**National University of Computer and Emerging Sciences**

# Lab Manual 02
# Fundamentals of Big Data Lab

| | |
|---|---|
| Course Instructor | Dr. Iqra Safdar |
| Lab Instructor (s) | Muhammad Mazarib<br>M Aiss Shahid |
| Section | B1,B2 |
| Semester | Spring 2022 |

## What is CSV File?

A CSV file is a comma-separated values file. The csv file format allows data to be saved in a tabular format. The CSV file is just a plain text file in which data is separated by commas.

Suppose we have a csv file: **data.csv** and its contents are as follow,

```
1,2,3,4,5

6,7,8,9,0

2,3,4,5,6

4,5,6,7,7
```

Now we want to load this CSV file into a NumPy Array.

There are multiple ways to Read CSV file into a NumPy Array in Python. Lets discuss all the methods one by one with proper approach and a working code example

## Read CSV File into a NumPy Array using loadtxt()

The numpy module has a loadtxt() function and it is used to load data from a text file. Each row in the text file must have same number of values.

## Syntax of loadtxt() function

```
numpy.loadtxt(fname, delimiter, skiprows)
```

- Parameters:
  - fname = Name or path of the file to be loaded.
  - delimiter = The string used to separate values, By default the delimiter is whitespace.
  - skiprows = The Number of rows to be skipped.
- Returns:
  - Returns an array.

## Approach:

1. Import numpy library.
2. Pass the path of the csv file and delimiter as comma (,) to loadtxt() method.
3. Print the Array returned by the loadtxt() method.

## Source Code

```
import numpy as np

# Reading csv file into numpy array

arr = np.loadtxt("data.csv", delimiter=",")
```

```
# printing the array

print(arr)
```

**Output:**

```
[[1. 2. 3. 4. 5.]

[6. 7. 8. 9. 0.]

[2. 3. 4. 5. 6.]

[4. 5. 6. 7. 7.]]
```

## Read CSV File into a NumPy Array using genfromtxt()

The numpy module have genfromtxt() function, and it is used to Load data from a text file. The genfromtxt() function can handle rows with missing values as specified.

### Syntax of genfromtxt() function

```
numpy.genfromtxt(fname, delimiter)
```

- Parameters:
  - fname = Name or path of the file to be loaded.
  - delimiter = The string used to separate values, By default the delimiter is whitespace.
- Returns:
  - Returns an array.

## Approach:

1. Import numpy library.
2. Pass the path of the csv file and delimiter as comma (,) to genfromtxt() method.
3. Print the Array returned by the genfromtxt() method.

## Source Code

```
import numpy as np

# Reading csv file into numpy array

arr = np.genfromtxt("data.csv",delimiter=",")

# printing the array

print(arr)
```

## Output:

```
[[1. 2. 3. 4. 5.]

[6. 7. 8. 9. 0.]

[2. 3. 4. 5. 6.]

[4. 5. 6. 7. 7.]]
```

## Read CSV File into a NumPy Array using read_csv()

The pandas module has a read_csv() method, and it is used to Read a comma-separated values (csv) file into DataFrame, By using the values property of the dataframe we can get the numpy array.

### Syntax of read_csv() function

```
numpy.read_csv(file_path, sep, header)
```

- Parameters:
    - file_path = Name or path of the csv file to be loaded.
    - sep = The string used to separate values i.e, delimiter , By default the delimiter is comma (,).
    - header = The names of the columns.
- Returns:
    - Returns an DataFrame.

### Approach:

1. Import pandas and numpy library.
2. Pass the path of the csv file and header as None to read_csv() method.
3. Now use the **values** property of DataFrame to get numpy array from the dataframe.
4. Print the numpy array.

### Source Code

```
import numpy as np

import pandas as pd

# Reading csv file into numpy array

arr = pd.read_csv('data.csv', header=None).values

# printing the array

print(arr)
```

**Output:**

```
[[1 2 3 4 5]

[6 7 8 9 0]

[2 3 4 5 6]

[4 5 6 7 7]]
```

## Read CSV File into a NumPy Array using file handling and fromstring()

Python supports file handling and provides various functions for reading, writing the files. The numpy module provides fromstring() method, and it is used to make a numpy array from a string. Now to convert a CSV file into numpy array, read the csv file using file handling. Then, for each row in the file, convert the row into numpy array using fromstring() method, and join all the arrays.

**Syntax of read_csv() function**

```
numpy.fromstring(str, sep)
```

- Parameters:
  - str = A string containing the data..
  - sep = The string separating numbers in the data by default the sep is a whitespace.
- Returns:
  - Returns an numpy array.

**Syntax of open() function**

```
open(file, mode)
```

- Parameters:
  - file = Name or path of the file to be loaded.
  - mode = This specifies the access mode of opening a file by default the mode is read mode.
- Returns:
  - Returns a file object.

**Approach:**

1. Import numpy library.
2. Open the csv file in read mode and read each row of the csv file.

3. pass each row of the csv file and sep=”,” to the fromstring() method.
4. the fromstring method will return a numpy array append it to a list
5. Repeat step 3 and 4 till the last row of csv file.
6. Convert the list into numpy array and print it.

**Source Code**

```
import numpy as np

# Reading csv file into numpy array

file_data = open('data.csv')

l=[]

for row in file_data:

   r = list(np.fromstring(row, sep=","))

   l.append(r)

# printing the array

print(np.array(l))
```

**Output:**

```
[[1. 2. 3. 4. 5.]

[6. 7. 8. 9. 0.]

[2. 3. 4. 5. 6.]

[4. 5. 6. 7. 7.]]
```

## Summary

Great! you made it, We have discussed All possible methods to Read CSV file into a NumPy Array in Python. Happy learning.

### Example 1: Write into CSV files with csv.writer()

Suppose we want to write a CSV file with the following entries:

```
SN,Name,Contribution
1,Linus Torvalds,Linux Kernel
2,Tim Berners-Lee,World Wide Web
3,Guido van Rossum,Python Programming
```

Here's how we do it.

```
import csv
with open('innovators.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["SN", "Name", "Contribution"])
    writer.writerow([1, "Linus Torvalds", "Linux Kernel"])
    writer.writerow([2, "Tim Berners-Lee", "World Wide Web"])
    writer.writerow([3, "Guido van Rossum", "Python Programming"])
```

When we run the above program, an innovators.csv file is created in the current working directory with the given entries.

Here, we have opened the innovators.csv file in writing mode using `open()` function.

Next, the `csv.writer()` function is used to create a `writer` object. The `writer.writerow()` function is then used to write single rows to the CSV file.

---

## Example 2: Writing Multiple Rows with writerows()

If we need to write the contents of the 2-dimensional list to a CSV file, here's how we can do it.

```
import csv
row_list = [["SN", "Name", "Contribution"],
             [1, "Linus Torvalds", "Linux Kernel"],
             [2, "Tim Berners-Lee", "World Wide Web"],
             [3, "Guido van Rossum", "Python Programming"]]
with open('protagonist.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(row_list)
```

The output of the program is the same as in Example 1.

Here, our 2-dimensional list is passed to the `writer.writerows()` function to write the content of the list to the CSV file.

---

Now let's see how we can write CSV files in different formats. We will then learn how to customize the `csv.writer()` function to write them.

---

## CSV Files with Custom Delimiters

By default, a comma is used as a delimiter in a CSV file. However, some CSV files can use delimiters other than a comma. Few popular ones are `|` and `\t`.

Suppose we want to use `|` as a delimiter in the innovators.csv file of Example 1. To write this file, we can pass an additional `delimiter` parameter to the `csv.writer()` function.

Let's take an example.

### Example 3: Write CSV File Having Pipe Delimiter

```
import csv
data_list = [["SN", "Name", "Contribution"],
             [1, "Linus Torvalds", "Linux Kernel"],
             [2, "Tim Berners-Lee", "World Wide Web"],
             [3, "Guido van Rossum", "Python Programming"]]
with open('innovators.csv', 'w', newline='') as file:
    writer = csv.writer(file, delimiter='|')
    writer.writerows(data_list)
```

Output

```
SN|Name|Contribution
1|Linus Torvalds|Linux Kernel
2|Tim Berners-Lee|World Wide Web
3|Guido van Rossum|Python Programming
```

As we can see, the optional parameter `delimiter = '|'` helps specify the `writer` object that the CSV file should have `|` as a delimiter.

---

## CSV files with Quotes

Some CSV files have quotes around each or some of the entries.

Let's take quotes.csv as an example, with the following entries:

```
"SN";"Name";"Quotes"
1;"Buddha";"What we think we become"
2;"Mark Twain";"Never regret anything that made you smile"
3;"Oscar Wilde";"Be yourself everyone else is already taken"
```

Using `csv.writer()` by default will not add these quotes to the entries.

In order to add them, we will have to use another optional parameter called `quoting`.

Let's take an example of how quoting can be used around the non-numeric values and `;` as delimiters.

## Example 4: Write CSV files with quotes

```python
import csv
row_list = [
    ["SN", "Name", "Quotes"],
    [1, "Buddha", "What we think we become"],
    [2, "Mark Twain", "Never regret anything that made you smile"],
    [3, "Oscar Wilde", "Be yourself everyone else is already taken"]
]
with open('quotes.csv', 'w', newline='') as file:
    writer = csv.writer(file, quoting=csv.QUOTE_NONNUMERIC, delimiter=';')
    writer.writerows(row_list)
```

Output

```
"SN";"Name";"Quotes"
1;"Buddha";"What we think we become"
2;"Mark Twain";"Never regret anything that made you smile"
3;"Oscar Wilde";"Be yourself everyone else is already taken"
```

Here, the quotes.csv file is created in the working directory with the above entries.

As you can see, we have passed `csv.QUOTE_NONNUMERIC` to the `quoting` parameter. It is a constant defined by the `csv` module.

`csv.QUOTE_NONNUMERIC` specifies the `writer` object that quotes should be added around the non-numeric entries.

There are 3 other predefined constants you can pass to the `quoting` parameter:

- `csv.QUOTE_ALL` - Specifies the `writer` object to write CSV file with quotes around all the entries.
- `csv.QUOTE_MINIMAL` - Specifies the `writer` object to only quote those fields which contain special characters (delimiter, quotechar or any characters in lineterminator)
- `csv.QUOTE_NONE` - Specifies the `writer` object that none of the entries should be quoted. It is the default value.

---

## CSV files with custom quoting character

We can also write CSV files with custom quoting characters. For that, we will have to use an optional parameter called `quotechar`.

Let's take an example of writing quotes.csv file in Example 4, but with `*` as the quoting character.

### Example 5: Writing CSV files with custom quoting character

```
import csv
row_list = [
    ["SN", "Name", "Quotes"],
    [1, "Buddha", "What we think we become"],
    [2, "Mark Twain", "Never regret anything that made you smile"],
    [3, "Oscar Wilde", "Be yourself everyone else is already taken"]
]
with open('quotes.csv', 'w', newline='') as file:
    writer = csv.writer(file, quoting=csv.QUOTE_NONNUMERIC,
                        delimiter=';', quotechar='*')
    writer.writerows(row_list)
```

Output

```
*SN*;*Name*;*Quotes*
1;*Buddha*;*What we think we become*
2;*Mark Twain*;*Never regret anything that made you smile*
3;*Oscar Wilde*;*Be yourself everyone else is already taken*
```

Here, we can see that `quotechar='*'` parameter instructs the `writer` object to use `*` as quote for all non-numeric values.

---

## Dialects in CSV module

Notice in Example 5 that we have passed multiple parameters (`quoting`, `delimiter` and `quotechar`) to the `csv.writer()` function.

This practice is acceptable when dealing with one or two files. But it will make the code more redundant and ugly once we start working with multiple CSV files with similar formats.

As a solution to this, the `csv` module offers `dialect` as an optional parameter.

---

Dialect helps in grouping together many specific formatting patterns like `delimiter`, `skipinitialspace`, `quoting`, `escapechar` into a single dialect name.

It can then be passed as a parameter to multiple `writer` or `reader` instances.

## Example 6: Write CSV file using dialect

Suppose we want to write a CSV file (office.csv) with the following content:

```
"ID"|"Name"|"Email"
"A878"|"Alfonso K. Hamby"|"alfonsokhamby@rhyta.com"
"F854"|"Susanne Briard"|"susannebriard@armyspy.com"
"E833"|"Katja Mauer"|"kmauer@jadoop.com"
```

The CSV file has quotes around each entry and uses `|` as a delimiter.

Instead of passing two individual formatting patterns, let's look at how to use dialects to write this file.

```
import csv
row_list = [
    ["ID", "Name", "Email"],
    ["A878", "Alfonso K. Hamby", "alfonsokhamby@rhyta.com"],
    ["F854", "Susanne Briard", "susannebriard@armyspy.com"],
    ["E833", "Katja Mauer", "kmauer@jadoop.com"]
]
csv.register_dialect('myDialect',
                     delimiter='|',
                     quoting=csv.QUOTE_ALL)
with open('office.csv', 'w', newline='') as file:
    writer = csv.writer(file, dialect='myDialect')
    writer.writerows(row_list)
```

Output

```
"ID"|"Name"|"Email"
"A878"|"Alfonso K. Hamby"|"alfonsokhamby@rhyta.com"
"F854"|"Susanne Briard"|"susannebriard@armyspy.com"
"E833"|"Katja Mauer"|"kmauer@jadoop.com"
```

Here, office.csv is created in the working directory with the above contents.

From this example, we can see that the `csv.register_dialect()` function is used to define a custom dialect. Its syntax is:

```
csv.register_dialect(name[, dialect[, **fmtparams]])
```

The custom dialect requires a name in the form of a string. Other specifications can be done either by passing a sub-class of the `Dialect` class, or by individual formatting patterns as shown in the example.

---

While creating the `writer` object, we pass `dialect='myDialect'` to specify that the writer instance must use that particular dialect.

The advantage of using `dialect` is that it makes the program more modular. Notice that we can reuse myDialect to write other CSV files without having to re-specify the CSV format.

---

## Write CSV files with csv.DictWriter()

The objects of `csv.DictWriter()` class can be used to write to a CSV file from a Python dictionary.

The minimal syntax of the `csv.DictWriter()` class is:

```
csv.DictWriter(file, fieldnames)
```

Here,

- `file` - CSV file where we want to write to
- `fieldnames` - a `list` object which should contain the column headers specifying the order in which data should be written in the CSV file

## Example 7: Python csv.DictWriter()

```
import csv

with open('players.csv', 'w', newline='') as file:
    fieldnames = ['player_name', 'fide_rating']
    writer = csv.DictWriter(file, fieldnames=fieldnames)

    writer.writeheader()
    writer.writerow({'player_name': 'Magnus Carlsen', 'fide_rating': 2870})
    writer.writerow({'player_name': 'Fabiano Caruana', 'fide_rating': 2822})
    writer.writerow({'player_name': 'Ding Liren', 'fide_rating': 2801})
```

Output

The program creates a players.csv file with the following entries:

```
player_name,fide_rating
Magnus Carlsen,2870
Fabiano Caruana,2822
```

```
Ding Liren,2801
```

---

The full syntax of the `csv.DictWriter()` class is:

```
csv.DictWriter(f, fieldnames, restval='', extrasaction='raise', dialect='excel',
*args, **kwds)
```

---

## CSV files with lineterminator

A `lineterminator` is a string used to terminate lines produced by `writer` objects. The default value is `\r\n`. You can change its value by passing any string as a `lineterminator` parameter.

However, the `reader` object only recognizes `\n` or `\r` as `lineterminator` values. So using other characters as line terminators is highly discouraged.

---

## doublequote & escapechar in CSV module

In order to separate delimiter characters in the entries, the `csv` module by default quotes the entries using quotation marks.

So, if you had an entry: `He is a strong, healthy man`, it will be written as: `"He is a strong, healthy man"`.

Similarly, the `csv` module uses double quotes in order to escape the quote character present in the entries by default.

If you had an entry: `Go to "programiz.com"`, it would be written as: `"Go to ""programiz.com"""`.

Here, we can see that each `"` is followed by a `"` to escape the previous one.

---

## doublequote

It handles how `quotechar` present in the entry themselves are quoted. When `True`, the quoting character is doubled and when `False`, the `escapechar` is used as a prefix to the `quotechar`. By default its value is `True`.

## escapechar

`escapechar` parameter is a string to escape the delimiter if quoting is set to `csv.QUOTE_NONE` and quotechar if doublequote is `False`. Its default value is None.

## Example 8: Using escapechar in csv writer

```
import csv
row_list = [
    ['Book', 'Quote'],
    ['Lord of the Rings',
        '"All we have to decide is what to do with the time that is given us."'],
    ['Harry Potter', '"It matters not what someone is born, but what they grow to be."']
]
with open('book.csv', 'w', newline='') as file:
    writer = csv.writer(file, escapechar='/', quoting=csv.QUOTE_NONE)
    writer.writerows(row_list)
```

Output

```
Book,Quote
Lord of the Rings,/"All we have to decide is what to do with the time that is given us./"
Harry Potter,/"It matters not what someone is born/, but what they grow to be./"
```

Here, we can see that `/` is prefix to all the `"` and `,` because we specified `quoting=csv.QUOTE_NONE`.

If it wasn't defined, then, the output would be:

```
Book,Quote
Lord of the Rings,"""All we have to decide is what to do with the time that is given us."""
Harry Potter,"""It matters not what someone is born, but what they grow to be."""
```

Since we allow quoting, the entries with special characters(`"` in this case) are double-quoted. The entries with `delimiter` are also enclosed within quote characters.(Starting and closing quote characters)

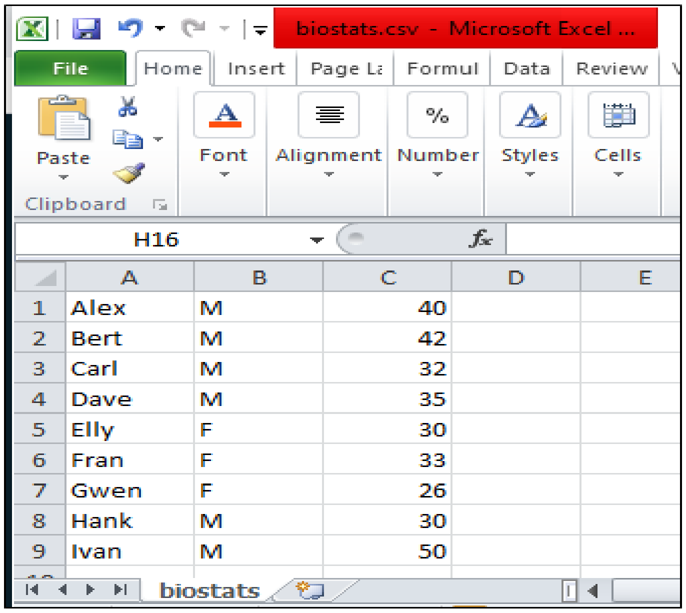
The remaining quote characters are to escape the actual `"` present as part of the string, so that they are not interpreted as quotechar.

---

Note: The csv module can also be used for other file extensions (like: .txt) as long as their contents are in proper structure.

**Example 1: Visualizing the column of different persons through bar plot.**

The below CSV file contains different person name, gender, and age saved as 'biostats.csv':



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Alex | M | 40 | | |
| 2 | Bert | M | 42 | | |
| 3 | Carl | M | 32 | | |
| 4 | Dave | M | 35 | | |
| 5 | Elly | F | 30 | | |
| 6 | Fran | F | 33 | | |
| 7 | Gwen | F | 26 | | |
| 8 | Hank | M | 30 | | |
| 9 | Ivan | M | 50 | | |

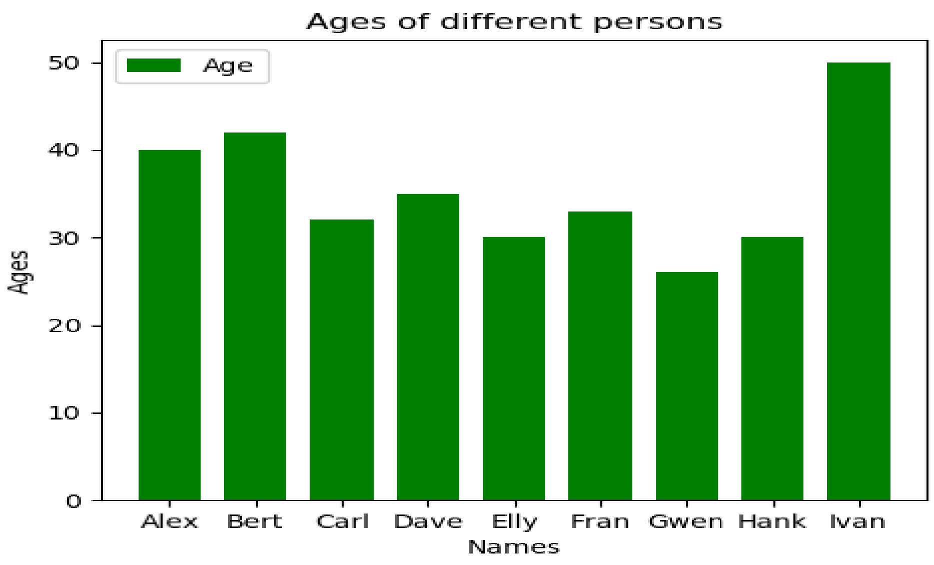
**The approach of the program:**

1. Import required libraries, matplotlib library for visualizing, and CSV library for reading CSV data.
2. Open the file using open( ) function with 'r' mode (read-only) from CSV library and read the file using csv.reader( ) function.
3. Read each line in the file using for loop.
4. Append required columns into a list.
5. After reading the whole CSV file, plot the required data as X and Y axis.
6. In this example, we are plotting names as X-axis and ages as Y-axis.

**Below is the implementation:**

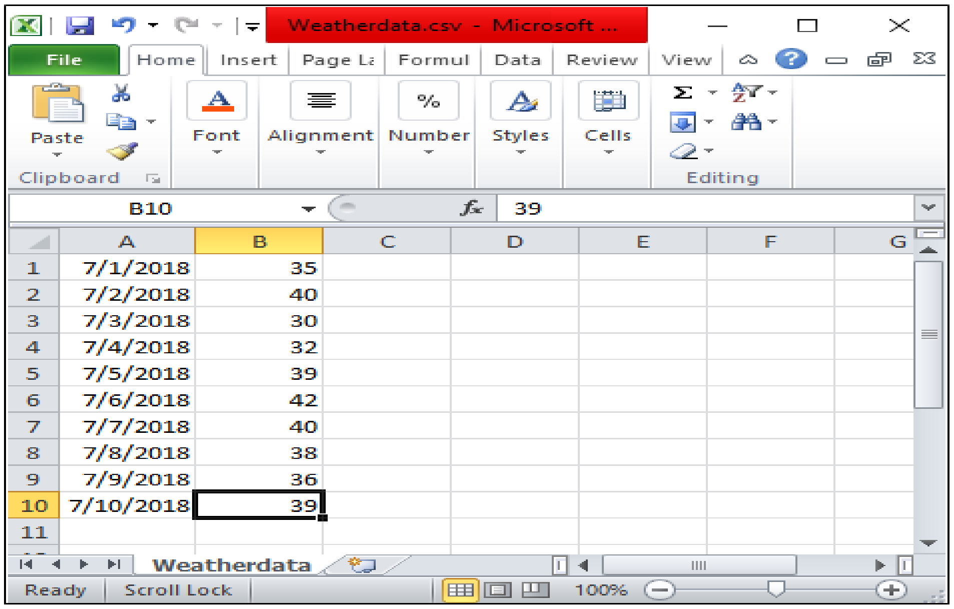
```
import matplotlib.pyplot as plt
import csv

x = []
y = []

with open('biostats.csv','r') as csvfile:
    plots = csv.reader(csvfile, delimiter =
',')

    for row in plots:
        x.append(row[0])
        y.append(int(row[2]))

plt.bar(x, y, color = 'g', width = 0.72, label
= "Age")
plt.xlabel('Names')
plt.ylabel('Ages')
plt.title('Ages of different persons')
plt.legend()
plt.show()
```

**Output :**



**Example 2: Visualizing Weather Report on different Dates”** through-line **plot.**

Temperature(°C) on different dates is stored in a CSV file as ‘Weatherdata.csv’. These two rows ‘Dates’ and ‘Temperature(°C )’ are used as X and Y-axis for visualizing weather reports.



B10 = 39

| | A | B |
|---|---|---|
| 1 | 7/1/2018 | 35 |
| 2 | 7/2/2018 | 40 |
| 3 | 7/3/2018 | 30 |
| 4 | 7/4/2018 | 32 |
| 5 | 7/5/2018 | 39 |
| 6 | 7/6/2018 | 42 |
| 7 | 7/7/2018 | 40 |
| 8 | 7/8/2018 | 38 |
| 9 | 7/9/2018 | 36 |
| 10 | 7/10/2018 | 39 |

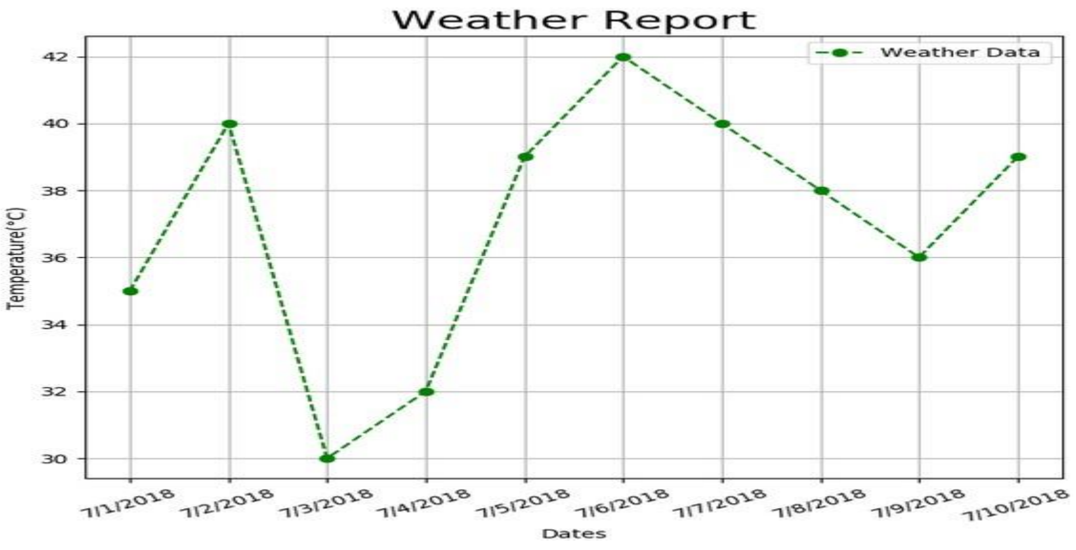
**Approach of the program:**

1. Import required libraries, matplotlib library for visualizing, and csv library for reading CSV data.
2. Open the file using open( ) function with ‘r’ mode (read-only) from CSV library and read the file using csv.reader( ) function.
3. Read each line in the file using for loop.
4. Append required columns of the CSV file into a list.
5. After reading the whole CSV file, plot the required data as X and Y axis.
6. In this Example, we are plotting Dates as X-axis and Temperature(°C ) as Y-axis.

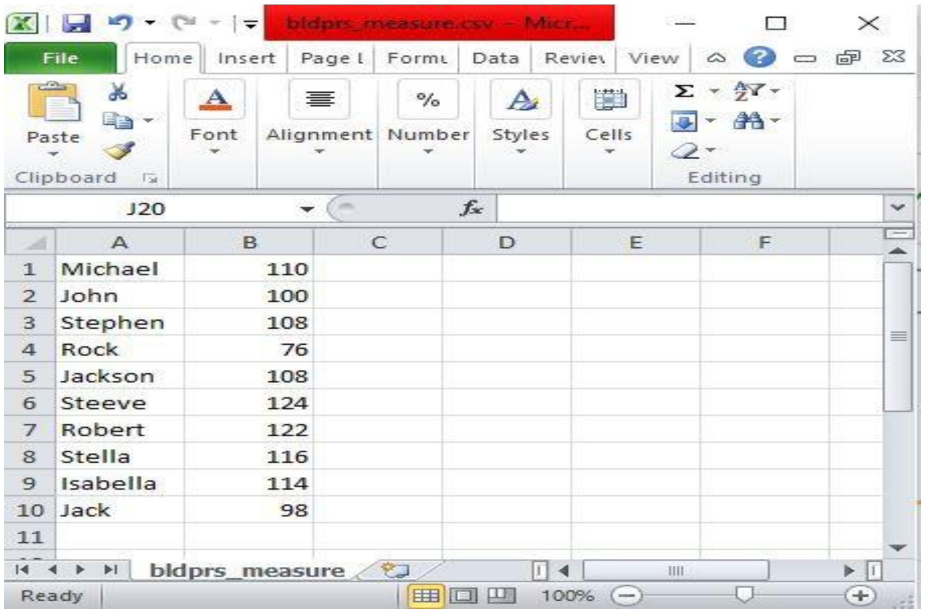
**Below is the implementation:**

```
import matplotlib.pyplot as plt
import csv

x = []
y = []

with open('Weatherdata.csv','r') as
csvfile:
    lines = csv.reader(csvfile,
delimiter=',')
    for row in lines:
        x.append(row[0])
        y.append(int(row[1]))

plt.plot(x, y, color = 'g', linestyle =
'dashed',
         marker = 'o',label = "Weather
Data")

plt.xticks(rotation = 25)
plt.xlabel('Dates')
plt.ylabel('Temperature(°C)')
plt.title('Weather Report', fontsize =
20)
plt.grid()
plt.legend()
plt.show()
```

**Output :**



**Example 3: Visualizing patients blood pressure report of a hospital through Scatter plot**



| | A | B |
|---|---|---|
| 1 | Michael | 110 |
| 2 | John | 100 |
| 3 | Stephen | 108 |
| 4 | Rock | 76 |
| 5 | Jackson | 108 |
| 6 | Steeve | 124 |
| 7 | Robert | 122 |
| 8 | Stella | 116 |
| 9 | Isabella | 114 |
| 10 | Jack | 98 |

**Approach of the program "Visualizing patients blood pressure report" through Scatter plot** :

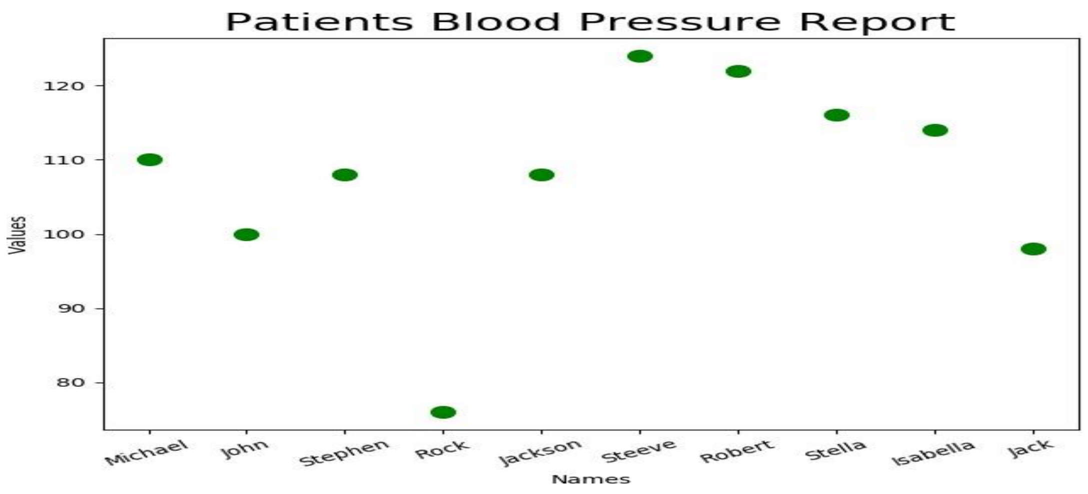
1. Import required libraries, matplotlib library for visualization and importing csv library for reading CSV data.
2. Open the file using open( ) function with 'r' mode (read-only) from CSV library and read the file using csv.reader( ) function.
3. Read each line in the file using for loop.

4. Append required columns of the CSV file into a list.
5. After reading the whole CSV file, plot the required data as X and Y axis.
6. In this example, we are plotting the Names of patients as X-axis and Blood pressure values as Y-axis.

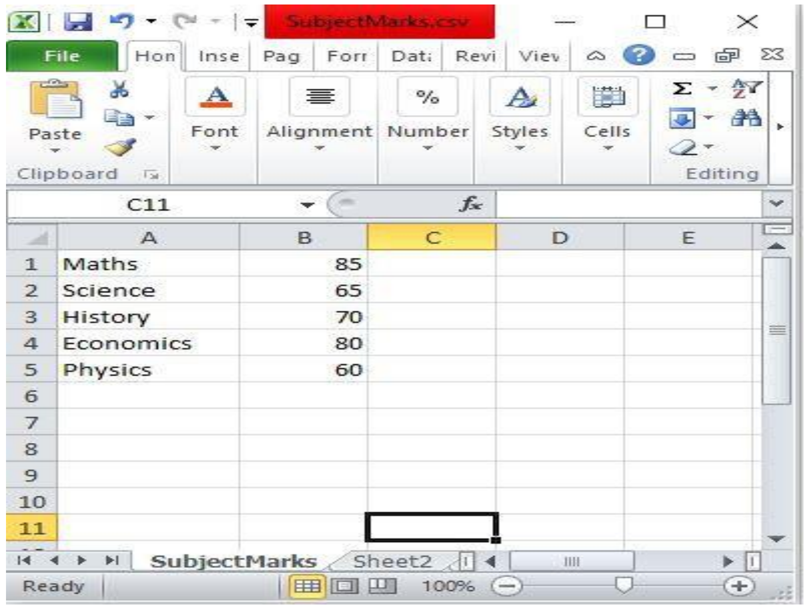
**Below is the implementation:**

```
import matplotlib.pyplot as plt
import csv

Names = []
Values = []

with open('bldprs_measure.csv','r') as csvfile:
    lines = csv.reader(csvfile, delimiter=',')
    for row in lines:
        Names.append(row[0])
        Values.append(int(row[1]))

plt.scatter(Names, Values, color = 'g',s = 100)
plt.xticks(rotation = 25)
plt.xlabel('Names')
plt.ylabel('Values')
plt.title('Patients Blood Pressure Report',
fontsize = 20)

plt.show()
```

**Output :**



**Example 4: Visualizing Student marks in different subjects using a pie plot**



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Maths | 85 | | | |
| 2 | Science | 65 | | | |
| 3 | History | 70 | | | |
| 4 | Economics | 80 | | | |
| 5 | Physics | 60 | | | |

**Approach of the program:**

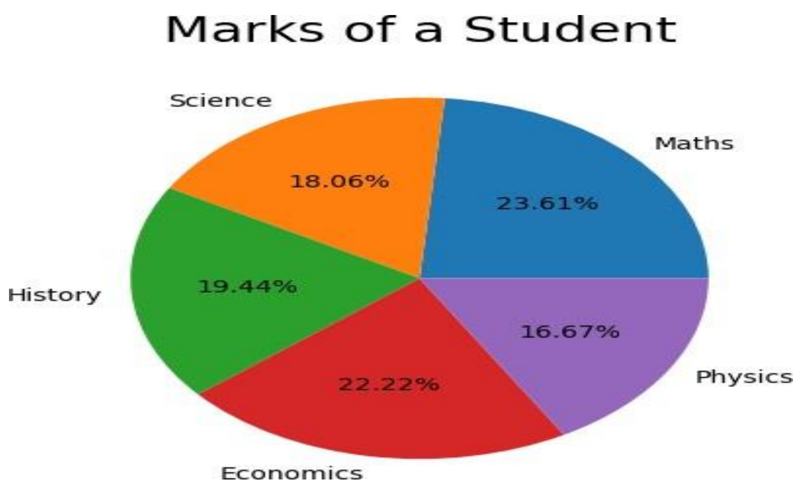
1. Import required libraries, matplotlib library for visualization and importing csv library for reading CSV data.
2. Open the file using open( )  function with ‘r’ mode (read-only) from CSV library and read the file using csv.reader( ) function.
3. Read each line in the file using for loop.
4. Append required columns of the CSV file into lists.
5. After reading the whole CSV data, plot the required data as pie plot

using plt.pie( ) function.

**Below is the implementation:**

```
import matplotlib.pyplot as plt
import csv

Subjects = []
Scores = []

with open('SubjectMarks.csv', 'r') as
csvfile:
    lines = csv.reader(csvfile, delimiter =
',')
    for row in lines:
        Subjects.append(row[0])
        Scores.append(int(row[1]))

plt.pie(Scores,labels = Subjects,autopct =
'%.2f%%')
plt.title('Marks of a Student', fontsize =
20)
plt.show()
```

**Output :**



**Important features of scikit-learn:**

- Simple and efficient tools for data mining and data analysis. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
- Accessible to everybody and reusable in various contexts.
- Built on the top of NumPy, SciPy, and matplotlib.
- Open source, commercially usable – BSD license.

In this article, we are going to see how we can easily build a machine learning model using scikit-learn.

**Installation:**

The latest version of Scikit-learn is 1.1 and it requires Python 3.8 or newer.

Scikit-learn requires:

- NumPy
- SciPy as its dependencies.

Before installing scikit-learn, ensure that you have NumPy and SciPy installed. Once you have a working installation of NumPy and SciPy, the easiest way to install scikit-learn is using pip:

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

Let us get started with the modeling process now.

**Step 1: Load a dataset**

A dataset is nothing but a collection of data. A dataset generally has two main components:

- **Features**: (also known as predictors, inputs, or attributes) they are simply the variables of our data. They can be more than one and hence represented by a **feature matrix** (‘X’ is a common notation to represent feature matrix). A list of all the feature names is termed **feature names**.
- **Response**: (also known as the target, label, or output) This is the output variable depending on the feature variables. We generally have a single response column and it is represented by a **response vector** (‘y’ is a common notation to represent response vector). All the possible values taken by a response vector are termed **target names**.

**Loading exemplar dataset:** scikit-learn comes loaded with a few example datasets like the iris and digits datasets for classification and the boston house prices dataset for regression.

Given below is an example of how one can load an exemplar dataset:

- Python

```
# load the iris dataset as an example
from sklearn.datasets import load_iris
iris = load_iris()

# store the feature matrix (X) and response
vector (y)
X  =  iris.data
y = iris.target

# store the feature and target names
feature_names = iris.feature_names
target_names = iris.target_names

# printing features and target names of our
dataset
print("Feature names:", feature_names)
print("Target names:", target_names)

# X and y are numpy arrays
print("\nType of X is:", type(X))

# printing first 5 input rows
print("\nFirst 5 rows of X:\n", X[:5])
```

Output:

```
Feature names: ['sepal length (cm)','sepal width (cm)',
                'petal length (cm)','petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']

Type of X is:

First 5 rows of X:
 [[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]]
```

**Loading external dataset:** Now, consider the case when we want to load an external dataset. For this purpose, we can use the **pandas library** for easily loading and manipulating datasets.

To install pandas, use the following pip command:

```
pip install pandas
```

In pandas, important data types are:

**Series**: Series is a one-dimensional labeled array capable of holding any data type.

**DataFrame**: It is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object.

- Python

```
import pandas as pd

# reading csv file
data = pd.read_csv('weather.csv')

# shape of dataset
print("Shape:", data.shape)

# column names
print("\nFeatures:", data.columns)

# storing the feature matrix (X) and response
vector (y)
X = data[data.columns[:-1]]
y = data[data.columns[-1]]

# printing first 5 rows of feature matrix
print("\nFeature matrix:\n", X.head())

# printing first 5 values of response vector
print("\nResponse vector:\n", y.head())
```

Output:

```
Shape: (14, 5)

Features: Index([u'Outlook', u'Temperature', u'Humidity',
               u'Windy', u'Play'], dtype='object')
```

```
Feature matrix:
     Outlook Temperature Humidity  Windy
0  overcast         hot     high  False
1  overcast        cool   normal   True
2  overcast        mild     high   True
3  overcast         hot   normal  False
4     rainy        mild     high  False

Response vector:
0    yes
1    yes
2    yes
3    yes
4    yes
Name: Play, dtype: object
```

### Step 2: Splitting the dataset

One important aspect of all machine learning models is to determine their accuracy. Now, 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.

But this method has several flaws in it, like:

- The goal is to estimate the likely performance of a model on **out-of-sample** data.
- Maximizing training accuracy rewards overly complex models that won't necessarily generalize our model.
- Unnecessarily complex models may over-fit the training data.

A better option is to split our data into two parts: the first one for training our machine learning model, and the second one for testing our model.

**To summarize:**

- Split the dataset into two pieces: a training set and a testing set.
- Train the model on the training set.
- Test the model on the testing set, and evaluate how well our model did.

**Advantages of train/test split:**

- The model can be trained and tested on different data than the one used for training.
- Response values are known for the test dataset, hence predictions can be evaluated
- Testing accuracy is a better estimate than training accuracy of out-of-sample performance.

Consider the example below:

- Python

```
# load the iris dataset as an example
from sklearn.datasets import load_iris
iris = load_iris()

# store the feature matrix (X) and response vector (y)
X  =  iris.data
y = iris.target

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.4, random_state=1)

# printing the shapes of the new X objects
print(X_train.shape)
print(X_test.shape)

# printing the shapes of the new y objects
print(y_train.shape)
print(y_test.shape)
```

Output:

```
(90L, 4L)
```

```
(60L, 4L)
(90L,)
(60L,)
```

The **train_test_split** function takes several arguments which are explained below:

- **X, y**: These are the feature matrix and response vector which need to be split.
- **test_size**: It is the ratio of test data to the given data. For example, setting test_size = 0.4 for 150 rows of X produces test data of 150 x 0.4 = 60 rows.
- **random_state**: If you use random_state = some_number, then you can guarantee that your split will be always the same. This is useful if you want reproducible results, for example in testing for consistency in the documentation (so that everybody can see the same numbers).

**Step 3: Training the model**

Now, it’s time to train some prediction models using our dataset. Scikit-learn provides a wide range of machine learning algorithms that have a unified/consistent interface for fitting, predicting accuracy, etc.

The example given below uses KNN (K nearest neighbors) classifier.

**Note**: We will not go into the details of how the algorithm works as we are interested in understanding its implementation only.

Now, consider the example below:

```
# load the iris dataset as an example
from sklearn.datasets import load_iris
iris = load_iris()

# store the feature matrix (X) and response vector (y)
X  =  iris.data
y = iris.target

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.4, random_state=1)

# training the model on training set
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# making predictions on the testing set
y_pred = knn.predict(X_test)

# comparing actual response values (y_test) with predicted
response values (y_pred)
from sklearn import metrics
print("kNN model accuracy:", metrics.accuracy_score(y_test,
y_pred))

# making prediction for out of sample data
sample = [[3, 5, 4, 2], [2, 3, 5, 4]]
preds = knn.predict(sample)
pred_species = [iris.target_names[p] for p in preds]
print("Predictions:", pred_species)

# saving the model
from sklearn.externals import joblib
joblib.dump(knn, 'iris_knn.pkl')
```

Output:

```
kNN model accuracy: 0.983333333333
Predictions: ['versicolor', 'virginica']
```

Important points to note from the above code:

- We create a knn classifier object using:

```
knn = KNeighborsClassifier(n_neighbors=3)
```

- The classifier is trained using X_train data. The process is termed **fitting**. We pass the feature matrix and the corresponding response vector.

```
knn.fit(X_train, y_train)
```

- Now, we need to test our classifier on the X_test data. **knn.predict** method is used for this purpose. It returns the predicted response vector, **y_pred**.

```
y_pred = knn.predict(X_test)
```

- Now, we are interested in finding the accuracy of our model by comparing **y_test** and **y_pred**. This is done using the metrics module's method **accuracy_score**:

```
print(metrics.accuracy_score(y_test, y_pred))
```

- Consider the case when you want your model to make predictions **out of sample** data. Then, the sample input can simply be passed in the same way as we pass any feature matrix.

```
sample = [[3, 5, 4, 2], [2, 3, 5, 4]]
preds = knn.predict(sample)
```

- If you are not interested in training your classifier again and again and using the pre-trained classifier, one can save their classifier using **joblib**. All you need to do is:

```
joblib.dump(knn, 'iris_knn.pkl')
```

- In case you want to load an already saved classifier, use the following method:

```
knn = joblib.load('iris_knn.pkl')
```

As we approach the end of this article, here are some benefits of using scikit-learn over some other machine learning libraries(like R libraries):

- **Consistent interface** to machine learning models
- Provides many **tuning parameters** but with **sensible defaults**
- Exceptional **documentation**
- Rich set of functionality for **companion tasks**.
- **Active community** for development and support.

**Task A:**

1. Read the data provided
2. Plot a bar graph
3. Plot a line graph
4. Plot a pie chart
5. Scatter Plot

**Task B:**

1. Write a csv file, having the following columns:
   Name of student
   Age of Student
   Height of student in inches
   Gpa of student
2. Plot bar and line graph respectively for age, height and gpa.

**Task C:**

1. Train knn model on given data set