**Data-cleaning workflow (Jupyter Notebook) — step-by-step with code & short descriptions**

Below is a ready-to-run, cell-by-cell workflow you can paste into a Jupyter notebook. Each numbered cell shows the Python commands (using **numpy** and **pandas**) and a short description of what it does. Replace column names and file paths with those from your dataset.

### Cell 1 — Imports & display settings

# Cell 1: imports and handy display options

import numpy as np

import pandas as pd

# Optional: make outputs easier to inspect in notebook

pd.set\_option('display.max\_columns', 120)

pd.set\_option('display.width', 120)

pd.set\_option('display.max\_colwidth', 200)

Description: Load the libraries and make Jupyter output wider to inspect tables.

### Cell 2 — Load data

# Cell 2: load dataset (CSV or Excel)

df = pd.read\_csv('path/to/your\_file.csv') # or pd.read\_excel('file.xlsx')

df.head()

Description: Read the file into df and show the first rows.

### Cell 3 — Quick initial exploration

# Cell 3: quick overview

print("Shape:", df.shape)

display(df.head())

display(df.sample(5, random\_state=42))

df.info()

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

Description: Get number of rows/cols, sample rows, dtypes, and basic statistics.

### Cell 4 — Make a working copy

# Cell 4: work on a copy to avoid accidental edits

df\_clean = df.copy()

Description: Keep original df intact; operate on df\_clean.

### Cell 5 — Detect missing values

# Cell 5: missing value summary

missing\_count = df\_clean.isna().sum().sort\_values(ascending=False)

missing\_pct = (df\_clean.isna().mean().sort\_values(ascending=False) \* 100).round(2)

pd.concat([missing\_count, missing\_pct], axis=1, keys=['missing\_count', 'missing\_pct']).head(20)

Description: See which columns have missing values and percentage missing.

### Cell 6 — Drop columns / rows with too many missing values

# Cell 6: drop columns with > 60% missing; drop rows missing more than 50% of columns

col\_thresh = 0.6

df\_clean = df\_clean.loc[:, df\_clean.isnull().mean() < col\_thresh]

row\_thresh = 0.5

df\_clean = df\_clean.dropna(thresh=int(df\_clean.shape[1] \* (1 - row\_thresh)))

Description: Remove very sparse columns and rows to reduce noise.

### Cell 7 — Convert data types (string → datetime / numeric / category)

# Cell 7: dtype conversions

# Example: parse dates

df\_clean['signup\_date'] = pd.to\_datetime(df\_clean['signup\_date'], errors='coerce')

# Example: coerce numeric strings to numbers

df\_clean['monthly\_charges'] = pd.to\_numeric(df\_clean['monthly\_charges'], errors='coerce')

# Example: convert small cardinality strings to category

df\_clean['gender'] = df\_clean['gender'].astype('category')

Description: Use errors='coerce' to turn invalid parse results into NaT/NaN so you can handle them.

### Cell 8 — Handle remaining missing values (impute or drop)

# Cell 8: targeted imputations

# Numeric: fill with median (robust to outliers)

num\_cols = df\_clean.select\_dtypes(include=[np.number]).columns.tolist()

for c in ['monthly\_charges', 'tenure']: # replace with your numeric columns

if c in df\_clean:

df\_clean[c].fillna(df\_clean[c].median(), inplace=True)

# Categorical: fill with mode

cat\_cols = df\_clean.select\_dtypes(include=['object', 'category']).columns.tolist()

for c in ['gender', 'internet\_service']: # replace with your categorical columns

if c in df\_clean:

df\_clean[c].fillna(df\_clean[c].mode().iloc[0], inplace=True)

# Time series: interpolation (if appropriate)

if 'value\_ts' in df\_clean:

df\_clean['value\_ts'] = df\_clean['value\_ts'].interpolate(method='linear')

Description: Use median for numeric, mode for categorical. Choose method depending on data semantics.

### Cell 9 — Remove duplicates

# Cell 9: check and drop duplicates

print("Duplicates:", df\_clean.duplicated().sum())

df\_clean = df\_clean.drop\_duplicates()

Description: Remove duplicate records; you can pass subset=[...] to target columns.

### Cell 10 — Clean string columns (trim, case, empty→NaN)

# Cell 10: tidy text fields

str\_cols = df\_clean.select\_dtypes(include='object').columns

for c in str\_cols:

df\_clean[c] = df\_clean[c].str.strip() # trim whitespace

df\_clean[c] = df\_clean[c].replace('', np.nan) # empty -> NaN

# optional: normalize case

# df\_clean[c] = df\_clean[c].str.lower()

Description: Remove stray spaces and normalize empty strings.

### Cell 11 — Standardize categories / map values

# Cell 11: normalize known variants to consistent labels

if 'state' in df\_clean:

mapping = {'lagos':'Lagos', 'lag':'Lagos', 'lagos state':'Lagos'}

df\_clean['state'] = df\_clean['state'].str.lower().replace(mapping).str.title()

Description: Fix typos / aliasing for categorical labels.

### Cell 12 — Detect & handle outliers (IQR and Z-score examples)

# Cell 12: Outlier detection (IQR method) for one numeric column

col = 'monthly\_charges'

Q1 = df\_clean[col].quantile(0.25)

Q3 = df\_clean[col].quantile(0.75)

IQR = Q3 - Q1

outlier\_mask = (df\_clean[col] < (Q1 - 1.5 \* IQR)) | (df\_clean[col] > (Q3 + 1.5 \* IQR))

print("Outliers (IQR) in", col, ":", outlier\_mask.sum())

# Option A: inspect outliers

df\_clean.loc[outlier\_mask, [col]].head()

# Option B: remove outliers (if justified)

df\_clean = df\_clean.loc[~outlier\_mask]

# Z-score method across multiple numeric columns

num\_cols = df\_clean.select\_dtypes(include=[np.number]).columns

z = np.abs((df\_clean[num\_cols] - df\_clean[num\_cols].mean()) / df\_clean[num\_cols].std())

multicol\_outliers = (z > 3).any(axis=1)

print("Rows with z-score > 3 in any numeric column:", multicol\_outliers.sum())

Description: Choose an outlier strategy (inspect, cap, remove) based on domain knowledge.

### Cell 13 — Feature engineering (create meaningful derived columns)

# Cell 13: example feature engineering

# Create age groups

if 'age' in df\_clean:

df\_clean['age\_group'] = pd.cut(df\_clean['age'], bins=[0,18,35,60,200],

labels=['child','young\_adult','adult','senior'])

# Example: total charges from components

if {'call\_charges','internet\_charges'}.issubset(df\_clean.columns):

df\_clean['total\_charges'] = df\_clean[['call\_charges','internet\_charges']].sum(axis=1)

Description: Add columns that make modeling/analysis easier.

### Cell 14 — Encode categorical variables

# Cell 14: encoding

# One-hot encoding for non-ordinal categories:

df\_encoded = pd.get\_dummies(df\_clean, columns=['internet\_service', 'contract'], drop\_first=True)

# Ordinal mapping example:

priority\_map = {'low':1, 'medium':2, 'high':3}

if 'priority' in df\_clean:

df\_encoded['priority\_ord'] = df\_clean['priority'].map(priority\_map)

Description: Choose encoding to suit downstream use (models vs reporting).

### Cell 15 — Scale numeric features (if needed)

# Cell 15: simple standardization (mean=0, std=1) without sklearn

num\_cols = df\_encoded.select\_dtypes(include=[np.number]).columns.tolist()

# avoid scaling ID-like columns

num\_cols = [c for c in num\_cols if c not in ['ID','customer\_id']]

df\_encoded[num\_cols] = (df\_encoded[num\_cols] - df\_encoded[num\_cols].mean()) / df\_encoded[num\_cols].std()

Description: Scaling helps ML algorithms that are distance-based; not always needed for reporting.

### Cell 16 — Final checks, reset index & save cleaned data

# Cell 16: final checks & export

df\_encoded.reset\_index(drop=True, inplace=True)

df\_encoded.info()

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

# Save

df\_encoded.to\_csv('cleaned\_data.csv', index=False)

# or binary for speed: df\_encoded.to\_pickle('cleaned\_data.pkl')

Description: Inspect the final dataframe and persist the cleaned dataset.

## Handy one-liners & utilities (useful in notebook)

# Show columns with any nulls

[col for col in df\_clean.columns if df\_clean[col].isnull().any()]

# Count unique values per column (helpful to decide category -> numeric)

df\_clean.nunique().sort\_values(ascending=False).head(20)

# Quick type coercion for many columns (numeric)

df\_clean = df\_clean.apply(lambda s: pd.to\_numeric(s, errors='ignore') if s.dtype == 'object' and s.str.isnumeric().any() else s)

# Save change log (very useful)

changes = []

changes.append("Dropped columns with >60% missing")

changes.append("Imputed monthly\_charges with median")

# ... append as you go, then save

pd.Series(changes).to\_csv('cleaning\_log.csv', index=False)

## Quick checklist before finishing

* Did you back up raw data (df) before edits?
* Did you document every transformation (log file or notebook markdown)?
* Are imputations and outlier treatments defensible for your use case?
* Did you re-check df.info() and df.describe() after major transforms?
* Did you save the cleaned file and (optionally) a sample for review?