MUSHROOM CLASSIFICATION

Report

Name : Vishal Thapa

Registration Number : 11905984

Section : KM018

Github link : https://github.com/Vishalthapa0807/Mushroom-Classification

ABSTRACT

Mushroom is found to be one of the best nutritional foods with high proteins, vitamins and minerals. It contains antioxidants that prevent people from heart disease and cancer. Around 45000 species of mushroom are found to be existing in the world-wide. Among these, only some of the mushroom varieties were found to be edible. Some of them are really dangerous to consume. In order to distinguish between the edible and poisonous mushrooms in the mushroom dataset which was obtained from UCI Machine Learning Repository, some data mining techniques are used.

INTRODUCTION

Mushroom is considered to be one of the super food sources of vitamins, minerals and several nutrients. Mushrooms are low in calories, they are free of fat, cholesterol and gluten and the sodium levels in mushroom are found to be low. Thus, these are some of the facts that make mushroom to be one of the healthier foods. Thus, it is important to classify the mushrooms as edible and poisonous. For classifying the mushrooms as edible and poisonous, a dataset containing 8124 instances and 23 attributes of mushroom was obtained from UCI Machine Learning Repository.

Mushroom classification is a beginner machine learning problem and the objective is to correctly classify if the mushroom is edible or poisonous by its specifications like cap shape, cap colour, gill colour, etc. using different classifiers.

In this project I have used the following classifiers to make the prediction:-

- Logistic Regression
- SVM
- Random Forest Classifier

DATASET INFORMATION

This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. The dataset used in this project contains 8124 instances of mushrooms with 23 features like cap-shape, cap-surface, cap-color, bruises, odor, etc.

The python libraries and packages used in this project are namely:

- NumPy
- Pandas
- Seaborn
- Matplotlib
- Scikit-learn

Attributes Information

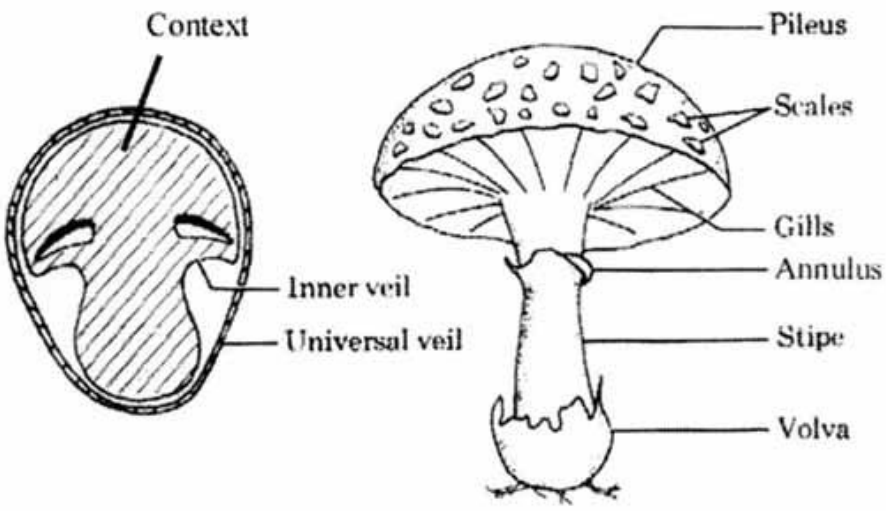
This dataset is named Mushroom Classification. The dataset contains a set of 8,124 records under 23 attributes:

Column Name	Description	Values
class	Edible or poisonous	e = Edible, p = poisonous
cap-shape	The shape of the expanded, upper part of the mushroom	b= bell, c = conical, x = convex, f = flat, k = knobbed, s = sunken
cap-surface	The structure of the upper part of the mushroom	f = fibrous, g = grooves, y = scaly, s = smooth
cap-color	The color of the surface of the upper part of the mushroom	n = brown, b = buff, c = cinnamon, g = grey, r = green, p = pink, u = purple, e = red, w = white, y = yellow
bruises	Indicates of there are bruises on the mushroom	t = yes, f = no
odor	The smell the mushroom omits	a = almond, l = anise, c = creosote, y = fishy, f=foul, m = musty, n = none, p = pungent, s = spicy
gill-attachment	The way the gill is growing on the mushroom	a= attached, d = descending, f = free, n = notched
gill-spacing	The gap of space between each gill	c = close, w = crowded, d = distant
gill-size	The size of the gills	b = broad, n = narrow
gill-color	The color of the gills	k = black, n = brown, b = buff, h = chocolate, g = gray, r = green, o = orange, p = pink, u = purple, e = red, w = white , y = yellow
stalk-shape	The Stalk`s form	e = enlarging, t = tapering
stalk-root	The root of the mushroom	b = bulbous, c = club, u = cup, e = equal, z = rhizomorphs, r = rooted, ? = missing
stalk-surface-above-ring	The surface of the stalk above the mushrooms ring	f = fibrous, y = scaly, k = silky, s = smooth
stalk-surface-below-ring	The surface of the stalk below the mushrooms ring	f = fibrous, y = scaly, k = silky, s = smooth

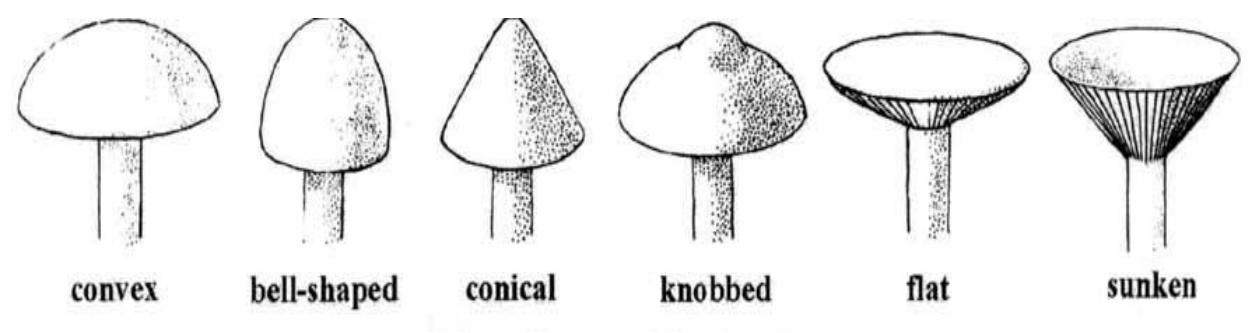
stalk-color-above-ring	The color of the stalk above the mushrooms ring	n = brown, b = buff, c = cinnamon, g = grey, o = orange, p = pink, e = red, w = white, y = yellow
stalk-color-below-ring	The color of the stalk below the mushrooms ring	n = brown, b = buff, c = cinnamon, g = grey, o = orange, p = pink, e = red, w = white, y = yellow
veil-type	The type of the mushroom`s veil	p = partial, u = universal
veil-color	The color of the mushroom`s veil	n = brown, o = orange, w = white, y = yellow
ring-number	The amount fo rings the mushroom has	n= none, o = one, t = two
ring-type	The type of the mushroom`s ring	c = cobwebby, e = evanescent, f = flaring, l = large ,n = none, p = pendant, s = sheathing, z = zone
spore-print-color	The color of the mushroom`s spore	k = black, n = brown, b = buff, h = chocolate, r = green, o = orange, u = purple, w = white , y = yellow
population	The population spread	a = abundant, c = clustered, n = numerous, s = scattered, v = several, y = solitary
habitat	The mushroom`s environment	g = grasses, l = leaves, m = meadows, p = paths, u = urban, w = waste, d = woods

Features

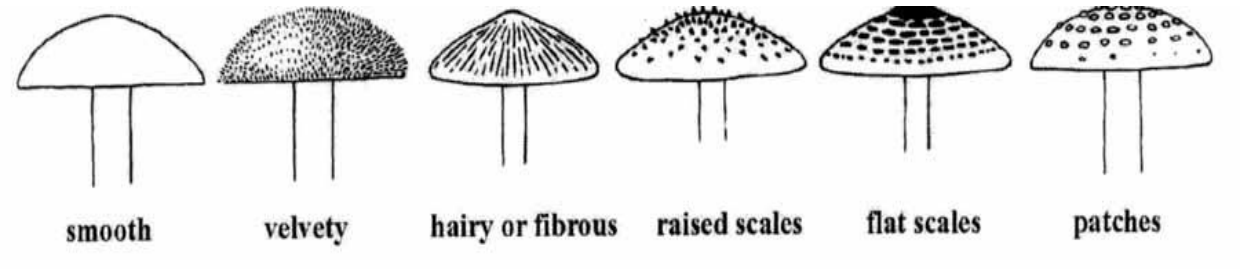
- Mushroom Structure:



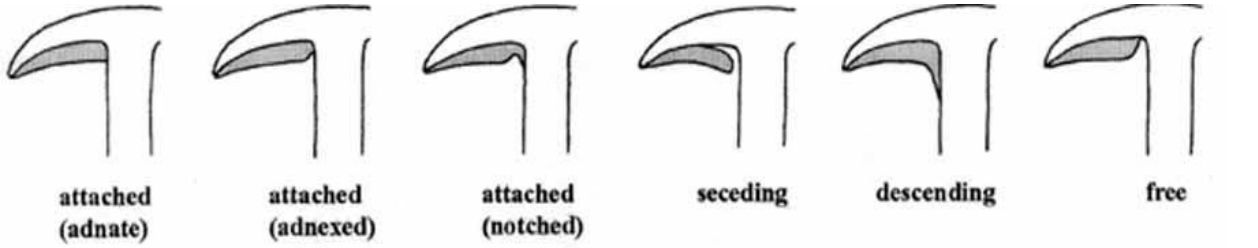
- Mushroom cap shape:



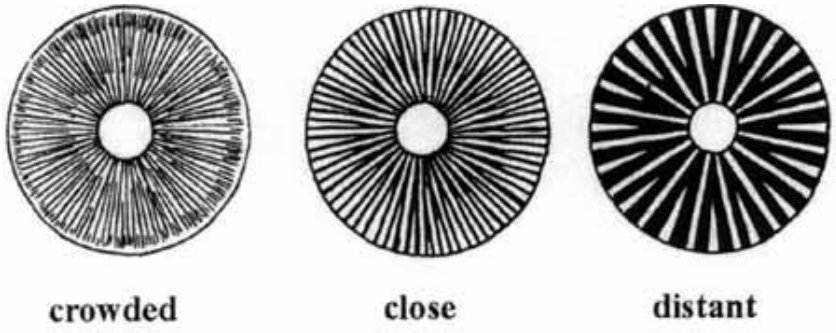
- Mushroom cap surface:



- Mushroom gill attachment:

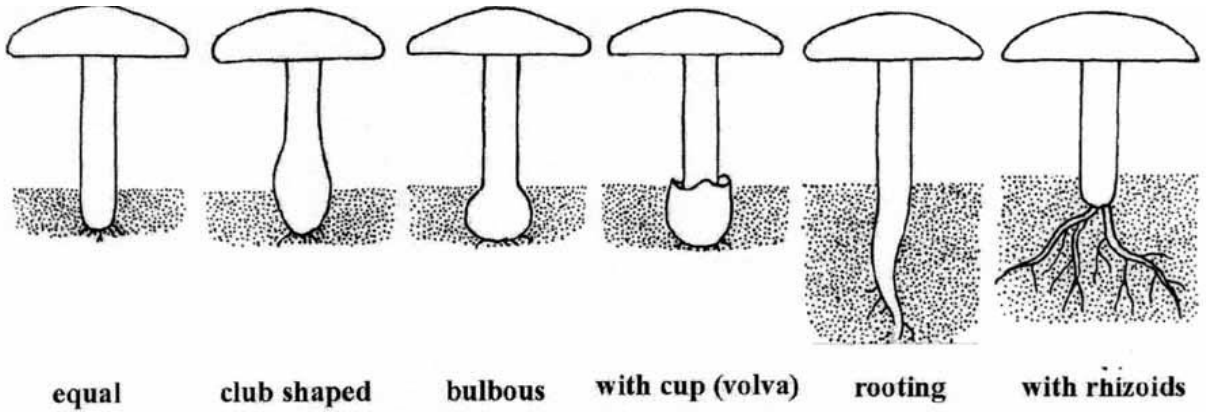


- Mushroom gill spacing:

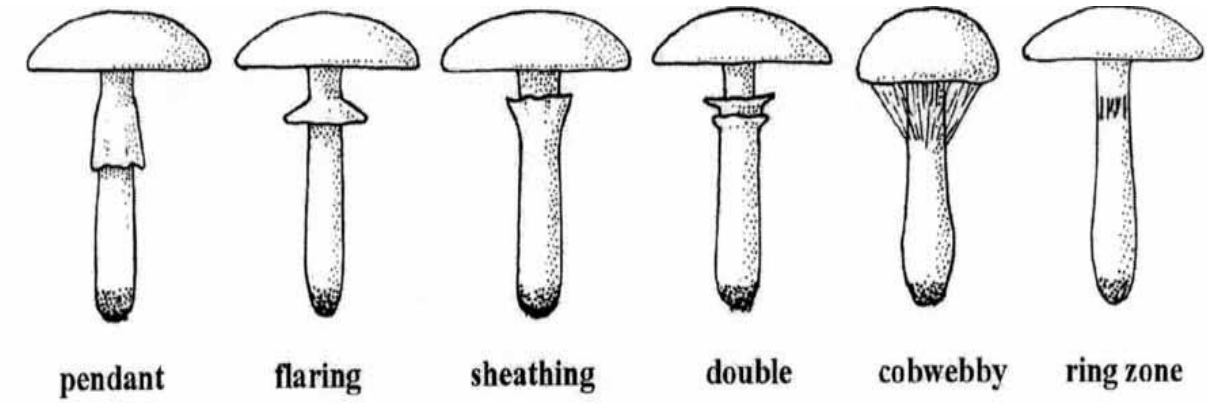


- Mushroom gill tissue arrangement:

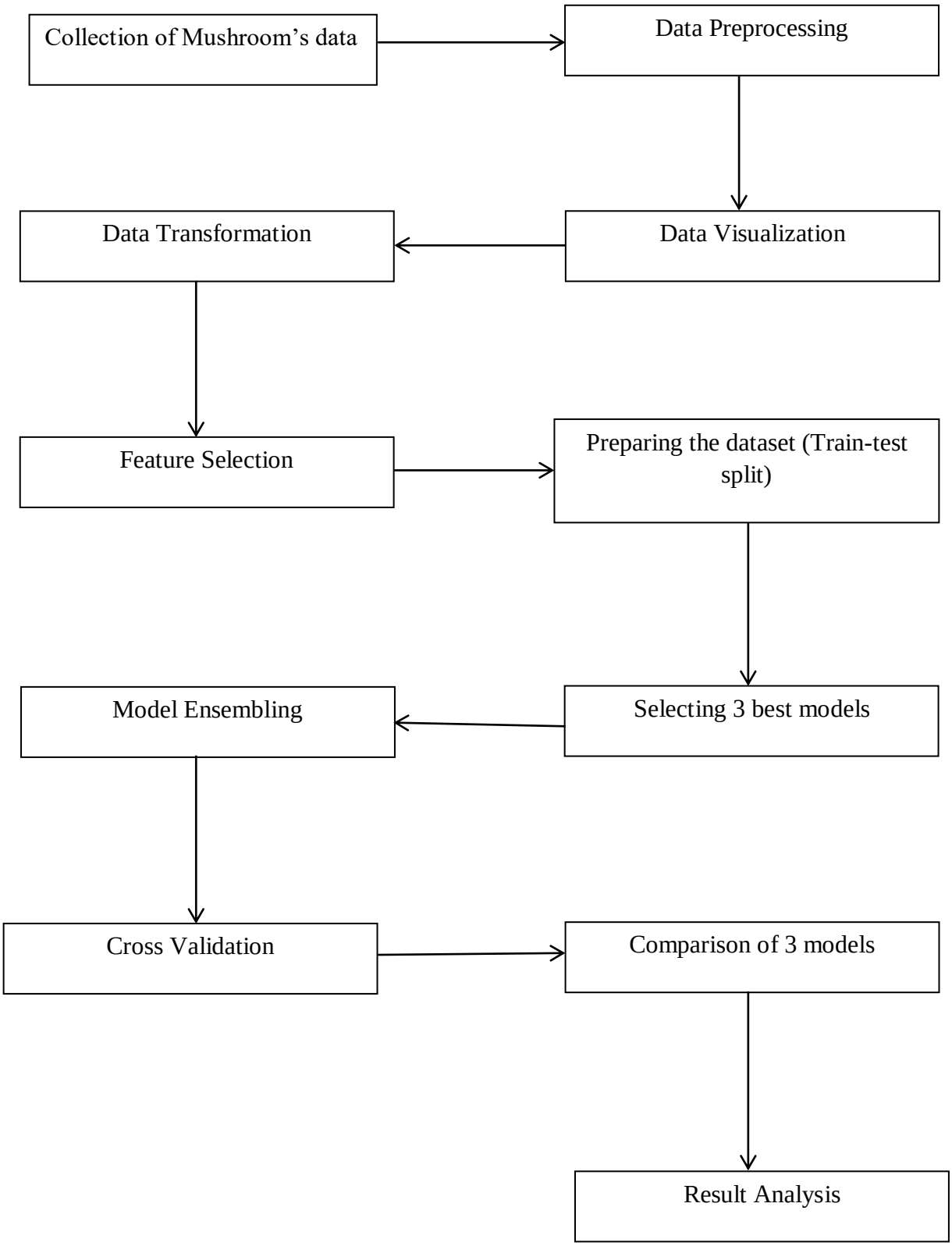
- Mushroom stalk type:



- Mushroom ring type:



METHODOLOGY



Importing the libraries:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, accuracy_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

Loading the dataset:

```
df = pd.read_csv("mushrooms.csv")
df.head()
```

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

5 rows × 23 columns

Examining the data:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   class                     8124 non-null   object
 1   cap-shape                 8124 non-null   object
 2   cap-surface               8124 non-null   object
 3   cap-color                 8124 non-null   object
 4   bruises                   8124 non-null   object
 5   odor                      8124 non-null   object
 6   gill-attachment           8124 non-null   object
 7   gill-spacing              8124 non-null   object
 8   gill-size                 8124 non-null   object
 9   gill-color                8124 non-null   object
 10  stalk-shape               8124 non-null   object
 11  stalk-root                8124 non-null   object
 12  stalk-surface-above-ring  8124 non-null   object
 13  stalk-surface-below-ring  8124 non-null   object
 14  stalk-color-above-ring    8124 non-null   object
 15  stalk-color-below-ring    8124 non-null   object
 16  veil-type                 8124 non-null   object
 17  veil-color                8124 non-null   object
 18  ring-number               8124 non-null   object
 19  ring-type                 8124 non-null   object
 20  spore-print-color         8124 non-null   object
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
```

Checking Null values:

```
df.isnull().sum()
```

```
class                       0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                  0
stalk-shape                 0
stalk-root                  0
stalk-surface-above-ring    0
stalk-surface-below-ring    0
stalk-color-above-ring      0
stalk-color-below-ring      0
veil-type                   0
veil-color                  0
ring-number                 0
ring-type                   0
spore-print-color           0
population                  0
habitat                     0
dtype: int64
```

Data Visualization:

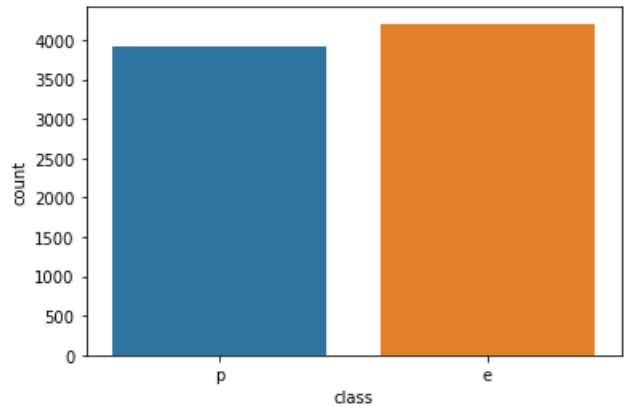
The process of finding trends and correlations in our data by representing it pictorially is called Data Visualization. To perform data visualization in python, we can use various python data visualization modules such as Matplotlib, Seaborn, Plotly, etc.

Target Plot

```
edible = df[df['class'] == 'e']['class'].count()
poisionous = df[df['class'] == 'p']['class'].count()
print("poisionous count: {} \nedible count: {}".format(poisionous,edible))
sns.countplot(df['class'])
```

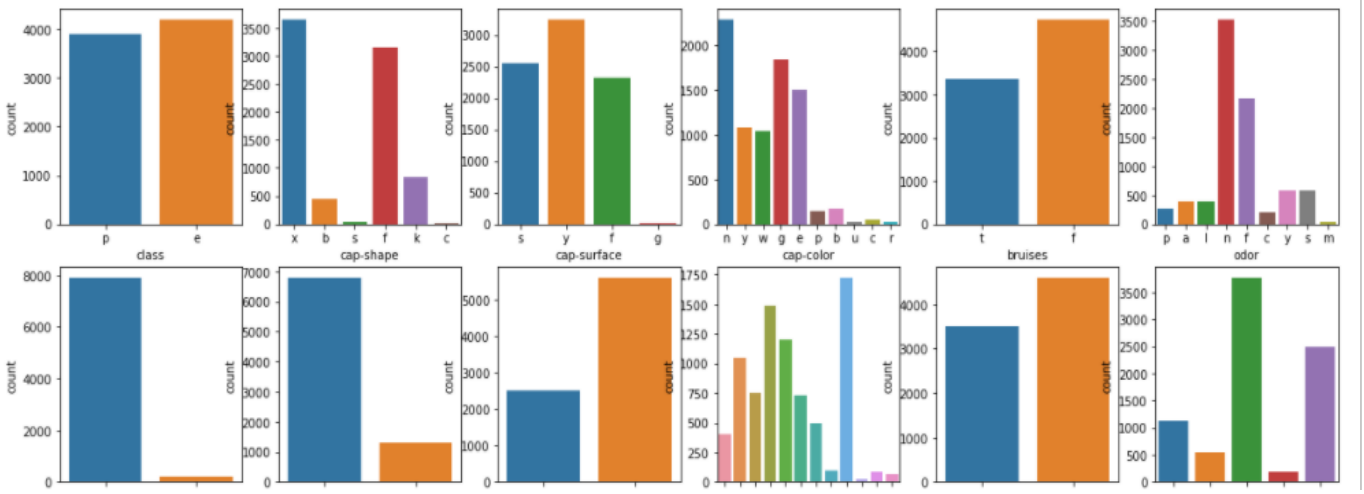
```
poisionous count: 3916
edible count: 4208

<matplotlib.axes._subplots.AxesSubplot at 0x1bce30b7f70>
```



Unvariate Plots:

```
plt.figure(figsize=(20,15))
plotNum=1
for column in df:
    if plotNum<=23:
        ax=plt.subplot(4,6,plotNum)
        sns.countplot(df[column])
        plt.xlabel(column,fontsize=10)
    plotNum+=1
plt.show()
```



Data Manipulation:

The data is categorical so we'll use LabelEncoder to convert it to ordinal. LabelEncoder converts each value in a column to a number.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df = df.apply(LabelEncoder().fit_transform)
df.head()
```

lass	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
1	5	2	4	1	6	1	0	1	4	...	2	7	7	0	2	1	4	2	3	5
0	5	2	9	1	0	1	0	0	4	...	2	7	7	0	2	1	4	3	2	1
0	0	2	8	1	3	1	0	0	5	...	2	7	7	0	2	1	4	3	2	3
1	5	3	8	1	6	1	0	1	5	...	2	7	7	0	2	1	4	2	3	5
0	5	2	3	0	5	1	1	0	4	...	2	7	7	0	2	1	0	3	0	1

's x 23 columns

From the above data we can take class as independent variable and all others are dependent variables. So, we are separating class column with the rest.

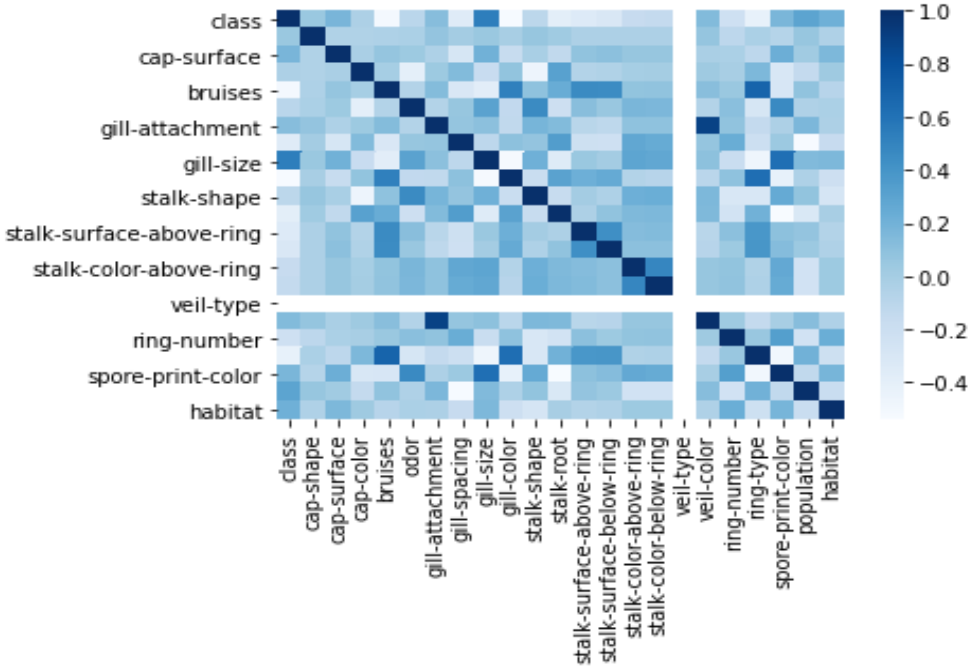
```
x = df.drop('class', axis=1)
y = df['class']
```

Correlation:

Correlation explains how one or more variables are related to each other. These variables can be input data features which have been used to forecast our target variable. Correlation, statistical technique which determines how one variables moves/changes in relation with the other variable. It gives us the idea about the degree of the relationship of the two variables.

```
sns.heatmap(df.corr(),cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e1ae4292b0>
```



Making DataFrame of the correlated values:

```
corr_data = pd.DataFrame({'correlation': corr}, index=x.columns)
```

```
corr_data
```

	correlation
cap-shape	0.052951
cap-surface	0.178446
cap-color	0.031384
bruises	0.501530
odor	0.093552
gill-attachment	0.129200
gill-spacing	0.348387
gill-size	0.540024
gill-color	0.530566
stalk-shape	0.102019
stalk-root	0.379361
stalk-surface-above-ring	0.334593
stalk-surface-below-ring	0.298801
stalk-color-above-ring	0.154003
stalk-color-below-ring	0.146730

Selecting features with high correlation (>=0.2):

```
corr_data = corr_data.sort_values(by = 'correlation', ascending=False)
```

```
corr_imp = corr_data[corr_data['correlation'] >= 0.2]
```

```
corr_imp
```

	correlation
gill-size	0.540024
gill-color	0.530566
bruises	0.501530
ring-type	0.411771
stalk-root	0.379361
gill-spacing	0.348387
stalk-surface-above-ring	0.334593
stalk-surface-below-ring	0.298801
population	0.298686
habitat	0.217179
ring-number	0.214366

Train-test split:

```
x_train , x_test , y_train , y_test = train_test_split(corr_X,y,test_size = 0.3, random_state = 0)
```

CLASSIFICATION METHODS

1. **Logistic Regression:**

Logistic regression, despite its name, is a classification model rather than regression model. Logistic regression is a simple and more efficient method for binary and linear classification problems. It is a classification model, which is very easy to realize and achieves very good performance with linearly separable classes. It is an extensively employed algorithm for classification in industry. Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

```
accuracies = list()
lr = LogisticRegression(max_iter=1000)
lr.fit(x_train,y_train)
lr_pred = lr.predict(x_test)
print("Mean-Squared-Error:", mean_squared_error(y_test,lr_pred)*100,"%")
print("Accuracy:",accuracy_score(y_test,lr_pred)*100,"%")
alt = ['Logistic Regression', accuracy_score(y_test,lr_pred), mean_squared_error(y_test,lr_pred)]
accuracies.append(alt)
```

```
Mean-Squared-Error: 6.234618539786711 %
Accuracy: 93.76538146021329 %
```

2. Support Vector Machine (SVM):

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

```
svc_ = SVC()
svc_.fit(x_train,y_train)
svm_pred = svc_.predict(x_test)
print("Mean-Squared-Error:", mean_squared_error(y_test,svm_pred)*100,"%")
print("Accuracy:",accuracy_score(y_test,svm_pred)*100,"%")
alt2 = ['Support Vector Machine', accuracy_score(y_test,svm_pred), mean_squared_error(y_test,svm_pred)]
accuracies.append(alt2)
```

```
Mean-Squared-Error: 2.3789991796554553 %
Accuracy: 97.62100082034455 %
```

3. Random Forest Classifier:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

.

```
rfc = RandomForestClassifier(max_depth = 5)
rfc.fit(x_train , y_train)
rfc_pred = rfc.predict(x_test)
print("Mean Square Error:", mean_squared_error(y_test,rfc_pred)*100,"%")
print("Accuracy:",accuracy_score(y_test,rfc_pred)*100,"%")
alt3 = ['Random Forest Classifier', accuracy_score(y_test,rfc_pred), mean_squared_error(y_test,rfc_pred)]
accuracies.append(alt3)
```

```
Mean Square Error: 1.1484823625922889 %
Accuracy: 98.85151763740771 %
```

Comparing the three models:

1. Logistic Regression - 93.76%

2. Support Vector Machine - 97.62%

3. Random Forest Accuracy - 98.60%

Hence, Ramdom Forest is comparitively better

```
accuracy_df  = pd.DataFrame(list(accuracies),columns = ['Model Name', 'Accuracy Score', 'Mean-Squared Error'])
accuracy_df
```

	Model Name	Accuracy Score	Mean-Squared Error
0	Logistic Regression	0.937654	0.062346
1	Support Vector Machine	0.976210	0.023790
2	Random Forest Classifier	0.988515	0.011485

ENSEMBLE MODEL

Ensemble methods are techniques that create multiple models and then combine them to produce improved results. Ensemble methods usually produce more accurate solutions than a single model would.

Max-Voting:

```
model1 = LogisticRegression(max_iter=1000)
model2 = SVC(probability=True)
model3= RandomForestClassifier()
model = VotingClassifier(estimators=[('lr', model1), ('svc', model2), ('rfc',model3)])
model.fit(x_train,y_train)
ans = model.score(x_test,y_test)
print(ans*100,'%')
```

```
97.62100082034455 %
```

Hyperparameter Tuning:

Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.

```
params = {'voting': ['hard','soft'],
          'weights': [(1,1,1),(2,1,1),(1,2,1),(1,1,2)]
          }
```

```
grid = GridSearchCV(model,params)
grid.fit(x_train,y_train)
grid.best_params_
```

```
{'voting': 'soft', 'weights': (1, 1, 2)}
```

```
ans2 = grid.score(x_test,y_test)
print(ans2*100,'%')
```

```
100.0 %
```

With hyperparameter tuning the result increased from 97.62% to 100%

Bagging:

```
from sklearn.ensemble import BaggingClassifier
Model = BaggingClassifier(
    base_estimator = RandomForestClassifier(max_depth=5),
    n_estimators=100,
    max_samples=0.8,
    oob_score=True,
    random_state=0
)
Model.fit(x_train,y_train)
Model_pred= Model.predict(x_test)
print("Mean-Squared-Error:", mean_squared_error(y_test,Model_pred)*100,"%")
print("Accuracy:",accuracy_score(y_test,Model_pred)*100,"%")
```

```
Mean-Squared-Error: 1.3125512715340444 %
Accuracy: 98.68744872846595 %
```

Finding the best set of parameters:

```
model_params = {
    'svm': {
        'model': SVC(gamma='auto'),
        'params' : {
            'C': [1,10,20],
            'kernel': ['rbf','linear']
        }
    },
    'random_forest': {
        'model': RandomForestClassifier(),
        'params' : {
            'n_estimators': [1,5,10]
        }
    },
    'logistic_regression' : {
        'model': LogisticRegression(solver='liblinear',multi_class='auto'),
        'params': {
            'C': [1,5,10]
        }
    }
}
```

```
scores = []

for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=5, return_train_score=False)
    clf.fit(x_train,y_train)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_
    })

data = pd.DataFrame(scores,columns=['model','best_score','best_params'])
data
```

	model	best_score	best_params
0	svm	1.000000	{'C': 10, 'kernel': 'rbf'}
1	random_forest	1.000000	{'n_estimators': 5}
2	logistic_regression	0.936511	{'C': 5}

K-FOLD CROSS –VALIDATION

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into.

As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

Splitting the data into folds:

```
kf = KFold(n_splits=5)
kf
```

```
KFold(n_splits=5, random_state=None, shuffle=False)
```

```
for train_index, test_index in kf.split(df):
    print(train_index, test_index)
```

```
[1625 1626 1627 ... 8121 8122 8123] [   0    1    2 ... 1622 1623 1624]
[   0    1    2 ... 8121 8122 8123] [1625 1626 1627 ... 3247 3248 3249]
[   0    1    2 ... 8121 8122 8123] [3250 3251 3252 ... 4872 4873 4874]
[   0    1    2 ... 8121 8122 8123] [4875 4876 4877 ... 6497 6498 6499]
[   0    1    2 ... 6497 6498 6499] [6500 6501 6502 ... 8121 8122 8123]
```

Logistic regression model performance using cross_val_score

```
cross_val_score(LogisticRegression(solver='liblinear',multi_class='ovr'),x_train,y_train,cv=kf)
```

```
array([0.93760984, 0.93667546, 0.93755497, 0.93227792, 0.92875989])
```

```
score = cross_val_score(LogisticRegression(solver='liblinear',multi_class='ovr'),x_train,y_train,cv=kf)
print("Average: ",np.average(score))
```

```
Average:  0.9345756183215782
```

random forest performance using cross_val_score

```
cross_val_score(RandomForestClassifier(n_estimators=50),x_train,y_train,cv=kf)
```

```
array([1., 1., 1., 1., 1.])
```

```
score = cross_val_score(RandomForestClassifier(n_estimators=50),x_train,y_train,cv=kf)
print("Average: ",np.average(score))
```

```
Average:  1.0
```

SVM model performance using cross_val_score

```
cross_val_score(SVC(gamma='auto'), x_train,y_train,cv=kf)
```

```
array([0.99297012, 0.99912049, 0.99384345, 0.99560246, 0.99384345])
```

```
score = cross_val_score(SVC(gamma='auto'), x_train,y_train,cv=kf)
print("Average: ",np.average(score))
```

```
Average:  0.9950759947013152
```

Random forest classifier gives the best result

CONCLUSION

Machine Learning is a hypnotizing point, it empowers machines to make sense of how to accomplish errands that irrefutably required a man to do. While machine learning used to require the most serious supercomputers just a few years earlier, the section of circulated registering, more affordable CPUs, and much better estimations allowed Machine Learning to end up open more broadly.

In general, various machine learning techniques are used to analyse the features obtained from the datasets. So, in another way we can say that a machine learning is a system or model that takes the documents, analyse the input, and learns automatically with the involvement of any human. They have the ability to understand how to adjust the methods of learning according to some actions.

After running the various learning models on our dataset to determine poisonousness of mushrooms, we learn that while all the learning models are fairly decent, some exceptionally good, the default decision tree model wins out in accuracy and reliability.

Using dataset, we can conclude with 100% certainty that a mushroom is edible or poisonous. These are the 23 features with their correlation with the target, class. A negative correlation means if a mushroom has that feature it is more likely to be edible. A positive correlation means if a mushroom has that feature it is more likely to be poisonous.

According to the analysis of the dataset features, the accuracy of logistic regression is 93.76%. The accuracy of Support Vector machine is 97.62% while the accuracy of Random forest classifier is 98.60%. Hence, random forest classifier gives a better result compared to other models.

At present, research on toxins in poisonous mushrooms is still underway, and cases of mushroom poisoning still occur. Therefore, it is necessary to establish an automatic model for the appearance feature recognition of mushrooms' toxicity. Not only that, this comparative study has allowed us to gain greater insight into the inner workings of the major machine learning models and how they play out in real world situations. The conclusion can be scaled and applied to countless other problems of the same nature and happens to be a valuable tool for any data scientist.

REFRENCES

1. Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]

2. Eusebi, C., Gliga, C., John, D. and Maisonave, A., (2008). Data Mining on Mushroom Database. Journal of CSIS, Pace University, pp.1-9

3. Z. Chaoqun, Recognition and Research of Poisonous Mushroom Based on Machine Learning, Taigu: Shanxi Agricultural University, Jinzhong, China, 2019.

4. Husaini, M., (2018). A Data Mining Based On Ensemble Classifier Classification Approach for Edible Mushroom Identification.

5. L. Breiman, “Random forests,” Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.

6. Classification of Mushrooms to Detect their Edibility Based on Key Attributes, Biosc.Biotech.Res.Comm. Special Issue vol. 13 No 11 (2020) Pp-37-41

7. Pinky, N.J., Islam, S.M. and Rafia, S.A., (2019). Classification Edibility Detection of Mushroom Using Ensemble Methods, p.55.