CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test 3 –March 2023

Sub:	Data Analytics using Python							Sub Code:	20MCA31
Date:	13/03//2023	Duration:	90 min's	Max Marks:	50	Sem:	III	Branch:	MCA

Note: Answer FIVE FULL Questions, choosing ONE full question from each Module

		MARKS	OBE CO	OBE RBT
	PART I			
1	Explain with an example "The GroupBy object "-- aggregate, filter, transform, and apply.	10	CO 2	L3
	OR			
2	How to customize the default plot settings of Matplotlib w.r.t runtime configuration and stylesheets? Give example.	10	CO 2	L4
	PART II			
3	Write Python code to create an account object with two functions and constructor. [Hint :The "deposit" method takes an amount to deposit as an argument, adds it to the account balance, and prints a message with the new balance. The "withdraw" method takes an amount to withdraw as an argument, checks if the account has sufficient funds, and either subtracts the amount from the account balance or prints an error message. The constructor takes an initial balance as an argument and initializes the account with that balance.]	10	CO 3	L4
	OR			
4	Write a class Rectangle containing numeric attributes width and height. This class should contain another attribute corner which is an instance of another class Point. Implement following functions – i) function to print corner point as an ordered-pair ii) function find_center() to compute center point of the rectangle iii) function resize() to modify the size of rectangle	10	CO 3	L4
	PART III			

			CO 1	L2
5.	Discuss any five methods to handle the missing data with python code Elaborate on Seaborn versus Matplotlib with suitable examples	10	CO 3	L1
	OR			
6	Discuss any 10 built in String methods with an example? Explain Reshape and Pivot operations with example	10	CO 3	L3
	PART IV			
7	Write a python program to demonstrate data visualization for Line Plot, Histogram, heatmap, Boxplot, and Distribution Plots	10	CO 3	L3
	OR			
8	Write a Pandas program to create a data frame with the test data , split the dataframe by school code and get mean, min, and max value of i) age ii) weight for each school. Test Data: school class name age height weight S1 s001 V Ram 12 173 35 S2 s002 V Kiran 12 192 32 S3 s003 VI Ryan 13 186 33 S4 s001 VI Bhim 13 167 30 S5 s002 VI Sita 14 151 31 S6 s004 V Bhavana 12 159 32	10	CO 3	L2
	PART V			
9.	Write python code for the following using Pandas: I. read from and write into CSV II. read from and write into JSON	10	CO3	L2
	OR			
10	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	10	CO3	L2

SOLUTION

1. Explain with an example “The GroupBy object “-- aggregate, filter, transform, and apply

The GroupBy object is a very flexible abstraction. In many ways, you can simply treat it as if it's a collection of DataFrames, and it does the difficult things under the hood. Let's see some examples using the Planets data.

Perhaps the most important operations made available by a GroupBy are *aggregate*, *filter*, *transform*, and *apply*. We'll discuss each of these more fully in "Aggregate, Filter, Transform, Apply", but before that let's introduce some of the other functionality that can be used with the basic GroupBy operation.

Aggregate, filter, transform, apply

The preceding discussion focused on aggregation for the combine operation, but there are more options available. In particular, GroupBy objects have aggregate(), filter(), transform(), and apply() methods that efficiently implement a variety of useful operations before combining the grouped data.

For the purpose of the following subsections, we'll use this DataFrame:

Code

```
rng = np.random.RandomState(0)
df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],
                   'data1': range(6),
                   'data2': rng.randint(0, 10, 6)},
                   columns = ['key', 'data1', 'data2'])
df
```

Out[18]:

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

Aggregation

The aggregate() method allows for even more flexibility. It can take a string, a function, or a list thereof, and compute all the aggregates at once. Here is a quick example combining all these:

```
In [21]: df.groupby('key').aggregate(['min', np.median, max])
```

Out[21]:

	data1			data2		
	min	median	max	min	median	max
key						
A	0	1.5	3	3	4.0	5
B	1	2.5	4	0	3.5	7
C	2	3.5	5	3	6.0	9

Filtering

A filtering operation allows you to drop data based on the group properties. For example, we might want to keep all groups in which the standard deviation is larger than some critical value:

In [29]:
```
def filter_func(x):
    return x['data2'].std() > 4

display('df', "df.groupby('key').std()", df.groupby('key').filter(filter_func))
```

'df'

"df.groupby('key').std()"

	key	data1	data2
1	B	1	0
2	C	2	3
4	B	4	7
5	C	5	9

The filter function should return a Boolean value specifying whether the group passes the filtering. Here because group A does not have a standard deviation greater than 4, it is dropped from the result.

Transformation

While aggregation must return a reduced version of the data, transformation can return some transformed version of the full data to recombine. For such a transformation, the output is the same shape as the input. A common example is to center the data by subtracting the group-wise mean:

In [24]:
```
df.groupby('key').transform(lambda x: x - x.mean())
```

Out[24]:

	data1	data2
0	-1.5	1.0
1	-1.5	-3.5
2	-1.5	-3.0
3	1.5	-1.0
4	1.5	3.5
5	1.5	3.0

The apply() method

The `apply()` method lets you apply an arbitrary function to the group results. The function should take a `DataFrame`, and return either a Pandas object (e.g., `DataFrame`, `Series`) or a scalar; the combine operation will be tailored to the type of output returned.

For example, here is an `apply()` that normalizes the first column by the sum of the second:

In [28]:
```
def norm_by_data2(x):
    # x is a DataFrame of group values
    x['data1'] /= x['data2'].sum()
    return x

display('df', df.groupby('key').apply(norm_by_data2))
```

'df'

	key	data1	data2
0	A	0.000000	5
1	B	0.142857	0
2	C	0.166667	3
3	A	0.375000	3
4	B	0.571429	7
5	C	0.416667	9

2. How to customize the default plot settings of Matplotlib w.r.t runtime configuration and stylesheets? Give example.

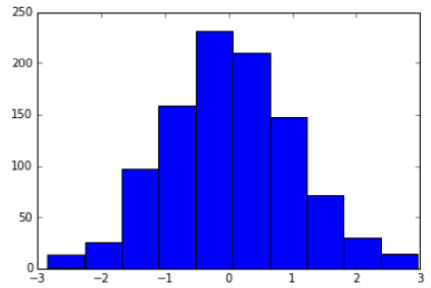
   a. Each time Matplotlib loads, it defines a runtime configuration (rc) containing the default styles for every plot element we create.
   b. We can adjust this configuration at any time using the **plt.rc** convenience routine.
   c. To modify the rc parameters, we'll start by saving a copy of the current rcParams

dictionary, so we can easily reset these changes in the current session:

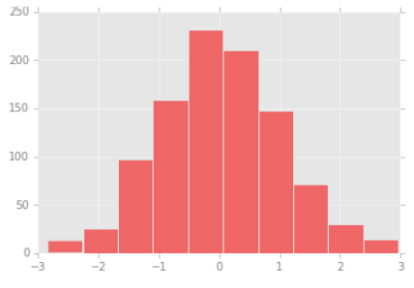
IPython_default = plt.rcParams.copy()

   d. Now we can use the plt.rc function to change some of these settings:

```
from matplotlib import cycler

colors = cycler('color',
                ['#EE6666', '#3388BB', '#9988DD',
                 '#EECC55', '#88BB44', '#FFBBBB'])

plt.rc('axes', facecolor='#E6E6E6', edgecolor='none',
axisbelow=True, grid=True, prop_cycle=colors)

plt.rc('grid', color='w', linestyle='solid')
plt.rc('xtick', direction='out', color='gray')
plt.rc('ytick', direction='out', color='gray')
plt.rc('patch', edgecolor='#E6E6E6')
plt.rc('lines', linewidth=2)
```



A histogram in Matplotlib's default style



A customized histogram using rc settings

3. Write Python code to create an account object with two functions and constructor. [Hint :The "deposit" method takes an amount to deposit as an argument, adds it to the account balance, and prints a message with the new balance. The "withdraw" method takes an amount to withdraw as an argument, checks if the account has sufficient funds, and either subtracts the amount from the account balance or prints an error message. The constructor takes an initial balance as an argument and initializes the account with that balance.]

```
class Account:
    def __init__(self, initial_balance):
        self.balance = initial_balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposit of {amount} successful. New balance: {self.balance}")

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
            print(f"Withdrawal of {amount} successful. New balance: {self.balance}")
        else:
            print(f"Error: Insufficient balance. Cannot withdraw {amount}. Current balance: {self.balance}")

# create account with initial balance of 1000
acct = Account(1000)

# deposit 500
acct.deposit(500)

# withdraw 200
acct.withdraw(200)
```

4. Write a class Rectangle containing numeric attributes width and height. This class should contain another attribute corner which is an instance of another class Point. Implement following functions –

i) function to print corner point as an ordered-pair
ii) function find_center() to compute center point of the rectangle
iii) function resize() to modify the size of rectangle

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f"({self.x}, {self.y})"

class Rectangle:
    def __init__(self, width, height, x, y):
        self.width = width
        self.height = height
        self.corner = Point(x, y)

    def __str__(self):
        return f"Rectangle with width {self.width} and height {self.height} at {self.corner}"

    def find_center(self):
        x = self.corner.x + (self.width / 2)
        y = self.corner.y + (self.height / 2)
        return Point(x, y)

    def resize(self, width, height):
        self.width = width
        self.height = height
def print_corner(self):
        print(self.corner)

# create a rectangle with width 5, height 3, and corner at (0, 0)
r = Rectangle(5, 3, 0, 0)

# print the rectangle's corner
r.print_corner()

# compute and print the center of the rectangle
center = r.find_center()
print(f"Center of rectangle: {center}")

# resize the rectangle to have width 7 and height 4
r.resize(7, 4)

# print the rectangle's new dimensions and corner
print(r)
r.print_corner()

output:
(0, 0)
Center of rectangle: (2.5, 1.5)
Rectangle with width 7 and height 4 at (0, 0)
(0, 0)
```

5. Discuss any five methods to handle the missing data with python code
Elaborate on Seaborn versus Matplotlib with suitable examples.

i) The isnull() method returns a DataFrame object where all the values are replaced with a Boolean value True for NULL values, and otherwise False.

dataframe.isnull()

ii) notnull is a pandas function that will examine one or multiple values to validate that they are not null. In Python, null values are reflected as NaN (not a number) or None to signify no data present. . notnull will return False if either NaN or None is detected. If these values are not present, it will return True.

iii) The dropna() method removes the rows that contains NULL values.

iv) The fillna() method replaces the NULL values with a specified value.

The fillna() method returns a new DataFrame object unless the inplace parameter is set to True, in that case the fillna() method does the replacing in the original DataFrame instead.

v). 'The ffill() method replaces the NULL values with the value from the previous row (or previous column, if the axis parameter is set to 'columns').

Seaborn library is basically based on Matplotlib. Here is a detailed comparisonbetween the two:

	Seaborn	**Matplotlib**
Functionality	Seaborn, on the other hand, provides a variety of visualization patterns. It uses fewer syntax and has easily interesting default themes. It specializes in statistics visualization and is used if one has to summarize data in visualizations and also show the distribution in the data.	Matplotlib is mainly deployed for basic plotting. Visualization using Matplotlib generally consists of bars, pies, lines, scatter plots andso on.
Handling Multiple Figures	Seaborn automates the creation of multiple figures. This sometimes leads to OOM (out of memory) issues.	Matplotlib has multiple figures can be opened, but need to be closed explicitly. plt.close() only closes the current figure. plt.close('all') would close them all.
Visualization	Seaborn is more integrated for working with Pandas data frames. It extends the Matplotlib library for creating beautiful graphics with Python using a more straightforward set of methods.	Matplotlib is a graphics package for data visualization in Python. It is well integrated with NumPy and Pandas. The pyplot module mirrors the MATLAB plotting commands closely. Hence, MATLAB users can easily transit to plotting with Python.
Data frames and Arrays	Seaborn works with the dataset as a whole and is much more intuitive than Matplotlib. For Seaborn, replot() is the entry API with 'kind' parameter to specify the type of plot which could be line, bar, or many of the other types. Seaborn is not stateful. Hence, plot() would require passing the object.	Matplotlib works with data frames and arrays. It has different stateful APIs for plotting. The figures and aces are represented by the object and therefore plot() like calls without parameters suffices, without having to manage parameters.
Flexibility	Seaborn avoids a ton of boilerplate by providing default themes which are commonly used.	Matplotlib is highly customizable and powerful.
Use Cases	Seaborn is for more specific use cases. Also, it is Matplotlib under the hood. It is specially meant for statistical plotting.	Pandas uses Matplotlib. It is a neat wrapper around Matplotlib.

6. Discuss any 10 built in String methods with an example? Explain Reshape and Pivot operations with example.

The various String methods are:

1. lower(): Converts all uppercase characters in a string into lowercase

Syntax*: string.lower()*

```
txt = "Hello my FRIENDS"

x = txt.lower()

print(x)
```

O/P : hello my friends

2. upper(): Converts all lowercase characters in a string into uppercase

Syntax: string.upper()

```
txt = "Hello my FRIENDS"

x = txt.lower()

print(x)
```

O/P : HELLO MY FRIENDS

3. swapcase(): Swap the cases of all characters in a string
   Syntax: string_name.swapcase()
   txt= "HelloMyNameIsPETER"

   x=txt.swapcase()

   print(x)
   O/P : hELLOmYnAMEiSpeter
4. Capitalize():Converts the first character of the string to a capital (uppercase) letter
   Syntax: string_name.capitalize()
   txt= "hello,andwelcometomyworld."

   x=txt.capitalize()

   print (x)
   O/P: Hello,andwelcometomyworld
5. count():Returns the number of occurrences of a substring in the string.
   Syntax: string.count(*value, start, end*)
   txt= "Iloveapples
   txt.count("apple")

   print(x)
   O/P: 1
6. Replace():Replaces all occurrences of a substring with another substring
   Syntax: *string*.replace(*oldvalue, newvalue, count*)
   txt= "Ilikebananas"

   x=txt.replace("bananas", "apples")

   print(x)
   O/P: Ilikeapples
7. Endswith():Returns “True” if a string ends with the given suffix
   Syntax: string.endswith(*value, start, end*)
   txt= "Hello,welcometomyworld."

   x=txt.endswith(".")

   print(x)
   O/P: True
8. Isalnum():Checks whether all the characters in a given string is alphanumeric or not
   Syntax: *string*.isalnum()
   txt= "Company12"

   x=txt.isalnum()

   print(x)
   O/P: True
9. Isalpha():Returns “True” if all characters in the string are alphabets
   Syntax: *string*.isalpha()
   txt= "CompanyX"

   x=txt.isalpha()

   print(x**)**
   O/P- True
10. **Isdigit():**Returns “True” if all characters in the string are digits
    Syntax: string.isdigit()
    Ex.
    txt=”5000”
    x=txt.isdigit()
    print(x)
    O/P- False

A *pivot table* is a similar operation that is commonly seen in spreadsheets and other programs that operate on tabular data.

The pivot table takes simple column-wise data as input, and groups the entries into a two-dimensional table that provides a multidimensional summarization of the data.

The difference between pivot tables and `GroupBy` : pivot tables as essentially a *multidimensional* version of `GroupBy` aggregation. That is, you split-apply-combine, but both the split and the combine happen across not a one-dimensional index, but across a two-dimensional grid.

Pivot Table Syntax

Here is the equivalent to the preceding operation using the `pivot_table` method of `DataFrame`s:

```
In [5]: titanic.pivot_table('survived', index='sex', columns='class')
```

Out[5]:

sex	class	First	Second	Third
female		0.968085	0.921053	0.500000
male		0.368852	0.157407	0.135447

his is eminently more readable than the `groupby` approach, and produces the same result. As you might expect of an early 20th-century transatlantic cruise, the survival gradient favors both women and higher classes. First-class women survived with near certainty (hi, Rose!), while only one in ten third-class men survived (sorry, Jack!).

Multi-level pivot tables

Just as in the `GroupBy`, the grouping in pivot tables can be specified with multiple levels, and via a number of options. For example, we might be interested in looking at age as a third dimension. We'll bin the age using the `pd.cut` function:

```
In [6]: age = pd.cut(titanic['age'], [0, 18, 80])
        titanic.pivot_table('survived', ['sex', age], 'class')
```

Out[6]:

sex	age	class	First	Second	Third
female	(0, 18]		0.909091	1.000000	0.511628
	(18, 80]		0.972973	0.900000	0.423729
male	(0, 18]		0.800000	0.600000	0.215686
	(18, 80]		0.375000	0.071429	0.133663

7. Write a python program to demonstrate data visualization for Line Plot, Histogram, heatmap, Boxplot, and Distribution Plots

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

cars_data = pd.read_csv('Toyota.csv', index_col=0,na_values=["??","????"])
fig, ax = plt.subplots()
ax.plot(cars_data['Age'],cars_data['Price'],c='red')
ax.hist(cars_data['Age'],cars_data['Price'],c='red')
sns.heatmap(cars_data['Age'],cars_data['Price'],c='red')
sns.boxplot(cars_data['Fuel Type'])
sns.distplot(cars_data['Age'])
```

8. Write a Pandas program to create a data frame with the test data , split the dataframe by school code and get mean, min, and max value of i) age ii) weight for each school.

Test Data:

	school	class	name	age	height	weight
S1	s001	V	Ram	12	173	35
S2	s002	V	Kiran	12	192	32
S3	s003	VI	Ryan	13	186	33
S4	s001	VI	Bhim	13	167	30
S5	s002	VI	Sita	14	151	31
S6	s004	V	Bhavana	12	159	32

Ans

```
import pandas as pd

# create test data frame
data = {
   'name': ['Ram','Kiran','Ryan','Bhim','Sita','Bhavana'],
   'age': [12,12,13,13,14,12],
   'weight': [35,32,33,30,31,32],
 'height' : [173,192,186,167,151,159]
   'school_code': ['s001','s002','s003','s001','s002','s004']
 's_id' : ['S1','S2','S3','S4','S5','S6']
}
df = pd.DataFrame(data)

# split data frame by school code
groups = df.groupby('school_code')

# get mean, min, and max values of age and weight for each school
for name, group in groups:
   print(f"School {name}:")
   print(f"Mean age: {group['age'].mean()}")
   print(f"Minimum age: {group['age'].min()}")
   print(f"Maximum age: {group['age'].max()}")
   print(f"Mean weight: {group['weight'].mean()}")
   print(f"Minimum weight: {group['weight'].min()}")
   print(f"Maximum weight: {group['weight'].max()}")
   print()
```

9. Write python code for the following using Pandas:
   I. read from and write into CSV
   II. read from and write into JSON

```
import pandas as pd

# read from CSV file
df_csv = pd.read_csv('example.csv')

# write into CSV file
df_csv.to_csv('example_output.csv', index=False)

# read from JSON file
df_json = pd.read_json('example.json')

# write into JSON file
df_json.to_json('example_output.json', orient='records')
```

10. Write python code to interact with database and perform the following task
    I. Create table
    II. Insert 3 records into table
    III. Display all records
    IV. ORDER BY statement to sort the result n ascending order.

V. Update existing records.

```
import sqlite3

# connect to database
conn = sqlite3.connect('example.db')

# create table
conn.execute('''CREATE TABLE IF NOT EXISTS example_table
            (ID INT PRIMARY KEY NOT NULL,
            NAME TEXT NOT NULL,
            AGE INT NOT NULL,
            EMAIL TEXT NOT NULL);''')

# insert 3 records into table
conn.execute("INSERT INTO example_table (ID, NAME, AGE, EMAIL) \
             VALUES (1, 'Alice', 25, 'alice@example.com')")
conn.execute("INSERT INTO example_table (ID, NAME, AGE, EMAIL) \
             VALUES (2, 'Bob', 30, 'bob@example.com')")
conn.execute("INSERT INTO example_table (ID, NAME, AGE, EMAIL) \
             VALUES (3, 'Charlie', 35, 'charlie@example.com')")

# commit changes
conn.commit()

# display all records
cursor = conn.execute("SELECT * FROM example_table")
for row in cursor:
    print(f"ID = {row[0]}, NAME = {row[1]}, AGE = {row[2]}, EMAIL = {row[3]}")

# close database connection
conn.close()
#Order By
cursor = conn.execute("SELECT * FROM example_table ORDER BY NAME”)
conn.commit()

 #Update
cursor = conn.execute("UPDATE example_table SET AGE = 40 WHERE AGE=35”)
```