Name: Krish Sukhani

ML Lab 8

Mushroom Classification

Batch D

59

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression,PassiveAggressiveClassifier,RidgeClassifier,SGDClassifier
from sklearn.neighbors import KNeighborsClassifier,RadiusNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier
from sklearn.svm import LinearSVC, SVC,NuSVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate
from time import perf_counter
import matplotlib.pyplot as plt
import seaborn as sns

from IPython.display import Markdown, display

def printmd(string):
    # Print with Markdowns
    display(Markdown(string))

import warnings
warnings.filterwarnings(action='ignore')
```

1. Data Description

```
from google.colab import drive
drive.mount("/content/gdrive")
```

```
     Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive
```

```
df = pd.read_csv('/content/gdrive/My Drive/datasets/mushrooms.csv',encoding= 'unicode_escape')
```

```
from sklearn.preprocessing import LabelEncoder
def label_encoded(feat):
    le = LabelEncoder()
    le.fit(feat)
    print(feat.name,le.classes_)
#     print(le.classes_)
    return le.transform(feat)
```

```
for col in df.columns:
    df[str(col)] = label_encoded(df[str(col)])
```

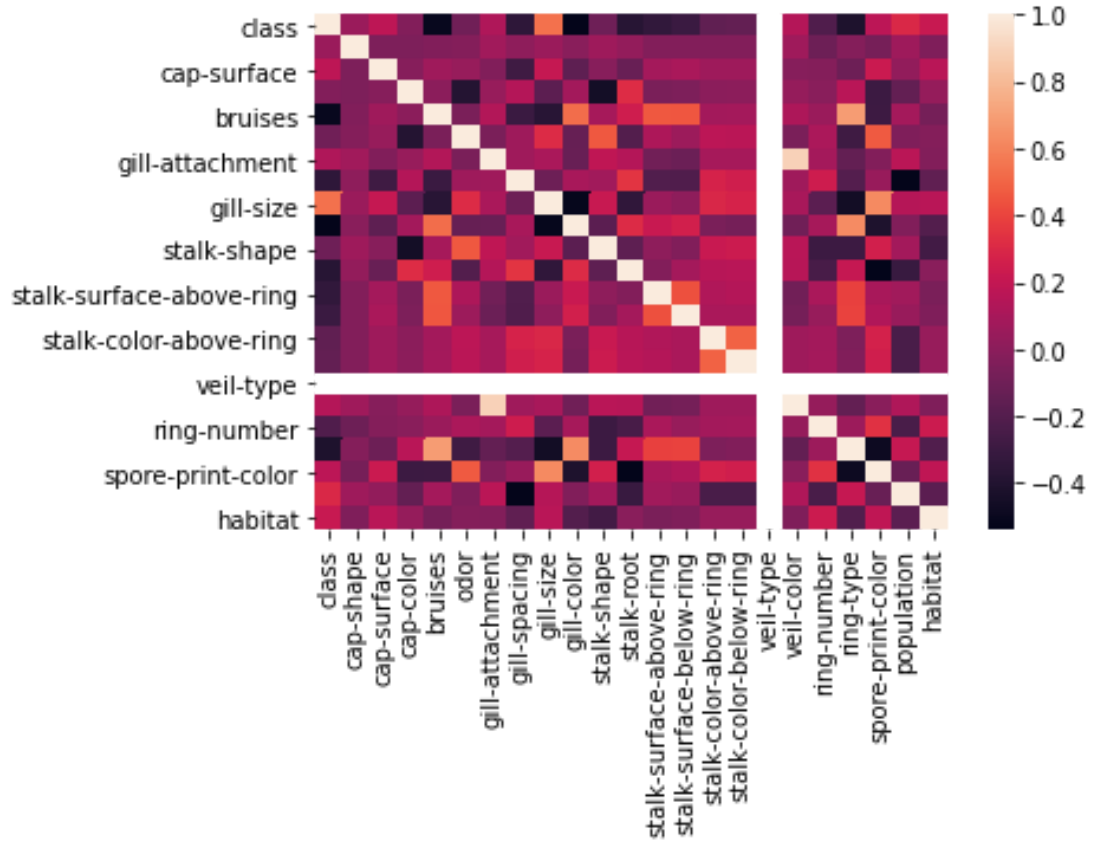
```
 class ['e' 'p']
 cap-shape ['b' 'c' 'f' 'k' 's' 'x']
 cap-surface ['f' 'g' 's' 'y']
 cap-color ['b' 'c' 'e' 'g' 'n' 'p' 'r' 'u' 'w' 'y']
 bruises ['f' 't']
 odor ['a' 'c' 'f' 'l' 'm' 'n' 'p' 's' 'y']
 gill-attachment ['a' 'f']
 gill-spacing ['c' 'w']
 gill-size ['b' 'n']
 gill-color ['b' 'e' 'g' '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']
```

```
df.head()
```

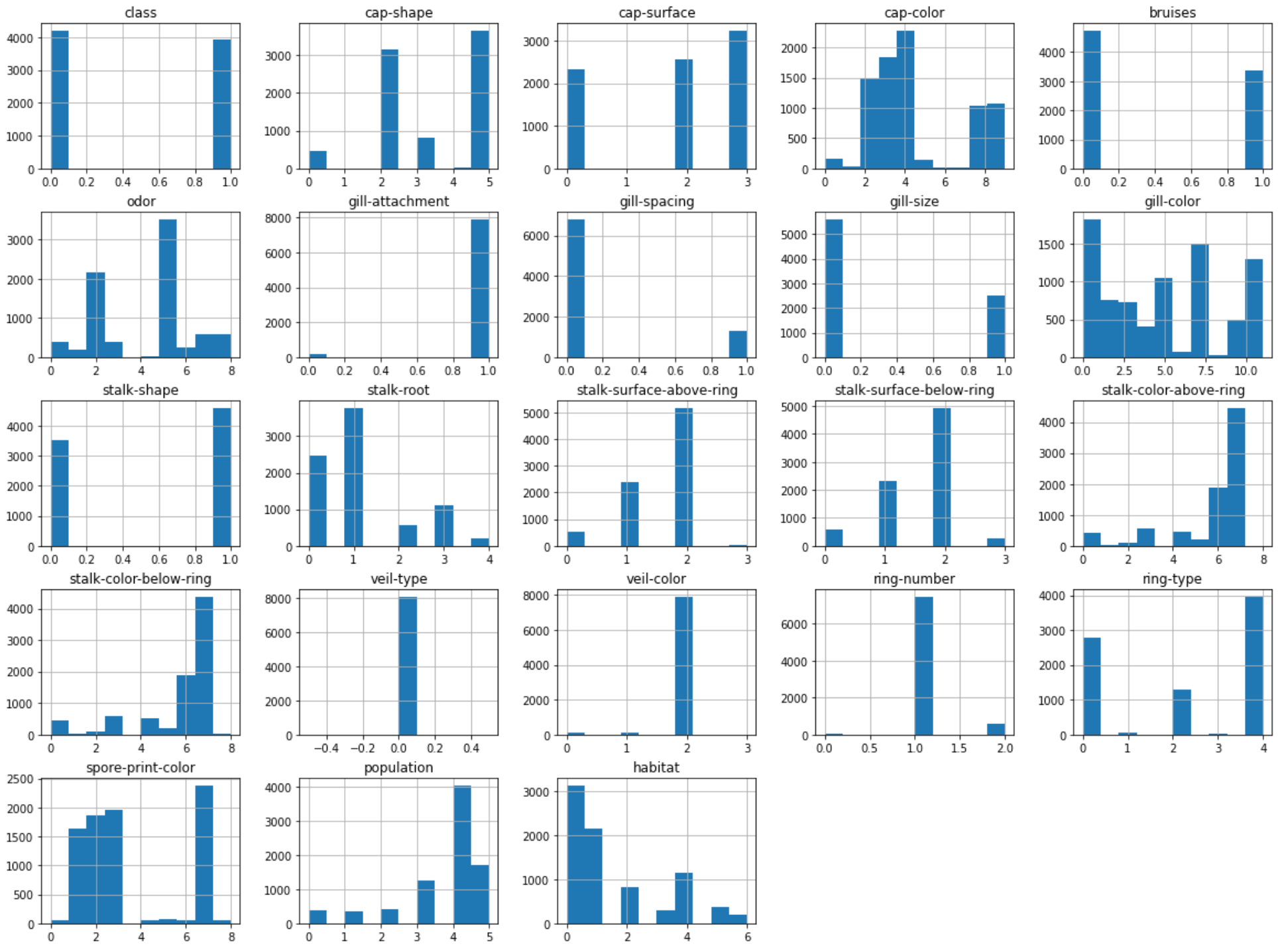
	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	stalk-root	stalk-surface-above-ring
0	1	5	2	4	1	6	1	0	1	4	0	3	2
1	0	5	2	9	1	0	1	0	0	4	0	2	2
2	0	0	2	8	1	3	1	0	0	5	0	2	2
3	1	5	3	8	1	6	1	0	1	5	0	3	2
4	0	5	2	3	0	5	1	1	0	4	1	3	2

```
sns.heatmap(df.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c2ff0d250>
```



```
fig = plt.figure(figsize = (20,15))
ax = fig.gca()
df.hist(ax=ax)
plt.show()
```




```
from google.colab import drive
drive.mount("/content/gdrive")
```

```
     Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive
```

```
df = pd.read_csv('/content/gdrive/My Drive/datasets/mushrooms.csv',encoding= 'unicode_escape')

# Change the names of the class to be more explicit
# with "edible" and "poisonous"
df['class'] = df['class'].map({"e": "edible", "p": "poisonous"})
# df.iloc[:5,:8]


# df.describe()
```

2. Data Preprocessing

```
df.shape
```

```
(8124, 23)
```

```
df.isnull()
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	stalk-root	sta surfa abo r
0	False	False	False	False	False	False	False	False	False	False	False	False	F
1	False	False	False	False	False	False	False	False	False	False	False	False	F
2	False	False	False	False	False	False	False	False	False	False	False	False	F
3	False	False	False	False	False	False	False	False	False	False	False	False	F
4	False	False	False	False	False	False	False	False	False	False	False	False	F
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8119	False	False	False	False	False	False	False	False	False	False	False	False	F
8120	False	False	False	False	False	False	False	False	False	False	False	False	F
8121	False	False	False	False	False	False	False	False	False	False	False	False	F
8122	False	False	False	False	False	False	False	False	False	False	False	False	F
8123	False	False	False	False	False	False	False	False	False	False	False	False	F

8124 rows × 23 columns

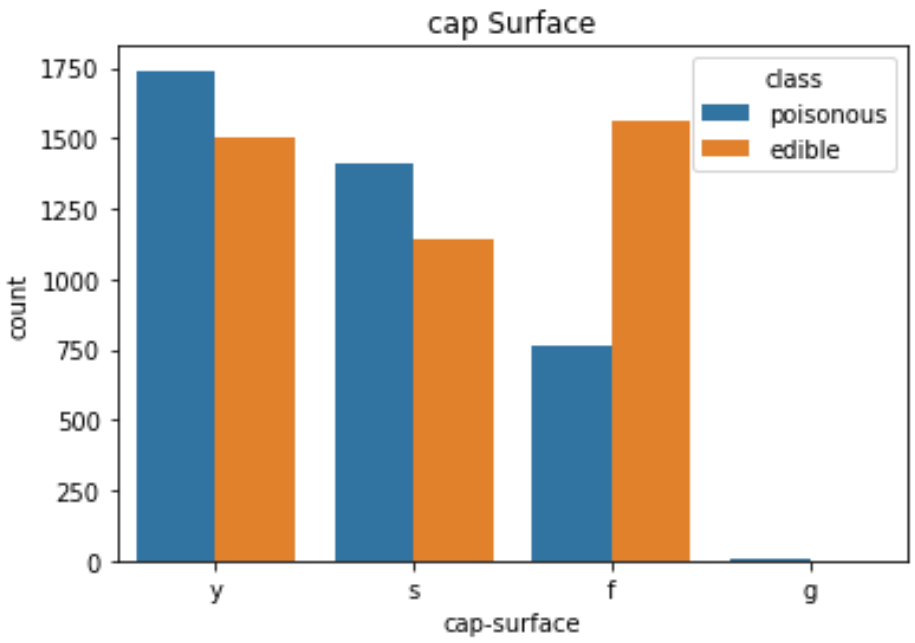
```
df.isna().sum()
```

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

```
import seaborn as sns
figsize=(15,8)
```

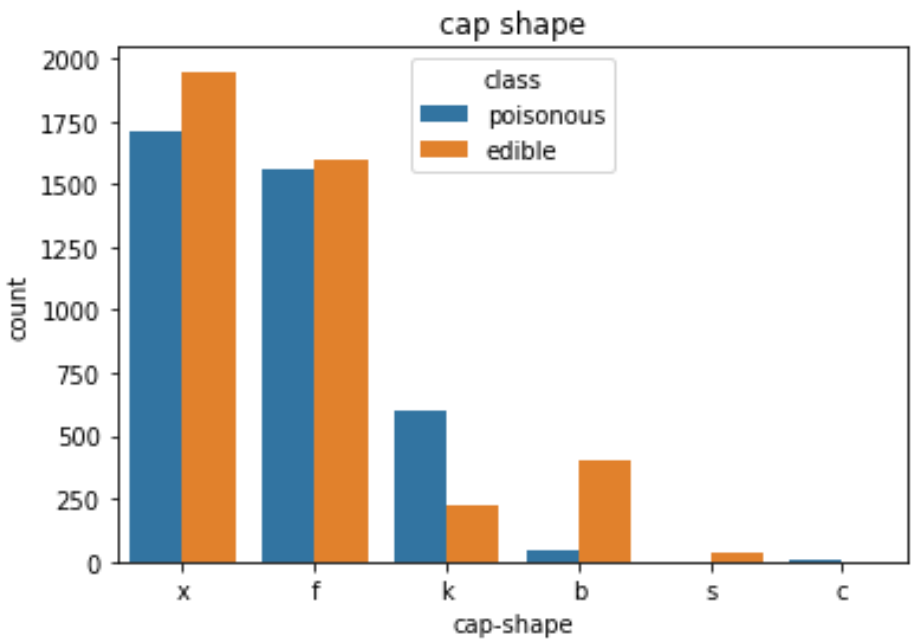
```
plt.title('cap Surface')
sns.countplot(data = df, x='cap-surface',hue='class', order=df['cap-surface'].value_counts().index,)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c277b9d10>
```



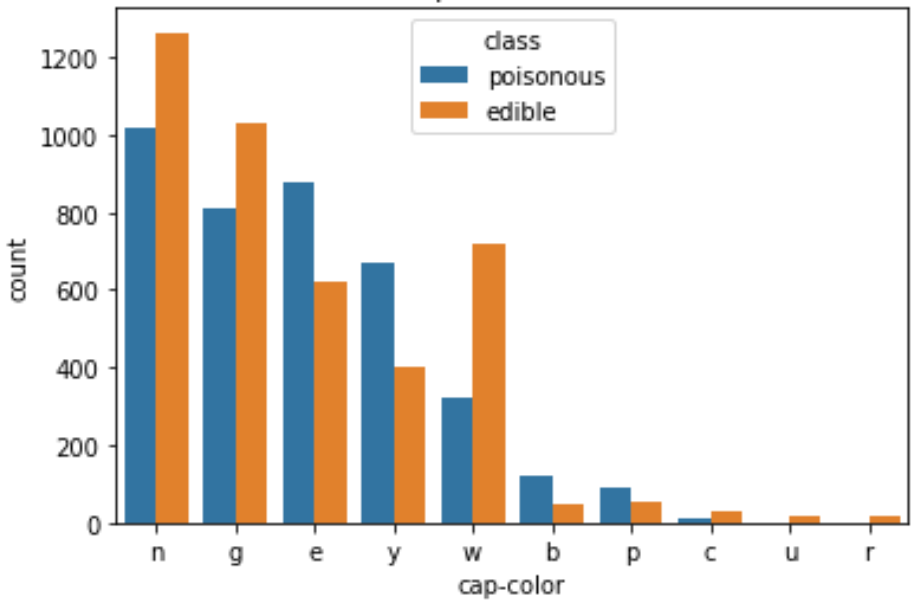

```
plt.title('cap shape')
sns.countplot(data = df, x='cap-shape',hue='class', order=df['cap-shape'].value_counts().index)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c275029d0>
```




```
plt.title('cap-color effect')
sns.countplot(data = df, x='cap-color',hue='class', order=df['cap-color'].value_counts().index,)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c26d62350>
```

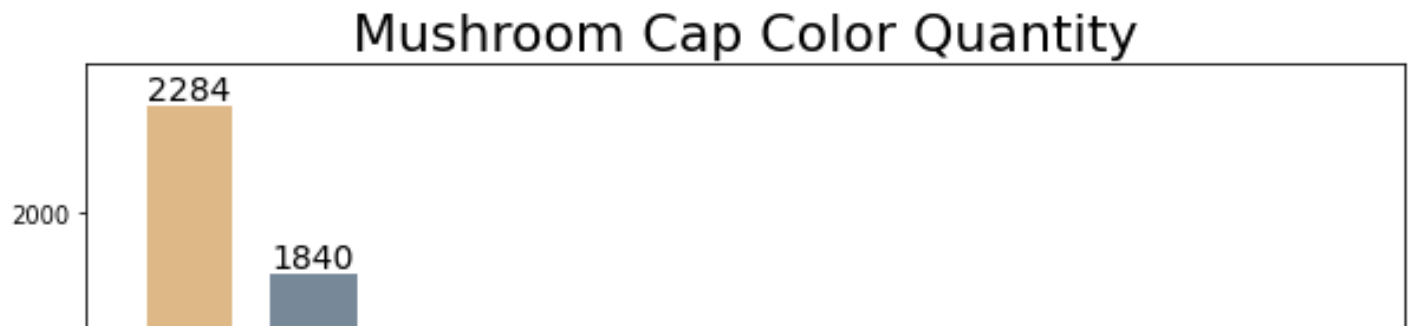



```
from sklearn.preprocessing import LabelEncoder

X = df.drop("class", axis = 1).copy()
y = df['class'].copy()

label_encoder_data = X.copy()
label_encoder = LabelEncoder()
```

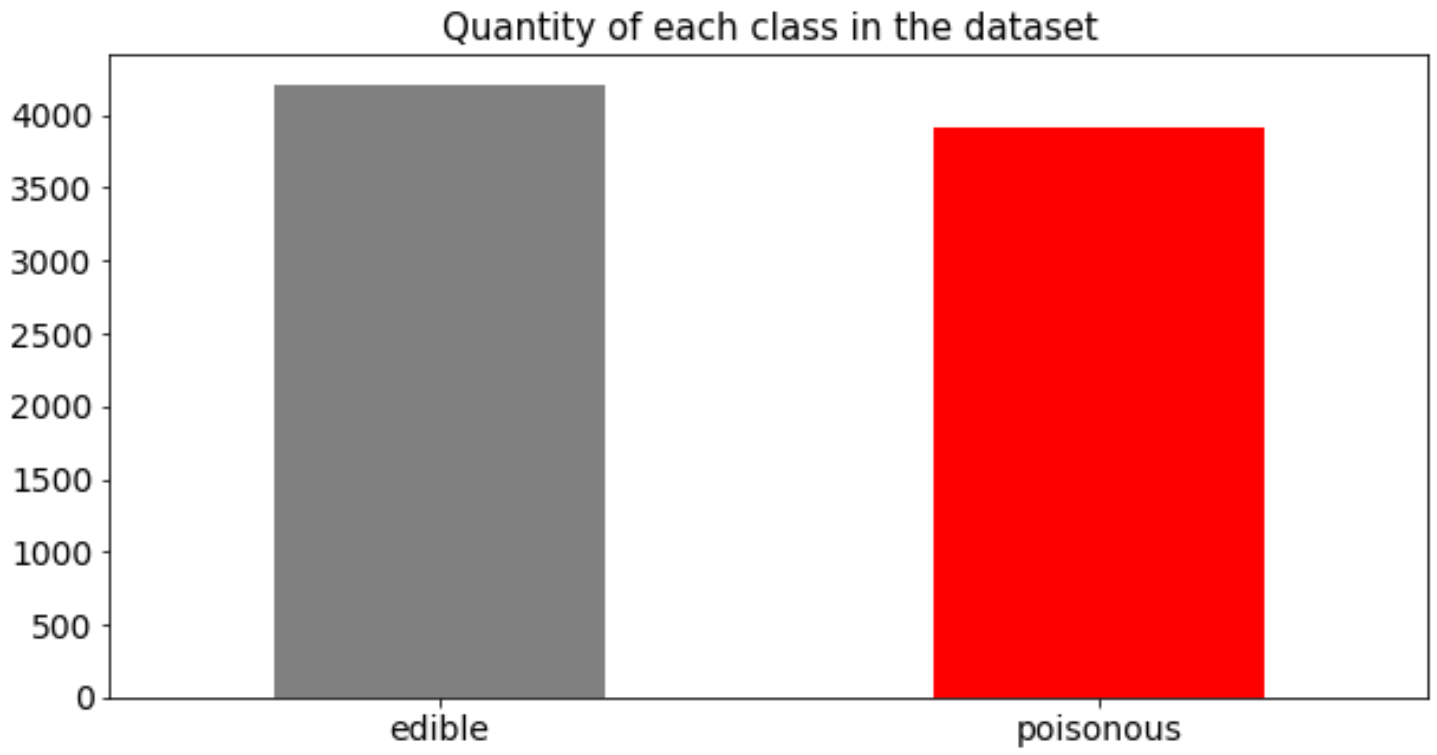
```
label_encoder = LabelEncoder()
for col in X.columns:
    label_encoder_data[col] = label_encoder.fit_transform(label_encoder_data[col])

X = label_encoder_data


# df.head()
```

2. Data Visualization

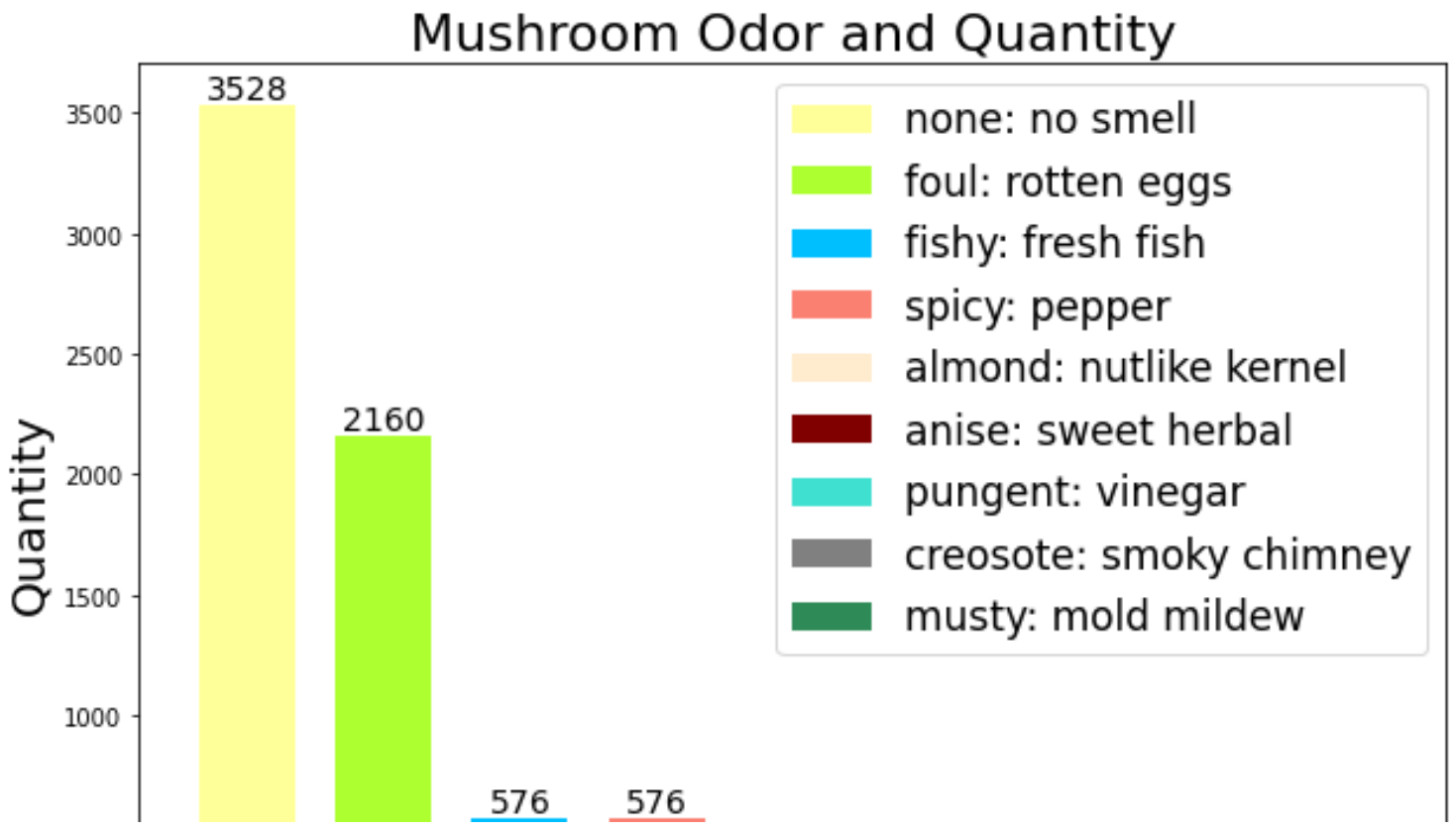
```
#Obtain total number of mushrooms for each 'cap-color' (Entire DataFrame)
cap_colors = df['cap-color'].value_counts()
m_height = cap_colors.values.tolist() #Provides numerical values
cap_colors.axes #Provides row labels
cap_color_labels = cap_colors.axes[0].tolist() #Converts index object to list

#=====PLOT Preparations and Plotting====#
ind = np.arange(10)  # the x locations for the groups
width = 0.7        # the width of the bars
colors = ['#DEB887','#778899','#DC143C','#FFFF99','#f8f8ff','#F0DC82','#FF69B4','#D22D1E','#C000C5','g']
#FFFFF0
fig, ax = plt.subplots(figsize=(10,7))
mushroom_bars = ax.bar(ind, m_height , width, color=colors)

#Add some text for labels, title and axes ticks
ax.set_xlabel("Cap Color",fontsize=20)
ax.set_ylabel('Quantity',fontsize=20)
ax.set_title('Mushroom Cap Color Quantity',fontsize=22)
ax.set_xticks(ind) #Positioning on the x axis
ax.set_xticklabels(('brown', 'gray','red','yellow','white','buff','pink','cinnamon','purple','green'),
                   fontsize = 12)

#Auto-labels the number of mushrooms for each bar color.
def autolabel(rects,fontsize=14):
    """
    Attach a text label above each bar displaying its height
    """
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., 1*height,'%d' % int(height),
                ha='center', va='bottom',fontsize=fontsize)
autolabel(mushroom_bars)
plt.show() #Display bars.
```




```
df['class'].value_counts().plot.bar(figsize = (10,5), color = ['grey','red'])
plt.xticks(rotation=0)
plt.title('Quantity of each class in the dataset', fontsize = 15)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```




```
#Obtain total number of mushrooms for each 'odor' (Entire DataFrame)
odors = df['odor'].value_counts()
odor_height = odors.values.tolist() #Provides numerical values
odor_labels = odors.axes[0].tolist() #Converts index labels object to list

#=====PLOT Preparations and Plotting====#
width = 0.7
ind = np.arange(9)  # the x locations for the groups
colors = ['#FFFF99','#ADFF2F','#00BFFF','#FA8072','#FFEBCD','#800000','#40E0D0','#808080','#2E8B57']

fig, ax = plt.subplots(figsize=(10,7))
odor_bars = ax.bar(ind, odor_height , width, color=colors)

#Add some text for labels, title and axes ticks
ax.set_xlabel("Odor",fontsize=20)
ax.set_ylabel('Quantity',fontsize=20)
ax.set_title('Mushroom Odor and Quantity',fontsize=22)
ax.set_xticks(ind) #Positioning on the x axis
ax.set_xticklabels(('none', 'foul','fishy','spicy','almond','anise','pungent','creosote','musty'),
                   fontsize = 12)
ax.legend(odor_bars, ['none: no smell','foul: rotten eggs', 'fishy: fresh fish','spicy: pepper',
                      'almond: nutlike kernel', 'anise: sweet herbal', 'pungent: vinegar',
                     'creosote: smoky chimney', 'musty: mold mildew'],fontsize=17)
autolabel(odor_bars)
plt.show() #Display bars.
```




```
#Get the population types and its values for Single Pie chart
populations = df['population'].value_counts()
pop_size = populations.values.tolist() #Provides numerical values
pop_types = populations.axes[0].tolist() #Converts index labels object to list
print(pop_size)
# Data to plot
pop_labels = 'Several', 'Solitary', 'Scattered', 'Numerous', 'Abundant', 'Clustered'
colors = ['#F38181','#EAFFD0','#95E1D3','#FCE38A','#BDE4F4','#9EF4E6']
explode = (0, 0.1, 0, 0, 0, 0)  # explode 1st slice
fig = plt.figure(figsize=(12,8))
# Plot
plt.title('Mushroom Population Type Percentange', fontsize=22)
patches, texts, autotexts = plt.pie(pop_size, explode=explode, labels=pop_labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=150)
for text,autotext in zip(texts,autotexts):
    text.set_fontsize(14)
    autotext.set_fontsize(14)

plt.axis('equal')
plt.show()
```

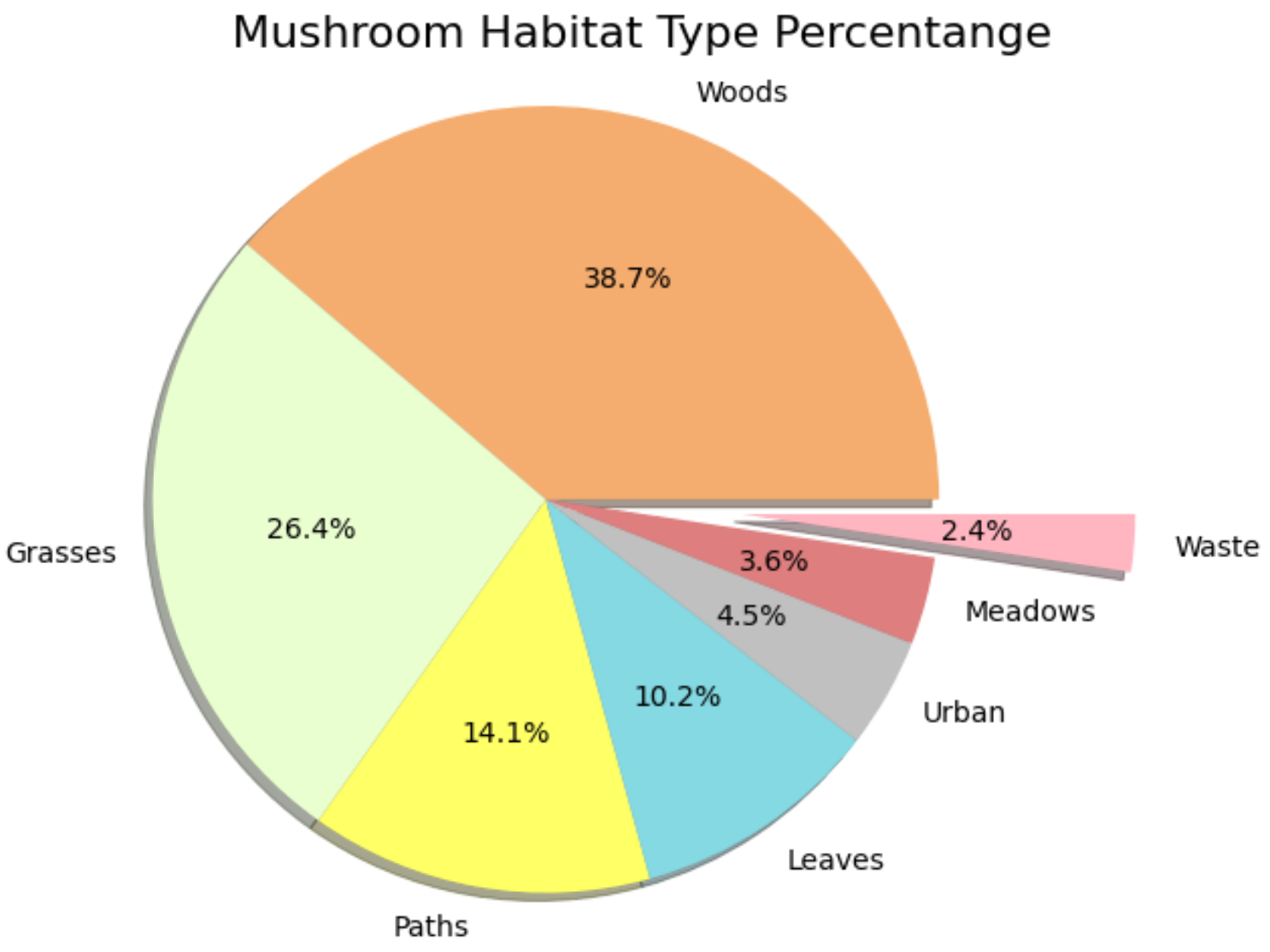
[4040 1712 1248 400 384 340]

Mushroom Habitat

Numerous Scattered

```
#Get the habitat types and its values for a Single Pie chart
habitats = df['habitat'].value_counts()
hab_size = habitats.values.tolist() #Provides numerical values
hab_types = habitats.axes[0].tolist() #Converts index labels object to list
print(habitats)
# Data to plot
hab_labels = 'Woods', 'Grasses', 'Paths', 'Leaves', 'Urban', 'Meadows', 'Waste'
colors = ['#F5AD6F','#EAFFD0','#FFFF66','#84D9E2','#C0C0C0','#DE7E7E', '#FFB6C1']
explode = (0, 0, 0, 0, 0, 0,0.5)  # explode 1st slice
fig = plt.figure(figsize=(12,8))
# Plot
plt.title('Mushroom Habitat Type Percentange', fontsize=22)
patches, texts, autotexts = plt.pie(hab_size, explode=explode, labels=hab_labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=360)
for text,autotext in zip(texts,autotexts):
    text.set_fontsize(14)
    autotext.set_fontsize(14)
plt.axis('equal')
plt.show()
```

```
d    3148
g    2148
p    1144
l     832
u     368
m     292
w     192
Name: habitat, dtype: int64
```



Double-click (or enter) to edit

```
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3. Model comparison using cross validation

```
# Create a dictionary with the model which will be tested
models = {
    "GaussianNB":{"model":GaussianNB()},
    "PassiveAggressiveClassifier":{"model":PassiveAggressiveClassifier() },
    "RidgeClassifier":{"model":RidgeClassifier() },
    "SGDClassifier":{"model":SGDClassifier() },
    "KNeighborsClassifier":{"model":KNeighborsClassifier() },
    "DecisionTreeClassifier":{"model":DecisionTreeClassifier() },
    "ExtraTreeClassifier":{"model":ExtraTreeClassifier() },
    "LinearSVC":{"model":LinearSVC() },
    "SVC":{"model":SVC() },
    "NuSVC":{"model":NuSVC() },
    "MLPClassifier":{"model":MLPClassifier() },
    "RandomForestClassifier":{"model":RandomForestClassifier() },
    "GradientBoostingClassifier":{"model":GradientBoostingClassifier() },
    "AdaBoostClassifier":{"model":AdaBoostClassifier() }
}

# Use the 10-fold cross validation for each model
# to get the mean validation accuracy and the mean training time
for name, m in models.items():
    # Cross validation of the model
    model = m['model']
    # print(model)
    result = cross_validate(model, X_train,y_train,cv = 10)

    # Mean accuracy and mean training time
    mean_val_accuracy = round( sum(result['test_score']) / len(result['test_score']), 4)
    mean_fit_time = round( sum(result['fit_time']) / len(result['fit_time']), 4)

    # Add the result to the dictionary witht he models
    m['val_accuracy'] = mean_val_accuracy
    m['Training time (sec)'] = mean_fit_time

    # Display the result
    print(f"{name:27} accuracy : {mean_val_accuracy*100:.2f}% - mean training time {mean_fit_time} sec")
```

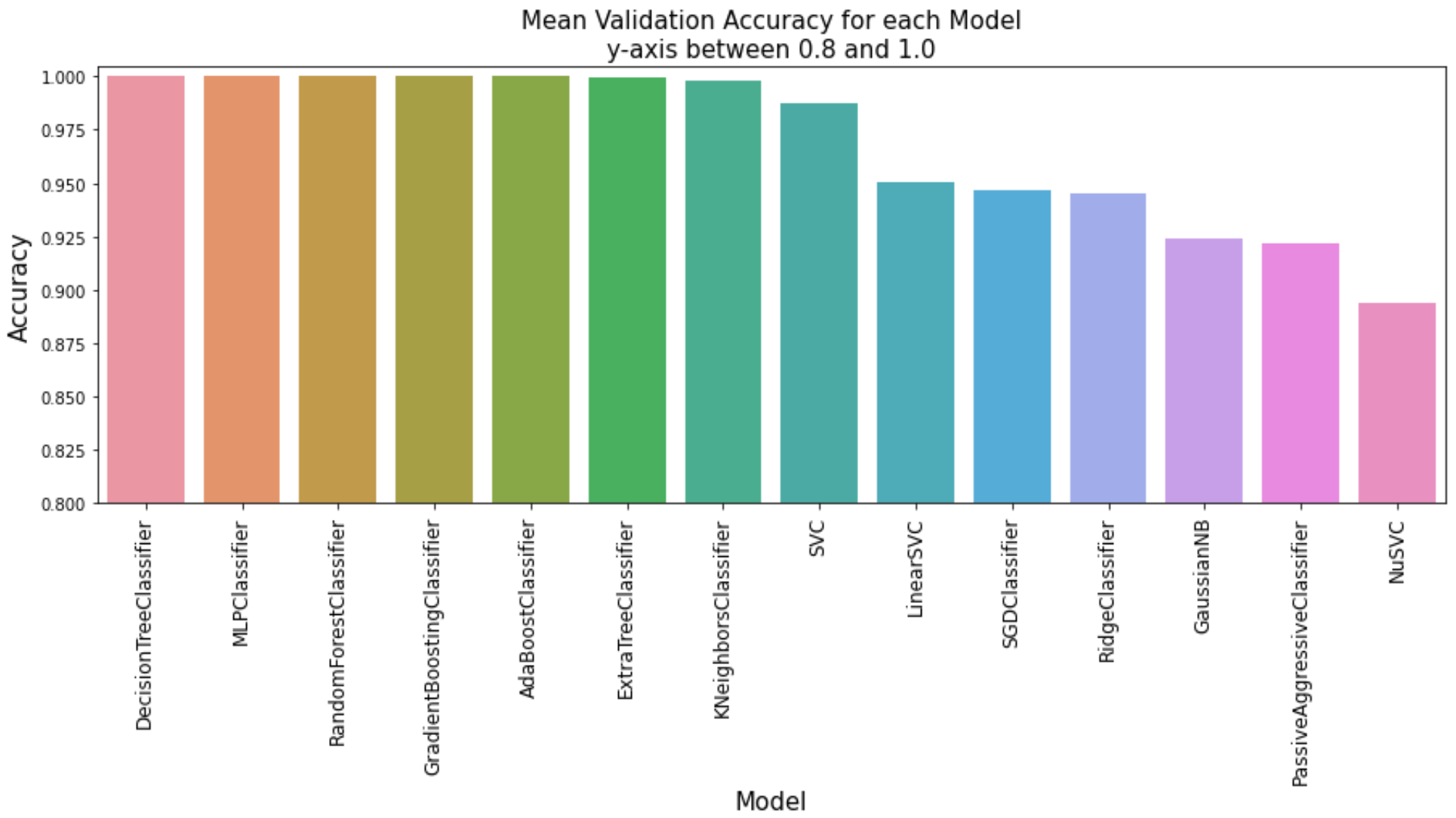
```
 GaussianNB                  accuracy : 92.40% - mean training time 0.0126 sec
 PassiveAggressiveClassifier accuracy : 92.21% - mean training time 0.0276 sec
 RidgeClassifier             accuracy : 94.54% - mean training time 0.0398 sec
 SGDClassifier               accuracy : 94.69% - mean training time 0.0941 sec
 KNeighborsClassifier        accuracy : 99.80% - mean training time 0.0244 sec
 DecisionTreeClassifier      accuracy : 100.00% - mean training time 0.0152 sec
 ExtraTreeClassifier         accuracy : 99.95% - mean training time 0.0098 sec
 LinearSVC                   accuracy : 95.06% - mean training time 0.4615 sec
 SVC                         accuracy : 98.75% - mean training time 0.3627 sec
 NuSVC                       accuracy : 89.40% - mean training time 1.8529 sec
 MLPClassifier               accuracy : 100.00% - mean training time 3.6505 sec
 RandomForestClassifier      accuracy : 100.00% - mean training time 0.316 sec
 GradientBoostingClassifier  accuracy : 100.00% - mean training time 0.642 sec
 AdaBoostClassifier          accuracy : 100.00% - mean training time 0.5504 sec
```

```
# Create a DataFrame with the results
models_result = []

for name, v in models.items():
    lst = [name, v['val_accuracy'],v['Training time (sec)']]
    models_result.append(lst)

df_results = pd.DataFrame(models_result,
                          columns = ['model','val_accuracy','Training time (sec)'])
df_results.sort_values(by='val_accuracy', ascending=False, inplace=True)
```

```
df_results.reset_index(inplace=True,drop=True)
df_results
```

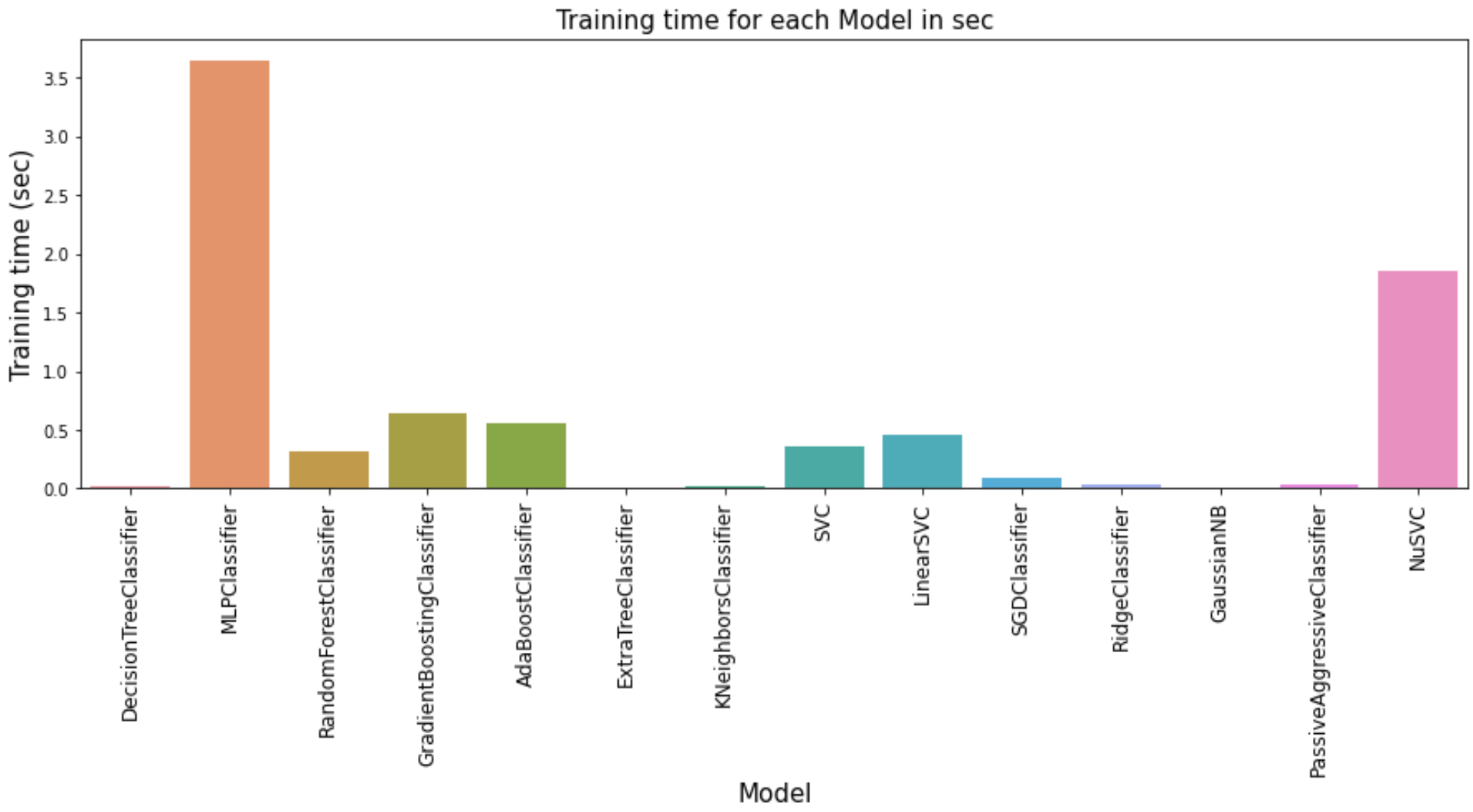
	model	val_accuracy	Training time (sec)
0	DecisionTreeClassifier	1.0000	0.0152
1	MLPClassifier	1.0000	3.6505
2	RandomForestClassifier	1.0000	0.3160
3	GradientBoostingClassifier	1.0000	0.6420
4	AdaBoostClassifier	1.0000	0.5504
5	ExtraTreeClassifier	0.9995	0.0098
6	KNeighborsClassifier	0.9980	0.0244
7	SVC	0.9875	0.3627
8	LinearSVC	0.9506	0.4615
9	SGDClassifier	0.9469	0.0941
10	RidgeClassifier	0.9454	0.0398
11	GaussianNB	0.9240	0.0126
12	PassiveAggressiveClassifier	0.9221	0.0276
13	NuSVC	0.8940	1.8529

```
plt.figure(figsize = (15,5))
sns.barplot(x = 'model', y = 'val_accuracy', data = df_results)
plt.title('Mean Validation Accuracy for each Model\ny-axis between 0.8 and 1.0', fontsize = 15)
plt.ylim(0.8,1.005)
plt.xlabel('Model', fontsize=15)
plt.ylabel('Accuracy',fontsize=15)
plt.xticks(rotation=90, fontsize=12)
plt.show()
```




```
plt.figure(figsize = (15,5))
sns.barplot(x = 'model', y = 'Training time (sec)', data = df_results)
```

```
plt.title('Training time for each Model in sec', fontsize = 15)
plt.xticks(rotation=90, fontsize=12)
plt.xlabel('Model', fontsize=15)
plt.ylabel('Training time (sec)',fontsize=15)
plt.show()
```

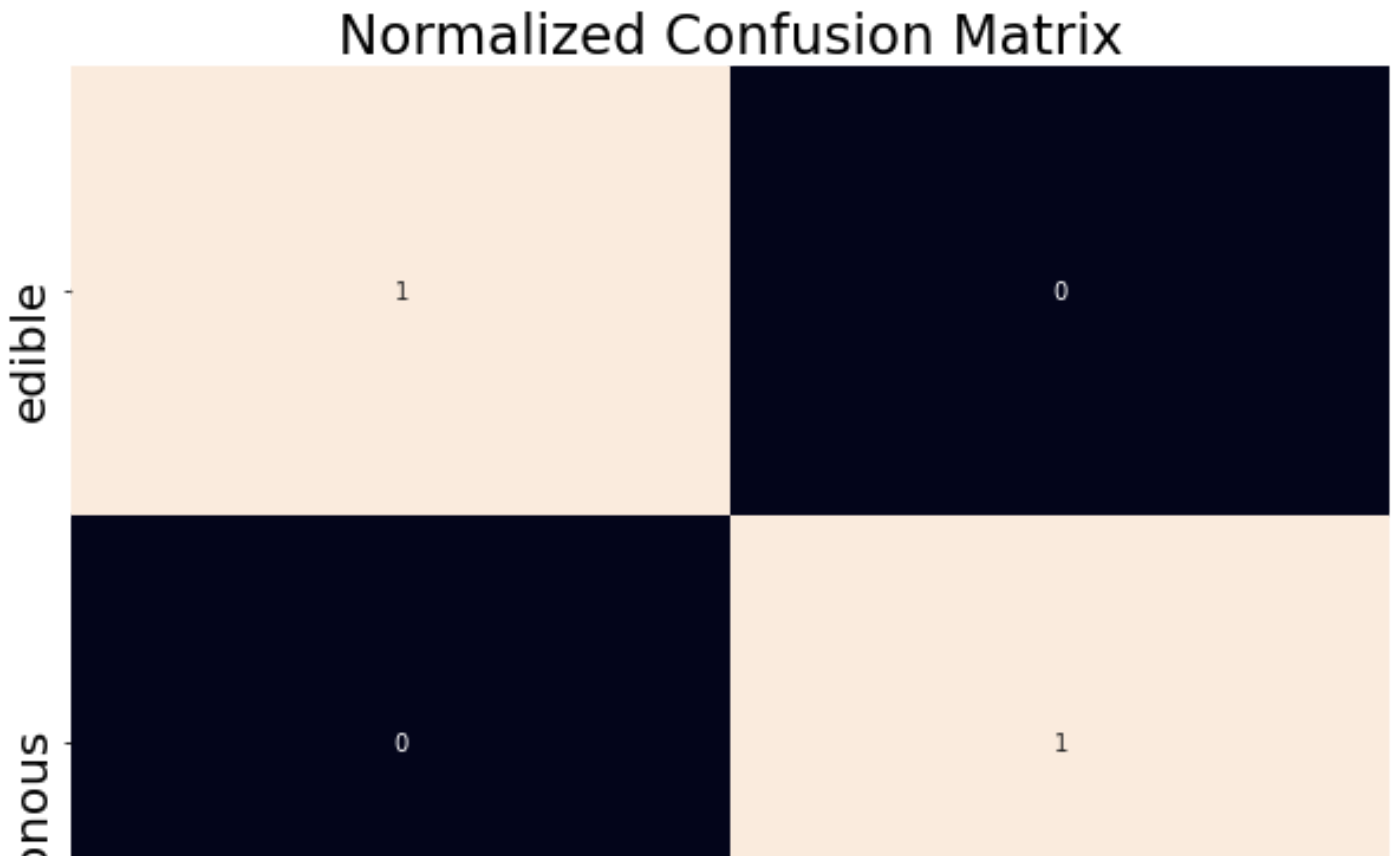


4. Prediction metrics of the best model using the test set

```
# Get the model with the highest mean validation accuracy
best_model = df_results.iloc[0]

# Fit the model
model = models[best_model[0]]['model']
model.fit(X_train,y_train)

# Predict the labels with the data set
pred = model.predict(X_test)

# Display the results
printmd(f'## Best Model: {best_model[0]} with {best_model[1]*100}% accuracy on the test set')
printmd(f'## Trained in: {best_model[2]} sec')

# Display a confusion matrix
from sklearn.metrics import confusion_matrix
cf_matrix = confusion_matrix(y_test, pred, normalize='true')
plt.figure(figsize = (10,7))
sns.heatmap(cf_matrix, annot=True, xticklabels = sorted(set(y_test)), yticklabels = sorted(set(y_test)),cbar=Fal
plt.title('Normalized Confusion Matrix', fontsize = 23)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()
```

Best Model: DecisionTreeClassifier with 100.0% accuracy on the test set

Trained in: 0.0152 sec



Conclusion : Hence we compared the different models and hence we get the output as the Decision tree classifier to be the Best Classifier for this dataset