

pandas works with tables of data, like a spreadsheet inside Python. A DataFrame is the whole table; a Series is one column. Most sections use the 4-row table `df` shown under "Look at the data". Most methods return a new table and leave `df` as it was.

CREATE

```
import pandas as pd

s = pd.Series([3.5, 2.0],
              index=["tea", "milk"])
float(s["milk"])
→ 2.0
a Series: values, each with a label (the labels are the index)
```

```
df = pd.DataFrame({"name": ["Ada"],
                    "score": [90]})
a DataFrame: each dict key becomes a column
```

```
rows = [{"name": "Ada", "score": 90},
        {"name": "Hedy", "score": 96}]
pd.DataFrame(rows).shape
→ (2, 2)
a DataFrame from a list of dicts: one dict per row
```

READ AND SAVE FILES

```
import io
import pandas as pd

df = pd.read_csv("scores.csv")
reads a CSV file; its first line gives the column names
```

```
text = "name,score\nAda,90"
df = pd.read_csv(io.StringIO(text))
df.shape
→ (1, 2)
io.StringIO makes text act like a file, to test read_csv
```

```
df.to_csv("out.csv", index=False)
saves the table; index=False leaves out the 0, 1, 2 row labels
```

LOOK AT THE DATA

```
import pandas as pd
df = pd.DataFrame({
    "name": ["Ada", "Grace", "Linus", "Hedy"],
    "team": ["A", "B", "A", "A"],
    "score": [90, 88, 75, 96]})

df.head()
shows the first 5 rows; df.head(2) shows 2
```

```
df.shape
→ (4, 3)
4 rows and 3 columns
```

```
df.info()
shows the column types and how many values are missing
```

```
df.describe()
the count, mean, min and max of each number column
```

```
float(df["score"].mean())
→ 87.25
the average score; float() gives a plain Python number
```

SELECT COLUMNS AND ROWS

```
df["name"].tolist()
→ ['Ada', 'Grace', 'Linus', 'Hedy']
one column name gives a Series
```

```
df[["name", "score"]].shape
→ (4, 2)
a list of column names gives a DataFrame
```

```
df.loc[1, "name"]
→ 'Grace'
loc picks by label: row label 1, column "name"
```

```
df.iloc[0, 0]
→ 'Ada'
iloc picks by position: row 0, column 0
```

```
df.loc[0:2, "name"].tolist()
→ ['Ada', 'Grace', 'Linus']
loc includes the end label (2)
```

```
df.iloc[0:2, 0].tolist()
→ ['Ada', 'Grace']
iloc stops before position 2, like a list slice
```

FILTER ROWS

```
df[df["score"] > 85]["name"].tolist()
→ ['Ada', 'Grace', 'Hedy']
keeps the rows where the test is True
```

```
df[(df["team"] == "A") & (df["score"] > 80)]["name"].tolist()
→ ['Ada', 'Hedy']
& means and, | means or; put brackets around each test
```

```
df[df["name"].isin(["Ada", "Linus"])].shape
→ (2, 3)
the rows where the name is in the list
```

```
df.query("score > 89")["name"].tolist()
→ ['Ada', 'Hedy']
the same kind of filter, written as a string
```

ADD AND CHANGE COLUMNS

```
df["pct"] = df["score"] / 100
df["pct"].tolist()
→ [0.9, 0.88, 0.75, 0.96]
a new column, calculated for every row
```

```
df.loc[df["score"] < 80, "score"] = 80
df["score"].tolist()
→ [90, 88, 80, 96]
changes the score only in rows where it is below 80
```

```
df.rename(columns={"score":
                  "pts"})
→ 'pts'
returns a new table with the column renamed
```

```
df = df.drop(columns=["pct"])
removes a column; drop() returns a new table, so
assign it to df
```

SORT

```
df.sort_values("score", ascending=False)
["name"].tolist()
→ ['Hedy', 'Ada', 'Grace', 'Linus']
highest score first
```

```
df.nlargest(2, "score")["name"].tolist()
→ ['Hedy', 'Ada']
the 2 rows with the highest scores
```

GROUP AND COUNT

```
df.groupby("team")
["score"].mean().to_dict()
→ {'A': 87.0, 'B': 88.0}
splits the rows by team, then gives one average per team
```

```
df.groupby("team")["score"].agg(["min",
                                 "max"]).loc["A"].tolist()
→ [75, 96]
several results per team at once: team A's min and max
```

```
df["team"].value_counts().to_dict()
→ {'A': 3, 'B': 1}
how many rows have each value
```

MISSING VALUES

```
import pandas as pd
s = pd.Series([4.0, None, 2.5])

s.isna().tolist()
→ [False, True, False]
NaN means a missing value
```

```
s.fillna(0).tolist()
→ [4.0, 0.0, 2.5]
replaces each missing value with 0
```

```
s.dropna().tolist()
→ [4.0, 2.5]
removes the missing values; df.dropna() removes whole rows
```

```
float(s.mean())
→ 3.25
mean() skips missing values
```

COMBINE TABLES

```
df.merge(cities, on="name")
["city"].tolist()
→ ['London', 'Vienna']
keeps only the names found in both tables
```

```
df.merge(cities, on="name",
        how="left").shape
→ (4, 4)
keeps every row of df; a row with no match gets NaN
```

```
len(pd.concat([df, df]))
→ 8
puts one table under the other
```

RUN A FUNCTION ON A COLUMN

```
df["name"].apply(len).tolist()
→ [3, 5, 5, 4]
apply() calls the function on every value
```

```
df["score"].apply(lambda x:
                  "pass" if x >= 80 else "fail")
).tolist()
→ ['pass', 'pass', 'fail', 'pass']
your own rule, written as a lambda (a one-line function)
```

```
df["name"].str.upper().tolist()
→ ['ADA', 'GRACE', 'LINUS', 'HEDY']
.str gives string methods for a whole column
```

WATCH OUT

```
df[df["score"] > 80]["bonus"] = 1
SettingWithCopyWarning: this may change a filtered
copy, not df. Use df.loc[df["score"] > 80, "bonus"] = 1
```

```
df["score"] > 80 and df["score"] < 95
→ raises ValueError
and/or do not work on columns: use & and |, with
brackets around each test
```

```
len(df.loc[0:2]), len(df.iloc[0:2])
→ (3, 2)
loc includes the end label; iloc does not
```

```
df = df.sort_values("score",
                    inplace=True)
inplace=True returns None, so df is now None. Write df
= df.sort_values("score")
```

```
df["Score"]
→ raises KeyError
column names are case-sensitive: check df.columns
```

```
df.append(df)
→ raises AttributeError
removed in pandas 2.0: use pd.concat([df, other])
```