ROC Curve (IMP)

USED WITH BINARY CLASSIFICATION (0, 1)

```
y_scores = model.predict_proba(X_test)[:,1]

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_scores)


from sklearn.metrics import roc_auc_score
roc_auc = roc_auc_score(y_test, y_scores)
```

Threshold selection:

- In classification problems, the model does not give us ~~0 or 1 output~~.
- It gives us a number from 0 to 1.
  - eg. 0.45, 0.34, 0.56, etc.
- You convert these 👆 numbers to → 0 or 1 by deciding a **threshold**.
- ***eg. if Threshold= 0.5:***
  - ***Scores below 0.5 → 0***
  - ***Scores above 0.5 → 1***
- But the 0.5 threshold is not always perfect.
- Eg. in case of spam detection, you wouldn't want an important mail going to spam.
  - So, you can increase the threshold from **0.5 to → 0.75**.

    - Here, we want to **reduce** the **false positive.**
- **WE USE ROC CURVE TO DECIDE THIS THRESHOLD.**

Confusion Matrix

- The **confusion matrix** is a **2×2** matrix (in binary classification) that shows the performance of a classification model in terms of its true positive, true negative, false positive, and false negative predictions.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Overview

- **ROC Curve (Receiver Operating Characteristic Curve)**: A graphical plot that illustrates the performance of a binary classification model.
- **Purpose**: Evaluates the trade-off between the **True Positive Rate (TPR)** and **False Positive Rate (FPR)** across different threshold settings.
- **Use Case**: Commonly used to compare models and select the best threshold for classification

1. **True Positive Rate (TPR)**, also known as **Sensitivity** or **Recall**:

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

- **TP**: True Positives (correctly predicted positive cases)
- **FN**: False Negatives (incorrectly predicted negative cases)

2. **False Positive Rate (FPR)**:

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

- **FP**: False Positives (incorrectly predicted positive cases)
- **TN**: True Negatives (correctly predicted negative cases)

True Positive Rate (TPR) (Benefit):

- ***In en email spam classifier, out of all the spams, how many emails are we able to label spam?***
- **If there are 100 spam mails & you label 80 as spam → TPR = 80%**
- You want to maximize this.
- **TPR will be 100% when FN is zero.**

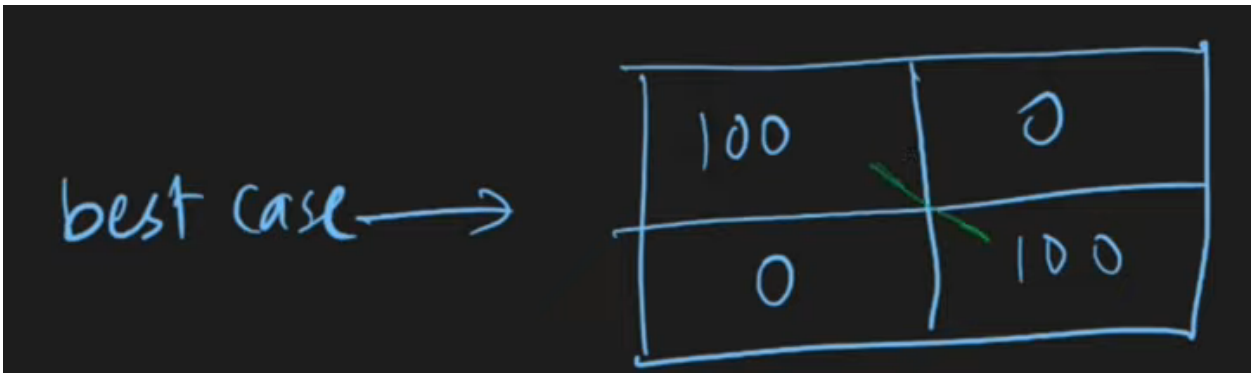
False Positive Rate (FPR) (Cost):

💡 **FPR Intuition: Out of all the non spam emails, how many you classified as spam?**

Best Case

TPR =1 (When FN = 0)

FPR = 0 (When FP=0)



The ROC curve is plotted with:

- **False Positive Rate (FPR)** on the x-axis (1 - Specificity).
- **True Positive Rate (TPR)** on the y-axis (Sensitivity).

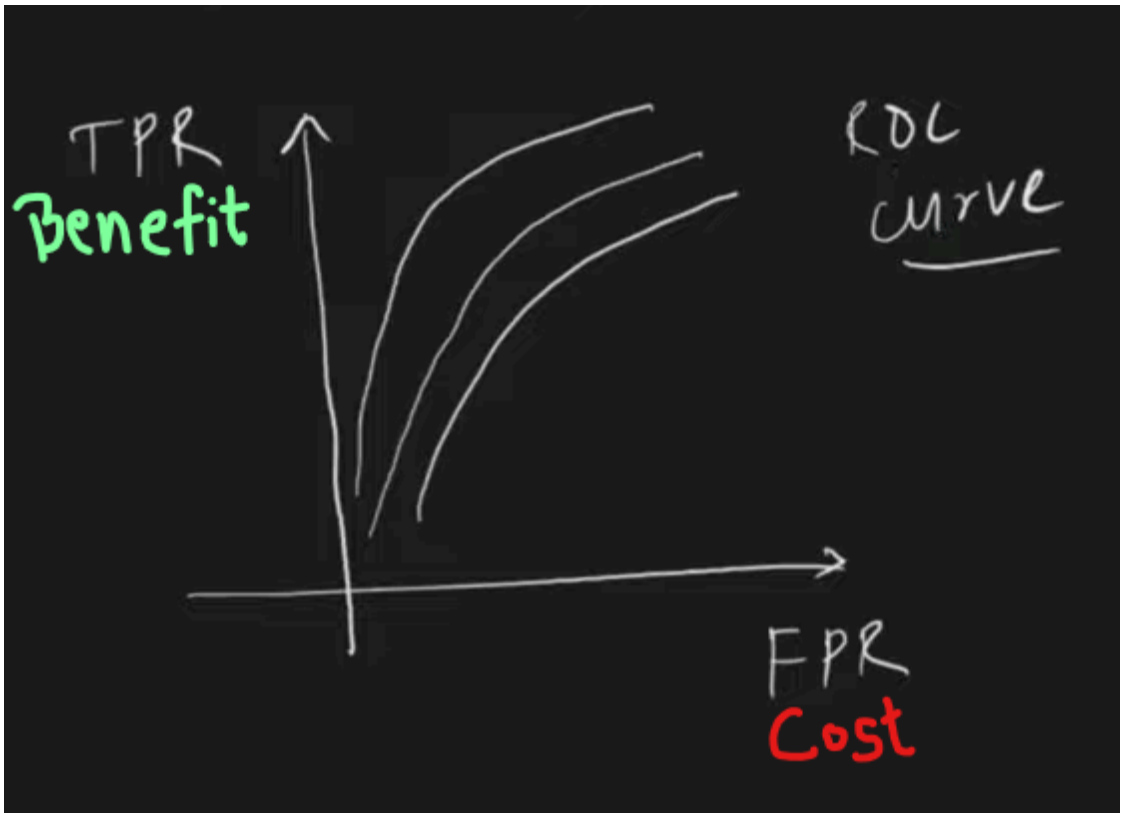
AUC (Area Under the Curve):

- Represents the model's ability to distinguish between classes.
- **AUC = 1**: Perfect classifier.
- **AUC = 0.5**: Random classifier.
- **AUC < 0.5**: The classifier is worse than random guessing.

How to Plot an ROC Curve

1. **Train a Binary Classifier**: Use a model like Logistic Regression, SVM, or Random Forest.
2. **Get Predicted Probabilities**: Use `predict_proba()` to get the probability of the positive class.

3. **Try different Thresholds**: Compute TPR and FPR for different thresholds.
    a. You'll get 1 confusion matrix for every threshold
    b. Calculate TPR & FPR
4. **Plot the Curve**: Plot TPR vs. FPR.
5. **Calculate AUC**: The **AUC (Area Under the Curve)** is a measure of classifier performance and can be computed using `roc_auc_score`.



Deciding the Threshold

- If threshold is small → ~0.1, most of the emails will go to spam.
    - TPR will **increase**
    - FPR will **increase**
- If it's large → ~0.99, most of the spams will go to inbox.

- TPR will **decrease**
- FPR will **decrease**

Python Code for ROC

- We'll use the diabetes dataset

```
import pandas as pd

data = pd.read_csv('https://raw.githubusercontent.com/npradaschnor/Pima-Indians-Diabetes-Dataset/master/diabetes.csv')

data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
X = data.drop('Outcome', axis=1)
y = data['Outcome']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)
```

Train logical regression model on this data:

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

```
y_scores = model.predict_proba(X_test)[:,1]
```

Why [:, 1] ?

- **Output**: For a binary classification problem, it returns a 2D array where:
  - The first column (`[:, 0]`) contains the probabilities of the **negative class** (class 0).
  - The second column (`[:, 1]`) contains the probabilities of the **positive class** (class 1).

```
y_scores
```

```
array([0.04953981, 0.17459294, 0.09362961, 0.25508355, 0.63549111,
       0.11671452, 0.06567364, 0.421926  , 0.04865994, 0.57564702,
       0.3387436 , 0.41301333, 0.69854608, 0.19964664, 0.02001639,
       0.82469745, 0.86657883, 0.03101331, 0.25509887, 0.89492209,
       0.9524559 , 0.8347648 , 0.11741172, 0.44671697, 0.08923667,
       0.06882315, 0.651158  , 0.41198913, 0.1786156 , 0.28690104,
       0.24547132, 0.43901472, 0.00942479, 0.24260196, 0.35170374,
```

Apply ROC

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_scores)
```

```
thresholds
```

```
array([       inf, 0.99355918, 0.95264287, 0.9524559 , 0.82469745,
       0.76881955, 0.76200343, 0.68385951, 0.67531274, 0.67422254,
       0.651158  , 0.65085411, 0.63549111, 0.58629047, 0.57564702,
       0.56972966, 0.52515955, 0.4702016 , 0.46129519, 0.45043552,
       0.43901472, 0.43865415, 0.42226641, 0.4199088 , 0.41301333,
       0.41198913, 0.40477451, 0.37971089, 0.36866773, 0.31018821,
       0.30880881, 0.27772611, 0.2720497 , 0.25509887, 0.25508355,
       0.24742118, 0.24547132, 0.23663505, 0.22888031, 0.18185267,
       0.1786156 , 0.17459294, 0.17327014, 0.14637618, 0.14076528,
       0.11671452, 0.11494775, 0.11000816, 0.10666912, 0.08923667,
       0.08672697, 0.02001639, 0.0193133 , 0.00160795])
```

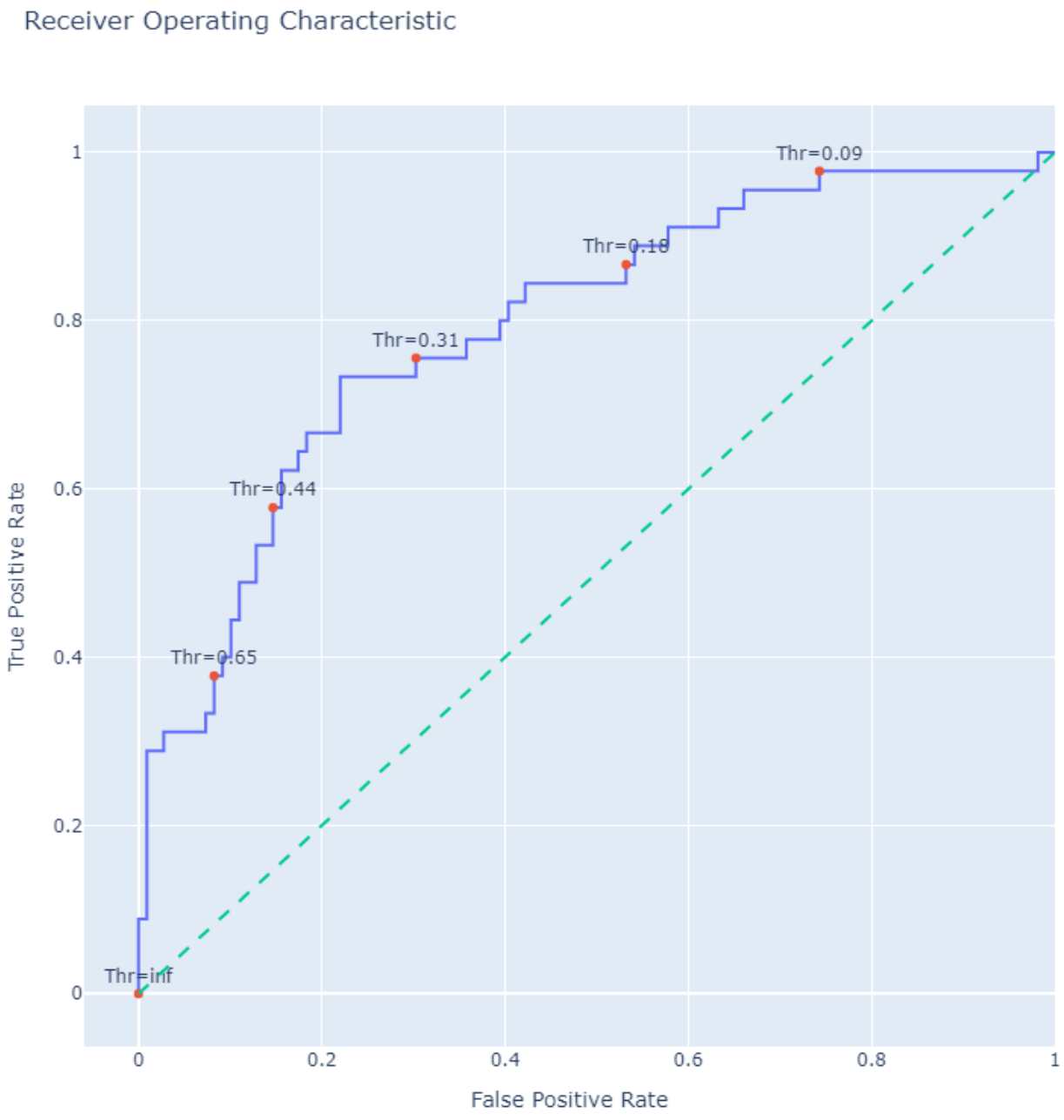
- It generated thresholds from **0 to 1**
- We calculated FPR & TPR for every threshold

Now plot these values on a curve

```
import plotly.graph_objects as go
import numpy as np


# Generate a trace for ROC curve
trace0 = go.Scatter(
    x=fpr,
    y=tpr,
    mode='lines',
    name='ROC curve'
)

# Only label every nth point to avoid cluttering
n = 10
indices = np.arange(len(thresholds)) % n == 0  # Choose indices where index
mod n is 0
```

```
trace1 = go.Scatter(
    x=fpr[indices],
    y=tpr[indices],
    mode='markers+text',
    name='Threshold points',
    text=[f"Thr={thr:.2f}" for thr in thresholds[indices]],
    textposition='top center'
)


# Diagonal line
trace2 = go.Scatter(
    x=[0, 1],
    y=[0, 1],
    mode='lines',
    name='Random (Area = 0.5)',
    line=dict(dash='dash')
)

data = [trace0, trace1, trace2]

# Define layout with square aspect ratio
layout = go.Layout(
    title='Receiver Operating Characteristic',
    xaxis=dict(title='False Positive Rate'),
    yaxis=dict(title='True Positive Rate'),
    autosize=False,
    width=800,
    height=800,
    showlegend=False
)

# Define figure and add data
fig = go.Figure(data=data, layout=layout)
```

```
# Show figure
fig.show()
```



Find out optimal threshold:

```
# Assume that fpr, tpr, thresholds have already been calculated
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print("Optimal threshold is:", optimal_threshold)

Output:
Optimal threshold is: 0.36866773215712084
```

AUC-ROC

- The AUC-ROC measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1).
- AUC provides an aggregate measure of performance across all possible classification thresholds.

- An AUC of **1** indicates that the model has **perfect discrimination**: it correctly classifies all positive and negative instances.
- An AUC of **0.5** suggests the model has **no discrimination ability**: it is as good as random guessing.
- An AUC of **0** indicates that the model is perfectly **wrong**: it classifies all positive instances as negative and all negative instances as positive.

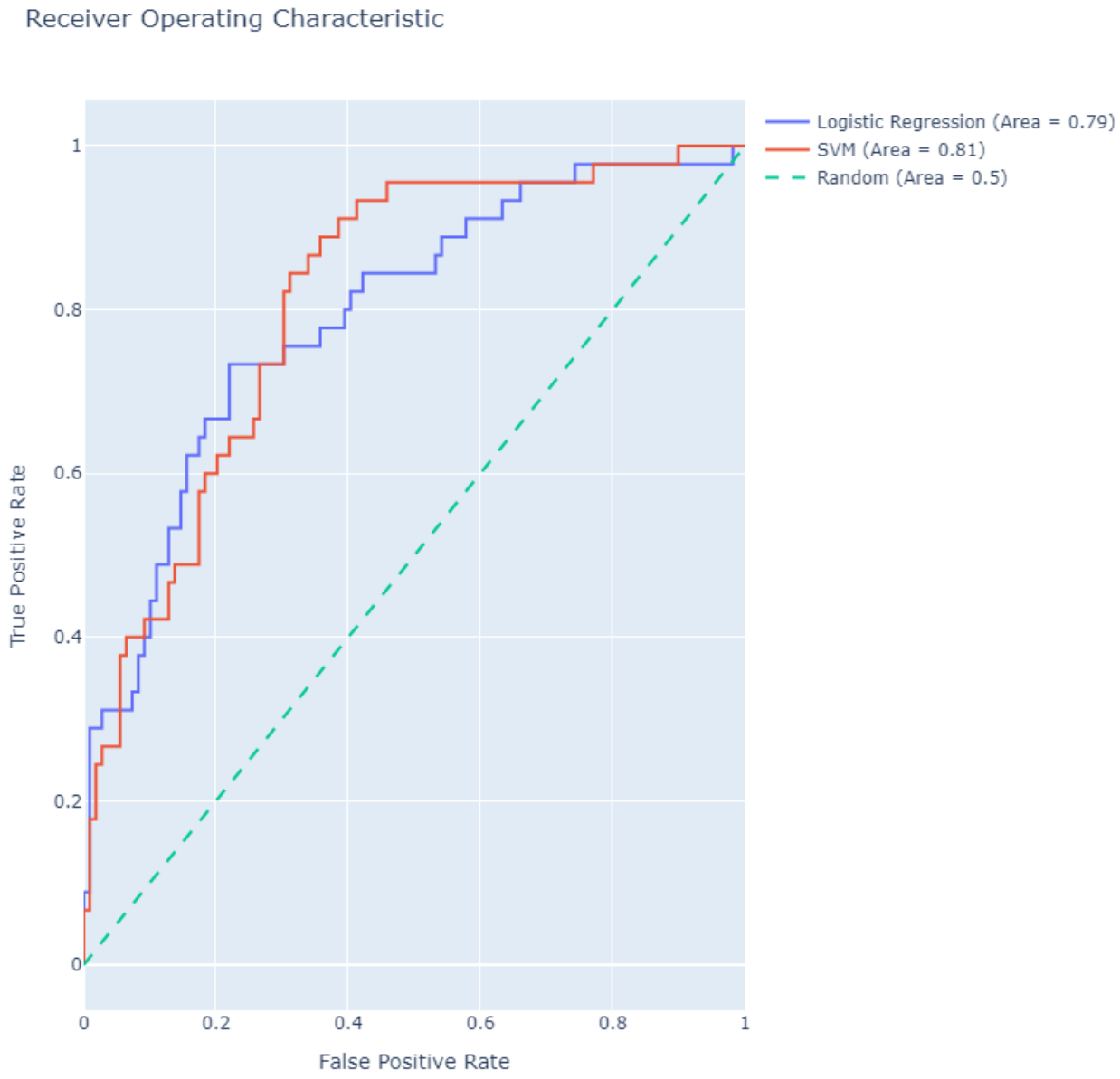
> 💡 **In practice, AUC values usually fall between 0.5 (random) and 1 (perfect), with higher values indicating better classification performance.**

Compare Logistic Regression vs SVM

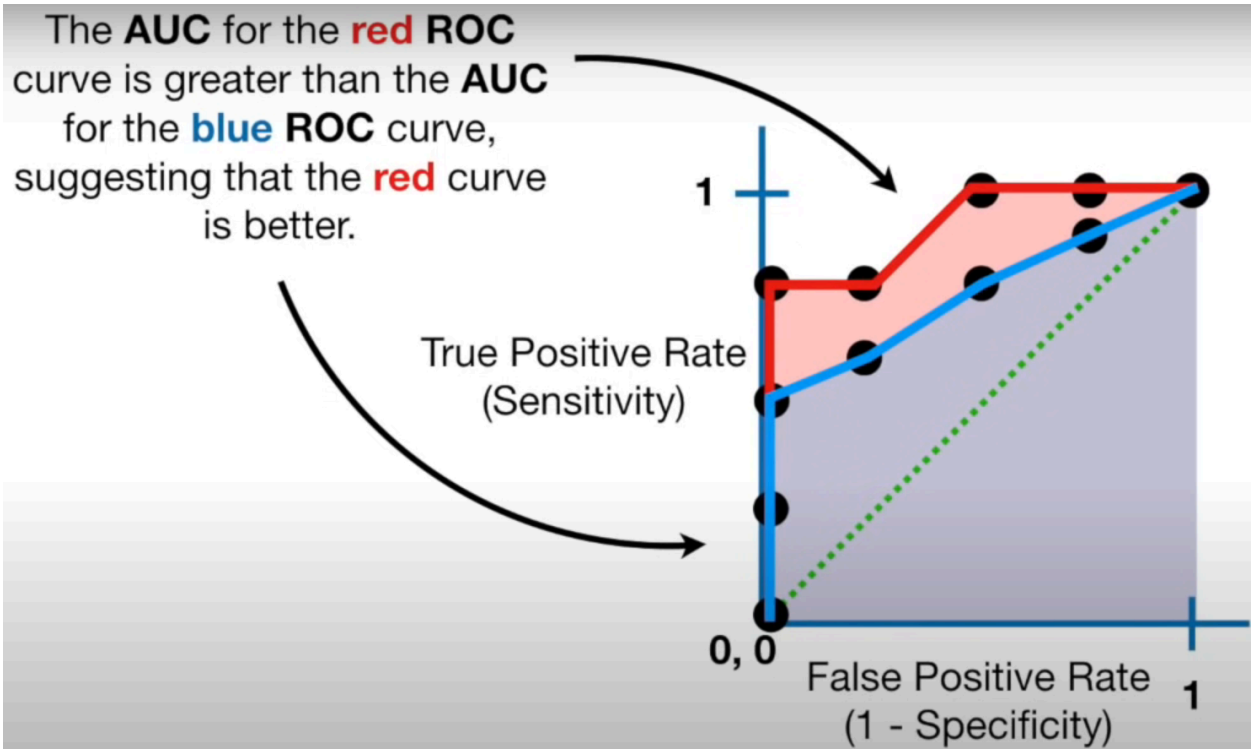
```
import numpy as np
import plotly.graph_objects as go
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import StandardScaler

# Assuming that X_train, X_test, y_train, y_test are already defined

# SVM requires feature scaling for better performance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Logistic Regression model
lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train, y_train)
lr_scores = lr_model.predict_proba(X_test)[:,1]

# SVM model
svm_model = SVC(probability=True)
svm_model.fit(X_train_scaled, y_train)
svm_scores = svm_model.predict_proba(X_test_scaled)[:,1]

# Generate ROC curve data for logistic regression model
lr_fpr, lr_tpr, lr_thresholds = roc_curve(y_test, lr_scores)
lr_auc = roc_auc_score(y_test, lr_scores)

# Generate ROC curve data for SVM model
svm_fpr, svm_tpr, svm_thresholds = roc_curve(y_test, svm_scores)
svm_auc = roc_auc_score(y_test, svm_scores)

# Generate a trace for the Logistic Regression ROC curve
trace0 = go.Scatter(
    x=lr_fpr,
```

```
    y=lr_tpr,
    mode='lines',
    name=f'Logistic Regression (Area = {lr_auc:.2f})'
)

# Generate a trace for the SVM ROC curve
trace1 = go.Scatter(
    x=svm_fpr,
    y=svm_tpr,
    mode='lines',
    name=f'SVM (Area = {svm_auc:.2f})'
)

# Diagonal line
trace2 = go.Scatter(
    x=[0, 1],
    y=[0, 1],
    mode='lines',
    name='Random (Area = 0.5)',
    line=dict(dash='dash')
)

data = [trace0, trace1, trace2]

# Define layout with square aspect ratio
layout = go.Layout(
    title='Receiver Operating Characteristic',
    xaxis=dict(title='False Positive Rate'),
    yaxis=dict(title='True Positive Rate'),
    autosize=False,
    width=800,
    height=800,
    showlegend=True
)

# Define figure and add data
```

```
fig = go.Figure(data=data, layout=layout)

# Show figure
fig.show()
```



- Here, SVM is better classifier.



Precision

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

- Use precision in case of rare disease.
- It does not include true negative in its calculation.

Q. Where does `model.predict()` come into play and how do we adjust the threshold to obtain desired results?

`model.predict()`

- **What it does**: Predicts class labels (0 or 1) for the input data.
- **Default Behavior**: Uses a **threshold of 0.5** to convert predicted probabilities into class labels.
  - If the predicted probability of the positive class (`P(y=1)`) is ≥ 0.5, the sample is classified as **1**.
  - If `P(y=1) < 0.5`, the sample is classified as **0**.

`model.predict_proba()`

- **What it does**: Returns the predicted probabilities for each class.
- **Output**: A 2D array where:
  - The first column (`[:, 0]`) contains probabilities for the **negative class** (class 0).
  - The second column (`[:, 1]`) contains probabilities for the **positive class** (class 1).

Ex. `y_pred_proba = model.predict_proba(X_test)`

Adjusting the Threshold:

- **Why Adjust the Threshold?**:
  - To balance metrics like **precision**, **recall**, and **FPR** based on your use case.
  - For example:
    - **High Recall**: Lower the threshold to reduce false negatives (e.g., medical diagnosis).
    - **High Precision**: Increase the threshold to reduce false positives (e.g., spam detection).
- **How to Adjust**:
  1. Use `predict_proba()` to get probabilities.

2. Apply a custom threshold to convert probabilities into class labels.

Example: Adjusting the Threshold

```
import numpy as np
from sklearn.metrics import classification_report

# Get predicted probabilities for the positive class
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Default threshold (0.5)
y_pred_default = (y_pred_proba >= 0.5).astype(int)

# Custom threshold (e.g., 0.3 for higher recall)
custom_threshold = 0.3
y_pred_custom = (y_pred_proba >= custom_threshold).astype(int)

# Compare results
print("Default Threshold (0.5):")
print(classification_report(y_test, y_pred_default))

print(f"\nCustom Threshold ({custom_threshold}):")
print(classification_report(y_test, y_pred_custom))
```

Default Threshold (0.5):

	precision	recall	f1-score	support
0	0.85	0.90	0.87	50
1	0.89	0.84	0.86	50
accuracy			0.87	100
macro avg	0.87	0.87	0.87	100
weighted avg	0.87	0.87	0.87	100

Custom Threshold (0.3):

	precision	recall	f1-score	support
0	0.80	0.94	0.86	50
1	0.92	0.76	0.83	50
accuracy			0.85	100
macro avg	0.86	0.85	0.85	100
weighted avg	0.86	0.85	0.85	100

`y_pred_default = (y_pred_proba >= 0.5).astype(int)`

- **Purpose**: Converts predicted probabilities into class labels using the **default threshold of 0.5**.
- **Steps**:
  1. `y_pred_proba >= 0.5`: Compares each predicted probability to the threshold (0.5).
     - Returns `True` if the probability is ≥ 0.5.
     - Returns `False` if the probability is < 0.5.
  2. `.astype(int)`: Converts the boolean values (`True`/`False`) to integers (`1`/`0`).
     - `True` becomes `1` (positive class).

    - `False` becomes `0` (negative class).

- **Precision**: How many predicted positives are actually positive.
- **Recall**: How many actual positives are correctly predicted.
- **F1-Score**: A balance between precision and recall.
- **Support**: Number of samples in each class.

Key Takeaways

- `model.predict()` : Uses a default threshold of 0.5 to predict class labels.
- `model.predict_proba()` : Provides predicted probabilities, allowing you to adjust the threshold.
- **Adjusting the Threshold**:
    - **Lower the threshold** to **increase recall** (**reduce false negatives**).
    - **Raise the threshold** to **increase precision** (**reduce false positives**).