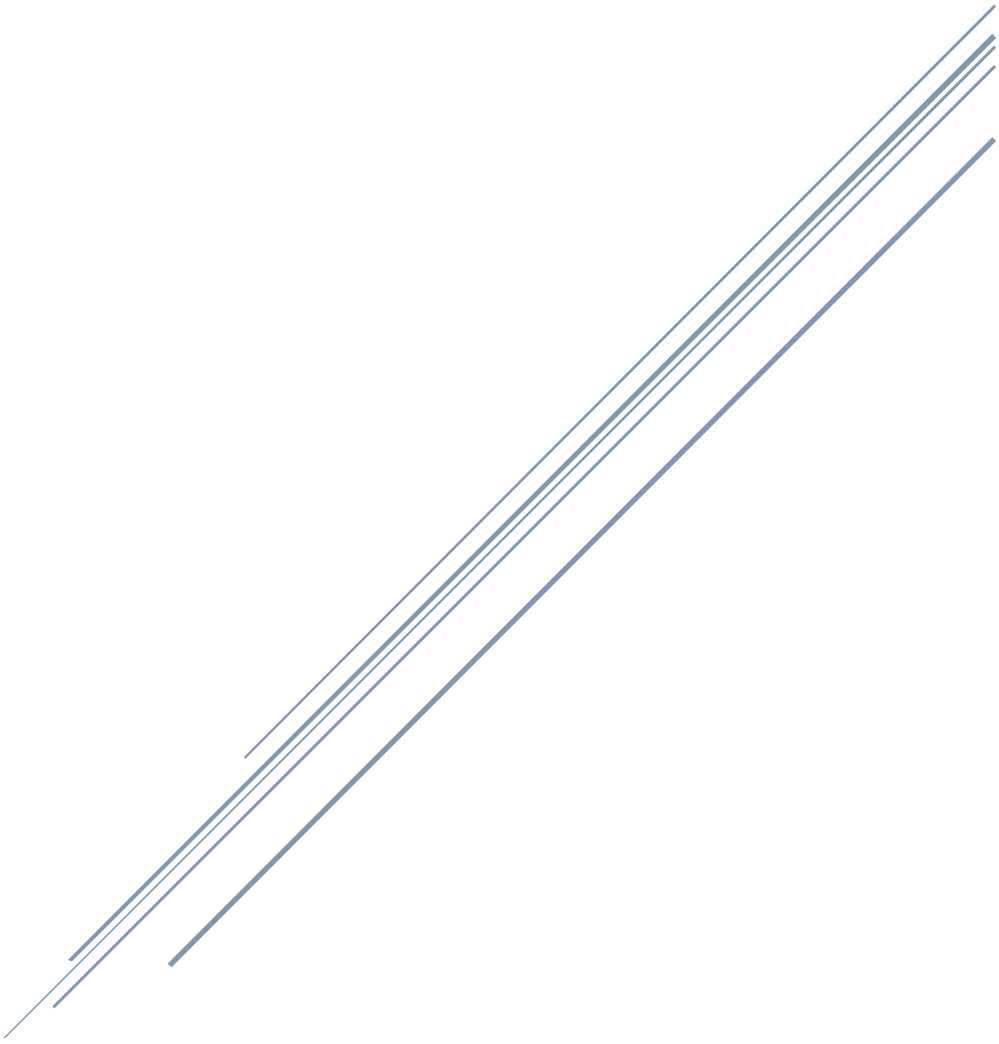
# MATH569 STATISTICAL LEARNING

## Summative Course Project (Module 9)

### Customer Churn Prediction in Telecommunications: From Logistic Regression to Deep Neural Networks



Code in Jupyter Notebook is available at:

Math569_Project/Final_Project_Math569_Zixiang_Zhou.ipynb at master · yz2001zzx/Math569_Project

Zixiang Zhou
July 20 2025

# Table of Contents

# List of Tables



# List of Figures

## Executive Summary

This project focuses on predicting customer churn in the telecom industry using the Telco Customer Churn dataset. Multiple machine learning models were visited and compared, including Logistic Regression, SVM, Random Forest, and multiple variants of Deep Neural Networks (DNNs). The best-performing model was a deep neural network with dropout, batch normalization, and early stopping, which achieved 81.1% test accuracy, 79.7% precision, 82.4% recall, and an F1-score of 81.0%. These results are on par with those reported in recent literature on the same dataset, showing that proper data processing, appropriate model selection, and effective training and hyperparameter tuning can yield strong predictive performance. This process also served as a valuable experiential learning opportunity within the course.

Among these metrics, recall is the most important to identify actual churners correctly as actual churners represent the minority class in the dataset. Feature importance analysis from interpretable models showed *Tenure Months, Monthly Charges*, and *Internet Service Fiber optic* as the most influential predictors. These features align with practical business intuition, pointing to loyalty, cost sensitivity, and service expectations as key drivers of churn. The findings can guide telecom providers in designing targeted and data-driven retention strategies.

# 1. Introduction

## 1.1 Background

Customer churn is an important business problem for subscription-based businesses such as telecommunications. When customers discontinue service, organizations not only lose recurring revenue but also incur additional costs related to customer acquisition and retention strategies. As a result, the ability to predict customer churn in advance can provide substantial value, enabling telecom providers to intervene early and improve customer retention outcomes. In this project, we focus on developing predictive models to identify customers who are likely to churn based on the Telco Customer Churn (IBM Version) dataset available on Kaggle.

## 1.2 Problem Statement

We divide the goal of this project into three main objectives. First and foremost, we aim to investigate whether customer churn can be predicted using structured data collected by a telecom company with adequate prediction accuracy. Specifically, we seek to determine whether at least 60% accuracy can be achieved by leveraging customer account information, demographic characteristics, and service usage patterns.

Moreover, identifying the most influential features that contribute to a customer's decision to leave is crucial for businesses to take prescriptive actions, such as enabling targeted retention strategies. Last but not least, we will compare several machine learning models in terms of their predictive performance and computational efficiency. This comparison will help identify models that are not only effective but also practical for real-world deployment in a production environment.

## 1.3 Dataset Description

The dataset used in this project can be accessed via:

https://www.kaggle.com/datasets/yeanzc/telco-customer-churn-ibm-dataset

which has 7,043 customer records and over 33 variables, capturing a wide range of information including demographic characteristics, service usage behavior, account details, and billing history.

The target variable, Churn, is a binary label which indicates whether a customer has stopped using the service ("Yes") or not ("No"). Each row represents a unique customer and includes demographic attributes, service subscriptions, geographic location, account and billing information, as well as engineered fields like churn score and customer lifetime value (CLTV). We can split these variables into the following categories:

Identifiers

| **Feature** | **Description** |
|---|---|
| customerID | Unique ID for each customer |
| Count | Constant count field for aggregation/reporting |

Geographic Information

| **Feature** | **Description** |
|---|---|
| Country | Country of primary residence |
| State | State or province of residence |
| City | City of residence |
| Zip Code | Postal/ZIP code |
| Lat Long | Concatenated latitude and longitude |
| Latitude | Latitude of customer's location |
| Longitude | Longitude of customer's location |

Demographic Information

| **Feature** | **Description** |
|---|---|
| Gender | Customer's gender (Male, Female) |
| Senior Citizen | Whether the customer is 65 or older (Yes, No) |
| Partner | Whether the customer has a partner (Yes, No) |
| Dependents | Whether the customer has dependents (children, parents, etc.) (Yes, No) |

Service Subscriptions

| **Feature** | **Description** |
|---|---|
| Phone Service | Subscribes to home phone service (Yes, No) |
| Multiple Lines | Subscribes to multiple telephone lines (Yes, No) |
| Internet Service | Internet type: No, DSL, Fiber Optic, or Cable |
| Online Security | Subscribes to additional online security service (Yes, No) |
| Online Backup | Subscribes to online backup service (Yes, No) |
| Device Protection | Subscribes to device protection plan (Yes, No) |
| Tech Support | Subscribes to technical support with reduced wait times (Yes, No) |

| | |
|---|---|
| Streaming TV | Uses internet to stream TV programs (Yes, No) |
| Streaming Movies | Uses internet to stream movies (Yes, No) |

Account and Billing Information

| **Feature** | **Description** |
|---|---|
| Tenure Months | Number of months the customer has stayed with the company |
| Contract | Current contract type (Month-to-Month, One Year, Two Year) |
| Paperless Billing | Whether the customer uses paperless billing (Yes, No) |
| Payment Method | How the customer pays the bill (Bank Withdrawal, Credit Card, Mailed Check) |

Financial Information

| **Feature** | **Description** |
|---|---|
| Monthly Charge | Total monthly charge for all services |
| Total Charges | Total lifetime charges by the end of the recorded quarter |

Churn Information

| **Feature** | **Description** |
|---|---|
| Churn Label | Indicates if customer churned (Yes, No) |
| Churn Value | Binary version of Churn Label (1 = churned, 0 = retained) |
| Churn Score | Score from 0 to 100 estimating churn likelihood (generated by SPSS model) |
| Churn Reason | Text description of why the customer churned (if applicable) |

Engineered Feature

| **Feature** | **Description** |
|---|---|
| CLTV | Predicted Customer Lifetime Value (higher = more valuable) |

## 2. Related Work

Customer churn prediction is a classical binary classification problem on which many studies have been examined. It serves not only as a practical exercise for students, like myself, to apply statistical techniques learned in class, but also as a real-world application with significant business value, particularly in subscription-based industries such as telecommunications and streaming services. Before attempting my approach to the dataset, the following briefly summarizes key findings from recent literature.

A wide range of classification algorithms have been applied to churn prediction problems with varying levels of success. In one study, Random Forest, J48 Decision Tree, Decision Stump, Naïve Bayes, and Logistic Regression were tested across two real-world telecom datasets with 64,000 and 3,333 records, respectively [1]. The results showed that Random Forest outperformed the other models on one dataset, while J48 and the Attribute Selected Classifier performed better on the other. This highlights how model effectiveness can vary depending on dataset characteristics and feature composition.

Another study evaluated Logistic Regression, Random Forest, and Gradient Boosting on telecom churn data, finding that Gradient Boosting delivered the best overall performance, achieving an accuracy close to 80% after tuning [2]. The most influential predictors identified were “Contract,” “tenure,” and “Monthly Charges”, all of which are also present in our dataset. This allows us to validate whether they are equally significant in our own diagnostic analysis.

Ensemble techniques have also proven effective in large-scale datasets. A dual-ensemble approach combining Random Forest and AdaBoost was proposed for a dataset with over 900,000 customer records [3]. The model achieved impressive performance, with a recall of 93% and precision of 99%, outperforming both standalone classifiers and neural networks. These findings underscore the potential of ensemble models, particularly in handling imbalanced data.

Beyond traditional feature-based modeling, some studies have explored alternative data representations. For example, Óskarsdóttir et al. [4] incorporated customer interaction networks and applied link-based logistic regression combined with collective inference. Their results demonstrated that using relational data can substantially improve classification performance over purely attribute-based approaches.

Finally, a more focused comparison between Logistic Regression, SVM, and Random Forest found that Logistic Regression slightly outperformed the others, achieving an accuracy of approximately 79% [5]. The authors emphasized the interpretability and speed of Logistic Regression, making it a reliable baseline for binary classification. In this project, I will revisit all three models to examine whether the same conclusions hold true on our dataset.

*Table 1- Summary of Related Work*

| **Ref** | **Authors** | **Algorithms Evaluated** | **Dataset** | **Key Findings** | **Relevance to This Project** |
|---|---|---|---|---|---|
| [1] | Ullah et al. | RF, J48, Decision Stump, Naïve Bayes, LR | Two sets: 64,000 and 3,333 records | RF and J48 perform differently across datasets; performance depends on dataset characteristics | Highlights model variability depending on data |

| | | | | | |
|---|---|---|---|---|---|
| [2] | Nnaji & Nwodo | Logistic Regression, RF, Gradient Boosting | Not specified | GB achieved ~80% accuracy; Contract, tenure, Monthly Charges were top predictors | Enables validation of same feature importance |
| [3] | Zhou et al. | RF, AdaBoost, Neural Networks | 900,000+ entries | RF-AdaBoost hybrid model yielded 93% recall, 99% precision | Demonstrates power of ensembles on large datasets |
| [4] | Óskarsdóttir et al. | Link-based Logistic Regression + Collective Inference | Not specified | Incorporating network data improves performance beyond attribute-only models | Suggests possible improvement using relationship data |
| [5] | Nurtriana et al. | Logistic Regression, SVM, Random Forest | 7,043 records (the same Kaggle dataset as ours) | Logistic Regression was the most accurate (~79%) and interpretable | We will test the same models for comparison |

## 3. Exploratory Data Analysis

Before conducting exploratory data analysis, I began by validating the dataset, specifically checking the data type of each feature (column). One feature that immediately stood out was 'Total Charges', which was incorrectly stored as an object (string) type. Since 'Total Charges' is a numerical variable and conceptually aligned with 'Monthly Charges', it should be converted to float64 to ensure consistency and proper numerical analysis.

*Table 2- Summary of Datatypes in the Dataset before Processing*

| **Column** | **Type** | **Column** | **Type** | **Column** | **Type** | **Column** | **Type** |
|---|---|---|---|---|---|---|---|
| CustomerID | object | Phone Service | object | Gender | object | Contract | object |
| Count | int64 | Multiple Lines | object | Senior Citizen | object | Paperless Billing | object |
| Country | object | Internet Service | object | Partner | object | Payment Method | object |
| State | object | Online Security | object | Dependents | object | Monthly Charges | float64 |
| City | object | Online Backup | object | Tenure Months | int64 | Total Charges | **object** |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Zip Code | int64 | Device Protection | object | Churn Label | object | Churn Value | int64 |
| Lat Long | object | Tech Support | object | Churn Score | int64 | CLTV | int64 |
| Latitude | float64 | Streaming TV | object | Churn Reason | object | Longitude | float64 |

The 'CustomerID' column is a unique identifier for each customer and does not have any predictive value related to churn. Therefore, it can be safely dropped before performing exploratory analysis. Similarly, the 'Count' column contains a constant value of 1 for all entries and is intended solely for aggregation or reporting purposes, should also be removed from our dataset.

Through the following process, we aim to clean and refine the dataset and select meaningful predictors to build better models that yield improved predictive accuracy.

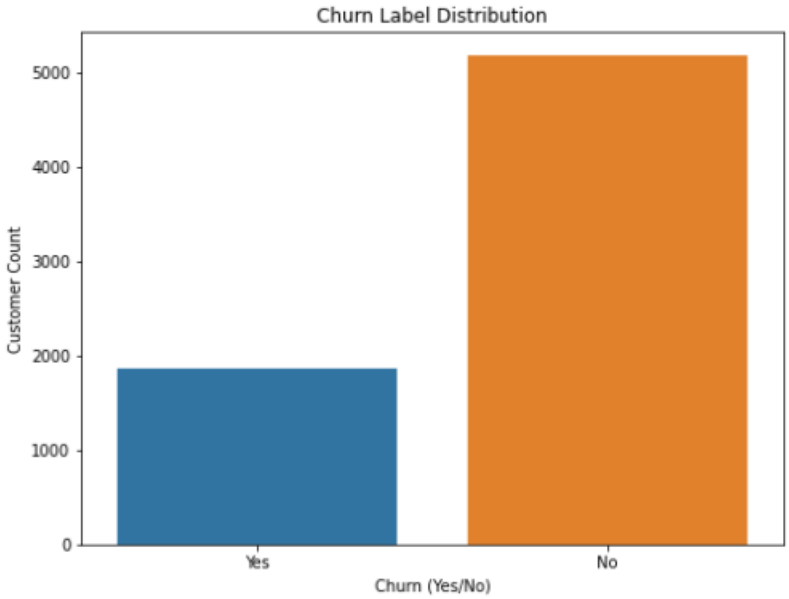



*Figure 1- Label Distribution*

3.1 Churn Label Distribution

This bar char below (Figure-1) shows a clear class imbalance, with more customers who did not churn compared to those who did. This imbalance can bias models toward the majority ('No') class, reducing their ability to detect customers who churn. There are various ways of handing

this imbalance, such as oversampling the minority, undersampling the majority and etc. The way I will handle it for the convenience is to assign class weights, which will penalize the model more for misclassifying the minority class. The details are to be discussed in the model training section.

## 3.2 Exploratory Analysis for Numerical Variables

The Figure-2 below shows that 'Zip Code', 'Latitude' and 'Longitude' have nearly identical distributions across churn and non-churn groups, implying little to no predictive value. Therefore, these features can be considered irrelevant and removed. However, 'Tenure Months' shows a clear distinction, with churners having obviously lower tenure than non-churners, making it a good candidate to be included in the dataset.

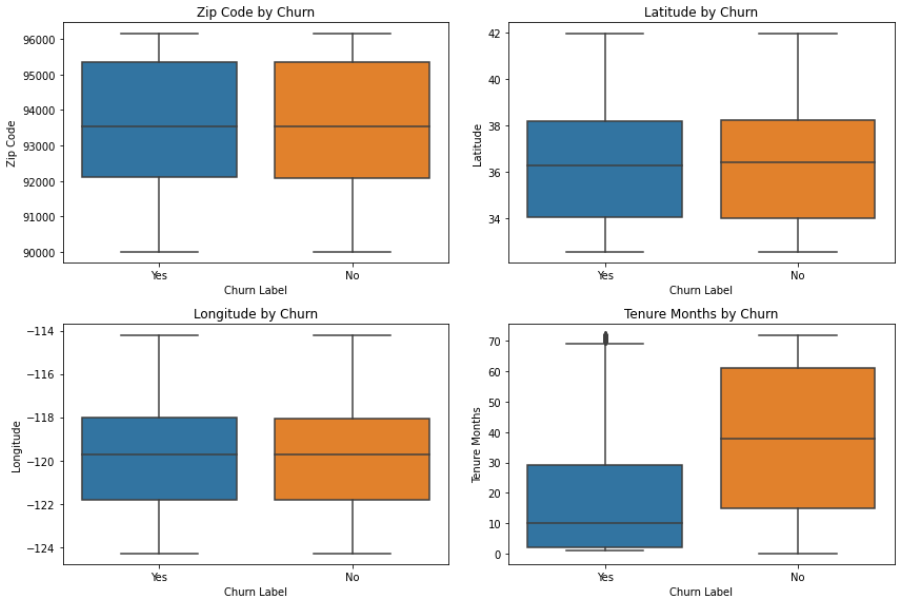



*Figure 2 - Box Plots for Numerical Variables (1)*

The 2nd group of box plots (Figure-3) show that 'Monthly Charges' are slightly higher for churners, suggesting predictive relevance. 'Total Charges' are noticeably lower among churners, likely reflecting their shorter tenure, which makes it a relevant feature.

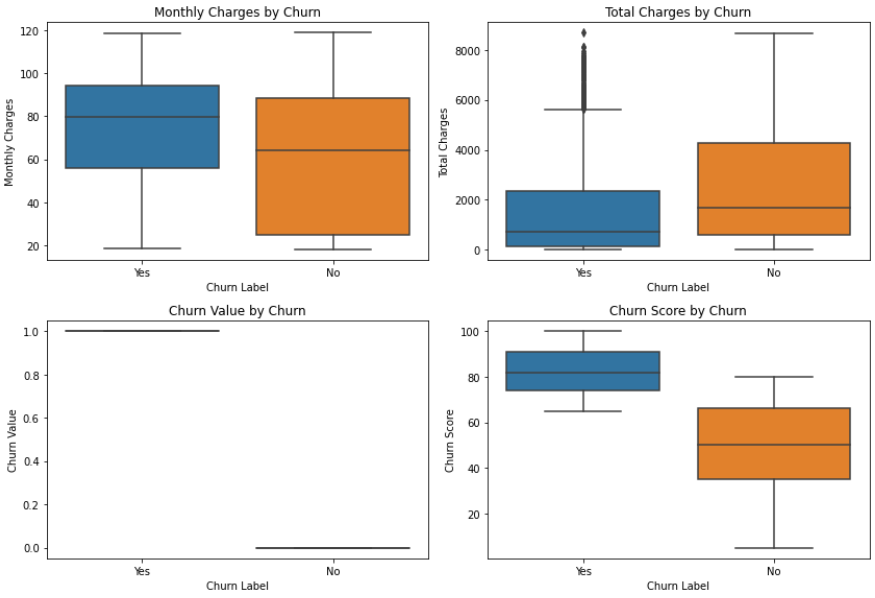



*Figure 3 - Box Plots for Numerical Variables (2)*

However, 'Churn Value' is just a binary label mirroring the target label and offers no additional information for modelling. Also, 'Churn Score' is a derived feature generated by a prior preditive model, using it in our dataset for training will lead to data leakage as we see a near-perfect separation between churners and non-churners by this feature. Therefore, it must be dropped.

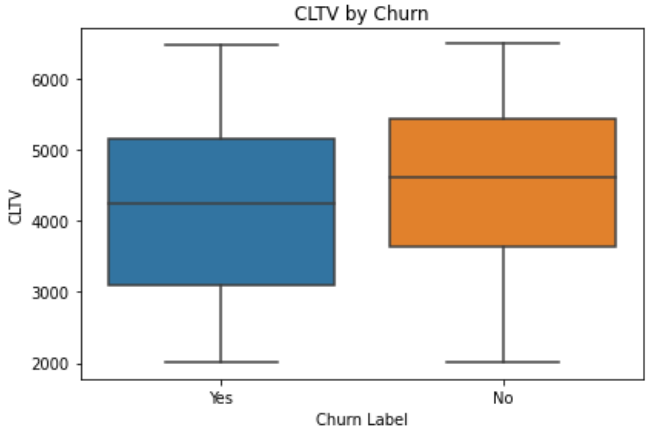



*Figure 4 - Box Plots for Numerical Variables – CLTV*

The box plot of 'CLTV' (Customer Lifetime Value) in Figure-4 shows a slight trend where non-churners tend to have slightly higher 'CLTV' values. While the separation is not as strong as with variables like 'Tenure Months' or 'Churn Score', 'CLTV' may still carry useful predictive information so that it will be retained in our dataset for modelling.

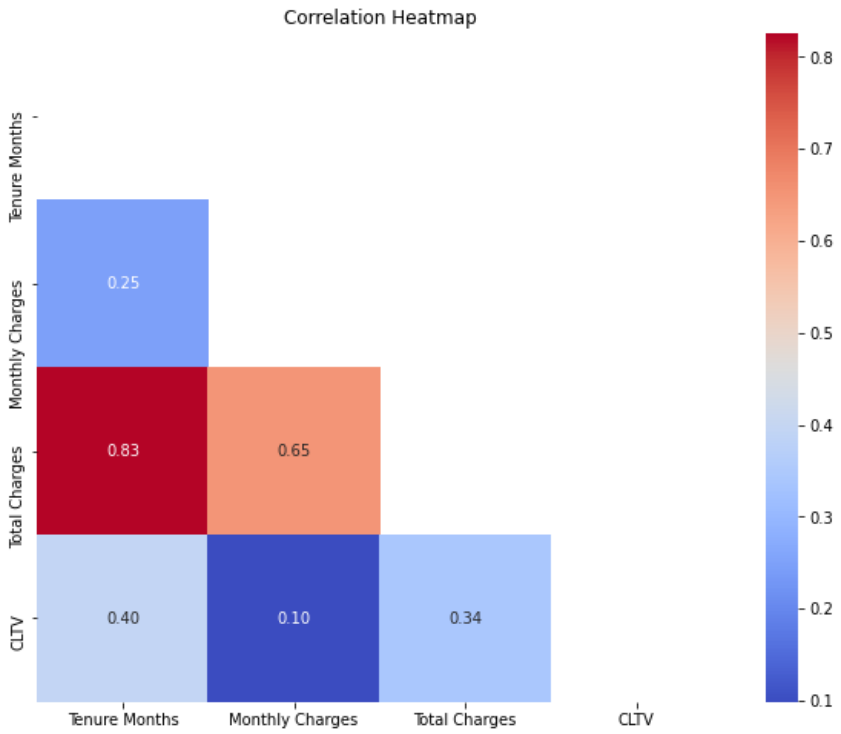



*Figure 5- Heatmap for Correlations among Numerical Variables*

Before starting analyzing the categorical variables, one more action to take is to check for correlations among the numerical variables retained in our dataset. In models such as Logistic Regression to be used in this project, high multicollinearity can lead to unstable or hard-to-interpret coefficient estimates. It could also inflate the standard errors of the coefficient estimates, reducing the precision and weakening the reliability of the inferences of our models.

The correlation heatmap in Figure-5 shows a strong positive correlation (0.83) between 'Total Charges' and 'Tenure Months', indicating that 'Total Charges' is basically redundant, as it is just a function of 'Tenure Months' and 'Monthly Charges'. Including both in the model could introduce multicollinearity. Therefore, we will drop 'Total Charges' to avoid redundancy and improve model interpretability.

### 3.3 Exploratory Analysis for Categorical Variables

The bar plots below comparing mean churn rates across categorical features highlight several key patterns. First, demographic variables like gender show minimal difference between churners and non-churners, indicating limited predictive value. Plus, we need to exclude `Gender` based on

ethical considerations. Gender-based modeling risks reinforcing bias or discrimination in customer treatment strategies. However, customers identified as senior citizens exhibit a significantly higher churn rate than non-seniors. Similarly, customers without partners or dependents are more likely to churn, suggesting that household context influences retention.

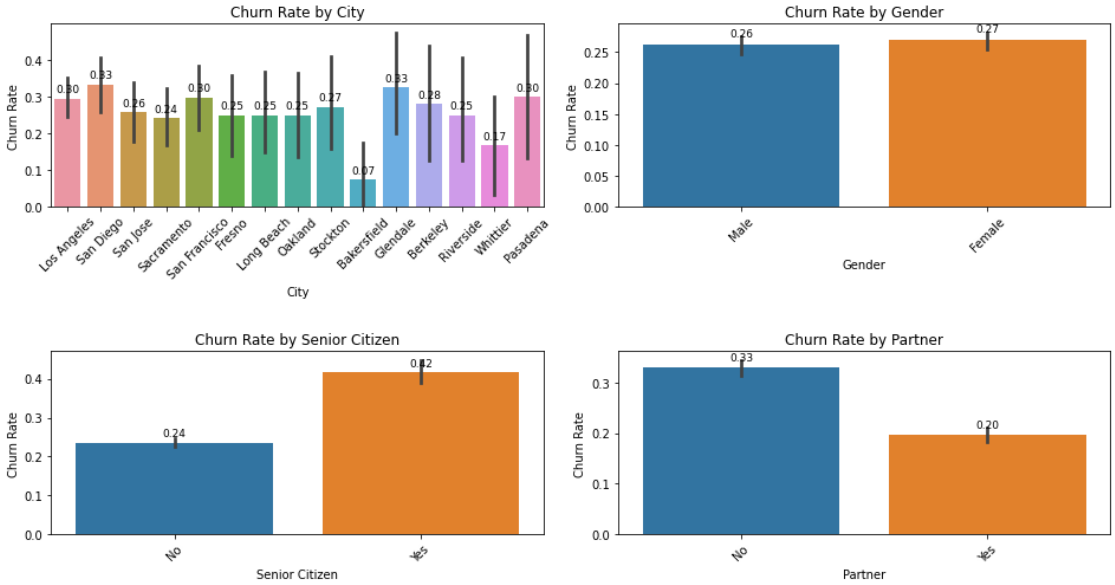



*Figure 6 - Bar Plots for Categorical Variables (1)*

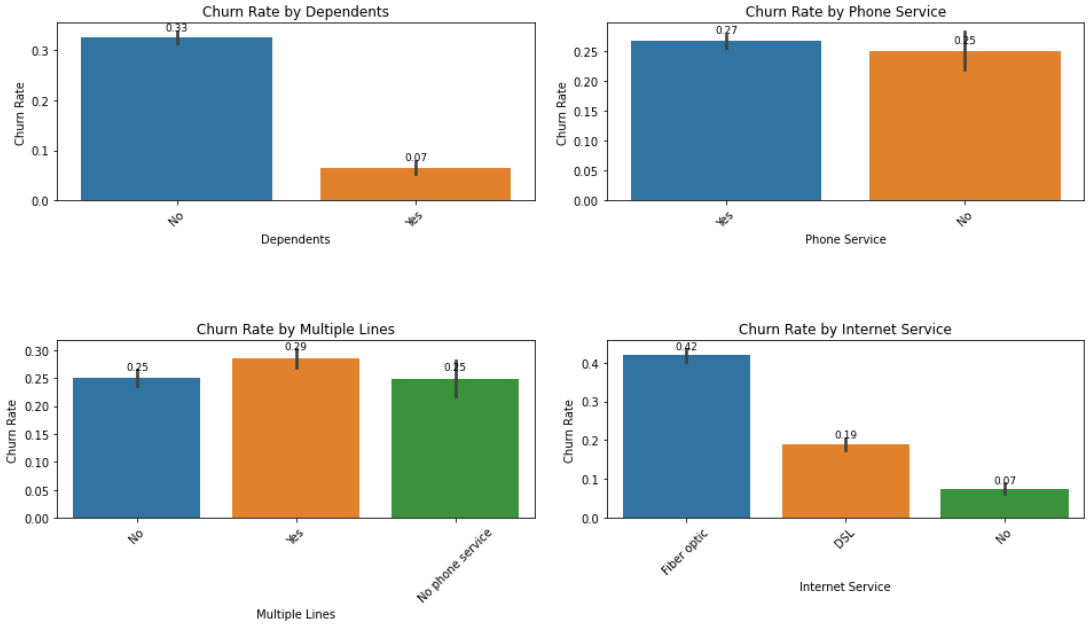



*Figure 7 - Bar Plots for Categorical Variables (2)*

Service-related attributes show stronger associations with churn. Customers who do not subscribe to add-on services like online security, backup, tech support, or device protection consistently display higher churn rates. Those with fiber optic internet also churn more frequently compared to DSL users or those without internet. Additionally, payment method is a strong behavioral indicator—churn is highest among customers who pay via electronic check and much lower among those using automatic payment methods like bank transfer or credit card.

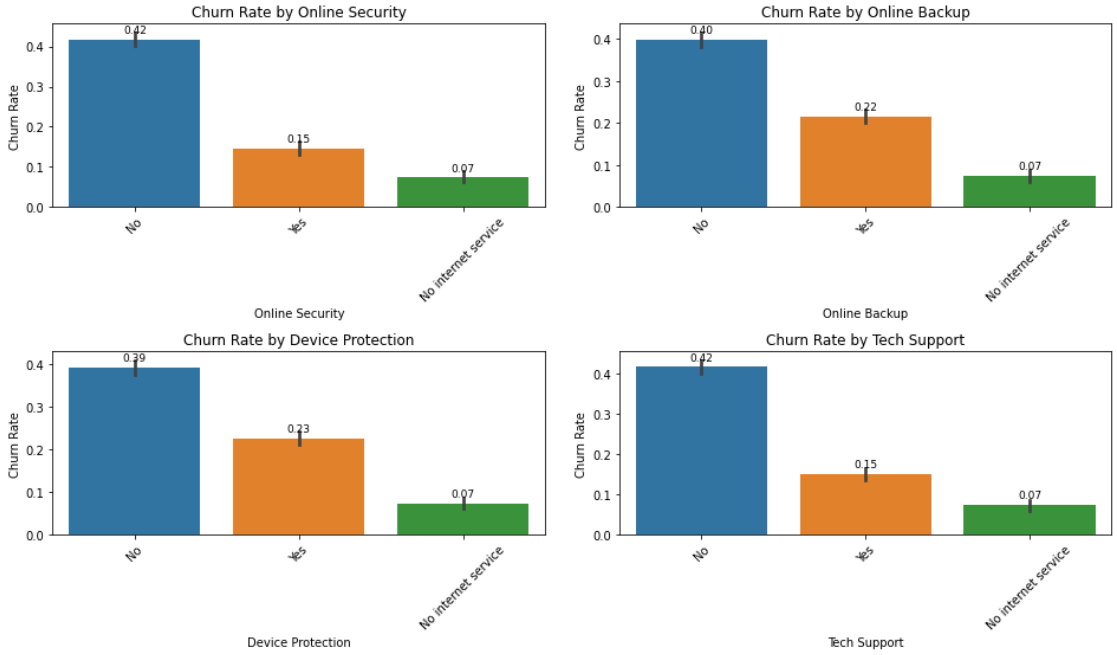



*Figure 8 - Bar Plots for Categorical Variables (3)*

Several features were also deemed irrelevant and are removed for modeling. ‘Country’ and ‘State’ each contain only one unique value across all records, offering no predictive value. Likewise, geographic features such as Zip Code, Latitude, and Longitude were found uninformative in numerical EDA section. Therefore, it makes sense to remove ‘City’ as it also showed no consistent churn pattern and is high in cardinality. These removals help streamline the dataset while preserving the most meaningful categorical information.

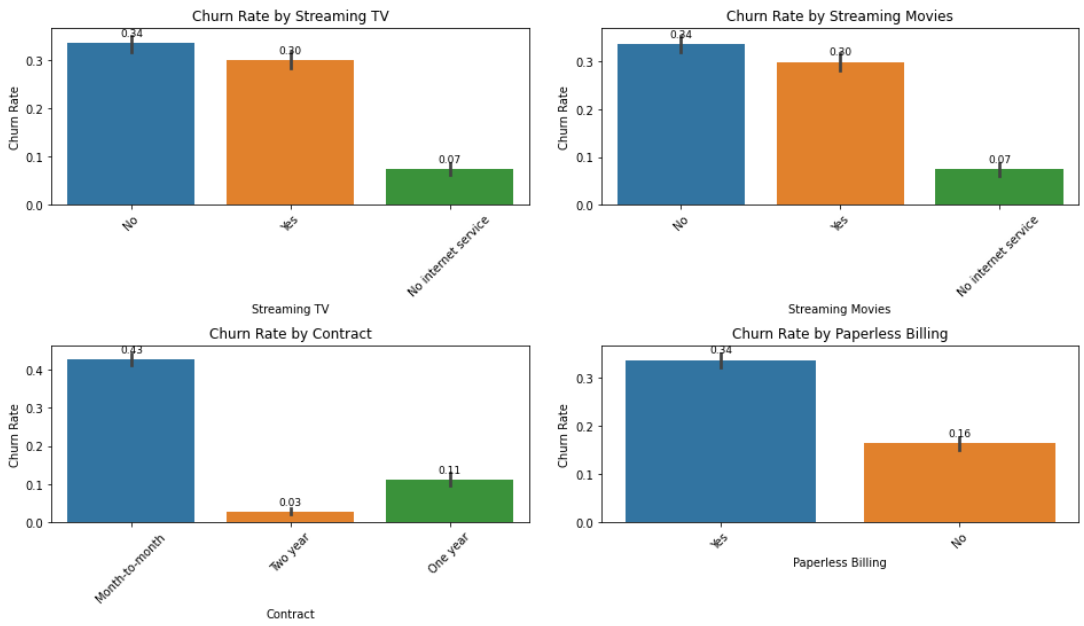



Figure 9 - Bar Plots for Categorical Variables (4)

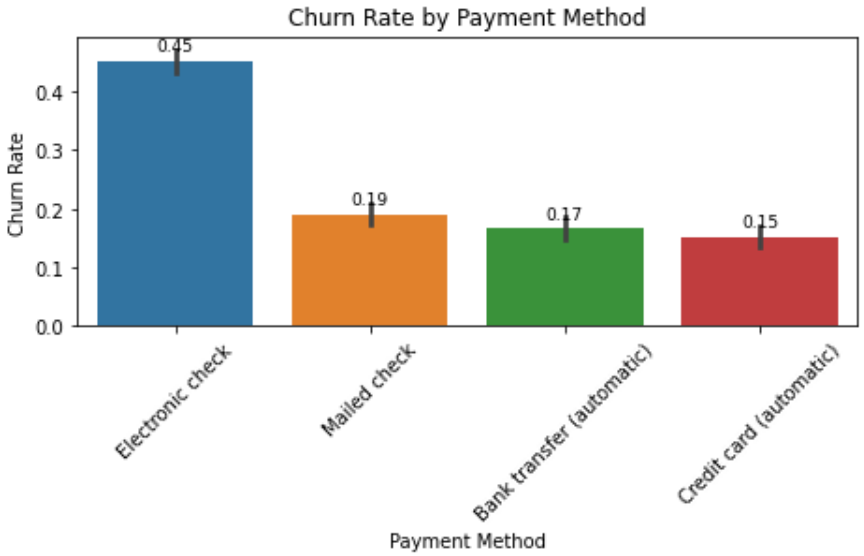



*Figure 10 - Bar Plots for Categorical Variables - Payment Method*

## 4. Data Pre-Processing

After completing EDA, we can proceed with data preprocessing to prepare the dataset for modeling. First, to ensure data consistency, we verified the structure of the dataset with retained columns. It consists of 7,043 observations and 19 columns, including 16 categorical (object type)

columns (with 'Churn Label'), 2 integer variables ('Tenure Months' and 'CLTV'), and 1 float variable ('Monthly Charges').

To prepare our final dataset, the Churn Label target variable was converted into binary values (Yes → 1, No → 0). Categorical features were then transformed into numerical format using one-hot encoding, with the first category dropped to avoid multicollinearity.

As discussed in Module 3 Lesson 1-2 [6]: *"Linear Regression with an Indicator Matrix"*, it is essential to transform nominal features into a numerical format without introducing unintended ordinal relationships. One-hot encoding achieves this by converting each category into a binary vector representation that preserves its categorical nature.

Formally, for a categorical variable:

$$\boldsymbol{x} \in \{ c_1, c_2, \dots, c_K \}$$

with *K* distinct categories, the one-hot encoded vector

$$\boldsymbol{x_{one-hot}} \in \{0, 1\}^k$$

is defined as:

$$\boldsymbol{x}_{\text{one-hot}} = [\, \mathbb{1}(x = c_2), \mathbb{1}(x = c_3), \dots, \mathbb{1}(x = c_k) \,]$$

where $\mathbb{1}(\cdot)$ is the indicator function. The first category $c_1$ is dropped to avoid multicollinearity (also known as the "dummy variable trap") when fitting linear models. This ensures the resulting predictors remain linearly independent.

For example, consider a categorical variable with three levels: A, B, and C. The one-hot encoded representation, **dropping A**, would be:

~~**A → (0, 0)**~~
B → (1, 0)
C → (0, 1)

In our code, this transformation was achieved by *pd.get_dummies()* with the argument drop_first=True.

In doing so, this transformation allows classification models (e.g., logistic regression, SVMs, decision trees) to treat each category independently without assuming any artificial ordering. It also enables coefficient estimates for each retained category relative to the reference (dropped) level.

It was also confirmed that there were no missing values across any of the columns, confirming that the dataset is clean and ready for model training.

Finally, the dataset was then split into training (80%), validation (10%), and test (10%) sets using stratified sampling to preserve the churn rate across subsets.

# 5. Model Training, Evaluation and Testing

## 5.1 A Brief Introduction to Models to be Applied

This section is to provide a brief introduction to each model to be visited in this project for broader audience who may not be familiar with these basic concepts.

### 5.1.1 Logistic Regression for Binary Classification

Logistic Regression [6] is born for binary classification tasks and serves as a 'Vanilla' model in this project. Unlike linear regression, which usually models a continuous outcome, logistic regression estimates the probability that a given input belongs to the positive class (in this case, churn). It does so by applying the sigmoid function to a linear combination of input features:

$$P(y = 1 \mid x) = \frac{1}{1 + e^{-(\beta_0 + \beta^T X)}}$$

The model parameters are learned by maximizing the log-likelihood function, or equivalently, minimizing the binary cross-entropy loss:

$$l(\beta) = -(1/n) \sum_{i=1}^{n} [\, y_i \, log(p_i) + (1 - y_i) \, log(1 - p_i) \,]$$

Logistic Regression is not only computationally efficient but also easy-to-interpret since each coefficient $\beta_i$ quantifies the effect of the corresponding predictor on the log-odds of churn. This transparency makes it valuable for diagnostic analysis in business context on which predictor(s) would be more influential.

### 5.1.2 Support Vector Classifier

In our class, we know Support Vector Machine (SVM) [7] is a powerful classification algorithm which attempts to find the optimal hyperplane that maximally separates the classes in the feature space. For linearly separable data, the goal is to find a weight vector **w** and bias **b** such that the decision boundary:

$$f(X) = w^T X + b = 0$$

such that it separates two classes with the largest possible margin. This margin is defined as the distance between the decision boundary and the nearest data points from each class, known as support vectors. The objective is to:

$$Minimize \left(\frac{1}{2}\right) * ||w||^2 \text{ subject to } y_i(w^T x_i + b) \geq M \; for \; all \; i$$

For non-linearly separable data, we can apply a soft margin with slack variables [7] and can map data into higher-dimensional spaces using kernel functions (e.g. RBF) to separate two classes in transformed space. In this project, we will try using both linear and radial basis function (RBF) kernels to observe their abilities to separate churners from non-churns in the presence of complex patterns.

For Soft-Margin SVM, the optimization problem is:

$$Minimize \left(\frac{1}{2}\right) * \left|\left|w\right|\right|^2 + C \times \sum_i \xi_i \text{ subject to } y_i(w^T x_i + b) \geq M \times (1 - \xi_i) \; for \; all \; i$$

Where $\sum_i \xi_i \leq some \; specified \; constant \; and \; \xi_i \geq 0$

The $\boldsymbol{\xi}_i$ is the slack variable which enables the decision boundary to tolerate certain classification error. With the regularization term C, the sum of the slack variables gets penalized. C is a hyperparameter to be tuned during our training process such that:

a. The higher C, the lower sum of slack variables, the lower tolerance of classification errors, the lower bias, the higher variance.
b. The lower C, the higher sum of slack variables, the higher tolerance of classification errors, the higher bias, the lower variance.

### 5.1.3 Random Forest

Random Forest [8] is a popular ensemble learning algorithm that builds multiple decision trees and combines their outputs, typically through majority voting, to improve classification accuracy and reduce overfitting. Each tree is trained on a bootstrap sample [8] of the data, and at each split, a random subset of features is considered to encourage diversity among trees.

This model handles both numerical and categorical features well, requires minimal preprocessing, and is robust to noise and overfitting. A key advantage is its ability to provide feature importance scores, helping to identify the most influential predictors.

### 5.1.4 DNN – Multi-Layer Perceptron

Deep neural networks (DNNs) are powerful models used to handle complex, non-linear classification problems. In this project, we will try to use a fully connected feed-forward neural network—also known as a Multi-Layer Perceptron (MLP) to predict customer churn. The MLP consists of an input layer, one or more hidden layers, and an output layer. Each neuron in a layer is connected to every neuron in the next, and non-linear activation functions (ReLu) are applied to enable the model to capture intricate relationships among features. The output layer uses a sigmoid activation function to produce a probability, the same we use in the logistic regression.

$$\hat{y} = \sigma(Z) = \frac{1}{1 + e^{-Z}}$$

The loss function using binary cross-entropy is:

$$l(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

The trainable parameters (weights and biases) are learned via iterative updates using back-propagation and gradient descent [9]. To improve training stability and generalization, techniques such as dropout, batch normalization, and early stopping can be employed. These techniques can help prevent overfitting and ensure the model captures meaningful patterns without excessive training.

## 5.2 Training and Hyperparameter Tunings

In this section, we cover the training process for each model introduced previously. For each selected hyperparameter, the model is trained on the training set and evaluated on the validation set. This process allows us to identify the optimal hyperparameter values corresponding to the best performance metric, which, in our case, is the highest validation accuracy.

For the logistic regression model, we performed hyperparameter tuning on the “max_iter” parameter, which specifies the maximum number of iterations the optimization algorithm can execute. In scikit-learn, the default solver is 'lbfgs', a quasi-Newton method that iteratively minimizes the loss function using both gradient and curvature information [9]. The “max_iter”

parameter limits how many optimization steps this solver can take before stopping. Proper tuning of this value ensures that the model has sufficient opportunity to converge while avoiding unnecessarily excessive computation.

The plot of validation accuracy versus maximum iterations showed that accuracy increased sharply at first and plateaued around 200 iterations. The optimal number of iterations 35 corresponded to the highest validation accuracy of 80.82%, marked in the plot.

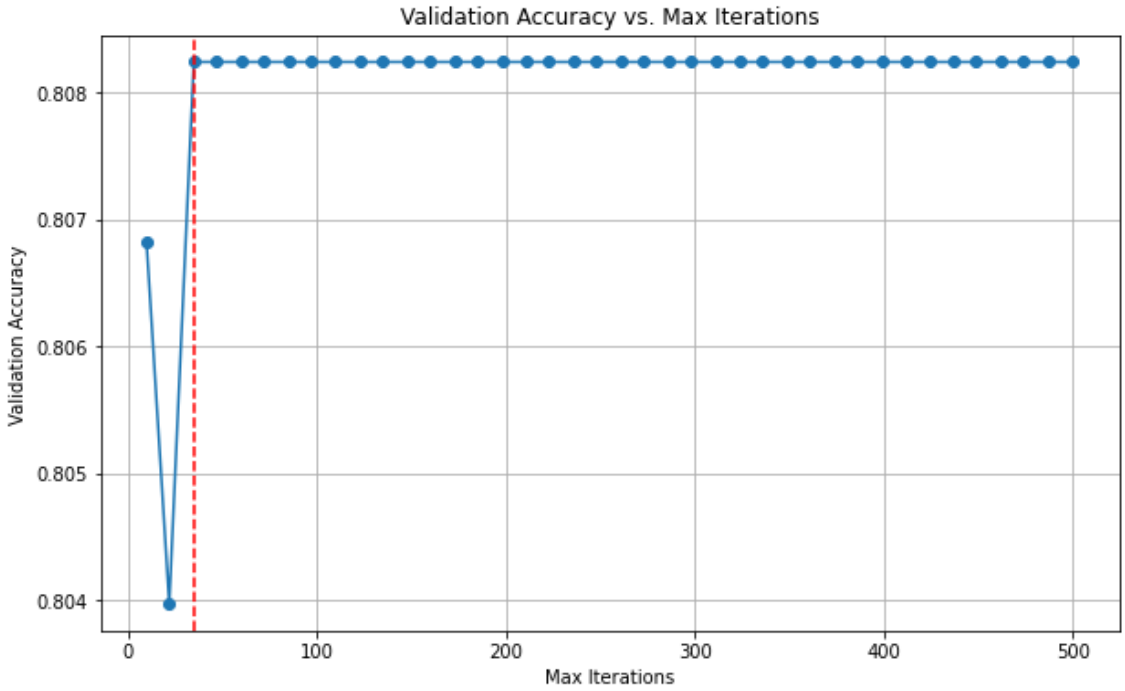



*Figure 11 - Logistic Hyperparameter Tuning - Max Iterations*

To train the SVM classifier, as discussed earlier in the model introduction section, we conducted hyperparameter tuning on the regularization parameter $C$, which controls the trade-off between maximizing the margin and minimizing classification error. A wide range of $C$ values was attempted, spanning from 0.1 to 15 using 100 evenly spaced intervals. For each value of $C$, a linear-kernel SVM model was trained on the training set and evaluated on the validation set, and the corresponding validation accuracy was recorded. The resulting plot below of validation accuracy versus $C$ values showed some fluctuation, with the optimal performance observed around 1. This value yielded the highest validation accuracy and was marked in the Figure-12. Selecting the optimal C helps the SVM balance underfitting (margin too big) and overfitting (margin too narrow), enabling the model to generalize better to unseen data.

In addition, I also explored using the RBF kernel in the training and tuning the SVM classifier. There are two hyperparameters to be tuned. One is C, serves the same purpose as it does in the linear model, the other is gamma γ which controls the influence radius of a single training example. Smaller the gamma is, the larger the influence, the smoother and the more global decision boundary is. But as shown in Figure-13, there has been no improvement compared to the SVM with the linear kernel. Therefore, this alternative is excluded from the scope of work going forward in this project.

For the Random Forest classifier, we performed hyperparameter tuning on the "n_estimators" parameter, which defines the number of decision trees in the ensemble. A range of values from 5 to 100 was used to identify the optimal number of trees that yields the highest validation accuracy. For each setting, a Random Forest model was trained on the training set and evaluated on the validation set.

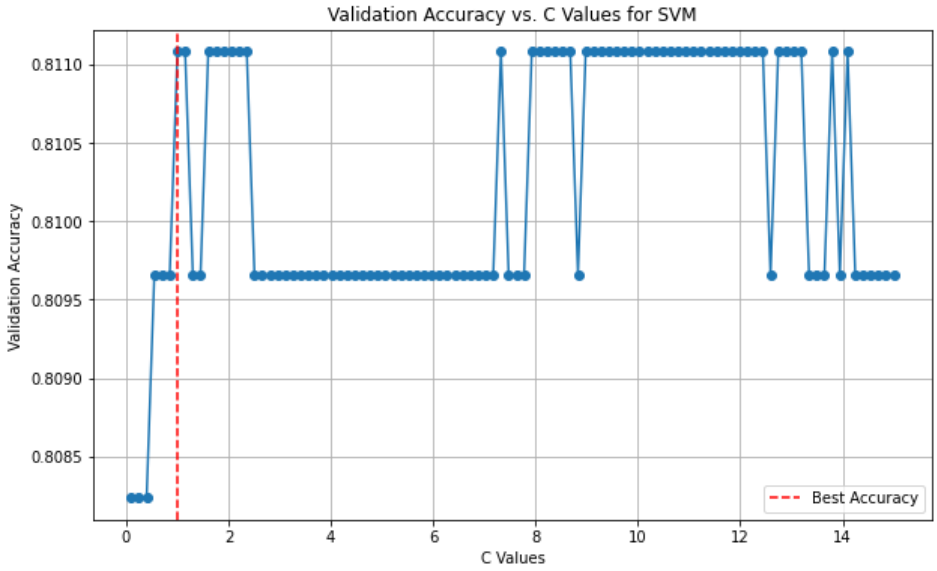



*Figure 12 - SVM Hyperparameter Tunning - Linear Kernel*

The validation accuracy was recorded across all values of "n_estimators", and the results were visualized in Figure-14. The plot revealed that validation accuracy improves with more trees initially, then stabilizes, peaking at a specific value (0.7969). The highest validation accuracy was achieved when the number of estimators was set to the optimal value identified in the plot (n=85). This tuning process helped ensure a good balance between model complexity and generalization without overfitting or underutilizing the ensemble structure.

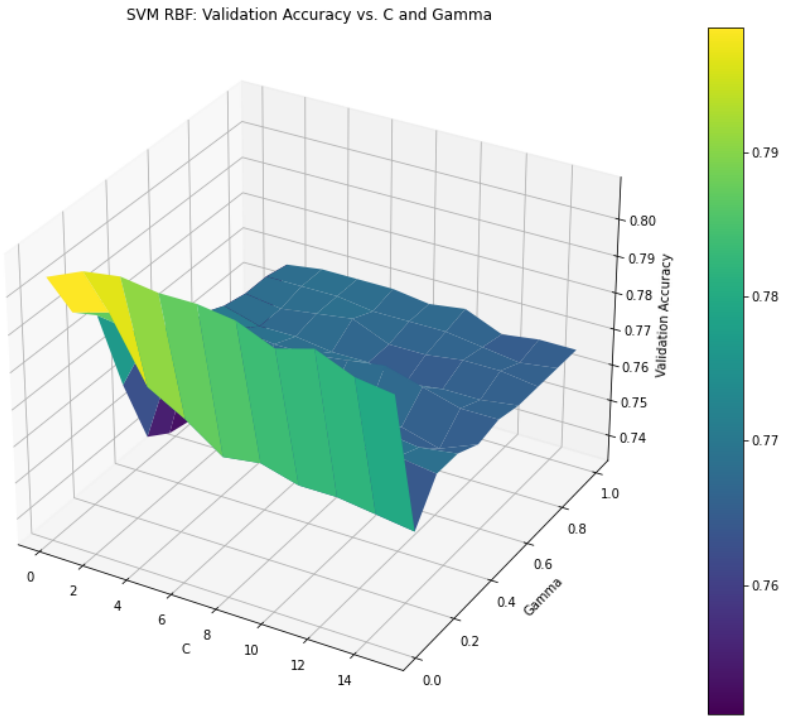



*Figure 13 - SVM Hyperparameter Tunning - RBF Kernel*

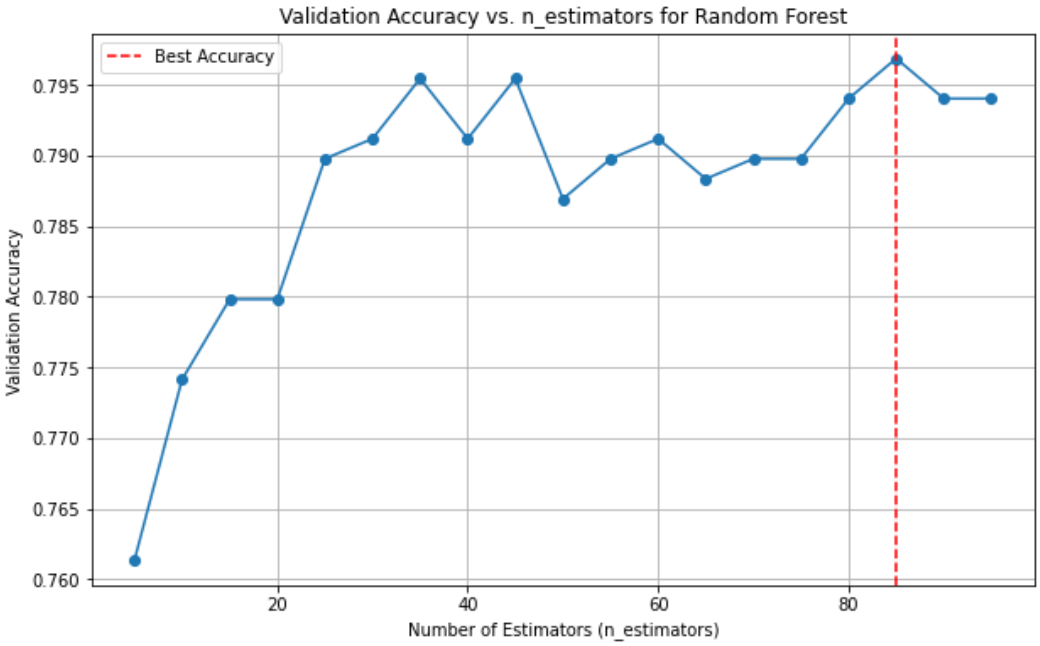



*Figure 14 - Validation Accuracy vs. Number of Decision Trees*

Before training the deep neural network (DNN) as a multi-layer perceptron for binary classification, feature standardization was applied using *StandardScaler* to ensure that all input features had zero mean and unit variance. This step is essential for neural networks because gradient-based optimization algorithms converge more effectively when input features are on a consistent scale.

In this project, the implementation of the neural network is based on Keras. Unlike using PyTorch or raw TensorFlow, which can be more complex and take a longer time to learn, Keras is a high-level API built into TensorFlow that makes building and training deep learning models much simpler and more intuitive [10]. Even though it feels like a separate tool, Keras is really just a user-friendly interface while all the heavy lifting still happens behind the scenes with TensorFlow. So, when we write code using Keras, we're still using TensorFlow to handle things like tensor operations, backpropagation, and GPU acceleration. For beginners and students like me, Keras is super helpful because it makes it easy to prototype the neural networks without having to deal with low-level code. With Keras, we can try out different model architectures, loss functions, and optimizers in just a few lines.

(a) Base MLP:

The first variation of the DNN model was implemented using the Keras' Sequential API, consisting of two hidden layers with 64 and 32 neurons respectively, both using ReLU activation, followed by a final output layer with a sigmoid activation for binary classification. The model was compiled using the Adam optimizer and binary cross-entropy loss, and trained over 50 epochs with a batch size of 32. Validation data was used during training to monitor generalization performance. This initial architecture served as a baseline for further experimentation with deeper structures and regularization techniques in subsequent variations. The highest validation accuracy as shown in the result below is 80.82%. Can we do better than this?

(a) Adding Dropout Layers:

To explore ways to improve the model's generalization and reduce overfitting, a second variation of our MLP was trained with dropout layers included. Dropout is a regularization technique so that a random fraction of the neurons is "dropped out" at each training step (each mini-batch update) during the training, which prevents complex adaptation on the training data. In this model, a dropout rate of 0.3 was applied after each hidden layer, which means 30% of the neurons were randomly disabled during each training pass. Conceptually, dropout can be viewed as training an ensemble of many different subnetworks and averaging their predictions, which often improves model robustness.

```
Model: "sequential_25"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_96 (Dense)            (None, 64)                1920

 dense_97 (Dense)            (None, 32)                2080

 dense_98 (Dense)            (None, 1)                 33

=================================================================
Total params: 4033 (15.75 KB)
Trainable params: 4033 (15.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Best validation accuracy: 0.8082
```

*Figure 15- Base MLP and Validation Accuracy*

```
Model: "sequential_26"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_99 (Dense)            (None, 64)                1920

 dropout_44 (Dropout)        (None, 64)                0

 dense_100 (Dense)           (None, 32)                2080

 dropout_45 (Dropout)        (None, 32)                0

 dense_101 (Dense)           (None, 1)                 33

=================================================================
Total params: 4033 (15.75 KB)
Trainable params: 4033 (15.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Best validation accuracy: 0.8125
```

*Figure 16 -MLP with dropout and Prediction Accuracy*

However, as we can see, the highest validation accuracy now has been slightly increased to 0.8125, showing the performance of this model could be potentially better than the previous one on the test set. Let's see if performing the batch normalization can improve the validation accuracy.

(b) Adding Batch Normalization Layers:

Batch normalization normalizes the output of each previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. This helps reduce internal covariate shift [11], allowing for faster convergence and improved generalization. The modified architecture (Figure -17) includes a batch normalization layers after each dense layer, in addition to the dropout layers used in the previous architecture. After training this updated model, the validation accuracy decreased from 0.8125 to 0.8082, showing that the batch normalization does not necessarily improve the model's performance. What if we make the network deeper? Does a model trained with a deeper network always offer better results?

```
Model: "sequential_27"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_102 (Dense)           (None, 64)                1920

 batch_normalization_37 (Ba  (None, 64)                256
 tchNormalization)

 dropout_46 (Dropout)        (None, 64)                0

 dense_103 (Dense)           (None, 32)                2080

 batch_normalization_38 (Ba  (None, 32)                128
 tchNormalization)

 dense_104 (Dense)           (None, 1)                 33

=================================================================
Total params: 4417 (17.25 KB)
Trainable params: 4225 (16.50 KB)
Non-trainable params: 192 (768.00 Byte)
_________________________________________________________________
Best validation accuracy: 0.8082
```

*Figure 17 - MLP with Added Batch Normalization and Prediction Accuracy*

(c) Making the network deeper:

To explore the effect of network depth on model performance, I constructed a deeper neural network by stacking multiple dense layers with the decreasing number of neurons, each followed by a Batch Normalization and Dropout layer. The motivation behind this architecture was to capture more complex patterns in the data while maintaining model's ability to generalize to unseen data through dropout and normalization. Despite the increased number of parameters (53,377 in total), the deeper model failed to outperform the previous, more compact architecture. The best validation accuracy achieved was 0.8097, which is higher lower than the model with fewer layers and fewer parameters with batch normalization layer. This suggests that increasing

network depth can potentially yield better validation accuracy (generalization capability) and but may introduce more overfitting risk if not tuned properly.

We could further enhance the MLP by experimenting with techniques such as applying kernel regularization or using different optimizers with adaptive learning rates. However, due to time constraints for this project, we will proceed with the models built so far, which demonstrated adequate validation performance on the test dataset.

```
Model: "sequential_30"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_114 (Dense)           (None, 256)               7680

 batch_normalization_45 (Ba  (None, 256)               1024
 tchNormalization)

 dropout_52 (Dropout)        (None, 256)               0

 dense_115 (Dense)           (None, 128)               32896

 batch_normalization_46 (Ba  (None, 128)               512
 tchNormalization)

 dropout_53 (Dropout)        (None, 128)               0

 dense_116 (Dense)           (None, 64)                8256

 batch_normalization_47 (Ba  (None, 64)                256
 tchNormalization)

 dropout_54 (Dropout)        (None, 64)                0

 dense_117 (Dense)           (None, 32)                2080

 batch_normalization_48 (Ba  (None, 32)                128
 tchNormalization)

 dropout_55 (Dropout)        (None, 32)                0

 dense_118 (Dense)           (None, 16)                528

 dense_119 (Dense)           (None, 1)                 17

=================================================================
Total params: 53377 (208.50 KB)
Trainable params: 52417 (204.75 KB)
Non-trainable params: 960 (3.75 KB)
_________________________________________________________________
Best validation accuracy: 0.8097
```

*Figure 18- Deeper MLP with Added Batch Normalization*

One thing I would like to try before the wrap-up is re-running the training loop for the deeper version (Figure-18) with an early stopping mechanism [12]. This technique monitors the model's performance on a validation set after each epoch and halts training once performance stops improving or reaches a certain level. Based on the previous result from Figure-18, I had a rough

idea of the maximum validation accuracy achievable without early stopping, so I created a custom mechanism called “stop_at_threshold,” which stops training once that threshold is reached. The results show very promising performance: the validation accuracy improved from 0.8097 to 0.8111, as shown below in Figure-19.

```
Model: "sequential_34"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_138 (Dense)           (None, 256)               7680

 batch_normalization_61 (Ba  (None, 256)               1024
 tchNormalization)

 dropout_68 (Dropout)        (None, 256)               0

 dense_139 (Dense)           (None, 128)               32896

 batch_normalization_62 (Ba  (None, 128)               512
 tchNormalization)

 dropout_69 (Dropout)        (None, 128)               0

 dense_140 (Dense)           (None, 64)                8256

 batch_normalization_63 (Ba  (None, 64)                256
 tchNormalization)

 dropout_70 (Dropout)        (None, 64)                0

 dense_141 (Dense)           (None, 32)                2080

 batch_normalization_64 (Ba  (None, 32)                128
 tchNormalization)

 dropout_71 (Dropout)        (None, 32)                0

 dense_142 (Dense)           (None, 16)                528

 dense_143 (Dense)           (None, 1)                 17

=================================================================
Total params: 53377 (208.50 KB)
Trainable params: 52417 (204.75 KB)
Non-trainable params: 960 (3.75 KB)
_________________________________________________________________
Best validation accuracy: 0.8111
```

*Figure 19 - Deeper MLP with Added Batch Normalization (Early-Stop)*

## 6. Testing and Comparison

After completing the hyperparameter tuning and selecting the best performing configuration for each model using the validation set, we proceeded to evaluate all final models based on the same set-aside test dataset. One of our goals, as stated in the problem statement is to compare different model's predictive performance on unseen data in a fair and unbiased way.

The models tested include:

(a) Logistic Regression (with optimized max_iter)
(b) Support Vector Machine (SVM) Classifier with linear Kernel (with optimized C)
(c) Random Forest Classifier with optimized number of trees (n_estimators)
(d) Multi-Layer Perceptron and its variants built based on deep neural network (DNN)

Each model was evaluated on the test set using accuracy as the primary performance metric. Additionally, confusion matrices and classification reports were generated to provide deeper insight into each model's strengths and weaknesses, especially in distinguishing the minority (churn) class from the majority.

*Table 3 - Summary of Test Results*

| Models | Variants | Validation Accuracy | Test Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|---|---|
| Logistic Regression | Max_Iter = 35 | 0.8082 | 0.7957 | 0.79 | 0.80 | 0.79 |
| SVM Classifier | Linear Kernel where C = 1 | 0.8111 | 0.7915 | 0.78 | 0.79 | 0.78 |
| Random Forest Classifier | n_estimator = 85 | 0.7969 | 0.7929 | 0.78 | 0.79 | 0.78 |
| Multi-Layer Perceptron | Base model | 0.8082 | 0.7603 | 0.76 | 0.76 | 0.76 |
| | With Dropout | 0.8125 | 0.7844 | 0.77 | 0.78 | 0.77 |
| | With Dropout and BatchNormalization | 0.8082 | 0.7957 | 0.79 | 0.79 | 0.79 |
| | Deeper with Dropout and BatchNormalization | 0.8097 | 0.7957 | 0.79 | 0.80 | 0.79 |
| | Deeper with Dropout, BatchNormalization and Early Stop | 0.8111 | 0.8099 | 0.80 | 0.81 | 0.80 |

Among all models, as shown in Table-3, the deeper multi-layer perceptron (MLP) with Dropout and Batch Normalization trained with early-stopping mechanism demonstrated the strongest generalization ability, achieving the highest overall test accuracy and balanced performance across all metrics.

When comparing the models specially on their ability to identify churners, which is our priority in customer retention tasks, **recall** as a metric becomes the most critical, as it reflects the proportion of actual churners correctly identified. From this perspective, the deeper MLP trained with early-stopping again stands out, achieving the highest recall alongside precision and F1-score, outperforming logistic regression, SVM, and random forest models. Other models showed comparably good but slightly lower performance, suggesting they were less effective in flagging potential churners. Given the balance between high test accuracy and superior recall, the deeper MLP model is considered the most effective model covered in this project for identifying customers at risk of churning.

Another aim of this project is to find out the most influential predictors so that the business can make appropriate retention strategies to keep the potential churners.

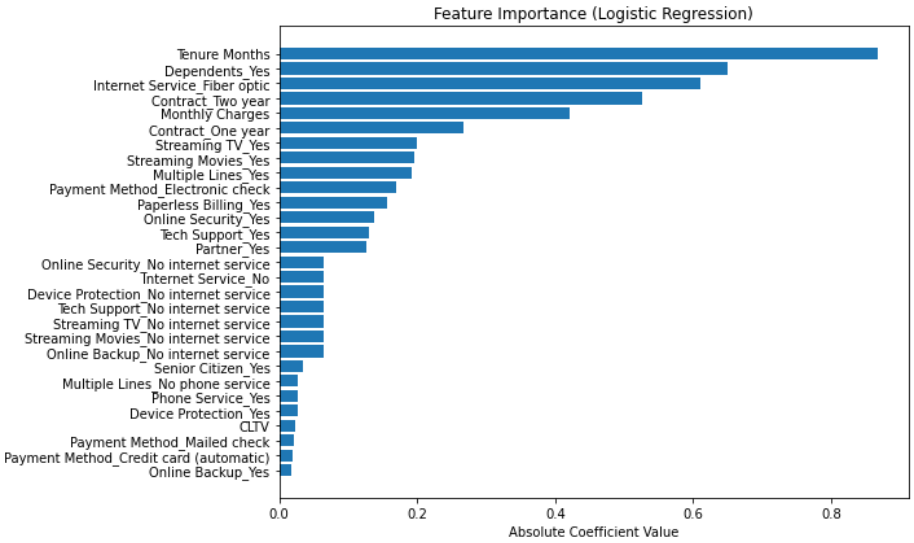



*Figure 20 - Importance of Features – Logistic*

The way I evaluated the importance of the features is by looking at their corresponding absolute coefficient values, as the input feature set was scaled by the standard scaler. The higher the value, the more important the predictor is.

The three figures (only from three interpretable models, as DNN models are not very interpretable) display feature importance rankings from Logistic Regression, Linear SVM, and

Random Forest models. While the specific ordering and magnitudes vary across models, a consistent pattern emerges: “Tenure Months,” “Monthly Charges,” and “Internet Service Fiber optic” appear among the top predictors in all three approaches. This agreement suggests that these features are robust and influential indicators of customer churn, regardless of the underlying modeling technique. Their consistent prominence across models highlights their critical role in distinguishing between customers who stay and those who churn.

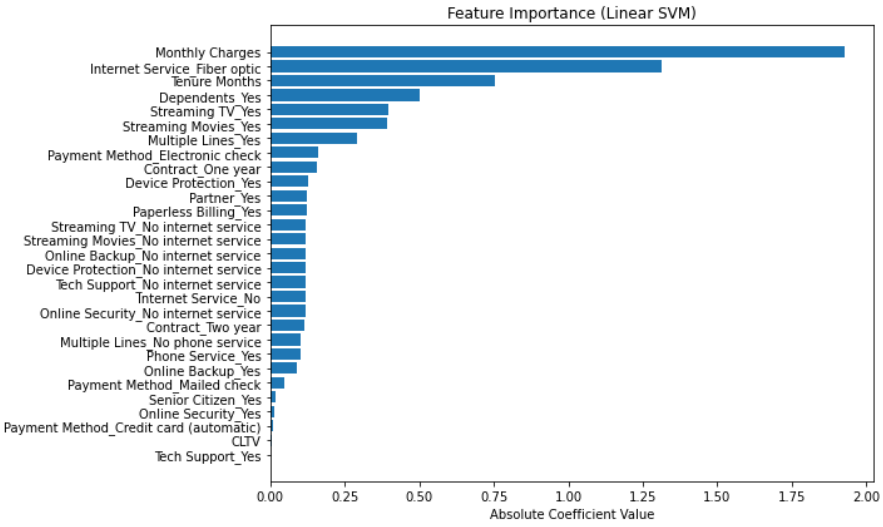



*Figure 21- Importance of Features - Linear SVM*

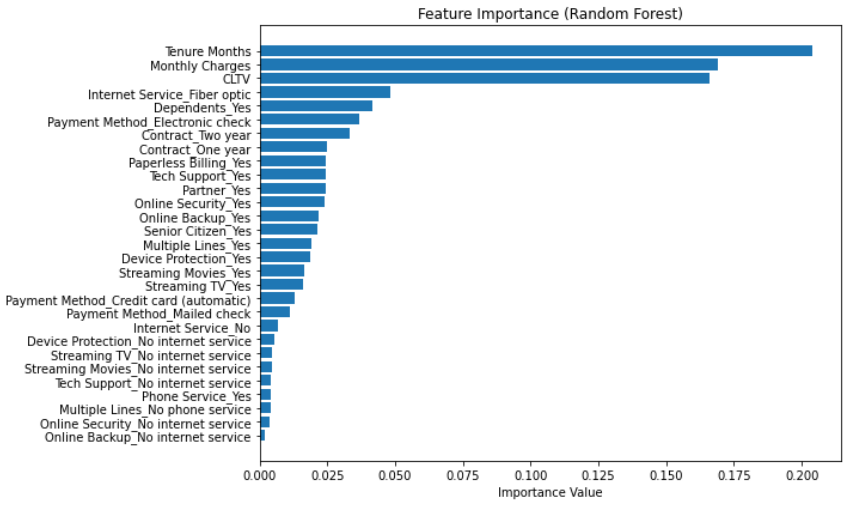



*Figure 22- Importance of Features - Random Forest*

The fact that “Tenure Months,” “Monthly Charges,” and “Internet Service Fiber optic” consistently rank among the top predictors across all models makes intuitive sense in the business context.

“Tenure Months” captures customer loyalty and engagement duration: short-tenure customers are typically more prone to churn as they are still evaluating the service. “Monthly Charges” reflect the customer's financial burden: higher bills can lead to dissatisfaction or price sensitivity, prompting customers to explore cheaper alternatives. “Internet Service Fiber optic,” often associated with premium pricing and higher expectations, can be a double-edged sword: while it offers better performance, unmet expectations or cost concerns can drive churn. Together, these features encapsulate key drivers of customer behavior, including satisfaction, value perception, and commitment.

## 7. Conclusion

In our project, we explored whether customer churn in the telecommunications industry can be predicted using structured customer data, and to determine which machine learning models offer both acceptable accuracy and practically for real-world deployment. Through preliminary exploratory data analysis and feature engineering, the dataset was cleaned and refined to retain the most relevant variables. Multiple models were visited and compared, including Logistic Regression, Support Vector Machine (SVM), Random Forests, and Deep Neural Networks (DNNs).

Among these, the best performance was achieved by a deeper Multi-Layer Perceptron (MLP) with dropout layers, batch normalization, and a custom early stopping mechanism. This model delivered a test accuracy of 81.1%, precision of 80.0% and recall of 81.0%, outperforming traditional models in both overall performance and its ability to identify churners correctly. Given the recall is particularly important for churn prediction, where the cost of failing to identify a churner is often higher than a false positive, this model’s high recall makes it especially suitable for customer retention use cases.

We have also uncovered the most influential features affecting churn, which is important for guiding actionable business strategies. Based on three interpretable models we visited, “Tenure Months”, “Monthly Charges”, and “Internet Service Fiber Optic” consistently emerged as the top predictors. These results are both statistically supported and intuitively meaningful, showing the impact of customer engagement duration, cost sensitivity, and service expectation on churn behavior. In conclusion, the work in this project demonstrates the practical value of statistical learning in enhancing customer retention and provides a foundation for future improvements and business integration.

## 8. Limitations

There are several limitations we should acknowledge despite the solid predictive performance achieved. First, the analysis was limited to the structured feature set. Real-world churn behavior is influenced by more complex and dynamic factors such as customer support interactions, service quality metrics (e.g., call drop rate or service downtime), promotional offers, or sentiment from social media, all of which were not captured in this dataset. Telecommunication companies with well-established data pipelines should be able to collect these types of information so that model performance can be further enhanced. Secondly, due to time constraints, I did not have much time to take further steps to fine-tune the models, although the performance achieved is already on par with that reported in recent literature on the same dataset (81%–84% test accuracy) [13]. Last but not least, in our feature engineering step, we could further improve class balance using more advanced techniques like SMOTE or ensemble-based methods to boost sensitivity to the minority (churn) class.

## 9. Recommendations and Future Work

Based on the findings of this project, telecom companies can integrate churn prediction models—particularly deep learning architectures with strong recall performance—into their customer retention workflows. Special attention should be given to customers with short tenure, high monthly charges, and fiber optic plans, as these features consistently indicate higher churn risk. For further improvement, telecom companies should expand their data collection efforts to include behavioral and interaction-level data such as support call logs, service quality metrics, and promotional response history. Incorporating these additional data sources could significantly enhance model accuracy and actionable insight. Lastly, continuous model monitoring, periodic retraining, and fairness evaluations should be adopted in the production workflow to ensure long-term reliability and ethical deployment.

In order to further improve the performance of our models, firstly, we need to enhance our feature engineering process. This includes exploring interaction terms between existing variables and generating new derived features that may capture latent customer behavior. Secondly, we can investigate some dimensionality reduction or feature selection techniques such PCA, Best Subset Selection and etc. Finally, using more advanced hyperparameter tuning techniques for complex models like neural networks may yield further performance gains.

References:

[1] M. Ullah, F. Jabeen, and R. Khan, “Customer churn prediction using machine learning: A case study of telecom sector,” DIVA Portal, 2019. Available: https://www.diva-portal.org/smash/get/diva2%3A1774181/FULLTEXT01.pdf

[2] C. Nnaji and C. Nwodo, “Predicting customer churn in the telecommunication industry using machine learning algorithms,” ResearchGate, 2022.

[3] J. Zhou, X. Jin, H. Huang, and Y. Liu, “A dual ensemble approach for churn prediction using telecom big data,” PLOS ONE, vol. 18, no. 10, 2023.

[4] M. Óskarsdóttir, S. A. Gudmundsson, and W. Verbeke, “Churn prediction with social network classifiers and collective inference,” arXiv preprint, arXiv:2001.06700, 2020. Available: https://arxiv.org/abs/2001.06700

[5] N. Nurtriana, E. Wahyuni, and D. Yudistira, “Churn prediction analysis of telecom customers using SVM, Random Forest, and Logistic Regression models using Orange data mining tools,” ResearchGate, 2024. Available: https://www.researchgate.net/publication/379049394

[6] S. Jamshidi, Module 3, Lesson 1: Linear Regression and the Indicator Matrix, MATH 569: Statistical Learning, Illinois Institute of Technology, lecture notes.

[7] S. Jamshidi, Module 9, Lesson 8: Support Vector Machines, MATH 569: Statistical Learning, Illinois Institute of Technology, lecture notes.

[8] S. Jamshidi, Module 8, Lesson 1: Tree Models, MATH 569: Statistical Learning, Illinois Institute of Technology, lecture notes.

[9] B. Ghojogh and A. Ghodsi, “Backpropagation and Optimization in Deep Learning: Tutorial and Survey,” 2024.

[10] F. Chollet et al., “Keras API Reference,” TensorFlow Developers, 2023. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization

[11] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” *arXiv preprint arXiv:1502.03167*, 2015. Available: https://arxiv.org/abs/1502.03167

[12] M. A. Hussein and M. Shareef, “An Empirical Study on the Correlation Between Early Stopping Patience and Epochs in Deep Learning,” 2024.

[13] J. Zhou, X. Jin, H. Huang, and Y. Liu, “Customer churn prediction using composite deep learning technique,” *Scientific Reports*, vol. 14, no. 1, Art. no. 30707, 2023. Available: https://www.nature.com/articles/s41598-023-44396-w