ANSWER KEY
Internal Assessment Test 2 – Dec, 2021

Sub:	**Data Analytics using Python**							**Sub Code:**	**20MCA31**
Date:	**16/12//2021**	**Duration:**	**90 min's**	**Sem**	**III**			**Sec**	**A & B**

1	**Write any five Basic Array Statistical Methods with an example**

NumPy Statistical Functions

1. np.amin()- This function determines the minimum value of the element along a specified axis.
2. np.amax()- This function determines the maximum value of the element along a specified axis.
3. np.mean()- It determines the mean value of the data set.
4. np.median()- It determines the median value of the data set.
5. np.std()- It determines the standard deviation
6. np.var – It determines the variance.
7. np.percentile()- It determines the nth percentile of data along the specified axis.

np.amin() & np.amax()-

```
import numpy as np
arr= np.array([[1,23,78],[98,60,75],[79,25,48]])
print(arr)
#Minimum Function
print(np.amin(arr))
#Maximum Function
print(np.amax(arr))
```

Output

```
[[ 1 23 78]
[98 60 75]
[79 25 48]]
1
98
```

Mean

Mean is the sum of the elements divided by its sum and given by the following formula:

$$\bar{x} = \frac{1}{n}(x_1 + x_2 + \ldots + x_n)$$

It calculates the mean by adding all the items of the arrays and then divides it by the number of elements. We can also mention the axis along which the mean can be calculated.

```
import numpy as np
a = np.array([5,6,7])
print(a)
```

```
print(np.mean(a))
```

Output

```
[5 6 7]
6.0
```

Median

Median is the middle element of the array. The formula differs for odd and even sets.

Odd	Even
$\frac{n+1}{2}$	$\frac{n}{2}, \frac{n}{2}+1$

It can calculate the median for both one-dimensional and multi-dimensional arrays. Median separates the higher and lower range of data values.

```
import numpy as np
a = np.array([5,6,7])
print(a)
print(np.median(a))
```

Output

```
[5 6 7]
6.0
```

Standard Deviation

Standard deviation is the square root of the average of square deviations from mean. The formula for standard deviation is:

$$\text{SD} = \sqrt{\frac{\sum |x - \bar{x}|^2}{n}}$$

```
import numpy as np
a = np.array([5,6,7])
print(a)
print(np.std(a))
```

Output

```
[5 6 7]
0.816496580927726
```

Variance

Variance is the average of the square deviations. Following is the formula for the same:

	$$\sigma^2 = \frac{\sum (x_r - \overline{x})^2}{n}$$

```
import numpy as np
a = np.array([5,6,7])
print(a)
print(np.var(a))
```

Output

```
[5 6 7]
0.6666666666666666
```

NumPy Percentile Function

It has the following syntax:
numpy.percentile(input, q, axis)
The accepted parameters are:

- **input:** it is the input array.
- **q:** it is the percentile which it calculates of the array elements between 0-100.
- **axis:** it specifies the axis along which calculation is performed.

```
a = np.array([2,10,20])
print(a)
print(np.percentile(a,10,0))
```

Output

```
[ 2 10 20]
3.6
```

2 **Write python code to interact with database and perform the following task:**

i) Create table
ii) Insert 3 records into the table
iii) Display all records

SQL Based relational Databases are widely used to store data. Eg - SQL Server, PostgreSQL, MySQL, etc. Many alternative databases have also become quite popular.
The choice of DataBase is usually dependant on performance, data integrity and scalability nneds of the application.
Loading data from SQl to DataFrame is straightforward. pandas has some functions to simplify the process.
In this example, we create a SQLite database using Python's built in sqlite3 driver.

In [ ]:

```
In [ ]:  import sqlite3

         query = """
         CREATE TABLE test
         (USN  VARCHAR(20), name VARCHAR(20),
         height REAL, age INTEGER);
         """

         con = sqlite3.connect('mydata.sqlite')
         con.execute(query)
         con.commit()
```

```
In [ ]:  data = [('1CR20MCA01', 'RAM', 165.5, 23),
                 ('1CR20MCA02', 'JOHN', 170.6, 25),
                 ('1CR20MCA01', 'KRISH', 177, 225)]

         stmt = "INSERT INTO test VALUES(?,?,?,?)"

         con.executemany(stmt, data)

Out[ ]: <sqlite3.Cursor at 0x7f5f97559110>
```

```
In [ ]:  con.commit()
```

Most SQL Drivers (PyODBC, psycopg2, MySQLdb, pymssql, etc.) return a list of tuples when selecting data from table. We can use these list of tuples for the DataFrame, but the column names are present in the cursor's 'description' attribute.

```
In [ ]:  cursor = con.execute('select * from test')
         rows = cursor.fetchall()
         rows

Out[ ]: [('1CR20MCA01', 'RAM', 165.5, 23),
         ('1CR20MCA02', 'JOHN', 170.6, 25),
         ('1CR20MCA01', 'KRISH', 177.0, 225)]
```

```
In [ ]:  cursor.description

Out[ ]: (('USN', None, None, None, None, None, None),
         ('name', None, None, None, None, None, None),
         ('height', None, None, None, None, None, None),
         ('age', None, None, None, None, None, None))
```

```
In [9]:  import pandas as pd
         pd.DataFrame(rows, columns=[x[0] for x in cursor.description])
```

Out[9]:

	USN	name	height	age
0	1CR20MCA01	RAM	165.5	23
1	1CR20MCA02	JOHN	170.6	25
2	1CR20MCA01	KRISH	177.0	225

3(a) **Write a Python program to write and read the contents of text file.**

```
with open("test.txt", 'w', encoding = 'utf-8') as f:
   f.write("my first file\n")
   f.write("This file\n")
   f.write("contains three lines\n")
f.close()
f = open("test.txt", mode='r', encoding='utf-8')
print(f.read())
```

3(b) **Write a Python program to sort integer elements using Bubble sort**

```
def bubbleSort(array):

  for i in range(len(array)):
   for j in range(0, len(array) - i - 1):      # loop to compare array elements
      if array[j] > array[j + 1]:      # compare two adjacent elements change > to < to sort in descending order
         temp = array[j]   # swapping elements if elements
         array[j] = array[j+1]
         array[j+1] = temp

## Main code
x = [-2, 45, 0, 11, -9]
bubbleSort(x)

print('Sorted Array in Ascending Order:')
print(x)
```

Output

Sorted Array in Ascending Order:
[-9, -2, 0, 11, 45]

4 Write python code for the following :

i) read data and write into CSV
ii) read data from JSON
iii) read data and write into EXCEL

In [5]:

```
# If the file is comma-delimited, we can just use read_csv to read it.
import pandas as pd

df = pd.read_csv('C:/Users/mca/Desktop/21-22/data.csv')
df
```

Out[5]:

	a	b	c	d	messages
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	great

In [8]:
```
pd.read_csv('C:/Users/mca/Desktop/21-22/data.csv', header=None)
```

Out[8]:

	0	1	2	3	4
0	a	b	c	d	messages
1	1	2	3	4	hello
2	5	6	7	8	world
3	9	10	11	12	great

In [16]:
```
pd.read_csv('C:/Users/mca/Desktop/21-22/data.csv', names=['a','b','c','d','message'])
```

Out[16]:

	a	b	c	d	message
0	a	b	c	d	messages
1	1	2	3	4	hello
2	5	6	7	8	world
3	9	10	11	12	great

In [17]:
```
names = ['a','b','c','d','message']
pd.read_csv('C:/Users/mca/Desktop/21-22/data.csv', names=names, index_col='message')
```

Out[17]:

	a	b	c	d
message				
messages	a	b	c	d
hello	1	2	3	4
world	5	6	7	8
great	9	10	11	12

In [43]:
```
data.to_csv('C:/Users/mca/Desktop/21-22/out.csv')
```

In [44]:
```
pd.read_table('C:/Users/mca/Desktop/21-22/out.csv',sep=',')
```

Out[44]:

	Unnamed: 0	area	price
0	0	8450	208500
1	1	9600	181500
2	2	11250	223500
3	3	9550	140000
4	4	14260	250000

In [1]:
```
obj = """
{"name": "Wes",
 "places_lived": ["United States", "Spain", "Germany"],
 "pet": null,
 "siblings": [{"name": "Scott", "age": 30, "pets": ["Zeus", "Zuko"]},
              {"name": "Katie", "age": 38,
               "pets": ["Sixes", "Stache", "Cisco"]}]
}
"""
```

In [2]:
```
import json

result = json.loads(obj)
result
```

Out[2]:
```
{'name': 'Wes',
 'pet': None,
 'places_lived': ['United States', 'Spain', 'Germany'],
 'siblings': [{'age': 30, 'name': 'Scott', 'pets': ['Zeus', 'Zuko']},
  {'age': 38, 'name': 'Katie', 'pets': ['Sixes', 'Stache', 'Cisco']}]}
```

In [3]:
```
asjson = json.dumps(result)
asjson
```

Out[3]:
```
'{"name": "Wes", "places_lived": ["United States", "Spain", "Germany"], "pet": null, "siblings": [{"
tache", "Cisco"]}]}'
```

In [6]:
```
import pandas as pd
data = pd.read_json('ex1.json')
data
```

Out[6]:

	a	b	c
0	1	2	3
1	4	5	6
2	7	8	9

In [7]:
```
import pandas as pd
data2 = pd.read_json('ex2.json')
data2
```

Out[7]:

	ex1
0	{'a': 1, 'b': 2, 'c': 3}
1	{'a': 4, 'b': 5, 'c': 6}
2	{'a': 7, 'b': 8, 'c': 9}

In [10]:
```
import json

# Opening JSON file
f = open('ex2.json')

# returns JSON object as
# a dictionary
data = json.load(f)

# Iterating through the json
# list
for i in data['ex1']:
    print(i)

# Closing file
f.close()
```

```
{'a': 1, 'b': 2, 'c': 3}
{'a': 4, 'b': 5, 'c': 6}
{'a': 7, 'b': 8, 'c': 9}
```

storing df into json

In [13]:
```python
import pandas as pd

# Creating Dataframe
df = pd.DataFrame([['Stranger Things', 'Money Heist'],
                   ['Most Dangerous Game', 'The Stranger']],
                  columns=['Netflix', 'Quibi'])

# Convert DataFrame to JSON
data = df.to_json('export.json', orient='index')
print(data)
```

None

5 Discuss any five methods to handle the missing data with python code

The possible ways to do this are:
i) Deleting the columns **with** missing data
ii) Deleting the rows **with** missing data
iii) Filling the missing data **with** a value – Imputation- mean , median
iv) Filling the missing data **with** mode **if** it's a categorical value**.**
v) Filling **with** a Regression Model

method -1 - Deleting the columns with missing data

```python
df = pd.DataFrame(np.random.randn(7,3))
df.iloc[:4, 1] = np.nan
df.iloc[:2, 2] =np.nan
df
# Deleting the columns with missing data
df.dropna(axis=1)
```

Out[40]:

	0	1	2
0	0.600266	NaN	NaN
1	-0.974051	NaN	NaN
2	-1.328396	NaN	0.622720
3	0.495976	NaN	-0.289645
4	-0.628878	0.485675	-0.359567
5	-0.726077	-0.595948	-0.353329
6	1.190391	0.057517	0.394117

In [41]:
```python
df.dropna(axis=1)
```

Out[41]:

	0
0	0.600266
1	-0.974051
2	-1.328396
3	0.495976
4	-0.628878
5	-0.726077
6	1.190391

method -2 - Deleting the rows with missing data

```python
df = pd.DataFrame(np.random.randn(7,3))
df.iloc[:4, 1] = np.nan
df.iloc[:2, 2] =np.nan
df
# Deleting the columns with missing data
df.dropna(axis=0)
```

Out[42]:

	0	1	2
0	-0.093437	NaN	NaN
1	1.211963	NaN	NaN
2	0.746372	NaN	1.251347
3	-0.665433	NaN	0.040110
4	-1.612605	-0.147173	-1.297247
5	0.549162	-0.640737	-0.866029
6	0.620318	0.934725	0.500383

In [43]:
```
df.dropna(axis=0)
```

Out[43]:

	0	1	2
4	-1.612605	-0.147173	-1.297247
5	0.549162	-0.640737	-0.866029
6	0.620318	0.934725	0.500383

Drop all NAN rows
df = pd.DataFrame(np.random.randn(7,3))
df.iloc[:4, 1] = np.nan
df.iloc[:2, 2] =np.nan
df
df.dropna()

In [49]:
```
## Drop all NAN rows
df.dropna()
```

Out[49]:

	0	1	2
4	1.497088	0.014630	0.000073
5	0.586680	-2.273256	-1.514476
6	-0.900232	0.199116	-0.041506

fill with zero
df = pd.DataFrame(np.random.randn(7,3))
df.iloc[:4, 1] = np.nan
df.iloc[:2, 2] =np.nan
df
df.fillna(0, inplace=**True**)

Out[52]:

	0	1	2
0	0.101121	0.000000	0.000000
1	-0.055915	0.000000	0.000000
2	-0.827596	0.000000	-1.447228
3	-0.215230	0.000000	-1.035341
4	-0.062545	-1.005836	-0.938878
5	0.897645	-0.301323	1.216324
6	0.291353	-1.293540	0.681730

In [ ]:

Threshold -keyword
df = pd.DataFrame(np.random.randn(7,3))

df.iloc[:4, 1] = np.nan
df.iloc[:2, 2] =np.nan
df
df.dropna(thresh=2)

```
In [50]:
## Threshold -keyword
df = pd.DataFrame(np.random.randn(7,3))
df.iloc[:4, 1] = NA
df.iloc[:2, 2] = NA

df.dropna(thresh=2)
```

Out[50]:

	0	1	2
2	0.690770	NaN	-0.677230
3	-0.042602	NaN	-1.806489
4	-1.264985	2.028101	0.015351
5	0.117044	-0.003779	1.679544
6	-0.189678	0.107043	0.181052

In [ ]:

method -3 Filling the missing data with a value –Imputation - mean
df = pd.DataFrame(np.random.randn(7,3))
df.iloc[:4, 1] = np.nan
df.iloc[:2, 2] =np.nan
df
df.fillna(df.mean(), inplace=**True**)

```
In [34]: df.fillna(df.mean())
```

Out[34]:

	0	1	2
0	-1.126739	-0.565815	-0.024705
1	0.453015	-0.565815	-0.024705
2	0.963050	-0.565815	1.568997
3	-0.073262	-0.565815	-1.220403
4	0.924161	-1.676777	0.774737
5	0.095059	0.536180	-0.198273
6	1.390191	-0.556849	-1.048582

In [ ]:

method -3 - Filling the missing data with a value – Imputation-median
df = pd.DataFrame(np.random.randn(7,3))
df.iloc[:4, 1] = np.nan
df.iloc[:2, 2] =np.nan
df.info()
d
df.fillna(df.median(), inplace=**True**

```
In [36]: df.fillna(df.median())
```

Out[36]:

	0	1	2
0	-1.624149	-0.955995	-0.506525
1	-0.507248	-0.955995	-0.506525
2	1.076113	-0.955995	-0.753004
3	-0.767770	-0.955995	-0.506525
4	1.536892	-0.673738	-0.777791
5	-0.642732	-0.955995	-0.420421
6	-0.945062	-1.443510	0.088299

6 What are step in data preprocessing? Explain with an example

- Step 1: Analyze the dataset
- Step 2: Import the libraries
- Step 3 (a): Import the dataset
  - ✓ Setting current working directory
  - ✓ Import the dataset
- Step 4: Handling Missing Values
  - ✓ There are numerous methods to handle missing values in data frame.
- Step 5: Categorical Data
- Step 6: Splitting the dataset into train and test set
- Step 7: Feature Scaling

1. Analyze the dataset

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40		Yes
France	35	58000	Yes
Spain		52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

2. Import the libraries

```
In [12]: import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
```

3. (a): Import the dataset
   a. Setting current working directory
   b. Import the dataset

```
In [16]: df = pd.read_csv('Data.csv')

In [17]: df
```

Out[17]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

3.(b) Create matrix of features

```
In [18]: #To create a matrix of features
         X = dataset.iloc[:, :-1].values
         Y = dataset.iloc[:, 3].values

In [20]: X

Out[20]: array([['France', 44.0, 72000.0],
                ['Spain', 27.0, 48000.0],
                ['Germany', 30.0, 54000.0],
                ['Spain', 38.0, 61000.0],
                ['Germany', 40.0, nan],
                ['France', 35.0, 58000.0],
                ['Spain', nan, 52000.0],
                ['France', 48.0, 79000.0],
                ['Germany', 50.0, 83000.0],
                ['France', 37.0, 67000.0]], dtype=object)

In [21]: Y

Out[21]: array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
               dtype=object)
```

4. Handling Missing Values

There are numerous methods to handle missing values in data frame.

df.method()	**description**
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing

5: Categorical Data

- ✓ ML models are based on mathematical equations.
- ✓ **Country-** France, Germany, Spain
- ✓ **Purcahsed-** Yes, No

```
In [36]: from sklearn.preprocessing import LabelEncoder, OneHotEncoder

In [ ]: labelenc=LabelEncoder()
        x=labelenc.fit_transform(X)

In [ ]: onehotenc=OneHotEncoder()
        data=onehotenc.fit_transform(x)
```

6.Splitting the dataset into train and test set

```
In [37]: from sklearn.model_selection import train_test_split

In [ ]: X_train,X_test,Y_test,Y_train=train_test_split(X,Y,test_size=0.20)
```

- Step 7: Feature Scaling
  - ✓Age: 27 to 50
  - ✓Salary: 40,000 to 90,000
  - ✓Not on same scale
- Ways to scale:

1) Standardization:

$$x_{scaled} = \frac{x - mean}{sd}$$

```
In [47]: from sklearn.preprocessing import StandardScaler
In [48]: scaler=StandardScaler()
In [49]: x = scaler.fit_transform(x)
```

2) Normalization:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

```
In [40]: from sklearn.preprocessing import MinMaxScaler
In [43]: minmax=MinMaxScaler()
In [44]: x=minmax.fit_transform(x)
```

7 (a) **Write a python statement to Remove Row duplicates from the Data frame, null values**

An important part of Data analysis is analyzing Duplicate Values and removing them. Pandas **drop_duplicates()** method helps in removing duplicates from the data frame.

Syntax: *DataFrame.drop_duplicates(subset=None, keep='first', inplace=False)*
Parameters:
subset: *Subset takes a column or list of column label. It's default value is none. After passing columns, it will consider them only for duplicates.*
keep: *keep is to control how to consider duplicate value. It has only three distinct value and default is 'first'.*

- *If 'first', it considers first value as unique and rest of the same values as duplicate.*
- *If 'last', it considers last value as unique and rest of the same values as duplicate.*
- *If False, it consider all of the same values as duplicates*

inplace: *Boolean values, removes rows with duplicates if True.*
Return type: *DataFrame with removed duplicate rows depending on Arguments passed.*

Python's pandas library provides a function to remove rows or columns from a dataframe which contain missing values or NaN i.e.

DataFrame.dropna(self, axis=0, how='any', thresh=**None**, subset=**None**, inplace=**False**)

Arguments :

axis:
- 0 , to drop rows with missing values
- 1 , to drop columns with missing values

how:
- 'any' : drop if any NaN / missing value is present
- 'all' : drop if all the values are missing / NaN

thresh: threshold for non NaN values
inplace: If True then make changes in the dataplace itself

It removes rows or columns (based on arguments) with missing values / NaN

7(b) **Discuss loc and iloc functions with an example**

- ➢ loc is label-based, which means that you have to specify rows and columns based on their row and column **labels**.
- ➢ iloc is integer position-based, so you have to specify rows and columns by their **integer position values** (0-based integer position).

Example :

df= pd.read_csv('data/data.csv', **index_col=['Day']**)

Day	Weather	Temperature	Wind	Humidity
Mon	Sunny	12.79	13	30
Tue	Sunny	19.67	28	96
Wed	Sunny	17.51	16	20
Thu	Cloudy	14.44	11	22
Fri	Shower	10.51	26	79
Sat	Shower	11.07	27	62
Sun	Sunny	17.50	20	10

i) **Selecting via a single value :** Both loc and iloc allow input to be a single value.
Syntax for data selection:

- **loc[row_label, column_label]**
- **iloc[row_position, column_position]**

For example, let's say we would like to retrieve Friday's temperature value. With loc, we can pass the row label 'Fri' and the column label 'Temperature'.

```
# To get Friday's temperature using loc
 df.loc['Fri', 'Temperature']

Output:
10.51
```

The equivalent iloc statement should take the row **number 4** and the column **number 1** .

```
# To get Friday's temperature using iloc
 df.iloc[4, 1]

Output:
10.51
```

2) **To get all rows / To get all columns**

We can also use : to return all data. For example, to get all rows:

```
# To get all rows using loc
>>> df.loc[:, 'Temperature']

Output:
Day
Mon    12.79
Tue    19.67
Wed    17.51
Thu    14.44
Fri    10.51
Sat    11.07
Sun    17.50
Name: Temperature, dtype: float64#

# To get all rows using iloc
>>> df.iloc[:, 1]
```

And to get all columns:

```
# To get all columns using loc
>>> df.loc['Fri', :]

Output:
Weather        Shower
Temperature    10.51
Wind             26
```

Humidity 79
Name: Fri, dtype: object

# To get all columns using iloc
>>> df.**iloc[4, :]**

3. Selecting via a list of values

We can pass a list of labels to loc to select multiple rows or columns:

# Multiple rows using loc
>>> df.**loc[['Thu', 'Fri'], 'Temperature']**

Output:
Day
Thu 14.44
Fri 10.51
Name: Temperature, dtype: float64# Multiple columns

# Multiple rows using iloc
>>> df.**loc['Fri', ['Temperature', 'Wind']]**
Output:
Temperature 10.51
Wind 26
Name: Fri, dtype: object

Similarly, a list of integer values can be passed to iloc to select multiple rows or columns. Here are the equivalent statements using iloc:

>>> df.**iloc[[3, 4], 1]**
Output:
Day
Thu 14.44
Fri 10.51
Name: Temperature, dtype: float64

All the above outputs are **Series** because their results are 1-dimensional data.
4) The output will be a **DataFrame** when the result is 2-dimensional data,

# Multiple rows and columns using loc
rows = ['Thu', 'Fri']
cols=['Temperature','Wind']
df.**loc[rows, cols]**

	Temperature	Wind
Day		
Thu	14.44	11
Fri	10.51	26

# Multiple rows and columns using loc
rows = [3, 4]
cols = [1, 2]
df.**iloc[rows, cols]**

8 **Discuss different types of joins can be used in Pandas tools to implement a wide array of functionality with an example.**

These three types of joins can be used with other Pandas tools to implement a wide array of functionality

The pd.merge() function implements a number of types of joins:

the one-to-one,
many-to-one, **and**
many-to-many joins

```
import pandas as pd
import numpy as np
df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})
df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
                    'hire_date': [2004, 2008, 2012, 2014]})
display('df1', df1 , 'df2', df2 )
```

'df1'

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

'df2'

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

```
#One-to-one joins
df3 = pd.merge(df1, df2)
df3
```

Out[3]:

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

9 **What are the different ways a DataFrame can be created in python? Explain with an example**

DataFrame is a two-dimensional labeled data structures with columns of potentially different types. In general, DataFrame like a spreadsheet and it contains three components: index, columns and data. Dataframes can be created by different ways.

1. <u>**Create pandas DataFrame from dictionary of lists**</u>

The dictionary keys represent the columns names and each list represents a column contents.

```
# Import pandas library
import pandas as pd  # Create a dictionary of list
dictionary_of_lists = {
```

```
    'Name': ['Emma', 'Oliver', 'Harry', 'Sophia'],
    'Age': [29, 25, 33, 24],
    'Department': ['HR', 'Finance', 'Marketing', 'IT']}# Create the DataFrame
df1 = pd.DataFrame(dictionary_of_lists)
df1
```

2. Create pandas DataFrame from dictionary of numpy array.

The dictionary keys represent the columns names and each array element represents a column contents.

```
# Import pandas and numpy libraries
import pandas as pd
import numpy as np  # Create a numpy array
nparray = np.array(
    [['Emma', 'Oliver', 'Harry', 'Sophia'],
     [29, 25, 33, 24],
     ['HR', 'Finance', 'Marketing', 'IT']]) # Create a dictionary of nparray
dictionary_of_nparray = {
    'Name': nparray[0],
    'Age': nparray[1],
    'Department': nparray[2]}  # Create the DataFrame
df2 = pd.DataFrame(dictionary_of_nparray)
df2
```

3. Create pandas DataFrame from list of lists

Each inner list represents one row.

```
# Import pandas library
import pandas as pd# Create a dictionary of list
dictionary_of_lists = {
    'Name': ['Emma', 'Oliver', 'Harry', 'Sophia'],
    'Age': [29, 25, 33, 24],
    'Department': ['HR', 'Finance', 'Marketing', 'IT']}# Create the DataFrame
df1 = pd.DataFrame(dictionary_of_lists)
df1
```

4. Create pandas DataFrame from list of dictionaries

Each dictionary represents one row and the keys are the columns names.

```
# Import pandas library
import pandas as pd # Create a list of dictionaries
list_of_dictionaries = [
    {'Name': 'Emma', 'Age': 29, 'Department': 'HR'},
    {'Name': 'Oliver', 'Age': 25, 'Department': 'Finance'},
    {'Name': 'Harry', 'Age': 33, 'Department': 'Marketing'},
    {'Name': 'Sophia', 'Age': 24, 'Department': 'IT'}]
# Create the DataFrame
df4 = pd.DataFrame(list_of_dictionaries)
df4
```

5. Create pandas Dataframe from dictionary of pandas Series

The dictionary keys represent the columns names and each Series represents a column contents.

```
# Import pandas library
```

<table>
<tr><td></td><td>

```
import pandas as pd  # Create Series
series1 = pd.Series(['Emma', 'Oliver', 'Harry', 'Sophia'])
series2 = pd.Series([29, 25, 33, 24])
series3 = pd.Series(['HR', 'Finance', 'Marketing', 'IT']) # Create a dictionary of Series
dictionary_of_nparray = {'Name': series1, 'Age': series2, 'Department':series3} # Create the DataFrame
df5 = pd.DataFrame(dictionary_of_nparray)
df5
```

o/p:

	Name	Age	Department
0	Emma	29	HR
1	Oliver	25	Finance
2	Harry	33	Marketing
3	Sophia	24	IT

</td></tr>
<tr><td>10</td><td>

Demonstrate the following using NumPy python code

a) Searching, b) Sorting and c) Splitting

```
###Sorting
# importing Numpy package
import numpy as np

a = np.array([[1,4],[3,1]])
print("sorted array : ",np.sort(a))              # sort along the last axis
print("\n sorted flattened array:", np.sort(a, axis=0))   # sort the flattened array

x = np.array([3, 1, 2])

print("\n indices that would sort an array",np.argsort(x))

print("\n sorting complex number :" ,np.sort_complex([5, 3, 6, 2, 1]))
```

Output:

```
sorted array :  [[1 4]
 [1 3]]

 sorted flattened array: [[1 1]
 [3 4]]

 indices that would sort an array [1 2 0]

 sorting complex number : [1.+0.j 2.+0.j 3.+0.j 5.+0.j 6.+0.j]
```

In [7]:

```
### Searching

# where( ) : search an array for a certain value, and return the indexes that get a match.

# searchsorted( ) which performs a binary search in the array,
# and returns the index where the specified value would be inserted to maintain the search order.
```

In [8]:

</td></tr>
</table>

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)

arr = np.array([6, 7, 8, 9])
x = np.searchsorted(arr, 5)
print(x)

arr = np.array([1, 3, 5, 7])
x = np.searchsorted(arr, [2, 4, 6])
print(x)
```

Output:

```
(array([3, 5, 6], dtype=int64),)
0
[1 2 3]
```

###Splitting

#np.split: split an array into multiple sub-arrays as views into ary.

#np.array_split: Split an array into multiple sub-arrays of equal or near-equal size. Does not raise an exception if an equal division cannot be made.

#np.hsplit: Split array along horizontal axis.

#np.vsplit: Split array along vertical axis.

#np.array_split: Split array along specified axis.

In [10]:

```
import numpy as np

x = np.arange(9.0)
print(x)
print(np.split(x, 3))           # with no of partitions N,
print(np.split(x, [3, 5, 6, 10]))   # with indices

#the array will be divided into N equal arrays along axis. If such a split is not possible, an error is raised.

x = np.arange(9)
np.array_split(x, 4)

#Split an array into multiple sub-arrays of equal or near-equal size. Does not raise an exception if an equal division cannot be made.

a = np.array([[1, 3, 5, 7, 9, 11],
        [2, 4, 6, 8, 10, 12]])

# horizontal splitting
print("Splitting along horizontal axis into 2 parts:\n", np.hsplit(a, 2))

# vertical splitting
print("\nSplitting along vertical axis into 2 parts:\n", np.vsplit(a, 2))
```

	Output: [0. 1. 2. 3. 4. 5. 6. 7. 8.] [array([0., 1., 2.]), array([3., 4., 5.]), array([6., 7., 8.])] [array([0., 1., 2.]), array([3., 4.]), array([5.]), array([6., 7., 8.]), array([], dtype=float64)] Splitting along horizontal axis into 2 parts: [array([[1, 3, 5], [2, 4, 6]]), array([[7, 9, 11], [8, 10, 12]])] Splitting along vertical axis into 2 parts: [array([[1, 3, 5, 7, 9, 11]]), array([[2, 4, 6, 8, 10, 12]])]