Introduction to Pandas

Pandas is a powerful and open-source Python library. The Pandas library is used for data manipulation and analysis. Pandas consist of data structures and functions to perform efficient operations on data.
Pandas is well-suited for working with **tabular data**, such as **spreadsheets** or **SQL tables**.

Need for Pandas

- Data set cleaning, merging, and joining.
- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data.
- Columns can be inserted and deleted from DataFrame and higher-dimensional objects.
- Powerful group by functionality for performing split-apply-combine operations on data sets.
- Data Visualization.

Data Structures in Pandas Library

Pandas generally provide two data structures for manipulating data. They are:

- **Series**
- **DataFrame**

Pandas Series

A 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 **indexes**.

Creating a Series

Pandas Series is created by loading the datasets from existing storage (which can be a SQL database, a CSV file, or an Excel file).
Pandas Series can be created from lists, dictionaries, scalar values, etc.

Examples

```
import pandas as pd
import numpy as np

# Creating empty series
ser = pd.Series()
print("Pandas Series: ", ser)

# simple array
data = np.array(1,2,3,4,5])

ser = pd.Series(data)
print("Pandas Series:\n", ser)
```

Pandas DataFrame
Pandas DataFrame is a two-dimensional data structure with labeled axes (rows and columns).
Creating DataFrame
Pandas DataFrame is created by loading the datasets from existing storage (which can be a SQL database, a CSV file, or an Excel file).
Pandas DataFrame can be created from lists, dictionaries, a list of dictionaries, etc.

Examples:

```
import pandas as pd

# Calling DataFrame constructor
df = pd.DataFrame()
print(df)

# list of strings
lst = ['Welcome','to','all','genious']

# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
print(df)
```

Week-6

Perform following operations using pandas

a. Creating data frame

b. concat()

c. Setting conditions

d. Adding a new column

1. Creating a DataFrame

```
import pandas as pd

# Creating a sample DataFrame
data = {
   'Name': ['Alice', 'Bob', 'Charlie'],
   'Age': [25, 30, 35],
   'City': ['New York', 'Los Angeles', 'Chicago']
}

df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
```

2. Using concat() to Combine DataFrames

```
# Creating another DataFrame
data2 = {
   'Name': ['David', 'Eve'],
   'Age': [28, 22],
   'City': ['San Francisco', 'Houston']
}

df2 = pd.DataFrame(data2)

# Concatenating the two DataFrames
df_combined = pd.concat([df, df2], ignore_index=True)
print("\nConcatenated DataFrame:")
print(df_combined)
```

3. Setting Conditions

```
# Selecting rows where Age is greater than 25
df_filtered = df[df['Age'] > 25]
print("\nFiltered DataFrame (Age > 25):")
print(df_filtered)
```

4. Adding a New Column

```
# Adding a new column 'Salary' with random values
df['Salary'] = [50000, 60000, 70000]
print("\nDataFrame after adding 'Salary' column:")
```

```
print(df)
```

Week-7

Perform following operations using pandas

a. Filling NaN with string

b. Sorting based on column values

c. groupby()

1. Filling NaN with a String

```
import pandas as pd
import numpy as np

# Creating a DataFrame with NaN values
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, np.nan, 35, 40],
    'City': ['New York', 'Los Angeles', np.nan, 'Chicago']
}

df = pd.DataFrame(data)

# Filling NaN values with a string
df_filled = df.fillna('Unknown')
print("\nDataFrame after filling NaN with 'Unknown':")
print(df_filled)
```

---

2. Sorting Based on Column Values

```
# Sorting by Age column in ascending order
df_sorted = df.sort_values(by='Age', ascending=True)
print("\nDataFrame sorted by Age:")
print(df_sorted)
```

---

3. Using groupby()

```
# Creating a sample DataFrame
data = {
    'Department': ['HR', 'IT', 'HR', 'IT', 'Finance'],
    'Employee': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Salary': [50000, 60000, 55000, 65000, 70000]
}
```

```
df_group = pd.DataFrame(data)

# Grouping by 'Department' and calculating the mean salary
df_grouped = df_group.groupby('Department')['Salary'].mean()
print("\nAverage Salary by Department:")
print(df_grouped)
```

Week-8

Read the following file formats using pandas

a. Text files

b. CSV files

c. Excel files

d. JSON files

```
#importing required modules
import pandas as pd
```

```
#using this code upload all 4 categories of files from respective folder
from google.colab import files
uploaded = files.upload()
```

```
#reading text files
df_txt = pd.read_csv('sample1.txt.txt')
print("\nData from Text File:")
print(df_txt)
```

```
#reading csv files
df_csv = pd.read_csv('sample2.csv')
print("\nData from CSV File:")
print(df_csv)
```

```
#reading excel files
df_excel = pd.read_excel('sample3.xlsx', sheet_name='Sheet1')  # Specify sheet name if
needed
print("\nData from Excel File:")
print(df_excel)
```

```
#reading json files
df_json = pd.read_json('house-price.json',lines=True)
print("\nData from JSON File:")
print(df_json)
```