JB INSTITUTE OF ENGINEERING & TECHNOLOGY

UGC Autonomous

Accredited by NAAC & NBA, Approved by AICTE & Permanently Affiliated to JNTUH

MACHINE LEARNING LAB OBSERVATION

B.Tech AIML - R20 Regulation

Department of Artificial Intelligence & Machine Learning

Name of the Student	
Roll Number	
Class & Section	

S.No	Name of the Experiment	Date of Completion	Faculty Signature
1	Familiarizing with Anaconda and Jupyter for importing modules and dependencies for ML		
2	Familiarization with NumPy, Panda and Matplotlib by Loading Dataset in Python		
3	Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.		
4	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.		
5	Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.		
6	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.		
7	Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.		
8	Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in python classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.		
9	Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Python ML library classes/API.		
10	Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Python ML library classes/API in the program.		
11	Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Python ML library classes can be used for this problem.		
12	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.		
13	Carry out the performance analysis of classification algorithms on a specific dataset.		
14	Case Study on ML Problem		

Machine Learning Lab Syllabus

Pre-Requisites:

1. Data Structures
2. Design and Analysis of Algorithms
3. Python Programming
4. Mathematics for Machine Learning

Course objectives:

The student will:

1. Understand the usage of .csv files for organising data in the form of datasets.
2. Design and analyze the performance of various machine learning algorithms.
3. Identify the real-world problems that can be solved by applying machine learning algorithms.
4. Identify suitable machine learning algorithms for solving real world problems.
5. Understand the limitations of machine learning algorithms.

Lab Experiments:

1. Familiarizing with Anaconda and Jupyter for importing modules and dependencies for ML

2. Familiarization with NumPy, Panda and Matplotlib by Loading Dataset in Python

3. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

4. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

5. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

6. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

7. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

8. Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in python classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

9. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Python ML library classes/API.

10. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Python ML library classes/API in the program.

11. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Python ML library classes can be used for this problem.

12. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

13. Carry out the performance analysis of classification algorithms on a specific dataset.

14. **Case Study:** You are owing a supermarket mall and through membership cards, you have some basic data about your customers like Customer ID, age, gender, annual income, and spending score. Spending Score is something you assign to the customer based on your defined parameters like customer behaviour and purchasing data.

Problem Statement

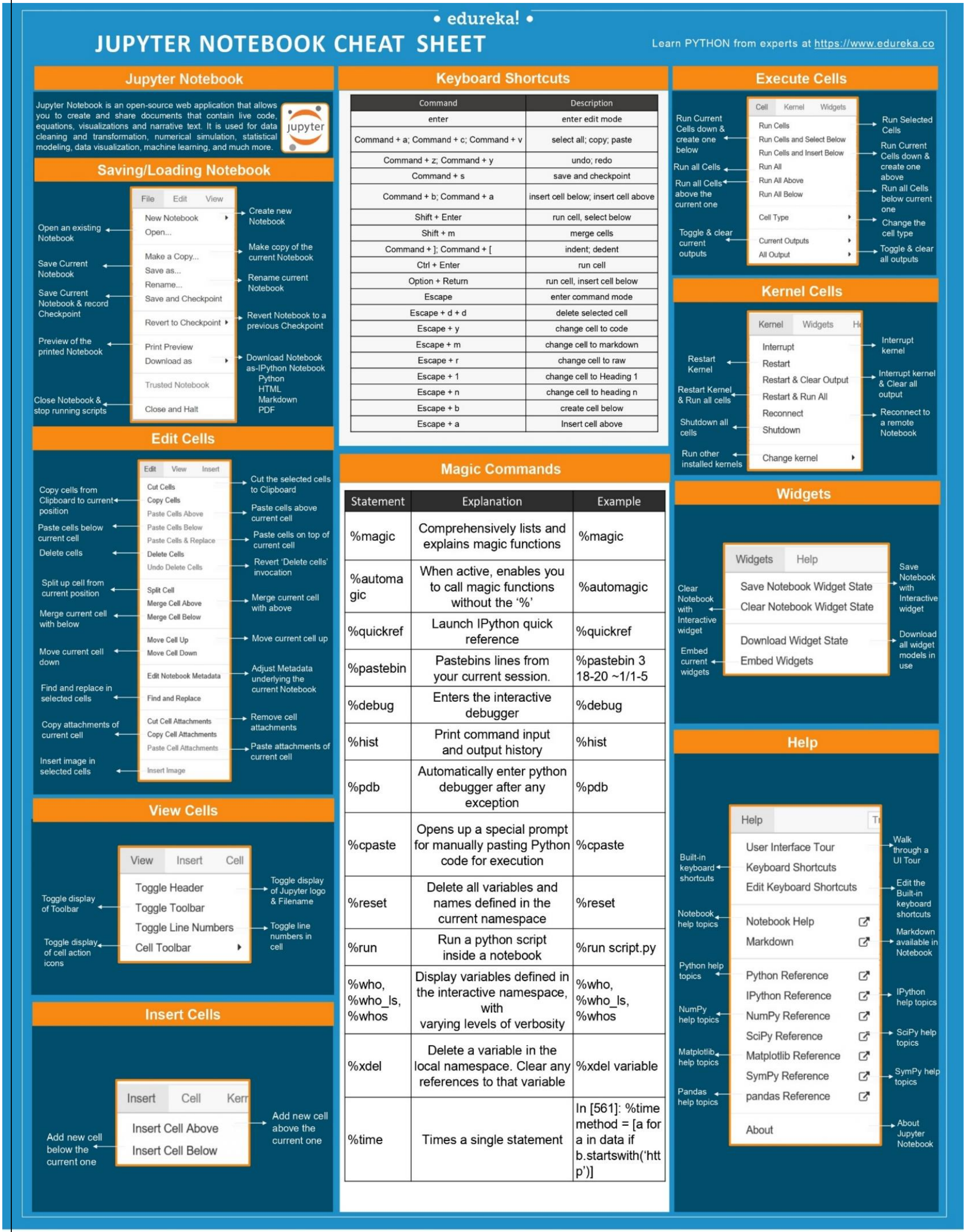
By being the managing director of your Supermarket Mall, You wanted to understand the customers like who can be easily converge [Target Customers] so that the sense can be given to marketing team and plan the strategy accordingly.
After carrying out this case study, answer the questions given below.
1- How to achieve customer segmentation using machine learning algorithm (KMeans Clustering) in Python in simplest way.
2- Who are your target customers with whom you can start marketing strategy [easy to converse]
3- How the marketing strategy works in real world?

Course Outcomes:

The student will be able to:

1. Create .csv files for organising data in the form of datasets.
2. Implement and compare the performance metrics of various machine learning algorithms.
3. Translate the real-world problems into a well posed learning problem that can be solved by a suitable machine learning algorithm.
4. Decide suitable machine learning algorithms for solving real world problems.
5. Tackle the limitations of machine learning algorithms.

LAB EXPERIMENTS



• edureka! •

JUPYTER NOTEBOOK CHEAT SHEET

Learn PYTHON from experts at https://www.edureka.co

Jupyter Notebook

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. It is used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Saving/Loading Notebook

File
Edit
View
New Notebook → Create new Notebook
Open... → Open an existing Notebook
Make a Copy... → Make copy of the current Notebook
Save as... → Save Current Notebook
Rename... → Rename current Notebook
Save and Checkpoint → Save Current Notebook & record Checkpoint
Revert to Checkpoint → Revert Notebook to a previous Checkpoint
Print Preview → Preview of the printed Notebook
Download as → Download Notebook as-IPython Notebook Python HTML Markdown PDF
Trusted Notebook
Close and Halt → Close Notebook & stop running scripts

Edit Cells

Edit
View
Insert
Cut Cells → Cut the selected cells to Clipboard
Copy Cells → Copy cells from Clipboard to current position
Paste Cells Above → Paste cells above current cell
Paste Cells Below → Paste cells below current cell
Paste Cells & Replace → Paste cells on top of current cell
Delete Cells → Delete cells
Undo Delete Cells → Revert 'Delete cells' invocation
Split Cell → Split up cell from current position
Merge Cell Above → Merge current cell with above
Merge Cell Below → Merge current cell with below
Move Cell Up → Move current cell up
Move Cell Down → Move current cell down
Edit Notebook Metadata → Adjust Metadata underlying the current Notebook
Find and Replace → Find and replace in selected cells
Cut Cell Attachments → Remove cell attachments
Copy Cell Attachments → Copy attachments of current cell
Paste Cell Attachments → Paste attachments of current cell
Insert Image → Insert image in selected cells

View Cells

View
Insert
Cell
Toggle Header → Toggle display of Jupyter logo & Filename
Toggle Toolbar → Toggle display of Toolbar
Toggle Line Numbers → Toggle line numbers in cell
Cell Toolbar → Toggle display of cell action icons

Insert Cells

Insert
Cell
Kern
Insert Cell Above → Add new cell above the current one
Insert Cell Below → Add new cell below the current one

Keyboard Shortcuts

Command	Description
enter	enter edit mode
Command + a; Command + c; Command + v	select all; copy; paste
Command + z; Command + y	undo; redo
Command + s	save and checkpoint
Command + b; Command + a	insert cell below; insert cell above
Shift + Enter	run cell, select below
Shift + m	merge cells
Command +]; Command + [	indent; dedent
Ctrl + Enter	run cell
Option + Return	run cell, insert cell below
Escape	enter command mode
Escape + d + d	delete selected cell
Escape + y	change cell to code
Escape + m	change cell to markdown
Escape + r	change cell to raw
Escape + 1	change cell to Heading 1
Escape + n	change cell to heading n
Escape + b	create cell below
Escape + a	Insert cell above

Magic Commands

Statement	Explanation	Example
%magic	Comprehensively lists and explains magic functions	%magic
%automagic	When active, enables you to call magic functions without the '%'	%automagic
%quickref	Launch IPython quick reference	%quickref
%pastebin	Pastebins lines from your current session.	%pastebin 3 18-20 ~1/1-5
%debug	Enters the interactive debugger	%debug
%hist	Print command input and output history	%hist
%pdb	Automatically enter python debugger after any exception	%pdb
%cpaste	Opens up a special prompt for manually pasting Python code for execution	%cpaste
%reset	Delete all variables and names defined in the current namespace	%reset
%run	Run a python script inside a notebook	%run script.py
%who, %who_ls, %whos	Display variables defined in the interactive namespace, with varying levels of verbosity	%who, %who_ls, %whos
%xdel	Delete a variable in the local namespace. Clear any references to that variable	%xdel variable
%time	Times a single statement	In [561]: %time method = [a for a in data if b.startswith('http')]

Execute Cells

Cell
Kernel
Widgets
Run Cells → Run Selected Cells
Run Cells and Select Below → Run Current Cells down & create one below
Run Cells and Insert Below → Run Current Cells down & create one above
Run All → Run all Cells
Run All Above → Run all Cells above the current one
Run All Below → Run all Cells below current one
Cell Type → Change the cell type
Current Outputs → Toggle & clear current outputs
All Output → Toggle & clear all outputs

Kernel Cells

Kernel
Widgets
Interrupt → Interrupt kernel
Restart → Restart Kernel
Restart & Clear Output → Interrupt kernel & Clear all output
Restart & Run All → Restart Kernel & Run all cells
Reconnect → Reconnect to a remote Notebook
Shutdown → Shutdown all cells
Change kernel → Run other installed kernels

Widgets

Widgets
Help
Save Notebook Widget State → Save Notebook with Interactive widget
Clear Notebook Widget State → Clear Notebook with Interactive widget
Download Widget State → Download all widget models in use
Embed Widgets → Embed current widgets

Help

Help
User Interface Tour → Walk through a UI Tour
Keyboard Shortcuts → Built-in keyboard shortcuts
Edit Keyboard Shortcuts → Edit the Built-in keyboard shortcuts
Notebook Help → Notebook help topics
Markdown → Markdown available in Notebook
Python Reference → Python help topics
IPython Reference → IPython help topics
NumPy Reference → NumPy help topics
SciPy Reference → SciPy help topics
Matplotlib Reference → Matplotlib help topics
SymPy Reference → SymPy help topics
pandas Reference → Pandas help topics
About → About Jupyter Notebook

1. Familiarizing with Anaconda and Jupyter for importing modules and dependencies for ML &
2. Familiarization with NumPy, Panda and Matplotlib by Loading Dataset in Python

What is Jupyter Notebook?
The Jupyter Notebook is an incredibly powerful tool for interactively developing and presenting data science projects. This article will walk you through how to use Jupyter Notebooks for data science projects and how to set it up on your local machine.

First, though: **what is a "notebook"?**

A notebook integrates code and its output into a single document that combines visualizations, narrative text, mathematical equations, and other rich media. In other words: it's a single document where you can run code, display the output, and also add explanations, formulas, charts, and make your work more transparent, understandable, repeatable, and shareable.

Using Notebooks is now a major part of the data science workflow at companies across the globe. If your goal is to work with data, using a Notebook will speed up your workflow and make it easier to communicate and share your results.

Best of all, as part of the open source Project Jupyter, Jupyter Notebooks are completely free. You can download the software on its own, or as part of the Anaconda data science toolkit.

Although it is possible to use many different programming languages in Jupyter Notebooks, this article will focus on Python, as it is the most common use case. (Among R users, R Studio tends to be a more popular choice).

How to Follow This Tutorial
To get the most out of this tutorial you should be familiar with programming — Python and pandas specifically. That said, if you have experience with another language, the Python in this article shouldn't be too cryptic, and will still help you get Jupyter Notebooks set up locally.

Jupyter Notebooks can also act as a flexible platform for getting to grips with pandas and even Python, as will become apparent in this tutorial.

We will:

- Cover the basics of installing Jupyter and creating your first notebook
- Delve deeper and learn all the important terminology
- Explore how easily notebooks can be shared and published online.

(In fact, this article was written as a Jupyter Notebook! It's published here in read-only form, but this is a good example of how versatile notebooks can be. In fact, most of our programming tutorials and even our Python courses were created using Jupyter Notebooks).

Example Data Analysis in a Jupyter Notebook
First, we will walk through setup and a sample analysis to answer a real-life question. This will demonstrate how the flow of a notebook makes data science tasks more intuitive for us as we work, and for others once it's time to share our work.

So, let's say you're a data analyst and you've been tasked with finding out how the profits of the largest companies in the US changed historically. You find a data set of Fortune 500 companies spanning over 50 years since the list's first publication in 1955, put together from Fortune's public archive. We've gone ahead and created a CSV of the data you can use here.

As we shall demonstrate, Jupyter Notebooks are perfectly suited for this investigation. First, let's go ahead and install Jupyter.

Installation

The easiest way for a beginner to get started with Jupyter Notebooks is by installing Anaconda.

Anaconda is the most widely used Python distribution for data science and comes pre-loaded with all the most popular libraries and tools.

Some of the biggest Python libraries included in Anaconda include NumPy, pandas, and Matplotlib, though the full 1000+ list is exhaustive.

Anaconda thus lets us hit the ground running with a fully stocked data science workshop without the hassle of managing countless installations or worrying about dependencies and OS-specific (read: Windows-specific) installation issues.

To get Anaconda, simply:

1. Download the latest version of Anaconda for Python 3.8.
2. Install Anaconda by following the instructions on the download page and/or in the executable.

If you are a more advanced user with Python already installed and prefer to manage your packages manually, you can just use pip:

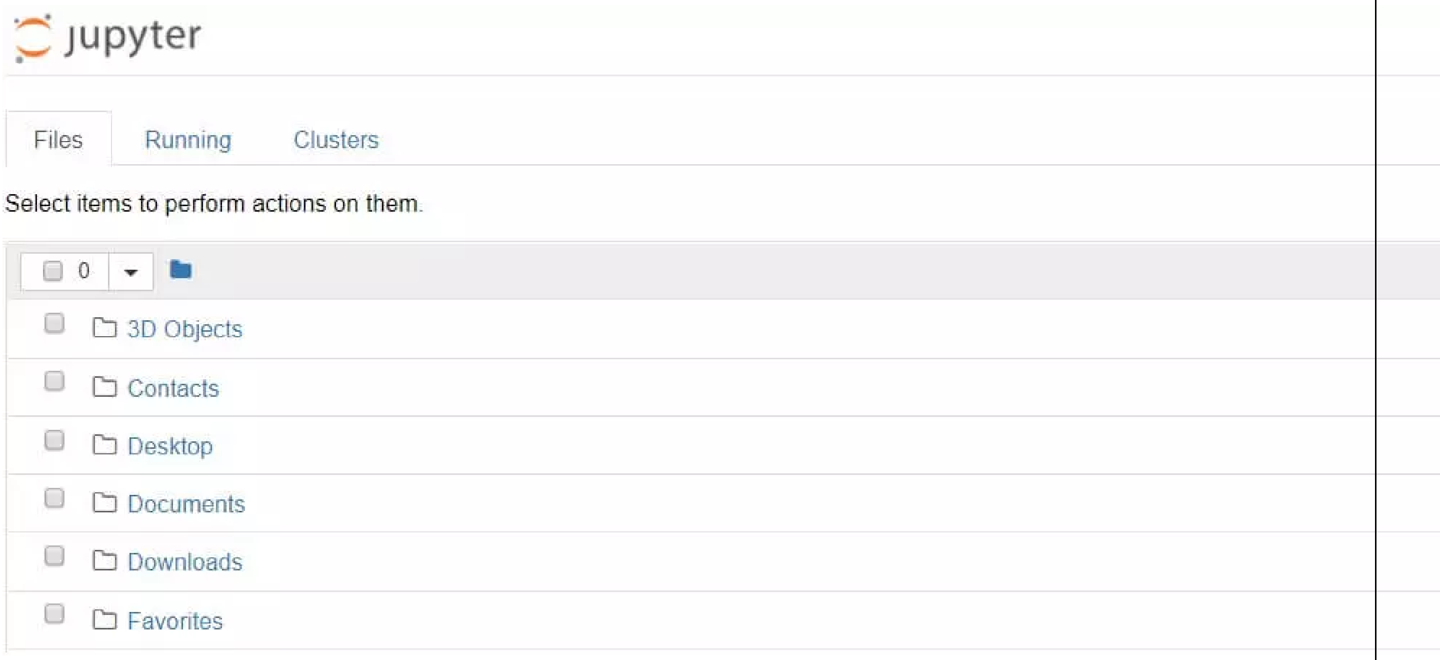
```
pip3 install jupyter
```

Creating Your First Notebook

In this section, we're going to learn to run and save notebooks, familiarize ourselves with their structure, and understand the interface. We'll become intimate with some core terminology that will steer you towards a practical understanding of how to use Jupyter Notebooks by yourself and set us up for the next section, which walks through an example data analysis and brings everything we learn here to life.

Running Jupyter

On Windows, you can run Jupyter via the shortcut Anaconda adds to your start menu, which will open a new tab in your default web browser that should look something like the following screenshot.



Select items to perform actions on them.

3D Objects
Contacts
Desktop
Documents
Downloads
Favorites

This isn't a notebook just yet, but don't panic! There's not much to it. This is the Notebook Dashboard, specifically designed for managing your Jupyter Notebooks. Think of it as the launchpad for exploring, editing and creating your notebooks.

Be aware that the dashboard will give you access only to the files and sub-folders contained within Jupyter's start-up directory (i.e., where Jupyter or Anaconda is installed). However, the start-up directory can be changed.

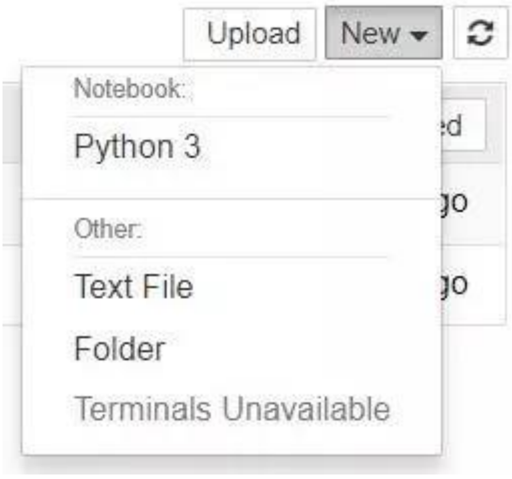
It is also possible to start the dashboard on any system via the command prompt (or terminal on Unix systems) by entering the command jupyter notebook; in this case, the current working directory will be the start-up directory.

With Jupyter Notebook open in your browser, you may have noticed that the URL for the dashboard is something like https://localhost:8888/tree. Localhost is not a website, but indicates that the content is being served from your *local* machine: your own computer.

Jupyter's Notebooks and dashboard are web apps, and Jupyter starts up a local Python server to serve these apps to your web browser, making it essentially platform-independent and opening the door to easier sharing on the web.

(If you don't understand this yet, don't worry — the important point is just that although Jupyter Notebooks opens in your browser, it's being hosted and run on your local machine. Your notebooks aren't actually on the web until you decide to share them.)

The dashboard's interface is mostly self-explanatory — though we will come back to it briefly later. So what are we waiting for? Browse to the folder in which you would like to create your first notebook, click the "New" drop-down button in the top-right and select "Python 3":



Hey presto, here we are! Your first Jupyter Notebook will open in new tab — each notebook uses its own tab because you can open multiple notebooks simultaneously.

If you switch back to the dashboard, you will see the new file Untitled.ipynb and you should see some green text that tells you your notebook is running.

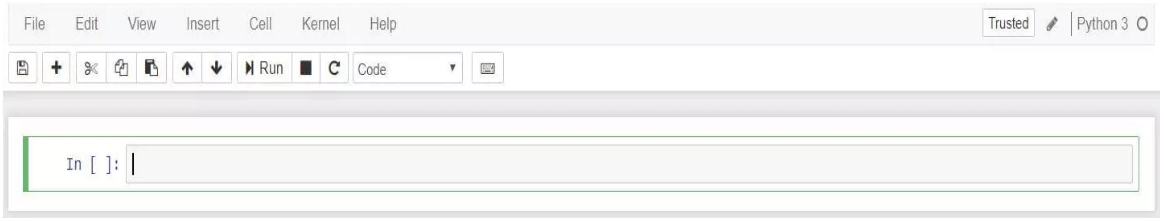
What is an ipynb File?
The short answer: each .ipynb file is one notebook, so each time you create a new notebook, a new .ipynb file will be created.

The longer answer: Each .ipynb file is a text file that describes the contents of your notebook in a format called JSON. Each cell and its contents, including image attachments that have been converted into strings of text, is listed therein along with some metadata.

You can edit this yourself — if you know what you are doing! — by selecting "Edit > Edit Notebook Metadata" from the menu bar in the notebook. You can also view the contents of your notebook files by selecting "Edit" from the controls on the dashboard

However, the key word there is *can*. In most cases, there's no reason you should ever need to edit your notebook metadata manually.

The Notebook Interface
Now that you have an open notebook in front of you, its interface will hopefully not look entirely alien. After all, Jupyter is essentially just an advanced word processor.



Why not take a look around? Check out the menus to get a feel for it, especially take a few moments to scroll down the list of commands in the command palette, which is the small button with the keyboard icon (or Ctrl + Shift + P). There are two fairly prominent terms that you should notice, which are probably new to you: *cells* and *kernels* are key both to understanding Jupyter and to what makes it more than just a word processor. Fortunately, these concepts are not difficult to understand.

- A **kernel** is a "computational engine" that executes the code contained in a notebook document.
- A **cell** is a container for text to be displayed in the notebook or code to be executed by the notebook's kernel.

Cells
We'll return to kernels a little later, but first let's come to grips with cells. Cells form the body of a notebook. In the screenshot of a new notebook in the section above, that box with the green outline is an empty cell. There are two main cell types that we will cover:

- A **code cell** contains code to be executed in the kernel. When the code is run, the notebook displays the output below the code cell that generated it.
- A **Markdown cell** contains text formatted using Markdown and displays its output in-place when the Markdown cell is run.

The first cell in a new notebook is always a code cell.

Let's test it out with a classic hello world example: Type print('Hello World!') into the cell and click the run button in the toolbar above or press Ctrl + Enter.

The result should look like this:

```
print('Hello World!')
```

```
Hello World!
```

When we run the cell, its output is displayed below and the label to its left will have changed from In [ ] to In [1].

The output of a code cell also forms part of the document, which is why you can see it in this article. You can always tell the difference between code and Markdown cells because code cells have that label on the left and Markdown cells do not.

The "In" part of the label is simply short for "Input," while the label number indicates *when* the cell was executed on the kernel — in this case the cell was executed first.

Run the cell again and the label will change to In [2] because now the cell was the second to be run on the kernel. It will become clearer why this is so useful later on when we take a closer look at kernels.

From the menu bar, click *Insert* and select *Insert Cell Below* to create a new code cell underneath your first and try out the following code to see what happens. Do you notice anything different?

```
import time

time.sleep(3)
```

This cell doesn't produce any output, but it does take three seconds to execute. Notice how Jupyter signifies when the cell is currently running by changing its label to In [*].

In general, the output of a cell comes from any text data specifically printed during the cell's execution, as well as the value of the last line in the cell, be it a lone variable, a function call, or something else. For example:

```
def say_hello(recipient):
    return 'Hello, {}!'.format(recipient)


say_hello('Tim')
```

```
'Hello, Tim!'
```

You'll find yourself using this almost constantly in your own projects, and we'll see more of it later on.

Keyboard Shortcuts

One final thing you may have observed when running your cells is that their border turns blue, whereas it was green while you were editing. In a Jupyter Notebook, there is always one "active" cell highlighted with a border whose color denotes its current mode:

- **Green outline** — cell is in "edit mode"
- **Blue outline** — cell is in "command mode"

So what can we do to a cell when it's in command mode? So far, we have seen how to run a cell with Ctrl + Enter, but there are plenty of other commands we can use. The best way to use them is with keyboard shortcuts

Keyboard shortcuts are a very popular aspect of the Jupyter environment because they facilitate a speedy cell-based workflow. Many of these are actions you can carry out on the active cell when it's in command mode.

Below, you'll find a list of some of Jupyter's keyboard shortcuts. You don't need to memorize them all immediately, but this list should give you a good idea of what's possible.

- Toggle between edit and command mode with Esc and Enter, respectively.
- Once in command mode:
  - Scroll up and down your cells with your Up and Down keys.
  - Press A or B to insert a new cell above or below the active cell.
  - M will transform the active cell to a Markdown cell.
  - Y will set the active cell to a code cell.
  - D + D (D twice) will delete the active cell.
  - Z will undo cell deletion.
  - Hold Shift and press Up or Down to select multiple cells at once. With multiple cells selected, Shift + M will merge your selection.
- Ctrl + Shift + -, in edit mode, will split the active cell at the cursor.
- You can also click and Shift + Click in the margin to the left of your cells to select them.

Go ahead and try these out in your own notebook. Once you're ready, create a new Markdown cell and we'll learn how to format the text in our notebooks.

Markdown

Markdown is a lightweight, easy to learn markup language for formatting plain text. Its syntax has a one-to-one correspondence with HTML tags, so some prior knowledge here would be helpful but is definitely not a prerequisite.

Remember that this article was written in a Jupyter notebook, so all of the narrative text and images you have seen so far were achieved writing in Markdown. Let’s cover the basics with a quick example:

~~~~
# This is a level 1 heading

## This is a level 2 heading

This is some plain text that forms a paragraph. Add emphasis via **bold** and __bold__, or *italic* and _italic_.

Paragraphs must be separated by an empty line.

* Sometimes we want to include lists.
* Which can be bulleted using asterisks.

1. Lists can also be numbered.
2. If we want an ordered list.

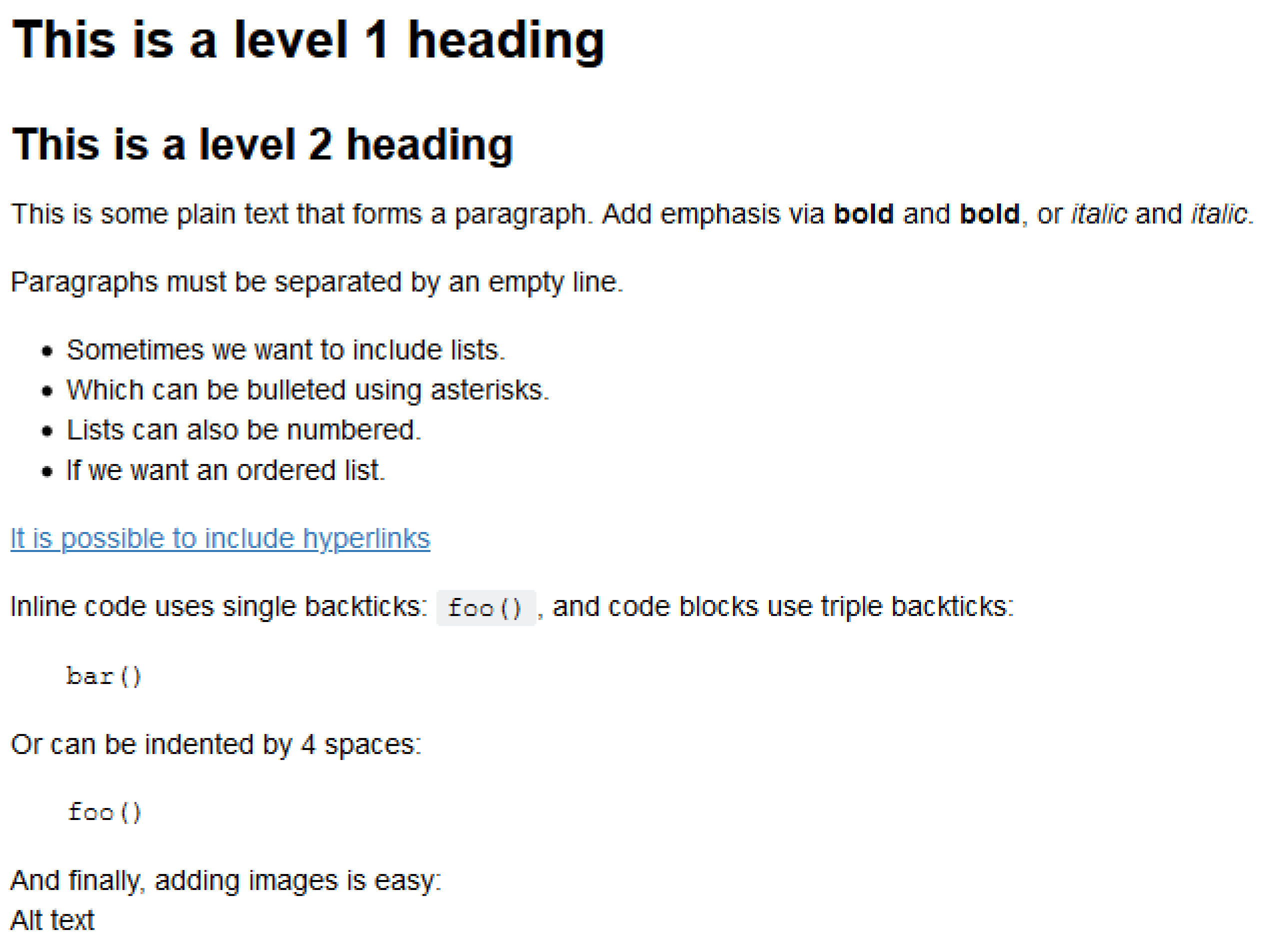
[It is possible to include hyperlinks](https://www.example.com)

Inline code uses single backticks: foo(), and code blocks use triple backticks:
```
bar()
```
Or can be indented by 4 spaces:

    foo()

And finally, adding images is easy: ![Alt text](https://www.example.com/image.jpg)
~~~~

Here’s how that Markdown would look once you run the cell to render it:



This is a level 1 heading

This is a level 2 heading

This is some plain text that forms a paragraph. Add emphasis via **bold** and **bold**, or *italic* and *italic*.

Paragraphs must be separated by an empty line.

- Sometimes we want to include lists.
- Which can be bulleted using asterisks.
- Lists can also be numbered.
- If we want an ordered list.

It is possible to include hyperlinks

Inline code uses single backticks: `foo()`, and code blocks use triple backticks:

```
bar()
```

Or can be indented by 4 spaces:

```
foo()
```

And finally, adding images is easy:
Alt text

(Note that the alt text for the image is displayed here because we didn't actually use a valid image URL in our example)

When attaching images, you have three options:

- Use a URL to an image on the web.
- Use a local URL to an image that you will be keeping alongside your notebook, such as in the same git repo.
- Add an attachment via "Edit > Insert Image"; this will convert the image into a string and store it inside your notebook .ipynb file. Note that this will make your .ipynb file much larger!

There is plenty more to Markdown, especially around hyperlinking, and it's also possible to simply include plain HTML. Once you find yourself pushing the limits of the basics above, you can refer to the official guide from Markdown's creator, John Gruber, on his website.

Kernels
Behind every notebook runs a kernel. When you run a code cell, that code is executed within the kernel. Any output is returned back to the cell to be displayed. The kernel's state persists over time and between cells — it pertains to the document as a whole and not individual cells.

For example, if you import libraries or declare variables in one cell, they will be available in another. Let's try this out to get a feel for it. First, we'll import a Python package and define a function:

```
import numpy as np
```

```
def square(x):
    return x * x
```

Once we’ve executed the cell above, we can reference np and square in any other cell.

```
x = np.random.randint(1, 10)
y = square(x)
print('%d squared is %d' % (x, y))
```

```
1 squared is 1
```

This will work regardless of the order of the cells in your notebook. As long as a cell has been run, any variables you declared or libraries you imported will be available in other cells.

You can try it yourself, let’s print out our variables again.

```
print('Is %d squared %d?' % (x, y))
```

```
Is 1 squared 1?
```

No surprises here! But what happens if we change the value of y?

```
y = 10
print('Is %d squared is %d?' % (x, y))
```

If we run the cell above, what do you think would happen?

We will get an output like: Is 4 squared 10?. This is because once we’ve run the y = 10 code cell, y is no longer equal to the square of x in the kernel.

Most of the time when you create a notebook, the flow will be top-to-bottom. But it’s common to go back to make changes. When we do need to make changes to an earlier cell, the order of execution we can see on the left of each cell, such as In [6], can help us diagnose problems by seeing what order the cells have run in.

And if we ever wish to reset things, there are several incredibly useful options from the Kernel menu:

- Restart: restarts the kernel, thus clearing all the variables etc that were defined.
- Restart & Clear Output: same as above but will also wipe the output displayed below your code cells.
- Restart & Run All: same as above but will also run all your cells in order from first to last.

If your kernel is ever stuck on a computation and you wish to stop it, you can choose the Interrupt option.

Choosing a Kernel

You may have noticed that Jupyter gives you the option to change kernel, and in fact there are many different options to choose from. Back when you created a new notebook from the dashboard by selecting a Python version, you were actually choosing which kernel to use.

There kernels for different versions of Python, and also for over 100 languages including Java, C, and even Fortran. Data scientists may be particularly interested in the kernels for R and Julia, as well as both imatlab and the Calysto MATLAB Kernel for Matlab.

The SoS kernel provides multi-language support within a single notebook.

Each kernel has its own installation instructions, but will likely require you to run some commands on your computer.

Example Analysis

Now we've looked at *what* a Jupyter Notebook is, it's time to look at *how* they're used in practice, which should give us clearer understanding of *why* they are so popular.

It's finally time to get started with that Fortune 500 data set mentioned earlier. Remember, our goal is to find out **how the profits of the largest companies in the US changed historically**.

It's worth noting that everyone will develop their own preferences and style, but the general principles still apply. You can follow along with this section in your own notebook if you wish, or use this as a guide to creating your own approach.

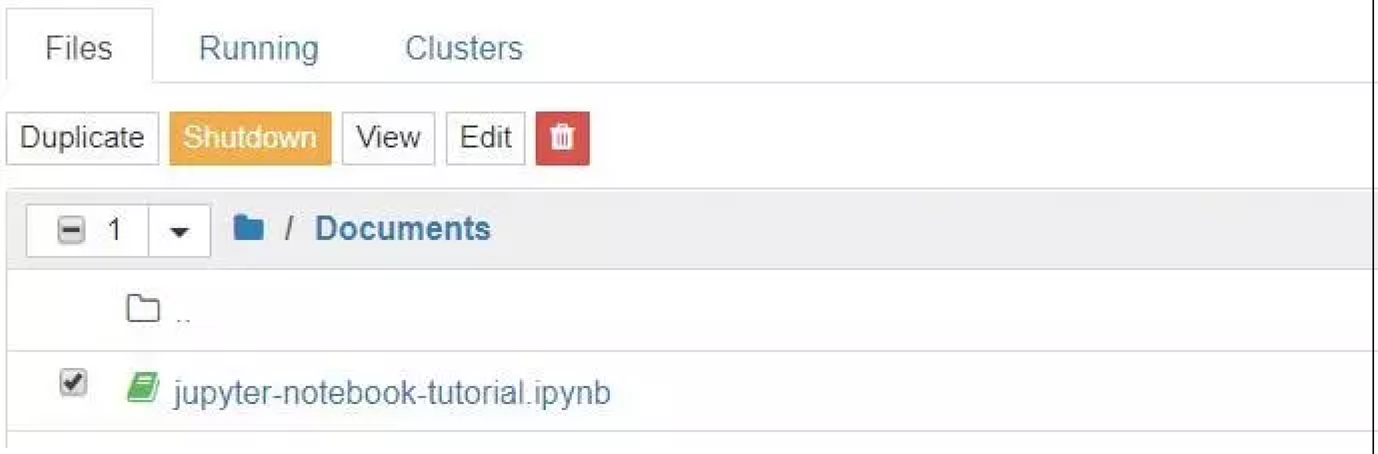
Naming Your Notebooks

Before you start writing your project, you'll probably want to give it a meaningful name. file name Untitled in the upper left of the screen to enter a new file name, and hit the Save icon (which looks like a floppy disk) below it to save.

Note that closing the notebook tab in your browser will **not** "close" your notebook in the way closing a document in a traditional application will. The notebook's kernel will continue to run in the background and needs to be shut down before it is truly "closed" — though this is pretty handy if you accidentally close your tab or browser!

If the kernel is shut down, you can close the tab without worrying about whether it is still running or not.

The easiest way to do this is to select "File > Close and Halt" from the notebook menu. However, you can also shutdown the kernel either by going to "Kernel > Shutdown" from within the notebook app or by selecting the notebook in the dashboard and clicking "Shutdown" (see image below).



Setup

It's common to start off with a code cell specifically for imports and setup, so that if you choose to add or change anything, you can simply edit and re-run the cell without causing any side-effects.

```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns sns.set(style="darkgrid")
```

We'll import pandas to work with our data, Matplotlib to plot charts, and Seaborn to make our charts prettier. It's also common to import NumPy but in this case, pandas imports it for us.

That first line isn't a Python command, but uses something called a line magic to instruct Jupyter to capture Matplotlib plots and render them in the cell output. We'll talk a bit more about line magics later, and they're also covered in our advanced Jupyter Notebooks tutorial.

For now, let's go ahead and load our data.

```
df = pd.read_csv('fortune500.csv')
```

It's sensible to also do this in a single cell, in case we need to reload it at any point.

Save and Checkpoint

Now we've got started, it's best practice to save regularly. Pressing Ctrl + S will save our notebook by calling the "Save and Checkpoint" command, but what is this checkpoint thing?

Every time we create a new notebook, a checkpoint file is created along with the notebook file. It is located within a hidden subdirectory of your save location called .ipynb_checkpoints and is also a .ipynb file.

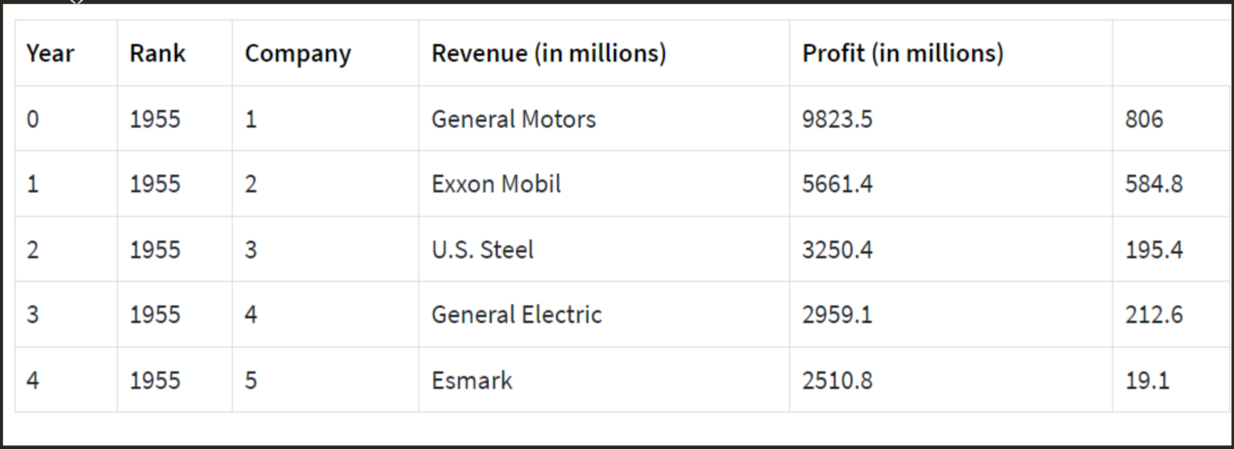
By default, Jupyter will autosave your notebook every 120 seconds to this checkpoint file without altering your primary notebook file. When you "Save and Checkpoint," both the notebook and checkpoint files are updated. Hence, the checkpoint enables you to recover your unsaved work in the event of an unexpected issue.

You can revert to the checkpoint from the menu via "File > Revert to Checkpoint."

Investigating Our Data Set

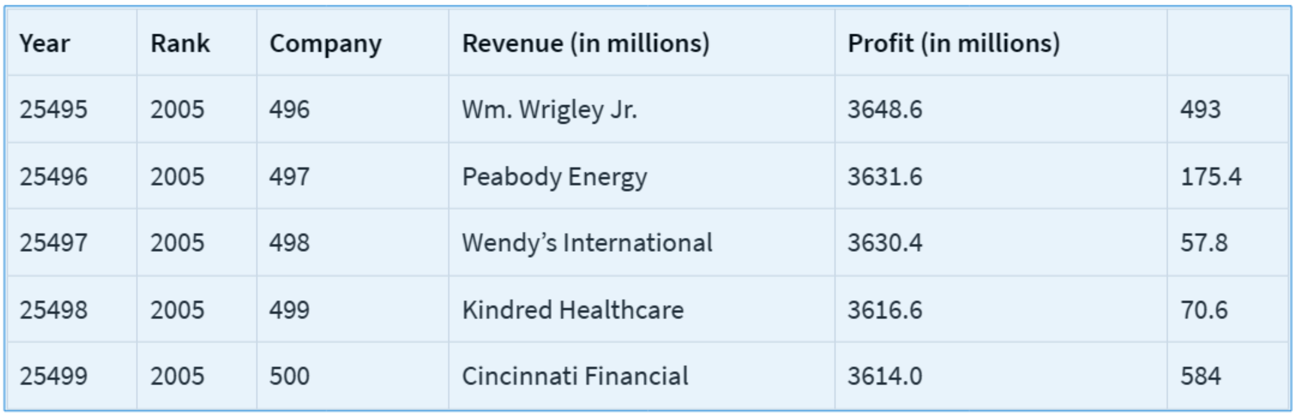
Now we're really rolling! Our notebook is safely saved and we've loaded our data set df into the most-used pandas data structure, which is called a DataFrame and basically looks like a table. What does ours look like?

```
df.head()
```



Year	Rank	Company	Revenue (in millions)	Profit (in millions)	
0	1955	1	General Motors	9823.5	806
1	1955	2	Exxon Mobil	5661.4	584.8
2	1955	3	U.S. Steel	3250.4	195.4
3	1955	4	General Electric	2959.1	212.6
4	1955	5	Esmark	2510.8	19.1

```
df.tail()
```



Year	Rank	Company	Revenue (in millions)	Profit (in millions)	
25495	2005	496	Wm. Wrigley Jr.	3648.6	493
25496	2005	497	Peabody Energy	3631.6	175.4
25497	2005	498	Wendy’s International	3630.4	57.8
25498	2005	499	Kindred Healthcare	3616.6	70.6
25499	2005	500	Cincinnati Financial	3614.0	584

Looking good. We have the columns we need, and each row corresponds to a single company in a single year.

Let’s just rename those columns so we can refer to them later.

```
df.columns = ['year', 'rank', 'company', 'revenue', 'profit']
```

Next, we need to explore our data set. Is it complete? Did pandas read it as expected? Are any values missing?

```
len(df)
```

```
25500
```

Okay, that looks good — that’s 500 rows for every year from 1955 to 2005, inclusive.

Let’s check whether our data set has been imported as we would expect. A simple check is to see if the data types (or dtypes) have been correctly interpreted.

```
df.dtypes
```

```
year int64 rank int64 company object revenue float64 profit object dtype: object
```

Uh oh. It looks like there’s something wrong with the profits column — we would expect it to be a float64 like the revenue column. This indicates that it probably contains some non-integer values, so let’s take a look.

```
non_numberic_profits = df.profit.str.contains('[^0-9.-]')
df.loc[non_numberic_profits].head()
```

year	rank	company	revenue	profit	
228	1955	229	Norton	135.0	N.A.
290	1955	291	Schlitz Brewing	100.0	N.A.
294	1955	295	Pacific Vegetable Oil	97.9	N.A.
296	1955	297	Liebmann Breweries	96.0	N.A.
352	1955	353	Minneapolis-Moline	77.4	N.A.

Just as we suspected! Some of the values are strings, which have been used to indicate missing data. Are there any other values that have crept in?

```
set(df.profit[non_numberic_profits])
```

```
{'N.A.'}
```

That makes it easy to interpret, but what should we do? Well, that depends how many values are missing.

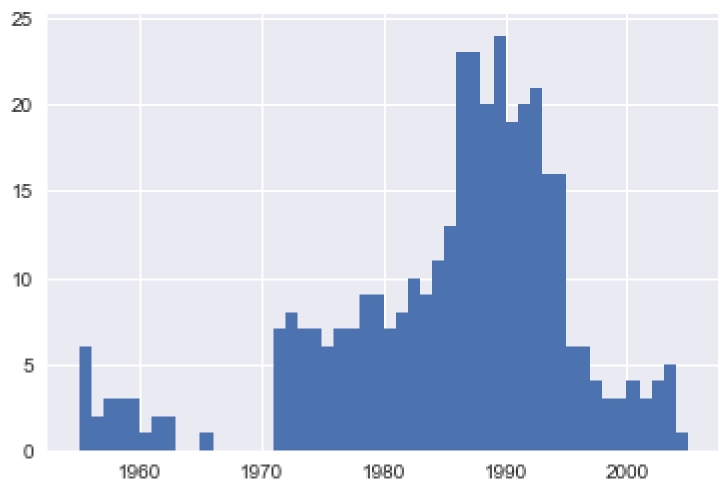
```
len(df.profit[non_numberic_profits])
```

```
369
```

It's a small fraction of our data set, though not completely inconsequential as it is still around 1.5%.

If rows containing N.A. are, roughly, uniformly distributed over the years, the easiest solution would just be to remove them. So let's have a quick look at the distribution.

```
bin_sizes, _, _ = plt.hist(df.year[non_numberic_profits], bins=range(1955, 2006))
```



At a glance, we can see that the most invalid values in a single year is fewer than 25, and as there are 500 data points per year, removing these values would account for less than 4% of the data for the worst years. Indeed, other than a surge around the 90s, most years have fewer than half the missing values of the peak.

For our purposes, let's say this is acceptable and go ahead and remove these rows.

```
df = df.loc[~non_numberic_profits]
df.profit = df.profit.apply(pd.to_numeric)
```

We should check that worked.

```
len(df)
```

```
25131
```

```
df.dtypes
```

```
year int64 rank int64 company object revenue float64 profit float64 dtype: object
```

Great! We have finished our data set setup.

If we were going to present your notebook as a report, we could get rid of the investigatory cells we created, which are included here as a demonstration of the flow of working with notebooks, and merge relevant cells (see the Advanced Functionality section below for more on this) to create a single data set setup cell.

This would mean that if we ever mess up our data set elsewhere, we can just rerun the setup cell to restore it.

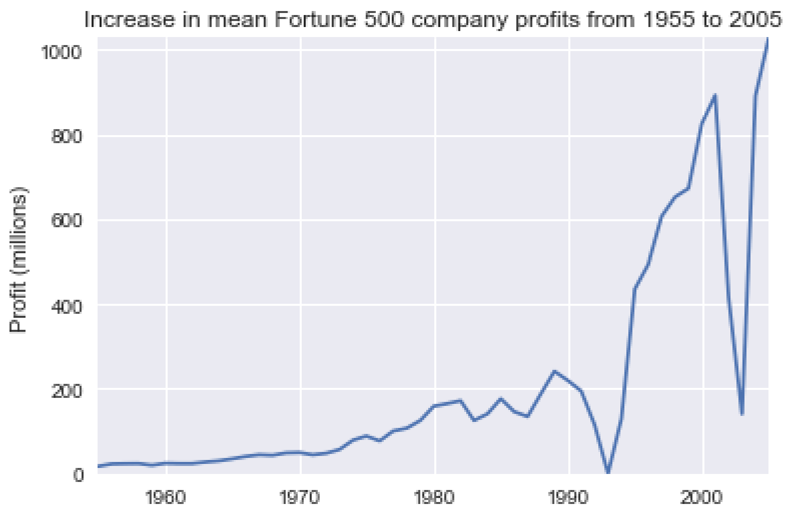
Plotting with matplotlib

Next, we can get to addressing the question at hand by plotting the average profit by year. We might as well plot the revenue as well, so first we can define some variables and a method to reduce our code.

```
group_by_year = df.loc[:, ['year', 'revenue', 'profit']].groupby('year')
avgs = group_by_year.mean()
x = avgs.index
y1 = avgs.profit
def plot(x, y, ax, title, y_label):
    ax.set_title(title)
    ax.set_ylabel(y_label)
    ax.plot(x, y)
    ax.margins(x=0, y=0)
```

Now let’s plot!

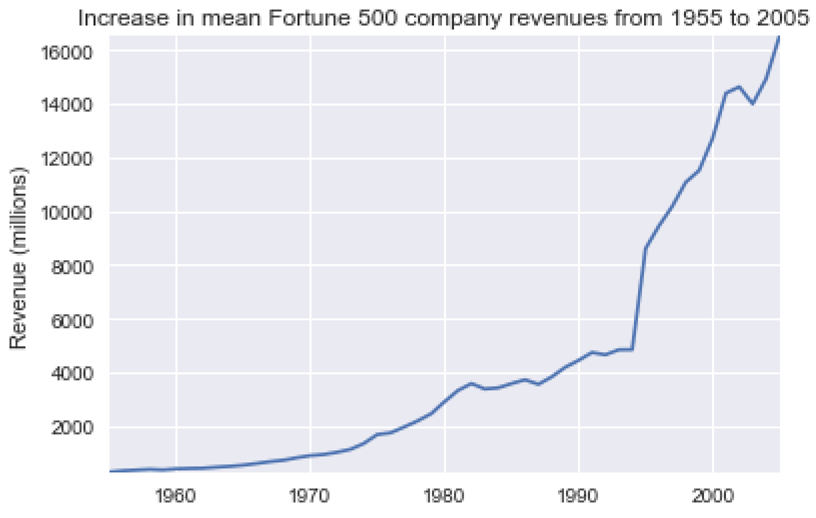
```
fig, ax = plt.subplots()
plot(x, y1, ax, 'Increase in mean Fortune 500 company profits from 1955 to 2005', 'Profit (millions)')
```



Wow, that looks like an exponential, but it's got some huge dips. They must correspond to the early 1990s recession and the dot-com bubble. It's pretty interesting to see that in the data. But how come profits recovered to even higher levels post each recession?

Maybe the revenues can tell us more.

```
y2 = avgs.revenue
fig, ax = plt.subplots()
plot(x, y2, ax, 'Increase in mean Fortune 500 company revenues from 1955 to 2005', 'Revenue (millions)')
```

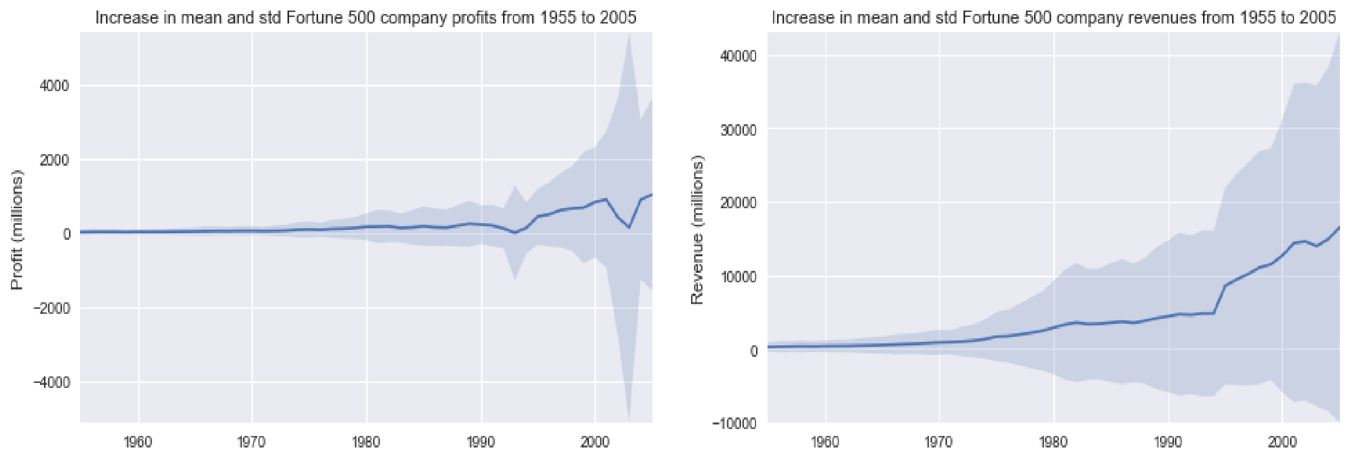


That adds another side to the story. Revenues were not as badly hit — that's some great accounting work from the finance departments.

With a little help from Stack Overflow, we can superimpose these plots with +/- their standard deviations.

```
def plot_with_std(x, y, stds, ax, title, y_label):
    ax.fill_between(x, y - stds, y + stds, alpha=0.2)
    plot(x, y, ax, title, y_label)
```

```
fig, (ax1, ax2) = plt.subplots(ncols=2)
title = 'Increase in mean and std Fortune 500 company %s from 1955 to 2005'
stds1 = group_by_year.std().profit.values
stds2 = group_by_year.std().revenue.values
plot_with_std(x, y1.values, stds1, ax1, title % 'profits', 'Profit (millions)')
plot_with_std(x, y2.values, stds2, ax2, title % 'revenues', 'Revenue (millions)')
fig.set_size_inches(14, 4)
fig.tight_layout()
```



That’s staggering, the standard deviations are huge! Some Fortune 500 companies make billions while others lose billions, and the risk has increased along with rising profits over the years.

Perhaps some companies perform better than others; are the profits of the top 10% more or less volatile than the bottom 10%?

There are plenty of questions that we could look into next, and it’s easy to see how the flow of working in a notebook can match one’s own thought process. For the purposes of this tutorial, we’ll stop our analysis here, but feel free to continue digging into the data on your own!

This flow helped us to easily investigate our data set in one place without context switching between applications, and our work is immediately shareable and reproducible. If we wished to create a more concise report for a particular audience, we could quickly refactor our work by merging cells and removing intermediary code.

3. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

FIND-S Algorithm

```
1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
     For each attribute constraint $a_i$ in h
        If the constraint $a_i$ is satisfied by x
        Then do nothing
        Else replace $a_i$ in h by the next more general constraint that is satisfied by x
3. Output hypothesis h
```

Training Examples:

Example	**Sky**	**AirTemp**	**Humidity**	**Wind**	**Water**	**Forecast**	**EnjoySport**
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Program:

```
import csv

a = []

with open('enjoysport.csv', 'r') as csvfile:
    for row in csv.reader(csvfile):
        a.append(row)
    print(a)

print("\n The total number of training instances are : ",len(a))

num_attribute = len(a[0])-1

print("\n The initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)

for i in range(0, len(a)):
    if a[i][num_attribute] == 'yes':
        for j in range(0, num_attribute):
            if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
                hypothesis[j] = a[i][j]
            else:
                hypothesis[j] = '?'
    print("\n The hypothesis for the training instance {} is :
\n" .format(i+1),hypothesis)

print("\n The Maximally specific hypothesis for the training
instance is ")
print(hypothesis)
```

Data Set:

sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Output:

```
The Given Training Data Set

['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']

The total number of training instances are :  4

The initial hypothesis is :
 ['0', '0', '0', '0', '0', '0']

The hypothesis for the training instance 1 is :
 ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

The hypothesis for the training instance 2 is :
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 3 is :
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 4 is :
 ['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally specific hypothesis for the training instance is
 ['sunny', 'warm', '?', 'strong', '?', '?']
```

4. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

CANDIDATE-ELIMINATION Learning Algorithm

The CANDIDATE-ELIMINTION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

```
Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H
For each training example d, do
  • If d is a positive example
      • Remove from G any hypothesis inconsistent with d
      • For each hypothesis s in S that is not consistent with d
          • Remove s from S
          • Add to S all minimal generalizations h of s such that
              • h is consistent with d, and some member of G is more general than h
          • Remove from S any hypothesis that is more general than another hypothesis in S

  • If d is a negative example
      • Remove from S any hypothesis inconsistent with d
      • For each hypothesis g in G that is not consistent with d
          • Remove g from G
          • Add to G all minimal specializations h of g such that
              • h is consistent with d, and some member of S is more specific than h
          • Remove from G any hypothesis that is less general than another hypothesis in G
```

CANDIDATE- ELIMINTION algorithm using version spaces

Training Examples:

Example	**Sky**	**AirTemp**	**Humidity**	**Wind**	**Water**	**Forecast**	**EnjoySport**
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Program:

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print(general_h)
    for i, h in enumerate(concepts):
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
                print(specific_h)
        print(specific_h)
        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print(" steps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
    indices = [i for i, val in enumerate(general_h) if val ==
['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

Data Set:

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Output:

```
Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']

Final General_h:
[['sunny', '?', '?', '?', '?', '?'],
 ['?', 'warm', '?', '?', '?', '?']]
```

5. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

<u>ID3 Algorithm</u>

```
ID3(Examples, Target_attribute, Attributes)

    Examples are the training examples. Target_attribute is the attribute whose value is to be
    predicted by the tree. Attributes is a list of other attributes that may be tested by the
    learned decision tree. Returns a decision tree that correctly classifies the given Examples.

• Create a Root node for the tree
• If all Examples are positive, Return the single-node tree Root, with label = +
• If all Examples are negative, Return the single-node tree Root, with label = -
• If Attributes is empty, Return the single-node tree Root, with label = most common value
  of Target_attribute in Examples

• Otherwise Begin
    • A ← the attribute from Attributes that best* classifies Examples
    • The decision attribute for Root ← A
    • For each possible value, $v_i$, of A,
        • Add a new tree branch below Root, corresponding to the test A = $v_i$
        • Let Examples $_{v_i}$, be the subset of Examples that have value $v_i$ for A
        • If Examples $_{v_i}$ , is empty
            • Then below this new branch add a leaf node with label = most common
              value of Target_attribute in Examples
            • Else below this new branch add the subtree
                ID3(Examples $_{v_i}$, Targe_tattribute, Attributes – {A}))
• End
• Return Root
```

* The best attribute is the one with highest information gain

ENTROPY:

Entropy measures the impurity of a collection of examples.

$$Entropy\ (S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Where, p_+ is the proportion of positive examples in S

p_- is the proportion of negative examples in S.

INFORMATION GAIN:

- ***Information gain,*** is the expected reduction in entropy caused by partitioning the examples according to this attribute.
- The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Training Dataset:

Day	***Outlook***	***Temperature***	***Humidity***	***Wind***	***PlayTennis***
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Test Dataset:

Day	***Outlook***	***Temperature***	***Humidity***	***Wind***
T1	Rain	Cool	Normal	Strong
T2	Sunny	Mild	Normal	Strong

Program:

```
import math
import csv

def load_csv(filename):
    lines=csv.reader(open(filename,"r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset,headers

class Node:
    def___init__(self,attribute):
        self.attribute=attribute
        self.children=[]
        self.answer=""

def subtables(data,col,delete):
    dic={}
    coldata=[row[col] for row in data]
    attr=list(set(coldata))

    counts=[0]*len(attr)
    r=len(data)
    c=len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col]==attr[x]:
                counts[x]+=1

    for x in range(len(attr)):
        dic[attr[x]]=[[0 for i in range(c)] for j in
range(counts[x])]
        pos=0
        for y in range(r):
            if data[y][col]==attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos]=data[y]
                pos+=1
    return attr,dic
```

```
def entropy(S):
    attr=list(set(S))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums

def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in
dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy

def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]

    attr,dic=subtables(data,split,delete=True)
    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node

def print_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)
        return

    print("  "*level,node.attribute)
    for value,n in node.children:
        print("  "*(level+1),value)
        print_tree(n,level+2)


def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)

'''Main program'''
dataset,features=load_csv("data3.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm
is")
print_tree(node1,0)
testdata,features=load_csv("data3_test.csv")
for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:",end="   ")
    classify(node1,xtest,features)
```

Output:

The decision tree for the dataset using ID3 algorithm is

```
 Outlook
   rain
         Wind
                  strong
                          no
                  weak
                         yes
   overcast
         yes

   sunny
         Humidity
                  normal
                         yes
                  high
                         no
```

```
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance:  no

The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance: yes
```

6. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

<u>BACKPROPAGATION Algorithm</u>

BACKPROPAGATION (*training_example, η, n_{in}, n_{out}, n_{hidden}*)

Each training example is a pair of the form ($\vec{x}$, $\vec{t}$), where ($\vec{x}$) is the vector of network input values, ($\vec{t}$) and is the vector of target network output values.

η is the learning rate (e.g., .05). n_i, is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}

- Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do

  - For each ($\vec{x}$, $\vec{t}$), in training examples, Do

    Propagate the input forward through the network:

    1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

    Propagate the errors backward through the network:

    2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

    3. For each hidden unit *h*, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

    4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta \delta_j x_{i,j}$$

Training Examples:

Example	Sleep	Study	Expected % in Exams
1	2	9	92
2	1	5	86
3	3	6	89

Normalize the input

Example	Sleep	Study	Expected % in Exams
1	2/3 = 0.66666667	9/9 = 1	0.92
2	1/3 = 0.33333333	5/9 = 0.55555556	0.86
3	3/3 = 1	6/9 = 0.66666667	0.89

Program:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) # maximum of X array longitudinally
y = y/100

#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
epoch=5000    #Setting training iterations
lr=0.1        #Setting learning rate
inputlayer_neurons = 2      #number of features in data set
hiddenlayer_neurons = 3     #number of hidden layers neurons
output_neurons = 1          #number of neurons at output layer
```

```
#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neur
ons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neuron
s))
bout=np.random.uniform(size=(1,output_neurons))


#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):

#Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)


#Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)

#how much hidden layer wts contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad

# dotproduct of nextlayererror and currentlayerop
     wout += hlayer_act.T.dot(d_output) *lr
     wh += X.T.dot(d_hiddenlayer) *lr

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Output:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]

Predicted Output:
 [[0.89726759]
 [0.87196896]
 [0.9000671]]
```

7. Write a program to implement the naïve Bayesian classifier for a sample training data set storedas a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Bayes' Theorem is stated as:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Where,
P(h|D) is the probability of hypothesis h given the data D. This is called the **posterior probability**.
P(D|h) is the probability of data d given that the hypothesis h was true.
P(h) is the probability of hypothesis h being true. This is called the **prior probability of h.**
P(D) is the probability of the data. This is called the **prior probability of D**

After calculating the posterior probability for a number of different hypotheses h, and is interested in finding the most probable hypothesis h ∈ H given the observed data D. Any such maximally probable hypothesis is called a ***maximum a posteriori (MAP) hypothesis***.

Bayes theorem to calculate the posterior probability of each candidate hypothesis is $\boldsymbol{h_{MAP}}$ is a MAP hypothesis provided

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

$$= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)}$$

$$= \arg\max_{h \in H} P(D|h)P(h)$$

(Ignoring P(D) since it is a constant)

Gaussian Naive Bayes

A Gaussian Naive Bayes algorithm is a special type of Naïve Bayes algorithm. It's specifically used when the features have continuous values. It's also assumed that all the features are following a Gaussian distribution i.e., normal distribution

Representation for Gaussian Naive Bayes

We calculate the probabilities for input values for each class using a frequency. With real-valued inputs, we can calculate the mean and standard deviation of input values (x) for each class to summarize the distribution.

This means that in addition to the probabilities for each class, we must also store the mean and standard deviations for each input variable for each class.

Gaussian Naive Bayes Model from Data

The probability density function for the normal distribution is defined by two parameters (mean and standard deviation) and calculating the mean and standard deviation values of each input variable (x) for each class value.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Mean

$$\sigma = \left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2\right]^{0.5}$$

Standard deviation

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Normal distribution

Example: Refer the link

http://chem-eng.utoronto.ca/~datamining/dmc/naive_bayesian.htm

<u>Examples:</u>

- The data set used in this program is the ***Pima Indians Diabetes problem***.
- This data set is comprised of 768 observations of medical details for Pima Indians patents. The records describe instantaneous measurements taken from the patient such as their age, the number of times pregnant and blood workup. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute.
- The attributes are ***Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabeticPedigreeFunction, Age, Outcome***
- Each record has a class value that indicates whether the patient suffered an onset of diabetes within 5 years of when the measurements were taken (1) or not (0)

Sample Examples:

Examples	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetic Pedigree Function	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0	0.232	54	1

Program:

```
import csv
import random
import math

def loadcsv(filename):
     lines = csv.reader(open(filename, "r"));
     dataset = list(lines)
     for i in range(len(dataset)):
        #converting strings into numbers for processing
          dataset[i] = [float(x) for x in dataset[i]]

     return dataset

def splitdataset(dataset, splitratio):
    #67% training size
     trainsize = int(len(dataset) * splitratio);
     trainset = []
     copy = list(dataset);
     while len(trainset) < trainsize:
#generate indices for the dataset list randomly to pick ele for
training data
          index = random.randrange(len(copy));
          trainset.append(copy.pop(index))
     return [trainset, copy]

def separatebyclass(dataset):
     separated = {} #dictionary of classes 1 and 0
#creates a dictionary of classes 1 and 0 where the values are
#the instances belonging to each class
     for i in range(len(dataset)):
          vector = dataset[i]
          if (vector[-1] not in separated):
               separated[vector[-1]] = []
          separated[vector[-1]].append(vector)
     return separated

def mean(numbers):
     return sum(numbers)/float(len(numbers))

def stdev(numbers):
     avg = mean(numbers)
     variance = sum([pow(x-avg,2) for x in
numbers])/float(len(numbers)-1)
     return math.sqrt(variance)
```

```
def summarize(dataset): #creates a dictionary of classes
    summaries = [(mean(attribute), stdev(attribute)) for
attribute in zip(*dataset)];
    del summaries[-1] #excluding labels +ve or -ve
    return summaries

def summarizebyclass(dataset):
    separated = separatebyclass(dataset);
   #print(separated)
    summaries = {}
    for classvalue, instances in separated.items():
#for key,value in dic.items()
#summaries is a dic of tuples(mean,std) for each class value
        summaries[classvalue] = summarize(instances)
#summarize is used to cal to mean and std
    return summaries

def calculateprobability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/
(2*math.pow(stdev,2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calculateclassprobabilities(summaries, inputvector):
# probabilities contains the all prob of all class of test data
    probabilities = {}
    for classvalue, classsummaries in summaries.items():
#class and attribute information as mean and sd
        probabilities[classvalue] = 1
        for i in range(len(classsummaries)):
            mean, stdev = classsummaries[i] #take mean and
sd of every attribute for class 0 and 1 seperaely
            x = inputvector[i] #testvector's first attribute
            probabilities[classvalue] *=
calculateprobability(x, mean, stdev);#use normal dist
    return probabilities

def predict(summaries, inputvector): #training and test data
is passed
    probabilities = calculateclassprobabilities(summaries,
inputvector)
    bestLabel, bestProb = None, -1
    for classvalue, probability in probabilities.items():
#assigns that class which has the highest prob
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classvalue
    return bestLabel
```

```
def getpredictions(summaries, testset):
     predictions = []
     for i in range(len(testset)):
          result = predict(summaries, testset[i])
          predictions.append(result)
     return predictions

def getaccuracy(testset, predictions):
     correct = 0
     for i in range(len(testset)):
          if testset[i][-1] == predictions[i]:
               correct += 1
     return (correct/float(len(testset))) * 100.0

def main():
     filename = 'naivedata.csv'
     splitratio = 0.67
     dataset = loadcsv(filename);

     trainingset, testset = splitdataset(dataset, splitratio)
     print('Split {0} rows into train={1} and test={2}
rows'.format(len(dataset), len(trainingset), len(testset)))
     # prepare model
     summaries = summarizebyclass(trainingset);
     #print(summaries)
    # test model
     predictions = getpredictions(summaries, testset) #find the
predictions of test data with the training data
     accuracy = getaccuracy(testset, predictions)
     print('Accuracy of the classifier is :
{0}%'.format(accuracy))

main()
```

Output:

```
Split 768 rows into train=514 and test=254 rows

Accuracy of the classifier is : 71.65354330708661%
```

8. Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate theaccuracy, precision, and recall for your data set.

<u>Naive Bayes algorithms for learning and classifying text</u>

LEARN_NAIVE_BAYES_TEXT (Examples, V)

Examples is a set of text documents along with their target values. V is the set of all possible target values. This function learns the probability terms $P(w_k | v_j,)$, describing the probability that a randomly drawn word from a document in class v_j will be the English word w_k. It also learns the class prior probabilities $P(v_j)$.

```
1. collect all words, punctuation, and other tokens that occur in Examples
     • Vocabulary ← c the set of all distinct words and other tokens occurring in any text
       document from Examples

2. calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms

     • For each target value $v_j$ in V do

         • $docs_j$ ← the subset of documents from Examples for which the target value is vj

         • $P(v_j)$ ← | $docs_j$ | / |Examples|

         • $Text_j$ ← a single document created by concatenating all members of $docs_j$

         • n ← total number of distinct word positions in $Text_j$

         • for each word $w_k$ in Vocabulary

             • $n_k$ ← number of times word $w_k$ occurs in $Text_j$

             • $P(w_k|v_j)$ ← ( $n_k$ + 1) / (n + | Vocabulary| )
```

CLASSIFY_NAIVE_BAYES_TEXT (Doc)

Return the estimated target value for the document Doc. a_i denotes the word found in the i^{th} position within Doc.

```
• positions ← all word positions in Doc that contain tokens found in Vocabulary
• Return $V_{NB}$, where
```

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_{i \in positions} P(a_i | v_j)$$

Data set:

	Text Documents	**Label**
1	I love this sandwich	pos
2	This is an amazing place	pos
3	I feel very good about these beers	pos
4	This is my best work	pos
5	What an awesome view	pos
6	I do not like this restaurant	neg
7	I am tired of this stuff	neg
8	I can't deal with this	neg
9	He is my sworn enemy	neg
10	My boss is horrible	neg
11	This is an awesome place	pos
12	I do not like the taste of this juice	neg
13	I love to dance	pos
14	I am sick and tired of this place	neg
15	What a great holiday	pos
16	That is a bad locality to stay	neg
17	We will have good fun tomorrow	pos
18	I went to my enemy's house today	neg

Program:

```
import pandas as pd

msg=pd.read_csv('naivetext.csv',names=['message','label'])

print('The dimensions of the dataset',msg.shape)

msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum

print(X)
print(y)

#splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,y)

print ('\n The total number of Training Data :',ytrain.shape)
print ('\n The total number of Test Data :',ytest.shape)


#output of count vectoriser is a sparse matrix
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print('\n The words or Tokens in the text documents \n')
print(count_vect.get_feature_names())

df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_fe
ature_names())

# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)

#printing accuracy, Confusion matrix, Precision and Recall
from sklearn import metrics
print('\n Accuracy of the classifer is’,
metrics.accuracy_score(ytest,predicted))
```

```
print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))

print('\n The value of Precision' ,
metrics.precision_score(ytest,predicted))


print('\n The value of Recall' ,
metrics.recall_score(ytest,predicted))
```

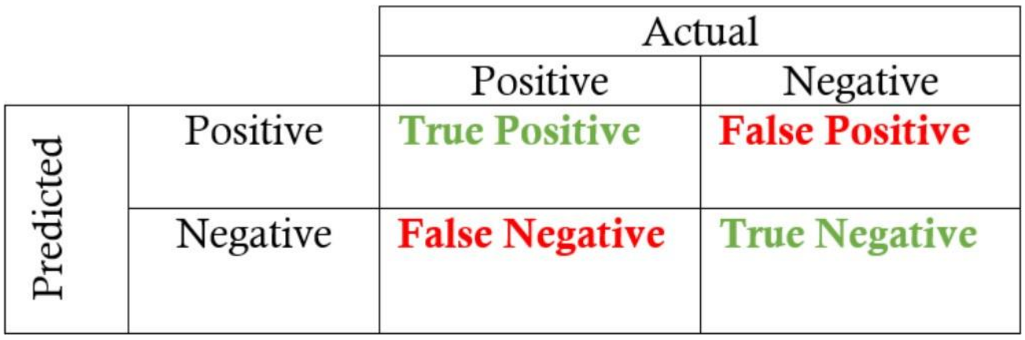
Output:

```
The dimensions of the dataset (18, 2)
0       I love this sandwich
1       This is an amazing place
2       I feel very good about these beers
3       This is my best work
4       What an awesome view
5       I do not like this restaurant
6       I am tired of this stuff
7       I can't deal with this
8       He is my sworn enemy
9       My boss is horrible
10      This is an awesome place
11       I do not like the taste of this juice
12      I love to dance
13      I am sick and tired of this place
14      What a great holiday
15      That is a bad locality to stay
16      We will have good fun tomorrow
17      I went to my enemy's house today
```

```
Name: message, dtype: object
0     1
1     1
2     1
3     1
4     1
5     0
6     0
7     0
8     0
9     0
10    1
11    0
12    1
13    0
14    1
15    0
16    1
17    0
Name: labelnum, dtype: int64

The total number of Training Data: (13,)

The total number of Test Data: (5,)


The words or Tokens in the text documents

['about', 'am', 'amazing', 'an', 'and', 'awesome', 'beers', 'best', 'can', 'deal', 'do', 'enemy', 'feel',
'fun', 'good', 'great', 'have', 'he', 'holiday', 'house', 'is', 'like', 'love', 'my', 'not', 'of', 'place',
'restaurant', 'sandwich', 'sick', 'sworn', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very',
'view', 'we', 'went', 'what', 'will', 'with', 'work']


Accuracy of the classifier is 0.8

Confusion matrix

[[2 1]

 [0 2]]

The value of Precision 0.6666666666666666

The value of Recall 1.0
```

Basic knowledge

Confusion Matrix



		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

True positives: data points labelled as positive that are actually positive

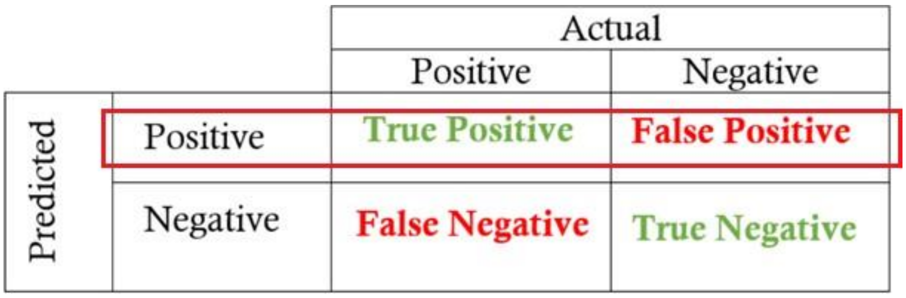
False positives: data points labelled as positive that are actually negative

True negatives: data points labelled as negative that are actually negative

False negatives: data points labelled as negative that are actually positive

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

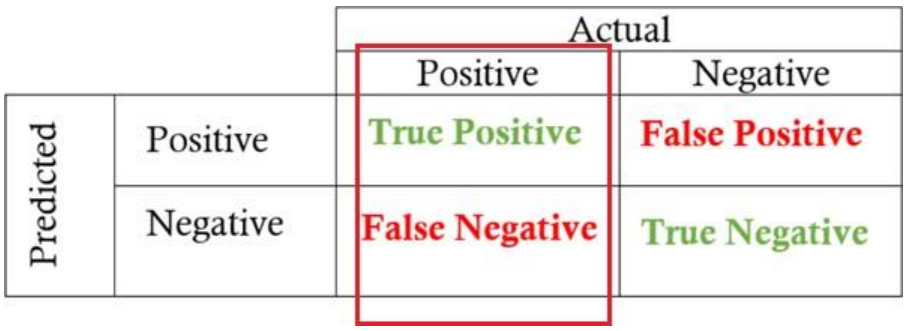
$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$



		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

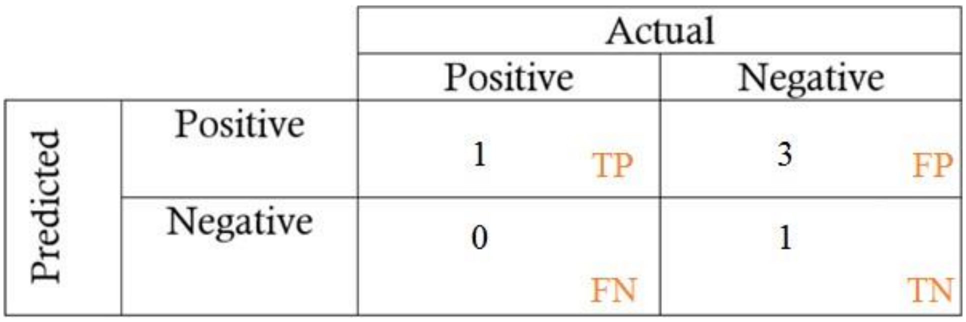
$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$= \frac{True\ Positive}{Total\ Predicted\ Positive}$$



		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Example:



		Actual	
		Positive	Negative
Predicted	Positive	1 TP	3 FP
	Negative	0 FN	1 TN

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{1}{1+3} = 0.25$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{1}{1+0} = 1$$

Accuracy: how often is the classifier correct?

$$\text{Accuracy} = \frac{\text{TP + TN}}{\text{Total}} = \frac{1+1}{5} = 0.4$$

Example: Movie Review

Doc	Text	Class
1	I loved the movie	+
2	I hated the movie	-
3	a great movie. good movie	+
4	poor acting	-
5	great acting. good movie	+

Unique word

< I, loved, the, movie, hated, a, great, good, poor, acting>

Doc	I	loved	the	movie	hated	a	great	good	poor	acting	Class
1	1	1	1	1							+
2	1		1	1	1						-
3				2		1	1	1			+
4									1	1	-
5				1			1	1		1	+

Doc	I	loved	the	movie	hated	a	great	good	poor	acting	Class
1	1	1	1	1							+
3				2		1	1	1			+
5				1			1	1		1	+

$$P(+) = \frac{3}{5} = 0.6$$

$P(I\ |+) = \dfrac{1+1}{14+10} = 0.0833$

$P(a\ |+) = \dfrac{1+1}{14+10} = 0.0833$

$P(loved\ |+) = \dfrac{1+1}{14+10} = 0.0833$

$P(great\ |+) = \dfrac{2+1}{14+10} = 0.125$

$P(the\ |+) = \dfrac{1+1}{14+10} = 0.0833$

$P(good\ |+) = \dfrac{2+1}{14+10} = 0.125$

$P(movie\ |+) = \dfrac{4+1}{14+10} = 0.2083$

$P(poor\ |+) = \dfrac{0+1}{14+10} = 0.0416$

$P(hated\ |+) = \dfrac{0+1}{14+10} = 0.0416$

$P(acting\ |+) = \dfrac{1+1}{14+10} = 0.0833$

Doc	I	loved	the	movie	hated	a	great	good	poor	acting	Class
2	1		1	1	1						-
4									1	1	-

$P(-) = \dfrac{2}{5} = 0.4$

$P(I\ |-) = \dfrac{1+1}{6+10} = 0.125$

$P(a\ |-) = \dfrac{0+1}{6+10} = 0.0625$

$P(loved\ |-) = \dfrac{0+1}{6+10} = 0.0625$

$P(great\ |-) = \dfrac{0+1}{6+10} = 0.0625$

$P(the\ |-) = \dfrac{1+1}{6+10} = 0.125$

$P(good\ |-) = \dfrac{0+1}{6+10} = 0.0625$

$P(movie|-) = \dfrac{1+1}{6+10} = 0.125$

$P(poor|-) = \dfrac{1+1}{6+10} = 0.125$

$P(hated\ |-) = \dfrac{1+1}{6+10} = 0.125$

$P(acting|-) = \dfrac{1+1}{6+10} = 0.125$

Let's classify the new document

I hated the poor acting

If $V_j = +$

then,

$= P(+)\ P(I \mid +)\ P(hated \mid +)\ P(the \mid +)\ P(poor \mid +)\ P(acting \mid +)$

$= 0.6 * 0.0833 * 0.0416 * 0.0833 * 0.0416 * 0.0833$

$= 6.03 \text{ X } 10^{-2}$

If $V_j = -$

then,

$= P(-)\ P(I \mid -)\ P(hated \mid -)\ P(the \mid -)\ P(poor \mid -)\ P(acting \mid -)$

$= 0.4 * 0.125 * 0.125 * 0.125 * 0.125 * 0.125$

$= 1.22 \text{ X } 10^{-5}$

$= 1.22 \text{ X } 10^{-5} > 6.03 \text{ X } 10^{-2}$

So, the new document belongs to $(-)$ class

9. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You canuse Java/Python ML library classes/API

<u>Theory</u>

A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable.

Bayesian network consists of two major parts: a directed acyclic graph and a set of conditional probability distributions

- The directed acyclic graph is a set of random variables represented by nodes.
- The conditional probability distribution of a node (random variable) is defined for every possible outcome of the preceding causal node(s).

For illustration, consider the following example. Suppose we attempt to turn on our computer, but the computer does not start (observation/evidence). We would like to know which of the possible causes of computer failure is more likely. In this simplified illustration, we assume only two possible causes of this misfortune: electricity failure and computer malfunction.
The corresponding directed acyclic graph is depicted in below figure.

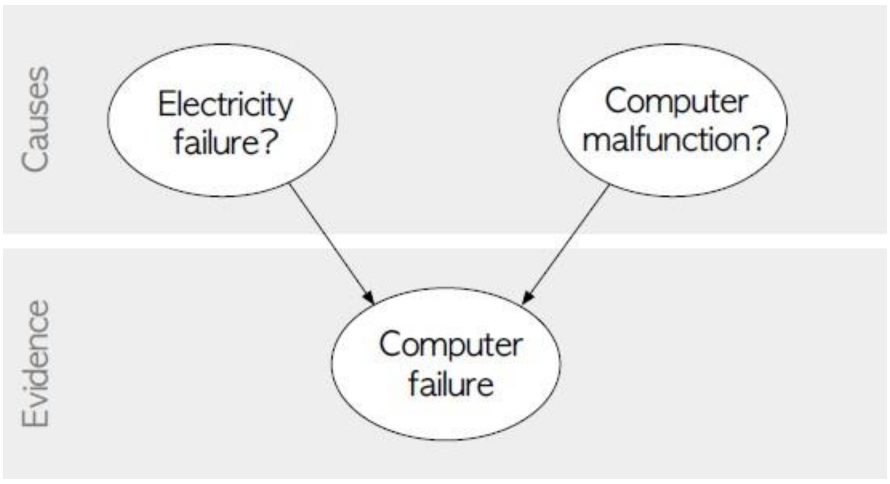


Fig: Directed acyclic graph representing two independent possible causes of a computer failure.

The goal is to calculate the posterior conditional probability distribution of each of the possible unobserved causes given the observed evidence, i.e. P [Cause | Evidence].

Data Set:

Title: Heart Disease Databases

The Cleveland database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

Database:	0	1	2	3	4	Total
Cleveland:	164	55	36	35	13	303

Attribute Information:

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type
    - Value 1: typical angina
    - Value 2: atypical angina
    - Value 3: non-anginal pain
    - Value 4: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
    - Value 0: normal
    - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: downsloping
12. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
13. Heartdisease: It is integer valued from 0 (no presence) to 4.

Some instance from the dataset:

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
60	1	4	130	206	0	2	132	1	2.4	2	2	7	4

Program:

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination


#read Cleveland Heart Disease data
heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

#display the data
print('Sample instances from the dataset are given below')
print(heartDisease.head())

#display the Attributes names and datatyes

print('\n Attributes and datatypes')
print(heartDisease.dtypes)


#Creat Model- Bayesian Network
model =
BayesianModel([('age','heartdisease'),('sex','heartdisease'),(
'exang','heartdisease'),('cp','heartdisease'),('heartdisease',
'restecg'),('heartdisease','chol')])
```

```
#Learning CPDs using Maximum Likelihood Estimators

print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)


# Inferencing with Bayesian Network

print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)


#computing the Probability of HeartDisease given restecg
print('\n 1.Probability of HeartDisease given evidence=
restecg :1')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evi
dence={'restecg':1})
print(q1)

#computing the Probability of HeartDisease given cp

print('\n 2.Probability of HeartDisease given evidence= cp:2 ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evi
dence={'cp':2})
print(q2)
```

Output:

```
================ RESTART: E:\ML Lab - 2020-21\MLLab-7\ML7.py ================
Few examples from the dataset are given below
   age  sex  cp  trestbps  chol  ...  oldpeak  slope  ca  thal  heartdisease
0   63    1   1       145   233  ...      2.3      3   0     6             0
1   67    1   4       160   286  ...      1.5      2   3     3             2
2   67    1   4       120   229  ...      2.6      2   2     7             1
3   37    1   3       130   250  ...      3.5      3   0     3             0
4   41    0   2       130   204  ...      1.4      1   0     3             0

[5 rows x 14 columns]

 Attributes and datatypes
age               int64
sex               int64
cp                int64
trestbps          int64
chol              int64
fbs               int64
restecg           int64
thalach           int64
exang             int64
oldpeak         float64
slope             int64
ca               object
thal             object
heartdisease      int64
dtype: object


Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg

   +-----------------+---------------------+
   | heartdisease    |   phi(heartdisease) |
   +=================+=====================+
   | heartdisease(0) |              0.1012 |
   +-----------------+---------------------+
   | heartdisease(1) |              0.0000 |
   +-----------------+---------------------+
   | heartdisease(2) |              0.2392 |
   +-----------------+---------------------+
   | heartdisease(3) |              0.2015 |
   +-----------------+---------------------+
   | heartdisease(4) |              0.4581 |
   +-----------------+---------------------+
2. Probability of HeartDisease given evidence= cp

   +-----------------+---------------------+
   | heartdisease    |   phi(heartdisease) |
   +=================+=====================+
   | heartdisease(0) |              0.3610 |
   +-----------------+---------------------+
   | heartdisease(1) |              0.2159 |
   +-----------------+---------------------+
   | heartdisease(2) |              0.1373 |
   +-----------------+---------------------+
   | heartdisease(3) |              0.1537 |
   +-----------------+---------------------+
   | heartdisease(4) |              0.1321 |
   +-----------------+---------------------+
```

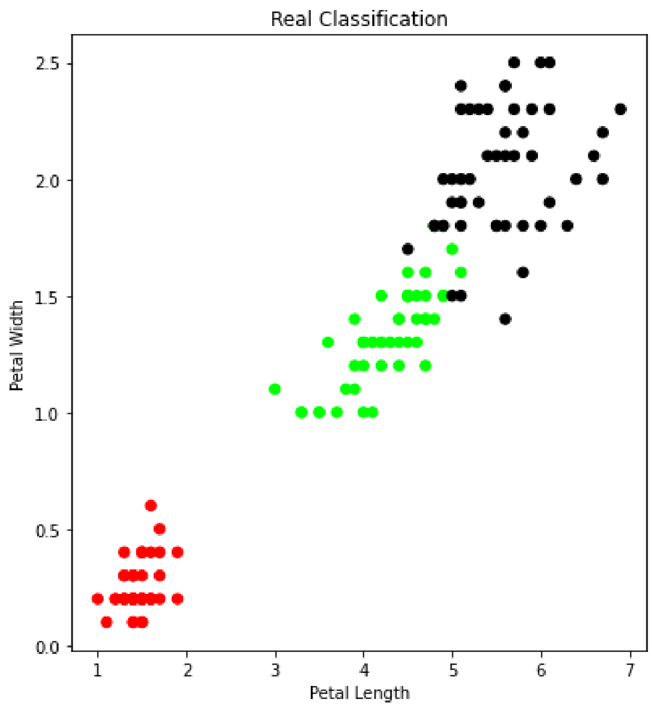
10. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data setfor clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Python ML library classes/APIin the program.

SOURCE CODE:

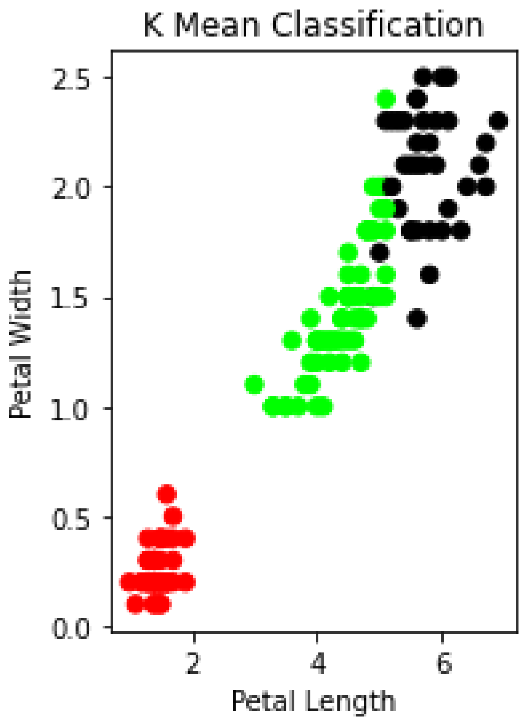
```
import matplotlib.pyplot as plt from sklearn
import datasets
from sklearn.cluster import KMeans import
sklearn.metrics as sm
import pandas as pd import numpy as np

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns =
['Sepal_Length','Sepal_Width','Petal_Length',
'Petal_Width']

y = pd.DataFrame(iris.target) y.columns =
['Targets']

model = KMeans(n_clusters=3) model.fit(X)


plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width,
c=colormap[y.Targets], s=40) plt.title('Real
Classification')
plt.xlabel('Petal Length') plt.ylabel('Petal
Width')
# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width,
c=colormap[model.labels_], s=40) plt.title('K
Mean Classification')
plt.xlabel('Petal Length') plt.ylabel('Petal
Width')
print('The accuracy score of K-Mean:
',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean:
',sm.confusion_matrix(y, model.labels_))
```

```
from sklearn import preprocessing scaler =
preprocessing.StandardScaler() scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm = gmm.predict(xs) #y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width,
c=colormap[y_gmm], s=40) plt.title('GMM
Classification')
plt.xlabel('Petal Length') plt.ylabel('Petal
Width')

print('The accuracy score of EM:
',sm.accuracy_score(y, y_gmm)) print('The
Confusion matrix of EM:
',sm.confusion_matrix(y, y_gmm))
```

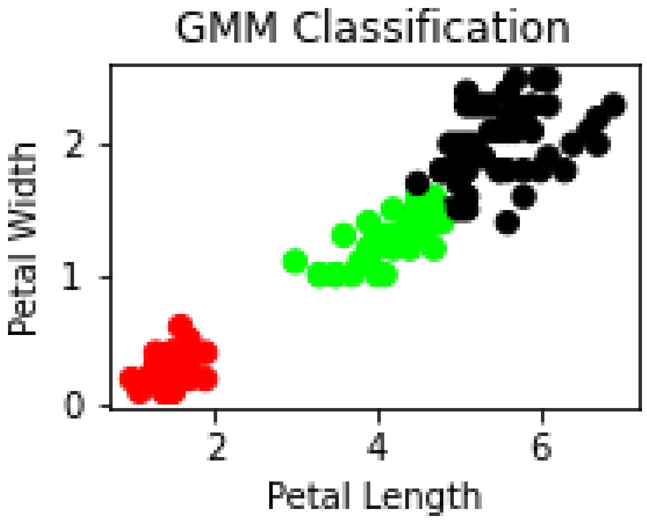
Output:



```
The accuracy score of K-Mean:  0.8933333333333333
The Confusion matrixof K-Mean:
[[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
```



```
The accuracy score of EM:  0.9666666666666667
The Confusion matrix of EM:  [[50  0  0]
 [ 0 45  5]
 [ 0  0 50]]
```



11. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

K-Nearest Neighbor Algorithm

Training algorithm:

- For each training example (x, f (x)), add the example to the list training examples

Classification algorithm:

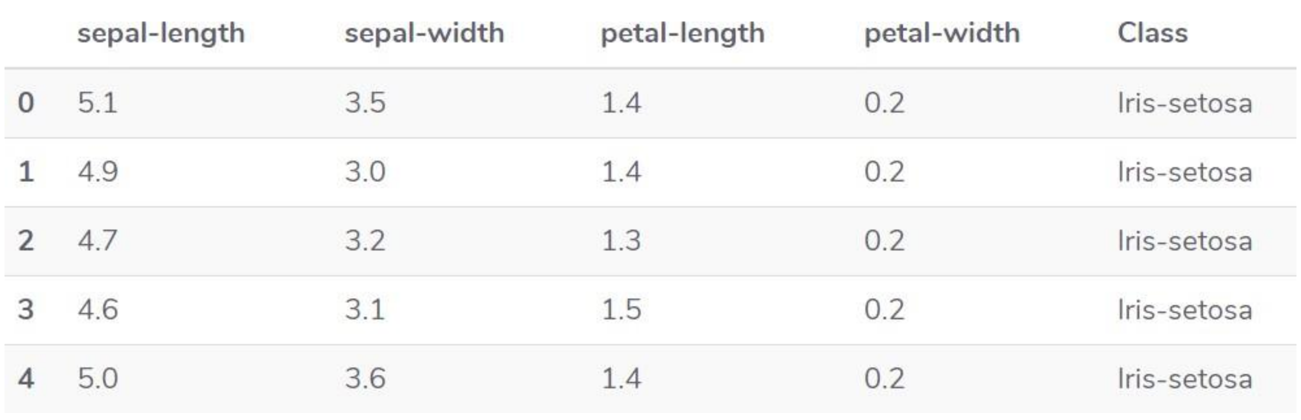
- Given a query instance x_q to be classified,
    - Let $x_1 \dots x_k$ denote the k instances from training examples that are nearest to x_q
    - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

- Where, $f(x_i)$ function to calculate the mean value of the k nearest training examples.

Data Set:

Iris Plants Dataset: Dataset contains 150 instances (50 in each of three classes)
Number of Attributes: 4 numeric, predictive attributes and the Class



	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Program:

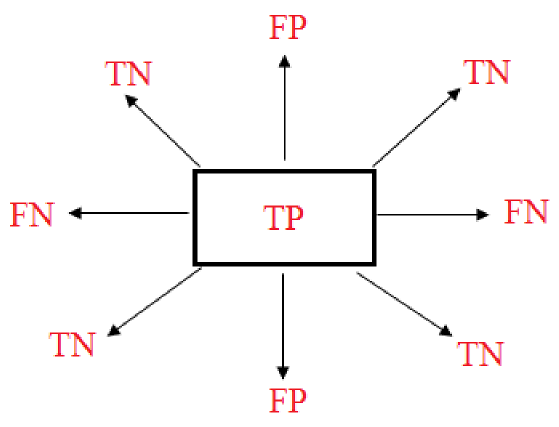
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets

""" Iris Plants Dataset, dataset contains 150 (50 in each of three
classes)Number of Attributes: 4 numeric, predictive attributes and
the Class
"""
iris=datasets.load_iris()

""" The x variable contains the first four columns of the dataset
(i.e. attributes) while y contains the labels.
"""
x  =  iris.data
y = iris.target

print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)

""" Splits the dataset into 70% train data and 30% test data. This
means that out of total 150 records, the training set will contain
105 records and the test set contains 45 of those records
"""
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.3)

#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

#to make predictions on our test data
y_pred=classifier.predict(x_test)

""" For evaluating an algorithm, confusion matrix, precision, recall
and f1 score are the most commonly used metrics.
"""
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

Output:

```
sepal-length sepal-width petal-length petal-width
[[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]
 .   .    .    .    .
 .   .    .    .    .

 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]

class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
[0 0 0 ………0 0 1 1 1 …………1 1 2 2 2 ………… 2 2]

Confusion Matrix
[[20  0  0]
 [ 0 10  0]
 [ 0  1 14]]

Accuracy Metrics

                Precision  recall    f1-score   support

          0       1.00      1.00      1.00        20
          1       0.91      1.00      0.95        10
          2       1.00      0.93      0.97        15

avg / total       0.98      0.98      0.98        45
```

Basic knowledge

Confusion Matrix



True positives: data points labelled as positive that are actually positive
False positives: data points labelled as positive that are actually negative
True negatives: data points labelled as negative that are actually negative
False negatives: data points labelled as negative that are actually positive

$$\text{Recall} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Negative}}$$

$$= \frac{\textit{True Positive}}{\textit{Total Actual Positive}}$$

$$\text{Precision} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Positive}}$$

$$= \frac{\textit{True Positive}}{\textit{Total Predicted Positive}}$$

Accuracy: how often is the classifier correct?

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total}}$$

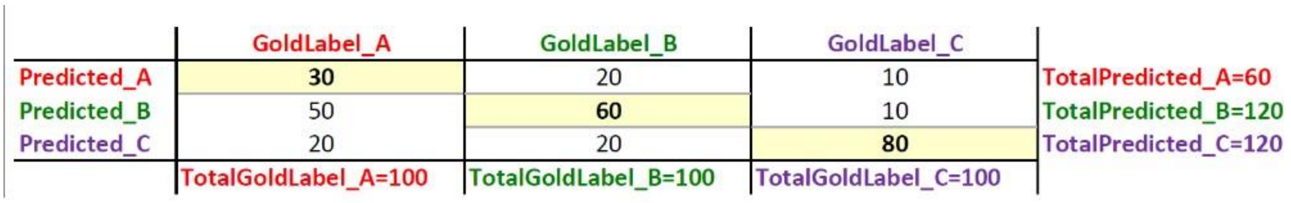
F1-Score:

$$\text{F1 Score} = \frac{2.\text{TP}}{2.\text{TP} + \text{FP} + \text{FN}}$$

Support: Total Predicted of Class.

$$\text{Support} = \text{TP} + \text{FN}$$

Example:



	GoldLabel_A	GoldLabel_B	GoldLabel_C	
Predicted_A	**30**	20	10	TotalPredicted_A=60
Predicted_B	50	**60**	10	TotalPredicted_B=120
Predicted_C	20	20	**80**	TotalPredicted_C=120
	TotalGoldLabel_A=100	TotalGoldLabel_B=100	TotalGoldLabel_C=100	

This is an example confusion matrix for 3 labels: A,B and C

- Now, let us compute **recall** for Label A:

```
= TP_A/(TP_A+FN_A)
= TP_A/(Total Gold for A)
= TP_A/TotalGoldLabel_A
= 30/100
= 0.3
```

- Now, let us compute **precision** for Label A:

```
= TP_A/(TP_A+FP_A)
= TP_A/(Total predicted as A)
= TP_A/TotalPredicted_A
= 30/60
= 0.5
```

- Now, let us compute **F1-score** for Label A:

$$\text{F1 Score} = \frac{2.\text{TP}}{2.\text{TP} + \text{FP} + \text{FN}}$$

$= 2*30 \;/\; (2*30 + 60 + 100)$

$= 0.27$

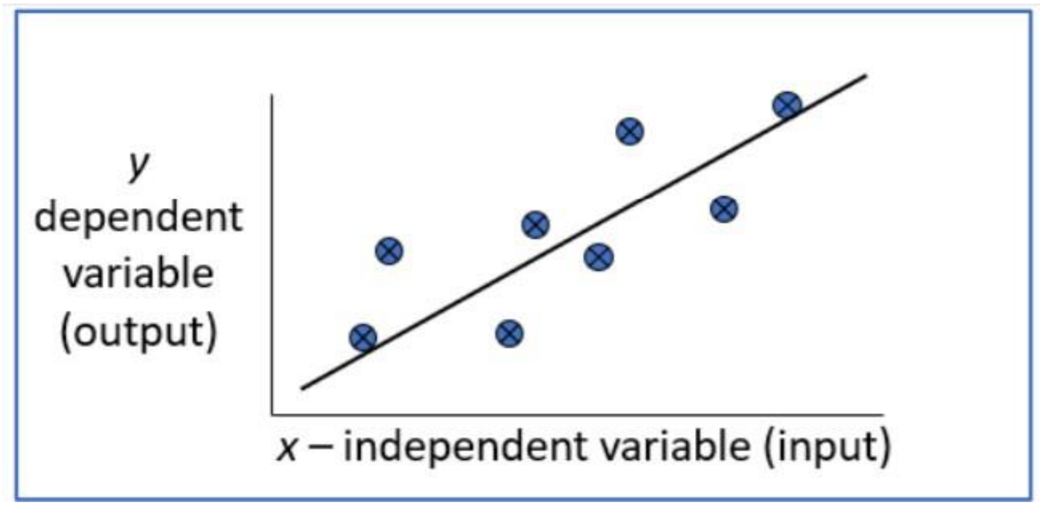
- Support _ A = TP_A + FN_A
  = 30 + (20 + 10)
  = 60

12. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

<u>Locally Weighted Regression Algorithm</u>

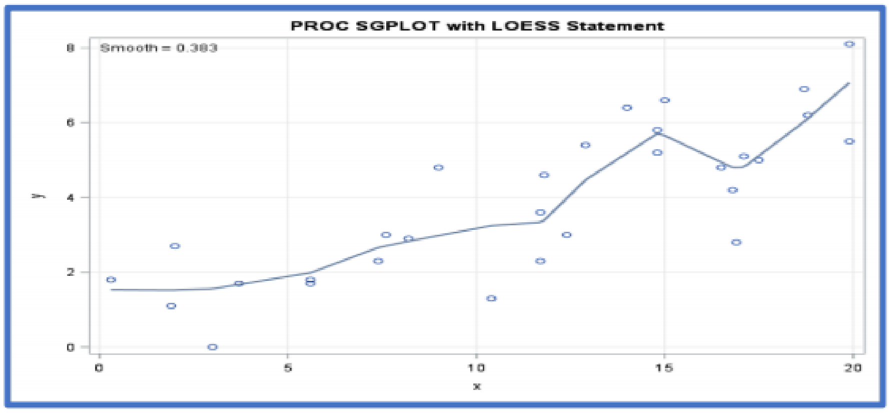
Regression:

- Regression is a technique from statistics that is used to predict values of a desired target quantity when the target quantity is continuous.
- In regression, we seek to identify (or estimate) a continuous variable y associated with a given input vector x.
    - y is called the dependent variable.
    - x is called the independent variable.



Loess/Lowess Regression:

Loess regression is a nonparametric technique that uses local weighted regression to fit a smooth curve through points in a scatter plot.



Lowess Algorithm:

- Locally weighted regression is a very powerful nonparametric model used in statistical learning.
- Given a dataset X, y, we attempt to find a model parameter β(x) that minimizes residual sum of weighted squared errors.
- The weights are given by a kernel function (k or w) which can be chosen arbitrarily

Algorithm

a. Read the Given data Sample to X and the curve (linear or non linear) to Y
b. Set the value for Smoothening parameter or Free parameter say τ
c. Set the bias /Point of interest set x0 which is a subset of X
d. Determine the weight matrix using :

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

e. Determine the value of model term parameter β using :

$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

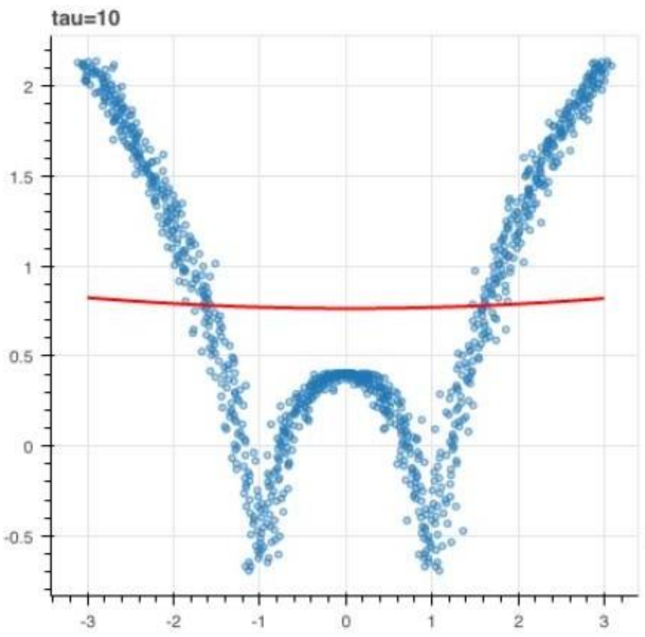
f. Prediction = x0*β:

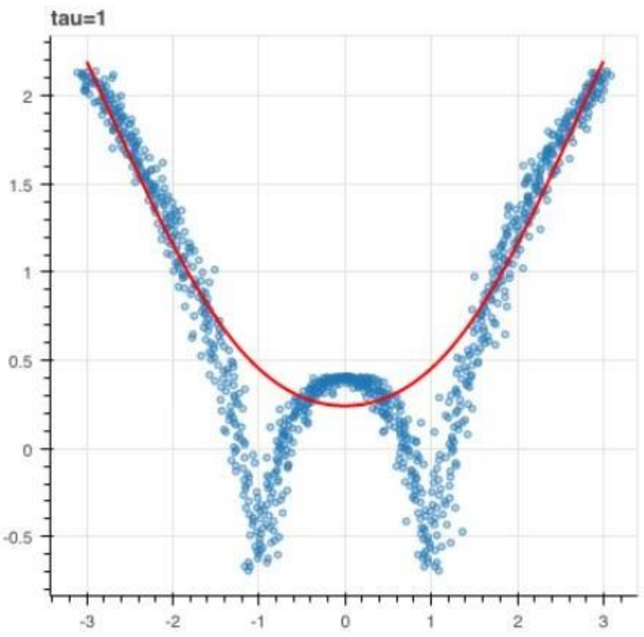
Program

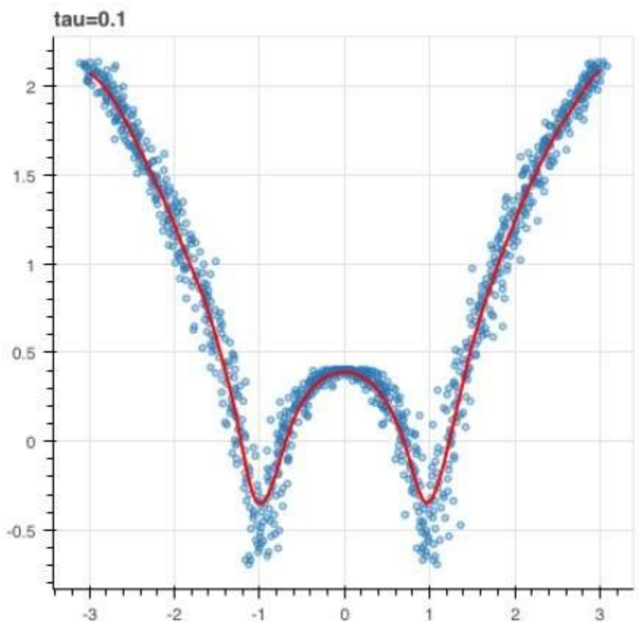
```
import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook

def local_regression(x0, X, Y, tau):# add bias term
 x0 = np.r_[1, x0] # Add one to avoid the loss in
information
 X = np.c_[np.ones(len(X)), X]

 # fit model: normal equations with kernel
 xw = X.T * radial_kernel(x0, X, tau) # XTranspose * W

 beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix
Multiplication or Dot Product
```

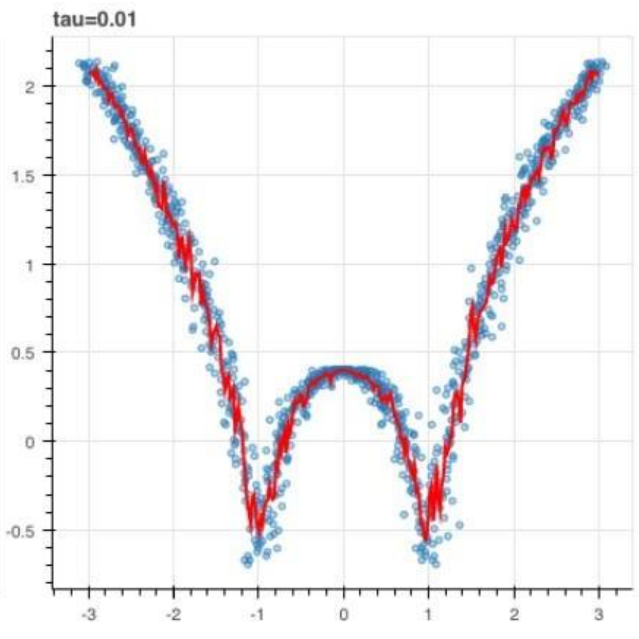
```
 # predict value
 return x0 @ beta # @ Matrix Multiplication or Dot Product
for prediction
def radial_kernel(x0, X, tau):
 return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau *
tau))
# Weight or Radial Kernal Bias Function

n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y
:\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])

domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10])
def plot_lwr(tau):
 # prediction through regression
 prediction = [local_regression(x0, X, Y, tau) for x0 in
domain]
 plot = figure(plot_width=400, plot_height=400)
 plot.title.text='tau=%g' % tau
 plot.scatter(X, Y, alpha=.3)
 plot.line(domain, prediction, line_width=2, color='red')
 return plot

show(gridplot([
 [plot_lwr(10.), plot_lwr(1.)],
 [plot_lwr(0.1), plot_lwr(0.01)]]))
```

Output









13. Carry out the performance analysis of classification algorithms on a specific dataset.

Performance Comparison of Multi-Class Classification Algorithms:

This experiment comprises the application and comparison of supervised multi-class classification algorithms to a dataset, which involves the chemical compositions (features) and types (four major types – target) of stainless steels. The dataset is quite small in numbers, but very accurate.

Stainless steel alloy datasets are commonly limited in size, thus restraining applications of Machine Learning (ML) techniques for classification. I explored the potential of 6 different classification algorithms, in the context of a small dataset of 62 samples, for outcome prediction in type classification.

In this article, multi-class classification was analyzed using various algorithms, with the target of classifying the stainless steels using their chemical compositions (15 elements). There are four basic types are of stainless steels and some alloys have very close compositions. Hyperparameter tuning by Grid Search was also applied for Random Forest and XGBoost algorithms in order to observe possible improvements on the metrics. Finally, the performances of the algorithms were compared. After the application of these algorithms, the successes of the models were evaluated with appropriate performance metrics.

The dataset was prepared using "High-Temperature Property Data: Ferrous Alloys".

Wikipedia's definition for multi-class classification is: "In machine learning, multiclass or multinomial classification is the problem of classifying instances into one of three or more classes (classifying instancesinto one of two classes is called binary classification)."

The following algorithms were used for classification analysis:

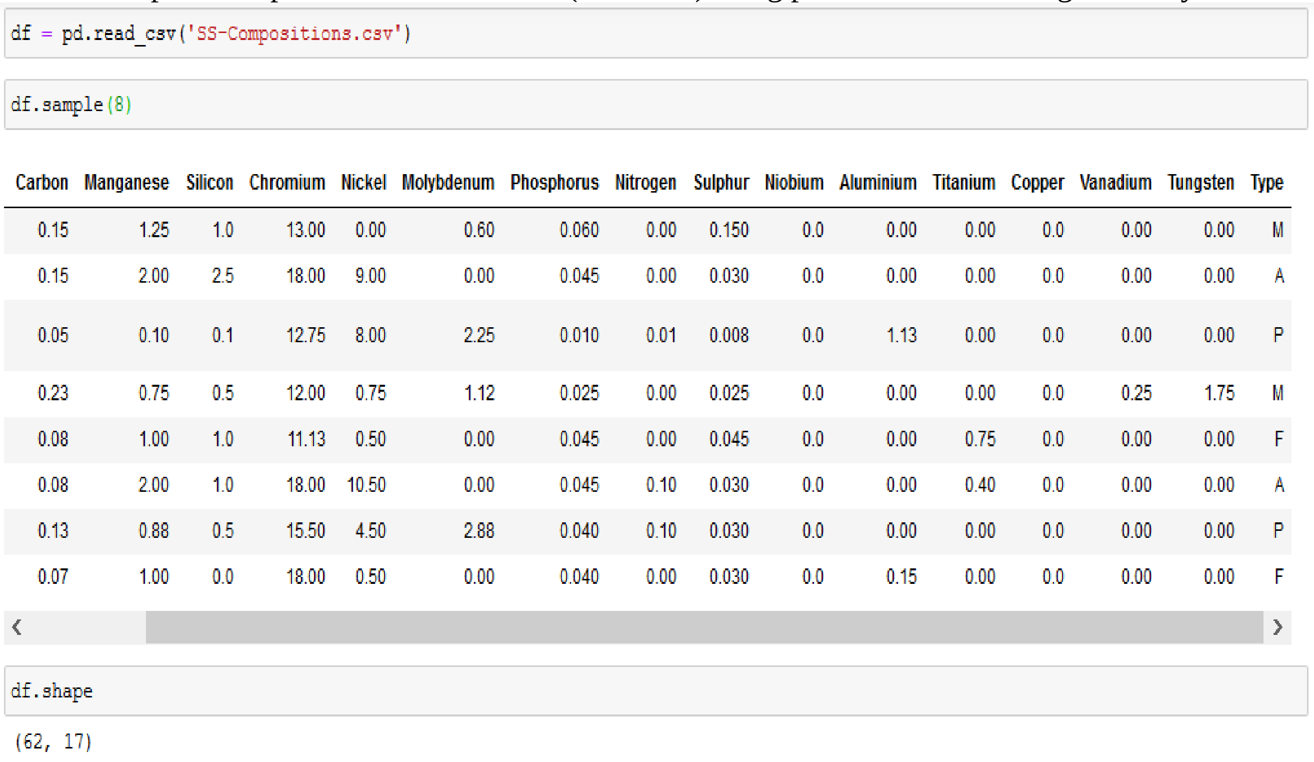
- Decision Tree Classifier,
- Random Forest Classifier,
- XGBoost Classifier,
- Naïve Bayes,
- Support Vector Machines (SVM),
- AdaBoost.

The following issues were the scope of this study:

- Which algorithm provided the best results for multi-class classification?
- Was hyperparameter tuning successful in improving the metrics?
- Reason for poor (if any) metrics.
- Is it safe to use these methods for multi-class classification of alloys.

Data Cleaning

The first step is to import and clean the data (if needed) using pandas before starting the analysis.



```
df = pd.read_csv('SS-Compositions.csv')
```

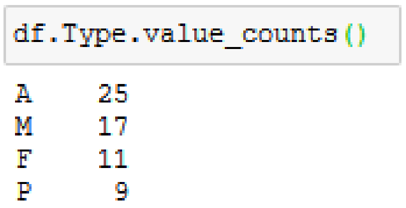
```
df.sample(8)
```

Carbon	Manganese	Silicon	Chromium	Nickel	Molybdenum	Phosphorus	Nitrogen	Sulphur	Niobium	Aluminium	Titanium	Copper	Vanadium	Tungsten	Type
0.15	1.25	1.0	13.00	0.00	0.60	0.060	0.00	0.150	0.0	0.00	0.00	0.0	0.00	0.00	M
0.15	2.00	2.5	18.00	9.00	0.00	0.045	0.00	0.030	0.0	0.00	0.00	0.0	0.00	0.00	A
0.05	0.10	0.1	12.75	8.00	2.25	0.010	0.01	0.008	0.0	1.13	0.00	0.0	0.00	0.00	P
0.23	0.75	0.5	12.00	0.75	1.12	0.025	0.00	0.025	0.0	0.00	0.00	0.0	0.25	1.75	M
0.08	1.00	1.0	11.13	0.50	0.00	0.045	0.00	0.045	0.0	0.00	0.75	0.0	0.00	0.00	F
0.08	2.00	1.0	18.00	10.50	0.00	0.045	0.10	0.030	0.0	0.00	0.40	0.0	0.00	0.00	A
0.13	0.88	0.5	15.50	4.50	2.88	0.040	0.10	0.030	0.0	0.00	0.00	0.0	0.00	0.00	P
0.07	1.00	0.0	18.00	0.50	0.00	0.040	0.00	0.030	0.0	0.15	0.00	0.0	0.00	0.00	F

```
df.shape
```

```
(62, 17)
```

There are 25 austenitic (A), 17 martensitic (M), 11 ferritic (F) and 9 precipitation-hardening (P) stainless steels in the dataset.

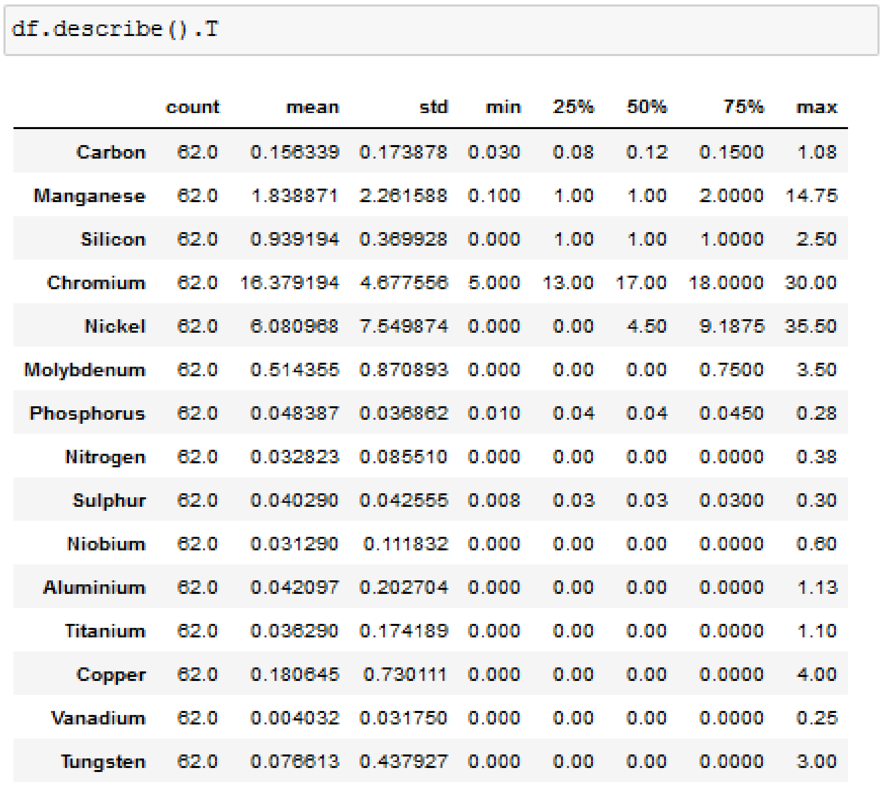


```
df.Type.value_counts()
```

```
A    25
M    17
F    11
P     9
```

There are 62 rows (stainless steels) and 17 columns (attributes) of data. 15 columns cover the chemical composition information of the alloys. The first column is the AISI designation and the last column is the type of the alloy. Our target is to estimate the type of the steel.

Descriptive statistics of the dataset are shown below. Some element percentages are almost stable (Sulphur), but Chromium and Nickel percentages have a very wide range (and these two elements are thedefining elements of stainless steels).

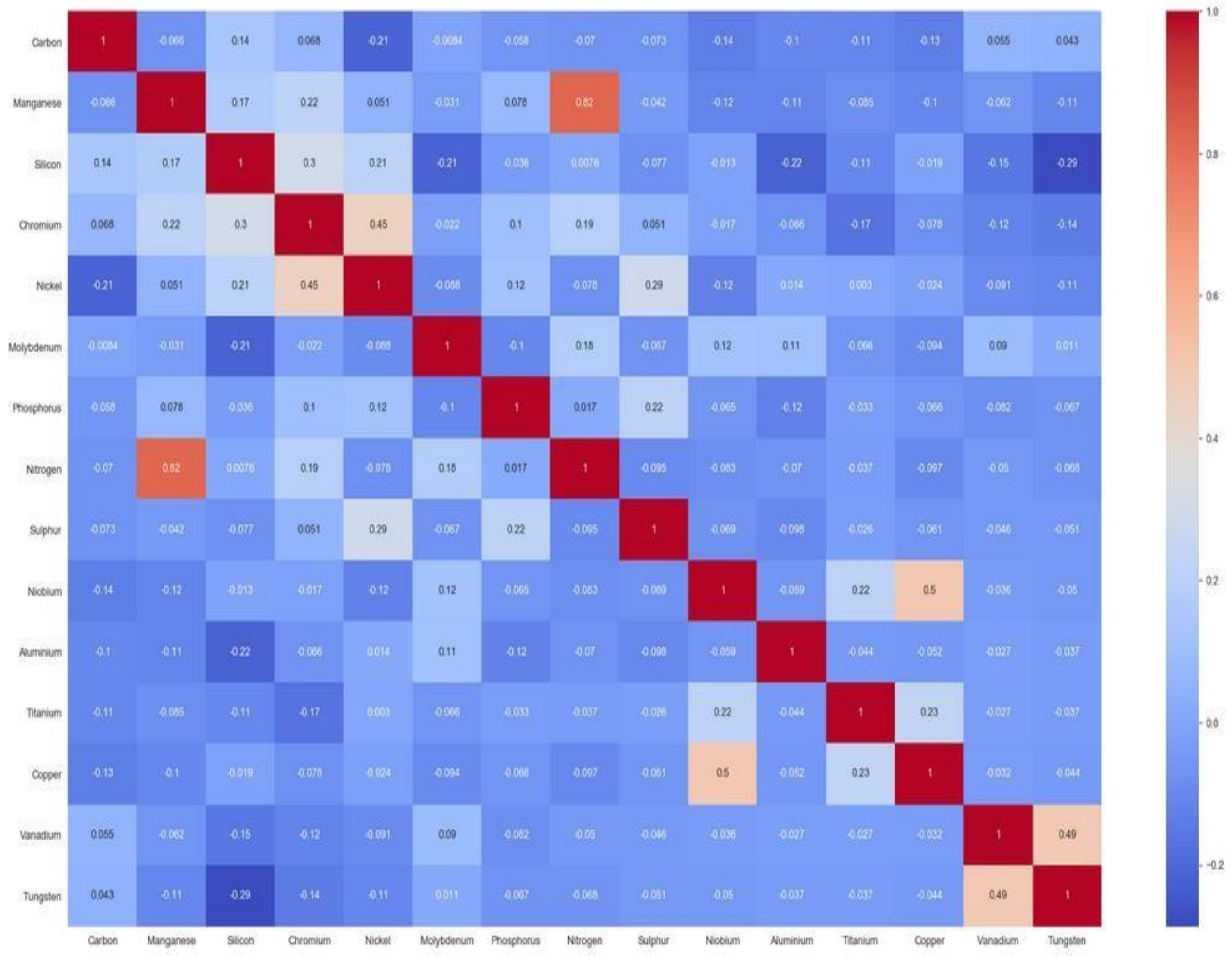


```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Carbon	62.0	0.156339	0.173878	0.030	0.08	0.12	0.1500	1.08
Manganese	62.0	1.838871	2.261588	0.100	1.00	1.00	2.0000	14.75
Silicon	62.0	0.939194	0.369928	0.000	1.00	1.00	1.0000	2.50
Chromium	62.0	16.379194	4.677556	5.000	13.00	17.00	18.0000	30.00
Nickel	62.0	6.080968	7.549874	0.000	0.00	4.50	9.1875	35.50
Molybdenum	62.0	0.514355	0.870893	0.000	0.00	0.00	0.7500	3.50
Phosphorus	62.0	0.048387	0.036862	0.010	0.04	0.04	0.0450	0.28
Nitrogen	62.0	0.032823	0.085510	0.000	0.00	0.00	0.0000	0.38
Sulphur	62.0	0.040290	0.042555	0.008	0.03	0.03	0.0300	0.30
Niobium	62.0	0.031290	0.111832	0.000	0.00	0.00	0.0000	0.60
Aluminium	62.0	0.042097	0.202704	0.000	0.00	0.00	0.0000	1.13
Titanium	62.0	0.036290	0.174189	0.000	0.00	0.00	0.0000	1.10
Copper	62.0	0.180645	0.730111	0.000	0.00	0.00	0.0000	4.00
Vanadium	62.0	0.004032	0.031750	0.000	0.00	0.00	0.0000	0.25
Tungsten	62.0	0.076613	0.437927	0.000	0.00	0.00	0.0000	3.00

Correlation can be defined as a measure of the dependence of one variable on the other one. Two featuresbeing highly correlated with each other will provide too much and useless information on finding the target. The heatmap below shows that the highest correlation is between manganese and nitrogen. Manganese is present in every steel, but nitrogen is not even present in most of the alloys; so, I kept both.

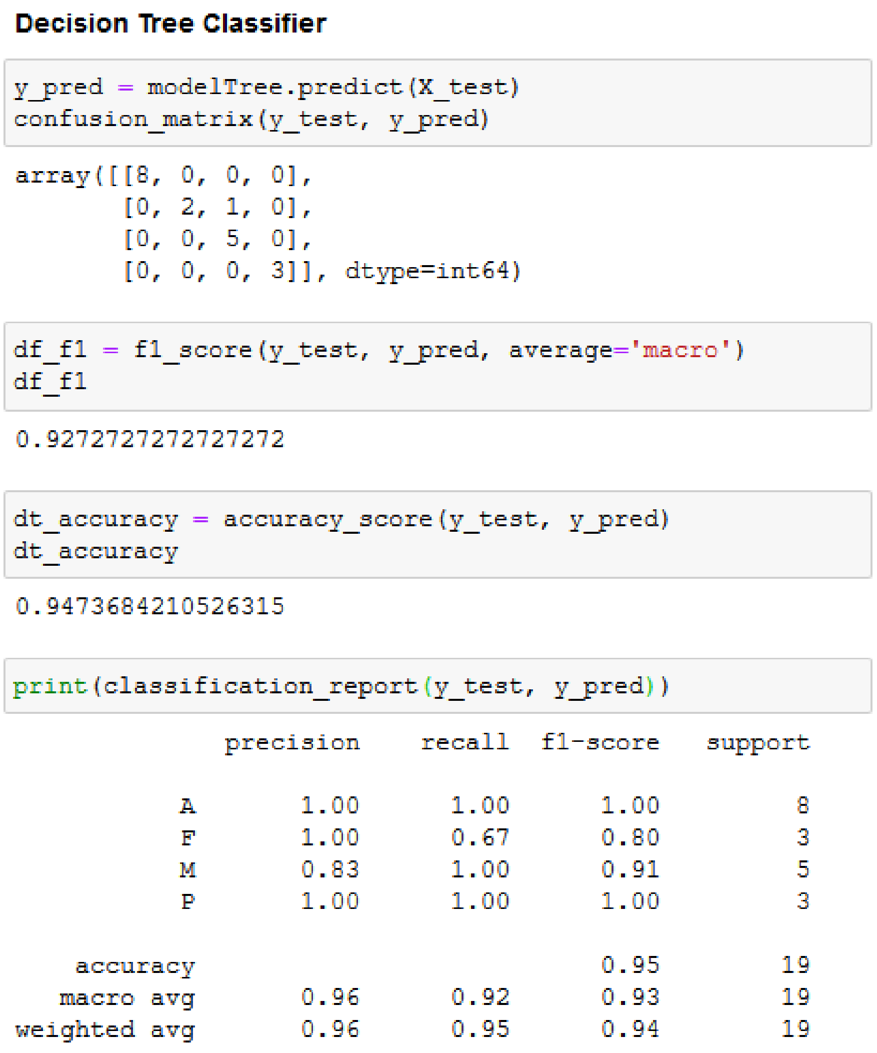
```
plt.figure(figsize=(25, 13))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm');
```



The dataset is clean (there are no NaNs, Dtype are correct), so we will directly start by Train-Test-Split andthen apply the algorithms.

Decision Tree Classifier

First algorithm is the Decision Tree Classifier. It uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves).



Decision Tree Classifier

```
y_pred = modelTree.predict(X_test)
confusion_matrix(y_test, y_pred)
```

```
array([[8, 0, 0, 0],
       [0, 2, 1, 0],
       [0, 0, 5, 0],
       [0, 0, 0, 3]], dtype=int64)
```

```
df_f1 = f1_score(y_test, y_pred, average='macro')
df_f1
```

```
0.9272727272727272
```

```
dt_accuracy = accuracy_score(y_test, y_pred)
dt_accuracy
```

```
0.9473684210526315
```

```
print(classification_report(y_test, y_pred))
```

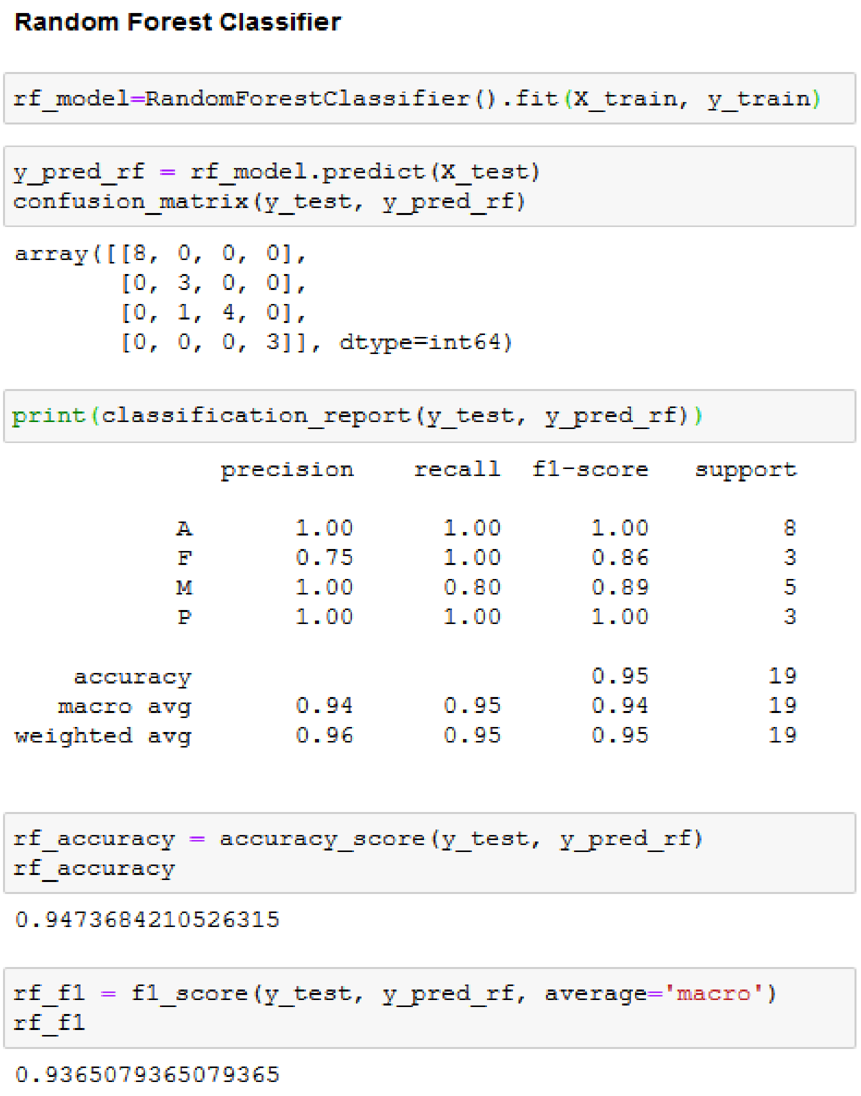
```
              precision    recall  f1-score   support

           A       1.00      1.00      1.00         8
           F       1.00      0.67      0.80         3
           M       0.83      1.00      0.91         5
           P       1.00      1.00      1.00         3

    accuracy                           0.95        19
   macro avg       0.96      0.92      0.93        19
weighted avg       0.96      0.95      0.94        19
```

The results are very good; actually, only one alloy type was classified mistakenly.

Random Forest Classifier

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of theindividual trees.

Random forests generally outperform decision trees, but their accuracy is lower than gradient boostedtrees. However, data characteristics can affect their performance [ref].



Random Forest Classifier

```
rf_model=RandomForestClassifier().fit(X_train, y_train)
```

```
y_pred_rf = rf_model.predict(X_test)
confusion_matrix(y_test, y_pred_rf)
```

```
array([[8, 0, 0, 0],
       [0, 3, 0, 0],
       [0, 1, 4, 0],
       [0, 0, 0, 3]], dtype=int64)
```

```
print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

           A       1.00      1.00      1.00         8
           F       0.75      1.00      0.86         3
           M       1.00      0.80      0.89         5
           P       1.00      1.00      1.00         3

    accuracy                           0.95        19
   macro avg       0.94      0.95      0.94        19
weighted avg       0.96      0.95      0.95        19
```

```
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_accuracy
```

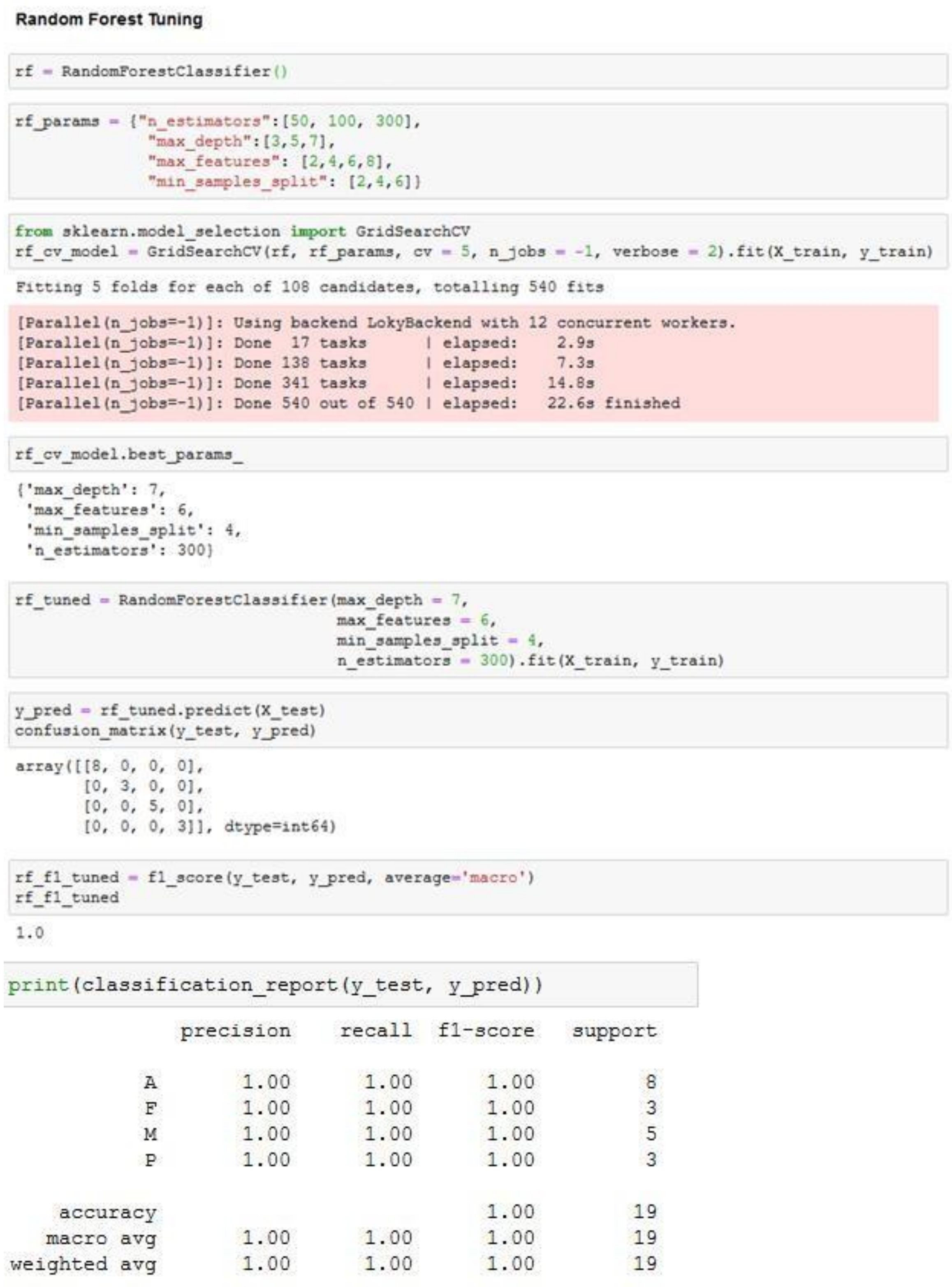
```
0.9473684210526315
```

```
rf_f1 = f1_score(y_test, y_pred_rf, average='macro')
rf_f1
```

```
0.9365079365079365
```

Hyperparameter Tuning with Grid Search

Even though I got satisfactory results with Random Forest Analysis, I applied hyperparameter tuning withGrid Search. Grid search is a common method for tuning a model’s hyperparameters. The grid search algorithm is simple: feed it a set of hyperparameters and the values to be tested for each hyperparameter, and then run an exhaustive search over all possible combinations of these values, training one model foreach set of values. The algorithm then compares the scores of each model it trains and keeps the best one. Here are the results:



Random Forest Tuning

```
rf = RandomForestClassifier()
```

```
rf_params = {"n_estimators":[50, 100, 300],
             "max_depth":[3,5,7],
             "max_features": [2,4,6,8],
             "min_samples_split": [2,4,6]}
```

```
from sklearn.model_selection import GridSearchCV
rf_cv_model = GridSearchCV(rf, rf_params, cv = 5, n_jobs = -1, verbose = 2).fit(X_train, y_train)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:    2.9s
[Parallel(n_jobs=-1)]: Done 138 tasks      | elapsed:    7.3s
[Parallel(n_jobs=-1)]: Done 341 tasks      | elapsed:   14.8s
[Parallel(n_jobs=-1)]: Done 540 out of 540 | elapsed:   22.6s finished
```

```
rf_cv_model.best_params_
```

```
{'max_depth': 7,
 'max_features': 6,
 'min_samples_split': 4,
 'n_estimators': 300}
```

```
rf_tuned = RandomForestClassifier(max_depth = 7,
                                  max_features = 6,
                                  min_samples_split = 4,
                                  n_estimators = 300).fit(X_train, y_train)
```

```
y_pred = rf_tuned.predict(X_test)
confusion_matrix(y_test, y_pred)
```

```
array([[8, 0, 0, 0],
       [0, 3, 0, 0],
       [0, 0, 5, 0],
       [0, 0, 0, 3]], dtype=int64)
```

```
rf_f1_tuned = f1_score(y_test, y_pred, average='macro')
rf_f1_tuned
```

```
1.0
```

```
print(classification_report(y_test, y_pred))
```

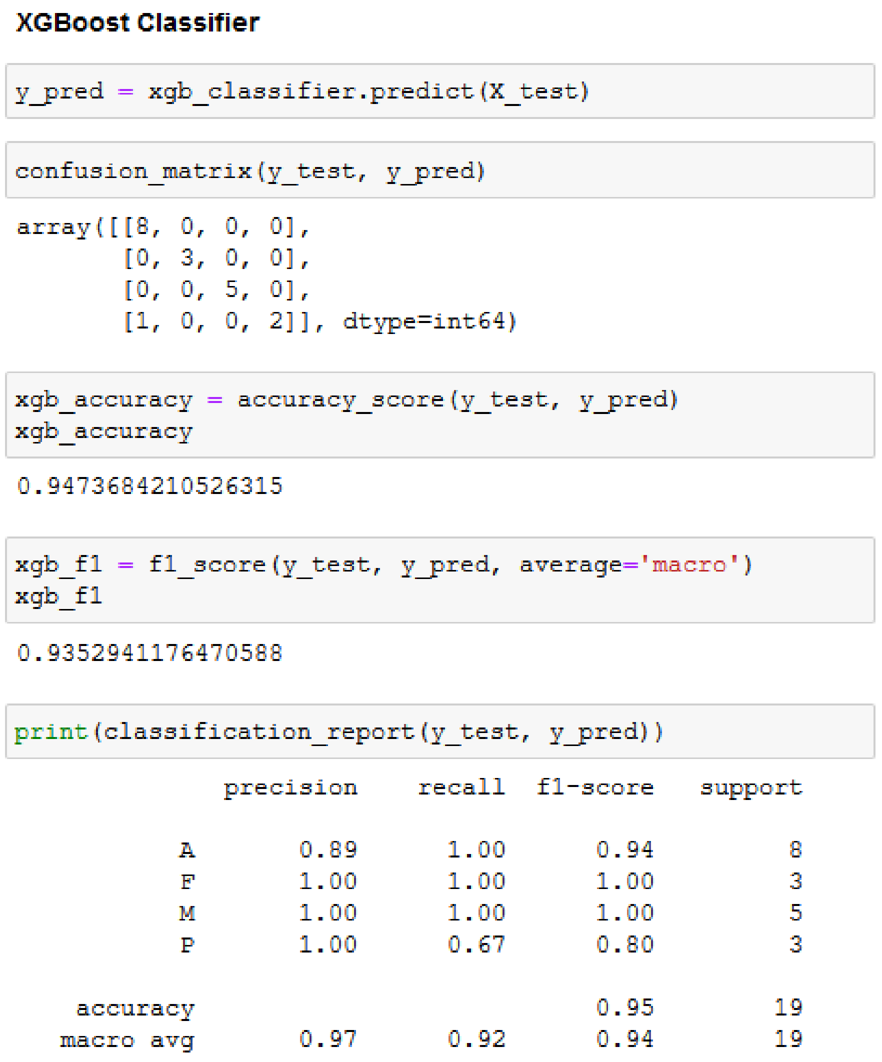
```
              precision    recall  f1-score   support

           A       1.00      1.00      1.00         8
           F       1.00      1.00      1.00         3
           M       1.00      1.00      1.00         5
           P       1.00      1.00      1.00         3

    accuracy                           1.00        19
   macro avg       1.00      1.00      1.00        19
weighted avg       1.00      1.00      1.00        19
```

Hyperparameter tuning with Grid Search took the results to the perfect level – or overfitting.

XGBoost Classifier

XGBoost is well known to provide better solutions than other machine learning algorithms. In fact, since its inception, it has become the "state-of-the-art" machine learning algorithm to deal with structured data.

The results of the XGBoost Classifier provided the best results for this classification study.



XGBoost Classifier

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

```
confusion_matrix(y_test, y_pred)
```

```
array([[8, 0, 0, 0],
       [0, 3, 0, 0],
       [0, 0, 5, 0],
       [1, 0, 0, 2]], dtype=int64)
```

```
xgb_accuracy = accuracy_score(y_test, y_pred)
xgb_accuracy
```

```
0.9473684210526315
```

```
xgb_f1 = f1_score(y_test, y_pred, average='macro')
xgb_f1
```

```
0.9352941176470588
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           A       0.89      1.00      0.94         8
           F       1.00      1.00      1.00         3
           M       1.00      1.00      1.00         5
           P       1.00      0.67      0.80         3

    accuracy                           0.95        19
   macro avg       0.97      0.92      0.94        19
```

Hyperparameter Tuning with Grid Search

Once again, I applied the hyperparameter tuning with Grid Search, even though the results were nearperfect.

Tuning XGBoost

```
xgb = XGBClassifier()
```

```
xgb_params = {"n_estimators": [50, 100, 300],
             "subsample":[0.5,0.8,1],
             "max_depth":[3,5,7],
             "learning_rate":[0.1,0.01,0.3]}
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
```

```
xgb_cv_model = GridSearchCV(xgb, xgb_params, cv = 3,
                            n_jobs = -1, verbose = 2).fit(X_train, y_train)
```

```
Fitting 3 folds for each of 81 candidates, totalling 243 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:    8.2s
[Parallel(n_jobs=-1)]: Done 243 out of 243 | elapsed:    9.9s finished
```

```
xgb_cv_model.best_params_
```

```
{'learning_rate': 0.3, 'max_depth': 3, 'n_estimators': 300, 'subsample': 0.5}
```

```
xgb_tuned = XGBClassifier(learning_rate= 0.3,
                                max_depth= 3,
                                n_estimators= 300,
                                subsample= 0.5).fit(X_train, y_train)
```

```
y_pred = xgb_tuned.predict(X_test)
confusion_matrix(y_test, y_pred)
```

```
array([[8, 0, 0, 0],
       [0, 3, 0, 0],
       [0, 0, 5, 0],
       [0, 0, 0, 3]], dtype=int64)
```

```
xgb_f1_tuned = f1_score(y_test, y_pred, average='macro')
xgb_f1_tuned
```

```
1.0
```

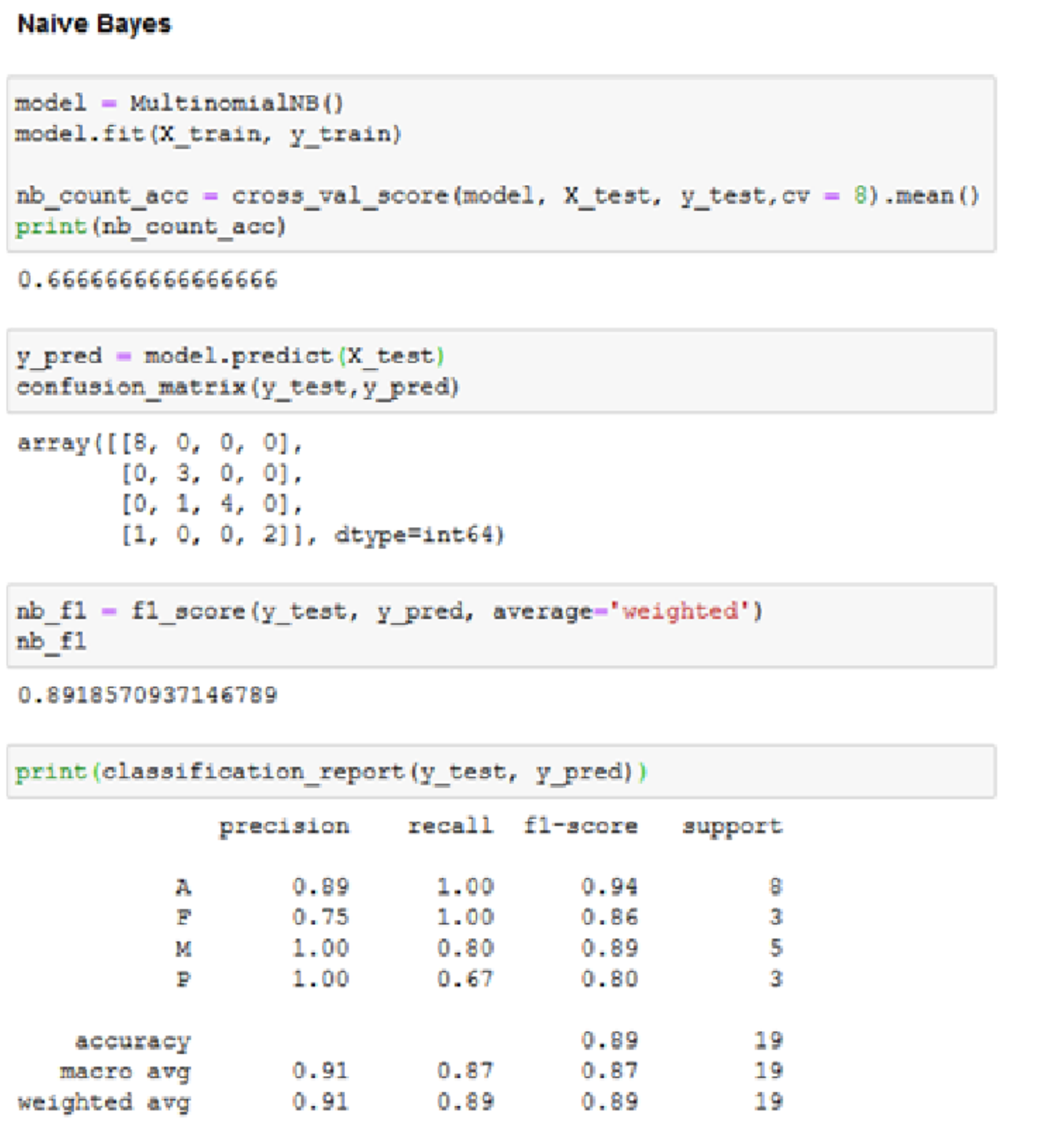
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           A       1.00      1.00      1.00         8
           F       1.00      1.00      1.00         3
           M       1.00      1.00      1.00         5
           P       1.00      1.00      1.00         3

    accuracy                           1.00        19
   macro avg       1.00      1.00      1.00        19
weighted avg       1.00      1.00      1.00        19
```

Naïve Bayes Classifier

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.

The results are shown below:



Naive Bayes

```
model = MultinomialNB()
model.fit(X_train, y_train)

nb_count_acc = cross_val_score(model, X_test, y_test,cv = 8).mean()
print(nb_count_acc)
```

```
0.6666666666666666
```

```
y_pred = model.predict(X_test)
confusion_matrix(y_test,y_pred)
```

```
array([[8, 0, 0, 0],
       [0, 3, 0, 0],
       [0, 1, 4, 0],
       [1, 0, 0, 2]], dtype=int64)
```

```
nb_f1 = f1_score(y_test, y_pred, average='weighted')
nb_f1
```

```
0.8918570937146789
```

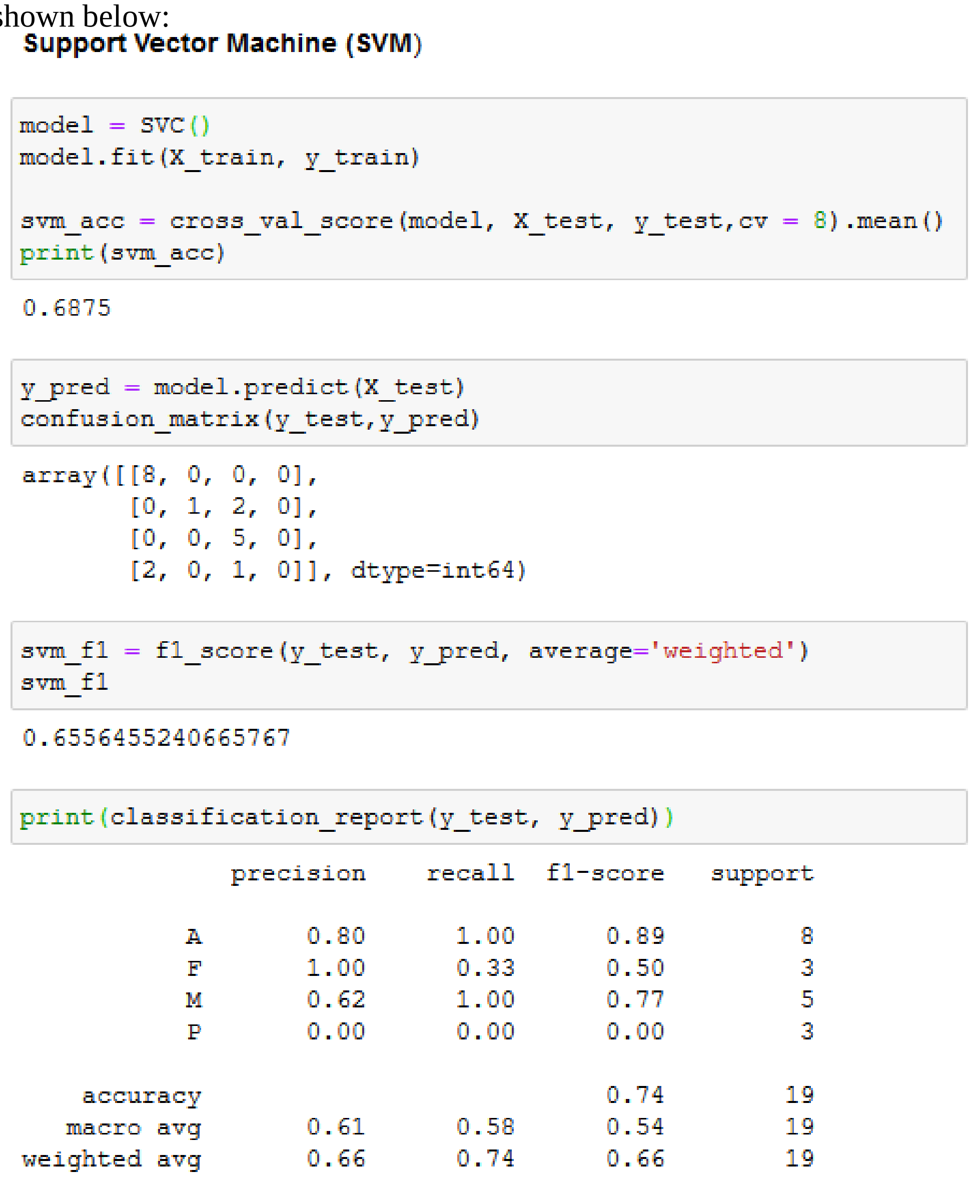
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           A       0.89      1.00      0.94         8
           F       0.75      1.00      0.86         3
           M       1.00      0.80      0.89         5
           P       1.00      0.67      0.80         3

    accuracy                           0.89        19
   macro avg       0.91      0.87      0.87        19
weighted avg       0.91      0.89      0.89        19
```

Support Vector Machines (SVM)

Support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. An SVM maps training examples to points in space to maximize the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

The results are shown below:



Support Vector Machine (SVM)

```
model = SVC()
model.fit(X_train, y_train)

svm_acc = cross_val_score(model, X_test, y_test,cv = 8).mean()
print(svm_acc)
```

```
0.6875
```

```
y_pred = model.predict(X_test)
confusion_matrix(y_test,y_pred)
```

```
array([[8, 0, 0, 0],
       [0, 1, 2, 0],
       [0, 0, 5, 0],
       [2, 0, 1, 0]], dtype=int64)
```

```
svm_f1 = f1_score(y_test, y_pred, average='weighted')
svm_f1
```

```
0.6556455240665767
```

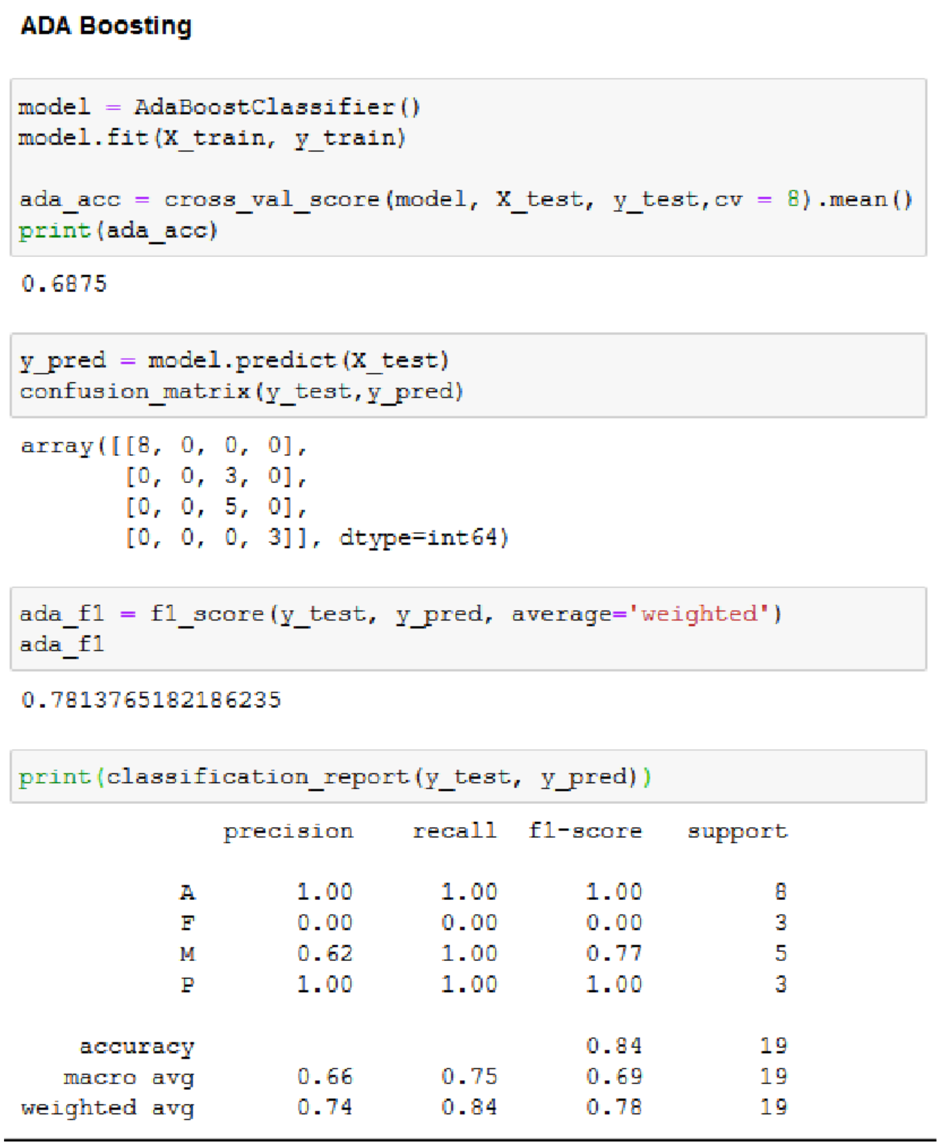
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           A       0.80      1.00      0.89         8
           F       1.00      0.33      0.50         3
           M       0.62      1.00      0.77         5
           P       0.00      0.00      0.00         3

    accuracy                           0.74        19
   macro avg       0.61      0.58      0.54        19
weighted avg       0.66      0.74      0.66        19
```

<u>AdaBoost</u>

AdaBoost, short for Adaptive Boosting, is a machine learning meta-algorithm, which can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier.

The results are shown below:



ADA Boosting

```
model = AdaBoostClassifier()
model.fit(X_train, y_train)

ada_acc = cross_val_score(model, X_test, y_test,cv = 8).mean()
print(ada_acc)
```

```
0.6875
```

```
y_pred = model.predict(X_test)
confusion_matrix(y_test,y_pred)
```

```
array([[8, 0, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 5, 0],
       [0, 0, 0, 3]], dtype=int64)
```

```
ada_f1 = f1_score(y_test, y_pred, average='weighted')
ada_f1
```

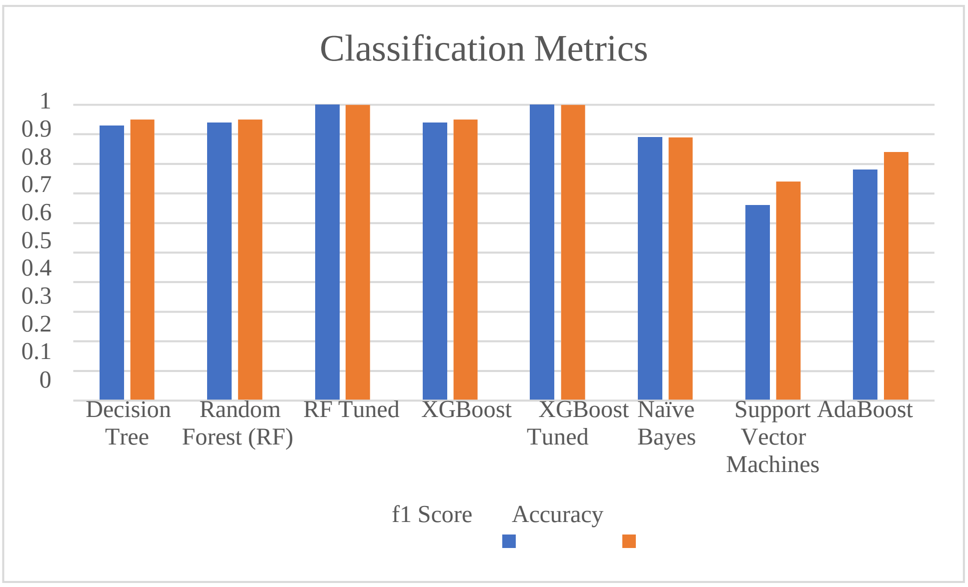
```
0.7813765182186235
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           A       1.00      1.00      1.00         8
           F       0.00      0.00      0.00         3
           M       0.62      1.00      0.77         5
           P       1.00      1.00      1.00         3

    accuracy                           0.84        19
   macro avg       0.66      0.75      0.69        19
weighted avg       0.74      0.84      0.78        19
```

Conclusion

In this experiment, we used six different Supervised Machine Learning (Classification) algorithms with the purpose of classifying four types of stainless steels (multi-class) according to their chemical compositions comprised of 15 elements in the alloy. The dataset included 62 alloys; which made it a small, but a very accurate dataset (all the information was taken from ASM International Sources (formerly known as American Society of Metals)).



The analysis provides evidence that:

- Considering the f1 scores, Random Forest and XGBoost methods produced the best results (0.94).
- After hyperparameter tuning by Grid Search, RF and XGBoost f1 scores jumped to 100 %.
- Multiple tries of the same algorithm resulted different results with a huge gap – most probably due to the limited data size.
- The poorest f1 scores were mostly for the classification of the types that have the least-numberedgroups; which were ferritic and precipitation-hardened steels.

- Finally, test classification accuracy of 95% achieved by 3 models (DT, RF and XGBoost) and 100% by 2 tuned models demonstrates that the ML approach can be effectively applied to steel classification despite the small number of alloys and heterogeneous input parameters (chemical compositions). Based on only 62 cases, the models achieved a very high level of performance for multi-class alloy type classification.

14. Case Study: You are owing a supermarket mall and through membership cards , you have some basic data about your customers like Customer ID, age, gender, annual income, and spending score. Spending Score is something you assign to the customer based on your defined parameters like customer behaviour and purchasing data.

Problem Statement

By being the managing director of your Supermarket Mall, You wanted to understand the customers like who can be easily converge [Target Customers] so that the sense can be given to marketing team and plan the strategy accordingly.
After carrying out this case study, answer the questions given below.
1- How to achieve customer segmentation using machine learning algorithm (KMeans Clustering) in Python in simplest way.
2- Who are your target customers with whom you can start marketing strategy [easy to converse]
3- How the marketing strategy works in real world?

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list
all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets
preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
outside of the current session
```

```
/kaggle/input/customer-segmentation-tutorial-in-python/Mall_Customers.csv
```

Importing the Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
```

Data Collection and Analysis

```
# loading the data from csv file to a pandas Dataframe

df = pd.read_csv('../input/customer-segmentation-tutorial-in-
python/Mall_Customers.csv')

# first five rows of the Dataframe

df.head()
```

```
   CustomerID  Gender  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
```

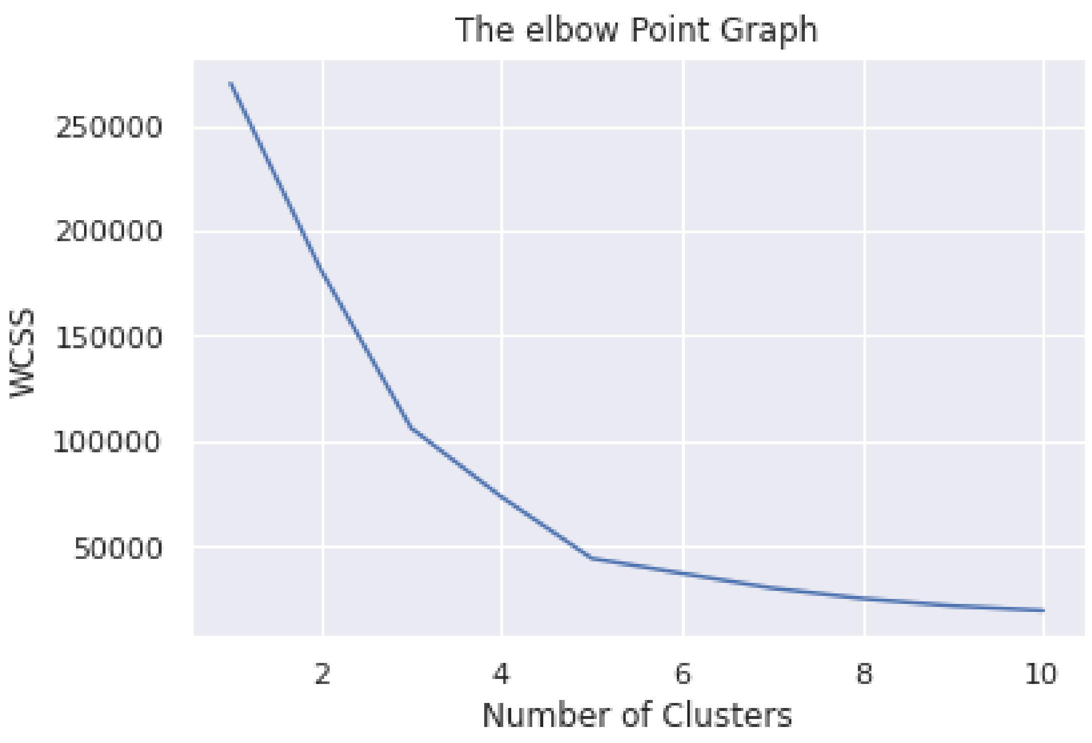
```
# finding the number of rows and columns

df.shape

(200, 5)

# getting information of the data

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   CustomerID              200 non-null    int64 
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64 
 3   Annual Income (k$)      200 non-null    int64 
 4   Spending Score (1-100)  200 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 7.9+ KB

# checking missing values

df.isnull().sum()

CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

choosing Only Annual Income & Spending Score Column

```
X = df.iloc[:,[3,4]].values
```

Choosing number of clusters

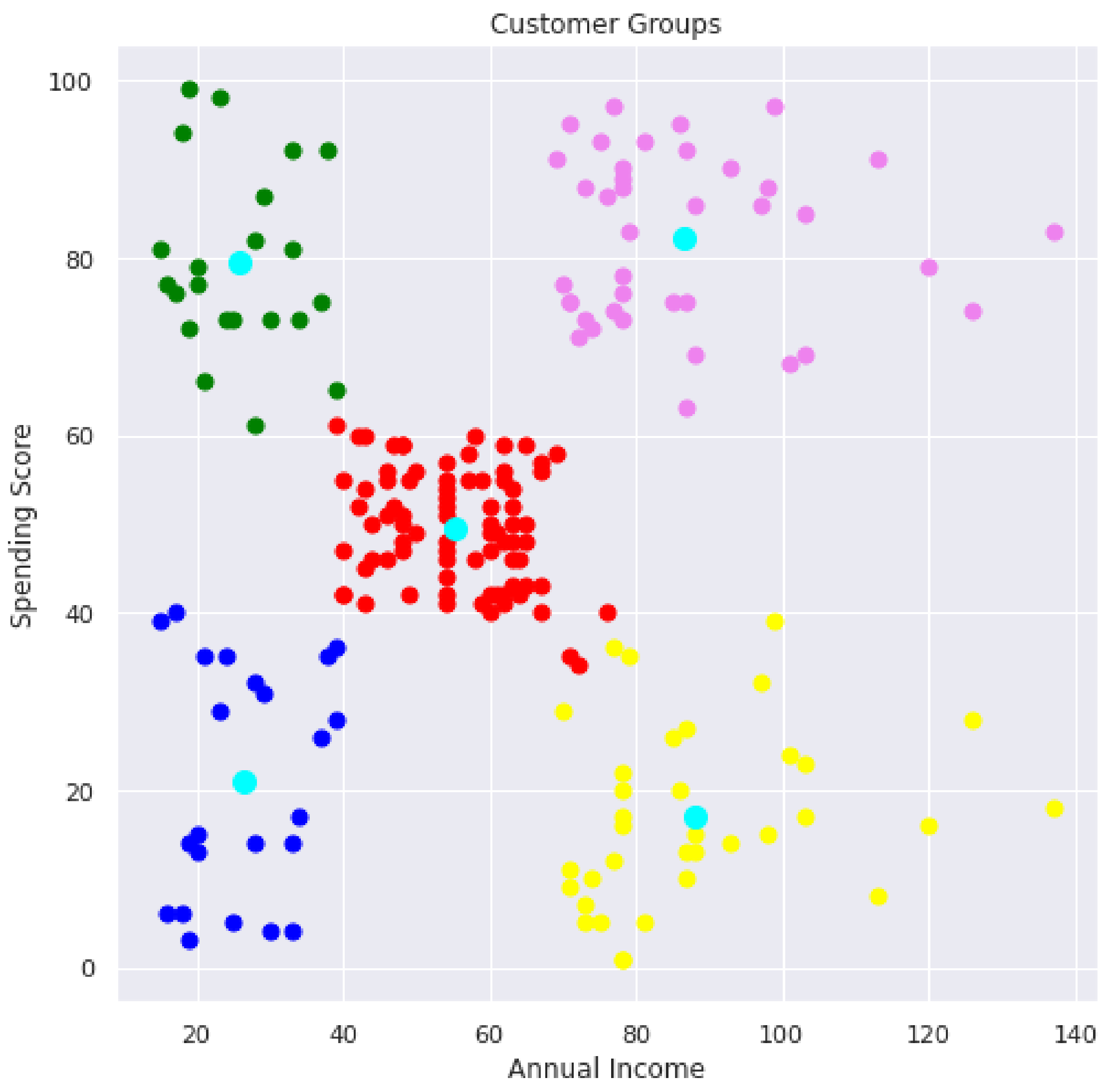
```
# WCSS - Within clusters sum of squares

# finding  wcss vlaue for different number of cluster

wcss = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i,init='k-means++',random_state=30)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# plot an elbow graph

sns.set()
plt.plot(range(1,11), wcss)
plt.title('The elbow Point Graph')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



- Optimum number of cluster = 5

Training the KMean Cluster Model

```
kmeans = KMeans(n_clusters=5,init='k-means++',random_state=30)

#return a label for each data point based on their cluster

Y = kmeans.fit_predict(X)

print(Y)

[4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4
 0 4 0 4 0 4 1 4 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 3 2 3 1 3 2 3 2 3 1 3 2 3 2 3 2 3 2 3 1 3 2 3 2 3
 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2
 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3]

# Plotting all the clusters and their Centroids

plt.figure(figsize=(8,8))
plt.scatter(X[Y==0,0],X[Y==0,1],s=50,c='green',label='Cluster 1')
plt.scatter(X[Y==1,0],X[Y==1,1],s=50,c='red',label='Cluster 2')
plt.scatter(X[Y==2,0],X[Y==2,1],s=50,c='yellow',label='Cluster 3')
plt.scatter(X[Y==3,0],X[Y==3,1],s=50,c='violet',label='Cluster 4')
plt.scatter(X[Y==4,0],X[Y==4,1],s=50,c='blue',label='Cluster 5')

# plot the centroids
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],s=100,
c='cyan',label='Centroids')
plt.title('Customer Groups')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.show()
```



Observations

- As we can see in the above figure that some person who has greater annual income spends less but some person has greater annual income spend more in blue color
- Also, person has less income with less spending in color violet.
- person has greater income spent more in color green.

we can give this information to different teams of the company they can make offers accordingly.