Data analytics (ACSDS0603)

UNIT-4 NOTES

Exploratory Data Analysis (EDA) is a crucial approach in data science that involves summarizing and understanding the main characteristics of a dataset, often using statistical graphics and data visualization techniques to uncover patterns, relationships, and insights.

Key aspects of EDA:

Purpose and Benefits:

- **Understanding Data:**

EDA helps data scientists gain a comprehensive understanding of the data, including its structure, distribution, and potential relationships between variables.

- **Identifying Patterns:**

It allows for the discovery of patterns, trends, outliers, and anomalies that might not be immediately apparent.

- **Formulating Hypotheses:**

EDA can help formulate hypotheses and questions that can be further investigated through more rigorous statistical analysis.

- **Data Cleaning and Preparation:**

EDA can also identify data quality issues (e.g., missing values, inconsistencies) that need to be addressed before further analysis.

- **Choosing Appropriate Methods:**

EDA helps determine which statistical techniques or machine learning models are most suitable for a given dataset.

Steps in EDA:

1. **Formulate Questions:** Start with clear questions or hypotheses that you want to explore with the data.
2. **Data Preparation:** Clean and prepare the data by handling missing values, outliers, and inconsistencies.

3. **Univariate Analysis:** Analyze individual variables to understand their distributions and characteristics.
4. **Bivariate Analysis:** Explore relationships between pairs of variables.
5. **Multivariate Analysis:** Examine relationships between multiple variables.
6. **Data Visualization:** Use plots and charts to visualize data patterns and relationships.
7. **Refine Questions:** Use the insights gained from EDA to refine your questions or formulate new ones.

In Python, you can perform EDA techniques by importing necessary libraries, loading your dataset, and using functions to display basic information, summary statistics, check for missing values, and visualize distributions and relationships between variables. Here's a basic example:

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt


# Load the dataset

data = pd.read_csv('your_dataset.csv')


# Display basic information

print(data.info())


# Display summary statistics

print(data.describe())


# Check for missing values

print(data.isnull().sum())


# Visualize distributions

    # Histogram

    plt.hist(data['column_name'])
```

```
plt.show()

# Scatter plot

plt.scatter(data['x_column'], data['y_column'])

plt.show()

# Box plot

sns.boxplot(data['column_name'])

plt.show()

# Correlation matrix

sns.heatmap(data.corr(), annot=True)

plt.show()
```

Types of Exploratory Data Analysis (EDA)

1. Univariate Analysis
    - Definition: Focuses on analyzing a single variable at a time.
    - Purpose: To understand the variable's distribution, central tendency, and spread.
    - Techniques:
        - Descriptive statistics (mean, median, mode, variance, standard deviation).
        - Visualizations (histograms, box plots, bar charts, pie charts).
2. Bivariate Analysis
    - Definition: Examines the relationship between two variables.
    - Purpose: To understand how one variable affects or is associated with another.
    - Techniques:
        - Scatter plots.
        - Correlation coefficients (Pearson, Spearman).
        - Cross-tabulations and contingency tables.
        - Visualizations (line plots, scatter plots, pair plots).
3. Multivariate Analysis
    - Definition: Investigates interactions between three or more variables.

- Purpose: To understand the complex relationships and interactions in the data.
- Techniques:
    - Multivariate plots (pair plots, parallel coordinates plots).
    - Dimensionality reduction techniques (PCA, t-SNE).
    - Cluster analysis.
    - Heatmaps and correlation matrices.

4. Descriptive Statistics
    - Definition: Summarizes the main features of a data set.
    - Purpose: To provide a quick overview of the data.
    - Techniques:
        - Measures of central tendency (mean, median, mode).
        - Measures of dispersion (range, variance, standard deviation).
        - Frequency distributions.
5. Graphical Analysis
    - Definition: Uses visual tools to explore data.
    - Purpose: To identify patterns, trends, and data anomalies through visualization.
    - Techniques:
        - Charts (bar charts, histograms, pie charts).
        - Plots (scatter plots, line plots, box plots).
        - Advanced visualizations (heatmaps, violin plots, pair plots).
6. Dimensionality Reduction
    - Definition: Reduces the number of variables under consideration.
    - Purpose: To simplify models, reduce computation time, and mitigate the curse of dimensionality.
    - Techniques:
        - Principal Component Analysis (PCA).
        - t-Distributed Stochastic Neighbor Embedding (t-SNE).
        - Linear Discriminant Analysis (LDA).

Exploratory Data Analysis Tools

Using the following tools for exploratory data analysis, data scientists can effectively gain deeper insights and prepare data for advanced analytics and modeling.

1. Python Libraries

- Pandas: Provides data structures and functions needed to manipulate structured data seamlessly.
    - Use: Data cleaning, manipulation, and summary statistics.
- Supports large, multi-dimensional arrays and matrices and a collection of mathematical functions.
    - Use: Numerical computations and data manipulation.
- Matplotlib: A plotting library that produces static, animated, and interactive visualizations.
    - Use: Basic plots like line charts, scatter plots, and bar charts.
- Seaborn: Built on Matplotlib, it provides a high-level interface for drawing attractive statistical graphics.
    - Use: Advanced visualizations like heatmaps, violin plots, and pair plots.
- SciPy: Builds on NumPy and provides many higher-level scientific algorithms.
    - Use: Statistical analysis and additional mathematical functions.
- Plotly: A graphing library that makes interactive, publication-quality graphs online.
    - Use: Interactive and dynamic visualizations.

Principal Component Analysis (PCA) is a powerful technique used in data analysis, particularly for reducing the dimensionality of datasets while preserving crucial information. It does this by transforming the original variables into a set of new, uncorrelated variables called principal components.

PCA Example

Let's say we have a data set of dimension 300 (n) × 50 (p). n represents the number of observations, and p represents the number of predictors. Since we have a large p = 50, there can be p(p-1)/2 scatter plots, i.e., more than 1000 plots possible to analyze the variable relationship. Wouldn't it be a tedious job to perform exploratory analysis on this data?

PCA vs LDA vs Factor Analysis

Technique	Description
PCA (Principal Component Analysis)	Unsupervised dimension reduction technique. Reduces dimensions without considering class labels. Transforms correlated variables into linearly uncorrelated principal components that capture most of the data variance. Useful for data visualization and simplifying complex data.
LDA (Linear Discriminant Analysis)	Supervised dimension reduction technique. Takes class labels into account to find a feature combination that maximizes class separation. Useful for classification tasks and finding discriminant features.
Factor Analysis	Used to identify underlying, unmeasured variables (factors) that explain the variability across observed variables. Focuses on understanding latent structures in the data. Useful for revealing relationships and reducing dimensions based on these latent factors.

How Principal Component Analysis (PCA) Work ?

1. **Standardize the Data**

If the features of your dataset are on different scales, it's essential to standardize them (subtract the mean and divide by the standard deviation).

2. **Compute the Covariance Matrix**

Calculate the covariance matrix for the standardized dataset.

3. **Compute Eigenvectors and Eigenvalues**

Find the eigenvectors and eigenvalues of the covariance matrix. The eigenvectors represent the directions of maximum variance, and the corresponding eigenvalues indicate the magnitude of variance along those directions.

4. **Sort Eigenvectors by Eigenvalues**

Sort the eigenvectors based on their corresponding eigenvalues in descending order.

5. **Choose Principal Components**

Select the top k eigenvectors (principal components) where k is the desired dimensionality of the reduced dataset.

6. **Transform the Data**

Multiply the original standardized data by the selected principal components to obtain the new, lower-dimensional representation of the data.

```
import numpy as np
```

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Example Data
np.random.seed(42)
X = np.random.rand(100, 3)  # 100 samples with 3 features

# Step 1: Standardize the Data
scaler = StandardScaler()
X_std = scaler.fit_transform(X)

# Step 2-5: PCA
pca = PCA()
X_pca = pca.fit_transform(X_std)

# Plot Explained Variance Ratio
explained_var_ratio = pca.explained_variance_ratio_
cumulative_var_ratio = np.cumsum(explained_var_ratio)

plt.plot(range(1, len(cumulative_var_ratio) + 1), cumulative_var_ratio, marker='o')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('Explained Variance Ratio vs. Number of Principal Components')
plt.show()
```

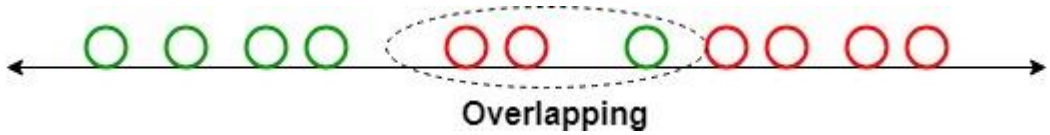
Linear Discriminant Analysis

When working with high-dimensional datasets it is important to apply dimensionality reduction techniques to make data exploration and modeling more efficient. One such technique is Linear Discriminant Analysis (LDA) which helps in reducing the dimensionality of data while retaining the most significant features for classification

tasks. It works by finding the linear combinations of features that best separate the classes in the dataset.

Maximizing Class Separability : Role of LDA

Linear Discriminant Analysis (LDA) also known as Normal Discriminant Analysis is supervised classification problem that helps separate two or more classes by converting higher-dimensional data space into a lower-dimensional space. It is used to identify a linear combination of features that best separates classes within a dataset.



For example, we have two classes that need to be separated efficiently. Each class may have multiple features and using a single feature to classify them may result in overlapping. To solve this LDA is used as it uses multiple features to improve classification accuracy.

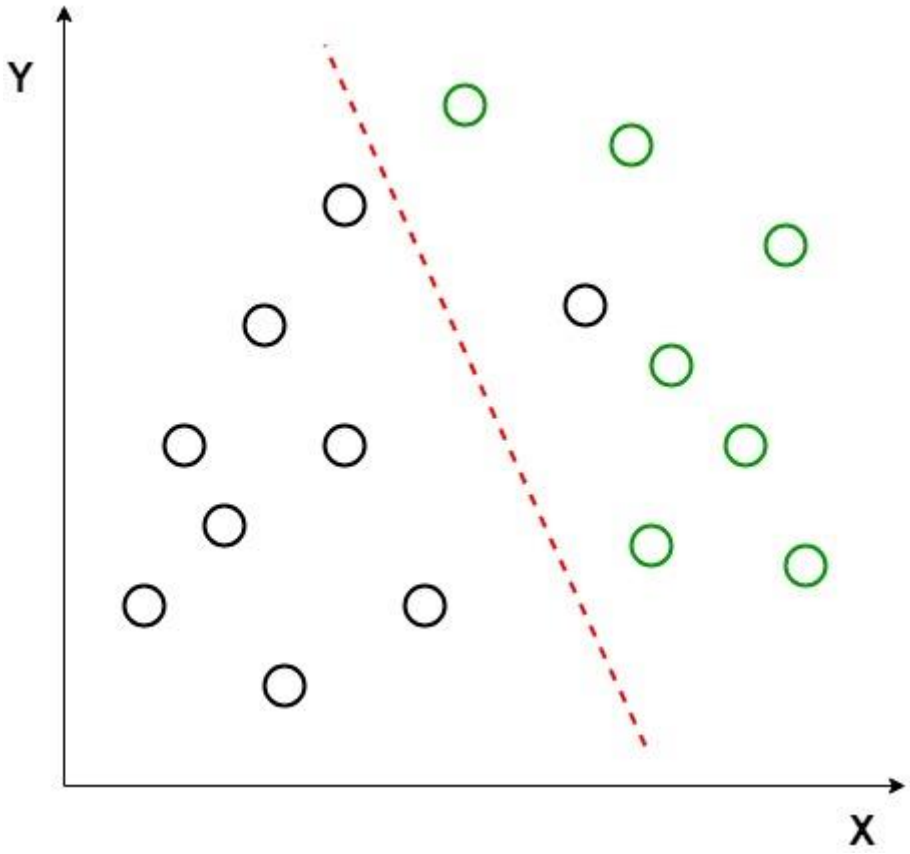
LDA works by some assumptions and we are required to understand them so that we have a better understanding of its working.

Core Assumptions of LDA

For LDA to perform effectively certain assumptions are made:

1. Gaussian Distribution: Data within each class should follow a Gaussian distribution.
2. Equal Covariance Matrices: Covariance matrices of the different classes should be equal.
3. Linear Separability: A linear decision boundary should be sufficient to separate the classes.

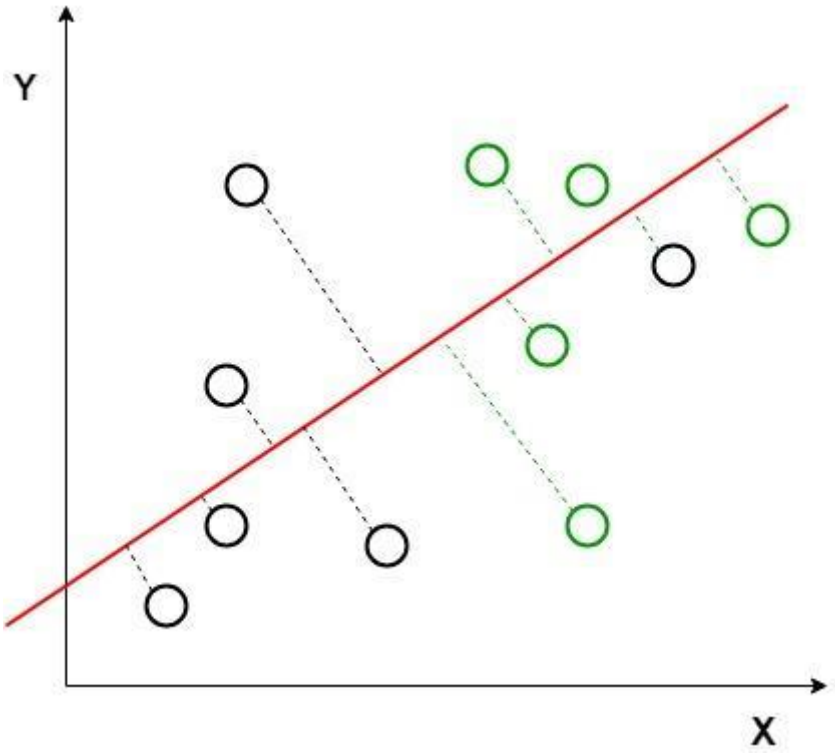
For example, when data points belonging to two classes are plotted if they are not linearly separable LDA will attempt to find a projection that maximizes class separability.



Linearly Separable Dataset

Image shows an example where the classes (black and green circles) are not linearly separable. LDA attempts to separate them using red dashed line. It uses both axes (X and Y) to generate a new axis in such a way that it maximizes the distance between the means of the two classes while minimizing the variation within each class. This transforms the dataset into a space where the classes are better separated.

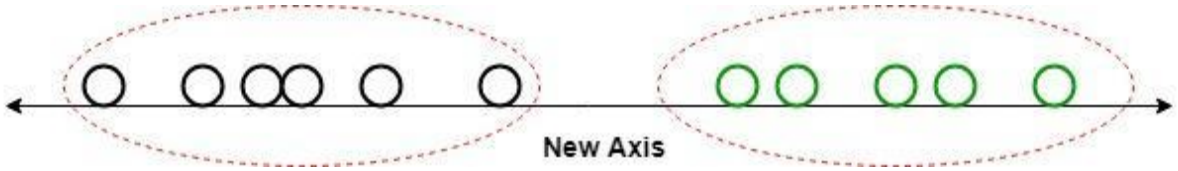
After transforming the data points along a new axis LDA maximizes the class separation. This new axis allows for clearer classification by projecting the data along a line that enhances the distance between the means of the two classes.



The perpendicular distance between the line and points

Perpendicular distance between the decision boundary and the data points helps us to visualize how LDA works by reducing class variation and increasing separability.

After generating this new axis using the above-mentioned criteria, all the data points of the classes are plotted on this new axis and are shown in the figure given below.



It shows how LDA creates a new axis to project the data and separate the two classes effectively along a linear path. But it fails when the mean of the distributions is shared as it becomes impossible for LDA to find a new axis that makes both classes linearly separable. In such cases we use non-linear discriminant analysis.

How does LDA work?

LDA works by finding directions in the feature space that best separate the classes. It does this by maximizing the difference between the class means while minimizing the spread within each class.

Python Code Implementation of LDA

In this implementation we will perform linear discriminant analysis using the Scikit-learn library on the Iris dataset.

```
# Importing required libraries:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.metrics import confusion_matrix


# Preparing dataset:


from sklearn.datasets import load_wine

dt = load_wine()

X = dt.data

y = dt.target


lda = LinearDiscriminantAnalysis()

lda_t = lda.fit_transform(X,y)

lda.explained_variance_ratio_

plt.xlabel('LD1')

plt.ylabel('LD2')

plt.scatter(lda_t[:,0],lda_t[:,1],c=y,cmap='rainbow',edgecolors='r')
```

- StandardScaler(): Standardizes the features to ensure they have a mean of 0 and a standard deviation of 1 removing the influence of different scales.
- fit_transform(): Standardizes the feature data by applying the transformation learned from the training data ensuring each feature contributes equally.
- LabelEncoder(): Converts categorical labels into numerical values that machine learning models can process.

- fit_transform() on y: Transforms the target labels into numerical values for use in classification models.
- LinearDiscriminantAnalysis(): Reduces the dimensionality of the data by projecting it into a lower-dimensional space while maximizing the separation between classes.
- transform() on X_test: Applies the learned LDA transformation to the test data to maintain consistency with the training data.

Advantages of LDA

- Simple and computationally efficient.
- Works well even when the number of features is much larger than the number of training samples.
- Can handle multicollinearity.

Disadvantages of LDA

- Assumes Gaussian distribution of data which may not always be the case.
- Assumes equal covariance matrices for different classes which may not hold in all datasets.
- Assumes linear separability which is not always true.
- May not always perform well in high-dimensional feature spaces.

Key Takeaways:

- LDA maximizes between-class scatter while minimizing within-class scatter
- It assumes Gaussian distribution and identical covariance matrices for classes
- LDA can be extended for multi-class problems and addresses some limitations of logistic regression
- Regularization techniques help overcome LDA's small sample size problem

What is Factor Analysis?

Factor analysis, a method within the realm of statistics and part of the general linear model (GLM), serves to condense numerous variables into a smaller set of factors. By doing so, it captures the maximum shared variance among the variables and condenses them into a unified score, which can subsequently be utilized for further analysis. Factor analysis operates under several assumptions: linearity in relationships, absence of multicollinearity among variables, inclusion of relevant variables in the analysis, and genuine correlations between variables and factors.

While multiple methods exist, principal component analysis stands out as the most prevalent approach in practice.

What does Factor mean in Factor Analysis?

In the context of factor analysis, a “factor” refers to an underlying, unobserved variable or latent construct that represents a common source of variation among a set of observed variables. These observed variables, also known as indicators or manifest variables, are the measurable variables that are directly observed or measured in a study.

How to do Factor Analysis (Factor Analysis Steps)?

Factor analysis is a statistical method used to describe variability among observed, correlated variables in terms of a potentially lower number of unobserved variables called factors. Here are the general steps involved in conducting a factor analysis:

1. Determine the Suitability of Data for Factor Analysis

- **Bartlett’s Test:** Check the significance level to determine if the correlation matrix is suitable for factor analysis.
- **Kaiser-Meyer-Olkin (KMO) Measure:** Verify the sampling adequacy. A value greater than 0.6 is generally considered acceptable.

2. Choose the Extraction Method

- **Principal Component Analysis (PCA):** Used when the main goal is data reduction.
- **Principal Axis Factoring (PAF):** Used when the main goal is to identify underlying factors.

3. Factor Extraction

- Use the chosen extraction method to identify the initial factors.
- Extract eigenvalues to determine the number of factors to retain. Factors with eigenvalues greater than 1 are typically retained in the analysis.
- Compute the initial factor loadings.

4. Determine the Number of Factors to Retain

- **Scree Plot:** Plot the eigenvalues in descending order to visualize the point where the plot levels off (the “elbow”) to determine the number of factors to retain.
- **Eigenvalues:** Retain factors with eigenvalues greater than 1.

5. Factor Rotation

- **Orthogonal Rotation (Varimax, Quartimax):** Assumes that the factors are uncorrelated.
- **Oblique Rotation (Promax, Oblimin):** Allows the factors to be correlated.
- Rotate the factors to achieve a simpler and more interpretable factor structure.
- Examine the rotated factor loadings.

6. Interpret and Label the Factors

- Analyze the rotated factor loadings to interpret the underlying meaning of each factor.
- Assign meaningful labels to each factor based on the variables with high loadings on that factor.

7. Compute Factor Scores (if needed)

- Calculate the factor scores for each individual to represent their value on each factor.

8. Report and Validate the Results

- Report the final factor structure, including factor loadings and communalities.
- Validate the results using additional data or by conducting a confirmatory factor analysis if necessary.

Factor Analysis Example (Factor Analyzer):

Here's an example of how you can perform factor analysis in Python using the factor_analyzer library:

```
# Install the factor_analyzer package

# !pip install factor_analyzer


import pandas as pd

from factor_analyzer import FactorAnalyzer

import matplotlib.pyplot as plt


# Load data

data = pd.read_csv('your_data.csv')


# Apply Bartlett's test
```

```
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity
chi_square_value, p_value = calculate_bartlett_sphericity(data)
print(f'Chi-square value: {chi_square_value}\nP-value: {p_value}')

# Apply KMO test
from factor_analyzer.factor_analyzer import calculate_kmo
kmo_all, kmo_model = calculate_kmo(data)
print(f'KMO Model: {kmo_model}')

# Create factor analysis object and perform factor analysis
fa = FactorAnalyzer(rotation="varimax")
fa.fit(data)

# Check Eigenvalues
eigen_values, vectors = fa.get_eigenvalues()
plt.scatter(range(1, data.shape[1]+1), eigen_values)
plt.plot(range(1, data.shape[1]+1), eigen_values)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
plt.show()

# Perform factor analysis with the determined number of factors
fa = FactorAnalyzer(n_factors=3, rotation="varimax")
fa.fit(data)

# Get factor loadings
loadings = fa.loadings_
```

```
print(loadings)

# Get variance of each factor
fa.get_factor_variance()

# Get factor scores
factor_scores = fa.transform(data)
print(factor_scores)
```

Term	Description
Factor	Latent variable representing a group of observed variables that are related and tend to co-occur.
Factor Loading	Correlation coefficient between the observed variable and the underlying factor.
Eigenvalue	A value indicating the amount of variance explained by each factor.
Communalities	The proportion of each observed variable’s variance that can be explained by the factors.
Extraction Method	The technique used to extract the initial factors from the observed variables (e.g., principal component analysis, maximum likelihood).
Rotation	A method used to rotate the factors to achieve simpler and more interpretable factor structure (e.g., Varimax, Promax).
Factor Matrix	A matrix showing the loadings of observed variables on extracted factors.

Term	Description
Scree Plot	A plot used to determine the number of factors to retain based on the magnitude of eigenvalues.
Kaiser-Meyer-Olkin (KMO) Measure	A measure of sampling adequacy, indicating the suitability of data for factor analysis. Values range from 0 to 1, with higher values indicating better suitability.
Bartlett’s Test	A statistical test used to determine whether the observed variables are intercorrelated enough for factor analysis.
Factor Rotation	The process of rotating the factors to achieve a simpler and more interpretable factor structure.
Factor Scores	Scores that represent the value of each factor for each individual observation.
Factor Variance	The amount of variance in the observed variables explained by each factor.
Loading Plot	A plot used to visualize the factor loadings of observed variables on the extracted factors.
Factor Rotation Criterion	A rule or criterion used to determine the appropriate rotation method and angle to achieve a simpler and more interpretable factor structure.

What is Data Munging?

Data munging is the process of transforming raw, unstructured data into a clean, usable format for analysis. Organizations use data munging to convert messy, inconsistent information from multiple sources into standardized datasets that power reporting, machine learning, and business intelligence.

Why is Data Munging Important?

- Ensures data quality by removing duplicates, fixing errors, and handling missing values
- Standardizes formats (e.g., converting different date formats into a single structure)
- Prepares data for analytics and automation, enabling accurate insights

Even the most advanced analytics tools can't deliver reliable results from poor-quality data, making data munging a critical first step in the data pipeline.

What is Data Wrangling?

Data wrangling is a comprehensive approach to data preparation that includes data munging along with additional processes like data discovery, cleaning, and enrichment.

While data munging focuses on format conversion, data wrangling covers the entire journey from raw data collection to analysis-ready datasets. Businesses often use specialized tools and programming languages like Python, R, and ETL platforms to streamline wrangling tasks.

Key Aspects of Data Wrangling

- **Data discovery:** Profiling and understanding raw data
- **Data structuring:** Converting unstructured data into structured formats
- **Data cleaning:** Removing duplicates, filling in missing values, and fixing inconsistencies
- **Data enrichment:** Integrating additional data sources to enhance insights
- **Mapping data:** Transforming and preparing raw data into a more usable format for analytics and machine learning

Data Munging vs. Data Wrangling: Key Differences

When diving into data preparation, it's crucial to understand the difference between data munging and data wrangling. While these terms are often used interchangeably, they serve distinct purposes in the data pipeline.

While both processes aim to prepare data for analysis, data wrangling encompasses a broader set of activities that includes data munging as one of its components. Think of data munging as the specific task of transforming data formats, while data wrangling is the complete toolkit for making data analysis-ready.

For example, when preparing customer data, munging might involve standardizing date formats, while wrangling would include discovering data quality issues, enriching the data with additional customer information, and validating the final dataset.