Mechatronics

Day 5
Supervised Learning
Classification

Outline

- Classification Problem
- Logistic Regression
- Classification model evaluation
  - Confusion matrix
  - ROC and AUC Score
- Support Vector Machine
- K-Nearest Neighbors
- Decision Tree
- Unsupervised Learning : K-Means Clustering

Classification Problem

Machine Failure Classification using sensor data

Features :

Temperature (°C), Vibration (Hz), Power Usage (kW), Humidity (%), Machine Type (e.g., "Drill", "Lathe", "Mill")

Targets/Classes

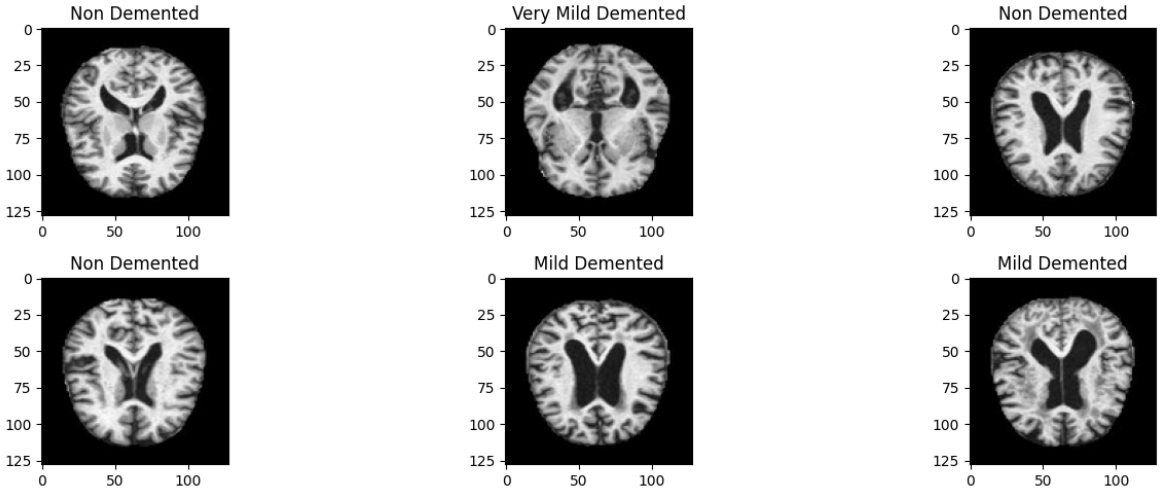
Failure Risk, It is a binary label (0 = normal operation and 1 = machine is at risk of failure)

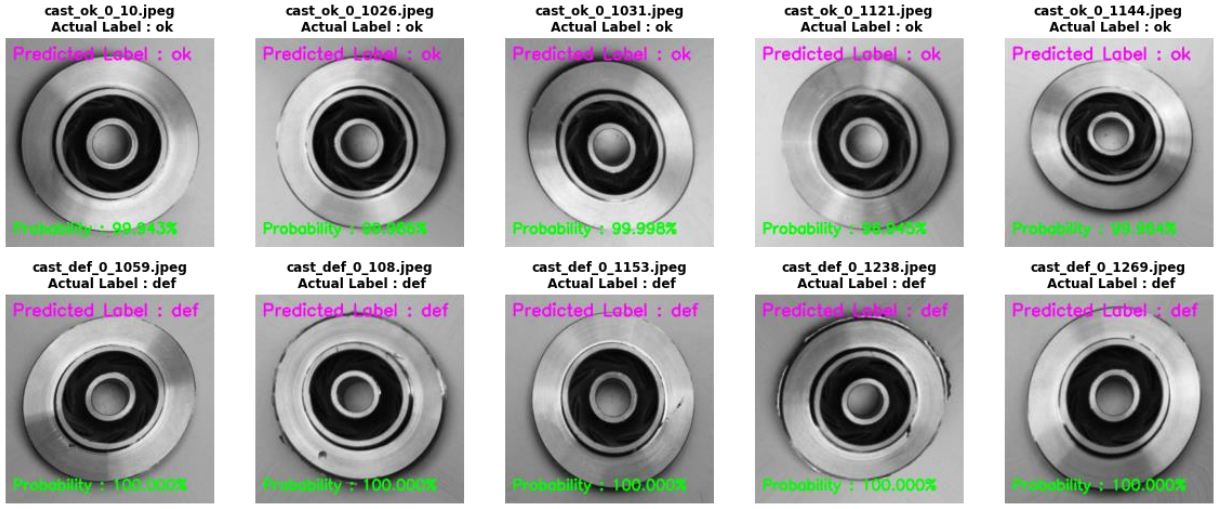
```
Logistic Regression
Accuracy: 0.7010309278350515
Confusion Matrix:
 [[136   0]
 [ 58   0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.70      1.00      0.82       136
           1       0.00      0.00      0.00        58

    accuracy                           0.70       194
   macro avg       0.35      0.50      0.41       194
weighted avg       0.49      0.70      0.58       194
```

Classification Problem (Image data) – DL models

Alzheimer MRI Disease Classification Dataset

https://www.kaggle.com/datasets/borhanitrash/alzheimer-mri-disease-classification-dataset





Casting product image data for quality inspection

https://www.kaggle.com/datasets/ravirajsinh45/real-life-industrial-dataset-of-casting-product

Logistic Regression

Probability (P)

$$P = \frac{Number\ of\ interest\ event}{Number\ of\ all\ posible\ event}$$

$$0 \leq P \leq 1$$

Odds ratio

probability that event A happens divided by probability that event A not happens.

$$odds = \frac{P}{1 - P}$$

Logistic Regression

Logit function (Natural Logarithms of Odds ratio)

$$Logit = \ln(odds) = \ln\left(\frac{P}{1-P}\right)$$

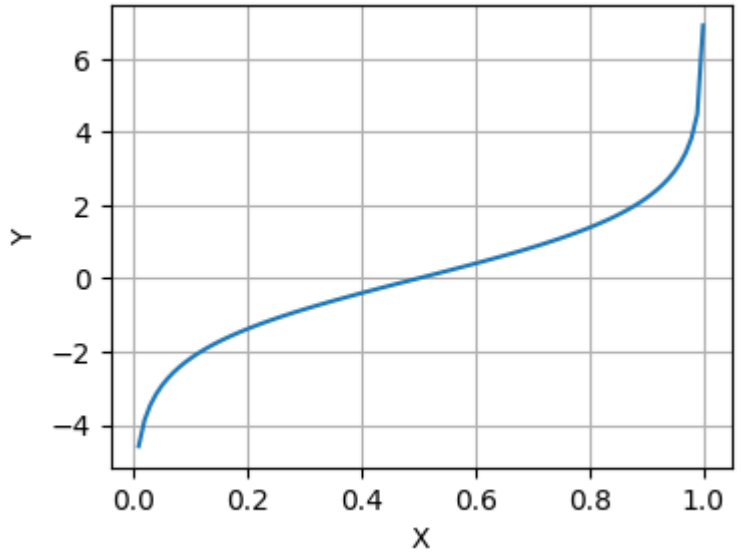
$$-\infty \leq Logit \leq \infty$$

$$\text{Logit}(0) = \ln\left(\frac{0}{1-0}\right) = \ln(0) = -\infty \qquad \text{Logit}(1) = \ln\left(\frac{1}{1-1}\right) = \ln(\infty) = \infty$$

Logistic Regression

Plot logit curve

```
# Logit curve
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 0.999, num=100)
y = np.log(x/(1-x))   # for numpy : method log() = Natural
Log

plt.figure(figsize=(4, 3))
plt.plot(x, y)
plt.grid()
plt.show()
```



Logistic Regression

Sigmoid function

$$Sigmoid = \frac{1}{1 + e^{-logit}}$$

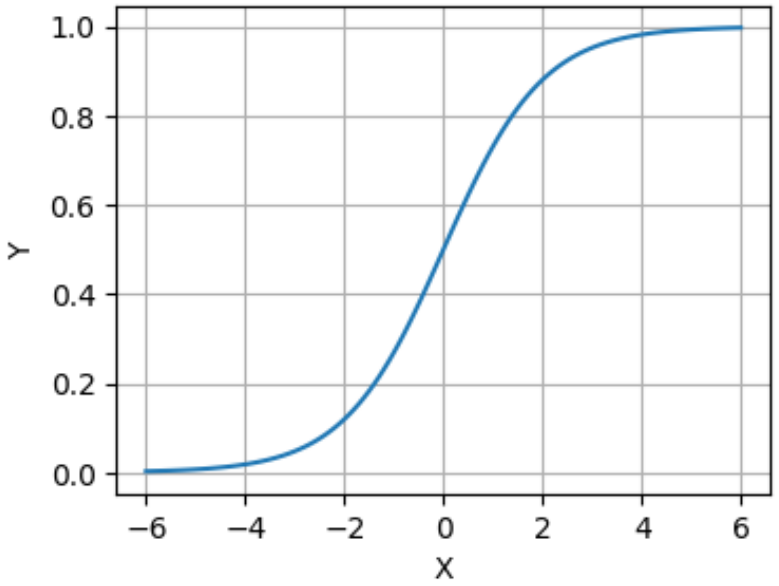
$$f(x) = \frac{1}{1 + e^{-x}}$$

$$0 \leq Sigmoid \leq 1$$

Logistic Regression

Plot Sigmoid curve for Logistic function

```
# Sigmoid curve
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-6, 6, 121)
y = 1/(1 + np.exp(-x))        # exp(x) = e**x

plt.figure(figsize=(4, 3))
plt.plot(x, y)
plt.xlabel('X')
plt.ylabel('Y')
plt.grid()
plt.show()
```



Logistic Regression

Sigmoid function

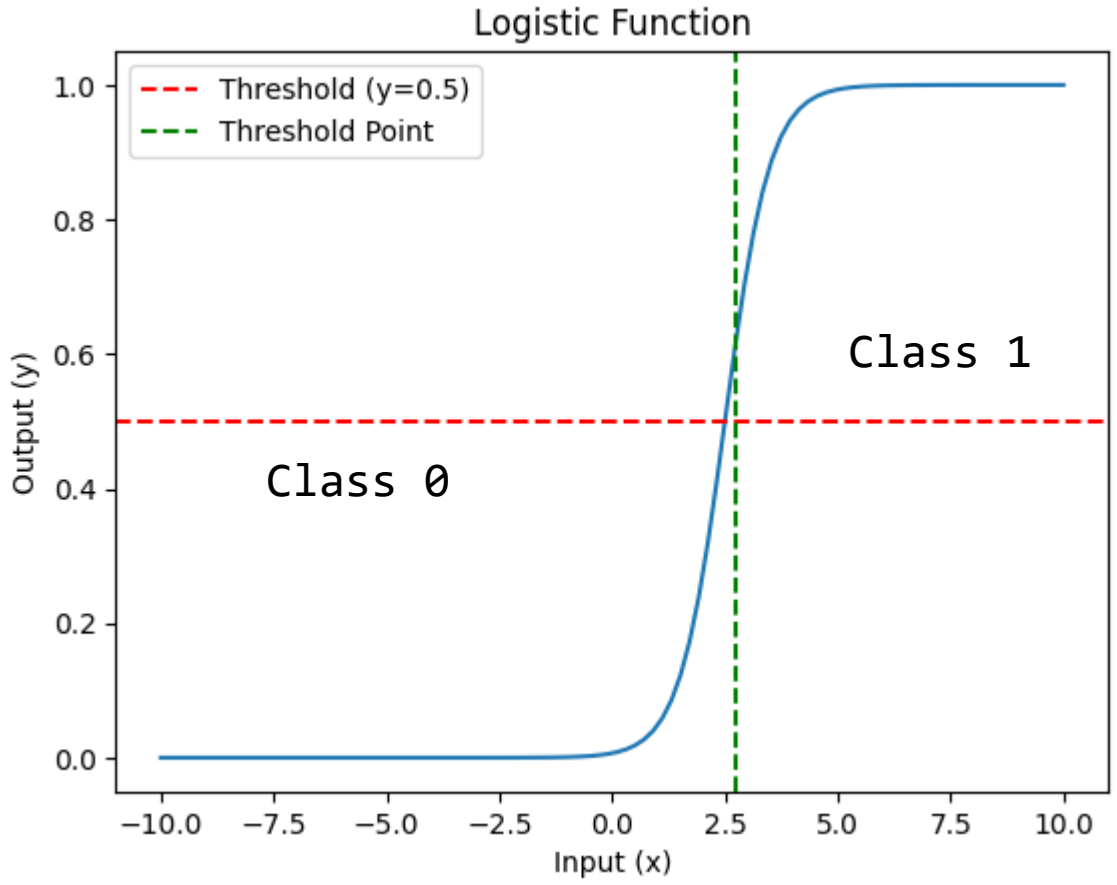
$$f\left(x^{(i)}\right) = y^{(i)} = wx^{(i)} + b$$

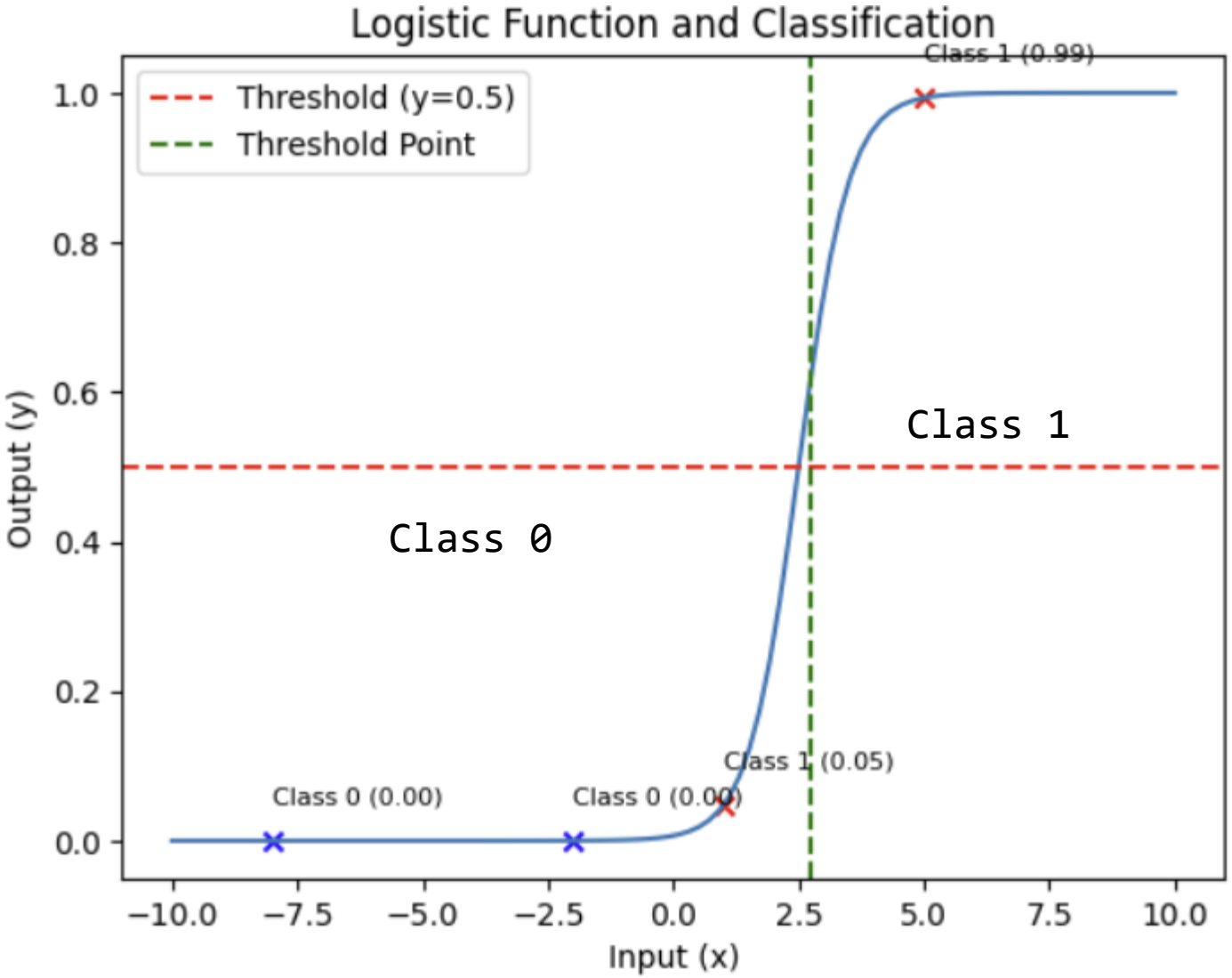
$$P\left(x^{(i)}\right) = \frac{1}{1 + e^{-(wx^{(i)}+b)}}$$

$$0 \leq P\left(x^{(i)}\right) \leq 1$$

$$0 \leq P\left(x^{(i)}\right) \leq 0.5 : Class\ 0$$

$$0.5 \leq P\left(x^{(i)}\right) \leq 0.1 : Class\ 1$$





Threshold Point (x): 2.75

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
```

Example 1 : Logistic Regression Model

```python
# Training Logistic Regrssion model

from sklearn.linear_model import LogisticRegression

model = LogisticRegression()

x = [[8, 6], [3, 5], [4, 9], [5, 8], [9, 9]]  #[[x1, x2], ...]
y = ['yes', 'no', 'no', 'yes', 'yes']

model.fit(x, y)

# Predict when x1=4, x2=4, y=? และ x1=5, x2=5, y=?
x_predict = [[4, 4], [5, 5]]
y_predict = model.predict(x_predict)

print('if x = [4, 4] then y =', y_predict[0])
print('if x = [5, 5] then y =', y_predict[1])
print()
print('Probability [1-P   P] : ')
prob = model.predict_proba(x_predict)
print(prob)
```

```
if x = [4, 4] then y = no
if x = [5, 5] then y = yes

Probability [1-P   P] :
[[0.69084069 0.30915931]
 [0.46545907 0.53454093]]
```

Example 2 : Predict student grading (Pass/Fail) from study hours

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
df = pd.read_csv('logis_hours_study.csv')
with pd.option_context('display.max_rows', 6): display(df)

x = np.array(df['hours_study']).reshape(-1, 1)
y = df['pass’]

model = LogisticRegression()
model.fit(x, y)
x_predict = [2.2, 3.3, 4.4]
x_predict = np.array(x_predict).reshape(-1, 1)
y_predict = model.predict(x_predict)
print('Prediction:’)

for (i, xp) in enumerate(x_predict):
    # defined: 0 = Fail, 1 = Pass
    r = 'Fail' if y_predict[i] == 0 else 'Pass'
    print(f'Study: {xp[0]} hour(s) => {r}')
print()
print('Logistic Function:')
ic = '{:.2f}'.format(model.intercept_[0])
ce = '{:.2f}'.format(model.coef_[0, 0])
print(f'P(x) = 1 / (1 + exp({ic} + ({ce})x))')
```

	hours_study	pass
0	0.50	0
1	0.75	0
2	1.00	0
...	...	...
17	4.75	1
18	5.00	1
19	5.50	1

20 rows × 2 columns

```
Prediction:
Study: 2.2 hour(s) => Fail
Study: 3.3 hour(s) => Pass
Study: 4.4 hour(s) => Pass

Logistic Function:
P(x) = 1 / (1 + exp(-3.14 + (1.15)x))
```

Example 3 : Classify Machine spec: OK/NOK (Not OK)

	machine_age_months	operate_hours_per_day	machine_meets_spec
0	57	4	1
1	73	5	0
2	22	5	1
...	...	...	...
17	71	7	0
18	35	4	0
19	44	5	1

20 rows × 3 columns

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
df = pd.read_excel('machine-spec.xlsx')
with pd.option_context('display.max_rows', 6): display(df)
x = df[['machine_age_months', 'operate_hours_per_day']]
y = df['machine_meets_spec']
scaler = StandardScaler()
x = scaler.fit_transform(x)
model = LogisticRegression()
model.fit(x, y)
x_predict = [[30, 6], [40, 8], [50, 5], [60, 3]]
x_predict_sc = scaler.transform(x_predict)
y_predict = model.predict(x_predict_sc)
print('Prediction:')
for (i, xp) in enumerate(x_predict):
    mp = 'NOK' if y_predict[i] == 0 else 'OK'
    t = f'Machine Age: {xp[0]} Month(s), '
    t += f'Operate: {xp[1]} Hour(s)/Day '
    t += f'=> Meet Spec: {mp}'
    print(t)
print('Logistic Function:')
ic = '{:.2f}'.format(model.intercept_[0])
ce1 = '{:.2f}'.format(model.coef_[0, 0])
ce2 = '{:.2f}'.format(model.coef_[0, 1])
print(f'P(x) = 1 / (1 + exp({ic} + ({ce1})age + ({ce2})hour))')
```

```
Prediction:
Machine Age: 30 Month(s), Operate: 6 Hour(s)/Day => Meet Spec: OK
Machine Age: 40 Month(s), Operate: 8 Hour(s)/Day => Meet Spec: NOK
Machine Age: 50 Month(s), Operate: 5 Hour(s)/Day => Meet Spec: NOK
Machine Age: 60 Month(s), Operate: 3 Hour(s)/Day => Meet Spec: OK

Logistic Function:
P(x) = 1 / (1 + exp(-0.14 + (-1.16)age + (-0.98)hour))
```

Classification model's Evaluation

1. Accuracy	Generally applicable, but can be misleading in imbalanced datasets.
2. Precision	Useful when minimizing false positives is crucial (e.g., spam detection).
3. Recall (Sensitivity)	Important when identifying all positive cases is critical (e.g., disease diagnosis).
4. F1-score	Provides a balance between precision and recall.
5. Confusion Matrix	Provides a comprehensive overview of model performance, especially for multi-class classification.
6. ROC Curve	Helps visualize the trade-off between sensitivity and specificity.
7. AUC	A common metric for comparing the performance of different models.

Confusion Matrix

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

Actual	Predicted	Result	Confusion matrix
N	N	T	TN
N	P	F	FP
P	N	F	FN
P	P	T	TP

Confusion Matrix

$$\text{Accuracy} = \frac{TN + TP}{Total} \qquad Error\ rate = \frac{FP + FN}{Total}$$

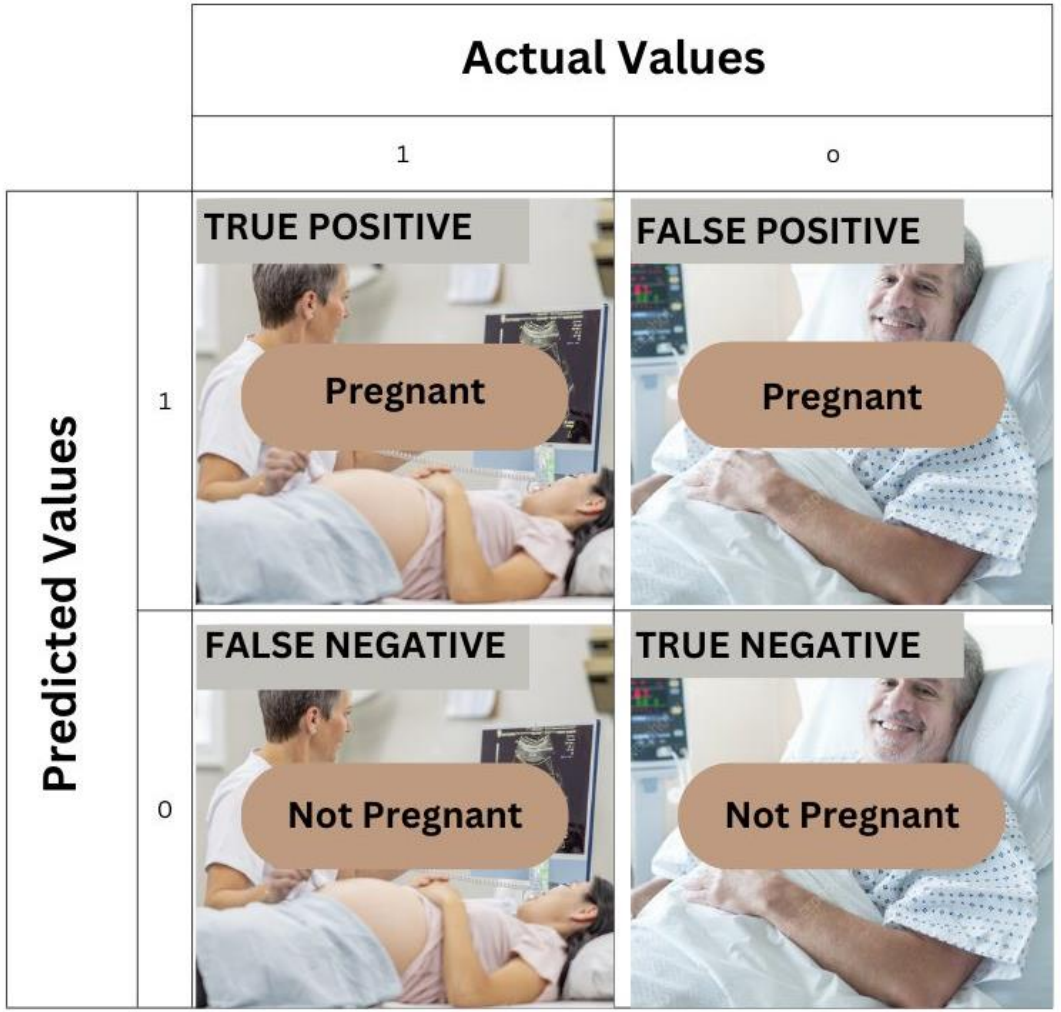
$$\text{Precision} = \frac{TP}{FP + TP} \qquad \text{Recall} = \frac{TP}{FN + TP}$$

$$\text{F1 Score} = \frac{2(Precision * Recall)}{Precision + Recall}$$

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

```
Accuracy, Precision, Recall and F1 => Correct Prediction
Error rate => Wrong Prediction
```

Confusion Matrix



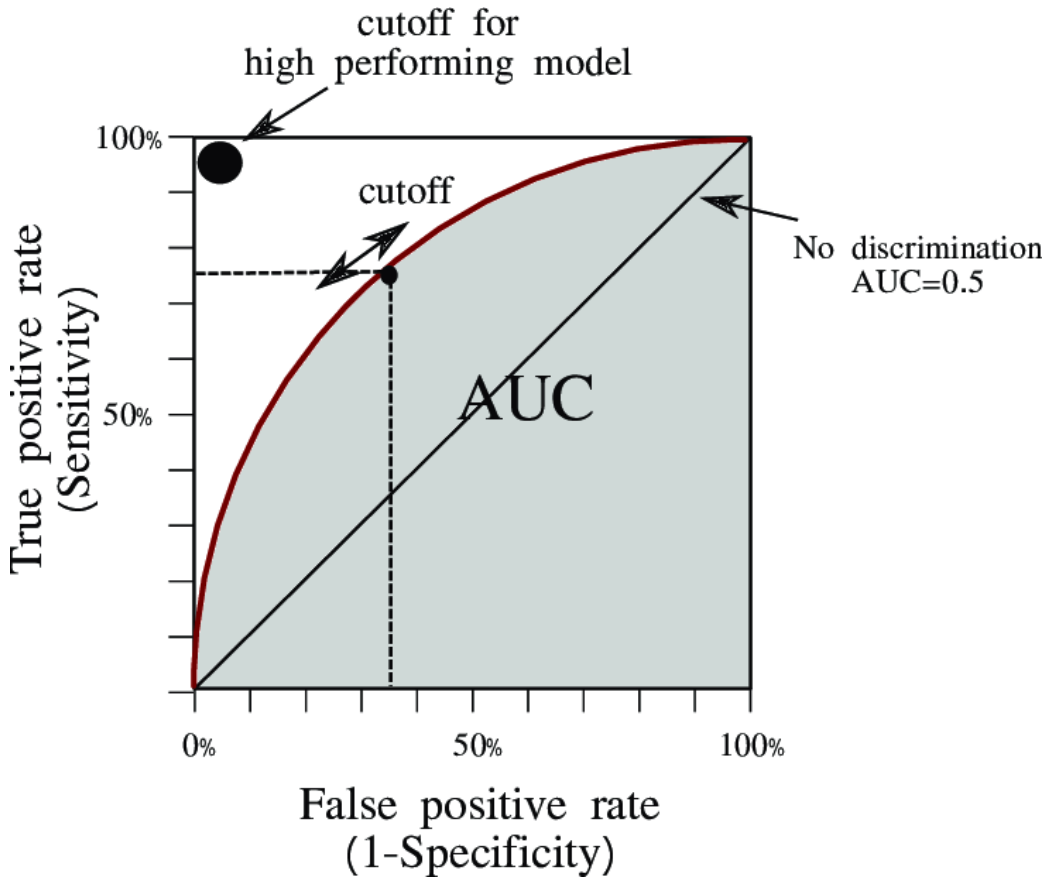
ROC and AUC curve

ROC (Receiver Operating Characteristic) Curve

- A graphical plot illustrating the diagnostic ability of a binary classifier system as its discrimination threshold is varied.
- X-axis: False Positive Rate (FPR) – The proportion of actual negatives that are incorrectly identified as positives.
- Y-axis: True Positive Rate (TPR) – The proportion of actual positives that are correctly identified as positives.

AUC (Area Under the Curve)

- A single metric representing the overall performance of a binary classification model based on the area under its ROC curve.
- Ranges from 0 to 1.
- A higher AUC generally indicates better performance.

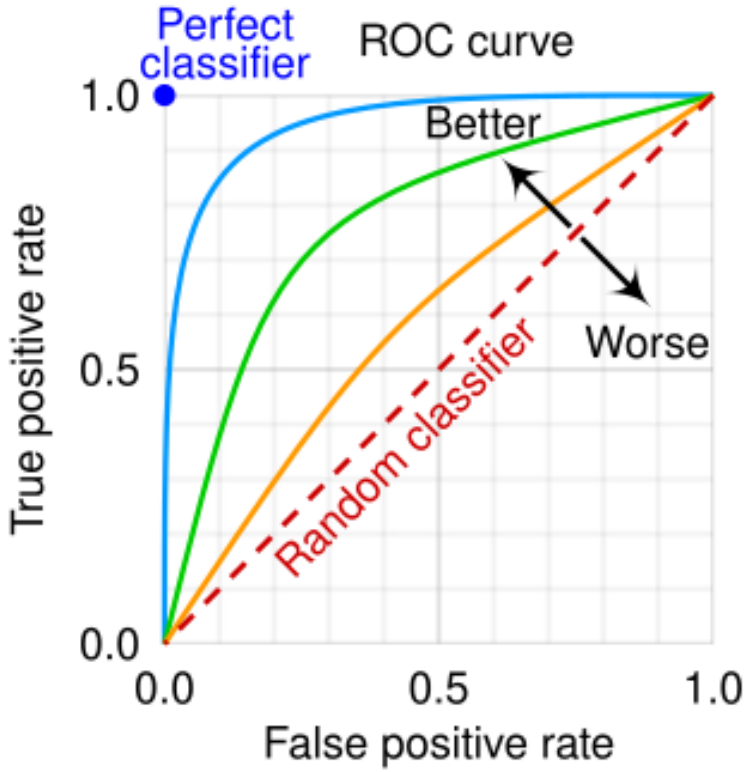


https://www.researchgate.net/figure/ROC-curves-and-area-under-curve-AUC_fig2_351506473

ROC and AUC curve

Interpretation

- Ideal Point (0,1) : A perfect classifier would have a TPR of 1 and an FPR of 0, meaning it correctly identifies all positives and no negatives.
- Random Guessing : A diagonal line from (0,0) to (1,1) – Represents a classifier that is no better than random guessing.
- Closer to the Top-Left Corner: A better classifier, as it has a higher TPR and lower FPR.



Example 4 : Predict customer purchase on online store

	gender	age	salary	purchased
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
...	...	...	...	...
396	Male	51	23000	1
397	Female	50	20000	1
398	Male	36	33000	0
399	Female	49	36000	1

400 rows × 4 columns

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score, precision_score
from sklearn.metrics import recall_score, f1_score
df = pd.read_excel('logis_customer.xlsx')
with pd.option_context('display.max_rows', 8): display(df)

# Encoding string to number
encoder = LabelEncoder()
df['gender'] = encoder.fit_transform(df['gender'])
features = ['gender', 'age', 'salary']
x = df[features]
y = df['purchased']
print("Data after encoded")
print(x)
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0)

# Scaling after splitting
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
model = LogisticRegression()

# Training Model
model.fit(x_train, y_train)
```

```
Data after encoded
     gender  age  salary
0         1   19   19000
1         1   35   20000
2         0   26   43000
3         0   27   57000
4         1   19   76000
..      ...  ...     ...
395       0   46   41000
396       1   51   23000
397       0   50   20000
398       1   36   33000
399       0   49   36000

[400 rows x 3 columns]
```

Evaluate Model

```
# Model Evaluation
y_pred_test = model.predict(x_test)
print('Confusion Matrix:')
cfmx = confusion_matrix(y_test, y_pred_test)
print(cfmx)
print()
print('Accuracy:', '{:.2f}'.format(accuracy_score(y_test, y_pred_test)))
print('Precision:', '{:.2f}'.format(precision_score(y_test, y_pred_test)))
print('Recall:', '{:.2f}'.format(recall_score(y_test, y_pred_test)))
print('F1 Score:', '{:.2f}'.format(f1_score(y_test, y_pred_test)))

#error_rate = (FP + FN) / Total
err = (cfmx[0, 1] + cfmx[1, 0]) / y_test.count()
print('Error Rate', '{:.2f}'.format(err))
print()
print(classification_report(y_test, y_pred_test))
```

```
Confusion Matrix:
[[65  3]
 [ 7 25]]

Accuracy: 0.90
Precision: 0.89
Recall: 0.78
F1 Score: 0.83
Error Rate 0.10
...
```

accuracy			0.90	100
macro avg	0.90	0.87	0.88	100
weighted avg	0.90	0.90	0.90	100

Deploy model with new data

```
gender = 'Male'
gender_enc = encoder.transform([gender])[0]  # Encode and extract the encoded value
age = 45
salary = 45000

# Prepare input data as a 2D array
x_pred = [[gender_enc, age, salary]]

# Scale the input data
x_pred_sc = scaler.transform(x_pred)

# Make prediction
y_pred = model.predict(x_pred_sc)

# 0 = Not Purchase, 1 = Purchase
print('Prediction:')
p = 'Not Purchase' if y_pred[0] == 0 else 'Purchase'
print(f'gender: {gender}, age: {age}, salary: {salary} ==>', p)
```

```
Prediction:
gender: Male, age: 45, salary: 45000 ==> Not Purchase
```

Change threshold value

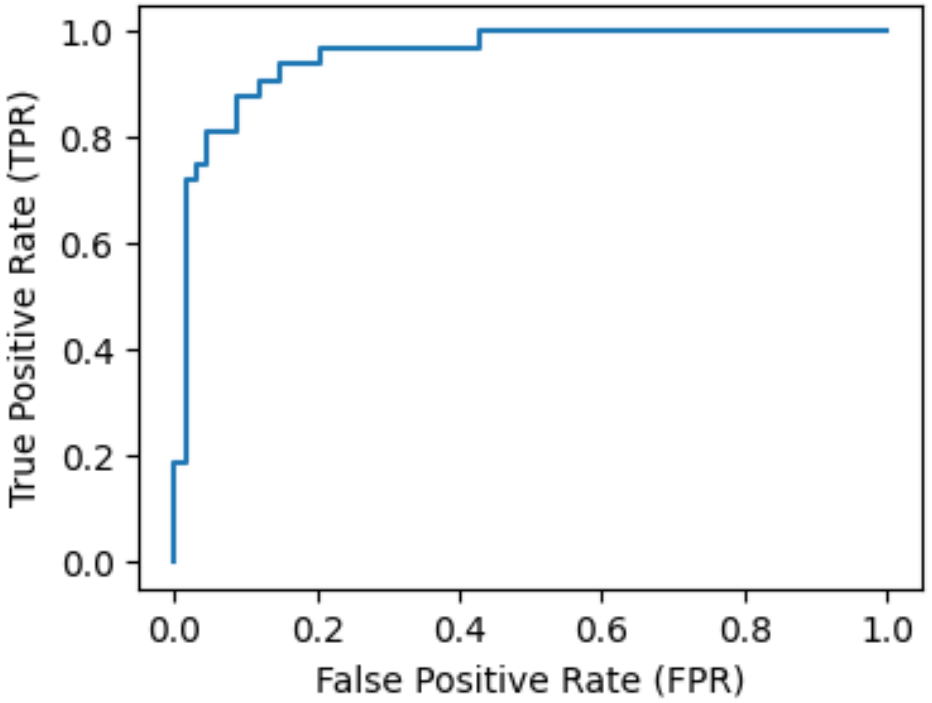
```
y_pred_th_0_5 = model.predict(x_test)
print('Confusion Matrix Threshold = 0.5')
print(confusion_matrix(y_test, y_pred_th_0_5))
print()

proba = model.predict_proba(x_test)
y_bin_th_0_25 = binarize(proba, threshold=0.25)
y_pred_th_0_25 = y_bin_th_0_25[:, 1]
print('Confusion Matrix Threshold = 0.25')
print(confusion_matrix(y_test, y_pred_th_0_25))
```

```
Confusion Matrix Threshold = 0.5
[[65  3]
 [ 7 25]]

Confusion Matrix Threshold = 0.25
[[53 15]
 [ 1 31]]
```

Calculate AUC Score

Solution 1:

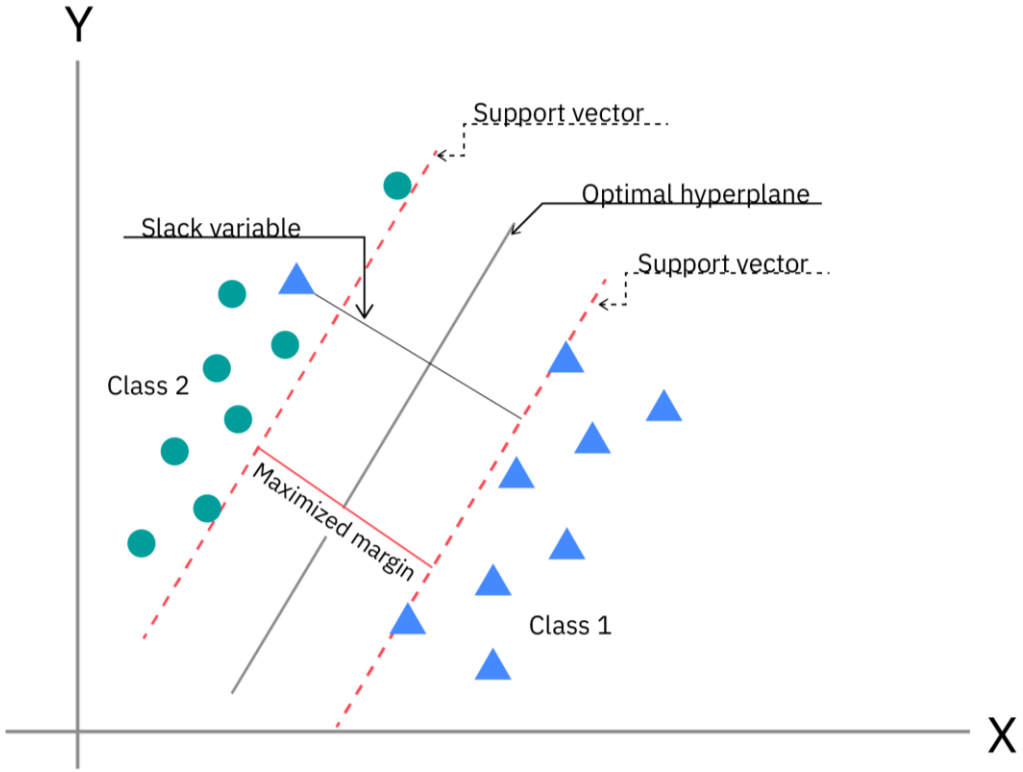
```
from sklearn.metrics import roc_curve, roc_auc_score, auc

probs = model.predict_proba(x_test)
probs = probs[:, 1]

fpr, tpr, threshold = roc_curve(y_test, probs)

plt.figure(figsize=(4, 3))
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.show()

auc_score = auc(fpr, tpr)
print('AUC Score:',
      '{:.2f}'.format(auc_score))
```



Solution 2:

```
# Extract probabilities for the positive class
probs = model.predict_proba(x_test)[:, 1]
auc_score = roc_auc_score(y_test, probs)
print('AUC Score:', f'{auc_score:.2f}')
```

Support Vector Machine (SVM)



SVMs were developed in the 1990s by Vladimir N. Vapnik and his colleagues, and they published this work in a paper titled "Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing"[1] in 1995.

https://www.ibm.com/think/topics/support-vector-machine#:~:text=A%20support%20vector%20machine%20(SVM,or%20hyperplane%20that%20maximizes%20the

Support Vector Machine (SVM)

Optimize separating hyperplane

$wx + b = 0$ where *w* is the weight vector, *x* is the input vector, and *b* is the bias term.

There are two approaches to calculating the margin, or the maximum distance between classes, which are hard-margin classification and soft-margin classification.

- For a hard-margin SVMs, the data points will be perfectly separated outside of the support vectors, or "off the street".

$$(wx_j + b)y_j \geq a$$

and then the margin is maximized

$$\max \gamma = \frac{a}{\|w\|}$$

Where a is the margin projected onto *w*

Support Vector Machine (SVM)

- For a soft-margin classification is more flexible, allowing for some misclassification through the use of slack variables (‘ξ‘) or hyperparameter C.

  a larger C value narrows the margin for minimal misclassification

  a smaller C value widens it, allowing for more misclassified data

(REF : Introduction to Support Vector Machines, Boswell, Dustin, Caltech, 2002: https://home.work.caltech.edu/~boswell/IntroToSVM.pdf)

Support Vector Machine (SVM)

SVM is a powerful supervised machine learning algorithm primarily used for multi-classes classification and regression tasks.

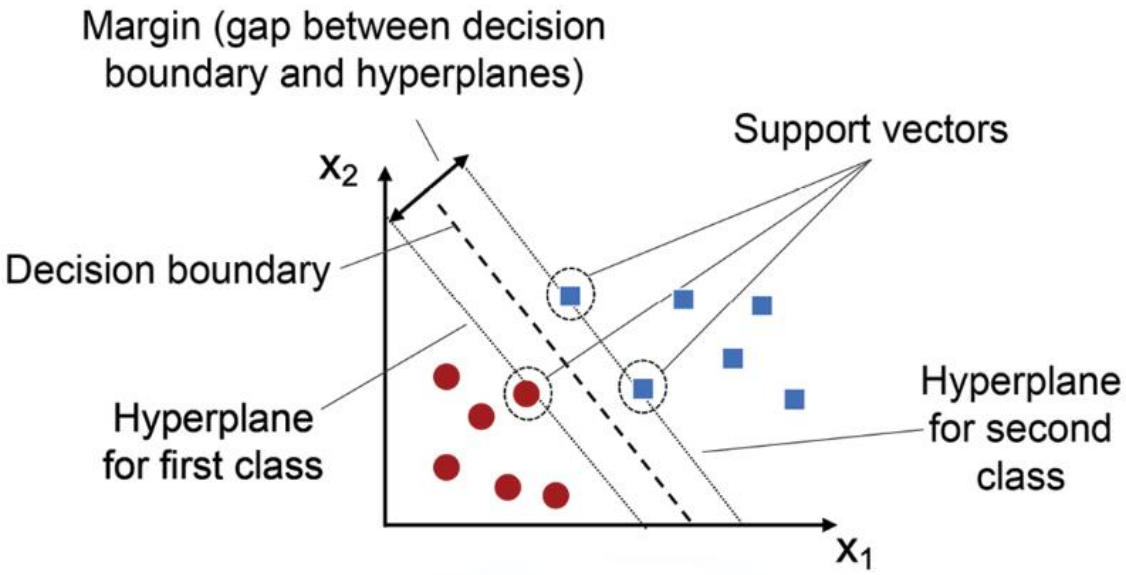
Concept

(1) Finding the Optimal Hyperplane

SVMs aim to find the best possible hyperplane that separates data points of different classes. In 2D, this is a line; in higher dimensions, it's a plane or a more complex surface.

(2) Maximizing the Margin

The key idea is to find the hyperplane that maximizes the distance (margin) to the nearest data points of each class. This maximizes the separation between the classes, making the model more robust to unseen data.



https://medium.com/@jainvidip/understanding-support-vector-machines-svms-1f7c78bad934

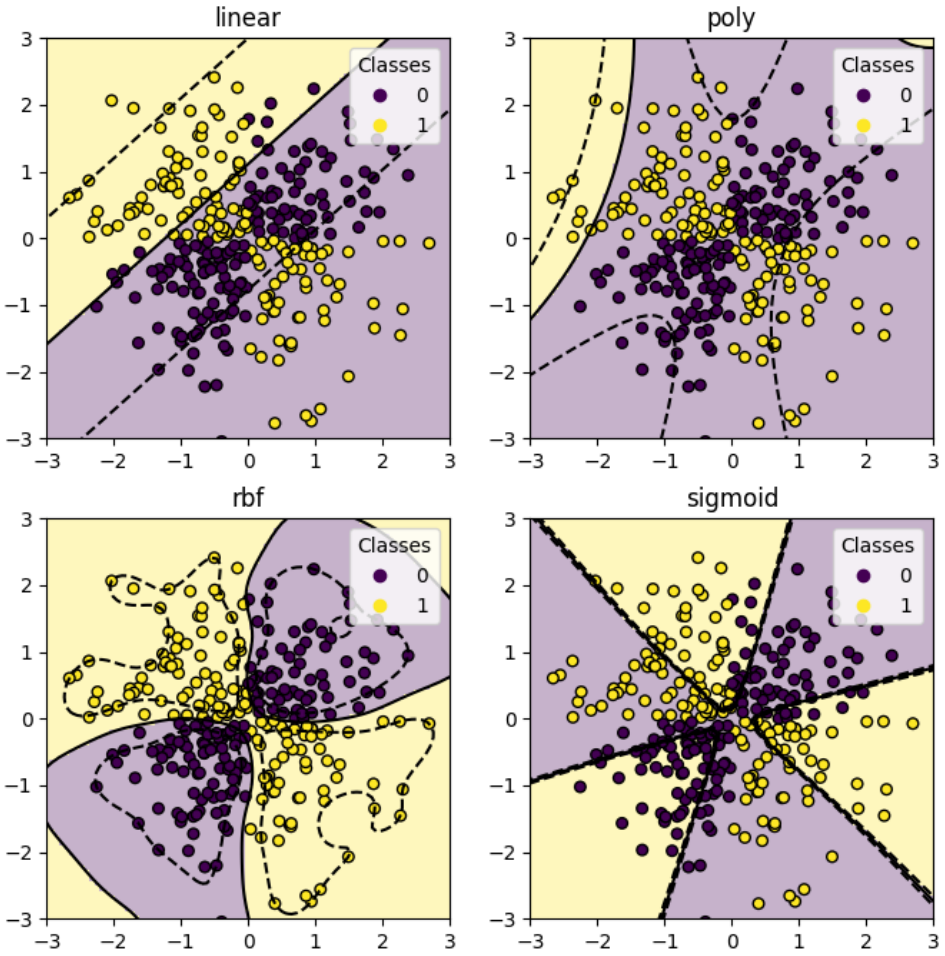
Support Vector Machine (SVM)

```
from sklearn.svm import SVC          # SVC = Support Vector Classifier

# Linear Kernel
model = SVC(kernel='linear')

# RBF kernel
model = SVC(kernel='rbf', gamma = 'γ-value', C = 'ξ-value')

# Polynoomial Kernel
model = SVC(kernel='poly', degree = 'polynomial degree', C = 'ξ-value')
```

SVM Kernels

- Linear
- Radial Basis Function Kernel (RBF)
- Polynomial Kernel
- Sigmoid Kernel

etc.

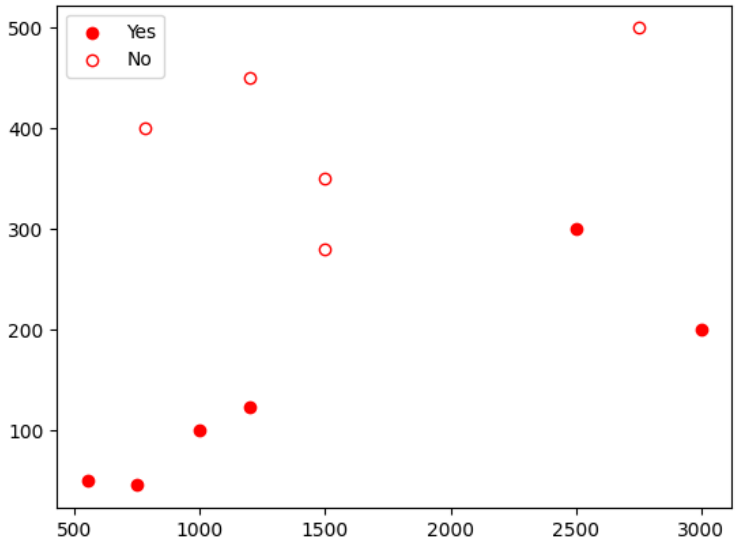


Hyperplane and boundary

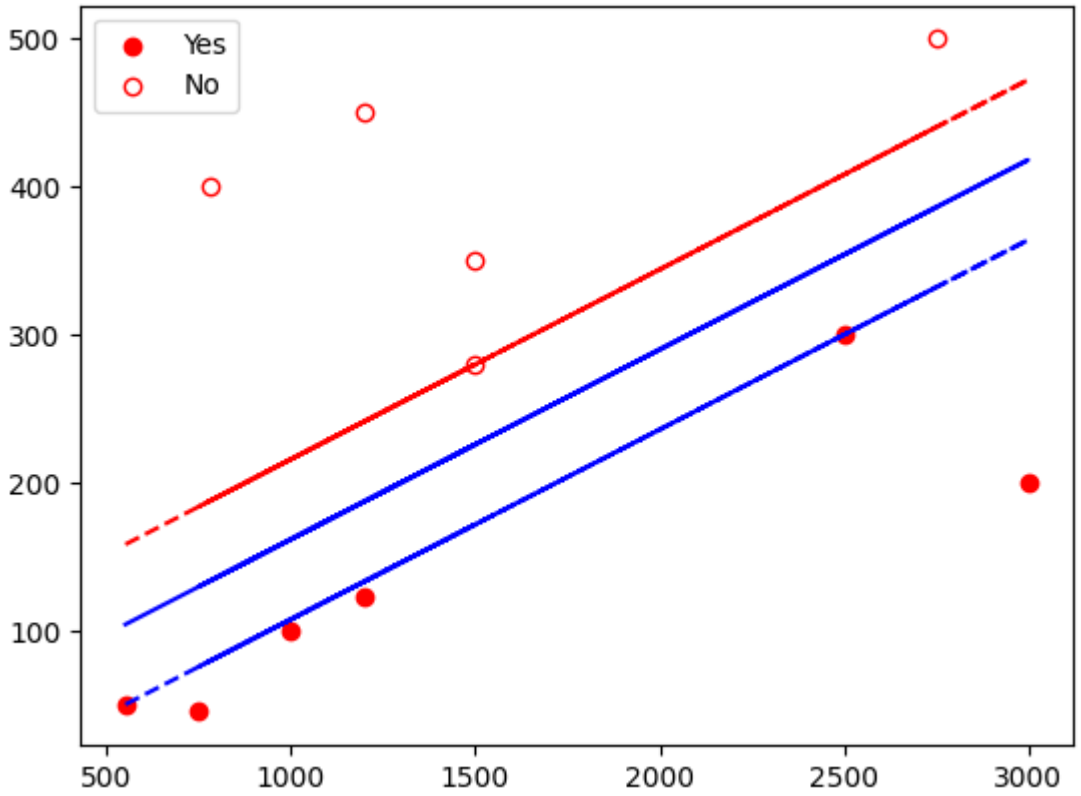
```
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import svm

df = pd.read_csv('svm_hyperplane.csv')
with pd.option_context('display.max_rows', 8): display(df)

# Part 1 : plot data
# There are 2 classes : Sold (Yes) and Unsold (No)
df_yes = df[df['sold'] == 'Yes']
x1 = df_yes['size']
x2 = df_yes['price']
plt.scatter(x1, x2, color='r', label='Yes')

df_no = df[df['sold'] == 'No']
x1 = df_no['size']
x2 = df_no['price']
plt.scatter(x1, x2, color='w', marker='o', edgecolors='r',
label='No')

plt.legend(loc='best’)
plt.show() # Comment this line before plot hyperplane
```

	size	price	sold
0	550	50	Yes
1	1000	100	Yes
2	1200	123	Yes
3	1500	350	No
...	...	...	...
7	1500	280	No
8	780	400	No
9	1200	450	No
10	2750	500	No

11 rows × 3 columns

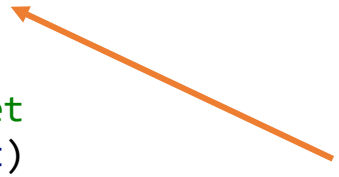


```
# Part 2 : plot hyperplane and boundary
df_yes = df[df['sold'] == 'Yes']
x1 = df_yes['size']
x2 = df_yes['price']
plt.scatter(x1, x2, color='r', label='Yes')
df_no = df[df['sold'] == 'No']
x1 = df_no['size']
x2 = df_no['price']
plt.scatter(x1, x2, color='w', marker='o', edgecolors='r', label='No')
plt.legend(loc='best')
# Train SVM model
x = df[['size', 'price']]
y = df['sold']
model = SVC(kernel='linear')
model.fit(x, y)
b = model.intercept_[0]
w1 = model.coef_[0, 0]
w2 = model.coef_[0, 1]
x1 = df['size']
# Plot Hyperplane : b + w1x1 + w2x2 = 0
x2 = (-b -w1 * x1) / w2
plt.plot(x1, x2, 'b-’)
# Plot Positive Boundary : b + w1x1 + w2x2 = 1
x2 = (1 -b -w1 * x1) / w2
plt.plot(x1, x2, 'b--’)
# Plot Negative Boundary : b + w1x1 + w2x2 = -1
x2 = (-1 -b -w1 * x1) / w2
plt.plot(x1, x2, 'r--')
plt.show()
```



RBF Kernel

```
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
# Load data from CSV file
data = pd.read_csv("svm_rbf.csv")
with pd.option_context('display.max_rows', 8): display(data)
# Separate features (X) and labels (y)
X = data.iloc[:, :-1].values  # Assuming the last column is the label
y = data.iloc[:, -1].values
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Create an RBF SVM classifier
svm_model = SVC(kernel='rbf', gamma=2.0 , C=2.0)
# Train the SVM model
svm_model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = svm_model.predict(X_test)
```



Adjust Gamma (γ) and C

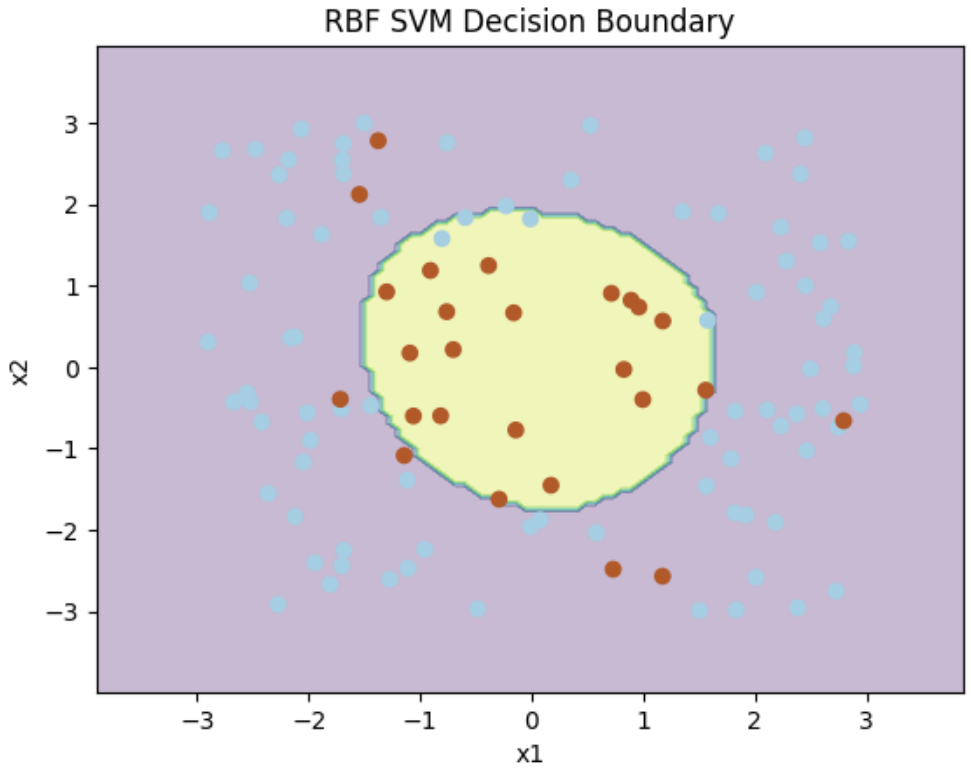
	x1	x2	y
0	-0.816684	-0.597198	1
1	0.524914	2.966416	0
2	0.890784	0.821454	1
3	-0.759416	2.753240	0
...	...	...	...
106	-0.703722	0.215382	1
107	0.729767	-2.479655	1
108	-1.715920	-0.393404	1
109	2.382873	-2.951074	0

110 rows × 3 columns

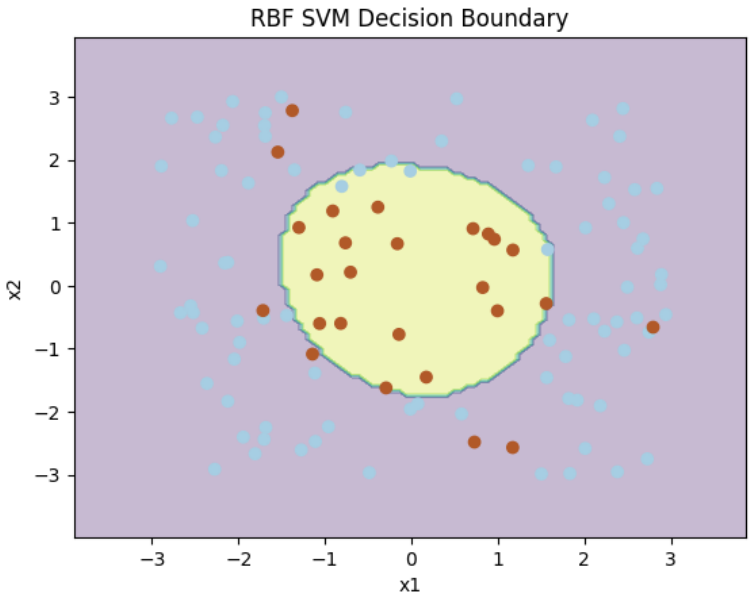
RBF Kernel

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
# Plot decision boundary (for 2D data)
if X.shape[1] == 2:
    # Create a meshgrid for plotting
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    h = (x_max - x_min) / 100
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    # Predict on the meshgrid
    Z = svm_model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    # Plot the decision boundary
    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.title("RBF SVM Decision Boundary")
    plt.show()
```

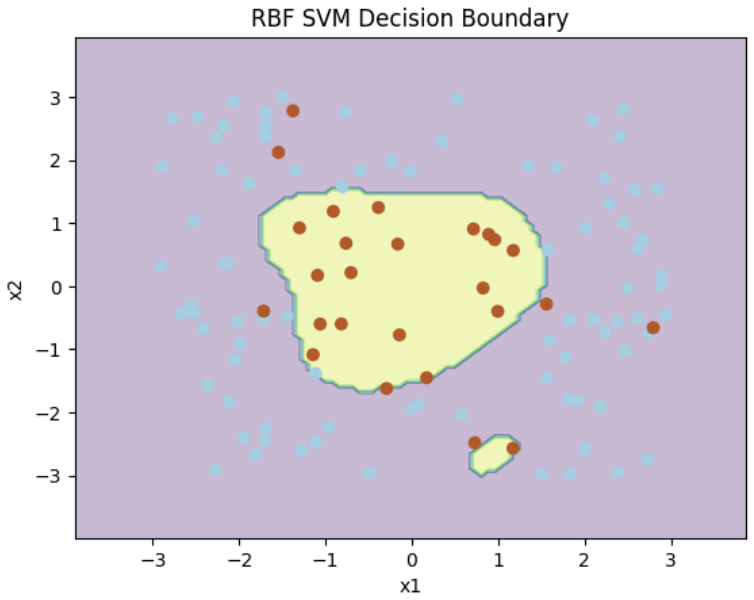
```
Accuracy: 0.9091
```



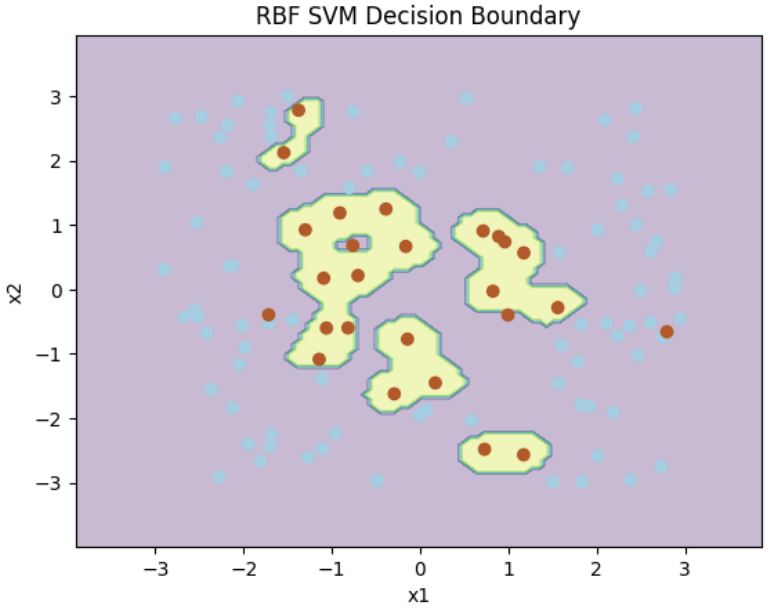
RBF Kernel (Adjust Gamma)



Gamma = 0.1
Accuracy: 0.9091

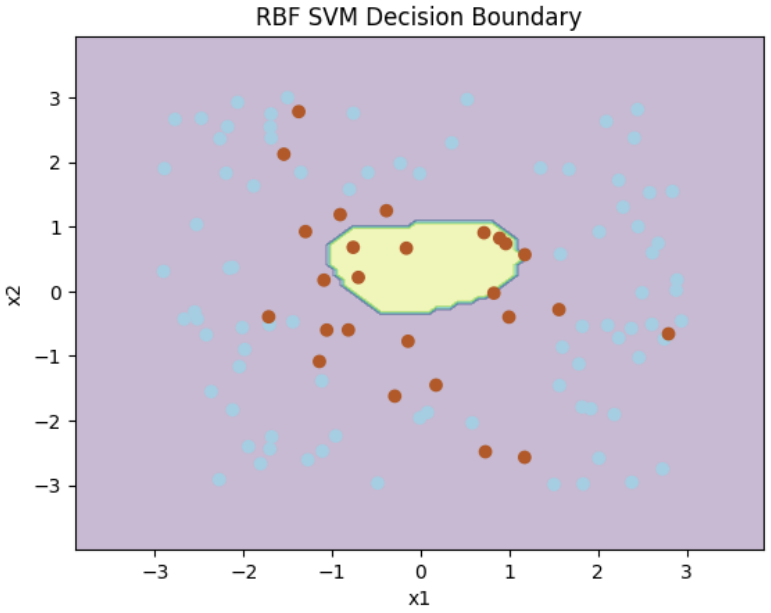


Gamma = 1.0
Accuracy: 0.9545

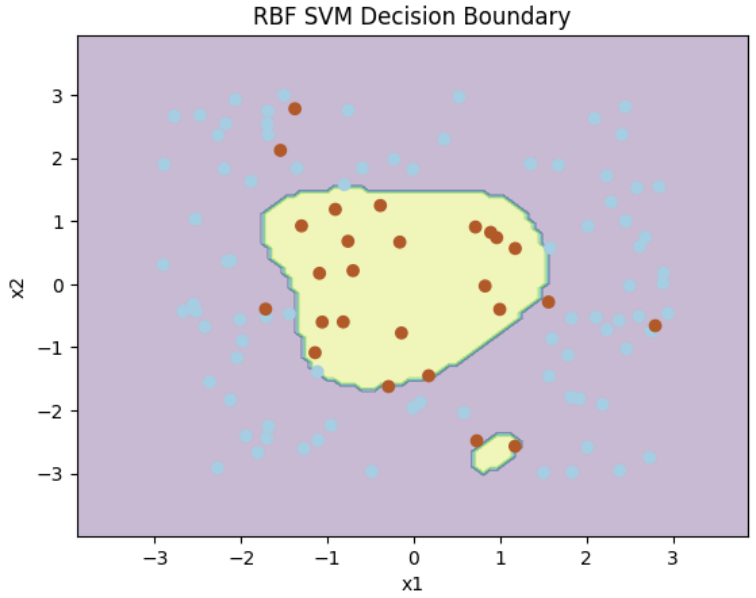


Gamma = 10.0
Accuracy: 0.8636

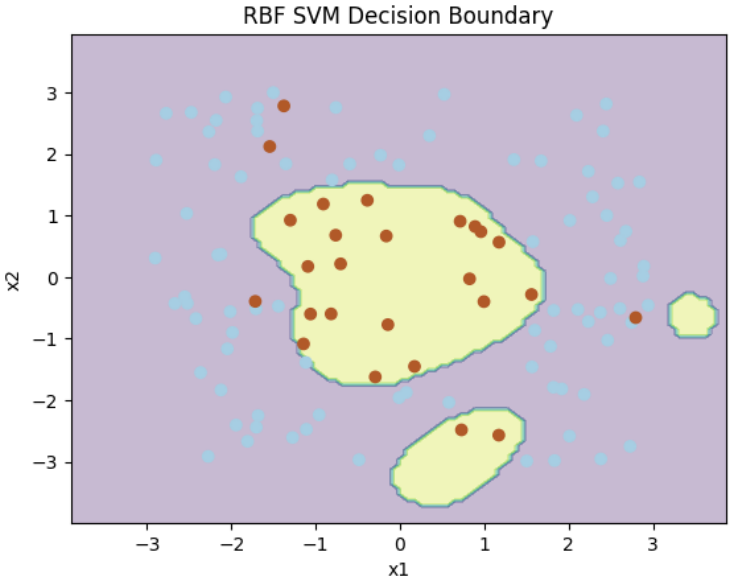
RBF Kernel (Adjust C)



Gamma = 1.0 , C = 0.25
Accuracy: 0.8636



Gamma = 1.0, C = 2.0
Accuracy: 0.9545



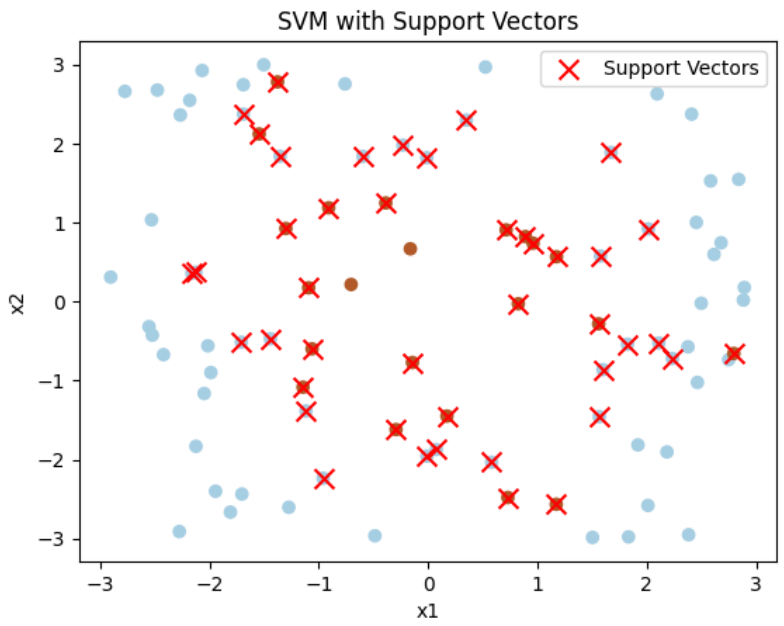
Gamma = 1.0, C = 10.0
Accuracy: 0.9545

RBF Kernel : Support Vector

```
# Get the indices of support vectors
support_vector_indices = svm_model.support_

# Get the support vectors themselves
support_vectors = X_train[support_vector_indices]

# Visualize support vectors (for 2D data)
if X.shape[1] == 2:
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=plt.cm.Paired)
    plt.scatter(support_vectors[:, 0], support_vectors[:, 1], color='red',
                marker='x', s=100, label='Support Vectors')
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.title("SVM with Support Vectors")
    plt.legend()
    plt.show()
print('Support Vectors:',svm_model.n_support_)
```

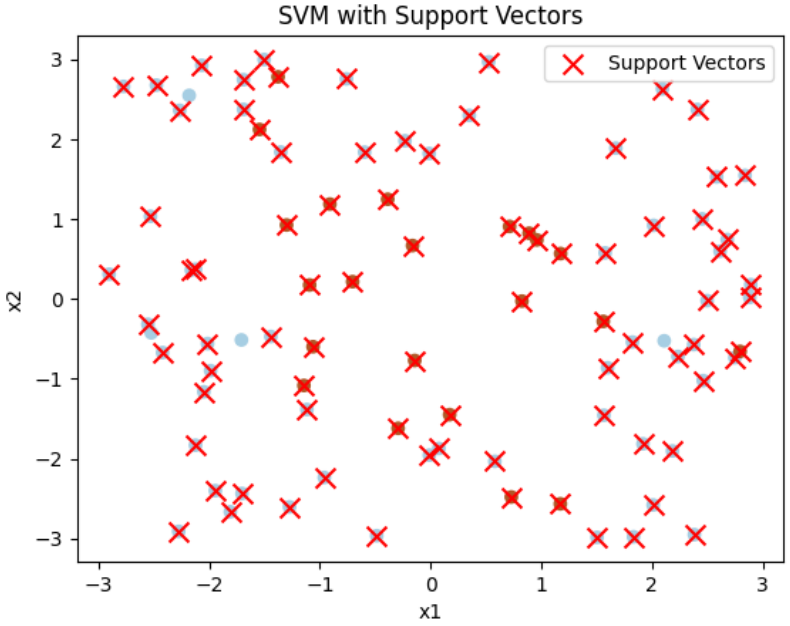

```
Support Vectors: [29 17]
```

RBF Kernel : Support Vector



Gamma = 0.1
Support Vectors: [23 20]

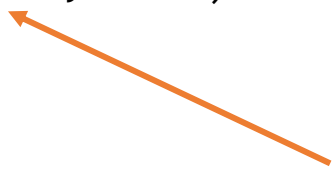
Gamma = 1.0
Support Vectors: [29 17]



Gamma = 10.0
Support Vectors: [62 22]

Polynomial Kernel

```
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
# Load data from CSV file
data = pd.read_csv("svm_poly.csv")
with pd.option_context('display.max_rows', 8): display(data)
# Separate features (X) and labels (y)
X = data.iloc[:, :-1].values  # Assuming the last column is the label
y = data.iloc[:, -1].values
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Create a Polynomial SVM classifier
svm_model = SVC(kernel='poly', degree=2, C=2.0)
# Train the SVM model
svm_model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = svm_model.predict(X_test)
```



Adjust degree and C

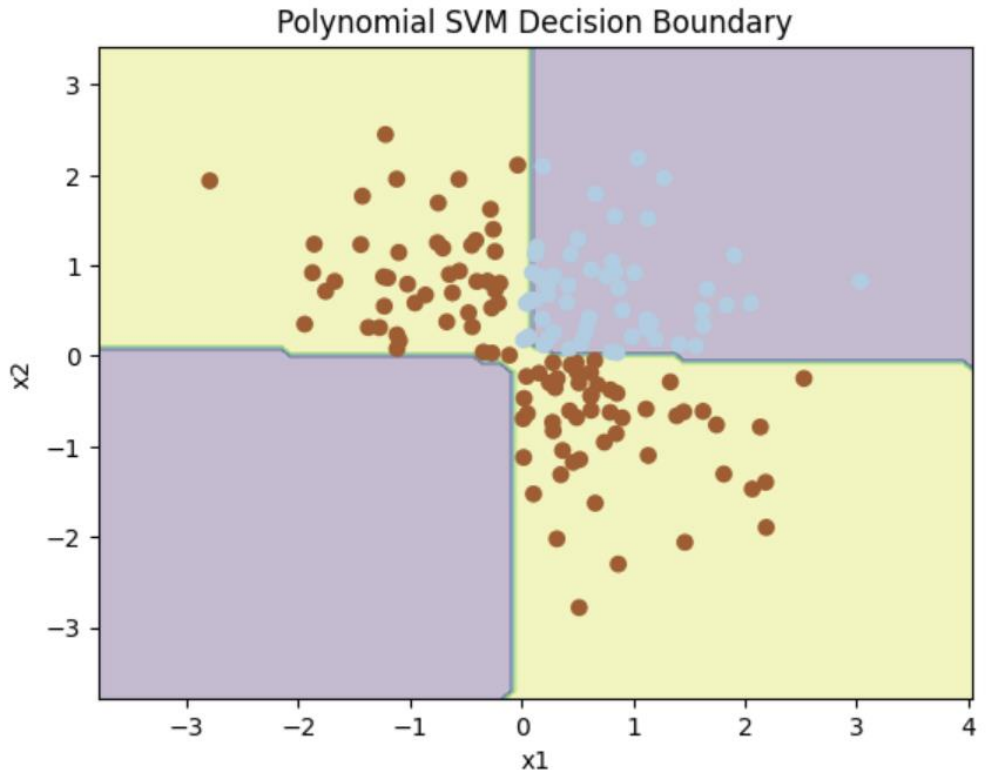
	x1	x2	y
0	0.042214	0.582815	-1
1	-1.100619	1.144724	1
2	0.901591	0.502494	-1
3	0.900856	-0.683728	1
...	...	...	...
150	-1.021886	0.794528	1
151	0.852704	0.035360	-1
152	-1.873161	0.920615	1
153	-0.035368	2.110605	1

154 rows × 3 columns

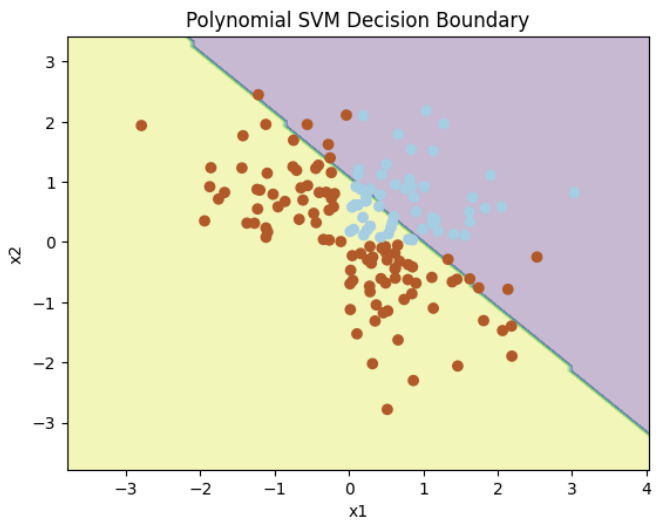
Polynomial Kernel

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Plot decision boundary (for 2D data)
if X.shape[1] == 2:
    # Create a meshgrid for plotting
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    h = (x_max - x_min) / 100
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    # Predict on the meshgrid
    Z = svm_model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    # Plot the decision boundary
    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.title("Polynomial SVM Decision Boundary")
    plt.show()
```

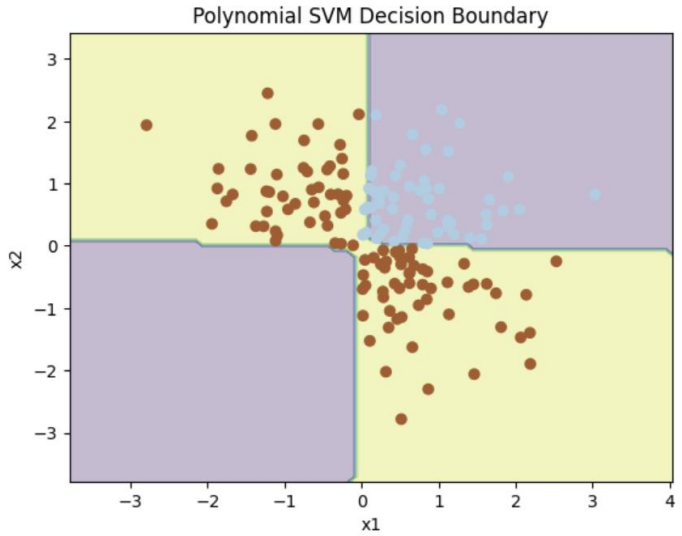
```
Accuracy: 1.0
```



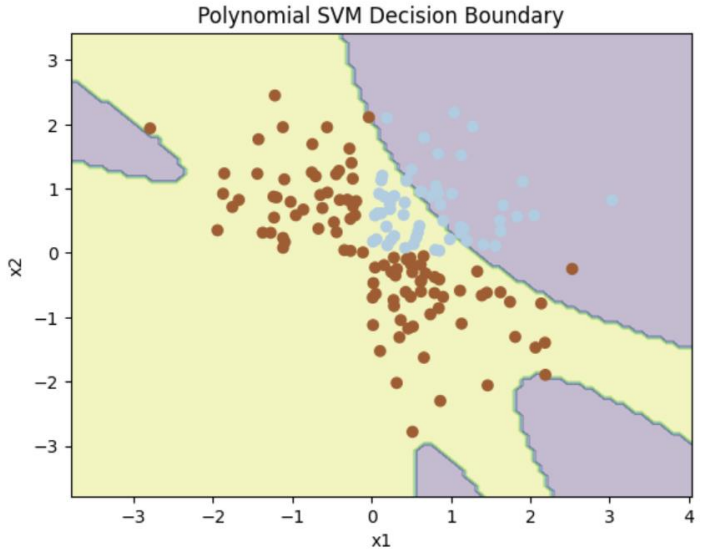
Polynomial Kernel



Degree = 1.0



Degree = 2.0



Degree = 5.0

K-Nearest Neighbors (KNN)

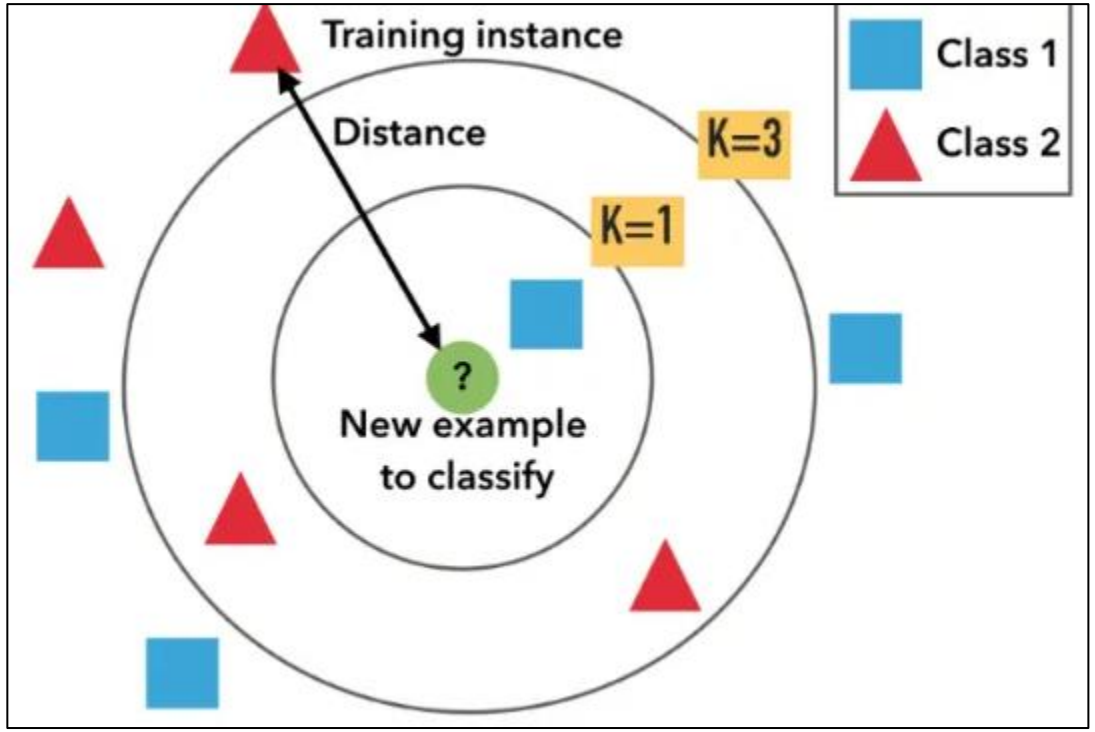
It is a simple yet powerful supervised machine learning algorithm used for both classification and regression tasks.

Class 1 = Class A (square)

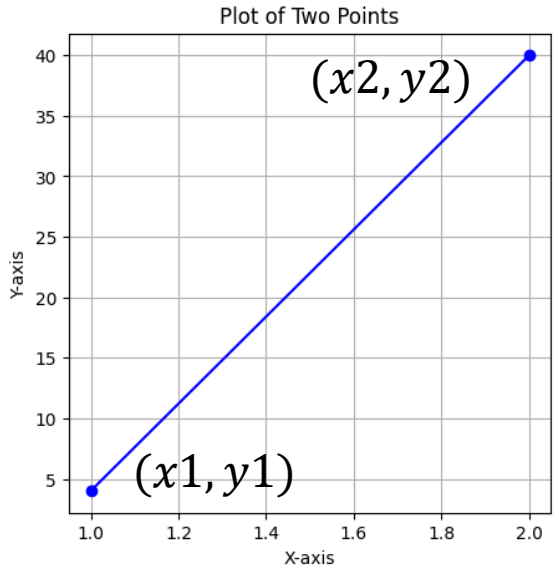
Class 2 = Class B (triangle)

1. Compare the new green circle point (New Point) to all points (Points) in the Training Set to find the closest points to the new point.
2. Determine the number of closest points to the new point, denoted as K.
3. If K=1, it means we consider only the closest point to the new point. If that closest point belongs to Class A, then the new point is also assigned to Class A.
4. If K=3, it means we consider the 3 closest points to the new point.

   For example, if there is 1 point from Class A and 2 points from Class B among these 3 closest points, we calculate the total distance (or similarity) of these 3 points. Since there are more points from Class B, the new point will be assigned to Class B.



Calculate Euclidean distance



$$distance = \sqrt{a^2 + b^2}$$
$$= \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

```
from sklearn.metrics.pairwise import euclidean_distances

a = [[1, 4]]
b = [[1, 12]]
c = [[2, 40]]

print('dist_ab =', euclidean_distances(a, b))
print('dist_bc =', euclidean_distances(b, c))
print('dist_ac =', euclidean_distances(a, c))
```

dist_ab = [[8.]]
dist_bc = [[28.01785145]]
dist_ac = [[36.01388621]]

K-Nearest Neighbors (KNN)

```
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor

model = KNeighborsClassifier(n_neighbors='k value')

model = KNeighborsRegressor(n_neighbors='k value')
```

Example 5 : Classify T-shirt size with KNN Classifier

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

df = pd.read_excel('knn_shirt_size.xlsx')
with pd.option_context('display.max_rows', 8): display(df)
x = df[['Height_cms', 'Weight_kgs']]
scaler = StandardScaler()
x = scaler.fit_transform(x)
y = df['T_Shirt_Size’]

k = 5
model = KNeighborsClassifier(n_neighbors=k)
model.fit(x, y)

x_pred = [[161, 61]]
x_pred_sc = scaler.transform(x_pred)
y_pred = model.predict(x_pred_sc)
h = x_pred[0][0]
w = x_pred[0][1]
s = y_pred[0]
print('K =', k)
print(f'Height: {h}, Weight: {w}, T-Shirt Size => {s}')
print('Accuracy:', '{:.2f}'.format(model.score(x, y)))
```

	Height_cms	Weight_kgs	T_Shirt_Size
0	158	58	M
1	158	59	M
2	160	64	L
3	163	64	L
...	...	...	...
14	170	63	L
15	163	61	M
16	170	64	L
17	170	68	L

18 rows × 3 columns

```
K = 5
Height: 161, Weight: 61, T-Shirt Size => M
Accuracy: 1.00
```

Example 6 : Predict weight from height and age with KNN Regressor

```
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler

df = pd.read_excel('knn_height_age_weight.xlsx')
with pd.option_context('display.max_rows', 10): display(df)
scaler = StandardScaler()
x = df[['height', 'age']]
x = scaler.fit_transform(x)
y = df['weight']

k = 3
model = KNeighborsRegressor(n_neighbors=k)
model.fit(x, y)

x_pred = [[170, 40]]
x_pred_sc = scaler.transform(x_pred)
y_pred = model.predict(x_pred_sc)
h = x_pred[0][0]
a = x_pred[0][1]
w = '{:.2f}'.format(y_pred[0])
print('K =', k)
print(f'Height: {h}, Age: {a} => Weight: {w}')
print('Accuracy:', '{:.2f}'.format(model.score(x, y)))
```

	height	age	weight
0	150.00	45	77
1	155.75	26	47
2	170.68	30	55
3	179.83	34	59
4	146.30	40	72
5	176.78	36	60
6	161.54	19	40
7	176.78	28	60
8	167.64	23	45
9	170.68	32	58

```
K = 3
Height: 170, Age: 40 => Weight: 59.00
Accuracy: 0.87
```

Example 7 : KNN for multiclass classification

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

df = pd.read_excel('knn_fruit_data.xlsx')
with pd.option_context('display.max_rows', 8): display(df)
x = df[['width', 'height', 'mass', 'color_score']]
y = df['fruit_name']
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0)

# scaling after splitting
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# Training model
k = 5
model = KNeighborsClassifier(n_neighbors=k)
model.fit(x_train, y_train)
```

	width	height	mass	color_score	fruit_name
0	8.4	7.3	192	0.55	apple
1	6.2	4.7	86	0.80	mandarin
2	6.0	4.6	84	0.79	mandarin
3	8.0	6.8	180	0.59	apple
...	...	...	...	...	...
55	6.1	8.1	118	0.70	lemon
56	7.6	8.2	180	0.79	orange
57	7.2	7.2	154	0.82	orange
58	7.2	10.3	194	0.70	lemon

59 rows × 5 columns

Example 7 : KNN for multiclass classification

```
# Make prediction

x_pred = [[7.5, 7.5, 175, 0.75]]
x_pred_sc = scaler.transform(x_pred)
y_pred = model.predict(x_pred_sc)
w = x_pred[0][0]
h = x_pred[0][1]
m = x_pred[0][2]
c = x_pred[0][3]
f = y_pred[0]

print('K =', k)
print('Prediction:')
print(f'width: {w}, height: {h}, mass: {m}, ', end='')
print(f'color score {c} => fruit: {f}')
print()
print('Accuracy:', '{:.2f}'.format(model.score(x_test, y_test)))
```

```
K = 5
Prediction:
width: 7.5, height: 7.5, mass: 175, color score 0.75 => fruit: orange

Accuracy: 0.93
```

Select optimal 'k' value

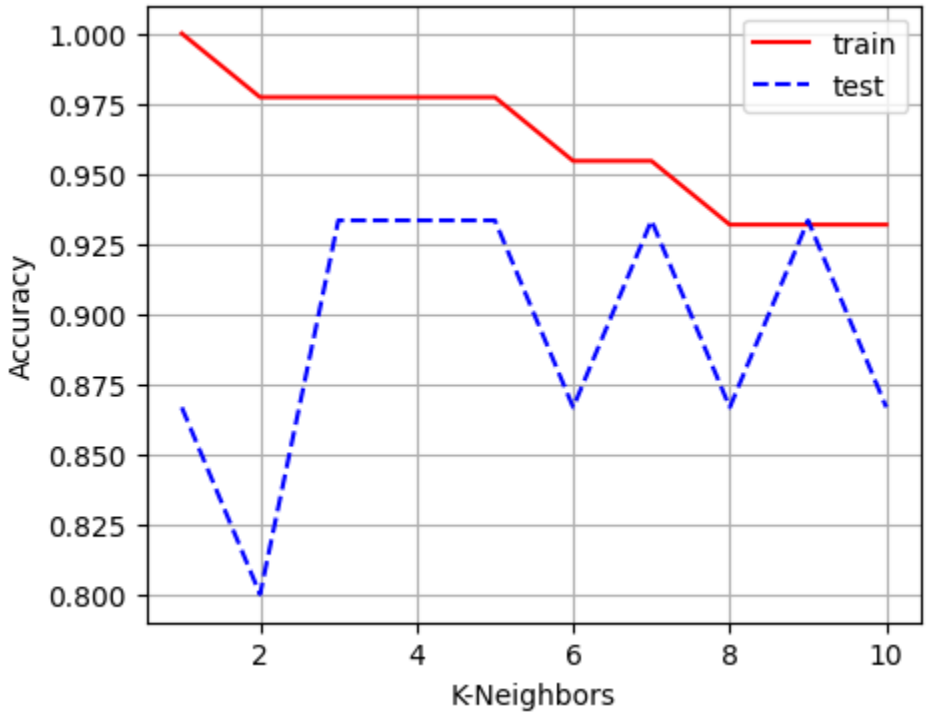
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

df = pd.read_excel('knn_fruit_data.xlsx')
with pd.option_context('display.max_rows', 6): display(df)

x = df[['width', 'height', 'mass', 'color_score']]
y = df['fruit_name']

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0)
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

	width	height	mass	color_score	fruit_name
0	8.4	7.3	192	0.55	apple
1	6.2	4.7	86	0.80	mandarin
2	6.0	4.6	84	0.79	mandarin
...	...	...	...	...	...
56	7.6	8.2	180	0.79	orange
57	7.2	7.2	154	0.82	orange
58	7.2	10.3	194	0.70	lemon

59 rows × 5 columns

Select optimal 'k' value

```
acc_train = []
acc_test = []
n_neighbors = range(1, 11)  #1 - 10

for k in n_neighbors:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(x_train, y_train)
    acc_train.append(model.score(x_train, y_train))
    acc_test.append(model.score(x_test, y_test))

plt.figure(figsize=(5, 4))
plt.plot(n_neighbors, acc_train, 'r-', label='train')
plt.plot(n_neighbors, acc_test, 'b--', label='test')
plt.xlabel('K-Neighbors')
plt.ylabel('Accuracy')
plt.grid()
plt.legend(loc='best')
plt.show()
```




```
Best Score: 0.7333333333333332
Best Params: {'n_neighbors': 1}
Optimal K: 1
```

Decision Tree

A decision tree is a flowchart-like structure used in machine learning to visualize and make decisions based on a series of rules. It resembles a tree with a root node, branches, and leaf nodes.

(1) Root Node

The starting point of the tree, representing the entire dataset.

(2) Internal Nodes

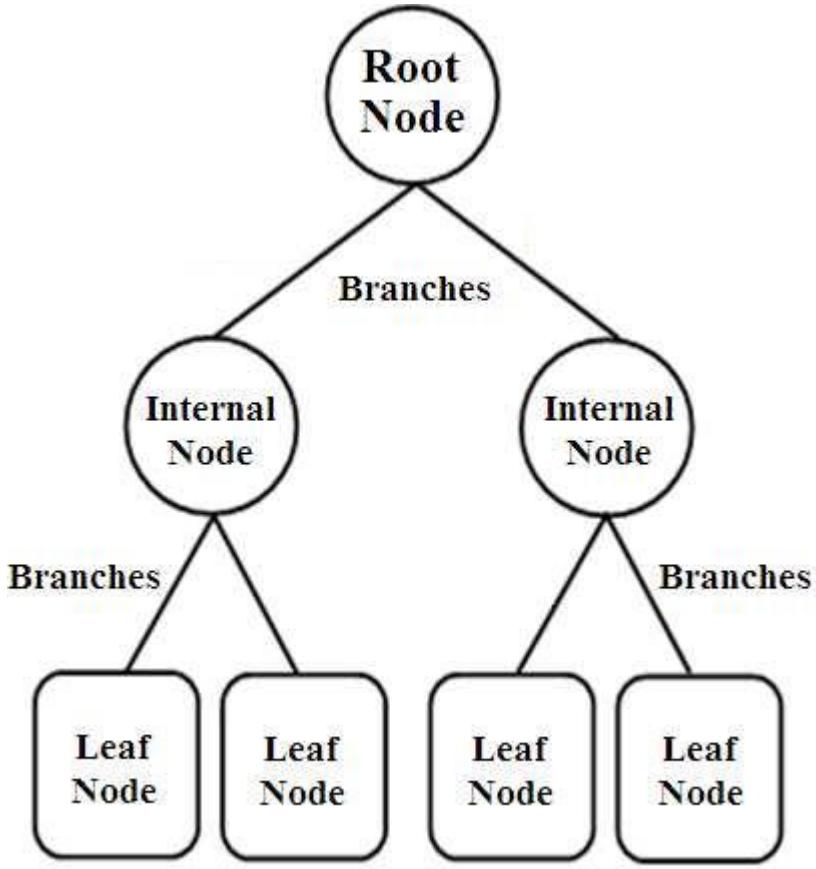
These nodes represent the features or attributes used to make decisions. They ask questions about the data and branch out based on the answers.

(3) Branches

These represent the possible outcomes of a decision at a particular node.

(4) Leaf Nodes

These are the final nodes of the tree, representing the predicted class or value for a given set of input conditions.



https://www.researchgate.net/figure/a-describes-the-components-of-a-decision-tree-the-Nodes-represent-the-possible_fig2_303773171?__cf_chl_tk=qjwTbtCixyLXUeT3x2VZb7pc0UWTqsIJNeYQx_8xidg-1737370383-1.0.1.1-keAprX5p6_Mofk8dGRleVu8t24ooXXakJ8lO7UPwOkM

Decision Tree

```
From sklearn.tree import DicisionTreeClassifier

Model = DecisionTreeClassifier(criterion = ‘entropy’)
Model = DecisionTreeClassifier(criterion = ‘gini’)
Model = DecisionTreeClassifier()                    # defult is CART (gini)
```

Decision Tree Algorithms

1. **ID3 (Iterative Dichotomiser 3) – entropy**

- Selects the attribute that provides the highest information gain at each node.
- Information gain measures how much information about the class label is gained by splitting the data based on that attribute.

2. **CART (Classification and Regression Trees) – gini**

Can be used for both classification and regression tasks.

- For classification, it uses the 'Gini' impurity as the splitting criterion.
- Gini impurity measures the probability of incorrectly classifying a randomly chosen sample from the dataset.
- For regression, it uses the mean squared error as the splitting criterion.

Training Algorithm	CART (Classification and Regression Trees)	ID3 (Iterative Dichotomiser 3)
Target(s)	Classification and Regression	Classification
Metric	Gini Index	Entropy function and Information Gain
Cost function (Based on what to split?)	Select its splits to achieve the subsets that minimize Gini Impurity	Yield the largest Information Gain for categorical targets

Example 8 : Classify animal species

```
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
import numpy as np

df = pd.read_excel('tree_animals.xlsx')
with pd.option_context('display.max_rows', 10): display(df)

# Columns' names below: 'Name', 'Blood_Temperature', 'Give_Birth', 'Can_Fly', 'Live_In_Water',
'Have_Legs', 'Species'
df.drop(columns=['Name'], inplace=True)
```

	Name	Blood_Temperature	Give_Birth	Can_Fly	Live_In_Water	Have_Legs	Species
0	Human	Warm	Yes	No	No	Yes	Mammals
1	Python	Cold	No	No	No	No	Reptiles
2	Bat	Warm	Yes	Yes	No	Yes	Mammals
3	Frog	Cold	No	No	Sometimes	Yes	Amphibians
4	Salmon	Cold	No	No	Yes	No	Fishes
...	...	...	...	...	...	...	...
15	Platypus	Warm	No	No	No	Yes	Mammals
16	Owl	Warm	No	Yes	No	Yes	Birds
17	Dolphin	Warm	Yes	No	Yes	No	Mammals
18	Eel	Cold	No	No	Yes	No	Fishes
19	Eagle	Warm	No	Yes	No	Yes	Birds

20 rows × 7 columns

```
encoders = []
for i in range(0, len(df.columns) - 1):
    enc = LabelEncoder()
    df.iloc[:, i] = enc.fit_transform(df.iloc[:, i])
    encoders.append(enc)
with pd.option_context('display.max_rows', 6): display(df)

x = df.iloc[:, 0:5]
y = df['Species']
model = DecisionTreeClassifier(criterion='entropy')  # using ‘entropy’ for ID3, using 'gini’ for CART method
model.fit(x, y)
```

	Blood_Temperature	Give_Birth	Can_Fly	Live_In_Water	Have_Legs	Species
0	1	1	0	0	1	Mammals
1	0	0	0	0	0	Reptiles
2	1	1	1	0	1	Mammals
...	...	...	...	...	...	...
17	1	1	0	2	0	Mammals
18	0	0	0	2	0	Fishes
19	1	0	1	0	1	Birds

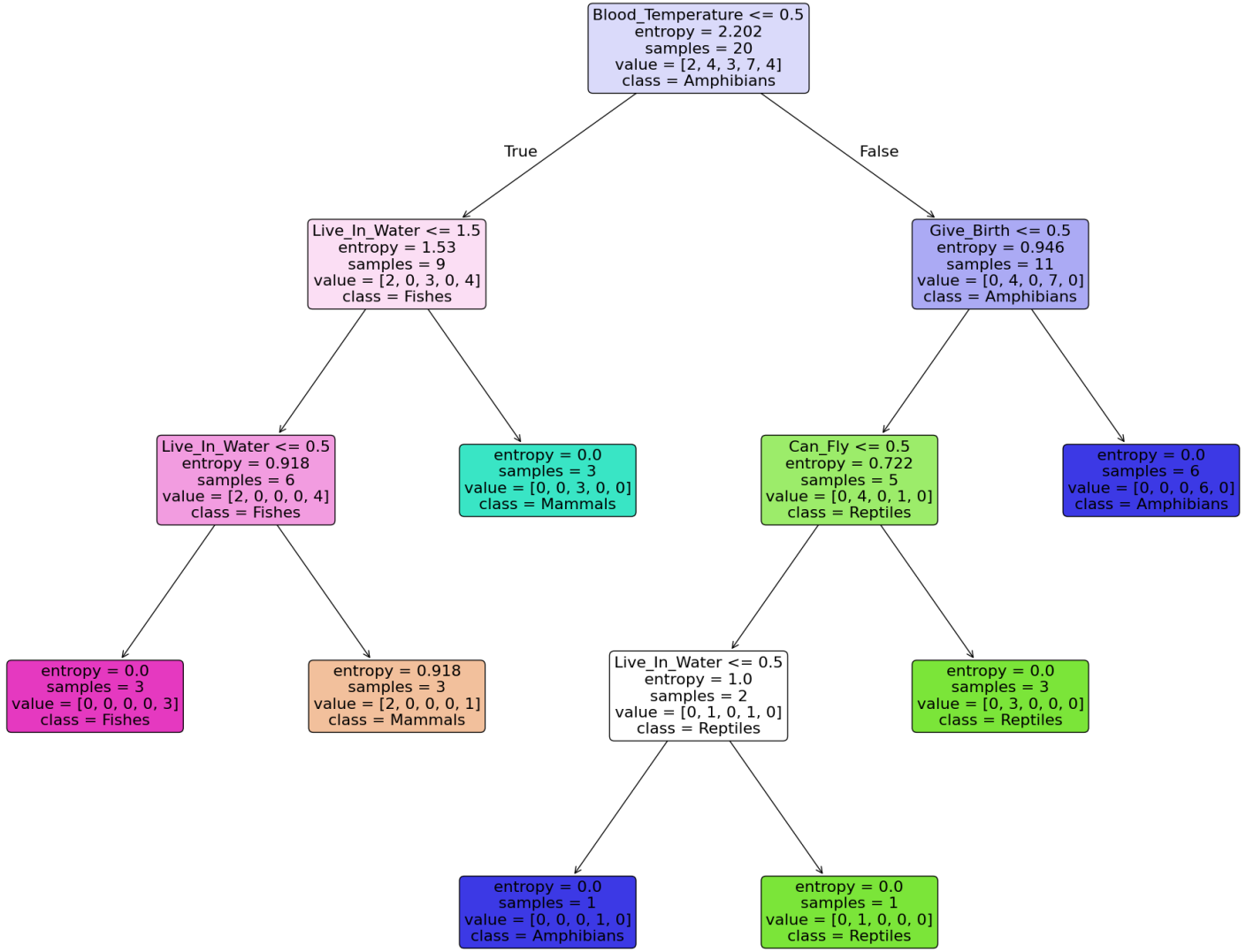
20 rows × 6 columns

```
# Make Prediction

x_pred = ['Warm', 'No', 'Yes', 'Sometimes', 'No’] # Blood_Tem, Give_Birth, Can_Fly, Live_In_Water, Have_Legs
for i in range(0, len(df.columns) - 1):
    x_pred[i] = encoders[i].transform([x_pred[i]])
x_pred_a = np.array(x_pred).reshape(-1, 5) # or len(df.columns)-1
y_pred = model.predict(x_pred_a)
print('Prediction:', y_pred[0])
score = model.score(x, y)
print('Accuracy:', '{:.2f}'.format(score))
```

```
Prediction: Birds
Accuracy: 0.95
```

Plot Tree

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(25, 20))
t = plot_tree(model,
              feature_names=x.columns,
              class_names=y,
              label='all',
              impurity=True,
              precision=3,
              filled=True,
              rounded=True,
              fontsize=16)

plt.show()
```



Example 9 : Forecast number of players form weather

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

df = pd.read_excel('tree_weather_players.xlsx')
# with pd.option_context('display.max_rows', 6): display(df)

encoders = []
for i in range(0, len(df.columns) - 1):
    enc = LabelEncoder()
    df.iloc[:, i] = enc.fit_transform(df.iloc[:, i])
    encoders.append(enc)
with pd.option_context('display.max_rows', 10): display(df)
```

	Outlook	Temp	Humidity	Wind	Players
0	2	1	0	1	25
1	2	1	0	0	30
2	0	1	0	1	46
3	1	2	0	1	45
4	1	0	1	1	52
...	...	...	...	...	...
9	1	2	1	1	46
10	2	2	1	0	48
11	0	2	0	0	52
12	0	1	1	1	44
13	1	2	0	0	30

14 rows × 5 columns

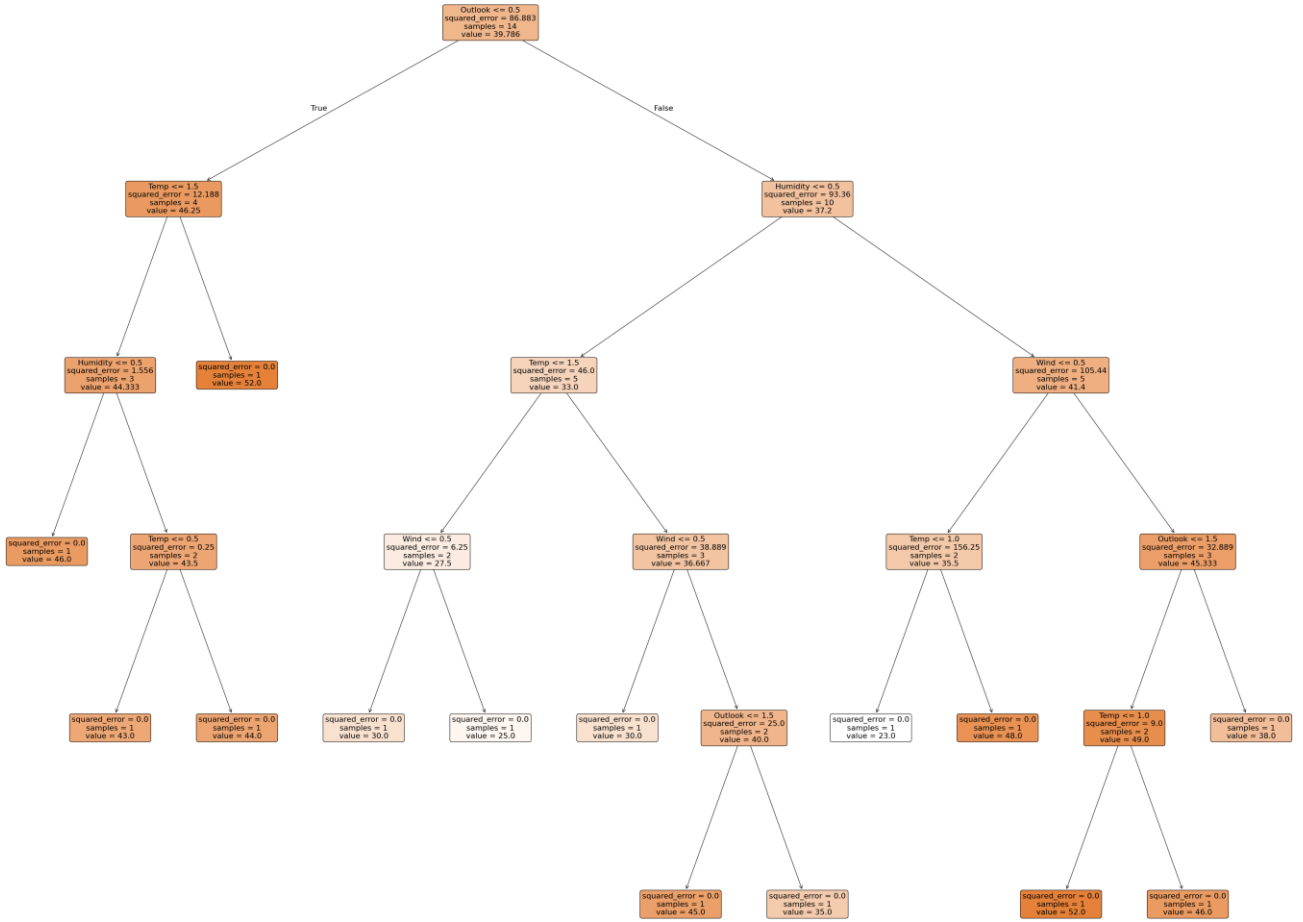
Example 9 : Forecast number of players form weather

```
x = df.iloc[:, 0:4]
y = df['Players']

model = DecisionTreeRegressor(criterion='squared_error')
model.fit(x, y)

x_pred = ['Overcast', 'Cool', 'High', 'Weak']

for i in range(0, len(df.columns) - 1):
    x_pred[i] = encoders[i].transform([x_pred[i]])
x_pred_a = np.array(x_pred).reshape(-1, 4) # or len(df.columns)-1
y_pred = model.predict(x_pred_a)
for i in range(0, len(df.columns) - 1):
    s = encoders[i].inverse_transform(x_pred[i])
    print(f'{df.columns[i]}: {s[0]}, ', end='')
print('Players:', '{:.0f}'.format(y_pred[0]))
score = model.score(x, y)
print('Accuracy:', '{:.2f}'.format(score))
```

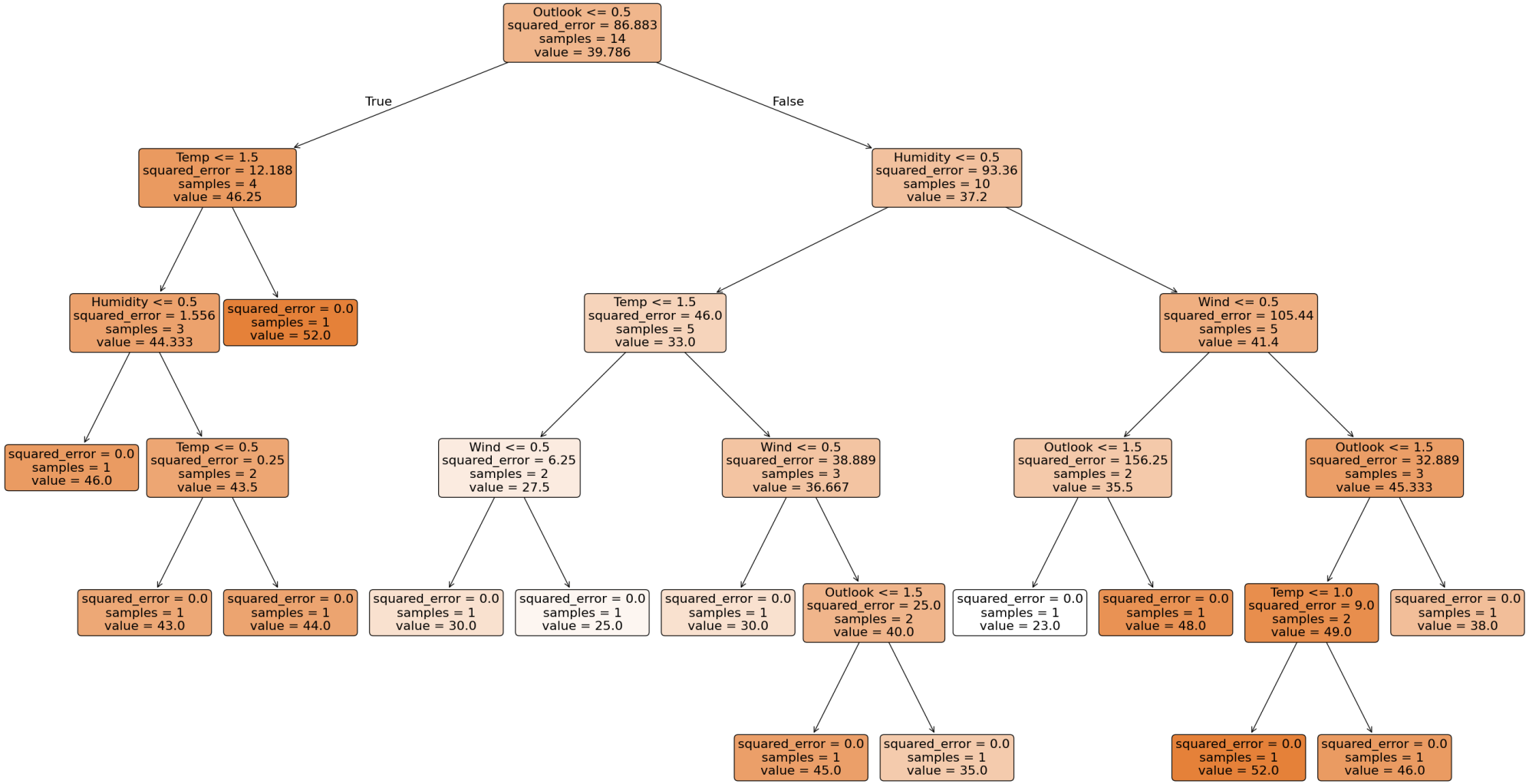
```
Outlook: Overcast, Temp: Cool, Humidity: High, Wind: Weak, Players: 46
Accuracy: 1.00
```

Example 9 : Forecast number of players form weather

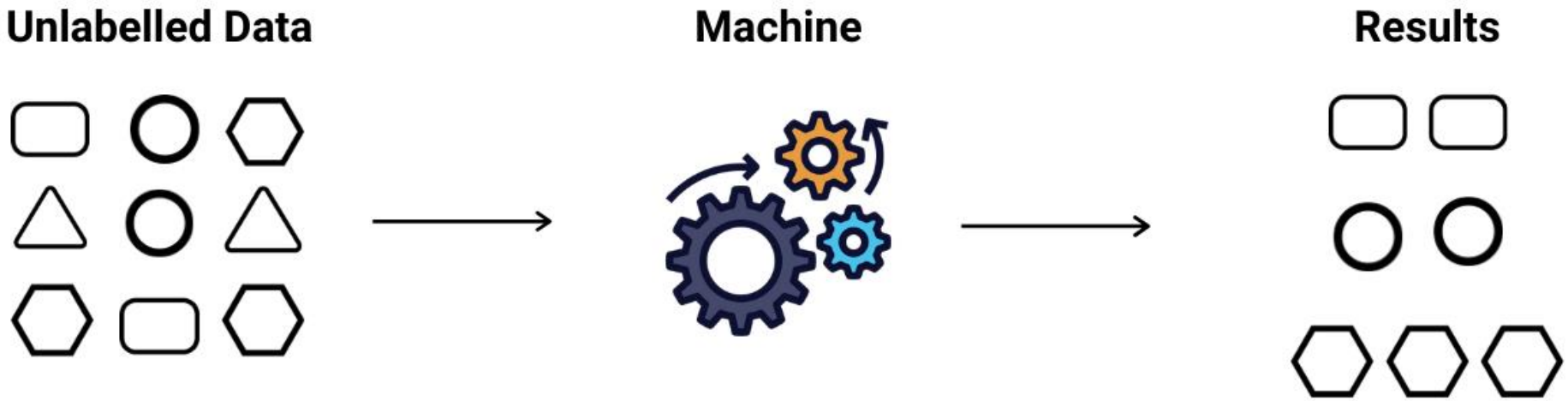
```
plt.figure(figsize=(50, 40)) # or (35, 20)
tree = plot_tree(model,
                 feature_names=x.columns,
                 class_names=y,
                 label='all',
                 impurity=True,
                 precision=3,
                 filled=True,
                 rounded=True,
                 fontsize=16)

plt.show()
```



Example 9 : Forecast number of players form weather

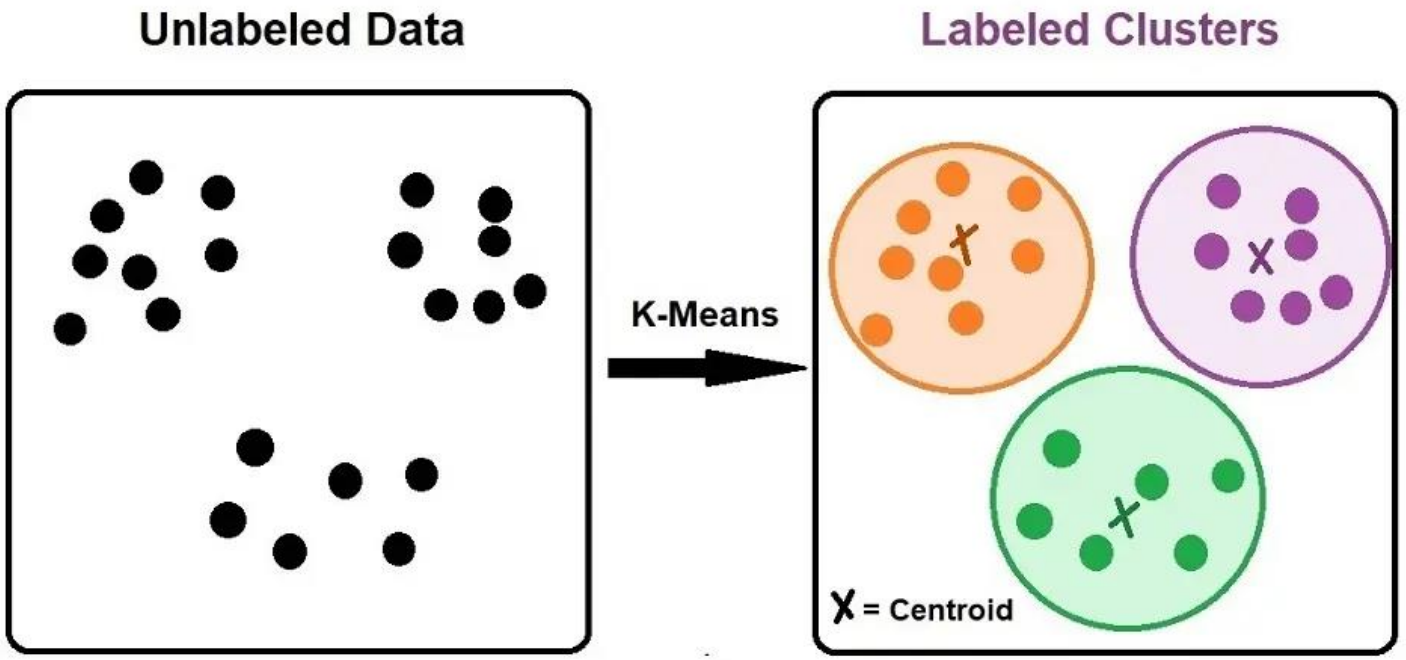


Unsupervised Learning



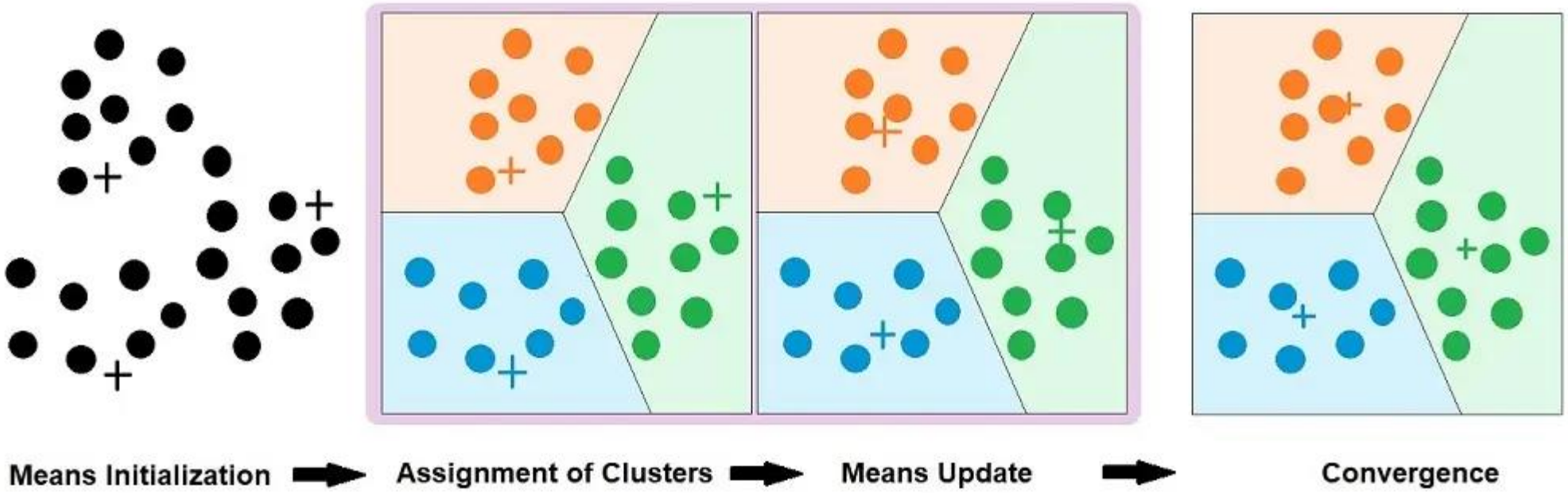
Unsupervised Learning : K-Means Clustering

It is a popular unsupervised machine learning algorithm used to partition a dataset into a pre-defined number of clusters (K).



https://www.ejable.com/tech-corner/ai-machine-learning-and-deep-learning/k-means-clustering/

K-Means Clustering



https://www.ejable.com/tech-corner/ai-machine-learning-and-deep-learning/k-means-clustering/

K-Means Clustering

```
from sklearn.cluster import Kmeans

model = KMeans(n_clusters= ‘Number of cluster’, random_state= ‘sets of random data’))
```

Clustering

```
from sklearn.cluster import KMeans
import pandas as pd

df = pd.read_csv('k-means.csv')
with pd.option_context('display.max_rows', 10): display(df)

model = KMeans(n_clusters=3, random_state=0) # N-Cluster = 3
model.fit(df)

print("=== Model labeled ==")
print(model.labels_,"\n")
print("=== Cluster centers ==")
print(model.cluster_centers_,"\n")
print("=== Prediction for point (30,40) ==")

# Make prediction
pd = model.predict([[30, 40]])
print(pd[0])
```

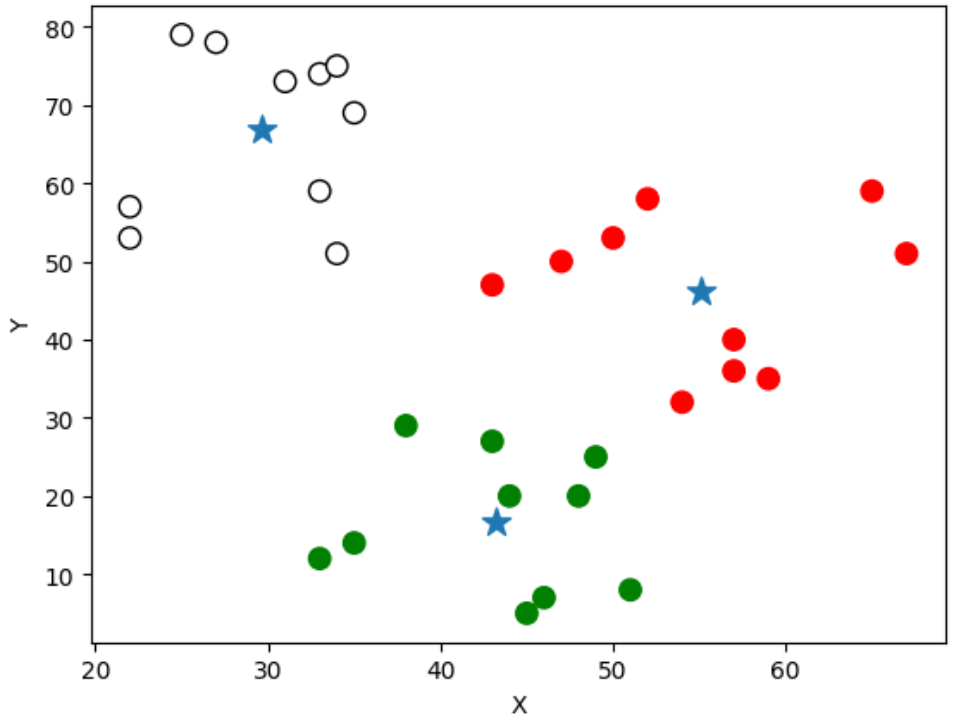
	x	y
0	25	79
1	34	51
2	22	53
3	27	78
4	33	59
...	...	...
25	45	5
26	38	29
27	43	27
28	51	8
29	46	7

30 rows × 2 columns

```
=== Model labeled ==
[1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2]

=== Cluster centers ==
[[55.1 46.1]
 [29.6 66.8]
 [43.2 16.7]]

=== Prediction for point (30,40) ==
0
```

Plot Data point and Cluster Centers

```
predict = model.predict(df)
c = ['red', 'white', 'green']
colors = []
edge_colors = []

for i in predict:
    colors.append(c[i])
    edge_colors.append('black' if c[i] == 'white' else c[i])

centroids = model.cluster_centers_

plt.scatter(df['x'], df['y'], c=colors,
            edgecolor=edge_colors, s=80)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=150)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



Example 10 : bmx data for body measurement

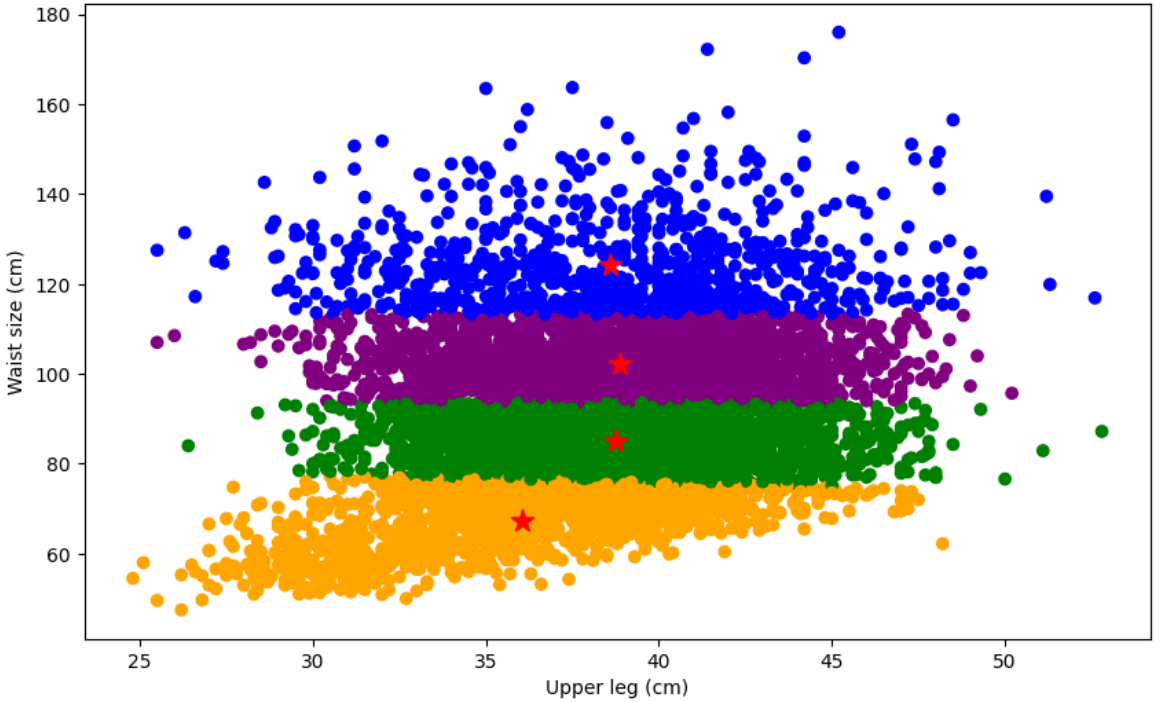
```
# data source : https://data.world/rhoyt/body-measurements
# bmxwaist = size of waist (cm), bmxleg = size of upper leg (cm)
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

df = pd.read_csv('kmeans_bmx.csv')
df = df[['bmxleg', 'bmxwaist']].dropna()
with pd.option_context('display.max_rows', 10): display(x)

model = KMeans(n_clusters=4)
model.fit(df)

centroids = model.cluster_centers_
print('Centroids:\n', centroids)
```

	width	height	mass	color_score
0	8.4	7.3	192	0.55
1	6.2	4.7	86	0.80
2	6.0	4.6	84	0.79
3	8.0	6.8	180	0.59
4	7.4	7.2	176	0.60
...	...	...	...	...
54	6.5	8.5	152	0.72
55	6.1	8.1	118	0.70
56	7.6	8.2	180	0.79
57	7.2	7.2	154	0.82
58	7.2	10.3	194	0.70

59 rows × 4 columns

```
Centroids:
 [[ 38.8833557  102.3190604 ]
 [ 38.73323392  85.11712825]
 [ 38.59412442 124.28997696]
 [ 36.04064872  67.30131125]]
```

```
clusters = model.predict(df)
cluster_colors = ['purple', 'green', 'blue', 'orange']
data_colors = []
for i in clusters:
    data_colors.append(cluster_colors[i])

plt.figure(figsize=(10, 6))
# plot data points
plt.scatter(df['bmxleg'], df['bmxwaist'], c=data_colors)
# plot centroid points for each cluster
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*',
            color='red', s=150)   #s = marker size
plt.xlabel("Upper leg (cm)")
plt.ylabel("Waist size (cm)")
plt.show()
```



Determine optimal 'K' using Silhouette Coefficient

A Silhouette Coefficient, or the Silhouette Score, is a metric to calculate the efficiency of a clustering technique, ranging from the values -1 to 1.

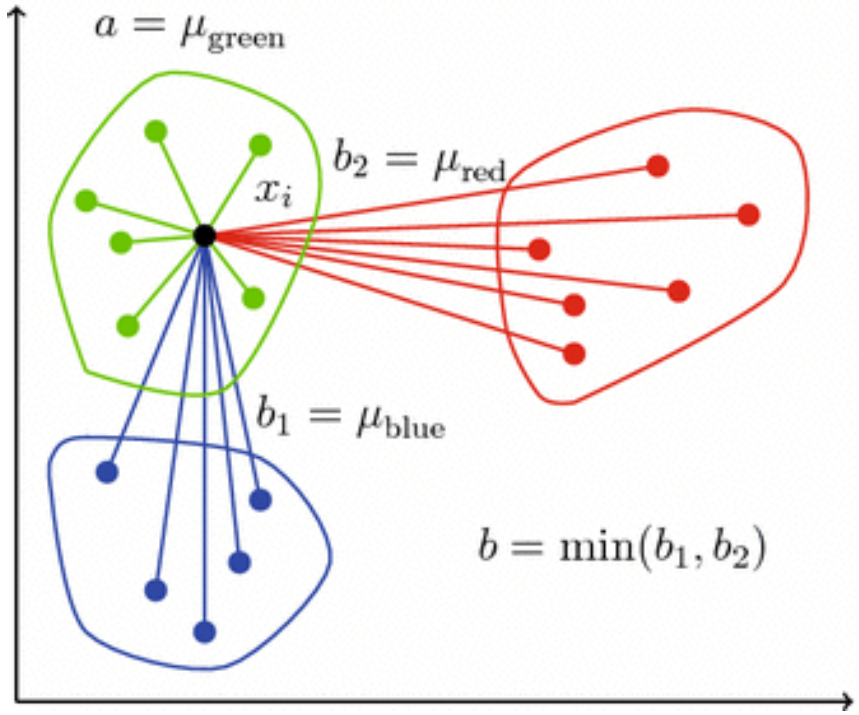
The Silhouette Coefficient is calculated using average distances:

(1) Determine the average distance from the point to the other points in its cluster, denoted as 'a'.

(2) Measure the average distance from the point to the points in the nearest cluster that it is not a part of, denoted as 'b'.

$$S(x) = \frac{b - a}{\max(a, b)}$$

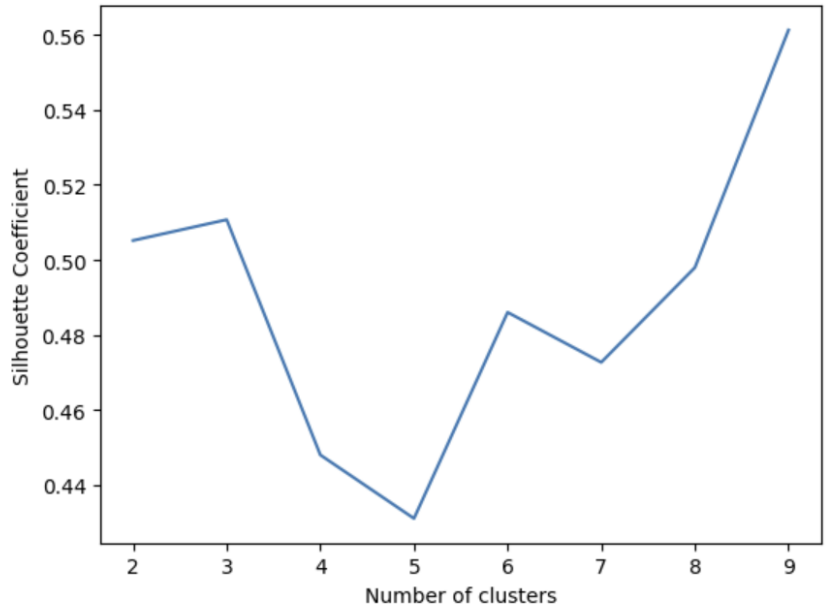
```
from sklearn.metrics import silhouette_score
```



Determine optimal 'K' using Silhouette Coefficient

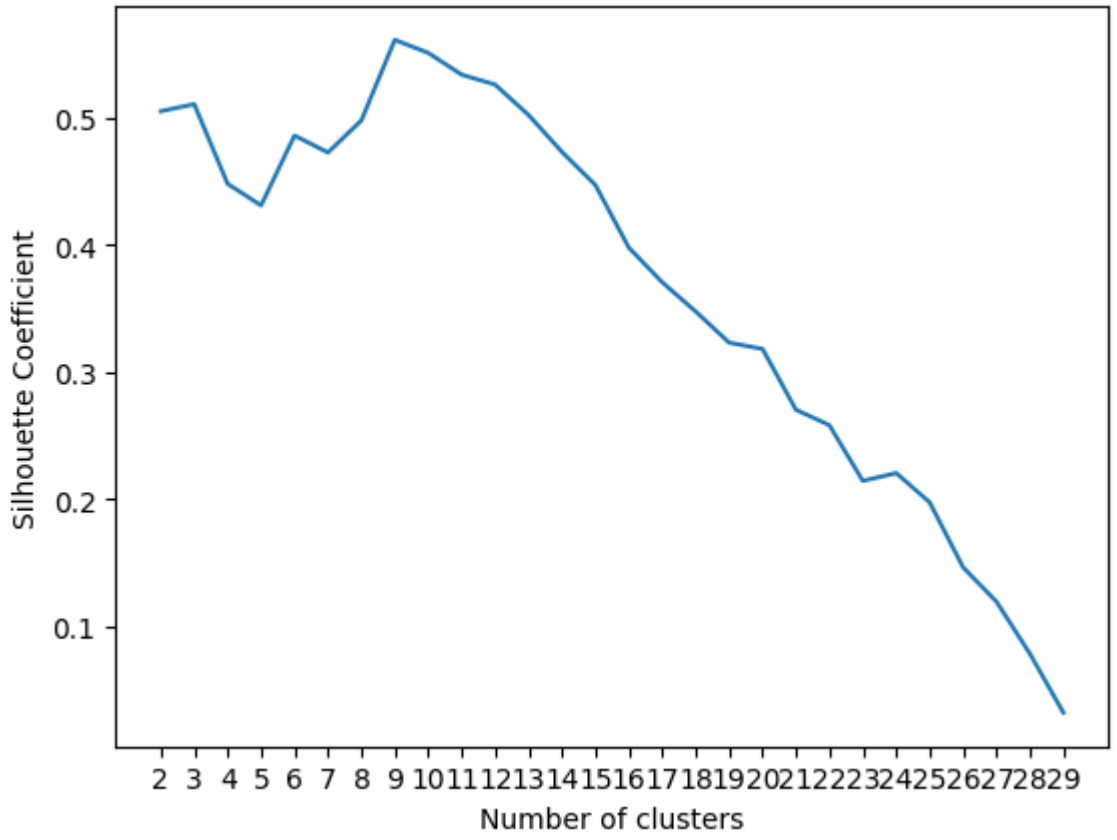
```
from sklearn.cluster import KMeans
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics

df = pd.read_csv('k-means.csv')
silhouette_avgs = []
min_k = 2
max_k = 10 # using len(df) for All possible ‘K’

for k in range(min_k, max_k):
    model = KMeans(n_clusters=k,random_state=0).fit(df)
    s_score = metrics.silhouette_score(df, model.labels_)
    print('Silhouette Coeff for k =', k, 'is', s_score)
    silhouette_avgs.append(s_score)

optimal_k = silhouette_avgs.index(max(silhouette_avgs)) +
min_k
print('Optimal K is ', optimal_k)
plt.plot(range(min_k, max_k), silhouette_avgs)
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Coefficient')
plt.xticks(range(min_k, max_k))
plt.show()
```

```
Silhouette Coeff for k = 2 is 0.5051959802345012
Silhouette Coeff for k = 3 is 0.5107383117230143
Silhouette Coeff for k = 4 is 0.4480846057409834
Silhouette Coeff for k = 5 is 0.43116909607087694
Silhouette Coeff for k = 6 is 0.4860676829904247
Silhouette Coeff for k = 7 is 0.4727516100080185
Silhouette Coeff for k = 8 is 0.4979798159711021
Silhouette Coeff for k = 9 is 0.5612456481793094
Optimal K is  9
```



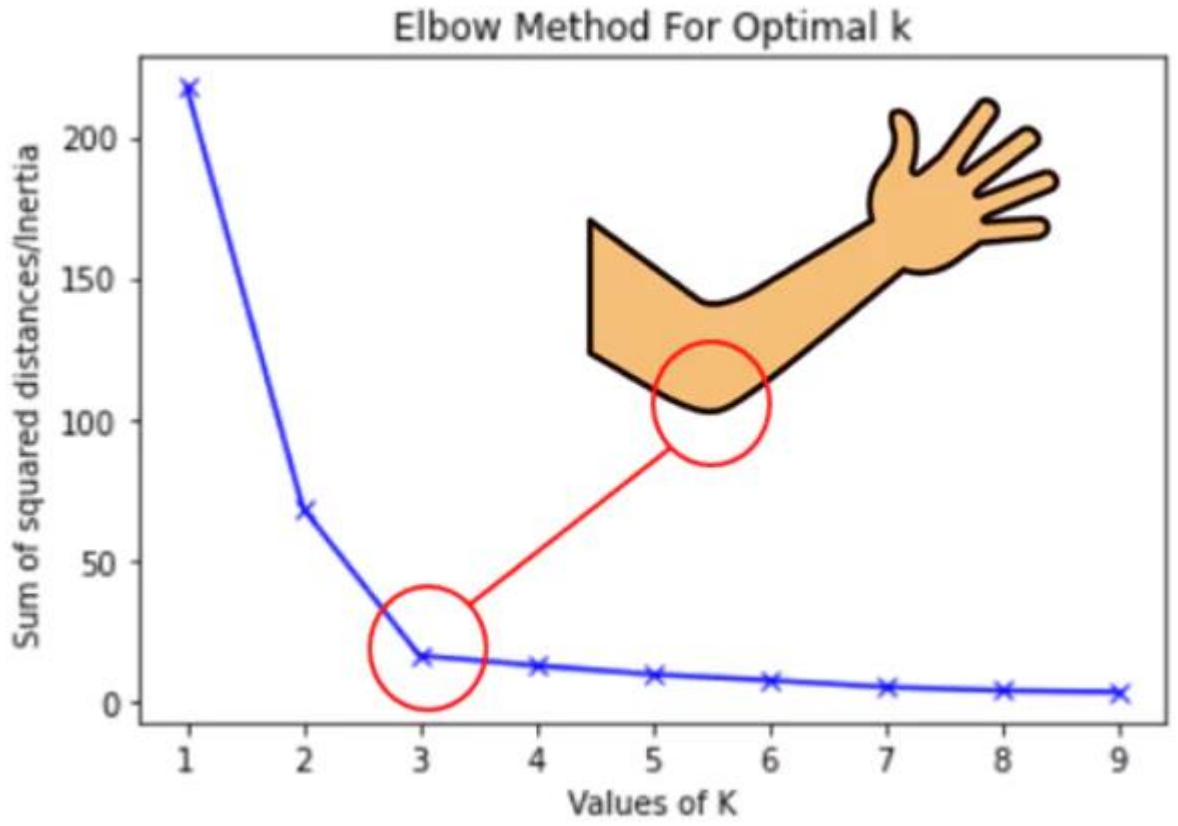
Determine optimal 'K' using Silhouette Coefficient

```
# using len(df) for All possible ‘K’
```



Determine optimal 'K' using Elbow Method

It is a popular tool that applies k-means for a range of values of k, and we plot it against the Sum of Squared Error, referred to as a scree plot.



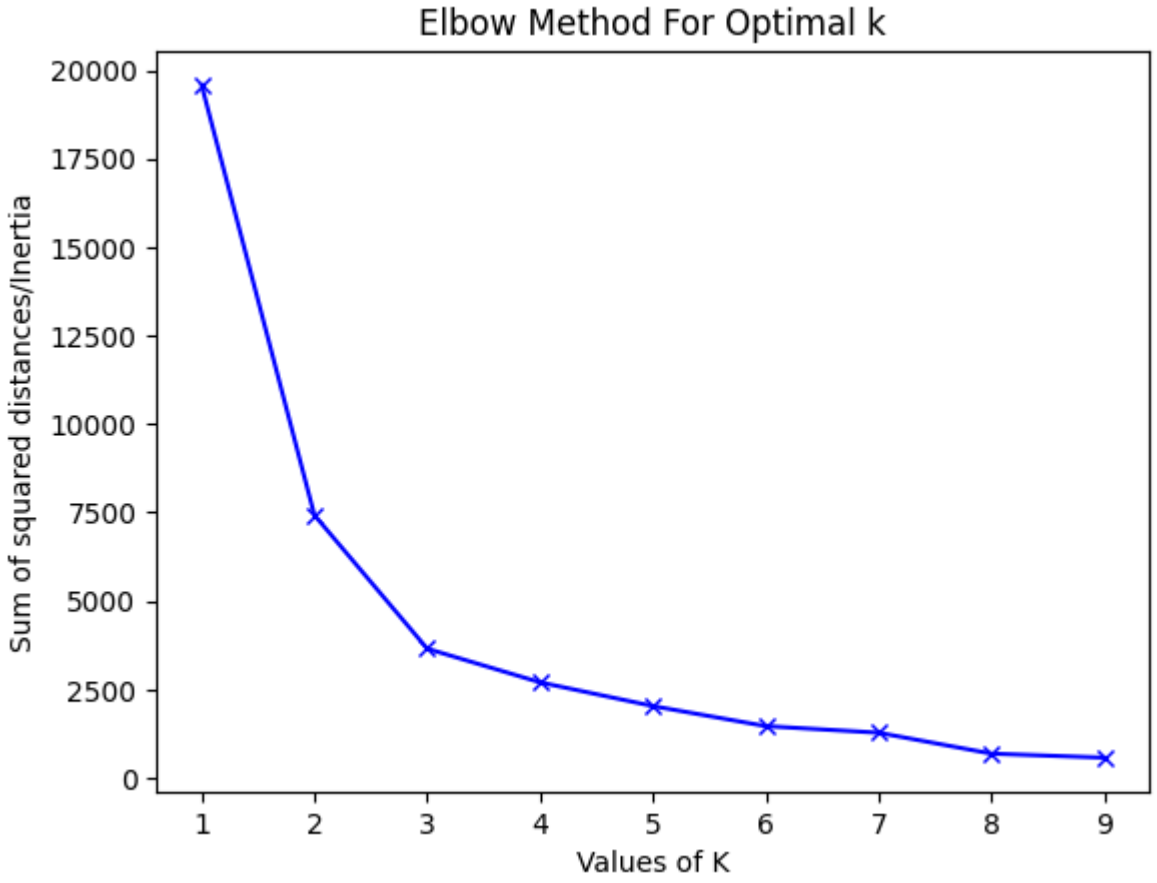
Line plot between K and inertia

https://www.analyticsvidhya.com/blog/2021/05/k-mean-getting-the-optimal-number-of-clusters/

Determine optimal 'K' using Elbow Method

```
from sklearn.cluster import KMeans
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics

df = pd.read_csv('k-means.csv')
#with pd.option_context('display.max_rows', 10): display(df)

silhouette_avgs = []
min_k = 2
# max_k = 10 # len(df)
max_k = len(df)
Sum_of_squared_distances = []
K = range(1,10)
for num_clusters in K :
    kmeans = KMeans(n_clusters=num_clusters)
    kmeans.fit(df)
    Sum_of_squared_distances.append(kmeans.inertia_)
plt.plot(K,Sum_of_squared_distances,'bx-')
plt.xlabel('Values of K')
plt.ylabel('Sum of squared distances/Inertia')
plt.title('Elbow Method For Optimal k')
plt.show()
```

Determine optimal 'K' using Elbow Method

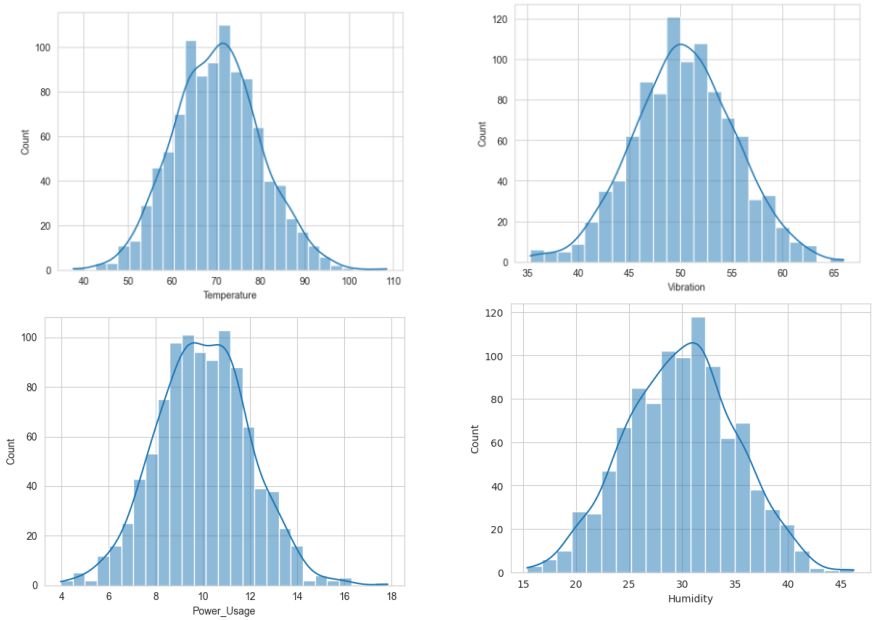


Assignment : Machine Failure Classification using sensor data

Develop predictive models for detecting machine failures

1. Develop model e.g. Logistic Regression, KNN, SVM, Decision Tree etc.

2. Illustrate confusion matrix and evaluate model
   (require at least 70% overall accuracy for test set)




```
Accuracy: 0.715
Confusion Matrix:
 [[143   0]
 [ 57   0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.71      1.00      0.83       143
           1       0.00      0.00      0.00        57

    accuracy                           0.71       200
   macro avg       0.36      0.50      0.42       200
weighted avg       0.51      0.71      0.60       200
```

Mechatronics

Day 6

Ensemble Learning, Hyperparameter tunnig and Anomaly detection

Outline

- Ensemble Learning
  - Random Forest
  - Gradient Boosting
  - Stacking Model
- Hyperparameters tunning
  - GridSeacrhCV and RandomizedSearchCV
- Handling imbalanced data
  - SMOTE
- Anomaly Detection

Ensemble Learning

It combines multiple models to improve predictive accuracy and robustness.

Bagging (Bootstrap Aggregating)

1. Trains multiple models (often decision trees) on different subsets of the training data, created by sampling with replacement (bootstrapping).
2. Combines predictions from individual models (e.g., by averaging or voting) to obtain a final prediction.

Example: **Random Forest**

Boosting

1. Trains models sequentially, where each subsequent model focuses on correcting the errors of the previous ones.
2. Weights are assigned to training samples, with more weight given to misclassified samples.

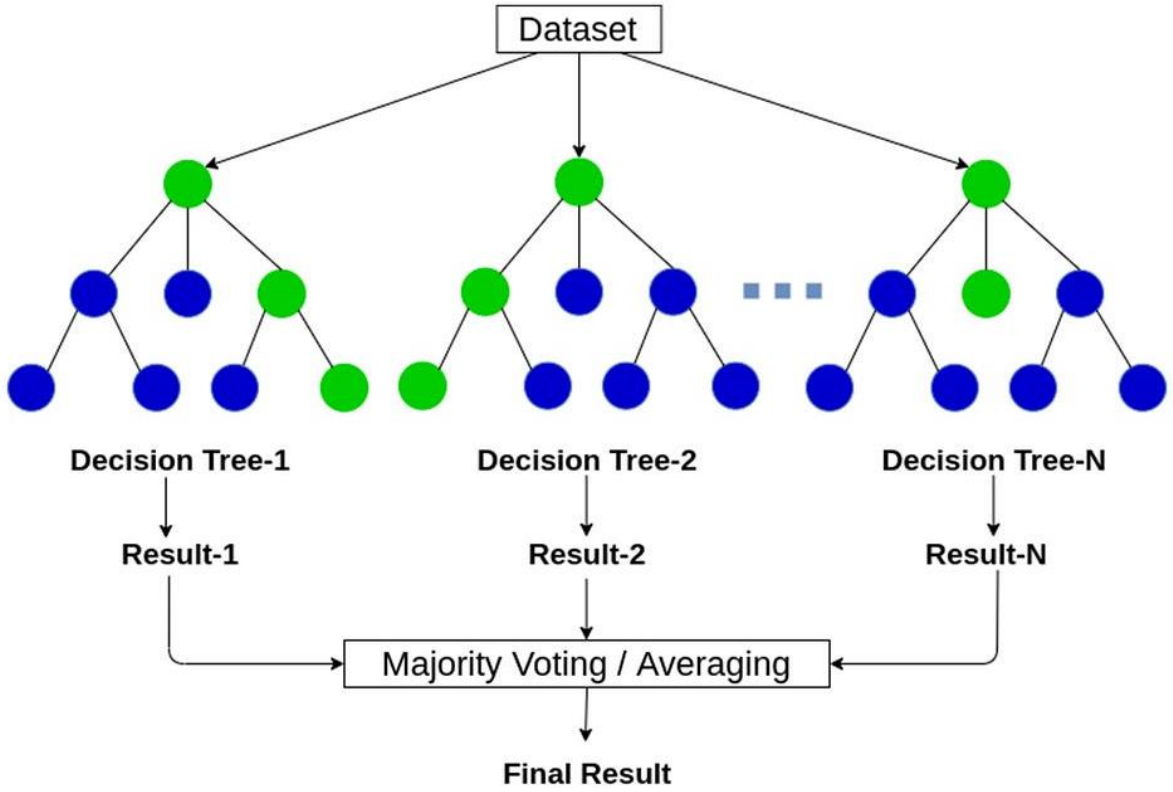
Examples: **AdaBoost, Gradient Boosting and XGBoost**

Stacking

1. Trains multiple base models on the original data.
2. Trains a meta-model (e.g., a linear regression or logistic regression) on the predictions of the base models as input features.
3. The meta-model learns to combine the predictions of the base models in an optimal way.

Random Forest model

It is a powerful machine learning algorithm that combines the predictions of multiple decision trees to improve overall performance.



https://medium.com/@abhishekjainindore24/everything-about-random-forest-90c106d63989

Random Forest model

```
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor

RandomForestClassifier(n_estimators = ‘Number of models’, criterion = [‘gini’, ‘entropy’],
                max_depth = ‘Number of layers/branches, random_state = ‘sets of random data’)

RandomForestRegressor(n_estimators = ‘Number of models’, criterion = [‘gini’, ‘entropy’],
                max_depth = ‘Number of layers/branches, random_state = ‘sets of random data’)
```

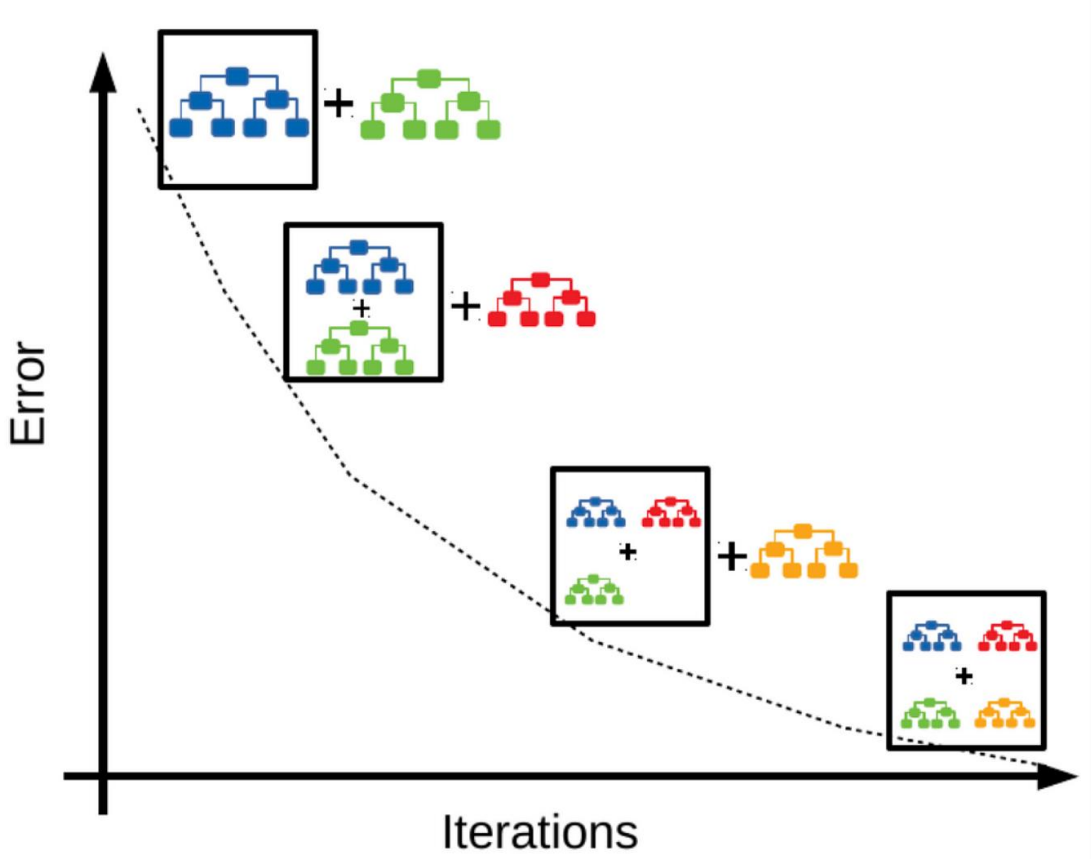
Random Forest model

```
# Random Forest for Classification
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Create Mockup data for Classification
x, y = make_classification(n_samples=100, n_features=5,
                           n_classes=2, random_state=0)
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0)
forest = RandomForestClassifier(n_estimators=5, random_state=0)
forest.fit(x_train, y_train)
score = forest.score(x_test, y_test)
print('Random Forest Score:', score)

######## Compare with Decision Tree ##########
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier()
tree.fit(x_train, y_train)
tree_score = tree.score(x_test, y_test)
print('Decision Tree Score:', tree_score)
```

```
Random Forest Score: 0.88
Decision Tree Score: 0.84
```

Gradient Boosting

It is a powerful machine learning technique that builds upon the concept of boosting. It sequentially combines multiple weak learners (typically decision trees) to create a strong predictive model.



https://medium.com/@hemashreekilari9/understanding-gradient-boosting-632939b98764

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor


GradientBoostingClassifier(learning_rate = ‘a value of learning rate’,
                           random_state = ‘sets of random data’)

GradientBoostingRegressor(learning_rate = ‘a value of learning rate’,
                           random_state = ‘sets of random data’)
```

Gradient Boosting for Classification

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier

x, y = make_classification(n_samples=500, n_features=5,
                           n_classes=2, random_state=0)

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0)

gb = GradientBoostingClassifier(learning_rate=0.1, random_state=0)
gb.fit(x_train, y_train)

score = gb.score(x_test, y_test)
print('Score:', score)
```

```
Score: 0.912
```

Gradient Boosting for Regression

```
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor

x, y = make_regression(n_samples=500, n_features=5, random_state=0)

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0)

gb = GradientBoostingRegressor(learning_rate=0.1, random_state=0)
gb.fit(x_train, y_train)

score = gb.score(x_test, y_test)
print('Score:', score)
```

```
Score: 0.9406053737252129
```

Stacking Model

It is an ensemble learning technique that combines the predictions of multiple base models (also known as first-level models or base learners) to produce a more accurate final prediction.

(1) Train Base Models (Level 0)

- Multiple base models are trained independently on the original training data.

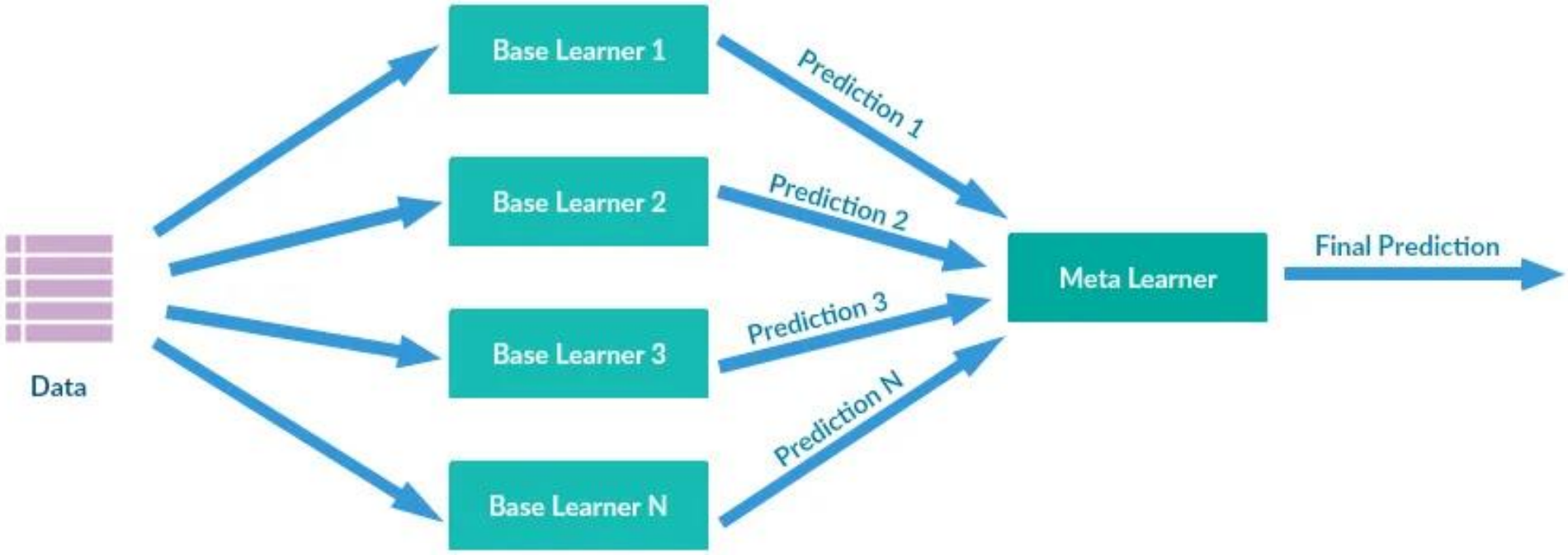
(2) Create a New Dataset

- The predictions from the base models are used to create a new dataset.
- Each base model's predictions become a new feature in this dataset.

(3) Train a Meta-model (Level 1)

- A new model, called the "meta-model" or "blender," is trained on this new dataset.
- The meta-model learns how to best combine the predictions of the base models to make the final prediction.
- Common meta-models usually are **linear regression and logistic regression**.

Stacking Model



https://supunsetunga.medium.com/stacking-in-machine-learning-357db1cfc3a

Stacking Model

```
from sklearn.ensemble import StackingClassifier, StackingRegressor

# Base Learnner (Level 0)
base_estimators = [
    (‘model name 1’, MODEL1(HYPERPARAMETER)),
    ('model name 2', MODEL2(HYPERPARAMETER)),
    ('model name 3', MODEL3(HYPERPARAMETER)),
    ('model name n', MODELn(HYPERPARAMETER))]

# Meta Learnner (Level 1) for Classification
StackingClassifier(estimators=base_estimators,
                   final_estimator=LogisticRegression())

# Meta Learnner (Level 1) for Regression
StackingRegressor(estimators=base_estimators,
                  final_estimator=LinearRegression())
```

```python
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.ensemble import StackingClassifier

x, y = make_classification(n_samples=1000, n_features=4,
                           n_classes=2, random_state=100)

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=100)

# Base Learnner (Level 0) : from many models
base_estimators = [
    ('tree', DecisionTreeClassifier(random_state=100)),
    ('svm', SVC(kernel='linear')),
    ('knn', KNeighborsClassifier()),
    ('bayes', GaussianNB()),]
# Meta Learnner (Level 1) : defult is Logistic Regression
stack = StackingClassifier(estimators=base_estimators,
                           final_estimator=LogisticRegression())
stack.fit(x_train, y_train)
score = stack.score(x_test, y_test)
print('Score:', score)
```

```
Score: 0.932
```

Anomaly Detection

It, also known as outlier detection, is a data mining technique used to identify data points that deviate significantly from the norm. These anomalies can be indicative of errors, fraud, or other interesting events.

The real-world applications of anomaly detection

- Finance and fraud detection
- Cybersecurity
- Healthcare
- Industrial equipment monitoring
- Network intrusion detection
- Energy grid monitoring
- E-commerce and user behavior analysis
- Quality control in manufacturing

Anomaly Detection models

1. Supervised Learning Models
- Support Vector Machines (SVM)
- k-Nearest Neighbors (KNN)
- Classification Trees

2. Unsupervised Learning Models
- One-Class SVM
- Isolation Forest
- Local Outlier Factor (LOF)
- Autoencoders
- Gaussian Mixture Models (GMM):
- Hidden Markov Models (HMM)

3. Other Techniques
- k-means Clustering
- Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

Example 11 : Fraud Detection

Dataset : https://www.kaggle.com/datasets/valakhorasani/bank-transaction-dataset-for-fraud-detection

Example 11 : Fraud Detection

Key Features

TransactionID	A unique identifier for each transaction.
AccountID	A unique identifier for the account associated with the transaction.
TransactionAmount	The monetary value of each transaction, varying from small expenses to large purchases.
TransactionDate	The date and time when the transaction occurred.
TransactionType	Categorical value indicating whether the transaction was a 'Credit' or 'Debit'.
Location	The geographic location where the transaction took place.
DeviceID	A unique identifier for the device used to perform the transaction.
IP Address	The IP address associated with the transaction.
MerchantID	A unique identifier for merchants involved in the transaction.
Channel	Indicates the channel through which the transaction was conducted (e.g., Online, ATM, Branch).
AccountBalance	The balance remaining in the account after the transaction.
TransactionDuration	Duration of the transaction in seconds.
LoginAttempts	The number of login attempts made before the transaction.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.ensemble import IsolationForest
from sklearn.cluster import DBSCAN
from sklearn.neighbors import LocalOutlierFactor

import warnings
warnings.filterwarnings("ignore")

# Load the dataset
df = pd.read_csv('Anomaly_bank_transactions_data_2.csv')
# Display basic information about the dataset
print("Shape of the dataset:", df.shape)
# display(df.head())
print("\nDataset Information:")
print(df.info())
print("\nStatistical Summary:")
display(df.describe().T)
```

```
Shape of the dataset: (2512, 16)

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2512 entries, 0 to 2511
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   TransactionID           2512 non-null   object
 1   AccountID               2512 non-null   object
 2   TransactionAmount       2512 non-null   float64
 3   TransactionDate         2512 non-null   object
 4   TransactionType         2512 non-null   object
 5   Location                2512 non-null   object
 6   DeviceID                2512 non-null   object
 7   IP Address              2512 non-null   object
 8   MerchantID              2512 non-null   object
 9   Channel                 2512 non-null   object
 10  CustomerAge             2512 non-null   int64
 11  CustomerOccupation      2512 non-null   object
 12  TransactionDuration     2512 non-null   int64
 13  LoginAttempts           2512 non-null   int64
 14  AccountBalance          2512 non-null   float64
 15  PreviousTransactionDate  2512 non-null   object
dtypes: float64(2), int64(3), object(11)
memory usage: 314.1+ KB
None
```

Statistical Summary:

	count	mean	std	min	25%	50%	75%	max
TransactionAmount	2512.0	297.593778	291.946243	0.26	81.885	211.14	414.5275	1919.11
CustomerAge	2512.0	44.673965	17.792198	18.00	27.000	45.00	59.0000	80.00
TransactionDuration	2512.0	119.643312	69.963757	10.00	63.000	112.50	161.0000	300.00
LoginAttempts	2512.0	1.124602	0.602662	1.00	1.000	1.00	1.0000	5.00
AccountBalance	2512.0	5114.302966	3900.942499	101.25	1504.370	4735.51	7678.8200	14977.99

```python
# Check for missing and duplicated values
print(f'\nMissing values: {df.isna().sum().sum()}')
print(f'Duplicated values: {df.duplicated().sum()}')
```

```
Missing values: 0
Duplicated values: 0
```

```python
# Display the number of unique values in each column
print("\nUnique Values in Each Column:")
print(df.nunique())
```

```
Unique Values in Each Column:
TransactionID              2512
AccountID                   495
TransactionAmount          2455
TransactionDate            2512
TransactionType               2
Location                     43
DeviceID                    681
IP Address                  592
MerchantID                  100
Channel                       3
CustomerAge                  63
CustomerOccupation            4
TransactionDuration         288
LoginAttempts                 5
AccountBalance             2510
PreviousTransactionDate     360
dtype: int64
```

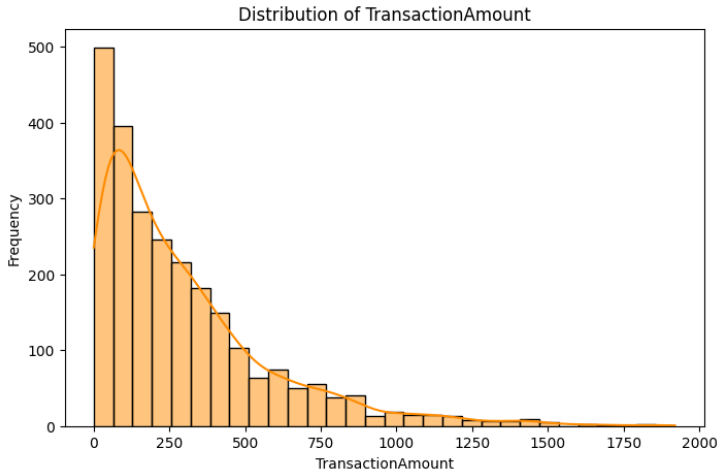
```python
# Separate numerical and categorical columns
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
non_numerical_columns = df.select_dtypes(include=['object']).columns.tolist()

# Display the lists of numerical and categorical columns
print("\nNumerical Columns:", numerical_columns)
print("Categorical Columns:", non_numerical_columns)
```

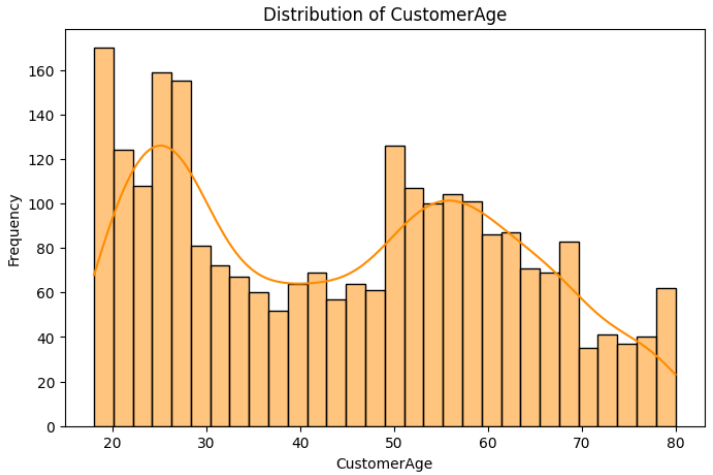
```
Numerical Columns: ['TransactionAmount', 'CustomerAge', 'TransactionDuration', 'LoginAttempts', 'AccountBalance’]

Categorical Columns: ['TransactionID', 'AccountID', 'TransactionDate', 'TransactionType', 'Location', 'DeviceID',
'IP Address', 'MerchantID', 'Channel', 'CustomerOccupation', 'PreviousTransactionDate']
```

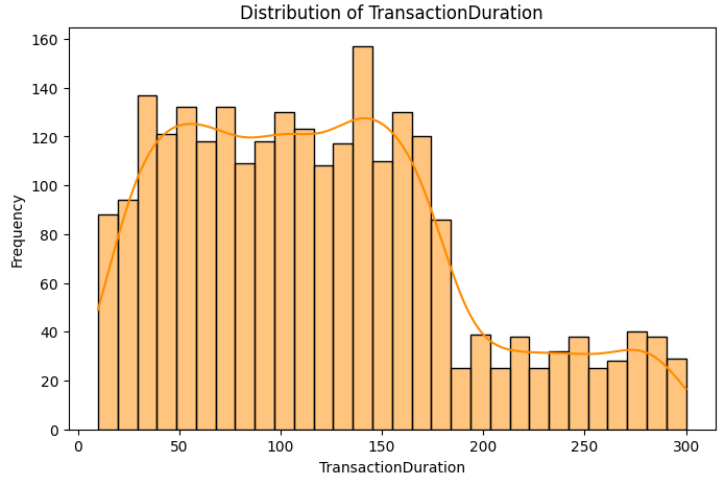
Numerical Data : Exploratory Data Analytics (EDA)

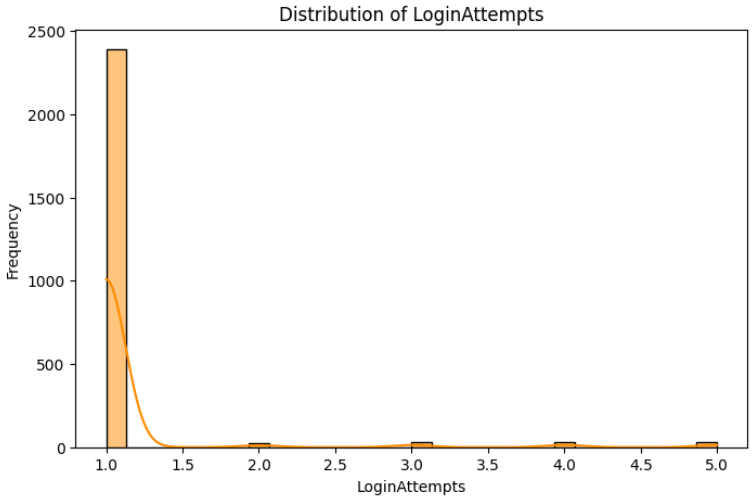
```
def plot_histograms(df, numerical_columns):
    for column in numerical_columns:
        plt.figure(figsize=(8, 5))
        # Create a histogram with KDE
        sns.histplot(df[column], bins=30, kde=True, color='#ff8c00')
        plt.title(f'Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Frequency')
        plt.show()


numerical_columns = ['TransactionAmount', 'CustomerAge', 'TransactionDuration', 'LoginAttempts',
'AccountBalance']

plot_histograms(df, numerical_columns)
```

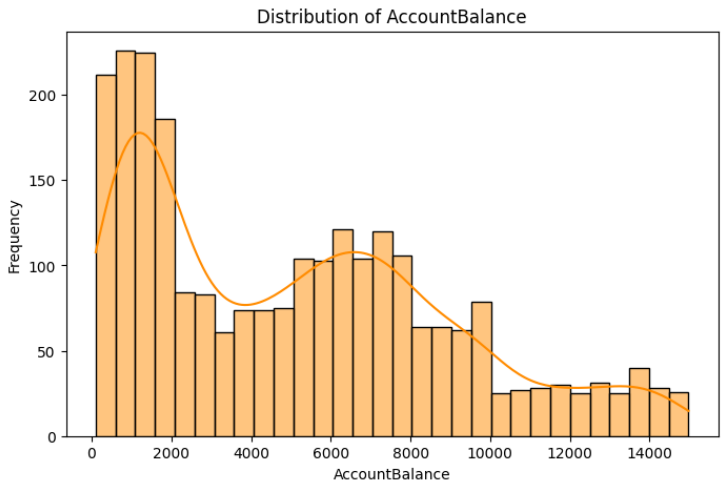
Numerical Data : Exploratory Data Analytics (EDA)







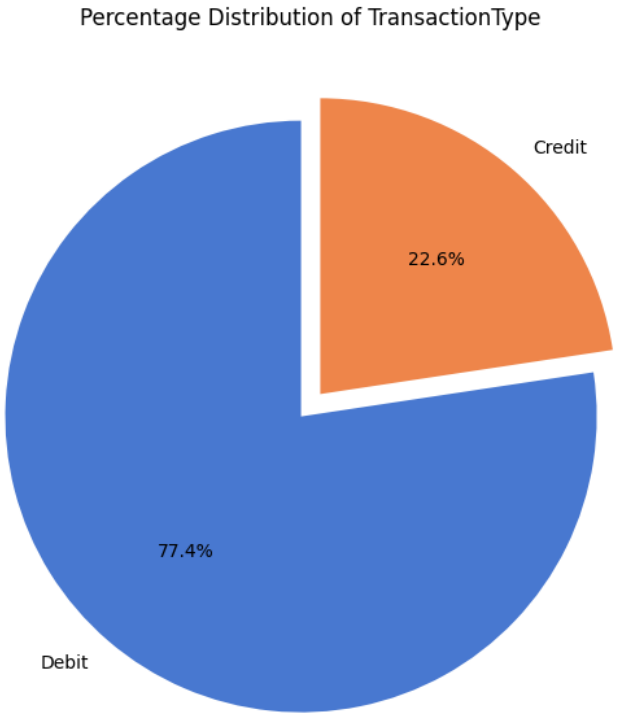


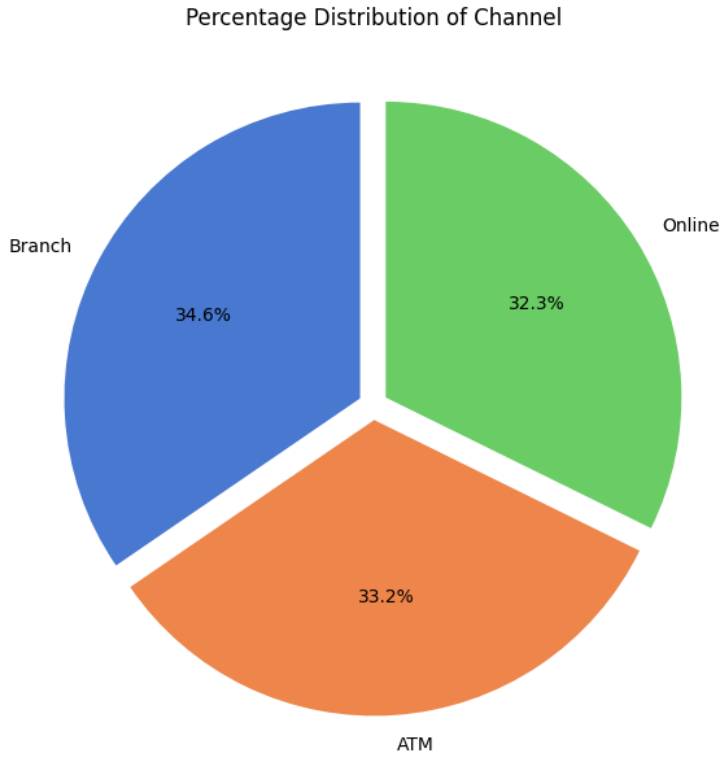


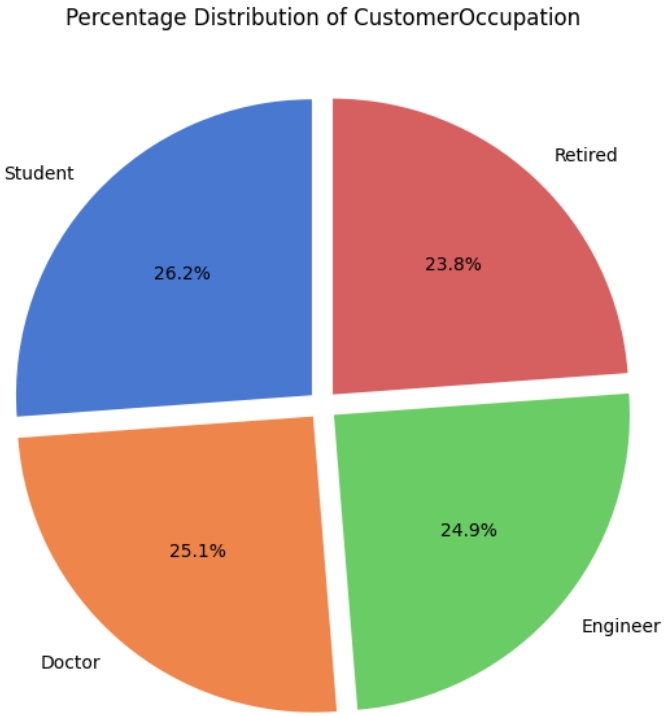
Categorical Data : Exploratory Data Analytics (EDA)

```
def plot_categorical_distributions(columns, data=df, palette='muted'):
    plt.figure(figsize=(18, 6))

    # Loop through each column and create a pie chart
    for i, column_name in enumerate(columns):
        plt.subplot(1, 3, i + 1)
        value_counts = data[column_name].value_counts()
        value_counts.plot.pie(autopct='%1.1f%%', colors=sns.color_palette(palette),
                              startangle=90, explode=[0.05] * value_counts.nunique())

        plt.title(f'Percentage Distribution of {column_name}')
        plt.ylabel('')

    plt.tight_layout()
    plt.show()

# Example usage
columns_to_plot = ['TransactionType', 'Channel', 'CustomerOccupation']
plot_categorical_distributions(columns_to_plot)
```

Categorical Data : Exploratory Data Analytics (EDA)

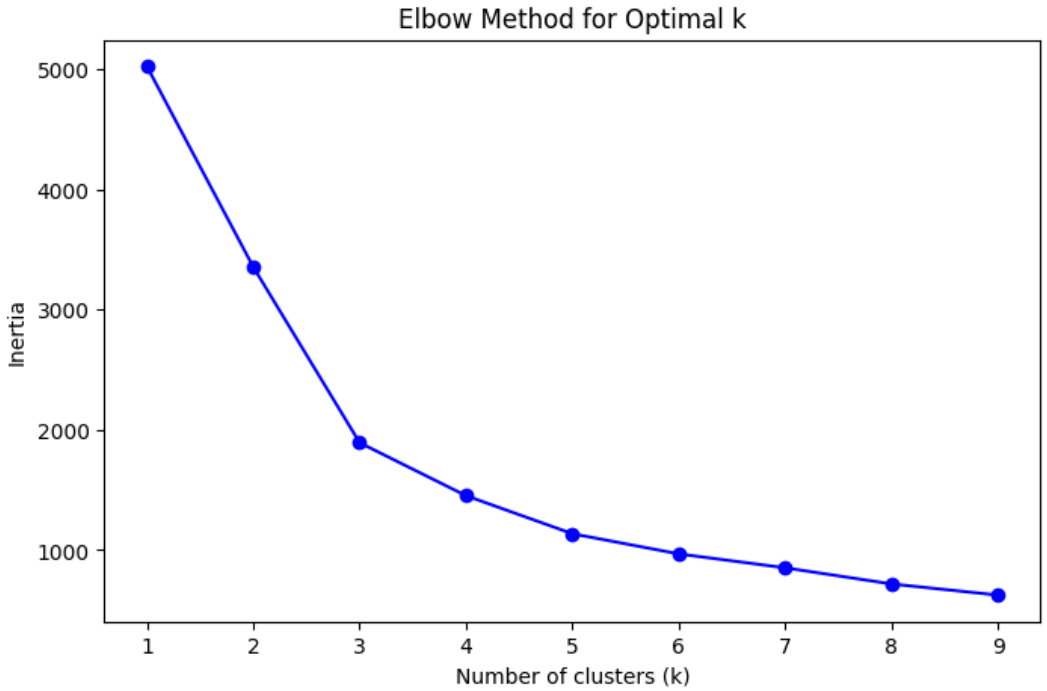






Identifying Potential Frauds with **K-means Clustering**

```
# Select features for clustering
features = ['TransactionAmount', 'TransactionDuration']
X = df[features].copy()
# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Determine the optimal number of clusters using the Elbow
Method
inertia = []
K = range(1, 10)  # Test for clusters from 1 to 10
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow Method
plt.figure(figsize=(8, 5))
plt.plot(K, inertia, 'bo-')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```



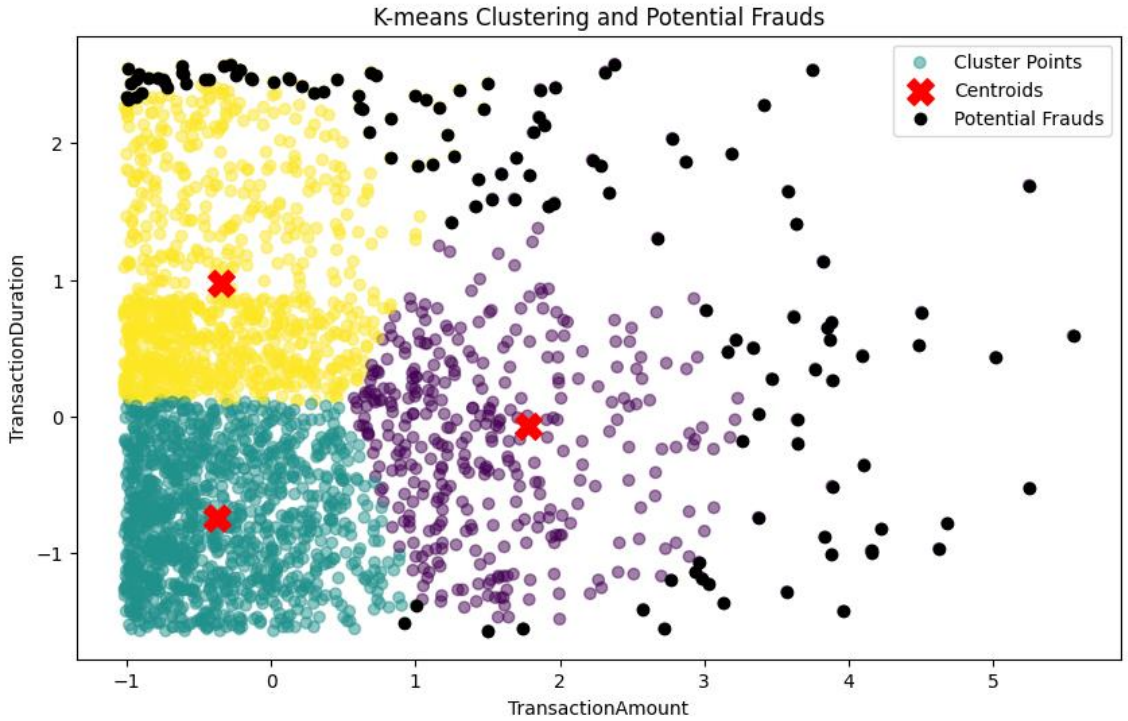
```python
# Fit K-means with the chosen number of clusters (k=3)
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X_scaled)

# Assign clusters and calculate distance to cluster centroid
df['Cluster'] = kmeans.labels_
df['DistanceToCentroid'] = np.linalg.norm(X_scaled - kmeans.cluster_centers_[kmeans.labels_], axis=1)

# Identify potential frauds based on distance threshold
threshold = df['DistanceToCentroid'].quantile(0.95)
potential_frauds = df[df['DistanceToCentroid'] > threshold]

print(f"Number of potential frauds detected: {len(potential_frauds)}")
display(potential_frauds)
```

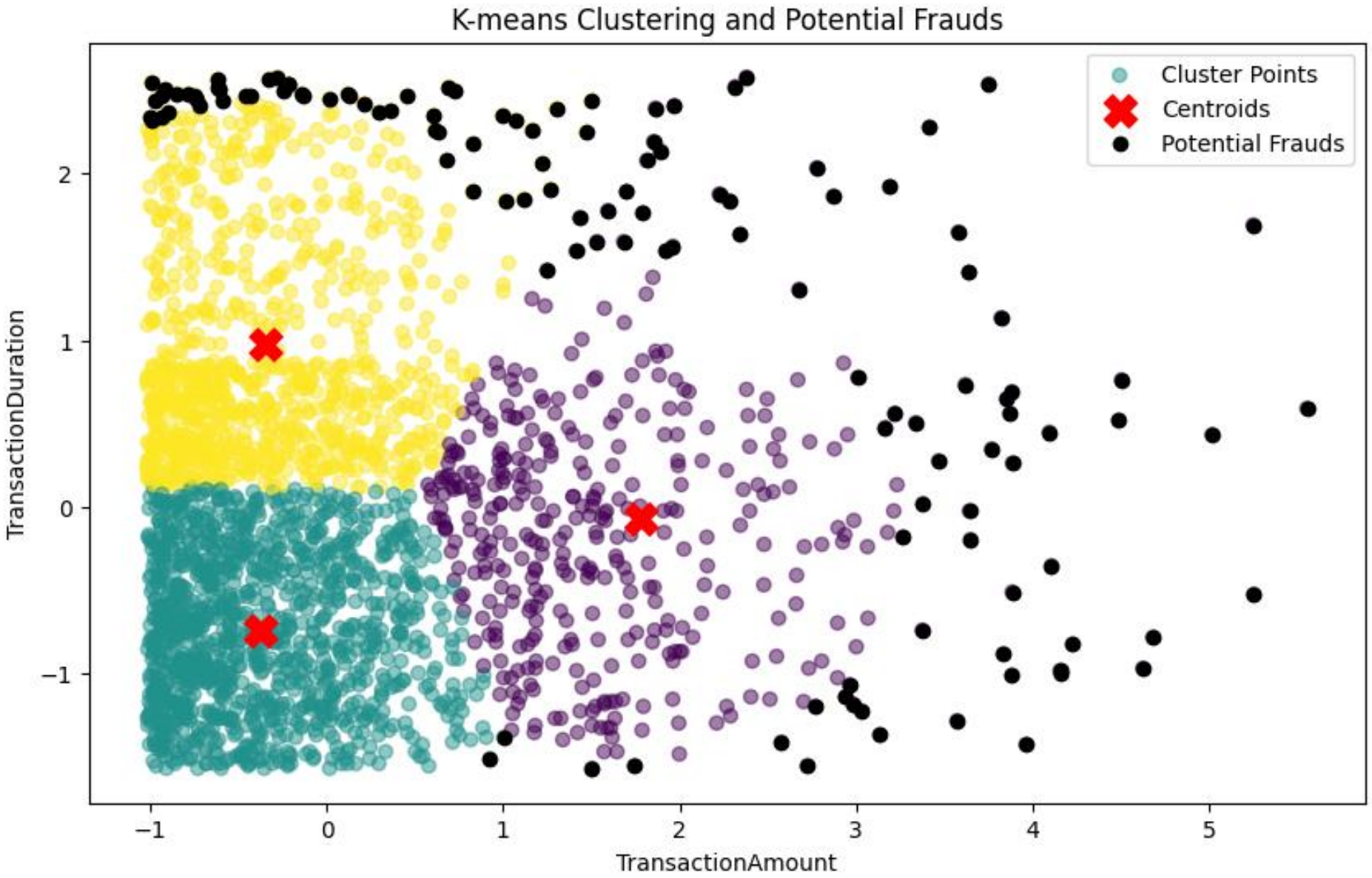
```
Number of potential frauds detected: 126
```

	TransactionID	AccountID	TransactionAmount	TransactionDate	TransactionType	Location	DeviceID	IP Address	MerchantID	Channel	...	TransactionDuration	LoginAt
74	TX000075	AC00265	1212.51	2023-10-04 16:36:29	Debit	Indianapolis	D000231	193.83.0.183	M036	Branch	...	24	
85	TX000086	AC00098	1340.19	2023-09-29 17:22:10	Credit	Austin	D000574	165.114.224.47	M012	Online	...	30	
141	TX000142	AC00114	1049.92	2023-10-23 16:50:33	Debit	Detroit	D000522	121.67.144.20	M052	ATM	...	21	
142	TX000143	AC00163	227.14	2023-07-03 17:42:08	Debit	Charlotte	D000439	197.162.55.147	M057	ATM	...	294	
146	TX000147	AC00385	973.39	2023-08-30 17:23:20	Debit	Sacramento	D000292	202.194.199.70	M026	Branch	...	296	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2403	TX002404	AC00111	1493.00	2023-06-07 17:05:41	Debit	Colorado Springs	D000344	136.162.111.135	M096	ATM	...	151	
2414	TX002415	AC00028	1664.33	2023-09-25 17:11:19	Debit	San Antonio	D000072	116.106.207.139	M064	Branch	...	65	
2439	TX002440	AC00439	538.17	2023-09-26 17:27:17	Credit	Washington	D000430	116.44.12.250	M055	Branch	...	252	
2445	TX002446	AC00439	403.01	2023-09-04 17:32:35	Debit	Washington	D000677	223.32.70.156	M029	Online	...	286	
2458	TX002459	AC00312	430.83	2023-08-14 16:09:21	Debit	Los Angeles	D000195	87.50.72.69	M079	Branch	...	292	

126 rows × 22 columns

```
# Visualize clusters and potential frauds (2D plot for simplicity with legend)
plt.figure(figsize=(10, 6))

# Plot clusters
scatter = plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=kmeans.labels_, cmap='viridis', alpha=0.5, label='Cluster
Points')
# Plot cluster centroids
centroids = plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=200, c='red', marker='X',
label='Centroids')
# Plot potential frauds
frauds = plt.scatter(X_scaled[potential_frauds.index, 0], X_scaled[potential_frauds.index, 1], c='black',
label='Potential Frauds', edgecolors='k')

plt.xlabel(features[0])
plt.ylabel(features[1])
plt.title('K-means Clustering and Potential Frauds')
plt.legend(loc='upper right')
plt.show()
```




K-means Clustering and Potential Frauds
Cluster Points
Centroids
Potential Frauds
TransactionDuration
TransactionAmount
2
1
0
−1
−1
0
1
2
3
4
5

DBSCAN Clustering for Anomaly Detection

```
# Map the cluster labels to descriptive names
label_mapping = {
    -1: 'Fraud (Outliers)',  # Default noise label for DBSCAN
    0: 'Normal',
    1: 'Suspicious Group 1',
    2: 'Suspicious Group 2',
    3: 'Suspicious Group 3',
    4: 'Suspicious Group 4',
}

# Select relevant features for DBSCAN
features = ['TransactionAmount', 'TransactionDuration', 'AccountBalance', 'LoginAttempts']
X = df[features].copy()
X = X.fillna(X.mean())

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

DBSCAN Clustering for Anomaly Detection

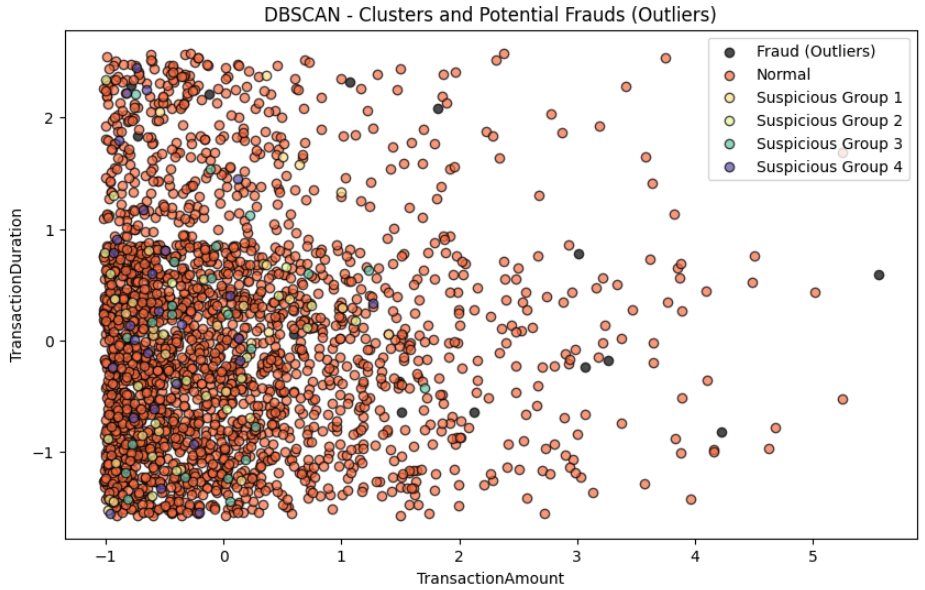
```
# Apply DBSCAN
dbscan = DBSCAN(eps=1.5, min_samples=5)
dbscan.fit(X_scaled)

# Add labels to the dataframe
df['DBSCAN_Cluster'] = dbscan.labels_

# Map cluster labels to descriptive names
df['Cluster_Description'] = df['DBSCAN_Cluster'].map(label_mapping)

# Identify outliers (noise points) labeled as -1
potential_frauds = df[df['DBSCAN_Cluster'] == -1]
print(f"Number of potential frauds detected by DBSCAN: {len(potential_frauds)}")
display(potential_frauds.head(5))
```

```
Number of potential frauds detected by DBSCAN: 17
```

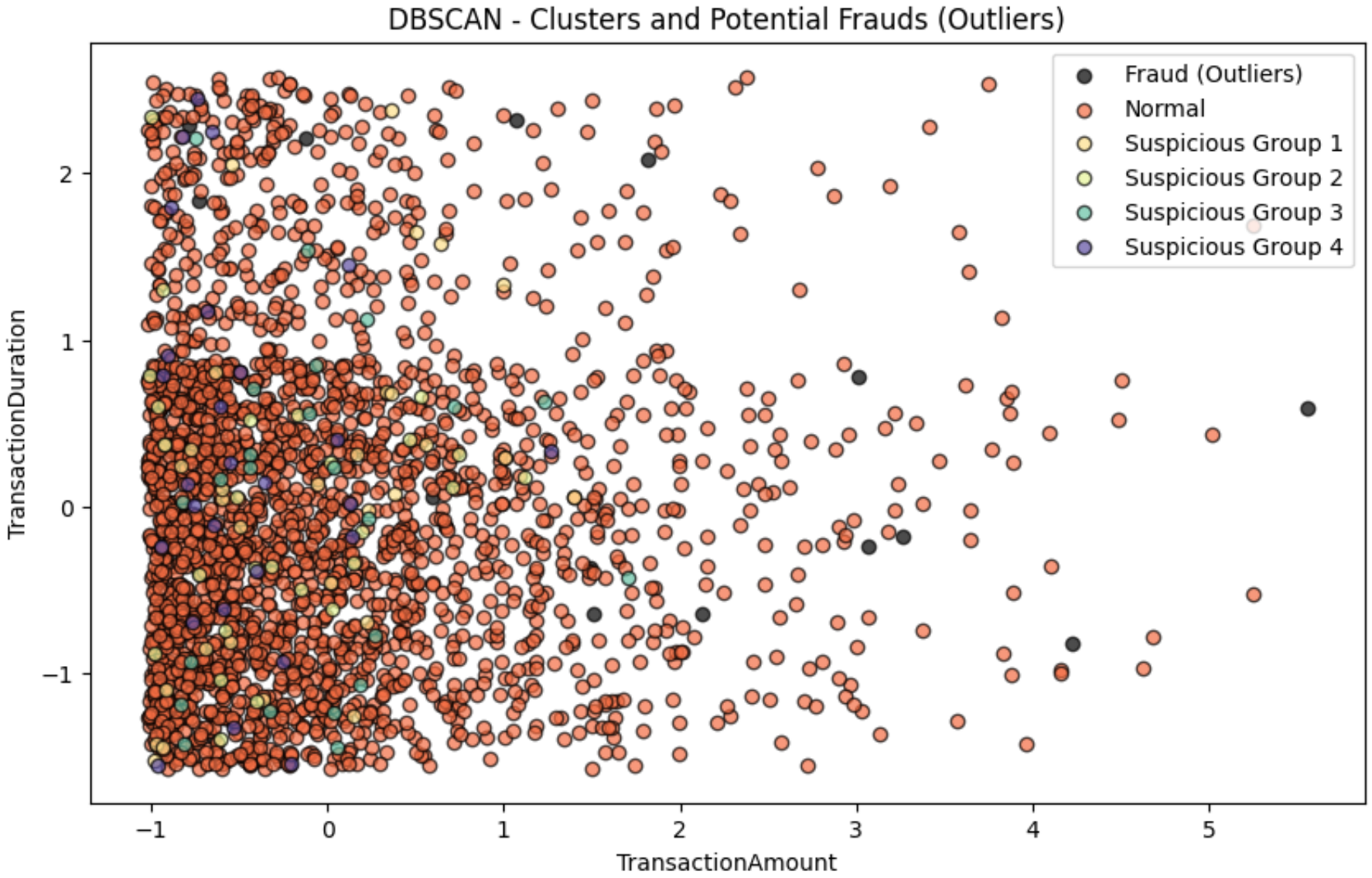
	TransactionID	AccountID	TransactionAmount	TransactionDate	TransactionType	Location	DeviceID	IP Address	MerchantID	Channel	CustomerAge	CustomerOccupation
274	TX000275	AC00454	1176.28	2023-12-20 16:08:02	Credit	Kansas City	D000476	50.202.8.53	M074	ATM	54	Engineer
454	TX000455	AC00264	611.11	2023-10-18 18:32:31	Debit	Detroit	D000215	141.201.46.191	M045	ATM	20	Student
653	TX000654	AC00423	1919.11	2023-06-27 17:48:25	Debit	Portland	D000191	207.157.126.125	M033	ATM	30	Doctor
693	TX000694	AC00011	733.29	2023-03-15 18:42:16	Debit	Virginia Beach	D000618	16.51.235.240	M032	ATM	52	Doctor
754	TX000755	AC00153	84.34	2023-06-08 16:27:56	Debit	Memphis	D000493	200.136.146.93	M039	Online	58	Retired

DBSCAN Clustering for Anomaly Detection

```
# Visualize clusters and potential frauds
plt.figure(figsize=(10, 6))
unique_labels = np.unique(dbscan.labels_)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]

for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise (outliers)
        col = [0, 0, 0, 1]
    class_member_mask = (dbscan.labels_ == k)
    xy = X_scaled[class_member_mask]
    plt.scatter(xy[:, 0], xy[:, 1], color=tuple(col), edgecolor='k',
         alpha=0.7, label=label_mapping.get(k, f'Cluster {k}'))

plt.title('DBSCAN - Clusters and Potential Frauds (Outliers)')
plt.xlabel(features[0])  # TransactionAmount
plt.ylabel(features[1])  # TransactionDuration
plt.legend()
plt.show()
```



DBSCAN Clustering for Anomaly Detection



Isolation Forest for Anomaly Detection

```
# Define outlier mapping
outlier_mapping = {1: 'Normal', -1: 'Potential Fraud'}

# Select relevant features for fraud detection
features = ['TransactionAmount', 'TransactionDuration', 'AccountBalance', 'LoginAttempts']  # Modify as needed
X = df[features].copy()
X = X.fillna(X.mean())

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Isolation Forest for Anomaly Detection

```
# Fit the Isolation Forest model
iso_forest = IsolationForest(n_estimators=100, contamination=0.05, random_state=42)
iso_forest.fit(X_scaled)

# Predict anomalies
df['AnomalyScore'] = iso_forest.decision_function(X_scaled)
df['IsAnomaly'] = iso_forest.predict(X_scaled)

# Map results to descriptive labels
df['AnomalyLabel'] = df['IsAnomaly'].map(outlier_mapping)
# Filter out detected anomalies
potential_frauds = df[df['IsAnomaly'] == -1]
print(f"Number of potential frauds detected: {len(potential_frauds)}")
display(potential_frauds.head(5))
```

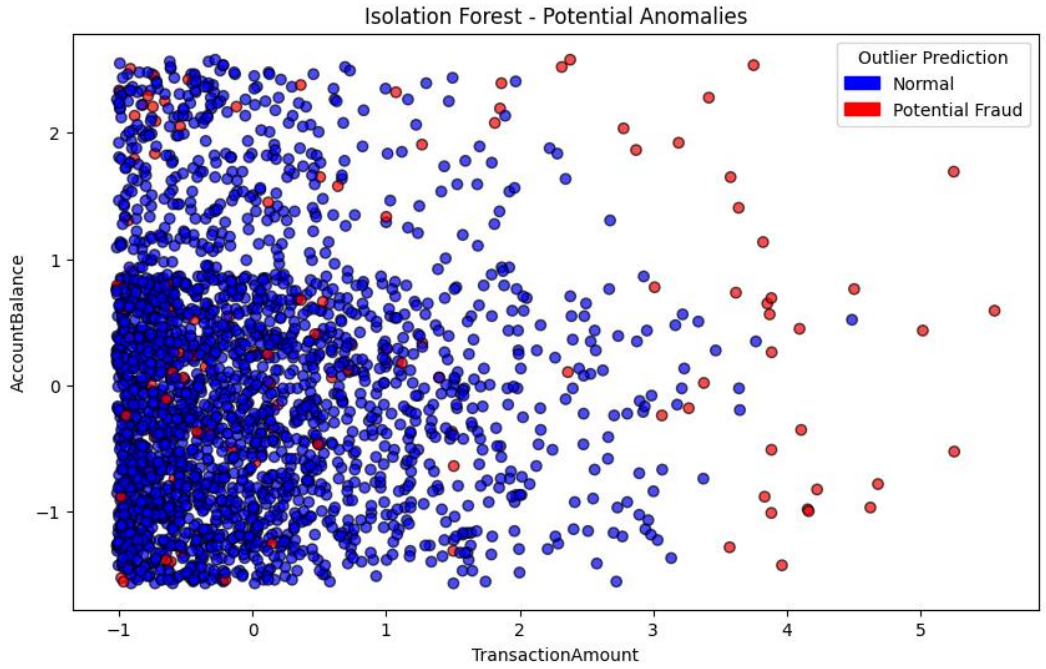
```
Number of potential frauds detected: 126
```

	TransactionID	AccountID	TransactionAmount	TransactionDate	TransactionType	Location	DeviceID	IP Address	MerchantID	Channel	...	CustomerOccupation	Transact
26	TX000027	AC00441	246.93	2023-04-17 16:37:01	Debit	Miami	D000046	55.154.161.250	M029	ATM	...	Student	
32	TX000033	AC00060	396.45	2023-09-25 16:26:00	Debit	New York	D000621	133.67.250.163	M007	ATM	...	Engineer	
85	TX000086	AC00098	1340.19	2023-09-29 17:22:10	Credit	Austin	D000574	165.114.224.47	M012	Online	...	Engineer	
91	TX000092	AC00310	223.85	2023-10-02 16:36:10	Debit	Kansas City	D000481	133.223.159.151	M009	ATM	...	Engineer	
146	TX000147	AC00385	973.39	2023-08-30 17:23:20	Debit	Sacramento	D000292	202.194.199.70	M026	Branch	...	Retired	

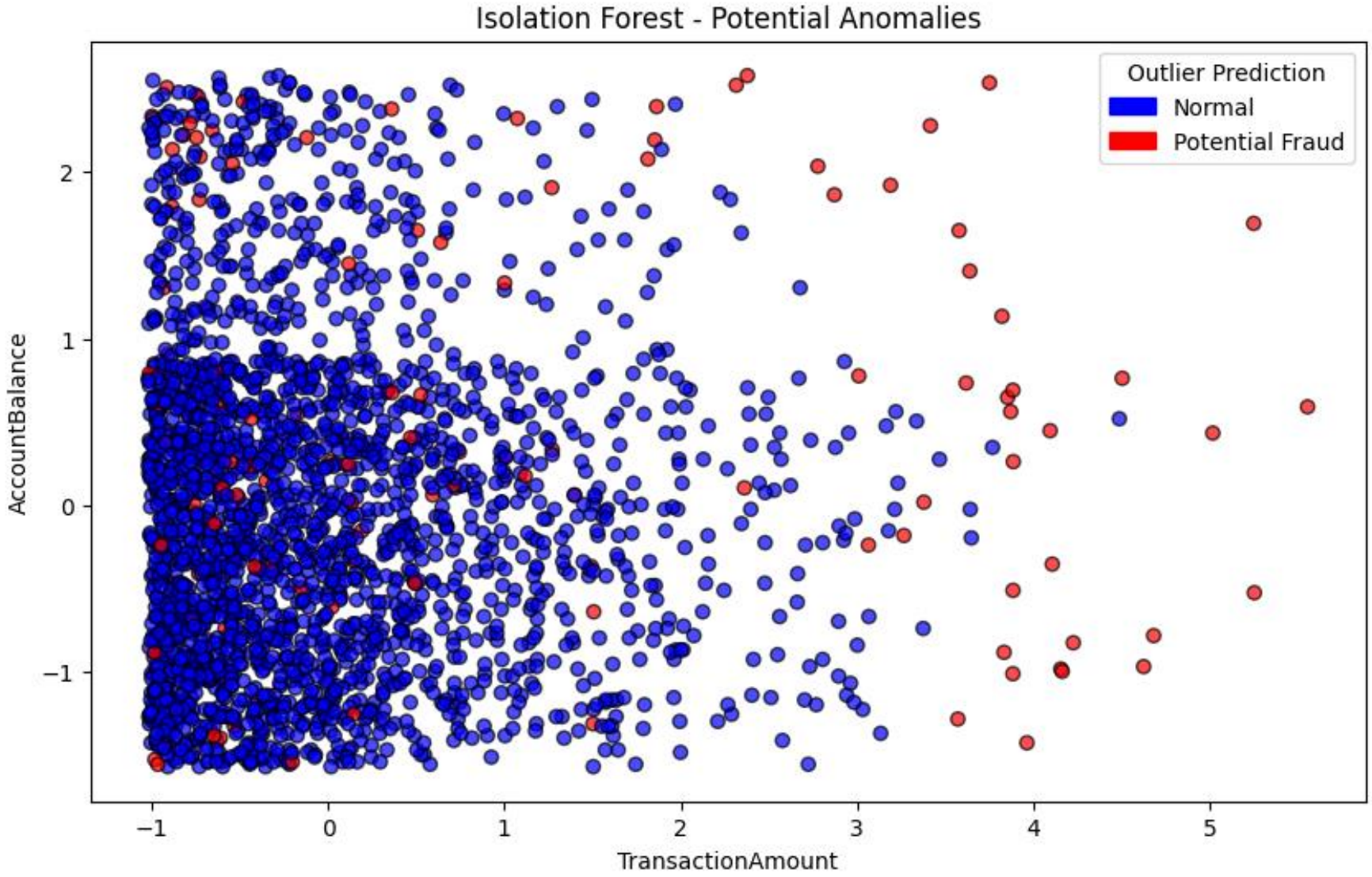
5 rows × 21 columns

Isolation Forest for Anomaly Detection

```
colors = np.where(df['IsAnomaly'] == -1, 'r', 'b')
# Visualize potential frauds (TransactionAmount vs AccountBalance)
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=colors, cmap='coolwarm', alpha=0.7,
    edgecolors='k', label='Data Points')
# Custom legend
import matplotlib.patches as mpatches
normal_patch = mpatches.Patch(color='b', label='Normal')
fraud_patch = mpatches.Patch(color='r', label='Potential Fraud')
plt.legend(handles=[normal_patch, fraud_patch], title='Outlier Prediction')

plt.title('Isolation Forest - Potential Anomalies')
plt.xlabel(features[0])  # TransactionAmount
plt.ylabel(features[2])  # AccountBalance
plt.show()
```

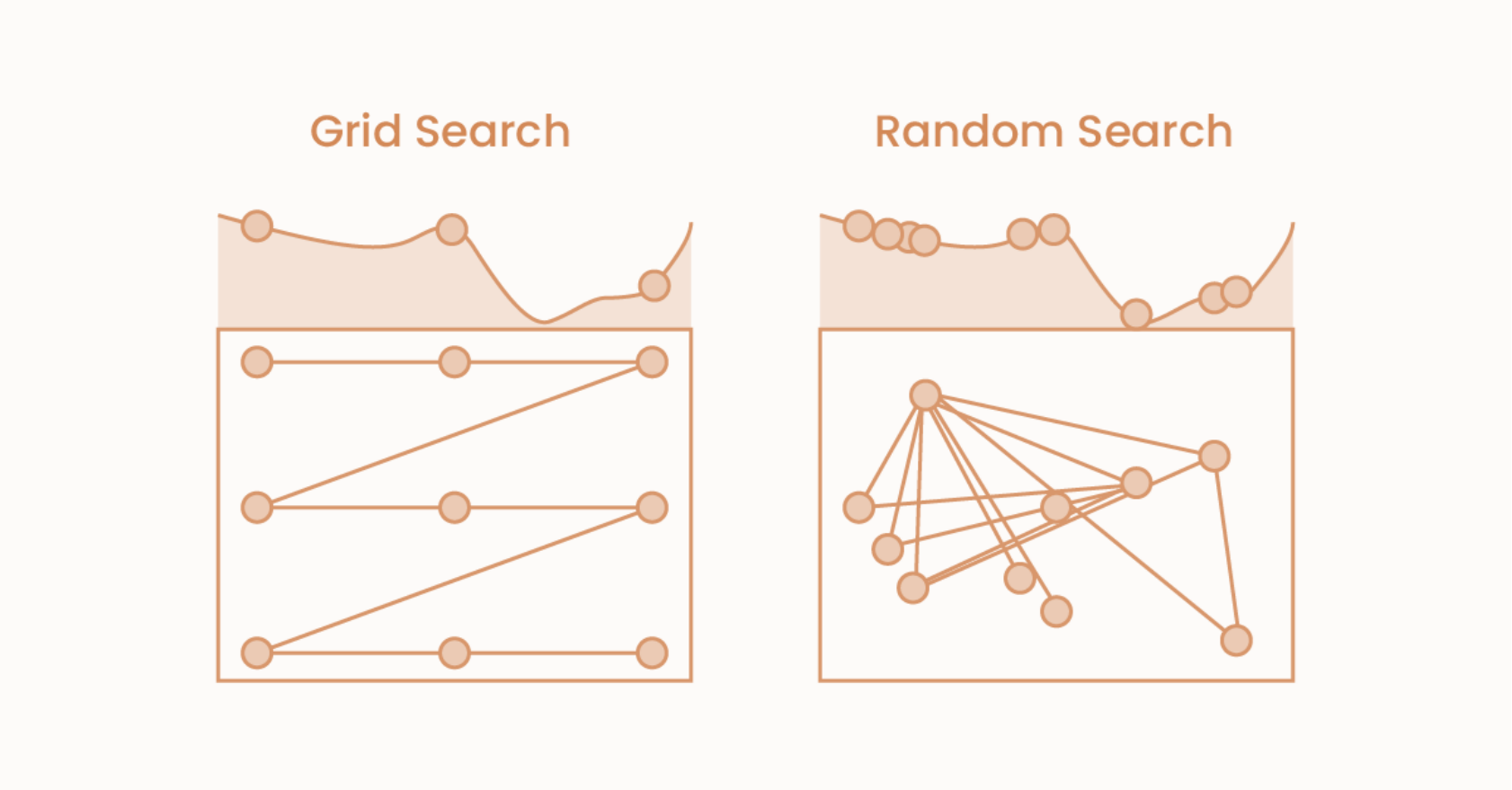


Isolation Forest for Anomaly Detection



Hyperparameters choosing for Machine Learning

`GridSearchCV and RandomizedSearchCV`



Hyperparameters choosing for Machine Learning

- GridSearchCV

```
grid.GridSearchCV(estimator='model's name', param_grid = 'tunning parameter')
grid.fit(x,y)
```

- RandomizedSearchCV

```
Random_model.RandomizedSearchCV(estimator='model's name', param_distribution
= 'tunning parameter', n_iter = 'number of iterations')
Random_model.fit(x,y)
```

Using GridSearchCV for tunning model

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
import time

x, y = make_classification(n_samples=1000, n_features=4,
                           n_classes=2, random_state=0)

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0)

time_start = time.time()
```

Using GridSearchCV for KNN model

```
knn = KNeighborsClassifier()
knn_params = {'n_neighbors': range(1, 50, 2)}

knn_grid = GridSearchCV(estimator=knn, param_grid=knn_params)
knn_grid.fit(x_train, y_train)

print('KNN Best Score:', knn_grid.best_score_)
print('KNN Best Params:', knn_grid.best_params_)
```

```
KNN Best Score: 0.9613333333333334
KNN Best Params: {'n_neighbors': 9}
```

Using GridSearchCV for Random Forest

```
forest = RandomForestClassifier(random_state=0)
forest_params = {
    'n_estimators': range(10, 100, 5),
    'criterion': ['entropy', 'gini'],
    #'max_depth': range(1, 80)
}
forest_grid = GridSearchCV(estimator=forest, param_grid=forest_params)
forest_grid.fit(x_train, y_train)

print ('Random Forest Best Score:', forest_grid.best_score_)
print ('Random Forest Best Params:', forest_grid.best_params_)
```

```
Random Forest Best Score: 0.9626666666666667
Random Forest Best Params: {'criterion': 'gini', 'n_estimators': 65}
```

Using GridSearchCV for SVM model

```
svm = SVC()
svm_params = [
    {'kernel': ['linear'], 'C': [0.1, 1, 5, 10, 50]},
    {'kernel': ['rbf'], 'C': [0.1, 1, 5, 10, 50], 'gamma': range(1, 10)},
    {'kernel': ['poly'], 'C': [0.1, 1, 5, 10, 50], 'degree': range(1, 10)},
]

svm_grid = GridSearchCV(estimator=svm, param_grid=svm_params)
svm_grid.fit(x_train, y_train)

print('SVM Best Score:', svm_grid.best_score_)
print('SVM Best Params:', svm_grid.best_params_)
```

```
SVM Best Score: 0.9653333333333334
SVM Best Params: {'C': 5, 'gamma': 3, 'kernel': 'rbf'}
```

Using GridSearchCV for tunning model

```
# Summary time

time_end = time.time()
print(f'Total Time: {time_end - time_start} - Second')
```

```
Total Time: 82.99333548545837 - Second
```

Using RandomizedSearchCV for tunning model

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV
import time

x, y = make_classification(n_samples=1000, n_features=4,
                           n_classes=2, random_state=1)

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0)
time_start = time.time()
```

Using RandomizedSearchCV for KNN model

```
# KNN model

knn = KNeighborsClassifier()
knn_params = {'n_neighbors': range(1, 50, 2)}
random_knn = RandomizedSearchCV(estimator=knn, param_distributions=knn_params)
random_knn.fit(x_train, y_train)

print('KNN best score: ', random_knn.best_score_)
print('KNN best params: ', random_knn.best_params_)
```

```
KNN best score:  0.868
KNN best params:  {'n_neighbors': 17}
```

Using RandomizedSearchCV for Random Forest

```
# Random Forest model
forest = RandomForestClassifier(random_state=0)
forest_params = {
    'n_estimators': range(10, 100, 5),
    'criterion': ['entropy', 'gini'],
    #'max_depth': range(1, 80)
}
random_forest = RandomizedSearchCV(estimator=forest, param_distributions=forest_params, n_iter=4)
random_forest.fit(x_train, y_train)

print('Random Forest best score: ', random_forest.best_score_)
print('Random Forest best params: ', random_forest.best_params_)
```

```
Random Forest best score:  0.8546666666666667
Random Forest best params:  {'n_estimators': 75, 'criterion': 'entropy'}
```

Using RandomizedSearchCV for SVM model

```
# SVM model

svm = SVC()
svm_params = [
    {'kernel': ['linear'], 'C': [0.1, 1, 5, 10, 50]},
    {'kernel': ['rbf'], 'C': [0.1, 1, 5, 10, 50], 'gamma': range(1, 10)},
    {'kernel': ['poly'], 'C': [0.1, 1, 5, 10, 50], 'degree': range(1, 10)},
]

random_svm = RandomizedSearchCV(estimator=svm, param_distributions=svm_params)
random_svm.fit(x_train, y_train)

print('SVM best score:', random_svm.best_score_)
print('SVM best params:', random_svm.best_params_)
```

```
SVM best score: 0.868
SVM best params: {'kernel': 'rbf', 'gamma': 4, 'C': 1}
```

Using RandomizedSearchCV for tunning model

```
# Summary time

time_end = time.time()
print(f'Total time: {time_end - time_start} second')
```

```
Total time: 4.21240758895874 second
```

Compare GridSearchCV vs. RandomizedSearchCV

```
Using GridSearchCV
KNN Best Score: 0.9613333333333334
KNN Best Params: {'n_neighbors': 9}

Random Forest Best Score: 0.9626666666666667
Random Forest Best Params: {'criterion': 'gini', 'n_estimators': 65}

SVM Best Score: 0.9653333333333334
SVM Best Params: {'C': 5, 'gamma': 3, 'kernel': 'rbf'}

Total Time: 58.11018347740173 - Second
```

```
Using RandomizedSearchCV
KNN best score:  0.868
KNN best params:  {'n_neighbors': 17}

Random Forest best score:  0.8573333333333333
Random Forest best params:  {'n_estimators': 90, 'criterion': 'gini'}

SVM best score: 0.8693333333333333
SVM best params: {'kernel': 'rbf', 'gamma': 6, 'C': 1}

Total time: 2.9451003074645996 second
```

Assignment

Objective:
This assignment aims to develop and evaluate a classification model for a given dataset. You will be assessed on your ability to select appropriate models, perform model evaluation, tune hyperparameters, and implement ensemble learning techniques.

1. **Data Exploration and Preparation:**
    1. Load and explore the dataset.
    2. Handle missing values (if any).
    3. Perform feature scaling or normalization (if necessary).
    4. Split the data into training and testing sets (e.g., 80% train, 20% test).
2. **Model Selection and Training:**
    1. Select **three** different classification models (e.g., Logistic Regression, Support Vector Machine, Decision Tree, Random Forest, KNN).
    2. Train each model on the training data using default parameters.
3. **Model Evaluation:**
    1. Evaluate the performance of each model on the testing set using the following metrics:
        1. Accuracy
        2. Precision
        3. Recall
        4. F1-score
        5. Confusion Matrix
    2. Present the results in a clear and concise manner (e.g., tables, plots).

Assignment

4. **Hyperparameter Tuning:**
    1. Select one of the three models that showed promising initial results.
    2. Use **GridSearchCV** or a similar technique to tune the hyperparameters of the chosen model.
    3. Evaluate the performance of the tuned model on the testing set.
5. **Ensemble Learning:**
    1. Implement an ensemble learning technique (e.g., Bagging, Boosting, Stacking) using the chosen model or a combination of models.
    2. Evaluate the performance of the ensemble model on the testing set.
6. **Analysis and Discussion:**
    1. Compare the performance of the different models (baseline, tuned, ensemble).
    2. Discuss the strengths and weaknesses of each model and the impact of hyperparameter tuning and ensemble learning.
    3. Analyze the confusion matrix and discuss the implications of different types of errors.
    4. Discuss any limitations or challenges encountered during the assignment.


SUT
SURANAREE
UNIVERSITY OF
TECHNOLOGY
INSTITUTE OF
ENGINEERING
Mecha tronics
Modern Automotive