Primary imports

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

import dataset by pandas read.csv() fuction.

In [2]:
```
data = pd.read_csv('mushrooms.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	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	habit
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

Checking shape of dataset

In [4]:
```
data.shape
```

Out[4]:
```
(8124, 23)
```

- Dataset is having 8124 rows and 23 columns.

In [5]:
```
data.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
```

In [6]:

```
# checking missing values
data.isnull().sum()
```

Out[6]:

```
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
```

- There is no any missing value in dataset.

In [7]:
```
# checking duplicated values
data.duplicated().sum()
```

Out[7]:
```
0
```

- Dataset has no any duplicated row.

In [8]:
```
# set option for display all columns in dataset
pd.set_option('display.max_columns',None)
data
```

Out[8]:

	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-type	veil-color	ring-number	ring-type	s
0	p	x	s	n	t	p	f	c	n	k	e	e	s	s	w	w	p	w	o	p	
1	e	x	s	y	t	a	f	c	b	k	e	c	s	s	w	w	p	w	o	p	
2	e	b	s	w	t	l	f	c	b	n	e	c	s	s	w	w	p	w	o	p	
3	p	x	y	w	t	p	f	c	n	n	e	e	s	s	w	w	p	w	o	p	
4	e	x	s	g	f	n	f	w	b	k	t	e	s	s	w	w	p	w	o	e	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
8119	e	k	s	n	f	n	a	c	b	y	e	?	s	s	o	o	p	o	o	p	
8120	e	x	s	n	f	n	a	c	b	y	e	?	s	s	o	o	p	n	o	p	
8121	e	f	s	n	f	n	a	c	b	n	e	?	s	s	o	o	p	o	o	p	
8122	p	k	y	n	f	y	f	c	n	b	t	?	s	k	w	w	p	w	o	e	

	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-type	veil-color	ring-number	ring-type	s
8123	e	x	s	n	f	n	a	c	b	y	e	?	s	s	o	o	p	o	o	p	

8124 rows × 23 columns

checking how many unique values in each categorical columns and which are those unique values in each categorical feature:

In [9]:

```
for i in data.columns:
    print("****",i,"****")
    print('No. of unique values in ', i, ' : ',data[i].nunique())
    print('Unique values for ',i,' :',data[i].unique())

    print()
```

```
**** class ****
No. of unique values in  class  :  2
Unique values for  class  : ['p' 'e']

**** cap-shape ****
No. of unique values in  cap-shape  :  6
Unique values for  cap-shape  : ['x' 'b' 's' 'f' 'k' 'c']

**** cap-surface ****
No. of unique values in  cap-surface  :  4
Unique values for  cap-surface  : ['s' 'y' 'f' 'g']

**** cap-color ****
No. of unique values in  cap-color  :  10
Unique values for  cap-color  : ['n' 'y' 'w' 'g' 'e' 'p' 'b' 'u' 'c' 'r']

**** bruises ****
No. of unique values in  bruises  :  2
Unique values for  bruises  : ['t' 'f']

**** odor ****
No. of unique values in  odor  :  9
Unique values for  odor  : ['p' 'a' 'l' 'n' 'f' 'c' 'y' 's' 'm']

**** gill-attachment ****
No. of unique values in  gill-attachment  :  2
Unique values for  gill-attachment  : ['f' 'a']
```

```
**** gill-spacing ****
No. of unique values in  gill-spacing  :  2
Unique values for  gill-spacing  : ['c' 'w']

**** gill-size ****
No. of unique values in  gill-size  :  2
Unique values for  gill-size  : ['n' 'b']

**** gill-color ****
No. of unique values in  gill-color  :  12
Unique values for  gill-color  : ['k' 'n' 'g' 'p' 'w' 'h' 'u' 'e' 'b' 'r' 'y' 'o']

**** stalk-shape ****
No. of unique values in  stalk-shape  :  2
Unique values for  stalk-shape  : ['e' 't']

**** stalk-root ****
No. of unique values in  stalk-root  :  5
Unique values for  stalk-root  : ['e' 'c' 'b' 'r' '?']

**** stalk-surface-above-ring ****
No. of unique values in  stalk-surface-above-ring  :  4
Unique values for  stalk-surface-above-ring  : ['s' 'f' 'k' 'y']

**** stalk-surface-below-ring ****
No. of unique values in  stalk-surface-below-ring  :  4
Unique values for  stalk-surface-below-ring  : ['s' 'f' 'y' 'k']

**** stalk-color-above-ring ****
No. of unique values in  stalk-color-above-ring  :  9
Unique values for  stalk-color-above-ring  : ['w' 'g' 'p' 'n' 'b' 'e' 'o' 'c' 'y']

**** stalk-color-below-ring ****
No. of unique values in  stalk-color-below-ring  :  9
Unique values for  stalk-color-below-ring  : ['w' 'p' 'g' 'b' 'n' 'e' 'y' 'o' 'c']

**** veil-type ****
No. of unique values in  veil-type  :  1
Unique values for  veil-type  : ['p']

**** veil-color ****
No. of unique values in  veil-color  :  4
Unique values for  veil-color  : ['w' 'n' 'o' 'y']

**** ring-number ****
No. of unique values in  ring-number  :  3
Unique values for  ring-number  : ['o' 't' 'n']
```

```
**** ring-type ****
No. of unique values in  ring-type  :  5
Unique values for  ring-type  : ['p' 'e' 'l' 'f' 'n']

**** spore-print-color ****
No. of unique values in  spore-print-color  :  9
Unique values for  spore-print-color  : ['k' 'n' 'u' 'h' 'w' 'r' 'o' 'y' 'b']

**** population ****
No. of unique values in  population  :  6
Unique values for  population  : ['s' 'n' 'a' 'v' 'y' 'c']

**** habitat ****
No. of unique values in  habitat  :  7
Unique values for  habitat  : ['u' 'g' 'm' 'd' 'p' 'w' 'l']
```

Independent and dependent features spliting

In [10]:
```
# split
X = data.iloc[:,1:]
y = data.iloc[:,0]  # data['class']
```

Train test split

In [11]:
```
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

In [12]:
```
data.columns
```

Out[12]:
```
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-type', 'veil-color', 'ring-number',
       'ring-type', 'spore-print-color', 'population', 'habitat'],
      dtype='object')
```

In [13]:
```
X_train.columns
```

Out[13]:
```
Index(['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-type', 'veil-color', 'ring-number',
       'ring-type', 'spore-print-color', 'population', 'habitat'],
      dtype='object')
```

OneHotEncoding

- We are using One Hot Encoding for categorical columns which having nominal values.

In [14]:

```
# one  hot encoding
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(drop='first',sparse=False,dtype=np.int32)
ohe.fit(X_train)
X_train_enc = ohe.transform(X_train)

# make dataframe
X_train_encode = pd.DataFrame(X_train_enc, columns = ohe.get_feature_names(X_train.columns))

# X_test
X_test_enc = ohe.transform(X_test)

# make dataframe for X_test_encode
X_test_encode = pd.DataFrame(X_test_enc, columns = ohe.get_feature_names(X_test.columns))
```

In [15]:

```
help(ohe.get_feature_names)
```

```
Help on method get_feature_names in module sklearn.preprocessing._encoders:

get_feature_names(input_features=None) method of sklearn.preprocessing._encoders.OneHotEncoder instance
    Return feature names for output features.

    Parameters
    ----------
    input_features : list of str of shape (n_features,)
        String names for input features if available. By default,
        "x0", "x1", ... "xn_features" is used.

    Returns
    -------
    output_feature_names : ndarray of shape (n_output_features,)
        Array of feature names.
```

In [16]:
```
X_train_encode.head()
```

Out[16]:

	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_g	cap-surface_s	cap-surface_y	cap-color_c	cap-color_e	cap-color_g	cap-color_n	cap-color_p	cap-color_r	cap-color_u	cap-color_w	cap-color_y
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
3	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0

label Encoding on target column

- We are using Label Encoder for encode the categorical target column's values.

In [17]:
```
# target column encoding by using LabelEncoder
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

# fit and transform on y_train
y_train_encode = le.fit_transform(y_train)

# only transform on y_test data
y_test_encode = le.transform(y_test)
```

In [18]:
```
y_train_encode.size
```

Out[18]:
```
6499
```

In [19]:
```
X_train_encode.shape
```

Out[19]:
```
(6499, 95)
```

Machine Learning Models

1) Logistic Regression

In [ ]:

In [20]:
```
from sklearn.linear_model import LogisticRegression
lg = LogisticRegression()
lg.fit(X_train_encode, y_train_encode)

y_pred_lg = lg.predict(X_test_encode)
```

In [21]:
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
cm = confusion_matrix(y_test_encode, y_pred_lg)
cp = classification_report(y_test_encode, y_pred_lg)
accuracy_score_lg = accuracy_score(y_test_encode, y_pred_lg)
```

In [22]:
```
# confusion matrix for Logistic Regression
print(cm)
```

```
[[831   0]
 [  0 794]]
```

In [23]:
```
# classification report for Logistic Regression
print(cp)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       831
           1       1.00      1.00      1.00       794

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

In [24]:
```
# Accuracy score for Logistic Regression
print(accuracy_score_lg)
```

```
1.0
```

2) Support vector machine

In [25]:
```
from sklearn.svm import SVC
clf = SVC()
clf.fit(X_train_encode, y_train_encode)
y_pred_svm = clf.predict(X_test_encode)
```

In [26]:
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
cm_svm = confusion_matrix(y_test_encode, y_pred_svm)
cp_svm = classification_report(y_test_encode, y_pred_svm)
accuracy_score_svm = accuracy_score(y_test_encode, y_pred_svm)
```

In [27]:
```
# confusion matrix for svm
print(cm_svm)
```

```
[[831   0]
 [  0 794]]
```

In [28]:
```
# classification report for svm
print(cp_svm)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       831
           1       1.00      1.00      1.00       794

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

In [29]:
```
# accuracy score for svm
print(accuracy_score_svm)
```

```
1.0
```

I have tried One Hot Encoding and Label Encoding on this dataset.

We can store both machine learning models we trained by using above data in the pickle file

In [30]:
```
import joblib

# store logistic regression model in pkl file
joblib.dump(lg, 'logistic_regression_model.pkl')

# store support vector classifier model in pkl file
joblib.dump(clf, 'support_vector_classifier_model.pkl')
```

Out[30]:
```
['support_vector_classifier_model.pkl']
```