

pandas Quick Reference

DataFrames, selection, aggregation, merging, and more

DataFrames

Creating DataFrames

```
import pandas as pd
df = pd.DataFrame({
    "name": ["Alice", "Bob", "Carol"],
    "age": [25, 30, 35],
    "score": [88, 92, 79]
})
```

Inspection

df.head(n)	First n rows (default 5)
df.tail(n)	Last n rows
df.shape	Tuple of (rows, columns)
df.dtypes	Data type of each column
df.info()	Column types, non-null counts
df.describe()	Statistics for numeric columns
df.columns	Column names as Index
df.index	Row labels

Reading Data

Common Readers

```
df = pd.read_csv("data.csv")
df = pd.read_excel("data.xlsx")
df = pd.read_json("data.json")
df = pd.read_sql(query, connection)
```

Writing Data

```
df.to_csv("out.csv", index=False)
df.to_excel("out.xlsx", index=False)
df.to_json("out.json", orient="records")
```

Read Options

sep=";"	Custom delimiter
header=None	No header row in file
usecols=[0, 2]	Read specific columns only
nrows=100	Read first 100 rows
na_values=["N/A"]	Treat as NaN

Selection

Columns

```
df["name"] # single column (Series)
df[["name", "age"]] # multiple columns (DataFrame)
df.name # attribute access (simple names)
```

Rows with loc / iloc

```
df.loc[0] # row by label
df.loc[0:2, "name"] # rows 0-2, column "name"
df.iloc[0] # row by position
df.iloc[0:2, 0:2] # first 2 rows, 2 cols
```

loc vs iloc

df.loc[row, col]	Select by **label** (inclusive end)
df.iloc[row, col]	Select by **position** (exclusive end)
df.at[row, col]	Fast scalar access by label
df.iat[row, col]	Fast scalar access by position

Filtering

Boolean Filtering

```
df[df["age"] > 25]
df[df["name"].str.contains("li")]
df[(df["age"] > 25) & (df["score"] > 80)]
df[df["name"].isin(["Alice", "Bob"])]
```

Handling Missing Data

```
df.isna().sum() # NaN count per column
df.dropna() # drop rows with any NaN
df.fillna(0) # fill NaN with 0
df["col"].fillna(df["col"].mean())
```

Sorting

```
df.sort_values("age") # ascending
df.sort_values("age", ascending=False)
df.sort_values(["age", "score"]) # multi
```

Aggregation

Common Aggregations

```
df["col"].sum() Sum of column
df["col"].mean() Mean
df["col"].median() Median
df["col"].std() Standard deviation
df["col"].min() / .max() Min / max
df["col"].count() Non-null count
df["col"].nunique() Number of unique values
df["col"].value_counts() Frequency of each value
```

Multiple Aggregations

```
df.agg({"age": "mean", "score": ["min", "max"]})
df.describe() # summary stats for all numeric
```

GroupBy

Basic Grouping

```
df.groupby("dept")["salary"].mean()
df.groupby("dept").agg(
    avg_sal=("salary", "mean"),
    count=("salary", "count")
)
```

Multiple Groups

```
df.groupby(["dept", "year"])["sales"].sum()
df.groupby("dept").size() # rows per group
```

Transform & Apply

```
df["z_score"] = df.groupby("dept")["salary"] \
    .transform(lambda x: (x - x.mean()) / x.std())
df.groupby("dept").apply(lambda g: g.nlargest(3, "salary"))
```

Merging

Merge (SQL-style Join)

```
pd.merge(df1, df2, on="id") # inner
pd.merge(df1, df2, on="id", how="left")
pd.merge(df1, df2, left_on="uid",
    right_on="user_id")
```

Join Types

how="inner"	Keep only matching rows (default)
how="left"	Keep all left rows, NaN for no match
how="right"	Keep all right rows
how="outer"	Keep all rows from both sides

Concatenation

```
pd.concat([df1, df2]) # stack rows
pd.concat([df1, df2], axis=1) # side by side
pd.concat([df1, df2], ignore_index=True)
```

Pivot Tables

Pivot Table

```
df.pivot_table(
    values="sales", index="region",
    columns="quarter", aggfunc="sum"
)
```

Reshaping

```
df.melt(id_vars=["name"],
    value_vars=["q1", "q2"],
    var_name="quarter", value_name="sales")
```

Cross Tabulation

```
pd.crosstab(df["dept"], df["gender"])
pd.crosstab(df["dept"], df["gender"],
    normalize="index") # row percentages
```

Time Series

DateTime Basics

```
df["date"] = pd.to_datetime(df["date"])
df["year"] = df["date"].dt.year
df["month"] = df["date"].dt.month
df["weekday"] = df["date"].dt.day_name()
```

Date Ranges & Resampling

```
pd.date_range("2025-01-01", periods=12, freq="ME")
df.set_index("date").resample("ME")["sales"].sum()
```

freq="ME" requires pandas ≥ 2.2. Use "M" on older versions.

Accessor Attributes

.dt.year / .dt.month / .dt.day	Extract date components
.dt.hour / .dt.minute	Extract time components
.dt.day_name()	Weekday name (Monday, etc.)
.dt.days_in_month	Days in that month

Common Patterns

Rename Columns

```
df.rename(columns={"old": "new"})
df.columns = ["a", "b", "c"] # replace all
```

Add / Modify Columns

```
df["total"] = df["q1"] + df["q2"]
df["grade"] = df["score"].apply(
    lambda x: "A" if x >= 90 else "B"
)
```

Drop Columns / Rows

```
df.drop(columns=["temp"])
df.drop_duplicates(subset=["name"])
df.reset_index(drop=True)
```

String Operations

```
df["name"].str.lower()
df["name"].str.contains("ali", case=False)
df["name"].str.split(" ").str[0] # first name
```