## Activity 1

**Aim:** Install NumPy, pandas, matplotlib, Seaborn, sklearn in python 3.

# **Learning outcome**: Able to install and different operation in python.

###### Duration: 3.5 hours

# List of Hardware/Software requirements:

* Laptop/Computer with Windows
* Internet connection
* Python

# Program / Procedure:

##### Installing pandas.

##### Installing with pip

It is a package installation manager that makes installing Python libraries and frameworks straightforward.

As long as you have a newer version of Python installed (> Python 3.4), pip will be installed on your computer along with Python by default.

However, if you’re using an older version of Python, you will need to install pip on your computer before installing Pandas. The easiest way to do this is to upgrade to the latest version of Python available on <https://www.python.org>.

**Step 1**. Launch Command Prompt.

Press the Windows key on your keyboard or click on the Start button to open the start menu. Type “cmd,” and the Command Prompt app should appear as a listing in the start menu.

Open up the command prompt so you can install Pandas.

|  |
| --- |
|  |

**Step 2.** Now enter the command pip install pandas on terminal. This should launch the pip installer. The required files will be downloaded, and pandas will be ready to run on your computer.

|  |
| --- |
|  |

After the installation is complete, you will be able to use Pandas in your Python programs.

##### Installing numpy.

**Step 3.** Now enter the command pip install numpy on terminal. This should launch the pip installer. The required files will be downloaded, and numpy will be ready to run on your computer.

|  |
| --- |
|  |

##### Installing matplotlib.

**Step 4.** Now enter the command pip install matplotlib on terminal. This should launch the pip installer. The required files will be downloaded, and matplotlib will be ready to run on your computer.

|  |
| --- |
|  |

##### Installing seaborn.

**Step 5.** Now enter the command pip install seaborn on terminal. This should launch the pip installer. The required files will be downloaded, and seaborn will be ready to run on your computer.

|  |
| --- |
|  |

##### Installing sklearn.

**Step 6.** Now enter the command pip install seaborn on terminal. This should launch the pip installer. The required files will be downloaded, and sklearn will be ready to run on your computer.

|  |
| --- |
|  |

If you want to install some library into the Jupiter, use these commands.

* !pip install pandas

|  |
| --- |
|  |

* !pip install numpy

|  |
| --- |
|  |

* !pip install matplotlib

|  |
| --- |
|  |

* !pip install seaborn

|  |
| --- |
|  |

## Activity 2

**Aim:** Creating arrays in NumPy.

# **Learning outcome**: Able to install and different operation in python.

###### Duration: 3.5 hours

# List of Hardware/Software requirements:

* Laptop/Computer with Windows
* Internet connection
* Python
* Jupiter notebook / google colab

# Program / Procedure:

##### Here I am working on Jupiter notebook, you can use google colab also.

## Input:

|  |
| --- |
| # First, we need to import the numpy library and give it the alias 'np'  import numpy as np  # Create a NumPy array containing the elements 1, 2, and 3  arr = np.array([1, 2, 3])  # Print the contents of the 'arr' array  print(arr)  # The following line is not needed; it just evaluates 'arr' and doesn't do anything with it.  # If you want to see the output of 'arr' without using 'print', you can simply enter 'arr' in an interactive environment.  # arr  # Check and print the data type of the 'arr' array  type(arr) |

## Output:

|  |
| --- |
|  |

## Activity 3

**Aim:** Creating multidimensional array in NumPy.

# **Learning outcome**: Able to install and different operation in python.

###### Duration: 3.5 hours

# List of Hardware/Software requirements:

* Laptop/Computer with Windows
* Internet connection
* Python
* Jupiter notebook / google colab

# Program / Procedure:

## Input:

|  |
| --- |
| # Creating a 0-D array (Scalar) with a single element (42)  arr\_0d = np.array(42)  print('0-D array -',arr\_0d) # Print the 0-D array (scalar)  # Creating a 1-D array with elements 1, 2, 3, 4, and 5  arr\_1d = np.array([1, 2, 3, 4, 5])  print('\n1-D array',arr\_1d) # Print the 1-D array  # Creating a 2-D array (matrix) with two rows and three columns  arr\_2d = np.array([[1, 2, 3], [4, 5, 6]])  print('\n2-D array',arr\_2d) # Print the 2-D array (matrix)  # Creating a 3-D array with two "matrices" where each "matrix" is a 2-D array  arr\_3d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])  print('\n3-D array',arr\_3d) # Print the 3-D array |

## Output:

|  |
| --- |
|  |

## Activity 4

**Aim:** Numpy Operations, methods and attributes.

# **Learning outcome**: Able to install and different operation in python.

###### Duration: 3.5 hours

# List of Hardware/Software requirements:

* Laptop/Computer with Windows
* Internet connection
* Python
* Jupiter notebook / google colab

# Program / Procedure:

## Input:

|  |
| --- |
| # Create two example arrays  arr1 = np.array([1, 2, 3, 4, 5])  arr2 = np.array([6, 7, 8, 9, 10])  # Basic NumPy array attributes and operations  # Shape attribute: Get the shape of the array (dimensions)  shape = arr1.shape  print("Shape of arr1:", shape)  # Data type attribute: Get the data type of the array  dtype = arr1.dtype  print("Data type of arr1:", dtype)  # Size attribute: Get the number of elements in the array  size = arr1.size  print("Size of arr1:", size)  # Reshape method: Reshape the array to a different shape  reshaped\_arr = arr1.reshape(1, 5)  print("Reshaped arr1:", reshaped\_arr)  # Element-wise operations  # Addition  addition = arr1 + arr2  print("arr1 + arr2:", addition)  # Subtraction  subtraction = arr2 - arr1  print("arr2 - arr1:", subtraction)  # Multiplication  multiplication = arr1 \* arr2  print("arr1 \* arr2:", multiplication)  # Division  division = arr2 / arr1  print("arr2 / arr1:", division)  # NumPy array methods  # Sum method: Compute the sum of all elements in the array  arr1\_sum = arr1.sum()  print("Sum of arr1:", arr1\_sum)  # Mean method: Compute the mean (average) of the elements  arr1\_mean = arr1.mean()  print("Mean of arr1:", arr1\_mean)  # Max method: Find the maximum element in the array  arr2\_max = arr2.max()  print("Max of arr2:", arr2\_max)  # Min method: Find the minimum element in the array  arr1\_min = arr1.min()  print("Min of arr1:", arr1\_min)  # Indexing and Slicing  # Accessing elements by index  element\_at\_index\_2 = arr1[2]  print("Element at index 2 of arr1:", element\_at\_index\_2)  # Slicing: Get a subset of the array  subarray = arr1[1:4]  print("Subarray of arr1:", subarray) |

## Output:

|  |
| --- |
|  |

## Activity 5

**Aim:** Numpy case studies.

# **Learning outcome**: Able to install and different operation in python.

###### Duration: 3.5 hours

# List of Hardware/Software requirements:

* Laptop/Computer with Windows
* Internet connection
* Python
* Jupiter notebook / google colab

# Program / Procedure:

Outline of this article:

The following are the brief contents of what we will be covering in this article:

1. Installing Pandas to your computer
2. Reading a CSV file in Pandas and checking the read file
3. Getting some basic information about the data read in Pandas

As I am planning to continue Python Pandas as a series of articles, it is a good idea to mention the topics we will cover in the upcoming articles:

1. Filtering in Pandas
2. Adding/removing columns/rows and updating them
3. Sorting data, grouping and aggregating in Pandas
4. Cleaning issues in Pandas and examples

We are going to answer the following questions in this article:

1. How can I install Pandas to my computer?
2. How can I load a CSV file as a Pandas DataFrame?
3. How can I explore the basic information about a loaded CSV file (Pandas table)?

So, let’s get to work and start exploring!!

We are going to use a CSV file downloaded from Kaggle named as “Are your employees burning out?”. It was already in my local drive. This dataset is also available for public use in Kaggle.com. Just go ahead and download it if you would like to do the same exercises with me in this article.

**Step 1.** Install Pandas into the Jupiter notebook.

|  |
| --- |
|  |

**Step 2.** Import the Pandas library and give it the alias 'pd'.

|  |
| --- |
| import pandas as pd |

**Step 3.** Read a CSV file named "train.csv" with a comma separator and store it in the 'data' DataFrame.

|  |
| --- |
| data = pd.read\_csv("train.csv", sep=",") |

**Step 4.** Now print the data.

|  |
| --- |
|  |

**Step 5.** Display the first 10 rows of the 'data' DataFrame.

|  |
| --- |
|  |

**Step 6.** Display the last 10 rows of the 'data' DataFrame.

|  |
| --- |
|  |

**Step 7.** Print the column names of the 'data' DataFrame.

|  |
| --- |
|  |

**Step 8.** Print the shape (number of rows and columns) of the 'data' DataFrame.

|  |
| --- |
|  |

**Step 9.** Print information about the 'data' DataFrame, including data types, non-null counts, and memory usage.

|  |
| --- |
|  |

Information displayed about the DataFrame after data.info() function. By default, this function shows the number of non-null entries in each column.

Above, we see that first five columns are probably filled with variables with a string type (or maybe with a Date type); and the remaining four columns have a float variable type. Please note that some other information is also included just above the variables/columns table.

The number of non-null values in each column are also shown.

As a final tool in this article, we can use the describe() function to see the basic statistical information about our columns with numerical values, which are Designation, Resource Allocation, Mental Fatigue Score and Burn Rate in this example.

Step 10. Print a summary of statistics for the 'data' DataFrame.

|  |
| --- |
|  |

The basic statistics shown after data.describe() function

Total number of counts in each column (variable), averages, standard deviations, minimum values, maximum values and 25%, 50%(median) and 75% percentiles are shown after the describe function. With additional parameters passed to describe function, additional information may also be displayed.

## Activity 6

**Aim:** Understanding Pandas series and dataframe.

# **Learning outcome**: Able to install and different operation in python.

###### Duration: 3.5 hours

# List of Hardware/Software requirements:

* Laptop/Computer with Windows
* Internet connection
* Python
* Jupiter notebook / google colab

# Program / Procedure:

**Step 1.** Creating a simple Series.

|  |
| --- |
| # Import the Pandas library  import pandas as pd  # Create a simple Series from a list of values  data = [10, 20, 30, 40, 50]  series = pd.Series(data)  # Display the Series  print(series) |

Output:

|  |
| --- |
|  |

**Step 2.** Check type of Series.

|  |
| --- |
|  |

**Step 3.** Creating Dataframe from Series.

|  |
| --- |
| import pandas as pd  # Create two Series  series1 = pd.Series([1, 2, 3, 4, 5], name='Series1')  series2 = pd.Series(['A', 'B', 'C', 'D', 'E'], name='Series2')  # Create a DataFrame by combining the Series  df = pd.DataFrame({'Column1': series1, 'Column2': series2})  # Display the DataFrame  print(df) |

Output:

|  |
| --- |
|  |

**Step 4.** Add series externally in dataframe.

|  |
| --- |
| import pandas as pd  # Create a DataFrame  data = {'Name': ['Alice', 'Bob', 'Charlie'],  'Age': [25, 30, 35]}  df = pd.DataFrame(data)  # Create a new Series  new\_series = pd.Series(['Engineer', 'Manager', 'Designer'], name='Job')  # Add the new Series to the DataFrame  df['Job'] = new\_series  # Display the updated DataFrame  print(df) |

Output:

|  |
| --- |
|  |

**Step 5.** Missing value in dataframe.

|  |
| --- |
| import pandas as pd  # Create a DataFrame  data = {'Name': ['Alice', 'Bob', 'Charlie'],  'Age': [25, 30, 35]}  df = pd.DataFrame(data)  # Create a new Series  new\_series = pd.Series(['Engineer', 'Manager'], name='Job')  # Add the new Series to the DataFrame  df['Job'] = new\_series  # Display the updated DataFrame  print(df) |

Output:

|  |
| --- |
|  |

**Step 6.** Data Plot on graph

|  |
| --- |
| import pandas as pd  import matplotlib.pyplot as plt  # Create a DataFrame  data = {'Name': ['Alice', 'Bob', 'Charlie'],  'Age': [25, 30, 35]}  df = pd.DataFrame(data)  # Create a bar plot using the 'Name' and 'Age' columns  plt.figure(figsize=(8, 4)) # Set the figure size  plt.bar(df['Name'], df['Age']) # Create a bar plot  plt.title('Age of Individuals') # Set the plot title  plt.xlabel('Name') # Label for the x-axis  plt.ylabel('Age') # Label for the y-axis  plt.grid(True) # Add grid lines  # Show the plot  plt.show() # Display the plot |

Output:

|  |
| --- |
|  |

## Activity 7

**Aim:** Pandas ingestion of data from csv, json, html, excel, text files.

# **Learning outcome**: Able to install and different operation in python.

###### Duration: 3.5 hours

# List of Hardware/Software requirements:

* Laptop/Computer with Windows
* Internet connection
* Python
* Jupiter notebook / google colab

# Program / Procedure:

**Streamlined Data Ingestion with Pandas**

Data Ingestion is the process of, transferring data, from varied sources to an approach, where it can be analyzed, archived, or utilized by an establishment. The usual steps, involved in this process, are drawing out data, from its current place, converting the data, and, finally loading it, in a location, for efficient research. Python provides many such tools, and, frameworks for data ingestion. These include Bonobo, Beautiful Soup4, Airflow, Pandas, etc. In this article, we will learn about Data Ingestion with Pandas library.

**Data Ingestion with Pandas:**

Data Ingestion with Pandas, is the process, of shifting data, from a variety of sources, into the Pandas DataFrame structure. The source of data can be varying file formats such as Comma Separated Data, JSON, HTML webpage table, Excel. In this article, we will learn about, transferring data, from such formats, into the destination, which is a Pandas dataframe object.

**File Formats for Ingestion:**

In this article, we will be converting, data present in the following files, to dataframe structures –

* Read data from CSV file
* Read data from Excel file
* Read data from JSON file

**Read data from csv file.**

|  |
| --- |
| # Step 1: Import the Pandas library  import pandas as pd  # Step 2: Read the CSV file into a DataFrame  # We use the `pd.read\_csv()` function to read the data from the "**Employee.csv**" file  # and store it in a Pandas DataFrame named 'df'.  df = pd.read\_csv('Employee.csv')  # Step 3: Display the DataFrame  # We can now display the contents of the DataFrame 'df', which contains the data from the CSV file.  df |

Output:

|  |
| --- |
|  |

**Read data from excel file.**

|  |
| --- |
| # Step 1: Import the Pandas library  import pandas as pd  # Step 2: Read the Excel file into a DataFrame  # We use the `pd.read\_excel()` function to read the data from the "**sample\_submission\_excel.xlsx**" Excel file  # and store it in a Pandas DataFrame named 'df'.  df = pd.read\_excel('sample\_submission\_excel.xlsx')  # Step 3: Display the DataFrame  # We can now display the contents of the DataFrame 'df', which contains the data from the Excel file.  df |

Output:

|  |
| --- |
|  |

**Read data from json file.**

|  |
| --- |
| import pandas as pd  # Step 1: Read the JSON file into a DataFrame  # Use the `pd.read\_json()` function to read the data from the "**text.json**" file  # and store it in a Pandas DataFrame.  df = pd.read\_json('text.json')  # Step 2: Display the DataFrame  # You can display the contents of the DataFrame 'df' to inspect the data.  print(df) |

Output:

|  |
| --- |
|  |

## Activity 8

**Aim:** Pandas functionalities for Series & Data Frames.

# **Learning outcome**: Able to install and different operation in python.

###### Duration: 3.5 hours

# List of Hardware/Software requirements:

* Laptop/Computer with Windows
* Internet connection
* Python
* Jupiter notebook / google colab

# Program / Procedure:

**Python | Pandas Series**

Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index. Pandas Series is nothing but a column in an excel sheet.

Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

**Creating a Pandas Series.**

|  |
| --- |
| # Step 1: Import the Pandas library  import pandas as pd  # Step 2: Create a Series from a list of values  # We create a simple Series by using the `pd.Series()` constructor.  # You can pass a list of values as the data to create the Series.  data = [10, 20, 30, 40, 50]  series = pd.Series(data)  # Step 3: Display the Series  # We can display the created Series by using the `print()` function.  print(series) |

**Output:**

|  |
| --- |
|  |

**Creating a Series from Lists.**

|  |
| --- |
| # Step 1: Import the Pandas library  import pandas as pd  # Step 2: Define lists  # We start by defining two lists: one for names and one for ages.  names = ["Alice", "Bob", "Charlie", "David"]  ages = [25, 30, 35, 40]  # Step 3: Create a Pandas Series  # We use the `pd.Series()` constructor to create a Series from the lists.  # The 'names' list becomes the data, and the 'ages' list becomes the index.  series = pd.Series(data=names, index=ages, name='Name')  # Step 4: Display the Series  # We can display the Series to inspect the results.  print(series) |

**Output:**

|  |
| --- |
|  |

**Accessing element of Series.**

|  |
| --- |
| import pandas as pd  import numpy as np  data=np.array([1,2,3,'a','b','c','d'])  ser=pd.Series(data)  ser[:5] |

**Output:**

|  |
| --- |
|  |

**Accessing Element Using Label (index), and loc[ ].**

|  |
| --- |
| # Step 1: Import the Pandas library  import pandas as pd  # Step 2: Create a Pandas Series  # We create a simple Series using a dictionary where keys are labels (indexes).  data = {'Alice': 25, 'Bob': 30, 'Charlie': 35, 'David': 40}  series = pd.Series(data, name='Age')  # Step 3: Accessing an element using square brackets  # We can access an element in the Series using its label (index) with square brackets.  # For example, to access the age of 'Alice':  alice\_age = series['Alice']  # Step 4: Accessing an element using .loc[]  # Alternatively, we can use the .loc[] accessor to access an element by label.  # For example, to access the age of 'Charlie':  charlie\_age = series.loc['Charlie']  # Step 5: Display the accessed elements  # We can display the accessed elements to inspect the results.  print("Alice's Age:", alice\_age)  print("Charlie's Age:", charlie\_age) |

**Output:**

|  |
| --- |
|  |

**Creating a Pandas DataFrame.**

In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc. Dataframe can be created in different ways here are some ways by which we create a dataframe:

|  |
| --- |
| # Step 1: Import the Pandas library  import pandas as pd  # Step 2: Create a Pandas DataFrame  # We create a DataFrame from a dictionary where keys become column names.  data = {  'Name': ['Alice', 'Bob', 'Charlie'],  'Age': [25, 30, 35]  }  df = pd.DataFrame(data)  # Step 3: Display the DataFrame  # We can display the DataFrame to inspect the results.  print(df) |

**Output:**

|  |
| --- |
|  |

## Activity 9

**Aim:** Grouping, Merging, concatenating, joining, segregation.

# **Learning outcome**: Able to install and different operation in python.

###### Duration: 3.5 hours

# List of Hardware/Software requirements:

* Laptop/Computer with Windows
* Internet connection
* Python
* Jupiter notebook / google colab

# Program / Procedure:

**Grouping**

|  |
| --- |
| # Step 1: Import the Pandas library  import pandas as pd  # Step 2: Create a DataFrame  data = {'Category': ['A', 'B', 'A', 'B', 'A'],  'Sales': [100, 150, 200, 250, 300]}  df = pd.DataFrame(data)  # Step 3: Grouping by 'Category' and calculating the sum of 'Sales'  grouped = df.groupby('Category')['Sales'].sum()  print("Grouped Data:")  print(grouped) |

**Output:**

|  |
| --- |
|  |

**Merging**

|  |
| --- |
| # Step 4: Create two DataFrames  df1 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['Alice', 'Bob', 'Charlie']})  df2 = pd.DataFrame({'ID': [2, 3, 4], 'Salary': [50000, 60000, 70000]})  # Step 5: Merging the DataFrames on 'ID'  merged\_df = pd.merge(df1, df2, on='ID', how='inner')  print("Merged Data:")  print(merged\_df) |

**Output:**

|  |
| --- |
|  |

**Concatenating**

|  |
| --- |
| # Step 6: Create two DataFrames  df1 = pd.DataFrame({'A': ['A0', 'A1'], 'B': ['B0', 'B1']})  df2 = pd.DataFrame({'A': ['A2', 'A3'], 'B': ['B2', 'B3']})  # Step 7: Concatenating the DataFrames along rows  concatenated = pd.concat([df1, df2])  print("Concatenated Data:")  print(concatenated) |

**Output:**

|  |
| --- |
|  |

**Joining**

|  |
| --- |
| # Step 8: Create two DataFrames  df1 = pd.DataFrame({'key': ['A', 'B', 'C'], 'value1': [1, 2, 3]})  df2 = pd.DataFrame({'key': ['B', 'C', 'D'], 'value2': [4, 5, 6]})  # Step 9: Joining the DataFrames on 'key' using an outer join  joined\_df = df1.join(df2.set\_index('key'), on='key', how='outer')  print("Joined Data:")  print(joined\_df) |

**Output:**

|  |
| --- |
|  |

**Data Segregation**

|  |
| --- |
| # Step 10: Filtering data based on a condition  high\_sales = df[df['Sales'] > 200]  low\_sales = df[df['Sales'] <= 200]  print("High Sales Data:")  print(high\_sales)  print("Low Sales Data:")  print(low\_sales) |

**Output:**

|  |
| --- |
|  |

## Activity 10

**Aim:** Python lambda function operations on series or data frames.

# **Learning outcome**: Able to install and different operation in python.

###### Duration: 3 hours

# List of Hardware/Software requirements:

* Laptop/Computer with Windows
* Internet connection
* Python
* Jupiter notebook / google colab

# Program / Procedure:

Input:

|  |
| --- |
| import pandas as pd  # Create a DataFrame  df = pd.DataFrame({'A': [1, 2, 3, 4, 5]})  # 1. Filtering Rows:  # Filter rows where column 'A' is greater than 2  filtered\_df = df[df['A'].apply(lambda x: x > 2)]  print("Filtering Rows:",filtered\_df)  # 2. Applying Functions to Columns:  # Apply a lambda function to calculate the square of each element in column 'A'  df['A\_squared'] = df['A'].apply(lambda x: x\*\*2)  print("\nApplying functions to columns:",df)  # 3. Combining Columns:  # Create a new column 'A\_plus\_B' by adding the values of columns 'A' and 'B'  df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})  df['A\_plus\_B'] = df.apply(lambda row: row['A'] + row['B'], axis=1)  print("\nCombining Columns:",df)  # 4. Custom Aggregation:  # Calculate the average value of 'Value' within each 'Category' using a lambda function  df = pd.DataFrame({'Category': ['A', 'A', 'B', 'B'], 'Value': [1, 2, 3, 4]})  result = df.groupby('Category')['Value'].agg(lambda x: sum(x) / len(x))  print("\nCustom Aggregation:",result)  # 5. Conditional Operations:  # Create a new column 'A\_category' based on whether 'A' is even or odd  df = pd.DataFrame({'A': [1, 2, 3, 4, 5]})  df['A\_category'] = df['A'].apply(lambda x: 'Even' if x % 2 == 0 else 'Odd')  print("\nCategory Operations:", df)  # 6. Sorting:  # Sort the DataFrame based on a custom key using a lambda function  df = pd.DataFrame({'A': [1, 3, 2, 5, 4]})  sorted\_df = df.sort\_values(by='A', key=lambda x: x % 2)  print("\nSorting",sorted\_df) |

**Output:**

|  |
| --- |
|  |

## Activity 11

**Aim:** Dealing with missing and noisy data.

# **Learning outcome**: Able to install and different operation in python.

###### Duration: 3 hours

# List of Hardware/Software requirements:

* Laptop/Computer with Windows
* Internet connection
* Python
* Jupiter notebook / google colab

# Program / Procedure:

**Dealing with Missing data**

**Input:**

|  |
| --- |
| import pandas as pd  df = pd.DataFrame({'Job Position': ['CEO', 'Senior Manager', 'Junior Manager', 'Employee', 'Assistant Staff'], 'Years of Experience':[5, 4, 3, None, 1], 'Salary':[100000,80000,None,40000, 20000]})  df |

**Output**

|  |
| --- |
|  |

**Some of the ways to handle missing data are listed below:**

**Data Removal**

Remove the missing data rows (data points) from the dataset. However, when using this technique will decrease the available dataset and in turn result in less robustness of data point if the size of dataset is originally small.

**Input:**

|  |
| --- |
| dropped\_df=df.drop([2,3], axis=0)  dropped\_df |

**Output:**

|  |
| --- |
|  |

**Fill missing value through statistical imputation**

Fill the missing data by taking the mean or median of the available data points. Generally, the median of the data points is used to fill the missing values as it is not affected heavily by outliers like the mean. Here, we have used the median to fill the missing data.

**Input:**

|  |
| --- |
| import pandas as pd  df = pd.DataFrame({'Job Position': ['CEO', 'Senior Manager', 'Junior Manager', 'Employee', 'Assistant Staff'], 'Years of Experience':[5, 4, 3, None, 1], 'Salary':[100000,80000,None,40000, 20000]})  # Filling each column with their mean values  df['Years of Experience'] = df['Years of Experience'].fillna(df['Years of Experience'].mean())  df['Salary'] = df['Salary'].fillna(df['Salary'].mean())  # Viewing the dataframe  df |

**Output:**

|  |
| --- |
|  |

## Activity 12

**Aim:** Finding outliers.

# **Learning outcome**: Able to install and different operation in python.

###### Duration: 3 hours

# List of Hardware/Software requirements:

* Laptop/Computer with Windows
* Internet connection
* Python
* Jupiter notebook / google colab

# Program / Procedure:

An Outlier is a data-item/object that deviates significantly from the rest of the (so-called normal) objects. They can be caused by measurement or execution errors. The analysis for outlier detection is referred to as outlier mining. There are many ways to detect the outliers, and the removal process is the data frame same as removing a data item from the panda’s data frame.

Here pandas data frame is used for a more realistic approach as in real-world project need to detect the outliers arouse during the data analysis step, the same approach can be used on lists and series-type objects.

Dataset:

Dataset used is Boston Housing dataset as it is preloaded in the sklearn library.

**Input:**

|  |
| --- |
| import numpy as np  from sklearn import datasets  # Load the Iris dataset  iris = datasets.load\_iris()  X = iris.data # Feature matrix  # Calculate the Z-Scores for each feature  z\_scores = np.abs((X - X.mean(axis=0)) / X.std(axis=0))  # Set a threshold for identifying outliers (e.g., z\_score > 2.0)  threshold = 2.0  # Find the indices of outliers  outlier\_indices = np.argwhere(z\_scores > threshold)  # Print the indices and values of outliers  for i, j in outlier\_indices:  print(f"Outlier at index {i}, feature {j}: Z-Score = {z\_scores[i, j]}") |

**Output:**

|  |
| --- |
|  |

## Activity 13

**Aim:** Visualizing your data through matplotlib under basic charts.

# **Learning outcome**: Able to install and different operation in python.

###### Duration: 5 hours

# List of Hardware/Software requirements:

* Laptop/Computer with Windows
* Internet connection
* Python
* Jupiter notebook / google colab

# Program / Procedure:

* 1. **Line Chart:**

**Input:**

|  |
| --- |
| import matplotlib.pyplot as plt  # Sample data  x = [1, 2, 3, 4, 5]  y = [10, 8, 12, 6, 14]  # Create a line chart  plt.plot(x, y, marker='o', linestyle='-')  # Add labels and a title  plt.xlabel('X-axis')  plt.ylabel('Y-axis')  plt.title('Line Chart Example')  # Show the chart  plt.show() |

**Output:**

|  |
| --- |
|  |

* 1. **Bar Chart:**

**Input:**

|  |
| --- |
| import matplotlib.pyplot as plt  # Sample data  categories = ['Category A', 'Category B', 'Category C', 'Category D']  values = [15, 7, 10, 12]  # Create a bar chart  plt.bar(categories, values)  # Add labels and a title  plt.xlabel('Categories')  plt.ylabel('Values')  plt.title('Bar Chart Example')  # Show the chart  plt.show() |

**Output:**

|  |
| --- |
|  |

* 1. **Scatter Plot**

**Input:**

|  |
| --- |
| import matplotlib.pyplot as plt  # Sample data  x = [1, 2, 3, 4, 5]  y = [10, 8, 12, 6, 14]  # Create a scatter plot  plt.scatter(x, y, marker='o', color='blue')  # Add labels and a title  plt.xlabel('X-axis')  plt.ylabel('Y-axis')  plt.title('Scatter Plot Example')  # Show the chart  plt.show() |

**Output:**

|  |
| --- |
|  |

* 1. **Histogram**

Input:

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  # Generate some random data as an example  data = np.random.randn(1000) # Replace with your dataset  # Create a histogram  plt.hist(data, bins=20, color='blue', edgecolor='black')  # Add labels and a title  plt.xlabel('Value')  plt.ylabel('Frequency')  plt.title('Histogram Example')  # Show the histogram  plt.show() |

**Output:**

|  |
| --- |
|  |

* 1. **Pie Chart**

**Input:**

|  |
| --- |
| import matplotlib.pyplot as plt  # Sample data  categories = ['Category A', 'Category B', 'Category C', 'Category D']  values = [30, 15, 25, 20]  # Create a pie chart  plt.pie(values, labels=categories, autopct='%1.1f%%', startangle=140, shadow=True)  # Add a title  plt.title('Pie Chart Example')  # Show the pie chart  plt.show() |

**Output:**

|  |
| --- |
|  |

If you want to save any fig then run this command before the show.

plt.savefig('mtert.jpg', format='jpg')