**Experiment 1:** Write steps install and configure git, create a repository and record changes, staging and commit changes, view history of changes. Creating and Switching to New Branches , Switching between branches , Merging local branches together .   
  
**Answer:**

**What is Git?**

Git is a distributed, open-source version control system. It enables developers and data scientists to track code, merge changes and revert to older versions. It allows you to synchronize changes with a remote server. Due to its flexibility and popularity, Git has become an industry standard as it supports almost all development environments, command-line tools, and operating systems.   
  
**Git installation and setup**install and set up Git, you can follow these steps:   
  
 **Download Git**: Visit the official Git website at https://git-scm.com/downloads. The website should automatically detect your operating system and provide the appropriate download link. Click on the download link to start the download.

 **Run the Installer**: Once the download is complete, run the installer file you downloaded.

 **Complete the Installation:** Once you've made your desired selections, proceed with the installation. The installer will copy the necessary files and complete the setup process.

 **Verify the Installation**: After the installation is complete, open a command prompt or terminal window and run the following command to verify that Git is installed correctly:

git --version

 You should see the Git version information displayed in the command prompt or terminal, indicating that Git is installed and ready to use.

 **Configure Git**: Before using Git, it's recommended to configure your name and email address, which will be associated with your commits. Open the “Git Bash” Run the following commands, replacing the placeholders with your own name and email:

git config --global user.name "Your Name" git config --global user.email "your\_email@example.com"

 **NOTE**: These global configuration settings will be used by default for all your Git repositories. If you want to configure repository-specific settings, you can navigate to the repository's directory and use the same commands without the --global flag.

**Basic local Git operations Creating a repository:**

 Create a project folder with your name in the “D” directory.

mkdir your\_name

 Navigate to that directory where you want to create the repository using the Git Bash.

cd your\_name

 Run the following command to initialize a new Git repository:

git init

 “git init” creates an empty Git repository in the current directory.

 Optionally, you can add files to the repository by copying them into the project directory or creating new files within it.

**Making and Recording Changes:**

 Make changes to your files: Use your preferred text editor (ex. nano) or any other appropriate tool to modify the files in your working directory. Add, delete, or modify the content as needed.

nano readme.txt   
  
 Check the status of your repository: Run the following command to see the current state of your repository and the changes you have made:

git status

 “git status” command provides information about the modified files, untracked files, and staged changes.

 **Note**: Red color files are untracked files, Green color files are staged files.

**Staging and Committing Changes: Staging** allows you to carefully select and prepare the changes you want to include in a commit, while **committing** permanently records those changes in the repository's history.

 Stage changes: Before committing your changes, you need to stage them. Use the following command to stage specific files:

git add file\_name

 file\_name is the names of the files you want to stage.

 To stage multiple file at once, you can use the following command.

git add file1\_name file2\_name

 To stage all files at once, You can use the following command.

git add .

 Verify the staged changes: Run “git status” again to ensure that the desired changes are staged and ready to be committed. The staged files are in Green color.

 Commit changes: To commit staged files to record in the repository. Use the following command.

git commit -m "Commit\_message"

 Replace "Commit\_message" with a descriptive message that summarizes the changes you made in the commit. The commit message helps others understand the purpose and impact of the changes.

**Viewing the history of all the changes**

 To view the commit history and verify that your commit was recorded, use the following command:

git log

 This command displays a list of commits, including the commit message, author, date, and a unique commit hash.   
  
  
**Branching** is a fundamental and powerful feature of Git that allows for parallel development, isolating changes, and managing different lines of development within a repository. Each branch in Git represents an independent line of development, enabling multiple contributors to work on different features or bug fixes simultaneously.   
  
**Merging** is used to combine changes from different branches into the current branch. It allows you to integrate the changes made in one branch into another branch, incorporating new features, bug fixes, or updates from one branch to another.   
  
**Creating and Switching to New Branches:**

 Run the following command to see the currently active branch.

git branch

 The active branch will be indicated with an asterisk (\*) next to its name.

 Create a new branch: Use the following command to create a new branch:

git branch <new-branch-name>

 Replace <new-branch-name> with the desired name for your new branch.

 To switch to the newly created branch, use the following command:

git checkout <new-branch-name>

 Replace <new-branch-name> with the name of your new branch.

 Alternatively, you can combine branch creation and switching into a single command by using the -b flag:

git checkout -b <new-branch-name>

 This command creates a new branch and immediately switches to it.

 Verify the branch switch: Run git branch again to confirm that you have switched to the new branch. The active branch should now have an asterisk (\*) next to its name.

**Switching between branches**

 Run the following command to view all the branches in your repository:

git branch

 The branch with an asterisk (\*) next to its name indicates the currently active branch.

 Use the following command to switch to an existing branch:

git checkout <branch-name>

 Replace <branch-name> with the name of the branch you want to switch to.

 For example, if you want to switch to a branch named "feature-branch," run:

git checkout feature-branch

 Git will switch you’re working directory and current branch to the specified branch.

 After running the checkout command, you can confirm that you have successfully switched to the desired branch by running following command.

git branch

 The branch with an asterisk (\*) should now reflect the branch you switched to.   
  
**Merging local branches together** Before merging branches, make sure you are on the branch that you want to merge other branches into. You can switch to the desired branch using the git checkout command. For example:  
  
git checkout main

 This command switches to the main branch.

 Merge the branches: Use the following command to merge another branch into the currently active branch:

git merge <branch-name>

 Replace <branch-name> with the name of the branch you want to merge into the current branch. For example, to merge a branch named feature-branch into the main branch, run:

git merge feature-branch

 Git will attempt to automatically merge the changes from the specified branch into the current branch. If there are no conflicts, the merge will be performed automatically. If conflicts occur, Git will indicate the conflicting files and you will need to manually resolve the conflicts.

 Commit the merge: Once the conflicts have been resolved (or if there were no conflicts), you need to commit the merge. Run the following command to create a merge commit:

git commit -m "Merge branch-name into current-branch"

 Provide an appropriate commit message that describes the merge operation.

 After committing the merge, you can verify that the branches have been successfully merged by running “git log”.

**Experiment 2:** Write steps to creation and configuration GitHub account, create a repository in Git hub and clone it to your local git, create and switch to a new branch, make changes, commit changes, Push changes to Git Hub.   
  
**Answer:**  
**GitHub:** is a web-based platform designed to help developers collaborate on software projects. It primarily focuses on version control, allowing multiple people to work on the same codebase simultaneously and keep track of changes over time

 **Repositories (Repos):** A repository is like a project folder that contains all the files, folders, and version history for your project. It's where your codebase lives and is managed. Repositories can be created on GitHub and cloned (downloaded) to your local machine for work.

 **Commits:** A commit represents a snapshot of your code at a specific point in time. It captures a set of changes you've made to your files. Commits have unique identifiers (hashes) and are accompanied by commit messages describing the changes.

 **Branches:** A branch is a separate line of development within a repository. The default branch (often named "main" or "master") is where the main codebase resides. You can create new branches to work on features, bug fixes, or experiments without affecting the main branch.

 **Pull Requests (PRs):** A pull request is a request to merge changes from one branch (the "source" branch) into another (the "target" branch). PRs facilitate code review, discussion, and collaboration among team members before changes are integrated.

 **Merge:** Merging combines the changes from one branch into another. It's typically used to incorporate finished features or fixes into the main codebase. Merge conflicts may arise if changes in different branches affect the same lines of code.

 **GitHub Actions**: GitHub Actions is a powerful automation platform that enables you to define workflows (series of steps) for tasks like building, testing, and deploying your code. Workflows can be triggered by events such as pushes, pull requests, or scheduled intervals.

**Account creation and configuration:**

 **Visit GitHub**: Go to the GitHub website by typing "github.com" in your web browser's address bar.

 **Sign Up**: On the GitHub homepage, you'll find a "Sign Up" button. Click on it.

 **Provide Information**: You will be prompted to provide the following information:

o Email address: Use a valid email address to receive GitHub notifications and updates.

o Password: Create a strong password to secure your account.

o Username: Choose a unique username that will be your GitHub handle. This will be used in your GitHub URL (github.com/username).

o Type “n” for this question. “Would you like to receive product updates and announcements via email?”

 **Complete the CAPTCHA**: GitHub might require you to complete a CAPTCHA to verify that you're not a bot. click on “create account”.

 **Verification**: Verify your email address by clicking on the link sent to your email inbox from GitHub. or Verification code is sent to your email id, copy and paste that in github.

**Generating an SSH key and adding it to your GitHub**Generating an SSH key and adding it to your GitHub account allows you to securely authenticate and interact with your GitHub repositories without needing to enter your username and password each time. Here's how you can generate an SSH key and add it to your GitHub account:   
  
**Generating an SSH Key**:

 Open Git Bash to generate an SSH key.

 Generate a New SSH Key: Enter the following command, replacing <your\_email@example.com> with the email associated with your GitHub account. This command generates an SSH key pair.

ssh-keygen -t ed25519 -C "<your\_email@example.com>"

 You can also use RSA instead of ed25519, but ed25519 is recommended for better security.

 **Choose a Location and Passphrase**: The command will prompt you to choose a location to save the key pair and set a passphrase. Press Enter to accept the default location.

 Press enter one more time.

 **View Your Public Key**: Once the key pair is generated, you can view the public key by running the following command:

cat ~/.ssh/id\_ed25519.pub   
**OUTPUT**: ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIHMvMPYOyR2Py/4Y7bnatzpm/jCZjc+F5Oh95LuMWiln fifthcse@gmail.com

 Copy the entire output of this command. This is your public key.

**Adding SSH Key to GitHub**:

 Log in to Your GitHub Account: Open your web browser and log in to your GitHub account.

**Access SSH and GPG Keys Settings**:

 Click on your profile icon (your avatar) in the top-right corner.

 Select "Settings" from the dropdown menu.

 In the left sidebar, click on "SSH and GPG keys."

**Add New SSH Key**:

 Click on the "New SSH key" button.

 Give your key a meaningful title in the "Title" field. This title is for your reference.

 Paste the copied public key into the "Key" field. Make sure you paste the entire key.

**Add SSH Key**:

 Click the "Add SSH key" button.

 GitHub will prompt you to enter your account password to confirm the addition of the SSH key.

**Verify and Use SSH Key**:

 To verify that your SSH key is properly set up, you can run the following command in your Terminal or Git Bash:

ssh -T [git@github.com](mailto:git@github.com)   
**OUTPUT**: The authenticity of host 'github.com (20.207.73.82)' can't be established. ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU. This key is not known by any other names Are you sure you want to continue connecting (yes/no/[fingerprint])? yes

 You should receive a message indicating successful authentication.

**Create and push to repositories: (This code you will get from Github repositiry)**$ git remote add origin git @ github.com:FifthCSE/project.git   
$ git branch -M main   
$ gitpush -u origin main   
  
**Versioning** Versioning in GitHub typically refers to the practice of using version control systems (VCS), such as Git, to manage the history and changes of your codebase. GitHub is a popular platform for hosting Git repositories, making it easier for teams to collaborate and manage their code versions. Here is a step-by-step workflow for versioning in GitHub, along with example code.   
  
**Create a Repository**:   
 Start by creating a new repository on GitHub. Let's call it "sample\_project".

 Creating a repository in GitHub is a straightforward process. Follow these steps to create a new repository:

 **Sign In to GitHub**: If you don't already have a GitHub account, sign up at https://github.com. If you have an account, sign in. Once you're signed in, you'll land on your GitHub dashboard..

 **Create a New Repository**: Click on "New" button on the left side to start creating a new repository.

 **Configure Repository Settings**: You'll be taken to the "Create a new repository" page. Here, you'll need to provide some information:

 **Repository Name**: Choose a name for your repository. This should be a short, descriptive name for your project. (in our example “sample\_project”)

 **Description**: Optionally, provide a brief description of your project.

 **Visibility**: Choose whether your repository should be public (visible to everyone) or private (accessible only to you and collaborators you invite).

 **Initialize this repository with**: You can choose to initialize the repository with a README file, which is helpful for providing initial documentation about your project. (Check that check box).

 Ignore remaining options.

 **Create Repository**: Once you've configured the settings, click the green "Create repository" button.

 **Repository Created**: Congratulations! Your repository has been created. You'll be taken to the main page of your repository, where you can start adding code, documentation, and other project files.

**Clone the Repository**:

 Clone the repository to your local machine using the git clone command:

 To Copy the command from your repository, click on “Code” button, select “HTTP” option, click copy. And paste it in the gitbash, where you want to clone the repository.

$ git clone **https://github.com/** **FifthCSE /sample\_project.git   
Create and Switch to a Branch**:

 Create a new branch for your feature or change. Let's create a branch called "feature/add-function" and switch to it:

$ cd sample\_project   
$ git checkout -b arithmatic/add\_function   
  
**Make Changes**:

 Create a simple Python function in a file, e.g., main.py:

$ nano main.py   
  
Type the below code in nano editor.   
Python code:   
# main.py   
def add(a, b):   
 return a + b   
result = add(5, 3)   
print("Result:", result)

 To Save and eixt from nano Editor:   
CtrlX   
Type Y   
Enter   
  
**Commit Changes**:

 Stage and commit your changes with a meaningful commit message:   
$ git status   
$ git add main.py   
$ git status   
$ git commit -m "Add a basic add function"   
  
**Push Changes to GitHub**:

 Push your changes to the remote repository on GitHub:

$ git push origin feature/add-function   
  
**Experiment 3:** Lee decides to walk 10000 steps every day to combat the effect that lockdown has had on his body’s agility, mobility, flexibility and strength. Consider the following data from fitness tracker over a period of 10 days

|  |  |
| --- | --- |
|  | 1.Represent the above data in a 10x2 array. In each row, the first element should contain day number and second element should contain steps walked.  2. Lee notices that the tracker’s battery dies every day at 7 pm. Lee discovers that on an average, he walks 2000 steps every day after 7 pm. Perform an appropriate operation on your array to add 2000 steps to all the observations.  3. Write a program that returns the steps walked if the steps walked are more than 9000.  4. Print an array containing steps walked in sorted order. |

# 1) Represent the above data in a 10x2 array:

import numpy as np

import pandas as pd

data = np.array([[1, 6012], [2, 7079], [3, 6886],

[4, 7230], [5, 4598], [6, 5564],

[7, 6971], [8, 7763], [9, 8032], [10, 9569]])

print(data)  
  
  
# 2.Perform an appropriate operation on your array to add 2000 steps to all the observations.

df = pd.DataFrame(data, columns =['Day Number', 'Steps Walked'])

print("\n Steps after adding 2000 steps to all observations")

df["Steps Walked"] = df["Steps Walked"] + 2000

print(df)  
  
# iii) Write a program that returns the steps walked if the steps walked are more than 9000:

steps\_walked\_more\_than\_9000 = df[df['Steps Walked'] > 9000]

print(steps\_walked\_more\_than\_9000)  
  
# iv) Print an array containing steps walked in sorted order:

sorted\_steps\_walked = np.sort(steps\_walked)

print(sorted\_steps\_walked)

**Experiment 4:** Create a dataframe with following data.



a) Make a pivot table which shows average salary of each type of employee for each department.

b) Make a pivot table which shows the sum and mean of the salaries of each type of employee and the number of employees of each type.

c) Make a pivot table which shows standard deviation for salary column.   
  
  
import pandas as pd  
data = {'Employee': ['Emp1', 'Emp2', 'Emp3', 'Emp4', 'Emp5', 'Emp6', 'Emp7', 'Emp8', 'Emp9', 'Emp10'],'Department': ['HR', 'IT', 'IT', 'HR', 'HR', 'IT', 'IT', 'HR', 'IT', 'IT'],'Type': ['Manager', 'Manager', 'Developer', 'Developer', 'Analyst', 'Analyst', 'Developer', 'Analyst', 'Developer', 'Analyst'],'Salary': [50000, 60000, 55000, 52000, 45000, 48000, 55000, 49000, 60000, 51000]}  
df = pd.DataFrame(data)

**a) Make a pivot table which shows average salary of each type of employee for each department:**

avg\_salary\_pivot = df.pivot\_table(values='Salary', index='Department', columns='Type', aggfunc='mean')  
print(avg\_salary\_pivot)

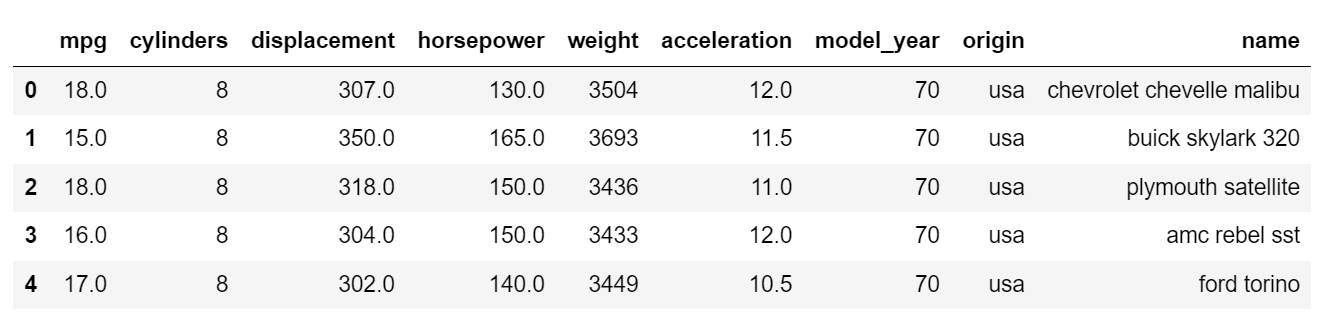
**b) Make a pivot table which shows the sum and mean of the salaries of each type of employee and the number of employees of each type:**

summary\_pivot=df.pivot\_table(values='Salary',index='Type', aggfunc=['sum','mean','count'])print(summary\_pivot)

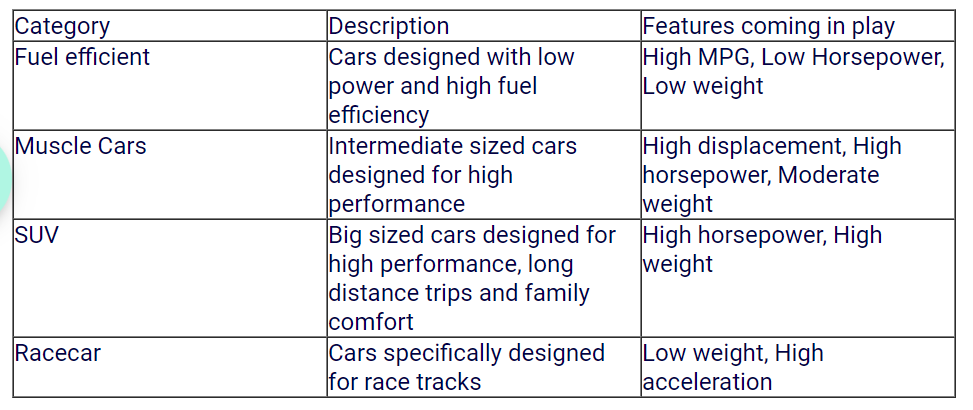
**c) Make a pivot table which shows standard deviation for salary column:**

std\_pivot = df.pivot\_table(values='Salary', index='Type',aggfunc='std')  
print(std\_pivot)

**Experiment 5:** Perform the following operations on Car manufacturing company dataset auto\_mpg.csv given below using pandas



1. statistical details of dataset
2. Get all cars with 8 cylinders
3. Retrieve details of all the cars built in year 72.
4. Retrieve details of all the cars built in Japan having 6 cylinders
5. XYZ Custom Cars want to categorize cars in different categories as follows:



1. XYZ Custom cars want the data sorted according to the number of cylinders.
2. There is a requirement in which the cars that have lowest acceleration must be assessed. It is also to be checked that which cars have higher horsepower despite having lower acceleration.
3. XYZ custom cars wants to know about minimum and maximum of all the numerical columns.
4. XYZ custom cars want to know the number of cars manufactured in each year.
5. Some senior engineers in XYZ custom cars want to understand about the effect of model year and number of cylinders on horsepower.
6. The engineers at XYZ Custom Cars want to know about the relationship between model year and acceleration of cars.
7. The engineers at XYZ Custom Cars want to know the frequency distribution of different number of cylinders across different years.
8. The engineers at XYZ custom cars want to know the mean of all the numerical attributes of cars for each year.

**Experiment 6:** Write python code to demonstrate Map, apply(), reduce(), lambda() & filter() functions.

**# map() function applies a given function to all the items in an input list.**

arr = [1, 2, 3, 4, 5]

arr = list(map(lambda x: x\*\*2, arr))

print(arr)  
  
**# we can apply any functions on its data using the apply() method.**import pandas as pd

import numpy as np

students = pd.Series(data=[180, 175, 168, 190],

index=['Vik', 'Mehdi', 'Bella', 'Chriss'])

display(students)

def cm\_to\_feet(h):

return np.round(h/30.48, 2)

print(students.apply(cm\_to\_feet))  
  
**# reduce() function applies a given function to all the items in an input list and returns a single value.**

from functools import reduce

lst= [2,4,6,8,10]

print(reduce(lambda x, y: x + y,lst))   
  
**# lambda is used to create small, anonymous functions.**

b = lambda x: 'Even' if x%2==0 else 'Odd'

b(9)  
  
**# filter() function is used to filter the elements of an iterable based on a certain condition.**

fruits = ['mango','orange','apple','cherry','grapes']

print(list(filter(lambda fruit: 'g' in fruit,fruits)))

**Experiment 7:**  plots using matplot lib.

**Experiment 7:**  plots using seaborn.

**1. Line Chart**A line chart, also referred to as a line graph or a line plot, connects a series of data points using a line.    
  
# importing the required module

import matplotlib.pyplot as plt

# x axis values

x = [1,2,3]

# corresponding y axis values

y = [2,4,1]

# plotting the points

plt.plot(x, y)

# naming the x axis

plt.xlabel('x - axis')

# naming the y axis

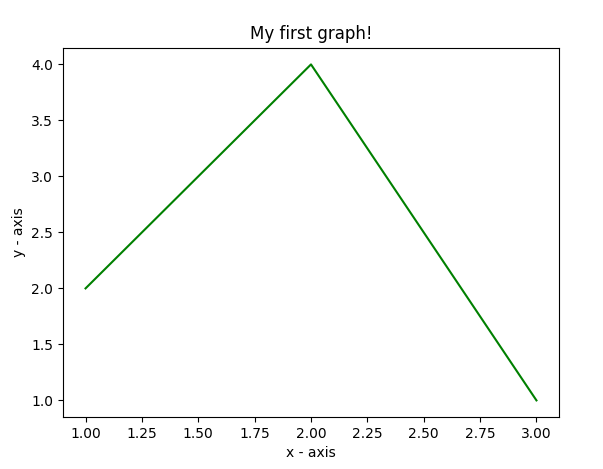
plt.ylabel('y - axis')

# giving a title to my graph

plt.title('My first graph!')

# function to show the plot

plt.show()  
  
  
**Output:**



The code seems self-explanatory. Following steps were followed:

* Define the x-axis and corresponding y-axis values as lists.
* Plot them on canvas using **.plot()** function.
* Give a name to x-axis and y-axis using **.xlabel()** and **.ylabel()** functions.
* Give a title to your plot using **.title()** function.
* Finally, to view your plot, we use **.show()** function.

**2. Bar Chart**Bar plots are a type of data visualization used to represent data in the form of rectangular bars.  
  
import matplotlib.pyplot as plt

# x-coordinates of left sides of bars

left = [1, 2, 3, 4, 5]

# heights of bars

height = [10, 24, 36, 40, 5]

# labels for bars

tick\_label = ['one', 'two', 'three', 'four', 'five']

# plotting a bar chart

plt.bar(left, height, tick\_label = tick\_label,

width = 0.8, color = ['red', 'green'])

# naming the x-axis

plt.xlabel('x - axis')

# naming the y-axis

plt.ylabel('y - axis')

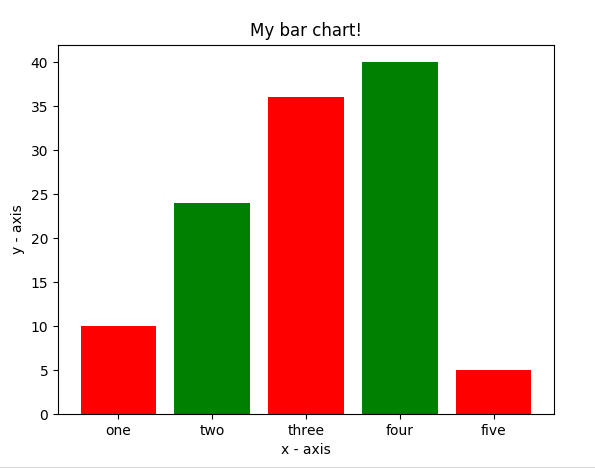
# plot title

plt.title('My bar chart!')

# function to show the plot

plt.show()

**Output:**



* Here, we use **plt.bar()** function to plot a bar chart.
* x-coordinates of the left side of bars are passed along with the heights of bars.
* you can also give some names to x-axis coordinates by defining **tick\_labels**

**3. Histogram**Histogram. A histogram is a graph showing frequency distributions. It is a graph showing the number of observations within each given interval.   
  
import matplotlib.pyplot as plt

# frequencies

ages = [2,5,70,40,30,45,50,45,43,40,44,

60,7,13,57,18,90,77,32,21,20,40]

# setting the ranges and no. of intervals

range = (0, 100)

bins = 10

# plotting a histogram

plt.hist(ages, bins, range, color = 'green',

histtype = 'bar', rwidth = 0.8)

# x-axis label

plt.xlabel('age')

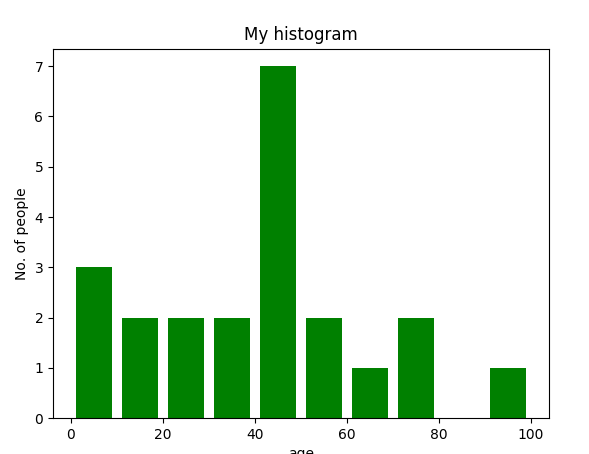
# frequency label

plt.ylabel('No. of people')

# plot title

plt.title('My histogram')

# function to show the plot

plt.show()  
 **Output:**  


* Here, we use **plt.hist()** function to plot a histogram.
* frequencies are passed as the **ages** list.
* The range could be set by defining a tuple containing min and max values.
* The next step is to “**bin**” the range of values—that is, divide the entire range of values into a series of intervals—and then count how many values fall into each interval. Here we have defined **bins** = 10. So, there are a total of 100/10 = 10 intervals.

**4. Scatter plot**

A scatter plot is a visual representation of how two variables relate to each other.  
  
import matplotlib.pyplot as plt

# x-axis values

x = [1,2,3,4,5,6,7,8,9,10]

# y-axis values

y = [2,4,5,7,6,8,9,11,12,12]

# plotting points as a scatter plot

plt.scatter(x, y, label= "stars", color= "green", marker= "\*", s=30)

# x-axis label

plt.xlabel('x - axis')

# frequency label

plt.ylabel('y - axis')

# plot title

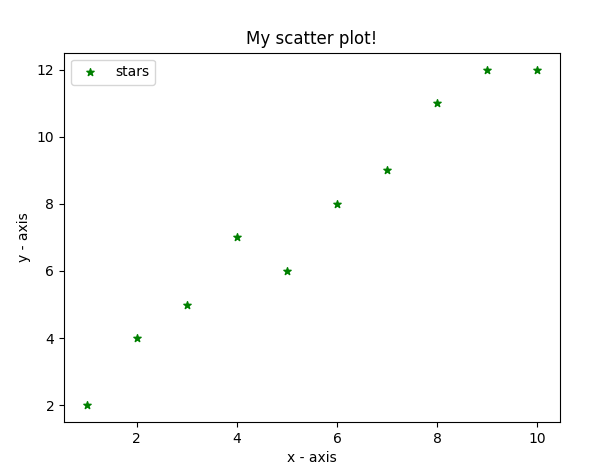
plt.title('My scatter plot!')

# showing legend

plt.legend()

# function to show the plot

plt.show()

**Output:**  


* Here, we use **plt.scatter()** function to plot a scatter plot.
* As a line, we define x and corresponding y-axis values here as well.
* **marker** argument is used to set the character to use as a marker. Its size can be defined using the **s** parameter.

**5. Pie-chart**

A Pie Chart is a circular statistical plot that can display only one series of data. The area of the chart is the total percentage of the given data. The area of slices of the pie represents the percentage of the parts of the data.   
  
import matplotlib.pyplot as plt

# defining labels

activities = ['eat', 'sleep', 'work', 'play']

# portion covered by each label

slices = [3, 7, 8, 6]

# color for each label

colors = ['r', 'y', 'g', 'b']

# plotting the pie chart

plt.pie(slices, labels = activities, colors=colors,

startangle=90, shadow = True, explode = (0, 0, 0.1, 0),

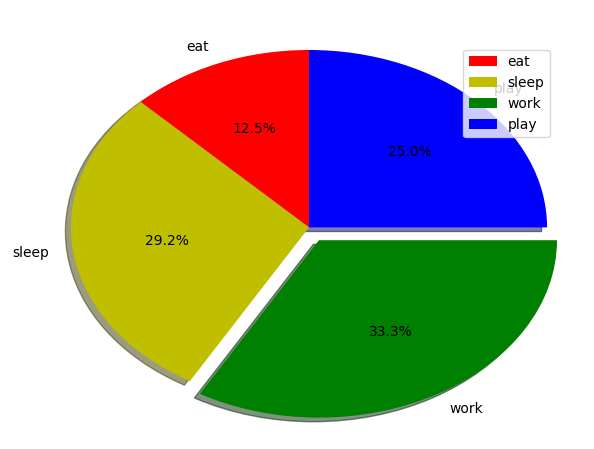
radius = 1.2, autopct = '%1.1f%%')

# plotting legend

plt.legend()

# showing the plot

plt.show()  
 **Output:**



* Here, we plot a pie chart by using **plt.pie()** method.
* First of all, we define the **labels** using a list called **activities**.
* Then, a portion of each label can be defined using another list called **slices**.
* Color for each label is defined using a list called **colors**.
* **shadow = True** will show a shadow beneath each label in pie chart.
* **startangle** rotates the start of the pie chart by given degrees counterclockwise from the x-axis.
* **explode** is used to set the fraction of radius with which we offset each wedge.
* **autopct** is used to format the value of each label. Here, we have set it to show the percentage value only upto 1 decimal place.