**Pandas Data Cleaning & Wrangling — Complete Guide**

**Overview**

This document is a complete, practical reference for cleaning and wrangling data using **pandas**. It covers everything you asked for: inspection, missing data, categorical (nominal + ordinal), text, dates, types, outliers, joins, reshaping, scaling, memory/performance, validation, pipelines and a final checklist you can use before modeling.

**Table of Contents**

1. Principles and mindset
2. Quick inspection and exploration
3. Missing values (detection & imputation)
4. Duplicates
5. Data types & conversions
6. Text data cleaning
7. Categorical variables (nominal & ordinal)
8. Numerical cleaning & outliers
9. Date/time
10. Joins & merges
11. Reshaping: melt/pivot/stack
12. Grouping, aggregation, transforms
13. Feature engineering patterns
14. Scaling & normalization
15. Large datasets & memory optimization
16. Validation, testing & QA
17. Automating pipelines
18. Saving & exporting
19. Common pitfalls & debugging tips
20. Pre-modeling checklist
21. Exercises & resources
22. Appendix: handy snippets

**1. Principles and mindset**

* **Make data *accurate, consistent, and explainable*.** Cleaning isn't magic — document transformations.
* **Work reproducibly**: keep code, use functions, use a pipeline.
* **Prefer vectorized operations**: avoid Python loops over rows.
* **Start with exploration** (don’t fix until you understand the problem).

**2. Quick inspection and exploration**

import pandas as pd

df = pd.read\_csv('data.csv')

# Quick checks

df.shape # rows, cols

df.head()

df.tail()

df.sample(10)

# Structural info

df.info() # dtypes, non-null counts

df.describe(include='all').T

# Value-level checks

df.isna().sum() # missing counts per column

for c in df.columns:

print(c, df[c].nunique(), df[c].dtype)

# For categorical columns

df['col'].value\_counts(dropna=False)

# Look at memory usage

df.info(memory\_usage='deep')

**3. Missing values (detection & imputation)**

**Detect**

* df.isna().sum() and percent: df.isna().mean()
* Visual: missingno (library) or a heatmap
* Check suspicious sentinels: -999, "", 'NA', 'N/A', 'Unknown'

**Common strategies**

* Drop rows/columns (dropna(axis=0/1, thresh=...)).
* Fill with a constant: df.fillna(0) or fillna('missing').
* Forward/backward fill: ffill(), bfill() useful for time series.
* Impute with statistics: mean/median/mode per column or per-group.
* Model-based imputation: KNN, iterative (e.g., sklearn IterativeImputer).
* Use pd.NA and pandas nullable dtypes for consistency.

**Examples**

# Percent missing

(df.isna().mean()\*100).sort\_values(ascending=False)

# Drop columns with > 80% missing

df = df.loc[:, df.isna().mean() < 0.8]

# Fill numeric by group median

df['age'] = df.groupby('country')['age'].apply(lambda x: x.fillna(x.median()))

# Fill categorical with mode

mode = df['city'].mode().iloc[0]

df['city'] = df['city'].fillna(mode)

# Use sklearn SimpleImputer for numeric

from sklearn.impute import SimpleImputer

imp = SimpleImputer(strategy='median')

df['num'] = imp.fit\_transform(df[['num']])

**Tip:** When imputing, create indicator columns capturing where you imputed (e.g., col\_missing = df['col'].isna().astype(int))—models often find these signals useful.

**4. Duplicates**

# Find duplicates across all columns

dups = df[df.duplicated()]

# Drop duplicates keeping first

df = df.drop\_duplicates()

# Duplicates based on subset

df[df.duplicated(subset=['name','dob'])]

Consider whether duplicates are exact copies (drop) or event duplicates (need dedup rules: aggregate, keep latest, sum counts).

**5. Data types & conversions**

**Why important:** dtypes affect memory, operations, and correctness.

**Convert safely**

* pd.to\_numeric(df[col], errors='coerce') to coerce bad strings to NaN.
* pd.to\_datetime(df[col], errors='coerce', utc=True) for dates.
* df[col].astype('category') for low-cardinality strings.

**Nullable dtypes**

* Use Int64, string, boolean dtypes if you need NA with integers/booleans.

**Example**

# Clean numeric stored as strings

df['price'] = (df['price'].str.replace('[\$,]', '', regex=True)

.replace('', pd.NA)

.pipe(pd.to\_numeric, errors='coerce'))

# Date parsing

df['order\_dt'] = pd.to\_datetime(df['order\_date'], errors='coerce', dayfirst=False)

# Use category to save memory

df['country'] = df['country'].astype('category')

**Check for mixed types**  
df['col'].apply(type).value\_counts() — if you see many types, clean them first.

**6. Text data cleaning**

**Common tasks:** normalize case, strip whitespace, remove punctuation, fix encodings, remove non-printables, collapse repeated spaces.

# basic

s = df['text'].astype('string')

s = s.str.strip().str.lower()

# remove punctuation

s = s.str.replace(r"[^\w\s]", '', regex=True)

# replace multiple spaces

s = s.str.replace(r'\s+', ' ', regex=True)

# extract numbers or tokens

numbers = s.str.extract(r'(\d+)')

# fillna

s = s.fillna('')

**Advanced NLP cleaning**: use nltk, spaCy, or textacy for tokenization, lemmatization, stop-word removal, and named-entity extraction.

**7. Categorical variables (nominal & ordinal)**

**Definitions**

* *Nominal*: categories without order (e.g., color: red/blue/green).
* *Ordinal*: categories with order (e.g., size: small/medium/large).

**Strategies**

* Keep as category dtype for memory and speed.
* For nominal features: One-hot encoding (pd.get\_dummies or OneHotEncoder) or frequency encoding.
* For high-cardinality nominal features: target/frequency/hash/embedding.
* For ordinal features: map to integers preserving order and use CategoricalDtype to enforce order.

**Examples**

# One-hot

df = pd.get\_dummies(df, columns=['color'], dummy\_na=False, drop\_first=False)

# Ordinal mapping

df['size'] = df['size'].map({'small': 0, 'medium': 1, 'large': 2}).astype('Int64')

# Category dtype with order

from pandas.api.types import CategoricalDtype

order = CategoricalDtype(categories=['low','medium','high'], ordered=True)

df['risk'] = df['risk'].astype(order)

# Frequency encoding

freq = df['city'].value\_counts(normalize=True)

df['city\_freq'] = df['city'].map(freq)

**Target leakage warning:** avoid using target-based encodings computed on the whole dataset without cross-validation or proper folds — this leaks target information to training features.

**8. Numerical cleaning & outliers**

**Inspect distributions**: df['col'].hist() or df['col'].describe()

**Detect outliers**

* IQR method: lower = Q1 - 1.5*IQR, upper = Q3 + 1.5*IQR
* Z-score: abs((x-mean)/std) > 3
* Robust: MAD-based

**Handle outliers**

* Remove rows
* Cap (winsorize)
* Transform (log/boxcox)
* Keep but mark with an outlier flag

# IQR

q1 = df['x'].quantile(0.25)

q3 = df['x'].quantile(0.75)

iqr = q3 - q1

mask = (df['x'] >= q1 - 1.5\*iqr) & (df['x'] <= q3 + 1.5\*iqr)

df\_clean = df[mask]

# winsorize using scipy

from scipy.stats.mstats import winsorize

# apply on numpy array before assigning back

**Transforms**

* np.log1p, np.sqrt, or sklearn.preprocessing.PowerTransformer (Yeo-Johnson) can make skewed distributions more normal.

**9. Date/time**

# Parse

df['dt'] = pd.to\_datetime(df['dt'], errors='coerce')

# Extract

df['year'] = df['dt'].dt.year

df['month'] = df['dt'].dt.month

df['dayofweek'] = df['dt'].dt.dayofweek

# Time series resampling

df.set\_index('dt', inplace=True)

daily = df.resample('D').agg({'value': 'sum'})

# Rolling

df['rolling\_7'] = df['value'].rolling(window=7).mean()

**Timezone**: use dt.tz\_localize() and dt.tz\_convert() when needed.

**10. Joins & merges**

**Types**: inner, left, right, outer. Use indicator=True to inspect matching rows.

merged = df.merge(other, how='left', on='id', indicator=True)

merged['\_merge'].value\_counts()

**Common issues**

* Wrong key type (string vs int) — cast before merge.
* Duplicated keys: expect duplication and aggregate or dedupe first.

**11. Reshaping: melt / pivot / stack**

# Wide -> long

long = df.melt(id\_vars=['id','date'], value\_vars=['var1','var2'], var\_name='metric', value\_name='value')

# Long -> wide

wide = long.pivot\_table(index=['id','date'], columns='metric', values='value', aggfunc='first').reset\_index()

**12. Grouping, aggregation, transforms**

**Aggregate**

agg = df.groupby('user').agg(total=('amt','sum'), avg=('amt','mean'), n=('amt','size'))

**Transform** (returns aligned series)

# group mean

df['user\_mean'] = df.groupby('user')['amt'].transform('mean')

# ranks within group

df['rank'] = df.groupby('user')['amt'].rank(ascending=False)

**Apply vs transform**: apply can change shape; transform must return same shape as input.

**13. Feature engineering patterns**

* Binning numeric -> pd.cut or pd.qcut
* Interaction features: multiply/cross features
* Temporal features: hour, dayofweek, is\_month\_end, time\_since\_last\_event
* Aggregations as features: groupby + transform
* Lag features: groupby().shift(1) for time series

# example lag

df['prev\_amt'] = df.sort\_values('dt').groupby('user')['amt'].shift(1)

**14. Scaling & normalization**

* For tree models often unnecessary.
* For linear models or distance-based models: StandardScaler, MinMaxScaler, RobustScaler.
* Apply scaling **after** train/test split to avoid leakage.

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df[['x','y']] = scaler.fit\_transform(df[['x','y']])

**15. Large datasets & memory optimization**

**Tips**

* Use usecols and dtype when reading CSV.
* Convert object columns with limited unique values to category.
* Downcast numeric types: pd.to\_numeric(col, downcast='unsigned').
* Use chunksize to process file in streaming fashion.
* Store as parquet for faster re-loads.
* For very large data consider Dask, Vaex, or Spark.

# downcast

df['int\_col'] = pd.to\_numeric(df['int\_col'], downcast='integer')

# read in chunks

for chunk in pd.read\_csv('big.csv', chunksize=100000):

process(chunk)

**16. Validation, testing & QA**

* Keep holdout or validation splits.
* Use assertions: assert df['id'].is\_unique.
* Use df.sample(50) and visually inspect.
* Use Great Expectations or pandera for schema checks.

Example checks:

# ensure no negative prices

assert (df['price'] >= 0).all(), 'Negative price present'

# check referential integrity

missing\_keys = set(df['fk']) - set(ref['id'])

**17. Automating pipelines**

* Build reusable transformer functions or classes.
* Use sklearn.pipeline.Pipeline or ColumnTransformer for modeling.
* For pure pandas pipelines, build functions that accept and return DataFrames.

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder, StandardScaler

num\_cols = ['age','income']

cat\_cols = ['city']

num\_pipe = Pipeline([('imputer', SimpleImputer(strategy='median')), ('scaler', StandardScaler())])

cat\_pipe = Pipeline([('imputer', SimpleImputer(strategy='most\_frequent')), ('ohe', OneHotEncoder(handle\_unknown='ignore'))])

pre = ColumnTransformer([('num', num\_pipe, num\_cols), ('cat', cat\_pipe, cat\_cols)])

# then embed pre in a full pipeline with an estimator

**Custom pandas transformer** (scikit-learn compatible) - create class with fit/transform that returns numpy arrays or DataFrames.

**18. Saving & exporting**

* df.to\_parquet('clean.parquet') preserves dtypes and is fast.
* df.to\_csv('clean.csv', index=False) for simple exchange (note dtype loss for categories).
* Use compression: to\_parquet(..., compression='snappy').

**19. Common pitfalls & debugging tips**

* **Mixed dtypes** in a column (strings + numbers) — coerce early.
* **Silent coercions**: astype(int) will fail on NaN; use nullable Int64 or fillna first.
* **Timezones**: beware when joining time-series from different tz.
* **Leaky features**: features derived using future information or full-target encodings.
* **Imputation leakage**: do fit only on training folds when modeling.
* **Memory explosions**: creating many large temporary copies — use inplace sparsely and mind copying semantics.

**20. Pre-modeling checklist (quick)**

* No missing values where model does not accept them
* Correct dtypes
* Target leakage removed
* Train/test split before any target-aware transforms
* Categorical encodings chosen appropriately
* Outliers handled or flagged
* ID columns removed or used intentionally
* Time-based splits for time series
* Validation checks (unique keys, referential integrity)

**21. Exercises & resources**

**Exercises:**

1. Load a messy CSV and write a function clean(df) that returns a cleaned DataFrame and a list of changes applied.
2. Create fold-aware target encoding scheme without leakage.
3. Process a 5GB CSV using chunksize and compute aggregates per user.

**Libraries to explore:** ydata-profiling (pandas-profiling), sweetviz, missingno, Great Expectations, pandera, category\_encoders.

**22. Appendix: Handy snippets**

**Detect columns likely numeric but object-typed**

maybe\_num = [c for c in df.columns if df[c].dtype == 'object' and df[c].str.match(r'^-?\d+(\.\d+)?$').any()]

**Make readable report of nulls and uniques**

report = pd.DataFrame({

'dtype': df.dtypes.astype(str),

'n\_missing': df.isna().sum(),

'pct\_missing': df.isna().mean(),

'n\_unique': df.nunique(dropna=False)

}).sort\_values('pct\_missing', ascending=False)

report

**Downcast numeric**

for col in df.select\_dtypes(include=['int','float']).columns:

df[col] = pd.to\_numeric(df[col], downcast='integer' if df[col].dtype=='int64' else 'float')

**Safe left join with indicator and dedupe keys**

left = df

right = other.drop\_duplicates(subset='id')

merged = left.merge(right, on='id', how='left', indicator=True)

**Final notes**

* This document is intended as both a tutorial and a checklist. Use the code snippets as building blocks and adapt them to your dataset and modeling goals.
* If you want, I can walk you step-by-step through cleaning *your* dataset — upload the CSV and I will show a tailored cleaning plan and code.

*End of guide.*