ddd

August 31, 2024

0.0.1 1.Getting Familiar with Pandas

Introduction to Pandas :

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

- **Series:** A one-dimensional array-like object that can hold any data type (integers, strings, floats, etc.).
- **DataFrame:** A two-dimensional table where data is organized into rows and columns.

Creating DataFrames and Series :

Creating a Series: for creating a series we have to first import the pandas later we have to create a series using list and dictionaries.

[ ]:
```
import pandas as pd

# From a list
data = [10, 20, 30, 40]
series = pd.Series(data)
print(series)

# From a dictionary
data_dict = {'a': 1, 'b': 2, 'c': 3}
series_from_dict = pd.Series(data_dict)
print(series_from_dict)
```

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

Creating a DataFrame: for creating a dataframe here i used dictionary.

[ ]:
```
# From a dictionary
data = {
    'Name': ['kiran', 'Bobby', 'gayle'],
    'Age': [25, 30, 35],
    'City': ['india', 'Los Angeles', 'west indies']
}
df = pd.DataFrame(data)
print(df)
```

```
    Name  Age         City
0  kiran   25        india
1  Bobby   30  Los Angeles
2  gayle   35  west indies
```

0.0.2 2.Data Handling with Pandas:

here let us take an example and read a csv file and apply the data handling operations for the pandas.

[ ]:
```
import pandas as pd

# Reading data from a CSV file
df = pd.read_csv('/content/Toy-Sales-dataset - Training.csv')

# Display the first few rows
print("First few rows of the DataFrame:")
print(df.head())
```

```
First few rows of the DataFrame:
   Month  Sales  PromExp  Price  AdExp
0      1  73959    61.13   8.75  50.04
1      2  71544    60.19   8.99  50.74
2      3  78587    59.16   7.50  50.14
3      4  80364    60.38   7.25  50.27
4      5  78771    59.71   7.40  51.25
```

[28]:
```
# Handling missing data
# Fill missing values with a specified value
df.fillna(0, inplace=True)

# Remove duplicate rows
df.drop_duplicates(inplace=True)

# Display the cleaned DataFrame
print("Cleaned DataFrame:")
print(df)
```

```
Cleaned DataFrame:
```

```
  Category  Values
0        X       1
1        Y       2
2        X       3
3        Y       4
4        X       5
```

###3.Data Analysis with Pandas:

a. Generating Summary Statistics:

[ ]:
```python
import pandas as pd

# Example DataFrame
data = {'A': [1, 2, 3, 4, 5],
        'B': [10, 20, 30, 40, 50],
        'C': [5, 4, 3, 2, 1]}

df = pd.DataFrame(data)

# Summary statistics
summary_stats = df.describe()
print(summary_stats)
```

```
              A          B         C
count  5.000000   5.000000  5.000000
mean   3.000000  30.000000  3.000000
std    1.581139  15.811388  1.581139
min    1.000000  10.000000  1.000000
25%    2.000000  20.000000  2.000000
50%    3.000000  30.000000  3.000000
75%    4.000000  40.000000  4.000000
max    5.000000  50.000000  5.000000
```

describe() provides count, mean, std deviation, min, max, and percentiles.

b. Grouping Data and Applying Aggregate Functions:

[ ]:
```python
# Adding a categorical column
data = {'Category': ['X', 'Y', 'X', 'Y', 'X'],
        'Values': [1, 2, 3, 4, 5]}

df = pd.DataFrame(data)

# Grouping by 'Category' and calculating the mean
grouped_data = df.groupby('Category').mean()
print(grouped_data)
```

```
          Values
Category
```

```
X            3.0
Y            3.0
```

- groupby() groups the data.
- mean() calculates the average for each group.

c. Advanced Data Manipulation: Merging, Joining, and Concatenating DataFrames

[ ]:
```
# Creating two DataFrames to merge
df1 = pd.DataFrame({'Key': ['K0', 'K1', 'K2'],
                    'A': ['A0', 'A1', 'A2']})

df2 = pd.DataFrame({'Key': ['K0', 'K1', 'K3'],
                    'B': ['B0', 'B1', 'B3']})

# Merging on 'Key'
merged_df = pd.merge(df1, df2, on='Key', how='outer')
print(merged_df)
```

```
  Key    A    B
0  K0   A0   B0
1  K1   A1   B1
2  K2   A2  NaN
3  K3  NaN   B3
```

pd.merge() allows merging based on a common key. The how='outer' performs an outer join.

[ ]:
```
# Concatenating DataFrames
concat_df = pd.concat([df1, df2], axis=0, ignore_index=True)
print(concat_df)
```

```
  Key    A    B
0  K0   A0  NaN
1  K1   A1  NaN
2  K2   A2  NaN
3  K0  NaN   B0
4  K1  NaN   B1
5  K3  NaN   B3
```

pd.concat() stacks DataFrames either vertically (axis=0) or horizontally (axis=1).

0.0.3 4.Application in DataScience:

a. Explanation of Pandas in Data Science:

Efficiency: Pandas provides fast and efficient data manipulation capabilities that are crucial for data science tasks. It enables handling of large datasets that would be cumbersome with traditional Python lists or dictionaries.

Flexibility: With its ability to work with different types of data and perform complex operations (e.g., groupby, pivot, etc.), Pandas is indispensable for data wrangling and preprocessing, which

are key steps in data science.

b. Comparison with Traditional Python Data Structures:

Lists and Dictionaries: While lists and dictionaries are fundamental to Python, they are not designed for large-scale data analysis. Pandas DataFrames offer labeled indexing, powerful operations, and memory-efficient handling of large datasets.

c. Real-World Examples:

Data Cleaning: Pandas excels at handling missing data, filtering outliers, and transforming data, which are essential tasks before model training.

Exploratory Data Analysis (EDA): Pandas simplifies the process of exploring and visualizing data, allowing data scientists to quickly uncover patterns and insights.

Summary: "Pandas significantly enhances the efficiency of data analysis by providing powerful tools for data manipulation, such as groupby operations and merging/joining of datasets. These features allow data scientists to handle large and complex datasets with ease, making Pandas an indispensable tool in the field of data science."