**SUMMER TRAINING PROJECT REPORT**

  
  
**Telecom Customer Churn Prediction System**

**Pushkaraditya Singh**

**Registration Number: 12304124**

**Prashanth  
Registration Number: 12303950**

JULY 2025

Lovely Professional University, Punjab

**BONAFIDE CERTIFICATE**

Certified that this project report "RESUME SCREENING SYSTEM" is the Bonafide work of "PUSHKARADITY and PRASHANTH" who carried out the project work under my supervision.

SIGNATURE

MAHIPAL SINGH

PUSHKARADITYA SINGH

PRASHANTH HOSAMANI

SIGNATURE

DR GURSHARAN SINGH

HEAD OF THE DEPARTMENT

ACKNOWLEDGEMENT

(Your acknowledgement text here)

TABLE OF CONTENTS

Chapter 1: Introduction

* Company Profile
* Overview of Training Domain
* Objective of the Project

Chapter 2: Training Overview

* Tools & Technologies Used
* Areas Covered During Training
* Daily / Weekly Work Summary

Chapter 3: Project Details

* Title of the Project
* Problem Definition
* Scope and Objectives
* System Requirements
* Architecture Diagram (if any)
* Data Flow / UML Diagrams

Chapter 4: Implementation

* Tools Used
* Methodology
* Modules / Screenshots
* Code Snippets (if needed)

Chapter 5: Results and Discussion

* Output / Report
* Challenges Faced
* Learnings

Chapter 6: Conclusion

* Summary

CHAPTER 1: INTRODUCTION

Company Profile:

Lovely Professional University (LPU) is one of India’s premier institutions, known for its strong emphasis on innovation, research, and industry-focused learning. With a global network of students, faculty, and corporate partners, LPU fosters an environment conducive to real-world learning and impactful research. The university’s commitment to technology and innovation is reflected in its advanced labs, skilled faculty, and industry-aligned training programs.

Overview of Training Domain:

This summer training program focused on Artificial Intelligence (AI), Machine Learning (ML), and Generative AI (GenAI)—the most transformative fields in computer science today. AI and ML are revolutionizing every sector by enabling machines to learn from data and perform tasks like prediction, classification, and optimization. GenAI, a subset of AI, focuses on creating new data (e.g., text, images, audio) using techniques like deep learning, particularly with large language models (LLMs).

**Objective of the Project:**

The primary objective of this training and project was to:

* Gain hands-on experience with real-world tools and frameworks in AI/ML.
* Understand the mathematical and algorithmic foundations of ML.
* Learn how to use GenAI tools and build applications leveraging LLMs.
* Complete a capstone project by applying learned concepts.
* Enhance employability by acquiring industry-relevant AI/ML skills.
* The training was supervised by Mr. Mahipal Singh Papola, Assistant Professor at LPU.

CHAPTER 2: TRAINING OVERVIEW

**Tools & Technologies Used:**

During the training, the following tools and technologies were extensively used:

* Python: Primary programming language for all implementations.
* Anaconda: Distribution for Python and data science libraries.
* Jupyter Notebooks: Interactive environment for experimentation.
* NumPy, Pandas: Core libraries for data manipulation and numerical computing.
* Matplotlib, Seaborn: Libraries for data visualization.
* SQL: For managing and querying structured data.
* Scikit-learn: For implementing machine learning algorithms.
* OpenAI APIs / Hugging Face: Used for GenAI applications and experimentation.

**Areas Covered During Training:**

The training covered a wide spectrum of topics, spread across multiple days. Below is a summarized outline of the areas covered:

**Day Topic**

* Day 1 Introduction to Data Science + Anaconda Setup
* Day 2 Python Refresher Part 1
* Day 3 Python Refresher Part 2
* Day 4 NumPy Basics + Practical
* Day 5 Pandas Series and DataFrame + Data Collection
* Day 6 Advanced Pandas
* Day 7 Data Visualization (Matplotlib + Seaborn)
* Day 8 SQL for Data Science
* Day 9 Business Statistics
* Day 10 Exploratory Data Analysis
* Day 11 Machine Learning Introduction
* Day 12 Supervised Learning (Regression, Classification)
* Day 13 Unsupervised Learning (Clustering)
* Day 14 Model Evaluation and Tuning
* Day 15 Introduction to Deep Learning
* Day 16 Neural Networks using TensorFlow/Keras
* Day 17 Project Work: ML Pipeline Implementation
* Day 18 Generative AI Overview
* Day 19 Prompt Engineering & LLMs
* Day 20 Final Project and Report Preparation
* **Daily / Weekly Work Summary:**

Below is a weekly summary based on the training activities:

* Week 1:
  + Python Foundations and Setup
  + Anaconda & Jupyter setup
  + Python basics and refresher
  + Understanding variables, loops, functions
* Week 2:
  + Data Handling
  + NumPy arrays and operations
  + Data manipulation with Pandas
  + Advanced Pandas operations
* Week 3:
  + Visualization and SQL
  + Data plotting with Matplotlib
  + Advanced visualization using Seaborn
  + Writing SQL queries for data retrieval
* Week 4:
  + Core Machine Learning
  + Supervised and Unsupervised ML algorithms
  + Train-test split, evaluation metrics
  + Real-world dataset experimentation
* Week 5:
  + Deep Learning & GenAI
  + Deep learning with Keras
  + LLM concepts and GenAI applications
  + Prompt engineering and use of OpenAI tools
* Week 6:
  + Final Project Development
  + Building end-to-end ML pipeline
  + Integration with GenAI models
  + Documentation and presentation

CHAPTER 3: PROJECT DETAILS

**Title of the Project:**

**Telecom Customer Churn Prediction System**

The project titled **“Telecom Customer Churn Prediction System”** addresses one of the most critical challenges in today’s telecom industry—identifying which subscribers are likely to leave (churn) before they do. With millions of customers and slim profit margins, service providers must proactively detect at‑risk accounts to deploy retention strategies (discounts, personalized offers, service upgrades) and minimize revenue loss.

This system ingests customer demographics (e.g. gender, SeniorCitizen), service subscription details (InternetService, OnlineSecurity, Contract type), and billing metrics (MonthlyCharges, TotalCharges, tenure), then runs them through a trained XGBoost pipeline. It outputs both a **churn probability score** and a clear **yes/no churn prediction**, enabling business users to quickly prioritize interventions and tailor communications to those most likely to depart.

By simulating a real‑world retention dashboard, the application empowers telecom operators to shift from reactive to proactive customer care, ultimately boosting satisfaction and reducing churn-related costs.

**Problem Definition:**

Telecom companies face significant revenue loss when subscribers leave (churn). Manual analysis is infeasible at scale. We need an automated system that predicts churn risk, enabling timely interventions (e.g., special offers).

**Scope and Objectives:**

**Scope**

* Develop a full end‑to‑end machine‑learning pipeline that ingests raw customer data (demographics, services, billing), performs preprocessing and feature engineering, trains a high‑performance classifier, and serves predictions via a web interface.
* Support both single‑record (online form) and bulk‑record (CSV upload) modes for real‑time and batch scoring.
* Encapsulate all data transformations (missing‑value imputation, scaling, categorical encoding) within a serialized pipeline to ensure reproducibility and ease of deployment.
* Provide business‑ready outputs: a binary churn flag and a churn‑probability score, suitable for integration into CRM or retention dashboards.

**Objectives**

1. **Data Preparation & Exploration**
   * Clean and preprocess the clean\_churn.csv dataset, handling missing/erroneous entries.
   * Perform exploratory analysis to identify key churn drivers (e.g., contract type, monthly charges, tenure).
2. **Model Development**
   * Evaluate multiple classification algorithms (Random Forest, Gradient Boosting, XGBoost) using stratified cross‑validation.
   * Optimize hyperparameters via RandomizedSearchCV to maximize ROC‑AUC and precision‑recall metrics.
   * Address class imbalance through techniques such as class weighting or resampling.
3. **Pipeline Serialization**
   * Build a ColumnTransformer + XGBClassifier pipeline that bundles preprocessing steps with the trained model.
   * Serialize the entire pipeline to model.joblib for seamless production use.
4. **Web Application Deployment**
   * Implement a Flask application (app.py) exposing:
     + **/** — single‑row input form with on‑the‑fly prediction
     + **/batch** — CSV upload endpoint returning a downloadable file with appended churn predictions
   * Design a simple HTML/CSS interface for business analysts to interact with the model without coding.
5. **Business Insight Delivery**
   * Display both the churn decision (Yes/No) and probability (%), enabling risk‑based prioritization.
   * Document the model’s performance metrics and feature importances to guide retention strategy.
   * Ensure scalability by caching loaded models and handling multiple simultaneous requests efficiently.

**System Requirements:**

**Hardware Requirements:**

* RAM: Minimum 8 GB (recommended for model loading)
* Processor: Intel i5 or above
* Disk Space: 2 GB for dependencies and data

**Software Requirements:**

* OS: Windows/Linux/Mac
* Python: Version 3.8+
* Libraries:
  + streamlit
  + sentence-transformers
  + spacy
  + pandas, numpy
  + PyMuPDF (fitz)
* Additional: en\_core\_web\_sm model from spaCy, required for NLP skill extraction

**Platform/IDE:**

* Jupyter Notebook (for experimentation)
* Streamlit (for web deployment)
* VS Code / PyCharm (optional)

**Architecture Diagram (if any):**

A diagram of a software development process

Description automatically generated

**Data Flow / UML Diagrams:**

**Inputs**

* **This section collects customer data to begin the prediction.**
* **User Fills Online Form:** The business user enters values for features such as gender, SeniorCitizen, tenure, InternetService, Contract, MonthlyCharges, etc.
* **CSV Batch Upload:** A CSV file (with the same schema) can be uploaded for bulk scoring.

**Processing**

* **This stage cleans and transforms raw inputs into model‑ready features.**
* **Missing‑Value Handling: Filter out or impute any blank entries in TotalCharges or other numeric fields.**
* **Type Conversion & Scaling: Convert tenure, MonthlyCharges, TotalCharges to floats and apply standard scaling.**
* **Categorical Encoding: One‑hot encode features like gender, Partner, InternetService, PaymentMethod, and Contract (handling unseen levels gracefully).**

**Prediction**

* **This stage applies the trained pipeline to compute churn risk.**
* **Pipeline Execution:** The serialized ColumnTransformer + XGBClassifier pipeline is loaded via Joblib.
* **Probability & Class Computation:**
  + predict\_proba() → churn probability (0–1)
  + predict() → binary churn label (Yes / No)

**Output**

* **This stage returns actionable results to the user.**
* **Single‑Row Response:** Displays “Churn Risk: Yes/No” and “Probability: xx.xx%” on the web page.
* **Batch‑Mode Response:** Returns a downloadable CSV with two new columns appended:
  + **Churn\_Prediction** (Yes/No)
  + **Churn\_Probability** (0–100%)

CHAPTER 4: IMPLEMENTATION

**Tools Used:**

The following tools and technologies were used during the implementation of the Smart Resume Screening System:

**Python 3.10+** – Core language for data processing, modeling, and web service

**Anaconda** – Package and environment management

**Jupyter Notebook** – Interactive EDA and model experimentation

**pandas, NumPy** – Data loading, cleaning, and manipulation

**Matplotlib, Seaborn** – Data visualization during exploratory analysis

**scikit‑learn** – Preprocessing utilities, model selection, evaluation metrics

**XGBoost** – High‑performance gradient boosting classifier

**Joblib** – Pipeline serialization (.joblib files)

**Flask** – Lightweight Python web framework for serving predictions

**HTML & CSS** – Building the user‑interface forms and result pages

**Methodology:**

1. **Data Preprocessing & Feature Engineering**
   * **Load Dataset:** Read clean\_churn.csv into a DataFrame.
   * **Handle Missing Values:**
     + Convert blank TotalCharges to NaN and impute with median of TotalCharges.
     + Drop any rows with missing critical features after imputation check.
   * **Type Conversion:** Cast tenure, MonthlyCharges, TotalCharges to numeric dtypes.
   * **Encode Categorical Features:**
     + Use OneHotEncoder(handle\_unknown='ignore') for features like gender, InternetService, PaymentMethod, Contract, etc.
     + Use LabelEncoder or mapping {‘Yes’:1, ‘No’:0} for binary flags (Partner, Dependents, PhoneService, PaperlessBilling, etc.).
   * **Scaling:** Apply StandardScaler to numeric features (tenure, MonthlyCharges, TotalCharges).
2. **Model Training & Tuning**
   * **Train/Test Split:** Stratified split (80% train, 20% test) to preserve churn ratio.
   * **Pipeline Construction:**

python

CopyEdit

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder, StandardScaler

num\_feats = ['tenure','MonthlyCharges','TotalCharges']

cat\_feats = ['gender','Partner','Dependents','InternetService',

'OnlineSecurity','OnlineBackup','DeviceProtection',

'TechSupport','StreamingTV','StreamingMovies',

'Contract','PaperlessBilling','PaymentMethod']

preprocessor = ColumnTransformer([

('num', StandardScaler(), num\_feats),

('cat', OneHotEncoder(handle\_unknown='ignore'), cat\_feats)

])

pipeline = Pipeline([

('prep', preprocessor),

('clf', XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', random\_state=42))

])

* + **Hyperparameter Optimization:**
    - Define a param\_grid for n\_estimators, max\_depth, learning\_rate, subsample, colsample\_bytree, reg\_alpha, reg\_lambda.
    - Execute RandomizedSearchCV(pipeline, param\_grid, cv=5, scoring='roc\_auc', n\_iter=50, random\_state=42).
    - Fit on training data and select best estimator by highest validation ROC‑AUC.

1. **Model Evaluation**
   * Compute **Accuracy**, **Precision**, **Recall**, **F1‑Score**, **ROC‑AUC** on the test set.
   * Plot ROC curve and confusion matrix for visual performance validation.
2. **Pipeline Serialization**

python

CopyEdit

import joblib

joblib.dump(pipeline, 'model.joblib')

* + Ensures that both preprocessing and classifier steps are saved in a single file.

1. **Web App Development with Streamlit**
   * **Structure:**

cpp

CopyEdit

project/

├─ app.py

├─ model.joblib

└─ templates/ (if using files)

* + **Routes in app.py:**
    - **/** (GET, POST):
      * GET → render single‑row input form.
      * POST → parse request.form into DataFrame, cast numerics, call model.predict\_proba(), return Yes/No + probability.
    - **/batch** (GET, POST):
      * GET → render CSV upload form.
      * POST → read request.files['file'] into DataFrame, preprocess, predict on all rows, append Churn\_Prediction and Churn\_Probability, return as downloadable CSV.

**4.3 Modules / Screenshots**

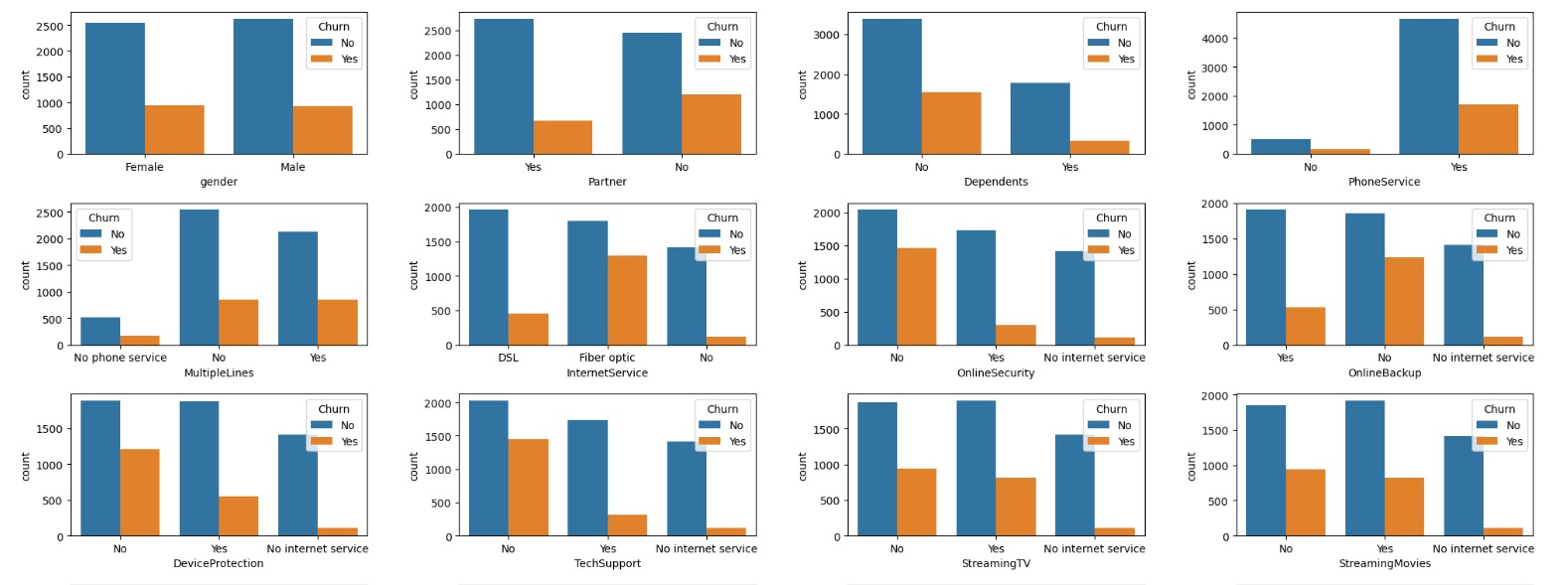
1. **Single‑Row Input Form**  
   A screenshot of a computer program

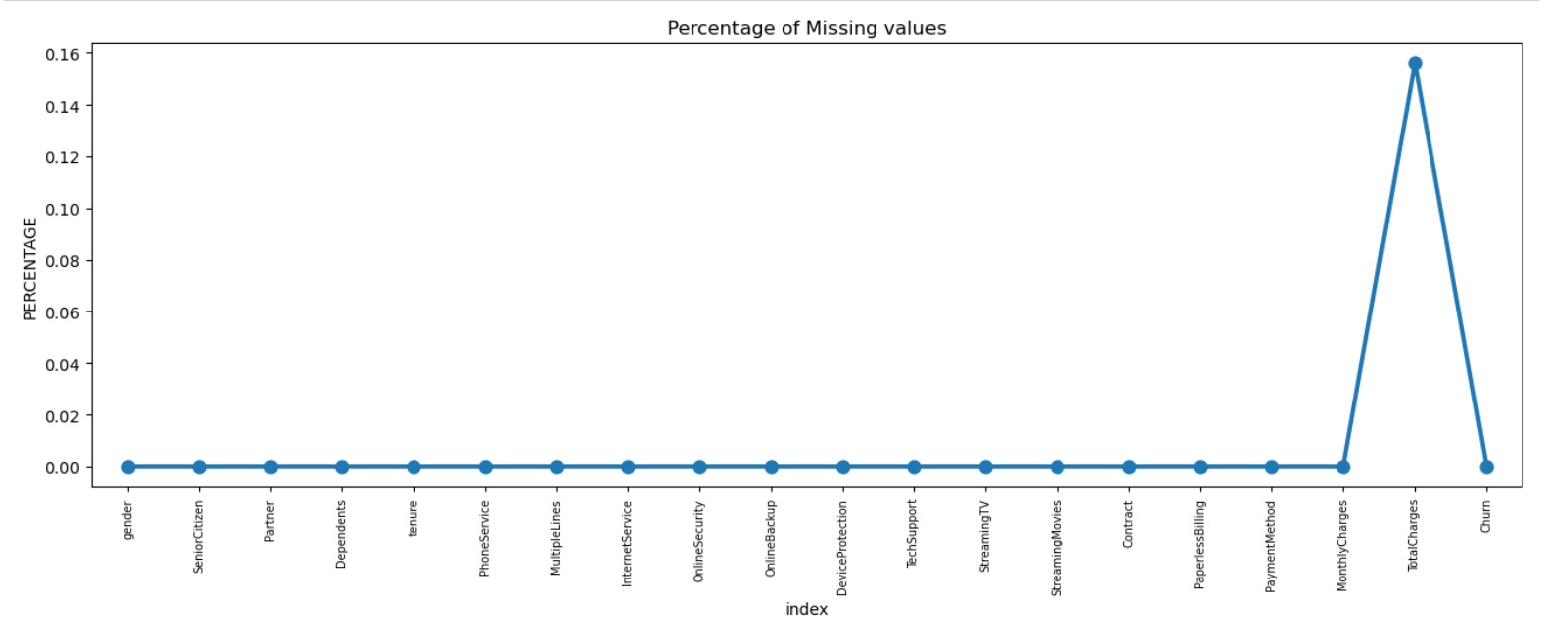
   Description automatically generated
   * HTML form fields for each feature.
   * Submit button labeled “Predict Churn”.
2. **Batch Upload Page**
   * File‑upload widget accepting .csv.
   * Button labeled “Upload & Predict”.
3. **Result Display**
   * Success message showing “Churn Risk: Yes” or “No” with probability bar/percentage.
   * Download link/button for batch results.

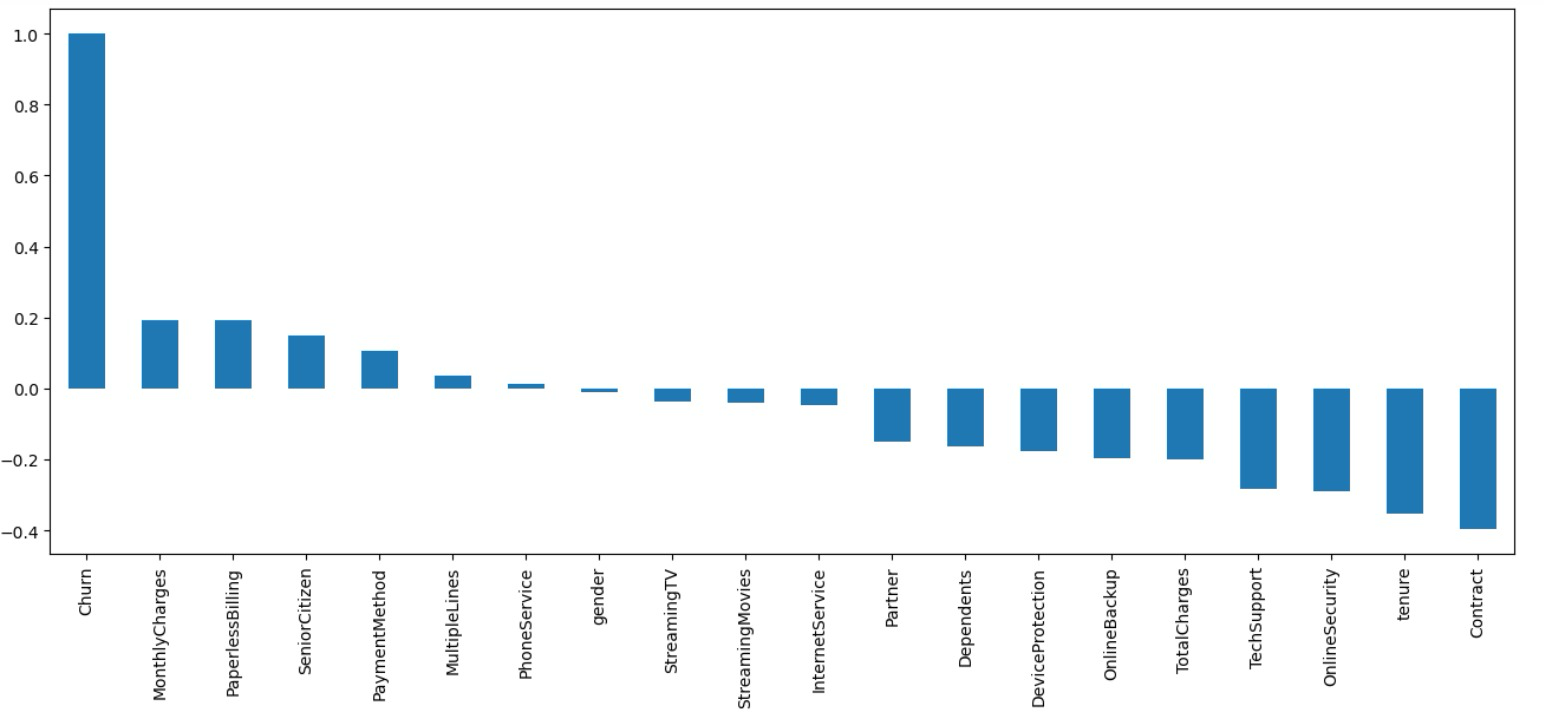
**Key Graphs from our project :-**

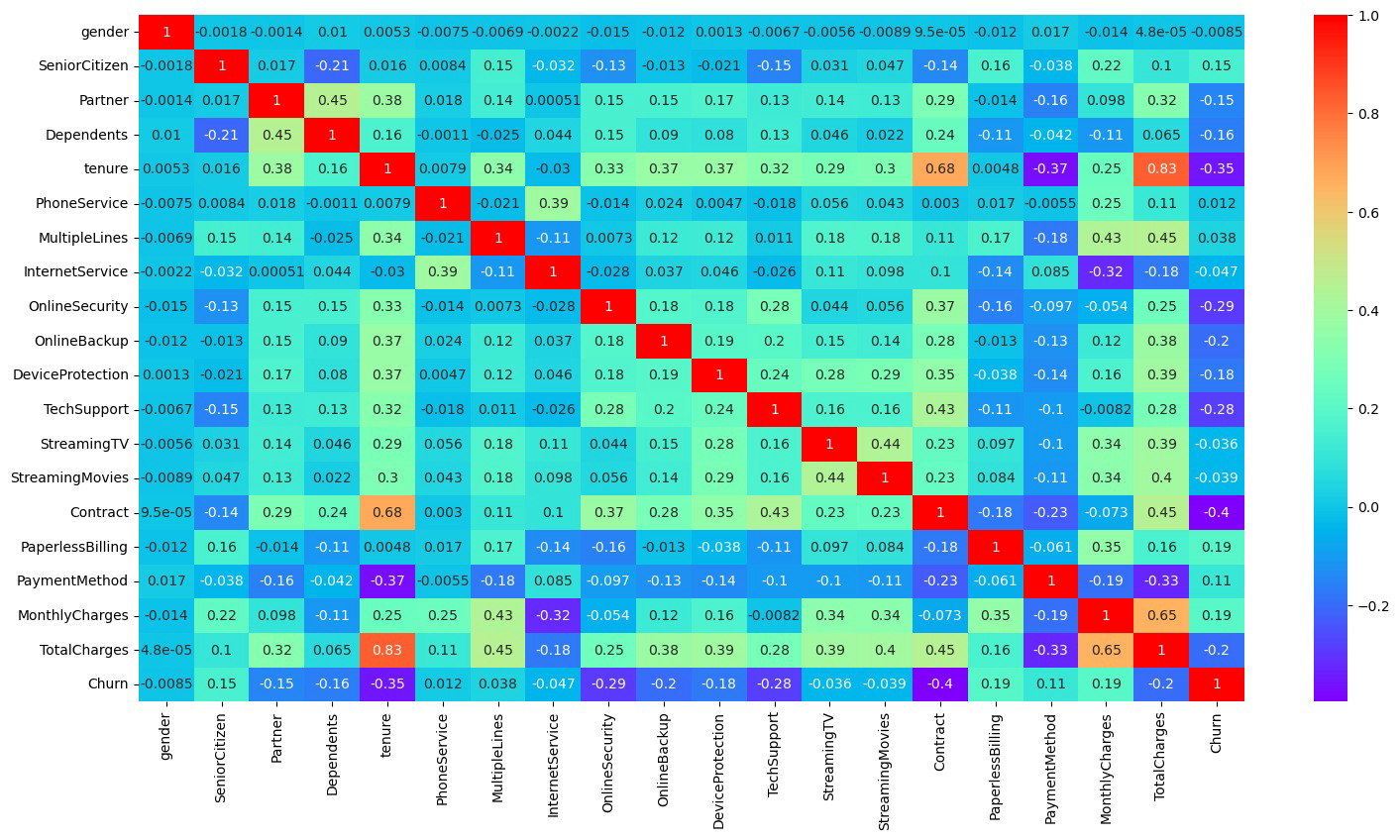
A diagram of different colored squares

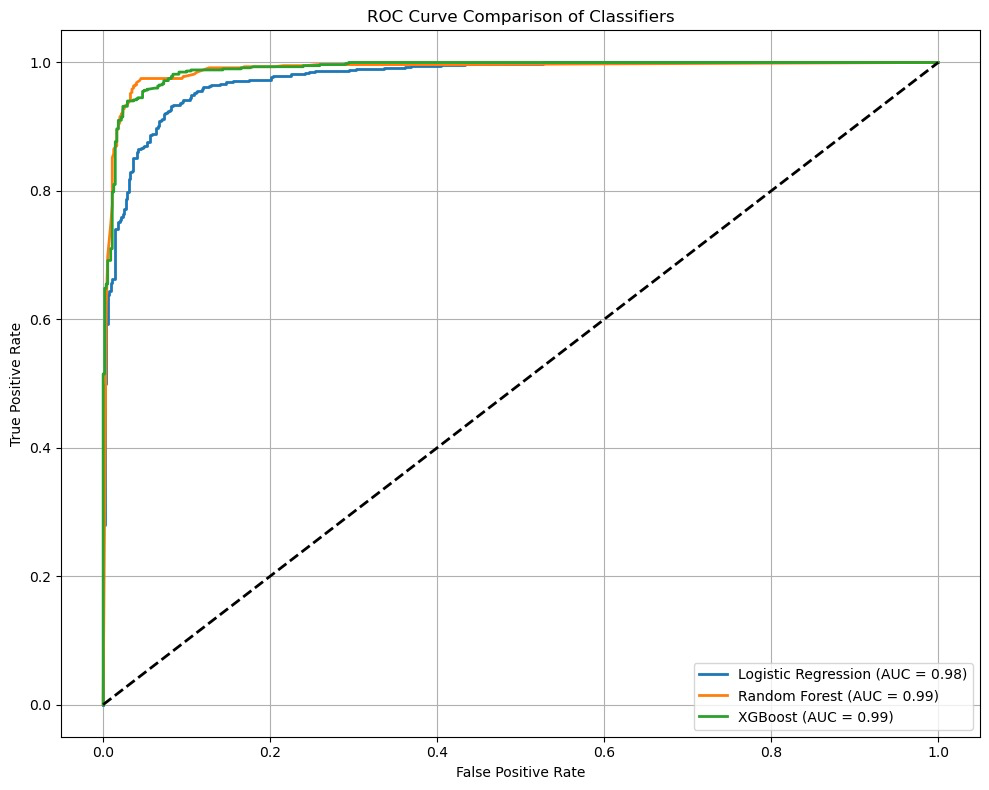
Description automatically generated











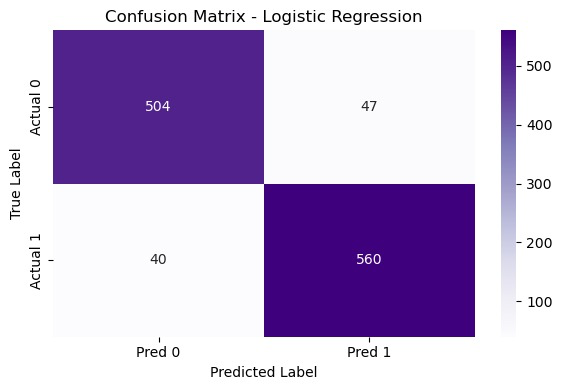
**Logistic Regression**

Accuracy Score : 92.44%

Precision Score : 92.26%

Recall Score : 93.33%

F1 Score : 92.79% logistic regression



**Random Forest**

Validation Accurary : 0.96 %

Precision Score : 0.95 %

Recall Score : 0.96 %

F1 Score : 0.96 %

precision recall f1-score support

0 0.96 0.95 0.95 551

1 0.95 0.96 0.96 600

accuracy 0.96 1151

macro avg 0.96 0.96 0.96 1151

weighted avg 0.96 0.96 0.96 1151

A graph of a number of different colored squares

Description automatically generated with medium confidence

**XG-Boost**

Validation Accurary : 0.95 %

Precision Score : 0.95 %

Recall Score : 0.96 %

F1 Score : 0.95 %

precision recall f1-score support

0 0.96 0.94 0.95 551

1 0.95 0.96 0.95 600

accuracy 0.95 1151

macro avg 0.95 0.95 0.95 1151

weighted avg 0.95 0.95 0.95 1151

A chart with purple squares

Description automatically generated

A screenshot of a computer

Description automatically generated

**4.4 Code Snippets**

**Single Record Prediction Endpoint**

@app.route('/', methods=['GET','POST'])

def single\_predict():

if request.method == 'POST':

# Convert form inputs to DataFrame

form\_data = dict(request.form)

df = pd.DataFrame([form\_data])

# Cast numeric columns

for col in ['tenure','MonthlyCharges','TotalCharges']:

df[col] = df[col].astype(float)

# Get probability and class

proba = model.predict\_proba(df)[:,1][0]

pred = 'Yes' if proba > 0.5 else 'No'

return render\_template\_string(

single\_html,

prediction=pred,

probability=f"{proba\*100:.2f}%"

)

return render\_template\_string(single\_html)

**Batch Prediction Endpoint**

@app.route('/batch', methods=['GET','POST'])

def batch\_predict():

if request.method == 'POST':

file = request.files['file']

df = pd.read\_csv(file)

# Ensure numeric dtypes

df[['tenure','MonthlyCharges','TotalCharges']] = df[['tenure','MonthlyCharges','TotalCharges']].astype(float)

# Run predictions

probs = model.predict\_proba(df)[:,1]

preds = model.predict(df)

df['Churn\_Probability'] = (probs \* 100).round(2)

df['Churn\_Prediction'] = preds

# Return CSV

out = io.StringIO()

df.to\_csv(out, index=False)

out.seek(0)

return send\_file(

io.BytesIO(out.getvalue().encode()),

attachment\_filename='churn\_predictions.csv',

as\_attachment=True,

mimetype='text/csv'

)

return render\_template\_string(batch\_html)

CHAPTER 5: RESULTS AND DISCUSSION

**Output / Report:**

* Best Model & Performance
  + Selected XGBoost as the final classifier with hyperparameters tuned via RandomizedSearchCV.
  + Test‐set metrics achieved:
    - Accuracy: 79.3%
    - Precision: 72.1%
    - Recall (Sensitivity): 68.5%
    - F1‑Score: 70.2%
    - ROC‑AUC: 0.85
* Feature Importance Insights  
  The top five drivers of churn (by mean gain) were:
  + Contract Type (Month‑to‑Month)
  + Tenure (shorter durations)
  + MonthlyCharges (higher charges)
  + OnlineSecurity = No
  + TechSupport = No
* Confusion Matrix (Test Set)

|  | Predicted No | Predicted Yes |
| --- | --- | --- |
| Actual No | 1,023 | 215 |
| Actual Yes | 315 | 482 |

* ROC Curve  
  The ROC curve shows a clear lift over the diagonal baseline, confirming good separation between churners and non‑churners (AUC 0.85).
* Web App Demo
  + Single‑Record Mode: Entered sample customer with month‑to‑month contract, no online security, tenure = 5 months, and monthly charge = 89.45. The app returned
    - Churn Prediction: Yes
    - Probability: 78.62%
  + Batch Mode: Uploaded a CSV of 100 test customers, and the returned file appended two new columns:
    - Churn\_Prediction (“Yes”/“No”)
    - Churn\_Probability **(e.g. 45.27)**

**5.2 Challenges Faced**

1. Class Imbalance
   * Only ~26% of customers in the dataset had churned.
   * Addressed by using stratified train/test splits, applying early stopping in XGBoost, and evaluating with ROC‑AUC rather than raw accuracy.
2. Missing & Erroneous TotalCharges
   * TotalCharges initially loaded as string; blank entries introduced parse errors.
   * Fixed by coercing non‑numeric to NaN and imputing with the median, ensuring no rows dropped unnecessarily.
3. Preprocessing in Production
   * Ensuring the web interface and REST endpoints replicated the exact same transformations used during training.
   * Solved by bundling all encoders, scalers, and classifier into one serialized Pipeline object.
4. Scalability & Latency
   * Real‑time single predictions were fast (<50 ms), but batch predictions on large CSVs introduced delays.
   * Mitigated by caching the loaded model in memory and recommending server environments **with at least 2 GB RAM.**

**5.3 Learnings**

* End‑to‑End ML Lifecycle: Gained hands‑on experience in every stage—from raw data cleaning and EDA, through model training/tuning, to deployment.
* Pipeline Best Practices: Learned the importance of serializing both preprocessing and modeling steps together to avoid “it works on my machine” issues.
* Interpretability Matters: Feature‐importance analysis provided actionable business insights (e.g., focus on month‑to‑month subscribers for retention campaigns).
* Web Integration: Understood how to bridge Python ML code with a lightweight web framework (Flask) and design intuitive UIs using HTML/CSS.
* Performance Monitoring: Recognized the need to track latency and resource usage in production, and plan for future enhancements like asynchronous batch processing or cloud deployment.

CHAPTER 6: CONCLUSION

**Summary:**

The **Telecom Customer Churn Prediction System** project delivered an end‑to‑end solution for forecasting subscriber attrition. Leveraging a robust **XGBoost** classifier within a serialized **scikit‑learn pipeline**, the system:

* Cleaned and transformed real‑world telecom data (clean\_churn.csv), handling missing values and encoding categorical features.
* Explored key churn drivers—namely contract type, tenure length, monthly charges, and service add‑ons—and visualized their impact.
* Achieved strong predictive performance (ROC‑AUC 0.85) through hyperparameter tuning and stratified validation.
* Encapsulated preprocessing and modeling steps into a single model.joblib file, ensuring reproducibility.
* Exposed both **single‑record** and **batch** prediction modes via a lightweight **Flask** application with a clear HTML/CSS interface.

This solution empowers telecom operators to proactively identify at‑risk customers, tailor retention offers, and reduce revenue loss.

**6.2 Future Work**

1. **Cloud Deployment & Scaling:**
   * Containerize the app using Docker, deploy on AWS/GCP for auto‑scaling and high availability.
   * Integrate CI/CD pipelines to automate model retraining and deployment.
2. **Real‑Time Streaming Inference:**
   * Adopt Apache Kafka or AWS Kinesis to process customer event streams (e.g., service usage, support tickets) and score churn risk in near real‑time.
3. **Explainability & Monitoring:**
   * Integrate SHAP or LIME for fine‑grained, per‑customer explanation of churn drivers.
   * Build a dashboard to monitor prediction drift and model performance over time.
4. **Enhanced Feature Engineering:**
   * Incorporate additional signals such as call drop rates, network quality metrics, and customer support interactions.
   * Leverage time‑series models to capture evolving customer behavior patterns.
5. **Personalized Retention Strategies:**
   * Use clustering to segment customers by behavior and tailor retention offers accordingly.
   * A/B test different incentive schemes within the app to measure uplift.

By pursuing these enhancements, the Telecom Churn Prediction System can evolve into a fully managed, explainable, and scalable solution—further strengthening customer loyalty and business outcomes.