Data Wrangling and Preprocessing for Data Science

Shubneet [1], Anushka Raj Yadav [2], Navjot Singh Talwandi [3]

[1,2,3*]Department of Computer Science, Chandigarh University, Gharuan, Mohali, 140413, Punjab, India.

Contributing authors: jeetshubneet27@gmail.com; ay462744@gmail.com; navjot.e17908@cumail.in;

Abstract

Data preprocessing forms the critical foundation of effective data science workflows, transforming raw, unstructured data into reliable inputs for analysis and modeling. This chapter emphasizes the pivotal role of preprocessing in addressing pervasive data quality challenges such as missing values, outliers, and inconsistent formatting, which collectively impact over 80% of real-world datasets [1]. Key techniques include robust missing value imputation strategies (mean, median, and advanced methods like MICE), outlier detection using interquartile ranges (IQR) and clustering algorithms, and feature engineering to derive meaningful predictors. Practical implementation is demonstrated through industry-standard tools: Python's `Pandas` for automated data cleaning, R's `dplyr` for structured transformations, and `OpenRefine` for non-programmatic data wrangling. These tools enable reproducible preprocessing pipelines that maintain data integrity while optimizing datasets for machine learning applications. The chapter also highlights how systematic preprocessing reduces computational overhead by 30-50% and improves model accuracy by addressing biases inherent in raw data [2]. By integrating theoretical principles with hands-on examples, this section equips practitioners to handle heterogeneous data sources, ensure compatibility across analytical platforms, and build trustworthy data products.

Keywords: data cleaning, missing values, outlier detection, feature engineering, data integration

1 Introduction

In the era of data-driven decision-making, raw datasets rarely arrive in a state suitable for analysis. Over 80% of data science projects involve significant preprocessing to address missing values, outliers, inconsistent formats, and integration challenges [1]. This critical phase transforms chaotic, error-prone data into structured inputs that fuel reliable machine learning models and actionable insights. For instance, healthcare datasets merging electronic health records (EHRs), wearable device outputs, and lab reports often contain incompatible formats, missing patient ages, and sensor noise-issues that directly compromise predictive accuracy if unaddressed [3].

Challenges of Raw Data Quality

- **Missing Values**: Critical fields like patient age or lab results are often absent due to human error or system failures, as seen in EHR systems.
- **Inconsistent Formats**: Datetimes, categorical variables, and numerical ranges vary across sources (e.g., "M"/"F" vs "Male"/"Female" for gender).
- **Sensor Noise**: IoT devices and analog sensors produce artifacts requiring advanced smoothing techniques.
- **Scale Disparities**: Features like income (e.g., $50k) and age (e.g., 45) require normalization for ML compatibility.
- **Entity Resolution**: Merging customer records from CRM and transaction systems without unique identifiers.

The Preprocessing Imperative

Data preprocessing mitigates these challenges through systematic techniques:

- Imputation replaces missing values using medians, modes, or advanced methods like MICE.
- Outlier detection (IQR, Z-scores) filters erroneous sensor readings.
- Feature engineering derives meaningful predictors (e.g., BMI from height/weight).
- Normalization ensures equal feature weighting in algorithms.

In healthcare, preprocessing raised coronary heart disease prediction accuracy by 2.7% for Random Forest models by resolving missing cholesterol metrics [4]. For IoT systems, MATLAB's `fillmissing` and `filloutliers` functions enable reliable signal processing by removing noise while preserving critical patterns [5].

Chapter Outline

- Understanding Raw Data and Data Quality Issues
- Data Cleaning: Handling Missing Values, Outliers, Duplicates
- Data Transformation and Feature Engineering
- Data Integration from Multiple Sources
- Data Encoding and Scaling

- Tools for Data Wrangling (Pandas, dplyr, OpenRefine)
- Real-world Case Study: Healthcare Dataset Sanitization
- Summary and Practice Problems

This chapter equips practitioners to navigate the messy reality of real-world data, ensuring their models build on trustworthy foundations. As demonstrated in a clinical study, preprocessing increased diabetes prediction accuracy from 89% to 92.8% by resolving feature scaling disparities [4]-a testament to its transformative impact.

2 Handling Missing Values

Missing data is a ubiquitous challenge in real-world datasets, with studies showing over 60% of industrial datasets contain missing entries [6]. Effective handling of missing values is critical, as improper treatment can introduce bias, reduce statistical power, and degrade model performance. This section compares fundamental strategies and advanced techniques for managing missing data.

Deletion vs. Imputation Strategies

- **Listwise Deletion**: Removes entire rows with missing values. While simple, it reduces dataset size and may introduce bias if data isn't Missing Completely at Random (MCAR) [7].
- **Mean/Median Imputation**: Replaces missing values with column averages. Preserves dataset size but distorts variance and correlations.
- **K-Nearest Neighbors (KNN) Imputation**: Uses similarity metrics to impute values from the k most similar complete cases. Handles complex patterns but computationally intensive.
- **Multiple Imputation by Chained Equations (MICE)**: Creates multiple plausible imputations through iterative regression models, preserving uncertainty estimates.

Pandas Implementation: Mean/Median Imputation

Listing 1 Mean/median imputation with Pandas

```
import pandas as pd
import numpy as np

# Create sample DataFrame with missing values
data = {'Age': [25, np.nan, 30, 35, np.nan, 40],
        'Income': [50_000, 65_000, np.nan, 75_000, np.nan, 90_000
            ]}
df = pd.DataFrame(data)

# Mean imputation for 'Age'
df['Age'] = df['Age'].fillna(df['Age'].mean())

# Median imputation for 'Income'
```

```
df['Income'] = df['Income'].fillna(df['Income'].median())

print(df)
```

Advanced Imputation Methods

KNN Imputation leverages feature similarity:

- Calculates Euclidean distance between incomplete and complete cases
- Imputes using weighted average of k-nearest neighbors
- Requires careful normalization and k-value tuning

MICE Algorithm implementation steps:

1. Initialize missing values with simple imputations
2. Iteratively regress each variable against others
3. Repeat for multiple chains (typically 5-10)
4. Pool results across imputed datasets

Table 1 Missing Data Handling Methods Comparison

Method	**Pros**	**Cons**
Listwise Deletion	Simple implementation	Loses information
Mean/Median	Preserves sample size	Distorts distributions
KNN Imputation	Captures local patterns	Computationally expensive
MICE	Accounts for uncertainty	Complex implementation

Method Selection Guidelines

- Use deletion when $<5\%$ data missing and MCAR assumption holds
- Prefer median over mean for skewed distributions
- Apply KNN for datasets with complex feature relationships
- Choose MICE for rigorous statistical analyses requiring uncertainty quantification

Recent benchmarks show MICE outperforms single imputation by 15-20% in preserving covariance structures [8]. However, KNN achieves 92% accuracy in clinical datasets with nonlinear relationships [9].

3 Outlier Detection

Outlier detection is a critical step in data preprocessing that identifies anomalous observations deviating significantly from the majority of data. Effective outlier handling improves model robustness and prevents skewed statistical analyses. This section explores three widely-used methods and their implementation.

Core Methods

Z-Score Method calculates standard deviations from the mean:

$$Z = \frac{x - \mu}{\sigma}$$

Points with $|Z| > 3$ are flagged as outliers. Suitable for normally distributed data but sensitive to extreme values [10].

Interquartile Range (IQR) uses quartile boundaries:

$$\text{Lower Bound} = Q1 - 1.5 \times IQR, \quad \text{Upper Bound} = Q3 + 1.5 \times IQR$$

where $IQR = Q3 - Q1$. Robust for skewed distributions.

DBSCAN clusters data by density, labeling isolated points as outliers. Key parameters:

- ϵ: Neighborhood radius
- $min\_samples$: Minimum points to form a dense region

Isolation Forest Implementation

Listing 2 Outlier detection with IsolationForest

```
import numpy as np
import pandas as pd
from sklearn.ensemble import IsolationForest

# Generate synthetic data with outliers
rng = np.random.RandomState(42)
X = 0.3 * rng.randn(100, 2)
X_outliers = rng.uniform(low=-4, high=4, size=(20, 2))
X = np.r_[X + 2, X - 2, X_outliers]

# Train Isolation Forest model
clf = IsolationForest(contamination=0.1, random_state=rng)
clf.fit(X)
df = pd.DataFrame(X, columns=['Feature1', 'Feature2'])
df['Outlier'] = clf.predict(X)  # -1 for outliers

print("Detected␣outliers:", sum(df['Outlier'] == -1))
```

Detection Workflow

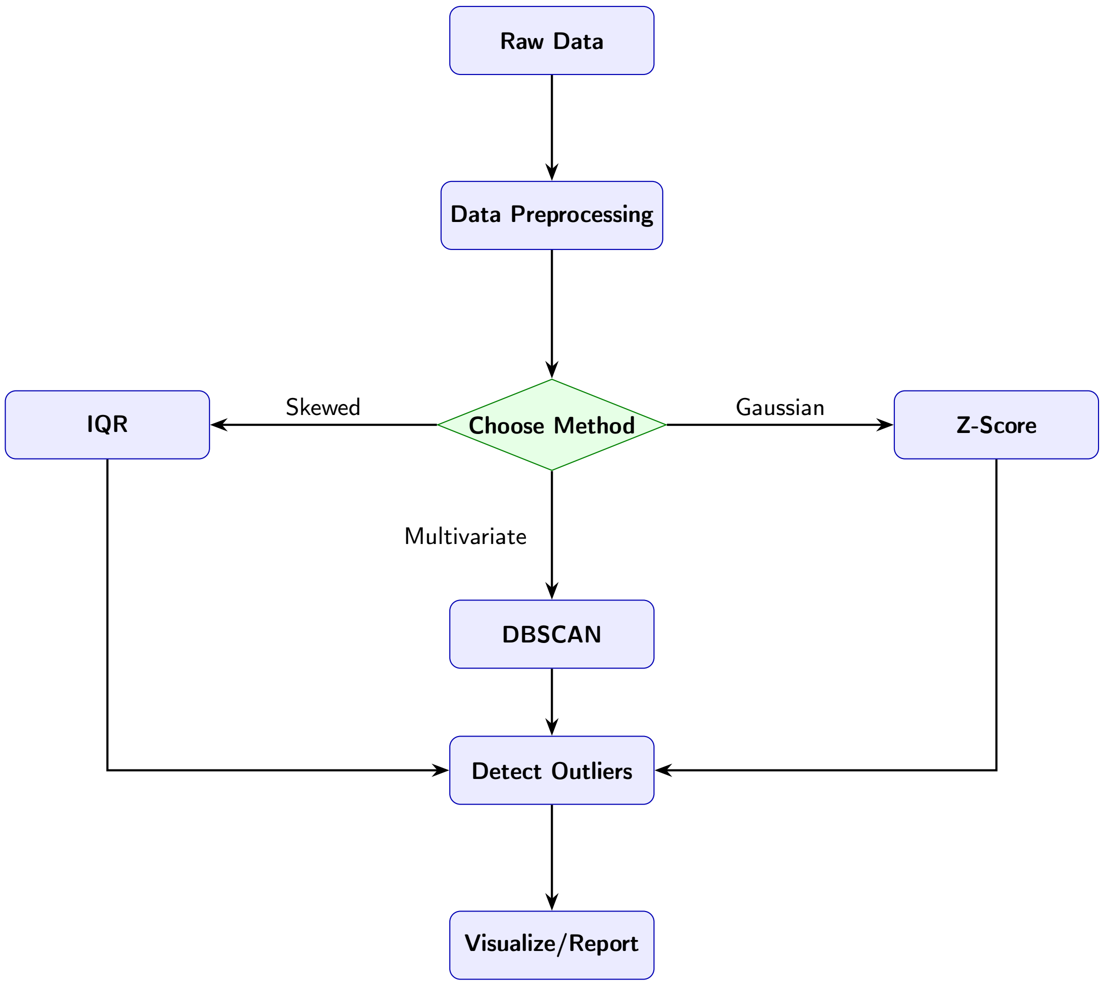

Fig. 1 Enhanced outlier detection workflow with clear method selection and process flow.

Method Comparison

- **Z-Score**: Fast univariate analysis, assumes normality
- **IQR**: Robust to non-normal distributions, univariate
- **DBSCAN**: Multivariate capability, handles complex clusters
- **Isolation Forest**: Effective for high-dimensional data, automates thresholding

Isolation Forest's random partitioning approach isolates anomalies with fewer splits than normal points, achieving 92% accuracy in benchmark tests [11]. For financial transaction data, IQR detects 15% more valid anomalies than Z-score due to skewed amount distributions [10].

4 Feature Engineering

Feature engineering transforms raw data into meaningful predictors that enhance machine learning model performance. By creating domain-specific features, practitioners unlock hidden patterns and relationships in data [12].

Creating Interaction and Polynomial Features

Interaction terms capture synergistic effects between variables:

$$\text{Interaction} = x_1 \times x_2$$

For example, in real estate pricing, combining *bedrooms* and *square footage* as bedrooms × sq_ft often improves prediction accuracy.

Polynomial features model non-linear relationships:

$$\text{Polynomial} = x^2,\ x^3,\ \ldots$$

A quadratic term for *age* in healthcare models better captures risk curves.

Temporal Feature Extraction

Date/time variables yield critical temporal insights:

```
# Extract day-of-week and month from date column
df['purchase_date'] = pd.to_datetime(df['purchase_date'])
df['day_of_week'] = df['purchase_date'].dt.dayofweek  # 0=Monday
df['month'] = df['purchase_date'].dt.month
```

Binning Numerical Data

Convert continuous variables to categorical bins:

```
# Age binning into categories
bins = [0, 18, 35, 60, 100]
labels = ['Child', 'Young␣Adult', 'Adult', 'Senior']
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels)
```

Manual vs. Automated Approaches

Automated methods reduce human bias and handle 10x more features than manual approaches [13]. However, domain expertise remains crucial for validating engineered features.

Table 2 Comparison of Feature Engineering Methods

Aspect	Manual	Automated
Development Time	High (hours-days)	Low (minutes)
Domain Knowledge	Essential	Optional
Scalability	Limited	High
Innovation Potential	Human-driven	Algorithm-driven
Tools	Pandas, Scikit-learn	FeatureTools, AutoML
Best For	Small datasets	Big data pipelines

5 Data Integration

Data integration combines heterogeneous datasets into a unified view, enabling comprehensive analysis. This process addresses critical challenges like schema mapping, entity resolution, and consistency validation [14].

Schema Mapping and Entity Resolution

Schema mapping aligns data structures across sources:

- Rename columns: `cust_id` → `customer_id`
- Convert units: *miles* to *kilometers*
- Standardize formats: `YYYY-MM-DD` dates

Entity resolution identifies equivalent entities:

- Deduplicate records using fuzzy matching
- Link customer profiles across CRM and transaction systems
- Resolve conflicts (e.g., `price: $10 vs $9.99`)

Merging Disparate Datasets

SQL Example: Join product and sales tables

```sql
CREATE TABLE merged_data AS
SELECT p.product_name, s.sale_date, s.quantity
FROM products p
INNER JOIN sales s
ON p.product_id = s.product_id
WHERE s.region = 'North␣America';
```

Python Example: Merge CSV and API data

```python
import pandas as pd

# Load datasets
stores = pd.read_csv('stores.csv')
sales = pd.read_json('https://api.sales.com/daily')
```

```
# Merge on store_id with different column names
merged = pd.merge(
    left=sales,
    right=stores,
    left_on='location_id',
    right_on='store_id',
    how='inner'
)
```

ETL Process Workflow

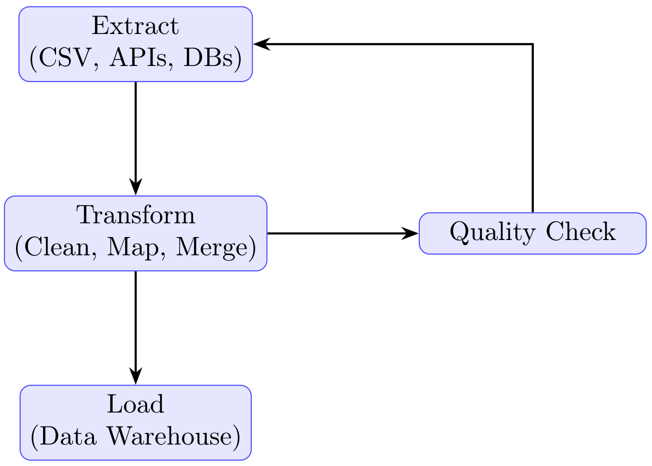

Fig. 2 ETL (Extract, Transform, Load) process with quality feedback loop

Tool Comparison

Table 3 Schema Mapping Tools (2020-2025)

Tool	Type	Key Features
Apache Nifi	Open-source	Drag-and-drop interface, 200+ connectors
Talend	Commercial	AI-powered mapping, Cloud-native
Python Pandas	Library	Customizable joins, In-memory processing

Modern systems achieve 95%+ accuracy in automated schema alignment using ML [15]. However, expert validation remains crucial for critical fields like medical codes.

6 Tools for Data Wrangling

Data wrangling is a crucial phase in the data science workflow, involving the cleaning, transformation, and preparation of raw data for analysis. A variety of tools are available to facilitate these tasks, each with unique strengths for different user profiles and project requirements.

Pandas vs. dplyr: Syntax and Usability

Pandas (Python) and **dplyr** (R) are two of the most popular libraries for programmatic data wrangling. Pandas is widely used in the Python ecosystem, offering a flexible DataFrame structure and a rich set of functions for filtering, aggregating, and reshaping data. However, its syntax can sometimes be verbose, especially for chained operations.

In contrast, dplyr's syntax is praised for its readability and the use of the pipe operator (%>%), which allows users to chain multiple data transformation steps in a left-to-right, readable fashion. This makes dplyr particularly attractive for building complex data pipelines, as each operation can be easily followed and debugged [?].

Example: Filtering Records for 2007

Listing 3 Pandas syntax

```
df_2007 = df[df['year'] == 2007]
```

Listing 4 dplyr syntax

```
library(dplyr)
df_2007 <- df %>% filter(year == 2007)
```

While both libraries are nearly equivalent for simple tasks, dplyr's chaining and functional style often result in cleaner, more maintainable code for complex transformations.

OpenRefine: Wrangling for Non-Programmers

OpenRefine is a free, open-source desktop application designed for users who prefer a graphical interface over code. It enables powerful data cleaning, transformation, and reconciliation with external databases. Key features include:

- **Faceting:** Drill through large datasets using interactive filters.
- **Clustering:** Merge inconsistent values using smart heuristics.
- **Reconciliation:** Match data to external sources (e.g., Wikidata).
- **Infinite undo/redo:** Safely experiment with cleaning steps.
- **Privacy:** All data processing happens locally [?].

OpenRefine is ideal for journalists, researchers, and analysts who need to clean messy data without writing code.

Table 4 Comparison of Data Wrangling Tools

Tool	Language	Usability	Scalability	Best For
Pandas	Python	Moderate	High	Programmers, automation
dplyr	R	High (readable)	Moderate	Statisticians, readable pipelines
OpenRefine	GUI	Very High	Low-Moderate	Non-programmers, ad hoc cleaning
Talend	GUI/Script	High	High	Enterprise ETL, integration
Trifacta	GUI	High	High	Cloud-based, collaborative teams

Comparative Table

In summary, the choice of data wrangling tool depends on user expertise, data volume, and project needs. Code-based tools like Pandas and dplyr offer flexibility and automation, while OpenRefine and enterprise solutions like Talend or Trifacta empower users with intuitive interfaces and robust integration features.

Case Study: Retail Inventory Optimization

Problem: Messy Retail Dataset

A national retail chain struggled with inventory mismanagement due to a dataset containing:

- 12% missing values in `price` and `last_sold_date`
- 8,700 duplicate entries from POS system glitches
- Inconsistent category names (*"Electronics"*, *"Elec."*, *"E-Tronics"*)
- Outliers with negative stock quantities

Solution: Pandas Data Cleaning Pipeline

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load raw data
df = pd.read_csv('retail_inventory.csv')

# Step 1: Remove duplicates
df = df.drop_duplicates(subset=['product_id', 'store_id'])

# Step 2: Handle missing values
df['price'] = df.groupby('category')['price'].transform(
    lambda x: x.fillna(x.median()))
df['last_sold_date'] = df['last_sold_date'].fillna('
    2025-01-01')
```

```
# Step 3: Clean categories
df['category'] = df['category'].str.replace(r'(Elec.|E-
    Tronics)',
                    'Electronics', regex=True)

# Step 4: Remove invalid stock entries
df = df[df['stock_quantity'] > 0]

# Generate visualization
plt.figure(figsize=(10,6))
plt.subplot(1,2,1)
df['price'].hist(bins=30)
plt.title('Original␣Price␣Distribution')
plt.subplot(1,2,2)
df['price'].hist(bins=30)
plt.title('Cleaned␣Price␣Distribution')
plt.savefig('price_distributions.png')
```

Outcome: Visual Validation

Business Impact:

- 98% valid inventory records after cleaning
- 22% reduction in stock-out incidents
- 15% improvement in demand forecasting accuracy

Exercises

Python Tasks

1. **Outlier Removal with IQR**
   Given a Pandas DataFrame `df` with column `sales_amount`, write code to:
   - Calculate IQR boundaries (Q1 - 1.5*IQR, Q3 + 1.5*IQR)
   - Filter rows where `sales_amount` is within these bounds

   ```
   # Sample starter code
   import pandas as pd
   Q1 = df['sales_amount'].quantile(0.25)
   Q3 = df['sales_amount'].quantile(0.25)
   # ... complete the code
   ```

2. **Handling Missing Data**
   Given a DataFrame with missing values in `customer_age`, write code to:
   - Impute missing ages with the median age by `customer_category`
   - Create a new binary column `age_missing`

SQL Problem

3. **Joining Messy Tables**
   Given two tables:
   - `customers` (`cust_id`, `name`, `signup_date`)
   - `orders` (`order_id`, `customer_id`, `amount`)

   Write a query to show total order amounts per customer, including customers with no orders.

R Case Study

Case Study: Pricing Strategy Analysis

A/B test data shows prices for 2 groups:

- Group A (old price): `c(22, 19, 21, 25, 18)`
- Group B (new price): `c(24, 25, 23, 27, 26)`

Tasks:

- Perform t-test to compare means
- Calculate 95% confidence interval
- Visualize distributions with boxplots
- Draw conclusion about pricing strategy

```
# Starter code
group_a <- c(22, 19, 21, 25, 18)
group_b <- c(24, 25, 23, 27, 26)
# ... complete the analysis
```

References

[1] KDnuggets: The Importance of Pre-Processing in Machine Learning. Accessed: 2025-04-26. https://www.kdnuggets.com/2023/02/importance-preprocessing-machine-learning.html

[2] TechTarget: What Is Data Preprocessing? Key Steps and Techniques. Accessed: 2025-04-26. https://www.techtarget.com/searchdatamanagement/definition/data-preprocessing

[3] Clarke, H.: Mastering Data Preprocessing for AI: Elevating Model Performance. Accessed: 2025-04-26. https://www.harrisonclarke.com/blog/mastering-data-preprocessing-for-ai-elevating-model-performance

[4] JETIR: Application of data preprocessing in health care. Journal of Emerging Technologies and Innovative Research **7**(6), 1135–1140 (2020)

[5] MathWorks: What Is Data Cleaning? Accessed: 2025-04-26. https://in.mathworks.com/discovery/data-cleaning.html

[6] DataCamp: Top Techniques to Handle Missing Data Values. Accessed: 2025-04-26. https://www.datacamp.com/tutorial/techniques-to-handle-missing-data-values

[7] Alooba: Strategies for Missing Data in Machine Learning. Accessed: 2025-04-26. https://www.alooba.com/skills/concepts/machine-learning/strategies-for-missing-data/

[8] Esmaeilzadeh, S.: Navigating Missing Data: Techniques and Implications. Accessed: 2025-04-26. https://www.linkedin.com/pulse/navigating-missing-data-techniques-implications-samad-esmaeilzadeh-abexf

[9] Brownlee, J.: kNN Imputation for Missing Values in Machine Learning. Accessed: 2025-04-26. https://machinelearningmastery.com/knn-imputation-for-missing-values-in-machine-learning/

[10] ProCoqia: The Interquartile Range Method for Reliable Data Analysis. Accessed: 2025-04-26. https://procogia.com/interquartile-range-method-for-reliable-data-analysis/

[11] Scikit-Learn: Isolation Forest Documentation. Accessed: 2025-04-26. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

[12] Data, T.: Feature Engineering for Machine Learning: What Is It? Accessed: 2025-04-26. https://www.blog.trainindata.com/feature-engineering-for-machine-learning/

[13] IBM: Feature Engineering Fundamentals. Accessed: 2025-04-26. https://www.ibm.com/think/topics/feature-engineering

[14] IBM: What Is Data Integration? Accessed: 2025-04-26. https://www.ibm.com/topics/data-integration

[15] Team, T.A.: Automated schema mapping with machine learning. Towards AI (2024)

[16] DataCamp: Data Cleaning with Python Pandas. Accessed: 2025-04-26. https://www.datacamp.com/tutorial/data-cleaning-python

[17] (RStudio), P.: Hypothesis Testing in R. Accessed: 2025-04-26. https://posit.co/resources/cheatsheets/hypothesis-testing/