Pandas

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

Series: A one-dimensional labeled array capable of holding any data type. DataFrame: A two-dimensional labeled data structure with columns of potentially different types

Creating DataFrames and Series

In [7]:
```python
#From Lists:

import pandas as pd

# Creating a Series from a list
series = pd.Series([1, 2, 3, 4, 5])

# Creating a DataFrame from a list of lists
data = [[1, 'Alice', 23], [2, 'Bob', 25], [3, 'Charlie', 22]]
df = pd.DataFrame(data, columns=['ID', 'Name', 'Age'])
print(df)
```

```
   ID     Name  Age
0   1    Alice   23
1   2      Bob   25
2   3  Charlie   22
```

In [8]:
```python
#From Dictionaries:

# Creating a DataFrame from a dictionary
data = {'ID': [1, 2, 3], 'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [23, 25, 22]}
df = pd.DataFrame(data)
print(df)
```

```
   ID     Name  Age
0   1    Alice   23
1   2      Bob   25
2   3  Charlie   22
```

In [11]:
```python
# Reading data from a CSV file
df = pd.read_csv('students.csv')
print(df.head())
```

```
      Name  Age Grade
0    Alice   20     A
1      Bob   22     B
2  Charlie   19     C
```

Common Operations

In [13]:
```python
#Selecting Data:

# Selecting a column
names = df['Name']

# Selecting multiple columns
subset = df[['Name', 'Age']]
print(names)
print(subset)
```

```
0      Alice
1        Bob
2    Charlie
Name: Name, dtype: object
      Name  Age
0    Alice   20
1      Bob   22
2  Charlie   19
```

In [15]:
```python
#Filtering Rows:

# Filtering rows based on a condition

filtered_df = df[df['Age'] > 23]
print(filtered_df)
```

```
Empty DataFrame
Columns: [Name, Age, Grade]
Index: []
```

In [17]:
```python
#Modifying Data:

# Adding a new column
df['Score'] = [85, 90, 88]

# Modifying existing data
df.loc[0, 'Age'] = 24
print(df)
```

```
      Name  Age Grade  Score
0    Alice   24     A     85
1      Bob   22     B     90
2  Charlie   19     C     88
```

Data Handling with Pandas

Here’s a Python program demonstrating data handling using Pandas:

In [33]:
```python
import pandas as pd

# Sample DataFrame
data = {
    'ord_no': [70001, None, 70002, 70004, None, 70005, None, 70010, 70003, 70012, None, 70013],
    'purch_amt': [150.5, None, 65.26, 110.5, 948.5, None, 5760.0, 1983.43, None, 250.45, 75.29, 3045.6],
    'sale_amt': [10.5, 20.65, None, 11.5, 98.5, None, 57.0, 19.43, None, 25.45, 75.29, 35.6],
    'ord_date': ['2012-10-05', '2012-09-10', None, '2012-08-17', '2012-09-10', '2012-07-27', '2012-09-10', '2012-10-10', '2012-10-10', '2012-06-27', '2012-08-17', '2012-04-25'],
    'customer_id': [3002, 3001, 3001, 3003, 3002, 3001, 3001, 3004, 3003, 3002, 3001, 3001],
    'salesman_id': [5002, 5003, 5001, None, 5002, 5001, 5001, None, 5003, 5002, 5003, None]
}

df = pd.DataFrame(data)

# Replace missing values with the most frequent values in each column
df = df.apply(lambda x: x.fillna(x.mode()[0]) if x.dtype == 'O' or x.dtype == 'float' else x)

print("DataFrame after replacing missing values with the most frequent values:")
print(df)
```

```
DataFrame after replacing missing values with the most frequent values:
     ord_no  purch_amt  sale_amt    ord_date  customer_id  salesman_id
0   70001.0     150.50     10.50  2012-10-05         3002       5002.0
1   70001.0      65.26     20.65  2012-09-10         3001       5003.0
2   70002.0      65.26     10.50  2012-09-10         3001       5001.0
3   70004.0     110.50     11.50  2012-08-17         3003       5001.0
4   70001.0     948.50     98.50  2012-09-10         3002       5002.0
5   70005.0      65.26     10.50  2012-07-27         3001       5001.0
6   70001.0    5760.00     57.00  2012-09-10         3001       5001.0
7   70010.0    1983.43     19.43  2012-10-10         3004       5001.0
8   70003.0      65.26     10.50  2012-10-10         3003       5003.0
9   70012.0     250.45     25.45  2012-06-27         3002       5002.0
10  70001.0      75.29     75.29  2012-08-17         3001       5003.0
11  70013.0    3045.60     35.60  2012-04-25         3001       5001.0
```

Data Analysis with Pandas

Using Pandas to perform data analysis:

In [37]:
```python
import pandas as pd

# Sample data
data = {'ID': [1, 2, 3, 4, 5],
        'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Edward'],
        'Age': [23, 25, 22, 24, 23],
        'Score': [85, 90, 88, 92, 85]}

df = pd.DataFrame(data)

# Check data types
print("Data Types:\n", df.dtypes)

# Generating summary statistics
summary = df.describe()
print("Summary Statistics:\n", summary)

# Inspect the DataFrame before grouping
print("\nDataFrame Before Grouping:\n", df)

# Grouping data and applying aggregate functions
try:
    grouped = df.groupby('Age').mean()
    print("\nGrouped Data:\n", grouped)
except TypeError as e:
    print("\nTypeError encountered:", e)

# Creating another DataFrame for merging
df2 = pd.DataFrame({'ID': [1, 2, 3], 'Subject': ['Math', 'Science', 'English']})

# Inspect the second DataFrame
print("\nSecond DataFrame:\n", df2)

# Merging DataFrames
try:
    merged_df = pd.merge(df, df2, on='ID', how='left')
    print("\nMerged DataFrame:\n", merged_df)
except Exception as e:
    print("\nError encountered during merging:", e)
```

```
Data Types:
 ID        int64
Name     object
Age       int64
Score     int64
dtype: object
Summary Statistics:
              ID        Age      Score
count  5.000000   5.000000   5.000000
mean   3.000000  23.400000  88.000000
std    1.581139   1.140175   3.082207
min    1.000000  22.000000  85.000000
25%    2.000000  23.000000  85.000000
50%    3.000000  23.000000  88.000000
75%    4.000000  24.000000  90.000000
max    5.000000  25.000000  92.000000

DataFrame Before Grouping:
    ID     Name  Age  Score
0   1    Alice   23     85
1   2      Bob   25     90
2   3  Charlie   22     88
3   4    David   24     92
4   5   Edward   23     85

TypeError encountered: agg function failed [how->mean,dtype->object]

Second DataFrame:
    ID  Subject
0   1     Math
1   2  Science
2   3  English

Merged DataFrame:
    ID     Name  Age  Score  Subject
0   1    Alice   23     85     Math
1   2      Bob   25     90  Science
2   3  Charlie   22     88  English
3   4    David   24     92      NaN
4   5   Edward   23     85      NaN
```

Application in Data Science:

Pandas is essential for data science professionals due to its powerful data manipulation and analysis capabilities:

Advantages: -Ease of Use: Pandas provides intuitive and flexible data structures. -Efficiency: Optimized for performance, handling large datasets efficiently. -Integration: Works seamlessly with other libraries like NumPy, Matplotlib, and Scikit-learn.

Real-World Examples: -Data Cleaning: Removing duplicates, handling missing values, and transforming data types. -Exploratory Data Analysis (EDA): Generating summary statistics, visualizing data, and identifying patterns. -Machine Learning: Preprocessing data, feature engineering, and model evaluation.

Summary: Pandas enhances the efficiency and effectiveness of data handling and analysis, making it a cornerstone for data science. Its ability to handle large datasets, perform complex operations, and integrate with other libraries makes it indispensable for data analysis and machine learning tasks.