MUSHROOM CLASSIFICATION

Math Concepts Machine Learning

Submitted to:

Mohammad Fadaee

PREPARED BY:
SHAURYA NEGI (101372208)
KRISH SHAH (101363412)
RESHMA MADHAVANKUTTY (101357044)
RISHAB MAHAJAN (101354622)
SAI GANESH CHALLA (101368077)

Contents

Introduction

Mushrooms being the most sustainably produced foods, not only have good taste but also hold a great nutritional value. They contain proteins, vitamins, minerals, and antioxidants. These can have various health benefits. Mushrooms aid in strengthening our immune system and help us to lose weight. They are a beguiling mixture of lucrative as well as speculative features.

But aside from the healthy mushrooms, there also exists poisonous and wild mushrooms whose consumption may result in severe illnesses in humans and can even cause death. It is not easy for a layman to differentiate wild mushrooms from healthy mushrooms.

In this project, we have used dataset which simulates various characteristics of mushrooms' physical appearances such as cap shapes, cap colours, cap surfaces, odor, gills spacing, gills attachment, gill colour and stalk shape are among many other features.

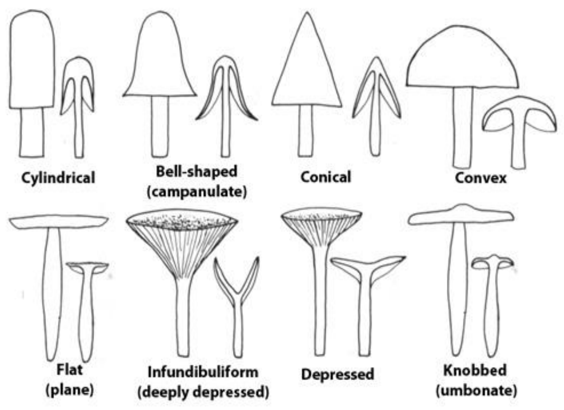


Fig (1) Cap Shapes

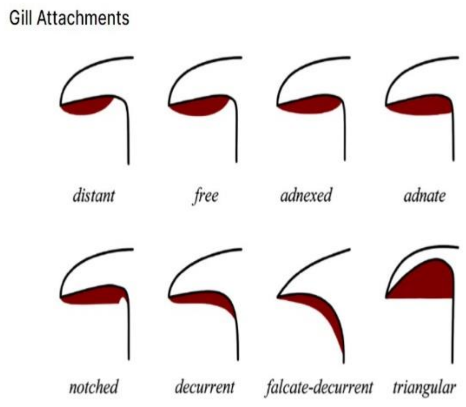


Fig (2) Gill Attachments

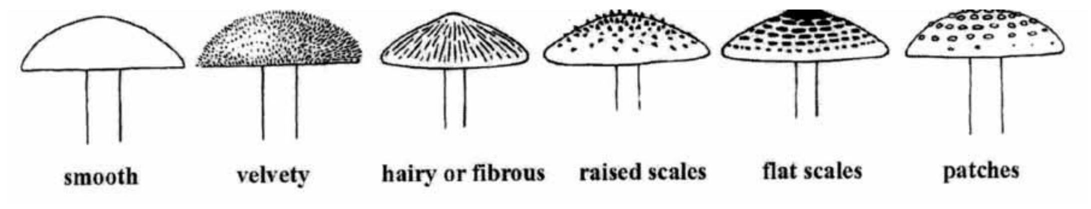


Fig (3) Cap Surface

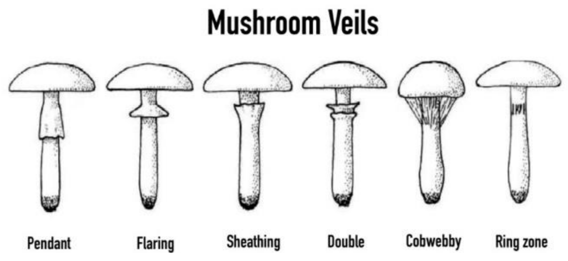
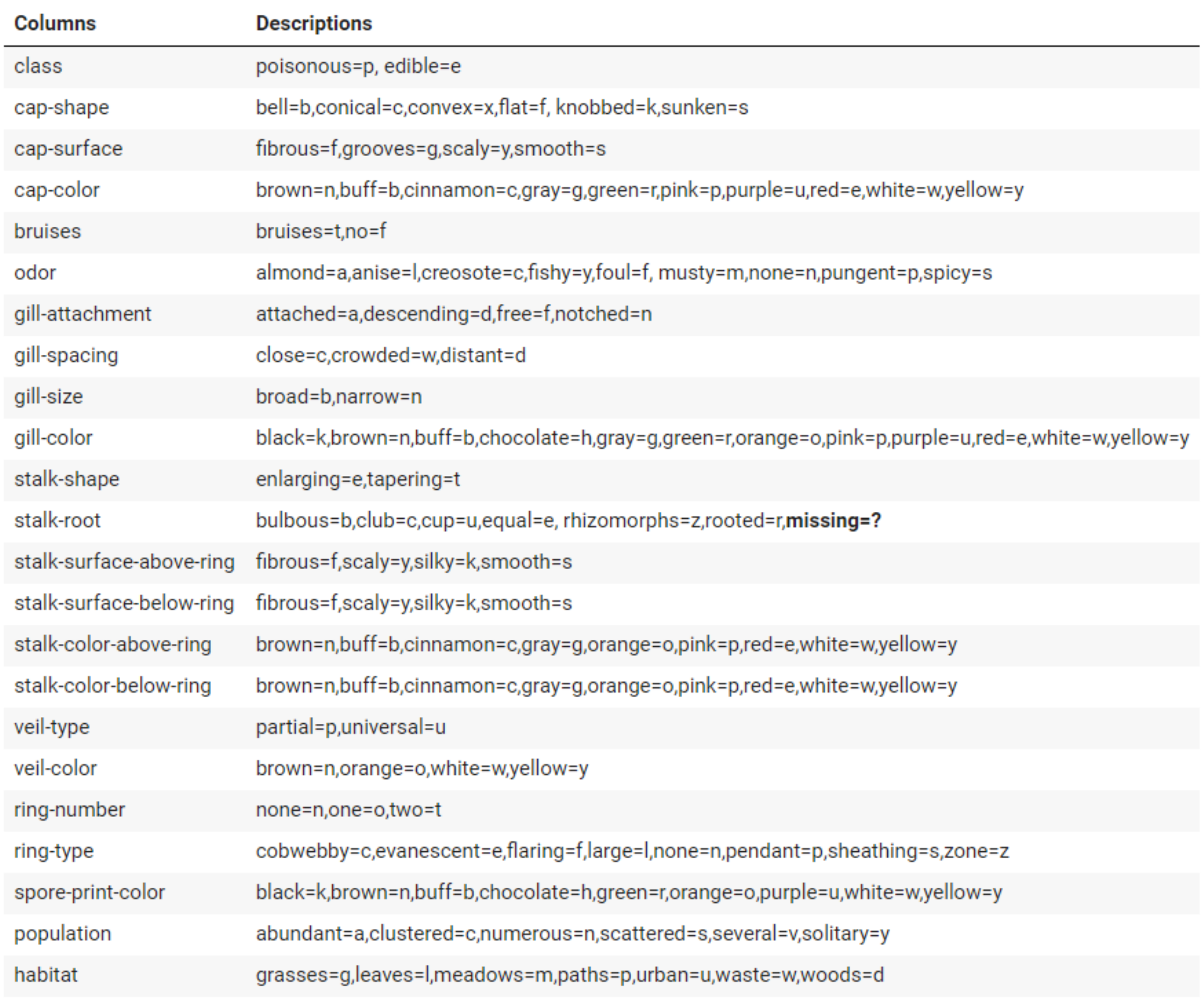


Fig (4) Mushroom Veils

In the following report we will discuss various steps we took to create a classification model to differentiate between edible and poisonous mushroom.

Dataset Description

Source: [UCI Machine Learning Repository: Mushroom Data Set]



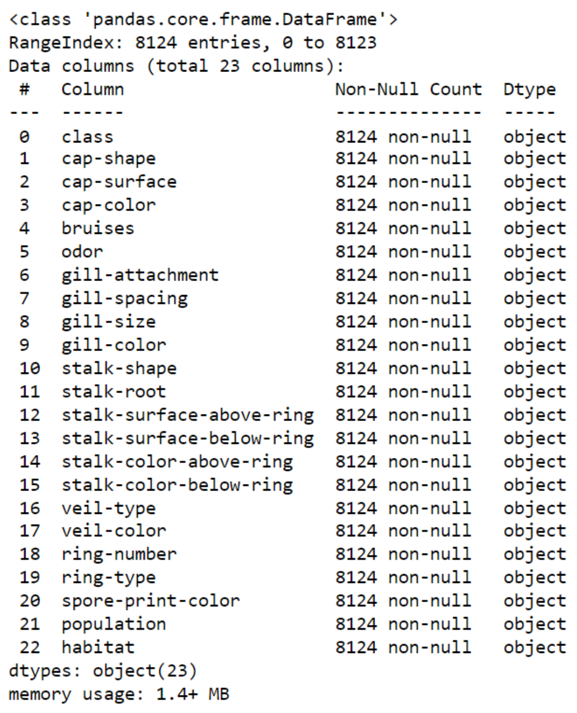
Columns	Descriptions
class	poisonous=p, edible=e
cap-shape	bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
cap-surface	fibrous=f,grooves=g,scaly=y,smooth=s
cap-color	brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y
bruises	bruises=t,no=f
odor	almond=a,anise=l,creosote=c,fishy=y,foul=f,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

Fig (4)

Key Observations:

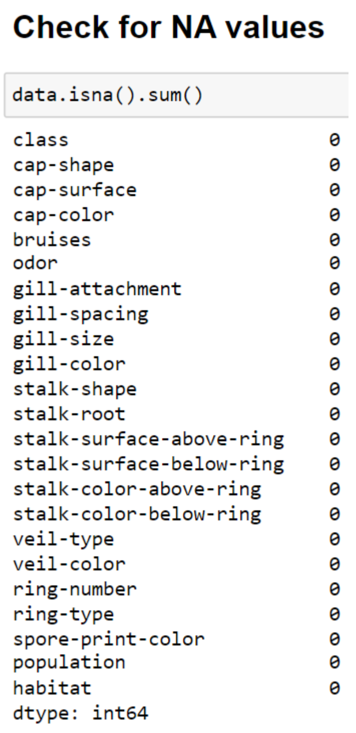
- There are 15 Nominal Categorical which describe the characteristics of mushrooms including the texture, colors, population and habitat.
- The mushroom dataset includes descriptions of hypothetical samples corresponding to 23 species.
- Each species is identified as *edible*, poisonous.
- Hence, the task given is a binary classification problem whereby, given the features of mushrooms, we are to classify the mushrooms into **p=Poisonous** or **e=edible**
- This dataset contains 22 attributes with 8124 instances of mushrooms. Figure below gives the attribute information of the dataset.
- Absence of null values in the dataset.

- As we can see from the plot, our data is balanced. Therefore, we do not need to think about oversampling/undersampling data or use SMOTE.
- There are no outliers in our data as all of our features are categories and we would encode them later.
- We do not have any duplicate rows/columns in our data.



```
<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
```

Fig (5)



Check for NA values

```
data.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
```

Fig (6)

Exploratory Data Analysis

Column Type	Column Names
Constant Value Column (1 Unique Value):	"veil-type"
Binary Columns (2 Unique Values):	"is-edible"(label), "bruises", "gill-attachment", "gill-spacing", "gill-size", "stalk-shape"
Nominal Categorical Columns(>2 Unique Values):	"cap-shape", "cap-surface", "cap-color", "odor", "gill-color", "stalk-root", "stalk-surface-above-ring", "stalk-surface-below-ring", "stalk-color-above-ring", "stalk-color-below-ring", "veil-color", "ring-type", "spore-print-color", "population", "habitat", "ring-number"
Discrete Numerical Columns(Countable Values):	None

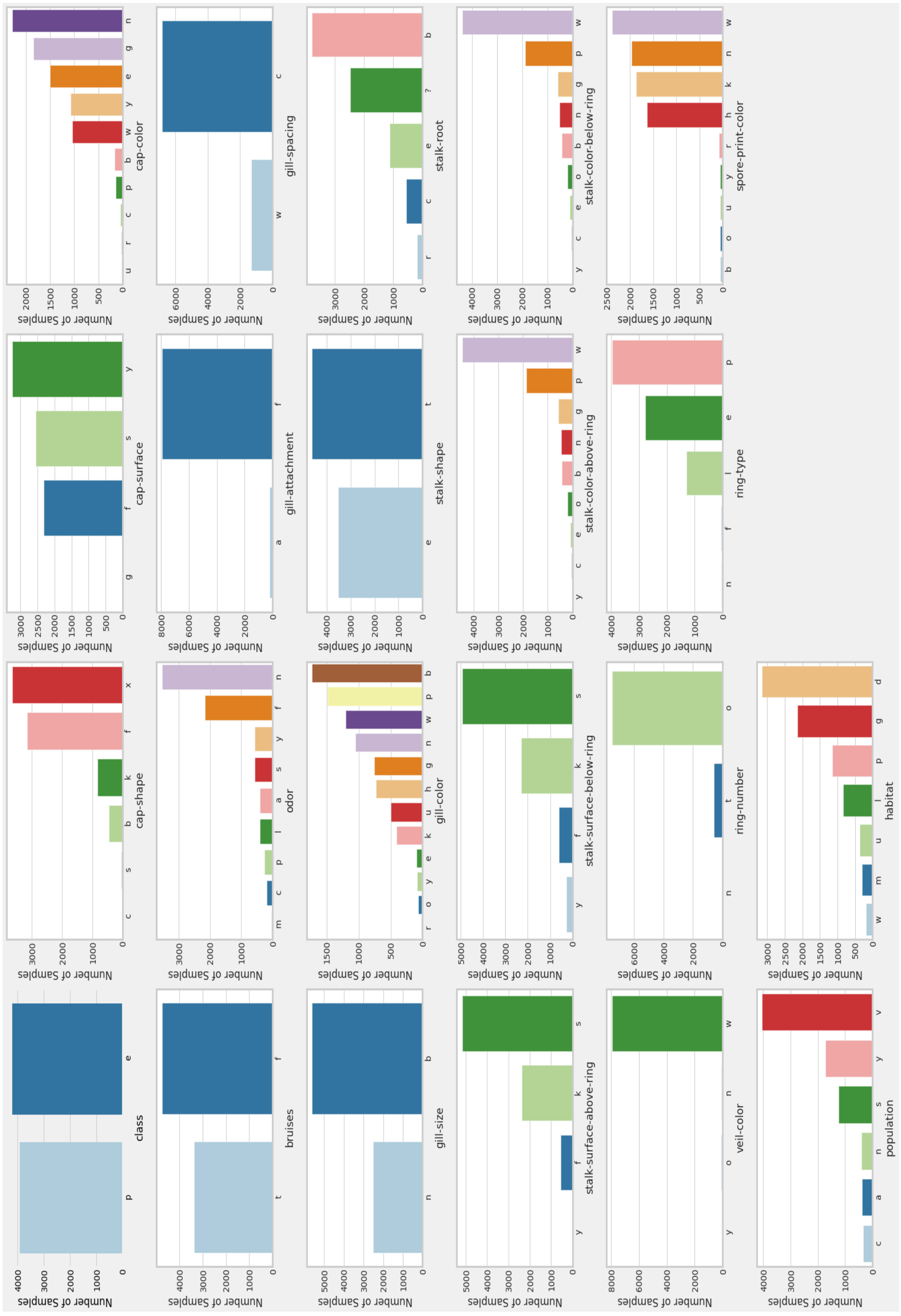
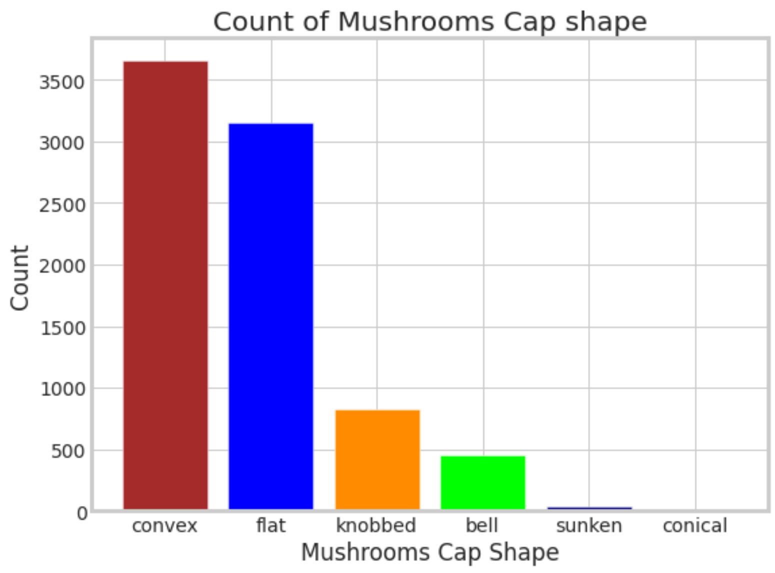
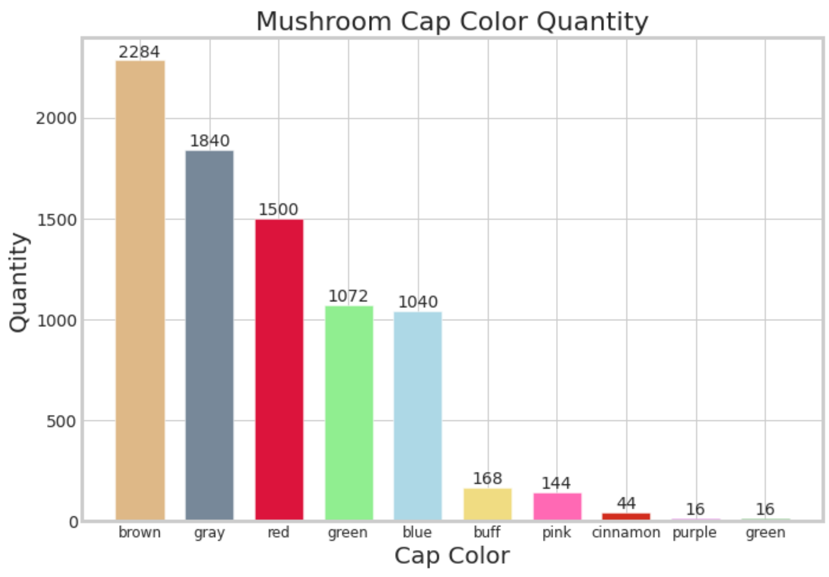

Number of Samples
class
bruises
gill-size
stalk-surface-above-ring
veil-color
population
cap-shape
odor
gill-color
stalk-surface-below-ring
ring-number
habitat
cap-surface
gill-attachment
stalk-shape
stalk-color-above-ring
ring-type
cap-color
gill-spacing
stalk-root
stalk-color-below-ring
spore-print-color

Fig (7)



Count of Mushrooms Cap shape
Count
3500
3000
2500
2000
1500
1000
500
0
convex
flat
knobbed
bell
sunken
conical
Mushrooms Cap Shape

Fig (8)



Mushroom Cap Color Quantity
Quantity
2000
1500
1000
500
0
2284
1840
1500
1072
1040
168
144
44
16
16
brown
gray
red
green
blue
buff
pink
cinnamon
purple
green
Cap Color

Fig (9)

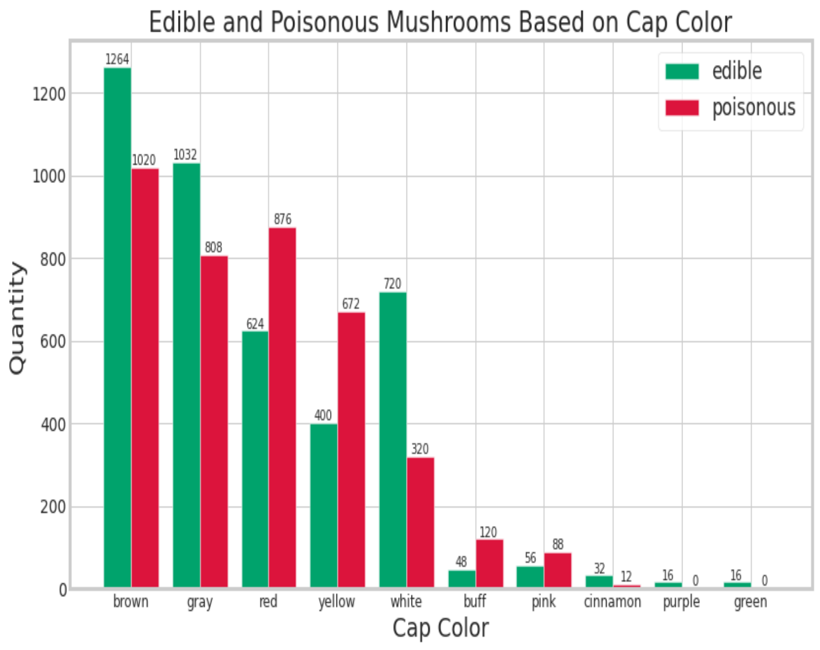

Edible and Poisonous Mushrooms Based on Cap Color
edible
poisonous
Quantity
Cap Color
brown
gray
red
yellow
white
buff
pink
cinnamon
purple
green
1264
1020
1032
808
624
876
400
672
720
320
48
120
56
88
32
12
16
0
16
0

Fig (10)

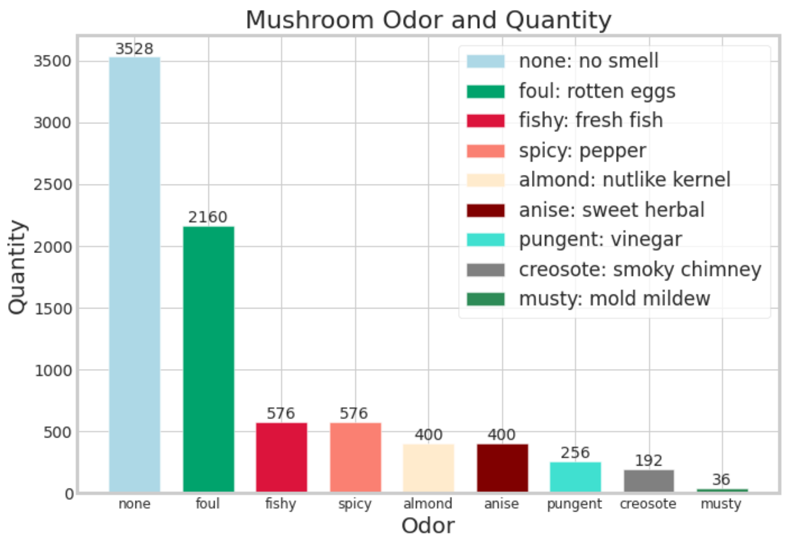

Mushroom Odor and Quantity
none: no smell
foul: rotten eggs
fishy: fresh fish
spicy: pepper
almond: nutlike kernel
anise: sweet herbal
pungent: vinegar
creosote: smoky chimney
musty: mold mildew
Quantity
Odor
none
foul
fishy
spicy
almond
anise
pungent
creosote
musty
3528
2160
576
576
400
400
256
192
36

Fig (11)

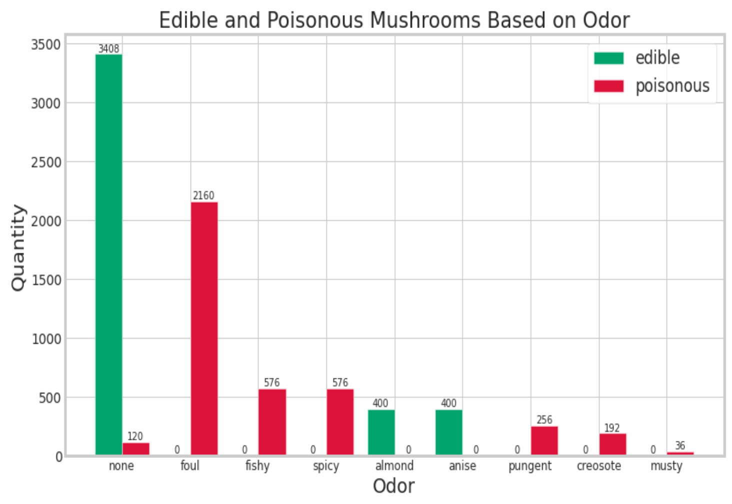


Fig (12)

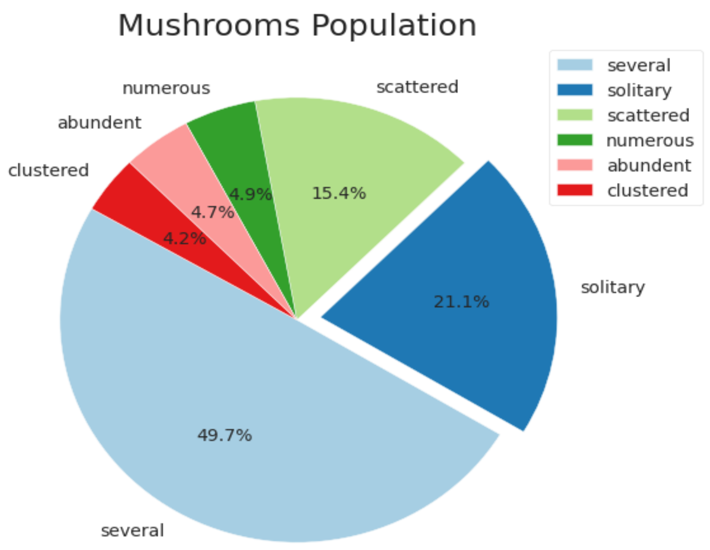


Fig (13)

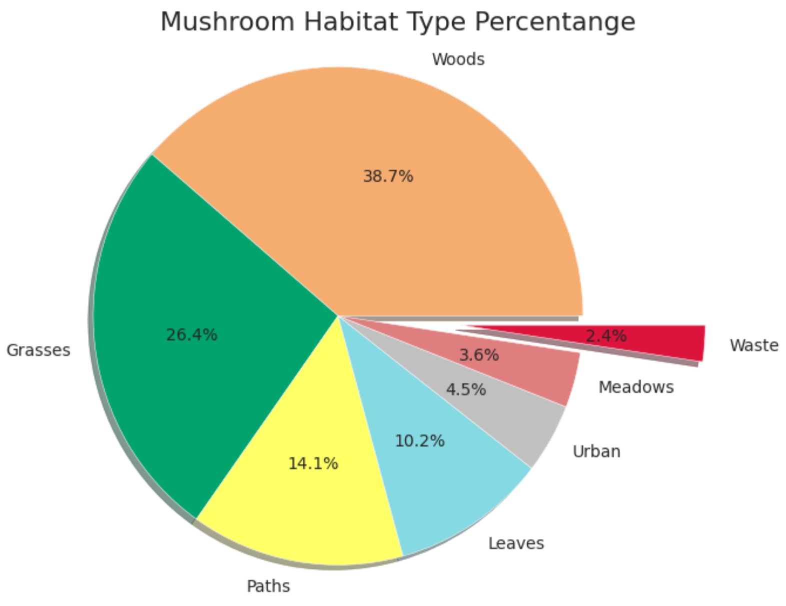
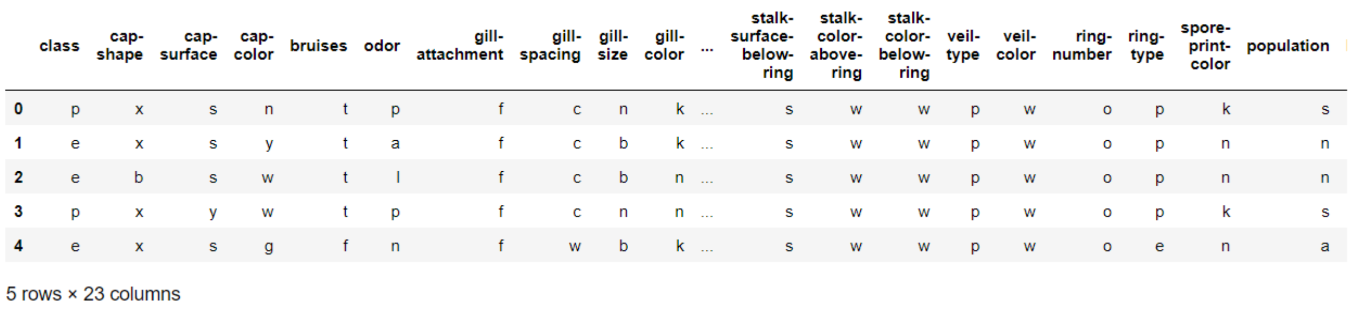


Fig (14)

Data Preprocessing

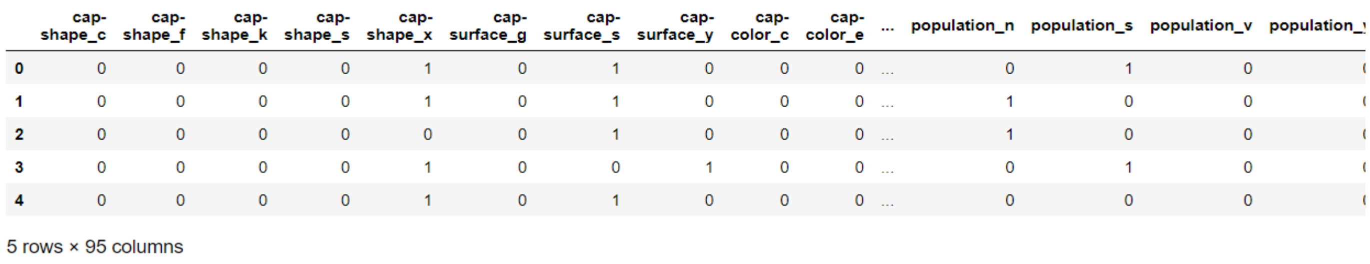
- As discussed earlier, since all our columns are encoded in string and are categorical, we Label encoded y - dependent variable (Target)
- We have used get dummies to preserve the names of the columns to do further data exploration an d feature selection. As a result, we get 95 features to reduce those n features, We are going to use PCA for reducing dimensionality.
- After reducing some dimensionalities, We are going to fit those new features to the models and see which one gives best results.



	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
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

Fig (15) Original Dataset



	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	...	population_n	population_s	population_v	population_
0	0	0	0	0	1	0	1	0	0	0	...	0	1	0	(
1	0	0	0	0	1	0	1	0	0	0	...	1	0	0	(
2	0	0	0	0	0	0	1	0	0	0	...	1	0	0	(
3	0	0	0	0	1	0	0	1	0	0	...	0	1	0	(
4	0	0	0	0	1	0	1	0	0	0	...	0	0	0	(

5 rows × 95 columns

Fig (16) Label Encoded Data with Dummy Variables

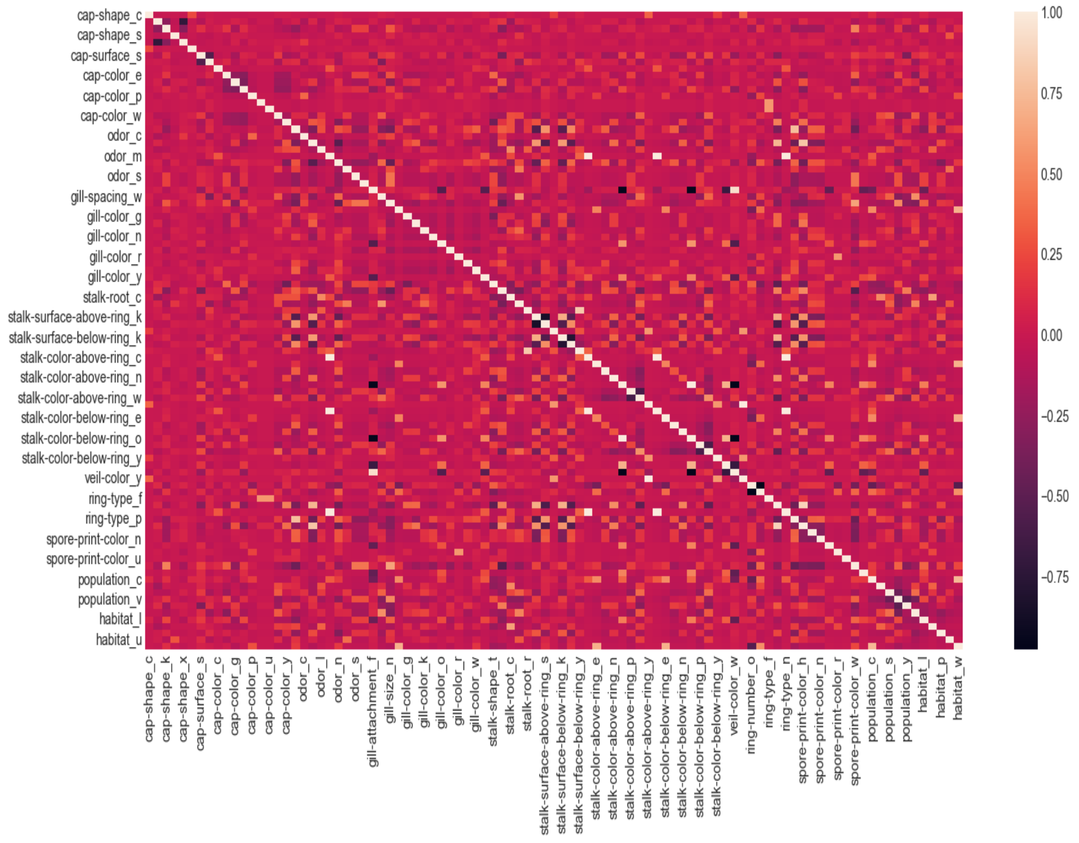


Fig (17) Heat Map

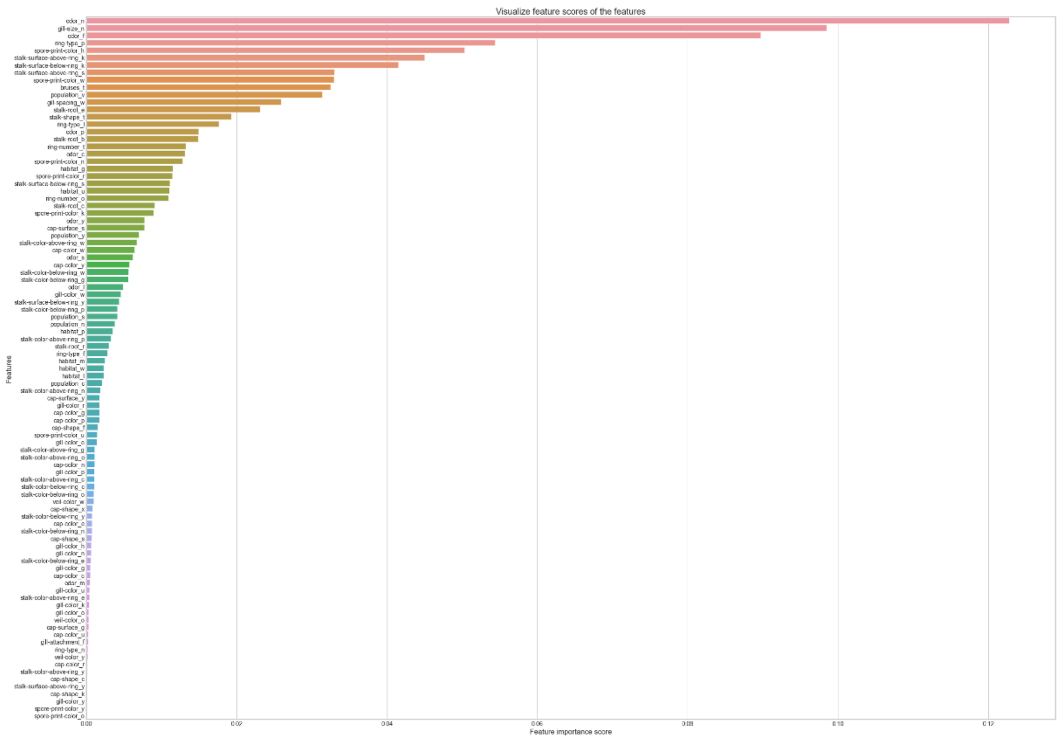


Fig (18) Feature Scores using Random Forest

Data Splitting

Data splitting is a process used to separate a given dataset into at least two subsets called ‘training’ and ‘test’. This step is usually implemented after data preprocessing. Using train_test_split() from the data science library scikit-learn, the data is split into subsets i.e., training and test which contains 70% and 30% data respectively. This minimizes the potential for bias in the evaluation and validation process. Stratify preserves the same proportions of examples in each class as observed in the original dataset.

Feature Scaling

Feature Scaling is done to standardize the independent features present in the data in a fixed range. We have used StandardScaler() to perform feature scaling. It performs the task of Standardization. StandardScaler() will normalize the features i.e., each column of X, individually, so that each feature/column will have μ = 0 and σ = 1.

$$x_{new} = \frac{x-\mu}{\sigma}$$

StandardScaler() will normalize the features i.e. each column of X, individually, so that each feature/column will have μ = 0 and σ = 1. The Standard Scaler assumes data is normally distributed within each feature and scales them such that the distribution centered around 0, with a standard deviation of 1

Dimensionality Reduction

PCA is a technique from linear algebra that can be used to automatically perform dimensionality reduction. Reducing the number of features in a dataset can reduce the risk of overfitting and also improves the accuracy of the model.

The Principal Component Analysis (PCA) algorithm is used to select the best features from the mushroom dataset. PCA is a technique from linear algebra that can be used to automatically perform dimensionality reduction. We have used PCA with n_components = 2 for reducing the dimensions of the dataset.

It is necessary to normalize data before performing PCA. The PCA calculates a new projection of the data set. In normalization of data, all variables have the same standard deviation, thus all variables have the same weight and PCA calculates relevant axis.

Key Observations

- For a sparse dataset as ours, reducing the number of features in the dataset make our model much simpler, can reduce the risk of overfitting and also improves the accuracy of the model.
- Using n_components = 2 allows us to visualize our model performance and plot decision boundaries which helps in increasing interpretability of the model.

Classification Modelling

After the feature extraction and selection, the supervised machine learning methods are applied to the data obtained. The machine learning methods to be applied:

- Logistic Regression (LR)
- Decision Tree (DT)
- Support Vector Machines (SVM)
- Random Forest (RF)

Evaluation of the prediction results using various evaluation metrics like AUC-ROC, confusion matrix, classification accuracy, precision, recall, f1-score. is done.

Confusion Matrix -

It is a matrix of size 2×2 for binary classification with actual values on one axis and predicted on another. It describes the complete performance of the model.

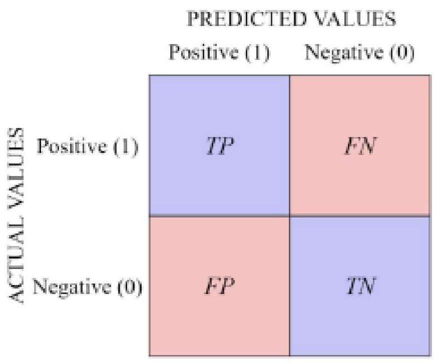


Fig (18) Confusion Matrix

Where *TP* = True Positives,
TN = True Negatives,
FP = False Positives,
FN = False Negatives.

Classification Accuracy -

It is the ratio of the number of correct predictions to the total number of input samples. It is given as:

$$Accuracy = \frac{Number\ of\ Correct\ Predicitons}{Total\ Number\ of\ Predictions}$$

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision -

Precision is the number of correct positive results divided by the number of positive results predicted by the classifier. Precision is defined as follows:

$$Precision = \frac{TP}{TP + FP}$$

Recall / Sensitivity / True Positive Rate (TPR) -
It is the number of correct positive results divided by the number of all relevant samples. Mathematically, recall is defined as follows:

$$Recall = \frac{TP}{TP + FN}$$

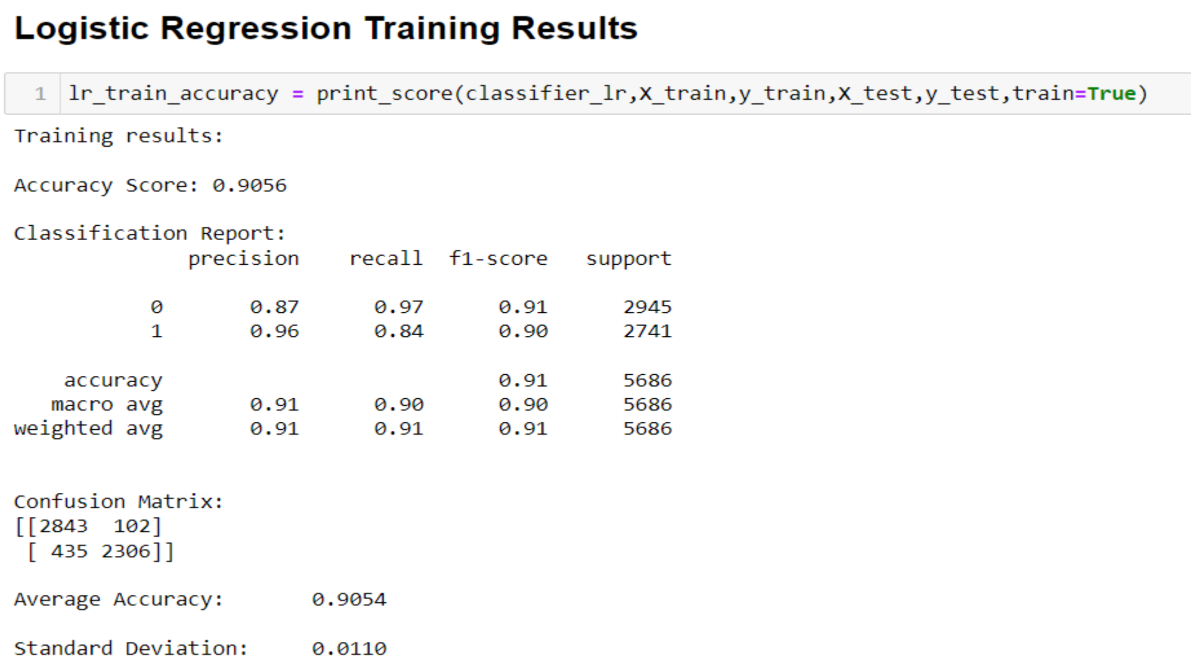
F1 Score -
It is used to measure a test's accuracy. F1 Score is the Harmonic Mean between precision and recall. The range for the F1 Score is [0, 1]. It tells you how precise your classifier is as well as how robust it is. Mathematically, the F1 Score is defined as follows:

$$F1 - Score = 2 * \frac{1}{\left(\frac{1}{Precision}\right) + \left(\frac{1}{Recall}\right)}$$

F1 Score tries to find the balance between precision and recall.

MODELLING RESULTS:

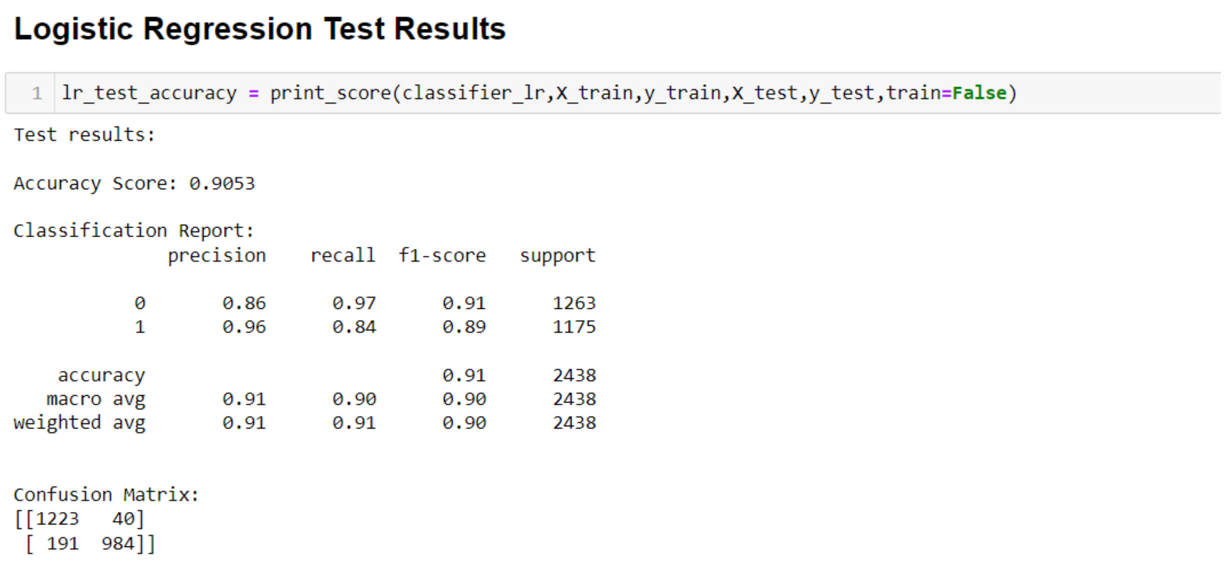
1. **Logistic Regression**



Logistic Regression Training Results

```
lr_train_accuracy = print_score(classifier_lr,X_train,y_train,X_test,y_test,train=True)
```

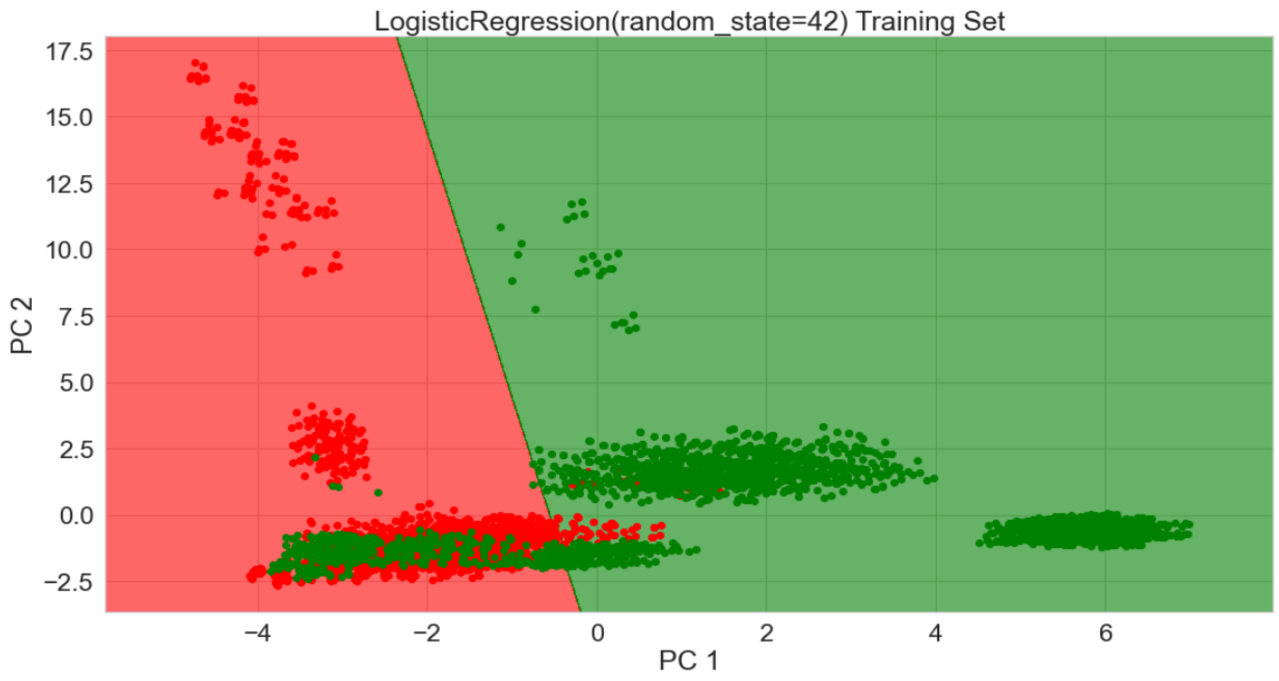
```
Training results:

Accuracy Score: 0.9056

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.97      0.91      2945
           1       0.96      0.84      0.90      2741

    accuracy                           0.91      5686
   macro avg       0.91      0.90      0.90      5686
weighted avg       0.91      0.91      0.91      5686


Confusion Matrix:
[[2843  102]
 [ 435 2306]]

Average Accuracy:        0.9054

Standard Deviation:      0.0110
```

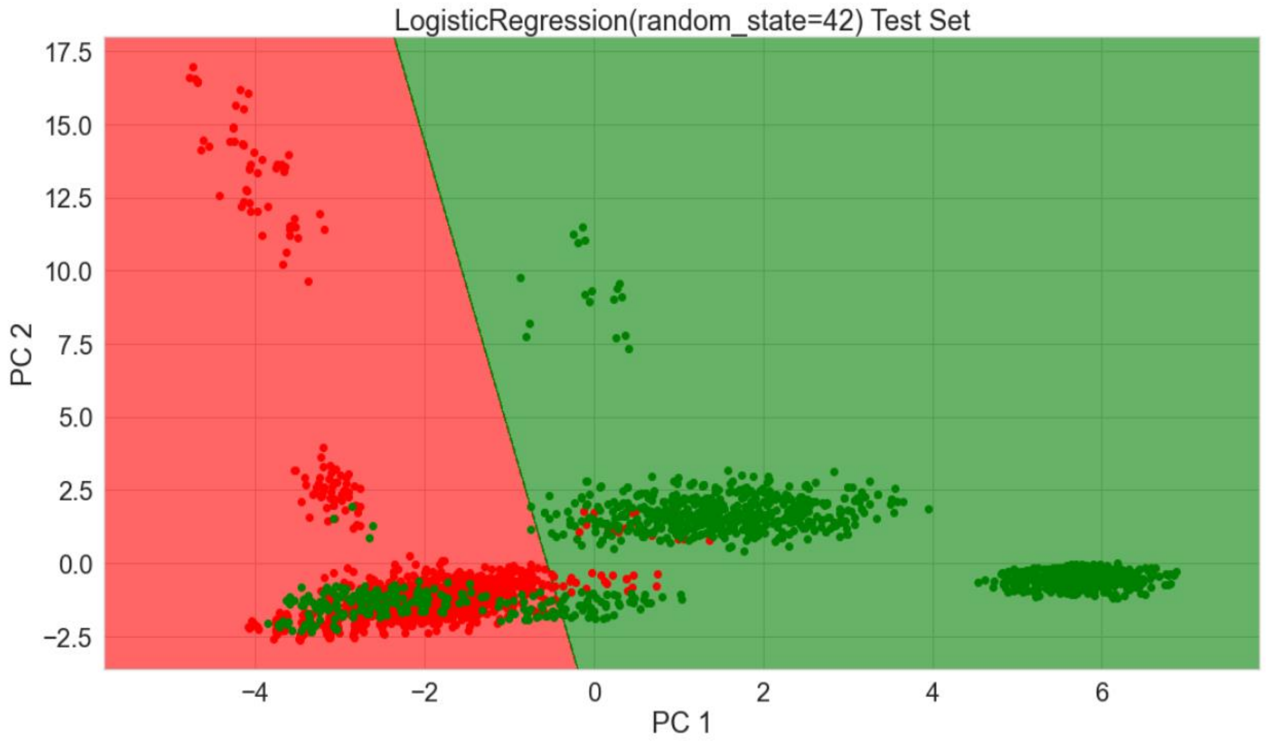


Logistic Regression Test Results

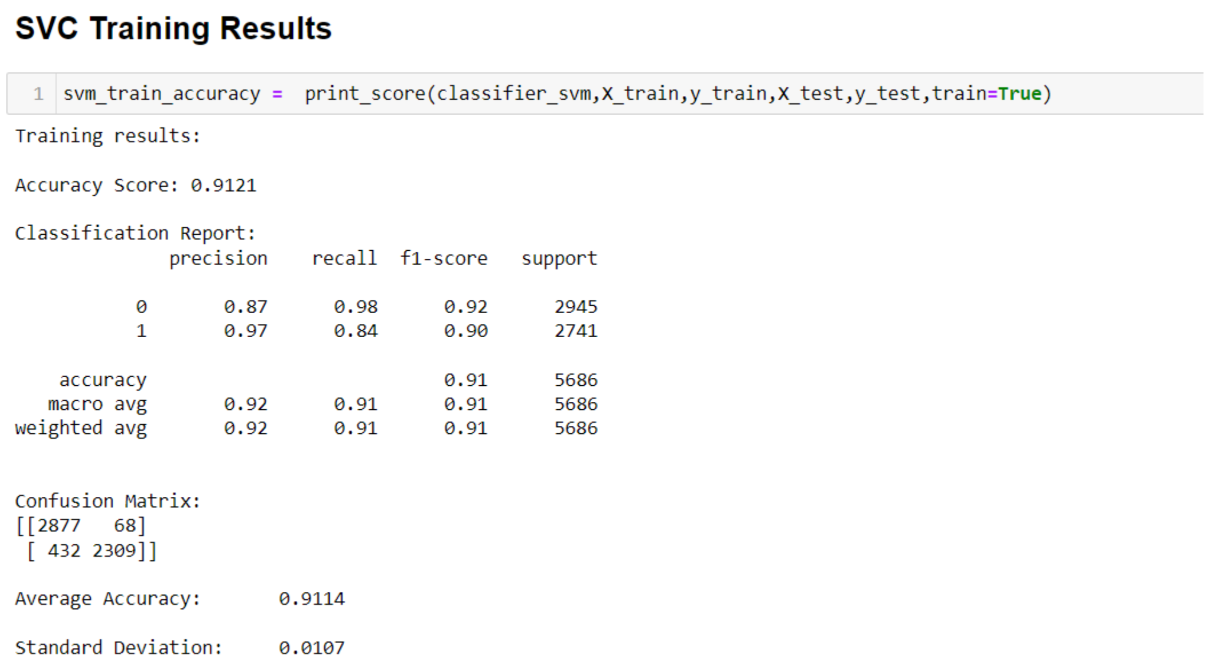
```
lr_test_accuracy = print_score(classifier_lr,X_train,y_train,X_test,y_test,train=False)
```

```
Test results:

Accuracy Score: 0.9053

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.97      0.91      1263
           1       0.96      0.84      0.89      1175

    accuracy                           0.91      2438
   macro avg       0.91      0.90      0.90      2438
weighted avg       0.91      0.91      0.90      2438


Confusion Matrix:
[[1223   40]
 [ 191  984]]
```


LogisticRegression(random_state=42) Training Set
PC 2
PC 1
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−4
−2
0
2
4
6


LogisticRegression(random_state=42) Test Set
PC 2
PC 1
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−4
−2
0
2
4
6

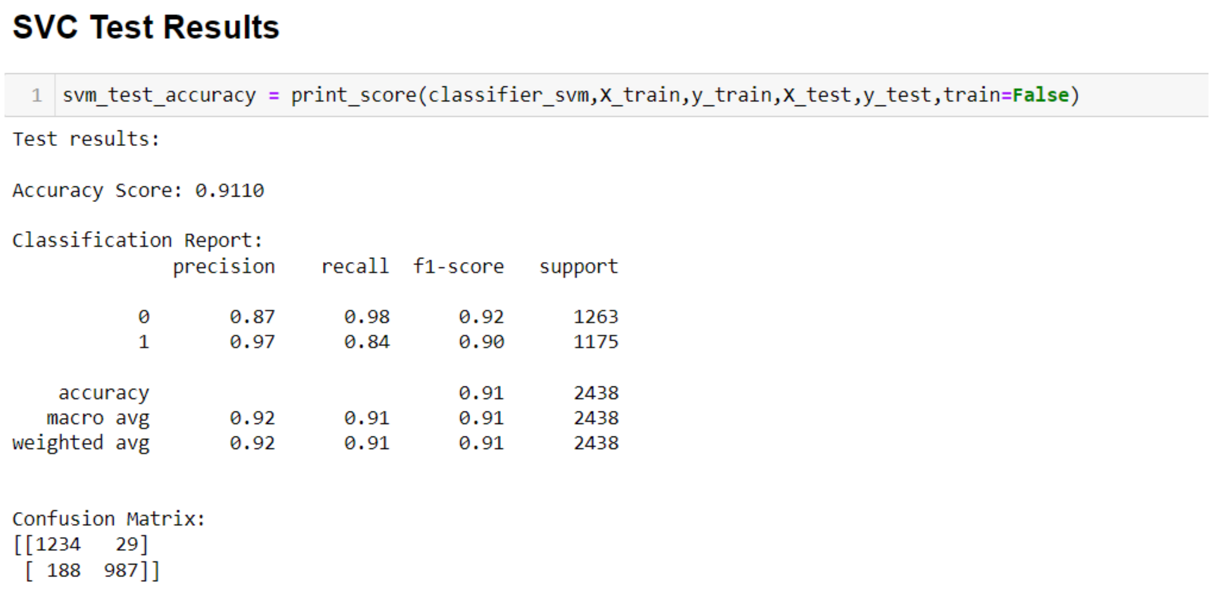
2. Support Vector Classifiers:



SVC Training Results

```
svm_train_accuracy =  print_score(classifier_svm,X_train,y_train,X_test,y_test,train=True)
```

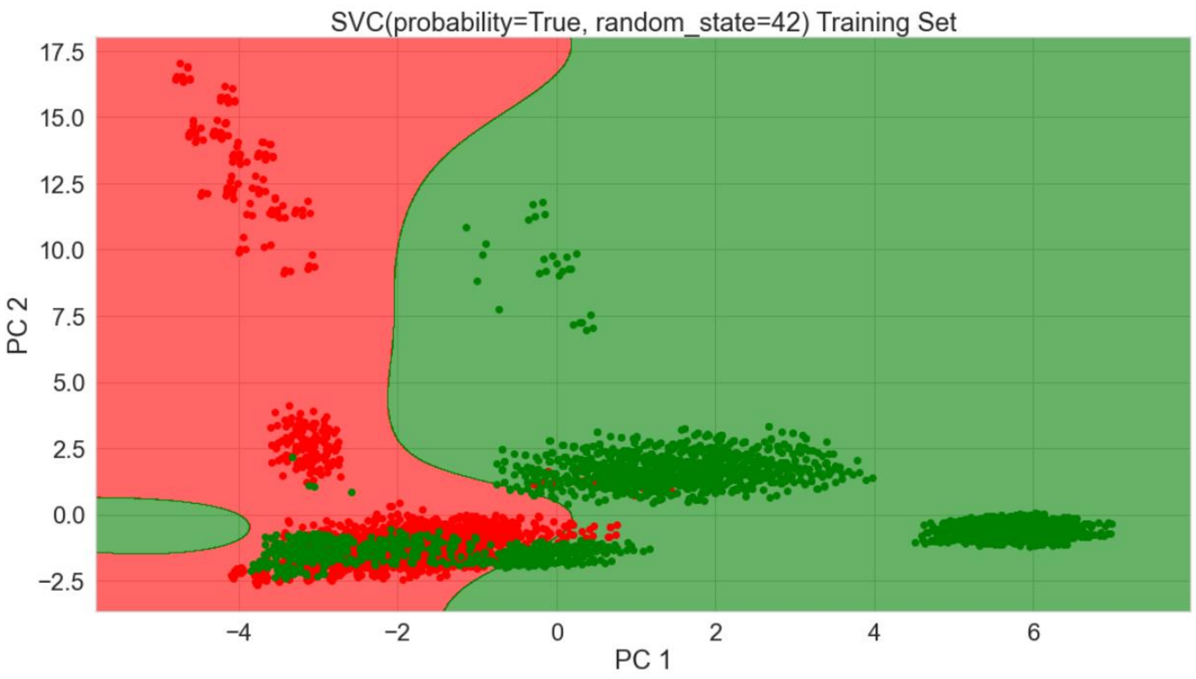
```
Training results:

Accuracy Score: 0.9121

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.98      0.92      2945
           1       0.97      0.84      0.90      2741

    accuracy                           0.91      5686
   macro avg       0.92      0.91      0.91      5686
weighted avg       0.92      0.91      0.91      5686


Confusion Matrix:
[[2877   68]
 [ 432 2309]]

Average Accuracy:        0.9114

Standard Deviation:      0.0107
```

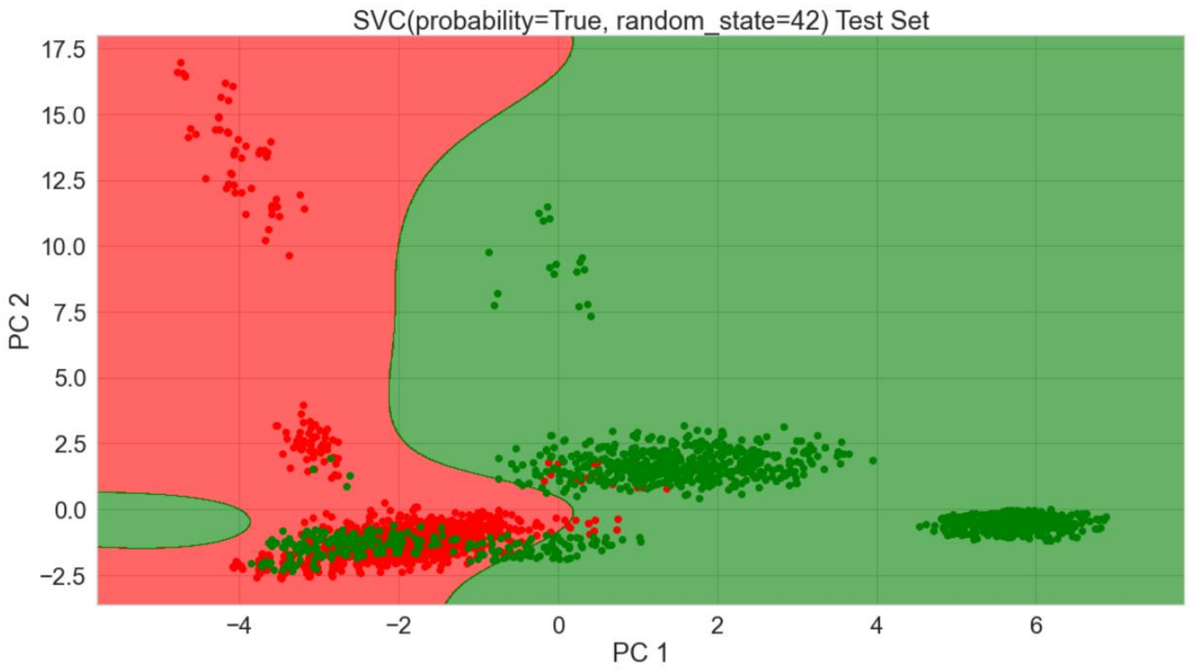


SVC Test Results

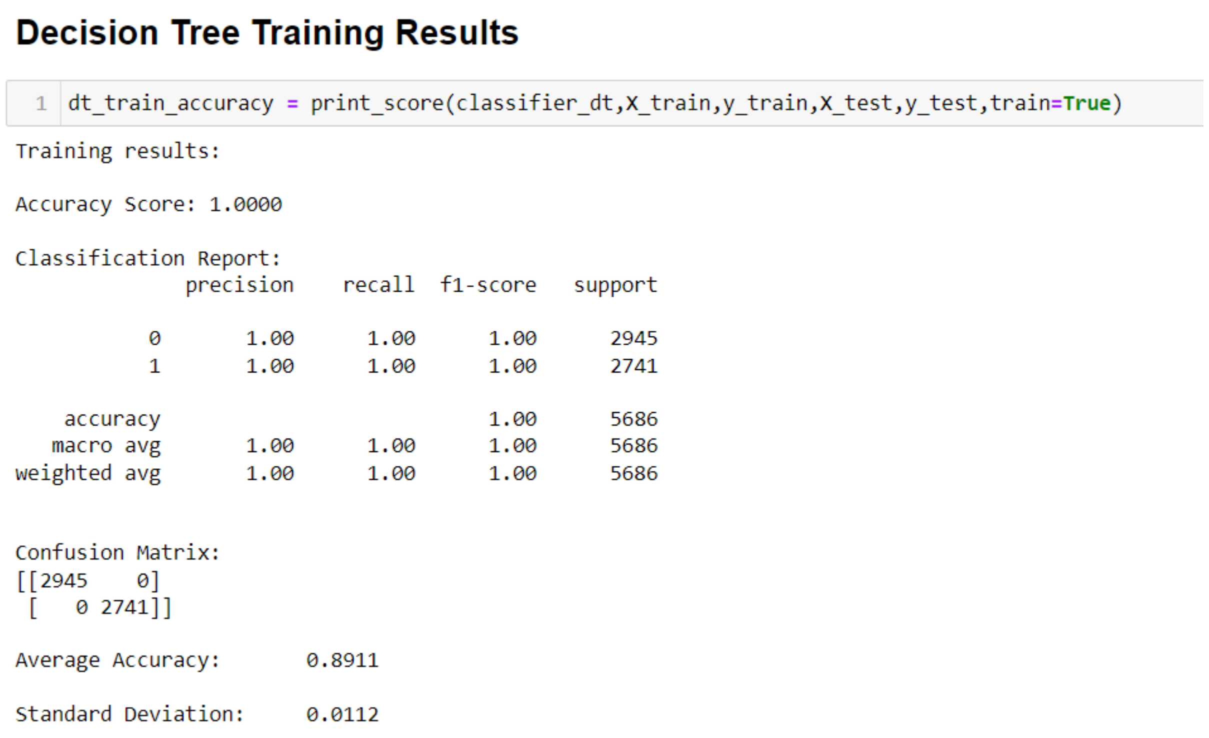
```
svm_test_accuracy = print_score(classifier_svm,X_train,y_train,X_test,y_test,train=False)
```

```
Test results:

Accuracy Score: 0.9110

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.98      0.92      1263
           1       0.97      0.84      0.90      1175

    accuracy                           0.91      2438
   macro avg       0.92      0.91      0.91      2438
weighted avg       0.92      0.91      0.91      2438


Confusion Matrix:
[[1234   29]
 [ 188  987]]
```


SVC(probability=True, random_state=42) Training Set
PC 2
PC 1
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−4
−2
0
2
4
6


SVC(probability=True, random_state=42) Test Set
PC 2
PC 1
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−4
−2
0
2
4
6

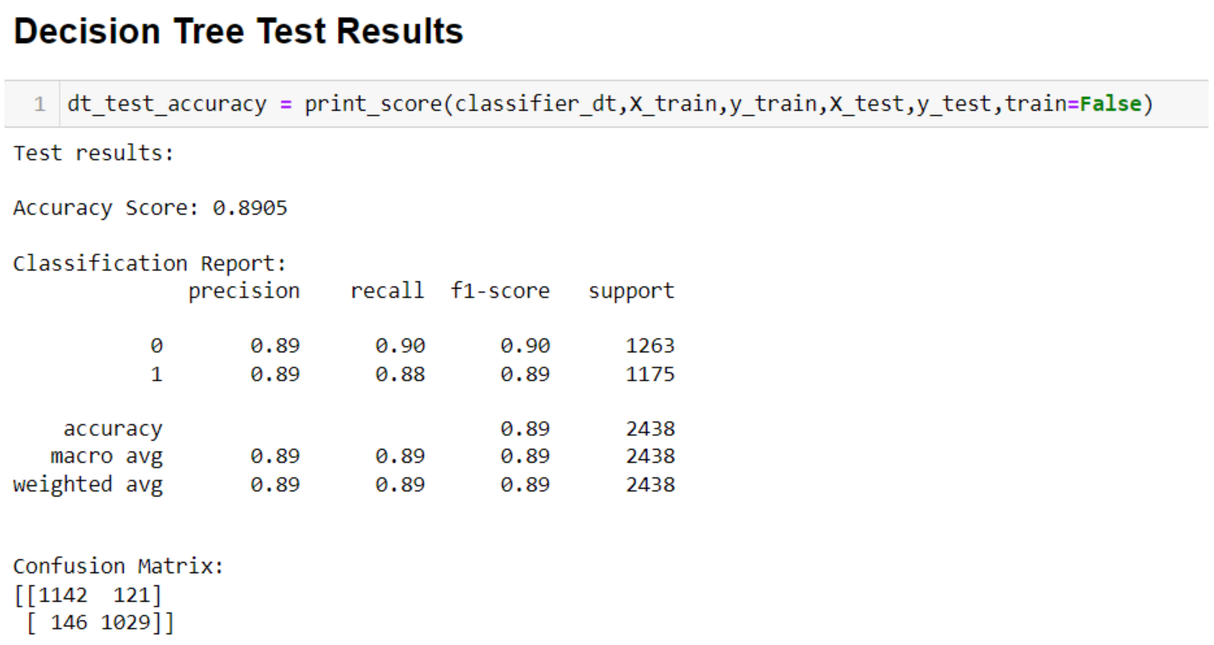
3. Decision Trees:



Decision Tree Training Results

```
dt_train_accuracy = print_score(classifier_dt,X_train,y_train,X_test,y_test,train=True)
```

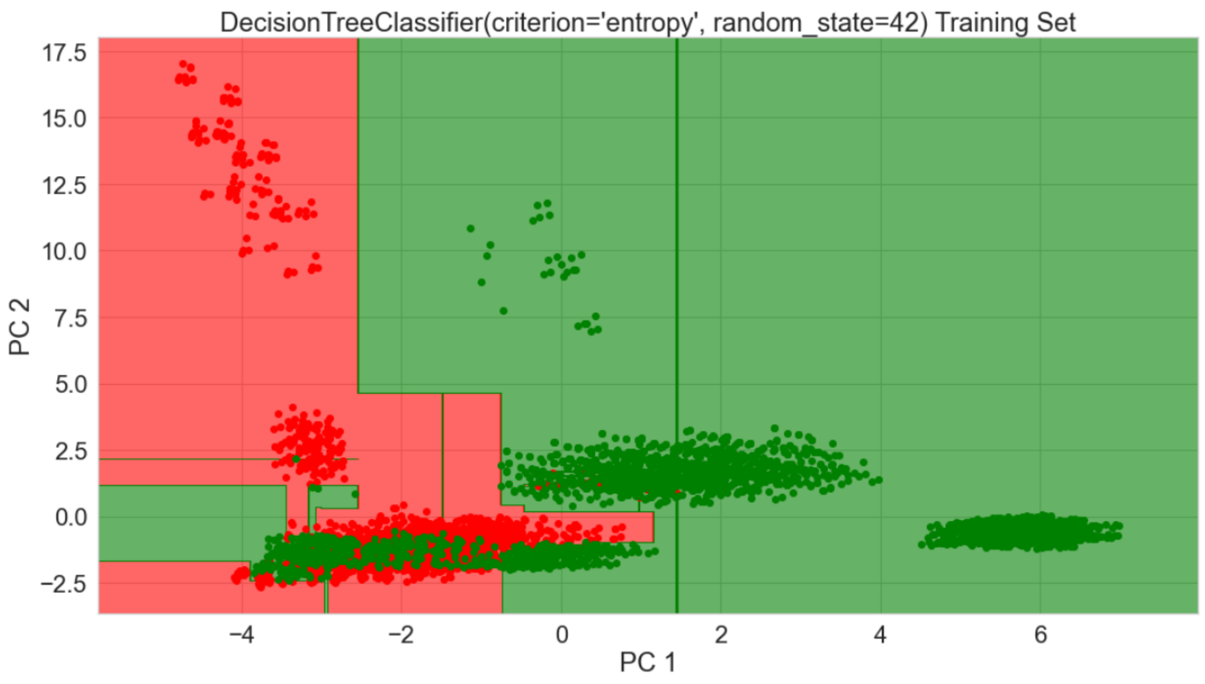
```
Training results:

Accuracy Score: 1.0000

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2945
           1       1.00      1.00      1.00      2741

    accuracy                           1.00      5686
   macro avg       1.00      1.00      1.00      5686
weighted avg       1.00      1.00      1.00      5686


Confusion Matrix:
[[2945    0]
 [   0 2741]]

Average Accuracy:       0.8911

Standard Deviation:     0.0112
```

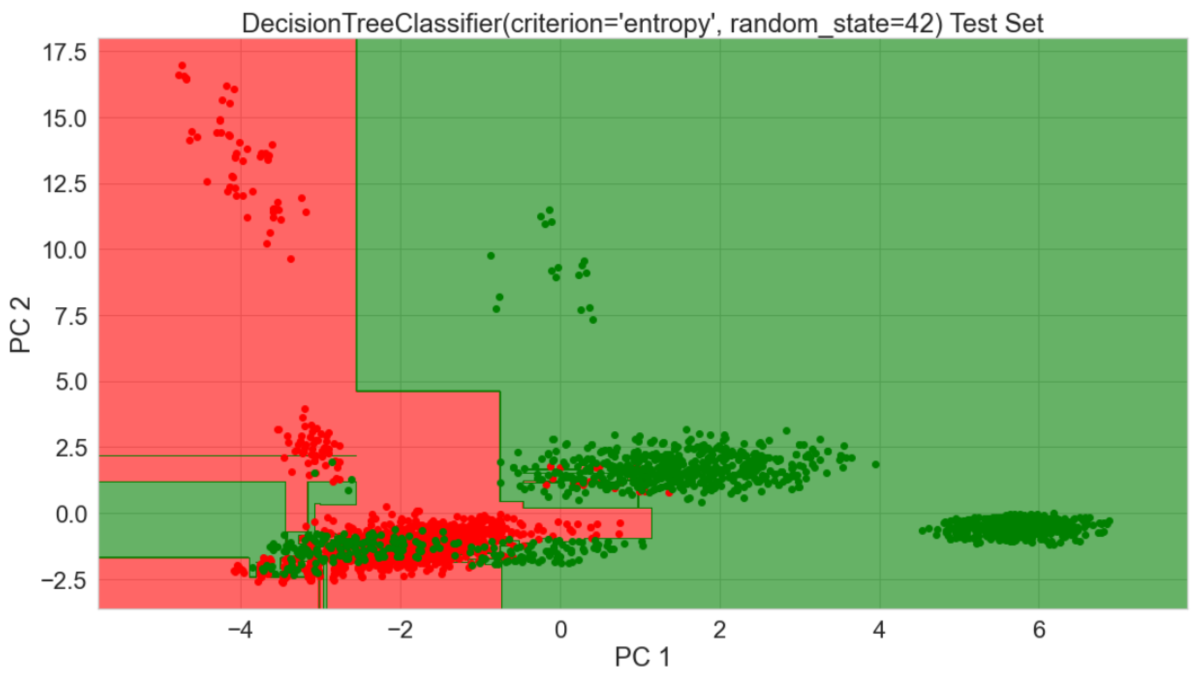


Decision Tree Test Results

```
dt_test_accuracy = print_score(classifier_dt,X_train,y_train,X_test,y_test,train=False)
```

```
Test results:

Accuracy Score: 0.8905

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.90      0.90      1263
           1       0.89      0.88      0.89      1175

    accuracy                           0.89      2438
   macro avg       0.89      0.89      0.89      2438
weighted avg       0.89      0.89      0.89      2438


Confusion Matrix:
[[1142  121]
 [ 146 1029]]
```



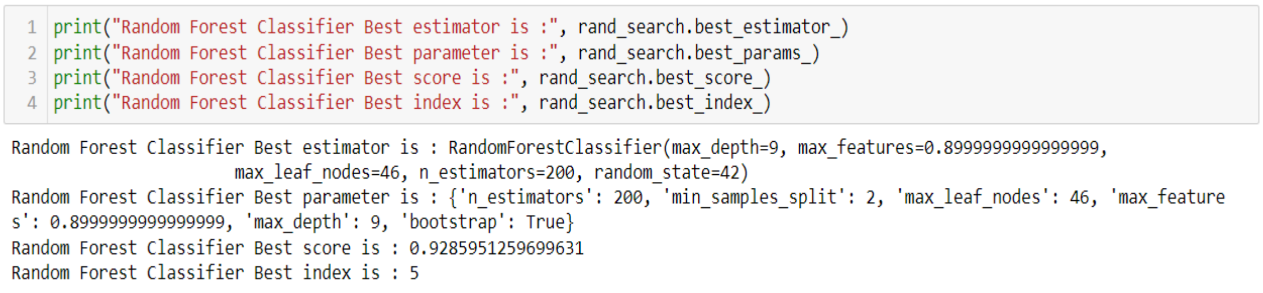


This is a clear case of overfitting as the decision boundary is very refined.

What is the solution to overfitting in a Decision Tree?

A hyperparameter tuned Random Forest should solve this issue for us.

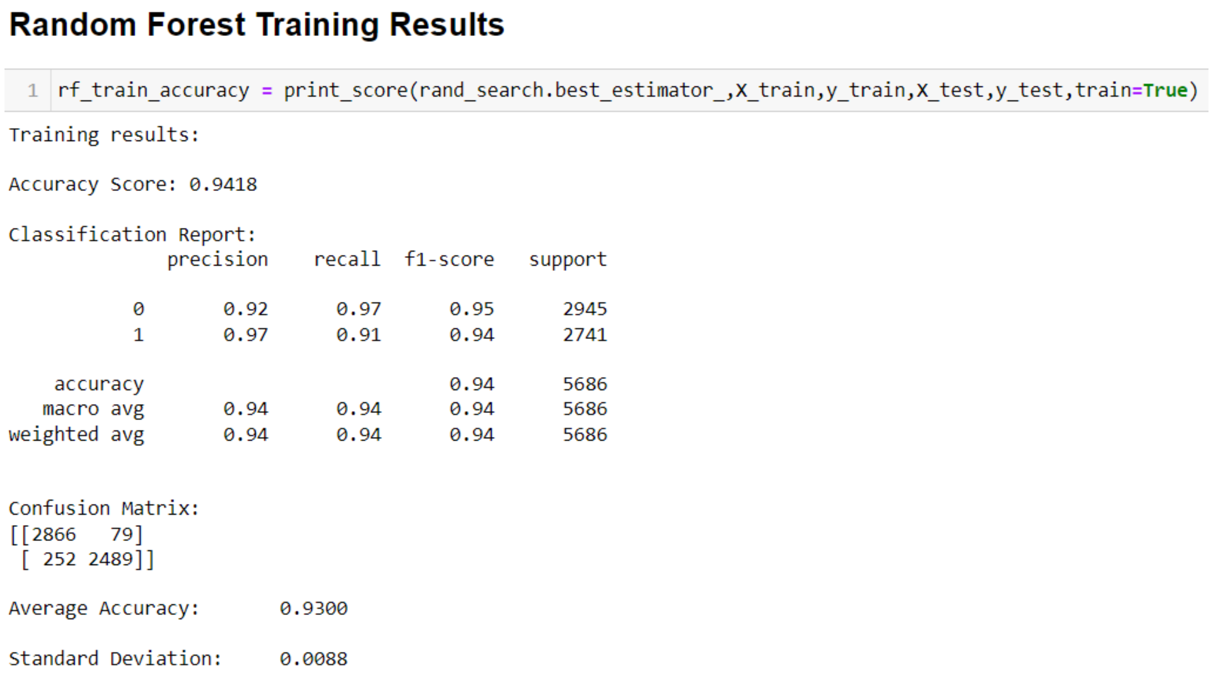
4. Random Forests (Hyperparameter Tuned)



```python
print("Random Forest Classifier Best estimator is :", rand_search.best_estimator_)
print("Random Forest Classifier Best parameter is :", rand_search.best_params_)
print("Random Forest Classifier Best score is :", rand_search.best_score_)
print("Random Forest Classifier Best index is :", rand_search.best_index_)
```

```
Random Forest Classifier Best estimator is : RandomForestClassifier(max_depth=9, max_features=0.8999999999999999,
                       max_leaf_nodes=46, n_estimators=200, random_state=42)
Random Forest Classifier Best parameter is : {'n_estimators': 200, 'min_samples_split': 2, 'max_leaf_nodes': 46, 'max_feature
s': 0.8999999999999999, 'max_depth': 9, 'bootstrap': True}
Random Forest Classifier Best score is : 0.9285951259699631
Random Forest Classifier Best index is : 5
```

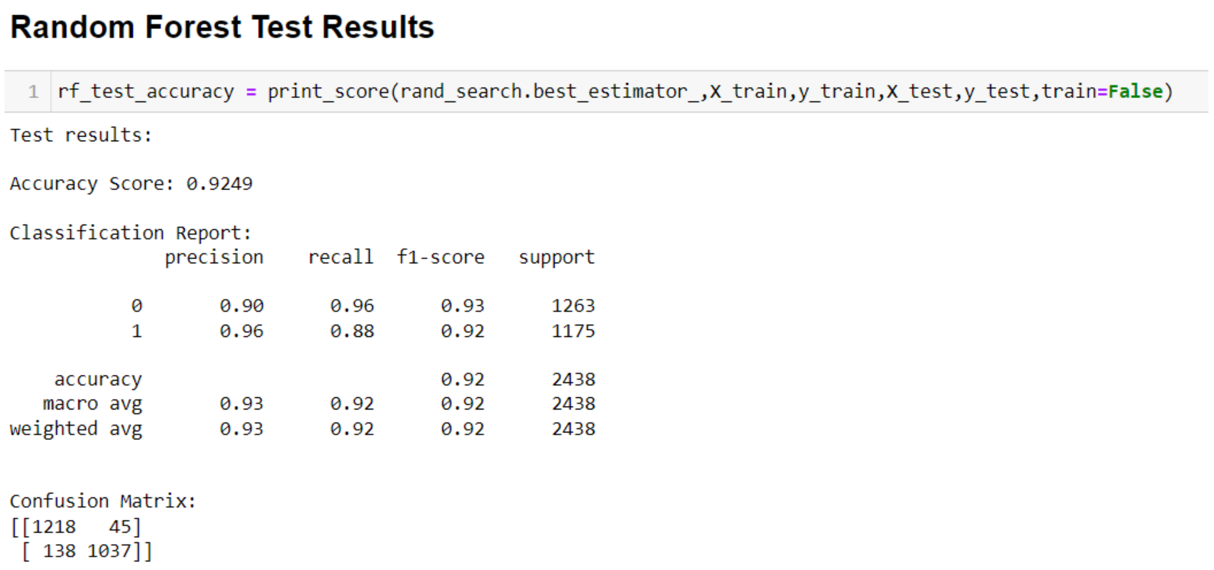
Fig (19) Best Estimators of the Tuned Random Forest



Random Forest Training Results

```python
rf_train_accuracy = print_score(rand_search.best_estimator_,X_train,y_train,X_test,y_test,train=True)
```

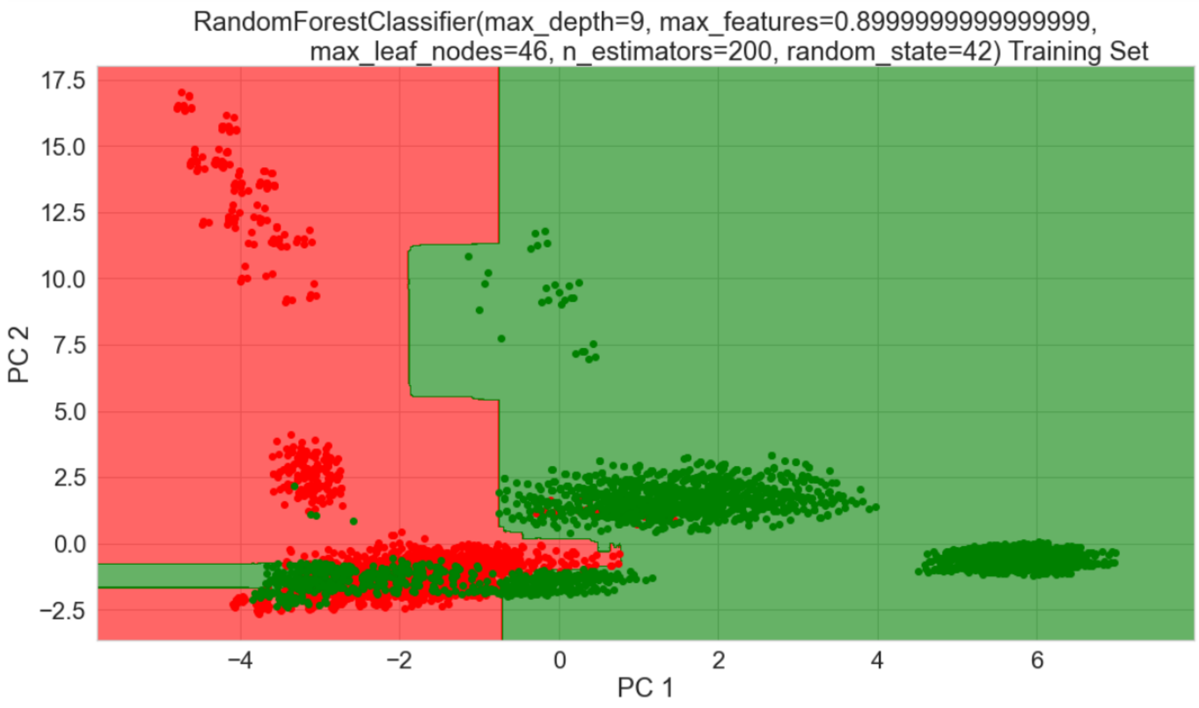
```
Training results:

Accuracy Score: 0.9418

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.97      0.95      2945
           1       0.97      0.91      0.94      2741

    accuracy                           0.94      5686
   macro avg       0.94      0.94      0.94      5686
weighted avg       0.94      0.94      0.94      5686


Confusion Matrix:
[[2866   79]
 [ 252 2489]]

Average Accuracy:       0.9300

Standard Deviation:     0.0088
```

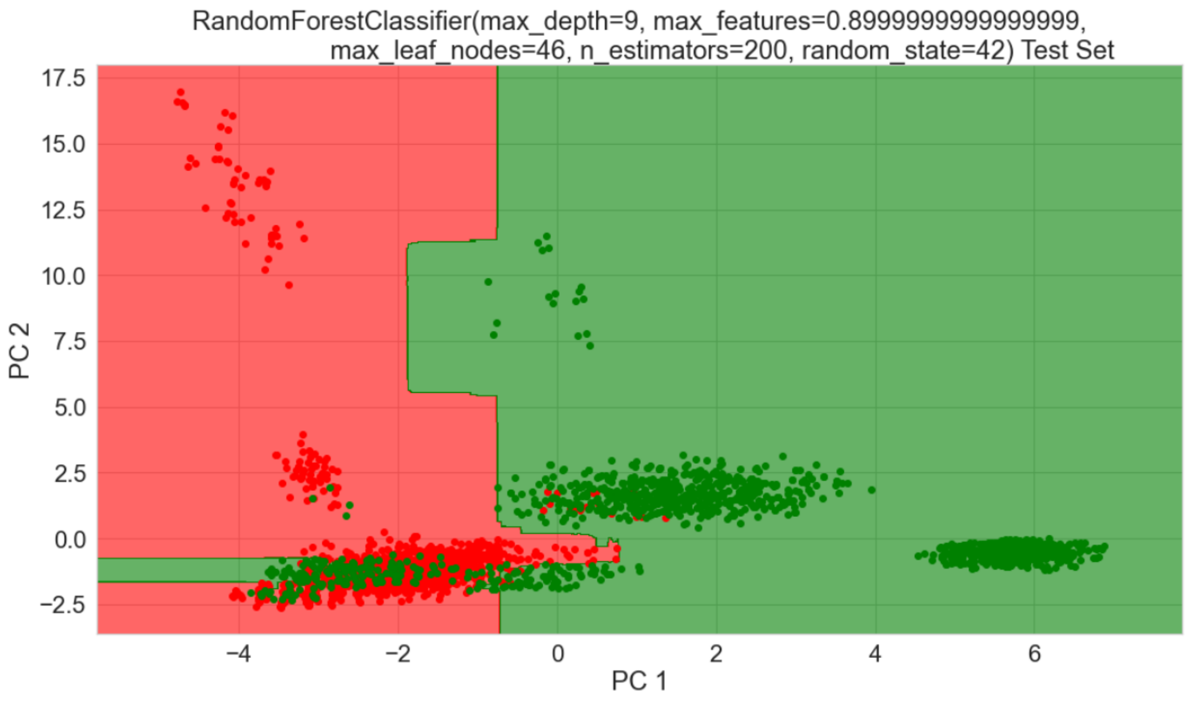


Random Forest Test Results

```python
rf_test_accuracy = print_score(rand_search.best_estimator_,X_train,y_train,X_test,y_test,train=False)
```

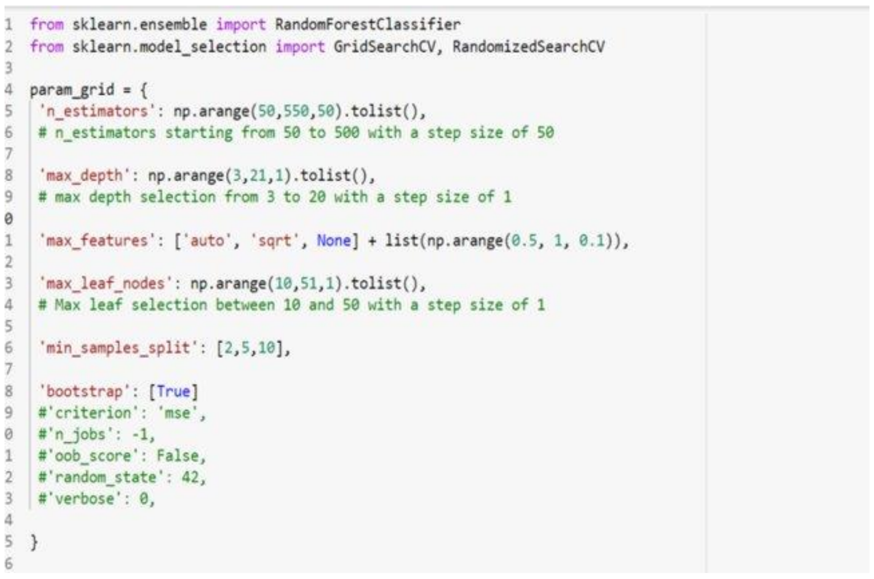
```
Test results:

Accuracy Score: 0.9249

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.96      0.93      1263
           1       0.96      0.88      0.92      1175

    accuracy                           0.92      2438
   macro avg       0.93      0.92      0.92      2438
weighted avg       0.93      0.92      0.92      2438


Confusion Matrix:
[[1218   45]
 [ 138 1037]]
```


RandomForestClassifier(max_depth=9, max_features=0.8999999999999999,
max_leaf_nodes=46, n_estimators=200, random_state=42) Training Set
PC 2
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−4
−2
0
2
4
6
PC 1


RandomForestClassifier(max_depth=9, max_features=0.8999999999999999,
max_leaf_nodes=46, n_estimators=200, random_state=42) Test Set
PC 2
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−4
−2
0
2
4
6
PC 1

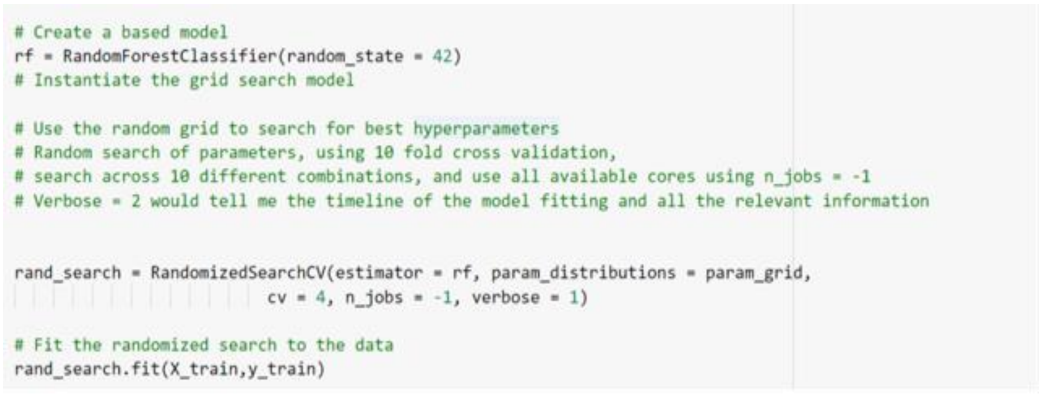
WHY RANDOMIZED SEARCH?

RCV only uses a random search over a specified parameter value, they are faster than GSCV.
RCV can outperform a GSCV, especially if only a small number of hyperparameters affect the performance of the machine learning algorithm.
RCV is also used over GSCV when we have a lot of hyperparameters and using all combinations would be computationally very expensive. For example, Boosted Trees and Neural Networks.



```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

param_grid = {
 'n_estimators': np.arange(50,550,50).tolist(),
 # n_estimators starting from 50 to 500 with a step size of 50

 'max_depth': np.arange(3,21,1).tolist(),
 # max depth selection from 3 to 20 with a step size of 1

 'max_features': ['auto', 'sqrt', None] + list(np.arange(0.5, 1, 0.1)),

 'max_leaf_nodes': np.arange(10,51,1).tolist(),
 # Max leaf selection between 10 and 50 with a step size of 1

 'min_samples_split': [2,5,10],

 'bootstrap': [True]
 #'criterion': 'mse',
 #'n_jobs': -1,
 #'oob_score': False,
 #'random_state': 42,
 #'verbose': 0,

}
```

Fig (20): Code to find the best hyperparameters



```python
# Create a based model
rf = RandomForestClassifier(random_state = 42)
# Instantiate the grid search model

# Use the random grid to search for best hyperparameters
# Random search of parameters, using 10 fold cross validation,
# search across 10 different combinations, and use all available cores using n_jobs = -1
# Verbose = 2 would tell me the timeline of the model fitting and all the relevant information


rand_search = RandomizedSearchCV(estimator = rf, param_distributions = param_grid,
                                 cv = 4, n_jobs = -1, verbose = 1)

# Fit the randomized search to the data
rand_search.fit(X_train,y_train)
```

Fig (21): Initiate a RandomForest classifier model with the best hyperparameters

ROC – AUC Curve:

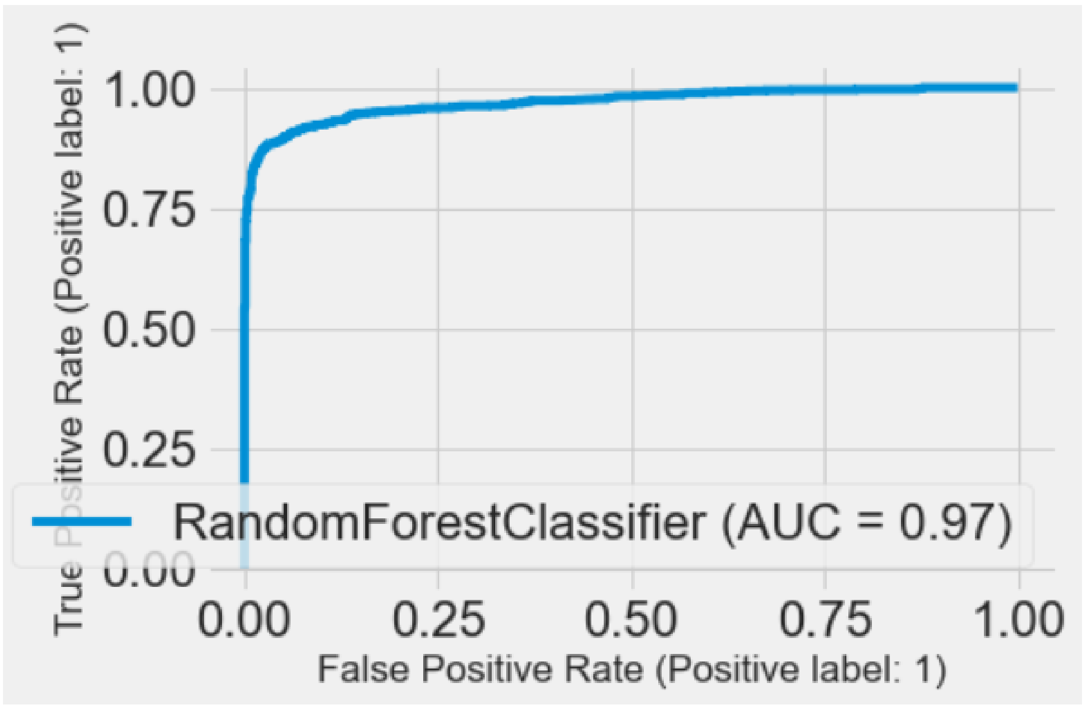


Fig (22): ROC-AUC curve for our best model (RandomForestClassifier)

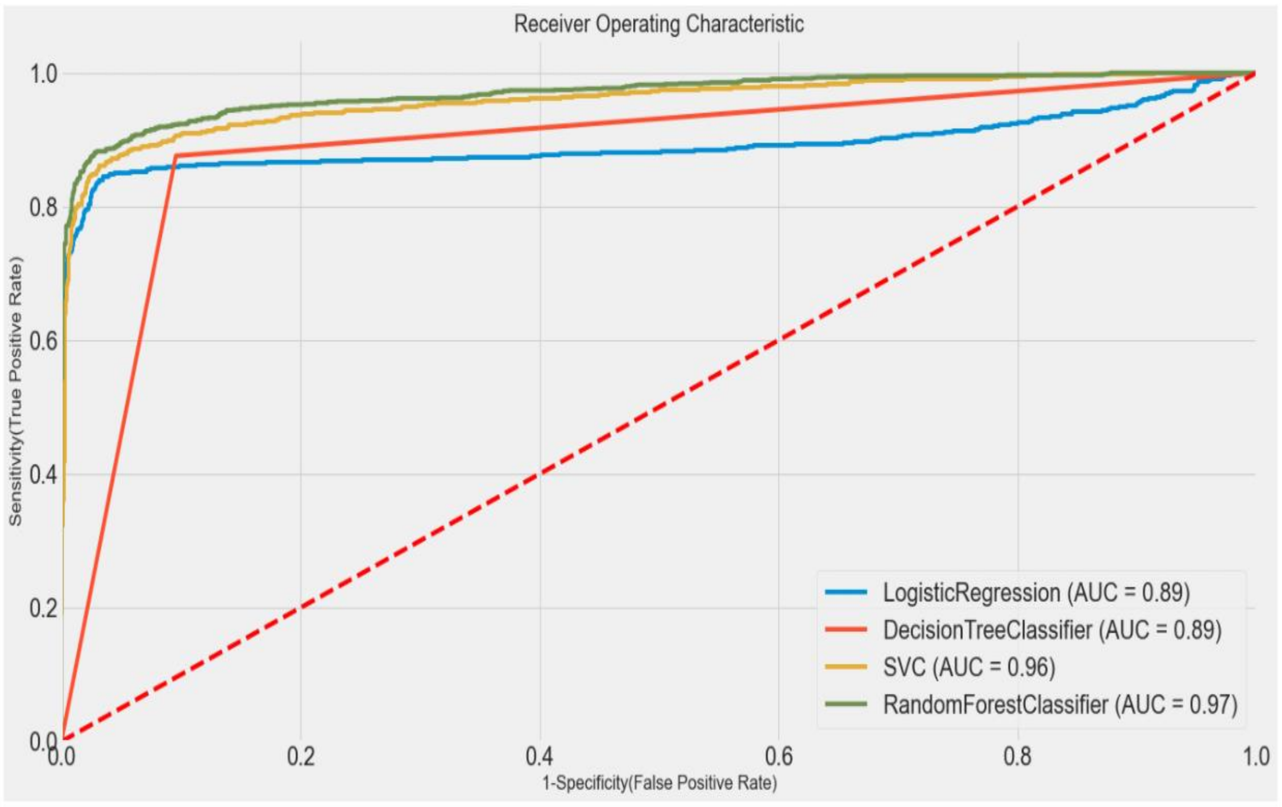
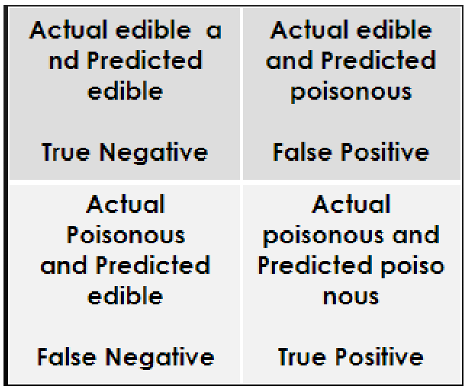


Fig (23): ROC-AUC curve for all of our models

CONFUSION MATRIX:

For our Logistic Regression, Support Vector Classifiers, Decision Tree and Random Forest.



Actual edible a nd Predicted edible True Negative	Actual edible and Predicted poisonous False Positive
Actual Poisonous and Predicted edible False Negative	Actual poisonous and Predicted poiso nous True Positive

Fig (23): Definition of Confusion Matrix

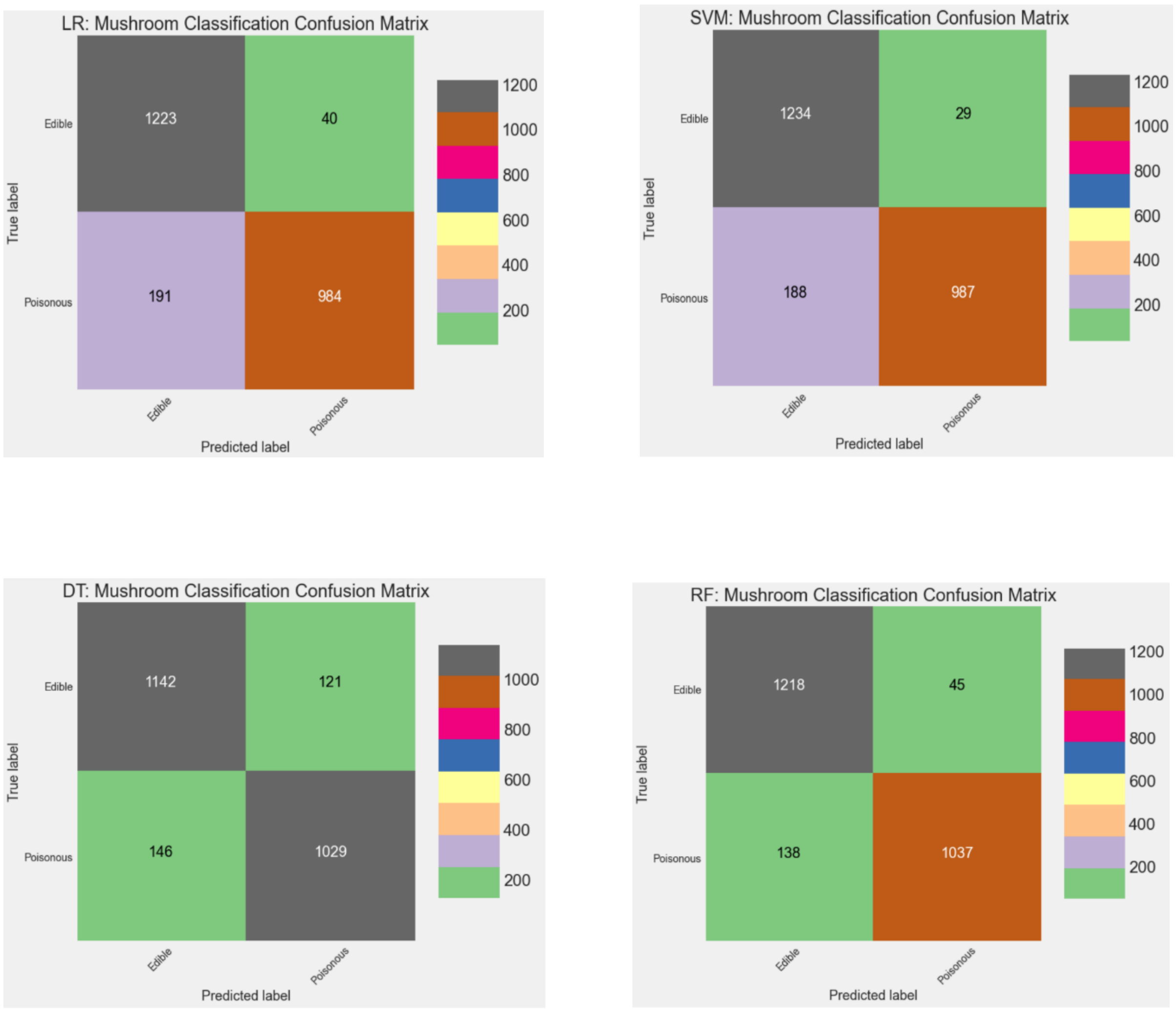
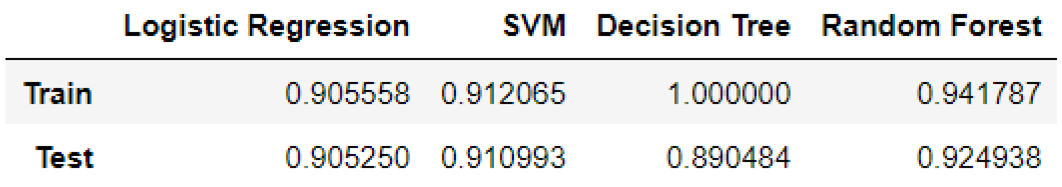


Fig (24): Confusion Matrices for all the models

RESULTS:



	Logistic Regression	SVM	Decision Tree	Random Forest
Train	0.905558	0.912065	1.000000	0.941787
Test	0.905250	0.910993	0.890484	0.924938

In this experimental study, six machine learning algorithms were used. These algorithms are LR, DT, SVM, and RF. All these algorithms were applied to the UCI Mushroom Classification Dataset. Data was divided into two portions, training data, and testing data, both these portions consisting of 70% and 30% data respectively. Feature scaling using StandardScaler() was performed. The Principal Component Analysis (PCA) algorithm was used with n_components = 2 for reducing the dimensions and selecting the best features from the dataset. All six algorithms were applied to the same dataset and results were obtained. Predicting accuracy is the main evaluation parameter that is used in this work. Accuracy is the overall success rate of the algorithm. True Positives (TP), True Negatives (TN), False Negatives (FN), and False Positives (FP) predicted by all the algorithms are presented in Table 1. In our case, TP means actual edible mushrooms. TN, actual poisonous mushrooms. FP, 'actually poisonous' but predicted to be edible. FN, 'actually edible' but predicted to be poisonous.

The training and test set visualizations are given below:

- **Logistic Regression**
- **Support Vector Machine**
- **Decision Tree**
- **Random Forest Classifier**

We plotted a Receiver Operator Characteristic (ROC) curve which is an evaluation metric for binary classification problems, in our case, mushroom classification. It is a probability curve that plots the TPR against FPR at various threshold values and essentially separates the 'signal' from the 'noise'. The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

It is evident from the plot that the AUC for the Random Forest ROC curve is higher than others. Therefore, we can say that Random Forest performed better than other classifiers. The training accuracy score, average accuracy score, standard deviation, and test accuracy score of all six algorithms is given in the tables mentioned above:

CONCLUSION:

In this paper, six popular supervised machine learning algorithms are used for classifying mushrooms into edible or poisonous. These include LR, DT, SVM and RF. Predictions were made about mushrooms (whether edible or poisonous) on the UCI mushroom classification dataset consisting of 8124 records. Principal Component Analysis (PCA) algorithm is used with n_components = 2 for reducing the dimensions of the dataset. There are a total of 23 categorical variables in this dataset which were converted into dummy/indicator variables. These 23 variables (which became 95 after conversion), were reduced to only 2 variables i.e., Principal Components. All six classification models were trained over these two principal components.
From the experimental results obtained, Random Forest gave the highest test accuracy of 92.49% followed by Support Vector Machine with 91.09% test accuracy, Logistic Regression with 90.52% test accuracy, and Decision Tree with 89.04% test accuracy.

REFERENCES:

[1] Bhandari, Aniruddha. "Feature Scaling for Machine Learning: Understanding the Difference Between Normalization vs. Standardization." Analytics Vidhya, 2020. https://www.analyticsvidhya.com/blog/2020/04 /feature-scaling-machine-learningnormalization-standardization.

[2] Brownlee, Jason. "Principal Component Analysis for Dimensionality Reduction in Python." Machine Learning Mastery, 2020. https://machinelearningmastery.com/principalcomponents-analysis-for-dimensionalityreduction-in-python/

[3] Kishan, Maladkar why-is-random-search-better-than-grid-search-for-machine-learning://analyticsindiamag.com/why-is-random-search-better-than-grid-search-for-machine-learning