

# **IE7374 FINAL PROJECT REPORT**

## **Mushroom Edibility Classification**

### **Group 10**

Alisha Kevin Braganza

Prathyush Pothuneedi

Harshini Daggubati

Srithan Reddy Savela

[alisha.b@northeastern.edu](mailto:alisha.b@northeastern.edu)

[pothuneedi.p@northeastern.edu](mailto:pothuneedi.p@northeastern.edu)

[daggubati.h@northeastern.edu](mailto:daggubati.h@northeastern.edu)

[savela.s@northeastern.edu](mailto:savela.s@northeastern.edu)

Percentage of Effort Contributed by Student 1: 25%

Percentage of Effort Contributed by Student 2: 25%

Percentage of Effort Contributed by Student 3: 25%

Percentage of Effort Contributed by Student 4: 25%

Signature of Student 1: Alisha Kevin Braganza

Signature of Student 2: Prathyush Pothuneedi

Signature of Student 3: Harshini Daggubati

Signature of Student 4: Srithan Reddy Savela

**Submission Date:** 08/08/2022

## **PROBLEM SETTING & DEFINITION:**

Poisoning by wild mushrooms is common and may be fatal or produce merely mild gastrointestinal disturbance or slight allergic reaction. It is important that every mushroom intended for eating be accurately identified. Using a variety of machine learning approaches and algorithms, we will do our best to classify the 2 different types of mushrooms with the aid of this project. Each mushroom is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended (the latter class was combined with the poisonous class).

## **DATA SOURCES:**

Link to the UCI repository:

<https://archive.ics.uci.edu/ml/datasets/Secondary+Mushroom+Dataset>

## **DATA DESCRIPTION:**

We have 21 attributes, 61,069 instances, and 7 outcome classes.

One binary class divided in edible=e and poisonous=p (with the latter one also containing mushrooms of unknown edibility).

Twenty remaining variables (n: nominal, m: metrical)

1. cap-diameter (m): float number in cm
2. cap-shape (n): bell=b, conical=c, convex=x, flat=f, sunken=s, spherical=p, others=o
3. cap-surface (n): fibrous=i, grooves=g, scaly=y, smooth=s, shiny=h, leathery=l, silky=k, sticky=t, wrinkled=w, fleshy=e
4. cap-color (n): brown=n, buff=b, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, blue=l, orange=o, black=k
5. does-bruise-bleed (n): bruises-or-bleeding=t, no=f
6. gill-attachment (n): adnate=a, adnexed=x, decurrent=d, free=e, sinuate=s, pores=p, none=f, unknown=?
7. gill-spacing (n): close=c, distant=d, none=f
8. gill-color (n): see cap-color + none=f
9. stem-height (m): float number in cm

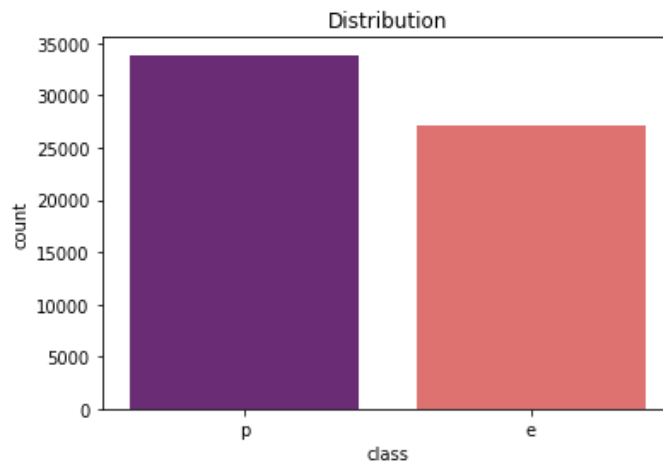
10. stem-width (m): float number in mm
11. stem-root (n): bulbous=b, swollen=s, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r
12. stem-surface (n): see cap-surface + none=f
13. stem-color (n): see cap-color + none=f
14. veil-type (n): partial=p, universal=u
15. veil-color (n): see cap-color + none=f
16. has-ring (n): ring=t, none=f
17. ring-type (n): cobwebby=c, evanescent=e, flaring=r, grooved=g, large=l, pendant=p, sheathing=s, zone=z, scaly=y, movable=m, none=f, unknown=?
18. spore-print-color (n): see cap color
19. habitat (n): grasses=g, leaves=l, meadows=m, paths=p, heaths=h, urban=u, waste=w, woods=d
20. season (n): spring=s, summer=u, autumn=a, winter=w

## SPLITTING THE DATASET

The first step we take before proceeding is to split the dataset into a train, validate and test set. This helps us get rid of any inherent bias that we would incorporate during the Feature Selection/EDA stage. We split our data in the ratios of 80, and 10 and 10. Training 80%, Validation 10% and Testing 10%.

## EXPLORATORY DATA ANALYSIS

We plot a bar chart for the number of values present in each class.



The count of each class varies as we can see from the above graph. The poisonous mushroom data is slightly higher than edible mushroom data. We handle this using undersampling to ensure accurate results for the model.

## DATA TRANSFORMATION

One hot encoding is performed on the categorical values of the dataset, to make them more useful in the model. We also perform standardization on the values that are not encoded. The standardization is performed using standard scalar functions.

## DATA CLEANING - NULL handling

The dataset was checked for null values and the below was observed

class	0
cap-diameter	0
cap-shape	0
cap-surface	14120
cap-color	0
does-bruise-or-bleed	0
gill-attachment	9884
gill-spacing	25063
gill-color	0
stem-height	0
stem-width	0
stem-root	51538
stem-surface	38124
stem-color	0
veil-type	57892
veil-color	53656
has-ring	0
ring-type	2471
spore-print-color	54715
habitat	0
season	0

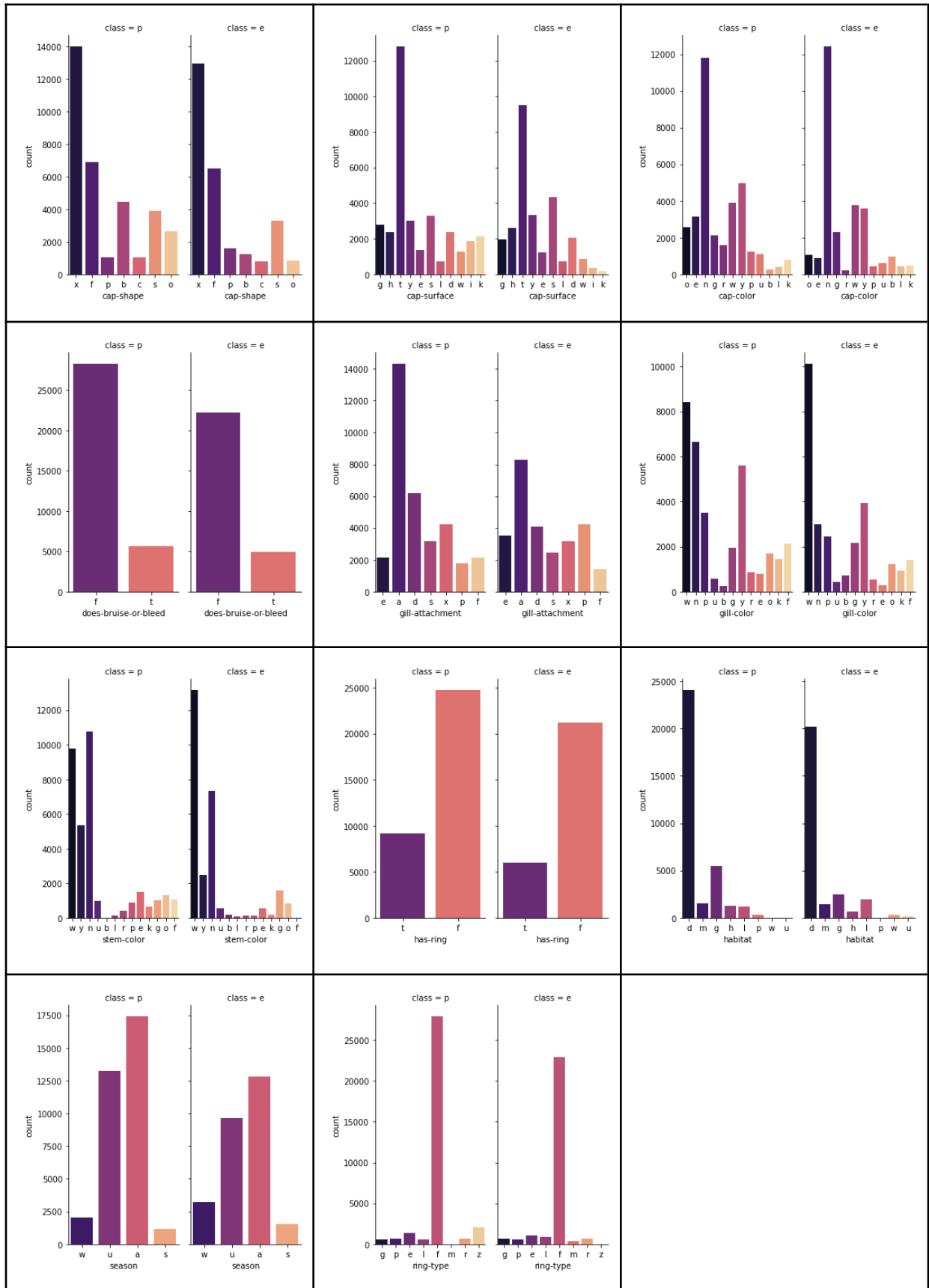
The variables stem-root, veil-type, veil-color, spore-print-color, gill-spacing, and stem-surface are removed from the dataset as they contain ~50% null values. The other variables of cap-surface, gill-attachment and ring-type are imputed using categorical functions based on maximum occurring value.

## UNIVARIATE ANALYSIS OF THE FEATURES

The distribution of the variable categories are studied across both classes to determine distinguishing features for the class.

The cap-shape feature has an almost equal number of poisonous and edible units for the values x, f, s and c. However, the count of poisonous and edible units differ by a large scale for the values p, b and o.

When the cap-surface feature has a value of 't', it is evident that the number of poisonous class members stood high and above 12000 and the number of edible class members stood below 10,000. Except for the number "k," where the count of the poisonous class is significantly higher than the count of the edible class, all other values for the feature are different but not significantly different.



For the “cap-color” feature, the count of 2 classes differs significantly for the values o, e, and r but not much for the other values.

The feature “does-bruise-or-bleed” is simple because of only 2 values - f and t. When the value is equal to f, the count of poisonous class outnumbers the edible class. However, when the value is t, the count of both classes stays almost the same.

The feature “gill-attachment” is simple because of only 2 values - f and t. When the value is equal to f, the count of poisonous class outnumbers the edible class. However, when the value is t, the count of both classes stays almost the same. Coming to the “gill-color” graph, except for a few raises and falls, all the values have a count of classes that looks identical.

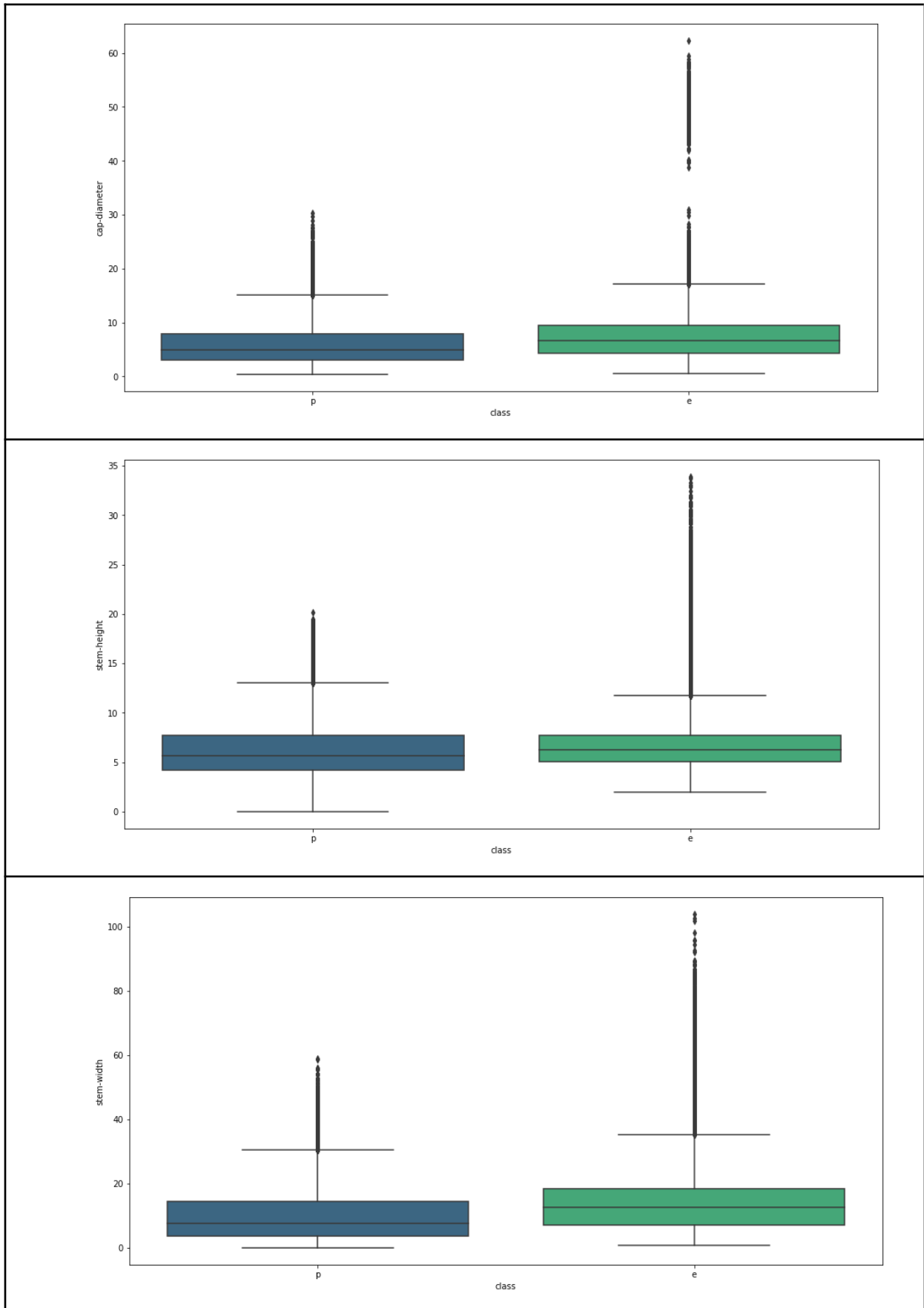
The graph of “stem-color” distribution is quite interesting. The value ‘w’ has higher count of edible class and the values ‘y’, ‘n’ have substantially higher count of poisonous class. The ‘has-ring’ attribute has only 2 possible values - t and f. And both of those values have a slightly higher count of poisonous class. The next feature ‘habitat’ is quite simple. Except for the value d, the count of classes for other values is relatively very low. The value ‘d’ has a higher count of poisonous class.

---

There are 3 boxplots plotted below for features: cap-diameter, stem-height, and stem-width.

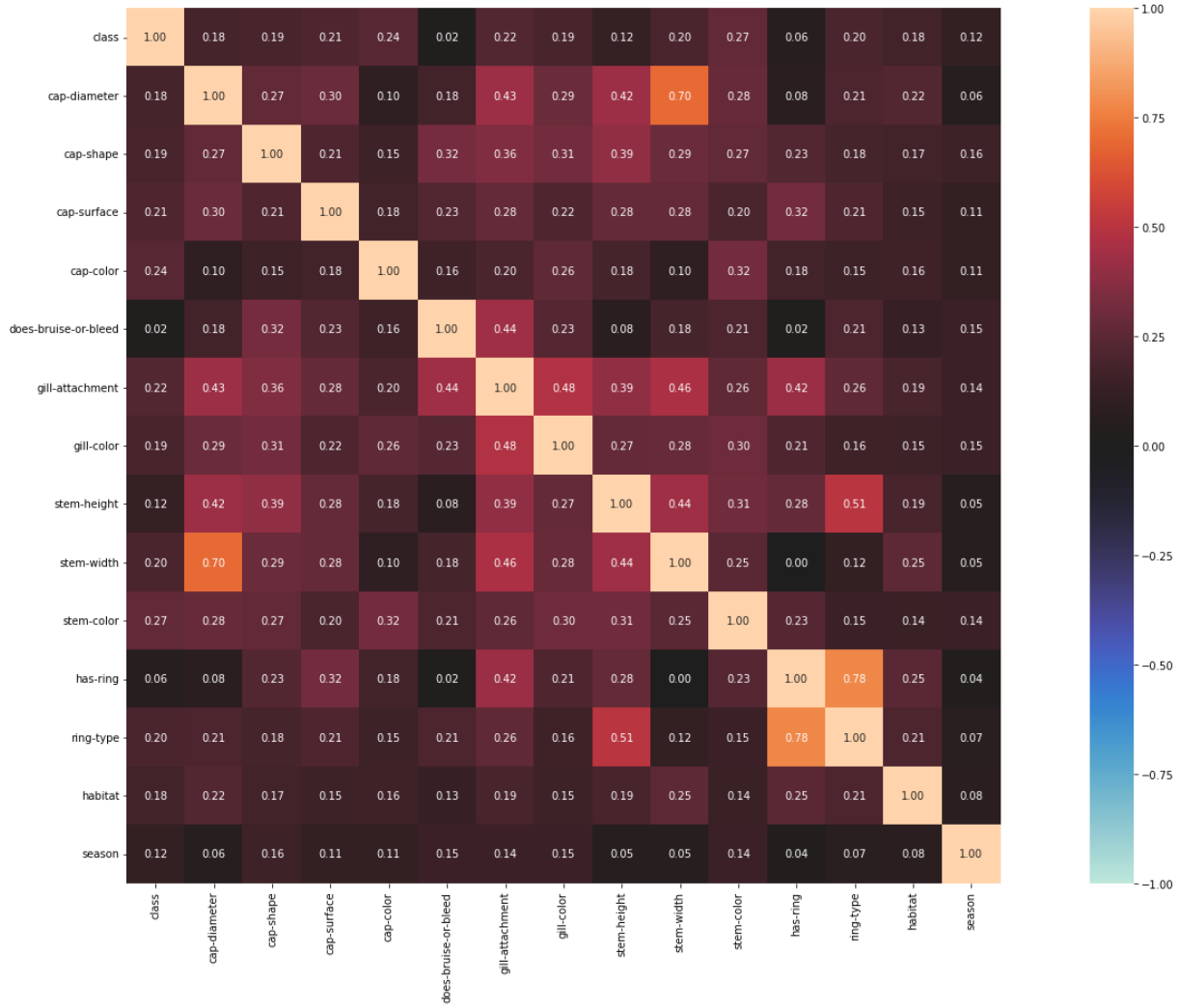
In the cap-diameter feature, we can see that the mean of poisonous class is lower than the mean of edible class. And also there are a lot of outliers in the edible class. In the stem-height feature, it is evident that the mean of both poisonous and edible classes are almost the same. However, in the stem-width feature, the mean varies again. The mean is very less for the poisonous class.

All the box plots are right skewed.

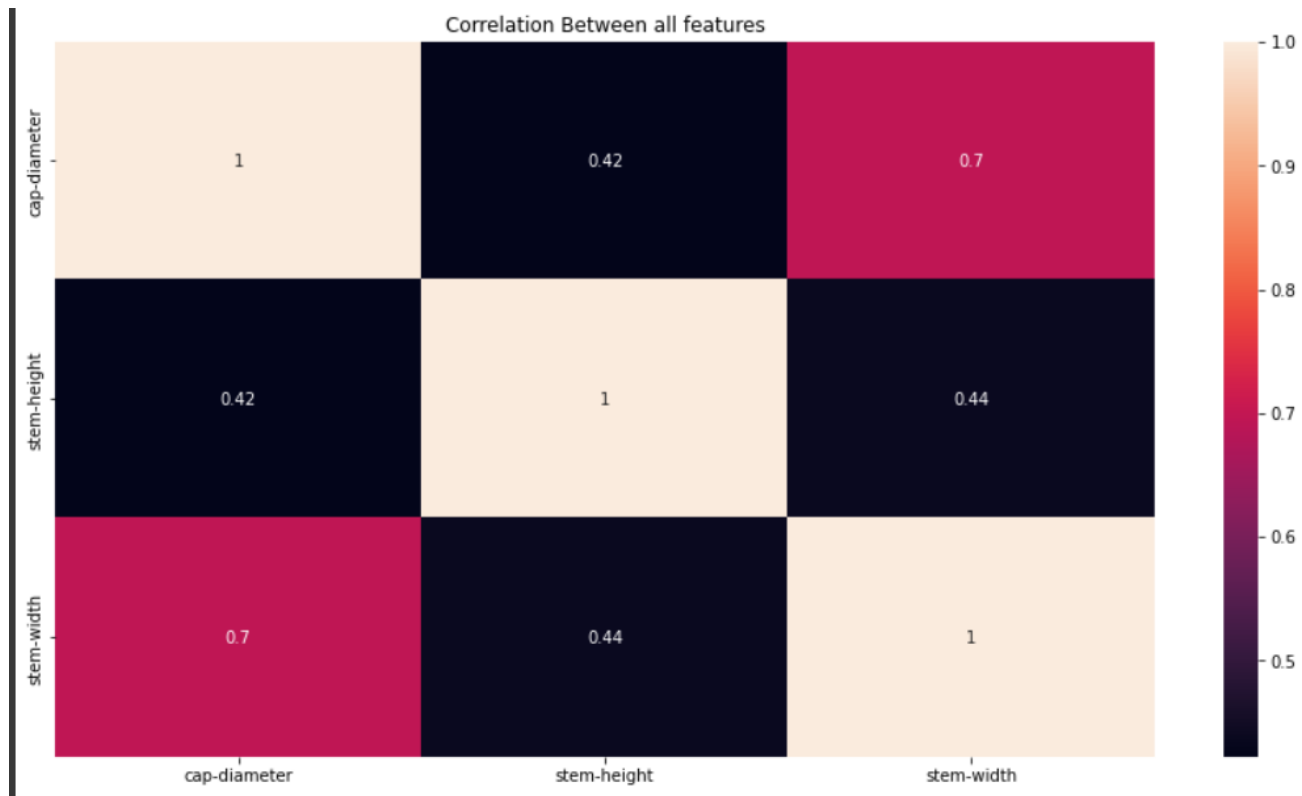




## MULTIVARIATE ANALYSIS OF THE FEATURES



In this, we took the heat map between all the parameters. we see that there is a strong correlation between the features “has-ring” and “ring-type”.

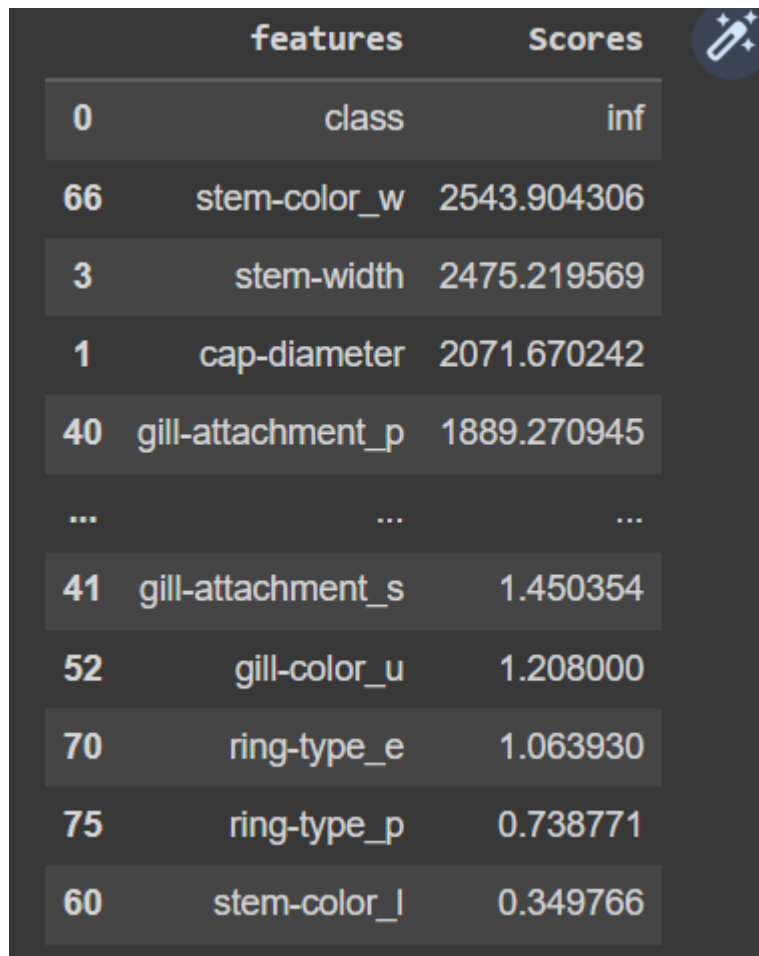


Similarly we can see that there is a strong positive correlation between cap-diameter and stem-width.

## UNIVARIATE FEATURE SELECTION

### ANOVA / F-VALUE

F-test estimates the degree of linear dependency between a feature and the target variable. The below image shows the top 5 scores and the bottom 5 scores of the dataset.



	features	Scores
0	class	inf
66	stem-color_w	2543.904306
3	stem-width	2475.219569
1	cap-diameter	2071.670242
40	gill-attachment_p	1889.270945
...	...	...
41	gill-attachment_s	1.450354
52	gill-color_u	1.208000
70	ring-type_e	1.063930
75	ring-type_p	0.738771
60	stem-color_l	0.349766

## DIMENSIONALITY REDUCTION

### 1. Principal Component Analysis:

Principal component analysis selects the components that have the highest variance captured. The components are calculated by projecting values onto the eigenvectors of the respective covariance matrices. We do eigenvalue decomposition to find the eigenvalues and eigenvectors. If the data set is not a square matrix, we can decompose using singular value decomposition.

The first n components from the singular values decomposition, when calculated the maximum variance captured from the formula:

$$\frac{\sum_{i=1}^n s_i}{\sum_{i=1}^m s_i}$$

n = first n eigenvalues

M = sum of all eigenvalues

This returns the variance captured by first n components.

Step1: make the values mean-centered

Step2: find the covariance of the matrix

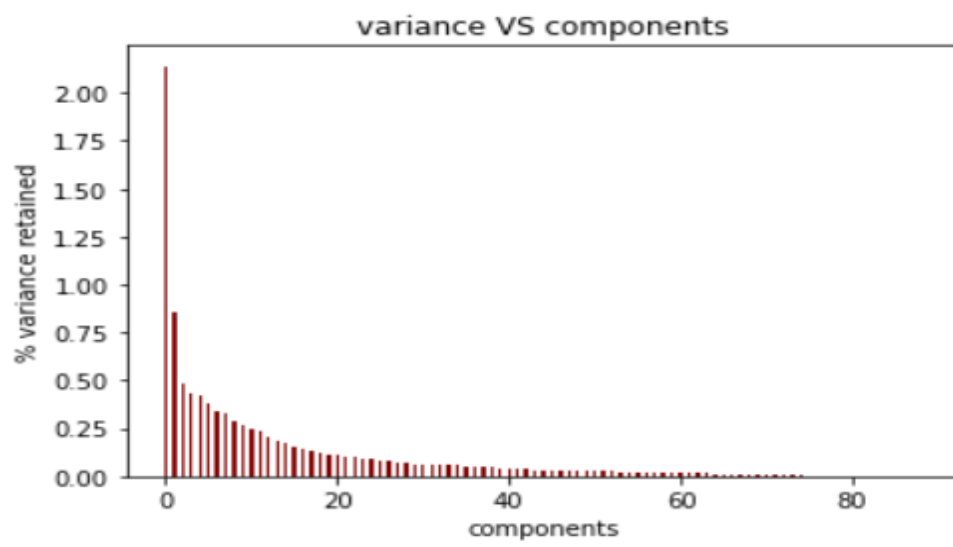
Step3: find eigenvalues and eigenvectors of the covariance matrix

Step4: sort eigenvectors

Step6: pick top n eigenvectors

Step4: project data onto the eigenvectors

We see that the first 50 components help us capture 96.15% of the variance in the data. Therefore we proceed with this data as our dimensionality reduced data.



## MODEL IMPLEMENTATION

We believe that we can solve this classification problem using various Machine Learning algorithms. Some of which are:

1. Logistic Regression - Baseline model
2. Support Vector Machines
3. Naive Bayes
4. Neural Networks

### Logistic Regression- Baseline Model

Logistic regression, despite its name, is a classification model rather than a regression model. It uses input variables (features) to predict a categorical outcome variable (label) that can take on one of a limited set of class values. A binomial logistic regression is limited to two binary output categories, while a multinomial logistic regression allows for more than two classes. Examples of logistic regression include classifying a binary condition as “healthy”/“not healthy”, or an image as “bicycle”/“train”/“car”/“truck”.

#### Algorithm:

- 1) We have used sigmoid in logistic regression where the dependent variable is only one class. We used the sigmoid function since we have only 2 classes.

```
def sigmoid(self, z):  
    sig = 1 / (1 + np.exp(-z))  
    return sig
```

- 2) The categorical cross-entropy loss function calculates the loss of an example by computing the following sum:

$$\text{Loss} = - \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

```
def costFunction(self, X, y):  
    pred_ = np.log(np.ones(X.shape[0]) + np.exp(X.dot(self.w))) - X.dot(self.w)*y  
    cost = pred_.sum()  
    return cost
```

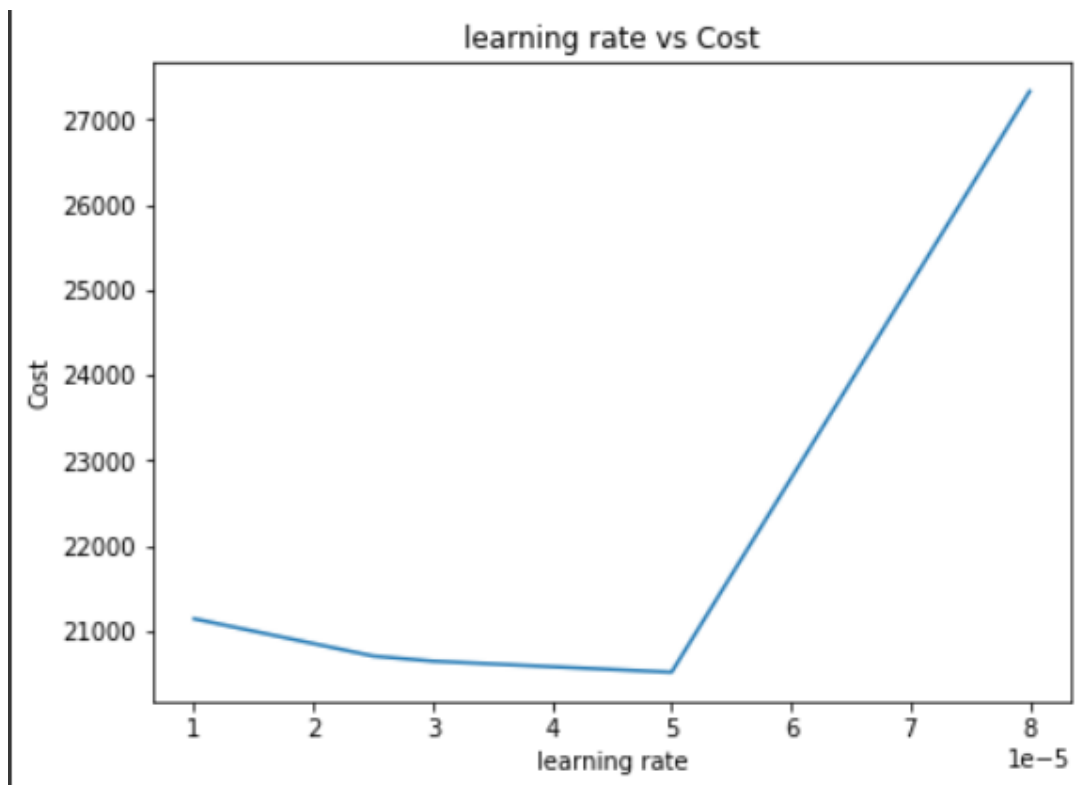
### Check for Overfitting/Underfitting

The Model was first tested on the training and Validation datasets. We don't see a huge difference between the training and validation error. This tells us that the model has not been over or under fitted. Had there been overfitting / underfitting it would need to be addressed using regularization.

#### Results on Validation dataset

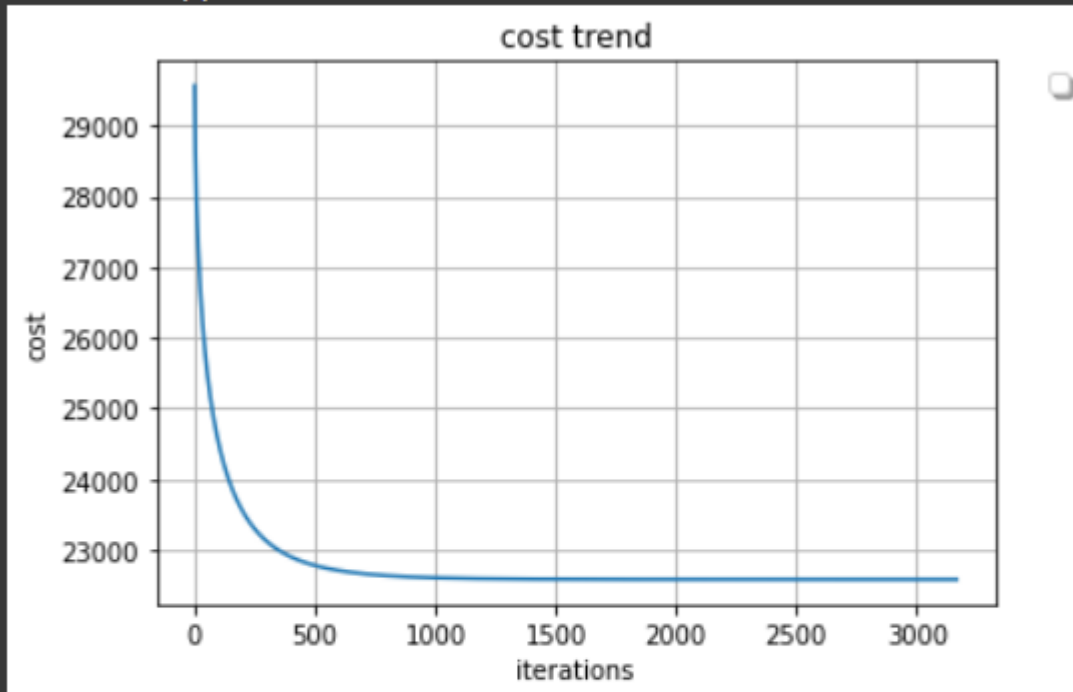
Learning Rate	Accuracy	Precision	Recall	F1 Score	Iterations needed
0.000025	0.801814	0.78274	0.776722	0.789069	4000
0.00003	0.80290	0.7834	0.7772	0.78985	4000
0.00005	0.80217	0.7831	0.7770	0.78942	4000
0.00008	0.9647	0.6420	0.5897	0.7320	4
0.00001	0.800	0.7825	0.777	0.7887	4000
0.005	0.6762	0.4860	0.4948	0.57147	1

We can observe that an overall good performance on the validation set is given by the model with a learning rate of 0.00001,0.000025,0.00003,0.00005. We will use 0.00001 as we see the cost increases as the learning rate increases in the below graph.



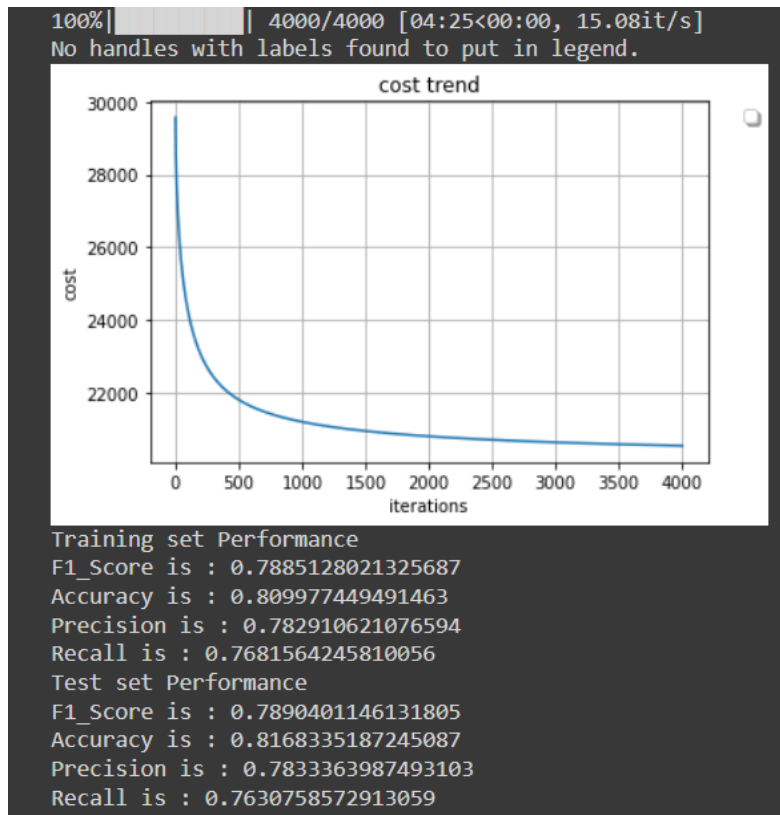
Applying the learning rate to the PCA dataset we see a slight drop in model performance across all parameters. However the iterations and time taken to converge significantly reduce.

```
79%|███████ | 3165/4000 [00:54<00:14, 57.77it/s]  
No handles with labels found to put in legend.  
Model Stopped
```

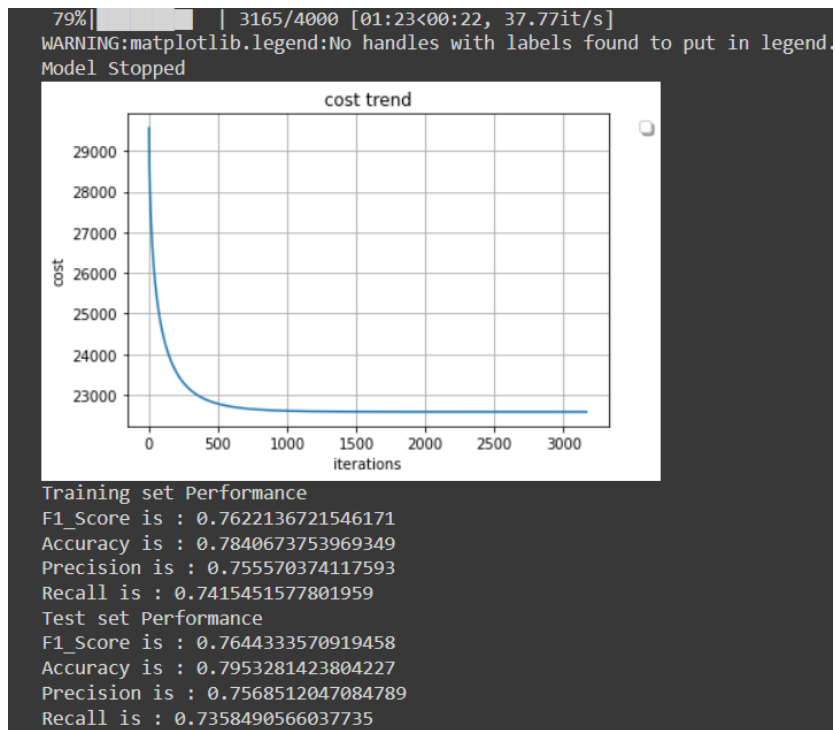


```
Training set Performance  
F1_Score is : 0.7622136721546171  
Accuracy is : 0.7840673753969349  
Precision is : 0.755570374117593  
Recall is : 0.7415451577801959  
Test set Performance  
F1_Score is : 0.7692033505613973  
Accuracy is : 0.7833030852994556  
Precision is : 0.7617733627667402  
Recall is : 0.7556022408963585
```

## Results on test data set, without using PCA



## Results on test data set, using PCA





## Support Vector Machine

“Support Vector Machine” (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.

Training time required for the regular SVM model was too high, therefore we adopted SMO as our dataset is small and this will satisfy our requirements.

Sequential minimal optimization (SMO) is an algorithm for solving the quadratic programming (QP) problem that arises during the training of support-vector machines (SVM). The dataset that is being used in this project is not a huge dataset. It is fairly simple in terms of attributes and records. For this dataset, Simplified version of SMO is the best and ideal choice.

### SMO Algorithm

- Input: C, kernel, kernel parameters, epsilon
- Initialize b and all  $\alpha$ 's to 0
- Repeat until KKT satisfies (to within epsilon):
  - Find an example e1 that violates KKT (prefer unbound examples here, choose randomly among those)
  - Choose a second example e2. Prefer one to maximize step size (in practice, faster to just maximize  $|E1 - E2|$ ). If that fails to result in change, randomly choose unbound example. If that fails, randomly choose an example. If that fails, re-choose e1.
  - Update  $\alpha 1$  and  $\alpha 2$  in one step
  - Compute new threshold b

### RESULTS ON VALIDATION DATA SET

Kernel	C	Data	Accuracy	Time(Mins)
Linear	3	Regular	76.93	16.40
Gaussian	3	Regular	79.06	17.06
Quadratic	3	Regular	77.04	16.99

Linear	5	Regular	77.2	16.73
Gaussian	5	Regular	77.09	17.23
Quadratic	5	Regular	75.05	16.67
Gaussian	3	PCA	76	11.48

As PCA with Gaussian Kernel gives us good accuracy within minimum time we use this on the test data.

```
[53] model = SVM(max_iter=3, kernel_type='gaussian', C=3.0, epsilon=0.001)
      start_time = time.time()
      model.fit(x_train_pca, data_y)
      print("--- %s mins ---" % ((time.time() - start_time)/60.0))

Iteration number exceeded the max of 3 iterations
--- 11.489021480083466 mins ---

Y_predicted = [model.predict(x) for x in x_test_pca]
cm = confusion_matrix(data_y_test, Y_predicted)
accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[0][1] + cm[1][0] + cm[1][1])
print(cm)
print(accuracy)

[[2117  580]
 [ 766 1974]]
0.7524370057016737
```

## Gaussian Naive Bayes

Gaussian Naive Bayes is a Machine Learning Generative Algorithm that makes an assumption that the data is Gaussian. We feed in a separate data frame because of this assumption. Some columns of which have been transformed due to the Gaussian Assumption. It considers the data to follow a gaussian distribution.

### The algorithm is as follows:

1. We first create a function to fit the data in a gaussian distribution to find the likelihood. This is done by finding the mean and the standard deviation and these 2 values are fed into the 'norm' function that scipy provides us. It calculates the probability density for us for each dimension.
2. Next we proceed to find the priors for each class. This is defined as the number of rows that give us a class 'K' divided by the total number of rows.
3. We then fit distributions for each dimension for each of the classes.

Having computed the priors and the likelihoods for each class, we then are left with feeding in the test dataset to compute the probability of a certain row belonging to a certain class. This is done with the help of nested loops of an array and dictionary

5. We compare each of the probabilities and find the maximum and then assign that input to that class and continue this process until we compute it for the whole test set

## Results-Gaussian Naive Bayes

Naive bayes requires more time for testing therefore we only tested a small batch of 100 data points.

Error Metric: Number of correct classifications/ Total Classification

Count in the below image is defined as the number of instances misclassified as some other class

```
testing error : 46.0
count for class 0 : 38
count for class 1 : 9
Test Time 0:06:45.817180
```

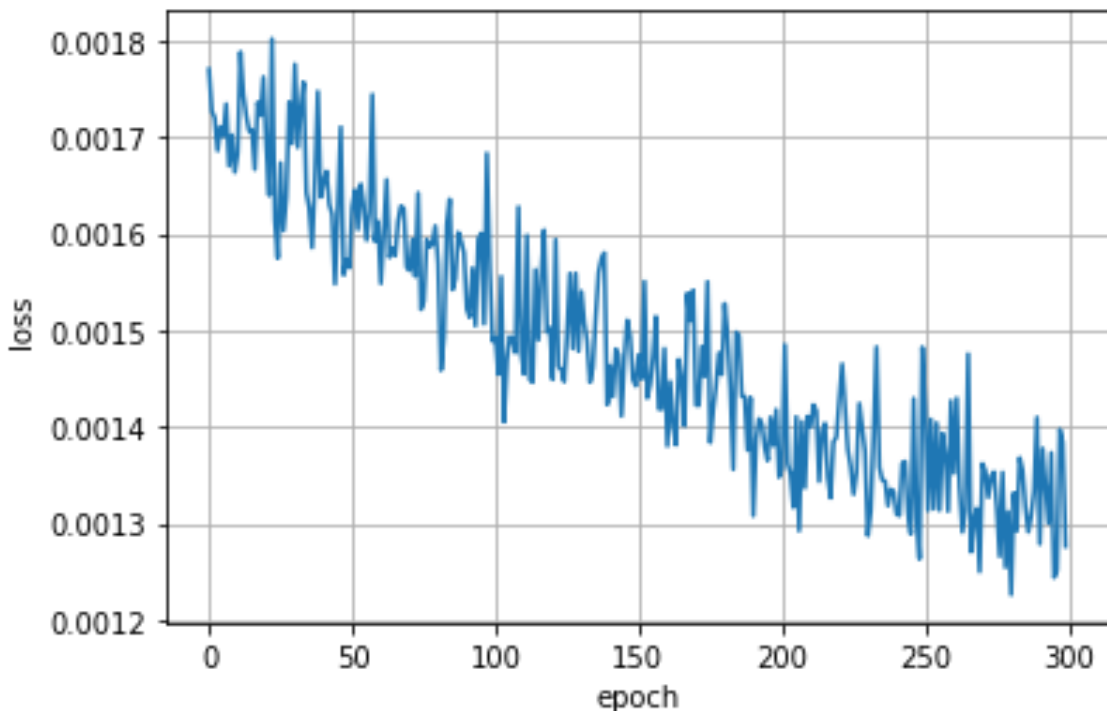
## Neural Networks

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

We have used the RELU activation function in our hidden layers, because the main reason why ReLu is used is because it is simple, fast, and empirically it seems to work well. Empirically, early papers observed that training a deep network with ReLu tended to converge much more quickly and reliably than training a deep network with sigmoid activation.

```
model = Sequential()  
model.add(Dense(89, input_dim=89, activation='relu'))  
model.add(Dense(512, activation='relu'))  
model.add(Dense(256, activation='relu'))  
model.add(Dense(128, activation='relu'))  
model.add(Dense(2, activation='softmax'))
```

**Loss function on training data vs epoch:**



We tried different combinations of the number of neurons in hidden layers, number of epochs and batch size. We achieved accuracy of 99.9% and recall of 99.98% which are better than the results obtained by our baseline model (Logistic Regression). Our run time was 9m 36s.

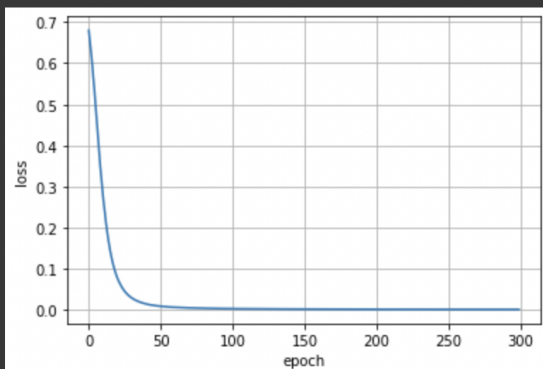
## **RESULTS ON VALIDATION DATA SET**

Epochs	Validation Split	Batch size	Accuracy	Recall	Time taken
300	0.15	256	99.9	99.9	8.37 mins
250	0.2	156	99.9	99.9	10.36 mins
100	0.1	300	99.9	99.9	3.36 mins

Results with PCA

```
Accuracy: 0.999
Recall: 0.999
```

```
loss=history.history['loss']
def plot(loss):
    axis=list(range(0, len(loss),1))
    fig, ax = plt.subplots()
    ax.plot(axis, loss)
    ax.set_xlabel('epoch')
    ax.set_ylabel('loss')
    ax.grid()
    plt.show()
plot(loss)
```



## FINAL RESULTS

We can summarize the results of the models created above as below:

1. Neural Network is the best performing model in comparison to baseline model.
2. Naive bayes is simple to fit, but takes a substantial amount of time to predict values. This should be studied and made more optimal.
3. SVM used with SMO does not perform better than our baseline model.