

# Pandas handbook

written by *Eng Mohamed Amir*

## What is Pandas?

**Pandas** is a powerful, open-source Python library used for **data manipulation**, **cleaning**, and **analysis**.

It provides two main data structures:

**Series**: A one-dimensional labeled array

**DataFrame**: A two-dimensional labeled table (like an Excel sheet or SQL table)

Pandas makes working with structured data fast, expressive, and flexible. If you're working with tables, spreadsheets, or CSVs in Python—Pandas is your best friend.

## why use pandas?

اللي هو ايوا يعني اي لازمتها او هستخدمنها في ايه الاجابة ممكن تبيان من الجدول دا

Task	Without Pandas	With Pandas
Load a CSV	open() + loops	pd.read_csv()
Filter rows	Custom loop logic	df[df["col"] > 5]
Group & summarize	Manual aggregation	df.groupby()
Merge two datasets	Nested loops	pd.merge()

تقدر تقول كدا ان

**Pandas saves time, reduces code, and increases readability.**

طب انا اقدر اناديها ازاي ببساطه

### Importing Pandas

```
import pandas as pd
```

**pd** is the standard alias used by the data science community

## Pandas vs Excel vs SQL vs NumPy

## 1.Excel:

من مميزاته انه واجهة مستخدم سهلة ومناسب لو بتعامل مع داتا صغيره تقدر تعمل جداول وتحليلات بسرعة من غير كود اما عيوبه انه بطئ لما البيانات تكبر

## 2.SQL

سرع جدا لو عايز تجيب او تصفي بيانات معينة querying for big data قوي جدا في ال

طب طالما جامد كدا ليه مش بنستخدمه في شغلنا في التعلم الالي بص يافنان له عيب مش مساعدنا معقدة بمعنى لو عايز تعمل transformation Logic نعتمد في شغلنا انه مش ممتاز لو عايز تعمل عمليات حسابية كتير او معالجة متقدمة و تحتاج حلول تانية مثل بسيط: لو عندك جدول درجات، وعايز تحسب المعدل لكل طالب، أو تعمل فلترة حسب شرط معين، أو تجمع بيانات بطريقة معينة.

كويس للفلترة أو البحث البسيط، لكن لو العمليات معقدة، زي: جمع أعمدة معينة بطريقة SQL يعني هنا SQL خاصة تعديل شكل البيانات أو ترتيبها بطرق غير تقليدية حسابات متقدمة على كل الصفوف مش هيشهلك، لازم كود إضافي أو أداة تانية

## 3.Numpy

ممتاز للحسابات العلمية (low-level) منخفضة المستوى arrays سريع جداً مناسب للعمليات على والرياضية

يعني لو عندك جدول بيانات مش هتعرف تسمى الصفوف أو الأعمدة بسهولة، اما عيوبه مفيش جداول زي (tabular) أصعب شوية لو البيانات Excel)

## فك معایا کدا

سهل، بس بطئ على البيانات الكبيرة، ومعقد لو عايز تعمل تحليلات أو: Excel في الأول كان عندنا تغييرات متقدمة.

سرع جداً وعالي الأداء، لكن: NumPy كمان كان عندنا

(labels) مش فيه أسماء للأعمدة أو الصفوف

صعب تتعامل مع بيانات جداول كبيرة ومعقدة

ومن هنا ظهر نجمنا الـ

## Pandas

Pandas يكون:

Label-aware تقدر تسمى الأعمدة والصفوف زي → Excel

سرع ومرن → الأداء بتاعه جاي من NumPy

بسهولة merge، قوي في تحليل البيانات → تعمل فلترة، تجميع، تعديل

## Summary

### Pandas vs Excel vs SQL vs NumPy

Tool	Strengths	Weaknesses
Excel	Easy UI, great for small data	Slow, manual, not scalable
SQL	Efficient querying of big data	Not ideal for transformation logic
NumPy	Fast, low-level array operations	No labels, harder for tabular data
Pandas	Label-aware, fast, flexible	Slightly steep learning curve

#### Notes:

- Pandas bridges the gap between NumPy performance and Excel-like usability.
- Pandas is built on top of NumPy.

## Core Data Structures in Pandas

Pandas is built on two main data structures:

1. **Series** → One-dimensional (like a single column in Excel)
  2. **DataFrame** → Two-dimensional (like a full spreadsheet or SQL table)
- 

## Series — 1D Labeled Array

A **Series** is like a list with labels (index).

```
import pandas as pd

s = pd.Series([10, 20, 30, 40])
print(s)
```

#### Output:

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

Notice the automatic index: 0, 1, 2, 3 . You can also define a custom index:

```
s = pd.Series([10, 20, 30], index=["a", "b", "c"])
```

A Pandas Series may look similar to a Python dictionary because both store data with labels, but a Series offers much more:

- Fast vectorized operations
  - Automatic index alignment during arithmetic
  - Handles missing data using `NaN`
  - Allows label-based and position-based access
  - Integrates seamlessly with Pandas DataFrames
- 

## DataFrame — 2D Labeled Table

A **DataFrame** is like a dictionary of Series — multiple columns with labels.

```
data = {  
    "name": ["Alice", "Bob", "Charlie"],  
    "age": [25, 30, 35],  
    "city": ["Delhi", "Mumbai", "Bangalore"]  
}  
  
df = pd.DataFrame(data)  
print(df)  
Output:
```

	name	age	city
0	Alice	25	Delhi
1	Bob	30	Mumbai
2	Charlie	35	Bangalore

- Each column in a DataFrame is a **Series**.
- Every Series and DataFrame has an **Index** — helps with:
  - Fast lookups
  - Aligning data
  - Merging & joining
  - Time series operations

```
df.index      # Row Labels  
df.columns   # Column Labels
```

- You can change them:

```
df.index = ["a", "b", "c"]  
df.columns = ["Name", "Age", "City"]
```

---

## Why Learn Series & DataFrame?

Most Pandas operations are built on these foundations:

- Selection
- Filtering
- Merging
- Aggregation

#### Summary:

- **Series** = 1D array with labels
  - **DataFrame** = 2D table with rows + columns
  - Both come with index and are the heart of Pandas
- 

## Creating DataFrames

### From Python Lists

```
data = [
    ["Alice", 25],
    ["Bob", 30],
    ["Charlie", 35]
]

df = pd.DataFrame(data, columns=["Name", "Age"])
```

### From Dictionary of Lists

```
data = {
    "Name": ["Alice", "Bob", "Charlie"],
    "Age": [25, 30, 35]
}

df = pd.DataFrame(data)
```

### From NumPy Arrays

```
import numpy as np

arr = np.array([[1, 2], [3, 4]])
df = pd.DataFrame(arr, columns=["A", "B"])
```

### From CSV Files

```
df = pd.read_csv("data.csv", usecols=["Name", "Age"], nrows=10)
```

### From Excel Files

```
df = pd.read_excel("data.xlsx", sheet_name="Sales")
```

## From JSON

```
df = pd.read_json("data.json")
```

## From SQL Databases

```
import sqlite3
conn = sqlite3.connect("mydb.sqlite")
df = pd.read_sql("SELECT * FROM users", conn)
```

## From the Web

```
url = "https://raw.githubusercontent.com/mwaskom/seaborn-
data/master/tips.csv"
df = pd.read_csv(url)
```

---

# Exploratory Data Analysis (EDA)

EDA helps you understand the dataset:

- Spot patterns
- Identify anomalies
- Generate summary statistics
- Visualize data

### Common Methods:

```
df.head()      # First 5 rows
df.tail()      # Last 5 rows
df.info()      # Column info: types, non-nulls
df.describe()  # Stats for numeric columns
df.columns     # List of column names
df.shape       # (rows, columns)
```

---

# Data Selection & Filtering

## Selecting Columns

```
df["column_name"]      # Single column (Series)
df[["col1", "col2"]]   # Multiple columns (DataFrame)
```

## Selecting Rows

```
df.loc[0]      # First row by Label  
df.iloc[0]     # First row by position
```

## Specific Rows & Columns

```
df.loc[0, "Name"]  
df.iloc[0, 1]  
df.loc[0:2, ["Name", "Age"]]  
df.iloc[0:2, 0:2]
```

## Fast Single Element Access

```
df.at[0, "Name"]  
df.iat[0, 1]
```

## Filtering with Conditions

```
df[df["Age"] > 30]  
df[(df["Age"] > 25) & (df["City"] == "Delhi")]  
df.query("Age > 25 and City == 'Delhi'")
```

---

# Data Cleaning & Preprocessing

## Handling Missing Values

```
df.isnull()  
df.isnull().sum()  
df.dropna()  
df.fillna(0)  
df["Age"].fillna(df["Age"].mean())  
df.ffill()  
df.bfill()
```

## Detecting Duplicates

```
df.duplicated()  
df.drop_duplicates()  
df.duplicated(subset=["Name", "Age"])
```

## String Operations

```
df["Name"].str.lower()  
df["City"].str.contains("delhi", case=False)  
df["Email"].str.split("@")
```

## Type Conversion

```
df["Age"] = df["Age"].astype(int)
df["Date"] = pd.to_datetime(df["Date"])
df["Category"] = df["Category"].astype("category")
```

## Apply Functions

```
df["Age Group"] = df["Age"].apply(lambda x: "Adult" if x >= 18 else "Minor")
df["Gender"] = df["Gender"].map({"M": "Male", "F": "Female"})
df["City"].replace({"Del": "Delhi", "Mum": "Mumbai"})
```

---

## Data Transformation

### Sorting & Ranking

```
df.sort_values("Age")
df.sort_values(["Age", "Salary"])
df.reset_index(drop=True, inplace=True)
df.sort_index()
df["Rank"] = df["Score"].rank(method="dense")
```

### Renaming & Reordering Columns

```
df.rename(columns={"oldName": "newName"}, inplace=True)
df.columns = ["Name", "Age", "City"]
df = df[['City', 'Name', 'Age']]
```

### Reshaping: Melt & Pivot

```
df.melt(id_vars=["Name"], value_vars=["Math", "Science"],
var_name="Subject", value_name="Score")
df.pivot(index="Name", columns="Subject", values="Score")
df.pivot_table(index="Name", columns="Subject", values="Score",
aggfunc="mean")
```

---

## Aggregation & Grouping

```
df.groupby("Department")["Salary"].mean()
df.groupby(["Team", "Gender"])["Salary"].agg(["mean", "max", "min"])
df.groupby("Team")["Salary"].transform("mean")
df.groupby("Team").filter(lambda x: x["Salary"].mean() > 80)
```

---

## Merging & Joining

```
pd.merge(employees, departments, on="DeptID", how="inner") # Inner join
pd.merge(employees, departments, on="DeptID", how="left") # Left join
pd.merge(employees, departments, on="DeptID", how="outer") # Outer join
pd.concat([df1, df2])          # Stack vertically
pd.concat([df1, df2], axis=1)   # Stack horizontally
```

---

## Reading & Writing Files

```
# CSV
df = pd.read_csv("data.csv")
df.to_csv("output.csv", index=False)

# Excel
df = pd.read_excel("data.xlsx", sheet_name="Sales")
df.to_excel("output.xlsx", index=False)

# JSON
df = pd.read_json("data.json")
```

---

### Summary:

- Pandas is built on **Series (1D)** and **DataFrame (2D)**
- DataFrames can be created from lists, dicts, NumPy arrays, CSV/Excel/JSON files, SQL, or web
- Selection, filtering, cleaning, transformation, aggregation, reshaping, merging — all are built on these structures
- Mastering Series & DataFrame is **the foundation of data analysis in Pandas**

In [ ]:

This set of 100 practical exercises covers the progression of Pandas as outlined in the sources, starting from basic imports and Series creation, moving through DataFrame manipulation and data cleaning, and concluding with complex grouping operations.

## Section 1: Importing and Series Basics

1. **Import** both NumPy and Pandas using their standard aliases.
2. Create a Pandas **Series** from a list of five country names.
3. Create a Series named "marks" using five integer values.
4. Create a Series with **custom index labels** representing four subjects.
5. **Name** a Series "Jack Marks" and assign it to a variable.
6. Create a Series named `marks_series` directly from a **Python dictionary**.
7. Explain what happens to the keys when a Series is converted to a dictionary via `to_dict()`.
8. Use the `.size` attribute to find the total number of elements in `marks_series`.

9. Check the **data type** of the values in a Series using the correct attribute.
10. Retrieve the **name attribute** of a Series.
11. Use an attribute to check if all values in a Series are **unique** (returns True/False).
12. Access only the **index labels** of a Series.
13. Extract the data of a Series as a **NumPy array**.
14. Check the **Python type** of the array returned by `.values`.
15. **Hint:** Use `subs.csv`. Read the file using `pd.read_csv`. What is the default data structure created?
16. Use the `squeeze=True` parameter to force a single-column CSV into a Series.
17. **Hint:** Use `kohli_ipl.csv`. Read it as a Series while setting "match\_no" as the index.
18. **Hint:** Use `bollywood.csv`. Read it as a Series while setting "movie" as the index.

## Section 2: Series Methods and Statistics

19. Display the **first 5 rows** of the `sub` Series.
20. Display the **last 5 rows** of the `k1` Series.
21. Retrieve a **random sample** row from the `movies` Series.
22. Use `value_counts()` to find how many movies each lead actor has done in the `movies` Series.
23. Sort the `k1` (Kohli) Series by **runs scored** in ascending order.
24. **Method Chaining:** Sort `k1` in descending order and retrieve only the top score's value.
25. Make a sort **permanent** without re-assigning the variable.
26. Sort the `movies` Series alphabetically by its **index (movie titles)**.
27. Sort the `movies` Series by index in **descending order**.
28. Explain the difference between `count()` and `size` regarding missing values.
29. Find the **sum** of all subscribers in the `sub` Series.
30. Use the `product()` method to multiply all items in a Series.
31. Calculate the **mean** number of subscribers gained.
32. Find the **median** runs scored by Kohli.
33. Identify the **mode** (most frequent lead actor) in the `movies` Series.
34. Calculate the **standard deviation** of the `sub` Series.
35. Find the **variance** of the `sub` Series.
36. Identify the **minimum** value in the `sub` Series.
37. Identify the **maximum** value in the `sub` Series.
38. Use `describe()` to generate a statistical summary of the `movies` Series.
39. Use `describe()` on the `k1` Series and note the **quartile values**.

## Section 3: Indexing and Slicing

40. Create a Series `x` and access the element at **integer index position 1**.
41. Access the **very last element** of the `movies` Series using negative indexing.
42. **Slice** the `k1` Series to show matches 5 through 10.

43. Use **negative slicing** to get the last 5 rows of the `sub` Series.
44. Retrieve every **second element** in the `movies` Series using slicing.
45. **Fancy Indexing:** Retrieve runs for matches 1, 8, 22, 11, and 2 simultaneously.
46. Access the lead actor for 'Evening Shadows' using its **label index**.
47. Update the mark for 'english' in `marks_series` to 88 using its **index position**.
48. **Add a new entry** ('social': 90) to an existing `marks_series`.
49. Update the lead actor for 'Hum Tumhare Hain Sanam' to 'Jack' using its **index label**.
50. Use the `len()` function to find the length of the `sub` Series.
51. Convert `marks_series` into a standard **Python list**.
52. Convert `marks_series` into a **Python dictionary**.
53. Check if 'Jack' exists in the **values** of the `movies` Series using the membership operator.
54. **Looping:** Iterate through the `movies` Series and print every lead actor.
55. **Broadcasting:** Subtract every value in `marks_series` from 100 in one operation.
56. Use a **relational operator** to create a boolean Series showing where Kohli scored  $\geq 50$ .

## Section 4: Boolean Indexing and Filtering

57. Filter the `k1` Series to show only scores **greater than or equal to 50**.
58. Count how many times Kohli scored **0 runs (ducks)**.
59. Filter the `sub` Series for days with **more than 200 subscribers** and find the count.
60. Find actors in the `movies` Series who have appeared in **more than 20 movies**.
61. Plot a **line graph** of the `sub` Series.
62. Plot a **pie chart** of the top 20 lead actors.
63. Plot a **bar chart** of the top 20 lead actors.

## Section 5: Advanced Cleaning and Manipulation

64. Use `sys.getsizeof()` to check the memory usage of the `k1` Series.
65. **Type Conversion:** Convert the `k1` Series to `int16` to save memory.
66. Use `between()` to find all Kohli scores between 50 and 60.
67. Use `clip()` to cap the `sub` Series values between 100 and 200.
68. Remove **duplicate values** from a Series while keeping the first occurrence.
69. Remove duplicates but **keep the last occurrence**.
70. Use `duplicated().sum()` to find the total number of duplicate values in `k1`.
71. Use `isnull().sum()` to find the total count of missing values.
72. Remove all rows with **missing values** using `dropna()`.
73. Replace missing values with **0** using `fillna()`.
74. Replace missing values with the **mean of the Series**.
75. Use `isin()` to check if Kohli scored exactly 49 or 99.
76. Use `apply()` with a lambda function to split actor names into lists.
77. Use `apply()` to extract only the **first name** of every lead actor.
78. Convert all actor names to **uppercase** using `apply()`.

79. Apply a conditional lambda to `sub` to label days as '**good day**' or '**bad day**' based on the mean.
80. Create a **deep copy** of the first 5 rows of `k1` to avoid `SettingWithCopy` warnings.

## Section 6: DataFrame Operations

81. Create a **DataFrame from a list of lists** with columns 'iq', 'marks', and 'package'.
82. Create a DataFrame from a **dictionary of lists**.
83. **Set the index** of a DataFrame to the 'name' column permanently.
84. **Hint:** Use `movies.csv` and `ipl-matches.csv`. Read both into DataFrames and check their **shapes**.
85. List the **data types** of all columns in the `movies` DataFrame.
86. List all **column names** in the `ipl` DataFrame.
87. View a **random sample of 2 rows** from the `ipl` DataFrame.
88. Use `.info()` to get a summary of the `movies` DataFrame.
89. Count **missing values** in every column of the `movies` DataFrame.
90. **Rename** columns 'marks' to 'percent' and 'package' to 'ipa'.
91. Calculate the **row-wise sum** of a student DataFrame.
92. Select a **single column** ('title\_x') from the `movies` DataFrame.
93. Select **multiple columns** at once from the `movies` DataFrame.
94. Select the **second row** of the `movies` DataFrame using `iloc`.
95. Use `loc` to select a row by the index label 'parle'.
96. Select **rows 5 to 10** and the **first 3 columns** simultaneously using `iloc`.
97. **Filter** the `ipl` DataFrame to find all matches where 'MatchNumber' is 'Final'.
98. Find matches where 'City' is 'Kolkata' **AND** 'WinningTeam' is 'Chennai Super Kings'.
99. **Add a new column** 'country' to `movies` and set it to 'India'.
100. **Find the genre with the highest average IMDB rating** using `groupby`, `mean`, and `sort_values`.

**Note on Outside Information:** Some exercises suggest using specific CSV files (like `subs.csv` or `kohli_ipl.csv`). These are referenced directly from the examples provided in **the sources**. For exercise 100, the specific logic for finding the highest average rating utilizes the **groupby** and **aggregation** principles detailed in the final pages of the provided notes.

In [49]:

```
import numpy as np
import pandas as pd

sub = pd.Series(
    [1200, 1500, 1800, 1700, 1600, 2000, 2200, 2100, 1900, 2300],
    index=[
        "Jan", "Feb", "Mar", "Apr", "May",
        "Jun", "Jul", "Aug", "Sep", "Oct"
    ],
    name="Subscribers"
```

```

)
k1 = pd.Series(
    [45, 78, 102, 35, 89, 120, 66, 54],
    index=[
        "Match1", "Match2", "Match3", "Match4",
        "Match5", "Match6", "Match7", "Match8"
    ],
    name="Kohli Runs"
)

movies = pd.Series(
[
    "SRK", "Salman Khan", "Aamir Khan",
    "SRK", "Akshay Kumar", "SRK",
    "Salman Khan", "Akshay Kumar"
],
index=[
    "DDLJ", "Tiger Zinda Hai", "3 Idiots",
    "My Name Is Khan", "Kesari", "Don",
    "Bajrangi Bhaijaan", "Housefull"
],
name="Lead Actor"
)

```

## Section 1: Importing and Series Basics

In [1]: # Import both NumPy and Pandas using their standard aliases.

```

import numpy as np
import pandas as pd

```

In [3]: #Create a Series named "marks" using 6 integer values.

```

marks = pd.Series([87,80,83,82,93,75])
marks

```

Out[3]:

0	87
1	80
2	83
3	82
4	93
5	75

dtype: int64

In [4]: #Create a Pandas Series from a list of five country names.

```

country = pd.Series(['Egypt', 'USA', 'UK', 'Turkey', 'Palestine'])
country

```

```
Out[4]: 0      Egypt
        1      USA
        2      UK
        3    Turkey
        4  Palastine
       dtype: object
```

```
In [6]: #Create a Series with custom index labels representing four subjects.
grades = [87,80,82,93]
subjects = pd.Series(grades,index = ['Robotics','Image processing','OS','Software E
subjects
```

```
Out[6]: Robotics      87
        Image processing 80
        OS                82
        Software Engineering 93
       dtype: int64
```

```
In [9]: #Name a Series "medo Marks" and assign it to a variable.
grades = [87,80,82,93]
medo_marks = pd.Series(grades,index=['Robotics','Image processing','OS','Software E
medo_marks
```

```
Out[9]: Robotics      87
        Image processing 80
        OS                82
        Software Engineering 93
       Name: Medo's Marks, dtype: int64
```

```
In [11]: #Create a Series named marks_series directly from a Python dictionary.
```

```
marks_dict = {
    'Robotics':87,
    'Image processing':80,
    'OS':82,
    'Software Engineering':93
}

marks_series = pd.Series(marks_dict,name = 'medo')
marks_series
```

```
Out[11]: Robotics      87
        Image processing 80
        OS                82
        Software Engineering 93
       Name: medo, dtype: int64
```

```
In [ ]:
```

**Explain what happens to the keys when a Series is converted to a dictionary via to\_dict().**

When you convert a Series → Dictionary using to\_dict(), this happens exactly:

Each label (index) in the Series becomes a key in the dictionary.

Each value in the Series becomes the value in the dictionary.

The result is a regular dictionary, just like any dictionary in Python.

- ✓ In short: the Series' index becomes the dictionary keys, and the Series'

```
In [12]: # Series جملة
marks_series = pd.Series([85, 90, 78, 92], index=["Math", "Physics", "Chemistry", "English"])

# تحويل Series إلى dictionary
marks_dict = marks_series.to_dict()

print(marks_dict)

{'Math': 85, 'Physics': 90, 'Chemistry': 78, 'English': 92}
```

```
In [21]: #Use the .size attribute to find the total number of elements in marks_series.
marks_series = pd.Series([85, 90, 78, 92], index=["Math", "Physics", "Chemistry", "English"])

marks_series = pd.Series([85, 90, 78, 92], index=["Math", "Physics", "Chemistry", "English"])

marks_series.size
```

```
Out[21]: 4
```

```
In [22]: #Check the data type of the values in a Series using the correct attribute.
marks_series = pd.Series([85, 90, 78, 92], index=["Math", "Physics", "Chemistry", "English"])
marks_series.dtype
```

```
Out[22]: dtype('int64')
```

```
In [27]: #Retrieve the name attribute of a Series.
marks_series = pd.Series([85, 90, 78, 92], index=["Math", "Physics", "Chemistry", "English"])
marks_series = pd.Series([85, 90, 78, 92], index=["Math", "Physics", "Chemistry", "English"])
marks_series.name
```

```
Out[27]: 'mo Ayman'
```

```
In [28]: #Use an attribute to check if all values in a Series are unique (returns True/False)
marks_series = pd.Series([85, 90, 78, 92, 85])

marks_series = pd.Series([85, 90, 78, 92, 85])
marks_series.is_unique
```

```
Out[28]: False
```

```
In [31]: #Access only the index labels of a Series.
marks_series = pd.Series([85, 90, 78, 92], index=["Math", "Physics", "Chemistry", "English"])
#marks_series = pd.Series([85, 90, 78, 92, 85]) -> RangeIndex(start=0, stop=5, step=1)

marks_series = pd.Series([85, 90, 78, 92, 85])
```

```
#marks_series = pd.Series([85, 90, 78, 92], index=["Math", "Physics", "Chemistry"]  
marks_series.index
```

```
Out[31]: RangeIndex(start=0, stop=5, step=1)
```

```
In [33]: #Extract the data of a Series as a NumPy array.  
#nm = pd.Series([20,11,5])
```

```
nm = pd.Series([20,11,5])  
nm.to_numpy()
```

```
Out[33]: array([20, 11, 5])
```

```
In [35]: #Check the Python type of the array returned by .values.  
#nm = pd.Series([85, 90, 78, 92, 85])  
nm = pd.Series([85, 90, 78, 92, 85])  
nm.values
```

```
Out[35]: array([85, 90, 78, 92, 85])
```

## Section 2: Series Methods and Statistics

```
In [9]: import numpy as np  
import pandas as pd  
  
sub = pd.Series(  
    [1200, 1500, 1800, 1700, 1600, 2000, 2200, 2100, 1900, 2300],  
    index=[  
        "Jan", "Feb", "Mar", "Apr", "May",  
        "Jun", "Jul", "Aug", "Sep", "Oct"  
    ],  
    name="Subscribers"  
)  
  
k1 = pd.Series(  
    [45, 78, 102, 35, 89, 120, 66, 54],  
    index=[  
        "Match1", "Match2", "Match3", "Match4",  
        "Match5", "Match6", "Match7", "Match8"  
    ],  
    name="Kohli Runs"  
)  
  
movies = pd.Series(  
    [  
        "SRK", "Salman Khan", "Aamir Khan",  
        "SRK", "Akshay Kumar", "SRK",  
        "Salman Khan", "Akshay Kumar"  
    ],
```

```
index=[  
    "DDLJ", "Tiger Zinda Hai", "3 Idiots",  
    "My Name Is Khan", "Kesari", "Don",  
    "Bajrangi Bhaijaan", "Housefull"  
,  
    name="Lead Actor"  
)
```

In [37]: #19. Display the \*\*first 5 rows\*\* of the `sub` Series.

```
sub.head(5)
```

Out[37]:

Jan	1200
Feb	1500
Mar	1800
Apr	1700
May	1600

Name: Subscribers, dtype: int64

In [38]: #20. Display the \*\*last 5 rows\*\* of the `kl` Series.

```
kl.tail(5)
```

Out[38]:

Match4	35
Match5	89
Match6	120
Match7	66
Match8	54

Name: Kohli Runs, dtype: int64

In [44]: #21. Retrieve a \*\*random sample\*\* row from the `movies` Series.

```
movies.sample()
```

Out[44]:

Tiger Zinda Hai	Salman Khan
-----------------	-------------

Name: Lead Actor, dtype: object

In [51]: #22. Use \*\*`value\_counts()`\*\* to find how many movies each Lead actor has done in t

```
print("index: ",movies.index)  
print()  
print("values: ",movies.values)  
print()  
print(movies)  
print()  
print("value_count: ", movies.value_counts())
```

```
index: Index(['DDLJ', 'Tiger Zinda Hai', '3 Idiots', 'My Name Is Khan', 'Kesari',  
           'Don', 'Bajrangi Bhaijaan', 'Housefull'],  
           dtype='object')  
  
values: ['SRK' 'Salman Khan' 'Aamir Khan' 'SRK' 'Akshay Kumar' 'SRK' 'Salman Khan'  
        'Akshay Kumar']  
  
DDLJ SRK  
Tiger Zinda Hai Salman Khan  
3 Idiots Aamir Khan  
My Name Is Khan SRK  
Kesari Akshay Kumar  
Don SRK  
Bajrangi Bhaijaan Salman Khan  
Housefull Akshay Kumar  
Name: Lead Actor, dtype: object  
  
value_count: Lead Actor  
SRK 3  
Salman Khan 2  
Akshay Kumar 2  
Aamir Khan 1  
Name: count, dtype: int64
```

In [53]: #23. Sort the `kl` (Kohli) Series by \*\*runs scored\*\* in ascending order.

```
print(kl)  
print("\n\n")  
print(kl.sort_values())
```

```
Match1    45  
Match2    78  
Match3   102  
Match4    35  
Match5    89  
Match6   120  
Match7    66  
Match8    54  
Name: Kohli Runs, dtype: int64
```

```
Match4    35  
Match1    45  
Match8    54  
Match7    66  
Match2    78  
Match5    89  
Match3   102  
Match6   120  
Name: Kohli Runs, dtype: int64
```

In [6]: #24. \*\*Method Chaining:\*\* Sort `kl` in descending order and retrieve only the top s  
kl.sort\_values(ascending=False).iloc[0]

Out[6]: np.int64(120)

```
In [8]: #25. Make a sort **permanent** without re-assigning the variable.  
kl.sort_values(inplace=True)  
kl
```

```
Out[8]: Match4      35  
Match1      45  
Match8      54  
Match7      66  
Match2      78  
Match5      89  
Match3     102  
Match6     120  
Name: Kohli Runs, dtype: int64
```

```
In [12]: #26. Sort the `movies` Series alphabetically by its **index (movie titles)**.  
print(movies)  
print()  
print()  
movies.sort_index()
```

```
DDJ           SRK  
Tiger Zinda Hai    Salman Khan  
3 Idiots        Aamir Khan  
My Name Is Khan       SRK  
Kesari          Akshay Kumar  
Don             SRK  
Bajrangi Bhaijaan   Salman Khan  
Housefull        Akshay Kumar  
Name: Lead Actor, dtype: object
```

```
Out[12]: 3 Idiots        Aamir Khan  
Bajrangi Bhaijaan   Salman Khan  
DDJ           SRK  
Don             SRK  
Housefull        Akshay Kumar  
Kesari          Akshay Kumar  
My Name Is Khan       SRK  
Tiger Zinda Hai    Salman Khan  
Name: Lead Actor, dtype: object
```

```
In [13]: #27. Sort the `movies` Series by index in **descending order**.  
movies.sort_index(ascending=False)
```

```
Out[13]: Tiger Zinda Hai    Salman Khan  
My Name Is Khan       SRK  
Kesari          Akshay Kumar  
Housefull        Akshay Kumar  
Don             SRK  
DDJ           SRK  
Bajrangi Bhaijaan   Salman Khan  
3 Idiots        Aamir Khan  
Name: Lead Actor, dtype: object
```

```
In [ ]: #28. Explain the difference between **`count()`** and **`size`** regarding missing values
```

```
In [16]: #29. Find the **sum** of all subscribers in the `sub` Series.  
print(sub.count()) # excludes missing (NaN) values  
print(sub.size ) # includes missing (NaN) values  
#####very important #####
```

```
10
```

```
10
```

```
In [21]: #30. Use the **`product()`** method to multiply all items in a Series.  
sr = pd.Series([1,2,3,4,5])  
print(sr)  
print()  
print()  
print(sr.product())
```

```
0    1  
1    2  
2    3  
3    4  
4    5  
dtype: int64
```

```
120
```

```
In [22]: #31. Calculate the **mean** number of subscribers gained.  
  
sub.mean()
```

```
Out[22]: np.float64(1830.0)
```

```
In [25]: #32. Find the **median** runs scored by Kohli.  
kl.median()
```

```
Out[25]: 72.0
```

```
In [28]: #33. Identify the **mode** (most frequent Lead actor) in the `movies` Series.  
  
movies.mode()
```

```
Out[28]: 0    SRK  
Name: Lead Actor, dtype: object
```

```
In [29]: #34. Calculate the **standard deviation** of the `sub` Series.  
  
sub.std()
```

```
Out[29]: 340.0980250849256
```

```
In [30]: #35. Find the **variance** of the `sub` Series.  
  
sub.var()
```

```
Out[30]: 115666.66666666667
```

```
In [31]: #36. Identify the **minimum** value in the `sub` Series.
```

```
sub.min()
```

```
Out[31]: 1200
```

```
In [32]: #37. Identify the **maximum** value in the `sub` Series.
```

```
sub.max()
```

```
Out[32]: 2300
```

```
In [35]: #38. Use **`describe()`** to generate a statistical summary of the `movies` Series.
```

```
print(movies)  
print()  
print()  
movies.describe()
```

```
          DDLJ           SRK  
Tiger Zinda Hai      Salman Khan  
3 Idiots            Aamir Khan  
My Name Is Khan       SRK  
Kesari              Akshay Kumar  
Don                 SRK  
Bajrangi Bhaijaan    Salman Khan  
Housefull           Akshay Kumar  
Name: Lead Actor, dtype: object
```

```
Out[35]: count      8  
unique      4  
top        SRK  
freq       3  
Name: Lead Actor, dtype: object
```

```
In [36]: #39. Use `describe()` on the `kl` Series and note the **quartile values**.
```

```
print(kl)  
print()  
print()  
print()  
kl.describe()
```

```
          Match1     45  
Match2     78  
Match3    102  
Match4     35  
Match5     89  
Match6    120  
Match7     66  
Match8     54  
Name: Kohli Runs, dtype: int64
```

```
Out[36]: count      8.000000
          mean      73.625000
          std       29.193627
          min       35.000000
          25%      51.750000
          50%      72.000000
          75%      92.250000
          max      120.000000
Name: Kohli Runs, dtype: float64
```

## Section 3: Indexing and Slicing

```
In [45]: #40. Create a Series `x` and access the element at **integer index position 1**.
```

```
x = pd.Series([10,20,30,40,50],index = [9,8,7,6,5])
print(x.iloc[1])
print()

try:
    print(x[1])#####
except:
    print("error")

print(x[8]) #my custom index 8 give 10
```

```
20
```

```
error
20
```

```
In [48]: #41. Access the **very last element** of the `movies` Series using negative indexing
```

```
print(movies)
print()
print()
print()

movies.iloc[-1]
```

```
DDJ                  SRK
Tiger Zinda Hai      Salman Khan
3 Idiots             Aamir Khan
My Name Is Khan      SRK
Kesari                Akshay Kumar
Don                  SRK
Bajrangi Bhaijaan    Salman Khan
Housefull            Akshay Kumar
Name: Lead Actor, dtype: object
```

```
Out[48]: 'Akshay Kumar'
```

```
In [64]: #42. **Slice** the `kl` Series to show matches 5 through 10.  
arkl = np.array([1,2,3,4,5,6,7,8])  
print('normal array',arkl[4:8])  
print()  
print()  
  
print(kl)  
print()  
print()  
  
kl.iloc[4:8]
```

```
normal array [5 6 7 8]
```

```
Match1      45  
Match2      78  
Match3     102  
Match4      35  
Match5      89  
Match6     120  
Match7      66  
Match8      54  
Name: Kohli Runs, dtype: int64
```

```
Out[64]: Match5      89  
Match6     120  
Match7      66  
Match8      54  
Name: Kohli Runs, dtype: int64
```

```
In [65]: #43. Use **negative slicing** to get the last 5 rows of the `sub` Series.
```

```
print(sub)  
print()  
sub.tail()
```

```
Jan      1200  
Feb      1500  
Mar      1800  
Apr      1700  
May      1600  
Jun      2000  
Jul      2200  
Aug      2100  
Sep      1900  
Oct      2300  
Name: Subscribers, dtype: int64
```

```
Out[65]: Jun    2000
          Jul    2200
          Aug    2100
          Sep    1900
          Oct    2300
Name: Subscribers, dtype: int64
```

```
In [66]: #44. Retrieve every **second element** in the `movies` Series using slicing.
```

```
print(movies)
print()
movies.iloc[::2]
```

```
DDJ           SRK
Tiger Zinda Hai      Salman Khan
3 Idiots          Aamir Khan
My Name Is Khan       SRK
Kesari          Akshay Kumar
Don             SRK
Bajrangi Bhaijaan     Salman Khan
Housefull         Akshay Kumar
Name: Lead Actor, dtype: object
```

```
Out[66]: DDLJ           SRK
          3 Idiots        Aamir Khan
          Kesari          Akshay Kumar
          Bajrangi Bhaijaan     Salman Khan
Name: Lead Actor, dtype: object
```

## Fancy Indexing

يعني إيه Fancy Indexing؟

لغوياً:

Fancy = ذكي / متقدم

Indexing = الوصول للعناصر

يعني برمجياً NumPy:

## Fancy Indexing

يعني إنك تجيب عناصر من ال

array

باستخدام ليست من الإنديكسات مش رقم واحد ولا

slice.

الجملة معناها إيه؟

Fancy Indexing: runs for matches 1, 8, 22, 11, and 2

يعني:

للماتشات أرقام: 1 , 8 , 22 , 11 , 2 runs هات عدد الـ

باستخدام Fancy Indexing

مثال عملي 🔥

في كل array بيمثل runs افترض عندك

```
import numpy as np

runs = np.array([45, 60, 10, 80, 33, 25, 70, 90, 15, 40,
                 55, 65, 20, 30, 75, 85, 95, 50, 35, 28,
                 48, 58, 68])
```

للماتشات: 1, 8, 22, 11, 2 عايز تجيب

⚠️ يبدأ العد من 0: خلي بالك NumPy

الحل باستخدام Fancy Indexing

```
matches = [0, 7, 21, 10, 1] # index = match - 1
selected_runs = runs[matches]

print(selected_runs)
```

👉 من الإنديكسات list عشان استخدمنا Fancy Indexing ده اسمه

عادي |----|---- Index | مقارنة سريعة 💡 |----|---- Slicing | runs[3] | Fancy Indexing | runs[2:6] | runs[[0, 7, 21]]

```
In [68]: #45. **Fancy Indexing:** Retrieve runs for matches 1, 8, 22, 11, and 2 simultaneous

# يعني إنك تجيب عناصر من array
# باستخدام ليست من الإنديكسات مش رقم واحد ولا slice
# k1.loc[['Match1', 'Match8', 'Match7', 'Match4', 'Match2']]
```

```
Out[68]: Match1    45
          Match8    54
          Match7    66
          Match4    35
          Match2    78
          Name: Kohli Runs, dtype: int64
```

```
In [81]: #46. Access the Lead actor for 'Evening Shadows' using its **label index**.

lead_actor = pd.Series(
    ['SRK', 'Salman Khan', 'Aamir Khan'],
```

```

        index=['DDLJ', 'Evening Shadows', '3 Idiots']
    )

print(lead_actor)
print()
print()

print(lead_actor.loc['Evening Shadows'])
print()
print(lead_actor.iloc[1])

```

```

DDLJ              SRK
Evening Shadows   Salman Khan
3 Idiots          Aamir Khan
dtype: object

```

Salman Khan

Salman Khan

## loc vs iloc

لو محatar بينهم

- ◆ loc

يعني بتعامل مع اسم الصف أو العمود

- ◆ iloc

يعني بتعامل مع رقم المكان (...2,1,0)

```

In [82]: #47. Update the mark for 'OS' in `marks_series` to 88 using its **index position**.

#Create a Series named marks_series directly from a Python dictionary.

marks_dict = {
    'Robotics':87,
    'Image processing':80,
    'OS':82,
    'Software Engineering':93
}

marks_series = pd.Series(marks_dict, name = 'medo')

#method 1
marks_series["OS"] = 88
#method 2
marks_series.iloc[marks_series.index.get_loc("OS")] = 88

marks_series

```

```
Out[82]: Robotics      87  
          Image processing    80  
          OS                  88  
          Software Engineering 93  
          Name: medo, dtype: int64
```

```
In [83]: #48. **Add a new entry** ('social': 90) to an existing `marks_series`.
```

```
print(marks_series)  
print()  
print()  
marks_series["social"] = 90  
marks_series
```

```
Robotics      87  
Image processing    80  
OS                  88  
Software Engineering 93  
Name: medo, dtype: int64
```

```
Out[83]: Robotics      87  
          Image processing    80  
          OS                  88  
          Software Engineering 93  
          social              90  
          Name: medo, dtype: int64
```

```
In [85]: #49. Update the lead actor for 'Bajrangi Bhaijaan' to 'Jack' using its **index Label
```

```
print(movies)  
  
print()  
print()  
  
movies.loc["Bajrangi Bhaijaan"] = "Jack"  
print(movies)
```

```
DDLJ           SRK
Tiger Zinda Hai   Salman Khan
3 Idiots        Aamir Khan
My Name Is Khan    SRK
Kesari          Akshay Kumar
Don             SRK
Bajrangi Bhaijaan  Salman Khan
Housefull       Akshay Kumar
Name: Lead Actor, dtype: object
```

```
DDLJ           SRK
Tiger Zinda Hai   Salman Khan
3 Idiots        Aamir Khan
My Name Is Khan    SRK
Kesari          Akshay Kumar
Don             SRK
Bajrangi Bhaijaan  Jack
Housefull       Akshay Kumar
Name: Lead Actor, dtype: object
```

```
In [86]: #50 .Find the Length of the sub Series

len(sub)
```

```
Out[86]: 10
```

```
In [88]: #51. Convert `marks_series` into a standard **Python List**.

print(marks_series)
print()
print()
marks_series.tolist()
```

```
Robotics      87
Image processing 80
OS            88
Software Engineering 93
social         90
Name: medo, dtype: int64
```

```
Out[88]: [87, 80, 88, 93, 90]
```

```
In [89]: #52. Convert `marks_series` into a **Python dictionary**.

marks_series.to_dict()
```

```
Out[89]: {'Robotics': 87,
          'Image processing': 80,
          'OS': 88,
          'Software Engineering': 93,
          'social': 90}
```

```
In [90]: #53. Check if 'Jack' exists in the **values** of the `movies` Series using the memb
```

```
"jack" in movies.values  
مُش محتاجة يعني #
```

Out[90]: False

```
In [94]: #54. **Looping:** Iterate through the `movies` Series and print every Lead actor.  
print(movies)  
print()  
print()  
for actor in movies:  
    print(actor)  
  
## this equal of saying  
  
print()  
print()  
print(movies.values) # diff is here in list
```

```
DDJ           SRK  
Tiger Zinda Hai      Salman Khan  
3 Idiots          Aamir Khan  
My Name Is Khan       SRK  
Kesari           Akshay Kumar  
Don              SRK  
Bajrangi Bhaijaan     Jack  
Housefull          Akshay Kumar  
Name: Lead Actor, dtype: object
```

```
SRK  
Salman Khan  
Aamir Khan  
SRK  
Akshay Kumar  
SRK  
Jack  
Akshay Kumar
```

```
['SRK' 'Salman Khan' 'Aamir Khan' 'SRK' 'Akshay Kumar' 'SRK' 'Jack'  
'Akshay Kumar']
```

```
In [95]: #55. **Broadcasting:** Subtract every value in `marks_series` from 100 in one opera  
100 - marks_series
```

```
Out[95]: Robotics      13  
Image processing    20  
OS                  12  
Software Engineering 7  
social               10  
Name: medo, dtype: int64
```

```
In [98]: #56. Use a **relational operator** to create a boolean Series showing where Kohli s
```

```

print(kl >= 50) # this answer of question

#some additions

print()
print()

print(kl[kl>=50]) #print that achieve the condition

```

```

Match1    False
Match2    True
Match3    True
Match4    False
Match5    True
Match6    True
Match7    True
Match8    True
Name: Kohli Runs, dtype: bool

```

```

Match2    78
Match3    102
Match5    89
Match6    120
Match7    66
Match8    54
Name: Kohli Runs, dtype: int64

```

## Section 4: Boolean Indexing and Filtering

note: i will use matplotlib just

try to understand what i do inshallah i will cover matplotlib in next pdf

```

In [111...]: import pandas as pd
           import matplotlib.pyplot as plt

           sub = pd.Series(
               [1200, 1500, 1800, 1700, 1600, 2000, 2200, 2100, 1900, 2300],
               index=["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct"],
               name="Subscribers"
           )

           kl = pd.Series(
               [45, 78, 102, 35, 89, 120, 66, 0],
               index=["Match1", "Match2", "Match3", "Match4", "Match5", "Match6", "Match7", "Match8"],
               name="Kohli Runs"
           )

           movies = pd.Series(
               ["SRK", "Salman", "Aamir", "SRK", "Akshay", "SRK", "Salman", "Akshay", "SRK", "Salman",
                "SRK", "Akshay", "Aamir", "SRK", "Salman", "Akshay", "SRK", "SRK", "Salman", "SRK"],
               index=[f"Movie{i}" for i in range(1, 21)],
               name="Lead Actor"
           )

```

```

)

# # 61. Line graph of sub
# plt.figure()
# sub.plot()
# plt.title("Subscribers Over Time")
# plt.show()

# # 62. Pie chart of top 20 lead actors
# plt.figure()
# movies.value_counts().head(20).plot(kind="pie", autopct="%1.1f%%")
# plt.title("Top 20 Lead Actors")
# plt.show()

# # 63. Bar chart of top 20 lead actors
# plt.figure()
# movies.value_counts().head(20).plot(kind="bar")
# plt.title("Top 20 Lead Actors")
# plt.show()

```

In [113...]

```
#57. Filter the `kl` Series to show only scores **greater than or equal to 50**.
```

```
kl[kl>=50]
```

Out[113...]

	Match2	Match3	Match5	Match6	Match7
	78	102	89	120	66

```
Name: Kohli Runs, dtype: int64
```

In [121...]

```
#58. Count how many times Kohli scored **0 runs (ducks)**.
```

```
#method 1
print((kl==0).sum())
print()
#method 2
print(kl.value_counts()[0])
print()
#method 3
print(kl.loc[kl == 0])
print()
```

```
kl == 0
```

1

1

```
Match8    0
Name: Kohli Runs, dtype: int64
```

```
Out[121...]: Match1    False  
Match2    False  
Match3    False  
Match4    False  
Match5    False  
Match6    False  
Match7    False  
Match8    True  
Name: Kohli Runs, dtype: bool
```

```
In [137...]: #59. Filter the `sub` Series for days with **more than 200 subscribers** and find the count of such days.  
print(sub)  
print()  
print('*****')  
  
print('more than 200 subscribers: ',(sub > 2000).sum())  
print()  
print()  
  
print('more than 200 subscribers: ',sub.loc[sub > 2000].count())
```

```
Jan      1200  
Feb      1500  
Mar      1800  
Apr      1700  
May      1600  
Jun      2000  
Jul      2200  
Aug      2100  
Sep      1900  
Oct      2300  
Name: Subscribers, dtype: int64
```

```
*****  
more than 200 subscribers:  3
```

```
more than 200 subscribers:  3
```

```
In [153...]: #60. Find actors in the `movies` Series who have appeared in **more than 4 movies**  
  
print(movies)  
  
print('-----')  
print()  
print()  
  
print()  
print()  
#first i want to know each actor and number of movies he act in  
print(movies.value_counts())  
print()  
#now we will get actors that acted more than 4 movies  
print(movies.value_counts()[movies.value_counts() > 4])
```

```
Movie1      SRK
Movie2      Salman
Movie3      Aamir
Movie4      SRK
Movie5      Akshay
Movie6      SRK
Movie7      Salman
Movie8      Akshay
Movie9      SRK
Movie10     Salman
Movie11     SRK
Movie12     Akshay
Movie13     Aamir
Movie14     SRK
Movie15     Salman
Movie16     Akshay
Movie17     SRK
Movie18     SRK
Movie19     Salman
Movie20     SRK
Name: Lead Actor, dtype: object
```

---

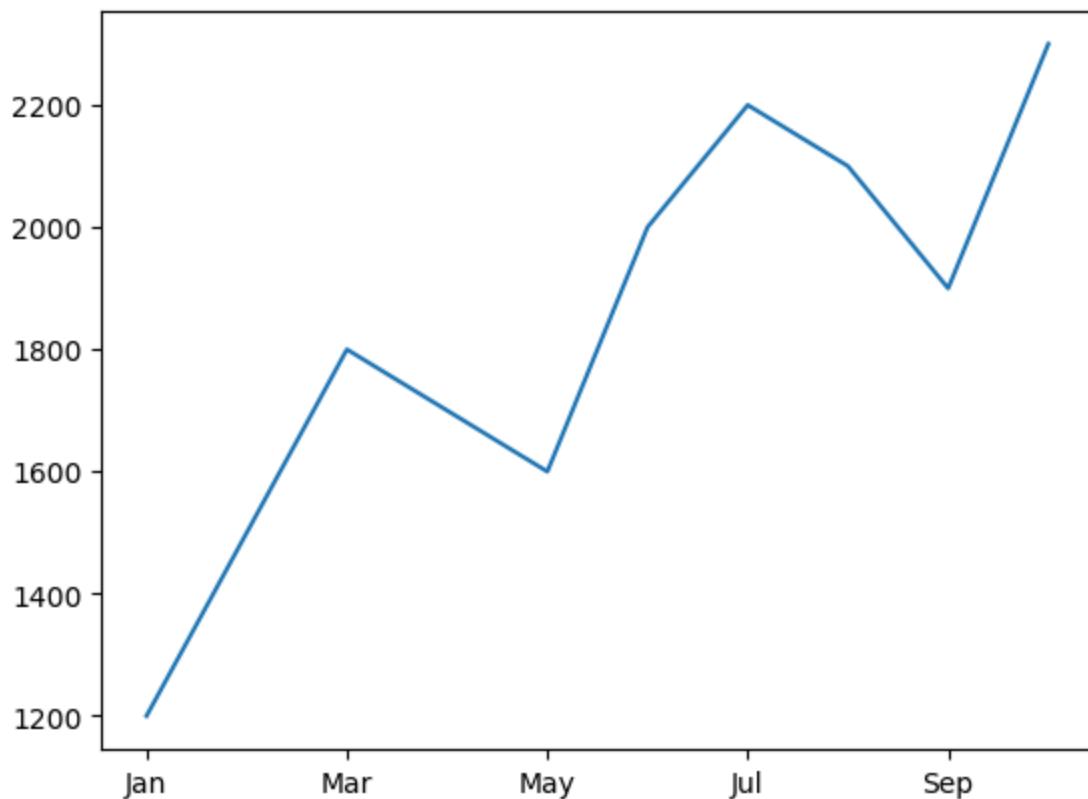
```
Lead Actor
SRK      9
Salman   5
Akshay   4
Aamir   2
Name: count, dtype: int64
```

```
Lead Actor
SRK      9
Salman   5
Name: count, dtype: int64
```

```
In [110...]: #61. Plot a **line graph** of the `sub` Series.
```

```
# 61. Line graph of sub
plt.figure()
sub.plot()
plt.title("Subscribers Over Time")
plt.show()
```

## Subscribers Over Time

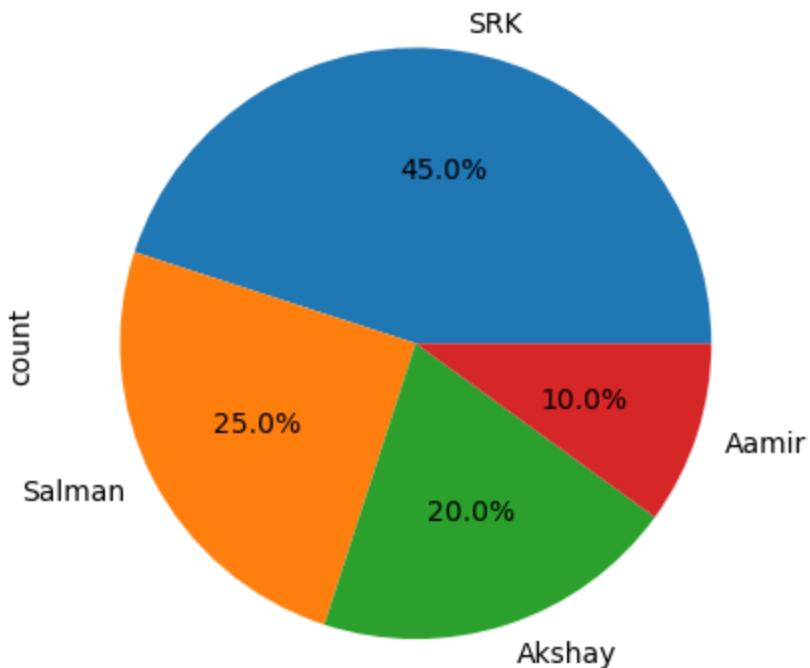


In [109]:

```
#62. Plot a **pie chart** of the top 20 lead actors.
```

```
# 62. Pie chart of top 20 lead actors
plt.figure()
movies.value_counts().head(20).plot(kind="pie", autopct="%1.1f%%")
plt.title("Top 20 Lead Actors")
plt.show()
```

## Top 20 Lead Actors

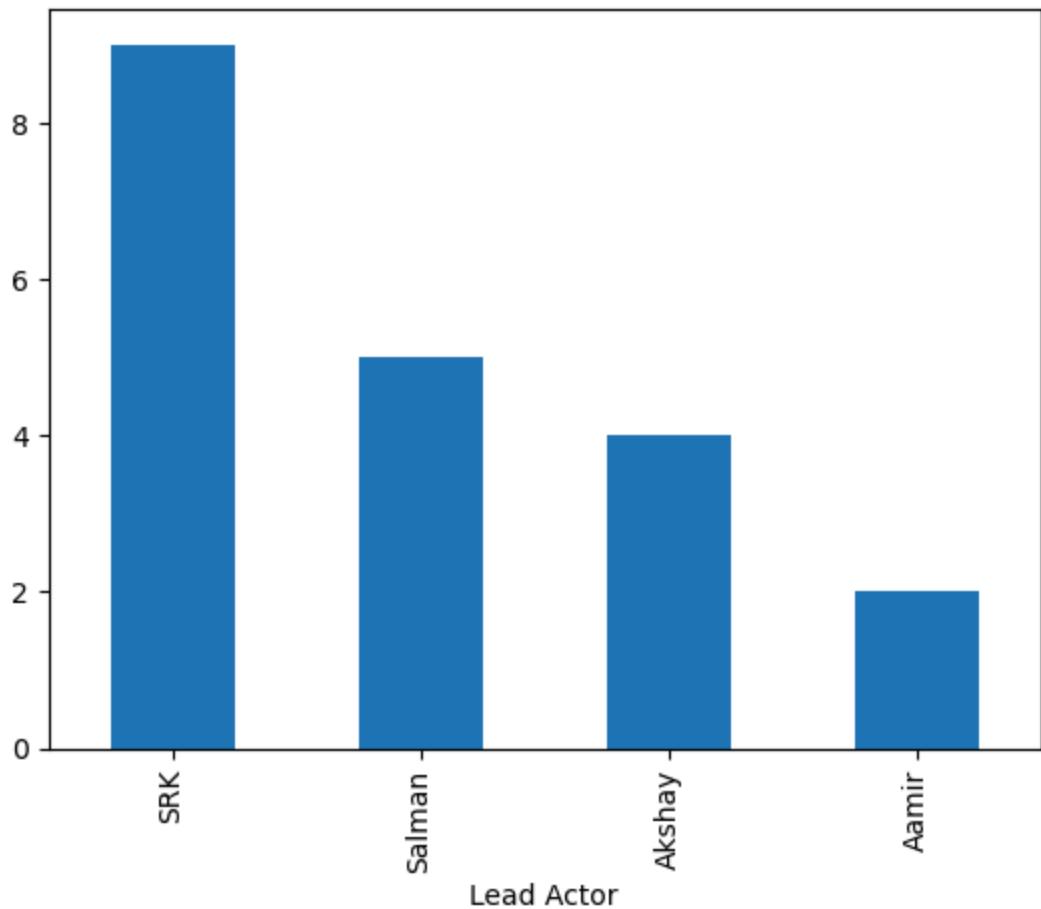


In [108]:

```
#63. Plot a **bar chart** of the top 20 lead actors.

# 63. Bar chart of top 20 lead actors
plt.figure()
movies.value_counts().head(20).plot(kind="bar")
plt.title("Top 20 Lead Actors")
plt.show()
```

Top 20 Lead Actors



In [ ]: #

## Section 5: Advanced Cleaning and Manipulation

In [154...]: #64. Use `sys.getsizeof()` to check the memory usage of the `kl` Series.

```
import sys  
sys.getsizeof(kl)
```

Out[154...]: 836

In [158...]: #65. \*\*Type Conversion:\*\* Convert the `kl` Series to `int16` to save memory.

```
kl_float64 = kl.astype("float64")  
kl_float64
```

```
Out[158...]: Match1    45.0
Match2    78.0
Match3   102.0
Match4    35.0
Match5    89.0
Match6   120.0
Match7    66.0
Match8     0.0
Name: Kohli Runs, dtype: float64
```

```
In [166...]: #66. Use **`between()`** to find all Kohli scores between 45 and 90.
print(kl)
print()
print()
kl[kl.between(45,90)]
```

```
Match1    45
Match2    78
Match3   102
Match4    35
Match5    89
Match6   120
Match7    66
Match8     0
Name: Kohli Runs, dtype: int64
```

```
Out[166...]: Match1    45
Match2    78
Match5    89
Match7    66
Name: Kohli Runs, dtype: int64
```

```
In [170...]: #67. Use **`clip()`** to cap the `sub` Series values between 100 and 200.
# يعني:
# أي قيمة أقل من 100 → تتعذر وتبقى 100
# أي قيمة أكبر من 200 → تتعذر وتبقى 200
# القيم بين 100 و 200 → تفضل زي ما هي

print(sub)
print()
print()
print()

sub.clip(1800,2200)
```

```
Jan    1200
Feb    1500
Mar    1800
Apr    1700
May    1600
Jun    2000
Jul    2200
Aug    2100
Sep    1900
Oct    2300
Name: Subscribers, dtype: int64
```

```
Out[170...]: Jan    1800
              Feb    1800
              Mar    1800
              Apr    1800
              May    1800
              Jun    2000
              Jul    2200
              Aug    2100
              Sep    1900
              Oct    2200
Name: Subscribers, dtype: int64
```

```
In [175...]: #68. Remove **duplicate values** from a Series while keeping the first occurrence.
```

```
print(movies)
print()
print()
print()
mv = movies.drop_duplicates()

mv
```

```
Movie1      SRK
Movie2      Salman
Movie3      Aamir
Movie4      SRK
Movie5      Akshay
Movie6      SRK
Movie7      Salman
Movie8      Akshay
Movie9      SRK
Movie10     Salman
Movie11     SRK
Movie12     Akshay
Movie13     Aamir
Movie14     SRK
Movie15     Salman
Movie16     Akshay
Movie17     SRK
Movie18     SRK
Movie19     Salman
Movie20     SRK
Name: Lead Actor, dtype: object
```

```
Out[175... Movie1      SRK
Movie2      Salman
Movie3      Aamir
Movie5      Akshay
Name: Lead Actor, dtype: object
```

```
In [176... #69. Remove duplicates but **keep the last occurrence**.
# لو القيمة اتكررت أكثر من مرة #
# امسح التكرار #
# وسّيّب آخر مرة ظهرت فيها القيمة #

print(movies)
print()
print()
print()
mv = movies.drop_duplicates(keep='last')

mv
```

```
Movie1      SRK
Movie2      Salman
Movie3      Aamir
Movie4      SRK
Movie5      Akshay
Movie6      SRK
Movie7      Salman
Movie8      Akshay
Movie9      SRK
Movie10     Salman
Movie11     SRK
Movie12     Akshay
Movie13     Aamir
Movie14     SRK
Movie15     Salman
Movie16     Akshay
Movie17     SRK
Movie18     SRK
Movie19     Salman
Movie20     SRK
Name: Lead Actor, dtype: object
```

```
Out[176... Movie13    Aamir
Movie16    Akshay
Movie19    Salman
Movie20    SRK
Name: Lead Actor, dtype: object
```

```
In [186... #70. Use **`duplicated().sum()`** to find the total number of duplicate values in `

sr = pd.Series(np.random.randint(1,10,15),index = [f'f{i}' for i in range(15)])`


print(sr)
print()
print()

print(sr[sr.duplicated()])
print()
sr.duplicated().sum()
```

```
f0      4
f1      5
f2      2
f3      7
f4      2
f5      5
f6      8
f7      2
f8      8
f9      6
f10     9
f11     8
f12     5
f13     9
f14     7
dtype: int32
```

```
f4      2
f5      5
f7      2
f8      8
f11     8
f12     5
f13     9
f14     7
dtype: int32
```

```
Out[186... np.int64(8)
```

```
In [187... #71. Use **`isnull().sum()`** to find the total count of missing values.
```

```
k1.isnull().sum()
```

```
Out[187... np.int64(0)
```

```
In [188... #72. Remove all rows with **missing values** using `dropna()`.
```

```
k1.dropna()
```

```
Out[188... Match1    45
Match2    78
Match3   102
Match4    35
Match5    89
Match6   120
Match7    66
Match8     0
Name: Kohli Runs, dtype: int64
```

```
In [189... #73. Replace missing values with **0*** using `fillna()`.
```

```
k1.fillna(k1.mean())
```

```
Out[189...]: Match1    45  
Match2     78  
Match3   102  
Match4     35  
Match5     89  
Match6   120  
Match7     66  
Match8      0  
Name: Kohli Runs, dtype: int64
```

```
In [190...]: #74. Replace missing values with the **mean of the Series**.
```

```
kl.fillna(kl.mean())
```

```
Out[190...]: Match1    45  
Match2     78  
Match3   102  
Match4     35  
Match5     89  
Match6   120  
Match7     66  
Match8      0  
Name: Kohli Runs, dtype: int64
```

```
In [192...]: #75. Use **`isin()`** to check if Kohli scored exactly 49 or 99.
```

```
kl[kl.isin([49,99])]
```

```
Out[192...]: Series([], Name: Kohli Runs, dtype: int64)
```

---

## المعنى العام

◆ **map**

واحدة عنصر عنصر **Series** تحويل

◆ **apply**

على **function** تشغيل:

- Series
  - أو DataFrame (rows / columns)
- 

**map**



تستخدم إمتنى؟

✓ لما:

- شغال على **Series** فقط
- كل عنصر لوحده
- transformation بسيط

مثال:

```
s = pd.Series([1, 2, 3])
```

```
s.map(lambda x: x * 2)
```

الناتج:

```
2 4 6
```

كمان **map** ينفع مع **dict**

```
s.map({1: 'A', 2: 'B', 3: 'C'})
```

**apply** 

تستخدم إمتنى؟

✓ لما:

- أعقد **logic** تحتاج
- أو شغال على **DataFrame**
- كاملة **row** أو تحتاج تتعامل مع

على **Series**

```
s.apply(lambda x: x * 2)
```

نفس نتائج **map**

على **DataFrame**

```
df.apply(sum)      # column-wise (default)  
df.apply(sum, axis=1) # row-wise
```

فرق مهم جدًا 

**apply** مع **DataFrame**

- `axis=0` → أعمدة
  - `axis=1` → صفوف
- 

## مقارنة سريعة 🔥

المقارنة	map	apply
يشتغل على Series	✓	✓
يشتغل على DataFrame	✗	✓
يقبل dict	✓	✗
أسرع	✓	✗
معقد	✗	✓

---

## متى أستخدم إيه؟

- Series → بسيطة → `map`
  - استبدال قيم → `map`
  - Row logic / conditions → `apply`
  - DataFrame → `apply`
- 

## خطأ شائع ✗

`df.map(...)` # خطأ

---

```
In [198...]: #76. Use `apply()` with a Lambda function to split actor names into lists.  
#movies = pd.concat([movies,pd.Series(["medo Amir","ezz aldin"],index=["Movies 21",  
movies.apply(lambda x: x.split())
```

```
Out[198...]: Movie1      [SRK]
          Movie2      [Salman]
          Movie3     [Aamir]
          Movie4      [SRK]
          Movie5     [Akshay]
          Movie6      [SRK]
          Movie7     [Salman]
          Movie8     [Akshay]
          Movie9      [SRK]
          Movie10    [Salman]
          Movie11    [SRK]
          Movie12    [Akshay]
          Movie13    [Aamir]
          Movie14    [SRK]
          Movie15    [Salman]
          Movie16    [Akshay]
          Movie17    [SRK]
          Movie18    [SRK]
          Movie19    [Salman]
          Movie20    [SRK]
          Movies 21  [medo, Amir]
          Movies 22  [ezz, aldin]
          Movies 21  [medo, Amir]
          Movies 22  [ezz, aldin]
          dtype: object
```

In [199...]: #77. Use `apply()` to extract only the \*\*first name\*\* of every lead actor.

```
movies.apply(lambda x: x.split()[0])
```

```
Out[199...]: Movie1      SRK
          Movie2      Salman
          Movie3     Aamir
          Movie4      SRK
          Movie5     Akshay
          Movie6      SRK
          Movie7     Salman
          Movie8     Akshay
          Movie9      SRK
          Movie10    Salman
          Movie11    SRK
          Movie12    Akshay
          Movie13    Aamir
          Movie14    SRK
          Movie15    Salman
          Movie16    Akshay
          Movie17    SRK
          Movie18    SRK
          Movie19    Salman
          Movie20    SRK
          Movies 21  medo
          Movies 22  ezz
          Movies 21  medo
          Movies 22  ezz
          dtype: object
```

```
In [200...]: #78. Convert all actor names to **uppercase** using `apply()`.
```

```
movies.apply(lambda x: x.upper())
```

```
Out[200...]:
```

Movie1	SRK
Movie2	SALMAN
Movie3	AAMIR
Movie4	SRK
Movie5	AKSHAY
Movie6	SRK
Movie7	SALMAN
Movie8	AKSHAY
Movie9	SRK
Movie10	SALMAN
Movie11	SRK
Movie12	AKSHAY
Movie13	AAMIR
Movie14	SRK
Movie15	SALMAN
Movie16	AKSHAY
Movie17	SRK
Movie18	SRK
Movie19	SALMAN
Movie20	SRK
Movies 21	MEDO AMIR
Movies 22	EZZ ALDIN
Movies 21	MEDO AMIR
Movies 22	EZZ ALDIN

dtype: object

```
In [205...]: #79. Apply a conditional Lambda to `sub` to label days as **'good day' or 'bad day'
```

```
print(sub)
print()
print()
print("mean: ", sub.mean())

sub.apply(lambda x: "good" if x > sub.mean() else "bad")
```

```
Jan    1200
Feb    1500
Mar    1800
Apr    1700
May    1600
Jun    2000
Jul    2200
Aug    2100
Sep    1900
Oct    2300
Name: Subscribers, dtype: int64
```

```
mean: 1830.0
```

```
Out[205...]:
```

Jan	bad
Feb	bad
Mar	bad
Apr	bad
May	bad
Jun	good
Jul	good
Aug	good
Sep	good
Oct	good

Name: Subscribers, dtype: object

```
In [207...]: #80. Create a **deep copy** of the first 5 rows of `k1` to avoid `SettingWithCopy`
```

```
k1_copy = k1.head().copy(deep=True)  
k1_copy
```

```
Out[207...]:
```

Match1	45
Match2	78
Match3	102
Match4	35
Match5	89

Name: Kohli Runs, dtype: int64

### deep copy vs shallow copy

---

## الأول: المشكلة أصلًا

لما تكتب في Pandas:

```
k1_head = k1.head()  
k1_head[0] = 999  
Pandas:
```

ف يطلع Copy View ولا هو ده 😳 ؟ SettingWithCopyWarning

---

## Shallow Copy (نسخة سطحية)

```
k1_copy = k1.head()
```

### بيعمل إيه؟

- ممكن يشاور على نفس البيانات في الذاكرة
- أي تعديل ممكن:
  - يأثر على k1
  - أو لا

- سلوك غير مضمون ❌
- 

## Deep Copy (نسخة عميقه) ✅

```
k1_copy = k1.head().copy(deep=True)
```

### ب يعمل إيه؟

- ينسخ البيانات نفسها
  - ينسخ الـ index
  - أي تعديل:
    - مش هيأثر على k1
    - ومفيش warnings 🎉
- 

## الفرق بالمثال 🔥

```
k1 = pd.Series([10, 20, 30, 40, 50, 60])

# shallow
a = k1.head()
a.iloc[0] = 999

# deep
b = k1.head().copy()
b.iloc[0] = 888
```

### النتيجة:

- k1 ممكن يتغير بسبب a ❌
  - k1 مش هيتغير بسبب b ✅
- 

## Mهم؟ SettingWithCopyWarning ليه

Pandas: بيحذرك:

"View ولا Copy أنا مش متأكد إنت بتعدل في"

والحل الصح دائمًا:

```
.copy()
```

---

## مقارنة سريعة

المقارنة	Shallow	Deep
نفس الذاكرة	ممكн	✗
آمن للتعديل	✗	✓
SettingWithCopy	ممكн	✗
استخدامه	نادر	الأفضل

## الخلاصة الذهبية

Deep Copy `copy(deep=True)` = راحة بال 😎

ال코드 بتاعك:

`k1_copy = k1.head().copy(deep=True)`  
✓ احترافي ✓ بدون ✓ صح warnings

In [ ]:

## Section 6: DataFrame Operations

81. Create a **DataFrame from a list of lists** with columns 'iq', 'marks', and 'package'.
82. Create a DataFrame from a **dictionary of lists**.
83. **Set the index** of a DataFrame to the 'name' column permanently.
84. **Hint: Use `movies.csv` and `ipl-matches.csv`**. Read both into DataFrames and check their **shapes**.
85. List the **data types** of all columns in the `movies` DataFrame.
86. List all **column names** in the `ipl` DataFrame.
87. View a **random sample of 2 rows** from the `ipl` DataFrame.
88. Use `.info()` to get a summary of the `movies` DataFrame.
89. Count **missing values** in every column of the `movies` DataFrame.
90. **Rename** columns 'marks' to 'percent' and 'package' to 'ipa'.
91. Calculate the **row-wise sum** of a student DataFrame.
92. Select a **single column** ('title\_x') from the `movies` DataFrame.
93. Select **multiple columns** at once from the `movies` DataFrame.
94. Select the **second row** of the `movies` DataFrame using `iloc`.
95. Use `loc` to select a row by the index label 'parle'.
96. Select **rows 5 to 10** and the **first 3 columns** simultaneously using `iloc`.
97. **Filter** the `ipl` DataFrame to find all matches where 'MatchNumber' is 'Final'.
98. Find matches where 'City' is 'Kolkata' **AND** 'WinningTeam' is 'Chennai Super Kings'.
99. **Add a new column** 'country' to `movies` and set it to 'India'.

100. Find the genre with the highest average IMDB rating using `groupby`, `mean`, and `sort_values`.

```
In [211... import pandas as pd

# **** 81. DataFrame from list of lists ****
students_df = pd.DataFrame([[110, 85, 12], [120, 90, 15], [100, 78, 10]],
                           columns=["iq", "marks", "package"])
print("81. Students DataFrame from list of lists:\n", students_df, "\n")
```

81. Students DataFrame from list of lists:

	iq	marks	package
0	110	85	12
1	120	90	15
2	100	78	10

```
In [212... # **** 82. DataFrame from dictionary of lists ****
data = {"name": ["Ali", "Sara", "Omar"], "marks": [80, 90, 85], "package": [10, 15,
df = pd.DataFrame(data)
print("82. DataFrame from dictionary of lists:\n", df, "\n")
```

82. DataFrame from dictionary of lists:

	name	marks	package
0	Ali	80	10
1	Sara	90	15
2	Omar	85	12

```
In [213... # **** 83. Set 'name' as index permanently ****
df.set_index("name", inplace=True)
print("83. DataFrame with 'name' as index:\n", df, "\n")
```

83. DataFrame with 'name' as index:

	marks	package
name		
Ali	80	10
Sara	90	15
Omar	85	12

```
In [214... # **** 84. Mock movies and ipl DataFrames ****
movies = pd.DataFrame({
    "title_x": ["Movie1", "Movie2", "Movie3", "Movie4", "Movie5"],
    "imdb_rating": [7.8, 8.2, 6.5, 9.0, 7.0],
    "genre": ["Action", "Drama", "Action", "Comedy", "Drama"]
})
ipl = pd.DataFrame({
    "MatchNumber": ["1", "Final", "2", "3", "Final", "6"],
    "City": ["Kolkata", "Kolkata", "Mumbai", "Chennai", "Kolkata", "Delhi"],
    "WinningTeam": ["TeamA", "Chennai Super Kings", "TeamB", "TeamC", "Chennai Supe
})
print("84. Shape of movies:", movies.shape)
print("     Shape of ipl:", ipl.shape, "\n")
```

```
84. Shape of movies: (5, 3)
    Shape of ipl: (6, 3)
```

```
In [215... # **** 85. Data types of movies ****
print("85. Data types:\n", movies.dtypes, "\n")
```

```
85. Data types:
  title_x      object
  imdb_rating   float64
  genre        object
  dtype: object
```

```
In [216... # **** 86. Column names in ipl ****
print("86. Column names in ipl:\n", ipl.columns.tolist(), "\n")
```

```
86. Column names in ipl:
['MatchNumber', 'City', 'WinningTeam']
```

```
In [217... # **** 87. Random sample 2 rows from ipl ****
print("87. Random sample 2 rows from ipl:\n", ipl.sample(2), "\n")
```

```
87. Random sample 2 rows from ipl:
  MatchNumber      City      WinningTeam
4       Final    Kolkata  Chennai Super Kings
3           3    Chennai             TeamC
```

```
In [218... # **** 88. Info of movies ****
print("88. Info of movies:")
movies.info()
print("\n")
```

```
88. Info of movies:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
---  -- 
 0   title_x      5 non-null      object 
 1   imdb_rating   5 non-null      float64
 2   genre        5 non-null      object 
dtypes: float64(1), object(2)
memory usage: 252.0+ bytes
```

```
In [219... # **** 89. Count missing values ****
print("89. Missing values:\n", movies.isnull().sum(), "\n")
```

```
89. Missing values:
  title_x      0
  imdb_rating   0
  genre        0
  dtype: int64
```

```
In [220... # **** 90. Rename columns in students_df ****  
students_df.rename(columns={"marks": "percent", "package": "lpa"}, inplace=True)  
print("90. Students DataFrame after renaming:\n", students_df, "\n")
```

90. Students DataFrame after renaming:

	iq	percent	lpa
0	110	85	12
1	120	90	15
2	100	78	10

```
In [221... # **** 91. Row-wise sum of students_df ****  
print("91. Row-wise sum:\n", students_df.sum(axis=1), "\n")
```

91. Row-wise sum:

0	207
1	225
2	188

dtype: int64

```
In [222... # **** 92. Select single column 'title_x' ****  
print("92. 'title_x' column:\n", movies["title_x"], "\n")
```

92. 'title\_x' column:

0	Movie1
1	Movie2
2	Movie3
3	Movie4
4	Movie5

Name: title\_x, dtype: object

```
In [223... # **** 93. Select multiple columns ****  
print("93. ['title_x','imdb_rating','genre'] columns:\n", movies[["title_x","imdb_r
```

93. ['title\_x','imdb\_rating','genre'] columns:

	title_x	imdb_rating	genre
0	Movie1	7.8	Action
1	Movie2	8.2	Drama
2	Movie3	6.5	Action
3	Movie4	9.0	Comedy
4	Movie5	7.0	Drama

```
In [224... # **** 94. Second row using iloc ****  
print("94. Second row:\n", movies.iloc[1], "\n")
```

94. Second row:

title_x	Movie2
imdb_rating	8.2
genre	Drama

Name: 1, dtype: object

```
In [225... # **** 95. loc with index 'parle' (mock index) ****  
movies_with_index = movies.copy()
```

```
movies_with_index.index = ["film1","film2","film3","parle","film5"]
print("95. Row with index 'parle':\n", movies_with_index.loc["parle"], "\n")
```

```
95. Row with index 'parle':
    title_x      Movie4
    imdb_rating   9.0
    genre        Comedy
Name: parle, dtype: object
```

```
In [226...]: # **** 96. Rows 5-10 & first 3 columns using iloc (only 5 rows here) ****
print("96. Rows 0-4 and first 3 columns:\n", movies.iloc[0:5, 0:3], "\n")
```

```
96. Rows 0-4 and first 3 columns:
    title_x  imdb_rating  genre
0  Movie1          7.8  Action
1  Movie2          8.2  Drama
2  Movie3          6.5  Action
3  Movie4          9.0  Comedy
4  Movie5          7.0  Drama
```

```
In [227...]: # **** 97. IPL matches where MatchNumber == 'Final' ****
print("97. IPL matches with MatchNumber 'Final':\n", ipl[ipl["MatchNumber"]=="Final"])
```

```
97. IPL matches with MatchNumber 'Final':
    MatchNumber      City           WinningTeam
1        Final    Kolkata  Chennai Super Kings
4        Final    Kolkata  Chennai Super Kings
```

```
In [230...]: # **** 98. Matches where City is Kolkata AND WinningTeam is Chennai Super Kings ***
print("98. Kolkata & Chennai Super Kings wins:\n", ipl[(ipl["City"]=="Kolkata") &
    (ipl["WinningTeam"]=="Chennai Super Kings")], "\n")
```

```
98. Kolkata & Chennai Super Kings wins:
    MatchNumber      City           WinningTeam
1        Final    Kolkata  Chennai Super Kings
4        Final    Kolkata  Chennai Super Kings
```

```
In [229...]: # **** 99. Add 'country' column to movies ****
movies["country"] = "India"
print("99. Movies with country column:\n", movies[["title_x","country"]], "\n")
```

```
99. Movies with country column:
    title_x  country
0  Movie1    India
1  Movie2    India
2  Movie3    India
3  Movie4    India
4  Movie5    India
```

```
In [ ]:
```

```
In [ ]:
```

