databricks

Data Preparation and Feature Engineering

LECTURE

Fundamentals of Data Preparation and Feature Engineering

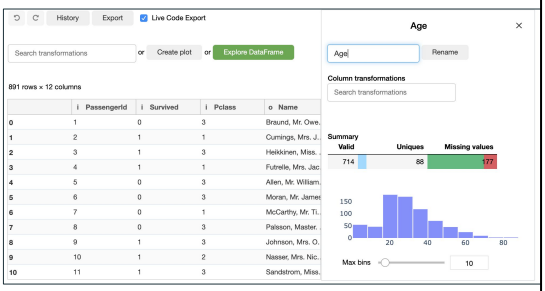
In this lecture, we cover the fundamentals of data preparation and feature engineering, including data preparation steps, feature extraction and selection, data splitting, and sampling methods for machine learning projects.

Data Preparation for ML projects

Goal: Optimize input quality for accurate model predictions

Data preparation includes the following tasks:

- **Cleaning and formatting data:** This includes tasks such as **handling missing values** or outliers, ensuring data is in the correct format, and removing unneeded columns.
- **Feature Engineering:** This includes tasks like numerical **transformations**, **aggregating data**, encoding text or image data, and **creating new features**.



Data preparation is a critical phase in the machine learning workflow, shaping the quality of input data for model training.

Let's delve into the key steps involved:

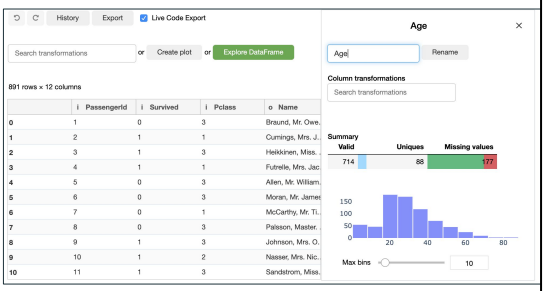
- Data Cleaning:
  - Data cleaning involves identifying and rectifying errors or inconsistencies in the dataset. For example, removing duplicate records, handling missing values, or addressing outliers to ensure data integrity.
- Data Imputation:
  - Imputing missing values is crucial for maintaining a complete dataset. Techniques such as mean, median, or predictive modeling can be employed to fill in missing values and preserve the integrity of the dataset.

Data Preparation for ML projects

Goal: Optimize input quality for accurate model predictions

Data preparation includes the following tasks:

- **Cleaning and formatting data:** This includes tasks such as **handling missing values** or outliers, ensuring data is in the correct format, and removing unneeded columns.
- **Feature Engineering:** This includes tasks like numerical **transformations**, **aggregating data**, encoding text or image data, and **creating new features**.



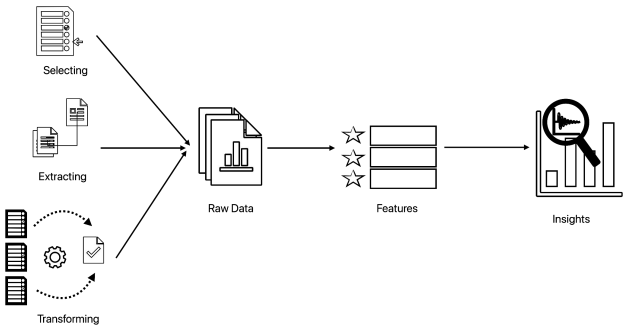
- Data Standardization:
  - Standardizing data ensures that all features have a consistent scale, preventing certain features from dominating others. For instance, standardizing numerical variables to have a mean of 0 and a standard deviation of 1.
- Accuracy Optimization:
  - This step involves optimizing data to enhance model accuracy. It may include adjusting class imbalances, choosing appropriate evaluation metrics, and ensuring that the dataset is well-suited for the specific machine learning task.
- Feature Engineering:
  - Feature engineering involves creating new features or modifying existing ones to improve model performance. For example, converting categorical variables into numerical representations, creating interaction terms, or scaling features for better interpretability.

Effective data preparation sets the foundation for a successful machine learning model. These steps ensure that the data used for training and testing is clean, complete, and standardized, ultimately contributing to the model's ability to generalize well to new, unseen data.

Feature Engineering

Transforming raw data into model-friendly features

- It is a critical step in preparing data for machine learning models.
- It transforms raw data into a format that allows ML models to extract meaningful patterns and make accurate predictions.



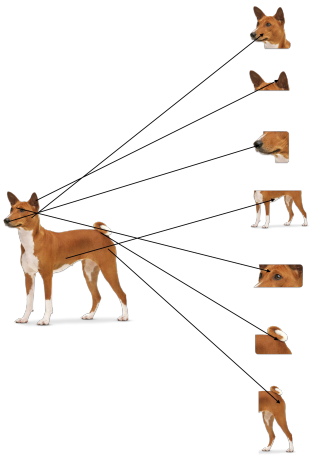
Feature engineering is the art of transforming raw data into a more suitable format for machine learning models. Let's explore the significance of this process:

- It is foundational in the data preprocessing pipeline. It involves manipulating and creating features to enhance the model's understanding of the underlying patterns in the data. For example, in a dataset with a timestamp, extracting day, month, and year as separate features can provide additional temporal information for the model.
- Raw data is often complex and may not be in a form directly interpretable by machine learning models. Feature engineering transforms this raw data into a model-friendly format, enabling the model to extract meaningful insights. In the context of text data, converting text into numerical representations using techniques like TF-IDF or word embeddings is a common feature engineering practice.
- Feature engineering goes beyond the original data, involving the creation of new features or representations. For instance, in a dataset with location information, calculating distances from specific landmarks or creating a new feature indicating proximity to certain points of interest can provide valuable additional information for the model.

Feature Extraction

Transforming raw data into a set of features that better represent the underlying patterns

- Transforming Raw Data for Enhanced Modeling
- Dimensionality Reduction for Improved Performance
- Simplifying Feature Engineering



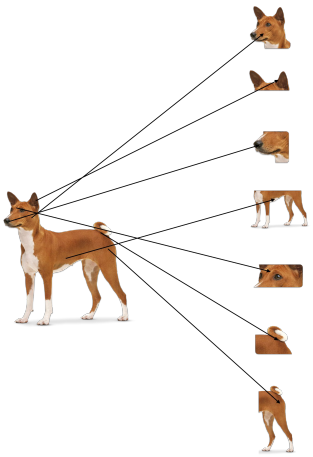
Feature extraction is a crucial step in the machine learning pipeline, focusing on transforming raw data into a more meaningful and compact representation. Let's break down the key aspects:

- Transforming Raw Data for Enhanced Modeling:
  - Feature extraction involves converting raw data into a format that enhances a model's ability to identify patterns and relationships. For example, in natural language processing, converting text into numerical representations using techniques like TF-IDF or word embeddings.
- Dimensionality Reduction for Improved Performance:
  - As datasets grow in size, the number of features can become overwhelming. Dimensionality reduction techniques, such as Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE), help streamline the dataset by reducing the number of features while preserving essential information.
- Simplifying Feature Engineering:
  - Feature extraction simplifies the feature engineering process. Instead of using the raw, potentially noisy data, extracting relevant features streamlines the modeling process, making it more efficient and effective.

Feature Extraction

Transforming raw data into a set of features that better represent the underlying patterns

- Transforming Raw Data for Enhanced Modeling
- Dimensionality Reduction for Improved Performance
- Simplifying Feature Engineering



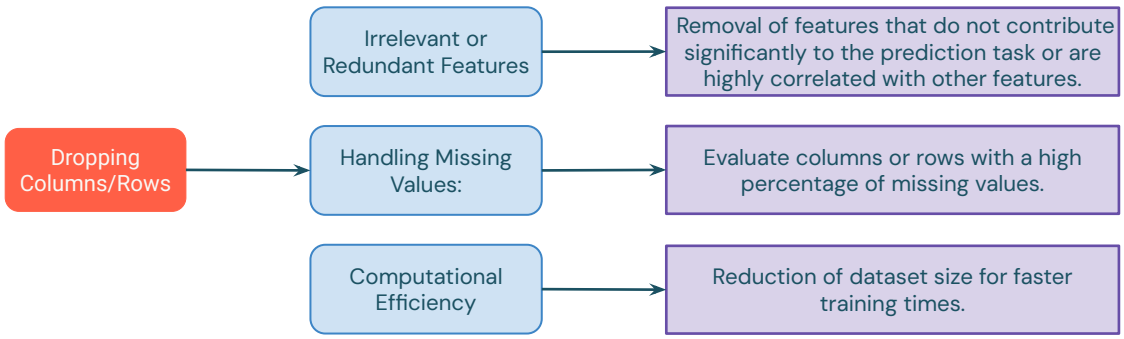
Example:

Consider a dataset containing images for facial recognition. The raw data consists of pixel values for each image. Feature extraction in this context could involve techniques like Principal Component Analysis (PCA) to identify the most informative features capturing facial features such as eyes, nose, and mouth. This transformed set of features not only reduces dimensionality but also focuses on the critical aspects for facial recognition, improving the model's accuracy and efficiency. Feature extraction, in this example, simplifies the representation of facial features, making it more conducive for effective modeling.

Feature Selection: Dropping Columns/Rows

Enhancing Model Efficiency

Streamlining the dataset by selectively removing columns (features) or rows (instances) to improve model efficiency and effectiveness.

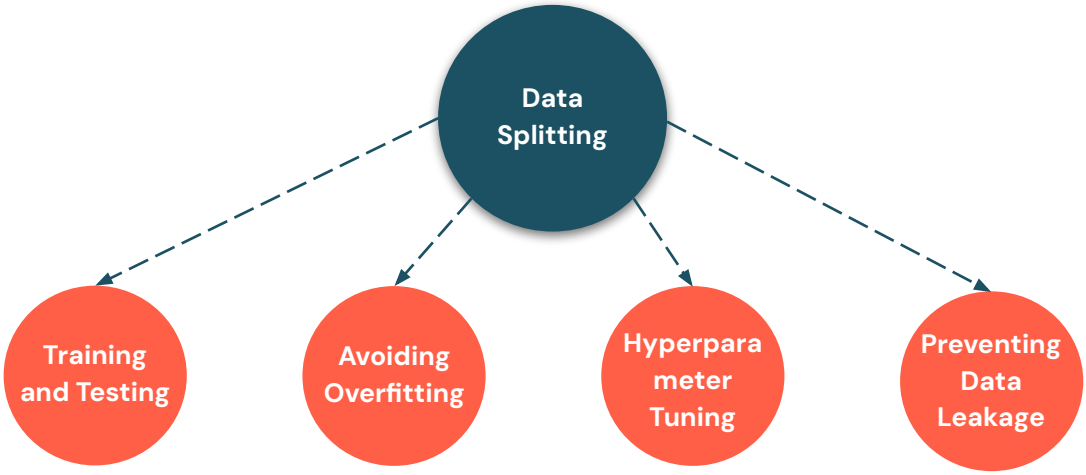


Feature selection is vital for preparing data for machine learning. Key steps include:

- **Dropping Irrelevant Data:** Remove constant or uninformative columns/rows to improve efficiency.
  Managing Redundancy: Eliminate highly correlated features to avoid multicollinearity and ensure diverse inputs.
- **Handling Missing Values:** Impute or drop features with excessive gaps to maintain data integrity.
- **Outlier Treatment:** Detect and address outliers to prevent skewed results and improve model robustness.

Why Do We Need Splitting Data

Optimizing Model Evaluation

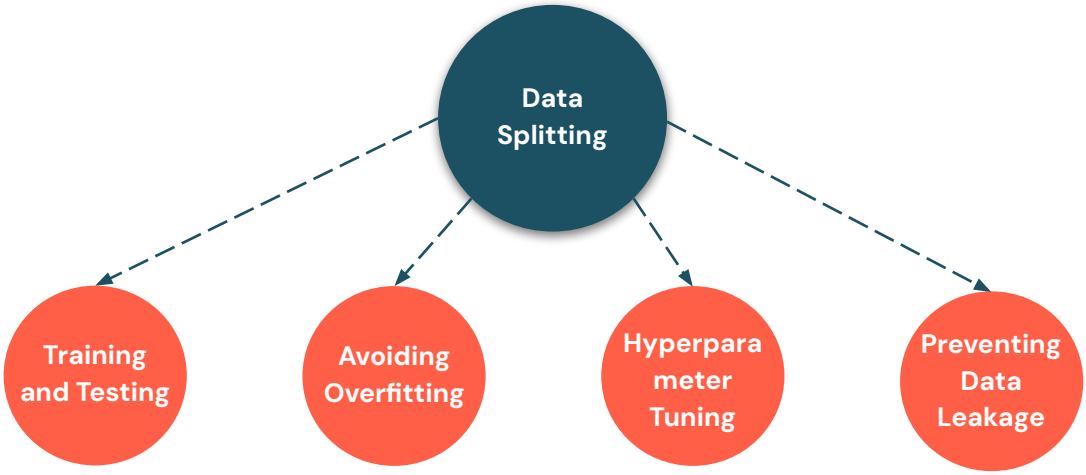


Optimizing model evaluation involves strategic data splitting, and various techniques can enhance this process.
Let's explore the key aspects:

- Detecting Overfitting:
  - Splitting data into training and validation sets helps identify overfitting. Overfitting occurs when a model performs exceptionally well on the training data but fails to generalize to new, unseen data. By comparing performance on the training and validation sets, one can detect overfitting.
- Evaluating with Different Data Folds:
  - Instead of a single train-validation split, techniques like cross-validation involve multiple data folds. For example, in k-fold cross-validation, the data is divided into k subsets, and the model is trained and validated k times, with each subset serving as the validation set exactly once.

Why Do We Need Splitting Data

Optimizing Model Evaluation



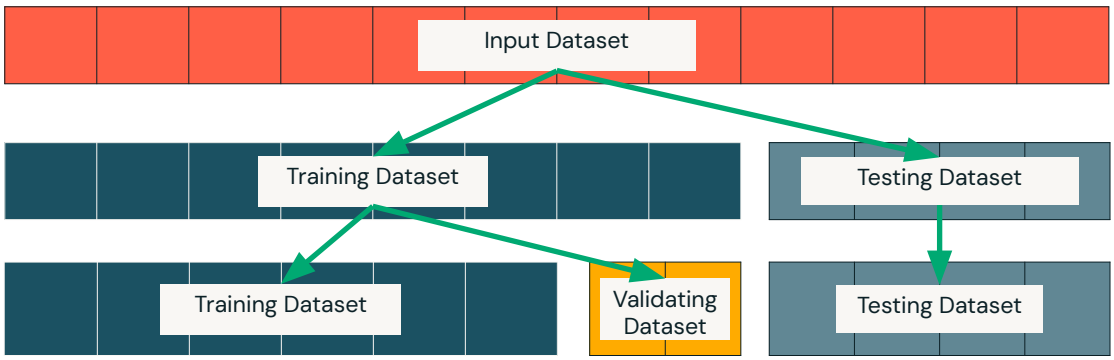
- Validating Performance on New Data:
  - Ensuring that model evaluation includes a test set not seen during training is crucial. This set represents new, unseen data, providing a more realistic assessment of the model's ability to generalize to different scenarios.
- Identifying Training Excellence vs. Generalization:
  - By splitting data strategically, one can distinguish between training excellence and generalization. A model may achieve high accuracy on the training set, but its true test lies in its performance on validation or test sets, which represent real-world scenarios.

Strategically splitting data and employing techniques like cross-validation contribute to a robust model evaluation process. It helps ensure that the model's performance is not biased by specific subsets of the data, fostering confidence in its ability to generalize and make accurate predictions on new, unseen instances.

Splitting Data into Multiple Sets

Optimizing Model Training, Validation, and Testing

Splitting the dataset is a fundamental step in data preparation to facilitate effective model training, validation, and testing.



Splitting the input dataset into training, validation, and testing sets is an essential step before model building.

- The training set is used to train the model.
- The validation set helps tune model parameters and avoid overfitting.
- The test set is used to evaluate the final model performance on unseen data.

This structure ensures that the model is trained effectively and can generalize well.

Sampling Methods

Sampling methods play a crucial role in selecting subsets of data for training, validation, and testing, influencing the performance and generalization of machine learning models. Always take care in how you are performing your sampling for your particular dataset.

- ***Splitting data involves dividing the entire dataset into separate subsets, Sampling is crucial for cases where it is impractical or unnecessary to use the entire dataset***
    - strategically select which data points go into each subset.

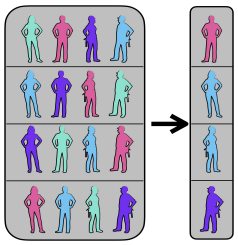
Sampling is used to create subsets of data for training, validation, and testing. It affects how well a machine learning model performs and generalizes. It's important to choose the right sampling method based on your dataset.

Sampling Methods

Sampling methods play a crucial role in selecting subsets of data for training, validation, and testing, influencing the performance and generalization of machine learning models. Always take care in how you are performing your sampling for your particular dataset.

Here's an overview of common sampling methods:

Random Sampling:



This is the most common and basic sampling technique in machine learning. It randomly selects data points from a dataset without following any specific pattern, assuming that each data point has an equal probability of being chosen.

Example with a pandas DataFrame

```
# my_data is a pandas DataFrame and we want to sample 50 data points

my_random_sample = my_data.sample(n=50)
```


- **Random Sampling:**
  - Selects a random subset of data points from the entire dataset.
  - Useful when the dataset is large and representative of the overall population.

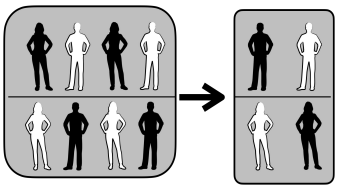
Random sampling selects data points from the dataset without following a specific pattern. Every data point has an equal chance of being picked. It’s one of the simplest ways to create a sample from a larger dataset.

Sampling Methods

Sampling methods play a crucial role in selecting subsets of data for training, validation, and testing, influencing the performance and generalization of machine learning models. Always take care in how you are performing your sampling for your particular dataset.

Here's an overview of common sampling methods:

Stratified Sampling:



This sampling method is used when a dataset consists of subgroups, ensuring that samples are taken from each. The population is divided into homogeneous subgroups, known as strata, and an appropriate number of instances is selected from each stratum to ensure the sample accurately represents the entire population.

Example with a pandas DataFrame & scikit-learn

```
# my_data is a pandas DataFrame
# group_label is the column containing subgroups

from sklearn.model_selection import train_test_split

stratified_train, stratified_test = train_test_split(my_data, test_size =0.2, stratify = my_data['group_label'])
```


- **Stratified Sampling:**
  - Divide into subgroups based on specific characteristics or classes.
  - then selects samples from each subgroup proportionally
    - **useful when dealing with imbalanced datasets** or when **preserving class distributions** is important.

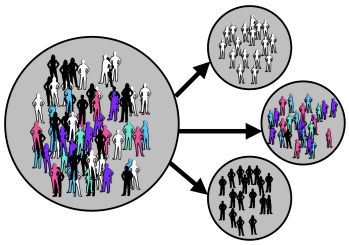
Stratified sampling divides the dataset into subgroups or strata, and samples are taken from each group. This helps maintain the original distribution of the subgroups in the sample.

Sampling Methods

Sampling methods play a crucial role in selecting subsets of data for training, validation, and testing, influencing the performance and generalization of machine learning models. Always take care in how you are performing your sampling for your particular dataset.

Here's an overview of common sampling methods:

Cluster Sampling:



Cluster sampling is a sampling technique where the population is divided into naturally occurring groups, or clusters, and a subset of these clusters is randomly selected for study. Instead of sampling individuals directly, researchers collect data from all individuals within the chosen clusters.

Example with a pandas DataFrame and numpy

```
# my_data is a pandas DataFrame
# my_col is the column containing subgroup identifiers
# Randomly select M clusters from the range 1 to N

import pandas as pd
import numpy as np

clusters = np.random.choice(np.arange(1, N+1), size=M, replace=False)
cluster_sample = my_data[my_data['my_col'].isin(clusters)]
```


- **Cluster Sampling:**
  - *Divides* the dataset into clusters and selects entire clusters as samples.
  - The main reason behind the cluster sampling is the cost reduction
  - typically formed based on some natural grouping or geographical proximity.
  - This method is useful when the dataset has a natural clustering structure, and it can help capture the variability within each cluster.
  - introduces a potential source of bias.
  - *Example:*
    - *Divide the entire country into clusters based on states or regions*
    - *Randomly select a subset of these clusters for detailed study.*
      - *you might choose 5 states randomly from all geographic regions (Northeast, Midwest, South, West).*
- ***Choosing the Right Method: Depends on the data structure, research goals, and resource availability.***

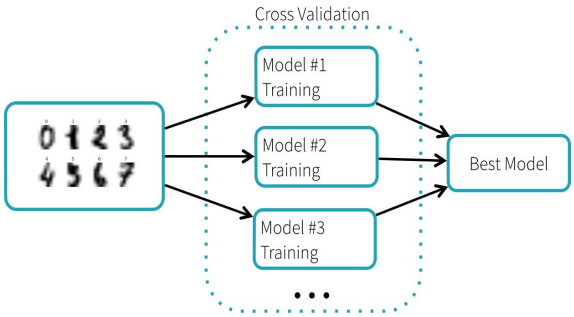
In cluster sampling, the data is divided into natural groups, or clusters. A few clusters are selected randomly, and all data points from those clusters are included in the sample.

Cross-Validation

Cross-validation is a resampling technique used to assess the performance of a machine learning model. It involves partitioning the dataset into subsets to train and evaluate the model multiple times, providing a more robust estimate of its generalization performance.

Here are the key elements of cross-validation:

- K-Fold Cross-Validation
- Stratified K-Fold Cross-Validation
- Shuffle Split Cross-Validation
- Nested Cross-Validation



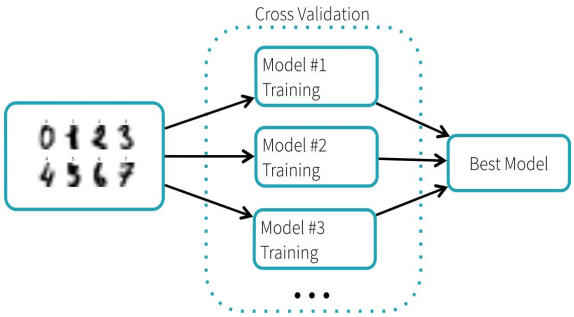
- **Cross Validation =** dividing data into subsets to train and test the model **repeatedly.**
  a. **Ensures consistent model performance across various data subsets**,
  b. enhancing confidence in its ability to generalize.
- **K-Fold Cross-Validation**
  - **Process**: Divide the dataset into **K equal-sized parts (folds)**. Train the model on "**K minus 1" folds** and test it on the **remaining fold**.
  - **Repetition**: This is **done K times**, *each fold = test set once*.
  - **Final**: Metrics from **each fold are averaged** to estimate the model's overall performance.
  - *Use Case: Ideal for most scenarios, ensuring that all data is used for both training and testing.*

Cross-Validation

Cross-validation is a resampling technique used to assess the performance of a machine learning model. It involves partitioning the dataset into subsets to train and evaluate the model multiple times, providing a more robust estimate of its generalization performance.

Here are the key elements of cross-validation:

- K-Fold Cross-Validation
- Stratified K-Fold Cross-Validation
- Shuffle Split Cross-Validation
- Nested Cross-Validation



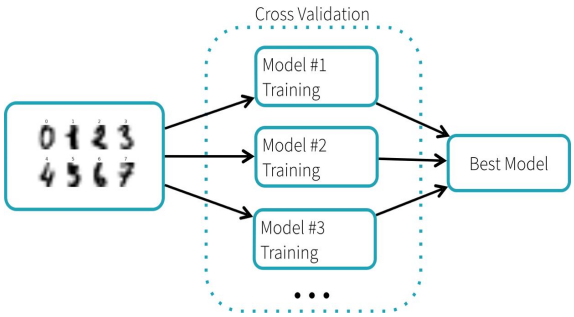
    - **Stratified K-Fold Cross-Validation**
    - Each fold maintains a **similar distribution of target classes**, crucial for **imbalanced datasets**.
    - **Advantage =** Prevents model bias towards the majority class.
- **Shuffle Split Cross-Validation**
    - **Random shuffling** of data **before splitting** into training and testing subsets.
    - Allows for precise adjustments of the training and testing subset sizes.
    - *Applicability: Ideal for studies requiring specific configurations of data volumes..*
- **Nested Cross-Validation**
    - **Set up K-Fold Cross-Validation**
    - Each subset's **training data is split into training/test** and assess across **different hyperparameters…**Best is chose for that subset
    - done for each K-fold
- ***The choice of technique depends on the dataset's size, balance, and the specific requirements of the project.***

Cross-Validation

Cross-validation is a resampling technique used to assess the performance of a machine learning model. It involves partitioning the dataset into subsets to train and evaluate the model multiple times, providing a more robust estimate of its generalization performance.

Here are the key elements of cross-validation:

- K-Fold Cross-Validation
- Stratified K-Fold Cross-Validation
- Shuffle Split Cross-Validation
- Nested Cross-Validation



Choosing the Right Technique

- **K-Fold**: Commonly used due to its simplicity and effectiveness.
- **Stratified K-Fold**: Preferred for datasets with unequal class distributions to ensure fairness.
- **Shuffle Split**: Offers control over the split ratio, ideal for experiments with varying data sizes.
- **Nested Cross-Validation**: Best for projects requiring rigorous model evaluation and selection.

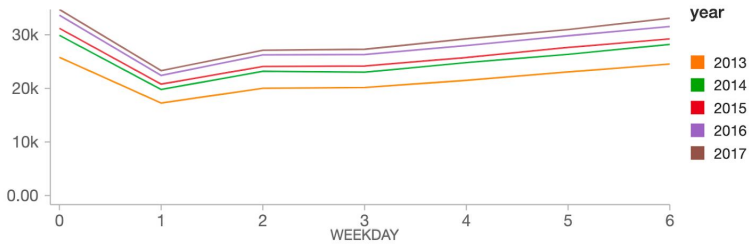
Cross-validation is a resampling method used to evaluate the performance of a machine learning model.It works by splitting the dataset into multiple subsets. The model is trained on some subsets and validated on the remaining ones. This process is repeated multiple times to get a reliable estimate of how well the model performs.

Common types of cross-validation include:

- K-Fold Cross-Validation, where the data is divided into K parts and the model is trained and tested K times.
- Stratified K-Fold, which ensures that each fold maintains the original distribution of the target variable.
- Shuffle Split, which creates random train-test splits.
- Nested Cross-Validation, used when tuning hyperparameters to avoid data leakage between training and testing.

Sampling for Time-Series Data

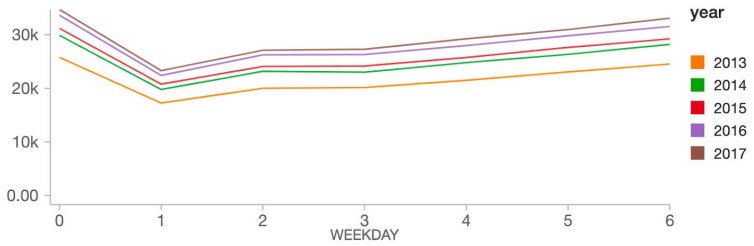
Sampling for **time-series data requires special consideration** due to the temporal dependencies inherent in the data. **Traditional random sampling or shuffling may not be appropriate**, as the order of events in time is crucial.



- **Time-series data** capture events in a sequential order that is **critical for accurate analysis**.
  - **Random** shuffling or sampling **disrupts these dependencies**, potentially skewing results.
- **Techniques for Effective Sampling**:
  - **Sequential Sampling**: Select data in **consecutive segments**
  - **Sampling Frequency**: determines the **granularity of the data**.
  - **Sliding Windows**: Employ a **fixed-size window** of consecutive time intervals .
- **Considerations**:
  - **Validate Samples**: check if patterns and trends are preserved
  - Software and Libraries designed for time-series analysis = **pandas for Python**, **timeSeries package in R**.
  - **To ensure reproducibility Document the Process**

Sampling for Time-Series Data

Sampling for **time-series data requires special consideration** due to the temporal dependencies inherent in the data. **Traditional random sampling or shuffling may not be appropriate**, as the order of events in time is crucial.



Time-series data requires special handling because the order of data points matters. Random sampling or shuffling breaks the sequence and can lead to incorrect model evaluation.

Instead, sampling should preserve the time order.
For example, we might train a model on earlier data points and test it on later ones. This approach respects the natural flow of time and helps the model learn from past trends to predict future outcomes.

This is important in domains like finance, IoT, or web traffic forecasting, where events are time-dependent.

Thank you for completing this lesson and continuing your journey to develop your skills with us.