**Data Processing with Pandas**

**Data Processing**is an important part of any task that includes data-driven work. It helps us to provide meaningful insights from the data. As we know Python is a widely used programming language, and there are various libraries and tools available for data processing.

In this article, we are going to see ***Data Processing in Python, Loading, Printing rows and Columns, Data frame summary, Missing data values Sorting and Merging Data Frames, Applying Functions, and Visualizing Dataframes.***

* What is Data Processing in Python?
* What is Pandas?
* Loading Data in Pandas DataFrame
* Printing rows of the Data
* Printing the column names of the DataFrame
* Summary of Data Frame
* Descriptive Statistical Measures of a DataFrame
* Missing Data Handing
* Sorting DataFrame values
* Merge Data Frames
* Apply Function
* By using the lambda operator
* Visualizing DataFrame

**Loading Data in Pandas DataFrame**

Reading CSV file using **pd.read\_csv** and loading data into a data frame. Import pandas as using pd for the shorthand. You can download the data from here.

#Importing pandas library

import pandas as pd

#Loading data into a DataFrame

data\_frame=pd.read\_csv('Mall\_Customers.csv')

**Printing rows of the Data**

By default, **data\_frame.head()** displays the first five rows and **data\_frame.tail()** displays last five rows. If we want to get first ‘n’ number of rows then we use, **data\_frame.head(n)**similar is the syntax to print the last n rows of the data frame.

#displaying first five rows

display(data\_frame.head())

#displaying last five rows

display(data\_frame.tail())

**Printing the column names of the DataFrame**

* Python3

# Program to print all the column name of the dataframe

print(list(data\_frame.columns))

**Summary of Data Frame**

data\_frame.info()

**Descriptive Statistical Measures of a DataFrame**

The **describe()**function outputs descriptive statistics which include those that summarize the central tendency, dispersion, and shape of a dataset’s distribution, excluding NaN values. For numeric data, the result’s index will include count, mean, std, min, and max as well as lower, 50, and upper percentiles. For object data (e.g. strings), the result’s index will include count, unique, top, and freq.

data\_frame.describe()

**Missing Data Handing**

**Find missing values in the dataset**

The **isnull( )** detects the missing values and returns a boolean object indicating if the values are NA. The values which are none or empty get mapped to true values and not null values get mapped to false values.

data\_frame.isnull( )

**Removing missing values**

The **data\_frame.dropna( )** function removes columns or rows which contains atleast one missing values.

data\_frame = data\_frame.dropna()

**Removing rows**

By using the **drop(index)**function we can drop the row at a particular index. If we want to replace the data\_frame with the row removed then add **inplace = True** in the drop function.

#Removing 4th indexed value from the dataframe

data\_frame.drop(4).head()

**Renaming rows**

The rename function can be used to rename the rows or columns of the data frame.

data\_frame.rename({0:"First",1:"Second"})

**Adding new column**

#Creates a new column with all the values equal to 1

data\_frame['NewColumn'] = 1

data\_frame.head()

**Sorting DataFrame values**

**Sort by column**

The **sort\_values( )** are the values of the column whose name is passed in the **by** attribute in the ascending order by default we can set this attribute to false to sort the array in the descending order.

data\_frame.sort\_values(by='Age', ascending=False).head()

**Sort by multiple columns**

data\_frame.sort\_values(by=['Age','Annual Income (k$)']).head(10)

**Merge Data Frames**

The **merge()** function in pandas is used for all standard database join operations. Merge operation on data frames will join two data frames based on their common column values. Let’s create a data frame.

#Creating dataframe1

df1 = pd.DataFrame({

'Name':['Jeevan', 'Raavan', 'Geeta', 'Bheem'],

'Age':[25, 24, 52, 40],

'Qualification':['Msc', 'MA', 'MCA', 'Phd']})

df1

#Creating dataframe2

df2 = pd.DataFrame({'Name':['Jeevan', 'Raavan', 'Geeta', 'Bheem'],

'Salary':[100000, 50000, 20000, 40000]})

df2

Now. let’s merge these two data frames created earlier.

* Python3

|  |
| --- |
| #Merging two dataframes  df **=** pd.merge(df1, df2)  df |

Now we will create another data frame.

**By defining a function beforehand**

The apply( ) function is used to iterate over a data frame. It can also be used with lambda functions.

* Python3

|  |
| --- |
| # Apply function  **def** fun(value):  **if** value > 70:  **return** "Yes"  **else**:  **return** "No"    data\_frame['Customer Satisfaction'] **=** data\_frame['Spending Score (1-100)'].apply(fun)  data\_frame.head(10) |

**By using the lambda operator**

This syntax is generally used to apply log transformations and normalize the data to bring it in the range of 0 to 1 for particular columns of the data.

* Python3

|  |
| --- |
| const **=** data\_frame['Age'].max()  data\_frame['Age'] **=** data\_frame['Age'].apply(**lambda** x: x**/**const)  data\_frame.head() |

**Visualizing DataFrame**

**Scatter plot**

The **plot( )** function is used to make plots of the data frames.

* Python3

|  |
| --- |
| # Visualization  data\_frame.plot(x **=**'CustomerID', y**=**'Spending Score (1-100)',kind **=** 'scatter') |

**Histogram**

The **plot.hist( )** function is used to make plots of the data frames.

* Python3

|  |
| --- |
| data\_frame.plot.hist() |