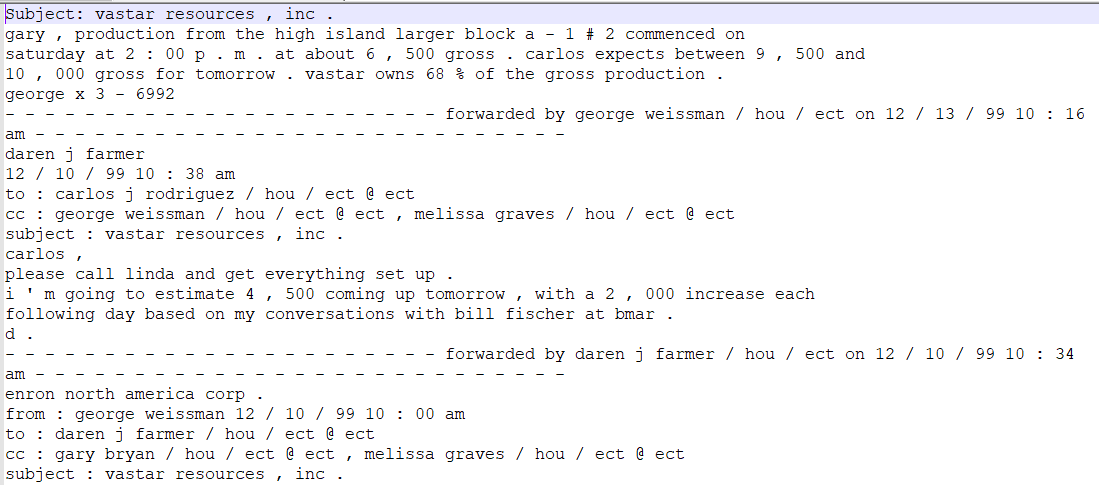
The objective here is to build a supervised classification model for detecting spam and ham emails.

The Dataset has been chosen from the link http://www.aueb.gr/users/ion/data/enron-spam/

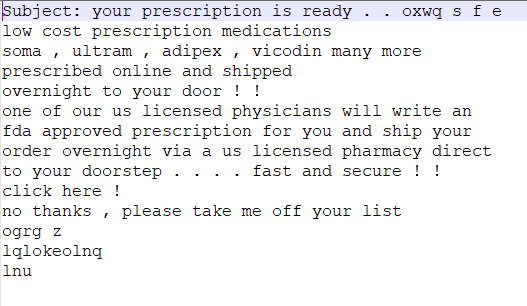
The Dataset contains 2 folders.

1. Ham - 3672 email files.
2. Spam - 1500 email files.

**Sample Ham Data**



**Sample Spam Data**



It is observed that each mail starts with a subject (‘subject:’ or ‘subject :’) followed by the content. This helps in cleaning the email files a bit efficiently and we need not use any algorithm for detecting the files patterns in the initial stages. This might induce a little bias while model building.

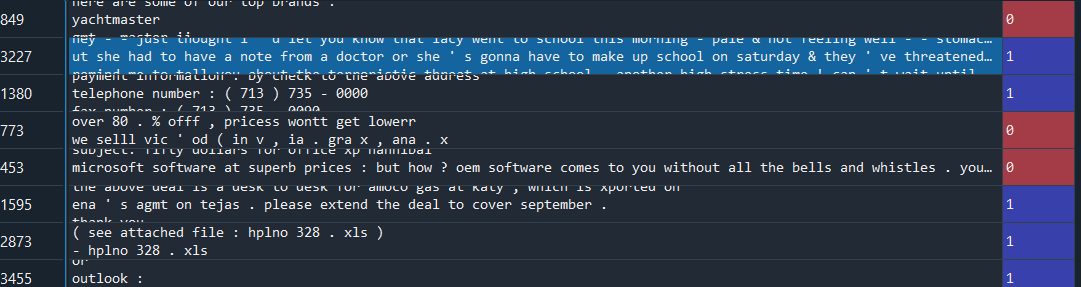
Here, cleaning refers to handling duplicate and empty emails. If a file is either fully empty or contains an empty email i.e. an email with just the subject but no content, then that data will not be processed. However, if a file contains duplicate emails, then that data will be processed after deduplication.

For example, if a file contains “Subject : some random subject\naaaa subject: some random subject\naaaa” then after cleaning the output of the file should be “Subject : some random subject\naaaa”. Here ‘\n’ denotes the Line Feed character. This check can be done by reading the emails line-by-line and saving it in a list which helps in checking duplicates.

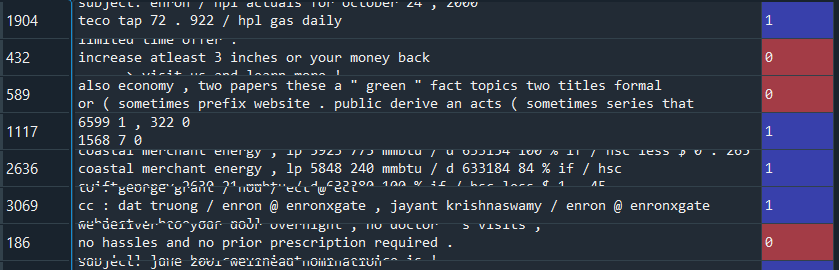
Once the data cleaning is done, we can start the start applying text processing algorithms.

First the data is randomly split into training (70%) and testing (30%). The model will be calibrated on the training data first, until the max performance is achieved. Post that testing data will be used for cross checking the accuracy and out-of-sample error.

**Sample Training Data**



**Sample Testing Data**



Here ‘1’ denotes ham email and ‘0’ denotes spam email.

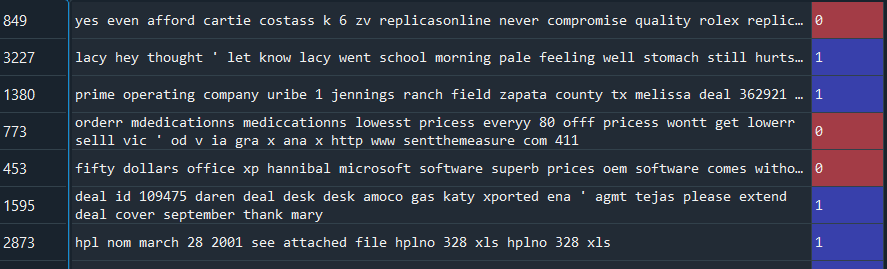
The total number of spam and ham emails in the training dataset are 995 and 4498 respectively.

The total number of spam and ham emails in the testing dataset are 434 and 1797 respectively.

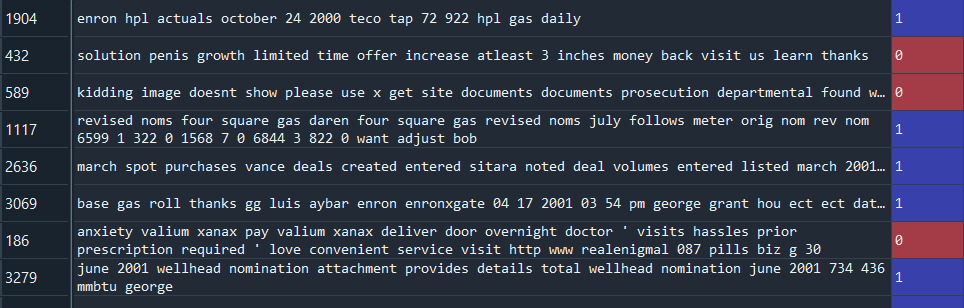


Text data requires some preparation before using it for predictive models. The text must be parsed to remove words called tokenization. This includes removal of stop words such as 'subject', 'subject:', 'subject:', 'bestregards', 'best', 'regards', 'lookingforward', 'forward', 'looking', 'email', 'emailaddress', 're', 're:', 're:', 'to:', 'to:', 'from', 'from:', 'from:', 'fw', 'fw:', 'fw:', '\_', 'cc:', 'cc', 'cc:', 'bcc:', 'bcc:', 'bcc', etc. We also need to perform stemming and lemmatization. This helps us achieve the root form of words and the word produced. This is done in python using Natural Language Processing Toolkit (NLTK) package.

**Sample Training Data after stemming and lemmatization**



**Sample Testing Data after stemming and lemmatization**



There are some words that are gibberish and not present in the dictionary, but it was extremely hard to remove them despite using the Finest NLTK algorithms. To get perfect data, NLTK must be clubbed with regular expressions and applied at a molecular level. After text processing, the words need to be encoded as numerical values for use as input to a machine learning algorithm. This is called feature extraction or vectorization. Usually a “Bag of Words” model is used to capture the word occurrences whilst completely ignoring the relative positional information of the words in the document.

If the “Bag of Words” model merely stores the raw occurrence, then there is a good chance for overfitting and misclassification. Also, overly frequent words such as “that”, “this”, “it”, “they” do not provide much information and hence they should be treated with dealt carefully. To avoid such discrepancies, it is ideal to divide the number of occurrences of each word divided by the total number of words in the document. This new feature is called Term Frequencies (TF). Another enhancement on top of TF is to downscale the weights for the words that occur in many documents. This is called Inverse Document Frequency.

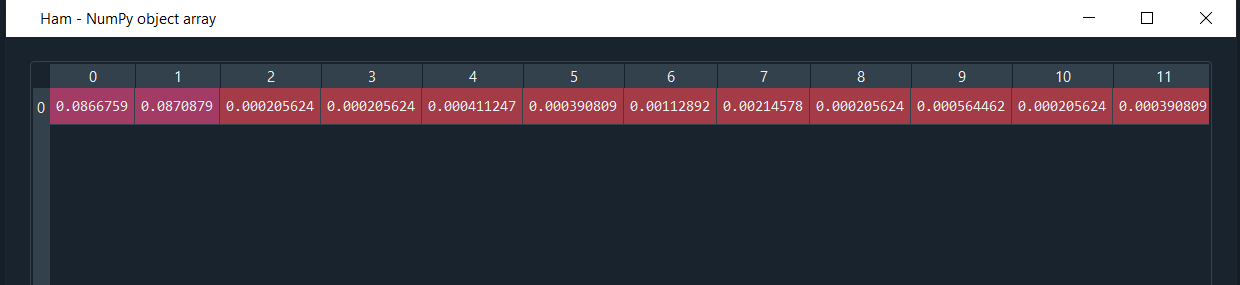
The TF is usually log transformed for reducing the discrepancies.

TF = 1 + log10(count(t, d)) if count(t, d) > 0

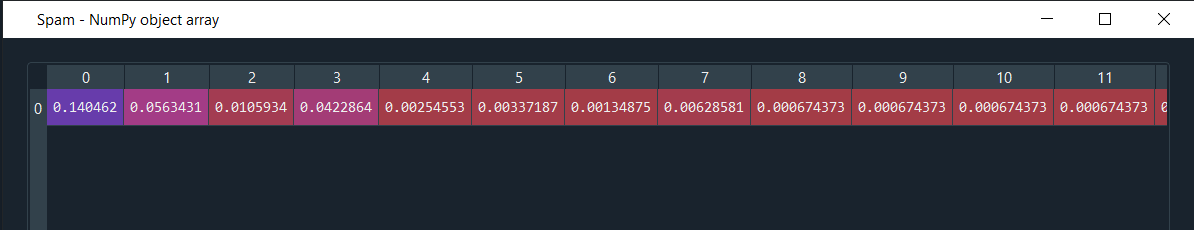
IDF = log(N/df) where N = total number of documents in the collection and df is the number of documents that contain a certain word. Therefore, the TF-IDF value for word ‘t’ in document ‘d’ equals the product of the TF and IDF.

The Scikit-Learn package in python contains the ‘TfidfVectorizer’ function, which is used for tokenizing documents, learning the vocab, IDF weightings and encoding new documents.

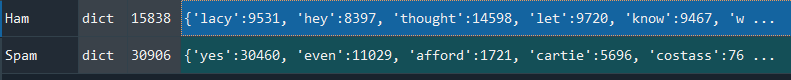
**Sample Ham Tokenized Data**

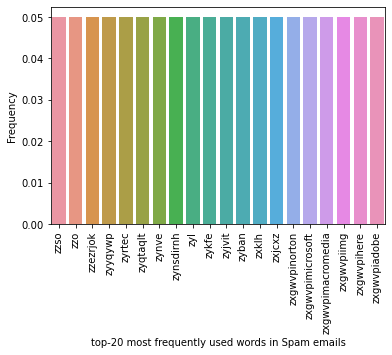


**Sample Spam Tokenized Data**

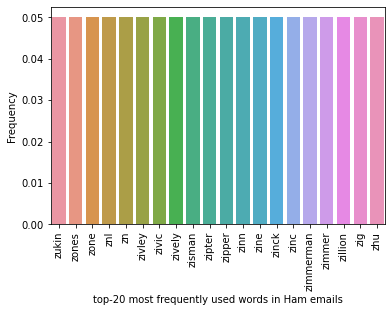


**Sample Ham and Spam Bag of Words**

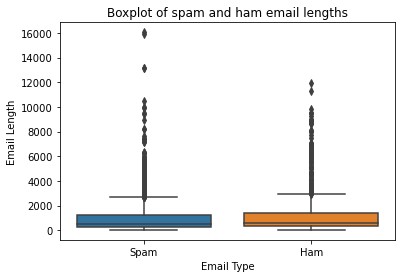




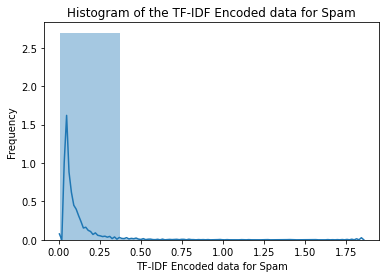
The plot shows the relative frequency of the top-20 most frequently words in Spam emails. Some words are not dictionary words which is due to the inbuilt libraries. To eliminate these words completely some manual intervention is needed.



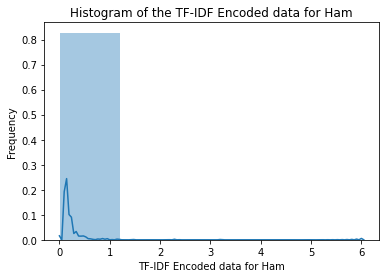
The plot shows the relative frequency of the top-20 most frequently words in Ham emails. The distribution is more spread out due to the higher count of words present in the spam emails. Spam emails tend to possess some words which are often misspelled and may also contain abbreviations. Sometimes, even emojis are misclassified as text and this can lead to some error during predictions.



The boxplot highlights the distribution of ham and spam emails. The spam email lengths are more distributed than ham emails. There are many outliers present in the Spam emails than the Ham emails. This is because spam emails contain too much unnecessary and irrelevant information. Sometimes spam emails have fancy catchphrases which grabs people’s attention. This distribution would not change much even if the words are optimally filtered and tokenized.



The distribution of the TF-IDF spam encoding is skewed to the right which means that the mean<median<mode. This means that the encoder can process majority of the words and rarely leaves room for error.

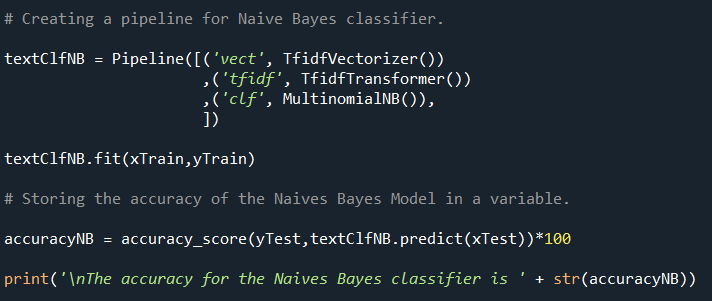


The distribution of Ham email encoding is quite like spam encoding and we can see that there is not much bias in the encoding algorithm.

Once the text processing and feature extraction is fully complete, we can go for model building. We would be using cross validation algorithm in the models to obtain the best estimate. Before performing cross validation, the model must be trained manually. k-cross fold validation is the process in which the training data is randomly split into 'k' folds without replacement, where 'k-1' folds are used for model training and one-fold is used for testing. This algorithm is applied k-times to obtain 'k' models and performance estimates. The models are independent to each other and hence, the average performance is calculated to obtain a performance estimate. This is extremely reliable while processing unseen data and balances the variance-bias trade off.

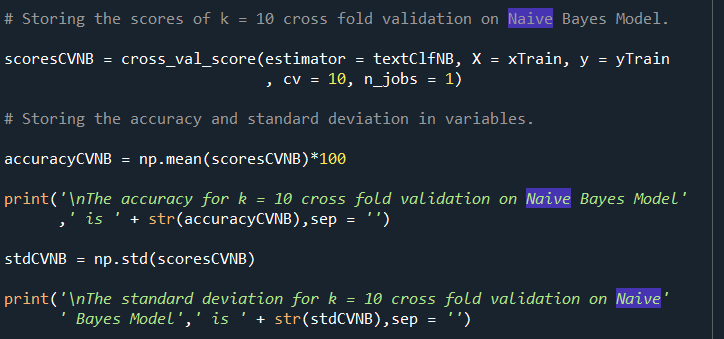
k = 10 is the most used value for small sample sizes. Using a large 'k' value will consume more training data at each iteration and will result in low bias while estimating the model performance. However, computation complexity will increase along with the overall error for every high value.

Here, we are checking the performance of 5 models namely Multinomial Naïve Bayes Classifier, Decision Tree Classifier, Random Forest, Logistic Regression and Support Vector Machines (SVM).



Naïve Bayes Classifier is manually trained before applying cross fold validation technique.

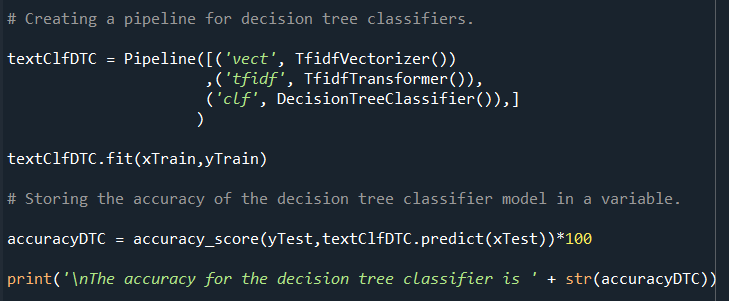
The accuracy for the Naive Bayes classifier is 89.61.



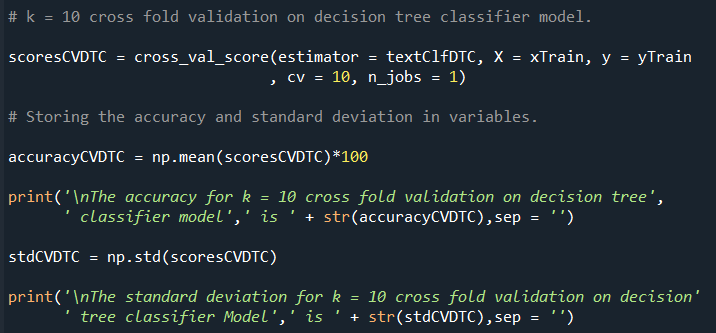
The accuracy for k = 10 cross fold validation on Naive Bayes Model is 88.84.

The standard deviation for k = 10 cross fold validation on Naive Bayes Model is 0.0118.

The accuracy after cross fold validation is not much different from the accuracy of manual training.

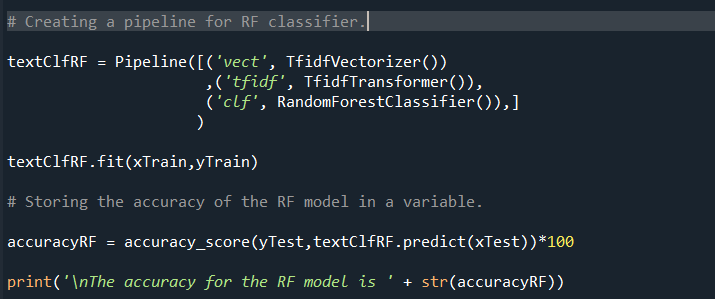


The accuracy for the decision tree classifier is 91.91.

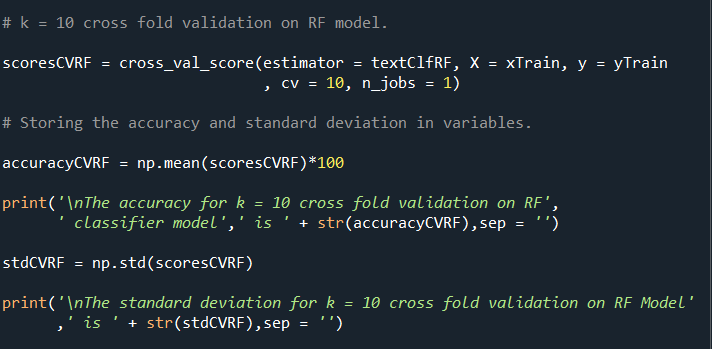


The accuracy for k = 10 cross fold validation on decision tree classifier model is 91.94.

The standard deviation for k = 10 cross fold validation on decision tree classifier Model is 0.0156.

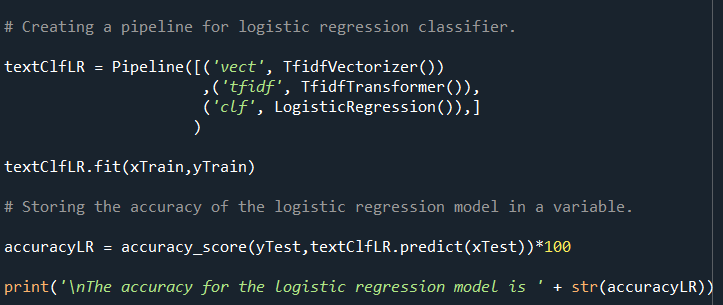


The accuracy for the RF model is 96.91.

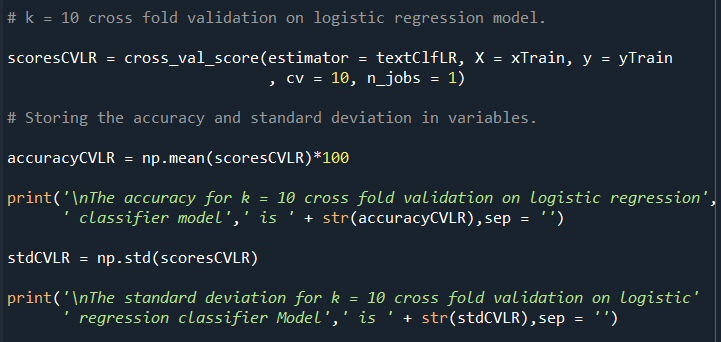


The accuracy for k = 10 cross fold validation on RF classifier model is 97.09.

The standard deviation for k = 10 cross fold validation on RF Model is 0.007.

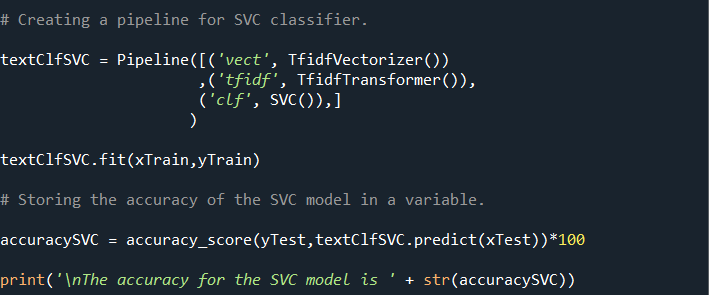


The accuracy for the logistic regression model is 98.81.

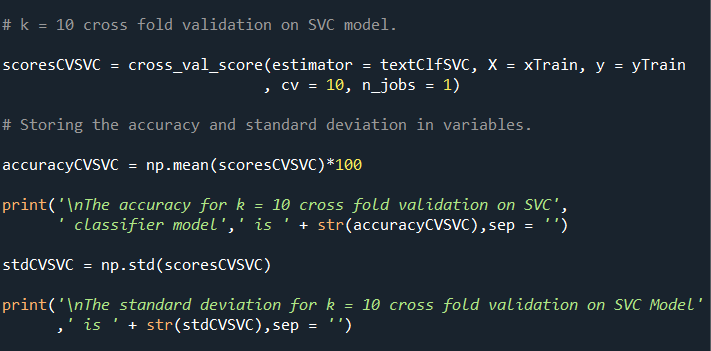


The accuracy for k = 10 cross fold validation on logistic regression classifier model is 97.69.

The standard deviation for k = 10 cross fold validation on logistic regression classifier Model is 0.0075.

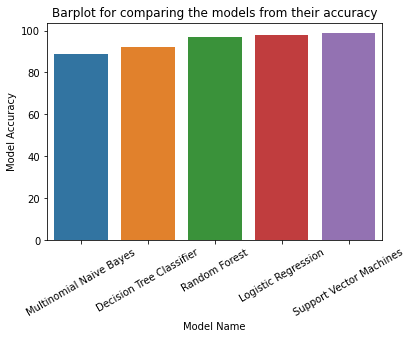


The accuracy for the SVC model is 99.145.



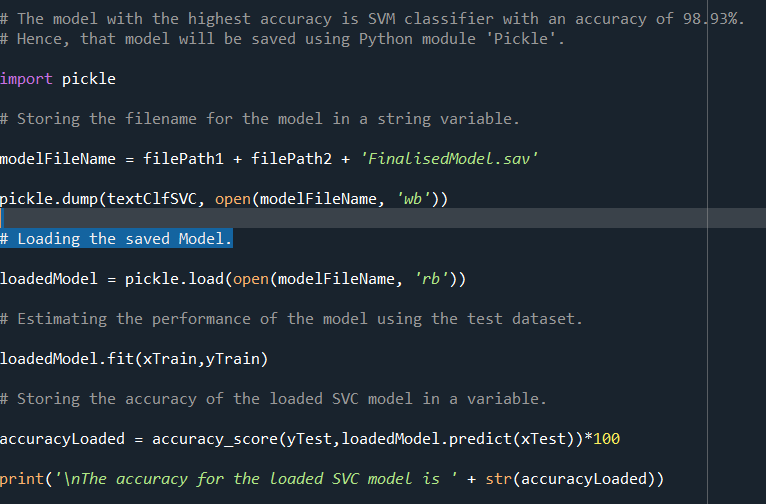
The accuracy for k = 10 cross fold validation on SVC classifier model is 98.929.

The standard deviation for k = 10 cross fold validation on SVC Model is 0.00516.

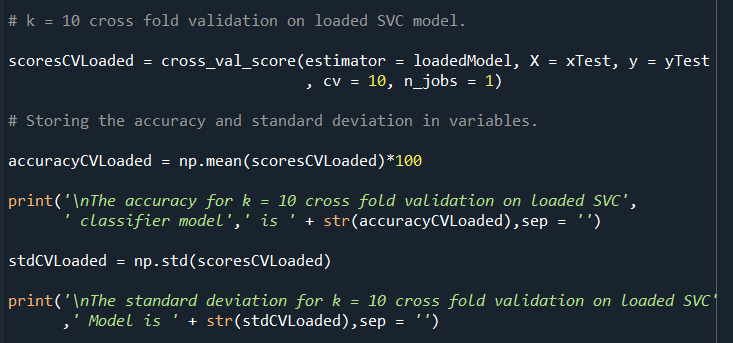


The model with the highest accuracy is SVM classifier with an accuracy of 98.93%.

So far, the model building was done on the training dataset. Now, the testing dataset will be used for assessing the model.



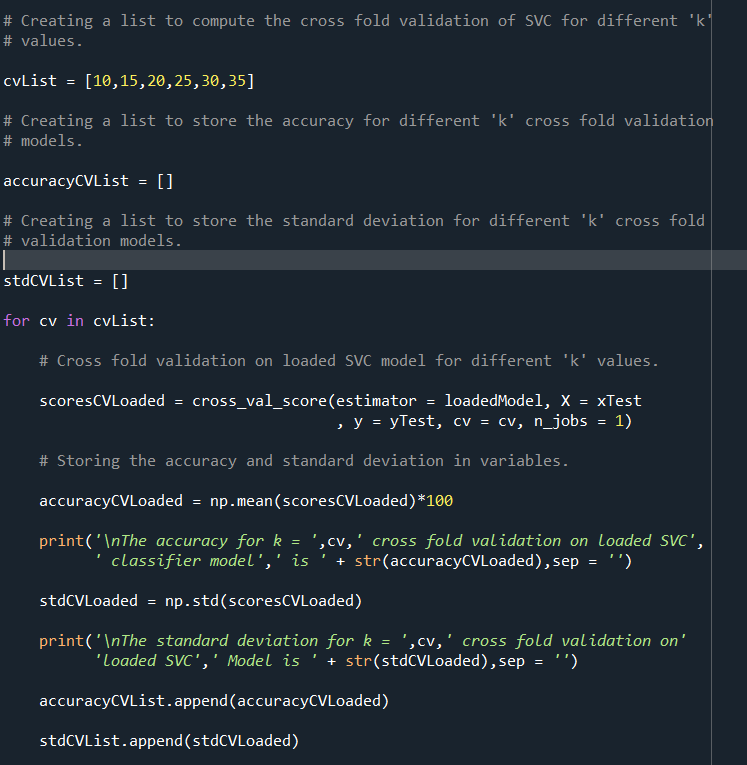
The accuracy for the loaded SVC model is 99.14.



The accuracy for k = 10 cross fold validation on loaded SVC classifier model is 97.635.

The standard deviation for k = 10 cross fold validation on loaded SVC Model is 0.0114.

Now, we will be computing the cross-fold validation of SVC for different 'k' values 10, 15, 20, 25, 30 and 35.



The accuracy for k = 10 cross fold validation on loaded SVC classifier model is 97.635.

The standard deviation for k = 10 cross fold validation onloaded SVC Model is 0.0114.

The accuracy for k = 15 cross fold validation on loaded SVC classifier model is 97.766.

The standard deviation for k = 15 cross fold validation onloaded SVC Model is 0.01046.

The accuracy for k = 20 cross fold validation on loaded SVC classifier model is 97.834.

The standard deviation for k = 20 cross fold validation onloaded SVC Model is 0.0172.

The accuracy for k = 25 cross fold validation on loaded SVC classifier model is 97.7650.

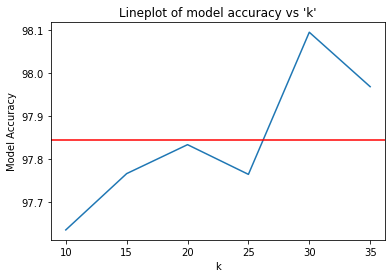
The standard deviation for k = 25 cross fold validation onloaded SVC Model is 0.02018.

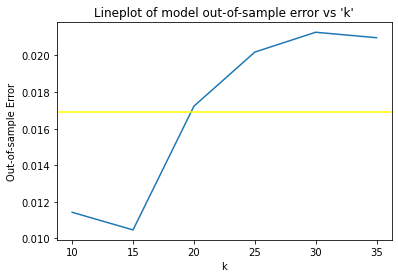
The accuracy for k = 30 cross fold validation on loaded SVC classifier model is 98.0954.

The standard deviation for k = 30 cross fold validation onloaded SVC Model is 0.0212.

The accuracy for k = 35 cross fold validation on loaded SVC classifier model is 97.9688.

The standard deviation for k = 35 cross fold validation onloaded SVC Model is 0.02097.





The best value for 'k' lies between 10 to 20 which can be observed from the plots. This way we can benchmark any model based on its accuracy and out-of-sample error.