HACETTEPE UNIVERSITY

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ELE489 – Fundamentals of Machine Learning

–Homework II–

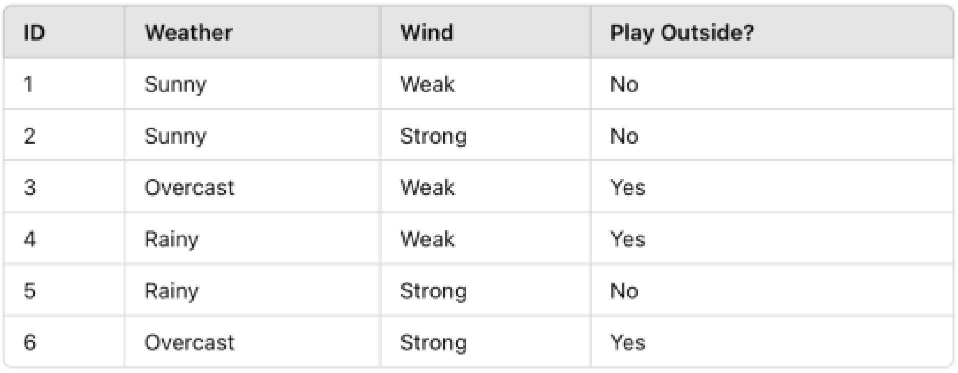
Murat DOĞAN

2200357075

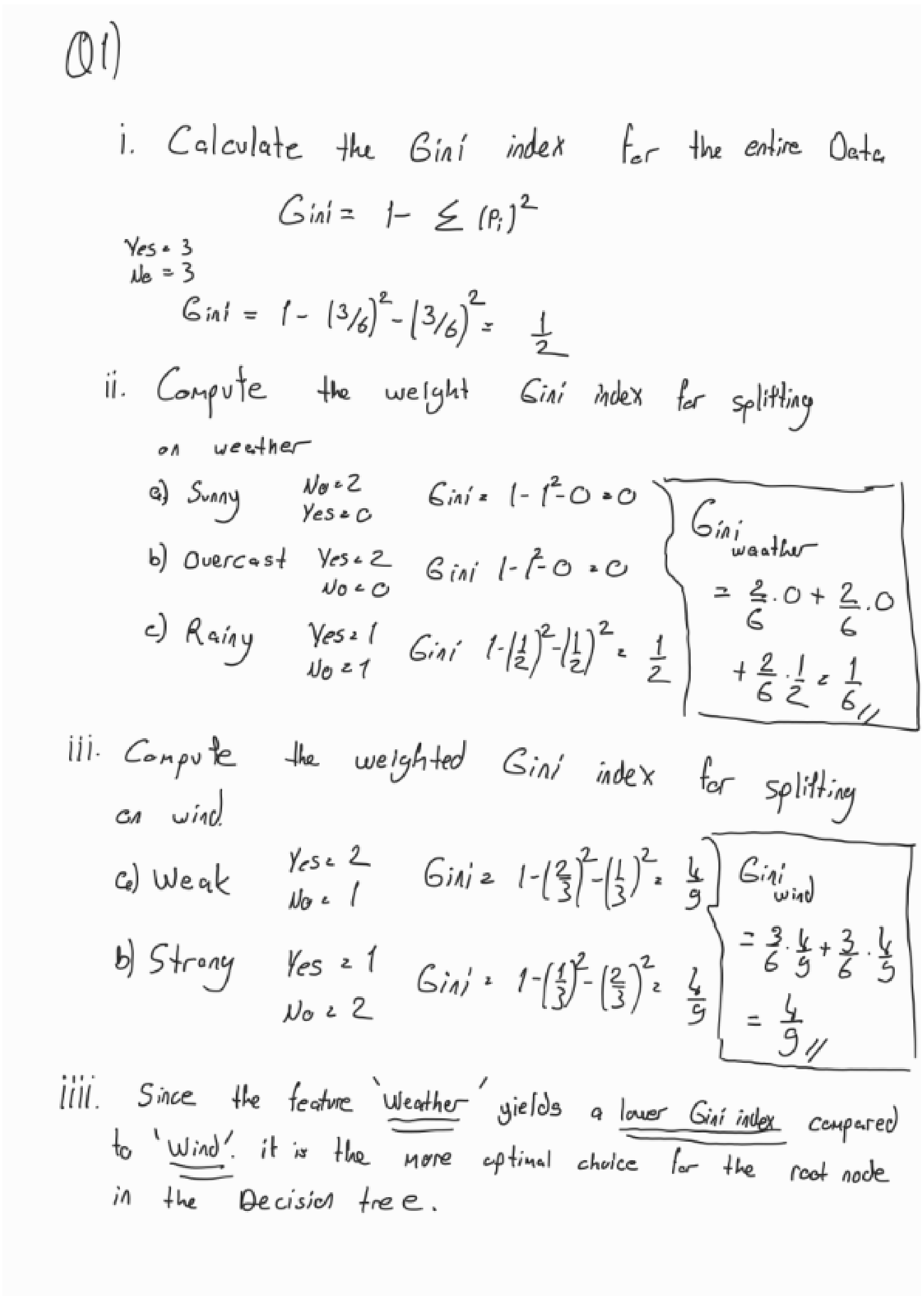
GitHub

06.04.2025

Q1) Gini Index Calculation



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



Q1)

i. Calculate the Gini index for the entire Data

$$Gini = 1 - \sum (P_i)^2$$

Yes = 3
No = 3

$$Gini = 1 - (3/6)^2 - (3/6)^2 = \frac{1}{2}$$

ii. Compute the weight Gini index for splitting on weather

a) Sunny No = 2, Yes = 0 $\quad Gini = 1 - 1^2 - 0 = 0$

b) Overcast Yes = 2, No = 0 $\quad Gini \; 1 - 1^2 - 0 = 0$

c) Rainy Yes = 1, No = 1 $\quad Gini \; 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = \frac{1}{2}$

$$\boxed{Gini_{weather} = \frac{2}{6} \cdot 0 + \frac{2}{6} \cdot 0 + \frac{2}{6} \cdot \frac{1}{2} = \frac{1}{6}}$$

iii. Compute the weighted Gini index for splitting on wind

a) Weak Yes = 2, No = 1 $\quad Gini = 1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2 = \frac{4}{9}$

b) Strong Yes = 1, No = 2 $\quad Gini = 1 - \left(\frac{1}{3}\right)^2 - \left(\frac{2}{3}\right)^2 = \frac{4}{9}$

$$\boxed{Gini_{wind} = \frac{3}{6} \cdot \frac{4}{9} + \frac{3}{6} \cdot \frac{4}{9} = \frac{4}{9}}$$

iiii. Since the feature '<u>Weather</u>' yields a <u>lower Gini index</u> compared to '<u>Wind</u>', it is the more optimal choice for the root node in the Decision tree.

Q2) Implement the decision tree algorithm for the Banknote Authentication Dataset from the UCI Machine Learning Repository

2.1) **Variance** :measures how spread out or dispersed pixel values in an image are. A higher variance indicates more contrast, while a lower variance shows that pixel values are closer to each other.

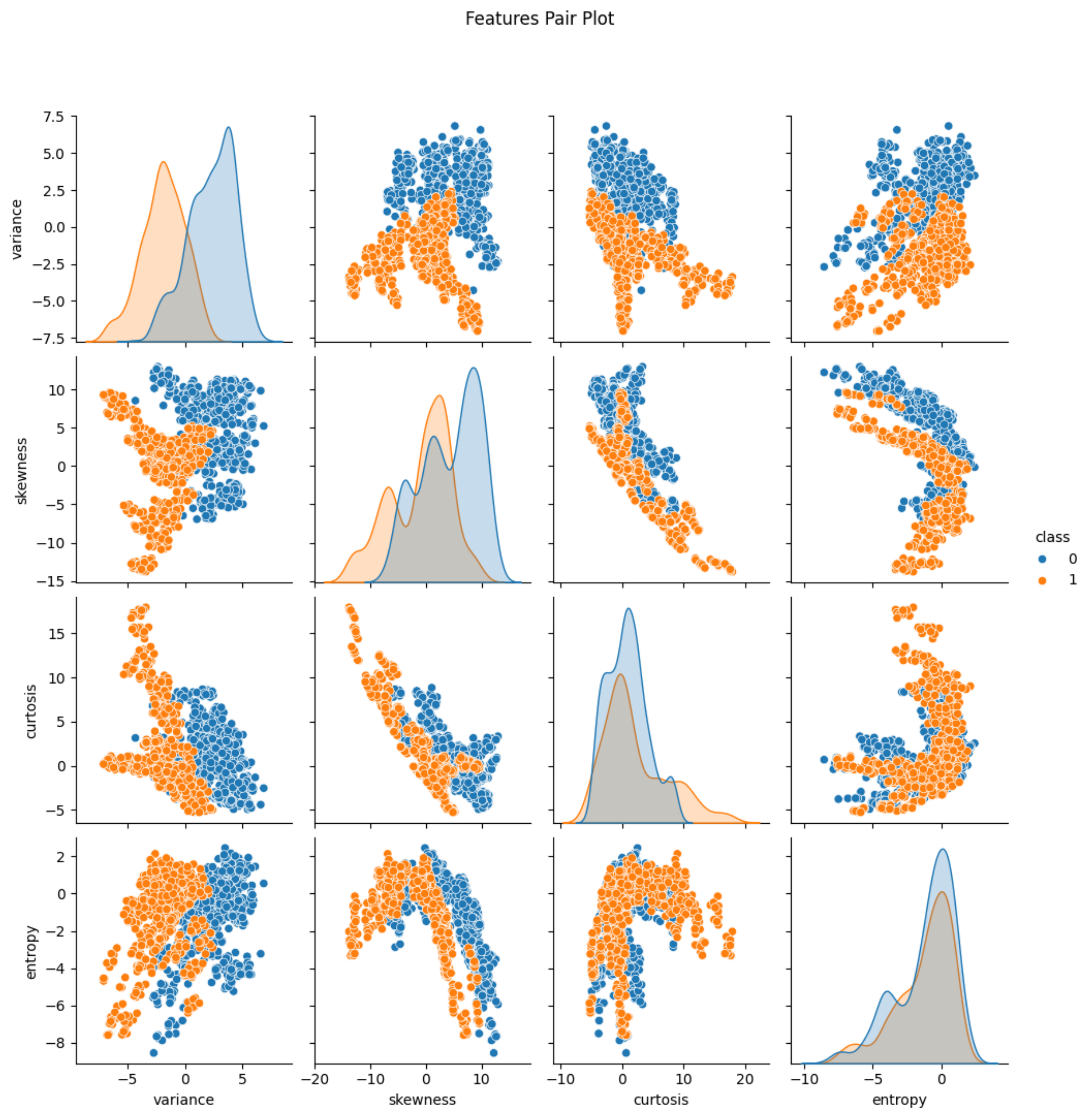
Skewness: shows if the pixel intensity distribution is symmetric or shifted towards brighter or darker values. Positive skewness means the image is darker overall, while negative skewness means it's brighter.

Kurtosis: measures how peaked or flat the distribution of pixel values is. High kurtosis indicates pixel values clustered around the mean (sharp peak), and low kurtosis means the distribution is flatter, indicating more varied pixel values.

Entropy: quantifies the randomness or complexity in an image. Higher entropy means more complexity or detail, while lower entropy suggests the image is simpler or more uniform.

2.2)

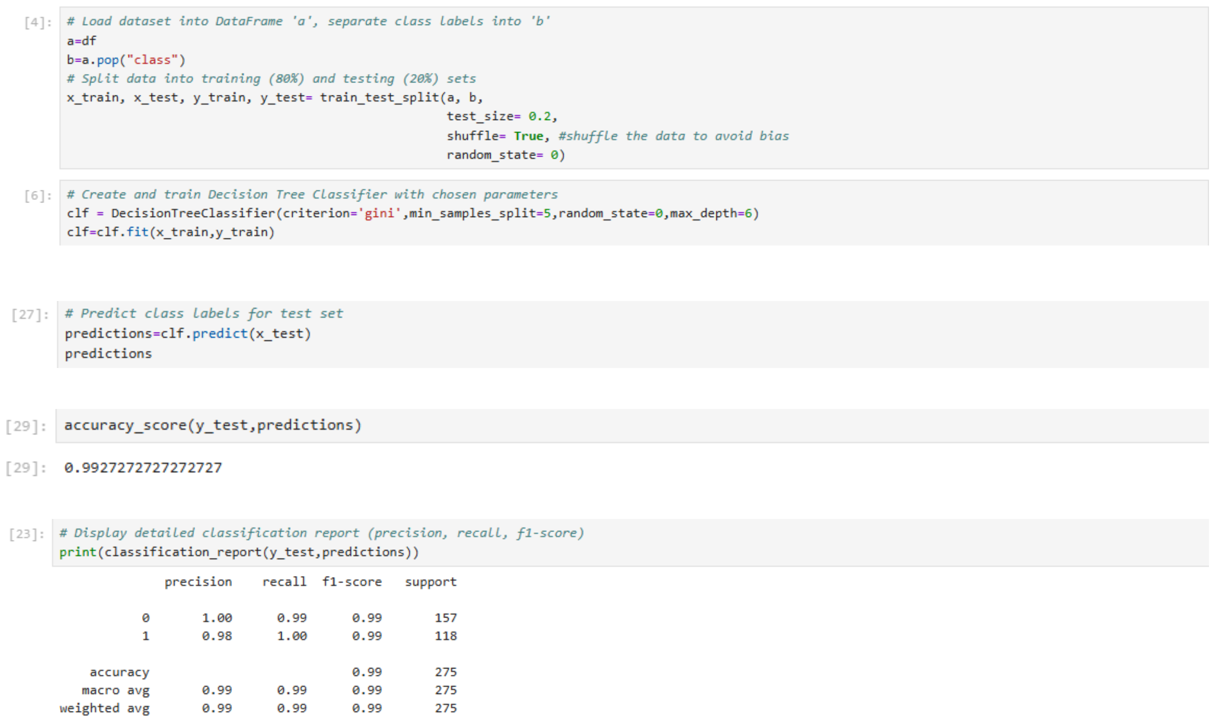
I downloaded the provided dataset, loaded it into a Pandas DataFrame, and visualized the features in groups of two to better understand the distribution and relationships between different classes ("fake" and "authentic").



Based on your visualizations, a Decision Tree seems like a good algorithm to use for your dataset. The reason is that your data shows clear differences between the two classes ("fake" and "authentic") in many of the feature plots. The groups are well-separated, making it easy for the decision tree to split the data effectively. Also, your data does not show simple linear boundaries, but a decision tree can easily handle these non-linear relationships. Overall, the patterns seen in your visualizations suggest that a Decision Tree classifier would perform well for this data.

2.3)

In my experiments, I first split the dataset into training (80%) and testing (20%) subsets using the **`train_test_split()`** function from sklearn. To find the best parameters for the Decision Tree algorithm, I tried several different parameter combinations manually and also conducted an automated hyperparameter search using **`GridSearchCV`**. After testing various settings for **`max_depth`**, **`min_samples_split`**, and the splitting criterion (**`"gini"`** vs **`"entropy"`**), I identified that the best parameters were **`criterion='gini'`**, **`min_samples_split=2`**, **`max_depth=6`**, and **`random_state=0`**. With these chosen parameters, my Decision Tree achieved an accuracy of approximately **0.9927**. Additionally, I evaluated the model's performance using metrics like precision, recall, and F1-score through a classification report and visualized the model's effectiveness with a confusion matrix. The high accuracy and strong metrics confirm that this parameter configuration successfully classifies banknotes as fake or authentic.



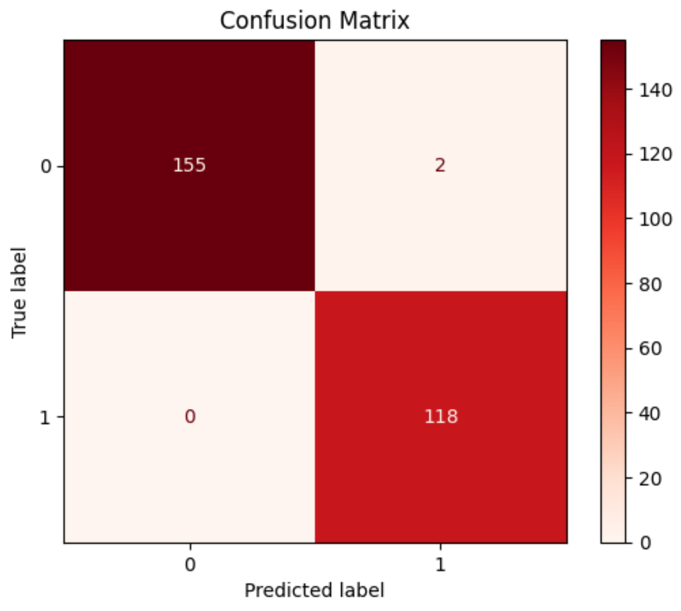
```
[4]:  # Load dataset into DataFrame 'a', separate class labels into 'b'
      a=df
      b=a.pop("class")
      # Split data into training (80%) and testing (20%) sets
      x_train, x_test, y_train, y_test= train_test_split(a, b,
                                                          test_size= 0.2,
                                                          shuffle= True, #shuffle the data to avoid bias
                                                          random_state= 0)
```

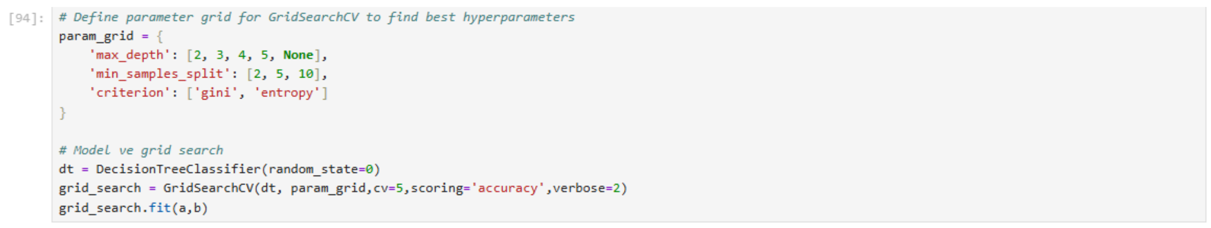
```
[6]:  # Create and train Decision Tree Classifier with chosen parameters
      clf = DecisionTreeClassifier(criterion='gini',min_samples_split=5,random_state=0,max_depth=6)
      clf=clf.fit(x_train,y_train)
```

```
[27]: # Predict class labels for test set
      predictions=clf.predict(x_test)
      predictions
```

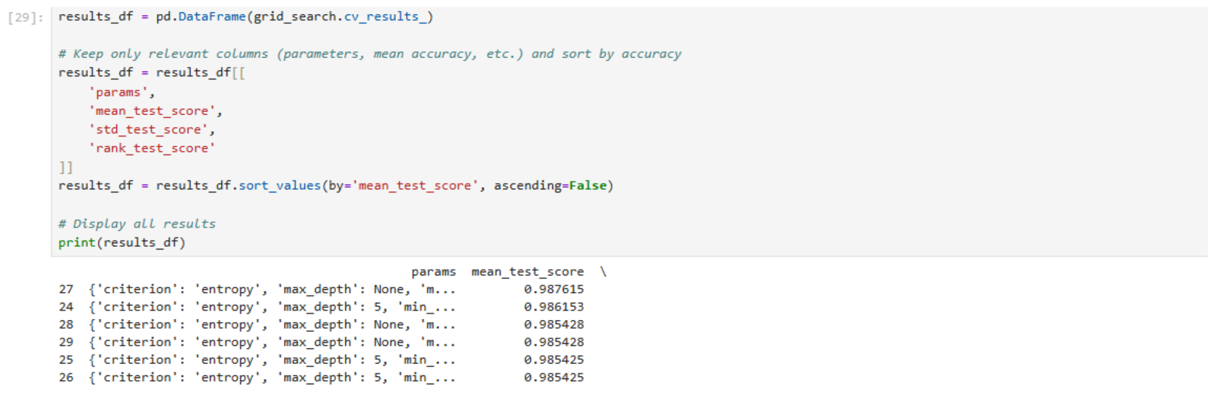
```
[29]: accuracy_score(y_test,predictions)

[29]: 0.9927272727272727
```

```
[23]: # Display detailed classification report (precision, recall, f1-score)
      print(classification_report(y_test,predictions))

              precision    recall  f1-score   support

           0       1.00      0.99      0.99       157
           1       0.98      1.00      0.99       118

    accuracy                           0.99       275
   macro avg       0.99      0.99      0.99       275
weighted avg       0.99      0.99      0.99       275
```





```
[94]: # Define parameter grid for GridSearchCV to find best hyperparameters
      param_grid = {
          'max_depth': [2, 3, 4, 5, None],
          'min_samples_split': [2, 5, 10],
          'criterion': ['gini', 'entropy']
      }

      # Model ve grid search
      dt = DecisionTreeClassifier(random_state=0)
      grid_search = GridSearchCV(dt, param_grid,cv=5,scoring='accuracy',verbose=2)
      grid_search.fit(a,b)
```



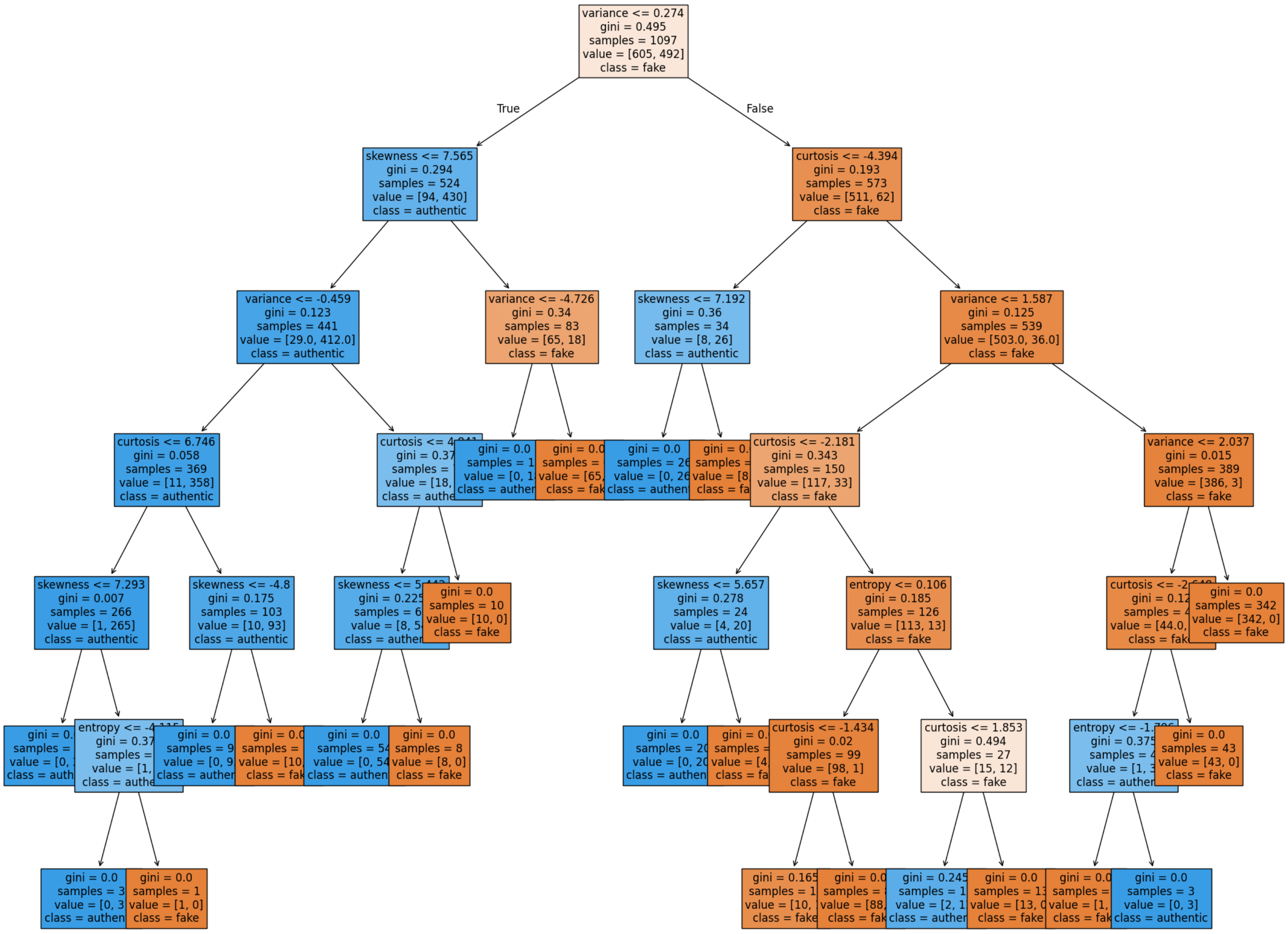
```
[29]: results_df = pd.DataFrame(grid_search.cv_results_)

      # Keep only relevant columns (parameters, mean accuracy, etc.) and sort by accuracy
      results_df = results_df[[
          'params',
          'mean_test_score',
          'std_test_score',
          'rank_test_score'
      ]]
      results_df = results_df.sort_values(by='mean_test_score', ascending=False)

      # Display all results
      print(results_df)
```

```
                                               params  mean_test_score  \
27  {'criterion': 'entropy', 'max_depth': None, 'm...         0.987615
24  {'criterion': 'entropy', 'max_depth': 5, 'min_...         0.986153
28  {'criterion': 'entropy', 'max_depth': None, 'm...         0.985428
29  {'criterion': 'entropy', 'max_depth': None, 'm...         0.985428
25  {'criterion': 'entropy', 'max_depth': 5, 'min_...         0.985425
26  {'criterion': 'entropy', 'max_depth': 5, 'min_...         0.985425
```

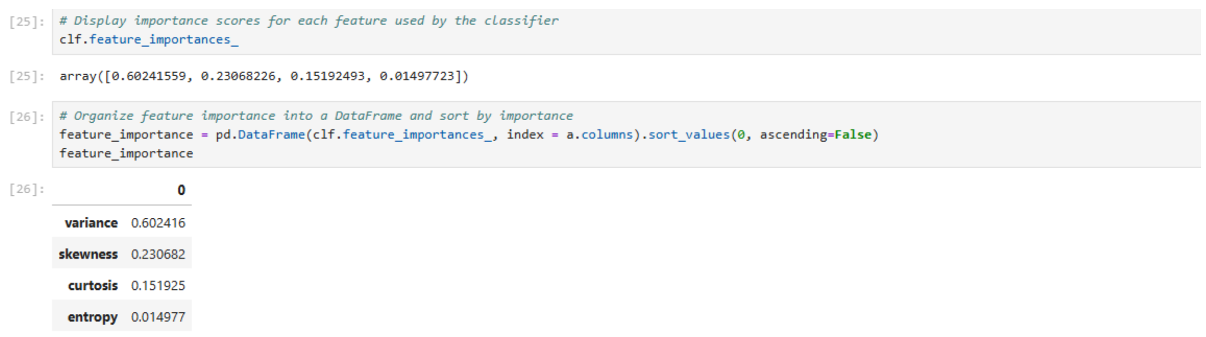
The results obtained from GridSearchCV differ slightly from the parameters I manually selected through my initial experiments. Specifically, GridSearchCV identified the best-performing parameter combination using the criterion as **`'entropy'`**, which is different from my manually chosen **`'gini'`**. Additionally, the maximum depth selected by GridSearchCV was **`None`**, meaning no restriction on tree depth, compared to the **`max_depth=6`** I selected manually. Despite these differences, the manual parameter choice (**`criterion='gini',max_depth=6,min_samples_split=2`**) provided an accuracy of approximately `0.9927`, which slightly exceeds the performance of the best GridSearchCV result (`0.9876`). This highlights the effectiveness of manual tuning in certain contexts.

2.4)

Analyzing how tree depth affects interpretability, it's clear that deeper trees (higher maximum depth) can capture more complex patterns in the data, potentially improving accuracy. However, increasing the depth also makes the tree more complex, harder to interpret, and prone to overfitting. In contrast, shallower trees are easier to interpret and visualize, making them preferable when explainability is crucial. The chosen depth of 6 in my model represents a balanced choice, achieving excellent accuracy (0.9927) while maintaining sufficient interpretability, clearly illustrating how individual features contribute to decision-making without becoming overly complex.


variance <= 0.274 gini = 0.495 samples = 1097 value = [605, 492] class = fake
True
False
skewness <= 7.565 gini = 0.294 samples = 524 value = [94, 430] class = authentic
curtosis <= -4.394 gini = 0.193 samples = 573 value = [511, 62] class = fake
variance <= -0.459 gini = 0.123 samples = 441 value = [29.0, 412.0] class = authentic
variance <= -4.726 gini = 0.34 samples = 83 value = [65, 18] class = fake
skewness <= 7.192 gini = 0.36 samples = 34 value = [8, 26] class = authentic
variance <= 1.587 gini = 0.125 samples = 539 value = [503.0, 36.0] class = fake
curtosis <= 6.746 gini = 0.058 samples = 369 value = [11, 358] class = authentic
curtosis <= -2.181 gini = 0.343 samples = 150 value = [117, 33] class = fake
variance <= 2.037 gini = 0.015 samples = 389 value = [386, 3] class = fake
skewness <= 7.293 gini = 0.007 samples = 266 value = [1, 265] class = authentic
skewness <= -4.8 gini = 0.175 samples = 103 value = [10, 93] class = authentic
gini = 0.0 samples = 10 value = [10, 0] class = fake
skewness <= 5.657 gini = 0.278 samples = 24 value = [4, 20] class = authentic
entropy <= 0.106 gini = 0.185 samples = 126 value = [113, 13] class = fake
gini = 0.0 samples = 342 value = [342, 0] class = fake
gini = 0.0 samples = 8 value = [8, 0] class = fake
curtosis <= -1.434 gini = 0.02 samples = 99 value = [98, 1] class = fake
curtosis <= 1.853 gini = 0.494 samples = 27 value = [15, 12] class = fake
gini = 0.0 samples = 43 value = [43, 0] class = fake
gini = 0.0 samples = 3 value = [0, 3] class = authentic
gini = 0.0 samples = 1 value = [1, 0] class = fake

2.5)



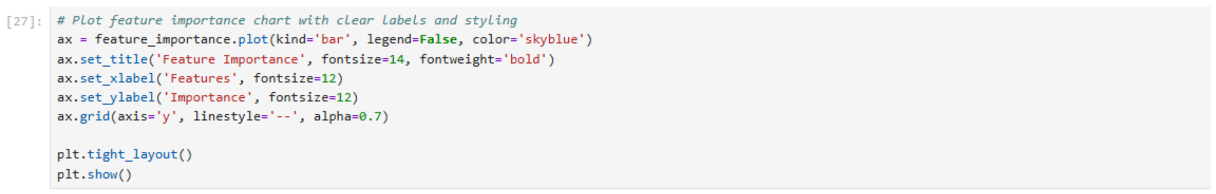
```
[25]: # Display importance scores for each feature used by the classifier
      clf.feature_importances_
```

```
[25]: array([0.60241559, 0.23068226, 0.15192493, 0.01497723])
```

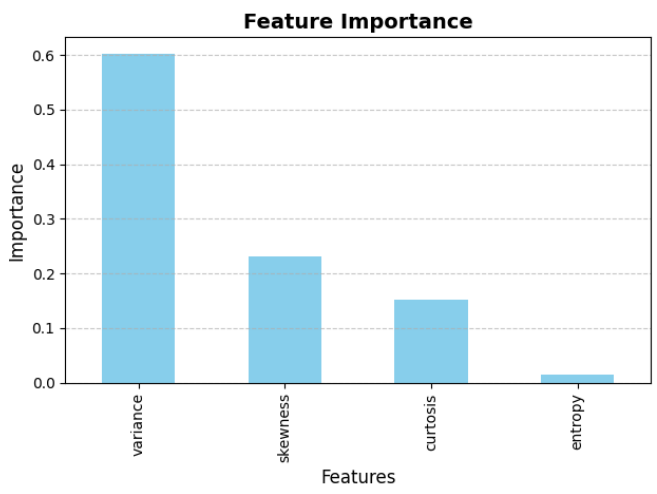
```
[26]: # Organize feature importance into a DataFrame and sort by importance
      feature_importance = pd.DataFrame(clf.feature_importances_, index = a.columns).sort_values(0, ascending=False)
      feature_importance
```

[26]:

	0
variance	0.602416
skewness	0.230682
curtosis	0.151925
entropy	0.014977



```
[27]: # Plot feature importance chart with clear labels and styling
      ax = feature_importance.plot(kind='bar', legend=False, color='skyblue')
      ax.set_title('Feature Importance', fontsize=14, fontweight='bold')
      ax.set_xlabel('Features', fontsize=12)
      ax.set_ylabel('Importance', fontsize=12)
      ax.grid(axis='y', linestyle='--', alpha=0.7)

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



2.6)

From this question, I learned how to apply the Decision Tree algorithm to a real-world dataset and evaluate its performance using different metrics and parameter settings. By experimenting with both manual tuning and GridSearchCV, I gained insight into how hyperparameters like **`max_depth`**, **`min_samples_split`**, and **`criterion`** affect the model's accuracy and complexity. I also explored the importance of each feature and how tree depth influences interpretability. Based on the results, I still believe that the Decision Tree is a very suitable model for the Banknote Authentication dataset. It achieved high accuracy (up to 0.9927), provided clear visual explanations through its tree structure, and effectively captured the non-linear patterns in the data. These qualities make it both powerful and interpretable, which are key strengths in practical machine learning applications.