CSE 587 Data Intensive Computing **Fall 2024**

PROJECT REPORT PHASE - 2

Topic - Buys Now Pay Later

Instructor: Naeem Maroof **Deadline: Nov/30/2024**

Team Members:

Name : Aniket Kumar **UBID:** akumar74
Name : Ashutosh Sharan **UBID:** asharan2
Name : Praanav Bhowmik **UBID:** praanavb

Problem Statement:

A Retail Company has introduced “Buy now pay later” (BNPL) feature within the mobile app platform. We need to predict if a customer will ignore or enroll in the BNPL feature.

The problem of predicting whether a customer will ignore or enroll in the Buy Now, Pay Later (BNPL) feature is significant for several reasons, both for a business and consumer perspective.

Growth of BNPL Services: BNPL services let customers buy something now and pay for it later in small installments, often without needing traditional credit checks. This has become popular, especially with younger shoppers. Businesses want to understand which customers are likely to use BNPL, so they can make their offers more appealing and relevant.

Revenue and Customer Retention: When customers choose BNPL, they tend to spend more, and they're more likely to shop again. If a business knows who is interested in BNPL, they can create better promotions and offers, leading to more sales and loyal customers.

In this phase, we have implemented various Machine Learning techniques to our dataset and analysed the outcomes of each ML model.

Dataset Overview:

We observe that column; ‘age’,’numscreen’ are slightly skewed

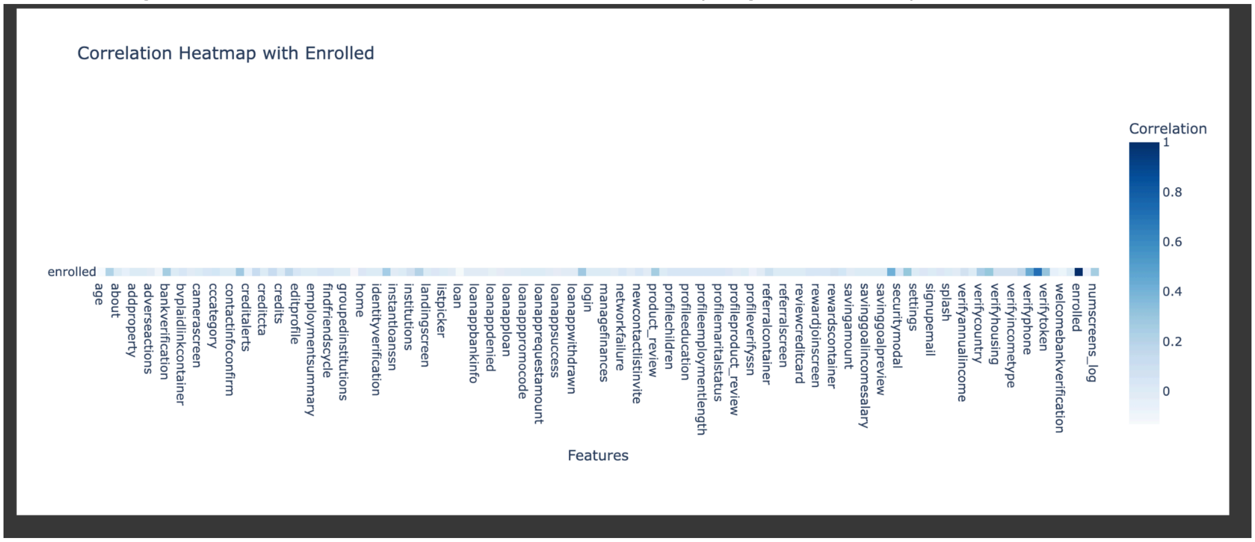
Day of Week Distribution: The distribution of user activity across days of the week varies, with a peak on day 4 (possibly Thursday) and relatively lower activity on days 2 and 3 (Tuesday and Wednesday).
Hourly Distribution: User activity is highest during the 15th to 20th hours of the day, suggesting peak usage during late afternoon and early evening.
Activity decreases during the early morning hours (0-9) and gradually increases from morning to late afternoon.
Age Distribution: The age distribution is right-skewed, with the majority of users in their early to mid-20s.
Numscreens Distribution: The distribution of the number of screens viewed by users is right-skewed, with a few users having a significantly higher number of screens.
Minigame and Used Premium Feature Distribution: The majority of users did not play the minigame (0), while a smaller percentage engaged in the minigame (1). A larger proportion of users did not use the premium feature (0), while a smaller percentage used the premium feature (1).
Liked Distribution: The distribution of the 'liked' feature suggests that a significant number of users did not indicate liking (0), while a smaller percentage did indicate liking (1).

Our target variable is ‘enrolled’ for which we want to implement various classifying algorithms to find out whether a new user will enroll or not.
We handle class imbalance using SMOTE based techniques, such as SMOT, adasyn, SMOTE + tomek links, smote + ENN.

Exploratory data Analysis:

The Exploratory Data Analysis (EDA) section of the presentation provides valuable insights into the dataset used for the Buy Now Pay Later (BNPL) Prediction Model. Here's a detailed explanation of the EDA findings:

Performing the correlation Heatmap with enrolled(target variable):



Therefore the length of important features that can be used is 25 which includes the target column as well ‘enrolled’.

```
imp_features=['age_log','numscreens_log','alerts','bankverif
             'profileverifyssn','scanpreview','selectinstit
             ]
len(imp_features)
```

```
24
```

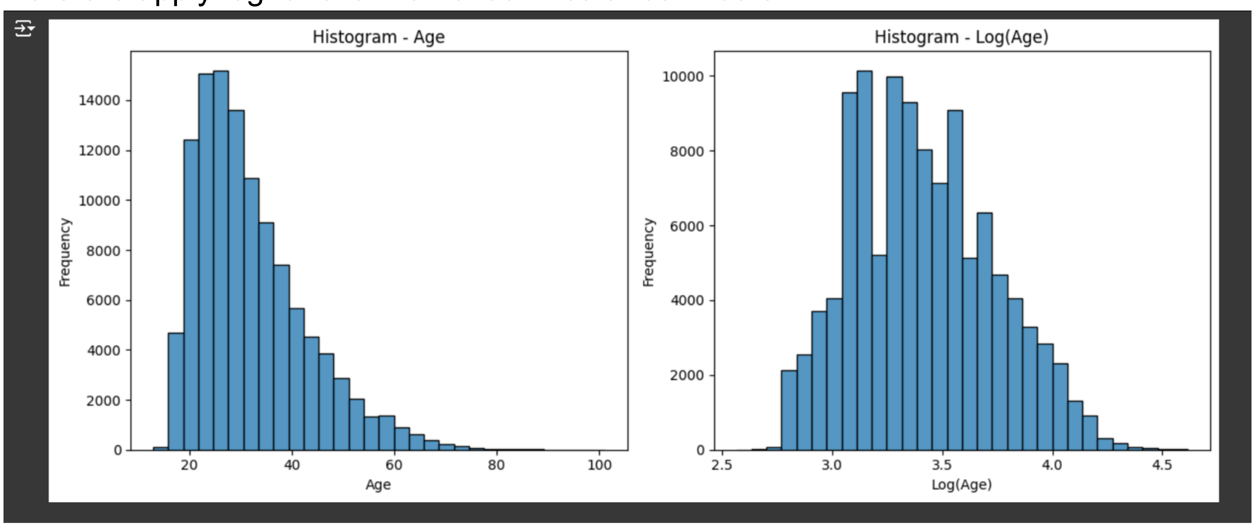
```
24
```

```
[ ] feature_engineered_df = pd.DataFrame()
    feature_engineered_df = engineered_df[['enrolled']+imp_features]
    len(feature_engineered_df.columns)
```

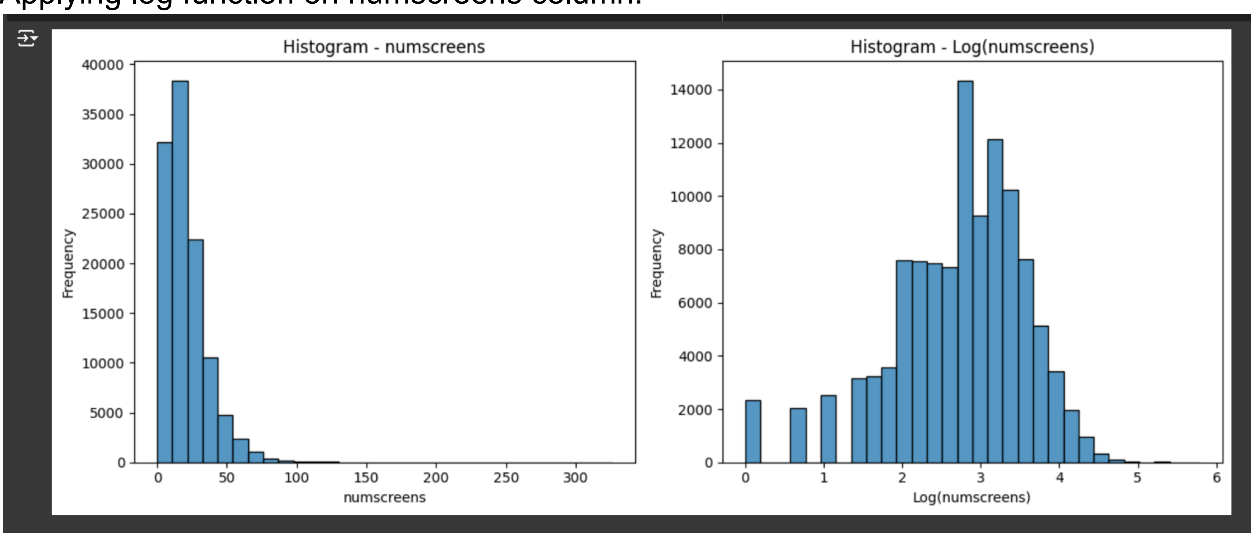
```
25
```

Algorithms Training Report:

Apply log function of Age column and Age column: SInce the data was skewed, therefore apply log function removed imbalance in data.



Applying log function on numscreens column:



- **Logistic Regression:**

- For this algorithm we got the accuracy of 91.9%.

```
Confusion Matrix
============================================================
[[9525  582]
 [2801 9588]]

Classification Report
============================================================
              precision    recall  f1-score   support

           0       0.77      0.94      0.85     10107
           1       0.94      0.77      0.85     12389

    accuracy                           0.85     22496
   macro avg       0.86      0.86      0.85     22496
weighted avg       0.87      0.85      0.85     22496


AUC-ROC
============================================================
0.9196529613561082
```

- On **SMOTE** dataset:

```
Confusion Matrix
============================================================
[[9665  442]
 [2968 9421]]

Classification Report
============================================================
              precision    recall  f1-score   support

           0       0.77      0.96      0.85     10107
           1       0.96      0.76      0.85     12389

    accuracy                           0.85     22496
   macro avg       0.86      0.86      0.85     22496
weighted avg       0.87      0.85      0.85     22496


AUC-ROC
============================================================
0.9196406146539717
```

- On **ADASYN** dataset:

```
Confusion Matrix
============================================================
[[9736  371]
 [3071 9318]]

Classification Report
============================================================
              precision    recall  f1-score   support

           0       0.76      0.96      0.85     10107
           1       0.96      0.75      0.84     12389

    accuracy                           0.85     22496
   macro avg       0.86      0.86      0.85     22496
weighted avg       0.87      0.85      0.85     22496


AUC-ROC
============================================================
0.9196790283908902
```

- On **SMOTE + Tomek Links** dataset:

```
Confusion Matrix
============================================================
[[9671  436]
 [2975 9414]]

Classification Report
============================================================
              precision    recall  f1-score   support

           0       0.76      0.96      0.85     10107
           1       0.96      0.76      0.85     12389

    accuracy                           0.85     22496
   macro avg       0.86      0.86      0.85     22496
weighted avg       0.87      0.85      0.85     22496


AUC-ROC
============================================================
0.9194078321999805
```

- On **SMOTE + ENN** dataset:

```
Confusion Matrix
============================================================
[[9704  403]
 [2982 9407]]

Classification Report
============================================================
              precision    recall  f1-score   support

           0       0.76      0.96      0.85     10107
           1       0.96      0.76      0.85     12389

    accuracy                           0.85     22496
   macro avg       0.86      0.86      0.85     22496
weighted avg       0.87      0.85      0.85     22496


AUC-ROC
============================================================
0.9158087126236636
```

Descriptive overview for each dataset given below:

	model	resample	precision	recall	f1-score	AUC-ROC
0	Logistic Regression	actual	0.942773	0.773912	0.850038	0.919653
1	Logistic Regression	smote	0.955186	0.760433	0.846755	0.919641
2	Logistic Regression	adasyn	0.961709	0.752119	0.844098	0.919679
3	Logistic Regression	smote+tomek	0.955736	0.759868	0.846621	0.919408
4	Logistic Regression	smote+enn	0.958919	0.759303	0.847516	0.915809

SVM:

For this algorithm we got the accuracy of 82.89%.

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

           0       0.72      1.00      0.84     10107
           1       1.00      0.69      0.82     12389

    accuracy                           0.83     22496
   macro avg       0.86      0.84      0.83     22496
weighted avg       0.88      0.83      0.83     22496

Accuracy Score: 0.8289473684210527
```

Algorithms Evaluation Report:

The evaluation report for our models is given below.

- **Logistic Regression:** Using different values of hyperparameters to train the model using different c values from the range of 0.001 till 1000.

Hypertuning Logistic Regression

```
params = {
    'penalty':['l1', 'l2'],
    'C':[0.001, 0.01, 0.1, 1, 10, 100, 1000]

}
model = LogisticRegression()
# using recall as scoring metrics because we do
grid_search = GridSearchCV(estimator=model, par
grid_search.fit(X_train_scaled,y_train)
```

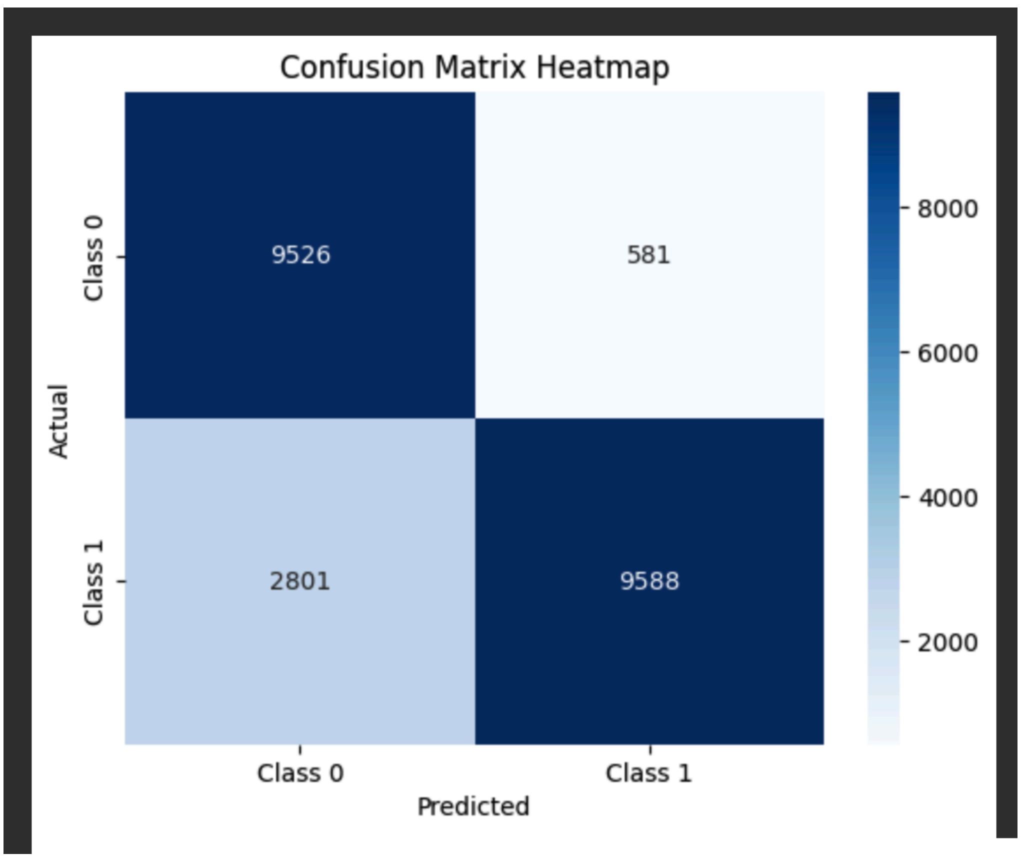
```
Fitting 5 folds for each of 14 candidates, tota
GridSearchCV
best_estimator_: LogisticRegression
LogisticRegression
```

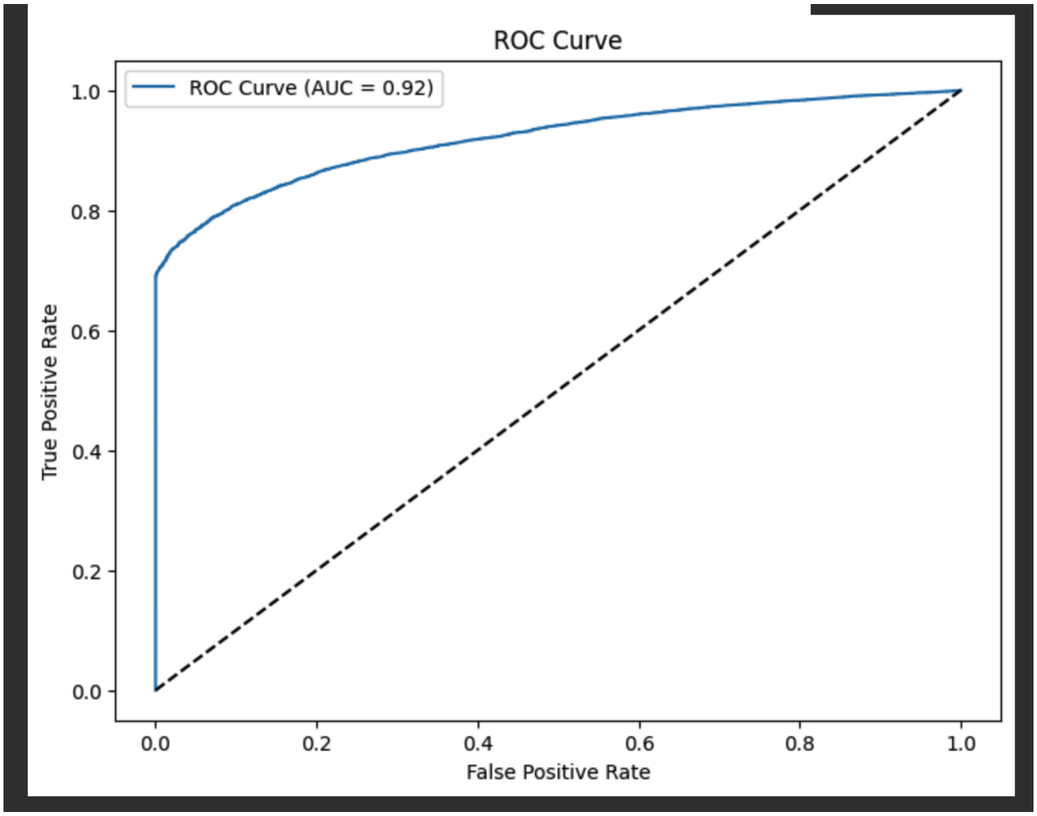
```
[ ] grid_search.best_params_
```

```
{'C': 100, 'penalty': 'l2'}
```

For this model we got train accuracy :85.24% and test accuracy :84.96%.

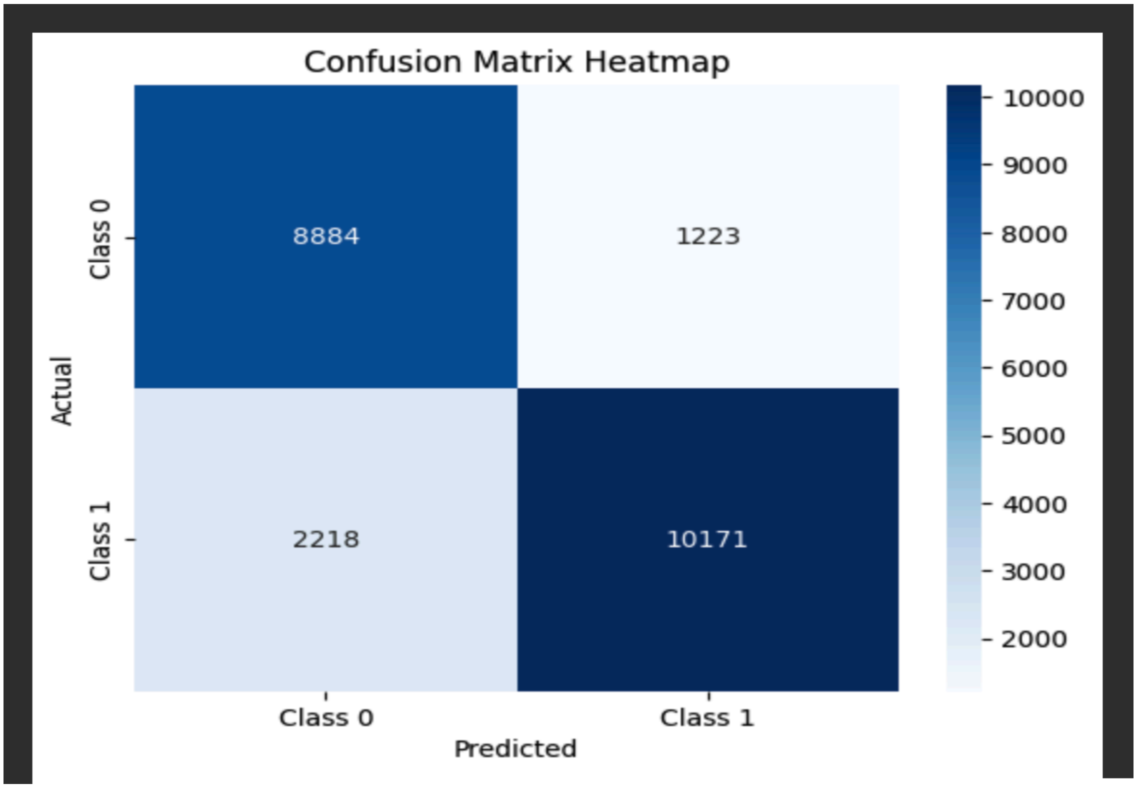
```
Training accuracy:85.2416567573875
Test accuracy:84.96621621621621

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.94      0.85     10107
           1       0.94      0.77      0.85     12389

    accuracy                           0.85     22496
   macro avg       0.86      0.86      0.85     22496
weighted avg       0.87      0.85      0.85     22496
```

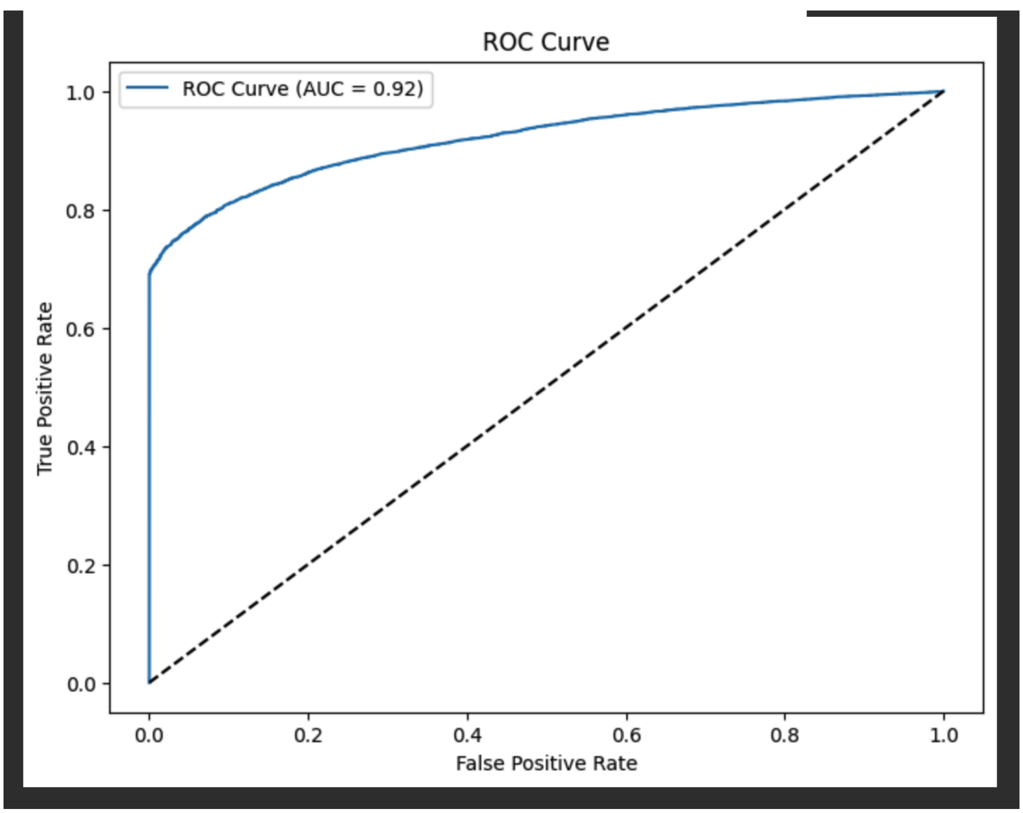




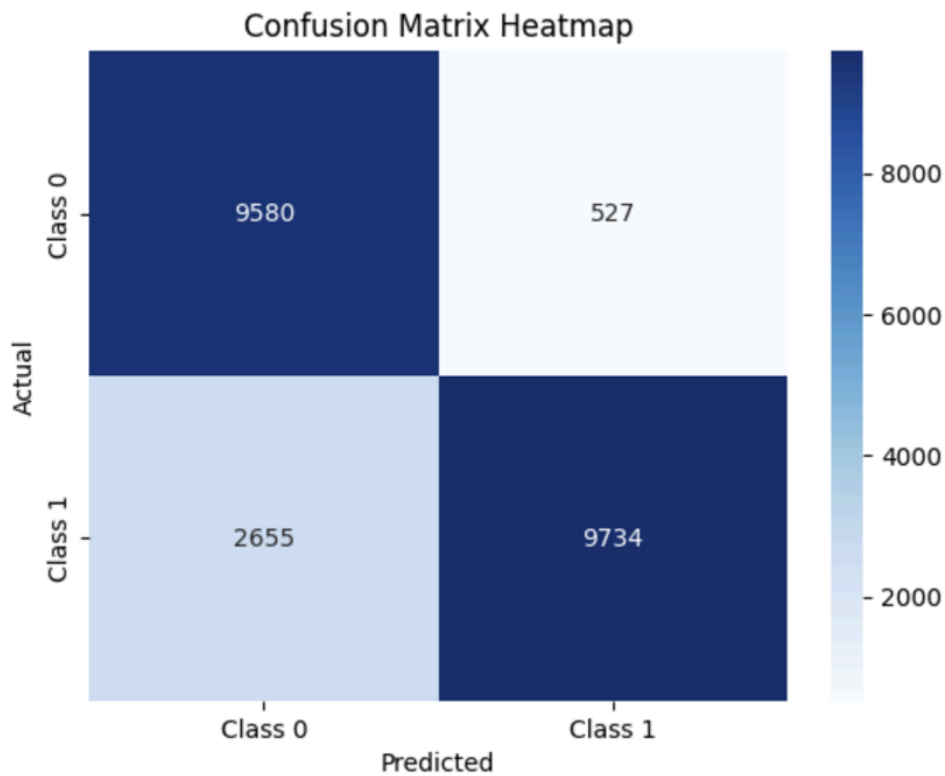
Using a different threshold to reduce the number of False Negative:

```
Training accuracy:85.23165486814176
Test accuracy:84.96177098150783

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.88      0.84     10107
           1       0.89      0.82      0.86     12389

    accuracy                           0.85     22496
   macro avg       0.85      0.85      0.85     22496
weighted avg       0.85      0.85      0.85     22496
```


Confusion Matrix Heatmap
8884
1223
2218
10171
Class 0
Class 1
Actual
Class 0
Class 1
Predicted
10000
9000
8000
7000
6000
5000
4000
3000
2000

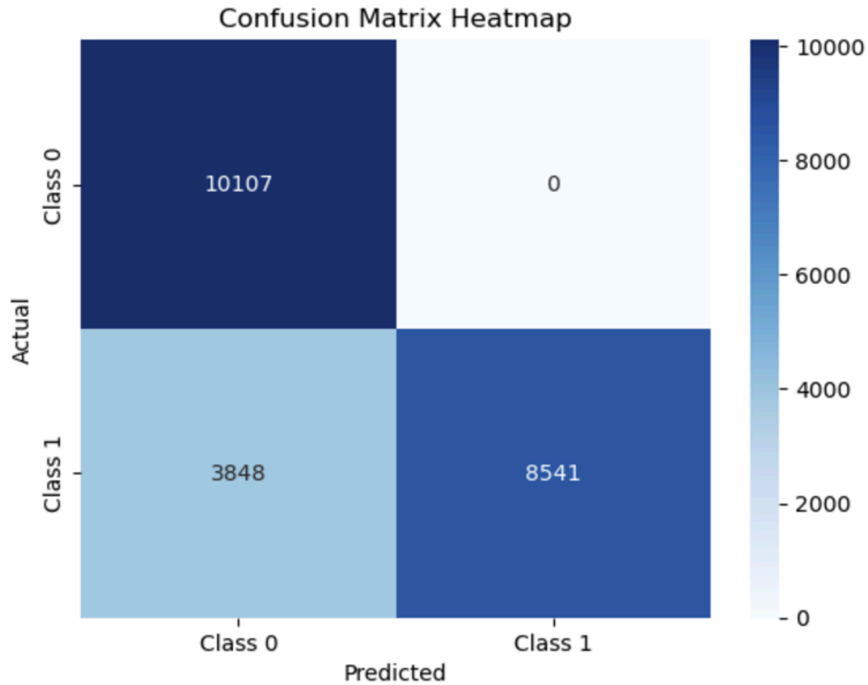

ROC Curve
ROC Curve (AUC = 0.92)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

- **Gradient Boosting:**
  - For this model we got train accuracy :86.16% and test accuracy :85.85%.



- **SVM:**
  - For this model we got train accuracy :83.38% and test accuracy :82.89%.

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

           0       0.72      1.00      0.84     10107
           1       1.00      0.69      0.82     12389

    accuracy                           0.83     22496
   macro avg       0.86      0.84      0.83     22496
weighted avg       0.88      0.83      0.83     22496

Accuracy Score: 0.8289473684210527
```



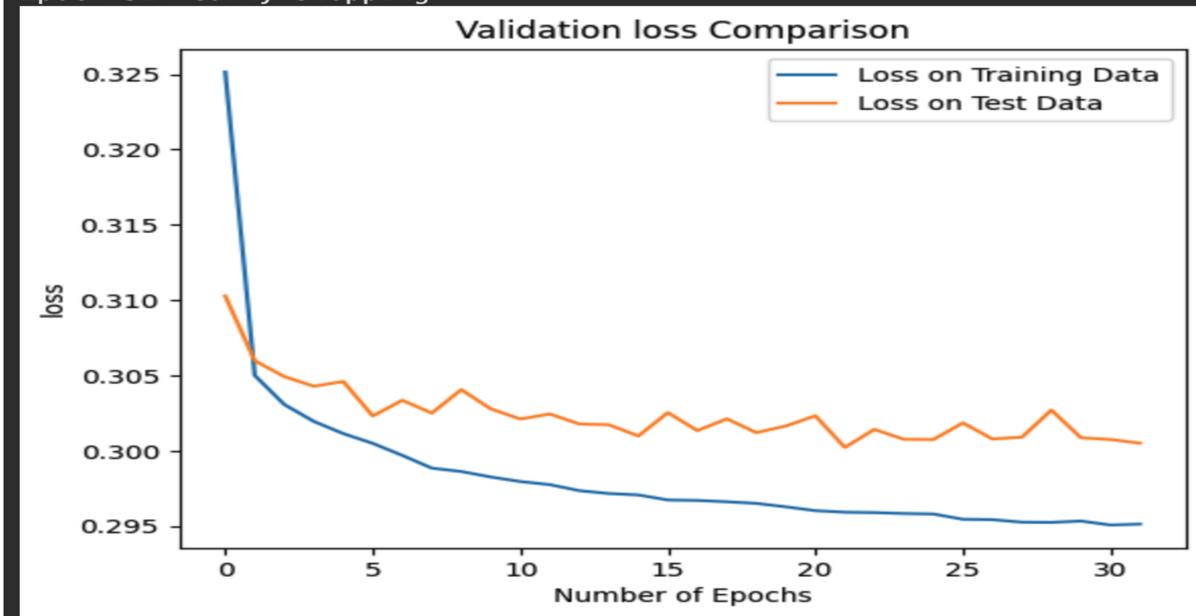
- **Neural Network Model** with 3 different optimiser for the given dataset: Optimizers used are Adam(), SGD() and RMSprop().
  - For Neural network with 1 hidden layer and 1 output layer:
  - **Adam** as optimiser:
  - we got train accuracy: 86.2 and validation loss: 29.86%

```
Optimizer: adam
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 15)	375
dense_1 (Dense)	(None, 1)	16

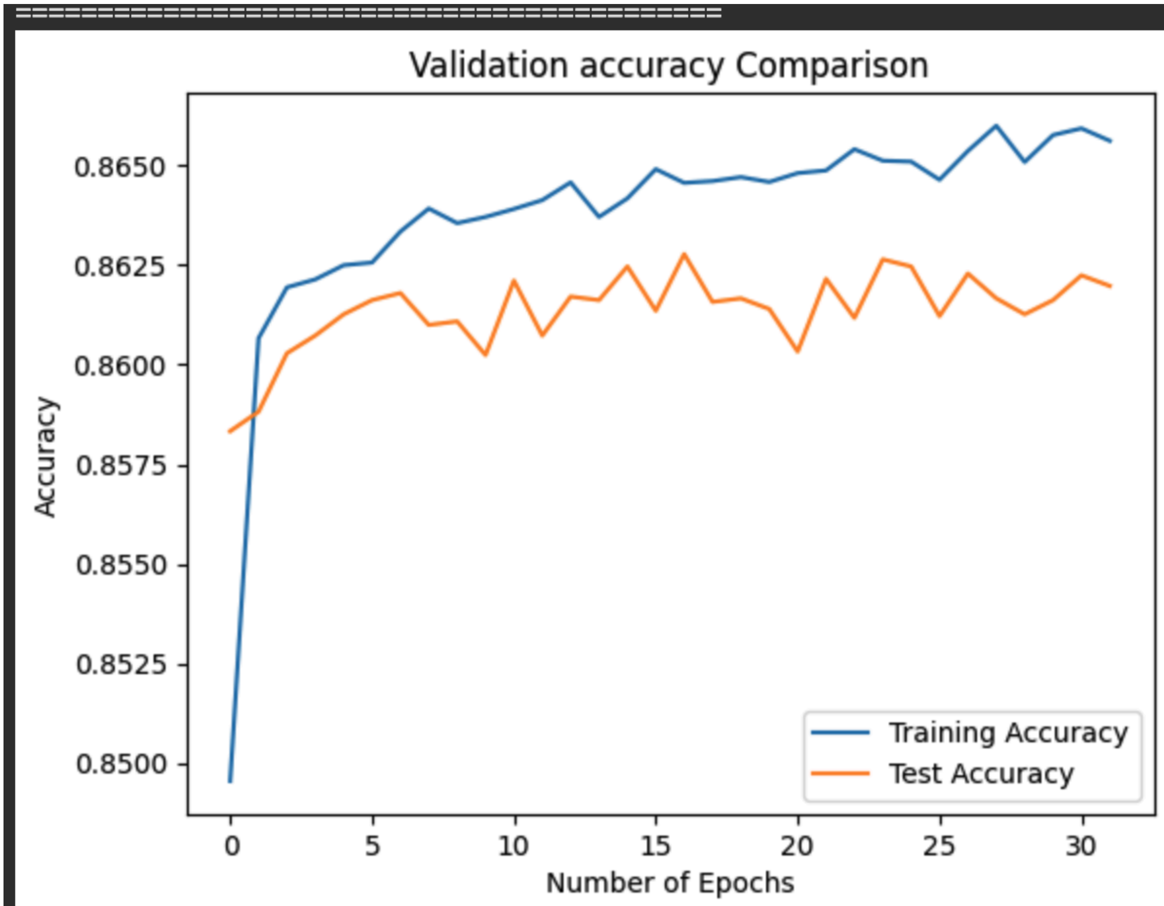
```
 Total params: 391 (1.53 KB)
 Trainable params: 391 (1.53 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 14s 2ms/step - accuracy: 0.8247 - loss: 0.3667 - val_accuracy: 0.8
Epoch 2/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 12s 2ms/step - accuracy: 0.8609 - loss: 0.3052 - val_accuracy: 0.8
Epoch 3/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 12s 2ms/step - accuracy: 0.8620 - loss: 0.3020 - val_accuracy: 0.8
```

```
Epoch 32: early stopping
```




```
==============================================
```

```
============================================
```

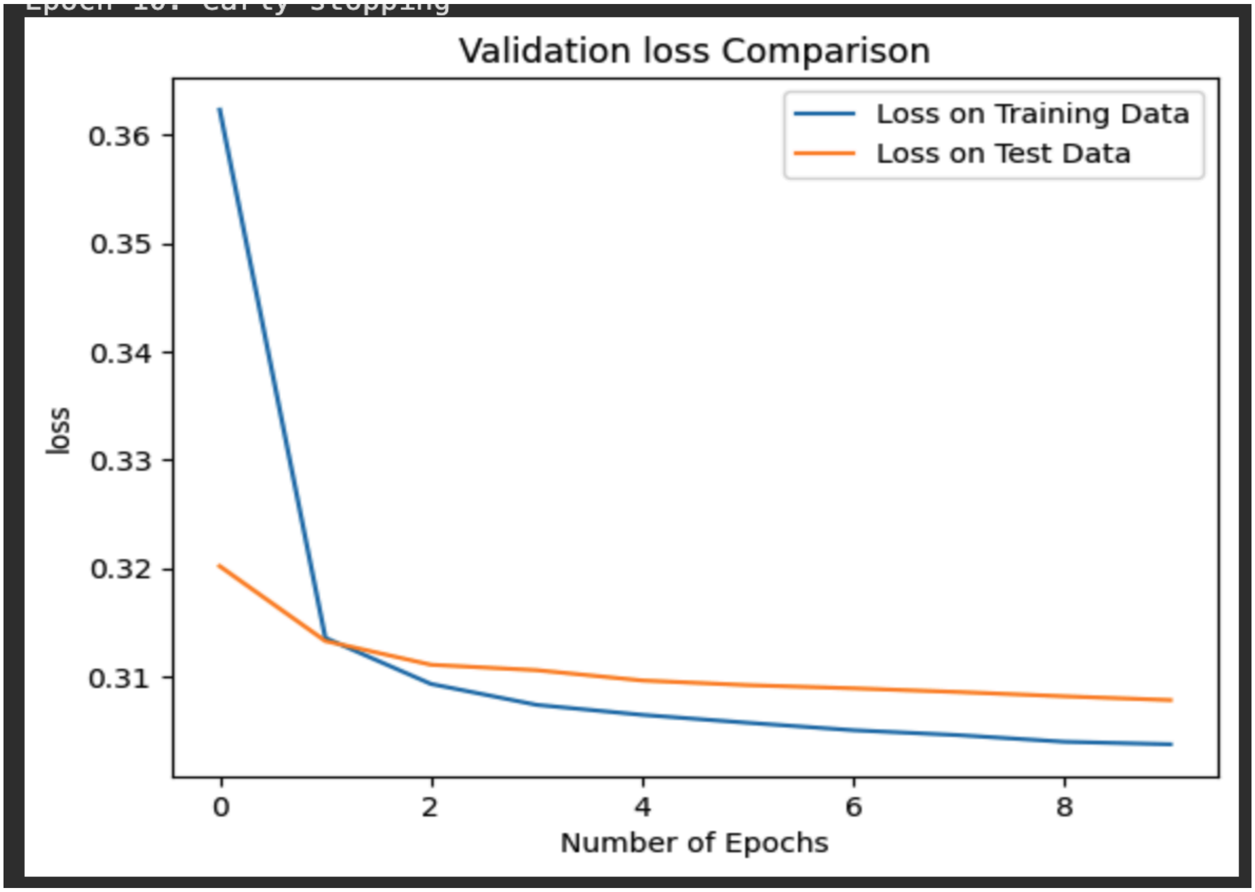


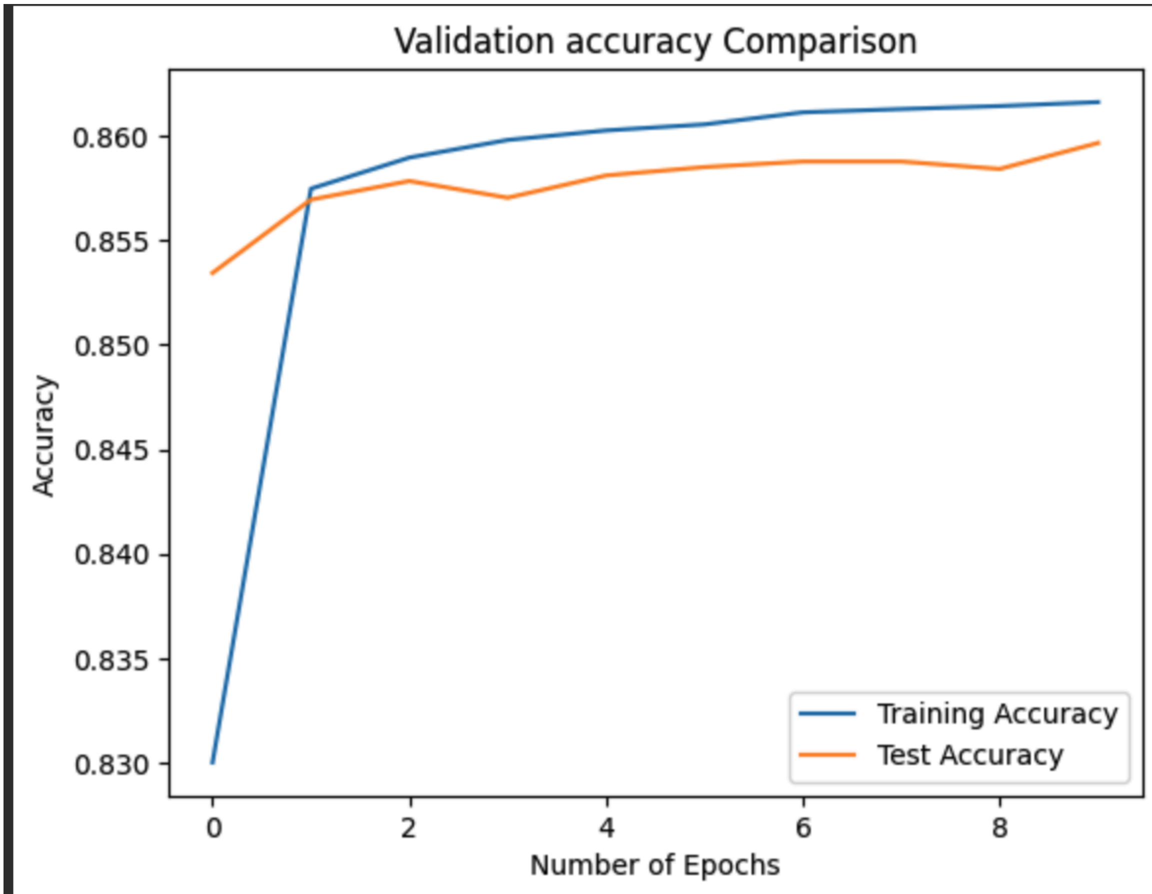

```
703/703 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8640 - loss: 0.2983
Loss: 0.3005, Accuracy: 0.8620
```

- For Neural network with 1 hidden layer and 1 output layer:
- **SGD** as optimiser:
- we got train accuracy: 86.2 and validation loss: 29.86%

```
Optimizer: SGD
Model: "sequential_1"
┌─────────────────────────┬─────────────────────┬──────────┐
│ Layer (type)            │ Output Shape        │  Param # │
├─────────────────────────┼─────────────────────┼──────────┤
│ dense_2 (Dense)         │ (None, 15)          │      375 │
├─────────────────────────┼─────────────────────┼──────────┤
│ dense_3 (Dense)         │ (None, 1)           │       16 │
└─────────────────────────┴─────────────────────┴──────────┘
 Total params: 391 (1.53 KB)
 Trainable params: 391 (1.53 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 13s 2ms/step - accuracy: 0.7867 - loss: 0.4292 - val_accuracy: 0.8534 - val_loss: 0.3202
Epoch 2/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 20s 2ms/step - accuracy: 0.8569 - loss: 0.3146 - val_accuracy: 0.8570 - val_loss: 0.3132
Epoch 3/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 11s 2ms/step - accuracy: 0.8597 - loss: 0.3092 - val_accuracy: 0.8578 - val_loss: 0.3110
Epoch 4/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 12s 2ms/step - accuracy: 0.8599 - loss: 0.3088 - val_accuracy: 0.8570 - val_loss: 0.3105
Epoch 5/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 12s 2ms/step - accuracy: 0.8607 - loss: 0.3066 - val_accuracy: 0.8581 - val_loss: 0.3096
```

```
Epoch 10: early stopping
```





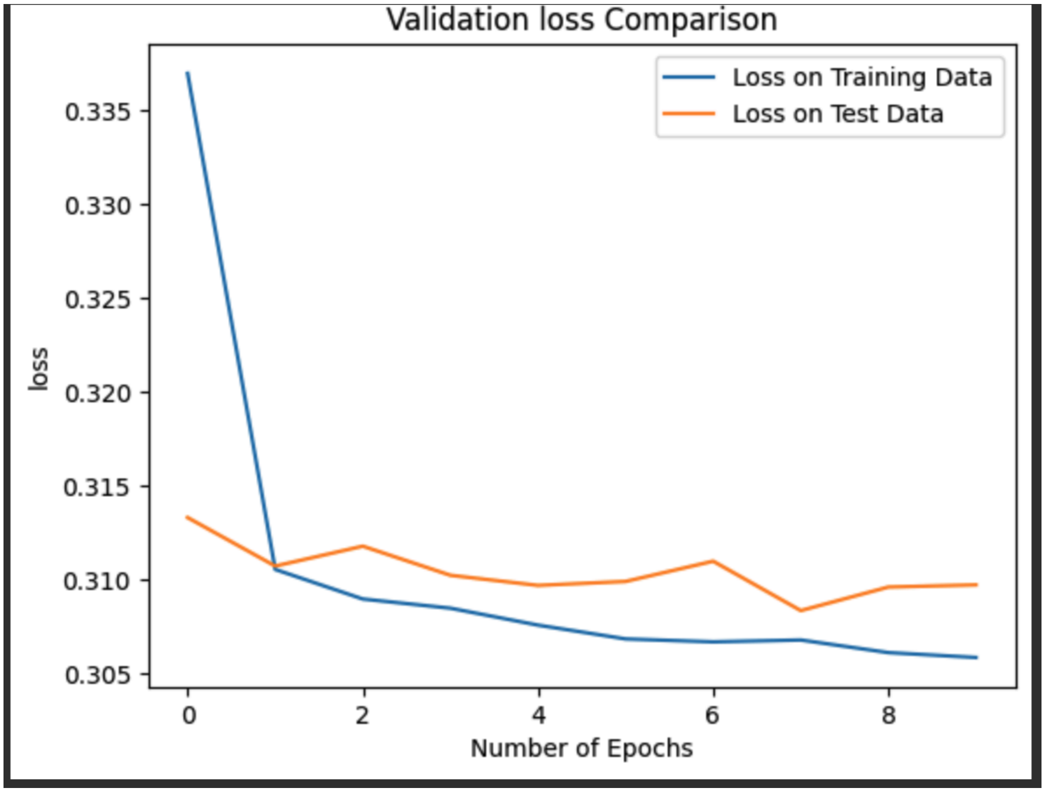

```
703/703 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8624 - loss: 0.3067
Loss: 0.3078, Accuracy: 0.8597
```

- For Neural network with 1 hidden layer and 1 output layer:
- **RMSprop** as optimiser:
- we got train accuracy: 86.2 and validation loss: 29.86%

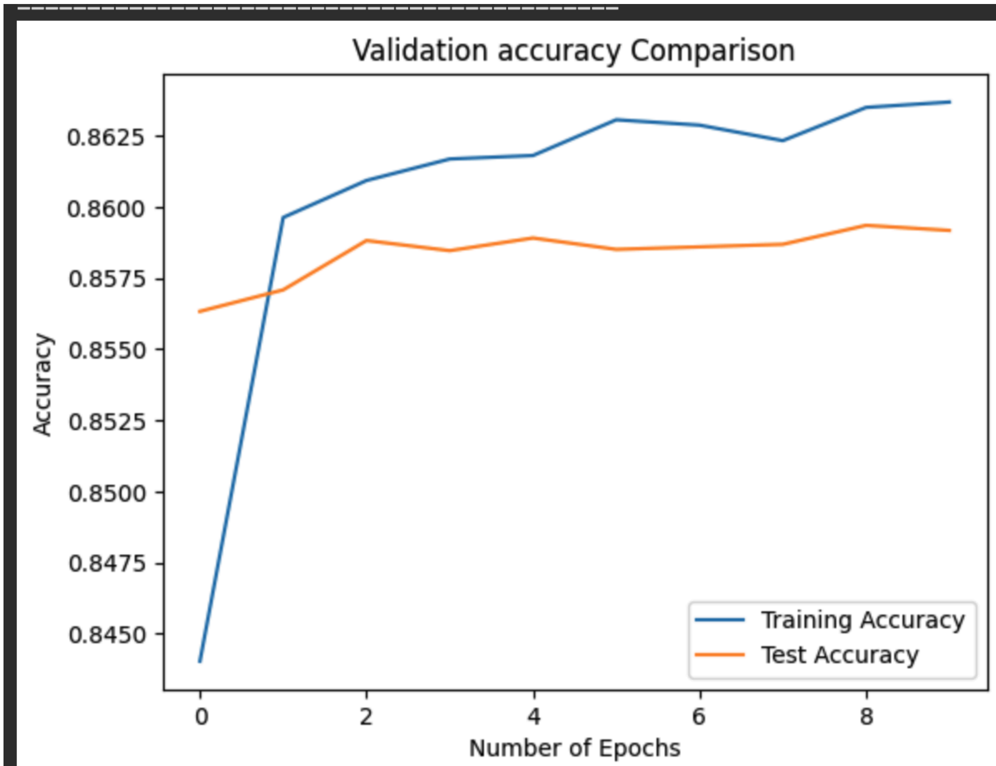
```
Optimizer: rmsprop
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 15)	375
dense_5 (Dense)	(None, 1)	16

```
 Total params: 391 (1.53 KB)
 Trainable params: 391 (1.53 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 13s 2ms/step - accuracy: 0.8092 - loss: 0.3895 - val_accuracy: 0.8563 -
Epoch 2/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 12s 2ms/step - accuracy: 0.8614 - loss: 0.3083 - val_accuracy: 0.8571 -
Epoch 3/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 20s 2ms/step - accuracy: 0.8612 - loss: 0.3088 - val_accuracy: 0.8588 -
Epoch 4/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 12s 2ms/step - accuracy: 0.8610 - loss: 0.3086 - val_accuracy: 0.8585 -
```



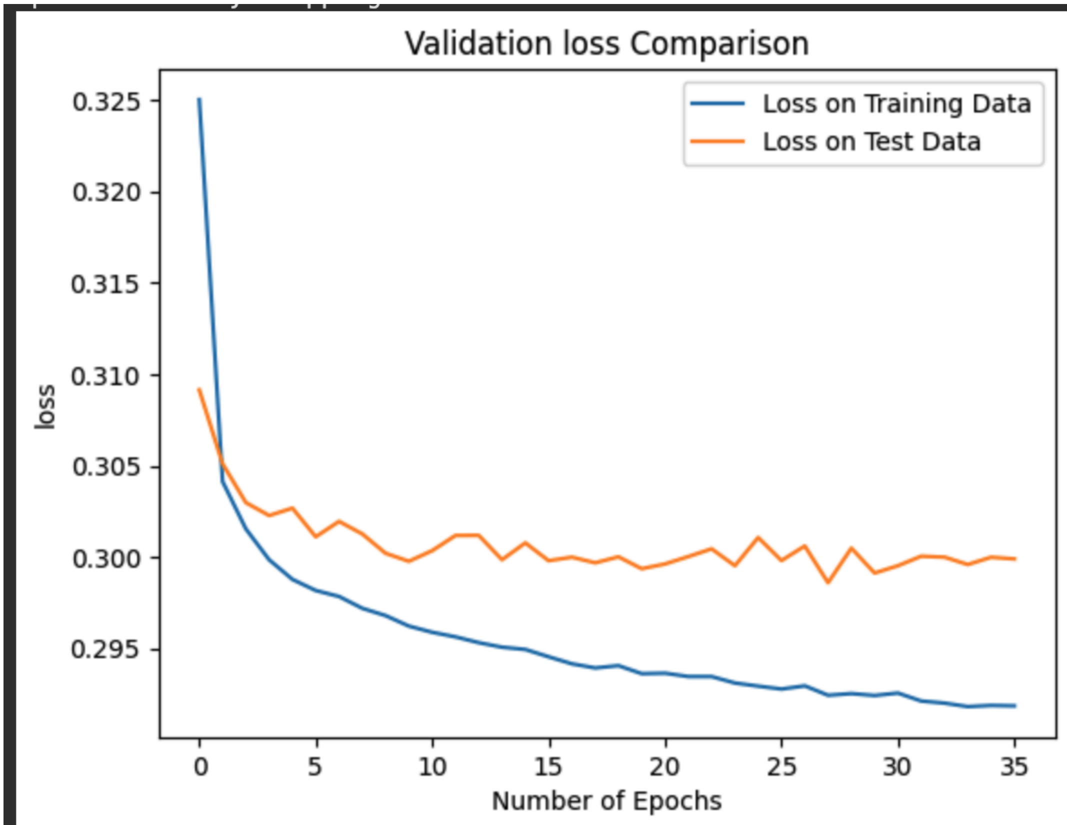

```
-------------------------------------------
```



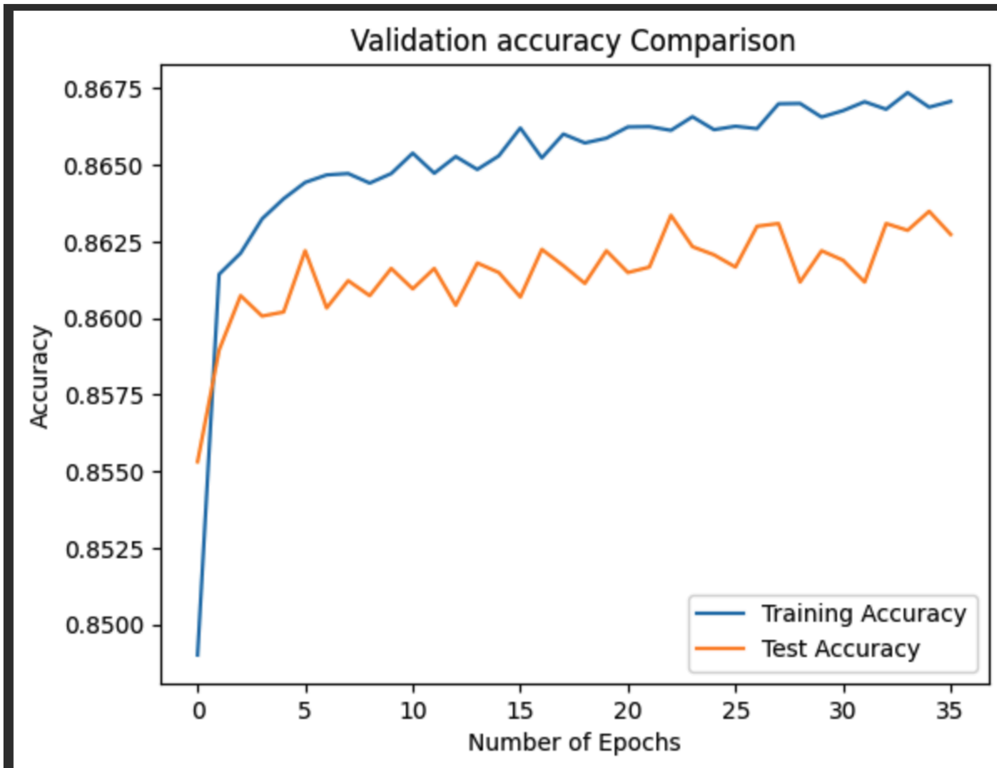

```
703/703 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8618 - loss: 0.3074
Loss: 0.3097, Accuracy: 0.8592
```

- For Neural network with 2 hidden layer and 1 output layer:
- **Adam** as optimiser:
- we got train accuracy: 86.2 and validation loss: 29.86%

```
Optimizer: adam
Model: "sequential_3"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape           ┃    Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━┩
│ dense_6 (Dense)                      │ (None, 15)             │        375 │
├──────────────────────────────────────┼────────────────────────┼────────────┤
│ dense_7 (Dense)                      │ (None, 10)             │        160 │
├──────────────────────────────────────┼────────────────────────┼────────────┤
│ dense_8 (Dense)                      │ (None, 1)              │         11 │
└──────────────────────────────────────┴────────────────────────┴────────────┘
 Total params: 546 (2.13 KB)
 Trainable params: 546 (2.13 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step - accuracy: 0.8197 - loss: 0.3678 - val_accuracy
Epoch 2/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 19s 2ms/step - accuracy: 0.8615 - loss: 0.3046 - val_accuracy
```




```
==================================================
```



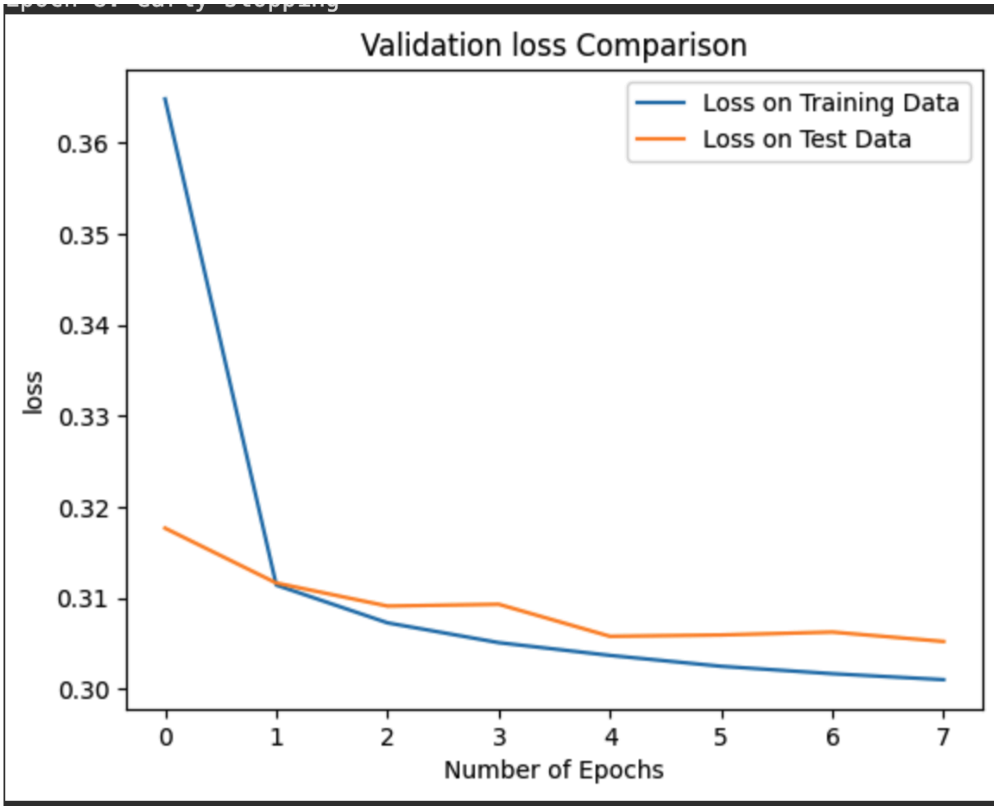

```
703/703 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8635 - loss: 0.2986
Loss: 0.2999, Accuracy: 0.8627
```

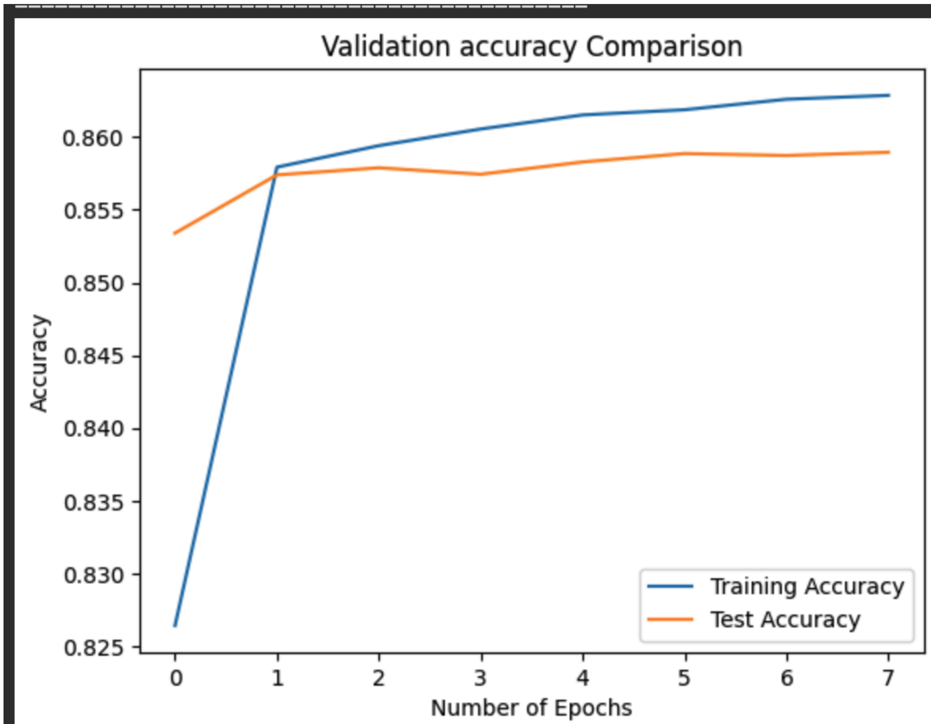
- For Neural network with 2 hidden layer and 1 output layer:
- **SGD** as optimiser:
- we got train accuracy: 86.2 and validation loss: 29.86%

```
Optimizer: SGD
Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 15)	375
dense_10 (Dense)	(None, 10)	160
dense_11 (Dense)	(None, 1)	11

```
 Total params: 546 (2.13 KB)
 Trainable params: 546 (2.13 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step - accuracy: 0.7725 - loss: 0.4416 - val_accuracy: 0.8
Epoch 2/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 17s 2ms/step - accuracy: 0.8573 - loss: 0.3129 - val_accuracy: 0.8
Epoch 3/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 12s 2ms/step - accuracy: 0.8571 - loss: 0.3102 - val_accuracy: 0.8
```






```
703/703 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8600 - loss: 0.3043
Loss: 0.3052, Accuracy: 0.8590
```

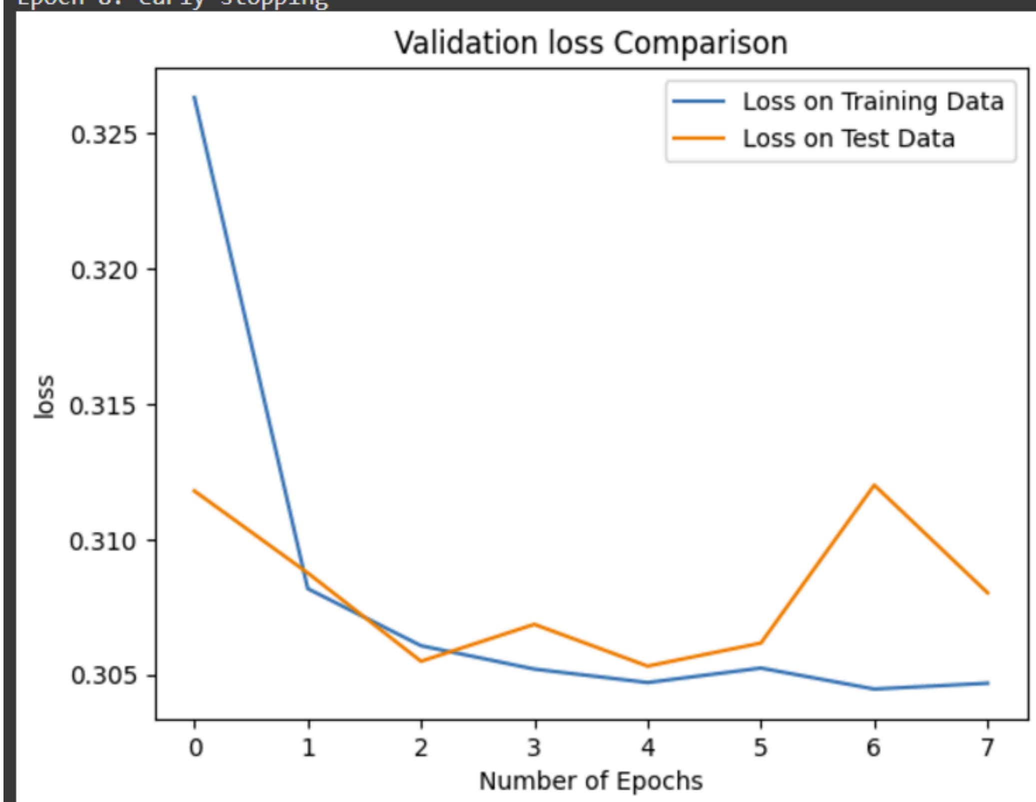
- For Neural network with 2 hidden layer and 1 output layer:
- **RMSprop** as optimiser:
- we got train accuracy: 86.01 and validation loss: 30.68%

```
Optimizer: rmsprop
Model: "sequential_5"
```

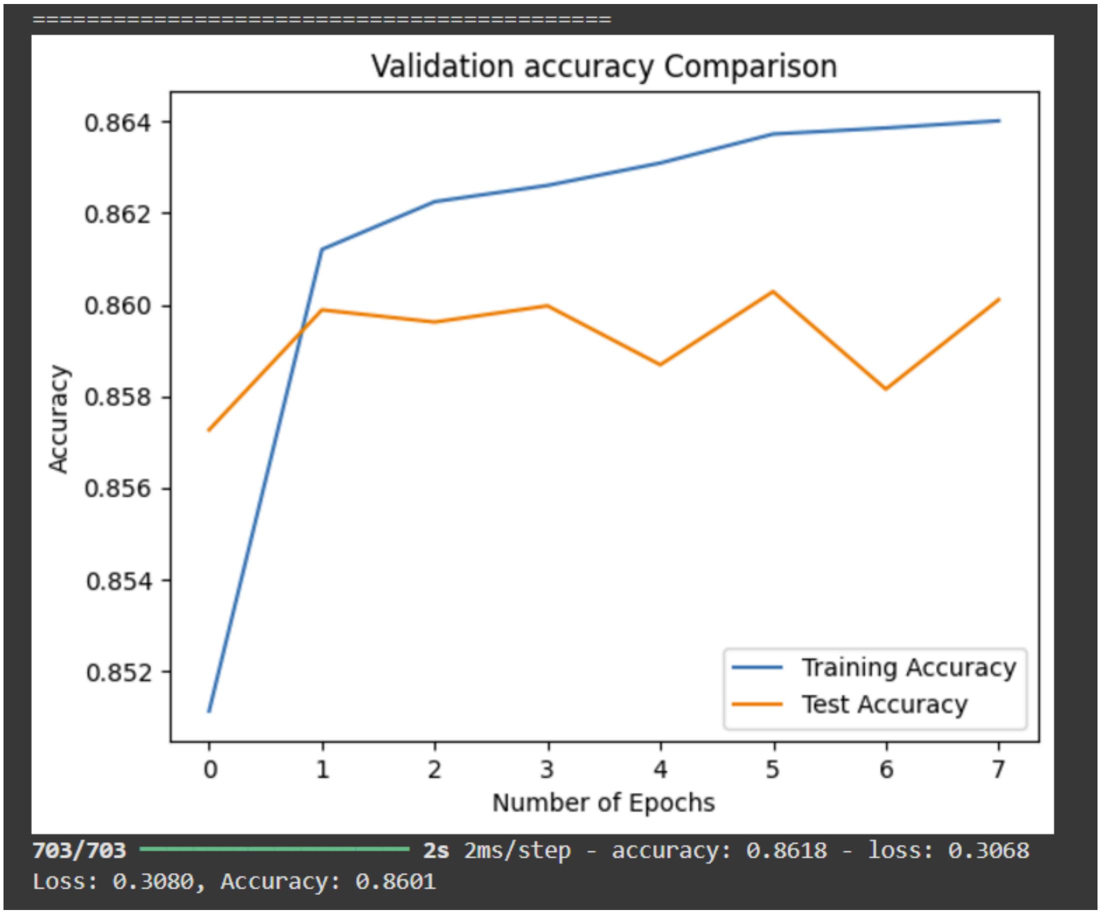
Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 15)	375
dense_13 (Dense)	(None, 10)	160
dense_14 (Dense)	(None, 1)	11

```
 Total params: 546 (2.13 KB)
 Trainable params: 546 (2.13 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 14s 2ms/step - accuracy: 0.8276 - loss: 0.3647 - val_accuracy: 0.8573 - val_loss: 0.3118
Epoch 2/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 13s 2ms/step - accuracy: 0.8621 - loss: 0.3072 - val_accuracy: 0.8599 - val_loss: 0.3088
Epoch 3/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 22s 2ms/step - accuracy: 0.8650 - loss: 0.3032 - val_accuracy: 0.8596 - val_loss: 0.3055
Epoch 4/100
5999/5999 ━━━━━━━━━━━━━━━━━━━━ 12s 2ms/step - accuracy: 0.8620 - loss: 0.3064 - val_accuracy: 0.8600 - val_loss: 0.3069
```

```
Epoch 8: early stopping
```




```
==========================================
```



Observation:

Logistic Regression provides a balanced performance with a minimal gap between train and test accuracy (0.27%), indicating a well-generalized model. This model might be suitable if interpretability and simplicity are priorities.

Gradient Boosting shows the best performance among the models in both train and test accuracy. The small difference between train and test accuracy (0.31%) suggests that the model is neither overfitting nor underfitting. Given its strong predictive power, this model is well-suited for production use if computational complexity is manageable.

SVM performs the worst among the models in both train and test accuracy. A minimal gap between train and test accuracy (0.49%) indicates good generalization. The lower accuracy could be due to the model's inability to capture non-linear relationships effectively or inadequate feature scaling.

The **neural network** achieves the highest training accuracy (86.27%) among the models, indicating its ability to learn complex patterns. However, the validation loss is significantly high, suggesting potential overfitting or insufficient tuning of hyperparameters. Regularization techniques like dropout, batch normalization, or hyperparameter optimization could improve validation performance.

Summary:

- **Gradient Boosting** emerges as the best model overall, achieving high accuracy with excellent generalization. It is the recommended model for deployment if accuracy is the main priority.
- **Logistic Regression** is a close second, offering decent accuracy with lower computational complexity and interpretability, making it ideal for quick insights.
- **SVM** underperforms compared to other models. Consider using non-linear kernels or further feature engineering to improve its performance.
- **Neural Network** shows promise with high training accuracy but requires additional tuning to reduce validation loss and generalize better.