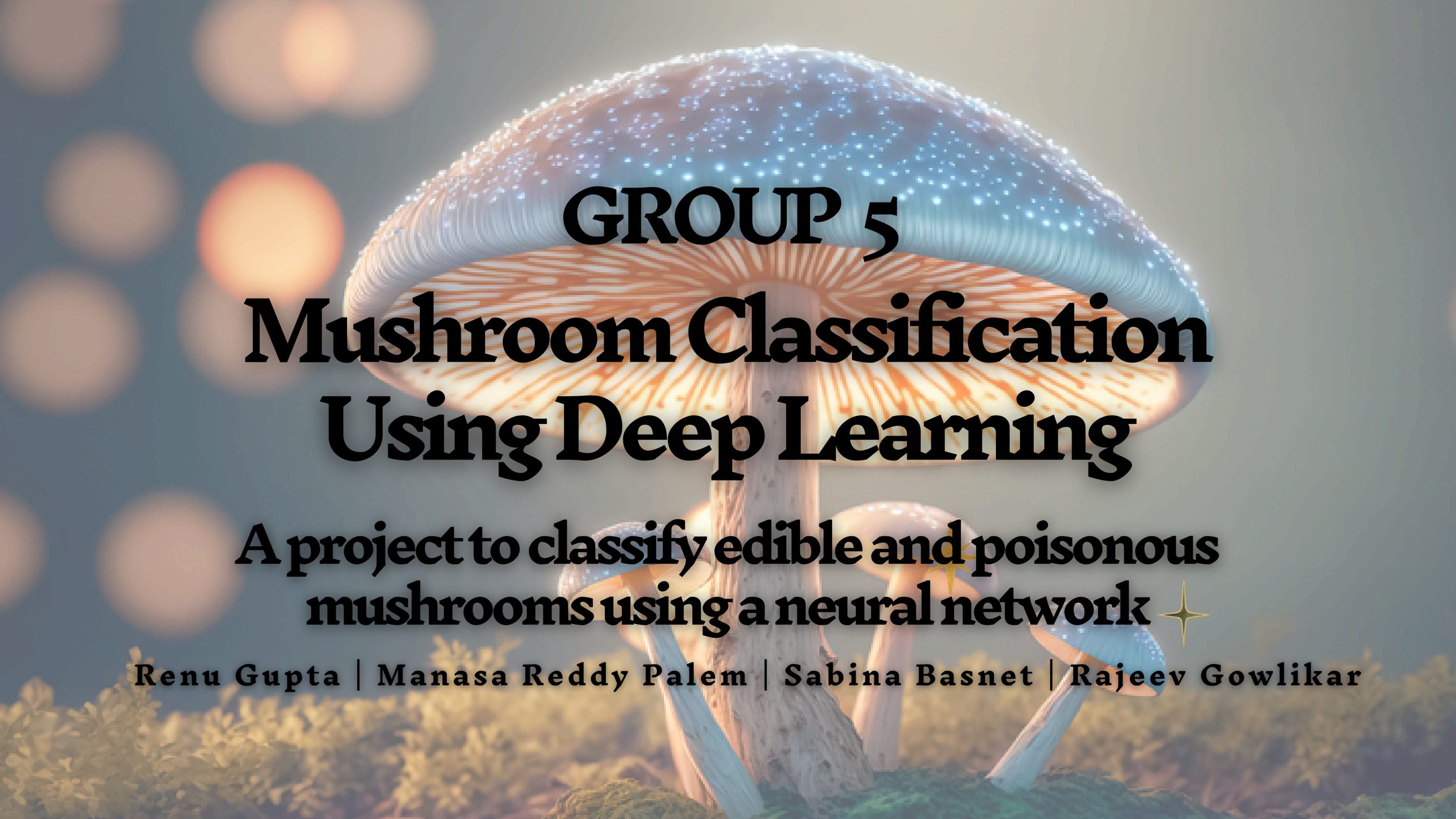


GROUP 5

Mushroom Classification Using Deep Learning

A project to classify edible and poisonous mushrooms using a neural network

Renu Gupta | Manasa Reddy Palem | Sabina Basnet | Rajeev Gowlikar

Presentation Outline

1. **Problem Statement**
2. **Mushroom Dataset**
3. **Exploratory Data Analysis (EDA)**
4. **Data Split | Standardization | One-Hot Encoding**
5. **Building and Training the Model**
6. **Model Testing**
7. **Model Evaluation**
8. **New Predictions**

PROBLEM STATEMENT

Gathering mushrooms can be an extremely rewarding and interesting hobby. However, those who do it must proceed with the utmost caution.

Throughout history, people around the world have foraged mushrooms for food. Though many mushrooms are highly nutritious, delicious, and safe to consume, others pose a serious risk to your health and can even cause death if ingested.

For this reason, it's critical to identify edible and poisonous mushrooms before hunting.


MUSHROOM DATASET

https://www.kaggle.com/datasets/prishasawhney/mushroom-dataset?resource=download

Overview

Cleaned version of the original Mushroom Dataset available at UCI Library.

It includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family.

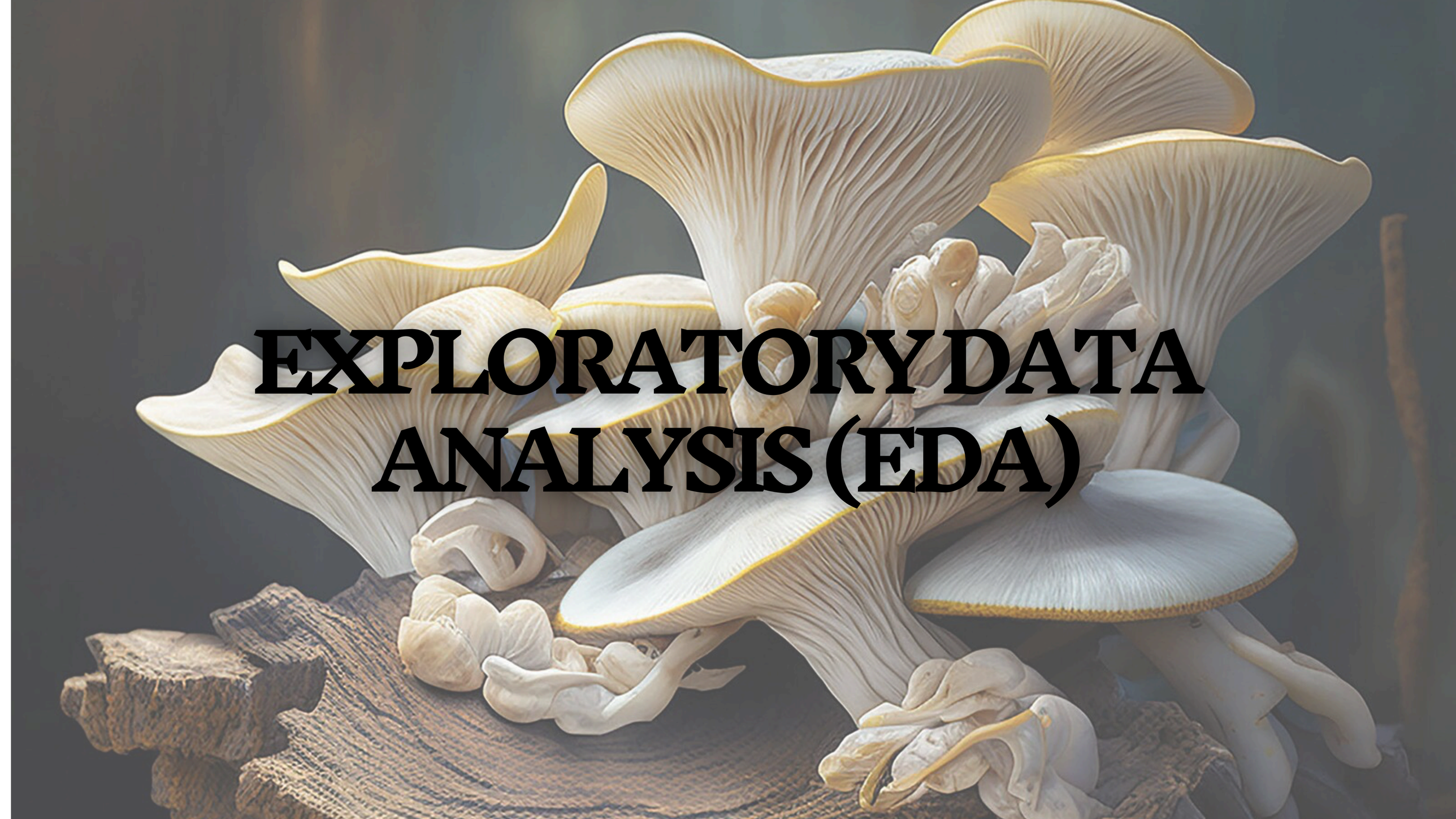
Each species is identified as edible(e) or poisonous(p).

Feature Variables:

1. **Cap Diameter**
2. **Cap Shape**
3. **Gill Attachment**
4. **Gill Color**
5. **Stem Height**
6. **Stem Width**
7. **Stem Color**
8. **Season**

Target Class:

1. **Edible: “1”**
2. **Poisonous: “0”**


EXPLORATORY DATA
ANALYSIS (EDA)

Libraries

```
# Required Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Input
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
```

To understand the dataset, let's just see the first few rows.

```
# Load the Mushroom dataset
df = pd.read_csv('mushroom_cleaned.csv')

# View the dataset
df.sample(n=10).head(10)
```

	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	class
43273	1059	6	4	11	1.093796	2257	6	0.888450	0
40390	315	0	0	4	0.134860	231	11	1.804273	0
39712	858	2	2	4	3.314072	1171	11	0.888450	0
30269	563	6	2	7	0.422199	975	6	0.943195	0
29143	562	5	1	6	0.282083	1801	11	0.888450	0
26563	616	5	1	5	0.632234	985	6	0.943195	1
42169	140	6	0	5	0.469946	219	6	0.888450	1
33723	437	6	6	10	0.874952	912	6	0.943195	0
35229	54	0	0	4	1.102606	203	6	0.943195	1
29097	841	5	1	7	1.141544	1667	8	0.943195	1

Columns

```
# Display columns
df.columns.tolist()
```

Output:

```
['cap-diameter',
 'cap-shape',
 'gill-attachment',
 'gill-color',
 'stem-height',
 'stem-width',
 'stem-color',
 'season',
 'class']
```

Shape

```
# Display the shape
df.shape
```

Output: `(53720, 9)`

Number of Rows: 53732

Number of Columns: 9

Basic Information

```
# Display basic information
df.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54023 entries, 0 to 54022
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   cap-diameter     54023 non-null  int64
 1   cap-shape        54023 non-null  int64
 2   gill-attachment  54023 non-null  int64
 3   gill-color       54023 non-null  int64
 4   stem-height      54023 non-null  float64
 5   stem-width       54023 non-null  int64
 6   stem-color       54023 non-null  int64
 7   season           54023 non-null  float64
 8   class            54023 non-null  int64
dtypes: float64(2), int64(7)
memory usage: 3.7 MB
```

Missing Values

```
# Check for missing values
df.isnull().sum()
```

Output:

```
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
```

There are no missing values in the dataset.

Drop Duplicate Values

```
# Drop duplicates
df=df.drop_duplicates()
```

Summary Statistics

```
# Display summary statistics
df.describe()
```

Output:

	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	class
count	53720.000000	53720.000000	53720.000000	53720.000000	53720.000000	53720.000000	53720.000000	53720.000000	53720.000000
mean	568.705473	4.005957	2.142889	7.345346	0.752148	1057.097040	8.453555	0.952311	0.546649
std	360.381837	2.165262	2.232745	3.190272	0.646018	780.321241	3.235577	0.303806	0.497824
min	0.000000	0.000000	0.000000	0.000000	0.000426	0.000000	0.000000	0.027372	0.000000
25%	290.000000	2.000000	0.000000	5.000000	0.270146	430.000000	6.000000	0.888450	0.000000
50%	528.000000	5.000000	1.000000	8.000000	0.589316	929.500000	11.000000	0.943195	1.000000
75%	783.000000	6.000000	4.000000	10.000000	1.046900	1527.000000	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

The output provides descriptive statistics for each feature in the dataset :

Count : Number of non-null entries for each feature.
Mean : Average value of each feature.
Std : Standard deviation, indicating the spread of values around the mean.
Min : Minimum value observed for each feature.
25% : 25th percentile (first quartile) of the data.
50% : 50th percentile (median) of the data.
75% : 75th percentile (third quartile) of the data.
Max : Maximum value observed for each feature.

Class Distribution

```
# Display entries of the output column - 'Class'
df['class'].value_counts()
```

Output:

```
class
1    29366
0    24354
Name: count, dtype: int64
```

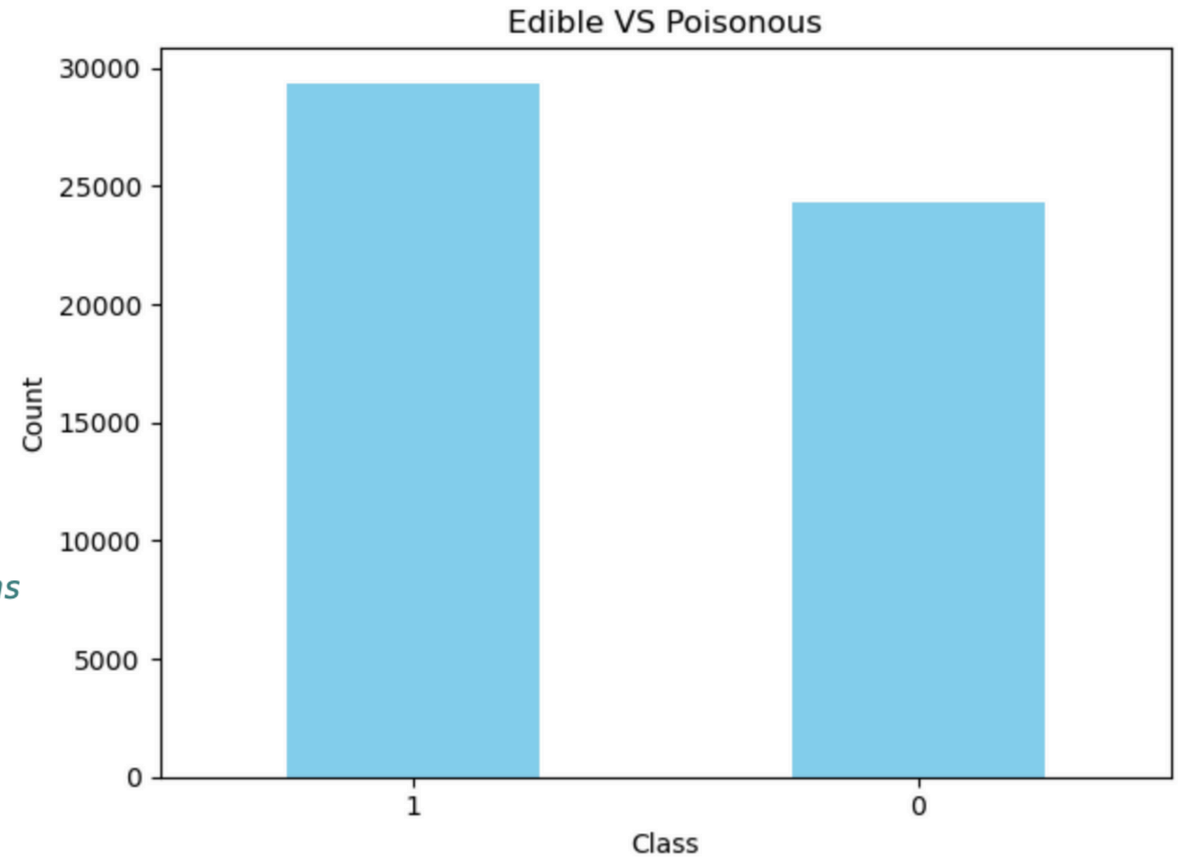
1 = Edible → 54.67%

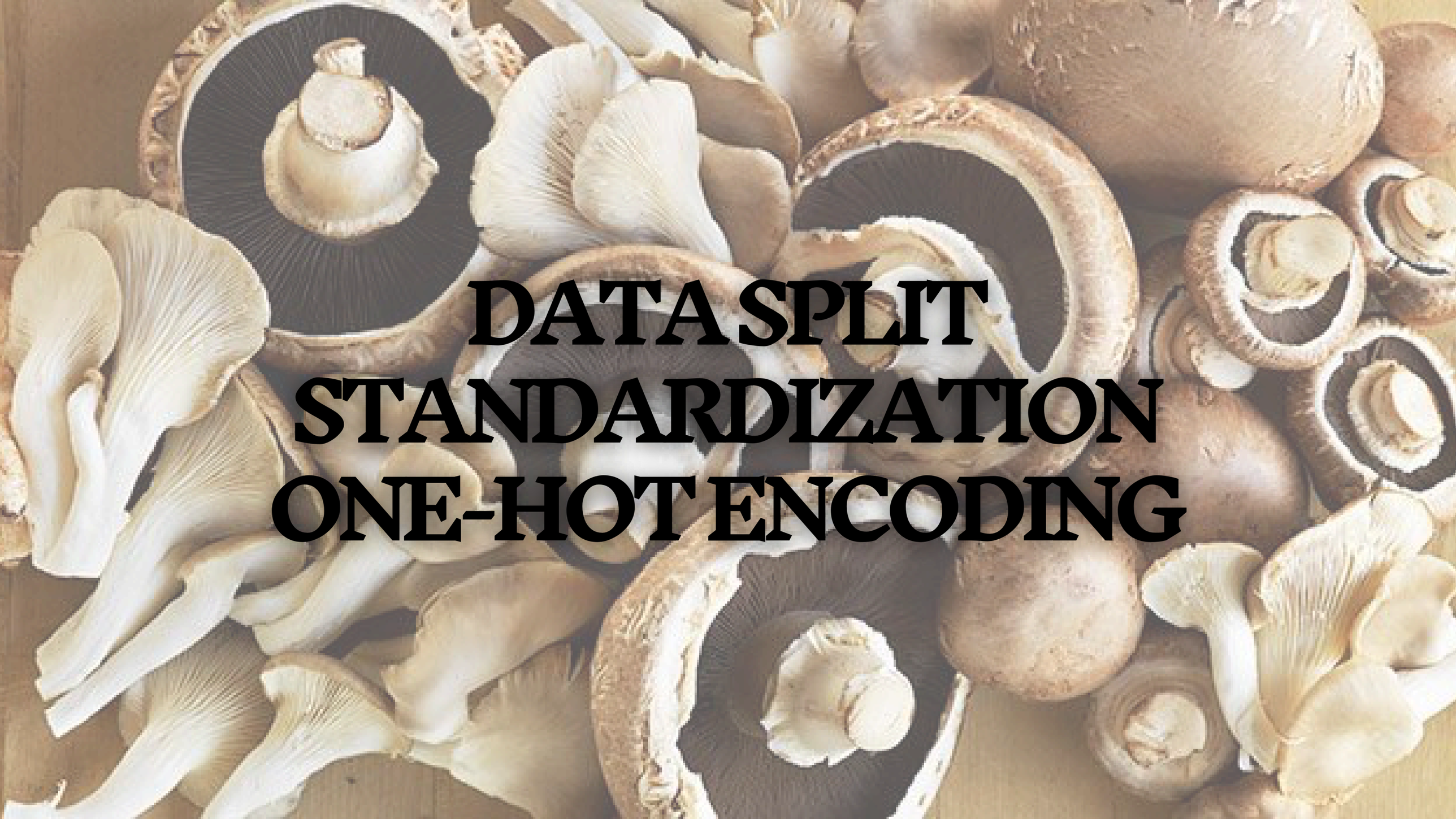
0 = Poisonous → 45.32%

Visualization of Class Distribution

```
# Display entries of the output column - 'Class'
df['class'].value_counts()

# Visualize distributions of edible and poisonous mushrooms
target_column = 'class'
target_balance = df[target_column].value_counts()
target_balance.plot(kind='bar', color='skyblue')
plt.title('Edible VS Poisonous')
plt.xlabel('Class')
plt.ylabel('Count')
plt.xticks(rotation = 0)
plt.grid(axis = 'y', linestyle = '--', alpha = 0.7)
plt.tight_layout()
plt.show()
```





DATA SPLIT
STANDARDIZATION
ONE-HOT ENCODING

Data Split

```
# Splitting the dataset into training and testing sets (80% for training and 20% for testing.)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, shuffle=True, random_state=1)
```

- This function splits the dataset (X and y) into training and testing sets (X_train, X_test, y_train, y_test).
- It allocates 80% of the data to training (X_train, y_train) and 20% to testing (X_test, y_test).
- The shuffle = True parameter ensures that the data is shuffled before splitting, and random_state = 1 sets a seed for reproducibility.

Output:

```
X_train.shape=(42976, 8)
X_test.shape=(10744, 8)
y_train.shape=(42976, 1)
y_test.shape=(10744, 1)
```

- There are 42,976 samples in the training data, each with 8 features.
- There are 10,744 samples in the testing data, each with 8 features.

Standardizing the Features

```
# Standardizing the features such that it has a mean of 0 and a standard deviation of 1.
X_scaler = StandardScaler().fit(X_train)
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

- **Fitting the Scaler:**
  X_scaler = StandardScaler().fit(X_train): Calculates the mean and standard deviation for each feature in the training set.
- **Transforming the Training and TestingData:**
  X_train_scaled = X_scaler.transform(X_train): Applies the calculated mean and standard deviation to scale the training data.
  X_test_scaled = X_scaler.transform(X_test): Uses the same scaling parameters from the training data to scale the testing data.

Output:

- **Standardization scales the features to have a mean of 0 and a standard deviation of 1.**
- **This ensures that each feature contributes equally to the model's performance, preventing any feature from dominating due to its scale.**

One-Hot Encoding the Target Variable

```
# One-hot encoding the target variable
y_train_categorical = to_categorical(y_train)
y_test_categorical = to_categorical(y_test)

y_train_categorical
```

Transforms the training and testing target variable (Class) into a binary matrix.

Output:

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

BUILDING AND TRAINING THE MODEL

Building the Model

```
# Initializing a Sequential model
model = Sequential()

# Adding layers to the model:
model.add(Input(shape = (8,)))  # Input layer with shape (8,)
model.add(Dense(units = 100, activation='relu'))
model.add(Dense(units = 100, activation='relu'))
model.add(Dense(units = 2, activation='softmax'))

# Compiling the model
model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])

# Printing the summary of the model
model.summary()
```

Model Summary

Model: "sequential_7"

Layer (type)	Output Shape	Param #
dense_21 (Dense)	(None, 100)	900
dense_22 (Dense)	(None, 100)	10,100
dense_23 (Dense)	(None, 90)	9,090
dense_24 (Dense)	(None, 2)	182

Total params: 20,272 (79.19 KB)

Trainable params: 20,272 (79.19 KB)

Non-trainable params: 0 (0.00 B)

Training the Model

```
# Fit (train) the model
model.fit(X_train_scaled, y_train_categorical,
    epochs = 100,
    batch_size = 60,
    shuffle = True,
    verbose = 1)
```

Training Accuracy

```
Epoch 100/100
721/721 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9907 - loss: 0.0261
<keras.src.callbacks.history.History at 0x13d886ad0>
```

TESTING THE MODEL

```
# Evaluate the model using the testing data
model_loss, model_accuracy = model.evaluate(X_test_scaled, y_test_categorical, verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

Output:

```
338/338 - 0s - 1ms/step - accuracy: 0.9907 - loss: 0.0241
Loss: 0.024064391851425171, Accuracy: 0.9906525015830994
```

Making Predictions with new data

```
new_data = pd.read_csv('new_data.csv')

# Preprocess the new data
X_new = new_data.values[:, 0:8]  # Assuming the features are in the same format as the original data
X_new_scaled = X_scaler.transform(X_new)  # 'X_scaler' is the StandardScaler object fitted on the training data

# Make predictions
predictions = model.predict(X_new_scaled)

# Convert predictions to class labels
predicted_classes = np.argmax(predictions, axis=1)

# Print the predicted classes
print("Predicted classes for new data:")
print(predicted_classes)
```

Output:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 358ms/step
Predicted classes for new data:
[1 1 1 1 1 1 0 0 0 0 0 0]
```

Prediction

Actual: 111111000000

cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	Class
97	6	3	2	0.509736286	527	12	0.943194554	1
73	5	3	2	0.887740239	569	12	0.943194554	1
82	2	3	2	1.186164412	490	12	0.943194554	1
82	5	3	2	0.915593162	584	12	0.888450288	1
79	2	3	2	1.034962831	491	12	0.888450288	1
72	5	3	2	1.158311489	492	12	0.888450288	1
381	6	1	6	0.008383675	668	7	0.943194554	0
353	6	1	6	0.274975936	611	7	0.943194554	0
383	2	1	6	0.230355664	870	11	1.804272709	0
391	2	1	6	0.139690311	610	11	0.943194554	0
335	2	1	6	0.024299631	776	11	0.943194554	0
397	2	1	6	0.158733862	817	7	0.943194554	0

Comparison with other classifier:

	Accuracy
Neural Model:	99%
Gradient Boosting Classifier:	88%
Random Forest Classifier:	98%

```
X_train.shape=(42976, 8) X_test.shape=(10744, 8)
Gradient Boosting classifier Accuracy: [0.88198064]
```

```
X_train.shape=(43218, 8) X_test.shape=(10805, 8)
Random Forest Classifier Accuracy: 0.989819527996298
```


THANK YOU!