PYTHON FOR A.I.

MUSHROOM EDIBILITY PREDICTION USING MACHINE LEARNING

RAYEN GHORDO

1 INTRODUCTION

This project focuses on building an AI model that predicts whether a mushroom is edible or poisonous, based on a set of biological and physical characteristics. The dataset consists of several categorical and numerical characteristics of mushrooms. The project includes data exploration, preprocessing, model training, evaluation and storing the model using Python and scikit-learn.

Some parts of the code and methodology used in this project were inspired by examples from the following book:

Géron, A. (2022). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (3rd ed.)*. O'Reilly Media. ISBN: 978-1-098-12597-4.

2 INHOUDSOPGAVE

3 DATASET

The dataset used in this project contains structured information on various characteristics of mushrooms, collected to determine whether a mushroom is edible or poisonous. The data contains both numeric (metric) and categorical (nominal) variables, describing physical properties and environmental factors, among others.

3.1 TARGET FEATURE

Class (binary):

- e = Edible
- p = Poisonous

This is the column we will try to predict, using machine learning.

3.2 FEATURE VARIABLES

Below is an overview of all input features with their data type and possible values:

Metrical (Numerical) Features

1. **cap-diameter**:
    - Unit: centimetres
    - Description: Diameter of the mushroom cap
2. **stem-height**:
    - Unit: centimetres
    - Description: Height of the mushroom stem
3. **stem-width**:
    - Unit: millimetres
    - Description: Width (thickness) of the mushroom stem

Nominal (Categorical) Features

4. **cap-shape**:
    - Values: bell (b), conical (c), convex (x), flat (f), sunken (s), spherical (p), others (o)
5. **cap-surface**:
    - Values: fibrous (i), grooves (g), scaly (y), smooth (s), shiny (h), leathery (l), silky (k), sticky (t), wrinkled (w), fleshy (e)
6. **cap-color**:
    - Values: brown (n), buff (b), gray (g), green (r), pink (p), purple (u), red (e), white (w), yellow (y), blue (l), orange (o), black (k)
7. **does-bruise-bleed**:
    - Values: bruises/bleeds (t), no (f)
8. **gill-attachment**:
    - Values: adnate (a), adnexed (x), decurrent (d), free (e), sinuate (s), pores (p), none (f), unknown (?)
9. **gill-spacing**:
    - Values: close (c), distant (d), none (f)
10. **gill-color**:

- Values: same as cap-color plus none (f)

11. **stem-root**:

- Values: bulbous (b), swollen (s), club (c), cup (u), equal (e), rhizomorphs (z), rooted (r)

12. **stem-surface**:

- Values: same as cap-surface plus none (f)

13. **stem-color**:

- Values: same as cap-color plus none (f)

14. **veil-type**:

- Values: partial (p), universal (u)

15. **veil-color**:

- Values: same as cap-color plus none (f)

16. **has-ring**:

- Values: ring (t), none (f)

17. **ring-type**:

- Values: cobwebby (c), evanescent (e), flaring (r), grooved (g), large (l), pendant (p), sheathing (s), zone (z), scaly (y), movable (m), none (f), unknown (?)

18. **spore-print-color**:

- Values: same as cap-color

19. **habitat**:

- Values: grasses (g), leaves (l), meadows (m), paths (p), heaths (h), urban (u), waste (w), woods (d)

20. **season**:

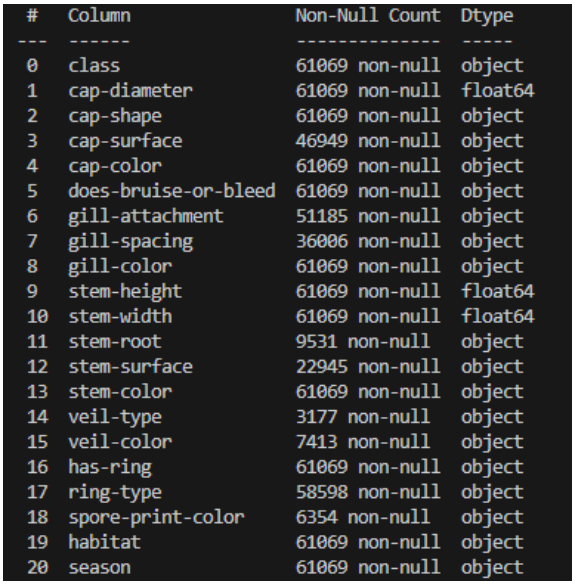
- Values: spring (s), summer (u), autumn (a), winter (w)

4 DATA ANALYSIS AND STRUCTURE

4.1 DATA EXPLORATION

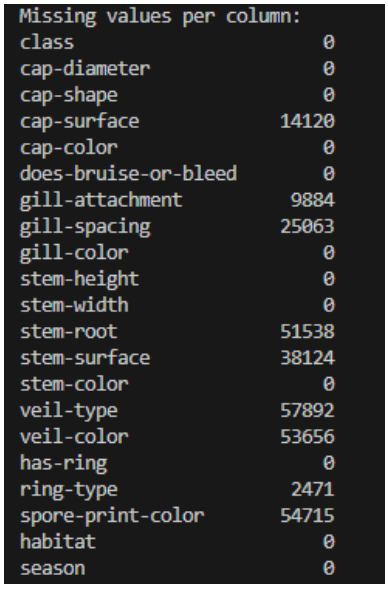
Before training any machine learning models, I have to perform a detailed exploration of the dataset to gain insight of its structure, quality and internal relations.

The initial overview of the dataset (*Figure 2*) revealed that it consists of 21 features, with a mix of numerical and categorical features. While some columns are complete, others have a lot of missing values. Identifying these gaps early is essential, as they need to be dealt with during the pre-processing phase to ensure the model can effectively learn from the data.



#	Column	Non-Null Count	Dtype
0	class	61069 non-null	object
1	cap-diameter	61069 non-null	float64
2	cap-shape	61069 non-null	object
3	cap-surface	46949 non-null	object
4	cap-color	61069 non-null	object
5	does-bruise-or-bleed	61069 non-null	object
6	gill-attachment	51185 non-null	object
7	gill-spacing	36006 non-null	object
8	gill-color	61069 non-null	object
9	stem-height	61069 non-null	float64
10	stem-width	61069 non-null	float64
11	stem-root	9531 non-null	object
12	stem-surface	22945 non-null	object
13	stem-color	61069 non-null	object
14	veil-type	3177 non-null	object
15	veil-color	7413 non-null	object
16	has-ring	61069 non-null	object
17	ring-type	58598 non-null	object
18	spore-print-color	6354 non-null	object
19	habitat	61069 non-null	object
20	season	61069 non-null	object

Figure 2: Overview of dataset



Missing values per column:	
class	0
cap-diameter	0
cap-shape	0
cap-surface	14120
cap-color	0
does-bruise-or-bleed	0
gill-attachment	9884
gill-spacing	25063
gill-color	0
stem-height	0
stem-width	0
stem-root	51538
stem-surface	38124
stem-color	0
veil-type	57892
veil-color	53656
has-ring	0
ring-type	2471
spore-print-color	54715
habitat	0
season	0

Figure 1: Missing values

I plotted histograms (*Figure 3*) for each feature in the dataset to visualise their distributions. Numerical features such as **cap diameter** and **stem height** were found to be right-skewed, indicating that most mushrooms are relatively small, with a few much larger examples. This kind of skewness can negatively affect some machine learning models. In preprocessing, this problem can be addressed by applying normalisation techniques such as standard scales or transformations to reduce the influence of outliers and bring all features to a similar scale.

The categorical features showed varying degrees of imbalances. Some were dominated by one category, which may limit their predictive power, while others were more evenly distributed. These categorical variables typically require careful encoding, such as one-hot or ordinal encoding, to convert them into a format suitable for machine learning models. Dealing with missing values is another important concern, as some features had a large number of missing values. These can be handled by imputing the missing values or treating 'missing' as a separate category, depending on the context and importance of the feature.

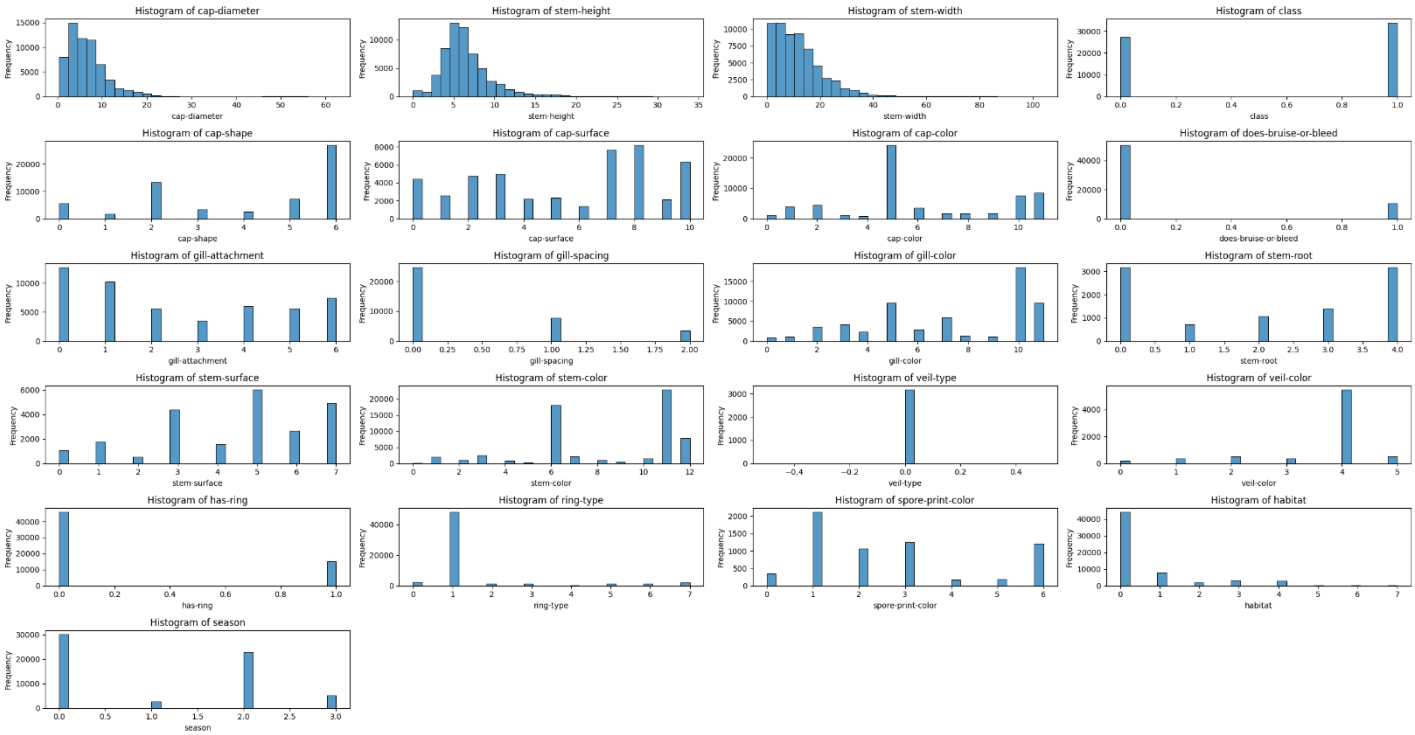


Figure 3: Histograms

4.2 FEATURE RELATIONS

A correlation matrix (*Figure 4*) was then generated by encoding the categorical values numerically and combining them with the original numerical features. This matrix revealed some interesting patterns: for example, there was a strong correlation between cap diameter and stem width, which is biologically logical- larger mushrooms usually have thicker stems. And the feature "**veil-type**" has no specific relations with other features. However, the target variable class (indicating edibility) showed only weak correlations with most individual features, suggesting that the prediction probably relies on a combination of multiple features rather than a single feature.

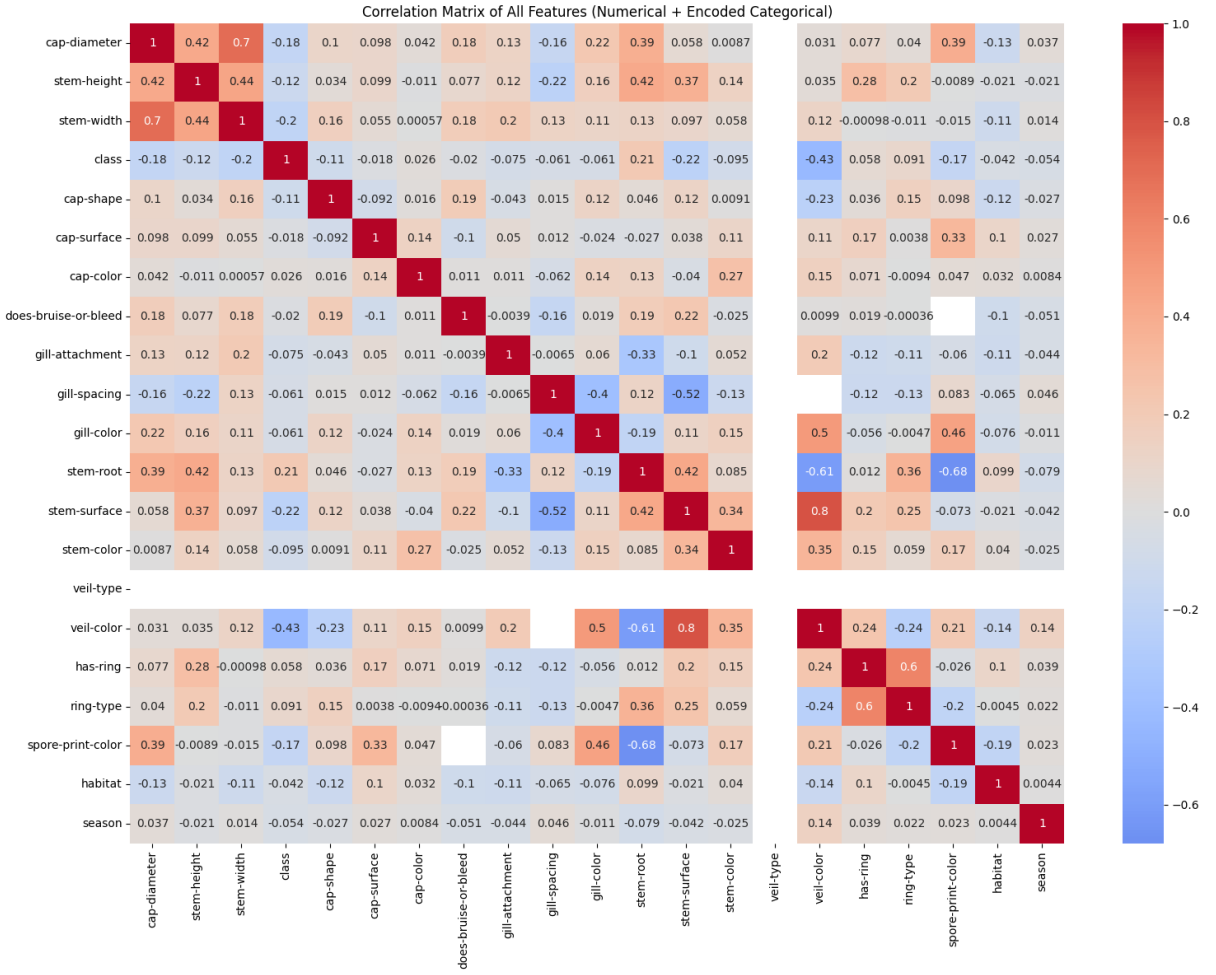


Figure 4: Correlation Matrix

To further explore the relationships between categorical characteristics, Chi-square tests were used to analyse the relationships between categorical characteristics. Two heatmaps were generated: the Chi-squared statistical matrix (*Figure 6*) and the Chi-squared p-value matrix (*Figure 5*).

At first glance, the Chi-square statistical matrix shows the strength of the correlation between categorical variables. High values (visualised in darker shades of red or orange) indicate stronger statistical correlation, meaning that the two characteristics are not independent. However, it is important to note that high Chi-squared values do not always imply a meaningful correlation. They are highly dependent on sample size and distribution. This is where the p-value matrix becomes essential.

The p-value matrix tells us whether the correlation we see in the Chi-squared statistic is statistically meaningful. A low p-value (usually below 0.05) suggests that the relationship is unlikely to be due to coincidence and can be considered meaningful. Conversely, high p-values (closer to 1, shown as dark blue in the heatmap) suggest that there is no significant relationship, even if the Chi-squared statistic appears high.

For example, in the Chi-squared statistic matrix, the feature **veil type** may show light colouring (indicating that the values are computed), but in the p-value matrix it appears dark blue, indicating very high p-values, which are usually close to 1. This means that although the Chi-squared values have been computed, they are not statistically meaningful. In other words, veil type has no meaningful correlation with the other categorical features.

This distinction is crucial. When analysing feature relationships, we should not only rely on the Chi-squared statistic. Always check p-values to confirm whether the observed relationships are statistically relevant. In our case, traits such as **veil type** and **spore print colour** probably contain too many missing or uniform values, leading to unreliable or irrelevant relationships, which is confirmed by their high p-values.

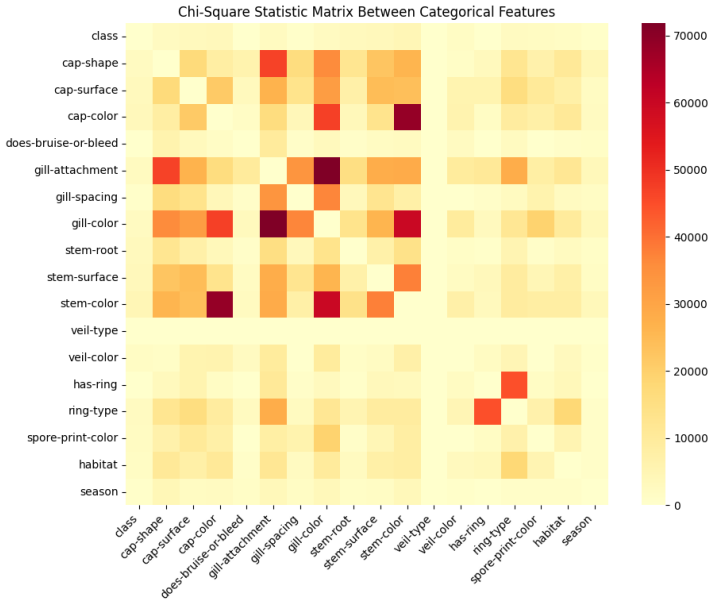


Figure 5: Chi-Square Matrix

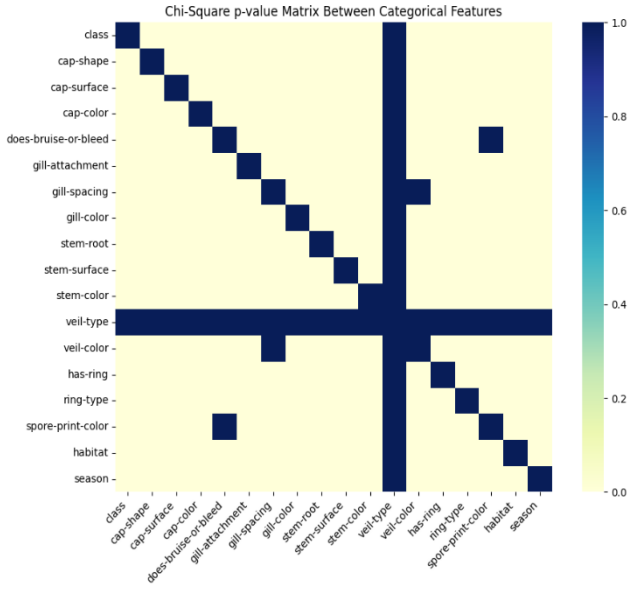


Figure 6: P-Value Matrix

5 DATA SPLITTING STRATEGY

To prepare the dataset for training and testing, it is essential to split it into a training set and a testing set. However, a simple random split may result in uneven distributions of important features or unbalanced target classes across the two sets, which may affect the performance and reliability of the model. To avoid this, I used a stratified sampling.

In this case, stratification was not performed directly on the target variable, but on a key numerical feature: **stem width**. Since **stem width** is a continuous variable, it was first transformed into discrete bins. Specifically, values from 0 to 49 were divided into individual categories and any value above 50 was placed in a special bin labelled >50. This created a new categorical feature, **stem-width-binned**, which represents the distribution of trunk widths in a more manageable form for stratification purposes.

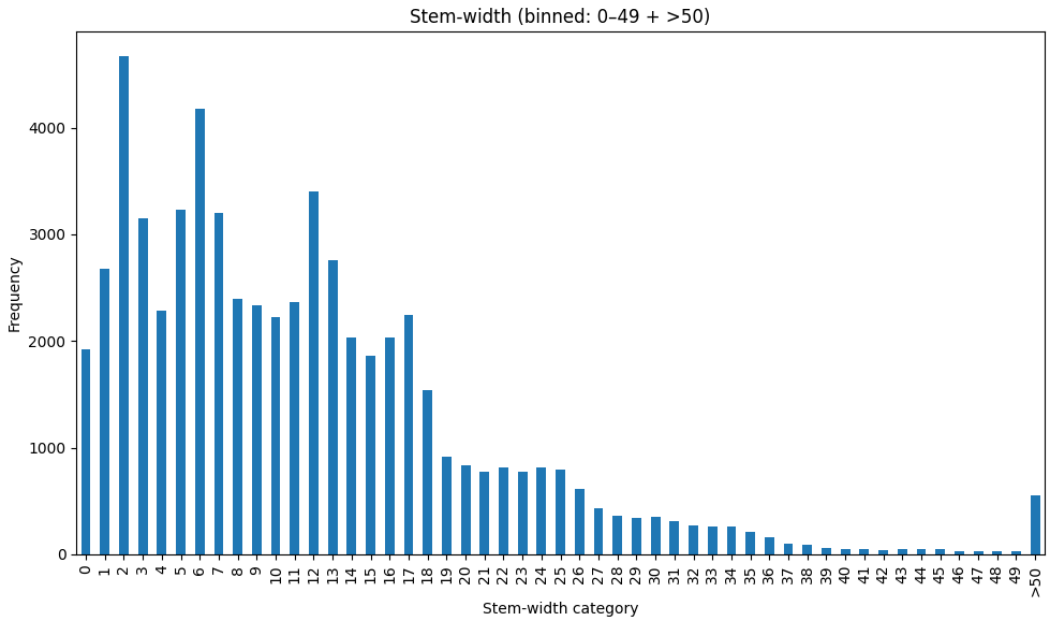


Figure 7: Histogram of stem-width-binned

By stratifying on this binned version of **stem width**, both the training and test sets maintain a representative distribution of mushrooms with thin, medium and thick stems. This is important because stem thickness can be a predictive feature and unbalanced distributions between splits can affect the model.

After splitting, the temporary column **stem-width-binned** was removed as it was only used for the stratification process.

6 PREPROCESSING

Before feeding the data into machine learning models, preprocessing is essential to prepare the data in a way that models can understand and learn from it.

6.1 TRANSFORMING THE DATA

First I define the preprocessing pipeline. This process transforms raw input data into a format that machine-learning algorithms can correctly understand.

For **numerical features**, the following transformations are applied:

- Missing values are filled using the median, which is more robust to outliers and works well for skewed data.

- Standard scaling is then applied to centre each feature around zero with unit of variance. This is especially important for algorithms that are sensitive to feature size, such as k-Nearest Neighbors and Support Vector Machines.

For **categorical features**:

- Missing values are filled using the most frequent feature category, so that the feature can still be used without removing large parts of the data.

- One-hot encoding is then applied to convert each categorical value into a separate binary attribute. This avoids false ordinal relationships and allows the model to deal effectively with non-numeric data.

6.2 TRANSFORMATION

To streamline this process, both pipelines (numerical and categorical) are combined using a **ColumnTransformer**. This structure ensures that preprocessing is applied consistently and automatically during model training and prediction. It also simplifies code maintenance and reproducibility, as the same transformation logic is applied to all models in the pipeline.

7 MODEL TRAINING

After preprocessing, I trained multiple machine-learning models to classify mushrooms as edible or poisonous. This allows me to compare the strengths and weaknesses of different models when applied to the same task.

The following models were evaluated:

- Decision Tree
- Random Forest
- k-Nearest Neighbors (KNN)
- Support Vector Machine (SVM)
- Naive Bayes
- Stochastic Gradient Descent (SGD)

8 BASE MODEL PERFORMANCE

8.1 DECISION TREE

A Decision Tree is a flowchart-like structure that splits the data into smaller “choices" and smaller groups based on the most informative features.

Results:

- Accuracy: 99.80%
- Precision (e): 99.73% — Recall (e): 99.83%
- Precision (p): 99.87% — Recall (p): 99.78%

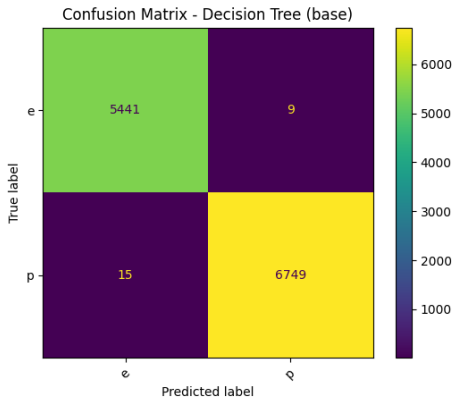


Figure 8: Decision Tree

Most important features:

stem-width, cap-diameter, gill-spacing_d, stem-color_w

Analysis:

The Decision Tree performs well because its very flexible and can adapt to patterns in the data. With just 24 total errors on over 12,000 test samples, it clearly captures strong rules in the dataset. This could be overfitting but if we look at the important features, they all make logical sense in mushroom identification.

Conclusion:
Very high-performing model with a slight risk of overfitting due to how flexible trees are. Still, the important features and error patterns suggest good generalization.

8.2 RANDOM FOREST

A Random Forest combines many decision trees, each trained on slightly different data and using a random subset of features. The final prediction is made by voting.

Results:

- Accuracy: 100%
- Precision & Recall for both classes: 100%

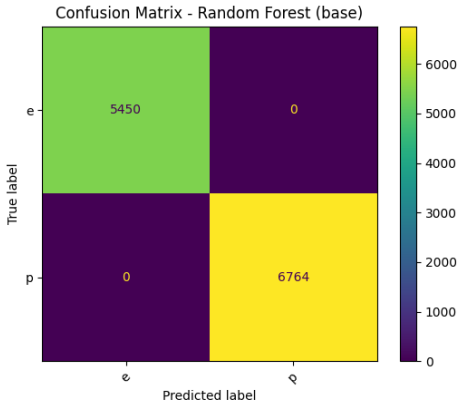


Figure 9: Random Forest

Most important features:

stem-width, stem-height, gill-spacing_d, stem-color_w

Analysis:

The Random Forest performs perfectly because it combines many trees to reduce overfitting of individual trees. On well-structured data, this can lead to near-perfect performance. It is still suspicious, a perfect score on a test set is rare. It could mean the model overfit to noise or that the test set is too similar to the training data, especially if the model is very deep.

But if we look at the top features, they make sense.

Conclusion:
Excellent result. It is likely to generalise well, but 100% accuracy needs cautiousness. It should ideally be tested on unseen, external data to rule out overfitting.

8.3 K-NEAREST NEIGHBOURS

K-Nearest Neighbours (KNN) does not actually train a model. Instead, it stores the training data and classifies new inputs based on the most common class among the k closest points.

Results:

- Accuracy: 99.99%
- Precision (e): 100% — Recall (e): 99.98%
- Precision (p): 99.99% — Recall (p): 100%

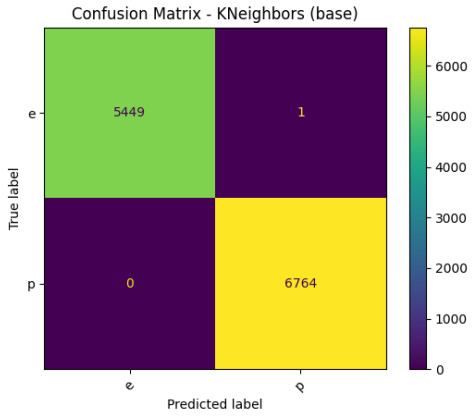


Figure 10: k-Nearest Neighbours

Most important features:

KNN does not calculate feature importances.

Analysis:

The model's performance is nearly perfect and this is how KNN works. Since similar mushrooms likely have similar features, KNN benefits from clear patterns in the data and generalizes well. Its simplicity also limits the risk of overfitting.

Conclusion:
A reliable model for this dataset with no signs of overfitting. Performance is consistent with how KNN is expected to behave on structured data.

8.4 SUPPORT VECTOR

Support Vector finds the best separating line between two classes by maximizing the distance to the closest data points. A larger distance usually means better performance on new data.

Results:

- Accuracy: 99.90%
- Precision (e): 99.96% — Recall (e): 99.89%
- Precision (p): 99.85% — Recall (p): 99.97%

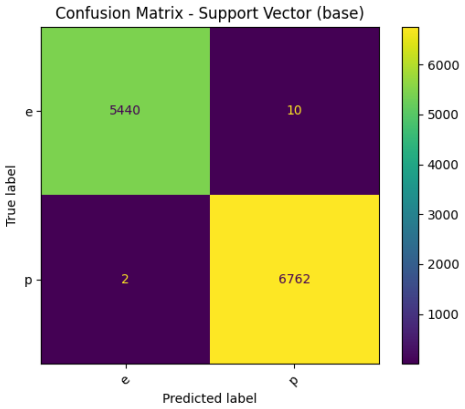


Figure 11: Support Vector

Most important features:

Support Vector does not calculate feature importances.

Analysis:

The model performs very well and appears to be slightly misclassifying edible mushrooms as poisonous. This fits how Support Vector works. If one class (e.g. poisonous) has more consistent or distinct patterns, the Support Vector will place the boundary closer to the other class (e.g. edible). This means that borderline edible mushrooms are more likely to be classified as poisonous, not because the model "knows" edible is safer, but because the poisonous class is easier to separate.

Conclusion:
A reliable model for this dataset with no signs of overfitting. Performance is consistent with how Support Vector is expected to behave on structured data.

8.5 NAIVE BAYES

Naive Bayes assumes that all features are independent from each other

Results:

- Accuracy: 61.00%
- Precision (e): 53.37% — Recall (e): 99.76%
- Precision (p): 99.36% — Recall (p): 29.78%

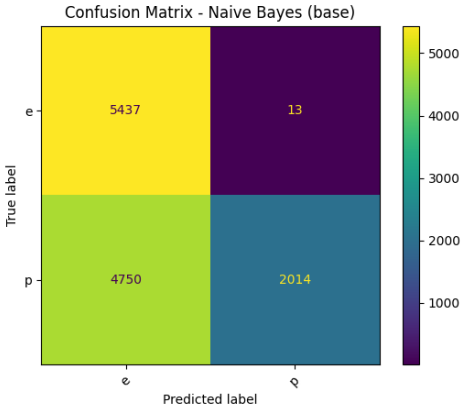


Figure 12: Naive Bayes

Most important features:

Naive Bayes does not calculate feature importances.

Analysis:

The model makes a critical amount of mistakes by misclassifying poisonous mushrooms as edible. This is highly unsafe. The poor performance results from the incorrect assumption that features are independent, which is not true in this dataset.

Conclusion:
Not suitable for this task. The model fails to capture necessary relations between features.

8.6 STOCHASTIC GRADIENT DESCENT

SGD trains a linear model by updating weights one sample at a time.

Results:

- Accuracy: 84.58 %
- Precision (e): 78.19 % — Recall (e): 90.77%
- Precision (p): 91.46% — Recall (p): 79.60%

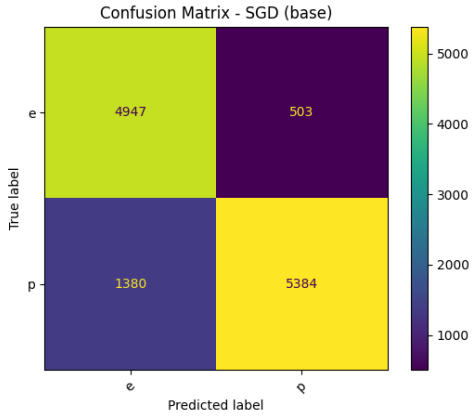


Figure 13: Stochastic Gradient Descent

Most important features:

stem-root_b, ring-type_m, stem-color_l, ring-type_z, stem-surface_t

Analysis:

The model underperforms due to its linear structure. Mushroom edibility depends on complex feature interactions, which SGD cannot learn.

Conclusion:
Too simplistic for this task, so too much risk of overfitting.

9 FINE-TUNED MODELS PERFORMANCE

While base models use default values, fine-tuned models can explore a range of parameter settings to improve their performance. I used GridSearch, specifically StratifiedKFold with 5 folds to ensure each fold maintains the class distribution of the dataset.

The models that were fine-tuned in this project include:

- Decision Tree
- Random Forest
- k-Nearest Neighbors (KNN)

9.1 DECISION TREE

Parameter Grid:

- max_depth: [3, 5, 10, 20, None]
- min_samples_split: [2, 5, 10, 20]

Results:

- Accuracy: 99.78%
- Precision (e): 99.71% — Recall (e): 99.80%
- Precision (p): 99.84% — Recall (p): 99.76%

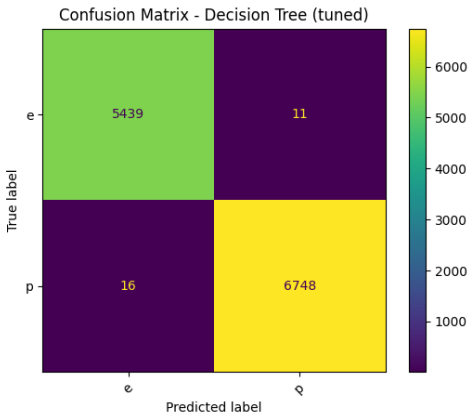


Figure 14: Decision Tree Tuned

9.2 RANDOM FOREST

Parameter Grid:

- n_estimators: [50, 100, 200]
- max_depth: [3, 5, 10, 20, None]

Results:

- Accuracy: 100%
- Precision & Recall for both classes: 100%

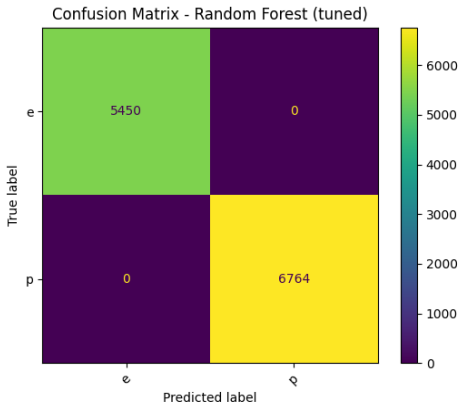


Figure 15: Random Forest Tuned

9.3 K- -NEAREST NEIGHBORS (KNN)

Parameter Grid:

- n_neighbors: [1, 3, 5, 7, 9]

weights: ['uniform', 'distance']

Results:

- Accuracy: 99.99%
- Precision (e): 100% — Recall (e): 99.98%
- Precision (p): 99.99% — Recall (p): 100%

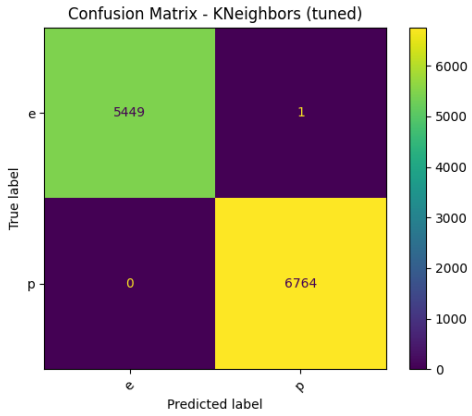


Figure 16: k-Nearest Neighbours Tuned

10 CONCLUSION

10.1 MODEL COMPARISON

Among all tested models, Random Forest and k-Nearest Neighbours achieved near-perfect accuracy on both the test and validation sets.

Support Vector also performed well but its misclassifications was almost exclusively false negatives (poisonous mushrooms predicted as edible), which could be problematic in real-world applications.

Decision Tree showed solid results, but was slightly more vulnerable to overfitting than Random Forest. This was clearly seen in the difference between cross-validation and test accuracy, since it performs slightly better on the test set.

Naive Bayes performed poorly in comparison with the other models, probably due to the strong independence assumptions that do not hold for many features in this dataset.

Stochastic Gradient Descent also fell short and performed reasonably well, but did not reach the performance of the top models.

10.2 EFFECT OF FINE-TUNING

- Decision Tree: Fine-tuning slightly improved generalization by preventing the model from becoming too deep. The tuned model performed nearly identically to the base version.
- Random Forest: The base model already achieved 100% accuracy. Fine-tuning confirmed that a reduced maximum depth maintained this perfect accuracy, while slightly reducing model complexity.
- KNN: Tuning the number of neighbors and weighting method showed that using a moderate number of neighbors (e.g., 5) with distance-based weighting further solidified the model's robustness.

10.3 Performance of All Tested Models

Method	Preprocessing	Data split	Accuracy (CV)	Accuracy (Test)	Training time (s)	Tuned parameters
Decision Tree	Norm + OneHot	Stratified 5-fold	0.9982	0.9980	1.76	-
Decision Tree (tuned)	Norm + OneHot	Stratified 5-fold	0.9984	0.9978	70.59	max_depth=10, min_samples_split=5
Random Forest	Norm + OneHot	Stratified 5-fold	0.9999	1.0000	11.00	-
Random Forest (tuned)	Norm + OneHot	Stratified 5-fold	1.0000	1.0000	230.37	n_estimators=100, max_depth=10
KNeighbors	Norm + OneHot	Stratified 5-fold	0.9999	0.9999	0.33	-
KNeighbors (tuned)	Norm + OneHot	Stratified 5-fold	1.0000	1.0000	0.32	n_neighbors=5, weights='distance'
Support Vector	Norm + OneHot	Stratified 5-fold	0.9989	0.9990	30.19	-
Naive Bayes	Norm + OneHot	Stratified 5-fold	0.6058	0.6100	0.36	-
SGD	Norm + OneHot	Stratified 5-fold	0.8493	0.8458	0.75	-

11 LIST OF FIGURES