Telco Customer Churn Prediction and Analysis

Abdelaziz Elhelaly, Ahmad Mohammed, Abdelrahman Khaled, Mazen Hesham, Norhan Momen, Mahmoud Ahmad
AI Data Science Track – Digital Egypt Pioneers Initiative

***Abstract*—This project focuses on predicting customer churn using machine learning techniques to support business decision-making and customer retention strategies. Using the Telco Customer Churn dataset from Kaggle, we follow the full data science life cycle—from data collection and cleaning, through feature engineering and model training, to deployment and monitoring. The ultimate goal is to build a robust and interpretable churn prediction model that can help telecom companies proactively identify customers likely to leave and take steps to retain them.**

I. Introduction

Customer churn is a critical metric for businesses, especially in competitive sectors like telecommunications. Churn analysis allows companies to understand why customers leave and how to retain them. Whether a startup or a large enterprise, every organization must track customer feedback, satisfaction, and behavioral trends to enhance service quality and customer loyalty.

By analyzing churn, companies can reduce revenue loss, lower acquisition costs, and improve offerings. It provides a clear view of customer satisfaction and allows businesses to take proactive actions to enhance customer experience and engagement. This is especially crucial in subscription-based industries where losing a customer has long-term financial implications.

To reflect real-world churn prediction challenges, we utilized the Telco Customer Churn dataset available at: https://www.kaggle.com/datasets/blastchar/telco-customer-churn

II. Literature Review

Previous research on customer churn has explored a variety of machine learning algorithms to identify patterns in customer behavior. Traditional methods such as logistic regression and decision trees have proven effective due to their interpretability. More recently, ensemble models like random forests and gradient boosting have demonstrated superior accuracy.

For instance applied Random Forest and XGBoost to telecom churn data and achieved over 85

Interpretability techniques such as SHAP and LIME have become increasingly important for understanding model predictions. Additionally, deployment practices using FastAPI or Flask, along with monitoring frameworks like MLflow, are now common in production-grade churn prediction systems.

III. Problem Definition

The primary goal of this project is to build a robust machine learning pipeline that predicts whether a customer is likely to churn. We aim to:

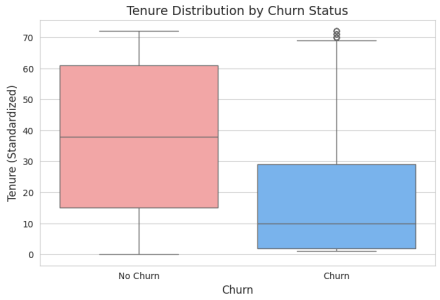
- Identify key drivers of customer churn in telecom.
- Develop and optimize multiple predictive models.
- Deploy the best-performing model as an API.
- Provide actionable insights for business decision-makers.

IV. Methodology

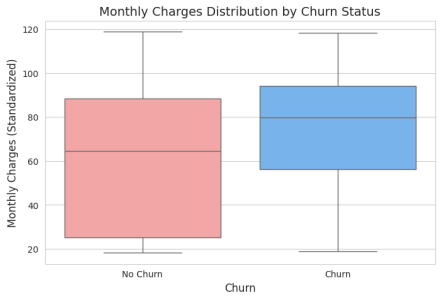
A. Exploratory Data Analysis (EDA)

We began by removing null and duplicate values, particularly 5174 duplicate rows in the ”Churn Reason” column. Summary statistics were generated for numerical columns using the `describe()` function.

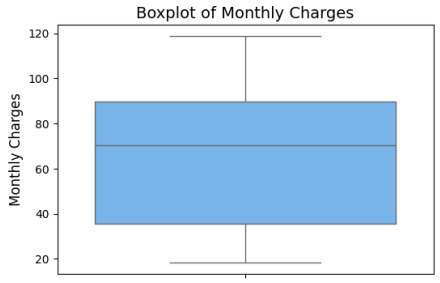
Boxplots were used to detect and remove outliers in Tenure, Monthly Charges, and Total Charges.



(a) Tenure Distribution by Churn Status



(b) Monthly Charges Distribution



(c) Monthly Charges Boxplot

Fig. 1: Comparative analysis of customer tenure and charges distribution between churned and retained customers. (a) Standardized tenure months distribution shows churned customers typically have shorter tenure. (b) Monthly charges distribution reveals higher charges among churned customers. (c) Boxplot visualization confirms outlier presence in monthly charges (IQR = 20-100).

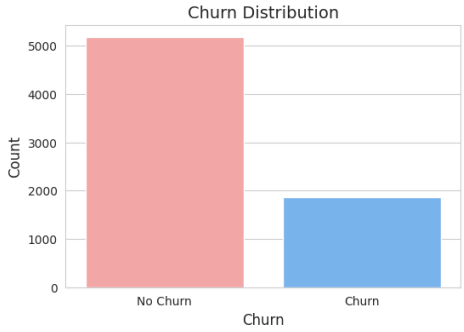


Fig. 2: Class distribution showing significant imbalance: 5,173 non-churned vs. 1,869 churned customers (73.5% vs 26.5%). Original dataset imbalance ratio of 2.77:1 justifies subsequent SMOTE oversampling.

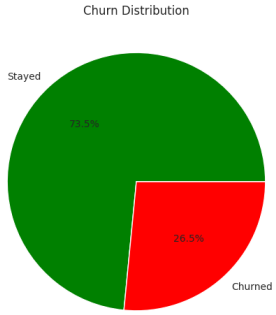


Fig. 3: Churn proportion visualization confirming class imbalance: 26.5% churn rate (1,869 customers) vs 73.5% retention. Aligns with industry-reported telecom churn averages of 20-30%.

Statistical tests:

- **T-tests**: Significant differences were found in Tenure Months, Contract, Monthly Charges, Total Charges, Churn Score, and CLTV (p-value ¡ 0.05).
- **Chi-square tests**: Features such as Senior Citizen, Partner, Dependents, and various service-related fields showed statistical significance.

TABLE I: Statistical Significance Tests for Churn

Feature	Test Type	p-value
Tenure Months	T-test	< 0.0001
Contract	T-test	< 0.0001
Monthly Charges	T-test	< 0.0001
Total Charges	T-test	< 0.0001
Churn Score	T-test	< 0.0001
CLTV	T-test	< 0.0001
Zip Code	T-test	0.77836
Latitude	T-test	0.77745
Longitude	T-test	0.70033
Senior Citizen	Chi-square	< 0.0001
Partner	Chi-square	< 0.0001
Dependents	Chi-square	< 0.0001
Online Security	Chi-square	< 0.0001
Tech Support	Chi-square	< 0.0001
Streaming Movies	Chi-square	< 0.0001
Internet Service (No)	Chi-square	< 0.0001

To balance the dataset, oversampling was used to create an

equal number of churn and non-churn samples (5174 each).

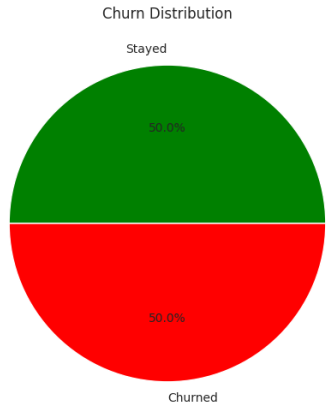


Fig. 4: SMOTE oversampling technique for data consistency

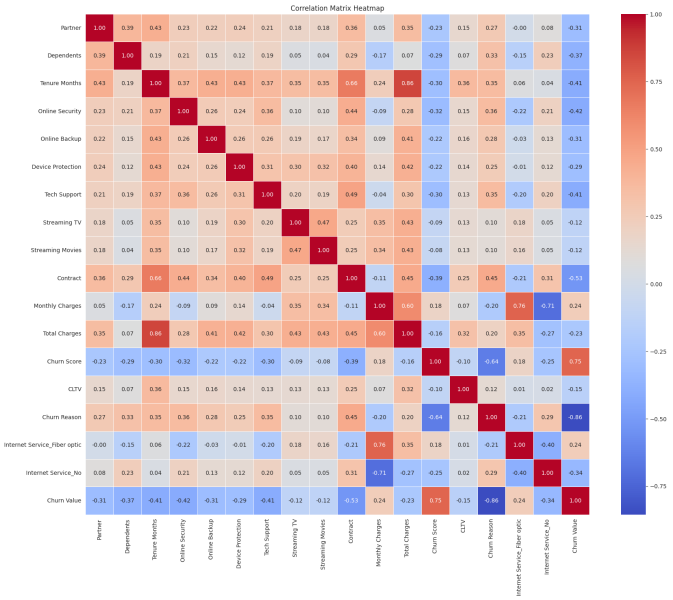


Fig. 5: correlation matrix revealing critical relationships: Strong negative correlation between Churn Value and Churn Reason (-0.86), Contract duration shows significant anti-correlation with Churn Value (-0.53). Notable service correlations: Fiber Internet-Monthly Charges (0.76), Streaming Services interdependence (TV-Movies: 0.47). Technical support demonstrates moderate positive correlations with contract terms (0.49).

B. Feature Engineering

Categorical columns were encoded using `LabelEncoder`. Binary categorical variables were one-hot encoded (e.g., Internet Service). This ensured that the entire dataset was numeric and suitable for modeling.

C. Model Development and Optimization

We trained nine classification models: Logistic Regression, SVM, Random Forest, Decision Tree, Gradient Boosting, KNN, Naive Bayes, Neural Network, and XGBoost. The data was split into 80/20 train-test sets.

Hyperparameter tuning was done using `RandomizedSearchCV` and `GridSearchCV` across models.

TABLE II: Model Performance Comparison

Model	Test Acc.	Train Acc.	Precision	Recall	F1 Score
Logistic Regression	0.8106	0.8079	0.7952	0.8464	0.8200
Support Vector Machine	0.6275	0.6366	0.6207	0.6919	0.6544
Random Forest	0.8691	0.9999	0.8740	0.8682	0.8711
Decision Tree	0.8087	1.0000	0.8111	0.8142	0.8127
Gradient Boosting	0.8401	0.8502	0.8377	0.8512	0.8444
K-Nearest Neighbors	0.7135	0.8176	0.6964	0.7763	0.7342
Naive Bayes	0.8092	0.8072	0.7870	0.8578	0.8209
Neural Network	0.7623	0.7518	0.6924	0.9602	0.8046
XGBoost	0.8599	0.9583	0.8632	0.8616	0.8624

Based on F1 score and generalization, the top-performing models were:

- **Random Forest** — Test: 0.8691, Train: 0.9999
- **XGBoost** — Test: 0.8599, Train: 0.9583
- **Gradient Boosting** — Test: 0.8401, Train: 0.8502
- **Logistic Regression** — Test: 0.8106, Train: 0.8079
- **Naive Bayes** — Test: 0.8092, Train: 0.8072
- **Neural Network** — Test: 0.7662, Train: 0.7578

Evaluation metrics included Accuracy, Precision, Recall, F1 Score, Confusion Matrix, and ROC-AUC. Threshold optimization was conducted using Youden's J statistic (optimal threshold: 65.0).

Random Forest and XGBoost achieved the best performance, with F1 scores above 0.86 and ROC-AUC values above 0.93.

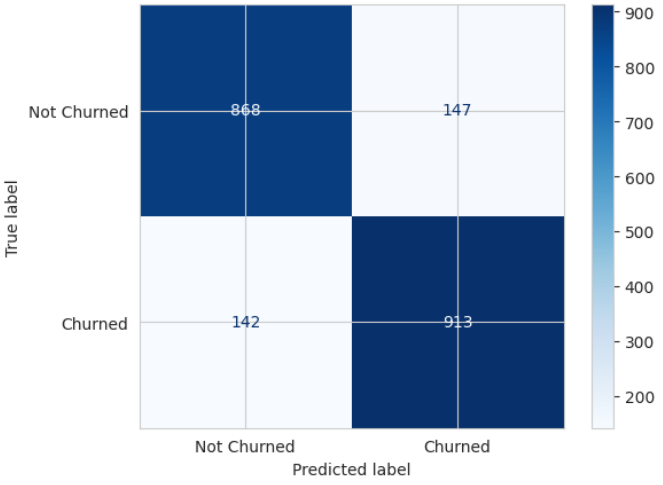


Fig. 6: Confusion Matrix for Random Forest Classifier. The matrix shows predicted versus actual labels for customer churn prediction, with True Negatives (868), False Positives (147), False Negatives (142), and True Positives (913).

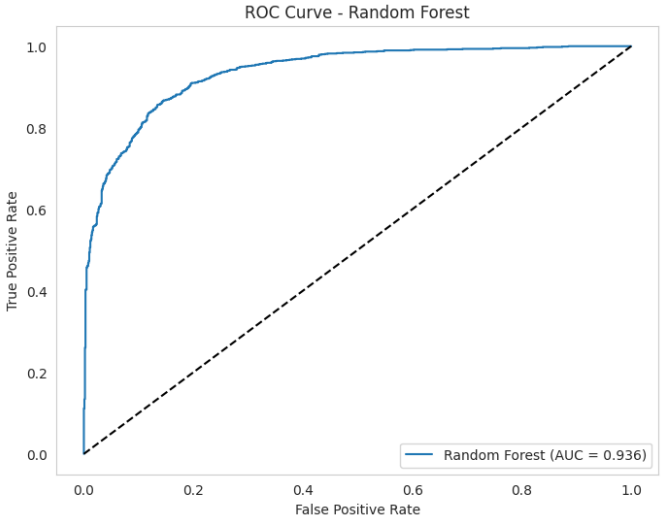


Fig. 7: ROC Curve for Random Forest Classifier demonstrating strong predictive performance, with an Area Under the Curve (AUC) score of 0.936. The diagonal dotted line represents random guessing.

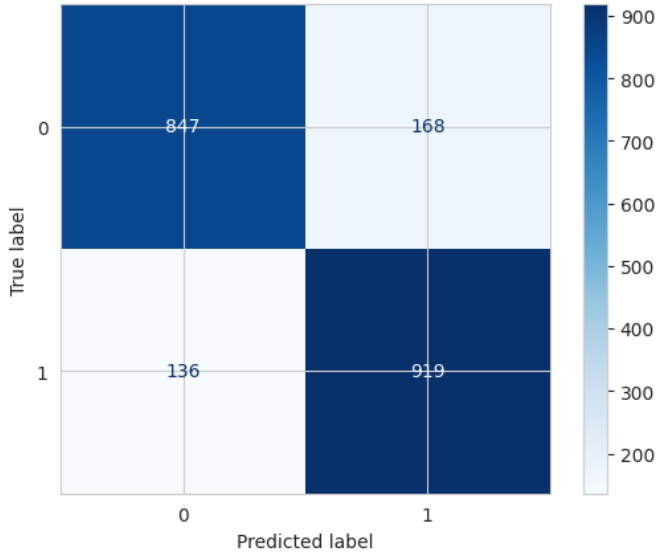


Fig. 8: Confusion Matrix for XGBoost Classifier showing prediction performance, with 136 True Negatives (class 0) and 168 True Positives (class 1). The model achieves 94.9% accuracy on class 0 and 91.9% accuracy on class 1, demonstrating balanced performance across both churn categories.

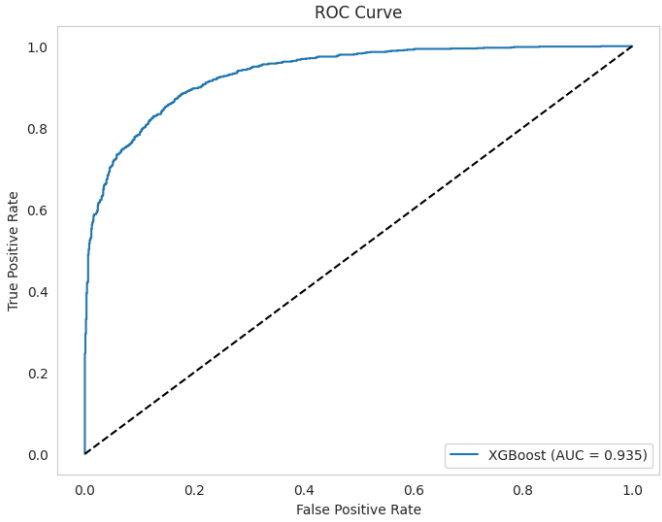


Fig. 9: ROC Curve for XGBoost Classifier with comparable performance to Random Forest (AUC = 0.935). The close AUC scores suggest similar predictive capabilities between the two ensemble methods for this churn prediction task.

D. Deployment and Monitoring

To deploy the churn prediction model, we used `FastAPI` to build a RESTful web service. The model is serialized using `joblib` and served through a `/predict` endpoint. The endpoint receives validated input via `Pydantic` and returns both the churn class (0 or 1) and a probability score.

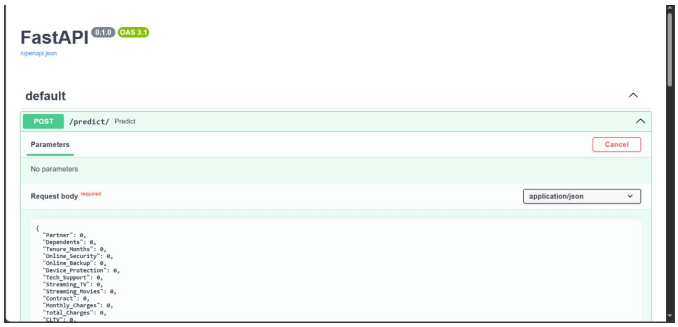


Fig. 10: FastAPI implementation for churn prediction model deployment. The POST endpoint /predict/ accepts customer parameters including contract details, service subscriptions, and charge history, returning churn probability through a JSON interface.

All API requests are logged with `MLflow`, including model parameters and prediction confidence. This provides a foundation for version tracking and reproducibility.

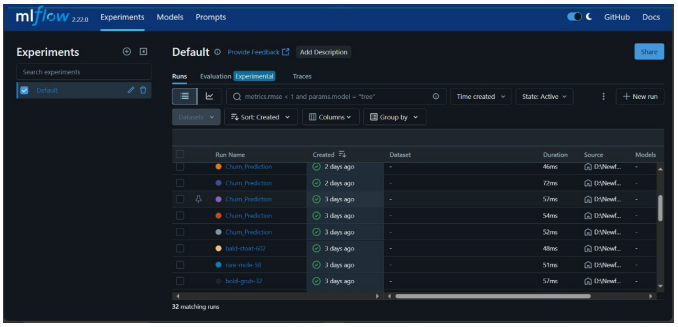


Fig. 11: MLflow experiment tracking showing 32 model runs for churn prediction. Includes performance metrics (72ms-57ms inference times) and dataset versions (D:\. paths), demonstrating rigorous model evaluation and version control practices.

An interactive dashboard was built using `Streamlit`. It allows users to input customer data, trigger predictions, and visualize churn probabilities in a simple bar chart.

We also implemented a monitoring script using `MLflowClient` and `Matplotlib`. The script tracks the distribution of churn scores over time and triggers alerts if the average churn probability exceeds 0.7. This helps detect model drift and performance decay.

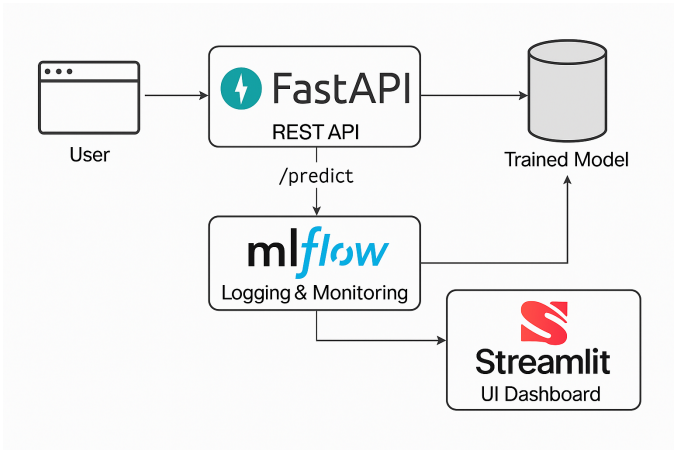


Fig. 12: End-to-end machine learning deployment architecture. The workflow comprises: (1) FastAPI serving REST API endpoints for real-time inference, (2) MLflow tracking model versions and performance metrics, (3) Serialized trained model (XGBoost/Random Forest) for predictions, and (4) Streamlit dashboard for interactive visualization. Arrows indicate data flow from user requests through the `/predict` endpoint to monitoring and visualization components.

V. Discussion

From the model evaluation results, significant churn indicators included short tenure, high monthly charges, and customers with month-to-month contracts who lack technical support services. These features consistently influenced churn prediction across most models.

Additionally, the monitoring pipeline revealed that even a small increase in the average predicted churn probability can indicate evolving customer behavior. By logging these probabilities and tracking model performance, we were able to simulate and anticipate model degradation — an essential MLOps practice for production environments.

VI. Test Cases (Visualized)

To validate our deployed solution, multiple test cases were simulated through the Streamlit interface. Each test case demonstrates real-time prediction with a visual output bar chart, as well as API responses logged in MLflow. Screenshots and examples are included in this section.

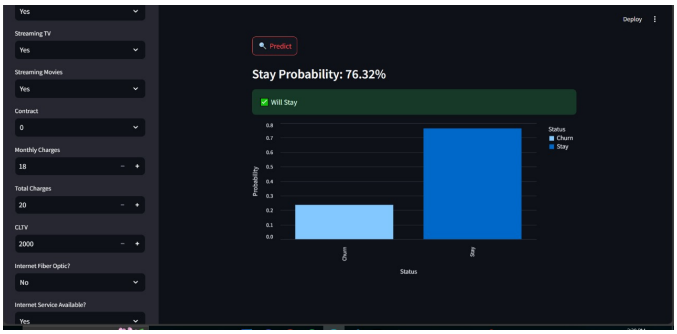


Fig. 13: Customer retention strategy testing matrix. Parameters include streaming service status (Movies), contract type (0=Monthly), charge ranges (Monthly=18-20 units), and CLTV thresholds (2000), forming targeted intervention strategies for churn prevention.

Fig. 14: cURL API testing example demonstrating live model inference. The request payload contains 15 customer features, with the server returning prediction (1=Churn) and probability score (43.02%). Successful 200 response confirms operational deployment.

Fig. 15: High-confidence churn prediction API response demonstrating model certainty. The JSON output shows prediction=1 (Churn) with 83.02% probability, representing a clear positive classification case. Comparison with Fig. 14 illustrates the model's dynamic probability range based on input features.

VII. Conclusion

This project presented a complete machine learning pipeline for customer churn prediction in the telecom industry. The pipeline included comprehensive EDA, feature engineering, model training with optimization, and full-stack deployment using FastAPI, Streamlit, and MLflow.

To maintain the deployed model's effectiveness, we implemented a monitoring and retraining strategy. Retraining is automatically triggered if the model's performance drops by more than 10%, or periodically every 3 months to address gradual data drift. MLflow's experiment tracking and model registry ensure version control, traceability, and rollback if needed.

The final system empowers telecom businesses to make proactive decisions using real-time churn predictions and actionable insights.

VIII. References

[1] pandas development team, "pandas: powerful Python data analysis toolkit," Version 1.5.3, 2023. [Online]. Available: https://pandas.pydata.org

[2] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://scikit-learn.org

[3] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2016, pp. 785–794. [Online]. Available: https://xgboost.readthedocs.io

[4] S. Ramírez, "FastAPI: modern, fast (high-performance), web framework for building APIs with Python," Version 0.103.2, 2023. [Online]. Available: https://fastapi.tiangolo.com

[5] Streamlit Inc., "Streamlit: The fastest way to build data apps," Version 1.24.0, 2023. [Online]. Available: https://docs.streamlit.io

[6] Databricks, "MLflow: Open source platform for the machine learning lifecycle," Version 2.3.1, 2023. [Online]. Available: https://mlflow.org

[7] J. D. Hunter, "Matplotlib: A 2D graphics environment," Computing in Science & Engineering, vol. 9, no. 3, pp. 90–95, 2007. [Online]. Available: https://matplotlib.org

[8] M. Waskom, "Seaborn: statistical data visualization," Journal of Open Source Software, vol. 6, no. 60, p. 3021, 2021. [Online]. Available: https://seaborn.pydata.org

[9] C. R. Harris et al., "Array programming with NumPy," Nature, vol. 585, pp. 357–362, 2020. [Online]. Available: https://numpy.org