pandas

September 1, 2024

1 getting familiarity with pandas

under standing two primary data structures in pandas

installing pandas

[2]:
```
pip install pandas
```

```
Defaulting to user installation because normal site-packages is not
writeableNote: you may need to restart the kernel to use updated packages.

Requirement already satisfied: pandas in c:\users\lokesh
naidu\appdata\roaming\python\python312\site-packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in c:\users\lokesh
naidu\appdata\roaming\python\python312\site-packages (from pandas) (2.0.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\lokesh
naidu\appdata\roaming\python\python312\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\lokesh
naidu\appdata\roaming\python\python312\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\lokesh
naidu\appdata\roaming\python\python312\site-packages (from pandas) (2024.1)
Requirement already satisfied: six>=1.5 in c:\users\lokesh
naidu\appdata\roaming\python\python312\site-packages (from python-
dateutil>=2.8.2->pandas) (1.16.0)


[notice] A new release of pip is available: 24.1.1 -> 24.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

importing pandas

[3]:
```
import pandas as pd
```

2 data series

creating data series

[4]:
```
#creating 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
```

[5]:
```
# Creating a Series from a dictionary
data = {'a': 100, 'b': 200, 'c': 300}
s2= pd.Series(data)
print(s2)
```

```
a    100
b    200
c    300
dtype: int64
```

accessing elements

[6]:
```
print(series[1])
```

```
20
```

giving indices

[7]:
```
series = pd.Series(data, index=['a', 'b', 'c', 'd'])
print(series)
```

```
a    100.0
b    200.0
c    300.0
d      NaN
dtype: float64
```

3 data frames

craeting a data frame

from dictionary

[8]:
```
data = {
    'City': ['Vizag', 'Hyd', 'Goa', 'Pune'],
    'Population': [2000000, 3000000, 1500000, 3500000],
    'Income': [50000, 70000, 45000, 75000]
}

df = pd.DataFrame(data)
print(df)
```

```
    City  Population  Income
0  Vizag     2000000   50000
1    Hyd     3000000   70000
2    Goa     1500000   45000
3   Pune     3500000   75000
```

from list of list

[9]:
```
data2 = [
    ['loki',20,'vizag'],['jayanth',29,'vzm'],['sasank',30,'goa']
]

df2 = pd.DataFrame(data, columns=['Name', 'Age', 'City'])
print(df2)
```

```
  Name  Age   City
0  NaN  NaN  Vizag
1  NaN  NaN    Hyd
2  NaN  NaN    Goa
3  NaN  NaN   Pune
```

from a csv file

[10]:
```
df3= pd.read_csv('student.csv')
print(df3)
```

```
     rollno  1year  2year  3year
0  aaabb121    9.5    9.2    8.5
1   aaabb18    8.7    9.9    8.6
2   aaabb29    9.2    9.5    9.7
```

accessing

columns

[11]:
```
# Accessing the 'City' column
print(df['City'])

# Accessing the 'Population' column
print(df['Population'])
```

```
0    Vizag
1      Hyd
2      Goa
3     Pune
Name: City, dtype: object
0    2000000
1    3000000
2    1500000
3    3500000
Name: Population, dtype: int64
```

rows

[12]:
```
# Accessing the first row using .loc
print(df.loc[0])

# Accessing the second row using .iloc
print(df.iloc[1])
```

```
City            Vizag
Population    2000000
Income          50000
Name: 0, dtype: object
City              Hyd
Population    3000000
Income          70000
Name: 1, dtype: object
```

3.1 basic operations

Adding a New Column

[13]:
```
df['Area'] = [550, 650, 400, 700]
print(df)
```

```
    City  Population  Income  Area
0  Vizag     2000000   50000   550
1    Hyd     3000000   70000   650
2    Goa     1500000   45000   400
3   Pune     3500000   75000   700
```

dropping a column

[14]:
```
df = df.drop('Area', axis=1)
print(df)
```

```
    City  Population  Income
0  Vizag     2000000   50000
1    Hyd     3000000   70000
2    Goa     1500000   45000
3   Pune     3500000   75000
```

filtering

[15]:
```
filtered_df = df[df['Population'] > 2000000]
print(filtered_df)
```

```
   City  Population  Income
1   Hyd     3000000   70000
3  Pune     3500000   75000
```

calculating new column

[16]:
```
df['Income per Capita'] = df['Income'] / df['Population']
print(df)
```

```
    City  Population  Income  Income per Capita
0  Vizag     2000000   50000           0.025000
1    Hyd     3000000   70000           0.023333
2    Goa     1500000   45000           0.030000
3   Pune     3500000   75000           0.021429
```

modifying

[17]:
```
# Example: Changing the population of 'Vizag'
df.loc[df['City'] == 'Vizag', 'Population'] = 2500000
print(df)
```

```
    City  Population  Income  Income per Capita
0  Vizag     2500000   50000           0.025000
1    Hyd     3000000   70000           0.023333
2    Goa     1500000   45000           0.030000
3   Pune     3500000   75000           0.021429
```

[18]:
```
# Example: Modifying the entire 'Population' column
df['Population'] = [2500000, 3200000, 1600000, 3700000]
print(df)
```

```
    City  Population  Income  Income per Capita
0  Vizag     2500000   50000           0.025000
1    Hyd     3200000   70000           0.023333
2    Goa     1600000   45000           0.030000
3   Pune     3700000   75000           0.021429
```

4 data handling

reading the csv file into a data frame

[22]:
```
fr=pd.read_csv('student_info_20_records.csv')
print(fr)
```

```
       Name   Age  CGPA     Fee
0   Krishna   NaN   3.7  1800.0
1   Lakshay  22.0   3.8  2000.0
2     Kabir  21.0   3.7  1700.0
3     Aditi  22.0   NaN  1500.0
4     Kavya  21.0   3.7     NaN
5     Aarav  23.0   3.5     NaN
6    Ananya  20.0   NaN  1500.0
7    Vikram  23.0   NaN  1500.0
8    Ishita  23.0   3.5  2000.0
9      Tara  23.0   3.9     NaN
10     Riya  22.0   3.5     NaN
```

```
11    Pooja  23.0   3.6     NaN
12    Rohan  23.0   3.8  1500.0
13   Manish   NaN   3.7  1700.0
14    Rahul  21.0   3.5     NaN
15     Maya  21.0   3.5     NaN
16    Nisha  22.0   3.8  1700.0
17    Aryan  20.0   3.5  1500.0
18    Arjun  23.0   3.9  2000.0
19    Sanya  21.0   3.5  1600.0
```

checking missing values in each column

[23]:
```
mv=fr.isnull().sum()
print(mv)
```

```
Name    0
Age     2
CGPA    3
Fee     7
dtype: int64
```

filling missing values w.r.t their mean

[33]:
```
fr['Age']=fr['Age'].fillna(fr['Age'].mean())
print(fr)
```

```
      Name        Age  CGPA     Fee Age_Group
0   Krishna  21.888889   3.7  1800.0     Adult
1   Lakshay  22.000000   3.8  2000.0     Adult
2     Kabir  21.000000   3.7  1700.0     Young
3     Aditi  22.000000   3.5  1500.0     Adult
4     Kavya  21.000000   3.7     NaN     Young
5     Aarav  23.000000   3.5     NaN     Adult
6    Ananya  20.000000   3.5  1500.0     Young
7    Vikram  23.000000   3.5  1500.0     Adult
8    Ishita  23.000000   3.5  2000.0     Adult
9      Tara  23.000000   3.9     NaN     Adult
10     Riya  22.000000   3.5     NaN     Adult
11    Pooja  23.000000   3.6     NaN     Adult
12    Rohan  23.000000   3.8  1500.0     Adult
13   Manish  21.888889   3.7  1700.0     Adult
14    Rahul  21.000000   3.5     NaN     Young
15     Maya  21.000000   3.5     NaN     Young
16    Nisha  22.000000   3.8  1700.0     Adult
17    Aryan  20.000000   3.5  1500.0     Young
18    Arjun  23.000000   3.9  2000.0     Adult
19    Sanya  21.000000   3.5  1600.0     Young
```

converting the 'Age' column to integer type explicitly

[40]:
```
fr['Age'] = fr['Age'].astype(int)
print(fr['Age'])
```

```
0     21
1     22
2     21
3     22
4     21
5     23
6     20
7     23
8     23
9     23
10    22
11    23
12    23
13    21
14    21
15    21
16    22
17    20
18    23
19    21
Name: Age, dtype: int64
```

filling missing values w.r.t mode

[41]:
```
fr['CGPA']=fr['CGPA'].fillna(fr['CGPA'].mode()[0])
print(fr)
```

```
       Name  Age  CGPA     Fee Age_Group
0   Krishna   21   3.7  1800.0     Adult
1   Lakshay   22   3.8  2000.0     Adult
2     Kabir   21   3.7  1700.0     Young
3     Aditi   22   3.5  1500.0     Adult
4     Kavya   21   3.7     NaN     Young
5     Aarav   23   3.5     NaN    Senior
6    Ananya   20   3.5  1500.0     Young
7    Vikram   23   3.5  1500.0    Senior
8    Ishita   23   3.5  2000.0    Senior
9      Tara   23   3.9     NaN    Senior
10     Riya   22   3.5     NaN     Adult
11    Pooja   23   3.6     NaN    Senior
12    Rohan   23   3.8  1500.0    Senior
13   Manish   21   3.7  1700.0     Adult
14    Rahul   21   3.5     NaN     Young
15     Maya   21   3.5     NaN     Young
16    Nisha   22   3.8  1700.0     Adult
17    Aryan   20   3.5  1500.0     Young
```

```
18    Arjun   23   3.9  2000.0    Senior
19    Sanya   21   3.5  1600.0     Young
```

creating a new column based on age categorie

[42]:
```
# Binning 'Age' into categories
bins = [0, 21, 22, 24]
labels = ['Young', 'Adult', 'Senior']
fr['Age_Group'] = pd.cut(fr['Age'], bins=bins, labels=labels)
print(fr)
```

```
       Name  Age  CGPA     Fee Age_Group
0   Krishna   21   3.7  1800.0     Young
1   Lakshay   22   3.8  2000.0     Adult
2     Kabir   21   3.7  1700.0     Young
3     Aditi   22   3.5  1500.0     Adult
4     Kavya   21   3.7     NaN     Young
5     Aarav   23   3.5     NaN    Senior
6    Ananya   20   3.5  1500.0     Young
7    Vikram   23   3.5  1500.0    Senior
8    Ishita   23   3.5  2000.0    Senior
9      Tara   23   3.9     NaN    Senior
10     Riya   22   3.5     NaN     Adult
11    Pooja   23   3.6     NaN    Senior
12    Rohan   23   3.8  1500.0    Senior
13   Manish   21   3.7  1700.0     Young
14    Rahul   21   3.5     NaN     Young
15     Maya   21   3.5     NaN     Young
16    Nisha   22   3.8  1700.0     Adult
17    Aryan   20   3.5  1500.0     Young
18    Arjun   23   3.9  2000.0    Senior
19    Sanya   21   3.5  1600.0     Young
```

5 data analysis

generating summary statistics using describe method:

[43]:
```
summary_stats=fr.describe()
print(summary_stats)
```

```
             Age       CGPA          Fee
count  20.000000  20.000000    13.000000
mean   21.800000   3.630000  1692.307692
std     1.056309   0.149032   201.913919
min    20.000000   3.500000  1500.000000
25%    21.000000   3.500000  1500.000000
50%    22.000000   3.550000  1700.000000
75%    23.000000   3.725000  1800.000000
max    23.000000   3.900000  2000.000000
```

Grouping data by 'Age' and calculating the mean CGPA and Fee for each age group

[44]:
```
grouped = fr.groupby('Age')[['CGPA', 'Fee']].mean()

print("Grouped Data (Mean CGPA and Fee by Age):")
print(grouped)
```

```
Grouped Data (Mean CGPA and Fee by Age):
         CGPA          Fee
Age
20   3.500000  1500.000000
21   3.614286  1700.000000
22   3.650000  1733.333333
23   3.671429  1750.000000
```

Applying Aggregate Functions

Grouping data by 'Age' and applying multiple aggregate functions

[57]:
```
aggregated = fr.groupby('Age').agg({
    'CGPA': ['mean', 'min', 'max'],
    'Fee': ['sum', 'mean']
})

print("Aggregated Data (Various Aggregations by Age):")
print(aggregated)
```

```
Aggregated Data (Various Aggregations by Age):
         CGPA               Fee
         mean  min  max     sum         mean
Age
20   3.500000  3.5  3.5  3000.0  1500.000000
21   3.614286  3.5  3.7  6800.0  1700.000000
22   3.650000  3.5  3.8  5200.0  1733.333333
23   3.671429  3.5  3.9  7000.0  1750.000000
```

5.1 merging

Combines DataFrames based on common columns or indices. Using pd.merge().

[47]:
```
df1 = pd.DataFrame({'key': ['A', 'B', 'C', 'D'],
                    'value': [1, 2, 3, 4]})
df2 = pd.DataFrame({'key': ['B', 'D', 'E', 'F'],
                    'value': [100, 200, 300, 400]})

result = pd.merge(df1, df2, on='key', how='inner')
print(result)
```

```
  key  value_x  value_y
0   B        2      100
1   D        4      200
```

5.2 joining

Combines DataFrames based on indices. Using df.join().

[49]:
```
df1 = pd.DataFrame({'value1': [1, 2, 3]}, index=['A', 'B', 'C'])
df2 = pd.DataFrame({'value2': [4, 5]}, index=['B', 'C'])

result = df1.join(df2)
print(result)
```

```
   value1  value2
A       1     NaN
B       2     4.0
C       3     5.0
```

5.3 concatenating

combine DataFrames along a particular axis (rows or columns).

[56]:
```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                    'B': ['B0', 'B1', 'B2']})
df2 = pd.DataFrame({'A': ['A3', 'A4', 'A5'],
                    'B': ['B3', 'B4', 'B5']})
print(f"dataframe 1:\n{df1}")
print(f"dataframe 2:\n{df2}")
result1 = pd.concat([df1, df2], axis=0)  # Concatenate along rows
result2 = pd.concat([df1, df2], axis=1)  # Concatenate along cloumns
print(f"concatinated along rows:\n{result1}")
print(f"concatinated along cloumns:\n{result2}")
```

```
dataframe 1:
    A   B
0  A0  B0
1  A1  B1
2  A2  B2
dataframe 2:
    A   B
0  A3  B3
1  A4  B4
2  A5  B5
concatinated along rows:
    A   B
0  A0  B0
1  A1  B1
2  A2  B2
0  A3  B3
1  A4  B4
2  A5  B5
concatinated along cloumns:
```

```
    A   B   A   B
0  A0  B0  A3  B3
1  A1  B1  A4  B4
2  A2  B2  A5  B5
```

6 Application in data science

use of pandas in my program

- here i used pandas DataFrames (like spreadsheets) and Series (like lists) to handle and analyze data quickly.
- Operations like data handling and analysis are performed very effeciently and effectively
- Built-in methods make me easy to manage and clean missing data
- by using aggregating functions like mean,median and mode one colud able to easily analyse the data
- we can easily combine data from different sources using merging and joining features and Adding rows or columns from different DataFrames is simple with Pandas concatenating.

advantages of using pandas

- The basic structures like lists and dictionaries don't have the advanced features for data analysis that Pandas provides. Pandas is faster and more efficient for large datasets.
- Pandas is built on NumPy, which makes it faster for numerical operations.
- Pandas simplifies data work, is faster, and offers more features than traditional Python data structures, making it essential for data science tasks

real world examples

data cleaning

- Fill missing values with df.fillna()
- remove them with df.dropna().
- Remove duplicates using df.drop_duplicates().

EDA

- Get summary statistics with df.describe().
- Analysing cgpa and fee by age using fr.groupby('Age')[['CGPA', 'Fee']].mean()