

IMPORT & READ FILES	
<code>import pandas as pd</code> <code>import numpy as np</code>	standard aliases
<code>pd.read_csv("f.csv")</code>	load a CSV into a DataFrame
<code>sep=";", encoding="latin-1"</code>	delimiter / encoding fixes
<code>usecols=["a","b"], nrows=1000</code>	read only some columns / rows
<code>dtype={"id": str}</code> <code>parse_dates=["date"]</code>	force column types parse columns as datetime
<code>index_col="id", header=0</code>	choose index / header row
<code>na_values=["NA", "-"]</code>	extra strings treated as NaN
<code>pd.read_excel("f.xlsx", sheet_name="S1")</code>	Excel sheet (needs openpyxl)
<code>pd.read_json()</code> / <code>read_parquet()</code> / <code>read_sql(q, con)</code>	other sources

INSPECT	
<code>df.head(5)</code> / <code>df.tail()</code> / <code>df.sample(5)</code>	peek at rows
<code>df.shape</code>	(rows, columns)
<code>df.info()</code>	dtypes, non-null counts, memory
<code>df.describe(include="all")</code> <code>df.dtypes</code> / <code>df.columns</code>	summary stats types / column names
<code>df["c"].value_counts(normalize=True)</code>	frequencies (as proportions)
<code>df.isna().sum()</code>	missing values per column

TIPS	
Avoid chained assignment <code>df[m]["a"]=1</code> ; use <code>df.loc[m, "a"] = 1</code> .	
Use <code>.copy()</code> after slicing before modifying.	
Always check <code>df.info()</code> after loading: dates and numbers often arrive as <code>object</code> .	
Compare row counts before and after a merge to catch duplicated keys.	

SELECT & FILTER		
<code>df["a"]</code> / <code>df[["a","b"]]</code>	one column (Series) / several (DataFrame)	
<code>df.loc[row_labels, cols]</code>	label-based, end inclusive	
<code>df.loc[df["x"] > 5, ["a","b"]]</code>	rows by condition, chosen cols	
<code>df.iloc[0:5, 0:2]</code>	position-based, end exclusive	
<code>df[(df.a > 1) & (df.b == "x")]</code>	AND: use &, wrap each test in ()	
<code>df[(df.a > 1) (df.b == "x")]</code>	OR: NOT: ~	
<code>df[df["c"].isin(["x","y"])]</code>	membership	
<code>df[df["a"].between(1, 5)]</code>	inclusive range	
<code>df.query("a > 1 and b == 'x'")</code>	string-based filter	

CLEAN: MISSING & DUPLICATES		
<code>df.dropna(subset=["a"])</code>	drop rows missing a	
<code>df.ffill()</code> / <code>df.bfill()</code>	carry values forward/back	
<code>df["a"].fillna(df["a"].median())</code>	fill with a value	
<code>df.duplicated().sum()</code>	count duplicate rows	
<code>df.drop_duplicates(subset=["id"], keep="first")</code>	remove duplicates	
<code>df.drop(columns=["x"])</code>	drop columns	

CLEAN: TYPES & NAMES		
<code>df["a"].astype("int64")</code>	change dtype	
<code>df["c"].astype("category")</code>	low-cardinality text, saves memory	
<code>pd.to_numeric(s, errors="coerce")</code>	bad values become NaN	
<code>df.rename(columns={"old": "new"})</code>	rename columns	
<code>df.columns = df.columns.str.lower().str.replace(" ", "_")</code>	tidy all names	

STRINGS .STR & DATES .DT		
<code>s.str.strip()</code> / <code>.lower()</code> / <code>.title()</code>	trim / change case	
<code>s.str.contains("ab", case=False, na=False)</code>	boolean mask for text match	
<code>s.str.replace(r"\d+", "", regex=True)</code>	regex replace	
<code>s.str.split("-", expand=True)</code>	split into columns	
<code>s.dt.year</code> / <code>.month</code> / <code>.day</code>	date parts	
<code>s.dt.day_name()</code> / <code>.dt.dayofweek</code>	weekday name / number (Mon=0)	
<code>(d2 - d1).dt.days</code>	day difference	

CREATE COLUMNS		
<code>df["rev"] = df.qty * df.price</code>	vectorised arithmetic	
<code>df = df.assign(rev=lambda d: d.qty * d.price)</code>	chainable, returns a new frame	
<code>np.where(df.a > 0, "pos", "neg")</code>	if / else	
<code>np.select([c1, c2], ["A","B"], default="C")</code>	multiple conditions	
<code>df["g"].map({"M": "Male", "F": "Female"})</code>	map via dict	
<code>df["a"].apply(func)</code>	per value (slow; prefer vectorised)	
<code>pd.cut(df.age, [0,18,65,120], labels=["kid","adult","senior"])</code>	bin into ranges	

SORT		
<code>df.sort_values("a", ascending=False)</code>	sort by a column	
<code>df.sort_values(["a","b"], ascending=[True, False])</code>	multi-column sort	

GROUPBY & AGG		
<pre>out = (df.groupby("region") .agg(total=("sales", "sum"), avg=("sales", "mean"), orders=("id", "count"), cust=("cust", "nunique")) .reset_index())</pre>		
<code>df.groupby(["a","b"])["x"].sum()</code>	multiple keys, one column	
<code>df.groupby("a")["x"].transform("mean")</code>	group stat aligned to each row	
<code>df.groupby("a").size()</code>	rows per group	
<code>df.groupby("a", as_index=False)["x"].sum()</code>	keep key as a column	

RESHAPE		
<code>df.pivot_table(index="r", columns="c", values="v", aggfunc="sum", fill_value=0, margins=True)</code>	Excel-style pivot with totals	
<code>pd.crosstab(df.a, df.b, normalize="index")</code>	count table / row %	
<code>df.melt(id_vars="id", var_name="k", value_name="v")</code>	wide to long	
<code>df.pivot(index="id", columns="k", values="v")</code>	long to wide (unique pairs)	

COMBINE TABLES		
<code>pd.merge(a, b, on="id", how="left")</code>	SQL-style join	
<code>how="inner" / "left" / "right" / "outer"</code>	matching only / keep left / keep right / keep all	
<code>a.join(b, on="id")</code>	join on index (default left)	
<code>pd.concat([a, b], ignore_index=True)</code>	stack rows (axis=0)	
<code>pd.concat([a, b], axis=1)</code>	place side by side	

TIME SERIES		
<code>df["d"] = pd.to_datetime(df["d"], format="%d/%m/%Y")</code>	parse text to datetime	
<code>df = df.set_index("d").sort_index()</code>	datetime index	
<code>df.loc["2025-01":"2025-03"]</code> <code>df["x"].resample("ME").sum()</code>	slice by date strings month-end totals ("D","W","QE","YE")	
<code>df["x"].rolling(7).mean()</code>	7-row moving average	
<code>df["x"].shift(1)</code>	previous row's value (lag)	
<code>df["x"].pct_change()</code> <code>df["x"].cumsum()</code>	change vs previous row running total	

EXPORT		
<code>df.to_csv("out.csv", index=False)</code>	CSV without the index	
<code>df.to_excel("out.xlsx", sheet_name="S", index=False)</code>	Excel	
<code>df.to_parquet("out.parquet")</code>	fast, typed, compact	