**MUSHROOM PROJECT**

**Developing Machine Learning Model for predicting edibility of mushroom**

Table of Contents

[Introduction 2](#_Toc65419184)

[The Problem Statement 2](#_Toc65419185)

[About Dataset 2](#_Toc65419186)

[Load Dataset 2](#_Toc65419187)

[Data Analysis 3](#_Toc65419188)

[Renaming Columns 3](#_Toc65419189)

[Checking General Information 3](#_Toc65419190)

[Label Encoding 5](#_Toc65419191)

[Exploratory Data Analysis (EDA) and Visualisation Concluding Remarks 5](#_Toc65419192)

[Univariate Analysis 6](#_Toc65419193)

[Bivariate Analysis 7](#_Toc65419194)

[Multi Variate Analysis 8](#_Toc65419195)

[Correlation of Features 8](#_Toc65419196)

[Representation of Correlation of feature w.r.t. Target 9](#_Toc65419197)

[Prepare Dataset for Model Training 9](#_Toc65419198)

[Outlier(s): Detection & Removal 10](#_Toc65419199)

[Separate Input and Output/Target Variable 10](#_Toc65419200)

[Skewness: Detection & Treatment 10](#_Toc65419201)

[Scale Data for Model Training 11](#_Toc65419202)

[Model Training: Finding the best model 11](#_Toc65419203)

[Define the function(s) for getting best random\_state and best CV 11](#_Toc65419204)

[Prepare Model List and Test to get Best Model 12](#_Toc65419205)

[1. LogisticRegression 12](#_Toc65419206)

[2. GaussianNB 13](#_Toc65419207)

[3. RandomForestClassifier 14](#_Toc65419208)

[Compare Model Performances 15](#_Toc65419209)

[Model Selection: The Final Model 15](#_Toc65419210)

[Conclusion 15](#_Toc65419211)

[Model Prediction 16](#_Toc65419212)

[Pre-Processing Pipeline 16](#_Toc65419213)

[Load Model from Serialized Object/Pickle file 16](#_Toc65419214)

[Save Predicted Values 16](#_Toc65419215)

[Concluding Remarks 16](#_Toc65419216)

# Introduction

Mushrooms are free of cholesterol and contain small amounts of essential amino acids and B vitamins. However, their chief worth is as a specialty food of delicate, subtle flavour and agreeable texture. By fresh weight, the common commercially grown mushroom is more than 90 percent water, less than 3 percent protein, less than 5 percent carbohydrate, less than 1 percent fat, and about 1 percent mineral salts and vitamins. But this doesn’t mean that every type of mushrooms can be consumed because there are many variety or species of mushrooms present out of which some are edible and some are poisonous in nature. Therefore, it is important that every mushroom intended for eating be accurately identified. Here in this article, we are going do the same by developing a machine learning model which can identify whether a mushroom is edible or poisonous.

# The Problem Statement

To identify each species of mushroom as definitely edible, definitely poisonous, or of unknown edibility and not recommended which can also be termed as poisonous.

# About Dataset

This data set includes descriptions of hypothetical samples corresponding to 23 species with 8124 records of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like ``leaflets three, let it be'' for Poisonous Oak and Ivy.

**Feature Details:**

1. class: edible=e, poisonous=p
2. cap-shape: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
3. cap-surface: fibrous=f, grooves=g, scaly=y, smooth=s
4. cap-color: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y
5. bruises: bruises=t, no=f
6. odor: almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s
7. gill-attachment: attached=a, descending=d, free=f, notched=n
8. gill-spacing: close=c, crowded=w, distant=d
9. gill-size: broad=b, narrow=n
10. gill-color: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y
11. stalk-shape: enlarging=e, tapering=t
12. stalk-root: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?
13. stalk-surface-above-ring: fibrous=f, scaly=y, silky=k, smooth=s
14. stalk-surface-below-ring: fibrous=f, scaly=y, silky=k, smooth=s
15. stalk-color-above-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
16. stalk-color-below-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
17. veil-type: partial=p, universal=u
18. veil-color: brown=n, orange=o, white=w, yellow=y
19. ring-number: none=n, one=o, two=t
20. ring-type: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z
21. spore-print-color: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
22. population: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
23. habitat: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

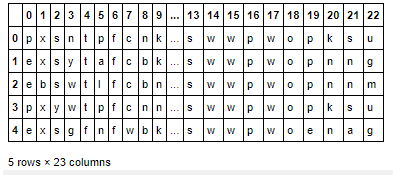
**Dataset Link:** <https://github.com/dsrscientist/dataset1/blob/master/mushrooms.csv>

# Load Dataset

**import** **pandas** **as** **pd**

df\_mushroom = pd.read\_csv('mushroom\_data.csv',header=**None**)

df\_mushroom.head()



After giving a close look, we found that all the features are in categorical form and the column index ranges from 0 to 22. For ease of understanding we will replace all the column index with named index and then will move further with the data analysis. Also, the target value is of categorical type, therefore, it is a classification problem.

# Data Analysis

In this part, we will first prepare the data by renaming column index, performing type conversion, detecting & removing null values and encode the categorical features for further processing.

## Renaming Columns

*#Copy data set to new df and renaming it's columns*

headings = {0:"class",1:"cap-shape",2:"cap-surface",3:"cap-color",4:"bruises",5:"odor",6:"gill-attachment",7:"gill-spacing",

8:"gill-size",9:"gill-color",10:"stalk-shape",11:"stalk-root",12:"stalk-surface-above-ring",

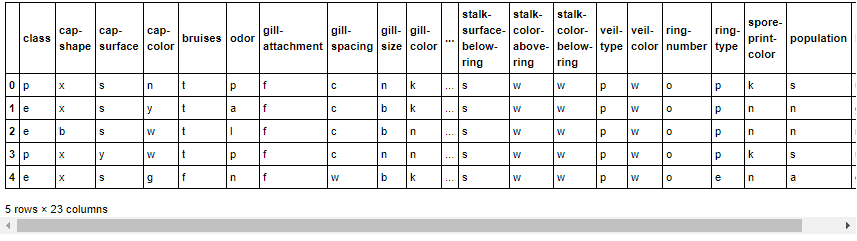
13:"stalk-surface-below-ring",14:"stalk-color-above-ring",15:"stalk-color-below-ring",16:"veil-type",

17:"veil-color",18:"ring-number",19:"ring-type",20:"spore-print-color",21:"population",22:"habitat"}

df = df\_mushroom.copy()

df.rename(columns=headings,inplace=**True**)

df.head() *#Printing first 5 rows*



As we can see that we have successfully renamed the column index from 0 to 22 with named index as accordance with the feature details.

## Checking General Information

*#Checking general information of dataset*

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

dtypes: object(23)

memory usage: 1.4+ MB

From the above information, it is clear that all the features are of discrete (object) type and hence encoding is required to proceed further with the analysis but before that we will need to check them for the unique values, so that we can decide which encoding technique and features needs to considered. Also, from above, it clearly shows that there are no null values present in dataset.

*#Checking unique values of dataset before encoding*

**for** x **in** headings.values():

print("**%-30s**: **%8d**"%(x,len(df[x].unique())))

class : 2

cap-shape : 6

cap-surface : 4

cap-color : 10

bruises : 2

odor : 9

gill-attachment : 2

gill-spacing : 2

gill-size : 2

gill-color : 12

stalk-shape : 2

stalk-root : 5

stalk-surface-above-ring : 4

stalk-surface-below-ring : 4

stalk-color-above-ring : 9

stalk-color-below-ring : 9

veil-type : 1

veil-color : 4

ring-number : 3

ring-type : 5

spore-print-color : 9

population : 6

habitat : 7

From above we can see that except feature veil-type, all other have 2 or more unique values, therefore, we can use either get\_dummies from pandas or LabelEncoder from sklearn.preprocessing to encode the features. Also, in feature veil-type only single value is present, we can consider dropping this feature from dataset as it is of no use. Now, we will move further with label encoding using LabelEncoder.

## Label Encoding

To deal with object types in the data, label encoding is used to transform these features into numerical form which can be provided to the machine learning models for training. Here, we are using LabelEncoder, which encodes categorical features with value between 0 and n\_classes-1, to transform these features into numerical form.

**from** **sklearn.preprocessing** **import** LabelEncoder

le = {}

encoded\_values = {}

**for** i **in** headings.values():

le[i] = LabelEncoder()

encoded\_values[i] = le[i].fit\_transform(df[i])

print(f"%-30s: **{**le[i].classes\_**}**"%(i))

class : ['e' 'p']

cap-shape : ['b' 'c' 'f' 'k' 's' 'x']

cap-surface : ['f' 'g' 's' 'y']

cap-color : ['b' 'c' 'e' 'g' 'n' 'p' 'r' 'u' 'w' 'y']

bruises : ['f' 't']

odor : ['a' 'c' 'f' 'l' 'm' 'n' 'p' 's' 'y']

gill-attachment : ['a' 'f']

gill-spacing : ['c' 'w']

gill-size : ['b' 'n']

gill-color : ['b' 'e' 'g' 'h' 'k' 'n' 'o' 'p' 'r' 'u' 'w' 'y']

stalk-shape : ['e' 't']

stalk-root : ['?' 'b' 'c' 'e' 'r']

stalk-surface-above-ring : ['f' 'k' 's' 'y']

stalk-surface-below-ring : ['f' 'k' 's' 'y']

stalk-color-above-ring : ['b' 'c' 'e' 'g' 'n' 'o' 'p' 'w' 'y']

stalk-color-below-ring : ['b' 'c' 'e' 'g' 'n' 'o' 'p' 'w' 'y']

veil-type : ['p']

veil-color : ['n' 'o' 'w' 'y']

ring-number : ['n' 'o' 't']

ring-type : ['e' 'f' 'l' 'n' 'p']

spore-print-color : ['b' 'h' 'k' 'n' 'o' 'r' 'u' 'w' 'y']

population : ['a' 'c' 'n' 's' 'v' 'y']

habitat : ['d' 'g' 'l' 'm' 'p' 'u' 'w']

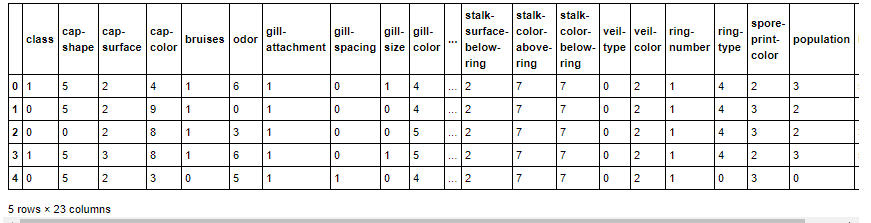
*#Replacing the actual values of columns with encoded values*

**for** x **in** headings.values():

df[x] = encoded\_values[x]

*#Printing first 5 rows of encoded dataset*

df.head()



After apply label encoding, all the categorical features have been transformed to numeric (continuous) type. Now moving further with Exploratory Data Analysis to get more information about the dataset.

# Exploratory Data Analysis (EDA) and Visualisation Concluding Remarks

Exploratory Data Analysis (EDA) is an approach to analysing data sets to summarize their main characteristics, often using statistical graphics and other data visualization methods. This can divided into three parts namely Univariate, Bivariate and Multivariate Analysis.

**Importing visualisation libraries**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

## Univariate Analysis

Univariate Analysis means analysis of one variable or one feature at a time and it basically tells us how data in each feature is distributed and also tells us about central tendencies like mean, median, and mode as well as presence of outliers in the dataset.

**Data Distribution using distplot:** Distribution of data basically tells us about the mean, median, mode, maximum, minimum, standard deviation and skewness of data. We can get and visualize them using distribution plot of searborn library as shown below:

**import** **warnings**

warnings.simplefilter('ignore')

*#Checking data distribution using distplot.*

cols = 4

rows = 6

fig, axes = plt.subplots(rows,cols,figsize=(rows\*cols,rows\*cols))

plt.subplots\_adjust(hspace=0.5)

k=0

**for** i **in** range(rows):

**for** j **in** range(cols):

sns.distplot(df[df.columns[k]],ax=axes[i][j])

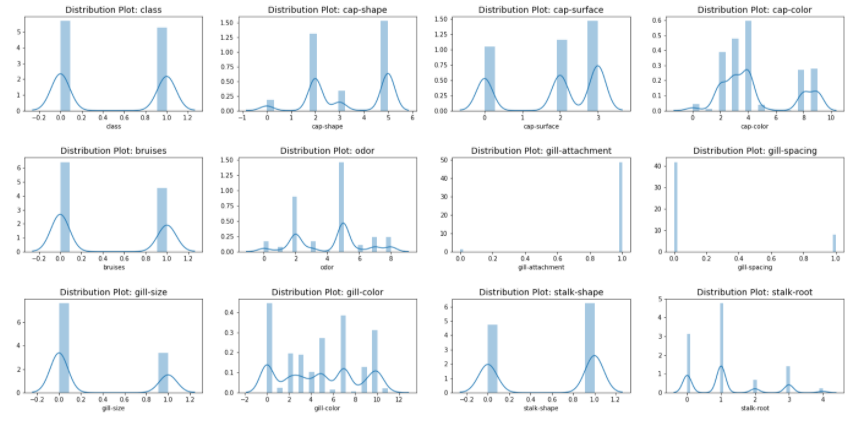
axes[i][j].set\_title(f"Distribution Plot: **{**df.columns[k]**}**", fontsize=14)

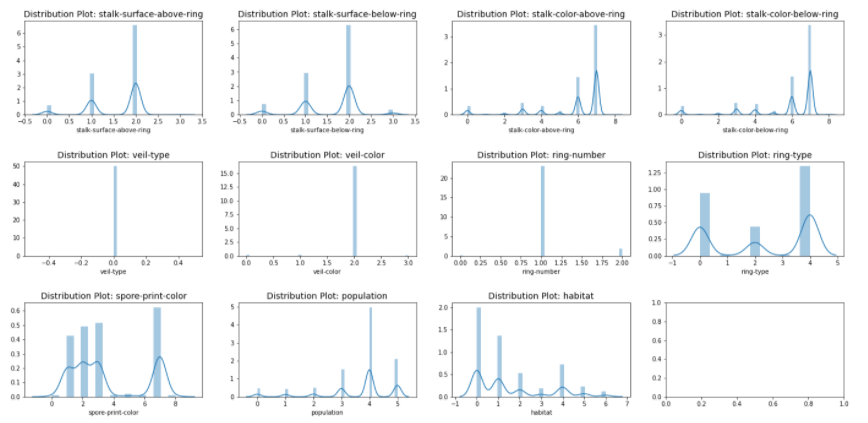
k += 1

**if** k == 23:

**break**;

plt.show()





We can see that distribution plot shows, the data is not distributed normally in all of the features and this is understandable because all of the features are of categorical type. Now moving further with Bivariate analysis to check the relationship between features.

## Bivariate Analysis

Bivariate analysis is one of the simplest forms of quantitative analysis. It involves the analysis of two variables (namely X and Y), for the purpose of determining the empirical relationship between them. Therefore, checking the relationship of feature RATING and VOTES with target variable COST to get more information on it.

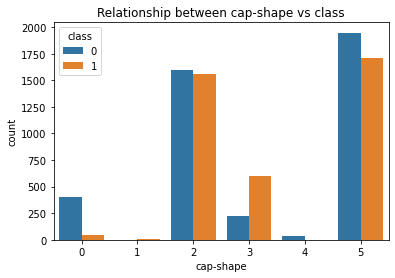
*#Checking Relationship between cap-shape and class*

f='cap-shape'

sns.countplot(x=f,hue='class',data=df)

plt.title(f"Relationship between **{**f**}** vs class")

plt.show()



From above graph we can state that most of the mushroom belongs to cap-shape 2 and 5. Further it is clear that mushroom with cap-shape 0 and 4 are more edible in nature while mushroom with cap-shape 1 and 3 are more poisonous in nature.

For all other features we can take the observations by following same method and conclude the remarks on the basis of that.

Now moving further with multi variate analysis to get the complete inference of data and their relationship with each other.

## Multi Variate Analysis

Multivariate analysis refers to any statistical technique used to analyse more complex sets of data.

### Correlation of Features

*#Checking correlation of feature*

df\_corr = df.corr()

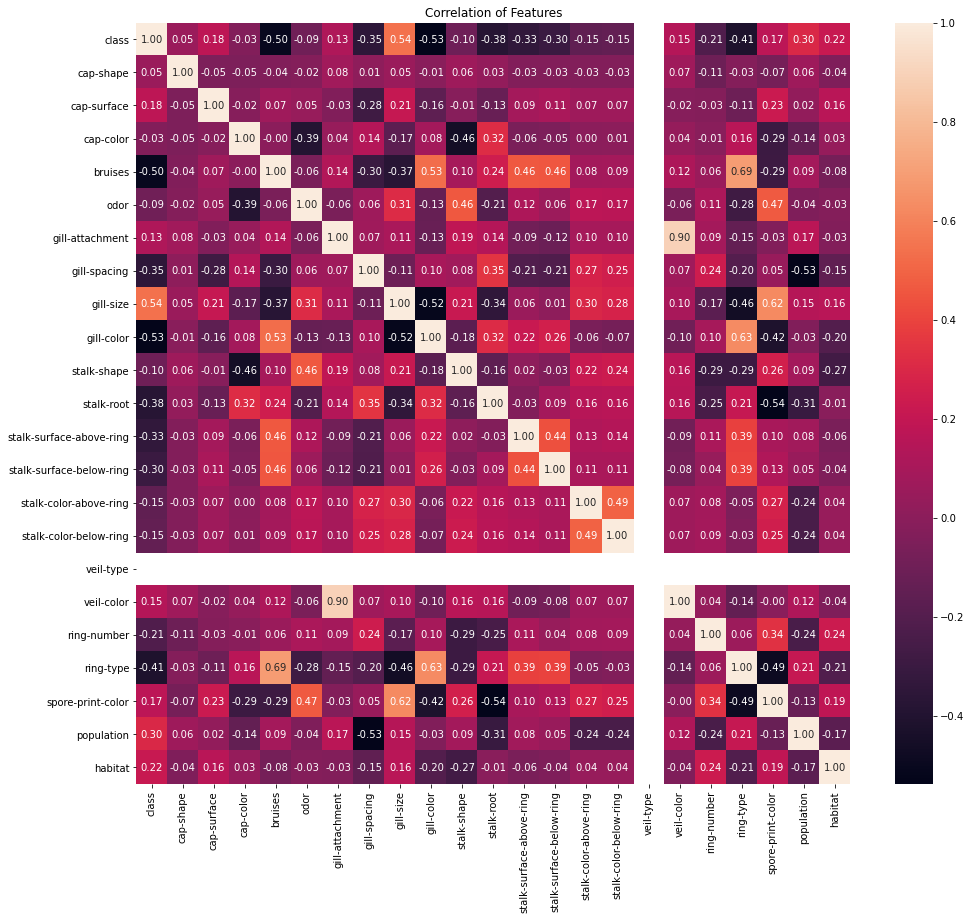
*#Displaying correlation with the help of heatmap*

plt.figure(figsize=(16,14))

sns.heatmap(df\_corr,annot=**True**,fmt='.2f')

plt.title('Correlation of Features')

plt.show()



From the above correlation depiction by heatmap, we can clearly infer that:

* Feature gill-size and population is positively high correlated to target (class).
* Feature gill-color, bruises, ring-type, stalk-root, gill-spacing, stalk-surface-above-ring and stalk-surface-below-ring are negatively high correlated to target.
* Feature habitat, spore-print-color, veil-color, gill-attachment and cap-surface are positively correlated to target.
* Feature ring-number, stalk-color-above-ring and stalk-color-below-ring are negatively correlated to target.
* Rest Features are weakly correlated to target.
* Also, feature veil-type is not showing any correlation with target.

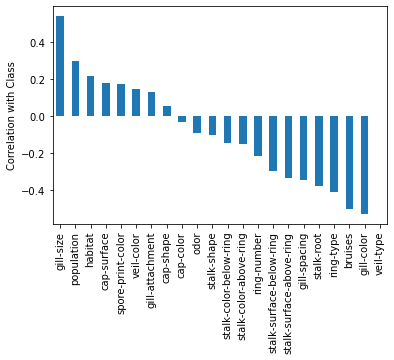
### Representation of Correlation of feature w.r.t. Target

*#Ploting correlation with features with bar plot*

df\_corr['class'].sort\_values(ascending=**False**).drop(['class']).plot.bar()

plt.ylabel('Correlation with Class')

plt.show()



From the above depiction, it is clear that from gill-size to cap-shape correlation value decreases while from cap-color to gill-color correlation value increases negatively. Also, feature veil-type shows no correlation with the target therefore, it can be dropped from the dataset.

*#Dropping veil-type from dataset*

df.drop(columns=['veil-type'],inplace=**True**)

df.columns

Index(['class', 'cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',

'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',

'stalk-shape', 'stalk-root', 'stalk-surface-above-ring',

'stalk-surface-below-ring', 'stalk-color-above-ring',

'stalk-color-below-ring', 'veil-color', 'ring-number', 'ring-type',

'spore-print-color', 'population', 'habitat'],

dtype='object')

# Prepare Dataset for Model Training

Preparing dataset Model training is the one of core part of machine learning model building and includes different types data modification and transformation to achieve the better model performance.

**Importing Libraries:**

**from** **sklearn.preprocessing** **import** OrdinalEncoder, StandardScaler, power\_transform

**from** **scipy.stats** **import** zscore

## Outlier(s): Detection & Removal

Outliers are extreme values that fall a long way outside of the other observations. It can be detected and removed using either Z-Score or Interquartile Range (IQR) methods. Here we are going to use z-score for this purpose.

*#Checking and removing outliers using zscore (threshold value -3 to +3)*

z = np.abs(zscore(df))

*#Printing location of outliers*

np.where(z>3)

(array([1816, 2128, 2128, ..., 8121, 8123, 8123], dtype=int64),

array([15, 14, 15, ..., 16, 6, 16], dtype=int64))

*#Removing outliers*

df\_wo = df[(z<=3).all(axis=1)]

print(f"Original Shape: **{**df.shape**}**")

print(f"New Shape: **{**df\_wo.shape**}**")

print(f"% Loss: **{**(len(df)-len(df\_wo))\*100/len(df)**}**%")

Original Shape: (8124, 22)

New Shape: (6568, 22)

% Loss: 19.153126538650913%

From above, we can see that, with this method of outlier removal, more than 5% of the records has been removed therefore discarding the outlier removal and proceeding with original dataset.

## Separate Input and Output/Target Variable

Now, we can separate the features into input as X and output/target as Y to continue further with data preparation.

*#Seperating Input and Target data.*

x = df.drop(columns=['class'])

y = df['class']

print(x.shape)

print(y.shape)

(8124, 21)

(8124,)

After separating input and output variable we can see that the variable X has 8,124 records with 21 features while variable Y has 8,124 records with a single target feature.

## Skewness: Detection & Treatment

*#Checking and treating skewness in contineous features (optimum value -0.5 to +0.5)*

df.skew()

class 0.071946

cap-shape -0.247052

cap-surface -0.590859

cap-color 0.706965

bruises 0.342750

odor -0.080790

gill-attachment -5.977076

gill-spacing 1.840088

gill-size 0.825797

gill-color 0.061410

stalk-shape -0.271345

stalk-root 0.947852

stalk-surface-above-ring -1.098739

stalk-surface-below-ring -0.757703

stalk-color-above-ring -1.835434

stalk-color-below-ring -1.791593

veil-color -6.946944

ring-number 2.701657

ring-type -0.290018

spore-print-color 0.548426

population -1.413096

habitat 0.985548

dtype: float64

*#Treating Skewness of data*

x = power\_transform(x)

From the above, we can see that, the feature gill-attachment, gill-spacing, stalk-surface-above-ring, stalk-color-above-ring, stalk-color-below-ring, veil-color, ring-number and population are heavily skewed and has been treated using power\_transform to reduce the skewness which is needed for model training.

## Scale Data for Model Training

Scaling data for model training is refers to normalize the range of independent variables or features of data. Here we are using StandardScaler for this purpose:

*#Scaling Data for model using Standard Scaler*

**from** **sklearn.preprocessing** **import** StandardScaler

sc = StandardScaler()

x = sc.fit\_transform(x)

# Model Training: Finding the best model

The models that I have decided to train for this dataset are LogisticRegression, GaussianNB and RandomForestClassifier models. The goal here is to find the best hyper-tuned models for further processing.

**Importing Libraries:**

**from** **sklearn.model\_selection** **import** train\_test\_split, GridSearchCV, cross\_val\_score

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report

## Define the function(s) for getting best random\_state and best CV

*#Defining function for best random\_state*

**def** get\_best\_random\_state(r,model,x,y,test\_size=0.25):

best\_rState = 0

best\_aScore = 0

**for** i **in** range(r):

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=test\_size,random\_state=i)

model.fit(x\_train,y\_train)

predict\_y = model.predict(x\_test)

temp\_aScore = accuracy\_score(y\_test,predict\_y)

**if** temp\_aScore>best\_aScore:

best\_aScore = temp\_aScore

best\_rState = i

**return** best\_rState, best\_aScore

*#Defining function for best CV*

**def** get\_best\_cv(model,parameters,x\_train,y\_train):

best\_cv = 0

best\_cvScore = 0

**for** i **in** range(2,20):

gscv = GridSearchCV(model,parameters)

gscv.fit(x\_train,y\_train)

temp\_cvScore = cross\_val\_score(gscv.best\_estimator\_,x\_train,y\_train,cv=i).mean()

**if** temp\_cvScore>best\_cvScore:

best\_cvScore = temp\_cvScore

best\_cv = i

**return** best\_cv, best\_cvScore

## Prepare Model List and Test to get Best Model

This part includes the preparation of model list with parameters and then train and test with best parameter to get the best hyper-tuned model performances.

### 1. LogisticRegression

**from** **sklearn.linear\_model** **import** LogisticRegression

*#Finding best random\_state for train test split*

best\_rState, best\_aScore = get\_best\_random\_state(500,LogisticRegression(),x,y)

print(f"Best random\_state: **{**best\_rState**}** with best accuracy\_score: **{**best\_aScore**}**")

Best random\_state: 180 with best accuracy\_score: 0.9620876415558838

*#Spilting train test data with random\_state = 180*

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.25,random\_state=180)

*#Hypertuning Parameters*

parameters = {

"solver": ['sag','saga'],

"penalty": ['l2'],

"max\_iter": [50,100,150]

}

*#Finding best CV*

best\_cv, best\_cvScore = get\_best\_cv(LogisticRegression(),parameters,x\_train,y\_train)

print(f"Best CV: **{**best\_cv**}** with best cross\_val\_score: **{**best\_cvScore**}**")

Best CV: 2 with best cross\_val\_score: 0.9502699661960432

*#Builing Final LogisticRegression Model with Hypertuned Parameters*

gscv\_Lr = GridSearchCV(LogisticRegression(),parameters,cv=2)

gscv\_Lr.fit(x\_train,y\_train)

print(gscv\_Lr.best\_params\_)

{'max\_iter': 150, 'penalty': 'l2', 'solver': 'saga'}

*#Checking Final Performace of Model*

gscv\_Lr\_predict\_y = gscv\_Lr.best\_estimator\_.predict(x\_test)

gscv\_Lr\_aScore = accuracy\_score(y\_test,gscv\_Lr\_predict\_y)

gscv\_Lr\_cMatrix = confusion\_matrix(y\_test,gscv\_Lr\_predict\_y)

gscv\_Lr\_cReport = classification\_report(y\_test,gscv\_Lr\_predict\_y)

print(f"Accuracy Score: **{**gscv\_Lr\_aScore**}\n**")

print(f"========Confusion Matrix=========**\n{**gscv\_Lr\_cMatrix**}\n**")

print(f"======Classification Report======**\n{**gscv\_Lr\_cReport**}**")

Accuracy Score: 0.9620876415558838

========Confusion Matrix=========

[[1021 36]

[ 41 933]]

======Classification Report======

precision recall f1-score support

0 0.96 0.97 0.96 1057

1 0.96 0.96 0.96 974

accuracy 0.96 2031

macro avg 0.96 0.96 0.96 2031

weighted avg 0.96 0.96 0.96 2031

### 2. GaussianNB

**from** **sklearn.naive\_bayes** **import** GaussianNB

*#Finding the best random\_state for train test split*

best\_rState, best\_aScore = get\_best\_random\_state(500,GaussianNB(),x,y)

print(f"Best random\_state: **{**best\_rState**}** with best accuracy\_score: **{**best\_aScore**}**")

Best random\_state: 82 with best accuracy\_score: 0.9453471196454948

*#Splitting the train test data with random\_state = 82*

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.25,random\_state=82)

*#Hypertuning parameters*

parameters = {

"var\_smoothing": [1e-10,1e-9,1e-8,1e-7]

}

*#Finding best cv*

best\_cv, best\_cvScore = get\_best\_cv(GaussianNB(),parameters,x\_train,y\_train)

print(f"Best CV: **{**best\_cv**}** with best cross\_val\_score: **{**best\_cvScore**}**")

Best CV: 9 with best cross\_val\_score: 0.9328737895946168

*#Building Final Model with hypertuned parameters*

gscv\_Gnb = GridSearchCV(GaussianNB(),parameters,cv=9)

gscv\_Gnb.fit(x\_train,y\_train)

print(gscv\_Gnb.best\_params\_)

{'var\_smoothing': 1e-10}

*#Checking Final Performance of the Model*

gscv\_Gnb\_predict\_y = gscv\_Gnb.best\_estimator\_.predict(x\_test)

gscv\_Gnb\_aScore = accuracy\_score(y\_test,gscv\_Gnb\_predict\_y)

gscv\_Gnb\_cMatrix = confusion\_matrix(y\_test,gscv\_Gnb\_predict\_y)

gscv\_Gnb\_cReport = classification\_report(y\_test,gscv\_Gnb\_predict\_y)

print(f"Accuracy Score: **{**gscv\_Gnb\_aScore**}\n**")

print(f"========Confusion Matrix=======**\n{**gscv\_Gnb\_cMatrix**}\n**")

print(f"=====Classification Report=====**\n{**gscv\_Gnb\_cReport**}**")

Accuracy Score: 0.9453471196454948

========Confusion Matrix=======

[[1041 29]

[ 82 879]]

=====Classification Report=====

precision recall f1-score support

0 0.93 0.97 0.95 1070

1 0.97 0.91 0.94 961

accuracy 0.95 2031

macro avg 0.95 0.94 0.94 2031

weighted avg 0.95 0.95 0.95 2031

### 3. RandomForestClassifier

**from** **sklearn.ensemble** **import** RandomForestClassifier

*#Finding best random\_state for train test split*

best\_rState, best\_aScore = get\_best\_random\_state(500,RandomForestClassifier(),x,y)

print(f"Best random\_state: **{**best\_rState**}** with best accuracy\_score: **{**best\_aScore**}**")

Best random\_state: 0 with best accuracy\_score: 1.0

*#Splitting train test data with random\_state = 0*

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.25,random\_state=0)

*#Hypertuning Parameters*

parameters = {

"n\_estimators": [50,100,150],

"criterion": ['gini','entropy'],

"max\_features": ['auto','sqrt','log2'],

"bootstrap": [**True**,**False**],

"oob\_score": [**True**,**False**],

}

*#Finding best CV*

best\_cv, best\_cvScore = get\_best\_cv(RandomForestClassifier(),parameters,x\_train,y\_train)

print(f"Best CV: **{**best\_cv**}** with best cross\_val\_score: **{**best\_cvScore**}**")

Best CV: 2 with best cross\_val\_score: 1.0

*#Building Final Model with Hypertuned parameters*

gscv\_Rfc = GridSearchCV(RandomForestClassifier(),parameters,cv=2)

gscv\_Rfc.fit(x\_train,y\_train)

print(gscv\_Rfc.best\_params\_)

{'bootstrap': True, 'criterion': 'gini', 'max\_features': 'auto', 'n\_estimators': 50, 'oob\_score': True}

*#Checking Final Performance of the Model*

gscv\_Rfc\_predict\_y = gscv\_Rfc.best\_estimator\_.predict(x\_test)

gscv\_Rfc\_aScore = accuracy\_score(y\_test,gscv\_Rfc\_predict\_y)

gscv\_Rfc\_cMatrix = confusion\_matrix(y\_test,gscv\_Rfc\_predict\_y)

gscv\_Rfc\_cReport = classification\_report(y\_test,gscv\_Rfc\_predict\_y)

print(f"Accuracy Score: **{**gscv\_Rfc\_aScore**}\n**")

print(f"==========Confusion Matrix==========**\n{**gscv\_Rfc\_cMatrix**}**")

print(f"=======Classification Report========**\n{**gscv\_Rfc\_cReport**}**")

Accuracy Score: 1.0

==========Confusion Matrix==========

[[1061 0]

[ 0 970]]

=======Classification Report========

precision recall f1-score support

0 1.00 1.00 1.00 1061

1 1.00 1.00 1.00 970

accuracy 1.00 2031

macro avg 1.00 1.00 1.00 2031

weighted avg 1.00 1.00 1.00 2031

### Compare Model Performances

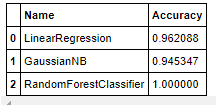
model\_names = ['LinearRegression','GaussianNB','RandomForestClassifier']

model\_scores =[gscv\_Lr\_aScore,gscv\_Gnb\_aScore,gscv\_Rfc\_aScore]

*#Comparing models*

models = pd.DataFrame({"Name":model\_names,"Accuracy":model\_scores})

models



From the above model performances we can easily conclude that the model RandomForestClassifer performs well with Accuracy Score of 100% as compare to model LinearRegression with 96.20% accuracy and GaussianNB with 94.53% accuracy. Therefore, moving further with **RandomForestClassifier**.

# Model Selection: The Final Model

In this step, we will save or serialize the final model which gives the highest performance into an object or pickle file.

**Importing Libraries and Saving Model:**

**import** **joblib**

filename = "mushroom\_project\_model.obj"

joblib.dump(gscv\_Rfc.best\_estimator\_,open(filename,'wb'))

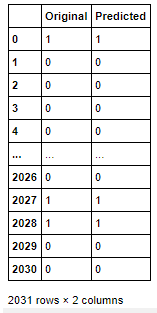
## Conclusion

The final model performance is good with **Accuracy: 100% and cross\_val\_score: 100%**.

This can be depicted using a comparison table of Original and Predicted data:

cnc = pd.DataFrame({"Original":np.array(y\_test),"Predicted":np.array(gscv\_Rfc\_predict\_y)})

cnc



We can see that the original and predicted values are almost similar which is a good sign but with the more training with new data may produce more stable model as well.

# Model Prediction

In this phase we will supply the set of test data to the serialized model to get the predicted value by using following steps:

## Pre-Processing Pipeline

Pipelines are a simple way to keep data pre-processing and modelling code organized. Specifically, a pipeline bundles pre-processing and modelling steps so that one can use the whole bundle as if it were a single step. This process includes the following steps which has been already discussed above:

* 1. Load Test Data
  2. Treat Null Values
  3. Encode Discrete Features (using LabelEncoder)
  4. Remove Outliers using zscore
  5. Treat skewness using power\_transform
  6. Scale data using StandardScaler
  7. Load Serialized Model and Make Prediction for Test Data

## Load Model from Serialized Object/Pickle file

model\_file\_name = 'mushroom\_project\_model.obj'

loaded\_model = joblib.load(model\_file\_name)

*#Predicting Test Data*

predict\_test = loaded\_model.predict(test\_x)

## Save Predicted Values

In this step, we will save the predicted value to a .csv (Comma Separated Values) file.

*## Saving predicted data to .csv file*

pred\_t.to\_csv('predicted\_test\_data.csv')

# Concluding Remarks

To expand on this project to get a better final model, I would like to see if there is similar data available for further analysis. This way I can test my model further and perhaps find some more insight into what will work best in a real-world situation. My model performed perfectly on my dataset, so it’s worth questioning if the dataset was perhaps compromised and therefore, testing my model on another dataset could provide further validation.