Classifying Mushroom Edibility Based on Machine Learning Models

Jamie Phillips
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, United Kingdom
j.phillips@se20.qmul.ac.uk
Contribution: 33.3%

Phu Pham
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, United Kingdom
p.pham@se20.qmul.ac.uk
Contribution: 33.3%

Li Wang
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, United Kingdom
l.wang@se20.qmul.ac.uk
Contribution: 33.3%

I. Introduction and Background

Mushroom hunting has become more and more popular in recent years. This has accentuated the need to find efficient ways of identifying which mushrooms are poisonous and which are safe to eat (Chen et al., 2006). Many mushrooms that are deadly look like perfectly edible mushrooms. Every year, huge numbers of people die from eating poisonous mushrooms (White et al., 2019). Considering a mushrooms' particular features can be a highly effective way of identifying its toxicity/edibility and recognising mushrooms' toxicity/edibility has crucial social value in preventing food poisoning. This is a real-world problem, and our report could be consulted when trying to identify whether or not mushrooms are toxic in real-life.

Existing methods of identifying poisonous mushrooms are usually perceived to belong to one of four categories: chemical determination, fungal classification, folk experience and animal experimentation. However, identifying poisonous mushrooms using these methods has proved to be flawed and unreliable.

The use of toxic chemical determination procedures to identify poisonous mushrooms is gaining popularity. However, given the large number of unstable toxins and the awkwardness of handling the mushrooms, chemical determination methods remain unable to satisfactorily distinguish between edible and poisonous mushrooms.

Humans have been recognising poisonous mushrooms through observing shape, colour, odour and secretion features empirically for a long time. However, the high frequency of annual poisoning events signals the low accuracy of this approach. The huge number of fungal species in the world also suggests that this approach is limited. Recent estimates using high-throughput sequencing techniques indicate that 5.1 million species of fungi exist (Blackwell, 2011). Humans cannot readily accumulate the extensive background knowledge needed to identify edible mushrooms within a wide range of species.

Yet there is great potential for machine learning algorithms to reliably perform this task. Machine learning techniques are not dependent on slowly acquired background knowledge and have the added advantage of being able to identify unknown species. Machine learning approaches have increasingly been used in recent years to tell the toxicity of mushrooms. For example, Chaoqun (2019) has applied machine learning models to recognise mushrooms' toxicity within an application.

II. Project Aims

This project aims to use data provided by the UCI machine learning repository on the defining features and toxicity of mushrooms to create machine learning models capable of predicting whether or not a mushroom is poisonous based on its other features. In achieving this aim, we will identify and visualise trends in the data. We will evaluate our models using accuracy score and cross-validation to demonstrate accuracy. We will also analyse the strengths and weaknesses of the models we have created. We will identify possible improvements to our methods which could be useful for future data analysis.

III. Literature Review

In this section, we will present a review of relevant literature. This literature will be discussed in connection with the research problem we are tackling.

The literature on mushroom classification has flourished in recent years. Eusebi et al. (2008) set out to apply a range of classifiers to Schlimmer's compilation of the *Audubon Society's Mushroom* data. These methods include using a voted perceptron algorithm, decision trees, the nearest neighbour classifier and a covering algorithm which creates only correct rules. The PRISM classifier received high accuracy but only one attribute - 'stalk-root' (the attribute with missing values) - was removed. An unpruned tree is also utilised to generate a human-machine interactive application. The application was a human-machine interactive, web-enabled client-side classification tool. The Mushroom Database application required human interaction within the classification process. One way to make our classification methods useful to the end user would be to incorporate them into a classification tool in this style.

Wibowo et al. (2018) drew comparisons between the performance of 3 data mining algorithms: Naïve Bayes, C4.5 based decision Tree and Support Vector Machines using the Audubon Society Field Guide dataset. C4.5 can be used as a statistical classifier and builds decision trees from a set of training data using the concept of entropy (the average level of information inherent in a variable's possible outcomes). SVM and C4.5 both demonstrated better accuracy than Naïve Bayes. C4.5 proved to be 0.02 seconds faster than SVM. C4.5 performed the best out of the algorithms overall and in terms of processing speed. The C4.5 decision tree produced a decision tree reduced to five attributes (by pruning): odor, spore-print-color, gill-size, gill-spacing, population. The high accuracy score found with SVM reinforced our intuition that SVM might be a useful method to apply to our dataset (since support vector machines are effective in high dimensional spaces and our dataset contains quite a lot of attributes).

Our research has showed that certain features are often present in poisonous mushrooms. For example, mushrooms with white gills or a red colour on the cap or stem are often poisonous. This validates the usefulness of our project's objective to find trends in the mushroom dataset by identifying patterns and relationships between variables.

IV. Data Retrieval

This dataset was found on Kaggle and the data originates from the Audubon Society Field Guide to North American Mushrooms in the UCI machine learning repository (See Appendix 2). The Audubon strives the conserve natural ecosystems. The data assembled by Schlimmer contains samples relating to 23 species of gilled mushroom. Each species is labelled as being either definitely edible, definitely toxic and of unknown edibility. The unknown class' values were placed inside the definitely toxic class due to the dangers of unknown toxicity. The Audubon Society Field Guide claims that there is no clear-cut rule for identifying a mushrooms' edibility, highlighting the need to find machine learning methods for classifying mushrooms.

The shape of the dataset is 8125R x 23C. The dataset we are using has 22 attributes in addition to the target attribute (class) which reveals whether a mushroom is poisonous or edible-p=poisonous, e=edible. Along with the target variable, the dataset includes the following features cap shape, cap surface, cap color, bruises, odor, gill appendages, gill spacing, gill size, gill color, stalk shape, stalk root, stem-surface-above-ring, stem-surface-below-ring, stem-color-above-ring, stem-color-below-ring, veil type, veil color, ring number, ring type, spore print color, population, and habitat. There are 8124 recording of mushroom data, which can be divided into two classes: edible (4208) and poisonous (3916).

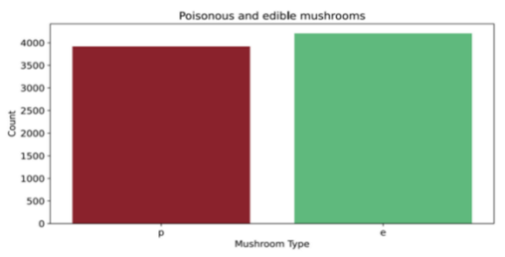


Fig. 1. Histogram of Poisonous and Edible Mushrooms

Before working on the dataset, we had to extract the data and put into a convenient format for processing. We thought it would be suitable to represent the data as a Pandas DataFrame since the original mushroom dataset was in csv format. (See Appendix 1 & 2)

V. Data Representation

We decided to use Python3 to carry out our analysis since it contains powerful libraries and packages which are geared towards data manipulation, visualisation and machine learning. In particular, we used the following libraries:

Matplotlib- this is a cross-platform, graphical plotting library. It can be used to make scatterplots, providing a variety of colour, themes and more to customise plots. We will use this library for the visualisation and exploration stages of our project.

Pandas- this library offers operations and data structures useful for manipulating numerical tables and time series. The Pandas library facilitates data manipulation and analysis. In this project, we mainly use the Series and DataFrame object functionalities of this library. DataFrames have two dimensions, are structured similarly to SQL tables and excel spreadsheets, and have columns with one or more data types.

Seaborn- this library is used for data visualisation offers informative statistical graphics and state of the art visualisations including heatmaps.

Scikit-learn- this library contains many efficient tools for machine learning and statistical modelling such as classification, regression, clustering and dimensionality reduction.

Numpy- this library is useful for working with multi-dimensional arrays, matrix data structures and high-level mathematical functions which are essential for operating on these arrays.

Chefboost- this lightweight framework provides support with regular decision tree algorithms including C4. 5, ID3, CHAID and CART. It also supports advanced methods including gradient boosting trees, random forest and adaboost. It contains categorical features support.

VI. Data Cleaning

Pre-processing is a crucial step. It handles entries that are invalid, complete or missing due to reasons such as data entry issues. Incomplete data is an inevitable problem when dealing with real-world data sources. Transformations to the data should also be carried out before the data is divided into test, train and validation data since filling in missing values depends on the statistical values of the features. Our general process will be to identify missing values, replace them and transform the data.

Deciding on our data cleaning process

First, we will display the values which are unique in each column. (See Appendix 3) This allows us to find different ways that null values have been represented in the data, so that we can standardise their representation by converting them to numpy NaN values.

We proceed to identify the missing values in each column using isnull(). Whilst we have found that there are missing values ('?') in the 'stalk-root' column, all the other columns contain appropriate values. There are 2480 '?' values in the stalk-root column. This means that 2480/8124 of the values in the 'stalk-root' column are missing (31%). The range of values in the 'stalk-root' column are bulbous=b, club=c,cup=u, equal=e, rhizomorphs=z, rooted=r and missing=?. We will replace the '?' values with numpy.Nan values using the pandas.DataFrame.replace method as this is the standard representation for null values and makes the missing values recognisable to KNNImputer().

Missing values must be treated with care since there are various reasons why the values might be absent and differing approaches might be required to deal with them. Missing values could also be dependent on other variables' values. We must ask: is the value missing since it was not recorded, or does it simply not exist? If a value is missing since it does not exist, nothing is gained in guessing what that value might be. Since in our case these values may well be missing because they were not recorded, it may be helpful to impute them based on the other values in their columns and rows.

One way to resolve the issue of missing values is to drop all rows with missing values or to drop the stalk-root column (the only column to contain missing values). The limitations or this are that we lose valuable data that could be used to train our model and improve its predictive potential. But our dataset is probably big enough to counteract this risk- it is only when a training dataset is small that there might not be enough data for our models to effectively learn from. Additionally, the validation data must not be too small otherwise there will be a large variance in the accuracy, precision, recall and F1 score.

Missing values can also be overcome by inserting the most common value in the attribute with the missing values in place of the unknown values, or by substituting the mean. We will not apply this approach since it is too imprecise, whereas KNNImputer provides a more circumstance-specific replacement for missing values.

Another issue we find is that the machine learning algorithms we will use only allow numerical data as input and output. Our dataset contains categorical values which are expressed by letters, so we will need to encode these letters to numbers.

We could leave the missing values as '?'- a kind of special value which we encode to a number just as we would any other letter. We could also set up a predictive model to predict the value. We will try using scikit-learn's KNNImputer to impute the missing data.

Upon reflection, we will implement three distinct approaches to clean the data and apply our machine learning models to each cleaned dataset.

These different approaches are:

1. Keep the dataset in its original form, treating the missing values in the 'stalk-root' column as a special value, '?'
2. Remove the '?' values and impute them using KNNImputer

Using KNN to impute missing values

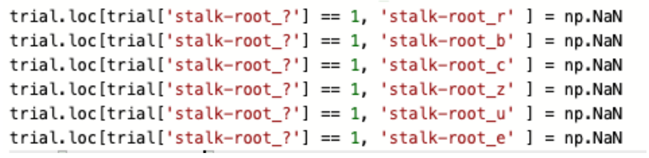
KNN is an algorithm used to match a point with its nearest k neighbours in a multi-dimensional space. A point value can be approximated by the values of the points nearest to it, based on other variables. To find a missing value we can look for its k nearest neighbours and get their 'stalk-root' value.

The KNN method can only be applied to numeric data. We have used the pd.get_dummies method to encode these letters to numbers. As the values are all non-numeric, it is necessary to keep the values still meaningful after encoding. The method is changing the classification data into binary data. 0 represents false, and 1 represents true. Every column has been split into several columns based on what values it contains in the original dataset. For example, the column 'cap-shape' in the original dataset contains 6 unique values ('x','b','s','f','k','c'). The pd.get_dummies method has separated this 1 column into 6 columns: 'cap-shape-x', 'cap-shape-b', 'cap-shape-s', 'cap-shape-f', 'cap-shape-k', and 'cap-shape-c'. If a mushroom instance had a 'c' value in the cap-shape column in the original dataset, in the encoded dataset it will have a 1 value in 'cap-shape-c' column, and a 0 value in all the other 'cap-shape' columns. Therefore, the dataset has been converted into 119 columns which overall represent the same meaning as the original dataset.

Appendix 4 is the example of transformed dataset.

For missing values in the column 'stalk-root', the encoding has generated a new column called 'stalk-root-?'. The basic idea is that we treated all the missing values in 'stalk-root' as being one of the other unique values in 'stalk-root' and set out to identify which values they were. Then the KNN method will learn from the neighbours of the rows containing the missing values to impute these values.

The way we have approached this is to delete all the other 'stalk-root' columns' values ('stalk-root-e','stalk-root-c', and etc.) if a row has a '1' value in 'stalk-root-?' column. After this we delete the column 'stalk-root-?'. We can then apply KNN imputation to generate a value for the missing values in the 'stalk-root' columns.

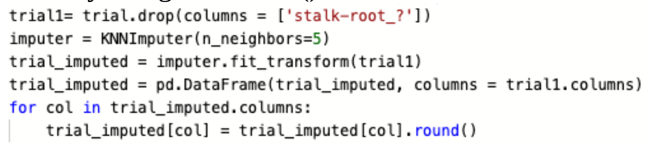


```
trial.loc[trial['stalk-root_?'] == 1, 'stalk-root_r' ] = np.NaN
trial.loc[trial['stalk-root_?'] == 1, 'stalk-root_b' ] = np.NaN
trial.loc[trial['stalk-root_?'] == 1, 'stalk-root_c' ] = np.NaN
trial.loc[trial['stalk-root_?'] == 1, 'stalk-root_z' ] = np.NaN
trial.loc[trial['stalk-root_?'] == 1, 'stalk-root_u' ] = np.NaN
trial.loc[trial['stalk-root_?'] == 1, 'stalk-root_e' ] = np.NaN
```

Fig. 2. Example Code of Data Cleaning

Applying KNN requires us to take several parameters into account. Firstly, taking a low k value will increase the influence of noise and make the results less generalisable. Taking a high k will dilute local effects which are crucial to imputing the values accurately. The default k value of 'n_neighbors' for KNN is 5.

Here is a sample of the imputed data. We notice that some null values have been replaced by floats. (See Appendix 5) Since we are dealing with categorical data, only whole numbers can be used to reference a particular category. We must therefore round values in the stalk-root column to the nearest integer. This is done by using the round () function.



```
trial1= trial.drop(columns = ['stalk-root_?'])
imputer = KNNImputer(n_neighbors=5)
trial_imputed = imputer.fit_transform(trial1)
trial_imputed = pd.DataFrame(trial_imputed, columns = trial1.columns)
for col in trial_imputed.columns:
    trial_imputed[col] = trial_imputed[col].round()
```

Fig. 3. Example Code of Data Cleaning

VII. Data Exploration and Visualisation

Before modelling, we spent a lot of time exploring the data. This stage is essential in allowing us to pick up on underlying trends and correlations. A great amount of insight is found by analysing our models' performance in relation to general patterns and shapes in the data. Exploring the data has allowed us to gather fundamental information about the dataset that can be used in our analysis.

Appendix 6 are histograms showing frequencies for each value in the dataset. We created these stacked histograms using 'histogram ()' from the Seaborn package. This allowed us to compare, for the unique values of each column, the number of poisonous and edible mushrooms with those unique values. Some columns appear to have a weak relationship with the column 'class', for example, 'gill-attachment', 'stalk-shape' 'veil-type', 'veil-color' and 'ring-number'. There are similar numbers of edible and poisonous instances with the unique values in these columns, suggesting that these columns are less important for predicting mushroom toxicity. However, some columns appear to have strong links to whether the mushroom is edible or not. For the columns 'odor', 'gill-color', 'ring-type' and 'spore-print-color', there is a big similarity between the counts of each unique value in the class-e and class-p bars. Meanwhile, other columns display medium relationship with column 'class' as there are some differences between two bars in 'class'-p and 'class'-e.

We also made the histogram for the column 'stalk-root'. The left diagram treats '?' as a special value. The right one contains data which has been imputed by KNN (the second approach). (See Appendix 7)

In order to observe the correlation between the values of the column 'class' and each other column, we also created histograms for each column against the column 'class'.

As we have found that 'odor', 'gill-color', 'ring-type' and 'spore-print-color' have strong links to mushrooms' toxicity, it is important to evaluate the values of these columns. (See Appendix 8)

For the class 'odor', it can be seen that any mushroom with the value 'p', 'f', 'c', 'y', 's', and 'm' is poisonous. Mushrooms with the values 'a' and 'i' are all edible. Most mushrooms with the value 'n' (over 96%) are edible. 'n' is the most numerous of all the values- there are 3528 instances with the value 'n', followed by 'f' with a count of 2160.

For the class 'gill-color', the values 'b' and 'r' are only found in poisonous mushrooms, with a count of 1728 and 24 respectively. More than half the mushrooms with the values 'g' and 'h' are poisonous. Whereas mushrooms with the rest of the values ('k', 'n', 'p', 'w', 'u', 'e', 'y', 'o') are mostly edible.

For the class 'ring-type', the values 'p' and 'e' take the most percentage, where over 75% of mushrooms with value 'p' are edible and over half of mushrooms with value 'e' are poisonous. Mushrooms with value 'l' and 'n' are necessarily poisonous, and the value 'f' is necessarily 'edible'.

For the attribute 'spore-print-color', 'k', 'n', 'h' and 'w' are much more numerous than the other values. Over 90% of the mushrooms which have 'k' or 'n' as their value are edible, whereas nearly 98% of mushrooms with 'h' and 75% with 'w' are poisonous.

As with Appendix 7, two diagrams with histograms relating to the column 'stalk-root' have been appended. The left diagram treats '?' as a special value. The right one contains data which has been imputed using KNN. (See Appendix 9)

Normally, to find out which single feature best predicts the target variable, we might work out the covariance of each feature, target-variable pair and choose the one with the highest absolute value. But since the entire dataset consists of categorical values, covariance is unsuitable. This suggests that evaluating uncertainty coefficients (entropy) might be a better option.

Attempting to calculate Pearson's R to find the correlation between variables will not work since it is undefined when the data is categorical. With our dataset it seems unfeasible to subtract an attribute's value from the average of the values for that attribute, e.g. you cannot subtract 'foul' from 'fishy' for the attribute 'odor'. Calculating Pearson's R for our one-hot encoded data is also unproductive since the correlation table contains 112 attributes- it is fine-grained and convoluted.

To overcome this challenge, we have created a horizontal bar graph using Cramér's V, which is well-suited to categorical data. It draws on Pearson's chi-squared test (which evaluates how likely any observed difference between variables occurred by chance). This is a great alternative to correlation since, similarly to correlation, it returns an easily understood output between 0 (no association) and 1 (full association).

However, the output of Cramer's V means that we are losing valuable information about the relationship between class and the other variables due to Cramer's V's symmetry.

This problem is solved by obtaining an asymmetrical appraisal of the association between our data's variables. Here we have created a heatmap showing the level of association between 'class' and the other attributes. This involved applying the Theil's U formula (the uncertainty coefficient) to the dataset to get an asymmetric measure ($U(x,y) \neq U(y,x)$) of the association between class and the other attributes. Theil's U draws on the conditional entropy between x and y, the number of possible states that y has and the frequency of their occurrence. This helps us to identify if there is a feature which helps us to better understand class' values. Applying Theil's U gives us an output value between 0 and 1, where 1 means very high association between x and y. This heatmap gives us a clearer idea than the bar graph of how useful each attribute would be in predicting 'class' values since the information gain from knowing 'class' values for understanding other attributes' values is not considered in the association calculation.

The heat map displays some very high associations i.e., between 'class' and 'odor'/'spore-print-color'/'gill-color'/'ring-type'. We could remove the columns which are less associated with 'class' from our predictive model- the columns which are less useful for predicting 'class' could be removed, i.e., 'veil-type', 'gill-attachment', 'veil-color', 'stalk-shape'.

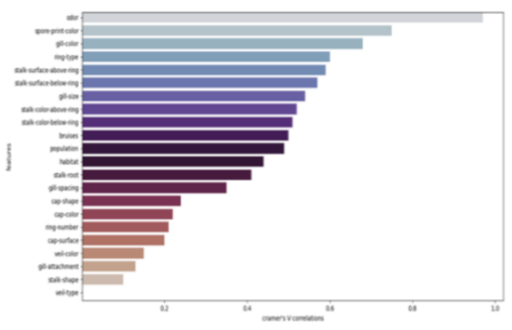


Fig. 4. Cramer' V correlation (Click the hyperlink to View)

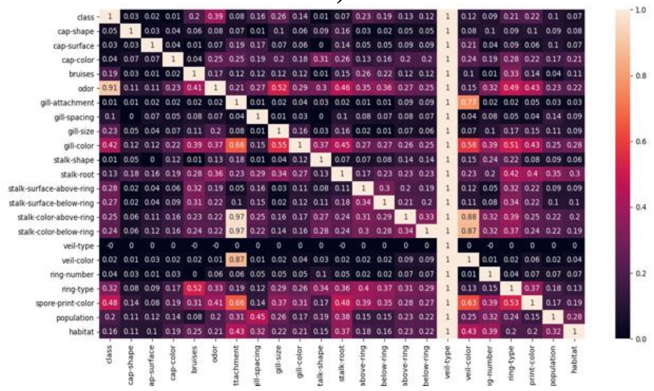


Fig. 5. Theil's U heat map (Click the hyperlink to View)

VIII. MODELLING

Models:

Choosing which learning algorithm to use is a critical stage. How we evaluate a classifier is often based on prediction accuracy (the percentage of correct predictions divided by the total number of predictions). We will split the dataset by using 70% for training and 30% for estimating performance.

Cross-validation divides the dataset into subsets of equal size. For each subset, the classifier is trained on the ensemble of all the other subsets.

Random Forest

We are using the Random Forest Classifier since accurate prediction of mushroom toxicity is a key aim and Random Forest's ensemble learning builds multiple decision trees, merging them together to form a more precise and robust prediction. Using multiple decision trees also counteracts overfitting as the mean of each output serves as the answer. The testing performance of Random Forests does not decrease due to overfitting but after a certain number the performance tends to stay at a certain value. We will therefore choose a reasonably high number of trees. Inputting our training set's attributes' values into the model, the algorithm will generate a series of rules allowing us to predict the edibility of our testing dataset's mushrooms based the other features' values.

The decision tree rules will be used to calculate the mushrooms' edibility in the testing dataset using the other features' values in the test dataset.

PCA

After the dataset was One-hot encoded, we proceeded to fit the data to the Logistic Regression

Model and Decision Tree Classification model. Both models got 100% in accuracy and 0 false negative, 0 false positive. These results apply to all three of our data cleaning approaches.

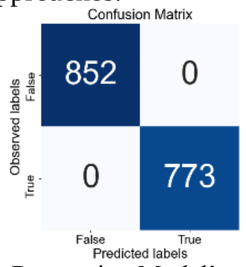


Fig. 6. Logistic Regression Model's confusion matrix

The perfect accuracy for both models suggested that they were being overfitted with the dataset. This means that they would fail to generalise and would struggle if given new data. Checking the usual suspects of overfitting, we found out that the target has not been added in as one of the features nor was the test data part of the training data, thus eliminating the risk of Data Leakage. Next, we re-evaluated our validation method of choice, Train/test split, which we determined to be appropriate for the dataset and would not cause overfitting. This leaves us with overfitting due to an increase in dimensions. The dataset's number of dimensions has increased from 23 to 119 due to being One-hot encoded, the number of features has increased up to 5 times, but the number of training samples remains the same, growing the feature space. In order to correct this "Curse of Dimensionality", we opted to use Principal Component Analysis (PCA) to reduce dimensionality.

Keeping '?' in column 'stalk-root' as a special value and applying PCA method, the Logistic Regression Model gave us the new accuracy score of 0.8806153846153846 and the following confusion matrix:

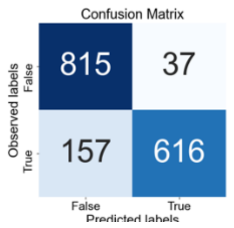


Fig. 7. Logistic Regression Model' s confusion matrix after applying PCA.

Although the accuracy is now lowered, the model is now much less likely to be overfitted. The n components chosen for PCA in this example is 3, this is an arbitrary choice, but it still showed that PCA can be use in this situation to reduce overfitting.

Decision Tree Classifier

We will use sklearn's Decision Tree Classifier inbuilt model to create a training model that can predict the value of the target variable by learning simple decision rules inferred from the training data. We are using this model since it is effective and easy to implement. We decided to stick with the default criterion for the decision tree, gini index, since we are already applying an entropy-based model (C4.5). Gini index measures how often a randomly chosen element would be wrongly identified.

SVC

We will also be using sklearn's SVC. Support vector machines are highly effective with multi-dimensional data, and our dataset involves many attributes. Support vector machines are a set of supervised learning methods which can be utilised for classification. The SVC classifier takes in two arrays. The objective of SVC is to fit the data you input and return a 'best fit' hyperplane which categorises that data.

Because SVM optimisation happens by minimizing the decision vector *w*, the optimal hyperplane is affected by the scale of the input features. We must therefore standardise the data to have a mean of 0 and variance of 1 before SVM model training. We used StandardScaler to transform our data in this way.

The standard score of a sample x is calculated as: z = (x - u) / s, where u is the mean of the training samples and s is the standard deviation for the training samples. Z-scores are written in terms of standard deviations from their means. These z-scores therefore have a distribution with a mean of 0 and a standard deviation of 1.

We combined the steps of applying standard scaler to the data and implementing SVC on the data into a single step using sklearn.pipeline.Pipeline as this streamlines the processes.

C4.5

We decided to create a C4.5 model since our literature review showed us C4.5 models' potential to achieve high accuracy and speed (Wibowo et al., 2018). Additionally, our research indicated that the C4.5 algorithm is well-suited to nominal values as found in our dataset and provides a single pass pruning service to prevent overfitting (which is a risk that needs to be mitigated due to the particularity of the dataset).

To get the data into the correct format for use in the model, the target variable 'class' must be renamed as 'Decision' and placed to the right of the other columns to be recognised by the algorithm. We then separated the data into 80% train and 20% test data using indexing and inputted the train data into the model.

We applied the C4.5 model to build a decision tree stored as python if statements. C4.5 uses the concept of information entropy. Decision tree rules are found using the ratio between entropy (the average level of uncertainty inherent in a variable's possible outcomes) and information gain for each feature.

The feature with the maximum gain ratio is the decision rule. (See Appendix 10)

For the C4.5 model, we inputted a set S of instances (the train data) described by nominal attributes, with each instance belonging to one class. The highest informational gain is calculated using the following formulae:

Entropy: $E(S) = \sum_{i=1}^{n} - \Pr(C_i) * log_2 \Pr(C_i)$

Gain: $G(S, A) = E(S) - \sum_{i=1}^{m} \Pr(A_i)\, E(S_{Ai})$

E(S) – the information entropy of S
G(S,A) – the gain of S after a split on attribute A
n – the nr of classes in S
Pr(Ci) – frequency of the class Ci in S
m – nr of values of the attribute A in S
Pr(Ai) – the frequency of cases that have the Ai value in S
E(SAi) – the subset of S with items that have Ai value

TESTING AND RESULTS

C4.5

We used the inbuilt C4.5 model from the python library chefboost importing it as follows:

'from chefboost import Chefboost as chef'

- Accuracy Score: 0.9901538461538462

Fed by the train data, the algorithm divided the attributes into two groups – the most dominant attributes and others- to build a tree. It then calculated the ratio between information gain and entropy for each attribute to find the most dominant attribute- the most dominant attribute was put on the tree as a decision node. Entropy and gain score ratios are calculated amongst the other attributes to find the next most dominant attribute. This process continues until a decision is reached for that branch. The feature with the maximum gain ratio will be the decision rule.
Since the algorithm is suited to nominal data, it is not suitable to carry out PCA or KNN imputation on the training set before feeding the model.
To see the resulting rules for our C4.5 decision tree, refer to Appendix 11. A pruned tree narrowed down to 'odor', 'spore-print-color', 'gill-size', 'stalk-shape', 'stalk-surface-below-ring' and 'ring-type' was created. The rules from our tree are corroborated by our previous finding with Theil's U that these variables are strongly associated with class. The high metric-values for the variables odor, spore-print-color and gill-size in the C4.5 tree (0.986, 0.754 and 0.9968 respectively) validate that a lot of information is discovered about 'class' through knowing the values for odor, spore-print-color and gill-size.

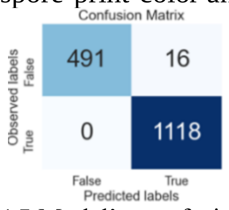


Fig 8. C4.5 Model' s confusion matrix

Random Forest Results

We chose to use the inbuilt random forest model from the python library sklearn.

"from sklearn.ensemble import RandomForestClassifier"

The following decision tree was created by the train data. 'Gill-size-b' is at the root of the tree suggesting that it is an important attribute. The first rule is to check whether 'gill-size-b' = 1 or 0 which allows it to make an initial separation. As with the C4.5 algorithm, the variables 'odor', 'spore-print-color' and 'gill-size' appear in the pruned tree, suggesting their importance in identifying mushroom toxicity and adding support to our exploratory analysis findings.

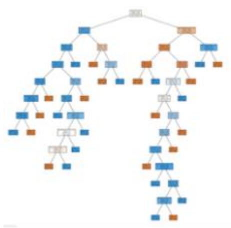


Fig. 9. Decision Tree Result (Click the hyperlink to View)

For all data cleaning approaches we must find an optimal number of n components for PCA. Using a for loop would find out the number n components that would give us the highest accuracy. (See Appendix 12)
Running the loop for up to 20 n components, shows us:

- Optimal n components for PCA: 6
- Accuracy score: 1.0
- Cross Validation Accuracy: 0.9357336870026526

The accuracy results are shown in the plot below:

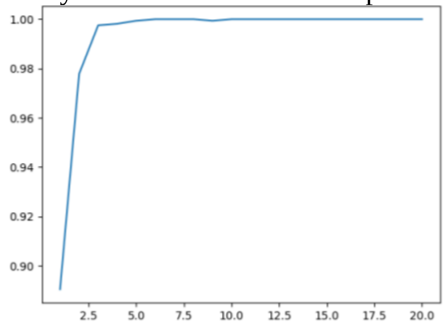


Fig. 10. Random Forest model's Accuracy Result

The second approach of using KNN to estimate the missing value would need us to find the optimal number of k neighbours. We ran a testing loop for up to 21 k neighbours, the results of the optimal test are:

- Optimal number of k neighbours: 3
- Accuracy Score: 0.9993846153846154
- Cross Validation Accuracy: 0.9338863963622585

The accuracy results of the tests are shown in the plot below:

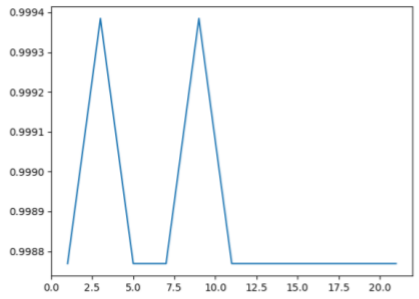


Fig. 11. Random Forest model's Accuracy Result

Considering the optimal accuracy of the two data cleaning approaches, we choose to go with the first data cleaning approach and 6 components for PCA. This would give us the accuracy of 100% and the following confusion matrix:

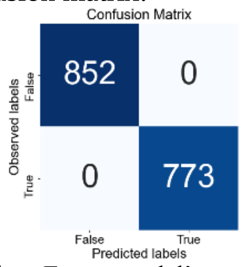


Fig. 12 Random Forest model's confusion matrix

Logistic Regression Model

We choose to use theinbuilt Logistic Regression Model from the python library sklearn since it is a fast, simple algorithm which delivers easily interpretable results. It is particularly suited to binary classification and we aim to receive a binary result for 'class' where 1 = poisonous and 0 = edible.

"from sklearn.linear_model import LogisticRegression"

For all data cleaning approaches we must find an optimal number of n components for PCA. Using a for loop would find out the number n components that would give us the highest accuracy.

Running the loop for up to 20 n components, shows us:

- Optimal n components for PCA: 19
- Accuracy score: 0.984
- Cross Validation Accuracy: 0.9193427813565744

The accuracy results are shown in the plot below:

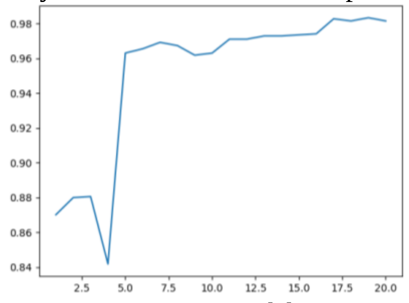


Fig. 13. Logistic Regression Model**'s** Accuracy Result

The second approach of using KNN to estimate the missing value would need us to find the optimal number of k neighbours. We ran a testing loop for up to 21 k neighbours, the results of the optimal test are:

- Optimal number of k neighbours: 5
- Accuracy Score: 0.932923076923077
- Cross Validation Accuracy: 0.9251320197044335

The accuracy results of the tests are shown in the plot below:

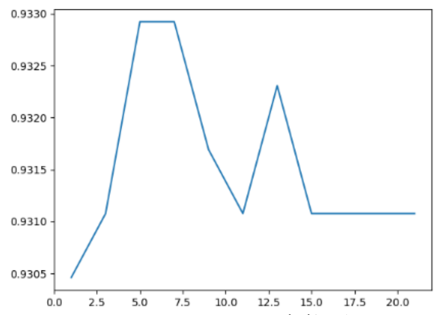


Fig. 14. Logistic Regression Model's Accuracy Result

Considering the optimal accuracy of the two data cleaning approaches, we chose to go with the first data cleaning approach and 19 components for PCA. This would give us the accuracy of 98.4% and the following confusion matrix:

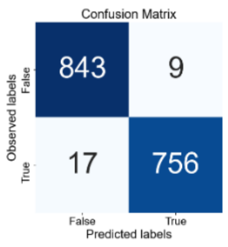


Fig. 15 Logistic Regression Model**'s** confusion matrix

SVC

We choose to use the the inbuilt SVC from the python library sklearn

"from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC"

For all data cleaning approaches we must find an optimal number of n components for PCA. Using a for loop would find out the number n components that would give us the highest accuracy.

Running the loop for up to 20 n components, shows us:

- Optimal n components for PCA: 16
- Accuracy score: 1.0
- Cross Validation Accuracy: 0.8735420992800303

The accuracy results are shown in the plot below:

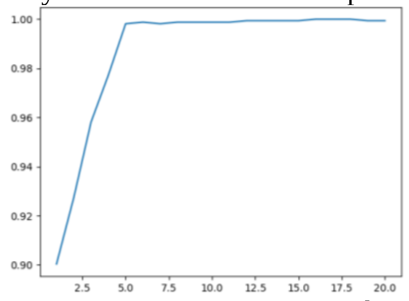


Fig. 16. SVC's Accuracy Result

The second approach of using KNN to estimate the missing value would need us to find the optimal number of k neighbours. We ran a testing loop for up to 21 k neighbours, the results of the optimal test are:

- Optimal number of k neighbours: 1
- Accuracy Score: 0.9889230769230769
- Cross Validation Accuracy: 0.8734189465706708

The accuracy results of the tests are shown in the plot below:

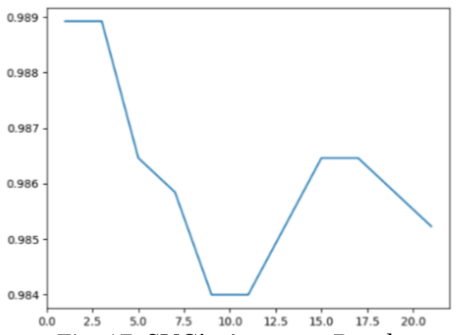


Fig. 17. SVC's Accuracy Result

Considering the optimal accuracy of the two data cleaning approaches, we choose to go with the first data cleaning approach and 16 components for PCA. This would give us the accuracy of 100% and the following confusion matrix:

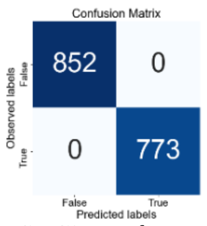


Fig. 18.SVC's confusion matrix

Decision Tree Model

We choose to use the inbuilt Decision Tree Classifier from the python library sklearn,
"from sklearn.tree import DecisionTreeClassifier"
For all data cleaning approaches we must find an optimal number of n components for PCA. We used a for loop to find out the number n components that would give us the highest accuracy.
Running the loop for up to 20 n components, shows us:

- Optimal n components for PCA: 12
- Accuracy score: 1.0
- Cross Validation Accuracy: 0.9963058734369079

The accuracy results are shown in the plot below:

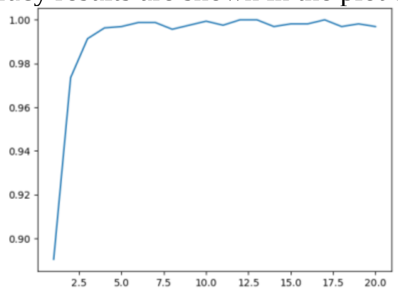


Fig. 19. Decision Tree Model's Accuracy Result

The second approach of using KNN to estimate the missing value would need us to find the optimal number of k neighbours. We ran a testing loop for up to 21 k neighbours, the results of the optimal test are:

- Optimal number of k neighbours: 3
- Accuracy Score: 0.9987692307692307
- Cross Validation Accuracy: 0.996182796513831

The accuracy results of the tests are shown in the plot below:

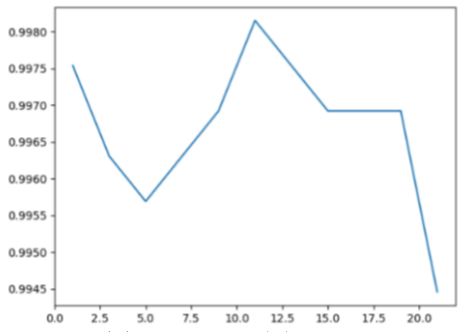


Fig. 20. Decision Tree Model's Accuracy Result

Considering the optimal accuracy of the two data cleaning approaches, we choose to go with the first data cleaning approach and 12 components for PCA. This would give us the accuracy of 100% and the following confusion matrix:

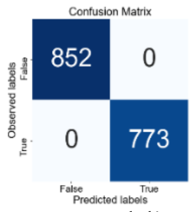


Fig. 21 Decision Tree Model's confusion matrix

Printing the results of every model using PCA without KNN imputation gives us the following graph:

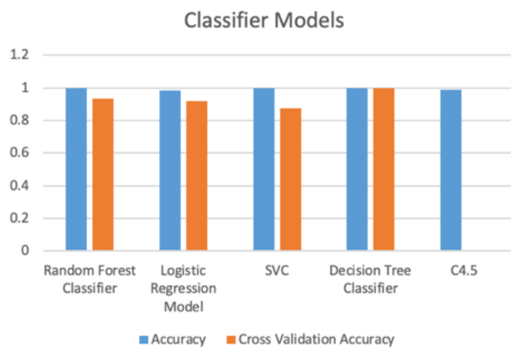


Fig. 22. Classifier Models

Evaluating model efficiency

In our evaluation of the models, we have found the cross-validation score along with the accuracy score to help us understand how the statistical analysis would generalise to an independent dataset. For each model we have used k-fold cross-validation to divide the entire dataset into 5 folds. For each fold, our model predicts the y values by applying the model to the x values. For each model, we have found the mean of these scores to check that the accuracy is similar in all the folds. This helps prevent overfitting as a high mean score reassures us that our algorithm is consistent and would lead to similar performance when deployed in production. When splitting the data set into a training and test set, the accuracy and metrics might be biased on how the split was carried out and not be representative of the model's ability to generalise.

The SVC model performed worse than the other models- despite receiving a very high accuracy score, its cross-validation score was significantly lower than the other models. This suggests that the model has been overfitted. In terms of metrics, the Decision Tree Classifier performed the best out of all the models, with an accuracy score of 1.0 and cross validation score of 0.996 (3 d.p.). However, we are still inclined to favour the Random Forest Classifier model since it received almost as high accuracy and cross-validation scores and uses ensemble methods which help prevent overfitting. For each model, using KNN imputation instead of keeping the missing values as a special value '?' generally decreased the accuracy and cross validation scores slightly (though the models still achieved high scores). This suggests that the incomplete records in the dataset mostly had the same unrecorded value type for 'stalk-root'.

IX. CONCLUDING REMARKS

Overview

We believe that we succeeded in achieving the aims of our analysis. Our data cleaning, visualisation, exploration and preparation have allowed us to observe distinct and interesting patterns in the data. We were also able to find important indicators of edibility and toxicity from our analysis. We gained a better understanding of how our features influenced one another by measuring and studying the association between them. Applying our models to different variations of the data allowed us to compare accuracy scores before and after cleaning, transforming and preparing the data with KNN imputation, special values and PCA.

We have faced big hurdles due to the nature of our categorical variables. For instance, our data contained complex trends and nuances that our models struggled to pick up on- this undoubtedly lowered our models' performance. Despite this, applying PCA allowed us to overcome the problem of dimensionality created by one-hot-encoding our data (which exponentially increased the number of variables). We successfully identified overfitting in our models by incorporating cross-validation into our model evaluation. We tweaked our models to optimise performance (finding the ideal number of neighbours and PCA value).

Our model implementation validated our findings in our exploratory data analysis that 'odor', 'gill-size' and 'spore-print-color', in particular, provide a lot of information about mushrooms' toxicity levels. This is because these variables featured prominently and received high entropy scores in our pruned decision tree models.

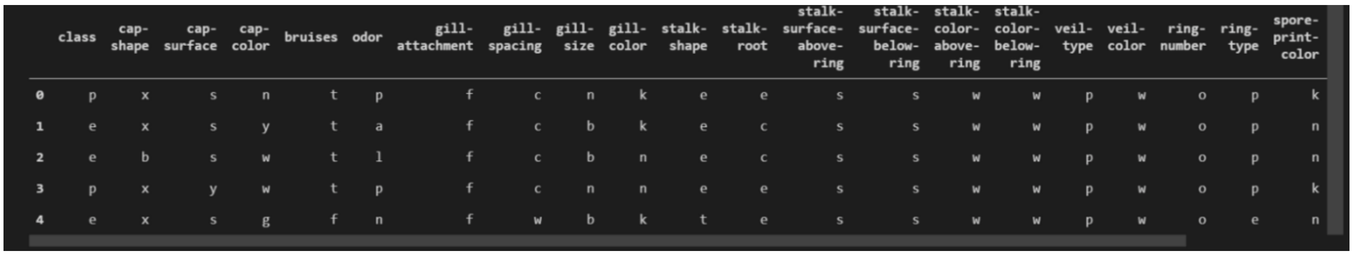
We implemented a range of models including advanced machine learning methods and evaluated their performance and efficiency. Some of our models achieved high accuracy scores (over 90%), other models were less successful, but this helped us understand the relationship between our attributes and their dependencies.

Real-life applications and future mapping

On reflection, we think that a bigger data sample of mushrooms would be needed to accurately apply our models to mushroom data in practice since our research tells us that there are many millions of mushroom species in the world and our dataset contains a disproportionately small number of species. Although our models performed highly on our training and test data, they are undertrained on data pertaining to species unknown to the dataset. E.g., due to the complex trends identified within in this Californian dataset, we could not expect the model to consistently identify the toxicity of mushrooms from another region. We were limited by the size and breadth of the dataset- for certain attribute values, there were only a few instances containing those values (not enough to properly identify trends). To further elaborate the work carried out in this project, we could develop our approach to data retrieval by finding a bigger dataset or combining multiple mushroom datasets. Incorporating expert knowledge into our models might be a good solution to overcome the data deficit about certain attributes' values.

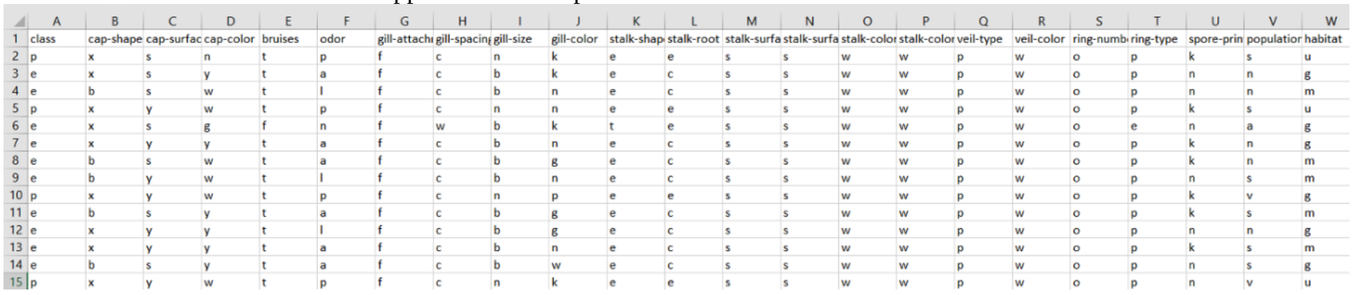
To make our models more appropriate for wider use, we could make them available in production where web applications and APIs can consume the trained model, providing new data points and generating updated predictions. This would improve accuracy as our models would be exposed to a wider range of mushroom data. We could also deploy a web application designed for entertainment purposes using streamlit which allows the public to input details about mushrooms to return an edibility meter score.

APPENDIX



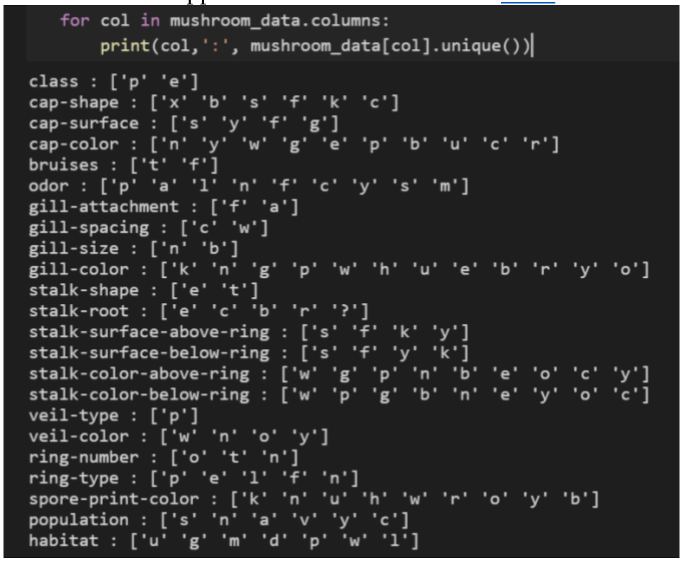
	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	stalk-root	stalk-surface-above-ring	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color
0	p	x	s	n	t	p	f	c	n	k	e	e	s	s	w	w	p	w	o	p	k
1	e	x	s	y	t	a	f	c	b	k	e	c	s	s	w	w	p	w	o	p	n
2	e	b	s	w	t	l	f	c	b	n	e	c	s	s	w	w	p	w	o	p	n
3	p	x	y	w	t	p	f	c	n	n	e	e	s	s	w	w	p	w	o	p	k
4	e	x	s	g	f	n	f	w	b	k	t	e	s	s	w	w	p	w	o	e	n

Appendix 1. Example of Dataset in Pandas Data Frame



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	class	cap-shape	cap-surfac	cap-color	bruises	odor	gill-attachı	gill-spacing	gill-size	gill-color	stalk-shap	stalk-root	stalk-surfa	stalk-surfa	stalk-color	stalk-color	veil-type	veil-color	ring-numb	ring-type	spore-prin	populatior	habitat
2	p	x	s	n	t	p	f	c	n	k	e	e	s	s	w	w	p	w	o	p	k	s	u
3	e	x	s	y	t	a	f	c	b	k	e	c	s	s	w	w	p	w	o	p	n	n	g
4	e	b	s	w	t	l	f	c	b	n	e	c	s	s	w	w	p	w	o	p	n	n	m
5	p	x	y	w	t	p	f	c	n	n	e	e	s	s	w	w	p	w	o	p	k	s	u
6	e	x	s	g	f	n	f	w	b	k	t	e	s	s	w	w	p	w	o	e	n	a	g
7	e	x	y	y	t	a	f	c	b	n	e	c	s	s	w	w	p	w	o	p	k	n	g
8	e	b	s	w	t	a	f	c	b	g	e	c	s	s	w	w	p	w	o	p	k	n	m
9	e	b	y	w	t	l	f	c	b	n	e	c	s	s	w	w	p	w	o	p	n	s	m
10	p	x	y	w	t	p	f	c	n	p	e	e	s	s	w	w	p	w	o	p	k	v	g
11	e	b	s	y	t	a	f	c	b	g	e	c	s	s	w	w	p	w	o	p	k	s	m
12	e	x	y	y	t	l	f	c	b	g	e	c	s	s	w	w	p	w	o	p	n	n	g
13	e	x	y	y	t	a	f	c	b	n	e	c	s	s	w	w	p	w	o	p	k	s	m
14	e	b	s	y	t	a	f	c	b	w	e	c	s	s	w	w	p	w	o	p	n	s	g
15	p	x	y	w	t	p	f	c	n	k	e	e	s	s	w	w	p	w	o	p	n	v	u

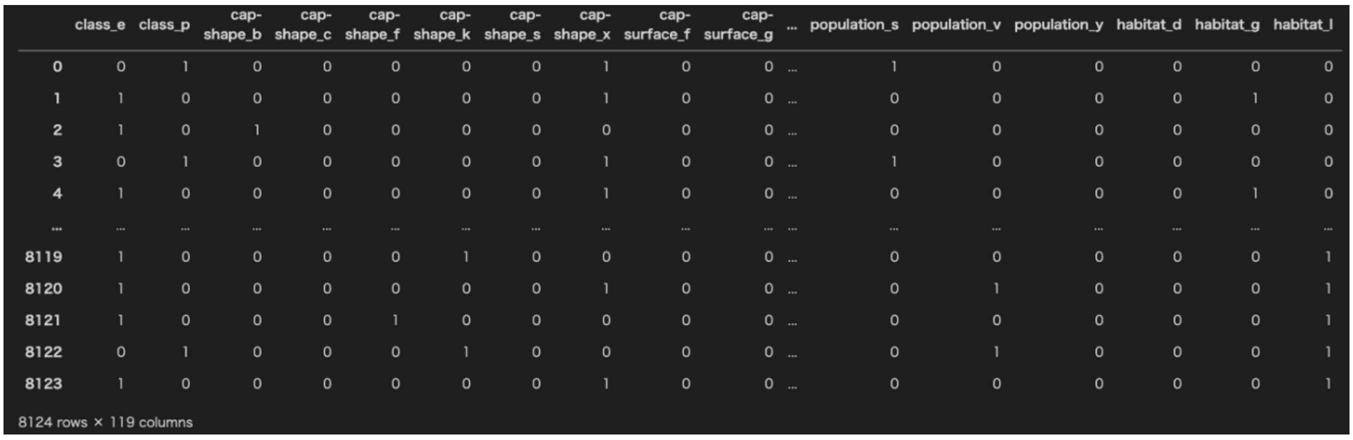
Appendix 2. Dataset in csv file Link



```
for col in mushroom_data.columns:
    print(col,':', mushroom_data[col].unique())
```

```
class : ['p' 'e']
cap-shape : ['x' 'b' 's' 'f' 'k' 'c']
cap-surface : ['s' 'y' 'f' 'g']
cap-color : ['n' 'y' 'w' 'g' 'e' 'p' 'b' 'u' 'c' 'r']
bruises : ['t' 'f']
odor : ['p' 'a' 'l' 'n' 'f' 'c' 'y' 's' 'm']
gill-attachment : ['f' 'a']
gill-spacing : ['c' 'w']
gill-size : ['n' 'b']
gill-color : ['k' 'n' 'g' 'p' 'w' 'h' 'u' 'e' 'b' 'r' 'y' 'o']
stalk-shape : ['e' 't']
stalk-root : ['e' 'c' 'b' 'r' '?']
stalk-surface-above-ring : ['s' 'f' 'k' 'y']
stalk-surface-below-ring : ['s' 'f' 'y' 'k']
stalk-color-above-ring : ['w' 'g' 'p' 'n' 'b' 'e' 'o' 'c' 'y']
stalk-color-below-ring : ['w' 'p' 'g' 'b' 'n' 'e' 'y' 'o' 'c']
veil-type : ['p']
veil-color : ['w' 'n' 'o' 'y']
ring-number : ['o' 't' 'n']
ring-type : ['p' 'e' 'l' 'f' 'n']
spore-print-color : ['k' 'n' 'u' 'h' 'w' 'r' 'o' 'y' 'b']
population : ['s' 'n' 'a' 'v' 'y' 'c']
habitat : ['u' 'g' 'm' 'd' 'p' 'w' 'l']
```

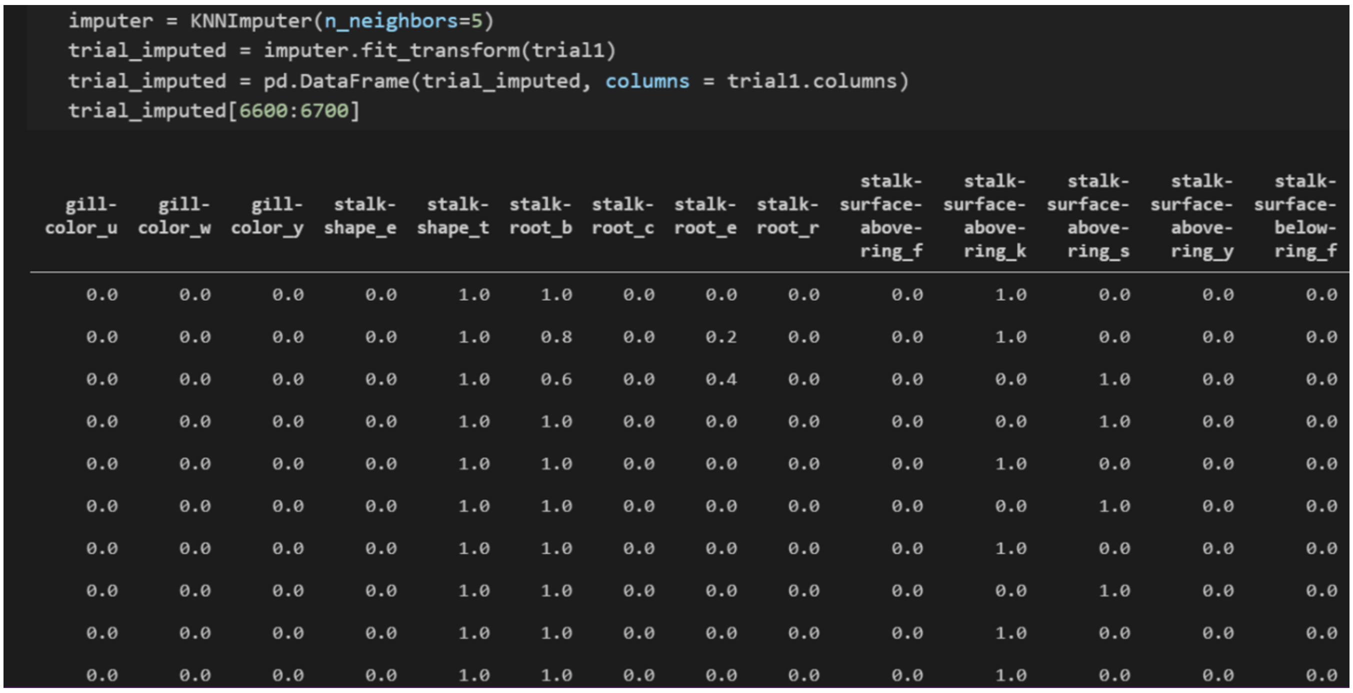
Appendix 3. Unique values of each column



	class_e	class_p	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_f	cap-surface_g	...	population_s	population_v	population_y	habitat_d	habitat_g	habitat_l
0	0	1	0	0	0	0	0	1	0	0	...	1	0	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	...	0	0	0	0	1	0
2	1	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0
3	0	1	0	0	0	0	0	1	0	0	...	1	0	0	0	0	0
4	1	0	0	0	0	0	0	1	0	0	...	0	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8119	1	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	1
8120	1	0	0	0	0	0	0	1	0	0	...	0	1	0	0	0	1
8121	1	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	1
8122	0	1	0	0	0	1	0	0	0	0	...	0	1	0	0	0	1
8123	1	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	1

8124 rows × 119 columns

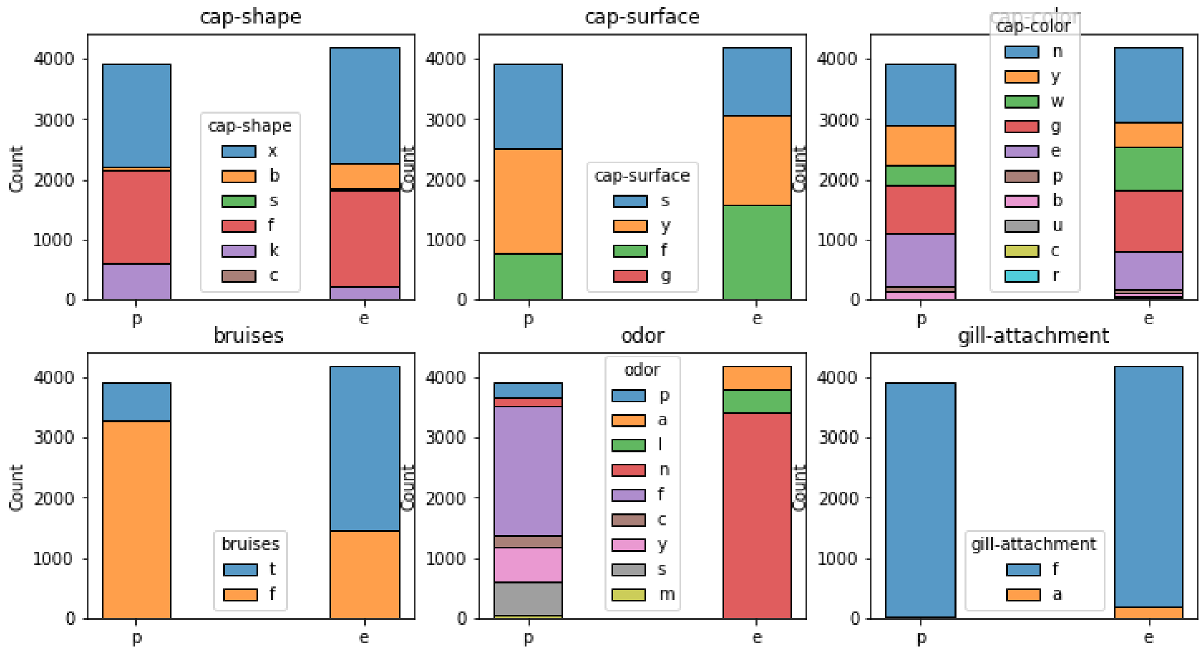
Appendix 4. Example of Transformed Dataset Link

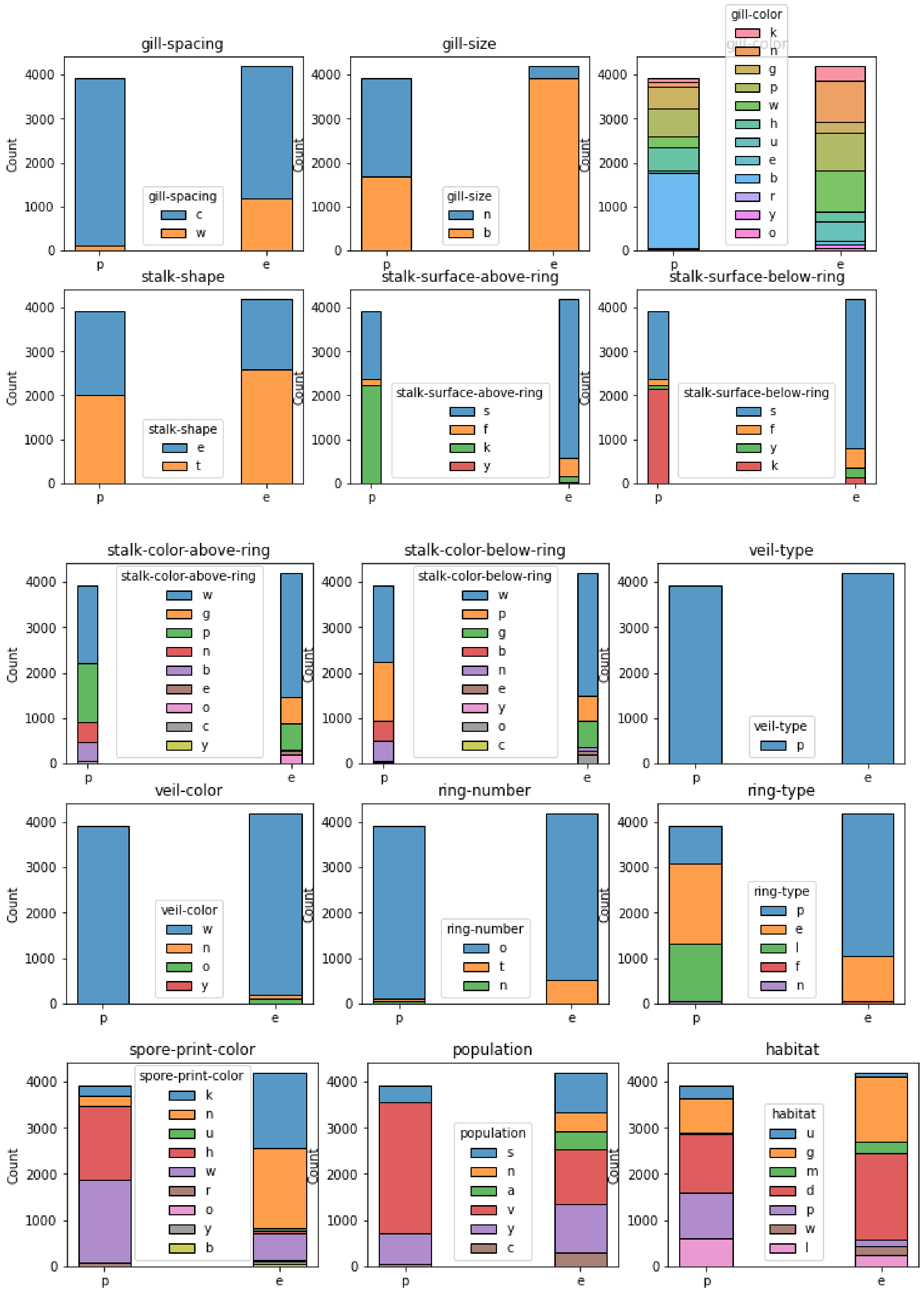


```
imputer = KNNImputer(n_neighbors=5)
trial_imputed = imputer.fit_transform(trial1)
trial_imputed = pd.DataFrame(trial_imputed, columns = trial1.columns)
trial_imputed[6600:6700]
```

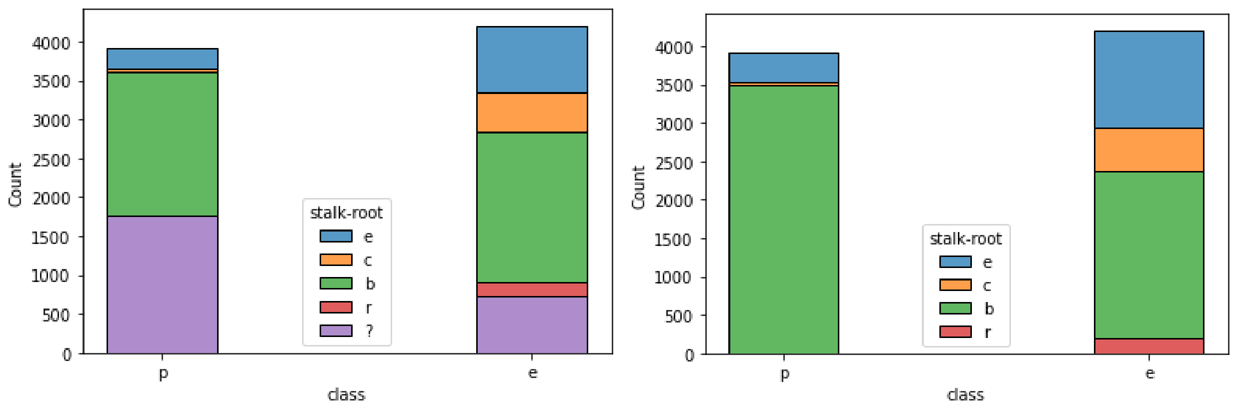
gill-color_u	gill-color_w	gill-color_y	stalk-shape_e	stalk-shape_t	stalk-root_b	stalk-root_c	stalk-root_e	stalk-root_r	stalk-surface-above-ring_f	stalk-surface-above-ring_k	stalk-surface-above-ring_s	stalk-surface-above-ring_y	stalk-surface-below-ring_f
0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	0.8	0.0	0.2	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	0.6	0.0	0.4	0.0	0.0	0.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0

Appendix 5. Example of Dataset after Applying KNN method

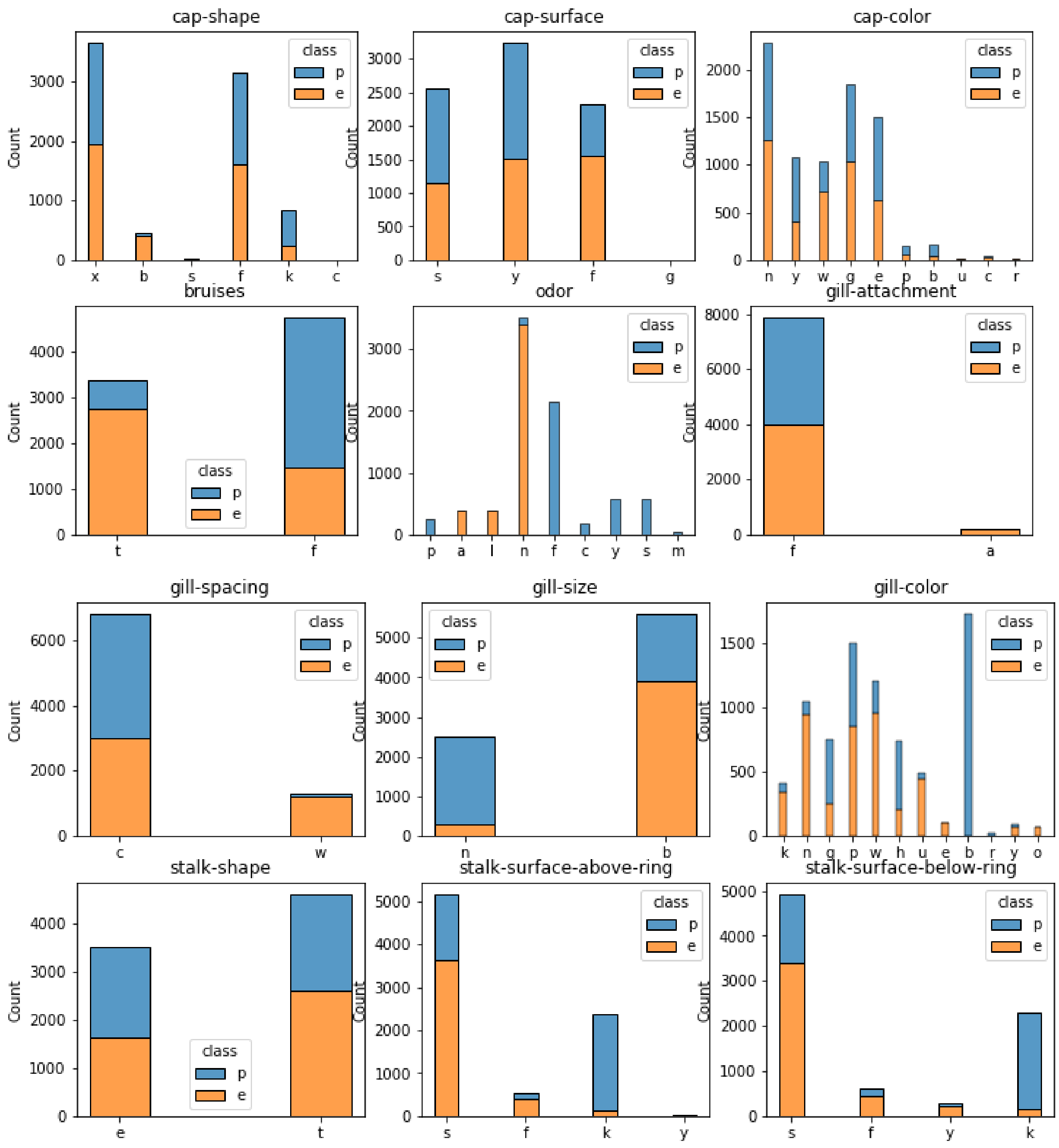


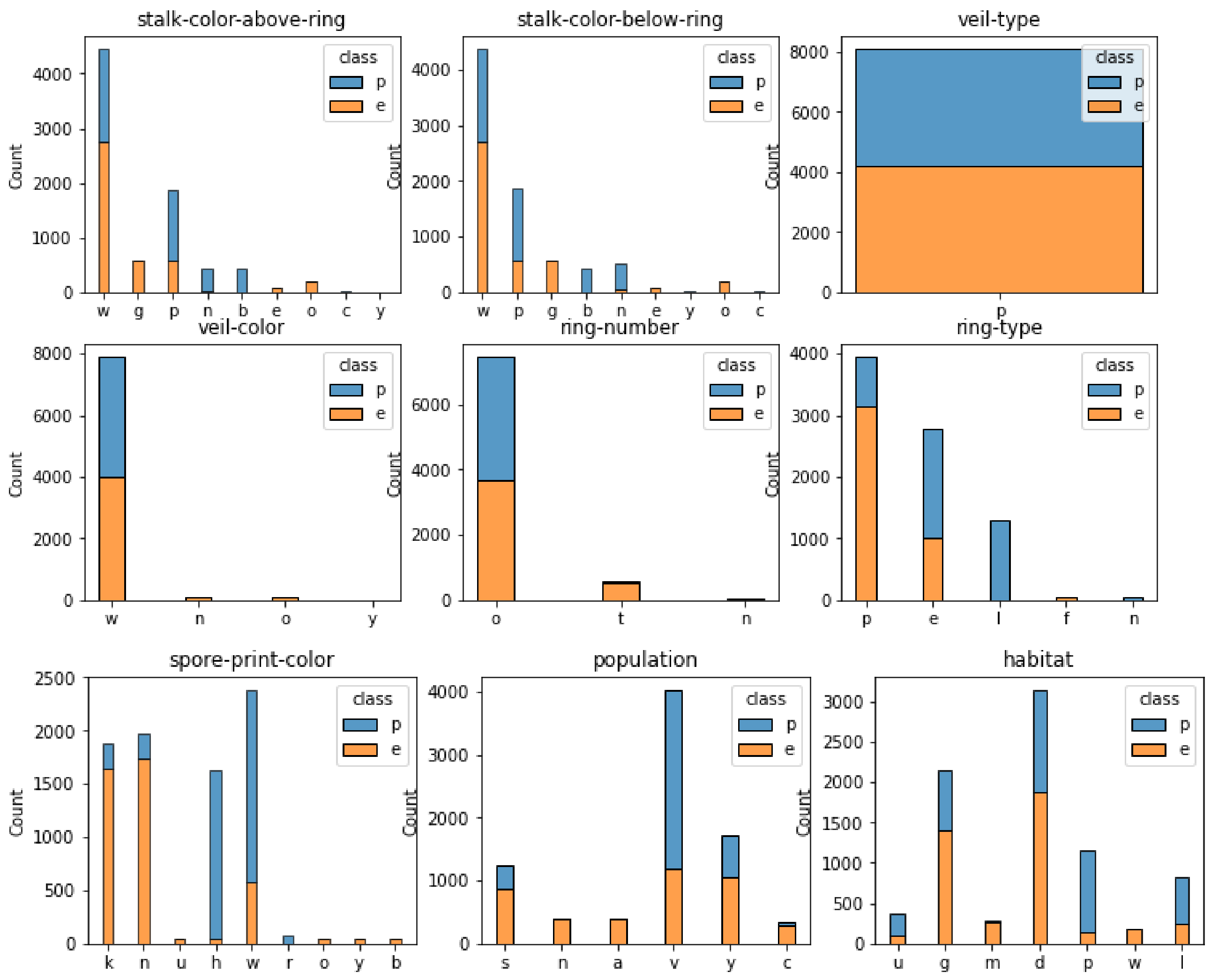


Appendix 6. Histogram for Each Column

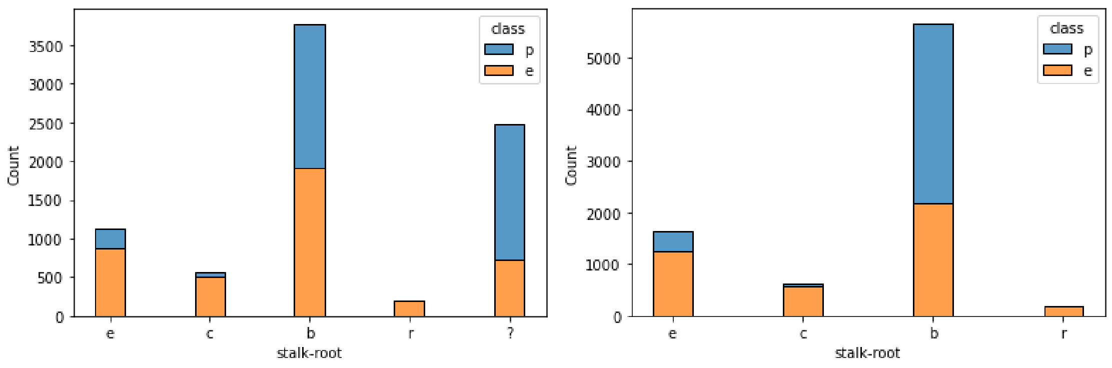


Appendix 7. Histogram for the Column with Missing Data & After Implementing KNN Method

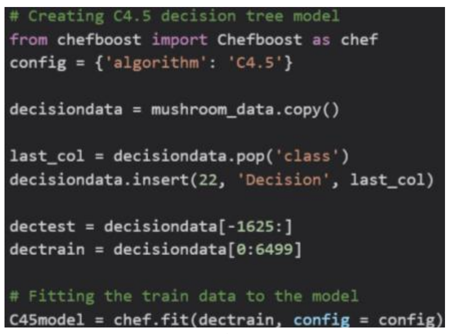




Appendix 8. Histogram for Each Column



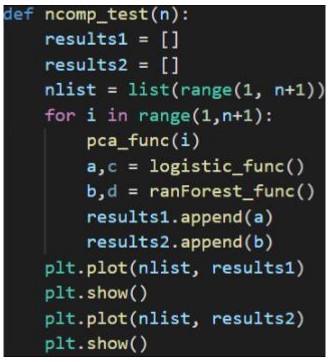
Appendix 9. Histogram for the Column with Missing Data & After Implementing KNN Method



```
# Creating C4.5 decision tree model
from chefboost import Chefboost as chef
config = {'algorithm': 'C4.5'}

decisiondata = mushroom_data.copy()

last_col = decisiondata.pop('class')
decisiondata.insert(22, 'Decision', last_col)

dectest = decisiondata[-1625:]
dectrain = decisiondata[0:6499]

# Fitting the train data to the model
C45model = chef.fit(dectrain, config = config)
```

Appendix 10. Example Code of Creating Decision Tree Model

Appendix 11. C4.5 Decision Tree Code Link



```python
def ncomp_test(n):
    results1 = []
    results2 = []
    nlist = list(range(1, n+1))
    for i in range(1,n+1):
        pca_func(i)
        a,c = logistic_func()
        b,d = ranForest_func()
        results1.append(a)
        results2.append(b)
    plt.plot(nlist, results1)
    plt.show()
    plt.plot(nlist, results2)
    plt.show()
```

Appendix 12. Example Code of Plotting Accuracy Results

Appendix 13. Data Cleaning Code Link

Appendix 14. Data Visualisation Link

Appendix 15. Testing Machine Learning Accuracy by PCA and KNN Link

Appendix 16. Testing Machine Learning accuracy by PCA only Link

Appendix 17. GitHub Repository Link

REFERENCES

[1] Chen, L. et al. (2006) "Mushroom poisoning surveillance analysis, Yunnan province, China, 2001-2006", *Outbreak, Surveillance & Investigation Reports Journal*, 1(1), pp. 8-11.

[2] White, J. et al. (2019) "Mushroom poisoning: A proposed new clinical classification", *Toxicon*, 157, pp. 53-65. doi: 10.1016/j.toxicon.2018.11.007.

[3] Blackwell, M. (2011) "The Fungi: 1, 2, 3 … 5.1 million species?", *American Journal of Botany*, 98(3), pp. 426-438. doi: 10.3732/ajb.1000298.

[4] Chaoqun, Z. (2019) "Recognition and Research of Poisonous Mushroom Based on Machine Learning", *Taigu: Shanxi Agricultural University, Jinzhong, China*.

[5] Eusebi, C. et al. (2008) "Data Mining on a Mushroom Database", *Journal of CSIS, Pace University*, pp. 1-9.

[6] Wibowo, A. et al. (2018) "Classification algorithm for edible mushroom identification", *2018 International Conference on Information and Communications Technology (ICOIACT)*, pp. 250-253. doi: 10.1109/icoiact.2018.8350746