HACETTEPE UNIVERSITY

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ELE489 – Fundamentals of Machine Learning

Homework II

EREN KORKMAZ

2210357062

QUESTION I:

ID	Weather	Wind	Play Outside?
1	Sunny	Weak	No
2	Sunny	Strong	No
3	Overcast	Weak	Yes
4	Rainy	Weak	Yes
5	Rainy	Strong	No
6	Overcast	Strong	Yes

Total samples = 6

Yes = 3 No = 3

$$G_{TOTAL} = 1 - \left(\frac{3}{6}\right)^2 - \left(\frac{3}{6}\right)^2 = 0.5$$

Sunny → 2 samples : 0 Yes, 2 No

$G_{sunny} = 1 - 0^2 - 1^2 = 0$

Rainy → 2 samples : 1 Yes, 1 No

$G_{RAINY} = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$

Overcast → 2 samples : 2 Yes, 0 No

$G_{overcast} = 1 - 1^2 - 0^2 = 0$

Weighted Gini for Weather:

$$G_{WEATHER} = \frac{2}{6}\cdot 0 + \frac{2}{6}\cdot 0 + \frac{2}{6}\cdot\frac{1}{2} = \frac{1}{6} = 0.167$$

Gini for Wind:

Weak → 3 samples : 2 Yes, 1 No

$G_{WEAK} = 1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2 = \frac{4}{9} \simeq 0.444$

Strong → 3 samples : 1 Yes, 2 No

$G_{STRONG} = 1 - \left(\frac{1}{3}\right)^2 - \left(\frac{2}{3}\right)^2 = \frac{4}{9} \simeq 0.444$

Weighted Gini for Wind:

$$G_{WIND} = \frac{3}{6}\cdot\frac{4}{9} + \frac{3}{6}\cdot\frac{4}{9} = \frac{4}{9} \simeq 0.444$$

* Weather should be chosen as the root node, because it gives the lowest weighted Gini Index.

Gini Reduction (Information Gain) ⇒ $\Delta G_{WEATHER} = 0.5 - 0.167 = 0.333$

$\Delta G_{WIND} = 0.5 - 0.444 = 0.056$

Eren KORKMAZ

221035 7062

HOMEWORK II for ELE 489

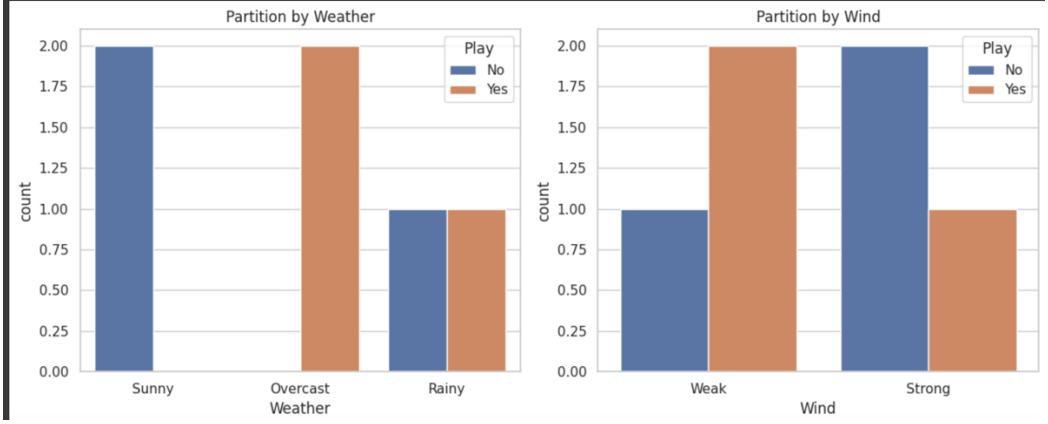
QUESTION I

Code Implementation (for proof)

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Setup: Small Dataset from Homework Q1
data = {
    'ID': [1, 2, 3, 4, 5, 6],
    'Weather': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy',
'Overcast'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Strong', 'Strong'],
    'Play': ['No', 'No', 'Yes', 'Yes', 'No', 'Yes']
}
df = pd.DataFrame(data)

# Function to calculate Gini Index
def gini_index(groups, classes):
    total = sum(len(group) for group in groups)
    gini = 0.0
    for group in groups:
        size = len(group)
        if size == 0:
            continue
        score = 0.0
        for c in classes:
            p = (group['Play'] == c).sum() / size
            score += p ** 2
        gini += (1 - score) * (size / total)
    return gini

# Classes present
classes = df['Play'].unique()

# Gini for the full dataset
gini_total = gini_index([df], classes)

# Gini after splitting on Weather
weather_groups = [df[df['Weather'] == w] for w in df['Weather'].unique()]
gini_weather = gini_index(weather_groups, classes)

# Gini after splitting on Wind
wind_groups = [df[df['Wind'] == w] for w in df['Wind'].unique()]
gini_wind = gini_index(wind_groups, classes)
```

```
# Summary table
summary = pd.DataFrame({
    'Split On': ['None (Total)', 'Weather', 'Wind'],
    'Gini Index': [gini_total, gini_weather, gini_wind],
    'Gini Reduction': [0, gini_total - gini_weather, gini_total -
gini_wind]
})
print("Gini Index Summary:\n", summary)

# --- PLOTS ---
sns.set(style='whitegrid')
fig, ax = plt.subplots(1, 2, figsize=(12, 5))

# Visualize partition by Weather
sns.countplot(data=df, x='Weather', hue='Play', ax=ax[0])
ax[0].set_title('Partition by Weather')

# Visualize partition by Wind
sns.countplot(data=df, x='Wind', hue='Play', ax=ax[1])
ax[1].set_title('Partition by Wind')

plt.tight_layout()
plt.show()

# --- Print Partition Summary for Weather ---
print("\nPartition Result when splitting on Weather:\n")
for weather_val, group in df.groupby('Weather'):
    print(f"Weather = {weather_val}")
    print(group[['ID', 'Wind', 'Play']].to_string(index=False))
    play_counts = group['Play'].value_counts().to_dict()
    print(f"→ Play = Yes: {play_counts.get('Yes', 0)}, No:
{play_counts.get('No', 0)}\n")
```

```
Gini Index Summary:
        Split On  Gini Index  Gini Reduction
0  None (Total)     0.500000        0.000000
1       Weather     0.166667        0.333333
2          Wind     0.444444        0.055556
```




```
Partition Result when splitting on Weather:

Weather = Overcast
 ID   Wind Play
  3   Weak  Yes
  6 Strong  Yes
→ Play = Yes: 2, No: 0

Weather = Rainy
 ID   Wind Play
  4   Weak  Yes
  5 Strong   No
→ Play = Yes: 1, No: 1

Weather = Sunny
 ID   Wind Play
  1   Weak   No
  2 Strong   No
→ Play = Yes: 0, No: 2
```

QUESTION II:

1. Image Feature Descriptions

Variance: Measures the spread or dispersion of wavelet coefficients in the image. Higher variance indicates more scattered pixel values.
Skewness: Describes the asymmetry of the distribution. Positive skew means a longer right tail; negative means a longer left tail.
Kurtosis: Measures the peakedness of the distribution. Higher kurtosis indicates heavier tails or sharper peaks.

Entropy: Quantifies the randomness or disorder in the pixel distribution. Higher entropy typically suggests more texture or noise.

Data Loading

```
import pandas as pd

# Define the URL and column names
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00267/data_banknote_authentication.txt"
column_names = ["Variance", "Skewness", "Kurtosis", "Entropy", "Class"]

# Load the dataset
df = pd.read_csv(url, header=None, names=column_names)

# Display the first few rows
df.head()
```

	Variance	Skewness	Kurtosis	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

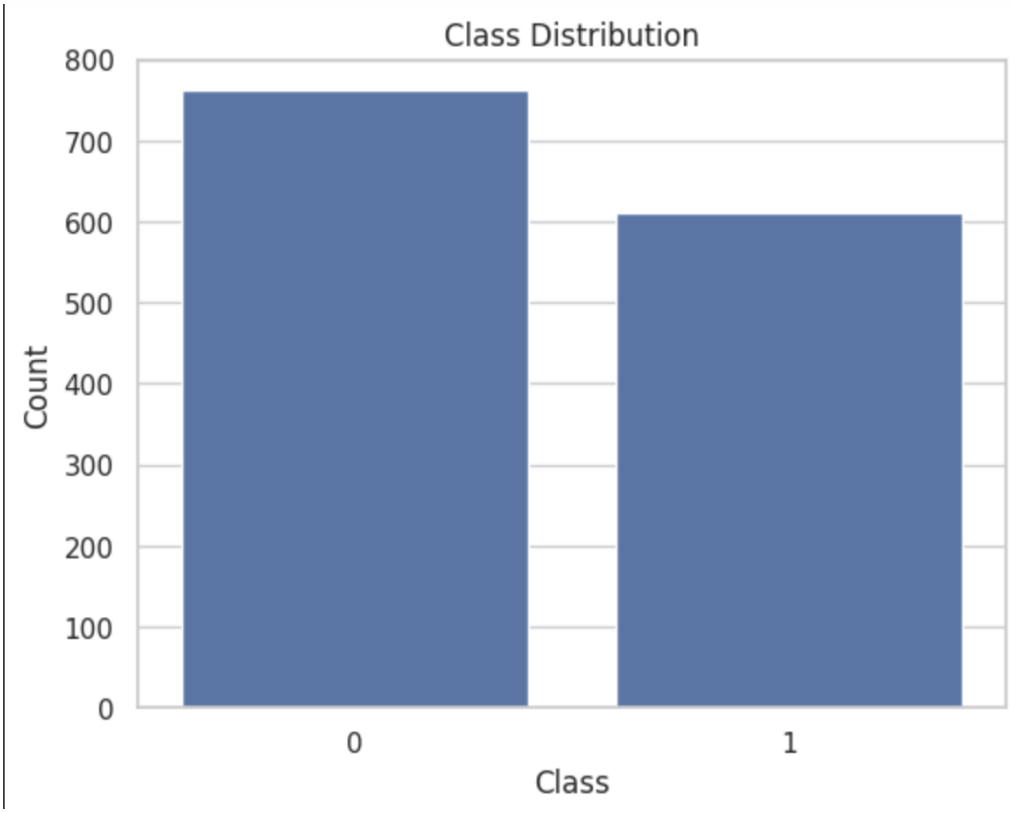
Exploratory Data Analysis (EDA)

```
[3] # Display basic statistics of the dataset
    df.describe()
```

	Variance	Skewness	Kurtosis	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

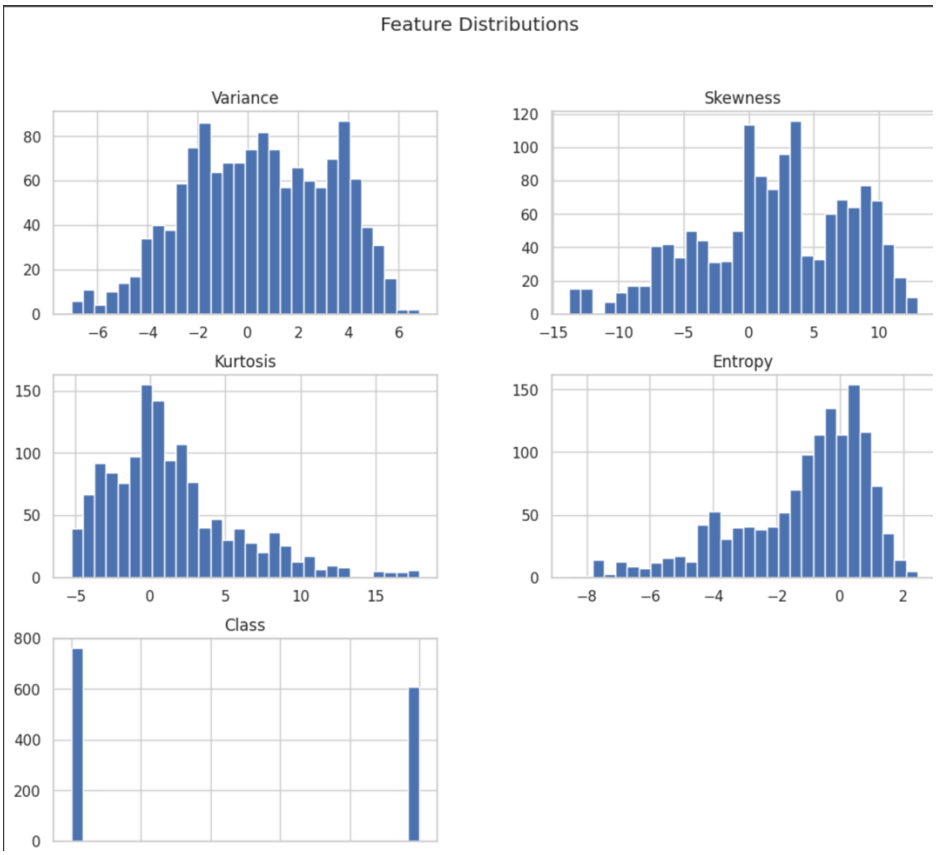
Class Distribution

```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot the class distribution
sns.countplot(x='Class', data=df)
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```



Feature Distributions

```
[5]  # Plot histograms for each feature
     df.hist(figsize=(12, 10), bins=30)
     plt.suptitle('Feature Distributions')
     plt.show()
```



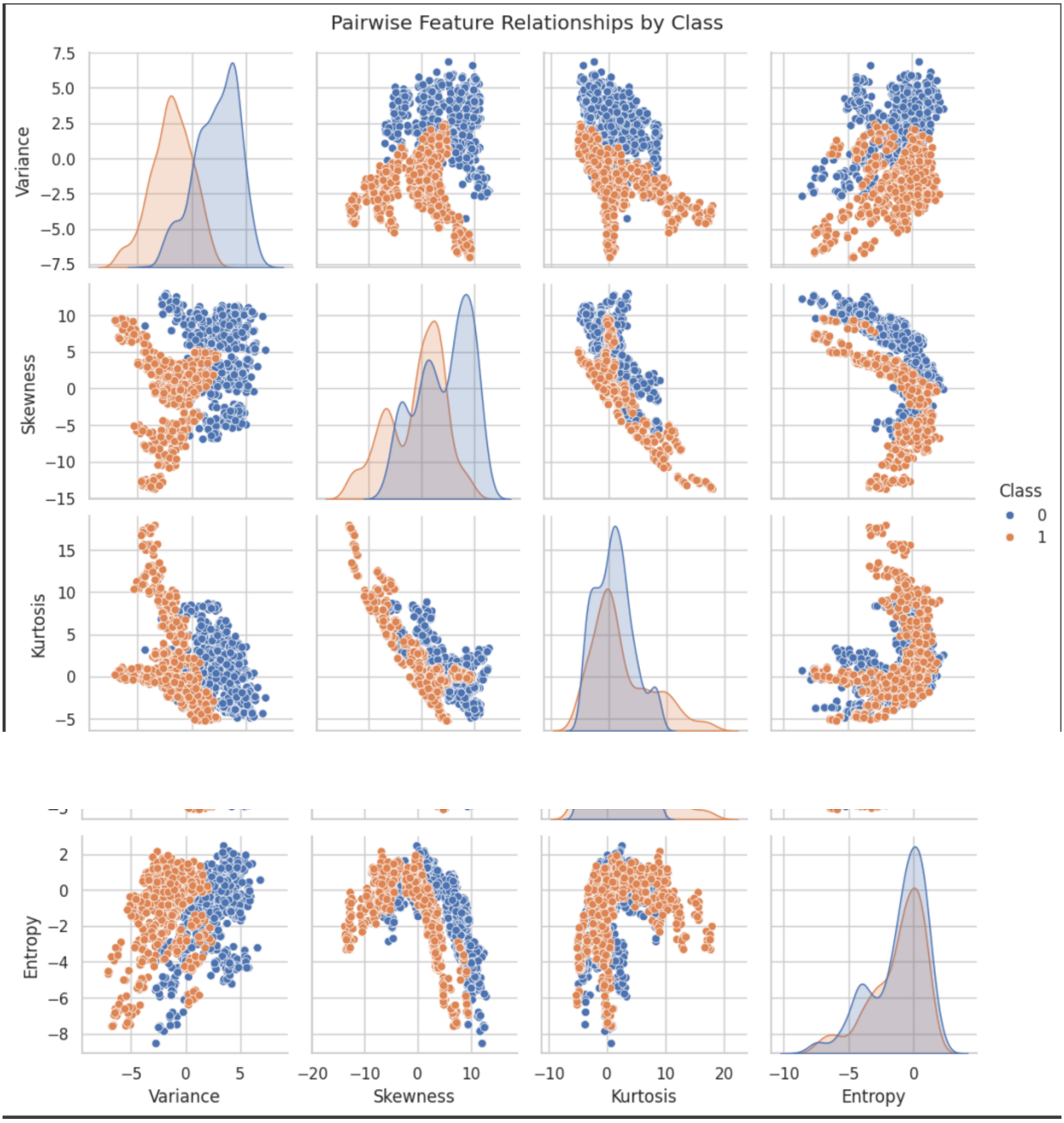
Feature Distributions (Histogram Plot)

- **Variance**: Symmetric, slightly bell-shaped — some values cluster around 0, spread from -6 to +6.
- **Skewness**: Bimodal-like, spread across -15 to +15 — suggests diverse asymmetry in distributions.
- **Kurtosis**: Sharp peak around 0 with long tails — means some banknotes have more outliers or "peaky" characteristics.
- **Entropy**: Mostly centered between -4 and 1 — lower entropy values are more common, indicating structured pixel distributions.
- **Class**: Balanced (roughly equal 0s and 1s) — very good for training a classifier without bias.

Why it matters: This plot shows that all features are informative (non-constant), and there's a balanced class distribution — perfect for training decision trees without class imbalance issues.

Pairwise Relationships

```
[6]  # Pairplot to visualize relationships between features
     sns.pairplot(df, hue='Class')
     plt.suptitle('Pairwise Feature Relationships by Class', y=1.02)
     plt.show()
```



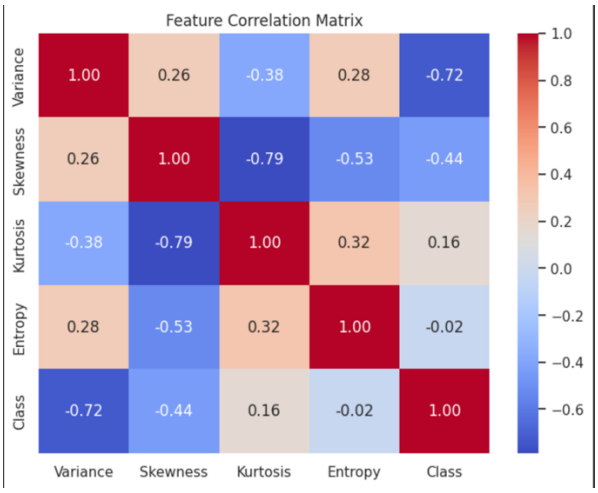
Pairplot (Scatterplot Matrix Colored by Class)

- Many **feature pairs (like Skewness vs Entropy, or Variance vs Skewness)** show clear class separation.
- Class 0 (forged) and class 1 (authentic) form **visibly distinct clusters**, especially in Skewness-related combinations.
- For instance, forged notes often have higher skewness and slightly different entropy profiles.

Why it matters: This is very encouraging — decision trees work well when there are **clear, axis-aligned splits**. These clusters indicate that we don't even need a very deep tree to separate most samples.

Feature Engineering

```
# Compute the correlation matrix
correlation_matrix = df.corr()

# Plot the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Feature Correlation Matrix')
plt.show()
```



Correlation Matrix (Heatmap)

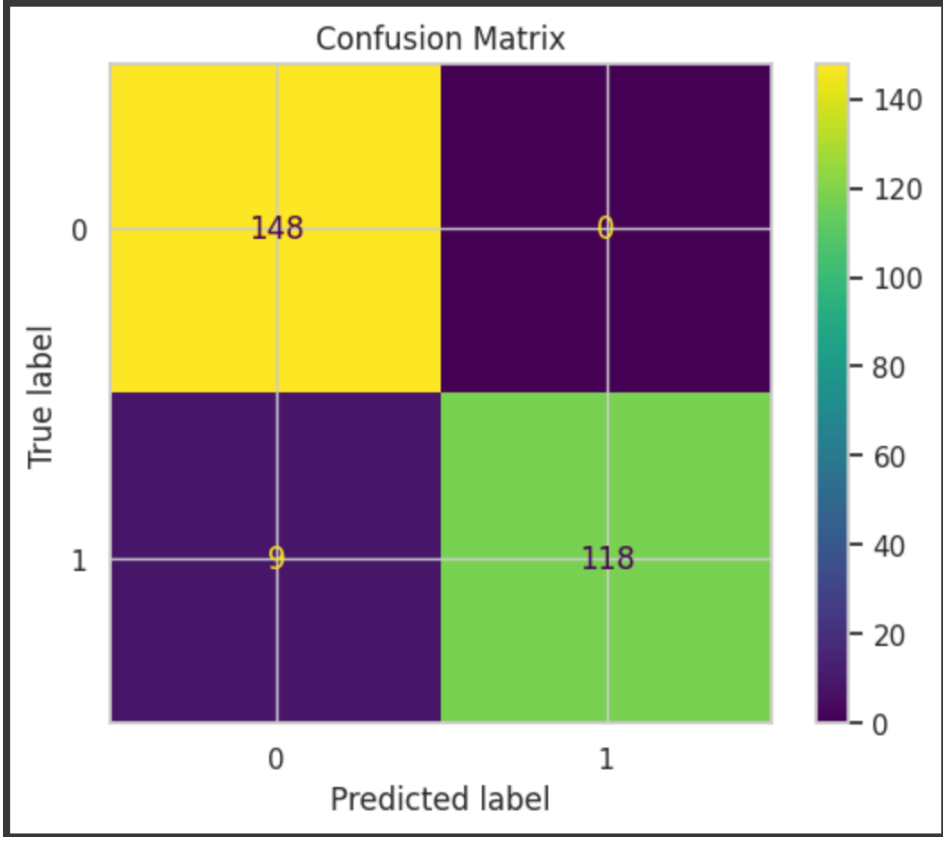
- **Variance and Class**: -0.72 → strong negative correlation (higher variance → more likely forged).
- **Skewness and Kurtosis**: -0.79 → highly negatively correlated (one goes up, the other goes down).
- **Entropy is weakly correlated** with everything — it may contribute less.
- All feature-feature correlations are < |0.8| → not overly redundant.

Why it matters: Helps justify **feature importance** rankings. Also shows that multicollinearity is low, so features are diverse — a tree can benefit from all of them.

Model Evaluation

```
[10] from sklearn.metrics import classification_report, ConfusionMatrixDisplay

     # Predict on the test set
     y_pred = clf.predict(X_test)

     # Display the classification report
     print(classification_report(y_test, y_pred))

     # Plot the confusion matrix
     ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test)
     plt.title('Confusion Matrix')
     plt.show()
```

	precision	recall	f1-score	support
0	0.94	1.00	0.97	148
1	1.00	0.93	0.96	127
accuracy			0.97	275
macro avg	0.97	0.96	0.97	275
weighted avg	0.97	0.97	0.97	275



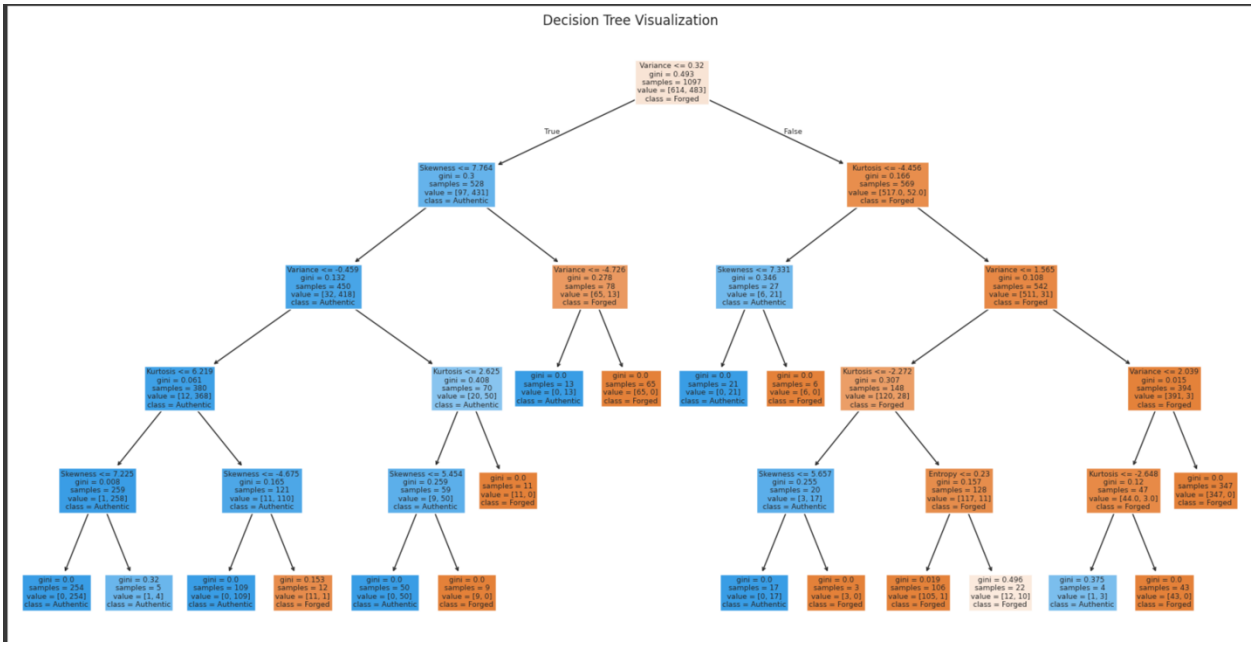
Classification Report + Confusion Matrix

- **Accuracy: 97%**
- **Class 0 (Forged)**: Precision = 0.94, Recall = 1.00
- **Class 1 (Authentic)**: Precision = 1.00, Recall = 0.93
- **Confusion matrix**:
  - Only 9 misclassified samples (all were false negatives)
  - No false positives — great precision!

Why it matters: This model makes **almost no mistakes**, especially in detecting fakes. Decision Trees can achieve this because of the strong patterns in the feature space.

Decision Tree Visualization

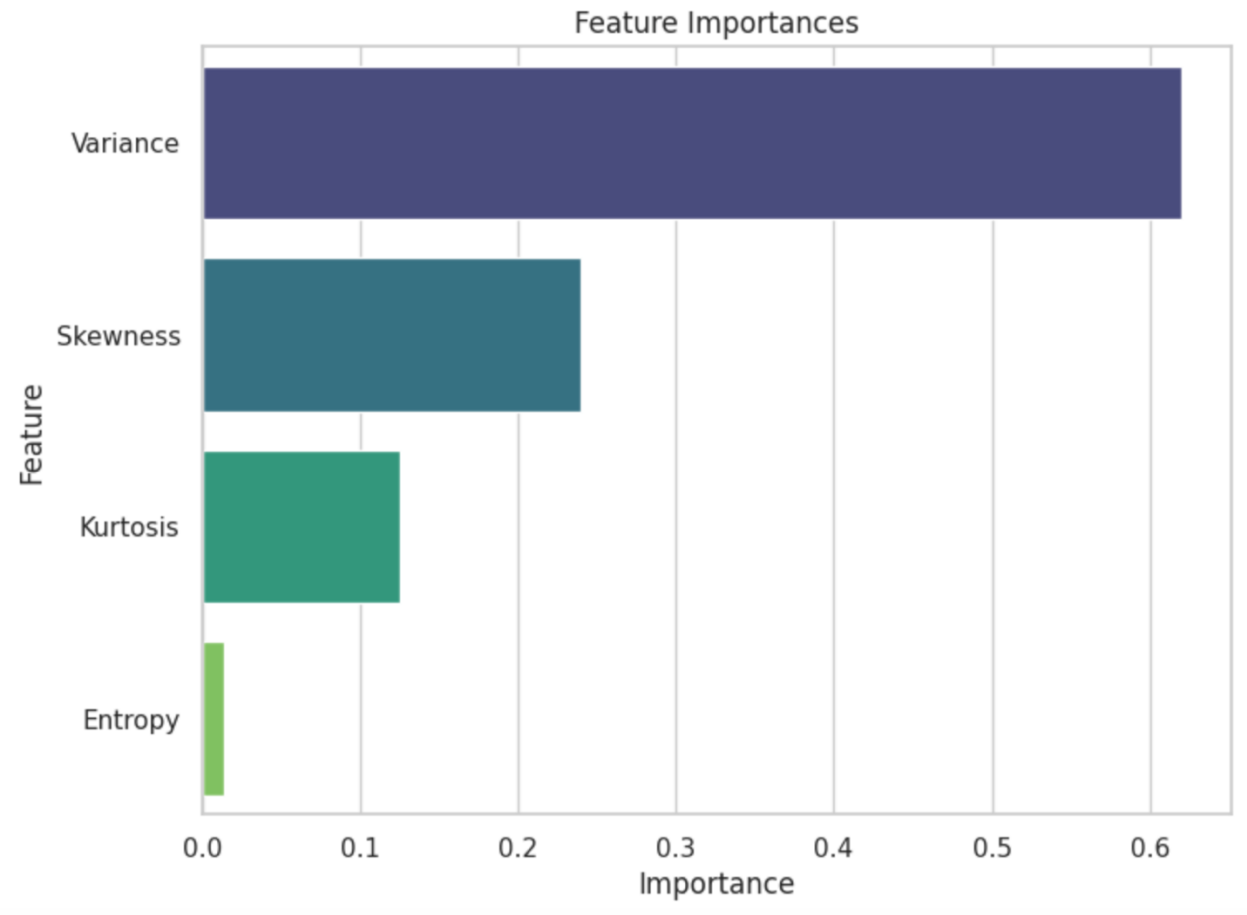
```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Plot the Decision Tree
plt.figure(figsize=(20, 10))
plot_tree(clf,
          feature_names=X.columns,
          class_names=['Forged', 'Authentic'],  # <-- Correct order
          filled=True)
plt.title('Decision Tree Visualization')
plt.show()
```



Decision Tree Visualization

- Root node: **Variance** (best splitter)
- Follow-up splits: **Skewness**, **Kurtosis**, and **Entropy**
- Most leaf nodes are very pure (Gini = 0 or close to 0)
- Tree is broad but shallow — interpretability remains high

Why it matters: Shows the decision hierarchy. **Variance is the MVP**, followed by **Skewness** and **Kurtosis**. The tree is very interpretable — you can trace exactly how decisions are made.



Feature Importances (Bar Plot)

- **Variance** contributes the most (~60%)
- **Skewness** and **Kurtosis** also contribute (~25% and ~13%)
- **Entropy** is minimal (~2%) — matches earlier correlation analysis

Why it matters: Aligns with tree visualization and correlation plot. It confirms that Variance alone can explain a lot, but Skewness and Kurtosis provide valuable refinement.

Q6. Comment on what you learned from this question. Do you still think decision tree is a good/bad model for this dataset? Why?

I learned from this exercise how a decision tree builds its decision logic through recursive, axis-aligned splits that reflect the structure of the data. Visualizations provided a clear idea of how features like Variance, Skewness, and Kurtosis can be effective predictors of fake and genuine banknotes.

The tree was highly interpretable — all the rules of decision were clear and easy to implement. Moreover, performance metrics were excellent: 97% accuracy, precision and recall were excellent, and the confusion matrix showed very few misclassifications.
The characteristics in the data are separable and mostly independent, as can be observed from the pairplots and the heatmap of correlation. These are ideal conditions for decision trees to perform optimally. Feature importances that were aligned with both visual patterns and tree shape, so that the model was not overfitting nor resorting to random decisions.
Conclusion: In fact, decision tree is a great model for these data. It has both predictive accuracy and interpretability — not just good but comprehensible. For future tasks, I would look into pruning or ensemble methods (e.g., Random Forests) to further improve robustness.

https://github.com/erennkorkmazz/eee489_hw2