Data Processing

Data processing is an essential stage in data science and machine learning workflows, encompassing a series of operations that transform raw data into structured and meaningful information. This module explores critical aspects of data processing, including data collection, cleaning, mining, transformation, handling outliers, feature engineering, feature ranking, feature selection, dimensionality reduction, and data splitting. Proper data preprocessing is a crucial prerequisite for building accurate and efficient machine learning models.

The data processing pipeline begins with **data collection**, where data is acquired from various sources, including structured databases, files, online repositories, and APIs. The next step, **data cleaning**, involves identifying and rectifying corrupt or inaccurate records within a dataset. Unclean data can introduce biases and lead to misleading conclusions; therefore, techniques such as handling missing values, correcting mislabeling, and standardizing formats are employed to improve data quality.

Data mining is another fundamental step in this pipeline. It refers to the process of extracting patterns, correlations, and insights from large datasets using statistical methods, machine learning algorithms, and data visualization techniques. This stage includes classification, regression, clustering, and association rule mining, which enable decision-making and predictive analysis.

Data transformation ensures that data is in the correct format for analysis by modifying its structure, scaling numerical values, or encoding categorical variables. Additionally, **outlier detection and handling** is crucial for maintaining data integrity, as extreme deviations in datasets may distort model accuracy.

Feature engineering plays a key role in model performance by constructing meaningful features from raw data. This process includes generating new features, creating interaction terms, and selecting relevant attributes. To optimize model efficiency, **feature ranking** and **feature selection** help identify the most significant predictors, reducing dimensionality while preserving predictive power.

Dimensionality reduction techniques, such as Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE), further simplify datasets by extracting essential components and eliminating redundancy. Finally, **data splitting** ensures that models are trained, validated, and tested using separate subsets to prevent overfitting and assess generalizability.

Each of these topics represents a critical step in data processing, and this module provides a comprehensive exploration of the methodologies and best practices involved. Theoretical concepts are reinforced with hands-on exercises using real datasets to demonstrate practical applications in

data science. The following sections will delve deeper into each aspect of data processing, offering insights into their implementation and significance in the machine learning workflow.

This document is an extension of the research and lecture notes completed at Johns Hopkins University, Whiting School of Engineering, Engineering for Professionals, Artificial Intelligence Master's Program, Computer Science Master's Program, and Data Science Master's Program.

Contents

1 Data Processing and Preprocessing **1**
- 1.1 Data Processing . . . 1
- 1.2 Differences Between Data Processing and Data Preprocessing . . . 2
- 1.3 Summary . . . 3

2 Data Mining **4**
- 2.1 Data Mining Process . . . 4
- 2.2 Feature Engineering . . . 5
- 2.3 Summary of the Data Mining Process . . . 6

3 Data Collection **7**
- 3.1 Online Dataset Sources . . . 7

4 Introduction to Data Preprocessing **8**

5 Data Curation **10**
- 5.1 Description . . . 10
- 5.2 Key Activities in Data Curation . . . 10
- 5.3 Data Curation in Data Processing . . . 10
- 5.4 Data Curation in Data Preprocessing . . . 11
- 5.5 Summary . . . 13

6 Data Cleaning **14**
- 6.1 Common Data Cleaning Tasks . . . 14
- 6.2 Summary . . . 20

7 Data Transformation **21**
- 7.1 Discretization . . . 21
- 7.2 Example - Convert Nominal Features to Numerical Features . . . 22
- 7.3 Normalization and Standardization . . . 23
- 7.4 Summary . . . 29

8 Outliers **31**
- 8.1 **Mahalanobis Distance for Outlier Removal** . . . 32
- 8.2 Summary . . . 34

9 Feature Engineering **36**
- 9.1 Example - Mathematical Transformation . . . 36
- 9.2 Summary . . . 37

10 Feature Exploration **39**
- 10.1 Summary . . . 44

11 Feature Ranking and Selection **46**
- 11.1 Model for Feature Ranking and Selection . . . 47

11.2 **Bhattacharyya Distance** . . . 47
11.3 **Fishers Linear Discriminant Ratio (FDR/F-Score)** . . . 48
11.4 Summary . . . 50

12 Dimensionality Reduction **51**
12.1 Principal Component Analysis (PCA) for Dimensionality Reduction . . . 51
12.2 Number of Components to Retain (Dillon and Goldstein, 1984) . . . 58
12.3 Summary . . . 62

13 Data Splitting **63**
13.1 Common Methods of Data Splitting . . . 63
13.2 K-Fold Cross Validation . . . 64
13.3 Summary . . . 64

14 Summary **66**

1 Data Processing and Preprocessing

Understanding the differences between Data Processing and Data Preprocessing is crucial in the field of data science. This module provides a detailed comparison using methods such as Data Collection, Data Mining, Data Cleaning, Data Transformation, handling Outliers, Feature Engineering, Feature Exploration, Feature Ranking, Feature Selection, Dimensionality Reduction, and Data Splitting.

1.1 Data Processing

Data Processing refers to the entire workflow of manipulating raw data to make it suitable for analysis, visualization, or machine learning applications. It encompasses all steps from the initial collection of data to the final stage of analysis or presentation. The key steps in the data processing pipeline for data science and machine learning are as follows:

Key Steps in Data Processing

- **Data Mining**: Data mining is the process of discovering patterns, relationships, and insights from large datasets using statistical, machine learning, and artificial intelligence techniques. It enables data-driven decision-making by uncovering hidden structures in data.
- **Data Collection**: Gathering raw data from various sources such as databases, APIs, web scraping, IoT devices, surveys, or structured files (e.g., CSV, JSON, XML).
- **Data Preprocessing**: Data preprocessing is a crucial step in the data pipeline that ensures raw data is transformed into a clean and structured format suitable for analysis and modeling. It encompasses multiple processes aimed at improving data quality and enhancing machine learning model performance.
- **Machine Learning Model Training**: Applying supervised, unsupervised, or reinforcement learning techniques to train predictive models on processed data.
- **Model Evaluation and Validation**: Assessing model performance using metrics such as accuracy, precision, recall, F1-score, RMSE, or ROC-AUC to ensure robustness.
- **Hyperparameter Tuning**: Optimizing model parameters using techniques like grid search, random search, or Bayesian optimization to improve predictive performance.
- **Model Deployment**: Integrating the trained machine learning model into production environments, APIs, or applications for real-world usage.
- **Model Monitoring and Maintenance**: Continuously evaluating model performance post-deployment and retraining with new data to prevent performance degradation.
- **Data Visualization and Reporting**: Presenting insights through dashboards, plots, or interactive reports to communicate findings effectively.

Data Preprocessing

Data Preprocessing is a subset of Data Processing, specifically focused on preparing raw data for analysis or modeling. It includes initial steps taken before the actual analysis or application of

algorithms.

Key Steps in Data Preprocessing

- **Data Curation**: The process of collecting, organizing, validating, and maintaining data to ensure its quality, reliability, and usability for analysis. This includes data integration, metadata management, and ensuring consistency across datasets.
- **Data Cleaning**: Identifying and rectifying corrupt or inaccurate records within a dataset by handling missing values, correcting errors, and standardizing formats.
- **Data Transformation**: Converting data from its initial form into a format that is better suited for analysis, such as normalization, scaling, and encoding categorical variables.
- **Handling Outliers**: Identifying and addressing data points that significantly deviate from other observations within a dataset to prevent them from skewing model performance.
- **Feature Engineering**: Utilizing domain knowledge to generate meaningful features that enhance the performance of machine learning algorithms, including feature creation and interaction terms.
- **Feature Exploration**: Analyzing and visualizing feature distributions, correlations, and relationships to understand their impact on predictive modeling.
- **Feature Ranking**: Assessing the importance of different features in relation to the output variable using techniques such as correlation analysis, mutual information, and feature importance scores.
- **Feature Selection**: Selecting the most relevant features for model training to reduce dimensionality, improve model performance, and mitigate overfitting.
- **Dimensionality Reduction**: Reducing the number of random variables in a dataset while preserving its essential information using techniques like Principal Component Analysis (PCA) and t-SNE.
- **Data Splitting**: Partitioning data into subsets, typically including training, validation, and test sets, to effectively train, tune, and evaluate machine learning models.

1.2 Differences Between Data Processing and Data Preprocessing

Understanding the distinction between Data Processing and Data Preprocessing is crucial in data science workflows, as both play essential roles in handling and preparing data for analysis. Data Processing refers to the entire pipeline of working with data, encompassing various steps from collection to final analysis. It includes operations such as data cleaning, transformation, feature engineering, and even advanced analytical techniques to extract meaningful insights. On the other hand, Data Preprocessing is a subset of Data Processing that focuses specifically on preparing raw data for analysis by handling missing values, transforming variables, detecting outliers, and ensuring data quality before feeding it into models.

The following table provides a structured comparison between Data Processing and Data Preprocessing, highlighting their scope, objectives, and the specific tasks involved.

Data Processing	Data Preprocessing
Includes the entire workflow from data collection to final analysis	Focuses on initial steps to prepare data for analysis
Involves Data Collection, Data Mining, Data Cleaning, Data Transformation, Handling Outliers, Feature Engineering, Feature Exploration, Feature Ranking, Feature Selection, Dimensionality Reduction, and Data Splitting	Involves Data Collection, Data Cleaning, Data Transformation, Handling Outliers, Feature Engineering, Feature Ranking, Feature Selection, Dimensionality Reduction, and Data Splitting
Aims to convert raw data into meaningful insights	Aims to prepare data for accurate analysis
Includes steps beyond preprocessing, such as analysis	Prepares data for subsequent processing and analysis

Table 1: Differences Between Data Processing and Data Preprocessing

1.3 Summary

Data Processing and Data Preprocessing are fundamental steps in the data science pipeline. While data processing encompasses the entire journey of data from collection to final analysis, data preprocessing is specifically focused on preparing data for accurate and efficient analysis. Understanding the distinctions and interconnections between these processes is essential for successful data science projects.

2 Data Mining

Data mining is a fundamental aspect of data preprocessing that focuses on uncovering patterns, correlations, and valuable insights from large datasets. It leverages a combination of statistical methods, machine learning algorithms, and artificial intelligence techniques to transform raw data into actionable knowledge [15]. The Knowledge Discovery in Databases (KDD) process encompasses several stages, including data selection, preprocessing, transformation, mining, and interpretation [10]. Through these steps, data mining enables organizations to derive meaningful insights that improve decision-making, optimize operations, and enhance predictive modeling capabilities.

The application of data mining spans diverse industries, demonstrating its versatility and impact. In finance, it plays a crucial role in fraud detection and risk assessment, helping financial institutions identify unusual transaction patterns that may indicate fraudulent activities [24]. In healthcare, data mining supports disease prediction and patient diagnostics, assisting medical professionals in identifying trends in clinical data to enhance patient outcomes [2]. Retail industries use data mining for customer segmentation and recommendation systems, personalizing consumer experiences based on purchasing behaviors [21]. Additionally, cybersecurity applications rely on anomaly detection techniques in data mining to identify intrusions and cyber threats in real-time [4]. The continued advancement of data mining techniques ensures its growing importance in modern data-driven applications.

2.1 Data Mining Process

The data mining process follows a structured approach to ensure high-quality insights and actionable results. This process is iterative and involves multiple stages, each crucial for transforming raw data into meaningful patterns and knowledge [15].

Data Collection

The first step involves gathering data from multiple sources, such as databases, APIs, sensor logs, web scraping, spreadsheets, and real-time streaming data. The quality and diversity of the collected data significantly impact the subsequent steps in data mining. Data can be structured (e.g., relational databases), semi-structured (e.g., JSON, XML), or unstructured (e.g., text, images, videos) [15].

Data Preprocessing

Raw data often contains missing values, inconsistencies, duplicate records, and outliers that can negatively impact model performance. To ensure data quality and improve the accuracy of machine learning models, preprocessing techniques are applied to clean, transform, and integrate data effectively [10].

Handling missing data is a critical preprocessing step, as incomplete data can introduce bias and reduce model reliability. Various imputation techniques are used to address this issue, including mean or mode replacement for numerical and categorical variables, k-nearest neighbors (KNN) imputation for more sophisticated estimations, and deep learning-based imputers that leverage neural networks to fill in missing values while preserving complex relationships within the dataset [15].

Noise removal is essential for filtering out irrelevant or erroneous information that could distort model predictions. This process often involves smoothing techniques such as moving averages, which reduce fluctuations in time-series data, and median filtering, which helps eliminate outliers while preserving essential data structures [15].

Data transformation ensures that numerical values are properly scaled and standardized, making them more suitable for machine learning algorithms. Common techniques include Min-Max scaling, which normalizes values within a fixed range (e.g., 0 to 1), and Z-score normalization, which standardizes values based on their deviation from the mean. Additionally, categorical variables need to be encoded into numerical formats using methods like one-hot encoding, which creates binary representations for each category, or label encoding, which assigns integer values to categorical classes [15].

Data integration is crucial when working with multiple datasets from different sources, ensuring consistency and coherence before analysis. This step involves resolving schema mismatches, handling duplicate records, and unifying data formats to create a seamless dataset for further processing [15].

By applying these preprocessing techniques, data scientists can enhance the quality of raw data, improve model efficiency, and ensure robust and accurate predictions across various machine learning tasks [15].

2.2 Feature Engineering

Feature engineering involves transforming raw data into meaningful variables that enhance predictive model performance. This step is crucial in data preprocessing as it helps improve model accuracy, interpretability, and generalization [15].

Feature creation is the process of generating new features based on domain knowledge and existing data. For example, time-based features can be extracted from timestamps to analyze trends over different periods, such as day-of-week effects in sales forecasting or seasonality in temperature data. Creating meaningful features can significantly improve a model's ability to capture hidden patterns in the data [21].

Feature selection focuses on identifying the most relevant features while discarding redundant or irrelevant ones. Techniques such as correlation analysis, mutual information, and recursive feature elimination (RFE) help determine which features contribute the most to the predictive performance of a model. Reducing the number of input variables not only enhances computational efficiency but also mitigates overfitting by eliminating noise from the dataset [15].

Feature extraction aims to reduce the dimensionality of high-dimensional datasets while retaining important information. Methods such as Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) transform original features into a lower-dimensional space while preserving variance or local structure. These techniques are particularly useful in applications like image processing and text analysis, where raw data contains a vast number of variables [10].

By applying these feature engineering techniques, data scientists can enhance model performance, reduce computational complexity, and improve the interpretability of machine learning models, ultimately leading to more accurate and efficient predictions [15].

2.3 Summary of the Data Mining Process

The data mining process is an iterative cycle where insights from the model evaluation phase may lead back to refining preprocessing steps, selecting different features, or trying alternative mining techniques [10]. A well-structured data mining pipeline enhances predictive power, model robustness, and real-world applicability.

3 Data Collection

3.1 Online Dataset Sources

The **UCI KDD** online repository has various datasets that can be used for analysis, machine learning, and several application fields, such as GIS, cybersecurity, NLP, etc. The origin of some datasets goes back more than 20 years, sourced from competitions, challenges, grants, etc. Researchers and students use these datasets and share their experiences using a common platform. Source: UCI Knowledge Discovery in Databases Archive https://kdd.ics.uci.edu/

The **Kaggle** data repository has various datasets used for Kaggle competitions. The website also has tools to examine the features on-site. This source is one of the largest. Source: https://www.kaggle.com/datasets

The **KDnuggets** is another web page that encompasses almost everything (posts, news, datasets, tutorials, forums, webinars, software, etc.) relevant to machine learning and data analysis. Source: KDnuggets Datasets for Data Mining and https://www.kdnuggets.com/datasets/index.html

The U.S. government's open data website **Data.gov** is another resource that provides access to datasets published by agencies across the federal government. Source: https://data.gov/

The rest of the notes will demonstrate three different datasets from these repositories.

1. UCI KDD archive → 1990 US Census data
2. Kaggle → Graduate Admissions data
3. Kaggle → The Human Freedom Index data

Download the data files from UCI KDD website and Kaggle website (by registering to Kaggle -using a disposable email address if necessary).

Important Note: About physical dataset file shared among teams. Comparing machine learning models and measuring performances for model selection depends heavily on the input dataset. Thus, if a comparison between models and comparison among different experiments or teams' results are at hand, then the dataset shared among teams or different sets of experiments **must** be the same dataset. Moreover, to ensure validity, the same file should be shared among multiple teams or between different model pipelines.

4 Introduction to Data Preprocessing

Real-world datasets are often incomplete, noisy, and inconsistent, presenting significant challenges for effective data analysis and machine learning. Even a small percentage of missing or incorrect data can lead to substantial performance degradation in predictive models. Proper data preprocessing is essential to improve data quality, enhance model accuracy, and ensure meaningful insights. In fact, it is widely acknowledged that data preprocessing constitutes the majority of the effort in machine learning workflows [27]. Without an appropriate preprocessing strategy, models may fail to generalize effectively, leading to biased results or poor predictions.

Figure 1 illustrates the structured flow of data preprocessing, encompassing several essential steps that transform raw data into a suitable format for model training. The process begins with Data Curation, which involves organizing and validating data sources to ensure reliability. This is followed by Data Cleaning, where inconsistencies, missing values, and errors are addressed. Data Transformation further standardizes the dataset by normalizing, encoding, or scaling features. Additionally, Handling Outliers is crucial to prevent extreme values from disproportionately influencing model performance. Advanced preprocessing steps include Feature Engineering, where new features are created to improve predictive power, and Feature Exploration, which involves analyzing relationships and distributions within the dataset. Feature Ranking and Feature Selection help identify the most informative attributes, reducing dimensionality and improving computational efficiency. Dimensionality Reduction techniques, such as Principal Component Analysis (PCA), further streamline the dataset while preserving essential information. Finally, Data Splitting partitions the dataset into training, validation, and testing subsets to ensure robust model evaluation before entering the Model Training phase.

Each component in Figure 1 represents a crucial stage in data preprocessing, with multiple subprocesses that refine data before modeling. To successfully preprocess data, it is important to first define the key problems and objectives. Identifying the nature of the dataset and its environment allows for appropriate preprocessing decisions. Depending on the data source, preprocessing may involve data cleaning, feature engineering, and dimensionality reduction to enhance model performance.

The following sections provide an in-depth exploration of each preprocessing technique. We will examine their importance, methodologies, implementation strategies, and best practices to ensure that data is transformed effectively for machine learning applications.

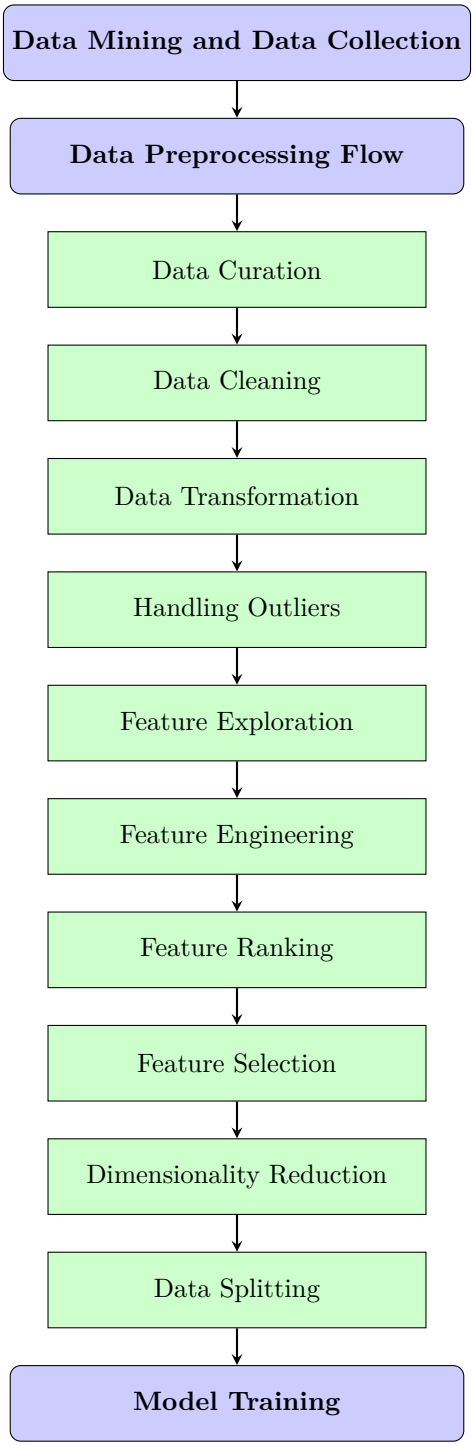

Figure 1: A structured workflow of data preprocessing steps in machine learning.

5 Data Curation

In the field of data science, Data Curation plays a pivotal role in ensuring the integrity, quality, and usability of data. This document provides a detailed description of Data Curation and explains how it integrates with Data Processing and Data Preprocessing.

5.1 Description

Data Curation refers to the comprehensive process of managing, preserving, and enhancing the value of data throughout its lifecycle. The goal of data curation is to ensure that data is accurate, accessible, and usable for long-term analysis and decision-making. This process includes various activities such as data collection, annotation, organization, storage, and maintenance. Data curation involves continuous monitoring and updating to keep the dataset relevant and useful.

5.2 Key Activities in Data Curation

- **Data Collection**: Gathering data from diverse sources such as databases, files, and APIs.
- **Data Annotation**: Adding metadata and documentation to describe the data context, sources, and quality.
- **Data Organization**: Structuring and categorizing data to ensure it is easily accessible and manageable.
- **Data Preservation**: Ensuring the long-term storage and integrity of data.
- **Data Maintenance**: Regularly updating and validating data to maintain its relevance and accuracy.

5.3 Data Curation in Data Processing

Data Curation is an integral part of the broader Data Processing workflow. Data Processing encompasses the entire lifecycle of data manipulation, from collection to final analysis and presentation. Within this context, Data Curation ensures that the data remains valuable and usable over time, supporting the various stages of data processing.

Integration with Data Processing

Data curation plays a vital role throughout the data processing lifecycle, ensuring that data is properly managed, maintained, and preserved for analysis and future use. In the initial stages, data curation involves collecting, annotating, and organizing data to ensure it is structured and ready for processing. This step is crucial in preparing datasets for effective analysis, reducing inconsistencies, and enhancing data reliability. As data processing progresses, ongoing management of curated data ensures that it remains accurate, up-to-date, and relevant to evolving requirements. By maintaining data quality and consistency, curation minimizes errors and enhances the overall integrity of the dataset. Furthermore, long-term usability is a key aspect of data curation, as it supports the preservation of data integrity while providing essential documentation and metadata. This ensures that datasets remain accessible and interpretable over time, facilitating reproducibility and effective data-driven decision-making.

5.4 Data Curation in Data Preprocessing

Data Preprocessing is a subset of Data Processing, specifically focused on preparing raw data for analysis or modeling. Data Curation plays a crucial role in this preparatory phase by ensuring that the data is clean, well-documented, and appropriately structured for analysis.

Integration with Data Preprocessing

Data curation plays a crucial role in various stages of data preprocessing, ensuring that datasets are well-managed, structured, and optimized for analysis. In data cleaning, curation provides essential metadata and contextual information that aids in identifying and rectifying errors or inconsistencies within the dataset. By maintaining accurate documentation, it enhances the reliability of cleaning processes and minimizes data-related anomalies. Additionally, data transformation benefits from proper data curation, as well-annotated and organized datasets facilitate conversion into formats that are more suitable for analysis, such as normalization, encoding, and scaling.

Moreover, feature engineering relies on curated data to generate meaningful features that improve model performance. High-quality data, supported by detailed documentation, ensures that relevant attributes can be derived effectively, enhancing predictive capabilities. Lastly, data splitting is streamlined through data curation, as well-structured datasets enable accurate partitioning for model training and validation. Properly annotated data ensures that subsets maintain statistical integrity, leading to more robust and generalizable machine learning models.

Example: Dataset Curation

In the following sections we use the downloaded and previously **cleaned** data files:

1. `USCensus1990.data.csv`
2. `Admission_Predict.csv`
3. `hfi_cc_2018_cleaned.csv`

Note that there are two kinds of dataset cleaning tasks before a machine learning model development can begin:

1. Cleaning the data so the framework understands the data right, i.e. formatting, removing confusing symbols, quotes, etc.
2. Cleaning (preprocessing) the data to improve the ML task, i.e. imputing values, removing outliers, removing incorrect dataset values, deriving variables, selecting variables, etc.

Both cleaning steps are crucial in preparing the dataset for model development.

`"As data scientists, our job is to extract signal from noise."` (ref: KDnuggets)

Graduate Admissions data

```
import pandas as pd
# Locate and load the data file
df = pd.read_csv('/EP_datasets/Admission_Predict_Ver1.1.csv')
# Sanity check
```

```
print(f'N rows={len(df)}, M columns={len(df.columns)}')
df.to_csv('table0203.csv', sep=',', encoding='utf-8')
```

Let's see what our data files contain using the script above. The output: `N rows=500, M columns=9`

The `DataFrame`:

Breast Cancer Dataset

Consider the breast cancer dataset file located at the module page. Load it with `pandas` library and check for (1.) duplicates, (2.) missing values, (3.) incorrect entries. In the following sections, for each problem that the dataset has, a correction is provided once the situation is determined.

```
import seaborn as sns; sns.set(style="ticks", color_codes=True)
# Locate and load the data file
df = pd.read_csv('/EP_datasets/breast_cancer_raw.csv')
print(f'N rows={len(df)} M columns={len(df.columns)}')
# Print some info and plots to get a feeling about the dataset
print(df.dtypes)
# Make sure use a '_variable' name to avoid shadowing variable names in other
 ↪  cells
def plot_bc(_df, xyscale=None):  # xyscale to use on the plots
    g = sns.FacetGrid(_df, col='deg-malig', hue='recurrence')
    g.fig.set_dpi(72)
    g.map(plt.scatter, 'age', 'tumor-size', alpha=.7)
    g.add_legend()
    if xyscale is not None:
        plt.xlim(xyscale[0], xyscale[1])
        plt.ylim(xyscale[0], xyscale[1])
plot_bc(df)
```

> Output:
>
> ```
> N rows=298 M columns=10
> age            float64
> menopause       object
> tumor-size     float64
> inv-nodes      float64
> node-caps       object
> deg-malig        int64
> breast          object
> breast-quad     object
> irradiat        object
> recurrence      object
> dtype: object
> ```

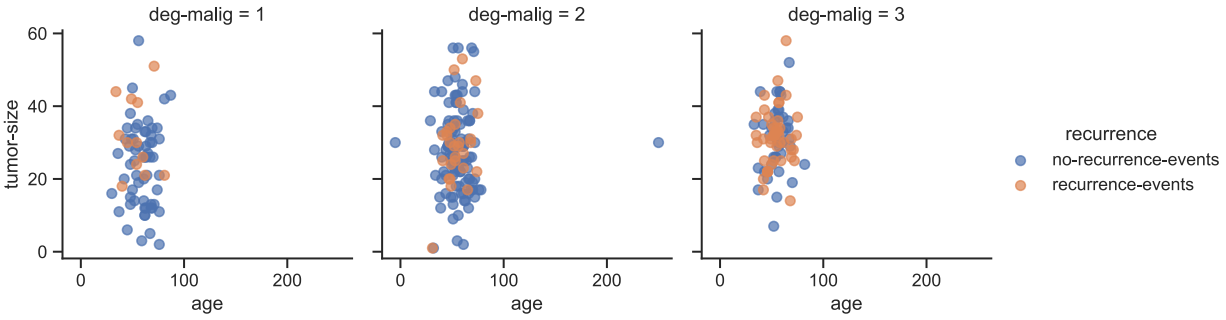

Figure 2: Breast Cancer Dataset Features

5.5 Summary

Data Curation is a vital component of both Data Processing and Data Preprocessing in data science. It ensures that data remains accurate, relevant, and usable throughout its lifecycle, supporting effective data analysis and decision-making. By integrating data curation activities into the data processing and preprocessing workflows, data scientists can enhance the quality and value of their datasets, leading to more reliable and insightful outcomes.

6 Data Cleaning

Data cleaning, also known as data cleansing, is an essential step in data preprocessing, particularly in data analytics, data science, and AI. The process of data cleansing involves recognizing and rectifying inconsistencies or inaccuracies in data from a record set, table, or database. This includes recognizing incomplete, incorrect, or irrelevant parts of the data and then replacing, modifying, or deleting the coarse data, noisy data, outliers, and repetitive data to enhance its quality and efficiency. In some application areas of data science, data collection and data cleaning are critical to the entire analysis process. By doing so, it improves data quality and hopes to increase overall productivity and robustness along with reduce error rate. In addition, this step may take data scientists a very large potion of their time.

Here's an overview of the data cleaning process, techniques, and considerations:

6.1 Common Data Cleaning Tasks

- **Handling Missing Values**: Managing missing values is a critical aspect of data preprocessing, as incomplete data can negatively impact model performance. One approach is to remove rows or columns containing missing values when the proportion of missing data is minimal. However, in cases where class labels are associated with the data, it is more effective to impute missing values based on metrics derived from the corresponding class rather than the entire dataset. Various imputation techniques can be applied, including statistical methods such as mean, median, and mode substitution, or more advanced approaches like k-Nearest Neighbors (k-NN) imputation, which estimates missing values based on the similarity of existing data points.
- **Removing Duplicates**: Identifying and eliminating duplicate records is an essential step in data preprocessing to prevent redundancy and ensure data integrity. Duplicate rows may arise due to data entry errors, repeated observations, or merging datasets from multiple sources. Removing these duplicates helps maintain dataset quality, reduces bias, and improves the efficiency of machine learning models by ensuring that repeated instances do not distort patterns or influence model training disproportionately.
- **Correcting Data Types**: Ensuring that each attribute in a dataset has the appropriate data type is a crucial step in data preprocessing. Data attributes should be correctly classified as numerical, categorical, or date-time to enable accurate computations and meaningful analysis. Incorrect data types can lead to errors in mathematical operations, misinterpretation of categorical variables, and inconsistencies in data transformations. By validating and converting data types where necessary, preprocessing enhances data consistency, improves model performance, and ensures compatibility with machine learning algorithms.
- **Detecting and Handling Outliers**: Outliers are extreme values that deviate significantly from the rest of the data and can arise due to measurement errors, data entry mistakes, or genuine variability in the dataset. Identifying and managing these outliers is essential to prevent them from skewing statistical analyses or negatively impacting machine learning models. Common techniques for handling outliers include clipping, where extreme values are limited

within a defined range, and transformation, such as logarithmic or Z-score normalization, to reduce the influence of extreme values while preserving data distribution characteristics.

- **Standardizing Values**: Ensuring data consistency is a crucial aspect of preprocessing, as variations in format, case, or units of measurement can introduce inconsistencies that affect analysis and model performance. Standardization involves transforming data into a uniform structure, such as converting all text to lowercase to avoid case-sensitive mismatches or ensuring numerical values adhere to a common unit of measurement. Additionally, techniques like normalization and standardization, which rescale numerical features to a consistent range or distribution, play a vital role in data transformation. These methods are further detailed in the Data Transformation section.
- **Error Correction**: Data preprocessing involves identifying and rectifying errors that may arise from human mistakes or system inconsistencies, such as typographical errors, misclassified categories, or incorrect numerical entries. These errors can significantly impact data integrity and analytical outcomes, making their detection and correction essential for reliable machine learning models. Automated techniques, including large language models (LLMs), can assist in error detection and correction by leveraging pattern recognition and contextual understanding to refine textual and categorical data efficiently.
- **Normalization, Standardization, and Scaling**: Ensuring consistency in data formatting is essential for effective data analysis and machine learning. This process involves transforming text data into a uniform structure, such as converting all text to lowercase to avoid case-sensitive mismatches, and standardizing numerical values to a common unit of measurement. Additionally, numerical features often need to be scaled to a standardized range, such as 0-1 normalization or Z-score standardization, to prevent features with larger magnitudes from dominating those with smaller scales. Proper scaling ensures that machine learning models interpret and process data more effectively. These techniques, including normalization and standardization, are further explored in the Data Transformation section.

Reflection on data cleaning is essential. There are several factors to consider when cleaning data, such as accuracy, completeness, consistency, and uniformity. Accuracy refers to the degree to which the data is correct and reliable. Completeness is the extent to which the data contains all the necessary information. Consistency is the degree to which the data is consistent with other data sets. Uniformity is the extent to which the data is formatted in a consistent manner. All of these considerations are important when cleaning data.

- Understanding the Data: Before cleaning, it's essential to understand the data, its context, and what constitutes an error or inconsistency.
- Documentation: Keeping track of all the cleaning steps to ensure reproducibility.
- Ethics and Compliance: Ensuring that data cleaning doesn't violate privacy or other legal regulations.
- Data Verification: Assessing the quality of data on an ongoing basis during the cleaning process is known as data verification.

Example: Data Cleaning Removing Duplicates

Duplicates

Let's check duplicate values in our dataset.

```
# Check for duplicates, this adds a new column to the dataset
df["is_duplicate"]= df.duplicated()
# Note that when using f-strings, the internal quote character must be
↪  different,
# such as 'is_duplicate' above
print(f"#total= {len(df)}")
print(f"#duplicated= {len(df[df['is_duplicate']==True])}")
# Drop the duplicate rows using index - best way to drop in pandas
index_to_drop = df[df['is_duplicate']==True].index  # have to use '==' instead
↪  of 'is'
df.drop(index_to_drop, inplace=True)
# Remove the duplicate marker column
df.drop(columns='is_duplicate', inplace=True)
print(f'#total= {len(df)}')
```

> Output:
>
> ```
> #total= 298
> #duplicated= 5
> #total= 293
> ```

Observe: Total number of rows (data points) reduced to 293.

Example: Data Formatting and Cleaning

Machine learning frameworks, such as `pandas`, `scikit-learn`, and `Weka` expect dataset files to be in certain formats to be able to process them. The Comma Separated Values **CSV** is one of the most common file formats. Such as, the file `breast_cancer_raw.csv` [**CourseDatasetFile**] and first 5 rows,

When examining datasets, sometimes we see the files might contain artifacts:

- single quotes in double quotes, i.e., `"Cote d'Azor"` or reversed? e.g., `'Cote d"Azor'`
- single quotes to differentiate between strings and values, i.e., `'1'` or `1`
- use of semicolon delimiter instead of commas, e.g., `1;50;red;` in a row

In addition to the data formats artifacts, we might also see:

- duplicates of data rows
- missing values (marked as `'?'` in Weka or `'NaN'` in pandas for numerical variables)
- incorrect entries (e.g. clerical errors)

Note that framework programs such as Weka learners are mature and strong enough to work with these problems without necessitating us cleaning them by a preprocessing stage. However, if we do the preprocessing ourselves, then we always increase the **quality of the dataset**, and this helps the following stages of the machine learning pipeline.

Missing Values

Let's impute missing values. If we do not handle missing values, the ML algorithms will often handle them internally.

The safest and most common approach: Use **mean** (or equally acceptable **median**) for numerical values and **mode** for nominal values to **impute** missing values. Note that a variable is the entire feature or column of data.

Mean: $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$

Median: $\tilde{x} = \frac{x[|x|/2] + x[|x|/2+1]}{2}$

Mode: $\hat{x} = \underset{x}{\texttt{argmax}} f(x)$

Mean vs. Mode

- Mean is the **average value** of the feature, mode is the **most frequent level** in the feature
- Mean is proper for numerical, mode is proper for nominal features
  - ◇ e.g., Mode might end up being 1 in a large column of real numbers when all levels are expressed for just once
- Mode is not sensitive to noise or outliers
- Mean value might not exist in the column, mode value is the most frequent level

```
# Do we have NaN in our dataset?
df.isnull().any()
# We do have NaN - three numerical variables - check first cell, it says float
 ↪  64
display(df[df['age'].isnull()])
display(df[df['tumor-size'].isnull()])
display(df[df['inv-nodes'].isnull()])
# Mean values of numerical columns
means = {c:df[c].mean() for c in df.columns if df[c].dtype != object}
print(f"mean-age= {means['age']}")
print(f"mean-tumor-size= {means['tumor-size']}")
print(f"mean-inv-nodes= {means['inv-nodes']}")
# Impute
df['age'] = df['age'].fillna(means['age'])
df['tumor-size'] = df['tumor-size'].fillna(means['tumor-size'])
df['inv-nodes'] = df['inv-nodes'].fillna(means['inv-nodes'])
```

```
# Check with the previous cell results
display(df.loc[[24,25,26,27,28]])
```

> Output:
>
> ```
> mean-age= 56.261168384879724
> mean-tumor-size= 28.343642611683848
> mean-inv-nodes= 3.5753424657534247
> ```

Missing Nominal Values

Finding missing values in nominal variables is more tricky. First, let's look at the nominal variables and then see what unique values these nominal variables take. i.e., this is the **level** of the nominal variable drawn from a finite alphabet. Unless a numerical type (`int64`, `float64`, etc.) `df.dtype` will correspond to an **object** which is a `object` class after being read into a `DataFrame` from a CSV file.

It is generally accepted to impute the **mode** of the feature when a level is missing, such as `no` for the missing `node-caps` levels as in below.

```
# Check unique levels and see any marker is used for a missing level
for col in df.columns:
    if df[col].dtype == object:
        print(col, df[col].unique())
```

> Output:
>
> ```
> menopause ['premeno' 'ge40' 'lt40']
> node-caps ['no' 'yes' '?']
> breast ['right' 'left']
> breast-quad ['left_up' 'central' 'left_low' 'right_up' 'right_low' '?']
> irradiat ['no' 'yes']
> recurrence ['no-recurrence-events' 'recurrence-events']
> ```

The variables `node-caps` and `breast-quad` has `'?'` levels which need to be **imputed** with values to help the preprocessing. Note that some classifiers in `sklearn` do not accept data points with `NaN` values.

```
# Check the next feature
display(df['node-caps'].value_counts())
print('mode-node-caps', df['node-caps'].value_counts().index[0])
```

> Output:
>
> ```
> menopause ['premeno' 'ge40' 'lt40']
> node-caps ['no' 'yes' '?']
> breast ['right' 'left']
> breast-quad ['left_up' 'central' 'left_low' 'right_up' 'right_low' '?']
> ```

```
irradiat ['no' 'yes']
recurrence ['no-recurrence-events' 'recurrence-events']
```

```
# Check the next feature
display(df['node-caps'].value_counts())
print('mode-node-caps', df['node-caps'].value_counts().index[0])
```

Output:

```
node-caps
no     227
yes     56
?       10
Name: count, dtype: int64
mode-node-caps no
```

```
# Check the next feature
display(df['breast-quad'].value_counts())
print('mode-breast-quad', df['breast-quad'].value_counts().index[0])
```

Output:

```
breast-quad
left_low     111
left_up       99
right_up      33
right_low     26
central       23
?              1
Name: count, dtype: int64
mode-breast-quad left_low
```

```
# Replace '?' with mode - value/level with the highest frequency in the feature
df['node-caps'] = df['node-caps'].replace({'?':'no'})
df['breast-quad'] = df['breast-quad'].replace({'?':'left_low'})
```

Incorrect Entries

Remember the age value 250 from previous sections?

Finding incorrect entries is more difficult than the previous steps as incorrect entries truly depend on the data column and **domain knowledge**. For this step, we will look at the plots of numerical columns and figure out possible incorrect entries, such as outliers. Also, subject-matter experts (SMEs) would help significantly in real-world projects about incorrect entries.

Correcting the incorrect entries may not be easy (or possible at all), and sometimes the best is dropping that data point.

6.2 Summary

Data cleaning is an essential step in the data processing cycle, which can have a major effect on the precision and dependability of any following examination or modeling. It necessitates cautious arranging, execution, and approval. For Python data cleansing pandas and NumPy are great packages to use for various techniques in cleansing data. Seaborn is a great package for statistical data visualization to identify and analyze the various tools and techniques used.

7 Data Transformation

In addition to data cleaning, certain algorithms require data feature properties in certain ways, such as **discretization**, **normalized** and **standardized**, to make the method work better. Data transformation is an integral part of data preprocessing in data analytics, data science, and AI. It involves converting data from its original form into a format that makes it more suitable for analysis or modeling. After features are generated it is necessary to preprocess the features that are to be used for classification. In many practical situations the classification model may receive input features whose values lie within different dynamic ranges. Thus, features with large values may inadvertently influence classification over features with small values. Another problem arises when a particular sample is not within the same area as the other features. Some of the procedures used for data preparation are feature standardization (Dillon and Goldstein, 1984, pp. 12-13) [9], feature min-max normalization (Theodoridis and Koutroumbas, 2006, pp. 214-215) [32], min-max global normalization (Guyon et al., 2006, pp. 254) [14], sigmoid normalization (Theodoridis and Koutroumbas, 2006, pp. 214-215) [32] and softmax scaling (Theodoridis and Koutroumbas, 2006, pp. 214-215) [32].

- Discretization
- Data Transformation
- Normalization and Standardization

7.1 Discretization

Discretization is the process where a numerical variable is mapped to some levels by binning. This step is a research/engineering area in machine learning. Recall that an example was provided in the past modules where the target (dependent) variable was discretized into three levels.

For our purposes, in this step, we will do the post-discretization, and apply one hot encoding to a nominal/discretized variable. Note that the variable might be a nominal variable naturally, such as the `'breast'` variable which takes values from the alphabet `'left'`, `'right'`.

Generally we keep the dependent variable as integer even if the cardinality is more than 2.

Now, we would like to continue preparing (preprocess) the dataset further to meet the requirements of the classifier that we would like to use - Random Forest classifier from `scikit-learn` library. This classifier works only on numerical data, thus we will convert the nominal variables into one hot encoded numerical variables, as explained in previous modules.

```
# pandas get_dummies function is the one-hot-encoder
def encode_onehot(_df, _f):
    _df2 = pd.get_dummies(_df[_f], prefix=_f, prefix_sep=' - ', dtype=int)
    _df3 = pd.concat([_df, _df2], axis=1)
    _df3 = _df3.drop([_f], axis=1)
    return _df3

# Print nominal variables
for f in list(df.columns.values):
```

```
    if df[f].dtype == object:
        print(f)
```

> Output:
>
> ```
> menopause
> node-caps
> breast
> breast-quad
> irradiat
> recurrence
> ```

```
# Display the original
display(df['menopause'][:10])

# Apply the onehot-encoding method
df_o = encode_onehot(df, 'menopause')

# Check the onehot-encoded version of this feature
cols = []
for f in list(df_o.columns.values):
    if 'menopause' in f:
        cols += [f]

display(df_o[cols][:10])
```

Output:

	menopause - ge40	menopause - lt40	menopause - premeno
0	0	0	1
1	0	0	1
5	1	0	0
6	1	0	0
7	0	0	1
8	0	0	1
9	0	0	1
10	0	0	1
11	0	0	1
14	1	0	0

Table 2: Menopause data table

7.2 Example - Convert Nominal Features to Numerical Features

Now that we preprocessed our dataset, let's transform it to help the learning model perform better. Load the cancer dataset and convert nominal features to numerical features, for the moment, disregarding ordinal or non-ordinal properties of the variables. A nominal `age` variable might be ordinal, but a nominal `country` variable may not. Our goal is to rank the features for classification without one-hot encoding them. Note that if one-hot encoding is used, the rank or score of a level of the variable will be lower than the entire variable itself. For example, if there are three levels of

a nominal variable, then make sure to sum up the scores of those three levels that will make up the total score of the variable.

```
# Locate and load the data file
df = pd.read_csv('/EP_datasets/breast_cancer_preprocessed.csv')
# Convert every feature to numericals
df['recurrence'] = df['recurrence'].astype("category").cat.codes

df['menopause'] = df['menopause'].astype("category").cat.codes.astype('float')
df['node-caps'] = df['node-caps'].astype("category").cat.codes.astype('float')
df['breast'] = df['breast'].astype("category").cat.codes.astype('float')
df['breast-quad'] =
↪  df['breast-quad'].astype("category").cat.codes.astype('float')
df['irradiat'] = df['irradiat'].astype("category").cat.codes.astype('float')

df['deg-malig'] = df['deg-malig'].astype('float')
```

Important: Note that encoding the non-ordinal nominal variables numerically is not always the correct approach. Some algorithms are impervious (e.g., decision tree), but others are sensitive (e.g., SVM, NN). The demonstration here is ranking the variables as in the next section.

7.3 Normalization and Standardization

In this subsection we use min-max normalization (feature normalization) and zero-mean normalization (feature standardization) along with a description of both methods. The training vectors in this section are represented by $x_i = [x_1, x_2, , x_n] \in \mathbf{X}$ with a dimension of d and the number of samples defined as n.

Min-max normalization

Mapping the values of a column to $[0, 1]$ range is normalization: $\dfrac{x_i - \min(x)}{\max(x) - \min(x)}$

The Min-max normalization performs a linear scaling on the original data. The normalization is calculated by estimates of the minimum and maximum of the values. The normalization technique is defined for the n available data samples and the k^{th} feature as:

$$\hat{x}_{ik} = \frac{x_{ik} - min(x_k)}{max(x_k) - min(x_k)}(b - a) + a, \qquad k = 1, 2, ..., l \text{ and } i = 1, 2, ..., n \tag{1}$$

where a and b are scaling factors. When a = 0 and b = 1 the individual feature values are in the range of [0,1]. In the event that the denominator of Equation (2.18) is equal to zero that feature is removed, avoiding the potential of normalizing a feature of constants.

Mapping the values of a column to $[0, 1]$ range is normalization: $\dfrac{x_i - \min(x)}{\max(x) - \min(x)}$

Z-score Normalization (Standardization)

Standardization is mapping the values to a 0-mean 1-standard-deviation distribution: $\frac{x_i - \text{mean}(x)}{\text{stdev}(x)}$

The Z-score normalization is based on the mean and standard deviation of each feature. Each feature in this method is separately standardized by subtracting its mean and dividing by the standard deviation as follows:

$$\hat{x}_{ik} = \frac{x_{ik} - \mu_k}{\sigma_k}, \qquad k = 1, 2, ..., l \text{ and } i = 1, 2, ..., n \tag{2}$$

where μ_k and σ_k are defined as:

$$\mu_k = \frac{1}{n} \sum_{i=1}^{n} x_{ik} \tag{3}$$

$$\sigma_k = \left(\frac{1}{n-1} \sum_{i=1}^{n} \left(x_{ik} - \mu_k \right)^2 \right)^{1/2} \tag{4}$$

and m is the number of samples. In the event that the standard deviation of a particular feature is zero (e.g., each element of the observed feature is a constant value), the feature is discarded.

Note: That $\hat{x}_{ik}$ in Equation (1) and Equation (2) are not equal to each other, the two equations produce two different and unique normalized sets of values.

Example: Data Transformation

Now that we preprocessed and used the data for classification we can move to other interesting problems.

Imagine, we did not have the ground truth, so that a supervised learning was not possible. A natural approach in this case is clustering the data to see if there are some patterns or models we can come up with that explains the cancer behavior. We will attempt answering questions like *"Is there a direct relation between menopause and cancer?"*

First, let's draw some plots where the x, y and z-dimensions are `'age'`, `'tumor-size'`, `'inv-nodes'` and color is `'recurrence'`.

```
df2 = df.copy()

# Convert every feature to numbers
df2['recurrence'] = df['recurrence'].astype("category").cat.codes

df2['menopause'] = df['menopause'].astype("category").cat.codes.astype('float')
df2['node-caps'] = df['node-caps'].astype("category").cat.codes.astype('float')
df2['breast'] = df['breast'].astype("category").cat.codes.astype('float')
```

```
df2['breast-quad'] =
 ↪  df['breast-quad'].astype("category").cat.codes.astype('float')
df2['irradiat'] = df['irradiat'].astype("category").cat.codes.astype('float')

df2['deg-malig'] = df['deg-malig'].astype('float')

def draw3d(_df, _mn, _mx, fn=None):
    fig = plt.figure(dpi=300)
    ax = fig.add_subplot(111, projection='3d')
    ax.set_xlim3d(_mn, _mx)
    ax.set_ylim3d(_mn, _mx)
    ax.set_zlim3d(_mn, _mx)
    ax.set_ylim(ax.get_ylim()[::-1])
    ax.scatter(_df['age'], _df['tumor-size'], _df['inv-nodes'],
     ↪  c=_df['recurrence'], s=30)
    ax.set_xlabel('age'); ax.set_ylabel('tumor-size'); ax.set_zlabel('inv-nodes')
    if fn is not None:
        plt.savefig(fn, dpi=300)

draw3d(df2, 0, 100, fn='fig0206.svg')
```

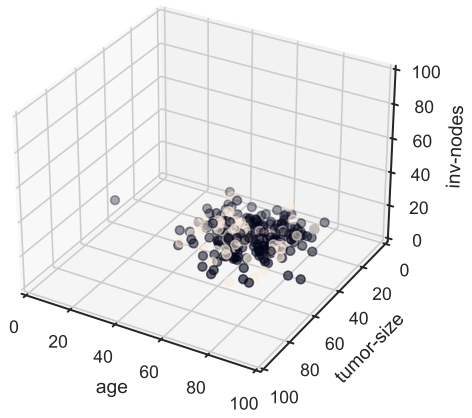

Figure 3: Breast Cancer Dataset Cluster Analysis

Question: Do the dimensions `'age'`, `'tumor-size'`, `'inv-nodes'` look fine in the above 3D plot?

Answer: The features are clumped and not nicely occupy $[0 - 100]$ range, i.e. we are not seeing a spherical cluster shape.

Let's cluster the cancer data, without using the ground truth. We have to convert the nominal variables to numerical by using the category codes like we applied to `'recurrence'` variable.

`Important:` Make sure every variable is of the same type, e.g. `float32`.

Important: Note that the values `'recurrence'` took {0,1}, and by looking at the 3D plot above, can we easily find out which values (`0` or `1`) corresponds to `'recurrence-events'` levels?

```
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score

def kmeans(_X, _y, niter):  # do it niter times to collect statistics
    accuracies = []
    for _ in range(niter):
        # We know that there are two levels in target variable - thus
        ↪  n_clusters=2
        km = KMeans(n_clusters=2, random_state=0, n_init=10)
        clusters = km.fit_predict(_X)
        accuracies += [accuracy_score(_y, clusters)]
    return np.mean(accuracies)

X = df2.loc[:, df2.columns != 'recurrence'].values
y = df2.loc[:, df2.columns == 'recurrence'].values.ravel()

print(f'Clustering error= {kmeans(X, y, 100):.3f}')
```

> Output:
>
> ```
> Clustering error= 0.509
> ```

Above performance is not very good as the error is almost equivalent to random choice, which would be $\frac{1}{2}$ since we have 2 classes.

Normalization makes the **optimization surface** more **spherical**, which helps the optimizer using each feature with equal importance. This is especially important and helping for **distance** based methods, such as neural networks, SVM, etc. Note that some probabilistic methods are Naive Bayes, decision trees, etc.

Let's try two scalers from `sklearn.preprocessing`

1. Normalization `MinMaxScaler()`
2. Standardization `scale()`

```
from sklearn import preprocessing

min_max_scaler = preprocessing.MinMaxScaler()
df2[['age', 'tumor-size', 'inv-nodes']] =
↪  min_max_scaler.fit_transform(df2[['age', 'tumor-size', 'inv-nodes']])

draw3d(df2, 0, 1, fn='fig0207.svg')
```

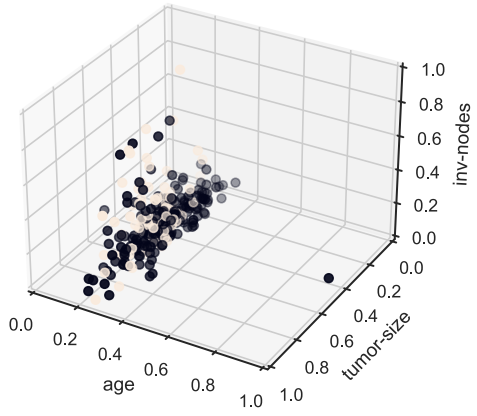

Figure 4: Breast Cancer Dataset Cluster Analysis (normalization)

By normalizing the values through expansion and contraction to $[0, 1]$ we achieve the **distance** between the data points are in the same "range" or unit. Thus, the distance metrics like Euclidean distance will weigh each **dimension** or feature **equally**.

Example: Data Transformation for Various Units

Imagine a dataset which has speed in miles $[0, 100]$ and time traveled in seconds $[0, 43200]$ (12 hours max). A proper approach would be mapping both features into $[0, 1]$ scale to treat the feature space spherically. For actual feature values an inverse transformation can be used to map back to the original units (for example to be presented to user).

A distance metric d in M dimensions (`Dataframe` has M number of columns) such as Euclidean $d_{ik} = \sqrt{\sum_{j=0}^{M}(x_{ij} - x_{kj})^2}$

As an example, clustering algorithms use some form of distance metric such as Euclidean distance between pairs of data points.

As can be seen from above example, normalization of variables is a necessary step for clustering.

```
df2[['deg-malig', 'breast-quad']] =
 ↪  min_max_scaler.fit_transform(df2[['deg-malig', 'breast-quad']])

X = df2.loc[:, df2.columns != 'recurrence'].values
y = df2.loc[:, df2.columns == 'recurrence'].values.ravel()

print(f'Clustering error= {kmeans(X, y, 100):.3f}')
```

> Output:
>
> ```
> Clustering error= 0.512
> ```

And now standardization.

```
df2[['age', 'tumor-size', 'inv-nodes']] = preprocessing.scale(df2[['age',
↪  'tumor-size', 'inv-nodes']])
draw3d(df2, 0, 10, fn='fig0208.svg')
```

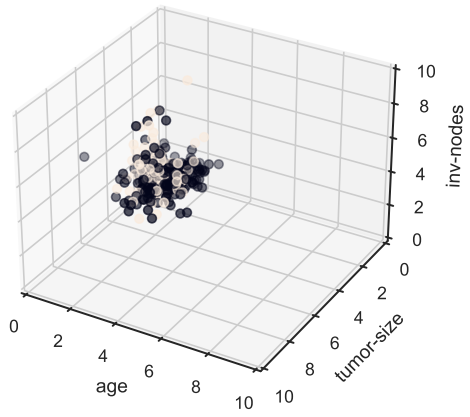

Figure 5: Breast Cancer Dataset Cluster Analysis (standardization)

```
# Scaled
plot_bc(df2, xyscale=[-3,3], fn='fig0209.svg')
```

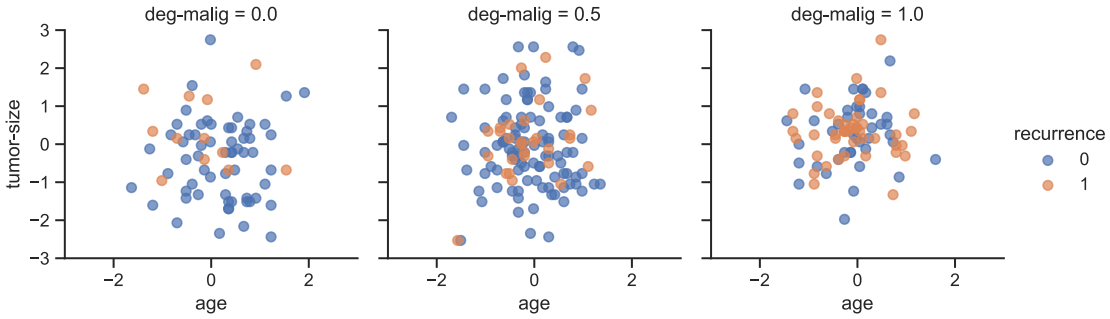

Figure 6: Breast Cancer Dataset Features (scaled)

Question: Do you see any difference/improvement on the variables compared to the first set of plots in cell 1, repeated below?

Answer: Shapes are same but axis scales are different.

```
# Original
plot_bc(df, xyscale=[0,100], fn='fig0210.svg')
```

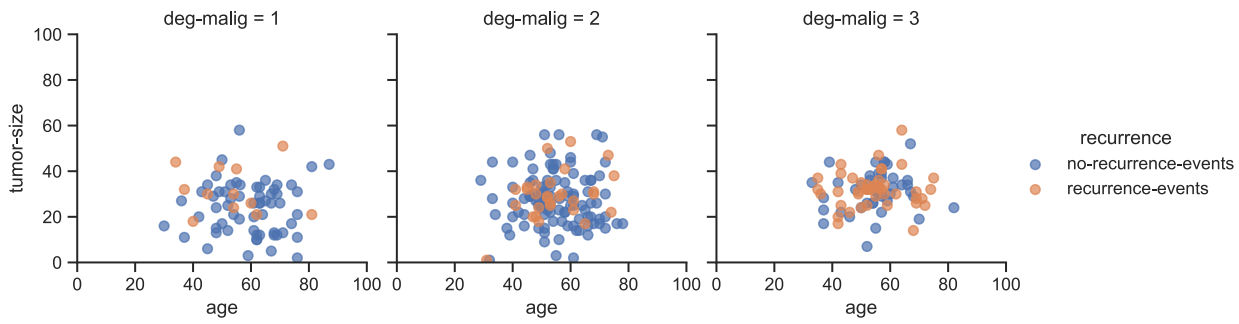

Figure 7: Breast Cancer Dataset Features (original)

Note that after variable transformation, variables become more spherical or Gaussian like, but then the levels or data points do not correspond to any meaningful value in the domain knowledge that the dataset originally belonged to. For example `'deg-malig'` had three levels {1, 2, 3} which probably meant something to the doctors dealing with cancer patients. However, depending on the dataset, such transformations make a difference, albeit a few percentage improvement on the performance.

7.4 Summary

In this section, we explored the fundamental concepts and techniques of data transformation, a critical step in data preprocessing that ensures datasets are properly structured for machine learning models. We began by discussing the necessity of transforming features, particularly when working with models that require specific feature properties such as discretization, normalization, and standardization. These transformations help improve model performance by addressing issues related to dynamic range differences and ensuring consistent feature scaling.

We introduced discretization, which involves mapping numerical variables to categorical levels, often followed by encoding techniques such as one-hot encoding. This transformation is essential for classifiers like Random Forest, which require numerical input. An example was provided using the breast cancer dataset, demonstrating how to convert nominal variables into numerical representations to enhance model compatibility.

Next, we covered normalization and standardization, highlighting two key methods: min-max normalization and Z-score standardization. Min-max normalization scales features to a defined range, often between $[0, 1]$, ensuring that numerical values are comparable. Standardization, on the other hand, transforms feature distributions to have zero mean and unit variance, which is particularly useful for distance-based models like k-means clustering and support vector machines (SVMs). We demonstrated these methods through Python implementations, visualizing the impact of scaling on feature distributions.

Additionally, we examined the role of data transformation in clustering algorithms, illustrating how normalization improves the structure of feature spaces and enhances the effectiveness of distance-based learning methods. Through hands-on Python examples, we observed how transforming variables into a uniform scale impacts clustering performance.

In summary, data transformation is a crucial preprocessing step that enhances model performance,

optimizes learning algorithms, and ensures consistency across datasets. The next sections will further build on these preprocessing techniques and their applications in real-world data science workflows.

8 Outliers

In the field of statistics, an outlier is an observation that is markedly different from the rest of the data. Outliers might be the result of variability in measurement, could represent new or unique information, or might arise from errors in the experiment. Such outliers can be excluded from the dataset if they are deemed to be errors. The presence of an outlier can signal an interesting discovery but can also pose challenges in statistical analysis.

In larger datasets, it's common to find some data points that deviate significantly from the mean. These deviations might be due to systematic errors, limitations in the theoretical models used, or simply because some observations are naturally distant from the data's central point. In such cases, outliers can point to data inaccuracies, incorrect experimental methods, or instances where certain theories may not apply. Nonetheless, in extensive datasets, a few outliers are normally expected and don't necessarily indicate any unusual phenomena.

Outliers are data points in a dataset that differ significantly from other observations. They can have a substantial impact on statistical analyses and machine learning models, potentially leading to misleading conclusions or poor predictions. Here is a closer look at outliers, how they can be identified, handled, and why they matter, especially in the fields of data analytics, data science, and AI.

Causes of Outliers:

Outliers in a dataset can arise due to various factors, some of which may introduce errors while others reflect genuine variations in the data. Mistakes in data entry can result in false outliers, where incorrect values deviate significantly from the expected range. Additionally, inaccurate readings may occur due to malfunctioning measurement devices or flawed measurement techniques, leading to inconsistencies in the data. However, not all outliers are erroneous; in certain cases, outliers may represent genuine and meaningful variations, providing valuable insights into underlying patterns, anomalies, or rare events within the dataset.

Identifying Outliers:

Barnett and Lewis (1994) [1] offer a range of instructions to identify a few outliers. If there are a lot of outliers, the analyst must be cautious.Here are a few methods to identify the outliers:

- Scatter plots, histograms, and box plots can be used to visually detect outliers.
- Z-scores, Tukey's fences, and the Mahalanobis distance are all popular statistical techniques used to identify outliers.
- Algorithms such as Isolation Forest and One-Class SVM can be utilized to identify anomalies, particularly in data with a high number of dimensions.

Assuming that the outliers are errors, they can be removed. In certain cases, the outliers themselves may be of interest and can be examined independently. Two strategies to eliminate outliers are used when dealing with multivariate outliers. The first is a method in which the mean is used to set an upper and lower boundary of a confidence interval to identify an outlier and take out the sample (Barnett and Lewis, 1994) [1]. The second is a multivariate outlier technique proposed by

Wilks (1963) [34]. The last method is the use of the Mahalanobis distance [23].

Outliers are an integral part of data analysis and preprocessing. It is essential to comprehend, recognize, and treat outliers correctly to obtain reliable and precise results. Outliers can distort the results and reduce the precision of models. Outliers can influence the comprehension of data patterns and associations. Managing outliers correctly can make models more resilient to changes in the data.

8.1 Mahalanobis Distance for Outlier Removal

$$D_i^2 = \left((x_i - \mu)\Sigma^{-1}(x_i - \mu)^T\right) \tag{5}$$

Here the Mahalanobis distance is calculated for each observation within a class using the class mean μ by feature and the class inverse covariance matrix Σ^{-1} [23]. Note that the following equation only differs from Equation 5 in that the square root is taken on each side of the equation as shown below.

$$D_i = \left((x_i - \mu)\Sigma^{-1}(x_i - \mu)^T\right)^{1/2} \tag{6}$$

In Figure 8.1 the Setosa flower type (species) is shown in blue, Versicolor flower type (species) is shown in red, and the Virginica flower type (species) is shown in green. The x-axis is the sepal length and the y-axis is the petal width for this figure.

The Mahalanobis distance can be used to identify the observations furthest from the mean by feature while accounting for the variance of each feature and the rotation of the data with the use of the covariance matrix. In this approach, the values returned from the Mahalanobis distance are sorted in descending with the value with the largest distance analyzed as a potential outlier. It should be apparent that the three observations for each class circled in black appear to be identified as outliers. These observations can be removed with the potential benefit of having a tighter standard deviation of the data being analyzed.

Example: Detecting Outliers using Simple Histogram

Outliers represent the most extreme values in a dataset, which might include the highest or lowest values. However, the highest and lowest values in a sample are not automatically considered outliers unless they are exceptionally distant from the other observations.

```
# Outlier
df.hist(column=['age', 'tumor-size', 'inv-nodes', 'deg-malig'],
        log=True, bins=30, sharey=True, figsize=(8, 8), layout=(2,2))
plt.savefig('fig0211.svg', dpi=300)
```

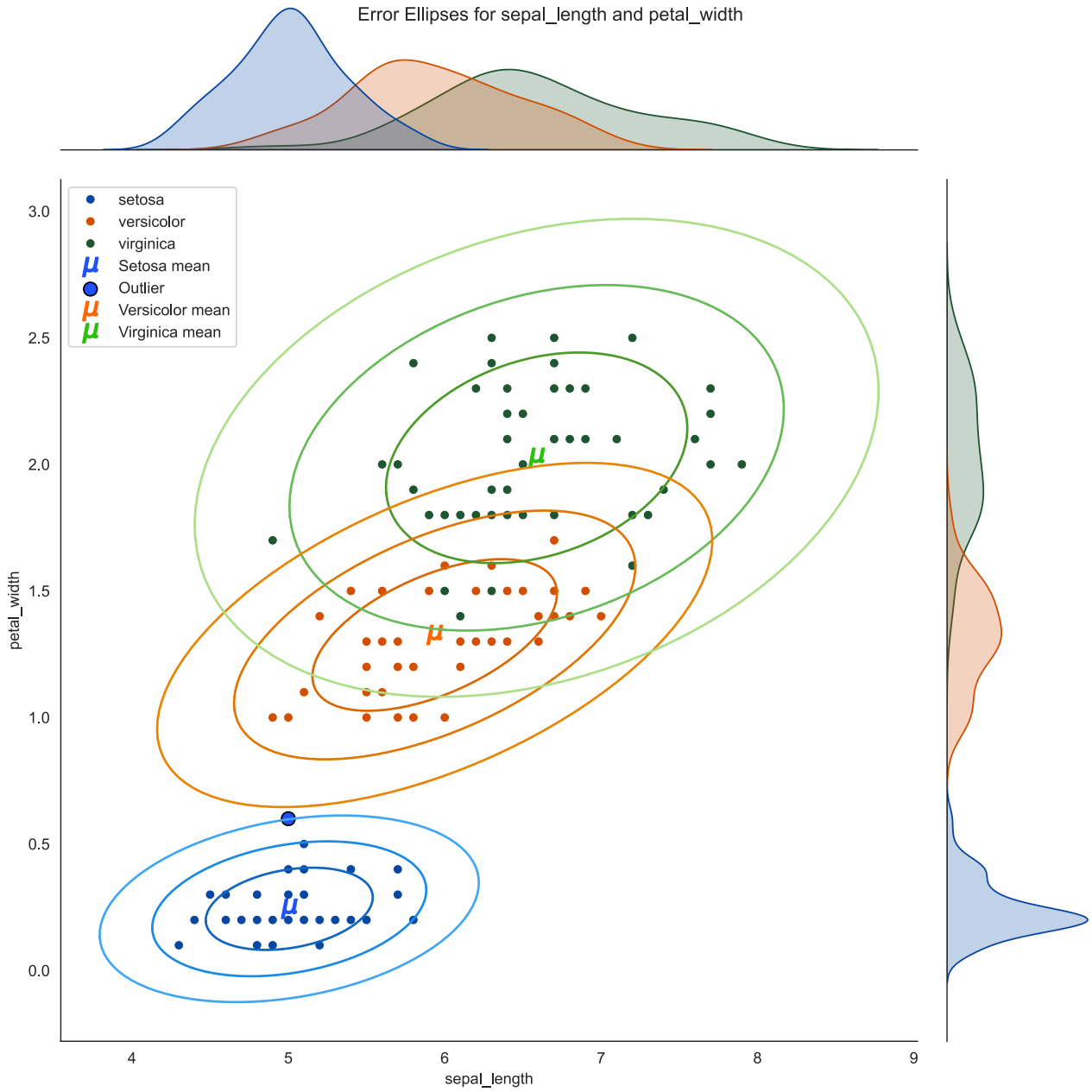

Figure 8: Outlier Identification - Here the Iris data by flower type has the observation with the largest observation identified based on the Mahalanobis distance greater than 3σ from the μ.

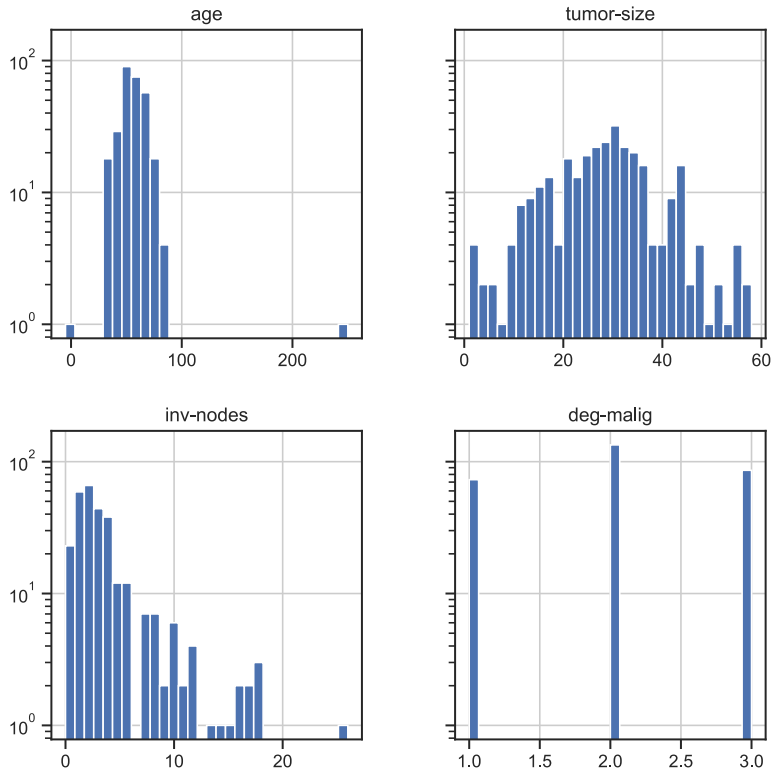

Figure 9: Breast Cancer Dataset Features (outliers)

The `age` and `inv-nodes` have data points that can be considered as outliers.

8.2 Summary

This section explored the concept of outliers, their causes, methods for identification, and approaches for handling them in data preprocessing. Outliers are observations that deviate significantly from other data points, and their presence can either indicate measurement errors, unique phenomena, or natural variations in data. They can distort statistical analyses, affect machine learning model performance, and lead to biased conclusions if not handled appropriately.

We examined the causes of outliers, which include data entry errors, measurement inaccuracies, and legitimate variations in data distribution. Various outlier detection techniques were discussed, including visual methods such as scatter plots and box plots, statistical techniques like Z-scores, Tukey's fences, and the Mahalanobis distance, as well as machine learning-based anomaly detection methods, including Isolation Forests and One-Class SVM.

A key focus was on Mahalanobis distance, a powerful technique for multivariate outlier detection that accounts for the relationships between different features. This method was demonstrated using an Iris dataset example, where outliers were identified based on their distance from the

mean within their respective classes. The visualization of Mahalanobis distance-based outlier detection illustrated its effectiveness in identifying extreme data points that could impact model performance.

Finally, we discussed outlier removal strategies, emphasizing that while some outliers may be discarded when they are errors, others might hold valuable insights and should be examined carefully. The choice of handling outliers depends on the dataset and the specific application in data science.

In summary, outliers are an integral aspect of data preprocessing that must be properly understood and managed. Effectively identifying and handling outliers improves model robustness, enhances accuracy, and ensures reliable data analysis. The next sections will build on these preprocessing techniques to further refine datasets for machine learning applications.

9 Feature Engineering

Feature engineering is a critical step in the machine learning pipeline that can greatly influence the performance of a model. It involves creating new features from existing ones, selecting only the most informative ones, or transforming features to a more suitable form. Here's an overview of feature engineering and its importance, especially in data analytics, data science, and artificial intelligence.

What is Feature Engineering? *Feature engineering is the process of using domain knowledge to generate features that make machine learning algorithms work. It is an art as much as a science, allowing the model to understand the data in a more nuanced way.*

Feature engineering can include two types,

- **Derived features** by mathematical transformations, such as discrete cosine, wavelets, derivative, linear combination, statistical measures, feature extraction, etc. In this type of feature engineering, new knowledge is not gained, but more proper features are generated.
- **Feature enrichment** by bringing in other data resources and datasets, such as adding more features from other datasets that help the problem at hand. In this type of feature engineering, knowledge and more patterns are expected to be added to the main dataset.

Importance of Feature Engineering:

Feature engineering plays a crucial role in machine learning by transforming raw data into meaningful representations that improve model efficiency and effectiveness. Well-crafted features can significantly enhance model performance by improving key metrics such as accuracy, precision, and recall. Additionally, selecting only relevant features helps in reducing computational resource consumption, allowing models to train faster while using less memory. Beyond performance gains, feature engineering also enhances interpretability, making machine learning models easier to understand and explain. Furthermore, integrating domain knowledge into feature engineering allows practitioners to bridge the gap between raw data and the underlying patterns a model needs to learn, ensuring that the extracted features align with real-world insights.

9.1 Example - Mathematical Transformation

In [28], the focus was on the fusion of two modalities, visual signals represented by images and vocal signals that encode sounds, to form a multi-modal dataset to use in supervised classification tasks [29]. As shown in Figure 10, a framework for generating features from raw data was presented. The raw image and vocal data is first pre-processed using discrete cosine transform (DCT), Discrete Fourier Transform (DFT), and wavelet decomposition using Daubechies wavelets. Subsequently, features are generated using Fisher's linear discriminant, eigenvalue decomposition, and first-order statistics. This allowed a set of features to be generated to represent the two modalities for the ML models for classifying the 10 classes.

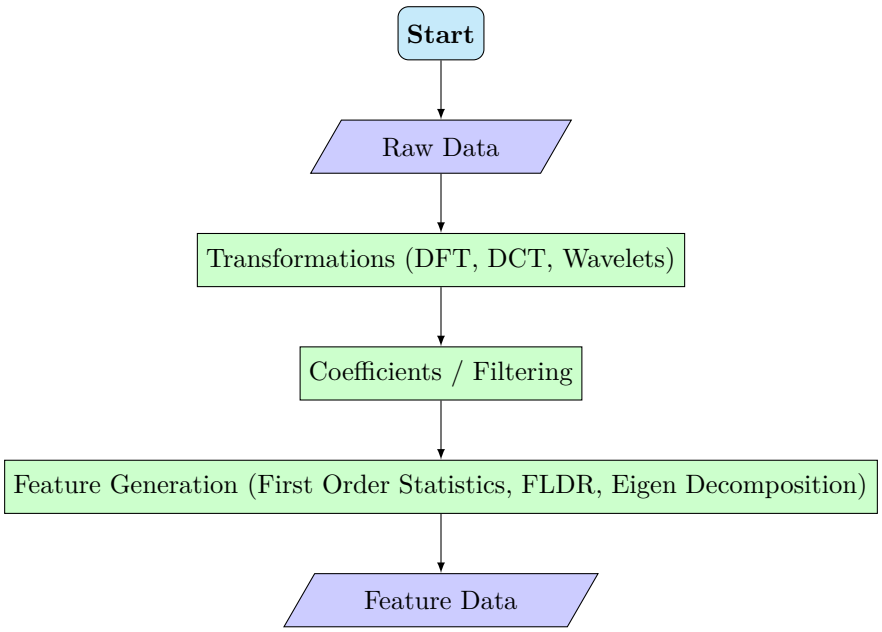

Figure 10: Feature Engineering process flow to generate features.

Feature engineering is a sophisticated and essential step requiring analytical skills and domain knowledge. It combines creativity, experimentation, and understanding of machine learning algorithms. In Data Science, AI, or Data Analytics, hands-on experience and insights into feature engineering allows tools to be built with more robust and effective models.

Note: This example will be used in the course to engineer features using a raw data set. There are other mathematical concepts that need to be presented before developing and implementing the new features.

9.2 Summary

Feature engineering is a crucial step in the data preprocessing pipeline that transforms raw data into meaningful representations, significantly improving machine learning model performance. This section covered the key aspects of feature engineering, including derived features and feature enrichment. Derived features involve applying mathematical transformations such as Fourier transforms, wavelet decomposition, and statistical measures to generate more suitable feature representations, while feature enrichment focuses on integrating additional datasets to enhance model performance.

The importance of feature engineering was emphasized in terms of its ability to improve accuracy, reduce computational resource consumption, enhance interpretability, and incorporate domain knowledge. Well-engineered features enable models to capture essential patterns and relationships in data, leading to better generalization and more reliable predictions.

A practical example was presented, illustrating how mathematical transformations such as Discrete Cosine Transform (DCT), Discrete Fourier Transform (DFT), and wavelet decomposition can be used to extract meaningful features from multi-modal data sources, including visual and vocal signals. This example demonstrated how a structured approach to feature engineering, involving

transformation, filtering, and feature selection, contributes to building robust machine learning models.

In summary, feature engineering is both a science and an art, requiring analytical skills, creativity, and domain expertise. It is an iterative process that involves experimenting with different feature transformations and evaluating their impact on model performance. The next sections will further explore feature ranking, selection, and dimensionality reduction techniques to optimize machine learning models by retaining the most informative features.

10 Feature Exploration

A data and feature exploration is crucial for any machine learning model. Ideally, the dataset feature exploration is conducted with a subject-matter expert (SME) next to the data scientist.

A feature exploration helps to understand the features, their relevance to the problem, and the dependent variable. Often, independent feature and dependent variable relations are visible, and machine learning algorithms find these patterns to build learning models.

Download the data file `Suicide Rates Overview 1985 to 2016` from the Kaggle website (by registering to Kaggle -using a disposable email address if necessary), and let's explore it.

```
# Visualizations
import seaborn as sns; sns.set(style="ticks", color_codes=True)
# Locate and load the data file
dfOrg =
 ↪  pd.read_csv('/EP_datasets/suicide-rates-overview-1985-to-2016/master.csv',
 ↪  thousands=',')
```

Let's see what our data files contain using the script above. The output:

	country	year	sex	age	suicides_no	population	suicides/100k pop	country-year	HDI for year	gdp_for_year ($)	gdp_per_capita ($)	generation
0	Albania	1987	male	15-24 years	21	312900	6.71	Albania1987	nan	2156624900	796	Generation X
1	Albania	1987	male	35-54 years	16	308000	5.19	Albania1987	nan	2156624900	796	Silent
2	Albania	1987	female	15-24 years	14	289700	4.83	Albania1987	nan	2156624900	796	Generation X
3	Albania	1987	male	75+ years	1	21800	4.59	Albania1987	nan	2156624900	796	G.I. Generation
4	Albania	1987	male	25-34 years	9	274300	3.28	Albania1987	nan	2156624900	796	Boomers
5	Albania	1987	female	75+ years	1	35600	2.81	Albania1987	nan	2156624900	796	G.I. Generation
6	Albania	1987	female	35-54 years	6	278800	2.15	Albania1987	nan	2156624900	796	Silent
7	Albania	1987	female	25-34 years	4	257200	1.56	Albania1987	nan	2156624900	796	Boomers

Checking the number of levels and asking about their semantic meaning to a Subject Matter Expert will help explore nominal features.

```
# Check unique values in generation
dfOrg['generation'].unique()
```

> Output:
>
> ```
> ['Generation X' 'Silent' 'G.I. Generation' 'Boomers' 'Millenials'
>  'Generation Z']
> ```

Examining the dependent variable and its behavior with respect to independent variables will inform the data scientists. In the following dataset, `suicides_no` is a variable to predict. Unless it is normalized to a rate such as $\frac{\texttt{suicides\_no}}{\texttt{population}}$, it is not possible to compare or predict.

```
# Plot the dependent variable
dfOrg['suicides/100k pop'].hist(bins=100, log=True)
plt.title('Normalized Dependent Variable')
plt.xlabel('suicides/100k pop')
plt.ylabel('Count')
```

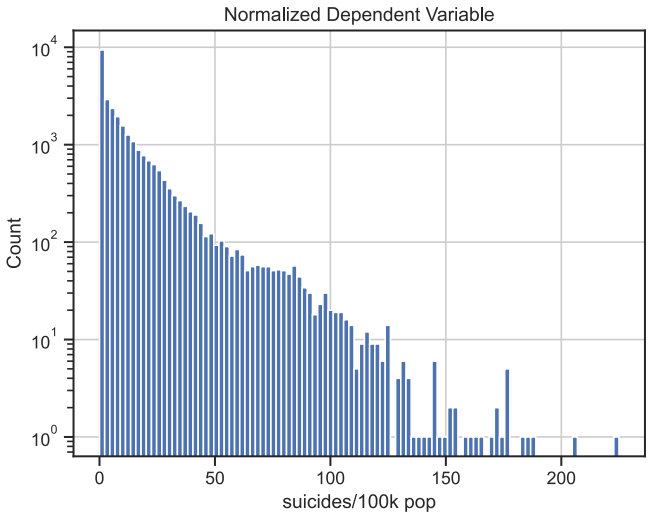

Figure 11: Normalized Dependent Variable

Exploring the data types will help guide data preprocessing, such as converting strings to nominal levels or creating bins to represent them as numerical levels.

```
# Check types of features
print(dfOrg.dtypes)
```

> Output:
>
> ```
> country                object
> year                    int64
> sex                    object
> age                    object
> suicides_no             int64
> population              int64
> suicides/100k pop     float64
> country-year           object
> HDI for year          float64
> gdp_for_year ($)       int64
> gdp_per_capita ($)     int64
> generation             object
> dtype: object
> ```

At every step, checking the `DataFrame` size will help avoid coding errors.

```
# Aggregate over sex, year and generation
df2 = dfOrg.copy()
df2 = df2.groupby(['sex', 'year', 'generation']).mean(numeric_only=True)
df2 = df2.reset_index()
print(len(df2))
```

> Output:
>
> 292

```
# Sanity
df2.to_markdown('table0303.md')
```

	sex	year	generation	suicides_no	population	suicides/100k pop	HDI for year	gdp_for_year ($)	gdp_per_capita ($)
0	female	1985	Boomers	101.958	1.80246e+06	4.74021	0.699162	1.92647e+11	6091.23
1	female	1985	G.I. Generation	136.125	1.12118e+06	9.50021	0.699162	1.92647e+11	6091.23
2	female	1985	Generation X	52.5104	2.03016e+06	2.54448	0.699162	1.92647e+11	6091.23
3	female	1985	Silent	197.417	2.58263e+06	5.83188	0.699162	1.92647e+11	6091.23
4	female	1986	Boomers	105.375	1.83648e+06	5.09021	nan	2.30225e+11	7126.1
5	female	1986	G.I. Generation	141.906	1.15195e+06	9.55073	nan	2.30225e+11	7126.1
6	female	1986	Generation X	53.375	2.05982e+06	2.54531	nan	2.30225e+11	7126.1
7	female	1986	Silent	209.312	2.66637e+06	5.59375	nan	2.30225e+11	7126.1

Many times, the subject matter experts would examine the feature relations. Below is a composition of four variables.

```
# Plot for data exploration
g = sns.FacetGrid(df2, col='sex', hue='generation')
g.map(plt.scatter, 'gdp_per_capita ($)', 'suicides/100k pop')
g.add_legend()
g.savefig('fig0304.svg', dpi=300)
```

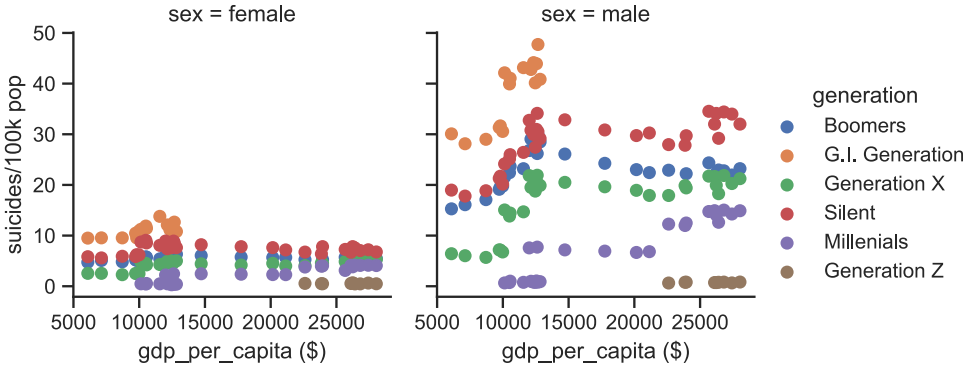

Figure 12: Features (gdp_per_capita, sex, generation) Related the Dependent Variable

Check the range of the year variable.

```
# Check the year range for the plot x-axis order
print(df2['year'].min(), df2['year'].max())
```

> Output:
>
> 1985 2016

More feature exploration.

```
# Plot for data exploration
g = sns.FacetGrid(df2, col='sex', hue='generation', height=4, aspect=1.5)
g.map(sns.barplot, 'year', 'suicides/100k pop', order=np.arange(1985,2017))
```

```
g.set_xticklabels(rotation=90, fontsize=9)
g.add_legend()
```

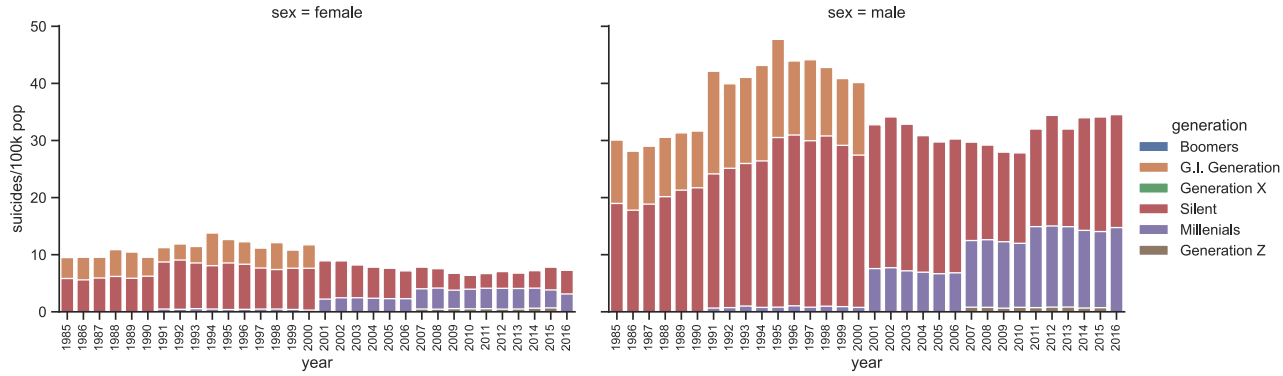

Figure 13: Feature Exploration

Aggregate the age and year variables to group data features. Check the length of the DataFrame again.

```
# Aggregate over age and year
df3 = dfOrg.copy()
df3 = df3.groupby(['age', 'year']).mean(numeric_only=True)
df3 = df3.reset_index()
print(len(df3))
```

> Output:
>
> 191

```
# Sanity
df3.to_markdown('table0306.md')
```

	age	year	suicides_no	population	suicides/100k pop	HDI for year	gdp_for_year ($)	gdp_per_capita ($)
0	15-24 years	1985	186.146	2.05182e+06	8.42969	0.699162	1.92647e+11	6091.23
1	15-24 years	1986	188.156	2.0754e+06	8.15208	nan	2.30225e+11	7126.1
2	15-24 years	1987	152.148	1.93137e+06	7.48787	nan	2.40386e+11	8712.59
3	15-24 years	1988	156.5	2.00036e+06	8.75092	nan	2.98568e+11	9983.86
4	15-24 years	1989	179.192	2.11468e+06	9.16048	nan	3.07081e+11	9725.04
5	15-24 years	1990	181.625	2.04863e+06	9.11516	0.7158	3.07896e+11	9806.33
6	15-24 years	1991	182.164	2.06003e+06	9.90563	nan	3.28259e+11	10132.9
7	15-24 years	1992	190.208	2.14407e+06	9.18654	nan	3.43728e+11	10506.7

```
# More plots
plt.figure(figsize=(6, 4), dpi=72)
sns.scatterplot(x='year', y='suicides/100k pop', hue='age', data=df3)
plt.xticks(rotation=90, fontsize=9)
plt.legend(bbox_to_anchor=(1.01, 1.0))
```

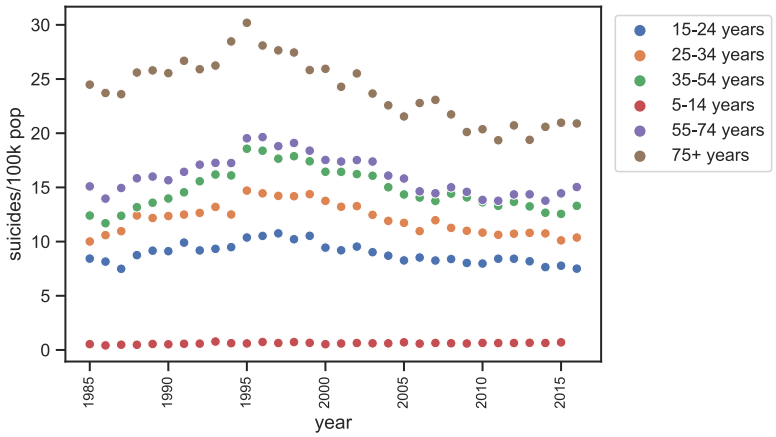

Figure 14: More Features

```
# Aggregate over generation and country
df4 = dfOrg.copy()
df4 = df4.groupby(['generation', 'country']).mean(numeric_only=True)
df4 = df4.reset_index()
print(len(df4))
```

> Output:
>
> 590

```
# Sanity
df4.to_markdown('table0308.md')
```

	generation	country	year	suicides_no	population	suicides/100k pop	HDI for year	gdp_for_year ($)	gdp_per_capita ($)
0	Boomers	Albania	1998	12.0208	344447	3.37771	0.656667	4.44387e+09	1580.21
1	Boomers	Antigua and Barbuda	1998.58	0.112903	8430.13	1.23855	0.781667	7.67784e+08	10183.2
2	Boomers	Argentina	1998.82	278.75	3.45253e+06	7.96015	0.776111	2.64442e+11	7769.97
3	Boomers	Armenia	2001.14	11.8036	377677	3.33929	0.685714	4.71864e+09	1624.14
4	Boomers	Aruba	2003.85	1.5	15467	10.3362	nan	2.17534e+09	24086.8
5	Boomers	Australia	1998.64	355.515	2.23113e+06	16.0932	0.910714	5.79461e+11	30523.5
6	Boomers	Austria	1999.31	213.414	969525	22.3163	0.844	2.54406e+11	32927.4
7	Boomers	Azerbaijan	1996.6	16.5	802334	2.11975	0.6245	6.82207e+09	969.45

Finally, a big picture of the dataset shows the information for each country where the questions are, such as, should we use the population variable as it is or should we normalize it?

```
# More plots
plt.figure(figsize=(18, 7), dpi=72)
sns.scatterplot(x='country', y='suicides/100k pop', hue='generation', data=df4)
plt.xticks(rotation=90, fontsize=10)
plt.legend(bbox_to_anchor=(1.01, 1.0))
```

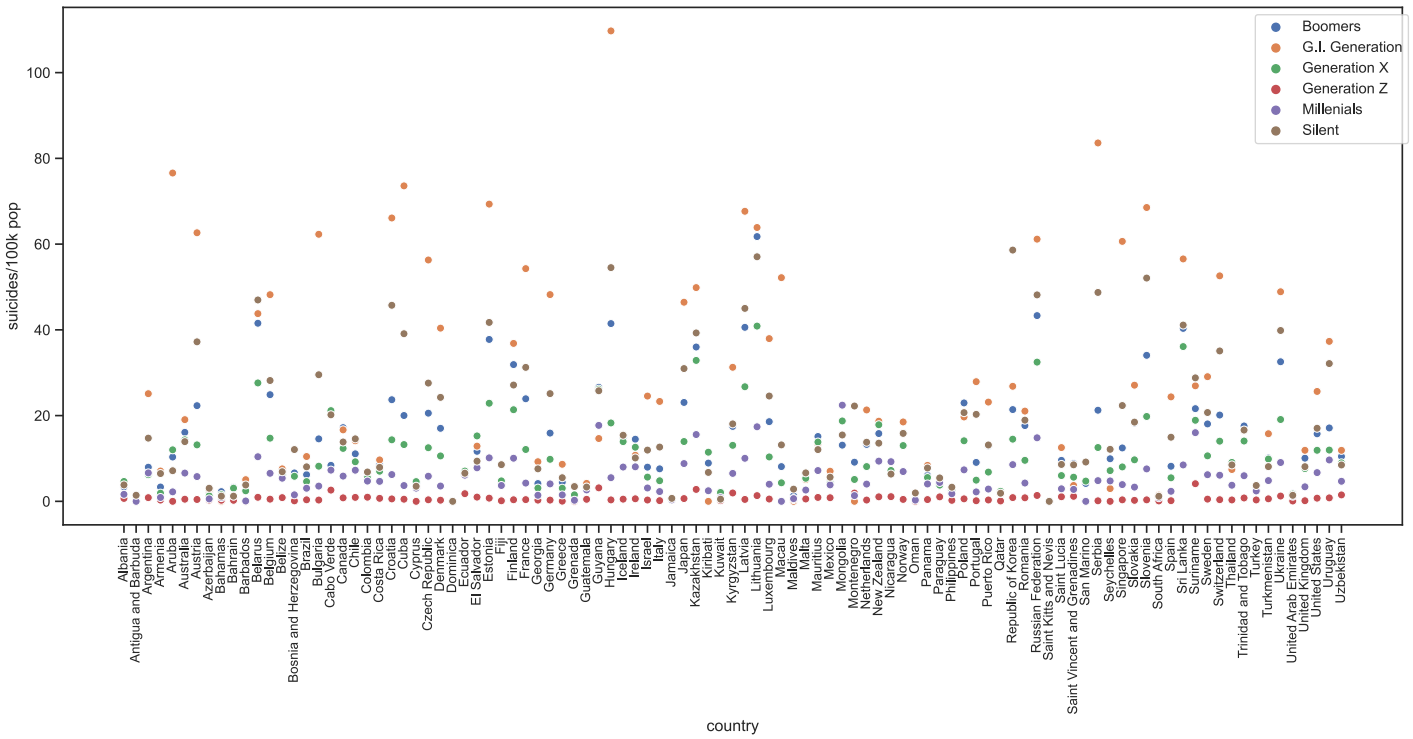

Figure 15: Features including country

10.1 Summary

Feature exploration is a fundamental step in the data preprocessing pipeline, ensuring that data scientists understand the structure, relationships, and distributions of features before applying machine learning models. This process involves inspecting data types, identifying relevant independent and dependent variables, and evaluating feature interactions with domain experts to ensure meaningful insights.

Throughout this section, we demonstrated how feature exploration can guide data preprocessing decisions. Key steps included examining nominal features using unique value counts, visualizing feature distributions through histograms, and normalizing dependent variables for comparability. Further, aggregating features such as age, sex, and generation provided deeper insights into trends and patterns in the dataset. Using visualization techniques, such as scatter plots and bar charts, facilitated understanding the relationships between different attributes, revealing potential correlations or discrepancies that may impact predictive modeling.

The case study using the Suicide Rates Overview (1985-2016) dataset illustrated various feature exploration techniques. We highlighted the importance of checking feature types, aggregating data for meaningful groupings, and using domain knowledge to guide transformations such as normalization. The dataset's visual representations demonstrated how GDP per capita, sex, generation, and suicide rates interact over time, reinforcing the necessity of careful feature engineering before modeling.

In summary, feature exploration lays the groundwork for effective data preprocessing and model development. By thoroughly investigating features, data scientists can identify inconsistencies,

detect biases, and determine necessary transformations to improve model performance. The next sections will delve into feature ranking, selection, and dimensionality reduction to optimize the dataset for machine learning tasks.

11 Feature Ranking and Selection

The major task in feature selection, given a large number of features, is to select the most important features and reduce the dimensionality while retaining class-discriminatory information. This procedure is essential when determining which features will be used to train the classification model. If features with little discrimination power are selected, the subsequent classification model will lead to poor classification performance. On the other hand, if information-rich features are selected, the design of the classifier can be greatly simplified. In a more quantitative description, feature selection leads to large between-class distances and small within-class distances in the feature space. Features should separate different classes by a large distance and have small distance values between objects in the same class. Several methods are available to identify individual features with linear separation, a few ranking and selection methods include divergence measure (Fukunaga, 1990; Theodoridis and Koutroumbas, 2006) [12] [32], Bhattacharyya distance (Bhattacharyya, 1943; Fukunaga, 1990) [25] [12] and Fishers linear discriminant ratio (Fisher, 1936; 1943; Dillon and Goldstein, 1984; Fukunaga, 1990; van der Heijden et al., 2004; Bishop, 1995, 2006; Theodoridis and Koutroumbas, 2006). [11] [9] [12] [16] [6] [5] [32]

Care must be taken when using feature ranking methods when measuring nonlinear class separability. Ranking methods developed for specific classifiers are often best suited for determining the best set of ranked features. For neural network classifiers, features are ranked and selected based on a saliency metric (Ruck et al., 1990; Belue and Bauer, 1995) [30] [3] and signal-to-noise ratio (Bauer et al., 2000). For kernel based classifiers, such as kernel Fishers discriminant and support vector machines, method-specific techniques are best suited for ranking. These techniques include recursive feature elimination (Guyon et al., 2002; Guyon, 2007) [14] [13], zero-norm feature ranking (Weston et al., 2003) [33], gradient calculations using recursive feature elimination (Rakotomamonjy, 2003) [26], and kernel Fishers discriminant using recursive feature elimination (Louw and Steel, 2006). [22]

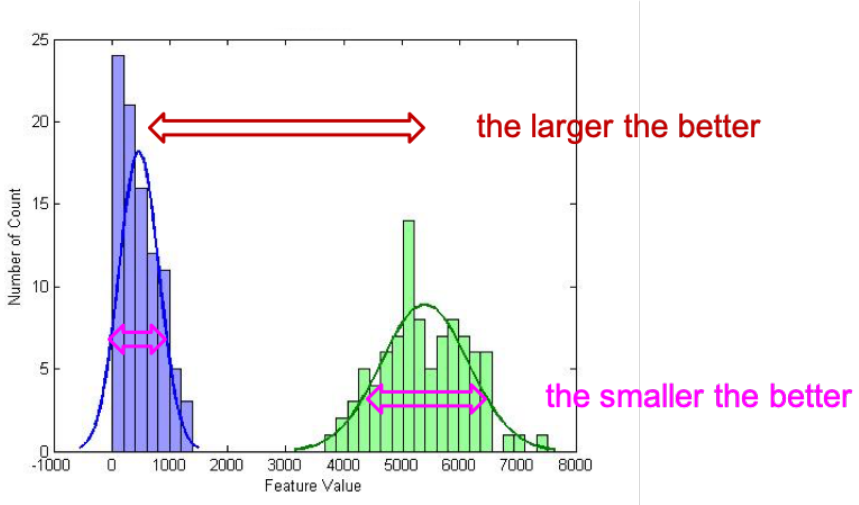

Figure 16: Feature Selection - ranking method based on between class means and within class variances.

11.1 Model for Feature Ranking and Selection

Steps for Feature Ranking and Selection:

- Features should be ranked in the vector space the classifier will use.
- Features should be ranked individually with a metric that gives a value on how well the feature classifies the observations.
- Once each feature is ranked, an iterative method should be used to select the first two features to identify the top two features well and classify them.
- Next, select the top three features to identify how well the features are classified.
- Continue this process until the classification accuracy falls or reaches a steady state.

Use the mean and standard deviation for the dataset with three classes and four features, and take the three mean and std values (μ_1, μ_2, μ_3, σ_1, σ_2, σ_3) for each class for a specific feature. You would then feed in observations 1 (x_1) and use something like the Mahalanobis distance to determine which class x_1 is closest to as follows:

- $d_1 = (x_1 - \mu_1)\sigma_1(x_1 - \mu_1)$
- $d_2 = (x_2 - \mu_2)\sigma_2(x_2 - \mu_2)$
- $d_3 = (x_3 - \mu_3)\sigma_3(x_3 - \mu_3)$

Now assign x_1 to the class with the smallest $d = [d_1, d_2, d_3]$ value. Continue this process for the remaining observations for feature 1. Compare how well the observations were classified to the original assigned classes. Save that value as the value for feature 1. Do this process for feature 2, feature 3, feature 4.

The feature with the largest classification is your top feature, the feature with the second highest classification is the second-ranked feature, and so on.

To select the top features, follow this process by taking the mean and covariance of the top two features, then calculating the Mahalanobis distance to determine how well the top two features classify (a classifier of your choice can be used). Continue this process for the top three features to determine how well the sets of features separate the data by class.

11.2 Bhattacharyya Distance

The Bhattacharyya distance is used as a class separability measure. For two-class normal distributions the Bhattacharyya distance is defined as:

$$B = \frac{1}{8}(\mu_{-1} - \mu_{+1})^T \left(\frac{\Sigma_{-1} + \Sigma_{+1}}{2} \right)^{-1} (\mu_{-1} - \mu_{+1}) + \frac{1}{2} ln \left(\frac{||\frac{\Sigma_{-1}+\Sigma_{+1}}{2}||}{\sqrt{||\Sigma_{-1}||||\Sigma_{+1}||}} \right) \tag{7}$$

where $|\dot{|}$ denotes the determinant of the respective matrix. The Bhattacharyya distance corresponds to the optimum Chernoff bound when $\Sigma_{-1} = \Sigma_{+1}$. It is readily seen that in this case the Bhattacharya distance becomes proportional to the Mahalanobis distance between the means. It should be noted that the Bhattacharya distance consists of two terms. The first term gives the class separability due to the mean difference and disappears when $\mu_{-1} = \mu_{+1}$. The second

term gives the class separability due to the covariance difference and disappears when $\Sigma_{-1} = \Sigma_{+1}$ (Fukunaga, 1990). [12]

The Bhattacharyya distance for the multi-class case is represented as:

$$B_{ij} = \frac{1}{8}(\mu_i - \mu_j)^T \left(\frac{\sigma_i + \sigma_j}{2}\right)^{-1} (\mu_i - \mu_j) + \frac{1}{2} ln\left(\frac{\sigma_i^2 + \sigma_j^2}{2\sigma_i \sigma_j}\right), \qquad for \quad i \neq j \tag{8}$$

where $i, j \in \mathbb{Z}$ in this case corresponding to the classes $C = C_j = [C_1, C_2, , C_c], j = 1, 2, , c$. In this case for each feature an individual class is compared to the remaining classes based on distance. The features are assigned a ranking value according to the greatest distance between classes.

11.3 Fishers Linear Discriminant Ratio (FDR/F-Score)

The FDR is used to quantify the separability capabilities of individual features (Fisher, 1936) [11]. FDR is a simple technique which measures the discrimination of sets of real numbers. The within-class scatter matrix is defined as

$$S_W = \sum_C P_C S_C \tag{9}$$

where S_c is the covariance matrix for class $\mathbf{C} \in -1, +1$

$$S_W = \sum_{\substack{i=1, \\ i \in C}}^{l_C} (\mathbf{x} - \mu_C)(\mathbf{x} - \mu_C)^T \tag{10}$$

and P_C is the *a priori* probability of class $\mathbf{C}$. That is, $P_C \approx \lambda_C / \lambda$, where λ_C is the number of samples in class $\mathbf{C}$, out of a total of λ samples. The between-class scatter matrix is defined as

$$S_B = \sum_{\mathbf{C}} (\mu - \mu_C)(\mu - \mu_C)^T \tag{11}$$

where μ is the global mean vector

$$\mu = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \tag{12}$$

and the class mean vector μ_C is defined as

$$\mu_C = \frac{1}{n_C} \sum_{\substack{i=1, \\ i \in C}}^{n_C} \mathbf{x}_i \tag{13}$$

These criteria take a special form in the one-dimensional, two-class problem. In this case, it is easy to see that for equiprobable classes $|S_W|$ is proportional to $\sigma_i^2 + \sigma_j^2$ and $|S_B|$ proportional to $(\mu_{-1} - \mu_{+1})^2$. Combining SB and SW, the Fishers Discriminant ratio results in the following equation

$$\boldsymbol{FDR} = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \tag{14}$$

FDR is sometimes used to quantify the separability capabilities of individual features. For the multi-class case, averaging forms of FDR can be used. One possibility is

$$\boldsymbol{FDR} = \sum_{i}^{M} \sum_{j \neq i}^{M} \frac{(\mu_i - \mu_j)^2}{\sigma_i^2 + \sigma_j^2} \tag{15}$$

where the subscripts i, j refer to the mean and variance corresponding to the feature under investigation for the classes C_i, C_j, respectively. For the one-dimensional multi-class case, the Fishers discriminant ratio is modified as:

$$FDR_{ij} = \frac{(\mu_i - \mu_j)^2}{\sigma_i^2 + \sigma_j^2} \tag{16}$$

Example: Ranking using Python `sklearn SelectPercentile`

Let's try to answer the question: Can we drop `age`, `menopause`, `breast`, `breast-quad` variables and redo the classification evaluation without a performance loss?

```
from sklearn.feature_selection import SelectPercentile, f_classif

X = df.loc[:, df.columns != 'recurrence'].values
y = df.loc[:, df.columns == 'recurrence'].values.ravel()

selector = SelectPercentile(f_classif, percentile=10)
# Fit the data
selector.fit(X, y)
scores = -np.log10(selector.pvalues_)
scores /= scores.max()

# Display
plt.figure(figsize=(6, 4), dpi=72)
cols = list(df.loc[:, df.columns != 'recurrence'].columns.values)
y_pos = np.arange(len(cols))
plt.bar(y_pos, scores)
plt.xticks(y_pos, cols, rotation=90)
```

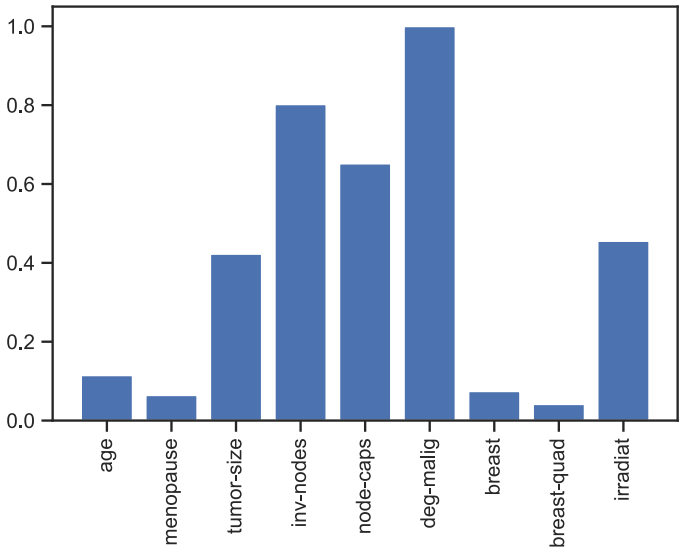

Figure 17: Features Rankings of Breast Cancer Dataset

According to the ranking, the variable `deg-malig` will perform best for a univariate model (i.e., a single variable data model).

11.4 Summary

Feature ranking and selection play a crucial role in optimizing machine learning models by reducing dimensionality while preserving class-discriminatory information. The selection of relevant features significantly impacts classification performance, ensuring that models focus on the most informative attributes while discarding redundant or irrelevant data. By selecting features with high between-class variance and low within-class variance, classification tasks can be performed more efficiently with improved accuracy.

Various ranking and selection techniques have been explored in this section. Methods such as divergence measures, Bhattacharyya distance, and Fishers linear discriminant ratio (FDR) offer different approaches for measuring the separability of classes within a dataset. Additionally, feature selection techniques tailored to specific machine learning models, such as recursive feature elimination (RFE) for kernel-based classifiers and saliency metrics for neural networks, provide targeted ways to refine input features. The use of Mahalanobis distance as a classification metric further demonstrates the importance of statistical measures in feature evaluation.

The application of these techniques ensures that the model is trained on the most relevant attributes, reducing computational complexity and mitigating the risk of overfitting. The feature ranking experiment conducted on the Breast Cancer Dataset illustrated the impact of feature importance, highlighting the effectiveness of deg-malig as the most informative feature. Understanding and applying these ranking and selection methods enable data scientists to design more efficient machine learning models while improving classification performance.

The next sections will explore dimensionality reduction techniques, which complement feature selection by further refining high-dimensional datasets for improved learning efficiency.

12 Dimensionality Reduction

Dimensionality reduction is a critical process in machine learning, data analytics, and data science. It involves reducing the number of input variables or features in a dataset. This can lead to models that are easier to interpret and can generalize better from the training data to unseen data.

An approach to reducing the dimension of the input features is to use a transformed space instead of the original feature space. For example, using a transformation $f(x)$ that maps the data points $\mathbf{x}$ of the input space, $x[n]$, into a reduced dimensional space $x'[p]$, where $n > p$, creates features in a new space that may have better discriminatory properties. Classification is based on the new feature space rather than the input feature space. The advantage of feature extraction with the use of dimensionality reduction as described here over feature selection is that no information from any of the elements of the measurement vector is removed. In some situations feature extraction is easier than feature selection. A disadvantage of feature extraction is that it requires the determination of a suitable transformation $f(x)$. Some methods include principal component analysis (Hotelling, 1933; Dillon and Goldstein, 1984) [18] [9] and kernel principal component analysis (Scholkopf et al., 1998; Bishop, 2006) [31] [5]. If the transformation chosen is too complex, the ability to generalize from a small dataset will be poor. On the other hand, if the transformation chosen is too simple, it may constrain the decision boundaries to a form that is inappropriate to discriminate between classes. Another disadvantage is that all features are used, even if some of them have noise like characteristics. This might be unnecessarily expensive in term of computation (van der Heijden et al., 2004) [16]. It should be noted that the transformation used for the input features in the training of the classification model should also be used for the testing features.

What is the importance of dimensionality reduction? To prevent overfitting, and high-dimensional data can cause overfitting, where a model learns the noise in the training data instead of the true pattern. To reduce computational complexity, fewer data points can make algorithms train faster. To improve interpretability, fewer dimensions can make the data easier to comprehend and visualize. Reducing dimensions to 2 or 3 can enable visualization, which can help in understanding the data.

Dimensionality reduction is an invaluable asset in the arsenal of any individual dealing with data. It is especially essential in the current data science field, where data of high dimensionality is commonplace. Comprehending, executing, and instructing this concept necessitates a combination of mathematical comprehension, practical abilities, and analytical thinking about the underlying issue and data.

12.1 Principal Component Analysis (PCA) for Dimensionality Reduction

The idea of feature extraction using PCA (Hotelling, 1933) [18] is to represent a new space in a way to extract mutually uncorrelated features from the current space. The new features are known as the principal components after transform mapping. The dimensionality assessment is accomplished by extracting the principal components from the correlation matrix and retaining only the factors described in Kaisers criterion (eigenvalues: $\lambda \geq 1$) (Kaiser, 1960) [19]. The criterion is used as a guideline to determine the number of principal components to retain by calculating the correlation matrix of the input features. Each observed variable contributes one unit of variance to the total variance in the dataset. Hence, any principal component that has an eigenvalue, λ greater than one, accounts for a more significant amount of variance than had

been contributed by one variable. Additionally, a principal component that displays an eigenvalue less than one indicates less variance than had been contributed by one variable. The covariance matrix, $\mathbf{C}$, is used to extract eigenvectors, v, retaining only the number of principal components corresponding to Kaisers criterion.

The basic concept of feature extraction using PCA is to map $\mathbf{X}$ onto a new space capable of reducing the dimensionality of the input space. The data is partitioned by variance using a linear combination of original factors. To perform PCA on a set of training vectors from the M-dimensional input space $\mathbf{X}$. The set of vectors $\hat{\mathbf{X}}$ is a lower dimensional representation of the input training vectors $\mathbf{X}$ in the $\hat{M}$-dimensional space $\hat{\mathbf{X}}$, where $\hat{M} < M$. The vectors $\hat{\mathbf{M}}$ are obtained by the linear orthonormal projection.

$$\hat{\mathbf{X}} = (\mathbf{X} - \mu)\mathbf{A} \tag{17}$$

where $\mathbf{A}$ is an $[M \times \hat{M}]$ matrix containing the top $\hat{M}$ eigenvectors and μ is the mean of each set of features from $\mathbf{X}$.

PCA can be described in the following steps:

1. Input date $\mathbf{X}$ of size $N \times M$.
2. Standardize the dataset so that each variable has zero mean and unit variance.
3. Compute the covariance matrix of the standardized data. The covariance matrix $\mathbf{C}$ is a matrix $M \times M$ where each element c_{ij} represents the covariance between the variable i and the variable j.
4. Perform eigenvalue decomposition on $\mathbf{C}$ to obtain eigenvalues λ and eigenvectors v_i.
5. Select the top $\hat{M}$ eigenvectors based on their corresponding eigenvalues to form the matrix of principal components $\mathbf{A}$.
6. Transform the original data $\mathbf{X}$ using the matrix $\mathbf{A}$ to obtain the reduced dataset $\hat{\mathbf{X}}$ of dimensions $n \times k$, where $\hat{\mathbf{X}} = (\mathbf{X} - \mu)\mathbf{A}$.

It should be noted that standardization is crucial because PCA is affected by the scale of the variables. The number of components k is chosen based on the cumulative variance that one wishes to retain in the reduced dataset. The transformation $\hat{\mathbf{X}} = (\mathbf{X} - \mu)\mathbf{A}$ projects the original data onto the new subspace defined by the selected principal components.

To find the eigenvectors and eigenvalues of a dataset begin with the covariance.

$$Cov(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N}(x_n - \mu_{\mathbf{x}})(x_n - \mu_{\mathbf{x}})^T \tag{18}$$

The covariance matrix can be represented as

$$\Sigma = \begin{bmatrix} Cov(\mathbf{x}_1, \mathbf{x}_1) & Cov(\mathbf{x}_2, \mathbf{x}_1) & ... & Cov(\mathbf{x}_n, \mathbf{x}_1) \\ Cov(\mathbf{x}_1, \mathbf{x}_2) & Cov(\mathbf{x}_2, \mathbf{x}_2) & ... & Cov(\mathbf{x}_n, \mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(\mathbf{x}_1, \mathbf{x}_n) & Cov(\mathbf{x}_2, \mathbf{x}_n) & ... & Cov(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \tag{19}$$

where $Cov(\mathbf{x}_1, \mathbf{x}_1) = Var(\mathbf{x}_1)$, which extends across the diagonal of the matrix.

To find the eigenvectors and eigenvalues of Σ, use

$$(\Sigma - \lambda I)\vec{e} = \vec{0} \tag{20}$$

Since $\vec{e} = \vec{0}$ would not yield a useful eigenvector, a solution must be found for $\det(\Sigma - \lambda I) = \vec{0}$.

Once we have calculated all of the eigenvectors of the covariance matrix, we can begin reducing to a lower dimension. The first step of this is to find the eigenvalues that best capture the variance of the data. To find these eigenvectors, we can use each eigenvector's corresponding eigenvalue. This can be done by creating a list of the eigenvalues, Λ, in descending order.

$$\Lambda = [\lambda_1, ..., \lambda_d] \tag{21}$$

where d is the number of dimensions or features, and $\lambda_1 \geq \ldots \geq \lambda_{d-1} \geq \lambda_d$

Then, we can find the percentage each eigenvalue represents of the total, or the explained variance using

$$\text{Explained Variance} = 100\left(\frac{1}{\sum \lambda_n}\lambda_n\right) \tag{22}$$

where λ_n represents an individual eigenvalue.

The eigenvectors of the covariance matrix are normally called the principal components. The top few principal components can be used to reduce the dimensions of the dataset.

Determinant of Covariance Matrix

The covariance is the relationship of two random variables. A positive covariance means that the variables tend to vary in the same direction, while a negative covariance would vary in opposite directions. Very similar to the correlation of two variables, except the covariances are not bounded between -1 and 1. Covariances are often organized into a matrix in which the determinant can be calculated. The determinant is a scalar value that indicative of how a matrix behaves and its properties.

Numbers within the matrix are denoted by $|C_{ij}|$ with the first as the row, second the column, e.g., $|C_{2,2}|$ would be a covariance of size $[2 \times 2]$.

The determinant of a 2x2 matrix is given by

$$\det(C) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = (a * d) - (b * c)$$

However, for matrices of n x n size, can be determined with the Leibniz formula as follows:

$$\det(A) = \sum_{\sigma \in \mathbf{S}_n} (sgn(\sigma) \prod_{i=1}^{n} a_{i\sigma_i})$$

$\prod$ is the product of sequence of products within the indexes. Sgn or sign (σ) is the number of permutations required for the matrix to return to the column order from least to greatest. If the number of permutation is even, then the sign of the product is positive. If the number of permutations is odd, then the sign of the product is negative.

Example: Mathematical Calculation of Eigenvectors for the Iris Dataset

The Iris dataset is a dataset with 150 observations and three different species. Each observation includes the factors sepal length $(x_{1,1}, x_{2,1}, \ldots, x_{10,1})$, sepal width $(x_{1,2}, x_{2,2}, \ldots, x_{10,2})$, petal length $(x_{1,3}, x_{2,3}, \ldots, x_{10,3})$, and petal width $(x_{1,4}, x_{2,4}, \ldots, x_{10,4})$. For now, we will calculate the eigenvectors and eigenvalues of the first 10 observations of the setosa species.

$$\begin{aligned}
\mathbf{x}_1 &= [5.1, 3.5, 1.4, 0.2] \\
\mathbf{x}_2 &= [4.9, 3.0, 1.4, 0.2] \\
\mathbf{x}_3 &= [4.7, 3.2, 1.3, 0.2] \\
\mathbf{x}_4 &= [4.6, 3.1, 1.5, 0.2] \\
\mathbf{x}_5 &= [5.0, 3.6, 1.4, 0.2] \\
\mathbf{x}_6 &= [5.4, 3.9, 1.7, 0.4] \\
\mathbf{x}_7 &= [4.6, 3.4, 1.4, 0.3] \\
\mathbf{x}_8 &= [5.0, 3.4, 1.5, 0.2] \\
\mathbf{x}_9 &= [4.4, 2.9, 1.4, 0.2] \\
\mathbf{x}_{10} &= [4.9, 3.1, 1.5, 0.1]
\end{aligned}$$

where each $\mathbf{x}_n$ is an individual observation and the features are represented as follows:

$$
\begin{aligned}
F_{sl} &= [5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9] \\
F_{sw} &= [3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1] \\
F_{pl} &= [1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5] \\
F_{pw} &= [0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1]
\end{aligned}
$$

First, calculate the mean by feature which results in $\boldsymbol{\mu} = [\mu_{sl}, \mu_{sw}, \mu_{pl}, \mu_{pw}]$ where $\mu_{sl} = 4.86$, $\mu_{sw} = 3.31$, $\mu_{pl} = 1.45$, and $\mu_{pw} = 0.22$.

Then, calculate the covariance matrix

$$
\Sigma = \begin{bmatrix}
0.0764 & 0.0634 & 0.0170 & 0.0128 \\
0.0634 & 0.0849 & 0.0155 & 0.0158 \\
0.0170 & 0.0155 & 0.0105 & 0.0020 \\
0.0128 & 0.0158 & 0.0020 & 0.0056
\end{bmatrix}
$$

The covariance matrix can then be used to calculate the eigenvectors and eigenvalues

$$
0.1509 \begin{bmatrix} -0.6679 \\ -0.7131 \\ -0.1616 \\ -0.1386 \end{bmatrix}, 0.0176 \begin{bmatrix} -0.7048 \\ 0.6782 \\ -0.1756 \\ 0.1118 \end{bmatrix}, 0.0067 \begin{bmatrix} 0.2272 \\ -0.0484 \\ -0.9400 \\ 0.2500 \end{bmatrix}, 0.0022 \begin{bmatrix} -0.0742 \\ -0.1709 \\ 0.2440 \\ 0.9517 \end{bmatrix}
$$

In this case (because of the use of the NumPy module) the eigenvalues, Λ, are already sorted in descending order.

$$
\Lambda = [0.1509, 0.0176, 0.0067, 0.0022]
$$

With these eigenvalues, the explained variance can be calculated

$$
\textbf{Explained Variance} = [85.0416, 9.9217, 3.7522, 1.2845]
$$

Given the first 10 values of each of the setosa features in the iris dataset, we can capture 85.0416% of the data using one principal component.

Python Code

```
import numpy as np
from numpy import linalg as la

class Eigen:
    def makeList(data, row):
        npList = data[row, :]
        return npList.tolist()

    def meanCalc(data, row):
        """ Find the mean of a dataset

        Keyword arguments:
        data -- the 2d numpy array
        row -- feature of data to calculate mean on

        """
        dataList = Eigen.makeList(data, row)

        sum = 0
        for item in dataList:
            sum += item
        n = len(dataList)
        mean = sum/n
        return mean

    def covarianceCalc(data, row1, row2):
        """ Find the covariance matrix of a 2d numpy array

        Keyword arguments:
        data -- the 2d numpy array
        row1, row2 -- the features of the dataset to calculate covariance on

        """

        dataList1 = Eigen.makeList(data, row1)
        dataList2 = Eigen.makeList(data, row2)

        mean1 = Eigen.meanCalc(data, row1)
        mean2 = Eigen.meanCalc(data, row2)

        sum = 0
        for i, item in enumerate(dataList1):
            zeroMean1 = dataList1[i] - mean1
            zeroMean2 = dataList2[i] - mean2
```

```
            sum += zeroMean1 * zeroMean2

        # population covariance, N, not N-1
        n = len(dataList1)

        covariance = sum/n

        return covariance

    def covarianceMatrix(data):
        """ Find the covariance matrix of a 2d numpy array

        Keyword arguments:
        data -- the 2d numpy array

        """

        numRows = np.shape(data)[0] #

        covarianceMatrix = np.empty(shape = [numRows, numRows])

        for i, row in enumerate(covarianceMatrix):
            for j, col in enumerate(row):
                covarianceMatrix[i][j] = Eigen.covarianceCalc(data, i, j)

        return covarianceMatrix

    def explainedVariance(eigenvalues):
        """ Find the Explained Variance of a list of eigenvalues

        Keyword arguments:
        eigenvalues -- the values for the Explained Variance to be calculated
        ↪  on

        """

        sum = 0
        for value in eigenvalues:
            sum += value

        explainedVariance = []

        for value in eigenvalues:
            explainedVariance.append((value/sum) * 100)
```

```
        return np.array(explainedVariance)


    def combined(data):
        """ Find the Explained Variance of a dataset

        Keyword arguments:
        data -- the data for the Explained Variance can be calculated on
        """

        covarianceMatrix = Eigen.covarianceMatrix(data)
        w, v = la.eig(covarianceMatrix)
        explainedVariance = Eigen.explainedVariance(w)
        return explainedVariance
```

Steps to do PCA

- Zero mean the data
- Take the covariance of the zero mean-ed data
- Calculate the Eigenvectors and Eigenvalues
- Sort the Eigenvalues and Eigenvectors
  - ◇ After sorting the Eigenvalues and Eignevectors
  - ◇ Sum the Eigenvalues $\Lambda_{sum} = \sum \Lambda$
  - ◇ Now take Λ and divide each of the values by Λ_{sum}
  - ◇ $\Lambda_{total} = \frac{\Lambda}{\Lambda_{sum}}$
  - ◇ $\sum \Lambda_{total} \times 100 = 100$
  - ◇ This will then lead to the variance explained based on percentage, for example, $\Lambda_{total} = [0.750.150.0750.025]$
  - ◇ So 90% of the variance are the top two Eigenvalues
- Calculate the variance explained from based on the Eigenvalues

This now leads to the next section!

12.2 Number of Components to Retain (Dillon and Goldstein, 1984)

Principal component analysis is frequently employed for the purpose of generating a reduces set of variates that account for most of the variability in the original data, and that can be used in more substantial subsequent analysis. We must therefore decide just how many components to retain. Unfortunately there is not universally accepted method for doing so. The decision is largely judgmental and a matter of taste.

A number of procedures for determining how many components to retain have been suggested. These "rules" range from methods that evoke formal significance tests to less formal approaches involving heuristic graphical arguments. The remainder of this section discusses several of the more commonly used procedures.

Variance - Covariance Input

A reasonable approach for determining the number of components to retain when factoring a variance-covariance matrix relies on the sampling distribution results. For example, we might suggest that only those components whose associated eigenvalues are statistically different from zero be retained.

The reader should note, however, that with even moderate sample sizes many of the components will typically be statistically significant. Yet, from a practical viewpoint, some of these significant components account for only a vary small proportion of the total variance. Hence, for reasons of parsimony and practical significance, the statistical criteria should be interpreted loosely with fewer components retained than are statistically significant. In this regard,, some of the graphical procedures soon to be discussed may prove useful.

An alternative and somewhat more ad hoc approach is the percentage of variance criterion. With this approach, the cumulative percentage of eh variance extracted by successive components is the criterion. Note, the cumulative proportion of total variance extracted by a set of components sis given by

$$\frac{\sum_{j=}^{m} l_{(j)}}{\sum_{j=}^{p} l_{(j)}} \tag{23}$$

where $m < p$. The stopping rule is subjective, since the number of components retained is based on some arbitrary determined criterion for the amount of variation accounted for.

Correlation Input

When factoring a correlation matrix, statistical testing procedures no longer apply, and the retained variance criterion lacks clear meaning. For these reasons, various graphical heuristics have been suggested as a rule of thumb.

Perhaps the most frequent used extraction approach is the "root greater than one" criterion. Originally suggested by Kaiser (1958) [20], this criterion retains those components whose eigenvalues are greater than one. The dimensionality assessment is accomplished by extracting the principal components from the correlation matrix and retaining only the factors described in Kaiser's criterion (eigenvalues: $\lambda \geq 1$). Principal component analysis partitions the data by variance using linear combination of original factors. The rationale for this criterion is that any component should account for more "variance" than any single variable in the standardized test score space.

Another approach, proposed by Cattell (1966) [7], is called the scree test. With this approach, named after the rubble at the bottom of a cliff, the eigenvalues of each component are plotted in successive order of their extraction, and then an elbow in the curve is identified by applying, say, a straightedge to the bottom portion of the eigenvalues to see where they form an approximate straight line. The number of components retained is given by the point at which the components curve above the straight line formed by the smaller eigenvalues. Cattell and Jaspers (1967) [8] suggested that the number of factors be taken as the number immediately before the straight line begins. The rationale for the scree test is simple: since the principal component solution extracts components in successive order of magnitude, the substantive factors appear first, followed by the numerous trivial components which account for only a small proportion of the total variance.

Though this approach is relatively simple, complications can surface. First, there may be no obvious break, in which case the test is inclusive. Second, There may be several breaks. This is particularly troublesome when, say, two breaks occur among the first half of the eigenvalues, since it will be difficult to decide which of the breaks reflects the correct number of components.

Horn (1965) [17] has suggested yet another approach for determining the number of components to retain which removes much of the ambiguity associated with the scree test. As in the scree test, the data are factored asn te resulting component eigenvalues are plotted in sucessinve order of their magnitude. Next, K setts of $n \times p$ normally and independently distributed (NID) random variates are sampled from a population whose correlation structure is known to be characterized by an identity matrix. Each $n \times p$ data matrix is factored. The average eigenvalue for each extracted component over the k sets of randomly generated data s then computed and plotted in the same graph with the real data. Because of samling variation, several of the extracted eigenvvalues may exceed unity. However, the average curve of eighenvalues can be expressed to cross the ordinate scale att the value 1.0 for component number $p/2$. The proincipal component solution for the simulatioed data represents the case in which the eigenvalues under the null hypothesis are unity. Consequently, Horn's criterion is to retain components n the basis of where the reference curve crosses the curve induced from the actual data.

Great Online Resource: http://sebastianraschka.com/Articles/2014_pca_step_by_step.html
https://blogs.sas.com/content/iml/2017/08/02/retain-principal-components.html

Example: Iris dataset

Data reduction can also help in the visualization of clusters, which are, in fact, can be considered as classes without labels.

Let's take Iris dataset and reduce the number of dimensions from 4 to 2. Then examine features on a plot to see clusters are visible.

```
from sklearn.datasets import load_iris
# locate and load the data file
iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=list(iris.feature_names))
iris_df['species'] = [iris.target_names[_] for _ in iris.target]
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
6	4.6	3.4	1.4	0.3	setosa
64	5.6	2.9	3.6	1.3	versicolor
116	6.5	3	5.5	1.8	virginica
115	6.4	3.2	5.3	2.3	virginica
19	5.1	3.8	1.5	0.3	setosa

```
from sklearn.preprocessing import StandardScaler

X = iris_df.loc[:, iris_df.columns != 'species'].values
y = iris_df.loc[:, iris_df.columns == 'species'].values.ravel()  # string at the
 ↪  moment
scaled_X = StandardScaler().fit_transform(X)

from sklearn.decomposition import PCA

# Visualize the models above in 2D
Xpca = PCA(n_components=2).fit_transform(scaled_X)

plt.figure(figsize=(8, 5), dpi=72)
CMAP = {'setosa':'skyblue', 'versicolor':'orangered', 'virginica':'seagreen'}
for _ in CMAP.keys():
    plt.scatter(Xpca[np.where(y==_),0], Xpca[np.where(y==_),1], c=CMAP[_],
     ↪  label=_)
plt.xlabel('Dimension 0')
plt.ylabel('Dimension 1')
plt.legend()
```

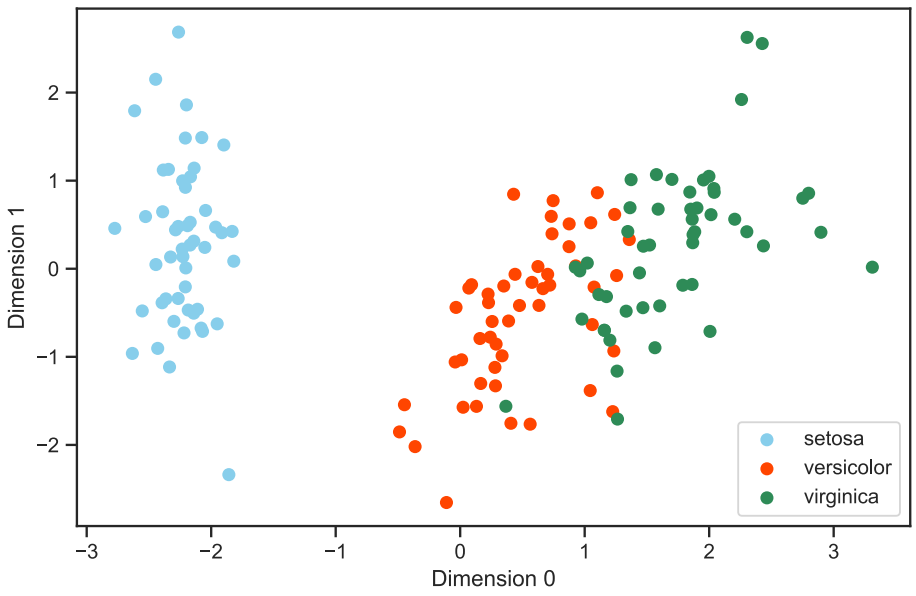

Figure 18: Iris Dataset in two dimensions

As can be seen from the figure, though PCA is a lossy transformation, the separation of three classes is preserved to some extent.

12.3 Summary

Dimensionality reduction is a fundamental technique in machine learning and data analytics, aimed at reducing the number of features while preserving essential information. This process improves model efficiency, enhances interpretability, and mitigates issues related to overfitting, particularly in high-dimensional datasets. By transforming data into a lower-dimensional space, models can generalize better to unseen data while reducing computational complexity.

Principal Component Analysis (PCA) is a widely used technique for dimensionality reduction. It identifies new feature axes (principal components) that maximize variance, allowing data representation with fewer dimensions while retaining the most informative aspects. The transformation process involves standardizing data, computing the covariance matrix, performing eigenvalue decomposition, and selecting the principal components based on explained variance criteria. The application of PCA on datasets, such as the Iris dataset, demonstrates its ability to reveal patterns and structure while maintaining class separability.

Choosing the appropriate number of components is crucial, with methods like Kaisers criterion, scree plots, and explained variance ratios guiding the selection process. While dimensionality reduction helps improve performance, it is important to balance transformation complexity and information retention.

The next section will further explore techniques that refine dataset structures and optimize model performance, building on the concepts introduced in dimensionality reduction.

13 Data Splitting

Data splitting is a key procedure in machine learning and data analytics, particularly when assessing model performance. It involves separating the dataset into distinct parts to guarantee that the model is trained, validated, and tested properly. Splitting data has the benefit of making sure the model is tested on unseen data, decreasing the danger of memorizing the training data. This helps in evaluating how well the model applies to new, unseen data. Utilizing a distinct validation set helps in hyperparameter tuning without influencing the final assessment.

13.1 Common Methods of Data Splitting

Data splitting is a crucial step in machine learning to ensure that models generalize well to unseen data and do not overfit to the training set. By dividing the dataset into distinct subsets, different methods allow for effective model training, hyperparameter tuning, and evaluation. The simplest approach is the train-test split, where the data is divided into training and testing sets, typically using a 70%-80% to 20%-30% ratio. A more refined approach, the train-validation-test split, introduces a validation set to fine-tune hyperparameters before final model assessment. For more robust performance evaluation, K-fold cross-validation partitions the dataset into multiple folds, iteratively training and validating on different subsets to minimize variance in performance metrics. In cases where datasets are small, Leave-One-Out Cross-Validation (LOOCV) can be used, training the model on all but one data point and iterating through the entire dataset, though this method is computationally expensive. The choice of a data splitting technique depends on the dataset size, computational constraints, and the need for precise model evaluation.

Train-Test Split

- Training Set: A subset of the data used to train the model.
- Testing Set: A subset to evaluate the model's final performance.
- Typical Split: Often, 70%-80% for training and 20%-30% for testing.

Train-Validation-Test Split

- Training Set: Used to train the model.
- Validation Set: Used to tune hyperparameters and select the best model.
- Testing Set: Used to assess the final model performance.
- Typical Split: A common split might be 60% training, 20% validation, and 20% testing.

K-Fold Cross-Validation

- Divides the dataset into 'K' equal folds.
- Uses 'K-1' folds for training and the remaining fold for validation.
- Repeats the process 'K' times, using a different fold for validation each time.
- Averages the results to obtain a more robust model evaluation.

Leave-one-out-cross-validation (LOOCV)

- Use all but a single data point for training.
- Use the single data point for testing.
- Repeat the procedure for 'N' times.
- Seldom used but might be necessary for very small and expensive datasets such as medical studies, etc.

Stratification: When target categories are unbalanced (e.g., there are more benign cases rather than malignant), then a correct validation requires the training/test datasets, or folds, preserving the percentage of samples for each class. `sklearn` library supports stratification fully, such as by the use of `StratifiedKFold` library function.

Data splitting is a fundamental step in building robust and generalizable models. It plays a vital role in model evaluation, tuning, and selection. Understanding and properly implementing data splitting techniques is crucial for anyone working with machine learning or statistical modeling.

13.2 K-Fold Cross Validation

K-fold cross-validation is a common data-splitting method used to evaluate the generalization capability of machine learning models on sets of data. 5-fold cross-validation is used to assess the prediction capability of classifiers. In a 5-fold approach, each of five iterations is an 80%/20% train/test split, in which no data point is used more than $k-1$ times as part of the training set (Table 3). Additionally, the test set is nonoverlapping at any of the iterative experiments. This allows the classifier's performance to be assessed robustly on unseen data. If the test accuracy of the classifier is sufficiently high for each fold, the classifier can be considered to have acceptable generalization capability (i.e., the model has learned the general pattern leading to successful predictions on unseen data and has not simply *memorized* the training data).

	Fold 1	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**
Experiment 1	Train	Train	Train	Train	**Test**
Experiment 2	Train	Train	Train	**Test**	Train
Experiment 3	Train	Train	**Test**	Train	Train
Experiment 4	Train	**Test**	Train	Train	Train
Experiment 5	**Test**	Train	Train	Train	Train

Table 3: 5-Fold Cross-Validation

13.3 Summary

Data splitting is an essential step in machine learning and statistical modeling, ensuring that models generalize well to unseen data and do not overfit to the training set. Properly partitioning a dataset into training, validation, and test subsets allows for effective model training, hyperparameter tuning, and evaluation.

Several data splitting methods exist, each serving different purposes. The train-test split is the simplest approach, allocating a portion of the dataset for training and another for testing, typically in a 70%-80% to 20%-30% ratio. The train-validation-test split introduces a validation set for

hyperparameter tuning, improving model selection. K-fold cross-validation provides a robust evaluation strategy by partitioning the dataset into multiple folds, training and validating iteratively on different subsets, reducing performance variability. Leave-One-Out Cross-Validation (LOOCV) is particularly useful for small datasets, where each instance serves as a test case once, but it is computationally expensive.

Stratification ensures that imbalanced datasets maintain class distributions in training and testing, improving model evaluation accuracy. K-fold cross-validation further strengthens generalization by using multiple training and test partitions, as demonstrated in the 5-fold cross-validation process.

Understanding and implementing appropriate data splitting strategies is crucial for building reliable machine learning models. The next section will explore additional techniques to enhance model performance and evaluation.

14 Summary

This module provided a comprehensive overview of the key concepts and methodologies involved in data processing and preprocessing, which are critical components of data science and machine learning workflows. The discussion began with an introduction to data processing, highlighting the distinctions between data processing and data preprocessing. This foundational understanding set the stage for exploring more specialized topics.

Key topics covered in this module included data mining, where techniques for extracting insights, patterns, and correlations from large datasets were examined. The data collection process was discussed, emphasizing the importance of acquiring high-quality and relevant datasets from various sources. Following this, data curation was introduced as a crucial step in organizing, validating, and maintaining data integrity.

The module then delved into data cleaning, which involves handling missing values, correcting data inconsistencies, and ensuring data quality. Data transformation techniques, including normalization, standardization, and discretization, were explored to prepare data for effective analysis. The role of outliers and strategies for detecting and handling them were also covered in detail.

Feature engineering was another critical topic, covering feature exploration, ranking, selection, and dimensionality reduction. These techniques enable the selection of relevant variables and improve model efficiency. Data splitting methods, such as train-test splits and cross-validation, were also discussed to ensure robust model evaluation.

Overall, this module provided a structured approach to data preprocessing, integrating theoretical concepts with hands-on applications. The methodologies discussed serve as foundational steps for data-driven decision-making, ensuring that raw data is refined into meaningful insights before applying machine learning models. The subsequent sections will build upon these preprocessing techniques, demonstrating their real-world applications in various data science projects.

References

[1] Vic Barnett and Toby Lewis. *Outliers in Statistical Data*. 3rd. Academic Press, 1994. ISBN: 0471930946.

[2] Riccardo Bellazzi and Blaz Zupan. "Predictive Data Mining in Clinical Medicine: Current Issues and Guidelines". In: *International Journal of Medical Informatics* 77.2 (2008), pp. 81–97.

[3] Lisa M. Belue and Kenneth W. Bauer Jr. "Determining input features for multilayer perceptrons". In: *Neurocomputing* (1995).

[4] Monowar H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. "Network Anomaly Detection: Methods, Systems and Tools". In: *IEEE Communications Surveys & Tutorials* 16.1 (2014), pp. 303–336.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. URL: https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

[6] Cristopher Bishop. *Neural Networks for Pattern Recognition*. 1st. Oxford University Press, 1996. ISBN: 9780198538646.

[7] Raymond B. Cattell. "The Scree Test For The Number Of Factors". In: *Multivariate Behavioral Research* (1966).

[8] Raymond B. Cattell and Joseph Jaspers. "A general plasmode (No. 30-10-5-2) for factor analytic exercises and research." In: *Multivariate Behavioral Research Monographs* (1967).

[9] W. R. Dillon and M. Goldstein. *Multivariate Analysis Method and Applications*. New York,NY: John Wiley & Sons,Inc, 1984.

[10] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "From Data Mining to Knowledge Discovery in Databases". In: *AI Magazine* 17.3 (1996), pp. 37–54.

[11] R.A. Fisher. "The use of Multiple Measurements in Taxonomic Problems". In: *Proceedings of Annals of Eugenics* 7 (1936), pp. 179–188.

[12] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition*. 2nd. Academic Press, 1990. ISBN: 0122698517.

[13] Isabelle Guyon, Constantin Aliferis, and AndréElissee. *Computational Methods of Feature Selection*. 1st. Chapman and Hall, 2007. ISBN: 9780429150418.

[14] Isabelle Guyon and Andre Elisseeff. "An Introduction to Variable and Feature Selection". In: *Journal of Machine Learning Research* (2003).

[15] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. 3rd ed. Morgan Kaufmann, 2011.

[16] Hans van der Heijden. "User Acceptance of Hedonic Information Systems". In: *MIS Quarterly* (2004).

[17] J. L. Horn. "A rationale and test for the number of factors in factor analysis." In: *Psychometrika* (1965).

[18] H. Hotelling. "Analysis of a complex of statistical variables into principal components". In: *Journal of Educational Psychology* 24 (1933), pp. 417–441.

[19] Henry F. Kaiser. "The Application of Electronic Computers to Factor Analysis". In: *Educational and Psychological Measurement* 20.1 (1960), pp. 141–151. DOI: 10.1177/001316446002000116. eprint: https://doi.org/10.1177/001316446002000116. URL: https://doi.org/10.1177/001316446002000116.

[20] Henry F. Kaiser. "The varimax criterion for analytic rotation in factor analysis". In: *Psychometrika* (1958).

[21] Bing Liu, Kevin J. Loughlin, and Sharon Guo. "Personalization in E-Commerce: A Data Mining Perspective". In: *ACM Transactions on Internet Technology* 12.1 (2012), 3:1–3:29.

[22] N. Louw and S. Steel. "Variable selection in kernel Fisher discriminant analysis by means of recursive feature elimination". In: *Computational Statistics & Data Analysis* (2006).

[23] Prasanta Chandra Mahalanobis. "On the generalized distance in statistics". In: *Proceedings of the National Institute of Sciences of India* 2 (1 1936), pp. 49–55.

[24] E.W.T. Ngai, Li Xiu, and Dorothy C.K. Chau. "Application of Data Mining Techniques in Fraud Detection". In: *Expert Systems with Applications* 38.10 (2011), pp. 13057–13062.

[25] *On a measure of divergence between two statistical populations defined by their probability distribution*. 1943.

[26] Alain Rakotomamonjy. "Variable Selection Using SVM-based Criteria". In: *Journal of Machine Learning Research 3* (2003).

[27] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning,3rd Ed.* 3rd ed. Birmingham,UK: Packt Publishing, 2019. ISBN: 978-1789955750.

[28] Casey J. Richards et al. "Multimodal data fusion using signal/image processing methods for multi-class machine learning". In: *Signal Processing,Sensor/Information Fusion,and Target Recognition XXXII*. Ed. by Ivan Kadar, Erik P. Blasch, and Lynne L. Grewe. Vol. 12547. International Society for Optics and Photonics. SPIE, 2023, 125470N. DOI: 10.1117/12.2664987. URL: https://doi.org/10.1117/12.2664987.

[29] Benjamin M. Rodriguez. "Multi-Class Classification for Identifying JPEG Steganography Embedding Methods". PhD thesis. Air Force Institute of Technology, 2008. URL: https://scholar.afit.edu/etd/2641/.

[30] Dennis W. Ruck et al. "The multilayer perceptron as an approximation to a Bayes optimal discriminant function". In: *IEEE Transactions on Neural Networks* (1990).

[31] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Muüller. "Nonlinear Component Analysis as a Kernel Eigenvalue Problem". In: (1996).

[32] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. 3rd. Academic Press, 2006. ISBN: 0123695317.

[33] Jason Weston et al. "Use of the Zero-Norm with Linear Models and Kernel Methods". In: *Journal of Machine Learning Research 3* (2003).

[34] Samuel S. Wilks. *Mathematical Statistics*. New York & London: John Wiley & Sons,Inc., 1963.