MACHINE LEARNING

MUSHROOM CLASSIFICATION (EDIBLE OR POISONOUS)

Data Set Information : This data set includes descriptions of hypothetical samples corresponding to 23 species 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.

Mushroom records drawn from The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf.

Attributes Information :

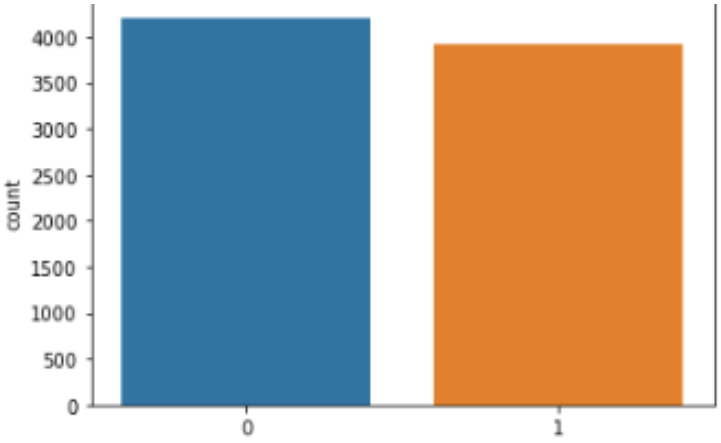
1. cap-shape : bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
2. cap-surface : fibrous=f,grooves=g,scaly=y,smooth=s
3. cap-color : brown=n,buff=b,cinnamon=c,gray=g,green=r, pink=p,purple=u,red=e,white=w,yellow=y
4. bruises : bruises=t,no=f
5. odor : almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s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

Libraries Used :

- **Pandas**
- **Numpy**
- **Seaborn**
- **Matplotlib**
- **Scipy**

The Mushroom dataset is a **Supervised Binary Classification** type of machine learning problem.

- The dataframe has 8124 rows and 23 columns originally.
- The target variable ‘class ’ has 2 types of results as ‘e’ edible and ‘p’ poisonous.
- Total no of edible mushrooms are 4208 in the record.
- total no of poisonous mushrooms are 3916 in the record.
- We can conclude that the data is available in approximity balance form for both types of classification.



Exploring & Cleaning of Data :

- duplicate values = '0'
- missing values = '?'
- null-values= '0'

There are missing values present in the column 'stalk-root'.

Handling missing value :

We fill the missing value with the most frequent occupied category in the respective feature.

```
# replacing the ? value with the mode 'b'
df['stalk-root'] = df['stalk-root'].str.replace('?', 'b', regex=True)
print(df['stalk-root'].value_counts())
```

```
b    6256
e    1120
c     556
r     192
```

Data Description :

Finding out the total count ,mean , standard deviation, quarter percentiles , minimum and maximum values of the continuous type numerical data columns.

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring
count	8124.000000	8124.000000	8124.000000	8124.000000	8124.000000	8124.000000	8124.000000	8124.000000	8124.000000	8124.000000	...	8124.000000
mean	0.482029	3.348104	1.827671	4.504677	0.415559	4.144756	0.974151	0.161497	0.309207	4.810684	...	1.603644
std	0.499708	1.604329	1.229873	2.545821	0.492848	2.103729	0.158695	0.368011	0.462195	3.540359	...	0.675974
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000
25%	0.000000	2.000000	0.000000	3.000000	0.000000	2.000000	1.000000	0.000000	0.000000	2.000000	...	1.000000
50%	0.000000	3.000000	2.000000	4.000000	0.000000	5.000000	1.000000	0.000000	0.000000	5.000000	...	2.000000
75%	1.000000	5.000000	3.000000	8.000000	1.000000	5.000000	1.000000	0.000000	1.000000	7.000000	...	2.000000
max	1.000000	5.000000	3.000000	9.000000	1.000000	8.000000	1.000000	1.000000	1.000000	11.000000	...	3.000000

EXPLORATORY DATA ANALYSIS

It is important for us to have insights about data .It helps us find informations which are hidden in the data .By presenting the data features on graph we can observe & draw certain conclusions.For the graph sketching we use libraries ‘Seaborn’ and ‘Matplotlib’.

Visualization

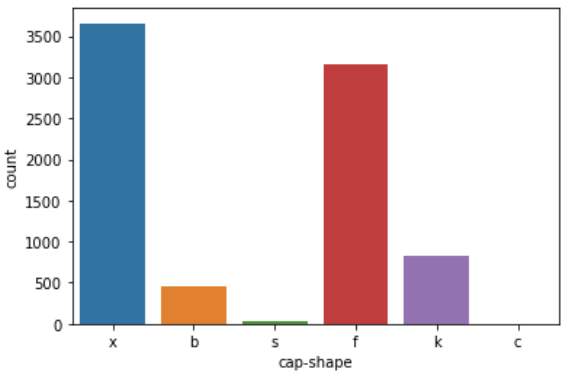
The dataframe has categorical type of columns data.

Uni-Variate Analysis :

- On categorical data we perform uni-variate analysis , which gives us information about certain category which is popularly occupied and least occupied category in specific column.
- Following are the feature’s graph with highest and lowest. occupied category in the record.

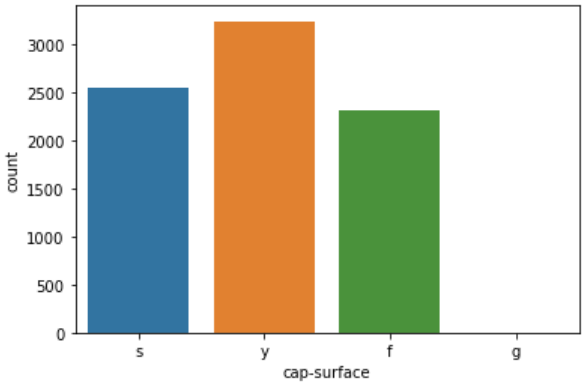
Cap-shape :

- Highest = x (convex)= 3656
- Lowest = c (sunken)= 4



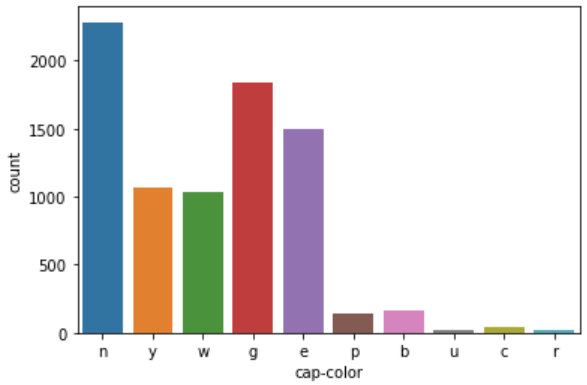
Cap-surface :

- Highest = y (scaly) = 3244
- Lowest = g (grooves) = 4



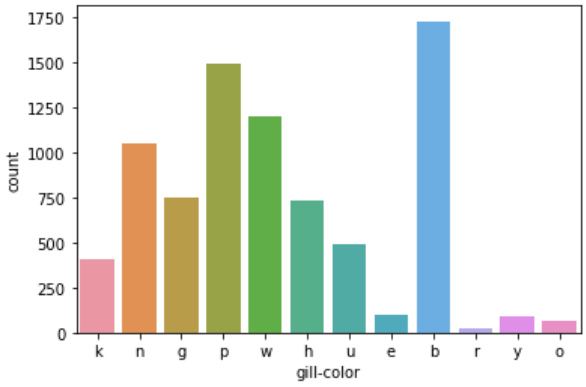
Cap-color :

- Highest = n (brown) = 2284
- Lowest = u (purple) = 16



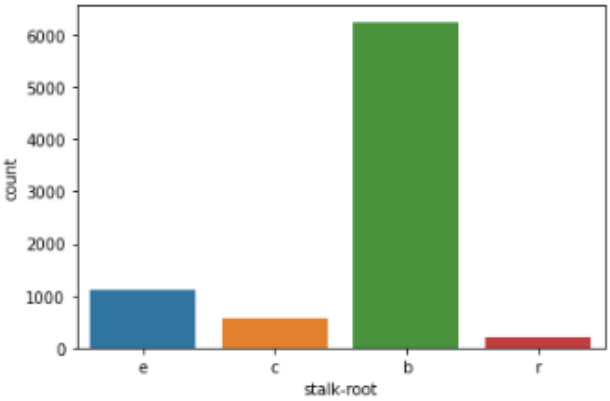
Gill-color :

- Highest = b (buff) = 1728
- Lowest = r (green) = 24



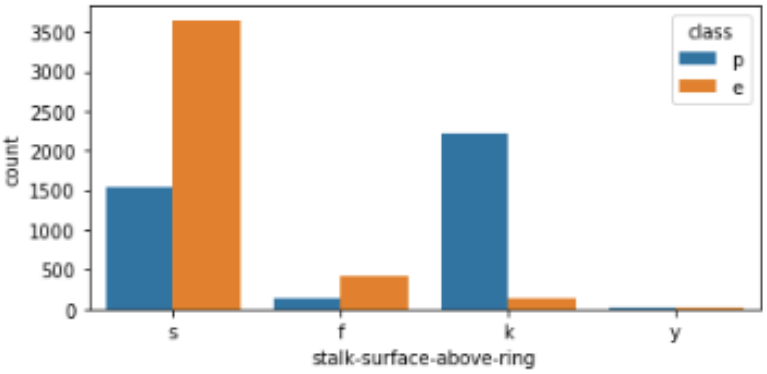
Stalk-root :

- Highest = b (bulbous) = 6256
- Lowest = r (rooted) = 192



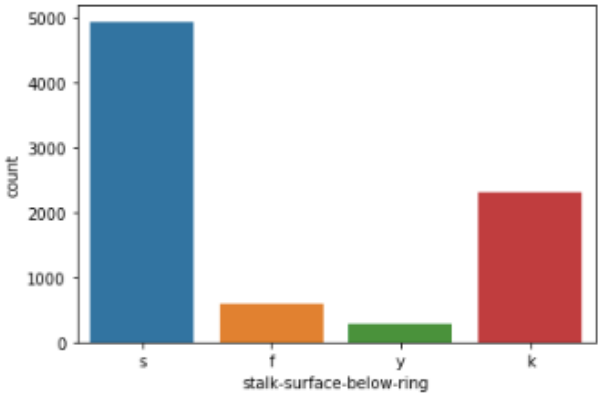
Stalk-surface-above-ring :

- Highest = s (smooth) = 5176
- Lowest = y (silky) =24



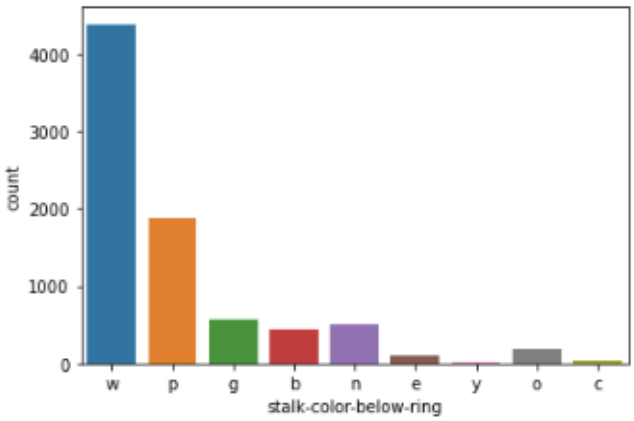
Stalk-surface-below-ring :

- Highest = s (smooth) = 4936
- Lowest = y (silky) = 284



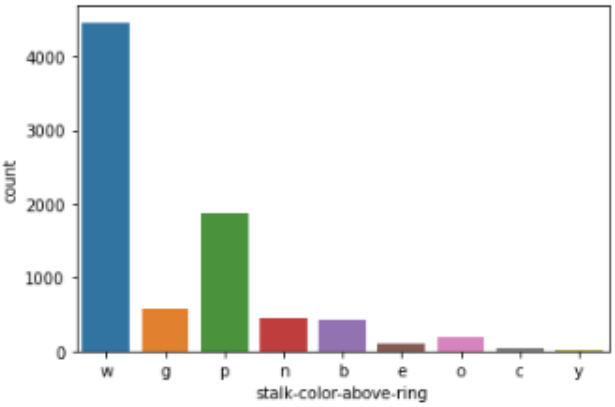
Stalk-color-below-ring :

- Highest = w (white) = 4384
- Lowest = y (yellow) = 24



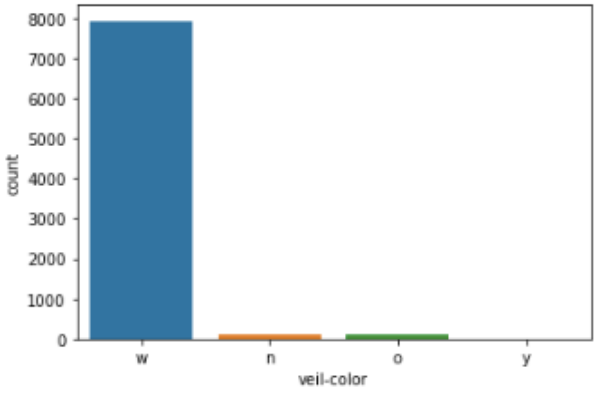
Stalk-color-above-ring :

- Highest = w (white) = 4464
- Lowest = y (yellow) = 8



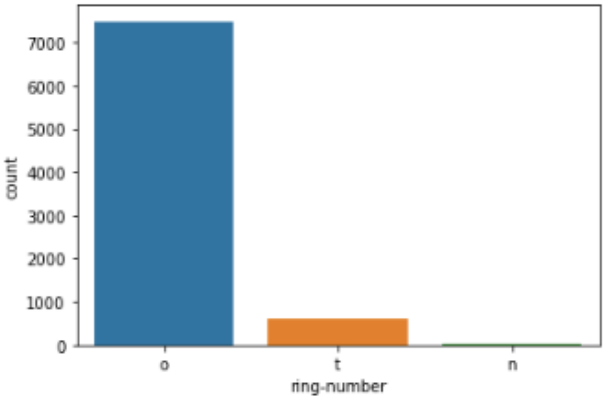
Veil-color :

- highest = w (white) = 7924
- Lowest = y (yellow) = 8



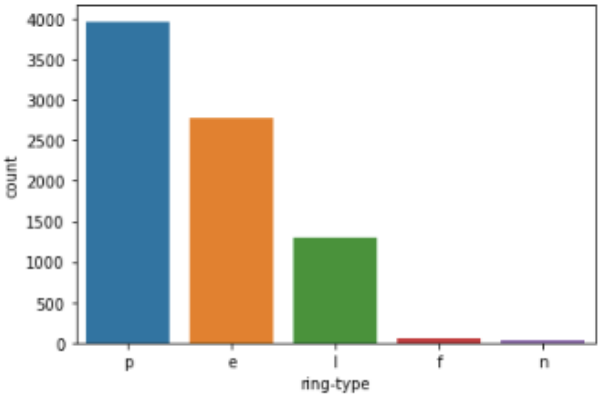
Ring-number :

- Highest = o (one) = 7488
- Lowest =n (none) = 36



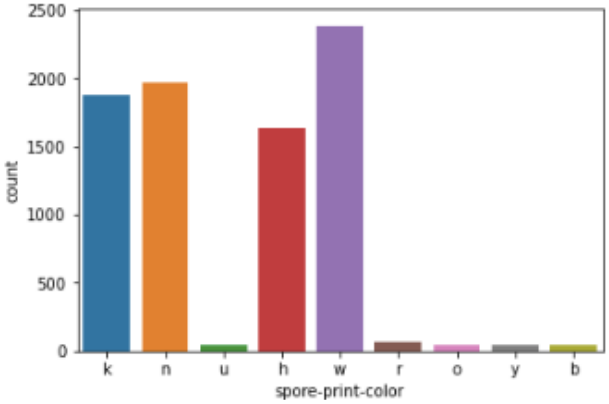
Ring-type :

- Highest = p (pendent) = 3968
- Lowest = f (flaring) = 48



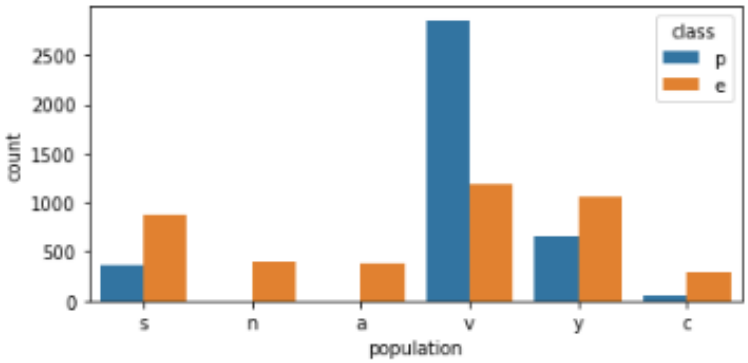
Spore-print-color :

- Highest = w (white) =2388
- Lowest = u (purple) = 48



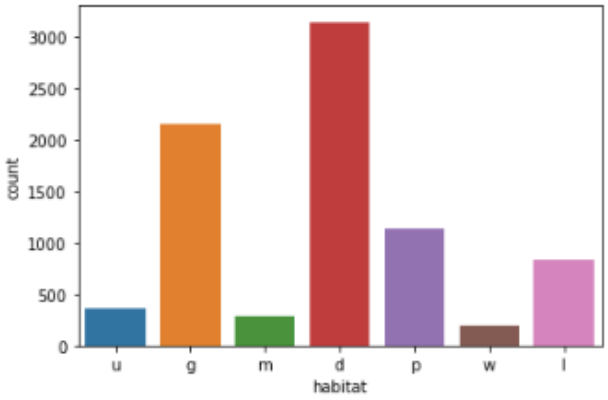
Population :

- Highest = v (several) = 4040
- Lowest = c (clustered) = 340



Habitat :

- Highest = d (woods) = 3148
- Lowest = w (waste) = 192



These graphs show the most occupied and least occupied category in the record in respective features.

To begin with Bi-variate analysis , we first encode the dataframe into numerical form so that we can pas the columns with alphabetic values into seaborn commands.

LABEL ENCODING :

Thorough cleaning of the data has been done but to begin with our work with models training & testing part we need to pre-process the data in order to make it ready to pass through the model.We need to check the data types of values in different columns of the dataset and encode them into numerical format first.

Now, a loop has been introduced in which the dataframe will be passed and whichever column has data-type as ‘object-type’ it will be encoded into numeric format.

```
le=preprocessing.LabelEncoder()
for col in df.columns:
    df[col]=le.fit_transform(df[col])
df
```

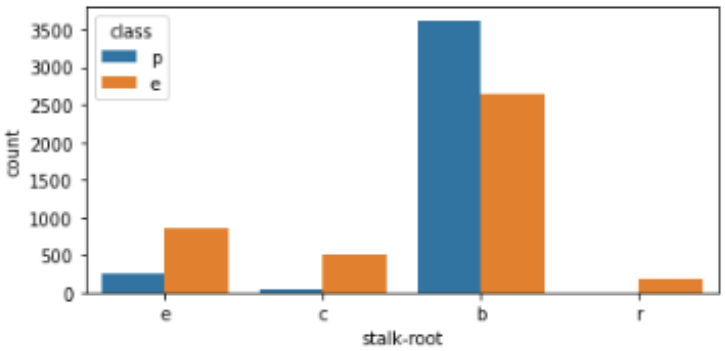
	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	pop
0	1	5	2	4	1	6	1	0	1	4	...	2	7	7	0	2	1	4	2	3
1	0	5	2	9	1	0	1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

Bi-Variate Analysis :

Now , we will sketch the graph with all features with respect to the ttarget variable i.e ‘class’.

Stalk-root vs class :

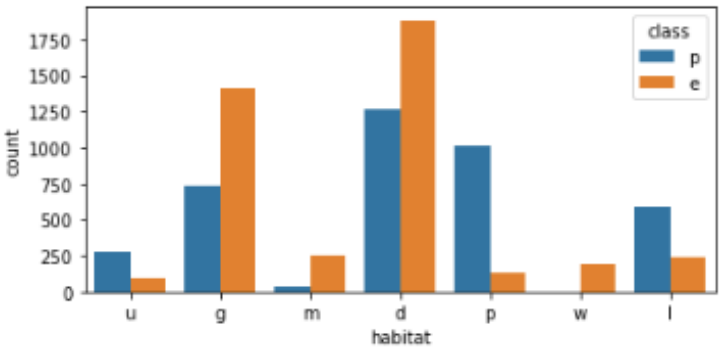
-The r-type (rooted) mushrooms are all edible.
-The c-type (club) mushrooms are least poisonous .



Habitat vs class :

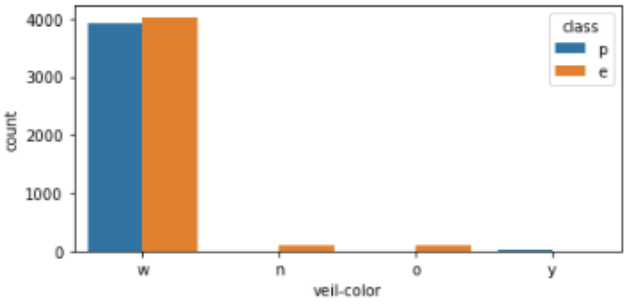
-The d-type (wood) mushrooms are highly poisonous as the difference between this category mushrooms into edible and poisonous is highest among all categories.

-The m-type (meadows) mushrooms are less poisonous , the difference between edible & poisonous is less.



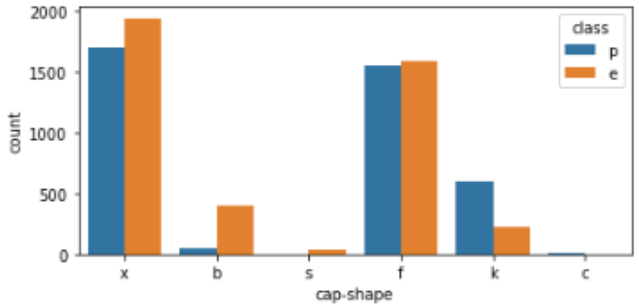
Veil-color vs class :

- The yellow coloured mushrooms are all posionous.
- The brown & orange are all edible.
- The white mushrooms can't be classified directly as edible or poisonous , we need more analyse them into edible or poisonous type.



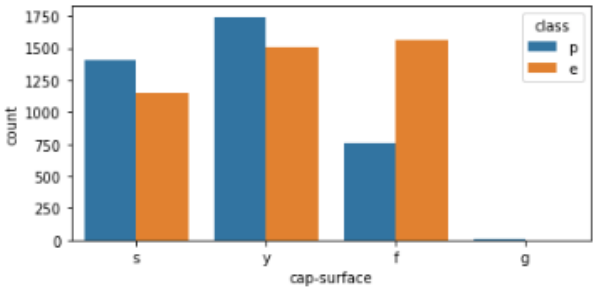
Cap-shape vs class :

- The x (convex) , f (flat) & k (knobbed) type mushrooms can't be classified on the basis of cap-shape as they are both edible and poisonous highly.
- The s (sunken) type mushrooms are all edible.
- The c (conical) type mushrooms are all poisonous.



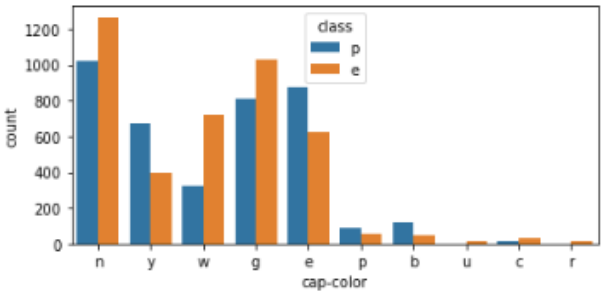
Cap-surface vs class :

- The g-type (grooves) mushrooms are all poisonous.
- The s (smooth) & y (scaly) can't be classified on the basis of cap surface.



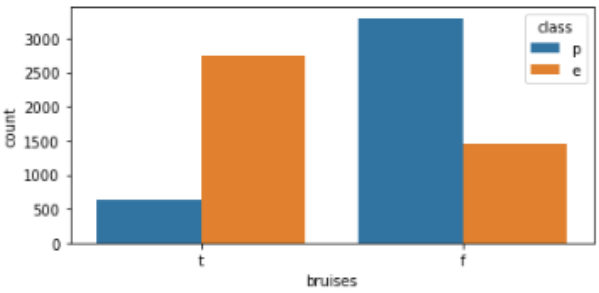
Cap-color vs class :

- The mushrooms with red and purple cap color are all edible as per the record.
- The mushrooms with cap color brown white & gray are highly edible.
- The mushrooms with cap color yellow , red , pink & buff are highly poisonous.



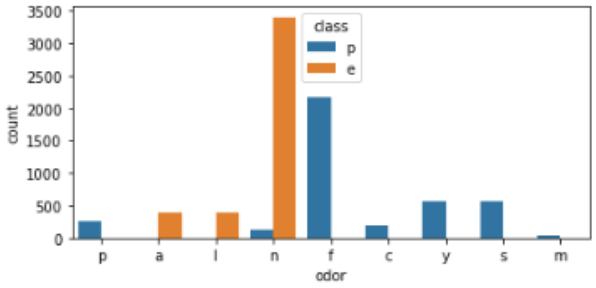
Bruises vs class :

- Mushrooms with bruises are highly edible than those which do not have bruises.



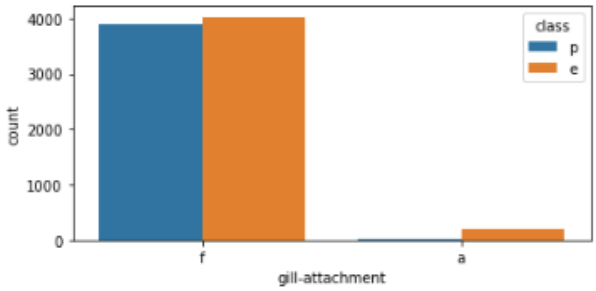
Odour vs class :

- Mushrooms with pungent , creosote , foul, fishy , spicy and musty odours are all poisonous.
- Mushrooms with almond and anise odours are all edible.
- Mushrooms with no odour are little poisonous.



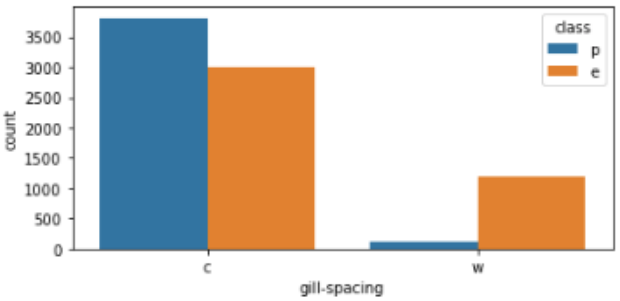
Gill-attachment vs class :

- The gill attachment free and notched does not make any difference in edibility of mushrooms.



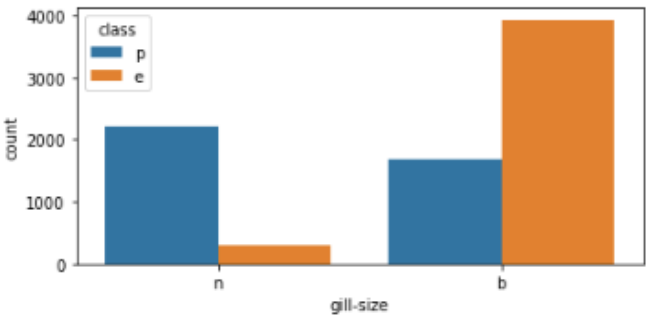
Gill-spacing vs class :

- The crowded gill spaced mushrooms are high in edibility.
- The distant gill spacing has no impact on edibility.



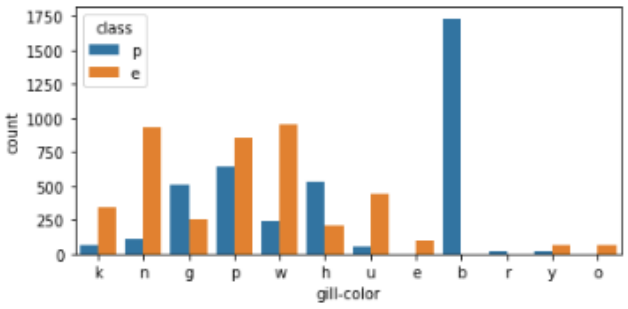
Gill-size vs class :

- The mushrooms with broad gill size are highly edible and less poisonous than with narrow size.



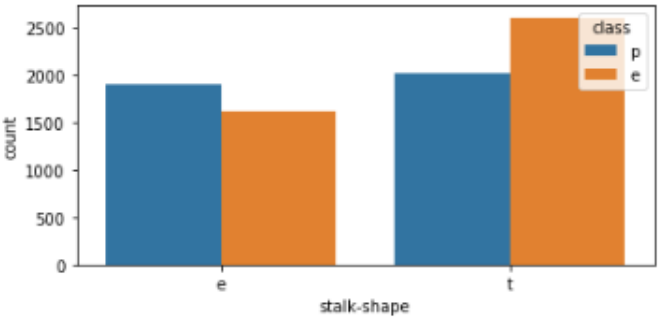
Gill-color vs class :

- The mushrooms with buff and green gill color are all poisonous.
- The mushrooms with red and orange gill color are all adible.
- The mushrooms with other colours are both edible and poisnous.



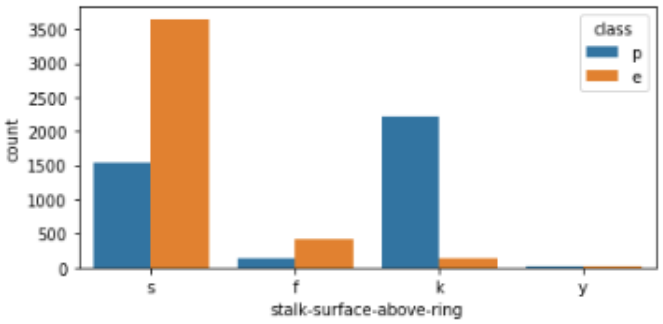
Stalk-Shape :

- The mushrooms with tapering stalk shape are high in edibility than with the enlarging stalk shape which are more poisonous.



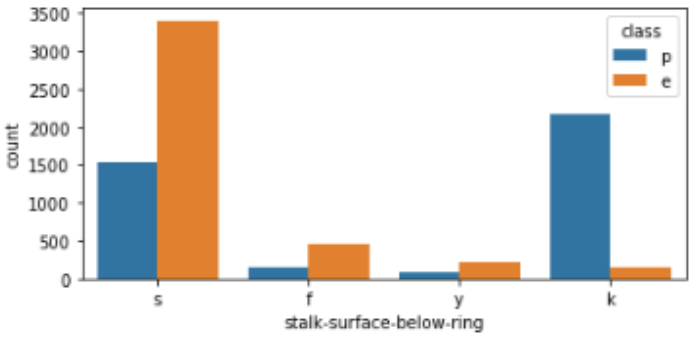
Stalk-surface-above-ring vs class :

- The mushrooms with smooth stalk surface above ring highly edible.
- The mushrooms with silky stalk surface above the ring are highly poisonous.



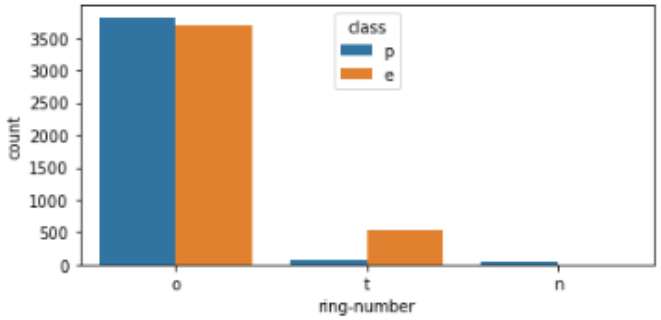
Stalk-surface-below-ring vs class :

- The mushrooms with silky stalk surface below the ring are highly poisonous.
- The mushrooms with smooth surface below the stalk ring are highly edible.



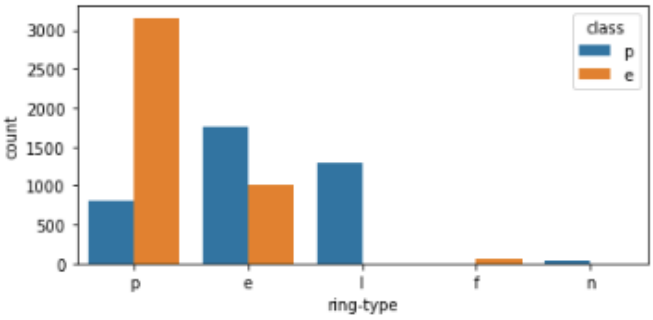
Ring-number vs class :

- The mushrooms with 2 rings are edible.
- the mushrooms with no ring cannot be classified.



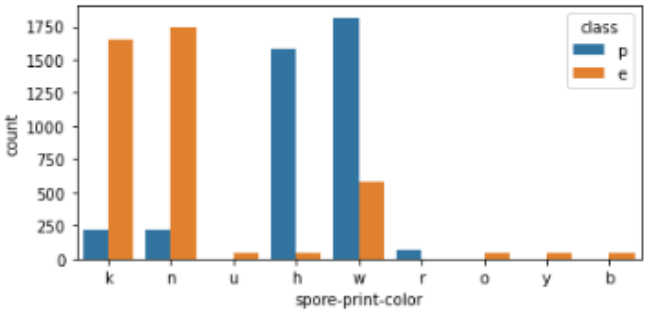
Ring-type vs class :

- The mushrooms with flaring ring type are all edible.
- The mushrooms with large ring type are all poisonous.



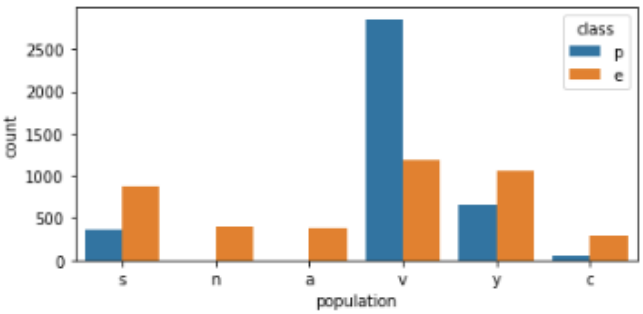
Spore-print-color vs class :

- The mushrooms with orange , yellow and buff type spore print color are all edible.
- The mushrooms with green color are all poisonous.



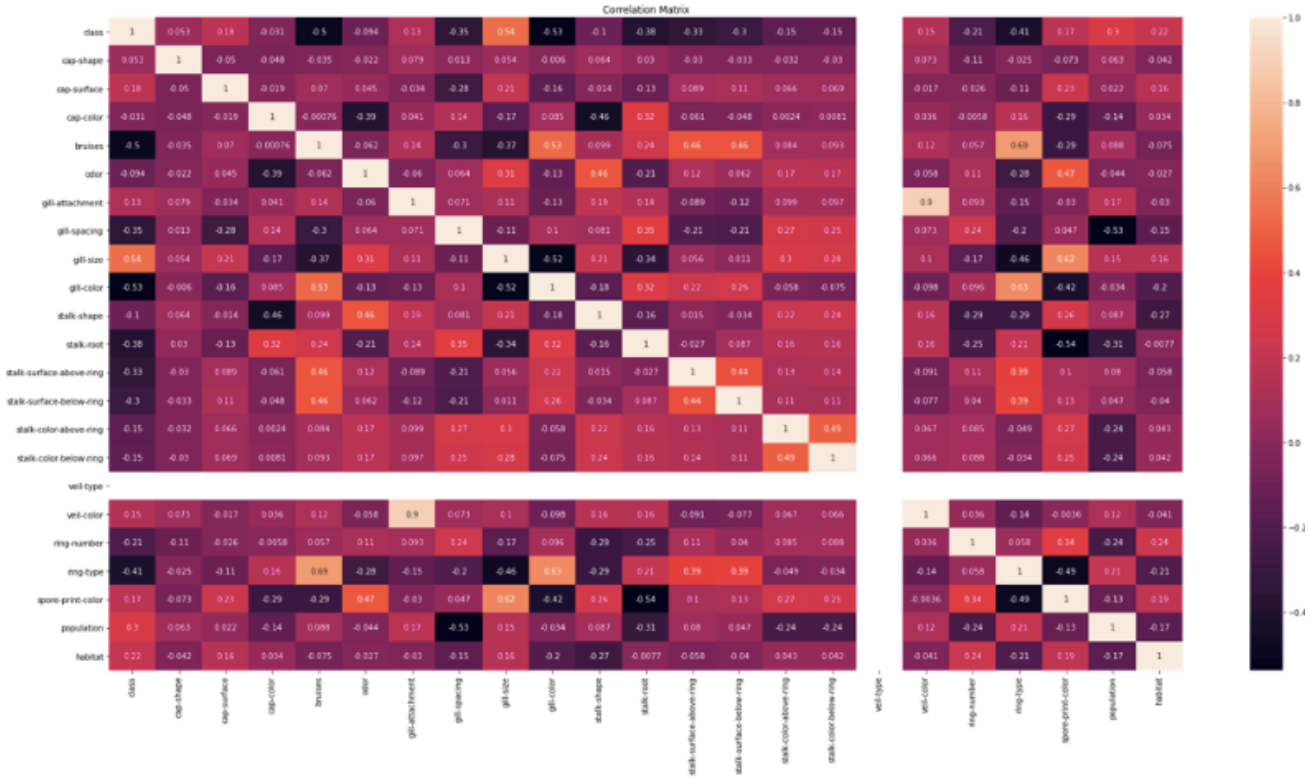
Population vs class :

- The mushrooms with population numerous and abundant are all edible.
- The mushrooms with population several and solitary are highly poisonous than edible.



Multi-Variate Analysis :

Correlation Heat-map

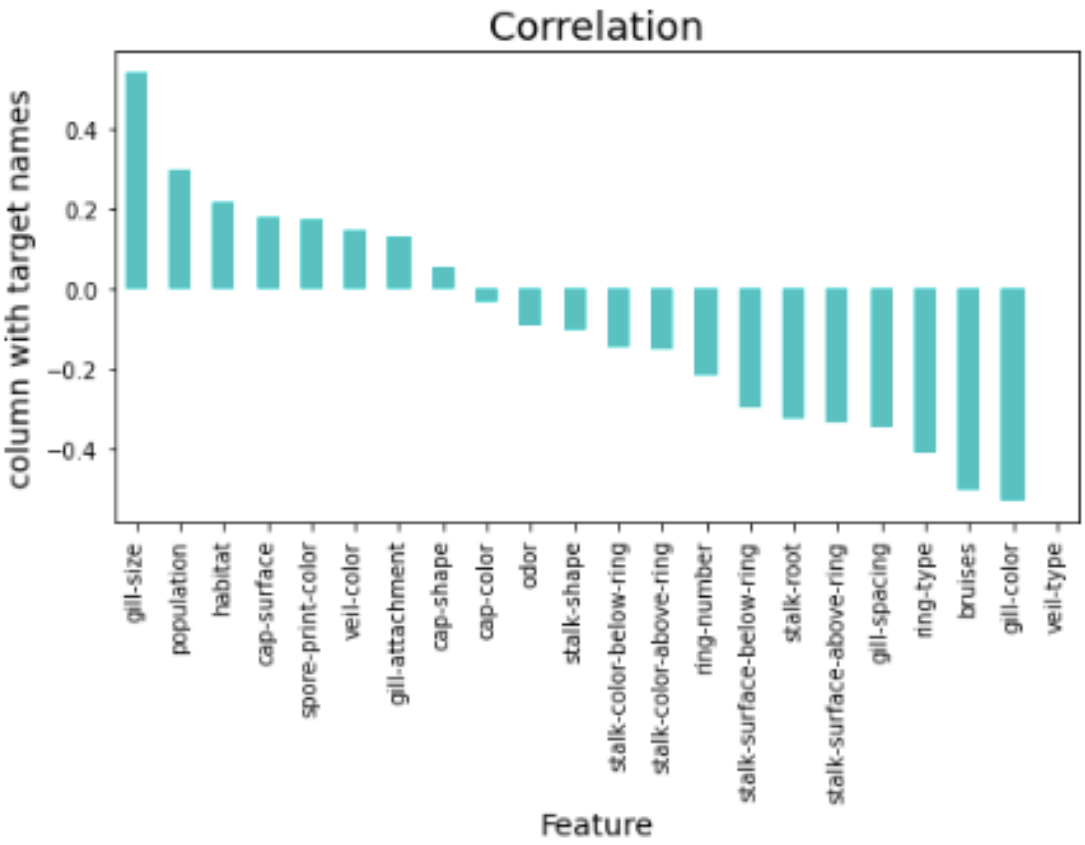


The graph indicates the correlation between the attributes which affect the edibility of the mushrooms.

- The darker the shade lower is the correlation between the attributes.
- The bright shades has comparatively high correlation with the attributes.
- The attributes 'veil-type' has no correlation with any attribute which implies that it does not affect the target variable 'class' , the edibilty of the mushroom.

The variable veil-type does not have any correlation with any feature of the data , so we will drop it .

Checking the negatively and positively correlated features with the target variable.

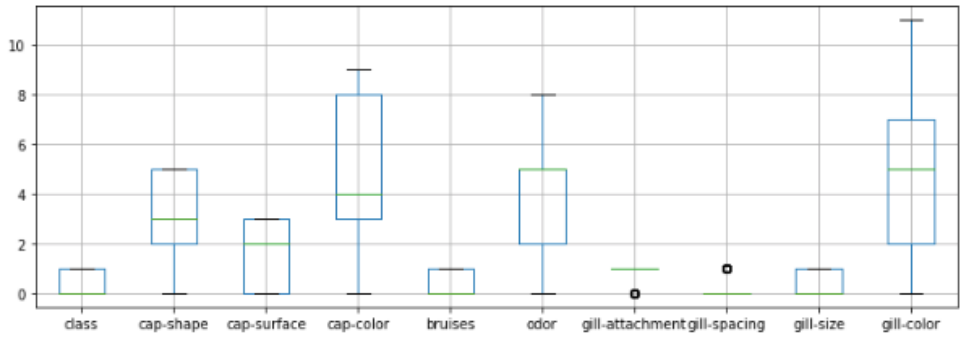


- Features below 0 are negatively related with the target variable that are odor, stalk-shape, stalk-color-above-ring , stalk-color-below-ring ,ring-number , stalk-root , stalk-surface-above ring , gill-spacing , ring-type , bruises , gill-color .
- Features above 0 are positvely related to the target variable that are gill-size , population , habitat , cap-surface , cap-shape , cap-color , veil-color , gill-attachment .
- The gill-size classifies mushrooms as edible or poisonous directly.

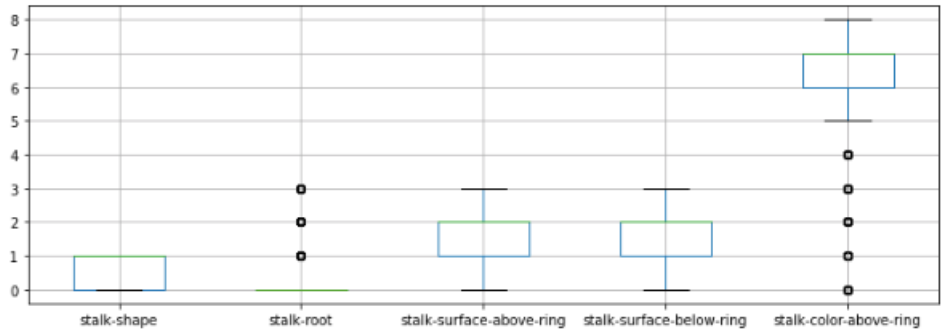
Box-Plot of dataframe :

The box plot graph shows the outliers present in the data , with 25th and 75th percentiles.

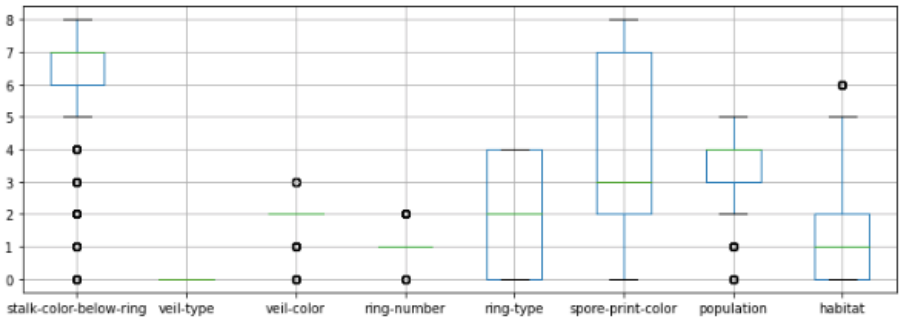
To get clear picture we will plot box plots of features with indexing .



- Outliers can be observed in gill-attachments and gill-spacing features.



- Outliers are present in stalk-root and stalk-color-above-ring features.



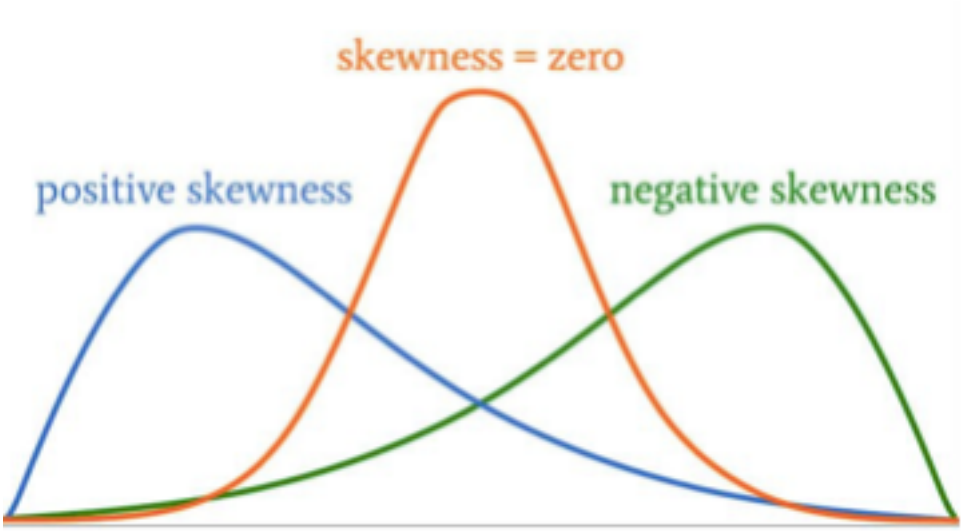
- Outliers are present in stalk-color-below-ring , veil-color , ring-number , population and habitat features.

Separating the target variable from features of the data :

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

Checking the skewness :

Checking the skewness in the features to see if they all are either right-tailed or left-tailed skewness.



The skewness range is +/- 0.5.

We will check the skewness of the feature columns of the dataframe and their range , in case they're out of range then we will perform pre-processing transformation for the skewness.

The skewness of the original data lies above the range +/- 0.5.

We will apply **Power Transformation** on the features for improving the skewness of the columns.

```
x.skew()
```

```
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
```

Improved Skewness :

```
from sklearn.preprocessing import power_transform
df=power_transform(x)
df=pd.DataFrame(df,columns=x.columns)
df.skew()
```

The improve skewness is alighned on same side tailed.

```
df_new.skew()
```

```
cap-shape                  -0.205860
cap-surface                -0.473764
cap-color                   0.004059
bruises                     0.342750
odor                       -0.144660
gill-attachment            -5.977076
gill-spacing                1.840088
gill-size                   0.825797
gill-color                 -0.246738
stalk-shape                -0.271345
stalk-root                  0.023182
stalk-surface-above-ring   -0.438332
stalk-surface-below-ring   -0.114150
stalk-color-above-ring     -0.964471
stalk-color-below-ring     -0.925320
veil-color                  5.797741
ring-number                -0.499098
ring-type                  -0.339232
spore-print-color           0.018174
population                 -0.368219
habitat                     0.123132
dtype: float64
```

Handling the outliers :

Outliers are the extreme values present in the data , these values are mis-interpreted or have been put by mistake or wrong informations that may occur because human error in data collection or any other mistake.

The outliers impact the machine learning models , the model efficiency decreases as it learns wrong informations.Therefore, we need to treat them before passing them into training phase into the models.

The outliers can be removed by z-score method or percentile method.I have used the zscore method to eliminate outliers.

Removing the outliers

```
from scipy.stats import zscore
z=np.abs(zscore(df))
z
```

```
array([[1.0366127 , 1.02971224, 0.14012794, ..., 0.67019486, 0.5143892 ,
        2.03002809],
       [0.96468045, 1.02971224, 0.14012794, ..., 0.2504706 , 1.31310821,
        0.29572966],
       [0.96468045, 2.08704716, 0.14012794, ..., 0.2504706 , 1.31310821,
        0.86714922],
       ...,
       [0.96468045, 0.8403434 , 0.14012794, ..., 1.50964337, 2.11182722,
        0.28570978],
       [1.0366127 , 0.21699152, 0.95327039, ..., 1.42842641, 0.28432981,
        0.28570978],
       [0.96468045, 1.02971224, 0.14012794, ..., 0.16925365, 2.11182722,
        0.28570978]])
```

```
threshold=3
print(np.where(z>3))
```

```
(array([1816, 2128, 2128, ..., 8121, 8123, 8123]), array([15, 14, 15, ..., 16,  6, 16]))
```

```
print(np.where(z<3))
```

```
(array([   0,    0,    0, ..., 8123, 8123, 8123]), array([ 0,  1,  2, ..., 19, 20, 21]))
```

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

Sample dataframe after outliers removal :

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

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	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
0	1	5	2	4	1	6	1	0	1	4	...	2	2	7	7	2	1	4	2
1	0	5	2	9	1	0	1	0	0	4	...	2	2	7	7	2	1	4	3
2	0	0	2	8	1	3	1	0	0	5	...	2	2	7	7	2	1	4	3
3	1	5	3	8	1	6	1	0	1	5	...	2	2	7	7	2	1	4	2
4	0	5	2	3	0	5	1	1	0	4	...	2	2	7	7	2	1	0	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

The data cleaning and exploring have been done , our refined is ready for the model's training and testing phases.

SCALING & BALANCING VARIABLE :

Scaling the data with the MinMax scaler to create uniformness in the data scale.

```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import f1_score
```

```
1 round(y.value_counts(normalize=True) * 100, 2).astype('str') + '
```

```
    51.8 %
    48.2 %
ame: class, dtype: object
```

The Target variable 'class' is already balanced.We can start testing the data on different models.

Testing :

Testing Models :-

Now , we will you this predicted best random state 199 for testing different classifier models.

```
x.shape   # x is the dependent variable.
```

```
(6568, 21)
```

```
y.shape   # y is the target variable
```

```
(6568,)
```

Finding out the best random state :

Before training the model we find a best random state where we get the best accuracy score.

Making a loop with **Logistic Regression** , in this we will pass the training data and print the best random state where accuracy score will be highest.

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression
maxAccuracy=0
maxRs=0
for i in range(100):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.22,random_state=i)
    lr = LogisticRegression()
    lr.fit(x_train,y_train)
    predrf=lr.predict(x_test)
    acc=accuracy_score(y_test,predrf)
    if acc>maxAccuracy:
        macAccuracy=acc
        maxRS=i
print('The best accuracy is', maxAccuracy,' on Random state',maxRS)
```

```
The best accuracy is 0 on Random state is 99.
```

TRAIN , TEST & SPLIT :

Creating train ,test and split with random state 99 and test_size (0.2).

Checking the shapes of x_train ,x_test ,y_train and y_test.

```
1 print("x_train shape:", x_train.shape)
2 print("x_test shape:", x_test.shape)
3 print("y_train shape:", y_train.shape)
4 print("y_test shape:", y_test.shape)
```

```
x_train shape: (6336, 22)
x_test shape: (1788, 22)
y_train shape: (6336,)
y_test shape: (1788,)
```

Naive Bayes Classifier :

We will pass the pre-processed data from the imported Naive Bayes model.

The accuracy score on random state 99 for multinomial model is **0.81.**

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
```

```
mnb=MultinomialNB()
mnb.fit(x_train,y_train)
predictionmnb=mnb.predict(x_test)
print(accuracy_score(y_test,predictionmnb))
print(confusion_matrix(y_test,predictionmnb))
print(classification_report(y_test,predictionmnb))
```

Confusion matrix & Classification Report :

```
0.8181818181818182
[[13  1]
 [ 3  5]]
              precision    recall  f1-score   support

           0       0.81      0.93      0.87        14
           1       0.83      0.62      0.71         8

    accuracy                           0.82        22
   macro avg       0.82      0.78      0.79        22
weighted avg       0.82      0.82      0.81        22
```

GaussianNB Classifier :

```
gnb=GaussianNB()
gnb.fit(x_train,y_train)
predictiongnb=gnb.predict(x_test)
print(accuracy_score(y_test,predictiongnb))
print(confusion_matrix(y_test,predictiongnb))
print(classification_report(y_test,predictiongnb))
```

The accuracy score on 99 random state is **0.81.**

Confusion matrix & Classification Report :

```
0.8181818181818182
[[13  1]
 [ 3  5]]
              precision    recall  f1-score   support

           0       0.81      0.93      0.87        14
           1       0.83      0.62      0.71         8

    accuracy                           0.82        22
   macro avg       0.82      0.78      0.79        22
weighted avg       0.82      0.82      0.81        22
```

Support Vector Classifier :

```
def svckernel(ker):
    svc=SVC(kernel=ker)
    svc.fit(x_train,y_train)
    svc.score(x_train,y_train)
    predictionsvc=svc.predict(x_test)
    print(accuracy_score(y_test,predictionsvc))
    print(confusion_matrix(y_test,predictionsvc))
    print(classification_report(y_test,predictionsvc))
```

Testing the SVC model on radial basis function (rbf) kernel :

```
svckernel('rbf')
```

```
1.0
[[14  0]
 [ 0  8]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      1.00      1.00         8

    accuracy                           1.00        22
   macro avg       1.00      1.00      1.00        22
weighted avg       1.00      1.00      1.00        22
```

The accuracy score on rbf kernel for SVC model is **1.0.**

Testing the SVC model on poly kernel :

```
svckernel('poly')
```

```
1.0
[[14  0]
 [ 0  8]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      1.00      1.00         8

    accuracy                           1.00        22
   macro avg       1.00      1.00      1.00        22
weighted avg       1.00      1.00      1.00        22
```

The accuracy score on ploy kernel for SVC model is **1.0.**

Decision Tree Classifier :

```
# decision tree classifier (criterion='gini') -- default
# decision trees classifier(criterion='entropy')
# gini and entropy
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(x_test,y_test)
predictiondtc=dtc.predict(x_test)
print(accuracy_score(y_test,predictiondtc))
print(confusion_matrix(y_test,predictiondtc))
print(classification_report(y_test,predictiondtc))
```

```
1.0
[[14  0]
 [ 0  8]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      1.00      1.00         8

    accuracy                           1.00        22
   macro avg       1.00      1.00      1.00        22
weighted avg       1.00      1.00      1.00        22
```

The accuracy score is **1.0.**

KNeighbors Classifier :

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=4)
knn.fit(x_train,y_train)
knn.score(x_train,y_train)
predictionknn=knn.predict(x_test)
print(accuracy_score(y_test,predictionknn))
print(confusion_matrix(y_test,predictionknn))
print(classification_report(y_test,predictionknn))
```

After passing the parameter of 4 values as neighbors for the model to obtain results.The accuracy score , confusion matrix and classification reports are as follows.-

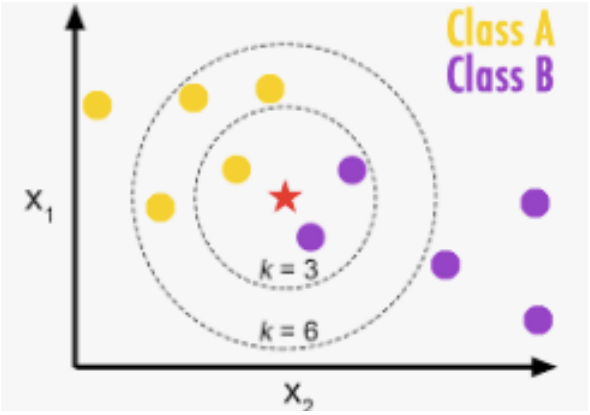
The accuracy with 4 nearest neighbors is 1.0.Now ,we will a loop in which we will pass different parameter of nearest neighbor and check at which number the model gives highest accuracy.

```
def kneighbors(k):
    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train,y_train)
    knn.score(x_train,y_train)
    predictionknn=knn.predict(x_test)
    print(accuracy_score(y_test,predictionknn))
    print(confusion_matrix(y_test,predictionknn))
    print(classification_report(y_test,predictionknn))
```

Kneighbors(3) - Accuracy score is 1.0 .

Kneighbors(5) - Accuracy score is 1.0.

Kneighbors(6) - Accuracy score is 1.0



Observation - On passing different no as the neighbors parameter in the model there wasn't a high difference of accuracy score observed.

We have tested the data on different classification models , but to come to a conclusion of best model we don't consider the accuracy score parameter only. We will check the **K-fold** values so that if there has been any over-fitting or under-fitting biased pattern followed by the models.After this we can conclude the best model for our data.

Cross-Validation Score

Repeated k-fold cross-validation provides a way to improve the estimated performance of a machine learning model.This mean result is expected to be a more accurate estimate of the true unknown underlying mean performance of the model on the dataset, as calculated using the standard error.

We will find out the cross validation score for each tested model and compare them.

Models comparison :

MODEL NAME	ACCURACY SCORE	K-FOLD VALUE	DIFFERENCE
Multinomial Classifier	0.81	0.800144	2
Gaussian Classifier	0.81	0.836975	-0.2
Decision Tree Classifier	1.0	0.969550	0.03
Kneighbors Classifier	1.0	0.942762	0.06
SVC	1.0	0.942759	0.06

RESULT :

The minimum difference between the cross-validation score and accuracy score is for the **DecisionTree Classifier** model.

Finding out the Cross Validation Score for each model tested

```
from sklearn.model_selection import cross_val_score
```

MultinomialNB Model

```
cvs=cross_val_score(mnb,x,y,cv=5)
print('The cross validation score for MultinomialNB model is :',cvs.mean(),'.')
print('The difference between accuracy and crossvalidation score is : 2 .')
```

```
The cross validation score for MultinomialNB model is : 0.8001447879245248 .
The difference between accuracy and crossvalidation score is : 2 .
```

GaussianNB Model

```
cvs=cross_val_score(gnb,x,y,cv=5)
print('The cross validation score for GaussianNB model is :',cvs.mean())
print('The difference between accuracy and crossvalidation score is : -2 .')
```

```
The cross validation score for GaussianNB model is : 0.8369751727543671
The difference between accuracy and crossvalidation score is : -2 .
```

Support Vector Classifier Model

```
cvs=cross_val_score(svc,x,y,cv=5)
print('The cross validation score for SupportVectorClassification model is :',cvs.mean())
print('The difference between accuracy and crossvalidation score is :0.06.')
```

```
The cross validation score for SupportVectorClassification model is : 0.9427590388122058
The difference between accuracy and crossvalidation score is :0.06.
```

Nearest Neighbor Model

```
cvs=cross_val_score(knn,x,y,cv=5)
print('The cross validation score for NearestNeighbor model is :',cvs.mean())
print('The difference between accuracy and crossvalidation score is :0.06.')
```

```
The cross validation score for NearestNeighbor model is : 0.9427620528122359
The difference between accuracy and crossvalidation score is :0.06.
```

Decision Tree Classifier model

```
cvs=cross_val_score(dtc,x,y,cv=5)
print('The cross validation score for DecisionTreeClassifier model is :',cvs.mean())
print('The difference between accuracy and crossvalidation score is :0.03.')
```

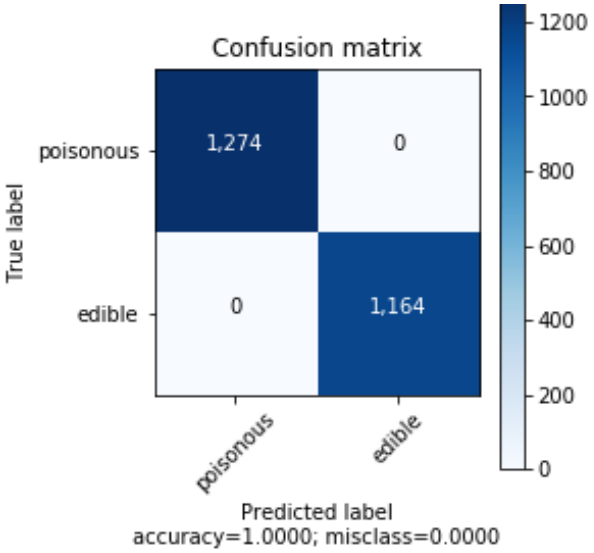
```
The cross validation score for DecisionTreeClassifier model is : 0.969550253233964
The difference between accuracy and crossvalidation score is :0.03.
```

- The minimum difference between the accuracy score and cross validatio score is for the Decision Tree Clasifier Model (0.03) so this is our best model.

HyperParameter Tuning :

As we have selected **DecisionTree Classifier** model as the best model

we will tune and check the confusion matrix.



	precision	recall	f1-score	support
0	0.97	0.88	0.93	4990
1	0.89	0.98	0.93	4898
accuracy			0.93	9888
macro avg	0.93	0.93	0.93	9888
weighted avg	0.93	0.93	0.93	9888

Hyper Parameter Tuning

```
from sklearn.model_selection import GridSearchCV
```

```
# creating parameter list to pass in GridSearchCV

parameters = {'max_depth': np.arange(2,15),'criterion':['gini','entropy']}
```

```
GCV=GridSearchCV(DecisionTreeClassifier(),parameters,cv=5)
```

```
GCV.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14
])})
```

```
GCV.best_params_
```

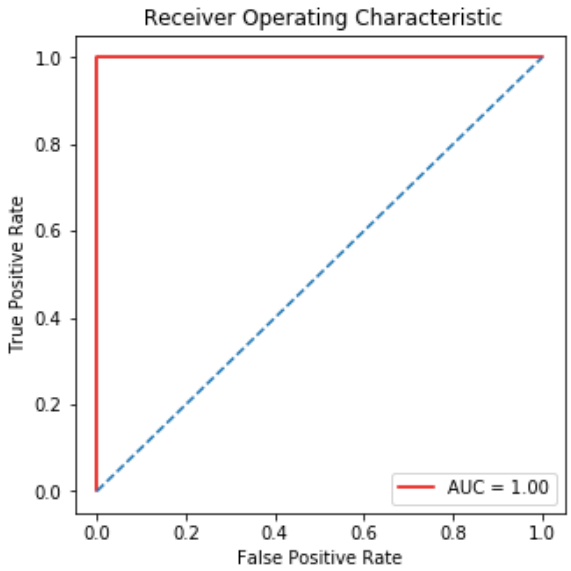
```
{'criterion': 'gini', 'max_depth': 6}
```

```
GCV_pred=GCV.best_estimator_.predict(x_test)
```

```
accuracy_score(y_test,GCV_pred)
```

```
1.0
```

ROC Curve for area under the curve :



-According to the obtained Training and Validation Accuracy, it can be concluded that the model is a good fit.

- The Area Under the Receiver Operator Characteristic in the graph is a descent one, as more the AUROC (towards 1.0), better the performance of the model.

Saving the model :

We can save the best model with pickle method or joblib.

I have used pickle method to dump and save my model.

The best model is **DecisionTree Classifier** with accuracy score as 1.0 and cross-validation score 0.03.

Saving the model

```
import pickle

filename='pickledtcfile.pkl'
pickle.dump(dtc,open('pickledtcfile','wb'))
```

```
saved_model=pickle.load(open('pickledtcfile','rb'))
saved_model.predict(x_test)
```

```
array([0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1])
```

Conclusion :

Using only 19 pieces of information, we can conclude with 100% certainty that a mushroom is edible or poisonous. These are the 19 features, ranked in descending order by the absolute value with their correlation with the target, class. Recall that in the target 'class' ,edible was marked as 0 and posionous was marked as 1.

- 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

Rank	Feature	Correlation with target
1	odor_n	-0.7855566222367798
2	odor_f	0.6238419598140276
3	stalk-surface-above-ring_k	0.587658257630713
4	stalk-surface-below-ring_k	0.5735240117095786
5	ring-type_p	-0.5404689127748091
6	gill-size_n	0.5400243574329782
7	gill-size_b	-0.5400243574329637
8	gill-color_b	0.5388081615534243
9	bruises_f	0.5015303774076804
10	bruises_t	-0.5015303774076804
11	stalk-surface-above-ring_s	-0.49131418071172045
12	spore-print-color_h	0.49022917056952875
13	ring-type_l	0.4516190663173296
14	population_v	0.44372237905537937
15	stalk-surface-below-ring_s	-0.42544404585499374
16	spore-print-color_n	-0.41664529710731296
17	spore-print-color_k	-0.3968322009032092
18	spore-print-color_w	0.3573840060325314
19	gill-spacing_c	0.34838678518425714

Selected-Features

Out original features (before engineering), the 19 listed above were engineered from 9 of the 22 originals. They were as follows;

1	Odor
2	Stalk-surface-above-ring
3	Stalk-surface-below-ring
4	Ring type
5	Gill size
6	Bruises
7	Spore-print-color
8	population

Rejected-Features

Thus , using these 19 features we can predict with the machine learning models if the mushroom is edible or poisonous in nature.

Jupyter Notebook (github-link) :

https://github.com/ShivaniKatariaO5/Analysis-Projects/blob/main/Datatrained-Project-Mushroom.ipynb

Declaration :

The Mushroom Project was the first independent project I worked upon , In the first attempt I missed out on some parts , while working on this project I have learnt a lot and tried to implement all the knowledge that I have received from my mentors.

I am obliged to the **Datatrained** team, teachers ,project mentors, technical & non-technical support teams for their guidance and continuous support.