# Week 2
# Input: dataset, features

# Content

- Machine Learning Workflow
- Data Mining Overview
- Data Preprocessing
- Data Cleaning Techniques
- Data Reduction
- Data Transformation
- Advanced Data Handling and Preprocessing
- Advanced Data Cleaning Techniques
- Association Rules
- Predictors, Features
- Feature Engineering and Selection
- Handling Imbalanced Data

# Machine Learning Workflow

What do you mean by

**Input**

**Cat**

**Training**

**AI Models**

**Learning**

**Output**

Cats' Features:

- Fur Color
- Eye Color
- Ear Shape
- Tail Length
- Whiskers

Doraemon's Features:

- Blue Body
- Bell Around Neck
- Pocket on Stomach
- Round Face
- Propeller Hat

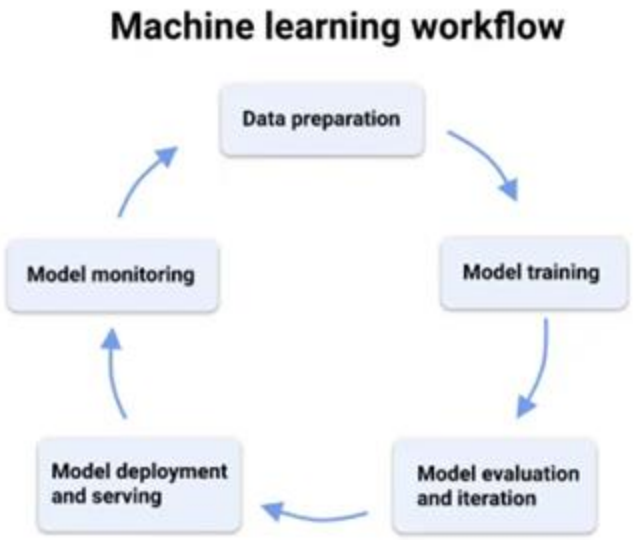
Lucky Cats' Features:

- Raised Paw
- Coin or Fish
- Base or Platform
- Expression
- Color

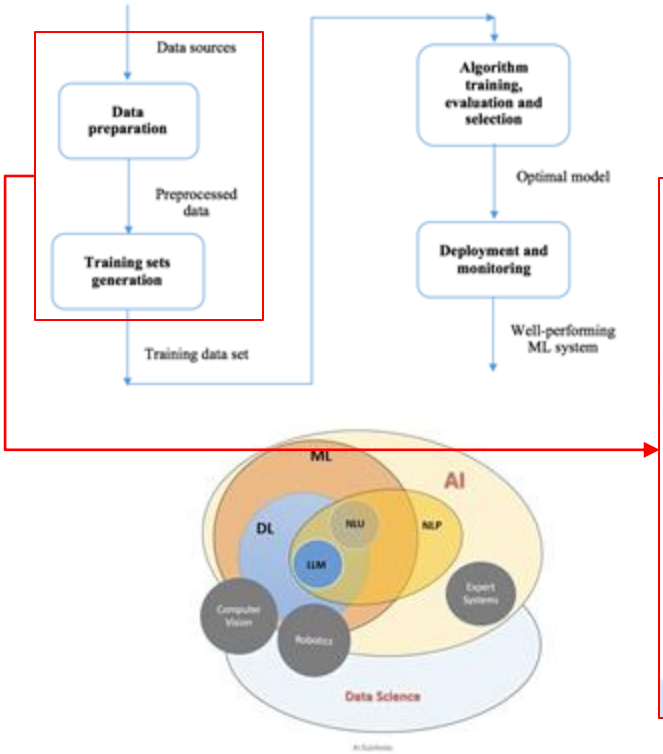
# Machine Learning Workflow

## Input → Model → Output

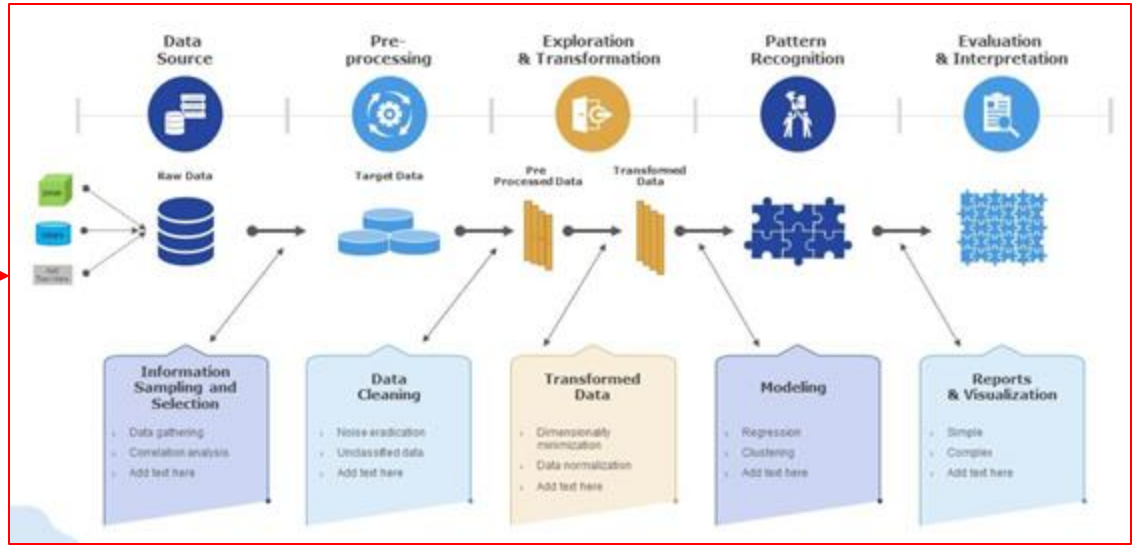
- **Input:** These are the features or attributes provided to the model. They can be numerical, categorical, or other types of data.
- **Model:** The model processes the input data using a specific algorithm or architecture. It learns patterns and relationships from the input to make predictions.
- **Output:** The model produces an output based on the input.
    - For example, in a classification task, the output could be a class label (e.g., spam or not spam), while in regression, it could be a numerical value (e.g., predicting house prices).



# Data Mining Overview




## Data Science, Data Mining



# Data Mining Overview

1. **Data Source**:
    - **Definition**: Data sources refer to the origin of data used for analysis. They can be internal (e.g., company databases, logs) or external (e.g., public datasets, APIs).
    - **Importance**: High-quality data is essential for accurate insights. Ensure data relevance, accuracy, and completeness.
    - **Examples**: Customer transaction records, sensor data, social media posts.
2. **Data Processing**:
    - **Purpose**: Data processing prepares raw data for analysis.
    - **Steps**:
        - *Data Cleaning*: Remove inconsistencies, handle missing values, and correct errors.
        - *Data Transformation*: Normalize, scale, or encode features.
    - **Quality Assurance**: Validate data quality to avoid biased results.
3. **Exploration and Transformation**:
    - **Exploratory Data Analysis (EDA)**:
        - Visualize data (histograms, scatter plots).
        - Calculate summary statistics (mean, variance).
    - **Feature Engineering**:
        - Create new features from existing ones (e.g., age groups from birthdates).
        - Feature selection (choosing relevant features).

# Data Mining Overview

4. **Pattern Recognition**:
    - **Machine Learning Models**:
        - Classification (predicting categories).
        - Regression (predicting continuous values).
        - Clustering (grouping similar data points).
    - **Model Training and Evaluation**:
        - Train models using historical data.
        - Evaluate performance (accuracy, precision, recall).
5. **Evaluation and Interpretation**:
    - **Metrics**:
        - Accuracy: Correct predictions divided by total predictions.
        - Precision: True positives divided by true positives plus false positives.
        - Recall: True positives divided by true positives plus false negatives.
    - **Interpretation**:
        - Understand model decisions (feature importance, coefficients).
        - Iterate based on insights.

# Data Mining Overview

**Q: why do we need data mining?**

- **Knowledge Discovery**:
  - Data mining helps uncover hidden patterns, relationships, and trends within data.
  - By analyzing historical data, we gain new knowledge that informs decision-making.
- **Business Intelligence**:
  - Organizations use data mining to enhance business strategies.
  - It aids in understanding customer behavior, optimizing processes, and identifying growth opportunities.
- **Predictive Modeling**:
  - Data mining builds predictive models based on historical data.
  - These models help forecast future outcomes, such as sales, stock prices, or disease outbreaks.
- **Risk Assessment**:
  - Data mining assesses risks by analyzing historical incidents.
  - For instance, credit scoring models predict creditworthiness based on past behavior.
- **Personalization**:
  - Companies personalize recommendations (e.g., Amazon product suggestions) using data mining.
  - It tailors experiences to individual preferences.
- **Scientific Research**:
  - Researchers analyze data to validate hypotheses and discover new scientific insights.
    - Fields like genomics, climate science, and social sciences benefit from data mining.

# Data Preprocessing

- **What is Data Preprocessing?**

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

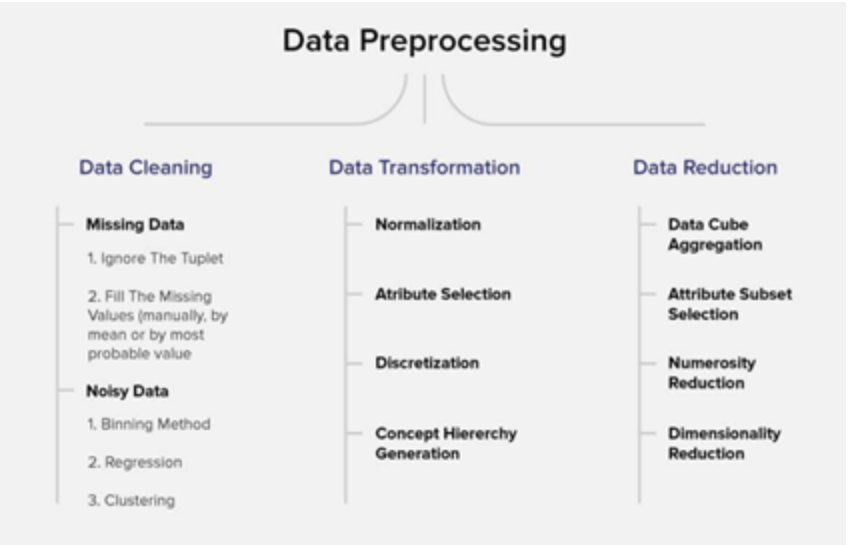
- **Why is Data Preprocessing Important?**

Preprocessing of data is mainly to check the data quality. The quality can be checked by the following:

- Accuracy: To check whether the data entered is correct or not.
- Completeness: To check whether the data is available or not recorded.
- Consistency: To check whether the same data is kept in all the places that do or do not match.
- Timeliness: The data should be updated correctly.
- Believability: The data should be trustable.
- Interpretability: The understandability of the data.

# Data Preprocessing

- **Major Tasks in Data Preprocessing**
  - **Data Cleaning**
    - **Missing Data**
      - Ignore The Tuple
      - Fill The Missing Values (Manually, by mean, or by most probable value)
    - **Noisy Data**
      - Binning Method
      - Regression
    - **Clustering**
  - **Data Transformation**
    - **Normalization**
    - **Attribute Selection**
      - Concept Hierarchy Generation
  - **Data Reduction**
    - **Data Cube Aggregation**
    - **Attribute Subset Selection**
    - **Numerosity Reduction**
    - **Dimensionality Reduction**



# Data Cleaning Techniques

Data cleaning is the process of removing incorrect data, incomplete data, and inaccurate data from the datasets, and it also replaces the missing values. Here are some techniques for data cleaning:

**Handling Missing Values**

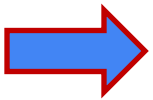
- Standard values like “Not Available” or “NaA” can be used to replace the missing values.
- Missing values can also be filled manually, but it is not recommended when that dataset is big.
- The attribute’s mean value can be used to replace the missing value when the data is normally distributed where in the case of non-normal distribution median value of the attribute can be used.
- While using regression or decision tree algorithms, the missing value can be replaced by the most probable value.

| PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 62 | 1 | 1 | Icard, Miss. Amelie | female | 38.0 | 0 | 0 | 113572 | 80.0000 | B28 | NaN |
| 830 | 1 | 1 | Stone, Mrs. George Nelson (Martha Evelyn) | female | 62.0 | 0 | 0 | 113572 | 80.0000 | B28 | NaN |
| 6 | 0 | 3 | Moran, Mr. James | male | NaN | 0 | 0 | 330877 | 8.4583 | NaN | Q |
| 18 | 1 | 2 | Williams, Mr. Charles Eugene | male | NaN | 0 | 0 | 244373 | 13.0000 | NaN | S |
| 20 | 1 | 3 | Masselmani, Mrs. Fatima | female | NaN | 0 | 0 | 2649 | 7.2250 | NaN | C |
| 27 | 0 | 3 | Emir, Mr. Farred Chehab | male | NaN | 0 | 0 | 2631 | 7.2250 | NaN | C |
| 29 | 1 | 3 | O'Dwyer, Miss. Ellen "Nellie" | female | NaN | 0 | 0 | 330959 | 7.8792 | NaN | Q |
| 30 | 0 | 3 | Todoroff, Mr. Lalio | male | NaN | 0 | 0 | 349216 | 7.8958 | NaN | S |
| 32 | 1 | 1 | Spencer, Mrs. William Augustus (Marie Eugenie) | female | NaN | 1 | 0 | PC 17569 | 146.5208 | B78 | C |
| 33 | 1 | 3 | Glynn, Miss. Mary Agatha | female | NaN | 0 | 0 | 335677 | 7.7500 | NaN | Q |

# Data Cleaning Techniques

- **Replacing with previous value – Forward fill**

| | Date | Temperature | Humidity | WindSpeed | Event |
|---|---|---|---|---|---|
| 0 | 9/1/23 | 46.85 | 55.90 | 0.39 | Snow |
| 1 | 9/2/23 | 72.78 | 45.50 | 23.21 | Rain |
| 2 | 9/3/23 | 62.94 | 51.40 | 10.70 | Snow |
| 3 | 9/4/23 | 56.94 | 61.36 | 24.17 | Rain |
| 4 | 9/5/23 | 37.02 | 80.90 | 24.09 | NaN |
| 5 | 9/6/23 | NaN | NaN | 21.33 | Rain |
| 6 | 9/7/23 | NaN | 40.35 | 7.36 | Sunny |
| 7 | 9/8/23 | 68.98 | 65.54 | NaN | Snow |
| 8 | 9/9/23 | 57.05 | 60.87 | 21.28 | NaN |
| 9 | 9/10/23 | 61.86 | 51.11 | 7.92 | Snow |
| 10 | 9/11/23 | 30.93 | 45.99 | 4.24 | Snow |
| 11 | 9/12/23 | 73.65 | 56.88 | 13.92 | NaN |
| 12 | 9/13/23 | 67.46 | 87.15 | 23.40 | Sunny |

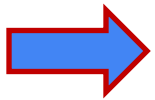


| | Date | Temperature | Humidity | WindSpeed | Event |
|---|---|---|---|---|---|
| 0 | 9/1/23 | 46.85 | 55.90 | 0.39 | Snow |
| 1 | 9/2/23 | 72.78 | 45.50 | 23.21 | Rain |
| 2 | 9/3/23 | 62.94 | 51.40 | 10.70 | Snow |
| 3 | 9/4/23 | 56.94 | 61.36 | 24.17 | Rain |
| 4 | 9/5/23 | 37.02 | 80.90 | 24.09 | Rain |
| 5 | 9/6/23 | 37.02 | 80.90 | 21.33 | Rain |
| 6 | 9/7/23 | 37.02 | 40.35 | 7.36 | Sunny |
| 7 | 9/8/23 | 68.98 | 65.54 | 7.36 | Snow |
| 8 | 9/9/23 | 57.05 | 60.87 | 21.28 | Snow |
| 9 | 9/10/23 | 61.86 | 51.11 | 7.92 | Snow |
| 10 | 9/11/23 | 30.93 | 45.99 | 4.24 | Snow |
| 11 | 9/12/23 | 73.65 | 56.88 | 13.92 | Snow |
| 12 | 9/13/23 | 67.46 | 87.15 | 23.40 | Sunny |

# Data Cleaning Techniques

- **Replacing with next value – Backward fill**

| | Date | Temperature | Humidity | WindSpeed | Event |
|---|---|---|---|---|---|
| 0 | 9/1/23 | 46.85 | 55.90 | 0.39 | Snow |
| 1 | 9/2/23 | 72.78 | 45.50 | 23.21 | Rain |
| 2 | 9/3/23 | 62.94 | 51.40 | 10.70 | Snow |
| 3 | 9/4/23 | 56.94 | 61.36 | 24.17 | Rain |
| 4 | 9/5/23 | 37.02 | 80.90 | 24.09 | NaN |
| 5 | 9/6/23 | NaN | NaN | 21.33 | Rain |
| 6 | 9/7/23 | NaN | 40.35 | 7.36 | Sunny |
| 7 | 9/8/23 | 68.98 | 65.54 | NaN | Snow |
| 8 | 9/9/23 | 57.05 | 60.87 | 21.28 | NaN |
| 9 | 9/10/23 | 61.86 | 51.11 | 7.92 | Snow |
| 10 | 9/11/23 | 30.93 | 45.99 | 4.24 | Snow |
| 11 | 9/12/23 | 73.65 | 56.88 | 13.92 | NaN |
| 12 | 9/13/23 | 67.46 | 87.15 | 23.40 | Sunny |



| | Date | Temperature | Humidity | WindSpeed | Event |
|---|---|---|---|---|---|
| 0 | 9/1/23 | 46.85 | 55.90 | 0.39 | Snow |
| 1 | 9/2/23 | 72.78 | 45.50 | 23.21 | Rain |
| 2 | 9/3/23 | 62.94 | 51.40 | 10.70 | Snow |
| 3 | 9/4/23 | 56.94 | 61.36 | 24.17 | Rain |
| 4 | 9/5/23 | 37.02 | 80.90 | 24.09 | Rain |
| 5 | 9/6/23 | 68.98 | 40.35 | 21.33 | Rain |
| 6 | 9/7/23 | 68.98 | 40.35 | 7.36 | Sunny |
| 7 | 9/8/23 | 68.98 | 65.54 | 21.28 | Snow |
| 8 | 9/9/23 | 57.05 | 60.87 | 21.28 | Snow |
| 9 | 9/10/23 | 61.86 | 51.11 | 7.92 | Snow |
| 10 | 9/11/23 | 30.93 | 45.99 | 4.24 | Snow |
| 11 | 9/12/23 | 73.65 | 56.88 | 13.92 | Sunny |
| 12 | 9/13/23 | 67.46 | 87.15 | 23.40 | Sunny |

# Example: Data Cleaning in Python

## 1. Reading the data (Add explanation to the function, such as parameters)

```python
import pandas as pd

df_weather = pd.read_csv('/Users/xiaotingzhou/Documents/GitHub/AI_Civil/Dataset/chicago_weather_sept_to_dec.csv')
df_weather.head(13)
```

✓ 0.0s Python

| | Date | Temperature | Humidity | WindSpeed | Event |
|---|---|---|---|---|---|
| 0 | 9/1/23 | 46.85 | 55.90 | 0.39 | Snow |
| 1 | 9/2/23 | 72.78 | 45.50 | 23.21 | Rain |
| 2 | 9/3/23 | 62.94 | 51.40 | 10.70 | Snow |
| 3 | 9/4/23 | 56.94 | 61.36 | 24.17 | Rain |
| 4 | 9/5/23 | 37.02 | 80.90 | 24.09 | NaN |
| 5 | 9/6/23 | NaN | NaN | 21.33 | Rain |
| 6 | 9/7/23 | NaN | 40.35 | 7.36 | Sunny |
| 7 | 9/8/23 | 68.98 | 65.54 | NaN | Snow |
| 8 | 9/9/23 | 57.05 | 60.87 | 21.28 | NaN |
| 9 | 9/10/23 | 61.86 | 51.11 | 7.92 | Snow |
| 10 | 9/11/23 | 30.93 | 45.99 | 4.24 | Snow |
| 11 | 9/12/23 | 73.65 | 56.88 | 13.92 | NaN |
| 12 | 9/13/23 | 67.46 | 87.15 | 23.40 | Sunny |

# Example: Data Cleaning in Python

## 2. Checking if there are missing values (Steps)

```
df_weather.isnull().sum()
```

✓ 0.0s Python

```
Date            0
Temperature     7
Humidity        7
WindSpeed       7
Event          44
dtype: int64
```

```
df_weather.isna().sum()
```

✓ 0.0s Python

```
Date            0
Temperature     7
Humidity        7
WindSpeed       7
Event          44
dtype: int64
```

# Example: Data Cleaning in Python

## 3. Print data description

```
df_weather.describe()
```

✓ 0.0s Python

| | Temperature | Humidity | WindSpeed |
|---|---|---|---|
| count | 115.000000 | 115.000000 | 115.000000 |
| mean | 51.781565 | 64.313130 | 12.722000 |
| std | 13.277521 | 14.515862 | 7.143882 |
| min | 30.250000 | 40.250000 | 0.360000 |
| 25% | 39.275000 | 52.035000 | 7.350000 |
| 50% | 52.220000 | 65.130000 | 13.520000 |
| 75% | 64.120000 | 75.380000 | 18.005000 |
| max | 74.410000 | 89.280000 | 24.750000 |

# Example: Data Cleaning in Python

## 4. Group data by date to calculate the average of each column

```
df_weather['Date'] = pd.to_datetime(df_weather['Date'], format='%m/%d/%y')
df_weather_grouped = df_weather.groupby(df_weather['Date'].dt.month).mean(numeric_only=True)
df_weather_grouped
```

✓ 0.0s Python

| Date | Temperature | Humidity | WindSpeed |
|---|---|---|---|
| 9 | 51.096071 | 61.448929 | 15.588571 |
| 10 | 52.134138 | 63.225862 | 11.588966 |
| 11 | 50.700357 | 65.377143 | 11.446786 |
| 12 | 53.089667 | 67.044333 | 12.332000 |

```
new_df = df_weather_grouped.reset_index()
new_df.rename(columns={'Date': 'Month'}, inplace=True)
new_df
```

✓ 0.0s Python

| | Month | Temperature | Humidity | WindSpeed |
|---|---|---|---|---|
| 0 | 9 | 51.096071 | 61.448929 | 15.588571 |
| 1 | 10 | 52.134138 | 63.225862 | 11.588966 |
| 2 | 11 | 50.700357 | 65.377143 | 11.446786 |
| 3 | 12 | 53.089667 | 67.044333 | 12.332000 |

# Example: Data Cleaning in Python

## 5. Filling missing values with mean value

```python
# Extract the month from df_weather
df_weather['Month'] = df_weather['Date'].dt.month

# Loop through new_df and fill missing values in df_weather based on matching months
for index, row in new_df.iterrows():
    # Find rows in df_weather where the month matches and fill NaN values
    mask = df_weather['Month'] == row['Month']

    df_weather.loc[mask, 'Temperature'] = df_weather.loc[mask, 'Temperature'].fillna(row['Temperature'])
    df_weather.loc[mask, 'Humidity'] = df_weather.loc[mask, 'Humidity'].fillna(row['Humidity'])
    df_weather.loc[mask, 'WindSpeed'] = df_weather.loc[mask, 'WindSpeed'].fillna(row['WindSpeed'])
    df_weather.loc[mask, 'Event'] = df_weather.loc[mask, 'Event'].bfill()

# Drop the temporary 'Month' column if no longer needed
df_weather = df_weather.drop(columns=['Month'])
```

✓ 0.0s Python

# Example: Data Cleaning in Python

## 5. Filling missing values with mean value

```
df_weather.head(13)
```

✓ 0.0s Python

| | Month | Temperature | Humidity | WindSpeed |
|---|---|---|---|---|
| 0 | 9 | 51.096071 | 61.448929 | 15.588571 |
| 1 | 10 | 52.134138 | 63.225862 | 11.588966 |
| 2 | 11 | 50.700357 | 65.377143 | 11.446786 |
| 3 | 12 | 53.089667 | 67.044333 | 12.332000 |

| | Date | Temperature | Humidity | WindSpeed | Event |
|---|---|---|---|---|---|
| 0 | 2023-09-01 | 46.850000 | 55.900000 | 0.390000 | Snow |
| 1 | 2023-09-02 | 72.780000 | 45.500000 | 23.210000 | Rain |
| 2 | 2023-09-03 | 62.940000 | 51.400000 | 10.700000 | Snow |
| 3 | 2023-09-04 | 56.940000 | 61.360000 | 24.170000 | Rain |
| 4 | 2023-09-05 | 37.020000 | 80.900000 | 24.090000 | Rain |
| 5 | 2023-09-06 | 51.096071 | 61.448929 | 21.330000 | Rain |
| 6 | 2023-09-07 | 51.096071 | 40.350000 | 7.360000 | Sunny |
| 7 | 2023-09-08 | 68.980000 | 65.540000 | 15.588571 | Snow |
| 8 | 2023-09-09 | 57.050000 | 60.870000 | 21.280000 | Snow |
| 9 | 2023-09-10 | 61.860000 | 51.110000 | 7.920000 | Snow |
| 10 | 2023-09-11 | 30.930000 | 45.990000 | 4.240000 | Snow |
| 11 | 2023-09-12 | 73.650000 | 56.880000 | 13.920000 | Sunny |
| 12 | 2023-09-13 | 67.460000 | 87.150000 | 23.400000 | Sunny |

# Example: Data Cleaning in Python

## 6. Filling NaN values in Backward Direction

```
new_df = df_weather.bfill()
new_df.head(13)
```

✓ 0.0s Python

| | Date | Temperature | Humidity | WindSpeed | Event |
|---|---|---|---|---|---|
| 0 | 9/1/23 | 46.85 | 55.90 | 0.39 | Snow |
| 1 | 9/2/23 | 72.78 | 45.50 | 23.21 | Rain |
| 2 | 9/3/23 | 62.94 | 51.40 | 10.70 | Snow |
| 3 | 9/4/23 | 56.94 | 61.36 | 24.17 | Rain |
| 4 | 9/5/23 | 37.02 | 80.90 | 24.09 | Rain |
| 5 | 9/6/23 | 68.98 | 40.35 | 21.33 | Rain |
| 6 | 9/7/23 | 68.98 | 40.35 | 7.36 | Sunny |
| 7 | 9/8/23 | 68.98 | 65.54 | 21.28 | Snow |
| 8 | 9/9/23 | 57.05 | 60.87 | 21.28 | Snow |
| 9 | 9/10/23 | 61.86 | 51.11 | 7.92 | Snow |
| 10 | 9/11/23 | 30.93 | 45.99 | 4.24 | Snow |
| 11 | 9/12/23 | 73.65 | 56.88 | 13.92 | Sunny |
| 12 | 9/13/23 | 67.46 | 87.15 | 23.40 | Sunny |

# Example: Data Cleaning in Python

## 7. Filling NaN values with forward fill value

```
new_df = df_weather.ffill()
new_df.head(13)
```

✓ 0.0s Python

| | Date | Temperature | Humidity | WindSpeed | Event |
|---|---|---|---|---|---|
| 0 | 9/1/23 | 46.85 | 55.90 | 0.39 | Snow |
| 1 | 9/2/23 | 72.78 | 45.50 | 23.21 | Rain |
| 2 | 9/3/23 | 62.94 | 51.40 | 10.70 | Snow |
| 3 | 9/4/23 | 56.94 | 61.36 | 24.17 | Rain |
| 4 | 9/5/23 | 37.02 | 80.90 | 24.09 | Rain |
| 5 | 9/6/23 | 37.02 | 80.90 | 21.33 | Rain |
| 6 | 9/7/23 | 37.02 | 40.35 | 7.36 | Sunny |
| 7 | 9/8/23 | 68.98 | 65.54 | 7.36 | Snow |
| 8 | 9/9/23 | 57.05 | 60.87 | 21.28 | Snow |
| 9 | 9/10/23 | 61.86 | 51.11 | 7.92 | Snow |
| 10 | 9/11/23 | 30.93 | 45.99 | 4.24 | Snow |
| 11 | 9/12/23 | 73.65 | 56.88 | 13.92 | Snow |
| 12 | 9/13/23 | 67.46 | 87.15 | 23.40 | Sunny |

# Example: Data Cleaning in Python

## 8. Setting filling NaN values limit to 1

```
df_weather['Temperature'] = df_weather['Temperature'].fillna(42, limit=1)
df_weather.head(13)
```

✓ 0.0s Python

| | Date | Temperature | Humidity | WindSpeed | Event |
|---|---|---|---|---|---|
| 0 | 9/1/23 | 46.85 | 55.90 | 0.39 | Snow |
| 1 | 9/2/23 | 72.78 | 45.50 | 23.21 | Rain |
| 2 | 9/3/23 | 62.94 | 51.40 | 10.70 | Snow |
| 3 | 9/4/23 | 56.94 | 61.36 | 24.17 | Rain |
| 4 | 9/5/23 | 37.02 | 80.90 | 24.09 | NaN |
| 5 | 9/6/23 | 42.00 | NaN | 21.33 | Rain |
| 6 | 9/7/23 | NaN | 40.35 | 7.36 | Sunny |
| 7 | 9/8/23 | 68.98 | 65.54 | NaN | Snow |
| 8 | 9/9/23 | 57.05 | 60.87 | 21.28 | NaN |
| 9 | 9/10/23 | 61.86 | 51.11 | 7.92 | Snow |
| 10 | 9/11/23 | 30.93 | 45.99 | 4.24 | Snow |
| 11 | 9/12/23 | 73.65 | 56.88 | 13.92 | NaN |
| 12 | 9/13/23 | 67.46 | 87.15 | 23.40 | Sunny |

# Example: Data Cleaning in Python

## 9. Interpolate of missing values

```
df_weather = df_weather.interpolate()
df_weather.head(13)
```

✓ 0.0s Python

| | Date | Temperature | Humidity | WindSpeed | Event |
|---|---|---|---|---|---|
| 0 | 9/1/23 | 46.850000 | 55.900 | 0.39 | Snow |
| 1 | 9/2/23 | 72.780000 | 45.500 | 23.21 | Rain |
| 2 | 9/3/23 | 62.940000 | 51.400 | 10.70 | Snow |
| 3 | 9/4/23 | 56.940000 | 61.360 | 24.17 | Rain |
| 4 | 9/5/23 | 37.020000 | 80.900 | 24.09 | NaN |
| 5 | 9/6/23 | 47.673333 | 60.625 | 21.33 | Rain |
| 6 | 9/7/23 | 58.326667 | 40.350 | 7.36 | Sunny |
| 7 | 9/8/23 | 68.980000 | 65.540 | 14.32 | Snow |
| 8 | 9/9/23 | 57.050000 | 60.870 | 21.28 | NaN |
| 9 | 9/10/23 | 61.860000 | 51.110 | 7.92 | Snow |
| 10 | 9/11/23 | 30.930000 | 45.990 | 4.24 | Snow |
| 11 | 9/12/23 | 73.650000 | 56.880 | 13.92 | NaN |
| 12 | 9/13/23 | 67.460000 | 87.150 | 23.40 | Sunny |

The interpolate() method in pandas is used to fill missing values (NaN) in a DataFrame or Series by estimating values based on neighboring data points.

It uses different interpolation methods (linear, polynomial, spline, etc.) to compute the missing values. By default, interpolate() uses linear interpolation.

# Example: Data Cleaning in Python

## 10. Dropna()

```
df_weather.isna().sum()
```

✓ 0.0s Python

```
Date            0
Temperature     7
Humidity        7
WindSpeed       7
Event          44
dtype: int64
```

```
df_weather = df_weather.dropna()
df_weather.isna().sum()
```

✓ 0.0s Python

```
Date           0
Temperature    0
Humidity       0
WindSpeed      0
Event          0
dtype: int64
```

# Example: Data Cleaning in Python

## 10. Deleting the rows having all NaN values

```
# Add a new row with NaN values
df_weather.loc[len(df_weather)] = [np.nan, np.nan, np.nan, np.nan, np.nan]
df_weather.isna().sum()
```

✓ 0.0s Python

```
Date           1
Temperature    8
Humidity       8
WindSpeed      8
Event         45
dtype: int64
```

Those rows in which all the values are NaN values will be deleted. If the row even has one value even then it will not be dropped.

```
df_weather = df_weather.dropna(how='all')
df_weather.isna().sum()
```

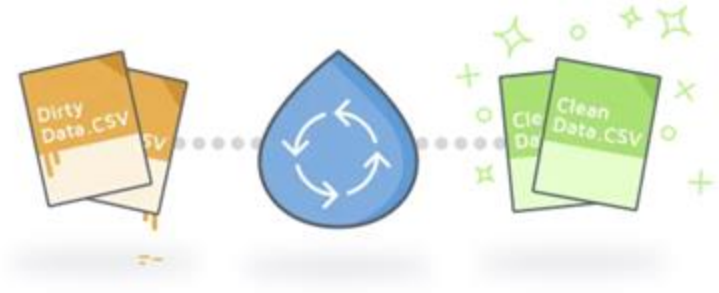
✓ 0.0s Python

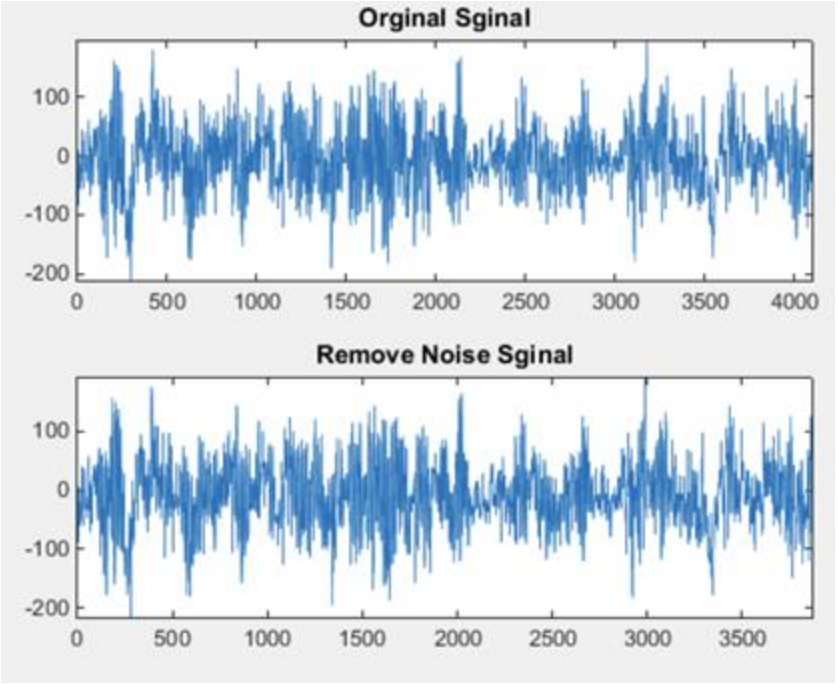
```
Date           0
Temperature    7
Humidity       7
WindSpeed      7
Event         44
dtype: int64
```

# Data Cleaning Techniques

**Handling Noisy Data**

Noisy generally means random error or containing unnecessary data points. Handling noisy data is one of the most important steps as it leads to the optimization of the model we are using Here are some of the methods to handle noisy data.



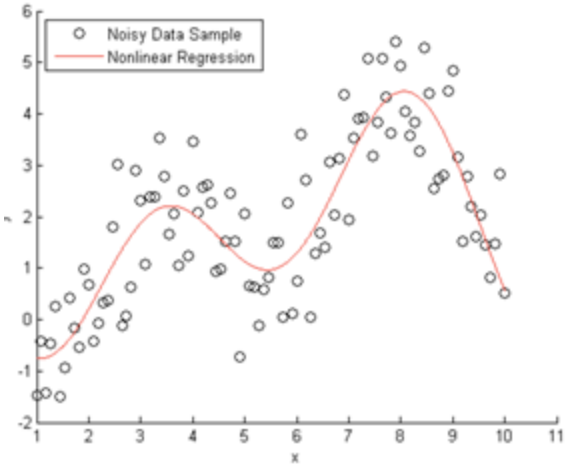



# Data Cleaning Techniques

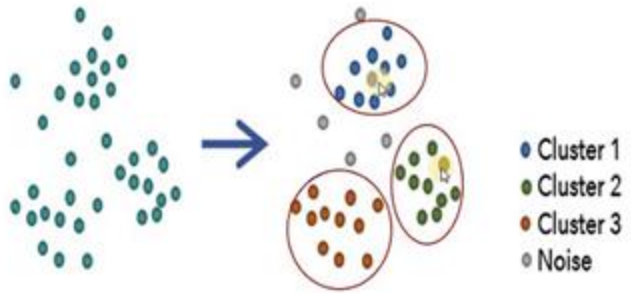
- **Binning**: This method is to smooth or handle noisy data. First, the data is sorted then, and then the sorted values are separated and stored in the form of bins. There are three methods for smoothing data in the bin. Smoothing by bin mean method: In this method, the values in the bin are replaced by the mean value of the bin; Smoothing by bin median: In this method, the values in the bin are replaced by the median value; Smoothing by bin boundary: In this method, the using minimum and maximum values of the bin values are taken, and the closest boundary value replaces the values.

# Data Cleaning Techniques

- **Regression**: This is used to smooth the data and will help to handle data when unnecessary data is present. For the analysis, purpose regression helps to decide the variable which is suitable for our analysis.
- **Clustering**: This is used for finding the outliers and also in grouping the data. Clustering is generally used in unsupervised learning.




*Regression*

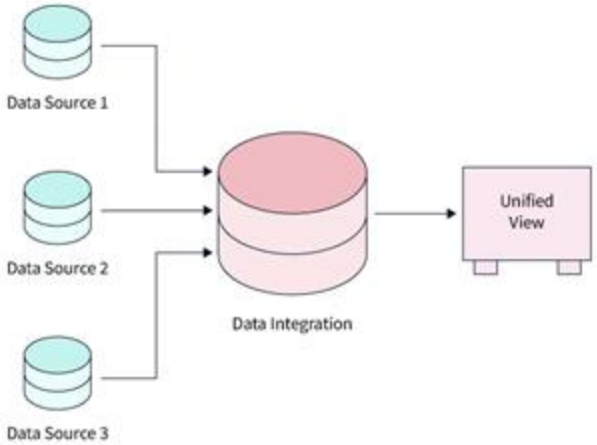



*Clustering*

# Data Integration

The process of combining multiple sources into a single dataset. The Data integration process is one of the main components of data management. There are some problems to be considered during data integration.

- **Schema integration**: Integrates metadata (a set of data that describes other data) from different sources.
- **Entity identification problem**: Identifying entities from multiple databases.
    - For example, the system or the user should know the student *id of one database and student*name of another database belonging to the same entity.
- **Detecting and resolving data value concepts**: The data taken from different databases while merging may differ. The attribute values from one database may differ from another database.
    - For example, the date format may differ, like “MM/DD/YYYY” or “DD/MM/YYYY”.



# Data Reduction

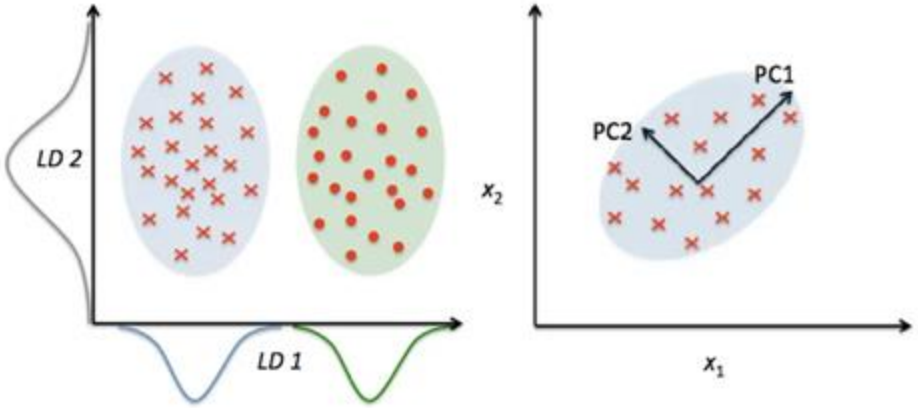
- **Dimensionality reduction:**

  **Linear Dimensionality Reduction Methods**

  1. **PCA** (Principal Component Analysis) : Popularly used for dimensionality reduction in continuous data, PCA rotates and projects data along the direction of increasing variance. The features with the maximum variance are the principal components.
  2. **Factor Analysis** : a technique that is used to reduce a large number of variables into fewer numbers of factors. The values of observed data are expressed as functions of a number of possible causes in order to find which are the most important. The observations are assumed to be caused by a linear transformation of lower dimensional latent factors and added Gaussian noise.
  3. **LDA** (Linear Discriminant Analysis): projects data in a way that the class separability is maximised. Examples from same class are put closely together by the projection. Examples from different classes are placed far apart by the projection

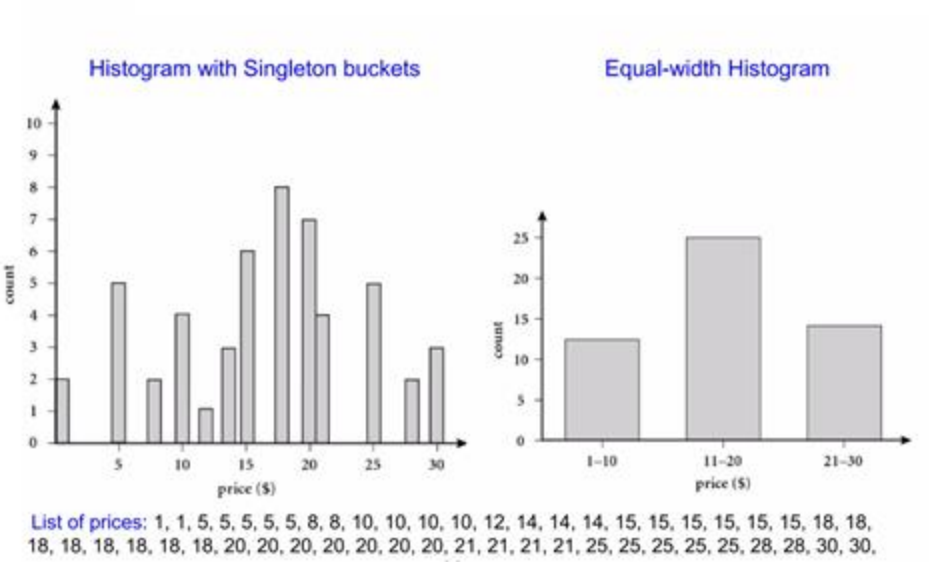
# Data Reduction

PCA orients data along the direction of the component with maximum variance whereas LDA projects the data to signify the class separability



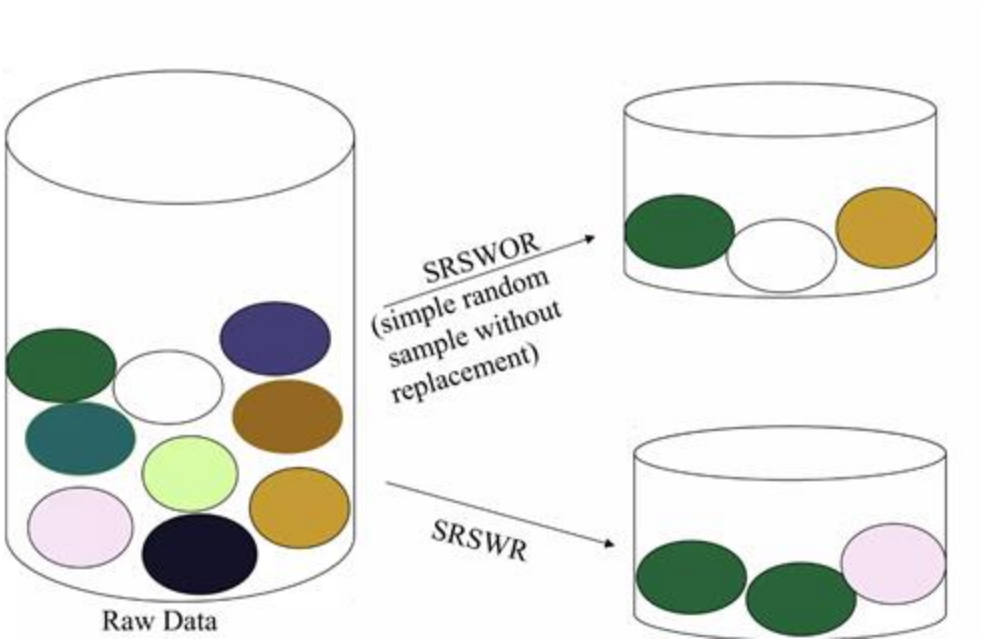
# Data Reduction

- **Numerosity Reduction:**
  - Reduce data volume by choosing alternative, smaller forms of data representation.
  - Parametric methods
    - Assume the data fits some model, estimate model parameters, store only the parameters, and discard the data (except possible outliers)
    - Example: Log-liner models, Regression
  - Non-parametric methods
    - Do not assume models
    - Major families, histograms, clustering, sampling



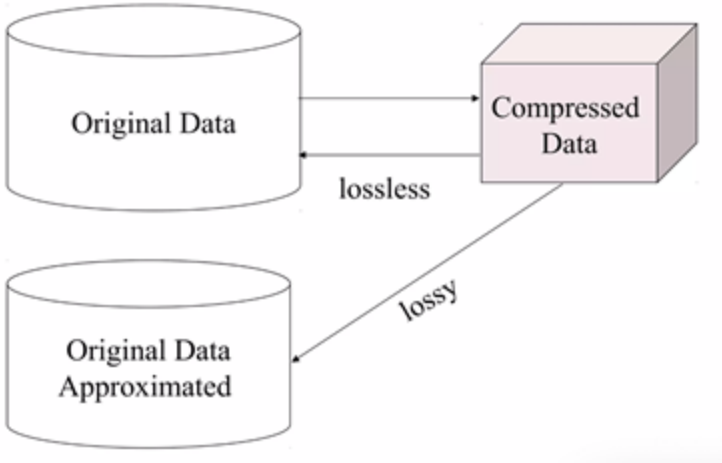
# Data Reduction

**Sampling: with or without replacement**



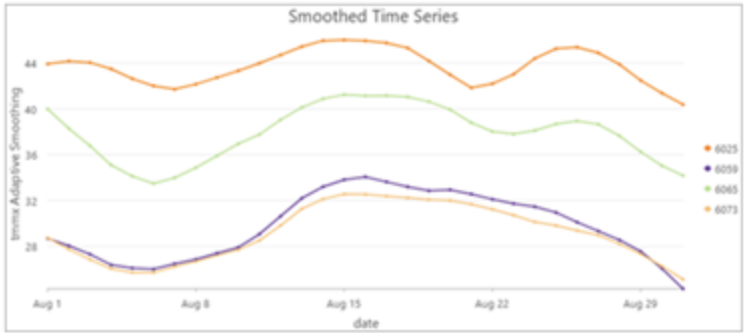
# Data Reduction

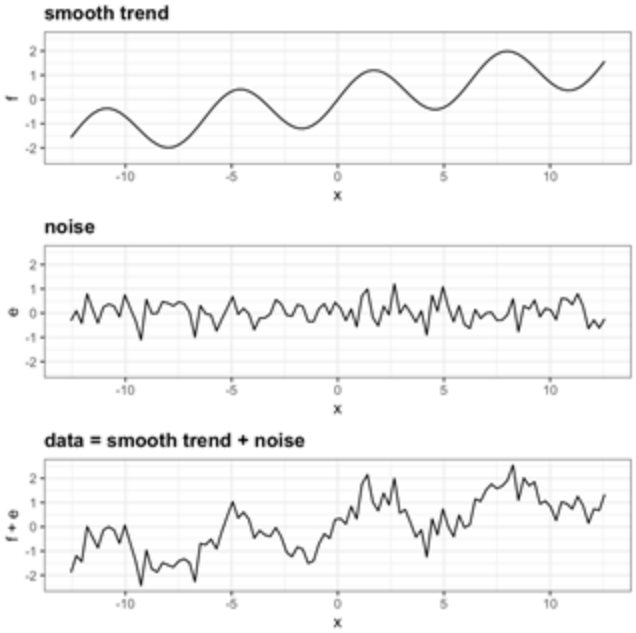
- **Data compression:**
  - The compressed form of data is called data compression. This compression can be lossless or lossy.
  - When there is no loss of information during compression, it is called lossless compression.
  - Whereas lossy compression reduces information, but it removes only the unnecessary information.
  - Application:
    - String compression
    - Audio/video compression



# Data Transformation

- **Smoothing:**
  - Remove noise from the dataset, which helps in knowing the important features of the dataset.
  - By smoothing, we can find even a simple change that helps in prediction.



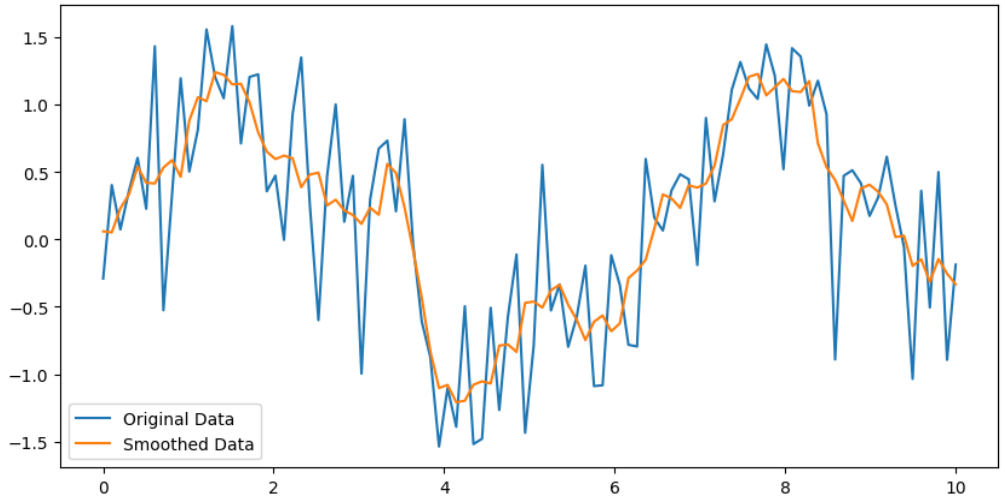



# Example: Data Transformation in Python

- **Smoothing:**
  - Smooth the data using uniform_filter1d (moving average)

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import uniform_filter1d

# Create some example data
x = np.linspace(0, 10, 100)
y = np.sin(x) + np.random.normal(0, 0.5, x.size)

# Smooth the data using uniform_filter1d (moving average)
smoothed_y = uniform_filter1d(y, size=5)

# Plot the original and smoothed data
plt.figure(figsize=(10, 5))
plt.plot(x, y, label='Original Data')
plt.plot(x, smoothed_y, label='Smoothed Data')
plt.legend()
plt.show()

```

✓ 0.0s

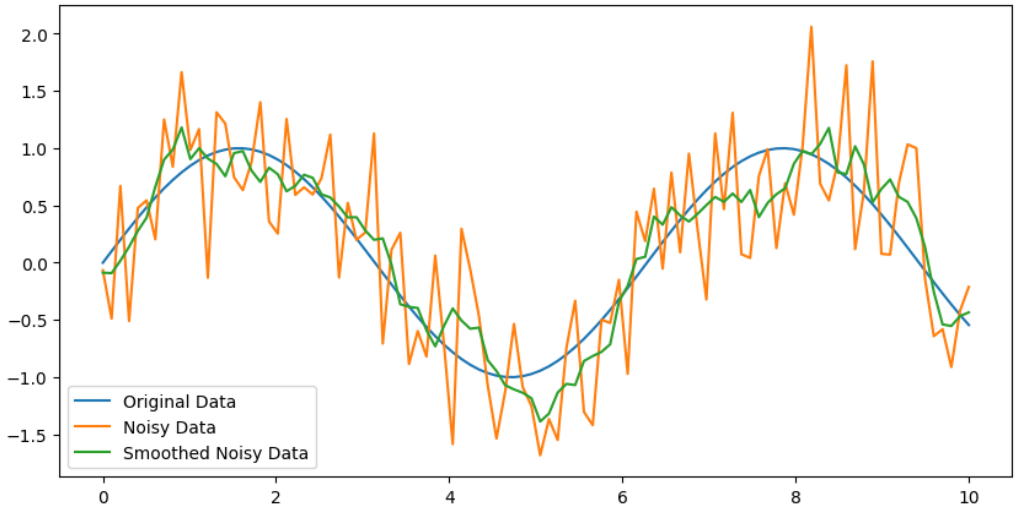


# Example: Data Transformation in Python

- **Smoothing:**
  - Smooth the data using uniform_filter1d (moving average)

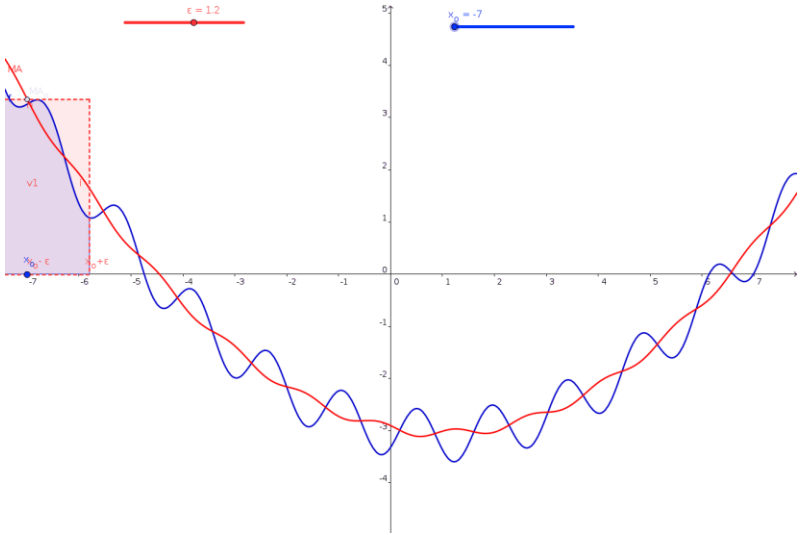
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import uniform_filter1d

# Create some example data
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Add noise to the data
noisy_y = y + np.random.normal(0, 0.5, x.size)

# Smooth the noisy data using uniform_filter1d (moving average)
smoothed_noisy_y = uniform_filter1d(noisy_y, size=5)

# Plot the original, noisy, and smoothed data
plt.figure(figsize=(10, 5))
plt.plot(x, y, label='Original Data')
plt.plot(x, noisy_y, label='Noisy Data')
plt.plot(x, smoothed_noisy_y, label='Smoothed Noisy Data')
plt.legend()
plt.show()

```

✓ 0.1s



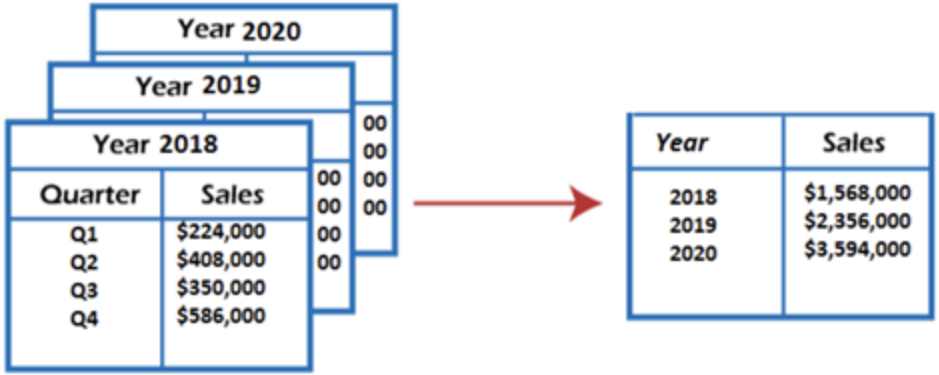
# Data Transformation

- **Moving Average:**
  - The moving average is a simple and widely-used technique for smoothing data. It calculates the average of a set of data points within a specified window and moves that window across the entire dataset. The size of the window determines the level of smoothing:
    - **Small window size**: Less smoothing; more detail retained.
    - **Large window size**: More smoothing; less detail retained.



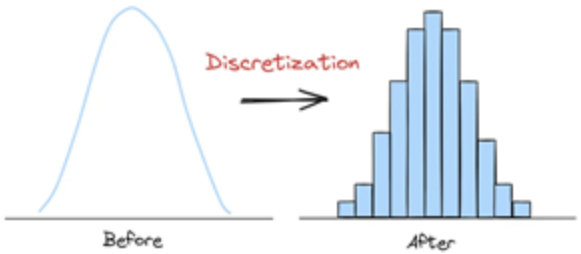
# Data Transformation

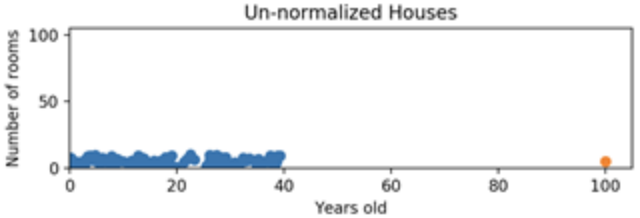
- **Aggregation:**
  - Data Aggregation is the process of compiling large volumes of data and transforming it into an organized and summarized format that is more consumable and comprehensive.
  - Data Aggregation can enable the capability to forecast future trends and aid in predictive analysis.
  - For example, a company may look at monthly sales data of a product instead of raw sales data to understand its performance better and forecast future sales.

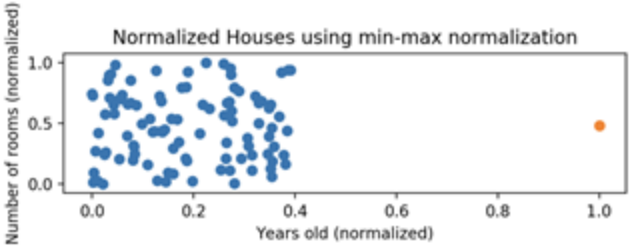


# Data Transformation

- **Discretization:**
  - The continuous data here is split into intervals. Discretization reduces the data size.
  - For example, rather than specifying the class time, we can set an interval like (3 pm-5 pm, or 6 pm-8 pm).
- **Normalization:**
  - It is the method of scaling the data so that it can be represented in a smaller range. For example, ranging from -1.0 to 1.0.



# Association Rules - Association Rule Discovery

**Supermarket shelf management –**
**Market-basket model:**

- **Goal:** Identify items that are bought together by sufficiently many customers
- **Approach:** Process the sales data collected with barcode scanners to find dependencies among items
- **A classic rule:**
  - If someone buys diaper and milk, then he/she is likely to buy beer
  - Don't be surprised if you find six-packs next to diapers!

# Association Rules - The Market-Basket Model

- A **large set** of **items**
  - e.g., things sold in a supermarket
- A **large set** of **baskets**
  - Each basket is a **small subset of items**
    - e.g., the things one customer buys on one day
- **Discover association rules:**

  People who bought {x,y,z} tend to buy {v,w}
  - Example application: Amazon

Input:

| *Basket* | *Items* |
|---|---|
| 1 | Bread, Coke, Milk |
| 2 | Beer, Bread |
| 3 | Beer, Coke, Diaper, Milk |
| 4 | Beer, Bread, Diaper, Milk |
| 5 | Coke, Diaper, Milk |

Output:

**Rules Discovered:**

{Milk} --> {Coke}

{Diaper, Milk} --> {Beer}

# Association Rules - Frequent Itemsets

- **Simplest question:** Find sets of items that appear together "frequently" in baskets
- ***Support*** for itemset $I$: Number of baskets containing all items in $I$
  - (Often expressed as a fraction of the total number of baskets)
- Given a ***support threshold*** $s$, then sets of items that appear in at least $s$ baskets are called ***frequent itemsets***

| TID | Items |
|---|---|
| 1 | Bread, Coke, Milk |
| 2 | Beer, Bread |
| 3 | Beer, Coke, Diaper, Milk |
| 4 | Beer, Bread, Diaper, Milk |
| 5 | Coke, Diaper, Milk |

Support of {Beer, Bread} = 2

# Association Rules - Frequent Itemsets Example

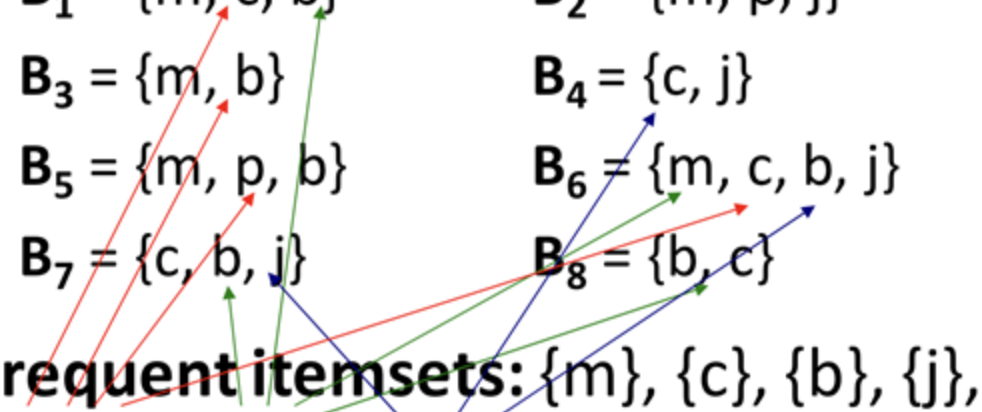
- **Items** = {milk, coke, pepsi, beer, juice}
- **Support threshold** = 3 baskets

$B_1$ = {m, c, b}  $B_2$ = {m, p, j}

$B_3$ = {m, b}  $B_4$ = {c, j}

$B_5$ = {m, p, b}  $B_6$ = {m, c, b, j}

$B_7$ = {c, b, j}  $B_8$ = {b, c}



- **Frequent itemsets:** {m}, {c}, {b}, {j}, {m,b} , {b,c} , {c,j}.

# Association Rules - Association Rules

- **Define: Association Rules:**
  If-then rules about the contents of baskets
- $\{i_1, i_2, \ldots, i_k\} \rightarrow j$ means: "if a basket contains all of $i_1, \ldots, i_k$ then it is ***likely*** to contain $j$"
- **In practice there are many rules, want to find significant/interesting ones!**
- ***Confidence*** of association rule is the probability of $j$ given $I = \{i_1, \ldots, i_k\}$

$$\text{conf}(I \rightarrow j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}$$

$$\text{conf}(I \rightarrow j) = P(j|I) = \frac{P(I,j)}{P(I)}$$

# Association Rules - Association Rules

- **Not all high-confidence rules are interesting**
  - The rule $X \rightarrow$ ***milk*** may have high confidence for many itemsets $X$, because milk is just purchased very often (independent of $X$)
- **Interest** of an association rule $I \rightarrow j$:
  abs. difference between its confidence and the fraction of baskets that contain $j$

$$Interest(I \rightarrow j) = |conf(I \rightarrow j) - P[j]| = |P(j|I) - P(j)|$$

  - Interesting rules: those with high interest values (usually above 0.5)
  - Why absolute value? Want to capture both *positive* and *negative* associations between itemsets and items

# Example: Confidence and Interest

$B_1 = \{m, c, b\}$ $B_2 = \{m, p, j\}$

$B_3 = \{m, b\}$ $B_4 = \{c, j\}$

$B_5 = \{m, p, b\}$ $B_6 = \{m, c, b, j\}$

$B_7 = \{c, b, j\}$ $B_8 = \{b, c\}$

- **Association rule: {m, b} → c**
  - Support = 2
  - Confidence = 2/4 = 0.5
  - Interest = |0.5 – 5/8| = 1/8
    - Item ***c*** appears in 5/8 of the baskets
    - The rule is not very interesting!

# Association Rule Mining

- **Problem: Find all association rules with support $\geq s$ and confidence $\geq c$**
  - **Note:** Support of an association rule is the support of the entire set of items in the rule (left side + right side)
- **Hard part: Finding the frequent itemsets!**
  - If $\{i_1, i_2, \ldots, i_k\} \rightarrow \{j\}$ has high support and confidence, then both $\{i_1, i_2, \ldots, i_k\}$ and $\{i_1, i_2, \ldots, i_k, j\}$ will be "frequent"

$$\text{conf}(I \rightarrow j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}$$

# Mining Association Rules

$$\text{conf}(I \to j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}$$

- **Step 1:** Find all frequent itemsets $I$
  - (we will explain this next)
- **Step 2:** **Rule generation**
  - For every subset $A$ of $I$, generate a rule $A \to I \setminus A$
    - Since $I$ is frequent, $A$ is also frequent
    - **Variant 1:** Single pass to compute the rule confidence
      - confidence(**A,B→C,D**) = support(**A,B,C,D**) / support(**A,B**)
    - **Variant 2:**
      - **Observation:** If **A,B,C→D** is below confidence, then so is **A,B→C,D**
      - Can generate "bigger" rules from smaller ones!
  - **Output the rules above the confidence threshold**

# Example: Mining Association Rules

$B_1$ = {m, c, b} $B_2$ = {m, p, j}

$B_3$ = {m, c, b, n} $B_4$= {c, j}

$B_5$ = {m, p, b} $B_6$ = {m, c, b, j}

$B_7$ = {c, b, j} $B_8$ = {b, c}

- Support threshold $s = 3$, confidence $c = 0.75$
- Step 1) Find frequent itemsets:
  - {b,m} {b,c} {c,m} {c,j} {m,c,b}
- Step 2) Generate rules:
  - ~~b→m: $c$=4/6~~ b→c: $c$=5/6 ~~b,c→m: $c$=3/5~~
  - m→b: $c$=4/5 … b,m→c: $c$=3/4
  - ~~b→c,m: $c$=3/6~~

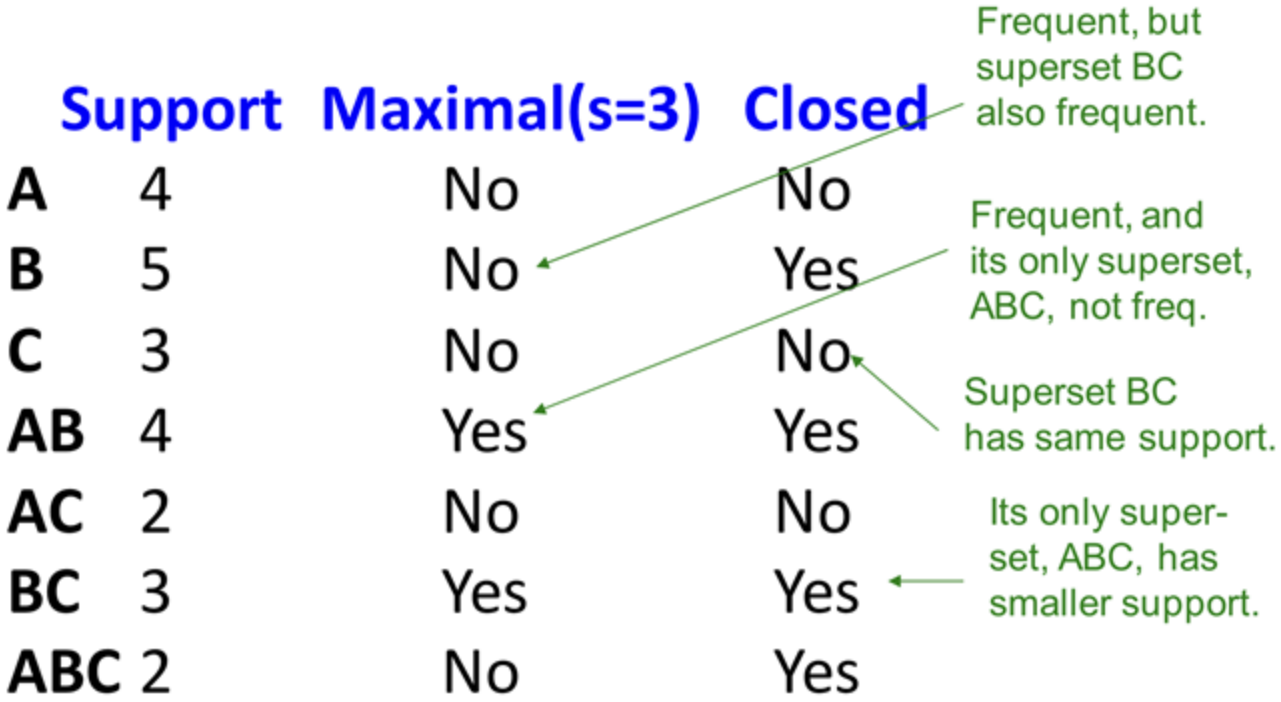
# Compacting the Output

- **To reduce the number of rules, we can post-process them and only output:**
  - **Maximal frequent itemsets:**
    No immediate superset is frequent
    - Gives more pruning

  **or**

  - **Closed itemsets:**
    No immediate superset has the same support (> 0)
    - Stores not only frequent information, but exact supports/counts

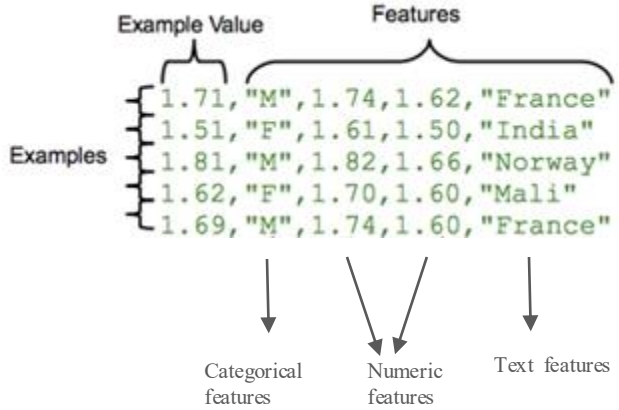
# Example: Maximal/Closed



# Predictors, Features

**Predictors**, also known as **features** or independent variables, are essential components in machine learning models.

- Machine learning algorithms process input data, which is structured in rows (instances) and columns (attributes known as features).
- These features represent the input data that the model uses to make predictions.
- Examples of predictors include:
  - Numeric features (e.g., age, temperature, salary)
  - Categorical features (e.g., gender, product category, country)
  - Text features (e.g., product descriptions, customer reviews
- Features are crucial attributes that impact or are useful for solving a problem. The importance of features outweighs everything else in determining the result, and no algorithm alone can replace the benefits of proper feature engineering.
- Properly selecting and preprocessing predictors significantly impact model performance.



# Predictors, Features - Example

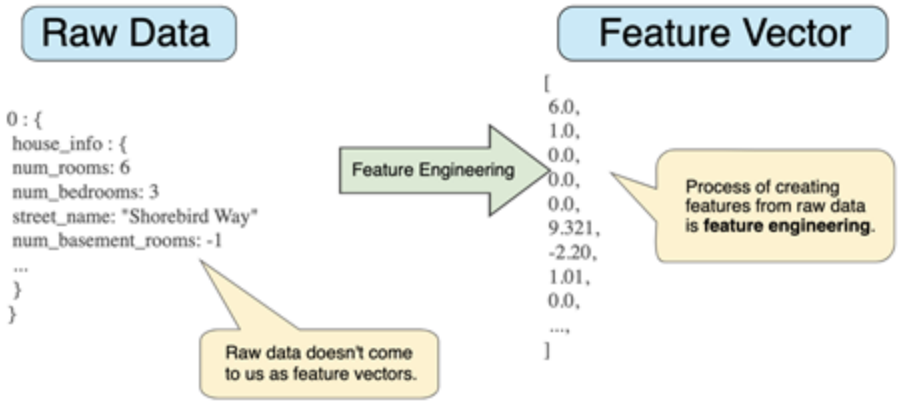
**Can you determine the Numeric, Categorical, and Text features?**

- Dataset: [World Happiness Report 2021]
- Examples of predictors include:
  - Numeric features
  - Categorical features
  - Text features

| | Country name | Regional indicator | Ladder score | Standard error of ladder score | upperwhisker | lowerwhisker | Logged GDP per capita | Social support | Healthy life expectancy | Freedom to make life choices |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Finland | Western Europe | 7.842 | 0.032 | 7.904 | 7.780 | 10.775 | 0.954 | 72.0 | 0.949 |
| 1 | Denmark | Western Europe | 7.620 | 0.035 | 7.687 | 7.552 | 10.933 | 0.954 | 72.7 | 0.946 |
| 2 | Switzerland | Western Europe | 7.571 | 0.036 | 7.643 | 7.500 | 11.117 | 0.942 | 74.4 | 0.919 |
| 3 | Iceland | Western Europe | 7.554 | 0.059 | 7.670 | 7.438 | 10.878 | 0.983 | 73.0 | 0.955 |
| 4 | Netherlands | Western Europe | 7.464 | 0.027 | 7.518 | 7.410 | 10.932 | 0.942 | 72.4 | 0.913 |

# Feature Engineering and Selection

- **What is feature engineering?**
  - Feature engineering is the pre-processing step of machine learning, which extracts features from raw data.
  - It helps to represent an underlying problem to predictive models in a better way, which as a result, improve the accuracy of the model for unseen data.
  - The predictive model contains predictor variables and an outcome variable, and while the feature engineering process selects the most useful predictor variables for the model.



# Feature Engineering and Selection

- **Why use feature engineering?**
  - **Better features mean flexibility**

    In machine learning, we always try to choose the optimal model to get good results. However, sometimes after choosing the wrong model, still, we can get better predictions, and this is because of better features. The flexibility in features will enable you to select the less complex models. Because less complex models are faster to run, easier to understand and maintain, which is always desirable.

  - **Better features mean simpler models**

    If we input the well-engineered features to our model, then even after selecting the wrong parameters (not much optimal), we can have good outcomes. After feature engineering, it is not necessary to do hard for picking the right model with the most optimized parameters. If we have good features, we can better represent the complete data and use it to best characterize the given problem.

  - **Better features mean better results**

    As already discussed, in machine learning, as data we will provide will get the same output. So, to obtain better results, we must need to use better features.

# Feature Engineering and Selection

- **5 Steps to Feature Engineering**
  1. Data Cleansing
  2. Data Transformation
  3. Feature Extraction
  4. Feature Selection
  5. Feature Iteration

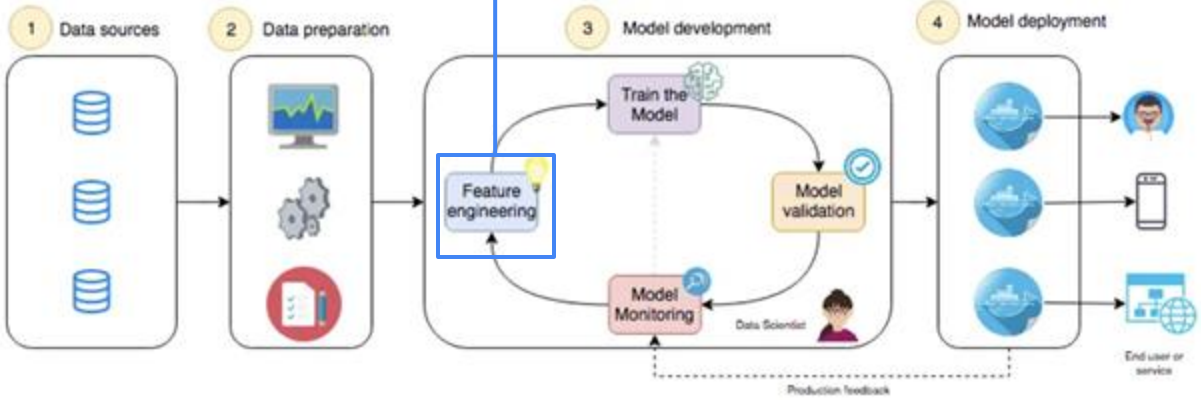
Missing Information

Entire Column Requires Encoding

Requires Formatting

| CUSTIMER ID | CUSTOMER NAME | LOCATION | CLICK ON AD? |
|---|---|---|---|
| 1 | Steve | USA | Yes |
| 2 | Mitch | Canada | 1 |
| 3 | Chanel | France | 0 |
| 4 | Bird | | 1 |
| 5 | Cynthia | Netherlands | 0 |
| 6 | Chanel | France | 0 |

Duplicate Entry



# Feature Engineering and Selection

- **5 Steps to Feature Engineering**

    1. **Data Cleansing**

        Data cleansing is the process of dealing with errors or inconsistencies in the data. This step involves identifying incorrect data, missing data, duplicated data, and irrelevant data. Moreover, Data cleansing is the process of deleting, replacing, or modifying data to remove outliers and incorrect values.

| Before Data Cleansing | | | | | After Data Cleansing | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Car Make and Model | Value USD | Passenger Capacity | Passenger Doors | Fuel Economy | Car Make and Model | Value USD | Passenger Capacity | Passenger Doors | Fuel Economy |
| Acura RDX | 43600 | 5 | 4 | N/A | Acura RDX | 43600 | 5 | 4 | **0** |
| Audi A5 | 51200 | 4 | 2 | 27 | Audi A5 | 51200 | 4 | 2 | 27 |
| Audi TTS | 51900 | | | | ~~Audi TTS~~ | ~~51900~~ | | | |
| BMW 2-Series | 32850 | 4 | 2 | N/A | BMW 2-Series | 32850 | 4 | 2 | **0** |
| Chevrolet Corvette | 55495 | 2 | | 19 | Chevrolet Corvette | 55495 | 2 | **2** | 19 |

# Feature Engineering and Selection

- **5 Steps to Feature Engineering**

    2. **Data Transformation**

        Data transformation is the process of transforming the data from one layout to another. Transformation needs to occur in a way that does not change the meaning of the original data. There are several techniques to transform the data depending on the desired outcome:

        - Transformation
        - Standardization
        - Data Encoding

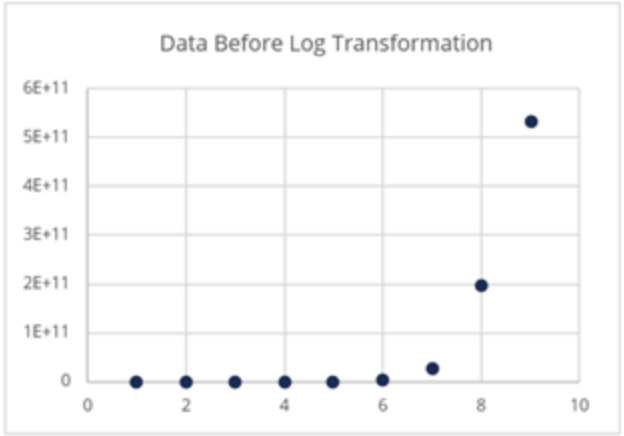
# Feature Engineering and Selection

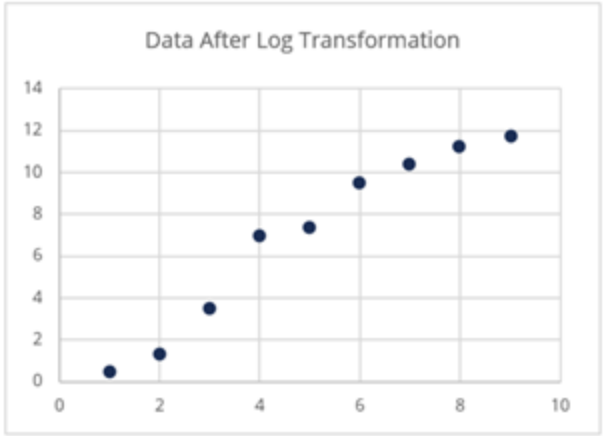
- **5 Steps to Feature Engineering**

  **2. Data Transformation**

  - Transformation

    Transformation refers to the application of a mathematical function to every data point. Transformation is a great way to handle highly skewed data.



# Feature Engineering and Selection

- **5 Steps to Feature Engineering**

  **2. Data Transformation**

  - Standardization

    Standardization refers to the process of converting the data into a uniform format. Data standardization is a great way of handling data with different units.


# Feature Engineering and Selection

- **5 Steps to Feature Engineering**

  2. **Data Transformation**

     - Data Encoding

       Encoding refers to the process of converting categorical variables to numerical variables. Data encoding is a great way of handling nominal and ordinal variables.

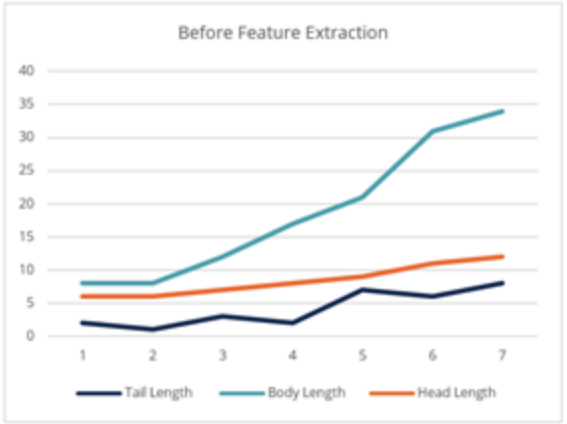
| Data Before Encoding | Data After Encoding |
|---|---|
| Age Group (< 18) | 1 |
| Age Group (18-25) | 2 |
| Age Group (26-50) | 3 |
| Age Group (> 50) | 4 |

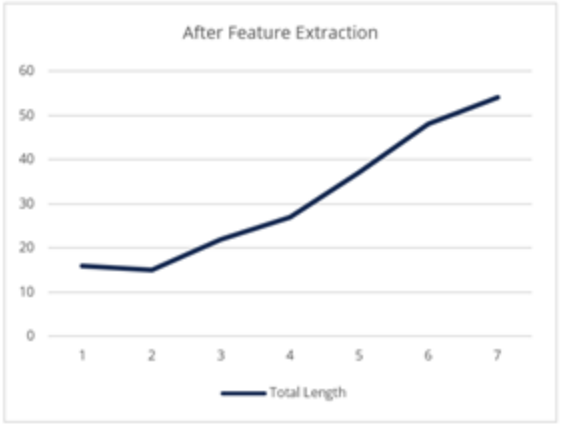
# Feature Engineering and Selection

- **5 Steps to Feature Engineering**

### 3. Feature Extraction

Feature extraction is the process of extracting new features from the existing attributes. This process is primarily concerned with reducing the number of features in the model. Feature extraction can be a lengthy process that requires the use of advanced analytics techniques (e.g., Principal Component Analysis).



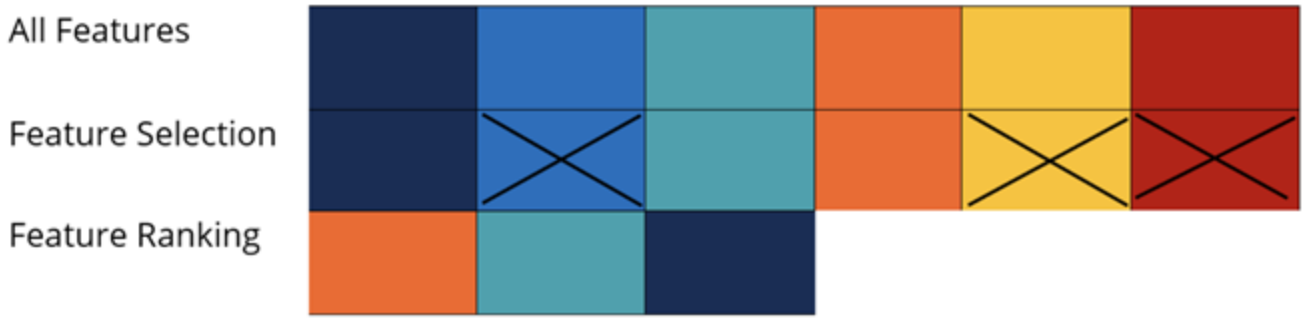



# Feature Engineering and Selection

- **5 Steps to Feature Engineering**

    4. **Feature Selection**

        Feature selection is the process of selecting the correct subset of features to ensure that the most relationship with the target variable is captured. It consists of eliminating features that do not explain the behavior of the target variable.

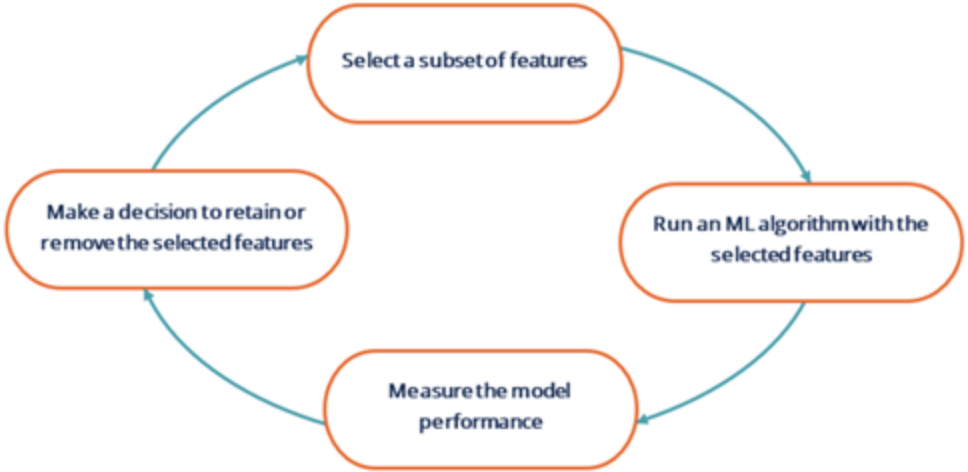


# Feature Engineering and Selection

- **5 Steps to Feature Engineering**

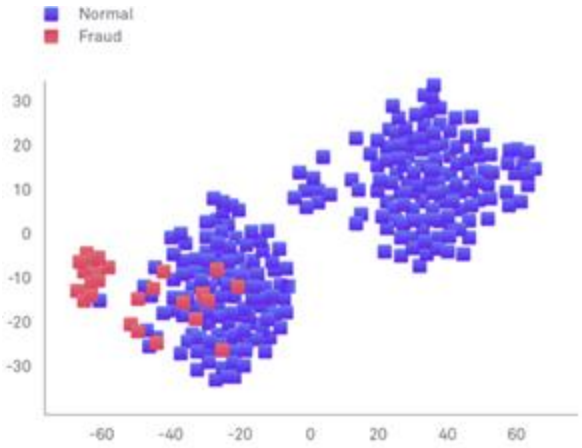
  5. **Feature Iteration**

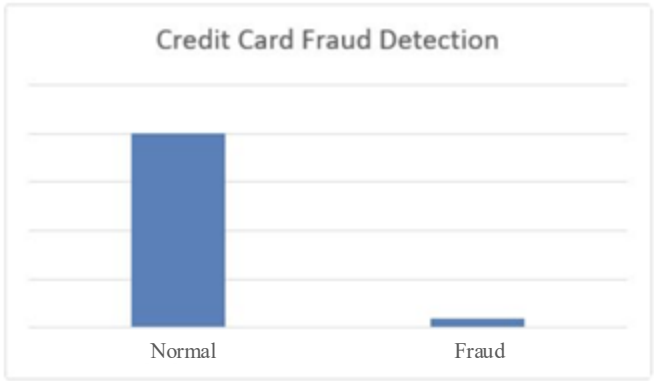
     Feature iteration, also known as the wrapper method of feature selection, is the final step in feature engineering. It is an iterative process involving the four steps below:



# Handling Imbalanced Data

Imbalanced data refers to datasets where the target class has an uneven distribution of observations, i.e., one class label has a very high number of observations, and the other has a deficient number of observations.



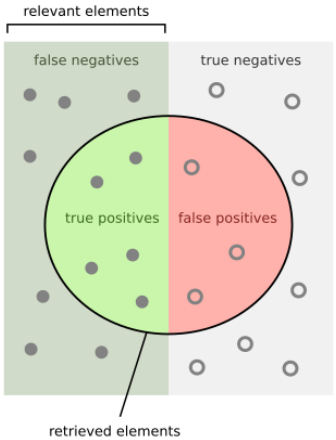
# Handling Imbalanced Data

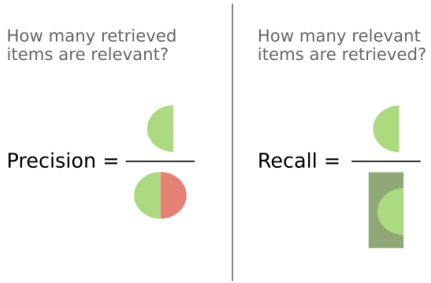
- **Why is Imbalanced Data a Problem?**
  - **Skewed Class Distribution**: Imbalanced dataset occurs when one class (the minority class) is significantly underrepresented compared to another class (the majority class) in a classification problem. This can skew the model's learning process because it may prioritize the majority class, leading to poor performance on the minority class.
  - **Biased Model Training**: Machine learning models aim to minimize errors, often measured by metrics like accuracy. In imbalanced datasets, a model can achieve high accuracy by simply predicting the majority class for all instances, ignoring the minority class completely. As a result, the model is biased towards the majority class and fails to capture patterns in the minority class accurately.
  - **Poor Generalization**: Imbalanced data can result in models that generalize poorly to new, unseen data, especially for the minority class. Since the model hasn't learned enough about the minority class due to its scarcity in the training data, it may struggle to make accurate predictions for instances belonging to that class in real-world scenarios.
  - **Costly Errors**: In many real-world applications, misclassifying instances from the minority class can be more costly or have higher consequences than misclassifying instances from the majority class. Imbalanced data exacerbates this issue because the model tends to make more errors on the minority class, potentially leading to significant negative impacts.
  - **Evaluation Metrics Misleading**: Traditional evaluation metrics like accuracy can be misleading in imbalanced datasets. For instance, a model achieving high accuracy may perform poorly on the minority class, which is often the class of interest. Using metrics like precision, recall, F1-score, or area under the ROC curve (AUC-ROC) can provide a more nuanced understanding of the model's performance across different classes.

# Handling Imbalanced Data

- **How to handle Imbalanced Data?** (Add source)
  - Choose Proper Evaluation Metric
    - For an imbalanced class dataset, the F1 score is a more appropriate metric.
    - Precision
    - Recall

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$



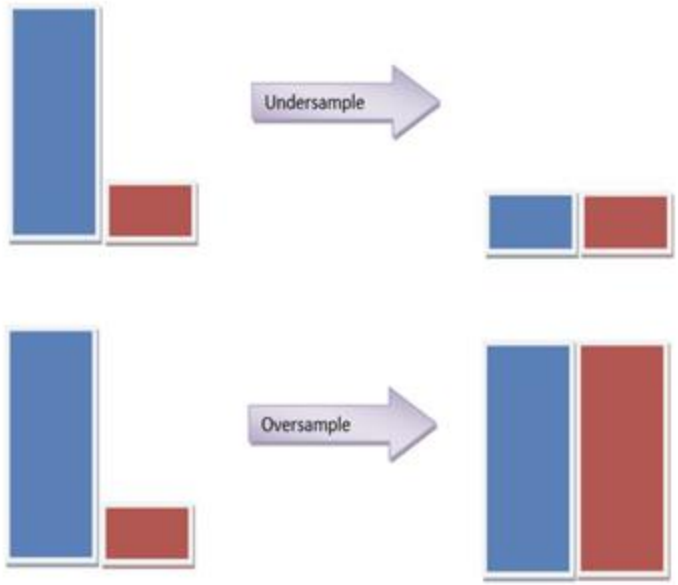

| | | Predicted 0 | Predicted 1 |
|---|---|---|---|
| Actual | 0 | TN | FP |
| | 1 | FN | TP |

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

# Handling Imbalanced Data

- **How to handle Imbalanced Data?**
  - Resampling (Blue, Red explanation, what they do
    - Oversampling
    - Undersampling



# Handling Imbalanced Data

- **How to handle Imbalanced Data?**
  - Resampling
    - Oversampling
    - Undersampling

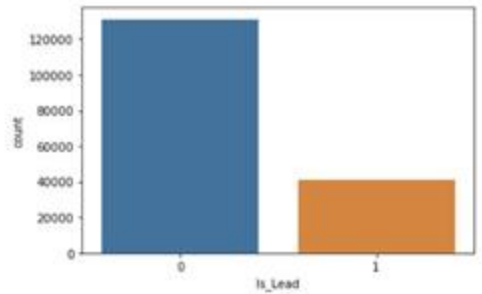
```
from sklearn.utils import resample
#create two different dataframe of majority and minority class
df_majority = df_train[(df_train['Is_Lead']==0)]
df_minority = df_train[(df_train['Is_Lead']==1)]
# upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with replacement
                                 n_samples= 131177, # to match majority class
                                 random_state=42)  # reproducible results
# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_minority_upsampled, df_majority])
```

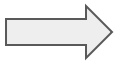
```
df_train['Is_Lead'].value_counts()
```

```
0    131177
1     40830
Name: Is_Lead, dtype: int64
```

```
import seaborn as sns
sns.countplot(df_train['Is_Lead'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x27c0bdbf430>
```



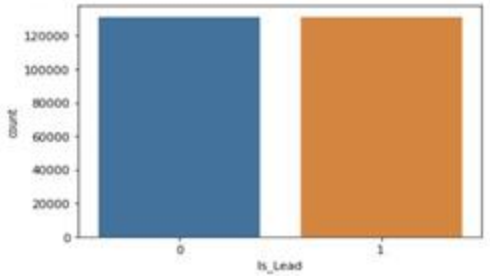



```
# Display new class counts
df_upsampled['Is_Lead'].value_counts()
```

```
1    131177
0    131177
Name: Is_Lead, dtype: int64
```

```
sns.countplot(df_upsampled['Is_Lead'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x27c0b83feb0>
```



# Handling Imbalanced Data

- **How to handle Imbalanced Data?**
  - SMOTE (Synthetic Minority Oversampling Technique)
    - Oversimple the minority class
    - In SMOTE new instances are synthesized from the existing data.

```
from imblearn.over_sampling import SMOTE
# Resampling the minority class. The strategy can be changed as required.
sm = SMOTE(sampling_strategy='minority', random_state=42)
# Fit the model to generate the data.
oversampled_X, oversampled_Y = sm.fit_sample(df_train.drop('Is_Lead', axis=1), df_train['Is_Lead
oversampled = pd.concat([pd.DataFrame(oversampled_Y), pd.DataFrame(oversampled_X)], axis=1)
```

```
10 oversampled['Is_Lead'].value_counts()
```

```
1    131177
0    131177
Name: Is_Lead, dtype: int64
```

# Handling Imbalanced Data

- **How to handle Imbalanced Data?**
  - BalancedBaggingClassifier
    - The same as a sklearn classifier but with additional balancing.
    - This classifier takes two special parameters, “sampling_strategy” and “replacement”. The sampling_strategy decides the type of resampling required (e.g., ‘majority’ – resample only the majority class, ‘all’ – resample all classes, etc.), and replacement decides whether it is going to be a sample with replacement or not.

```
from imblearn.ensemble import BalancedBaggingClassifier
from sklearn.tree import DecisionTreeClassifier
#Create an instance
classifier = BalancedBaggingClassifier(base_estimator=DecisionTreeClassifier(),
                                       sampling_strategy='not majority',
                                       replacement=False,
                                       random_state=42)
classifier.fit(X_train, y_train)
preds = classifier.predict(X_test)
```

# Handling Imbalanced Data

- **How to handle Imbalanced Data?**
  - Threshold Moving
    - Threshold for Classifiers:
      - ➢ Classifiers predict probabilities of class membership.
      - ➢ Default threshold: 0.5 (above belongs to one class, below to the other).
      - ➢ Imbalanced classes may require adjusting the threshold.
    - Optimal Threshold:
      - ➢ For imbalanced data, find an optimal threshold.
      - ➢ Use ROC Curves and Precision-Recall Curves.
      - ➢ Grid search or explore a range of values.

```
rom sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train,y_train)
rf_model.predict_proba(X_test) #probability of the class label
Output:

array([[0.97, 0.03],
       [0.94, 0.06],
       [0.78, 0.22],
       ...,
       [0.95, 0.05],
       [0.11, 0.89],
       [0.72, 0.28]])
After getting the probability we can check for the optimum value.

step_factor = 0.05
threshold_value = 0.2
roc_score=0
predicted_proba = rf_model.predict_proba(X_test) #probability of prediction
while threshold_value <=0.8: #continue to check best threshold upto probability 0.8
    temp_thresh = threshold_value
    predicted = (predicted_proba [:,1] >= temp_thresh).astype('int') #change the class boundary
    print('Threshold',temp_thresh,'--',roc_auc_score(y_test, predicted))
    if roc_score<roc_auc_score(y_test, predicted): #store the threshold for best classification
        roc_score = roc_auc_score(y_test, predicted)
        thrsh_score = threshold_value
    threshold_value = threshold_value + step_factor
print('---Optimum Threshold ---',thrsh_score,'--ROC--',roc_score)
```

Output:

```
Threshold 0.2 -- 0.7778223115708524
Threshold 0.25 -- 0.7851940092858101
Threshold 0.3 -- 0.7857354318529113
Threshold 0.35 -- 0.7770714585514514
Threshold 0.3999999999999997 -- 0.7700307442691748
Threshold 0.4499999999999996 -- 0.7617379569415947
Threshold 0.4999999999999994 -- 0.7520380808429419
Threshold 0.5499999999999993 -- 0.7412452995865506
Threshold 0.6 -- 0.7305482030821056
Threshold 0.65 -- 0.7183108354090917
Threshold 0.7000000000000001 -- 0.6977172374709829
Threshold 0.7500000000000001 -- 0.6742221224415976
---Optimum Threshold --- 0.3 --ROC-- 0.7857354318529113
```

# Data Preprocessing Steps in Machine Learning

## 1. Import Libraries and the Dataset

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import SimpleImputer
from sklearn.datasets import fetch_openml

# Step 1: Acquire the Dataset
boston = fetch_openml(name='boston', as_frame=True)
boston_df = pd.DataFrame(boston.data, columns=boston.feature_names)
boston_df['target'] = boston.target
```

✓ 0.0s Python

# Data Preprocessing Steps in Machine Learning

## 2. Check for Missing Values

```python
# Step 2: Check for Missing Values
print(boston_df.isnull().sum())
```

0.0s Python

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
target     0
dtype: int64
```

# Data Preprocessing Steps in Machine Learning

## 3. Encode Categorical Variables (How to handle different types of data?)

```python
from sklearn.preprocessing import LabelEncoder
# Step 3: Encode Categorical Variables
# Label Encoding
label_encoder = LabelEncoder()
boston_df['CRIM_LEVEL_ENCODED'] = label_encoder.fit_transform(boston_df['CRIM_LEVEL'])

# One-Hot Encoding
one_hot_encoded = pd.get_dummies(boston_df['CRIM_LEVEL'], prefix='CRIM_LEVEL')
df_encoded = pd.concat([boston_df, one_hot_encoded], axis=1)

# Display results
df_encoded
```

✓ 0.0s Python

| M | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | target | CRIM_LEVEL | CRIM_LEVEL_ENCODED | CRIM_LEVEL_High | CRIM_LEVEL_Low | CRIM_LEVEL_Medium |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 65.2 | 4.0900 | 1 | 296.0 | 15.3 | 396.90 | 4.98 | 24.0 | Low | 1 | False | True | False |
| 1 | 78.9 | 4.9671 | 2 | 242.0 | 17.8 | 396.90 | 9.14 | 21.6 | Low | 1 | False | True | False |
| 5 | 61.1 | 4.9671 | 2 | 242.0 | 17.8 | 392.83 | 4.03 | 34.7 | Low | 1 | False | True | False |
| 3 | 45.8 | 6.0622 | 3 | 222.0 | 18.7 | 394.63 | 2.94 | 33.4 | Low | 1 | False | True | False |
| 7 | 54.2 | 6.0622 | 3 | 222.0 | 18.7 | 396.90 | 5.33 | 36.2 | Low | 1 | False | True | False |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 3 | 69.1 | 2.4786 | 1 | 273.0 | 21.0 | 391.99 | 9.67 | 22.4 | Low | 1 | False | True | False |
| 0 | 76.7 | 2.2875 | 1 | 273.0 | 21.0 | 396.90 | 9.08 | 20.6 | Low | 1 | False | True | False |
| 5 | 91.0 | 2.1675 | 1 | 273.0 | 21.0 | 396.90 | 5.64 | 23.9 | Low | 1 | False | True | False |
| 4 | 89.3 | 2.3889 | 1 | 273.0 | 21.0 | 393.45 | 6.48 | 22.0 | Low | 1 | False | True | False |
| 0 | 80.8 | 2.5050 | 1 | 273.0 | 21.0 | 396.90 | 7.88 | 11.9 | Low | 1 | False | True | False |

# Data Preprocessing Steps in Machine Learning

## 4. Impute Missing Values

```python
# Step 4: Impute Missing Values
imputer = SimpleImputer(strategy='mean')
boston_df.iloc[:, :-1] = imputer.fit_transform(boston_df.iloc[:, :-1])
```

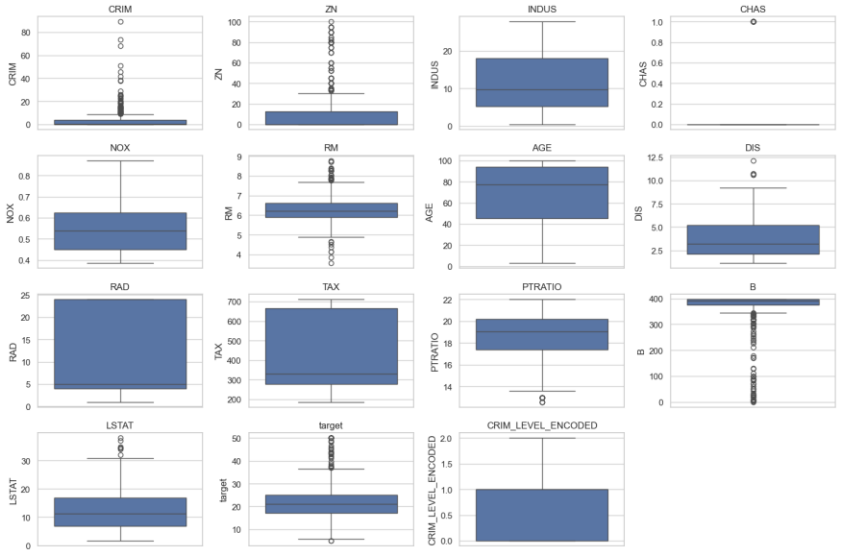
✓ 0.0s Python

# Data Preprocessing Steps in Machine Learning

## 5. Outlier Detection and Handling

```python
# Step 5: Outlier Detection and Handling
z_scores = np.abs((boston_df.iloc[:, :-1] - boston_df.iloc[:, :-1].mean()) / boston_df.iloc[:, :-1].std())
boston_df = boston_df[(z_scores < 3).all(axis=1)]
```

✓ 0.0s Python



# Data Preprocessing Steps in Machine Learning

## 8. Scale the Data (Normalization)

```python
# Step 7: Scale the Data
scaler = StandardScaler()
boston_df.iloc[:, :-1] = scaler.fit_transform(boston_df.iloc[:, :-1])
```

✓ 0.0s Python

# Data Preprocessing Steps in Machine Learning

## 9. Feature Selection (

```python
# Step 8: Feature Selection
X = boston_df.iloc[:, :-1]
y = boston_df.iloc[:, -1]
model = RandomForestRegressor()
model.fit(X, y)
importance = model.feature_importances_
feature_importance = pd.DataFrame(importance, index=X.columns, columns=["Importance"]).sort_values(by="Importance", ascending=False)
selected_features = feature_importance[feature_importance['Importance'] > 0.01].index.tolist()

X_selected = X[selected_features]
```

✓ 1.5s Python

```python
X_selected
```

✓ 0.0s Python

| | RM LSTAT | AGE target | NOX LSTAT | B LSTAT | DIS TAX | target | target | NOX target | target | target | DIS LSTAT | LSTAT | LSTAT | LS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -1.185703 | 0.197199 | -0.919871 | -1.070165 | -0.246662 | 0.204028 | 0.204028 | 0.254940 | 0.204028 | 0.204028 | -0.889855 | -1.128221 | -1.128221 | -1.128 |
| 1 | -0.448073 | 0.386245 | -0.589002 | -0.368421 | -0.261370 | -0.093457 | -0.093457 | -0.419227 | -0.093457 | -0.093457 | 0.178497 | -0.484935 | -0.484935 | -0.484 |
| 2 | -1.293398 | 0.950140 | -1.082313 | -1.237390 | -0.261370 | 1.530316 | 1.530316 | 1.069849 | 1.530316 | 1.530316 | -0.904827 | -1.275125 | -1.275125 | -1.275 |
| 3 | -1.531692 | 0.149640 | -1.194196 | -1.417125 | -0.015549 | 1.369179 | 1.369179 | 0.833033 | 1.369179 | 1.369179 | -0.998492 | -1.443679 | -1.443679 | -1.443 |
| 4 | -1.033596 | 0.735756 | -0.968881 | -1.011124 | -0.015549 | 1.716245 | 1.716245 | 1.143844 | 1.716245 | 1.716245 | -0.380102 | -1.074098 | -1.074098 | -1.074 |

# Data Preprocessing Steps in Machine Learning

## 10. Split the Dataset (Explain Why)

```python
# Step 9: Split the Dataset
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)
```

✓ 0.0s Python