**Virtual iNeuBytes Internship Project Documentation**

# **Name - Mohd Umair**

# **Registration Number - INBT0417**

**Domain - Artificial Intelligence**

# **Month - June**

# **Year - 2023**

**Submitted on : 7 July 2023**

# 

# **Github Link - https://github.com/Umair98392**

# **LinkedIn - https://www.linkedin.com/in/umair98392**

**Task 1**

# **Implement a Machine Learning Model with Supervised and unsupervised Models.**

## **Overview**

The Iris flower dataset was originally introduced by Ronald Fisher, a British statistician and biologist, in 1936. Fisher's paper, titled "The use of multiple measurements in taxonomic problems," presented this multivariate dataset. It is also known as Anderson's Iris dataset, named after Edgar Anderson, who collected the data to quantify the morphologic variation among three related species of Iris flowers.

The Iris flower dataset is a popular and widely used dataset in machine learning. It consists of measurements of four features of three different species of Iris flowers: sepal length, sepal width, petal length, and petal width. The goal of this project is to analyse and classify the Iris flowers based on these features using various machine learning algorithms.

The dataset comprises 50 samples from each of the three Iris species: Iris Setosa, Iris virginica, and Iris versicolor. Each sample includes measurements of four features: sepal length, sepal width, petal length, and petal width, all measured in centimetres.

Since its inception, this dataset has become a widely adopted benchmark for statistical classification techniques in machine learning, including support vector machines.

## **Objectives**

The aim is to classify iris flowers among three species :

Setosa,

Versicolor,

or Virginica

from sepals' and petals' length and width measurements.

The iris data set contains fifty instances of each of the three species.

The central goal is to design a model that makes proper classifications for new flowers.

## **Datasets**

Load the Iris dataset from Scikit Learn library module name datasets. The Iris Flower dataset contains 150 samples, with 50 samples for each of the three species: setosa, versicolor, and virginica. Each sample has the following four attributes:

* Sepal Length (in centimetres)
* Sepal Width (in centimetres)
* Petal Length (in centimetres)
* Petal Width (in centimetres)

## **Project Steps**

The main steps of this project include:

* Data Exploration: Analyse the dataset to gain insights into the distribution and characteristics of the features.
* Data Preprocessing: Handle missing values,scaling features, outliers, and perform any necessary data transformations.
* Feature Selection: Identify the most relevant features that contribute to the classification task.
* Model Building: Train and evaluate various machine learning models on the dataset.
* Model Selection: Choose the best-performing model based on evaluation metrics.
* Model Deployment: Deploy the selected model to make predictions on new, unseen data.

## **Tools and Libraries**

The task will be implemented using the following tools and libraries:

* Programming Language: Python
* Data Analysis: Pandas, NumPy
* Data Visualization: Matplotlib, Seaborn
* Machine Learning: Scikit-learn
* Data preprocessing: Standar scaler

## **Development Environment:**

Jupyter Notebook or any Python IDE.

## **Workflow**

The task workflow will involve the following steps:

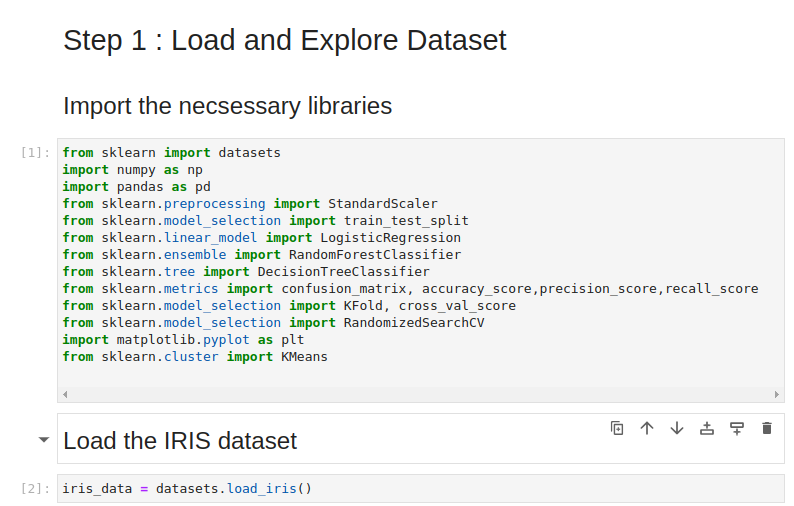
* Load the Iris Flower dataset into a pandas DataFrame.
* Perform exploratory data analysis (EDA) to understand the dataset's structure and characteristics.
* Preprocess the data by handling missing values, outliers, and performing any necessary transformations.
* Visualise the data using appropriate plots to gain further insights.
* Split the dataset into training and testing sets.
* Implement and evaluate multiple classification algorithms, such as logistic regression, decision trees, random forests, etc.
* Select the best-performing model based on evaluation metrics (e.g., accuracy, precision, recall, F1-score).
* Deploy the selected model and create a prediction function for new, unseen data.
* Document the findings, including the model's performance and insights gained from the analysis.

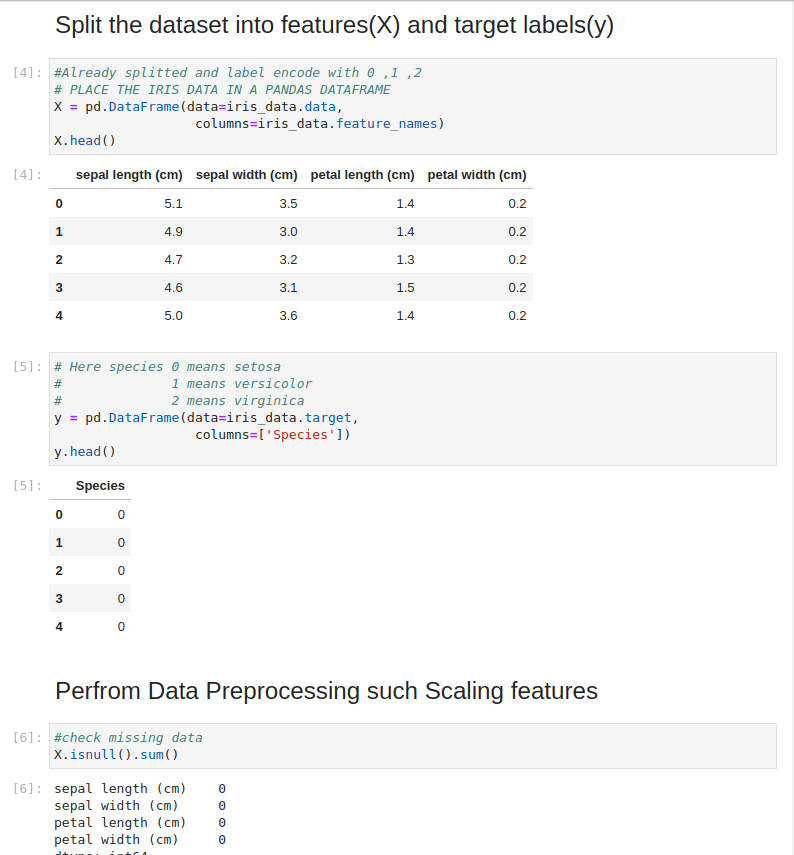
## **Expected Output**

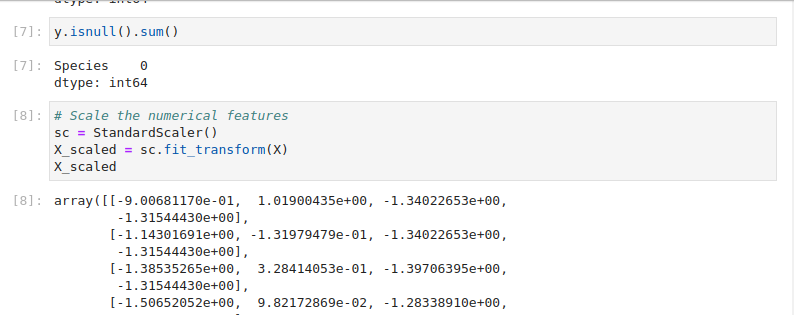
The expected output of this task will include the following:

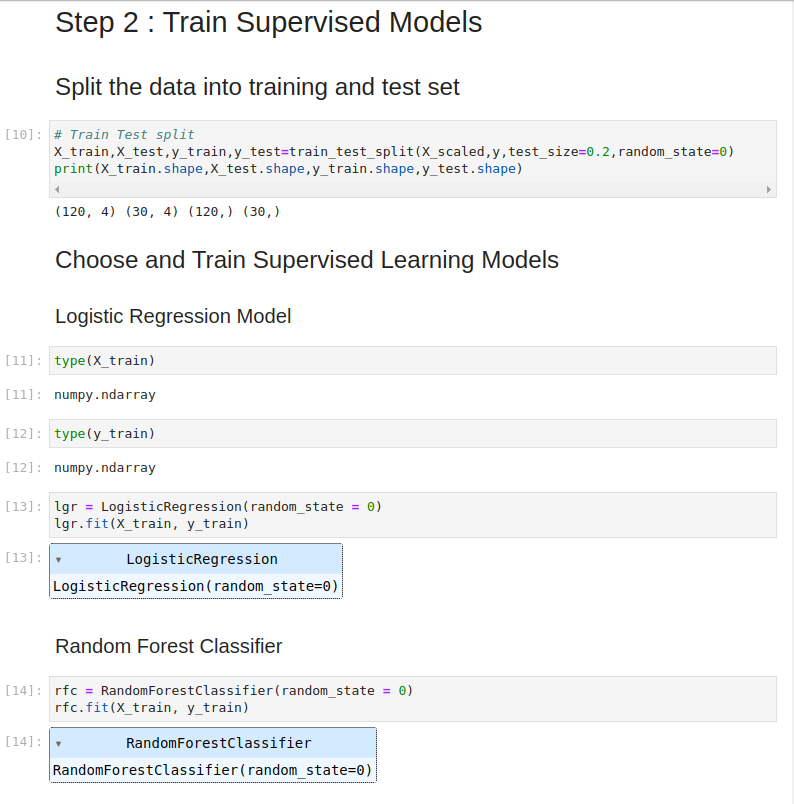
* Data exploration and visualisation results.
* Preprocessed dataset ready for machine learning algorithms.
* Evaluation metrics and performance comparison of different models.
* Selected model for deployment and associated code.
* Documentation summarising the project's process, findings, and conclusions.

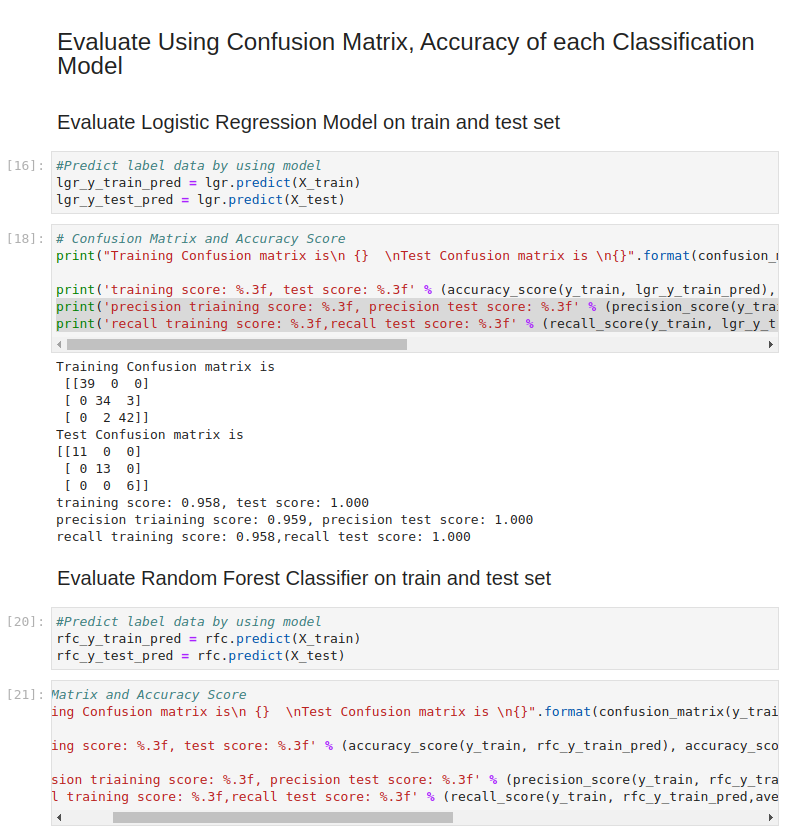
## **Code and Output**

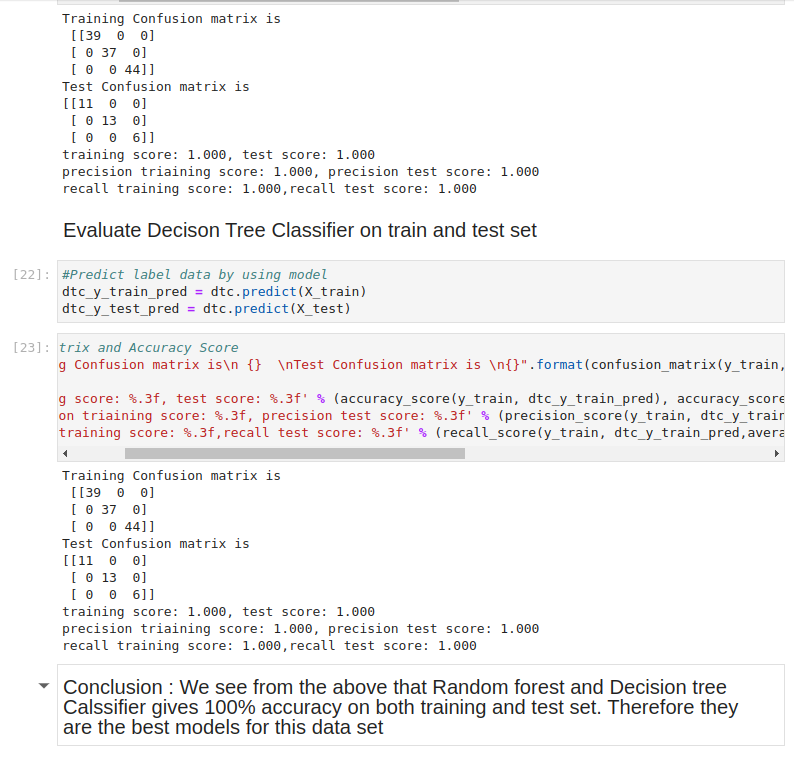


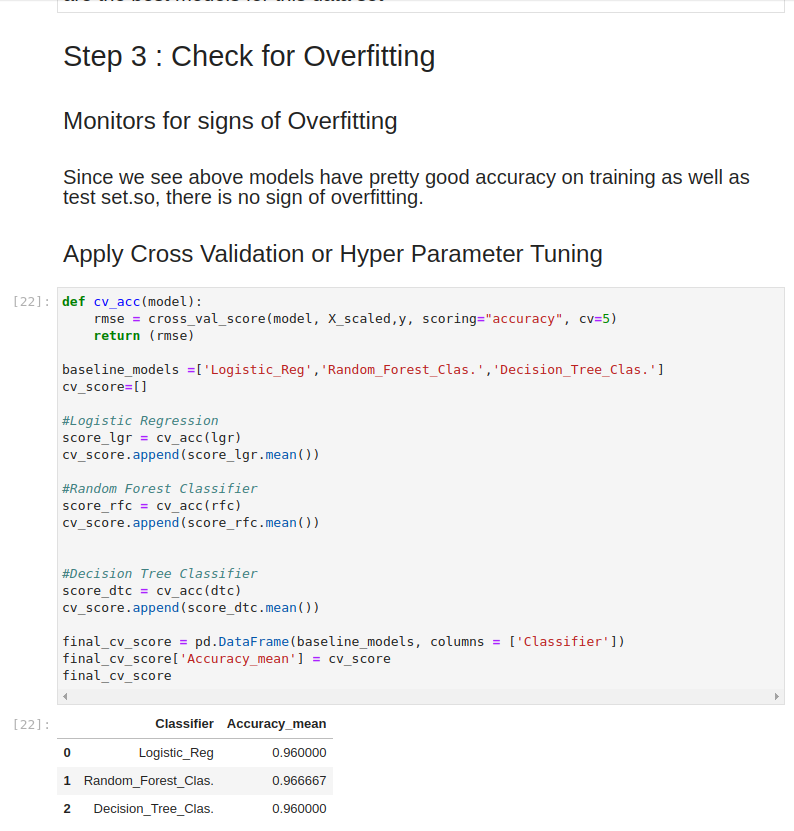


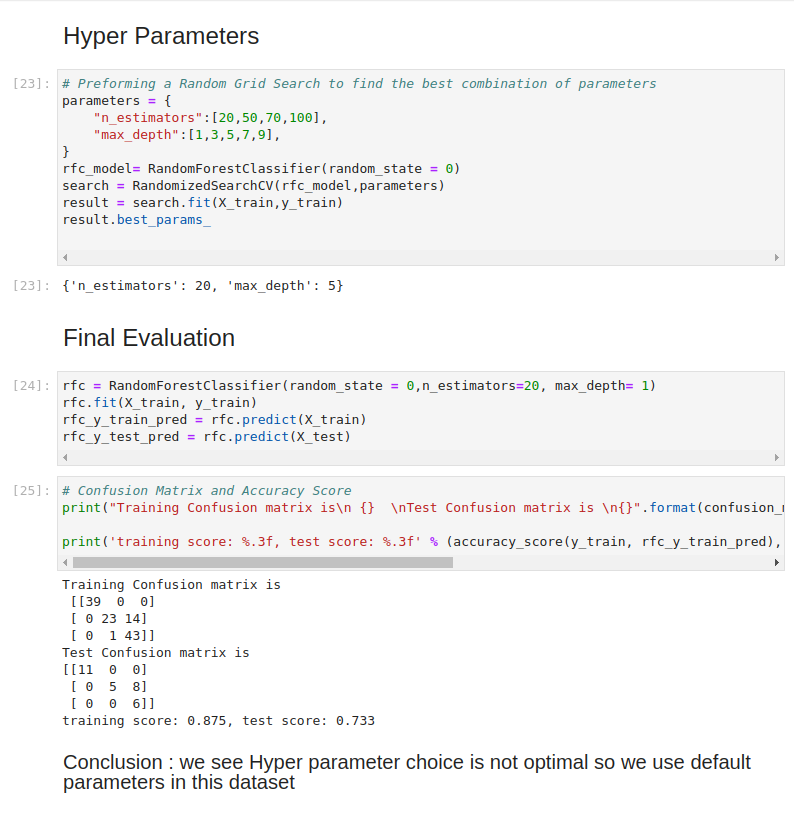


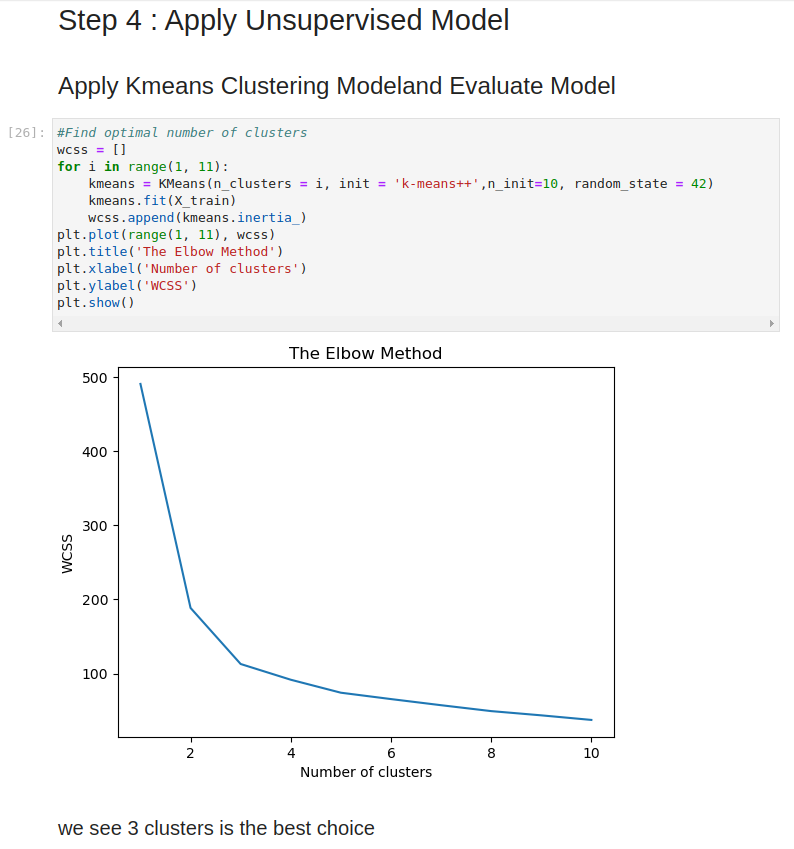


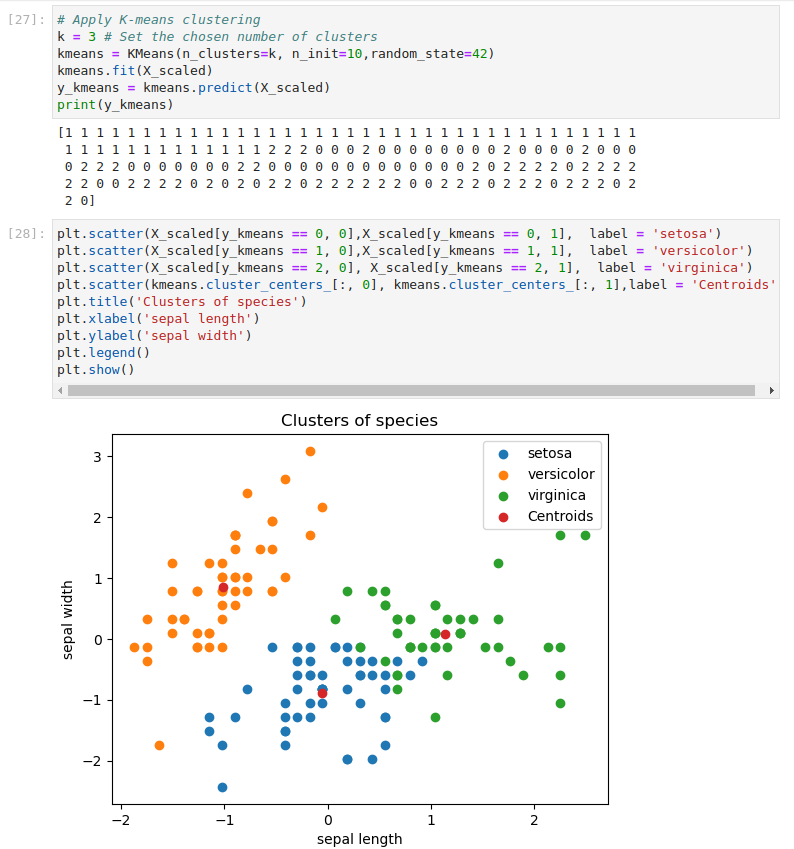












## **Questions Related to Task 1**

**Which python libraries did you find most useful in loading and exploring dataset ?**

There are several libraries that are most useful in loading and exploring dataset are :

* Pandas
* Numpy
* Scikit Learn
* Matplotlib
* Seaborn

**What preprocessing steps did you find necessary to apply to the IRIS dataset ?**

The preprocessing steps i find necessary to apply to the IRIS dataset are :

* Handling Missing values
* Features Scaling
* Encoding Categorical Variables
* Train Test Split

**What metrics were used to evaluate the supervised models and why ?**

* **Accuracy:** Accuracy is a commonly used metric that measures the overall correctness of the model's predictions. It is calculated as the ratio of the number of correctly predicted instances to the total number of instances.
* **Precision:** Precision measures the proportion of correctly predicted positive instances (true positives) out of all instances predicted as positive (true positives + false positives). Precision is useful when the cost of false positives is high, and it focuses on the model's ability to avoid false positives.
* **Recall:** Recall calculates the proportion of correctly predicted positive instances (true positives) out of all actual positive instances (true positives + false negatives).
* **Confusion Matrix:** It gives the counts of true positives, true negatives, false positives, and false negatives.

**How did you detect overfitting in the model and what strategies did you use to mitigate it?**

If the model performs better on the training set as compared to the testing set, it may indicate overfitting.

You can also detect outliers by plotting the graph .

* **Cross-validation:** Cross-validation is a more useful technique to detect overfitting. Instead of a single train-test split, it involves dividing the dataset into multiple folds and performing multiple training and evaluation cycles. This allows for a more comprehensive assessment of the model's performance. If the model consistently performs significantly better on the training folds compared to the validation folds, it suggests overfitting.
* **Learning Curves:** Learning curves visualise the model's performance on the training and validation sets as the amount of training data increases. By plotting the training and validation scores against the number of training samples, you can identify if the model is overfitting. If the training score continues to improve while the validation score plateaus or starts to decline, it indicates overfitting. Adding more training data can help mitigate overfitting in such cases.
* **Early Stopping:** Early stopping is a technique used during the model training process. It involves monitoring the model's performance on a validation set and stopping the training when the validation loss starts to increase. This prevents the model from continuing to learn from the training data and potentially overfitting.

**How did you choose the number of clusters for K-means algorithm ?**

By using Elbow method I choose the number of clusters for K-means algorithm. The elbow method is a common approach to estimate the optimal number of clusters. It involves calculating the sum of squared distances (SSD) between each data point and its assigned cluster centroid for different values of k (number of clusters).

The SSD is then plotted against the number of clusters, and the plot resembles an elbow shape. The "elbow" point, where the SSD starts to level off, indicates a suitable number of clusters.

**What insights were gained from the application of unsupervised learning on IRIS dataset ?**

* **Cluster Analysis:** In Unsupervised learning algorithms like K-means clustering can identify natural clusters or groups within the Iris dataset based on the feature similarities. The algorithm can group similar flowers together, allowing for the discovery of inherent clusters in the data. This can provide insights into the different species of Iris flowers present in the dataset and their distinct characteristics.
* **Data Visualization:** Unsupervised learning methods often involve data visualization techniques that help in understanding the dataset better. Techniques like scatter plots, 2D projections taht can provide visual representations of the Iris dataset, revealing any inherent patterns, clusters, or separations between the different species of flowers.
* **Outlier Detection:** Unsupervised learning algorithms can also identify potential outliers or anomalies in the Iris dataset. Outliers can be data points that deviate significantly from the typical patterns or clusters observed in the data. Detecting and analysing outliers can provide insights into potential data quality issues or rare instances that differ from the majority.

By applying unsupervised learning techniques to the Iris dataset, these insights can aid in understanding the underlying structure, characteristics, and relationships between the features and species of Iris flowers. Such insights can be valuable for further analysis, modelling, or decision-making processes related to the Iris dataset.

## **Conclusion**

The Iris Flower dataset project aims to provide a comprehensive analysis and classification of Iris flowers based on their features. By implementing various machine learning algorithms like Logistic Regression, Random Forest Classifier and Decision Tree Classifier and selecting the best model, this project will facilitate accurate predictions and provide valuable insights into the dataset.

**Task 2**

# **Utilize the “CIFAR-10” Dataset to Train a Convolutional neural Network (CNN) Model**

## **Overview**

CIFAR-10 is an established computer-vision dataset used for object recognition. It is a subset of the 80 million tiny images dataset and consists of 60,000 32x32 colour images containing one of 10 object classes, with 6000 images per class. It was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

Kaggle is hosting a CIFAR-10 leaderboard for the machine learning community to use for fun and practice. You can see how your approach compares to the latest research methods on Rodrigo Benenson's classification results page.

The CIFAR-10 dataset consists of 60,000 images, divided into 50,000 training images and 10,000 test images. Each image is a 32x32 colour image, with three channels (RGB). The dataset is labeled with 10 classes, including airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

## **Objectives**

The goal of this project is to build and train a CNN model that can accurately classify the images into their respective classes.

This documentation outlines the steps involved in the project, including data preprocessing, model architecture, training, and evaluation.

## **Datasets**

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The CIFAR-10 dataset contains the following 10 classes:

* Airplane
* Automobile
* Bird
* Cat
* Deer
* Dog
* Frog
* Horse
* Ship
* Truck

The dataset is divided into two subsets: a training set and a test set. The training set consists of 50,000 images, while the test set contains 10,000 images.

## **Data Preprocessing**

Before training the CNN model, the CIFAR-10 dataset undergoes the following preprocessing steps:

* **Normalization:** The pixel values of the images are normalized to a range of [0, 1] to ensure that the model can effectively learn from the data.
* **Label Encoding:** The class labels are label encoded, converting them into binary vectors. This step is necessary for training a multi-class classification model.
* **Data Split:** The training dataset is further split into training and validation sets. The training set is used for model training, while the validation set is used to monitor the model's performance during training and tune hyperparameters.

## **Tools and Libraries**

The task will be implemented using the following tools and libraries:

* Programming Language: Python
* Data Analysis: Pandas, NumPy
* Data Visualization: Matplotlib, Seaborn
* Machine Learning: Scikit-learn,Tensorflow,keras

## **Development Environment:**

Jupyter Notebook or any Python IDE.

## **Workflow**

## **Model Architecture**

The CNN model used in this project consists of the following layers:

* **Convolutional Layers:** The model starts with a series of convolutional layers to extract spatial features from the input images. Each convolutional layer is followed by a rectified linear unit (ReLU) activation function to introduce non-linearity.
* **Pooling Layers**: After each convolutional layer, a max pooling layer is added to reduce the spatial dimensions of the features and provide translational invariance.
* **Flattening:** The output of the last convolutional layer is flattened into a 1-dimensional vector, which serves as the input to the fully connected layers.
* **Fully Connected Layers:** The flattened features are passed through one or more fully connected layers, which learn to classify the input based on the extracted features. The last fully connected layer is followed by a softmax activation function to produce class probabilities.

# **Training Model**

The model is trained using the following parameters :

* **Loss Function:** The categorical cross-entropy loss function is used, which is suitable for multi-class classification problems.
* **Optimizer:** The model is optimized using stochastic gradient descent (SGD) with momentum. The learning rate and momentum hyperparameters are tuned to achieve better convergence.
* **Batch Size:** The training data is divided into mini-batches, and the model updates its weights after each batch. The batch size is an important hyperparameter that affects training speed and model generalization.
* **Epochs:** The model is trained for a fixed number of epochs, with each epoch iterating over the entire training dataset. The number of epochs is another hyperparameter that determines how many times the model sees the training data.

## **Evaluation of Model**

Once the model is trained, it is evaluated using the test dataset. The evaluation involves the following steps:

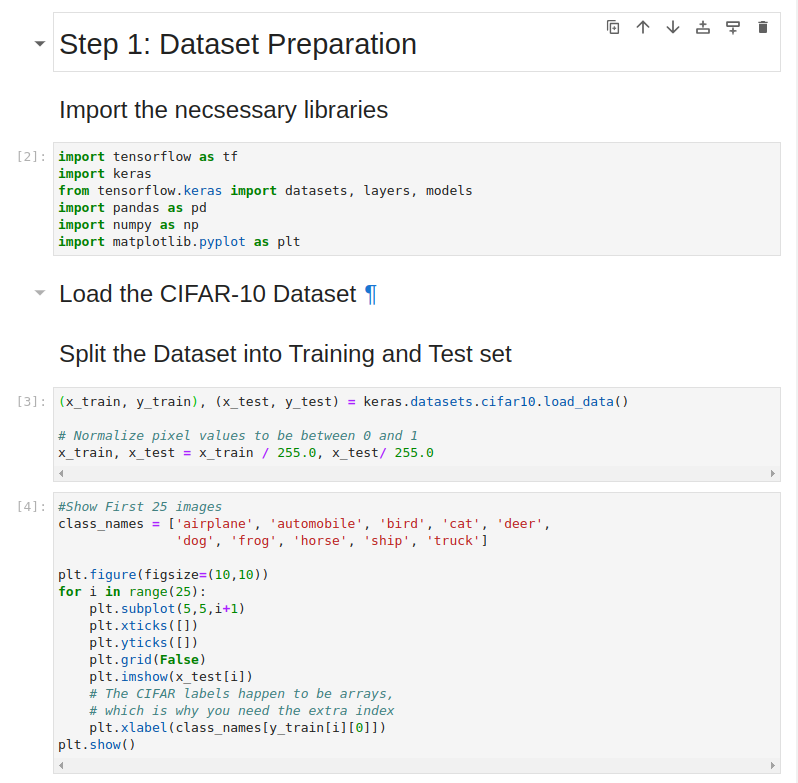
* **Prediction:** The model predicts the class labels for the test images.
* **Accuracy:** The accuracy of the model is calculated by comparing the predicted labels with the ground truth labels. It represents the percentage of correctly classified images.
* **Confusion Matrix:** A confusion matrix is created to visualize the model's performance across different classes. It provides insights into the types of errors made by the model.
* **Precision, Recall, and F1 Score:** Additional metrics such as precision, recall, and F1 score can be computed to evaluate the model's performance on individual classes.

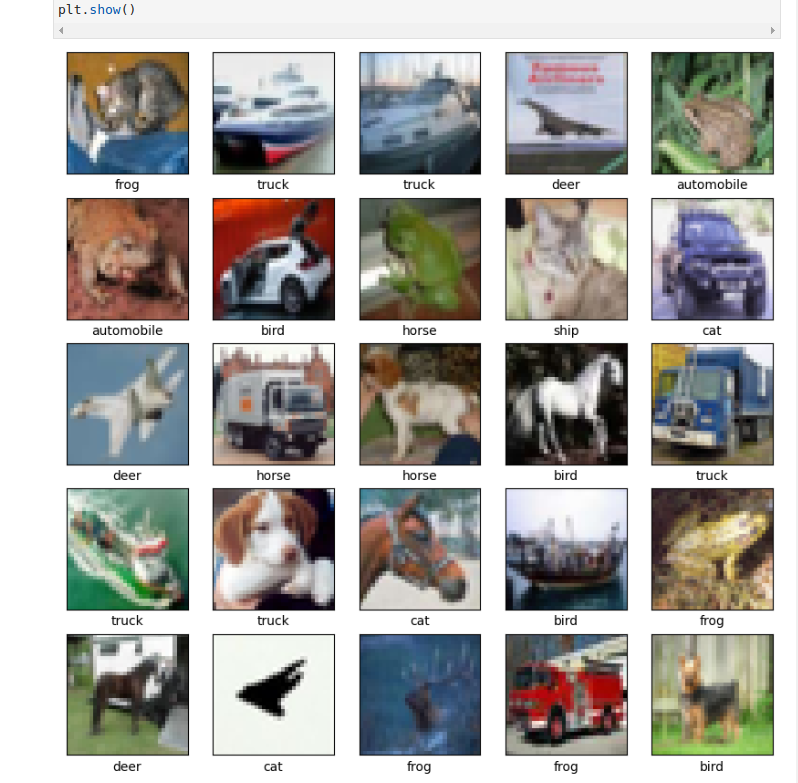
## **Expected Output**

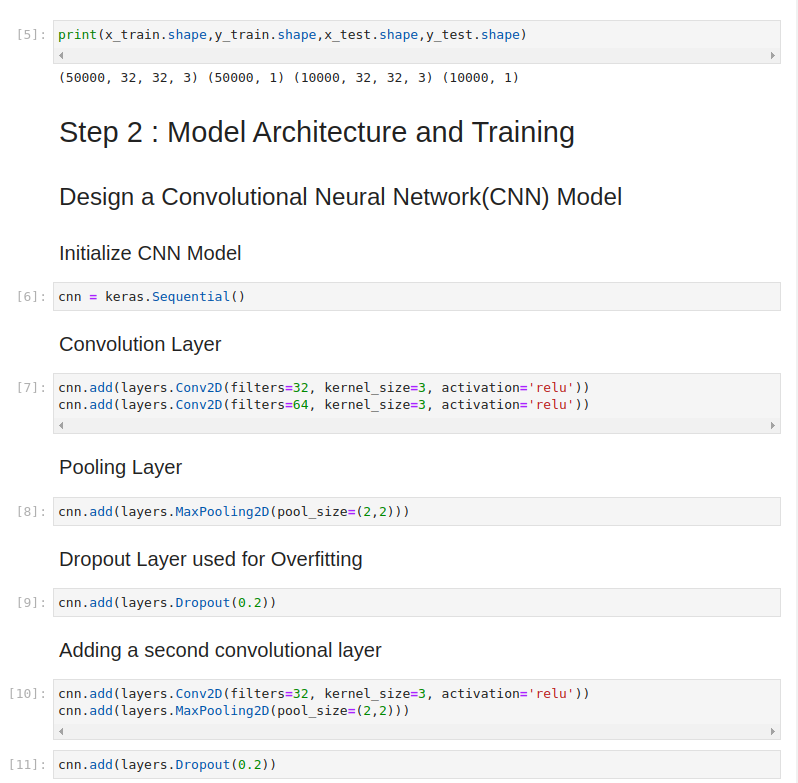
The expected output of this task will include the following:

* Data exploration and visualisation results.
* Preprocessed dataset ready for machine learning algorithms.
* Evaluation metrics and performance comparison of different models.
* Selected model for deployment and associated code.
* Documentation summarising the project's process, findings, and conclusions.

## **Code and Output**

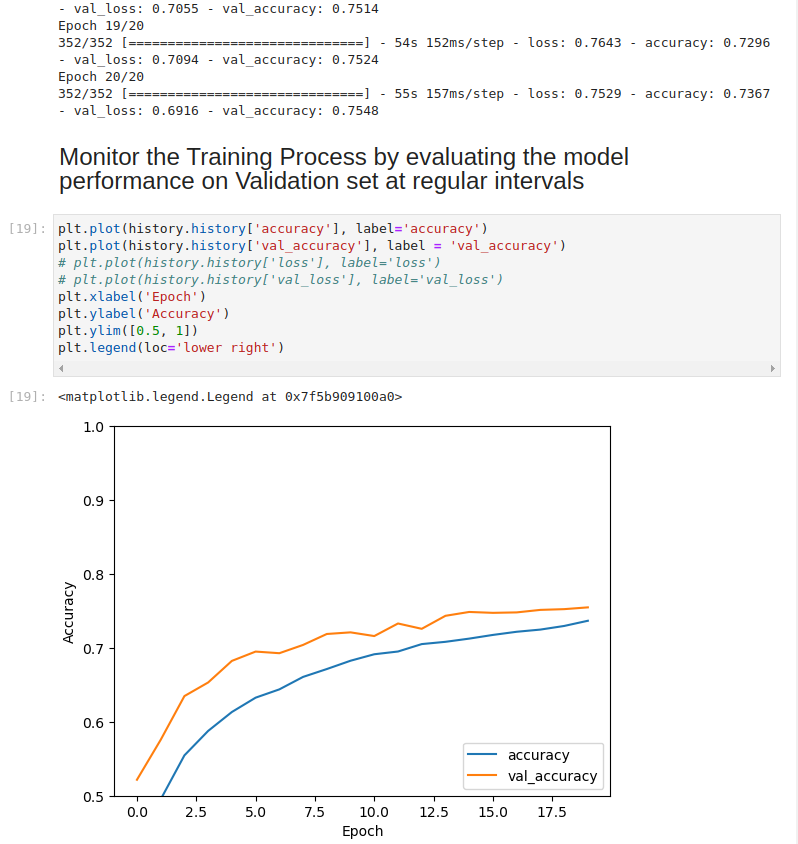


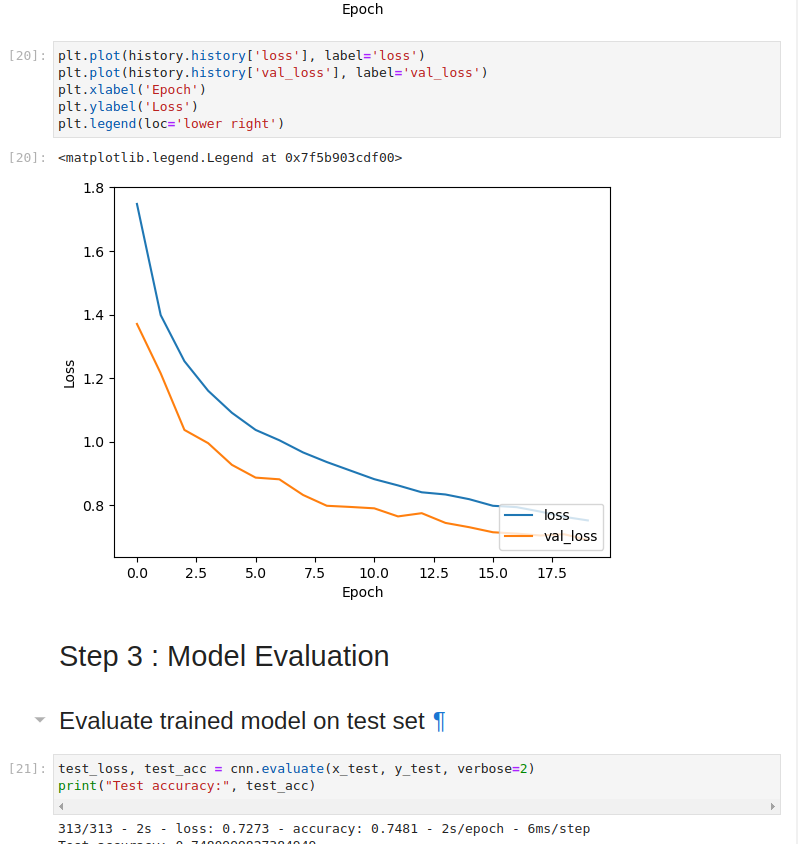


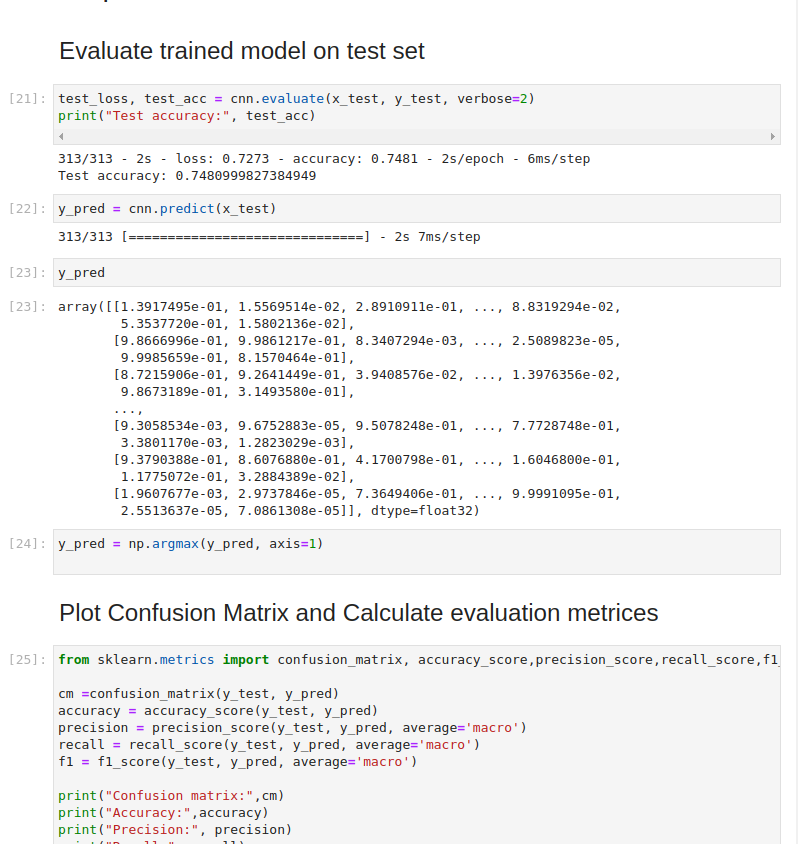


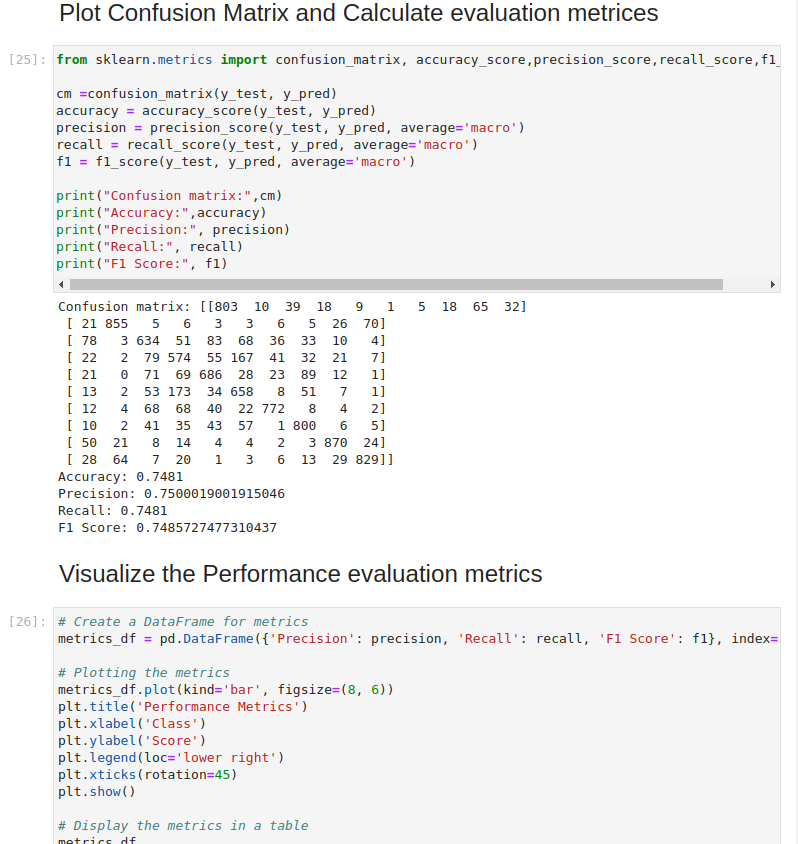


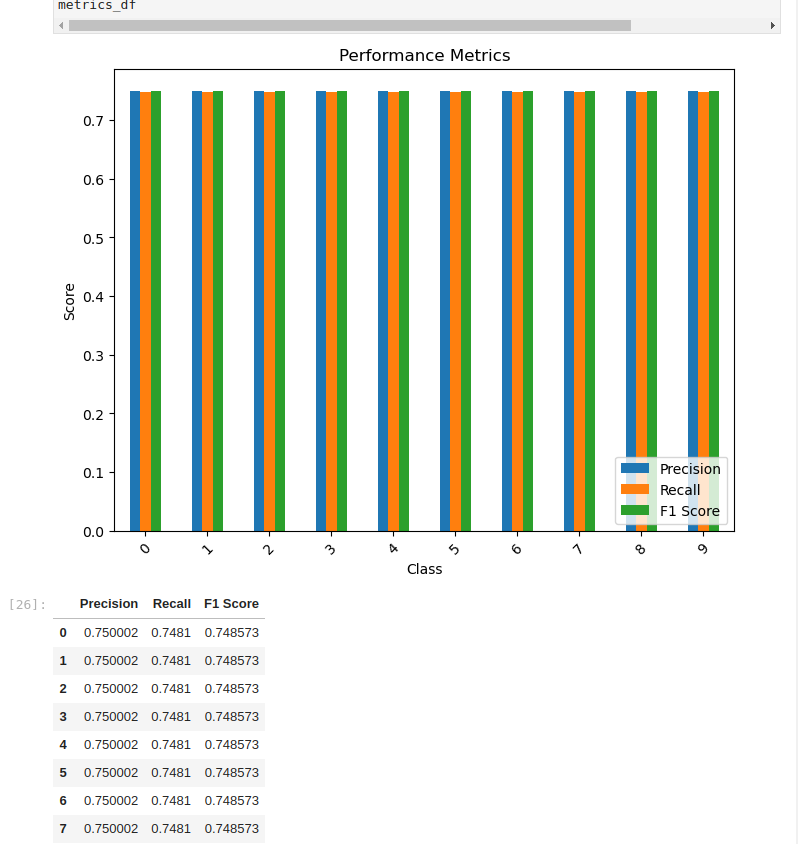


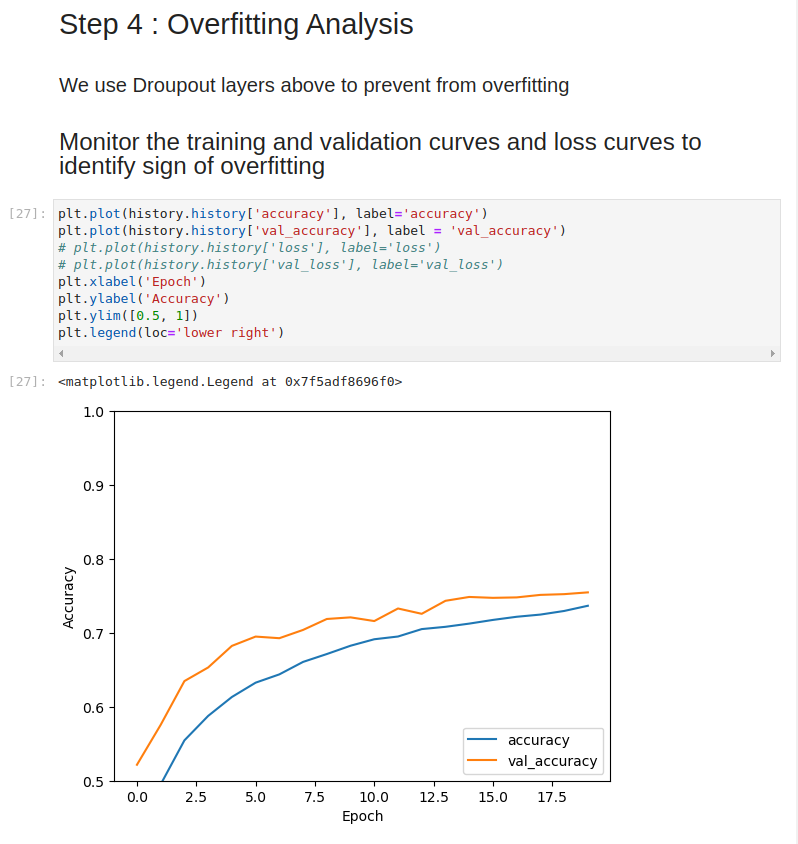


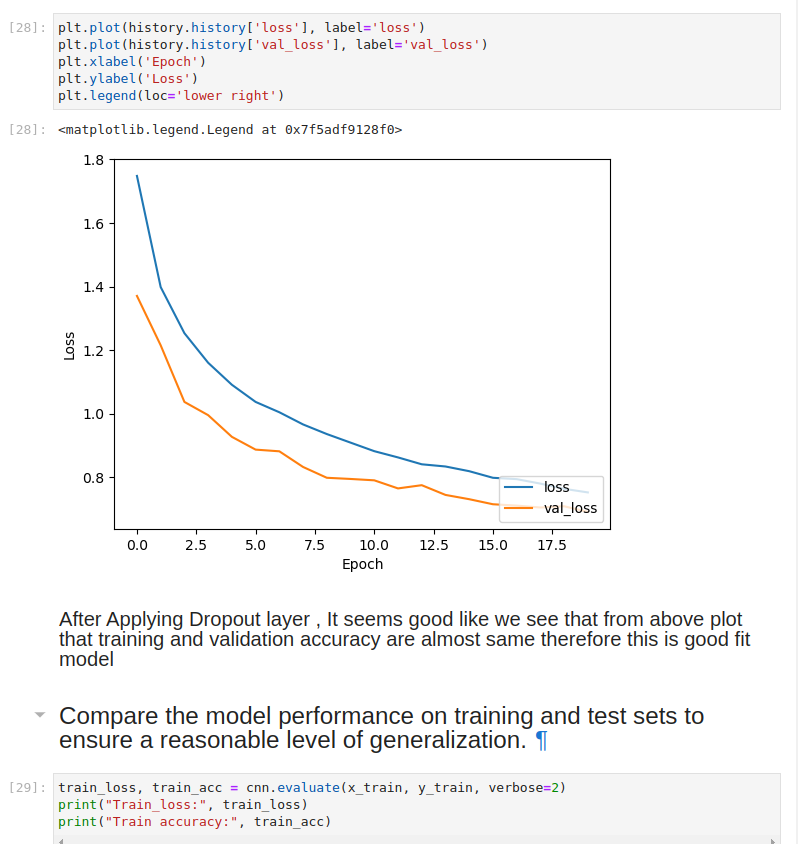


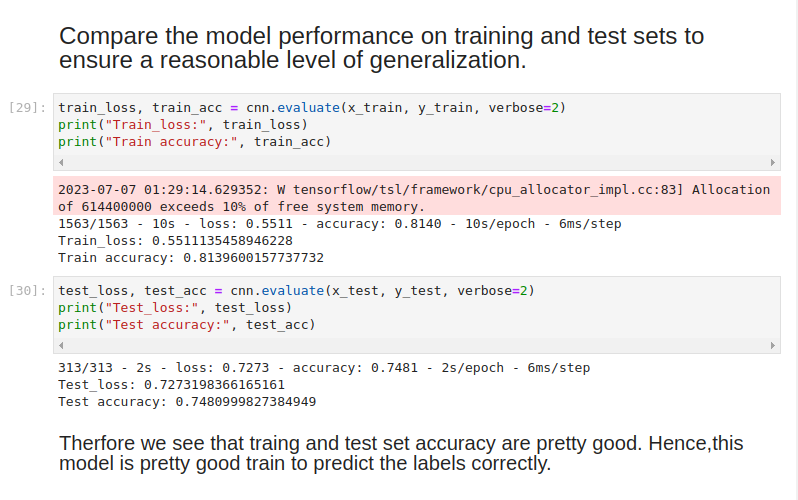












## **Questions Related to Task 2**

**What challenges did you encounter while preparing the CIFAR-10 dataset for the CNN Model ?**

* Challenges in installing Tensor flow and Cuda driver for faster processing.
* How to load already splitted train and test data
* Feature Normalization
* How to load an image from a dataset.
* Complexity in CNN architecture model
* Computational resources limited in my system.

**Describe the architecture of you CNN model. Why did you choose this specific architecture ?**

The architecture of the CNN model used for the CIFAR-10 dataset consists of the following layers:

* **Convolutional Layers:** The model starts with a stack of convolutional layers, each followed by a rectified linear unit (ReLU) activation function. The number of filters in each convolutional layer increases gradually, allowing the model to capture increasingly complex features in the images.
* **Pooling Layers:** After each convolutional layer, a max pooling layer is added to downsample the spatial dimensions of the feature maps. Max pooling helps in reducing the computational complexity and providing translational invariance.
* **Flattening Layer:** The output of the last convolutional layer is flattened into a 1-dimensional vector. This flattening step is necessary to connect the convolutional layers with the fully connected layers.
* **Fully Connected Layers:** The flattened features are passed through one or more fully connected layers. Each fully connected layer is followed by a ReLU activation function. The number of neurons in the fully connected layers decreases gradually, allowing the model to learn more abstract representations as it progresses.
* **Output Layer:** The last fully connected layer is connected to the output layer, which consists of 10 neurons (corresponding to the 10 classes in CIFAR-10). The output layer uses a softmax activation function to produce probabilities for each class.

This specific architecture is choose based on its effectiveness in image classification tasks, including the CIFAR-10 dataset. It strikes a balance between model complexity and computational efficiency. However, it's worth noting that the architecture choice can be subjective, and further experimentation with different architectures could yield even better results.

**What parameters did you tune during the model training and how didi they impact the model’s performance ?**

During the model training, several parameters were tuned to optimize the model's performance. The parameters that were commonly tuned include:

* **Learning Rate:** A lower learning rate can lead to slower convergence but potentially better accuracy. Tuning the learning rate helps in finding the balance between convergence speed and model performance.
* **Batch Size:** A larger batch size can result in faster training due to more efficient utilization of computational resources, but it may also lead to slower convergence or suboptimal solutions.Tuning the batch size helps in finding the right trade-off between training speed and model accuracy.
* **Number of Epochs:** The number of epochs defines the number of times the model iterates over the entire training dataset.
* **Model Architecture:** The architecture of the CNN model, including the number of layers, the number of filters, and the size of the layers, can also be tuned to optimize performance.

The impact of tuning these parameters on the model's performance can vary. For example:

* Adjusting the learning rate can influence the convergence speed and the quality of the final solution. An overly high learning rate may cause the loss function to fluctuate or diverge, while an extremely low learning rate may lead to slow convergence or getting stuck in local minima.
* Smaller batch sizes may introduce more noise in the weight updates, potentially leading to slower convergence. Conversely, larger batch sizes can result in less noise but might make the model more prone to getting stuck in sharp minima.
* Tuning the number of epochs can help find the optimal training duration. Too few epochs may result in an undertrained model, while too many epochs can lead to overfitting.
* Modifying the model architecture can have a significant impact on performance.

**How did you evaluate the performance of the CNN Model?**

Once the model is trained, it is evaluated using the test dataset. The evaluation involves the following steps:

* **Prediction:** The model predicts the class labels for the test images.
* **Accuracy:** The accuracy of the model is calculated by comparing the predicted labels with the ground truth labels. It represents the percentage of correctly classified images.
* **Confusion Matrix:** A confusion matrix is created to visualize the model's performance across different classes. It provides insights into the types of errors made by the model.
* **Precision, Recall, and F1 Score:** Additional metrics such as precision, recall, and F1 score can be computed to evaluate the model's performance on individual classes.

**Did you observe any signs of overfitting during training ? If so,how did you handle it ?**

* Yes, Firstly my model is overfitting on the training set.
* I use many dropout layers in architecture to handle or remove overfitting.

**How well did your model perform on the testing set compare to the training set ?**

* My CNN model have 80% accuracy on training set.
* And have 75% accuracy on testing set .
* It seems pretty good that training and testing set are near each other i.e. have only defer by 5%.

## 

## **Conclusion**

In this project, a CNN model was built and trained on the CIFAR-10 dataset for image classification. By following the outlined steps, including data preprocessing, model architecture, training, and evaluation, a model capable of accurately classifying images into the 10 different classes was developed. The model's performance can be further improved by fine-tuning hyperparameters, using data augmentation techniques, or exploring more advanced architectures.

**Major Project**

# **Data Cleaning, Tokenization, and Naive Bayes Classification on IMDb Dataset**

## **Overview**

This project focuses on classifying movie reviews from the IMDB dataset using the Naive Bayes classification algorithm, specifically the Multinomial Naive Bayes variant. The goal is to develop a model that can accurately predict whether a given movie review is positive or negative based on the text content.

The IMDB dataset is a popular benchmark dataset for sentiment analysis. It consists of 50,000 movie reviews, split evenly into training and testing sets. Each review is labeled as either "positive" or "negative". The dataset is balanced, with an equal number of positive and negative reviews.

## **Objectives**

The aim of the IMDB dataset is to provide a labelled dataset for sentiment analysis tasks, specifically for movie reviews. The dataset contains a collection of movie reviews along with their corresponding sentiment labels (positive or negative).

The goal of using the IMDB dataset is to develop and evaluate machine learning models and algorithms that can accurately classify the sentiment expressed in a given movie review. By training a model on this dataset, researchers and practitioners can build systems that automatically analyze and categorize movie reviews based on their sentiment.

## **Datasets**

The IMDB dataset, used for sentiment analysis in this project, is a publicly available dataset commonly used for benchmarking text classification models.

It consists of 50,000 movie reviews, with an equal number of positive and negative reviews. The dataset is divided into a training set and a testing set, each containing 25,000 reviews.

## 

## **Project Steps**

The general steps involved in the approach are as follows:

* **Preprocess the text data:** This includes steps like tokenization, removing stopwords, and stemming/lemmatization.
* **Feature extraction:** Convert the text data into numerical features that can be used by the Naive Bayes classifier. Common techniques include bag-of-words representation or TF-IDF vectorization.
* **Train the Naive Bayes classifier:** Fit the training data to the Multinomial Naive Bayes model and estimate the probabilities.
* **Predict the sentiment:** Apply the trained model to predict the sentiment (positive or negative) for new, unseen reviews.
* **Evaluate the model:** Measure the performance of the classifier using evaluation metrics such as accuracy, precision, recall, and F1-score.

## **Tools and Libraries**

The task will be implemented using the following tools and libraries:

* Programming Language: Python
* Data Analysis: Pandas, NumPy, Regular Expression(re),
* Data Visualization: Matplotlib, Seaborn
* Machine Learning: Scikit-learn
* Data preprocessing: Standar scaler

## **Development Environment:**

Jupyter Notebook or any Python IDE.

## 

## **Training Model**

The model used in this project is the Multinomial Naive Bayes classifier. Naive Bayes is a probabilistic classification algorithm that is based on Bayes' theorem and assumes independence between the features.

The Multinomial Naive Bayes variant is specifically designed for discrete features, making it suitable for text classification tasks where the features often represent word counts or frequencies. It is commonly used in sentiment analysis, spam filtering, and document categorization.

The Multinomial Naive Bayes classifier calculates the probabilities of each class (positive or negative) given a set of features (words or tokens) in a document. It uses the training data to estimate the likelihood probabilities of each feature occurring in each class and the prior probabilities of each class. Then, it applies Bayes' theorem to calculate the posterior probabilities and makes predictions based on the class with the highest probability.

## **Evaluation of Model**

Once the model is trained, it is evaluated using the test dataset. The evaluation involves the following steps:

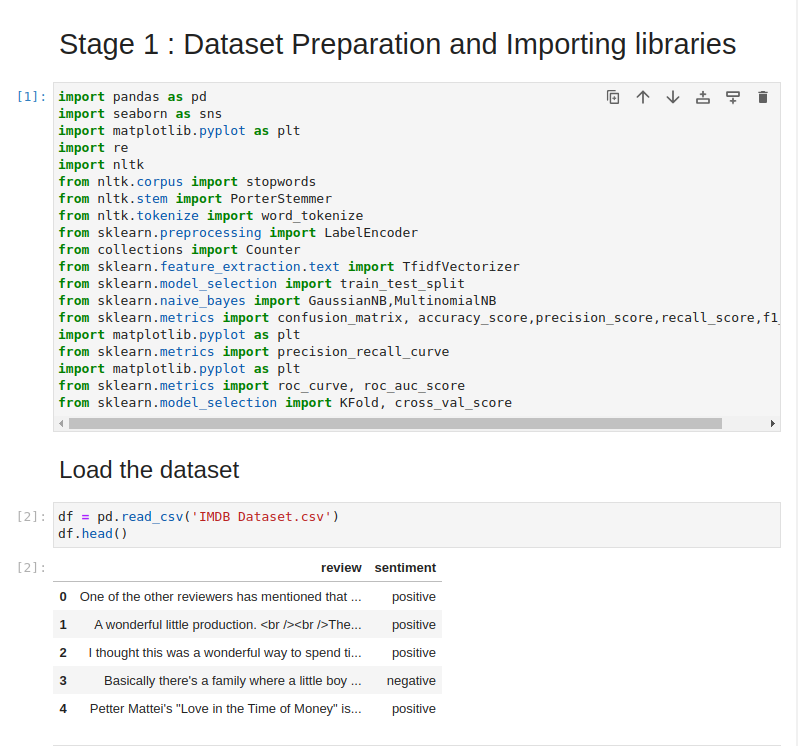
* **Prediction:** The model predicts the positive or negative review.
* **Accuracy:** The accuracy of the model is calculated by comparing the predicted sentiment labels with the ground truth labels. It represents the percentage of correctly classified review.
* **Confusion Matrix:** A confusion matrix is created to visualize the model's performance across different classes. It provides insights into the types of errors made by the model.
* **Precision, Recall, and F1 Score:** Additional metrics such as precision, recall, and F1 score can be computed to evaluate the model's performance on individual classes.

## **Expected Output**

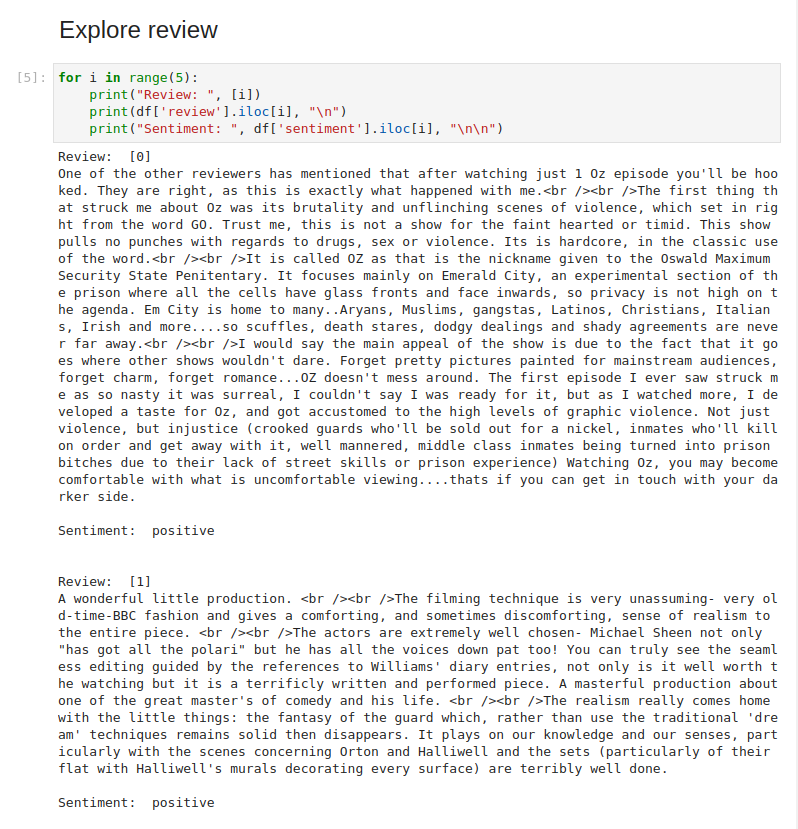
The expected output of this task will include the following:

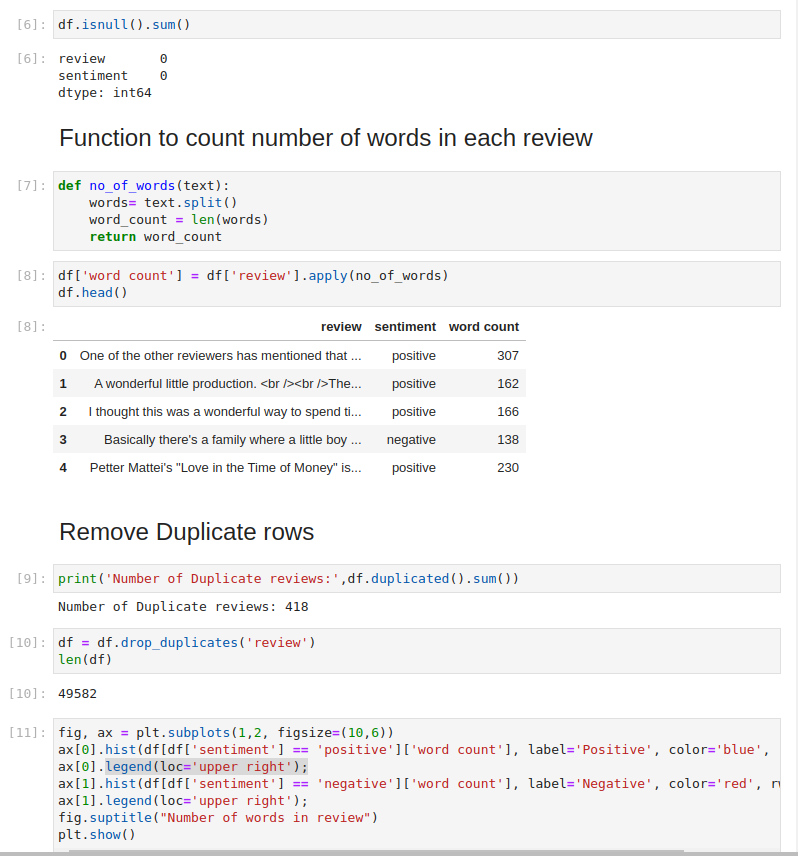
* Data exploration and visualisation results.
* Preprocessed dataset ready for machine learning algorithms.
* Evaluation metrics .
* Selected model for deployment and associated code.
* Predicting review using command line interface.

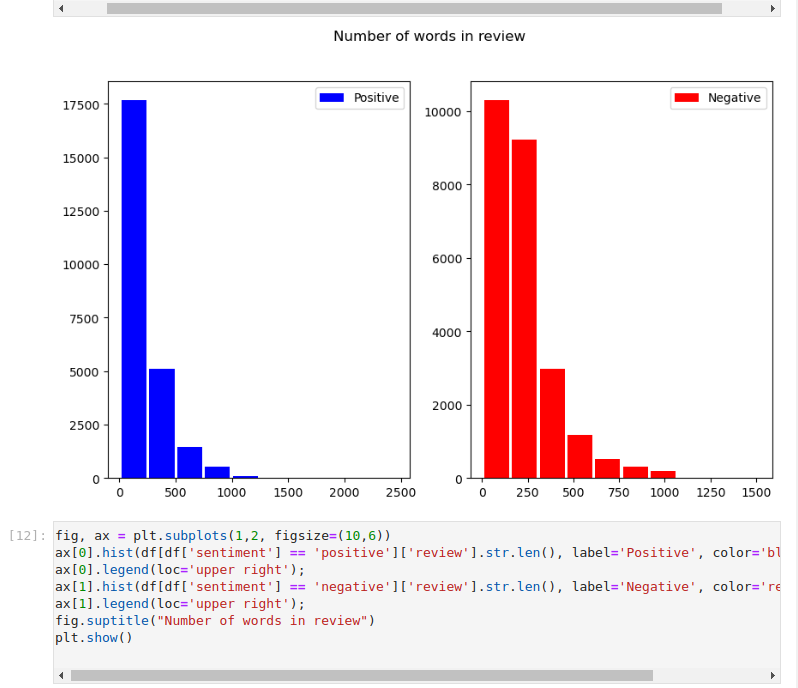
## **Code and Output**

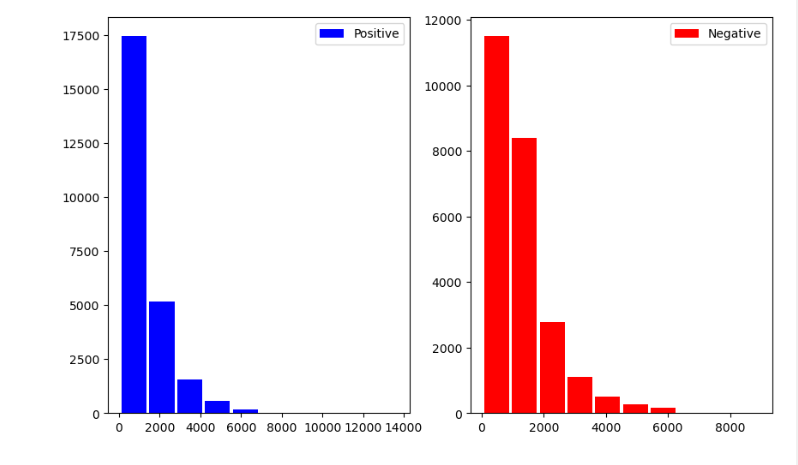


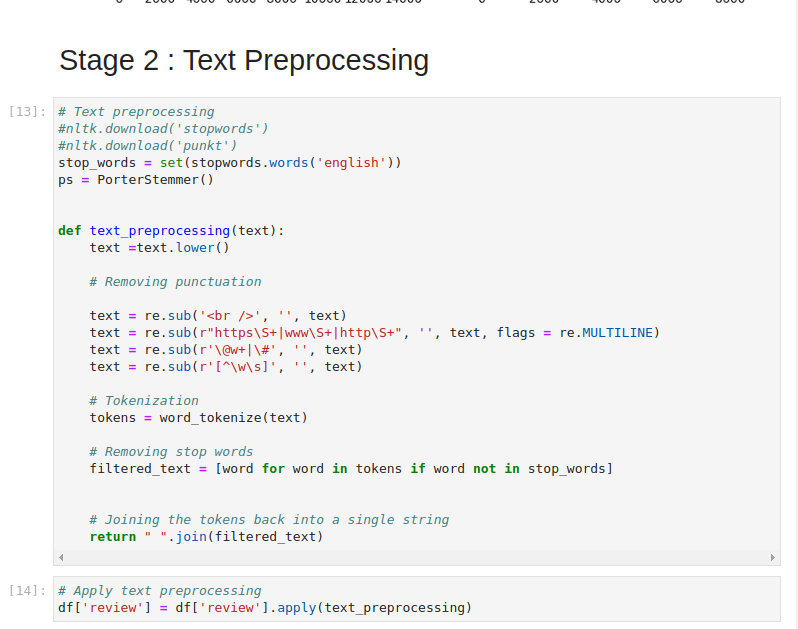


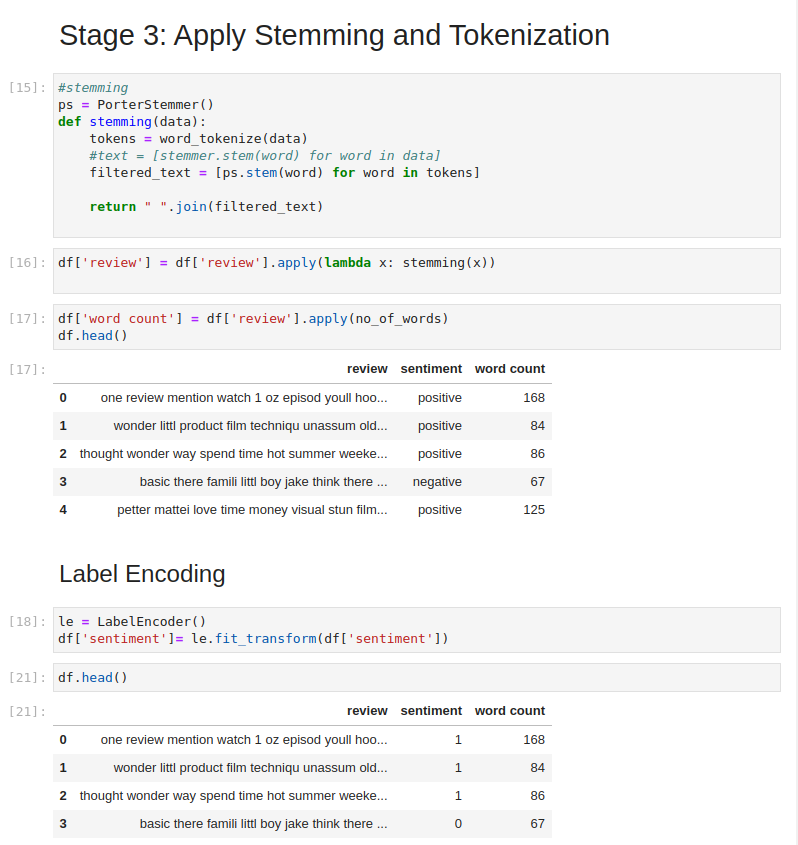


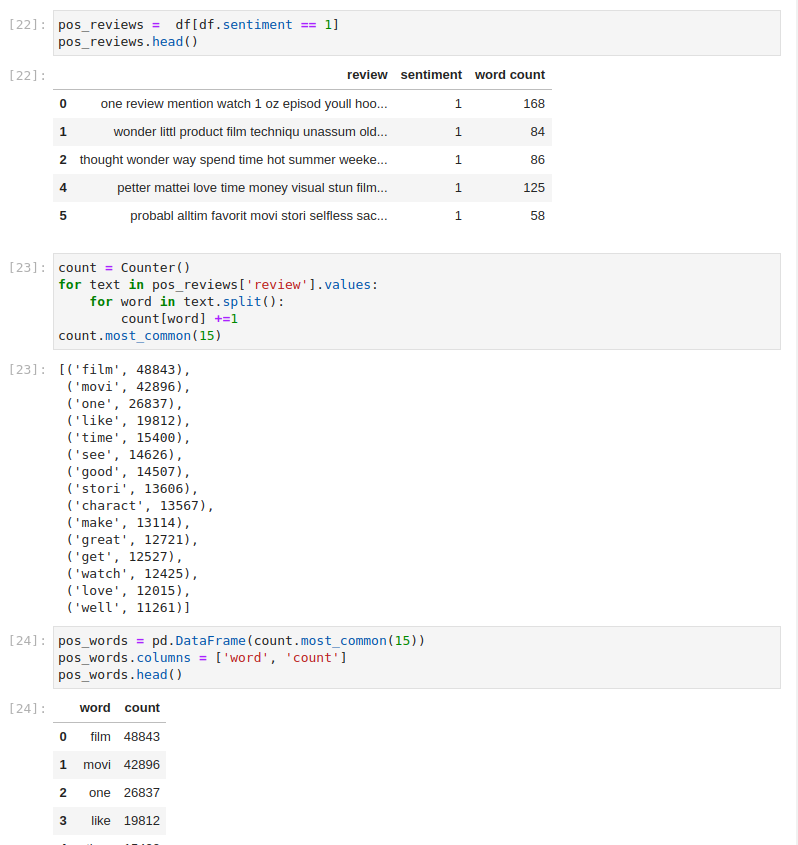


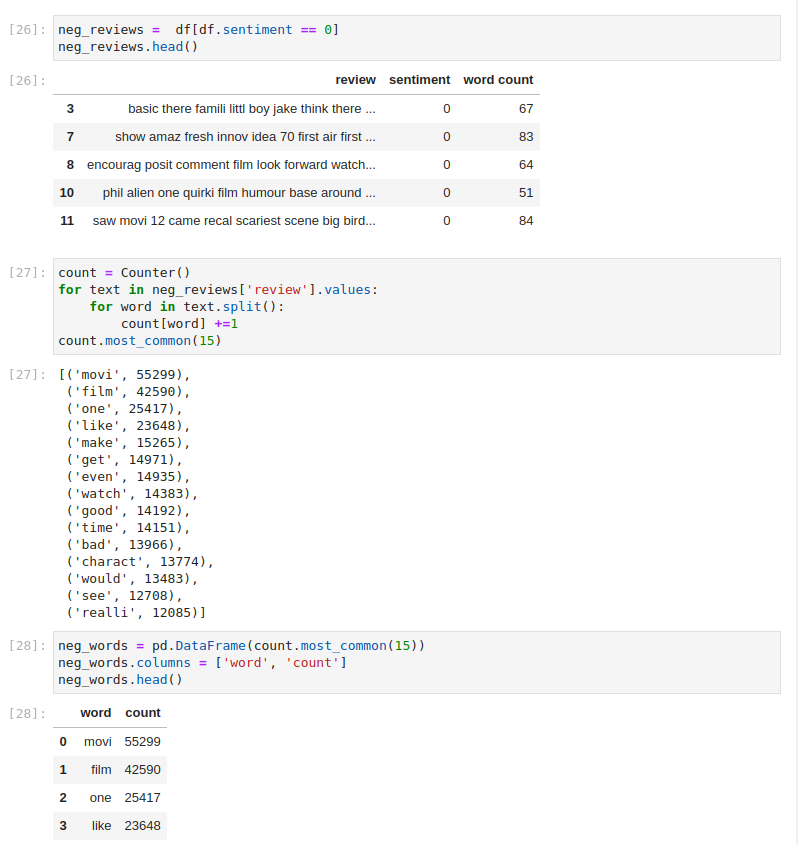




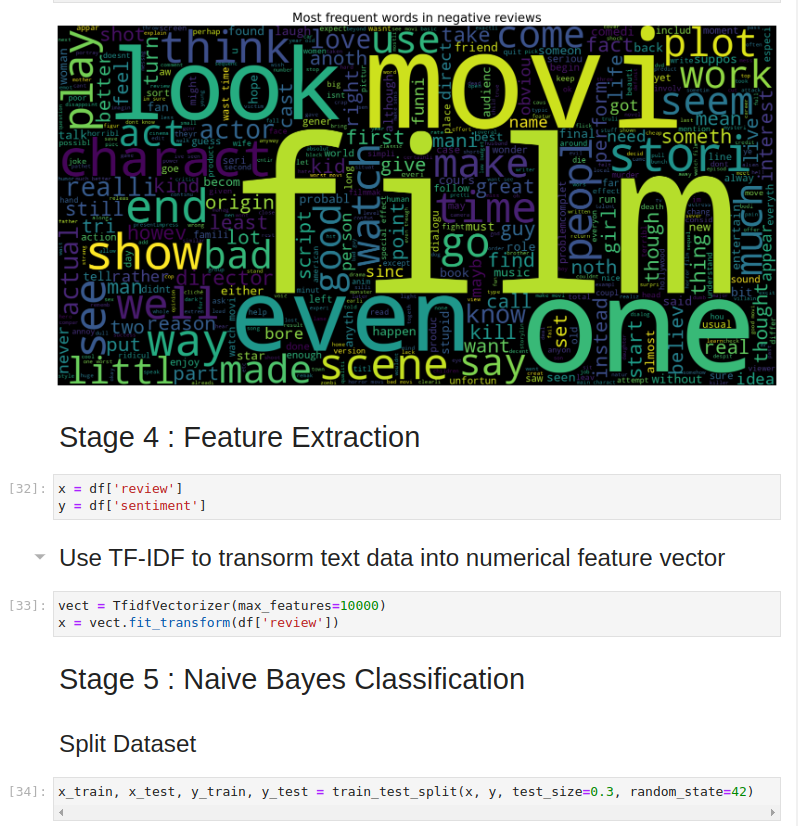


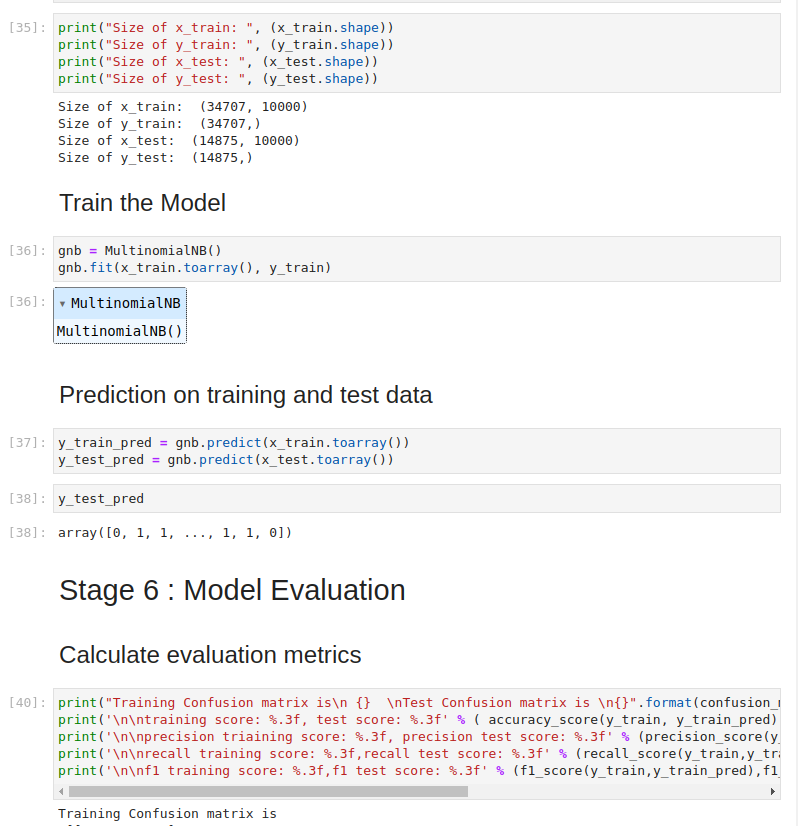


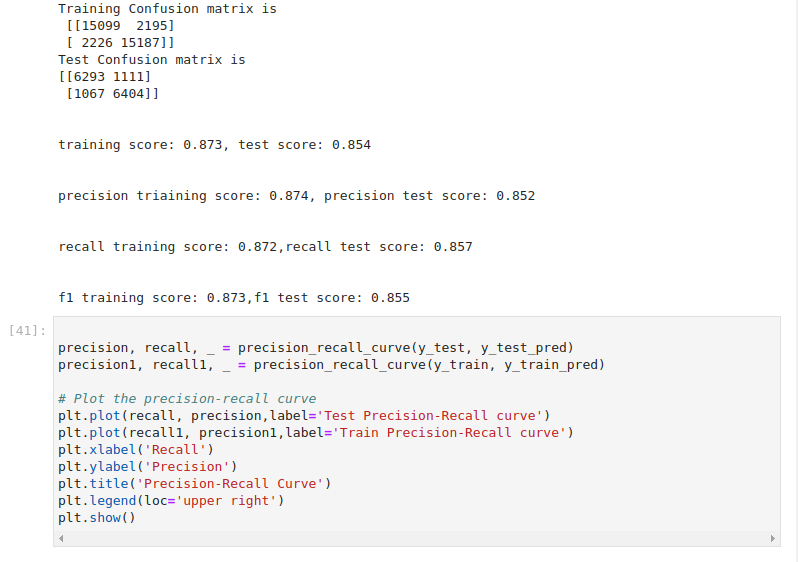


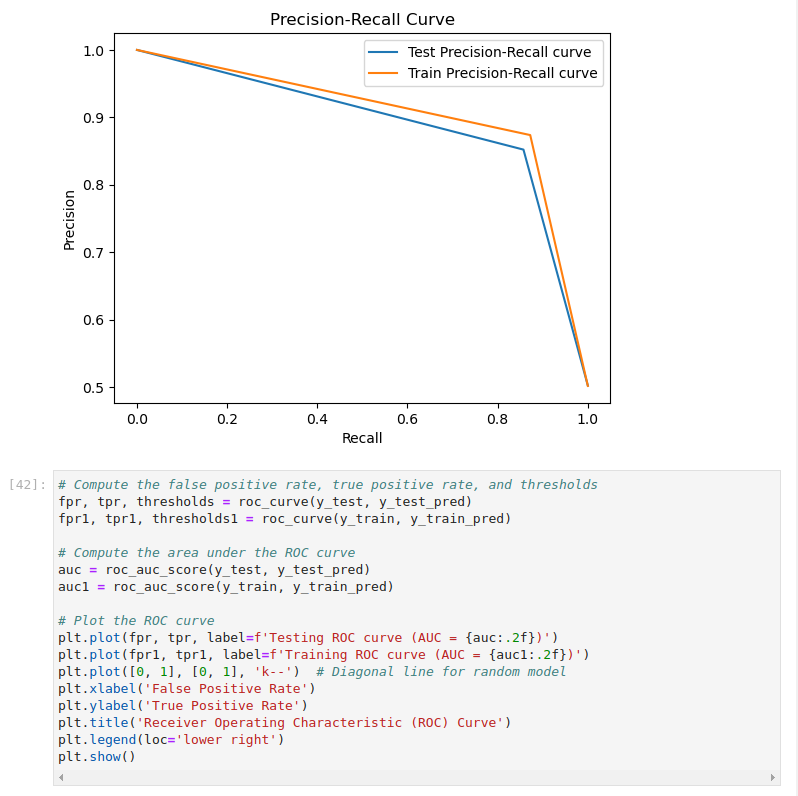


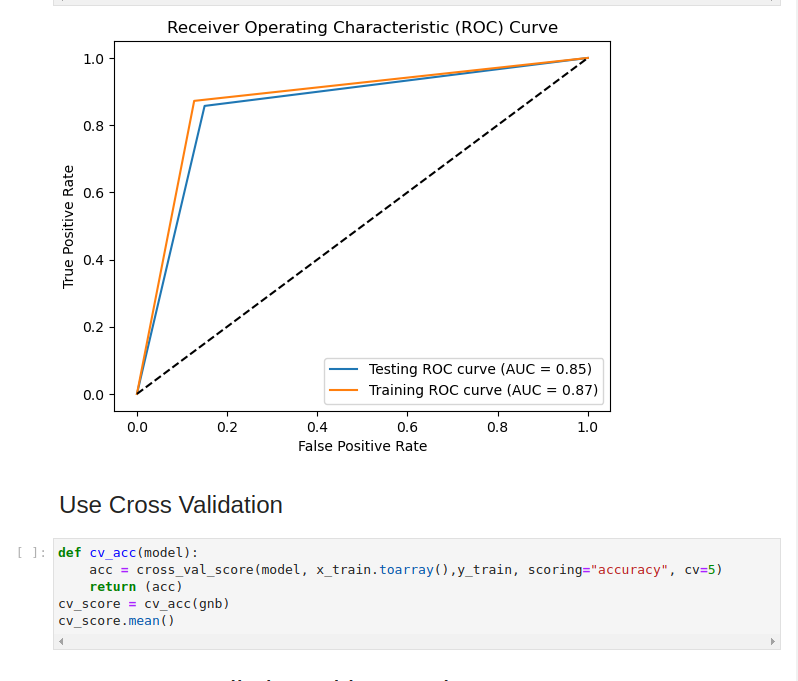


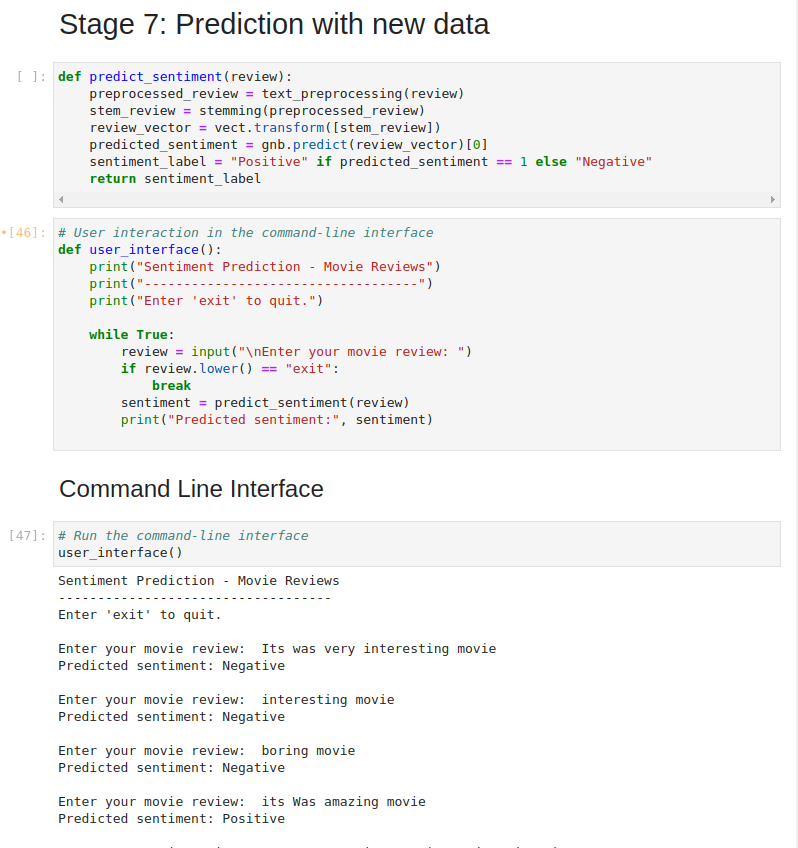












## **Conclusion**

In this project, we successfully applied the Multinomial Naive Bayes algorithm to classify movie reviews from the IMDB dataset as positive or negative. By employing appropriate data preprocessing and feature extraction techniques, we were able to train a model that achieved satisfactory performance in sentiment prediction. Naive Bayes is known for its simplicity and efficiency, making it a popular choice for text classification tasks.

## **Future Enhancements**

To further improve the classification accuracy and model performance, the following enhancements can be considered:

* **Advanced preprocessing techniques:** Explore advanced text preprocessing techniques such as word embeddings, n-grams, or word sense disambiguation.
* **Hyperparameter tuning:** Optimize the hyperparameters of the Naive Bayes classifier to achieve better results.
* **Ensemble methods:** Consider using ensemble methods, such as Random Forest or Gradient Boosting, to combine multiple classifiers for improved predictions.
* **Domain-specific features:** Incorporate domain-specific features, such as movie genre or director, to enhance the model's understanding of the data.