```
#importing pandas for data manipulation
import pandas as pd
# importing seaborn library for data visualization
import seaborn as sns
# importing the matplotlib module for ploting the data
import matplotlib.pyplot as plt


#reading the dataset by calling pandas
data = pd.read_csv("mushroom.csv")


#collecting the information of the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54035 entries, 0 to 54034
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   cap-diameter     54035 non-null  int64  
 1   cap-shape        54035 non-null  int64  
 2   gill-attachment  54035 non-null  int64  
 3   gill-color       54035 non-null  int64  
 4   stem-height      54035 non-null  float64
 5   stem-width       54035 non-null  int64  
 6   stem-color       54035 non-null  int64  
 7   season           54035 non-null  float64
 8   class            54035 non-null  int64  
dtypes: float64(2), int64(7)
memory usage: 3.7 MB
```

```
#returns the shape of the data i.e. how many rowas and cols it contains
data.shape
```

```
(54035, 9)
```

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

```
cap-diameter       0
cap-shape          0
gill-attachment    0
gill-color         0
stem-height        0
stem-width         0
stem-color         0
season             0
class              0
dtype: int64
```

```
#describes the statistical measures of the data
data.describe()
```

	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	class
count	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000
mean	567.257204	4.000315	2.142056	7.329509	0.759110	1051.081299	8.418062	0.952163	0.549181
std	359.883763	2.160505	2.228821	3.200266	0.650969	782.056076	3.262078	0.305594	0.497580
min	0.000000	0.000000	0.000000	0.000000	0.000426	0.000000	0.000000	0.027372	0.000000
25%	289.000000	2.000000	0.000000	5.000000	0.270997	421.000000	6.000000	0.888450	0.000000
50%	525.000000	5.000000	1.000000	8.000000	0.593295	923.000000	11.000000	0.943195	1.000000
75%	781.000000	6.000000	4.000000	10.000000	1.054858	1523.000000	11.000000	0.943195	1.000000
max	1891.000000	6.000000	6.000000	11.000000	3.835320	3569.000000	12.000000	1.804273	1.000000

```
x = data.drop(['class'], axis=1)
#creating new set of the data by dropping the class and assigned to the variable x
y = data['class']
#creating the new set of the data by assigning the dropped column class to the new varaible y


x
```

	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season
0	1372	2	2	10	3.807467	1545	11	1.804273
1	1461	2	2	10	3.807467	1557	11	1.804273
2	1371	2	2	10	3.612496	1566	11	1.804273
3	1261	6	2	10	3.787572	1566	11	1.804273
4	1305	6	2	10	3.711971	1464	11	0.943195
...	...	...	...	...	...	...	...	...
54030	73	5	3	2	0.887740	569	12	0.943195
54031	82	2	3	2	1.186164	490	12	0.943195
54032	82	5	3	2	0.915593	584	12	0.888450
54033	79	2	3	2	1.034963	491	12	0.888450
54034	72	5	3	2	1.158311	492	12	0.888450

54035 rows × 8 columns

```
y
```

```
0        1
1        1
2        1
3        1
4        1
        ..
54030    1
54031    1
54032    1
54033    1
54034    1
Name: class, Length: 54035, dtype: int64
```

```
# split x and y into training and testing sets

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4, random_state = 0)


# check the shape of x_train and x_test

x_train.shape, x_test.shape
```

```
((32421, 8), (21614, 8))
```

```
#assigns the column names of the training data (x_train) to a variable named cols.
cols = x_train.columns
cols
```

```
Index(['cap-diameter', 'cap-shape', 'gill-attachment', 'gill-color',
       'stem-height', 'stem-width', 'stem-color', 'season'],
      dtype='object')
```

```
#importing the StandardScaler class from the preprocessing submodule of the scikit-learn library. This class is used for standardizing t
from sklearn.preprocessing import StandardScaler
#creates an instance of the StandardScaler class.
scaler = StandardScaler()
#fitting the standardscaler to x_train
x_train = scaler.fit_transform(x_train)
#takes the testing data (x_test) and transforms it using the same mean and standard deviation calculated earlier during the fit step of
x_test = scaler.transform(x_test)
#This line converts the standardized training data (now a NumPy array) back into a Pandas DataFrame with the original column names.
x_train = pd.DataFrame(x_train, columns=[cols])


# import SVC classifier
from sklearn.svm import SVC
# import metrics to compute accuracy
from sklearn.metrics import accuracy_score
# instantiate classifier with default hyperparameters
svc=SVC(kernel='linear', random_state=50)
# fit classifier to training set
svc.fit(x_train,y_train)
```

```
SVC
SVC(kernel='linear', random_state=50)
```

```
# make predictions on test set
y_pred=svc.predict(x_test)
```

```
# compute and print accuracy score
 # evaluating the model
accuracy = accuracy_score(y_test, y_pred)
# printing the accuracy of the model
print("Accuracy:", accuracy)
```

```
Accuracy: 0.6393078560192468
```

```
#generating a classification report
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
# printing the classification report
print("Classification Report:")
print(report)
```

```
Classification Report:
                  precision    recall  f1-score   support

               0       0.60      0.58      0.59      9721
               1       0.67      0.69      0.68     11893

        accuracy                           0.64     21614
       macro avg       0.63      0.63      0.63     21614
    weighted avg       0.64      0.64      0.64     21614
```

Start coding or generate with AI.