Project Plan:

Project Overview:

The project aims to perform sentiment analysis on IMDb movie reviews using a machine learning model. The primary objective is to predict the sentiment (positive/negative) based on the textual content of movie reviews.

1. Project Initiation:
   - I defined the project scope and objectives.
     Objectives:
     - Create more than one model to train and compile in order to predict the sentiment.
     - More than one model was created so that the models could be compared and then further fine tuned.
     - Study results and reports.

   - Gathered resources, including the IMDb movie review dataset.
   - Set up the development environment and load all packages required for the model.

2. Data Collection and Preprocessing:
   - Collected the IMDb movie review dataset.
     - Pre-processed data
     - including cleaning
     - HTML tag removal
     - lowercase conversion
     - punctuation removal
     - Tokenization
     - Label encoding(added another column to the dataset with the numerical values of the sentiments)

3. Feature Analysis and Selection:
   - Analysed the distribution of using histogram
     - review lengths.
     - Number of words
   - Explored dataset characteristics by printing out the head of the dataset to get an idea of columns and their regards and also printing out number of values.

4. Model Building and Training:
   - Implemented a Convolutional Neural Network (CNN), variation of MLP and RNN for sentiment analysis.
   - Trained and compiled the model using the pre-processed data.

5. Experiment Design:
   - Designed experiments to evaluate model performance.
   - Split data into training, validation, test sets and padded them.

6. Model Evaluation:
   - Evaluated the model on the test set.
   - Analysed model performance metrics, including accuracy.
     - CNN:`0.8548`
     - MLP:`0.8051`
     - RNN:`0.5004`

Machine Learning Algorithms:

- CNN:
  The embedding layer converts vocabulary that is integer encoded into dense vectors of a fixed size.
  A Convolution layer of 1D input data was created.
  A global max pooling operation for temporal data was created.
  Created ‘dense’ being a fully connected layer.
  To prevent overfitting a dropout layer was created as it is a regularisation technique. It does this by randomly setting a part of the input units to 0 after each update while the training of the model occurs.

  The model is designed for binary text classification tasks, where the goal is to predict the binary labels namely 1 and 0 based on the text data that is being input(test data from dataset.

- MLP(variation):
  This model implements a binary text classification model using a neural network with an Embedding layer, Conv1D layer, GlobalMaxPooling1D layer, and Dense layers.

  It starts with an `Embedding` layer, which is often used for text data to convert words into dense vectors.
  The `Flatten` layer flattens the output from the embedding layer.
  Two fully connected (`Dense`) layers follow with ReLU activation functions.
  Dropout layers are added to reduce overfitting.
  The final layer has a single neuron with a sigmoid activation function, suitable for binary classification problems.

  Model Compilation
  The model is compiled using the Adam optimizer and binary cross-entropy loss
  The accuracy metric is used for evaluation.

The model takes sequences of words, embeds them into lower-dimensional vectors, and then processes them through fully connected layers for classification. The use of dropout layers helps prevent overfitting.

- RNN:The code provided implements a machine learning model using the TensorFlow and Keras libraries for natural language processing (NLP) tasks. Here's a step-by-step explanation of the key components:

  The model is a sequential model defined using the Keras API.
  It starts with an embedding layer, which is responsible for converting integer-encoded words into dense vectors of fixed size (`embedding_dim`).
  Two Gated Recurrent Unit (GRU) layers are used for sequence processing. The first GRU layer returns sequences (`return_sequences=True`), and the second GRU layer processes those sequences and outputs a fixed-size vector.
  A dense layer with ReLU activation follows the GRU layers, introducing non-linearity to the model.
  A dropout layer is added to mitigate overfitting by randomly setting a fraction of input units to 0 during training.
  The final dense layer with a sigmoid activation function outputs binary predictions.

The code demonstrates a simple recurrent neural network (RNN) architecture for text classification.

Results Analysis:

Confusion Matrix:
- Generated and analysed a confusion matrix for all model predictions. The RNN model suffered from class imbalance and I was unable to rectify it.

Plot:
-I plotted two learning curves for each of the models in regard to training accuracy vs. validation accuracy and training loss vs. validation loss to better understand the learning of the model.

Performance Metrics:
- Evaluated and compared all models performances using accuracy, precision, recall, and F1 score.
CNN:
```
875/875 [==============================] - 45s 51ms/step
Confusion Matrix:
[[11952  2035]
 [ 2030 11983]]

Classification Report:
              precision    recall  f1-score   support
```

```
           0       0.85      0.85      0.85     13987
           1       0.85      0.86      0.85     14013

    accuracy                           0.85     28000
   macro avg       0.85      0.85      0.85     28000
weighted avg       0.85      0.85      0.85     28000
```

MLP:
```
875/875 [==============================] - 17s 19ms/step
Confusion Matrix:
[[11251  2736]
 [ 2721 11292]]

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.80      0.80     13987
           1       0.80      0.81      0.81     14013

    accuracy                           0.81     28000
   macro avg       0.81      0.81      0.81     28000
weighted avg       0.81      0.81      0.81     28000
```

Upon comparison of both the MLP model and the CNN model's Classification report which takes into account the confusion matrix and also calculates the precision, recall, f1 score, and support, the CNN model provided better results after training. The model training was however done significantly faster by the MLP model.

RNN:
```
875/875 [==============================] - 423s 483ms/step
Confusion Matrix:
[[    0 13987]
 [    0 14013]]

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00     13987
           1       0.50      1.00      0.67     14013

    accuracy                           0.50     28000
   macro avg       0.25      0.50      0.33     28000
weighted avg       0.25      0.50      0.33     28000
```

The RNN model's validation accuracy is close to 50% implying that it is very close to performing random guessing. The training loss and accuracy do not seem to be improving upon epochs implying there is something wrong with either the data or the model itself. I

made several attempts in modifying the parameters and the model architecture but was unable to correct it aptly.

Deployment:
- An attempt was made to explore deployment options for the model, such as web application integration or API deployment. This can be seen in the code for CNN implementation(AI2OG).

Continuous Improvement:
- Considered future improvements, including model fine-tuning based on user feedback or incorporating additional features.