In [1]:

```
==================== Data Analysis ============================
1. Pandas         :  Data frame analysis
2. Numpy          :  Numerical analysis
3. matplotlib     :  Graphs and plots
4. seaborn        :  plots
5. Bokhe         :   plots
6. plotly         :  plots

==================== Machine Learning ============================
7. stats          : for statistical models
8. scikit-learn(sklearn) :  ML models

====================  Deep learning ============================
9. opencv         : Computer vision
10. pillow        : Image operations
11. tensorflow    : Neural Network creations (developed by google)
12. keras         : NN
13. pytorch       : alternative package of tensorflow (facebook meta)

==================== NLP ========================================
14. NLTK          : Natural language toolkit
15. SpaCy         : Alternative to NLTK
16. wordcloud     : most frequent occured words

======================== Scarapping ================================
17. SqLite        : Data base creation
18. Beautiful soup
19. Selenium

======================== API creations ============================
20. Flask
21. FastAPI
22. Gradio
23. Django
======================== UI app creation ==========================
24.Streamlit

==================== Transfer learning models(DL) ===========
25. Mobilenet
26. Resnet
27. VGGnet
28. Inception
29. Yolo    :  Ultralytics

==================== Transfer learning models(NLP) ===========
30. Word2vec
31. GloVe

======================== Huging Face Transformers ===============
32. BERT   : Bi Directional Encoder representation of Transformers

========================== Allen NLP ======================
33. Allen NLP packages

================== GenAI ============================
34. LangChain
35. Google GeminiAi realted pcakges
36. OpenAI GPT realted packages
```

```
37. Amazon BedRock realted packages
38. Meta Llama  related packages

==================== Image and Video Generations ==============
39. GAN models realted packages
40. SORA related packages

=================== Model deployment (MlOps)==================
41. mlflow using databricks
42. kubeflow (GCP account)

==================== Cloud applications ====================
43. Azure ML realted packages
44. GCP VertexAI related packages
45. AWS Sagemaker realted packages

==================== small ==================================
46. time
47. logging
48. math
49. random
50. env
51. os
```

```
  Cell In[1], line 36
    24.Streamlit
      ^
SyntaxError: invalid decimal literal
```

Step-1: Import packages

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Step-2: create a Dataframe using list

- data : your data either list or dict
- index : we need to provide a list
- columns : we need to provide a list

In [4]:
```
names=['Ramesh','Suresh','Satish']
pd.DataFrame(names,
             columns=['Names'])
# pd.DataFrame(data,index,columns)
```

Out[4]:

	Names
0	Ramesh
1	Suresh
2	Satish

Step-3: change the index

In [6]:
```
names=['Ramesh','Suresh','Satish']
pd.DataFrame(names,
             index=['A','B','C'],
             columns=['Names'])
```

Out[6]:

	Names
A	Ramesh
B	Suresh
C	Satish

In [7]:
```
names=['Ramesh','Suresh','Satish']
idx=['A','B','C']
cols=['Names']
pd.DataFrame(names,
             index=idx,
             columns=cols)
```

Out[7]:

	Names
A	Ramesh
B	Suresh
C	Satish

Step-4: Add multiple columns

In [9]:
```
names=['Ramesh','Suresh','Satish']
age=[20,22,24]
pd.DataFrame(zip(names,age),
             columns=['Names','Age'])
```

Out[9]:

	Names	Age
0	Ramesh	20
1	Suresh	22
2	Satish	24

Step-5: Create empty dataframe and update the columns

In [11]:
```
df=pd.DataFrame()
df['Names']=['Ramesh','Suresh','Satish']
df['Age']=[20,22,24]
df
```

Out[11]:

	Names	Age
0	Ramesh	20
1	Suresh	22
2	Satish	24

Step-6: Add a new column with existing dataframe

In [17]:
```
names=['Ramesh','Suresh','Satish']
age=[20,22,24]
df=pd.DataFrame(zip(names,age),
                columns=['Names','Age'])
df['City']=['Hyd','Pune','Blr']
df
```

Out[17]:

	Names	Age	City
0	Ramesh	20	Hyd
1	Suresh	22	Pune
2	Satish	24	Blr

Step-7: overwrite the column values

- I want to overwrite my age values
- originally age is = [20,22,24]
- now i want to update the age = [30,32,34]

In [22]:
```
df['Age']=[30,32,34]
df['age']=[30,32,34]
df
```

Out[22]:

	Names	Age	City	age
0	Ramesh	30	Hyd	30
1	Suresh	32	Pune	32
2	Satish	34	Blr	34

Step-8:Drop the column

- labels
- axis
- index
- columns
- inplace

In [25]:
```
df
```

Out[25]:

	Names	Age	City	age
0	Ramesh	30	Hyd	30
1	Suresh	32	Pune	32
2	Satish	34	Blr	34

In [27]:
```
df.drop('age')   # error
df.drop('age',axis=0) # error
df.drop('age',axis=1) # correct
# here python assume age is a label
# python ask qn : is it index label or column label
# axis=1 means for columns
# axis=0 means for rows
# by default axis=0
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[27], line 1
----> 1 df.drop('age')   # error
      2 df.drop('age',axis=0) # error
      3 df.drop('age',axis=1)

File E:\Anaconda\Lib\site-packages\pandas\core\frame.py:5581, in DataFrame.drop(s
elf, labels, axis, index, columns, level, inplace, errors)
   5433 def drop(
   5434     self,
   5435     labels: IndexLabel | None = None,
   (...)
   5442     errors: IgnoreRaise = "raise",
   5443 ) -> DataFrame | None:
   5444     """
   5445     Drop specified labels from rows or columns.
   5446
   (...)
   5579             weight  1.0     0.8
   5580     """
-> 5581     return super().drop(
   5582         labels=labels,
   5583         axis=axis,
   5584         index=index,
   5585         columns=columns,
   5586         level=level,
   5587         inplace=inplace,
   5588         errors=errors,
   5589     )

File E:\Anaconda\Lib\site-packages\pandas\core\generic.py:4788, in NDFrame.drop(s
elf, labels, axis, index, columns, level, inplace, errors)
   4786 for axis, labels in axes.items():
   4787     if labels is not None:
-> 4788         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4790 if inplace:
   4791     self._update_inplace(obj)

File E:\Anaconda\Lib\site-packages\pandas\core\generic.py:4830, in NDFrame._drop_
axis(self, labels, axis, level, errors, only_slice)
   4828         new_axis = axis.drop(labels, level=level, errors=errors)
   4829     else:
-> 4830         new_axis = axis.drop(labels, errors=errors)
   4831     indexer = axis.get_indexer(new_axis)
   4833 # Case for non-unique axis
   4834 else:

File E:\Anaconda\Lib\site-packages\pandas\core\indexes\base.py:7070, in Index.dro
p(self, labels, errors)
   7068 if mask.any():
   7069     if errors != "ignore":
-> 7070         raise KeyError(f"{labels[mask].tolist()} not found in axis")
   7071     indexer = indexer[~mask]
   7072 return self.delete(indexer)

KeyError: "['age'] not found in axis"
```

In [ ]:
```
df.drop('age',
```

```
        axis=1)
```

In [ ]:
```
df # we are able to see old column
# becuase we did not apply inplace=True
```

In [ ]:
```
# whenever we use columns label
# no need of axis
df.drop(columns=['age'],
        inplace=True)
```

In [ ]:
```
df
```

In [ ]:
```
# drop index= 2
df.drop(2,
        axis=0)
```

In [ ]:
```
df.drop(index=2)
```

Step-9:rename the column

In [ ]:
```
mapper: dictionary {'City':'city'}
index:
columns: {'City'}
axis:
inplace:
```

In [ ]:
```
df
# I want to change City column: city
```

In [ ]:
```
dict1={'City':'city'}
df.rename(dict1)
```

In [ ]:
```
dict1={'City':'city'}
df.rename(dict1,axis=1)
```

In [ ]:
```
dict1={'City':'city'}
df.rename(columns=dict1)
```

In [37]:
```
df
```

Out[37]:

	Names	Age	City	age
0	Ramesh	30	Hyd	30
1	Suresh	32	Pune	32
2	Satish	34	Blr	34

In [39]:
```
dict1={'City':'city'}
df.rename(columns=dict1,
          inplace=True)
df
```

Out[39]:

	Names	Age	city	age
0	Ramesh	30	Hyd	30
1	Suresh	32	Pune	32
2	Satish	34	Blr	34

In [41]:
```
# can you change 2 to 'B'
# sir whats the difference between labels and index sir?
# index 0,1,2
# columms Names, Age,city
# label : Names  0
```

In [43]:
```
dict1={2:'B'}
df.rename(dict1,axis=0)
```

Out[43]:

	Names	Age	city	age
0	Ramesh	30	Hyd	30
1	Suresh	32	Pune	32
B	Satish	34	Blr	34

In [45]:
```
dict1={2:'B'}
df.rename(index=dict1)
```

Out[45]:

	Names	Age	city	age
0	Ramesh	30	Hyd	30
1	Suresh	32	Pune	32
B	Satish	34	Blr	34

In [47]:
```
df1=pd.DataFrame()
df1['value']=[i for i in range(1,10)]
df1['square_value']=[i**2 for i in range(1,10)]
df1['cube_value']=[i**3 for i in range(1,10)]
df1
```

Out[47]:

	value	square_value	cube_value
0	1	1	1
1	2	4	8
2	3	9	27
3	4	16	64
4	5	25	125
5	6	36	216
6	7	49	343
7	8	64	512
8	9	81	729

- len
- columns
- shape
- dtypes
- head
- tail

In [54]:
```
df1.columns
```

Out[54]:
```
Index(['value', 'square_value', 'cube_value'], dtype='object')
```

In [56]:
```
list(df1.columns)
```

Out[56]:
```
['value', 'square_value', 'cube_value']
```

In [58]:
```
df1.columns.to_list()
```

Out[58]:
```
['value', 'square_value', 'cube_value']
```

In [60]:
```
df1.shape
# 9 rows and 3 columns
```

Out[60]:
```
(9, 3)
```

In [62]:
```
df1.dtypes
# integer we will get integer
# float we will get float
# string we will get object
```

Out[62]:
```
value           int64
square_value    int64
cube_value      int64
dtype: object
```

In [64]:
```
df.dtypes
```

Out[64]:
```
Names    object
Age       int64
city     object
age       int64
dtype: object
```

In [66]:
```
df1.head(2)
```

Out[66]:

	value	square_value	cube_value
0	1	1	1
1	2	4	8

In [68]:
```
df1.tail()
```

Out[68]:

	value	square_value	cube_value
4	5	25	125
5	6	36	216
6	7	49	343
7	8	64	512
8	9	81	729

In [70]:
```
len(df1)
```

Out[70]:
```
9
```

In [72]:
```
df1.isnull()
# we are asking a qn
# a null value availbel or not
# True or False
# True means yes NULL available
# False means No NULL value not available
```

Out[72]:

	value	square_value	cube_value
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False
5	False	False	False
6	False	False	False
7	False	False	False
8	False	False	False

In [74]:
```
df1.isnull().sum()
# column wise null value count will display
```

Out[74]:
```
value           0
square_value    0
cube_value      0
dtype: int64
```

In [76]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   value         9 non-null      int64
 1   square_value  9 non-null      int64
 2   cube_value    9 non-null      int64
dtypes: int64(3)
memory usage: 348.0 bytes
```

Step-10

- How to change a specific value of a column
- df1.drop
  - either it will drop column or index
- df1.rename
  - either it will rename column or index
- df1.replace

In [79]:
```
df1
# replace cube_vlaue 64 to 464
```

Out[79]:

	value	square_value	cube_value
0	1	1	1
1	2	4	8
2	3	9	27
3	4	16	64
4	5	25	125
5	6	36	216
6	7	49	343
7	8	64	512
8	9	81	729

In [81]:
```
df1.replace(64,464) # all the data will replace
df1.replace({64:464,1:111}) # all the data will replace
```

Out[81]:

	value	square_value	cube_value
0	111	111	111
1	2	4	8
2	3	9	27
3	4	16	464
4	5	25	125
5	6	36	216
6	7	49	343
7	8	464	512
8	9	81	729

Step-11: Selection of a column

In [83]:
```
df1['cube_value']
# series
# series looks like dataframe
# rows and columns
```

Out[83]:
```
0      1
1      8
2     27
3     64
4    125
5    216
6    343
7    512
8    729
Name: cube_value, dtype: int64
```

In [85]:
```
type(df1)
```

Out[85]:
```
pandas.core.frame.DataFrame
```

In [87]:
```
type(df1['cube_value'])
```

Out[87]:
```
pandas.core.series.Series
```

In [89]:
```
df1.shape # 2D
```

Out[89]:
```
(9, 3)
```

In [91]:
```
df1['cube_value'].shape
# 1D data
```

Out[91]:
```
(9,)
```

In [ ]:
```
df1[['cube_value']]
```

In [93]:
```
type(df1[['cube_value']])
df1[['cube_value']].shape
```

Out[93]:
```
(9, 1)
```

In [95]:
```
# square_value
# cube_value
# multiple always keep those in list
df1[['cube_value','square_value']]
```

Out[95]:

	cube_value	square_value
0	1	1
1	8	4
2	27	9
3	64	16
4	125	25
5	216	36
6	343	49
7	512	64
8	729	81

In [99]:
```
df1['cube_value'].replace(64,464,inplace=True)
```

In [101…
```
df1
```

Out[101…

	value	square_value	cube_value
0	1	1	1
1	2	4	8
2	3	9	27
3	4	16	464
4	5	25	125
5	6	36	216
6	7	49	343
7	8	64	512
8	9	81	729

Step-12:How to append a row

- loc
- iloc

In [115…

```
# df1.loc[rows,columns]
# df1.iloc[rows,columns]

# we can select particular rows and columns using a list: [start:stop:step]

df1.iloc[3:7,1:3]
df1.loc[3:7,['square_value','cube_value']]
cols=['square_value','cube_value']
df1.loc[3:7,cols]
```

Out[115…

	square_value	cube_value
3	16	464
4	25	125
5	36	216
6	49	343
7	64	512

In [ ]:

```
df1.iloc[[2,3,4]] #
df1.iloc[[2]]
df1.iloc[2]
df1.iloc[[2,3,4],[0,2]]
df1.iloc[2,0]
df1.iloc[:,0]
df1.iloc[2,:]
df1.iloc[[2],:]
df1.iloc[2:6,0]
df1.iloc[2:6,[0]]
```

In [118…
```
df1.iloc[[2,3,4]]
# selecting 3 rows, all the columns
# dataframe
```

Out[118…

	value	square_value	cube_value
2	3	9	27
3	4	16	464
4	5	25	125

In [120…
```
df1.iloc[[2]] # 2D df
```

Out[120…

	value	square_value	cube_value
2	3	9	27

In [122…
```
df1.iloc[2]  # 1D series
```

Out[122…
```
value            3
square_value     9
cube_value      27
Name: 2, dtype: int64
```

In [124…
```
df1.iloc[[2,3,4],[0,2]]
# 2,3,4 row
# 0 and 2nd index (1,3)
```

Out[124…

	value	cube_value
2	3	27
3	4	464
4	5	125

In [126…
```
df1.iloc[2,0]
```

Out[126…
```
3
```

In [130…
```
df1
# in column1 (value) i want to replace value to 60
# based on specific index
df1.iloc[5,0]
```

Out[130…
```
6
```

In [132…
```
df1.iloc[:,0]
# rows and first column
```

Out[132…
```
0    1
1    2
2    3
3    4
4    5
5    6
6    7
7    8
8    9
Name: value, dtype: int64
```

In [134…
```
df1.iloc[2,:]
```

Out[134…
```
value            3
square_value     9
cube_value      27
Name: 2, dtype: int64
```

In [136…
```
df1
```

Out[136…

	value	square_value	cube_value
0	1	1	1
1	2	4	8
2	3	9	27
3	4	16	464
4	5	25	125
5	6	36	216
6	7	49	343
7	8	64	512
8	9	81	729

In [138…
```
df1.iloc[[2],:]
```

Out[138…

	value	square_value	cube_value
2	3	9	27

In [140…
```
df1.iloc[2:6,0]
```

Out[140…
```
2    3
3    4
4    5
5    6
Name: value, dtype: int64
```

In [142…
```
df1.iloc[2:6,[0]]
```

Out[142…

	value
2	3
3	4
4	5
5	6

how to append a row

In [145…
```
df1
```

Out[145…

	value	square_value	cube_value
0	1	1	1
1	2	4	8
2	3	9	27
3	4	16	464
4	5	25	125
5	6	36	216
6	7	49	343
7	8	64	512
8	9	81	729

In [ ]:
```
df1.loc[9]=[10,100,1000]
```

In [149…
```
len(df1)
```

Out[149… 10

In [147…
```
ID=len(df1)
df1.loc[ID]=[10,100,1000]
df1
```

Out[147...

	value	square_value	cube_value
0	1	1	1
1	2	4	8
2	3	9	27
3	4	16	464
4	5	25	125
5	6	36	216
6	7	49	343
7	8	64	512
8	9	81	729
9	10	100	1000

Step-13: Save the dataframe

- we can save dataframes in two ways
- .csv
- .xlsx
- where I want to save : directory
- what the the filename to save: filename
- what is the type of a file : extension
- different folder : r"C:\Users\omkar\OneDrive\Documents\Gen_AI\data1.csv"
- different folder : r"C:\Users\omkar\OneDrive\Documents\Gen_AI\data1.xlsx"
- same folder: r"data1.csv" or r"data1.xlsx"

In [153...

```
csv_path=r"C:\Users\omkar\OneDrive\Documents\Gen_AI\data1.csv"
excel_path=r"C:\Users\omkar\OneDrive\Documents\Gen_AI\data1.xlsx"
df1.to_csv(csv_path) # index column created extra
df1.to_excel(excel_path) # index column created extra
```

In [155...

```
csv_path=r"data1.csv"
excel_path=r"data1.xlsx"
df1.to_csv(csv_path) # index column created extra
df1.to_excel(excel_path) # index column created extra
```

Step-14 read the data

In [159...

```
csv_path=r"data1.csv"
pd.read_csv(csv_path)
```

Out[159…

	Unnamed: 0	value	square_value	cube_value
0	0	1	1	1
1	1	2	4	8
2	2	3	9	27
3	3	4	16	464
4	4	5	25	125
5	5	6	36	216
6	6	7	49	343
7	7	8	64	512
8	8	9	81	729
9	9	10	100	1000

In [161…

```
excel_path=r"data1.xlsx"
pd.read_excel(excel_path)
```

Out[161…

	Unnamed: 0	value	square_value	cube_value
0	0	1	1	1
1	1	2	4	8
2	2	3	9	27
3	3	4	16	464
4	4	5	25	125
5	5	6	36	216
6	6	7	49	343
7	7	8	64	512
8	8	9	81	729
9	9	10	100	1000

In [163…

```
csv_path=r"data1.csv"
excel_path=r"data1.xlsx"
df1.to_csv(csv_path,index=False) # index column created extra
df1.to_excel(excel_path,index=False) # index column created extra
```

In [165…

```
csv_path=r"data1.csv"
pd.read_csv(csv_path)
```

Out[165…

	value	square_value	cube_value
0	1	1	1
1	2	4	8
2	3	9	27
3	4	16	464
4	5	25	125
5	6	36	216
6	7	49	343
7	8	64	512
8	9	81	729
9	10	100	1000

In [ ]: