Abstract

This project explores the Mushroom Overload dataset using a full data-mining workflow that includes supervised learning, unsupervised clustering, and statistical association methods. I worked with a large dataset with mostly categorical features and applied a wide range of analytical techniques (logistic regression, Naive Bayes, decision trees, random forests, FP-Growth, Chi-square tests, Cramér's V correlations, and K-Modes clustering) to understand how mushroom traits relate to toxicity (aka edible vs. poisonous). After data cleaning and preparation, each method offered a different perspective on the structure of the dataset, allowing me to detect which features carry the strongest predictive signal and how the patterns emerge across models.

A consistent theme appeared throughout the analysis: the dataset contains strong and stable categorical relationships that make the edible/poisonous classification highly separable. Tree-based models achieved almost perfect performance, association rules identified toxic signatures, and even unsupervised clustering recovered the toxicity structure without labels. Completing this project gave me practical experience applying diverse data-mining methods, interpreting their outputs, and comparing their strengths in a real analytical workflow. The results collectively show that the Mushroom Overload dataset is not only highly predictive but also an excellent environment for developing and reinforcing core data-analysis skills.

1. Introduction

This project is an exercise on practicing data mining and predictive modeling techniques using the Mushroom Overload dataset. The primary goal is to apply a variety of supervised and unsupervised learning methods to explore relationships between mushroom features and class labels to develop experience, and hopefully, some proficiency in real-world model evaluation and optimization. The dataset's structure, containing both categorical and numeric variables, makes it suited for experimenting with different algorithms and learning how to handle large-scale, mixed-type data efficiently.

2. Dataset Description

The dataset used for this project is the Mushroom Overload dataset published on Kaggle by user **bwandowando.** It contains approximately 6.7 million synthetic records describing mushrooms through categorical features such as cap shape, odor, gill color, stalk type, and habitat. The dataset was created as part of a synthetic data competition and is labeled as “synthetic data (SD)” rather than real collected data from observations. While the Kaggle page does not include technical details on the data-generation process, the dataset clearly follows the structure of a well-known University of California Irvine (UCI) Mushroom Dataset (1987), which it replicates and scales up to a much larger size.

The original UCI Mushroom dataset, donated to the UCI Machine Learning Repository in 1987, contained 8,124 instances across 22 categorical features. It was based on descriptions of 23 species of gilled mushrooms from the *Agaricus* and *Lepiota* families taken from *The Audubon Society Field Guide to North American Mushrooms* (Lincoff, 1981). The UCI data was never a record of field measurements but instead consisted of hypothetical samples constructed from species descriptions to illustrate morphological variation. Each record was classified as edible or poisonous, with “unknown” edibility cases merged into the poisonous category. This dataset has always represented a simplified, educational version of a real biological problem rather than empirical field data.

In 2021, Wagner, Heider, and Hattab published “Mushroom data creation, curation and simulation to support classification tasks” (Scientific Reports, 2021). Their work describes a reproducible pipeline for generating large-scale simulated mushroom datasets. They began with descriptive data from field guides and created primary structured data that mirrored the UCI format. From this starting point, they generated secondary synthetic data by random sampling of feature distributions, maintaining plausible combinations of attributes. This approach provides an important reference for understanding how large synthetic mushroom datasets can be created while preserving internal consistency and the categorical relationships seen in the original UCI mushroom dataset.

Although the exact procedure for Mushroom Overload is not documented, it is reasonable to assume that it follows a similar process: starting from the categorical feature distributions of the 1987 UCI dataset and expanding from there to produce an artificial dataset. By creating a dataset with millions of rows, the author likely intended to provide a richer, statistically varied dataset suitable for machine learning/data mining training.

3. Data Preparation

The original dataset was downloaded from Kaggle as a compressed archive (archive.zip) containing a single CSV file (mushroom_overload.csv) of approximately 297 MB. The dataset was extracted and inspected in Jupyter Notebook using pandas for initial examination.

Because the dataset contains 6.7 million rows and 21 columns, efficient memory management was necessary. The CSV initially required over 5 GB of RAM when fully loaded. To optimize performance, all text-based (object) columns were converted to the pandas category type, which replaces repeated strings with integer codes while preserving their original labels. This change reduced memory usage from 5 GB to approximately 270 MB, without any loss of information or meaning in the categorical features.

After optimization, the entire dataset was saved in the Parquet format using the fastparquet engine. Parquet is a binary, columnar storage format that compresses data, preserves data types, and allows fast loading for future use.

4. Exploratory Data Analysis (EDA)

A review of missing values showed that several categorical features in the Mushroom Overload dataset had extremely high levels of missingness, with many columns missing between 60% and 95% of their values. Variables such as **veil-type**, **veil-color**, **stem-root**, **stem-surface**, **gill-spacing**, **gill-attachment**, **cap-surface**, and **spore-print-color** contained so many absent rows that their statistical reliability would be compromised in analysis. Because of this, I removed these eight variables from the dataset and retained only ring-type, which had comparatively minimal missingness and still provides some predictive value. This approach simplifies the model and ensures that the remaining predictors are based on sufficiently complete information to support meaningful analysis.

To evaluate how the continuous features relate to the edible/poisonous target, I computed the Pearson correlation between each numeric variable and a binary version of the class label (edible = 1, poisonous = 0). Pearson correlation measures the strength and direction of a linear relationship, with values ranging from −1 to +1. A positive coefficient means that larger values of the variable are associated with a higher likelihood of being edible, while a negative coefficient means that larger values correspond to a higher likelihood of being poisonous. Values close to 0 indicate little to no linear relationship between the feature and the target. Because the target is binary, these correlations are also interpreted as point biserial correlations, which are commonly used to assess how continuous measurements differ between two groups. Overall, this step helps identify whether stem height, stem width, or cap diameter provide meaningful linear separation between edible and poisonous mushrooms before modeling.

To understand which features have the strongest relationship with the edible/poisonous classification, I computed Cramér’s V between each categorical feature and the target variable. Cramér’s V is a statistical measure of association for categorical variables that ranges from 0 to 1, where values near 0 indicate little to no relationship and values near 1 indicate a very strong association. Unlike simple frequency tables, Cramér’s V adjusts for both the size of the contingency table and the distribution of categories, making it a good indicator of how strong each feature is associated with the target. By calculating Cramér’s V for each predictor individually, I obtained a ranked list showing which mushroom characteristics are most informative for distinguishing edible mushrooms from poisonous ones. Features with high Cramér’s V values carry the strongest predictive signal and are likely to be influential in the classification model, while features with low values contribute little meaningful information and may be redundant or irrelevant.

Overall, the EDA showed that both the continuous and categorical features in the cleaned Mushroom Overload dataset exhibit mostly weak individual relationships with the edible/poisonous target. Pearson correlations for the three continuous variables (cap-diameter, stem-height, and stem-width) were all below 0.20, indicating minimal linear separation between edible and poisonous mushrooms. Categorical variables, assessed using Cramér's V, demonstrated only moderate associations. The strongest predictors were stem-color and cap-color (≈0.25), followed by ring-type, gill-color, habitat, and cap-shape, all of which showed weak relationships with the target . Features such as has-ring and does-bruise-or-bleed showed almost no association, suggesting limited predictive value on their own. Together, these findings indicate that while no single feature strongly distinguishes edible from poisonous mushrooms, a logistic regression model may still extract useful signal by combining many weak predictors across multiple dimensions.

Logistic Regression

Why I Used Logistic Regression

I started with logistic regression because it gives me a clean, interpretable baseline model for binary classification. The goal is to predict whether each mushroom is edible or poisonous and logistic regression is a natural first step. LR shows me how far a simple linear decision boundary can go before I move on to more flexible models. Logistic regression also integrates naturally with one-hot encoded categorical features and is widely used as a benchmark in large, mixed-type datasets. My goal with this section was to establish a reference point for performance before applying more complex models like decision trees or random forests.

Understanding the Classification Report – How Results Are Presented

The classification report provides four key metrics that describe a model's performance for each class:

- **Precision** measures how often the model is correct when it predicts a given class (low precision → many false positives)

- **Recall** measures how well the model identifies all true instances of a class (low recall → many false negatives)
- **F1-score** is the harmonic mean of precision and recall, giving a balanced view of predictive performance
- **Support** is simply how many samples of each class appear in the test set

Together, these metrics help me understand how well the model distinguishes edible mushrooms (1) from poisonous mushrooms (0) and where it struggles.

Baseline Logistic Regression Interpretation

I started by training a baseline logistic regression model, which simply means fitting the algorithm with minimal preprocessing and default parameters so I can establish a reference point to compare all later models against. The baseline results are very typical for this dataset after removing the high-missingness features. The confusion matrix showed that the model performs moderately well on poisonous mushrooms, with a recall of **0.77**, but it struggles more with edible mushrooms, achieving a recall of **0.71.**

This imbalance shows a key limitation: logistic regression has difficulty separating the two classes because the remaining features only carry **weak individual signals**. Most categorical features show modest associations with toxicity (Cramér’s V ≈ 0.25 at best), so the linear boundary the model is trying to learn is working with limited structure. The baseline is a solid starting point but clearly leaves room for improvement through tuning or nonlinear models.

Test Accuracy Interpretation

The baseline logistic regression model achieved a test accuracy of **0.74**, meaning it correctly classified 74% of mushroom samples. This is reasonable for a first model.

Hyperparameter Tuning Overview

To push the model further, I used **GridSearchCV**, which automatically evaluates different combinations of hyperparameters and chooses the best one using cross-validation.

I tuned two parameters:

- **C** – the regularization strength
- **solver** – the optimization algorithm used during training

Testing different C values helps me see whether the model benefits from stronger or weaker regularization. Trying different solvers matters because one-hot encoding creates a large, sparse feature matrix, and some solvers handle that structure better than others.

I used **3-fold cross-validation**, meaning the training set was split into three parts. The model trained on two parts and validated on the third, rotating through all the splits. The best average score determined the best hyperparameters.

Why I Tuned Only These Parameters

I focused on **C** and **solver** because they have the largest impact on performance for logistic regression on sparse, high-dimensional data.

- **C** influences overfitting vs. underfitting
- **solver** determines whether the optimizer can efficiently handle the large one-hot encoded matrix

Keeping the search space small makes tuning computationally manageable while still giving the model a chance to improve.

Hyperparameter Tuning Results

The tuned model improved accuracy only slightly—from **0.7420** to **0.7445**.

This extremely small gain tells me that the **feature space** is the limiting factor, not the hyperparameters. Since the remaining variables each carry weak signal individually, a linear model simply cannot extract much more predictive structure from them.

Precision, recall, and F1-scores remained nearly identical, reinforcing that logistic regression was already doing everything it realistically could with this dataset.

This is normal and expected.

It also suggests that nonlinear models, like trees or ensembles, are much more appropriate for this problem.

Baseline vs. Tuned Logistic Regression

Metric	**Baseline**	**Tuned**
Test Accuracy	0.7420	0.7445
Recall (Poisonous, 0)	0.77	0.78
Recall (Edible, 1)	0.71	0.71
Precision (Poisonous, 0)	0.76	0.76
Precision (Edible, 1)	0.72	0.72
F1 (Poisonous, 0)	0.77	0.77
F1 (Edible, 1)	0.71	0.71

Comparison Summary

The tuned logistic regression model barely improved over the baseline. Accuracy increased by **0.0025**, and only the poisonous class recall shifted noticeably.

This tells me:

- The model has reached the **ceiling** of what a linear decision boundary can achieve here
- The categorical features do not carry enough linear structure for logistic regression to perform at a high level
- More flexible, nonlinear models (such as decision trees, random forests, or gradient boosting) are better suited to capture the interactions and nonlinear relationships that clearly exist in the data

In short, logistic regression served its purpose as a clean, interpretable baseline, but it is not the optimal modeling choice for this dataset.

Naive Bayes Classification

After establishing the logistic regression results, I wanted to train a Naive Bayes model as a second baseline. Naive Bayes is a family of extremely fast probabilistic classifiers that use Bayes' Theorem to estimate how likely a sample belongs to each class based on its features. What makes it unique is the "naive" assumption that all features are conditionally independent given the class. Even though this assumption is almost never true, Naive Bayes still performs well in high-dimensional, sparse, categorical datasets, which makes it a useful check and a strong baseline to compare more flexible models against.

Why I Used Bernoulli Naive Bayes

I chose **BernoulliNB** because my preprocessing pipeline converts all categorical variables into one-hot encoded binary indicators. BernoulliNB is designed for exactly this kind of data: feature columns represent whether a category is "present" or "absent," and the model learns simple probability-based patterns from those indicators. It is very fast on large datasets and fit well with my existing pipeline. Even though Naive Bayes makes very strong independence assumptions, it provides a fast, interpretable way to measure how much signal is available in the binary features before moving on to more complex models.

Why I Kept the Continuous Features

Even though BernoulliNB is traditionally used with binary data, scikit-learn's implementation can handle non-binary numeric values without any issues. My pipeline standardizes the three continuous variables using **StandardScaler**, and Naive Bayes treats the resulting values as continuous signals in its probability calculations. Keeping these features helps with consistency across models for comparison.

Results and Interpretation

Bernoulli Naive Bayes achieved a test accuracy of **about 0.686**, which is noticeably lower than logistic regression. Even after a light hyperparameter search (tuning *alpha* and *fit_prior*), the best model only reached **≈0.688**. The lack of improvement makes sense: the model's rigid independence assumptions prevent it from capturing the multi-attribute interactions that appear throughout this dataset. Most features only offer weak individual signals, and Naive Bayes is unable to combine them in a nonlinear way.

Despite its simplicity, Naive Bayes confirmed an important point:
there is structure in the data, but extracting it requires models that can learn interactions.

Random Forest Classification

After exploring the linear and probabilistic baselines, I moved on to a more flexible model: a **Random Forest classifier.** I chose Random Forests because they handle nonlinear relationships, complex feature interactions, and large categorical datasets without requiring heavy preprocessing. They also provide feature importance scores, which helped me understand which traits contribute the most to my toxicity prediction.

Changes to the Preprocessing Pipeline

Tree-based models do not require scaling or one-hot encoding. One-hot encoding can increase dimensionality and slow down training without providing any benefit to decision trees. Because of that, I updated my preprocessing pipeline to use an **OrdinalEncoder** for the categorical variables and passed the numeric variables through unchanged. This keeps the feature matrix compact and allows the Random Forest to split directly on the encoded category values. It also ensures that the model focuses on the structure of the data rather than a massive sparse matrix.

Model Configuration

To build a strong baseline, I configured the Random Forest with:

- **n_estimators = 200**
- **max_depth = None**
- **bootstrap = True**
- **n_jobs = -1** to use full parallelization on my machine

Results

The model’s performance was extremely strong. On the test set, the Random Forest achieved:

- **Accuracy ≈ 0.9982**
- **Precision = 1.00** (rounded)
- **Recall = 1.00** (rounded)
- **AUC ≈ 0.9999**

The confusion matrix showed only a very small number of errors out of more than **1.34 million** test samples. The ROC curve hugged the top-left corner, indicating almost perfect separability between edible and poisonous mushrooms.

Feature Importance Interpretation

The feature importance plot revealed a clear hierarchy among the predictors. The strongest features included:

- **stem-width**
- **gill-color**
- **stem-height**
- **stem-color**
- **cap-diameter**

Other features such as **cap-color**, **cap-shape**, and **ring-type** also had high influence. Variables, such as **habitat** and **season,** contributed smaller but still meaningful amounts of signal. Features like **does-bruise-or-bleed** and **has-ring** had moderate contributions.

Interpretation

This model confirms that the mushroom dataset has **very strong and very stable categorical signals** tied to toxicity. The Random Forest was able to capture these signals with almost no ambiguity, producing very good classification results. The consistency between its feature importances and the earlier EDA findings gives me confidence that the relationships in the dataset are legit rather than artifacts of preprocessing.

Overall Takeaway

Random Forest ended up being one of the strongest models in this entire project. It handled the categorical structure smoothly, required minimal preprocessing, and uncovered the dominant interactions that determine the target classification. The near-perfect accuracy demonstrates that the dataset is highly separable and that tree-based ensemble methods are exceptionally well-suited for this task.

Decision Tree Classification

After evaluating the Random Forest ensemble, I wanted to train a **single decision tree** to understand how well one interpretable model could perform on this dataset. A decision tree builds a sequence of rule-based splits that try to separate edible from poisonous mushrooms by maximizing class purity at each branch. Unlike an ensemble of trees like random forest, a single tree will show its entire decision process, making it a very transparent model.

Why I Trained a Single Decision Tree

Even though Random Forests provide excellent predictive power, they can be hard to interpret because their decisions are averaged over hundreds of trees. A single decision tree gives me a clearer picture of:

- which features create the strongest splits
- how categorical traits interact
- what kinds of rule patterns actually classify the mushrooms

Because so many of the traits in the dataset are characteristics that are descriptive of the mushrooms anatomy, interpretability can be very useful.

Updates to the Preprocessing Pipeline

Originally, the preprocessing pipeline used **OneHotEncoder** and **StandardScaler**, which made sense for logistic regression and Naive Bayes. But one-hot encoding vastly increased the number of features into the thousands, and a single decision tree would need to evaluate all those potential splits. That would cause serious run time issues on my machine.

To fix this, I rebuilt the pipeline using:

- **OrdinalEncoder** for all categorical features
- **passthrough** for numeric features

Decision trees do not require scaling or one-hot encoding, so this version is much more efficent. With this pipeline, the model trained cleanly on over **1.3 million rows** in a little over a minute.

Results

The decision tree reached an accuracy of **0.99697,** which is extremely close to the Random Forest's performance. The classification report showed:

- **Precision = 1.00**
- **Recall = 1.00**
- **F1-scores = 1.00** (rounded for both classes)

The confusion matrix confirmed almost no misclassifications across the entire test set of **1,344,624 samples.**

Interpretation

These results tell me a lot about the structure of the mushroom dataset:

1. **The signals are extremely strong.**
   A single decision tree, which is prone to overfitting, was able to nearly match the Random Forest. That means the boundary between edible and poisonous mushrooms is very clean and stable.
2. **The key features are visually meaningful.**
   Features like **gill color**, **ring type**, **stem and cap properties**, and **habitat** consistently drive the splits. These same features appeared repeatedly in the Random Forest.
3. **The dataset is highly separable.**
   There isn't much noise, and the traits that define toxicity are categorical and sharply partitioned. The tree wasn't forced to learn subtle or fragile patterns so the rules practically reveal themselves.
4. **The Random Forest is not rescuing a messy dataset.**
   Instead, it is only smoothing and stabilizing patterns that are already clear. The fact that one tree performs almost as well as 200 trees supports this.

Comparison to the Random Forest

The Random Forest achieved an accuracy of about **0.9982**, while the single tree reached **0.9970.** This tiny difference reveals two things:

- The **true decision boundary is almost perfectly learnable** by a simple model.
- The Random Forest's improvement comes mostly from variance reduction, not from discovering new relationships.

In other words, a single decision tree captures **almost all** the structure in the dataset. The forest just refines it.

Takeaway

The decision tree served its purpose perfectly: it gave me an interpretable view of the underlying structure and confirmed that the mushroom dataset is exceptionally well-behaved and highly separable. For understanding the classification logic, the tree is invaluable. For pushing performance to the limit, the Random Forest is still the better choice but only marginally so given how clean the patterns are.

Association Rule Mining with FP-Growth

After working through the supervised models, I wanted to shift gears and explore the dataset from a descriptive, unsupervised method. Instead of predicting toxicity directly, my goal here was to uncover **co-occurring patterns** within the categorical features and identify combinations of traits that strongly show whether a mushroom is edible or poisonous. Association rule mining gives me a different kind of insight: rather than building a classifier, I get a set of clear "if–then" statements that describe structural relationships in the data.

Why I Chose FP-Growth

For this phase, I used the **FP-Growth** algorithm instead of Apriori. FP-Growth is the modern alternative because it avoids generating a massive number of candidate itemsets. Instead, it compresses the dataset into a compact FP-tree and mines frequent patterns directly from that structure.

I chose FP-Growth because:

- the dataset is *enormous* and almost entirely categorical
- Apriori would have been too slow and memory-intensive
- FP-Growth is specifically designed for large categorical mining

How I Prepared the Data

Association rule mining treats each row as a "transaction" made up of items. To use that format, I had to:

- convert each (feature, value) pair into a token like **cap-color=brown**
- include the target label as an item
- drop numeric features
- convert the dataframe into a list-style format

Switching to a Sample Dataset

My first attempt at running FP-Growth on the full 6.7M rows caused runtime issues. FP-Growth itself completed but generating the association rules from all possible frequent itemsets became unfeasible. There were way too many combinations.

To fix this, I used:

- a **50,000-row stratified sample**
- a slightly higher minimum support threshold
- a maximum itemset length of **2**

Sampling is standard practice in large-scale association mining, and 50k rows is more than enough to hold on to the data's structure.

Results of the FP-Growth Analysis

After applying the constraints (min_support = 0.03, confidence ≥ 0.70, and max_len = 2), FP-Growth produced **two very strong rules**:

Rule 1 – Ring Type

If ring-type = "zone" → poisonous (class_binary = 0)

- Confidence: **1.00**

- Lift: **1.83**
- Support: **3.8%**

This means *every* mushroom with ring-type zone in the sample was poisonous. The lift value tells me that mushrooms with this ring type are almost twice as likely to be poisonous compared to a random mushroom.

Rule 2 – Cap Color

If cap-color = "green" → poisonous (class_binary = 0)

- Confidence: **0.87**
- Lift: **1.59**
- Support: **2.48%**

This indicates that green capped mushrooms are strongly associated with toxicity.

Discussion

These rules line up very well with what I observed earlier: **Random Forest feature importance** emphasized ring-type and cap-color

The fact that an unsupervised pattern mining algorithm revealed the same signals gives me independent validation of the dataset's structure.

FP-Growth also highlights something important: although mushroom toxicity often depends on combinations of traits, **some individual attributes carry disproportionately strong signals.** Ring type and cap color, in particular, are visual cues that mean: DANGER.

Takeaway

The FP-Growth results gave me a clean, easily understandable set of rules that reinforced the patterns seen by the supervised models. The association rules help explain *why* the classifiers work so well they reveal specific categorical traits that directly map to toxicity. Together, FP-Growth and the classification models provide a complementary result: the mushroom dataset contains strong categorical signals, and those signals are consistent across different analytical methods.

Chi-Square Feature Analysis

At this stage of the project, I wanted to step away from model-based measures of importance and evaluate the categorical features using a purely statistical approach. The Chi-square test let me do exactly that. Instead of relying on how a classifier uses a feature, Chi-square measures the **degree of dependence** between each categorical feature and the target variable. This gives me a ranked list of which features show the strongest stand-alone relationship with toxicity, without involving a predictive model at all. Because the mushroom dataset is dominated by categorical variables, this test is a great fit and helps confirm whether the patterns my previous analysis found are also reflected in the raw distributions.

Why I Ran Chi-Square

Up to this point, I had evaluated features through logistic regression coefficients, random forest importances, and decision tree splits. Chi-square adds a different angle: it tells me how much the observed frequencies in each feature deviate from what we would expect if the feature and the class label were totally independent. Large Chi-square values mean a strong association; small values mean a weak or no relationship.

In short, Chi-square helps answer a simple question:
"Which individual categorical features carry the strongest signal about whether a mushroom is edible or poisonous?"

Results

The Chi-square ranking showed a very clear pattern. The strongest associations with toxicity were:

- **cap-shape**
- **ring-type**
- **stem-color**
- **season**
- **habitat**
- **gill-color**

These variables produced large chi-square values, meaning that their category distributions differ significantly between edible and poisonous mushrooms. This aligns with what I saw in the decision tree and random forest models.

One interesting detail is that **does-bruise-or-bleed** scored low compared to the others. It still matters in some association rules, but its standalone frequencies do not line up cleanly with toxicity. This is a reminder that chi-square only tests *individual* features some traits may only show their importance through their relationships with other attributes which is something models can capture but chi-square cannot.

Interpretation

Overall, the chi-square analysis tells me that the dataset contains several very strong categorical predictors of toxicity. The highest-ranking features are exactly the ones that appeared in the supervised modeling and the association rule mining.

Chi-square also reassures me that the predictive models weren’t simply picking up noise or overfitting but were learning patterns that are statistically meaningful in the raw data itself.

Cramér’s V Correlation Analysis

After running the chi-square tests, I wanted a broader view of how the categorical features relate not only to the target but also to each other. Chi-square tells me which individual predictors have the strongest association with toxicity, but it doesn’t show how the categorical variables interact. Cramér’s V fills that gap by measuring the **pairwise association between every categorical feature** and scaling the result between 0 and 1. This gives me a full correlation matrix that highlights which attributes tend to move together, whether any features are redundant, and how the class label fits into the overall structure of the data.

Why I Generated the Cramér’s V Matrix

The mushroom dataset is almost entirely categorical, and many features have overlapping meaning. For example, ring-type and has-ring are obviously related, while gill-color and stem-color might share patterns depending on species anatomy. I wanted to see whether any features were duplicating the same information, and whether the class label had broad associations or was driven by only a handful of variables.

Heatmap Interpretation

The Cramér’s V heatmap showed that **most categorical features have fairly low association with one another**, meaning the dataset does not contain a lot of redundancy. Each attribute contributes a distinct signal, which matches with earlier results showing that many weak predictors combine to produce a very strong classifier.

There are a few notable patterns:

- **ring-type and has-ring** displayed a very high association
- The **class label** showed moderate association with several key categorical variables including cap-shape, ring-type, stem-color, and gill-color mirroring the strongest Chi-square features and the most important variables from the decision tree and random forest
- **does-bruise-or-bleed** again showed very low association with most features and with the target

Interpretation

The heatmap reinforces something I’ve been seeing in every part of this project:
the mushroom dataset has a clean, stable internal structure where a small set of categorical traits consistently carry the strongest signal.

Cramér’s V confirms:

- There is very little multicollinearity among the categorical features
- The most predictive traits (ring-type, stem-color, gill-color, cap-shape, habitat) stand out clearly
- The patterns identified by supervised models, association rules, and Chi-square tests are not artifacts of modeling

K-Modes Clustering

Once I completed the supervised modeling and the statistical association analyses, I wanted to explore whether the mushroom dataset showed any natural groupings on its own. Since almost all of the attributes are categorical, K-Means would not make sense here. Instead, I used **K-Modes**, which is specifically designed for categorical data. K-Modes measures similarity through simple matching (Hamming distance) and updates each cluster center using the **mode** of each feature rather than a numeric average. This makes it a perfect fit for the structure of this dataset.

To keep computation manageable, I sampled **50,000 rows** and removed the three continuous variables, since they do not fit naturally into this framework. I ran K-Modes across several values of *k* and used the cost curve to select **four clusters** as the most meaningful structure. After fitting the model, I examined the cluster centroids to see what trait combinations defined each group, and then compared their class distributions to understand how closely the unsupervised clusters matched up with toxicity.

K-Modes Cluster Interpretation

Even though K-Modes never saw the class label, the clusters aligned with toxicity in a clean and stable way:

- **Cluster 0**: ~76% edible
- **Cluster 1**: ~60% edible
- **Cluster 2**: ~96% poisonous
- **Cluster 3**: ~88% poisonous

Two clusters leaned strong edible and two were strong poisonous. This told me that the categorical attributes naturally fall into groups that reflect real distinctions between edible and poisonous mushrooms even without supervised learning techniques. This result supports the patterns I saw earlier in the random forest, Chi-square tests, association rules, and Cramér's V matrix.

The categorical traits that kept showing up across the project: **cap shape**, **gill color**, **ring type**, **stem color**, **and habitat** also were the stars in the K-Modes clusters. In other words, the unsupervised structure of the dataset mirrors the supervised structure very closely. This reinforces the idea that the dataset’s signals are strong.

Interpreting the Cluster Centroids

Because each centroid represents the most common combination of categories in its cluster, the centroids let me identify the “prototype mushroom” in each group.

Cluster 0 — Mostly Edible

- Cap shape: convex
- Stem color: white
- Habitat: woods
- Season: autumn

This cluster forms a safe, consistent edible grouping with no bruising and no strong toxic indicators.

Cluster 1 — Moderately Edible

- Cap shape: flat
- Cap color: white
- Gill color: white
- Stem color: white
- Ring-type: none

This group looks like a more variable version of Cluster 0, with traits that are common among edible samples but with slightly more overlap with poisonous mushrooms, which explains the weaker 60/40 split.

Cluster 2 — High-Risk Poisonous Group

- Cap shape: convex
- Stem color: yellow
- No bruising
- Habitat: woods

The standout feature here is stem-color = yellow, which showed up repeatedly as a toxic signal in the random forest and Chi-square scores. This cluster is extremely poisonous (95%+).

Cluster 3 — Second Poisonous Group

- Cap shape: convex
- Stem color: brown
- Ring-type: scaly
- Season: summer

Ring-type = scaly is a rare but highly predictive toxic marker, and this trait dominates the cluster’s strong poisonous classification.

Across clusters, the centroids revealed stable, interpretable combinations of features that match both biological expectations and the patterns detected in the supervised models.

Final Summary of K-Modes Insights

K-Modes provided a strong unsupervised confirmation of finding I observed throughout the project. Without ever using the class label, the algorithm discovered four meaningful groups that closely mirror the edible/poisonous separation. This tells me:

- The dataset has **extremely strong and clean categorical structure**
- The same few features repeatedly drive the separation across logistic regression, Naive Bayes, decision trees, random forests, Chi-square tests, association rules, and Cramér’s V
- The traits form coherent patterns that are visible no matter what analysis I apply

In short, K-Modes reinforced the idea that the Mushroom Overload dataset is not noisy or ambiguous and contains strong signals that consistently identify toxicity.

Applications and Final Summary

Across all of the analyses I ran the same underlying structure in the mushroom dataset kept appearing. The categorical features that carried the strongest signal, cap shape, gill color, stem color, ring type, and habitat showed up repeatedly, no matter which model or method I used. Supervised learning, unsupervised clustering, and statistical association measures all converged on the same story: toxicity in this dataset is driven by a small, stable set of mushroom traits that form clear and consistent patterns.

From a modeling perspective, the supervised algorithms demonstrated how strong these signals are. Logistic regression and Naive Bayes provided useful baselines but were limited by their linear and independence assumptions. The decision tree and random forest, on the other hand, captured the structure almost perfectly, with the forest reaching near-perfect accuracy. These models confirmed that the dataset is highly separable and that the predictive patterns are both strong and easy for tree-based methods to learn. The unsupervised methods, K-Modes clustering and association rule mining, reinforced this same structure by discovering the edible/poisonous separation without ever using the target label. Statistical tools like Chi-square and Cramér's V then provided an additional layer of validation by ranking and visualizing the feature relationships that the models were relying on.

Beyond the technical results, this project was valuable because it gave me hands-on experience applying a wide range of data mining methods and then interpreting their outputs in a meaningful way. I had to evaluate each technique on its own terms, understand what kinds of patterns it could detect, and compare the strengths and weaknesses across models. Working through these different approaches, not just running them, but understanding why they behave the way they do, helped me build intuition about how to match a method to a dataset and how to cross-check findings using multiple analytical methods. Having to combine supervised predictions, unsupervised clusters, statistical association scores, and model-based feature rankings gave me a new sense of how these techniques complement each other in real data analysis.

Overall, this project taught me how to manage a large categorical dataset, select appropriate methods, build and tune models, interpret their results, and compare insights across a full pipeline of data mining techniques. Even though the Mushroom Overload dataset turned out to be highly structured and easy to classify, it provided a clean and practical way to develop the skills I will use on more complex and less cooperative datasets in the future.