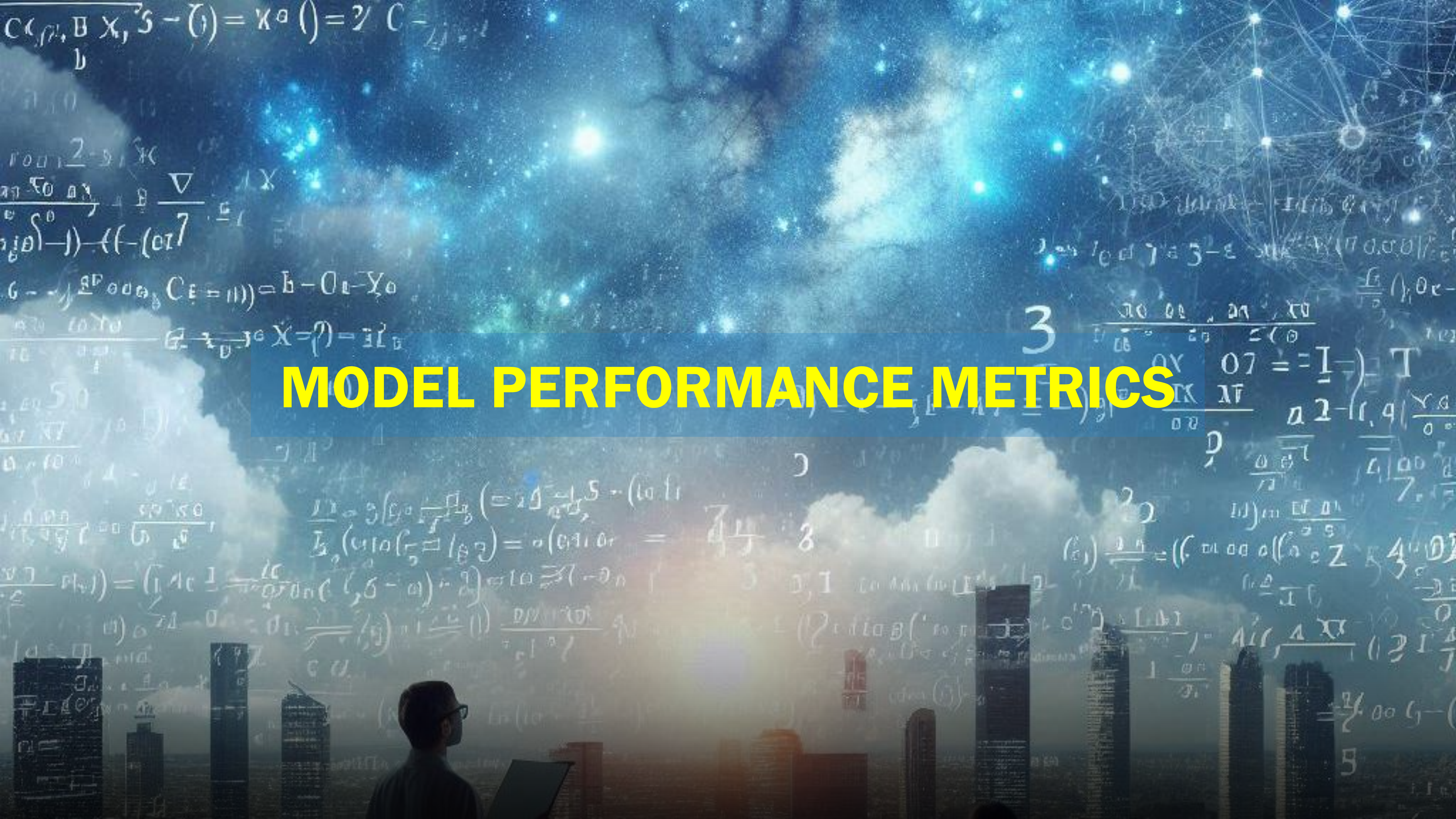
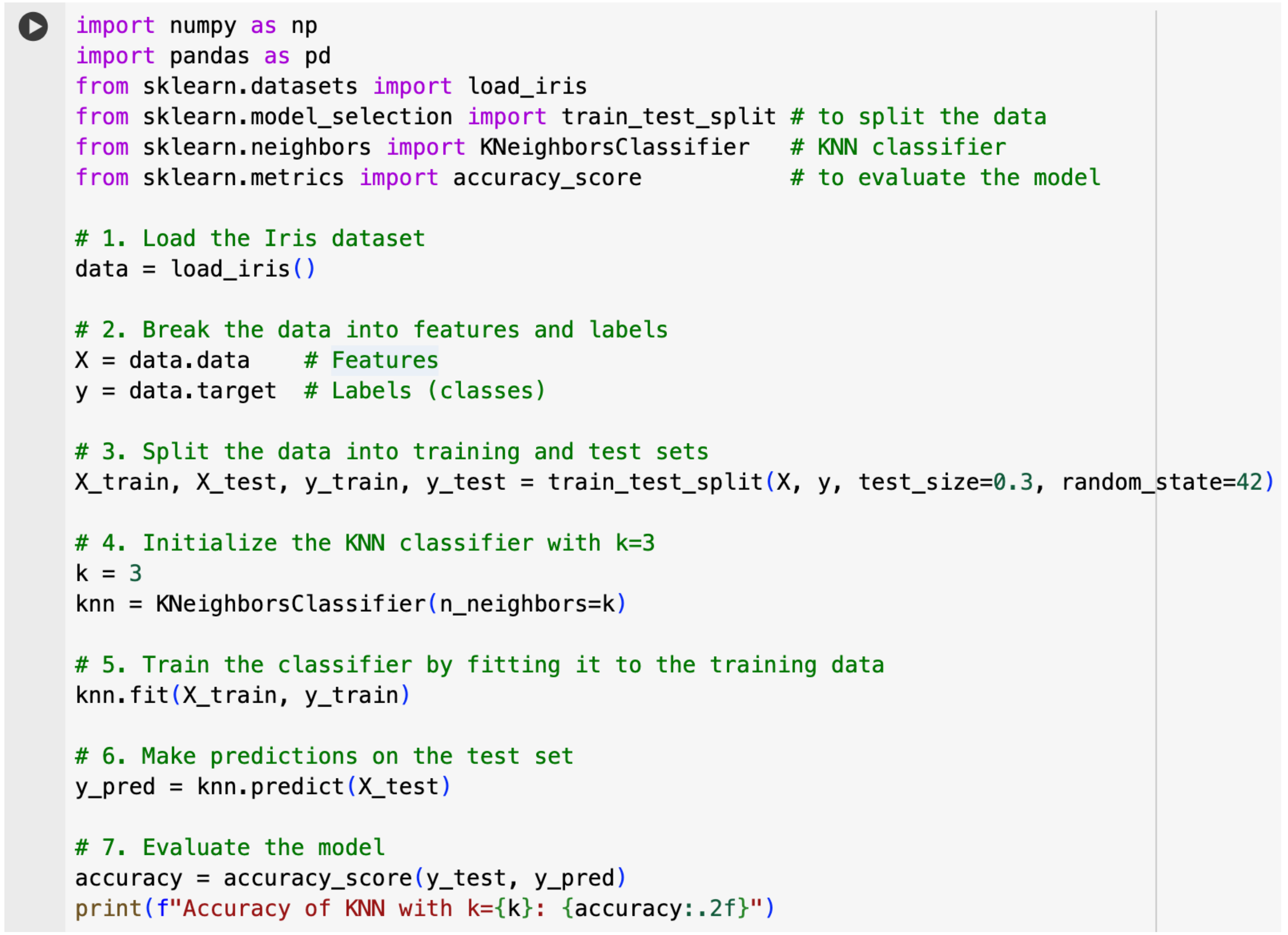


MODEL PERFORMANCE METRICS

THE SCIKIT-LEARN MODEL DEVELOPMENT PROCESS

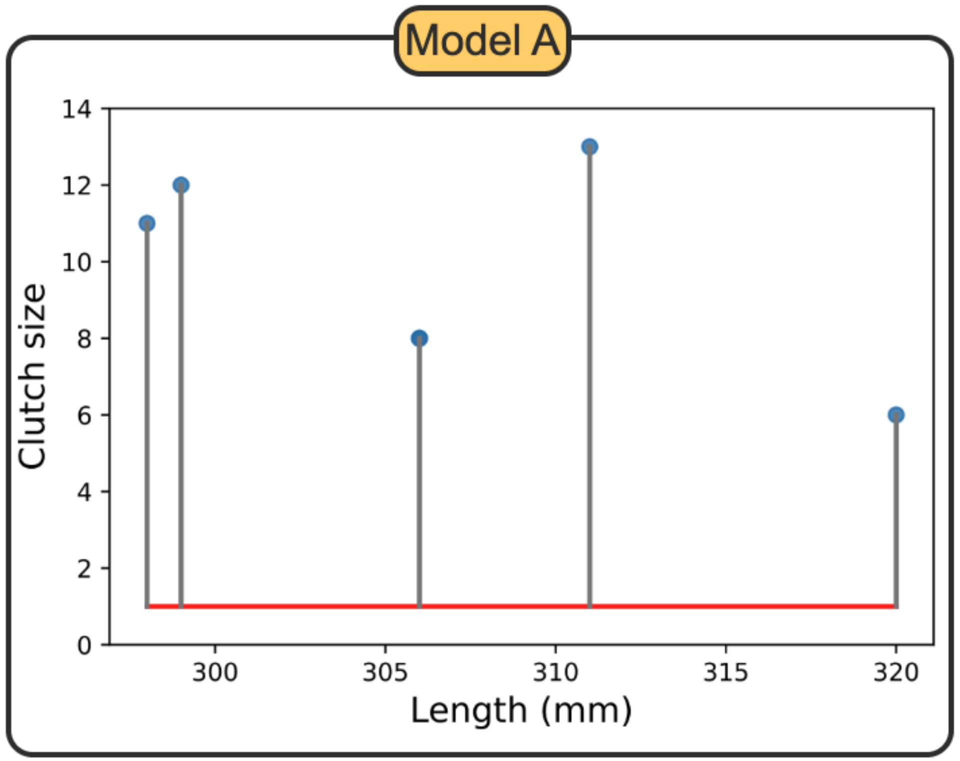
1. Load the Data set
2. Break data set into features and target
3. Create train/test split
4. Initiate the model
5. Fit the model to the training data
6. Use model to make predictions on the test data
7. **Calculate model performance**

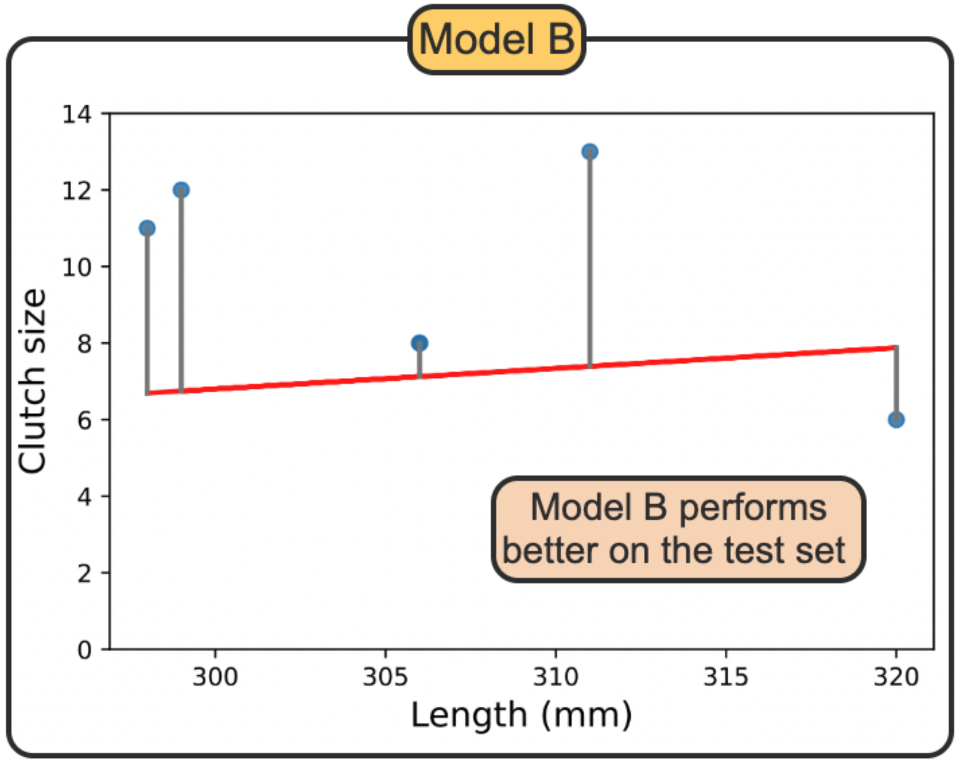


```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split # to split the data
from sklearn.neighbors import KNeighborsClassifier   # KNN classifier
from sklearn.metrics import accuracy_score           # to evaluate the model

# 1. Load the Iris dataset
data = load_iris()

# 2. Break the data into features and labels
X = data.data    # Features
y = data.target  # Labels (classes)

# 3. Split the data 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)

# 4. Initialize the KNN classifier with k=3
k = 3
knn = KNeighborsClassifier(n_neighbors=k)

# 5. Train the classifier by fitting it to the training data
knn.fit(X_train, y_train)

# 6. Make predictions on the test set
y_pred = knn.predict(X_test)

# 7. Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of KNN with k={k}: {accuracy:.2f}")
```

BYU

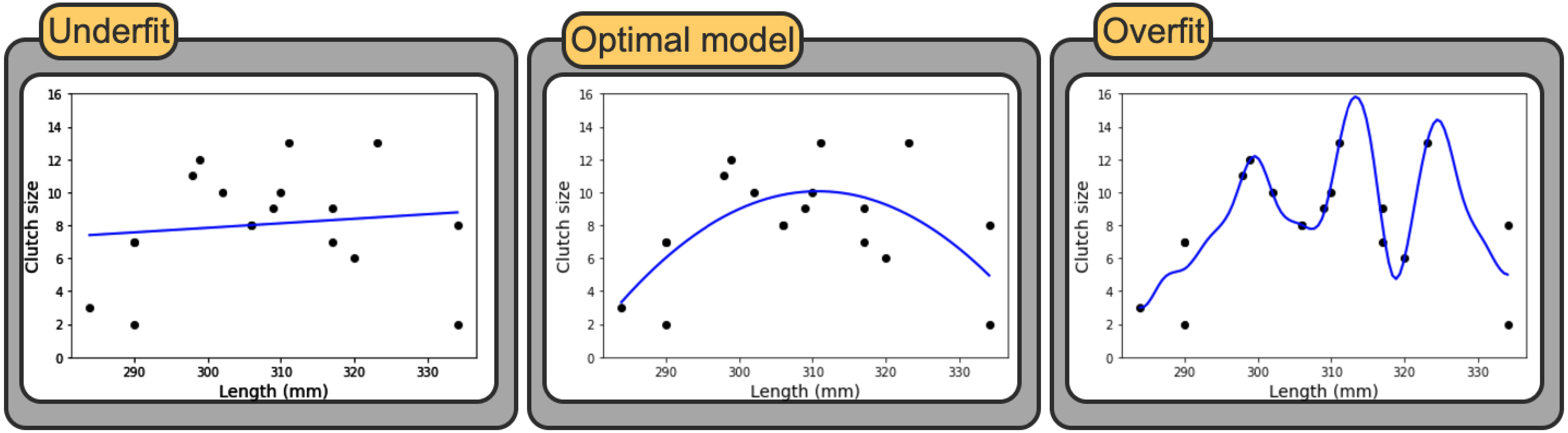
CALCULATING LOSS IN REGRESSION PROBLEMS





The vertical lines representing the losses are shorter for Model B, which means that the loss function's value is less for Model B. So Model B performs better on the test set.

UNDER- VS OVER- FITTING OF DATA TO MODELS



A quadratic regression fits the overall curved pattern of the data without passing through every point. As such, a model of moderate complexity is often optimal.

BYU

BIAS VARIANCE TRADEOFF

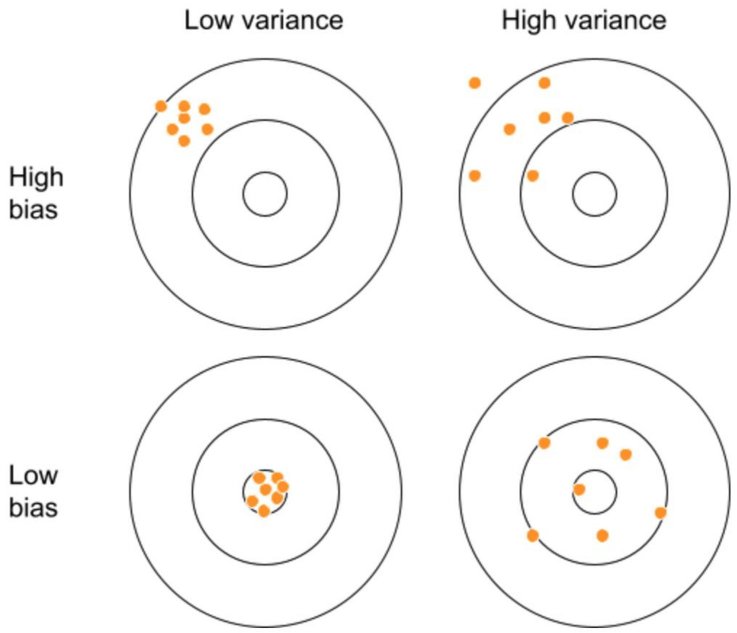
BIAS-VARIANCE TRADEOFF DEFINITION

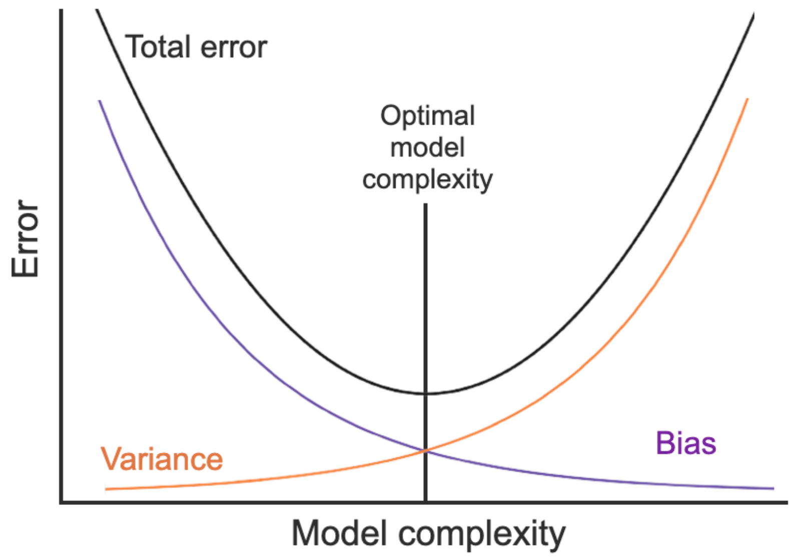
- Bias and Variance Defined
  - **Bias:** Error from oversimplifying a complex problem, leading to underfitting and failure to capture patterns.
  - **Variance:** Error from excessive sensitivity to training data, causing overfitting and capturing noise as patterns.
- Tradeoff:
  - The **bias-variance tradeoff** balances these errors: increasing model complexity reduces bias but raises variance, and vice versa. The goal is to find an optimal complexity that minimizes total error.

BIAS–VARIANCE TRADEOFF CONCEPT

$$\underbrace{E[(\hat{y}-y)^2]}_{\text{How far is a model from the ground truth}} = \underbrace{E[(\hat{y}-E[\hat{y}])^2]}_{\text{How far is a model from the «average model»}} + \underbrace{(E[\hat{y}]-y)^2}_{\text{How far is the «average model» from the ground truth}}$$

$$MSE = Variance + Bias^2$$

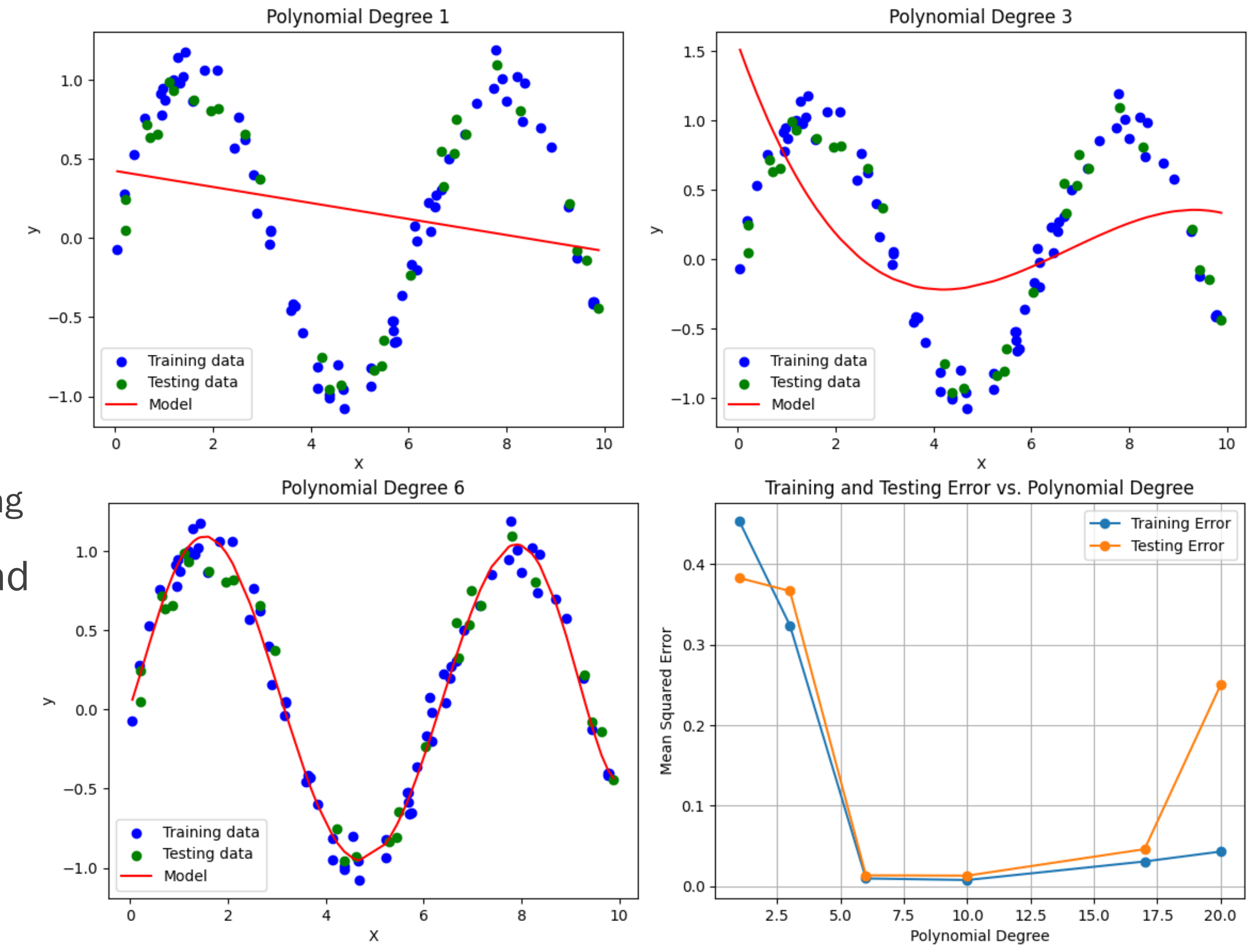




An optimal model minimizes total error, balancing bias and variance.

VISUALIZING THE TRADEOFF

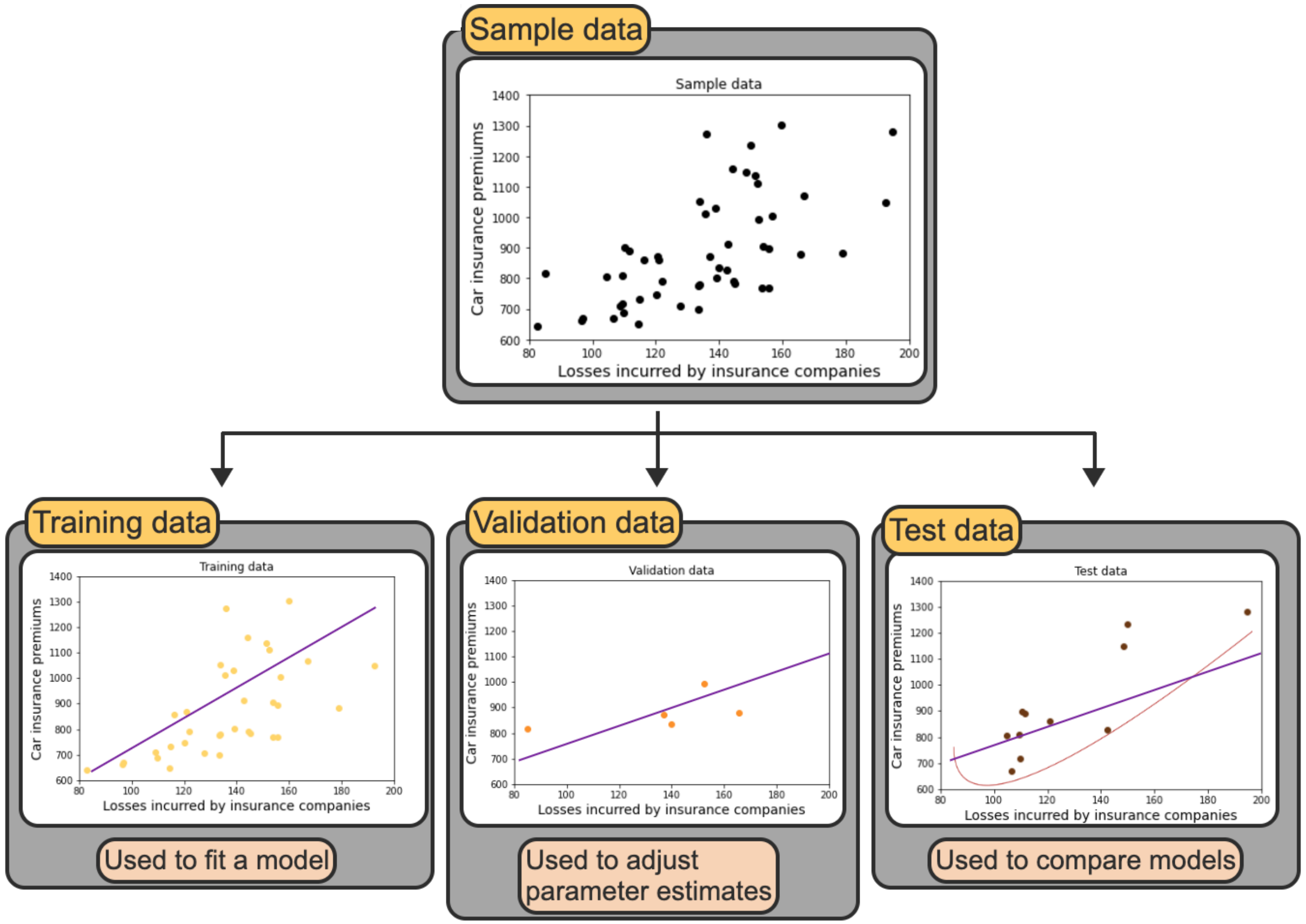
- Visualization: Plot training and validation errors vs. model complexity.
  - Training error ↓ as model fits training data.
  - Validation error forms a U-shape — decreases (↓ bias), then increases (↑ variance).
- Examples:
  - Linear regression: High bias, low variance ⟶ underfitting
  - High-degree polynomial: Low bias, high variance ⟶ overfitting
- Goal: Choose model complexity that balances bias and variance.



TRAIN TEST SPLIT

TRAINING, VALIDATION, AND TEST DATA SETS

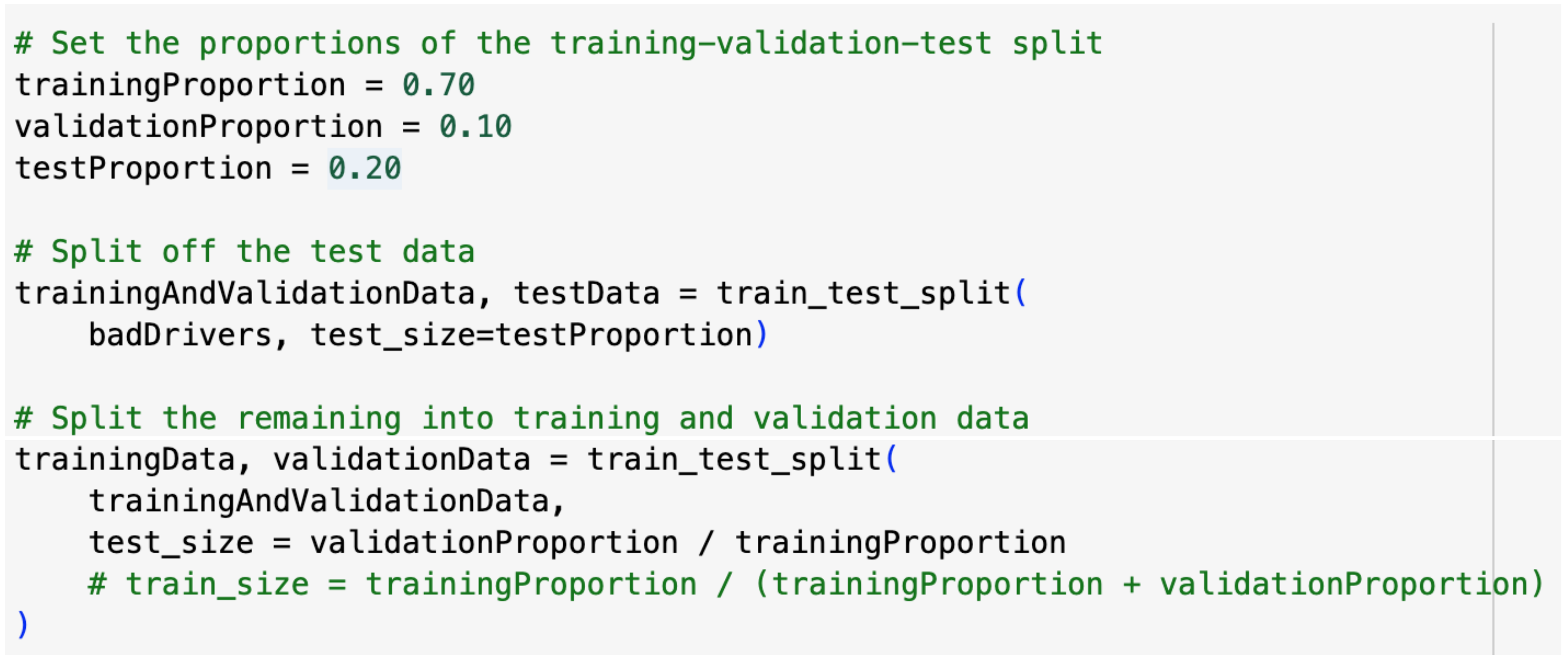
- ***Training data*** is used to fit a model.
- ***Validation data*** is used to evaluate model performance while adjusting parameter estimates and conducting feature selection.
- ***Test data*** is used to evaluate final model performance and compare different models.



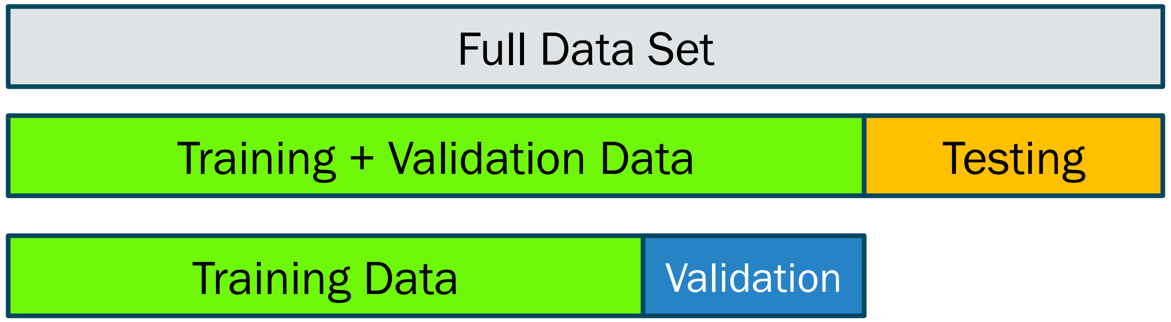
CREATING TRAINING, VALIDATION, AND TEST DATA SETS IN PYTHON

- **train_test_split** function splits the original dataset into two parts:
  - **Training / validation**
  - **test**
- **test_size** contains 20% of the data (as defined by **testProportion)**

- Next part further splits the trainingAndValidationData:
  - **trainingData:** contains data for training the model.
  - **validationData:** contains data for validating the model's performance during training.

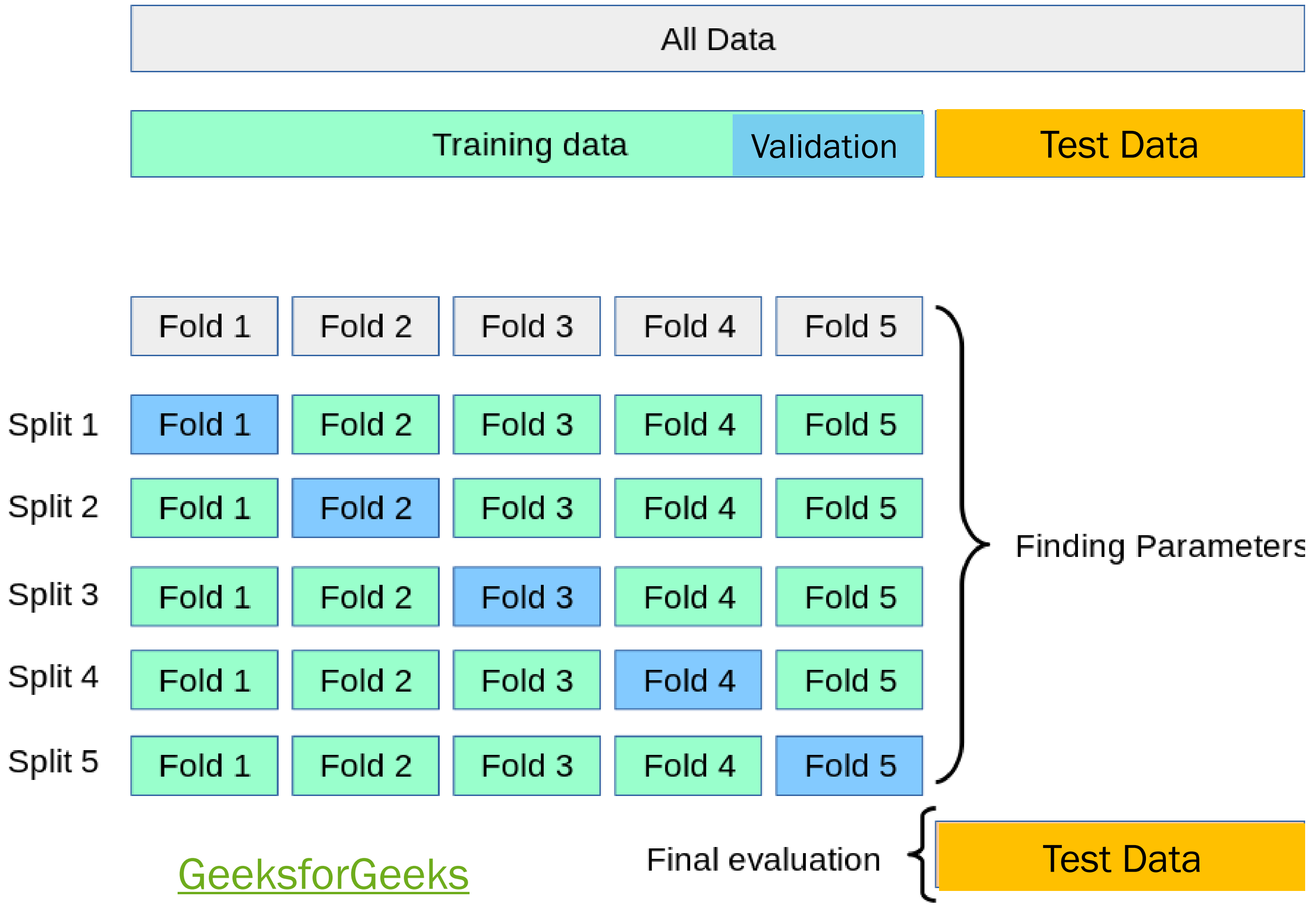


```
# Set the proportions of the training-validation-test split
trainingProportion = 0.70
validationProportion = 0.10
testProportion = 0.20

# Split off the test data
trainingAndValidationData, testData = train_test_split(
    badDrivers, test_size=testProportion)

# Split the remaining into training and validation data
trainingData, validationData = train_test_split(
    trainingAndValidationData,
    test_size = validationProportion / trainingProportion
    # train_size = trainingProportion / (trainingProportion + validationProportion)
)
```



CROSS-VALIDATION

- Split data into **k folds** (e.g., 5) – each fold takes a turn as validation.
- Repeat process **k times**, then **average results** for reliable performance.
- Ensures the model **generalizes well**, not just fits one data split.



GeeksforGeeks

REGRESSION METRICS

MEAN SQUARED ERROR (MSE), RMSE

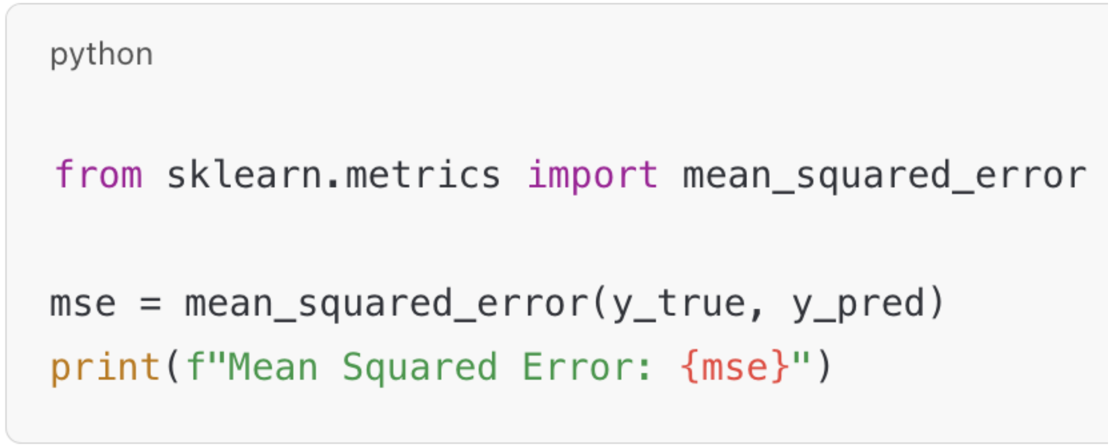
- **Definition:** MSE calculates the average of the squares of the errors, emphasizing larger errors due to the squaring process.

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- **Definition:** RMSE is the square root of MSE, providing error metrics in the same units as the target variable, making interpretation more intuitive.

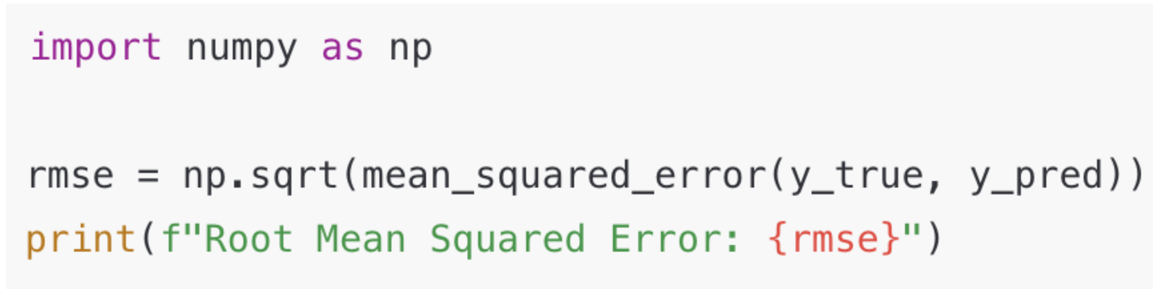
$$RMSE = \sqrt{MSE}$$

Python Implementation:



python

```python
from sklearn.metrics import mean_squared_error

mse = mean_squared_error(y_true, y_pred)
print(f"Mean Squared Error: {mse}")
```



```python
import numpy as np

rmse = np.sqrt(mean_squared_error(y_true, y_pred))
print(f"Root Mean Squared Error: {rmse}")
```

BYU

MEAN ABSOLUTE ERROR (MAE)

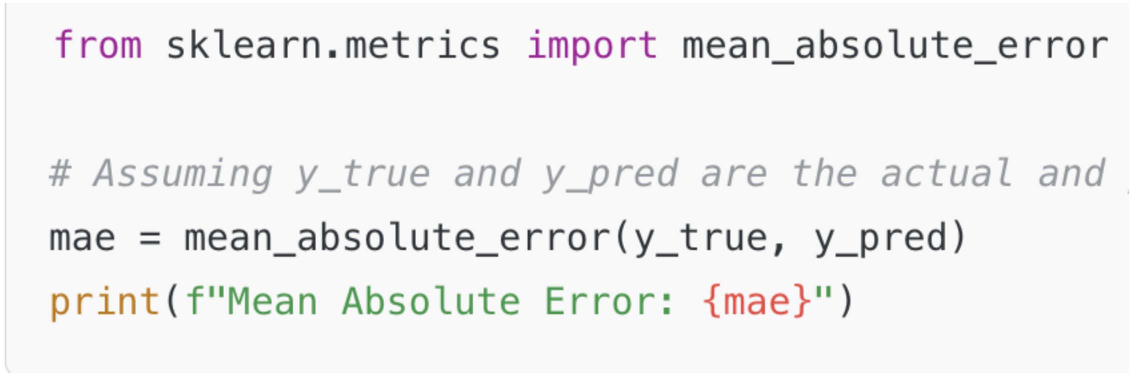
- **Definition:** MAE measures the average magnitude of errors between predicted and actual values, without considering their direction.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Where:

- n is the number of observations.
- y_i is the actual value.
- $\hat{y}_i$ is the predicted value.

Python Implementation:



```python
from sklearn.metrics import mean_absolute_error

# Assuming y_true and y_pred are the actual and
mae = mean_absolute_error(y_true, y_pred)
print(f"Mean Absolute Error: {mae}")
```

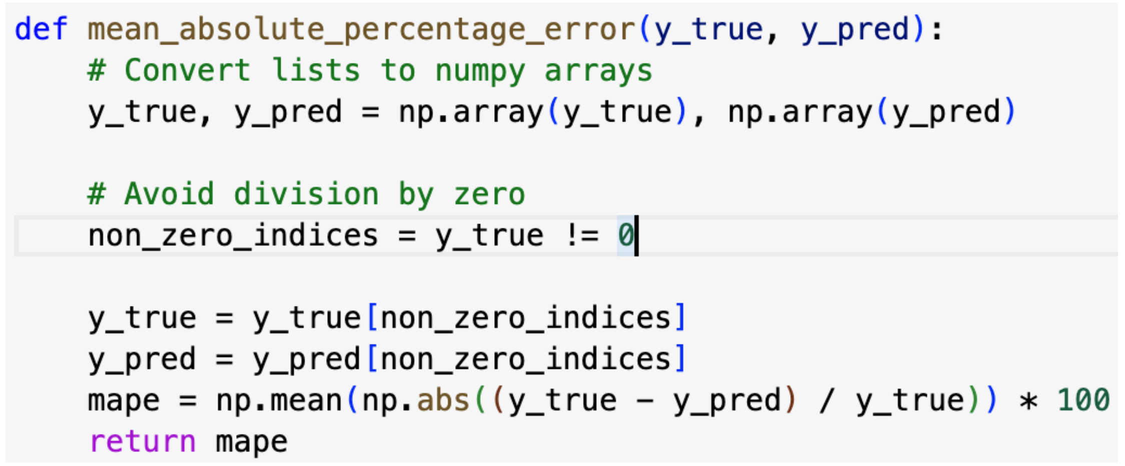
MAPE = MEAN ABSOLUTE PERCENTAGE ERROR

The Mean Absolute Percentage Error (MAPE) is a metric used to assess the accuracy of a forecasting method by expressing the average absolute error as a percentage of actual values.

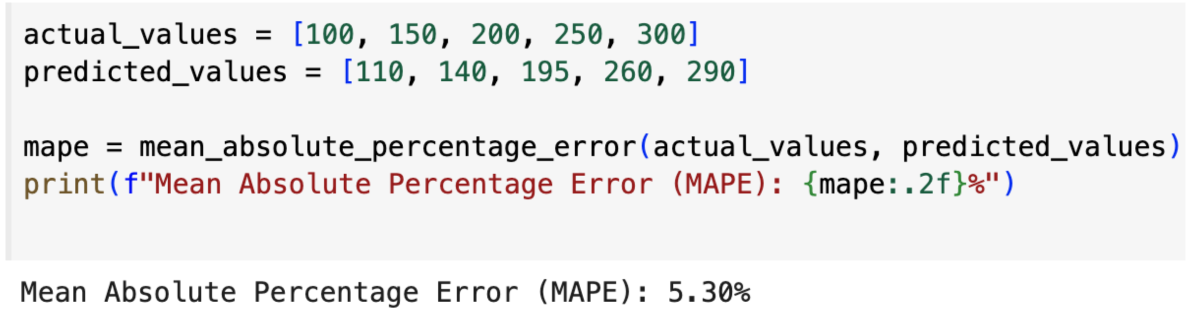
$$\text{MAPE} = 100\frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \widehat{y_i}}{y_i}\right|$$

Where:

- n is the number of observations
- y_i represents the actual value for observation i
- $\hat{y}$ denotes the predicted value for observation i



```python
def mean_absolute_percentage_error(y_true, y_pred):
    # Convert lists to numpy arrays
    y_true, y_pred = np.array(y_true), np.array(y_pred)

    # Avoid division by zero
    non_zero_indices = y_true != 0

    y_true = y_true[non_zero_indices]
    y_pred = y_pred[non_zero_indices]
    mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    return mape
```



```python
actual_values = [100, 150, 200, 250, 300]
predicted_values = [110, 140, 195, 260, 290]

mape = mean_absolute_percentage_error(actual_values, predicted_values)
print(f"Mean Absolute Percentage Error (MAPE): {mape:.2f}%")
```

```
Mean Absolute Percentage Error (MAPE): 5.30%
```

BYU

R-SQUARED, ADJUSTED R^2

- **Definition:** R-squared indicates the proportion of the variance in the dependent variable (target) that is predictable from the independent variables (features).

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Where $\bar{y}$ is the mean of the actual values.

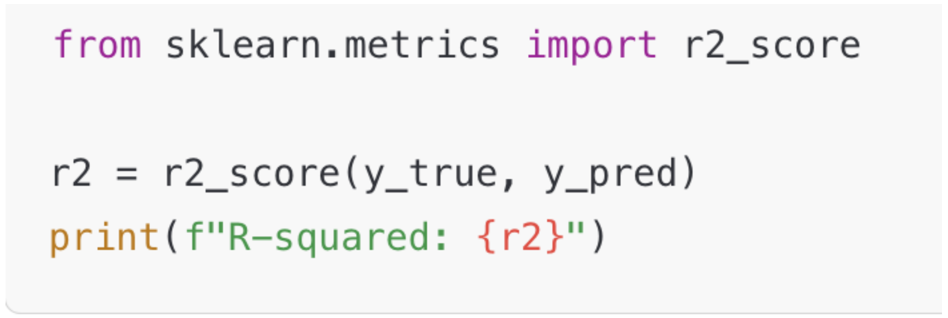
- **Definition:** Adjusted R-squared adjusts the R-squared value based on the number of features (aka predictors) in the model, providing a more accurate measure when multiple predictors are used.

$$\text{Adjusted } R^2 = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

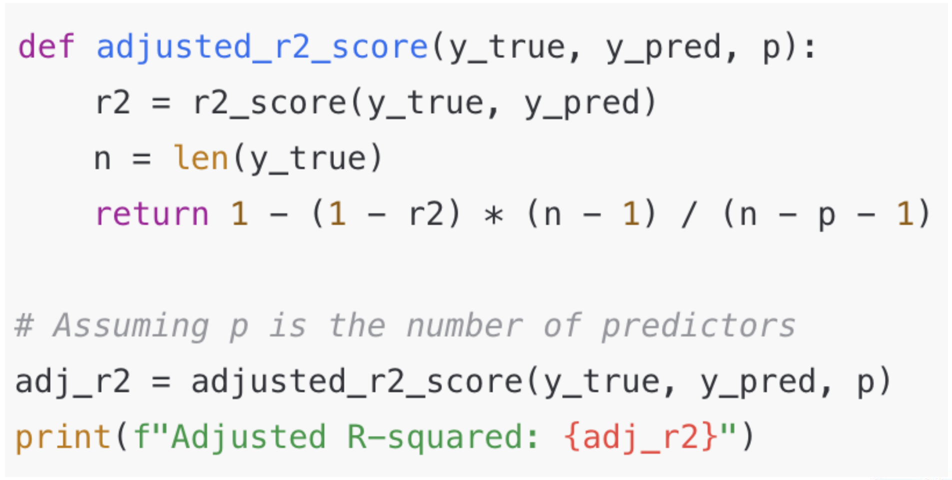
Where:

- n is the number of observations.
- p is the number of predictors.

Python Implementation:



```python
from sklearn.metrics import r2_score

r2 = r2_score(y_true, y_pred)
print(f"R-squared: {r2}")
```



```python
def adjusted_r2_score(y_true, y_pred, p):
    r2 = r2_score(y_true, y_pred)
    n = len(y_true)
    return 1 - (1 - r2) * (n - 1) / (n - p - 1)

# Assuming p is the number of predictors
adj_r2 = adjusted_r2_score(y_true, y_pred, p)
print(f"Adjusted R-squared: {adj_r2}")
```

BYU

REGRESSION METRICS - SUMMARY

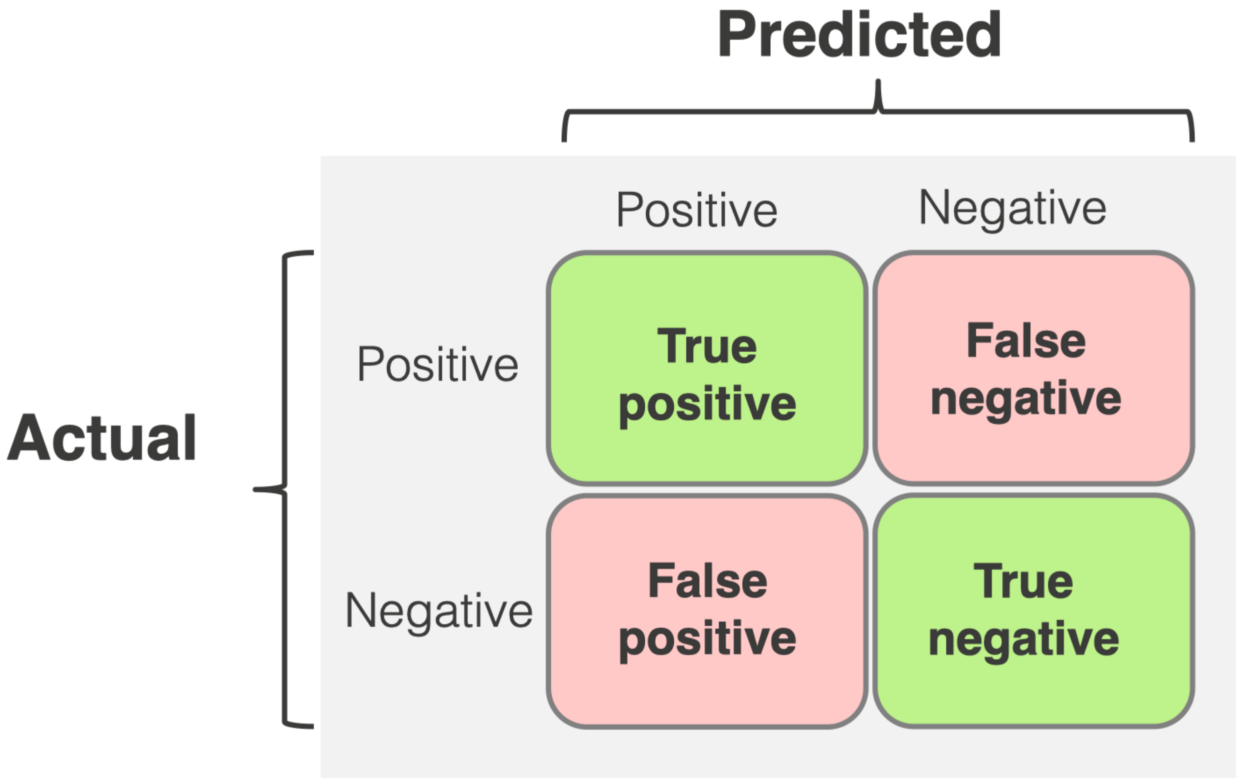
Metric	Definition	Math	When to Use?
MSE (Mean-Squared Error)	Measures average squared difference between estimated and actual values.	$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	When large errors are more critical.
RMSE (Root-Mean-Squared-Error)	Square root of MSE, in same units as response variable.	$\sqrt{MSE}$	When error scale should match target scale.
MAPE (Mean-Absolute Percentage-Error)	Percentage error between estimated and actual values.	$100\frac{1}{n}\sum_{i=1}^{n}\left\lvert\frac{y_i - \hat{y}_i}{y_i}\right\rvert$	For forecasting and percentage-based error analysis.
R-Squared	Proportion of variance explained by the model. 1 – (Residual Sum of Squares/Total Sum of Squares).	$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$	Indicates model's explanatory power.
Adjusted R^2	Statistical measure that modifies the R-Squared value to account for the number of predictors	$1 - (1 - R^2)\frac{n-1}{n-p-1}$	Useful in multiple regression with several independent variables

BYU

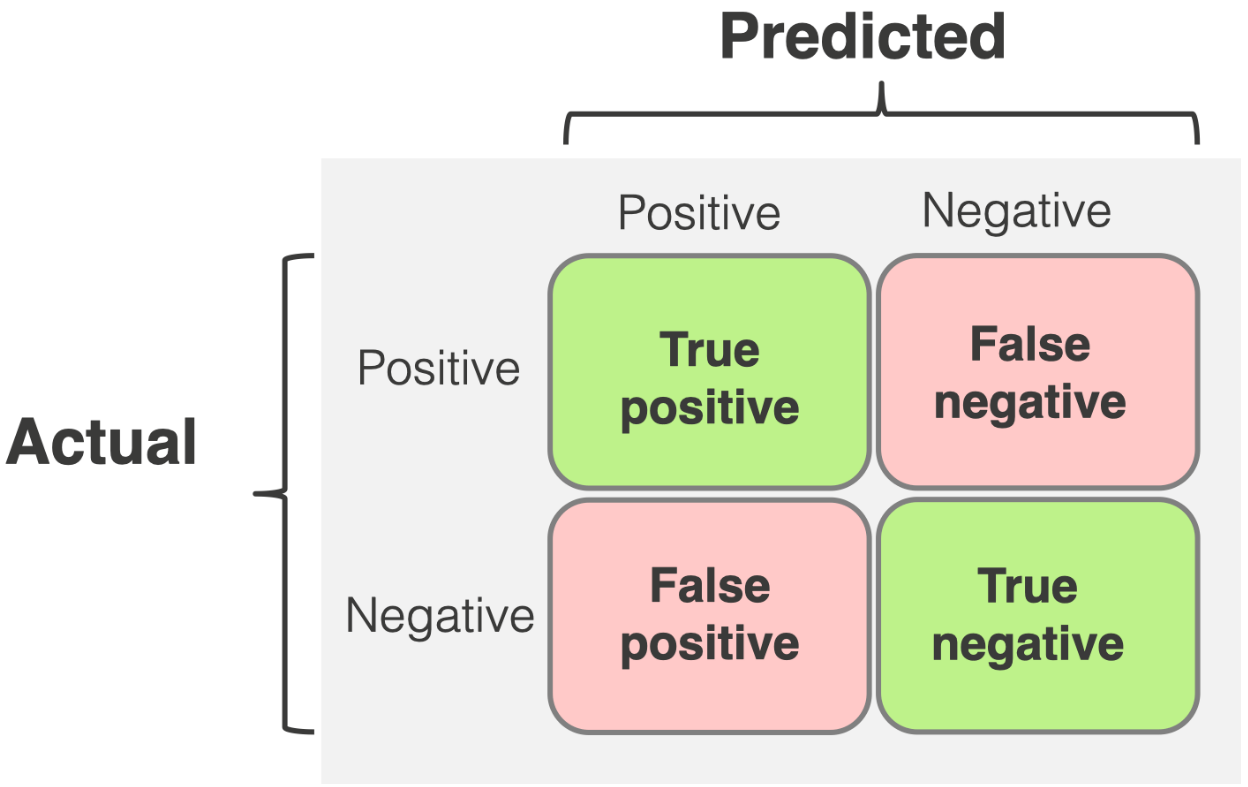
CLASSIFICATION METRICS

MEASURING MODEL PERFORMANCE WITH THE CONFUSION MATRIX

- A **confusion matrix** is a N x N matrix where N is total number of target categories (2 x 2 = Binary).
- It compares actual target values to those predicted by the **ML model**.
- Prediction results can be put into four categories:
  - **TP (True Positive):** Correctly predicted positive instances.
  - **TN (True Negative):** Correctly predicted negative instances.
  - **FP (False Positive):** Negative instances incorrectly predicted as positive.
  - **FN (False Negative):** Positive instances incorrectly predicted as negative.



ACCURACY



		Predicted	
		Positive	Negative
Actual	Positive	**True positive**	**False negative**
	Negative	**False positive**	**True negative**

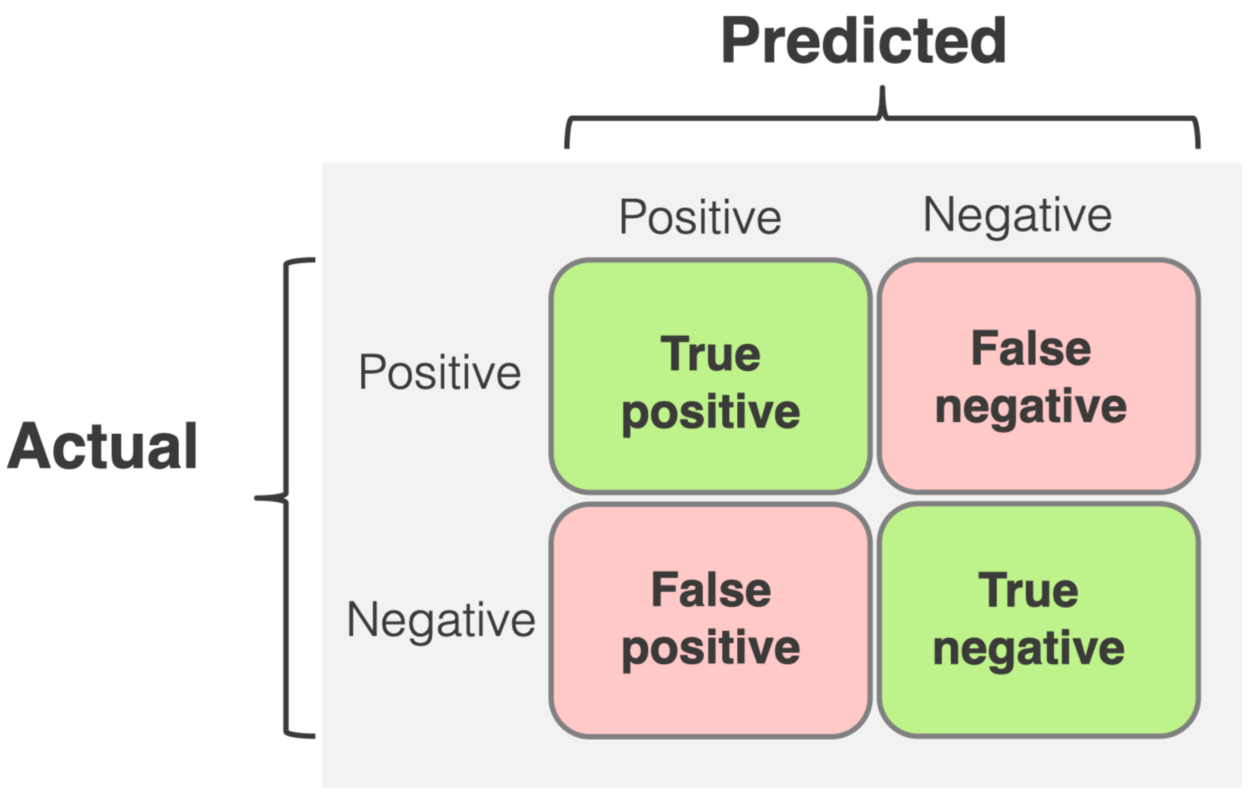
When is model accuracy NOT a good measure of performance?

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{All predictions}}$$

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

BYU

PRECISION AND RECALL



Precision is a metric that measures how often a machine learning model correctly predicts the positive class.

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

Recall is a metric that measures how often a machine learning model correctly identifies positive instances (true positives) from all the actual positive samples in the dataset. (**True Positive Rate)**

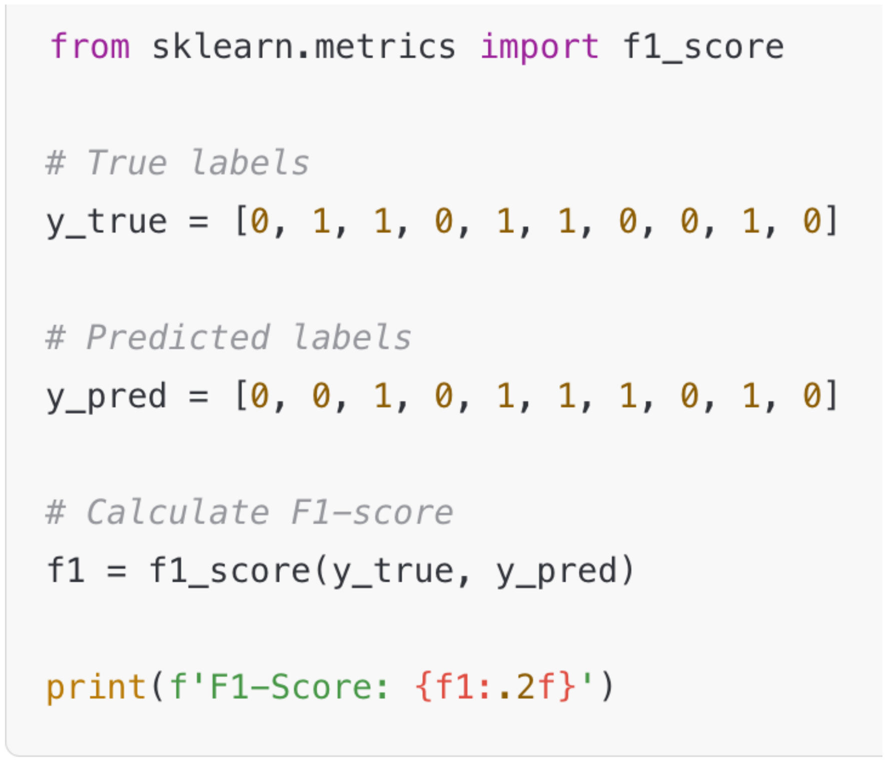
$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1-SCORE

- Combines two fundamental metrics: **Precision** and **Recall**, providing a single measure that balances both metrics.
- The F1-score is the **harmonic mean** of Precision and Recall, offering a balance between the two.
- It is particularly useful when the cost of false positives and false negatives is high, and when the class distribution is imbalanced.

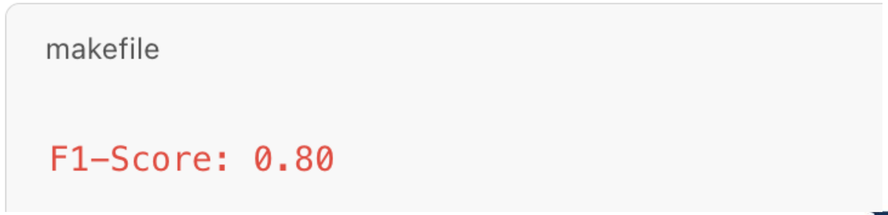
$$F1 - Score = 2 * \frac{Precision \ * Recall}{Precision + Recall}$$

- This formula ensures that the F1-score is high only when both Precision and Recall are high.



```python
from sklearn.metrics import f1_score

# True labels
y_true = [0, 1, 1, 0, 1, 1, 0, 0, 1, 0]

# Predicted labels
y_pred = [0, 0, 1, 0, 1, 1, 1, 0, 1, 0]

# Calculate F1-score
f1 = f1_score(y_true, y_pred)

print(f'F1-Score: {f1:.2f}')
```

Output:



```makefile
F1-Score: 0.80
```

MULTICLASS CONFUSION MATRIX

- Confusion matrix number of rows and columns equals the number of classes.
- Each cell (i, j) in the matrix represents the number of instances where the actual class is i and the predicted class is j.
- The diagonal elements indicate correct predictions, while off-diagonal elements represent misclassifications.

	Predicted (j)		
Actual (i)	Class 0	Class 1	Class 2
Class 0	50	2	1
Class 1	4	45	5
Class 2	3	6	47

In this matrix:

- **True Positives (TP):** Diagonal elements (e.g., 50 for Class 0).
- **False Positives (FP):** Sum of the corresponding column minus the diagonal element.
- **False Negatives (FN):** Sum of the corresponding row minus the diagonal element.

What is the Accuracy, Precision, Recall?

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

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

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

BYU

TRUE POSITIVE RATE (TPR) AND FALSE POSITIVE RATE (FPR)

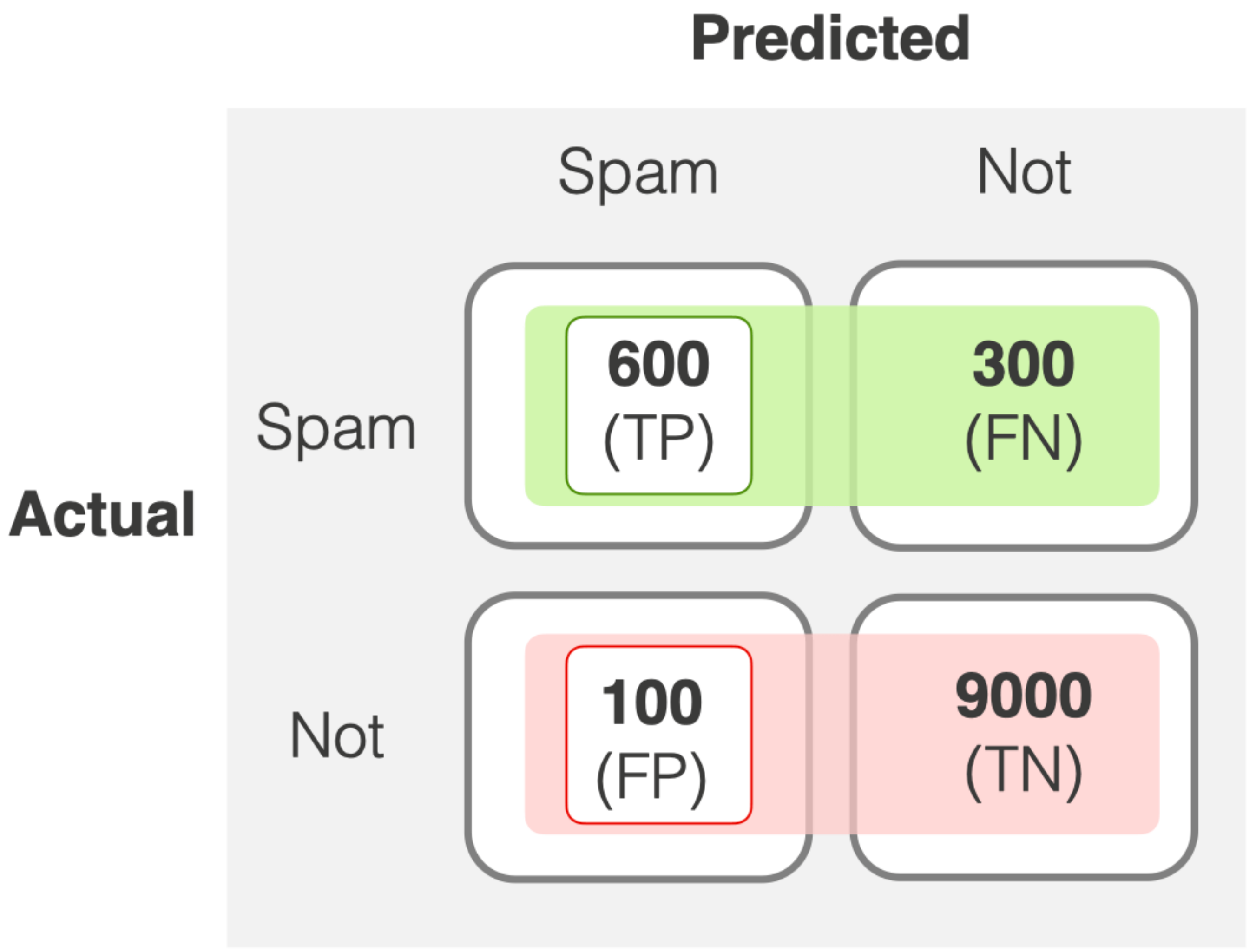
- **True Positive Rate (TPR) (aka Recall or Sensitivity)** measures the proportion of actual positives that are correctly identified by the model.
- A higher TPR indicates that the model effectively identifies positive cases, minimizing missed detections.
- **TPR** $= \frac{\text{True Positives (TP)}}{\text{True Positives (TP) + False Negatives (FN)}}$

- **False Positive Rate (FPR)**, also known as the **fall-out**, quantifies the proportion of negative instances that are incorrectly classified as positive.
- A lower FPR signifies that the model is proficient at correctly identifying negative cases, reducing the occurrence of false positives.
- **FPR** $= \frac{\text{False Positives (FP)}}{\text{False Positives (FP)+True Negatives (TN)}}$

- An ideal model aims for a high TPR (close to 1) and a low FPR (close to 0), indicating effective detection of positives with minimal false positives.

BYU

EXAMPLE TPR AND FPR

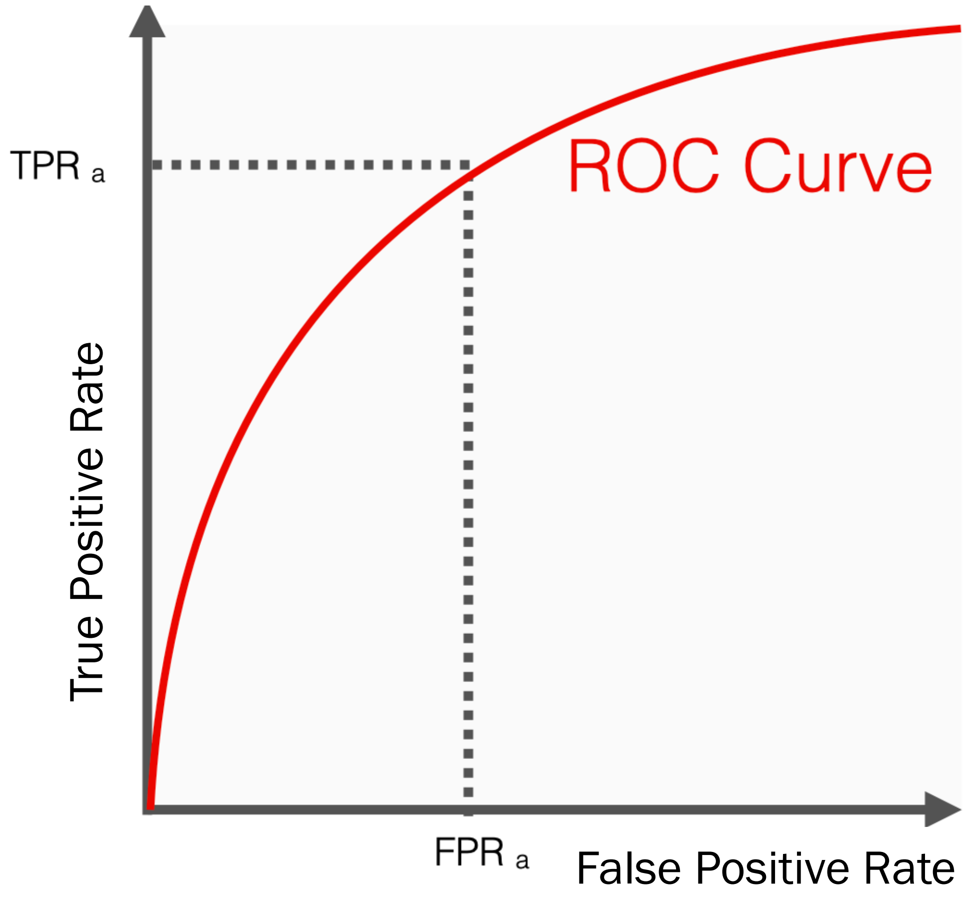

Predicted
Spam
Not
Actual
Spam
Not
600 (TP)
300 (FN)
100 (FP)
9000 (TN)

$$\mathrm{TPR} = \frac{\boxed{TP}}{TP + FN}$$

$$\mathrm{FPR} = \frac{\boxed{FP}}{FP + TN}$$

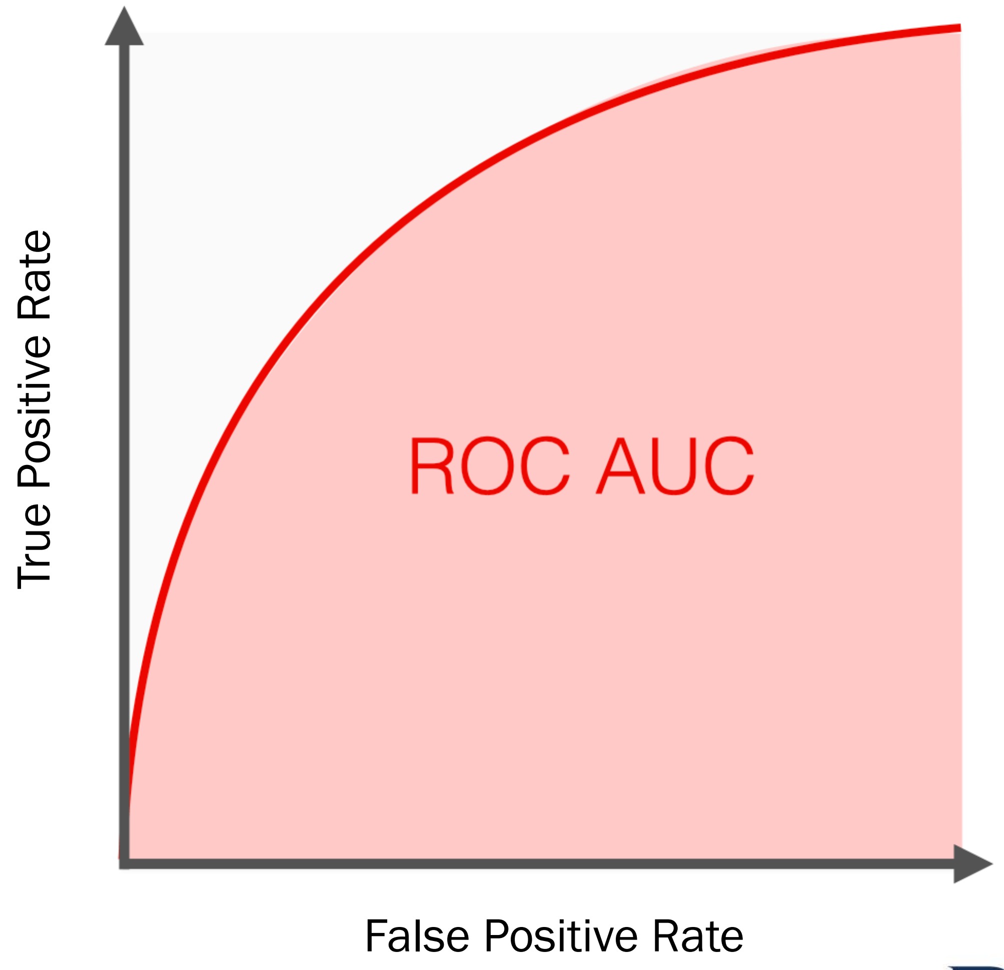
ROC CURVE

- The ROC curve stands for the **Receiver Operating Characteristic** curve.
- It is a graphical representation of the performance of a binary classifier at different classification thresholds.
- The curve plots the possible True Positive rates (TPR) against the False Positive rates (FPR).
- Each point on the curve represents a specific classification threshold with a corresponding True Positive rate and False Positive rate.



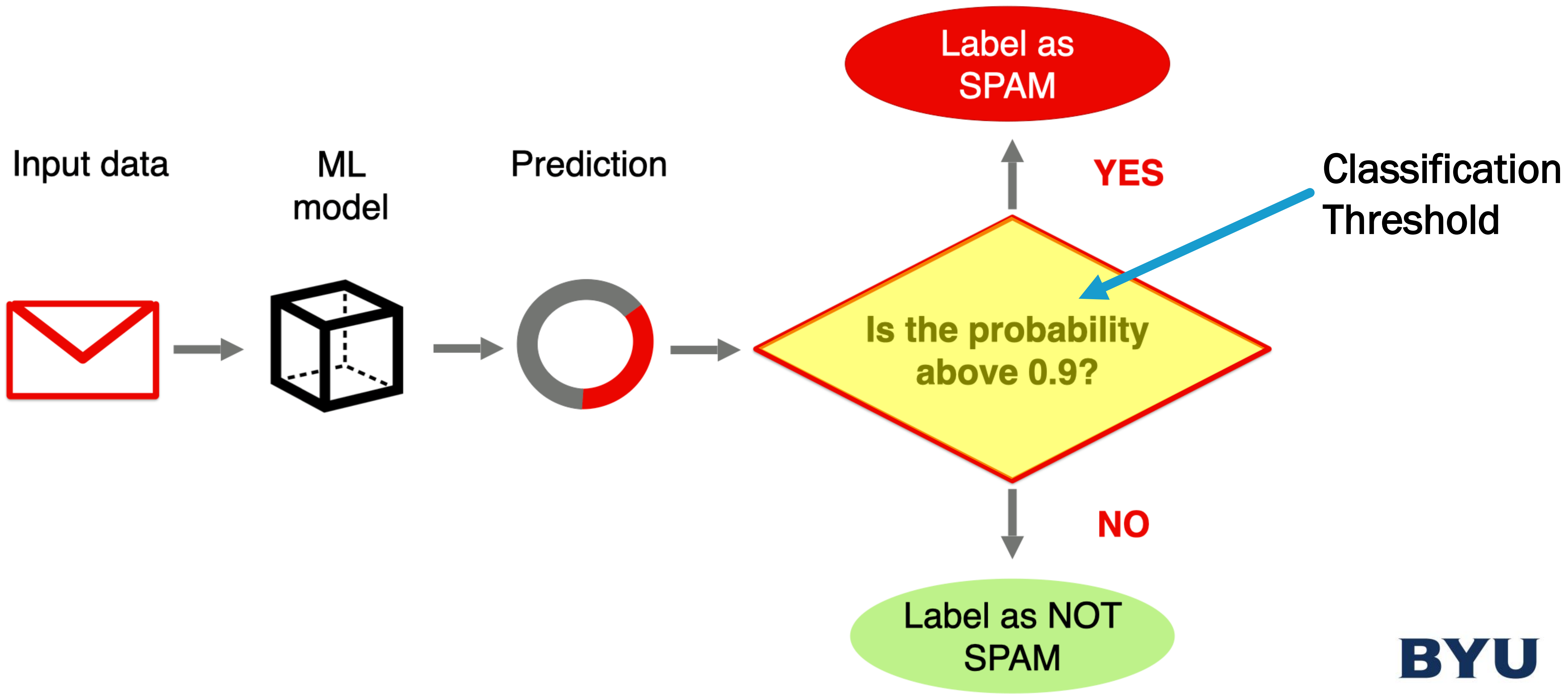
ROC AUC

- ROC AUC stands for **Receiver Operating Characteristic Area Under the Curve.**
- ROC AUC score is a single number that summarizes the classifier's performance across all possible classification thresholds.
- To get the score, you must measure the area under the ROC curve.
- ROC AUC score shows how well the classifier distinguishes positive and negative classes. It can take values from 0 to 1.
- A higher ROC AUC indicates better performance. A perfect model would have an AUC of 1, while a random model would have an AUC of 0.5.



BYU

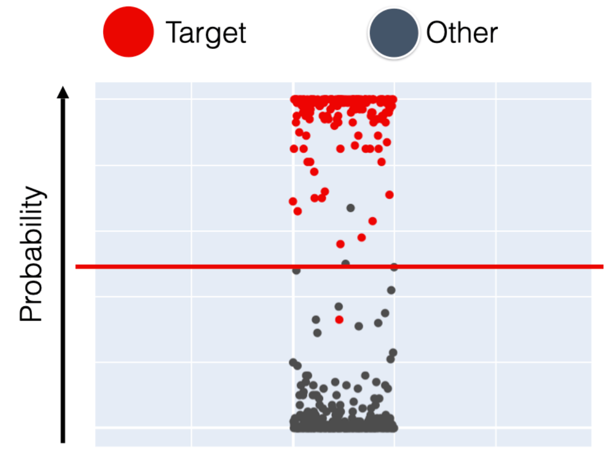
CLASSIFICATION THRESHOLD



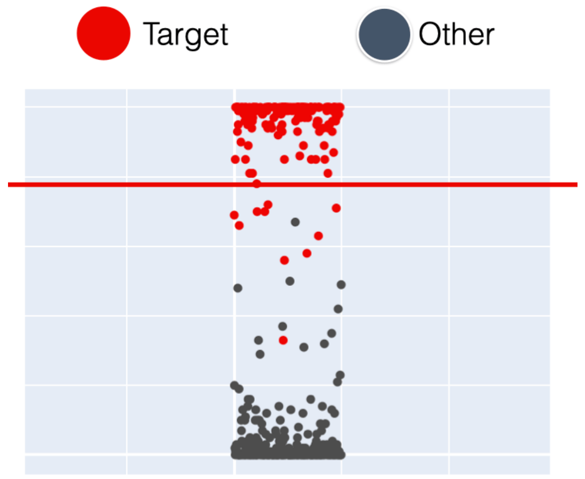
CLASSIFICATION THRESHOLD

- You need a **classification threshold** to convert the output of a probabilistic classifier into class labels.
- The threshold determines the minimum probability required for a positive prediction.
- Different thresholds can lead to different **trade-offs between** precision and recall.
- The **0.5 threshold** is often a default choice, but it might not be optimal in real-world problems.
- This depends on the specific task and the cost of false positives and false negatives errors.

Probability > 50%



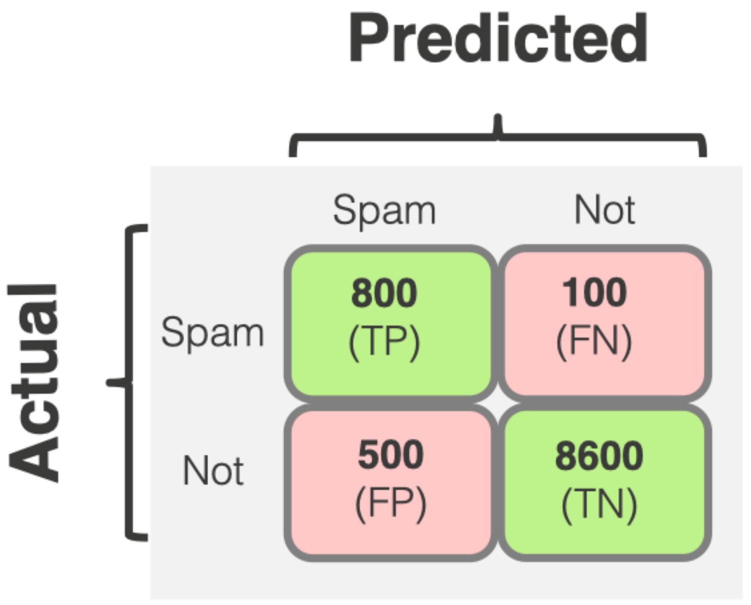
Probability > 80%



$$p = \sigma(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n)$$

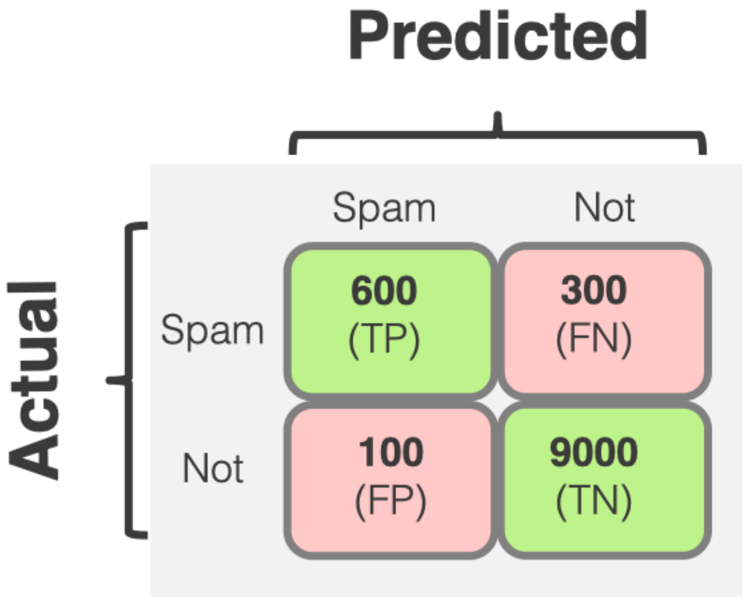
BYU

RESULTS OF DIFFERENT THRESHOLDS



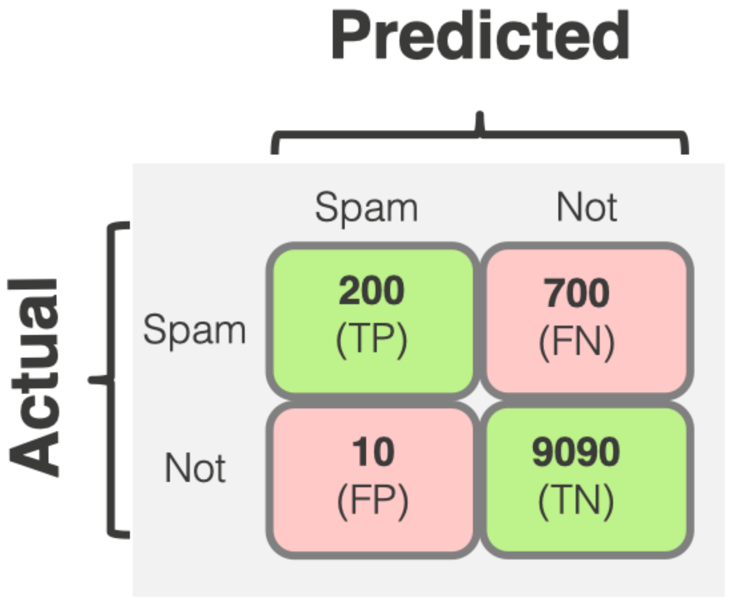
		Predicted	
		Spam	Not
Actual	Spam	800 (TP)	100 (FN)
	Not	500 (FP)	8600 (TN)

Threshold: 0.5



		Predicted	
		Spam	Not
Actual	Spam	600 (TP)	300 (FN)
	Not	100 (FP)	9000 (TN)

Threshold: 0.8



		Predicted	
		Spam	Not
Actual	Spam	200 (TP)	700 (FN)
	Not	10 (FP)	9090 (TN)

Threshold: 0.95

BYU

PRECISION-RECALL TRADEOFF EXAMPLES

1. Medical Diagnostics (Want High Recall)

- Catch all true cases (↓ false negatives).
- May increase false positives.

2. Email Spam Filtering (Want High Precision)

- Avoid flagging real emails as spam.
- May miss some spam messages.

3. Fraud Detection in Finance

- High recall catches more fraud.
- High precision reduces false alarms.

5. Product Recommendation Systems

- High precision: relevant suggestions.
- High recall: diverse recommendations.

ROC CURVE PYTHON EXAMPLE

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
```

```python
# 1. Load the Breast Cancer dataset
data = load_breast_cancer()

# 2. Split the dataset into features (X) and labels (y)
X = data.data
y = data.target

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

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

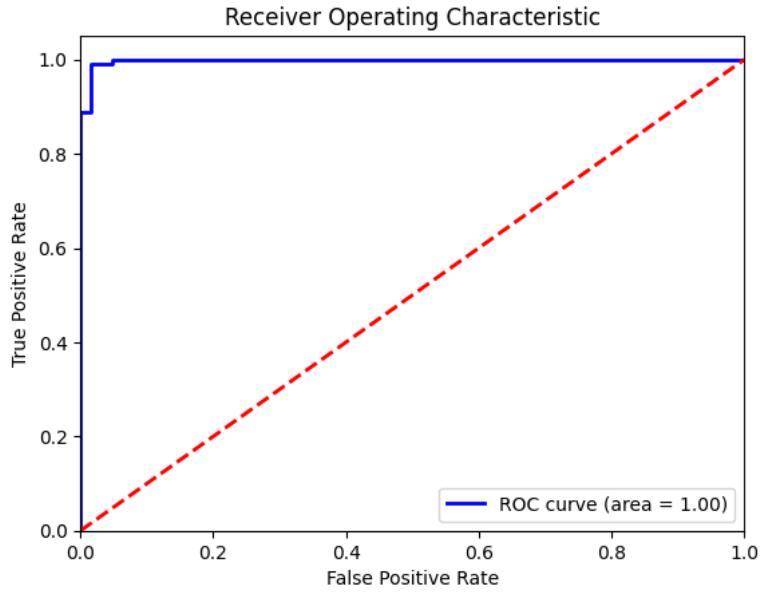
```python
# 4.  Initialize and train the model
model = LogisticRegression()
model.fit(X_train, y_train)

# 5. Predict probabilities for the positive class
y_prob = model.predict_proba(X_test)[:, 1]
```

```python
# 6. Compute ROC Curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
# Calculate the AUC
roc_auc = auc(fpr, tpr)
```

```python
# 7. Plot the ROC Curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC c
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyl
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```

```python
# Create a DataFrame
roc_df = pd.DataFrame({'FPR': fpr, 'TPR': tpr,
                       'Thresholds': thresholds})

# Display the DataFrame
roc_df
```

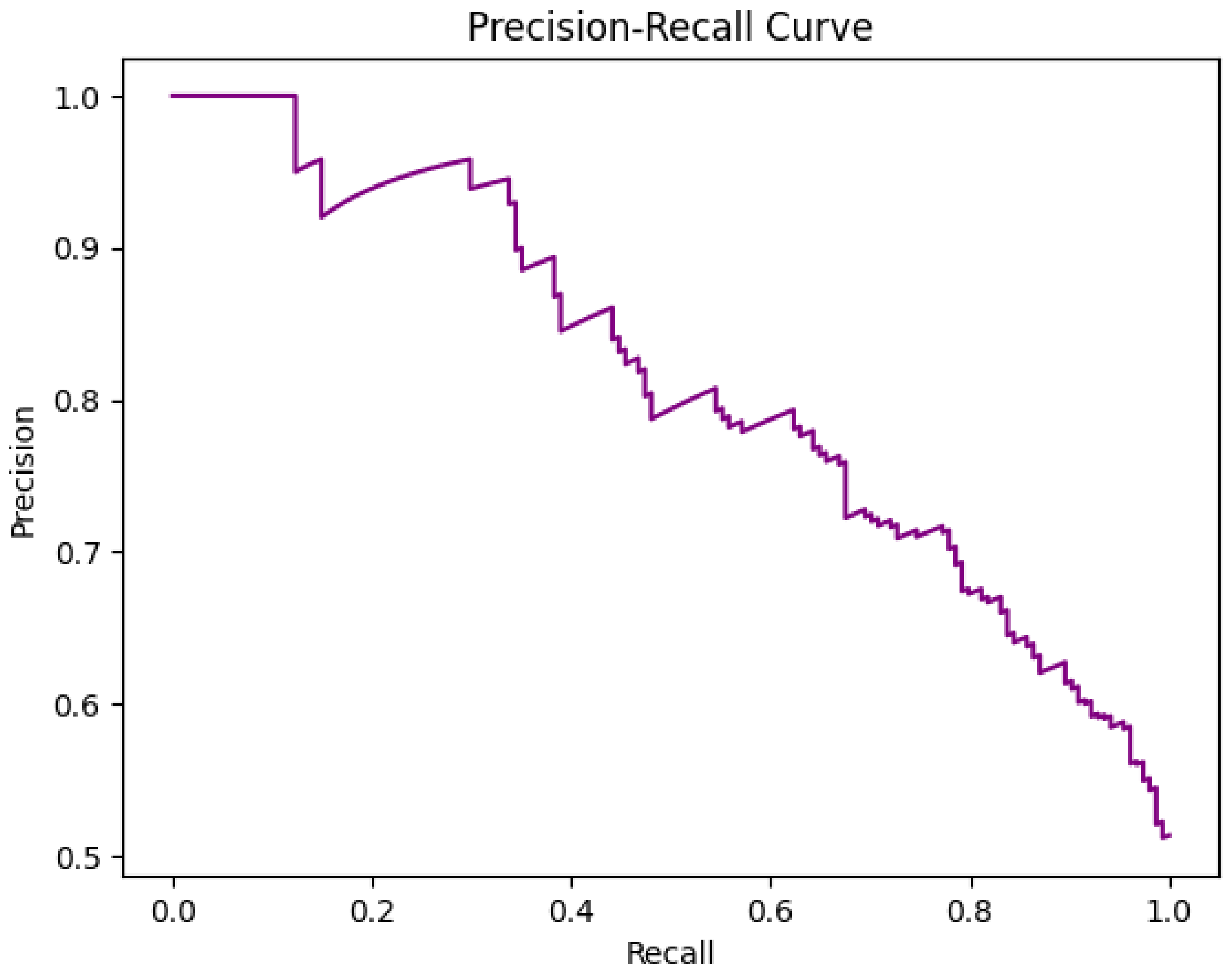
```
        FPR       TPR    Thresholds
0  0.000000  0.000000           inf
1  0.000000  0.009259  9.999999e-01
2  0.000000  0.888889  9.347860e-01
3  0.015873  0.888889  9.268869e-01
4  0.015873  0.990741  4.811814e-01
5  0.047619  0.990741  3.780302e-01
6  0.047619  1.000000  2.246681e-01
7  1.000000  1.000000  1.034436e-12
```



https://colab.research.google.com/drive/16DoMKs2_VVf5DoMXqtr4atTSi6Ij4PUd#scrollTo=SntAkaLG2aN9

PRECISION – RECALL CURVE

- The precision-recall curve plots **precision** (y-axis) against **recall** (x-axis) for different **threshold** values.
- Precision-recall curves are beneficial when dealing with **imbalanced datasets**, where the positive class is rare.
- In these scenarios, traditional metrics like accuracy can be misleading, as a model might achieve high accuracy by simply predicting the majority class.
- The precision-recall curve provides a more nuanced evaluation, focusing on the performance of the minority class.



SUMMARY

This lecture provided a comprehensive overview of various model performance metrics, emphasizing their importance in evaluating and comparing machine learning models. Understanding these metrics is crucial for building and deploying effective models in various applications. Some of the main topics discussed included:

- Understanding Model Fit: Bias-Variance Tradeoff
- Data Splitting: Ensuring Model Generalization
- Regression Metrics: Measuring Prediction Accuracy
- Classification Metrics: Evaluating Predictive Power
- ROC Curve and AUC: Visualizing and Quantifying Classifier Performance
- Precision-Recall Curve: Focusing on Positive Predictions

BYU

QUIZ

Instructions: Answer the following questions in 2-3 sentences each.

1. **Explain the difference between bias and variance in the context of machine learning.**
2. **What is the purpose of splitting data into training, validation, and test sets?**
3. **Describe the advantages of using RMSE over MSE as a regression metric.**
4. **When is accuracy not a reliable metric for evaluating model performance?**
5. **What are the four categories of prediction results in a confusion matrix?**
6. **Differentiate between precision and recall.**
7. **Why is the F1-score considered a balanced metric?**
8. **How does a ROC curve visualize the performance of a binary classifier?**
9. **Explain the concept of a classification threshold and its influence on precision and recall.**
10. **What are the benefits of using cross-validation in model evaluation?**

BYU