IntelliPaat

LAB 01 – **Pandas in Data Engineering**

Introduction

Pandas is a powerful data manipulation and analysis library that has become an essential tool for data engineers, data analysts, data scientists, and anyone working with data in Python.

Pandas has two main data structures as shown below,

1. Series (1-dimensional)
2. DataFrame (2-dimensional)

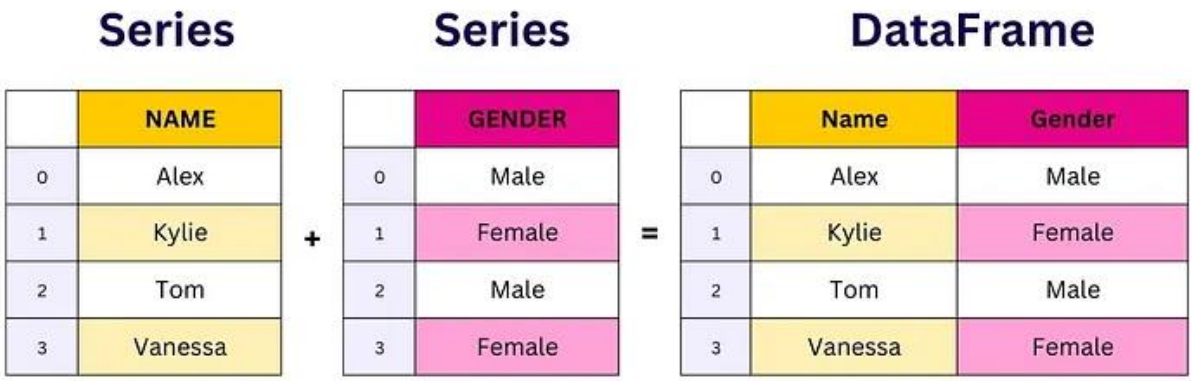


Figure – 01

cater to a wide range of needs in finance, statistics, social science, and engineering. The library is particularly adept at handling missing data, adjusting data sizes, intelligent data slicing, and indexing. Its flexibility extends to reshaping and pivoting datasets and robust input/output capabilities, including support for various file formats and databases.

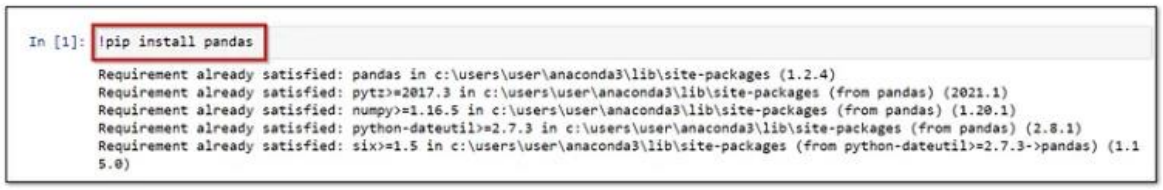
Please keep in mind that this Hands-on exercise is executed within a Jupyter Notebook environment. If you haven't configured Jupyter Notebook on your system. Please ensure that Jupyter Notebook has been installed/configured as explained/demonstrated during the pre-requisite training.

Let's start the implementation

1.) Install the package

Before starting with the demo, you will have to install the package. Run the below command to install the Pandas package.

```
pip install pandas
```



2.) Importing the Pandas package

```
import pandas as pd
```

The common shortcut of Pandas is pd so instead of writing pandas you can write pd as an alias.

** Now that the package is successfully installed, we will import the dataset.

3.) Import the dataset with read_csv

Note: For this Hands-on exercise, find the Supply Chain dataset from Kaggle.

```
scmDF = pd.read_csv('/path_of_your_file/ supply_chain_data.csv')
scmDF.head(5)
```

To view the contents of the dataset we are going to use the **scmDF.head().** If the () is left blank, then the first 5 elements of the table will be displayed. If you want a specific number of entries, then you can give the command as **scmDF.head(20)** to print the first 20 rows.

** You can use scmDF.tail() to display last 5 elements of the table

Run the describe() command to get a summary of numeric values in your dataset.

```
scmDF.describe()
```

Now if you want to read some specific columns only from the dataset, you can use the usecols argument to specify the column names that we want to work with. Let's work with just Product type, Price, Availability and Number of products sold columns.

```
scmDF = pd.read_csv("/path_of_your_file/ supply_chain_data.csv",
        usecols=[" Product type ", " Price "," Availability ", " Number of products sold"])
```

4.) Sort Columns based on specific criteria

We will sort columns based on numeric data. The sort_values() method sorts the DataFrame by the specified label.

```
scmDF.sort_values("Availability ").head(10)
```

To view the products based on Availablity in descending.

```
scmDF.sort_values("Availability ", ascending=False).head(10)
```

5.) Count the occurrences of variables

value_counts() function returns a Series containing counts of unique values. The resulting object will be in descending order so that the first element is the most frequently-occurring element. Excludes NA values by default.

To display the count by Product Type.

```
scmDF ["Product type "].value_counts()
```

6.) Data Filtering

Filtering data is a widely used method for extracting specific rows from a dataset by applying certain conditions. This process is akin to using the WHERE clause in SQL or employing the filter feature in Excel. In Python, the pandas library, built on top of NumPy, provides an effective approach for performing data filtering.

To display those product types which have more than 50.

```
scmDF[(scmDF["Product type"]=="skincare") & scmDF["Availability"]>50)].head(10)
```

7.) Null values (NaN)

One of the most common problems in data engineering is missing values. To detect them, there is a beautiful method which is called .isnull(). With this method, we can get a boolean series (True or False).

The isnull() and notnull() methods in Pandas address this issue by facilitating the identification and management of NULL values within a data frame DataFrame.

Display the products whose Number of products sold is unknown(null).

```
scmDF[scmDF["Number of products sold"].isna()]
```

8) Handling Missing Data

Dealing with missing values on **Number of products sold** using methods like dropna() and fillna().

```
scmDF.dropna(subset=["Number of products sold"], inplace=True)  # Remove rows with missing values
```

```
scmDF [Price].fillna(10, inplace=True)  # Fill missing values with default value as 10 to price column
```

9) Renaming Columns

Rename column **'Number of products sold' to 'Sold count'.**

```
scmDF.rename(columns={ 'Number of products sold': 'Sold count' }, inplace=True)
```

10) Changing Data Types

Convert data type of **Availablity** column to Float data type.

scmDF [Availablity] = scmDF ['Age'].astype(float)

11) Grouping and Aggregating

Group data and calculate aggregate statistics based on Product type column.

group_by_producttype = scmDF.groupby('Product type')

group_by_producttype.first()

group_by_producttype = scmDF.groupby('Product type').mean()

12) Pivoting Data

Reshape the products data using pivot table.

scm_pivot_table = scmDF.pivot_table(index='Product type', columns=' SKU', values=' Availability ')

13) Applying Functions

Use apply() to apply a function to double the price for all the products.

scmDF[IncreasedPrice] = scmDF['Price'].apply(lambda x: x ** 2)

14) Working with Excel

Finally let's export the manipulated/processed data to an Excel file.

scmDF.to_excel('output_scmproducts.xlsx', sheet_name='Sheet1', index=False)

Hurray, you have successfully completed the exercise.