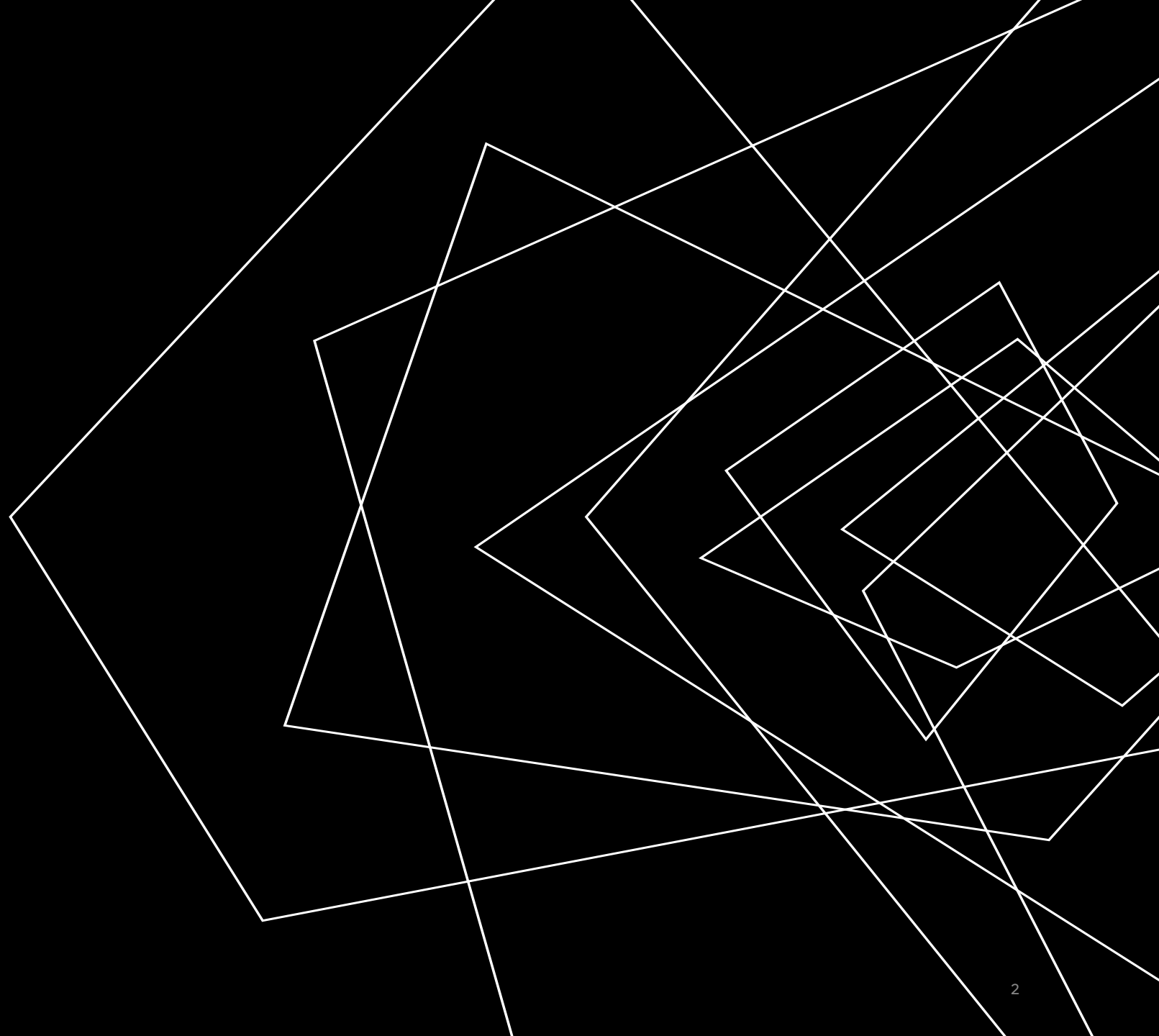
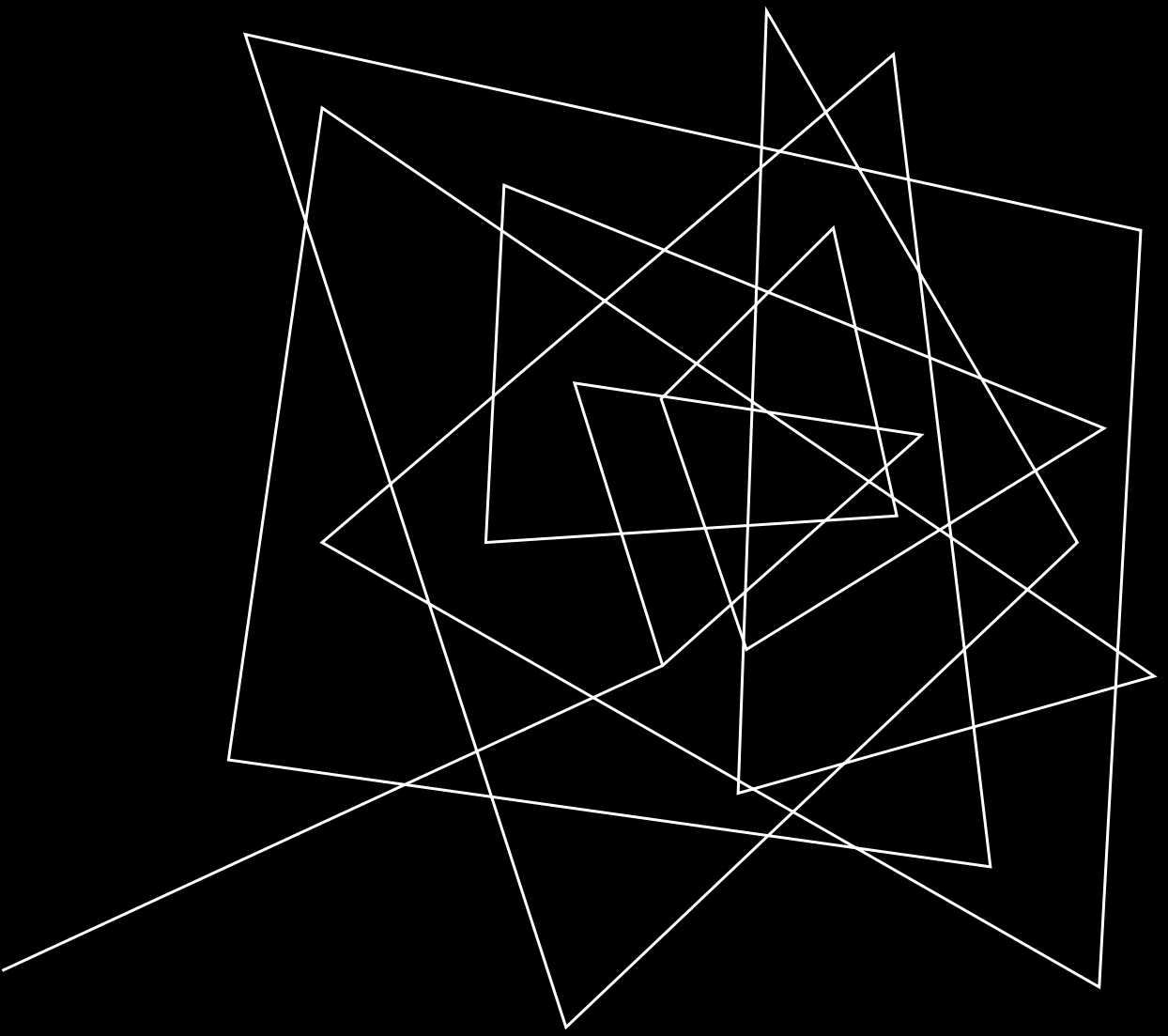
ML Project Workflow: Data Preprocessing & Hyperparameter Tuning

BY
SAIFUL BARI IFTU
LECTURER, DEPT. OF CSE, BRAC UNIVERSITY

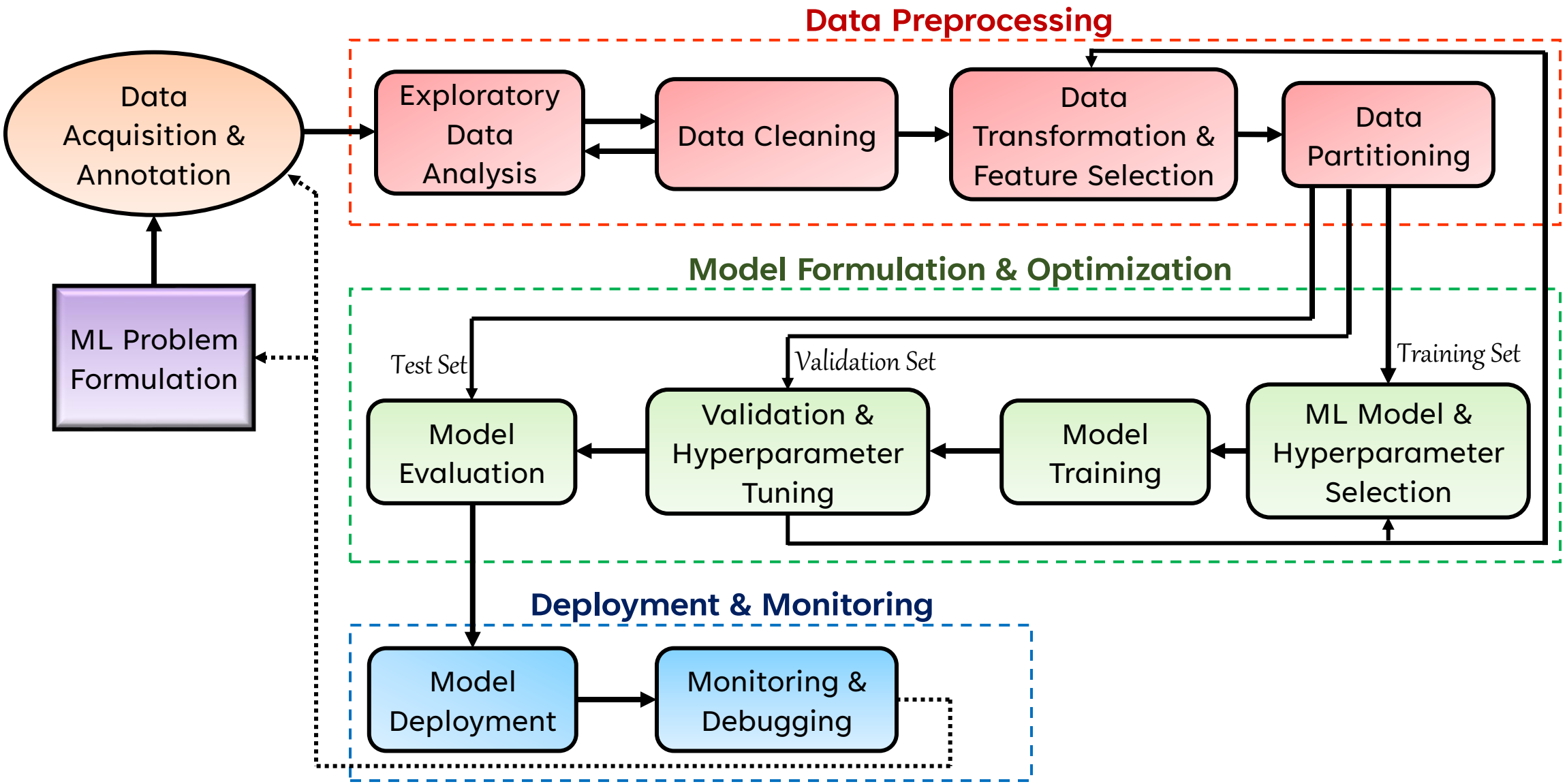
CONTENTS





MACHINE LEARNING WORKFLOW

MACHINE LEARNING WORKFLOW



DATA ACQUISITION & ANNOTATION

Data Acquisition/Collection

Data acquisition is the process of collecting raw data from various sources to build machine learning models. It involves gathering data relevant to the problem at hand, which can be structured (tables, spreadsheets) or unstructured (images, videos, text). Data can come from internal databases, APIs, sensors, web scraping, or third-party providers. Quality and quantity are crucial in this phase because poor or insufficient data can result in unreliable models.

Key Considerations in Data Collection

- **Relevance**: Data should be directly aligned with the problem the model aims to solve.
- **Quality**: Data cleanliness, consistency, and accuracy should be ensured.
- **Quantity**: There should be enough data to train a robust model.
- **Diversity**: Data should cover a wide range of scenarios to help the model generalize well.
- **Ethics**: Proper consent must be obtained for data collection, & privacy laws should be followed.
- **Reliability**: The credibility and trustworthiness of the data sources must be verified.

DATA ACQUISITION & ANNOTATION

Data Annotation

Data annotation refers to the process of labeling or tagging data so that machine learning models can understand and learn from it. In supervised learning, annotated data is critical because the model uses these labels during training to learn patterns and make predictions.

Depending on the task, annotation can take various forms, such as tagging objects in images, identifying sentiment in text, or labeling sound clips. Data annotation can be done manually or with the help of semi-automated tools and can be time-consuming, but it is essential for improving the model's accuracy and performance. Proper annotation ensures that the data is correctly labeled and provides the necessary context for the model to generalize effectively.

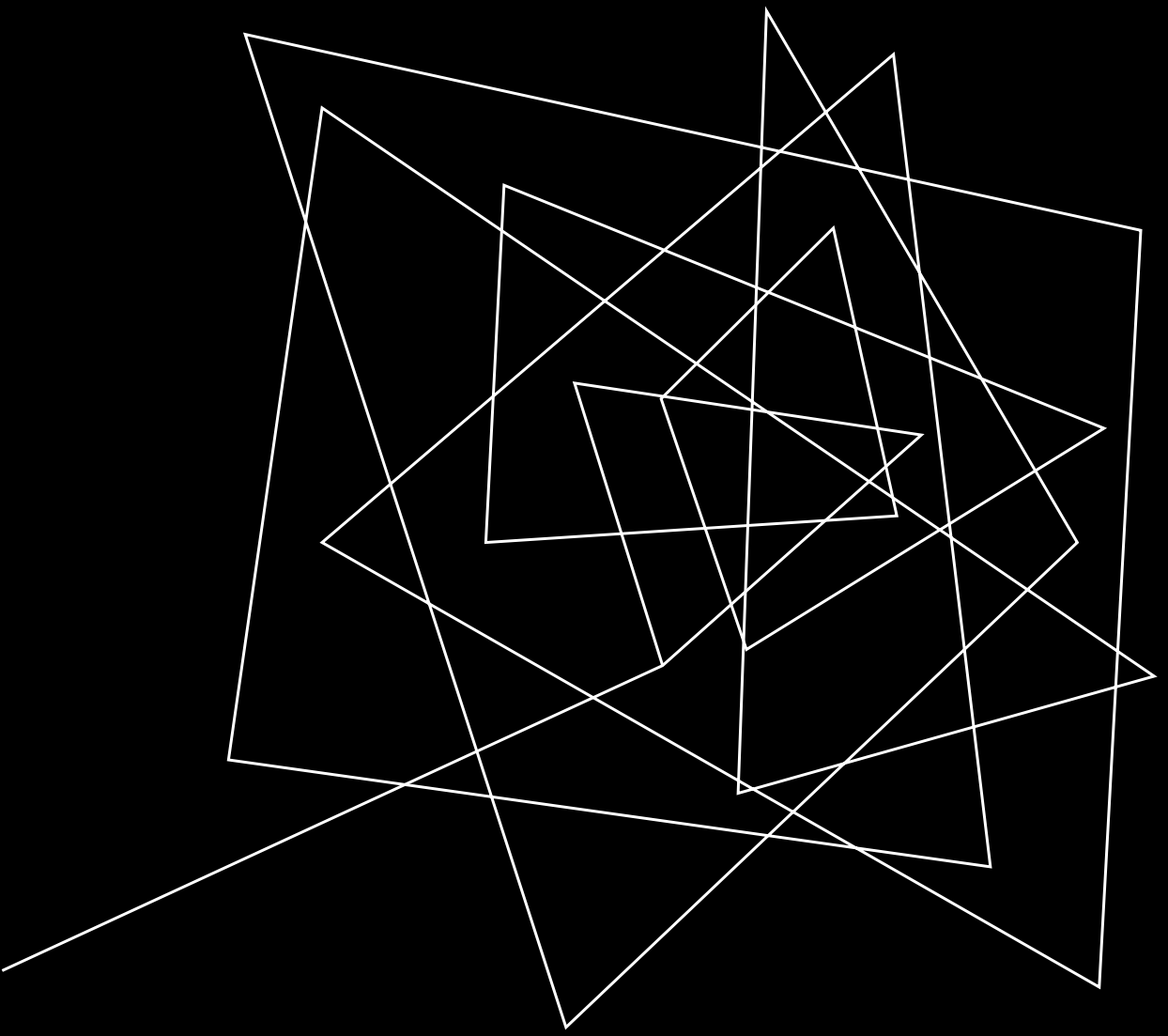
Annotators with the necessary domain expertise should be chosen to correctly interpret and label the data, especially for complex or specialized tasks (e.g., medical or legal data). This is critical because poor-quality annotations can jeopardize the whole process.

DATASETS

Datasets are a fundamental component of Machine Learning, providing the raw data necessary for training, validation, and testing. A dataset typically consists of features (input variables) and labels (output variables or targets) for supervised learning, or only features in the case of unsupervised learning. Machine Learning is basically ‘machines’ ‘learning’ from ‘data’. *Datasets provide that data.* **Datasets are basically the results of Data Collection and Annotation.**

Here are some common sources for publicly available datasets:

- ❑ Kaggle
- ❑ UCI Machine Learning Repository
- ❑ Google Dataset Search
- ❑ Data.gov
- ❑ Amazon AWS Public Datasets
- ❑ OpenML
- ❑ University Repositories (e.g. Stanford, MIT)
- ❑ GitHub Repositories



DATA WRANGLING/ PREPROCESSING

"Data Preprocessing is more than 80% of an ML Project." ----- Andrew NG

EDA (EXPLORATORY DATA ANALYSIS)

Exploratory Data Analysis (EDA) is an essential step in the data science & ML process that involves examining and summarizing the main characteristics of a dataset. The goal of EDA is to understand the data, uncover patterns, spot anomalies, test hypotheses, and check assumptions using both statistical and graphical methods before building predictive models. Techniques like descriptive statistics (mean, median, mode, distribution) and visualizations (histograms, scatter plots) are used to gain an overview of the data's characteristics.

EDA Techniques

- **Summary Statistics**: Mean, median, mode, standard deviation, quartiles, etc.
- **Frequency Distribution**: The number of times each value in a dataset occurs.
- **Correlation Analysis**: Quantifying correlations/relationships between features/variables.
- **Visualization**: Box Plot, Scatter Plot, Histograms, Bar Chart, Heatmap, Geographic Map, etc.

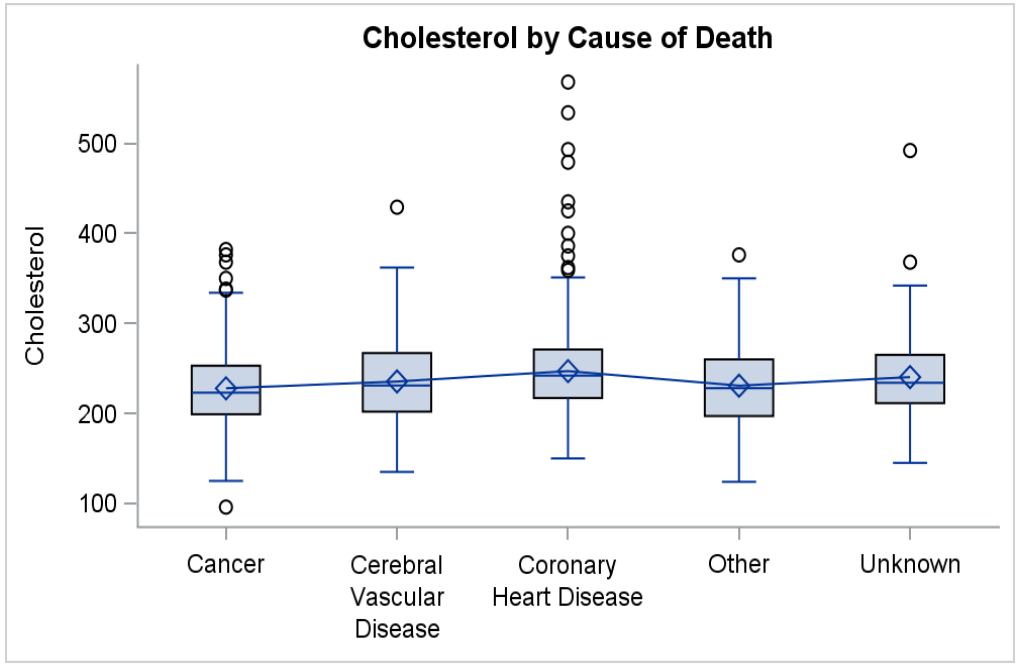
  These are very important to understand the distribution and correlations in data.
- **Dimensionality Reduction**: Visualizing high-dimensional data in a 2D or 3D space (e.g., PCA).

EDA (EXPLORATORY DATA ANALYSIS)

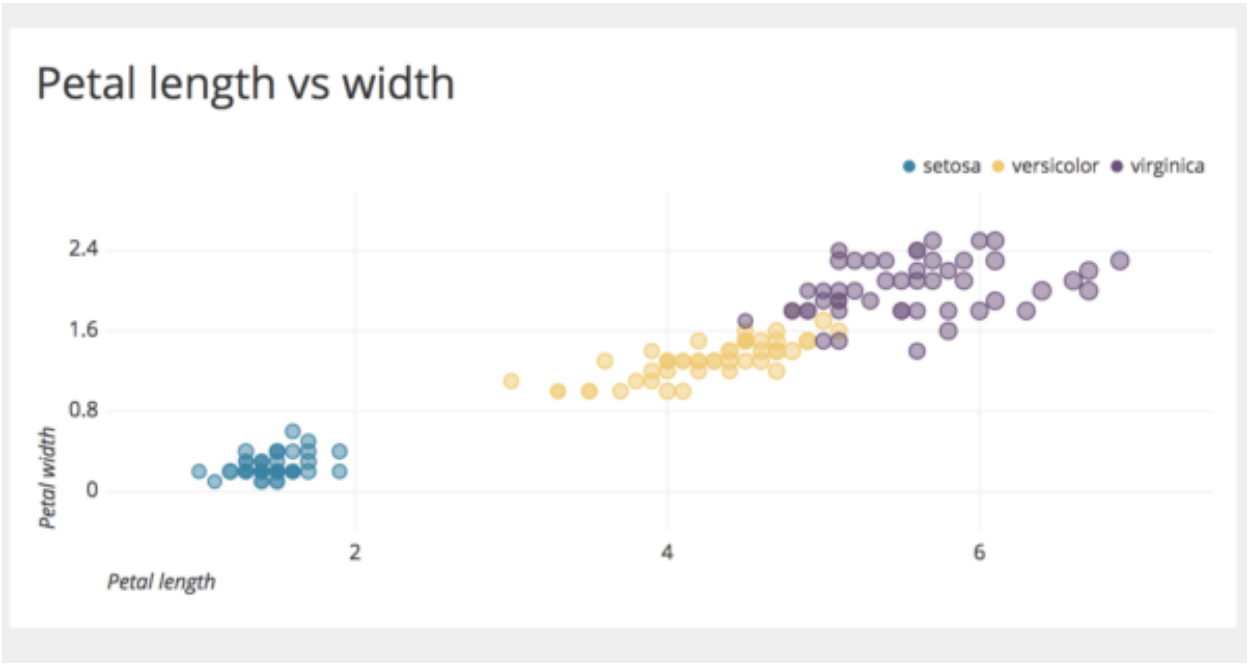
Key Objectives of EDA

- **Understanding Data Distribution:** Determining how the data is distributed (normal/skewed) and identifying central tendencies (Mean, Median, etc.).
- **Identifying Missing Values:** EDA helps in spotting missing data points and understanding their patterns (e.g., missing at random, missing completely).
- **Spotting Outliers:** Detecting unusual data points that may skew the results of the analysis or model.
- **Testing Relationships Between Variables:** Testing for multicollinearity and helping in determining which features might be relevant for modeling and which can be removed or transformed.
- **Forming Hypotheses:** By visually exploring data, hypotheses can be generated about how different variables interact and whether there are potential insights to uncover.
- **Checking Assumptions:** Validating assumptions such as linearity or independence between features.
- **Data Transformation:** Identifying needs for feature scaling, encoding, or transformations (log transformations, polynomial features) based on insights gathered from initial exploration.

EDA: VISUALIZATION

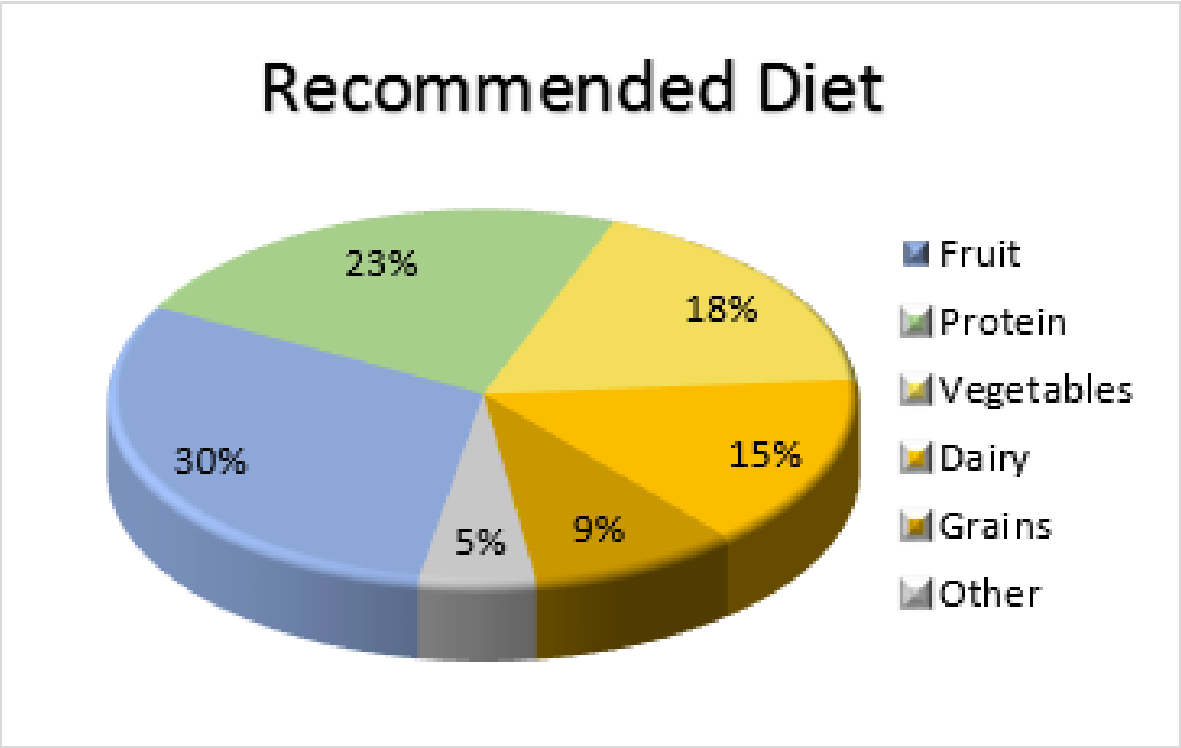


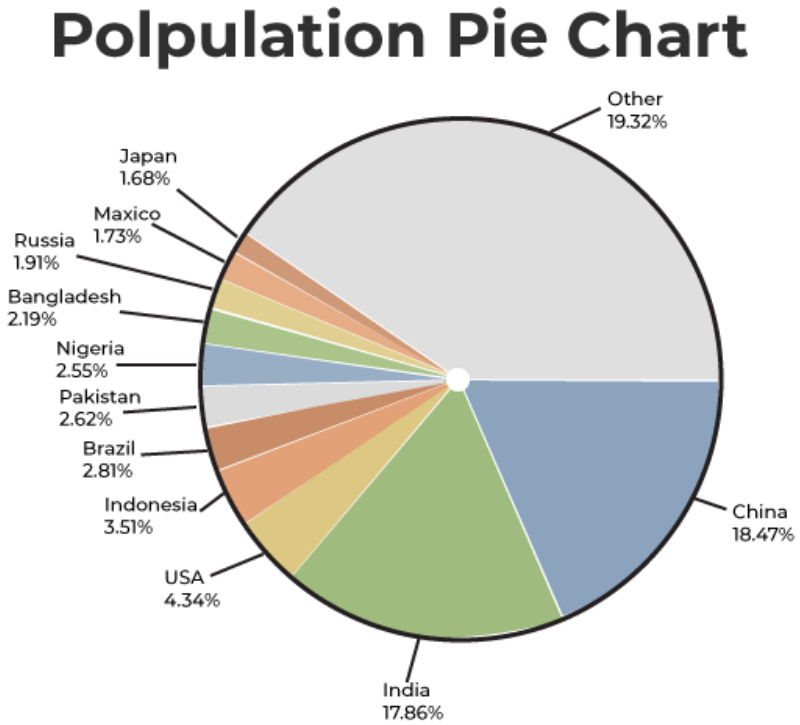
Box Plot



Scatter Plot

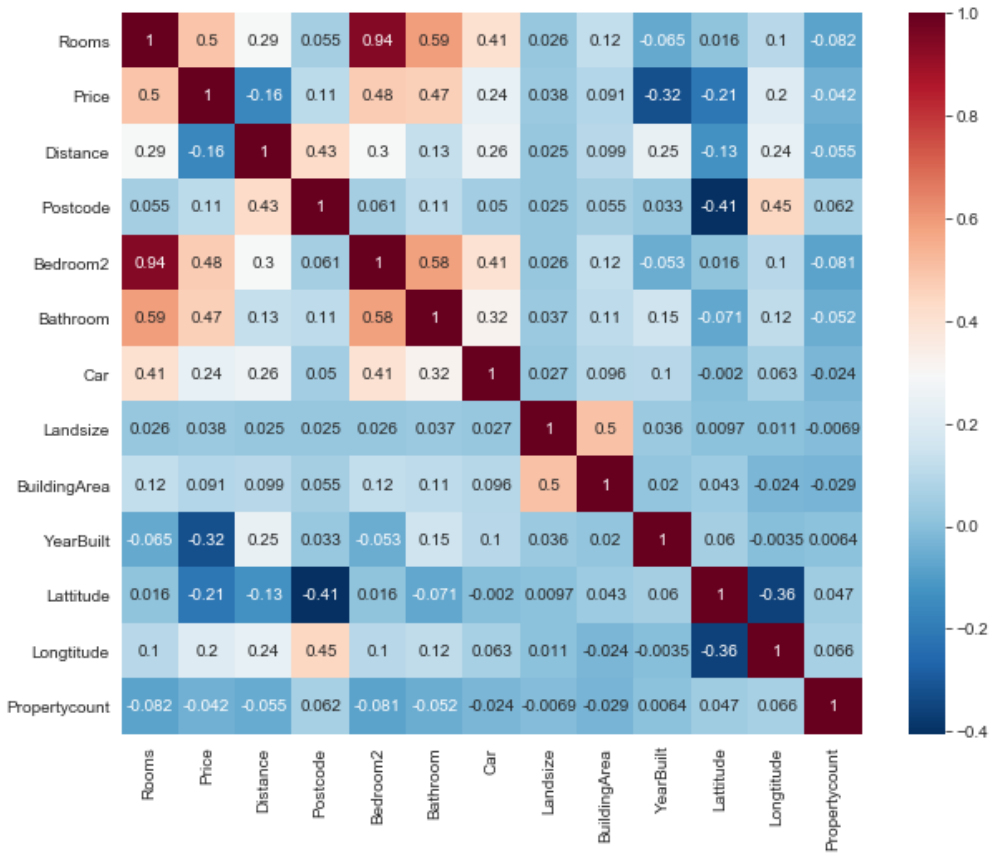
EDA: VISUALIZATION

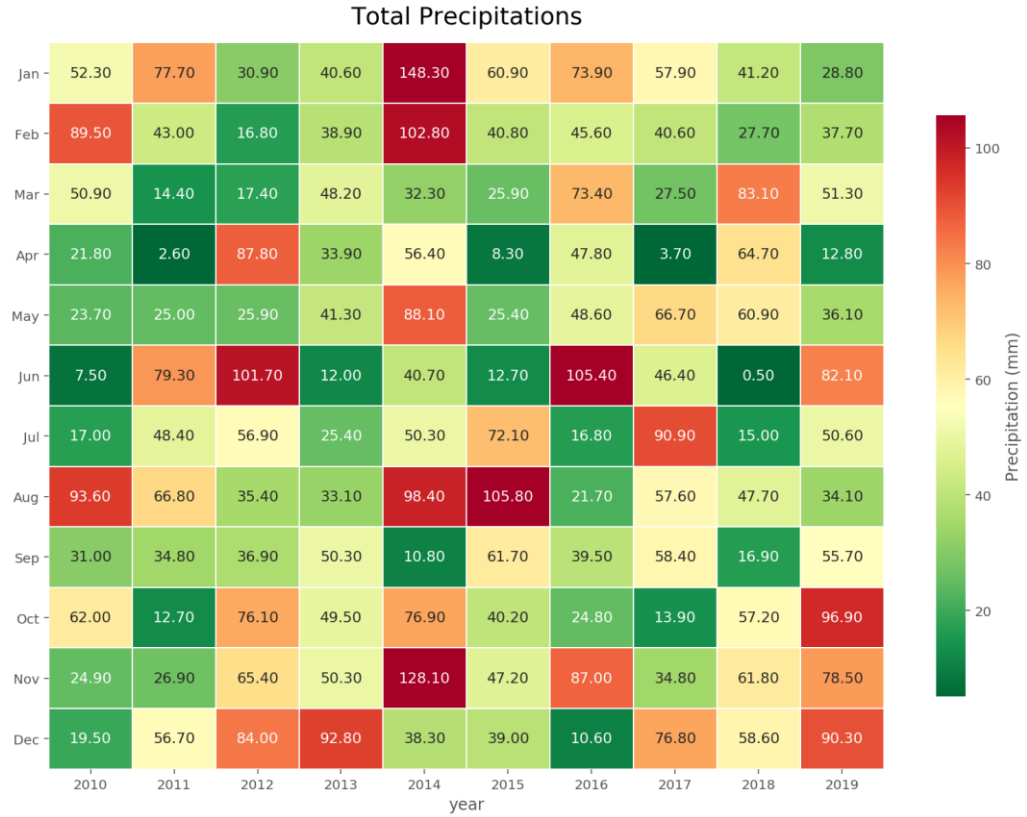




Pie Chart

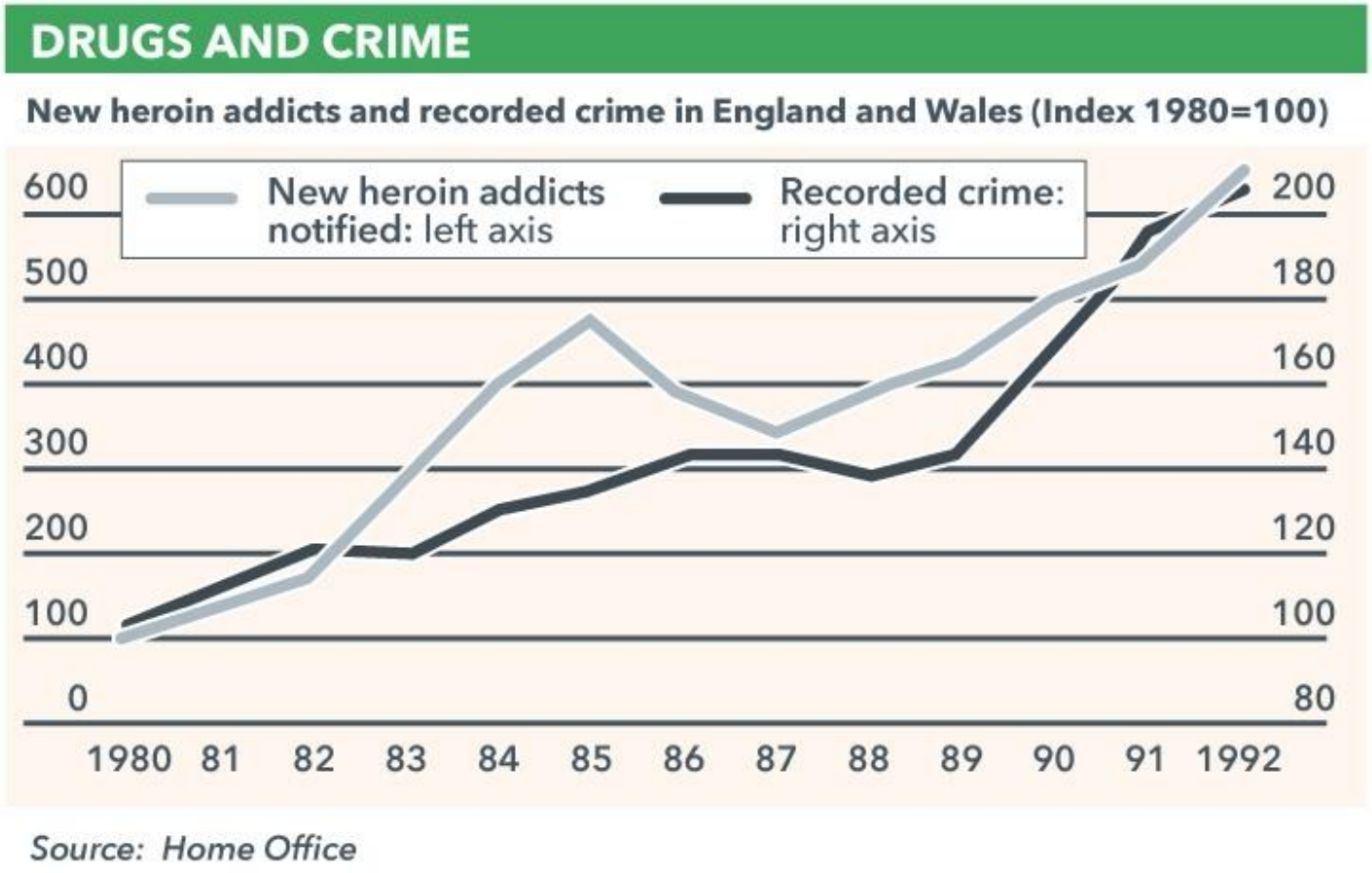
EDA: VISUALIZATION





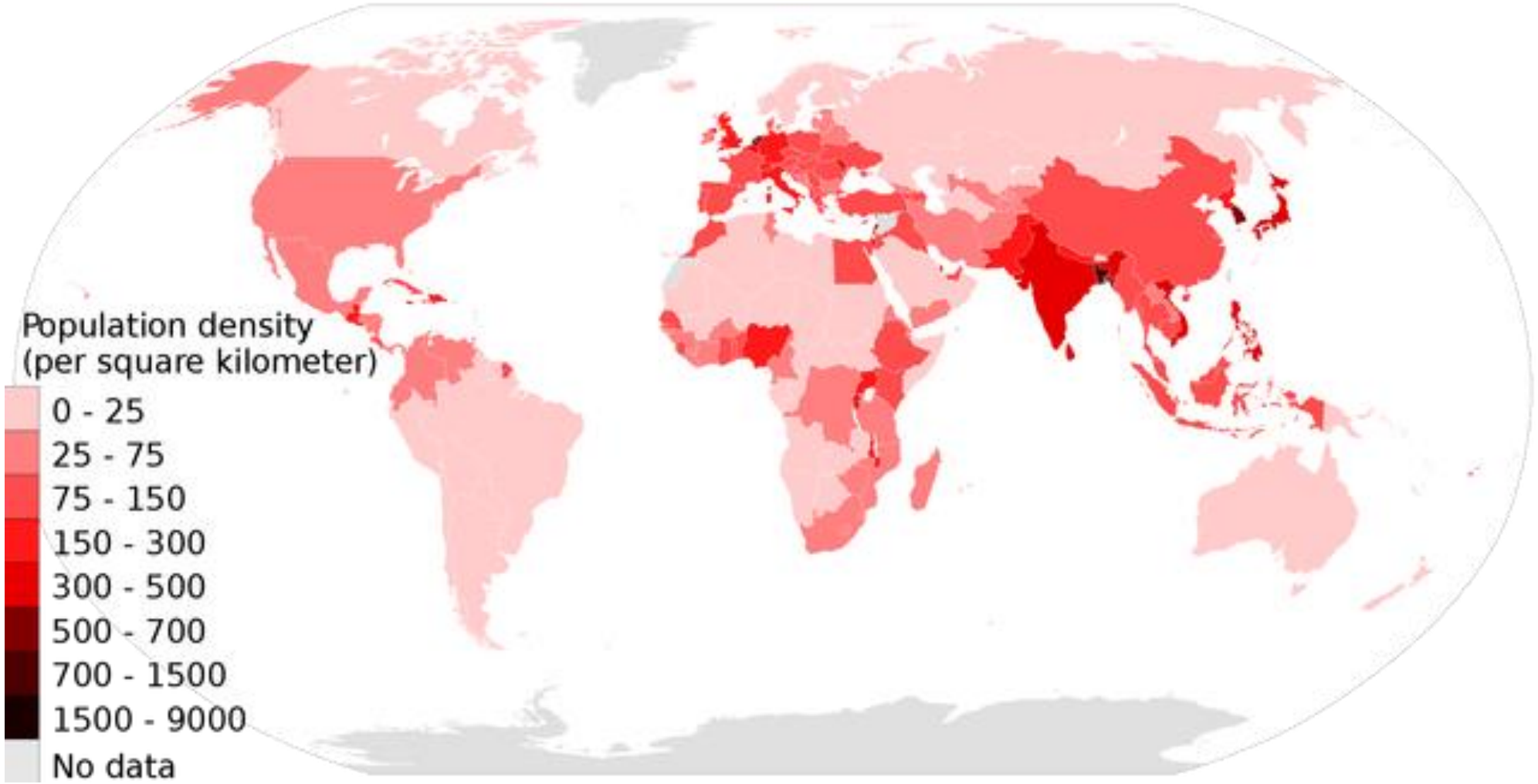
Heatmap

EDA: VISUALIZATION



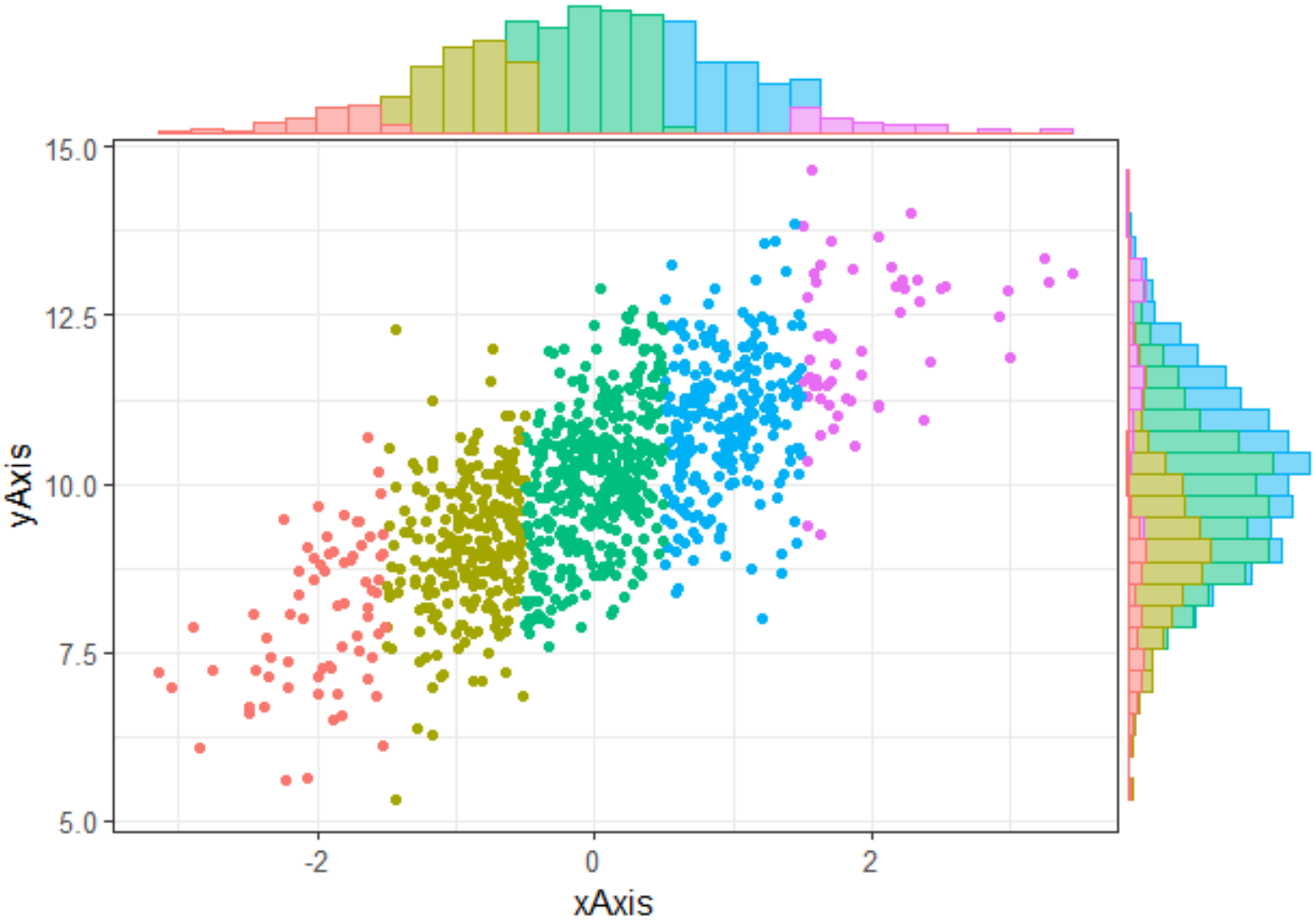
Time Series

EDA: VISUALIZATION



Geographic Map

EDA: VISUALIZATION



Mixed Plot

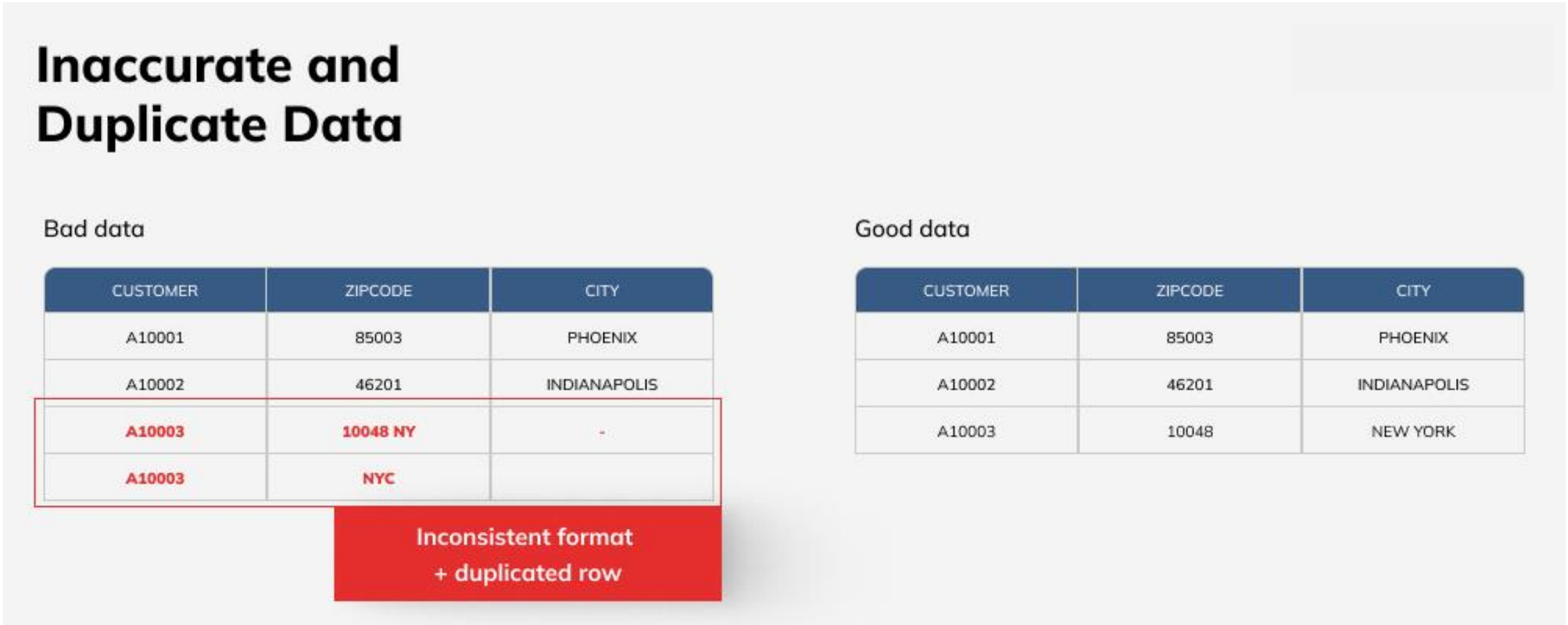
DATA CLEANING

Data Cleaning is the process of identifying and rectifying errors, inconsistencies, and inaccuracies in a dataset. Data cleaning is a crucial part of the broader data preprocessing process. It primarily focuses on improving the **quality** of the data, making sure that it is clean, reliable, and free from errors or inconsistencies.

Examples of Data Cleaning Tasks

- Handling missing values (imputing or removing them).
- Correcting data entry errors (e.g., typos or incorrect entries).
- Dealing with outliers and anomalies.
- Removing duplicate records.
- Standardizing formats (e.g., consistent date formats, casing in text).

DATA CLEANING



Source: https://www.clicdata.com/blog/data-cleaning-the-five-essential-steps/

DATA CLEANING

Missing and Incomplete Data

CUSTOMER	ZIPCODE	CITY
A10001	85003	PHOENIX
A10002		
A10003	10048	

CUSTOMER	ZIPCODE	CITY
A10001	85003	PHOENIX
A10002		
A10003	10048	NEW YORK

Missing data — the data can be completed with sales intelligence, or through data brokers.

Incomplete data — the missing data can be completed based on other data points (ZIP code)

Source: https://www.clicdata.com/blog/data-cleaning-the-five-essential-steps/

DATA TRANSFORMATION

Data Transformation is a key step in preparing raw data for analysis or machine learning. It involves converting data into a format that is suitable for the specific tasks or algorithms being used. Data transformation can involve scaling, encoding, normalizing, or creating new features, depending on the nature of the data and the goals of the analysis.

Data Transformation Techniques

- ❑ **Scaling**
- ❑ **Normalization**
- ❑ **Encoding Categorical Variables**
- ❑ **Binning**
- ❑ **Logarithmic Transformation**
- ❑ **Polynomial Feature Mapping**
- ❑ **Feature Engineering**
- ❑ **Clipping and Winsorizing**
- ❑ **Discretization**
- ❑ **Frequency Domain Transformations**

SCALING & NORMALIZATION

Scaling Ensures that features have the same scale, which is important for **distance-based algorithms** (e.g., k-NN, SVM) and **gradient-based optimization methods**. It adjusts the range of the data to a **fixed interval**, often ensuring that features with larger ranges don't dominate the learning process.

Normalization adjusts the magnitude of data without distorting differences in the range of values. It adjusts the data so that the **distribution of features is uniform**, often ensuring the data is well-conditioned for distance-based algorithms.

Scaling Techniques

- ❑ Min-Max Scaling
- ❑ MaxAbs Scaling
- ❑ Standardization (Z-Score)

Normalization Techniques

- ❑ L_2 Normalization
- ❑ L_1 Normalization

SCALING: COMMON TECHNIQUES

- **Min-Max Scaling:** Rescales data to a range of **[0, 1]** or **[-1, 1]**.

$$\text{Scaled Feature, } X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

- **Standardization (Z-score Normalization):** Centers data around the mean with a unit standard deviation.

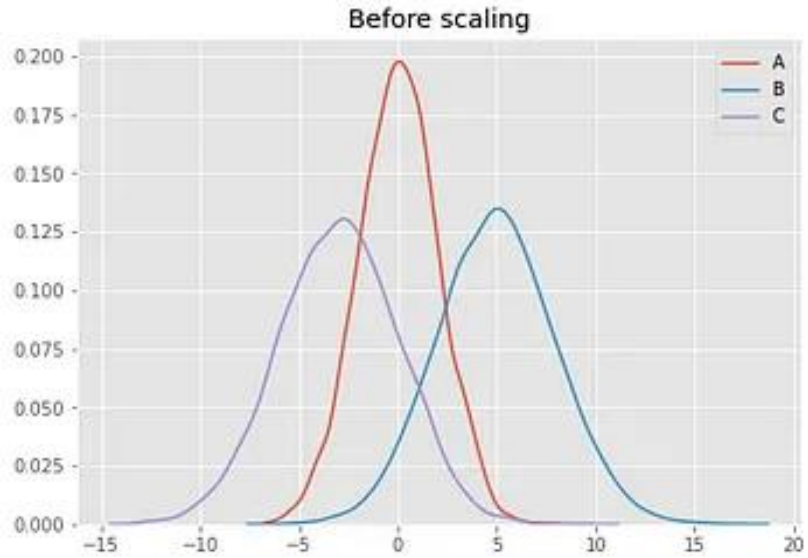
$$\text{Standardized Feature, } X' = \frac{X - \mu}{\sigma}$$

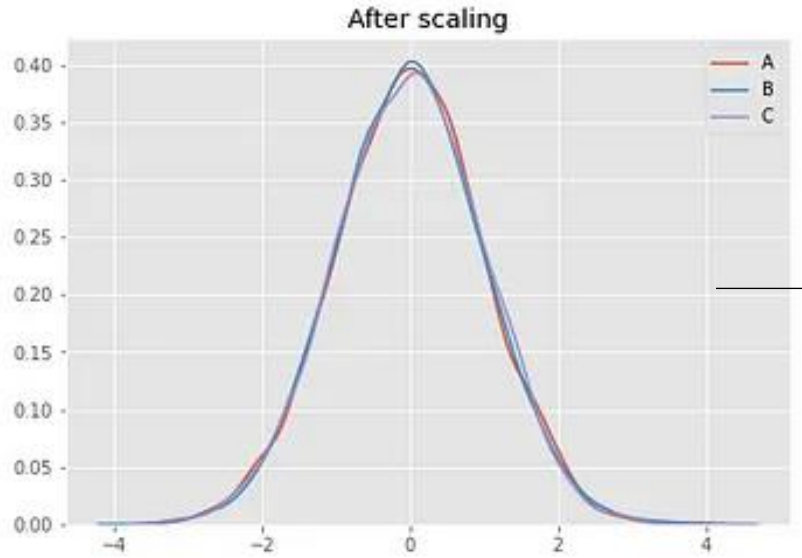
where, **μ = mean** & **σ = Standard Deviation**

- **MaxAbs Scaling:** Scales by the absolute maximum value of each feature to the range **[-1, 1].**

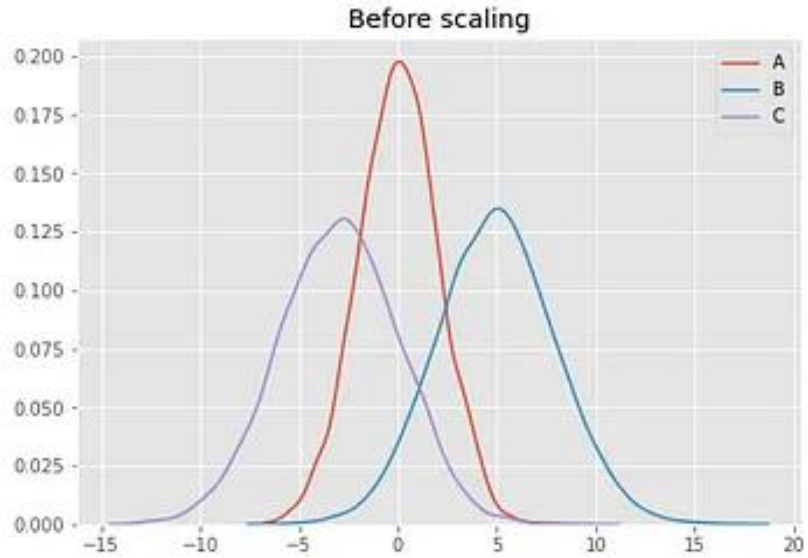
$$\text{Scaled Feature, } X' = \frac{X}{|X_{\max}|}$$

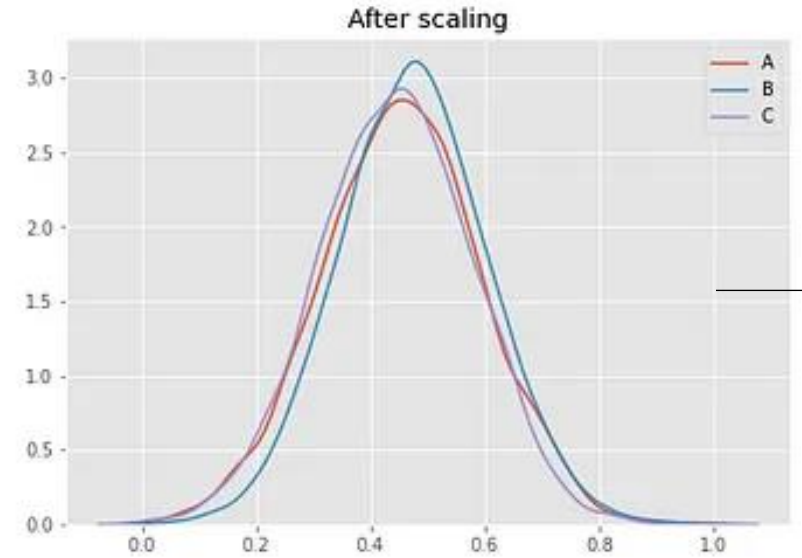
SCALING





→ **Standardization**





→ **MinMax Scaling**

Source: https://towardsdatascience.com/preprocessing-data-feature-scaling-cc28c508e8af

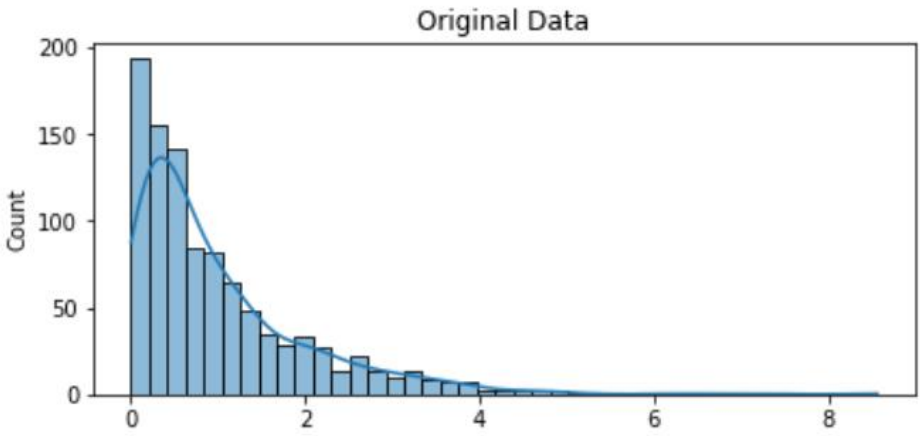
NORMALIZATION: COMMON TECHNIQUES

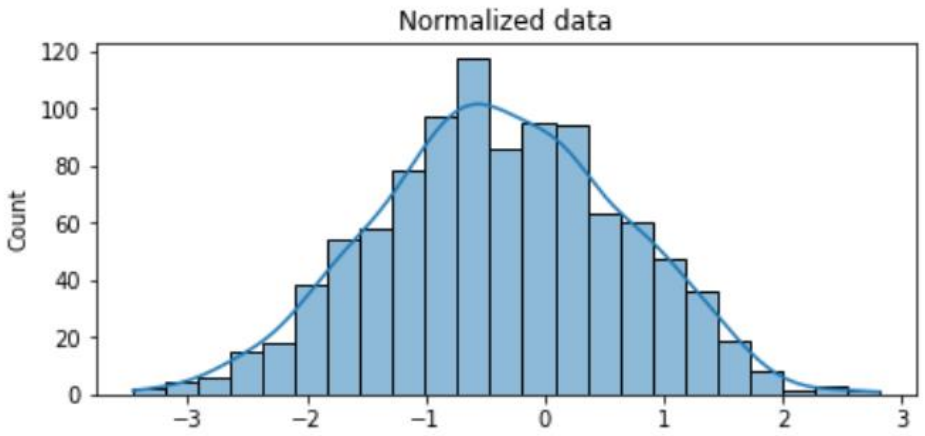
- **L_2 Normalization:** Scales values so that the sum of the squared elements is 1.

$$\text{Normalized Feature, } X' = \frac{X}{\sqrt{\sum X^2}}$$

- **L_1 Normalization:** Scales values so that the sum of the squared elements is 1.

$$X' = \frac{X}{\sum |X|}$$





CATEGORICAL VARIABLES: LABEL ENCODING

- **Label Encoding:** Assigns a unique integer to each category. Used for both nominal (where order doesn't matter) and ordinal data (where order matters). This method is Simple and does not increase dimensionality. However, models might misinterpret the values as having a mathematical relationship (e.g., 2 > 1 > 0) in case of nominal categories.

Original Data

Team	Points
A	25
A	12
B	15
B	14
B	19
B	23
C	25
C	29

Label Encoded Data

Team	Points
0	25
0	12
1	15
1	14
1	19
1	23
2	25
2	29

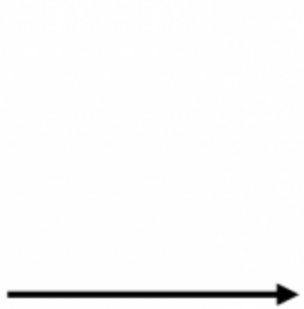
Source: https://www.statology.org/label-encoding-vs-one-hot-encoding/

CATEGORICAL VARIABLES: ONE-HOT ENCODING

One-Hot Encoding: Converts each category into a new binary column (0 or 1). Used for nominal data with no inherent order. Each unique category is converted into a new binary feature. If there are n categories, n binary features are created, with one feature "hot" (1) and the rest "cold" (0). Increases dimensionality.

Original Data

Team	Points
A	25
A	12
B	15
B	14
B	19
B	23
C	25
C	29



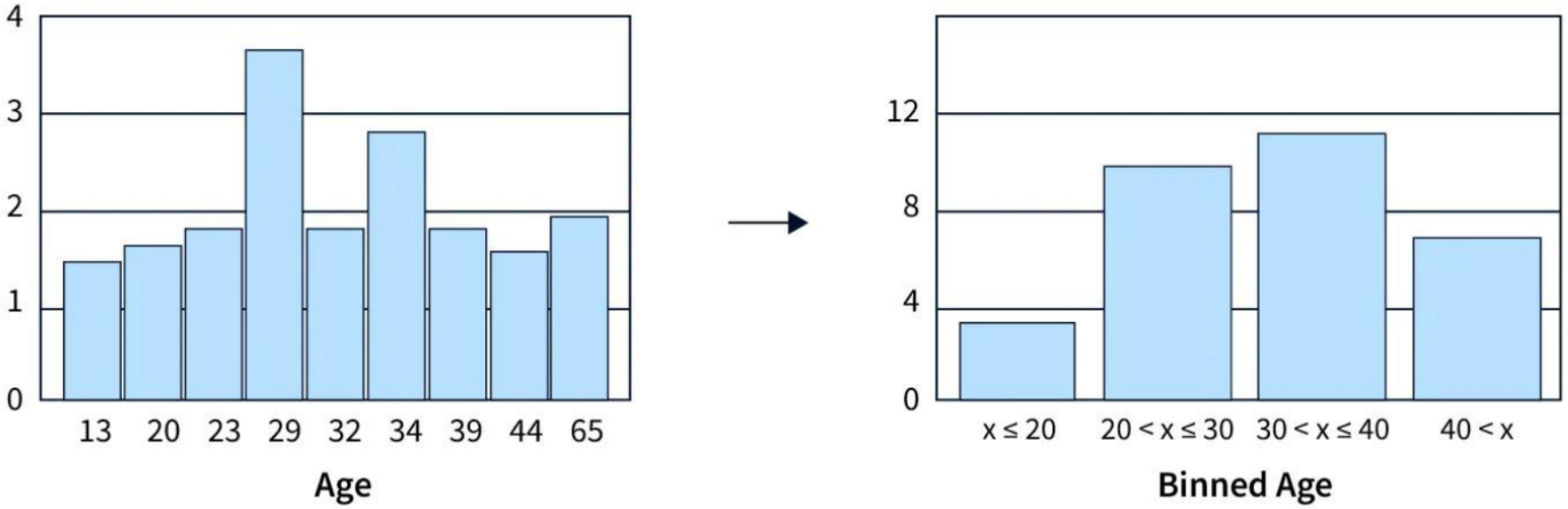
One-Hot Encoded Data

Team_A	Team_B	Team_C	Points
1	0	0	25
1	0	0	12
0	1	0	15
0	1	0	14
0	1	0	19
0	1	0	23
0	0	1	25
0	0	1	29

Source: https://www.statology.org/label-encoding-vs-one-hot-encoding/

DISCRETIZATION: BINNING

Binning (**Discretization**) is a technique used in data preprocessing where continuous numerical variables are converted into discrete categories or "bins." This process simplifies the model by reducing the impact of minor observation variations and transforming continuous data into categorical intervals. It helps to reduce the effects of small fluctuations in the data and to group data into meaningful intervals for easier analysis or interpretation. It can also improve model performance in certain cases, especially when dealing with noisy data.



Source: https://www.scaler.com/topics/binning-in-data-mining/

BINNING (TO RESIZE IMAGES)

Binning Off

Each pixel has it's own value of brightness. (value based on the image)

2	3	2	2	3	2
2	3	5	5	3	2
2	3	6	6	2	1
2	3	6	6	2	1
3	8	8	6	4	2
3	6	5	5	5	5

10	14	10
10	24	6
20	24	16

Binning On

A 2*2 matrix of 4 pixels are grouped together and their individual values are combined. This combined value is substituted for original value.

Source: https://www.scaler.com/topics/binning-in-data-mining/

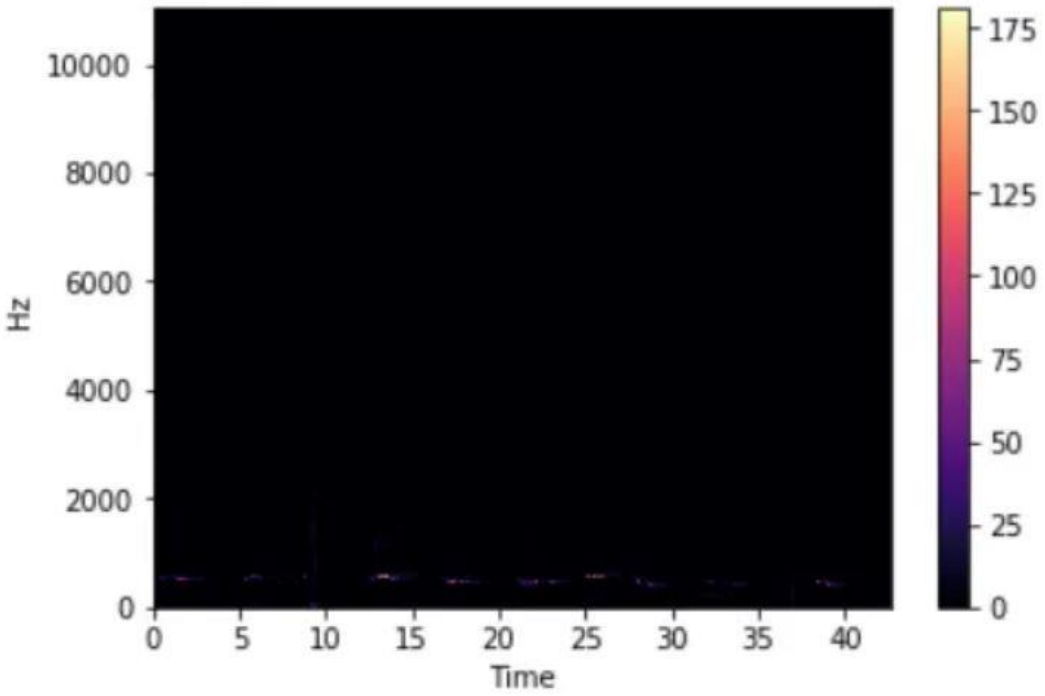
LOGARITHMIC TRANSFORMATION

Logarithmic Transformation is a data transformation technique where the logarithm of the data values is taken to compress the range of values. The idea is to reduce the effect of large values while preserving smaller ones, making the distribution of data more manageable and enhancing certain patterns that may not be apparent in the original scale. It aims to compress the dynamic range of data making data distribution more symmetrical or “normal”. It reduces the impact of extreme values & enhances subtle variations in lower-range values.

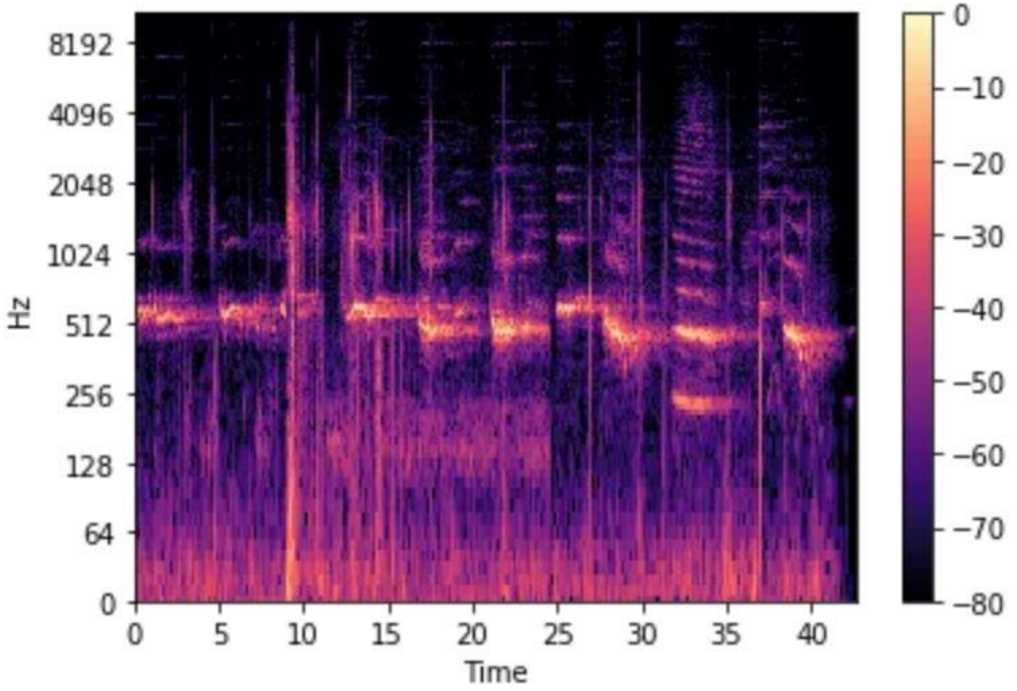
Applications in Audio and Image Processing

- **Audio Compression:** Logarithmic transformation is used in audio compression algorithms to store sound data more efficiently through dynamic range compression.
- **Spectrogram Analysis:** In audio analysis, the logarithmic transformation is often applied to spectrograms (displays frequency over time) to emphasize lower-magnitude frequencies.
- **Medical Imaging:** In medical images (such as MRI scans), log transformation helps in enhancing darker regions, revealing critical details that may not be visible in the raw image.
- **Astronomical Imaging:** Log transformations are used to enhance the contrast in astronomical images, making faint stars and galaxies visible against a bright background.

LOGARITHMIC TRANSFORMATION (SPECTROGRAMS)



Before Logarithmic Transformation



After Logarithmic Transformation

FREQUENCY DOMAIN TRANSFORMATIONS

Frequency Domain Transformations are techniques used to analyze and process signals (such as audio, images, or time-series data) by converting them **from the time** or **spatial domain** into **the frequency domain**. *This allows machine learning models to capture patterns and features that are not easily visible in the original domain*. Frequency domain transformations are particularly useful when the data exhibits periodic or oscillatory behavior, which is often missed by models that only operate in the time or spatial domain.

Key Frequency Domain Transformations

- ❑ **Fourier Transform (FT)**
- ❑ **Fast Fourier Transform (FFT)**
- ❑ **Short-Time Fourier Transform (STFT)**
- ❑ **Wavelet Transform**
- ❑ **Discrete Cosine Transform (DCT)**
- ❑ **Mel-Frequency Cepstral Coefficients (MFCC)**

FREQUENCY DOMAIN TRANSFORMATIONS

Importance in Machine Learning

- **Feature Extraction**: Frequency domain transformations help extract meaningful features from raw signals or data. For instance, in audio data, speech recognition models benefit from analyzing the frequency components rather than just raw sound waves. Similarly, in image processing, frequency features can reveal texture and periodic structures.
- **Noise Reduction**: Transforming data into the frequency domain can help remove noise by filtering out high-frequency or low-frequency components that do not contribute meaningfully to the signal.
- **Dimensionality Reduction**: Techniques like **DCT** and **MFCC** reduce the dimensionality of data by transforming it into a smaller set of frequency-domain features, improving the efficiency of machine learning models.
- **Analyzing Periodic Behavior**: Frequency domain transformations allow models to detect cycles, trends, and anomalies that would be difficult to spot in the time domain.

FOURIER TRANSFORM (FT)

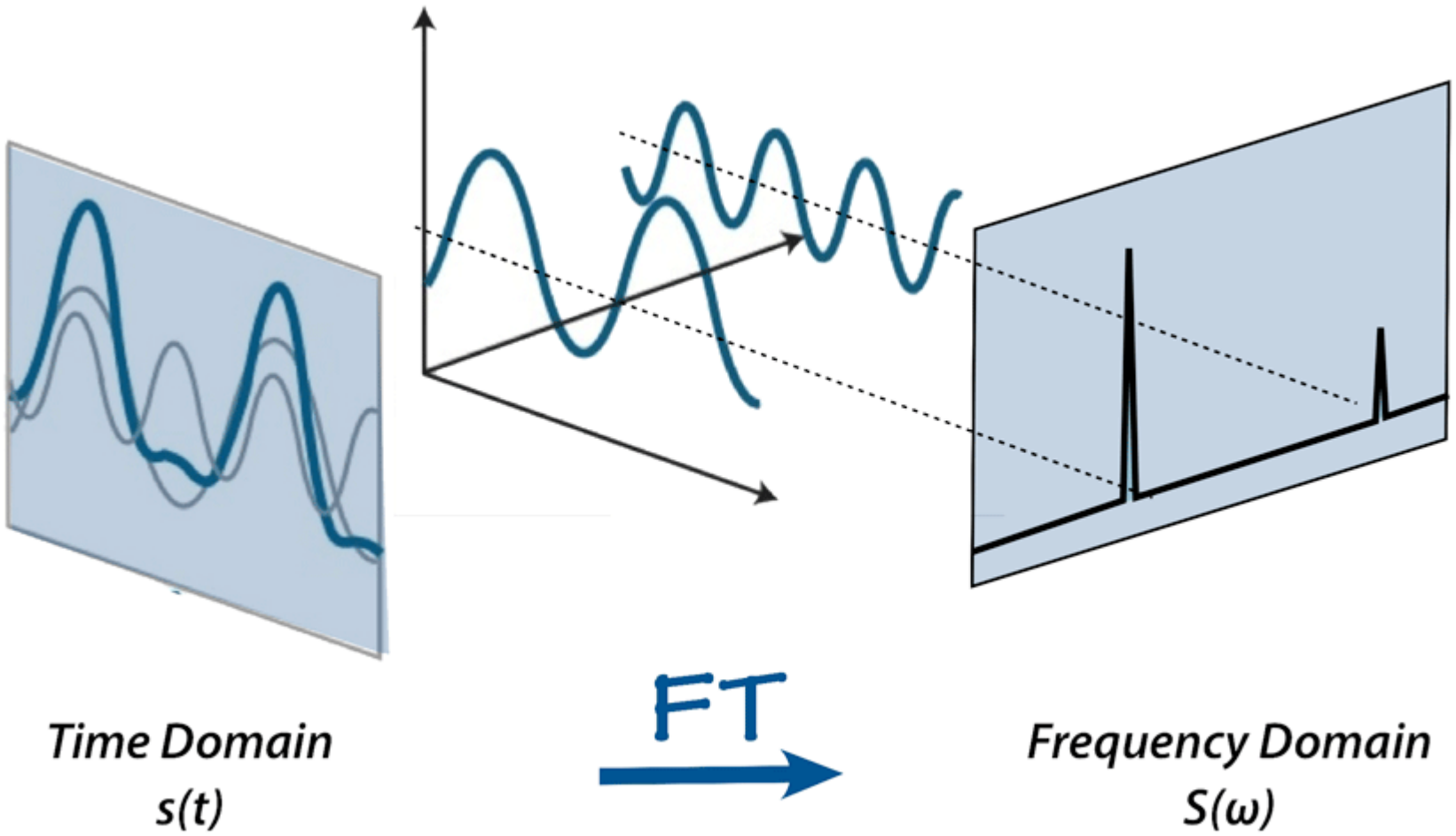
The **Fourier Transform** is a mathematical technique that transforms a signal from the time (or spatial) domain to the frequency domain. **It represents the signal as a sum of sine and cosine functions with different frequencies. It states that,**

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\Pi ft} \, dt$$

Where $x(t)$ is the original signal in the **time domain**, and $X(f)$ is its representation in the **frequency domain**.

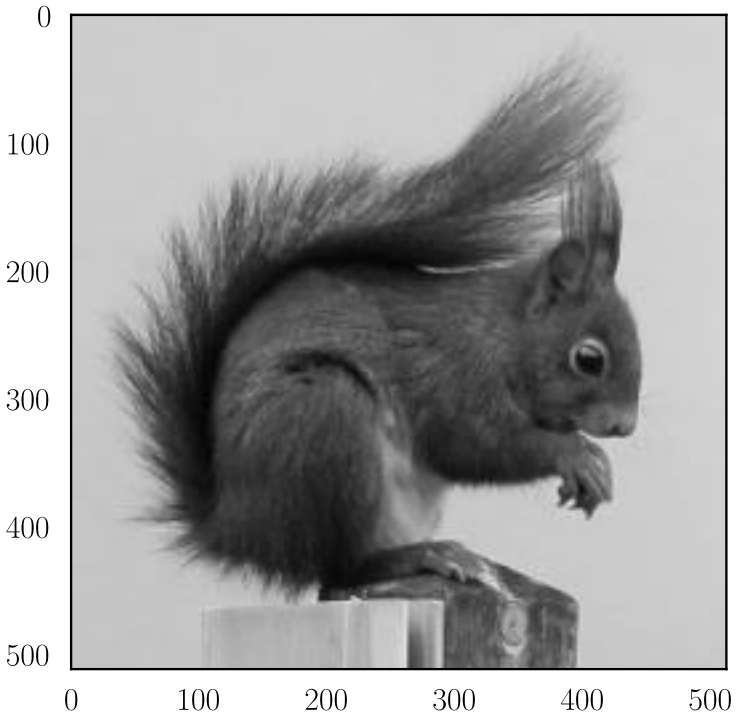
Fourier transforms are often used in audio processing (e.g., speech recognition) and image processing (e.g., texture analysis). The frequency components of the signal help reveal periodic patterns that can be crucial features for models. **FFT**, **STFT**, and **DCT** are different optimized versions of Fourier Transform that are essential in machine learning for extracting meaningful patterns, reducing noise, and providing a more compact and informative representation of data.

FOURIER TRANSFORM (FT)

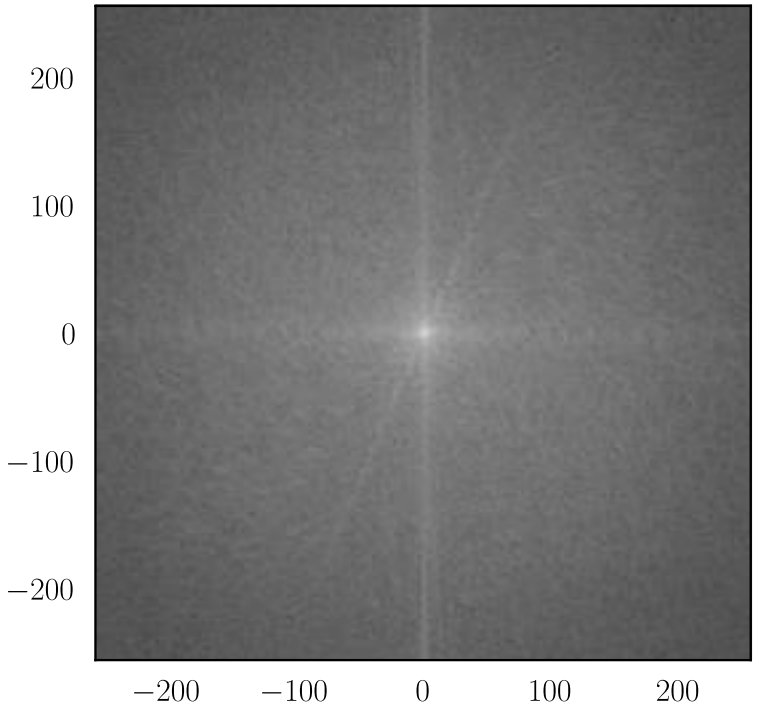


Source: https://medium.com/the-modern-scientist/the-fourier-transform-and-its-application-in-machine-learning-edecfac4133c

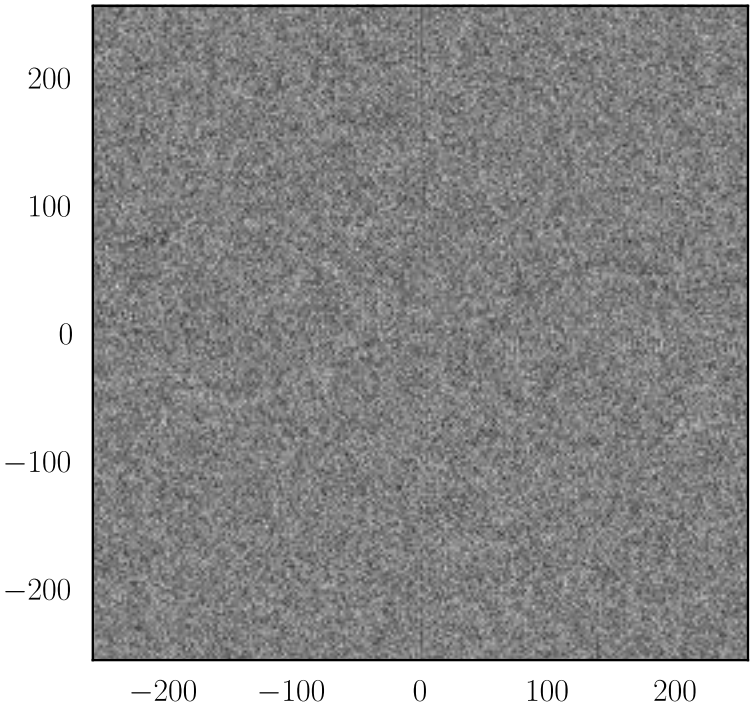
FOURIER TRANSFORM IN COMPUTER VISION



Image



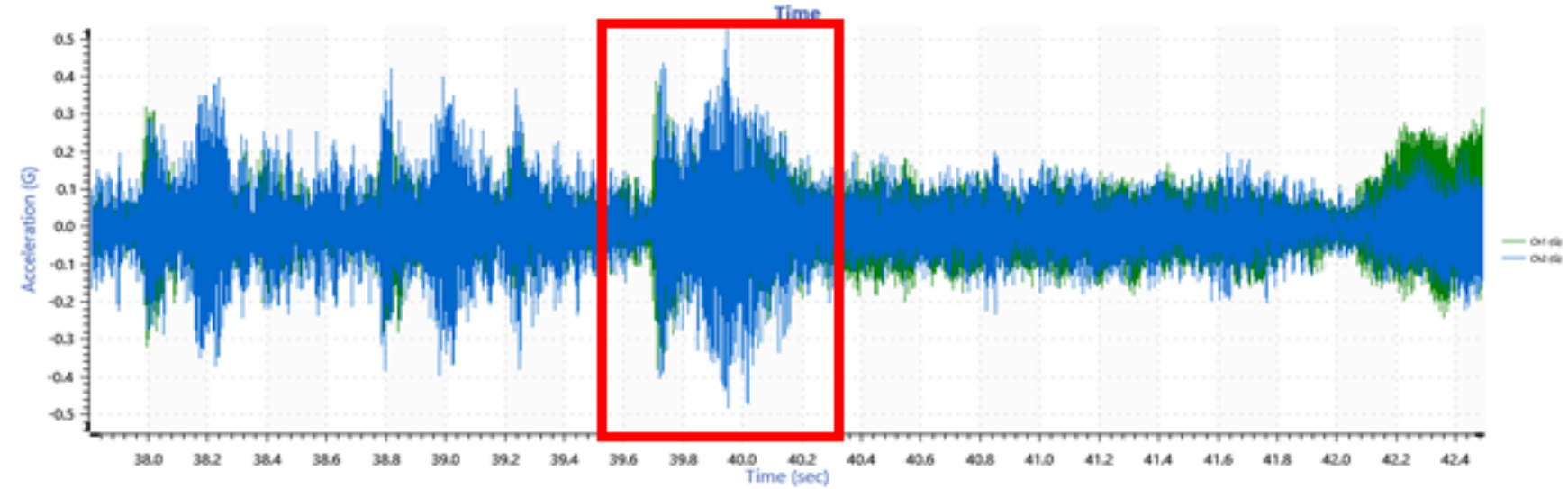
Amplitude of the DFT

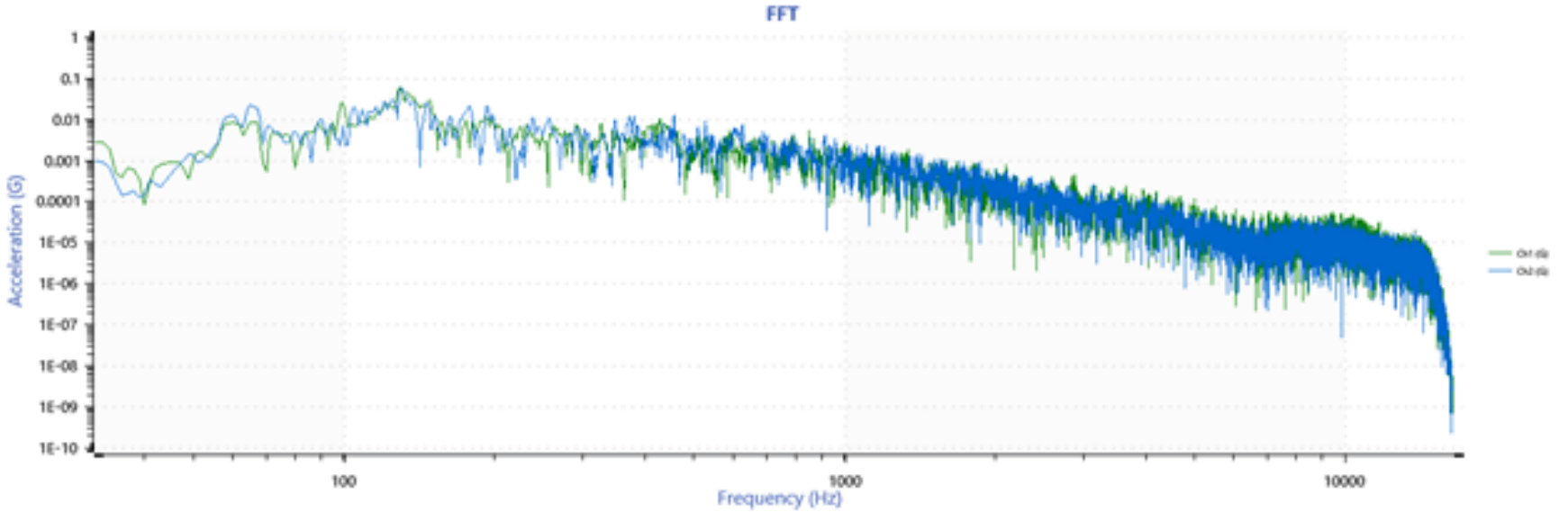


Phase of the DFT

Source: https://vincmazet.github.io/bip/filtering/fourier.html

FAST FOURIER TRANSFORM (FFT)





Time Domain Signal

FFT

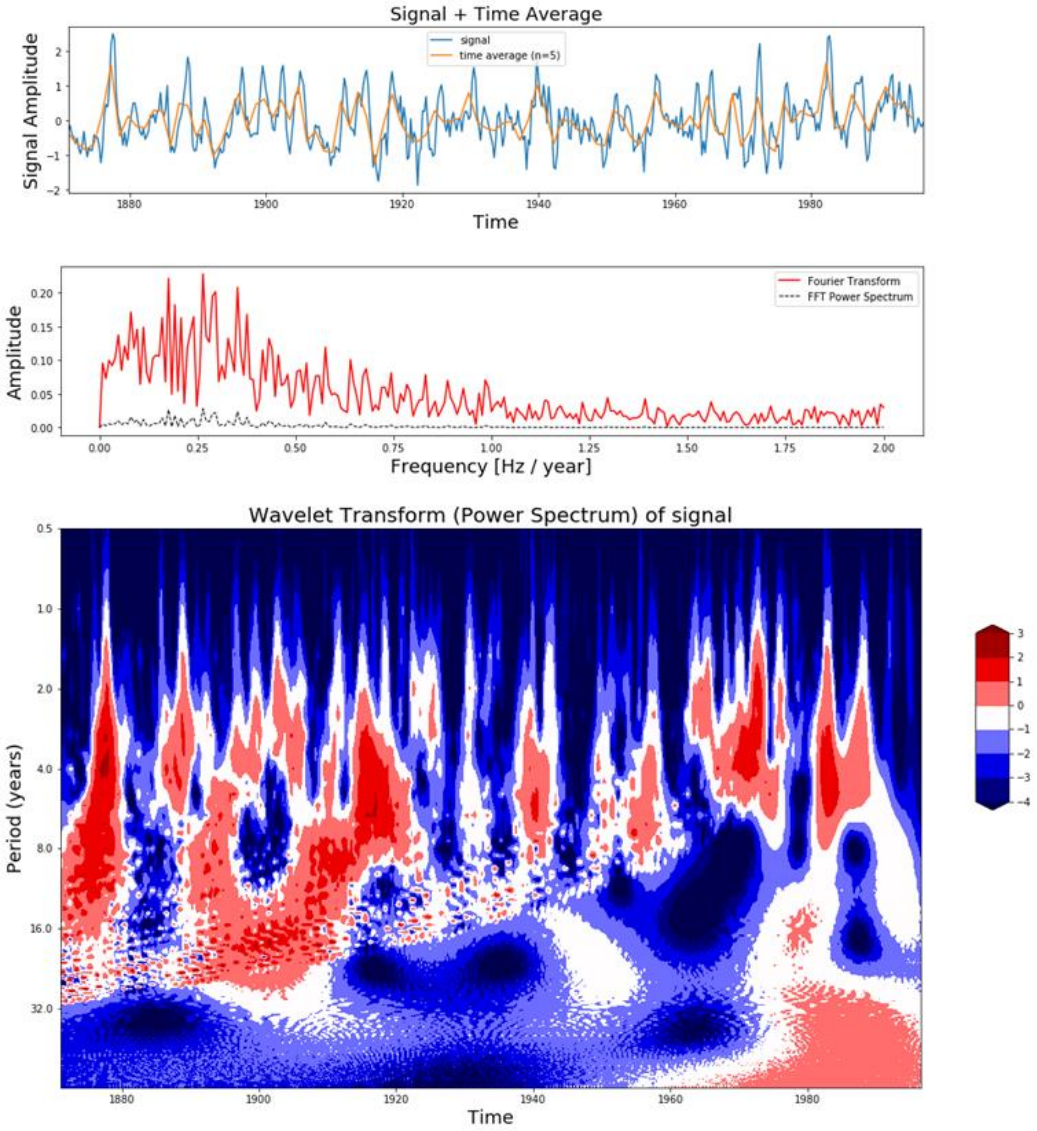
Frequency Domain Signal

Source: https://vru.vibrationresearch.com/lesson/time-vs-frequency-domain/

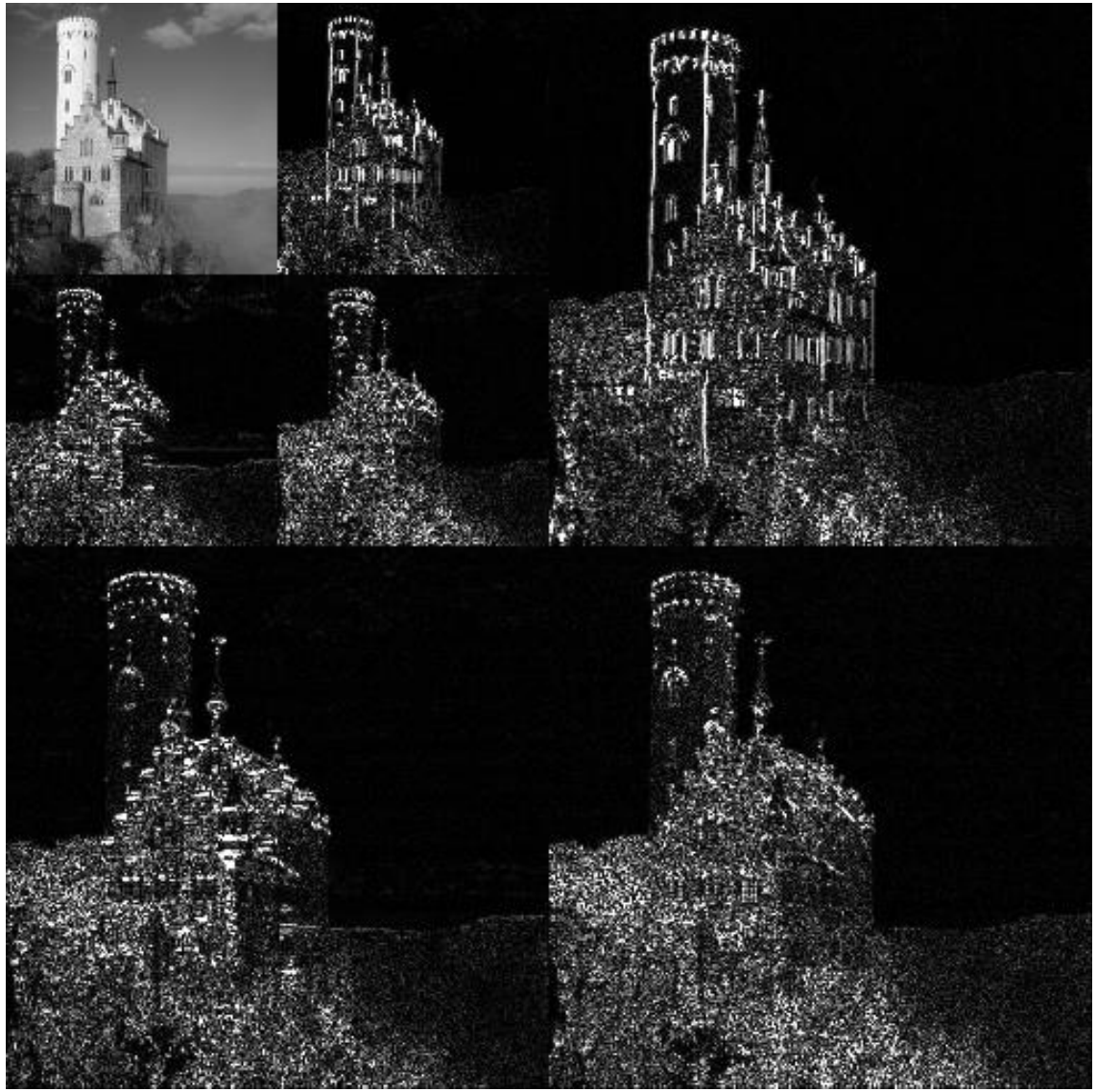
WAVELET TRANSFORM

The **Wavelet Transform** breaks down a signal into components at different scales, providing both time and frequency localization. Unlike the Fourier Transform, wavelets can capture both high-frequency and low-frequency components in the signal.

Wavelet transforms are used in time-series analysis, image compression, and denoising. They are particularly useful for analyzing **non-stationary data** where the **frequency characteristics change over time**. In speech recognition, wavelet transforms can help isolate phonemes and detect temporal variations. Here we can see in the figure the el-Nino dataset together with its time average, in the middle figure the Fourier Transform, and at the bottom figure the scalogram produced by the Continuous Wavelet Transform.



WAVELET TRANSFORM

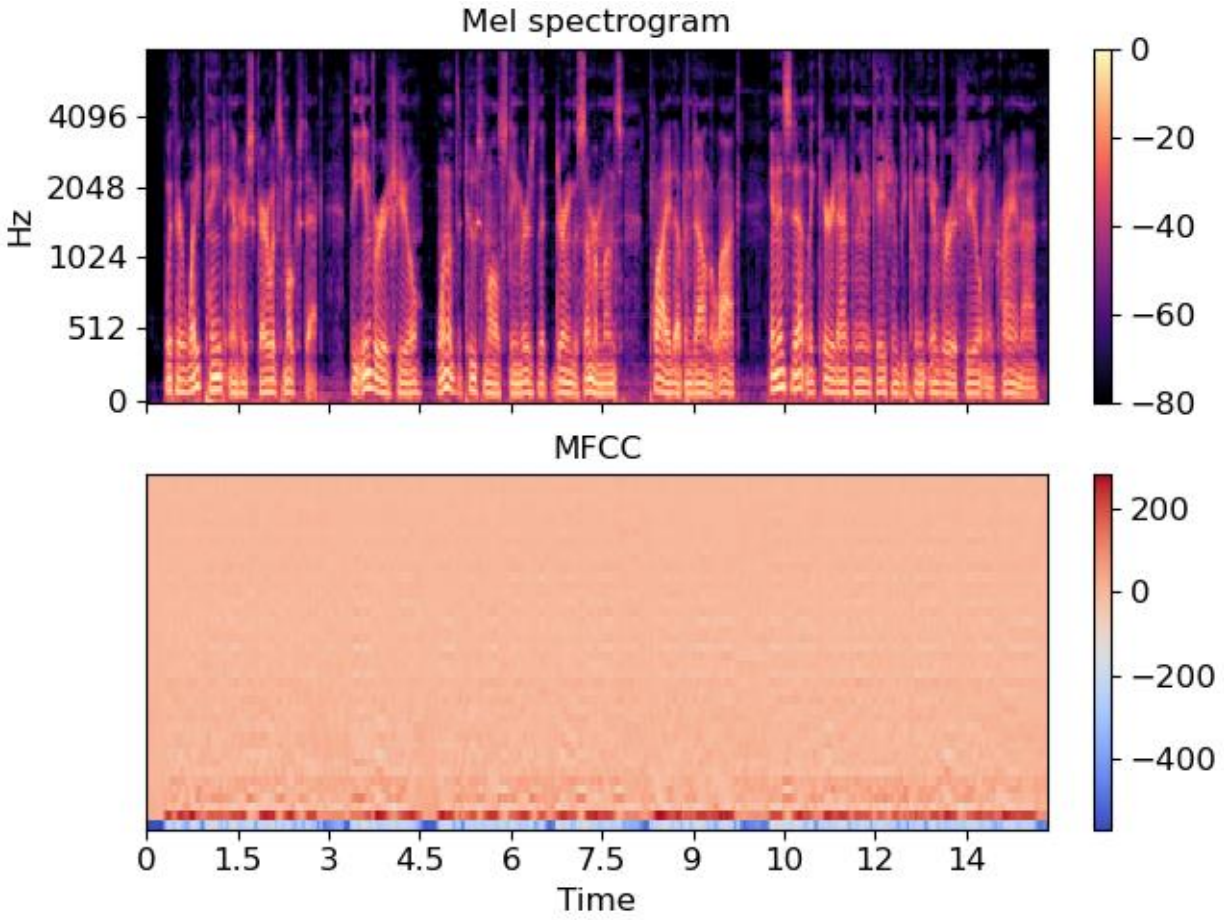


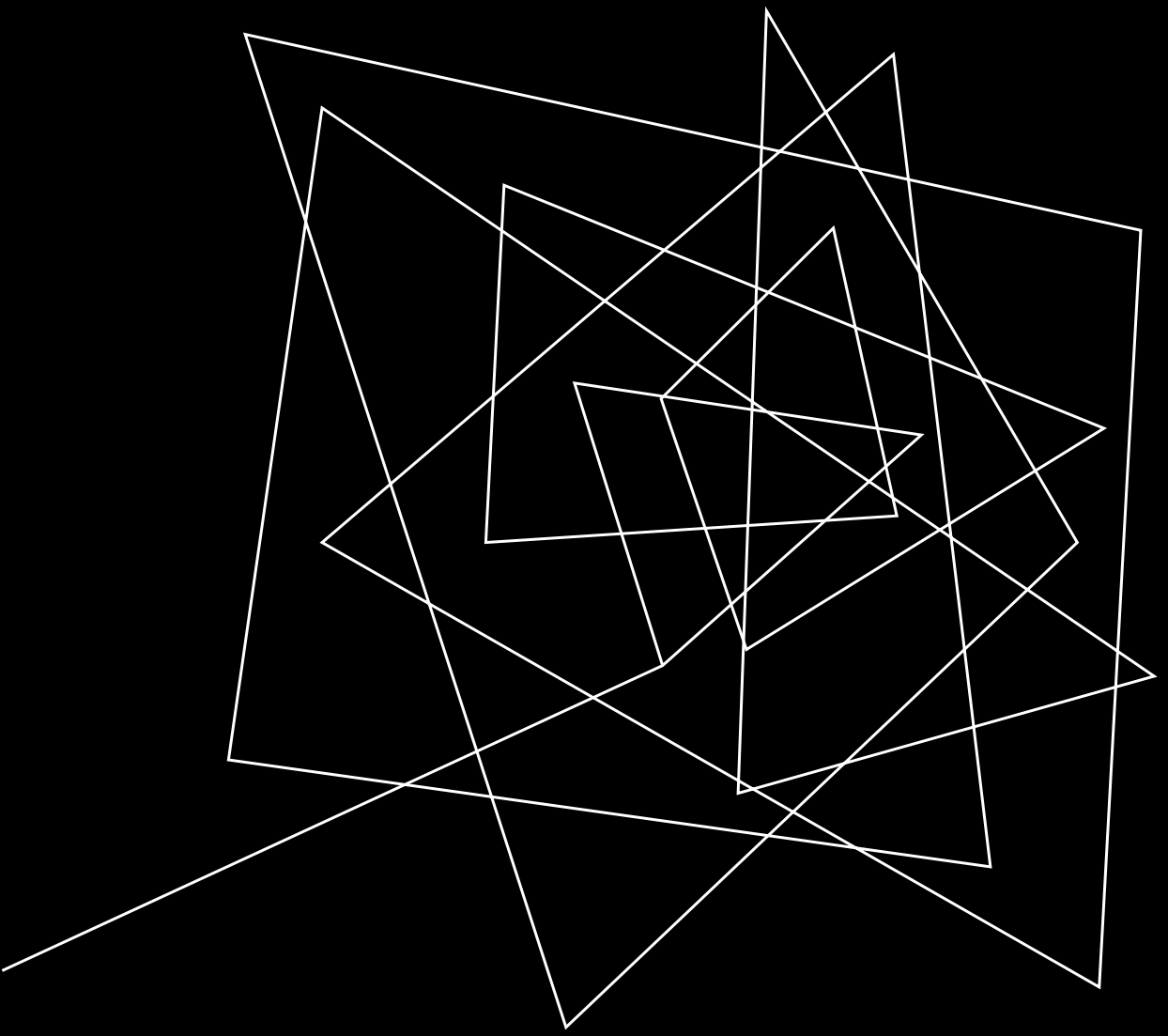
Wavelet Transform applied with different settings on the top left image. Clearly, lots of details can be seen in the bottom left, while the original image has almost no details because of the sky. Moreover, the edges of the castle are really sharp & can be clearly seen in the wavelet transform.

MEL-FREQUENCY CEPSTRAL COEFFICIENTS (MFCC)

MFCC is a transformation that converts an audio signal into a set of coefficients that approximate the way humans perceive sound, especially in speech. The audio signal is first passed through a filter bank that mimics the human ear's response to different frequencies, and then a log transform and DCT are applied.

MFCCs are the most used feature in speech and audio recognition tasks. They provide a compact representation of the audio signal, making it easier for machine learning algorithms to identify patterns in spoken words or musical notes.





DATA PARTITIONING

DATA PARTITIONING: WHY & HOW?

Data Partitioning is the process of splitting a dataset into distinct subsets, which are typically used for **training**, **validating**, and **testing** machine learning models. It ensures that models are evaluated on unseen data and helps prevent overfitting. Partitioning is crucial for assessing a model's performance and generalization ability beyond the training data.

Why?

Partitioning ensures the model’s performance is tested on data that it hasn't seen before, providing a more realistic evaluation. Partitioning helps evaluate whether the model generalizes well to unseen data. Also, a **Validation Set allows for tuning hyperparameters** without biasing the model performance.

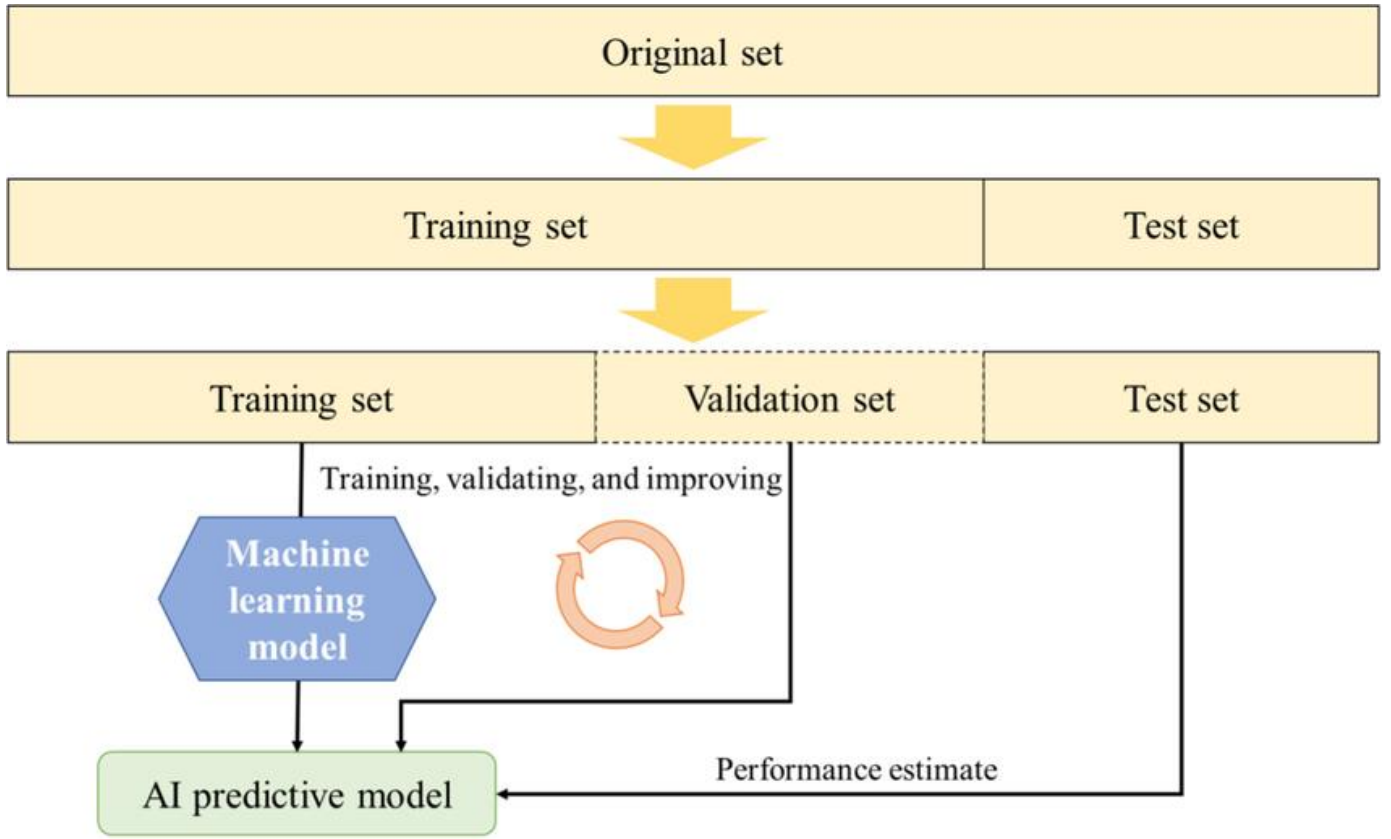
Why a separate Test Set?

A separate validation set is often used for tuning hyperparameters during training. If the test set were used for tuning, it could bias the results, and the model might effectively "cheat" by overfitting to the test data. A separate test set avoids this issue of **Data leakage** and ensures the final evaluation is on truly unseen data. Without a separate test set, the model could inadvertently gain information about the test data during training, leading to artificially high-performance scores.

DATA PARTITIONING: HOLDOUT METHOD

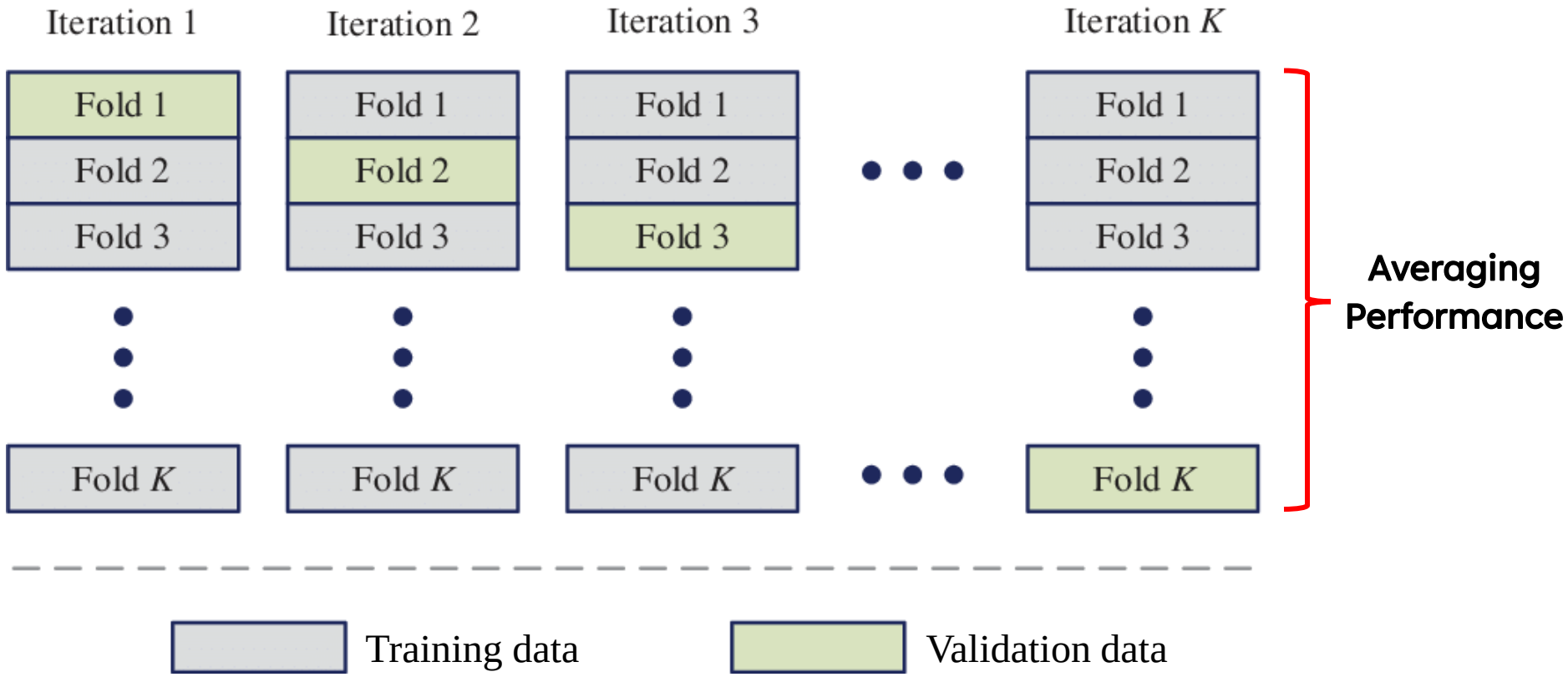
In this method, the dataset is **split into two or three parts** - commonly a **Training set** (used to fit the model), a **Validation set** (for tuning hyperparameters or model selection), and a **Test set** (for evaluating final performance).

Typical Splits: ~60% training, ~20% validation, ~20% testing.

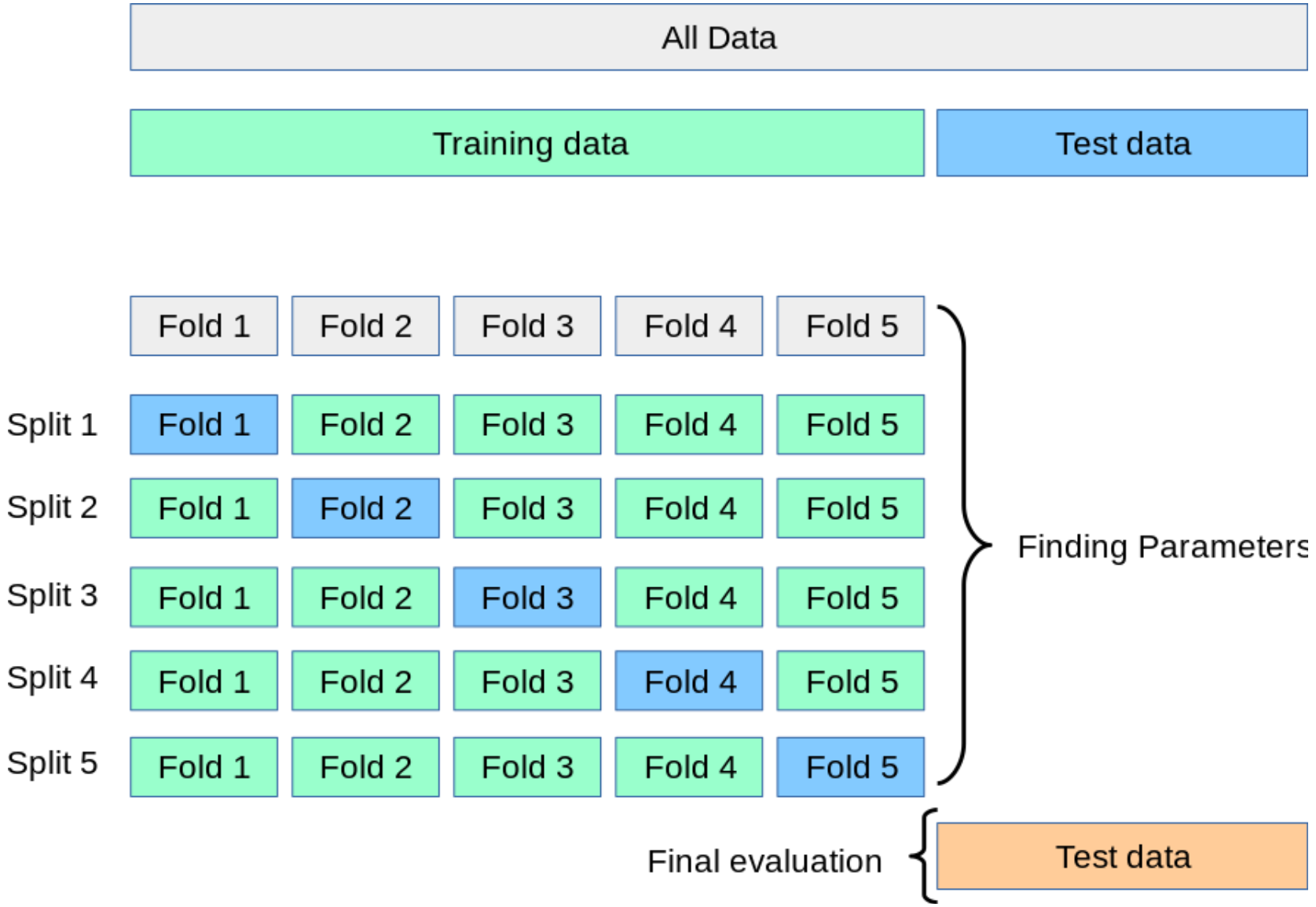


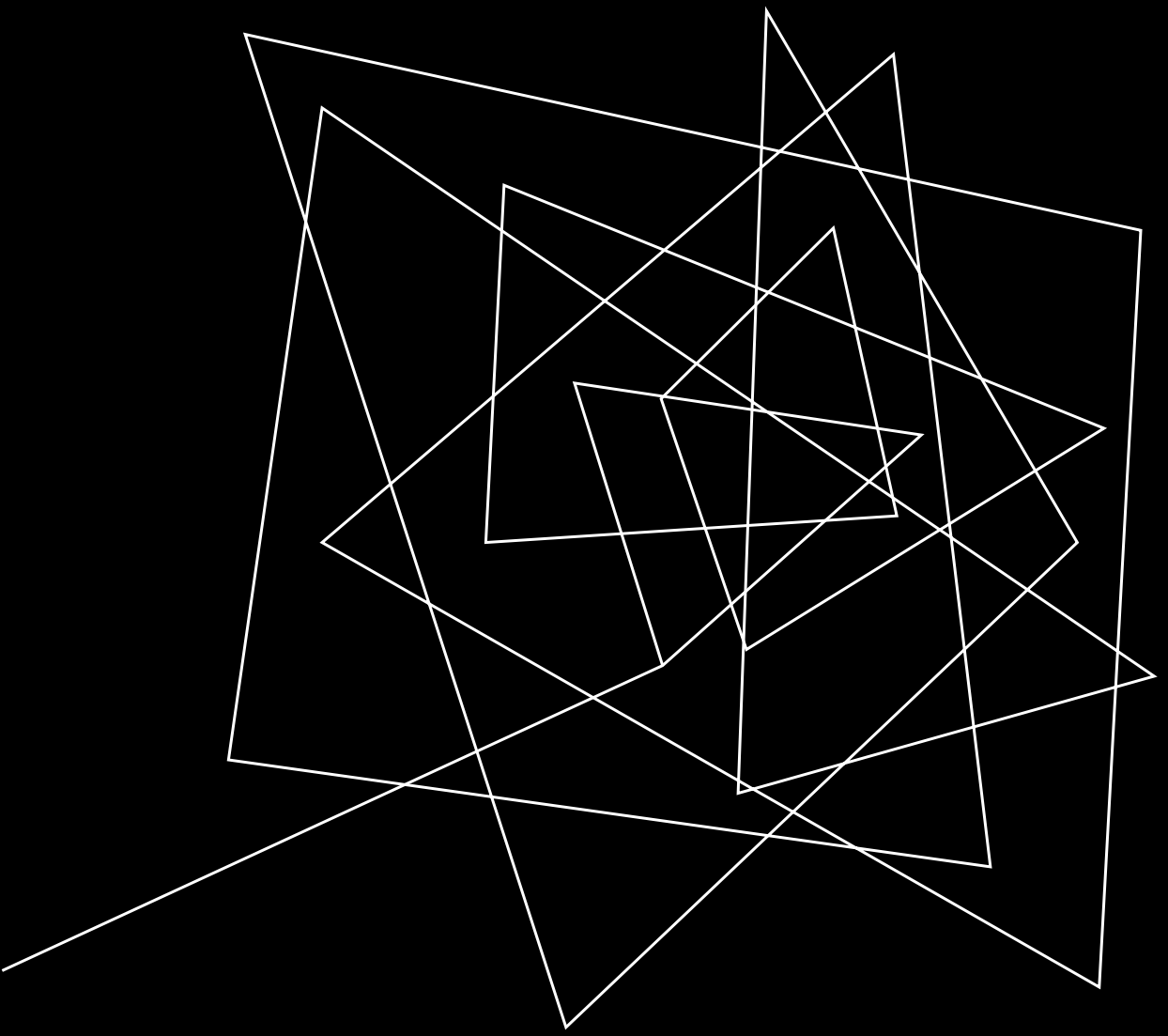
DATA PARTITIONING: K-FOLD CROSS VALIDATION

The dataset is divided into k equal-sized folds. The model is trained on $k-1$ folds and tested on the remaining fold. This process is repeated k times, each time using a different fold for testing, and the results are averaged. **k-Fold CV** method provides a better estimate of the model's performance, reduces bias, and makes full use of the data. Unlike the Holdout method, the model's performance remains independent of how the data was split.



DATA PARTITIONING: K-FOLD CROSS VALIDATION





DATA AUGMENTATION

DATA AUGMENTATION

Data Augmentation is a technique used in machine learning, especially in deep learning, to artificially expand the size and diversity of a training dataset **by creating modified versions of existing data samples**. This is particularly helpful when dealing with small datasets, as it improves model generalization by exposing it to more varied inputs without the need for collecting new data. Augmentation introduces variability, preventing models from overfitting to specific patterns in the training set. Providing additional, slightly altered data points, augmentation **helps the model generalize better to unseen data**. It teaches the model to recognize objects or patterns despite distortions, noise, or other real-world variations. **Typically, Augmentation must be applied *after* Data Partition, on Training data.**

Common Data Augmentation Techniques

- ❑ **Geometric Transformations**
- ❑ **Color Space Augmentation**
- ❑ **Noise Injection**
- ❑ **Random Cropping**
- ❑ **Cutout/Random Erasing**
- ❑ **MixUp & CutMix**
- ❑ **Time Warping (for Sequential Data)**
- ❑ **Feature Space Augmentation**

GEOMETRIC TRANSFORMATIONS

Original	Flipped vertically
	

Original	Left Rotated
	

GEOMETRIC TRANSFORMATIONS

Original	Rotated and cropped
	

Original	Skew tilt
	

GEOMETRIC TRANSFORMATIONS

Original	Shear
	

Original	Zoom
	

COLOR SPACE AUGMENTATION

Original	Increased Brightness
	

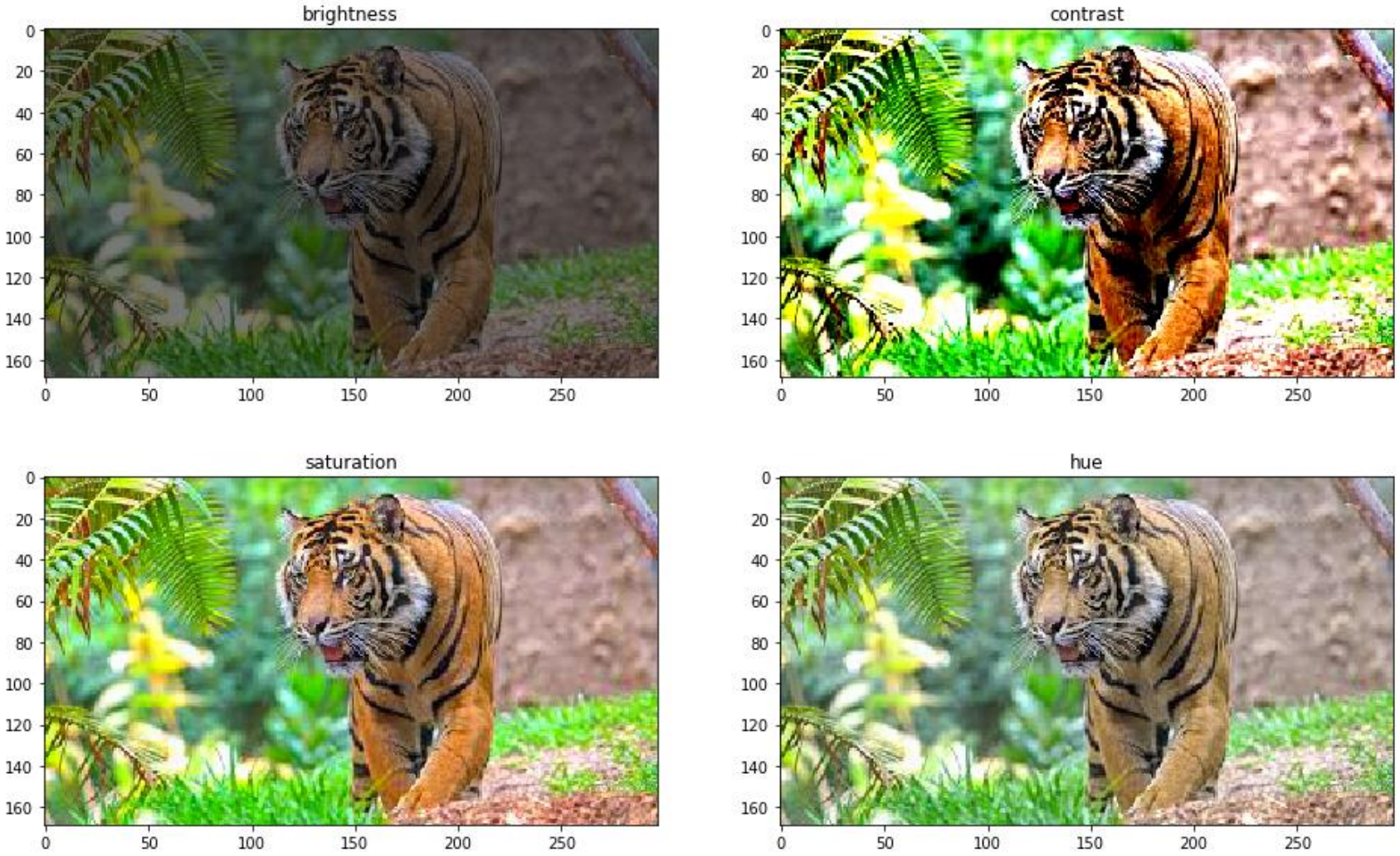
Original	Increased contrast
	

COLOR SPACE AUGMENTATION

Original	Gamma Adjustment
	

Original	HSV Shifted
	

COLOR SPACE AUGMENTATION



Source: https://www.v7labs.com/blog/data-augmentation-guide

NOISE INJECTION

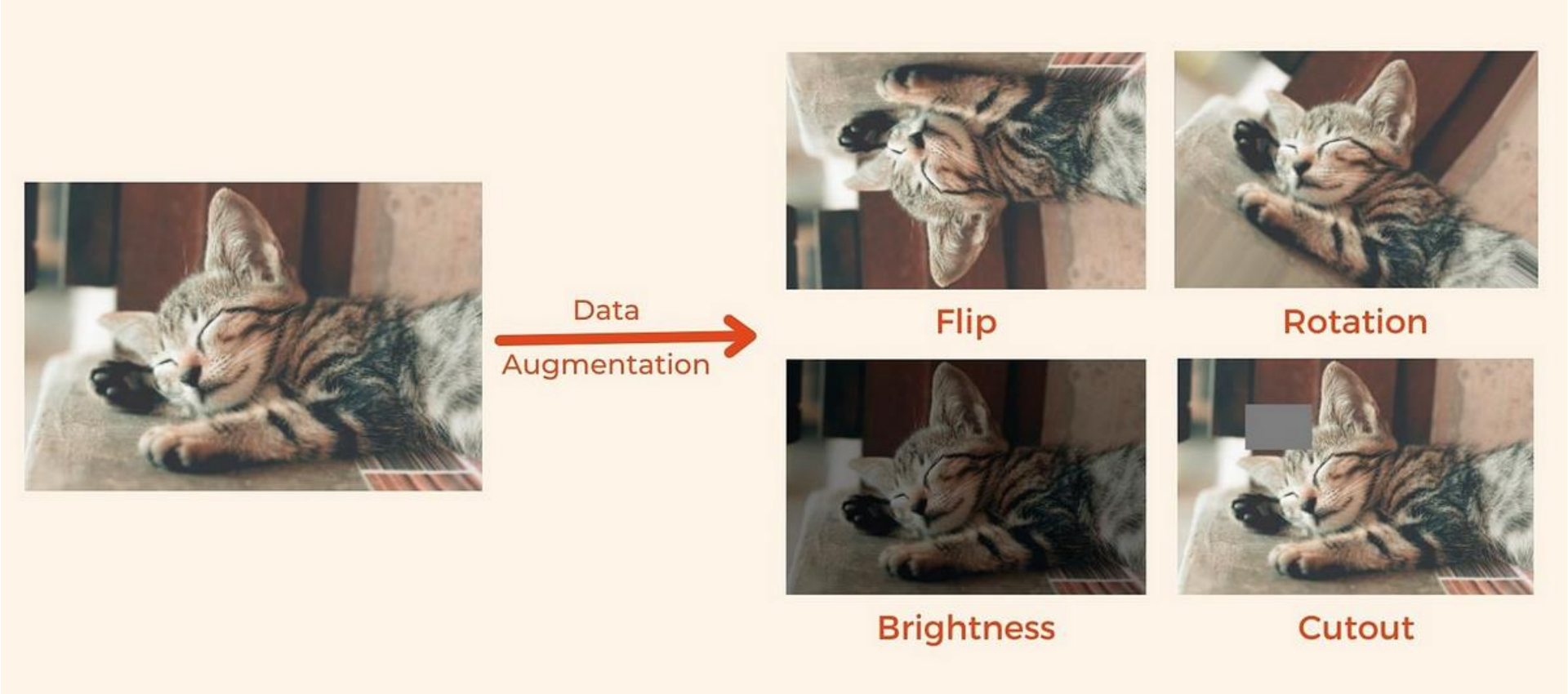
Original	Salt and Pepper noise
	

Original	Rain
	

CUTOUT/RANDOM ERASING

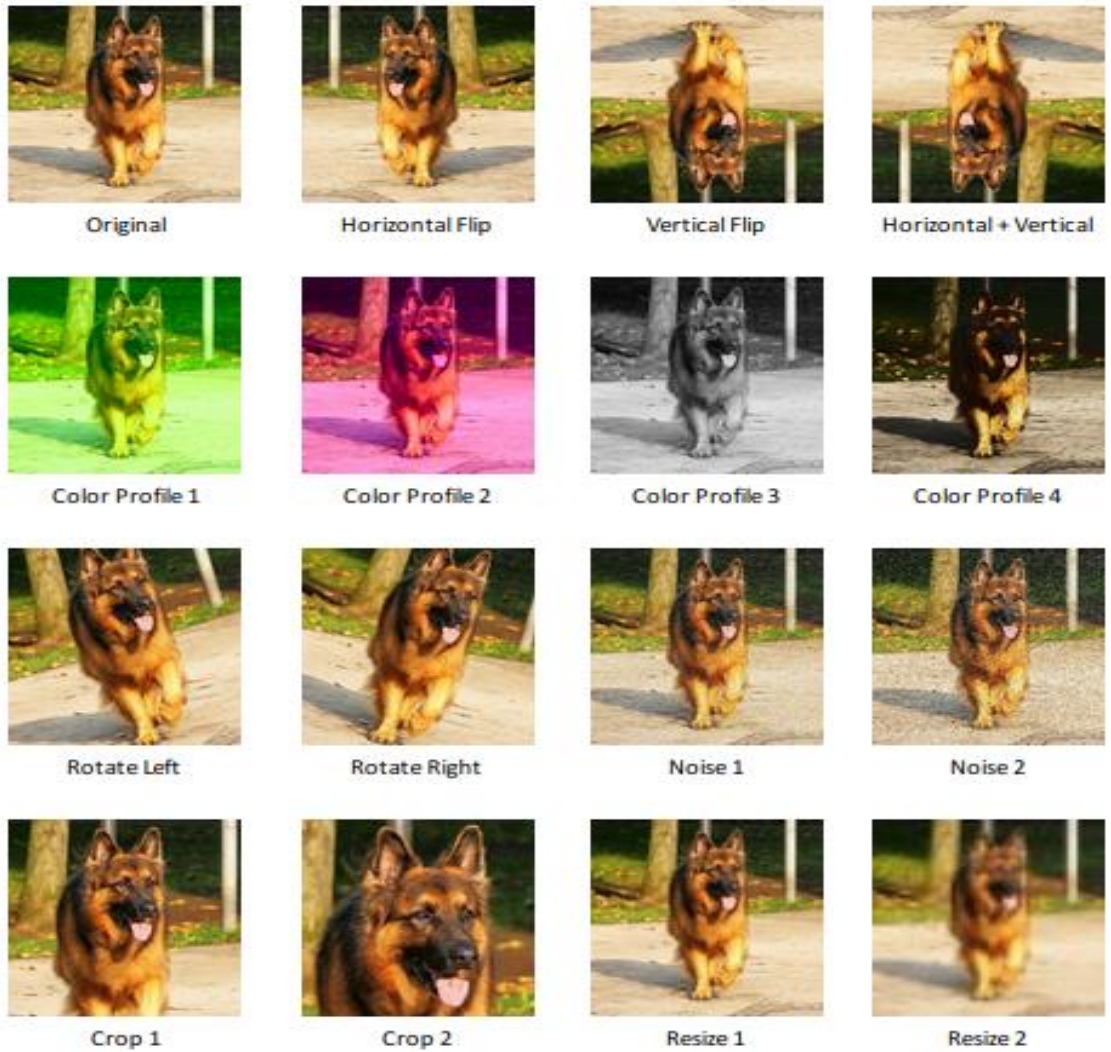
Original	Random Erasing
	

DATA AUGMENTATION TECHNIQUES

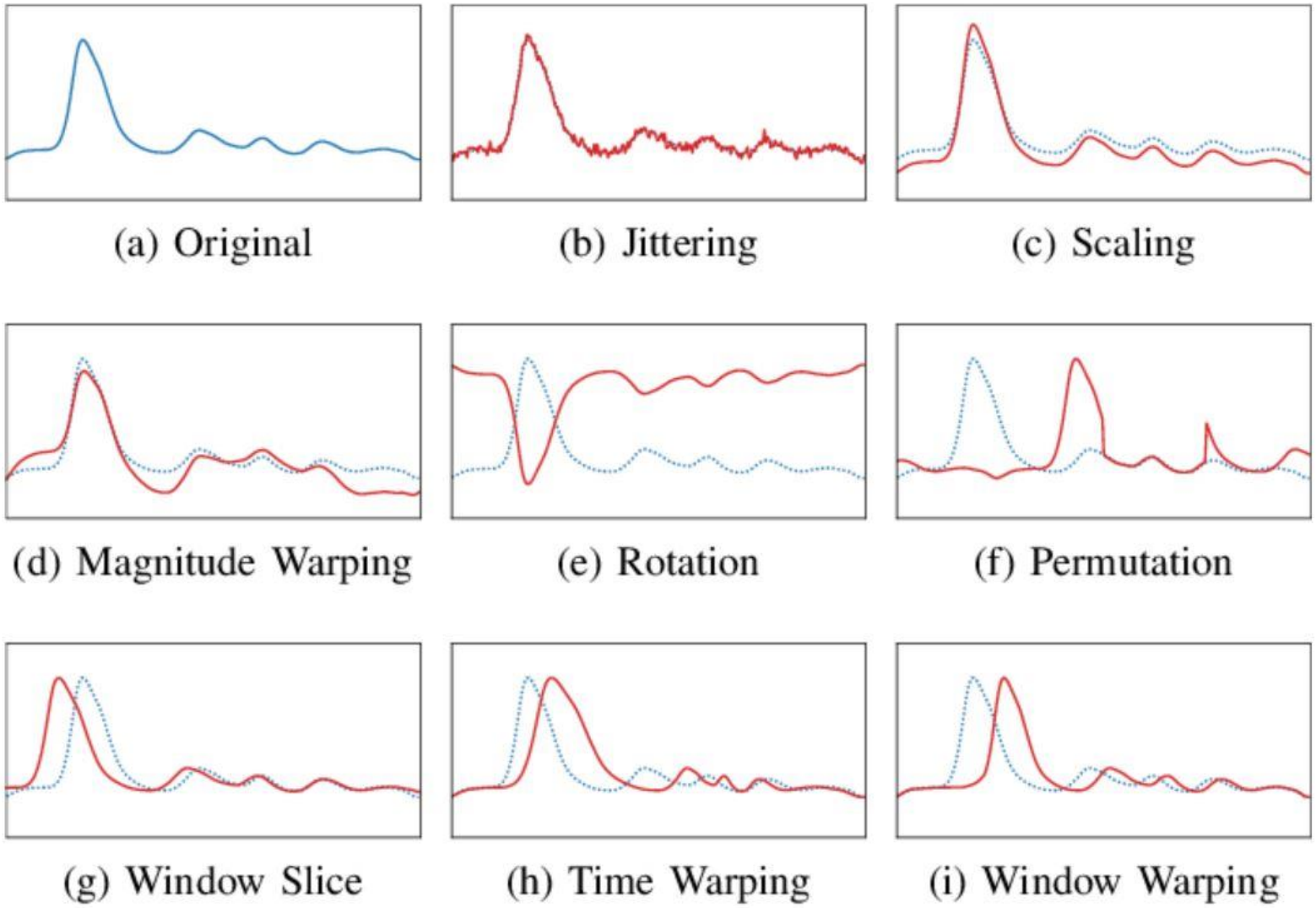


Source: https://just-merwan.medium.com/what-is-data-augmentation-in-computer-vision-6a7d19c53e

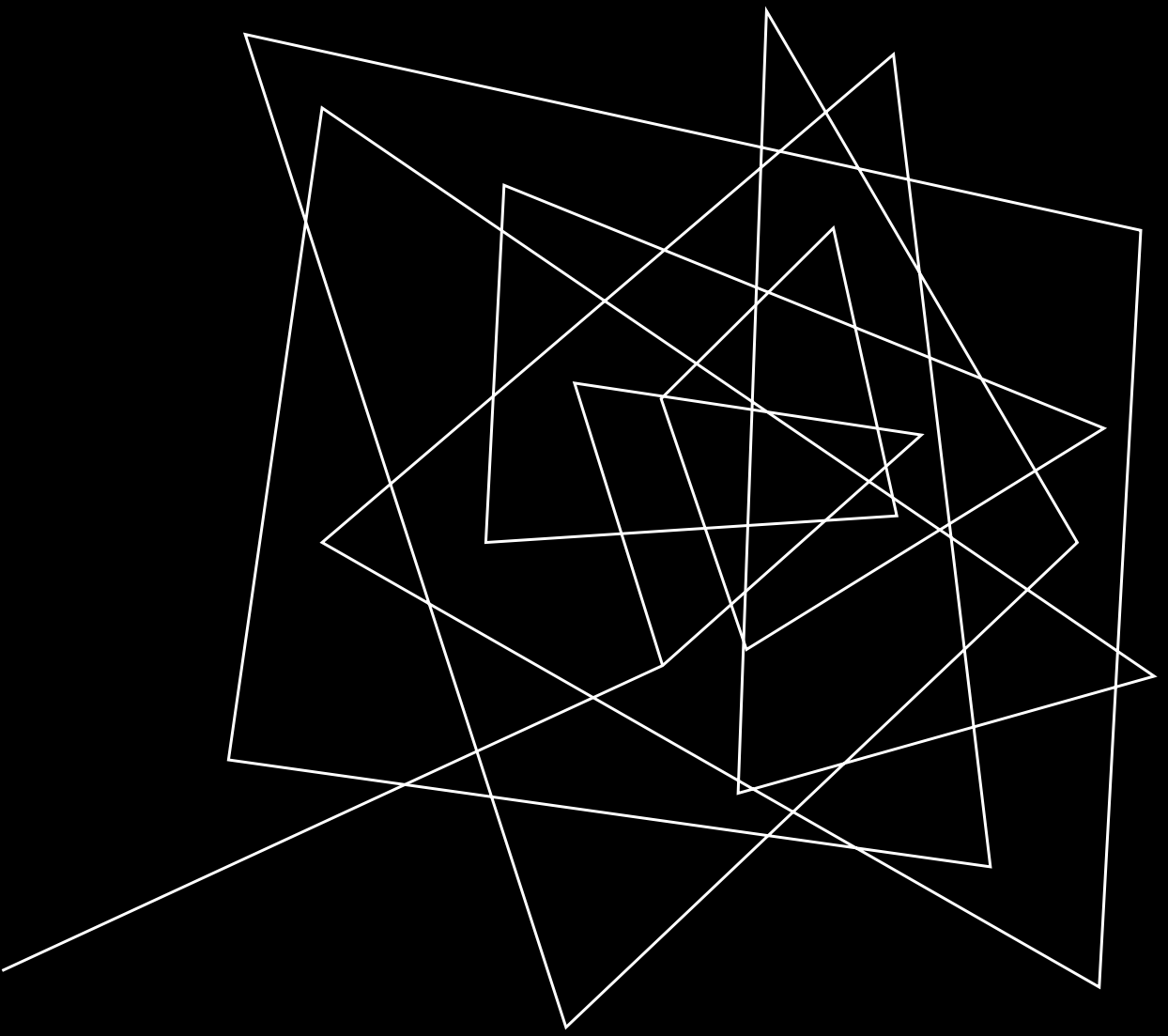
DATA AUGMENTATION TECHNIQUES



TIME SERIES DATA AUGMENTATION



Source: Iwana, Brian Kenji and Seiichi Uchida. "Time Series Data Augmentation for Neural Networks by Time Warping with a Discriminative Teacher." *2020 25th International Conference on Pattern Recognition (ICPR)* (2020): 3558-3565.



MODEL OPTIMIZATION: HYPERPARAMETER TUNING

HYPERPARAMETERS

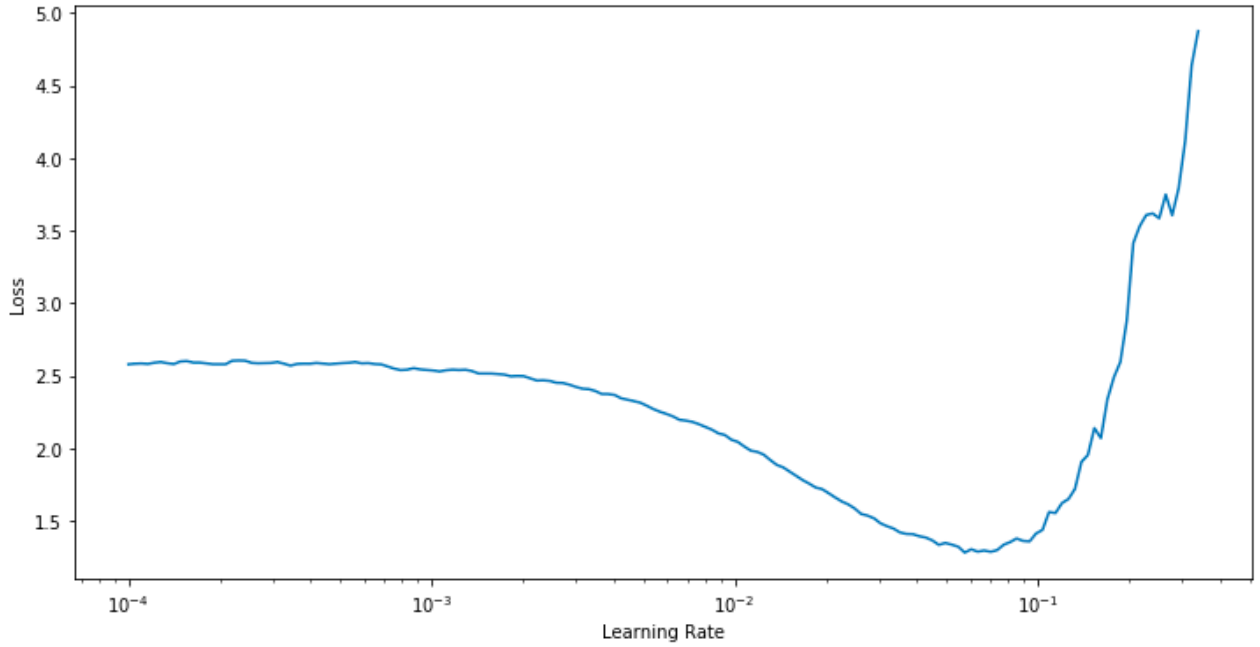
Hyperparameters are the external parameters set *before* the learning process of a machine learning model begins. They influence the training process, determining how the model learns, but **they are not learned by the model itself. Unlike model parameters (such as weights in linear regression/neural networks), hyperparameters are defined by the practitioner** and **need to be tuned manually** or via techniques like grid search, random search, or more advanced optimization methods like Bayesian optimization.

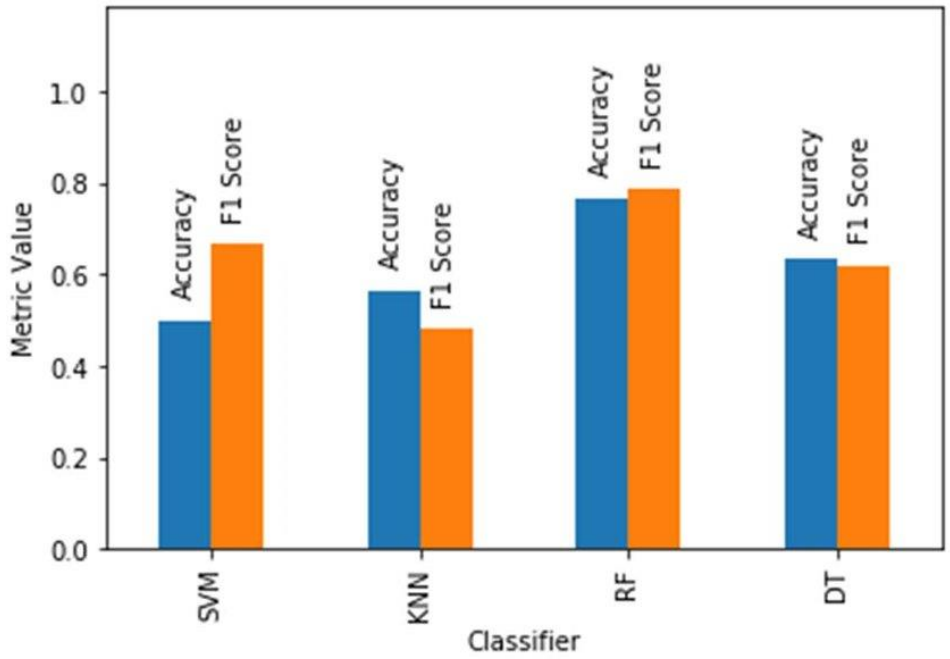
Types of Hyperparameters

- **Model-Specific:** Choice of Algorithm, Number of layers and Architecture (in Neural Networks), Number & Max depth of trees (in Random Forest), Degree of Polynomials (in Regression), etc.
- **Training-Specific:** Choice of Optimizing Algorithm, Cost Function, Learning Rate (α), Batch Size, Number of Epochs, etc.
- **Regularization Hyperparameters:** Choice of Regularizer (L1/L2/Dropout etc.), Regularization Parameter (λ), Dropout Rate, etc.
- **Data-Specific:** Choice of Feature Scaling technique, Extent of data augmentation (Rotation Angle, Noise Level, etc.).

HYPERPARAMETER TUNING: MANUAL TUNING

Manual Tuning involves adjusting a single hyperparameter by plotting the validation performance (such as **model type** or **accuracy** or **loss**) against different values/categories of that hyperparameter. Based on the curve/chart, we can visually inspect which value provides the best validation performance and choose the optimal value/settings for that hyperparameter. This approach is simple but effective for tuning one hyperparameter at a time when computational resources are limited.





HYPERPARAMETER TUNING: GRID SEARCH

Grid Search is an exhaustive search method where a grid of hyperparameter values is defined, and the model is trained and evaluated for every possible combination of these values. This method ensures extensive tuning but can be computationally expensive.

Steps

- A set of hyperparameter ranges for each parameter (e.g., learning rates, number of layers, dropout rates, max depth) is defined.
- The algorithm trains and evaluates the model for every possible combination within this grid.
- The best combination of hyperparameters is chosen based on performance metrics (e.g., accuracy, F1 score, log loss, etc.) on a validation set.

Sample Code

```
from sklearn.model_selection import GridSearchCV
param_grid = {'learning_rate': [0.001, 0.01, 0.1],
              'n_estimators': [100, 200],
              'max_depth': [3, 5, 10]}

grid_search = GridSearchCV(model, param_grid, cv=3)
grid_search.fit(X_train, y_train)
```

HYPERPARAMETER TUNING: RANDOM SEARCH

Random Search is a more efficient alternative to grid search, where the hyperparameter values are randomly sampled from the defined ranges. Instead of testing all possible combinations, only a random subset is evaluated. This method can find a good combination faster by avoiding the exhaustive search of all combinations, but there's no guarantee that it will find the best combinations.

Steps

- Similar to grid search, a set of hyperparameter ranges for each parameter is defined.
- The algorithm selects random combinations from these ranges and evaluates the model performance for each.

Sample Code

```
from sklearn.model_selection import RandomizedSearchCV
param_distributions = {'learning_rate': [0.001, 0.01, 0.1],
            'n_estimators': [100, 200],
            'max_depth': [3, 5, 10]}

random_search = RandomizedSearchCV(model, param_distributions , n_iter=10, cv=3)
random_search.fit(X_train, y_train)
```

HYPERPARAMETER TUNING: BAYESIAN OPTIMIZATION

Bayesian Optimization is a more sophisticated approach that models the performance of the hyperparameters using a probabilistic surrogate model (usually a Gaussian Process) and updates this model iteratively based on past evaluations. This method finds optimal hyperparameters in fewer iterations than grid or random search.

Steps

- A prior distribution is placed over the hyperparameter space.
- The model is trained on a few initial sets of hyperparameters and then updates the posterior distribution to predict which sets of hyperparameters are most likely to improve performance.
- It selects the next set of hyperparameters to try by balancing exploration (trying new values) and exploitation (focusing on the values that have worked well so far).

Sample Code

```
from skopt import BayesSearchCV
param_space = {'learning_rate': (0.001, 0.1, 'log-uniform'),
               'n_estimators': (100, 300), 'max_depth': (3, 10)}

opt = BayesSearchCV(model, param_space, n_iter=32, cv=3)
opt.fit(X_train, y_train)
```

HYPERPARAMETER TUNING: HYPERBAND

Hyperband is a hyperparameter optimization method based on successive halving, which evaluates multiple configurations with fewer resources and then progressively allocates more resources (such as training time) to the more promising configurations. This method can prune unpromising hyperparameter settings early in the process.

Steps

- The algorithm starts by testing many different hyperparameter combinations with limited resources (e.g., using fewer epochs or a subset of the data).
- As training progresses, the configurations with poor performance are discarded, and more resources are allocated to the better-performing configurations.
- Continues until a final set of hyperparameters is selected based on the resources available.

Sample Code

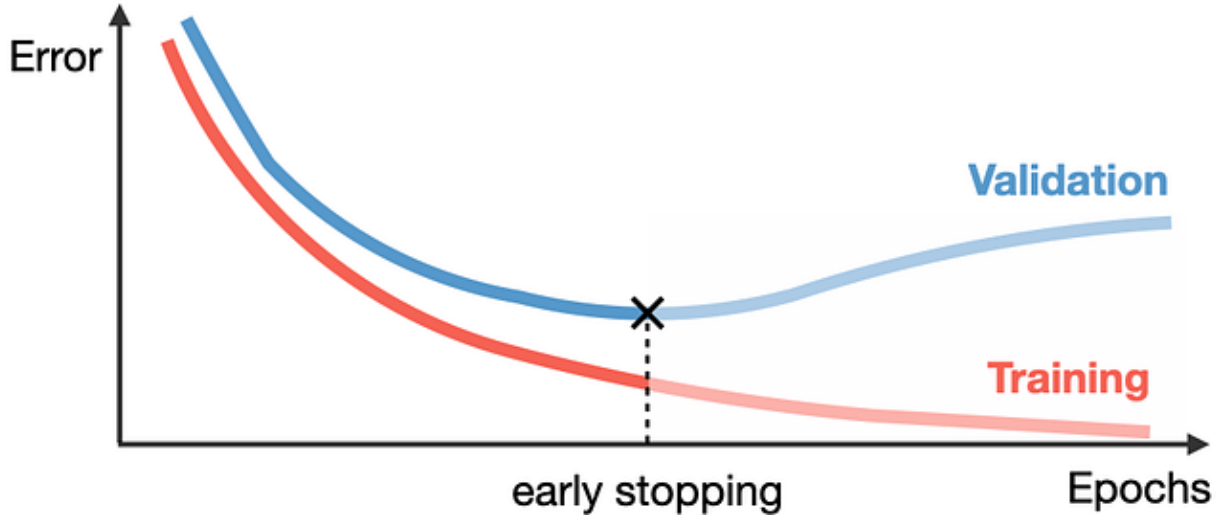
```
from keras_tuner import Hyperband
tuner = Hyperband(model_builder, max_epochs=40,
                  objective='val_accuracy', executions_per_trial=3)

tuner.search(X_train, y_train, epochs=20, validation_data=(X_val, y_val))
```

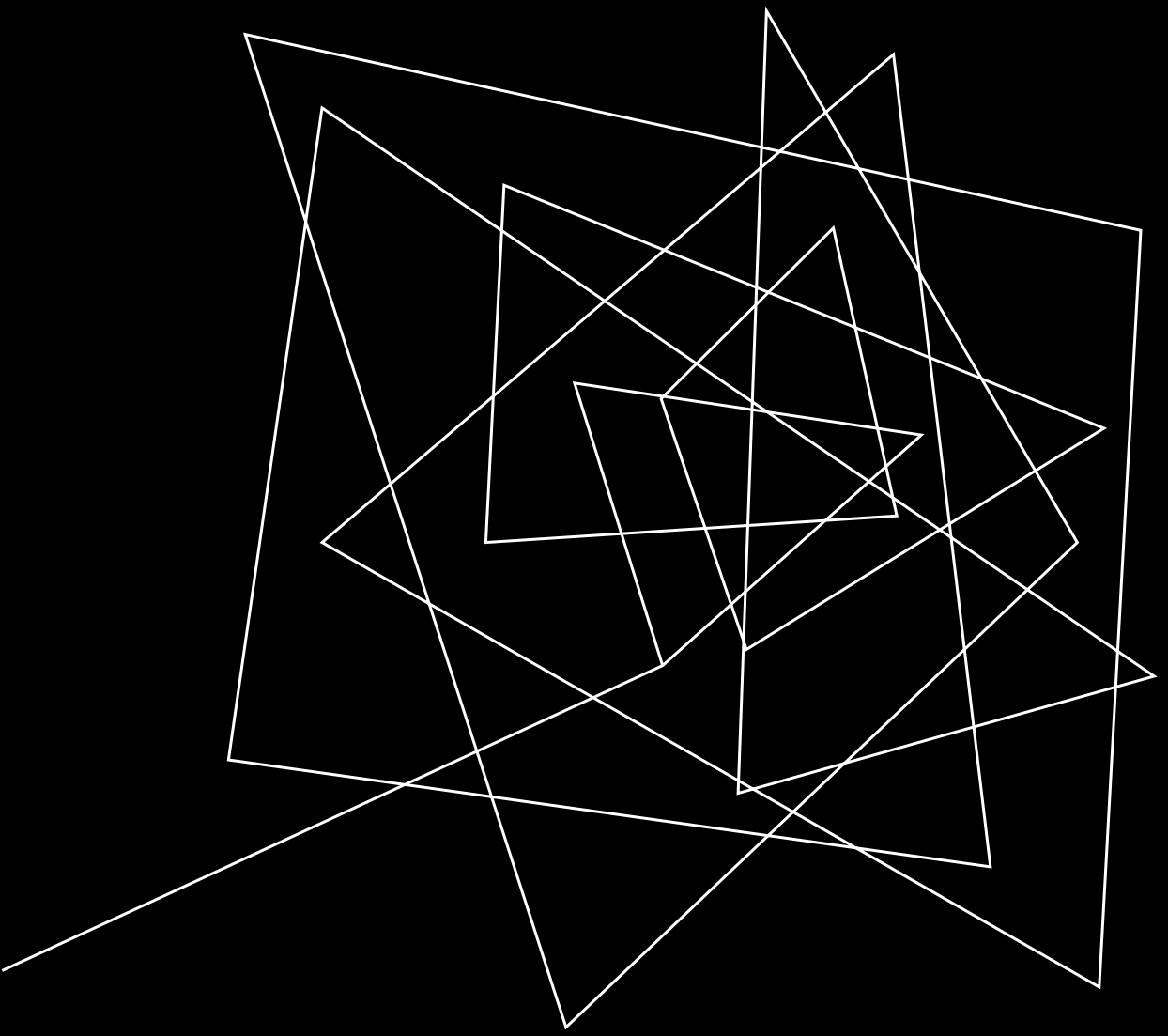
EARLY STOPPING

While not a hyperparameter tuning method per se, **Early Stopping** is a technique used to monitor the performance of a model on validation data and stop the training process when performance stops improving. This helps in preventing overfitting and acts as an implicit hyperparameter optimization.

Steps

- During training, you monitor a performance metric on a validation set.
- If the performance on the validation set stops improving for a predefined number of epochs (called the "patience"), the training is stopped.
- The best weights obtained during training are restored.

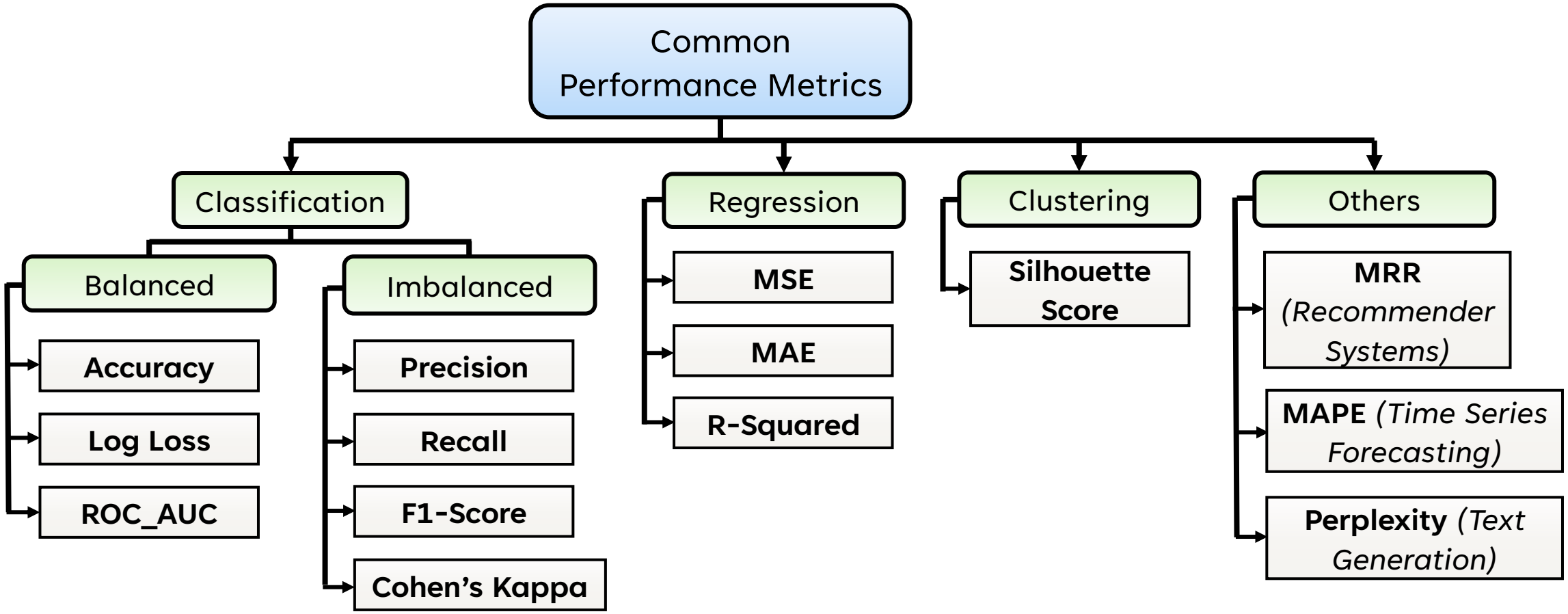




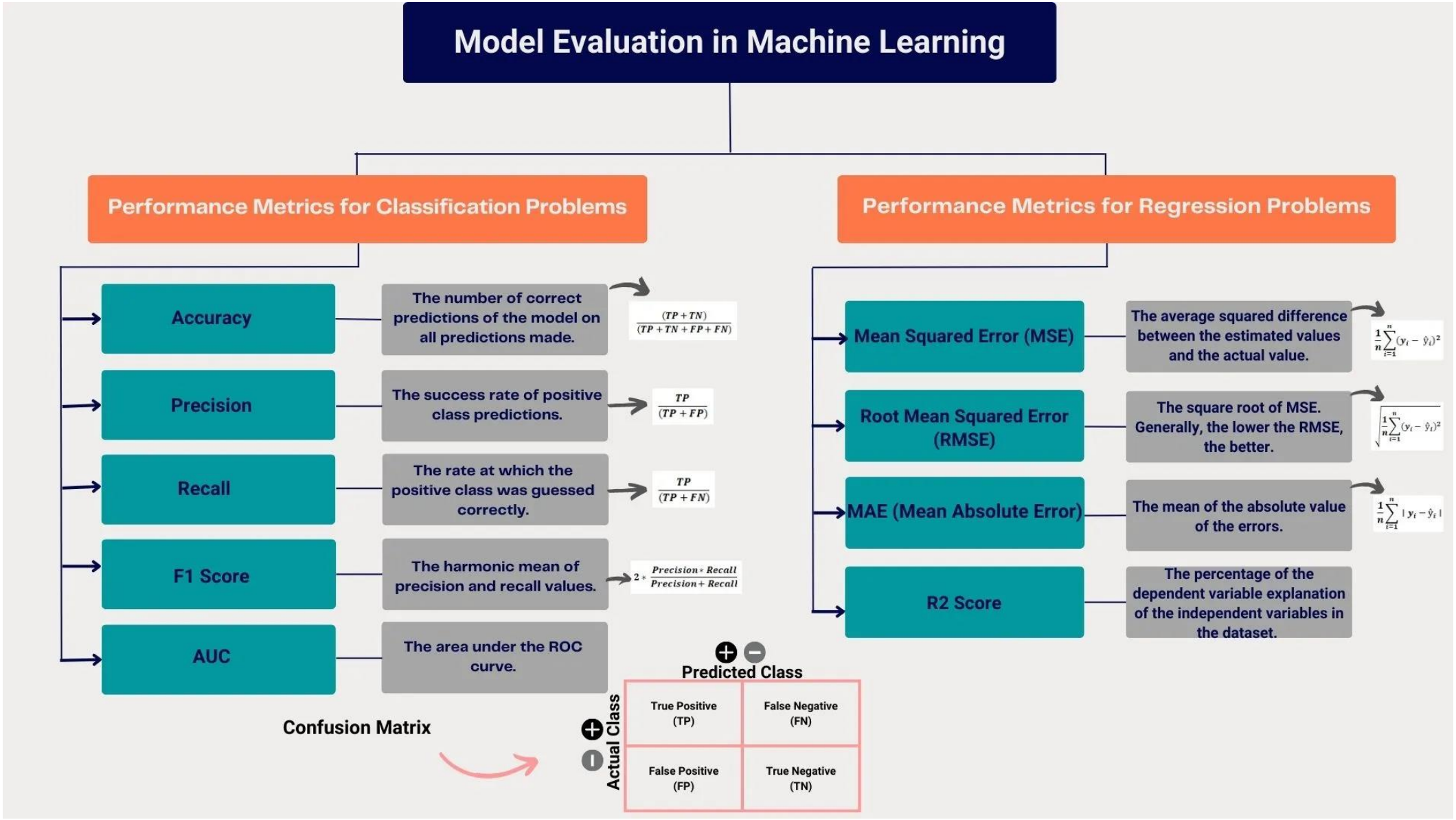
MODEL EVALUATION

EVALUATION: PERFORMANCE METRICS

ML **Performance Metrics** are used to **evaluate** how well a model performs on a specific task. These metrics provide quantitative insights into the model's accuracy, generalization, and robustness. The choice of metric depends on the type of problem (classification, regression, etc.) and the goals of the analysis.



PERFORMANCE METRICS



ACCURACY & RATE OF ERROR

Accuracy and **Rate of Error** are fundamental metrics used to evaluate the performance of ML models, especially in classification tasks. These metrics provide insight into how well the model performs by measuring the proportion of correctly or incorrectly predicted instances.

Accuracy: It is the most intuitive and widely used metric for evaluating classification models. It represents the percentage of correctly predicted instances out of the total instances.

$$Accuracy = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} = \frac{TP+TN}{TP+TN+FP+FN}$$

Where,
TP (True Positives): Correct predictions of the positive class.
TN (True Negatives): Correct predictions of the negative class.
FP (False Positives): Incorrect predictions of the positive class (**Type I Error**).
FN (False Negatives): Incorrect predictions of the negative class (**Type II Error**).

Assuming there's one **Positive (1)** & one **Negative (0)** class. This assumption can be extended to Multi-Class cases (All classes **negative** bar one).

Rate of Error (or **Error Rate**): It is the complement of accuracy. It measures the proportion of incorrect predictions the model makes, essentially highlighting the percentage of mistakes.

$$Rate\ of\ Error = 1 - Accuracy = \frac{FP+FN}{TP+TN+FP+FN}$$

CLASS DISTRIBUTION: BALANCED vs IMBALANCED

In Classification tasks, for imbalanced datasets(**where the number of samples varies significantly across different classes**), the model tends to be biased towards the majority class, predicting it more frequently since it has more training samples. As a result, overall accuracy can be misleadingly high even if the model is performing poorly on minority classes.

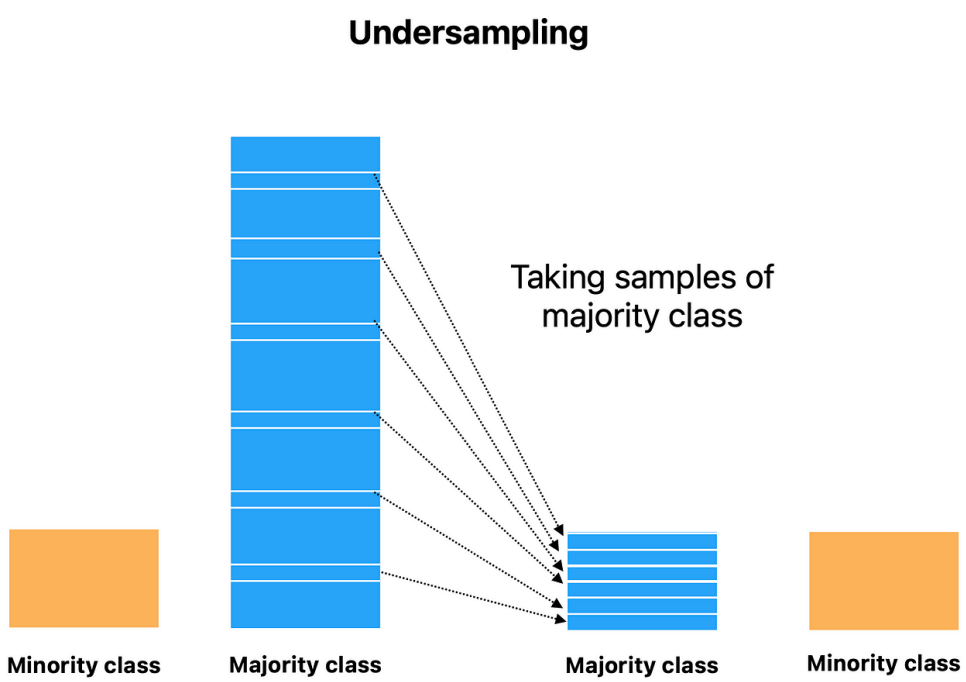
For example, In a dataset where ***95% of instances belong to Class A*** and ***only 5% belong to Class B***, a model predicting Class A all the time could still achieve **95% accuracy**, despite being completely ineffective at identifying Class B. Is this a good model? *Definitely not!*

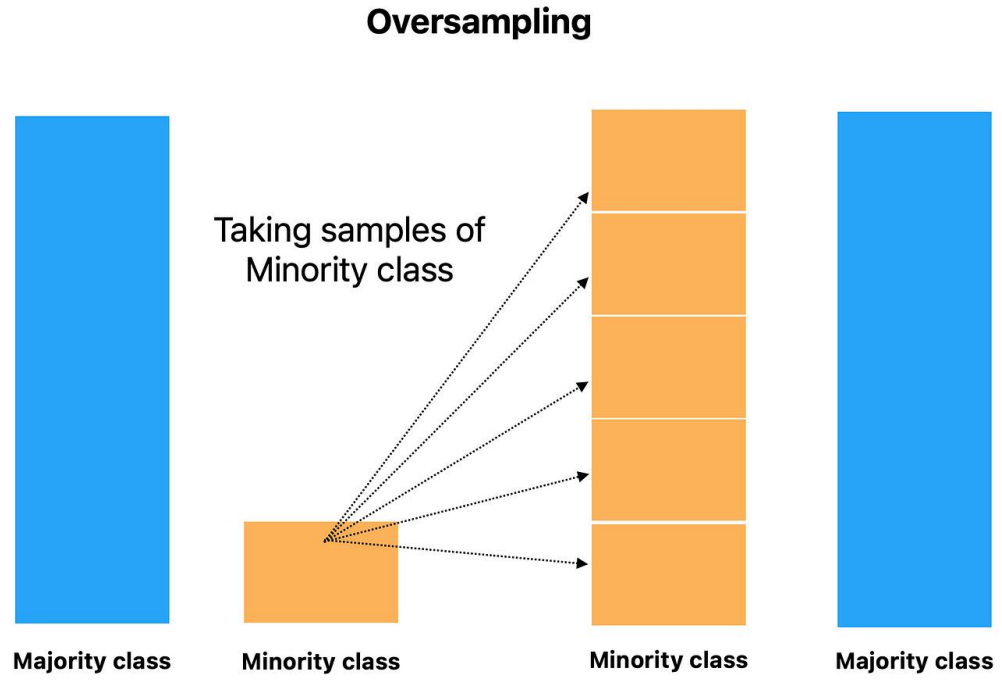
Also, Minority classes are often the most important to detect, such as *fraud detection (fraud cases are rare)* or *medical diagnosis (rare diseases).* If a model performs poorly on minority classes, its practical utility decreases significantly.

Metrics like **Accuracy**, when applied without class-wise evaluation, can hide poor performance on minority classes. Instead, metrics like **Precision**, **Recall**, and **F1-Score** should be computed per class to understand class-wise performance explicitly. Also, the training process of imbalanced datasets should be handled carefully to avoid majority class bias, possibly through **Oversampling** or **Undersampling**.

HANDLING IMBALANCE IN DATA

Oversampling and Undersampling are two common techniques used to handle class imbalance in datasets, where the number of instances in one class is significantly higher than in another. **Oversampling** involves increasing the number of instances in the minority class to balance the dataset. This can be done by duplicating existing instances or generating new synthetic instances. **Undersampling** involves reducing the number of instances in the majority class to match the size of the minority class. This method causes information loss from the majority class.





EVALUATING MODELS ON IMBALANCED DATA

For imbalanced datasets, metrics like **Precision, Recall, F1 Score, AUC-ROC,** and **Cohen's Kappa** (which accounts for chance agreement) are more informative than overall accuracy. Evaluating these metrics for each class ensures a balanced view of the model's effectiveness across all classes.

Precision (Specificity): It is the ratio of correctly predicted positive instances to the total predicted positives. It is important when false positives are costly (e.g., in spam detection).

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

Recall (Sensitivity): The ratio of correctly predicted positive instances to all actual positive instances. It is important when missing positive cases is costly (e.g., in medical diagnosis).

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

F1-Score: It is the harmonic mean of precision and recall, balancing the two metrics. Ideal for imbalanced classification problems where a balance between precision and recall is needed.

$$\boldsymbol{F1-Score} = 2 \bullet \frac{\boldsymbol{Precision} \bullet \boldsymbol{Recall}}{\boldsymbol{Precision} + \boldsymbol{Recall}}$$

ADJUSTING THE THRESHOLD

In the context of classification models, the **Threshold** is a value that determines how the model's predicted probabilities are converted into class labels (such as "positive" or "negative"). For a binary classification model, the model typically outputs a probability score between 0 and 1 for each instance. *The threshold is the point at which this probability is "cut off" to decide which class the instance belongs to*. It can be adjusted according to the need of the ML task to achieve better Precision/Recall Trade-off.

Suppose the model outputs a probability score for each instance. The **default threshold** is typically set to **0.5**, meaning that **if the probability score is ≥ 0.5, the instance is classified as positive (1) & vice-versa**.

We can **adjust the threshold** to control the balance between True Positives and False Positives. Example:

- **Higher Threshold (e.g., 0.7):** Classifies an instance as positive **only if the probability is ≥ 0.7.** This **increases Precision** (fewer false positives) but may decrease recall (more false negatives).
- **Lower Threshold (e.g., 0.3):** Classifies an instance as positive **if the probability is ≥ 0.3.** This **increases Recall** (fewer false negatives) but may decrease precision (more false positives).

In summary, the threshold directly controls the sensitivity of the classification decision, and adjusting it can fine-tune the balance between precision, recall, and other performance metrics. For example, a lower threshold(hence, high recall) can be advantageous in Cancerous Tumor Detection tasks as failing to detect a positive case can mean disastrous consequences.

AUC-ROC (AREA UNDER THE ROC CURVE)

AUC-ROC is a performance measurement for classification models at various threshold settings. The **ROC (Receiver Operating Characteristic) curve** plots the true positive rate (**Recall**) against the **false positive rate**. **AUC** represents the **area under this curve**.

$$\textbf{\textit{True Positive Rate}} = \frac{\text{TP}}{\text{TP+FN}} \quad \& \quad \textbf{\textit{False Positive Rate}} = \frac{\text{FP}}{\text{FP+TN}}$$

The ROC curve plots **TPR (Y-axis)** against **FPR (X-axis)** for different threshold values.

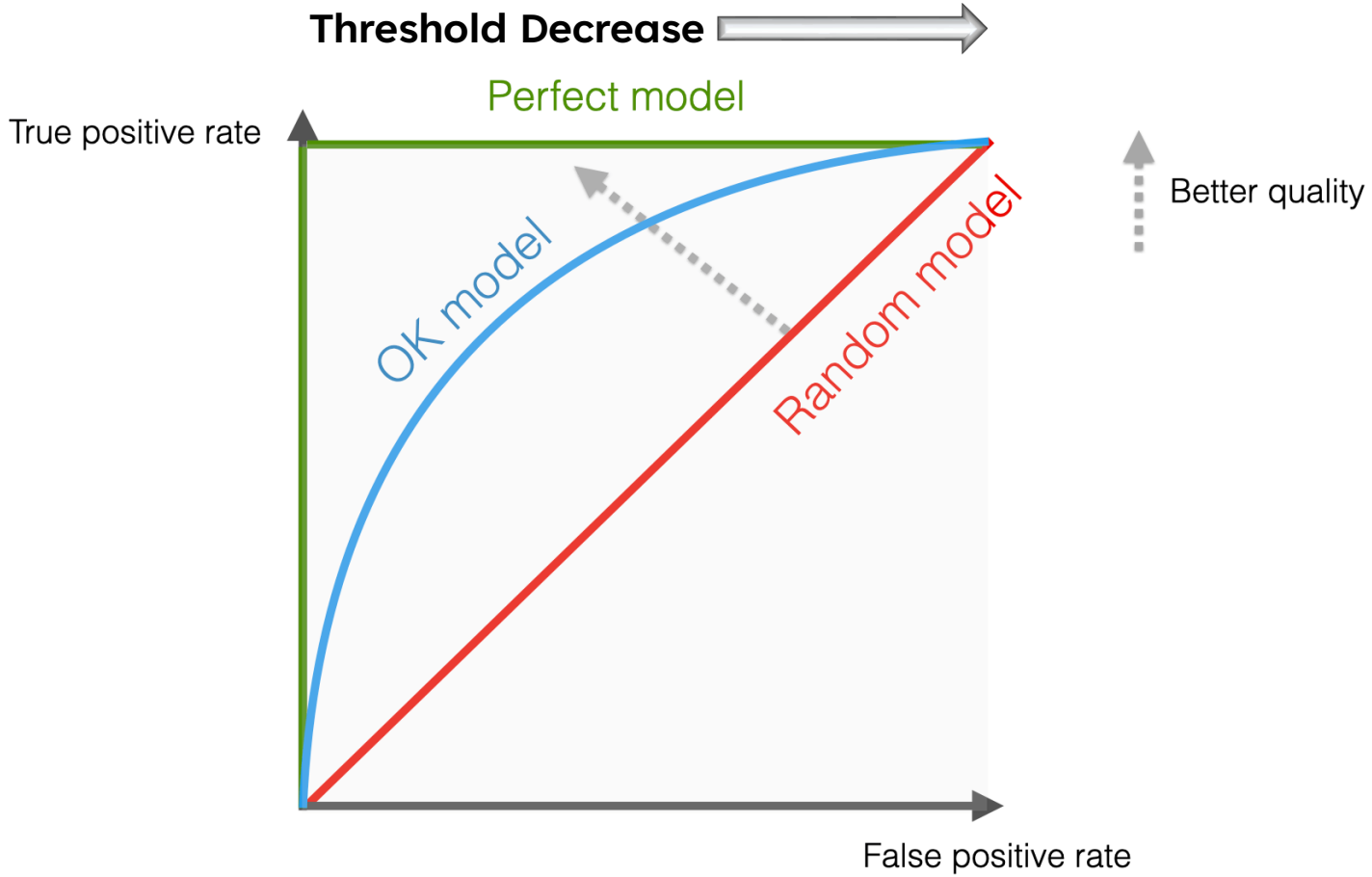
AUC stands for the *Area Under the Curve of the ROC plot*. It gives an aggregate measure of the model's performance across all classification thresholds. The AUC value ranges between 0 and 1.

- **AUC = 1:** This represents a perfect model.
- **AUC > 0.5:** TPR is greater than FPR in most cases. This means the model is at least better than random guessing and learning the relationship bet^{n} the features and the labels.
- **AUC = 0.5:** This represents a model with no discriminative ability (random guess).
- **AUC < 0.5:** This means the model performs even worse than random guessing.

AUC-ROC evaluates model performance **across all possible thresholds**, rather than a fixed threshold, giving a comprehensive view of how well the model can distinguish between the positive and negative classes, even if the final predictions are similar for a particular threshold.

THE ROC CURVE

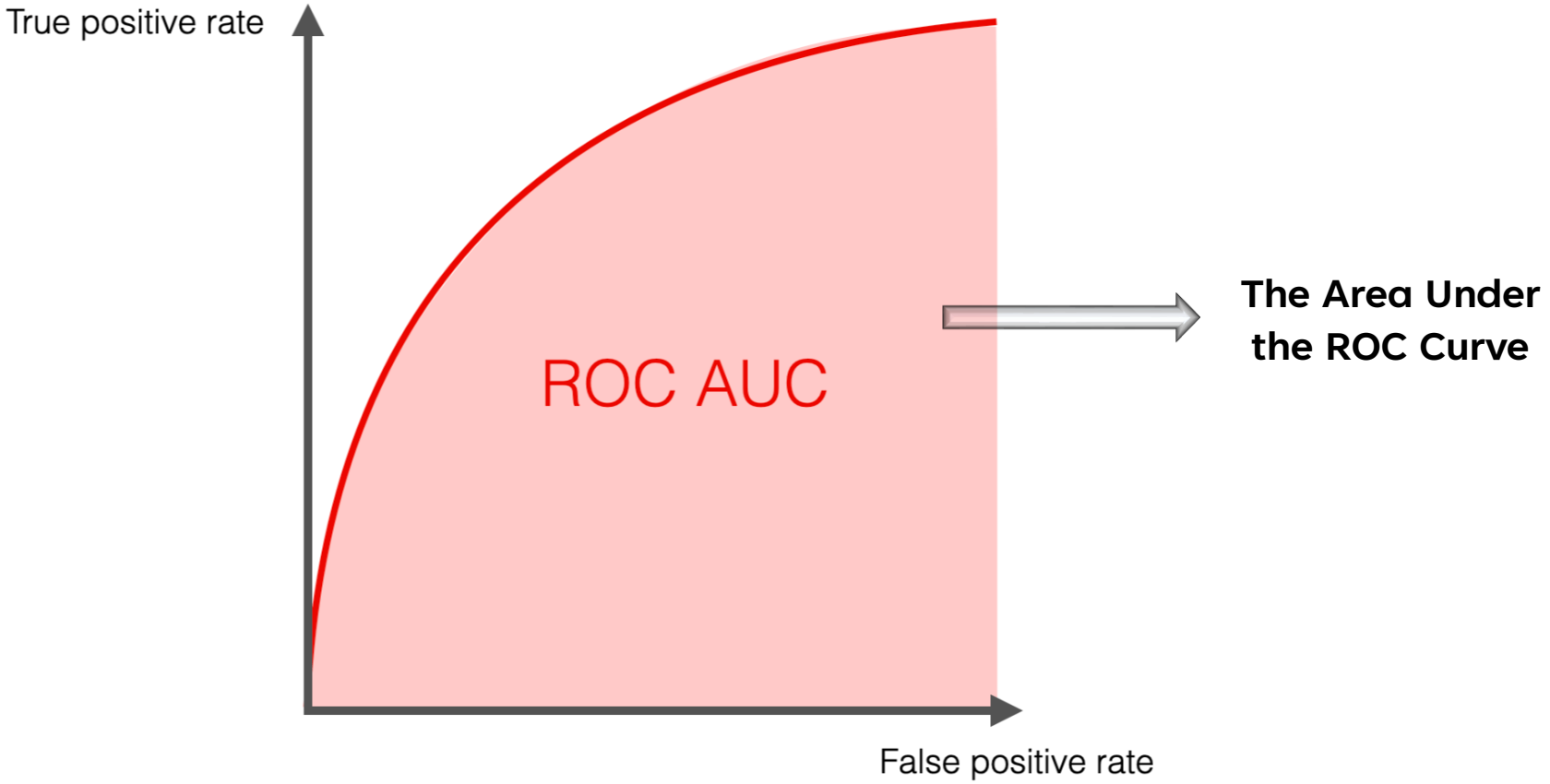
A curve closer to the top-left corner indicates a better model, meaning higher sensitivity and lower false positives. If the curve is closer to the diagonal line (random guessing line), it indicates poor performance.



Source: https://www.evidentlyai.com/classification-metrics/explain-roc-curve

AUC-ROC (AREA UNDER THE ROC CURVE)

A **ROC AUC score** or **AUC-ROC** is a single metric to summarize the performance of a classifier across different thresholds. To compute the score, we must measure **the area** under the ROC curve. ROC AUC score can be computed in Python using different libraries like Scikit-learn.



Source: https://www.evidentlyai.com/classification-metrics/explain-roc-curve

CONFUSION MATRIX

A **Confusion Matrix** is a table/matrix used to evaluate the performance of a classification model. It compares the actual target values (true labels) with the model's predicted values. The matrix is typically structured as follows for binary classification:

$$\begin{bmatrix} \textbf{TP} & \textbf{FN} \\ \textbf{FP} & \textbf{TN} \end{bmatrix}$$

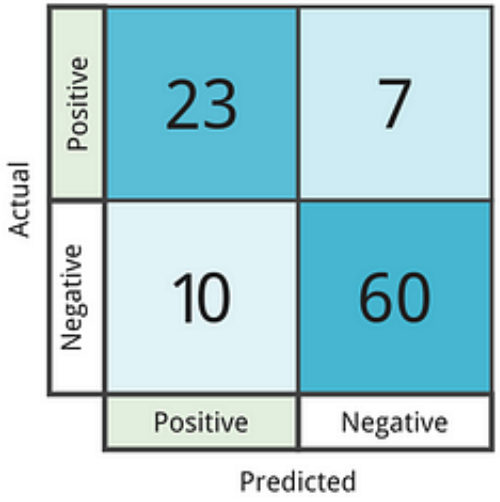
It can be interpreted as a heatmap too, where **True labels** are plotted along the **vertical axis (Top to Bottom)** and **Predicted labels** are plotted against the **horizontal axis (Left to Right)**.

Importance

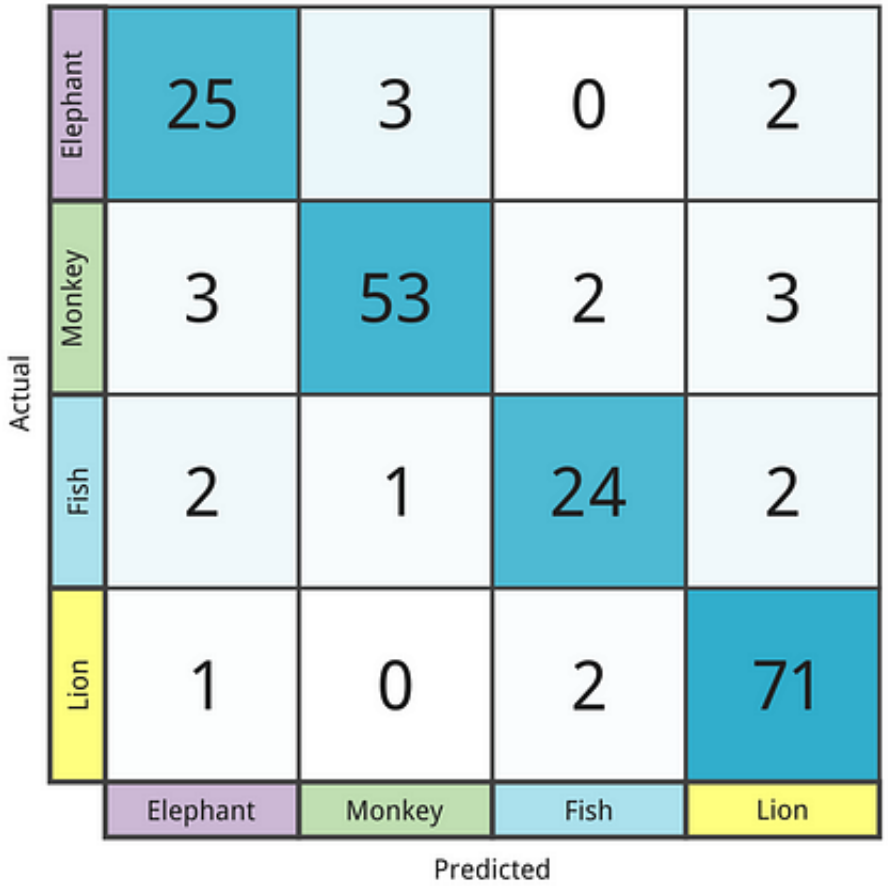
- It shows how well a model distinguishes between different classes, especially useful for understanding performance on individual classes in multi-class problems.
- Evaluates Precision and Recall: It allows for the calculation of metrics like precision (how many predicted positives are correct) and recall (how many actual positives are captured.
- By observing false positives and false negatives, it helps understand specific types of errors the model makes and provides guidance on where to focus model improvements.

CONFUSION MATRIX

Binary Classification



Multi-Class Classification

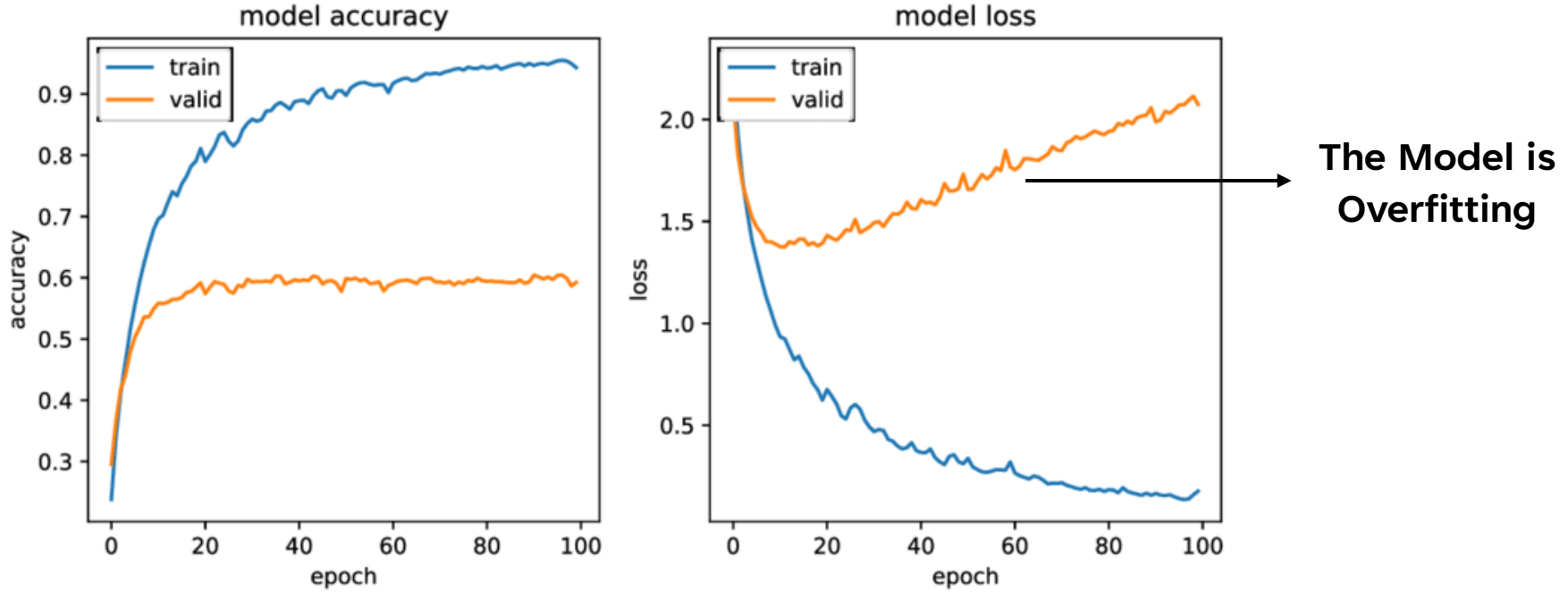


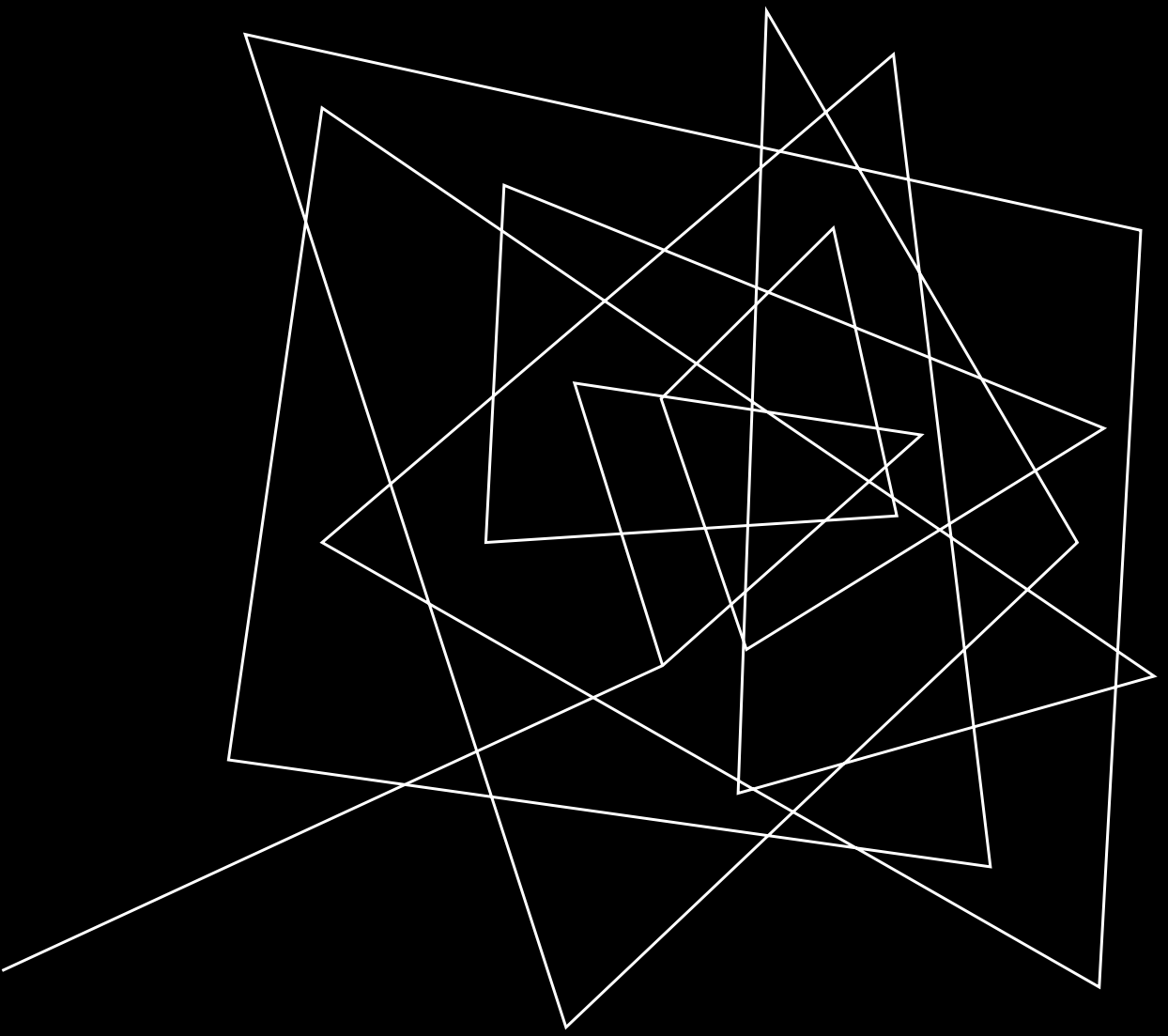
Source: https://towardsdatascience.com/visual-guide-to-the-confusion-matrix-bb63730c8eba

LOSS & ACCURACY CURVES

Loss and Accuracy Curves plot the loss (cost) and accuracy of a model over time (training epochs/iterations) during training and validation. **Loss curves** show how well the model is minimizing the cost on both the training and validation sets. **Accuracy curves** display the model's performance in correctly predicting outputs as the training progresses.

They are important because they help monitor training progress, identify overfitting or underfitting, and ensure proper convergence by diagnosing issues and adjusting accordingly.





MODEL DEPLOYMENT & MONITORING

MODEL DEPLOYMENT

Model Deployment is the process of integrating a machine learning model into a production environment where it can start making predictions on real-world data. It's a critical step after model training, enabling businesses or applications to use the insights derived from the model.

Key Steps

1. **Model Packaging:** The model is **saved** in a format that can be easily loaded in production (e.g., via TensorFlow's SavedModel format, PyTorch's .pt file, or ONNX).
2. **Integration with Applications:** The deployed model is **integrated with a system** or service, such as web apps, mobile apps, or APIs, allowing other software components to query it for predictions.
3. **Infrastructure Setup:** The model **runs on suitable infrastructure** (local servers, cloud platforms, edge devices). Cloud providers like AWS, Google Cloud, and Azure offer services (e.g., AWS SageMaker, GCP's AI Platform) for hosting models.
4. **Serving Predictions:** Once deployed, the model **serves real-time** or **batch predictions**. For real-time predictions, RESTful APIs or gRPC endpoints are often used to send data to the model and retrieve predictions.
5. **Security:** The deployed model needs to be **secured**. This involves encryption, managing access control, and making sure that sensitive data is protected during inference.

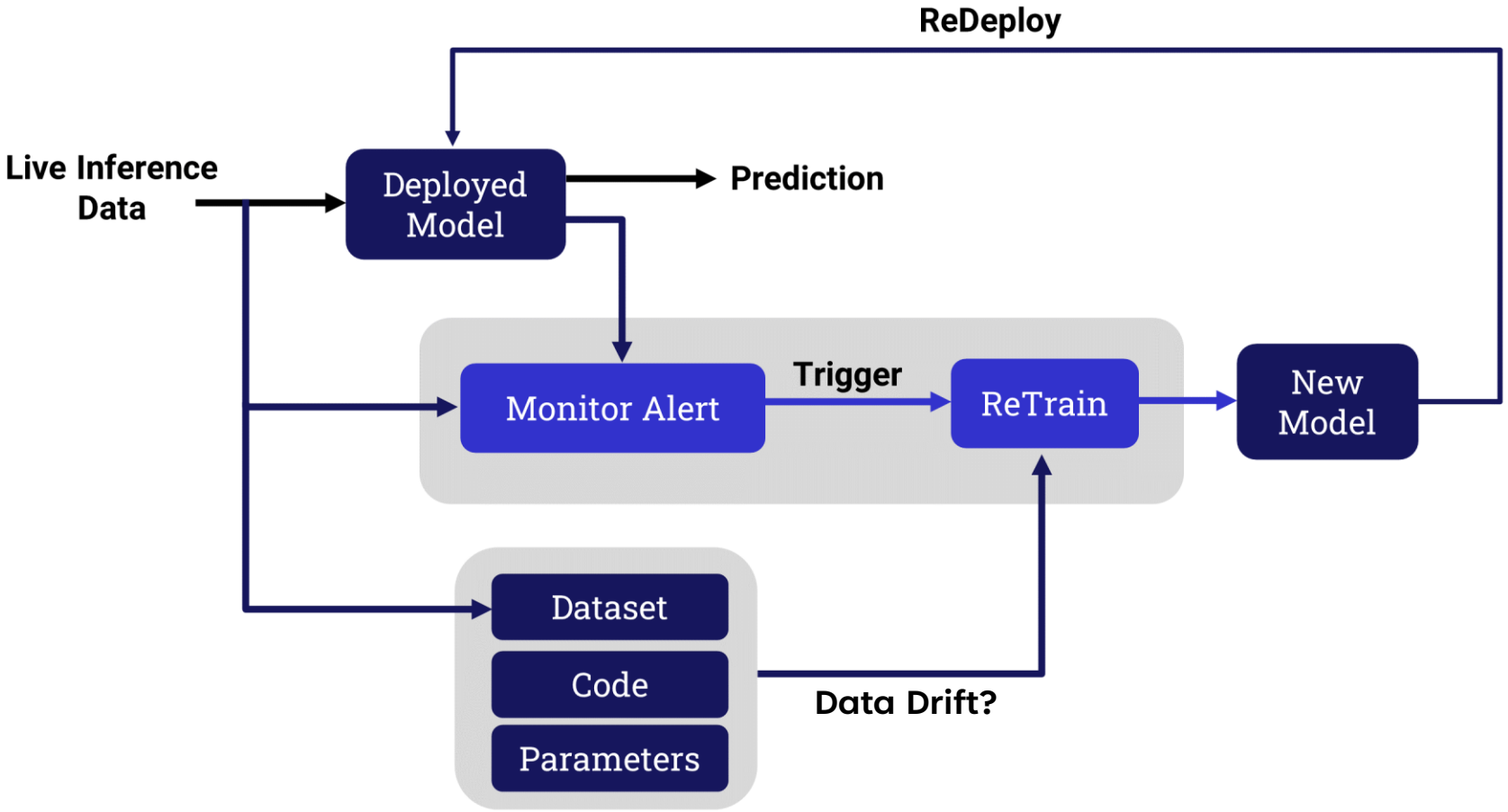
MONITORING & FEEDBACK

Monitoring involves continuously observing the model's performance after deployment to ensure it remains effective over time. This step is essential as real-world data often shifts, potentially degrading the model's accuracy. Monitoring ensures continuous feedback and improvement.

Key Aspects

- **Performance Tracking:** Monitoring metrics like accuracy, precision, recall, latency, and throughput to ensure the model performs well. Metrics may change due to data drift or changing patterns in data.
- **Data Drift Detection:** Monitoring for shifts in the input data distribution that may differ from the training data. Data drift can lead to model underperformance, and identifying it helps initiate retraining or adjustments.
- **Model Drift Detection:** Monitoring output for consistency with expectations. Over time, the model might become less effective if the underlying data or problem changes (model drift).
- **Alerting and Retraining:** When performance drops below a certain threshold, automated alerts can be set up, triggering a review or retraining process using newer data to maintain accuracy.
- **Versioning and Rollback:** Keeping track of different model versions is important for deploying updates or rolling back to a previous, stable version if a newer one underperforms.

MONITORING & FEEDBACK



ML Model Monitoring

PRACTICAL SCENARIOS

PRACTICAL SCENARIO 1

Living Area (feet^2)	#Bedrooms	Price (X1000$)
2104	3	**400**
1600	3	**330**
2400	3	**369**
1416	2	**232**
3000	4	**516**

The task is to **predict house prices** based on the living area & the number of Bedrooms (multiple features) using the above dataset.

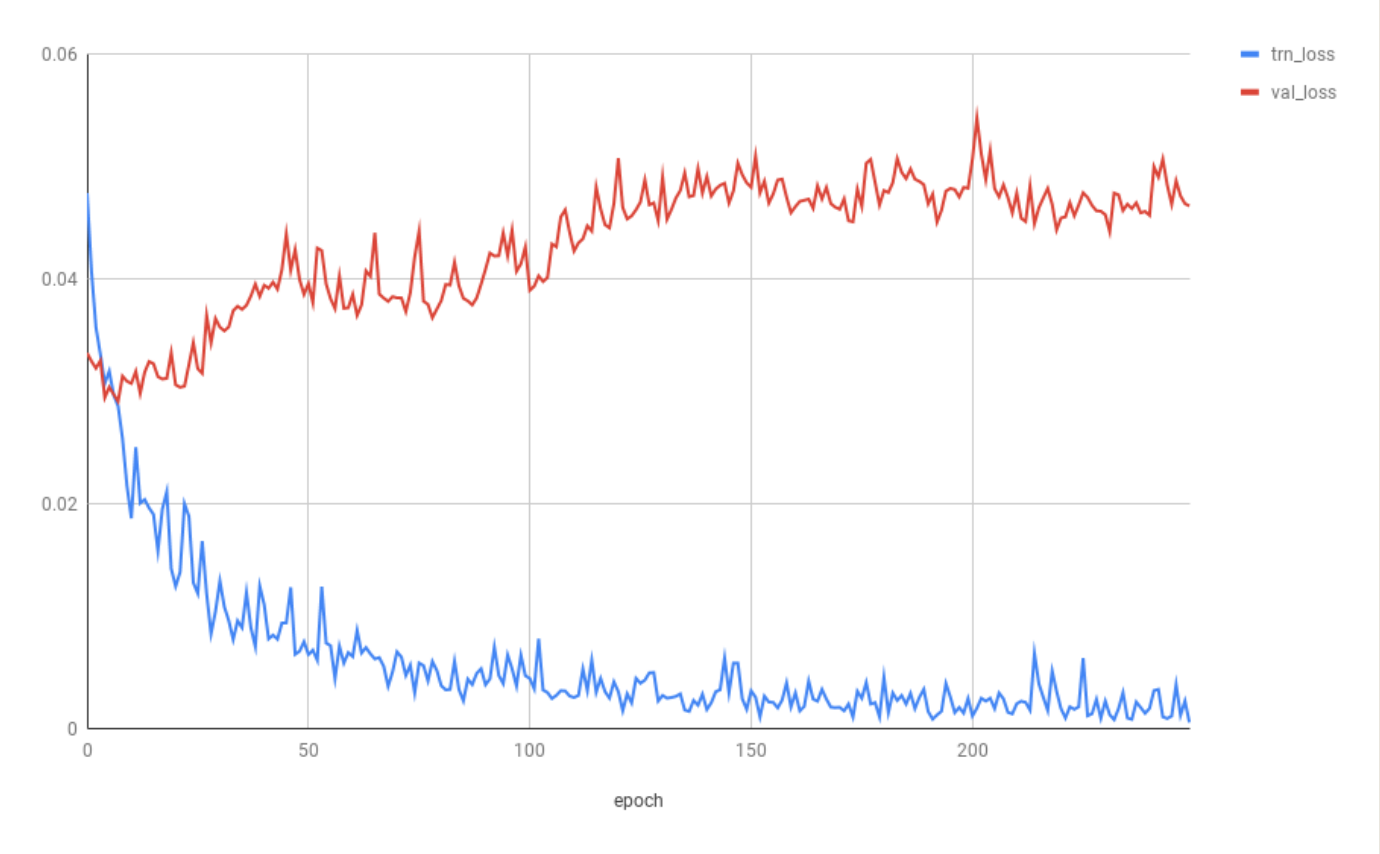
- ❑ What are the **Data Transformation** techniques that can be used to prepare data for this task?
- ❑ What are the **Hyperparameters** that may need **tuning** (based on on you chosen model)?

PRACTICAL SCENARIO 2

Order_ID	Customer_Name	Product	Quantity	Price	Date	Total_Sales	Target
101	John Doe	Product A	2	20.5	1/1/2023	41	**Successful**
102	jane doe	Product B	1	35		35	**Unsuccessful**
103	MARY LOU	Product C	5		1/3/2023	175	**Successful**
104	John Doe	Product A	-3	20.5	1/4/2023	-61.5	**Unsuccessful**
105	Mary Lou	Product B	2	35	1/5/2023	70	**Successful**
106		Product D	1	50	1/6/2023	50	**Successful**
107	jane DOE	Product C	10	17.5	1/7/2023	175	**Successful**
108	Mary Lou	Product A	2		1/8/2023		**Unsuccessful**
109	John Doe	Product B	1	35	1/9/2023	35	**Successful**
110		Product D	10		10-Jan	2023	**Unsuccessful**
111	jane doe	Product C	5	17.5	1/11/2023	87.5	**Successful**
112	John DOE	Product A	1		1/12/2023		**Unsuccessful**

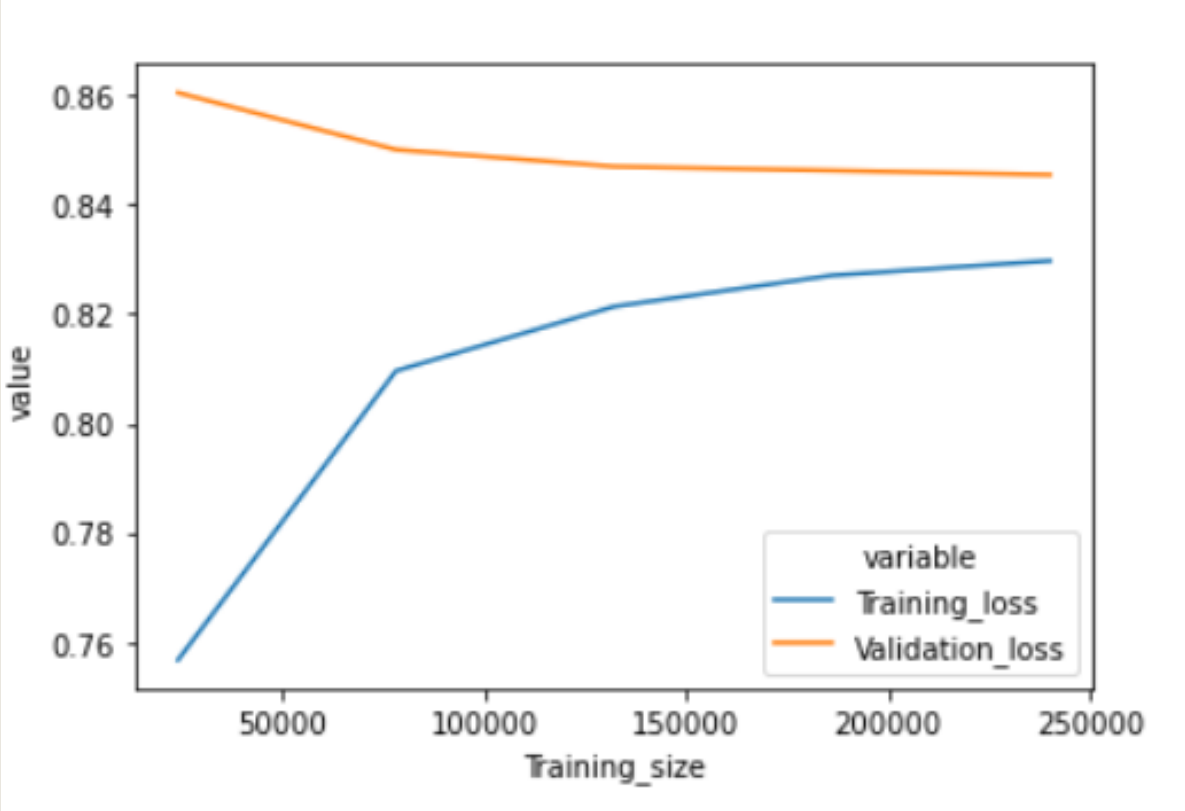
- ❑ What are some **Data Cleaning** techniques that you may need to clean this dataset & Why?
- ❑ What are some **EDA** techniques that you may need to analyze this dataset?
- ❑ What are some **Data Transformations** that you may need to prepare this dataset?
- ❑ What kind of ML model would you like to use to perform this Binary Classification task (Whether a sales transaction is **successful** based on the features in the dataset)?

PRACTICAL SCENARIO 3



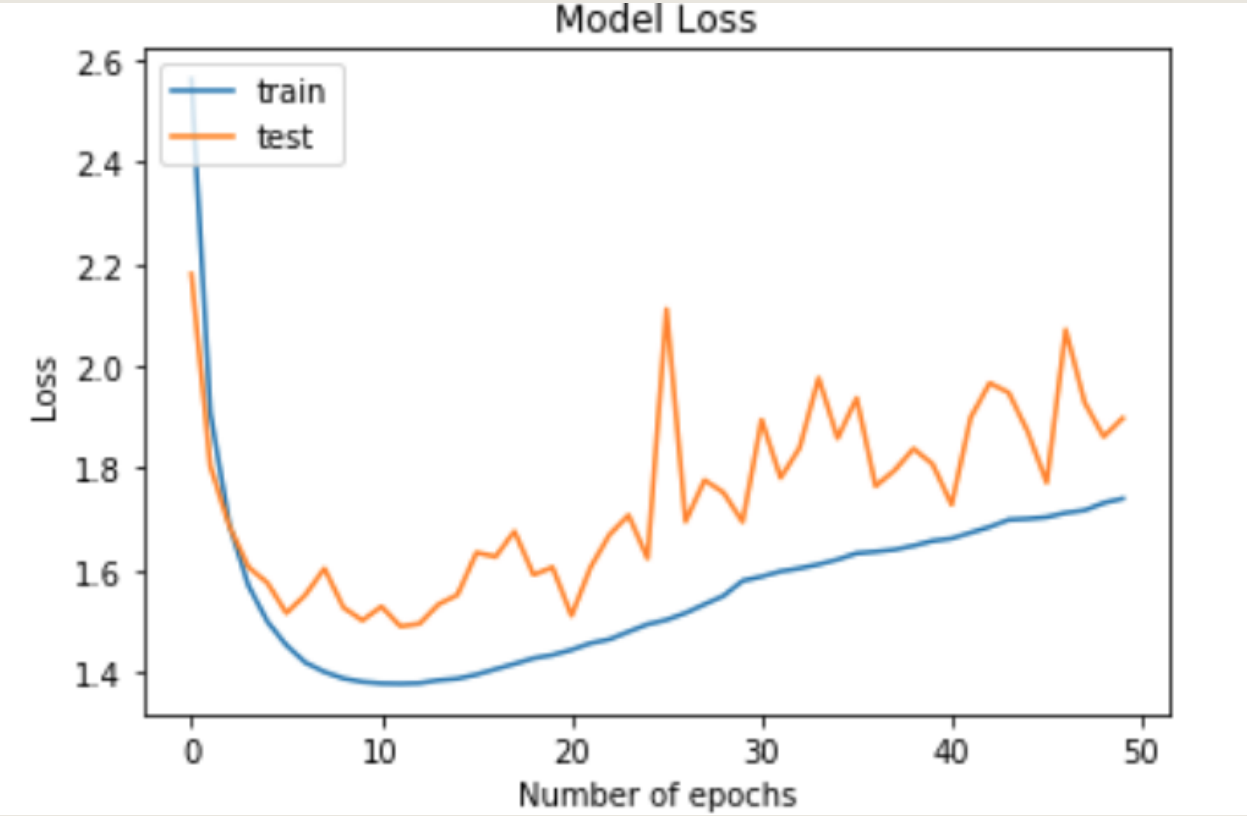
- ❑ What can you deduce from this **loss curve** generated by a neural network on a dataset?
- ❑ What do you think is the reason behind this issue (if there is one)?
- ❑ What do you suggest to address this issue (if there is one)?

PRACTICAL SCENARIO 4



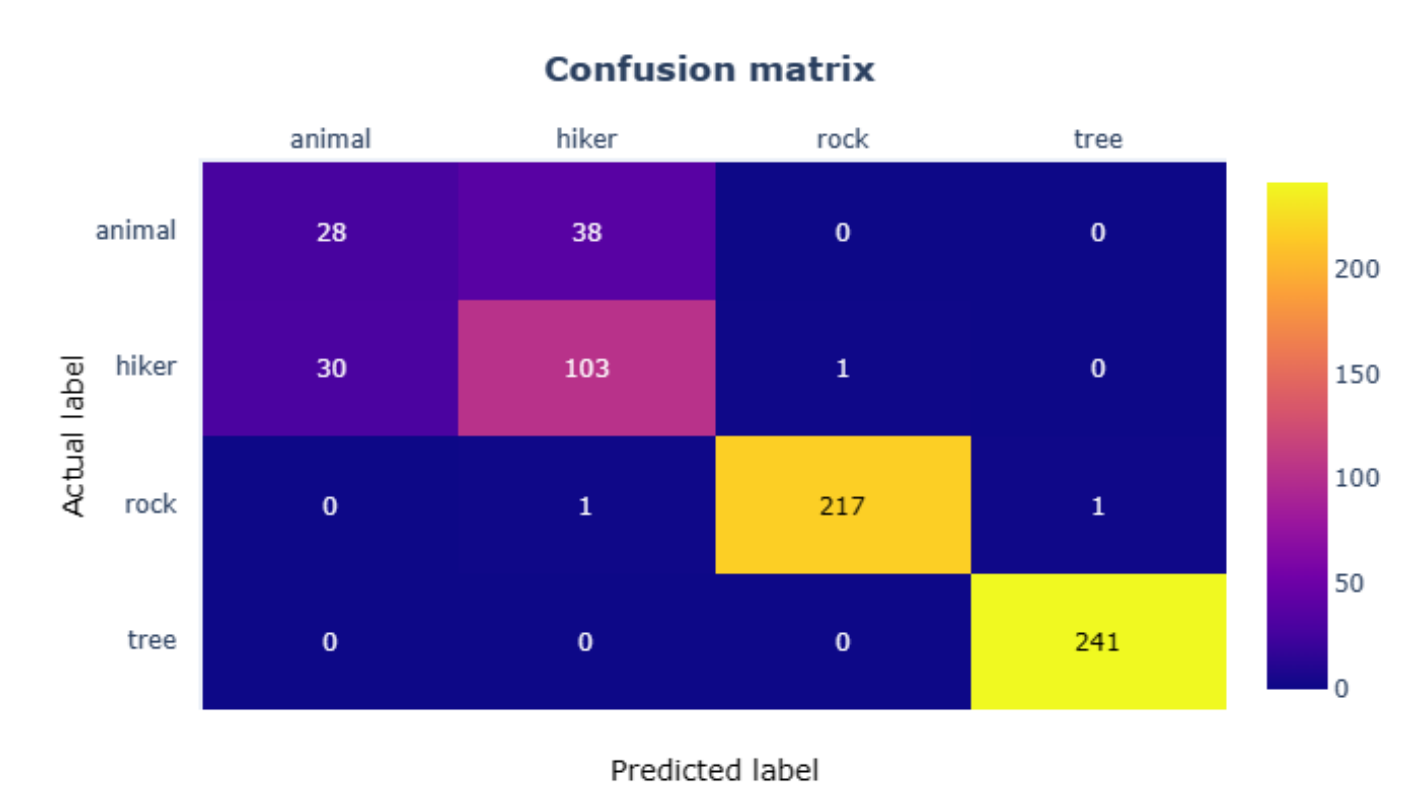
- What can you deduce from this **loss curve (against varying Training Set size)** generated by a polynomial regression model on a dataset?
- What do you think is the reason behind this issue (if there is one)?
- What do you suggest to address this issue (if there is one)?

PRACTICAL SCENARIO 5



- What can you deduce from this **loss curve** generated by a neural network on a dataset?
- What do you think is the reason behind this issue (if there is one)?
- Which **Hyperparameter** do you suggest to tune to address this issue?

PRACTICAL SCENARIO 6



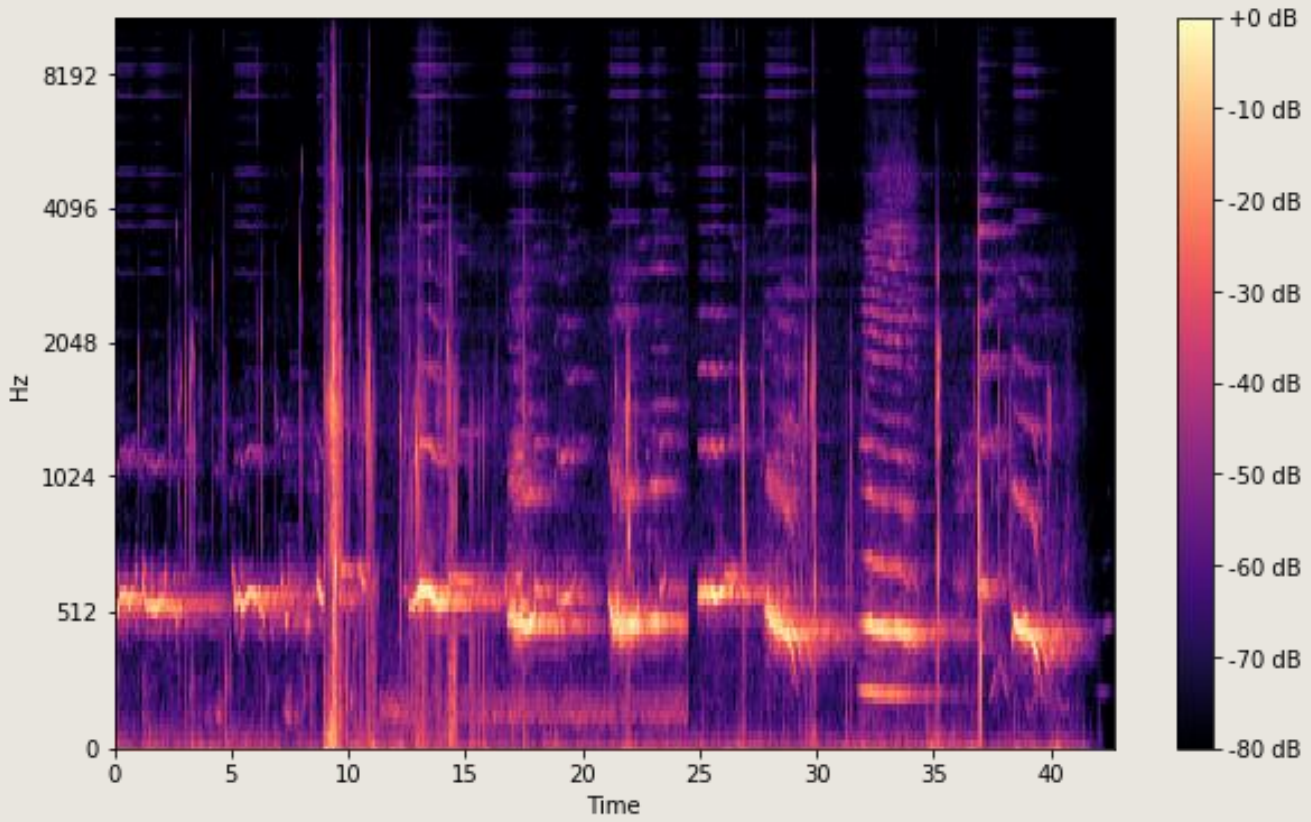
- What can you conclude about the data & the performance of the model from this **Confusion Matrix**?
- Which **Performance Metrics** would you like to use to evaluate the model for this task?
- What are some **techniques** that can be used to improve the performance of this model?

PRACTICAL SCENARIO 7

Here you can see some sample images from the **FER2013** Dataset for Facial Expression Recognition. It is one of the benchmark datasets for that task.

- ❑ What kind of **Data Augmentation** techniques would you like to use on this dataset?
- ❑ What kind of **Model** do you think would be suitable for this task?

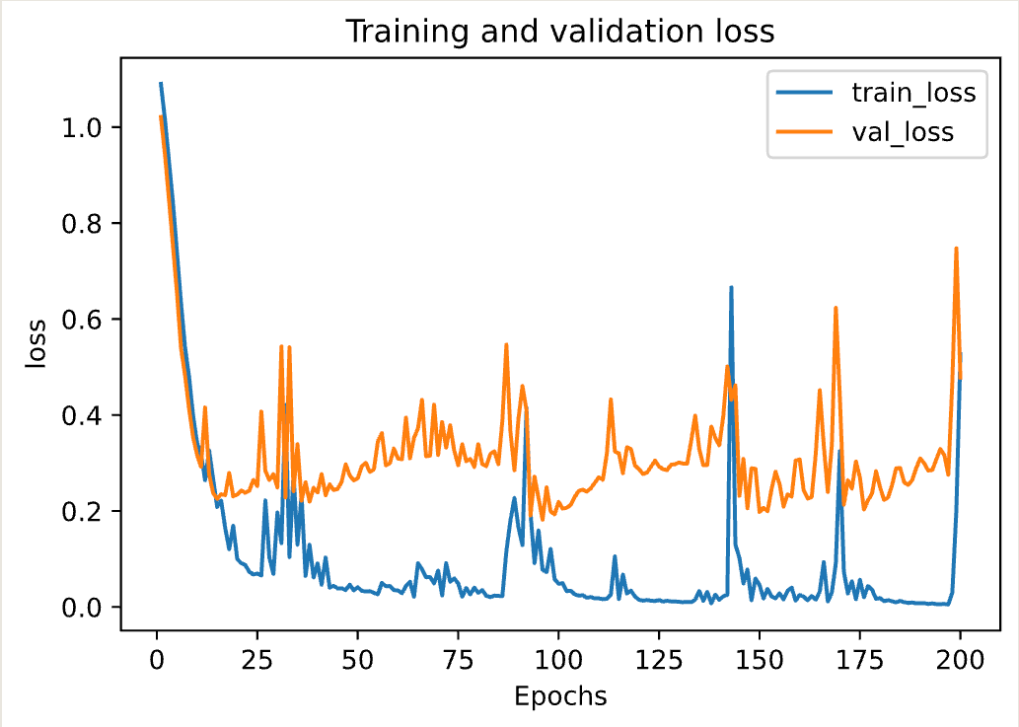
PRACTICAL SCENARIO 8



Here you can see a **Mel Spectrogram** generated from an Audio file.

- ❑ What kind of **Data Transformation** technique was used on the Frequency axis here?
- ❑ Would you use the same kind of **Data Augmentation** techniques as you suggested in the previous slide in this case too (since both are 2D data)? Explain your choice.

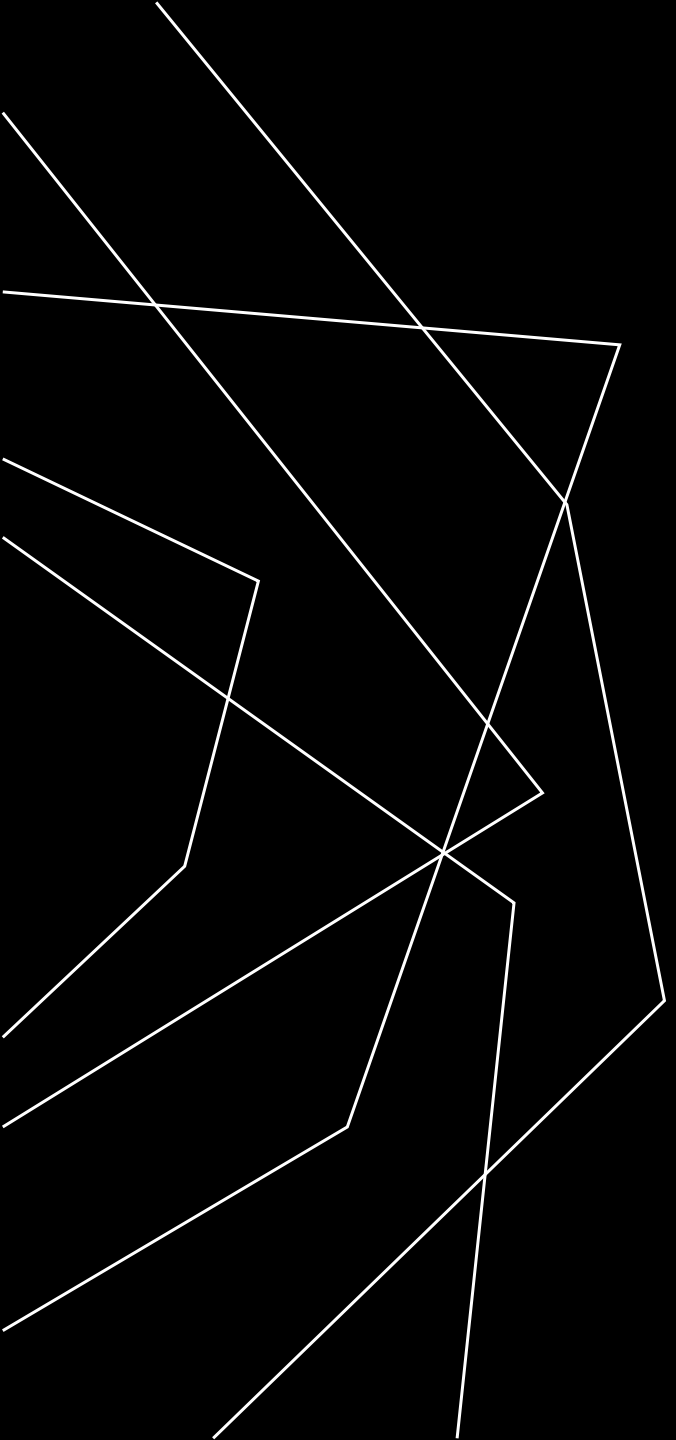
PRACTICAL SCENARIO 9



- In your opinion, which **Optimization Algorithm** may have been used to train the model that generated the loss curve above?
- If you want to make this curve smoother, what change would you make?
- Is it the sign of a bad model if it generates such a spiky loss curve like the one here?

REFERENCES

1. University of Toronto - CSC411/2515: Machine Learning and Data Mining (https://www.cs.toronto.edu/~rgrosse/courses/csc311_f20/)
2. Stanford cs231n Course Notes (https://cs231n.github.io/)
3. CMU's Introduction to Machine Learning (10-601) Lectures (https://www.cs.cmu.edu/%7Etom/10701_sp11/lectures.shtml)
4. Applied ML course at Cornell and Cornell Tech (https://github.com/kuleshov/cornell-cs5785-2020-applied-ml/)
5. Pattern Recognition and Machine Learning, Christopher Bishop
6. https://affine.ai/data-augmentation-for-deep-learning-algorithms/
7. Iwana, Brian Kenji and Seiichi Uchida. “Time Series Data Augmentation for Neural Networks by Time Warping with a Discriminative Teacher.” *2020 25th International Conference on Pattern Recognition (ICPR) (2020): 3558-3565*.
8. https://www.dkube.io/post/keep-your-models-accurate-through-monitoring



THANK YOU