UNIT 1: PYTHON PROGRAMMING-II

Title: Python Programming-II	**Approach**: Hands on, Team Discussion, Web search, Case studies

Summary:
This chapter provides a comprehensive review of fundamental python programming and techniques. Students will gain hands-on experience with essential Python libraries, preparing them for more advanced data analysis and machine learning tasks which can be incorporated in their capstone project.

Learning Objectives:

1. Review the basics of the NumPy and Pandas library, including arrays, and essential functions.
2. Efficiently import and export data between CSV files and Pandas Data Frames.
3. Implement Linear Regression algorithm, including data preparation, and model training.

Key Concepts:

1. Recap of NumPy library
2. Recap of Pandas library
3. Importing and Exporting Data between CSV Files and Data Frames
4. Handling missing values
5. Linear Regression algorithm

Learning Outcomes:

Students will be able to:

1. Apply the fundamental concepts of the NumPy and Pandas libraries to perform data manipulation and analysis tasks.
2. Import and export data between CSV files and Pandas Data Frames, ensuring data integrity and consistency.

Prerequisites: Foundational understanding of Python from class XI and familiarity with the basic programming.

1.1. Python Libraries

Python libraries are collections of pre-written codes that we can use to perform common tasks, making our programming life easier. They are like toolkits that provide functions and methods to help us avoid writing code from scratch.

In the realm of data science and analytics, two powerful libraries stand out for their efficiency and versatility: NumPy and Pandas. These libraries form the backbone of data manipulation and analysis in Python, enabling users to handle large datasets with ease and precision.

Let's recap a few of these libraries (covered in Class XI) that are incredibly valuable in the realm of Artificial Intelligence, Data science and analytics.

1.1. 1 NumPy Library

NumPy, short for Numerical Python is a powerful library in Python used for numerical computing. It is a general-purpose array-processing package.

In NumPy, the number of dimensions of the array is called the rank of the array

```
# Creating a rank 1 Array
import numpy as np
arr = np.array([1, 2, 3])
print("Array with Rank 1: \n",arr)
```

```
Array with Rank 1:
 [1 2 3]
```

```
# Creating a rank 2 Array
import numpy as np
arr = np.array([[1, 2, 3],[4,5,6]])
print("Array with Rank 2: \n",arr)
```

```
Array with Rank 2:
 [[1 2 3]
 [4 5 6]]
```

```
# Creating an array from tuple
arr = np.array((1, 3, 2))
print("\nArray created using tuple:\n", arr)
```

```

Array created using tuple:
 [1 3 2]
```

1.1.2 Pandas Library

Where and why do we use the Pandas library in Artificial Intelligence?
Suppose we have a dataset containing information about various marketing campaigns conducted by the company, such as campaign type, budget, duration, reach, engagement metrics, and sales performance. **Pandas** is used to load the dataset, display summary statistics, and perform group-wise analysis to understand the performance of different marketing campaigns. We can visualize the sales performance and average engagement metrics for each campaign type using **Matplotlib**, a popular plotting library in Python. **Pandas** provides powerful data manipulation and aggregation functionalities, making it easy for us to perform complex analyses and generate insightful visualizations. This capability is invaluable in AI and data-driven decision-making processes, allowing businesses to gain actionable insights from their data.

1.1.2.1 Pandas Data Structures

Pandas generally provides two data structures for manipulating data, They are:

- Series
- Data Frame

i) Creation of a Series from Scalar Values- A Series can be created using scalar values as shown below:

```
import pandas as pd #import Pandas with alias pd
series1 = pd.Series([10,20,30]) #create a Series
print(series1) #Display the series
```

```
0    10
1    20
2    30
dtype: int64
```

ii) Creation of a DataFrame from NumPy arrays

array1=np.array([90,100,110,120])

array2=np.array([50,60,70])

array3=np.array([10,20,30,40])

marksDF = pd.DataFrame([array1, array2, array3], columns=['A', 'B', 'C', 'D'])

print(marksDF)

	A	B	C	D
0	90	100	110	120.0
1	50	60	70	NaN
2	10	20	30	40.0

iii) Creation of a DataFrame from dictionary of array/lists:

```
import pandas as pd
# intialise data of lists.
data = {'Name':['Varun', 'Ganesh', 'Joseph', 'Abdul','Reena'],
      'Age':[37,30,38, 39,40]}
# Create DataFrame
df = pd.DataFrame(data)
# Print the output.
print(df)
```

	Name	Age
0	Varun	37
1	Ganesh	30
2	Joseph	38
3	Abdul	39
4	Reena	40

The dictionary keys become column labels by default in a DataFrame, and the lists become the columns of data.

iv) Creation of DataFrame from List of Dictionaries

```
# Create list of dictionaries
listDict = [{'a':10, 'b':20}, {'a':5,'b':10,'c':20}]
a= pd.DataFrame(listDict)
print(a)
```

```
     a   b     c
0   10  20   NaN
1    5  10  20.0
```

There will be as many rows as the number of dictionaries present in the list.

1.1.2.2 Dealing with Rows and Columns

i) Adding a New Column to a DataFrame:

```
ResultSheet={'Rajat': pd.Series([90, 91, 97],index=['Maths','Science','Hindi']),
             'Amrita': pd.Series([92, 81, 96],index=['Maths','Science','Hindi']),
             'Meenakshi': pd.Series([89, 91, 88],index=['Maths','Science','Hindi']),
             'Rose': pd.Series([81, 71, 67],index=['Maths','Science','Hindi']),
             'Karthika': pd.Series([94, 95, 99],index=['Maths','Science','Hindi'])}
Result = pd.DataFrame(ResultSheet)
print(Result)
```

```
         Rajat  Amrita  Meenakshi  Rose  Karthika
Maths       90      92         89    81        94
Science     91      81         91    71        95
Hindi       97      96         88    67        99
```

To add a new column for another student ‘Fathima’, we can write the following statement:

```
Result['Fathima']=[89,78,76]
print(Result)
```

```
         Rajat  Amrita  Meenakshi  Rose  Karthika  Fathima
Maths       90      92         89    81        94       89
Science     91      81         91    71        95       78
Hindi       97      96         88    67        99       76
```

ii) Adding a New Row to a DataFrame:

```
Result.loc['English'] = [90, 92, 89, 80, 90, 88]
print(Result)
```

```
         Rajat  Amrita  Meenakshi  Rose  Karthika  Fathima
Maths       90      92         89    81        94       89
Science     91      81         91    71        95       78
Hindi       97      96         88    67        99       76
English     90      92         89    80        90       88
```

DataFRame.loc[] method can also be used to change the data values of a row to a particular value. For example, to change the marks of science.

```
Result.loc['Science'] = [92, 84, 90, 72, 96, 88]
print(Result)
```

	Rajat	Amrita	Meenakshi	Rose	Karthika	Fathima
Maths	90	92	89	81	94	89
Science	92	84	90	72	96	88
Hindi	97	96	88	67	99	76
English	90	92	89	80	90	88

1.1.2.3 Deleting Rows or Columns from a DataFrame

To delete a row, the parameter axis is assigned the value 0 and for deleting a column, the parameter axis is assigned the value 1.

```
Result = Result.drop('Hindi', axis=0) #delete the row "Hindi"
print(Result)
```

	Rajat	Amrita	Meenakshi	Rose	Karthika	Fathima
Maths	90	92	89	81	94	89
Science	92	84	90	72	96	88
English	90	92	89	80	90	88

```
#delete multiple columns
Result = Result.drop(['Rajat','Meenakshi','Karthika'], axis=1)
print(Result)
```

	Amrita	Rose	Fathima
Maths	92	81	89
Science	84	72	88
English	92	80	88

1.1.2.4 Attributes of DataFrames

We are going to use following data as example to understand the attributes of a DataFrame.

```
import pandas as pd
# creating a 2D dictionary
dict = {"Student": pd.Series(["Arnav","Neha","Priya","Rahul"],
                    index=["Data 1","Data 2","Data 3","Data 4"]),
        "Marks": pd.Series([85, 92, 78, 83],
                    index=["Data 1","Data 2","Data 3","Data 4"]),
        "Sports": pd.Series(["Cricket","Volleyball","Hockey","Badminton"],
                    index=["Data 1","Data 2","Data 3","Data 4"])}
# creating a DataFrame
df = pd.DataFrame(dict)
# printing this DataFrame on the output screen
print(df)
```

```
        Student  Marks      Sports
Data 1   Arnav     85     Cricket
Data 2    Neha     92  Volleyball
Data 3   Priya     78      Hockey
Data 4   Rahul     83   Badminton
```

i) DataFrame.index

```
>>>df.index
Index(['Data 1', 'Data 2', 'Data 3', 'Data 4'], dtype='object')
```

ii) DataFrame.columns

```
>>>df.columns
Index(['Student', 'Marks', 'Sports'], dtype='object')
```

iii) DataFrame.shape

```
>>>df.shape
(4,3)
```

iv) DataFrame.head(n)

```
>>>df.head(2)
```

	Student	Marks	Sports
Data 1	Arnav	85	Cricket
Data 2	Neha	92	Volleyball

v) DataFrame.tail(n)

```
>>>df.tail(2)
```

	Student	Marks	Sports
Data 3	Priya	78	Hockey
Data 4	Rahul	83	Badminton

1.2. Import and Export Data between CSV Files and DataFrames

CSV files, which stand for Comma-Separated Values, are simple text files used to store tabular data. Each line in a CSV file represents a row in the table, and each value in the row is separated by a comma. This format is widely used because it is easy to read and write, both for humans and computers.

In Python, CSV files are incredibly important for data analysis and manipulation. We often use the Pandas library to load, manipulate, and analyze data stored in CSV files. Pandas provides powerful tools to read CSV files into Data Frames, which are data structures that allow us to perform complex operations on the data with ease. This makes CSV files a go-to format for data scientists and analysts working with Python.

1.2.1 Importing a CSV file to a DataFrame

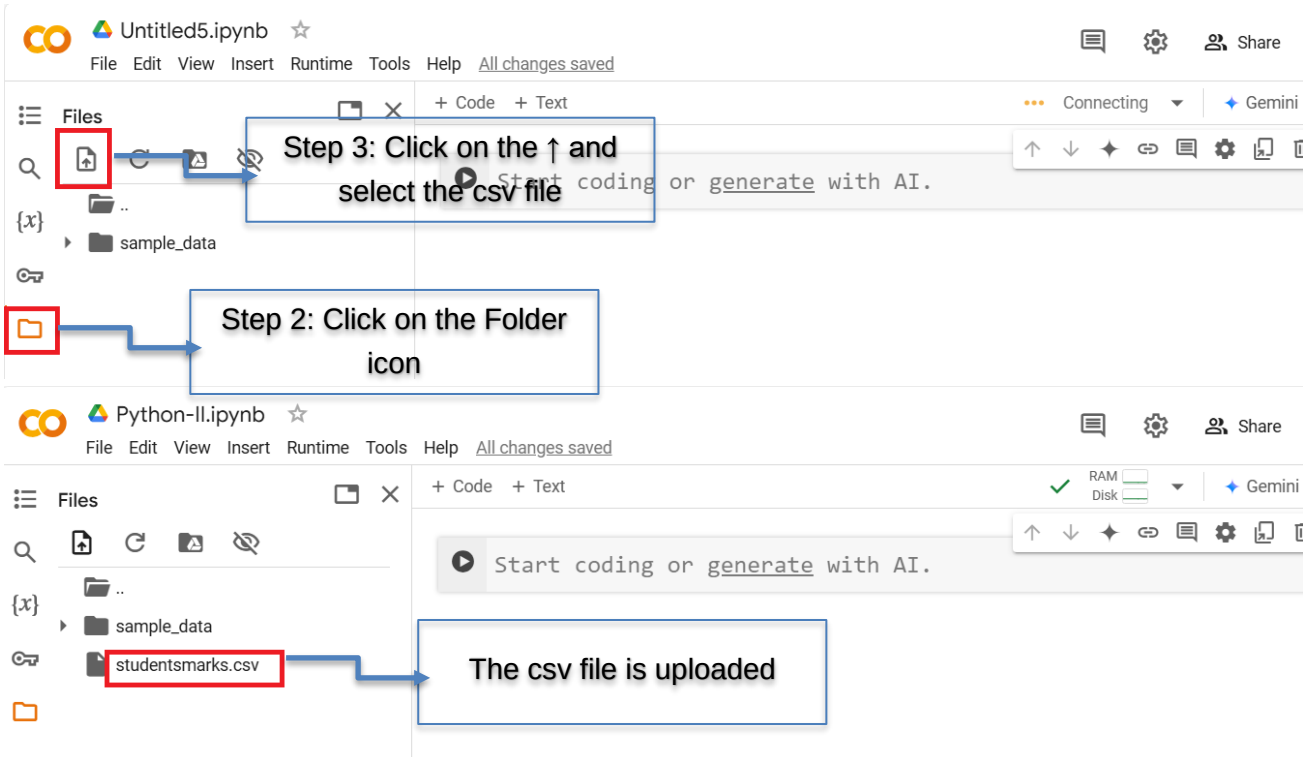
Using the read_csv() function, we can import tabular data from CSV files into pandas DataFrame by specifying a parameter value for the file name **(e.g. pd.read_csv("filename.csv")).**

Let us create a DataFrame from the "studentmarks.csv" file.

Follow the following steps to upload the csv file in the google colab file

Step1: Open the Google colab from the following link https://colab.research.google.com/

And create a new file from the File menu



Once the csv file is uploaded then we can execute the following code to convert the csv to DataFrame.

```
import pandas as pd
df=pd.read_csv("studentsmarks.csv")
print(df)
```

	Roll No	Name	AI Marks	Maths Marks
0	1	Akshita	89	91
1	2	Apoorva	91	87
2	3	Bhavik	88	76
3	4	Deepti	78	71
4	5	Farhan	84	84

On Python IDE we can directly give the complete path name with the csv file in parenthesis.

```
import pandas as pd
import io
df = pd.read_csv('C:/PANDAS/studentsmarks.csv',sep =",", header=0)
print(df)
```

```
   Roll No     Name  AI Marks  Maths Marks
0        1  Akshita        89           91
1        2  Apoorva        91           87
2        3   Bhavik        88           76
3        4   Deepti        78           71
4        5   Farhan        84           84
```

1.2.2 Exporting a DataFrame to a CSV file

We can use the to_csv() function to save a DataFrame to a text or csv file. For example, to save the DataFrame df created in above coding

```
print(df)
```

```
   Roll No     Name  AI Marks  Maths Marks
0        1  Akshita        89           91
1        2  Apoorva        91           87
2        3   Bhavik        88           76
3        4   Deepti        78           71
4        5   Farhan        84           84
```

Program to export this data-

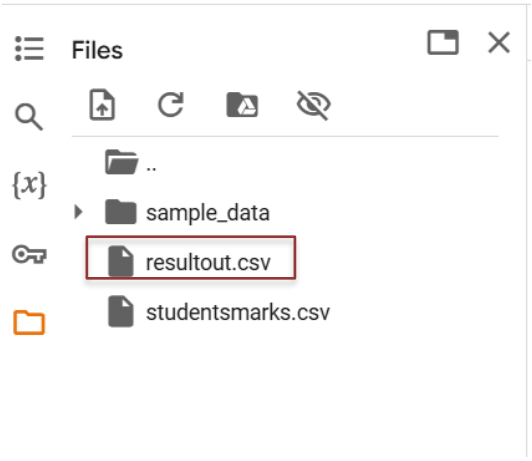
```
df.to_csv(path_or_buf='C:/PANDAS/resultout.csv', sep=',')
```

This creates a file by the name resultout.csv in the hard disk. When we open this file in any text editor or a spreadsheet, we will find the above data along with the row labels and the column headers, separated by comma.

On Google Colab we can write the following code

df.to_csv("resultout.csv",index=False)

The resultout.csv is created



1.3. Handling Missing Values

The two most common strategies for handling missing values explained in this section are:

i) Drop the row having missing values **OR**

ii) Estimate the missing value

Checking Missing Values

Pandas provide a function isnull() to check whether any value is missing or not in the DataFrame. This function checks all attributes and returns True in case that attribute has missing values, otherwise returns False

Drop Missing Values

Dropping will remove the entire row (object) having the missing value(s). This strategy reduces the size of the dataset used in data analysis, hence should be used in case of missing values on few objects.

The dropna() function can be used to drop an entire row from the DataFrame

Estimate the missing value

Missing values can be filled by using estimations or approximations e.g a value just before (or after) the missing value, average/minimum/maximum of the values of that attribute, etc. In some cases, missing values are replaced by zeros (or ones).

The fillna(num) function can be used to replace missing value(s) by the value specified in num.

For example, fillna(0) replaces missing value by 0. Similarly fillna(1) replaces missing value by 1.

1.4. CASE STUDY

Let's examine a scenario where we have the marks of certain students, but some data is missing in the columns due to specific circumstances. For example, Meera and Suhana couldn't attend the Science and Hindi respectively exam due to fever, Joseph participated in a national-level science exhibition on the day of the AI exam.

Let us feed the data in Python

```
ResultSheet={'Maths': pd.Series([90,91,97,89,65,93],
              index=['Heena','Shefali','Meera','Joseph','Suhana','Bismeet']),
              'Science':pd.Series([92,81,np.NaN,87,50,88],
              index=['Heena','Shefali','Meera','Joseph','Suhana','Bismeet']),
              'English': pd.Series([89, 91, 88,78,77,82],
              index=['Heena','Shefali','Meera','Joseph','Suhana','Bismeet']),
              'Hindi': pd.Series([81, 71, 67,82,np.NaN,89],
              index=['Heena','Shefali','Meera','Joseph','Suhana','Bismeet']),
              'AI': pd.Series([94, 95, 99,np.NaN,96,99],
              index=['Heena','Shefali','Meera','Joseph','Suhana','Bismeet'])}
marks = pd.DataFrame(ResultSheet)
print(marks)
```

	Maths	Science	English	Hindi	AI
Heena	90	92.0	89	81.0	94.0
Shefali	91	81.0	91	71.0	95.0
Meera	97	NaN	88	67.0	99.0
Joseph	89	87.0	78	82.0	NaN
Suhana	65	50.0	77	NaN	96.0
Bismeet	93	88.0	82	89.0	99.0

#check for missing values

```
>>>print(marks.isnull())
```

	Maths	Science	English	Hindi	AI
Heena	False	False	False	False	False
Shefali	False	False	False	False	False
Meera	False	True	False	False	False
Joseph	False	False	False	False	True
Suhana	False	False	False	True	False
Bismeet	False	False	False	False	False

We can see there are three “True” values. So three pieces of data are missing.

```
>>>print(marks['Science'].isnull().any())
True
print(marks['Maths'].isnull().any())
False
#To find the total number of NaN in the whole dataset
>>>marks.isnull().sum().sum()
3
```

#apply dropna() for the above case

```
drop=marks.dropna()
print(drop)
```

	Maths	Science	English	Hindi	AI
Heena	90	92.0	89	81.0	94.0
Shefali	91	81.0	91	71.0	95.0
Bismeet	93	88.0	82	89.0	99.0

#Estimate the missing value

```
FillZero = marks.fillna(0)
print(FillZero)
```

	Maths	Science	English	Hindi	AI
Heena	90	92.0	89	81.0	94.0
Shefali	91	81.0	91	71.0	95.0
Meera	97	0.0	88	67.0	99.0
Joseph	89	87.0	78	82.0	0.0
Suhana	65	50.0	77	0.0	96.0
Bismeet	93	88.0	82	89.0	99.0

1.5. PRACTICAL ACTIVITY (**For Advanced Learners)

Activity: To implement Linear Regression algorithm

Dataset: https://drive.google.com/drive/folders/1tLbVXWkKzcp6O_-FAvn9usEWuoF3jTp3?usp=sharing

import pandas as pd
df=pd.read_csv('USA_Housing.csv')
df.head()

	Income	House Age	Number of Rooms	Number of Bedrooms	Area	Price	Address
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05	USNS Raymond\nFPO AE 09386

```
#To check the size of the dataframe created from the csv file
df.shape
```

```
(5000, 7)
```

```
#Generates descriptive statistics that summarize the central tendency, shape of the dataset's distribution,excluding NaN values
df.describe()
```

	Income	House Age	Number of Rooms	Number of Bedrooms	Area	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562390	5.322283	6.299250	3.140000	29403.928700	9.975771e+05
50%	68804.286405	5.970429	7.002902	4.050000	36199.406690	1.232669e+06
75%	75783.338665	6.650808	7.665871	4.490000	42861.290770	1.471210e+06
max	107701.748400	9.519088	10.759588	6.500000	69621.713380	2.469066e+06

Upon examining the count, it's evident that all columns contain 5000 values. This suggests that there are no missing values in any of the columns.

EXPLORATORY DATA ANALYSIS

```
#Deleting columns which do not have relevance to the ML model prediction
df.drop(['Address'],inplace=True,axis=1)
df
```

	Income	House Age	Number of Rooms	Number of Bedrooms	Area	Price
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05
...	...	...	...	...	...	...

```
#Separating x(independent variables) and y(dependent variable-price)
x=df.drop(['Price'],axis=1)
x
```

	Income	House Age	Number of Rooms	Number of Bedrooms	Area
0	79545.45857	5.682861	7.009188	4.09	23086.80050
1	79248.64245	6.002900	6.730821	3.09	40173.07217
2	61287.06718	5.865890	8.512727	5.13	36882.15940
3	63345.24005	7.188236	5.586729	3.26	34310.24283
4	59982.19723	5.040555	7.839388	4.23	26354.10947
...	...	...	...	...	...

```
y=df['Price']
y.head()
```

	Price
0	1.059034e+06
1	1.505891e+06
2	1.058988e+06
3	1.260617e+06
4	6.309435e+05

```
#Splitting the data- 80% will be taken for training the model, and 20% for testing the model.
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y, test_size=0.20)
```

```
x_train.shape
```

```
(4000, 5)
```

From the above output, we understand that 4000 rows (80% of 5000 rows will be used for training the model)

```
#randomly selected 4000 rows
x_train.head()
```

	Income	House Age	Number of Rooms	Number of Bedrooms	Area
4388	54516.63198	7.123411	4.321939	2.32	51298.95040
2472	59278.94589	5.945139	8.847869	3.16	12351.71960
3674	77449.31607	5.034661	6.760959	2.02	30054.78687
3447	73092.74131	5.615460	6.524657	2.21	43509.45840
833	63856.30850	7.456390	6.844399	3.41	31114.89740

```
y_train.head()
```

	Price
4388	9.444910e+05
2472	1.015011e+06
3674	1.161458e+06
3447	1.336172e+06
833	1.311903e+06

```
#remaining data will be used for testing
print(x_test.shape)
print(y_test.shape)
```

```
(1000, 5)
(1000,)
```

Applying the Linear Regression Algorithm

```
#Applying the algorithm to train the model using 80% of data
from sklearn.linear_model import LinearRegression
m=LinearRegression()
m.fit(x_train,y_train)
```

```
LinearRegression
LinearRegression()
```

```
#Predicting the values for test data (20% of data)
y_predict=m.predict(x_test)  #y_predict is a numpy array created here by predicting the prices for test data
y_predict[0:5]
```

```
array([1164598.07733652, 1055980.82787463, 1686739.50414926,
       1349476.932423  , 1074158.53645449])
```

```
y_test.head()
```

	Price
2189	1.241699e+06
1367	9.050452e+05
4764	1.742351e+06
4495	1.331897e+06
3845	1.054856e+06

```
#Comparing the actual and predicted y values of test data
df1=pd.DataFrame({"Actual": y_test,"Predicted":y_predict })
df1.head()
```

	Actual	Predicted
2189	1.241699e+06	1.164598e+06
1367	9.050452e+05	1.055981e+06
4764	1.742351e+06	1.686740e+06
4495	1.331897e+06	1.349477e+06
3845	1.054856e+06	1.074159e+06

We observe that there is a difference between the actual and predicted value.
Further, we need to calculate the error, evaluate the model and test the accuracy of the model. This will be covered in the next chapter.

EXERCISES

A. Objective type questions

1. Which of the following is a primary data structure in Pandas?
a) List
b) Tuple
c) Series
d) Matrix

2. What does the fillna(0) function do in Pandas?
a) Removes rows with missing values
b) Fills missing values with zeros
c) Estimates missing values based on averages
d) Converts all data to zero

3. In Linear Regression, which library is typically used for importing and managing data?
a) NumPy
b) Pandas
c) Matplotlib
d) Scikit-learn

4. What is the correct syntax to read a CSV file into a Pandas DataFrame?
a) pd.DataFrame("filename.csv")
b) pd.read_csv("filename.csv")
c) pandas.read_file("filename.csv")
d) pd.file_read("filename.csv")

5. What is the result of the df.shape function?
a) Data type of the DataFrame
b) Number of rows and columns in the DataFrame
c) Memory usage of the DataFrame
d) Column names of the DataFrame

6. Which function can be used to export a DataFrame to a CSV file?
a) export_csv()
b) to_file()
c) to_csv()
d) save_csv()

B. Short Answer Questions

1. What is a DataFrame in Pandas?
2. How do you create a Pandas Series from a dictionary?
3. Name two strategies to handle missing values in a DataFrame.
4. What does the head(n) function do in a DataFrame?
5. What is the role of NumPy in Python programming?
6. Explain the use of the isnull() function in Pandas.

C. Long Answer Questions

1. Describe the steps to import and export data using Pandas.
2. Explain the concept of handling missing values in a DataFrame with examples.
3. What is Linear Regression, and how is it implemented in Python?
4. Compare NumPy arrays and Pandas DataFrames.
5. How can we add new rows and columns to an existing DataFrame? Explain with code examples.
6. What are the attributes of a DataFrame? Provide examples.

D. Case study

1. A dataset of student marks contains missing values for some subjects. Write Python code to handle these missing values by replacing them with the mean of the respective columns.

2. Write Python code to load the file into a Pandas DataFrame, calculate the total sales for each product, and save the results into a new CSV file. Click in the link below to access sales.csv dataset.
   https://drive.google.com/drive/folders/1tLbVXWkKzcp6O_-FAvn9usEWuoF3jTp3?usp=sharing

3. In a marketing dataset, analyze the performance of campaigns using Pandas. Describe steps to group data by campaign type and calculate average sales and engagement metrics.

4. A company has collected data on employee performance. Some values are missing, and certain columns are irrelevant. Explain how to clean and preprocess this data for analysis using Pandas.