UNIT-3

1Q) How do you convert series to Data Frame? (4MARKS)

A: To convert a Series to a DataFrame in Python, you can use the pd.DataFrame() constructor provided by the pandas library. Here's a simple example:

```
import pandas as pd

# Create a Series

my_series = pd.Series([1, 2, 3, 4, 5])

# Convert the Series to a DataFrame

my_dataframe = pd.DataFrame(my_series, columns=['Column_Name'])

# Display the DataFrame

print(my_dataframe)
```

OUTPUT:

```
   Column Name
0            1
1            2
2            3
3            4
4            5
```

In this example, the pd.Series() function is used to create a pandas Series, and then pd.DataFrame() is used to convert the Series into a DataFrame. The columns parameter is used to specify the name of the column in the resulting DataFrame.

If your Series already has a name, you can use that as the column name in the DataFrame. For example:

```
import pandas as pd

# Create a named Series

my_series = pd.Series([1, 2, 3, 4, 5], name='My_Column')

# Convert the named Series to a DataFrame

my_dataframe = pd.DataFrame(my_series)

# Display the DataFrame

print(my_dataframe)
```

OUTPUT:

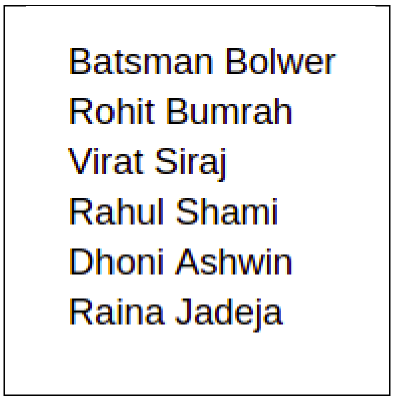
```
   My Column
0          1
1          2
2          3
3          4
4          5
```

In this case, the name of the Series is automatically used as the column name in the DataFrame. Adjust the column name as per your requirements.

2Q) How to read Text files with Pandas? (4MARKS)

A: In Python, the Pandas module allows us to load DataFrames from external files and work on them. The dataset can be in different types of files.

Text File Used:



```
Batsman Bolwer
Rohit Bumrah
Virat Siraj
Rahul Shami
Dhoni Ashwin
Raina Jadeja
```

Read Text Files with Pandas

Below are the methods by which we can read text files with Pandas:

Using read_csv()

Using read_table()

Using read_fwf()

Read Text Files with Pandas Using read_csv()

We will read the text file with pandas using the read_csv() function. Along with the text file, we also pass separator as a single space (‘ ’) for the space character because, for text files, the space character will separate each field. There are three parameters we can pass to the read_csv() function.

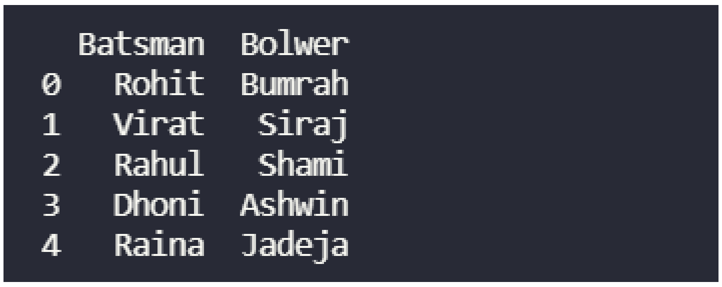
Syntax:

Syntax: data=pandas.read_csv(‘filename.txt’, sep=’ ‘, header=None, names=[“Column1”, “Column2”])

EXAMPLE1:

```
# Read Text Files with Pandas using read_csv()

# importing pandas

import pandas as pd

# read text file into pandas DataFrame

df = pd.read_csv("gfg.txt", sep=" ")

# display DataFrame

print(df)
```

OUTPUT:



```
  Batsman  Bolwer
0   Rohit  Bumrah
1   Virat   Siraj
2   Rahul   Shami
3   Dhoni  Ashwin
4   Raina  Jadeja
```

3Q) How are iloc and loc different?   (8MARKS)

Ans: loc () and iloc () are one of those methods. These are used in slicing data from the Pandas DataFrame. They help in the convenient selection of data from the DataFrame in Python. They are used in filtering the data according to some conditions.
To see and compare the difference between these two, we will create a sample Dataframe that we will use in the whole paragraph. The working of both of these methods is explained in the sample dataset of cars.

EXAMPLE:

```
# importing the module
import pandas as pd
# creating a sample dataframe
data = pd.DataFrame({'Brand': ['Maruti', 'Hyundai', 'Tata',
                               'Mahindra', 'Maruti', 'Hyundai',
                               'Renault', 'Tata', 'Maruti'],
                     'Year': [2012, 2014, 2011, 2015, 2012,
                              2016, 2014, 2018, 2019],
                     'Kms Driven': [50000, 30000, 60000,
                                    25000, 10000, 46000,
                                    31000, 15000, 2000],
                     'City': ['Gurgaon', 'Delhi', 'Mumbai',
                              'Delhi', 'Mumbai', 'Delhi',
                              'Mumbai', 'Chennai', 'Ghaziabad'],
                     'Mileage': [28, 27, 25, 26, 28,
                                 29, 24, 21, 24]})

# displaying the DataFrame
display(data)
```

OUTPUT:

```
   Brand     Year  Kms Driven   City      Mileage
0  Maruti    2012   50000     Gurgaon     28
1  Hyundai   2014   30000     Delhi       27
2  Tata      2011   60000     Mumbai      25
3  Mahindra  2015   25000     Delhi       26
4  Maruti    2012   10000     Mumbai      28
5  Hyundai   2016   46000     Delhi       29
6  Renault   2014   31000     Mumbai      24
7  Tata      2018   15000     Chennai     21
8  Maruti    2019   12000     Ghaziabad   24
```

<u>Python loc() function</u>:

The loc() function is label based data selecting method which means that we have to pass the name of the row or column which we want to select. This method includes the last element of the range passed in it, unlike iloc(). loc() can accept the boolean data unlike iloc(). Many operations can be performed using the loc() method like..

Example1: Selecting Data According to Some Conditions.

In this example, the code uses the loc function to select and display rows from the DataFrame where the brand is 'Maruti' and the mileage is greater than 25, showing relevant information about Maruti cars with high mileage.

```
# selecting cars with brand 'Maruti' and Mileage > 25

display(data.loc[(data.Brand == 'Maruti') & (data.Mileage > 25)])
```

OUTPUT:

```
   Brand   Year    Kms Driven   City       Mileage
0  Maruti 2012     50000        Gurgaon    28
4  Maruti 2012     10000        Mumbai     28
```

Python iloc() function:

The iloc() function is an indexed-based selecting method which means that we have to pass an integer index in the method to select a specific row/column. This method does not include the last element of the range passed in it unlike loc(). iloc() does not accept the boolean data unlike loc(). Operations performed using iloc() are:

Example 1: Selecting Rows Using Integer Indices

In this example, the code employs the iloc function to extract and display specific rows with indices 0, 2, 4, and 7 from the DataFrame, showcasing information about selected cars in the dataset.

CODE:

```
# selecting 0th, 2nd, 4th, and 7th index rows

display(data.iloc[[0, 2, 4, 7]])
```

OUTPUT:

```
   Brand   Year   Kms Driven    City     Mileage
0  Maruti 2012     50000      Gurgaon     28
2  Tata     2011     60000      Mumbai     25
4  Maruti 2012     10000     Mumbai     28
7  Tata     2018     15000      Chennai     21
```

4Q) How to set index to a Pandas DataFrame?   (8MARKS)

ANS: In pandas, you can set an index for a DataFrame using the set index() method. The index is a way to uniquely identify each row in the DataFrame. Here's a basic example:

```
import pandas as pd
# Create a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
      'Age': [25, 30, 35],
      'City': ['New York', 'San Francisco', 'Los Angeles']}

df = pd.DataFrame(data)

# Display the original DataFrame
print("Original DataFrame:")
print(df)
# Set the 'Name' column as the index
df.set_index('Name', inplace=True)
# Display the DataFrame with the new index
print("\nDataFrame with 'Name' as the index:")
print(df)
OUTPUT:
```

```
Original DataFrame:
      Name  Age           City
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   35    Los Angeles

DataFrame with 'Name' as the index:
         Age           City
Name
Alice     25       New York
Bob       30  San Francisco
Charlie   35    Los Angeles
```

In this example, the set_index() method is used to set the 'Name' column as the index. The inplace=True parameter modifies the DataFrame in place, and if you don't use inplace=True, a new DataFrame with the updated index will be returned.

If you want to set multiple columns as the index, you can pass a list of column names to set_index():

```
# Set multiple columns as the index
df.set_index(['Name', 'City'], inplace=True)
```

To reset the index and move the current index back to a regular column, you can use the reset_index() method:

```
# Reset the index
df.reset_index(inplace=True)
```

These methods provide flexibility in managing the DataFrame index based on your specific data analysis needs.

5Q) a) How to add a row to a Pandas Data Frame? (4MARKS)

b) How to add a column to a Pandas Data Frame? (4MARKS)

ANS**: a) How to add a row to a Pandas Data Frame:**

To add a row to a Pandas DataFrame, you can use the loc indexer. Here's an example:

```
import pandas as pd
# Create a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'San Francisco', 'Los Angeles']}
df = pd.DataFrame(data)
# Display the original DataFrame
print("Original DataFrame:")
print(df)
# Add a new row to the DataFrame
new_row = {'Name': 'David', 'Age': 28, 'City': 'Chicago'}
df.loc[len(df)] = new_row
# Display the DataFrame with the new row
print("\nDataFrame with the new row:")
print(df)
```

In this example, a new row with data for 'David' is added to the DataFrame using the loc indexer. The len(df) is used to find the next available index for the new row.

b) How to add a column to a Pandas Data Frame:

To add a column to a Pandas DataFrame, you can directly assign values to a new column or use the insert method. Here are both methods:

```
# Add a new column directly

df['Salary'] = [50000, 60000, 70000, 55000]

# Display the DataFrame with the new column

print("\nDataFrame with the new column added directly:")

print(df)

# Alternatively, use the insert method to specify the column position

df.insert(2, 'Department', ['HR', 'IT', 'Marketing', 'Finance'])

# Display the DataFrame with the new column inserted

print("\nDataFrame with the new column inserted:")

print(df)
```

In the first method, a new column 'Salary' is added directly by assigning a list of values to it. In the second method, the insert method is used to specify the column position (index 2) and provide the column name ('Department').

6Q) Explain about web scraping with an example? (8MARKS)

Ans: Web scraping is an automatic method to obtain large amounts of data from websites. Most of this data is unstructured data in an HTML format which is then converted into structured data in a spreadsheet or a database so that it can be used in various applications. There are many different ways to perform web scraping to obtain data from websites. These include using online services, particular API's or even creating your code for web scraping from scratch. Many large websites, like Google, Twitter, Facebook, StackOverflow, etc. have API's that allow you to access their data in a structured format. This is the best option, but there are other sites that don't allow users to access large amounts of data in a structured form or they are simply not that technologically advanced. In that situation, it's best to use Web Scraping to scrape the website for data.

Web scraping requires two parts, namely the crawler and the scraper. The crawler is an artificial intelligence algorithm that browses the web to search for the particular data required by following the links across the internet. The scraper, on the other hand, is a specific tool created to extract data from the website. The design of the scraper can vary greatly according to the complexity and scope of the project so that it can quickly and accurately extract the data.

```
# Install BeautifulSoup and requests if you haven't already

# You can install them using: pip install beautifulsoup4 requests

import requests
from bs4 import BeautifulSoup

# Specify the URL of the website you want to scrape
url = 'https://example.com'

# Send a GET request to the URL
response = requests.get(url)
# Check if the request was successful (status code 200)
if response.status_code == 200:
  # Parse the HTML content of the page
  soup = BeautifulSoup(response.text, 'html.parser')

  # Extract information from the page
  # In this example, let's extract all the links (anchor tags) on the page
  links = soup.find_all('a')

  # Print the extracted links
  for link in links:
    print(link.get('href'))
else:
  print('Failed to retrieve the web page. Status code:', response.status_code)
```

OUTPUT:

```
https://www.iana.org/domains/example
```

7Q) Write about Interacting with API's with an example. (8MARKS)

ANS: Interacting with APIs (Application Programming Interfaces) is a fundamental aspect of modern web development, enabling seamless communication and data exchange between different software systems. APIs allow developers to access specific functionalities or retrieve data from a remote server in a standardized way. Here's an explanation of how to interact with APIs, along with a simple example using Python and the requests library.

Understanding APIs:

APIs define a set of rules and protocols that allow one piece of software to interact with another. They can be categorized into different types, such as RESTful APIs, which use HTTP requests to perform operations like retrieving or sending data.

Interacting with APIs EXAMPLE:

```
import requests

# API key for OpenWeatherMap (replace with your own key)
api_key = 'your_api_key'
city = 'New York'

# API endpoint URL
api_url = f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}'

# Send a GET request to the API
response = requests.get(api_url)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Parse the JSON content of the response
    data = response.json()

    # Display the retrieved weather information
    print(f"Weather in {city}: {data['weather'][0]['description']}")
    print(f"Temperature: {data['main']['temp']} K")
else:
    print(f'Failed to fetch data. Status code: {response.status_code}')
```

OUTPUT:

```
Failed to fetch data. Status code: 401
```

8Q) Write about function arguments used for reading csv/excel files. (8MARKS)

ANS: When working with CSV (Comma-Separated Values) or Excel files in programming, various libraries and functions provide different ways to read and handle these files. Function arguments play a crucial role in customizing the behavior of these functions to meet specific requirements. Let's explore common function arguments used for reading CSV and Excel files in popular programming languages such as Python and R.

Python (using pandas library):
1. filepath_or_buffer:
Description: Specifies the path or URL of the CSV or Excel file.
Example: "data.csv"
2. sep:
Description: Defines the delimiter used in the CSV file. Default is a comma (,).
Example: sep=','
3**. header:**
Description: Specifies the row number to use as the column names. Default is 0 (first row).
Example: header=0
4. index_col:
Description: Specifies which column to use as the row labels. Default is None.
Example: index_col='ID'

5**. usecols:**
Description: Defines which columns to read from the file. Accepts column indices or names.
Example: usecols=['Name', 'Age']

6. **dtype:**
Description: Specifies the data type for columns. Helpful for optimizing memory usage.
Example: dtype={'ID': int, 'Score': float}

7. parse_dates:
Description: Parses specified columns as dates. Useful for datetime manipulation.
Example: parse_dates=['Date']

8. na_values:
Description: Specifies additional values to recognize as NaN (Not a Number).
Example: na_values=['NA', 'missing']

R (using readr and readxl libraries):

1. file:
Description: Specifies the path or URL of the CSV or Excel file.
Example: file = "data.csv"

2**. delim**:
Description: Defines the delimiter used in the CSV file. Default is a comma (,).
Example: delim = ','

3. col_names:
Description: Specifies whether to read the first row as column names. Default is TRUE.
Example: col_names = TRUE

4. range:
Description: Defines the range of cells to read in an Excel file.
Example: range = "A1:C10"

5. col_types:
Description: Specifies the data types for columns. Accepts a string of type characters.
Example: col_types = "cidi"

6**. na:**
Description: Specifies additional values to recognize as missing values.
Example: na = c("NA", "missing")

7. sheet:
Description: Specifies the sheet name or index when reading from an Excel file.
Example: sheet = "Sheet1"

8. skip:
Description: Specifies the number of lines to skip at the beginning of the file.
Example: skip = 2

These function arguments provide a flexible and powerful way to tailor the file reading process according to the specific needs of data analysis or manipulation tasks. Understanding and utilizing these arguments can enhance efficiency and ensure accurate interpretation of CSV and Excel files in various programming contexts.

9Q) How do you split a Data Frame according to a Boolean criterion? (4MARKS)

ANS:

To split a DataFrame based on a Boolean criterion in Python using the pandas library, you can use boolean indexing. Here's a step-by-step guide:

Let's assume you have a DataFrame named df and a Boolean Series named criterion. You want to split the DataFrame into two based on the True and False values in the criterion Series.

```
import pandas as pd
# Create a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Emma'],
        'Age': [25, 30, 22, 35, 28],
        'IsStudent': [False, True, False, False, True]}
df = pd.DataFrame(data)

# Create a Boolean criterion
criterion = df['IsStudent']

# Split the DataFrame based on the Boolean criterion
df_true = df[criterion]
df_false = df[~criterion]

# Display the results
print("DataFrame where 'IsStudent' is True:")
print(df_true)

print("\nDataFrame where 'IsStudent' is False:")
print(df_false)
```

OUTPUT:

```
DataFrame where 'IsStudent' is True:
   Name   Age  IsStudent
1  Bob    30     True
4 Emma    28     True

DataFrame where 'IsStudent' is False:
      Name  Age  IsStudent
0    Alice   25      False
2  Charlie   22      False
3    David  35      False
```

10Q) How can we convert NumPy array into a DataFrame? (4MARKS)

ANS: You can convert a NumPy array into a DataFrame in Python using the pandas library. Here's a simple example:

```
import numpy as np
import pandas as pd

# Create a NumPy array
numpy_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Convert the NumPy array to a DataFrame
df = pd.DataFrame(numpy_array)

# Display the DataFrame
print(df)
```

OUTPUT:

```
   0  1  2
0  1  2  3
1  4  5  6
2  7  8  9
```

In this example, pd.DataFrame(numpy_array) is used to create a DataFrame from the NumPy array. The resulting DataFrame will have default integer index and column labels.

If we want to specify custom index and column labels, we can do so as follows:

```
import numpy as np
import pandas as pd

# Create a NumPy array
numpy_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# Specify custom index and column labels
index_labels = ['row1', 'row2', 'row3']
column_labels = ['col1', 'col2', 'col3']
# Convert the NumPy array to a DataFrame with custom labels
df = pd.DataFrame(numpy_array, index=index_labels, columns=column_labels)
# Display the DataFrame
print(df)
```

OUTPUT:

Here, index is used to specify custom row labels, and columns is used to specify custom column labels.

```
      col1 col2 col3
row1   1    2    3
row2   4    5    6
row3   7    8    9
```