Pandas Cheat Sheet (Theory Only)

---

1. Introduction to Pandas Objects

Pandas is a Python library for **data analysis and manipulation**.
Provides **two main data structures**:

- **Series** → One-dimensional array with labeled index.
- **DataFrame** → Two-dimensional table with rows and columns.

Importing Pandas:

```
import pandas as pd
```

Creating a Series:

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

Creating a DataFrame:

```
data = {'Name': ['Alice', 'Bob'], 'Age': [25, 30]}
df = pd.DataFrame(data)
print(df)
```

---

2. Data Indexing & Selection

Selecting a Column:

```
print(df["Name"])  # Output: Series of names
```

Selecting a Row by Label:

```
print(df.loc[0])  # Output: First row as Series
```

Selecting a Row by Index:

```
print(df.iloc[1])  # Output: Second row as Series
```

Boolean Masking:

```
print(df[df["Age"] > 25])  # Filters rows where Age > 25
```

---

3. Operating on Data in Pandas

Mathematical Operations:

```
df["Age"] = df["Age"] + 5  # Adds 5 to each age
```

Applying Functions:

```
df["Age"] = df["Age"].apply(lambda x: x * 2)  # Multiply all ages by 2
```

Statistical Operations:

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

---

4. Handling Missing Data

Identifying Missing Values:

```
print(df.isnull())  # Returns True for missing values
```

Filling Missing Values:

```
df.fillna(value=0, inplace=True)  # Replace NaN with 0
```

Dropping Missing Values:

```
df.dropna(inplace=True)  # Removes rows with NaN values
```

---

5. Hierarchical Indexing

Multi-level index for better data organization.
Example:

```
arrays = [['A', 'A', 'B', 'B'], ['One', 'Two', 'One', 'Two']]
index = pd.MultiIndex.from_tuples(list(zip(*arrays)))

df = pd.DataFrame({'Values': [10, 20, 30, 40]}, index=index)
print(df)
```

Accessing Elements:

```
print(df.loc["A"])  # Retrieves all rows under "A"
```

---

6. Combining Data Sets

Concatenation (Stacking DataFrames Together):

```
df1 = pd.DataFrame({"A": [1, 2]}, index=[0, 1])
df2 = pd.DataFrame({"A": [3, 4]}, index=[2, 3])

df_combined = pd.concat([df1, df2])
print(df_combined)
```

Merging (SQL-style Join Operations):

```
df1 = pd.DataFrame({"ID": [1, 2], "Name": ["Alice", "Bob"]})
df2 = pd.DataFrame({"ID": [1, 2], "Age": [25, 30]})

df_merged = pd.merge(df1, df2, on="ID")
print(df_merged)
```

---

7. Aggregation & Grouping

Grouping Data:

```
df.groupby("Category").mean()  # Computes mean for each category
```

Applying Aggregation Functions:

```
df.groupby("Category").agg(["sum", "max"])
```

---

8. Pivot Tables

Creating a Pivot Table:

```
df.pivot_table(values="Sales", index="Region", columns="Year", aggfunc="sum")
```

---

9. Vectorized String Operations

String Operations on DataFrame Columns:

```
df["Name"] = df["Name"].str.upper()  # Convert names to uppercase
```

Filtering with String Methods:

```
df[df["Name"].str.contains("A")]  # Selects rows where "Name" contains 'A'
```

---

10. Working with Time Series

Parsing Dates:

```
df["Date"] = pd.to_datetime(df["Date"])
```

Setting Date as Index:

```
df.set_index("Date", inplace=True)
```

Resampling Data (Aggregating Over Time):

```
df.resample("M").sum()  # Monthly aggregation
```

---

11. High-Performance Pandas: eval() & query()

Using eval() for Efficient Computation:

```
df["Total"] = df.eval("Sales * 1.1")  # Calculates with 10% increase
```

Using query() for Fast Filtering:

```
filtered_df = df.query("Sales > 5000")  # Faster than df[df["Sales"] > 5000]
```

Key Takeaways

Pandas provides flexible data structures (Series, DataFrame).
Indexing & Selection → Allows accessing rows and columns efficiently.
Data Manipulation → Supports operations like filtering, sorting, and transformations.
Handling Missing Data → Use fillna() and dropna().
Combining Datasets → Use concat(), merge(), and join().
Aggregation & Grouping → Summarize data efficiently with groupby() and pivot_table().
High-Performance Pandas → eval() and query() improve efficiency in large datasets.

This **Pandas Cheat Sheet** covers **data structures, indexing, data operations, handling missing values, hierarchical indexing, merging, grouping, pivot tables, time series, and high-performance features**. Let me know if you need further explanations!