Pandas

October 28, 2025

Practical Pandas guide

Dr.Labeed Al-Saad

What is Pandas

A powerful Python library for data manipulation and analysis. Think "Excel on steroids" for programmers.

Scope of usage

1. Data cleaning - handle missing values, fix data types.
2. Data transformation - filter, sort, reshape data.
3. Data analysis - statistics, aggregations, trends.
4. Data visualization prep - get data ready for charts/graphs.
5. Import/Export - read/write CSV, Excel, JSON, databases

Why Pandas is important?

1. Handles real-world data - messy, incomplete data becomes usable.
2. Fast & efficient - optimized for large datasets (thousands/millions of rows).
3. Replaces spreadsheets - reproducible, automated analysis.
4. Essential for data science - foundation for ML, analytics, reporting.
5. Saves time - complex operations in few lines of code.

Conclusion: If you work with data in Python, you need pandas. It's the standard tool for turning raw data into insights.

1. Core Data Structures

DataFrame: a 2D table (like Excel spreadsheet)

[1]:
```
import pandas as pd
import numpy as np # optional

# Create DataFrame from dictionary
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'Diana'],
    'Age': [25, 30, 35, 28],
    'Salary': [50000, 60000, 70000, 55000],
    'Department': ['HR', 'Tech', 'Tech', 'Marketing']
}
df = pd.DataFrame(data)
```

```
print(df)
```

```
      Name  Age  Salary Department
0    Alice   25   50000         HR
1      Bob   30   60000       Tech
2  Charlie   35   70000       Tech
3    Diana   28   55000  Marketing
```

Why was NumPy imported together with Pandas in this example?

While you can use pandas without NumPy for basic tasks, in real-world data analysis you'll almost always need both for: 1. Advanced math operations. 2. Handling missing data. 3. Creating sample datasets. 4. Performance optimization

**Series*: a 1D column (single data column)

[4]:
```
# Continuing the above example
# Extracting a single column
ages = df["Age"]
print(ages)

# if I need more than one column
columns = df[["Name", "Age"]]
print("\n", columns)
```

```
0    25
1    30
2    35
3    28
Name: Age, dtype: int64

      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   35
3    Diana   28
```

2. Essential Operations

Reading Data (Most Common)

[5]:
```
# Read from CSV file from my computer
import pandas as pd
df = pd.read_csv("E:/Labeed data_New/lectures/python/Data sets/
 ↪case_control_dataset.csv")
print(df)
```

```
   id        age  smoking  lung_cancer
0   1  35.518347        1            1
1   2  32.909398        0            1
2   3  64.634076        0            1
```

```
3   4  66.847592        1            1
4   5  51.465840        1            1
5   6  51.580472        1            1
6   7  35.274396        1            1
7   8  44.034536        0            1
8   9  44.190124        0            1
9  10  53.020766        1            1
```

[6]:
```python
# read Excel file from my computer
import pandas as pd
df = pd.read_excel("E:/Labeed data_New/lectures/python/Data sets/
 ↪case_control_dataset.xlsx")
print(df)
```

```
   id        age  smoking  lung_cancer
0   1  35.518347        1            1
1   2  32.909398        0            1
2   3  64.634076        0            1
3   4  66.847592        1            1
4   5  51.465840        1            1
5   6  51.580472        1            1
6   7  35.274396        1            1
7   8  44.034536        0            1
8   9  44.190124        0            1
9  10  53.020766        1            1
```

[21]:
```python
# Basic inspections

# Showing head of table (first 5 rows)
print("Show table head:\n", df.head())

# Showing last 3 rows
print("\n Show last 3 rowsd:\n", df.tail(3))

# Data types and missing values
print("\n Show missing values:\n", df.info())

# Statistical summary
print("\n Show statistical summary:\n", df.describe())
print("\n Show statistical summary of age only:\n", df["age"].describe()) #␣
 ↪Show statistical summary for 'age'
                                                                          #␣
 ↪field only
print("\n Show statistical summary of age only:\n", df.age.describe()) # other␣
 ↪why

# Shape (rows, columns)
```

```
print("\n Show df shape:\n", df.shape) # 'shape' not need parentheses as it is␣
↪an attribute, not a method
```

```
Show table head:
    id        age  smoking  lung_cancer
0   1  35.518347        1            1
1   2  32.909398        0            1
2   3  64.634076        0            1
3   4  66.847592        1            1
4   5  51.465840        1            1

 Show last 3 rowsd:
    id        age  smoking  lung_cancer
7   8  44.034536        0            1
8   9  44.190124        0            1
9  10  53.020766        1            1
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   id           10 non-null     int64
 1   age          10 non-null     float64
 2   smoking      10 non-null     int64
 3   lung_cancer  10 non-null     int64
dtypes: float64(1), int64(3)
memory usage: 452.0 bytes

 Show missing values:
 None

 Show statistical summary:
              id        age    smoking  lung_cancer
count  10.00000  10.000000  10.000000         10.0
mean    5.50000  47.947555   0.600000          1.0
std     3.02765  11.821053   0.516398          0.0
min     1.00000  32.909398   0.000000          1.0
25%     3.25000  37.647394   0.000000          1.0
50%     5.50000  47.827982   1.000000          1.0
75%     7.75000  52.660692   1.000000          1.0
max    10.00000  66.847592   1.000000          1.0

 Show statistical summary of age only:
 count    10.000000
mean     47.947555
std      11.821053
min      32.909398
25%      37.647394
```

```
50%      47.827982
75%      52.660692
max      66.847592
Name: age, dtype: float64

 Show statistical summary of age only:
 count    10.000000
mean     47.947555
std      11.821053
min      32.909398
25%      37.647394
50%      47.827982
75%      52.660692
max      66.847592
Name: age, dtype: float64

 Show df shape:
 (10, 4)
```

Selecting data

[31]:
```
# Selecting a single column
import pandas as np
df= np.read_excel("E:/Labeed data_New/lectures/python/Data sets/
  ↪case_control_dataset.xlsx")
print(df)
print("\n Age: \n", df["age"])
# OR
print("\n Age: \n", df.age)
```

```
   id        age  smoking  lung_cancer
0   1  35.518347        1            1
1   2  32.909398        0            1
2   3  64.634076        0            1
3   4  66.847592        1            1
4   5  51.465840        1            1
5   6  51.580472        1            1
6   7  35.274396        1            1
7   8  44.034536        0            1
8   9  44.190124        0            1
9  10  53.020766        1            1

 Age:
 0    35.518347
1    32.909398
2    64.634076
3    66.847592
4    51.465840
```

```
5    51.580472
6    35.274396
7    44.034536
8    44.190124
9    53.020766
Name: age, dtype: float64

 Age:
 0    35.518347
1    32.909398
2    64.634076
3    66.847592
4    51.465840
5    51.580472
6    35.274396
7    44.034536
8    44.190124
9    53.020766
Name: age, dtype: float64
```

[32]:
```
# Selecting multiple ccolumns

print(df[["id", "age"]])
```

```
   id        age
0   1  35.518347
1   2  32.909398
2   3  64.634076
3   4  66.847592
4   5  51.465840
5   6  51.580472
6   7  35.274396
7   8  44.034536
8   9  44.190124
9  10  53.020766
```

[36]:
```
# Selecting rows by indexabs
print(df.iloc[0]) #first row
print("\n", df.iloc[0:3]) # First three rows
```

```
id              1.000000
age            35.518347
smoking         1.000000
lung_cancer     1.000000
Name: 0, dtype: float64

    id        age  smoking  lung_cancer
0   1  35.518347        1            1
1   2  32.909398        0            1
```

```
2   3  64.634076        0            1
```

[43]:
```
# Selecting rows by condition

print("Single condition selection:\n", df[df["age"] > 30]) # showing rows when␣
 ↪age > 30
print("\n Multiple condition selection:\n", df[(df["age"] > 25) &␣
 ↪(df["smoking"] == 1)])  # Multiple conditions
```

```
Single condition selection:
    id        age  smoking  lung_cancer
0   1  35.518347        1            1
1   2  32.909398        0            1
2   3  64.634076        0            1
3   4  66.847592        1            1
4   5  51.465840        1            1
5   6  51.580472        1            1
6   7  35.274396        1            1
7   8  44.034536        0            1
8   9  44.190124        0            1
9  10  53.020766        1            1

 Multiple condition selection:
    id        age  smoking  lung_cancer
0   1  35.518347        1            1
3   4  66.847592        1            1
4   5  51.465840        1            1
5   6  51.580472        1            1
6   7  35.274396        1            1
9  10  53.020766        1            1
```

3.Data Cleaning (Most Common Tasks)

Handling Missing Values

Filtering data

[55]:
```
import pandas as pd
df= np.read_excel("E:/Labeed data_New/lectures/python/Data sets/Data.xlsx")
print(df,"\n")
print("\n Statistical summary:\n", df.describe()) # Describe the data set␣
 ↪statistically

# Checking for missing values
print("\n The sum of missing values:\n", df.isnull().sum())
```

```
    patient_id      Name    age  lung_cancer  smoker     city
0            1       Ali  22.00            0       0   Basrah
1            2  Mohammed  30.00            1       1  Baghdad
2            3     Abass  27.00            0       0   Basrah
```

```
3            4    Hassan  40.00            0       1    Mysan
4            5     Hyder  48.15            1       1    Wasit
5            6    Fatima    NaN            0       0   Thiqar
6            7    Zainab  26.00            0       0    Mysan
7            8       NaN  56.00            0       1  Baghdad
8            9    Labeed  51.50            0       0   Basrah
9           10     Qusai    NaN            0       0   Basrah
10          11     Abass  27.00            0       0   Basrah
11          12    Hassan  44.00            1       1    Mysan


 Statistical summary:
        patient_id        age  lung_cancer     smoker
count   12.000000  10.000000    12.000000  12.000000
mean     6.500000  37.165000     0.250000   0.416667
std      3.605551  12.239736     0.452267   0.514929
min      1.000000  22.000000     0.000000   0.000000
25%      3.750000  27.000000     0.000000   0.000000
50%      6.500000  35.000000     0.000000   0.000000
75%      9.250000  47.112500     0.250000   1.000000
max     12.000000  56.000000     1.000000   1.000000

 The sum of missing values:
 patient_id     0
Name            1
age             2
lung_cancer     0
smoker          0
city            0
dtype: int64
```

[56]:
```
# Drop rows with any missing values
df_clean = df.dropna()
print("Cleaned data frame:\n", df_clean)
```

```
Cleaned data frame:
     patient_id      Name    age  lung_cancer  smoker     city
0            1       Ali  22.00            0       0   Basrah
1            2  Mohammed  30.00            1       1  Baghdad
2            3     Abass  27.00            0       0   Basrah
3            4    Hassan  40.00            0       1    Mysan
4            5     Hyder  48.15            1       1    Wasit
6            7    Zainab  26.00            0       0    Mysan
8            9    Labeed  51.50            0       0   Basrah
10          11     Abass  27.00            0       0   Basrah
11          12    Hassan  44.00            1       1    Mysan
```

[71]:
```
# Fill missing values
df= pd.read_excel("E:/Labeed data_New/lectures/python/Data sets/Data.xlsx")
df_fill = df
print("\n df_fill Befor processing:\n", df_fill)

df_fill =  df_fill.fillna({"Name": "Unknown"}) # Fill with text

df_fill = df_fill.fillna({"age": df_fill.age.mean()}) # Fill with mean

print("\n df_fill After processing:\n", df_fill)
```

```
 df_fill Befor processing:
    patient_id      Name    age  lung_cancer  smoker     city
0            1       Ali  22.00            0       0   Basrah
1            2  Mohammed  30.00            1       1  Baghdad
2            3     Abass  27.00            0       0   Basrah
3            4    Hassan  40.00            0       1    Mysan
4            5     Hyder  48.15            1       1    Wasit
5            6    Fatima    NaN            0       0   Thiqar
6            7    Zainab  26.00            0       0    Mysan
7            8       NaN  56.00            0       1  Baghdad
8            9    Labeed  51.50            0       0   Basrah
9           10     Qusai    NaN            0       0   Basrah
10          11     Abass  27.00            0       0   Basrah
11          12    Hassan  44.00            1       1    Mysan

 df_fill After processing:
    patient_id      Name     age  lung_cancer  smoker     city
0            1       Ali  22.000            0       0   Basrah
1            2  Mohammed  30.000            1       1  Baghdad
2            3     Abass  27.000            0       0   Basrah
3            4    Hassan  40.000            0       1    Mysan
4            5     Hyder  48.150            1       1    Wasit
5            6    Fatima  37.165            0       0   Thiqar
6            7    Zainab  26.000            0       0    Mysan
7            8   Unknown  56.000            0       1  Baghdad
8            9    Labeed  51.500            0       0   Basrah
9           10     Qusai  37.165            0       0   Basrah
10          11     Abass  27.000            0       0   Basrah
11          12    Hassan  44.000            1       1    Mysan
```

Removing Duplicates

[89]:
```
# Removing duplicate rows
df_nodup = df_fill.drop_duplicates()
print("\n Removing duplicate rows:\n", df_nodup) # duplicate rows in our data
```

```
# Remove duplicates based on specific columns
df_nodups = df_fill.drop_duplicates(subset=["Name"])
print("\n Removing duplicate rows based on name:\n", df_nodups)
```

```
 Removing duplicate rows:
    patient_id      Name     age  lung_cancer  smoker     city
0            1       Ali  22.000            0       0   Basrah
1            2  Mohammed  30.000            1       1  Baghdad
2            3     Abass  27.000            0       0   Basrah
3            4    Hassan  40.000            0       1    Mysan
4            5     Hyder  48.150            1       1    Wasit
5            6    Fatima  37.165            0       0   Thiqar
6            7    Zainab  26.000            0       0    Mysan
7            8   Unknown  56.000            0       1  Baghdad
8            9    Labeed  51.500            0       0   Basrah
9           10     Qusai  37.165            0       0   Basrah
10          11     Abass  27.000            0       0   Basrah
11          12    Hassan  44.000            1       1    Mysan

 Removing duplicate rows based on name:
    patient_id      Name     age  lung_cancer  smoker     city
0            1       Ali  22.000            0       0   Basrah
1            2  Mohammed  30.000            1       1  Baghdad
2            3     Abass  27.000            0       0   Basrah
3            4    Hassan  40.000            0       1    Mysan
4            5     Hyder  48.150            1       1    Wasit
5            6    Fatima  37.165            0       0   Thiqar
6            7    Zainab  26.000            0       0    Mysan
7            8   Unknown  56.000            0       1  Baghdad
8            9    Labeed  51.500            0       0   Basrah
9           10     Qusai  37.165            0       0   Basrah
```

Data Type Conversion

[11]:
```
# Convert to a different type of data
import pandas as pd
import numpy as np
df= pd.read_excel("E:/Labeed data_New/lectures/python/Data sets/Data.xlsx")
df_fill = df.copy()  # Use copy() to avoid modifying the original dataframe
print("\n Data frame before filling NaN values:\n",df_fill)

# Make sure all NaN values are filled before converting to int
# Then convert to int using numpy's int function which handles NaN values better
df_fill["age"] = df_fill["age"].fillna(df_fill.age.mean())  # Ensure no NaNs␣
 ↪remain
df_fill["age"] = df_fill["age"].astype(np.int64)  # Convert to integer using␣
 ↪numpy
```

```
df_converted = df_fill
print("\n Age converted to Integer:\n", df_converted)
```

```
 Data frame before filling NaN values:
    patient_id      Name    age  lung_cancer  smoker     city
0            1       Ali  22.00            0       0   Basrah
1            2  Mohammed  30.00            1       1  Baghdad
2            3     Abass  27.00            0       0   Basrah
3            4    Hassan  40.00            0       1    Mysan
4            5     Hyder  48.15            1       1    Wasit
5            6    Fatima    NaN            0       0   Thiqar
6            7    Zainab  26.00            0       0    Mysan
7            8       NaN  56.00            0       1  Baghdad
8            9    Labeed  51.50            0       0   Basrah
9           10     Qusai    NaN            0       0   Basrah
10          11     Abass  27.00            0       0   Basrah
11          12    Hassan  44.00            1       1    Mysan

 Age converted to Integer:
    patient_id      Name  age  lung_cancer  smoker     city
0            1       Ali   22            0       0   Basrah
1            2  Mohammed   30            1       1  Baghdad
2            3     Abass   27            0       0   Basrah
3            4    Hassan   40            0       1    Mysan
4            5     Hyder   48            1       1    Wasit
5            6    Fatima   37            0       0   Thiqar
6            7    Zainab   26            0       0    Mysan
7            8       NaN   56            0       1  Baghdad
8            9    Labeed   51            0       0   Basrah
9           10     Qusai   37            0       0   Basrah
10          11     Abass   27            0       0   Basrah
11          12    Hassan   44            1       1    Mysan
```

[21]:
```
# Filter with query method
import pandas as pd
df= pd.read_excel("E:/Labeed data_New/lectures/python/Data sets/Data.xlsx")
print("\n Original data frame:\n", df,"\n")
print("\n Statistical summary:\n", df.describe()) # Describe the data set␣
 ↪statistically
df_filtered = df.query("age >25 and age.notnull() and smoker == 0") # Filtering␣
 ↪by query
print("\n Filtered data:\n", df_filtered)
```

```
 Original data frame:
    patient_id      Name    age  lung_cancer  smoker     city
```

```
0            1       Ali  22.00            0       0   Basrah
1            2  Mohammed  30.00            1       1  Baghdad
2            3     Abass  27.00            0       0   Basrah
3            4    Hassan  40.00            0       1    Mysan
4            5     Hyder  48.15            1       1    Wasit
5            6    Fatima    NaN            0       0   Thiqar
6            7    Zainab  26.00            0       0    Mysan
7            8       NaN  56.00            0       1  Baghdad
8            9    Labeed  51.50            0       0   Basrah
9           10     Qusai    NaN            0       0   Basrah
10          11     Abass  27.00            0       0   Basrah
11          12    Hassan  44.00            1       1    Mysan


 Statistical summary:
        patient_id        age  lung_cancer     smoker
count   12.000000  10.000000    12.000000  12.000000
mean     6.500000  37.165000     0.250000   0.416667
std      3.605551  12.239736     0.452267   0.514929
min      1.000000  22.000000     0.000000   0.000000
25%      3.750000  27.000000     0.000000   0.000000
50%      6.500000  35.000000     0.000000   0.000000
75%      9.250000  47.112500     0.250000   1.000000
max     12.000000  56.000000     1.000000   1.000000

 Filtered data:
    patient_id    Name   age  lung_cancer  smoker    city
2            3   Abass  27.0            0       0  Basrah
6            7  Zainab  26.0            0       0   Mysan
8            9  Labeed  51.5            0       0  Basrah
10          11   Abass  27.0            0       0  Basrah
```

[24]:
```
# Filtering by string contains
import pandas as pd
df= pd.read_excel("E:/Labeed data_New/lectures/python/Data sets/Data.xlsx")
df_filtered = df.copy()
print("\n Original data frame:\n", df_filtered,"\n")
print("Rows contains Ali: \n", df_filtered[df_filtered["Name"].str.
 ↪contains("Abass", na = False)])
# Add na=False parameter to handle NaN values in the Name column
```

```
 Original data frame:
    patient_id      Name    age  lung_cancer  smoker     city
0            1       Ali  22.00            0       0   Basrah
1            2  Mohammed  30.00            1       1  Baghdad
2            3     Abass  27.00            0       0   Basrah
3            4    Hassan  40.00            0       1    Mysan
```

```
4            5     Hyder  48.15            1       1    Wasit
5            6    Fatima    NaN            0       0   Thiqar
6            7    Zainab  26.00            0       0    Mysan
7            8       NaN  56.00            0       1  Baghdad
8            9    Labeed  51.50            0       0   Basrah
9           10     Qusai    NaN            0       0   Basrah
10          11     Abass  27.00            0       0   Basrah
11          12    Hassan  44.00            1       1    Mysan

Rows contains Ali:
     patient_id   Name   age  lung_cancer  smoker    city
2            3  Abass  27.0            0       0  Basrah
10          11  Abass  27.0            0       0  Basrah
```

4. Data Transformation

Adding/Modifying Columns

[27]:
```python
# Add new column
import pandas as pd
df = pd.read_excel("E:/Labeed data_New/lectures/python/Data sets/Employee.xlsx")
df_new = df.copy()
print("\n Original data frame:\n", df_new,"\n")

# Add Bonus field (column)
df_new["Bonus"] = df_new["Salary"] * 0.2
print("\n Updated data frame:\n", df_new,"\n")
```

```
 Original data frame:
        Name  Age Department  Salary
0       Ali   22        IMS    1000
1  Mohammed   30        IMS    1000
2     Abass   27         CS     750
3    Hassan   40         IT     500
4     Hyder   48         IT     500
5    Fatima   30         CS     750
6    Zainab   26        CyS    1000
7      Alaa   25        IMS    1000
8    Labeed   50        IMS    1000
9     Qusai   23        IMS    1000


 Updated data frame:
        Name  Age Department  Salary  Bonus
0       Ali   22        IMS    1000  200.0
1  Mohammed   30        IMS    1000  200.0
2     Abass   27         CS     750  150.0
3    Hassan   40         IT     500  100.0
```

```
4     Hyder   48         IT     500  100.0
5    Fatima   30         CS     750  150.0
6    Zainab   26        CyS    1000  200.0
7      Alaa   25        IMS    1000  200.0
8    Labeed   50        IMS    1000  200.0
9     Qusai   23        IMS    1000  200.0
```

[28]:
```
# Modifying existing column
df_new["Salary"] = df_new["Salary"] + df_new["Bonus"]
print("\n Updated data frame:\n", df_new,"\n")
```

```
 Updated data frame:
        Name  Age Department  Salary  Bonus
0       Ali   22        IMS  1200.0  200.0
1  Mohammed   30        IMS  1200.0  200.0
2     Abass   27         CS   900.0  150.0
3    Hassan   40         IT   600.0  100.0
4     Hyder   48         IT   600.0  100.0
5    Fatima   30         CS   900.0  150.0
6    Zainab   26        CyS  1200.0  200.0
7      Alaa   25        IMS  1200.0  200.0
8    Labeed   50        IMS  1200.0  200.0
9     Qusai   23        IMS  1200.0  200.0
```

[30]:
```
# Conditional column "assigning the senior employees"

df_new["Senior"] = df_new["Age"].apply(lambda x: "Yes" if x > 30 else "No")
print("\n Updated data frame:\n", df_new,"\n")
```

```
 Updated data frame:
        Name  Age Department  Salary  Bonus Senior
0       Ali   22        IMS  1200.0  200.0     No
1  Mohammed   30        IMS  1200.0  200.0     No
2     Abass   27         CS   900.0  150.0     No
3    Hassan   40         IT   600.0  100.0    Yes
4     Hyder   48         IT   600.0  100.0    Yes
5    Fatima   30         CS   900.0  150.0     No
6    Zainab   26        CyS  1200.0  200.0     No
7      Alaa   25        IMS  1200.0  200.0     No
8    Labeed   50        IMS  1200.0  200.0    Yes
9     Qusai   23        IMS  1200.0  200.0     No
```

Grouping and Aggregation

[33]:
```
import pandas as pd
df = pd.read_excel("E:/Labeed data_New/lectures/python/Data sets/Employee.xlsx")
df_new = df.copy()
print("\n Original data frame:\n", df_new,"\n")

# Grouping by "Department" and calculate status
dept_status = df_new.groupby("Department").agg({
    "Salary": ["mean", "min", "max", "count"],
    "Age": "mean"
})
print("\n dept_status:\n", dept_status)
```

```
 Original data frame:
        Name  Age Department  Salary
0       Ali   22        IMS    1000
1  Mohammed   30        IMS    1000
2     Abass   27         CS     750
3    Hassan   40         IT     500
4     Hyder   48         IT     500
5    Fatima   30         CS     750
6    Zainab   26        CyS    1000
7      Alaa   25        IMS    1000
8    Labeed   50        IMS    1000
9     Qusai   23        IMS    1000


 dept_status:
             Salary                     Age
               mean   min   max count  mean
Department
CS            750.0   750   750     2  28.5
CyS          1000.0  1000  1000     1  26.0
IMS          1000.0  1000  1000     5  30.0
IT            500.0   500   500     2  44.0
```

[47]:
```
# Simple grouping

print("\n Simple groping table (1):\n", df_new.groupby("Department")["Salary"].
 ↪mean())
print("\n Simple groping table (2):\n", df_new.groupby("Department")["Salary"].
 ↪agg(["mean", "count"]))
```

```
 Simple groping table (1):
 Department
CS      750.0
CyS    1000.0
```

```
IMS    1000.0
IT      500.0
Name: Salary, dtype: float64

 Simple groping table (2):
                mean  count
Department
CS           750.0      2
CyS         1000.0      1
IMS         1000.0      5
IT           500.0      2
```

Sorting

[49]:
```
# Sort by single column
df_sorted = df_new.sort_values("Salary", ascending=False)
print("\n Sorted data frame by Salary:\n", df_sorted)

# Sort by multiple columns
df_sorted = df_new.sort_values(["Department", "Salary"], ascending=[True,False])
print("\n Sorted data frame by Department and Salary:\n", df_sorted)
```

```
 Sorted data frame by Salary:
        Name  Age Department  Salary
0       Ali   22        IMS    1000
1  Mohammed   30        IMS    1000
9     Qusai   23        IMS    1000
8    Labeed   50        IMS    1000
7      Alaa   25        IMS    1000
6    Zainab   26        CyS    1000
5    Fatima   30         CS     750
2     Abass   27         CS     750
4     Hyder   48         IT     500
3    Hassan   40         IT     500

 Sorted data frame by Department and Salary:
        Name  Age Department  Salary
2     Abass   27         CS     750
5    Fatima   30         CS     750
6    Zainab   26        CyS    1000
0       Ali   22        IMS    1000
1  Mohammed   30        IMS    1000
7      Alaa   25        IMS    1000
8    Labeed   50        IMS    1000
9     Qusai   23        IMS    1000
3    Hassan   40         IT     500
4     Hyder   48         IT     500
```

5.Most Useful Methods

Value Counts

[52]:
```
# Count unique values

import pandas as pd
df = pd.read_excel("E:/Labeed data_New/lectures/python/Data sets/Employee.xlsx")
df_new = df.copy()
print("\n Original data frame:\n", df_new,"\n")

print("\n Employees cont in each department:\n", df_new["Department"].
 ↪value_counts())
```

```
 Original data frame:
        Name  Age Department  Salary
0       Ali   22        IMS    1000
1  Mohammed   30        IMS    1000
2     Abass   27         CS     750
3    Hassan   40         IT     500
4     Hyder   48         IT     500
5    Fatima   30         CS     750
6    Zainab   26        CyS    1000
7      Alaa   25        IMS    1000
8    Labeed   50        IMS    1000
9     Qusai   23        IMS    1000


 Employees cont in each department:
 Department
IMS    5
CS     2
IT     2
CyS    1
Name: count, dtype: int64
```

[55]:
```
# Count with percentage
print("\n Employees cont in each department %:\n", df_new["Department"].
 ↪value_counts(normalize=True)*100)
```

```
 Employees cont in each department %:
 Department
IMS    50.0
CS     20.0
IT     20.0
CyS    10.0
Name: proportion, dtype: float64
```

String Operations

[56]:
```
import pandas as pd
df = pd.read_excel("E:/Labeed data_New/lectures/python/Data sets/Employee.xlsx")
df_new = df.copy()
print("\n Original data frame:\n", df_new,"\n")
```

```
 Original data frame:
        Name  Age Department  Salary Employ_Date              Email
0       Ali   22         IMS    1000  2022-01-06       Ali@gmail.com
1  Mohammed   30         IMS    1000  2023-01-07  Mohammed@gmail.com
2     Abass   27          CS     750  2021-02-08     Abass@gmail.com
3    Hassan   40          IT     500  2022-07-09    Hassan@gmail.com
4     Hyder   48          IT     500  2022-01-10     Hyder@gmail.com
5    Fatima   30          CS     750  2022-10-07    Fatima@gmail.com
6    Zainab   26         CyS    1000  2018-01-12    Zainab@gmail.com
7      Alaa   25         IMS    1000  2022-02-06      Alaa@gmail.com
8    Labeed   50         IMS    1000  2019-01-05    Labeed@gmail.com
9     Qusai   23         IMS    1000  2022-01-06     Qusai@gmail.com
```

[57]:
```
# String methods on string columns

print("\n Uppercase:\n", df_new["Name"].str.upper()) # Convert to uppercase␣
 ↪letters
```

```
 Uppercase:
 0         ALI
1    MOHAMMED
2       ABASS
3      HASSAN
4       HYDER
5      FATIMA
6      ZAINAB
7        ALAA
8      LABEED
9       QUSAI
Name: Name, dtype: object
```

[59]:
```
print("\n Lowercase:\n", df_new["Name"].str.lower()) # Convert to lowercase␣
 ↪letters
```

```
 Lowercase:
 0         ali
1    mohammed
2       abass
```

```
3      hassan
4       hyder
5      fatima
6      zainab
7        alaa
8      labeed
9       qusai
Name: Name, dtype: object
```

[61]:
```
# Remove whitespace from start/end - fixes common data entry issues
print("\n Remove whitespace from start/end:\n", df_new["Name"].str.strip())
```

```
 Remove whitespace from start/end:
 0         Ali
1    Mohammed
2       Abass
3      Hassan
4       Hyder
5      Fatima
6      Zainab
7        Alaa
8      Labeed
9       Qusai
Name: Name, dtype: object
```

[62]:
```
# Extract domain from email by splitting at '@' and taking second part

print("\n Extracted Email domain:\n", df_new["Email"].str.split("@").str[1])
```

```
 Extracted Email domain:
 0    gmail.com
1    gmail.com
2    gmail.com
3    gmail.com
4    gmail.com
5    gmail.com
6    gmail.com
7    gmail.com
8    gmail.com
9    gmail.com
Name: Email, dtype: object
```

Note: 1. .str accessor enables string operations on entire columns. 2. Chain operations - apply multiple transformations. 3. Real-world use: Data cleaning, feature extraction, and text normalization. 4. Common for: Emails, names, addresses, categories, text analysis

DateTime Operations

[63]:
```
# Extract date parts

df_new["Year"] = df_new["Employ_Date"].dt.year
df_new["Month"] = df_new["Employ_Date"].dt.month
df_new["Day"] = df_new["Employ_Date"].dt.day
print("\n Updated data frame:\n", df_new,"\n")
```

```
 Updated data frame:
        Name  Age Department  Salary Employ_Date               Email  Year  \
0       Ali   22         IMS    1000  2022-01-06       Ali@gmail.com  2022
1  Mohammed   30         IMS    1000  2023-01-07  Mohammed@gmail.com  2023
2     Abass   27          CS     750  2021-02-08     Abass@gmail.com  2021
3    Hassan   40          IT     500  2022-07-09    Hassan@gmail.com  2022
4     Hyder   48          IT     500  2022-01-10     Hyder@gmail.com  2022
5    Fatima   30          CS     750  2022-10-07    Fatima@gmail.com  2022
6    Zainab   26         CyS    1000  2018-01-12    Zainab@gmail.com  2018
7      Alaa   25         IMS    1000  2022-02-06      Alaa@gmail.com  2022
8    Labeed   50         IMS    1000  2019-01-05    Labeed@gmail.com  2019
9     Qusai   23         IMS    1000  2022-01-06     Qusai@gmail.com  2022

   Month  Day
0      1    6
1      1    7
2      2    8
3      7    9
4      1   10
5     10    7
6      1   12
7      2    6
8      1    5
9      1    6
```

6. Combining DataFrames

Concatination

Note: DataFrames don't need to have the same structure when use pd.concat()

[1]:
```
# Merge multiple data frames
import pandas as pd

# DataFrames with different columns
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'C': [7, 8]})  # Different column 'C'
df3 = pd.DataFrame({'B': [9, 10], 'D': [11, 12]})  # Different columns

combined = pd.concat([df1, df2, df3], ignore_index=True)
```

```
print(combined)
```

```
     A     B    C     D
0  1.0   3.0  NaN   NaN
1  2.0   4.0  NaN   NaN
2  5.0   NaN  7.0   NaN
3  6.0   NaN  8.0   NaN
4  NaN   9.0  NaN  11.0
5  NaN  10.0  NaN  12.0
```

[2]:
```
# DataFrames with identical columns
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})  # Same columns
df3 = pd.DataFrame({'A': [9, 10], 'B': [11, 12]})  # Same columns

combined = pd.concat([df1, df2, df3], ignore_index=True)
print(combined)
```

```
    A   B
0   1   3
1   2   4
2   5   7
3   6   8
4   9  11
5  10  12
```

Merging (like SQL JOIN) 1. Inner Join - Only matching records from both data frames.

[5]:
```
import pandas as pd

# Sample data frame
employees = pd.DataFrame({
    'emp_id': [1, 2, 3, 4],
    'name': ['Alice', 'Bob', 'Charlie', 'Diana'],
    'dept_id': [101, 102, 101, 103]
})
departments = pd.DataFrame({
    'dept_id': [101, 102, 104],
    'dept_name': ['HR', 'Tech', 'Finance']
})

# Inner Join - only employees with valid departments
inner_merged = pd.merge(employees, departments, on= 'dept_id', how='inner')
print("\n Inner Join Result:\n", inner_merged)
```

```
 Inner Join Result:
    emp_id     name  dept_id dept_name
0       1    Alice      101        HR
```

```
1       2      Bob      102      Tech
2       3  Charlie      101        HR
```

2. Left Join - All records from left data frame + matches from right one

[6]:
```
# Left Join - all employees, with department info if available
left_join = pd.merge(employees, departments, on='dept_id', how='left')
print("\n left Join Result:\n", left_join)
```

```
 left Join Result:
    emp_id     name  dept_id dept_name
0       1    Alice      101        HR
1       2      Bob      102      Tech
2       3  Charlie      101        HR
3       4    Diana      103       NaN
```

3. Complete Example with All Join Types.

[7]:
```
# More comprehensive example
customers = pd.DataFrame({
    'customer_id': [1, 2, 3, 4],
    'name': ['John', 'Jane', 'Bob', 'Alice'],
    'city': ['NYC', 'LA', 'Chicago', 'Miami']
})

orders = pd.DataFrame({
    'order_id': [101, 102, 103, 104],
    'customer_id': [1, 2, 1, 5],  # Note: customer_id 5 doesn't exist in␣
 ↪customers
    'amount': [150, 200, 300, 400]
})

print("Customers:")
print(customers)
print("\nOrders:")
print(orders)

# Different join types
inner = pd.merge(customers, orders, on='customer_id', how='inner')
left = pd.merge(customers, orders, on='customer_id', how='left')
right = pd.merge(customers, orders, on='customer_id', how='right')
outer = pd.merge(customers, orders, on='customer_id', how='outer')

print("\nInner Join (matching customers & orders):")
print(inner)

print("\nLeft Join (all customers + their orders):")
print(left)
```

```
print("\nRight Join (all orders + customer info):")
print(right)

print("\nFull Outer Join (all customers + all orders):")
print(outer)
```

```
Customers:
   customer_id   name     city
0            1   John      NYC
1            2   Jane       LA
2            3    Bob  Chicago
3            4  Alice    Miami

Orders:
   order_id  customer_id  amount
0       101            1     150
1       102            2     200
2       103            1     300
3       104            5     400

Inner Join (matching customers & orders):
   customer_id  name city  order_id  amount
0            1  John  NYC       101     150
1            1  John  NYC       103     300
2            2  Jane   LA       102     200

Left Join (all customers + their orders):
   customer_id   name     city  order_id  amount
0            1   John      NYC     101.0   150.0
1            1   John      NYC     103.0   300.0
2            2   Jane       LA     102.0   200.0
3            3    Bob  Chicago       NaN     NaN
4            4  Alice    Miami       NaN     NaN

Right Join (all orders + customer info):
   customer_id  name city  order_id  amount
0            1  John  NYC       101     150
1            2  Jane   LA       102     200
2            1  John  NYC       103     300
3            5   NaN  NaN       104     400

Full Outer Join (all customers + all orders):
   customer_id   name     city  order_id  amount
0            1   John      NYC     101.0   150.0
1            1   John      NYC     103.0   300.0
2            2   Jane       LA     102.0   200.0
3            3    Bob  Chicago       NaN     NaN
```

```
4            4  Alice    Miami       NaN     NaN
5            5    NaN      NaN     104.0   400.0
```

Join Types Summary:

1. inner: Only records with matches in BOTH DataFrames.
2. left: ALL records from the left DataFrame + matches from the right one.
3. right: ALL records from the right DataFrame + matches from the left.
4. outer: ALL records from BOTH DataFrames.

7. Exporting Data

[12]:
```python
import pandas as pd
# creating Data Frame

df = pd.DataFrame({
    'Name': ['Alice', 'Bob'],
    'Age': [25, 30],
    'Salary': [50000, 60000]
})

# Save WITH index (adds extra column to the df for index)
df.to_csv('E:/Labeed data_New/lectures/python/Data sets/with_index.csv')
# File content will be:
# , Name, Age, Salary
# 0, Alice, 25,50000
# 1 Bob, 30, 60000

# Save WITHOUT index (clean)
df.to_csv('E:/Labeed data_New/lectures/python/Data sets/without_index.csv',␣
 ↪index=False)
# File content will be:
# Name, Age, Salary
# Alice, 25, 50000
# Bob,30, 60000

# Save to Excel
df.to_excel('E:/Labeed data_New/lectures/python/Data sets/woutput.xlsx',␣
 ↪index=False)
 #Creates: employees.xlsx with clean data (no index column)

# Save to JSON
df.to_json('E:/Labeed data_New/lectures/python/Data sets/data.json',␣
 ↪orient='records') #orient='index'
```

8. Practical Example: Sales Data Analysis

[13]:
```python
# Sample sales analysis
import pandas as pd
```

```python
import numpy as np

# Create sample sales data
sales_data = {
    'Date': pd.date_range('2024-01-01', periods=100, freq='D'),
    'Product': np.random.choice(['A', 'B', 'C'], 100),
    'Sales': np.random.randint(100, 1000, 100),
    'Region': np.random.choice(['North', 'South', 'East', 'West'], 100)
}
sales_df = pd.DataFrame(sales_data)

# Monthly sales by product
monthly_sales = sales_df.groupby([
    sales_df['Date'].dt.month,
    'Product'
])['Sales'].sum().unstack() # .unstack() Reshape from long to wide format␣
 ↪(Products become columns,
                            # months become rows)

print(monthly_sales)
```

```
Product     A     B     C
Date
1        6479  3878  6432
2        3926  5237  6426
3        2932  8729  5497
4         279  2952  2207
```

Key Tips:

1. Use inplace=True to modify the original DataFrame.
2. iloc for position-based indexing, loc for label-based.
3. Chain operations: df.query('Age > 25').groupby('Dept')['Salary'].mean().
4. Always check df.info() after reading data.
5. Use copy() when you need to preserve original data

Pandas Commands & Methods Summary

Command/Method	Description	Example
pd.DataFrame()	Create DataFrame from dict	df=pd.DataFrame({'A':[1,2]})
pd.read_csv()	Read CSV file	df=pd.read_csv('data.csv')
pd.read_excel()	Read Excel file	df=pd.read_excel('data.xlsx')
df.head()	First 5 rows	df.head()
df.tail()	Last 5 rows	df.tail(3)
df.info()	Data types & missing values	df.info()
df.describe()	Statistical summary	df.describe()
df.shape	Dimensions (rows, cols)	df.shape
df['col']	Select single column	df['Name']
df[['c1','c2']]	Select multiple columns	df[['Name','Age']]

Command/Method	Description	Example
df.iloc[]	Select by position	df.iloc[0], df.iloc[0:3]
df.loc[]	Select by label	df.loc[0,'Name']
Boolean Filtering	Filter by condition	df[df['Age']>30]
df.query()	Filter with query string	df.query('Age>25 & Salary>50000')
df.isnull()	Find missing values	df.isnull().sum()
df.dropna()	Remove rows with NaN	df_clean=df.dropna()
df.fillna()	Fill missing values	df['Age'].fillna(df['Age'].mean())
df.drop_duplicates()	Remove duplicates	df.drop_duplicates()
df.astype()	Convert data types	df['Age']=df['Age'].astype(int)
pd.to_datetime()	Convert to datetime	df['Date']=pd.to_datetime(df['Date'])
df['new_col']	Add new column	df['Bonus']=df['Salary']*0.1
df.apply()	Apply function	df['Senior']=df['Age'].apply(lambda x:'Yes' if x>30 else 'No')
df.groupby()	Group for aggregation	df.groupby('Dept')['Salary'].mean()
df.agg()	Multiple aggregations	df.agg({'Sales':['sum','mean']})
df.sort_values()	Sort DataFrame	df.sort_values('Salary',ascending=False)
df.value_counts()	Count unique values	df['Dept'].value_counts()
df.str.upper()	String uppercase	df['Name']=df['Name'].str.upper()
df.str.strip()	Remove whitespace	df['Name']=df['Name'].str.strip()
df.str.split()	Split strings	df['Domain']=df['Email'].str.split('@').str[1]
df.dt.month	Extract month from date	df['Month']=df['Date'].dt.month
df.dt.year	Extract year from date	df['Year']=df['Date'].dt.year
pd.concat()	Combine DataFrames	combined=pd.concat([df1,df2])
pd.merge()	Join DataFrames (SQL join)	merged=pd.merge(df1,df2,on='id')
df.to_csv()	Save to CSV	df.to_csv('data.csv',index=False)
df.to_excel()	Save to Excel	df.to_excel('data.xlsx',index=False)
df.to_json()	Save to JSON	df.to_json('data.json',orient='records')
df.unstack()	Reshape long to wide	grouped.unstack()
df.pivot_table()	Create pivot table	df.pivot_table(values='Sales', index='Month',columns='Product')
df.rename()	Rename columns	df.rename(columns={'old':'new'})
df.set_index()	Set column as index	df.set_index('ID')
df.reset_index()	Reset index	df.reset_index()
df.corr()	Correlation matrix	df.corr()
df.plot()	Create plots	df['Sales'].plot(kind='bar')
df.nlargest()	Get largest values	df.nlargest(5,'Salary')
df.nsmallest()	Get smallest values	df.nsmallest(3,'Age')
df.sample()	Random sample	df.sample(n=10)
df.memory_usage()	Memory usage	df.memory_usage(deep=True)
df.select_dtypes()	Select by data type	df.select_dtypes(include=['number'])
df.where()	Replace where condition False	df.where(df>0,0)
df.mask()	Replace where condition True	df.mask(df<0,0)
df.explode()	Transform list to rows	df.explode('Categories')
pd.cut()	Bin values	pd.cut(df['Age'],bins=[0,30,50,100])
df.rolling()	Rolling calculations	df['Sales'].rolling(7).mean()

Command/Method	Description	Example
df.copy()	Create copy	df_copy=df.copy()
df.columns	Get column names	df.columns
df.index	Get index values	df.index
df.dtypes	Get data types	df.dtypes
df.isin()	Check if in list	df[df[‘Dept’].isin([‘HR’,‘Tech’])]
df.replace()	Replace values	df.replace({‘Yes’:True,‘No’:False})
df.iterrows()	Iterate over rows	for i,r in df.iterrows():
df.items()	Iterate over columns	for c,d in df.items():