CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING

(Ministry of Electronics and Information Technology

Meit, India)

PROJECT REPORT ON

Bank Customer Churn Prediction and Insights:

Using Python, ML, Streamlit, AWS & Tableau

Post Graduate Diploma

In

Big Data Analytics

Submitted by:

Sr.	PRN	Name
1	240870625002	Arnav Banerjee
2	240870625004	Ravi Kulkarni

CERTIFICATE OF APPROVAL

This is to Certify that the project report entitled **Bank Customer Churn Prediction and Insights: Using Python, ML, Streamlit, AWS & Tableau** submitted by **Arnav Banerjee & Ravi Kulkarni** is a bonafide work carried out by them under the supervision of **Mr. Sangam Chaturvedi, Scientist D, ICT&S, C-DAC Kolkata and Dr. Asok Bandyopadhyay, Group Head, Scientist- F, ICTS C-DAC, Kolkata.** It is approved for the partial fulfilment of the requirement Centre for *Development of Advanced Computing (C-DAC) Kolkata,* for the completion of report. This project report has not been earlier submitted to any other Institute or University for the award of any degree or diploma.

11/2/2025.

Mr. Sangam Chaturvedi
Project Guide
Scientist D, ICT&S
C-DAC, Kolkata

AsokBandyopadhyay
11/02/2025

Dr. Asok Bandyopadhyay
Group Head, ICT
Scientist - F
C-DAC, Kolkata

DECLARATION

We hereby declare that the project work entitled Development of AI Enabled Identification of Fake News using Textual Content Analysis **submitted** to the Centre for Advance Computing(C-DAC), Kolkata, is a record of an original work done by us under the guidance of **Mr. Sangam Chaturvedi, Scientist D, ICT&S, C-DAC Kolkata** and under the supervision of **Dr. Asok Bandyopadhyay, Group Head, Scientist -F, ICTS C-DAC, Kolkata**. This project work has been performed for the award of ***Post Graduate Diploma in Big Data*** **Analytics** ***(PG-DBDA) course*** of C-DAC, Kolkata only and this or any similar project will not be used for any other Degree or Diploma's associateship / fellowship

Name	**PRN**	**Signature**
Arnav Banerjee	240870625002	
Ravi Kulkarni	240870625004	

ACKNOWLEDGEMENT

We express our utmost sincere regards and deep gratitude to our project guide **Mr. Sangam Chaturvedi, Scientist D, ICT&S, C-DAC Kolkata** for his valuable guidance towards us to navigate through the difficulty of project execution at all times.

We are very much thankful to D**r. Asok Bandyopadhyay, Group Head, ICT, Scientist -F, C-DAC, Kolkata**, for their valuable support and guidance. We are also highly obliged to **Centre Head C-DAC, Kolkata**, for their unconditional help and inspiration for giving us a wonderful opportunity to enhance our skills. Finally, we would like to extend our thanks to all those who have contributed, directly or indirectly to make this project successful.

CONTENT

5. **Results and Analysis**
    - 5.1 Comparison of Classification Models
    - 5.2 Performance Metrics Comparison
    - 5.3 Feature Importance Using SHAP
6. **Deployment**
    - 6.1 Storing the model in pickle format.
    - 6.2 Streamlit App Deployment
    - 6.3 Tableau Dashboard Integration
    - 6.4 GitHub Repository Management
    - 6.5 AWS EC2 Hosting
7. **Interface**
8. **Future Scope**
9. **Conclusion**
10. **References**

GitHub Link: https://github.com/ravikulkarni1501/CustomerChurn.git

1. ABSTRACT

Customer churn is a significant challenge in the banking sector, affecting revenue and customer retention strategies. This project aims to predict customer churn using Machine Learning (ML) techniques and provide actionable insights for decision-making. The dataset comprises customer demographics, account details, and transactional behaviours.

Various ML models, including Logistic Regression, Random Forest, and XGBoost, have been evaluated using appropriate performance metrics such as precision, recall, and F1-score. Additionally, SHAP analysis has been utilised to identify the most influential factors contributing to customer churn, such as current balance, average monthly balance, and transaction history.

For practical implementation, the final model has been deployed using Streamlit to enable real-time predictions. A Tableau dashboard has been integrated into the Streamlit app, providing interactive visualisations to enhance data-driven decision-making. Furthermore, the application is hosted on AWS, ensuring scalability and accessibility.

Machine learning is a branch of artificial intelligence (AI) that enables computers to learn from data and make decisions without explicit programming. It allows systems to improve their performance over time by identifying patterns and making predictions.

2. INTRODUCTION

Globalization processes and market deregulation policies are rapidly changing the competitive environments of many economic sectors. The appearance of new competitors and technologies leads to an increase in competition and, with it, a growing preoccupation among service providing companies with creating stronger customer bonds. In this context, anticipating the customer's intention to abandon the provider, a phenomenon known as churn, becomes a competitive advantage. (1) It is a crucial business metric, especially in industries like banking, telecommunications, and e-commerce, where customer retention directly impacts revenue and growth. Understanding the reasons behind churn helps organisations devise strategies to improve customer satisfaction and loyalty. A business incurs much higher charges when attempting to win new customers than to retain existing ones. As a result, much research has been invested into new ways of identifying those customers who have a high risk of churning. (2)

The number of service providers are being increased very rapidly in every business. In these days, there is no shortage of options for customers in the banking sector when choosing where to put their money. (4) Predicting churn enables banks to proactively identify at-risk customers and implement targeted retention strategies, such as personalised offers, better financial advisory services, and improved customer engagement. In customer churn prediction in banking or other sectors, for example, a scoring approach supports the calculation of a potential churn probability per customer is based on their past data. The need to retain existing customers to maintain market share has led to an increased need for the development of various machine learning techniques for churn customer analysis. (3)

The goal of this project is to analyse customer churn patterns, identify influential factors, and build a predictive model that can accurately classify customers as likely to churn or retain. Good prediction models have to be constantly developed and a combination of the proposed methods has to be used. (15) By leveraging machine learning, we aim to provide banks with actionable insights that help mitigate churn and enhance customer experience. In future, with the upcoming concepts and frameworks in the field of reinforcement learning and deep learning sector, machine learning is proving to be one of the most efficient way to address problems like churn prediction with better accuracy and precision. (17)

2.1 Machine Learning

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without being explicitly programmed. (5) It allows systems to improve their performance over time by identifying patterns and making predictions.

Machine learning is broadly classified into three types:

- **Supervised learning-** The defining characteristic of supervised learning is the availability of annotated training data. (6)
- **Unsupervised learning-** Unsupervised learning studies how systems can learn to represent particular input patterns in a way that reflects the statistical structure of the overall collection of input patterns. (7)
- **Reinforcement learning-** Reinforcement learning is an extension of classical dynamic programming in that it greatly enlarges the set of problems that can practically be solved (8)

In the context of churn prediction, machine learning plays a crucial role in analysing customer behaviour, identifying potential churn indicators, and providing actionable insights. Techniques such as logistic regression, decision trees, random forests, and XGBoost are commonly used to build predictive models that help businesses proactively retain customers.

2.2 Python:

Python is a high-level, interpreted programming language known for its simplicity and versatility. It is a powerful, elegant programming language that is easy to read and to understand. It demonstrates most of these features common to lots of other languages and is useful for real-world applications, to boot! (9) It is widely used in data science, machine learning, web development, automation, and more due to its readable syntax and extensive library ecosystem. Python supports multiple programming paradigms, including object-oriented, functional, and procedural programming.

For data science and machine learning, Python is the go-to language due to libraries like **pandas** for data manipulation, **numpy** for numerical computing, **scikit-learn** for machine learning algorithms, and **matplotlib** and **seaborn** for data visualization. Python's ease of integration with other technologies and its broad community support make it a popular choice for professionals across various domains.

2.3 Streamlit:

Streamlit is an open-source Python library that enables developers to create beautiful, interactive web applications quickly and with minimal coding. It was

designed to help data scientists and machine learning engineers to turn data scripts into shareable web apps with ease. Streamlit's primary appeal is its simplicity and efficiency in creating data-driven applications. (10)

Streamlit's core appeal lies in its ability to turn Python scripts into fully functional web applications with just a few lines of code. It supports integration with popular data science libraries such as **pandas**, **matplotlib**, **seaborn**, and **scikit-learn**, making it easy to build interactive dashboards, visualizations, and even machine learning models directly in the browser.

One of Streamlit's standout features is its real-time interactivity. Changes to inputs or data are automatically reflected in the application, allowing users to experiment with the model's parameters or visualize data updates instantly. This makes it an ideal tool for showcasing machine learning models, data analysis projects, and any application that requires quick prototyping and deployment without the need for extensive front-end development.

2.4 Tableau:

Tableau is a leading data visualization tool used for business intelligence and data analytics. It allows users to create interactive, shareable dashboards that display trends, patterns, and insights from data. Tableau's powerful drag-and-drop interface makes it accessible to both technical and non-technical users, enabling them to analyze and visualize data without requiring advanced coding skills.

Tableau can connect to various data sources, including spreadsheets, databases, cloud services, and big data platforms. It offers a variety of visualizations, such as bar charts, line graphs, scatter plots, and heat maps, which help users better understand complex datasets. Tableau's real-time data processing capability and

user-friendly design make it an essential tool for data-driven decision-making in business environments.

2.5 AWS (Amazon Web Services):

Amazon Web Services (AWS) is a comprehensive and widely adopted cloud platform offered by Amazon. It provides a wide range of cloud computing services, including computing power, storage, databases, machine learning, and analytics, all delivered on-demand over the internet. AWS enables businesses and developers to scale their applications efficiently and cost-effectively without the need for significant infrastructure investments.

Some of the key services offered by AWS include **Amazon EC2** (for scalable computing power), **Amazon S3** (for object storage), **Amazon RDS** (for relational databases), **AWS Lambda** (for serverless computing), and **Amazon SageMaker** (for building, training, and deploying machine learning models). AWS also provides a set of tools for data analysis, security, and monitoring, making it a versatile solution for various use cases, including big data processing, artificial intelligence, and Internet of Things (IoT).

AWS's flexibility allows users to choose from a pay-as-you-go pricing model, which helps optimize costs based on the specific needs of their applications. Additionally, AWS's global infrastructure, including data centers in multiple regions around the world, ensures high availability and reliability for applications running in the cloud.

Amazon EC2:

Amazon **EC2 (Elastic Compute Cloud)** is a web service that provides scalable and flexible computing power in the cloud, allowing users to run virtual servers as

per their requirements. It enables businesses and developers to deploy applications quickly without the need for expensive physical infrastructure. EC2 offers different instance types suited for various workloads, supports auto-scaling to manage fluctuating traffic, and integrates with other AWS services for security, networking, and storage. Users can choose from operating systems like Ubuntu, Windows, or Amazon Linux and configure instances based on their specific needs. With its pay-as-you-go pricing model, EC2 ensures cost efficiency, making it a reliable and scalable solution for cloud computing.

2.6 Classification Models:

Logistic Regression:

Logistic regression is a statistical method used for binary classification tasks, where the outcome variable is categorical with two possible outcomes (e.g., yes/no, true/false, 0/1). Logistic regression models are used to study effects of predictor variables on categorical outcomes. (11) It models the relationship between one or more independent variables (predictors) and a binary dependent variable by estimating probabilities using a logistic function.

The core idea behind logistic regression is to apply a **logistic function (sigmoid function)** to the linear combination of input variables. This function maps any real-valued number into the range of 0 to 1, which can then be interpreted as a probability.

Logistic regression is widely used for applications such as fraud detection, customer churn prediction, and medical diagnoses. It is popular due to its simplicity, interpretability, and the fact that it provides a probabilistic output, which can be useful for decision-making processes.

Random Forest:

Random Forest is versatile enough to deal with both supervised classification and regression tasks. (12) It operates by constructing a collection of decision trees during training and outputs the prediction that is the majority vote (for classification) or average (for regression) of the individual trees.

The key advantage of Random Forest over a single decision tree is its ability to reduce overfitting and improve generalization by combining the predictions of multiple trees. It does this by using **bootstrap aggregation (bagging)**, where each tree is trained on a random subset of the training data with replacement, and each tree also considers a random subset of features when making splits.

Random Forest models are highly accurate and robust, capable of handling large datasets with high dimensionality and providing good performance even with missing values. Additionally, they can be used for feature importance estimation, helping identify which features have the most impact on the prediction.

While Random Forest is computationally more expensive than a single decision tree, it typically provides a much more reliable and stable model. It is widely used in various applications, including financial modeling, medical diagnosis, and customer segmentation.

XGBoost:

XGBoost (Extreme Gradient Boosting) is a powerful and highly efficient machine learning algorithm that belongs to the family of **gradient boosting** methods. The main goal of boosting is to improve classification performance through the combination of decisions from many classification models, which are called weak classifiers (or weak learning algorithms). (16) It is widely used for classification

and regression tasks and is known for its speed and performance in a variety of predictive modeling competitions. We provide insights on cache access patterns, data compression and sharding to build a scalable tree boosting system. By combining these insights, XGBoost scales beyond billions of examples using far fewer resources than existing systems. (13)

XGBoost builds an ensemble of decision trees sequentially, where each tree is trained to correct the errors (residuals) made by the previous trees. This iterative process improves the model's accuracy by focusing on the misclassified data points. Unlike traditional boosting methods, XGBoost optimizes both the gradient and the second-order derivatives (using **second-order gradient optimization**) to achieve better accuracy and faster convergence.

One of the key features of XGBoost is its **regularization techniques**, such as L1 (Lasso) and L2 (Ridge) regularization, which help prevent overfitting and improve the model's generalization ability. It also supports handling missing values, parallel computation, and can work well with imbalanced datasets by adjusting the weight of different classes.

Among machine learning algorithms, Extreme Gradient Boosting (XGBoost) is a recently popular prediction algorithm in many machine learning challenges as a part of ensemble method which is expected to give better predictions with imbalanced-classes data, a common characteristic of customers churn data. (14) XGBoost has become a popular choice for structured/tabular data tasks in machine learning competitions and real-world applications due to its high accuracy, scalability, and robustness.

2.7 Libraries Used:

1. pandas:

Used for data manipulation and analysis, providing easy-to-use data structures like DataFrames.

2. numpy:

Provides support for arrays and mathematical operations, essential for handling numerical data.

3. scikit-learn (sklearn):

- **LabelEncoder**: Converts categorical labels into numeric format.
- **MinMaxScaler** & **StandardScaler**: Scales features to improve model performance.
- **train_test_split**: Splits data into training and testing sets.
- **RandomForestClassifier** & **LogisticRegression**: Machine learning models for classification tasks.
- **XGBClassifier**: Efficient gradient boosting model for high performance.

4. matplotlib & seaborn:

Used for data visualization. **matplotlib** creates basic plots, while **seaborn** provides more advanced, aesthetically pleasing charts.

5. SMOTE:

Generates synthetic data for the minority class to handle class imbalance.

6. PCA:

Reduces the number of features while maintaining data variance, improving model efficiency.

7. pickle:

Serializes models and objects for saving and loading in later sessions.

8. streamlit:

Creates interactive web apps for real-time model interaction and visualization.

9. plotly.express:

Generates interactive and visually appealing plots for enhanced data exploration.

10. SHAP (SHapley Additive exPlanations):

Used for model interpretability, SHAP explains individual predictions by quantifying the contribution of each feature to the prediction.

2.8 Evaluation Metrics for Classification Models:

When evaluating classification models, we use several key metrics to measure performance. These metrics help assess the model's ability to make correct predictions and handle class imbalances effectively.

Accuracy:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Measures the overall correctness of a model.
- **Strength:** Useful when the dataset is balanced.
- **Weakness:** Can be misleading in imbalanced datasets, where one class dominates.

Precision:

$$Precision = \frac{TP}{TP + FP}$$

- Represents how many of the positive predictions were actually correct.
- **Strength:** Reduces false positives.
- **Weakness:** May ignore false negatives.

Recall:

$$Recall = \frac{TP}{TP + FN}$$

- Measures how many actual positives were correctly identified.
- **Strength:** Reduces false negatives.
- **Weakness:** May increase false positives.

F1-Score:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- Harmonic mean of precision and recall.
- **Strength:** Provides a balance between precision and recall.
- **Weakness:** Does not directly account for true negatives.

Confusion Matrix:

A confusion matrix is a table used to evaluate a model's predictions.

	Predicted Negative	**Predicted Positive**
Actual Negative	Correct (TN)	False Positive (FP)

Actual Positive	False Negative (FN)	Correct (TP)

- **True Positives (TP):** Correctly predicted positives.
- **False Positives (FP):** Incorrectly predicted positives.
- **False Negatives (FN):** Incorrectly predicted negatives.
- **True Negatives (TN):** Correctly predicted negatives.

3. METHODOLOGY

3.1 Data Collection & Preprocessing

- **Dataset:** Kaggle dataset with 28,382 customers and 21 features.

- **Handling Missing Values:**
  - Categorical: Filled with mode values.
  - Numerical: Filled with median values.
- **Feature Encoding:** Applied Label Encoding and One-Hot Encoding where necessary.
- **Feature Engineering:** Created new features such as days_since_last_transaction.
- **Outlier Detection:** Identified outliers using boxplots.
- **Data Standardisation:**
  - Applied Min-Max Scaling to normalise numerical features.
  - Standardisation used for Logistic Regression.

3.2 Exploratory Data Analysis (EDA)

- **Statistical Analysis:** Summary statistics were computed to understand feature distributions.
- **Correlation Analysis:** Heatmaps were used to identify relationships among features.
- **Visualisation Techniques:**
  - Histograms and density plots to observe feature distributions.
  - Boxplots to detect outliers.
  - Count plots to examine class distributions.

3.3 Model Selection & Training

- **Models Evaluated:** Logistic Regression, Random Forest, XGBoost.
- **Data Splitting:** 80% training, 20% testing (Stratified Sampling).
- **Handling Class Imbalance:** Applied SMOTE (Synthetic Minority Oversampling Technique) to balance the dataset.
- **Feature Scaling & Dimensionality Reduction:**
  - Standardisation used for Logistic Regression.
  - PCA was applied to balance-related features to reduce multicollinearity.

    - Feature selection based on SHAP values to improve model interpretability.
- **Hyperparameter Tuning:**
    - Grid Search and Random Search applied to optimise model parameters.
    - Used cross-validation to validate performance improvements.
- **Ensemble Learning:** Combined multiple models to improve overall prediction accuracy.

3.4 Model Evaluation

- **Performance Metrics:**
    - Accuracy, Precision, Recall, F1-score, and AUC-ROC score.
- **Confusion Matrix:**
    - Visualised true positive, true negative, false positive, and false negative rates.

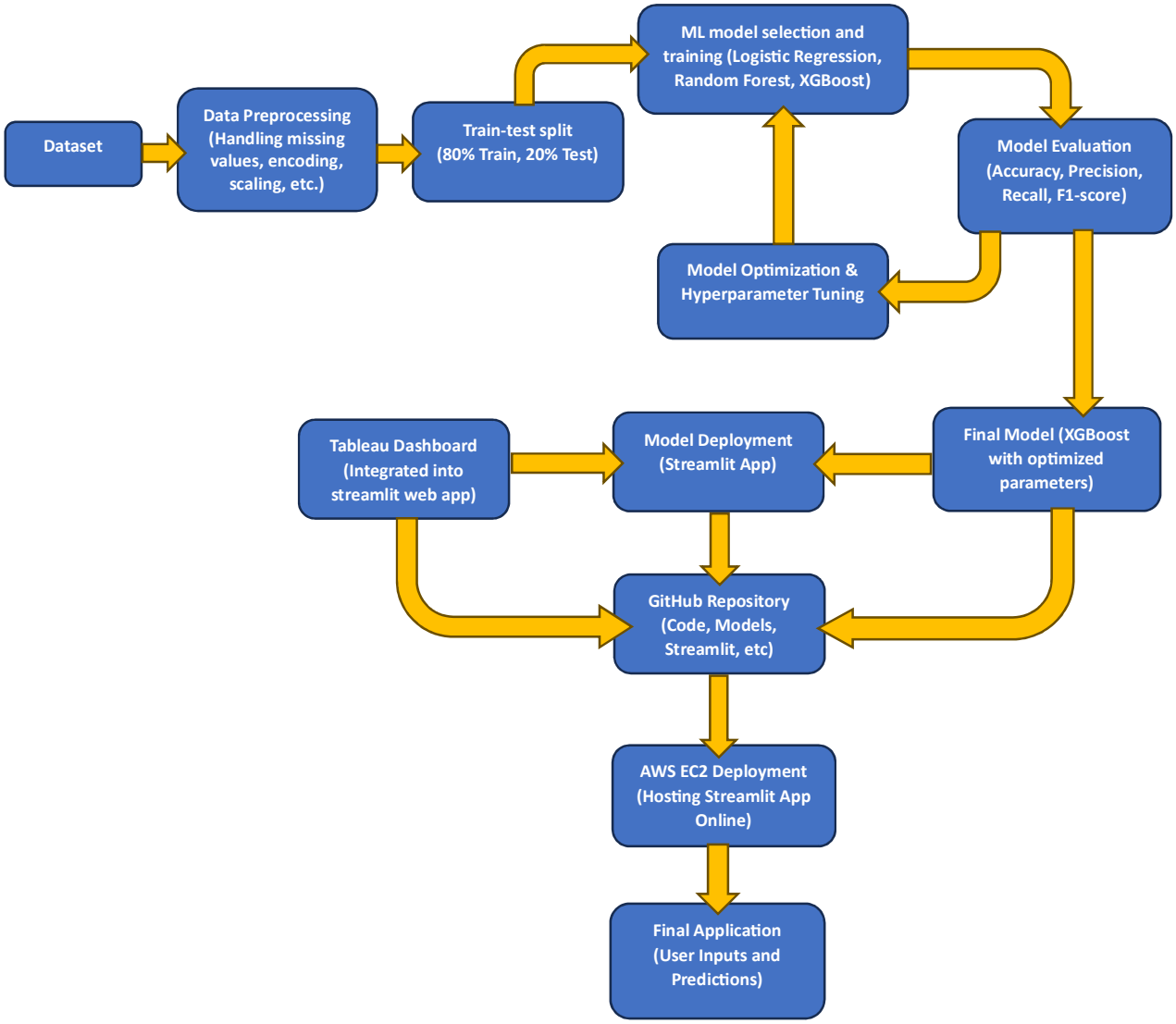
- **Model Comparison:**
    - Logistic Regression: Baseline performance.
    - Random Forest: Higher accuracy but prone to overfitting.
    - XGBoost: Best-performing model with superior generalisation.
- **Final Model Selection**: XGBoost was chosen based on performance across all evaluation metrics.

3.5 Model Deployment

- **Deployment Process**:
    - Deployed the final **XGBoost model** using **Streamlit** to create a user-friendly web interface, allowing users to input customer data and get churn predictions in real-time.

- The **model** was saved using **Pickle** and integrated into the Streamlit app for easy use.
- The application was hosted on **AWS** to provide scalability and ensure accessibility to stakeholders from anywhere.
- The deployed model provides continuous predictions, updates, and insights into customer churn, aiding decision-making for retention strategies.

3.6 Architecture:



4. IMPLEMENTATION

4.1 Importing Libraries:

At first, we have imported all the required libraries:

```python
# Importing the necessary libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import pickle
```

4.2 Data Collection & Preprocessing

Loading the dataset

```python
# Loading the dataset

df = pd.read_csv(r"C:\Users\itsme\OneDrive\Desktop\CDAC_Project\churn_prediction.csv")
df.head()
```

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_category	bı
0	1	2101	66	Male	0.0	self_employed	187.0	2	
1	2	2348	35	Male	0.0	self_employed	NaN	2	
2	4	2194	31	Male	0.0	salaried	146.0	2	
3	5	2329	90	NaN	NaN	self_employed	1020.0	2	
4	6	1579	42	Male	2.0	self_employed	1494.0	3	

5 rows × 21 columns

```python
# Number of rows and columns

df.shape
```

```
(28382, 21)
```

Handling Missing Values:

```
#Checking for null values

df.isnull().sum()
```

```
customer_id                          0
vintage                              0
age                                  0
gender                             525
dependents                        2463
occupation                          80
city                               803
customer_nw_category                 0
branch_code                          0
current_balance                      0
previous_month_end_balance           0
average_monthly_balance_prevQ        0
average_monthly_balance_prevQ2       0
current_month_credit                 0
previous_month_credit                0
current_month_debit                  0
previous_month_debit                 0
current_month_balance                0
previous_month_balance               0
churn                                0
last_transaction                     0
dtype: int64
```

```
# Filling the null values

# Used mode to fill Categorical Data
df['gender'].fillna(df['gender'].mode()[0], inplace=True)
df['occupation'].fillna(df['occupation'].mode()[0], inplace=True)
df['city'].fillna(df['city'].mode()[0], inplace=True)

# Used median to fill Numerical Data
df['dependents'].fillna(df['dependents'].median(), inplace=True)
```

```
# Check if all missing values resolved

df.isnull().sum()
```

```
customer_id                     0
vintage                         0
age                             0
gender                          0
dependents                      0
occupation                      0
city                            0
customer_nw_category            0
branch_code                     0
current_balance                 0
previous_month_end_balance      0
average_monthly_balance_prevQ   0
average_monthly_balance_prevQ2  0
current_month_credit            0
previous_month_credit           0
current_month_debit             0
previous_month_debit            0
current_month_balance           0
previous_month_balance          0
churn                           0
last_transaction                0
dtype: int64
```

Feature Engineering:

```
# Datetime format is not very suitable when creating models.
# Creating a new column 'days_since_last_transaction' i.e., difference in days from the most recent value in the data
df['days_since_last_transaction'] = (df['last_transaction'].max() - df['last_transaction']).dt.days

# Dropping the last_transaction column
df.drop(columns=['last_transaction'], inplace=True)

# Filling the missing values in ,days_since_last_transaction' with its the median values
df['days_since_last_transaction'].fillna(df['days_since_last_transaction'].median(), inplace = True)
```

Feature Encoding:

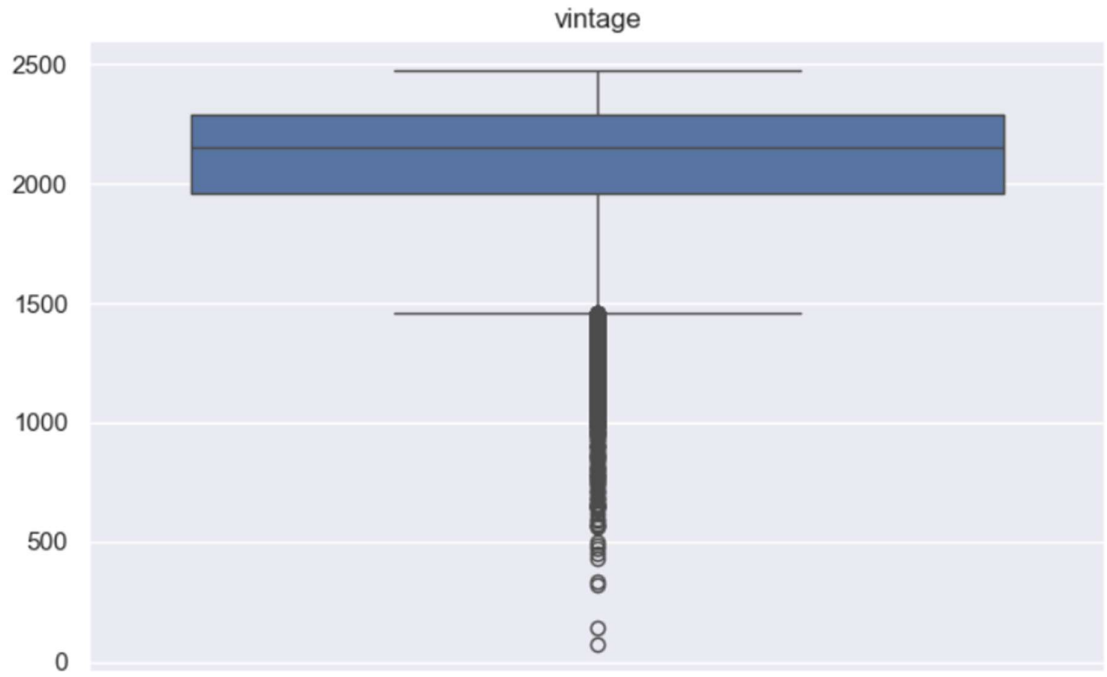
```
le = LabelEncoder()
```

```
df['gender'] = le.fit_transform(df['gender'])
df['occupation'] = le.fit_transform(df['occupation'])
df['city'] = le.fit_transform(df['city'])
df['customer_nw_category'] = le.fit_transform(df['customer_nw_category'])
```

Outlier Detection:

```
numcols = df.select_dtypes(include=['int64','float64'])
sns.set(rc = {'figure.figsize':(8,5)})
for c in numcols:
    x = df[c].values
    ax = sns.boxplot(x)
    print('The median is: ', df[c].median())
    plt.title(c)
    plt.show()
```

```
The median is:  2154.0
```




```
The median is:  46.0
```

Data Standardisation:

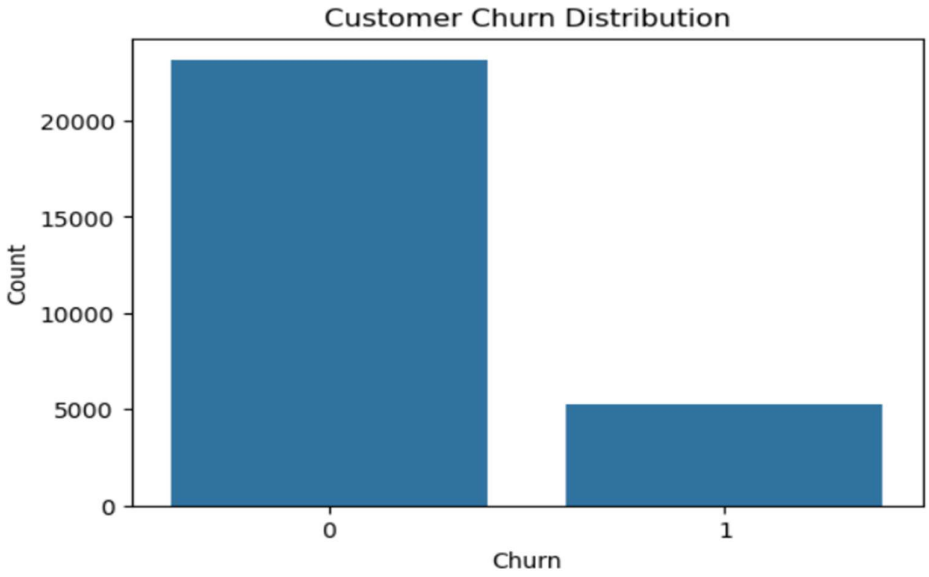
```
df['gender'] = le.fit_transform(df['gender'])
df['occupation'] = le.fit_transform(df['occupation'])
df['city'] = le.fit_transform(df['city'])
df['customer_nw_category'] = le.fit_transform(df['customer_nw_category'])
```

```
# Scaling our columns
scale_vars = ['vintage', 'age', 'dependents', 'current_balance', 'previous_month_end_balance',
       'average_monthly_balance_prevQ', 'average_monthly_balance_prevQ2',
       'current_month_credit', 'previous_month_credit', 'current_month_debit',
       'previous_month_debit', 'current_month_balance',
       'previous_month_balance', 'days_since_last_transaction']
scaler = MinMaxScaler()
df[scale_vars] = scaler.fit_transform(df[scale_vars])
df.head()
```

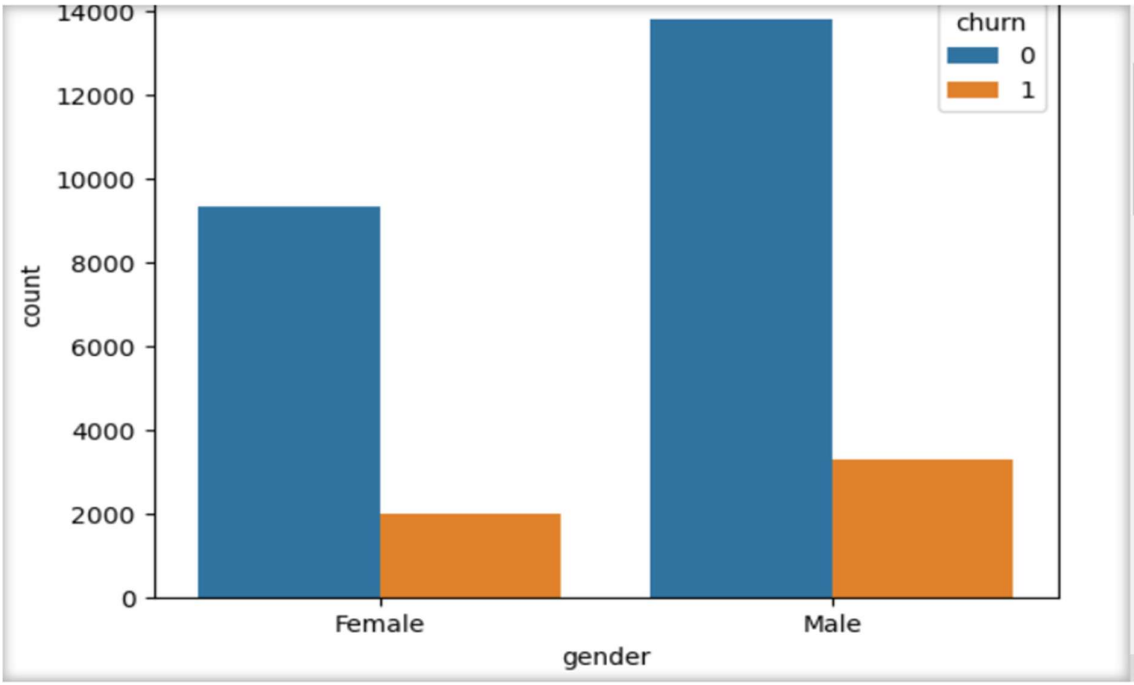
	vintage	age	gender	dependents	occupation	city	customer_nw_category	current_balance	previous_month_end_balance	average_month
0	0.843945	0.730337	1	0.000000	3	182	1	0.001178	0.000802	
1	0.946733	0.382022	1	0.000000	3	992	1	0.001843	0.002064	
2	0.882647	0.337079	1	0.000000	2	142	1	0.001593	0.001561	
3	0.938826	1.000000	1	0.000000	3	992	1	0.001319	0.000947	
4	0.626717	0.460674	1	0.038462	3	1452	2	0.001088	0.000792	

4.3 Exploratory Data Analysis (EDA)

```
plt.figure(figsize=(6, 4))
sns.countplot(x='churn', data=df)
plt.title("Customer Churn Distribution")
plt.xlabel("Churn")
plt.ylabel("Count")
plt.show()
```

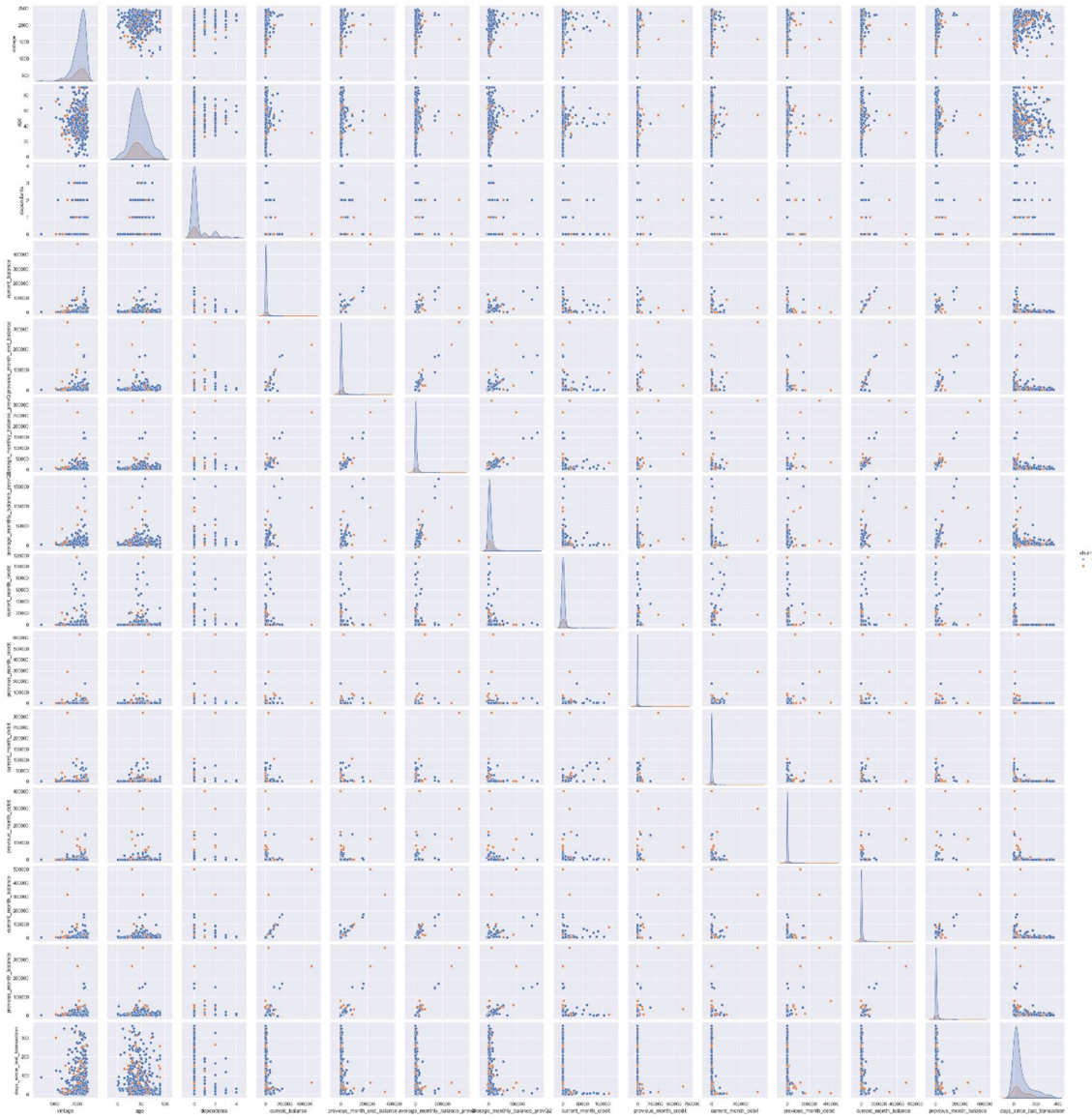



```
for f in ['gender','occupation','customer_nw_category']:
    plt.figure()
    ax = sns.countplot(x=f, data = df, hue = 'churn')
```



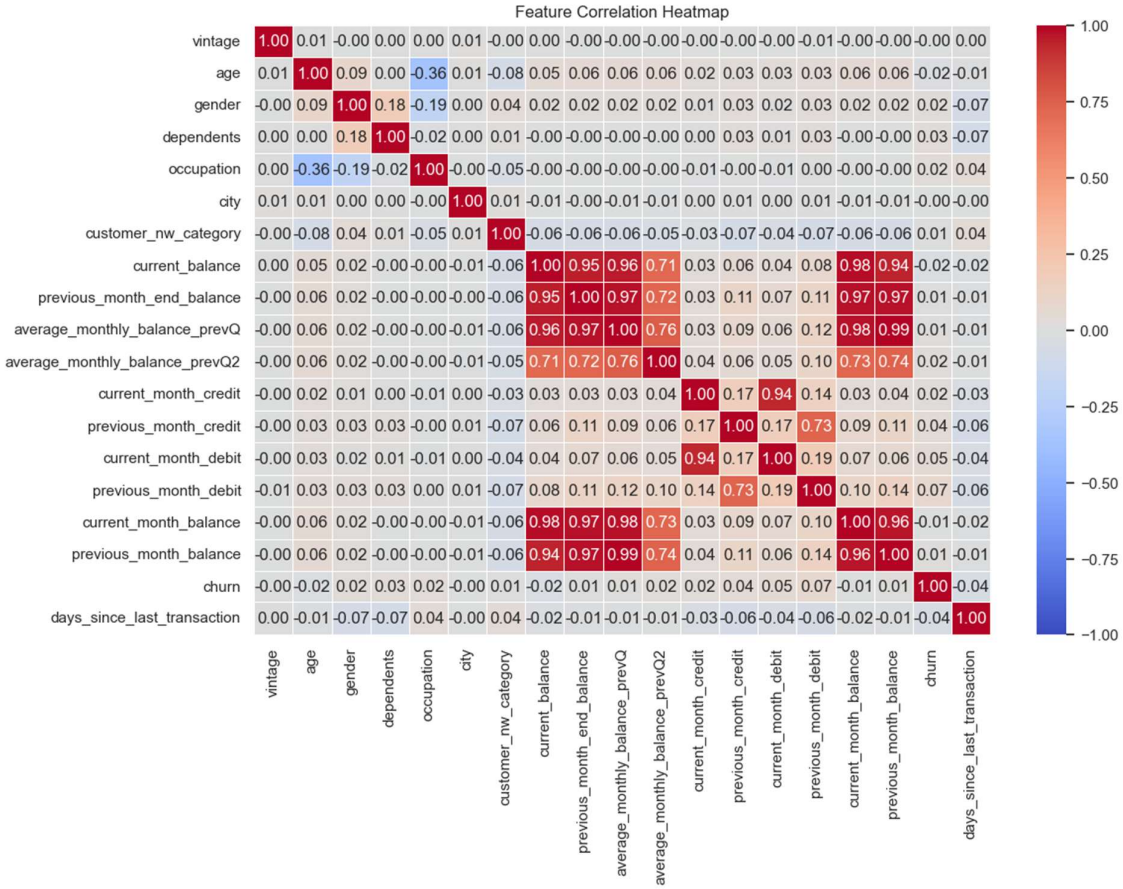
Statistical Analysis: Summary statistics were computed to understand feature distributions.

```
sample_df = df.sample(500, random_state=42)
sns.pairplot(sample_df, hue='churn', diag_kind='kde')
plt.show()
```



Correlation Analysis: Heatmaps were used to identify relationships among features

```
plt.figure(figsize=(12, 8))
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm', linewidths=0.5, vmin=-1, vmax = 1)
plt.title("Feature Correlation Heatmap")
plt.show()
```



4.4 Model Training and Evaluation:

Train-test Split:

```
x = df.drop('churn', axis = 1)
y =  df['churn']
```

```
x_train, x_test, y_train, y_test =  train_test_split(x,y, test_size = 0.2, random_state = 42, stratify = y)
```

Logistic Regression:

```
log_reg = LogisticRegression(max_iter=1000, random_state=42)
scaler = StandardScaler()
x_train_scaled = pd.DataFrame(scaler.fit_transform(x_train), columns=x.columns)
x_test_scaled = pd.DataFrame(scaler.transform(x_test), columns=x.columns)
highly_correlated = [
    'current_balance', 'previous_month_end_balance',
    'average_monthly_balance_prevQ',
    'current_month_balance', 'previous_month_balance'
]
pca_balance = PCA(n_components=1)
x_train_scaled['balance_pca'] = pca_balance.fit_transform(x_train_scaled[highly_correlated])
x_test_scaled['balance_pca'] = pca_balance.transform(x_test_scaled[highly_correlated])
```

```
# Drop original correlated features
x_train_scaled.drop(columns=highly_correlated, inplace=True)
x_test_scaled.drop(columns=highly_correlated, inplace=True)
```

```
log_reg.fit(x_train_scaled, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000, random_state=42)
```

```
# Predictions
y_pred_log =log_reg.predict(x_test_scaled)
y_pred_log_prob = log_reg.predict_proba(x_test_scaled)[:, 1]
```

```
print(f"Accuracy: {accuracy_score(y_test, y_pred_log):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_log))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_log))
```

```
Accuracy: 0.8149

Classification Report:
              precision    recall  f1-score   support

           0       0.82      1.00      0.90      4625
           1       0.51      0.02      0.03      1052

    accuracy                           0.81      5677
   macro avg       0.67      0.51      0.47      5677
weighted avg       0.76      0.81      0.74      5677


Confusion Matrix:
[[4608   17]
 [1034   18]]
```

Here we can see that logistic regression is not handling the data properly. Very low recall

Random Forest:

```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
rf_model.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# Predictions
y_pred_rf = rf_model.predict(x_test)
y_pred_rf_prob = rf_model.predict_proba(x_test)[:, 1]
```

```
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf):.4f}")
print("Classification Report:")
print(classification_report(y_test, y_pred_rf))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))
```

```
Accuracy: 0.8640
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      4625
           1       0.73      0.42      0.54      1052

    accuracy                           0.86      5677
   macro avg       0.80      0.69      0.73      5677
weighted avg       0.85      0.86      0.85      5677

Confusion Matrix:
[[4459  166]
 [ 606  446]]
```

Here we can see the recall very low. Which means it classifies the churning customers into retained category.

Random Forest After applying SMOTE:

```
# Applying SMOTE to oversample the minority class in the training set
smote = SMOTE(random_state=42)
x_train_resampled, y_train_resampled = smote.fit_resample(x_train, y_train)
```

[44]:

```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
```

[45]:

```
rf_model.fit(x_train_resampled, y_train_resampled)
```

[45]:

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

[46]:

```
y_train_pred = rf_model.predict(x_train_resampled)
y_test_pred = rf_model.predict(x_test)
```

```
print(f"Accuracy: {accuracy_score(y_test, y_test_pred):.4f}")
print("Classification Report:")
print(classification_report(y_test, y_test_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_test_pred))
```

```
Accuracy: 0.8286
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.87      0.89      4625
           1       0.53      0.63      0.57      1052

    accuracy                           0.83      5677
   macro avg       0.72      0.75      0.73      5677
weighted avg       0.84      0.83      0.83      5677

Confusion Matrix:
[[4046  579]
 [ 394  658]]
```

Still low precision and recall for the churned customers

XGBoost:

```
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
```

```
xgb_model.fit(x_train, y_train)
```

```
C:\Users\itsme\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning: [15:27:13] WARNIN
G: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-08cbc0333d8d4aae1-1\xgbo
ost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, random_state=42, ...)
```

```
y_train_pred_xgb = xgb_model.predict(x_train)
y_test_pred_xgb = xgb_model.predict(x_test)
```

```
print(f"Accuracy: {accuracy_score(y_test, y_test_pred_xgb):.4f}")
print("Classification Report:")
print(classification_report(y_test, y_test_pred_xgb))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_test_pred_xgb))
```

```
Accuracy: 0.8614
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      4625
           1       0.69      0.46      0.55      1052

    accuracy                           0.86      5677
   macro avg       0.79      0.71      0.74      5677
weighted avg       0.85      0.86      0.85      5677

Confusion Matrix:
[[4403  222]
 [ 565  487]]
```

Here we can see that precision and recall values are better than the Logistic regression and random forest. But further actions needed.

Using class weight to balance the dataset:

```
# Using Class weightage
neg, pos = y_train.value_counts()
scale_pos_weight = neg / pos  # Adjusts for class imbalance
```

```
xgb_model = XGBClassifier(scale_pos_weight=scale_pos_weight, eval_metric='logloss', random_stat
xgb_model.fit(x_train, y_train)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, random_state=42, ...)
```

```
y_train_pred = xgb_model.predict(x_train)
y_test_pred = xgb_model.predict(x_test)
```

```
print(f"Accuracy: {accuracy_score(y_test, y_test_pred):.4f}")
print("Classification Report:")
print(classification_report(y_test, y_test_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_test_pred))
```

```
Accuracy: 0.8416
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.89      0.90      4625
           1       0.57      0.62      0.59      1052

    accuracy                           0.84      5677
   macro avg       0.74      0.76      0.75      5677
weighted avg       0.85      0.84      0.84      5677

Confusion Matrix:
[[4124  501]
 [ 398  654]]
```

Precision and recall improved. Further actions needed

Applying hyperparametric tuning:

```
best_xgb = XGBClassifier(
    scale_pos_weight=scale_pos_weight,
    max_depth=4,  # Reduce overfitting
    n_estimators=500,  # More trees improve generalization
    learning_rate=0.05,  # Slower learning for better accuracy
    colsample_bytree=0.8,  # Feature selection
    subsample=0.8,  # Prevents overfitting
    eval_metric='logloss',
    random_state=42
)
best_xgb.set_params(early_stopping_rounds=10)

best_xgb.fit(
    x_train, y_train,
    eval_set=[(x_test, y_test)],
    verbose=False
)
```

[70]:

▸ XGBClassifier

[71]:

```
y_train_pred = best_xgb.predict(x_train)
y_test_pred = best_xgb.predict(x_test)
```

```
print(f"Accuracy: {accuracy_score(y_test, y_test_pred):.4f}")
print("Classification Report:")
print(classification_report(y_test, y_test_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_test_pred))
```

```
Accuracy: 0.8348
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.86      0.90      4625
           1       0.54      0.70      0.61      1052

    accuracy                           0.83      5677
   macro avg       0.73      0.78      0.75      5677
weighted avg       0.86      0.83      0.84      5677

Confusion Matrix:
[[3999  626]
 [ 312  740]]
```

Lowering the threshold to 0.4:

```
# Predictions
y_pred_xgb = best_xgb.predict(x_test)
y_pred_xgb_prob = best_xgb.predict_proba(x_test)[:, 1]
```

[76]:

```
# Adjust decision threshold
threshold = 0.4  # Lower threshold improves recall
y_pred_adjusted = np.where(y_pred_xgb_prob > threshold, 1, 0)
```

[77]:

```
# Evaluate the model
print(f"Accuracy: {accuracy_score(y_test, y_pred_adjusted):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_adjusted))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_adjusted))
```

```
Accuracy: 0.7939

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.80      0.86      4625
           1       0.47      0.77      0.58      1052

    accuracy                           0.79      5677
   macro avg       0.70      0.78      0.72      5677
weighted avg       0.85      0.79      0.81      5677


Confusion Matrix:
[[3701  924]
 [ 246  806]]
```

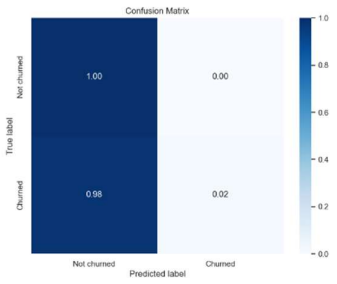
Here we can see that the model can classify both the churned and retained customers to some extent. So we will finalize this as our model.

Based on raw numbers, the last confusion matrix ([[3701 924] [246 806]]) has the highest TP (806) and the lowest FN (246), meaning it captures the most churn cases correctly.

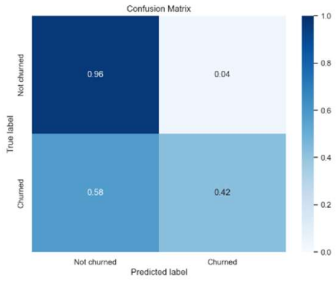
5. RESULTS AND ANALYSIS

5.1 Comparison of Classification Models

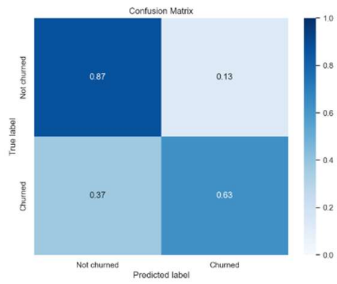
Logistic Regression:



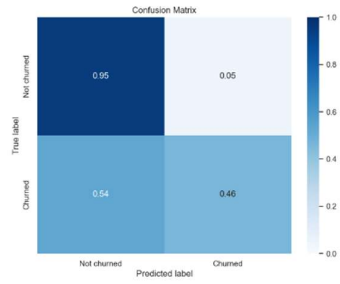
Random Forest:



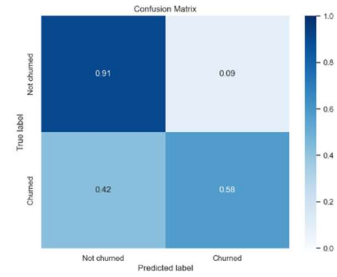
Random forest + SMOTE:



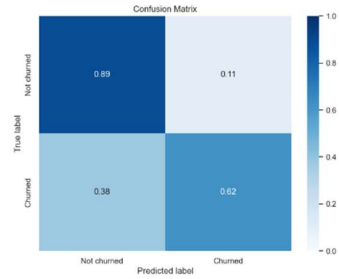
XGBoost:



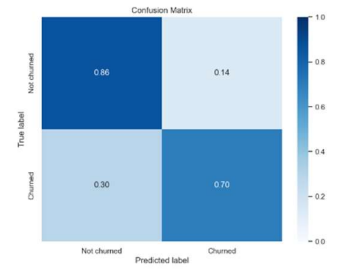
XGBoost + SMOTE:



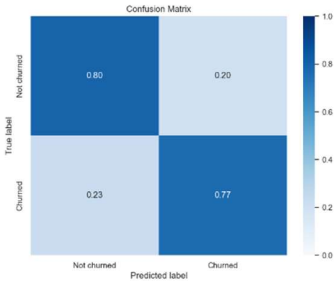
XGBoost + weight Balancing:



XGBoost + weight Balancing + hyperparameter tuning



XGBoost + weight Balancing + hyperparameter tuning + threshold reduction



5.2 Performance Metrics Comparison

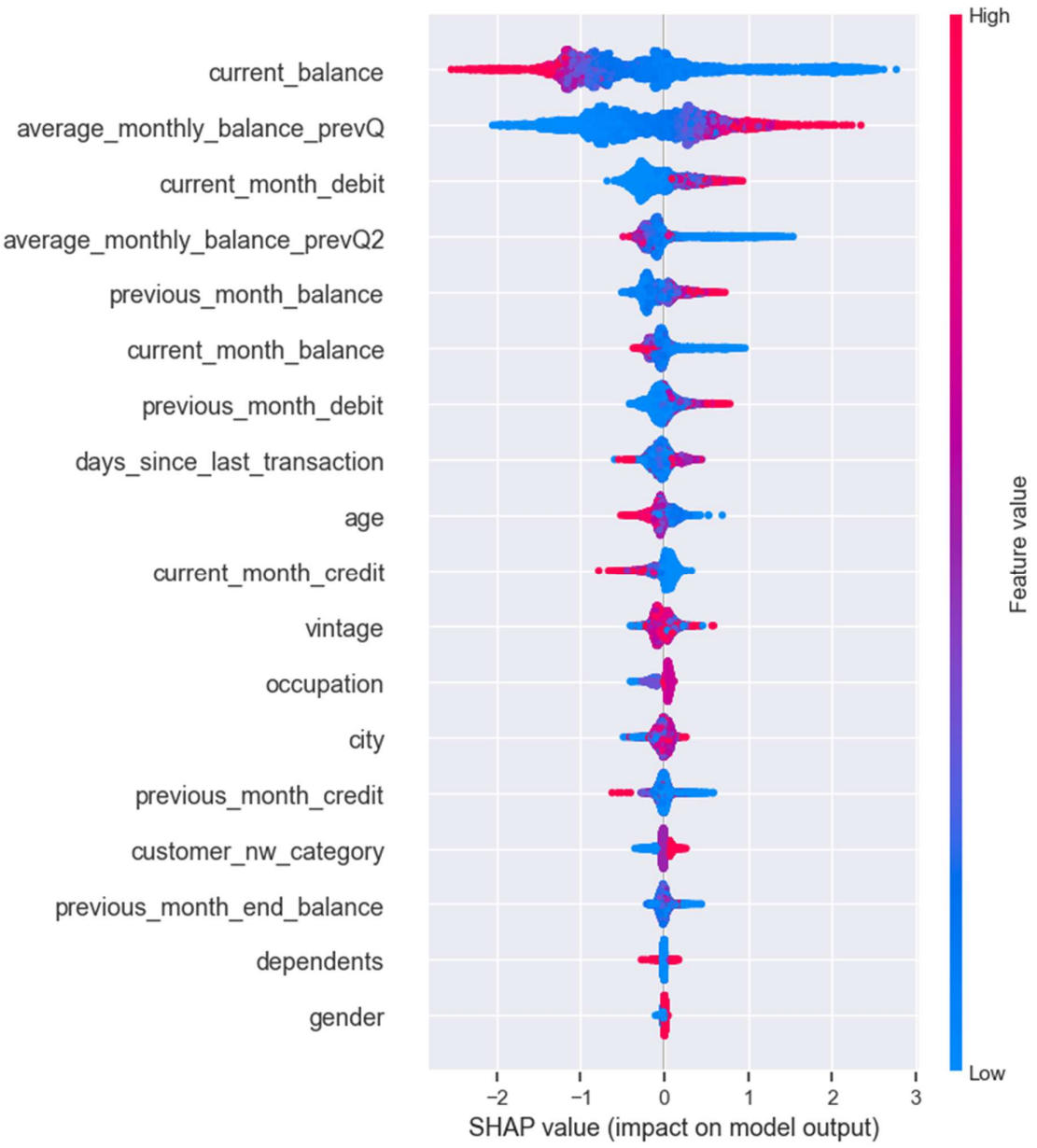
Matrix	TN	FP	FN	TP	Accuracy	Precision	Recall	F1-Score
Logistic Regression	4608	17	1034	18	**0.8149**	0.5143	0.0171	0.0331
Random forest	4459	166	606	446	**0.8640**	**0.7288**	0.4240	0.5361
Random Forest + SMOTE	4046	579	394	658	**0.8286**	0.5319	0.6255	0.5749
XGBoost	4403	222	565	487	**0.8614**	0.6869	0.4629	0.5531
XGBoost + SMOTE	4193	432	445	607	**0.8455**	0.5842	0.5770	0.5806
XGBoost + Weight Balancing	4124	501	398	654	**0.8416**	0.5662	0.6217	0.5927
XGBoost + Weight Balancing + hyperparameter tuning	3999	626	312	740	**0.8348**	0.5417	**0.7034**	**0.6121**
XGBoost + Weight Balancing + hyperparameter tuning + threshold reduction	3701	924	246	806	**0.7939**	0.4659	**0.7662**	0.5794

Classification report: In customer churn prediction, recall is prioritized over other metrics because identifying churned customers correctly is more important than minimizing false positives.

Hence we are choosing the last model as our best model for deployment as its having the highest recall.

5.3 Feature Importance using SHAP:

```
import shap

explainer = shap.Explainer(best_xgb)
shap_values = explainer(x_test)

# Summary plot to see overall impact
shap.summary_plot(shap_values, x_test)
```



Above SHAP analysis we can see the features which are contributing more to the churn rate of customers. We can see current balance is affecting the churn of the customers most followed by average month balance followed by current month debit and so on.

6. DEPLOYMENT:

6.1 Storing the model in pickle format.

```
# Adding predictions back to the dataset
df['Exited Prediction'] = best_xgb.predict(x)
df['Exited Prediction Probability'] = best_xgb.predict_proba(x)[:, 1]
```

```
# Exporting predictions
df.to_excel("xgboost_predictions.xlsx", index=False)
```

```
# Saving the XGBoost model
with open('best_xgboost_model.pkl', 'wb') as file:
    pickle.dump(best_xgb, file)
```

```
with open('scaler.pkl', 'wb') as file:
    pickle.dump(scaler, file)
```

6.2 Streamlit app Deployment:

Load Model and Scaler

```
import streamlit as st
import pickle
import pandas as pd
import plotly.express as px
import streamlit.components.v1 as components

st.set_page_config(layout="wide")

# Load trained XGBoost model
try:
    with open('best_xgboost_model.pkl', 'rb') as file:
        model = pickle.load(file)
except FileNotFoundError:
    st.error("`best_xgboost_model.pkl` not found! Please ensure the model file is in the correct
directory.")
    st.stop()

# Load MinMaxScaler
try:
    with open('scaler.pkl', 'rb') as file:
        scaler = pickle.load(file)
except FileNotFoundError:
    st.error("`scaler.pkl` not found! Please ensure the scaler file is available.")
    st.stop()
```

Define Features

```
feature_names = [
    'vintage', 'age', 'gender', 'dependents', 'occupation', 'city',
    'customer_nw_category', 'current_balance', 'previous_month_end_balance',
    'average_monthly_balance_prevQ', 'average_monthly_balance_prevQ2',
    'current_month_credit', 'previous_month_credit', 'current_month_debit',
    'previous_month_debit', 'current_month_balance',
    'previous_month_balance', 'days_since_last_transaction'
]

# Columns requiring scaling
scale_vars = [
    'vintage', 'age', 'dependents', 'current_balance', 'previous_month_end_balance',
    'average_monthly_balance_prevQ', 'average_monthly_balance_prevQ2',
    'current_month_credit', 'previous_month_credit', 'current_month_debit',
    'previous_month_debit', 'current_month_balance',
    'previous_month_balance', 'days_since_last_transaction'
```

Sidebar User Inputs (With Defaults)

```
st.sidebar.image("Pic 1.png", use_container_width=True)  # Display Pic 1
st.sidebar.header("User Inputs")

# Default values for user input (ensures valid predictions)
default_values = {
    "vintage": 60, "age": 35, "gender": "Male", "dependents": 2, "occupation":
"salaried",
    "city": "1020", "customer_nw_category": "1", "current_balance": 50000,
    "previous_month_end_balance": 45000, "average_monthly_balance_prevQ": 42000,
    "average_monthly_balance_prevQ2": 40000, "current_month_credit": 15000,
    "previous_month_credit": 13000, "current_month_debit": 12000,
    "previous_month_debit": 11000, "current_month_balance": 40000,
    "previous_month_balance": 43000, "days_since_last_transaction": 30
}

# Collect user inputs
user_inputs = {}
for feature in feature_names:
    if feature in scale_vars:
        user_inputs[feature] = st.sidebar.number_input(feature,
value=default_values[feature], step=1)
    else:
        if feature == "gender":
            user_inputs[feature] = st.sidebar.selectbox("Gender", options=
["Male", "Female"], index=0)
        elif feature == "occupation":
            user_inputs[feature] = st.sidebar.selectbox("Occupation", options=
["salaried", "self-employed", "unemployed"], index=0)
        elif feature == "customer_nw_category":
            user_inputs[feature] = st.sidebar.selectbox("Customer NW Category",
options=["1", "2", "3"], index=0)
        elif feature == "city":
            user_inputs[feature] = st.sidebar.selectbox("City", options=
["1020", "1030"], index=0)
        else:
            user_inputs[feature] = st.sidebar.text_input(feature,
value=default_values[feature])

# Convert to DataFrame
input_data = pd.DataFrame([user_inputs])

# Ensure categorical variables are properly encoded (manual encoding for
simplicity)
input_data.replace({
    "gender": {"Male": 1, "Female": 0},
    "occupation": {"salaried": 1, "self-employed": 2, "unemployed": 3},
    "customer_nw_category": {"1": 1, "2": 2, "3": 3},
    "city": {"1020": 1020, "1030": 1030}
}, inplace=True)
```

Apply MinMaxScaler:

```
try:
    input_data_scaled = input_data.copy()
    input_data_scaled[scale_vars] = scaler.transform(input_data[scale_vars])
except ValueError as ve:
    st.error(f"Invalid input! Please check the values of your fields. Details:
{ve}")
    st.stop()
```

Prediction:

```
st.image("Pic 2.png", use_container_width=True)  # Display Pic 2
st.title("Customer Churn Prediction")

# Page Layout
left_col, right_col = st.columns(2)

with left_col:
    st.header("Feature Importance")
    # Load feature importance data from the Excel file
    feature_importance_df = pd.read_excel("feature_importance.xlsx", usecols=
["Feature", "Feature Importance Score"])
    # Plot the feature importance bar chart
    fig = px.bar(
        feature_importance_df.sort_values(by="Feature Importance Score",
ascending=True),
        x="Feature Importance Score",
        y="Feature",
        orientation="h",
        title="Feature Importance",
        labels={"Feature Importance Score": "Importance", "Feature":
"Features"},
        width=400,  # Set custom width
        height=500  # Set custom height
    )
    st.plotly_chart(fig)
```

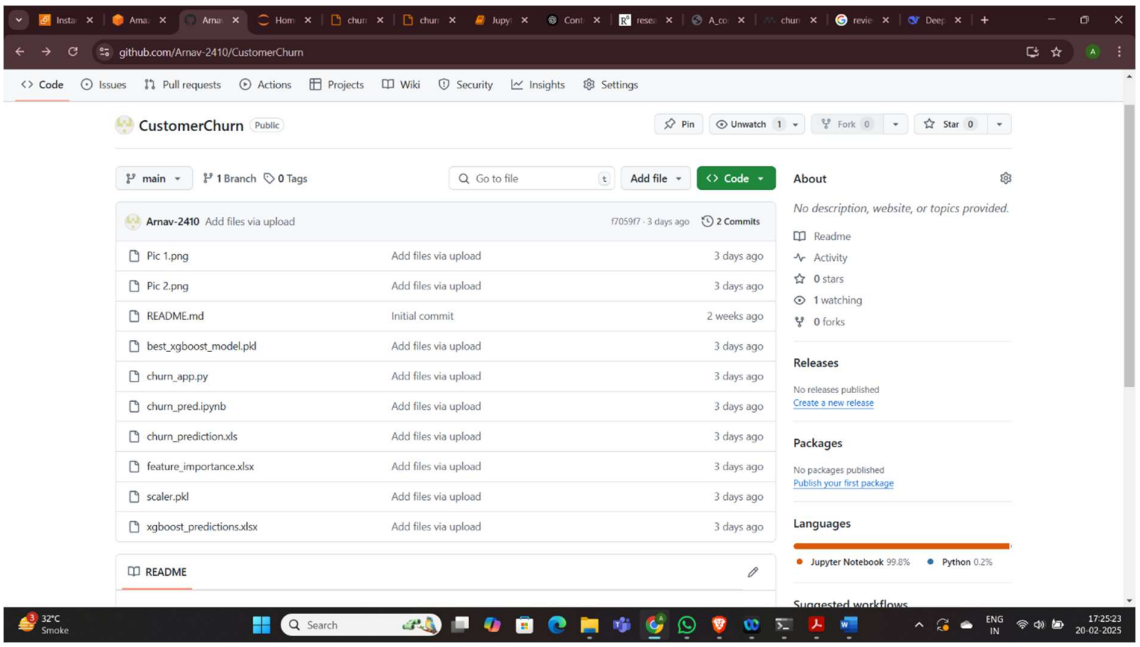
```
# Right Page: Prediction
with right_col:
    st.header("Prediction")
    if st.button("Predict"):
        try:
            # Get predicted probabilities and label
            probabilities = model.predict_proba(input_data_scaled)[0]
            prediction = model.predict(input_data_scaled)[0]

            # Map prediction to label
            prediction_label = "Churned" if prediction == 1 else "Retained"

            # Display results
            st.subheader(f"Predicted Value: **{prediction_label}**")
            st.write(f"**Churn Probability:** {probabilities[1]:.2%}")
            st.write(f"**Retention Probability:** {probabilities[0]:.2%}")

            # Display result with color indication
            if prediction == 1:
                st.error("High risk of churn! Take action to retain this
customer.")
            else:
                st.success("Customer is likely to stay!")

        except Exception as e:
            st.error(f" Error in prediction: {e}")
```

6.3 Tableau Dashboard Integration:

```
embed_code = '''
<div class='tableauPlaceholder' id='viz1739763468407' style='position:
relative'><noscript><a href='#'><img alt='Dashboard 1 '
src='https:&#47;&#47;public.tableau.com&#47;static&#47;images&#47;Ch&#47;Churn_D
ashboard_17397175233280&#47;Dashboard1&#47;1_rss.png' style='border: none' />
</a></noscript><object class='tableauViz'  style='display:none;'><param
name='host_url' value='https%3A%2F%2Fpublic.tableau.com%2F' /> <param
name='embed_code_version' value='3' /> <param name='site_root' value='' />
<param name='name' value='Churn_Dashboard_17397175233280&#47;Dashboard1' />
<param name='tabs' value='no' /><param name='toolbar' value='yes' /><param
name='static_image'
value='https:&#47;&#47;public.tableau.com&#47;static&#47;images&#47;Ch&#47;Churn
_Dashboard_17397175233280&#47;Dashboard1&#47;1.png' /> <param
name='animate_transition' value='yes' /><param name='display_static_image'
value='yes' /><param name='display_spinner' value='yes' /><param
name='display_overlay' value='yes' /><param name='display_count' value='yes' />
<param name='language' value='en-US' /></object></div>                <script
type='text/javascript'>                    var divElement =
document.getElementById('viz1739763468407');                    var vizElement
= divElement.getElementsByTagName('object')[0];                    if (
divElement.offsetWidth > 800 ) {
vizElement.style.width='1100px';vizElement.style.height='607px';} else if (
divElement.offsetWidth > 500 ) {
vizElement.style.width='1100px';vizElement.style.height='607px';} else {
vizElement.style.width='100%';vizElement.style.height='1627px';}
     var scriptElement = document.createElement('script');
scriptElement.src = 'https://public.tableau.com/javascripts/api/viz_v1.js';
              vizElement.parentNode.insertBefore(scriptElement, vizElement);
            </script>
'''
st.header("Customer Insights Dashboard")
components.html(embed_code, height=800)
```

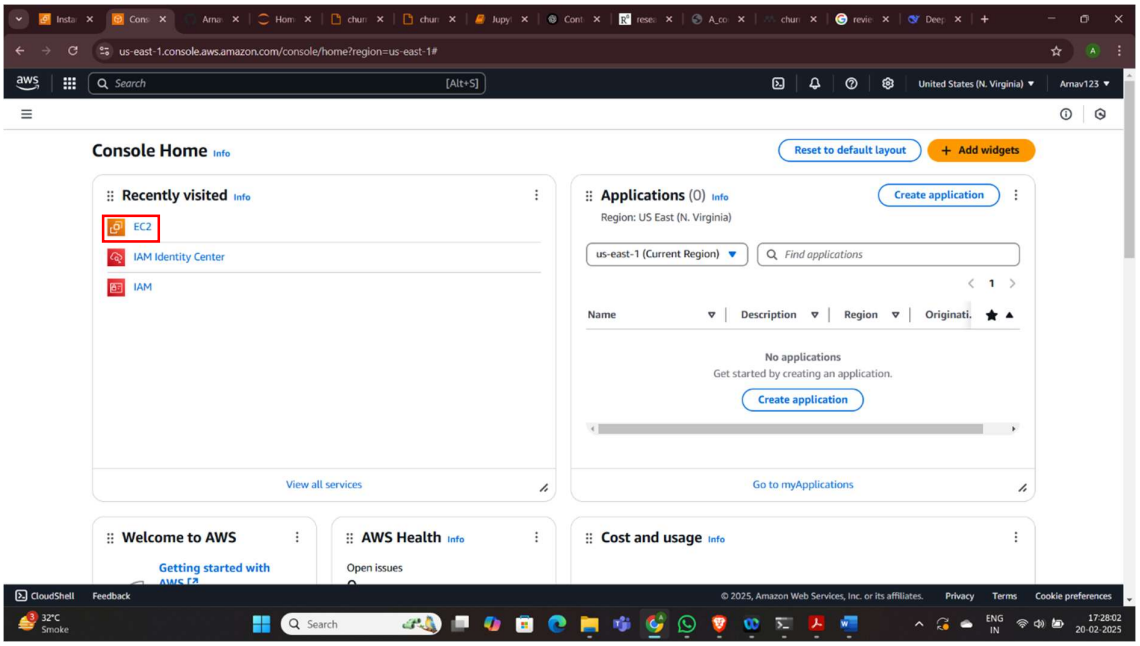
6.4 GitHub Repository Management

Upload the Streamlit app and other required files in github repository.

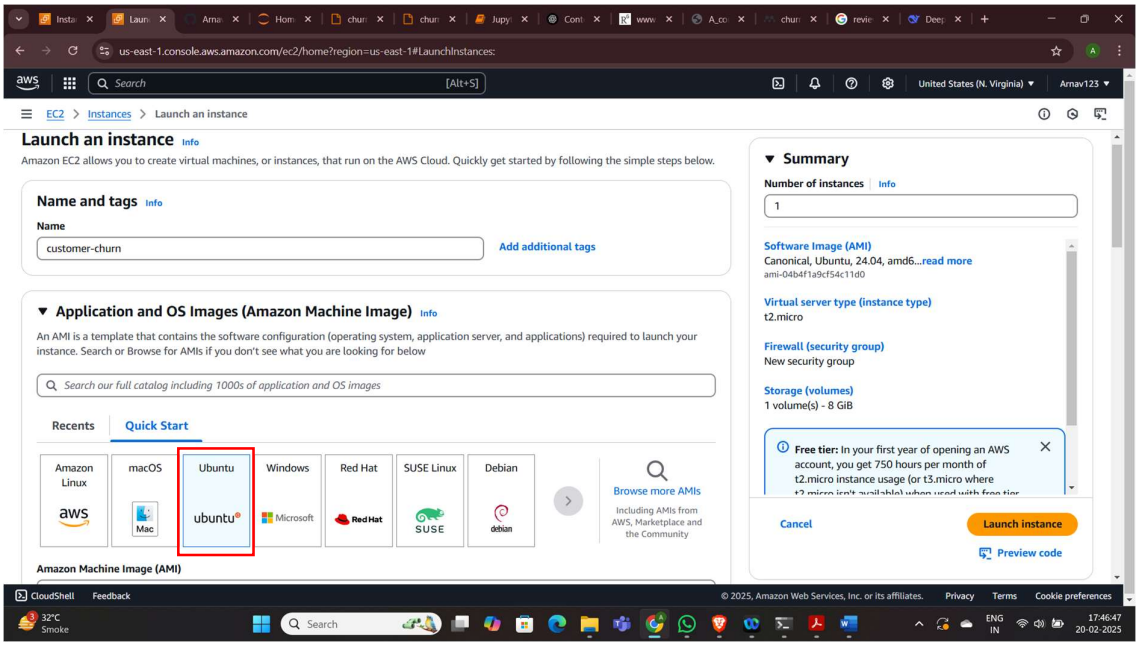


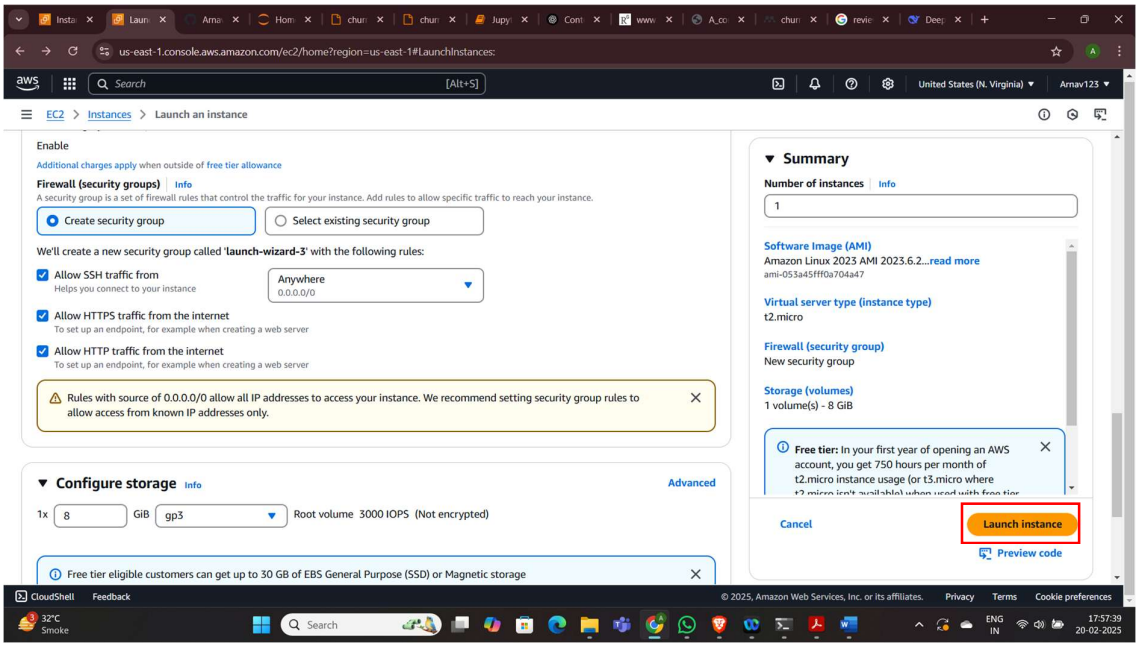
6.5 AWS EC2 Hosting

Login to Amazon AWS console and create an EC2 Instance:

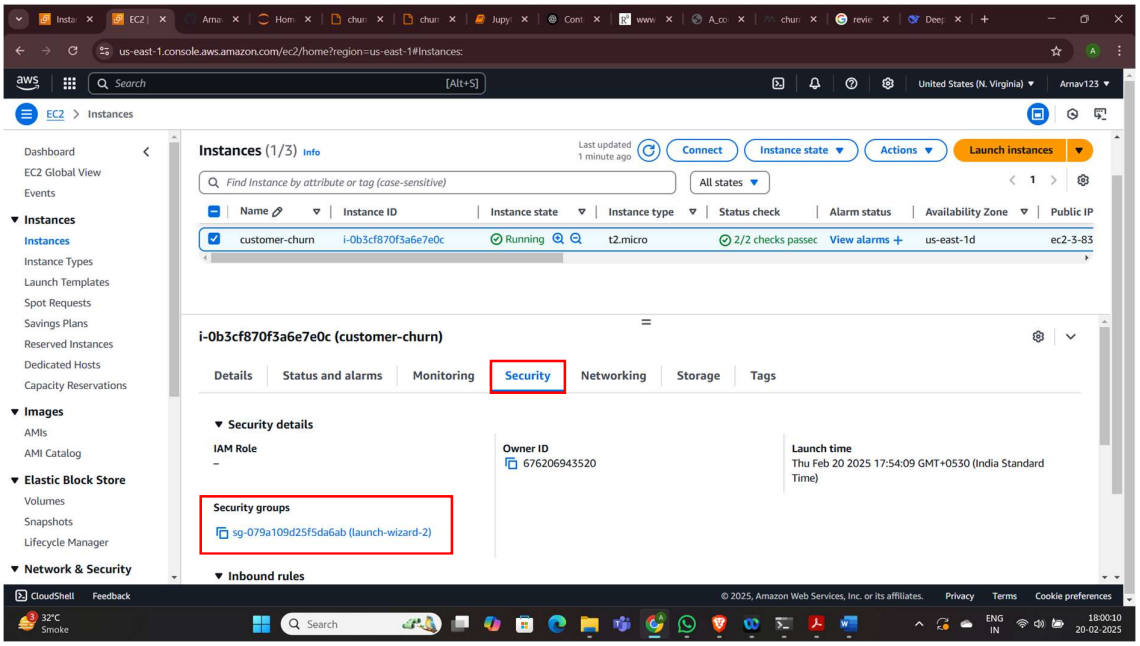


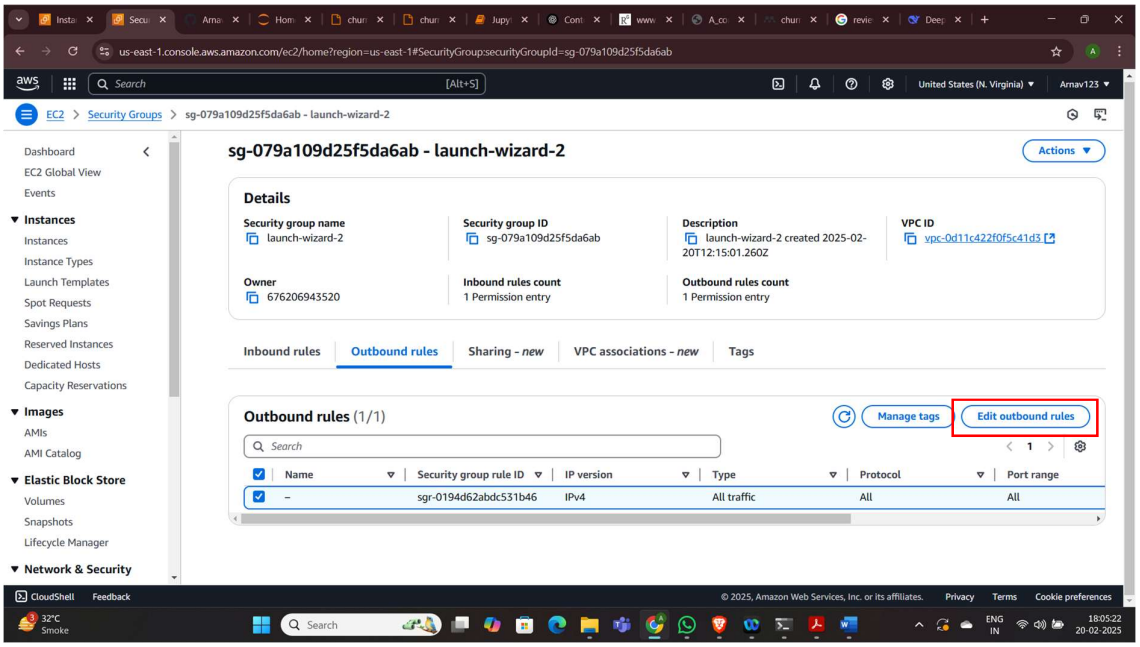
Created an EC2 instance with Ubuntu as the operating system and click on Launch Instance.

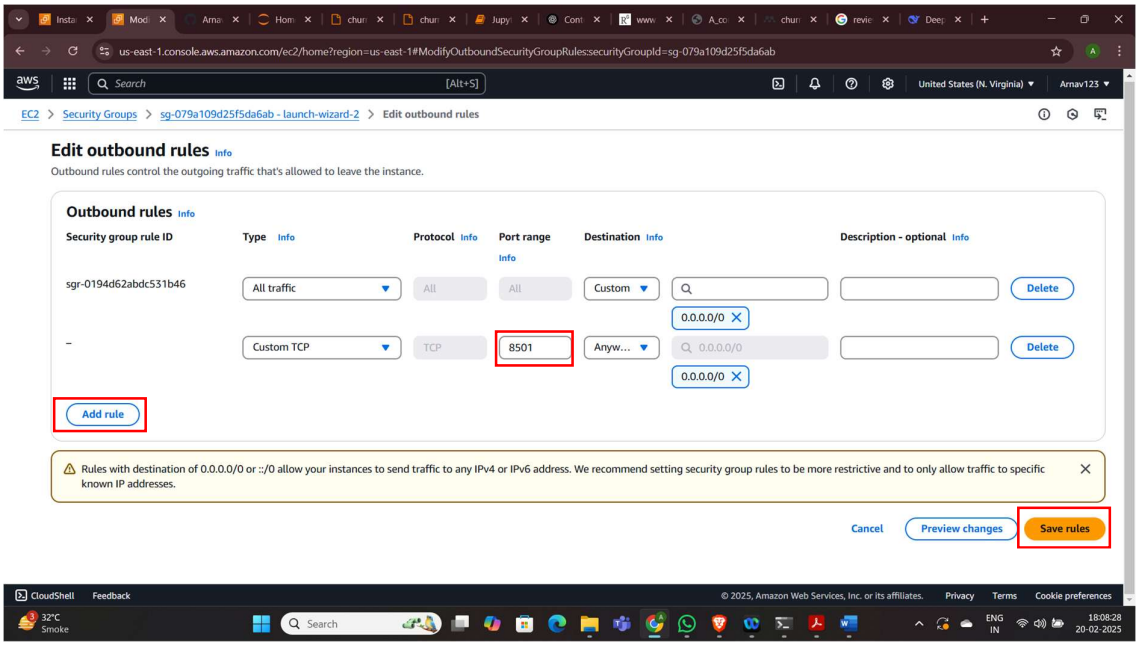




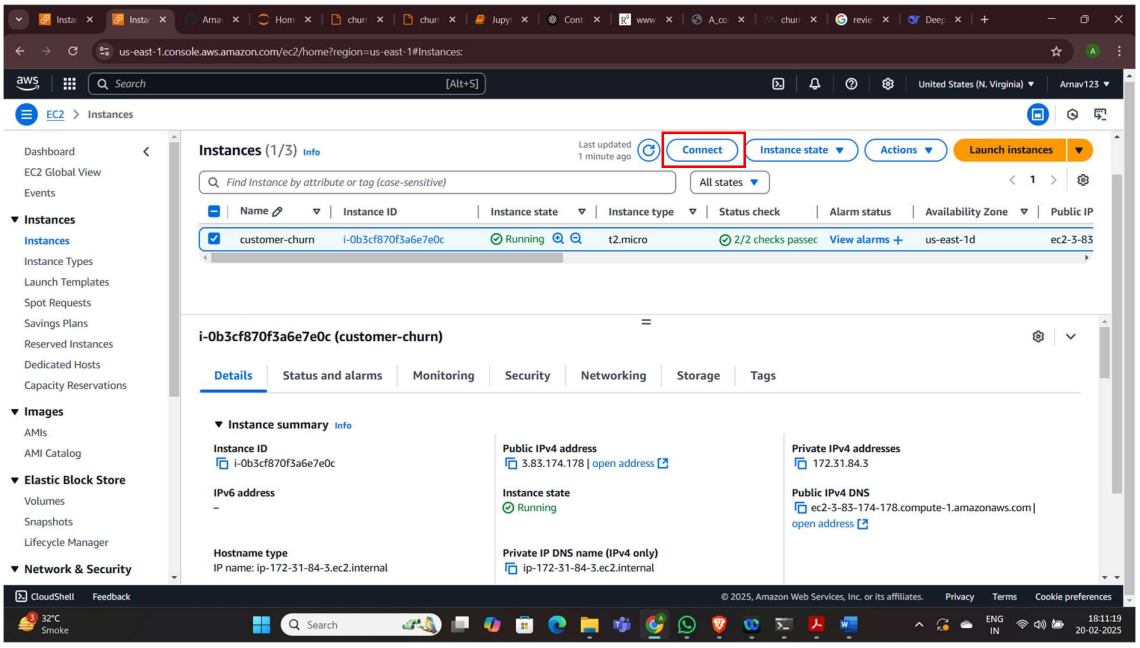
Port-mapping (Streamlit runs on port 8501):

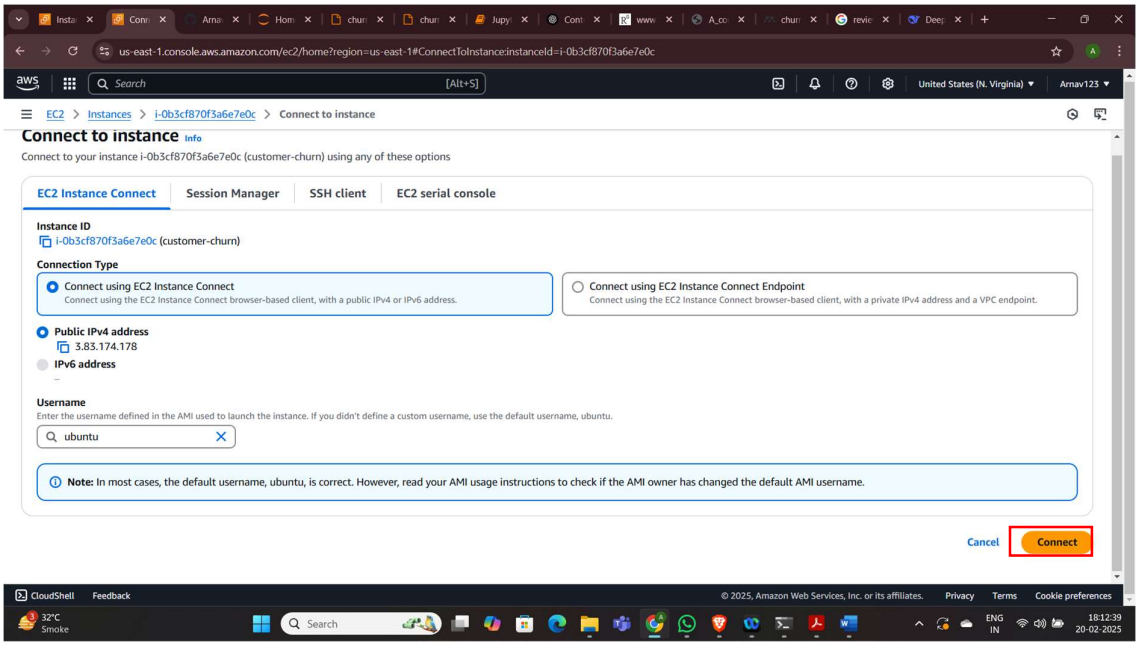


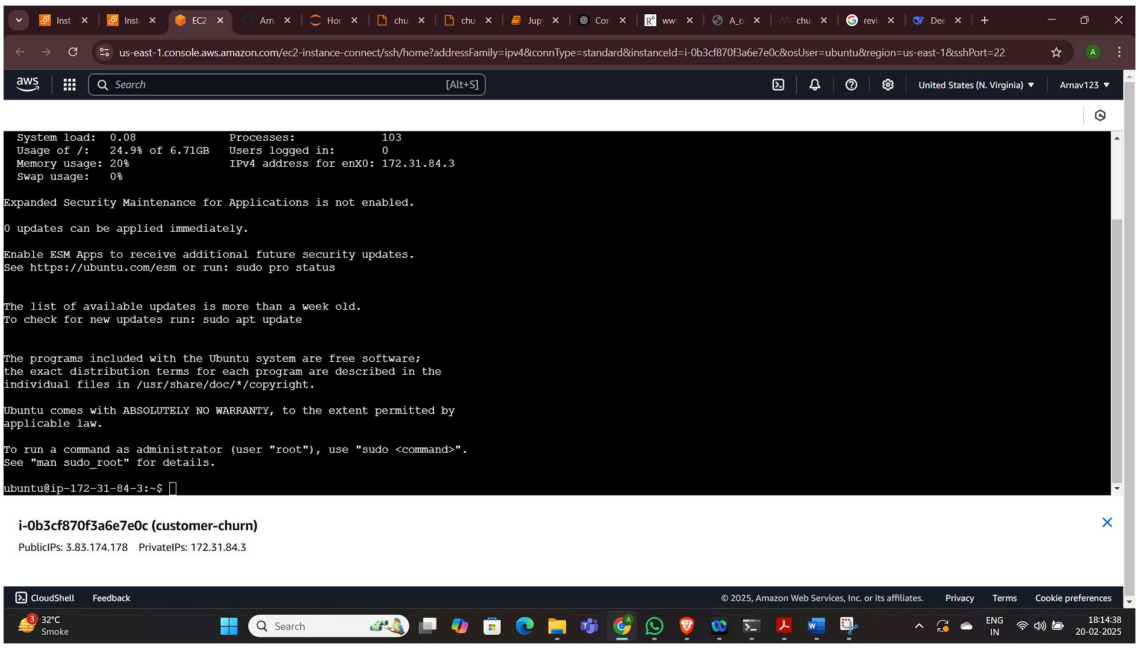




Connecting to the created EC2 Instance:

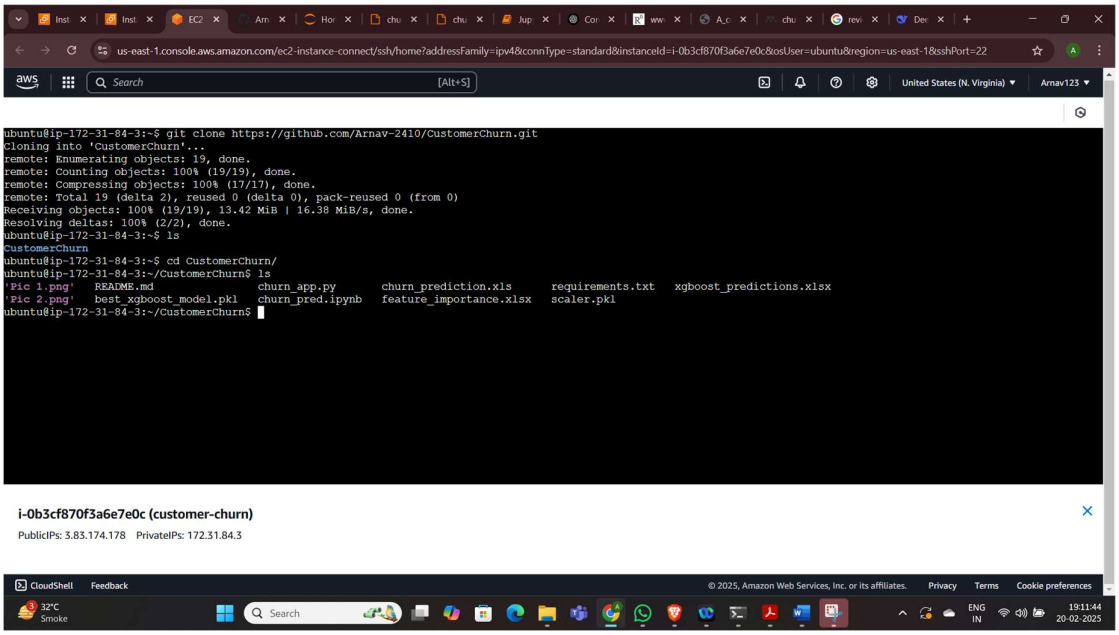






Ubuntu terminal successfully deployed.

Installed git and cloned my repository in the terminal.



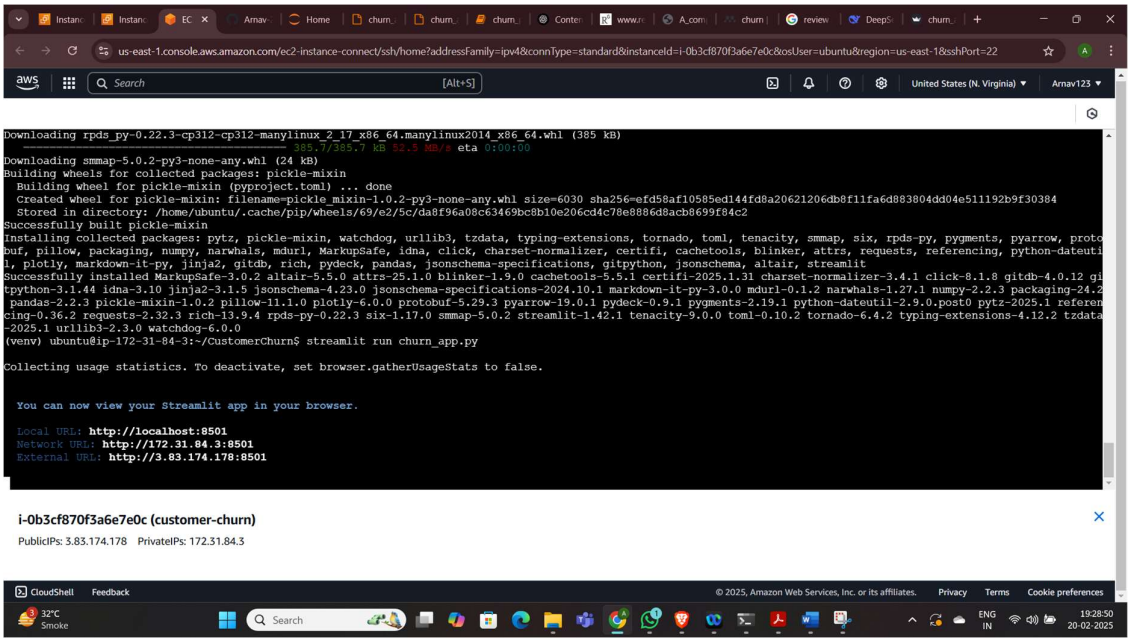
Install python: sudo apt install python3-pip

Install python virtual environment: apt install python3.12-venv

Create virtual environment: python3 -m venv venv

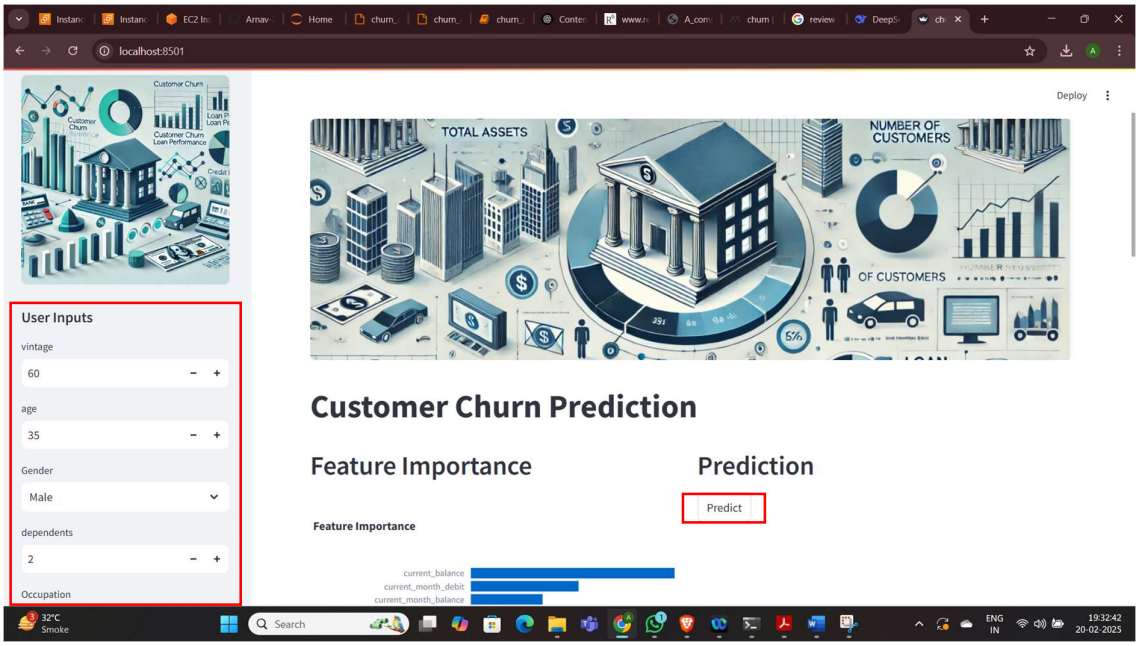
Install all the dependencies in the virtual environment

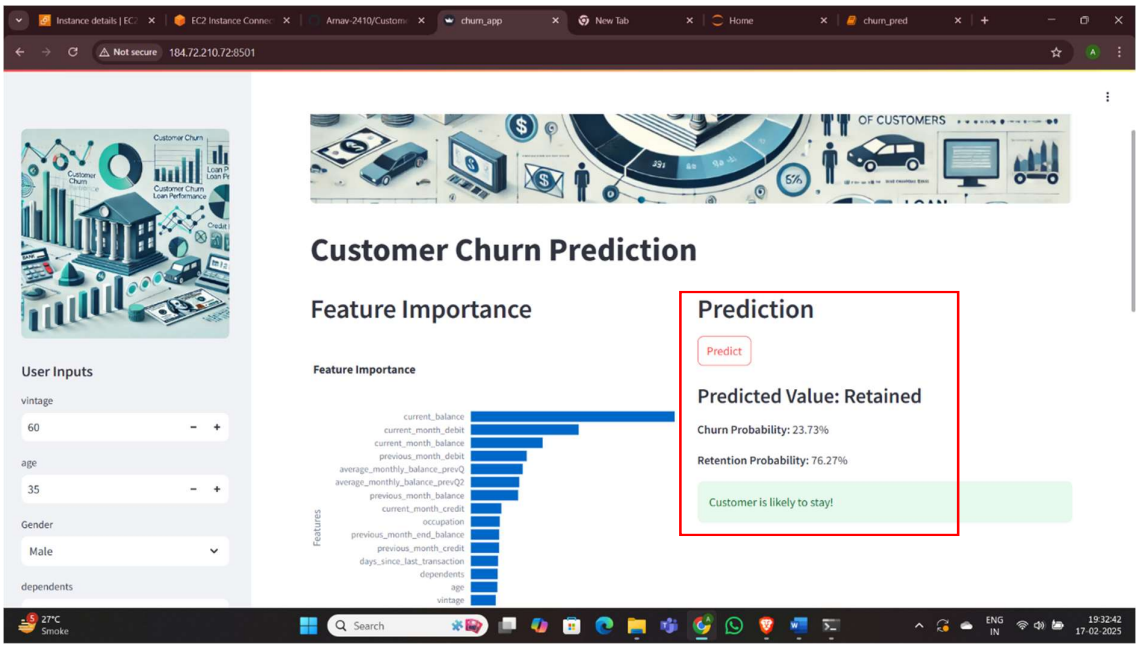
Deploy the streamlit app: streamlit run churn_app.py



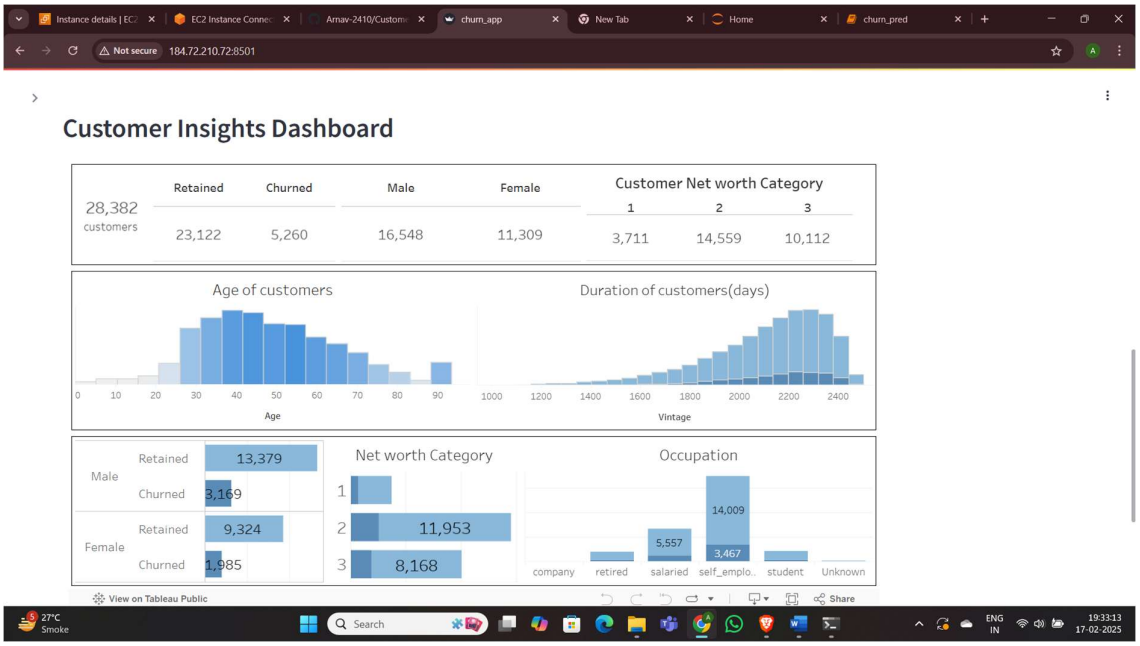
7. INTERFACE

Input the values for each feature in user inputs. Then click on predict:





I have also Integrated the tableau dashboard for customer insights into this web streamlit web app:



8. FUTURE SCOPE

With businesses increasingly focusing on customer retention, predicting churn has become essential for sustaining growth and profitability. Traditional methods often fail to capture the complex behavioural patterns that indicate a customer's likelihood of leaving, making machine learning-based approaches more effective. By leveraging advanced algorithms like deep learning and explainability techniques such as SHAP, companies can gain deeper insights into the factors driving churn. Real-time prediction systems integrated with streaming data can help organisations take proactive measures, while AI-driven automated retention strategies ensure personalised customer engagement. Expanding data sources, addressing biases, and deploying models across multi-cloud and edge environments will further enhance scalability and fairness. As industries like telecom, banking, and e-commerce adopt these advancements, churn prediction models will become indispensable tools for customer-centric decision-making.

9. CONCLUSION

The development of a **customer churn prediction system** using machine learning is a significant step towards improving customer retention and business sustainability. With rising competition across industries, it has become crucial for companies to identify customers likely to leave and take proactive measures to retain them. The model developed in this project helps businesses predict churn with greater accuracy, enabling data-driven decision-making and personalised engagement strategies. The system has wide-ranging applications across telecom, banking, e-commerce, and other customer-focused industries. However, further research is needed to enhance model interpretability, integrate real-time prediction capabilities, and expand feature sets for better accuracy. Overall, this system has great potential to help businesses reduce churn, improve customer satisfaction, and drive long-term growth.

10. REFERENCES

1. García, D. L., Nebot, À., & Vellido, A. (2017). Intelligent data analysis approaches to churn as a business problem: a survey. *Knowledge and Information Systems*, *51*(3), 719-774.
2. Hadden, J., Tiwari, A., Roy, R., & Ruta, D. (2007). Computer assisted customer churn management: State-of-the-art and future trends. *Computers & Operations Research*, *34*(10), 2902-2917.
3. Guliyev, H., & Yerdelen Tatoğlu, F. (2021). Customer churn analysis in banking sector: Evidence from explainable machine learning model. *Journal of Applied Microeconometrics*, *1*(2).
4. Rahman, M., & Kumar, V. (2020, November). Machine learning based customer churn prediction in banking. In *2020 4th international conference on electronics, communication and aerospace technology (ICECA)* (pp. 1196-1201). IEEE.
5. Mahesh, B. (2020). Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, *9*(1), 381-386.
6. Cunningham, P., Cord, M., & Delany, S. J. (2008). Supervised learning. In *Machine learning techniques for multimedia: case studies on organization and retrieval* (pp. 21-49). Berlin, Heidelberg: Springer Berlin Heidelberg.
7. Dayan, P., Sahani, M., & Deback, G. (1999). Unsupervised learning. *The MIT encyclopedia of the cognitive sciences*, 857-859.
8. Harmon, M. E., & Harmon, S. S. (1996). Reinforcement learning: A tutorial. *WL/AAFC, WPAFB Ohio*, *45433*, 237-285.
9. Python, W. (2021). Python. *Python releases for windows*, *24*.

10. Akkem, Y., Kumar, B. S., & Varanasi, A. (2023). Streamlit application for advanced ensemble learning methods in crop recommendation systems–a review and implementation. *Indian J Sci Technol*, *16*(48), 4688-4702.

11. Kleinbaum, D. G., Dietz, K., Gail, M., Klein, M., & Klein, M. (2002). *Logistic regression* (p. 536). New York: Springer-Verlag.

12. Breiman, L. (2001). Random forests. *Machine learning*, *45*, 5-32.

13. Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794).

14. Hanif, I. (2020). *Implementing extreme gradient boosting (xgboost) classifier to improve customer churn prediction*.

15. Saran Kumar, A., & Chandrakala, D. (2016). A survey on customer churn prediction using machine learning techniques. *International Journal of Computer Applications*, *975*, 8887.

16. Vafeiadis, T., Diamantaras, K. I., Sarigiannidis, G., & Chatzisavvas, K. C. (2015). A comparison of machine learning techniques for customer churn prediction. *Simulation Modelling Practice and Theory*, *55*, 1-9.

17. Lalwani, P., Mishra, M. K., Chadha, J. S., & Sethi, P. (2022). Customer churn prediction system: a machine learning approach. *Computing*, *104*(2), 271-294.