Mushroom dataset

1. Data Description :

General information :

- **Source:** UCI Machine Learning Repository
- License : Mushroom" dataset is licensed under the Creative Commons Attribution 4.0 International (CC BY 4.0) license.
- **Area:** Biology
- **Task Type:** Classification
- **Multivariate:** Yes (has multiple features)
- **Instances (rows):** 8124
- **Features (columns):** 22 (all are categorical)
- **Target Column: poisonous** (predicts if a mushroom is **poisonous or edible**)
- **Dataset Name:** Mushroom

GOAL : classify mushrooms into as **poisonous (p)** or **edible (e)** based on their decision variables.

Feature description :

Variable Name	Description	Data Type	Possible Values
poisonous	Classification of the mushroom: edible or poisonous.	Categorical	'e' = edible, 'p' = poisonous
cap-shape	Shape of the mushroom cap.	Categorical	'b' = bell, 'c' = conical, 'x' = convex, 'f' = flat, 'k' = knobbed, 's' = sunken
cap-surface	Surface texture of the cap.	Categorical	'f' = fibrous, 'g' = grooves, 'y' = scaly, 's' = smooth
cap-color	Color of the cap.	Categorical	'n' = brown, 'b' = buff, 'c' = cinnamon, 'g' = gray, 'r' = green, 'p' = pink, 'u' = purple, 'e' = red, 'w' = white, 'y' = yellow
bruises?	Indicates if the mushroom bruises when handled.	Categorical	't' = bruises, 'f' = no bruises
odor	Smell of the mushroom.	Categorical	'a' = almond, 'l' = anise, 'c' = creosote, 'y' = fishy, 'f' = foul, 'm' = musty, 'n' = none, 'p' = pungent, 's' = spicy

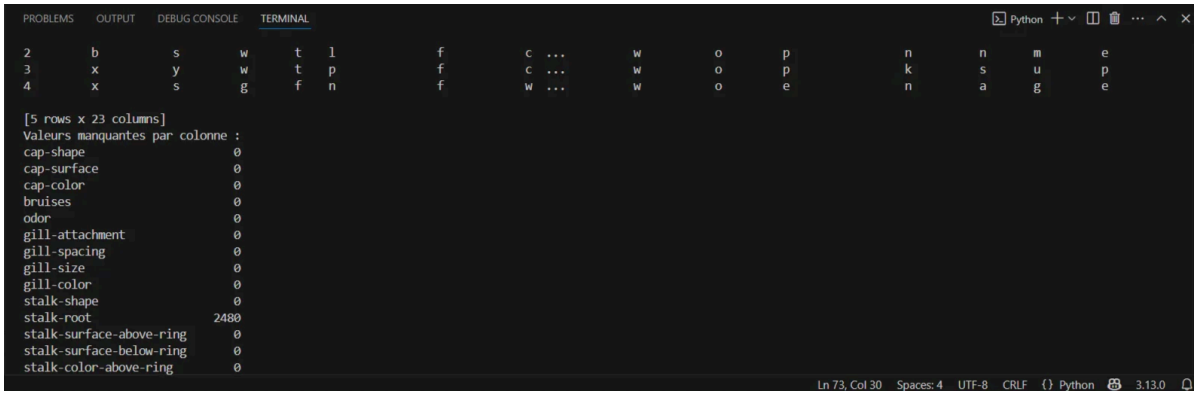
gill-attachment	Attachment of the gills to the stem.	Categorical	'a' = attached, 'd' = descending, 'f' = free, 'n' = notched
gill-spacing	Spacing between the gills.	Categorical	'c' = close, 'w' = crowded, 'd' = distant
gill-size	Size of the gills.	Categorical	'b' = broad, 'n' = narrow
gill-color	Color of the gills.	Categorical	'k' = black, 'n' = brown, 'b' = buff, 'h' = chocolate, 'g' = gray, 'r' = green, 'o' = orange, 'p' = pink, 'u' = purple, 'e' = red, 'w' = white, 'y' = yellow
stalk-shape	Shape of the mushroom's stalk.	Categorical	'e' = enlarging, 't' = tapering
stalk-root	Root type of the stalk.	Categorical	'b' = bulbous, 'c' = club, 'u' = cup, 'e' = equal, 'z' = rhizomorphs, 'r' = rooted, '?' = missing
stalk-surface-above-ring	Texture of the stalk above the ring.	Categorical	'f' = fibrous, 'y' = scaly, 'k' = silky, 's' = smooth

population	Population density where the mushroom was found.	Categorical	'a' = abundant, 'c' = clustered, 'n' = numerous, 's' = scattered, 'v' = several, 'y' = solitary
habitat	Natural habitat of the mushroom.	Categorical	'g' = grasses, 'l' = leaves, 'm' = meadows, 'p' = paths, 'u' = urban, 'w' = waste, 'd' = woods
spore-print-color	Color of the spore print.	Categorical	'k' = black, 'n' = brown, 'b' = buff, 'h' = chocolate, 'r' = green, 'o' = orange, 'u' = purple, 'w' = white, 'y' = yellow
ring-type	Type of ring on the stalk.	Categorical	'c' = cobwebby, 'e' = evanescent, 'f' = flaring, 'l' = large, 'n' = none, 'p' = pendant, 's' = sheathing, 'z' = zone
ring-number	Number of rings on the stalk.	Categorical	'n' = none, 'o' = one, 't' = two
veil-color	Color of the veil.	Categorical	'n' = brown, 'o' = orange, 'w' = white, 'y' = yellow
veil-type	Type of veil covering the mushroom.	Categorical	'p' = partial, 'u' = universal
stalk-color-below-ring	Color of the stalk below the ring.	Categorical	'n' = brown, 'b' = buff, 'c' = cinnamon, 'g' = gray, 'o' = orange, 'p' = pink, 'e' = red, 'w' = white, 'y' = yellow
stalk-color-above-ring	Color of the stalk above the ring.	Categorical	'n' = brown, 'b' = buff, 'c' = cinnamon, 'g' = gray, 'o' = orange, 'p' = pink, 'e' = red, 'w' = white, 'y' = yellow

stalk-surface-below-ring	Texture of the stalk below the ring.	Categorical	'f' = fibrous, 'y' = scaly, 'k' = silky, 's' = smooth

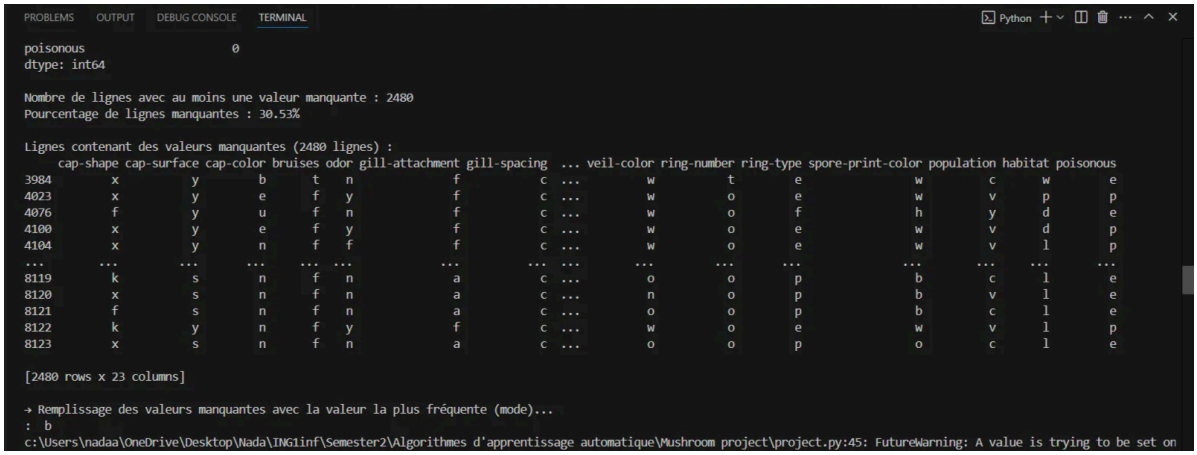
2. Data cleaning :

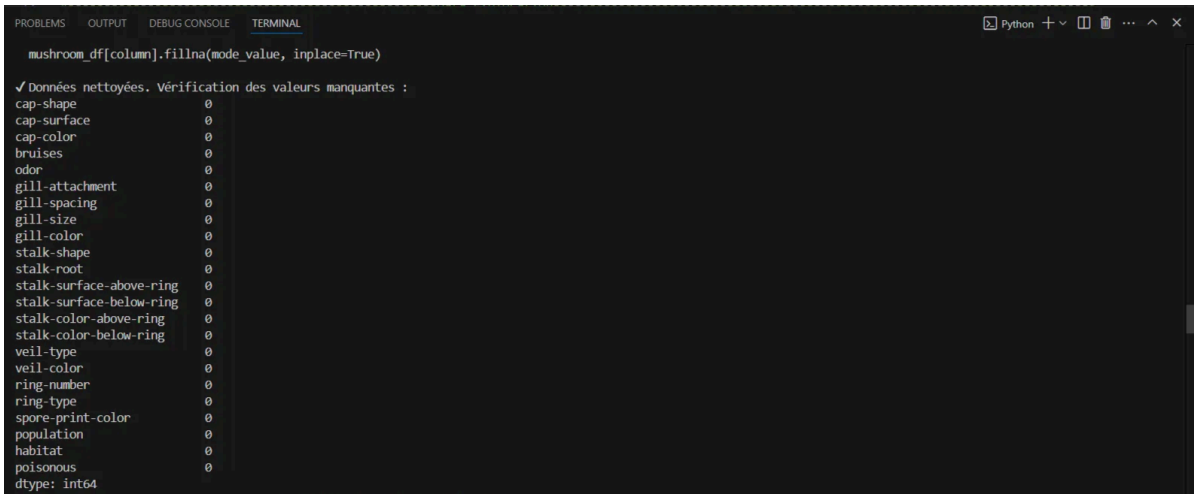
- Missing values :



Categorical variables :

we have calculated the percentage of the lines with inconsistent/ wrong data, we found out that it represents >5% of the dataset, so it'd be better if we replace them with the most frequent value instead of deleting them.





- Duplicates :

  The dataset doesn't have duplicate values :

3. Data Exploration :

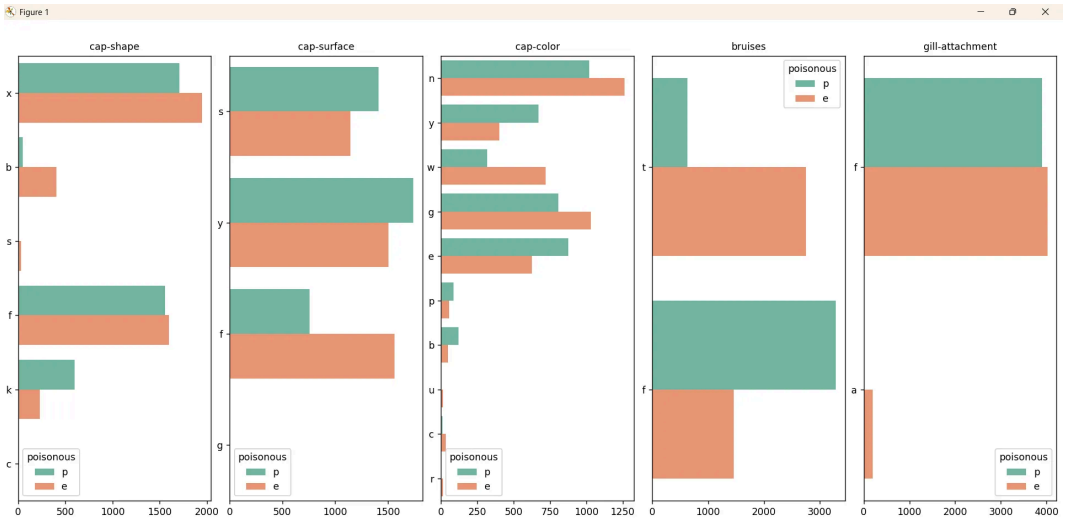
Data summary :

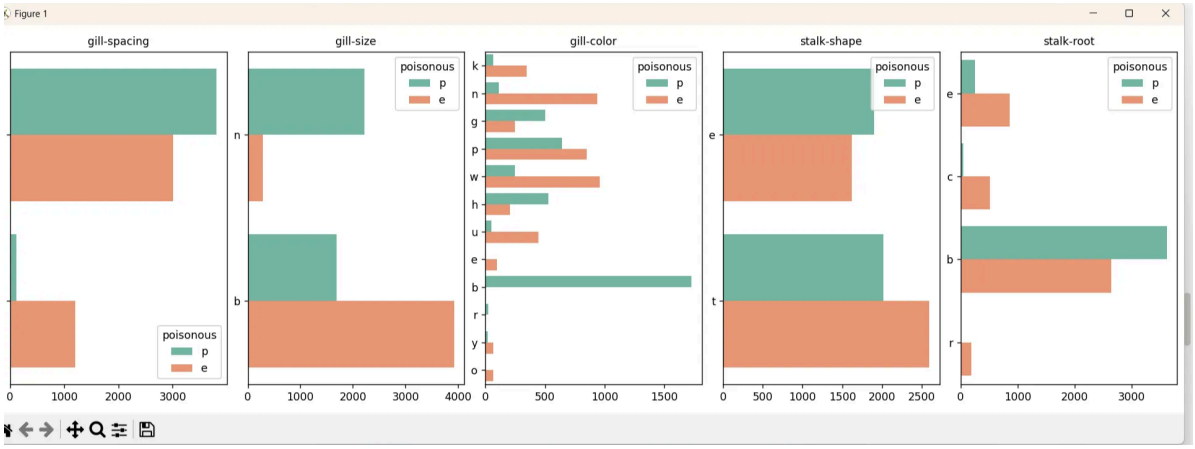
⇒ The number of non-null entries is the same among all variables because we have cleaned the data.

⇒ The most frequent shape of mushrooms is convex (x) , the most frequent mushroom color is brown.

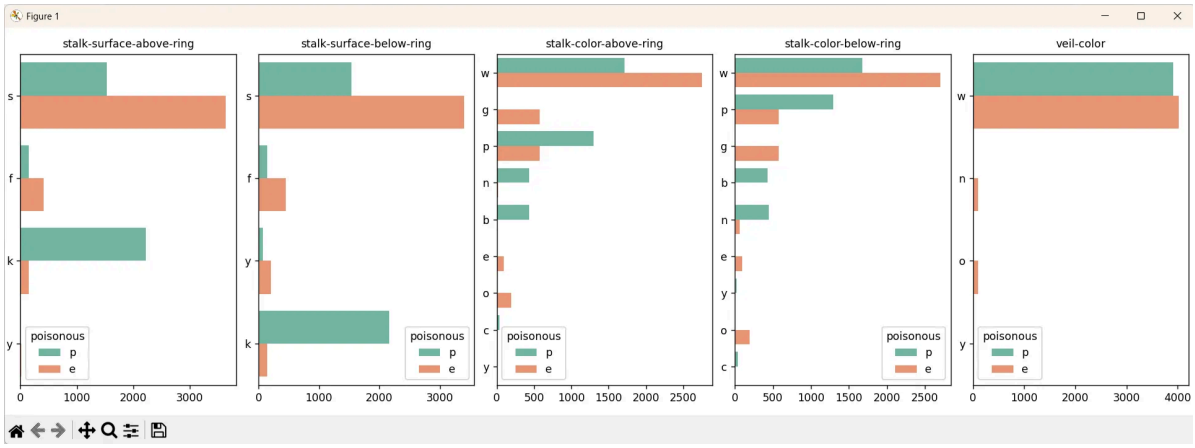
⇒ More than half of the mushrooms in the dataset are edible (4208 having the value "e") and the other portion are poisonous.

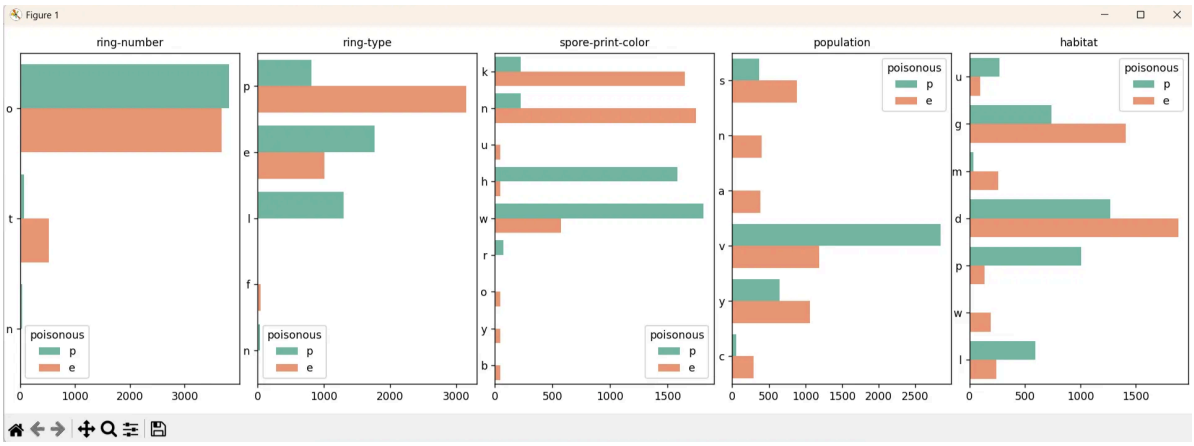
Count plots :



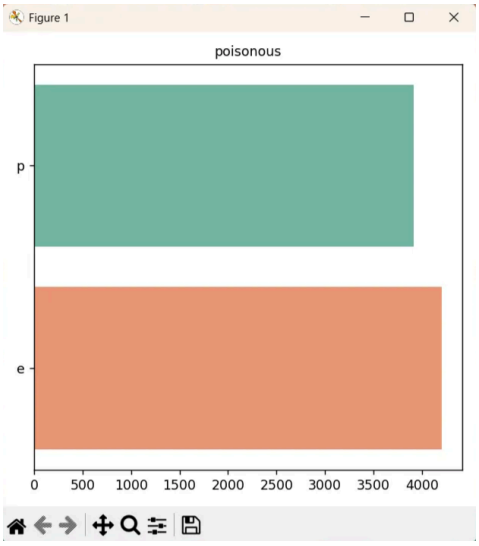


⇒ Mushrooms having buff (buff=b) as their gill-color are likely to be poisonous.





⇒ When the natural habitat of the mushrooms is woods (woods=d) , they have a higher chance of being edible rather than poisonous.



Variables that we're considering removing :

- Type of veil covering the mushroom in this dataset is partial . So it doesn't add much value in this dataset .

⇒ we're dropping veil-type .

- Odor is a pure attribute that gives the class based on the given attribute .

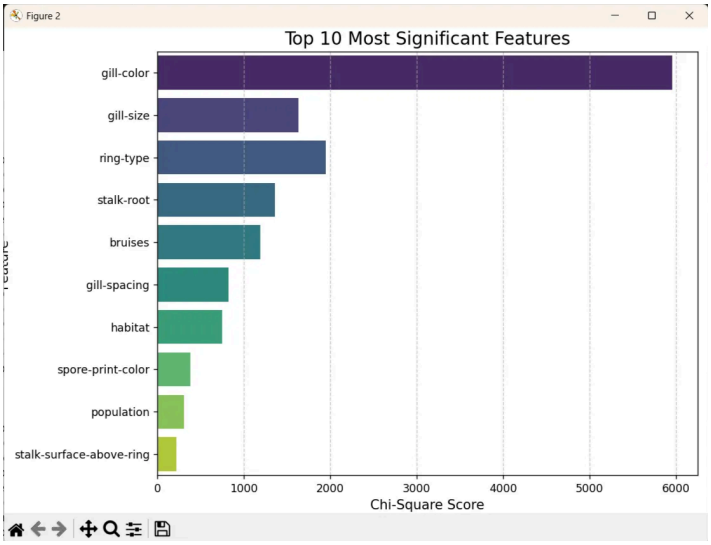
⇒ we're going to get rid of odor to make the model more complex.

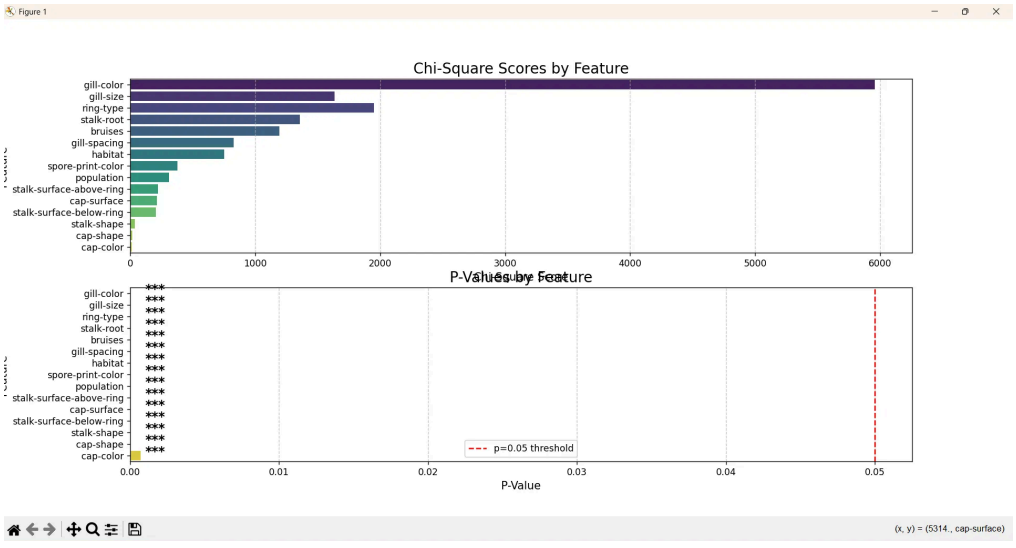
- ring-number : has limited categories (mostly '0' with some 't' and 'n') and less distinctive separation between poisonous and edible classes compared to other variables.
- veil-color has one type mostly 'w' which is the predominant category for both poisonous and edible mushrooms.
- gill-attachment we can almost distinguish a pattern across poisonous and edible.

⇒ we're dropping gill-attachement.

- Stalk-color-above-ring / stalk-color-below-ring: These features show a degree of variation between edible and poisonous mushrooms, but the patterns are less distinct compared to other stalk-related features. They don't stand out as highly informative on their own.

4. Feature selection : Q2 Test





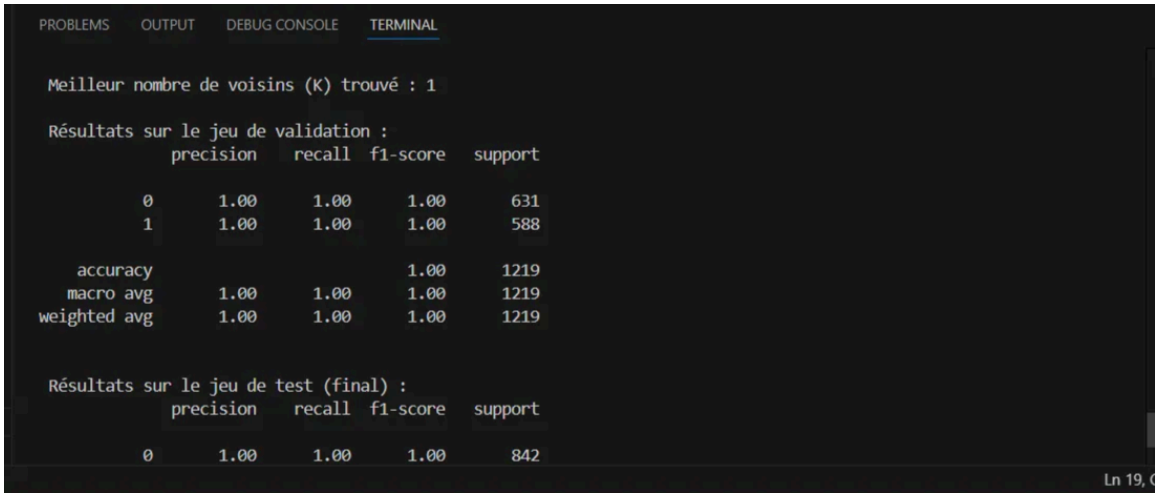
- ***** : Very significant: p < 0.001** Strong evidence of a relationship between the features & the target variable.
- **** : Significant p < 0.01** Moderate to strong evidence.

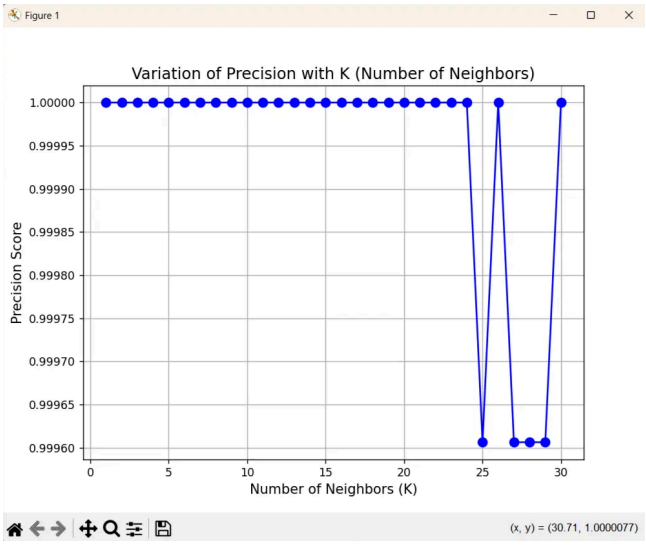
- * : **Weakly significant** **p < 0.05** Some evidence, worth considering.
  ⇒ So if we don't see these indicators , it'd be better if we drop those variables which is the case here in the variable cap color .

5. Model selection :

KNN :

- Hypertuning :
  - We’re using GridSearchCv **(cross validation)** and it picks the best parameter based on **precision** . It picks the K that provides **the best precision in the class poisonous** since errors consisting of classifying a poisonous mushroom as edible is more dangerous than classifying an edible mushroom as poisonous.





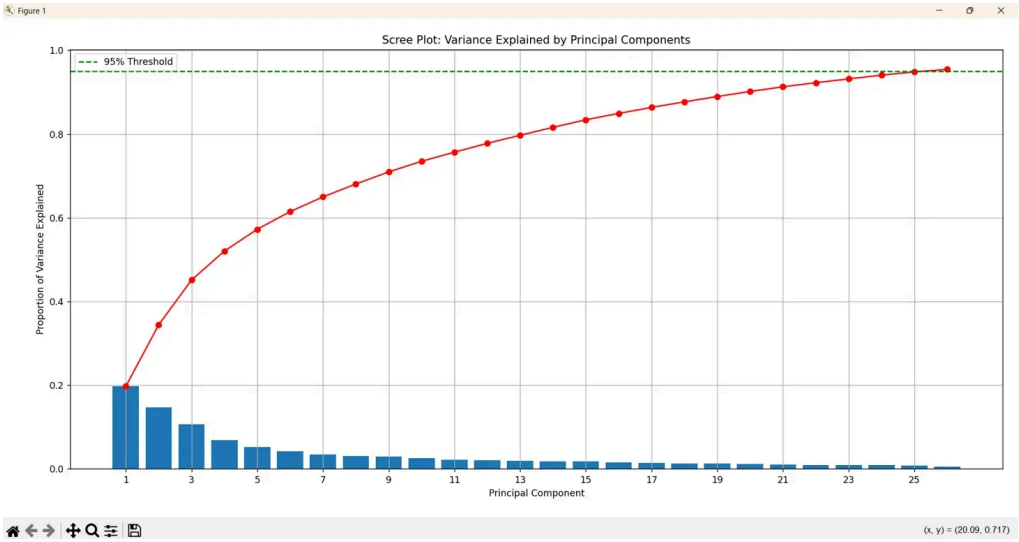
- Distance metric:

- Since our variables are categorical and nominal we need to transform them before calculating their distance ;:
  - One label encoder : One label encoder creates meaningless distance between variables having different values of an attribute because they’re not ordinal but it’s useful if we’re using random forest tree.
  - One hot encoder : Even though the precision is 1.00 in our case , using one hot encoder makes KNN ineffective due to curse of dimensionality .
- the chosen distance metric here is Manhattan after labeling our features with one hot encoding.

Solution :

- We already have 13 features left, one average each feature has 4 samples , as a result the dimension of a single point after using this distance metric is 13∗4=52 .
  - Exemple : **[0,1,0,0,1,1,0,1,0,0,1,1,1,0,0,1,0,1,1,0,0,0,1,1,1,0,0,0,1,0,1,0,1,1,1,0,1,0,0,0,1,0,1,0,1,0,0,1,1,0,1,0]**

⇒ We apply PCA (Principal Component Analysis) **to reduce dimensions** while **preserving key variance.** PCA transforms high-dimensional data into a smaller set of orthogonal components that capture most of the original variance. Here, we set the number of components to retain the 10 most important features.

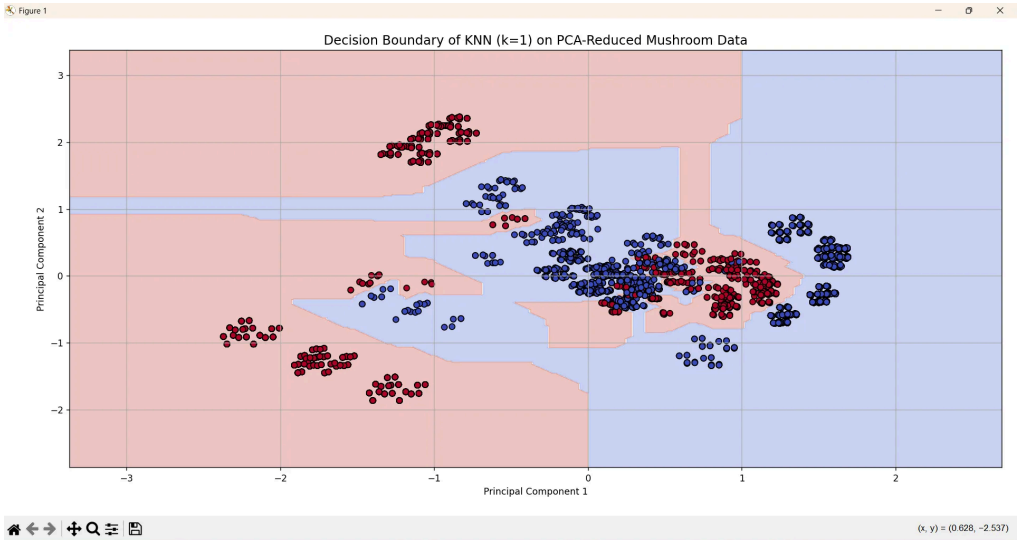


⇒ The Scree Plot shows that around 25 components capture 95% of the variance, with the first component explaining 20%. The cumulative variance rises steeply in the first 5 components (60% variance) before leveling off, suggesting that the dataset’s information is spread across multiple features.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                  Python

  sns.barplot(x='Chi2 Score', y='Feature', data=top_features, ax=ax3, palette='viridis')
Number of components selected to retain 95% variance: 26
Total variance retained: 0.9544

Top features contributing to each principal component:

Top 10 features for Principal Component 1:
ring-type_p                0.337324
bruises_f                  0.332729
bruises_t                  0.332729
stalk-surface-above-ring_k 0.244801
stalk-surface-above-ring_s 0.242023
stalk-surface-below-ring_k 0.239635
spore-print-color_w        0.234478
gill-color_b               0.230135
ring-type_e                0.228577
stalk-surface-below-ring_s 0.221794
Name: 0, dtype: float64

Top 10 features for Principal Component 2:
stalk-shape_t              0.324529
stalk-shape_e              0.324529
spore-print-color_h        0.285619
ring-type_l                0.282459
gill-size_b                0.251709
gill-size_n                0.251709
ring-type_e                0.240091
gill-color_b               0.227521
stalk-surface-below-ring_s 0.226043
stalk-surface-above-ring_s 0.209748

Ln 225, Col 38  Spaces: 4  UTF-8  CRLF  {} Python  3.13.0
```

⇒**Principal Component 1:** This part explains 20% of the differences between the mushrooms. It focuses on things like the type of ring on the mushroom, whether it has bruises, and the texture of the stalk (whether it's scaly or silky). These features are really important for telling different mushroom types apart.

Decision boundary plot :



⇒ The decision boundary plot shows how the KNN model (k=1) separates edible mushrooms (blue) from poisonous ones (red) using just the two most important features. And even if we're only using 2 of the most important features , the separation is done successfully with minor errors.

Time complexity :

- Since we're using manhattan as a distance metric, the Algorithm used in sklearn is using BallTree **by default** . Hence ,there's no need for additional configurations. Because BallTree /

KdTree are the best algorithms in terms of time complexity .

Data Structure	Best Case	Worst Case	Dimensions (d)	Notes
Brute-force	O(n)	O(n)	Any	No optimization, always checks all
KD-Tree	O(log n)	O(n)	**d ≤ ~20**	Only for **Euclidean**-based metrics
Ball Tree	O(log n)	O(n)	**d > 20**, any Lp metric	Better for **non-Euclidean** or high-dim
Approximate NN (e.g. Annoy, FAISS)	O(log n)	Approximate	Any	Useful for huge datasets

DECISION TREE :

Decision Trees are not sensitive to feature scaling or high dimensionality. They **naturally select features** by splitting on the most informative attributes.

⇒ We do not need distance metrics, encoding or reducing dimensionality.

Process:

We used a pipeline combining preprocessing and a decision tree to classify the data. After an initial training, we evaluate the model's performance on a validation set to detect potential overfitting. Then, we apply GridSearchCV to test different model configurations using cross-validation and automatically select the best one. The optimized model is re-evaluated on the validation set, and finally on a test set to assess its real-world performance.

Best given parameters : (prepruning)

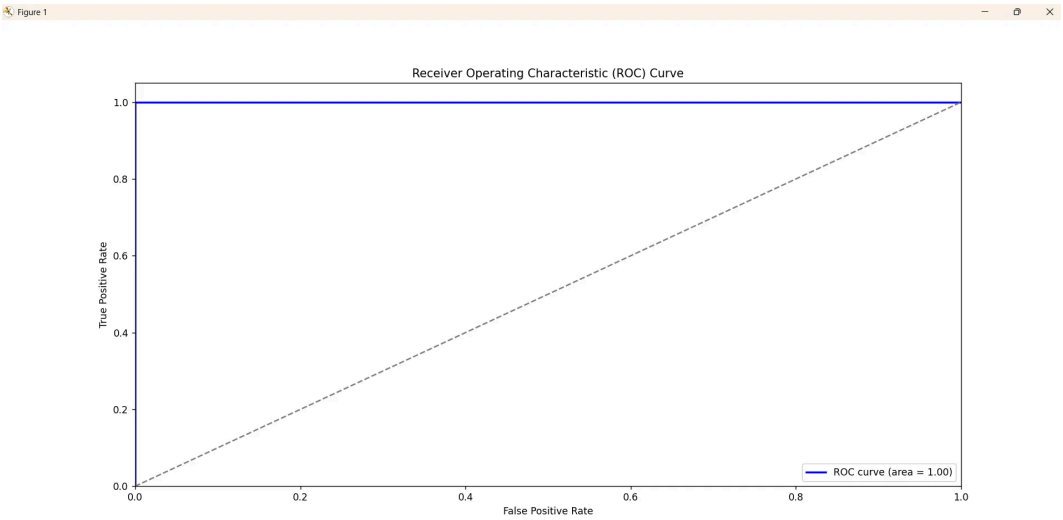
Performance across validation set, test set and training set (of the best model found by grid search) :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                                    Python

weighted avg       1.00      1.00      1.00      1219

Best parameters: {'classifier__criterion': 'gini', 'classifier__max_depth': 10, 'classifier__min_samples_leaf': 1, 'classifier__min_samples_split': 2}
Validation accuracy: 1.0

Validation Set Performance:
              precision    recall  f1-score   support

           e       1.00      1.00      1.00       631
           p       1.00      1.00      1.00       588

    accuracy                           1.00      1219
   macro avg       1.00      1.00      1.00      1219
weighted avg       1.00      1.00      1.00      1219


Test Set Performance (Final Evaluation):
              precision    recall  f1-score   support

           e       1.00      1.00      1.00       842
           p       1.00      1.00      1.00       783

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625

PS C:\Users\nadaa\OneDrive\Desktop\Nada\ING1inf\Semester2\Algorithmes d'apprentissage automatique\Mushroom project>
                                                  Ln 385, Col 38  Spaces: 4  UTF-8  CRLF  {} Python  3.13.0
```

⇒ The validation set gave a score of 1 to verify that it generalizes well beyond the cross-validation data. Since the performance remains high, this confirms that our model is robust and reliable.

The test set provided a score of 1.
⇒ The model's high performance is not just limited to the training and validation sets, but can generalize effectively.

Roc curve :



⇒ The AUC is 1 which can indicate either the model's performance is high or it is overfitting.

Postpruning :

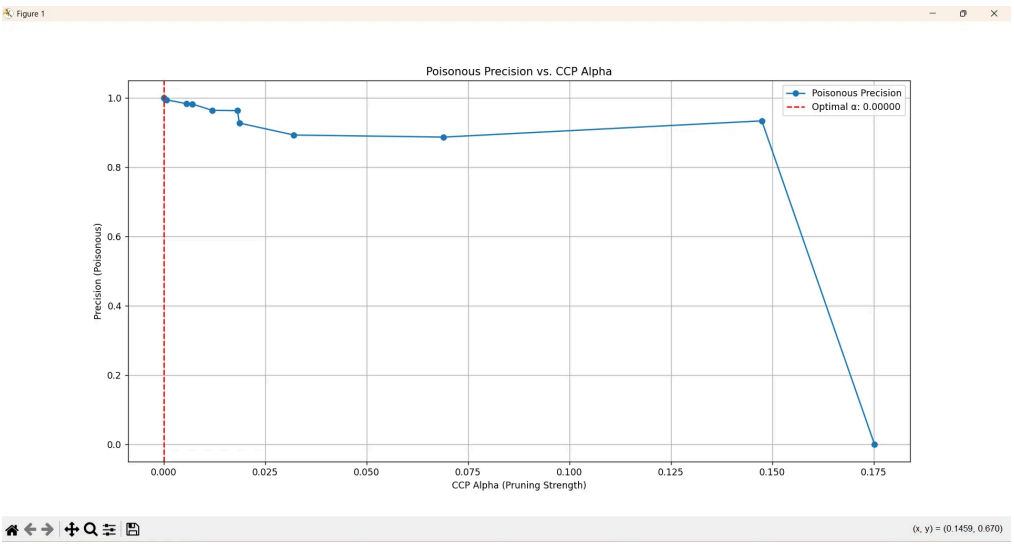
Our model shows an already good performance (on validation & test sets) . But, we're going to use post pruning just to:

1. verify whether or not it's overfitting.

2. reduce the tree's complexity and improve its interpretability without sacrificing performance.

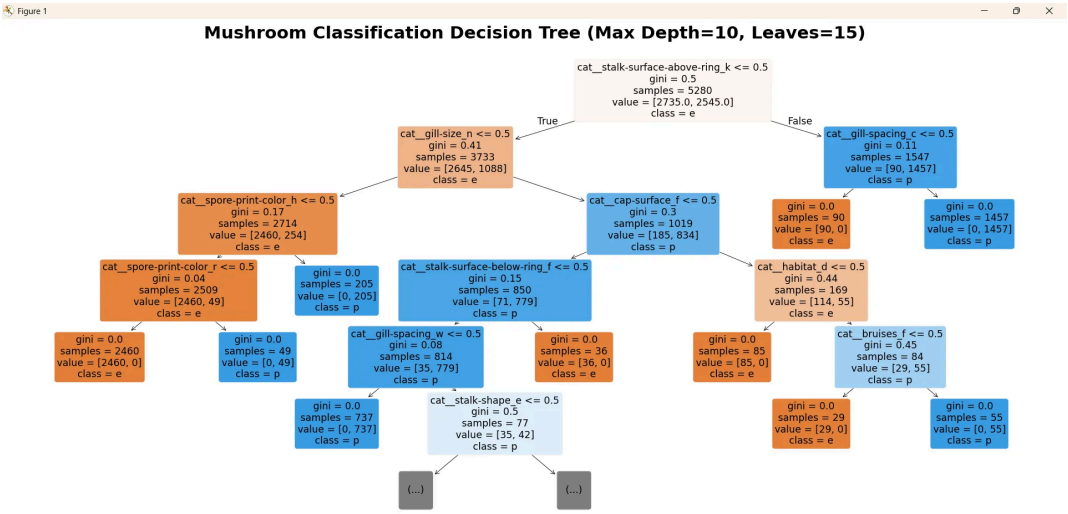
Explanation :
smaller tree (e.g. depth 6 instead of 10 out of 13 features) is easier to interpret, faster, and less likely to fail on
new, unseen data. Simpler models generalize better.

If post-pruning reduces tree complexity
without losing accuracy, it **confirms that we were overfitting before**, even if it wasn't visible yet. If accuracy drops, it tells us that the current model complexity is necessary.

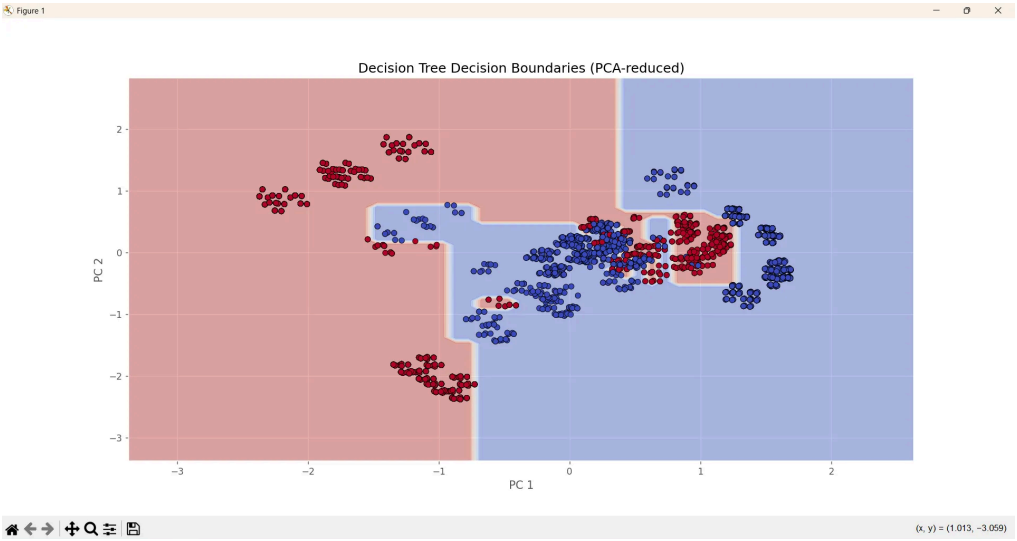


⇒The current model complexity is necessary, and we're not overfitting.

Plot the tree :



Decision boundary :



Decision path :

```
Decision path for sample 0:
Node 0: (X_test[0, 'cat__stalk-surface-above-ring_k'] = 1.0) > 0.5
Node 26: (X_test[0, 'cat__gill-spacing_c'] = 1.0) > 0.5
Node 28: (X_test[0, 'cat__habitat_u'] = 0.0) > -2.0
PS C:\Users\nadaa\OneDrive\Desktop\Nada\ING1inf\Semester2\Algorithmes d'apprentissage automatique\Mushroom project>
```

Picking the best model :

Models to choose :

- A KNN model (best K=1 from grid search)
- A Decision Tree model (pre-pruned version)

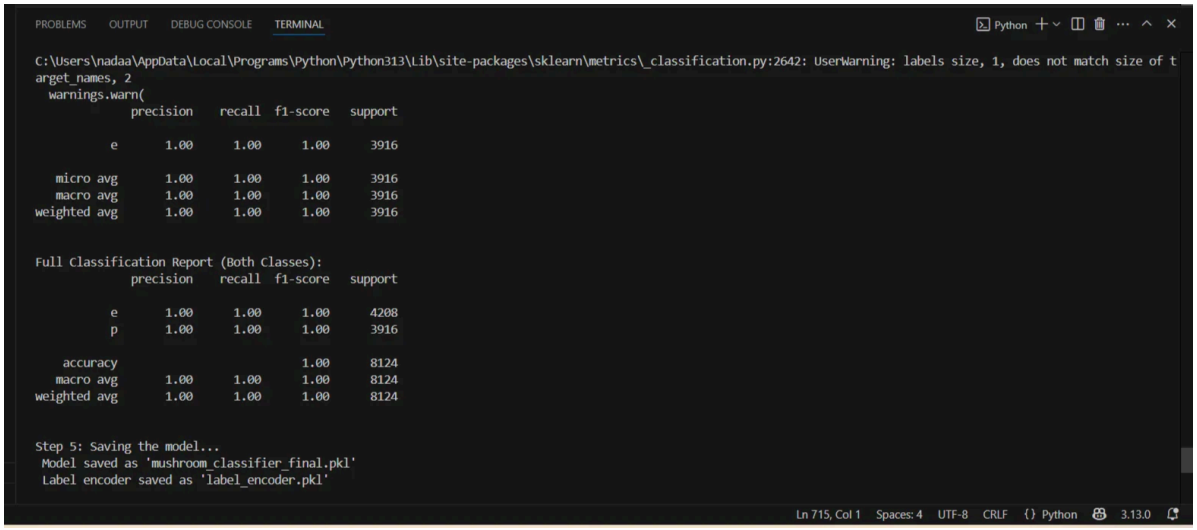
Which to choose?

- Our decision tree model achieved perfect accuracy (1.00) with pre-pruning
- The KNN also performed perfectly but might be less interpretable.
- Since both perform perfectly, the decision tree is likely better because:
  - More interpretable
  - Faster predictions
  - Less sensitive to the curse of dimensionality

⇒Since both models achieve perfect accuracy, the simplest solution is best - use the single decision tree. The ensemble or a more sophisticated solution won't provide additional benefits in

this case and adds complexity.

verifying the model on all of the dataset & saving it :



Test sample of the model :