Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant

Model Evaluation

Evaluation metrics, cross-validation methods.

AGENDA

1. Metrics for Evaluation
2. Cross-Validation Techniques
3. Hands-On Activity: Model Evaluation and Cross Validation

What are examples of good vs. bad predictions you've seen in real life?

The purpose of machine learning models is to **analyze data**, **identify patterns**, and **make predictions or decisions** based on that data without being explicitly programmed for specific tasks.

These models "learn" from historical data to solve various problems, such as:

- **Classifying** objects or categories (e.g., spam vs. not spam emails)
- **Predicting** outcomes (e.g., stock prices, customer behavior)
- **Detecting patterns** or anomalies (e.g., fraud detection)

The overall goal is to **create models** that can **generalize well** and make accurate predictions on new, unseen data.

Why Evaluate Models?

Model evaluation is **critical** before deployment because it ensures that the model can perform its task accurately and reliably in real-world settings.

Without proper evaluation, we risk deploying a model that could cause unintended consequences, especially in sensitive or high-stakes areas.

Let’s look at a couple of relatable examples to see why this is important:

1. **Spam Detection:**

   If a spam detection model is inaccurate, it might incorrectly *label important emails as spam* (false positives) or miss actual spam emails (false negatives).

   Consequence: While inconvenient, a few missed emails won’t cause major harm. Here, **accuracy** is important, but *we may tolerate* a small margin of error.

2. **Medical Diagnoses:**

   Imagine a model predicting the **likelihood of disease** based on patient data. If this model is not evaluated thoroughly, it might *misdiagnose patients*, leading to missed treatments (false negatives) or unnecessary stress and procedures (false positives).

   Consequence: In medical contexts, errors can have *serious implications* on health and lives, so even a small mistake rate is critical to address.

Evaluation allows us to identify any weaknesses, fine-tune its accuracy, and decide if it's ready for real-world use.

1. Metrics for Evaluation

Evaluation metrics are tools that allow us to measure how well a machine learning model performs its tasks.

They provide insight into a model's **accuracy**, **reliability**, and **ability to generalize to new data**, helping us identify where improvements are needed before deployment.

The choice of evaluation metrics depends on the type of task the model is performing:

- **Classification** tasks (like spam detection) use metrics that assess how well the model assigns data to categories, focusing on how often it’s correct versus how often it makes errors.
- **Regression** tasks (like predicting house prices) rely on metrics that quantify the difference between predicted and actual values, giving us a sense of the model’s accuracy in numerical predictions.

Classification Metrics

It's important to first understand the Confusion Matrix, as it forms the foundation for these metrics.

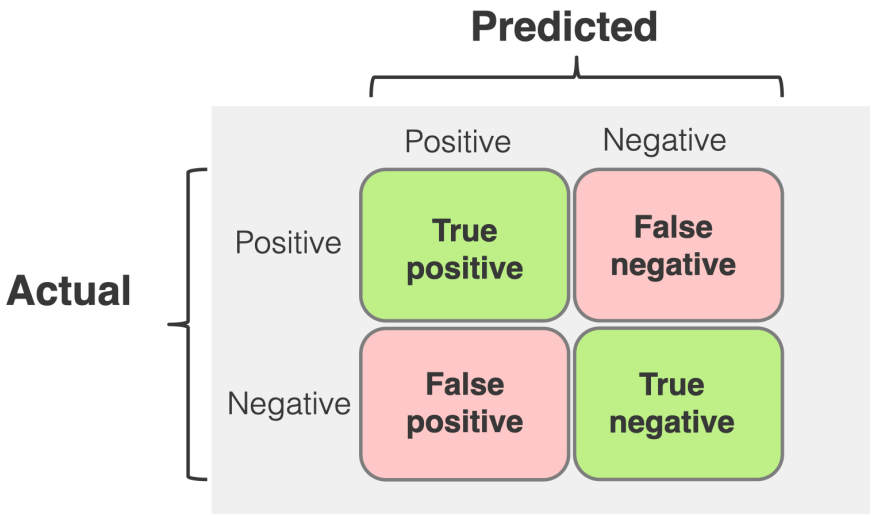
Confusion Matrix

The Confusion Matrix is a table that helps us visualize the performance of a classification algorithm. It shows the number of correct and incorrect predictions broken down by

class.

Components of a Confusion Matrix:

- **True Positives (TP)**: Correctly predicted positive instances.
- **True Negatives (TN)**: Correctly predicted negative instances.
- **False Positives (FP)**: Incorrectly predicted as positive (Type I error).
- **False Negatives (FN)**: Incorrectly predicted as negative (Type II error).

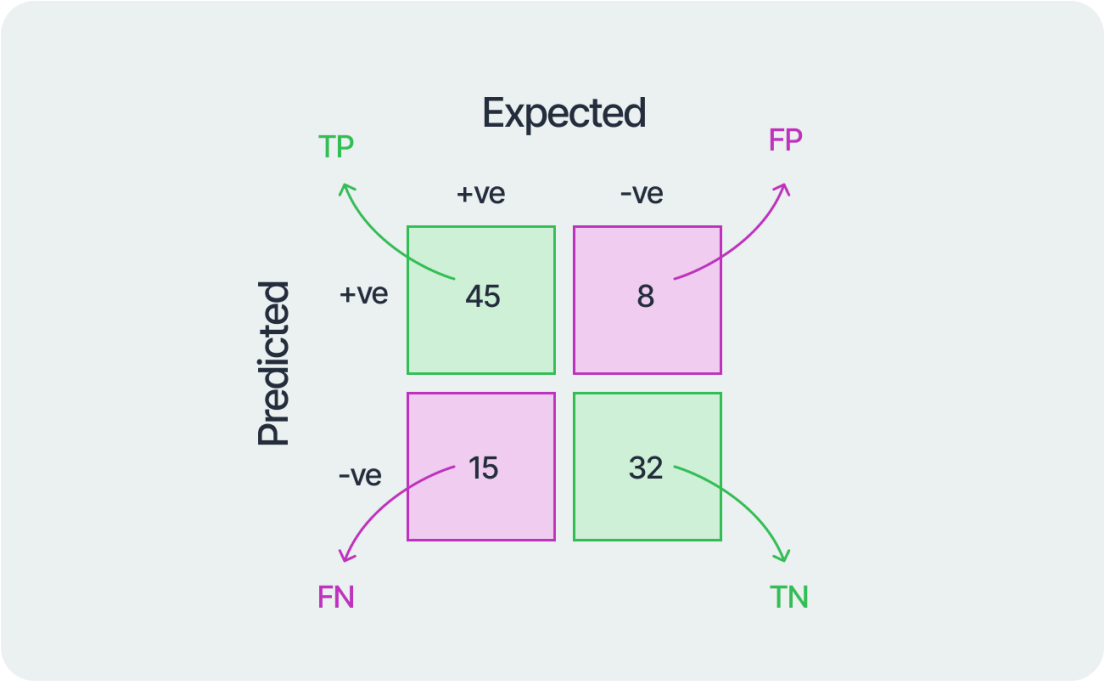


Example of this binary classification:

	Predicted Dog	Predicted Not Dog
Actual Dog	True Positive (TP)	False Negative (FN)
Actual Not Dog	False Positive (FP)	True Negative (TN)

Prediction -> Positive and Negative

Actual -> Is prediction correct or not -> True and False



Example

Confusion Matrix

			Ground Truth Label	
			has disease	*no disease*
	Total Observations (n)		Condition Positive (CP)	Condition Negative (CN)
Predicted Label	*test positive*	Test Outcome Positive (TOP)	True Positive (TP)	False Positive (FP)
	test negative	Test Outcome Negative (TON)	False Negative (FN)	True Negative (TN)

Figure 1: Basic colour coded confusion matrix with marginal sums

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

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

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

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

- https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall

1. Accuracy

- Accuracy is the ratio of correct predictions (both true positives and true negatives) to the total number of predictions.
- **Analogy**: Imagine a medical test designed to detect a disease. If out of 100 patients, 90 are correctly classified (either as having or not having the disease), the accuracy is 90%.
- **Key Point**: Accuracy is useful for an overall sense of correctness, but it can be misleading if there is an imbalance in the data (e.g., very few people actually have the disease).

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Let's use a running example: (Spam classification problem gives this result)

	Predicted Positive (Spam)	Predicted Negative (Not Spam)
Actual Positive (Spam)	True Positive (TP) = 50	False Negative (FN) = 10
Actual Negative (Not Spam)	False Positive (FP) = 5	True Negative (TN) = 100

Accuracy = (TP + TN) / (TP + TN + FP + FN)

measures the overall percentage of correct predictions

= (50 + 100)/(50+100+5+10)

= 150/165

= 0.909

or Accuracy = 90.9%

Conclusion:

- Accuracy (90.9%) is good, but we must be cautious if the dataset is imbalanced (e.g., if there are more non-spam emails than spam emails, accuracy alone may not reflect performance)

2. Precision

- Precision measures how many of the positive predictions made by the model are actually correct.
- **Analogy**: Think of precision like the reliability of a positive test result. If a medical test has high precision, when it says a person has a disease, it's likely correct. High precision is important in situations where false positives could lead to unnecessary stress or treatment.
- **Example**: If out of 10 people flagged by the test as "positive," only 7 actually have the disease, then precision is 70%.
- **Key Point**:
  - Precision focuses on minimizing **false positives** and is especially useful when false positives are costly or harmful.
  - Precision focuses on **how accurate the positive predictions** are
  - Precision: How many of the patients diagnosed with the disease actually have it (correct diagnoses)?

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

	Predicted Positive (Spam)	Predicted Negative (Not Spam)
Actual Positive (Spam)	True Positive (TP) = 50	False Negative (FN) = 10
Actual Negative (Not Spam)	False Positive (FP) = 5	True Negative (TN) = 100

Precision = TP/(TP + FP)

measures how accurate the positive (spam) predictions are. It's important when the cost of false positives (misclassifying non-spam as spam) is high

= 50/(50+5) = 0.909

Precision = 90.9%

Conclusion:

- Precision (90.9%) tells us that when the model predicts an email as spam, it's correct 90.9% of the time, which is great for preventing false alarms (non-spam emails being flagged as spam)

3. Recall (Sensitivity)

- Recall is the ratio of correctly identified positives to the total actual positives in the dataset.
- **Analogy**:

Recall is like the test's ability to catch everyone with the disease. A high-recall test is thorough in finding all cases, so it minimizes **false negatives** (cases where people have the disease but the test misses it).

- **Example**: If there are 20 patients with the disease, but the test correctly identifies only 15 of them, recall is 75%.
- **Key Point**:
  - Recall is crucial in scenarios where **missing a positive case** (false negative) is highly risky, such as in disease screening.
  - Recall focuses on **how well the model identifies** all the **positive instances**
  - Recall: How many of the patients who actually have the disease were correctly identified by the test (true positives)?

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

	Predicted Positive (Spam)	**Predicted Negative (Not Spam)**
Actual Positive (Spam)	True Positive (TP) = 50	False Negative (FN) = 10
Actual Negative (Not Spam)	False Positive (FP) = 5	True Negative (TN) = 100

Recall = TP/(TP+FN)

measures the ability of the model to correctly identify all positive (spam) cases. It's critical when the cost of false negatives (misclassifying spam as not spam) is high.

Recall = 50/(50+10) = 50/60 =0.833

Recall = 83.3%

Conclusion:

- Recall (83.3%) shows that the model correctly identifies 83.3% of all the spam emails, but it misses 16.7% of them.

However, these two metrics may not always give a full picture.

For example:

- **Precision** could be **high** if the model **rarely predicts positives**, but it may miss many actual positives (**low Recall**).
- **Recall** could be **high** if the model **predicts most positives**, but it could also include a lot of false positives (**low Precision)**.

trade-off between precision and recall is why the F1-Score is useful — it combines the two and provides a more balanced measure of performance.

---

4. F1-Score

- The F1-Score is the harmonic mean of Precision and Recall. It takes both into account in a way that balances both metrics into a single score, especially useful when there's an uneven class distribution.
- **Analogy**:

Imagine you're designing a test for a rare disease. You need a balance:

the test should not miss cases (high recall), but you also don't want it flagging too many healthy people as sick (high precision).

The F1-Score helps balance these competing needs to give an overall performance measure.

- **Example**: If a model has high recall but low precision, or vice versa, the F1-score provides a middle-ground metric that considers both.
- **Key Point**:
  - The F1-score is useful when you need a model that maintains both high precision and recall
  - When you want a single measure that reflects a balance between Precision (minimizing false positives) and Recall (minimizing false negatives)
  - In imbalanced datasets, where one class is much more common than the other (e.g., fraud detection), accuracy can be misleading.

    A model that always predicts the majority class can have high accuracy but poor performance on the minority class.

    The F1-Score is a better reflection of how the model handles both classes.

$$\text{F1 Score} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}$$

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

	Predicted Positive (Spam)	**Predicted Negative (Not Spam)**
Actual Positive (Spam)	True Positive (TP) = 50	False Negative (FN) = 10
Actual Negative (Not Spam)	False Positive (FP) = 5	True Negative (TN) = 100

F1 Score = 2* (Precision * Recall)/(Precision + Recall)

F1-Score is the harmonic mean of Precision and Recall, providing a balanced measure of the two. It's useful when you need to balance both false positives and false negatives.

F1 = 2 * (0.909*0.833)/(0.909+0.833) = 2* 0.758/1.742 = 0.869

F1 = 86.9%

Conclusion:

- F1-Score (86.9%) balances both precision and recall, which is important when we care about both types of errors (false positives and false negatives) equally.

Regression Metrics

1. Mean Absolute Error (MAE)

- https://pytorch.org/docs/stable/generated/torch.nn.L1Loss.html
- MAE is the average of the absolute differences between predicted and actual values. It gives us an idea of how far off, on average, the predictions are from the actual values.
- **Formula**:

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}.$$

- **Analogy**: Think of predicting the daily temperature for a week. If you predict that tomorrow's temperature will be 30°C, but the actual temperature is 28°C, the error is 2°C. MAE adds up all such errors over the week and gives the average of those errors.
- **Example**: If the predicted temperatures for a week (in °C) are [30, 32, 28, 35, 33], and the actual temperatures are [31, 30, 29, 34, 32], the MAE is the average of the absolute differences:

  MAE = (|31-30| + |30-32| + |29-28| + |34-35| + |32-33|)/5

  = (1 + 2 + 1 + 1 + 1)/5 = 1.2 °C
- **Key Point**: MAE is simple to understand and interprets the average error in the same units as the original data (e.g., °C for temperatures). It does not penalize larger errors more than smaller ones.

2. Mean Squared Error (MSE)

- **Definition**: MSE is the average of the squared differences between predicted and actual values. It penalizes larger errors more heavily than smaller ones because of the squaring operation.
- **Formula**:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

- **Analogy**: Imagine you're predicting the temperatures for each day in a week. If one prediction is off by 10°C, MSE will give a much higher penalty than if it's off by just 1°C, because squaring the difference makes larger errors much more costly.
- **Example**: Using the same temperature predictions:

  MSE = {(31-30)^2 + (30-32)^2 + (29-28)^2 + (34-35)^2 + (32-33)^2}/5

  = {1 + 4 + 1 + 1 + 1}/5 = 1.6 °C^2
- **Key Point**: MSE is more sensitive to large errors (outliers), as it squares the differences. It's useful when large errors are particularly undesirable.

3. R-Squared (R^2)

- R-Squared is a measure of how well the predicted values match the actual values, expressed as a percentage. It tells us the proportion of the variance in the dependent variable that is predictable from the independent variables. R^2 ranges from 0 to 1, where 1 indicates perfect prediction and 0 indicates no predictive power.

- **Formula**:
- **Analogy**: Imagine predicting temperatures over a week. If the predicted temperatures are almost the same as the actual ones, R^2 will be close to 1 (a perfect fit). If the predictions are way off, R^2 will be close to 0, indicating the model has little predictive power.
- **Example**: If the model predicts temperature perfectly every day, the error between predicted and actual values is 0, and (R^2 = 1). If the predictions are worse than just using the mean temperature for the week, (R^2) might be close to 0.
- **Key Point**: R^2 is useful for assessing the overall goodness of fit of a model. It tells you how well your model explains the variance in the data, but doesn't indicate how much error there is in the predictions themselves.

- **MAE**: Simple and interprets the error in the same units as the original data (e.g., °C for temperature). It's less sensitive to outliers.
- **MSE**: Gives more weight to larger errors (outliers), making it useful when you want to penalize bigger mistakes more heavily.
- **R^2**: Tells you how well the model explains the variance in the data. A higher R^2 means better explanatory power.

Each metric has its use case, and selecting the right one depends on the nature of your problem and what aspect of performance you care most about.

Hands-On: Manual Metric Calculation

Confusion Matrix Example: Medical Diagnosis for COVID-19 Disease Detection

A model is used to predict whether individuals are COVID-19 positive based on test results. The confusion matrix is as follows:

	Predicted Positive	**Predicted Negative**
Actual Positive	TP = 1200	FN = 300
Actual Negative	FP = 250	TN = 6250

Answers (for instructor reference): Reveal the cell!!!

<!---

1. **Accuracy**: 0.88 or 88%
2. **Precision**: 0.83 or 83%
3. **Recall**: 0.80 or 80%
4. **F1-Score**: 0.81 or 81%

---!>

Which metric do you choose now, to evaluate this model?

- accuracy alone might not provide a full picture in a COVID-19 testing scenario.
- Precision is important in situations where false positives carry a high cost. For instance, if a false positive leads to unnecessary quarantine, this might be acceptable in comparison to missing actual cases.
- But in a public health crisis, the focus on precision alone may overlook positive cases, especially when the disease is highly transmissible.
- In COVID-19 testing, recall is often prioritized because missing a positive case (false negative) can lead to further spread of the virus.
- Since recall measures how well the model catches actual positive cases, a higher recall reduces the likelihood of missing infected individuals.
- F1-score helps maintain a balance, ensuring that the model is reasonably accurate in predicting positives without too many false positives.

Thus, for this situation:

- Recall should be prioritized to minimize false negatives (missing actual COVID-19 cases). Higher recall means more infected individuals are identified, which is crucial for containment.

Recall is critical to reduce undetected cases and prevent virus spread.

- F1-Score is a good secondary metric to provide a balance, ensuring the model isn't overly focused on recall alone but maintains reasonable precision. To maintain some level of precision to avoid overburdening healthcare resources with false positives.

2. Cross-Validation Techniques

In machine learning, validating models is essential to assess their performance before deployment.

This helps ensure that the model generalizes well to new, unseen data rather than just performing well on the data it was trained on.

Without validation, we risk deploying models that might perform poorly in real-world applications, leading to unreliable predictions.

Introduction to Cross-Validation

Cross-validation divides the data into multiple subsets, training and testing the model on different combinations of these subsets.

This provides a better estimate of model performance across the dataset, helping identify if the model is overfitting or if there are patterns it consistently misses.

Cross-validation is especially useful when we have limited data, as it makes the most out of all available data by rotating through different training and testing sets.

Overfitting and Underfitting

- https://towardsdatascience.com/overfitting-and-underfitting-visually-explained-like-youre-five-8a389b511751

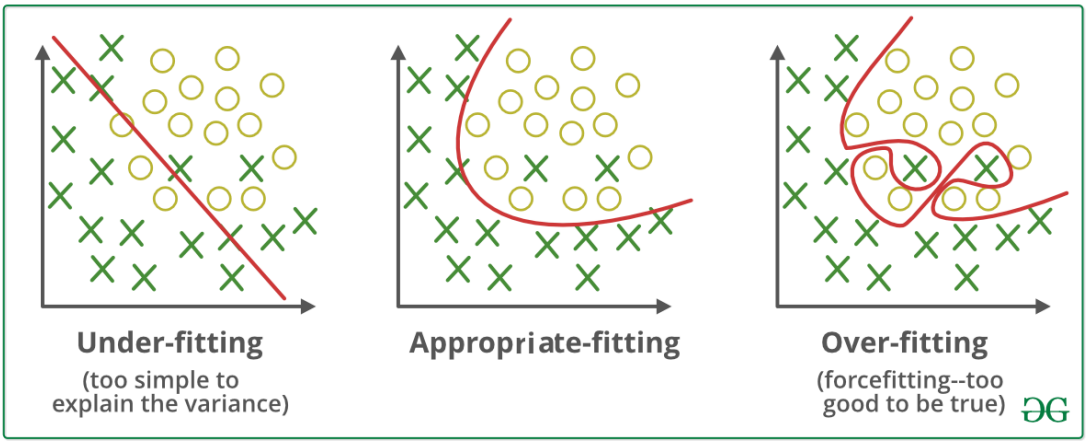
Overfitting is a concept in machine learning and statistics, where a model learns the data a bit too well.

It learns the data so much that it captures noise and random fluctuations in the data rather than just the underlying pattern.

It's like memorizing answers to specific questions without understanding the broader concept—so the model performs well on the training data but poorly on new data because it hasn't learned the underlying pattern.

Example: Imagine a student who memorizes every word in a specific book to prepare for an exam. If the exam questions come directly from the book, the student does great. But if the questions are on the general topic, the student struggles because they only memorized details rather than learning the topic.

In a machine learning context, overfitting might look like a very complex model (e.g., a decision tree that is extremely deep and captures every small detail) that fits perfectly on the training data but makes poor predictions on new data.



Underfitting occurs when a model is too simple to capture the patterns in the data.

It’s like trying to solve complex math problems with only basic arithmetic knowledge—the model can’t capture the structure of the data, resulting in poor performance on both training and new data.

Example: Imagine a student who doesn’t study much and only learns the most basic information about a topic. During the exam, they struggle to answer most questions, even the simpler ones, because they lack a sufficient understanding of the subject.

Finding the right balance—learning just enough patterns without noise—is key to building a model that performs well on new data.

Common Cross-Validation Methods

Train-Test Split

The Train-Test Split is one of the simplest methods for model validation. In this approach, the dataset is split into two parts:

- Training Set: Used to train the model on a subset of data.
- Testing Set: Used to evaluate the model’s performance on a separate subset that the model hasn’t seen during training.

Usually, the data is split in a ratio like 80-20 or 70-30 (training to testing) to ensure the model has enough data to learn from while reserving some data to test its generalization.

Limitations of Train-Test Split:

1. **Single Data Split**: The train-test split is based on just one division of the data, which may not be fully representative.

Because, in a single split, there’s no guarantee that both the training and testing sets capture all the patterns and variations present in the full dataset.

- For example, if the test set happens to contain a specific pattern or noise, the performance could be biased.

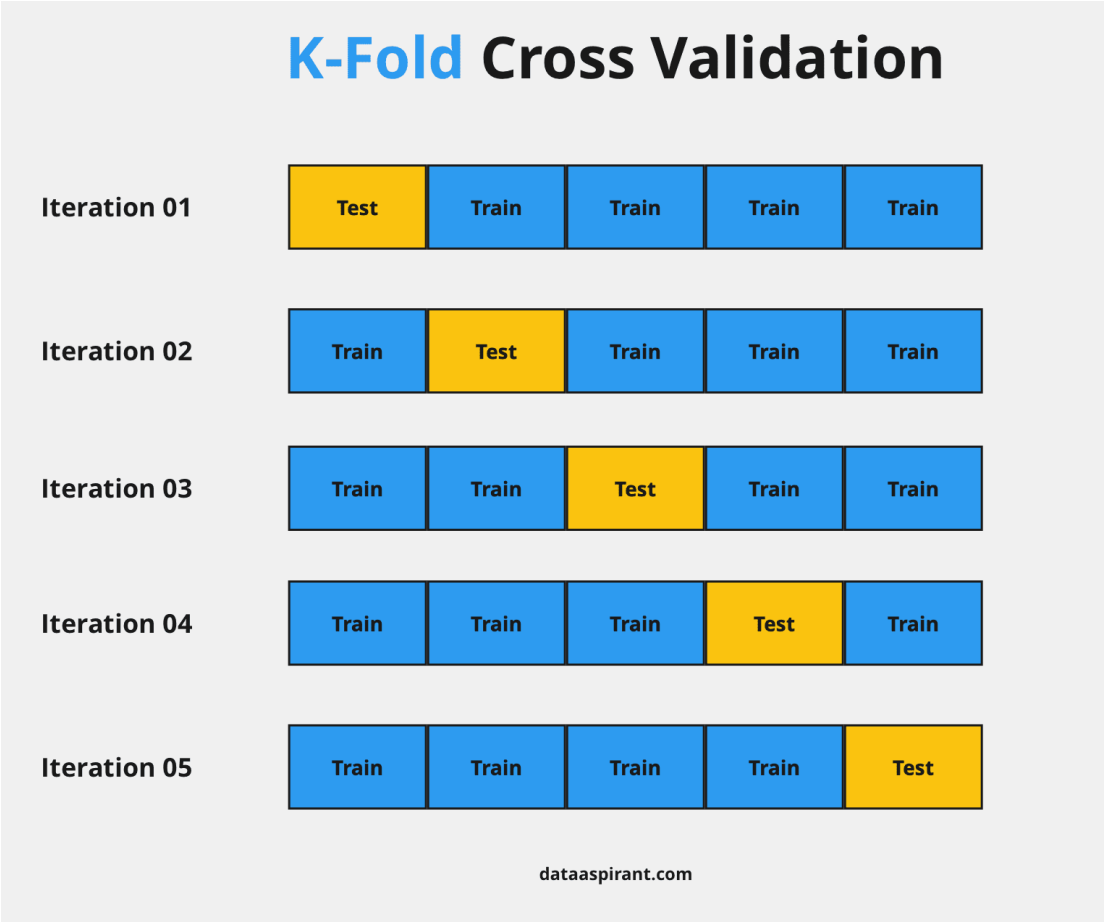
Like, let's suppose for a house price prediction model, the test set includes only houses from high-priced neighborhoods. In this case, the model might look like it’s performing well on high-value homes but could actually struggle with low- or mid-priced properties because it hasn’t encountered that pattern in the test set.

2. **Limited Information:** Since we only test the model on one subset, we may not get a comprehensive understanding of how it would perform on other unseen data.
3. **Risk of Overfitting or Underfitting:** If the training or testing set is too small, there’s a higher chance the model could overfit or underfit, leading to less reliable predictions

---

K-Fold Cross Validation

It is a method used to evaluate a model's performance by splitting the data into "K" equal parts, or folds, and testing the model multiple times to ensure a more reliable estimate of its performance.



How It Works:

1. Divide the Data:

- Split the entire dataset into K equal-sized "folds" (subsets).
- For example, in 5-Fold Cross-Validation, you divide the data into 5 parts.

2. Train and Test:

- First Round: Use 4 folds for training the model, and keep 1 fold for testing.
- Second Round: Use a different fold for testing and the remaining folds for training.
- Repeat this for each fold, so each part of the dataset gets a turn as the test set while the other folds are used for training.

3. Average the Results:

- After all the rounds, you get multiple performance scores (like accuracy, F1-Score, etc.).

- You average these scores to get a more reliable estimate of how the model will perform on new, unseen data.

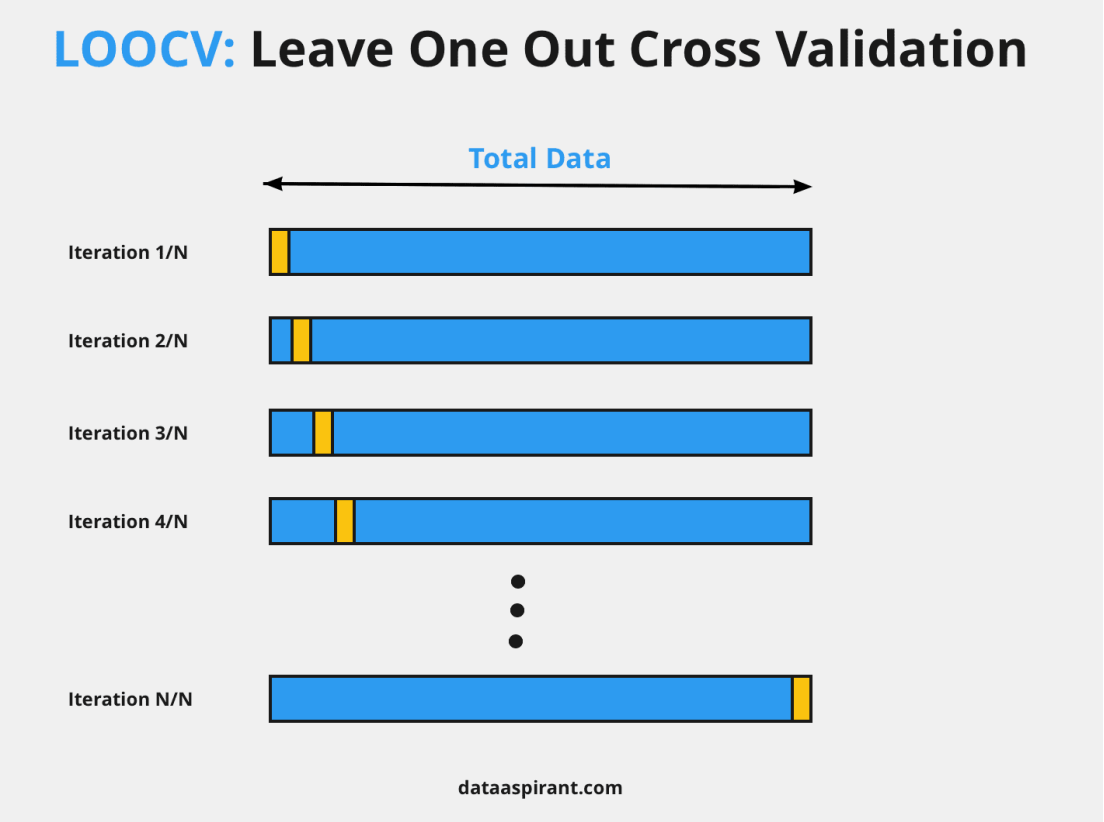
```
# Perform K-Fold Cross-Validation
for train_index, test_index in kf.split(X):
    # Split data into training and testing sets
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

Leave-One-Out Cross-Validation (LOO)

It is a special case of K-Fold Cross-Validation where the number of folds is equal to the number of data points in the dataset.

In each iteration, the model is trained on all the data points except for one, which is used as the test set.

This process is repeated for every single data point in the dataset.



How It Works:

1. Divide the dataset: For N data points, LOO uses N folds. In each iteration, 1 data point is used for testing, and the remaining N-1 data points are used for training.

2. Train and Test: In each iteration, train the model on N-1 points and test it on the single remaining point.

3. Repeat: Repeat the process for all N data points.

4. Average Results: After all iterations, average the results (e.g., accuracy) to get a final performance measure.

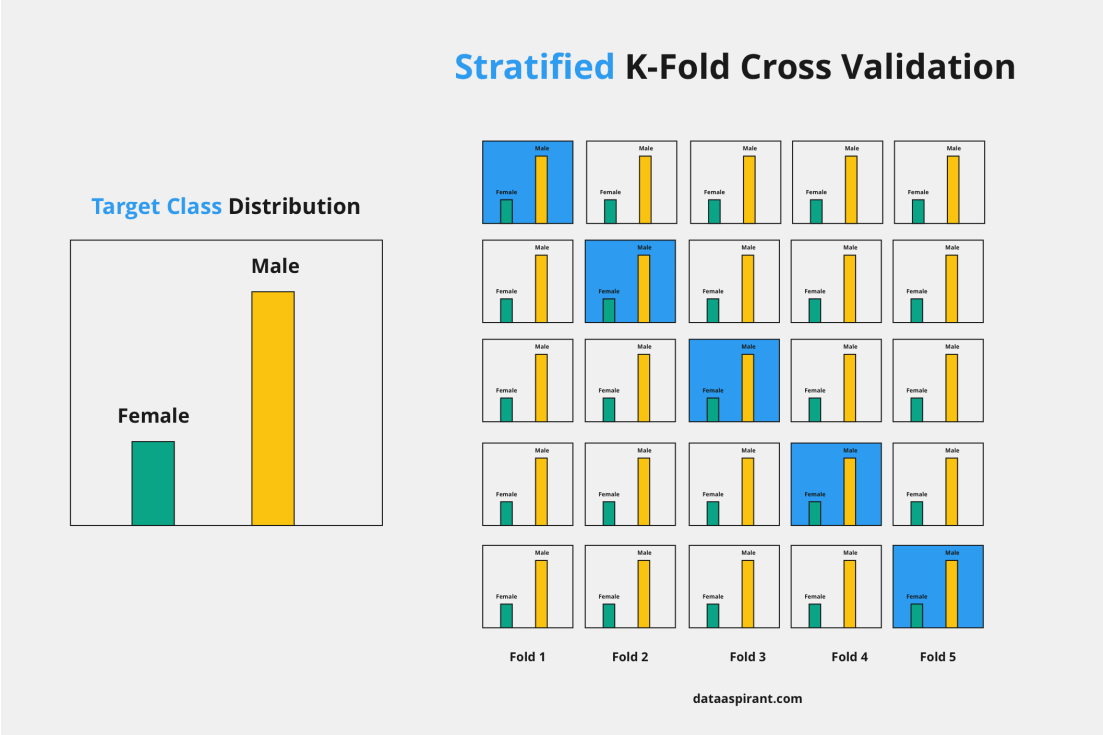
Example: For Iris dataset:

1. The dataset has 150 samples.
2. LOO will perform 150 iterations
    - 149 samples are used for training, and 1 sample is used for testing
    - In the first iteration, the model is trained on samples 2-150, and tested on sample 1.
    - In the second iteration, the model is trained on samples 1, 3-150, and tested on sample 2.
    - This continues for all 150 samples.
3. Performance metrics calculated and averaged

Stratified K-Fold Cross-Validation

It is a variation of K-Fold Cross-Validation that ensures that each fold has the same proportion of class labels as the original dataset.

It is especially useful when dealing with imbalanced datasets, where some classes have significantly more samples than others.



How It Works:

1. Divide the dataset: Similar to K-Fold, the dataset is split into K folds. However, the split is done in such a way that each fold maintains the same class distribution as the original dataset.
2. Train and Test: The process of training the model on K-1 folds and testing on the remaining fold is the same, but now each fold has a representative proportion of class labels.
3. Repeat: The process is repeated for all K folds.
4. Average Results: The results from each fold are averaged to get a final performance measure.

Example: For Iris dataset:

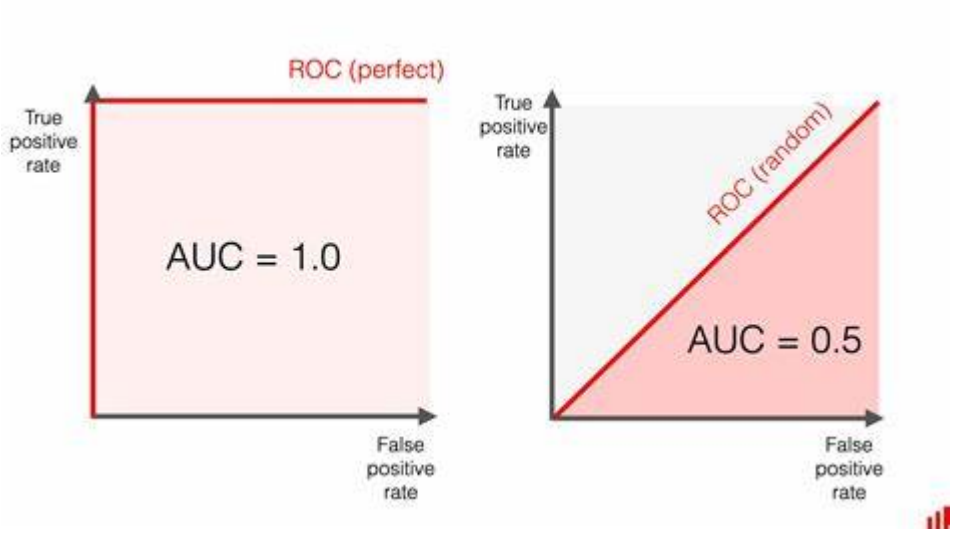
1. The dataset has 150 samples. Let's choose K = 5 as an example
2. The dataset will be split into 5 folds, with 30 samples per fold
    - the samples are split in such a way that each fold has the same proportion of classes as the original dataset.
    - Since the Iris dataset has three classes (setosa, versicolor, virginica), each fold will contain approximately 10 samples from each class.
    - First Fold: The model will train on the remaining 4 folds (120 samples), and test on the first fold (30 samples), ensuring it contains an equal number of setosa, versicolor, and virginica samples.
    - Second Fold: The model will train on the remaining 4 folds (after excluding the second fold) and test on the second fold, again ensuring class balance in both the training and testing sets.
3. Performance metrics calculated and averaged

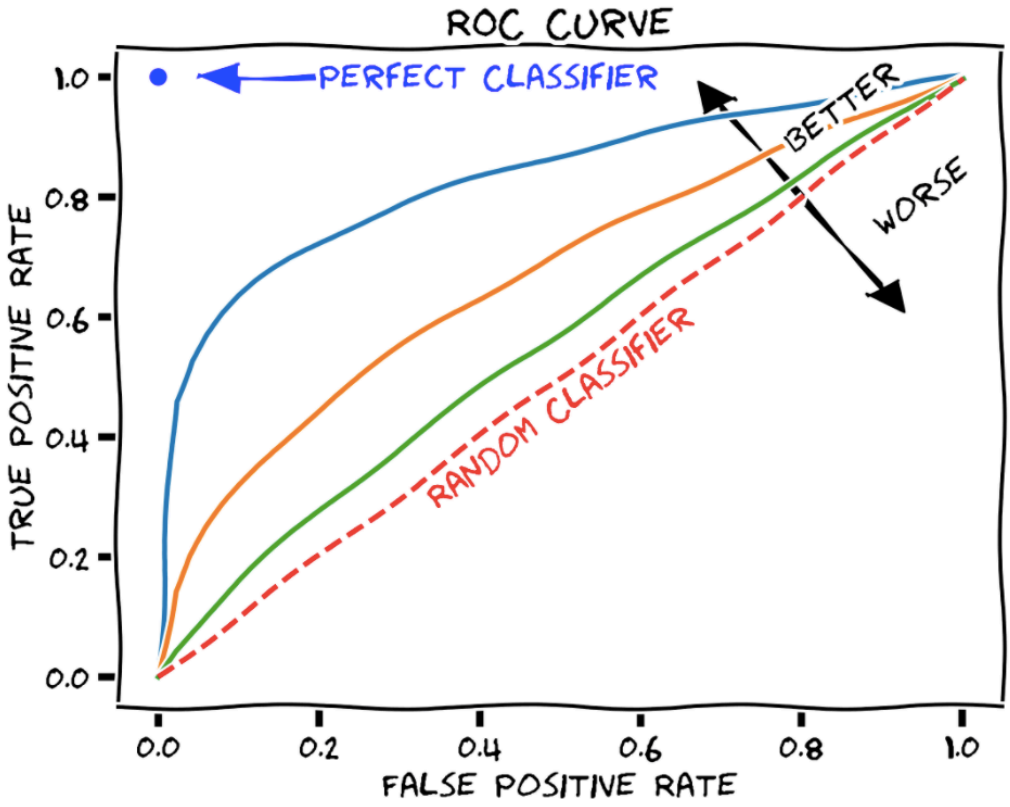
In both methods, the model gets trained multiple times, but **Stratified K-Fold is typically faster** since it doesn't need to train the model for every single data point (unlike LOO, which trains on N-1 samples upto N iterations).

Stratified K-Fold also **ensures each fold has a similar distribution** of the classes, which is especially useful if the dataset is imbalanced.

AUC (Area Under the Curve) (OPTIONAL)

AUC, or **Area Under the Curve**, is a performance metric often used in classification problems, particularly in binary classification, to evaluate how well a model separates different classes.





ROC Curve Basics: The AUC is calculated from the ROC (Receiver Operating Characteristic) curve. The ROC curve plots two key metrics:

- **True Positive Rate (TPR)**, or sensitivity, which shows how well the model identifies the positive class.
- **False Positive Rate (FPR)**, which shows how often the model incorrectly identifies the negative class as positive.

The **ROC Curve** (Receiver Operating Characteristic Curve) is a graph that helps us understand how well a classification model can distinguish between two classes (like "positive" and "negative").

Interpreting the AUC Value: The AUC represents the area under this ROC curve, providing a single number to evaluate model performance. The value of AUC ranges

from 0 to 1:

- **AUC of 1.0:** Perfect model that separates classes without any errors.
- **AUC of 0.5:** Model is no better than random guessing, meaning it has no discriminatory power.
- **AUC between 0.5 and 1.0:** Indicates the degree to which the model can separate classes; higher values mean better classification performance.

Sample code to implement AUC

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, roc_curve, auc
import matplotlib.pyplot as plt

# Load dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Train a classifier
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)

# Get probability predictions
y_probs = clf.predict_proba(X_test)[:, 1]  # Get probabilities for the
positive class

# Calculate AUC
auc_value = roc_auc_score(y_test, y_probs)
print("AUC:", auc_value)

# Plot the ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_probs)
plt.plot(fpr, tpr, label=f"AUC = {auc_value:.2f}")
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

3. Hands-On Activity: Model Evaluation and Cross Validation

Refer to the notebook: Hands-On Model Evaluation and Cross Validation.ipynb

Metric Selection Based on Problem Type (OPTIONAL PRACTICE)

In this part of the class, we focus on selecting the appropriate evaluation metric for different real-world problems. It's important to choose the right metric because different problems have different priorities, and using the wrong metric can lead to misleading conclusions.

To make this interactive, let's ask students to choose the most appropriate metric for the given scenarios. After each question, encourage students to justify their answer.

Scenario-Based Questions

1. Scenario: Detecting Credit Card Fraud

- **Question:** "For detecting credit card fraud, would accuracy be enough?"
- **Answer: No.** In credit card fraud detection, the dataset is often imbalanced (very few fraudulent transactions compared to legitimate ones). In this case, accuracy is misleading because even if the model predicts "no fraud" for most transactions, it can still achieve high accuracy. Instead, **Precision**, **Recall**, or **F1-Score** are more informative metrics. Precision measures how many of the predicted fraudulent transactions were actually frauds, while Recall measures how many fraudulent transactions were detected. F1-Score balances Precision and Recall.

---

2. Scenario: Predicting Disease in a Population

- **Question:** "If you are building a model to predict a disease in a population, would accuracy be a good metric?"
- **Answer: No.** Similar to the credit card fraud example, disease prediction often deals with imbalanced classes. If the disease is rare, a model that predicts "no disease" for everyone can still have high accuracy. **Recall** would be a more important metric in this case, as you want to minimize the false negatives (missed diagnoses). In some cases, **F1-Score** might be appropriate if you want a balance between Recall and Precision.

---

3. Scenario: Customer Churn Prediction

- **Question:** "For predicting customer churn (whether a customer will leave a service), would **Accuracy** be a reliable metric?"
- **Answer: No.** In churn prediction, the class distribution can be skewed (more customers staying than leaving). Using **Accuracy** might give a false sense of good performance. **Precision** and **Recall** would be more helpful to understand how well the model predicts churners and non-churners, and **F1-Score** could provide a balanced evaluation.

---

4. Scenario: Email Spam Detection

- **Question:** "For detecting spam emails, which metric would you prioritize?"
- **Answer: Precision** and **Recall** would both be important. You don't want too many legitimate emails (false positives) marked as spam, so **Precision** is crucial. However, you also want to make sure that as many spam emails as possible are detected, so **Recall** is also important. **F1-Score** can help balance both Precision and Recall.

---

5. Scenario: Movie Recommendation System

- **Question:** "For a movie recommendation system, would **Accuracy** be the most useful metric?"
- **Answer: No.** In a recommendation system, **Accuracy** doesn't capture the essence of a good recommendation. You'd want to focus on metrics like **Precision**, which measures how many relevant items (movies) are recommended in the top K predictions. **Recall@K** can also be used to measure how many relevant movies are found among the top recommendations.

---

6. Scenario: Predicting Student Performance

- **Question:** "For predicting whether a student will pass or fail a course, would **Accuracy** be a good metric to use?"
- **Answer: No.** In cases like this, there could be an imbalance in the number of students passing versus failing. **Accuracy** could be misleading because the model might predict "pass" for most students and still achieve high accuracy. Instead, **Precision** and **Recall** would be better metrics. **Recall** ensures you capture most of the failing students (to help intervene in time), and **Precision** would ensure you're correctly identifying those who are at risk of failing. **F1-Score** could provide a balanced view of both metrics.

---

7. Scenario: Predicting Loan Default

- **Question:** "For predicting loan defaults (whether a borrower will default on a loan), would **Accuracy** alone give a reliable assessment of the model?"
- **Answer: No.** In a loan default prediction problem, defaults may be much less frequent than non-defaults, creating a class imbalance. A model predicting "no default" for everyone could still show high **Accuracy**, but it wouldn't be useful. Instead, **Recall** would be more important, as we don't want to miss any potential defaults, and **Precision** would be crucial to avoid falsely classifying customers as likely to default. **F1-Score** balances Precision and Recall.

---

8. Scenario: Image Classification for Medical Imaging

- **Question:** "For classifying whether an X-ray image shows signs of a disease (like pneumonia), should **Accuracy** be the primary evaluation metric?"
- **Answer: No.** Medical images often present a **high class imbalance** where healthy images outnumber diseased ones. In this case, **Recall** is the most important metric, as we want to ensure that as many cases of disease as possible are detected, even if it means a higher number of false positives. **Precision** is also important to avoid unnecessary treatments. **F1-Score** helps balance both Precision and Recall, offering a more comprehensive metric for evaluating the model's performance.

---

9. Scenario: Predicting Housing Prices

- **Question:** "For predicting housing prices based on features like location, square footage, etc., would **Accuracy** be a good fit?"
- **Answer: No. Accuracy** isn't suitable for regression tasks like predicting housing prices. Instead, we use metrics like **Mean Absolute Error (MAE)** or **Mean Squared Error (MSE)** to evaluate how well the model's predictions match the actual housing prices. These metrics measure how far off predictions are from the true values.

---

10. Scenario: Recommending Products to Users

- **Question:** "For recommending products to users in an e-commerce platform, would **Accuracy** be a useful metric?"
- **Answer: No.** In recommendation systems, **Accuracy** doesn't provide much insight because it's not about whether a user clicked on a product or not, but about how relevant those recommendations are. **Precision@K** and **Recall@K** are more appropriate because they measure how many relevant products are included in the top K recommendations. These metrics focus on providing the most valuable items to users, leading to higher engagement.

Live Exercise

Now it's your turn!

Task 1: Based on the given data, perform the following tasks:

```
- Calculate Accuracy, Precision, Recall and F1 Score
- Perform Model training and evaluation using the K-Fold Cross
Validation techniques
```

Toy Dataset: Diabetes Prediction

For predicting whether a patient has a disease (e.g., predicting diabetes based on certain health parameters)

The dataset contains the following columns:

Age: The age of the patient. BMI: Body Mass Index. Blood Pressure: Blood pressure level. Insulin: Insulin level (fasting blood sugar). Glucose: Glucose level. Pregnancies: Number of pregnancies (for women). Outcome: 1 if the patient has diabetes, 0 otherwise.

In [2]:
```
# Sample Toy Dataset for Diabetes Prediction
data = {
    'Age': [45, 50, 60, 30, 70, 35, 40, 60, 25, 55],
    'BMI': [30.1, 32.2, 35.0, 28.5, 31.8, 25.6, 29.3, 33.5, 24.5, 27.2],
    'Blood Pressure': [130, 140, 150, 120, 160, 125, 135, 145, 118, 142],
    'Insulin': [180, 220, 300, 140, 350, 200, 210, 280, 120, 250],
    'Glucose': [120, 140, 160, 100, 180, 110, 115, 150, 95, 145],
    'Pregnancies': [2, 3, 0, 1, 4, 2, 1, 3, 0, 2],
    'Outcome': [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]
}

# Convert to DataFrame
df = pd.DataFrame(data)
```


In [3]:
```
# Sample SOlution to the problem
# This is not an exhaustive solution

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
from sklearn.model_selection import KFold
import numpy as np

# Convert to DataFrame
df = pd.DataFrame(data)

# Features and target variable
X = df.drop(columns=['Outcome'])
y = df['Outcome']
```

```python
# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Model: Logistic Regression
model = LogisticRegression()

# K-Fold Cross Validation (5 folds)
kf = KFold(n_splits=5, shuffle=True, random_state=42)

accuracies = []
precisions = []
recalls = []
f1_scores = []

# K-Fold Cross-Validation
for train_index, val_index in kf.split(X_train):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index

    # Train the model
    model.fit(X_train_fold, y_train_fold)

    # Make predictions
    y_pred = model.predict(X_val_fold)

    # Evaluate the model
    accuracies.append(accuracy_score(y_val_fold, y_pred))
    precisions.append(precision_score(y_val_fold, y_pred))
    recalls.append(recall_score(y_val_fold, y_pred))
    f1_scores.append(f1_score(y_val_fold, y_pred))

# Average metrics across all folds
print(f"Average Accuracy: {np.mean(accuracies):.2f}")
print(f"Average Precision: {np.mean(precisions):.2f}")
print(f"Average Recall: {np.mean(recalls):.2f}")
print(f"Average F1-Score: {np.mean(f1_scores):.2f}")
```

```
Average Accuracy: 0.70
Average Precision: 0.60
Average Recall: 0.50
Average F1-Score: 0.53
```

```
c:\Users\devid\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:146
9: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 due to no t
rue samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\devid\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:146
9: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to n
o predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\devid\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:146
9: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 due to no t
rue samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\devid\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:175
7: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no
true nor predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, "true nor predicted", "F-score is", len(true_sum))
```

Programming Interveiw Questions

1. topic:
    - question

In [ ]:

Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant