   Mean   :1.601   Mean   :0.6722
3rd Qu.:1.0000   3rd Qu.:1.60   3rd Qu.:2.000   3rd Qu.:1.0000
Max.   :1.0000   Max.   :6.20   Max.   :3.000   Max.   :3.0000
                                     NA's   :4

      Thal      AHD
fixed   : 18   No :164
normal  :166   Yes:139
reversable:117
NA's    : 2

```

4.5.2 The size of the data

Most of time, we also need to know the size or dimension of our data. Such as when you need to extract the response from the dataset, you need the number of column, or when you try to split your data into train and test data set, you need know the number of row.

Python

- Checking size in **Python**

```

nrow, ncol = rawdata.shape
print nrow, ncol

```

or you can use the follwing code

```

nrow=rawdata.shape[0] #gives number of row count
ncol=rawdata.shape[1] #gives number of col count
print(nrow, ncol)

```

Then you will get

```

Raw data size
303 14

```

R

- Checking size in **R**

```
dim(rawdata)
```

Or you can use the following code

```

nrow=nrow(rawdata)
ncol=ncol(rawdata)

c(nrow, ncol)

```

Then you will get

```
> dim(rawdata)
[1] 303 14
```

4.5.3 Data type of the features

Data type is also very important, since some functions or methods can not be applied to the qualitative data or some machine learning algorithm will take some types as categorical data, you need to remove those features or transform them into quantitative data.

Python

- Checking data type in **Python**

```
print(rawdata.dtypes)
```

Then you will get

```
Data Format:
Age          int64
Sex          int64
ChestPain    object
RestBP       int64
Chol         int64
Fbs          int64
RestECG      int64
MaxHR        int64
ExAng        int64
Oldpeak      float64
Slope        int64
Ca           float64
Thal         object
AHD          object
dtype: object
```

R

- Checking data format in **R**

```
# install the package
install.packages("mlbench")
library(mlbench)

sapply(rawdata, class)
```

Then you will get

```
> sapply(rawdata, class)
Age      Sex ChestPain  RestBP      Chol      Fbs  RestECG
"integer" "integer"  "factor" "integer" "integer" "integer" "integer"
```

(continues on next page)

(continued from previous page)

MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
"integer"	"integer"	"numeric"	"integer"	"integer"	"factor"	"factor"

4.5.4 The column names

Python

- Checking column names of the data in **Python**

```
colNames = rawdata.columns.tolist()

print "Column names:"
print colNames
```

Then you will get

```
Column names:
['Age', 'Sex', 'ChestPain', 'RestBP', 'Chol', 'Fbs', 'RestECG', 'MaxHR',
 'ExAng', 'Oldpeak', 'Slope', 'Ca', 'Thal', 'AHD']
```

R

- Checking column names of the data in **R**

```
colnames(rawdata)
attach(rawdata) # enable you can directly use name as features
```

Then you will get

```
> colnames(rawdata)
[1] "Age"      "Sex"      "ChestPain" "RestBP"   "Chol"
[6] "Fbs"      "RestECG"  "MaxHR"     "ExAng"    "Oldpeak"
[11] "Slope"    "Ca"       "Thal"      "AHD"
```

4.5.5 The first or last parts of the data

Python

- Checking first parts of the data in **Python**

```
print("\n Sample data:")
print(rawdata.head(6))
```

Then you will get

Sample data:										
	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak
→ \										
0	63	1	typical	145	233	1	2	150	0	2.3

(continues on next page)

(continued from previous page)

1	67	1	asymptomatic	160	286	0	2	108	1	1.5
2	67	1	asymptomatic	120	229	0	2	129	1	2.6
3	37	1	nonanginal	130	250	0	0	187	0	3.5
4	41	0	nontypical	130	204	0	2	172	0	1.4
5	56	1	nontypical	120	236	0	0	178	0	0.8

	Slope	Ca	Thal	AHD
0	3	0	fixed	No
1	2	3	normal	Yes
2	2	2	reversible	Yes
3	3	0	normal	No
4	1	0	normal	No
5	1	0	normal	No

R

- Checking first parts of the data in **R**

```
head(rawdata)
```

Then you will get

```
> head(rawdata)
  Age Sex   ChestPain RestBP Chol Fbs RestECG MaxHR ExAng Oldpeak
1  63  1    typical    145  233  1      2   150    0     2.3
2  67  1 asymptomatic    160  286  0      2   108    1     1.5
3  67  1 asymptomatic    120  229  0      2   129    1     2.6
4  37  1 nonanginal    130  250  0      0   187    0     3.5
5  41  0 nontypical    130  204  0      2   172    0     1.4
6  56  1 nontypical    120  236  0      0   178    0     0.8
  Slope Ca      Thal AHD
1     3  0    fixed  No
2     2  3    normal Yes
3     2  2 reversible Yes
4     3  0    normal  No
5     1  0    normal  No
6     1  0    normal  No
```

You can use the similar way (`tail`) to check the last part of the data, for simplicity, i will skip it.

4.5.6 Correlation Matrix

Python

- Computing correlation matrix in **Python**

```
print("\n correlation Matrix")
print(rawdata.corr())
```

Then you will get

correlation Matrix								
	Age	Sex	RestBP	Chol	Fbs	RestECG	MaxHR	\
Age	1.000000	-0.097542	0.284946	0.208950	0.118530	0.148868	-0.393806	
Sex	-0.097542	1.000000	-0.064456	-0.199915	0.047862	0.021647	-0.048663	
RestBP	0.284946	-0.064456	1.000000	0.130120	0.175340	0.146560	-0.045351	
Chol	0.208950	-0.199915	0.130120	1.000000	0.009841	0.171043	-0.003432	
Fbs	0.118530	0.047862	0.175340	0.009841	1.000000	0.069564	-0.007854	
RestECG	0.148868	0.021647	0.146560	0.171043	0.069564	1.000000	-0.083389	
MaxHR	-0.393806	-0.048663	-0.045351	-0.003432	-0.007854	-0.083389	1.000000	
ExAng	0.091661	0.146201	0.064762	0.061310	0.025665	0.084867	-0.378103	
Oldpeak	0.203805	0.102173	0.189171	0.046564	0.005747	0.114133	-0.343085	
Slope	0.161770	0.037533	0.117382	-0.004062	0.059894	0.133946	-0.385601	
Ca	0.362605	0.093185	0.098773	0.119000	0.145478	0.128343	-0.264246	

	ExAng	Oldpeak	Slope	Ca
Age	0.091661	0.203805	0.161770	0.362605
Sex	0.146201	0.102173	0.037533	0.093185
RestBP	0.064762	0.189171	0.117382	0.098773
Chol	0.061310	0.046564	-0.004062	0.119000
Fbs	0.025665	0.005747	0.059894	0.145478
RestECG	0.084867	0.114133	0.133946	0.128343
MaxHR	-0.378103	-0.343085	-0.385601	-0.264246
ExAng	1.000000	0.288223	0.257748	0.145570
Oldpeak	0.288223	1.000000	0.577537	0.295832
Slope	0.257748	0.577537	1.000000	0.110119
Ca	0.145570	0.295832	0.110119	1.000000

R

- Computing correlation matrix in R

```
# get numerical data and remove NAN
numdata=na.omit(rawdata[,c(1:2,4:12)])

# computing correlation matrix
cor(numdata)
```

Then you will get

```
> cor(numdata)
```

	Age	Sex	RestBP	Chol	Fbs
Age	1.00000000	-0.09181347	0.29069633	0.203376601	0.128675921
Sex	-0.09181347	1.00000000	-0.06552127	-0.195907357	0.045861783
RestBP	0.29069633	-0.06552127	1.00000000	0.132284171	0.177623291
Chol	0.20337660	-0.19590736	0.13228417	1.000000000	0.006664176
Fbs	0.12867592	0.04586178	0.17762329	0.006664176	1.000000000
RestECG	0.14974915	0.02643577	0.14870922	0.164957542	0.058425836
MaxHR	-0.39234176	-0.05206445	-0.04805281	0.002179081	-0.003386615
ExAng	0.09510850	0.14903849	0.06588463	0.056387955	0.011636935
Oldpeak	0.19737552	0.11023676	0.19161540	0.040430535	0.009092935
Slope	0.15895990	0.03933739	0.12110773	-0.009008239	0.053776677
Ca	0.36260453	0.09318476	0.09877326	0.119000487	0.145477522

(continues on next page)

(continued from previous page)

	RestECG	MaxHR	ExAng	Oldpeak	Slope
Age	0.14974915	-0.392341763	0.09510850	0.197375523	0.158959901
Sex	0.02643577	-0.052064447	0.14903849	0.110236756	0.039337394
RestBP	0.14870922	-0.048052805	0.06588463	0.191615405	0.121107727
Chol	0.16495754	0.002179081	0.05638795	0.040430535	-0.009008239
Fbs	0.05842584	-0.003386615	0.01163693	0.009092935	0.053776677
RestECG	1.00000000	-0.077798148	0.07408360	0.110275054	0.128907169
MaxHR	-0.07779815	1.00000000	-0.37635897	-0.341262236	-0.381348495
ExAng	0.07408360	-0.376358975	1.00000000	0.289573103	0.254302081
Oldpeak	0.11027505	-0.341262236	0.28957310	1.00000000	0.579775260
Slope	0.12890717	-0.381348495	0.25430208	0.579775260	1.00000000
Ca	0.12834265	-0.264246253	0.14556960	0.295832115	0.110119188
Ca					
Age	0.36260453				
Sex	0.09318476				
RestBP	0.09877326				
Chol	0.11900049				
Fbs	0.14547752				
RestECG	0.12834265				
MaxHR	-0.26424625				
ExAng	0.14556960				
Oldpeak	0.29583211				
Slope	0.11011919				
Ca	1.00000000				

4.5.7 Covariance Matrix

Python

- Computing covariance matrix in **Python**

```
print("\n covariance Matrix")
print(rawdata.corr())
```

Then you will get

covariance Matrix							
	Age	Sex	RestBP	Chol	Fbs	RestECG	\
Age	81.697419	-0.411995	45.328678	97.787489	0.381614	1.338797	
Sex	-0.411995	0.218368	-0.530107	-4.836994	0.007967	0.010065	
RestBP	45.328678	-0.530107	309.751120	118.573339	1.099207	2.566455	
Chol	97.787489	-4.836994	118.573339	2680.849190	0.181496	8.811521	
Fbs	0.381614	0.007967	1.099207	0.181496	0.126877	0.024654	
RestECG	1.338797	0.010065	2.566455	8.811521	0.024654	0.989968	
MaxHR	-81.423065	-0.520184	-18.258005	-4.064651	-0.063996	-1.897941	
ExAng	0.389220	0.032096	0.535473	1.491345	0.004295	0.039670	
Oldpeak	2.138850	0.055436	3.865638	2.799282	0.002377	0.131850	
Slope	0.901034	0.010808	1.273053	-0.129598	0.013147	0.082126	
Ca	3.066396	0.040964	1.639436	5.791385	0.048394	0.119706	

(continues on next page)

(continued from previous page)

	MaxHR	ExAng	Oldpeak	Slope	Ca
Age	-81.423065	0.389220	2.138850	0.901034	3.066396
Sex	-0.520184	0.032096	0.055436	0.010808	0.040964
RestBP	-18.258005	0.535473	3.865638	1.273053	1.639436
Chol	-4.064651	1.491345	2.799282	-0.129598	5.791385
Fbs	-0.063996	0.004295	0.002377	0.013147	0.048394
RestECG	-1.897941	0.039670	0.131850	0.082126	0.119706
MaxHR	523.265775	-4.063307	-9.112209	-5.435501	-5.686270
ExAng	-4.063307	0.220707	0.157216	0.074618	0.064162
Oldpeak	-9.112209	0.157216	1.348095	0.413219	0.322753
Slope	-5.435501	0.074618	0.413219	0.379735	0.063747
Ca	-5.686270	0.064162	0.322753	0.063747	0.878791

R

• Computing covariance matrix in R

```
# get numerical data and remove NAN
numdata=na.omit(rawdata[,c(1:2,4:12)])

# computing covariance matrix
cov(numdata)
```

Then you will get

```
> cov(numdata)
```

	Age	Sex	RestBP	Chol	Fbs
Age	81.3775448	-0.388397567	46.4305852	95.2454603	0.411909946
Sex	-0.3883976	0.219905277	-0.5440170	-4.7693542	0.007631703
RestBP	46.4305852	-0.544016969	313.4906736	121.5937353	1.116001885
Chol	95.2454603	-4.769354223	121.5937353	2695.1442616	0.122769410
Fbs	0.4119099	0.007631703	1.1160019	0.1227694	0.125923099
RestECG	1.3440551	0.012334179	2.6196943	8.5204709	0.020628044
MaxHR	-81.2442706	-0.560447577	-19.5302126	2.5968104	-0.027586362
ExAng	0.4034028	0.032861215	0.5484838	1.3764001	0.001941595
Oldpeak	2.0721791	0.060162510	3.9484299	2.4427678	0.003755247
Slope	0.8855132	0.011391439	1.3241566	-0.2887926	0.011784247
Ca	3.0663958	0.040964288	1.6394357	5.7913852	0.048393975

	RestECG	MaxHR	ExAng	Oldpeak	Slope
Age	1.34405513	-81.24427061	0.403402842	2.072179076	0.88551323
Sex	0.01233418	-0.56044758	0.032861215	0.060162510	0.01139144
RestBP	2.61969428	-19.53021257	0.548483760	3.948429889	1.32415658
Chol	8.52047092	2.59681040	1.376400081	2.442767839	-0.28879262
Fbs	0.02062804	-0.02758636	0.001941595	0.003755247	0.01178425
RestECG	0.98992166	-1.77682880	0.034656910	0.127690736	0.07920136
MaxHR	-1.77682880	526.92866602	-4.062052479	-9.116871675	-5.40571480
ExAng	0.03465691	-4.06205248	0.221072479	0.158455478	0.07383673
Oldpeak	0.12769074	-9.11687168	0.158455478	1.354451303	0.41667415
Slope	0.07920136	-5.40571480	0.073836726	0.416674149	0.38133824
Ca	0.11970551	-5.68626967	0.064162421	0.322752576	0.06374717

	Ca
Age	3.06639582

(continues on next page)

(continued from previous page)

Sex	0.04096429
RestBP	1.63943570
Chol	5.79138515
Fbs	0.04839398
RestECG	0.11970551
MaxHR	-5.68626967
ExAng	0.06416242
Oldpeak	0.32275258
Slope	0.06374717
Ca	0.87879060

4.6 Understand Data With Visualization

A picture is worth a thousand words. You will see the powerful impact of the figures in this section.

4.6.1 Summary plot of data in figure

Python

- Summary plot in **Python**

```
# plot of the summary
plot(rawdata)
```

Then you will get Figure *Summary plot of the data with Python*.

R

- Summary plot in **R**

```
# plot of the summary
plot(rawdata)
```

Then you will get Figure *Summary plot of the data with R*.

4.6.2 Histogram of the quantitative predictors

Python

- Histogram in **Python**

```
# Histogram
rawdata.hist()
plt.show()
```

Then you will get Figure *Histogram in Python*.

R

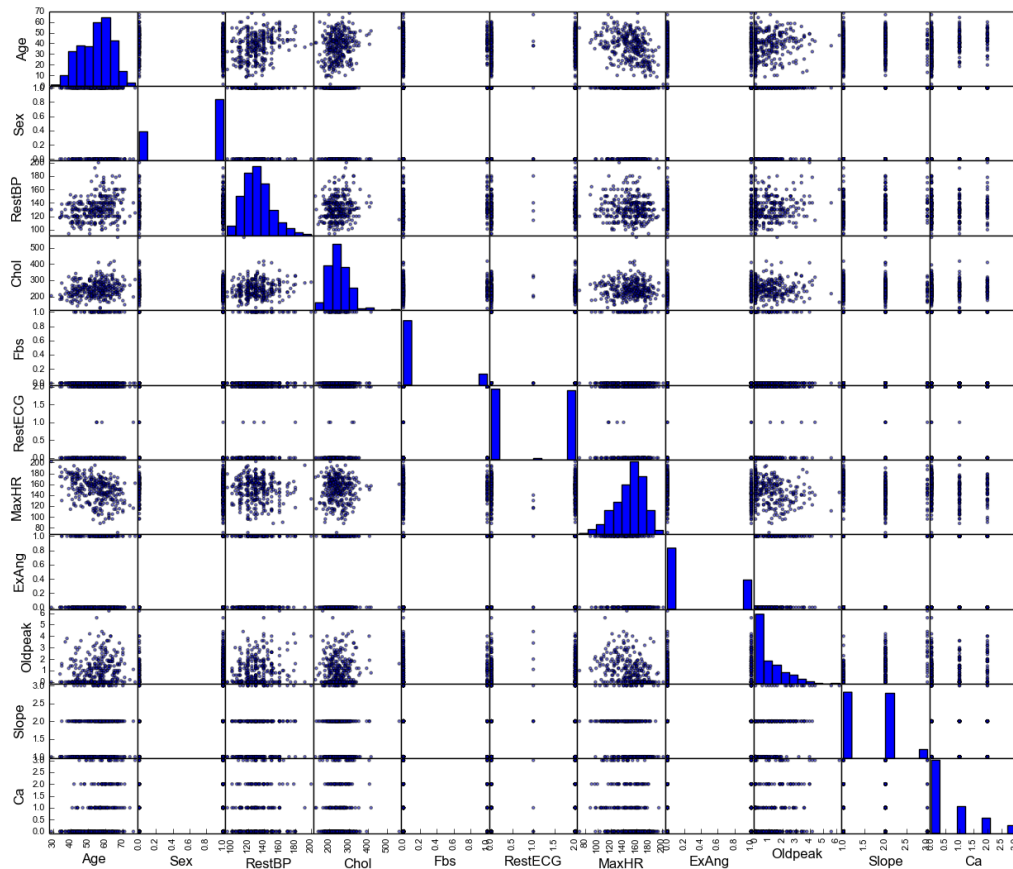


Fig. 1: Summary plot of the data with Python.

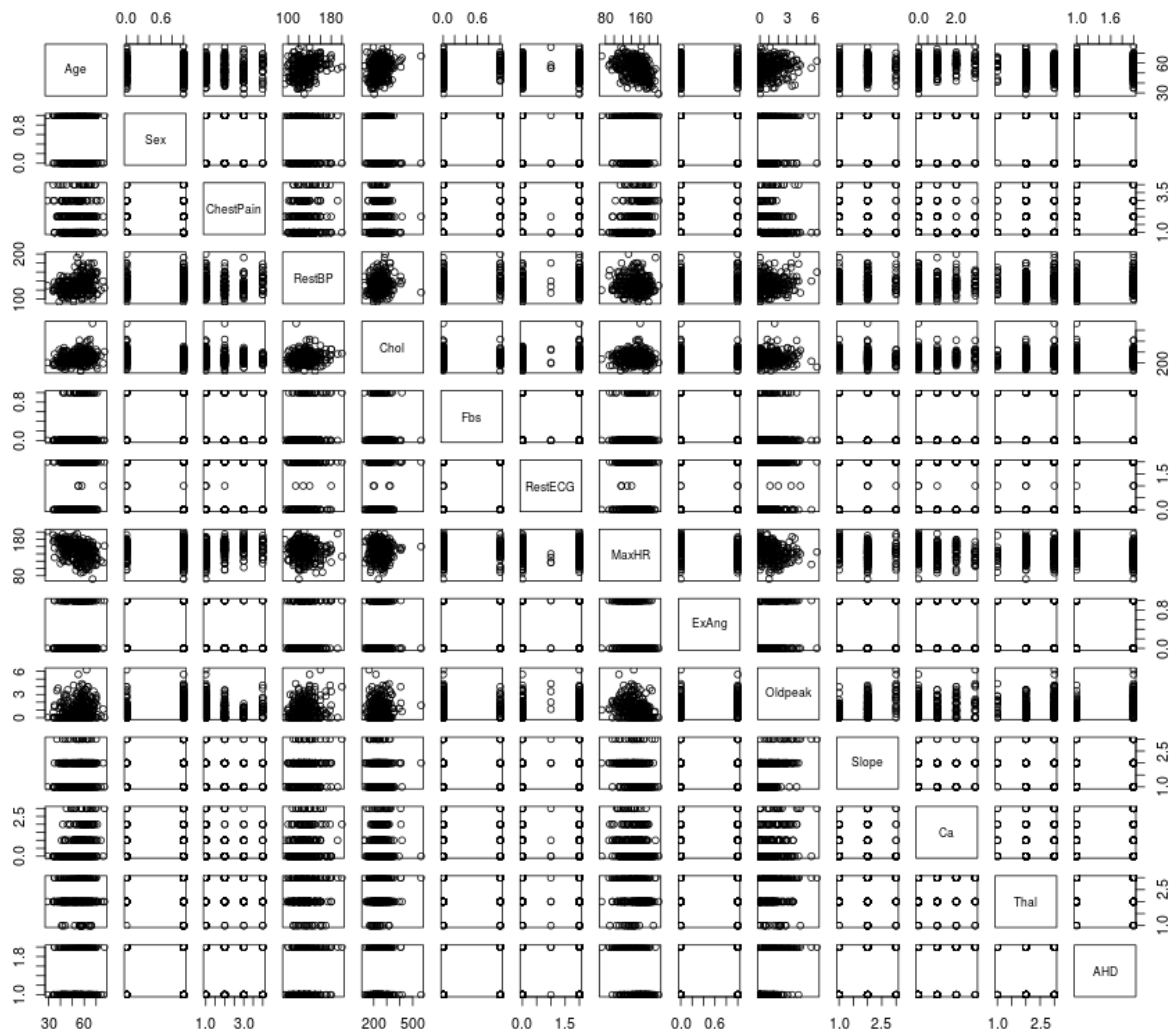


Fig. 2: Summary plot of the data with R.

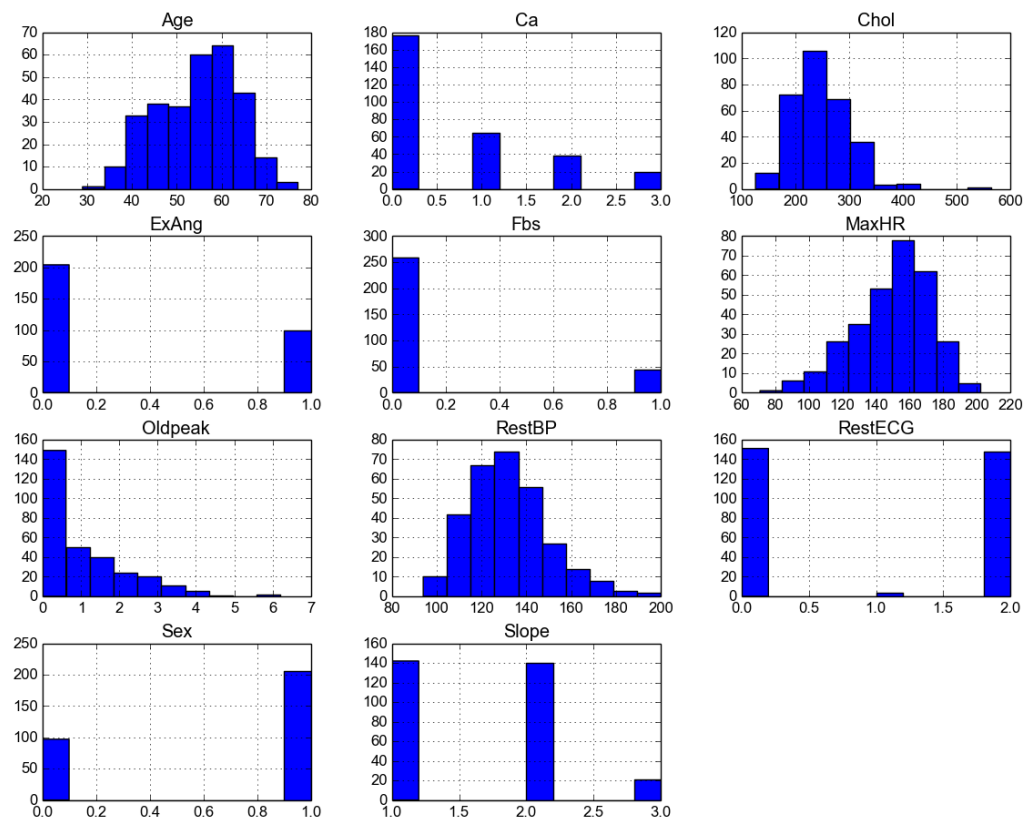


Fig. 3: Histogram in Python.

- Histogram in R

```
# Histogram with normal curve plot
dev.off()
Nvars=ncol(numdata)
name=colnames(numdata)
par(mfrow =c (4,3))
for (i in 1:Nvars)
{
  x<- numdata[,i]
  h<-hist(x, breaks=10, freq=TRUE, col="blue", xlab=name[i],main=" ",
          font.lab=1)
  axis(1, tck=1, col.ticks="light gray")
  axis(1, tck=-0.015, col.ticks="black")
  axis(2, tck=1, col.ticks="light gray", lwd.ticks="1")
  axis(2, tck=-0.015)
  xfit<-seq(min(x),max(x),length=40)
  yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))
  yfit <- yfit*diff(h$mids[1:2])*length(x)
  lines(xfit, yfit, col="blue", lwd=2)
}
```

Then you will get Figure *Histogram with normal curve plot in R*.

4.6.3 Boxplot of the quantitative predictors

Python

- Boxplot in Python

```
# boxplot
pd.DataFrame.boxplot(rawdata)
plt.show()
```

Then you will get Figure *Histogram in Python*.

R

- Boxplot in R

```
dev.off()
name=colnames(numdata)
Nvars=ncol(numdata)
# boxplot
par(mfrow =c (4,3))
for (i in 1:Nvars)
{
  #boxplot(numdata[,i]~numdata[,Nvars],data=data,main=name[i])
  boxplot(numdata[,i],data=numdata,main=name[i])
}
```

Then you will get Figure *Boxplots in R*.

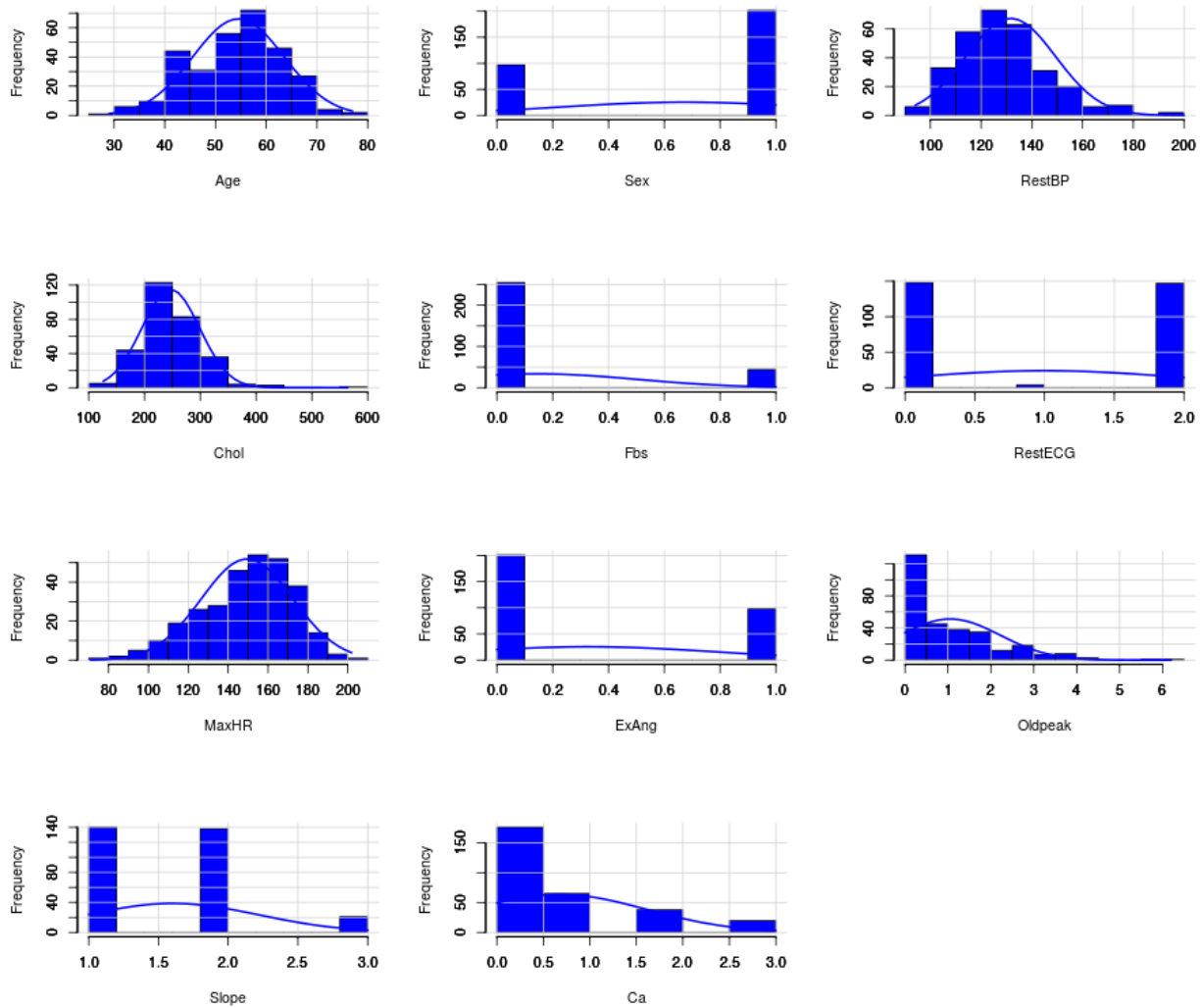


Fig. 4: Histogram with normal curve plot in R.

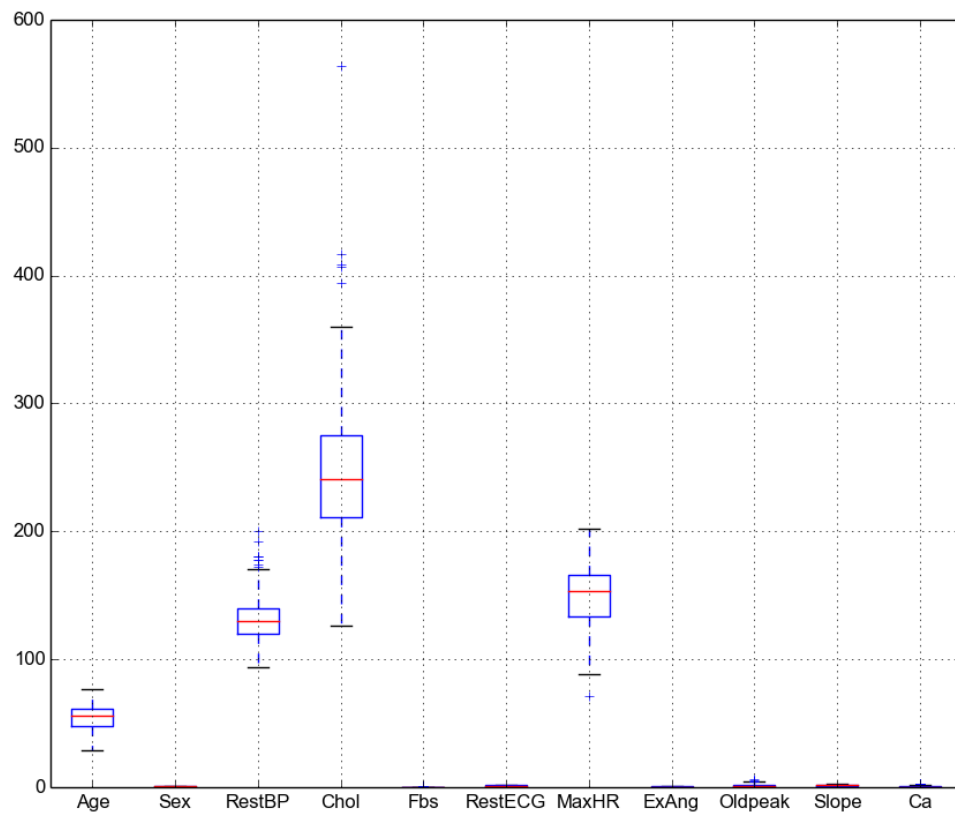


Fig. 5: Histogram in Python.

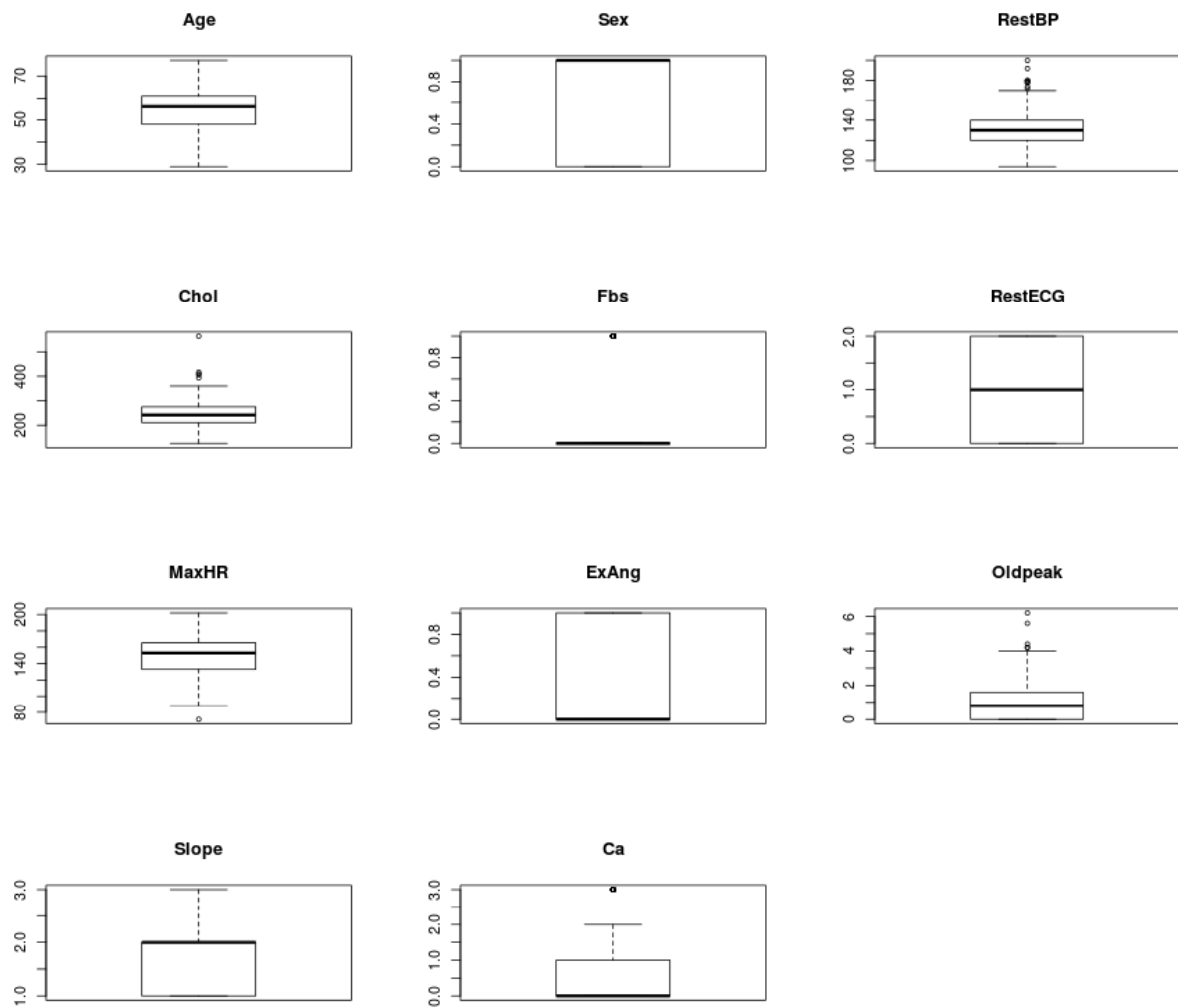


Fig. 6: Boxplots in R.

4.6.4 Correlation Matrix plot of the quantitative predictors

Python

- Correlation Matrix plot in **Python**

```
# cocorrelation Matrix plot
pd.DataFrame.corr(rawdata)
plt.show()
```

Then you will get get Figure *Correlation Matrix plot in Python*.



Fig. 7: Correlation Matrix plot in Python.

R

- Correlation Matrix plot in **R**

```
dev.off()
# load cocorrelation Matrix plot lib
library(corrplot)
M <- cor(numdata)
```

(continues on next page)

(continued from previous page)

```
#par(mfrow =c (1,2))
#corrplot(M, method = "square")
corrplot.mixed(M)
```

Then you will get Figure *Correlation Matrix plot in R*.

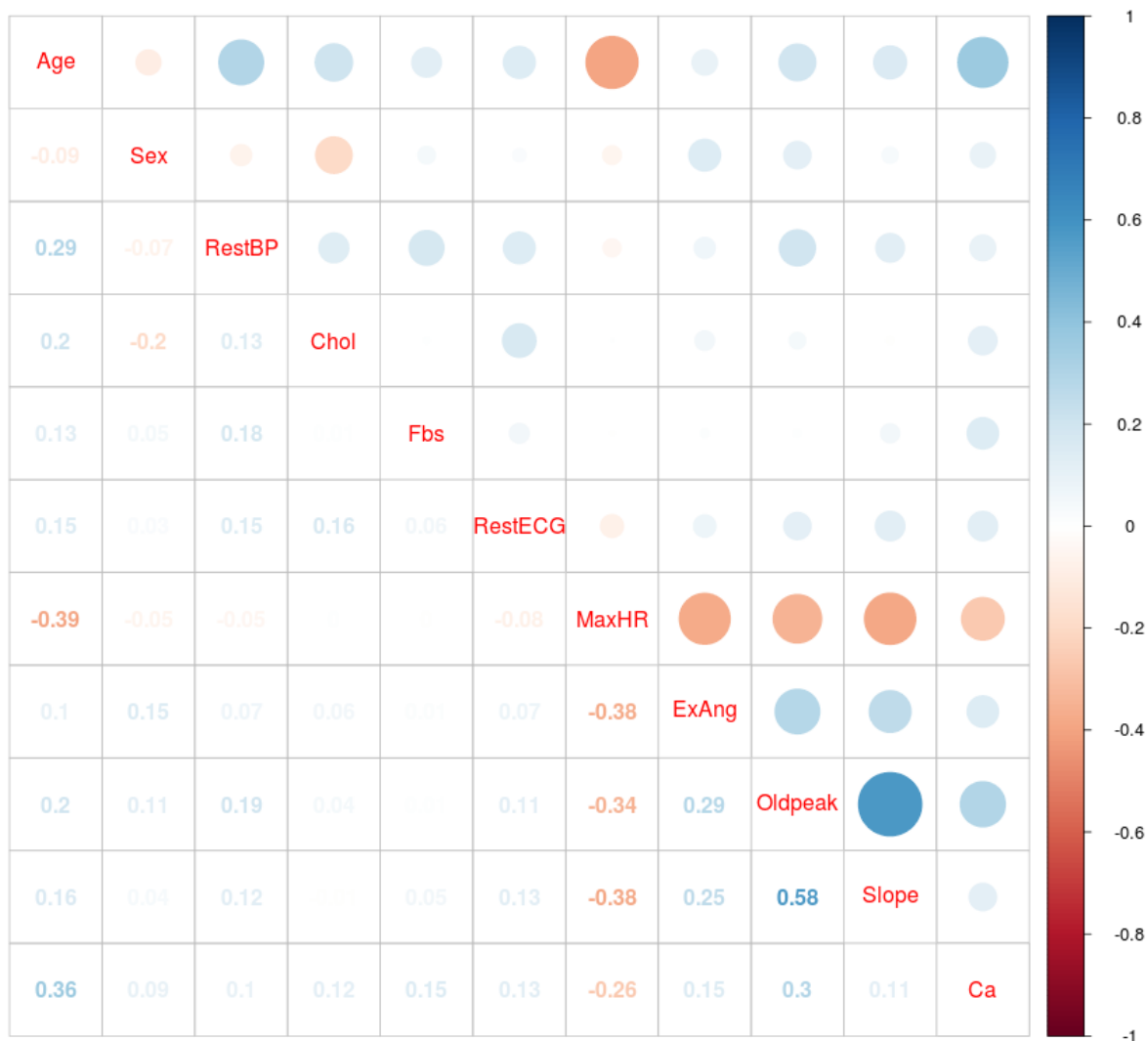


Fig. 8: Correlation Matrix plot in R.

4.7 Source Code for This Section

The code for this section is available for download for [R](#) for [Python](#),

Python

- Python Source code

```

'''
Created on Apr 25, 2016
test code
@author: Wenqiang Feng
'''

import pandas as pd
#import numpy as np
import matplotlib.pyplot as plt
from pandas.tools.plotting import scatter_matrix
from docutils.parsers.rst.directives import path

if __name__ == '__main__':
    path = '~/Dropbox/MachineLearningAlgorithms/python_code/data/Heart.csv'
    rawdata = pd.read_csv(path)

    print "data summary"
    print rawdata.describe()

    # summary plot of the data
    scatter_matrix(rawdata, figsize=[15,15])
    plt.show()

    # Histogram
    rawdata.hist()
    plt.show()

    # boxplot
    pd.DataFrame.boxplot(rawdata)
    plt.show()

    print "Raw data size"
    nrow, ncol = rawdata.shape
    print nrow, ncol

    path = ('/home/feng/Dropbox/MachineLearningAlgorithms/python_code/data/'
            'energy_efficiency.xlsx')
    path

    rawdataEnergy= pd.read_excel(path, sheetname=0)

    nrow=rawdata.shape[0] #gives number of row count
    ncol=rawdata.shape[1] #gives number of col count
    print nrow, ncol
    col_names = rawdata.columns.tolist()
    print "Column names:"
    print col_names
    print "Data Format:"
    print rawdata.dtypes

    print "\nSample data:"
    print (rawdata.head(6))

```

(continues on next page)

(continued from previous page)

```

print "\n correlation Matrix"
print rawdata.corr()

# cocorrelation Matrix plot
pd.DataFrame.corr(rawdata)
plt.show()

print "\n covariance Matrix"
print rawdata.cov()

print rawdata[['Age', 'Ca']].corr()
pd.DataFrame.corr(rawdata)
plt.show()

# define colors list, to be used to plot survived either red (=0) or
→green (=1)
colors=['red', 'green']

# make a scatter plot

# rawdata.info()

from scipy import stats
import seaborn as sns # just a conventional alias, don't know why
sns.corrplot(rawdata) # compute and plot the pair-wise correlations
# save to file, remove the big white borders
#plt.savefig('attribute_correlations.png', tight_layout=True)
plt.show()

attr = rawdata['Age']
sns.distplot(attr)
plt.show()

sns.distplot(attr, kde=False, fit=stats.gamma);
plt.show()

# Two subplots, the axes array is 1-d
plt.figure(1)
plt.title('Histogram of Age')
plt.subplot(211) # 21,1 means first one of 2 rows, 1 col
sns.distplot(attr)

plt.subplot(212) # 21,2 means second one of 2 rows, 1 col
sns.distplot(attr, kde=False, fit=stats.gamma);

plt.show()

```

R

• R Source code

```

rm(list = ls())
# set the environment
path = '~/Dropbox/MachineLearningAlgorithms/python_code/data/Heart.csv'
rawdata = read.csv(path)

# summary of the data
summary(rawdata)
# plot of the summary
plot(rawdata)

dim(rawdata)
head(rawdata)
tail(rawdata)

colnames(rawdata)
attach(rawdata)

# get numerical data and remove NAN
numdata=na.omit(rawdata[,c(1:2,4:12)])

cor(numdata)
cov(numdata)

dev.off()
# load cocorrelation Matrix plot lib
library(corrplot)
M <- cor(numdata)
#par(mfrow =c (1,2))
#corrplot(M, method = "square")
corrplot.mixed(M)

nrow=nrow(rawdata)
ncol=ncol(rawdata)
c(nrow, ncol)

Nvars=ncol(numdata)
# checking data format
typeof(rawdata)
install.packages("mlbench")
library(mlbench)
sapply(rawdata, class)

dev.off()
name=colnames(numdata)
Nvars=ncol(numdata)
# boxplot

```

(continues on next page)

(continued from previous page)

```

par(mfrow =c (4,3))
for (i in 1:Nvars)
{
  #boxplot (numdata[,i]~numdata[,Nvars],data=data,main=name[i])
  boxplot (numdata[,i],data=numdata,main=name[i])
}

# Histogram with normal curve plot
dev.off()
Nvars=ncol (numdata)
name=colnames (numdata)
par(mfrow =c (3,5))
for (i in 1:Nvars)
{
  x<- numdata[,i]
  h<-hist(x, breaks=10, freq=TRUE, col="blue", xlab=name[i],main=" ",
        font.lab=1)
  axis(1, tck=1, col.ticks="light gray")
  axis(1, tck=-0.015, col.ticks="black")
  axis(2, tck=1, col.ticks="light gray", lwd.ticks="1")
  axis(2, tck=-0.015)
  xfit<-seq(min(x),max(x),length=40)
  yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))
  yfit <- yfit*diff(h$mids[1:2])*length(x)
  lines(xfit, yfit, col="blue", lwd=2)
}

library(reshape2)
library(ggplot2)
d <- melt(diamonds[, -c(2:4)])
ggplot(d,aes(x = value)) +
  facet_wrap(~variable,scales = "free_x") +
  geom_histogram()

```


DATA MANIPULATION

5.1 Combining DataFrame

5.1.1 Mutating Joins

0. Datasets

Python

```
import pandas as pd
left = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3'],
                     'C': ['C0', 'C1', 'C2', 'C3'],
                     'D': ['D0', 'D1', 'D2', 'D3']},
                    index=[0, 1, 2, 3])

right = pd.DataFrame({'A': ['A0', 'A1', 'A6', 'A7'],
                     'F': ['B4', 'B5', 'B6', 'B7'],
                     'G': ['C4', 'C5', 'C6', 'C7'],
                     'H': ['D4', 'D5', 'D6', 'D7']},
                    index=[4, 5, 6, 7])

print(left)
print(right)
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

	A	F	G	H
4	A0	B4	C4	D4
5	A1	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

R

```
left = data.frame(A = c('A0', 'A1', 'A2', 'A3'),
                  B = c('B0', 'B1', 'B2', 'B3'),
                  C = c('C0', 'C1', 'C2', 'C3'),
                  D = c('D0', 'D1', 'D2', 'D3'))

left

right = data.frame(A = c('A0', 'A1', 'A6', 'A7'),
                  F = c('B4', 'B5', 'B6', 'B7'),
                  G = c('C4', 'C5', 'C6', 'C7'),
                  H = c('D4', 'D5', 'D6', 'D7'))

right

> left
  A B C D
1 A0 B0 C0 D0
2 A1 B1 C1 D1
3 A2 B2 C2 D2
4 A3 B3 C3 D3

> right
  A F G H
1 A0 B4 C4 D4
2 A1 B5 C5 D5
3 A6 B6 C6 D6
4 A7 B7 C7 D7
```

1. Left join

Python

- Code:

```
# left join
left_join = left.merge(right, on='A', how='left')
print(left_join)
```

- Result:

	A	B	C	D	F	G	H
0	A0	B0	C0	D0	B4	C4	D4
1	A1	B1	C1	D1	B5	C5	D5
2	A2	B2	C2	D2	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN

R

- Code:

```
library(dplyr)

# left join
dplyr::left_join(left, right, by = 'A')
```

(continues on next page)

(continued from previous page)

```
# or
left %>% left_join(right, by = 'A')
```

- Result:

```
> dplyr::left_join(left, right, by = 'A')
  A   B   C   D     F     G     H
1 A0 B0 C0 D0    B4    C4    D4
2 A1 B1 C1 D1    B5    C5    D5
3 A2 B2 C2 D2 <NA> <NA> <NA>
4 A3 B3 C3 D3 <NA> <NA> <NA>
Warning message:
Column `A` joining factors with different levels, coercing to character vector
> left %>% left_join(right, by = 'A')
  A   B   C   D     F     G     H
1 A0 B0 C0 D0    B4    C4    D4
2 A1 B1 C1 D1    B5    C5    D5
3 A2 B2 C2 D2 <NA> <NA> <NA>
4 A3 B3 C3 D3 <NA> <NA> <NA>
```

2. Right join

Python

- Code:

```
# right join
right_join = left.merge(right, on='A', how='right')
print(right_join)
```

- Result:

```
   A   B   C   D   F   G   H
0 A0 B0 C0 D0 B4 C4 D4
1 A1 B1 C1 D1 B5 C5 D5
2 A6 NaN NaN NaN B6 C6 D6
3 A7 NaN NaN NaN B7 C7 D7
```

R

- Code:

```
library(dplyr)

# right join
dplyr::right_join(left, right, by = 'A')
left %>% right_join(right, by = 'A')
```

- Result:

```
> dplyr::right_join(left, right, by = 'A')
  A   B   C   D     F     G     H
```

(continues on next page)

(continued from previous page)

```

1 A0 B0 C0 D0 B4 C4 D4
2 A1 B1 C1 D1 B5 C5 D5
3 A6 <NA> <NA> <NA> B6 C6 D6
4 A7 <NA> <NA> <NA> B7 C7 D7
Warning message:
Column `A` joining factors with different levels, coercing to character vector
> left %>% right_join(right, by = 'A')
  A B C D F G H
1 A0 B0 C0 D0 B4 C4 D4
2 A1 B1 C1 D1 B5 C5 D5
3 A6 <NA> <NA> <NA> B6 C6 D6
4 A7 <NA> <NA> <NA> B7 C7 D7
Warning message:
Column `A` joining factors with different levels, coercing to character vector

```

3. Inner join

Python

- Code:

```

# inner join
inner_join = left.merge(right, on='A', how='inner')
print(inner_join)

```

- Result:

```

  A B C D F G H
0 A0 B0 C0 D0 B4 C4 D4
1 A1 B1 C1 D1 B5 C5 D5

```

R

- Code:

```

library(dplyr)

# inner join
dplyr::inner_join(left, right, by = 'A')
left %>% inner_join(right, by = 'A')

```

- Result:

```

> dplyr::inner_join(left, right, by = 'A')
  A B C D F G H
1 A0 B0 C0 D0 B4 C4 D4
2 A1 B1 C1 D1 B5 C5 D5
Warning message:
Column `A` joining factors with different levels, coercing to character vector
> left %>% inner_join(right, by = 'A')
  A B C D F G H
1 A0 B0 C0 D0 B4 C4 D4

```

(continues on next page)

(continued from previous page)

```
2 A1 B1 C1 D1 B5 C5 D5
Warning message:
Column `A` joining factors with different levels, coercing to character vector
```

4. Full join

Python

- Code:

```
# full join
full_join = left.merge(right, on='A', how='outer')
print(full_join)
```

- Result:

	A	B	C	D	F	G	H
0	A0	B0	C0	D0	B4	C4	D4
1	A1	B1	C1	D1	B5	C5	D5
2	A2	B2	C2	D2	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN
4	A6	NaN	NaN	NaN	B6	C6	D6
5	A7	NaN	NaN	NaN	B7	C7	D7

R

- Code:

```
library(dplyr)

# full join
dplyr::full_join(left, right, by = 'A')
left %>% full_join(right, by = 'A')
```

- Result:

```
> dplyr::full_join(left, right, by = 'A')
  A     B     C     D     F     G     H
1 A0  B0  C0  D0  B4  C4  D4
2 A1  B1  C1  D1  B5  C5  D5
3 A2  B2  C2  D2  <NA> <NA> <NA>
4 A3  B3  C3  D3  <NA> <NA> <NA>
5 A6  <NA> <NA> <NA>  B6  C6  D6
6 A7  <NA> <NA> <NA>  B7  C7  D7
Warning message:
Column `A` joining factors with different levels, coercing to character vector
> left %>% full_join(right, by = 'A')
  A     B     C     D     F     G     H
1 A0  B0  C0  D0  B4  C4  D4
2 A1  B1  C1  D1  B5  C5  D5
3 A2  B2  C2  D2  <NA> <NA> <NA>
4 A3  B3  C3  D3  <NA> <NA> <NA>
```

(continues on next page)

(continued from previous page)

```
5 A6 <NA> <NA> <NA>    B6    C6    D6
6 A7 <NA> <NA> <NA>    B7    C7    D7
Warning message:
Column `A` joining factors with different levels, coercing to character vector
```

5.1.2 Filtering Joins

5.2 DataFrame Operations

TO DO

PRE-PROCESSING PROCEDURES

Note: Well begun is half done – old Chinese proverb

In my opinion, preprocessing is crucial for the data mining algorithms. If you get a good pre-processing, you will definitely get a better result. In this section, we will learn how to do a proper pre-processing in **R** and **Python**.

6.1 Rough Pre-processing

- **dealing with missing data**

Usually, we have two popular way to deal with the missing data: replacing by 0 or replacing by mean value.

- dealing with missing data in **R**
- dealing with missing data in **Python**

6.2 Source Code for This Section

The code for this section is available for download for **R**, for **Python**,

- R Source code

```
rm(list = ls())  
# set the environment  
path = '~/Dropbox/MachineLearningAlgorithms/python_code/data/  
↪Heart.csv'  
rawdata = read.csv(path)  
  
# summary of the data  
summary(rawdata)  
# plot of the summary  
plot(rawdata)
```

(continues on next page)

(continued from previous page)

```

dim(rawdata)
head(rawdata)
tail(rawdata)

colnames(rawdata)
attach(rawdata)

# get numerical data and remove NAN
numdata=na.omit(rawdata[,c(1:2,4:12)])

cor(numdata)
cov(numdata)

dev.off()
# load cocorrelation Matrix plot lib
library(corrplot)
M <- cor(numdata)
#par(mfrow =c (1,2))
#corrplot(M, method = "square")
corrplot.mixed(M)

nrow=nrow(rawdata)
ncol=ncol(rawdata)
c(nrow, ncol)

Nvars=ncol(numdata)
# checking data format
typeof(rawdata)
install.packages("mlbench")
library(mlbench)
sapply(rawdata, class)

dev.off()
name=colnames(numdata)
Nvars=ncol(numdata)
# boxplot
par(mfrow =c (4,3))
for (i in 1:Nvars)
{
  #boxplot(numdata[,i]~numdata[,Nvars],data=data,main=name[i])
  boxplot(numdata[,i],data=numdata,main=name[i])
}

# Histogram with normal curve plot
dev.off()
Nvars=ncol(numdata)
name=colnames(numdata)
par(mfrow =c (3,5))
for (i in 1:Nvars)

```

(continues on next page)

(continued from previous page)

```

{
  x<- numdata[,i]
  h<-hist(x, breaks=10, freq=TRUE, col="blue", xlab=name[i],main=
↪ " ",
      font.lab=1)
  axis(1, tck=1, col.ticks="light gray")
  axis(1, tck=-0.015, col.ticks="black")
  axis(2, tck=1, col.ticks="light gray", lwd.ticks="1")
  axis(2, tck=-0.015)
  xfit<-seq(min(x),max(x),length=40)
  yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))
  yfit <- yfit*diff(h$mids[1:2])*length(x)
  lines(xfit, yfit, col="blue", lwd=2)
}

library(reshape2)
library(ggplot2)
d <- melt(diamonds[, -c(2:4)])
ggplot(d,aes(x = value)) +
  facet_wrap(~variable,scales = "free_x") +
  geom_histogram()

```

- Python Source code

```

'''
Created on Apr 25, 2016
test code
@author: Wenqiang Feng
'''

import pandas as pd
#import numpy as np
import matplotlib.pyplot as plt
from pandas.tools.plotting import scatter_matrix
from docutils.parsers.rst.directives import path

if __name__ == '__main__':
    path = '~/Dropbox/MachineLearningAlgorithms/python_code/data/
↪Heart.csv'
    rawdata = pd.read_csv(path)

    print "data summary"
    print rawdata.describe()

    # summary plot of the data
    scatter_matrix(rawdata,figsize=[15,15])
    plt.show()

    # Histogram
    rawdata.hist()
    plt.show()

```

(continues on next page)

(continued from previous page)

```

# boxplot
pd.DataFrame.boxplot(rawdata)
plt.show()

print "Raw data size"
nrow, ncol = rawdata.shape
print nrow, ncol

path = ('/home/feng/Dropbox/MachineLearningAlgorithms/python_
↪code/data/'
'energy_efficiency.xlsx')
path

rawdataEnergy= pd.read_excel(path, sheetname=0)

nrow=rawdata.shape[0] #gives number of row count
ncol=rawdata.shape[1] #gives number of col count
print nrow, ncol
col_names = rawdata.columns.tolist()
print "Column names:"
print col_names
print "Data Format:"
print rawdata.dtypes

print "\nSample data:"
print (rawdata.head(6))

print "\n correlation Matrix"
print rawdata.corr()

# cocorrelation Matrix plot
pd.DataFrame.corr(rawdata)
plt.show()

print "\n covariance Matrix"
print rawdata.cov()

print rawdata[['Age', 'Ca']].corr()
pd.DataFrame.corr(rawdata)
plt.show()

# define colors list, to be used to plot survived either red
↪(=0) or green (=1)
colors=['red', 'green']

# make a scatter plot

# rawdata.info()

```

(continues on next page)

(continued from previous page)

```
from scipy import stats
import seaborn as sns # just a conventional alias, don't_
↪ know why
sns.corrplot(rawdata) # compute and plot the pair-wise_
↪ correlations
# save to file, remove the big white borders
#plt.savefig('attribute_correlations.png', tight_layout=True)
plt.show()

attr = rawdata['Age']
sns.distplot(attr)
plt.show()

sns.distplot(attr, kde=False, fit=stats.gamma);
plt.show()

# Two subplots, the axes array is 1-d
plt.figure(1)
plt.title('Histogram of Age')
plt.subplot(211) # 21,1 means first one of 2 rows, 1 col
sns.distplot(attr)

plt.subplot(212) # 21,2 means second one of 2 rows, 1 col
sns.distplot(attr, kde=False, fit=stats.gamma);

plt.show()
```


SUMMARY OF DATA MINING ALGORITHMS

Note: Know yourself and know your enemy, and you will never be defeated— idiom, from Sunzi's Art of War

Although the tutorials presented here is not plan to focus on the theoretical frameworks of Data Mining, it is still worth to understand how they are works and know what's the assumption of those algorithm. This is an important steps to know ourselves.

7.1 Diagram of Data Mining Algorithms

An awesome Tour of Machine Learning Algorithms was published online by [Jason Brownlee](#) in 2013, it still is a good category diagram.

7.2 Categories of Data Mining Algorithms

0. Dimensionality Reduction Algorithms

- Principal Component Analysis (PCA)
- Nonnegative Matrix Factorization (NMF)
- Independent Component Analysis (ICA)
- Linear Discriminant Analysis (LDA)

1. Regression Algorithms

- Ordinary Least Squares Regression (OLSR)
- Linear Regression
- Logistic Regression

2. Regularization Algorithms

- Ridge Regression

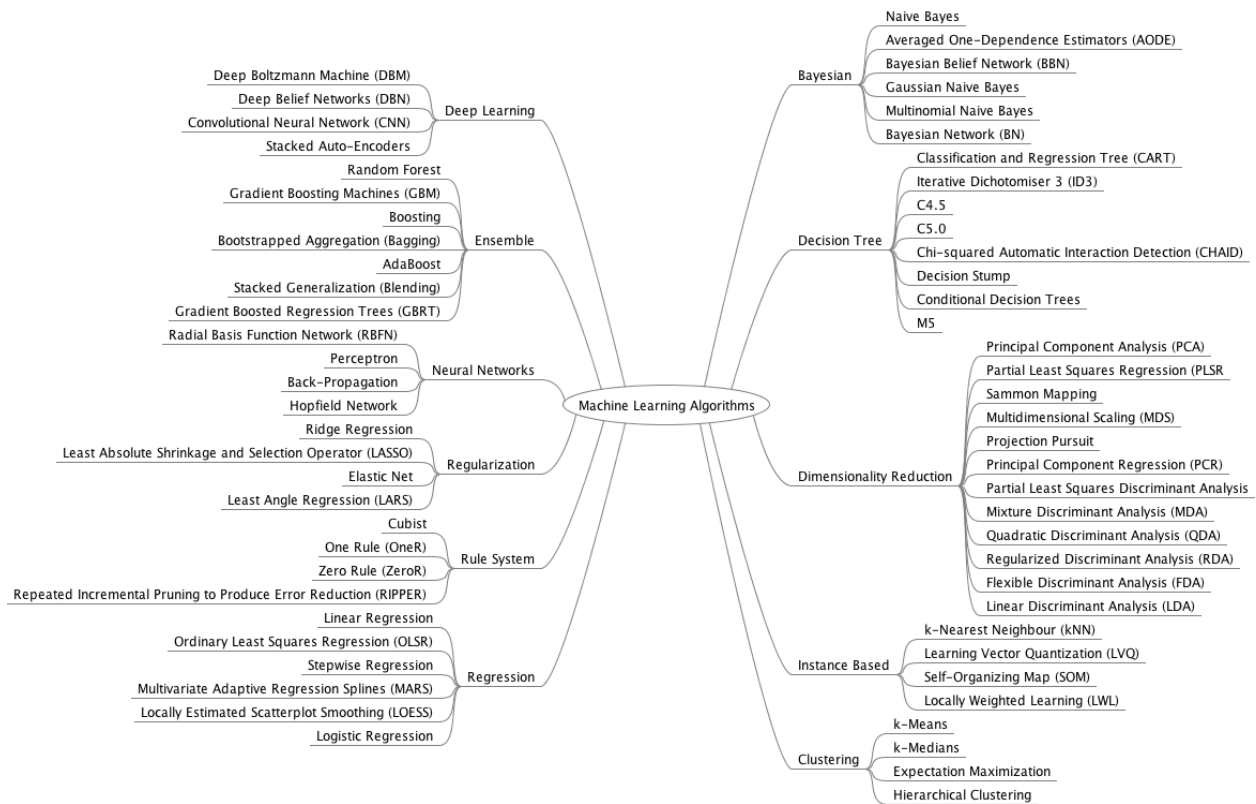


Fig. 1: **Figure** : Machine Learning Algorithms diagram from [Jason Brownlee](#) .

- Least Absolute Shrinkage and Selection Operator (LASSO)
 - Elastic Net
 - Least-Angle Regression (LARS)
3. Decision Tree Algorithms
- Classification and Regression Tree (CART)
 - Conditional Decision Trees
5. Bayesian Algorithms
- Naive Bayes
6. Clustering Algorithms
- k-Means
 - k-Medians
 - Expectation Maximisation (EM)
 - Hierarchical Clustering
8. Artificial Neural Network Algorithms
- Perceptron
 - Back-Propagation
 - Hopfield Network
 - Radial Basis Function Network (RBFN)
9. Deep Learning Algorithms
- Deep Boltzmann Machine (DBM)
 - Deep Belief Networks (DBN)
11. Ensemble Algorithms
- Boosting
 - Bootstrapped Aggregation (Bagging)
 - AdaBoost
 - Gradient Boosting Machines (GBM)
 - Gradient Boosted Regression Trees (GBRT)
 - Random Forest

DIMENSION REDUCTION ALGORITHMS

8.1 What is dimension reduction?

In machine learning and statistics, dimensionality reduction or dimension reduction is the process of reducing the number of random variables under consideration, via obtaining a set “uncorrelated” principle variables. It can be divided into feature selection and feature extraction. https://en.wikipedia.org/wiki/Dimensionality_reduction

8.2 Singular Value Decomposition (SVD)

At here, I will recall the three types of the SVD method, since some authors confused the definitions of these SVD method. SVD method is important for the the dimension reduction algorithms, such as Truncated Singular Value Decomposition (tSVD) can be used to do the dimension reduction directly, and the Full Rank Singular Value Decomposition (SVD) can be applied to do Principal Component Analysis (PCA), since PCA is a specific case of SVD.

1. Full Rank Singular Value Decomposition (SVD)

Suppose $\mathbf{X} \in \mathbb{R}^{n \times p}$, ($p < n$), then

$$\mathbf{X}_{n \times p} = \mathbf{U}_{n \times n} \mathbf{\Sigma}_{n \times p} \mathbf{V}_{p \times p}^T,$$

is called a full rank **SVD** of \mathbf{X} and

- σ_i – Singular values and $\mathbf{\Sigma} = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_p) \in \mathbb{R}^{n \times p}$
- u_i – left singular vectors, $\mathbf{U} = [u_1, u_2, \dots, u_n]$ and \mathbf{U} is unitary.
- v_i – right singular vectors, $\mathbf{V} = [v_1, v_2, \dots, v_p]$ and \mathbf{V} is unitary.

2. Reduced Singular Value Decomposition (rSVD)

Suppose $\mathbf{X} \in \mathbb{R}^{n \times p}$, ($n < p$), then

$$\mathbf{X}_{n \times p} = \hat{\mathbf{U}}_{n \times p} \hat{\mathbf{\Sigma}}_{p \times p} \hat{\mathbf{V}}_{p \times p}^T,$$

is called a Reduced Singular Value Decomposition **rSVD** of \mathbf{X} and

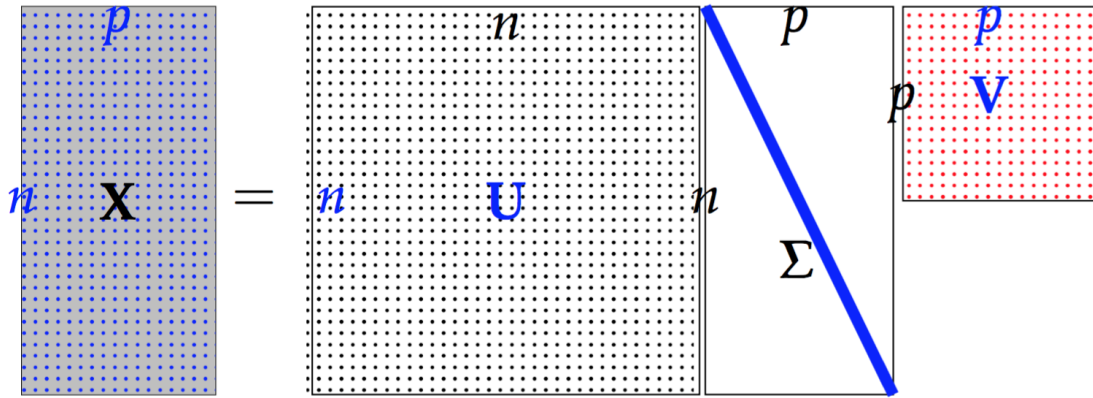


Fig. 1: Singular Value Decomposition

- σ_i — Singular values and $\hat{\Sigma} = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_p) \in \mathbb{R}^{p \times p}$
- u_i — left singular vectors, $\hat{U} = [u_1, u_2, \dots, u_p]$ is column-orthonormal matrix.
- v_i — right singular vectors, $\hat{V} = [v_1, v_2, \dots, v_p]$ is column-orthonormal matrix.

3. Truncated Singular Value Decomposition (tSVD)

Suppose $\mathbf{X} \in \mathbb{R}^{n \times p}$, ($r < p$), then

$$\underset{n \times p}{\mathbf{X}} = \underset{n \times r}{\hat{\mathbf{U}}} \underset{r \times r}{\hat{\Sigma}} \underset{r \times p}{\hat{\mathbf{V}}^T}, \quad (8.1)$$

is called a Truncated Singular Value Decomposition **tSVD** of \mathbf{X} and

- σ_i — Singular values and $\hat{\Sigma} = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_r) \in \mathbb{R}^{r \times r}$
- u_i — left singular vectors, $\hat{U} = [u_1, u_2, \dots, u_r]$ is column-orthonormal matrix.
- v_i — right singular vectors, $\hat{V} = [v_1, v_2, \dots, v_p]$ is column-orthonormal matrix.

Figure *Truncated Singular Value Decomposition* indicates that the dimension of \hat{U} is smaller than \mathbf{X} . We can use this property to do the dimension reduction. But, usually, we will use SVD to compute the Principal Components. We will learn more details in next section.

8.3 Principal Component Analysis (PCA)

Usually, there are two ways to implement the PCA. Principal Component Analysis (PCA) is a specific case of SVD.

$$\underset{n \times p}{\mathbf{X}} = \hat{\mathbf{U}} \quad (8.2)$$

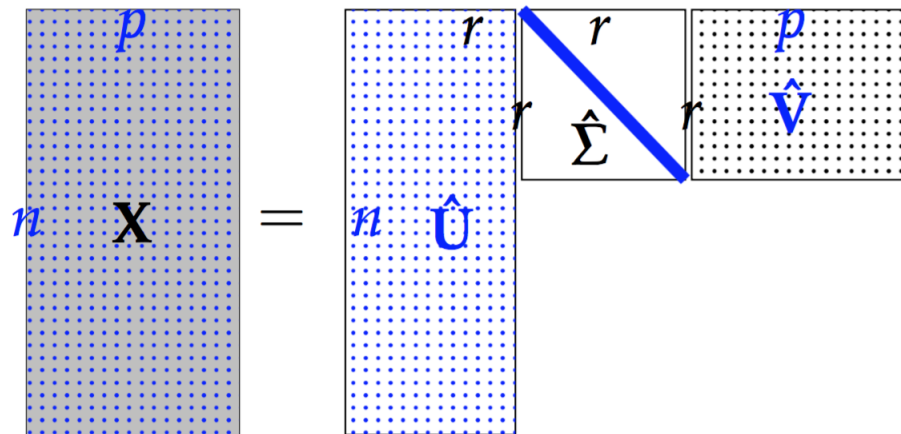


Fig. 2: Truncated Singular Value Decomposition

8.4 Independent Component Analysis (ICA)

8.5 Nonnegative matrix factorization (NMF)

TO DO.....

REGRESSION ALGORITHM

Note: A journey of a thousand miles begins with a single step – old Chinese proverb

In statistical modeling, regression analysis focuses on investigating the relationship between a dependent variable and one or more independent variables. [Wikipedia Regression analysis](#)

In data mining, Regression is a model to represent the relationship between the value of label (or target, it is numerical variable) and on one or more features (or predictors they can be numerical and categorical variables).

9.1 Introduction

Given that a data set $\{x_{i1}, \dots, x_{in}, y_i\}_{i=1}^m$ which contains n features (variables) and m samples (data points), in simple linear regression model for modeling m data points with j independent variables: x_{ij} , the formula is given by:

$$y_i = \beta_0 + \beta_j x_{ij}, \text{ where } i = 1, \dots, m, j = 1, \dots, n.$$

In matrix notation, the data set is written as $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$ with $\mathbf{x}_j = \{x_{ij}\}_{i=1}^m$, $\mathbf{y} = \{y_i\}_{i=1}^m$ (see Fig. *Feature matrix and label*) and $\boldsymbol{\beta}^\top = \{\beta_j\}_{j=1}^n$. Then the matrix format equation is written as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta}. \tag{9.1}$$

9.2 Ordinary Least Squares Regression (OLSR)

9.2.1 How to solve it?

Theoretically, you can apply all the following methods to solve (9.1) if you matrix \mathbf{X} have a good properties.

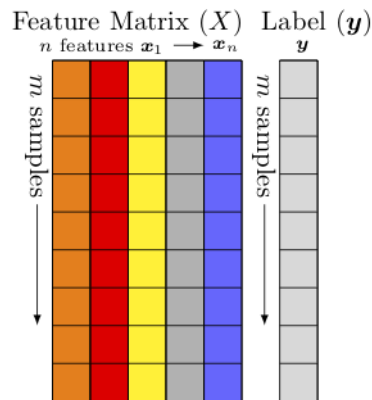


Fig. 1: Feature matrix and label

1. Direct Methods (For more information please refer to my [Prelim Notes for Numerical Analysis](#))

- For squared or rectangular matrices
 - Singular Value Decomposition
 - Gram-Schmidt orthogonalization
 - QR Decomposition
- For squared matrices
 - LU Decomposition
 - Cholesky Decomposition
 - Regular Splittings

2. Iterative Methods

- Stationary cases iterative method
 - Jacobi Method
 - Gauss-Seidel Method
 - Richardson Method
 - Successive Over Relaxation (SOR) Method
- Dynamic cases iterative method
 - Chebyshev iterative Method
 - Minimal residuals Method
 - Minimal correction iterative method
 - Steepest Descent Method
 - Conjugate Gradients Method

9.2.2 Ordinary Least Squares

In mathematics, (9.1) is a overdetermined system. The method of ordinary least squares can be used to find an approximate solution to overdetermined systems. For the system overdetermined system (9.1), the least squares formula is obtained from the problem

$$\min_x ||\mathbf{X}\beta - \mathbf{y}||, \quad (9.2)$$

the solution of which can be written with the normal equations:

$$\beta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (9.3)$$

where T indicates a matrix transpose, provided $(\mathbf{X}^T \mathbf{X})^{-1}$ exists (that is, provided \mathbf{X} has full column rank).

Note: Actually, (9.3) is derivated by the following way: multiply \mathbf{X}^T on side of (9.1) and then multiply $(\mathbf{X}^T \mathbf{X})^{-1}$ on both side of the former result.

9.3 Linear Regression (LR)

TO DO

CLASSIFICATION ALGORITHMS

10.1 Logistic Regression (LR)

10.2 k-Nearest Neighbour (kNN)

10.3 Linear Discriminant Analysis (LDA)

10.4 Quadratic Discriminant Analysis (QDA)

TO DO

REGULARIZATION ALGORITHMS

11.1 Subset Selection (SubS)

11.2 Ridge Regression (Ridge)

11.3 Least Absolute Shrinkage and Selection Operator (LASSO)

TO DO

RESAMPLING ALGORITHMS

TO DO

DEVELOPING YOUR OWN R PACKAGES

TO DO.....

DEVELOPING YOUR OWN PYTHON PACKAGES

It's super easy to wrap your own package in Python. I packed some functions which I frequently used in my daily work. You can download and install it from [My ststspy library](#). The hierarchical structure and the directory structure of this package are as follows.

14.1 Hierarchical Structure

```
├── README.md
├── __init__.py
├── requirements.txt
├── setup.py
├── statspy
│   ├── __init__.py
│   ├── basics.py
│   └── tests.py
├── test
│   ├── nb
│   │   └── t.test.ipynb
│   └── test1.py
3 directories, 9 files
```

From the above hierarchical structure, you will find that you have to have `__init__.py` in each directory. I will explain the `__init__.py` file with the example below:

14.2 Set Up

```
from setuptools import setup, find_packages

try:
    with open("README.md") as f:
        long_description = f.read()
except IOError:
    long_description = ""
```

(continues on next page)

(continued from previous page)

```
try:
    with open("requirements.txt") as f:
        requirements = [x.strip() for x in f.read().splitlines() if x.strip()]
except IOError:
    requirements = []

setup(name='statspy',
      install_requires=requirements,
      version='1.0',
      description='Statistics python library',
      author='Wenqiang Feng',
      author_email='von198@gmail.com',
      url='git@github.com:runawayhorse001/statspy.git',
      packages=find_packages(),
      long_description=long_description
    )
```

14.3 Requirements

```
pandas
numpy
scipy
patsy
matplotlib
```

14.4 ReadMe

```
# StatsPy

This is my statistics python library repositories.
The ``API`` can be found at: https://runawayhorse001.github.io/statspy.
If you want to clone and install it, you can use

- clone

``{bash}
git clone git@github.com:runawayhorse001/statspy.git
``

- install

``{bash}
cd statspy
pip install -r requirements.txt
python setup.py install
``

- uninstall
```

(continues on next page)

(continued from previous page)

```
```{bash}
pip uninstall statspy
```

- test

```{bash}
cd statspy/test
python test1.py
```
```


INDEX

A

Audit Data, [17](#)

D

Data Exploration, [12](#)

Datasets, [15](#)

Dimension Reduction Algorithms, [57](#)

I

Independent Component Analysis, [61](#)

Installing package, [10](#)

Installing programming language, [9](#)

Installing programming platform, [10](#)

L

Loading Datasets, [15](#)

N

Nonnegative matrix factorization, [61](#)

P

Pre-processing procedures, [48](#)

Principal Component Analysis, [60](#)

procedures, [15](#)

R

rough preprocessing, [49](#)

S

Singular Value Decomposition, [59](#)

Summary of Data Mining Algorithms,
[53](#)

U

Understand Data With Statistics
methods, [20](#)

Understand Data With Visualization,
[29](#)