

# **Advanced Data Modelling (CI6320) Data Preprocessing**

Identify the general data mining principles and techniques, and to be able to apply them in different contexts. .

**Bhimaja C. Goonatillaka**

(MSc, BSc, PGDE, PGD in CS, PGD in IT, MCS(SL), MIEEE)

# Purpose of Preprocessing

- Real-world data is often messy. It can be missing, inaccurate, and inconsistent for various reasons, including missing entries, recording errors, data limitations, and inconsistencies in coding or format.
- Dirty data can lead to unreliable analysis and mislead data mining efforts. Users may distrust the results, and algorithms may malfunction due to the inaccurate information.

# Solution and Benefits of Preprocessing

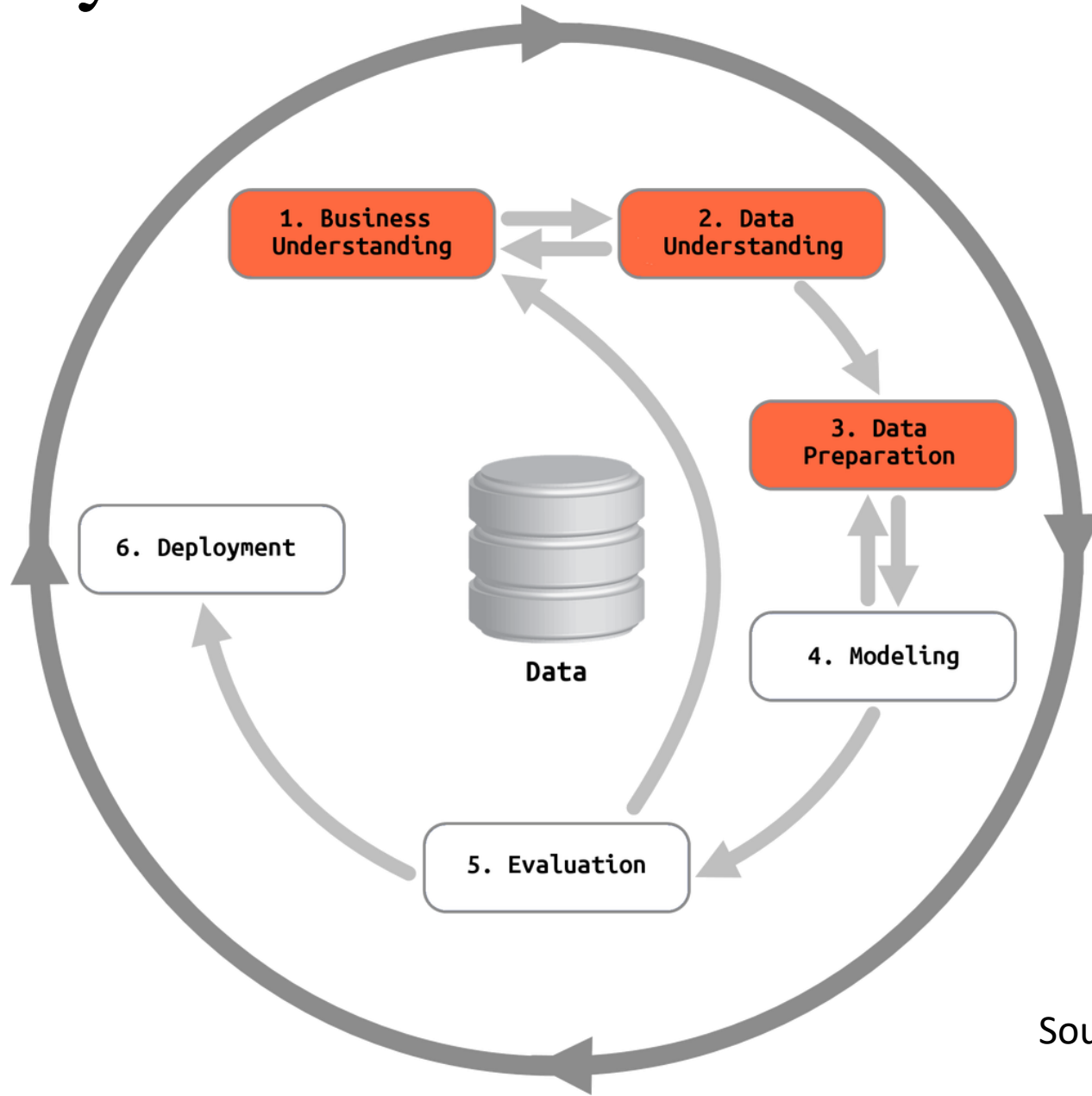
Data preprocessing is crucial. It involves cleaning the data by:

- Filling in missing values
- Smoothing noisy data
- Identifying and removing outliers
- Resolving inconsistencies

Clean data leads to:

- More accurate and reliable analysis
- More trustworthy results
- Improved performance of data mining algorithms

# The Life Cycle of KDD Process



Source: datacyper.com

# Data Cleaning

Real-world data often has missing values, which can hinder analysis. Data cleaning techniques address this by filling in missing values using various strategies:

- Ignore the tuple: Remove entries with missing data, but this can be inefficient if many values are missing.
- Manually fill in values: Time-consuming and often impractical for large datasets.
- Use a global constant: Replace missing values with a placeholder like "Unknown," but this can mislead analysis.
- Use the attribute mean: Fill in missing values with the average value for that attribute across the dataset.
- Use class-specific mean: If classifying data, fill in missing values with the average value for that attribute within the same class.
- Use the most probable value: Employ techniques like regression, Bayesian inference, or decision trees to predict missing values based on other attributes.

# Choosing the Best Method

- The optimal approach depends on the specific dataset and analysis goals.
- Consider factors like the amount of missing data, the importance of the attribute, and the potential impact of different methods on analysis outcomes.

# Noisy Data

Data often contains noise, which refers to random errors or variations that can distort analysis. Data smoothing techniques reduce noise and make underlying patterns more visible. Common methods include:

## 1. Binning:

- Sorts data and divides it into equal-sized bins (groups).
- Smooths values by considering their "neighborhood" within bins.

Techniques include:

- Smoothing by bin means: Replaces each value with the mean of its bin.
- Smoothing by bin medians: Replaces each value with the median of its bin.
- Smoothing by bin boundaries: Replaces each value with the closest bin boundary.

## 2. Regression

- Fits a function to the data to capture the overall trend and reduce noise.
- Linear regression finds the best-fitting line to represent the relationship between two attributes.
- Multiple linear regression extends this to multiple attributes, fitting a multidimensional surface.



# 3. Clustering

- Groups similar values together, identifying outliers that fall outside of clusters.
- Outliers can be considered noise and removed or investigated further.

# Choosing the Best Method

The optimal data smoothing technique depends on the nature of the noise, the type of data, and the analytical goals.

Consider factors like,

- the distribution of the noise,
- the importance of preserving relationships between variables, and
- the intended use of the smoothed data.

# Data Cleaning as a Process

Data quality can be hampered by:

- Missing values: Gaps in data due to various reasons like optional fields, data entry errors, or deliberate omissions.
- Noise: Random errors or fluctuations that distort the true signal.
- Inconsistencies: Discrepancies in data formats, codes, representations, and rules.

# Discrepancy Detection

Metadata analysis: Understanding data properties like domain, types, acceptable values, and dependencies.

Checking for:

- Inconsistent code usage and data representations.
- Field overloading.
- Violations of unique, consecutive, and null rules.
- Reasons for missing values.

# Data Transformation

Correcting discrepancies through automated tools or custom scripts.

An iterative process with potential pitfalls:

- Introducing new discrepancies.
- Nested discrepancies revealed later.
- Lack of interactivity and feedback during batch processing.

# New Approaches for Improved Data Cleaning

- Interactive tools: Allow users to build and refine transformations gradually on a visual interface.
- Declarative languages: Enable efficient specification of data cleaning operators.
- Metadata updates: Continuously improve data knowledge for future use.

# Data Integration

Data analysis often involves:

- Data from multiple sources: Databases, data cubes, flat files, etc.
- Data integration: Combining these sources into a coherent data store like a data warehouse.

# Challenges in Data Integration

- Schema integration and object matching: Mapping equivalent entities from different sources (e.g., customer ID vs. cust number).
- Metadata: Using data about data (name, type, range, null rules) to avoid errors in integration and data cleaning.
- Redundancy: Identifying and removing duplicate or derived attributes.



# Redundancy Detection

- Correlation analysis: Measuring how strongly one attribute implies another (numerical attributes only).
- Correlation coefficient: A value between -1 and 1 quantifying the correlation strength.
- Interpretation:
  - $r > 0$ : positive correlation (increasing values together).
  - $r = 0$ : no correlation.
  - $r < 0$ : negative correlation (one increases as the other decreases).

High correlation may indicate one attribute can be removed as redundant.

# Data Transformation

Data transformation prepares data for analysis by:

- Smoothing: Removing noise using techniques like binning, regression, and clustering (previously discussed).
- Aggregation: Computing summary statistics like daily sales to monthly or annual totals. Used for creating data cubes with multi-level analysis.
- Generalization: Replacing raw data with higher-level concepts. E.g., street addresses to city or country, or age values to youth, middle-aged, and senior. Discussed later in data cube computation.

# Normalization

Scaling data to a specific range like -1:1 or 0:1. Two common methods:

- Min-max normalization: Maps data to a new range based on minimum and maximum values.
- Z-score normalization: Standardizes data based on mean and standard deviation. Useful when minimum/maximum are unknown or outliers or the distribution is skewed.

# Benefits of Data Transformation

- Improved data quality for better analysis.
- Efficient storage and processing of data.
- Enhanced ability to discover patterns and insights.

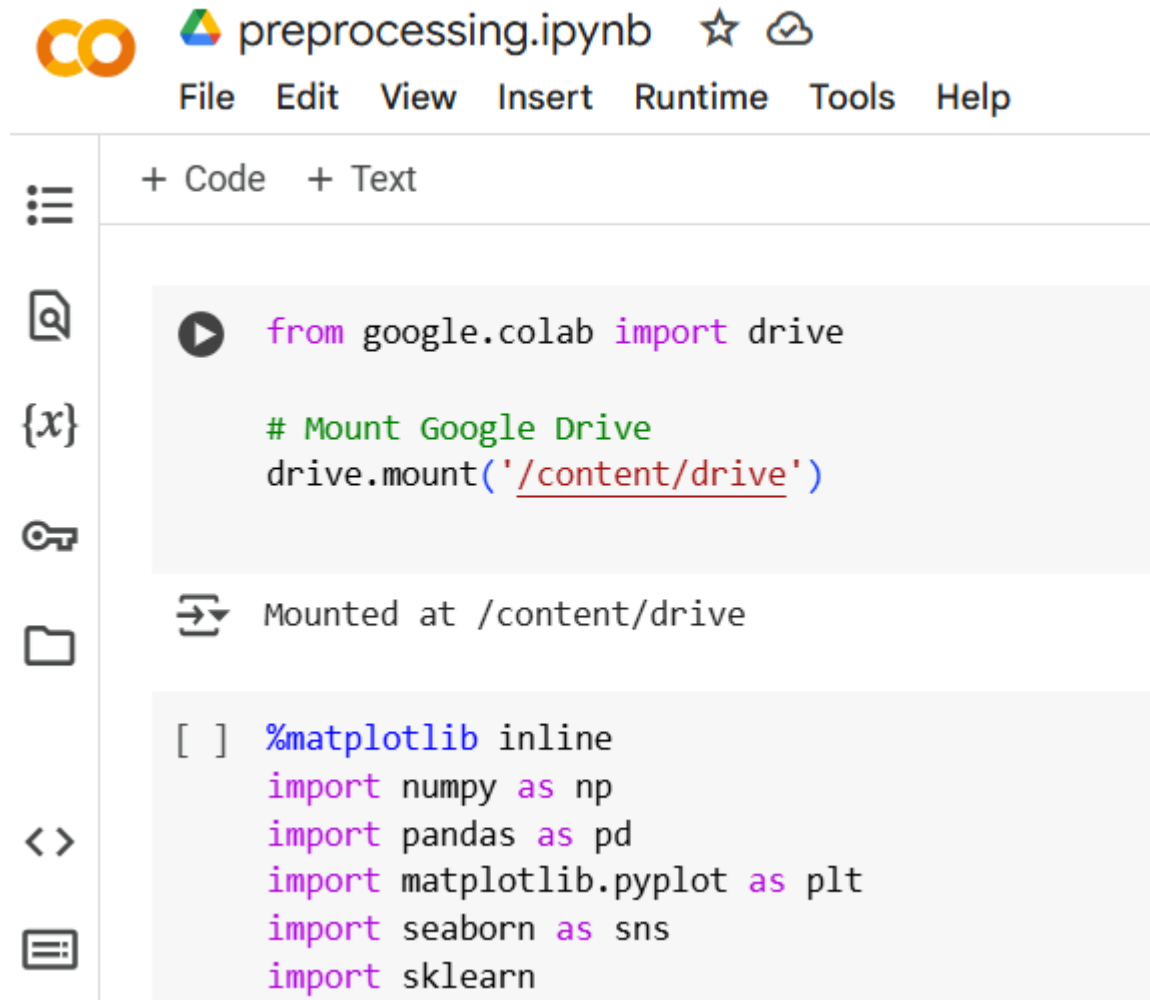
# Summary

- Purpose of Preprocessing
- Solution and Benefits of Preprocessing
- The Life Cycle of KDD Process
- Data Cleaning
- Choosing the Best Method
- Data Transformation
- Data Integration

# Thank You

- Qs and As

# Practical



The image shows a Google Colab interface. At the top, there's a header with the Colab logo, the file name 'preprocessing.ipynb', and icons for star and cloud. Below this is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. On the left side, there's a sidebar with icons for a menu, search, variables, keys, files, and a console. The main area contains two code cells. The first cell has a play button icon and contains code to import 'drive' from 'google.colab' and mount it. The second cell has a console icon and contains code to import various libraries like numpy, pandas, matplotlib, seaborn, and sklearn.

co preprocessing.ipynb ☆ ☁

File Edit View Insert Runtime Tools Help

+ Code + Text

```
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
```