```
import pandas as pd
import numpy as np

# import numpy as np
import pandas as pd
# Define the dictionary
dict1 = {
             'Name': ['Salman', 'Anwer', 'Imran'],
             'Age': [40, 30, 45],
             'Marital_Status': ['Married', 'Married', 'Married'],
             'Institute': ['NED-CCEE', 'NED-CCEE', 'NED-CCEE'],
             'Area': ['Khi-East', 'Khi-West', 'Khi South'],
}

# Create a DataFrame
df = pd.DataFrame(dict1)

# Display the DataFrame
df
```

```
     Name  Age Marital_Status Institute       Area
0  Salman   40        Married  NED-CCEE   Khi-East
1   Anwer   30        Married  NED-CCEE   Khi-West
2   Imran   45        Married  NED-CCEE  Khi South
```

For Saving File in Excel/CSV format in same Folder/location

df.to_csv('friend.csv')

for hiding/removing Index

df.to_csv('friend_index_false.csv', index=False)

```
df.to_csv('friend.csv')

df.to_csv('friend_index_false.csv', index=False)
df
```

```
     Name  Age Marital_Status Institute       Area
0  Salman   40        Married  NED-CCEE   Khi-East
1   Anwer   30        Married  NED-CCEE   Khi-West
2   Imran   45        Married  NED-CCEE  Khi South
```

```
df
```

```
     Name  Age Marital_Status Institute       Area
0  Salman   40        Married  NED-CCEE   Khi-East
1   Anwer   30        Married  NED-CCEE   Khi-West
2   Imran   45        Married  NED-CCEE  Khi South
```

```
df.describe
```

```
<bound method NDFrame.describe of      Name  Age Marital_Status
Institute       Area
0  Salman   40        Married  NED-CCEE   Khi-East
1   Anwer   30        Married  NED-CCEE   Khi-West
2   Imran   45        Married  NED-CCEE  Khi South>
```

```
df.tail()
```

```
     Name  Age Marital_Status Institute       Area
0  Salman   40        Married  NED-CCEE   Khi-East
1   Anwer   30        Married  NED-CCEE   Khi-West
2   Imran   45        Married  NED-CCEE  Khi South
```

df.head(2) #statistical analysis

```
Book1 = pd.read_csv("Book1.csv")
Book1
```

```
    train  speed       city
0    2366     45    karachi
1    5665     41  hyderabad
2    2225     87     thatta
3  845447     99     badeen
```

```
Book1
```

```
    train  speed       city
0    2366     45    karachi
1    5665     41  hyderabad
2    2225     87     thatta
3  845447     99     badeen
```

```
Book1["speed"]
```

```
0    45
1    41
2    87
3    99
Name: speed, dtype: int64
```

```
Book1["speed"][0] = 51
```

```
C:\Users\computer house\AppData\Local\Temp\
ipykernel_4824\3050492920.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  Book1["speed"][0] = 51
```

```
Book1["speed"][0] = 51
```

```
C:\Users\computer house\AppData\Local\Temp\
ipykernel_4824\3050492920.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  Book1["speed"][0] = 51
```

```
Book1["speed"]
```

```
0    51
1    41
2    87
3    99
Name: speed, dtype: int64
```

```
Book1["speed"][0]
```

```
51
```

```
Book1.index = ["first","second","third","fourth"]
```

for row access take index, for column access take column

```
Book1
```

	train	speed	city
first	2366	51	karachi
second	5665	41	hyderabad
third	2225	87	thatta
fourth	845447	99	badeen

What is pandas? (deep understanding of pandas)

```
# Pandas is an open source data analysis library written in python.
# It laverages the power and speed of numpy to make data analysis and
preprocessing easy for data scientists.
# It provides rich and highly robust data operations.
```

How many Data Structure in pandas?

Pandas has two types of data structures.

```
# a) Series-its a one dimensional array with indexex, it stores a
single column or row of data in a Dataframe.
# A one-dimensional array(labelled) capable of holding any type of
data-Series.
# Simple to say each row and column contain Uniform Data for example
flort, or integers

# b) Data frame- It's tebular spreadsheet like structure representing
rows each of which containsone or multiple columns.
# A two-dimensional data (labeled) structure with columns of
protentially different types of data - Dataframe.
# Simple to say each row and column contain different types of data
for example , object
```

jupyter note book

The Jypyter note book is an open-source web application that

allows you to create and share documents that contain live code, equations, visualizations and narrative text. the note book has supports over 40 programming languages, including Python, R, Jullia and scala. Note books can be shared with other using emails, Drop box, Git hub, For the jupyter notebook viewers.your code can produce rich, interactive output HTML, images, videos, La tex, and custom MIME types.

```
df
```

```
     Name  Age Marital_Status Institute       Area
0  Salman   40        Married  NED-CCEE   Khi-East
1   Anwer   30        Married  NED-CCEE   Khi-West
2   Imran   45        Married  NED-CCEE  Khi South
```

```
ser = pd.Series(np.random.rand)
```

```
ser = pd.Series(np.random.rand(29))
ser
```

```
0     0.282129
1     0.798332
2     0.844305
3     0.841623
4     0.429940
5     0.910805
6     0.579562
7     0.089934
8     0.573780
9     0.068924
10    0.992834
11    0.949809
12    0.501822
13    0.777753
14    0.739014
15    0.769582
16    0.299100
17    0.526444
18    0.639765
19    0.294458
20    0.290666
21    0.769649
22    0.425583
23    0.452863
24    0.225811
25    0.512495
26    0.758410
27    0.636529
28    0.742980
dtype: float64
```

```
type(ser)   # Uniform Data
```

```
pandas.core.series.Series
```

```
newdf = pd.DataFrame (np.random.rand(329,9), index=np.arange(329)) #
for making new data frame
newdf
```

```
            0         1         2         3         4         5
6  \
```

```
0    0.005543  0.691648  0.575305  0.298572  0.743221  0.557357
0.896658
1    0.860682  0.535629  0.981931  0.642592  0.928794  0.306520
0.706873
2    0.218846  0.931109  0.567860  0.322331  0.441055  0.868891
0.847720
3    0.582582  0.582261  0.956720  0.071358  0.117989  0.126393
0.104419
4    0.074360  0.422012  0.640737  0.339524  0.014567  0.065427
0.329898
..        ...       ...       ...       ...       ...       ...
...
324  0.543346  0.898171  0.546901  0.405070  0.803456  0.549683
0.893976
325  0.264135  0.602666  0.357348  0.007127  0.512140  0.441511
0.892912
326  0.086561  0.214469  0.059369  0.352675  0.339318  0.271827
0.826901
327  0.142090  0.605423  0.687937  0.946110  0.193188  0.284360
0.447017
328  0.569265  0.000982  0.135017  0.487801  0.898353  0.166087
0.042153

            7         8
0    0.077687  0.980681
1    0.198665  0.544949
2    0.757848  0.510303
3    0.651352  0.258316
4    0.946724  0.897591
..        ...       ...
324  0.536059  0.307619
325  0.406332  0.625183
326  0.628435  0.337168
327  0.652224  0.749315
328  0.967191  0.492755

[329 rows x 9 columns]
```

```
type(newdf) # Data Frame Multiple row and column
```

```
pandas.core.frame.DataFrame
```

```
type(newdf)
```

```
pandas.core.frame.DataFrame
```

```
newdf.describe()
```

```
                0           1           2           3           4
5  \
count  329.000000  329.000000  329.000000  329.000000  329.000000
```

```
329.000000
mean     0.497627    0.513502    0.515458    0.500191    0.497232
0.498807
std      0.293180    0.291218    0.291459    0.295151    0.290763
0.285689
min      0.002769    0.000982    0.006116    0.001267    0.004959
0.011843
25%      0.261003    0.274683    0.260724    0.261806    0.232593
0.254229
50%      0.472831    0.515510    0.532029    0.510823    0.485029
0.480177
75%      0.748170    0.769320    0.771342    0.758329    0.742706
0.752743
max      0.996945    0.999745    0.996825    0.998413    0.998152
0.997509

                6           7           8
count  329.000000  329.000000  329.000000
mean     0.487124    0.528085    0.502517
std      0.290709    0.298227    0.286587
min      0.002893    0.000848    0.004869
25%      0.242883    0.270585    0.271709
50%      0.467560    0.522054    0.499438
75%      0.750156    0.797766    0.757468
max      0.996314    0.998950    0.997287

newdf.dtypes

0    float64
1    float64
2    float64
3    float64
4    float64
5    float64
6    float64
7    float64
8    float64
dtype: object

newdf.head()

          0         1         2         3         4         5
6  \
0  0.005543  0.691648  0.575305  0.298572  0.743221  0.557357
0.896658
1  0.860682  0.535629  0.981931  0.642592  0.928794  0.306520
0.706873
2  0.218846  0.931109  0.567860  0.322331  0.441055  0.868891
0.847720
3  0.582582  0.582261  0.956720  0.071358  0.117989  0.126393
```

```
0.104419
4  0.074360  0.422012  0.640737  0.339524  0.014567  0.065427
0.329898

          7         8
0  0.077687  0.980681
1  0.198665  0.544949
2  0.757848  0.510303
3  0.651352  0.258316
4  0.946724  0.897591
```

```
newdf[0][0] = "Anwar_Salman"   # to change data object
```

```
C:\Users\computer house\AppData\Local\Temp\
ipykernel_4824\667473118.py:1: FutureWarning: Setting an item of
incompatible dtype is deprecated and will raise in a future error of
pandas. Value 'Anwar_Salman' has dtype incompatible with float64,
please explicitly cast to a compatible dtype first.
  newdf[0][0] = "Anwar_Salman"
```

```
newdf.dtypes
```

```
0     object
1    float64
2    float64
3    float64
4    float64
5    float64
6    float64
7    float64
8    float64
dtype: object
```

```
newdf.head()
```

```
              0         1         2         3         4         5
6  \
0  Anwar_Salman  0.691648  0.575305  0.298572  0.743221  0.557357
0.896658
1      0.860682  0.535629  0.981931  0.642592  0.928794  0.306520
0.706873
2      0.218846  0.931109  0.567860  0.322331  0.441055  0.868891
0.847720
3      0.582582  0.582261  0.956720  0.071358  0.117989  0.126393
0.104419
4       0.07436  0.422012  0.640737  0.339524  0.014567  0.065427
0.329898

          7         8
0  0.077687  0.980681
1  0.198665  0.544949
```

```
2  0.757848  0.510303
3  0.651352  0.258316
4  0.946724  0.897591

newdf.index

Index([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
       ...
       319, 320, 321, 322, 323, 324, 325, 326, 327, 328],
      dtype='int32', length=329)

newdf.columns

RangeIndex(start=0, stop=9, step=1)

newdf.to_numpy()    # for converting into numpy array

array([['Anwar_Salman', 0.6916476253723354, 0.5753046994969795, ...,
        0.8966581906598843, 0.07768672681111854, 0.9806805276667235],
       [0.8606822377881281, 0.5356288525318552,
0.9819312774927326, ...,
        0.7068727836560575, 0.1986651526733122, 0.5449489105927766],
       [0.2188458081431468, 0.9311089287766943,
0.5678595080563484, ...,
        0.8477204190324761, 0.7578481635493118, 0.5103028073789362],
       ...,
       [0.08656107284387438, 0.21446920601323316, 0.05936875832177557,
        ..., 0.8269011532693036, 0.6284346083514463,
0.33716750939855544],
       [0.14209010007279566, 0.605423282327755,
0.6879374526912122, ...,
        0.44701672418475713, 0.6522238805658233, 0.7493146650276346],
       [0.5692651595321653, 0.0009815453035336708,
0.13501654135572172,
        ..., 0.042152746321490464, 0.9671910948855007,
        0.4927547702384869]], dtype=object)

newdf.T    # for Transpose

             0         1         2         3         4         5
6    \
0  Anwar_Salman  0.860682  0.218846  0.582582   0.07436  0.119564
0.457088
1      0.691648  0.535629  0.931109  0.582261  0.422012  0.575415
0.599647
2      0.575305  0.981931   0.56786   0.95672  0.640737  0.770117
0.149334
3      0.298572  0.642592  0.322331  0.071358  0.339524  0.725189
0.359234
4      0.743221  0.928794  0.441055  0.117989  0.014567  0.093302
0.909894
```

```
5      0.557357   0.30652  0.868891  0.126393  0.065427   0.12692
0.193501
6      0.896658  0.706873   0.84772  0.104419  0.329898  0.037331
0.577084
7      0.077687  0.198665  0.757848  0.651352  0.946724  0.463735
0.343378
8      0.980681  0.544949  0.510303  0.258316  0.897591  0.495096
0.818915

        7         8         9    ...       319       320       321
322  \
0  0.520271  0.677993  0.119494  ...  0.745494  0.242044  0.091725
0.860305
1   0.25058  0.336777   0.91449  ...   0.40502  0.979154  0.668616
0.767634
2  0.555653  0.948243  0.730314  ...  0.723698  0.667129  0.417053
0.689314
3  0.813439  0.715417  0.778106  ...  0.686239  0.528486  0.479386
0.517791
4  0.219976  0.418892   0.55108  ...  0.050907  0.821914  0.676303
0.675315
5  0.029876  0.104007  0.993352  ...  0.169029  0.299948  0.393503
0.253512
6   0.20338  0.204857  0.564067  ...  0.373581  0.701829   0.45961
0.382862
7  0.567572  0.284178  0.340527  ...  0.356929  0.252004  0.676907
0.09094
8  0.699478  0.217919  0.125092  ...  0.841133  0.898724  0.834575
0.376951

        323       324       325       326       327       328
0  0.777453  0.543346  0.264135  0.086561   0.14209  0.569265
1  0.896085  0.898171  0.602666  0.214469  0.605423  0.000982
2  0.138099  0.546901  0.357348  0.059369  0.687937  0.135017
3  0.972882   0.40507  0.007127  0.352675   0.94611  0.487801
4  0.584429  0.803456   0.51214  0.339318  0.193188  0.898353
5  0.705389  0.549683  0.441511  0.271827   0.28436  0.166087
6  0.750453  0.893976  0.892912  0.826901  0.447017  0.042153
7    0.9562  0.536059  0.406332  0.628435  0.652224  0.967191
8  0.657847  0.307619  0.625183  0.337168  0.749315  0.492755

[9 rows x 329 columns]

newdf.head()

              0         1         2         3         4         5
6  \
0  Anwar_Salman  0.691648  0.575305  0.298572  0.743221  0.557357
0.896658
1      0.860682  0.535629  0.981931  0.642592  0.928794  0.306520
```

```
0.706873
2      0.218846  0.931109  0.567860  0.322331  0.441055  0.868891
0.847720
3      0.582582  0.582261  0.956720  0.071358  0.117989  0.126393
0.104419
4       0.07436  0.422012  0.640737  0.339524  0.014567  0.065427
0.329898

          7         8
0  0.077687  0.980681
1  0.198665  0.544949
2  0.757848  0.510303
3  0.651352  0.258316
4  0.946724  0.897591
```

```
newdf.sort_index(axis=0, ascending = False)  # for sorting according
to invert index, (axis = 0 row)
# by default ascending is true. for row sorting
```

```
                0         1         2         3         4         5
6  \
328      0.569265  0.000982  0.135017  0.487801  0.898353  0.166087
0.042153
327       0.14209  0.605423  0.687937  0.946110  0.193188  0.284360
0.447017
326      0.086561  0.214469  0.059369  0.352675  0.339318  0.271827
0.826901
325      0.264135  0.602666  0.357348  0.007127  0.512140  0.441511
0.892912
324      0.543346  0.898171  0.546901  0.405070  0.803456  0.549683
0.893976
..            ...       ...       ...       ...       ...       ...
...
4         0.07436  0.422012  0.640737  0.339524  0.014567  0.065427
0.329898
3        0.582582  0.582261  0.956720  0.071358  0.117989  0.126393
0.104419
2        0.218846  0.931109  0.567860  0.322331  0.441055  0.868891
0.847720
1        0.860682  0.535629  0.981931  0.642592  0.928794  0.306520
0.706873
0    Anwar_Salman  0.691648  0.575305  0.298572  0.743221  0.557357
0.896658

            7         8
328  0.967191  0.492755
327  0.652224  0.749315
326  0.628435  0.337168
325  0.406332  0.625183
324  0.536059  0.307619
```

```
..        ...       ...
4    0.946724  0.897591
3    0.651352  0.258316
2    0.757848  0.510303
1    0.198665  0.544949
0    0.077687  0.980681

[329 rows x 9 columns]
```

```
newdf.sort_index(axis=1, ascending = False)  # for sorting according
to invert index, (axis = 1 column)
# by default ascending is true. for columns sorting
```

```
             8         7         6         5         4         3
2  \
0    0.980681  0.077687  0.896658  0.557357  0.743221  0.298572
0.575305
1    0.544949  0.198665  0.706873  0.306520  0.928794  0.642592
0.981931
2    0.510303  0.757848  0.847720  0.868891  0.441055  0.322331
0.567860
3    0.258316  0.651352  0.104419  0.126393  0.117989  0.071358
0.956720
4    0.897591  0.946724  0.329898  0.065427  0.014567  0.339524
0.640737
..        ...       ...       ...       ...       ...       ...
...
324  0.307619  0.536059  0.893976  0.549683  0.803456  0.405070
0.546901
325  0.625183  0.406332  0.892912  0.441511  0.512140  0.007127
0.357348
326  0.337168  0.628435  0.826901  0.271827  0.339318  0.352675
0.059369
327  0.749315  0.652224  0.447017  0.284360  0.193188  0.946110
0.687937
328  0.492755  0.967191  0.042153  0.166087  0.898353  0.487801
0.135017

             1             0
0    0.691648  Anwar_Salman
1    0.535629      0.860682
2    0.931109      0.218846
3    0.582261      0.582582
4    0.422012       0.07436
..        ...           ...
324  0.898171      0.543346
325  0.602666      0.264135
326  0.214469      0.086561
327  0.605423       0.14209
328  0.000982      0.569265
```

```
[329 rows x 9 columns]
```