CMR
INSTITUTE OF TECHNOLOGY

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

Internal Assessment Test III – March 2024

Sub:	Data Analytics using Python							Sub Code:	22MCA31
Date:	12/03/2024	Duration:	90 mins	Max Marks:	50	Sem:	III	Branch:	MCA

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

	PART I	MARKS	OBE	
			CO	RBT
1	Explain webbrowser module and requests module with programs for each.	5+5	CO3	L4
	OR			
2	What is beautiful soup. Explain the key features of beautiful Soup.	5+5	CO4	L4
	PART II			
3	Demonstrate get() and how get() works in web scraping with a program	10	CO4	L4
	OR			
4	Explain the following in time series with program examples: a) Shifting of data b) saving plots in a file	5+5	CO2	L2
	PART III			
5	Explain the date and time data types in time series with a program example.	10	CO2	L2
	OR			
6	Explain period and period arithmetic in time series with a program example	5+5	CO2	L3
	PART IV			
7	Discuss about indexing,selection and subsetting in time series with program examples.	3+3+3+1	CO3	L3
	OR			
8	What is resampling?Explain the types of resampling with a program example.	10	CO3	L3
	PART V			
9	Write a Python program to demonstrate the generation of linear regression models.	10	CO4	L4
	OR			
10	Write a Python program to demonstrate the generation of logistic regression models.	10	CO4	L4

1. <u>Webbrowser module and requests module:</u>

The `webbrowser` and `requests` modules are both Python libraries commonly used in web development and automation tasks, but they serve different purposes.

1. `webbrowser` Module:

The `webbrowser` module allows you to launch and control a web browser from your Python script. It provides a simple interface to open web pages in a browser window. Below is a simple program demonstrating how to use the `webbrowser` module:

```python
import webbrowser

# URL to open in the browser
url = "https://www.example.com"

# Open the URL in a web browser
webbrowser.open(url)
```

This program will open the specified URL in the default web browser on your system.

2. `requests` Module:

The `requests` module allows you to send HTTP requests easily in Python. It provides methods to make GET, POST, PUT, DELETE, and other types of HTTP requests, and handle responses from web servers. Below is a simple program demonstrating how to use the `requests` module to make a GET request:

```python
import requests

# URL to send GET request
url = "https://jsonplaceholder.typicode.com/posts/1"
```

```
# Send GET request to the URL
response = requests.get(url)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Print the response content (usually JSON or HTML)
    print(response.json())
else:
    # Print an error message if the request was unsuccessful
    print("Error:", response.status_code)
```

This program sends a GET request to the specified URL and prints the response content if the request was successful.

Both of these modules are useful in different scenarios. The `webbrowser` module is handy for tasks where you need to interact with a web browser, such as opening a URL for user interaction. On the other hand, the `requests` module is useful for tasks involving HTTP requests, such as web scraping, API interactions, etc.

2. <u>Beautiful Soup and key features of beautiful soup:</u>

Beautiful Soup is a Python library used for web scraping data from HTML and XML files. It provides tools for parsing HTML and XML documents, navigating the parsed tree, and extracting data from them. Beautiful Soup creates a parse tree from the parsed HTML or XML file that can be traversed easily to extract the required information.

Key features of Beautiful Soup include:

1. **Parsing HTML and XML**: Beautiful Soup can parse both HTML and XML files, making it versatile for a wide range of web scraping tasks.

2. **Easy Navigation**: It provides simple and intuitive methods to navigate the parse tree, such as finding elements by tag name, class, id, or attributes.

3. **Robust Handling of Malformed Markup**: Beautiful Soup can handle poorly formatted HTML and XML documents, making it resilient to errors commonly encountered in web scraping tasks.

4. **Powerful Searching**: It offers powerful methods to search for specific elements in the parse tree, including CSS selectors, regular expressions, and custom functions.

5. **Data Extraction**: Beautiful Soup allows you to extract data from HTML and XML files effortlessly, whether it's text, attributes, or other content within the elements.

6. **Encoding Detection**: It automatically detects the encoding of the document and converts it to Unicode, making it easier to work with different encodings.

7. **Support for Multiple Parsers**: Beautiful Soup supports different parsers, including the built-in Python parsers (such as lxml and html.parser) and third-party parsers like html5lib, providing flexibility based on specific needs and performance considerations.

8. **Integration with Other Libraries**: It can be seamlessly integrated with other Python libraries such as Requests for fetching web pages and Pandas for data manipulation and analysis.

9. **Open Source and Active Community**: Beautiful Soup is an open-source project with an active community, ensuring ongoing development, bug fixes, and support.

Overall, Beautiful Soup simplifies the process of web scraping by providing an easy-to-use interface and robust features for extracting data from HTML and XML documents.

3. Get()Mand how get() works:

To demonstrate how the `get()` method works in web scraping, let's use the `requests` library to fetch the HTML content of a web page. We'll then use Beautiful Soup to parse the HTML and extract specific information from it. Here's a simple program that demonstrates the `get()` method in action:

```python
import requests
from bs4 import BeautifulSoup

# URL of the web page to scrape
url = "https://en.wikipedia.org/wiki/Web_scraping"

# Send a GET request to fetch the HTML content of the web page
response = requests.get(url)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Parse the HTML content using Beautiful Soup
    soup = BeautifulSoup(response.content, "html.parser")

    # Find the title of the web page
    title = soup.find("title").get_text()
    print("Title:", title)

    # Find all the links on the page
    links = soup.find_all("a")
    print("\nLinks on the page:")
    for link in links:
        href = link.get("href")
        if href:
            print(href)
else:
    # Print an error message if the request was unsuccessful
    print("Error:", response.status_code)
```

In this program:

1. We import the `requests` library to send HTTP requests and the `BeautifulSoup` class from the `bs4` module to parse HTML content.

2. We specify the URL of the web page we want to scrape.
3. We use the `requests.get()` method to send a GET request to the specified URL and store the response in the `response` variable.
4. We check if the request was successful (status code 200). If successful, we proceed with parsing the HTML content.
5. We create a Beautiful Soup object `soup` by passing the HTML content (`response.content`) and the parser type (`html.parser`).
6. We use Beautiful Soup methods like `find()` and `find_all()` to locate specific elements in the HTML document. In this example, we find the `<title>` tag to extract the title of the web page and all `<a>` tags to extract the links.
7. We use the `get_text()` method to extract the text content of the `<title>` tag and the `get()` method to extract the `href` attribute of each `<a>` tag.
8. Finally, we print the title of the web page and all the links found on the page.

This program demonstrates how to use the `get()` method in web scraping to extract information from a web page fetched using the `requests` library.

4.Explain:
a) Shifting of data
b) saving plots in a file

a) Shifting of Data in Time Series:

Shifting of data in time series refers to moving the data points forwards or backwards in time. This operation is useful for creating lag or lead variables, which are often required in time series analysis.

In Python, you can use the `shift()` method available in pandas, a popular library for data manipulation and analysis. This method allows you to shift the index of a DataFrame by a specified number of periods.

Here's an example demonstrating how to shift data in a time series using pandas:

```python
import pandas as pd

# Create a sample time series data
data = {'Date': ['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-04'],
        'Value': [10, 20, 30, 40]}
df = pd.DataFrame(data)
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)

# Shift the data forward by one period
df['Value_shifted'] = df['Value'].shift(periods=1)

print(df)
```

This will output:

```
            Value  Value_shifted
Date
2022-01-01     10            NaN
2022-01-02     20           10.0
2022-01-03     30           20.0
```

```
2022-01-04  40  30.0
```

As you can see, the `Value_shifted` column contains the values of the `Value` column shifted forward by one period.

b) Saving Plots in a File:

In Python, you can save plots generated using libraries like matplotlib or seaborn to various file formats such as PNG, PDF, SVG, etc. This is useful for sharing or further processing the plots.

Here's an example demonstrating how to save a plot generated using matplotlib:

```python
import matplotlib.pyplot as plt

# Create sample data
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 35]

# Create a line plot
plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Sample Plot')

# Save the plot to a file (PNG format)
plt.savefig('sample_plot.png')

# Show the plot
plt.show()
```

This code will generate a line plot and save it as a PNG file named `sample_plot.png`. You can change the filename and format as needed. The `savefig()` function allows you to specify the filename and file format for saving the plot.

These examples illustrate how to perform shifting of data in time series and save plots generated in Python to a file.

5.Date and time datatypes

In time series analysis, handling date and time data accurately is crucial. Python provides several data types and libraries to work with date and time data effectively. Two commonly used data types for representing date and time information are `datetime` and `Timestamp`.

1. `datetime` Data Type:

The `datetime` module in Python provides classes for manipulating date and time. It includes the `datetime` class, which represents a specific date and time. Here's an example of using `datetime` to work with date and time data:

```python
import datetime
```

```
# Create a datetime object representing a specific date and time
dt = datetime.datetime(2022, 1, 1, 12, 30, 15)

# Print the datetime object
print("Datetime:", dt)

# Access individual components of the datetime object
print("Year:", dt.year)
print("Month:", dt.month)
print("Day:", dt.day)
print("Hour:", dt.hour)
print("Minute:", dt.minute)
print("Second:", dt.second)
```

Output:
```
Datetime: 2022-01-01 12:30:15
Year: 2022
Month: 1
Day: 1
Hour: 12
Minute: 30
Second: 15
```

2. `Timestamp` Data Type:

The `Timestamp` data type is part of the pandas library and is specifically designed to handle date and time data efficiently in time series analysis. It extends the functionality of `datetime` and provides additional features for working with time series data. Here's an example of using `Timestamp`:

```python
import pandas as pd

# Create a Timestamp object representing a specific date and time
ts = pd.Timestamp('2022-01-01 12:30:15')

# Print the Timestamp object
print("Timestamp:", ts)

# Access individual components of the Timestamp object
print("Year:", ts.year)
print("Month:", ts.month)
print("Day:", ts.day)
print("Hour:", ts.hour)
print("Minute:", ts.minute)
print("Second:", ts.second)
```

Output:
```
Timestamp: 2022-01-01 12:30:15
Year: 2022
Month: 1
```

Day: 1
Hour: 12
Minute: 30
Second: 15
```

### Comparison:

- The `datetime` module provides basic functionalities for working with date and time data, including creating datetime objects, accessing individual components, and performing basic arithmetic operations.
- The `Timestamp` data type in pandas builds upon the `datetime` module and offers additional features tailored for time series analysis, such as handling time zones, frequency conversion, and alignment with other time series data.

Both `datetime` and `Timestamp` are widely used in time series analysis, and the choice between them depends on specific requirements and the tools being used for analysis.

**6.Period and period index:**

In time series analysis, a period represents a fixed frequency of time. It could be a day, month, year, etc., and is typically used to aggregate and analyze data over regular intervals. Period arithmetic involves performing arithmetic operations such as addition and subtraction on periods.

In Python, the `pandas` library provides the `Period` class to work with periods efficiently. It allows you to create, manipulate, and perform arithmetic operations on periods. Here's an example demonstrating period arithmetic using `pandas`:

```python
import pandas as pd

# Create two periods
period1 = pd.Period('2022-01')  # January 2022
period2 = pd.Period('2022-03')  # March 2022

# Print the periods
print("Period 1:", period1)
print("Period 2:", period2)

# Perform arithmetic operations on periods
# Addition
add_result = period1 + 2  # Add two periods to period1
print("Addition Result:", add_result)

# Subtraction
sub_result = period2 - period1  # Difference between period2 and period1
print("Subtraction Result:", sub_result)
```

Output:
```
Period 1: 2022-01
Period 2: 2022-03
Addition Result: 2022-03
Subtraction Result: <2 * MonthEnds>
```
```

In this example:

- We create two `Period` objects representing January 2022 (`period1`) and March 2022 (`period2`).
- We perform arithmetic operations on periods:
  - Addition: We add two periods to `period1`, resulting in March 2022.
  - Subtraction: We calculate the difference between `period2` and `period1`, resulting in a new period object representing the difference between the two periods. `<2 * MonthEnds>` indicates that the difference is two months.

Period arithmetic in `pandas` is flexible and supports various arithmetic operations such as addition, subtraction, multiplication, and division. It also handles edge cases such as overflow and underflow of periods automatically. This makes it convenient for working with time series data at different frequencies and performing calculations based on fixed intervals of time.

7. <u>Indexing,Selection and subsetting:</u>

In time series analysis, indexing, selection, and subsetting are essential operations for accessing and manipulating data within a time series dataset. Python libraries such as pandas provide powerful tools for performing these operations efficiently.

Indexing:
Indexing refers to the process of accessing specific elements or subsets of elements in a time series dataset. In pandas, time series data is often indexed by timestamps or periods.

```python
import pandas as pd

# Create a sample time series DataFrame
data = {'value': [10, 20, 30, 40, 50]}
dates = pd.date_range(start='2022-01-01', periods=5)
ts = pd.DataFrame(data, index=dates)

# Indexing using timestamp
print(ts.loc['2022-01-03'])  # Access data for a specific timestamp
```

Selection:
Selection involves choosing specific rows or columns based on certain criteria. In time series analysis, this can be done based on timestamps, periods, or other conditions.

```python
# Selecting a range of timestamps
print(ts.loc['2022-01-02':'2022-01-04'])

# Selecting data based on conditions
print(ts[ts['value'] > 30])  # Select rows where value is greater than 30
```

Subsetting:
Subsetting refers to extracting a subset of the time series dataset based on specific conditions or criteria. This could involve selecting a range of timestamps, filtering based on certain values, or extracting specific columns.

```python
```

```
# Subsetting based on a range of timestamps
print(ts['2022-01-02':'2022-01-04'])

# Subsetting based on conditions and specific columns
print(ts.loc[ts['value'] > 30, ['value']])
```

These are some basic examples demonstrating indexing, selection, and subsetting operations in time series analysis using pandas. These operations allow analysts and data scientists to effectively explore and analyze time series data, extract relevant information, and perform various statistical and machine learning tasks.

8. <u>Resampling:</u>

Resampling is a technique used in time series analysis to change the frequency of the time series data. It involves aggregating or interpolating the data over different time intervals. Resampling is useful for various purposes such as downsampling (reducing the frequency of data points) or upsampling (increasing the frequency of data points), as well as for adjusting the time periods to align with specific requirements.

There are two main types of resampling:

1. **Upsampling**: Increasing the frequency of data points by interpolating or filling in missing values for the new time intervals.
2. **Downsampling**: Decreasing the frequency of data points by aggregating multiple data points into a single data point for the new time intervals.

Here's a program example demonstrating both types of resampling using pandas:

```python
import pandas as pd
import numpy as np

# Create a sample time series DataFrame
np.random.seed(0)
dates = pd.date_range(start='2022-01-01', periods=10, freq='D')
data = {'value': np.random.randint(1, 100, size=10)}
ts = pd.DataFrame(data, index=dates)

# Upsample to hourly frequency and interpolate missing values
upsampled = ts.resample('H').asfreq()
interpolated = upsampled.interpolate(method='linear')

print("Upsampled and Interpolated Data:")
print(interpolated)

# Downsample to weekly frequency and aggregate with mean
downsampled = ts.resample('W').mean()

print("\nDownsampled Data (Mean Aggregation):")
print(downsampled)
```

Output:
```
Upsampled and Interpolated Data:
                     value
```

```
2022-01-01 00:00:00   47.000000
2022-01-01 01:00:00   47.400000
2022-01-01 02:00:00   47.800000
2022-01-01 03:00:00   48.200000
...
2022-01-09 20:00:00   85.714286
2022-01-09 21:00:00   86.142857
2022-01-09 22:00:00   86.571429
2022-01-09 23:00:00   87.000000

Downsampled Data (Mean Aggregation):
                value
2022-01-02  50.333333
2022-01-09  66.571429
```

In this example:

- We create a sample time series DataFrame `ts` with 10 data points spanning 10 days.
- We upsample the data to hourly frequency using the `resample()` method with a frequency of 'H', and then interpolate the missing values using linear interpolation.
- We downsample the data to weekly frequency using the `resample()` method with a frequency of 'W', and aggregate the data using the mean function to compute the average value for each week.

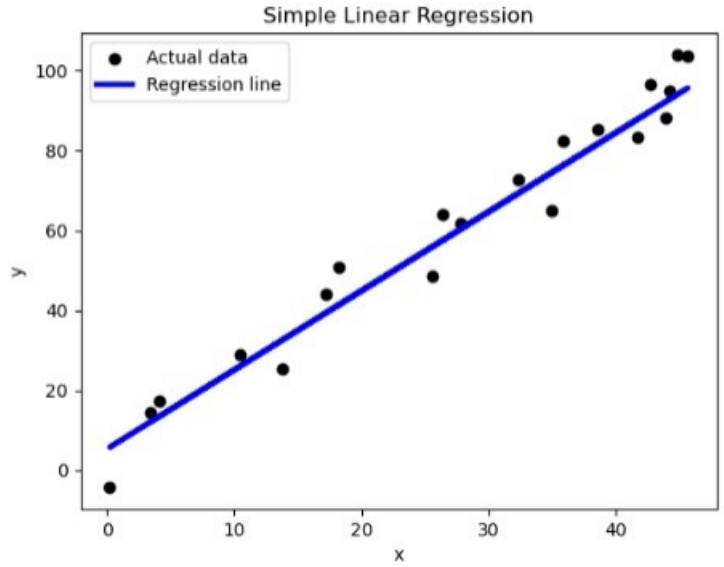
Resampling allows us to adjust the frequency of time series data according to our analysis requirements and can be useful for various applications such as data visualization, forecasting, and modeling.

9. <u>Linear regression:</u>

```
import matplotlib.pyplot as plt
from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept,r,p,std_err = stats.linregress(x, y)
def myfunc(x):
return slope * x + intercept
#code
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
# Load the dataset from the CSV file
df = pd.read_csv('sample_dataset.csv')
# Extract 'x' and 'y' columns
X = df[['x']]
y = df[['y']]
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Initialize the Linear Regression model
model = LinearRegression()
```

```
# Train the model
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False) # RMSE is the square root of MSE
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
cor_coe=df['x'].corr(df['y'])
# Print evaluation metrics
print(f'Mean Squared Error (MSE): {mse:.2f}')
print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
print(f'Mean Absolute Error (MAE): {mae:.2f}')
print(f'R-squared (R2): {r2:.2f}')
print(f'Coefficient_coefficent:{cor_coe:.2f}')
# Visualize the regression line
plt.scatter(X_test, y_test, color='black', label='Actual data')
plt.plot(X_test, y_pred, color='blue', linewidth=3, label='Regression line')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()
```

```
Mean Squared Error (MSE): 40.13
Root Mean Squared Error (RMSE): 6.33
Mean Absolute Error (MAE): 5.73
R-squared (R2): 0.96
Correlation Coefficient: 0.98
```



10. <u>Logistic regression:</u>

```
import numpy as np #to perform array
dataset = pd.read_csv('/DSWTC.csv')
print(dataset.shape)
print(dataset.head(5))
dataset.info()
X = dataset.iloc[:, :-1].values
X
Y = dataset.iloc[:, -1].values
```

```
Y
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.25, random_state = 1)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix,
classification_report, roc_curve, roc_auc_score
import matplotlib.pyplot as plt
# 1. Logistic Regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state=123)
lr.fit(X_train, y_train)
# Make predictions
predictions = lr.predict(X_test)
cm=confusion_matrix(y_test,predictions)
print("Confusion Matrix: ")
print(cm)
cr=classification_report(y_test,predictions)
print("Classification Report: ")
print(cr)
# Calculate metrics
lra = accuracy_score(predictions, y_test)*100
print("Accuracy: ",lra)
lrp = precision_score(predictions, y_test)*100
print("Precision: ",lrp)
lrr = recall_score(predictions, y_test)*100
print("Recall: ",lrr)
lrf = f1_score(predictions, y_test)*100
print("F1 Score: ",lrf)
y_pred_proba = lr._predict_proba_lr(X_test)[::,1]
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
auc = roc_auc_score(y_test, y_pred_proba)
#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.title("ROC CURVE")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

Output:

```
Confusion Matrix:
[[1 3]
 [2 3]]
Classification Report:
              precision    recall  f1-score   support

           0       0.33      0.25      0.29         4
           1       0.50      0.60      0.55         5

    accuracy                           0.44         9
   macro avg       0.42      0.42      0.42         9
weighted avg       0.43      0.44      0.43         9

Accuracy:  44.44444444444444
Precision:  60.0
Recall:  50.0
F1 Score:  54.54545454545454
```