Machine Learning Model Report
Churn Detection System

Name – Ajay
Email – ajaysinghpoonia805@gmail.com
Github link for project - GitHub - AjayPoonia112/Churn_Detection

AIM :

Churn Detection by fine tuning and exploratory data analysis with Machine learning models like logistic Regression, Random Forest, Gradient Boosting, XGBoost and AdaBoost

Experimental Procedure :

Platforms Used :

1. Python3 (Language)
2. Jupyter Lab
3. Conda to create enviroments and host notebook
4. Faker, Random (For data Generation)
5. Scikit-Learn (for preprocessing, evaluation, and model building)
6. GridSearchCV (Finetuning the model)
7. Numpy, Pandas (Data Manipulation)
8. Matplotlib, Seaborn (Visualization)
9. Pickle (For saving and loading trained models)

1. Dataset Description :

For the assignment, I created a synthetic dataset collection with predetermined properties.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	CustomerI	Age	Gender	ContractTy	TechSuppo	InternetSe	Tenure	PaperlessB	PaymentM	MonthlyCh	TotalCharg	Churn	average_n	customer_lifetime_value	
2	8e2ac3ff-3	46	Female	Two year	Yes	Fiber optic	13	No	IMPS/NEF	240.4907	2913.291	No	54.67078	710.7201	
3	96f01335-	46	Female	Two year	No	Fiber optic	45	No	UPI	211.8981	9096.154	No	62.53126	2813.907	
4	4082cbb9-	36	Male	One year	No	No	4	Yes	check/DD	489.4094	1032.705	Yes	70.12346	280.4938	
5	4d4f6814-	32	Male	One year	No	No	18	Yes	Card	562.2965	6463.181	No	74.37184	1338.693	
6	eb1313b7-	58	Male	One year	No	DSL	50	No	check/DD	189.033	4747.93	No	42.23757	2111.879	
7	186b2880-	30	Female	Month-to-	Yes	DSL	52	Yes	check/DD	662.8316	13805.57	Yes	96.12952	4998.735	
8	0b070a68-	66	Female	Two year		No	66	Yes	Card	559.6515	17820.7	No	62.21493	4106.185	
9	4648bbe8-	43	Female	Month-to-	No	No	11	No	IMPS/NEF	272.6577	778.5776	No	30.82459	339.0705	
10	9399ad15-	41	Male	One year	Yes	Fiber optic	54	Yes	UPI	387.3118	4907.239	No	40.80623	2203.537	
11	7f702973-	68	Female	Two year	Yes	Fiber optic	36	Yes	IMPS/NEF	419.6067	8378.737	No	75.45253	2716.291	
12	98e52499-	60	Male	Two year	No	Fiber optic	27	Yes	Card	335.7566	3948.739	Yes	64.24307	1734.563	
13	989d9d4a-	29	Male	One year	No	No	9	Yes	UPI	360.0872	1475.859	Yes	43.6682	393.0138	
14	baebe15d-	58	Male	One year	No	No	11	Yes	IMPS/NEF	358.9126	1757.474	No	59.64195	656.0615	
15	90c0e607-	33	Male	Month-to-	No	No	18	No	UPI	311.0626	7263.989	No	86.09918	1549.785	
16	102f4f5c-5	46	Male	Month-to-	No	No	3	Yes	UPI	416.4496	1091.51	No	83.631	250.893	
17	d038e153-	35	Female	One year	No	DSL	62	No	Card	461.0122	19757.59	No	77.74137	4819.965	
18	a9f948b2-	32	Female	One year	Yes	Fiber optic	24	Yes	UPI	365.6359	6694.658	Yes	72.43736	1738.497	
19	6374ed14-	20	Male	One year	No	No	30	No	IMPS/NEF	213.2278	5513.913	No	40.03948	1201.184	
20	1c654aa2-	45	Male	Month-to-	No	No	22	Yes	UPI	235.4802	2734.713	No	46.57043	1024.55	

Figure1: Screenshot of generated data in excel (file name - Customer_churn_data.csv)

Dataset Characterstics:

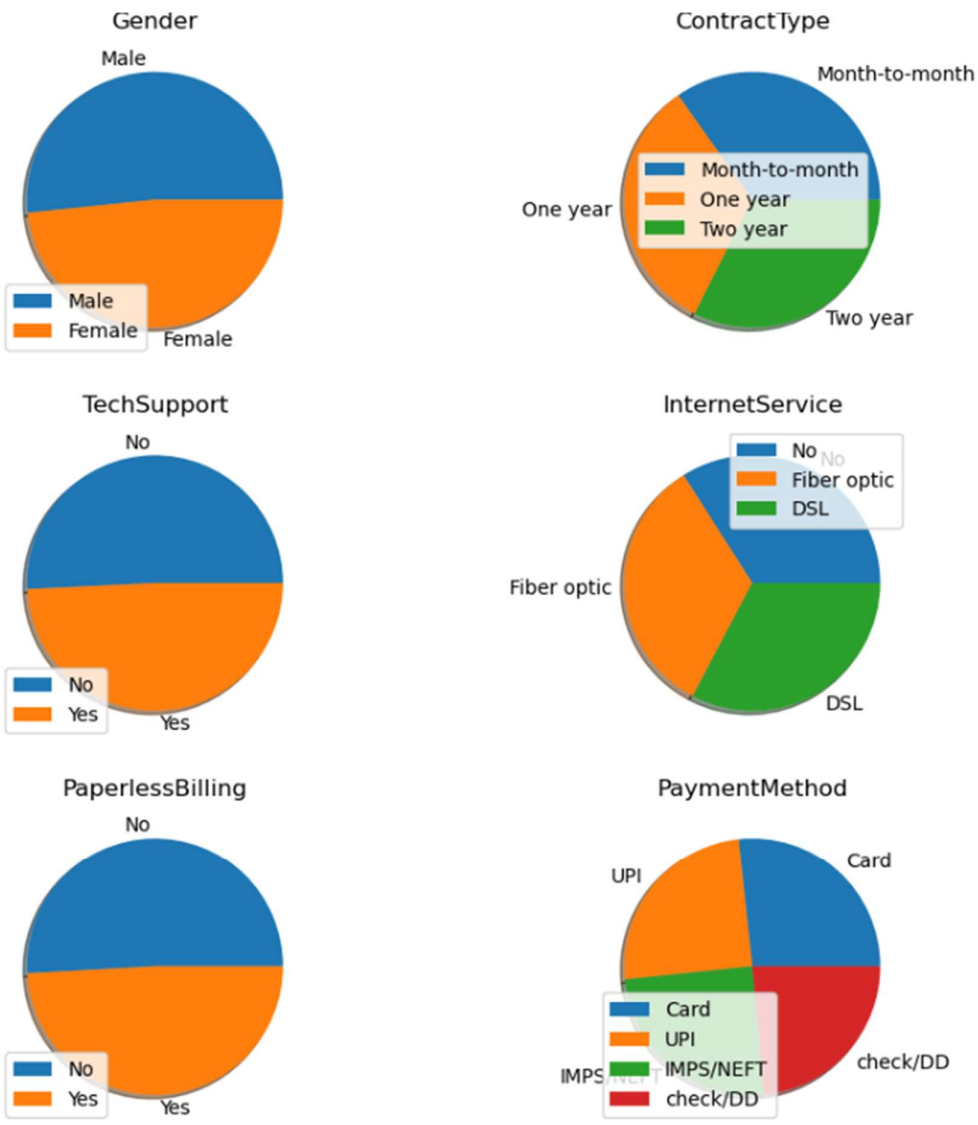
- A generated synthetic dataset of 5000 customer records containing the following features/columns:
  - CustomerID
  - Age
  - Gender
  - ContractType (Month-to-month, One year, Two year)
  - MonthlyCharges
  - TotalCharges
  - TechSupport
  - InternetService (DSL, Fiber optic, No)
  - Tenure
  - PaperlessBilling
  - PaymentMethod (IMPS/NEFT, UPI, Card)
  - Churn (Yes/No)
- Introduce realistic outliers to the data.
- Target churn rate of approximately 20%.
- Create derived features like average_monthly_charges, customer_lifetime_value.

2. Libraries Used

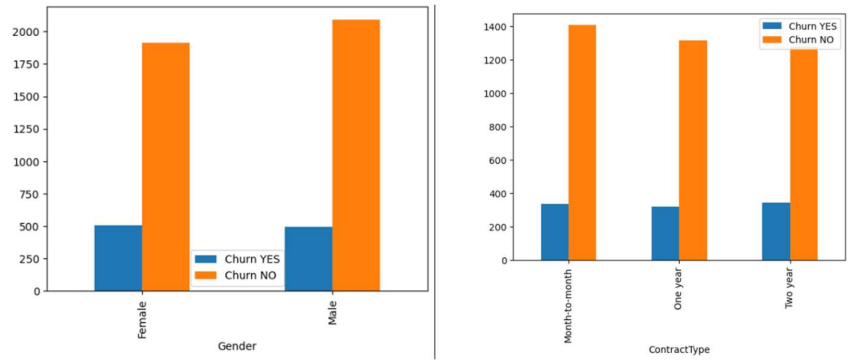
- Pandas
- Numpy
- Matplotlib
- Seaborn
- Plotly
- Scikit_learn
- XGBoost
- Pickle
- Warning

3. Exploratory Data analysis

The dataset comprises 5,000 examples, each with 14 features. The categorical variables include CustomerID, ContractType, TechSupport, InternetService, PaperlessBilling, PaymentMethod, and Churn. The numerical variables are Age, Tenure, MonthlyCharges, TotalCharges, average_monthly_charges, and customer_lifetime_value. Notably, null values are present in the TechSupport, MonthlyCharges, and TotalCharges features.



Figire2: Distribution of Categorical Data



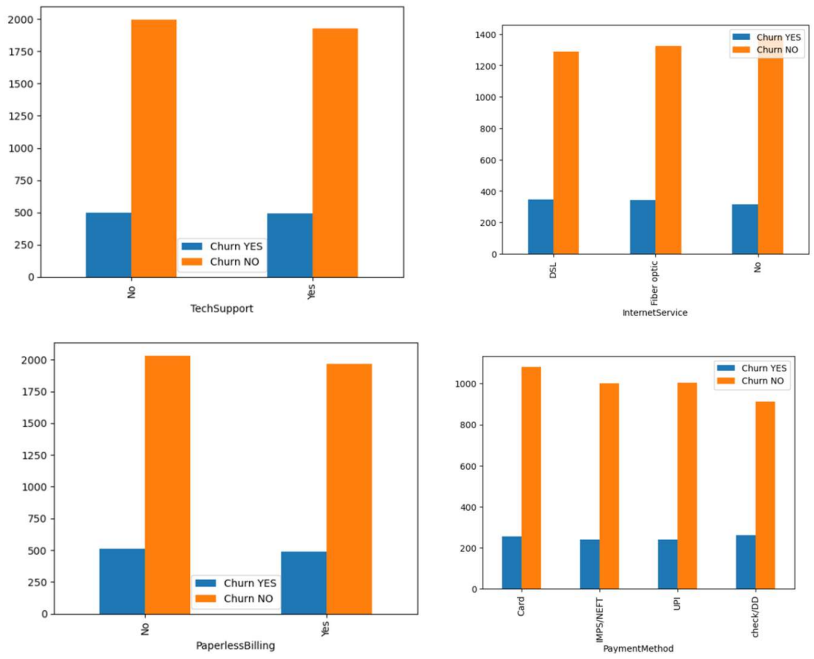
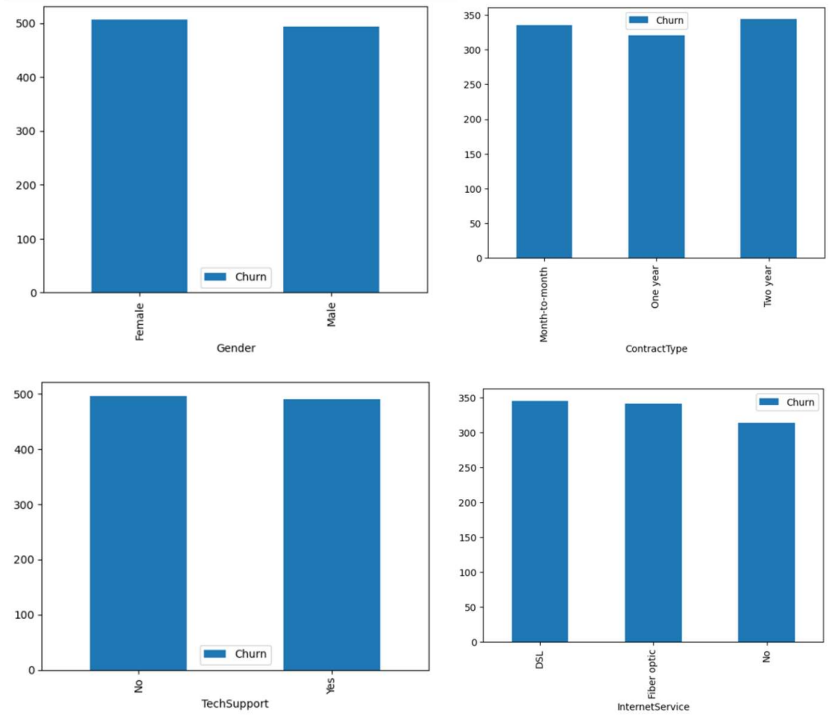


Figure3: Bivariant analysis of categories

The synthetic dataset is imbalanced, with no prominent characteristics evident in any specific category. This imbalance may impact the effectiveness of the analysis and model training.



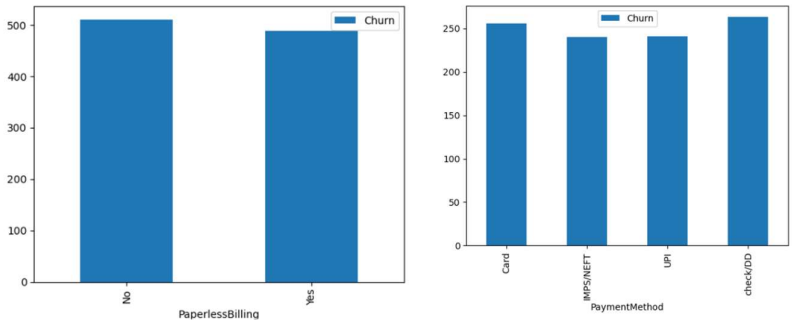


Figure 4: Catgorical count of Yes Churn Data

The analysis of the churn data reveals several noteworthy trends, despite the overall minor differences observed:

- **Gender**: Male customers exhibit a lower churn rate compared to female customers.
- **Contract Duration**: The churn rate is lowest for one-year contracts and highest for two-year contracts.
- **Tech Support**: Churn rates are relatively consistent across different levels of tech support.
- **Internet Service**: Customers with no internet service have the lowest churn rate, whereas those with DSL experience the highest churn rate.
- **Paperless Billing**: The churn rate is higher among customers without paperless billing compared to those with it.
- **Payment Method**: Customers using IMPS/NEFT exhibit the lowest churn rate, while those paying by check or DD have the highest churn rate.

	Age	Tenure	MonthlyCharges	TotalCharges	average_monthly_charges	customer_lifetime_value
count	5000.00000	5000.000000	4917.000000	4917.000000	5000.000000	5000.000000
mean	44.09140	36.299000	80.896910	2580.553263	64.664654	2344.918174
std	15.15284	20.646851	155.822616	2360.019593	15.265925	1469.218026
min	18.00000	1.000000	30.003698	30.594691	30.003698	30.594691
25%	31.00000	19.000000	54.127513	1165.082091	53.703804	1123.321576
50%	44.00000	36.000000	65.232735	2275.093798	64.408358	2203.703096
75%	57.00000	54.000000	77.329925	3478.257670	75.799435	3380.965641
max	70.00000	72.000000	8023.399006	54970.857169	99.962248	7184.120393

Figure 5: Stastical Description of numerical Data

The statistical description of the data includes standard deviation, mean, and quartiles for the features. Notably, MonthlyCharges and TotalCharges contain null values. Additionally, features such as MonthlyCharges, TotalCharges, average_monthly_charges, and customer_lifetime_value exhibit high standard

deviation, indicating potential outliers, which will be further investigated using boxplots.

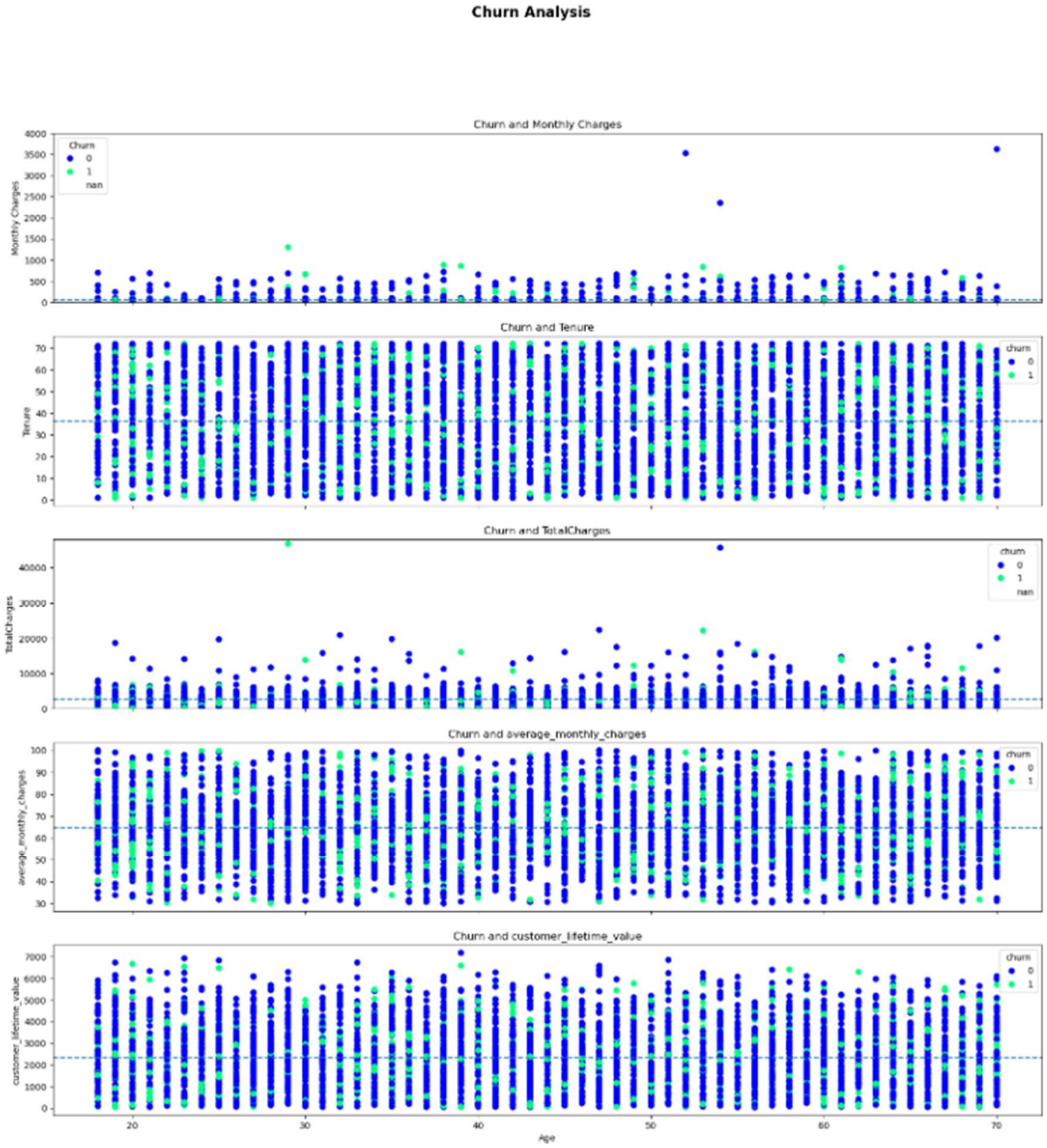


Figure 6 : Multivariant analysis of numerical Data

The multivariate analysis did not yield significant insights for effectively addressing the data imbalance, leaving us unable to identify a precise strategy for balancing the dataset based on the available features. To tackle this issue, I will implement random under-sampling. This approach involves reducing the size of the majority class by randomly selecting a subset of data points, which helps to balance the class distribution. This method can improve model performance by ensuring that the model is trained on a more balanced dataset, potentially leading to better generalization and reduced bias towards the majority class.

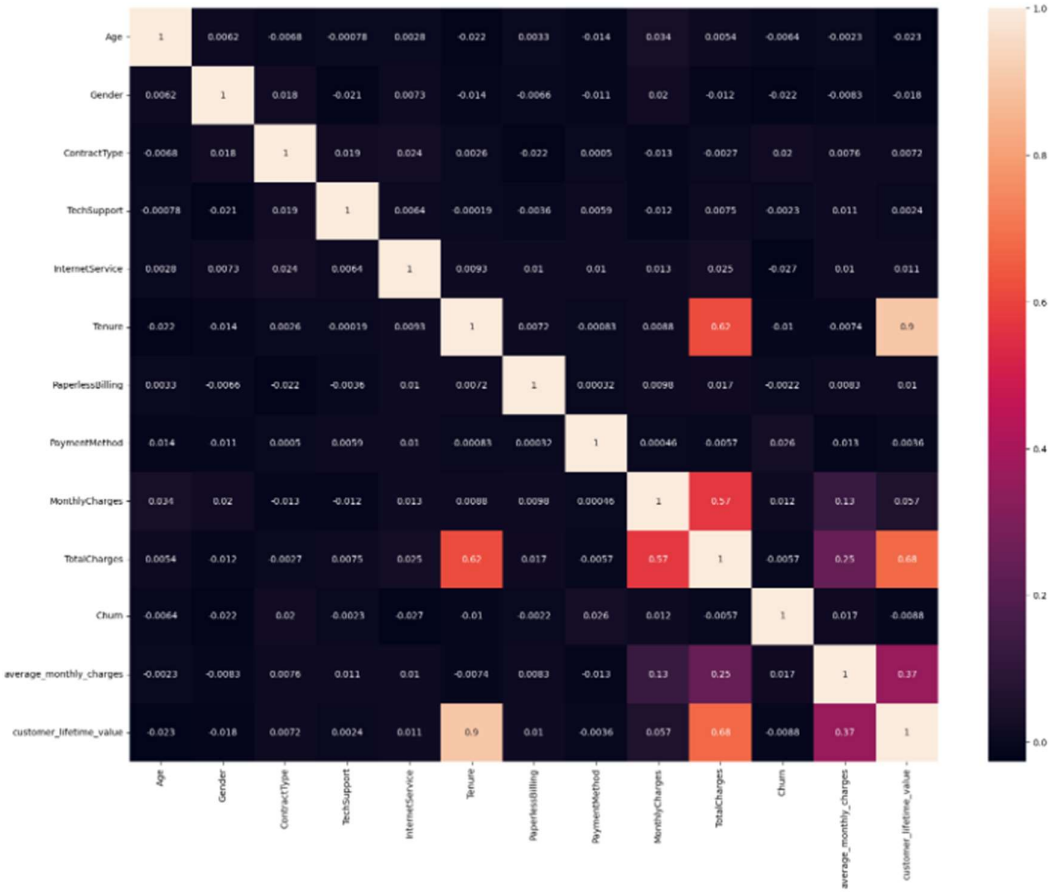


Figure 7: Correlation Matrix Represents linear relation between data

From the correlation analysis, it is evident that customer_lifetime_value, TotalCharges, PaperlessBilling, TechSupport, and Age show relatively low correlation with the target variable, Churn, compared to other features. This suggests these variables might have a weaker relationship with churn and may not be as influential in predicting it.

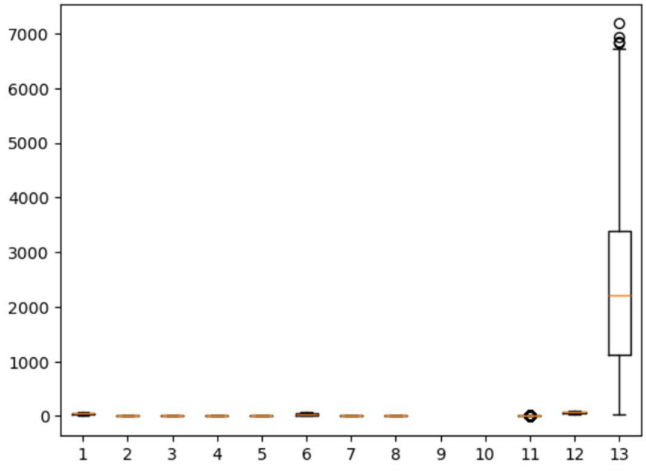


Figure 8: BoxPlot of data

The bar plots indicate that MonthlyCharges, TotalCharges, and customer_lifetime_value exhibit outlier characteristics. These features show considerable variability, suggesting the presence of extreme values that could affect the analysis and modeling.

Given the low quantity of missing values in the dataset:

1. **Handling Missing Values**:
    - **If missing values are very few**: Drop rows with null values.
    - **If missing values are more significant**: Impute the missing data.
2. **Imputation Strategy**:
    - **Numerical Features**:
        - **No Outliers**: Impute missing values with the mean.
        - **Presence of Outliers**: Use the geometric mean or median, as determined by exploratory data analysis (EDA).
    - **Categorical Features**:
        - Impute missing values using the mode (the most frequent value).

This approach ensures that the imputation method is tailored to the specific characteristics and distribution of the data.

3. Preprocessing

After dropping 83 rows with null values, the dataset now contains 4,821 examples out of the original 5,000. There are no remaining null values in the data.

I will use random under-sampling due to the significant class imbalance:

- **Churn (1)**: 970 examples
- **No Churn (0)**: 3,889 examples

Since the number of churn examples (970) is relatively small but not excessively so, random under-sampling is preferred. This technique will reduce the number of non-churn examples to better balance the dataset without introducing bias.

Why Not Oversampling: Oversampling would increase the number of churn examples, but it risks creating an imbalance where the model might incorrectly classify a large number of non-churn subscribers as churn, reducing the model's effectiveness in accurately detecting churn.

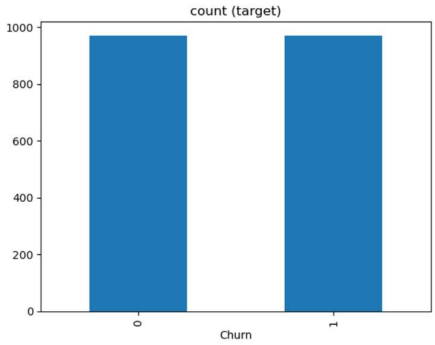


Figure 9: Data distribution after Under Sampling

With the data now balanced, you should re-examine the correlation matrix. This will help assess if the relationships between features and the target variable (Churn) have shifted or if new patterns have emerged due to the balancing process. This step can provide fresh insights into feature importance and correlations.

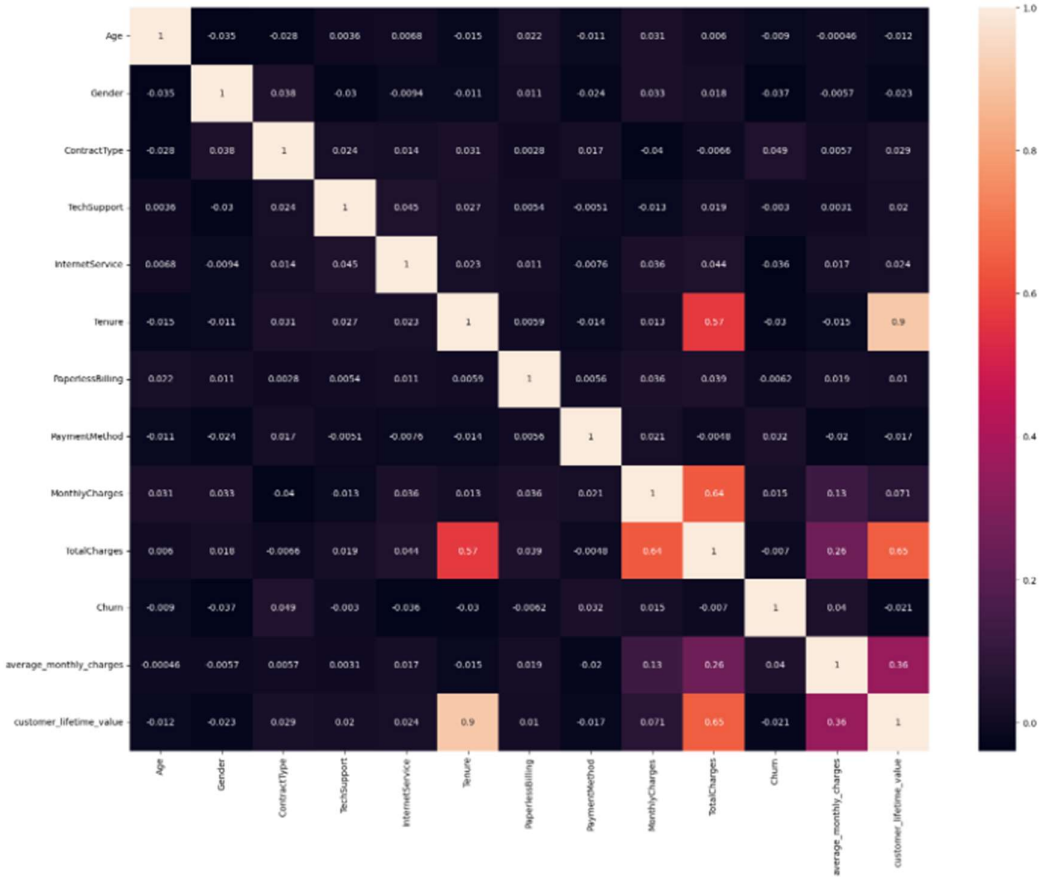


Figure 10: Correlation matrix after under sampling

Train-Test Splitting

Purpose: Train-test splitting is essential for evaluating the performance of a machine learning model. It ensures that the model is tested on unseen data, providing an indication of how well it generalizes to new, real-world data.

Process:

1. **Divide the Dataset**: The dataset is split into two distinct subsets:
    - **Training Set**: This subset is used to train the model. The model learns patterns and relationships from this data.
    - **Testing Set**: This subset is reserved for testing the model's performance. It provides a measure of how well the model performs on data it hasn't seen before.
2. **Typical Ratios**: Common split ratios are 70% for training and 30% for testing, or 80% for training and 20% for testing. The choice of ratio may vary based on the size of the dataset and the specific requirements of the analysis.
3. **Objective**: The goal is to ensure that the model is trained on one portion of the data and validated on another, helping to prevent overfitting and providing a realistic assessment of the model's performance.

One-Hot Encoding

Purpose: One-hot encoding is used to convert categorical data into a numerical format suitable for machine learning algorithms. This process transforms categorical variables into binary vectors.

Process:

1. **Convert Categorical Variables**: Each categorical feature is transformed into a set of binary columns, where each column represents a possible category of the feature. For instance, if a feature "InternetService" has three possible values—FiberOptic, No, and DSL—one-hot encoding will create three binary columns:
    - FiberOptic
    - No
    - DSL
2. **Binary Representation**: For each example in the dataset, the value of the categorical feature is represented by setting the corresponding binary column to 1 and all other binary columns to 0. For example, if the Internet Service is FiberOptic, then FiberOptic would be 1, and No, and DSL would be 0.
3. **Handling New Categories**: In practice, if new or unseen categories appear in the test data that were not present in the training data, additional steps might be needed to handle these cases effectively. This ensures that the model can manage and predict based on these new categories appropriately.

By implementing train-test splitting and one-hot encoding, you prepare your data for training and evaluation, ensuring that the machine learning models can learn effectively and perform well on new data.

4. **Modelling**

When experimenting with various classification algorithms and optimizing them, here's a structured approach you can follow:

1. Initialize Models

Logistic Regression:

- A fundamental algorithm used for binary classification problems. It estimates the probability that a given instance belongs to a particular class.

Random Forest:

- An ensemble method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Gradient Boosting:

- An ensemble technique that builds models sequentially, each model correcting the errors of its predecessor. It combines weak learners to form a strong model.

XGBoost:

- An optimized implementation of gradient boosting that improves performance and computational efficiency. It's known for its effectiveness in various machine learning competitions.

AdaBoost:

- An ensemble method that combines multiple weak classifiers to create a strong classifier. It adjusts the weights of incorrectly classified instances to focus on harder-to-classify cases.

2. Optimize Hyperparameters

Grid Search:

- An exhaustive search over a specified parameter grid. For each combination of hyperparameters, the model is trained and evaluated. It ensures that all possible combinations are tested.

Randomized Search:

- A more efficient alternative to grid search that samples a fixed number of hyperparameter settings from specified distributions. It can be faster and is useful when the parameter space is large.

Implementation:

- Use tools like Scikit-learn's GridSearchCV or RandomizedSearchCV for grid and randomized search, respectively. These tools allow you to specify hyperparameter ranges and evaluate model performance.

3. Evaluate Model Performance

Metrics:

- **Accuracy**: The ratio of correctly predicted instances to the total number of instances. Useful for balanced datasets.
- **Precision**: The ratio of true positives to the sum of true positives and false positives. Important when the cost of false positives is high.

- **Recall**: The ratio of true positives to the sum of true positives and false negatives. Important when the cost of false negatives is high.
- **F1-score**: The harmonic mean of precision and recall. Provides a single metric that balances precision and recall.
- **ROC Curve**: A graphical representation of the true positive rate versus the false positive rate across different threshold settings.
- **AUC (Area Under the Curve)**: Measures the overall performance of the model. An AUC of 1 indicates perfect performance, while an AUC of 0.5 indicates random guessing.

Implementation:

- Use Scikit-learn's metrics module to calculate these performance metrics. For instance, classification_report provides precision, recall, and F1-score, while roc_curve and auc can be used to plot the ROC curve and compute the AUC.

4. Consider Ensemble Methods

Ensemble methods combine predictions from multiple models to improve overall performance. You can use:

- **Voting Classifier**: Combines multiple models by majority voting. Can be a simple way to improve performance by aggregating predictions from different algorithms.
- **Stacking**: Combines multiple models and uses a meta-model to learn how to best combine them.
- **Bagging**: Builds multiple models (e.g., using Random Forest) on different subsets of the training data and aggregates their predictions.

Summary of Steps

1. **Initialize Models**:
    - Logistic Regression
    - Random Forest
    - Gradient Boosting
    - XGBoost
    - AdaBoost
2. **Optimize Hyperparameters**:
    - Use Grid Search or Randomized Search for hyperparameter tuning.
3. **Evaluate Performance**:
    - Use metrics like accuracy, precision, recall, F1-score, ROC curve, and AUC.
4. **Consider Ensemble Methods**:
    - Combine models using techniques like voting, stacking, or bagging for potentially improved performance.

By following this approach, you can systematically evaluate and optimize various classification algorithms to achieve the best performance for your dataset.

Logistic Regression

	precision	recall	f1-score	support
0	0.49	0.46	0.48	199
1	0.47	0.50	0.48	189
accuracy			0.48	388
macro avg	0.48	0.48	0.48	388
weighted avg	0.48	0.48	0.48	388

Random Forest

	precision	recall	f1-score	support
0	0.51	0.55	0.53	199
1	0.49	0.45	0.47	189
accuracy			0.50	388
macro avg	0.50	0.50	0.50	388
weighted avg	0.50	0.50	0.50	388

Gradient Boosting

	precision	recall	f1-score	support
0	0.51	0.45	0.48	199
1	0.49	0.55	0.52	189
accuracy			0.50	388
macro avg	0.50	0.50	0.50	388
weighted avg	0.50	0.50	0.50	388

XGBoost

	precision	recall	f1-score	support
0	0.52	0.52	0.52	199
1	0.49	0.49	0.49	189
accuracy			0.50	388
macro avg	0.50	0.50	0.50	388
weighted avg	0.50	0.50	0.50	388

AdaBoost

	precision	recall	f1-score	support
0	0.52	0.48	0.50	199
1	0.49	0.53	0.51	189
accuracy			0.51	388
macro avg	0.51	0.51	0.51	388
weighted avg	0.51	0.51	0.50	388

Figure 11: Classification report with default parameters

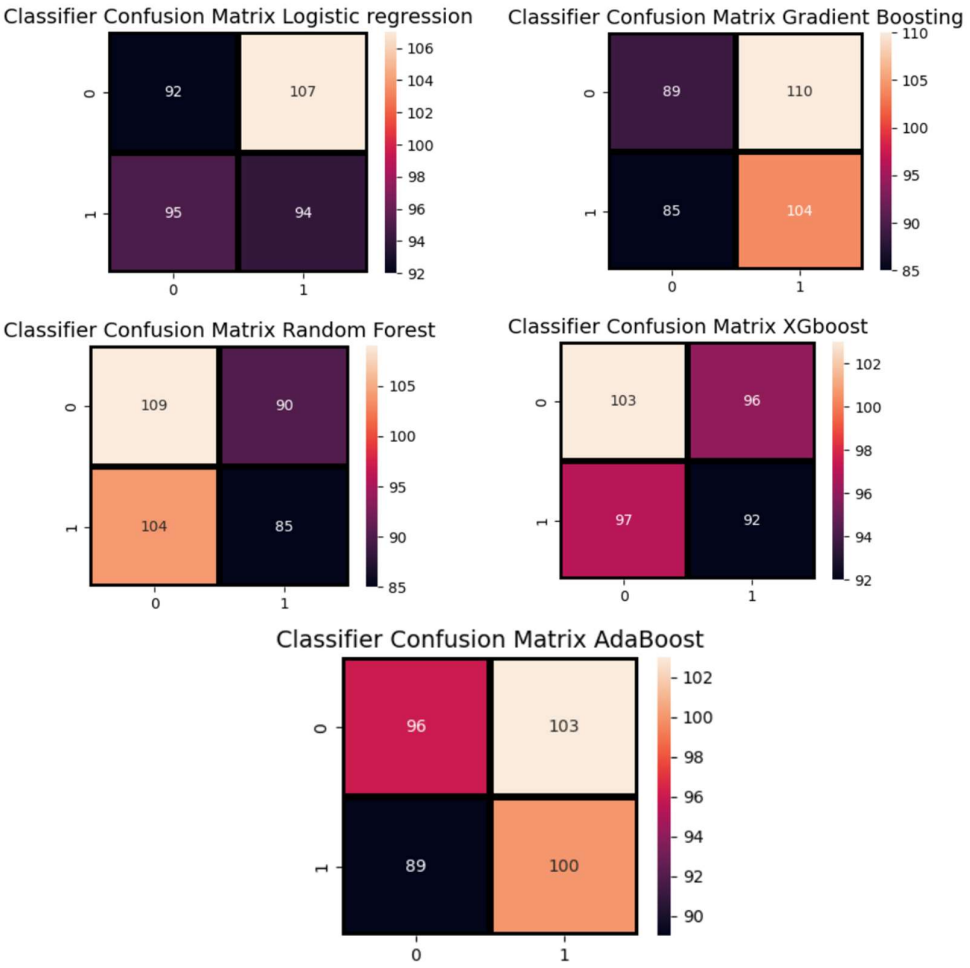
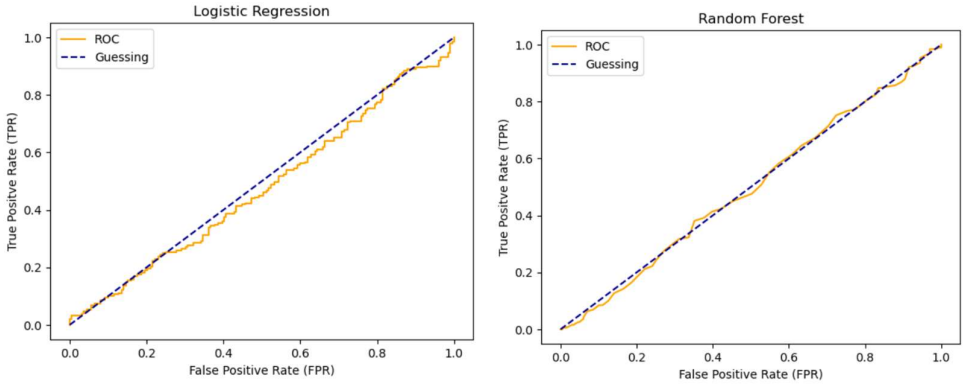


Figure 12: Confusion Matix with Default parameters



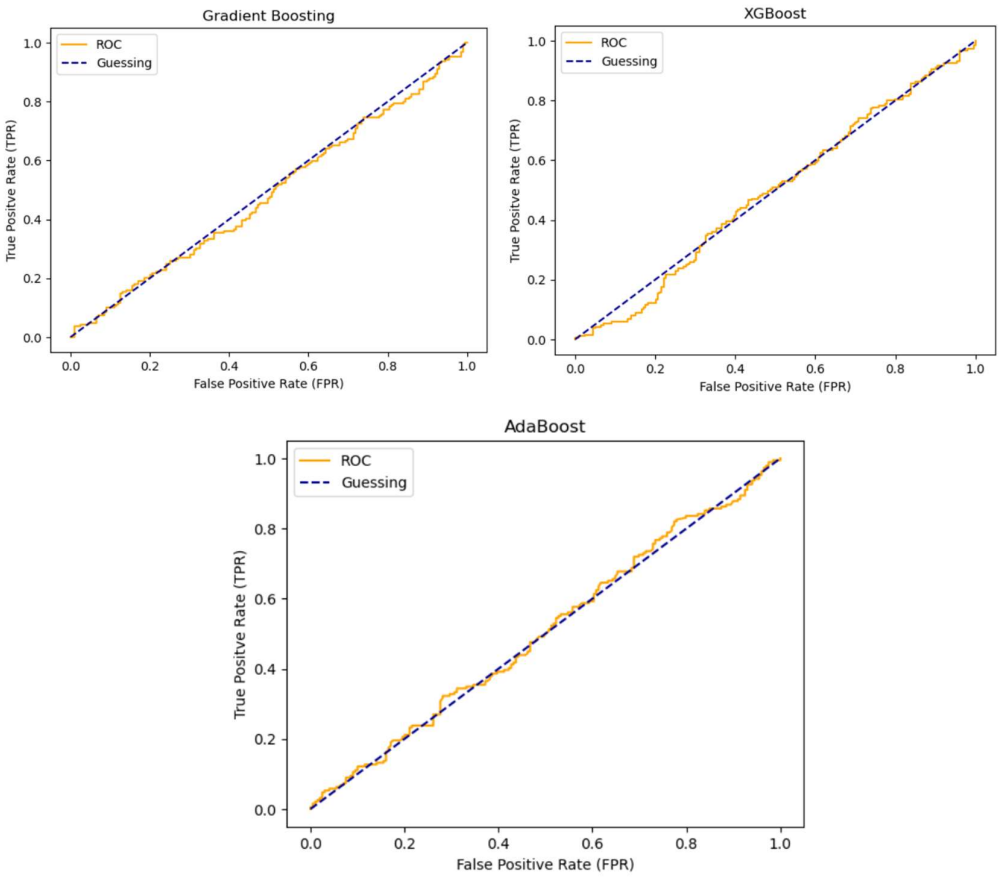


Figure 13: AUC/ROC Curve with Default parameters

Given the constraints, focus on fine-tuning key model hyperparameters and using rigorous evaluation techniques. Fine-tune parameters using Grid Search or Randomized Search to enhance performance. Evaluate models with metrics like AUC from ROC curves and confusion matrices to understand their effectiveness. Although further domain-specific data analysis could yield better results, prioritize refining your current models to maximize performance within the available timeframe.

I am currently not achieving optimal results, indicating a need for further data refinement. However, due to time constraints, I can only focus on fine-tuning a few models at this stage. Given more time and access to real data, a deeper understanding of the domain and features could lead to significant improvements. In the meantime, I have attached AUC curves and confusion matrices from other projects that demonstrate successful results as a reference.

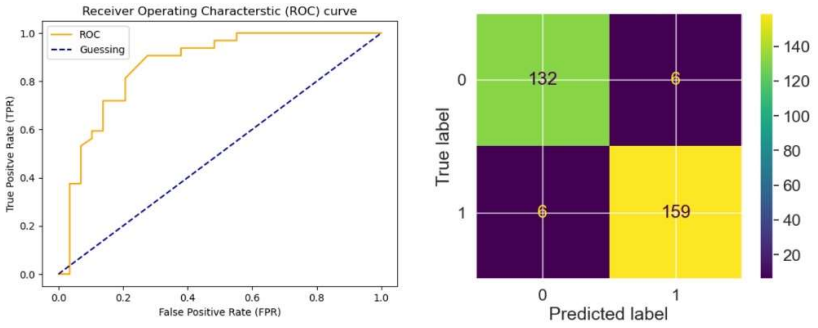


Figure 14: Confusion matrix and AUC/ROC curve with good performance

5. Finetuning

The provided parameters are used for tuning the hyperparameters of the XGBoost model to optimize its performance:

- **min_child_weight**: This parameter controls the minimum sum of instance weights needed in a child node. It affects how the model handles small, noisy data. Lower values can lead to overfitting, while higher values can prevent the model from capturing important patterns.
- **gamma**: This parameter specifies the minimum loss reduction required to make a further partition on a leaf node. It helps in controlling the complexity of the model. Higher values can prevent the model from making unnecessary splits and thus reduce overfitting.
- **subsample**: This parameter denotes the fraction of samples used for building each tree. It helps in preventing overfitting by introducing randomness into the training process. Values less than 1.0 mean that only a subset of data is used, which can improve generalization.
- **colsample_bytree**: This parameter controls the fraction of features used for building each tree. It introduces randomness in feature selection, which can help in reducing overfitting and improving model robustness. Like subsample, values less than 1.0 mean that only a subset of features is used.
- **max_depth**: This parameter specifies the maximum depth of each tree. It controls the complexity of the model. Smaller values create shallower trees that may underfit, while larger values create deeper trees that may overfit.

These parameters are adjusted to find the optimal settings for the model, balancing between underfitting and overfitting, and improving overall model performance.

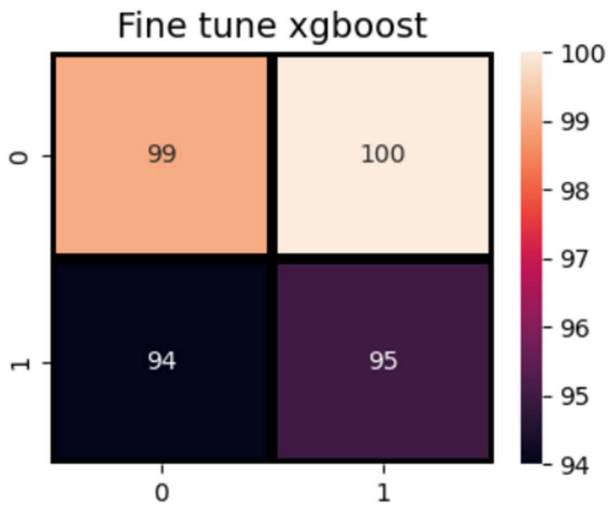
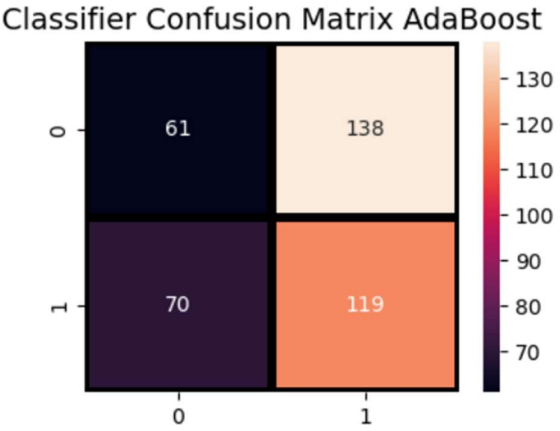


Figure 15: Fine-tuned XGBoost with GridSearch

The fine-tuned XGBoost model is not yielding satisfactory results, so let's explore other classification algorithms. We can consider alternatives such as Random Forest, or Gradient Boosting, which might better capture the underlying patterns in the data. Experimenting with these models could provide improved performance and insights.

The process described involves fine-tuning the AdaBoost model using a grid search approach. First, an AdaBoost classifier is defined. A grid of hyperparameters is created, including a range of values for the number of estimators (ranging from 10 to 500) and learning rates (ranging from 0.0001 to 1.2). A cross-validation strategy, specifically Repeated Stratified K-Fold with 10 splits and 3 repeats, is used to ensure robust evaluation. Grid search is then performed to explore different hyperparameter combinations, evaluating the model's performance using accuracy as the metric. The grid search identifies the best hyperparameter settings by comparing model performance across all combinations, with the results showing the best score and corresponding parameter values.



AdaBoost

	precision	recall	f1-score	support
0	0.47	0.31	0.37	199
1	0.46	0.63	0.53	189
accuracy			0.46	388
macro avg	0.46	0.47	0.45	388
weighted avg	0.46	0.46	0.45	388

Figure 16: Fine-tuned AdaBoost

In this process, a RandomForestClassifier is being fine-tuned using GridSearchCV. A grid of hyperparameters is defined, including various values for the number of estimators, maximum depth of the trees, maximum number of features considered for each split, and minimum samples required for a split or leaf. GridSearchCV performs an exhaustive search over these hyperparameter combinations using 5-fold cross-validation. The model is trained on the training data, and the best performing hyperparameters are identified based on the highest accuracy score achieved. The results display the optimal hyperparameters and the corresponding best score obtained during the search.

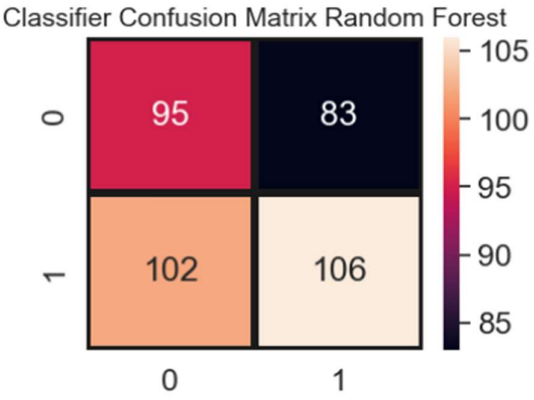


Figure 17: Fine Tunned Random Forest

For deployment, I have chosen to use the AdaBoost model because it currently provides the best performance in identifying true positive churn cases, which is crucial for our goals. AdaBoost, with its ability to focus on difficult-to-classify examples and its ensemble approach that combines multiple weak classifiers, has shown promising results in our evaluations.

However, due to time constraints, I have not been able to further refine the AdaBoost model. Fine-tuning involves more extensive hyperparameter optimization, experimenting with different feature engineering techniques, and potentially incorporating more advanced methods to improve performance.

If additional time were available, the refinement process would include:

1. **Advanced Hyperparameter Tuning**: Further exploration of hyperparameters, such as experimenting with different values for the learning rate, the number of estimators, and the base estimator's parameters, to find the optimal settings that maximize model performance.
2. **Feature Engineering**: Analyzing and potentially transforming features to better capture patterns relevant to churn prediction. This could involve creating new features, removing irrelevant ones, or applying techniques such as feature scaling or encoding.
3. **Cross-Validation and Model Evaluation**: Conducting more rigorous cross-validation to ensure the model's robustness and consistency. This would involve evaluating the model with different cross-validation strategies to gain a better understanding of its generalization capability.
4. **Ensemble Methods**: Exploring additional ensemble techniques or combining the AdaBoost model with other models (e.g., stacking or voting classifiers) to potentially enhance overall performance.

While AdaBoost is effective for our current needs, further refinement could improve its accuracy and robustness. The model's performance will be continually monitored and adjusted as more resources and time become available.

6. Deployment

To deploy a model using Flask, you first set up a Flask web application that can handle HTTP requests. Start by saving your trained model in a format that can be easily loaded, such as with pickle. In the Flask application, create an API endpoint that accepts input data, processes it using the loaded model to generate predictions, and returns the results. Run the Flask server to host the application locally and test it using tools like Spider. Below are the screenshots.

INPUTS

Enter Your Pesent Age

Age

Male

Gender

Month-To-Month

Contract Type

Enter Monthly Charges

Monthly Charges

Enter Total Charges

Total Charges

Yes

Tech Support

DSL

Internet Sevice

Enter Tenure

Tenure

Yes

Paperless Billing

IMPS/NEFT

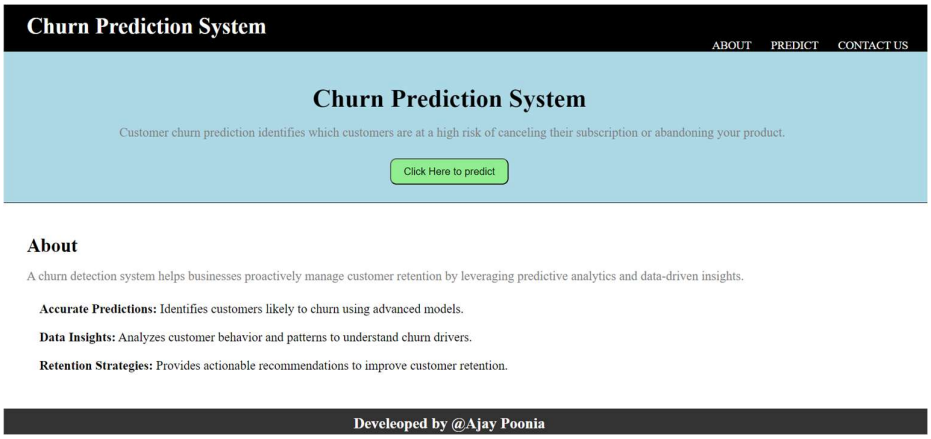
Payment Method

Yes

Churn

ROUND OFF UPTO 3 DECIMALS LIKE 3=3.000, 3.1425=3.143, 2.1423=3.142, 0=0.000

SUBMIT


Churn Prediction System
ABOUT
PREDICT
CONTACT US
Churn Prediction System
Customer churn prediction identifies which customers are at a high risk of canceling their subscription or abandoning your product.
Click Here to predict
About
A churn detection system helps businesses proactively manage customer retention by leveraging predictive analytics and data-driven insights.
Accurate Predictions: Identifies customers likely to churn using advanced models.
Data Insights: Analyzes customer behavior and patterns to understand churn drivers.
Retention Strategies: Provides actionable recommendations to improve customer retention.
Develeoped by @Ajay Poonia