**Program 1: ABOUT GIT**

**Git installation and setup:** To install and set up Git on your computer, follow these steps:

1. **Download Git:** Go to the official Git website (<https://git-scm.com/downloads>) and download the appropriate version of Git for your operating system. Git is available for Windows, macOS, and Linux.
2. **Install Git:** Once you have downloaded the Git installer, run the installation program and follow the prompts to install Git on your computer. During the installation process, you may be prompted to choose various options, such as the installation location, editor, and path environment.

**Fundamentals of Git:** GIT which stands for **Global Information Tracker**, which is powerful and widely-used **version control system** commonly used for software development and other collaborative projects. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

It is used for:

* Tracking code changes.
* Tracking who made changes.
* Coding collaboration.

In software development, the tool helps in Source Code Management. Git favors not only programmers but also non-technical users by keeping track of their project files.

While working on Git, we actively use two repositories.

### What does Git do?

* Manage projects with **Repositories.**
* **Clone** a project to work on a local copy.
* Control and track changes with **Staging** and **Committing.**
* **Branch** and **Merge** to allow for work on different parts and versions of a project.
* **Pull** the latest version of the project to a local copy.
* **Push** local updates to the main project.

### Working with Git

* Initialize Git on a folder, making it a **Repository.**
* Git now creates a hidden folder to keep track of changes in that folder.
* When a file is changed, added or deleted, it is considered **modified.**
* You select the modified files you want to **Stage.**
* The **Staged** files are **Committed**, which prompts Git to store a **permanent** snapshot of the files.
* Git allows you to see the full history of every commit.
* You can revert back to any previous commit.
* Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

### Why Git?

* Over 70% of developers use Git!
* Developers can work together from anywhere in the world.
* Developers can see the full history of the project.
* Developers can revert to earlier versions of a project.

### What is GitHub?

* Git is not the same as GitHub.
* GitHub makes tools that use Git.
* GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.

**Basic local Git operations**:

The first thing we need to do, is to check if Git is properly installed, the following command is used.

$> git -v

git version 2.42.0.windows.1

**Configure Git:** Now let Git know who you are. This is important for version control systems, as each Git commit uses this information:

$ git config --global user.name "Basappa"

$ git config --global user.email "basugokak.mnti@gmail"

**Note:** Use global to set the username and e-mail for **every repository** on your computer.

If you want to set the username/e-mail for just the current repo, you can remove global

**Creating Git Folder:** Now, let's create a new folder for our project.

mkdir **make**s a **new directory**.

cd **changes** the **current working directory**.

Now that we are in the correct directory. We can start by initializing Git!

**Initialize Git:** Once you have navigated to the correct folder, you can initialize Git on that folder:

$ git init

Initialized empty Git repository in C:/Users/BASAPPA SIR/myproject/.git/

You just created your first Git Repository!

**Note:** Git now knows that it should watch the folder you initiated it on. Git creates a hidden folder to keep track of changes.

## Git Adding New Files: You just created your first local Git repo. But it is empty. So let's add some files, or create a new file using your favourite text editor. Then save or move it to the folder you just created.

## ls will ****list**** the files in the directory. We can see that index.html is there. Then we check the Git status and see if it is a part of our repo:

$ git status

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

simple.html

nothing added to commit but untracked files present (use "git add" to track)

Now Git is **aware** of the file, but has not **added** it to our repository!

Files in your Git repository folder can be in one of 2 states:

* Tracked - files that Git knows about and are added to the repository.
* Untracked - files that are in your working directory, but not added to the repository.

  When you first add files to an empty repository, they are all untracked. To get Git to track them, you need to stage them, or add them to the staging environment.

**Git Staging Environment:** One of the core functions of Git is the concepts of the Staging Environment, and the Commit. As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.

**Staged** files are files that are ready to be **committed** to the repository you are working on.

**Example:** Now, we are done working with **simple.html**. So we can add it to the Staging Environment.

$ git add simple.html

## The file should be ****Staged****. Let's check the status.

$ git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: simple.html

Now the file has been added to the Staging Environment.

**Git Add More than One File:** You can also stage more than one file at a time. Let's add 2 more files to our working folder. Use the text editor again.

A README.md file that describes the repository (recommended for all repositories).

Now add all files in the current directory to the Staging Environment:

$ git add --all

BASAPPA SIR@DESKTOP-VDBU2NH MINGW64 ~/myproject (master)

$ git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: simple.css

new file: simple.html

Now all the files are added to the Staging Environment, and we are ready to do our first commit.

**Note:** The shorthand command for git add - -all / add -A

**Git Commit:** Since we have finished our work, we are ready move from stage to commit for our repo. Adding commits keep track of our progress and changes as we work. Git considers each commit change point or **"save point"**. It is a point in the project you can go back to if you find a bug, or want to make a change.

When we commit, we should **always** include a **message**.

By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when.

$ git commit -m "First release of Hello GIT"

[master (root-commit) 15d1492] First release of Hello GIT

2 files changed, 20 insertions(+)

create mode 100644 simple.css

create mode 100644 simple.html

The commit command performs a commit, and the -m "**message"** adds a message.

The Staging Environment has been committed to our repo, with the message, **"First release of Hello GIT!"**

**Git Commit without Stage:** Sometimes, when you make small changes, using the staging environment seems like a waste of time. It is possible to commit changes directly, skipping the staging environment. The **-a** option will automatically stage every changed, already tracked file.

Let's add a small update to **simple.html**. Now check the status of our repository. But this time, we will use the --short option to see the changes in a more compact way.

$ git status --short

M simple.html

**Note:** Short status flags are:

* ?? - Untracked files A - Files added to stage
* M - Modified files D - Deleted files

$ git commit -a -m "Updated simple.html with new line"

[master 8da265b] Updated simple.html with new line

1 file changed, 1 insertion(+)

**Warning:** Skipping the Staging Environment is not generally recommended.

**View commit history:** To view the history of commits for a repository, you can use the log command.

$ git log

commit 8da265b111b99e6b84777f3cb936961e482168c4 (HEAD -> master)

Author: Basappa <basugokak.mnti@gmail>

Date: Sun Sep 24 21:18:18 2023 +0530

Updated simple.html with new line

commit 15d1492147734d13a0fb5b30ca25c460fccf1b0f

Author: Basappa <basugokak.mnti@gmail>

Date: Sun Sep 24 21:02:56 2023 +0530

First release of Hello GIT

**Git Help:** If you are having trouble remembering commands or options for commands, you can use Git help. There are a couple of different ways you can use the help command in command line:

* **git command –help:** See all the available options for the specific command
* **git help –all:** See all possible commands.

$ git commit -help

usage: git commit [-a | --interactive

-q, --quiet suppress summary after successful commit

-v, --verbose show diff in commit message template

Commit message options

-F, --file <file> read message from file

**$ git help –all (for more information)**

**Working with Git Branches:** In Git, a branch is a new / separate version of the main repository.

**How would that work without and with Git:**

**Without Git:**

1. Make copies of all the relevant files to avoid impacting the live version.
2. Start working with the design and find that code depends on code in other files that also need to be changed!
3. Make copies of the dependant files as well. Making sure that every file dependency references the correct file name.
4. EMERGENCY! There is an unrelated error somewhere else in the project that needs to be fixed ASAP!
5. Save all your files, making a note of the names of the copies you were working on.
6. Work on the unrelated error and update the code to fix it.
7. Go back to the design, and finish the work there.
8. Copy the code or rename the files, so the updated design is on the live version.
9. (2 weeks later, you realize that the unrelated error was not fixed in the new design version because you copied the files before the fix).

**With Git:**

* With a new branch called new-design, edit the code directly without impacting the main branch.
* EMERGENCY! There is an unrelated error somewhere else in the project that needs to be fixed ASAP!
* Create a new branch from the main project called small-error-fix
* Fix the unrelated error and merge the small-error-fix branch with the main branch
* You go back to the new-design branch, and finish the work there
* Merge the new-design branch with main.

Branches allow you to work on different parts of a project without impacting the main branch. When the work is complete, a branch can be merged with the main project.

You can even switch between branches and work on different projects without them interfering with each other.

**Branching in Git is very lightweight and fast!**

**New Git Branch:** Let add some new features to our **simple.html** page. We are working in our local repository, and we do not want to disturb or possibly wreck the main project.

So we create a new branch:

$ git branch "new1"

Let's confirm that we have created a new branch:

$ git branch

\* master

new1

**\*** Beside master specifies that we are currently on that branch.

checkout is the command used to check out a branch. Moving us **from** the current branch, **to** the one specified at the end of the command.

$ git checkout new1

Switched to branch 'new1'

Now we have moved our current workspace from the master branch, to the new branch.

We have made changes to a file and added a new file in the working directory (same directory as the main branch).

Now check the status of the current branch:

$ git status

On branch new1

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: simple.html

no changes added to commit (use "git add" and/or "git commit -a")

So let's go through what happens here:

* There are changes to our simple.html, but the file is not staged for commit.
* Img\_hello\_world.jpg is not tracked.

So we need to add both files to the Staging Environment for this branch.

$ git add --all

Using --all instead of individual filenames will **Stage** all changed (new, modified, and deleted) files.

Check the status of the branch.

$ git add --all

BASAPPA SIR@DESKTOP-VDBU2NH MINGW64 ~/myproject (new1)

$ git status

On branch new1

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: simple.html

We are happy with our changes. So we will commit them to the branch:

$ git commit -m "Added to hello GIT"

[new1 649b5fb] Added to hello GIT

1 file changed, 3 insertions(+)

**Note**: **“new1”** branch, that is different from the master branch.

Now, let's see what happens when we change branch to master

$ git checkout master

Switched to branch 'master'

**Emergency Branch:** Now imagine that we are not yet done with **simple.css**, but we need to fix an error on master. I don't want to mess with master directly, and I do not want to mess with **simple.css**, since it is not done yet.

So we create a new branch to deal with the emergency:

$ git checkout -b emergency-fix

Switched to a new branch 'emergency-fix'

Now we have created a new branch from master, and changed to it. We can safely fix the error without disturbing the other branches.

Let's fix our imaginary error

$ git status

On branch emergency-fix

nothing to commit, working tree clean

Stage the file, and commit:

$ git status

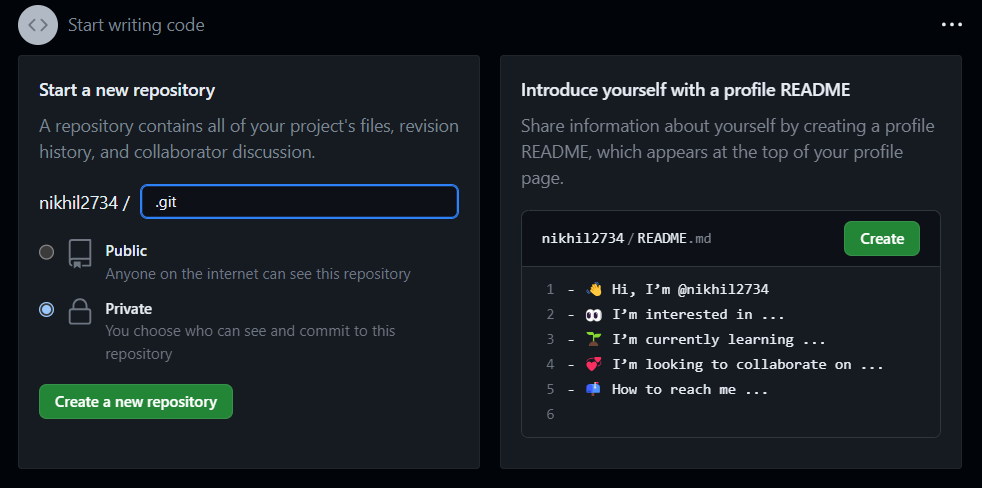
On branch emergency-fix

nothing to commit, working tree clean

After performing the this above commands and creating the files, we need to push the existing files into the Github repository we need to perform the following commands

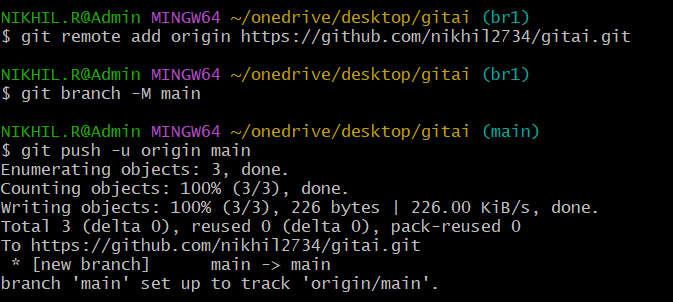
**PUSH COMMMANDS**

**Step1:** Open github.com and navigate the main page of the repository. (First we need to create repository in Github named as “.git”. Then click on create repository.)

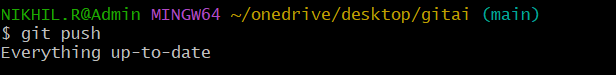


**Step2:** Push an existing repository from the command line. Drag and drop each command to the git bash, As shown below.





**Step3:** Type the following command to push the file to the repository created in Github.



**Alternative:**

1. On github.com, navigate to the main page of the Repository.
2. Above the list of files, using the add file drop down. Click upload.
3. Drag and drop file or folder like to upload your repository onto file tree.
4. Creating a commit with multiple co-authors.
5. Click commit changes.

**Program 2: Program to demonstrate array functions using NumPY**

import numpy as np

a = np.array([10,15,20,15,30])

print(a)

**# To find the sum of all element**

sum = np.sum(a)

print("The sum=",sum)

**# To find average**

avg= np.mean(a)

print(" The averge = ", avg)

**# To find median**

mid = np.median(a)

print(" The median = ", mid)

**# To find minimum element**

minimum = np.min(a)

print (" The minimum value =",minimum)

**# To find maximum element**

maximum = np.max(a)

print ("The maximum value =",maximum)

**# To identify index**

min\_index = np.argmin(a)

print("The minimum index is =",min\_index)

**# To identify index**

max\_index = np.argmax(a)

print("The maximum index is =",max\_index)

**# To find standard deviation**

std\_dev = np.std(a)

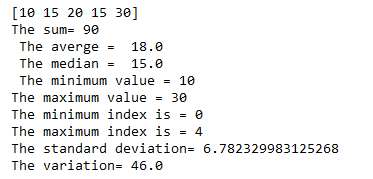
print("The standard deviation=",std\_dev)

**# To find variance**

variance = np.var(a)

print("The variation=",variance)

**OUTPUT**

****

**Pandas aggregate functions**

import pandas as pd

df=pd.DataFrame([[4,8,9], [10,7,6],[6,8,5]],

columns=['Maths', 'English','Science'])

print(df)

print(df.sum())

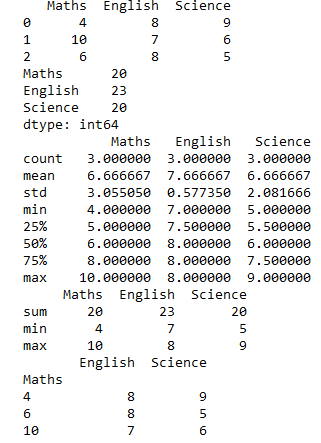
print(df.describe())

print(df.agg(['sum','min','max']))

a=df.groupby ('Maths')

print(a.first())

**OUTPUT**

****

**Program 3: Vectorized arithmetic operations using numPy**

**Program1:** To demonstrate vectorized operations

import numpy as np

print(np.sum(np.arange(10000)))

print(2\*np.array([10.2,3.5,-0.9]) - np.array([8.2,3.5,6.5]))

print(np.dot(np.array([1,-3,4]), np.array([2,0,1])))

**OUTPUT:**

49995000

[12.2 3.5 -8.3]

6

**Program2:** **To demonstrate the application of common mathematical operations using NumPy array**

A = np.array([[ 0, 1, 2],

[ 3, 4, 5],

[ 6, 7, 8]])

**#Print matrix A**

print ("\n The matrix A is \n",A)

**# Print square of each element of A**

print("\n Square of matrix A \n",A\*\*2)

**# Print cube of each element of A**

print("\n Cube of matrix A \n",A\*\*3)

**#To find square root of each element of A**

sqrt\_A =np.sqrt(A)

print("\n The square of eac element of A is \n",sqrt\_A)

**# Add .5 to each entry in row-0 (Slices return arrays of A**

**0.5 + A[0, :]**

B = np.array([[ 0, 1, 2],

[ 3, 4, 5],

[ 6, 7, 8]])

#Print matrix B

print ("\n The matrix B is \n",B)

print("The sum tarts from here....\n")

C=A+B

print("\n The sum of A and B is \n",C)

print("\nThe sum of A and B is \n",np.add(A,B))

print("\n The multiplication begins from here..\n")

Sca\_M=A\*B

print("\n The product of A and B is \n",Sca\_M)

print("\n The sum of elements of Matrix A is =", np.sum(A))

print("\n The average of elements of Matrix A is =", np.mean(A))

print("\n The standard deviation of elements of Matrix A is =", np.std(A))

print("\n The variance of elements of Matrix A is =", np.var(A))

print("\n The maximum of elements of Matrix A is =", np.max(A))

print("\n The minimum of elements of Matrix A is =", np.min(A))

print("\n The maximum of elements of Matrix A is =", np.max(A, axis=0))

print("\n The maximum of elements of Matrix A is =", np.max(A, axis=1))

print("\n The order of the matrix A \n",A.shape)

print("\n The number of element of the matrix A\n",A.size)

print("\n Returns class type \n",type(A))

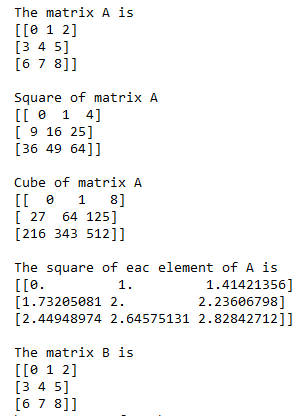
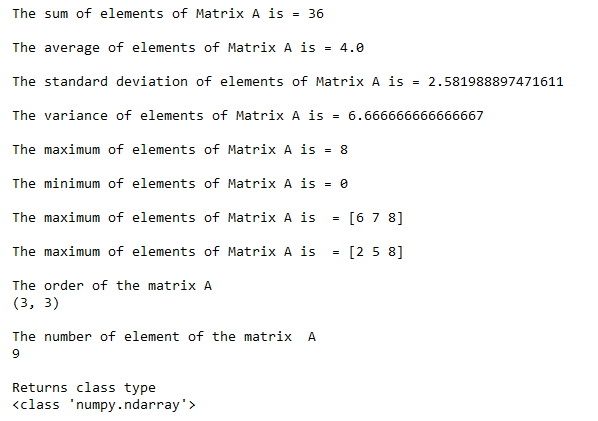
print("\n Returns\n",A[0,1:])

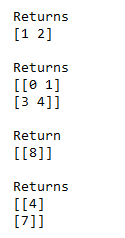
print("\n Returns \n",A[:2,:2])

print("\n Return\n",A[2:,2:])

print("\n Returns \n",A[1:3,1:2])

**OUTPUT:**

****



**Program 4: Time Series Operations in Pandas**

**Program1:** **Time series operations**

import pandas as pd

from datetime import datetime

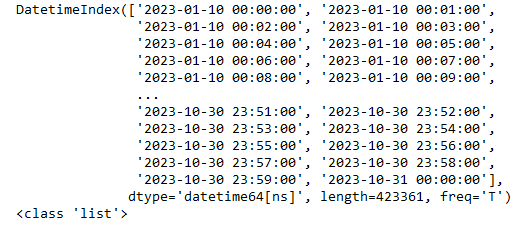
import numpy as np

range\_date = pd.date\_range(start ='1/10/2023', end ='31//10/2023',freq ='Min')

print(range\_date)

print (type([0]))

**OUTPUT:**



**Program2:** **Checking the type of our object named** **range\_date**.

import pandas as pd

from datetime import datetime

import numpy as np

range\_date = pd.date\_range(start ='1/10/2023', end ='31//10/2023',freq ='Min')

df = pd.DataFrame(range\_date, columns =['date'])

df['data'] = np.random.randint(0, 100, size =(len(range\_date)))

string\_data = [str(x) for x in range\_date]

print(string\_data[1:11])

**OUTPUT:**



**Program 5: Use Map, Filter, Reduce and Lambda functions**

**# Use Map function**

students = [

{"name": "John Doe",

"father name": "Robert Doe",

"Address": "123 Hall street"

},

{

"name": "Rahul Garg",

"father name": "Kamal Garg",

"Address": "3-Upper-Street corner"

},

{

"name": "Angela Steven",

"father name": "Jabob steven",

"Address": "Unknown"

}

]

print(list(map(lambda student: student['name'], students)))

print(list(map(lambda student: student['Address'], students)))

**OUTPUT:**

['John Doe', 'Rahul Garg', 'Angela Steven']

['123 Hall street', '3-Upper-Street corner', 'Unknown']

arr = [2,4,6,8]

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

print(arr)

**OUTPUT:**

[4, 16, 36, 64]

age = [12, 18, 20, 19, 22, 15, 17, 19]

result = map(lambda x: x, age)

print(result)

**OUTPUT:**

[12, 18, 20, 19, 22, 15, 17, 19]

**# Use Map and lambda function**

import numpy as np

import pandas as pd

tech={'Fees': [20000,25000,23000,np.NaN,26000],

'Duration':['30 Days','50 Days','30 Days','35 Dys','40 Days']}

df=pd.DataFrame(tech)

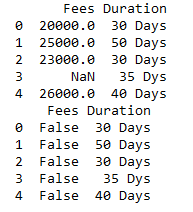
print(df)

**# lambda function**

df['Fees']= df['Fees'].map(lambda x:x==(x\*10/100))

print(df)

**OUTPUT:**

****

**Use of filter**

**# Filter unction**

import numpy as np

import pandas as pd

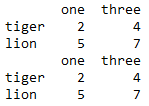
df=pd.DataFrame(np.array([[2,3,4],[5,6,7]]),

index=['tiger','lion'],columns=['one','two','three'])

print(df.filter(items=['one','three']))

print(df.filter(regex='e$',axis=1))

**OUTPUT:**

****

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

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

**OUTPUT:**

['mango', 'orange', 'grapes']

age = [12, 18, 20, 19, 22, 15, 17, 19]

result = list(filter(lambda x:x>=18, age))

print("Age greater than or equal to 18")

print(result)

**OUTPUT:**

Age greater than or equal to 18

[18, 20, 19, 22, 19]

**Using Reduce functions:**

lst = [2,4,6,8]

#find largest element

max=reduce(lambda x, y: x if x>y else y, lst)

print(max)

#find smallest element

min=reduce(lambda x, y: x if x<y else y, lst)

print(min)

**OUTPUT:**

8

2

from functools import reduce

numbers = [1, 5, 6, 7, 9, 10, 12, 45]

result = reduce(lambda a, b: a+b, numbers)

print("The sum of the list items is:", result)

**OUTPUT:**

****

# Program 6: Handling missing data

# Checking for Missing Values in Python:

import pandas as pd

import numpy as np

dict = {

'First Score':[100, 90, np.nan, 95],

'Second Score': [30, 45, 56, np.nan],

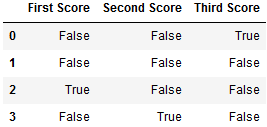
'Third Score':[np.nan, 40, 80, 98]

}

df = pd.DataFrame(dict)

df.isnull ()

**OUTPUT:**



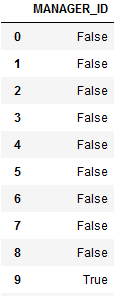
import pandas as pd

data = pd.read\_csv(r"C:\Users\BASAPPA SIR\Desktop\Dataset\Employee.csv")

df=pd.DataFrame(data["MANAGER\_ID"])

df.isnull ()

**OUTPUT:**



import pandas as pd

import numpy as np

dict = {

'First Score':[100, 90, np.nan, 95],

'Second Score': [30, 45, 56, np.nan],

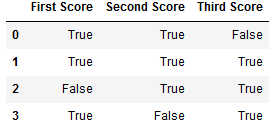
'Third Score':[np.nan, 40, 80, 98]

}

df = pd.DataFrame(dict)

df.notnull()

**OUTPUT:**



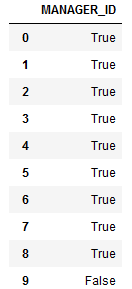
import pandas as pd

data = pd.read\_csv(r"C:\Users\BASAPPA SIR\Desktop\Dataset\Employee.csv")

df=pd.DataFrame(data["MANAGER\_ID"])

df.notnull ()

**OUTPUT:**



**Use of fillna():**

import pandas as pd

import numpy as np

dict = {

'First Score':[100, 90, np.nan, 95],

'Second Score': [30, 45, 56, np.nan],

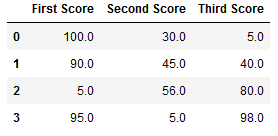
'Third Score':[np.nan, 40, 80, 98]

}

df = pd.DataFrame(dict)

df.fillna (5)

**OUTPUT:**



import pandas as pd

import numpy as np

dict = {

'First Score':[100, 90, np.nan, 95],

'Second Score': [30, 45, 56, np.nan],

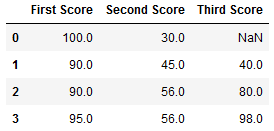
'Third Score':[np.nan, 40, 80, 98]

}

df = pd.DataFrame(dict)

df.fillna(method='pad') # method=’pad same as ffilll()

**OUTPUT:**



import pandas as pd

import numpy as np

dict = {

'First Score':[100, 90, np.nan, 95],

'Second Score': [30, 45, 56, np.nan],

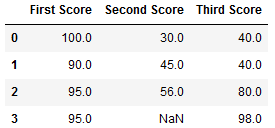
'Third Score':[np.nan, 40, 80, 98]

}

df = pd.DataFrame(dict)

df.fillna(method='bfill')

**OUTPUT:**



import pandas as pd

import numpy as np

dict = {

'First Score':[100, 90, np.nan, 95],

'Second Score': [30, 45, 56, np.nan],

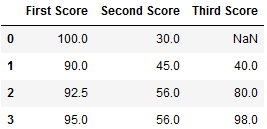
'Third Score':[np.nan, 40, 80, 98]

}

df = pd.DataFrame(dict)

df.interpolate ()

**OUTPUT:**



import pandas as pd

import numpy as np

dict = {

'First Score':[100, 90, np.nan, 95],

'Second Score': [30, 45, 56, np.nan],

'Third Score':[np.nan, 40, 80, 98]

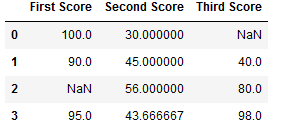
}

df = pd.DataFrame(dict)

df['Second Score']=df['Second Score'].fillna(df['Second Score'].mean())

df

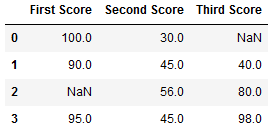
**OUTPUT:**



df['Second Score']=df['Second Score'].fillna(df['Second Score'].median())

df

**OUTPUT:**



# Dropping the rows or columns that contain missing values:

# import pandas as pd

# import numpy as np

# dict = { 'First Score':[100, 90, np.nan, 95],

# 'Second Score': [30, np.nan,45, 56,],

# 'Third Score':[np.nan, 40, 80, 98],

# 'Fourth Score':[np.nan, np.nan, np.nan, 65]

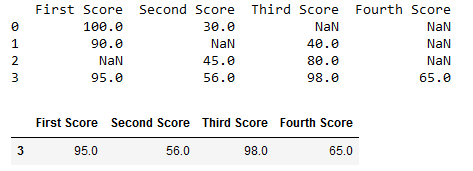
# }

# df = pd.DataFrame(dict)

# print(df)

# df.dropna ()

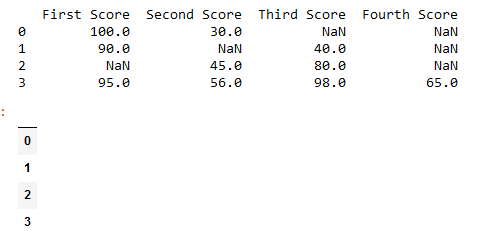
**OUTPUT:**



**print(df)**

**df.dropna(axis=1)**

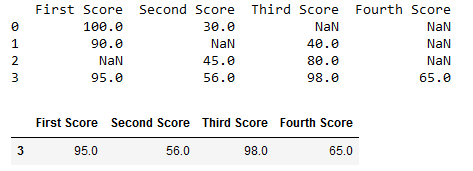
**OUTPUT:**

****

**print (df)**

**df.dropna(axis=0) # df.dropna(axis=0, how='any')**

**OUTPUT:**



import pandas as pd

import numpy as np

dict = { 'First Score':[100, 90, np.nan, 95],

'Second Score': [30, np.nan,45, 56,],

'Third Score':[np.nan, 40, 80, 98],

'Fourth Score':[np.nan, np.nan, np.nan, 65]

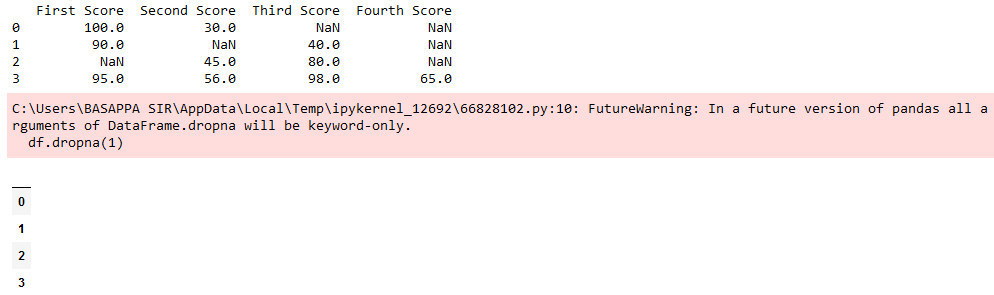
}

df = pd.DataFrame(dict)

print(df)

df.dropna(1)

**OUTPUT:**



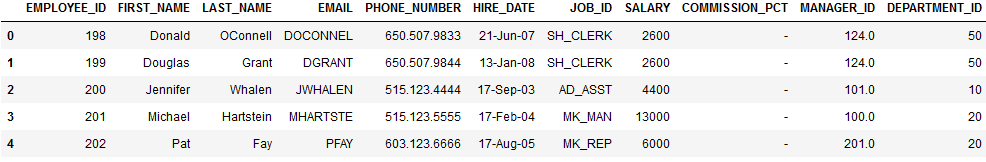
**Selection of rows:**

import pandas as pd

data = pd.read\_csv(r"C:\Users\BASAPPA SIR\Desktop\Dataset\Employee.csv")

data[0:5]

**OUTPUT:**

****

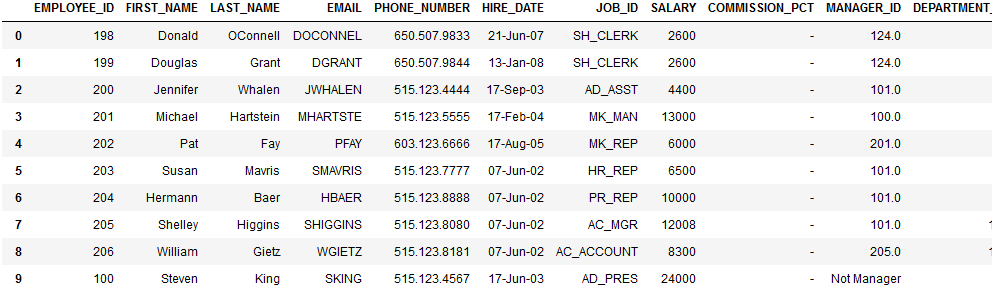
import pandas as pd

data = pd.read\_csv(r"C:\Users\BASAPPA SIR\Desktop\Dataset\Employee.csv")

data['MANAGER\_ID'].fillna("Not Manager", inplace=True)

data

**OUTPUT:**

****

# Program 7: Functions on Iris Dataset

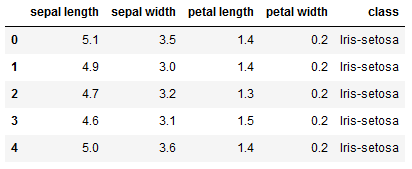
import pandas as pd

data=pd.read\_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', header=None)

data.columns=['sepal length ', 'sepal width', 'petal length', 'petal width','class']

data.head()

**OUTPUT:**



from pandas.api.types import is\_numeric\_dtype

for col in data**.**columns:

if is\_numeric\_dtype(data[col]):

print('%s'%(col))

print('\t Mean=%.2f'%data[col].mean())

print('\t Standard Deviation=%.2f'%data[col].std())

print('\t Minimum=%.2f'%data[col].min())

print('\t Maximum=%.2f'%data[col].max())

data['class']**.**value\_counts()

data.describe(include='all')

print('covariance')

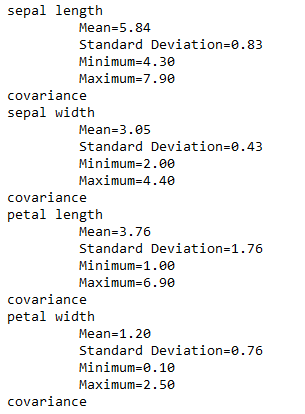
data.cov()

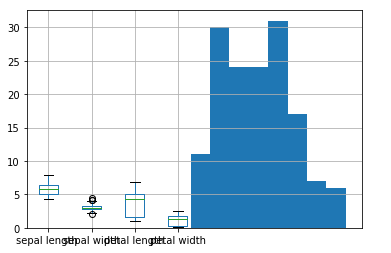
%matplotlib inline

data['sepal length '].hist(bins=8)

data.boxplot()

**OUTPUT:**

****

****

