Applied Data Science
Final Project:

Shintaro Osuga

August 29, 2024

Introduction

In nature, there are hundreds of thousands of different genus groups in the kingdoms Animalia, Plantae, and Fungi. For those many hundreds of thousands of organisms, there are hundreds if not thousands of different species contained. Those different species found within each genus are differentiated and classified as so by the characteristics they possess and use. These are very important yet quantifiable, with a single trait sometimes being the distinguishing factor between two species. In this vast ecosystem of millions of different species, with discernibly different yet sometime not apparent differences makes the task of species identification extremely difficult and labor intensive. This identification also is not able to be done by one singular individual and often times requiring multiple distinguished and knowledgeable individuals from each specialization, as the number of species is too large. To streamline and make this process less labor intensive in this ever expanding ecosystem of organisms, a more automated system which can accurately identify species has been yearned for as it would increase the speed of identification and processing new species being found yearly. One such solution in recent times were machine learning algorithms.

Since machine learning algorithms are capable of learning very complex patterns and identifying from a very large data source, it was seen as the most logical and appropriate facet to utilize them. In light of such ideas, when actually implementing such grand machine learning algorithm there was a problem, that being the issue of machine learning algorithms mostly taking in one size for input. Therefore necessitating either a robust encoding which could encompass every kingdom, every genus, and every species without information loss, a uniform method of information namely images, or to split up the task to species with similar qualities and traits. The simplest of those 3 solutions was to split up the task in to smaller more manageable sizes, namely the identification of fungi. Fungi was chosen for this task as they are simple organisms with similar traits amongst all of them which can be quantified and categorized while also being a very large group, comprising their own kingdom being separate from the other organisms in the world. Even though fungi were chosen as their characteristics can be more easily categorized and discerned. The number of total species in the fungi kingdom was very large and though not as large a Animalia with around 10 million species, Fungi had around 1 to 5 million different species. Creating a machine learning model which could identify all 1 to 5 million different species of fungi would be a monumental task.

Therefore, instead of classifying the species a smaller more attainable concept was brought up. The idea was to identify if a mushroom was poisonous or not. As mushrooms can be poisonous, a machine learning approach to identifying if one was based on their outward characteristics would pose to be a simple yet important first step in understanding and creating a model which can accurately identify mushroom species and in the future perhaps even every organisms specie name. As the traits input to the model would be similar to the larger task of classifying specie name, the overall model and capabilities would be similar. Yet the binary task of determining whether a mushroom was poisonous or not would be much simpler and more attainable than predicting millions of different specie names. Furthermore, the identification of mushroom toxicity would be greatly useful for people who wish to forage for their own mush-

rooms, as doing so without proper knowledge would be dangerous and a health concern. Though given as this is a machine learning model, instead of taking what it predicts to heart, it should be taken as a recommendation or first step in identifying the mushroom's toxicity.

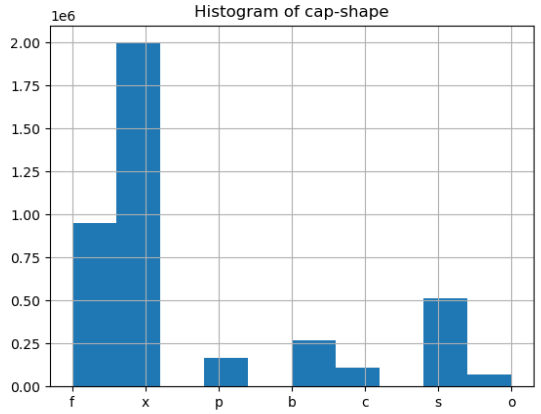
Analysis

The Data

The data consists of over 6 million values, with 21 columns including the class variable. Of the 21 original columns 14 were selected to use as the 7 left out contained little to no information about the mushrooms, with a significantly large portion of their values being Nan. The remaining columns which were used were cap-shape, cap-surface, cap-color, does-bruise-or-bleed, gill-attachment, gill-color, stem-color, ring-type, habitat, season, class, cap-diameter, stem-height, and stem-width. Of the 14 columns being used, 11 of them are categorical including the class. The other 3 columns were continuous numerical values. The numerical variables were discretized in to different numbers categories while training to see the impact of different more precise categorizations on accuracy and predicting whether a mushroom is poisonous or edible. These numbers of categories were 4, 10, 15, 35, 55, 115, which had a good range of small numbers of categories and a very large number of categories.

Cap-Shape

The cap-shape column is a categorical variable which aims to categorize and give information about the shape of the cap of the mushroom. The cap-shape column contained 7 unique categories. Those categories being bell (b), concical (c), convex (x), flat (f), sunken (s), spherical (p), and other (o).



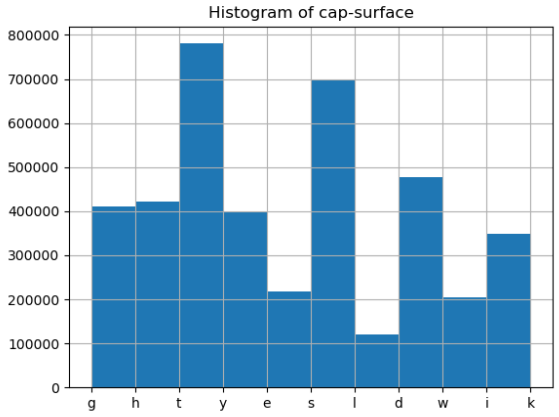
Looking at the histogram distribution of the values it can be seen that the category x, the convex cap shape has the largest number of values. Furthermore, the next most frequent shape is f with s coming right after. The number of f is about half the number of x's there are, with s being about half of f. This is a distribution often seen in the world being called the Pareto principle, where 80% of the values are made up of 20% of the possible outcomes. This can be confirmed by counting the number which are either category x or f and then diving it by the total. This comes out to 2949771 number of values which are x or f, dividing by the total giving 0.7236 or 72.36% of the values are either x or f which is about 28.57% of the possible outcomes. This is intriguing to see as it almost exactly matches the Pareto principle of 80% of the values being composed of 20% of the outcomes.

Cap-Surface

The cap-surface column is a categorical variable which is used to categorize the surface texture of the mushroom. This variable distinguishes the texture of the surface into 11 different categories. Those are

fibrous (i), grooves (g), scaly (y), smooth (s), dry (d), shiny (h), leathery (l), silky (k), sticky (t), wrinkle (w), and fleshy (e).

Looking at the histogram there seems to not be a singular distribution of the variables, with most of the variables having similar quantities of values, with only t, s, and d having proportionally larger number of values than the others. On the other side of the spectrum, category e, i and w seem to have a proportionally smaller number of values as well. Though somewhat similar in vein to the cap-surface in distribution after calculating the categories t, s, and d only amount to 47.98% of the total while being 27.27% of the possible outcomes. It is still very interesting and insightful to see that a very large quantity of the values are either t, s, or d, with the opposite end of the least number of values being e, i, and w.

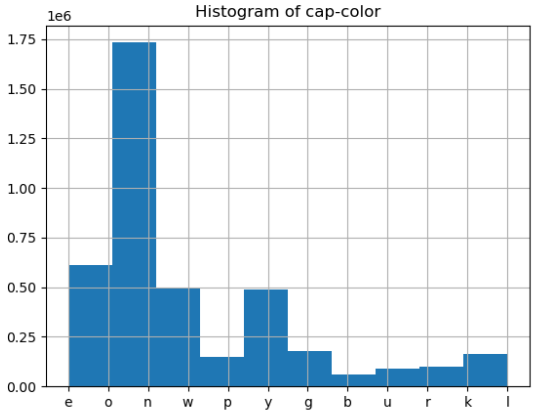


Cap-Color

The cap-color is a categorical variable, which describes the color of the cap of the mushroom into 12 different categories. Those color categories are brown (n), buff (b), gray (g), green (r), pink (p), purple (u), red (e), white (w), yellow (y), blue (l), orange (o), and black (k).

The histogram for this variable is very reminiscent of the cap-shape histogram where a singular category consisting of the vast majority of the values. The category n can be seen having 2 to 3 times more values than the next most frequent category.

By its self, the category n makes up for about 42.5% of the values whereas the next largest category only makes up for about 12.27% of the values. Adding top 20% of the categories, n, w, and y results in 66.76% of the values while consisting of 25% of the categories. Which like the cap-shape, is very close to the Pareto curve where about 80% of the values consists of about 20% of the categories.

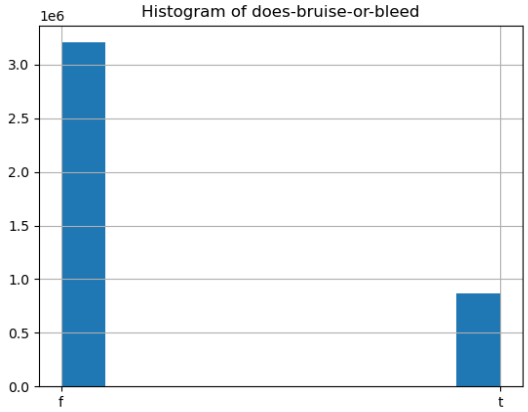


Does-Bruise-or-Bleed

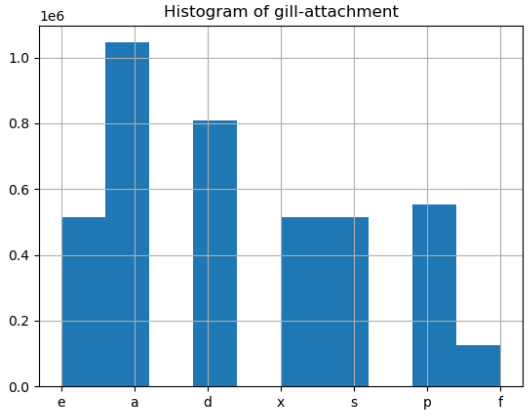
The does-bruise-or-bleed category is a binary category which describes if the mushroom bruises or bleeds. The categories for this column are true (t) or false (f).

The distribution for this variable can be seen to be fairly skewed toward the false category, with the false category having almost 3 times the number of values in it than the true category. About 78.66% of all of the values in the data set are seen to be in the false category while only about 22.34% of the values are seen to be in the true category. This is interesting as it suggests that among the many mushrooms in the

world, most of them bruise or bleed where as only a small portion of them do neither, indicating that most mushrooms bruise or bleed.



Gill-Attachment

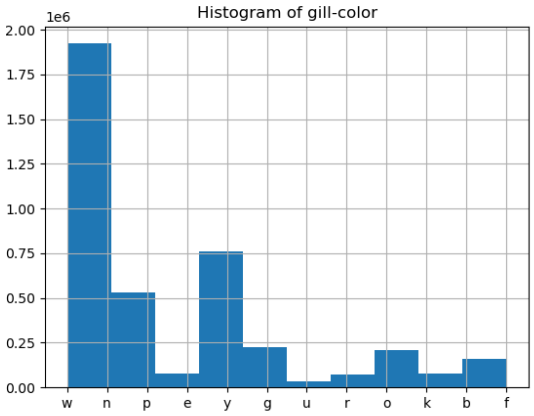


The gill-attachment variable aims to categorically describe the way gills are attached to the mushroom. The different ways a gill could attach to a mushroom are adnate (a), adnexed (x), decurrent (d), free (e), sinuate (s), pores (p), none (f), and unknown (?).

Comparatively to the other distribution of values in the previous variables, the gill-attachment variable seems to have a fairly uniform distribution with only 2 variables being larger than the rest. Category a and d can be seen being much larger than the other 5 variables, with the 5 other variables being relatively close to one another in terms of frequency except for f which is very small compared to even the second infrequent value. Though much larger than the rest, a and d do not really have a disproportionate amount of values looking at the data as a whole except for the f. Looking deeper it can be understood that about 45.5% of the values are either a or d. The smallest category f can be seen to be about 0.03% of the entire data set which is extremely small compared to the others.

Gill-Color

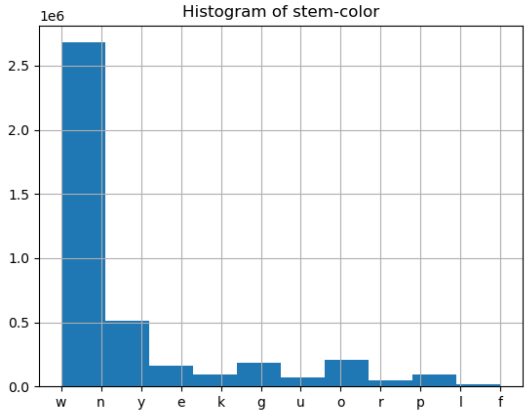
The gill-color variable is used to describe the color of the gill of the mushroom. There are 13 categories for this variable those being brown(n), buff (b), gray (g), green (r), pink(p), purple (u), red (e), white (w), yellow (y), blue (l), orange (o), black(k), and none (f). Those are the same colors seen in the cap-color except for the none which was not seen in the cap-color variable.



The distribution of this is similar to that of the distribution seen in cap-color with one color dominating the other. For gill-color it can be seen that the category w has the largest proportion of values, accounting for more than 29.3% of the total mushrooms in the data set. The next category containing the most number of values is the category y, accounting for 18.66% of the values and n accounting for 17.8% of the values. The category with the smallest proportion of values is u accounting for 0.8% of the total mushrooms. This variable like the cap-color and cap-shape has a similar distribution to the Pareto distribution with w, y, and n accounting for 65.8% of the total values while only being 23.1% of the possible outcomes.

Stem-Color

The stem-color variable like the gill-color and cap-color aims to categorize the color of the mushrooms stem into 13 categories. These categories are brown(n), buff (b), gray (g), green (r), pink(p), purple (u), red (e), white (w), yellow (y), blue (l), orange (o), black(k), and none (f). These are the same as the gill-color and also is similar to cap-color except for the none color.



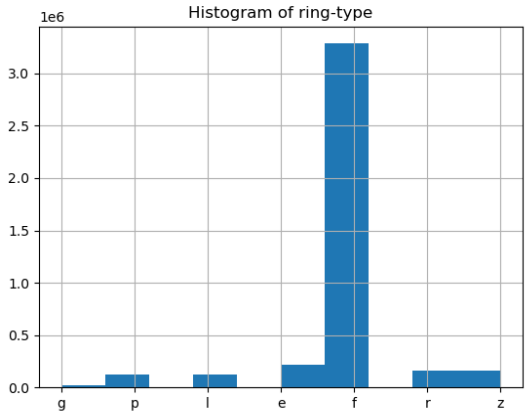
The distribution of the stem-color is very distinctly similar to the other 2 color variable distributions, with a few colors being dominant over the others. This color is n and w with y being about a third of those 2. The categories n and w are found in a very large proportion of values with w being found in 32.16% of all values and n being found in 33.6ôf all values. The category y, though smaller than the previous 2, still has a larger proportion than the rest, being contained in 12.60% of all values. Combing all of the values shows that n, w, and y are found in 78.36% of all values, while being only 23% of all possible outcomes. This is thus far the variable which follows the Pareto distribution the closest and shows only a few categories dominate the entire population.

Ring-Type

The ring-type variable aims to categorize the type of ring which is found on the mushroom. There are cobwebby (c), evanescent (e), flaring (r), grooved (g), large (l), pendant (p), sheathing (s), zone (z), scaly (y), movable (m), none (f), and unknown (?).

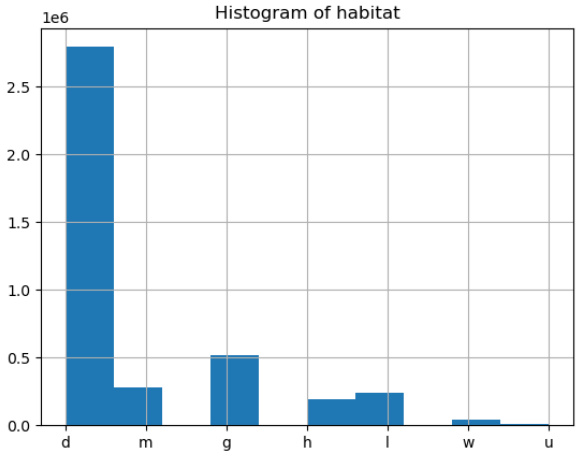
The distribution found for the ring-type variable is very skewed to the category f. It is found that the ring type f is found in 80% of all mushrooms while the next most frequent ring type was e with only 5% and the next being r with 3.88%. For this variable it can be said that one value, f which accounts to about

14% of all possible outcomes is contained in over 80% of all mushrooms. This like the color variables, very strongly resembles the Pareto distribution.



Habitat

The variable habitat categorizes the different habitats which the mushrooms are found in. There are 7 categories, those being grasses (g), leaves (l), meadows (m), paths (p), heaths (h), urban (u), waste (w), and woods (d).



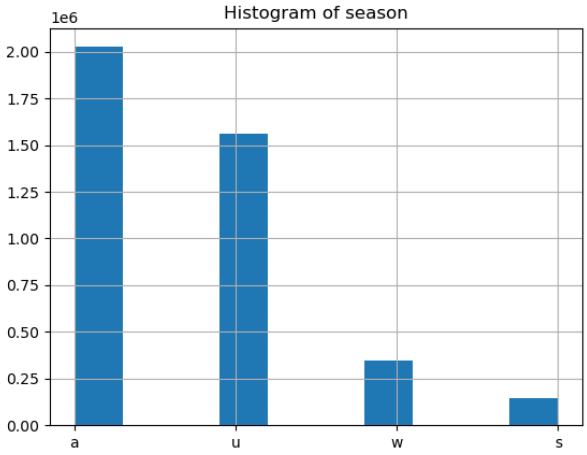
The distribution again is similar to the Pareto distribution where a few categories are found in the majority of mushrooms. It can be seen that the habitat d which is woods is the most common location where mushrooms are found pertaining to about 68.60% of the entire population. The next most frequent habitat which mushrooms are found is g, or the grasses, which pertain to about 12.67% of the entire data set. With those 2 variables about 81.27% of the entire population are found in these two habitats, while 2 habitats correlates to about 28% of the total possible habitats. Showing this variable like the cap-color, stem-color, and gill-color, also closely resembles a Pareto Distribution.

Season

The season column categorizes the 4 seasons in to 4 categories. These categories are spring (s), summer (u), autumn (a), and winter (w).

The distribution of the different categories show that most mushrooms grow in either autumn or summer while there being a very large decrease in the number of mushrooms in the winter and spring. Autumn is seen to contain about 49.7% of all the mushrooms while summer is seen to contain 38.25% of all mushrooms in the data set. Together it can be seen that over 87.96% of all mushrooms in the data set are

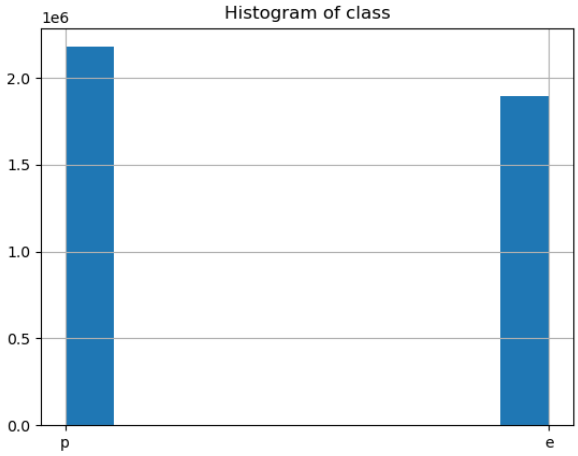
found in the autumn or summer. This shows that of all of the mushrooms 87.96% of them are found in 50% of the year.



Class

The class variable is the y or the predicted class of the project. This variable holds information on whether a mushroom is poisonous or edible. This is categorized as either poisonous (p) or edible (e).

The distribution of mushrooms which are poisonous and edible can be seen to be fairly equal and almost uniform. This is good as a skew in the predicted value will result in a skewed model, and therefore having equal or close to equal numbers of each category being classified will help the model fit the actual classes instead of the skew seen in the labels.

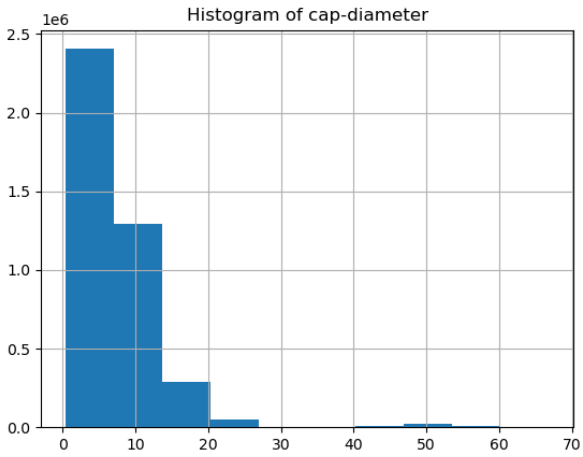


Cap-Diameter

The cap-diameter is 1 of the 3 numerical continuous variables which is present in the data set. The cap-diameter variable measures the cap diameter of the mushrooms in the data set in cm. This variable is later discretized into different number of categories.

The overall distribution of the values are fairly left skewed with the values tending to be between 0 and 10. The mean of this variable is calculated to be 7.269 which is what is observed in the histogram. Furthermore, the minimum value is seen to be 0.34 and the maximum value is seen to be 66.89. The standard deviation is found to be 6.07. It can be stated that the majority of the mushrooms in the data set have a cap-diameter between 0 and 7 from both the histogram and the descriptive analysis performed.

The mean, min, max, and standard deviation before the data cleaning was 6.79, 0.22, 66.89, and 5.279, which has a slightly smaller mean, min, and standard deviation compared to after the cleaning. The cleaning process must have removed many values which were smaller and therefore resulted in a slightly smaller mean, min, and standard deviation.

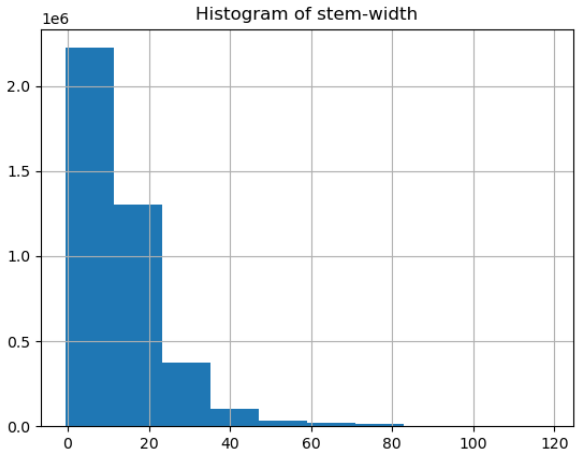


Stem-Width

The variable stem-width is the 2nd continuous numerical variable in the dataset. This variable measures the width of the stem of the mushroom in cm. This is later discretized into different numbers of categories later.

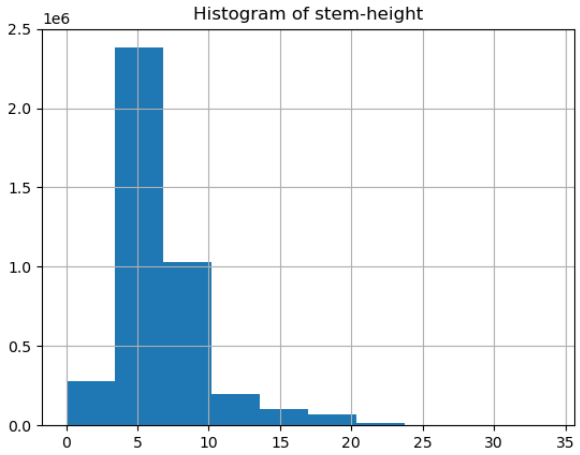
The distribution of stem-width can be seen to be similar to that of the cap-diameter variable. Having a very strong left skew with most of the values tending to be between 0 and 20. The stem-width variable is recorded to have a mean of 12.89 and a min of -0.64 and a max of 118.68 cm. The standard deviation was calculated to be 11.17 cm which is most likely due to the high max value of 118.68 cm, even though the mean is found to be only 11.17 cm. This shows that a vast majority of the mushrooms have a stem-width between 0 and 20 while there are some who have larger stems and furthermore even larger ones like 118 cm, which are not common but still recorded and seen.

The mean, min, max, and standard deviation before cleaning was 12.36, -0.64, 118.68, and 9.968 correspondingly. The mean and standard deviation before cleaning is seen to be smaller marginally. This indicates that some smaller values were lost during the cleaning of the data.



Stem-Height

The stem-height variable is the last of the 3 numerical continuous variables which are found in this data set. This variable records the height of the stem of the mushroom in cm. This variable is later discretized into different numbers of categories later.



The distribution of the stem-height can be seen to be almost Gaussian like with a extremely high peak in the center. There is seen a vast majority of the stem heights being between around 4 to a little under 10. The mean for stem-height was found to be 6.6 cm while the min was 0 and max of 33.9 cm. The standard deviation was found to be 3.166 cm which is consistent with what is seen in the histogram. There can be seen a larger arm to the right with the values reaching a max of 33.9, being further than the 0 which is only 6.6 cm from the mean while the max is over 27 cm away. This shows that the height of the stem is typically very short and only around a few centimeters to maybe 10 at most, while there exists some mushrooms with much longer stems.

The mean, min, max, and standard deviation before cleaning was 6.698, 0, 37.7, and 3.3 correspondingly. It can be seen that most of these values stayed relatively the same with the mean increasing by a very small margin, the max decreasing by a little, and the standard deviation decreasing marginally. This means that the data cleaning only removed "normal" values and the max value, keeping the overall shape of the distribution relatively the same.

EDA: Statistical Analysis and Visualizations

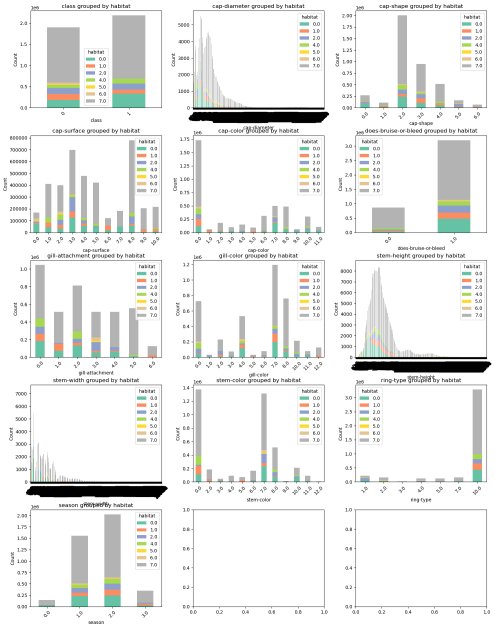


Figure 1: Habitat Grouping

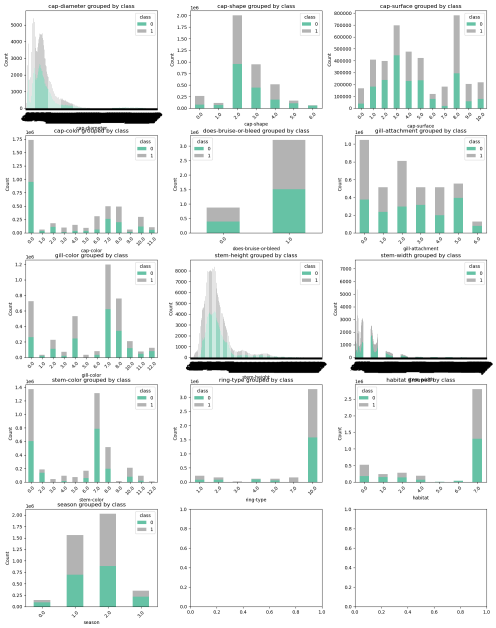


Figure 2: Class Grouping

Grouping the classes together to see the ratio distribution of classes throughout all of the variables, it can seen that for the most part the proportions of values which are edible and poisonous are similar and there are no crazy skews.

This is a very good but also an interesting distribution to see as it indicates that there are minimal differences between mushrooms which are poisonous and edible. On the other hand it is good to see that it is mostly similar in proportions in all categories as it shows that it is not one variable which determines whether a mushroom is poisonous or edible ensuring that the solution to this project is not trivial and requires some sort of algorithm to determine.

Grouping the habitats together to show the distribution of mushrooms of each habitat compared to the other variables shows that proportionally the habitat 7 has the largest distribution of values by far compared to the other habitats. Habitat 7 is the woods habitat, which was seen to be the largest group containing the majority of the mushrooms.

It can also be analyze that the most common cap-surface of the mushroom by habitat greatly changes, with cap-surface fibrous being even more common in habitat 0 or grasses than woods. These tables help analyze the frequency of each categorical value for each habitat.

This stacked bar plot shows the distribution of each variable for each season.

One common trend is that the 2 most common seasons, autumn and summer are just about always equal to one another in every category. The other seasons like spring and winter have increased proportions occasionally. One instance of this is gill-attachment 0 adnate being the most common gill attachment type for spring and winter, with 2 or decurrent gill attachment type being the second most common for winter while pores being the second most common gill attachment type for spring.

Algorithms

The Algorithms used for this Project were Naive Bayes, K-Means, KNN, Decision Tree, and Random Forest. SVMs were not used as the runtime necessary to run on the entire data spanned several days.

Naive Bayes

Naive Bayes algorithms are one of the very first machine learning categorization algorithms to be created and utilized, seeing its first use on text categorization. It is able to model data sets and patterns by "naively" assuming each variable is independent and is not related, then modeling each of the inputs to calculate the probability of any outcome occurring given the observed values in the train data set. As Naive Bayes works well for text classification where the inputs are more categorical and less continuous, it was chosen to be used as the data used in this project were mostly categorical with 3 continuous variables which were discretized.

```
def NaiveBayes_classifier(train_df:pd.DataFrame,
                          train_labels:pd.Series,
                          test:bool = False,
                          test_df:pd.DataFrame=None,
                          config:list[int] = [10]) -> int:

    from sklearn.naive_bayes import GaussianNB

    nb = GaussianNB(var_smoothing=config[0])

    if test == False:
        xtrain, xtest, ytrain, ytest = train_test_split(train_df,
                                                        train_labels,
                                                        test_size=0.3)

        nb.fit(xtrain, ytrain)
        ypred = nb.predict(xtest)
        acc = metrics.accuracy_score(y_true=ytest, y_pred=ypred)

        return "NaiveBayes", acc
    else:
        nb.fit(train_df, train_labels)
        ypred = nb.predict(test_df)
        pred_out = pd.DataFrame(ypred, columns=["class"])

        return "NaiveBayes", pred_out
```

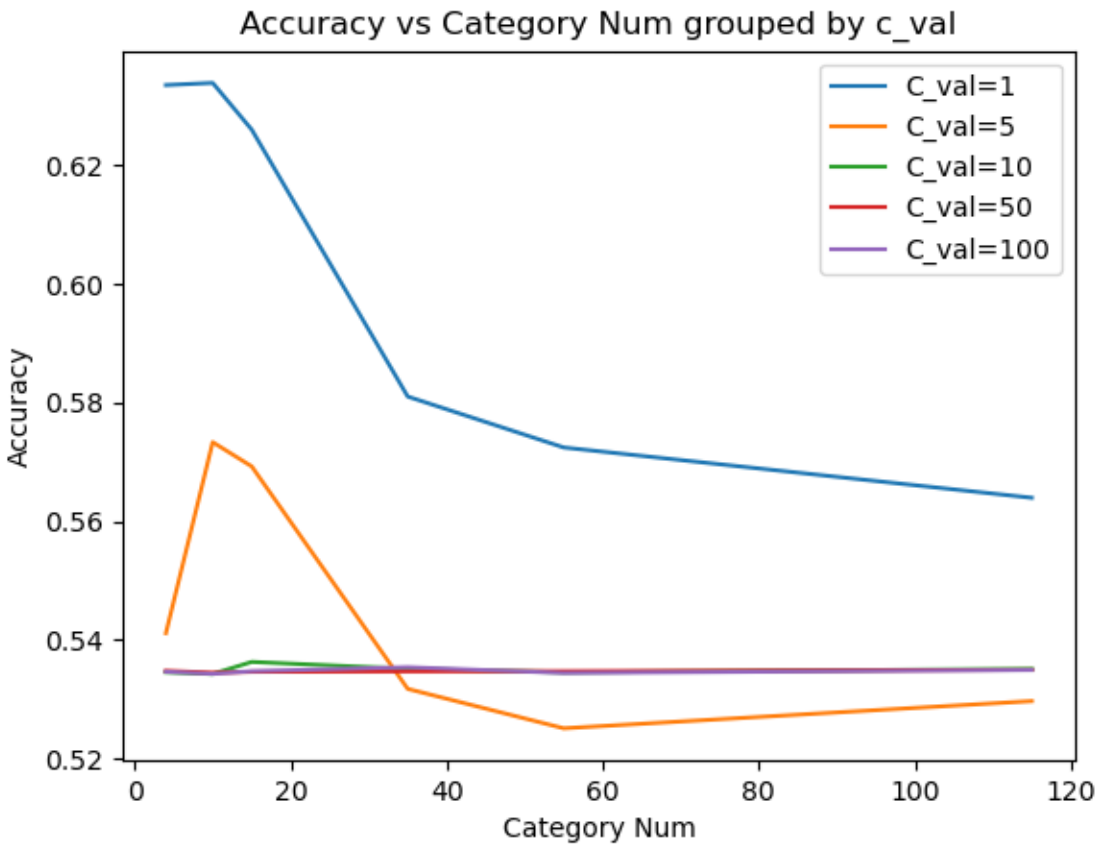
For the Naive Bayes algorithm, there is 1 main hyperparameter which can be tuned. This is the var_smoothing parameter. This parameter can change the variance of the curve which tries to fit the data, In the case of the Gaussian Naive Bayes algorithm which was used for this project, the curve is Gaussian and therefore adding variance to the data or the curve will help the algorithm to better fit and stabilize. The var_smoothing hyperparameter was changed between 1 and 5 with changing categorical numbers for the numerical variables which were discretized. The results are given in table 1 and table 2.

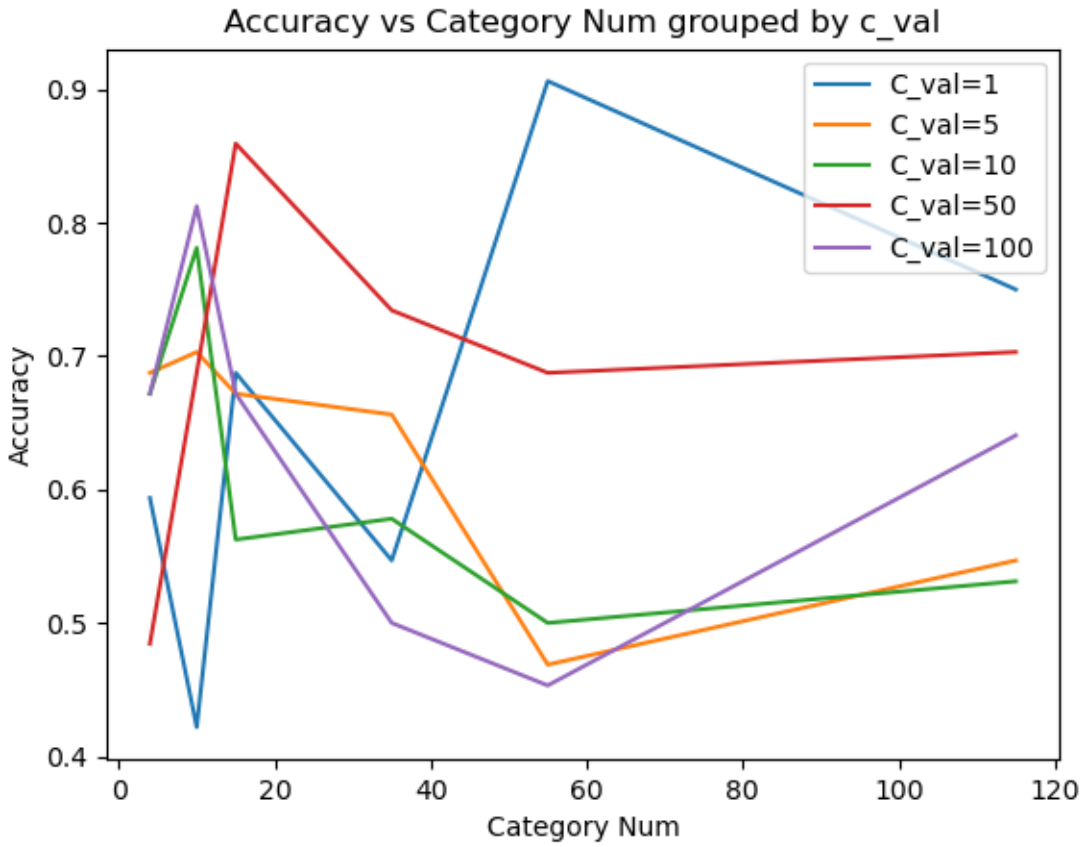
		Var Smoothing Value				
		1	5	10	50	100
Category Num	4	0.6335	0.5412	0.5346	0.5348	0.5347
	10	0.6339	0.5733	0.5343	0.5345	0.5343
	15	0.6260	0.5693	0.5363	0.5347	0.5348
	35	0.5810	0.5318	0.5352	0.5347	0.5354
	55	0.5724	0.5251	0.5346	0.5347	0.5345
	115	0.5640	0.5297	0.5351	0.5350	0.5345

Table 1: Accuracy given Category Number and Var Smoothing Value

		Var Smoothing Value				
		1	5	10	50	100
Category Num	4	0.59375	0.6875	0.671875	0.484375	0.671875
	10	0.421875	0.703125	0.78125	0.6875	0.8125
	15	0.6875	0.671875	0.5625	0.859375	0.671875
	35	0.546875	0.65625	0.578125	0.734375	0.5
	55	0.90625	0.46875	0.5	0.6875	0.453125
	115	0.75	0.546875	0.53125	0.703125	0.640625

Table 2: Runtime (s) given Category Number and Var Smoothing Value





As it can be seen from the table, the accuracy of the Naive Bayes model seems to benefit from having less categories and less var smoothing with the highest accuracy achieved with 10 categories and a var_smoothing value of 1. Naive Bayes preferring less categories might be due to the fact that the other originally categorical values are expressive enough of the patterns for poisonous mushrooms that increasing the number of discretization categories only confuses the model further. A low var smoothing variable may be because the original Gaussian curve is able to model the data well enough and adding further variation just increases the distribution and variation of the data points to the point where the model is more confused than fit.

The runtime seems to also prefer less categories, increasing in the runtime by around 40% but reaching its peak at around 55 categories. It makes sense as to why the increase in discretization categorical numbers increases the runtime as the more categories the data set has the more different variations and variables it must compute the probability for thereby increasing the total compute time. the variation of the runtime seen in the var smoothing indicates that there must not be much of an effect in increasing var smoothing which also makes sense as the var smoothing is simply a parameter which is added to the data points to change the variation and therefore should not significantly increase the over all runtime of the model.

Through this testing of the Naive Bayes on modeling and predicting toxicity of mushrooms, it was found that a Naive Bayes model with var_smoothing of 1 and 10 categories for the discretized variables resulted in the highest accuracy of 63.39%.

Model Name	Accuracy	Model Runtime (s)	Var Smoothing	Category Numbers
Naive Bayes	0.6339	0.4219	1	10

K-Means

K-Means is a classification algorithm which aims to classify data points by their distance from one another, more so by their distance to each other and their centroid, which there is a specified number of. It does very well in data which have can be spatially divided and "clustered" together, which hopefully is the case for this project's mushroom data. The theory is that there are clusters of mushrooms with similar traits which are poisonous and those which are not, and by spatially dividing those, the K-Means algorithm is able to cluster just the spatially close poisonous mushroom and non-poisonous mushrooms.

```
def Kmeans_classifier(train_df:pd.DataFrame,
                      train_labels:pd.Series,
                      test:bool = False,
                      test_df:pd.DataFrame=None,
                      config:list[Union[int, str]] = [2]) -> int:

    from sklearn.cluster import KMeans
    kmeans_model = KMeans(n_clusters=config[0])

    if test == False:

        xtrain, xtest, ytrain, ytest = train_test_split(train_df,
                                                        train_labels,
                                                        test_size=0.3)
        kmeans_model.fit(xtrain, ytrain)
        ypred = kmeans_model.predict(xtest)
        acc = metrics.accuracy_score(y_true=ytest, y_pred=ypred)

        return "Kmeans", acc
    else:
        kmeans_model.fit(train_df, train_labels)
        ypred = kmeans_model.predict(test_df)
        pred_out = pd.DataFrame(ypred, columns=["class"])

        return "Kmeans", pred_out
```

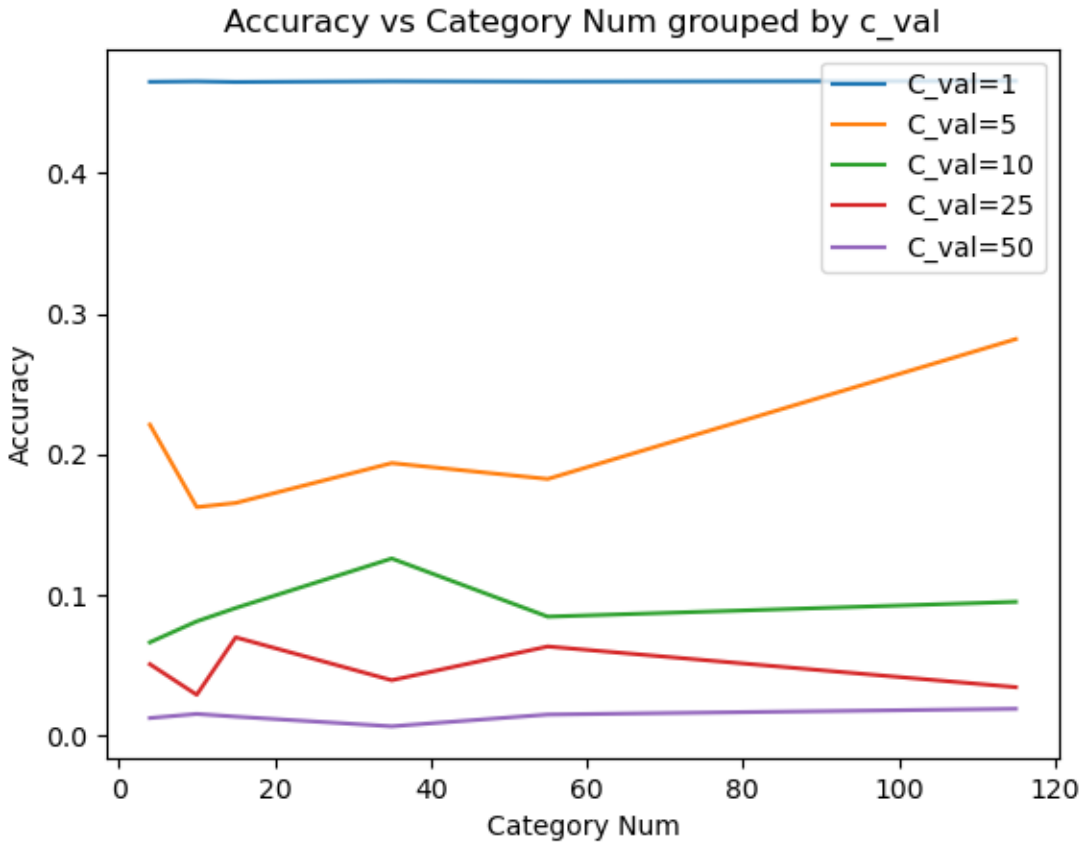
As so, the K-Means algorithm has 1 hypterparameter which was tuned, this was of course the n_clusters. This parameter changes the number of clusters or centroids the algorithm tries to fit data to. As too small of a value obscures the line between what should be spatially divided and what should not be, a cluster number can vary the accuracy of the model greatly. Along with the n_clusters the number of categories for the discretized variables are again changed. The results of this test was as seen in table 3 and table 4.

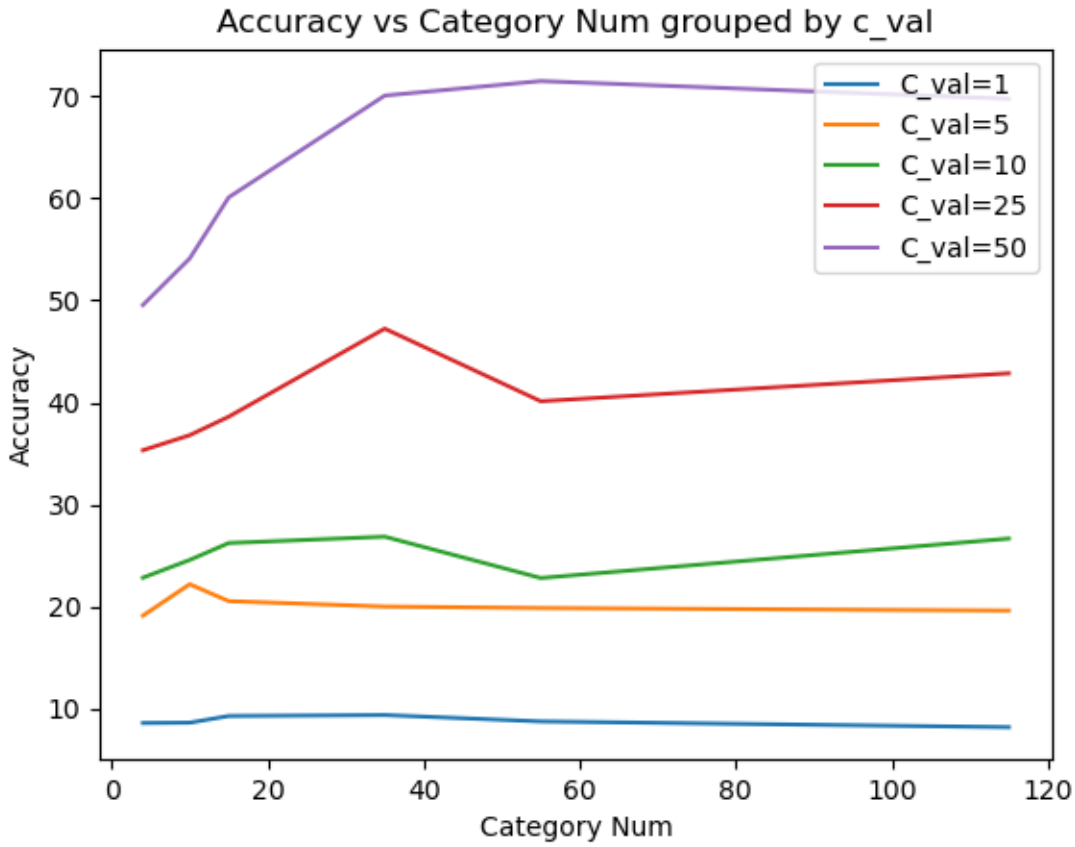
		Cluster Count				
		1	5	10	50	100
Category Num	4	0.4649	0.2212	0.0665	0.0510	0.0127
	10	0.4652	0.1626	0.0814	0.0292	0.0156
	15	0.4648	0.1656	0.0909	0.0701	0.0137
	35	0.4653	0.1939	0.1261	0.0396	0.0068
	55	0.4651	0.1826	0.0848	0.0636	0.0152
	115	0.4656	0.2821	0.0953	0.0347	0.0193

Table 3: Accuracy given Category Number and Cluster Count

Category Num	Cluster Count 1	5	10	50	100
4	8.609375	19.125	22.84375	35.328125	49.546875
10	8.640625	22.203125	24.578125	36.796875	54.109375
15	9.296875	20.546875	26.234375	38.609375	60.09375
35	9.390625	20.015625	26.859375	47.234375	70.0625
55	8.765625	19.875	22.8125	40.125	71.484375
115	8.203125	19.625	26.671875	42.859375	69.75

Table 4: Runtime (s) given Category Number and Cluster Count





Viewing the results from the test, it can be seen that the number of clusters greatly impacted the accuracy of the model. More so, the larger the number of clusters became the accuracy of the model became smaller or worse. Surprisingly the accuracy of the model did not change as the number of categories which the continuous variables were discretized into increased, all of them being within a percent of one another. The accuracy droping as the number of clusters increasing is to be expected but it was believed at least more than 1 cluster would increase the accuracy for it to drop at very higher numbers of clusters. This is because having more clusters would allow for the model to be able to make more different classification and spatially divide mushrooms amongst the same or similar traits. But in reality it was seen that past 1 the number of clusters only served to decrease the accuracy of the model. Furthermore, it was believed that more categories discretizing the continuous variables would increase the accuracy. This is because, more precised categorization of the mushrooms cap-diameter, stem-height, and stem-width should allow for more precise clustering and spatial dividing of the mushrooms between poisonous and non-poisonous mushrooms. Though in the test it was discovered that the number of categories did not change the accuracy of the model in any way which was distinguishable from randomness.

The runtime also seems to follow a similar pattern to the accuracy where the number of categories the variables were discretized into did not change the runtime a significant amount but the number of clusters did. the runtime greatly increasing at the number of clusters increase is to be expected, as the more clusters there are the more calculations between centroids the model must do, but the incremental change in runtime for the increase in categorical number was surprising but still understandable. This is most likely because the number of unique categories in the variables do not change the overall number of computations which must be done and therefore the runtime roughly stays the same between all of the different categories.

Through the test it was seen that the best model for K-Means utilized 1 cluster and any value of category numbers for discretized variables with a runtime of 8.2 seconds and a accuracy of 46.56%. Though the precise accuracy of the model changes incrementally from category number to category number, the whole percentage value does not change at 46%.

Model Name	Accuracy	Model Runtime (s)	Cluster Count	Category Numbers
K-Means	0.4656	8.2031	1	115

K-Nearest Neighbors

KNN stands for K-Nearest Neighbors. This is a classification algorithm which is similar to the K-Means algorithm but instead of classifying and clustering data points near a centroid, the algorithm aims to classify points by which cluster of points they are closest to. This algorithm therefore calculates the distance of a point against all other points and determines the point which is closest and then classifies the point as the classification the nearest point had. To be determined a neighborhood, k number of nearest neighbors are checked, which the algorithm uses to classify the point. Though similar in idea to K-Means, KNN algorithms are typically seen as more robust, being able to classify values better than K-Means. Theoretically, as the data points are able to be spatially positioned and divided similar to K-Means, it was thought that KNN would be great as well since it is able to not just spatially divide points but also divide points amongst the points themselves, since it is able to find neighborhoods which meet a specified threshold of neighbors.

```
def KNN_classifier(train_df:pd.DataFrame,
                   train_labels:pd.Series,
                   test:bool = False,
                   test_df:pd.DataFrame=None,
                   config:list[int]=[3]) -> int:

    from sklearn.neighbors import KNeighborsClassifier
    knn_model = KNeighborsClassifier(n_neighbors=config[0], n_jobs=10)

    if test == False:
        xtrain, xtest, ytrain, ytest = train_test_split(train_df,
                                                        train_labels,
                                                        test_size=0.3)

        knn_model.fit(xtrain, ytrain) if isinstance(xtrain, np.ndarray)
                                      else knn_model.fit(xtrain.values, ytrain)
        ypred = knn_model.predict(xtest) if isinstance(xtest, np.ndarray)
                                         else knn_model.predict(xtest.values)
        acc = metrics.accuracy_score(y_true=ytest, y_pred=ypred)

        return "KNN", acc
    else:
        knn_model.fit(train_df.values, train_labels)
        ypred = knn_model.predict(test_df.values)
        pred_out = pd.DataFrame(ypred, columns=["class"])

        return "KNN", pred_out
```

The KNN algorithm therefore has one hyperparameter which is tuned, that is the n_neighbors. The n_neighbors hyperparameter is able to change the number of neighbors which must be checked before determining a

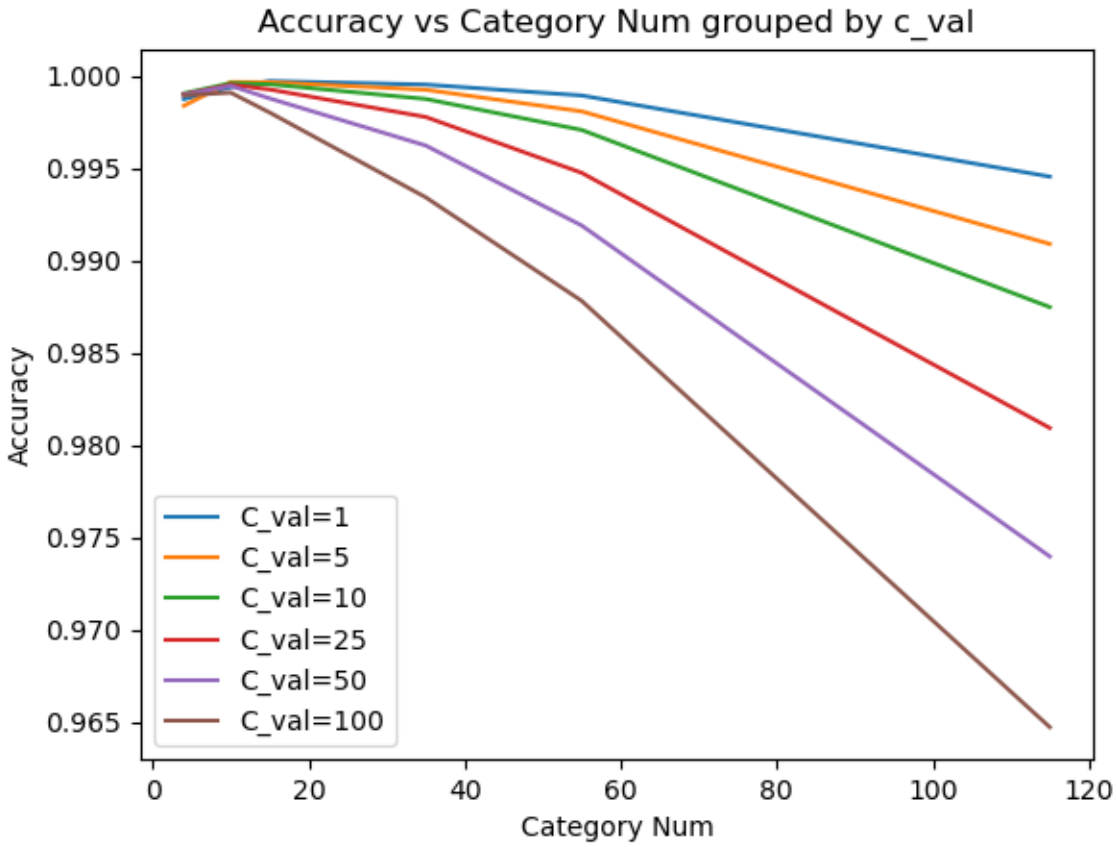
point is of a certain class. A lower neighbor count is prone to over fitting as it will only check one neighbor, whereas too many neighbors is prone to outliers and underfitting. Along with the K value the number of categories is also changed. The results of the tests are as seen in table 5 and 6.

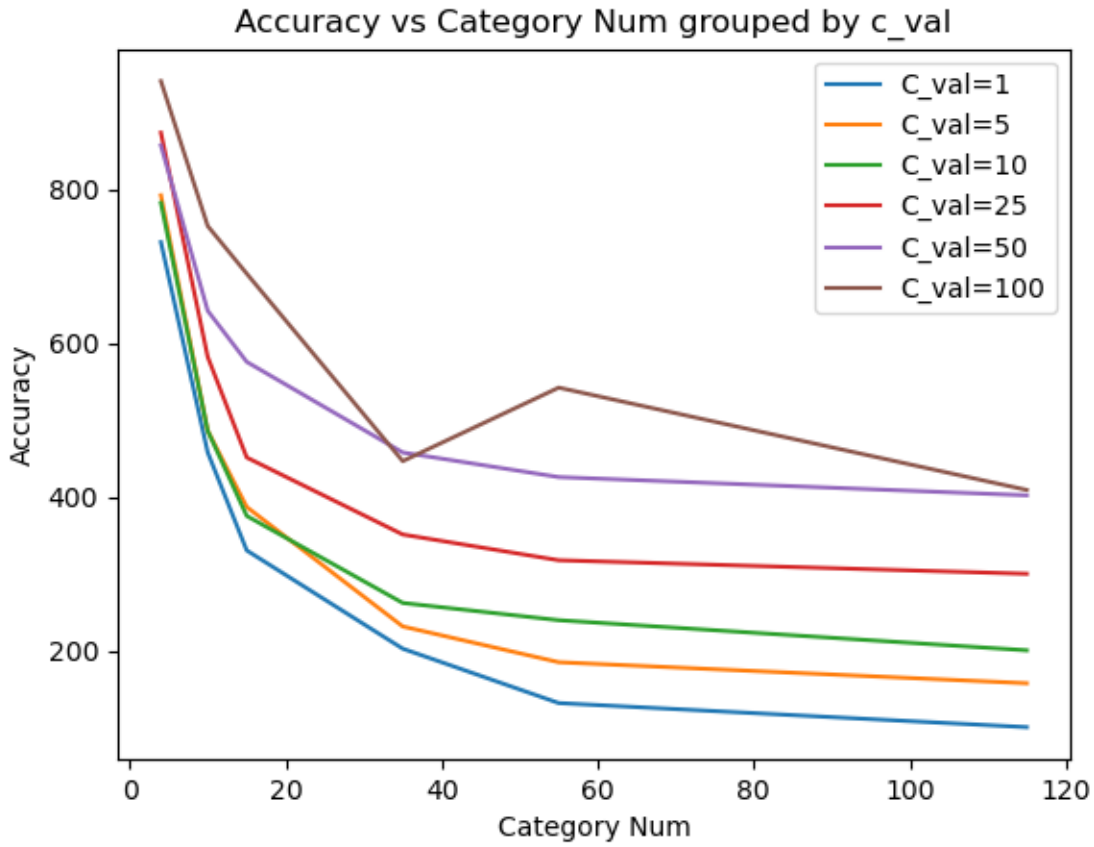
		Neighbor Count					
		1	5	10	25	50	100
Category Num	4	0.9987	0.9984	0.9990	0.9995	0.9990	0.9989
	10	0.9994	0.9996	0.9996	0.9992	0.9994	0.9991
	15	0.9997	0.9996	0.9995	0.9981	0.9988	0.9980
	35	0.9995	0.9992	0.9987	0.9978	0.9962	0.9934
	55	0.9989	0.9981	0.9971	0.9947	0.9919	0.9878
	115	0.9945	0.9909	0.9875	0.9809	0.9739	0.9647

Table 5: Accuracy given Category Number and Neighbor Count

		Neighbor Count					
		1	5	10	25	50	100
Category Num	4	731.796875	792.734375	782.90625	874.203125	857.859375	941.328125
	10	458.171875	486.515625	486.21875	582.484375	642.515625	752.765625
	15	330.703125	387.1875	375.578125	451.59375	576.015625	690.796875
	35	202.84375	231.78125	262.359375	351.40625	458.078125	446.828125
	55	132.09375	185.234375	239.890625	317.890625	426.09375	542.484375
	115	100.75	157.890625	200.71875	300.296875	402.390625	409.34375

Table 6: Runtime (s) values given Category Number and Neighbor Count


Accuracy vs Category Num grouped by c_val
1.000
0.995
0.990
0.985
0.980
0.975
0.970
0.965
Accuracy
0
20
40
60
80
100
120
Category Num
C_val=1
C_val=5
C_val=10
C_val=25
C_val=50
C_val=100



As seen in table 5, the accuracy of the KNN model seems to be fairly stable but minimally increases as the K increases from 1 to 25 and decreases past that by an incremental amount. It can also be seen that increasing the number of categories decreases the model's accuracy by almost 4%, but only when the number of neighbors is also high. This is a very interesting result as the number of categories does not significantly change the accuracy of the model for 1 neighbor but for every increase in neighbors the accuracy drops a little with a maximum of -4% for 100 neighbors. Having too many categories and too many neighbors may be confusing the model because it is underfitting what must be actually fit and taking into account points which should are not close but similar enough as the discretization become more and more precise. It also is not surprising that the model performs the best with a semi-large number of neighbors as 1 would most likely do well due to over fitting but a semi-large number would be able to fit the data well while also generalizing well for values which it has not yet seen.

The runtime in table 6 can be seen to be decreasing as the number of categories increase in the discretized variables. This is most likely due to the fact that the KNN is able to more easily distinguish classes with there being so many and there being less tie breakers, though this may not be the only reason as there is an over 600s second decrease in time from 4 categories to 115 categories. On the other hand the as the number of neighbors increase the runtime can be seen increasing. This is to be expected as, the quantity of neighbors the algorithm must check to classify a point increases the runtime will also increase.

Model Name	Accuracy	Model Runtime (s)	Neighbor Count	Category Numbers
KNN	0.9997	330.703125	1	15

From running the test it can be seen that though increasing the neighbor count decreased the accuracy by

a small margin, increasing the number of categories to 15 gave the highest accuracy, though most of the values resulted in a accuracy of over 99% with only a handful in the extremes being lower than that by 1-3%. Overall it can be said that for this data set in predicting mushroom toxicity, a KNN with 1 neighbor and 15 categories had the highest accuracy but most neighbor counts seemed to have very high accuracy's while most category numbers also had a high accuracy except for extremely high values like 55 and 115.

Decision Tree

A Decision Tree algorithm is a algorithm which uses the variable's information gain ratio to separate and make a tree out of the data and its variables. There are many controllable criterion which can be used to make the decision tree more optimal for each data set, as a Decision Tree is a fairly robust algorithm which can account for both numerical or continuous variables as well as categorical variables. There are many different approaches to using a decision tree where some applications prefer a shallow but wide tree, whereas some prefer a slim but deep tree. As Decision Tree algorithms are robust and can handle both categorical and numerical values it was chosen for this project, as utilizing algorithms solely made for categorical or numerical methods would leave a lot of information unused in the data set.

```
def Decision_tree_classifier(train_df:pd.DataFrame,
                             train_labels:pd.Series,
                             test:bool=False,
                             test_df:pd.DataFrame=None,
                             config:list[Union[str,int]]=["entropy", 1, 10, 100]) -> int:

    from sklearn.tree import DecisionTreeClassifier
    from sklearn.model_selection import cross_val_score

    Dtree = DecisionTreeClassifier(criterion=config[0],
                                   min_samples_leaf=config[1],
                                   min_samples_split=config[2],
                                   max_depth=config[3])

    if test == False:
        xtrain, xtest, ytrain, ytest = train_test_split(train_df,
                                                        train_labels,
                                                        test_size=0.3)

        Dtree.fit(xtrain, ytrain) if isinstance(xtrain, np.ndarray)
                                  else Dtree.fit(xtrain.values, ytrain)
        ypred = Dtree.predict(xtest) if isinstance(xtest, np.ndarray)
                                     else Dtree.predict(xtest.values)
        acc = metrics.accuracy_score(y_true=ytest, y_pred=ypred)

        return "DecisionTree", acc
    else:
        Dtree = Dtree.fit(train_df, train_labels)
        ypred = Dtree.predict(test_df)

        pred_df = pd.DataFrame(ypred, columns=["Label"])
```

```
    return pred_df
```

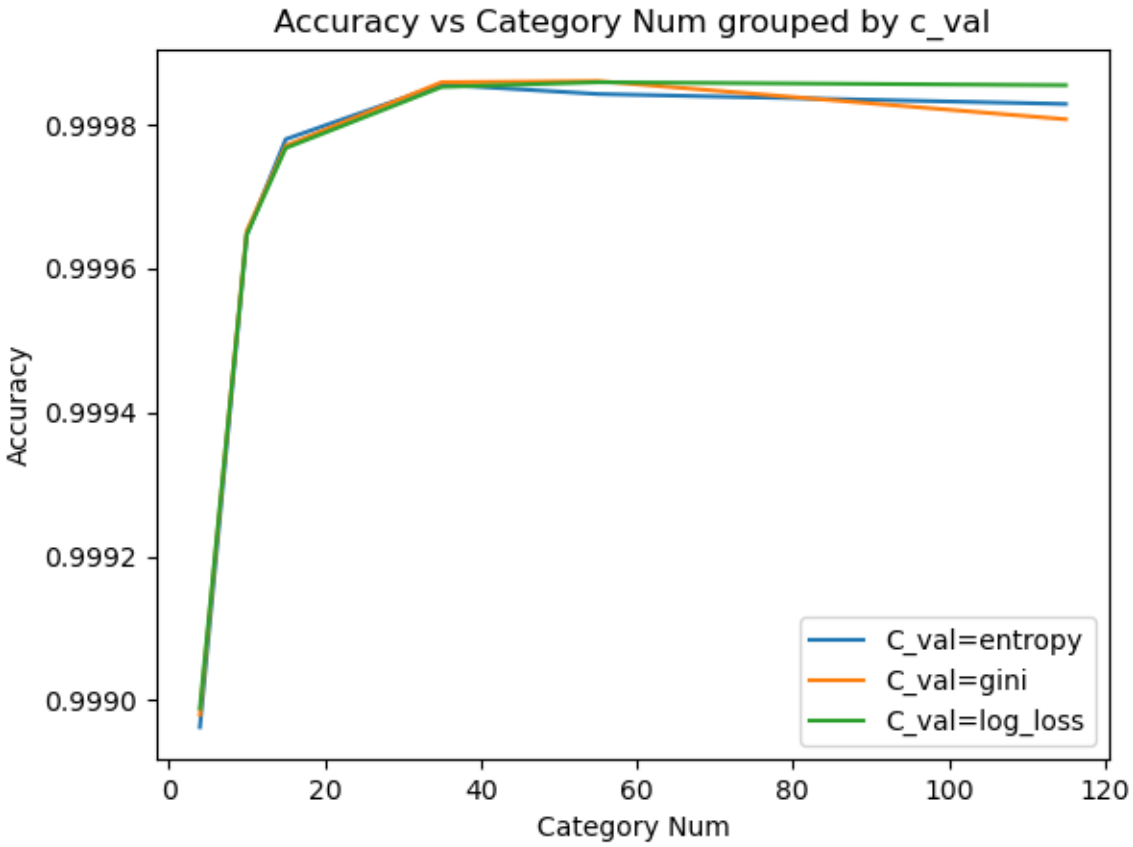
Of the many parameter involved when making a Decision Tree algorithm, 4 were mainly tuned. These were the loss_function, min_sample_leaf, min_sample_split, and max_depth. The loss_function parameter changed the equation used to calculate the information gained at each split. The min_sample_leaf parameter changed the minimum number of samples there had to be in a leaf, therefore not allowing trees where there are leafs with too little samples in the leaves. The min_sample_split parameter did a similar thing to min_sample_leaf where it changed the minimum samples that was needed to make a split, therefore not allowing splits where there were too small of a sample size on one side of the split, negating redundant splits. The last parameter max_depth changes the maximum depth the tree is allowed to go, negating any attempt to make a tree which fits every data point. Together these hyperparameters are able to control the model so it was not able to over or under fit the data.

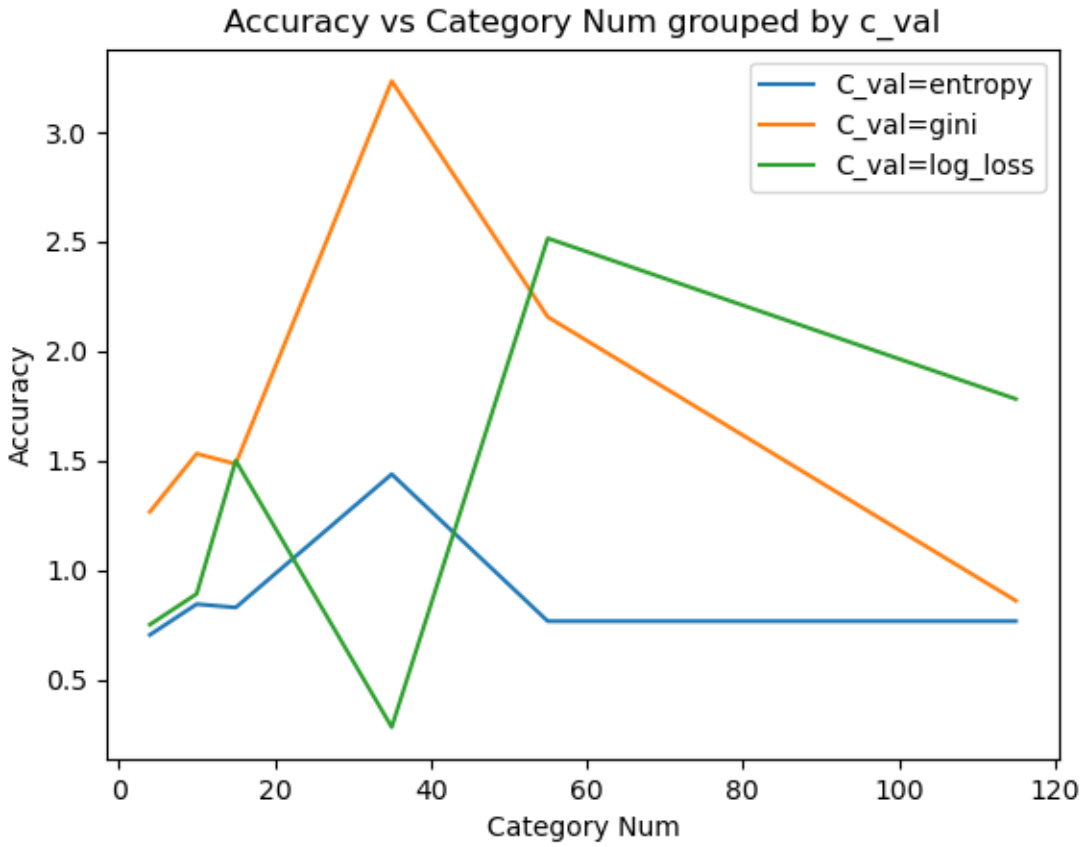
		Loss Function		
		log_loss	gini	entropy
Category Num	4	0.99899	0.99898	0.99896
	10	0.99965	0.99965	0.99965
	15	0.99977	0.99977	0.99978
	35	0.99985	0.99986	0.99986
	55	0.99986	0.99986	0.99984
	115	0.99986	0.99981	0.99983

Table 7: Accuracy given Category Number and Loss Function

		Loss Function		
		log_loss	gini	entropy
Category Num	4	0.75	1.265625	0.703125
	10	0.890625	1.53125	0.84375
	15	1.5	1.484375	0.828125
	35	0.28125	3.234375	1.4375
	55	2.515625	2.15625	0.765625
	115	1.78125	0.859375	0.765625

Table 8: Runtime (s) given Category Number and Loss Function


Accuracy vs Category Num grouped by c_val
Accuracy
0.9998
0.9996
0.9994
0.9992
0.9990
0
20
40
60
80
100
120
Category Num
C_val=entropy
C_val=gini
C_val=log_loss



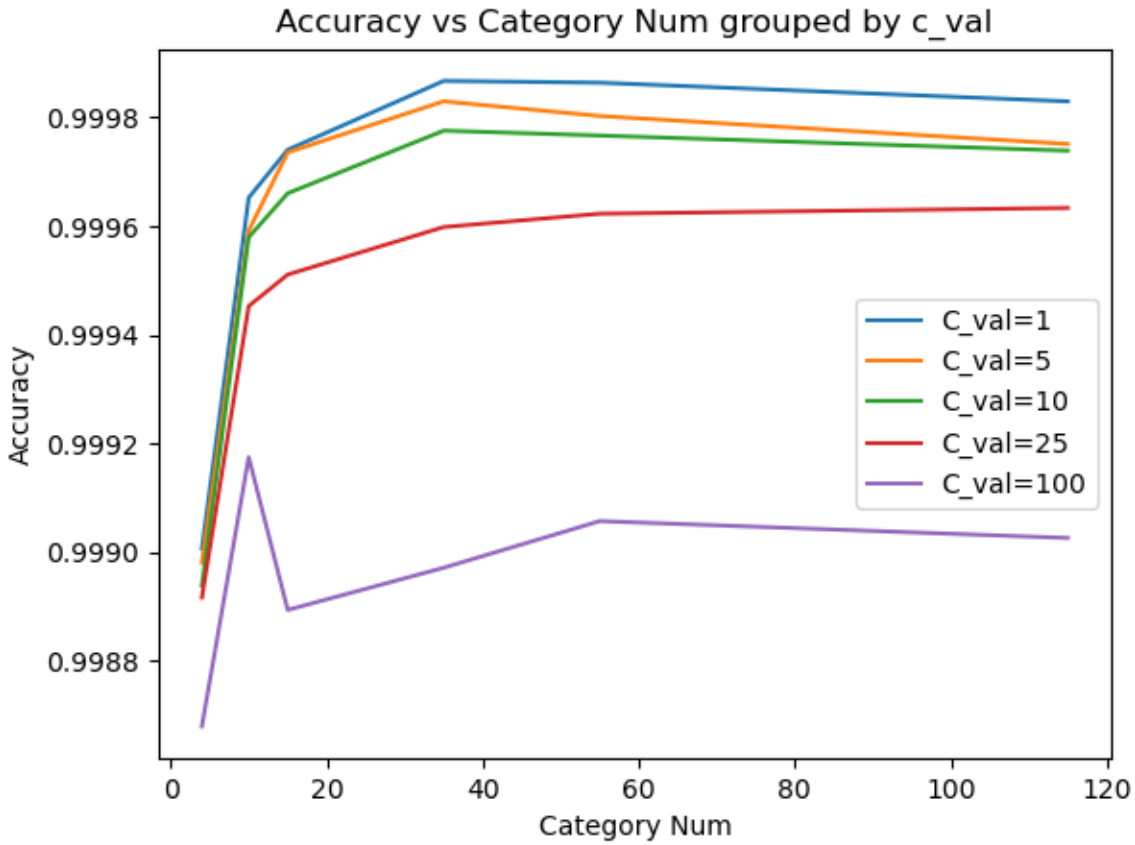
Looking at the results for the runtime and accuracy on table 7 and 8, with a loss function with a changing discretization category quantity, it can be seen that neither changing the loss function nor the categorical quantity significantly changed the accuracy of the model, with any change only equating to a accuracy change of less than 1%. There is a small trend of the accuracy being 0.005 higher for larger quantities of category number, for any type of loss function. The runtime is similar but can be seen to fluctuate randomly within within the range of its loss function. All in all the loss function seems to really only affect the runtime and not so much the accuracy.

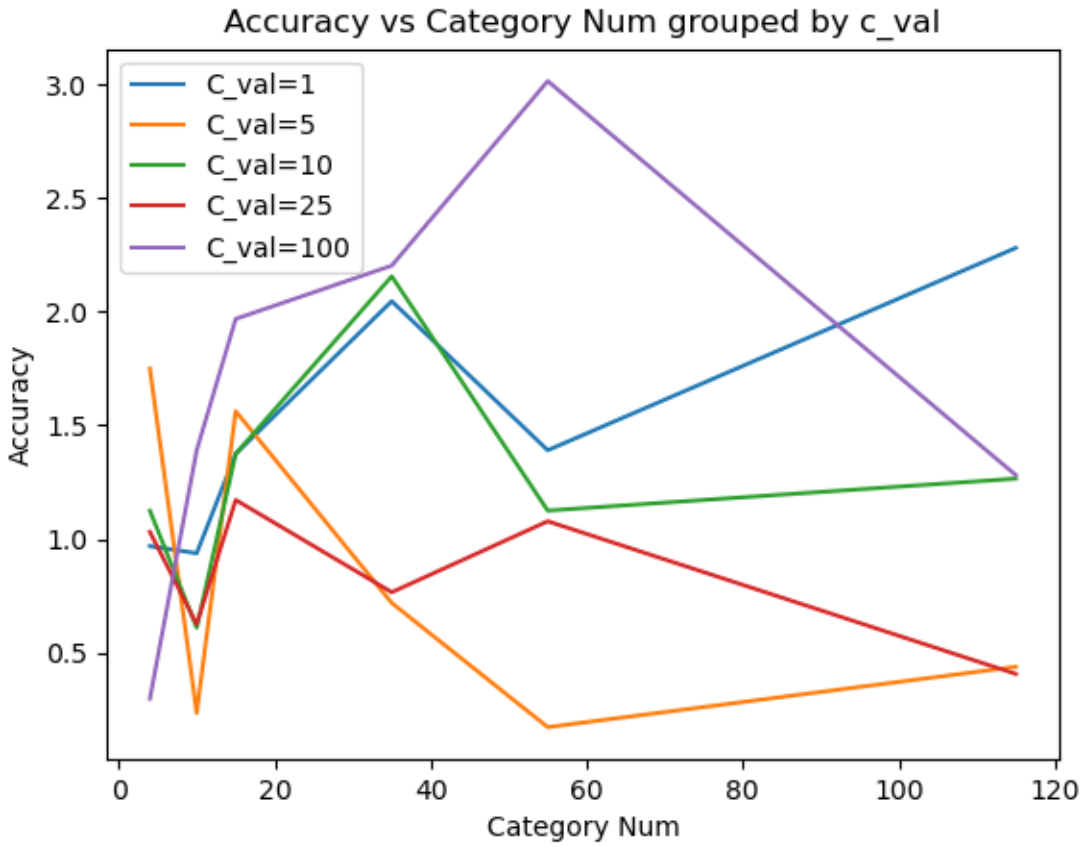
		Min Leaf Count				
		1	5	10	50	100
Category Num	4	0.999006	0.99898	0.998939	0.998916	0.998679
	10	0.999652	0.999592	0.99958	0.999453	0.999175
	15	0.999741	0.999736	0.999661	0.999511	0.998894
	35	0.999868	0.99983	0.999776	0.999598	0.998971
	55	0.999864	0.999803	0.999767	0.999623	0.999057
	115	0.99983	0.999751	0.999739	0.999634	0.999026

Table 9: Accuracy given Category Number and Min Leaf Count

		Min Leaf Count				
		1	5	10	25	100
Category Num	4	0.96875	1.75	1.125	1.03125	0.296875
	10	0.9375	0.234375	0.609375	0.625	1.390625
	15	1.375	1.5625	1.375	1.171875	1.96875
	35	2.046875	0.71875	2.15625	0.765625	2.203125
	55	1.390625	0.171875	1.125	1.078125	3.015625
	115	2.28125	0.4375	1.265625	0.40625	1.28125

Table 10: Runtime (s) given Category Number and Min Leaf Count





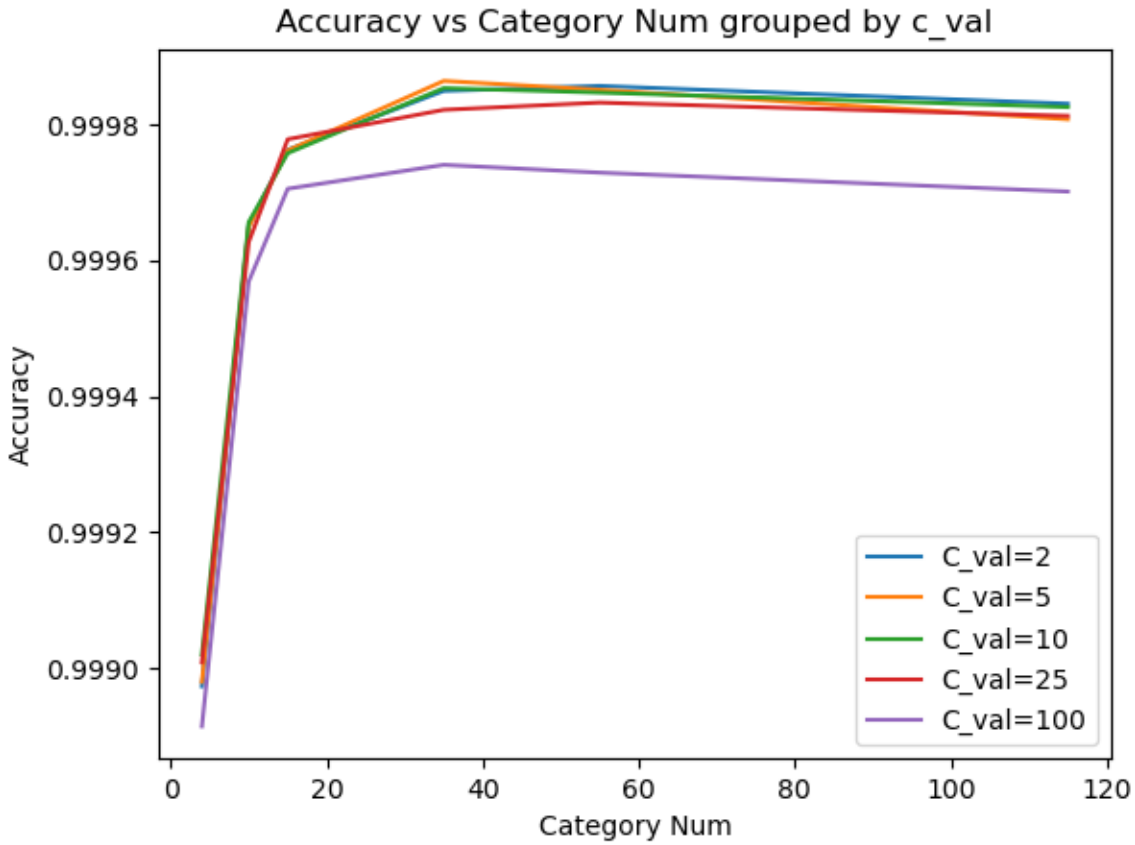
The Min Leaf Count variable also does not seem to benefit mush from being increased not from increased quantity of categorical numbers, as seen on table 9 and 10. As the model is already at 99.9% accuracy the changes seen in the accuracy with a change in Min Leaf Count and categorical quantity are very minimal with the largest increase being about 0.08%. These higher accuracy values are seen where there are higher values of categorical numbers with a low number of Min Leaf Count, with the highest being about 99.986%. This is somewhat expected as quantity of Min Leafs and the quantity of categories are not very correlated and as the model already achieves a high accuracy overall, it seems the benefits it can gain and the detriment it can weigh is also small from these two parameters.

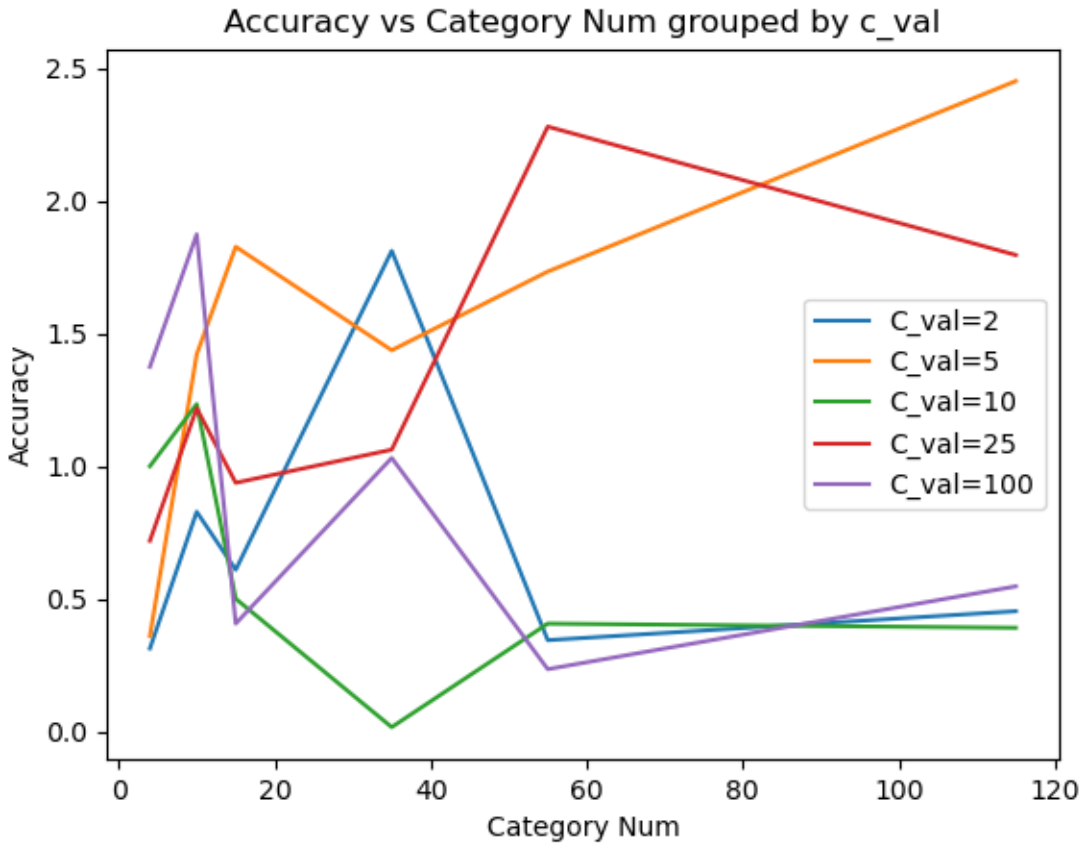
		Min Split Value				
		2	5	10	25	100
Category Num	4	0.998974	0.99898	0.99902	0.999009	0.998915
	10	0.999652	0.999649	0.999657	0.999627	0.999569
	15	0.999763	0.999762	0.999758	0.999778	0.999706
	35	0.99985	0.999864	0.999853	0.999822	0.999741
	55	0.999857	0.999851	0.999847	0.999832	0.99973
	115	0.999831	0.999808	0.999826	0.999813	0.999702

Table 11: Accuracy given Category Number and Min Split Value

		Min Split Value				
		1	5	10	50	100
Category Num	4	0.3125	0.359375	1.0	0.71875	1.375
	10	0.828125	1.421875	1.234375	1.21875	1.875
	15	0.609375	1.828125	0.5	0.9375	0.40625
	35	1.8125	1.4375	0.015625	1.0625	1.03125
	55	0.34375	1.734375	0.40625	2.28125	0.234375
	115	0.453125	2.453125	0.390625	1.796875	0.546875

Table 12: Runtime (s) given Category Number and Min Split Value





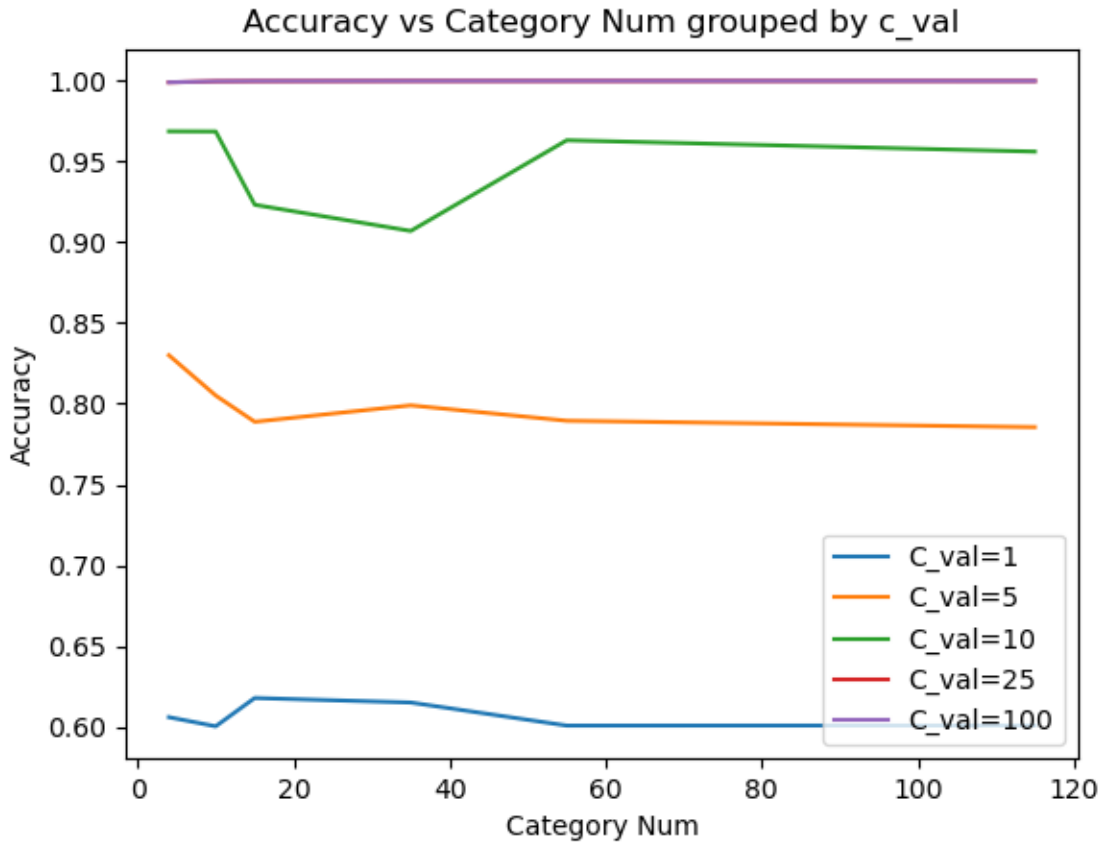
On table 11 and 12, the Min Split Value parameter can be seen to not really improve the accuracy of the model as it increases, there can be seen a trend of the model's accuracy increasing as the number of categorical values increase. This is the same trend as was seen on table 7,8 and 9,10 , but it is much less apparent in this table. This can be due to the fact that the model is predicting very accurately. This is interesting as the Min Split Value not making much of a difference while keeping a high accuracy suggests there are clear cut point where two large groups are separated fairly equally. If that is the case, it would suggest that there exists higher dimensional trends and clusters in the data which the models are able to see and grasp.

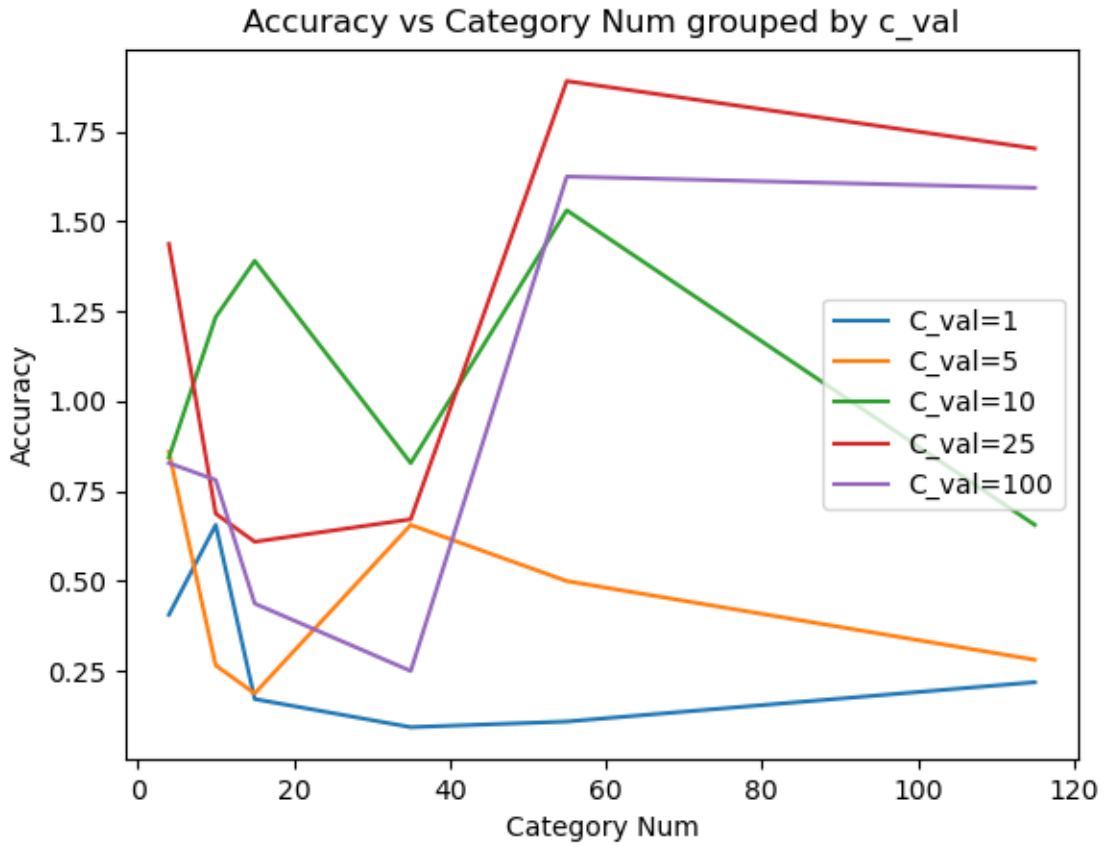
		Max Depth				
		1	5	10	50	100
Category Num	4	0.605955	0.830042	0.968662	0.998977	0.999012
	10	0.600309	0.805103	0.968546	0.999648	0.999657
	15	0.617831	0.788837	0.923183	0.999748	0.999791
	35	0.615035	0.799002	0.906976	0.999843	0.999852
	55	0.600757	0.789475	0.963128	0.999851	0.999872
	115	0.600811	0.785512	0.956113	0.999836	0.999847

Table 13: Accuracy given Category Number and Max Depth

		Max Depth				
		1	5	10	25	100
Category Num	4	0.40625	0.859375	0.84375	1.4375	0.828125
	10	0.65625	0.265625	1.234375	0.6875	0.78125
	15	0.171875	0.1875	1.390625	0.609375	0.4375
	35	0.09375	0.65625	0.828125	0.671875	0.25
	55	0.109375	0.5	1.53125	1.890625	1.625
	115	0.21875	0.28125	0.65625	1.703125	1.59375

Table 14: Runtime (s) given Category Number and Max Depth





The last hyperparemeter tuned was the Max Depth. Unlike the other parameters, it can be easily seen that there is a large correlation between having a deep tree and having a high accuracy, as the deeper the tree was allowed to go the higher the accuracy became. This increasing trend continued until the model was given a max depth of 50, and at a max depth of 100 the model was not able to continue improving at the same rate as previously. This is to be expected as half of the predictive power a Decision Tree comes from its depth, with the other half coming from its width. The trend of there being marginal increases in accuracy with higher category numbers also continues in in this test as even at a max depth of 100 there exists some, albeit very small increases in accuracy of the model when compared to lower quantities of category numbers.

Model Name	Accuracy	Model Runtime (s)	Min Leaf	Min Split	Max Depth	Category Numbers
Decision Tree	0.9999	0.5-2.5	1	5	100	115

Random Forest

The Random Forest classifier is ensemble of Decision Tree algorithms, aimed to utilize their robustness, efficiency, and their accuracy. By having a large quantity of trees which model the data in differing ways and then aggregating their results into one, allows for an even more robust model which can handle categorical and numerical values but also handle fitting very complex problems. As the Random Forest classifier is an ensemble of Decision Trees it was natural to utilize this in the project for mushroom toxicity as a more robust and complex algorithm to model the data.

```
def Random_Forest_classifier(train_df:pd.DataFrame,
                             train_labels:pd.Series,
                             test:bool = False,
                             test_df:pd.DataFrame=None,
                             config:list[int]=[100, 100]) -> int:
    from sklearn.ensemble import RandomForestClassifier
    rf_model = RandomForestClassifier(n_estimators=config[0],
                                      max_depth=config[1],
                                      n_jobs=10)

    if test == False:
        xtrain, xtest, ytrain, ytest = train_test_split(train_df,
                                                        train_labels,
                                                        test_size=0.3)

        rf_model.fit(xtrain, ytrain)
        ypred = rf_model.predict(xtest)
        acc = metrics.accuracy_score(y_true=ytest, y_pred=ypred)

        return "RandomForest", acc
    else:
        rf_model.fit(train_df, train_labels)
        ypred = rf_model.predict(test_df)
        pred_out = pd.DataFrame(ypred, columns=["class"])

        return "RandomForest",pred_out
```

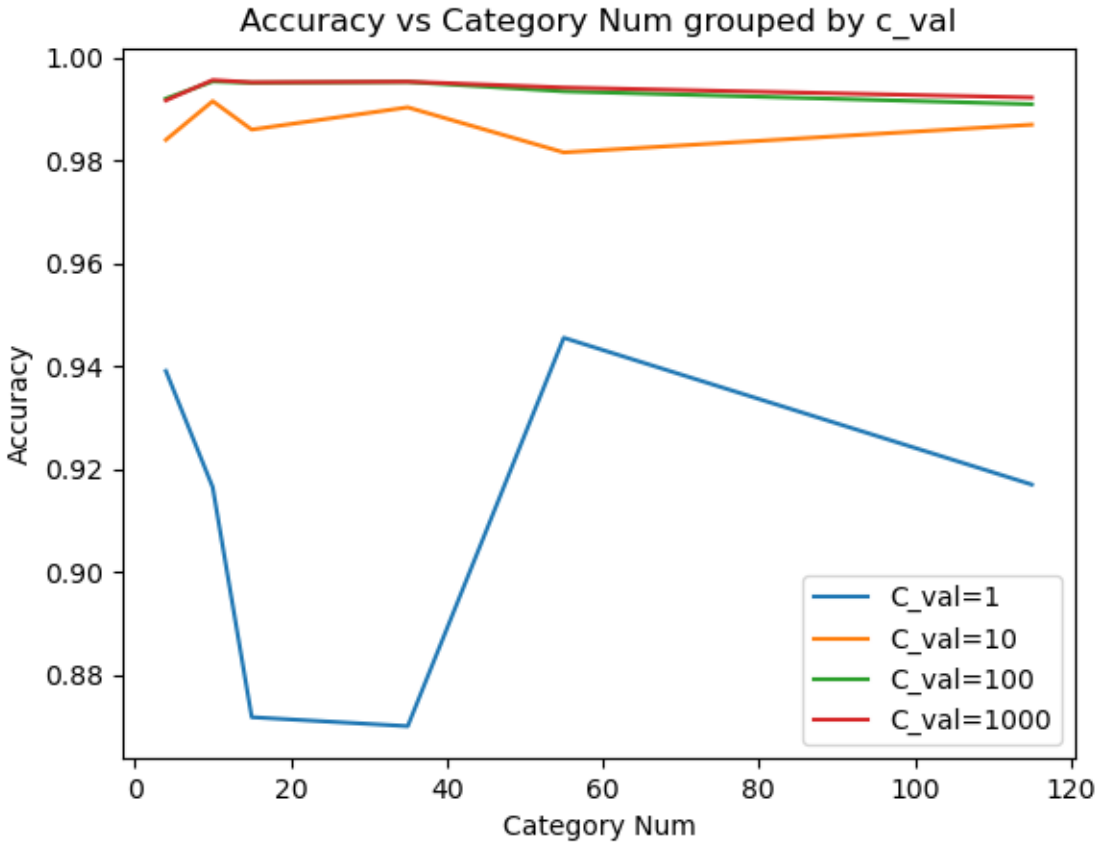
The Random Forest classifier has 2 main hyperparemeters which are tuned. These are the n_estimators and max_depth. The n_estimators is the unique hyperparameter for Random Forest, controlling the number of Decision Trees which are made to model the data. This is very important can greatly impact the performance of the model, as there being more Decision Trees allows the model to better fit the data but also generalize to the data so it is not over fit nor under fit. The max_depth hyperparameter like the parameter with the same name for Decision Trees, controls the depth which the tree is allowed to go to, the only difference being this parameter is the maximum depth for all of the Decision Trees in the forest.

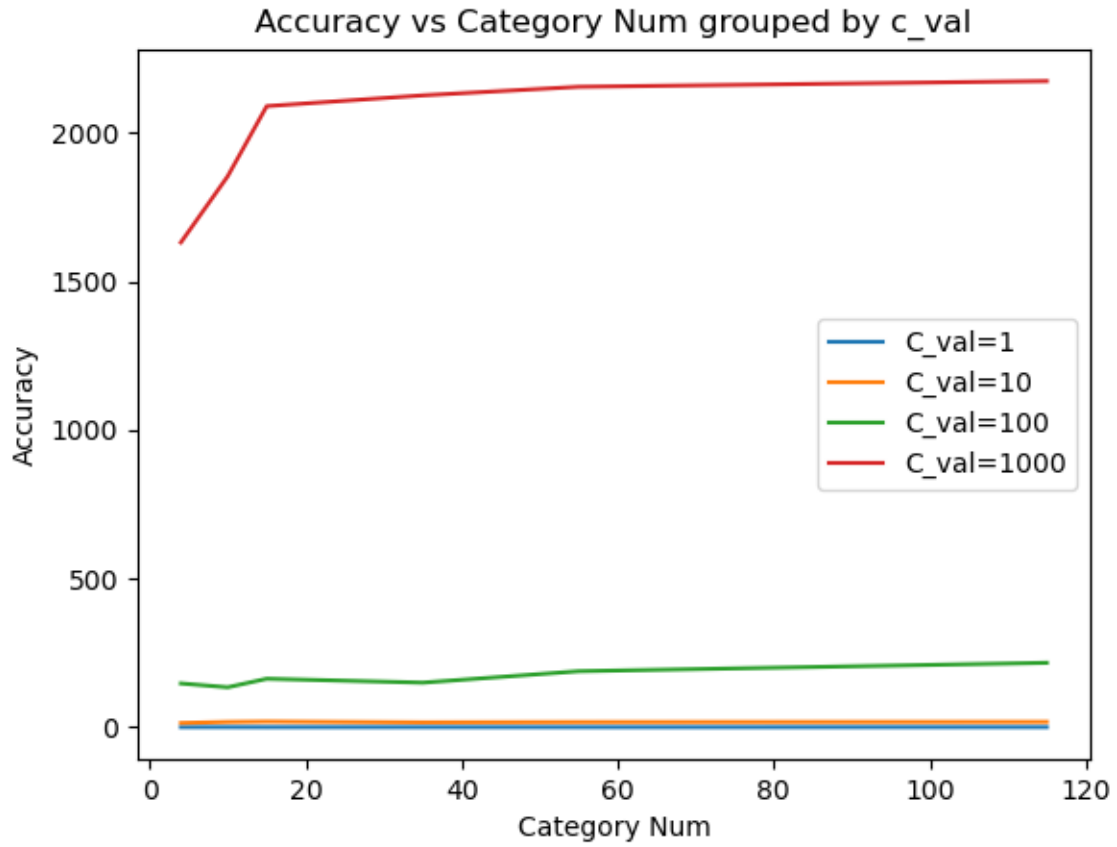
		Estimator Count			
		1	10	100	1000
Category Num	4	0.9391	0.9840	0.9921	0.9917
	10	0.9165	0.9916	0.9954	0.9957
	15	0.8719	0.9860	0.9951	0.9952
	35	0.8701	0.9904	0.9953	0.9953
	55	0.9456	0.9816	0.9935	0.9942
	115	0.9171	0.9870	0.9910	0.9923

Table 15: Accuracy values given Category Num and Estimators

		Estimator Count			
		1	10	100	1000
Category Num	4	0.453125	14.21875	147.4375	1631.75
	10	0.421875	17.796875	134.453125	1854.375
	15	0.265625	19.09375	163.171875	2090.34375
	35	0.40625	16.421875	150.0625	2126.546875
	55	0.28125	17.390625	188.5	2155.40625
	115	0.40625	18.0625	216.84375	2174.75

Table 16: Runtime values given Category Num and Estimators





On tables 15 and 16, the affects of changing the number of estimators and the number of categories can be seen. As it can be seen the number of estimators increases the accuracy greatly, boosting it from a lower of 87% to a high of 99.5%. This is to be expected as the estimator count essentially increases the number of Decision Trees which are being used to model the data, and from the previous tests for the Decision Tree it was seen that a single Decision Tree was able to model the data up to 99%, therefore having more would only allow the model to even better fit the data and further increase the generalization of the model. There can also be seen a interesting trend for the number of categories affecting the accuracy. At the beginning with 1 estimator having 4 categories or 55 are preferred, producing the highest accuracy, but as the estimator count increases it can be seen that having a category value between 10 and 35 is preferred, those values having higher accuracy than 4 and 55 which had the highest accuracy for a estimator count of 1.

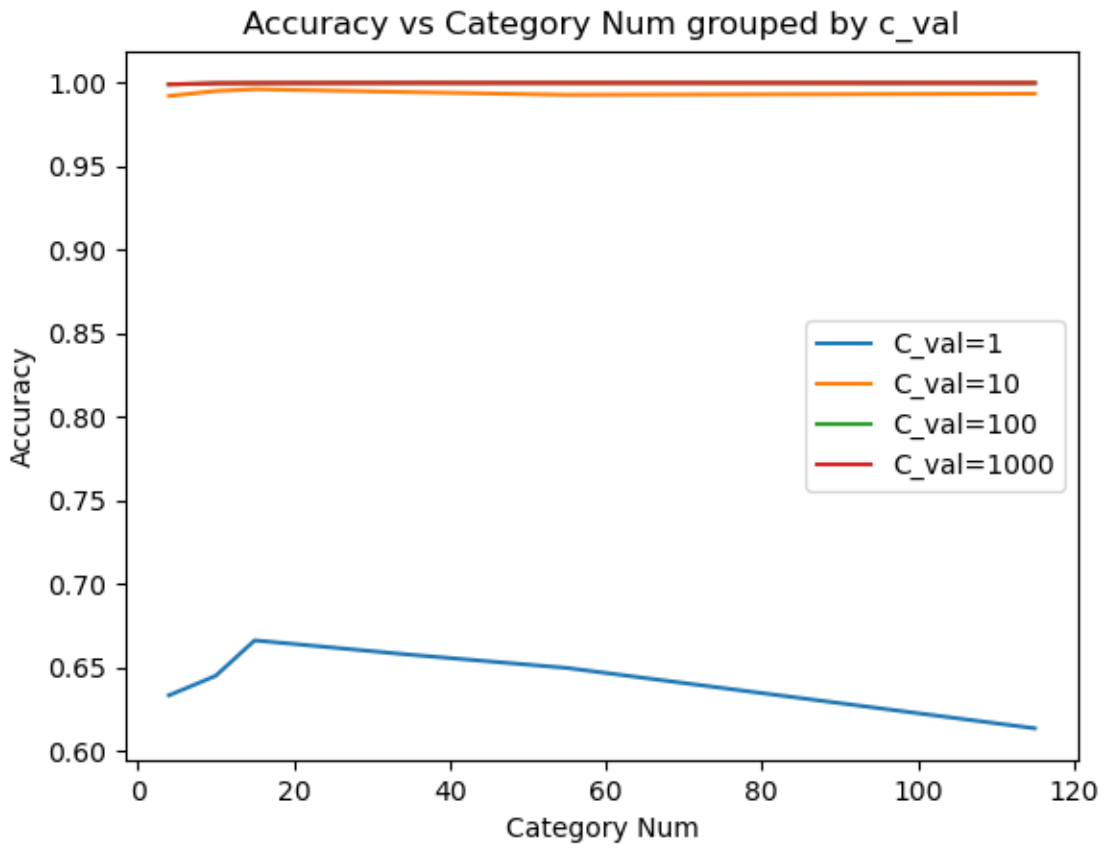
Understandably, the runtime of the model greatly increases at the number of estimators or Decision Trees are increased. With 1 estimator having the same time as 1 Decision Tree, and 10 having about 25 times that, 100 having about 10 times the previous runtime, and 1000 having a little under 15 times that. It is very interesting to see that the runtime is not linearly correlated to the number of estimators and actually is increasing slightly faster than the number of estimators, suggesting either a extra complexity introduced with higher numbers of estimators or decreased efficiency in processing and memory allocation for higher estimator counts.

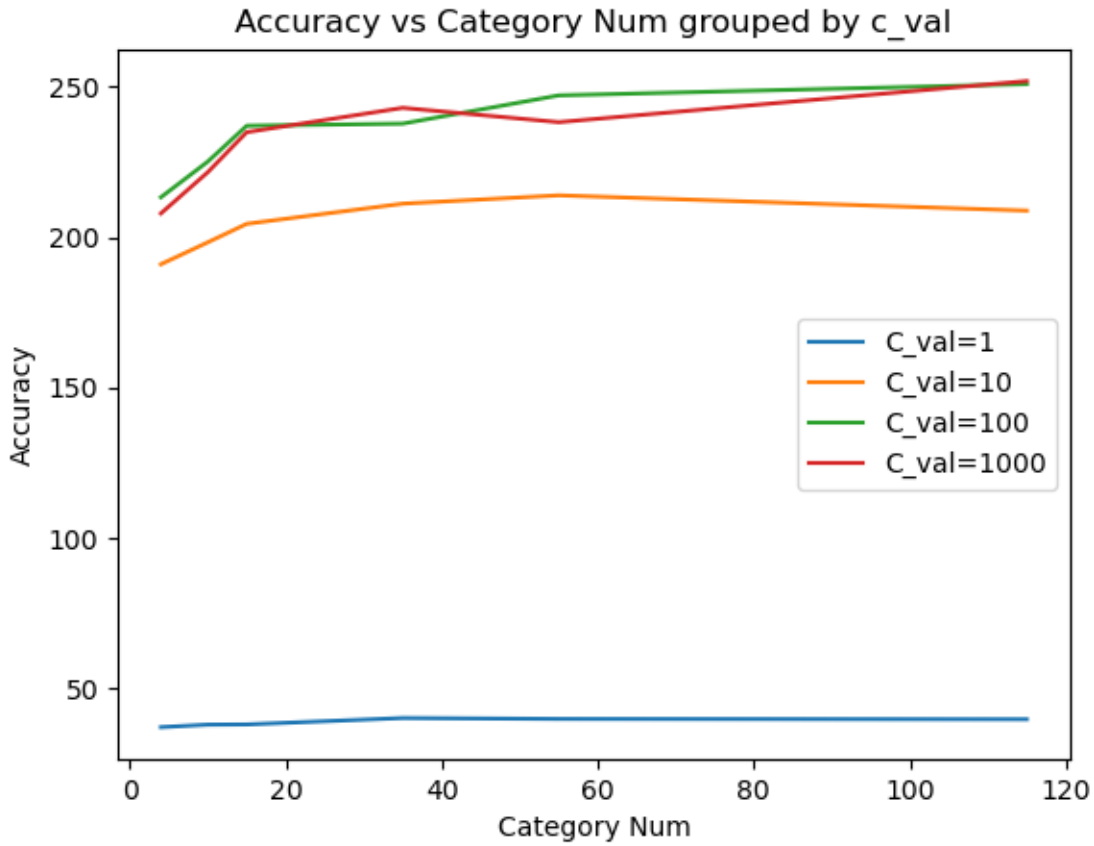
		Max Depth			
		1	5	10	50
Category Num	4	0.6335	0.9921	0.9990	0.9990
	10	0.6452	0.9951	0.9997	0.9997
	15	0.6664	0.9961	0.9998	0.9998
	35	0.6578	0.9944	0.9999	0.9999
	55	0.6499	0.9928	0.9999	0.9999
	115	0.6138	0.9934	0.9999	0.9999

Table 17: Runtime values given Category Num and Max Depth

		Max Depth			
		1	10	100	1000
Category Num	4	37.125	191.03125	213.296875	207.9375
	10	37.953125	198.3125	225.140625	221.609375
	15	38.03125	204.46875	237.078125	234.9375
	35	40.15625	211.171875	237.78125	243.03125
	55	39.859375	213.9375	247.171875	238.265625
	115	39.75	208.859375	250.921875	251.9375

Table 18: Runtime (s) values given Category Num and Max Depth





On tables 17 and 18 the accuracy and runtimes of Max Depth can be seen. Like Decision Trees it can be seen that the max depth greatly affects the accuracy of the Random Forest model. This is to be expected as the like stated in the Decision Tree section, the depth is about half of the predictive power a Decision Tree has, being able to model more complex patterns and problems the deeper it is. Therefore seeing such a large differences in accuracy changing the depth of the model is not surprising. What is somewhat surprising is the accuracy of the model being higher when there are lower category numbers when there is a smaller max depth but having a higher accuracy with larger category numbers at deeper depths. This is interesting as it suggests that when making a very shallow and wide tree, it is better to have much less categorical values than more, effectively making the tree less wide as well.

Understandably, the runtime for deeper models are longer, as the model must calculate more node per tree, which results in a much higher runtime.

Model Name	Accuracy	Model Runtime (s)	N-Estimators	Max Depth	Category Numbers
Random Forest	0.9999	100-200	100	50	115

Overall, the performance of the Random Forest classifier was similar to the results of the Decision Tree, where a deeper and larger quantity of categorical values, along with 100 estimators had the highest accuracy. That is not very surprising as having 100 Decision Trees with 50 max depth and 115 categorical values is definitely sufficient to have a high accuracy considering one Decision Tree with 100 depth and 115 categories got the same result.

Results

Naive Bayes

The Naive Bayes algorithm was able to achieve an accuracy of 0.6339 or 63.39%. Though not a bad accuracy compared to the later more complex algorithms which were used the accuracy was definitely amongst the worst. This might be due to the fact that the variables used were either not independent and had many strong dependent connections which wasn't able to be modeled by the algorithm.

K-Means

The highest accuracy the K-Means algorithm achieved was 0.4656 or 46.56%. This was the lowest accuracy amongst all of the other models. This might be due to the fact that the data were not spatially differentiable and that the patterns which dictated whether a mushroom was poisonous or edible were found else where, therefore a model which focuses on the distance between values and clusters then according to their distance from other points and clusters may not have done as well.

KNN

The highest accuracy achieved by the KNN algorithm was 0.9997 or 99.97%. This was one of the highest accuracy achieved among all of the models. This might be due to why the K-Means was not able to model well. With the toxicity of a mushroom being more about its relative connections to other points and less about the distance of all of its variables.

Decision Tree

The highest accuracy achieved by the Decision Tree algorithm was 0.9999 or 99.99%. This is the highest accuracy achieved among all of the models. This might be due to the fact that the variables were able to be easily modeled with concrete splitting points and because there were splitting decisions which had very high information ratios.

Random Forest

The highest accuracy achieved by the Random Forest algorithm was 0.9999 or 99.99%. This is also the same accuracy achieved by the Decision Tree model, which is to be expected, as the Random Forest utilized many Decision Trees to predict values. Like the Decision Tree this high accuracy might have been possible due to there being a distinct high information gain split which was able to be conducted in the trees which allowed for easier classification of poisonous or not.

Conclusions

Through the multitudes of testing conducted for the many models used in the project the highest accuracy was achieved by Decision Trees and Random Forest algorithms. Followed very closely by KNNs and then Naive Bayes and K-Means. This result was not very surprising as the 3 models which did very well were all very robust and strong algorithms which were predicted to be able to model this problem. It was somewhat disappointing to see that both Naive Bayes and K-Means were not able to classify the toxicity of a mushroom very well. But as with many applications of Naive Bayes, more Naive Bayes specific data augmentations could be done to further increase the accuracy of the model. For K-Means it is not known but there also must be some more data augmentations to help the data align to the strengths of the model more. Perhaps doing those augmentations will help more complex tasks, where Decision Trees, Random Forests, and KNNs are not able to perform as well.

The purpose of this project was to first create a model which can confidently detect if a mushroom was poisonous or not. To inform people of a mushroom's toxicity and become the first step of many into identifying whether a mushroom was poisonous or edible. The second but more futuristic and larger goal was to become the first stepping stone into creating a more robust model which could identify not just the toxicity of a mushroom but the specie name of one. The general idea of this project was to use characteristics which were common amongst many mushrooms and see if they were able to be used to predict something about a mushroom. In this case the toxicity, but in future research the hope is to used what was learned in this project to create one which could use the same or more similar characteristics of a mushroom to classify its specie name. Having achieved such a high accuracy with Decision Trees and Random Forest algorithms it shows further promise to being able to model mushroom specie names. As in theory the bulk of the processing power of the Random Forest algorithm were not used as one Decision Tree was found to be sufficient in modeling the toxicity of a mushroom.

All in all, the project is deemed a great success with a astoundingly high accuracy for predicting mushroom toxicity. It was also very informative with great new information on how these models works and how well they work for this type of mixed categorical and numerical data. With a new foundation on modeling mushroom's based on their characteristics, further development may be done to allow for fully confident prediction of the toxicity of a mushroom. This may be seen as a addition to the models made in this project along with others which will help ensure the correctness of the output as well as perhaps having the prediction being verified by another model or an expert which can detect species. This would allow for even better confidence in the toxicity of a mushroom as knowing the species would allow for user double checking of the output. As research and interest in the field of organism identification increases, the need for human time and energy will decrease allowing for not only faster processing of new species but also a wider more connected and more information dense ecosystem. Not only that, there will be more time and energy allocated for research, which would even further push the development of new technologies and finding in the field of biology and organism taxonomy.