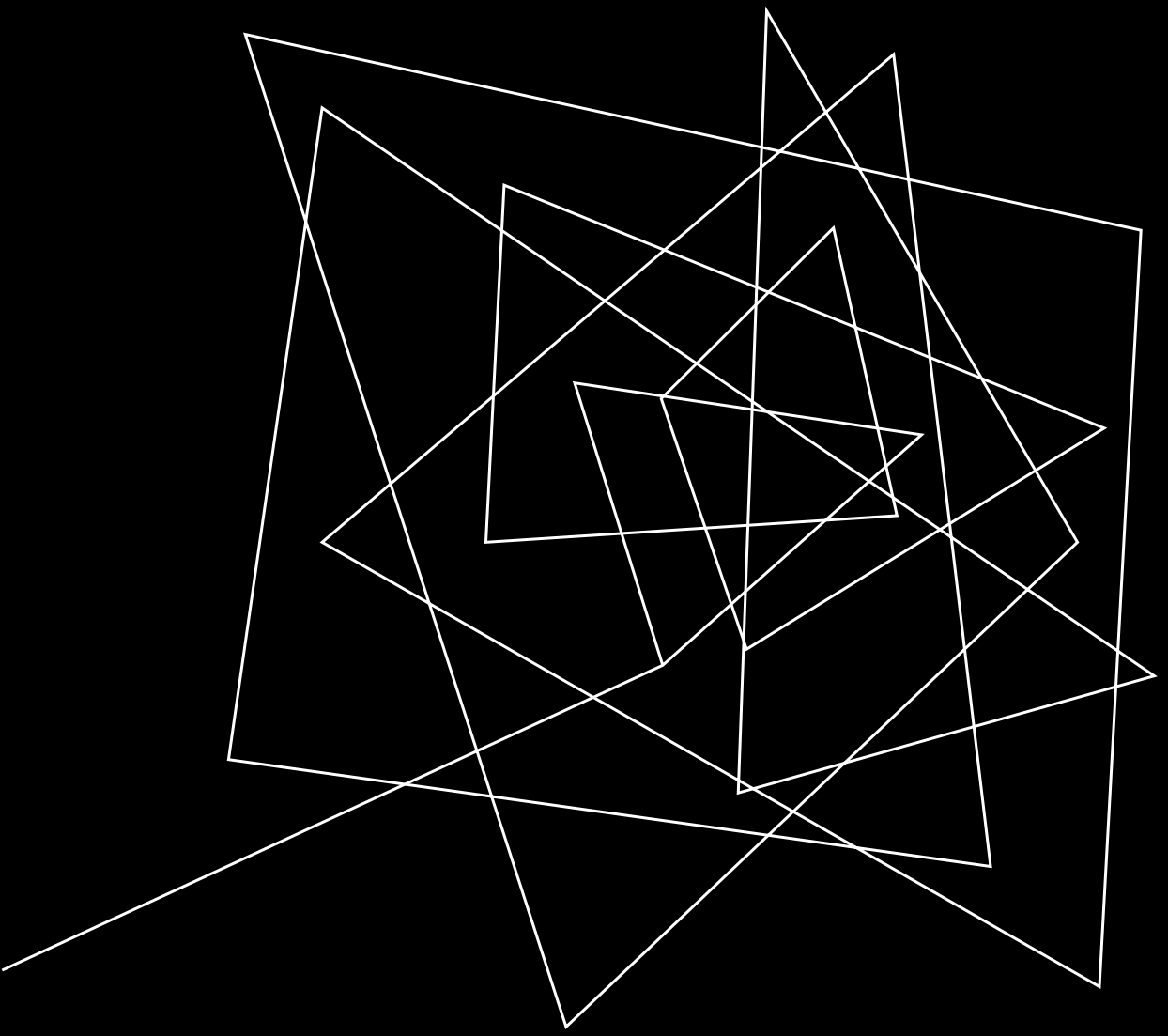
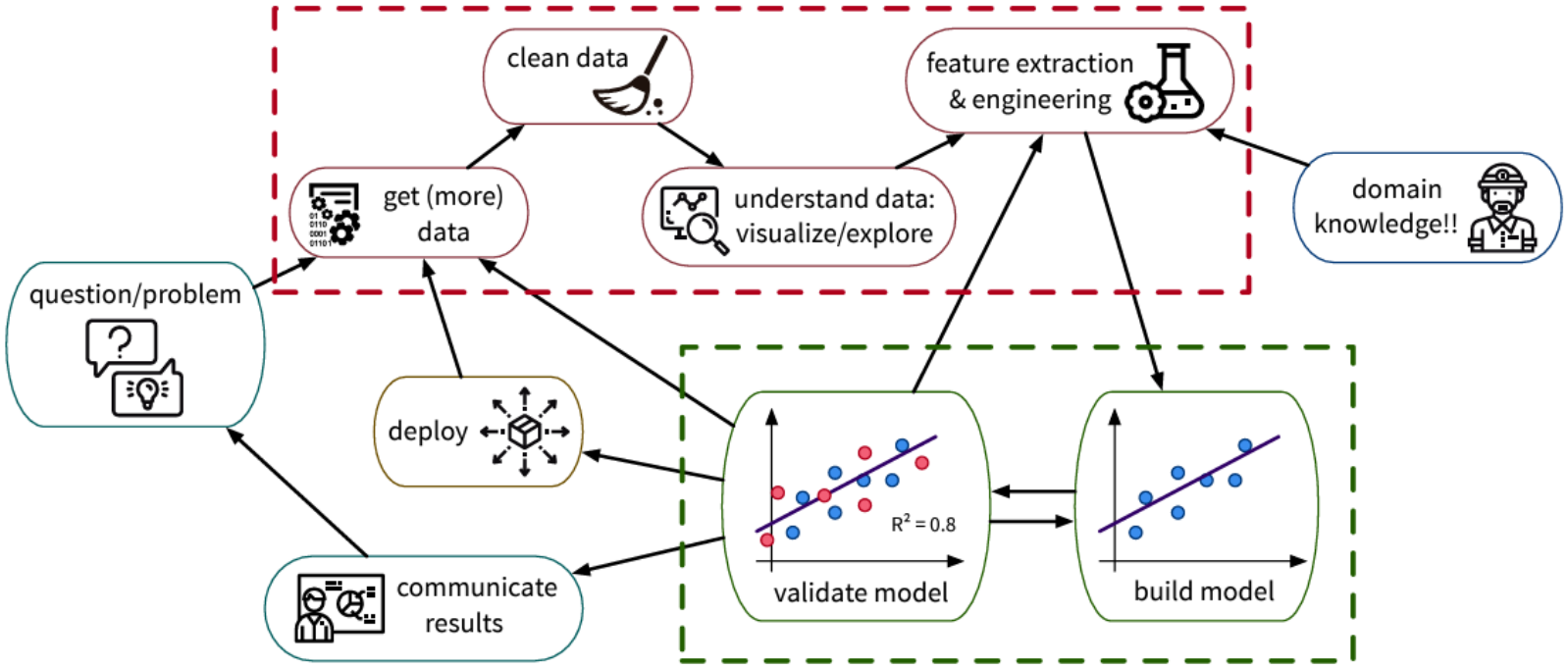


SESSION 4: FEATURE ENGINEERING AND MODEL EVALUATION

Feature engineering, types of feature engineering, and supervised ML evaluation metrics

Feature Engineering in Machine Learning

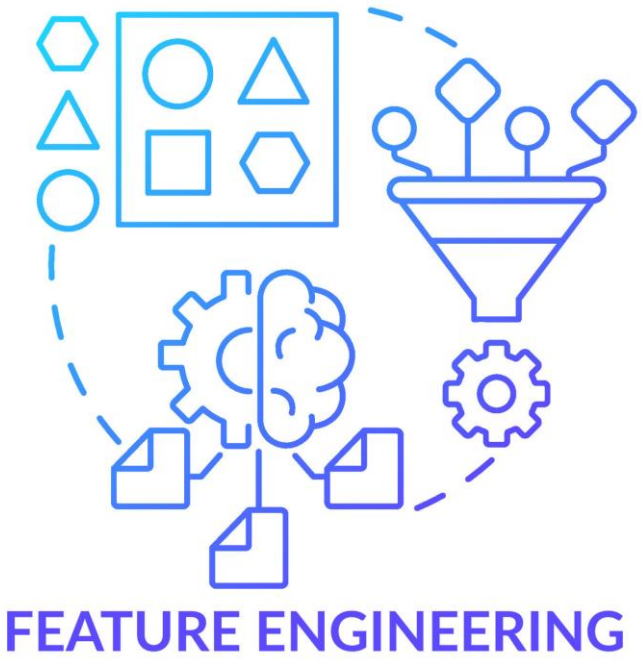
Feature engineering plays a pivotal role in the success of machine learning models. It involves the strategic refinement of input data to maximize a model's predictive capabilities. Essentially, it is the **art** and **science** of **crafting the right set of features to empower models** to uncover meaningful patterns and relationships within the data.



Source: A Practitioner's Guide to Machine Learning

Feature Engineering in Machine Learning

Feature engineering is the process of meticulously **selecting**, **transforming**, and **creating features** to **amplify** the **effectiveness** and **efficiency** of machine learning models. It's similar to preparing the raw ingredients before cooking a gourmet dish – the **quality** and **arrangement** of ingredients **significantly influence** the final **outcome**. In the realm of machine learning, features are the **ingredients** and **feature engineering** is the cooking expertise that brings out the best flavors in our models.
By carefully engineering features, practitioners can enhance the model's ability to discern/extract/understand patterns, capture relevant information, and ultimately make more accurate predictions.



Why Feature Engineering?
<Impact of Features on Model Performance>

Impact of Features on Model Performance: The features used in a machine learning model serve as the foundation upon which the entire predictive framework is built. The choice and quality of these features have a profound impact on the model's ability to generalize patterns from the data. In essence, the features act as the informational building blocks that guide the model in understanding the underlying relationships within the dataset.

Why Feature Engineering?
<"Garbage In, Garbage Out" Principle>

"**Garbage In, Garbage Out**" Principle: A fundamental principle in the realm of machine learning is the "Garbage In, Garbage Out" principle. This concisely/briefly captures the essence of the relationship between data quality and model outcomes. If the features provided to a model are of low quality or lack relevance to the prediction task, the model's predictions are likely to be inaccurate or unreliable.

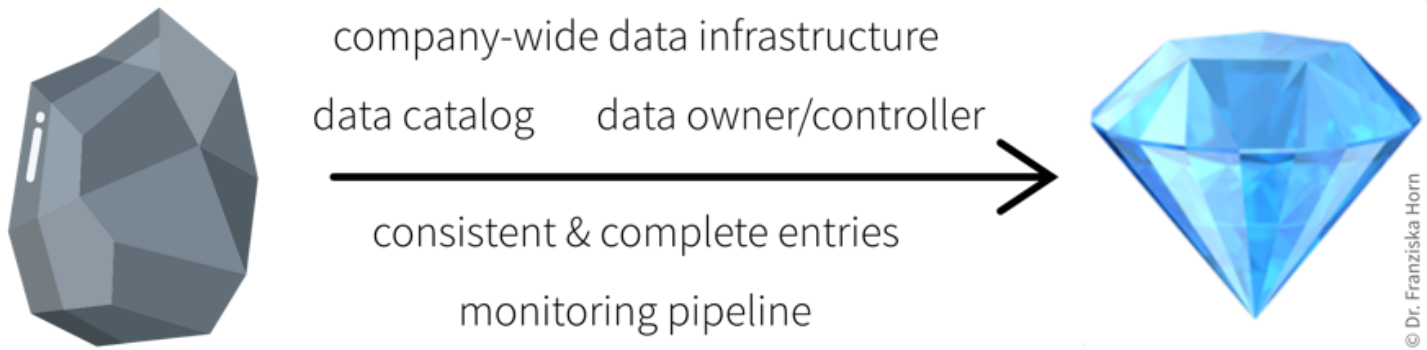
Note 1: Both quality and quantity matter

Note 2: It is not a one-off task, but rather a continuous process

Why Feature Engineering?

Quality of Features Directly Affects Predictions: The quality of predictions is directly tied to the quality of features. Well-crafted features can illuminate intricate patterns in the data, allowing the model to make informed decisions. Conversely, inadequate features can obscure meaningful information or introduce noise, hindering the model's ability to discern relevant patterns.

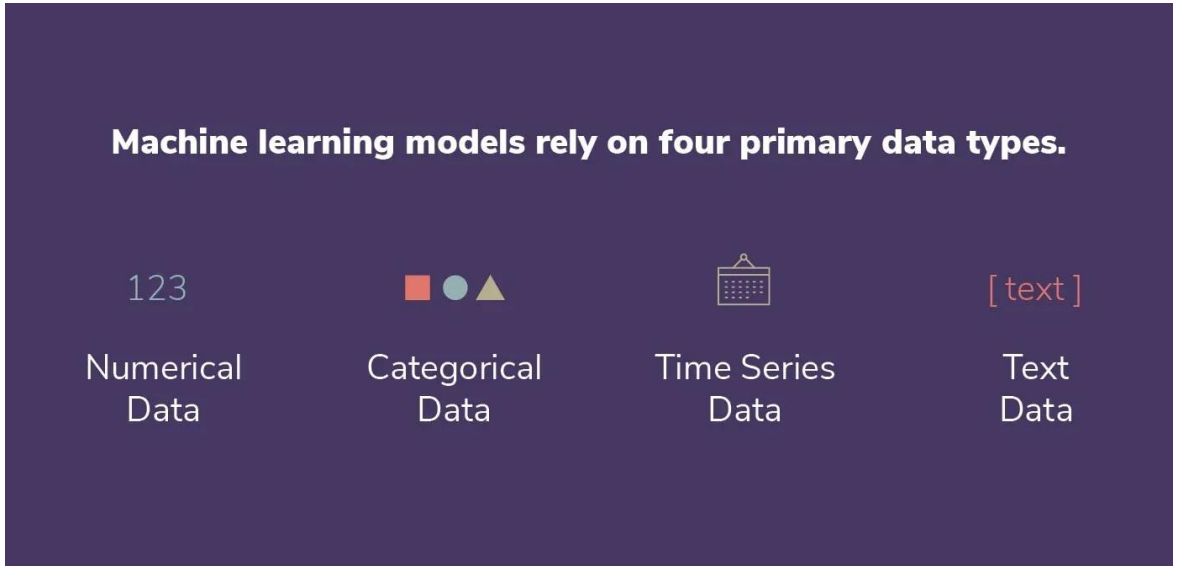
Note: keep in mind that the care and consideration given to features significantly influence the overall performance and reliability of machine learning models.



Source: A Practitioner's Guide to Machine Learning

Types of Features

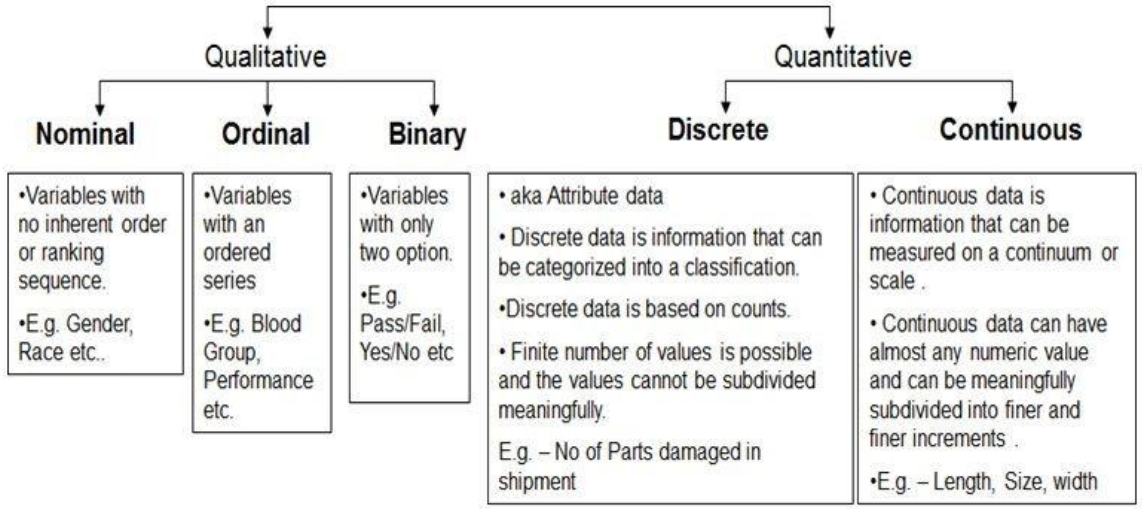
In the diverse landscape of machine learning, data comes in various forms, each requiring unique handling and consideration. Understanding the types of features is fundamental to effective feature engineering.



Source: datarobot.com

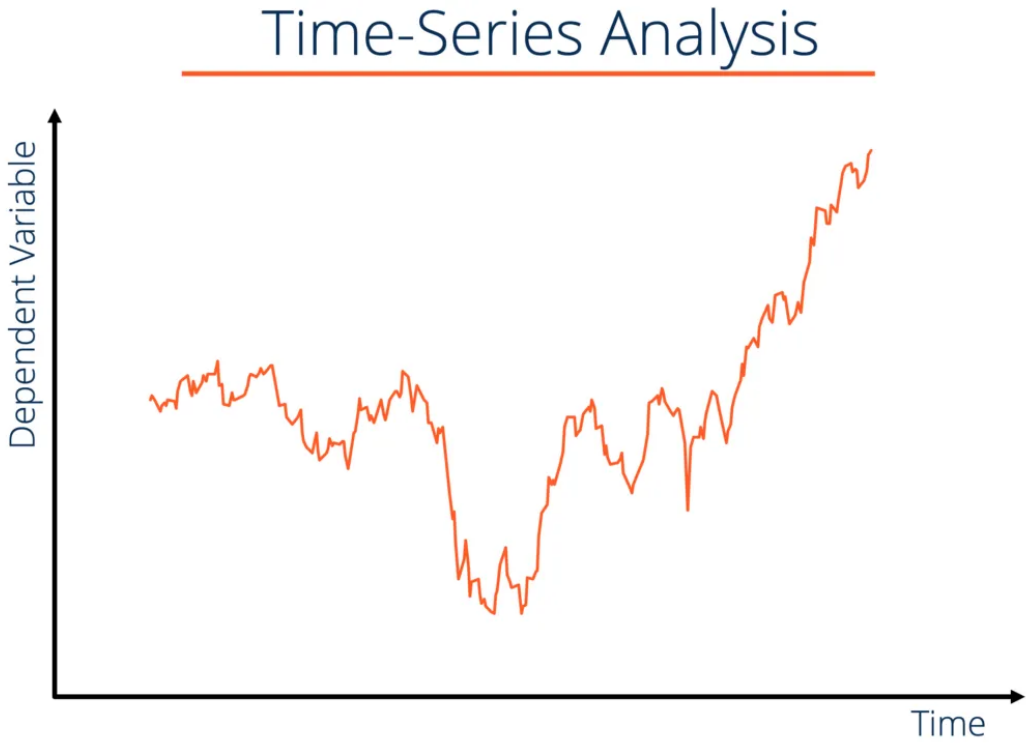
Types of Features: Categorical and Numerical Data

1. **Categorical Features:** Categorical features represent qualitative data and can take on discrete values that often denote categories or labels. Examples include colors, gender, or types of products. Proper encoding techniques, such as one-hot encoding or label encoding, are essential to translate categorical features into a format understandable by machine learning algorithms.
2. **Numerical Features:** Numerical features represent quantitative data and take on numeric values. These could include measurements such as height, weight, or temperature. Numerical features may further be divided into two subtypes: discrete (countable) and continuous (infinitely divisible). Scaling and normalization are common preprocessing steps for numerical features to ensure their comparable impact on the model.



Types of Features: Text and Time-series Features

3. **Text Features:** Text features involve unstructured data, such as sentences, paragraphs, or entire documents. NLP (Natural Language Processing) techniques are often employed to extract meaningful information from text features. Methods like tokenization, TF-IDF (Term Frequency-Inverse Document Frequency), and word embeddings are commonly used to convert text into a format suitable for machine learning models.
4. **Time Series Features:** Time series features involve data points collected or recorded over time. Examples include stock prices, temperature readings, or daily sales figures. Handling time series features requires specialized techniques like lag features, rolling statistics, and time-based aggregations to capture temporal patterns and trends effectively.



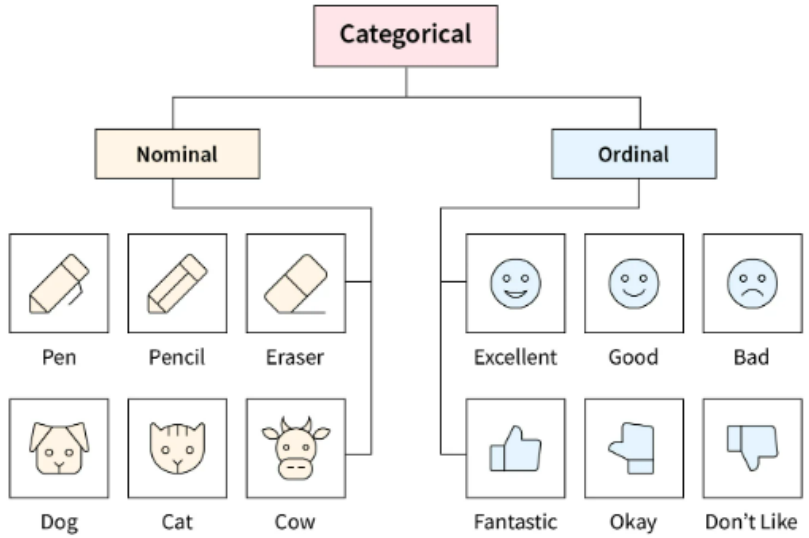
Feature Engineering: Missing Data

Techniques for Handling Missing Data:

- **Imputation:** Imputation involves filling in missing values with estimated or calculated values. Common imputation methods include mean, median, or mode imputation, where missing values are replaced with the mean, median, or mode of the observed values. This helps maintain the overall distribution of the feature.
- **Deletion:** Deletion involves removing instances or features with missing values. While straightforward, this approach can lead to a loss of valuable information, especially if the missing values are not randomly distributed. Careful consideration is needed to assess the impact of deletion on model performance.
- **Advanced Methods - Predictive Modeling for Imputation:** Leveraging advanced techniques, such as predictive modeling, offers a more sophisticated approach to handling missing data. This involves using the non-missing features to predict and impute missing values. Techniques like regression imputation or machine learning algorithms can be employed to create accurate predictions for missing values.

Feature Engineering: Handling/Encoding Categorical Data

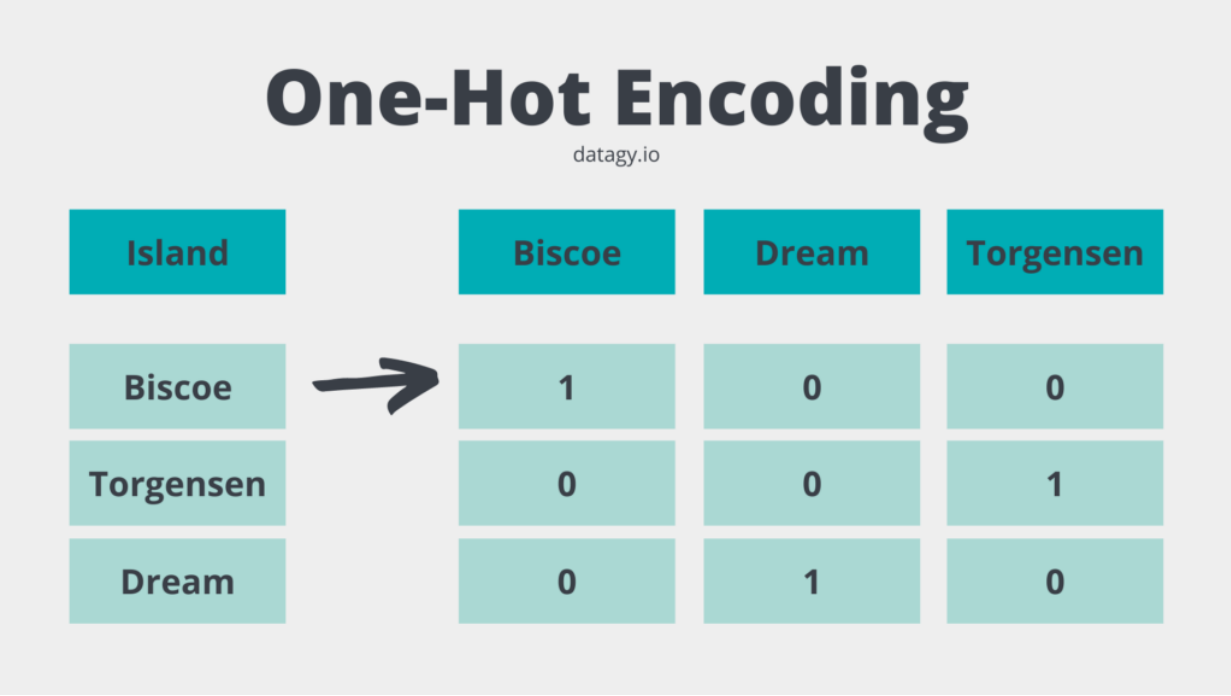
- **Definition:** Categorical encoding is the process of converting categorical data, which represents qualitative variables with discrete categories, into a numerical format that can be used by machine learning algorithms. Since many machine learning models require numerical input, encoding categorical variables is essential to ensure compatibility and effectiveness in the training process.
- **Question:** Why encoding categorical columns/data?
- **Answer:** Since most machine learning models only accept numerical variables, preprocessing the categorical variables becomes a necessary step. We need to convert these categorical variables to numbers such that the model is able to understand and extract valuable information.



Source: medium.com

Feature Engineering: Handling/Encoding Categorical Data <One Hot Encoding>

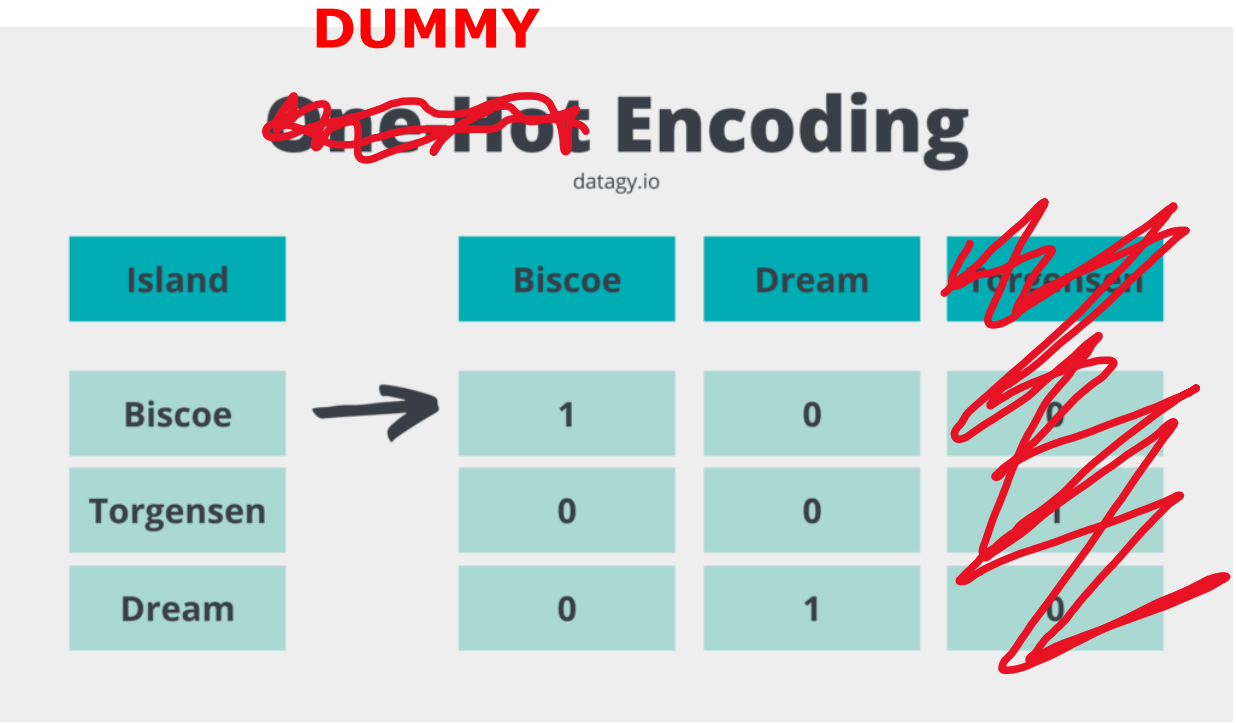
One-Hot Encoding is a popular technique for converting categorical variables into a binary matrix. Each category is transformed into a binary column, and only one of these columns is marked as "1" for the respective category while the others are set to "0". This method ensures that each category is represented independently, preventing the model from incorrectly assuming ordinal relationships between categories.



Source: datagy.io

Feature Engineering: Handling/Encoding Categorical Data <Dummy Encoding>

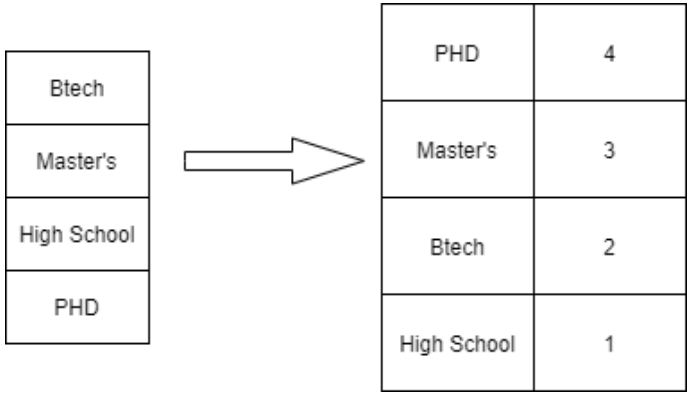
Dummy encoding technique which slightly differs from one-hot-encoding, dummy encoding uses N-1 features to represent N labels/categories.



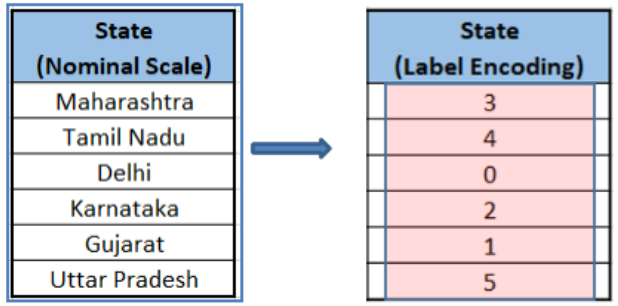
Source: datagy.io

Feature Engineering: Handling/Encoding Categorical Data <Label Encoding>

Label Encoding involves assigning a unique numerical label to each category. This method is suitable when there is an ordinal relationship between categories, as the numerical values can represent the order. **However**, caution is needed when using Label Encoding for non-ordinal categorical variables, as it might mislead the model into assuming a meaningful order.



Source: ai-ml-analytics.com



Source: mygreatlearning.com

Feature Engineering: Handling/Encoding Categorical Data <Target Encoding (Mean Encoding)>

Target Encoding (or Mean Encoding) involves replacing each category with the mean of the target variable for that category. This method is particularly useful when dealing with **high-cardinality categorical features***, as it provides a smooth mapping between the categorical variable and the target variable. However, it is crucial to avoid **data leakage** by calculating target means using only the training data.

id	job	job_mean	target
1	Doctor	0,50	1
2	Doctor	0,50	0
3	Doctor	0,50	1
4	Doctor	0,50	0
5	Teacher	1	1
6	Teacher	1	1
7	Engineer	0,50	0
8	Engineer	0,50	1
9	Waiter	1	1
10	Driver	0	0

Source: towardsdatascience.com

High-cardinality categorical features are those that have a **large number of unique values**, such as **product IDs**, **zip codes**, or **names**. These features can pose challenges for data encoding, as they can create a large number of dummy variables, increase memory usage, and reduce model performance.

Feature Engineering: Handling/Encoding Categorical Data <DEMO>

Demo: [Session 4 – Feature Engineering and Model Evaluation]

id	color
1	red
2	blue
3	green
4	blue

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Source: datacamp.com

Feature Engineering: Feature Scaling

Feature scaling is a crucial preprocessing step in machine learning, ensuring that numerical features contribute equally to model training. It involves transforming the range of features to a standardized scale, preventing certain features from dominating due to their inherent scale.

	Age	EstimatedSalary
count	400.000000	400.000000
mean	37.655000	69742.500000
std	10.482877	34096.960282
min	18.000000	15000.000000
25%	29.750000	43000.000000
50%	37.000000	70000.000000
75%	46.000000	88000.000000
max	60.000000	150000.000000

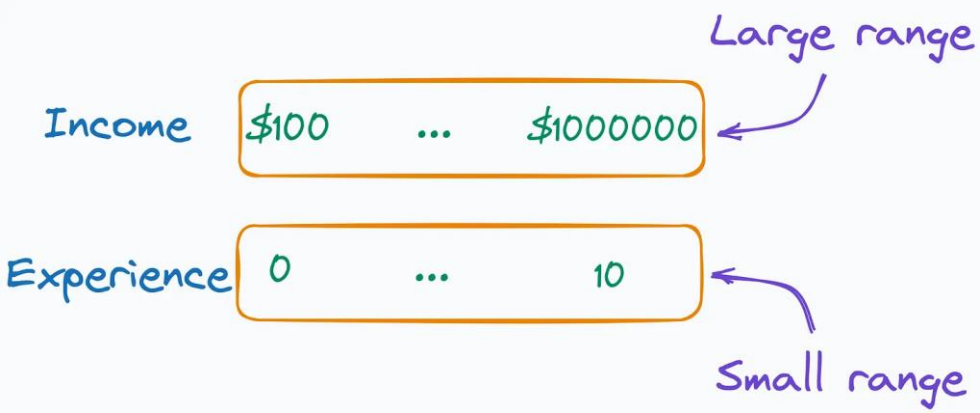
Before Min-Max Scaling

	Age	EstimatedSalary
count	400.000000	400.000000
mean	0.467976	0.405500
std	0.249592	0.252570
min	0.000000	0.000000
25%	0.279762	0.207407
50%	0.452381	0.407407
75%	0.666667	0.540741
max	1.000000	1.000000

After Min-Max Scaling

Feature Engineering: Why Feature Scaling?

Feature scaling is commonly used to improve the performance and stability of ML models.
This is because it **scales** the data to a **standard range**. This **prevents** a specific **feature** from having a **strong influence** on the model's **output**.

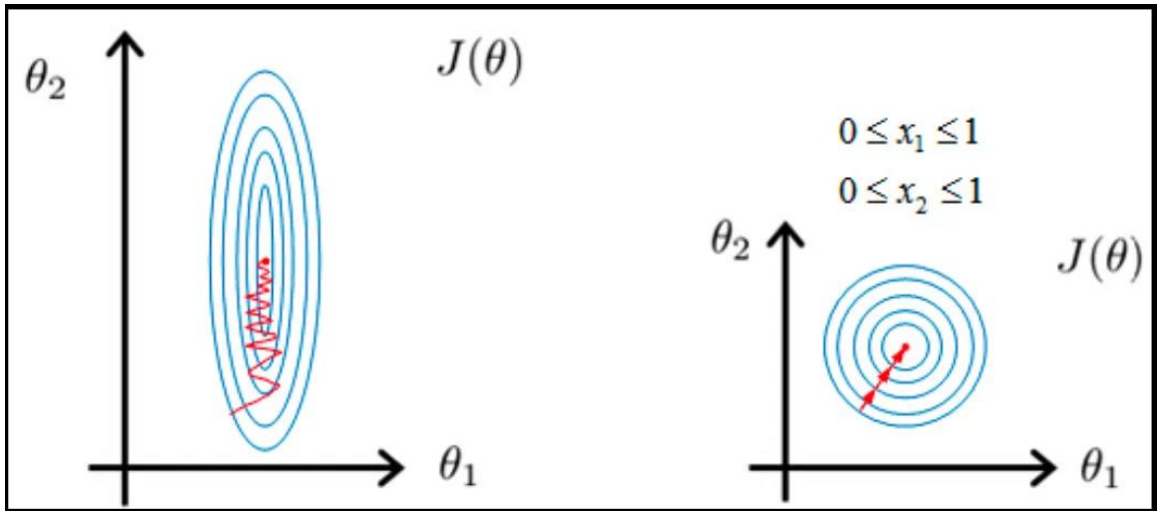


Source: dailydoseofds.com

For instance, in the image above, the scale of Income could massively impact the overall prediction. Scaling both features to the same range can mitigate this and improve the model's performance.

Feature Engineering: Why Feature Scaling?

Feature scaling is essential for algorithms that rely on distance measures, such as k-nearest neighbors or support vector machines. It helps in achieving better convergence for gradient-based optimization algorithms and ensures that the model is not biased towards features with larger magnitudes.

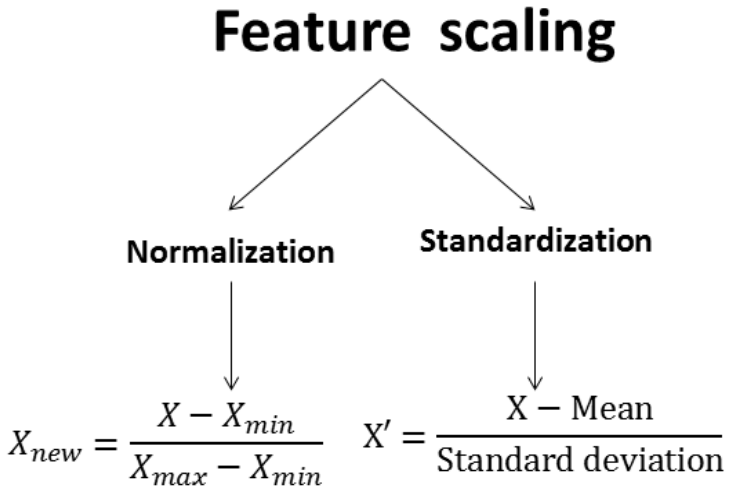


Gradient Descent: Attaining Global minimum before and after scaling
Source: medium.com

Feature Engineering: Feature Scaling <Normalization vs. Standardization>

Two common techniques for feature scaling are **normalization** and **standardization**, which include methods such as **Min-Max Scaling** and **Z-score Standardization**.

- **Normalization:** Scales the features to a range between 0 and 1. It's particularly useful when the features have varying ranges, and you want to bring them to a uniform scale. Normalization is performed using the formula: **X_scaled** = $(X - X_{min}) / (X_{max} - X_{min})$
- **Standardization:** Standardizes the features to have a mean of 0 and a standard deviation of 1. This is effective when the features follow a Gaussian distribution. Standardization is performed using the formula: X_**scaled** = (X – mean(X))/(std(X))



Feature Engineering: Feature Scaling
<Normalization: Min-Max Scaling>

Min-Max Scaling, a type of normalization, transforms the features to a specific range, usually between 0 and 1. This is achieved by using the formula:

$$X' = \frac{X - \min(X)}{\max(X) - \min(X)}$$

Note: Min-Max Scaling is sensitive to outliers, and it's important to handle extreme values appropriately.

Feature Engineering: Feature Scaling
<Standardization: Z-Score>

Z-score Standardization transforms the features to have a mean of 0 and a standard deviation of 1. This method is effective for features that follow a Gaussian distribution. The formula for Z-score Standardization is:

$$X' = \frac{X - \text{mean}(X)}{\text{std}(X)}$$

Note: Z-score Standardization is less sensitive to outliers compared to Min-Max Scaling.

Feature Engineering: Feature Scaling is Not Always Necessary

Feature Scaling is NOT Always Necessary

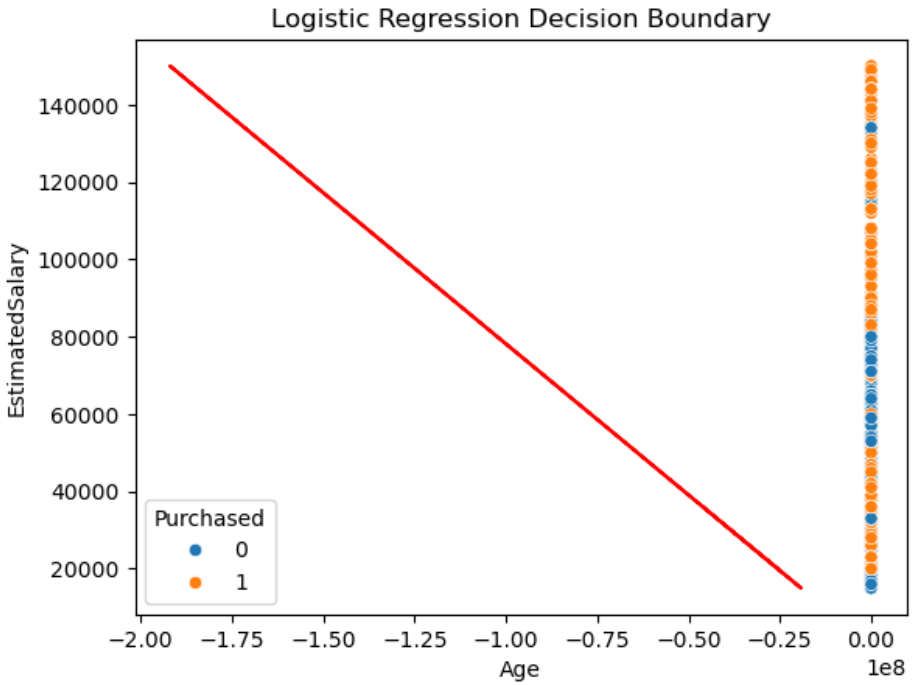
blog.DailyDoseofDS.com

Algorithm	Test Set Classification Performance: Without Feature Scaling	With Feature Scaling	Scaling Required?	
Logistic Regression	0.53	0.70	YES	Better performance with feature scaling
Support Vector Classifier	0.72	0.94	YES	
MLP Classifier	0.73	0.89	YES	
kNN Classifier	0.66	0.93	YES	
Decision Tree	0.83	0.83	NO	Same performance irrespective of feature scaling
Random Forest	0.91	0.91	NO	
Gradient Boosting	0.86	0.86	NO	
Naive Bayes	0.75	0.75	NO	

Source: dailydose.com

Feature Engineering: Feature Scaling <DEMO>

Demo: [Session 4 – Feature Engineering and Model Evaluation]



Before Min-Max Scaling

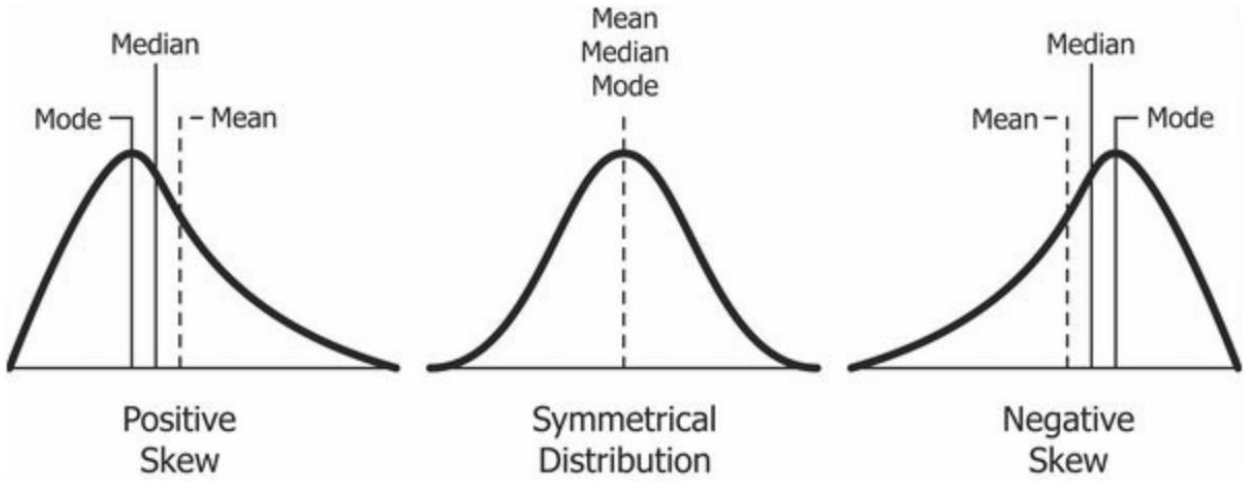
After Min-Max Scaling

Feature Engineering: Feature Transformation <Why Applying Feature Transformation>

Skewed Data Problem

One problem that can arise when working with datasets is skewed data. Skewed data occurs when the distribution of a variable is not evenly distributed, resulting in an unbalanced data set. This can negatively affect the performance of machine learning algorithms because they cannot learn patterns from data effectively.

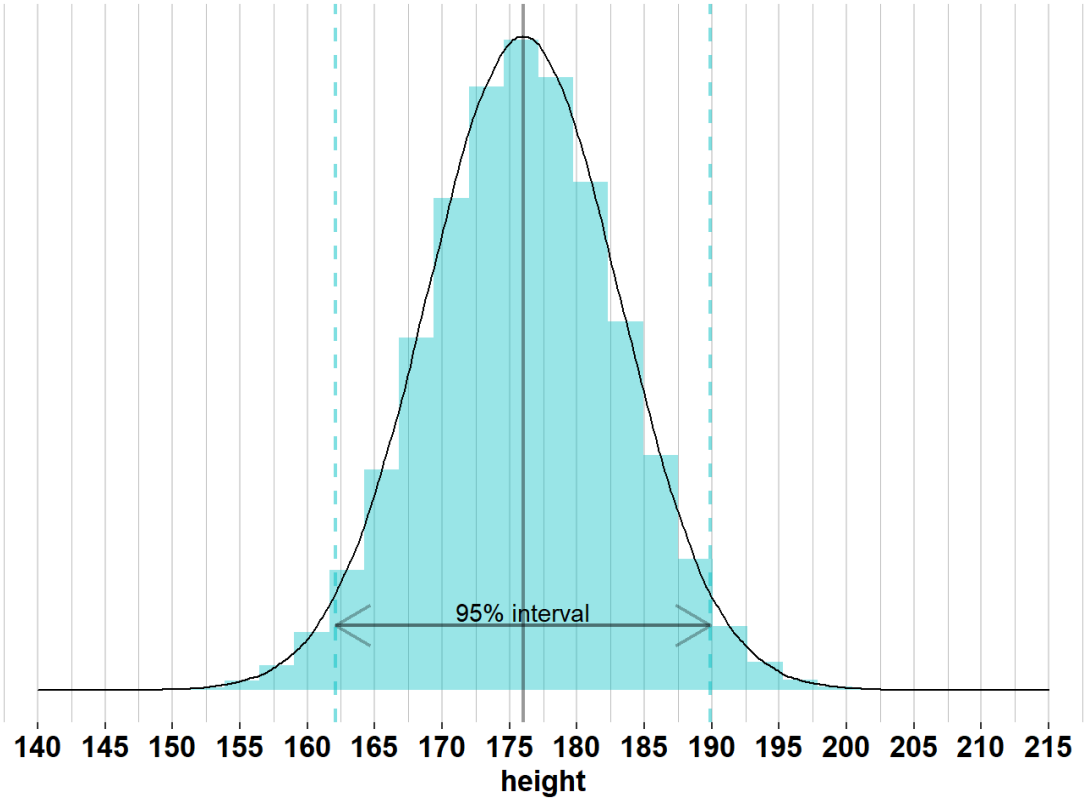
Note: Most of ML algorithms assume that the data is normally distributed!



Source: wikipedia.com

Feature Engineering: Feature Transformation
<Why Applying Feature Transformation>

Like **Linear** and **Logistic regression**, some data science models assume that the variables follow a **normal distribution**. More likely, variables in real datasets will follow a skewed distribution. By applying some transformations to these skewed variables, we can map this skewed distribution to a normal distribution to increase the performance of our models. As we know, **Normal Distribution** is a very important distribution in Statistics, which is key to many statisticians for solving problems in statistics. Usually, the data distribution in Nature follows a Normal distribution like - age, income, height, weight, etc. But the features in the real-life data are not normally distributed. However, it is the best approximation when we are unaware of the underlying distribution pattern.



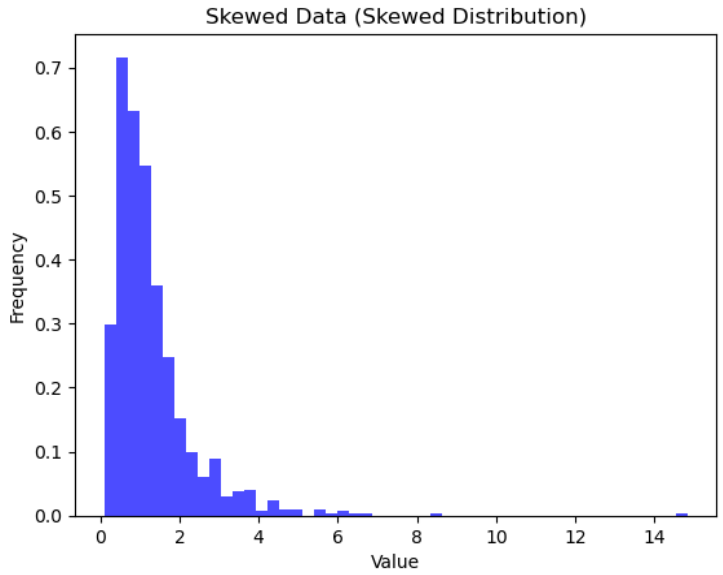
Source: javatpoint.com

Source: bookdown.org

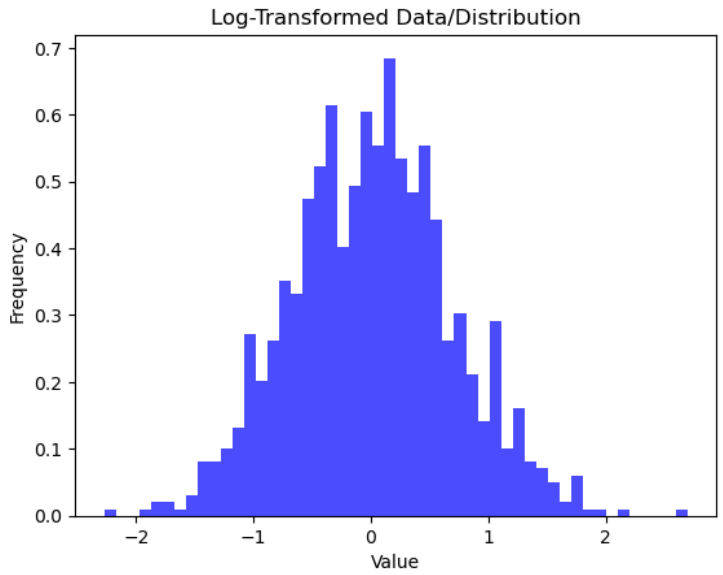
Feature Engineering: Feature Transformation <Logarithmic Transformation>

Logarithmic transformation is applied to features to reduce the impact of extreme values and make the distribution more symmetric. This is especially beneficial when dealing with skewed data. Logarithmic transformation is implemented using the logarithmic function.

Note: This transformation is not applied to those features which have negative values



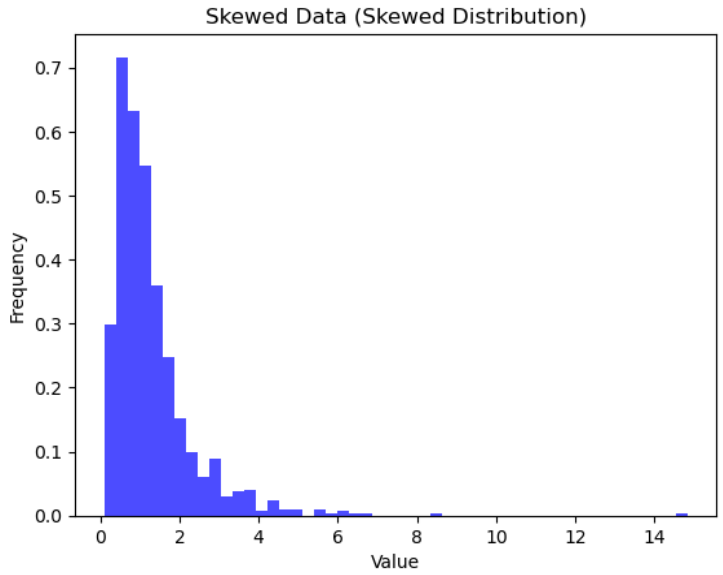
Before Log-Transformation



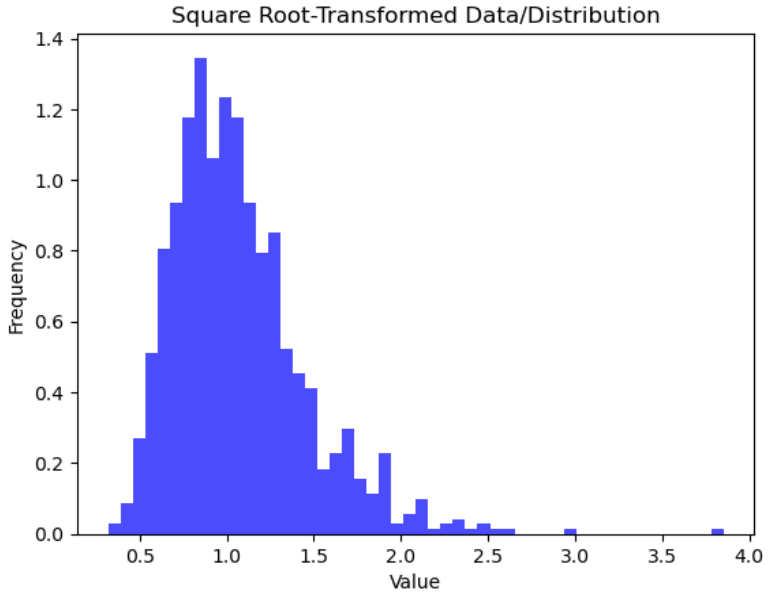
After Log-Transformation

Feature Engineering: Feature Transformation <Square Root Transformation>

Square Root Transformation: This transformation is defined **only** for **positive numbers**. This can be used for reducing the skewness of right-skewed data. This transformation is weaker than Log Transformation.



Before Sqrt-Transformation

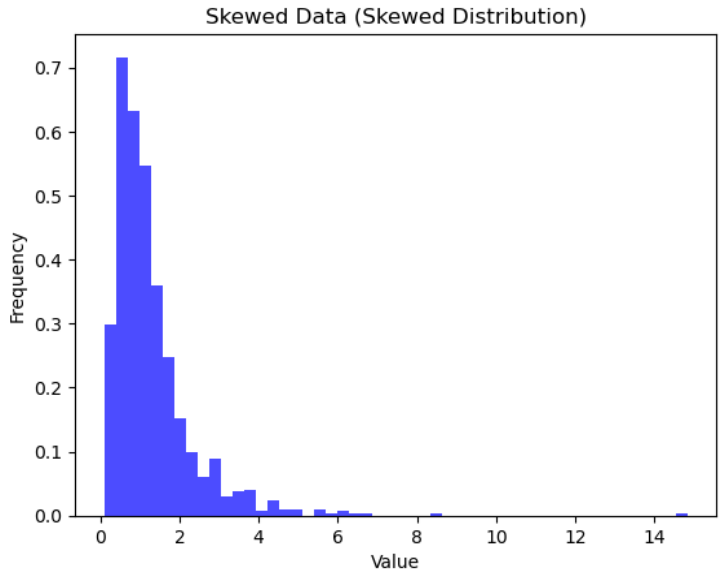


After Sqrt-Transformation

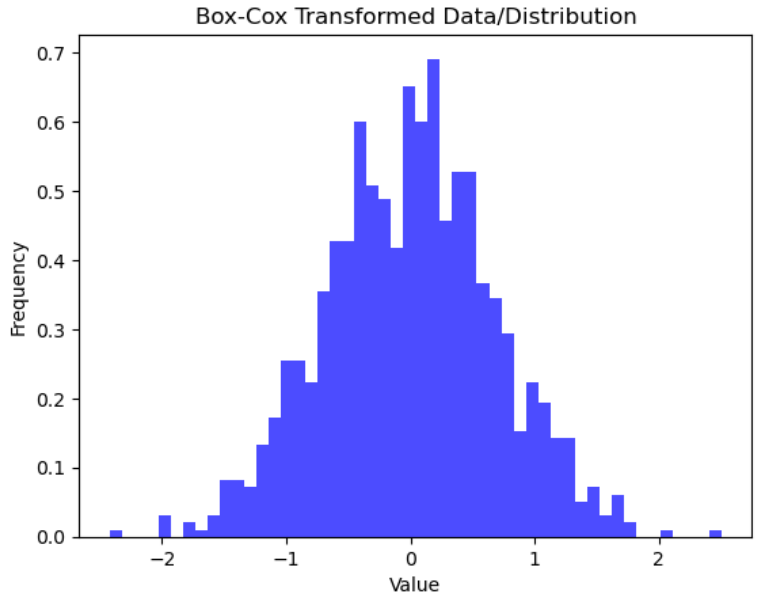
Feature Engineering: Feature Transformation <Box-Cox Transformation>

The Box-Cox transformation is a family of power transformations that includes logarithmic transformation as a special case. It is useful for stabilizing variance and making the data more closely approximate a normal distribution. The Box-Cox transformation requires the data to be strictly positive; if the data contains zeros or negative values, alternative transformations might be considered.

$$y_i^{(\lambda)} = \begin{cases} \dfrac{y_i^{\lambda} - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln(y_i) & \text{if } \lambda = 0, \end{cases}$$



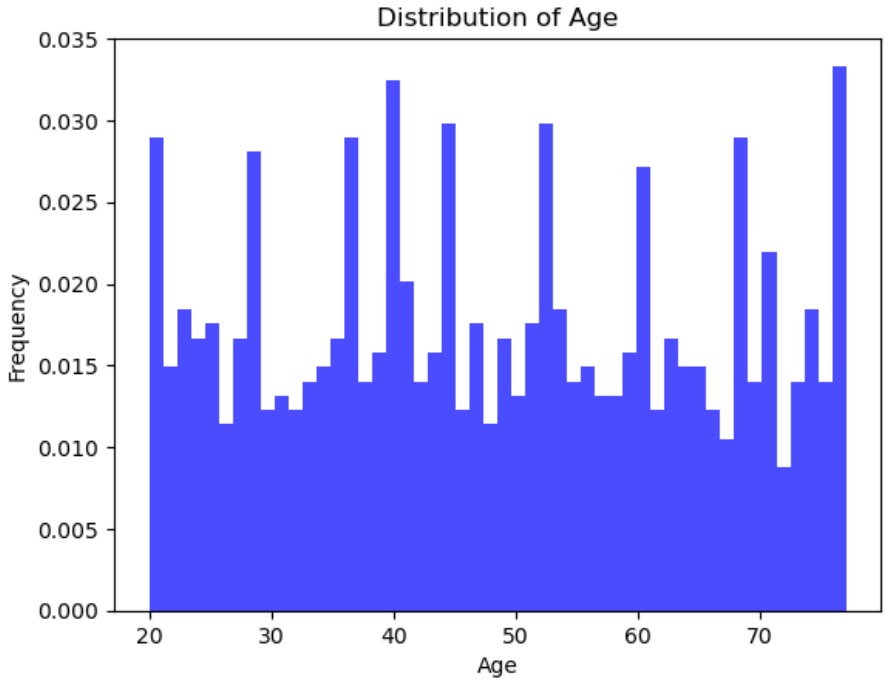
Before Box-Cox-Transformation

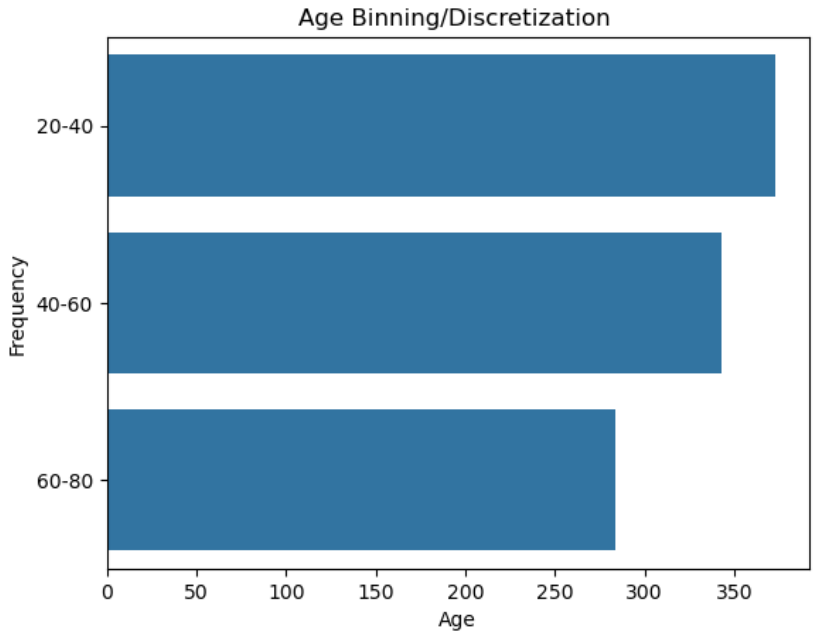


After Box-Cox-Transformation

Feature Engineering: Feature Transformation <Feature Binning>

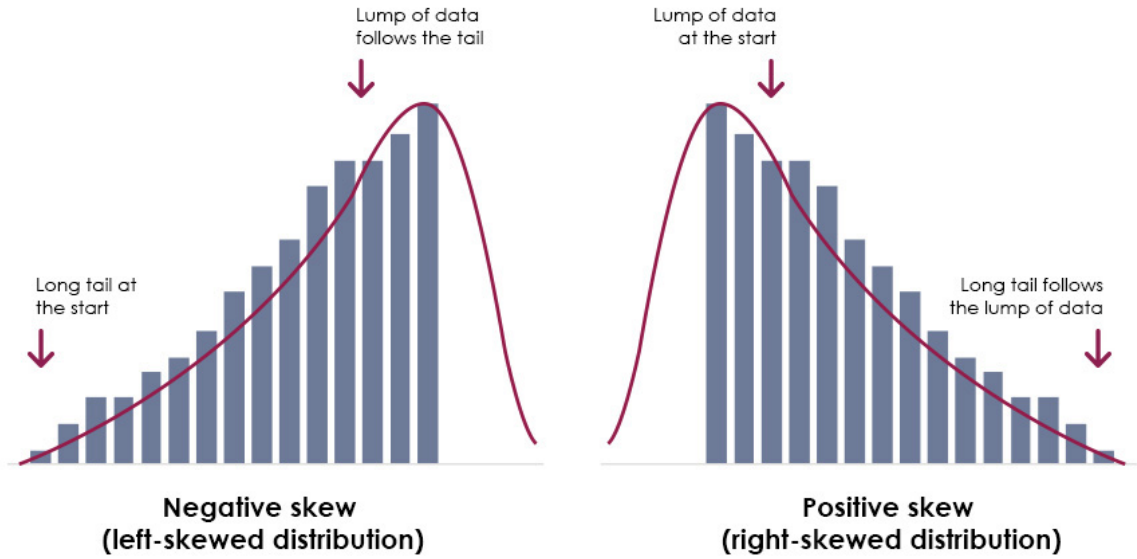
Feature binning, also known as discretization or bucketing, is a data preprocessing technique that involves grouping continuous or numerical features into discrete bins or intervals. The primary goal of feature binning is to convert numerical data into categorical data by partitioning the range of values into a set of predefined bins.





Feature Engineering: Feature Transformation <DEMO>

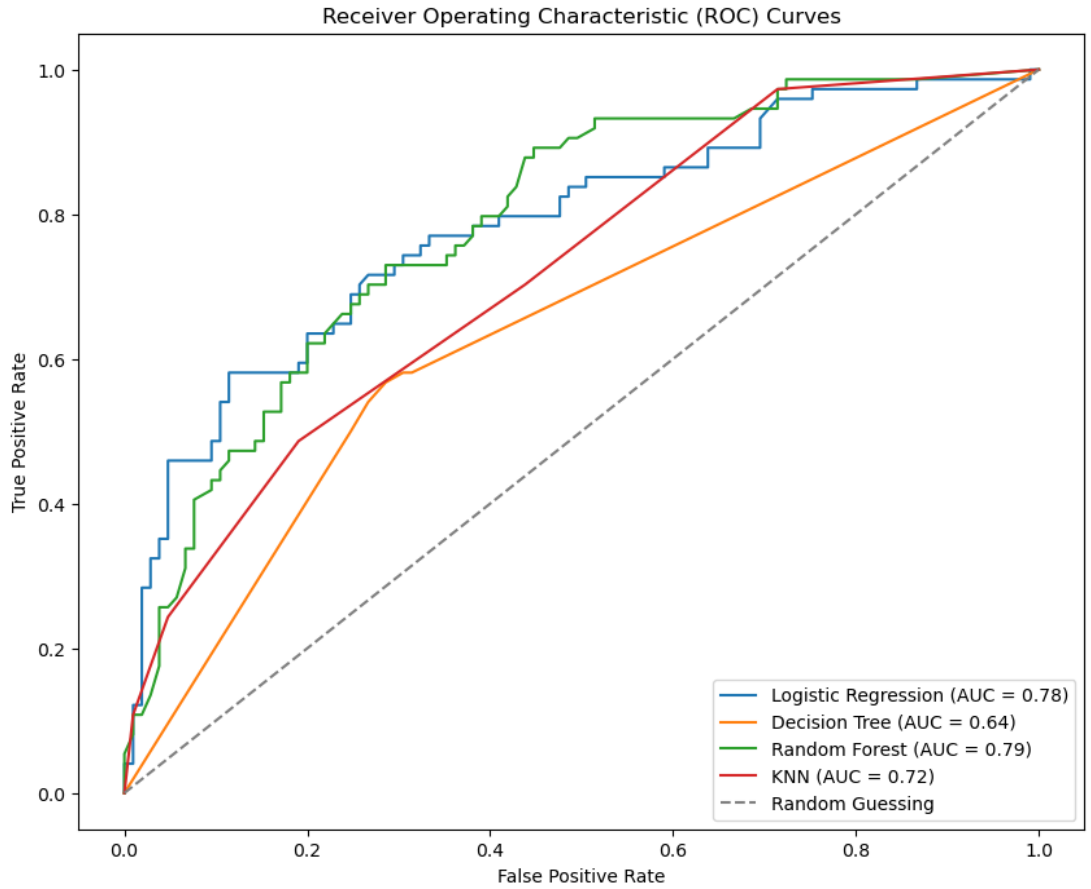
Demo: [Session 4 – Feature Engineering and Model Evaluation]



Source: cambridgemaths.org

Model Evaluation Metrics for Supervised Learning Models

Since in supervised learning problems, we know the ground truth (the actual/true outputs), we can objectively evaluate different models and benchmark them against each other.



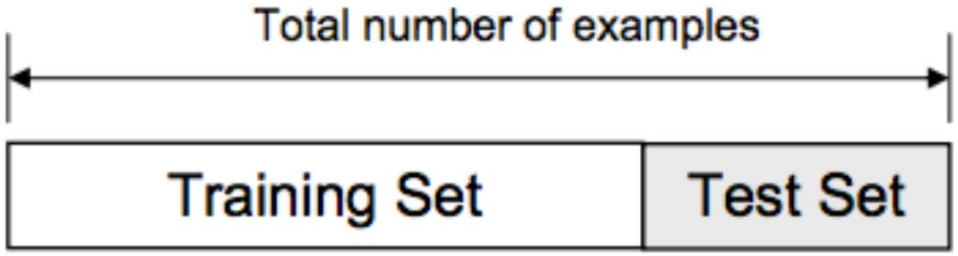
Model Evaluation Metrics for Supervised Learning Models

Is this a good model for the task?

I.e., does the model generate reliable predictions for new data points?

- Split the data into training and test sets to be able to get a reliable estimate of how the model will later perform when applied to new data points that it wasn't trained on.
- Quantify the quality of the model's predictions on the test set with a suitable evaluation metric (depending on the problem type, i.e., classification or regression).

Note: Compare the model to a **'stupid baseline'** that predicts the mean (→ regression) or most frequent class (→ classification).



Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: Accuracy>

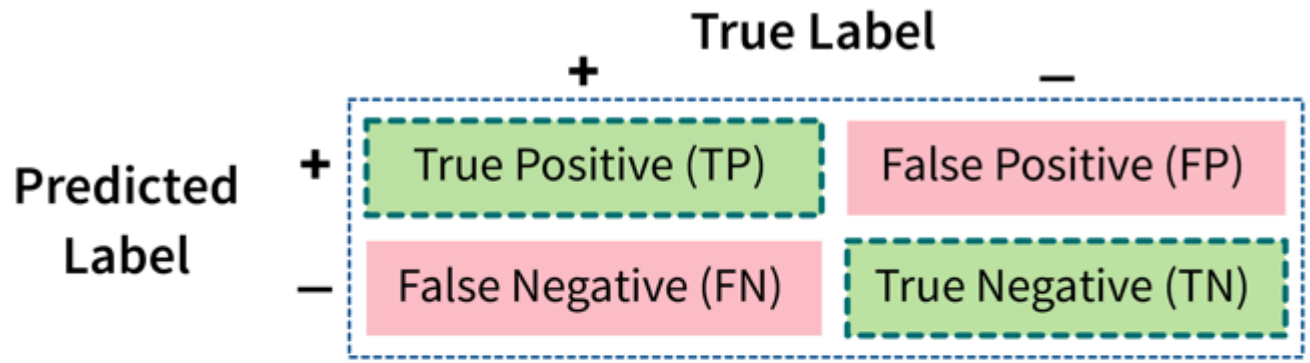
Actual Output	Predicted Output
0	1
0	0
1	1
1	1
0	0
1	0
1	0

The simplest way to evaluate this ML model is to compute the fraction of the correct predictions out of the number of predictions, as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

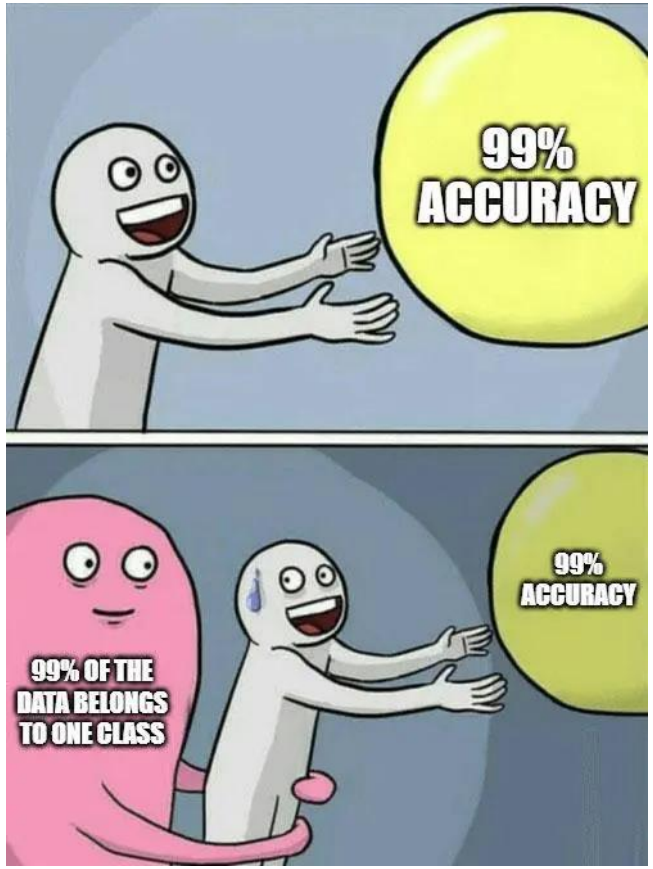
Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: Accuracy>

Confusion Matrix: A **confusion matrix** is a **table** used in **classification** to evaluate the performance of a machine learning model. It provides a summary of the predicted and actual classifications, breaking down the results into four categories: **true positives** (correctly predicted positive instances), **true negatives** (correctly predicted negative instances), **false positives** (incorrectly predicted as positive), and **false negatives** (incorrectly predicted as negative). This matrix is particularly useful for assessing the accuracy and effectiveness of a classification model by revealing the types and frequencies of classification errors.



$$\frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{n}$$

Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: Accuracy>



Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: Accuracy>

Some potential **problems** with relying only on **accuracy**

Problem	Description	Examples
Imbalanced Datasets	In datasets where one class significantly outweighs the others (class imbalance), a model can achieve high accuracy by simply predicting the majority class most of the time.	In a medical dataset where only 5% of patients have a rare disease, a model that always predicts "no disease" would still achieve 95% accuracy.
Misleading in Rare Events	In applications where detecting rare events is crucial (e.g., fraud detection, disease diagnosis), accuracy can be misleading. A high accuracy score may not reflect the model's ability to correctly identify these critical cases. Accuracy may not reflect the model's ability to detect rare, but important, occurrences.	In fraud detection, a model that misses rare instances of fraud (false negatives) could have high accuracy but would be ineffective.

Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: Recall and Precision>

Recall and precision are two important metrics used in the evaluation of classification models. They are often used together, especially when dealing with imbalanced datasets.

Recall (Sensitivity or True Positive Rate):

- **Definition:** Recall measures the ability of a classification model to capture and correctly identify all relevant instances in a dataset. It is the ratio of true positives to the sum of true positives and false negatives.
- **Formula:** Recall = True Positives / (True Positives + False Negatives)
- **Interpretation:** A high recall indicates that the model is good at identifying most of the positive instances in the dataset. However, it may also lead to a higher number of false positives.

$$Recall = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Negative(FN)}$$

Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: Recall and Precision>

Recall and precision are two important metrics used in the evaluation of classification models. They are often used together, especially when dealing with imbalanced datasets.

Precision (Positive Predictive Value):

- **Definition:** Precision measures the accuracy of the positive predictions made by a model. It is the ratio of true positives to the sum of true positives and false positives.
- **Formula:** Precision = True Positives / (True Positives + False Positives)
- **Interpretation:** A high precision indicates that when the model predicts a positive instance, it is likely to be correct. However, it may miss some positive instances, resulting in a lower recall.

$$Precision = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Positive(FP)}$$

Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: F1-Score>

- **Recall** focuses on minimizing false negatives, making sure that all actual positive instances are identified by the model.
- **Precision** focuses on minimizing false positives, ensuring that the positive predictions made by the model are accurate.
- The trade-off between recall and precision is often application-dependent. In some cases, you may want to prioritize recall over precision, and vice versa. The **F1 Score**, which is the harmonic mean of precision and recall, is a metric that combines both and is useful for finding a balance between the two when needed:

$$\text{F1 Score} = \frac{2}{\left(\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}\right)}$$

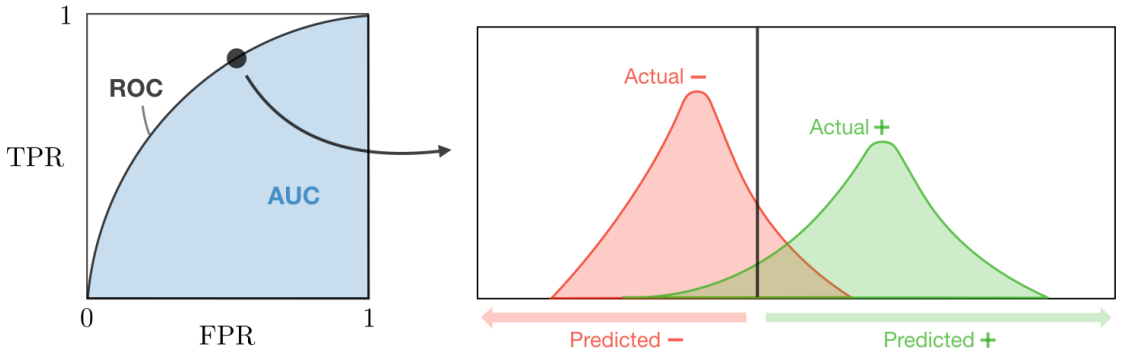
$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: ROC-AUC>

ROC: The receiver operating curve, also noted as ROC, is the plot of TPR versus FPR by varying the threshold. These metrics are summed up below:

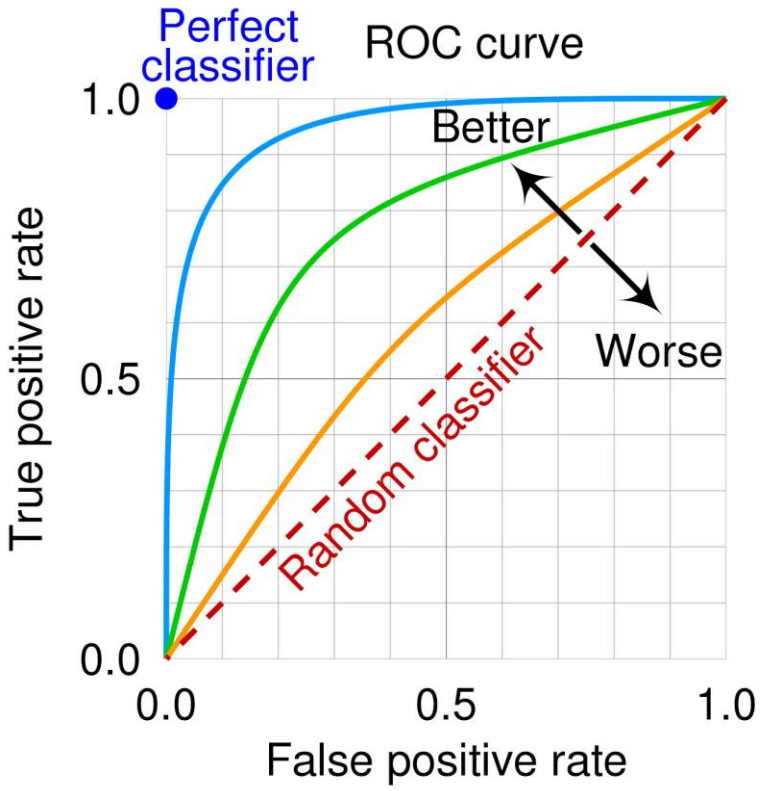
Metric	Formula	Equivalent
True Positive Rate TPR	$\frac{TP}{TP+FN}$	Recall, sensitivity
False Positive Rate FPR	$\frac{FP}{TN+FP}$	1-specificity

AUC: The area under the receiving operating curve, also noted AUC or AUROC, is the area below the ROC as shown in the following figure:



Source: https://stanford.edu/

Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: ROC-AUC>



Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: DEMO>

Example Scenario:

Let's say we have a binary classification problem for a medical test that identifies a disease:

- Out of 100 actual cases of the disease:
  - The test correctly identifies 80 as positive (True Positives).
  - The test incorrectly identifies 20 as negative (False Negatives).
- Out of 100 non-cases (healthy individuals):
  - The test correctly identifies 90 as negative (True Negatives).
  - The test incorrectly identifies 10 as positive (False Positives).

Find the confusion matrix and compute the accuracy, recall, precision, and specificity!!

Model Evaluation Metrics for Supervised Learning Models: <Regression Evaluation Metrics>

These regression metrics serve different purposes in evaluating the performance of regression models. **MAE**, **MSE**, and **RMSE** focus on the accuracy of predictions, while **R^2 (R-squared)** assesses the overall goodness of fit of the model. Depending on the specific characteristics of the problem, different metrics may be more appropriate for evaluation.

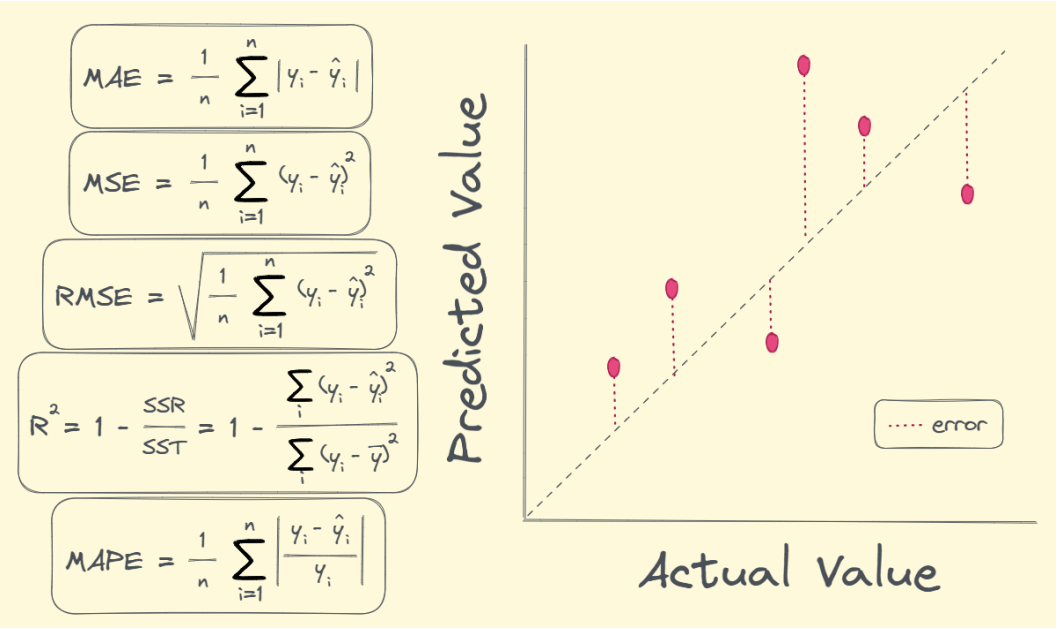
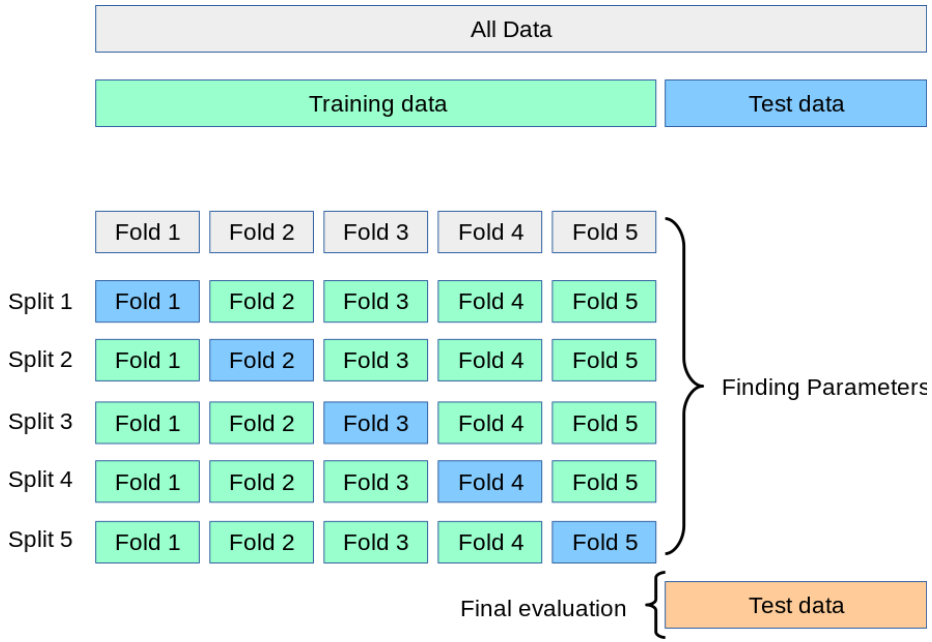


Fig. Regression evaluation metrics

Model Evaluation Metrics for Supervised Learning Models

K-fold cross-validation: One of the techniques to evaluate machine learning models on limited data samples and one of the employed techniques to detect overfitting and how well a model will generalize to new, unseen data.

- **Idea:** The k-fold cross-validation (CV) refers to the **k** number of the folds or subset division of the dataset (The model is then trained and evaluated **k** times, using each fold as the testing set once and the remaining **k-1** folds as the training set). It is generally a technique used in machine learning to evaluate the performance of the models on the entire dataset, it ensures that every data point from the dataset has the chances of appearing in both training and testing sets, and thus it is one of the best approaches to evaluate the models' performance.



Model Evaluation Metrics for Supervised Learning Models <DEMO>

Demo: [Session 4 – Feature Engineering and Model Evaluation]