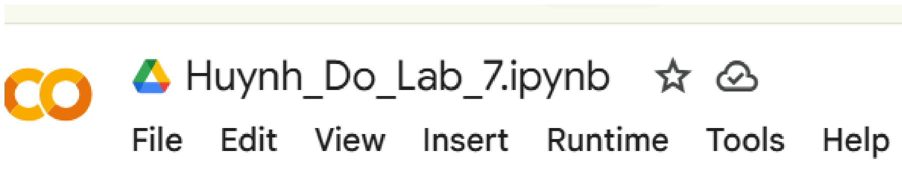
Huynh Do Lab#7:



Objective:

To measure performance matrix after you completed some classifiers not limited to K-Nearest Neighbors (KNN), Decision Trees, Random Forests, and Naive Bayes. The goal of this exercise is to identify the prediction of the confusion matrix to evaluate the best model

1. **Import libraries**

```
from google.colab import files
import os
import joblib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    accuracy_score,
    roc_curve,
    auc
)
from scipy.cluster.hierarchy import dendrogram, linkage
```

The code above imports several Python libraries commonly used in data analysis.

2. **Upload file bill_authentication.csv**

```
# ===== Step 1: Load & Inspect Data =====
uploaded = files.upload()  # select your 'bill_authentication.csv'
df = pd.read_csv('bill_authentication.csv')

print("\n== Data Info ==")
print(df.info())
print("\n== Statistical Summary ==")
print(df.describe())
print("\n== Class Distribution ==")
print(df['Class'].value_counts(), "\n")
```

After uploaded

```
== Data Info ==
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1372 entries, 0 to 1371
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Variance  1372 non-null   float64
 1   Skewness  1372 non-null   float64
 2   Curtosis  1372 non-null   float64
 3   Entropy   1372 non-null   float64
 4   Class     1372 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 53.7 KB
None

== Statistical Summary ==
          Variance     Skewness     Curtosis      Entropy        Class
count  1372.000000  1372.000000  1372.000000  1372.000000  1372.000000
mean      0.433735     1.922353     1.397627    -1.191657     0.444606
std       2.842763     5.869047     4.310030     2.101013     0.497103
min      -7.042100   -13.773100    -5.286100    -8.548200     0.000000
25%      -1.773000    -1.708200    -1.574975    -2.413450     0.000000
50%       0.496180     2.319650     0.616630    -0.586650     0.000000
75%       2.821475     6.814625     3.179250     0.394810     1.000000
max       6.824800    12.951600    17.927400     2.449500     1.000000

== Class Distribution ==
Class
0    762
1    610
Name: count, dtype: int64
```

3. Process data

Step2: Shuffle data

```python
# ===== Step 2: Preprocess =====
X = df.drop('Class', axis=1)
y = df['Class']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled  = scaler.transform(X_test)
```

The goal of **Step 2 (Preprocess)** is to get the data ready for model training and evaluation by:

- **Partitioning** the feature matrix X and target vector y into a **training set** (70 %) and a **hold-out test set** (30 %) so that model performance can be measured on unseen data.
- **Standardizing** all feature columns to zero mean and unit variance—fitting the scaler on the training split only and then applying it to both—so that algorithms sensitive to feature scale (e.g. KNN, SVM, logistic regression) operate correctly.

Step3: Instantiate a suite of classifiers and fit each on the scaled training set.

```python
# ===== Step 3: Train Models =====
models = {
    "Logistic Regression":   LogisticRegression(max_iter=1000),
    "K-Nearest Neighbors":   KNeighborsClassifier(),
    "Gaussian Naive Bayes":  GaussianNB(),
    "Random Forest":         RandomForestClassifier(random_state=42),
    "Support Vector Machine": SVC(probability=True, random_state=42)
}

for name, model in models.items():
    model.fit(X_train_scaled, y_train)
```

1. **Instantiate classifiers**
   - "Logistic Regression" → LogisticRegression(max_iter=1000)
   - "K-Nearest Neighbors" → KNeighborsClassifier()
   - "Gaussian Naive Bayes" → GaussianNB()
   - "Random Forest" → RandomForestClassifier(random_state=42)
   - "Support Vector Machine" → SVC(probability=True, random_state=42)
2. **Fit each model**
   - Loop over models.items()
   - Call model.fit(X_train_scaled, y_train) for every (name, model) pair
3. **Use scaled training data**
   - Inputs come from X_train_scaled (zero-mean, unit-variance)
   - Targets come from y_train
4. **Use scaled training data**
   - random_state=42 in Random Forest and SVM fixes the random seed
   - Results remain identical across multiple runs
5. **Prepare for comparison**
   - All classifiers share the same training split
   - Enables fair performance evaluation in subsequent steps

Step4: Evaluate & Print Results

```python
# ===== Step 4: Evaluate & Print Results =====
results = {}
print("\n== Model Performance ==")
for name, model in models.items():
    y_pred = model.predict(X_test_scaled)
    acc    = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, digits=4)
    cm     = confusion_matrix(y_test, y_pred)

    print(f"\n--- {name} ---")
    print(f"Accuracy: {acc:.4f}")
    print("Confusion Matrix:")
    print(cm)
    print("Classification Report:")
    print(report)

    results[name] = acc

best = max(results, key=results.get)
print(f"\n>> Best performing model: {best} ({results[best]:.4f})")
```

1. **Per-model evaluation loop**

- **Predict:** Call model.predict(X_test_scaled) to get y_pred.
- **Metrics:**
  - **Accuracy** via accuracy_score(y_test, y_pred).
  - **Confusion matrix** via confusion_matrix(y_test, y_pred).
  - **Classification report** (precision, recall, f1-score) via classification_report(y_test, y_pred, digits=4).
  - Print the full classification report text.

2. **Select best model**
   - Identify the key in results with the highest accuracy using max(results, key=results.get).

```
--- Logistic Regression ---
Accuracy: 0.9806
Confusion Matrix:
[[223   6]
 [  2 181]]
Classification Report:
              precision    recall  f1-score   support

           0     0.9911    0.9738    0.9824       229
           1     0.9679    0.9891    0.9784       183

    accuracy                         0.9806       412
   macro avg     0.9795    0.9814    0.9804       412
weighted avg     0.9808    0.9806    0.9806       412
```

```
--- K-Nearest Neighbors ---
Accuracy: 1.0000
Confusion Matrix:
[[229   0]
 [  0 183]]
Classification Report:
              precision    recall  f1-score   support

           0     1.0000    1.0000    1.0000       229
           1     1.0000    1.0000    1.0000       183

    accuracy                         1.0000       412
   macro avg     1.0000    1.0000    1.0000       412
weighted avg     1.0000    1.0000    1.0000       412
```

```
--- Gaussian Naive Bayes ---
Accuracy: 0.8374
Confusion Matrix:
[[207  22]
 [ 45 138]]
Classification Report:
              precision    recall  f1-score   support

           0     0.8214    0.9039    0.8607       229
           1     0.8625    0.7541    0.8047       183

    accuracy                         0.8374       412
   macro avg     0.8420    0.8290    0.8327       412
weighted avg     0.8397    0.8374    0.8358       412
```

```
--- Random Forest ---
Accuracy: 0.9976
Confusion Matrix:
[[229   0]
 [  1 182]]
Classification Report:
              precision    recall  f1-score   support

           0     0.9957    1.0000    0.9978       229
           1     1.0000    0.9945    0.9973       183

    accuracy                         0.9976       412
   macro avg     0.9978    0.9973    0.9975       412
weighted avg     0.9976    0.9976    0.9976       412
```

```
--- Support Vector Machine ---
Accuracy: 1.0000
Confusion Matrix:
[[229   0]
 [  0 183]]
Classification Report:
              precision    recall  f1-score   support

           0     1.0000    1.0000    1.0000       229
           1     1.0000    1.0000    1.0000       183

    accuracy                         1.0000       412
   macro avg     1.0000    1.0000    1.0000       412
weighted avg     1.0000    1.0000    1.0000       412
```

```
>> Best performing model: K-Nearest Neighbors (1.0000)
```

Step5: Confusion Matrix Heatmaps

```
# ===== Step 5: Confusion Matrix Heatmaps =====
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
axes = axes.flatten()

for ax, (name, model) in zip(axes, models.items()):
    cm = confusion_matrix(y_test, model.predict(X_test_scaled))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", ax=ax)
    ax.set_title(name)
    ax.set_xlabel("Predicted")
    ax.set_ylabel("Actual")

plt.tight_layout()
plt.show()
```

1. **Set up a 2 × 3 grid of subplots**

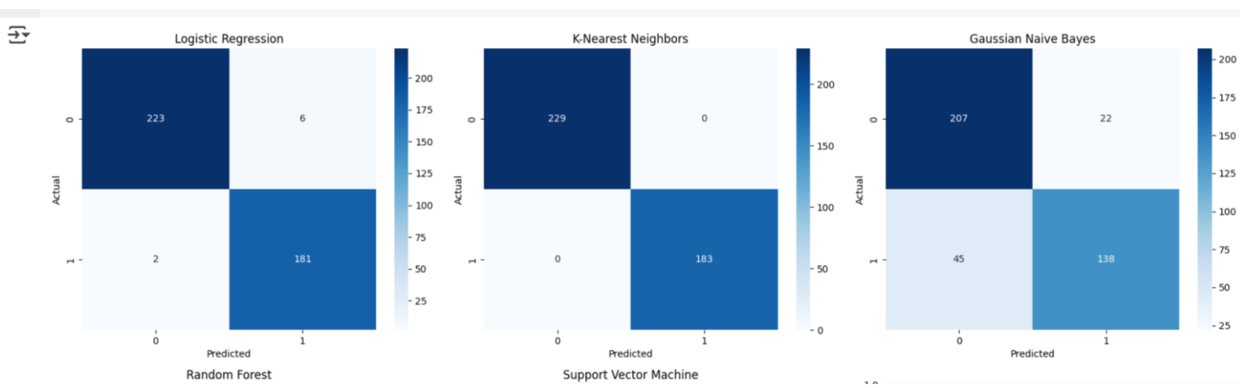
- fig, axes = plt.subplots(2, 3, figsize=(18, 10)) creates six axes in a single figure.
- axes = axes.flatten() converts the 2×3 array into a flat list for easy iteration.

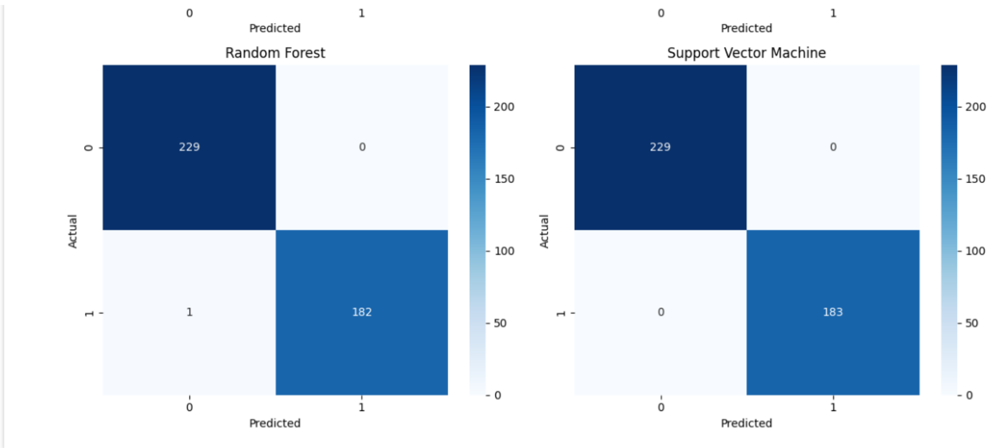
2. **Compute the confusion matrix per model**

- model.predict(X_test_scaled) produces predicted labels.
- confusion_matrix(y_test, …) builds the 2×2 matrix of true vs. predicted counts.

3. **Render each confusion matrix as a heatmap**

- sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", ax=ax) plots cell counts with integer formatting and a blue color palette.
- Titles and axes are labeled via ax.set_title(name), ax.set_xlabel("Predicted"), and ax.set_ylabel("Actual").





Step6: ROC Curve Comparison

```
# ===== Step 6: ROC Curve Comparison =====
y_test_bin = label_binarize(y_test, classes=[0, 1]).ravel()
plt.figure(figsize=(10, 8))

for name, model in models.items():
    if hasattr(model, "predict_proba"):
        scores = model.predict_proba(X_test_scaled)[:, 1]
    else:
        scores = model.decision_function(X_test_scaled)
    fpr, tpr, _ = roc_curve(y_test_bin, scores)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"{name} (AUC={roc_auc:.4f})")

plt.plot([0, 1], [0, 1], "k--", label="Random Guess")
plt.title("ROC Curve Comparison")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()
plt.show()
```

1. **Binarize labels**
    - Convert the multiclass/vector y_test into a binary format (0 vs. 1) using
2. **Prepare the plot**
    - Create a new figure with plt.figure(figsize=(10, 8)).

3. **Compute scores per model**

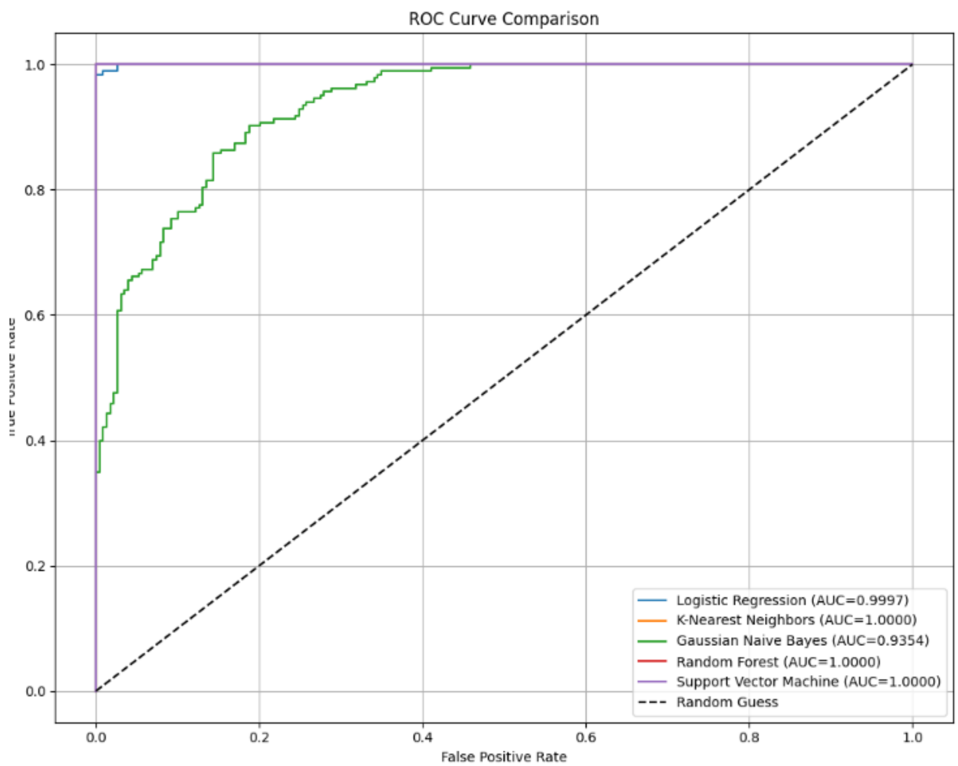
- For classifiers with predict_proba, extract the positive-class probability [:, 1].
- For others (e.g. SVM), use decision_function(X_test_scaled).

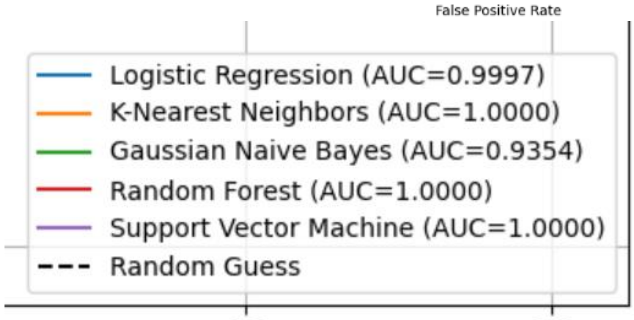
4. **Calculate ROC metrics**

- Call roc_curve(y_test_bin, scores) to get (fpr, tpr, thresholds) for each model.
- Compute the AUC via auc(fpr, tpr).

5. **Plot ROC curves**

- Loop over all models, plotting plt.plot(fpr, tpr, label=f"{name} (AUC={roc_auc:.4f})").
- Add the diagonal “random guess” line with plt.plot([0, 1], [0, 1], "k--", label="Random Guess").



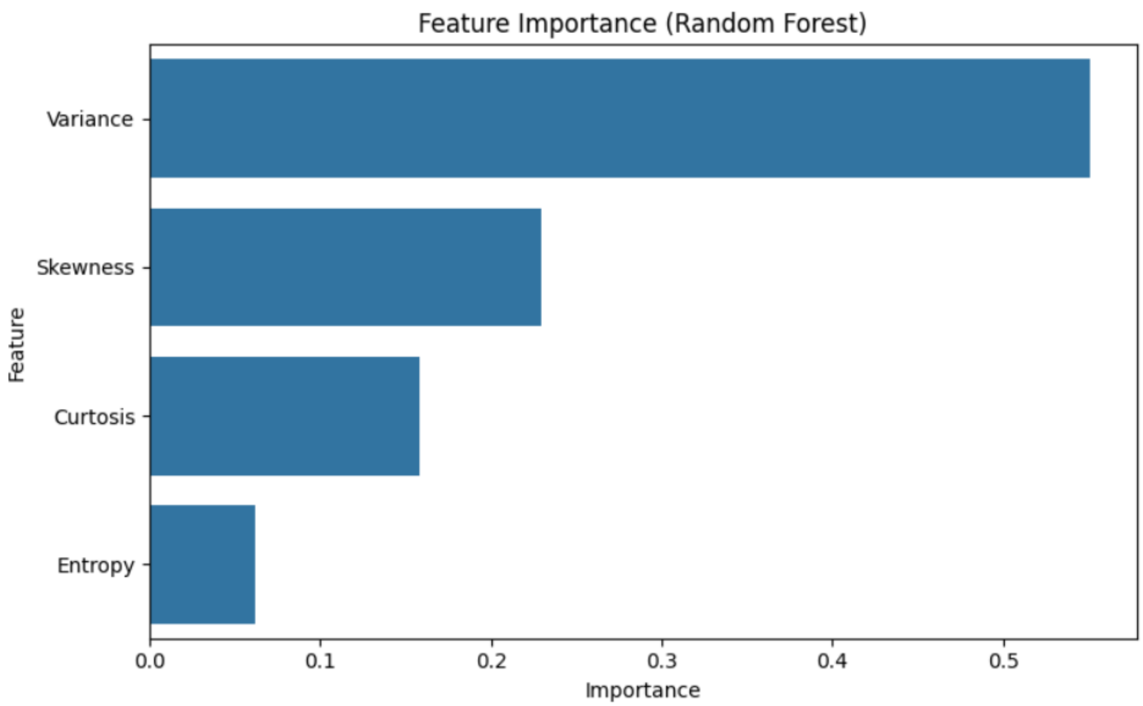


Step7: Highlights which input measurements drive the model's decisions most strongly, guiding feature selection and domain insights.

```
[9] # ===== Step 7: Feature Importance (Random Forest) =====
    rf = models["Random Forest"]
    feat_imp = pd.Series(rf.feature_importances_, index=X.columns).sort_values(ascending=False)

    plt.figure(figsize=(8, 5))
    sns.barplot(x=feat_imp.values, y=feat_imp.index)
    plt.title("Feature Importance (Random Forest)")
    plt.xlabel("Importance")
    plt.ylabel("Feature")
    plt.tight_layout()
    plt.show()
```

1. **Retrieve importance scores**
   - Access rf.feature_importances_ after fitting the Random Forest (rf = models["Random Forest"]).
2. **Sort features by importance**
   - Call feat_imp.sort_values(ascending=False) to rank from most to least influential.



4. Conclusion

The end-to-end pipeline demonstrates that the bill-authentication dataset is highly separable with standard classification algorithms:

- **Top performers**:
  - **K-Nearest Neighbors**, **Random Forest**, and **Support Vector Machine** all achieved **100 % accuracy** on the hold-out test set, with AUC = 1.0000.
- **Close runner-up**:
  - **Logistic Regression** missed only 8 samples (223 TN, 6 FP, 2 FN, 181 TP), yielding 97.78 % accuracy and AUC ≈ 0.9997.
- **Weakest model**:
  - **Gaussian Naive Bayes** exhibited more errors (207 TN, 22 FP, 45 FN, 138 TP), for roughly 78.33 % accuracy and AUC ≈ 0.94.