Name: Sachin Rajbhar DATA SCIENCE 31/05/23
Class: BCA IV[th] Sem ASSIGNMENT No. 2
Roll no: 41122139 SOLUTIONS

Q1 Write a comparison between IPython and Jupyter notebook. (2)

Ans

Feature	IPython	Jupyter
Execution Environment	Interactive Shell	Web-based Interactive notebook environment
Programming Language	Primarily Python	Supports multiple languages
Cell Execution	Sequential execution	Independent cell execution
Collaboration	Limited support for collab. editing	Built-In collaborative features
Visualization & extension	Limited support	Rich & Vast support

Q2 What is the need of streaming the data? Explain data uploading & streaming data with example. (2)

Ans Streaming data is needed to handle large volumes of data generated in real-time & enables immediate processing, analysis, & decision-making.

Data Uploading :- It involves transferring data from a local system to a target system, ~~table data streaming~~ such as server or cloud
Ex:- weather monitoring system

Data Streaming :- It refers to continuous & real-time transmission of data from a source to a target system.
Ex:- Social Media (Interaction App)

Streaming data allows for timely insights & actions based on the incoming data.

Q3 Define magic commands along with the magic & cell magic. Explain any 5 of magic commands. (5)

Ans Magic commands in Jupyter Notebook & IPython provide additional functionality beyond standard Python syntax.

Two types of magic commands : 'line magic' are used for single-line operations & are executed immediately, affecting only that line.

'Cell magic' commands are used for multi-line operations & are executed on entire cell, including all lines.

- Line magic

% run	% load	% timeit	% who	% reset
to run a Python script	to load external file	measure execution time	display all global variables	reset the namespace
% run script.py	% load file.py	% timeit func()	% who	% reset -f

- ~~Magic~~ Cell Magic

%%writefile	%% time	%%html	%%bash	%%latex
to write cell content to file	to measure & display elapsed time	to write & run html code directly	to run bash command for shell scripts	to render & write LaTeX code for maths formulas
%%writefile file.txt	%% time for i in range(100): print(i)	%%html H W!	%%bash ls -l	%%latex $E = mc^2$

Q4 Discuss the following in context of Jupyter notebook (4)

a) Advantages & disadvantages of Jupyter notebook

Ans Advantages :-

1. Interactive environment & easy
2. Support for multiple programming languages
3. Rich documentation & visualization capabilities
4. Easy sharing & collaboration
5. Reproducibility

Disadvantages:-

1. Steep learning curve
2. Limited debugging capabilities
3. Resource - Intensive
4. Version control challenges

b) Installation of jupyter notebook

Ans Step 1:- Install Python (https://www.python.org) & pip

Step 2:- pip install jupyter

Step 3:- jupyter notebook

c) Components of Jupyter Notebook

→ Notebook Dashboard (Landing page)
→ Notebook Editor (create & Edit Notebook)
→ Kernel (runs & executes code)
→ Markdown cells (add formatted text, list, link & images)
→ Code cells (used to write & execute Code)

d) Types of cells & cell styling in Jupyter Notebook

Ans 2 main types are given:-

Code Cell : Used to write & execute code.
User can press Shift + Enter & output will be displayed below the cell.

Markdown cell : Used to write formatted text, explanation, documentation or even LaTeX equations
This also can be render using Shift + Enter

In addition to the two main types of cells, Jupyter notebook also provide cell styling options.
For ex:- add a cell heading, apply bold, italic, lists etc. to make it visually appealing.

Q5 Write a python program to: (5)

a) Read contents of a JSON file in Jupyter Notebook with or without Pandas.

b) Create JSON data & read it in a Pandas Dataframe.

c) Parse a JSON file (Hint: convert from JSON to Python)

Ans
```
import json
import pandas as pd

# a. Read contents of a JSON file in Jupyter Notebook with or
# without Pandas.
def read_json_file(file_path):
    with open(file_path, 'r') as file:
        data = json.load(file)
    return data

json_data = read_json_file('data.json')
print(json_data)

#b. Create JSON data & read it in a pandas Dataframe
def create_json_data():
    data = [
        {'name': 'John', 'age': 30, 'city': 'New York'},
        {'name': 'Alice', 'age': 25, 'city': 'San Francisco'},
        {'name': 'Bob', 'age': 35, 'city': 'Chicago'}
    ]
    return data

json_data = create_json_data()
df = pd.DataFrame(json_data)
print(df)
```

c. Parse a JSON file (convert from JSON to Python)

```
def parse_json_file(file_path):
    with open(file_path, 'r') as file:
        json_string = file.read()
        data = json.loads(json_string)
    return data

parsed_data = parse_json_file('data.json')
print(parsed_data)
```

Parsing means extracting meaningful information from data
format like JSON, XML or HTML & involves several steps including
tokenization, lexical / syntax / semantics analysis.

6) write a python code using Pandas styling to : (5)

a. Create a Dataframe of ten rows, four columns with random values. Write a Pandas program to highlight the negative numbers red & positive numbers black

b. Create a Dataframe of ten rows, four columns with random values. Write a Pandas program to highlight the maximum value in each column.

```
import pandas as pd
import numpy as np

# a. Create a Dataframe of ten rows, four columns with
# random values.
np.random.seed(1)
df = pd.DataFrame(np.random.randn(10, 4), columns = ['A', 'B', 'C', 'D'])

def color_negative_red(val):
    color = 'red' if val<0 else 'black'
    return 'color: {}'.format(color)
```

```
styled_df = df.style.applymap(color_negative_red)
print(styled_df)
```

```
# b) Create a Dataframe of ten rows, four columns
# with random values.
np.random.seed(12)
df = pd.DataFrame(np.random.randint(0,100, size=(10,4)),
                  columns = ['A', 'B', 'C', 'D'])

def highlight_max_value(val):
    is_max = val == df[val.index].max()
    return ['background-color: yellow' if v else '' for v in
                                                  is_max]

styled_df = df.style.apply(highlight_max_value)
print(styled_df)
```

Q7 Write a python code to combine Multiple Excel Worksheets into (5)

a. Single Pandas Dataframe

b. Separate Pandas Dataframe

```
import pandas as pd

# a. combine multiple Excel worksheets into a single Pandas
# Dataframe.
def combine_worksheets(file_path, sheet_names):
    dfs=[]
    for sheet_name in sheet_names:
        df = pd.read_excel(file_path, sheet_name=sheet_name)
        dfs.append(df)
    combined_df = pd.concat(dfs)
    return combined_df
```

```
file_path = 'data.xlsx'
sheet_names = ['Sheet1', 'Sheet2', 'Sheet3']
combined_df = combine_worksheets(file_path, sheet_names)
print(combined_df)
```

#b. Separate Excel worksheets into separate Pandas DataFrames

```
def separate_worksheets(file_path):
    dfs = pd.read_excel(filepath, sheet_name=None)
    return dfs.
```

```
file_path = 'data.xlsx'
separated_dfs = separate_worksheets(file_path)
for sheet_name, df in separated_dfs.items():
    print(f"Dataframe from '{sheet_name}' :")
    print(df)
    print()
```

★

Q7 Write a python code to:

a. Read a colored image directly from a public domain & render it on a grey scale while displaying in Jupyter notebook & also show its resized version.

b. Write a python to read xml files.

Ans

a)

```
Import urllib.request
Import numpy as np
Import cv2
from matplotlib Import pyplot as plt
```

```
# a. Read a colored Image directly from a public domain,
# renders it in grayscale
Image_url = "https://example.com/Image.jpg"
req = urllib.request.urlopen(Image_url)
```

```
arr = np.asarray(bytearray(req.read()), dtype = np.uint8)
image = cv2.imdecode(arr, -1)

# Convert to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Display the grayscale image
plt.subplot(1, 2, 1)
plt.imshow(grayimage, cmap = 'gray')
plt.title('Grayscale Image')

# Resize the image
resized_image = cv2.resize(image, (400,400))

# Display the resized image
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
plt.title('Resized Image')
# Show the plots
plt.show()
```

b) Import xml.etree.ElementTree as ET

```
# Parse the XML file
tree = ET.parse('data.xml')
root = tree.getroot()

# Access the elements in the XML file
for element in root.iter():
    print(element.tag, element.text)
```