SUBJECT CODE : **3150713**

As per New Syllabus of

GUJARAT TECHNOLOGICAL UNIVERSITY

Semester - V (CE / CSE) - Open Elective - I

PYTHON FOR DATA SCIENCE

Iresh A. Dhotre

M.E. (Information Technology)

Ex-Faculty, Sinhgad College of Engineering,

Pune.

TECHNICAL PUBLICATIONS

SINCE 1993 — An Up-Thrust for Knowledge

PYTHON FOR DATA SCIENCE

Subject Code : 3150713

Semester - V (Computer Engineering / Computer Science & Engineering) - Open Elective - I

First Edition : August 2020

Published by :

TECHNICAL PUBLICATIONS
SINCE 1993 An Up-Thrust for Knowledge

Amit Residency, Office No.1, 412, Shaniwar Peth,
Pune - 411030, M.S. INDIA Ph.: +91-020-24495496/97
Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :
Yogiraj Printers & Binders
Sr. No. 10/1A,
Ghule Industrial Estate, Nanded Village Road,
Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-93-332-2142-9

9 789333 221429

Course 18

9789333221429 [1]

PREFACE

The importance of **Python for Data Science** is well known in various engineering fields. Overwhelming response to my books on various subjects inspired me to write this book. The book is structured to cover the key aspects of the subject **Python for Data Science**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

I wish to express my profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by my whole family. I wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Author
D. A. Dhotre

Dedicated to God.

SYLLABUS

Python for Data Science - (3150713)

Credits	Examination Marks				Total Marks
C	Theory Marks		Practical Marks		
	ESE (E)	PA(M)	ESE (V)	PA (I)	
3	70	30	30	20	150

1. Overview of Python and Data Structures :

Basics of Python including data types, variables, expressions, objects and functions. Python data structures including String, Array, List, Tuple, Set, Dictionary and operations them. **(Unit - I)**

2. Data Science and Python :

Discovering the match between data science and python :

Defining the Sexiest Job of the 21st Century, Considering the emergence of data science, Outlining the core competencies of a data scientist, Linking data science, big data, and AI , Understanding the role of programming, Creating the Data Science Pipeline, Preparing the data, Performing exploratory data analysis, Learning from data, Visualizing, Obtaining insights and data products, Understanding Python's Role in Data Science, Considering the shifting profile of data scientists, Working with a multipurpose, simple, and efficient language, Learning to Use Python Fast ,Loading data, Training a model, Viewing a result.

Introducing Python's Capabilities and Wonders :

Why Python?, Grasping Python's Core Philosophy, Contributing to data science, Discovering present and future development goals, Working with Python, Getting a taste of the language, Understanding the need for indentation, Working at the command line or in the IDE, Performing Rapid Prototyping and Experimentation, Considering Speed of Execution, Visualizing Power, Using the Python Ecosystem for Data Science, Accessing scientific tools using SciPy, Performing fundamental scientific computing using NumPy, Performing data analysis using pandas, Implementing machine learning using Scikit-learn, Going for deep learning with Keras and TensorFlow, Plotting the data using matplotlib, Creating graphs with NetworkX, Parsing HTML documents using Beautiful Soup. **(Unit - II)**

3. Getting Your Hands Dirty With Data :

Understanding the tools :

Using the Jupyter Console, Interacting with screen text, Changing the window appearance, Getting Python help, Getting IPython help, Using magic functions, Discovering objects, Using Jupyter Notebook, Working with styles, Restarting the kernel, Restoring a checkpoint, Performing Multimedia and Graphic Integration, Embedding plots and other images, Loading examples from online sites, Obtaining online graphics and multimedia.

Working with Real Data :

Uploading, Streaming, and Sampling Data, Uploading small amounts of data into memory, Streaming large amounts of data into memory, Generating variations on image data, Sampling data in different ways, Accessing Data in Structured Flat-File Form ,Reading from a text file Reading CSV delimited format, Reading Excel and other

Microsoft Office files, Sending Data in Unstructured File Form, Managing Data from Relational Databases, Interacting with Data from NoSQL Databases, Accessing Data from the Web.

Conditioning Your Data :

Juggling between NumPy and pandas, Knowing when to use NumPy, Knowing when to use pandas, Validating Your Data, Figuring out what's in your data, Removing duplicates, Creating a data map and data plan, Manipulating Categorical Variables, Creating categorical variables, Renaming levels, Combining levels, Dealing with Dates in Your Data, Formatting date and time values, Using the right time transformation, Dealing with Missing Data, Finding the missing data, Encoding missingness, Imputing missing data, Slicing and Dicing: Filtering and Selecting Data, Slicing rows, Slicing columns, Dicing, Concatenating and Transforming, Adding new cases and variables, Removing data, Sorting and shuffling, Aggregating Data at Any Level.

Shaping Data :

Working with HTML Pages, Parsing XML and HTML, Using XPath for data extraction, Working with Raw Text, Dealing with Unicode, Stemming and removing stop words, Introducing regular expressions, Using the Bag of Words Model and Beyond, Understanding the bag of words model, Working with ngrams, Implementing TF-IDF transformations, Working with Graph Data, Understanding the adjacency matrix, Using NetworkX basics. **(Unit - III)**

4. Data Visulization :

Visualizing Information :

Starting with a Graph, Defining the plot, Drawing multiple lines and plots, Saving your work to disk, Setting the Axis, Ticks, Grids, Getting the axes, Formatting the axes, Adding grids, Defining the Line Appearance, Working with line style, Using colors, Adding markers, Using Labels, Annotations, and Legends, Adding labels, Annotating the chart, Creating a legend.

Visualizing the Data :

Choosing the Right Graph, Showing parts of a whole with pie charts, Creating comparisons with bar charts, Showing distributions using histograms, Depicting groups using boxplots, Seeing data patterns using scatterplots, Creating Advanced Scatterplots, Depicting groups, Showing correlations, Plotting Time Series, Representing time on axes, Plotting trends over time, Plotting Geographical Data, Using an environment in Notebook, Getting the Basemap toolkit, Dealing with deprecated library issues, Using Basemap to plot geographic data, Visualizing Graphs, Developing undirected graphs, Developing directed graphs. **(Unit - IV)**

5. Data Wrangling :

Wrangling Data :

Playing with Scikit-learn, Understanding classes in Scikit-learn, Defining applications for data science, Performing the Hashing Trick, Using hash functions, Demonstrating the hashing trick, Working with deterministic selection, Considering Timing and Performance, Benchmarkin, with.timeit, Working with the memory profiler, Running in Parallel on Multiple Cores, Performing multicore parallelism, Demonstrating multiprocessing.

Exploring Data Analysis :

The EDA Approach, Defining Descriptive Statistics for Numeric Data, Measuring central tendency,Measuring variance and range ,Working with percentiles, Defining measures of normality, Counting for Categorical Data, Understanding frequencies, Creating contingency tables, Creating Applied Visualization for EDA ,Inspecting boxplots, Performing t-tests after boxplots, Observing parallel coordinates, Graphing distributions, Plotting scatterplots , Understanding Correlation, Using covariance and correlation, Using nonparametric correlation, Considering the chi-square test for tables, Modifying Data Distributions, Using different statistical distributions, Creating a Z-score standardization, Transforming other notable distributions. **(Unit - V)**

TABLE OF CONTENTS

Unit - I Overview of Python and Data Structures (1 - 1) to (1 - 14)

1.1 Introduction of Python ..... 1 - 2
1.1.1 Features of Python Programming ..... 1 - 3
1.1.2 Advantages & Disadvantages of Python ..... 1 - 3
1.2 Data Types & Variable ..... 1 - 4
1.3 Objects and Functions ..... 1 - 5
1.4 Python Data Structures ..... 1 - 7
1.4.1 Strings ..... 1 - 7
1.4.2 Array ..... 1 - 9
1.4.3 List ..... 1 - 9
1.4.4 Tuple ..... 1 - 10
1.4.5 Set ..... 1 - 11
1.4.6 Dictionary ..... 1 - 11
1.4.7 Difference between List and Tuple ..... 1 - 13

Unit - II Data Science and Python (2 - 1) to (2 - 16)

2.1 Discovering the Match between Data Science and Python ..... 2 - 2
2.1.1 Defining the Sexiest Job of the 21st Century ..... 2 - 2
2.1.2 Outlining the Core Competencies of a Data Scientist ..... 2 - 3
2.2 Creating the Data Science Pipeline ..... 2 - 4
2.3 Understanding Python's Role in Data Science ..... 2 - 5
2.3.1 Considering the Shifting Profile of Data Scientists ..... 2 - 5
2.3.2 Working with a Multipurpose, Simple, and Efficient Language ..... 2 - 6
2.4 Learning to Use Python Fast ..... 2 - 6
2.5 Introducing Python's Capabilities and Wonders : Why Python ? ..... 2 - 7
2.6 Working with Python ..... 2 - 8
2.6.1 Working at the Command Line or in the IDE ..... 2 - 9
2.6.2 Performing Rapid Prototyping and Experimentation ..... 2 - 10

2.6.3 Considering Speed of Execution ..... 2 - 11
2.7 Using the Python Ecosystem for Data Science ..... 2 - 11
2.7.1 Accessing Scientific Tools using SciPy ..... 2 - 11
2.7.2 Performing Fundamental Scientific Computing using NumPy ..... 2 - 12
2.7.3 Performing Data Analysis using Pandas ..... 2 - 12
2.8 Implementing Machine Learning using Scikit-learn ..... 2 - 13
2.8.1 Going for Deep Learning with Keras and TensorFlow ..... 2 - 13
2.8.2 Plotting the Data using Matplotlib ..... 2 - 14
2.8.3 Creating Graphs with NetworkX ..... 2 - 14
2.8.4 Parsing HTML Documents using Beautiful Soup ..... 2 - 15

Unit - III Getting Your Hands Dirty with Data (3 - 1) to (3 - 26)

3.1 Understanding the Tools : Using the Jupyter Console ..... 3 - 2
3.1.1 Magic Function ..... 3 - 4
3.1.2 Using Jupyter Notebook ..... 3 - 4
3.1.3 Restarting the Kernel ..... 3 - 5
3.1.4 Restoring a Checkpoint ..... 3 - 5
3.1.5 Performing Multimedia and Graphic Integration ..... 3 - 6
3.2 Working with Real Data : Uploading, Streaming, and Sampling Data ..... 3 - 7
3.2.1 Accessing Data in Structured Flat-File Form ..... 3 - 8
3.2.2 Reading from a Text File ..... 3 - 9
3.2.3 Sending Data in Unstructured File Form ..... 3 - 9
3.2.4 Managing Data from Relational Databases: ..... 3 - 10
3.3 Conditioning Your Data : Juggling between NumPy and Pandas ..... 3 - 11
3.3.1 Figuring out what's in Your Data ..... 3 - 12
3.3.2 Creating a Data Map and Data Plan ..... 3 - 13
3.3.3 Manipulating & Creating Categorical Variables ..... 3 - 13
3.3.4 Renaming Levels & Combining Levels ..... 3 - 14
3.3.5 Dealing with Dates and Times Values ..... 3 - 15
3.3.6 Finding the Missing Data ..... 3 - 16
3.4 Slicing and Dicing : Filtering and Selecting Data ..... 3 - 17
3.4.1 Concatenating and Transforming ..... 3 - 17

3.4.2 Sorting and Shuffling ..... 3 - 18
3.5 Shaping Data : Working with HTML Pages ..... 3 - 19
3.5.1 Parsing XML and HTML ..... 3 - 20
3.5.2 Using XPath for Data Extraction ..... 3 - 20
3.5.3 Dealing with Unicode ..... 3 - 21
3.5.4 Stemming and Removing Stop Words ..... 3 - 21
3.6 Introducing Regular Expressions ..... 3 - 22
3.7 Using the Bag of Words Model and Beyond ..... 3 - 24
3.7.1 Working with n-grams ..... 3 - 24
3.7.2 Implementing TF-IDF Transformations ..... 3 - 25
3.7.3 Understanding the Adjacency Matrix ..... 3 - 26

Unit - IV Data Visualization (4 - 1) to (4 - 22)

4.1 Visualizing Information : Starting with a Graph ..... 4 - 2
4.1.1 Drawing Multiple Lines and Plots ..... 4 - 3
4.1.2 Saving your Work To Disk ..... 4 - 3
4.1.3 Setting the Axis, Ticks, Grids ..... 4 - 4
4.1.4 Defining the Line Appearance & Working with Line Style ..... 4 - 5
4.1.5 Adding Markers ..... 4 - 6
4.1.6 Using Labels, Annotations, and Legends ..... 4 - 7
4.2 Visualizing the Data ..... 4 - 10
4.2.1 Creating Comparisons with Bar Charts ..... 4 - 11
4.3 Showing Distributions using Histograms ..... 4 - 12
4.4 Depicting Groups using Box Plots ..... 4 - 13
4.5 Seeing Data Patterns using Scatterplots ..... 4 - 14
4.5.1 Creating Advanced Scatterplots ..... 4 - 15
4.6 Representing Time on Axes ..... 4 - 16
4.7 Plotting Geographical Data ..... 4 - 17
4.8 Visualizing Graphs ..... 4 - 19

Unit - V Data Wrangling (5 - 1) to (5 - 24)

5.1 Introduction to Data Wrangling ..... 5 - 2
5.2 Playing with Scikit?Learn ..... 5 - 3
5.2.1 Understanding Classes in Scikit-learn ..... 5 - 3
5.2.2 Defining Applications for Data Science ..... 5 - 5
5.3 Performing the Hashing Trick ..... 5 - 5
5.3.1 Using hash Functions ..... 5 - 6
5.3.2 Performing the Hashing Trick ..... 5 - 6
5.3.3 Working with Deterministic Selection ..... 5 - 7
5.4 Considering Timing and Performance : Benchmarking with timeit ..... 5 - 8
5.4.1 Working with the Memory Profiler ..... 5 - 8
5.5 Running in Parallel on Multiple Cores ..... 5 - 9
5.5.1 Demonstrating Multiprocessing ..... 5 - 10
5.6 Exploring Data Analysis ..... 5 - 10
5.6.1 Defining Descriptive Statistics for Numeric Data ..... 5 - 11
5.6.2 Measuring Central Tendency ..... 5 - 11
5.6.3 Measuring Variance and Range ..... 5 - 13
5.6.4 Defining Measures of Normality ..... 5 - 14
5.6.5 Counting for Categorical Data ..... 5 - 14
5.6.6 Understanding Frequencies ..... 5 - 16
5.6.7 Creating Contingency Tables ..... 5 - 17
5.7 Creating Applied Visualization for EDA ..... 5 - 17
5.7.1 Inspecting Boxplots ..... 5 - 18
5.7.2 Performing t-tests after Boxplots ..... 5 - 19
5.7.3 Observing Parallel Coordinates ..... 5 - 20
5.8 Understanding Correlation ..... 5 - 20
5.9 Considering the Chi-Square Test for Tables ..... 5 - 22

Solved Model Question Paper (M - 1) to (M - 2)

Unit I

Overview of Python and Data Structures

Syllabus

Basics of Python including data types, variables, expressions, objects and functions. Python data structures including String, Array, List, Tuple, Set, Dictionary and operations them.

Contents

1.1 Introduction of Python

1.2 Data Types & Variable

1.3 Objects and Functions

1.4 Python Data Structures

1.1 Introduction of Python

- Python is a high-level scripting language which can be used for a wide variety of text processing, system administration and internet-related tasks.
- Python is a true object-oriented language, and is available on a wide variety of platforms.
- Python was developed in the early 1990's by Guido van Rossum, then at CWI in Amsterdam, and currently at CNRI in Virginia.
- Python 3.0 was released in Year 2008.
- Python statements do not need to end with a special character.
- Python relies on modules, that is, self-contained programs which define a variety of functions and data types.
- A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended.
- Within a module, the module's name (as a string) is available as the value of the global variable __name__.
- If a module is executed directly however, the value of the global variable __name__ will be "__main__".
- Modules can contain executable statements aside from definitions. These are executed only the first time the module name is encountered in an import statement as well as if the file is executed as a script.
- Integrated Development Environment (IDE) is the basic interpreter and editor environment that you can use along with Python. This typically includes an editor for creating and modifying programs, a translator for executing programs, and a program debugger. A debugger provides a means of taking control of the execution of a program to aid in finding program errors.
- Python is most commonly translated by use of an interpreter. It provides the very useful ability to execute in interactive mode. The window that provides this interaction is referred to as the Python shell.
- Python support two basic modes : ***Normal mode and interactive mode***
- Normal mode : The normal mode is the mode where the scripted and finished . py files are run in the Python interpreter. This mode is also called as script mode.
- Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory.

- Start the Python interactive interpreter by typing python with no arguments at the command line.
- To access the Python shell, open the terminal of your operating system and then type "python". Press the enter key and the Python shell will appear.

```
C:\Windows\system32>python
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- The >>> indicates that the Python shell is ready to execute and send your commands to the Python interpreter. The result is immediately displayed on the Python shell as soon as the Python interpreter interprets the command.
- For example, to print the text "Hello World", we can type the following :

```
>>> print("Hello World")
Hello World
>>>
```

- In script mode, a file must be created and saved before executing the code to get results. In interactive mode, the result is returned immediately after pressing the enter key.
- In script mode, you are provided with a direct way of editing your code. This is not possible in interactive mode.

1.1.1 Features of Python Programming

1. Python is a high-level, interpreted, interactive and object-oriented scripting language.
2. It is simple and easy to learn.
3. It is portable.
4. Python is free and open source programming language.
5. Python can perform complex tasks using a few lines of code.
6. Python can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh, etc
7. It provides a vast range of libraries for the various fields such as machine learning, web developer, and also for the scripting.

1.1.2 Advantages & Disadvantages of Python

Advantages of Python

- Ease of programming

- Minimizes the time to develop and maintain code
- Modular and object-oriented
- Large community of users
- A large standard and user-contributed library

Disadvantages of python

- Interpreted and therefore slower than compiled languages
- Decentralized with packages

1.2 Data Types & Variable

- A variable is a way of referring to a memory location used by a computer program. A variable is a symbolic name for this physical location. This memory location contains values, like numbers, text or more complicated types.
- A variable is a name that refers to a value. The equal (=) operator is used to assign value to a variable.
- Python's data types include : Numbers, Strings, Lists, Dictionaries, Tuples and Files.
- Python has no additional commands to declare a variable. As soon as the value is assigned to it, the variable is declared.

```
1  a = 20
2  # variable is declared as the value 20
```

- Rules for variables are as follows :
  a. Special characters are not allowed.
  b. Variables are case sensitive.
  c. Variable can only contain alpha-numeric characters and underscores.
  d. Variable name always start with character, not with number.
- Python allows us to assign a value to multiple variables in a single statement, which is also known as multiple assignments. For example :
  1) x = y = z = 51
  2) x, y, z = 11, 21, 31
- Numerical data types are Integers, Float, Complex Numbers and Boolean.
- Integers are used to represent whole number values. Float data type is used to represent decimal point values. Boolean is used for categorical output, since the output of boolean is either true or false.

- Consider the following example :

```
i = 13          # data type is implicitly set to integer
i = 13 + 0.7    # data type is changed to float
i = "fifty"     # and now it will be a string
```

- Python automatically takes care of the physical representation for the different data types, i.e. an integer values will be stored in a different memory location than a float or a string.
- **Int :** Integer value can be any length such as integers 11, 22, 9, -30, -151 etc. Python has no restriction on the length of an integer. Its value belongs to int.
- **Float :** Float is used to store floating-point numbers like 3.2, 4.99, 30.20 etc. It is accurate upto 15 decimal points.
- **Complex :** A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively. The complex numbers like 2.14j, 2.0 + 2.3j, etc.
- Python's basic number types support the normal mathematical operations. For instance, the plus sign (+) performs addition, a star (*) is used for multiplication, and two stars (**) are used for exponentiation.
- The operators supported by python are as follows :

Operator	Function	Operator	Function
+	Addition	-	Subtraction
*	Multiplication	/	Division
>>	Rigth bit shift	<<	Left bit shift
**	Exponentiation	%	Modulus

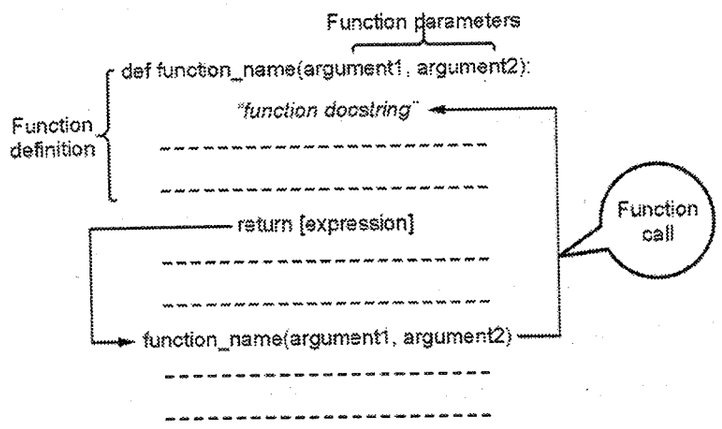
1.3 Objects and Functions

- In Python, functions are just one more type of object. Some functions are designed to return a value, while others are designed for other purposes.
- A function in Python is a logical unit of code containing a sequence of statements indented under a name given using the "def" keyword.
- Function is defined as follows:

```
def function_name(parameters):
    "function docstring"
```

```
    statement1
    statement2
    ...
    ...
    return [expr]
```

- There are two types of functions in Python.
  1. Built-in Functions : These are the built-in functions of Python with pre-defined functionalities.
  2. User-defined Functions : These are the functions defined by the users as per the requirement at different stages
  3. How does a function work in Python ?



- A function definition in Python starts with the keyword ***def*** followed by the function name. the first statement of the body function is a documentation string called doctsring.
- We can access the docstring by __doc__ attribute. For example function_name.__doc__.
- Example:

```
def Hello():
 """ Hello World """
 print ('Hi')
print ("The doctsring of the function Hello is: "+ Hello.__doc__)
```

- Output :

  The docstring of the function Hello is : Hello World
- The return statement is used to return values from a function. The return statement takes zero or more values, separated by commas. Using commas actually returns a single tuple. The default value is None.

- Example to find a sum of two numbers using function in Python

```
    #return type function
def add_r(a, b):
 """ return type function to find sum """
 c = a + b
 return c
#non return type function
def add_n(a, b):
 """ non return type function to return sum """
 cx = a + b
 print (c)
#call the return type function and print
print (add_r(10, 20))
#call the non return type function
add_n(10, 20)
```

Output:

```
30
30
```

1.4 Python Data Structures

- Python data structure include list, tuple, set, and dictionaries.

1.4.1 Strings

- Strings are a collection of characters which are stored together to represent arbitrary text inside a python program.
- Strings in python are used to represent unicode character values. Python does not have a character data type, a single character is also considered as a string.
- We denote or declare the string values inside single quotes or double quotes. To access the values in a string, we use the indexes and square brackets.
- The operator * is known as a repetition operator as the operation "Python" *2 returns 'Python Python'.
- String literals are written by enclosing a sequence of characters in single quotes ('Python'), double quotes ("Python") or triple quotes ("'Python'" or """Python""").
- Format strings consist of combined character data and markup. Character data is data which is transferred unchanged from the format string to the output string; markup is not transferred from the format string directly to the output.
- Example : "My name is {0}".format('Rakshita')
- The result of this is the string : "My name is Rakshita"

String Operations :

1. Concatenation : The addition operator (+) takes on the role of concatenation when used with strings. The result of "adding" two strings together is a new string which consists of the original strings.

- **Example:**

```
>>> first = 'Technical'
>>> second = 'Publications'
>>> first + second
'TechnicalPublications'
```

- No space or other character is inserted between concatenated strings.

```
>>> first + " " + second
'Technical Publications'
```

- Python will automatically concatenate string constants that are separated by whitespace:

```
>>> numbers = "First " "Second " "Third "
>>> numbers
'First Second Third '
```

2. Repetition

- The asterisk (*), when used between a string and an integer creates a new string with the old string repeated by the value of the integer.
- Example: following two statements will both print five characters

```
>>> 'A' * 5
    'AAAAA'
>>> 10 * 'A'
    'AAAAA'
```

3. Indexing and Slicing

- Python support indexing and slicing operations. To extract a single character from a string, follow the string with the index of the desired character surrounded by square brackets ([ ]), remembering that the first character of a string has index zero.
- Example :

```
>>> what = 'Python Book'
>>> what[5]
'n'
>>> what[0]
'P'
```


1.4.2 Array

- Python does not have built-in support for Arrays. Python lists are used to serve the purpose.
- A user can treat lists as arrays but cannot constrain the data type of values stored in a list. Only the same type of data can be stored in arrays.
- To create arrays in Python, we need to import the array module to create arrays. An example of this shown below:

```
import array as ary                    # importing the array module
b = ary.array('i',[2,4,6,8,10])        #defining an array of integers with 5 values
```

- Example :

```
import array as ary
b = ary.array('i', [2,4,6,8,10]) # defining an array of integers with 5 values
print("Value at index 2 is: ")
print(b[2])
```

- Output :

```
Value at index 2 is:
6
```

- **Example :**

```
colddrink = [ "coke", "pepsi", "fanta"]
colddrink1 = [ "mazza", "sprite"]
colddrink.Extend(colddrink1)
prints( colddrink)
Output:
"coke", "pepsi", "fanta", "mazza", "sprite"
```

1.4.3 List

- Lists provide a general mechanism for storing a collection of objects indexed by a number in python. A list is a linear data structure , meaning that its elements have a linear ordering.
- Each element can be accessed by an index. Note that Python indexes start with 0 instead of 1.
- Lists are a mutable collection of items. We use square brackets to surround the items [].
- The elements of the list are arbitrary, they can be numbers, strings, functions, user-defined objects or even other lists, making complex data structures very simple to express in python.

- Simple list of numbers can be created as follows :

```
>>> mylist = [1,7,9, 13, 22, 31]
```

- Indentation is very important in Python. Just like three chevrons >>> indicate an interactive prompt in Python, the three dots ... are Python's prompt for multiple lines.
- After creating a list in Python, it is possible to search, add, and remove items from the list.
- A list within another list is called nested list.
- Example : ['good',10,[100,99]]
- Lists are mutable, as they can be modified -
  1. Assign the new value by using the position of the data in the list
  2. Use the inbuilt functions to change or append the lists
- **Empty List :** A list that contains no elements is called empty list. It can be created with empty brackets, [].
- Example : Assigning List Values to Variables :

```
>>>numbers=[140, 21]
>>>characters=['a','b']
>>>print(numbers, characters)
```

- Output :

```
[140, 21]['a','b']
```

- There are no restrictions on the elements of lists. A list can contain a set or another list.

```
>>>  [[1,1+1,4-1],{2*2,5,6}, "yo"]
     [[1, 2, 3], {4, 5, 6}, 'yo']
```

1.4.4 Tuple

- A tuple is a sequence of values. The values can be any type and are indexed by integers, unlike lists. Tuples are immutable.
- Tuple is used to contain data that are related but not necessarily of the same type.
- Like a Python list, tuple supports random collection of objects. Tuple is a fixed length object.
- Tuple supports nesting of compound objects as in dictionary and list. As the tuples are immutable, you can't change the size of a tuple without making a copy if it.

- A tuple with a single element must have a comma inside the parentheses. tuple in the Python programming language is defined using parentheses with each item separated by commas.
- The general syntax of a tuple is :

```
Tuple1 = (item 1, item 2, item 3,....., item n)
```

- Tuples are used in scenarios where it is certain that the (set of) values belonging to some statement or a user-defined function will not change.
- Example :

```
>>> mytuple = (11, 22, 33)
>>>mytuple[0]
11
```

- Lets create Tuple named as item.

```
item = (0,'suresh','rupali',33.4)
```

- Tuple item contains objects of String, Integer and Float types. We can access these tuple elements in the same way we accessed elements in the list i.e. using indices inside square brackets.

```
T[1]      # This returns 'suresh'
```

1.4.5 Set

- Set is an unordered collection of simple objects in Python.
- You can use curly braces to give an expression whose value is a set. Python prints sets using curly braces.

```
>>> {1+2, 3, "a"}
{'a', 3}
>>> {2, 1, 3}
{1, 2, 3}
```

- Note that duplicates are eliminated and that the order in which the elements of the output are printed does not necessarily match the order of the input elements.

1.4.6 Dictionary

- Python provides collections, called dictionaries, that are suitable for representing such functions. Conceptually, a dictionary is a set of key-value pairs.
- The keys defined for a dictionary need to be unique. Though values in a dictionary can be mutable or immutable objects, only immutable objects are allowed for keys.

- A dictionary in Python is defined using key-value pairs, separated by commas, enclosed within curly braces. The key and value are separated using a colon. The general syntax of a dictionary is :

  d = {"Key1": "Value1", "Key2": "Value2"}

  a. Key and values are separated by a colon

  b. Paris of entries are separated by commas

  c. Dictionary is enclosed within curly braces

- Key and values in the dictionaries can be of any type, but the keys should be unique to that particular dictionary. The empty dictionary is denoted {}.
- In order to access any key-value pair, the key needs to be specified using the indexing operator. The del statement is used for deleting one or many key-value pairs from a dictionary. The in operator is there for checking whether a key-value pair exists in the dictionary or not.
- Python data structures include lists, dictionaries, tuples, and sets.

```
list = [4, 5, 6]
dictionary = {4: "four", 5: "five", 6: "six"}
tuple = (4, 5, 6)
set = {4, 5, 6}

print("list:", list)
print("dictionary:", dictionary)
print("tuple:", tuple)
print("set:", set)
```

- **Output :**

```
list: [4, 5, 6]
dictionary: {4: 'four', 5: 'five', 6: 'six'}
tuple: (4, 5, 6)
set: {4, 5, 6}
```

- Dictionaries don't support duplicate keys
- Dictionary keys must be immutable
- Dictionary values don't have the above restrictions.
- There are two ways to access the values in a dictionary; the square bracket notation, and the get() function.

1.4.7 Difference between List and Tuple

List	Tuple
Items are surrounded in square bracket [ ].	Items are surrounded in parenthesis ( ).
List objects are mutable	Tuple objects are Immutable.
Lists are slower than tuples.	Tuples are faster than list.
Synatx: list1 = [20, 'Ram', 33]	Syntax : tuple1 = (20, 'Ram', 33)
If the content is not fixed and keep on changing then we should go for list.	If the content is fixed and never changes then we should go for Tuple.
List object can not used as keys for dictionaries because keys should be hashable and immutable.	Tuple object can be used as keys for dictionaries because keys should be hashable and immutable

Notes

Unit II

Data Science and Python

Syllabus

Discovering the match between data science and python:
Defining the Sexiest Job of the 21st Century, Considering the emergence of data science, Outlining the core competencies of a data scientist, Linking data science, big data, and AI , Understanding the role of programming, Creating the Data Science Pipeline, Preparing the data, Performing exploratory data analysis, Learning from data, Visualizing, Obtaining insights and data products, Understanding Python's Role in Data Science, Considering the shifting profile of data scientists, Working with a multipurpose, simple, and efficient language, Learning to Use Python Fast ,Loading data, Training a model, Viewing a result.

Introducing Python's Capabilities and Wonders:
Why Python?, Grasping Python's Core Philosophy, Contributing to data science, Discovering present and future development goals, Working with Python, Getting a taste of the language, Understanding the need for indentation, Working at the command line or in the IDE, Performing Rapid Prototyping and Experimentation, Considering Speed of Execution, Visualizing Power, Using the Python Ecosystem for Data Science, Accessing scientific tools using SciPy, Performing fundamental scientific computing using NumPy, Performing data analysis using pandas, Implementing machine learning using Scikit-learn, Going for deep learning with Keras and TensorFlow, Plotting the data using matplotlib, Creating graphs with NetworkX, Parsing HTML documents using Beautiful Soup.

Contents

2.1 Discovering the Match between Data Science and Python
2.2 Creating the Data Science Pipeline
2.3 Understanding Python's Role in Data Science
2.4 Learning to Use Python Fast
2.5 Introducing Python's Capabilities and Wonders : Why Python ?
2.6 Working with Python
2.7 Using the Python Ecosystem for Data Science
2.8 Implementing Machine Learning using Scikit-learn

2.1 Discovering the Match between Data Science and Python

- Data science is a "concept to unify statistics, data analysis and their related methods" in order to "understand and analyze actual phenomena" with data. It employs techniques and theories drawn from many fields within the broad areas of mathematics, statistics, information science, and computer science.
- Choosing a data science language : here we discuss some of the language.
  1. Python : Python is at the top of all other languages and is the most popular language used by data scientists. Python is rapidly gaining mainstream appeal and becoming a more practical language to build products. Python is a powerful tool for medium-scale data processing. Python also has the advantage of a rich data community, offering vast amounts of toolkits and features.
  2. The Statistical Analysis System (SAS) language : SAS language is popular because it makes data analysis, business intelligence, data management, and predictive analytics easy. The SAS Institute originally created SAS as a means to perform statistical analysis.
  3. R language : R has been kicking around since 1997 as a free alternative to pricey statistical software, such as Matlab or SAS.
  4. SQL : Large organizations use some sort of relational database, which is normally accessible with SQL, to store their data.

2.1.1 Defining the Sexiest Job of the 21st Century

- This type of professional looks for patterns and trends in large sets of data, using a variety of tools, techniques and critical thinking to arrive at practical solutions to real-life data-centric problems.
- Harvard University labeled the profession "the sexiest job of the 21st century." And according to LinkedIn, the career has seen an exponential growth becoming the second fastest growing profession.
- Good data scientists will not just address business problems, they will pick the right problems that have the most value to the organization.
- Lacking in dedicated data scientist resources, many industries haven't found automated alternatives and aren't embracing the full potential of data science. Organizations might understand their data as a strategic asset, but often fail to make the jump to using it as actionable analytics.

Considering the emergence of data science

- Data science is emerging as a field that is revolutionizing science and industries alike. Work across nearly all domains is becoming more data driven, affecting both the jobs that are available and the skills that are required.
- As more data and ways of analyzing them become available, more aspects of the economy, society, and daily life will become dependent on data.
- It is imperative that educators, administrators, and students begin today to consider how to best prepare for and keep pace with this data-driven era of tomorrow.

2.1.2 Outlining the Core Competencies of a Data Scientist

- the data scientist requires knowledge of a broad range of skills in order to perform the required tasks. Data scientist requires the knowledge is the following area :

1. Data capture : The act of capturing data begins by managing a data source using database management skills. Most of the data available in raw data. So it is necessary that data scientist must requires data-modeling skills. Data Scientists need to know multiple modeling techniques, model validation, and model selection techniques. They also need to know how to deploy a validated model and monitor it to maintain the accuracy of results.
2. Analysis : It is necessary to perform analysis of the data. Using basic tools, analysis is performed.
3. Presentation : Graphical presentation of patterns helps to visualize.

Linking data science and big data

- Big data can be defined as very large volumes of data available at various sources, in varying degrees of complexity, generated at different speed i.e. velocities and varying degrees of ambiguity, which cannot be processed using traditional technologies, processing methods, algorithms, or any commercial off-the-shelf solutions.
- 'Big data' is a term used to describe collection of data that is huge in size and yet growing exponentially with time. In short, such a data is so large and complex that none of the traditional data management tools are able to store it or process it efficiently.
- It is necessary to perform ETL on data. The components of textual ETL (Extract , Transform, Load) processing are Textual ETL rules engine, User Interface, Taxonomies, output database.

- Textual ETL rules Engine takes large unstructured data and parse it to extract value for integration. Rules engines contain series of data processing steps and algorithms such as classification, clustering, affinity, proximity.
- Taxonomy integration, master data integration, metadata integration are some integration processing techniques are available in rules engine.
- User Interface helps business users to create and apply processing rules through drag drop and free-form text interface in different languages.
- After the ETL specialist finds the data, a programming language or other tool transforms it into a common format for analysis purposes. The loading process takes many forms.

Understanding the role of programming

- Data scientist may need to know several programming languages in order to achieve specific goals. For example, you may need SQL knowledge to extract data from relational databases. Python can help you perform data loading, transformation, and analysis tasks.
- Output Database in textual ETL is any RDBMS or NoSQL database. To integrate structured and unstructured database uses result sets which are key value pairs.
- NoSQL means Not Only SQL, it solves the problem of handling huge volume of data that relational databases cannot handle. NoSQL databases are schema free and are non-relational databases. Most of the NoSQL databases are open source.
- The immense datasets that data scientists rely on often require multiple levels of redundant processing to transform into useful processed data

2.2 Creating the Data Science Pipeline

- Data science pipelines are sequences of processing and analysis steps applied to data for a specific purpose.
- They're useful in production projects, and they can also be useful if one expects to encounter the same type of business question in the future, so as to save on design time and coding.

1. Preparing the data

- Big data enables organizations to gather, store, manage, and manipulate vast amounts of data at the right speed, and the right time, to gain insights.
- The raw data not only may vary substantially in format, but you may also need to transform it to make all the data sources cohesive and amenable to analysis

2. Performing exploratory data analysis

- Exploratory Data Analysis (EDA), also known as Data Exploration, is a step in the Data Analysis Process, where a number of techniques are used to better understand the dataset being used.
- By conducting EDA, you can turn an almost useable dataset into a completely useable dataset. The use of trial and error is part of the data science art.

3. Learning from data : Discovery is part of being a data scientist

4. Visualizing : Visualization means seeing the patterns in the data and then being able to react to those patterns. visualization is the presentation of quantitative information in a graphical form. In other words, data visualizations turn large and small datasets into visuals that are easier for the human brain to understand and process.

2.3 Understanding Python's Role in Data Science

- Data analysts are responsible for interpreting data and analyzing the results utilizing statistical techniques and providing ongoing reports.
- They develop and implement data analyses, data collection systems, and other strategies that optimize statistical efficiency and quality. They are also responsible for acquiring data from primary or secondary data sources and maintaining databases.
- Given the right data sources, analysis requirements, and presentation needs, you can use Python for every part of the data science pipeline. In fact, that's precisely what you do in this book. Every example uses Python to help you understand another part of the data science equation.

2.3.1 Considering the Shifting Profile of Data Scientists

- A data scientist simply makes the value of data using processes and tools that are machine-learning dependent. The person uses scientific methods and algorithms to gain insights and extract knowledge from structured and unstructured data.
- For example, a data scientist can use a data analytics and visualization application for extracting useful insights from data.
- Apart from excellent math skills, you will need to improve your model building skills, as well. This will help you to work with other people in solving challenging problems.

- From a business perspective, the necessity of fusing data science and application development is obvious : Businesses must perform various sorts of analysis on the huge databases it has collected, to make sense of the information and use it to predict the future.

2.3.2 Working with a Multipurpose, Simple, and Efficient Language

- The programming languages R and Python are perhaps the most famous tools in the data industry and it is very natural to ask the question which one do I choose.
- Python is a complete programming language which not only provides support for native data analytics algorithms but also has a rich library for web developments and software developments. Hence, it comes handy when one is looking for analytics embedded within a web page or a standalone application.
- Python support different coding styles :
  1. Functional : Treats every statement as a mathematical equation and avoids any form of state or mutable data.
  2. Imperative : Performs computations as a direct change to program state. This style is useful for manipulating data structures.
  3. Object-oriented : Relies on data fields that are treated as objects and manipulated only through prescribed methods. Its robust library allows coding professionals to choose from a large number of modules as per their specific requirements
  4. Procedural : Treats tasks as step-by-step iterations where common tasks are placed in functions that are called as needed. This coding style favours iteration, sequencing, selection, and modularization.

2.4 Learning to Use Python Fast

- Python programs generally are smaller than other programming languages where programmers have to type relatively less and indentation requirements of the language that makes them more readable all the time.
- Python has a tight grip within the data science community because of its rich repository of data science libraries. It allows data scientists to implement tasks with better stability, modularity, and code readability as it is an object-oriented language, with lots of extensions and farfetched community support.
  1. Loading Data : Loading data in python environment is the most initial step of analysing data. The code places the entire dataset in the variable and then places parts of that data in variables named X and y. Think of variables as you

would storage boxes. The variables are important because they make it possible to work with the data.

2. Training a model : In order to determine their accuracy, one can train the model using the given dataset and then predict the response values for the same dataset using that model and hence, find the accuracy of the model.

2.5 Introducing Python's Capabilities and Wonders : Why Python ?

- Python is said to be a simple, clear and intuitive programming language. Python codes are compiled line-by-line which makes debugging errors much easier and efficient.
- Python allows the code of other languages such as C, C++ to be embedded, which makes it much more powerful and versatile.
- Python has gone through a number of iterations and currently has two development paths.
- Python is a multi-paradigm programming language. the Python ecosystem is supported by various packages that extend and enhance the language.
- It is true that Python is an interpreted language and as such code may be often slower than compiled code tailored to a particular machine architecture. In that respect, although the source code in Python is interpreted on the fly, the main advantage is flexibility. This is an important point in the data science workflow as we are interested in the balance between implementation time versus execution.

Grasping Python's core philosophy

- Core concept was to create Python on any platform, simple, flexible and provides significant potential for extension. Python provides all these features and many more.
- Guido Van Rossum published the first version of Python code (version 0.9.0) at alt.sources in February 1991. This release included already exception handling, functions, and the core data types of list, dict, str and others. It was also object oriented and had a module system.
- Python version 1.0 was released in January 1994. The major new features included in this release were the functional programming tools lambda, map, filter and reduce, which Guido Van Rossum never liked.

- Six and a half years later in October 2000, Python 2.0 was introduced. This release included list comprehensions, a full garbage collector and it was supporting unicode.
- Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 (also known as "Python 3000" and "Py3K") was released. Python 3 is not backwards compatible with Python 2.x

Discovering present and future development goals

- Python is also better for data manipulation and repeated tasks.
- The future scope of Python is bright as it also helps in the analysis of a large amount of data through its high-performance libraries and tools.
- Python as a second language for developers who needed to create one-off code but who couldn't quite achieve their goals using a scripting language.
- Numerous programmers have increased the use of Python programming languages and it is certainly used worldwide. Python programmers would be the most demandable in the future of IT industries which makes Python future brighter.

2.6 Working with Python

- let us write a small program.

Python 2 :

```
>>> print " Technical Publication!"
```

Python 3:

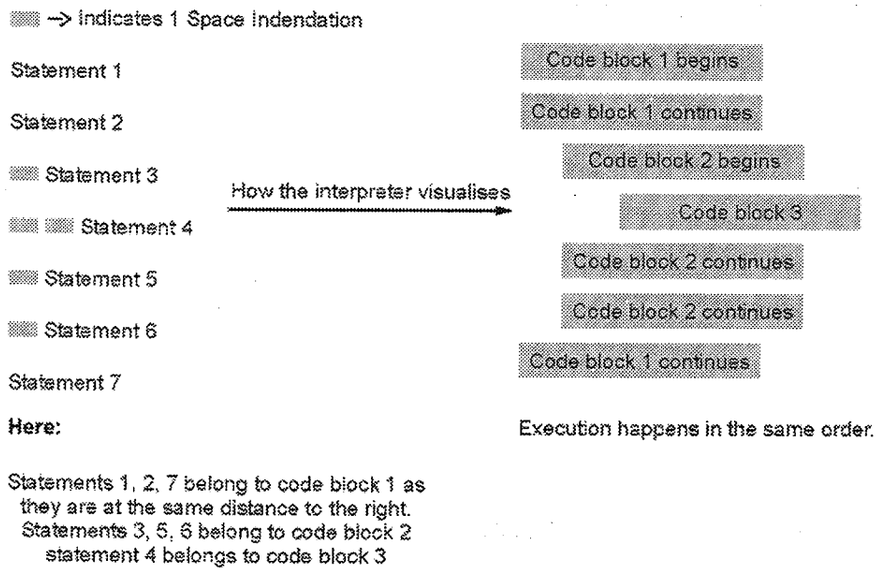
```
>>> print("Technical Publication !")
```

- If you're running Python 2.x, the print command is a statement rather than a function in Python 3.x.

1. Understanding the need for indentation

- In particular it is important to mention that in Python the whitespace is a meaningful character as it enables the definition of blocks of code by having the same level of indentation.
- Compilers/interpreters generally do not know the sequence in which they need to execute the statements in a piece of code. Hence, to make it easier, we divide the code into several blocks of code and indent it. This indentation helps them understand the order in which each block/statement should be executed.
- In Python, we use space/tab as indentation to indicate the same to the compiler.

- In simple, all the statements with the same distance (space) to the right, belong to the same block. In other words, statements belonging to a block will necessarily start from the same vertical line.



- By default, Python uses four spaces for indentation. However, the number of spaces you use is up to you, but a minimum of one space has to be used.

2.6.1 Working at the Command Line or in the IDE

- IPython is an interactive command-line terminal for Python. It was created by Fernando Perez in 2001. IPython offers an enhanced Read-Eval-Print Loop (REPL) environment particularly well adapted to scientific computing.
- IPython is closely tied with the Jupyter project, which provides a browser-based notebook that is useful for development, collaboration, sharing, and even publication of data science results.
- There are two primary means of using IPython : the IPython shell and the IPython notebook
- Spyder is a fully functional Integrated Development Environment (IDE). You use it to load scripts, edit them, run them, and perform debugging tasks. The Python Spyder IDE is written completely in Python.
- The Python Spyder IDE comes as a default implementation along with Anaconda Python distribution.
- Fig. 2.6.1 shows Spyder menu.

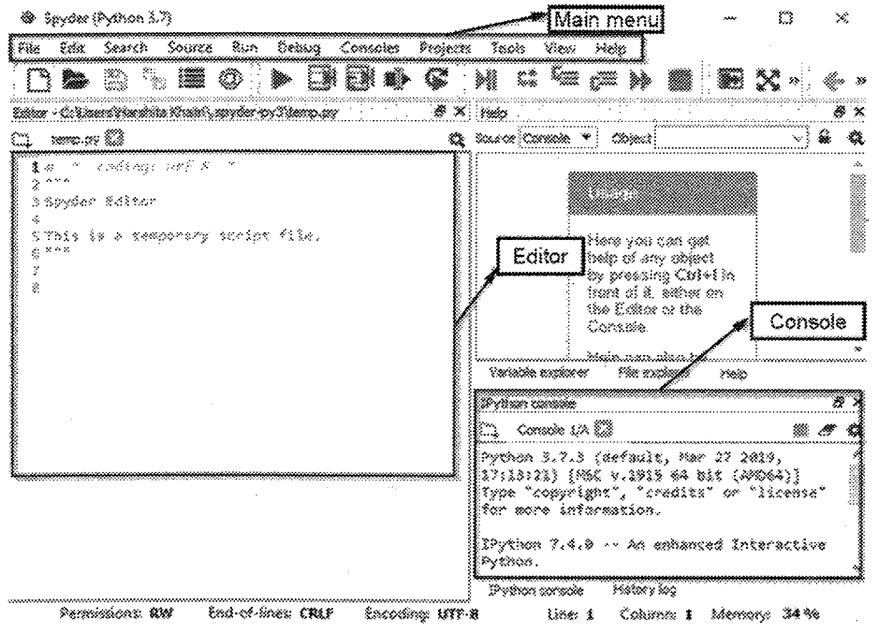


Fig. 2.6.1 Spyder Menu

- Writing code in Spyder becomes very easy with its multi-language code editor and a number of powerful tools.

2.6.2 Performing Rapid Prototyping and Experimentation

- The act of creating an application design incode without necessarily filling in all the details is prototyping. Python uses less code than other languages to perform tasks, so prototyping goes faster.
- Rapid prototyping languages share many of the same attributes as "Scripting" languages. Scripting can be thought of as a super-set of what might be done while prototyping.
- While prototyping focuses on rapid code development using higher level constructs, scripting focuses on rapid code development and the ability to "glue" existing components together.
- For example, the three most popular scripting languages: Perl, Python, and Tcl, all have well defined interfaces that allow large existing programs to be controlled and "programmed" via the native scripting language.

- The following list shows the phases in the order in which you normally perform them :
  1. Building a data pipeline: To work with the data, pipeline must be created. It is possible to load some data into memory.
  2. Performing the required shaping
  3. Analyzing the data
  4. Presenting a result

2.6.3 Considering Speed of Execution

- Following parameters are consider :

1. Size of Data : data science contain huge datasets. The application type determines the size of your dataset in part, but dataset size also relies on the size of the source data.
2. Loading technique : Loading techniques depends on storage places of data. Data is loaded in memory or disk. Working with data in memory is always faster than working with data stored on disk. Accessing local data is always faster than accessing it across a network.
3. Coding style : To create fast data science applications, you must use best-of-method coding techniques.
4. Machine capability: Running data science applications on a memory constrained system with a slower processor is impossible. The system you use needs to have the best hardware you can afford.
5. Analysis algorithm : The algorithm determines the kind of result you obtain and controls execution speed.

2.7 Using the Python Ecosystem for Data Science

2.7.1 Accessing Scientific Tools using SciPy

- SciPy contains many different packages and modules to assist in mathematics and scientific computing.
- It's difficult to state a single use case for SciPy considering that it contains so many different useful packages (including Numpy).

- Some of the important packages include :
  1. SciPy library : one of the core packages of the SciPy stack. This includes assistance with scientific computing, including those for numerical integration and optimization.
  2. Matplotlib: a 2D plotting library that can be used in Python scripts, the Python and IPython shell, web application servers, and more.
  3. IPython : An interactive console that runs your code like the Python shell, but gives you even more features, like support for data visualizations.

2.7.2 Performing Fundamental Scientific Computing using NumPy

- NumPy has risen to become one of the most popular Python science libraries and just secured a round of grant funding.
- NumPy's multidimensional array can perform very large calculations much more easily and efficiently than using the Python standard data types.
- To get started, NumPy has many resources on their website, including documentation and tutorials.
- NumPy (Numerical Python) is a perfect tool for scientific computing and performing basic and advanced array operations.
- The library offers many handy features performing operations on n-arrays and matrices in Python. It helps to process arrays that store values of the same data type and makes performing math operations on arrays easier. In fact, the vectorization of mathematical operations on the NumPy array type increases performance and accelerates the execution time.

2.7.3 Performing Data Analysis using Pandas

- Pandas is one of the most popular Python libraries in data science
- The pandas library provides support for data structures and data analysis tools. The library is optimized to perform data science tasks especially fast and efficiently.
- The basic principle behind pandas is to provide data analysis and modeling support for Python that is similar to other languages, such as R.
- Pandas is best suited for structured, labelled data, in other words, tabular data, that has headings associated with each column of data.
- Pandas has two core data structures used to store data: The Series and the DataFrame

- The series is a one-dimensional array-like structure designed to hold a single array (or 'column') of data and an associated array of data labels, called an index.
- The DataFrame represents tabular data, a bit like a spreadsheet. DataFrames are organised into colums and each column can store a single data-type, such as floating point numbers, strings, boolean values etc. DataFrames can be indexed by either their row or column names

2.8 Implementing Machine Learning using Scikit-learn

- Scikit-learn is probably the most useful library for machine learning in Python. It is on NumPy, SciPy and matplotlib, this library contains a lot of effiecient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.
- In scikit-learn, an estimator for classification is a Python object that implements the methods fit(X, y) and predict(T).
- It provides access to the following sorts of functionality:
  1. Classification
  2. Regression
  3. Clustering
  4. Dimensionality reduction
  5. Model selection
  6. Preprocessing

2.8.1 Going for Deep Learning with Keras and TensorFlow

- A Deep Neural Network is just a Neural Network with many layers stacked on top of each other – greater the number of layers, deeper the network
- Keras is a high-level neural networks API, capable of running on top of Tensorflow, Theano, and CNTK. It enables fast experimentation through a high level, user-friendly, modular and extensible API. Keras can also be run on both CPU and GPU.
- Keras provides seven different datasets, which can be loaded in using Keras directly. These include image datasets as well as a house price and a movie review datasets.
- Salient Features of Keras
  1. Keras is a high-level interface and uses Theano or Tensorflow for its backend.
  2. It runs smoothly on both CPU and GPU.

3. Keras supports almost all the models of a neural network – fully connected, convolutional, pooling, recurrent, embedding, etc. Furthermore, these models can be combined to build more complex models.
4. Keras, being modular in nature, is incredibly expressive, flexible, and apt for innovative research.
5. Keras is a completely Python-based framework, which makes it easy to debug and explore.

- TensorFlow is Google's gift to the developers involved in Machine Learning.
- TensorFlow is an open-sourced library that's available on GitHub. It is one of the more famous libraries when it comes to dealing with Deep Neural Networks.
- Some of the major applications of TensorFlow are :
  1. Tensorflow has been successfully implemented in DeepDream – the automated image captioning software – uses TensorFlow.
  2. Google's RankBrain, backed by TensorFlow, handles a substantial number of queries every minute and has effectively replaced the traditional static algorithm-based search.
  3. If you've used the Allo application, you must've seen a feature similar to Google's Inbox – you can reply to the last message from a few customized options. All thanks to Machine Learning with TensorFlow. Another feature analyses the images sent to you in order to suggest a relevant response.

2.8.2 Plotting the Data using Matplotlib

- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- Matplotlib is a is a plotting library for the Python programming language. It allows to make quality charts in few lines of code. Most of the other python plotting library are build on top of Matplotlib
- The library is currently limited to 2D output, but it still provides you with the means to express graphically the data patterns.

2.8.3 Creating Graphs with NetworkX

- NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

- Pygraphviz is a Python interface to the Graphviz graph layout and visualization package.
- It is possible to draw small graphs with NetworkX. You can export network data and draw with other programs
- When to use
  1. Unlike many other tools, it is designed to handle data on a scale relevant to modern problems
  2. Most of the core algorithms rely on extremely fast legacy code
  3. Highly flexible graph implementations

2.8.4 Parsing HTML Documents using Beautiful Soup

- Beautiful Soup is a Python library for pulling data out of HTML and XML files. Beautiful Soup helps you pull particular content from a webpage, remove the HTML markup, and save the information. It is a tool for web scraping that helps you clean up and parse the documents you have pulled down from the web.
- Importing the Beautiful Soup constructor function

```
from bs4 import BeautifulSoup
```

- The BeautifulSoup constructor function takes in two string arguments:
  a. The HTML string to be parsed.
  b. Optionally, the name of a parser.
- So, let's parse some HTML:

```
from bs4 import BeautifulSoup
htmltxt = "<p>Hello World</p>"
soup = BeautifulSoup(htmltxt, 'lxml')
```

- The BeautifulSoup object has a text attribute that returns the plain text of a HTML string. In the example of simple soup of Hello World , the text attribute returns :

```
soup.text
# 'Hello World'
```

❑❑❑

Notes

Unit III

Getting Your Hands Dirty with Data

Syllabus

Understanding the tools :

Using the Jupyter Console, Interacting with screen text, Changing the window appearance, Getting Python help, Getting IPython help, Using magic functions, Discovering objects, Using Jupyter Notebook, Working with styles, Restarting the kernel, Restoring a checkpoint, Performing Multimedia and Graphic Integration, Embedding plots and other images, Loading examples from online sites, Obtaining online graphics and multimedia.

Working with Real Data :

Uploading, Streaming, and Sampling Data, Uploading small amounts of data into memory, Streaming large amounts of data into memory, Generating variations on image data, Sampling data in different ways, Accessing Data in Structured Flat-File Form ,Reading from a text file Reading CSV delimited format, Reading Excel and other Microsoft Office files, Sending Data in Unstructured File Form, Managing Data from Relational Databases, Interacting with Data from NoSQL Databases, Accessing Data from the Web.

Conditioning Your Data :

Juggling between NumPy and pandas, Knowing when to use NumPy, Knowing when to use pandas, Validating Your Data, Figuring out what's in your data, Removing duplicates, Creating a data map and data plan, Manipulating Categorical Variables, Creating categorical variables, Renaming levels, Combining levels, Dealing with Dates in Your Data, Formatting date and time values, Using the right time transformation, Dealing with Missing Data, Finding the missing data, Encoding missingness, Imputing missing data, Slicing and Dicing: Filtering and Selecting Data, Slicing rows, Slicing columns, Dicing, Concatenating and Transforming, Adding new cases and variables, Removing data, Sorting and shuffling, Aggregating Data at Any Level.

Shaping Data :

Working with HTML Pages, Parsing XML and HTML, Using XPath for data extraction, Working with Raw Text, Dealing with Unicode, Stemming and removing stop words, Introducing regular expressions, Using the Bag of Words Model and Beyond, Understanding the bag of words model, Working with ngrams, Implementing TF-IDF transformations, Working with Graph Data, Understanding the adjacency matrix, Using NetworkX basics.

Contents

3.1 Understanding the Tools : Using the Jupyter Console

3.2 Working with Real Data : Uploading, Streaming, and Sampling Data

3.3 Conditioning Your Data : Juggling between NumPy and Pandas

3.4 Slicing and Dicing : Filtering and Selecting Data

3.5 Shaping Data : Working with HTML Pages

3.6 Introducing Regular Expressions

3.7 Using the Bag of Words Model and Beyond

3.1 Understanding the Tools : Using the Jupyter Console

- Jupyter is an open source project. The Jupyter console is a terminal frontend for kernels using the Jupyter protocol. The console can be installed with: pip install jupyter-console
- Jupyter is a web application perfect for this task. Jupyter works with Notebooks, documents that mix rich text including beautifully rendered math formulas, blocks of code and code output, including graphics.
- Jupyter main features are :
  1. inline code execution
  2. easy idea structuring
  3. nice displays of pictures and dataframe
- Jupyter Notebook is an open source web interface that enables to include text, video, audio, images
- The main difference between Jupyter console and Jupyter notebooks is that the console functions in interactive mode. Whenever you type a line of code, it is immediately executed, and you can see the results.
- If you want to write medium-length pieces of code, do a deep exploration of a dataset to tell a story, the notebook is better. If you want to test out code you're writing, or run quick commands, the console is better.
- A kernel is a program that runs and introspects the user's code: it provides computation and communication with the frontend interfaces, such as notebooks.
- The Jupyter Notebook Application has three main kernels : the IPython, IRkernel and IJulia kernels.
- Anaconda is the most widely used Python distribution for data science and comes pre-loaded with all the most popular libraries and tools.
- There are quite a number of packages that will help you to deploy your notebooks and that are part of the Jupyter ecosystem.
  1. docker-stacks will come in handy when user need stacks of Jupyter applications and kernels as Docker containers.
  2. ipywidgets provides interactive HTML & JavaScript widgets for the Jupyter architecture that combine front-end controls coupled to a Jupyter kernel.
  3. jupyter-drive allows IPython to use Google Drive for file management.

  4. jupyter-sphinx-theme to add a Jupyter Sphinx theme to your notebook. It will make it easier to create intelligent and beautiful documentation.
  5. kernel_gateway is a web server that supports different mechanisms for spawning and communicating with Jupyter kernels. Look here to see some use cases in which this package can be come in handy.
- To create a new notebook, click on the New button, and select Notebook. A new browser tab opens and shows the Notebook interface as follows:

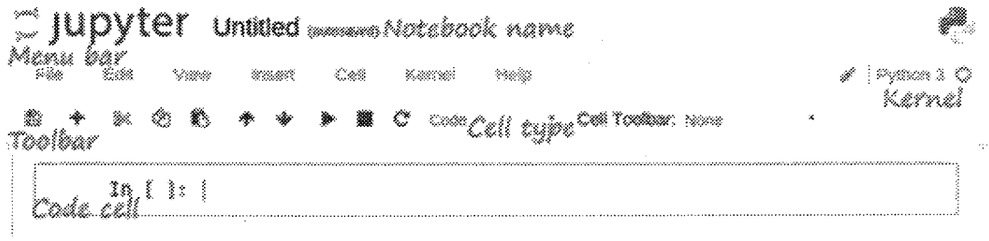
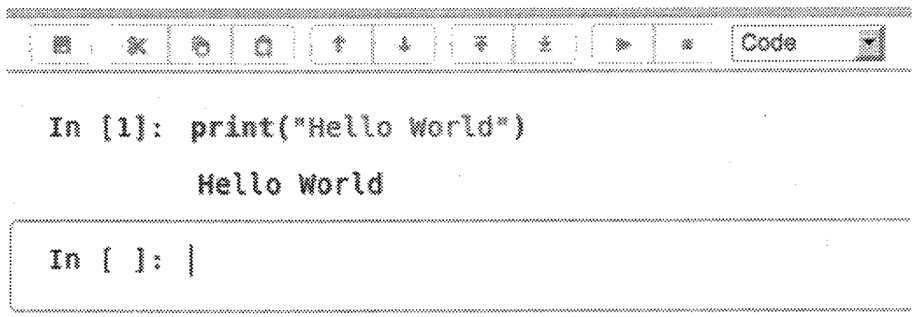


Fig. 3.1.1 Jupiter Notebook

- Notebook consists of two cells : Markdown cells and code cell.
- Markdown cell is used for text and code cell for executing code by the kernel.
- Code cell contains various section like prompt number, input area, widget area and output area.
- Notebook model interface support edit mode and command mode.
- Try typing something like print("Hello World") into the cell. To run the code in the cell and see the output, click the Run button on the toolbar, or type Shift-Enter:



- A Jupyter notebook is always organized as a sequence of so called 'cells' with each cell either containing some code or rich text created using the Markdown notation approach.

- When you click into the corresponding text field to add or modify the content of the cell, the bar color will change to green indicating that you are now in 'Edit mode'. Clicking anywhere outside of the text area of a cell will change back to 'Command mode'.

3.1.1 Magic Function

- Jupyter provides a number of so-called magic commands that can be used in code cells to simplify common tasks. Magic commands are interpreted by Jupyter and, for instance, transformed into Python code before the content is passed on to the kernel for execution
- Jupyter has a set of predefined 'magic functions' that you can call with a command line style syntax. There are two kinds of magics, line-oriented and cell-oriented.
- Line magics are prefixed with " % "and work like a command typed in your terminal. Line magics can be used as expression and their return value can be assigned to variable.
- Example of line magics are %autocall, %cd, %automagic, %dhist, %edit, %matplotlib and %env.
- Cell magics work on a block of code, not on a single line. They are prefixed with "%%". To close a block of code, when you are inside a cell magic function, hit Enter twice.

3.1.2 Using Jupyter Notebook

- The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text.
- Jupyter Notebook is often used for exploratory data analysis and visualization. The Jupyter Notebook has several menus that you can use to interact with your Notebook. Here is a list of some current menus:

  a) Edit b) View C) Insert D) Cell E)Kernel f) Widgets G)Help H)File
- File menu is used for creating new notebook and opening existing one.
- Edit menu perform operation like cut, copy, and paste cells.
- View menu is useful for toggling the visibility of the header and toolbar.
- Insert menu is for inserting cells above or below the currently selected cell.
- Cell menu allows you to run one cell, a group of cells, or all the cells.
- The Widgets menu is for saving and clearing widget state.
- The Kernel cell is for working with the kernel that is running in the background

3.1.3 Restarting the Kernel

- Juypter has a autocomplete feature that helps with the code writing and can save a lot of typing: While editing code in a code cell, you can press the TAB key and Jupyter will either automatically complement the name or keyword you are writing or provide you with a dropdown list of choices that you can pick from.
- Fig. 3.1.2 shows restarting the kernel.

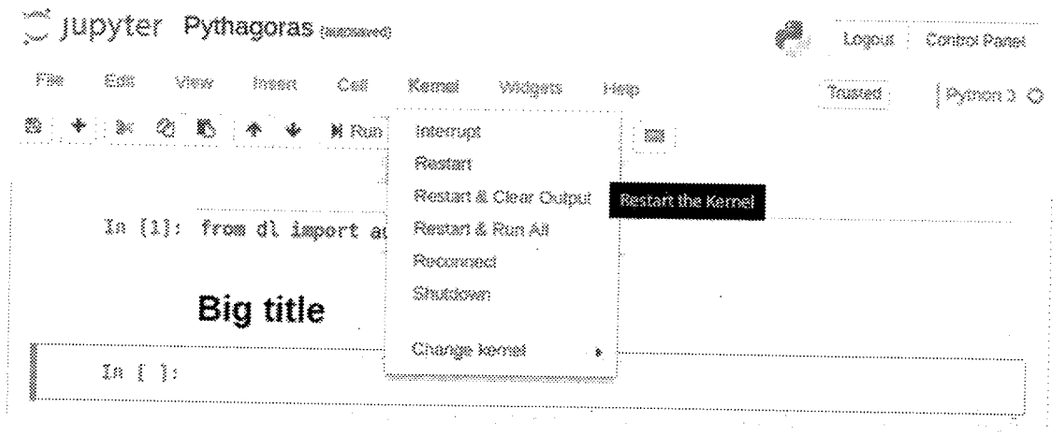


Fig. 3.1.2 Restarting the kernel

- Save your notebook locally to store your current progress
- In the notebook toolbar, click Kernel, then Restart
- Try testing your kernel by running a print statement in one of your notebook cells. If this is successful, you can continue to save and proceed with your work.
- If your notebook kernel is still timed out, try closing your browser and relaunching the notebook. When the notebook reopens, you will need to do "Cell -> Run All" or "Cell -> Run All Above" to regenerate the execution state.

3.1.4 Restoring a Checkpoint

- In Jupyter Notebook Workspaces, it's a good idea to save your checkpoints before you close the tab.
- To restore a checkpoint, choose the entry found on the File → Revert to Checkpoint menu.
- This menu makes it appear that you can have more than one checkpoint file, but the menu never has more than one entry in it.
- Fig. 3.1.4 shows menu for restoring a checkpoint.

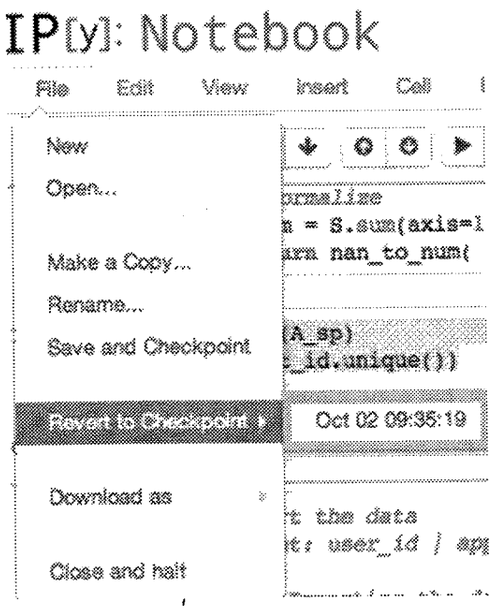


Fig. 3.1.3 Restoring a checkpoint

3.1.5 Performing Multimedia and Graphic Integration

- IPython Notebook support both a coding platform and a presentation platform.

Embedding plots and other images

- When you run %matplotlib inline, any plots you create appear as part of the document. Use %matplotlib inline - this only draws the images, not interactive / zoom-able but it works well.
- Following Fig. 3.1.4 shows example of embedding plot.

```
In [5]: 1 %matplotlib inline
        2 import matplotlib.pyplot as plt
        3 plt.plot([0.0,1.0],[0.0,1.0], linewidth=4, label='random diagonal')
        4 plt.show()
```

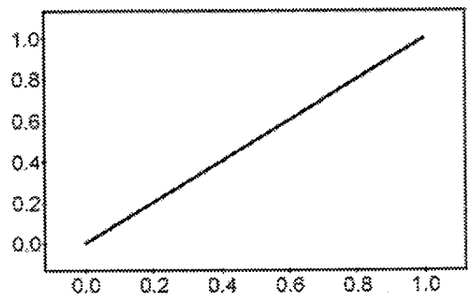


Fig. 3.1.4 Example of %matplotlib inline

- The display function is a general purpose tool for displaying different representations of objects. Think of it as print for these rich representations.

```
In [1]: from IPython.display import display
```

A few points :

a) Calling display on an object will send **all** possible representations to the Notebook.

b) These representations are stored in the Notebook document.

c) In general the Notebook will use the richest available representation.

- To work with images, use the Image class.

```
In [3]: from IPython.display import Image
In [4]: i = Image(url='img/python-logo.gif')
```

3.2 Working with Real Data : Uploading, Streaming, and Sampling Data

- If data is stored in local computer memory, then access is faster and reliable also. Normally user does not interact with storage location. But data is loaded into memory from storage location then processing starts.
- Scikit-learn library stores toy dataset. In data science, columns call as variables and rows are cases. Each row represents a collection of variables that you can analyze.

1. Uploading small amounts of data into memory

- Convenient method to work with data is when it directly load into main memory. Consider following file
- When you load a file, the entire dataset is available at all times. Syntax of loading file:

```
with open("file_name", 'access_mode') as open_file:
print 'file_name content:\n' + open_file.read()
```

- It start with open( ) command on file object. The open() function accepts the filename and an access mode. It perform the action according to the access mode, i.e. read, write.
- The entire dataset is loaded from the library into free memory.

1. Streaming large amounts of data into memory

- Large data set can not be loaded into memory because of memory size. Remote data set is loads slowly because of network connection. Streaming answers both needs by making it possible to work with the data a little at a time

- stream data using Python as follows :

```
with open("file_name", 'Access_mode') as open_file:
        for    observation in open_file:
                print 'Reading Data: ' + observation
```

- The open_file file object contains a pointer to the open file.

3.2.1 Accessing Data in Structured Flat-File Form

- Flat files are data files that contain records with no structured relationships between the records. A flat file presents the easiest kind of file to work with.
- A flat file can be a plain text file having a TSV, CSV format, or a binary file format.
- Comma Separated Values (CSV) files, which contain data values that are separated by ,. For example, a file saved with name "Data" in "CSV" format will appear as "Data.csv". By noticing ".csv" extension we can clearly identify that it is a "CSV" file and data is stored in a tabular format.
- The CSV file used for an example is quite simple :
  a) A header defines each of the fields
  b) Fields are separated by commas
  c) Records are separated by linefeeds
  d) Strings are enclosed in double quotes
  e) Integers and real numbers appear without double quotes
- Applications such as Excel can import and format CSV files so that they become easier to read.
- An **Excel file** uses a complex method to separate data fields and to provide a wealth of information about each field. Excel actually recognizes the header as a header.
- It is an XML-based file format created by Microsoft Excel. The XLSX format was introduced with Microsoft Office 2007.
- In XLSX data is organized under the cells and columns in a sheet. Each XLSX file may contain one or more sheets. So a workbook can contain multiple sheets.
- Let's load the data from XLSX file and define the sheet name. For loading the data you can use the Pandas library in python.

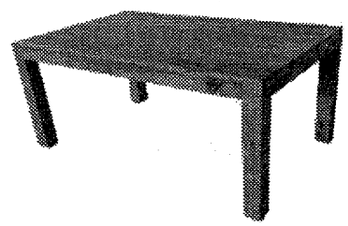
```
# read Excel file into a DataFrame
df = pd.read_excel(r'./Importing files/Indiacity.xlsx')
# print values
df
```


3.2.2 Reading from a Text File

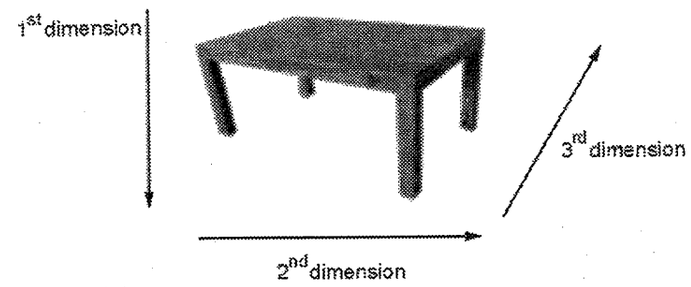
- Python provides three functions to read data from a text file :
  a) read(n) : This function reads n bytes from the text files or reads the complete information from the file if no number is specified. It is smart enough to handle the delimiters when it encounters one and separates the sentences
  b) readline(n) : This function allows you to read n bytes from the file but not more than one line of information
  c) readlines() : This function reads the complete information in the file but unlike read(), it doesn't bother about the delimiting character and prints them as well in a list format
- The following structure is typical for a CSV file:
  a) Definition of the columns in the header of the table
  b) A character is used to separate individual records
  c) A character is used to separate individual columns (commas, tabs, or spaces)
  d) Field delimiter HTML Special Character within the file to avoid confusion with the separators

3.2.3 Sending Data in Unstructured File Form

- Unstructured data files consist of a series of bits. The file doesn't separate the bits from each other in any way.
- Skimage provides easy-to-use functions for reading, displaying, and saving images. All of the popular image formats, such as BMP, PNG, JPEG, and TIFF are supported.
- Consider following wooden table.



- A visual representation of how this image is stored as a NumPy array is as follows :




```
import skimage.io
# read image
image = skimage.io.imread(fname="wtable.jpg")
import skimage.viewer
# display image
viewer = skimage.viewer.ImageViewer(image)
viewer.show()
```

3.2.4 Managing Data from Relational Databases

- Relational databases accomplish both the manipulation and data retrieval objectives. SQL perform all sorts of management tasks in a relational database, retrieve data as needed.
- Relational database consists of one or more tables of information. The rows in the table are called records and the columns in the table are called fields or attributes. A database that contains two or more related tables is called a relational database.
- When working on a data science project, you may want to connect Python scripts with databases. A library known as SQLAlchemy bridges the gap between SQL and Python.
- The first thing to do is to create an engine which is an object that is used to manage the connection to the database.

```
from sqlalchemy import create_engine
from sqlalchemy.orm import scoped_session, sessionmaker
engine = create_engine("postgresql://login:password@localhost:5432/flight")
```

- The general format to create an engine is:

```
create_engine("postgresql://login:passeword@localhost:5432/name_database")
```

- The sqlalchemy library provides support for SQL databases like SQLite, MySQL, PostgreSQL and SQL Server

Interacting with Data from NoSQL Databases

- The Not only SQL (NoSQL) databases are used in large data storage scenarios in which the relational model can become overly complex.
- NoSQL databases provide features of retrieval and storage of data in a much different way than their relational database counterparts.
- The first thing that we need to do in order to establish a connection is import the MongoClient class. We'll use this to communicate with the running database instance. Use the following code to do so :

```
from pymongo import MongoClient
client = MongoClient()
```

3.3 Conditioning Your Data : Juggling between NumPy and Pandas

1. NumPy

- Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays
- NumPy is the fundamental package needed for scientific computing with Python. It contains :
  a) A powerful N-dimensional array object
  b) Basic linear algebra functions
  c) Basic Fourier transforms
  d) Sophisticated random number capabilities
  e) Tools for integrating Fortran code
  f) Tools for integrating C/C++ code
- NumPy is an extension package to Python for array programming. It provides "closer to the hardware" optimization, which in Python means C implementation.

2. Pandas

- Pandas is a high-level data manipulation tool developed by Wes McKinney. It is built on the Numpy package and its key data structure is called the DataFrame.
- DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables.
- Pandas is built on top of the NumPy package, meaning a lot of the structure of NumPy is used or replicated in Pandas. Data in pandas is often used to feed

statistical analysis in SciPy, plotting functions from Matplotlib, and machine learning algorithms in Scikit-learn.

- Pandas is the library for data manipulation and analysis. Usually, it is the starting point for your data science tasks. It allows you to read/write data from/to multiple sources. Process the missing data, align your data, reshape it, merge and join it with other data, search data, group it, slice it.

3.3.1 Figuring out what's in Your Data

- Duplicate data creates the problem for data science project. If database is large, then processing duplicate data means wastage of time.
- Finding duplicates is important because it will save time, space false result. how to easily and efficiently you can remove the duplicate data using drop_duplicates() function in pandas.
- Create Dataframe with Duplicate data

```
import pandas as pd
raw_data = {'first_name': ['rupali', 'rupali', 'rakshita','sangeeta', 'mahesh', 'vilas'],
      'last_name': ['dhotre', 'dhotre', 'dhotre','Auti', 'jadhav', 'bagad'],
      'RNo': [12, 12, 1111111, 36, 24, 73],
      'TestScore1': [4, 4, 4, 31, 2, 3],
      'TestScore2': [25, 25, 25, 57, 62, 70]}
df = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age', 'preTestScore',
'postTestScore'])
df
```

	first_name	last_name	RNo	TestScore1	TestScore2
0	rupali	dhotre	12	4	25
1	rupali	dhotre	12	4	25
2	rakshita	dhotre	1111111	4	25
3	sangeeta	auti	36	31	57
4	mahesh	jadhav	24	2	62
5	vilas	bagad	73	3	70

Drop duplicates

```
df.drop_duplicates()
```

- Drop duplicates in the first name column, but take the last observation in the duplicated set

```
df.drop_duplicates(['first_name'], keep='last')
```


3.3.2 Creating a Data Map and Data Plan

- Overview of dataset is given by data map. Data map is used for finding potential problems in data, such as redundant variables, possible errors, missing values and variable transformations.
- Try creating a Python script that converts a Python dictionary into a Pandas DataFrame, then print the DataFrame to screen.

```
import pandas as pd

scottish_hills = {'Ben Nevis': (1345, 56.79685, -5.003508),
          'Ben Macdui': (1309, 57.070453, -3.668262),
          'Braeriach': (1296, 57.078628, -3.728024),
          'Cairn Toul': (1291, 57.054611, -3.71042),
          'Sgòr an Lochain Uaine': (1258, 57.057999, -3.725416)}

dataframe = pd.DataFrame(scottish_hills)
print(dataframe)
```

3.3.3 Manipulating & Creating Categorical Variables

- Categorical variable is one that has a specific value from a limited selection of values. The number of values is usually fixed.
- Categorical features can only take on a limited, and usually fixed, number of possible values. For example, if a dataset is about information related to users, then you will typically find features like country, gender, age group, etc. Alternatively, if the data you are working with is related to products, you will find features like product type, manufacturer, seller and so on.
- Method for creating a categorical variable and then using it to check whether some data falls within the specified limits.

```
import pandas as pd
cycle_colors = pd.Series(['Blue', 'Red', 'Green'], dtype='category')
cycle_data = pd.Series( pd.Categorical(['Yellow', 'Green', 'Red', 'Blue', 'Purple'],
                    categories=cycle_colors, ordered=False))
find_entries = pd.isnull(cycle_data)
print cycle_colors
print
print cycle_data
print
print find_entries[find_entries == True]
```

- Here cycle_color is a categorical variable. It contains the values Blue, Red, and Green as color.

3.3.4 Renaming Levels & Combining Levels

- Data frame variable names are typically used many times when wrangling data. Good names for these variables make it easier to write and read wrangling programs.
- Categorical data has a categories and a ordered property, which list their possible values and whether the ordering matters or not.
- Renaming categories is done by assigning new values to the Series.cat.categories property or by using the Categorical.rename_categories() method:

```
In [41]: s = pd.Series(["a","b","c","a"], dtype="category")
In [41]: s
Out[43]:
0  a
1  b
2  c
3  a
dtype: category
Categories (3, object): [a, b, c]

In [44]: s.cat.categories = ["Group %s" % g for g in s.cat.categories]

In [45]: s
Out[45]:
0  Group a
1  Group b
2  Group c
3  Group a
dtype: category
Categories (3, object): [Group a, Group b, Group c]

In [46]: s.cat.rename_categories([1,2,3])
Out[46]:
0  1
1  2
2  3
3  1
dtype: category
Categories (3, int64): [1, 2, 3]
```


3.3.5 Dealing with Dates and Times Values

- Dates are often provided in different formats and must be converted into single format DateTime objects before analysis.
- Python provides two methods of formatting date and time.
  1. str() = it turns a datetime value into a string without any formatting.
  2. strftime() function = it define how user want the datetime value to appear after conversion.

1. Using pandas.to_datetime() with a date

```
import pandas as pd
# input in mm.dd.yyyy format
date = ['21.07.2020']
# output in yyyy-mm-dd format
print(pd.to_datetime(date))
```

2. Using pandas.to_datetime() with a date and time

```
import pandas as pd
# date (mm.dd.yyyy) and time (H:MM:SS)
date = ['21.07.2020 11:31:01 AM']
# output in yyyy-mm-dd HH:MM:SS
print(pd.to_datetime(date))
```

- We can convert a string to datetime using strptime() function. This function is available in datetime and time modules to parse a string to datetime and time objects respectively.
- Python strptime() is a class method in datetime class. Its syntax is:

```
datetime.strptime(date_string, format)
```

- Both the arguments are mandatory and should be string

```
import datetime
format = "%a %b %d %H:%M:%S %Y"
today = datetime.datetime.today()
print 'ISO     :', today
s = today.strftime(format)
print 'strftime:', s
d = datetime.datetime.strptime(s, format)
print 'strptime:', d.strftime(format)
$ python datetime_datetime_strptime.py

ISO     : 2013-02-21 06:35:45.707450
strftime: Thu Feb 21 06:35:45 2013
strptime: Thu Feb 21 06:35:45 2013
```

- **Time Zones** : Within datetime, time zones are represented by subclasses of tzinfo. Since tzinfo is an abstract base class, you need to define a subclass and provide appropriate implementations for a few methods to make it useful.

3.3.6 Finding the Missing Data

- Data can have missing values for a number of reasons such as observations that were not recorded and data corruption. Handling missing data is important as many machine learning algorithms do not support data with missing values.
- You can load the dataset as a Pandas DataFrame and print summary statistics on each attribute.

```
# load and summarize the dataset
from pandas import read_csv
# load the dataset
dataset = read_csv('csv file name', header=None)
# summarize the dataset
print(dataset.describe())
```

- In Python, specifically Pandas, NumPy and Scikit-Learn, we mark missing values as NaN. Values with a NaN value are ignored from operations like sum, count, etc.
- Use the isnull() method to detect the missing values. Pandas Dataframe provides a function isnull(), it returns a new dataframe of same size as calling dataframe, it contains only True & False only. With True at the place NaN in original dataframe and False at other places.

Encoding missingness:

- The fillna() function is used to fill NA/NaN values using the specified method.
- Syntax:

```
DataFrame.fillna(value=None, method=None, axis=None, inplace=False, limit=None, downcast=None, **kwargs)
```

Where

1. value : It is a value that is used to fill the null values.
2. method : A method that is used to fill the null values.
3. axis : It takes int or string value for rows/columns.
4. inplace : If it is True, it fills values at an empty place.
5. limit : It is an integer value that specifies the maximum number of consecutive forward/backward NaN value fills.
6. downcast : It takes a dict that specifies what to downcast like Float64 to int64.

3.4 Slicing and Dicing : Filtering and Selecting Data

- Selecting single values from a list is just one part of the story. It's also possible to slice your list, which means selecting multiple elements from your list.
- The slice() function returns a slice object that can use used to slice strings, lists, tuple etc. The slice object is used to slice a given sequence (string, bytes, tuple, list or range) or any object which supports sequence protocol.
- When selecting data within Python, it's important to remember rows and columns start with 0 and selections of multiple columns are separated with commas (e.g. [0,4,5]).
- In addition, the selection of consecutive columns uses a colon to include every observation, except the very last number. So, [0:3] would return the first, second and third rows, but not the fourth.
- Dicing : The act of dicing a dataset means to perform both row and column slicing such that you end up with a data wedge.
- how to delete single or multiple rows from a DataFrame object : DataFrame provides a member function drop().

```
DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')
```

- It accepts a single or list of label names and deletes the corresponding rows or columns. As default value for axis is 0, so for dropping rows we need not to pass axis.
- Also, by default drop() doesn't modify the existing DataFrame, instead it returns a new dataframe. If we want to update the existing DataFrame in place then we need to pass another attribute.

3.4.1 Concatenating and Transforming

- In data science project, data is collected from various location with multiple databases. All locations have their own format. It is difficult to perform data analysis on such disparate sources of information with any accuracy.

Adding new cases and variables

- Dataset object provides access to the case data and variable information contained in a dataset, and allows you to read from the dataset, add new cases, modify existing cases, add new variables, and modify properties of existing variables.

- **Append Method :** The append() method in python adds a single item to the existing list. It doesn't return a new list of items but will modify the original list by adding the item to the end of the list.
- After executing the method append on the list the size of the list increases by one.
- The append() method takes a single item as an input parameter and adds that to the end of the list. The items inside a list can be numbers, strings, another list, dictionary.
- The append() method only modifies the original list. It doesn't return any value as a return but will just modify the created list.
- Both append() & extends() functions are used for string manipulation, i.e. adding a new element to the list. But they have different behaviour.
- Example :

```
import pandas as pd
mylist = [10, 20, 30]
print("Orignal list: ",mylist);
mylist.append([80, 40])
print("Append list: ",mylist)
```

Output:

```
Orignal list: [1, 2, 3]
Append list: [1, 2, 3, [8, 4]]
```

3.4.2 Sorting and Shuffling

- The shuffle() method takes a sequence (list, string, or tuple) and reorganize the order of the items.
- With a DataFrame, you can sort by index on either axis :

```
In [11]: frame = DataFrame(np.arange(8).reshape((2, 4)), index=['three', 'one'],
   ...:                    columns=['d', 'a', 'b', 'c'])
In [12]: frame.sort_index()
Out[12]:
       d a b c
one    4 5 6 7
three  0 1 2 3
```

```
In [13]: frame.sort_index(axis=1)
Out[13]:
         a b c d
three 1 2 3 0
one   5 6 7 4
```

- To sort the data, you use the sort_index() method and define which columns to use for indexing purposes. You can also determine whether the index is in ascending or descending order

- The random.shuffle() method will shuffle a list in Python. It uses the random module to reorder every item in a Python list. The shuffle can be controlled with an optional second parameter.
- random.shuffle() is part of the random library, which implements pseudo-random number generators in Python. This means that before we can use the random.shuffle() method, we need to import the random library into our code. We can do so by adding the following to our code: ***import random***
- The syntax for the random.shuffle() method is as follows :

```
import random
random.shuffle(list_name, function)
```

- The random.shuffle() method takes in two parameters :
  1. list_name: the list, tuple, or string whose order you want to shuffle (required)
  2. function: the name of a function that returns a random float between 0.0 and 1.0, or another float within that range (optional)

3.5 Shaping Data : Working with HTML Pages

- HTML specifies the contents and layout of web pages. The content contains text, table, form, image, links, information for search engine, etc. The layout is in the form of text format, background and frame. HTML is also used to specify links and which resources are associated with them.
- Hyper Text Markup Language (HTML) is intended as a common medium for typing together information from widely different sources.
- HTML documents are the Standard Generalized Markup Language (SGML) documents with generic semantic that are appropriate for representing information from a wide range of applications.
- HTML documents are in plain text format that contain embedded HTML tags. Documents can be created in any text editor. There are also many other tools, including editors, designed specifically to assist in creating HTML documents. To view an HTML document, the user needs a browser.
- HTML defines the structural elements in a document such as headers, and addresses, layout information and the use of inline graphics together with the ability to provide hyper text links. Web pages were written in HTML level 0.

3.5.1 Parsing XML and HTML

- XML stands for eXtensible Markup Language. XML is a markup language for documents containing structured information. XML is a set of rules for structuring, storing and transferring information.
- Define a website with simple the HTML for a table. It is loaded into BeautifulSoup and parse it, returning a pandas data frame of the contents.

```
import pandas as pd
from bs4 import BeautifulSoup

html_string = '''
  <table>
    <tr>
      <td> Hello! </td>
      <td> Table </td>
    </tr>
  </table>
'''

soup = BeautifulSoup(html_string, 'lxml') # Parse the HTML as a string

table = soup.find_all('table')[0] # Grab the first table

new_table = pd.DataFrame(columns=range(0,2), index = [0]) # I know the size

row_marker = 0
for row in table.find_all('tr'):
  column_marker = 0
  columns = row.find_all('td')
  for column in columns:
    new_table.iat[row_marker,column_marker] = column.get_text()
    column_marker += 1

new_table
```

3.5.2 Using XPath for Data Extraction

- XPath specifies path expression that matches XML data by navigating down the tree. XPath is used as the embedded query language for both XQuery 1.0 and XSLT 2.0.
- XPath provides a common syntax and semantics for functionality shared between XSLT and XPointer. XPath is used in XSL transformations to find information in an

XML document. It is used to navigate through elements and attributes in XML documents.

- Parsers are represented by parser objects. There is support for parsing both XML and (broken) HTML.

```
from pandas.io.parsers import TextParser
def parse_options_data(table):
rows = table.findall('.//tr')
header = _unpack(rows[0], kind='th')
data = [_unpack(r) for r in rows[1:]]
return TextParser(data, names=header).get_chunk()
```

3.5.3 Dealing with Unicode

- Unicode is not an encoding. Unicode is more like a big database of characters and properties and rules, and on its own says nothing about how to encode a sequence of characters into a sequence of bytes for actual use by a computer.
- Unicode doesn't work in terms of characters; it works in terms of things called code points.
- Python 2 comes with two different kinds of objects that can be used to represent strings, str and unicode.
- Since Python 3.0, all strings are stored as Unicode in an instance of the str type. Encoded strings on the other hand are represented as binary data in the form of instances of the bytes type.

3.5.4 Stemming and Removing Stop Words

- Stemming is the process of reducing words to their root word.
- Tokenization : The process of segmenting text into words, clauses or sentences.
- Stemming : Reducing related words to a common stem.
- Removal of stop words : Removal of commonly used words unlikely to be useful for learning.
- The act of stemming and removing stop words simplifies the text and reduces the number of textual elements so that just the essential elements remain.
- Stemming is as follows :

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
#is based on The Porter Stemming Algorithm
stopword = stopwords.words('english')
```

```
snowball_stemmer = SnowballStemmer('english')
text = "This is a Demo Text for NLP using NLTK. Full form of NLTK is Natural Language Toolkit"
word_tokens = nltk.word_tokenize(text)
stemmed_word = [snowball_stemmer.stem(word) for word in word_tokens]
print (stemmed_word)
```

[OUTPUT]: ['this', 'is', 'a', 'demo', 'text', 'for', 'nlp', 'use', 'nltk', '.', 'full', 'form', 'of', 'nltk', 'is', 'natur', 'languag', 'toolkit']

- Let's look at an example.

```
from nltk.stem import PorterStemmer
stemming = PorterStemmer()

my_list = ['frightening', 'frightened', 'frightens']

# Using a Python list comprehension method to apply to all words in my_list

print ([stemming.stem(word) for word in my_list])
```

Out: ['frighten', 'frighten', 'frighten']

Removing stop words

- 'Stop words' are commonly used words that are unlikely to have any benefit in natural language processing. These includes words such as 'a', 'the', 'is'.

```
from nltk.corpus import stopwords
stops = set(stopwords.words("english"))

def remove_stops(row):
    my_list = row['stemmed_words']
    meaningful_words = [w for w in my_list if not w in stops]
    return (meaningful_words)

imdb['stem_meaningful'] = imdb.apply(remove_stops, axis=1)
```

3.6 Introducing Regular Expressions

- Regular expressions are specially encoded text strings used as patterns for matching sets of strings. The Python "re" module provides regular expression support.
- In Python a regular expression search is typically written as :

```
match = re.search(pat, str)
```


- The re.search() method takes a regular expression pattern and a string and searches for that pattern within the string. If the search is successful, search() returns a match object or None otherwise
- **Pattern-Matching Characters Used in Python**

Meta character	Description
.	Period matches any single character except a line break.
[ ]	Character class. Matches any character contained between the square brackets.
[^]	Negated character class. Matches any character that is not contained between the square brackets.
*	Matches 0 or more repetitions of the preceding of the preceding symbol.
+	Matches 1 or more repetitions of the preceding symbol.
?	Makes the preceding symbol optional.
{n,m}	Braces. Marches at least "n" but not more than "m" repetitions of the preceding symbol.
(xyz)	Character group. Matches the characters xyz in that exact order.
|	Alternation. Matches either the characters before or the characters after the symbol.
\	Escapes the next character. This allows you to match reserved characters [ ] ( ) { } . * + ? ^ $ \ |
^	Matches the beginning of the input.
$	Matches the end of the input.

- Python has a module named re to work with RegEx. Here's an example :

```
import re

pattern = '^a...s$'
test_string = 'abyss'
result = re.match(pattern, test_string)

if result:
  print("Search successful.")
else:
  print("Search unsuccessful.")
```

3.7 Using the Bag of Words Model and Beyond

- Bag of words simply refers to a matrix in which the rows are documents and the columns are words. The values matching a document with a word in the matrix, could be a count of word occurrences within the document or use tf-idf.
- Classifiers are used to train the bag of words and a special kind of algorithm used to break words down into categories.
- Traditionally, text documents are represented in NLP as a bag-of-words. This means that each document is represented as a fixed-length vector with length equal to the vocabulary size.
- Each dimension of this vector corresponds to the count or occurrence of a word in a document. Being able to reduce variable-length documents to fixed-length vectors makes them more amenable for use with a large variety of Machine Learning (ML) models and tasks.
- Fig. 3.7.1 shows turning raw text into a bag of words representation.

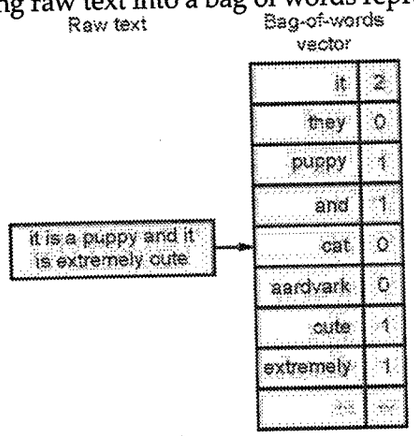


Fig. 3.7.1 Turning raw text into a bag of words representation

3.7.1 Working with n-grams

- The essential concepts in text mining is n-grams, which are a set of co-occurring or continuous sequence of n items from a sequence of large text or sentence. The item here could be words, letters, and syllables.
- 1-gram is also called as unigrams are the unique words present in the sentence. Bigram(2-gram) is the combination of 2 words. Trigram(3-gram) is 3 words and so on.
- An n-gram is a continuous sequence of items in the text you want to analyze. The items are phonemes, syllables, letters, words, or base pairs. The n in n-gram refers to a size.

- The code below generates n-grams in python :

```
import re

def generate_ngrams(text,n):

  # split sentences into tokens
  tokens=re.split("\\s+",text)
  ngrams=[]

  # collect the n-grams
  for i in range(len(tokens)-n+1):
    temp=[tokens[j] for j in range(i,i+n)]
    ngrams.append(" ".join(temp))

  return ngrams
```

3.7.2 Implementing TF-IDF Transformations

- TF-IDF stands for "Term Frequency, Inverse Data Frequency"
- Term Frequency (tf) gives us the frequency of the word in each document. Inverse Data Frequency (idf) is used to calculate the weight of rare words across all documents.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import numpy as np
import pandas as pd
import re
```

- Example:

```
from sklearn.feature_extraction.text import TfidfTransformer
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> from sklearn.pipeline import Pipeline
>>> import numpy as np
>>> corpus = ['this is the first document',
...           'this document is the second document',
...           'and this is the third one',
...           'is this the first document']
>>> vocabulary = ['this', 'document', 'first', 'is', 'second', 'the',
...               'and', 'one']
>>> pipe = Pipeline([('count', CountVectorizer(vocabulary=vocabulary)),
...                  ('tfid', TfidfTransformer())]).fit(corpus)
>>> pipe['count'].transform(corpus).toarray()
array([[1, 1, 1, 1, 0, 1, 0, 0],
       [1, 2, 0, 1, 1, 1, 0, 0],
```

```
    [1, 0, 0, 1, 0, 1, 1, 1],
    [1, 1, 1, 1, 0, 1, 0, 0]])
>>> pipe['tfid'].idf_
array([1.        , 1.22314355, 1.51082562, 1.        , 1.91629073,
       1.        , 1.91629073, 1.91629073])
>>> pipe.transform(corpus).shape
(4, 8)
```

3.7.3 Understanding the Adjacency Matrix

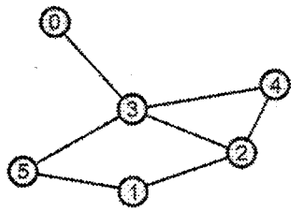
- An adjacency matrix represents the connections between nodes of a graph.
- The row and column indices represent the vertices: matrix[i][j] = 1matrix[i][j] = 1 means that there is an edge from vertices ii to jj, and matrix[i][j] = 0matrix[i][j] = 0 denotes that there is no edge between i and j.
- The advantage of the adjacency matrix is that it is simple, and for small graphs it is easy to see which nodes are connected to other nodes.
- Start Python and import NetworkX
- Different classes exist for directed and undirected networks. Let's create a basic undirected Graph:

```
g = nx.Graph() # empty graph
```

- The graph g can be grown in several ways. NetworkX provides many generator functions and facilities to read and write graphs in many formats.
- Example :

```
import networkx as nx

# Create a networkx graph object
my_graph = nx.Graph()

# Add edges to to the graph object
# Each tuple represents an edge between two nodes
my_graph.add_edges_from([ (1,2), (1,3), (3,4), (1,5), (3,5), (4,2), (2,3), (3,0)])

# Draw the resulting graph
nx.draw(my_graph, with_labels=True, font_weight='bold')
```

Output :



❑❑❑

Unit IV

Data Visualization

Syllabus

Visualizing Information :
Starting with a Graph, Defining the plot, Drawing multiple lines and plots, Saving your work to disk, Setting the Axis, Ticks, Grids, Getting the axes, Formatting the axes, Adding grids, Defining the Line Appearance, Working with line style, Using colors, Adding markers, Using Labels, Annotations, and Legends, Adding labels, Annotating the chart, Creating a legend.

Visualizing the Data :
Choosing the Right Graph, Showing parts of a whole with pie charts, Creating comparisons with bar charts, Showing distributions using histograms, Depicting groups using boxplots, Seeing data patterns using scatterplots, Creating Advanced Scatterplots, Depicting groups, Showing correlations, Plotting Time Series, Representing time on axes, Plotting trends over time, Plotting Geographical Data, Using an environment in Notebook, Getting the Basemap toolkit, Dealing with deprecated library issues, Using Basemap to plot geographic data, Visualizing Graphs, Developing undirected graphs, Developing directed graphs.

Contents

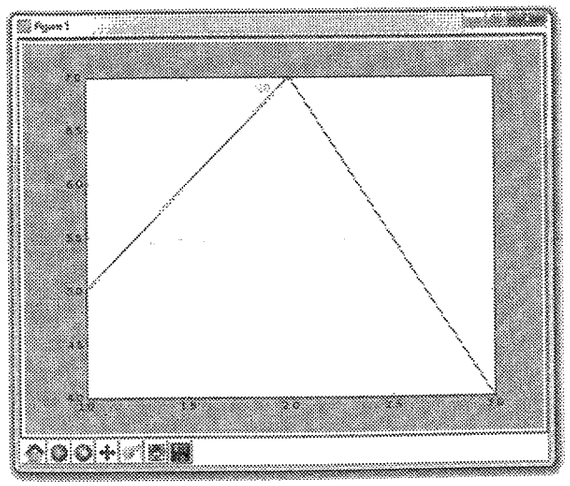
4.1 Visualizing Information : Starting with a Graph
4.2 Visualizing the Data
4.3 Showing Distributions using Histograms
4.4 Depicting Groups using Box Plots
4.5 Seeing Data Patterns using Scatterplots
4.6 Representing Time on Axes
4.7 Plotting Geographical Data
4.8 Visualizing Graphs

4.1 Visualizing Information : Starting with a Graph

- Data visualization is the presentation of quantitative information in a graphical form. In other words, data visualizations turn large and small datasets into visuals that are easier for the human brain to understand and process.
- Good data visualizations are created when communication, data science, and design collide. Data visualizations done right offer key insights into complicated datasets in ways that are meaningful and intuitive.
- A graph is simply a visual representation of numeric data. MatPlotLib supports a large number of graph and chart types.
- Matplotlib is a popular Python package used to build plots. Matplotlib can also be used to make 3D plots and animations.
- Line plots can be created in Python with Matplotlib's pyplot library. To build a line plot, first import Matplotlib. It is a standard convention to import Matplotlib's pyplot library as plt.
- To define a plot, you need some values, the matplotlib.pyplot module, and an idea of what you want to display.

```
import matplotlib.pyplot as plt
plt.plot([1,2,3],[5,7,4])
plt.show()
```

- The plt.plot will "draw" this plot in the background, but we need to bring it to the screen when we're ready, after graphing everything we intend to.
- plt.show() : With that, the graph should pop up. If not, sometimes it can pop under, or you may have gotten an error. Your graph should look like:



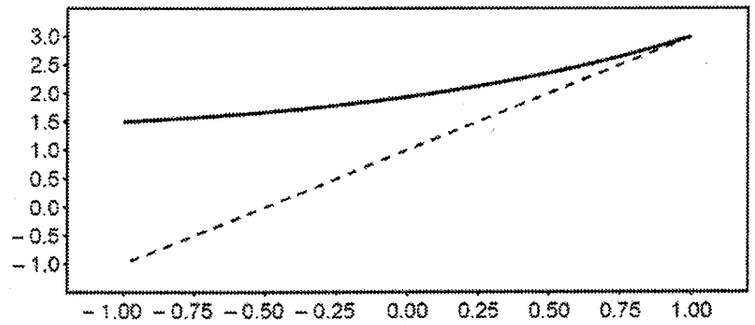
- This window is a matplotlib window, which allows us to see our graph, as well as interact with it and navigate it

4.1.1 Drawing Multiple Lines and Plots

- More than one line can be in the plot. To add another line, just call the plot(x,y) function again. In the example below we have two different values for y (y1,y2) that are plotted onto the chart.

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-1, 1, 50)
y1 = 2*x + 1
y2 = 2**x + 1

plt.figure(num = 3, figsize=(8, 5))
plt.plot(x, y2)
plt.plot(x, y1,
      linewidth=1.0,
      linestyle='--'
    )
plt.show()
```

- Output of the above code will look like this:



4.1.2 Saving your Work To Disk

- Matplotlib plots can be saved as image files using the plt.savefig() function.
- The .savefig() method requires a filename be specified as the first argument. This filename can be a full path. It can also include a particular file extension if desired. If no extension is provided, the configuration value of savefig.format is used instead.
- The .savefig() also has a number of useful optional arguments :

1. dpi can be used to set the resolution of the file to a numeric value.

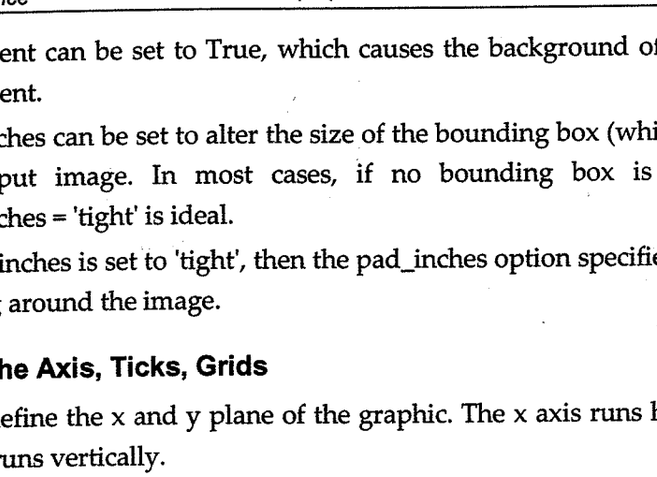
2. transparent can be set to True, which causes the background of the chart to be transparent.
3. bbox_inches can be set to alter the size of the bounding box (whitespace) around the output image. In most cases, if no bounding box is desired, using bbox_inches = 'tight' is ideal.
4. If bbox_inches is set to 'tight', then the pad_inches option specifies the amount of padding around the image.

4.1.3 Setting the Axis, Ticks, Grids

- The axes define the x and y plane of the graphic. The x axis runs horizontally, and the y axis runs vertically.
- An axis is added to a plot layer. Axis can be thought of as sets of x and y axis that lines and bars are drawn on. An Axis contains daughter attributes like axis labels, tick labels, and line thickness.
- The following code shows how to obtain access to the axes for a plot:

```
fig = plt.figure()
axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])     # left, bottom, width, height (range 0 to 1)
axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```

Output:



- A grid can be added to a Matplotlib plot using the plt.grid() command. By defaut, the grid is turned off. To turn on the grid use:

```
plt.grid(True)
```


- The only valid options are plt.grid(True) and plt.grid(False). Note that True and False are capitalized and are not enclosed in quotes.

4.1.4 Defining the Line Appearance & Working with Line Style

- Line styles help differentiate graphs by drawing the lines in various ways. Following line style is used by Matplotlib.

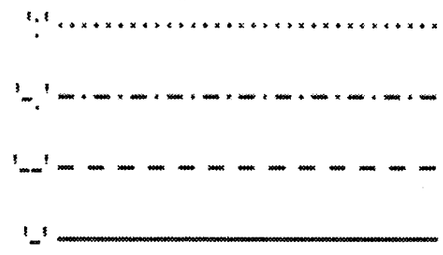


Fig. 4.1.1 Line style

- Matplotlib has an additional parameter to control the colour and style of the plot.

```
plt.plot(xa, ya 'g')
```

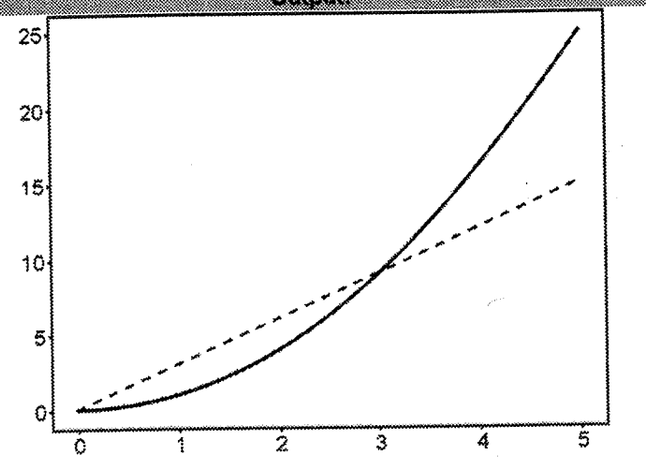
- This will make the line green. You can use any colour of red, green, blue, cyan, magenta, yellow, white or black just by using the first character of the colour name in lower case (use "k" for black, as "b" means blue).
- You can also alter the linestyle, for example two dashes -- makes a dashed line. This can be used added to the colour selector, like this:

```
plt.plot(xa, ya 'r--')
```

- You can use "-" for a solid line (the default), "-." for dash-dot lines, or ":" for a dotted line. Here is an example:

```
from matplotlib import pyplot as plt
import numpy as np

xa = np.linspace(0, 5, 20)

ya = xa**2
plt.plot(xa, ya, 'g')

ya = 3*xa
plt.plot(xa, ya, 'r--')

plt.show()
```

Output:



- MatPlotLib Colors are as follows :

Color	Character
Black	'k'
Yellow	'y'
Cyan	'c'
Blue	'b'
Green	'g'
Red	'r'
White	'w'
Magenta	'm'

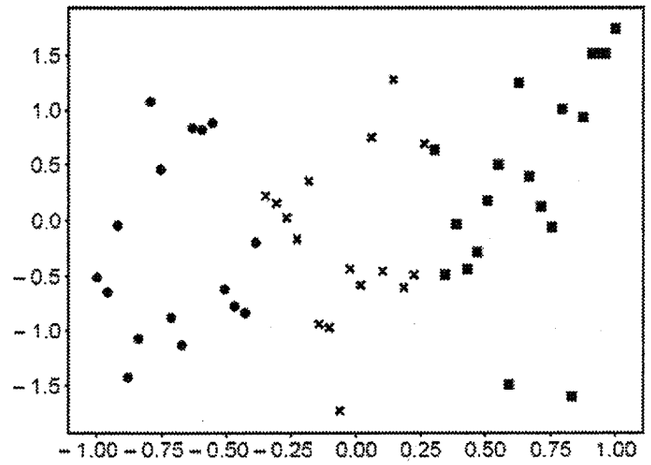
4.1.5 Adding Markers

- Markers add a special symbol to each data point in a line graph. Unlike line style and color, markers tend to be a little less susceptible to accessibility and printing issues.
- Basically, the matplotlib tries to have identifiers for the markers which look similar to the marker:
  1. Triangle-shaped: v, <, >, ^
  2. Cross-like: *, +, 1, 2, 3, 4
  3. Circle-like: o, ., h, p, H, 8

- Having differently shaped markers is a great way to distinguish between different groups of data points. If your control group is all circles and your experimental group is all X's the difference pops out, even to colorblind viewers.

```
N = x.size          // 3
ax.scatter(x[:N], y[:N], marker="o")
ax.scatter(x[N: 2 * N], y[N: 2 * N], marker="x")
ax.scatter(x[2 * N:], y[2 * N:], marker="s")
```

- There's no way to specify multiple marker styles in a single scatter() call, but we can separate our data out into groups and plot each marker style separately. Here we chopped our data up into three equal groups.



4.1.6 Using Labels, Annotations, and Legends

- To fully document your graph, you usually have to resort to labels, annotations, and legends. Each of these elements has a different purpose, as follows:
  1. Label : Make it easy for the viewer to know the name or kind of data illustrated
  2. Annotation : Help extend the viewer's knowledge of the data, rather than simply identify it.
  3. Legend : Provides cues to make identification of the data group easier.
- The following example shows how to add labels to your graph :

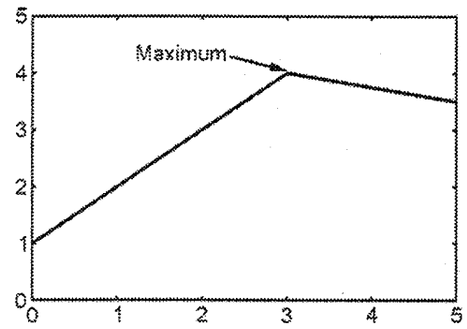
```
values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
import matplotlib.pyplot as plt
plt.xlabel('Entries')
plt.ylabel('Values')
```

```
plt.plot(range(1,11), values)
plt.show()
```

- following example shows how to add annotation to a graph:

```
import matplotlib.pyplot as plt
w = 4
h = 3
d = 70
plt.figure(figsize=(w, h), dpi=d)
plt.axis([0, 5, 0, 5])
x = [0, 3, 5]
y = [1, 4, 3.5]
label_x = 1
label_y = 4
arrow_x = 3
arrow_y = 4
arrow_properties = dict(
  facecolor="black", width=0.5,
  headwidth=4, shrink=0.1)
plt.annotate( "maximum", xy=(arrow_x, arrow_y),
              xytext=(label_x, label_y),
             arrowprops=arrow_properties)
plt.plot(x, y)
plt.savefig("out.png")
```

Output:



Creating a legend

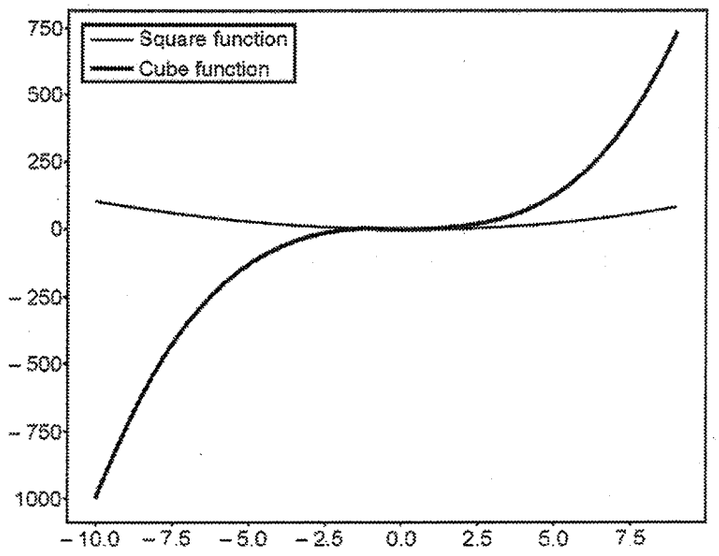
- There are several options available for customizing the appearance and behavior of the plot legend. By default the legend always appears when there are multiple series and only appears on mouseover when there is a single series. By default the

legend shows point values when the mouse is over the graph but not when the mouse leaves.

- A legend documents the individual elements of a plot. Each line is presented in a table that contains a label for it so that people can differentiate between each line.

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-10, 9, 20)
y = x ** 3
z = x ** 2

figure = plt.figure()
axes = figure.add_axes([0,0,1,1])

axes.plot(x, z, label="Square Function")
axes.plot(x, y, label="Cube Function")
axes.legend()
```

- In the script above we define two functions : square and cube using x, y and z variables. Next, we first plot the square function and for the label parameter, we pass the value Square Function.
- This will be the value displayed in the label for square function. Next, we plot the cube function and pass Cube Function as value for the label parameter.
- The output looks likes this:



4.2 Visualizing the Data

- A type of graph is which a circle is divided into sectors that each represent a proportion of whole. Each sector shows the relative size of each value.
- A pie chart displays data, information and statistics in an easy to read "pie slice" format with varying slice sizes telling how much of one data element exists.
- Fig. 4.2.1 shows pie chart.

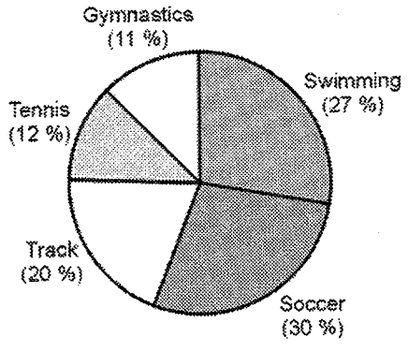


Fig. 4.2.1 Pie chart

- Pie chart is also known as circle graph. The bigger the slice, the more of that particular data was gathered. The main use of a pie chart is to show comparisons.
- Pie charts can be drawn using the function pie() in the pyplot module. The below python code example draws a pie chart using the pie() function.
- By default the pie() fucntion of pyplot arranges the pies or wedges in a pie chart in counter clockwise direction.
- Example:

```
# import the pyplot library
import matplotlib.pyplot as plotter

 # The slice names of a student distribution pie chart
pieLabels  = 'Rakshita', 'Ritesh', 'Rupali', 'Rutu', 'Rushi', 'Radhika'

 # marks data
marksShare    = [59.69, 16, 9.94, 7.79, 5.68, 0.54]
 figureObject, axesObject = plotter.subplots()

 # Draw the pie chart
axesObject.pie(marksShare,  labels=pieLabels, autopct='%1.2f', startangle=90)

# Aspect ratio - equal means pie is a circle
axesObject.axis('equal')
```



```
plotter.show()
```

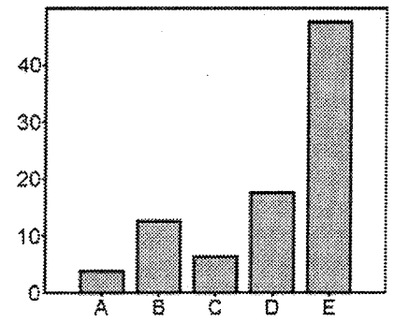
- The essential part of a pie chart is the values. You could create a basic pie chart using just the values as input.

4.2.1 Creating Comparisons with Bar Charts

- A bar chart is a way of summarizing a set of categorical data. The bar chart displays data using a number of bars, each representing a particular category. The height of each bar is proportional to a specific aggregation.
- The categories could be something like an age group or a geographical location. It is also possible to color or split each bar into another categorical column in the data, which enables you to see the contribution from different categories to each bar or group of bars in the bar chart.
- Bar charts make comparing values easy.
- **Example:**

```
import numpy as np
import matplotlib.pyplot as plt

# Make a fake dataset:
height = [3, 12, 5, 18, 45]
bars = ('A', 'B', 'C', 'D', 'E')
y_pos = np.arange(len(bars))

# Create bars
plt.bar(y_pos, height)

# Create names on the x-axis
plt.xticks(y_pos, bars)

# Show graphic
plt.show()
```

Output:



4.3 Showing Distributions using Histograms

- In a histogram, the data are grouped into ranges (e.g. 10–19, 20–29) and then plotted as connected bars. Each bar represents a range of data. The width of each bar is proportional to the width of each category, and the height is proportional to the frequency or percentage of that category.
- It provides a visual interpretation of numerical data by showing the number of data points that fall within a specified range of values called "bins".
- Fig. 4.3.1 shows histogram.

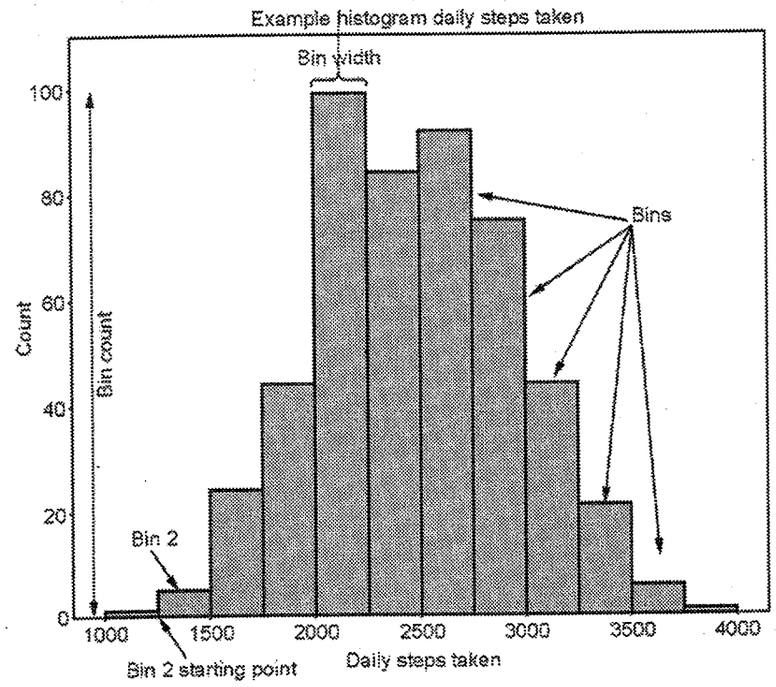


Fig. 4.3.1 Histogram

- Histograms can display a large amount of data and the frequency of the data values. The median and distribution of the data can be determined by a histogram. In addition, it can show any outliers or gaps in the data.
- Matplotlib provides a dedicated function to compute and display histograms: plt.hist()
- Code for creating histogram with randomized data:

```
import numpy as np
import matplotlib.pyplot as plt
x = 40 * np.random.randn(50000)
```



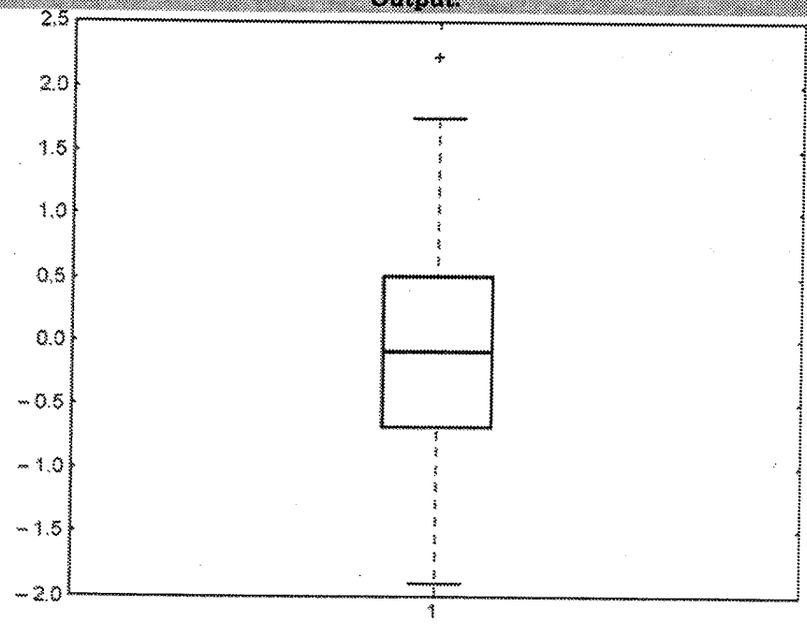
```
plt.hist(x, 20, range=(-50, 50), histtype='stepfilled',
align='mid', color='r', label='Test Data')
plt.legend()
plt.title(' Histogram')
plt.show()
```

4.4 Depicting Groups using Box Plots

- Box plot uses boxes and lines to depict the distributions of one or more groups of numeric data. Box plots are used to show distributions of numeric data values, especially when you want to compare them between multiple groups.
- A box plot may also have lines, called whiskers, indicating data outside the upper and lower quartiles.
- In most cases, it is possible to use numpy or Python objects, but pandas objects are preferable because the associated names will be used to annotate the axes. Additionally, you can use Categorical types for the grouping variables to control the order of plot elements.
- The following example shows how to create a box plot with randomized data:

```
import numpy as np
import matplotlib.pyplot as plt
data = np.random.randn(100)
plt.boxplot(data)
plt.show()
```

Output:



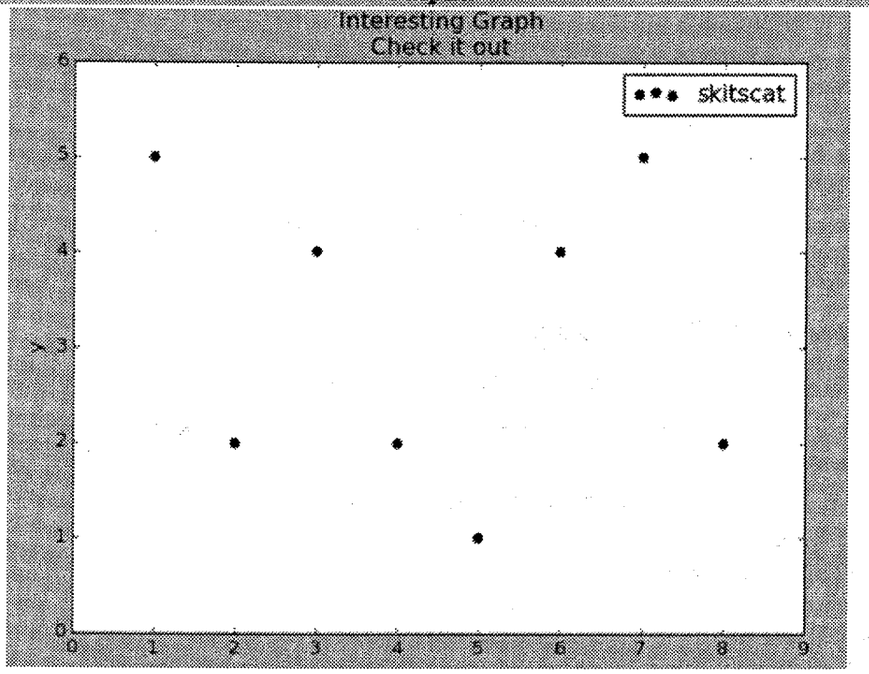
- The call to boxplot()requires only data as input. All other parameters have default settings.

4.5 Seeing Data Patterns using Scatterplots

- It displays collection of all the points for the set of data limited only for two values. Also called scatter plot, X-Y graph.
- While working with statistical data it is often observed that there are connections between sets of data. For example the mass and height of persons are related, the taller the person the greater his/her mass.
- To find out whether or not two sets of data are connected scatter diagrams can be used.
- A scatter diagram is a tool for analyzing relationship between two variables. One variable is plotted on the horizontal axis and the other is plotted on the vertical axis.
- The pattern of their intersecting points can graphically show relationship patterns. Commonly a scatter diagram is used to prove or disprove cause-and-effect relationships.
- While scatter diagram shows relationships, it does not by itself prove that one variable causes other. In addition to showing possible cause and effect relationships, a scatter diagram can show that two variables are from a common cause that is unknown or that one variable can be used as a surrogate for the other.
- The following example shows how to create a scatterplot :

```
import matplotlib.pyplot as plt
x = [1,2,3,4,5,6,7,8]
y = [5,2,4,2,1,4,5,2]
plt.scatter(x,y, label='skitscat', color='k', s=25, marker="o")
plt.xlabel('x')
plt.ylabel('y')
plt.title('Interesting Graph\nCheck it out')
plt.legend()
plt.show()
```


Output:



- The plt.scatter allows us to not only plot on x and y, but it also lets us decide on the color, size, and type of marker we use.
- The primary difference of plt.scatter from plt.plot is that it can be used to create scatter plots where the properties of each individual point (size, face color, edge color, etc.) can be individually controlled or mapped to data.

4.5.1 Creating Advanced Scatterplots

- Scatterplots are especially important for data science because they can show data patterns that aren't obvious when viewed in other ways.

```
import matplotlib.pyplot as plt

x_axis1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y_axis1 = [5, 16, 34, 56, 32, 56, 32, 12, 76, 89]

x_axis2 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y_axis2 = [53, 6, 46, 36, 15, 64, 73, 25, 82, 9]

plt.title("Prices over 10 years")
plt.scatter(x_axis1, y_axis1, color='darkblue', marker='x', label="item 1")
plt.scatter(x_axis2, y_axis2, color='darkred', marker='x', label="item 2")
```

```
plt.xlabel("Time (years)")
plt.ylabel("Price (dollars)")

plt.grid(True)
plt.legend()

plt.show()
```

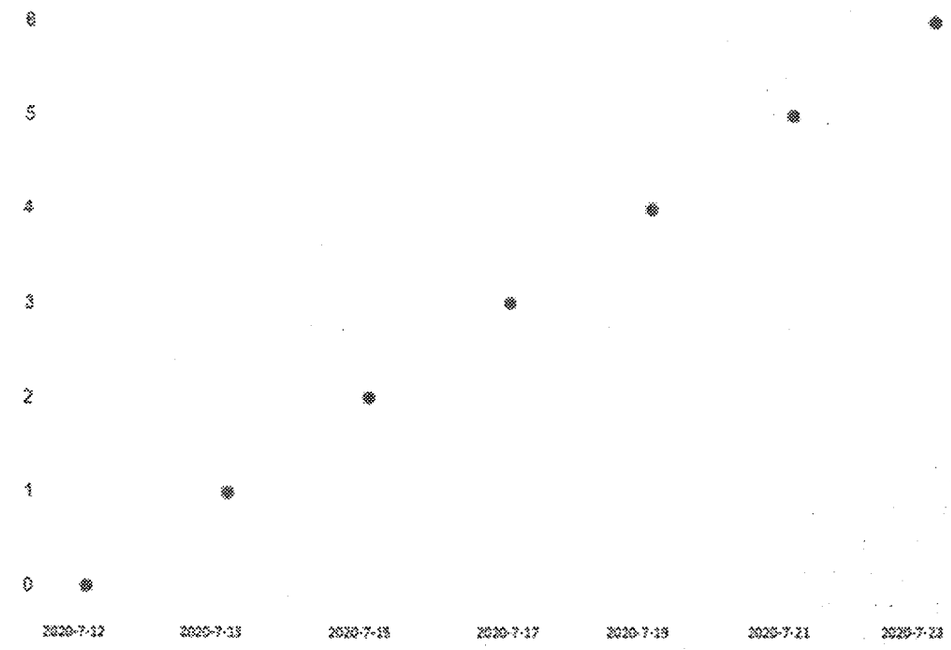
- The chart displays two data sets. We distinguish between them by the colour of the marker.

4.6 Representing Time on Axes

- First of all, we will create a scatter plot of dates and values in Matplotlib using plt.plot_date(). We will be using Python's built-in module called datetime(datetime, timedelta) for parsing the dates. So, let us create a python file called 'plot_time_series.py' and make necessary imports.

```
#  plot_time_series.py
   import matplotlib.pyplot as plt
   from datetime import datetime, timedelta
plt.style.use('seaborn')
dates = [
  datetime(2020, 7, 12),
  datetime(2020, 7, 13),
  datetime(2020, 7, 15),
  datetime(2019, 7, 17),
  datetime(2020, 7, 19),
  datetime(2020, 7, 21),
  datetime(2020, 7, 23),
]
y = [0, 1, 2, 3, 4, 5, 6]
plt.plot_date(dates, y)
plt.tight_layout()
plt.show()
```


Output:



4.7 Plotting Geographical Data

- The Basemap module includes data for drawing coastlines and country boundaries onto world maps.
- To set the extent is by defining the map bounding box in geographical coordinates :
- Table 4.7.1 : Setting the bounding box

Argument	Description
llcrnrx	The lower left corner x coordinate, in the projection units
llcrnry	The lower left corner y coordinate, in the projection units
urcrnrx	The upper right corner x coordinate, in the projection units
urcrnry	The upper right corner y coordinate, in the projection units

- Table 4.7.2 : Set the bounding box giving the center point in geographical coordinates

Argument	Description
width	The width of the map in the projection units
height	The height of the map in the projection units
lon_0	The longitude of the center of the map
lat_0	The latitude of the center of the map

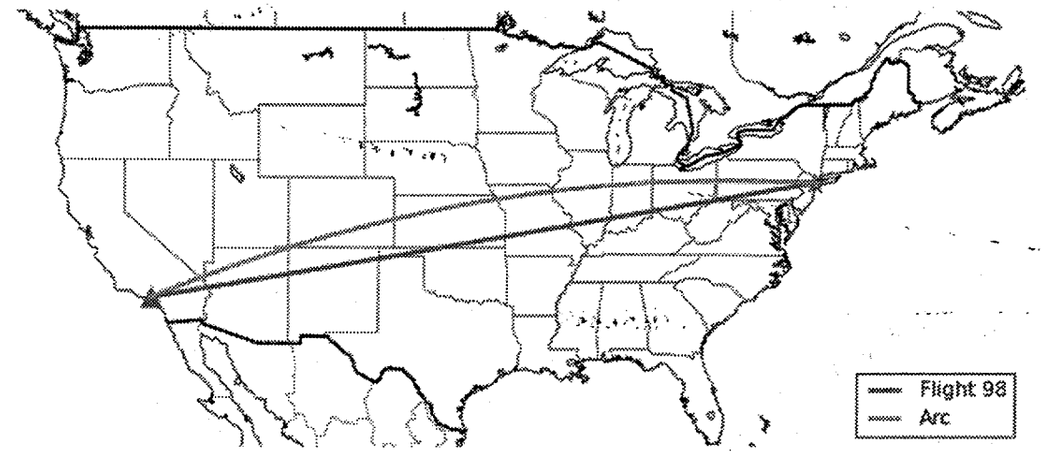
Example:

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
m = Basemap(projection='mill',
        llcrnrlat = 25,
        llcrnrlon = -130,
        urcrnrlat = 50,
        urcrnrlon = -60,
        resolution='l')
m.drawcoastlines()
m.drawcountries(linewidth=2)
m.drawstates(color='b')
#m.drawcounties(color='darkred')
#m.fillcontinents()
#m.etopo()
#m.bluemarble()
xs = []
ys = []
NYClat, NYClon = 40.7127, -74.0059
xpt, ypt = m(NYClon, NYClat)
xs.append(xpt)
ys.append(ypt)
m.plot(xpt, ypt, 'c*', markersize=15)
LAlat, LAlon = 34.05, -118.25
xpt, ypt = m(LAlon, LAlat)
xs.append(xpt)
ys.append(ypt)
m.plot(xpt, ypt, 'g^', markersize=15)
m.plot(xs, ys, color='r', linewidth=3, label='Flight 98')
```



```
m.drawgreatcircle(NYClon, NYClat, LAlon, LAlat, color='c', linewidth=3, label='Arc')
plt.legend(loc=4)
plt.title('Basemap Tutorial')
plt.show()
```

Output:



4.8 Visualizing Graphs

- The visualization of graphs has proven to be very useful for exploring structures in different application domains. A graph is a depiction of data showing the connections between data points using lines.
- Graph is just a set of objects which are connected in some way. The objects are called vertices or nodes. Pictorially, we usually draw the vertices as circles, and draw a line between two vertices if they are connected or related. These lines are called edges or links.
- Graphs naturally arise in many ways, as they are a convenient way to visualize various situations or complex symptoms. Computer systems in a local network form a graph. So do the landlines telephone cable systems and internet routing systems.
- The World Wide Web : One can form a graph of all web pages, and make an edge from Page A to Page B if there is a hyperlink from Page A to Page B. In this case, one should consider directed edges, meaning each edge has a direction which is pictorially indicated with an arrow.
- As used in graph theory, the term *graph* does not refer to data charts such as line graphs or bar graphs. Instead, it refers to a set of vertices (that is, points or nodes) and of edges (or lines) that connect the vertices.

- **Graph** - a visual representation of edges and vertices
- **Edge** - the line between two boundaries
- **Vertex** - a point where two or more lines meet
- Fig. 4.8.1 shows vertex and edge in graph.

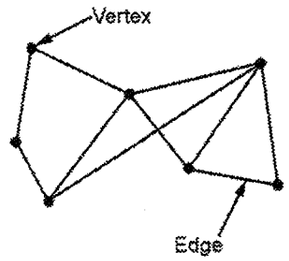


Fig. 4.8.1 : Vertex and edge in graph

- Two vertices that are contained in an edge are adjacent; two edges that share a vertex are adjacent; an edge and a vertex contained in that edge are incident.
- **Multigraph :** Multiple edges are allowed between vertices
- **Simple graph :** A graph without loops and with at most one edge between any two vertices
- **Pseudograph** : A graph that may contain multiple edges and graph loops
- An undirected graph is connected if every two nodes in the network are connected by some path in the network. Components of a graph (or network) are the distinct maximally connected sub-graphs.
- A directed graph is connected if the underlying undirected graph is connected. It is strongly connected if each node can reach every other node by a directed path.
- Graphs can also be valued or non-valued. A valued graph has numbers attached to the lines that indicate the strength or frequency or intensity or quantity of the tie between nodes. For example, Fig. 4.8.2 might record the amount of trade, in trillions of dollars, between some countries:

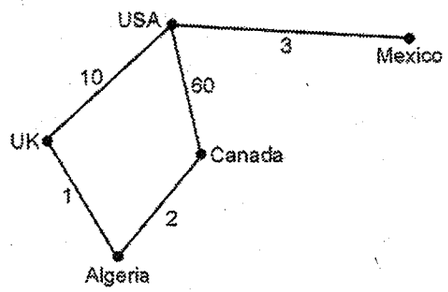


Fig. 4.8.2 : Valued graph

- If a line connects two points, they are said to be "adjacent". The two points connected by a line are called endpoints. An edge that originates or terminates at a given point is "incident" upon that point. Two edges that share a point are also said to be incident.
- A subgraph of a graph is a subset of its points together with all the lines connecting members of the subset. The subgraph of Fig. 4.8.2 that includes the UK, Canada and Algeria has two lines: (UK, Algeria) and (Algeria, Canada).
- The degree of a point is defined as the number of lines incident upon that node. In Fig. 4.8.2, the degree of USA is 3 because it has 3 ties. If a point has degree 0 it is called an isolate. If it has degree 1 it is called a pendant.
- In a directed graph, a point has both indegree and outdegree. The outdegree is the number of arcs from that point to other points.
- A node is reachable from another node if there exists a path of any length from one to the other.
- A connected component is a maximal subgraph in which all nodes are reachable from every other. Maximal means that it is the largest possible subgraph: you could not find another node anywhere in the graph such that it could be added to the subgraph and all the nodes in the subgraph would still be connected.
- For directed graphs, there strong components and weak components. A strong component is a maximal subgraph in which there is a path from every point to every point following all the arcs in the direction they are pointing. A weak component is a maximal subgraph which would be connected if we ignored the direction of the arcs.
- The following little Python script uses NetworkX to create an empty graph :

```
import networkx as nx
G=nx.Graph()
print(G.nodes())
print(G.edges())
print(type(G.nodes()))
print(type(G.edges()))
```

❑❑❑

Notes

Unit V

Data Wrangling

Syllabus

Wrangling Data :

Playing with Scikit-learn, Understanding classes in Scikit-learn, Defining applications for data science, Performing the Hashing Trick, Using hash functions, Demonstrating the hashing trick, Working with deterministic selection, Considering Timing and Performance, Benchmarkin, with,timeit, Working with the memory profiler, Running in Parallel on Multiple Cores, Performing multicore parallelism, Demonstrating multiprocessing.

Exploring Data Analysis :

The EDA Approach, Defining Descriptive Statistics for Numeric Data, Measuring central tendency,Measuring variance and range ,Working with percentiles, Defining measures of normality, Counting for Categorical Data, Understanding frequencies, Creating contingency tables, Creating Applied Visualization for EDA ,Inspecting boxplots, Performing t-tests after boxplots, Observing parallel coordinates, Graphing distributions, Plotting scatterplots , Understanding Correlation, Using covariance and correlation, Using nonparametric correlation, Considering the chi-square test for tables, Modifying Data Distributions, Using different statistical distributions, Creating a Z-score standardization, Transforming other notable distributions.

Contents

5.1 Introduction to Data Wrangling

5.2 Playing with Scikit-Learn

5.3 Performing the Hashing Trick

5.4 Considering Timing and Performance : Benchmarking with timeit

5.5 Running in Parallel on Multiple Cores

5.6 Exploring Data Analysis

5.7 Creating Applied Visualization for EDA

5.8 Understanding Correlation

5.9 Considering the Chi-Square Test for Tables

5.1 Introduction to Data Wrangling

- Data Wrangling is the process of transforming data from its original "raw" form into a more digestible format and organizing sets from various sources into a singular coherent whole for further processing.
- Data wrangling is also called as data munging.
- The primary purpose of data wrangling can be described as getting data in coherent shape. In other words, it is making raw data usable. It provides substance for further proceedings.
- Data wrangling covers the following processes :
  1. Getting data from the various source into one place
  2. Piecing the data together according to the determined setting
  3. Cleaning the data from the noise or erroneous, missing elements
- Data wrangling is the process of cleaning, structuring and enriching raw data into a desired format for better decision making in less time
- There are typically six iterative steps that make up the data wrangling process:
  1. Discovering : Before you can dive deeply, you must better understand what is in your data, which will inform how you want to analyze it. How you wrangle customer data, for example, may be informed by where they are located, what they bought, or what promotions they received.
  2. Structuring : This means organizing the data, which is necessary because raw data comes in many different shapes and sizes. A single column may turn into several rows for easier analysis. One column may become two. Movement of data is made for easier computation and analysis.
  3. Cleaning : What happens when errors and outliers skew your data ? You clean the data. What happens when state data is entered as AP or Andhra Pradesh or Arunachal Pradesh? You clean the data. Null values are changed and standard formatting implemented, ultimately increasing data quality.
  4. Enriching : Here you take stock in your data and strategize about how other additional data might augment it. Questions asked during this data wrangling step might be : what new types of data can I derive from what I already have or what other information would better inform my decision making about this current data?
  5. Validating : Validation rules are repetitive programming sequences that verify data consistency, quality, and security. Examples of validation include

ensuring uniform distribution of attributes that should be distributed normally (e.g. birth dates) or confirming accuracy of fields through a check across data.

6. Publishing : Analysts prepare the wrangled data for use downstream, whether by a particular user or software and document any particular steps taken or logic used to wrangle said data. Data wrangling gurus understand that implementation of insights relies upon the ease with which it can be accessed and utilized by others.

5.2 Playing with Scikit-Learn

- Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy .
- It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.
- Scikit-learn is a library, i.e. a collection of classes and functions that users import into Python programs. Using scikit-learn therefore requires basic Python programming knowledge
- The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes :
  1. NumPy : Base n-dimensional array package
  2. SciPy : Fundamental library for scientific computing
  3. Matplotlib : Comprehensive 2D/3D plotting
  4. IPython : Enhanced interactive console
  5. Sympy : Symbolic mathematics
  6. Pandas : Data structures and analysis
- Scikit-learn is the package for machine learning and data science experimentation favoured by most data scientists. It contains a wide range of well-established learning algorithms, error functions, and testing procedures.

5.2.1 Understanding Classes in Scikit-learn

- Scikit-learn takes a highly object-oriented approach to machine learning models. Every major Scikit-learn class inherits from ***sklearn.base.BaseEstimator.***
- Scikit-learn features some base classes on which all the algorithms are built. Apart from *BaseEstimator*, the class from which all other classes inherit, there are four class types covering all the basic machine learning functionalities:

1. Classifying
2. Regressing
3. Grouping by clusters
4. Transforming data

- Scikit-learn takes a highly object-oriented approach to machine learning models. Every major Scikit-learn class inherits from ***sklearn.base.BaseEstimator***.
- All objects within scikit-learn share a uniform common basic API consisting of three complementary interfaces: an **estimator** interface for building and fitting models, a **predictor** interface for making predictions and a **transformer** interface for converting data.

1. Estimators

- The estimator interface is at the core of the library. It defines instantiation mechanisms of objects and exposes a fit method for learning a model from training data.
- All supervised and unsupervised learning algorithms (e.g., for classification, regression or clustering) are offered as objects implementing this interface. Machine learning tasks like feature extraction, feature selection or dimensionality reduction are also provided as estimators.

2. Predictors

- The predictor interface extends the notion of an estimator by adding a predict method that takes an array X test and produces predictions for X test, based on the learned parameters of the estimator.
- In the case of supervised learning estimators, this method typically returns the predicted labels or values computed by the model.

3. Transformers

- Since it is common to modify or filter data before feeding it to a learning algorithm, some estimators in the library implement a transformer interface which defines a transform method.
- It takes as input some new data X test and yields as output a transformed version of X test.
- Preprocessing, feature selection, feature extraction and dimensionality reduction algorithms are all provided as transformers within the library.

5.2.2 Defining Applications for Data Science

- Data science applications are based on classification and regression problem. In classification problem, it is used for guessing a new observation is from a certain group and guessing the value of a new observation in regression problem.
- The Surat Housing Dataset consists of price of houses in various places in Surat city. Alongside with price, the dataset also provide information such as Crime (CRIM), areas of non-retail business in the town (INDUS), the age of people who own the house (AGE), and there are many other attributes that available here.
- The dataset itself is available here. However, because we are going to use scikit-learn, we can import it right away from the scikit-learn itself. In this story, we will use several python libraries as required here.
- First of all, just like what we do with any other dataset, we are going to import the Surat Housing dataset and store it in a variable called surat. To import it from scikit-learn we will need to run this snippet.

```
from sklearn.datasets import load_surat
boston = load_surat( )
```

- The Surat dataset, for instance, contains predictor variables that the example code can match against house prices, which helps build a predictor that can figure out the value of a house given some characteristics of it.

```
from sklearn.datasets import load_surat
boston = load_surat()
X, y = surat.data, surat.target
print X.shape, y.shape
```

- A hypothesis is a way to describe a learning algorithm trained with data. The hypothesis defines a possible representation of y given X that you test for validity. Therefore, it's a hypothesis in both scientific and machine-learning language.

5.3 Performing the Hashing Trick

- The class FeatureHasher is a high-speed, low-memory vectorizer that uses a technique known as feature hashing, or the "hashing trick".
- Instead of building a hash table of the features encountered in training, as the vectorizers do, instances of FeatureHasher apply a hash function to the features to determine their column index in sample matrices directly.
- The result is increased speed and reduced memory usage, at the expense of inspectability; the hasher does not remember what the input features looked like and has no inverse_transform method.

- Advantages:

1. Handling large data matrices based on text on the fly
2. Fixing unexpected values or variables in your textual data
3. Building scalable algorithms for large collections of documents

5.3.1 Using hash Functions

- Hash functions can transform any input into an output whose characteristics are predictable. Usually they return a value where the output is bound at a specific interval.
- In general, a good hash function is one where a small change in the input text will cause a large change in the output.
- In a certain sense, hash functions are like a secret code, transforming everything into numbers. Unlike secret codes, however, you can't convert the hashed code to its original value. In addition, in some rare cases, different words generate the same hashed result

5.3.2 Performing the Hashing Trick

- There is a solution called the hashing trick because it is based on the hash function and can deal with both text and categorical variables in integer or string form.
- It can also work with categorical variables mixed with numeric values from quantitative features. The core problem with one-hot encoding is that it assigns a position to a value in the feature vector after having mapped its feature to that position.
- The hashing trick can univocally map a value to its position without any prior need to evaluate the feature because it leverages the core characteristic of a hashing function, to transform a value or string into an integer value deterministically.
- A real problem with the hashing trick is the eventuality of a collision, which happens when two different tokens are associated to the same index.
- This is a rare but possible occurrence when working with large dictionaries of words. On the other hand, in a model composed of millions of coefficients, there are very few that are influential.
- Consequently, if a collision happens, probably it will involve two unimportant tokens. When using the hashing trick, probability is on your side because with a large enough output vector, though collisions are always possible, it will be highly unlikely that they will involve important elements of the model

5.3.3 Working with Deterministic Selection

- The worst-case performance of a randomized selection algorithm is $O(n^2)$. It is possible to improve on a section of the randomized selection algorithm to obtain a worst-case performance of O(n). This kind of algorithm is called deterministic selection.
- Sparse matrices store just the coordinates of the cells and their values, instead of storing the information for all the cells in the matrix.
- When an application requests data from an empty cell, the sparse matrix will return a zero value after looking for the coordinates and not finding them.
- For example : consider the following vector

```
[1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,0]
```

- The following code turns it into a sparse matrix:

```
from scipy.sparse import csc_matrix
print csc_matrix([1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0])
(0, 0)      1
(0, 5)      1
(0, 16)     1
(0, 18)     1
```

- the data representation is in coordinates and the cell value.
- By using two specialized functions from the Scikit learn package: ***HashingVectorizer***, a transformer based on the hashing trick that works on textual data, and ***FeatureHasher***, which is another transformer, specialized in converting a data row expressed as a Python dictionary to a sparse vector of features

```
In: from sklearn.feature_extraction.text import HashingVectorizer
h = HashingVectorizer(n_features=1000, binary=True, norm=None)
sparse_vector = h.transform(['A simple toy example will make clearhow it works.'])
print(sparse_vector)
```

Output:

```
(0, 61)     1.0
(0, 271)    1.0
(0, 287)    1.0
(0, 452)    1.0
(0, 462)    1.0
(0, 539)    1.0
(0, 605)    1.0
(0, 726)    1.0
(0, 918)    1.0
```

- The resulting vector has unit values only at certain indexes, pointing out an association between a token in the phrase (a word) and a certain position in the vector.

5.4 Considering Timing and Performance : Benchmarking with timeit

- Python comes with a module called timeit. You can use it to time small code snippets. The timeit module uses platform-specific time functions so that you will get the most accurate timings possible.
- Syntax: **timeit.timeit(stmt, setup,timer, number)**

Parameters

1. stmt : This will take the code for which you want to measure the execution time. The default value is "pass".
2. setup : This will have setup details that need to be executed before stmt. The default value is "pass."
3. timer : This will have the timer value, timeit() already has a default value set, and we can ignore it.
4. number : The stmt will execute as per the number is given here. The default value is 1000000.

- timeit defines a single public class, Timer. The constructor for Timer takes a statement to be timed, and a setup statement (to initialize variables, for example). The Python statements should be strings and can include embedded newlines.
- The timeit( ) method runs the setup statement one time, then executes the primary statement repeatedly and returns the amount of time which passes. The argument to timeit( ) controls how many times to run the statement; the default is 1,000,000.

5.4.1 Working with the Memory Profiler

- The memory_profiler module can be used for monitoring memory consumption in a process, or you can use it for a line-by-line analysis of the memory consumption of your code.
- This package is not provided as a default Python package and it requires installation. Use the following commands to install the package and its dependencies from the command line:

```
ipython –m pip install psutil
ipython –m pip install memory_profiler
```


- Once it's installed, we need some code to run it against. The memory_profiler actually works in much the same way as line_profiler in that when you run it, memory_profiler will inject an instance of itself into __builtins__ named profile that you are supposed to use as a decorator on the function you are profiling.

5.5 Running in Parallel on Multiple Cores

- A ***parallel computer*** is a collection of processing elements that cooperate to solve large problems fast
- A form of parallel computing in which the computing platform is a group of interconnected computers i.e. cluster.
- Parallel computing is dividing up tasks over multiple microprocessors in order to make computers to run faster instead of using a single processor to perform one task at a time.
- Parallel computing is the simultaneous use of multiple compute resources to solve a computational problem. To be run using multiple CPUs. A problem is broken into discrete parts that can be solved concurrently. Each part is further broken down to a series of instructions. Instructions from each part execute simultaneously on different CPUs.
- Multiprocessing works by replicating the same code and memory content in various new Python instances (the workers), calculating the result for each of them, and returning the pooled results to the main original console.
- The multiprocessing Python module contains two classes capable of handling tasks. The Process class sends each task to a different processor, and the Pool class sends sets of tasks to different processors. We will show how to multiprocess the example code using both classes.
- In particular, Scikit-learn allows multiprocessing when

1. Cross-validating : Testing the results of a machine-learning hypothesis using different training and testing data.
2. Grid-searching : Systematically changing the hyper-parameters of a machine-learning hypothesis and testing the consequent results.
3. Multilabel prediction : Running an algorithm multiple times against multiple targets when there are many different target outcomes to predict at the same time

4. Ensemble machine-learning methods : Modeling a large host of classifiers, each one independent from the other, such as when using RandomForest-based modeling

5.5.1 Demonstrating Multiprocessing

1. Digits Dataset sklearn
   - The sklearn digits dataset is made up of 1797 8 × 8 images. Each image is of a hand-written digit.
   - In order to utilize an 8 × 8 figure, we will need to transform it into a feature vector with length 64.
2. Digits Dataset Analysis : This section will focus on the analysis of the sklearn iris dataset and learn about the dataset before we dive into visualization.
3. Load the Digits Dataset : We can load the digits dataset from the sklearn.datasets by using the load_digits() method. This will save the object containing digits data and the attributes associated with it.

```
from sklearn import datasets
import matplotlib.pyplot as plt
digits = datasets.load_digits()
```

5.6 Exploring Data Analysis

- Exploratory Data Analysis (EDA) is a general approach to exploring datasets by means of simple summary statistics and graphic visualizations in order to gain a deeper understanding of data.
- EDA is an approach/philosophy for data analysis that employs a variety of techniques to

1. Maximize insight into a data set;
2. Uncover underlying structure;
3. Extract important variables;
4. Detect outliers and anomalies;
5. Test underlying assumptions;
6. Develop parsimonious models; and
7. Determine optimal factor settings.

- With EDA, following functions are performed :

1. Describe of user data
2. Closely explore data distributions

3. Understand the relations between variables
4. Notice unusual or unexpected situations
5. Place the data into groups
6. Notice unexpected patterns within groups
7. Take note of group differences

5.6.1 Defining Descriptive Statistics for Numeric Data

- Descriptive statistics are the first pieces of information used to understand and represent a dataset. There goal, in essence, is to describe the main features of numerical and categorical information with simple summaries.
- These summaries can be presented with a single numeric measure, using summary tables, or via graphical representation.
- numPy is used in almost all numerical computations in Python. It is used for high-performance vector and matrix computations
- NumPy provides fast precompiled functions for numerical routines and written in C and Fortran. It support vectorized computations.

5.6.2 Measuring Central Tendency

- The three main measures of central tendency are the mean, median and mode. In any given set of data, these may be three very different values or values that are the same or close to the same.
- A measure of central tendency is a single value that attempts to describe a set of data by identifying the central position within that set of data. As such, measures of central tendency are sometimes called measures of central location.
- The mean, median and mode are all valid measures of central tendency, but under different conditions, some measures of central tendency become more appropriate to use than others.
- Example:

```
import pandas as pd
dataMatrix = {"D1":[135, 137, 136, 138, 138],
              "D2":[43,  42, 42, 42, 42],
              "D3":[72, 73, 72, 72, 73],
              "D4":[100, 102, 100, 103, 104]
            };
  dataFrame = pd.DataFrame(data=dataMatrix);

print("DataFrame:");
```

```
print(dataFrame);

print("Mean:Computed column-wise:");
meanData = dataFrame.mean();
print(meanData);

print("Mean:Computed row-wise:");
meanData = dataFrame.mean(axis=1);
print(meanData);

print("Median:Computed column-wise:");
medianData = dataFrame.median();
print(medianData);

print("Median:Computed row-wise:");
medianData = dataFrame.median(axis=1);
print(medianData);

print("Mode:Computed column-wise:");
modeData = dataFrame.mode();
print(modeData);

print("Mode:Computed row-wise:");
modeData = dataFrame.mode(axis=1);
print(modeData);
```

Output:

```
DataFrame:
   D1   D2 D3  D4
0  135 43 72  100
1  137 42  73  102
2  136 42  72  100
3  138 42  72  103
4  138 42  73  104
Mean:Computed column-wise:
D1   136.8
D2    42.2
D3    72.4
D4   101.8
dtype: float64
Mean:Computed row-wise:
0   87.50
1   88.50
2   87.50
3   88.75
4   89.25
```



```
dtype: float64
Median:Computed column-wise:
D1   137.0
D2    42.0
D3    72.0
D4   102.0
dtype: float64
Median:Computed row-wise:
0   86.0
1   87.5
2   86.0
3   87.5
4   88.5
dtype: float64
Mode:Computed column-wise:
   D1  D2  D3   D4
0  138  42  72  100
Mode:Computed row-wise:
   0   1   2   3
0  43  72  100  135
1  42  73  102  137
2  42  72  100  136
3  42  72  103  138
4  42  73  104  138
```

- The median provides the central position in the series of values. When creating a variable, it is a measure less influenced by anomalous cases or by an asymmetric distribution of values around the mean.

5.6.3 Measuring Variance and Range

- Variance Function in python pandas is used to calculate variance of a given set of numbers, Variance of a data frame, Variance of column or column wise variance in pandas python and Variance of rows or row wise variance in pandas python, let's see an example of each.

```
# calculate variance
import numpy as np
print(np.var([1,9,5,6,8,7]))
print(np.var([4,-11,-5,16,5,7,9]))
```

output:

```
2.82842712475
8.97881103594
```

- **To create dataframe:**

```
import pandas as pd
import numpy as np

#Create a DataFrame
d = {
    'Name':['Rupali','Rakshita','Rutuja','Ritesh','Radhika','Rushikesh','Unnati',
    'Shreshta','Nidhi','Samriddhi','Gaurav','Shruti'],
   'Score1':[62,47,55,74,31,77,85,63,42,32,71,57],
   'Score2':[89,87,67,55,47,72,76,79,44,92,99,69],
   'Score3':[56,86,77,45,73,62,74,89,71,67,97,68]}
df = pd.DataFrame(d)
print df
```

- In addition, the range, which is the difference between the maximum and minimum value for each quantitative variable, is quite informative.

5.6.4 Defining Measures of Normality

- Skewness defines the asymmetry of data with respect to the mean. If the skew is negative, the left tail is too long and the mass of the observations are on the right side of the distribution. If it is positive, it is exactly the opposite.
- Kurtosis shows whether the data distribution, especially the peak and the tails, are of the right shape. If the kurtosis is above zero, the distribution has a marked peak. If it is below zero, the distribution is too flat instead.
- Skewness is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the centre point.
- Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution. That is, data sets with high kurtosis tend to have heavy tails, or outliers. Data sets with low kurtosis tend to have light tails, or lack of outliers. A uniform distribution would be the extreme case.
- The histogram is an effective graphical technique for showing both the skewness and kurtosis of data set.

5.6.5 Counting for Categorical Data

- The pandas package features two useful functions, cut and qcut, that can transform a metric variable into a qualitative one :

1. cut expects a series of edge values used to cut the measurements or an integer number of groups used to cut the variables into equal-width bins.
2. qcut expects a series of percentiles used to cut the variable

- Pandas cut() function is used to segregate array elements into separate bins. The cut() function works only on one-dimensional array-like objects.

Example:

```
import pandas as pd
import numpy as np
df = pd.DataFrame({
   'x': [3, 3, 1, 10, 1, 10],
   'y': [1, 2, 3, 4, 5, 60],
   'z': [6, 5, 4, 3, 2, 1]
})
df
```

Output:

	x	y	z
0	3	1	6
1	3	2	5
2	1	3	4
3	10	4	3
4	1	5	2
5	10	60	1

- The cut() function is useful when we have a large number of scalar data and we want to perform some statistical analysis on it.
- For example, let's say we have an array of numbers between 1 and 20. We want to divide them into two bins of (1, 10] and (10, 20] and add labels such as "Lows" and "Highs". We can easily perform this using the pandas cut() function.
- Furthermore, we can perform functions on the elements of a specific bin and label elements.
- **Syntax:**

```
cut(
  x,
  bins,
  right=True,
  labels=None,
```

```
  retbins=False,
  precision=3,
  include_lowest=False,
  duplicates="raise",
)
```

- x is the input array to be binned. It must be one-dimensional.
- **bins** defines the bin edges for the segmentation.
- **right** indicates whether to include the rightmost edge or not, default value is True.
- **labels** is used to specify the labels for the returned bins.
- **retbins** specifies whether to return the bins or not.
- **precision** specifies the precision at which to store and display the bins labels.
- **include_lowest** specifies whether the first interval should be left-inclusive or not.
- **duplicates** speicifies what to do if the bins edges are not unique, whether to raise ValueError or drop non-uniques.
- The major distinction is that qcut will calculate the size of each bin in order to make sure the distribution of data in the bins is equal. In other words, all bins will have (roughly) the same number of observations but the bin range will vary.
- On the other hand, cut is used to specifically define the bin edges. There is no guarantee about the distribution of items in each bin. In fact, you can define bins in such a way that no items are included in a bin or nearly all items are in a single bin.
- If you want equal distribution of the items in your bins, use qcut . If you want to define your own numeric bin ranges, then use cut .

5.6.6 Understanding Frequencies

- we can obtain a frequency for each categorical variable of the dataset, both for the predictive variable and for the outcome, by using the following code :

```
print iris_dataframe['group'].value_counts()
virginica    50
versicolor   50
setosa       50
print iris_binned['petal length (cm)'].value_counts()
[1, 1.6]        44
(4.35, 5.1]     41
(5.1, 6.9]      34
(1.6, 4.35]     31
```


- This example provides you with some basic frequency information as well, such as the number of unique values in each variable and the mode of the frequency (top and freq rows in the output).

5.6.7 Creating Contingency Tables

- The pandas.crosstab function can match variables or groups of variables, helping to locate possible data structures or relationships.
- This function is used to get an initial "feel" (view) of the data. Here, we can validate some basic hypothesis. For instance, in this case, "Credit_History" is expected to affect the loan status significantly. This can be tested using cross-tabulation as shown below:

```
pd.crosstab(data["Credit_History"],data["Loan_Status"],margins=True)
```

Loan_Status	N	Y	A
Credit_History			
10	82	7	84
20	97	378	45
31	192	422	62

- These are absolute numbers. But, percentages can be more intuitive in making some quick insights.
- The pandas.crosstab function ignores categorical variable ordering and always displays the row and column categories according to their alphabetical order.

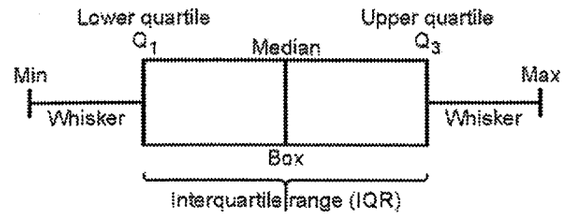
5.7 Creating Applied Visualization for EDA

- Box plots are an excellent tool for conveying location and variation information in data sets, particularly for detecting and illustrating location and variation changes between different groups of data.
- Exploratory Data Analysis is majorly performed using the following methods :
  1. Univariate analysis: - provides summary statistics for each field in the raw data set (or) summary only on one variable. Ex:- CDF,PDF,Box plot
  2. Bivariate analysis is performed to find the relationship between each variable in the dataset and the target variable of interest (or) using 2 variables and finding realtionship between them. Ex:-Box plot,Voilin plot.

3. Multivariate analysis is performed to understand interactions between different fields in the dataset (or) finding interactions between variables more than 2.

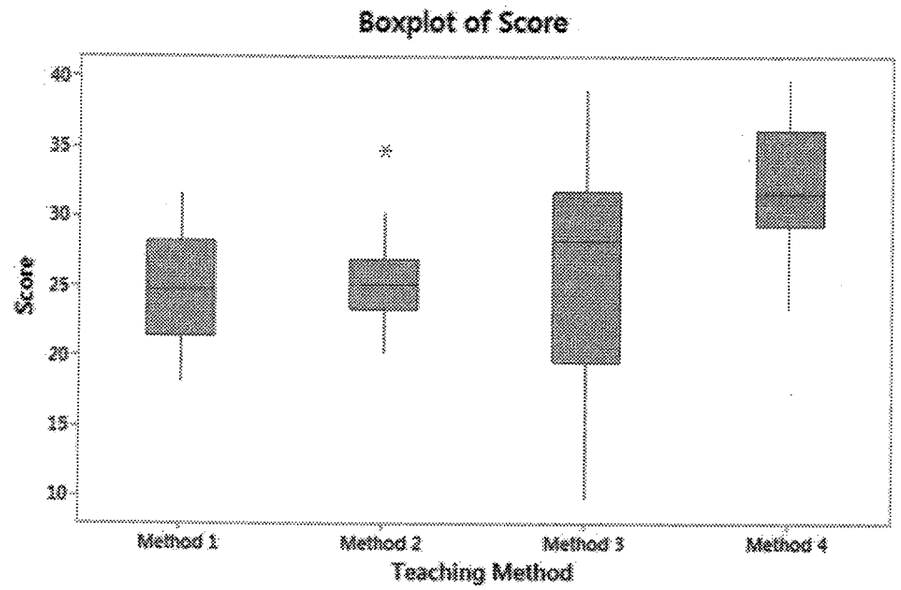
5.7.1 Inspecting Boxplots

- A box plot is a type of chart often used in explanatory data analysis to visually show the distribution of numerical data and skewness through displaying the data quartiles or percentile and averages.



1. Minimum Score : The lowest score, excluding outliers.
2. Lower Quartile : 25 % of scores fall below the lower quartile value.
3. Median: The median marks the mid-point of the data and is shown by the line that divides the box into two parts.
4. Upper Quartile : 75 % of the scores fall below the upper quartile value.
5. Maximum Score : The highest score, excluding outliers.
6. Whiskers : The upper and lower whiskers represent scores outside the middle 50 %
7. The Interquartile Range : This is the box plot showing the middle 50 % of scores.

- Boxplots are also extremely useful for visually checking group differences. Suppose we have four groups of scores and we want to compare them by teaching method. Teaching method is our categorical grouping variable and Score is the continuous outcome variable that the researchers measured.



5.7.2 Performing t-tests after Boxplots

- The t-test is a parametric test for comparing two sets of continuous data. It is used for comparing two sample means.
- The scipy.stats function ttest_1samp can be used to calculate a T-statistic and a P-value for a given mean value compared to a sample population.
- Syntax:

```
scipy.stats.ttest_ind_from_stats(mean1, std1, nobs1, mean2, std2, nobs2, equal_var=True)[source]
```

where

1. mean1array_like : The mean(s) of sample 1.
2. std1array_like : The standard deviation(s) of sample 1.
3. nobs1array_like: The number(s) of observations of sample 1.
4. mean2array_like: The mean(s) of sample 2.
5. std2array_like:The standard deviations(s) of sample 2.
6. nobs2array_like:The number(s) of observations of sample 2.
7. equal_varbool, optional: If True (default), perform a standard independent 2 sample test that assumes equal population variances. If False, perform Welch's t-test, which does not assume equal population variance.

5.7.3 Observing Parallel Coordinates

- In a parallel coordinates plot with px.parallel_coordinates, each row of the DataFrame is represented by a polyline mark which traverses a set of parallel axes, one for each of the dimensions.
- For other representations of multivariate data, also see parallel categories, radar charts and scatterplot matrix (SPLOM).
- Example:

```
def test_parallel_coords(pandas=False, outpath=None):
    """
    Runs the parallel coordinates visualizer on the dataset.
    Parameters
    ----------
    pandas : bool
        Run the pandas version of the function
    outpath : path or None
        Save the figure to disk rather than show (if None)
    """
    data = load_data('occupancy')                    # Load the data
    features = ['temp', 'humid', 'light', 'co2', 'hratio']
    classes = ['unoccupied', 'occupied']
    X = data[features].as_matrix()
    y = data.occupied.as_matrix()

    if pandas:
        parallel_coordinates(data[features + ['occupied']], 'occupied')
        if outpath:
            plt.savefig(outpath)
        else:
            plt.show()
    else:
        visualizer = ParallelCoordinates(                # Instantiate the visualizer
            classes=classes, features=features
        )
        visualizer.fit(X, y)             # Fit the data to the visualizer
        visualizer.transform(X)          # Transform the data
        visualizer.poof(outpath=outpath)  # Draw/show/poof the data
```

5.8 Understanding Correlation

- Correlation gives the changes in one variable due to changes in the other variable.
- Scatter diagram, correlation coefficient and rank correlation coefficient are the common measures of correlation coefficients.

- The Pearson's correlation is the foundation for complex linear estimation models.

Using covariance and correlation

- Covariance is the first measure of the relationship of two variables. It determines whether both variables have a coincident behavior with respect to their mean.
- If the single values of two variables are usually above or below their respective averages, the two variables have a positive association.
- Consider the two plots shown below. In both images, one thousand samples drawn from an underlying joint distribution. Clearly the two distributions are different and shown in Fig. 5.8.1. However, the mean and variance are the same in both the x and the y dimension. What is different ?

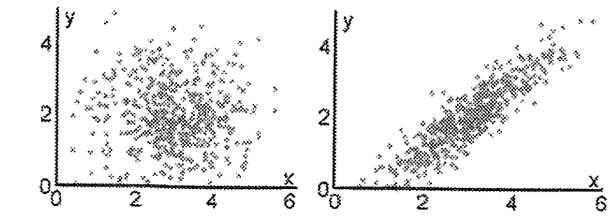


Fig. 5.8.1 Distribution

- Covariance is a quantitative measure of the extent to which the deviation of one variable from its mean matches the deviation of the other from its mean.
- Correlation : Covariance is interesting because it is a quantitative measurement of the relationship between two variables. Correlation between two random variables, $\varrho(X,Y)$ is the covariance of the two variables normalized by the variance of each variable. This normalization cancels the units out and normalizes the measure so that it is always in the range [0, 1].
- When people use the term correlation, they are actually referring to a specific type of correlation called "Pearson" correlation. It measures the degree to which there is a linear relationship between the two variables.
- An alternative measure is "Spearman" correlation, which has a formula almost identical to the correlation defined above, with the exception that the underlying random variables are first transformed into their rank
- The difference between variance, covariance, and correlation is :

a. Variance is a measure of variability from the mean

b. Covariance is a measure of relationship between the variability of 2 variables - covariance is scale dependent because it is not standardized

c. Correlation is a of relationship between the variability of of 2 variables - correlation is standardized making it not scale dependent

- The covariance matrix element Cij is the covariance of xi and xj. The element Cii is the variance of xi.
  a. If COV(xi, xj) = 0 then variables are uncorrelated
  b. If COV(xi, xj) > 0 then variables positively correlated
  c. If COV(xi, xj) > < 0 then variables negatively correlated
- **Syntax:**

```
numpy.cov(m, y=None, rowvar=True, bias=False, ddof=None, fweights=None, aweights=None)
```

where :

m : [array_like] A 1D or 2D variables. variables are columns

y : [array_like] It has the same form as that of m.

rowvar : [bool, optional] If rowvar is True (default), then each row represents a variable, with observations in the columns. Otherwise, the relationship is transposed:

bias : Default normalization is False. If bias is True it normalize the data points.

ddof : If not None the default value implied by bias is overridden. Note that ddof=1 will return the unbiased estimate, even if both fweights and aweights are specified.

fweights : fweight is 1-D array of integer frequency weights

aweights : aweight is 1-D array of observation vector weights.

Returns : It returns ndarray covariance matrix

5.9 Considering the Chi-Square Test for Tables

- Scientists will often use the Chi-square test to determine the goodness of fit between theoretical and experimental data. In this test, we compare observed values with theoretical or expected values.
- Observed values are those that the researcher obtains empirically through direct observation; theoretical or expected values are developed on the basis of some hypothesis. The chi-square measure value depends on how many cells the table has.

Modifying Data Distributions :

- Normal distribution is also called **Gauss distribution**.
- The normal distribution describes a special class of such distributions that are symmetric and can be described by the distribution mean and the standard deviation
- **What the normal distribution looks like ?**

1) The mean is located in the center of the distribution.
2) Distribution is symmetric about its mean.

3) Shape of distribution is determined by standard deviation : Large value of SD reduce the height and increase the spread of the curve, small value of SD increase the height and reduce the spread of the curve.
4) Almost all of the distribution will lie within 3 deviations of the mean.
5) The total area under the curve is 1.
6) The curve extends indefinitely in both directions, approaching, but never touching, the horizontal axis as it does so.

❑❑❑

Notes

SOLVED MODEL QUESTION PAPER
Python for Data Science
Semester - V (CE / CSE)

Time : Three Hours] **[Maximum Marks : 70**

Answer ALL Questions

Q.1 **a)** *What is python ? List its features.* **(Refer sections 1.1 and 1.1.1)** **[3]**

b) *Compare list and table.* **(Refer section 1.4.7)** **[4]**

c) *What is dictionary ? Explain with example.* **(Refer section 1.4.6)** **[7]**

Q.2 **a)** *Explain role of python in data science.* **(Refer section 2.3)** **[3]**

b) *Explain working at the command line.* **(Refer section 2.6.1)** **[4]**

c) *How to implement machine learning using scikit-learn ?* **(Refer section 2.8)** **[7]**

OR

c) *How to link data science with big data ?* **(Refer section 2.1.2)** **[7]**

Q.3 **a)** *What is Jupyter console ?* **(Refer section 3.1)** **[3]**

b) *Define an adjacency matrix. Explain with example.* **(Refer section 3.7.3)** **[4]**

c) *Explain finding missing data in python.* **(Refer section 3.3.6)** **[7]**

OR

Q.3 **a)** *What is Pandas ? Explain.* **(Refer section 3.3)** **[3]**

b) *How to restart the Kernel ?* **(Refer section 3.1.3)** **[4]**

c) *Explain different use / working of real data in python.* **(Refer section 3.2)** **[7]**

Q.4 **a)** *How to represent time on axes ?* **(Refer section 4.6)** **[3]**

b) *Define histograms. Write code for creating histogram.* **(Refer section 4.3)** **[4]**

c) *What is pie chart and bar chart ? Explain how these charts are used for visualizing the data.* **(Refer section 4.2)** **[7]**

OR

Q.4 **a)** *Define axis, ticks and grids.* **(Refer section 4.1.3)** **[3]**

b) *Define graph. What are methods used for visualizing information ?* **(Refer section 4.1)** **[4]**

c) *What is box plots ? How scatterplots helps for seeing data patterns ?* **(Refer sections 4.4 and 4.5)** **[7]**

Q.5 a) *Explain hash functions.* **(Refer section 5.3.1)** **[3]**

b) *Describe in detail covariance and correlation.* **[4]**

c) *What is EDA ? List the functions performed by EDA. How to measure control tendency ?* **(Refer section 5.6)** **[7]**

OR

Q.5 a) *What is memory profiler ?* **(Refer section 5.4.1)** **[3]**

b) *Explain classes used in scikit-learn.* **(Refer section 5.2.1)** **[4]**

c) *Define data wrangling ? Explain steps performed in data wrangling process.* **(Refer section 5.1)** **[7]**

❑❑❑

Notes