Data Cleaning & Preprocessing

Learn how to transform raw data into clean, structured datasets ready for analysis or machine learning — with clear Python examples.

Contents

What is Data Preprocessing?

Data preprocessing is the process of converting raw, unstructured, or inconsistent data into a clean, structured format that can be analyzed or used to train machine learning models.

Without preprocessing, raw data can lead to:

- Incorrect insights
- Poor model performance
- Misleading results

Why is Data Preprocessing Important?

The quality of your input data directly determines the quality of your output model or analysis. As the saying goes:

“Garbage in, garbage out.”

Preprocessing ensures:

- **Accuracy**: Eliminates inconsistencies.
- **Fairness**: Prevents features with large scales from dominating.
- **Completeness**: Deals with missing or duplicate values.
- **Efficiency**: Reduces computational load.

Key Steps in Data Preprocessing

1. Handling Missing Values

Missing values occur due to incomplete surveys, data corruption, or entry errors. Ignoring them can bias results.

Techniques:

- Drop missing rows/columns (only if few).
- Fill (impute) with mean, median, or mode.
- Advanced: Predict missing values (KNN, regression).
- Add “missingness flags” (useful in ML).

Example (Python):

```
from sklearn.impute import SimpleImputer

# Add missingness flags
for col in ['age', 'income']:
    df[col + '_missing'] = df[col].isnull().astype(int)

# Impute numeric with median
imputer = SimpleImputer(strategy='median')
df[['age','income']] = imputer.fit_transform(df[['age','income']])
```

---

2. Data Type Conversion (Dates, Prices)

Raw datasets often store dates as strings and prices with symbols. Converting them enables meaningful analysis.

Example (Python):

```
# Convert to datetime
df['signup_date'] = pd.to_datetime(df['signup_date'], errors='coerce')
```

```
# Clean currency symbols
df['price'] = df['price'].replace('[\$,]', '', regex=True).astype(float)
```

Enables feature engineering like extracting months, weekdays, or price comparisons.

---

3. Text Cleaning (HTML, Stopwords)

Text data is usually noisy and requires cleaning. Common problems include:

- HTML tags (`<p>`, `<br>`)
- Mixed casing
- Stopwords (e.g., “the”, “and”, “is”)

Example (Python):

```
from bs4 import BeautifulSoup
import re
from nltk.corpus import stopwords

# Remove HTML
df['review'] = df['review'].apply(lambda x: BeautifulSoup(x,
"html.parser").get_text())

# Lowercase, remove non-letters
df['review'] = df['review'].str.lower().str.replace(r'[^a-z ]','',regex=True)

# Remove stopwords
stops = set(stopwords.words('english'))
df['review'] = df['review'].apply(lambda x: ' '.join([w for w in x.split() if w
not in stops]))
```

---

4. Normalization & Standardization

Different scales (e.g., income in thousands, age in tens) bias models.

- **Normalization**: Scale to [0,1].
- **Standardization**: Mean = 0, Std = 1.

Example (Python):

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Normalization
df[['income']] = MinMaxScaler().fit_transform(df[['income']])

# Standardization
```

```
df[['age']] = StandardScaler().fit_transform(df[['age']])
```

---

5. Handling Duplicates & Outliers

- **Duplicates** → double counting.
- **Outliers** → skew mean, distort predictions.

Example (Python):

```
# Remove duplicates
df = df.drop_duplicates()

# Remove outliers with IQR
Q1, Q3 = df['income'].quantile([0.25, 0.75])
IQR = Q3 - Q1
df = df[(df['income'] >= Q1 - 1.5*IQR) & (df['income'] <= Q3 + 1.5*IQR)]
```

---

6. Encoding Categorical Data

Models require **numeric input**, not text labels.

Example (Python):

```
from sklearn.preprocessing import LabelEncoder

# Label encoding
df['gender'] = LabelEncoder().fit_transform(df['gender'])

# One-hot encoding
df = pd.get_dummies(df, columns=['city'], drop_first=True)
```

- Tree models can work with label encoding.
- Linear models prefer one-hot encoding.

---

7. Feature Engineering (Creating New Features)

Feature engineering is the process of **creating, transforming, or selecting features** to improve model performance.

Example (Python):

```
# Extract date parts
df['signup_month'] = df['signup_date'].dt.month

# Word count
```

```
df['review_word_count'] = df['review'].apply(lambda x: len(str(x).split()))

# Interaction feature
df['income_per_day'] = df['income'] / (df['signup_days'] + 1)
```

---

8. Merging Datasets

Datasets are often split across sources (e.g., users, transactions). Merging integrates them into one dataset.

Example (Python):

```
df = pd.merge(users_df, transactions_df, on='user_id', how='inner')
```

Creates a single unified dataset.

Tools for Data Preprocessing

- **Pandas**: cleaning, merging, missing values.
- **NumPy**: numerical operations.
- **Scikit-learn**: scaling, encoding, pipelines.
- **BeautifulSoup / NLTK**: text cleaning.
- **Cloud ETL tools** (AWS Glue, Azure Data Factory): big data preprocessing.

Best Practices

1. Understand your data (explore before cleaning).
2. Automate repetitive steps (e.g., `sklearn.Pipeline`).
3. Document decisions (why you dropped/kept certain rows).
4. Iterate — preprocessing is never "one and done."

Conclusion & Future Work

Conclusion

Data preprocessing is not just about cleaning; it's about **unlocking hidden insights** and ensuring fair, accurate models. By systematically handling missing values, cleaning text, scaling numbers, and engineering features, you transform raw data into a powerful foundation for AI and analytics.

👉 The best models are built on the **cleanest data**.

Future Work

- **Automated Data Cleaning** with AutoML pipelines.
- **Advanced Feature Engineering** with embeddings & autoencoders.
- **Scalable Big Data Preprocessing** using Spark/Dask.
- **Bias & Fairness Handling** for ethical AI.
- **Real-time Data Quality Monitoring** with drift detection.