untitled3-1

September 3, 2024

1 Pandas

1.1 Getting familiar with pandas

->Pandas is a powerful library in Python used for data manipulation and analysis. It provides two primary data structures: 1. Series 2. DataFrame

Import the pandas as pd

[280]:
```
import pandas as pd
```

1. Series A Series is essentially a one-dimensional labeled array that can hold any data type, including integers, floats, strings, etc. It is similar to a column in a table or a single column in a spreadsheet.

[283]:
```
# Creating a Series from a list
data = [10, 20, 30, 40]
series = pd.Series(data)
print(series)
```

```
0    10
1    20
2    30
3    40
dtype: int64
```

[285]:
```
# Creating a Series with a custom index
data = [10, 20, 30, 40]
index = ['a', 'b', 'c', 'd']
series = pd.Series(data, index=index)
print(series)
```

```
a    10
b    20
c    30
d    40
dtype: int64
```

2. DataFrame A DataFrame is a two-dimensional labeled data structure with columns of potentially different types. It can be thought of as a table or a spreadsheet. It is built from one or more Series.

[288]:
```
# Creating a DataFrame from a dictionary
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['New York', 'Los Angeles', 'Chicago']
}
df = pd.DataFrame(data)
print(df)
```

```
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
2  Charlie   35      Chicago
```

[290]:
```
# Creating a DataFrame with a custom index
index = ['1', '2', '3']
df = pd.DataFrame(data, index=index)
print(df)
```

```
      Name  Age         City
1    Alice   25     New York
2      Bob   30  Los Angeles
3  Charlie   35      Chicago
```

Pandas makes it easy to create DataFrames and Series from various data sources. Let's go through how to create these data structures from lists, dictionaries, and CSV files

->Creating DataFrames and Series from Lists

[293]:
```
# Creating a Series from a list
data = [10, 20, 30, 40]
series = pd.Series(data)
print(series)
```

```
0    10
1    20
2    30
3    40
dtype: int64
```

[295]:
```
#Creating DataFrame from Lists:
#You can also create a DataFrame from a list of lists or a list of dictionaries.

#List of Lists:Each inner list represents a row in the DataFrame.
```

```
data = [
    [1, 'Alice', 23],
    [2, 'Bob', 30],
    [3, 'Charlie', 35]
]
columns = ['ID', 'Name', 'Age']
df = pd.DataFrame(data, columns=columns)
print(df)
```

```
   ID     Name  Age
0   1    Alice   23
1   2      Bob   30
2   3  Charlie   35
```

[297]:
```
# List of dictionaries:Each dictionary represents a row, with the keys as␣
↪column names.
data = [
    {'ID': 1, 'Name': 'Alice', 'Age': 23},
    {'ID': 2, 'Name': 'Bob', 'Age': 30},
    {'ID': 3, 'Name': 'Charlie', 'Age': 35}
]
df = pd.DataFrame(data)
print(df)
```

```
   ID     Name  Age
0   1    Alice   23
1   2      Bob   30
2   3  Charlie   35
```

->Creating DataFrames and Series from Dictionaries

[300]:
```
# Creating a Series from a dictionary:A dictionary where keys are the index␣
↪labels and values are the data.
data = {'a': 10, 'b': 20, 'c': 30}
series = pd.Series(data)
print(series)
```

```
a    10
b    20
c    30
dtype: int64
```

[302]:
```
# Creating a DataFrame from a dictionary of lists
import numpy as np
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25,np.NAN ,35],
    'City': ['New York', 'Los Angeles', 'Chicago'],
```

```
    'salary':[2000,30000,40000]
}
df = pd.DataFrame(data)
print(df)
```

```
      Name   Age         City  salary
0    Alice  25.0     New York    2000
1      Bob   NaN  Los Angeles   30000
2  Charlie  35.0      Chicago   40000
```

[387]:
```
# Creating a DataFrame from a dictionary of Series
#Similar to a dictionary of lists, but each value is a Series.
import pandas as pd
data1 = {
    'Name': pd.Series(['Alice', 'Bob', 'Charlie']),
    'Age': pd.Series([25,np.NAN,35]),
    'City': pd.Series(['New York', 'Los Angeles', 'Chicago']),
    'salary':pd.Series([2000,30000,40000]) }
df2 = pd.DataFrame(data)
print(df2)
```

```
      Name   Age         City  salary
0    Alice  25.0     New York    2000
1      Bob   NaN  Los Angeles   30000
2  Charlie  35.0      Chicago   40000
```

->Reading CSV Files: Pandas can easily read data from CSV files into DataFrames using the read_csv function.

Reading a CSV file into a DataFrame

[308]:
```
#CSV files can be read and written easily using pd.read_csv() and df.to_csv().
df2.to_csv('sample.csv',index=False)
```

[310]:
```
df3 = pd.read_csv('sample.csv')
print(df3)
```

```
      Name   Age         City  salary
0    Alice  25.0     New York    2000
1      Bob   NaN  Los Angeles   30000
2  Charlie  35.0      Chicago   40000
```

Selecting data

[313]:
```
# Select a single column
name = df['Name']
print(name)
```

```
0      Alice
1        Bob
2    Charlie
Name: Name, dtype: object
```

[315]:
```
# Select multiple columns
mul= df[['Name', 'City']]
print(mul)
```

```
      Name         City
0    Alice     New York
1      Bob  Los Angeles
2  Charlie      Chicago
```

[317]:
```
# Select rows by index
row= df.iloc[1]
print(row)
```

```
Name              Bob
Age               NaN
City      Los Angeles
salary          30000
Name: 1, dtype: object
```

Filtering Rows

[320]:
```
# Filter rows where Age is greater than 25 and City is 'Chicago'
filtered = df[(df['Age'] > 25) & (df['City'] == 'Chicago')]
print(filtered)
```

```
      Name   Age     City  salary
2  Charlie  35.0  Chicago   40000
```

Modifying Data

[323]:
```
#Add a New Column:
df['Occupation'] = ['Engineer', 'Doctor', 'Artist']
print(df)
```

```
      Name   Age         City  salary Occupation
0    Alice  25.0     New York    2000   Engineer
1      Bob   NaN  Los Angeles   30000     Doctor
2  Charlie  35.0      Chicago   40000     Artist
```

[325]:
```
# Drop a column
df = df.drop(columns=['Occupation'])
print(df)
```

```
      Name   Age         City  salary
0    Alice  25.0     New York    2000
```

```
1      Bob   NaN  Los Angeles   30000
2  Charlie  35.0      Chicago   40000
```

[327]:
```
# Rename a column
df = df.rename(columns={'Name': 'Full Name'})
print(df)
```

```
  Full Name   Age         City  salary
0     Alice  25.0     New York    2000
1       Bob   NaN  Los Angeles   30000
2   Charlie  35.0      Chicago   40000
```

Handling missing data

[329]:
```
# Display rows with missing data
print("\nRows with missing data:")
print(df2[df2.isnull().any(axis=1)])
```

```

Rows with missing data:
  Name  Age         City  salary
1  Bob  NaN  Los Angeles   30000
```

[331]:
```
# Fill missing data with a specific value
df_fill= df2.fillna({
    'Age': df2['Age'].mean(),  # Fill missing Age with the mean age
    'City': 'Unknown',        # Fill missing City with 'Unknown'
})
print(df_fill)
```

```
      Name   Age         City  salary
0    Alice  25.0     New York    2000
1      Bob  30.0  Los Angeles   30000
2  Charlie  35.0      Chicago   40000
```

[333]:
```
# Drop rows with missing data
df_dropped = df2.dropna()
print("\nDataFrame after dropping rows with missing values:")
print(df_dropped)
```

```

DataFrame after dropping rows with missing values:
      Name   Age      City  salary
0    Alice  25.0  New York    2000
2  Charlie  35.0   Chicago   40000
```

Removing Duplicates

[335]:
```python
# Remove duplicate rows based on 'Name' column
df2.drop_duplicates(subset='Name', keep='first', inplace=True)
print("\nDataFrame after removing duplicates based on 'Name':")
print(df2)
```

```
DataFrame after removing duplicates based on 'Name':
      Name   Age         City  salary
0    Alice  25.0     New York    2000
1      Bob   NaN  Los Angeles   30000
2  Charlie  35.0      Chicago   40000
```

Data Type Conversions

[337]:
```python
# Convert 'Salary' to float type
df2['salary'] = df2['salary'].astype(float)
print("\nDataFrame after data type conversions:")
print(df2)
```

```
DataFrame after data type conversions:
      Name   Age         City   salary
0    Alice  25.0     New York   2000.0
1      Bob   NaN  Los Angeles  30000.0
2  Charlie  35.0      Chicago  40000.0
```

[339]:
```python
#Generating Summary Statistics
print("\nSummary Statistics:")
print(df2.describe(include='all'))
```

```
Summary Statistics:
         Name        Age      City        salary
count       3   2.000000         3      3.000000
unique      3        NaN         3           NaN
top     Alice        NaN  New York           NaN
freq        1        NaN         1           NaN
mean      NaN  30.000000       NaN  24000.000000
std       NaN   7.071068       NaN  19697.715604
min       NaN  25.000000       NaN   2000.000000
25%       NaN  27.500000       NaN  16000.000000
50%       NaN  30.000000       NaN  30000.000000
75%       NaN  32.500000       NaN  35000.000000
max       NaN  35.000000       NaN  40000.000000
```

[341]:
```python
#Grouping Data
grouped = df2.groupby('City')
print(grouped['Age'].mean())  # Mean age per city
```

```
City
Chicago        35.0
Los Angeles     NaN
New York       25.0
Name: Age, dtype: float64
```

Merging DataFrames: Merging combines DataFrames based on a common column.

[371]:
```
# Create DataFrames
df3= pd.DataFrame({
    'ID': [1, 2, 3],
    'Name': ['Alice', 'Bob', 'Charlie']})
df4= pd.DataFrame({
    'ID': [1, 2, 4],
    'Salary': [70000, 80000, 60000]})
# Merge DataFrames on 'ID'
merged_df = pd.merge(df3, df4, on='ID', how='left')
print("Merged DataFrame:")
print(merged_df)
```

```
Merged DataFrame:
   ID     Name   Salary
0   1    Alice  70000.0
1   2      Bob  80000.0
2   3  Charlie      NaN
```

Joining DataFrames: Joining combines DataFrames based on their index.

[373]:
```
df_left = pd.DataFrame({'A': [1, 2]}, index=['a', 'b'])
df_right = pd.DataFrame({'B': [3, 4]}, index=['a', 'c'])
df_joined = df_left.join(df_right, how='inner')
print(df_joined)
```

```
   A  B
a  1  3
```

Concatenating DataFrames Concatenating stacks DataFrames vertically or horizontally.

[377]:
```
df5 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df6 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})
df_concat = pd.concat([df5, df6])
print(df_concat)
```

```
   A  B
0  1  3
1  2  4
0  5  7
1  6  8
```

Advantages of Using Pandas for Data Handling and Analysis Pandas is a powerful and flexible library for data manipulation and analysis in Python ##### 1. Efficient Data Handling Data Structures: Pandas introduces two primary data structures: Series (1-dimensional) and DataFrame (2-dimensional). These structures are optimized for performance and memory efficiency compared to traditional Python lists and dictionaries. ##### 2. Rich Data Manipulation Functions Pandas provides a wide range of built-in functions for data manipulation:

->Merging and Joining

->Concatenation

->GroupBy and Aggregation

3. Handling Missing Data Pandas offers robust methods for dealing with missing values:

->Filling and Interpolation: Functions like fillna() and interpolate()

->Dropping Missing Values: Function like dropna()

4. Data Cleaning and Transformation Pandas simplifies data cleaning and transformation tasks:

->String Operations: Methods for string manipulation (e.g., str.contains(), str.replace()) are available for preprocessing text data.

->Data Type Conversion: Functions like astype()

5.Ease of Use ->Intuitive Syntax: Pandas provides a user-friendly syntax that simplifies data manipulation tasks.

->Data Exploration: Methods like describe(), info(), and head() make it easy to explore and understand datasets quickly.

Real-world examplesin data cleaning,exploratory data analysis(EDA)

1. Financial Analysis: ->Data Cleaning: Financial datasets often contain missing values or erroneous entries. Pandas can be used to clean this data by filling missing values, handling duplicates, and correcting data types.

->EDA: Financial analysts use EDA to understand trends and patterns in stock prices, transaction volumes, and financial ratios.Pandas helps in calculating descriptive statistics, visualizing data distributions, and performing time-series analysis. ##### 2. Healthcare Data Analysis

->Data Cleaning: Healthcare datasets can have missing values, inconsistencies, or incorrect data entries. Pandas helps in cleaning and transforming this data for accurate analysis.

->EDA: EDA helps in understanding patient demographics, treatment effectiveness, and disease prevalence. ##### 3.Marketing Campaign Analysis ->Data Cleaning: Marketing data often includes various sources like customer surveys, ad click logs, and sales data. Pandas can merge these datasets and clean them for further analysis.

-> EDA: EDA involves analyzing campaign responses, conversion rates, and ROI. Pandas allows for the computation of conversion rates and the visualization of campaign performance.