1. Introduction

The business problem our team likes to address is to predict whether a customer is likely to churn, meaning they may discontinue their relationship with the firm. This task is crucial for business, because retaining existing customers is generally more cost-effective than acquiring new ones. By predicting churn, firms can proactively implement targeted retention strategies for customers at risk, thereby enhancing customer loyalty and minimizing losses. Also, businesses need to anticipate the number of churn in order to help them make better decisions. For example, forecast future sales from churning in order to optimise material sourcing and make better expansion decisions. Therefore, we believe this area is worth exploring.

We formulate this task as a supervised learning classification problem, with the target variable indicating whether a customer will "Exited" or "Not Exited." Our initial idea is to try both the decision tree method and logistic regression, along with other modeling techniques if necessary.

2. Data Understanding and Preparation

Dataset: [Bank Customer Churn Prediction]

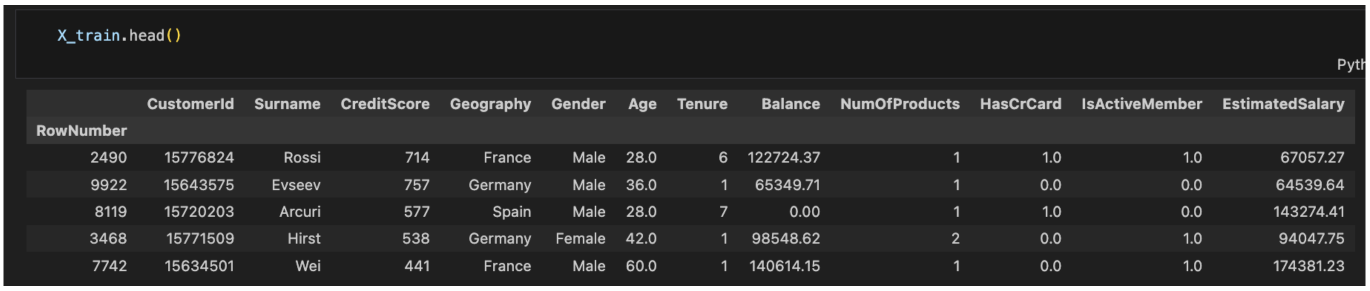
The dataset used for this project is from Kaggle, it includes 10,000 examples, each representing a bank customer. It includes 12 attributes and one target variable (Exited):

We first split the data into 70% training set and 30% testing set, resulting in 7001 instances for the training set and 2999 instances for testing set. We then visualise the training set and perform data preprocessing on the training set first. (For the testing set, we will not preprocess now to avoid data leakage, instead, we would preprocess the testing test after we finish building the model with the training set, with the same method and value of how we handle the training set.) The training set will be used to train the models, and the testing set will be used to evaluate their performance on unseen data. This ensures that we're not just memorizing the data but actually learning patterns that can be applied to new customers.

2.1 Data Understanding

The dataset was initially loaded. Following this, several exploratory steps were undertaken to gain a thorough understanding of the data's characteristics. The head(), info(), and describe() methods were employed to inspect the first few rows, data types, value distributions, and to detect any potential anomalies or missing values within the dataset.

Initial dataset



```
X_train.head()
```

```
           CustomerId  Surname  CreditScore  Geography  Gender   Age  Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  EstimatedSalary
RowNumber
     2490    15776824    Rossi          714     France    Male  28.0       6  122724.37              1        1.0             1.0         67057.27
     9922    15643575   Evseev          757    Germany    Male  36.0       1   65349.71              1        0.0             0.0         64539.64
     8119    15720203   Arcuri          577      Spain    Male  28.0       7       0.00              1        1.0             0.0        143274.41
     3468    15771509    Hirst          538    Germany  Female  42.0       1   98548.62              2        0.0             1.0         94047.75
     7742    15634501      Wei          441     France    Male  60.0       1  140614.15              1        0.0             1.0        174381.23
```

Attributes Description

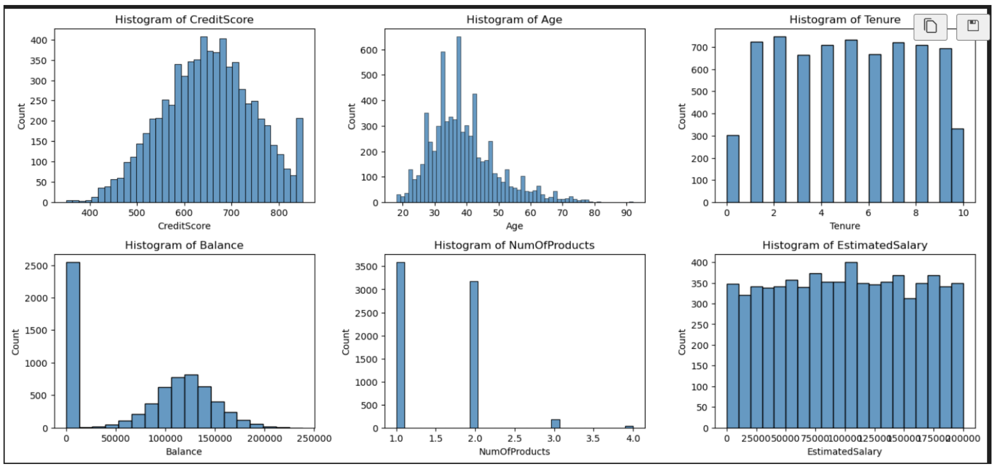
Column Name	Data Type	Description
RowNumber	Integer	Index of the record (redundant)
CustomerId	Integer	Unique identifier for each customer
Surname	String	Customer's last name
CreditScore	Integer	Credit score of the customer
Geography	Categorical (Nominal)	Country of residence (France, Germany, Spain)
Gender	Categorical (Binary)	Male or Female
Age	Integer	Age of the customer
Tenure	Integer	Number of years as a bank customer
Balance	Float	Account balance
NumOfProducts	Integer	Number of bank products used

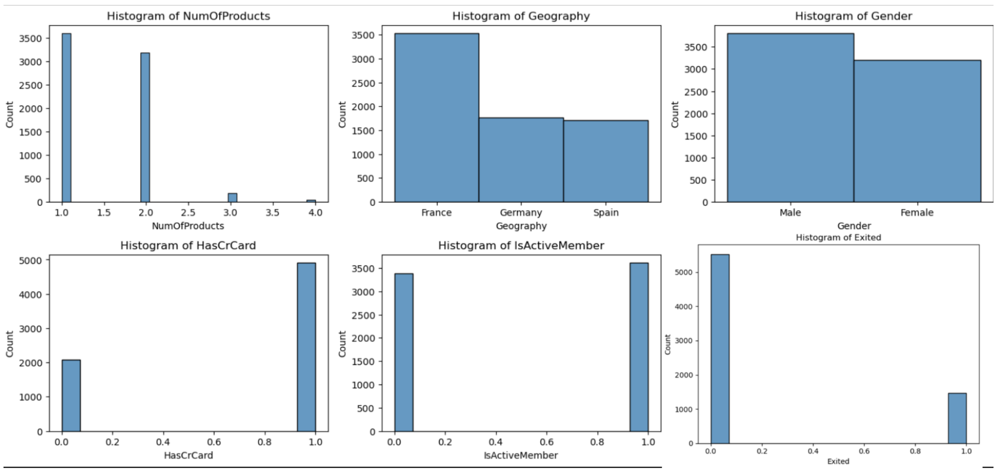
HasCrCard	Binary (0/1)	Whether the customer has a credit card
IsActiveMember	Binary (0/1)	Whether the customer is an active member
EstimatedSalary	Float	Estimated salary of the customer
Exited (Target)	Binary (0/1)	Whether the customer churned (1) or not (0)

Attribute Types

- Categorical: CustomerId, Surname, Geography, Gender, HasCrCard, IsActiveMember,
- Numerical: CreditScore, Age, Tenure, Balance, Number of Products, Estimated Salary

We plot graphs to understand the distribution of the numerical variable and to check whether there is a class imbalance problem for categorical variables.





These exploratory steps were crucial in determining the most appropriate preprocessing techniques to be applied to both the categorical and numerical columns in the dataset.

From our initial screening, we identified a few issues:

- Age: Slightly right-skewed
- Balance: Zero-inflated (many 0 value dominate)
- HasCrCard, Exited: Slightly class imbalance, but because our dataset is large, it might not need handling

Therefore, initial data cleaning processes will be necessary before proceeding with further modeling and analysis.

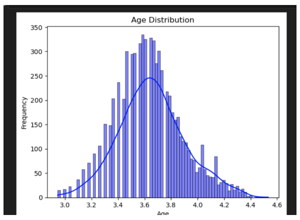
2.2 Data Preprocessing

2.2.1 Drop unuseful column / feature selection
We first use the .nunique() function to check how many unique values there are for each attribute. We discover that CustomerID has too many unique values, which means it is the identifier variables that are unique to each customer, and it doesn't have any predictive power, so we drop this column. Also, Surname is removed since it is not likely to influence churn prediction. This imply that we will select the rest features for model training and we will conduct further preprocessing on them.

2.2.2 Handle missing value
We handle missing values by using the. fillna() function. For numerical attributes, we fill the missing value with the average of that attribute in the training set, and for categorical attribute, we fill with the mode of that attribute in the training set.

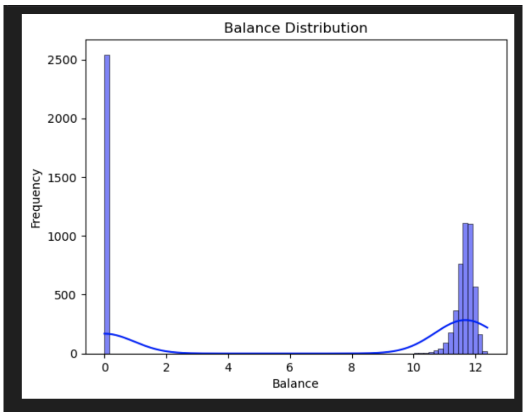
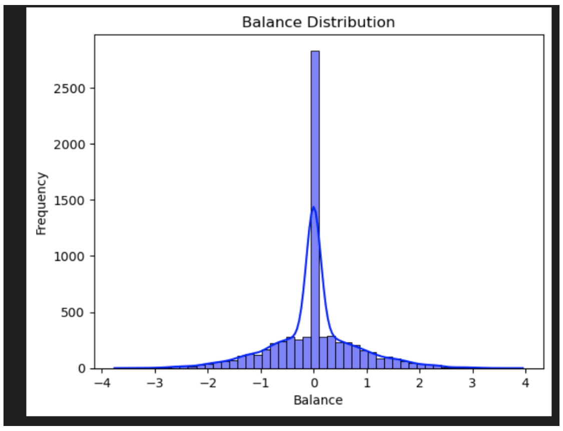
2.2.3 Remove duplicate records
We also removed any duplicate customer records to ensure we weren't training the model on redundant information.

2.2.4 Other handling
We do different data preprocessing for different models we use due to its different properties.

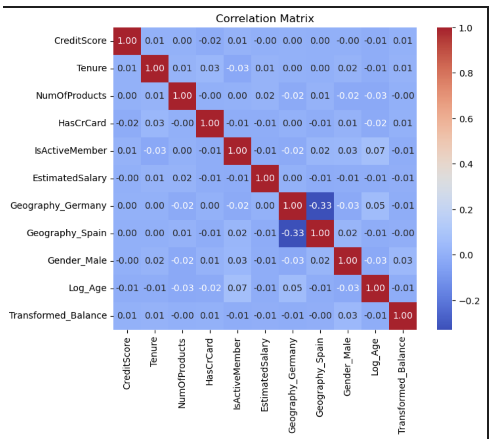
Attribute & its issue	Decision Tree	Logistic Regression	Gaussian Naive bayes classifier	Neural Network	Random Forest
Age: right-skew	No handling (Because split are based on thresholds, the distribution of value did not matter)	Apply log transformation to stabilise coefficients (After transform, the histogram distribution look like normal distribution) 		No handling	No handling
Balance: Zero-inflated		Instead of log transformation, we apply Yeo-johnson transformation on non-zero balances (pls check remark #1)		No handling	No handling
Handle categorical variable	One-hot encoding for geography and gender to convert the values into dummy columns. Gender is then replaced by a binary column (Gender_Male: True for Male, False for Female) Geography is converted into three dummy columns (Geography_France, Geography_Germany, Geography_Spain). And handing is not needed for HasCrCard, IsActiveMember because they are already represented in 0 and 1.				One-hot encoding used as in code (Geography, Gender, Age_Group)
Feature discretization	Not required				
Feature normalization (Scaled the numerical features to have a mean of zero and a standard deviation of one.)	Not required	Yes, use StandardScaler on ‘CreditScore', 'Tenure','NumOfProducts', 'EstimatedSalary', 'Log_Age','Transformed_Balance' to ensure that features with larger ranges don't dominate the model.	Not required (the model uses probability distributions instead of actual value)	Yes, use StandardScaler on all numerical features to ensure that features with larger ranges don't dominate the model.	Not required
Multicollinearity	Not required	Not required (no correlated	Not required	No handling	Not required

		features found, pls check remark #2)			

Remark #1: handing zero-inflation problem
At first, we try to apply log transformation on Balance, but it did not turn out to be normal distribution, so we tried Yeo-johnson transformation on non-zero balance

Remark #2: correlation matrix
Highly correlated features can negatively impact the performance of Logistic Regression models due to multicollinearity.
But from the correlation matrix, we did not find any highly correlated features, so no handling is required.

	Highly correlated features can negatively impact the performance of Logistic Regression models due to multicollinearity. But from the correlation matrix, we did not find any highly correlated features, so no handling is required.

3. Model Building
We tried both decision tree, logistic regression and Gaussian Naive Bayes classifier, for each we tried to use k-fold cross validation with 5 fold and GridSearchCV to find the best hyperparameter.

<u>3.1 Decision tree</u>

3.1.1 Idea explanation
The Decision Tree Classifier was chosen for its ability to model non-linear relationships and its interpretability.

Decision Trees are supervised learning models that recursively partitioning to split data into less impure subsets based on feature thresholds, aiming to maximize the information gain (evaluate by entropy reduction) in each split. They are interpretable but prone to overfitting without property pruning .

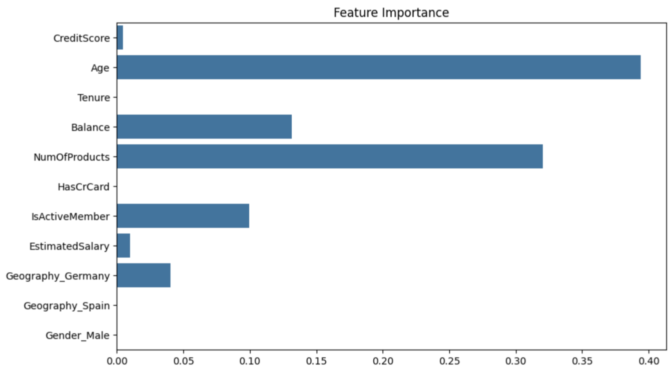
To avoid overfit, we use pre-pruning. We set the maximum depth in between 1 to 20, and the minimum samples required to split a node to be either 2, 5, or 10. We also limit the total number of leaf nodes to 100. To find the best hyperparameter combination, we use GridSearchCV, which involves training and validating the models with different hyperparameter values and selecting the combination that produced the best results.

After running the GridSearchCV with 5-fold, we find the best hyperparameter combination, with the max_depth is 8, max_leaf_nodes is 41 and min_samples_split is 2, which is a medium-size tree. This also gives the best cross-validation score of 0.8539, which we believe is high enough, but to further check whether this high score is due to overfit, we need to check with the testing set later and check the generalisation performance.

After evaluation on the model performance on the test set (detail to be elaborate in next session, 4.1), we build a final model using all data available and plot the tree.

3.1.2 Feature importance

We also checked for the feature importance. From the result, 'age' seems to be the biggest driving factor of whether a customer will churn, while Tenure and Gender is the smallest.



<u>3.2 Logistic regression</u>

3.2.1 Idea explanation

Logistic Regression was chosen for its efficiency and ability to provide probabilistic outputs.

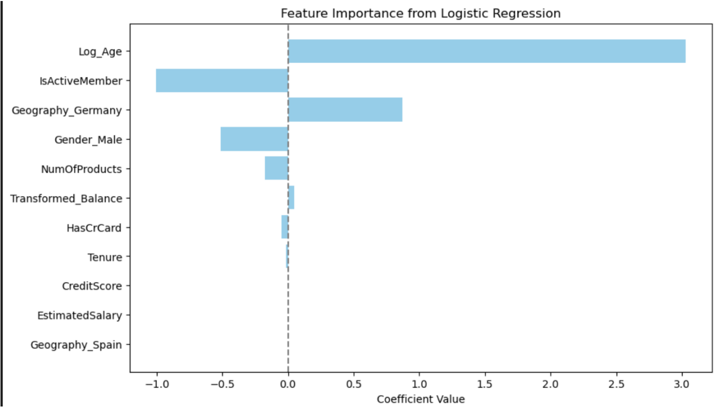
Logistic Regression is a linear classification algorithm that produces output values between 0 and 1. It estimates coefficients that maximize the likelihood of the observed data, capturing the relationship between the features and the target variable. Class membership can then be assigned using decision thresholds.

To avoid overfit, we train the model with L1 regularization, which means we penalise large coefficients, such that the model not only tries to minimize the error but also keeps the coefficients small. This helps to reduce the complexity of the model, and a simpler model is more likely to generalize well to unseen data. To achieve this, We use GridSearchCV to search for the best penalty hyperparameters C, and the resulting C is 0.25, which gives the best CV score of -0.1910 negative mean squared error, which imply that the model has effectively learned from the training data.

The GridSearchCV was configured to optimize the ROC-AUC score, a measure of the model's ability to discriminate between churned and non-churned customers.

3.2.2 Feature importance

From the result, 'Log_age', 'IsActiveNumber' seems to be the biggest driving factor of whether a customer will churn, while 'CreditScore' and 'EstimatedSalary', and 'Geography_Spain' is the smallest.



<u>3.3 Gaussian Naive Bayes classifier</u>

The Naive Bayes classifier is a supervised model used for classification tasks. It is based on Bayes' theorem, which calculates the probability of a class label (hypothesis) given the features of an example (evidence) . It assumes all features are conditionally independent given the class.

Multinomial Naive Bayes and Gaussian Naive Bayes are two types of the Naive Bayes classifier tailored for different types of data distributions. Multinomial is mostly used for discrete count data and it assumes that the features follow a multinomial distribution. In contrast, Gaussian Naive Bayes is suited for continuous data and assumes features are normally distributed.

Because our dataset contains a lot of numerical attributes, if we choose to use the Multinomial Naive Bayes as taught in the lab, it requires a lot of encoding (because it requires categorical input), which might lead to data loss. Therefore, we do further research, and learn to use the Gaussian Naive Bayes approach, which is another probabilistic classifier based on Bayes' theorem, but suitable for numerical, continuous data, and it assumes features follow a normal distribution.

The hyperparameters here is var_smoothing, which controls the variance smoothing applied to prevent division by zero (zero frequency problem). From GridSearchCV, the resulting var_smoothing is 1e-9, and the CV score is 0.79, which is relatively low compared with decision tree and logistic regression, implying that this algorithm might not work well on this dataset, or maybe due to the normal distribution assumption (not perfect transformation lead to the violation of this assumption).

3.4 Neural Network

Neural Network Architectures

We implemented five feed-forward neural network variants in Keras to benchmark against our Random Forest results. The baseline model consists of two hidden layers of 64 ReLU units each, followed by a sigmoid output. To test depth, the deep network adds two more 64-unit layers (four hidden layers total). To test width, the wide model replaces the baseline's 64-unit layers with a single 256-unit hidden layer. For regularization, we created a dropout variant by inserting 30 percent dropout after each hidden layer, and an L2-regularized variant by applying a kernel regularization factor of 1e-3 on each dense layer. All models were trained for 30 epochs with the Adam optimizer (learning rate = 0.001) and binary cross-entropy loss.

Results & Analysis

Across all architectures, training accuracy steadily climbs toward 0.89, but only the dropout and L2-regularized models sustain validation accuracy gains without overfitting. The baseline, deep, and wide models each plateau around 0.85 validation accuracy, with ROC AUCs of approximately 0.838, 0.833, and 0.838 respectively. In contrast, the dropout model achieves the highest generalization (val acc ≈ 0.867; AUC = 0.849), visibly closing the train/val gap in the epoch-wise accuracy curves. The L2-regularized model also improves over baseline (val acc ≈ 0.860; AUC = 0.841) by discouraging large weights, but does not match the dropout variant's robustness to overfitting.

Beyond this course, we could dive into adaptive learning-rate schedules—like cyclical, step-decay and cosine annealing—and implement them to observe their effects on convergence. We might investigate optimizers beyond Adam (e.g., RAdam, Lookahead or AdaBelief) to improve training stability and speed. Exploring normalization techniques (batch, layer) and advanced regularization methods (mixup, CutMix) could further boost generalization.

3.5 Random Forest

Random Forest is an ensemble learning method that combines the predictions of multiple decision trees to improve accuracy and robustness. We use this because it's effective in handling complex datasets, reducing the risk of overfitting, and providing reliable estimates of feature importance.

A Random Forest Classifier is trained to predict the target variable ('Exited'). RandomizedSearchCV is used to optimize the model's hyperparameters.

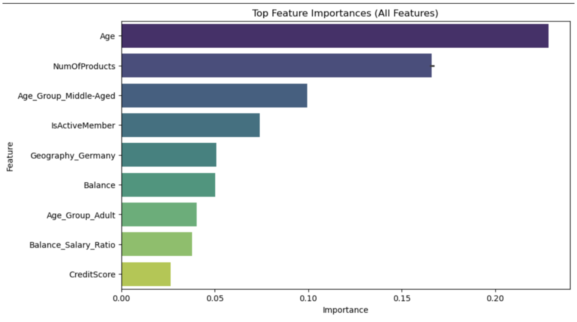
Originally, hyperparameter tuning was performed using GridSearchCV. However, this method takes lots of time on running the model (more than 10 minutes) as it involves an exhaustive search, where GridSearchCV evaluates every single combination of hyperparameter values specified in your param_grid. For each of these combinations, the method uses cross-validation, which multiplies the number of combinations by the number of cross-validation folds. This process results in a very high number of models being trained, making it computationally expensive, especially with large datasets or a wide range of hyperparameter values. In this original approach, the param_grid for the RandomForestClassifier included many values for n_estimators, max_depth, min_samples_split, and min_samples_leaf, leading to a large number of combinations and contributing to the computational expense.

To save time and improve model efficiency, we use RandomizedSearchCV to replace GridSearchCV. Because RandomizedSearchCV samples a fixed number of hyperparameter combinations from specified distributions or ranges. This method is more efficient because it explores a wider range of values within a set computational budget, allowing it to find effective hyperparameters more quickly than GridSearchCV. Additionally, the search space was reduced by narrowing the ranges for the hyperparameters, including n_estimators, max_depth, min_samples_split, and min_samples_leaf.

The hyperparameters explored include: `n_estimators`: The number of trees in the forest (values between 50 and 150). `max_depth`: The maximum depth of a tree (values between 3 and 7). `min_samples_split`: The minimum number of samples required to split an internal node (values between 2 and 5). `min_samples_leaf`: The minimum number of samples required to be at a leaf node (values between 1 and 3). `class_weight`: Balancing classes during training.
The RandomizedSearchCV process resulted in the following optimal parameter values: `n_estimators` = 103, `max_depth` = 7, `min_samples_leaf` = 2, `min_samples_split` = 4, and `class_weight` = None.

Then, we select the best performing Random Forest model, based on ROC AUC score.
Feature importances are also calculated and visualized, showing the relative contribution of each feature in the model's predictions. The most important feature is ‘age’, followed by ‘NumOfProducts’.

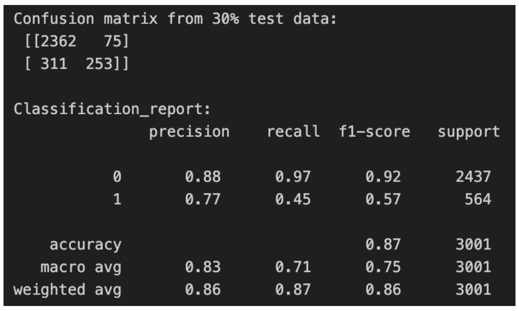
Feature importance



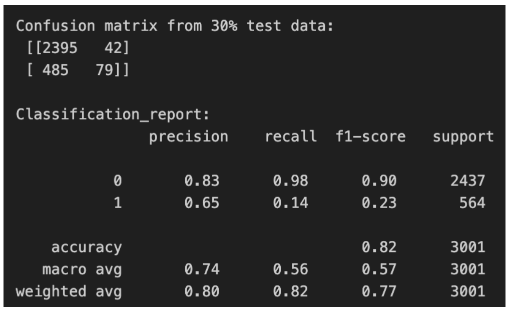
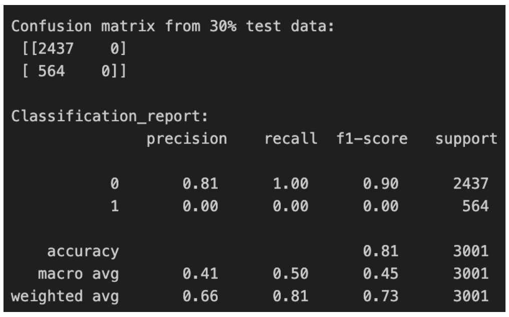
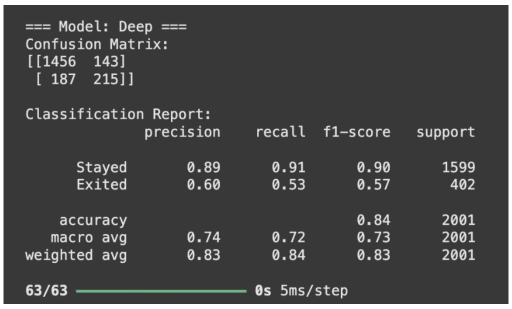
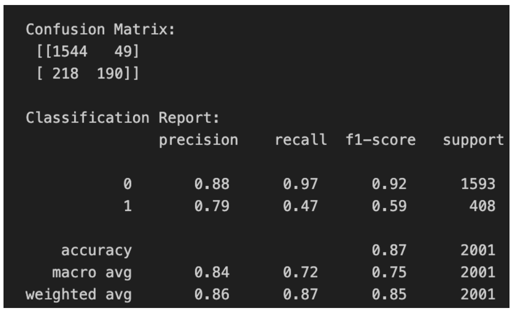
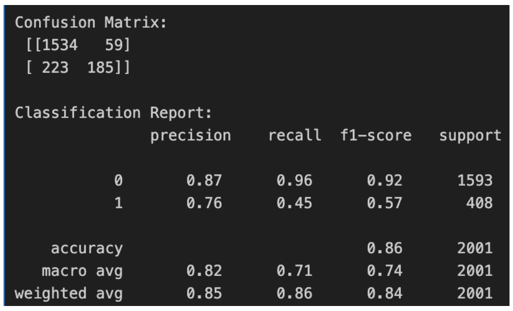
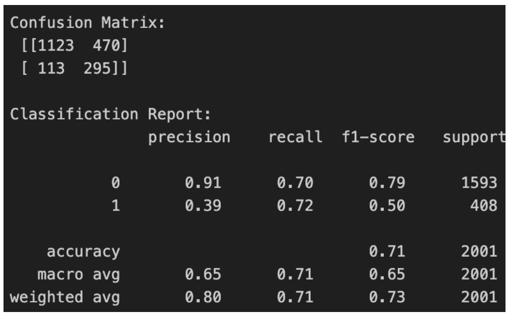
4. Performance Evaluation
While we used cross-validation as a performance measure as mentioned in the previous sessions, in this session we will focus on the evaluation on the hold-out test set.

Before evaluating the model on the test set, we first clean the test set, using the same method and value from the training set. Then, we measure the model performance using different performance measures, including accuracy, precision, recall, f1-score, AUC score, etc.

- **Accuracy:** How often the model correctly predicted churn.
- **AUC-ROC:** How well the model distinguishes between customers who churn and those who don't.
- **MSE:** The average squared difference between predicted and actual churn probabilities.
- **TPR:** The model's ability to correctly identify customers who actually churned.
- **Confusion Matrix:** A table that shows the number of correct and incorrect predictions for each class (churn or no churn).

	Accuracy	AUC	Confusion matrix and Classification report
4.1 Decision tree	0.8714	0.8439	

```
Confusion matrix from 30% test data:
 [[2362   75]
 [ 311  253]]

Classification_report:
               precision    recall  f1-score   support

           0       0.88      0.97      0.92      2437
           1       0.77      0.45      0.57       564

    accuracy                           0.87      3001
   macro avg       0.83      0.71      0.75      3001
weighted avg       0.86      0.87      0.86      3001
```

4.2 Logistic regression	0.8244	0.7662	
4.3 Gaussian Naive Bayes classifier	0.8121	0.6457	
4.4. Neural Network (Deep chosen)	0.8641	0.8470	
4.5 Random Forest	0.8666	0.8638	
4.6 Decision tree 2 *Ramark 3	0.8591	0.8529	
4.7 Logistic regression 2 *Ramark 3	0.7086	0.7845	

```
Confusion matrix from 30% test data:
 [[2395   42]
 [ 485   79]]

Classification_report:
               precision    recall  f1-score   support

           0       0.83      0.98      0.90      2437
           1       0.65      0.14      0.23       564

    accuracy                           0.82      3001
   macro avg       0.74      0.56      0.57      3001
weighted avg       0.80      0.82      0.77      3001
```

```
Confusion matrix from 30% test data:
 [[2437    0]
 [ 564    0]]

Classification_report:
               precision    recall  f1-score   support

           0       0.81      1.00      0.90      2437
           1       0.00      0.00      0.00       564

    accuracy                           0.81      3001
   macro avg       0.41      0.50      0.45      3001
weighted avg       0.66      0.81      0.73      3001
```

```
=== Model: Deep ===
Confusion Matrix:
[[1456  143]
 [ 187  215]]

Classification Report:
              precision    recall  f1-score   support

      Stayed       0.89      0.91      0.90      1599
      Exited       0.60      0.53      0.57       402

    accuracy                           0.84      2001
   macro avg       0.74      0.72      0.73      2001
weighted avg       0.83      0.84      0.83      2001

63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

```
Confusion Matrix:
 [[1544   49]
 [ 218  190]]

Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.97      0.92      1593
           1       0.79      0.47      0.59       408

    accuracy                           0.87      2001
   macro avg       0.84      0.72      0.75      2001
weighted avg       0.86      0.87      0.85      2001
```

```
Confusion Matrix:
 [[1534   59]
 [ 223  185]]

Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.96      0.92      1593
           1       0.76      0.45      0.57       408

    accuracy                           0.86      2001
   macro avg       0.82      0.71      0.74      2001
weighted avg       0.85      0.86      0.84      2001
```

```
Confusion Matrix:
 [[1123  470]
 [ 113  295]]

Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.70      0.79      1593
           1       0.39      0.72      0.50       408

    accuracy                           0.71      2001
   macro avg       0.65      0.71      0.65      2001
weighted avg       0.80      0.71      0.73      2001
```

For the ROC curve graphs, please check the ipynb code output.

*Ramark 3:
To further improve the model performances, we tried to do the decision tree and logistic regression model training again, but with different data preprocessing methods and with an extra feature selection.

	4.1 Decision tree	4.6 Decision tree 2
Handling missing value	Fill in mean for numerical variables, mode for categorical variable	Fill in median for numerical variables, mode for categorical variable
Feature Engineering	N/A	- Feature discretization on Age by binning them with [0, 25, 40, 60, 100], and labels ['Young', 'Adult', 'Middle-Aged', 'Senior']) - Create new feature:Balance_Salary_Ratio = Balance/ EstimatedSalary + 1e-6
Encoding	One-hot encoded Geography and Gender	One-hot encoded Geography, Gender, and Age_Group
Final Feature Set	['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary','Geography_Germany','Geography_Spain','Gender_Male']	['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary', 'Geography_Germany', 'Geography_Spain', 'Gender_Male', 'Age_Group_Adult', 'Age_Group_Middle-Aged', 'Age_Group_Senior', 'Balance_Salary_Ratio']
Algorithm Learning and Hyperparameter Tuning	Implements GridSearchCV focusing on max_depth, min_samples_split, and max_leaf_nodes.	Uses GridSearchCV based on criterion ('gini', 'entropy'), 'max_depth', 'min_samples_split', 'min_samples_leaf', 'class_weight': ['balanced', None] } (class_weight: Adjusts the weights associated with classes, allowing the model to handle imbalanced datasets more effectively.)
Cross-Validation	KFold with 5 splits and evaluates using accuracy.	Uses 3 folds to evaluate based on ROC AUC.
Feature selection	N/A	Include embedded feature selection method, with maximum features = 5
Result in different performance	Higher Accuracy: 0.8714 Lower AUC : 0.8439	Lower Accuracy: 0.8591 Higher AUC : 0.8529
Reason on relative performance	- uses mean to fill missing numerical values, which can be sensitive to outliers. - Employs minimal transformations, try to retain information in the original features .- Uses a simpler model tuning approach with fewer hyperparameters. This may allow the model to generalize better, leading to higher accuracy. - For the CV, it focuses on evaluating based on accuracy, which is a straightforward measure of performance but may not capture the model's effectiveness in handling class imbalances.	- Uses the median to fill missing numerical values → more robust to outliers. - Binning for Age can simplify the model but may lose some nuances in age-related patterns. The bins might not represent the data effectively. - Explores a broader set of hyperparameters, including different criteria (gini, entropy) and class weights. While this can optimize the model for certain metrics (like AUC), it might also introduce complexity that can lead to overfitting or instability, particularly with fewer folds in cross-validation. - For the CV, it evaluates using ROC AUC, which is more informative in assessing model performance on imbalanced datasets.

Similarly, the Logistic regression 2 has similar data preprocessing like Decision tree 2, and an extra feature selection using embedded method, with maximum features = 5..

5. Conclusion

5.1 Summary
The objective of this project is to predict customer churning, the models we built are all able to perform classification tasks on the target variable "Exited" (Ture = Exited, False = Not exited). The conclusions drawn from the built models will help us identify at-risk customers, which allow companies to tailor retention strategies to improve customer loyalty.

5.2 Reason behind performance difference between algorithms
Among all the models we trained, the Random Forest model achieved the highest performance, followed closely by the Decision Tree, while the Gaussian Naive Bayes classifier performed the worst in terms of both accuracy and AUC.

Random Forest achieved the best overall performance. This is because it is an ensemble learning method that builds multiple decision trees and aggregates their results to make a final prediction. This reduces the risk of overfitting, which is common in a single decision tree, and increases generalization capability. Random Forest is particularly powerful for capturing complex, non-linear relationships and handling high-dimensional data with mixed feature types. It achieved the highest accuracy and AUC among all models because of its robustness and ability to reduce variance.

Decision Tree, although slightly less powerful than Random Forest, also performed very well (accuracy = 0.8714, AUC = 0.8439). This is because it is able to model non-linear patterns and is highly interpretable. Our implementation controlled for overfitting by setting a maximum depth, which helped maintain a good balance between bias and variance. The tree structure is well-suited for datasets with complex relationships between features, contributing to its high performance.

Logistic Regression performed moderately, better than Naive Bayes but worse than the tree-based models. It assumes a linear relationship between the independent variables and the log-odds of the target variable, which limits its ability to capture non-linear interactions. Nonetheless, it works well for binary classification problems, which explains why its accuracy is still reasonable. However, its performance is restricted by its simplicity and the assumption of linear separability.

Gaussian Naive Bayes showed the lowest performance (accuracy = 0.8121, AUC = 0.6457). This is largely due to its strong assumption that all features are conditionally independent given the class label, which is rarely true in real-world datasets. Additionally, it assumes that feature distributions follow a Gaussian distribution, which may not hold in our dataset. These unrealistic assumptions lead to a loss of predictive power, especially in cases where feature dependencies and complex interactions are present.

5.2 Challenge encountered and learnings:

A challenge encountered was the computational time associated with GridSearch, which was slow due to the large parameter space. This was mitigated by reducing the size of the grid and using the n_jobs=-1 parameter to parallelize the search process.

5.3 Unsolved problem / Future work:

We recognize the class imbalance in the target variable (Exited). The dataset likely has a disproportionate number of non-churned customers compared to churned ones. In future, techniques such as SMOTE (Synthetic Minority Over-sampling Technique), adjusting class weights, or using evaluation metrics like the F1-score would be employed to handle this issue.

5.4 Implications for banking companies

5.4.1 Make use of decision tree graph

We ranked the feature importance for the algorithms, comparing all, we identified “age” as the most important feature that has the highest predictive power, so banks can further investigate the age impact of the churning prediction.

Out of all trained models, decision trees have the highest performance and it is the most interpretable. So the bank can make reference to the tree path graph to explore business insight. For example, from the tree graph, we can see that customers who are below 41.5 years old, and own less than 1.5 products on average, and with balance less than 1884.345, are very likely to churn, so bank might need to pay extra attention to future customer who have similar attributes value. Similarly, banks can base on other node paths that arrive at the leaf with ‘Existed’ label with high probability, to make targeted retention strategies.

5.4.2 Examples of tree interpretation and business strategy

Customers aged under 30 with limited products and lower balances often represent early-career individuals with fewer financial commitments. Banks should seize this opportunity to build loyalty and increase their product engagement. By offering cash rewards for consistent deposits and long-term savings plans, companies can establish valuable relationships early while increasing future switching costs.

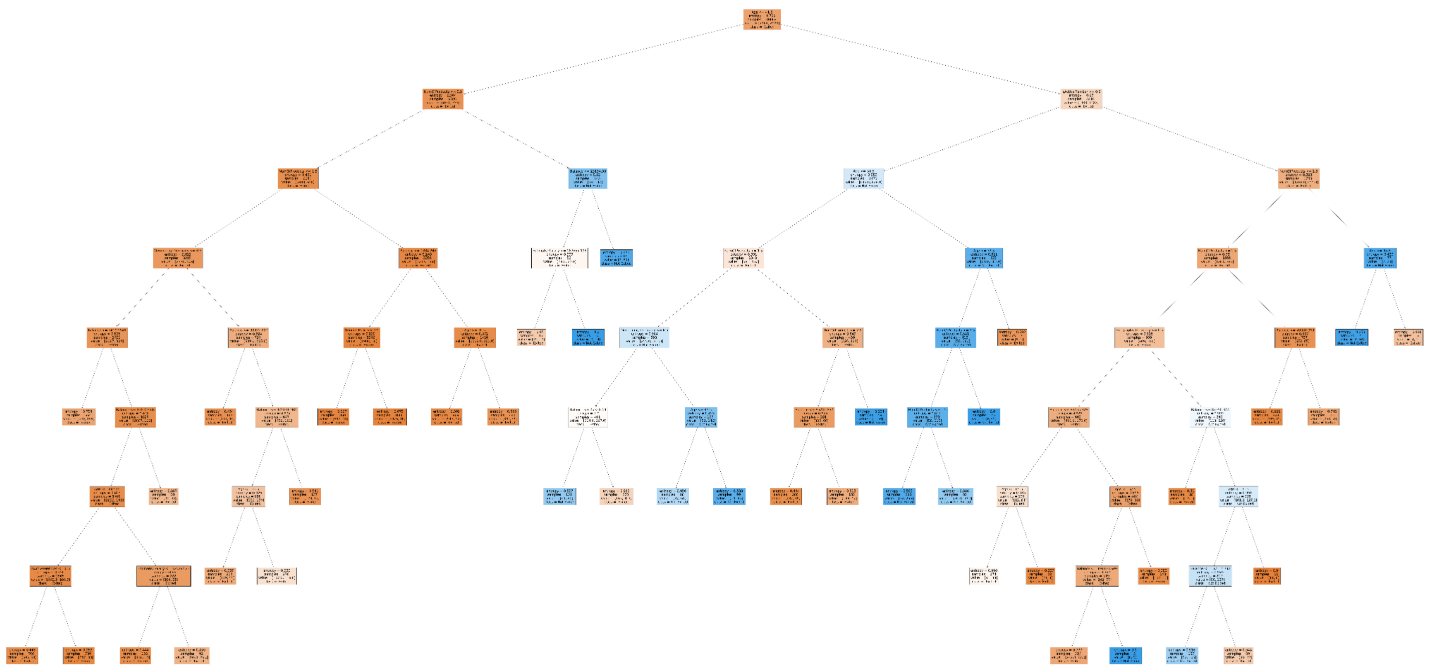
As customers enter the age of 31 to 41, their financial needs become more complex due to different major life events. Therefore, those with lower balances and minimal product engagement are particularly vulnerable, due to a high sensitivity in fees and a lack of established banking relationships. For these customers, banks should develop transitional support programs, such as offering a premium credit card with no annual fee when a customer’s salary deposit exceeds a threshold, providing loyalty interest rate boosts for every additional product purchase, and tiered mortgage rates for long-term customers. These strategies can address customers’ pain points while encouraging deeper product integration.

In conclusion, banks should utilize predictive models to identify potential customer churn. By examining the demographics of the high-risk customers, such as their age and product count, banks can prioritize outreach efforts and tailor retention strategies to each segment's specific needs.

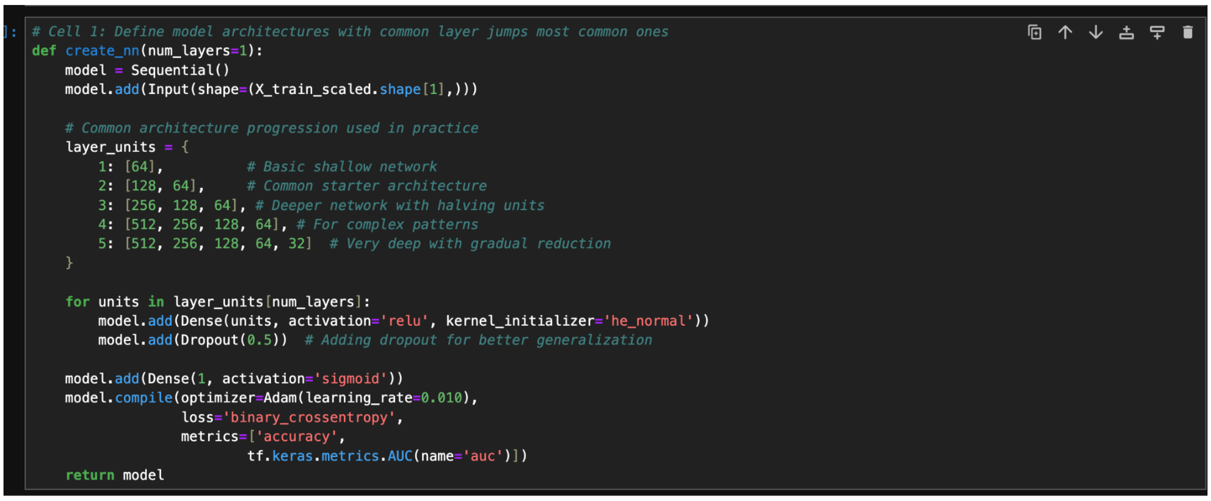
6. References

7. Appendix

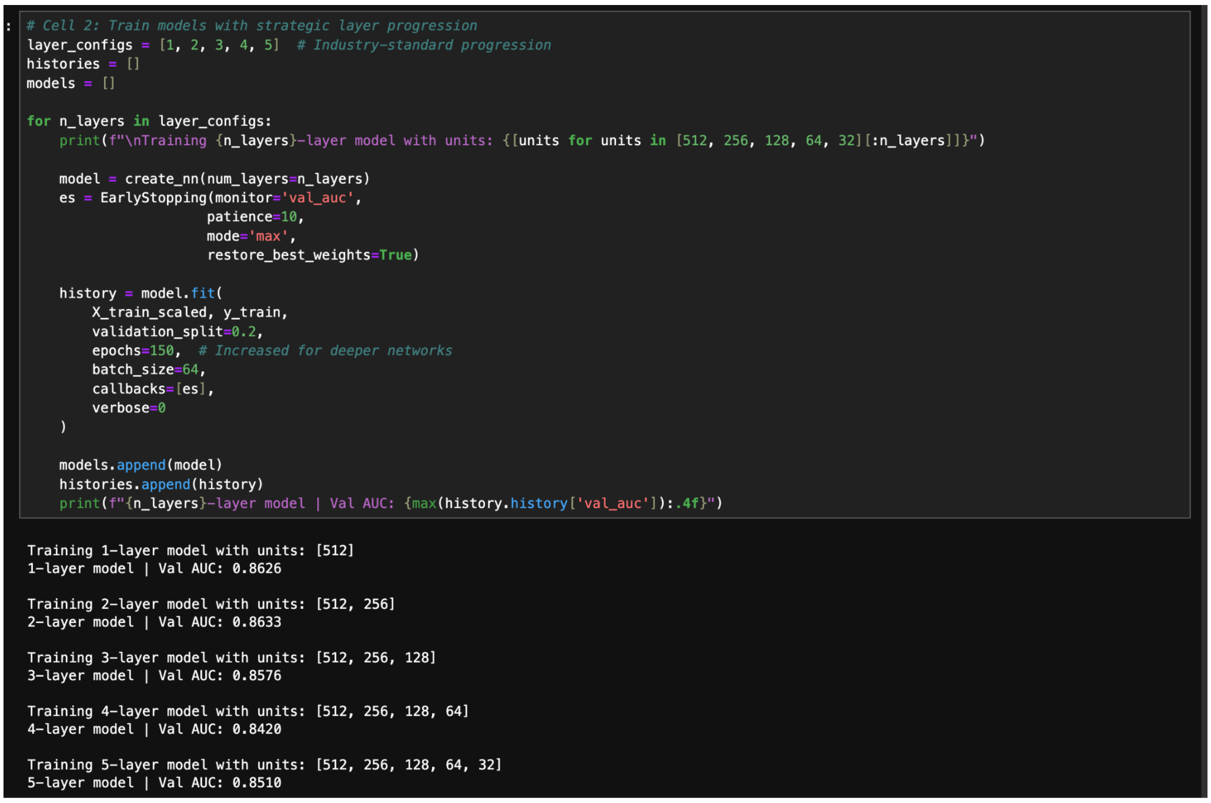
Decision tree



Neural Network Extra:

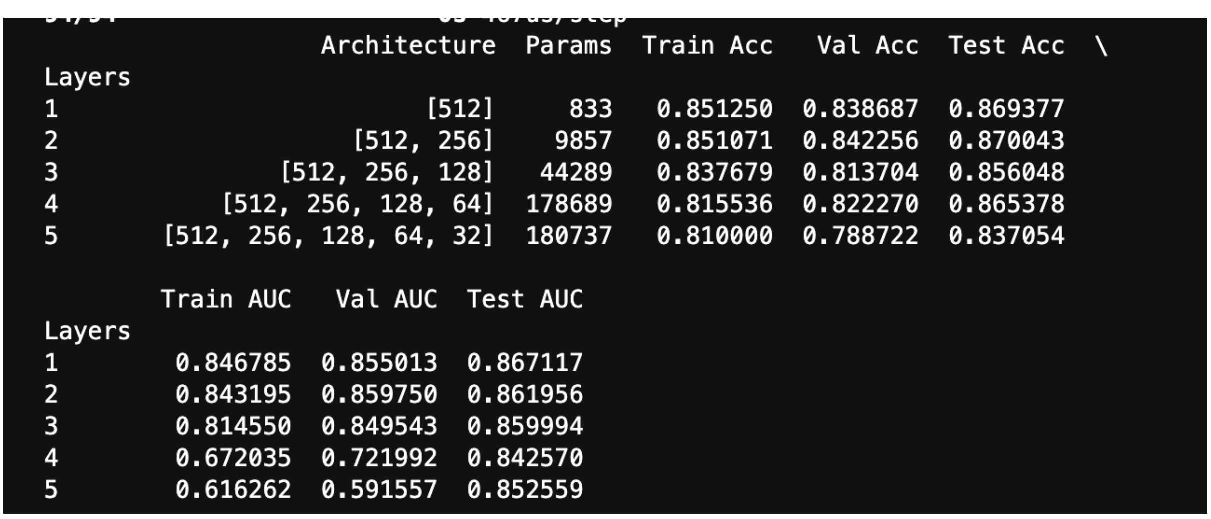


```
# Cell 1: Define model architectures with common layer jumps most common ones
def create_nn(num_layers=1):
    model = Sequential()
    model.add(Input(shape=(X_train_scaled.shape[1],)))

    # Common architecture progression used in practice
    layer_units = {
        1: [64],           # Basic shallow network
        2: [128, 64],      # Common starter architecture
        3: [256, 128, 64], # Deeper network with halving units
        4: [512, 256, 128, 64], # For complex patterns
        5: [512, 256, 128, 64, 32]  # Very deep with gradual reduction
    }

    for units in layer_units[num_layers]:
        model.add(Dense(units, activation='relu', kernel_initializer='he_normal'))
        model.add(Dropout(0.5))  # Adding dropout for better generalization

    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=0.010),
                  loss='binary_crossentropy',
                  metrics=['accuracy',
                          tf.keras.metrics.AUC(name='auc')])
    return model
```

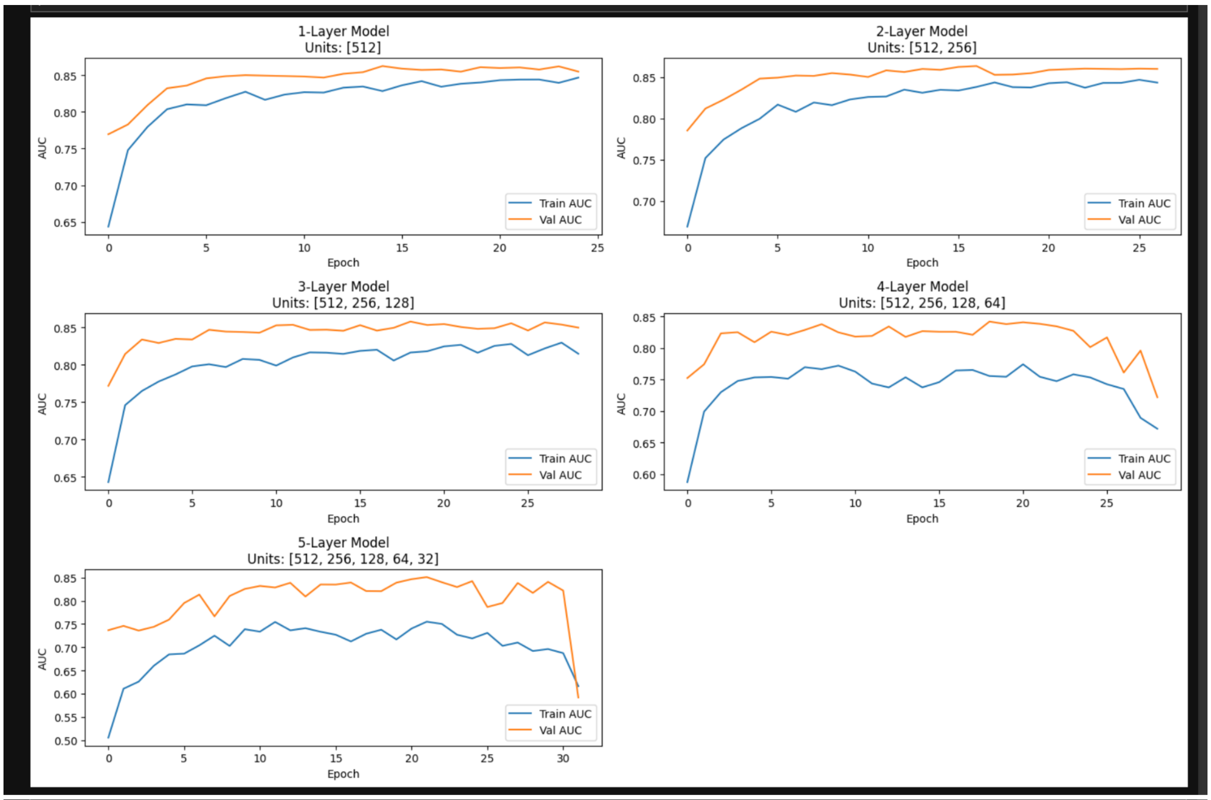


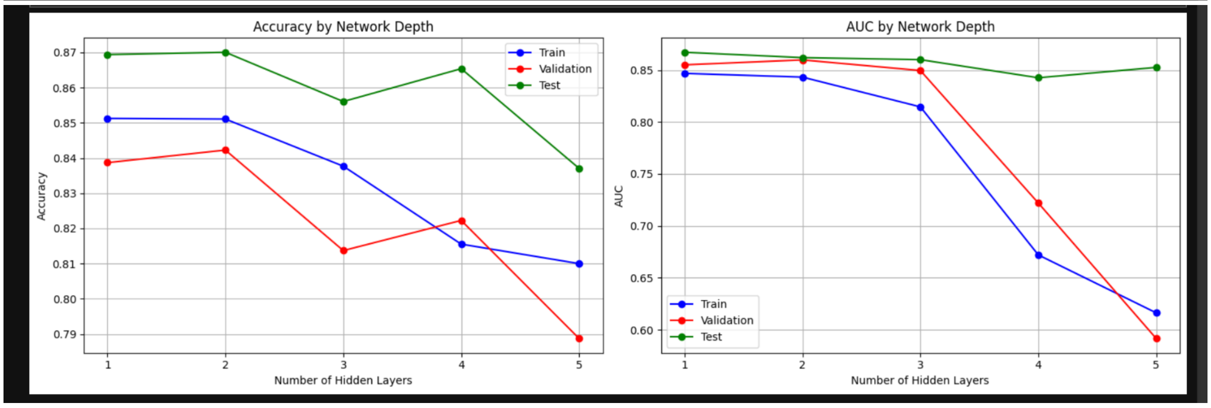
```python
# Cell 2: Train models with strategic layer progression
layer_configs = [1, 2, 3, 4, 5]  # Industry-standard progression
histories = []
models = []

for n_layers in layer_configs:
    print(f"\nTraining {n_layers}-layer model with units: {[units for units in [512, 256, 128, 64, 32][:n_layers]]}")

    model = create_nn(num_layers=n_layers)
    es = EarlyStopping(monitor='val_auc',
                      patience=10,
                      mode='max',
                      restore_best_weights=True)

    history = model.fit(
        X_train_scaled, y_train,
        validation_split=0.2,
        epochs=150,  # Increased for deeper networks
        batch_size=64,
        callbacks=[es],
        verbose=0
    )

    models.append(model)
    histories.append(history)
    print(f"{n_layers}-layer model | Val AUC: {max(history.history['val_auc']):.4f}")
```

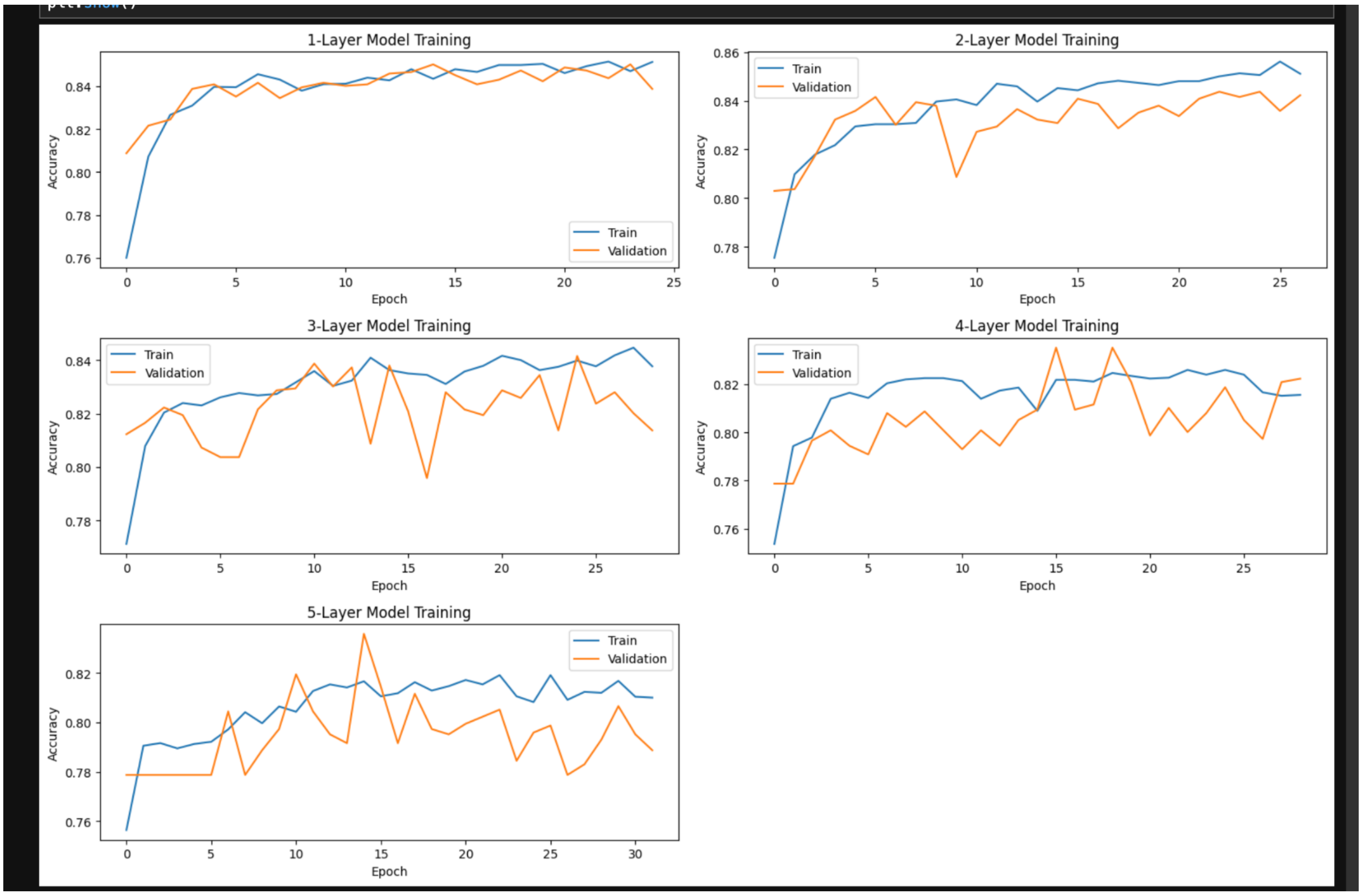
```
Training 1-layer model with units: [512]
1-layer model | Val AUC: 0.8626

Training 2-layer model with units: [512, 256]
2-layer model | Val AUC: 0.8633

Training 3-layer model with units: [512, 256, 128]
3-layer model | Val AUC: 0.8576

Training 4-layer model with units: [512, 256, 128, 64]
4-layer model | Val AUC: 0.8420

Training 5-layer model with units: [512, 256, 128, 64, 32]
5-layer model | Val AUC: 0.8510
```



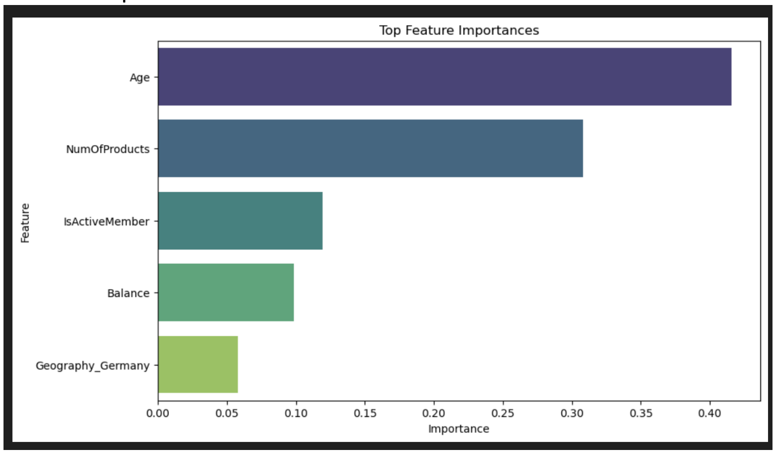
```
                   Architecture  Params  Train Acc   Val Acc  Test Acc  \
Layers
1                         [512]     833   0.851250  0.838687  0.869377
2                    [512, 256]    9857   0.851071  0.842256  0.870043
3               [512, 256, 128]   44289   0.837679  0.813704  0.856048
4           [512, 256, 128, 64]  178689   0.815536  0.822270  0.865378
5       [512, 256, 128, 64, 32]  180737   0.810000  0.788722  0.837054

        Train AUC   Val AUC  Test AUC
Layers
1        0.846785  0.855013  0.867117
2        0.843195  0.859750  0.861956
3        0.814550  0.849543  0.859994
4        0.672035  0.721992  0.842570
5        0.616262  0.591557  0.852559
```


1-Layer Model
Units: [512]
2-Layer Model
Units: [512, 256]
3-Layer Model
Units: [512, 256, 128]
4-Layer Model
Units: [512, 256, 128, 64]
5-Layer Model
Units: [512, 256, 128, 64, 32]
AUC
Epoch
Train AUC
Val AUC

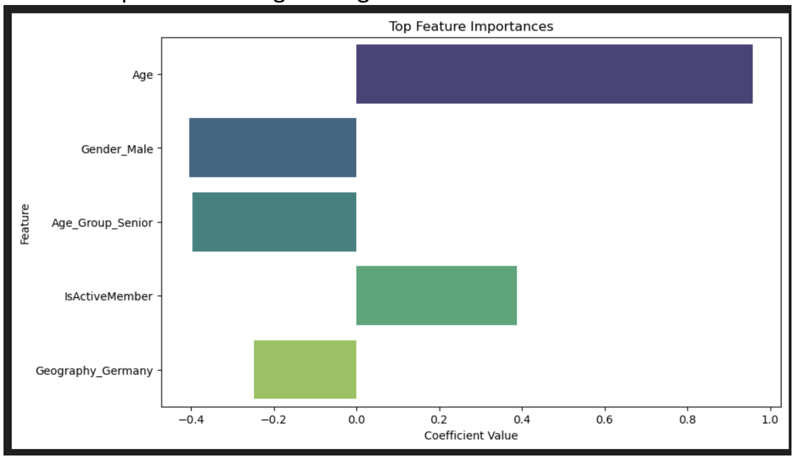

Accuracy by Network Depth
Accuracy
Number of Hidden Layers
AUC by Network Depth
AUC
Train
Validation
Test



Feature importance of decision tree with feature selection



Feature importance of logistic regression with feature selection


Top Feature Importances
Age
Gender_Male
Age_Group_Senior
IsActiveMember
Geography_Germany
Feature
Coefficient Value