ANSWER KEY
Internal Assessment Test 3 – Jan , 2022

Sub:	**Data Analytics using Python**							**Sub Code:**	**20MCA31**
Date:	**25/01//2022**	**Duration:**	**90 min's**	**Max Marks:**	**50**	**Sem:**	**III**	**Branch:**	**MCA**

1 Write a Python code to create an account object with at least two functions?

```
# Python program to create Bankaccount class
# with both a deposit() and a withdraw() function
class Bank_Account:
    def __init__(self):
        self.balance=0
        print("Hello!!! Welcome to the Deposit & Withdrawal Machine")

    def deposit(self):
        amount=float(input("Enter amount to be Deposited: "))
        self.balance += amount
        print("\n Amount Deposited:",amount)

    def withdraw(self):
        amount = float(input("Enter amount to be Withdrawn: "))
        if self.balance>=amount:
            self.balance-=amount
            print("\n You Withdrew:", amount)
        else:
            print("\n Insufficient balance ")

    def display(self):
        print("\n Net Available Balance=",self.balance)

# Driver code

# creating an object of class
s = Bank_Account()

# Calling functions with that class object
s.deposit()
s.withdraw()
s.display()
```

2 How to create dictionary in python. Explain five methods with a brief description with example

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

Dictionary

Dictionaries are used to store data values in key:value pairs.
A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

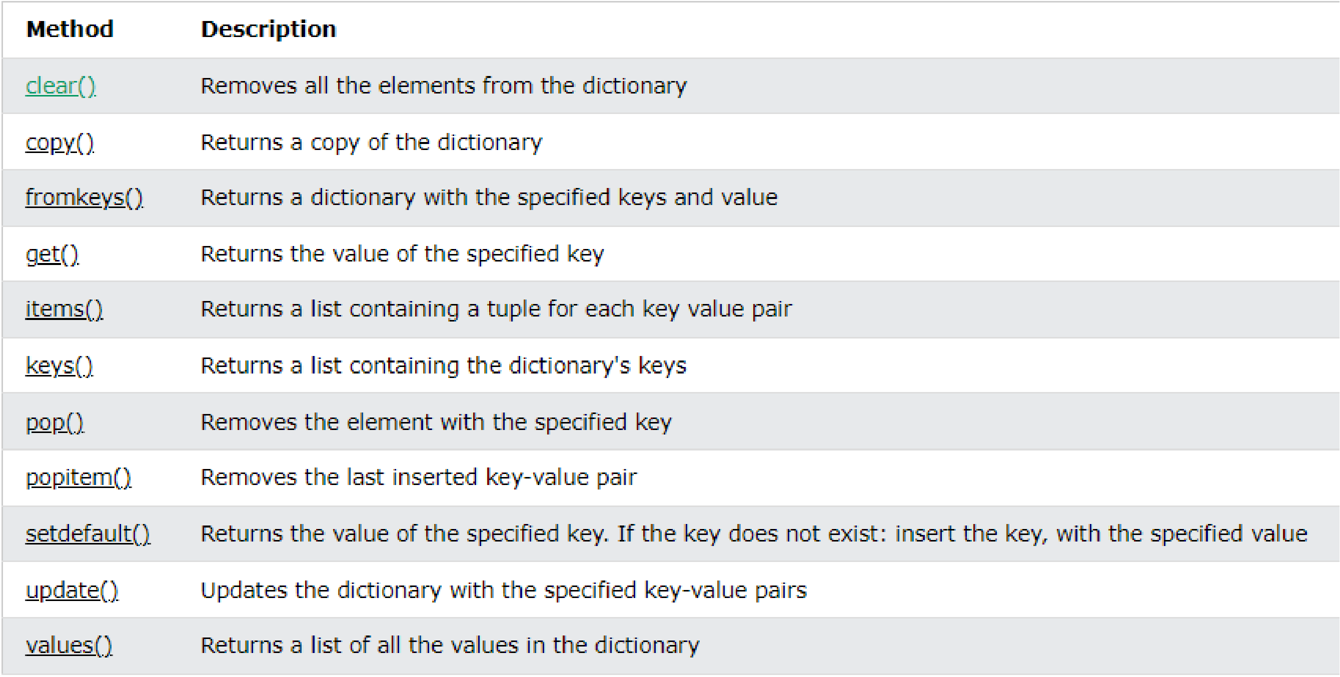
As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

Dictionaries are written with curly brackets, and have keys and values:

Example

Create and print a dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```



Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

3 How to create Constructors and method overriding in Python?

Constructors are generally used for instantiating an object. The task of constructors is to initialize(assign values) to the data members of the class when an object of the class is created. In Python the __init__() method is called the constructor and is always called when an object is created.

Syntax of constructor declaration :

```
def __init__(self):
    # body of the constructor
```

Types of constructors :

- **default constructor:** The default constructor is a simple constructor which doesn’t accept any arguments. Its definition has only one argument which is a reference to the instance being constructed.

- **parameterized constructor:** constructor with parameters is known as parameterized constructor. The parameterized constructor takes its first argument as a reference to the instance being constructed known as self and the rest of the arguments are provided by the programmer.

Example of default constructor :

Python3

```
class GeekforGeeks:

    # default constructor
    def __init__(self):
        self.geek = "GeekforGeeks"

    # a method for printing data members
    def print_Geek(self):
        print(self.geek)


# creating object of the class
obj = GeekforGeeks()

# calling the instance method using the object obj
obj.print_Geek()
Parameterised constructor
```

```
class Addition:
    first = 0
    second = 0
    answer = 0

    # parameterized constructor
    def __init__(self, f, s):
        self.first = f
        self.second = s

    def display(self):
        print("First number = " + str(self.first))
        print("Second number = " + str(self.second))
        print("Addition of two numbers = " + str(self.answer))

    def calculate(self):
        self.answer = self.first + self.second

# creating object of the class
# this will invoke parameterized constructor
obj = Addition(1000, 2000)

# perform Addition
obj.calculate()

# display result
obj.display()
```

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

Need of Data Preprocessing

- For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.
- Another aspect is that the data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithm are executed in one data set, and best out of them is chosen.

1. Rescale Data

- When our data is comprised of attributes with varying scales, many machine learning algorithms can benefit from rescaling the attributes to all have the same scale.
- This is useful for optimization algorithms in used in the core of machine learning algorithms like gradient descent.
- It is also useful for algorithms that weight inputs like regression and neural networks and algorithms that use distance measures like K-Nearest Neighbors.
- We can rescale your data using scikit-learn using the MinMaxScaler class.

Code: Python code to Rescale data (between 0 and 1)

Python

```
# importing libraries
import pandas
import scipy
import numpy
from sklearn.preprocessing import MinMaxScaler

# data set link
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"
# data parameters
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

# preparating of dataframe using the data at given link and defined columns list
dataframe = pandas.read_csv(url, names = names)
array = dataframe.values

# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]

# initialising the MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
# learning the statistical parameters for each of the data and transforming
rescaledX = scaler.fit_transform(X)

# summarize transformed data
numpy.set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

2. Binarize Data (Make Binary)

- We can transform our data using a binary threshold. All values above the threshold are marked 1 and all equal to or below are marked as 0.
- This is called binarizing your data or threshold your data. It can be useful when you have probabilities that you want to make crisp values. It is also useful when feature engineering and you want to add new features that indicate something

meaningful.

- We can create new binary attributes in Python using scikit-learn with the Binarizer class.

Code: Python code for binarization

Python

```
# import libraries
from sklearn.preprocessing import Binarizer
import pandas
import numpy

# data set link
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"
# data parameters
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 
'class']

# preparing of dataframe using the data at given link and defined columns 
list
dataframe = pandas.read_csv(url, names = names)
array = dataframe.values

# separate array into input and output components
X = array[:, 0:8]
Y = array[:, 8]
binarizer = Binarizer(threshold = 0.0).fit(X)
binaryX = binarizer.transform(X)

# summarize transformed data
numpy.set_printoptions(precision = 3)
print(binaryX[0:5,:])
```

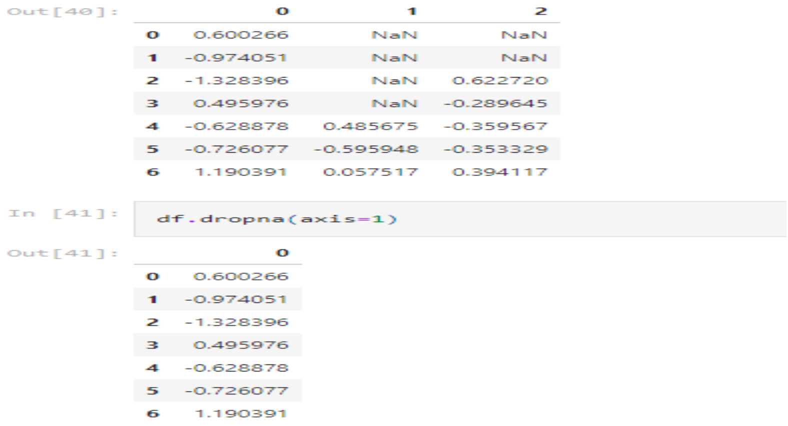
5 Discuss any five methods to handle the missing data with python code\

## *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

In [ ]:

```
## method -1 - Deleting the columns with missing data

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]:

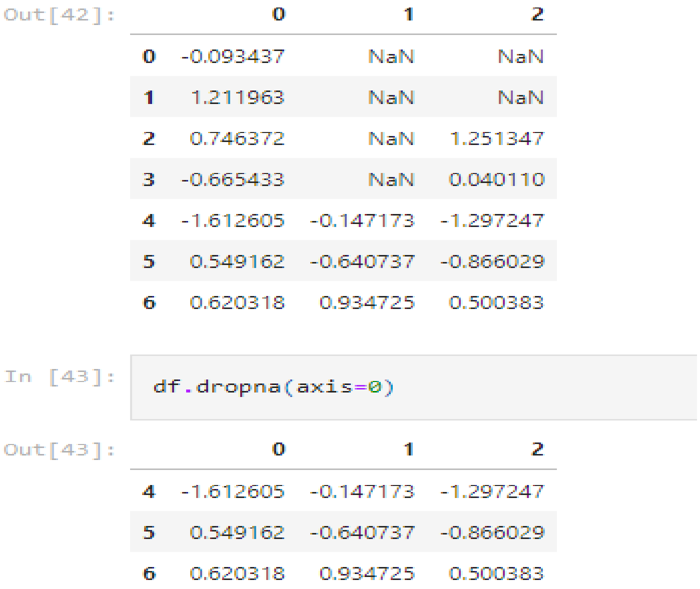
```
df.dropna(axis=1)
```

Out[41]:

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

In [ ]:

```
## method -2 - Deleting the rows with missing data
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

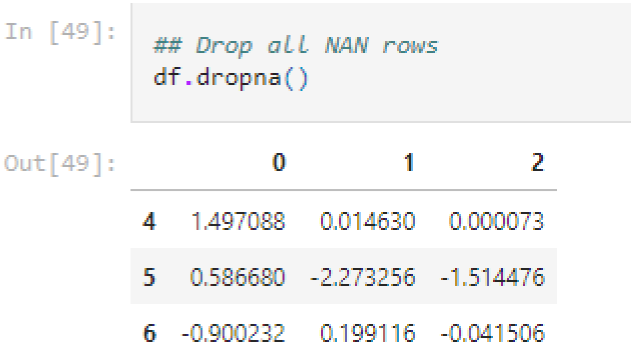
In [43]:

```
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

```
## Method -3 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]:

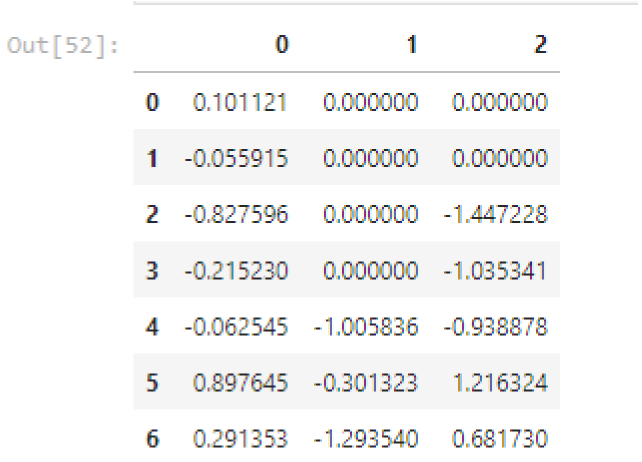
```
## Drop all NAN rows
df.dropna()
```

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

```
## Method -4 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)
```

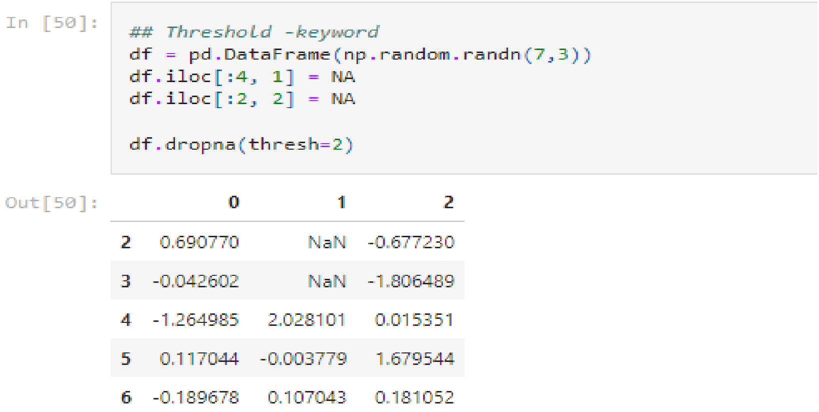


Out[52]:

	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 [ ]:

```
##Method-5  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]:

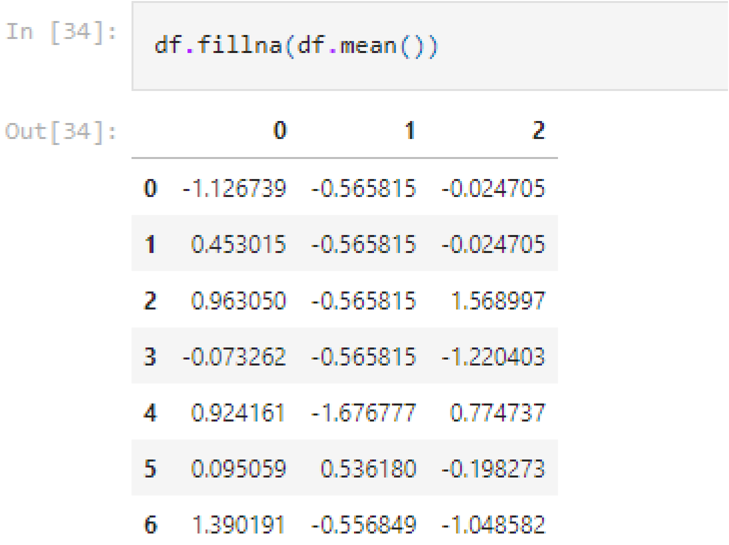
```
## Threshold -keyword
df = pd.DataFrame(np.random.randn(7,3))
df.iloc[:4, 1] = NA
df.iloc[:2, 2] = NA

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 -6 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)
```



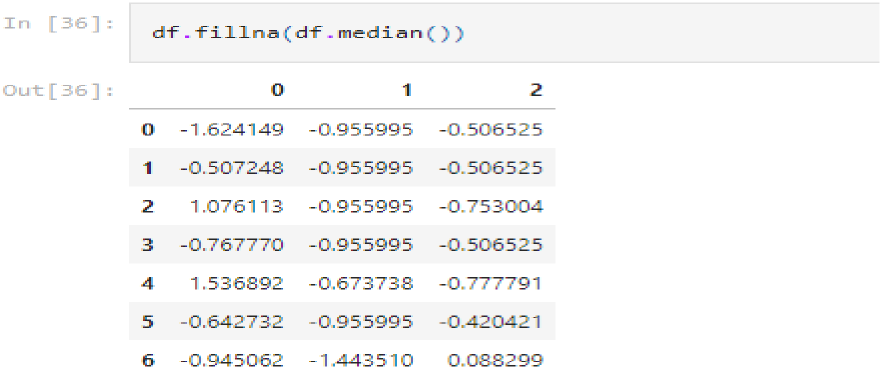
In [34]:
```
df.fillna(df.mean())
```

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 -7 - 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())
```

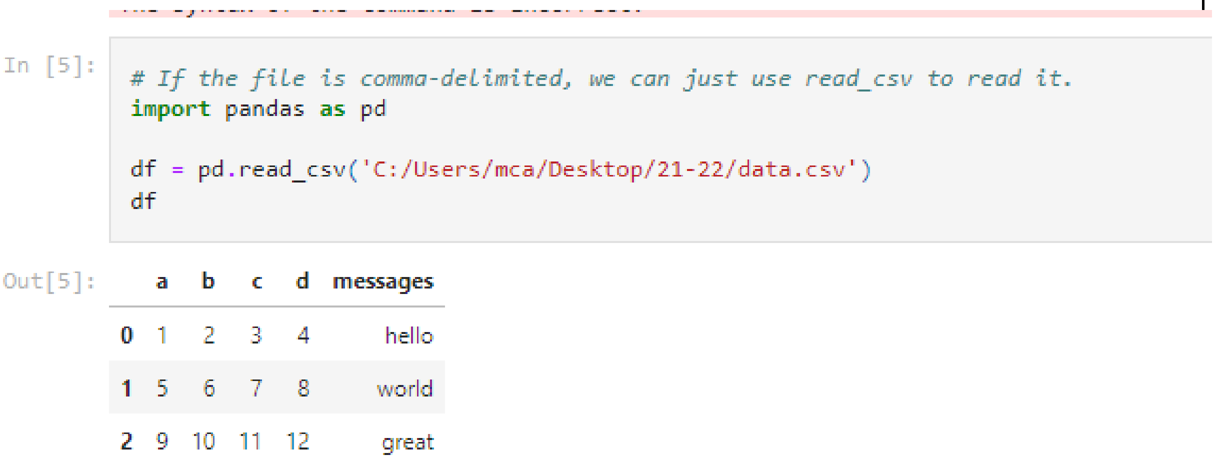
Out[36]:

	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 Write python code for the following using Pandas:

I. read from and write into CSV

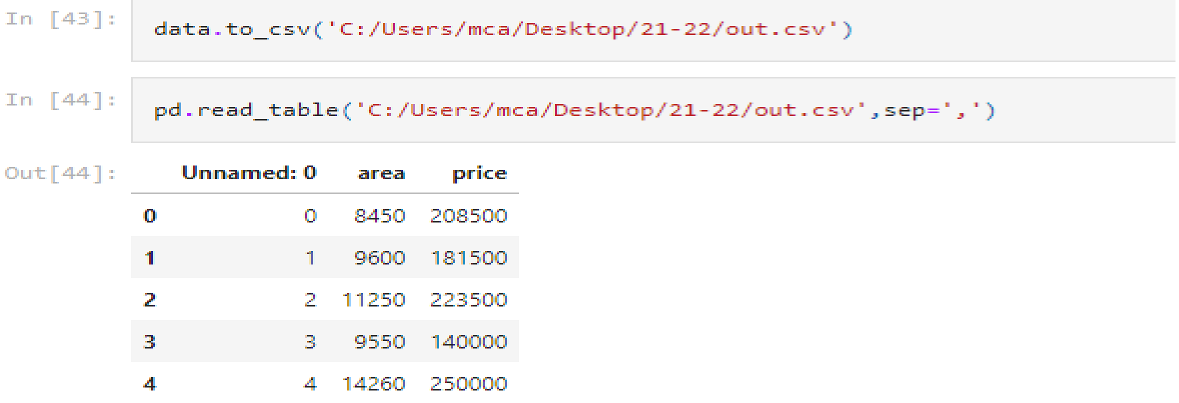
II. read from and write into JSON



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

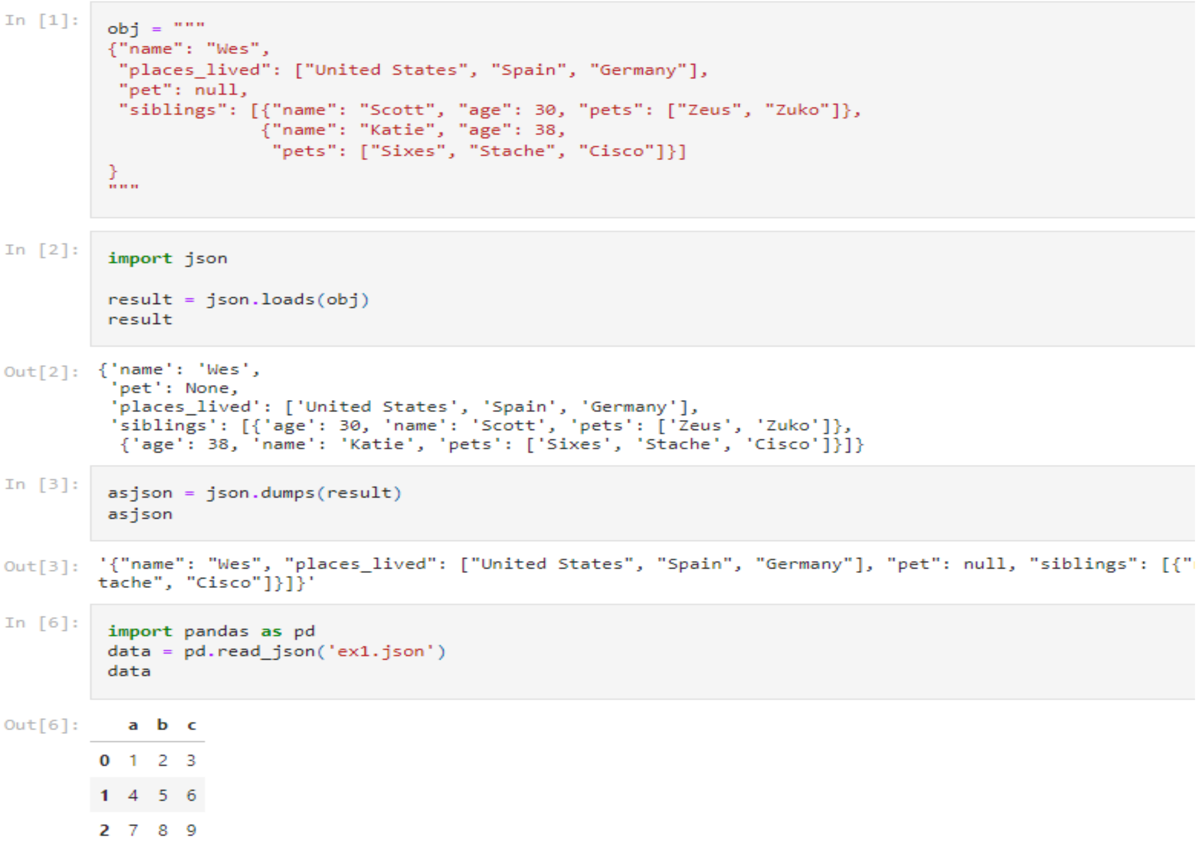


```
In [43]: data.to_csv('C:/Users/mca/Desktop/21-22/out.csv')
```

```
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']}]}
```

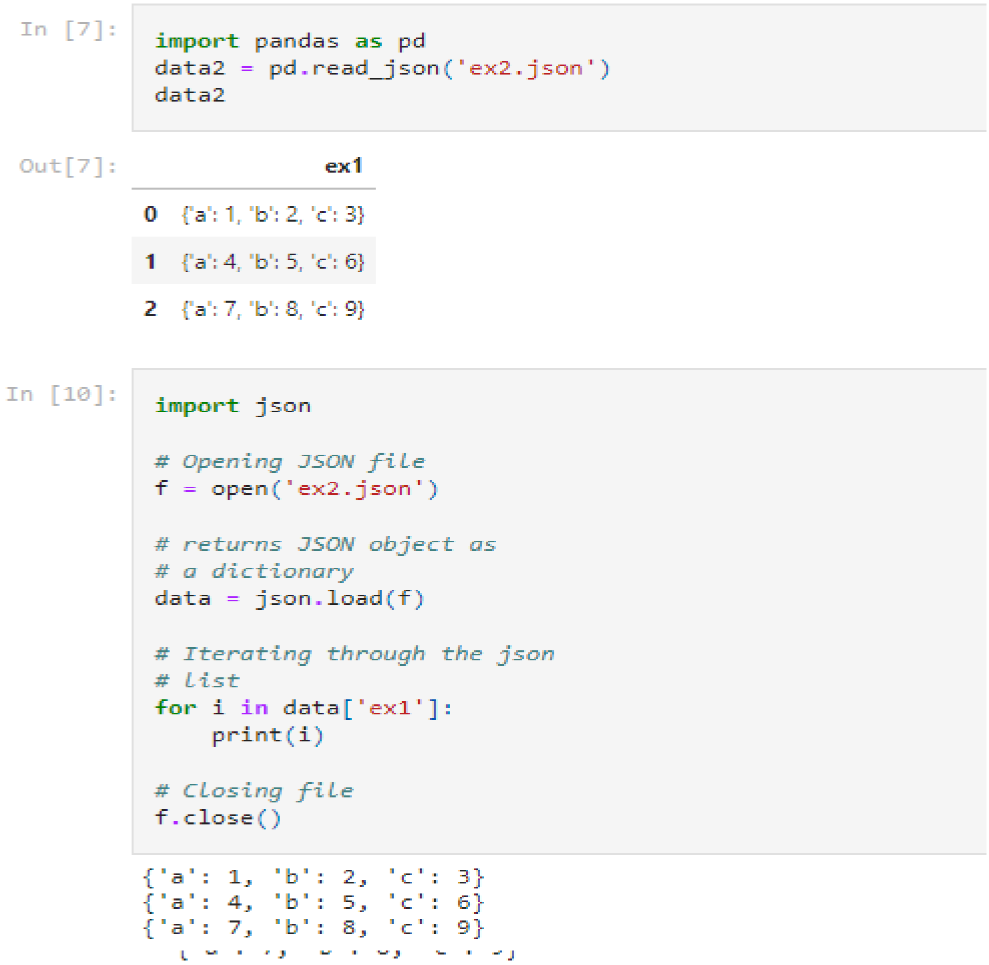
```
In [3]: asjson = json.dumps(result)
        asjson
```

```
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



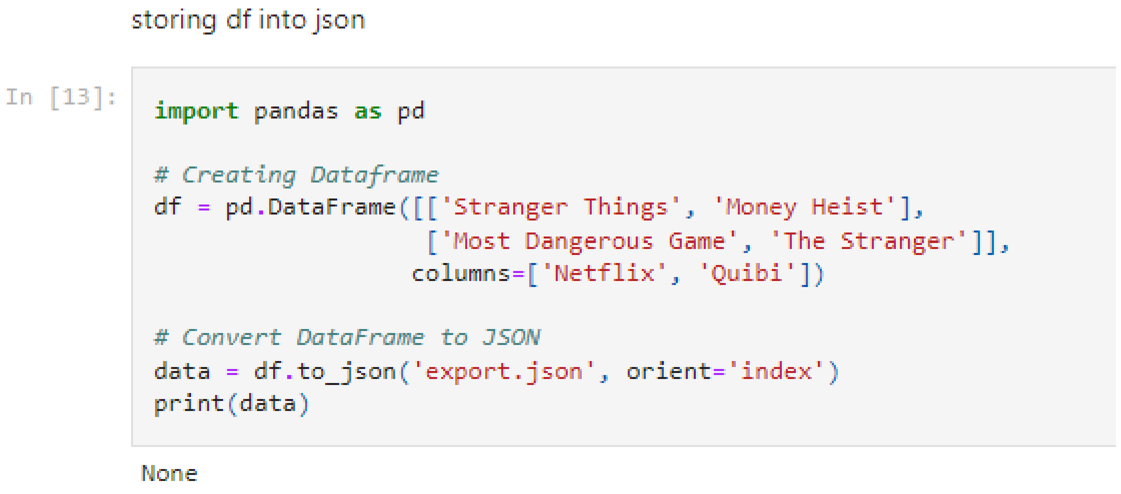
In [7]:
```
import pandas as pd
data2 = pd.read_json('ex2.json')
data2
```

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]:
```
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
```

7 Write python code to interact with database and perform the following task

I. Create table

II. Insert 3 record into table

III. Display all records

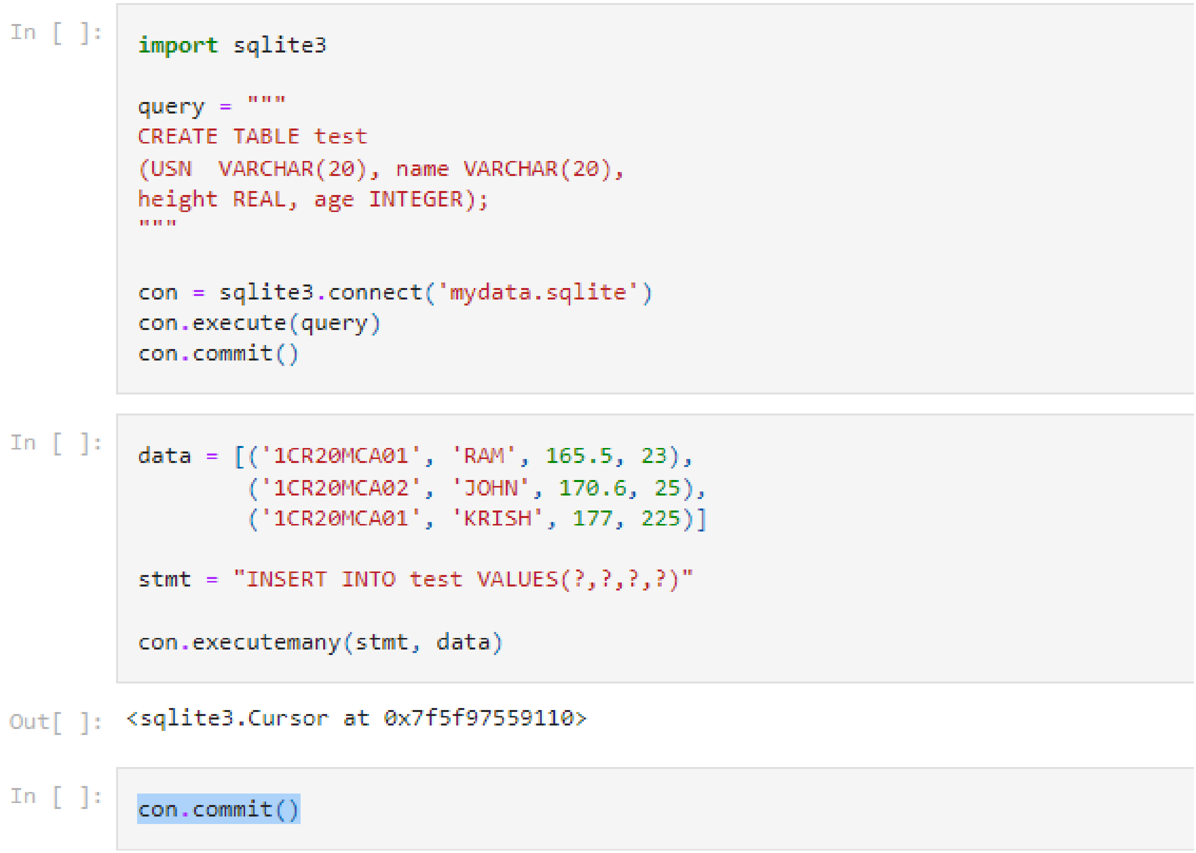
SQL Based relational Databases are widely used to store data. Eg - SQL Server, PostgreSQL, MySQL, etc. Many alternative databases have also become quite popular.

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.

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.



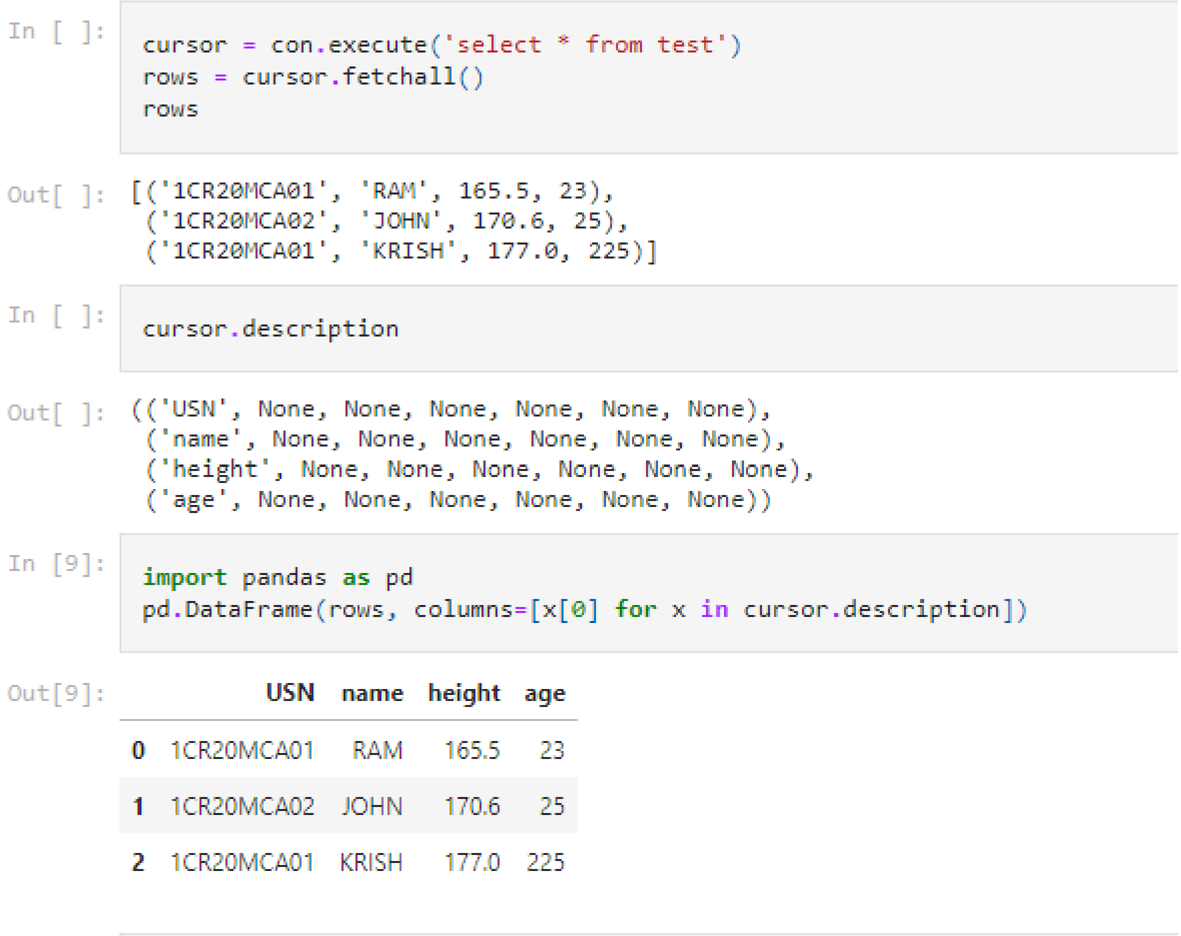
In [ ]:
```
import sqlite3

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

8 Write any 5 built-in Pandas aggregations functions to table summarizes . explain with an example

The groupby() function returns a GroupBy object, but essentially describes how the rows of the original data set has been split. the GroupBy object .groups variable is a dictionary whose keys are the computed unique groups and corresponding values being the axis labels belonging to each group. For example:

```
data.groupby(['month']).groups.keys()
Out[59]: ['2014-12', '2014-11', '2015-02', '2015-03', '2015-01']
len(data.groupby(['month']).groups['2014-11'])
Out[61]: 230
```

Functions like max(), min(), mean(), first(), last() can be quickly applied to the GroupBy object to obtain summary statistics for each group – an immensely useful function.