Handling Missing Values in Pandas

JUPYTER NOTEBOOK TECHNICAL HANDOUT

Data cleaning is arguably the most critical step in any data science pipeline. Missing information typically manifests as standard `NaN` (Not a Number) values or technical artifacts like empty strings (`""`) and whitespace rows. This handout outlines systematic discovery patterns and programmatic execution paradigms using the Python `pandas` library.

1. Diagnostic Phase: Detection & Profiling

Standard data loaders automatically map blank fields to `np.nan`, but user entries often contain empty strings or phantom whitespace that bypasses native detectors.

A. Standard NaN Detection

To count missing entries across all features globally, combine `.isna()` (or `.isnull()`) with a linear column summation:

```
import pandas as pd
import numpy as np

# Instantiate standard analytical environment
data = {
    'Age': [25, np.nan, 30, 22, 35],
    'City': ['New York', 'London', '', 'Paris', 'Tokyo'],
    'Salary': [50000, 60000, 70000, np.nan, 80000]
}
df = pd.DataFrame(data)

# Compute absolute frequency of missing data per feature
print(df.isna().sum())
```

B. Capturing Ghost Empty Strings

Empty text structures register as valid strings. You can catch them using vectorized string operations or direct boolean masks:

```
# Check for zero-length strings explicitly
empty_strings = (df['City'] == "").sum()

# Catch variations containing spaces or invisible formatting characters
whitespace_strings = (df['City'].str.strip() == "").sum()
```

> 💡 **Structural Best Practice: Unified Conversion**
>
> To optimize downstream processes, immediately map all whitespace patterns and empty strings to native `NaN` objects via regular expressions:
>
> ```
> df = df.replace(r'^\s*$', np.nan, regex=True)
> ```

2. Strategic Assessment Matrix

Choosing between structural removal and statistical imputation depends on your underlying missingness mechanism, column volume, and downstream analytical requirements.

Strategy	Ideal Conditions	Core Advantages	Inherent Disadvantages
Structural Drop `.dropna()`	• MCAR (Missing Completely at Random) • Negligible volume (< 5% of records) • Vital missing target labels (Y)	• Preserves statistical truth • Fast execution time	• Shrinks dataset volume • Can bias data if missingness is non-random
Statistical Imputation `.fillna()`	• Missingness spans critical rows • High-volume rows with active predictors • Strict requirements for non-null shapes	• Maintains sample size • Retains predictive power	• Risk of artifact insertion • Underestimates true variance

3. Implementation Mechanics

Strategy A: Execution Pathways for Dropping Data

Dropping data should be precise. Use the `subset` argument to protect non-critical dimensions, or establish threshold requirements.

```
# Pathway 1: Drop complete row if ANY single variable is missing
df_strict = df.dropna()

# Pathway 2: Narrow drop logic exclusively to vital operational columns
df_targeted = df.dropna(subset=['Salary'])

# Pathway 3: Columnar elimination based on missing density thresholds
# Example: Drop the whole column if more than 50% of values are missing
df_dense = df.dropna(thresh=len(df) * 0.5, axis=1)
```

Strategy B: Execution Pathways for Statistical Imputation

Imputation methodologies must align with the statistical data type of the underlying series.

1. Numerical Continuous Features (Mean vs. Median)

- **Mean (μ):** Highly viable for symmetric, perfectly Gaussian normal distributions without extreme outliers.
- **Median (M):** Recommended for highly skewed profiles or distributions with anomalous heavy-tail outliers.

```
# Compute safe statistics ignoring native NaN flags
median_age = df['Age'].median()

# Perform inplace replacement or safe reassignment
df['Age'] = df['Age'].fillna(median_age)
```

2. Categorical Nominal Features (Mode vs. Constant Sentinels)

- **Mode (Mo):** Recommended if a single class dominates the structural baseline distribution.
- **Constant Label/Sentinel:** Assigning a value like `"Unknown"` or `"Missing"` prevents arbitrary classification and preserves missingness structure for downstream ML pipelines.

```
# Scenario A: Populate utilizing the most common categorical string (Mode)
most_frequent_city = df['City'].mode()[0]
df['City'] = df['City'].fillna(most_frequent_city)

# Scenario B: Implement a constant placeholder to track absence safely
df['City'] = df['City'].fillna('Unknown')
```