

**PRAGUE UNIVERSITY OF ECONOMICS AND BUSINESS**

Faculty of Informatics and Statistics

**DATA-X PROJECT:  
DETERMINING THE EDIBILITY OF  
MUSHROOMS USING DATA SCIENCE AND  
MACHINE LEARNING METHODS**

Project within the course Data-X – Applied Data Analytics  
Models in Real World Tasks (4IT439)

# Contents

1	Problem Definition.....	3
1.1	Methodology of Work.....	3
2	Development Environment Characteristics.....	5
2.1	Environment creation and installation .....	6
2.2	Importing src_PN module .....	6
2.3	Loading fitted models and objects .....	7
3	Data Understanding.....	8
3.1	Data Import .....	8
3.2	Dataset.....	8
3.3	Initial Categorization.....	8
3.4	Duplicates and Missing Values.....	8
3.5	Data Exploration .....	8
3.5.1	Target and Feature Distribution .....	9
3.5.2	Dependency Analysis .....	11
3.6	General Preprocessing.....	13
3.7	Data Split.....	13
3.8	Binning of Variables .....	13
3.9	WoE Transformation.....	14
4	Modelling .....	16
4.1	Selection of Methods.....	16
4.1.1	Logistic Regression .....	16
4.1.2	Decision Trees .....	17
4.1.3	Random Forest .....	18

4.1.4	Gradient Boosting Classifier .....	18
4.2	Feature Selection.....	19
4.2.1	Bayesian Optimization .....	20
4.2.2	Recursive Feature Elimination .....	23
4.3	Hyperparameter Tuning & Final Model Selection .....	25
4.4	Building the Final Model .....	27
5	Evaluation.....	28
5.1	Confusion Matrix Measures.....	28
5.2	AUROC.....	29
5.3	KS Statistics .....	30
5.4	Brier Score .....	30
5.5	Evaluation Metrics of the Final Model .....	31
5.6	Limitations .....	34
	Sources .....	35
	List of Figures and Tables .....	38
	Appendix .....	39

# 1 Problem Definition

The goal of this project is to evaluate edibility of mushrooms using various data science and machine learning methods.

We have been provided with a mushroom dataset, based on which we have to create a model able to predict whether a particular mushroom is edible or not.

Tasks of this project include data understanding and exploration, data pre-processing with feature engineering, feature selection, building a model including hyperparameter tuning, and subsequent evaluation using various metrics.

The main observed metrics of model success rate are F-Score and AUC and we are looking for the model with the highest value of them.

## 1.1 Methodology of Work

The methodology we follow in our work is called CRISP-DM (Cross-Industry Standard Process for Data Mining). It aims to provide a universal approach to model the process of data mining. The life cycle of this process consists of 6 steps: business understanding, data understanding, data preparation, modelling, evaluation, and deployment (Provost & Fawcett, 2013).

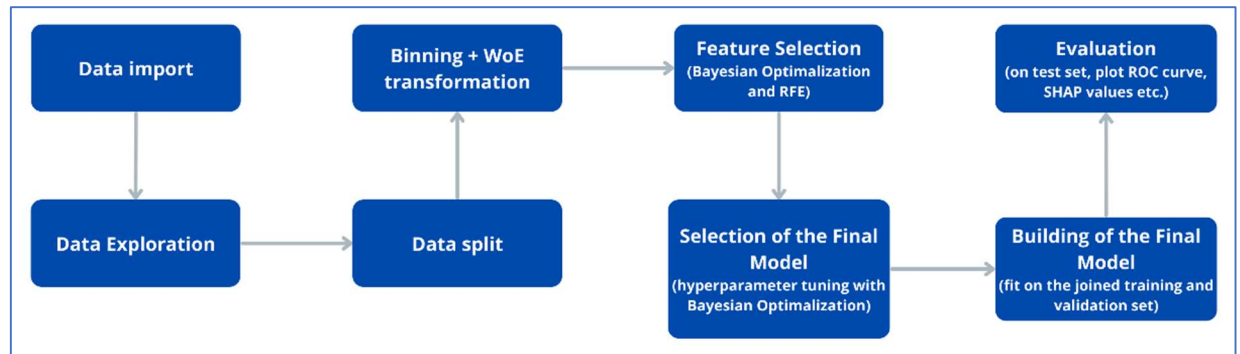
The following flowchart depicts our workflow. First, we import all our relevant data, i.e., raw data with the category full names. Afterward, we inspect the data within data understanding and exploration – such tasks include data description, visualization of variables’ distributions, and dependency analysis using Chi-squared hypothesis testing. Using information from data understanding, for instance omitting duplicates and erroneous values, we then move to the data partitions as we split our data into training, validation, and test set. Then, a data pre-processing follows with a binning of features’ categories which are subsequently transformed using Weight-of-Evidence Encoding.

Once we have our data transformed, we perform an iterative process of feature selection – each default model is tuned with Bayesian optimization and then used within the Recursive Feature Elimination (RFE) to select a subset of optimal features. Afterward, we perform a selection of the final model within hyperparameter tuning, where each model is tuned on each subset of features selected within the feature

selection and then evaluated on the validation set. Then, we choose the final model as the one with the best scores on the validation set.

Once we select our final model, we then build such model by fitting it on the joined training and validation set, and finally, we evaluate the given model on the test set.

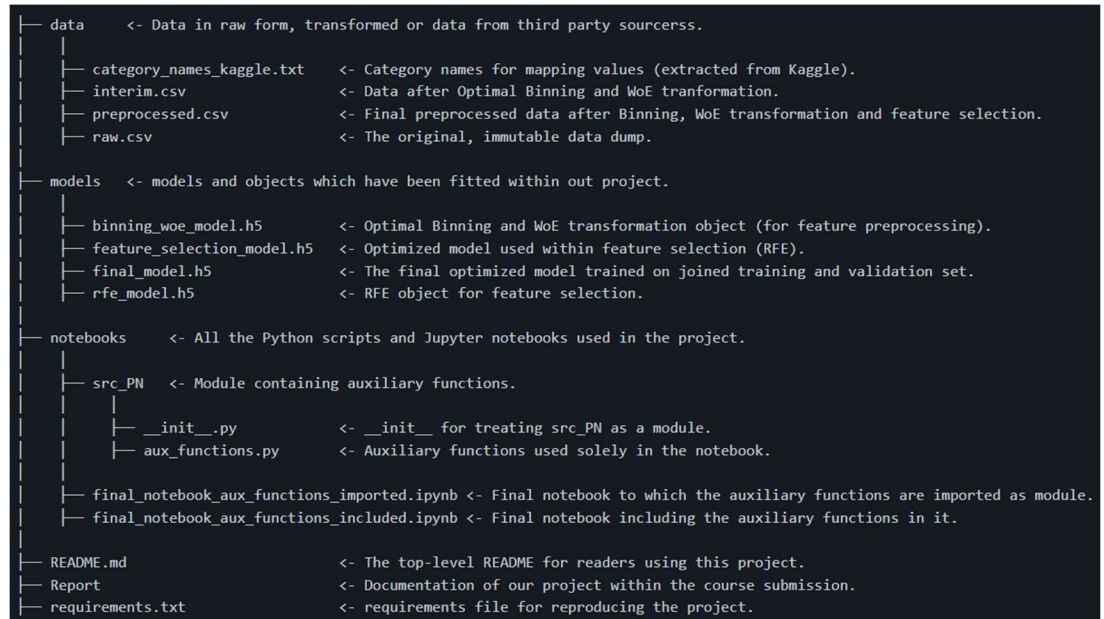
*Figure 1: Project workflow*



## 2 Development Environment Characteristics

We collaborated using the platform GitHub where we structured the shared files according to following scheme suggested by lecturers, which were adjusted with accordance to our tasks:

Figure 2: Scheme of files



```
├── data      <- Data in raw form, transformed or data from third party sources.
│   ├── category_names_kaggle.txt  <- Category names for mapping values (extracted from Kaggle).
│   ├── interim.csv               <- Data after Optimal Binning and WoE transformation.
│   ├── preprocessed.csv          <- Final preprocessed data after Binning, WoE transformation and feature selection.
│   └── raw.csv                   <- The original, immutable data dump.
├── models   <- models and objects which have been fitted within out project.
│   ├── binning_woe_model.h5      <- Optimal Binning and WoE transformation object (for feature preprocessing).
│   ├── feature_selection_model.h5 <- Optimized model used within feature selection (RFE).
│   ├── final_model.h5           <- The final optimized model trained on joined training and validation set.
│   └── rfe_model.h5             <- RFE object for feature selection.
├── notebooks <- All the Python scripts and Jupyter notebooks used in the project.
│   ├── src_PN                   <- Module containing auxiliary functions.
│   │   ├── __init__.py          <- __init__ for treating src_PN as a module.
│   │   └── aux_functions.py      <- Auxiliary functions used solely in the notebook.
│   ├── final_notebook_aux_functions_imported.ipynb <- Final notebook to which the auxiliary functions are imported as module.
│   └── final_notebook_aux_functions_included.ipynb <- Final notebook including the auxiliary functions in it.
├── README.md                    <- The top-level README for readers using this project.
├── Report                       <- Documentation of our project within the course submission.
└── requirements.txt              <- requirements file for reproducing the project.
```

- *data* includes an input (raw) and pre-processed datasets and other data third-party sources (particularly from Kaggle).
- *Report* is this PDF file in which the project's documentation is written.
- *models* are mainly outputs from the Python code. They include not only models for predictions but also objects for feature selection or feature transformation.
- *notebooks* contain the code in Python – they include our created module *src\_PN* containing auxiliary functions which are used solely in the notebooks. They also contain two notebooks in which the main part of the project is conducted. The first one uses auxiliary functions by importing the module *src\_PN*, and the second one includes such functions which are directly defined in it (therefore, no importing of the module is needed).
- *requirements.txt* is given here so that all collaborators employ the correct packages for Python development. Using the command *pip freeze* we obtained a superset of the required packages from which the most

important packages for our modelling case are pandas, numpy, matplotlib, pickle, optbinning, scikit-learn, scikit-optimize, scipy, seaborn and shap.

- Last but not least, another important file is *README.md* which summarizes this project, including detailed requirements pertaining to this project.

## 2.1 Environment creation and installation

Before running any scripts or notebooks, we recommend first creating your own environment with Python version 3.9.13, in which our module and models have been created. You can create it Anaconda terminal using this command:

```
conda create -n yourenv python=3.9.13
```

Once you have created your own environment, you should activate your environment before installing required packages (*requirements.txt*). You can again do it in the Anaconda terminal using command:

```
conda activate yourenv
```

Once you have environment activated, then you can install the requirements using command:

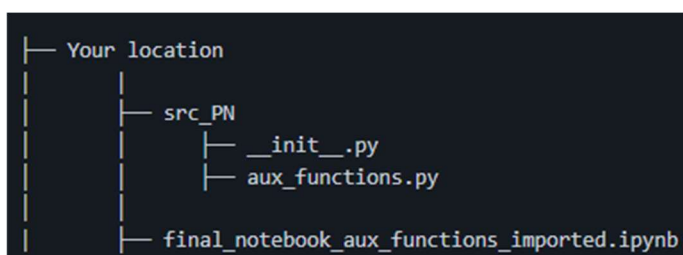
```
pip install -r requirements.txt
```

- Noted, if you do not change the directory in the terminal (using *cd* command), you should also add to it the path where the file is located).

## 2.2 Importing *src\_PN* module

The last thing before running the notebooks, you should also import the *src\_PN* module – this folder contains the auxiliary functions used in the notebook (*aux\_functions.py*) and the init script which determines the *src\_PN* folder as a module. You can either download the *src\_PN* folder from GitHub together with *final\_notebook\_aux\_functions\_imported.ipynb* or you can clone this whole repository using Git. Noted, the *src\_PN* folder needs to be in the same location as the notebook – in other words, it should be the following structure:

Figure 3: Module structure



Once you have cloned or downloaded the given files, now you will be able to run the notebook *final\_notebook\_aux\_functions\_imported.ipynb*. However, if you do not want to import the auxiliary functions from a module, you can download the notebook *final\_notebook\_aux\_functions\_included.ipynb* in which such functions are directly defined, therefore no module importing is needed. Both notebooks output the same results, thus you can use any of them based on your preference.

## 2.3 Loading fitted models and objects

To load the model or object successfully, we use a *pickle* package instead of the *load\_model* function from the *Tensorflow/Keras* package. If you want to load the model to the working space, then you should use the following code:

```
model_loaded = pickle.load(open('model.h5', 'rb'))
```



## **3 Data Understanding**

### **3.1 Data Import**

We used data provided by representatives of the Prague University of Economics and Business. This dataset is at the same time also available from the platform Kaggle. It was imported directly from the collaborative website GitHub. We also used data pertaining to the full categories' names, which have been explicitly extracted from Kaggle. The latter has been used in mapping the raw categories' names for better understanding and visualization.

### **3.2 Dataset**

Dataset is dedicated to mushroom characteristics where several variables are used to predict the target variable which is edibility. Independent variables include characteristics of cap, gill, stalk, veil, and ring, and further observations e.g., about bruises, odor, spore, population, and habitat.

In general, we obtained 22 independent variables, 1 target variable, and 8131 observations. Given the nature of the dichotomous target variable, we deal with a classification problem, i.e., if the mushroom from a particular observation is edible.

### **3.3 Initial Categorization**

Variables present in the dataset, both target and independent, have categorical characters, therefore these we transformed into a categorical format.

### **3.4 Duplicates and Missing Values**

Although we implemented procedures dealing with duplicates and missing observations, these are not a problem of this dataset since only 6 observations were removed due to duplicity and variable with numerous missing values is only stalk-root where missing values are treated as a single category – we deem it is some mushrooms do not have any stalk root.

### **3.5 Data Exploration**

Initially preprocessed data is used for starting univariate analysis where particular variables are analyzed in terms of their structure and their relationship to target variable. There are in total 23 variables in the dataset:

Figure 4: Exploration of variables

	count	unique	top	freq
class	8124	2	edible	4208
cap-shape	8124	6	convex	3656
cap-surface	8124	4	scaly	3244
cap-color	8124	10	brown	2284
bruises	8124	2	no	4748
odor	8124	9	none	3528
gill-attachment	8124	2	free	7914
gill-spacing	8124	2	close	6812
gill-size	8124	2	broad	5612
gill-color	8124	12	buff	1728
stalk-shape	8124	2	tapering	4608
stalk-root	8124	5	bulbous	3776
stalk-surface-above-ring	8124	4	smooth	5176
stalk-surface-below-ring	8124	4	smooth	4936
stalk-color-above-ring	8124	9	white	4464
stalk-color-below-ring	8124	9	white	4384
veil-type	8124	1	partial	8124
veil-color	8124	4	white	7924
ring-number	8124	3	one	7488
ring-type	8124	5	pendant	3968
spore-print-color	8124	9	white	2388
population	8124	6	several	4040
habitat	8124	7	woods	3148

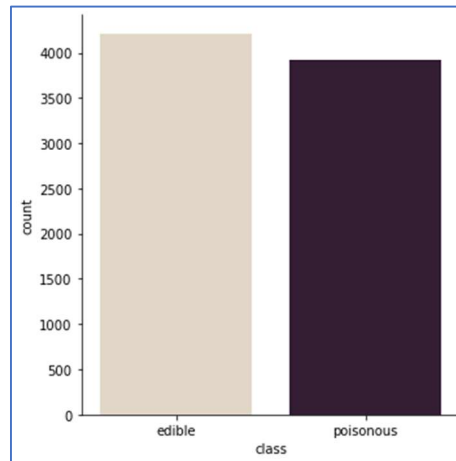
There was one observation, which had an invalid value “Ee” in variable stalk-shape, so this observation was also removed. After removing duplicated and invalid values, 8124 observations were left in the dataset. The values were remapped with the full names of the categories for better understanding and visualization.

### 3.5.1 Target and Feature Distribution

The distribution of the target variable seems almost uniform, therefore undersampling or oversampling is not necessary. The edible mushrooms are the majority category since this category has a 51.8 % proportion of the whole target variable.

In the project, we document also the distribution of categories for particular independent variables using a frequency histogram. These histograms may be unconditional or conditional concerning the target variable.

Figure 5: Distribution of target variable



For better understanding of the dataset, following conditional histograms visualize a few selected independent variables with respect to target variable.

Figure 6: Conditional distribution of bruises and odor, given the target variable

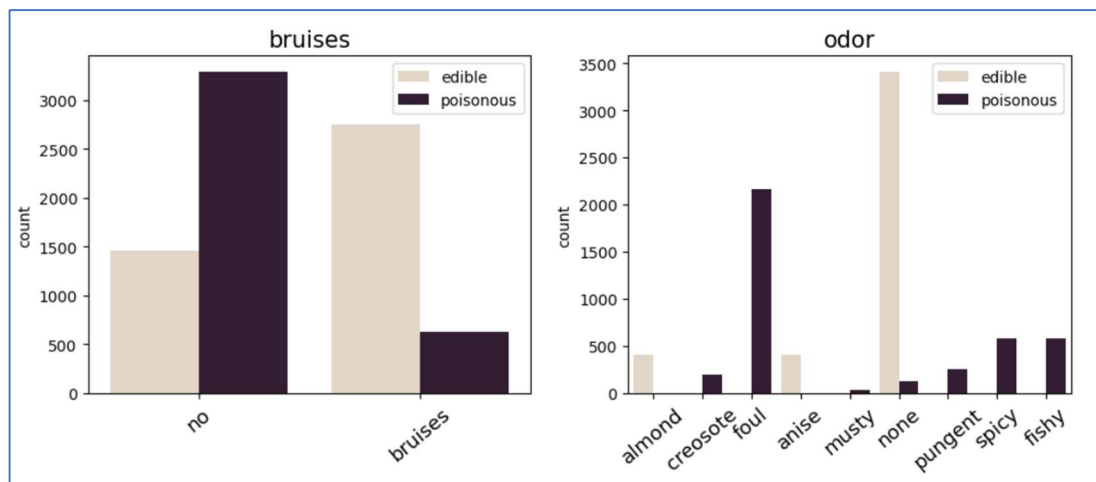
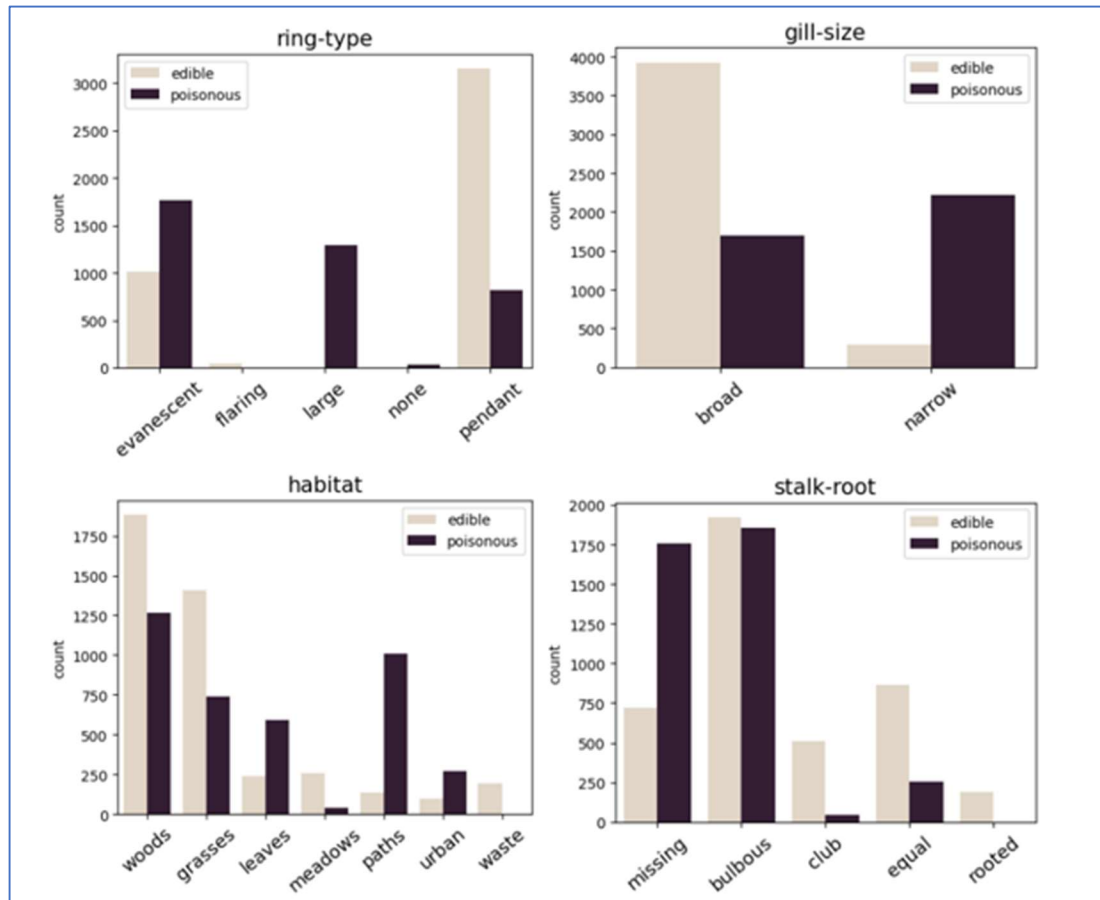


Figure 7: Conditional distribution of ring type and gill size, given the target variable



These graphs show at first glance that edible and poisonous mushrooms are not evenly distributed across categories, but that there is a relationship between the target variable "class" and other variables. This hypothesis is tested in the following section using contingency tables and chi-square tests.

### 3.5.2 Dependency Analysis

The following tables serve for testing whether there is a relationship between the target and other variables. Pearson's chi-squared test is used to determine whether there is a statistically significant difference between the expected frequencies and the observed frequencies in one or more categories of a contingency table.

Since this is statistical hypothesis testing, we first need to define the null and alternative hypotheses.

$H_0$ : At a given confidence level, the occurrence of outcomes is statistically independent.

$H_1$ : Alternative to  $H_0$ . There is some relationship between the variables.

Five variables were selected for the dependency analysis: bruises, odor, ring-type, gill-type and habitat. These variables were displayed in contingency tables on their relationship with the target variable – class.

The result of all the tests, interpreted by the p-value is 0. This means that  $H_0$  is rejected at all confidence levels. There is a statistically significant relationship between the tested variables and the edibility of the mushrooms. For example, from the contingency table that displays the relationship between odor and class, it is clear that all the observations in the dataset with the odor “almond”, “anise” and the majority of observations with odor “none” are edible. On the other hand, all the observations in a given dataset with odors “creosote”, “foul”, “musty”, “pungent”, “spicy” and “fishy” are poisonous. The following figure depicts one result of the dependency analysis, particularly between odor and class. Other results can be found in the notebooks.

Figure 8: Contingency analysis

The p-value is 0.0.  
Result: We reject the null hypothesis: There is a relationship between odor and class, thus they are dependent.

	observed values - edible	observed values - poisonous	expected values - edible	expected values - poisonous
odor				
almond	400	0	207.18858	192.81142
creosote	0	192	99.45052	92.54948
foul	0	2160	1118.81832	1041.18168
anise	400	0	207.18858	192.81142
musty	0	36	18.64697	17.35303
none	3408	120	1827.40325	1700.59675
pungent	0	256	132.60069	123.39931
spicy	0	576	298.35155	277.64845
fishy	0	576	298.35155	277.64845

The conditions of the Pearson chi-square test, which must always be verified before calculation, are as follows:

- The individual observations summarized in the contingency table are independent, i.e. each element of the sample is included in only one contingency table cell.
- At least 80 % of the cells of the contingency table have an expected frequency greater than 5 and all cells of the table have an expected frequency greater than 2. This condition is related to the asymptotic properties of statistics.

- The first condition is fulfilled, which follows from the nature of the data, where each observation takes just one of the values in each variable.
- The fulfillment of the second condition is not immediately apparent from the contingency table. However, by adding the expected values to the table, it is clear that none of the expected values is less than 2. Thus, the second condition for using this test is also satisfied. (Plackett, 1983)

### 3.6 General Preprocessing

Besides the main steps in dealing with duplicates and missing values, we have to further inspect the data quality of the provided dataset. Data quality is however very high and therefore, a typical operation that is conducted is the renaming of several observations with a typing error.

### 3.7 Data Split

According to approach in the course, we split the dataset into training set, validation set and testing set. The training set is used to develop the model on labeled data, i.e., with the known value of class. A validation set is used when we look for the best model. After several procedures final model is applied to the testing set (unlabelled data) on which we evaluate the performance of the model. Noted, we split our data with stratified sampling in order to preserve the same distribution of the target variable across the samples.

Our distribution of observations into such defined sets is 70% to training set, 15% to validation set and 15% to testing set.

### 3.8 Binning of Variables

The next step is a categorization of variables into smaller number of groups according to their relationship to the target variable, i.e., groups with a higher probability that the mushroom with given characteristic is edible or poisonous.

For this sake we use *optbinning* package described in Navas-Palencia (2022) and also fully implemented in Python environment. Optimal binning is technique of optimal discretization of independent variable into bin concerning discrete or continuous target, however, it is beneficial also in modelling of categorical variables.

The main goal of this method is a categorization of variables into smaller number of bins while fulfilling several general bin characteristics, e.g., binning missing values separately, holding to rule of at least 5% observations in each bin or that each bin has at least one of each binary response.

While binning of continuous variables consists mainly in looking for the best cut dividing into heterogeneous bins according to their relationship to target variable, in case of categorical variables we rather focus on combining various categories of each variable according to the similarity of their relationship to target variable. Categories are first ordered according to this relationship, and they are then through complex procedure gradually grouped into smaller bins until optimal bins are created. This approach is more relevant to our project because all observed independent variables are categorical. Binary independent variables do not require further binning.

### 3.9 WoE Transformation

The next step in transformation of input variables is so-called “WoEization”, respectively transformation of category into Weight of Evidence. Witzany (2017, p. 53) states that categorical variables are in regression usually present as dummies. However, an alternative approach suggests replacing of a single categorical variable with a single numerical variable, specifically expressed by its Weight of Evidence. The WoE is a measure that is calculated based on relationship of certain category of the feature to target variable. Assignment of numerical variables to bins leads to linear relationship between the explanatory variable and log-odds.

Weight of Evidence calculation follows this formula (Listen Data, 2015, own adjustment):

$$WoE = \ln \left( \frac{\text{distribution of } 0}{\text{distribution of } 1} \right).$$

For each categorical value, i.e., in our case for each bin, WoE is calculated as (Witzany, 2017, p. 49):

$$WoE(c) = \ln (\Pr((c|target = 0)) - \ln (\Pr((c|target = 1))),$$

where Pr refers to probability of the term in brackets, i.e., that categorical value of variable is c given the target result.

Given the fact that edible mushrooms are represented by 1, positive values of WoE represent lower probability of edible contrary to other categories whereas negative values represent higher probability of edible. In other words, negative values of WoE should indicate larger distribution of edibles compared to poisonous within given category.

In our application of data split into various sets, it is important to conduct binning and WoE transformation on the training set and then to transfer WoE values for particular categories of variables into other sets.

We export a Binning object as *binning\_woe\_model.h5* in h5 format. As it is built on the training set, it can be used to transform any sets of features (validation, test) that contains the same features and same features' categories to the WoE values.



## 4 Modelling

As mentioned previously, the general modelling approach in this project follows a classification problem, a subset of supervised learning problems, where input training labels are mapped to continuous valued predictions (usually in the form of probability) and output labels, also called predicted classes, coming in the form of a discrete category (Kuhn & Johnson, 2013, p. 247). For the practical application in our dataset, a discrete category prediction is necessary to decide whether a mushroom is edible or poisonous. It is desired that the estimated class probabilities are reflective of the true underlying probability of the sample.

### 4.1 Selection of Methods

The paper aims to apply four commonly used and generally successful methods (classifiers) that were the main subject of Data-X lectures and are well suited for classification problems. These selected methods are logistic regression, decision tree, random forest, and gradient boosting, implementation of which is based upon the recommended python library, Scikit-learn.

#### 4.1.1 Logistic Regression

The main goal of the project is to predict whether a mushroom is poisonous or not. This is also a principle of Logistic regression. It is used when the dependent variable is categorical. One of the most common logistic regression functions is logit and it is based on the Sigmoid function. The function has an “S” shape and takes any real-valued input, and outputs a value between 0 and 1.

$$S(x) = \frac{1}{1 + e^{-x}}$$

If independent variable  $x$  goes to positive infinity, the predicted output  $y$  will become 1 and if independent variable  $x$  goes to negative infinity the predicted output  $y$  reaches 0. The goal of prediction is binominal – whether mushroom is poisonous or not. (H. Belyadi, A. Haghighat, 2021). Noted, in general, logistic regression’s parameters are being estimated with maximum likelihood function. However, since we use Scikit-learn package for modelling, such package estimates LR’s parameters with a numerical optimization instead. By default, it uses Limited-memory Broyden–

Fletcher–Goldfarb–Shanno (*lbfgs*) which approximates the second derivative matrix updates with gradient evaluations (Towards Data Science, 2019).

### 4.1.2 Decision Trees

The decision trees algorithm is one of the non-parametric supervised learning methods. The main goal of using the decision tree algorithm is to train a model to predict the class or value of target variable's done by learning simple decision rules inferred from prior data – training set. One of the main assumptions is that values are preferred to be categorical.

The algorithm starts from the “root” of the tree, comparing the values from the root attribute. It has a flowchart structure and each decision step is represented by a node. (N.S. Chauhan, 2022). It chooses features and splits their values into nodes based on the largest purity in such node. The higher majority of the target class within the node, the larger purity. If the distribution of both classes is even, it has the highest impurity, or, so-called entropy. Our goal is to minimize entropy.

Examples of decision tree algorithms are following:

- ID3: “Iterative Dichotomiser” - algorithm leverages entropy and information gain as a metrics to evaluate splits.
- C4.5: enhanced ID3 leverages information gains to evaluate split points within the decision tree.
- CART: “Classification and regression tree”. It utilizes Gini impurity to identify the ideal attribute for splitting. Gini impurity measures how often a randomly chosen attribute is misclassified. The lower score, the better. (IBM Learn, 2022)

Since we use Scikit-learn package, the Decision Tree Classifier uses CART algorithm, but it does not support categorical variables for now, therefore they need to be encoded first. (Scikit-learn, 2009).

### 4.1.3 Random Forest

Random forests are based on Decision tree classification algorithms. Put simply, it is a set of decision trees. There are two main methods:

- Ensemble Method: made up of classifiers (decision trees) and the each prediction is aggregated to identify the most popular result. The most popular is bootstrap aggregation.
- Feature randomness method: extension of bootstrap aggregation in order to create uncorrelated forest of decision trees.

To avoid correlations among trees, the hyperparameters have to be chosen. These include node size, the number of trees, and the number of features sampled. (IBM Cloud Education, 2020).

In Scikit-learn, the random forests have  $n$  trees where each tree is built from a bootstrap sample (i.e., sample drawn with replacement) from the training set, which gives randomness to the model. The second source of randomness lies in splitting the nodes during the construction of a tree as the best split is found from a random subset of the size of the random subsets of features. Both sources of randomness can then decrease the variance of the forest estimator since each individual tree typically tend to overfit. (Scikit-learn, 2012).

### 4.1.4 Gradient Boosting Classifier

Gradient boosting represents an algorithm derived from decision trees and random forests. It is used to enhance prediction power by converting a number of weak learners to strong learners and it is training the minimize the loss function – in Scikit-learn the log-loss is being used.

This algorithm starts by building a decision stump – weak learner. Then assigning equal weights to all the data points. The third step increases the weights for all the points which are misclassified and lower the weight for those that are correctly classified. A new decision stump is made for these weighted data points. It is done by building a new model on the errors or residuals of the previous model. (Analytics Vidhya, 2021).

Particularly, within Gradient Boosting, each model tries to improve on its predecessor by reducing the errors – instead of fitting a model on the data at each

iteration, it fits a new model on the residual errors made by the previous predictor. (Towards Data Science, 2020).

## 4.2 Feature Selection

Given the nature of the dataset, determining which predictors should be included in a model presents a critical question. Feature selection is primarily focused on removing non-informative or redundant predictors from the model and it is dependent on the model used. Regression-based models will estimate parameters for every term in the model. Because of this, the presence of non-informative variables can add uncertainty to the predictions and reduce the overall effectiveness of the model.

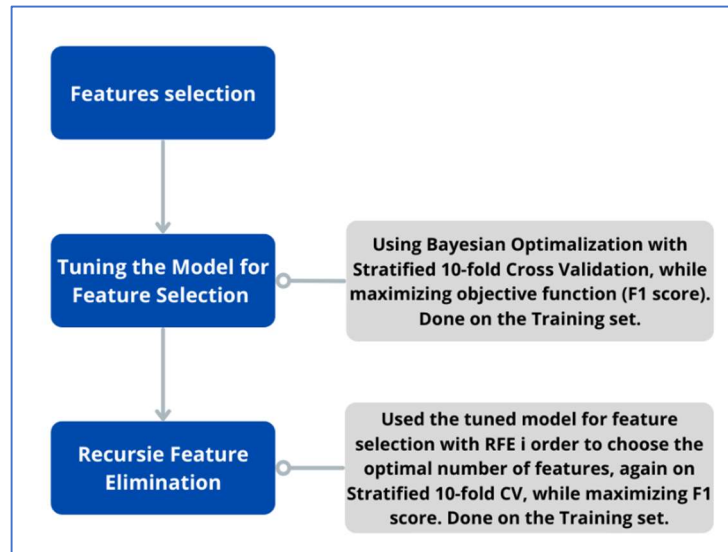
Statistically, it is often more attractive to estimate fewer parameters since a model with fewer predictors may be more interpretable. Some models may also be negatively affected by non-informative predictors. For supervised methods, predictors are specifically selected to increase accuracy or to find a subset of predictors to reduce the complexity of the model. The outcome is typically used to quantify the importance of the predictors (Kuhn & Johnson, 2013, p. 487-490).

The issues related to each type of feature selection are very different, and the literature for this topic is very extensive.

In the case of this paper, the approach for feature selection first uses stratified 10-fold Cross Validation within Bayesian Optimization algorithm (Frazier, 2018), exploring the most promising model hyperparameters, which are then used for feature selection. Bayesian Optimization finds the maximum (or minimum) to an objective function in large problem-spaces and is very applicable to continuous values, such as in our case after WoE. Possible class imbalance is taken into account through maximization of the objective function F1 score (combining precision and recall). Implementation in this case uses a popular library for model-based optimization, Scikit-optimize.

The optimal features are selected using Recursive feature elimination (RFE), and effective algorithm at selecting the most relevant features in predicting the target variable, where stratified 10-fold Cross Validation (maximizing the F1 score) is again applied within RFE.

Figure 9: Feature selection



Listed in the next 2 chapters is a deeper look on the methods used for feature selection. It should be noted that hyperparameter tuning and feature selection is done on the training set for each model.

After the optimal binning, found 4 features which had only one bin, or one category after binning. Therefore, we exclude such features since they have zero-variance and would not contribute anything in terms of predictions. In the following figure, we can see those 4 features having only one bin, hence zero value of WoE. After excluding such features, we end up with 18 features in overall.

Figure 10: List of exclude features having only 1 bin

Bin	Variable
[creosote, fishy, foul, musty, pungent, spicy, none, almond, anise]	odor
[free, attached]	gill-attachment
[partial]	veil-type
[yellow, white, brown, orange]	veil-color

For a reference, see the notebook, where you can find a WoE distribution of each feature and list of categories which should imply larger distribution of given class.

### 4.2.1 Bayesian Optimization

In tuning given model's hyperparameters, we are trying to find a combination maximizing the model's performance (based on some criteria). One way to approach this is to do a naive grid search over the possible combinations, however, for larger datasets or hyperparameter spaces, this will likely be costly to compute. A generally

more efficient solution is to define a search space as a bounded domain of hyperparameter values and randomly sample points in that domain (Bergstra & Bengio, 2012).

Bayesian Optimization, also called Sequential Model-Based Optimization (Bergstra et al., 2011) takes this a step further, in updating the sample search space based on outcomes of prior searches. In this way, it can reduce the number of evaluations necessary to find the optimal parameters. Let's assume a function that takes in hyperparameters and outputs a single real-valued score, the so-called objective function. For example, a random forest regressor outputting the RMSE score.

The algorithm builds a probability model of the objective function and uses it to select the most promising hyperparameters to evaluate in the true objective function (Dewancker, McCourt & Clark, 2016). Such model is also referred to as the surrogate model of the objective function, response surface, or simply the surrogate function. It represents the probability representation of the objective function built using previous evaluations (essentially a model trained on the [hyperparameter, true objective function score] pairs).

Evaluations are performed through pre-determined criteria, called an acquisition, or a selection function, since it determines which hyperparameters are to be used in the next surrogate model. There are several different forms of surrogate functions including Gaussian Processes or the Tree-structured Parzen Estimator (Bergstra et al., 2011), which aims to maximize the Expected improvement of the proposed hyperparameters.

Once a hyperparameter (or a set of them) maximizing the acquisition function is found, the objective function score of such parameter is evaluated and included (as well as the parameter) in the history of surrogate function samples. The surrogate model is then trained using the latest history, and the process repeats until a max number of iterations is reached. Sorting the history of scores yields the best parameters found.

In the case of this project, iteration count in feature selection is set to 50 for each model. Search space definitions for each model are summarized in the following table.

Noted,  $x\_train.columns$  refer to the features names from the training set (omitting the excluded features having only one bin).

*Table 1: Bayesian optimization setting*

Model	Random Forest	Decision Tree	Gradient Boosting
<b>N_estimators</b>	Integer (1, 1000)		Integer (1, 1000)
<b>Max_depth</b>	Integer (1, 15)	Integer (1, 15)	Integer (1, 15)
<b>Max_features</b>	$x\_train.columns$	$x\_train.columns$	$x\_train.columns$
<b>Min_samples_leaf</b>	Integer (5, 500)	Integer (5, 500)	Integer (5, 500)
<b>Criterion</b>	['gini', 'entropy']	['gini', 'entropy']	

For the case of Logistic Regression,  $fit\_intercept$  [True, False] and the degree of regularization - Real (0.001, 1000), are only included.

As mentioned previously, stratified 10-fold Cross Validation is used within Bayesian Optimization, with the Cross Validation F1 score serving as the objective function being maximized.

Cross-validation consists in the division of the training dataset into  $k$  parts (folds). Firstly,  $k-1$  parts are used for training and the remaining part for testing. This process is repeated  $k$  times. Stratification sampling describes the process where all samples have the same proportion of target variables and therefore do not differ from each other (Towards Data Science, 2021).

The F1 score (also called the F score) represents a measure of a model's accuracy. It is used to evaluate binary classification systems, which classify examples into 'positive' or 'negative', such as in the case of this project. It is a way of combining the precision and recall of a model, and it is defined as the harmonic mean of the model's precision and recall (Wood, 2019). More details about the F1 score in later chapters.

As hyperparameter combinations are being put forward by the surrogate and selection function, gradual focus on a specific area of each hyperparameter distribution is being set. Based on past evaluations, hyperparameters equating to better cross-validations scores are being found, thus are being tuned.

As a note, Random Search can outperform Bayesian Optimization in some cases, as it could bump onto the optimal set of hyperparameters right at the start by chance.

## 4.2.2 Recursive Feature Elimination

As datasets continue to increase in size, it is important to select the optimal feature subset from the original dataset to obtain the best performance in machine learning tasks. Highly dimensional datasets that have an excessive number of features can cause low performance in such tasks. Overfitting is a typical problem. In addition, datasets that are of high dimensionality can create shortages in space and require high computing power, and models fitted to such datasets can produce low classification accuracies. Thus, it is necessary to select a representative subset of features by utilizing an efficient selection algorithm (Jeon & Oh, 2020). As such algorithm, Recursive Feature Elimination is used in this project.

Recursive feature elimination (RFE) is a feature selection algorithm that attempts to select the optimal feature subset based on the learned model and classification accuracy. When a classifier is trained with a training dataset, feature weights that reflect the importance of each feature can be obtained. After all features are ranked according to their weights, the feature that has the lowest weight value is removed. Then, the classifier is re-trained with the remaining features until it has no features left with which to train. Finally, the entire ranking of the features using the feature-importance-based RFE method can be obtained (Jeon & Oh, 2020, p. 2-3).

In the case of this thesis, similar logic for feature selection is applied as with Bayesian Optimization (Cross Validation within RFE, F1 score maximization). Below listed is the number of features selected by each model after RFE applied, with Random Forest having 12 features selected as the relative minimum, Decision Tree, on the other hand, ended up with the most features selected (18). To see which particular features have been selected within each RFE iteration, check the notebook outputs.

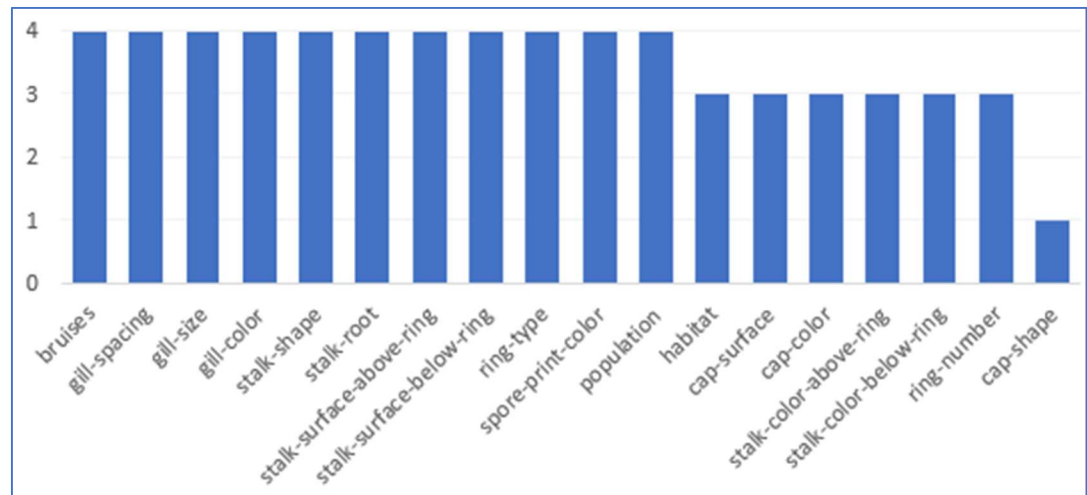
*Table 2: Recursive feature elimination results*

<b>Model</b>	<b># Features Selected</b>
<b>Random Forest</b>	12
<b>Logistic Regression</b>	16
<b>Gradient Boosting</b>	17
<b>Decision Tree</b>	18



List of features selected with their recurrence (count of how many times a feature has been selected amongst models) can also be seen on the following chart. The most recurrent features are related to the gill, ring type, spore color and population, as well as the stalk root or surface. Cap shape is the only feature having been selected only once, by Decision Tree. At the end, 4 sets of optimal features selected by each model are obtained.

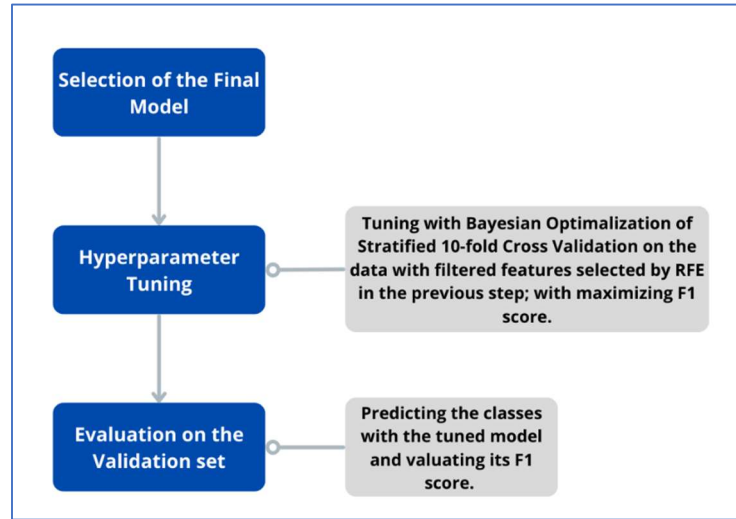
*Figure 11: Recurrence of the preselected features*



### 4.3 Hyperparameter Tuning & Final Model Selection

Now that the relevant features are selected, each model's hyperparameters are again tuned using Bayesian Optimization with Stratified 10-fold Cross Validation, maximizing F1 score. In this case, however, each set of optimal features is being used to tune a given model on the training set (evaluation is done on the validation set). Thus, we end up with 4x4 [model, feature set] combinations, each outputting tuned hyperparameters for the given combination. In other words, if we use  $n$  models as an input for feature selection and for final model selection, we end up with  $n^2$  tuned models.

Figure 12: Hyperparameter tuning and Final Model selection



Multiple metrics besides F1 score are tracked through this process, such as accuracy, precision, recall, AUC, Gini, Kolmogorov Smirnov distance or Brier score loss. The model with the highest F1 score on the validation set is preferred to be used as the final model, however, in the case with multiple models competing in this regard, other metrics are to be used to determine the final model selection. The model selection matrix can be found in Appendix A of this project.

The best performing model is Gradient Boosting using 12 features selected by Random Forest within RFE. Thus, this is the final model which will be used in the final phase of modelling and evaluation. More information about the model is listed in the table below.

Figure 13: General information about final model

<b>Tuned Model Name</b>	Gradient Boosting
<b>Feature Selection Model</b>	Random Forest
<b># Features</b>	12
<b>Final features</b>	bruises, gill-spacing, gill-size,
	gill-color, stalk- shape, stalk-root,
	stalk-surface-above- ring, habitat
	stalk-surface-below- ring, ring-type, spore-print-color, population

Figure 14: Hyperparameters of final model

Hyperparameters			
criterion	friedman_mse	min_impurity_decrease	0.0
max_depth	14	min_weight_fraction_leaf	0.0
max_features	12	n_estimators	1000
min_samples_leaf	5	n_iter_no_change	None
min_samples_split	2	subsample	1.0
max_leaf_nodes	None	tol	0.0001
ccp_alpha	0.0	validation_fraction	0.1
init	None	verbose	0
Learning_rate	100.0	warm_star	FALSE
loss	deviance		

To elaborate further, other non-linear models such as Random Forest or Decision Tree have performed perfectly as well. On the other hand, a linear model, Logistic Regression, did worse compared to the non-linear ones, but still performed well.

## 4.4 Building the Final Model

Once we have selected our final model, the next step is the building the model. Instead of fitting the model solely on the training set while leaving the validation set aside, in order to increase the model's validity and performance, we deem as reasonable to fit the model on the joined training and validation set. Such joined set has more observations, hence building the model tends to improve the model's performance even more. We export such model as *final\_model.h5* in h5 model. Besides that, we also export the feature selection model which has been used as an input within feature selection as *fs\_model.h5*, and the model RFE object as *rfe\_model.h5*. Of course, we export such feature selection and RFE object which are related to the final model.

Noted, we subset all the samples of features (training, validation, test) which contain only the final features selected by the *rfe\_model.h5*. Thus, the model is built on the joined training and validation set containing only the final features.

## 5 Evaluation

The last important step is to evaluate built model using several evaluation metrics. We decided to use the most common evaluation measures from which the most important in final model selection is F-score.

### 5.1 Confusion Matrix Measures

The confusion matrix is crucial for a wide variety of evaluation measures we use. It provides information about classification of cases by model compared to real class of these cases. Its structure is following:

Table 3: Confusion matrix

Confusion matrix		Classification by model	
		1/Yes	0/No
Real classification	1/Yes	TP	FN
	0/No	FP	TN

source: Berka (2003, p. 226), own processing

where TP stands for true positive, FP for false positive, FN for false negative, and TN for true negative. True positive and true negative represent correct classification of the cases. False positive and false negative are incorrectly classified cases (Berka, 2003, p. 226).

Measures assessing the model quality are derived using the absolute values from this 4-fold table. We can calculate precision and recall as a share of certain field on column or row sum. Their formulas are:

$$Precision = \frac{TP}{TP + FP},$$

$$Recall = \frac{TP}{TP + FN}.$$

Both precision and recall can be taken into account in the measure called F-score (ibid, p. 227-228), which is the harmonic mean of both measures:

$$F = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN}.$$

Besides that, we can also calculate the accuracy as a ratio of correctly classified instances (TP and TN) to the all instances.

## 5.2 AUROC

Area Under the Receiver Operating Characteristics Curve (AUROC) is also based on confusion matrix. It is calculated using true positive rate (TPR) and false positive rate (FPR), calculated as follows (Berka, 2003, p. 233-234):

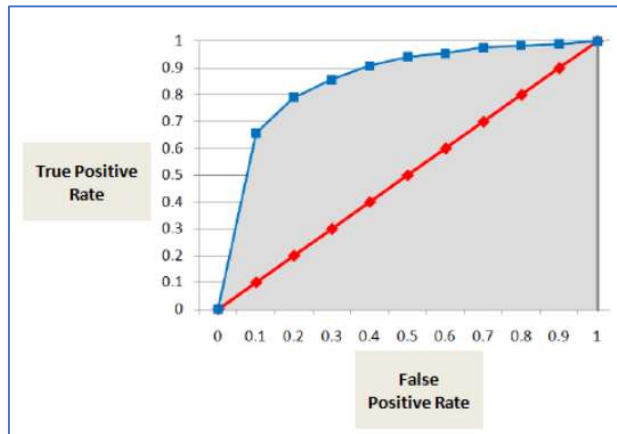
$$TPR = \frac{TP}{TP + FN},$$

$$FPR = \frac{FP}{FP + TN}.$$

These two measures are put into relationship and in the following picture, blue line represents ROC curve and grey field below stands for AUROC (also referred to as AUC). Similar to previous measures, we aim to maximize it. AUROC (AUC) measures the entire two-dimensional area underneath the entire ROC and ranges in value from 0 to 1. A model whose predictions are 100 % wrong has an AUROC (AUC) of 0.0; one whose predictions are 100 % correct has an AUC of 1.0. (Google Developers, 2022). Mathematically, we can derive AUC as an definite integral of TPR and FPR functions with respect to  $x$  which is particular threshold. Noted, using Scikit-learn, the AUC is being calculated with a numeric approximation using a trapezoidal rule.

$$AUROC = \int_0^1 TPR(FPR(x)^{-1}) dx$$

Figure 15: AUROC and ROC Curve



source: Listen Data (2019)

AUROC is also directly related to Gini coefficient according to formula as its linear transformation:

$$Gini = 2 * AUROC - 1 .$$

### 5.3 KS Statistics

The Kolmogorov Smirnov statistic tells you whether the model gets confused when it comes to predicting the different labels in your dataset. (K-S Statistic Plot - Machine Learning with Scikit-Learn Quick Start Guide [Book]). It compares cumulative distribution functions of both target classes while maximizing its vertical distance between these curves.

Mathematically, we can describe the KS as following, where we are looking for the maximum difference between the cumulative distribution of the positive cases  $F_{n,positive}(q)$  and negative cases  $F_{n,negative}(q)$ , assuming  $L$  and  $H$  are the minimum and maximum value of the scored probability.

$$KS = \max_{q \in [L,H]} |F_{n,positive}(q) - F_{n,negative}(q)|$$

We use *stats* module from SciPy package to calculate the KS statistics, particularly the function *k\_2samp*.

### 5.4 Brier Score

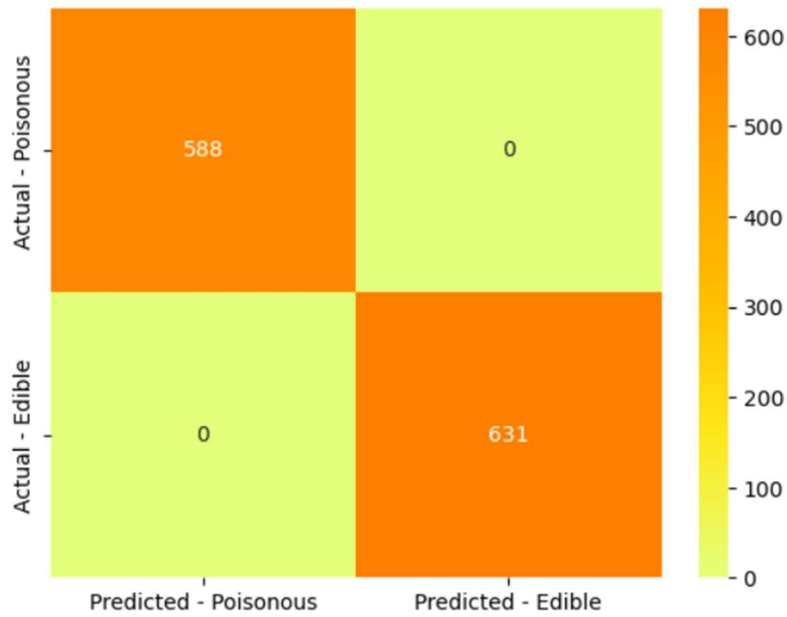
The Brier score measures the mean squared difference between the predicted probability and the actual outcome. The Brier score always takes on a value between zero and one, since this is the largest possible difference between a predicted probability (which must be between zero and one) and the actual outcome (which can take on values of only 0 and 1). It can be decomposed is the sum of refinement loss and calibration loss. („Sklearn.Metrics.Brier\_score\_loss". Scikit-Learn,). In contracts to the above-mentioned measures, or goal is to minimize the Brier score.

$$Brier = \frac{1}{n} \sum_{i=1}^n (y_{true} - y_{prob})^2$$

## 5.5 Evaluation Metrics of the Final Model

Down below, we can find the evaluation results of our final model – Gradient Boosting Classifier. The following plot depicts the confusion matrix evaluated on the test set. As expected from the validation set’s results, it distinguishes between both target classes perfectly as it correctly classifies edible mushrooms as edibles and poisonous mushrooms as poisonous, without any misclassifications.

Table 4: Confusion matrix of final model



From the confusion matrix, we can derive metrics such as F1 score, Recall, Precision or Accuracy, which logically are equal to 1. The same also applies to the metrics related to the estimated probability such as AUC, Gini or KS which are also equal to 1. Lastly, the Brier score loss function is 0, thus our model is performing perfectly on all the defined metrics.

Table 5: Other evaluation metrics of final model

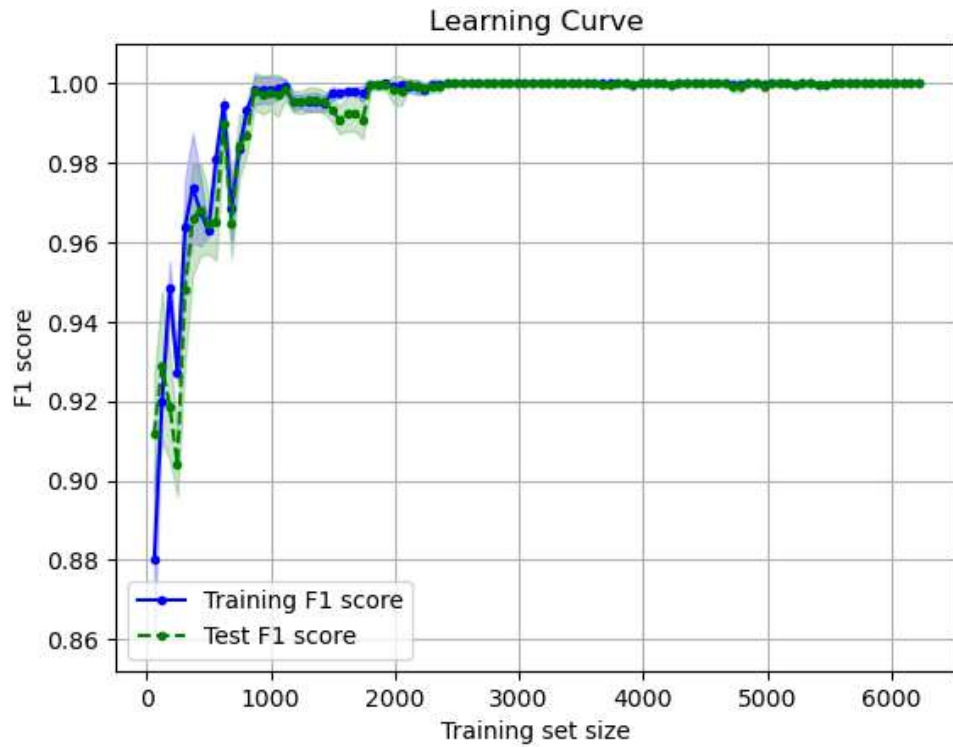
	Metric	Score
0	F1	1.00000
1	Recall	1.00000
2	Precision	1.00000
3	Accuracy	1.00000
4	AUC	1.00000
5	Gini	1.00000
6	KS	1.00000
7	Brier	0.00000



Besides other metrics, we can also plot the Learning curve which determines cross-validated training and test scores for different training set sizes. It splits the whole dataset  $k$  times into training and test data, where the subsets of the training set with varying sizes will be used to train the estimator and a score for each training subset size  $k$  runs for each training subset size. (Scikit-learn, 2019)

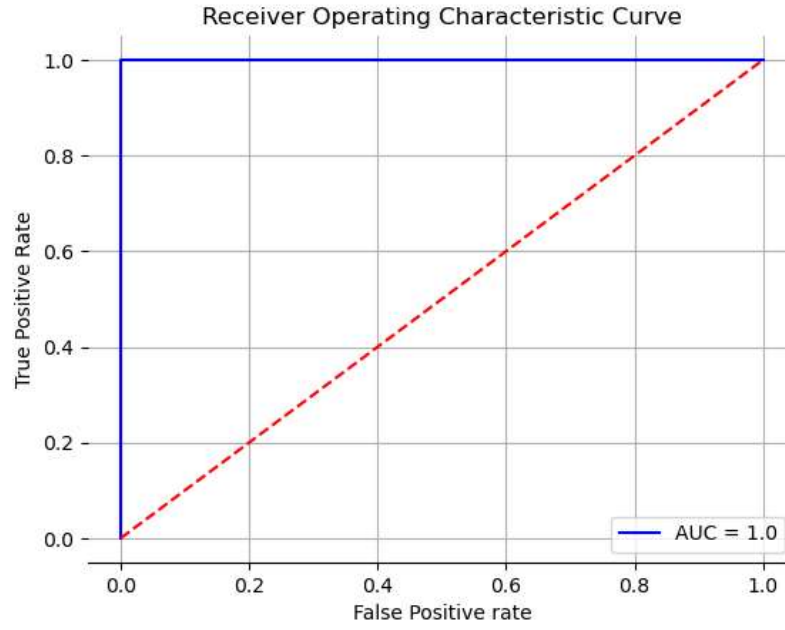
We fit the learning curve on the joined training and validation set with the final model, using stratified 10-fold cross validation with a scoring function F1. As can be seen, the variance of both samples becomes minimal with increasing size of the training sample. Furthermore with increasing number of the training sample, both curves are converging towards F1 score of being 1. Thus, our model should not tend to overfit.

Figure 16: Learning curve of final model



Pertaining to the AUC, we can also plot the ROC curve. As can be seen, the peak of the ROC curve is in the top left corner – there is no area left above the ROC curve, thus the AUC is 1, which is the best scenario regarding the model performance, as its TPR is 1 and FPR is 0.

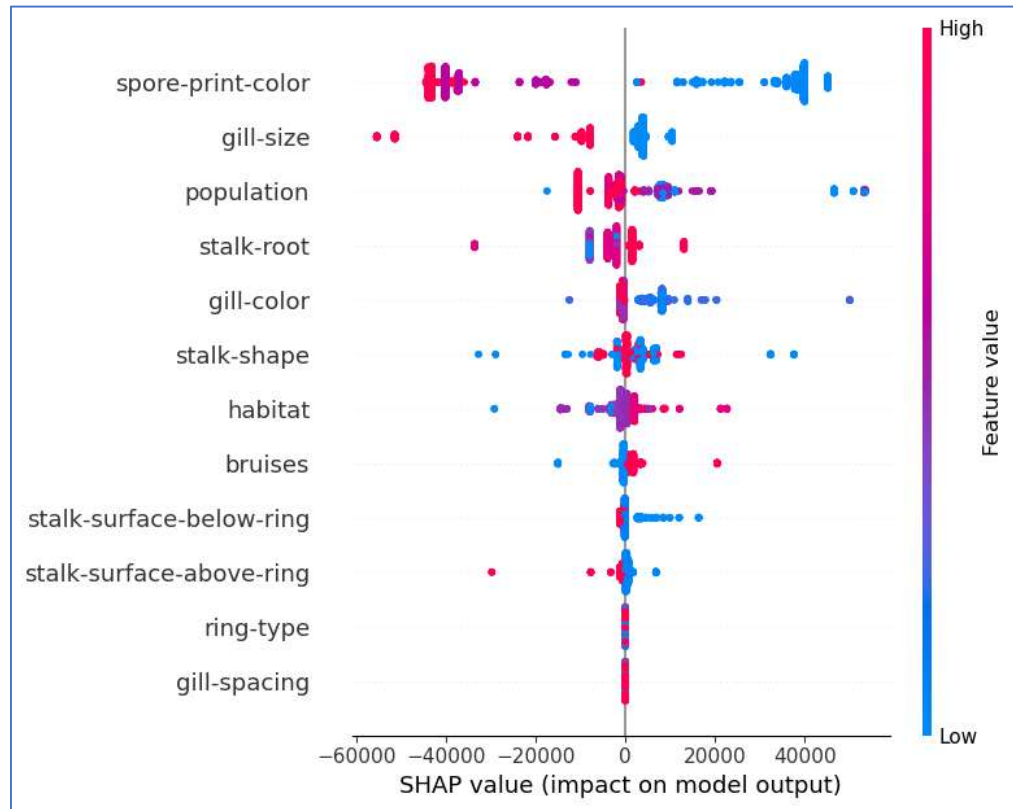
Figure 17: ROC curve of final model



SHAP (SHapley Additive exPlanations) is a model agnostic method to explain individual predictions or model-wise interpretation. The method is based on cooperative game theory and used to increase transparency and interpretability of machine learning models. SHAP shows the contribution or the importance of each feature on the prediction of the model, it does not evaluate the quality of the prediction itself.

For example, high values of the spore-print-color variable have a high negative contribution on the prediction, while low values have a high positive contribution. The spore-print-color variable has a really high positive contribution when its values are high, and a low negative contribution on low values. Features ring-type and gill-spacing has almost no contribution to the prediction, whether its values are high or low.

Figure 18: SHAP values of final model



## 5.6 Limitations

One of the limitations of our model is that the Gradient Boosting Classifier is a black-box model as its interpretability is not transparent as within the Logistic Regression or simple Decision Tree Classifier. If we prefer a performance over the interpretability, we would choose Gradient Boosting Classifier. Otherwise, if the interpretability is our main focus, we would choose logistic regression or decision tree. Noted, that the Decision Tree Classifier has almost perfect results as Gradient Boosting Classifier, so if we replace the final model with such Decision Tree Classifier, it would not make any difference.

Pertaining to the improvements, we can try to use more models such as Support Vector Machine, Neural Networks, Naïve Bayes or k-Nearest Neighbours and see how they would perform compared to our models. We also recommend using Stacking Classifier or Voting Classifier, which can use several models' predictions and aggregate them by a majority voting. Thus, we could combine predictions of Gradient Boosting Classifier and Decision Tree Classifier and boost the final model's performance even more.

## Sources

The main source of this project are the presentations and information obtained by the course Data-X – Applied Data Analytics Models in Real World Tasks (4IT439) taught by Mr. Pavel Zimmermann, Ph.D., Mr. Filip Habarta and Mr. William Galindez Arias.

Analytics Vidhya (2021). Gradient Boosting Algorithm: A Complete Guide for Beginners. Retrieved October 21, 2022, from <https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/>

Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. J. Mach. Learn. Res., 13(null), 281–305.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. Advances in neural information processing systems, 24.

Berka, P. (2003). Knowledge discovery in databases (Original in Czech: Dobývání znalostí z databází). Academia.

Dewancker, I., McCourt, M., & Clark, S. (2016). Bayesian optimization for machine learning: A practical guidebook. arXiv preprint arXiv:1612.04858.

Frazier, P. I. (2018). A Tutorial on Bayesian Optimization. StatML. Retrieved from <https://arxiv.org/pdf/1807.02811.pdf>

IBM. What is a Decision Tree. Retrieved October 20, 2022, from <https://www.ibm.com/topics/decision-trees>

IBM. What is a Random Forest. Retrieved October 21, 2022, from <https://www.ibm.com/cloud/learn/random-forest>

Jeon, H., & Oh, S. (2020). Hybrid-Recursive Feature Elimination for Efficient Feature Selection. Applied Sciences, 10(9), 3211. doi:10.3390/app10093211

KDnuggets. Decision Tree Algorithm Explained. Retrieved October 20, 2022, from <https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html>

Kuhn, M., & Johnson, K. (2013). Applied predictive modeling. Springer.

Listen Data. (2015). Weight of Evidence (WOE) and Information Value (IV) Explained. Retrieved October 15, 2022, from <https://www.listendata.com/2015/03/weight-of-evidence-woe-and-information.html>

Listen Data. (2019). Gini, Cumulative Accuracy Profile, AUC. Retrieved October 22, 2022, from <https://www.listendata.com/2019/09/gini-cumulative-accuracy-profile-auc.html>

Navas-Palencia, G. (2020). Optimal binning: mathematical programming formulation. arXiv preprint arXiv:2001.08025.

Provost, F., & Fawcett, T. (2013). Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking (First ed.). O'Reilly Media.

ScienceDirect. Logistic Regression. Retrieved October 20, 2022, from <https://www.sciencedirect.com/topics/computer-science/logistic-regression>

scikit learn. (2009) 1.10. Decision Trees — Scikit-Learn 0.22 Documentation. Scikit-Learn.org. Retrieved October 23, 2022, from <https://scikit-learn.org/stable/modules/tree.html>

scikit learn. (2012) 1.11. Ensemble Methods — Scikit-Learn 0.22.1 Documentation. Scikit-Learn.org. Retrieved October 23, 2022, from <https://scikit-learn.org/stable/modules/ensemble.html#forest>

scikit learn. (2009) Learning Curve — Scikit-Learn 0.21.3 Documentation. Scikit-Learn.org. Retrieved October 23, 2022, from [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.learning\\_curve.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.learning_curve.html)

Towards Data Science (2019). Don't Sweat the Solver Stuff. Retrieved October 22, 2022, from <https://www.towardsdatascience.com/dont-sweat-the-solver-stuff-aea7cddc3451>

Towards Data Science (2020). Gradient Boosting Classification Explained through Python. Retrieved October 22, 2022 from <https://www.towardsdatascience.com/gradient-boosting-classification-explained-through-python-60cc980eeb3d>

Towards Data Science (2021). What is Stratified Cross-Validation in Machine Learning? Medium. Retrieved October 15, 2022, from <https://towardsdatascience.com/what-is-stratified-cross-validation-in-machine-learning-8844f3e7ae8e>

Witzany, J. (2017). Credit Risk Management: Pricing, Measurement, and Modeling (1st ed. 2017 ed.). Springer.

Wood, T. (2019). F-score. DeepAI. Retrieved October 21, 2022, from <https://deepai.org/machine-learning-glossary-and-terms/f-score>

Plackett, R. L. (1983). Karl Pearson and the chi-squared test. International statistical review/revue internationale de statistique, 59-72.



# List of Figures and Tables

## *List of Figures*

Figure 1: Project workflow .....	4
Figure 2: Scheme of files .....	5
Figure 3: Exploration of variables .....	9
Figure 4: Distribution of target variable .....	10
Figure 5: Conditional distribution of target variable .....	10
Figure 6: Conditional distribution of target variable II .....	11
Figure 7: Contingency analysis .....	12
Figure 8: Feature selection.....	20
Figure 9: Preselected features .....	24
Figure 10: Hyperparameter tuning and Model selection .....	25
Figure 11: General information about final model .....	26
Figure 12: Hyperparameters of final model.....	26
Figure 13: AUROC and ROC Curve .....	29
Figure 14: ROC of final model.....	33
Figure 15: Learning curve of final model.....	32
Figure 16: SHAP values of final model.....	34

## *List of Tables*

Table 1: Bayesian optimization setting.....	22
Table 2: Recursive feature elimination results .....	23
Table 3: Confusion matrix .....	28
Table 4: Confusion matrix of final model .....	31
Table 5: Other evaluation metrics of final model .....	31

# Appendix

Appendix A, Model Selection Matrix

Tuned Model	FS <sup>1</sup> Model	# Features	F1	Precision	Recall	Accuracy	AUC	Gini	KS	Brier
Gradient Boosting	Random Forest	12	1	1	1	1	1	1	1	0
Gradient Boosting	Gradient Boosting	17	1	1	1	1	1	1	1	0
Gradient Boosting	Decision Tree	18	1	1	1	1	1	1	1	0
Decision Tree	Random Forest	12	1	1	1	1	1	1	1	0
Decision Tree	Logistic Regression	16	1	1	1	1	1	1	1	0
Decision Tree	Gradient Boosting	17	1	1	1	1	1	1	1	0
Decision Tree	Decision Tree	18	1	1	1	1	1	1	1	0
Random Forest	Random Forest	12	1	1	1	1	1	1	1	0.0001
Random Forest	Logistic Regression	16	1	1	1	1	1	1	1	0.0001
Random Forest	Gradient Boosting	17	1	1	1	1	1	1	1	0.0004
Gradient Boosting	Logistic Regression	16	1	1	1	1	1	1	1	0.0695
Random Forest	Decision Tree	18	1	1	1	1	1	1	1	0.0006
Logistic Regression	Logistic Regression	16	0.9784	0.9658	0.9912	0.9773	0.9936	0.9872	0.9586	0.0192
Logistic Regression	Gradient Boosting	17	0.9784	0.9658	0.9912	0.9773	0.9936	0.9872	0.9613	0.0191
Logistic Regression	Decision Tree	18	0.9773	0.9638	0.9912	0.9762	0.9936	0.9872	0.9599	0.0190
Logistic Regression	Random Forest	12	0.9607	0.9565	0.9649	0.9591	0.9892	0.9784	0.9349	0.0309

<sup>1</sup> Feature Selection model which was used as in input within RFE.