Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

Experiment No.1

Aim: To understand the process of data preparation using NumPy and Pandas

CO Mapped:
CO1: To apply the process of data preparation for the given dataset to solve real-world problems

Prerequisites: Python3, basic syntax of NumPy and Pandas

Theory:

Data preparation is the process of preparing raw data so that it is suitable for further processing and analysis. Key steps include collecting, cleaning, and labelling raw data into a form suitable for machine learning (ML) algorithms and then exploring and visualizing the data.

Derive an index field and add it to the data set

Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric python packages. Pandas is one of those packages and makes importing and analysing data much easier.

Pandas set_index() is a method to set a List, Series or Data frame as index of a Data Frame. Index column can be set while making a data frame too. But sometimes a data frame is made out of two or more data frames and hence later index can be changed using this method. Syntax:

DataFrame.set_index(keys, drop=True, append=False, inplace=False, verify_integrity=False)

Parameters:

keys: Column name or list of column name.
drop: Boolean value which drops the column used for index if True.
append: Appends the column to existing index column if True.
inplace: Makes the changes in the dataframe if True.
verify_integrity: Checks the new index column for duplicates if True.

Code #1: Changing Index column
In this example, First Name column has been made the index column of Data Frame.

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv")

# setting first name as index column
data.set_index("First Name", inplace = True)

# display
data.head()
```

Output:
As shown in the output images, earlier the index column was a series of number but later it has been replaced with First name.
Before operation –

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

After operation –

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

Code #2: Multiple index Column
In this example, two columns will be made as index column. Drop parameter is used to Drop the column and append parameter is used to append passed columns to the already existing index column.

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv")

# setting first name as index column
data.set_index(["First Name", "Gender"], inplace = True,
                        append = True, drop = False)

# display
data.head()
```

Output:
As shown in the output Image, the data is having 3 index columns.

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

	First Name	Gender	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

Code #3: Setting a single Float column as Index in Pandas DataFrame

```
# importing pandas library
import pandas as pd

# creating and initializing a nested list
students = [['jack', 34, 'Sydeny', 'Australia',85.96],
           ['Riti', 30, 'Delhi', 'India',95.20],
           ['Vansh', 31, 'Delhi', 'India',85.25],
           ['Nanyu', 32, 'Tokyo', 'Japan',74.21],
           ['Maychan', 16, 'New York', 'US',99.63],
           ['Mike', 17, 'las vegas', 'US',47.28]]

# Create a DataFrame object
df = pd.DataFrame(students,
                  columns=['Name', 'Age', 'City', 'Country','Agg_Marks'],
                      index=['a', 'b', 'c', 'd', 'e', 'f'])

# here we set Float column 'Agg_Marks' as index of data frame
# using dataframe.set_index() function
df = df.set_index('Agg_Marks')


# Displaying the Data frame
df
```

Output :

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

Agg_Marks	Name	Age	City	Country
85.96	jack	34	Sydeny	Australia
95.20	Riti	30	Delhi	India
85.25	Vansh	31	Delhi	India
74.21	Nanyu	32	Tokyo	Japan
99.63	Maychan	16	New York	US
47.28	Mike	17	las vegas	US

In the above example, we set the column 'Agg_Marks' as an index of the data frame.
Code #4: Setting three columns as MultiIndex in Pandas DataFrame

```
# importing pandas library
import pandas as pd

# creating and initializing a nested list
students = [['jack', 34, 'Sydeny', 'Australia',85.96,400],
          ['Riti', 30, 'Delhi', 'India',95.20,750],
          ['Vansh', 31, 'Delhi', 'India',85.25,101],
          ['Nanyu', 32, 'Tokyo', 'Japan',74.21,900],
          ['Maychan', 16, 'New York', 'US',99.63,420],
          ['Mike', 17, 'las vegas', 'US',47.28,555]]

# Create a DataFrame object
df = pd.DataFrame(students,
                  columns=['Name', 'Age', 'City', 'Country','Agg_Marks','ID'],
                    index=['a', 'b', 'c', 'd', 'e', 'f'])

# Here we pass list of 3 columns i.e 'Name', 'City' and 'ID'
# to dataframe.set_index() function
# to set them as multiIndex of dataframe
df = df.set_index(['Name','City','ID'])
```

Displaying the Data frame
df

Output :

Name	City	ID	Age	Country	Agg_Marks
jack	Sydeny	400	34	Australia	85.96
Riti	Delhi	750	30	India	95.20
Vansh	Delhi	101	31	India	85.25
Nanyu	Tokyo	900	32	Japan	74.21
Maychan	New York	420	16	US	99.63
Mike	las vegas	555	17	US	47.28

In the above example, we set the columns 'Name', 'City', and 'ID' as multiIndex of the data frame.

Find out the missing values

Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in a real-life scenarios. Missing Data can also refer to as NA(Not Available) values in pandas. In DataFrame sometimes many datasets simply arrive with missing data, either because it exists and was not collected or it never existed. For Example, Suppose different users being surveyed may choose not to share their income, some users may choose not to share the address in this way many datasets went missing. In Pandas missing data is represented by two value:

- None: None is a Python singleton object that is often used for missing data in Python code.
- NaN : NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

- isnull()
- notnull()
- dropna()
- fillna()
- replace()
- interpolate()

Checking for missing values using isnull() and notnull()

In order to check missing values in Pandas DataFrame, we use a function isnull() and notnull(). Both function help in checking whether a value is NaN or not. These function can also be used in Pandas Series in order to find null values in a series.

Checking for missing values using isnull()

In order to check null values in Pandas DataFrame, we use isnull() function this function return dataframe of Boolean values which are True for NaN values. **Code #1:**

```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np

# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

```
# creating a dataframe from list
df = pd.DataFrame(dict)

# using isnull() function
df.isnull()
```

	First Score	Second Score	Third Score
0	False	False	True
1	False	False	False
2	True	False	False
3	False	True	False

Output:

Code #2:

- Python

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv")

# creating bool series True for NaN values
bool_series = pd.isnull(data["Gender"])

# filtering data
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

---

displaying data only with Gender = NaN

data[bool_series]

Output: As shown in the output image, only the rows having Gender = NULL are displayed.

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
20	Lois	NaN	4/22/1995	7:18 PM	64714	4.934	True	Legal
22	Joshua	NaN	3/8/2012	1:58 AM	90816	18.816	True	Client Services
27	Scott	NaN	7/11/1991	6:58 PM	122367	5.218	False	Legal
31	Joyce	NaN	2/20/2005	2:40 PM	88657	12.752	False	Product
41	Christine	NaN	6/28/2015	1:08 AM	66582	11.308	True	Business Development
49	Chris	NaN	1/24/1980	12:13 PM	113590	3.055	False	Sales
51	NaN	NaN	12/17/2011	8:29 AM	41126	14.009	NaN	Sales
53	Alan	NaN	3/3/2014	1:28 PM	40341	17.578	True	Finance
60	Paula	NaN	11/23/2005	2:01 PM	48866	4.271	False	Distribution
64	Kathleen	NaN	4/11/1990	6:46 PM	77834	18.771	False	Business Development
69	Irene	NaN	7/14/2015	4:31 PM	100863	4.382	True	Finance
70	Todd	NaN	6/10/2003	2:26 PM	84692	6.617	False	Client Services
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
939	Ralph	NaN	7/28/1995	6:53 PM	70635	2.147	False	Client Services
945	Gerald	NaN	4/15/1989	12:44 PM	93712	17.426	True	Distribution
961	Antonio	NaN	6/18/1989	9:37 PM	103050	3.050	False	Legal
972	Victor	NaN	7/28/2006	2:49 PM	76381	11.159	True	Sales
985	Stephen	NaN	7/10/1983	8:10 PM	85668	1.909	False	Legal
989	Justin	NaN	2/10/1991	4:58 PM	38344	3.794	False	Legal
995	Henry	NaN	11/23/2014	6:09 AM	132483	16.655	False	Distribution

145 rows × 8 columns

Checking for missing values using notnull()

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

In order to check null values in Pandas Dataframe, we use notnull() function this function return dataframe of Boolean values which are False for NaN values.

Code #3:

```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np

# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe using dictionary
df = pd.DataFrame(dict)

# using notnull() function
df.notnull()
```

Output:

	First Score	Second Score	Third Score
0	True	True	False
1	True	True	True
2	False	True	True
3	True	False	True

Code #4:

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

- Python

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv")

# creating bool series True for NaN values
bool_series = pd.notnull(data["Gender"])

# filtering data
# displaying data only with Gender = Not NaN
data[bool_series]
```

Output: As shown in the output image, only the rows having Gender = NOT NULL are displayed.

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
5	Dennis	Male	4/18/1987	1:35 AM	115163	10.125	False	Legal
6	Ruby	Female	8/17/1987	4:20 PM	65476	10.012	True	Product
7	NaN	Female	7/20/2015	10:43 AM	45906	11.598	NaN	Finance
8	Angela	Female	11/22/2005	6:29 AM	95570	18.523	True	Engineering
9	Frances	Female	8/8/2002	6:51 AM	139852	7.524	True	Business Development
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
994	George	Male	6/21/2013	5:47 PM	98874	4.479	True	Marketing
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales

855 rows × 8 columns

Finding outliers using statistical methods

Since the data doesn't follow a normal distribution, we will calculate the outlier data points using the statistical method called interquartile range (IQR) instead of using Z-score. Using the IQR, the outlier data points are the ones falling below Q1–1.5 IQR or above Q3 + 1.5 IQR. The Q1 is the 25th percentile and Q3 is the 75th percentile of the dataset, and IQR represents the interquartile range calculated by Q3 minus Q1 (Q3–Q1).

Using the convenient pandas .quantile() function, we can create a simple Python function that takes in our column from the dataframe and outputs the outliers:

```
#create a function to find outliers using IQR

def find_outliers_IQR(df):

  q1=df.quantile(0.25)

  q3=df.quantile(0.75)

  IQR=q3-q1

  outliers = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]

  return outliers
```

Notice using .quantile() we can define Q1 and Q3. Next we calculate IQR, then we use the values to find the outliers in the dataframe. Since it takes a dataframe, we can input one or multiple columns at a time.

First run fare_amount through the function to return a series of the outliers.

```
outliers = find_outliers_IQR(df[“fare_amount”])

print(“number of outliers: “+ str(len(outliers)))

print(“max outlier value: “+ str(outliers.max()))

print(“min outlier value: “+ str(outliers.min()))
```

outliers validating the find_outliers_IQR function

```
number of outliers: 17167
max outlier value: 499.0
min outlier value: -52.0

6         24.50
30        25.70
34        39.50
39        29.00
48        56.80
          ...
199976    49.70
199977    43.50
199982    57.33
199985    24.00
199997    30.90
```

Using the IQR method, we find 17,167 fare_amount outliers in the dataset. I printed the min and max values to verify they match the statistics we saw when using the pandas describe() function, which helps confirm we calculated the outliers correctly.

We can also pass both fare_amount and passenger_count through the function to get back a dataframe of all rows instead of just the outliers. If the value is not an outlier, it will display as NaN (not a number):

outliers = find_outliers_IQR(df[[“passenger_count”,”fare_amount”]])

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

---

outliers find outliers IQR dataframe

	passenger_count	fare_amount
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	5.0	NaN
...	...	...
199995	NaN	NaN
199996	NaN	NaN
199997	NaN	30.9
199998	NaN	NaN
199999	NaN	NaN

Conclusion: -

In this experiment, we studied how using pandas and NumPy library we can pre-process the data.