Pandas Workflow Cheat Sheet

The Pandas 2.x / 3.0 workflow for 2026 — Arrow-backed strings, copy-on-write, and the ops that handle 90% of real data work.

01 · MODERN DEFAULTS

The Pandas 3.0 setup every new notebook needs

Pandas 3.0 (January 2026) enforces copy-on-write and makes PyArrow the default string backend. New code should opt in early so the upgrade is a no-op.

pd.options.mode.copy_on_write = True	Lazy copies — no more SettingWithCopyWarning
pd.options.future.infer_string = True	Force PyArrow string dtype on read_*
engine='pyarrow'	Multi-threaded CSV parsing — ~10-35x faster than the C engine
dtype_backend='pyarrow'	Read directly into Arrow-backed columns
pd.read_parquet(...)	Default to Parquet — typed columns, 5-10x smaller than CSV

02 · LOAD + INSPECT

First 30 seconds with a new dataset

- df = pd.read_csv(path, engine='pyarrow', dtype_backend='pyarrow') — modern entry point
- df.shape, df.dtypes, df.info(memory_usage='deep') — what you're holding
- df.head() / df.sample(5) — sample is unbiased, head is not
- df.describe(include='all') — numeric + categorical summary in one call
- df.isna().sum() — null counts per column, sorted is even better: .sort_values(ascending=False)

03 · CLEAN

The five operations that fix 80% of dirty data

df.drop_duplicates(subset=[...])	Always specify subset — full-row dedup is rarely what you want
df.dropna(subset=['col'])	Drop only when the relevant column is null, not the whole row
df['col'].fillna(df['col'].median())	Median > mean for numeric NA fill — robust to outliers
df['col'].str.strip().str.lower()	Arrow strings still expose .str — chain for normalisation
df.astype({'date': 'datetime64[ns]'})	Bulk type-cast — much faster than per-column .astype

04 · TRANSFORM

Vectorise everything; avoid .apply when possible

- df['z'] = (df['x'] - df['x'].mean()) / df['x'].std() — vectorised, no loop
- df['bucket'] = pd.cut(df['x'], bins=[0, 10, 50, 100], labels=['low', 'mid', 'hi'])
- df.assign(profit=lambda d: d.revenue - d.cost) — chainable, preserves original
- .pipe(func) — chain a function into the dataframe; great for reusable transforms
- Replace .apply(func, axis=1) with vectorised ops or pl.from_pandas(df) → Polars when the row-wise loop is the bottleneck

05 · GROUP + AGGREGATE

groupby is the engine of analysis

.groupby('k').size()	Count rows per group — fastest existence check
.groupby('k')['v'].mean()	Single-column aggregation — returns a Series
.groupby('k').agg({'v':'sum','c':'mean'})	Multiple aggregations in one pass
.groupby('k').agg(total=('v','sum'), avg=('v','mean'))	Named aggregation — explicit output column names
.groupby('k', dropna=False)	Include NaN as its own group — easy to miss otherwise
.transform('mean')	Broadcast group result back to original row count — keep df shape

06 · EXPORT

Save formats and what they're for

- df.to_parquet('out.parquet') — typed, compressed, fast; default for analysis pipelines
- df.to_csv(index=False) — only for sharing with humans / Excel; lossy on dtypes
- df.to_feather('out.feather') — Arrow-native format, even faster than Parquet for short-lived caches
- df.to_json(orient='records', lines=True) — newline-delimited JSON; great for streaming
- Never .to_pickle a dataframe for long-term storage — version-pinned, security risk, breaks on pandas upgrades

Beyond the basics

COMMON PITFALLS

- ! Chained assignment: df[df.x > 0]['y'] = 1 silently fails. Use .loc[df.x > 0, 'y'] = 1 instead
- ! merge order — pd.merge(left, right) defaults to inner join; specify how='left' explicitly when in doubt
- ! Forgetting reset_index after groupby — the grouping columns become an index, not regular columns
- ! Comparing floats with == — use np.isclose() or compare differences against a tolerance
- ! Using .iterrows() in production — it's 100x slower than vectorised ops; if you reach for it, you're probably wrong

FURTHER READING

- → Pandas 3.0 release notes (https://pandas.pydata.org/docs/whatsnew/v3.0.0.html)
- → PyArrow + Pandas user guide (https://pandas.pydata.org/docs/user_guide/pyarrow.html)
- → Effective Pandas — Matt Harrison (recommended book) (https://store.metasnake.com/effective-pandas-book)
- → Polars guide (for when Pandas is too slow) (https://docs.pola.rs/)