**AI Fusion Hackathon-Pickl.ai**

**Detailed Report**

On

**Bank’s Loan Campaign**

****

**By**

**Team**

**S.Digi.Aditya**

**Team Members**

Aditya Singh

Suyash Pandey

Vishal Sarup Mathur

**Contents**

1. **Problem Review**
   1. Initial Challenges
   2. Approach to the challenges
2. **Dataset Review**
   1. Initial Insights
   2. Visual EDA
   3. Non-Graphical/Graphical EDA
   4. Dashboard
   5. Final Dashboard Canvases
3. **Model Preparation**
   1. Initial Challenges/Insights
   2. Basic Data Preparation and Base Model Selection
   3. Accuracy Percentage Table
4. **Deployment**
   1. Initial Challenges
   2. Choosing Framework and Why
   3. Architecture Overview
   4. Final Deployment
   5. Testing
   6. Monitoring and Maintenance
5. **Useful Links**
   1. Github
6. **Final Review of the Project**

# **Problem Review**

## Initial Challenges

We faced a lot of initial challenges, some of them, along with the approach we used to proceed solving them, is given below:

1. We faced a big obstacle as soon as we got the data. We wondered what approach will be suitable for bringing this project to a reality. We thought of many things that we could do with the modeling perspective of things and some more things which we could include in the dashboard.
2. When we were in the initial phases of the planning, we observed that our schedules may not match exactly (due to the placement season), this meant there were very few times we may work together at the same time. This was a major problem since it very clearly implied we will not have a real time feedback loop, which may hamper the progress.

## Approach towards the Problems

Problems do bring headaches and worries but they also bring the opportunity to learn something new every time. Therefore, here are our approach towards the problems. It should also be remembered very precisely, that this is just ‘our’ approach to the problem which may or may not align with every other human being.

1. Initially, we stopped our overflowing creative juices, and decided to focus on just one model and one basic dashboard, honestly we thought that will be it, one model and one dashboard (foreshadowing). In a nutshell, we put a cork in our thoughts for a short time so we could stop procrastinating and start working.
2. This problem was a little bit more complex. To solve this we made use of multiple technologies, some of which are:
   1. Discord
   2. Google Colaboratory (Very Helpful)
   3. Google Docs
   4. Github

As you may have noticed, all these tools are used for setting up the initial real time feedback loop which I talked about earlier. Discord helped us to communicate, Google Colab helped us to share the notebooks, Google Docs helped us to document our process together at the same time and Github helped us to manage all that was being done.

# **Dataset Review**

## Initial Problems

1. For a total 48 entries in the dataset the experience was -1, -2 and -3 respectively.
   1. We converted the negative values to its respective positive values.

## Visual EDA

1. By a rough initial visual EDA, we came to the conclusion that almost all the features have some importance in the dataset but some held more importance than the others. These are:
   1. Age
   2. Education
   3. Fixed Deposit
   4. Income
   5. Mortgage (To be Explained)
2. Visual EDA posed a problem of not being sure whether our assumption about the importance of some features over others holds true in this case or not. Therefore, further analysis on the same is required.
3. Faced a problem in which we needed to calculate the Loan acceptance rate, so we decided to create a new measure by using a custom DAX query which calculated the yes\_percentage for the following.

Yes\_Percentage = DIVIDE(

COUNTROWS(FILTER('loan\_dataset', 'loan\_dataset'[Loan] = "yes")),

COUNTROWS('loan\_dataset'),

0

) \* 100

1. For the similar problems we made some more measures for more insights as per the following.

Average Professional Experience = AVERAGE('loan\_dataset'[T.Experience])

Digitally Enabled Customers Percentage =

DIVIDE(

COUNTROWS(

FILTER(

loan\_dataset,

loan\_dataset[Demat] = "yes" && loan\_dataset[Net Banking] = "yes"

)

),

COUNTROWS(loan\_dataset)

)\*100

LoanAcceptance Rate = DIVIDE(

COUNTROWS(FILTER('loan\_dataset', 'loan\_dataset'[Loan] = "yes")),

COUNTROWS('loan\_dataset'),

0

) \* 100

Percentage with Demat Account =

DIVIDE(

COUNTROWS(FILTER(loan\_dataset, loan\_dataset[Demat] = "yes")),

COUNTROWS(loan\_dataset)

)\*100

Percentage with Fixed Deposit =

DIVIDE(

COUNTROWS(FILTER(loan\_dataset, loan\_dataset[Fixed Deposit] = "yes")),

COUNTROWS(loan\_dataset)

)\*100

## **Graphical Non-Graphical EDA**

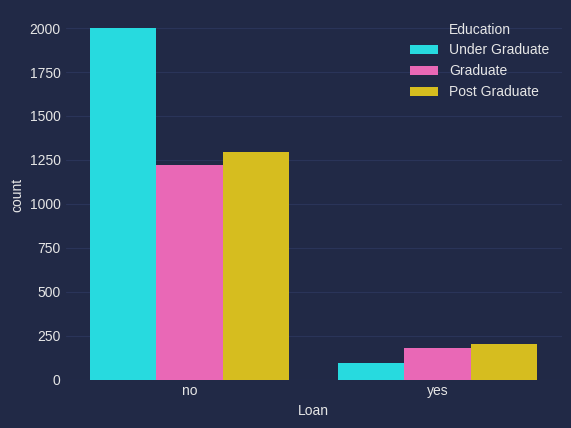
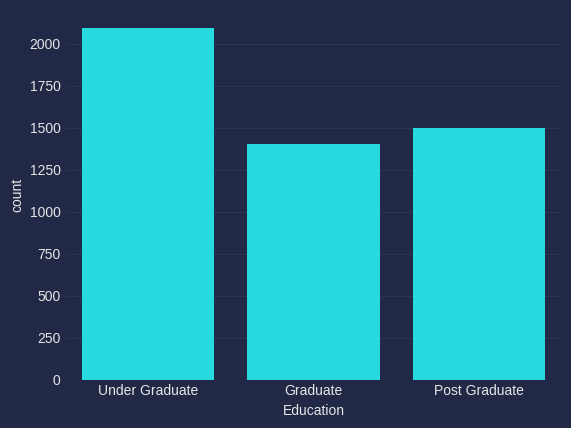
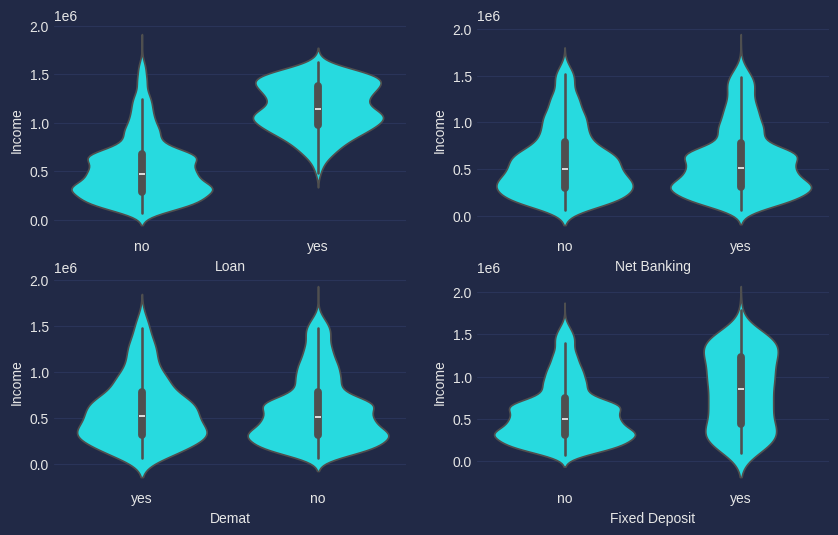
## 

## 

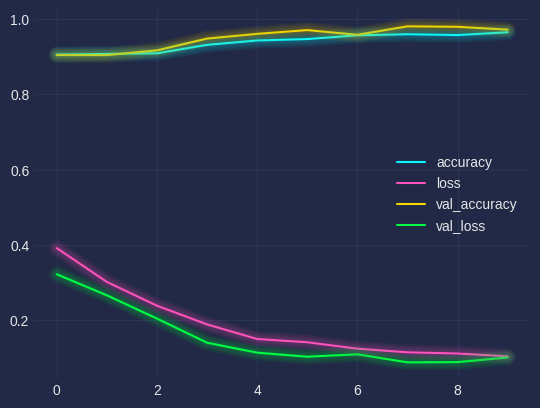
pandas.info() of the dataset showing the size, datatype, null value count of the dataset [with 0 null values].

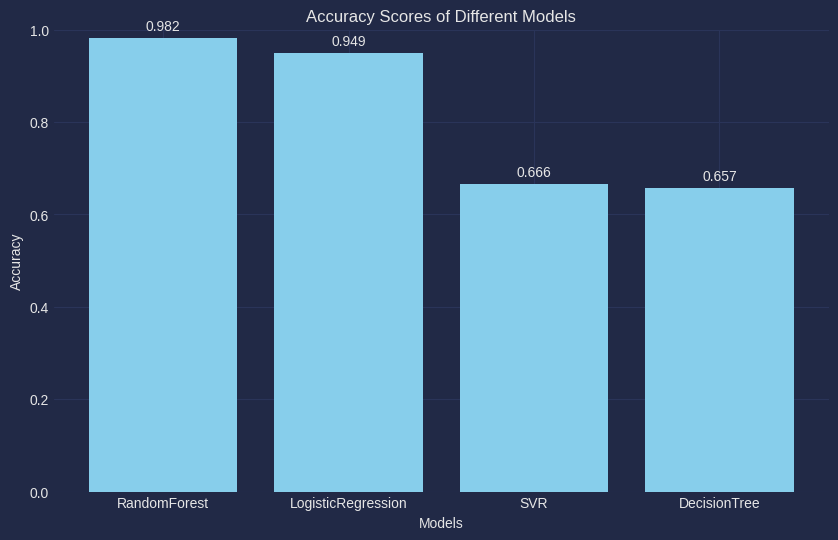
**Graphical EDA**

We plotted various plots and on the dataset to understand the distribution, training quality and outliers of different values across all the columns of our dataset.

****

## 





## 

## **Dashboard**

The Bank Dashboard provides a comprehensive visualization of key metrics and data insights related to customer banking behavior, segmented by various demographic factors such as age, income, education level, and geographical location. This dashboard is designed to assist bank management in making informed decisions by offering a deep dive into customer statistics and trends.

### **Dashboard Sections**

#### **1. Menu**

The Menu on the left side of the dashboard contains navigational options:

* **Home**: Returns to the main overview page of the dashboard.
* **Details**: Provides a detailed view of customer data and specific metrics.
* **Extra Insights**: Delivers additional analysis and deeper insights into the customer base.

#### **2. Home Page Insights**

The Home page provides a quick snapshot of the key metrics:

* **Age Slider**: Allows filtering of the data by age range.
* **Education Filter**: Select between Graduate and Post Graduate customers to see how education impacts various metrics.
  + **Count of Fixed Deposits and Median Age by Income**: This chart shows the distribution of fixed deposits across different income levels and the corresponding median age of customers within these income brackets.
  + **Count of Loans and Sum of Mortgages by Age**: This dual-axis graph illustrates the number of loans taken by customers of different ages, alongside the total sum of mortgages, highlighting potential age-related borrowing patterns.
  + **Loan Acceptance Rate by Age and Income**: A heatmap that reveals the loan acceptance rates across various age and income groups, offering insights into which demographic segments are more likely to get loans approved.
  + **Pin-code Visualization**: Geographical representation of the customer base within New Delhi, allowing the bank to visualize where most customers are located.
  + **Loan Acceptance Rate by Family Members**: A bar chart depicting how the number of family members influences loan acceptance rates, which could indicate financial stability considerations by the bank.

#### **3. Details Page**

The Details page offers a detailed breakdown of individual customer data and aggregate statistics:

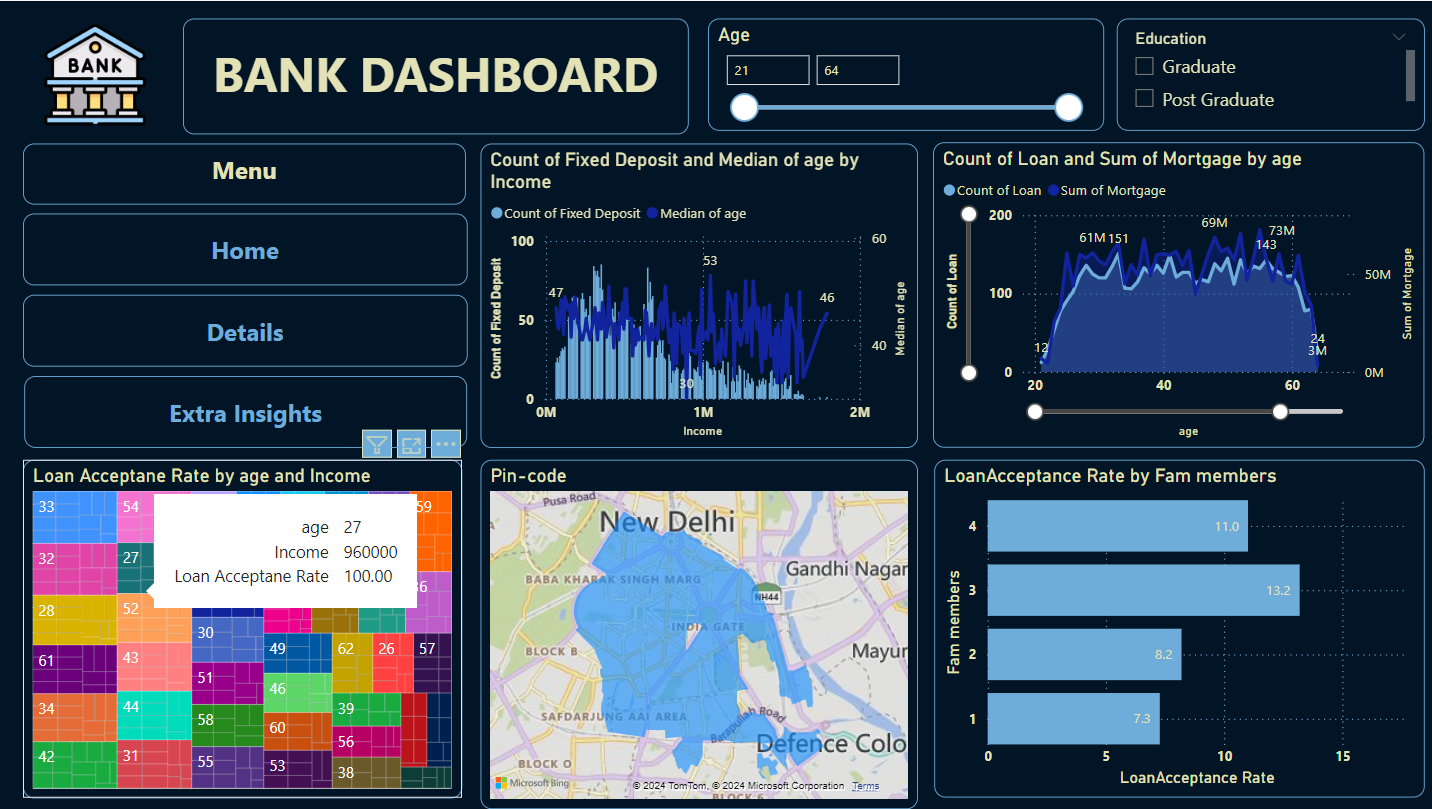
* **Key Metrics**: Summarized at the top, showing median family members, customers with fixed deposits, digitally active users, loan acceptance percent, average mortgage, and income.
* **Customer Table**: A detailed table lists customers by ID, age, pin-code, education, income, mortgage, and other financial products they hold. Filters for income and age are available, along with a selection for educational background (Graduate, Post Graduate, Under Graduate) to narrow down the data further.

#### **4. Extra Insights Page**

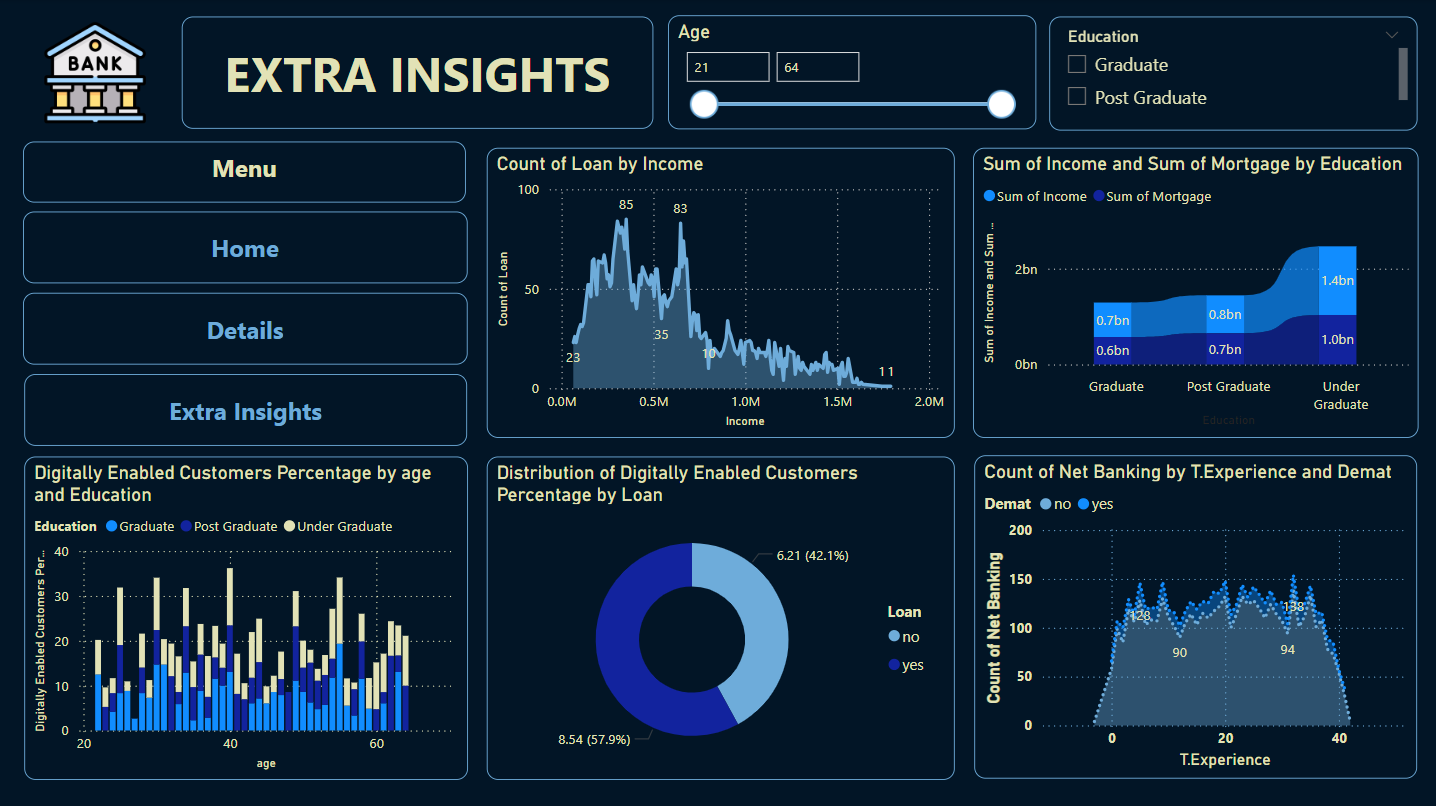
This page provides additional analytical views and insights:

* **Count of Loan by Income**: Shows how loan distribution varies across different income levels, identifying income segments with higher borrowing tendencies.
* **Sum of Income and Mortgage by Education**: Compares the total income and mortgage sums across different education levels, helping understand how education impacts financial capacity.
  + **Digitally Enabled Customers by Age and Education**: A bar chart analyzing the percentage of digitally active customers within different age groups and educational backgrounds, emphasizing the role of digital literacy in banking behavior.
  + **Distribution of Digitally Enabled Customers by Loan**: A pie chart breaking down digitally enabled customers by whether they have taken loans, underscoring the relationship between digital banking adoption and loan acquisition.
  + **Net Banking Usage by Experience and Demat Account**: A scatter plot showing how net banking usage correlates with customer experience and the presence of a Demat account, providing insights into the bank's digital engagement success.

## **Final Dashboard Canvases**







**Key Insights**

**Customer Demographics**

* **Age Range**: The dashboard includes filters for age, covering the range from 21 to 64 years.
* **Education Levels**: Customers are classified into three categories: Graduates, Post Graduates, and Under Graduates.

**Financial Products**

* **Fixed Deposits**: Trends in the number of fixed deposits are explored, with an analysis of their relationship to median age and income levels.
* **Loans**: The dashboard presents data on the number of loans issued and the total value of mortgages, segmented by age.

**Loan Acceptance Rates**

* The analysis focuses on how age and income levels impact loan acceptance rates.
* An additional section highlights loan acceptance rates by family size, showing variations based on the number of family members.

**Geographical Insights**

* A map of New Delhi is provided, illustrating pin codes and regions with different levels of customer distribution.

**Detailed Customer Data** The "Details" section includes specific metrics such as:

* **Median Family Members**: 3
* **Total Customers**: 562
* **Customers with Fixed Deposits**: 5.69%
* **Loan Acceptance Percentage**: 11.92%
* **Average Income**: 517.12K
* **Average Mortgage**: 419.66K

**Digitally Enabled Customers**

* The dashboard offers insights into the percentage of digitally enabled customers, categorized by age and education level.
* A significant portion of digitally enabled customers is identified as non-loan holders.

**Net Banking and Experience**

* The report analyzes the number of net banking users in relation to their years of experience and their demat account status.

# **Model Preparation**

## Initial Challenges and Insights

Initially, while we were studying the data for both EDA and predictive modeling, we came across the problem of the dataset being unbalanced, table 1 shows the percentage ‘yes’ and no values in the dataset.

|  |  |
| --- | --- |
| **Final Values** | **Percentage** |
| ‘yes’ | 9.6 |
| ‘no’ | 90.4 |

Table 1

This clearly shows the dataset being unbalanced and therefore, the need to balance the dataset arises. But before doing that we wanted to perform the predictive modeling on the original dataset just to see what happens. On the deployed website, we have given the user the option to choose between models that are trained on each of the dataset i.e, the original and the modified one.

Furthermore, as mentioned above there were negative values present in the dataset which needed to be changed to its corresponding absolute value. More precisely, there are some -1, -2 and -3 values ,in the T.Experience attribute, present in the dataset which we either need to remove or convert to its corresponding absolute value.

Therefore, before starting the predictive modeling, we performed the following two changes:

1. Converted the negative values present in the T.Experience column to positive values using a lambda function.

df['T.Experience'] = df['T.Experience'].apply(lambda x: abs(x))

df.head(10)

After this operation we confirm whether there are any negative values left in the latter exclaimed column. After we are sure that there are no such values present we begin the next step in the process which will be explained a little later.

1. We removed records containing ‘no’ values to balance the dataset and then retrain the models on the modified data.

We performed the first change before beginning the first training cycle and then after all our models are trained on the original data, we train the models again on the modified data.

## **Model Training and Base Model Selection**

### Models Used

**Logistic Regression**

Logistic regression is a commonly used supervised learning algorithm for classification tasks. It predicts the probability that a given instance belongs to a particular class. By modeling the relationship between a dependent variable and one or more independent variables, logistic regression estimates the likelihood of a specific outcome.

**Random Forest Classifier**

Random Forest is an ensemble learning method that constructs multiple decision trees during training. Each tree is built using a random subset of the data, and for each split, a random subset of features is chosen. This randomness reduces the risk of overfitting and enhances the overall predictive performance by aggregating the results from all the trees.

**Stochastic Gradient Descent (SGD)**

Stochastic Gradient Descent is an optimization algorithm used to minimize the cost function in machine learning. It iteratively updates the model parameters to reduce errors, aiming to find the optimal parameters that result in the highest accuracy for both training and test datasets. In essence, the algorithm adjusts the model's weights in the direction that reduces the prediction error.

**Support Vector Machine (SVM)**

Support Vector Machine is a versatile machine learning algorithm used for both classification and regression tasks. It works by finding the hyperplane that best separates different classes in the feature space. SVMs are particularly effective in high-dimensional spaces and are used in various applications such as text and image classification, anomaly detection, and more.

**ExtraTrees**

Extra Trees, similar to Random Forests, is an ensemble learning method that creates multiple decision trees. However, unlike Random Forests, Extra Trees select splits in a random manner, without considering the best split points. This randomness helps in creating diverse trees, which, when aggregated, provide robust predictions.

**AdaBoost**

AdaBoost, or Adaptive Boosting, is an ensemble technique that combines multiple weak classifiers to create a strong classifier. It works by focusing more on the instances that were previously misclassified, thereby improving the overall model accuracy. The final model is a weighted combination of all the weak classifiers, with more accurate models given higher weight.

**XGBoost**

XGBoost, short for Extreme Gradient Boosting, is a powerful gradient boosting algorithm designed for speed and performance. It combines the outputs of several weak models to produce a stronger predictive model. XGBoost is known for its ability to handle large datasets and is widely used in tasks requiring high accuracy, such as classification and regression.

**Artificial Neural Network (ANN)**

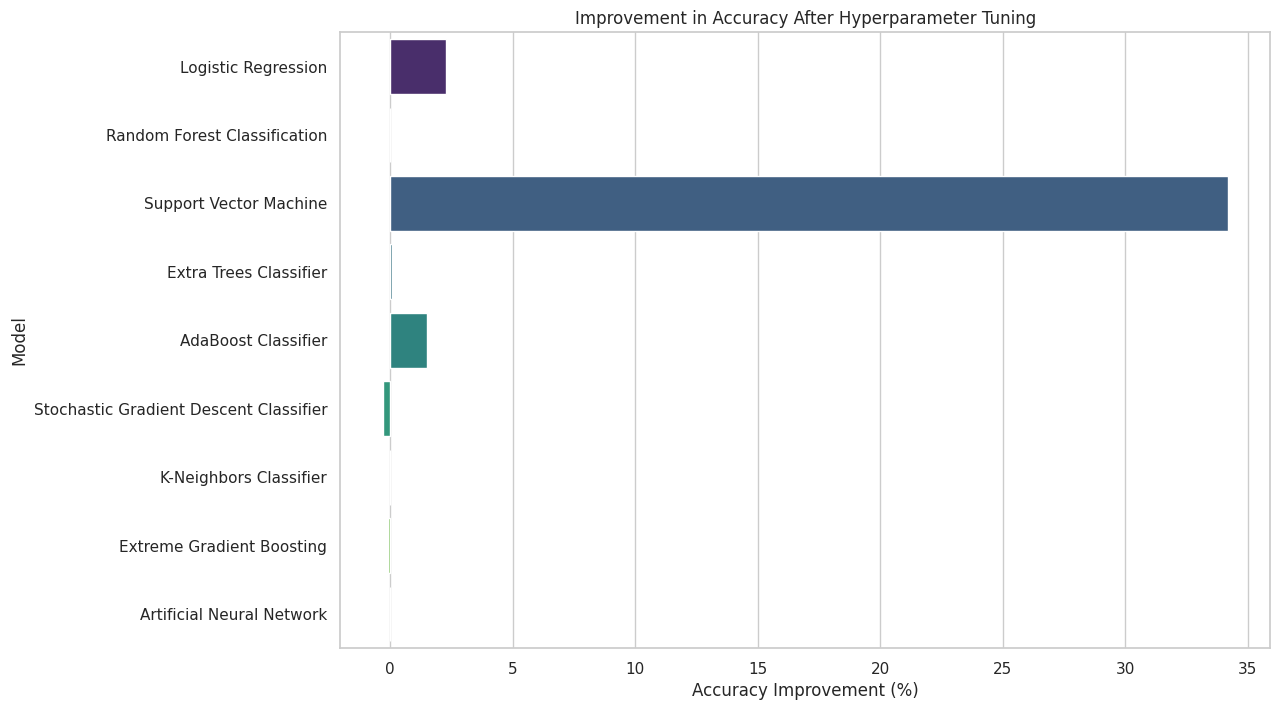
Artificial Neural Networks are computational models inspired by the human brain's structure and function. They consist of layers of interconnected neurons that process and learn from data. Neural networks are particularly useful in tasks like pattern recognition and decision-making. The network's components include neurons, connections, weights, biases, and activation functions. Through learning, the network adjusts its parameters to improve its performance, refining its predictions over multiple iterations.

## Accuracy Percentage Table

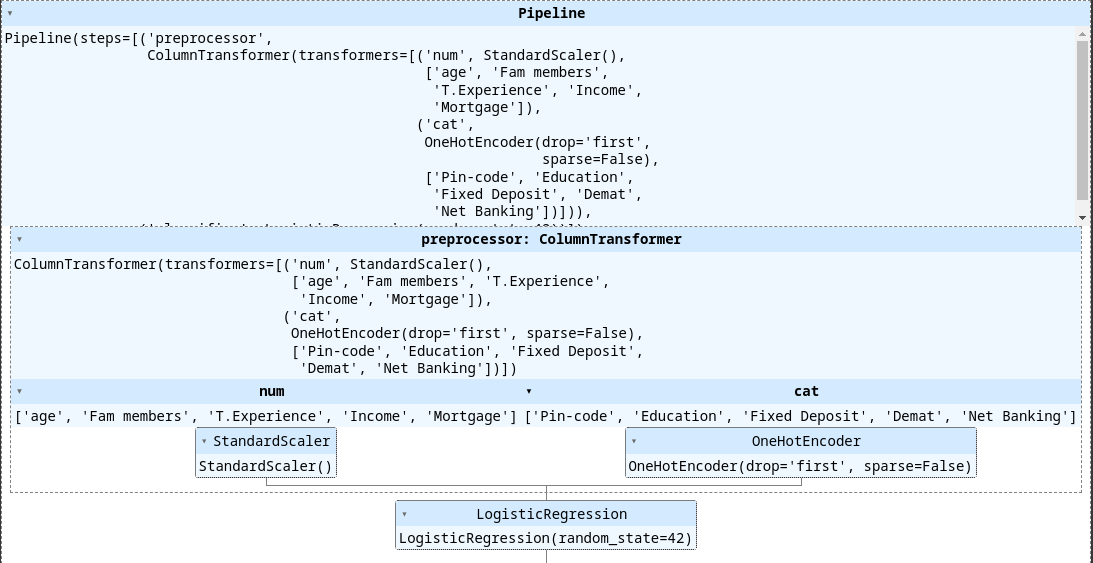
| **S.No** | **Model** | **%Accuracy** | **%Accuracy (Hyperparameter Tuned)** |
| --- | --- | --- | --- |
| 1 | Logistic Regression | 94 | 96.3 |
| 2 | Random Forest Classification | 98.2 | 98.2 |
| 3 | Support Vector Machine | 64 | 98.2 |
| 4 | Extra Trees Classifier | 97.7 | 97.8 |
| 5 | AdaBoost Classifier | 95 | 96.5 |
| 6 | Stochastic Gradient Descent Classifier | 95.7 | 95.4 |
| 7 | K-Neighbors Classifier | 97.8 | 97.8 |
| 8 | Extreme Gradient Boosting | 98.4 | 98.3 |
| 9 | Artificial Neural Network | 98 | 98 |

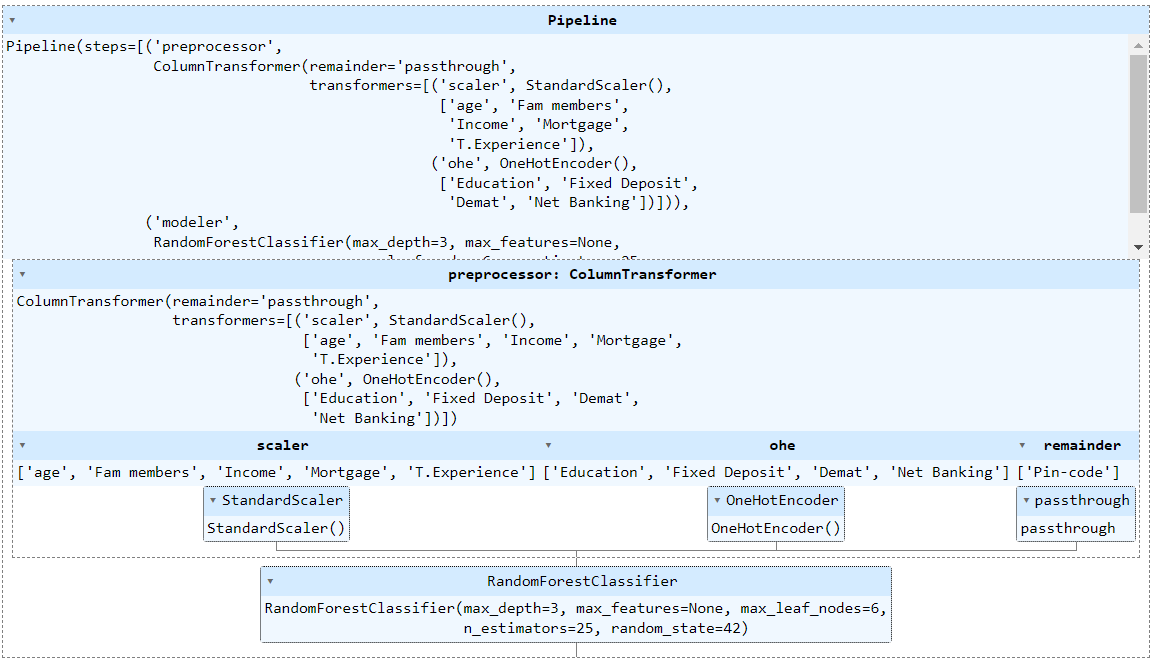
Table 2

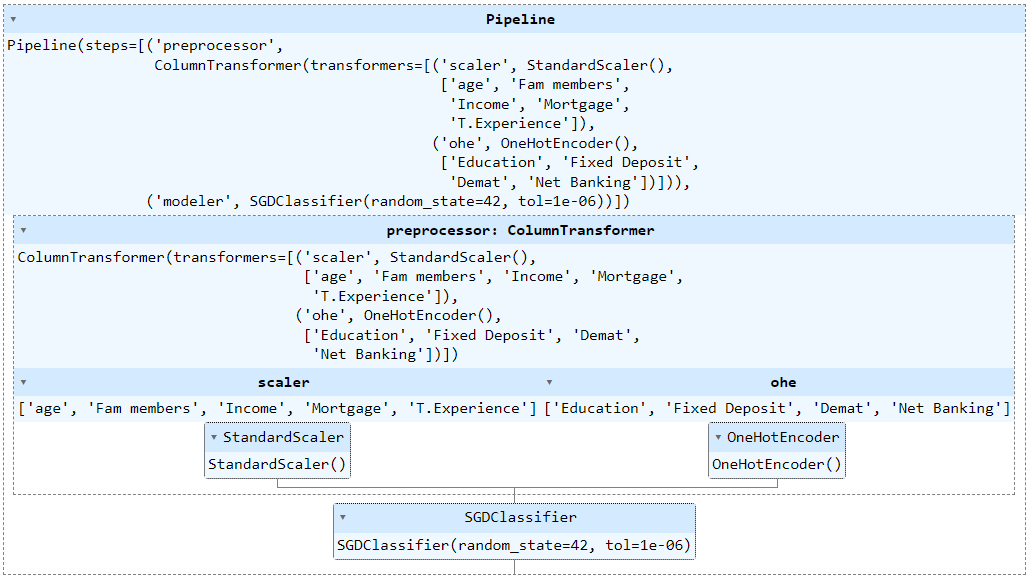
Table 2 shows the different accuracies we achieved after training our model in the first cycle and then hyperparameter tuned them for the second cycle, although many of them showed very little improvement some displayed a respectable jump in accuracy like the Logistic Regression algorithm which jumped from 94 to 96.3 of accuracy and a huge improvement for the SVM algorithm which for a massive difference. To make this a lot more clear, we also plotted a graph which shows the percentage improvement of the models after hyperparameter tuning.



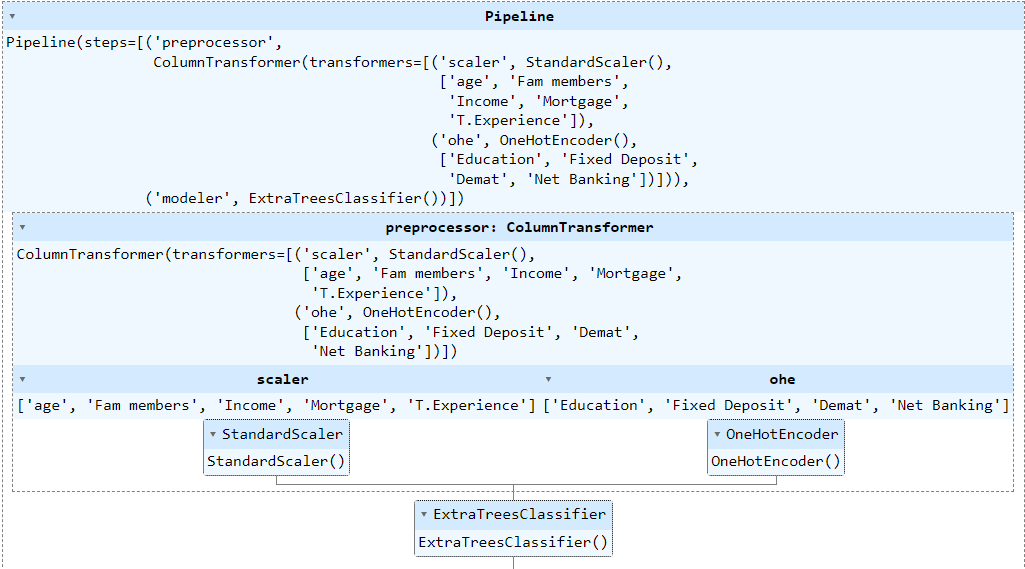
We created pipelines for each model individually so the process of assessing the model along with monitoring and later, viewing what parameters were used to train the model, became a whole lot manageable. A display of all the pipelines used for training the images were added below.

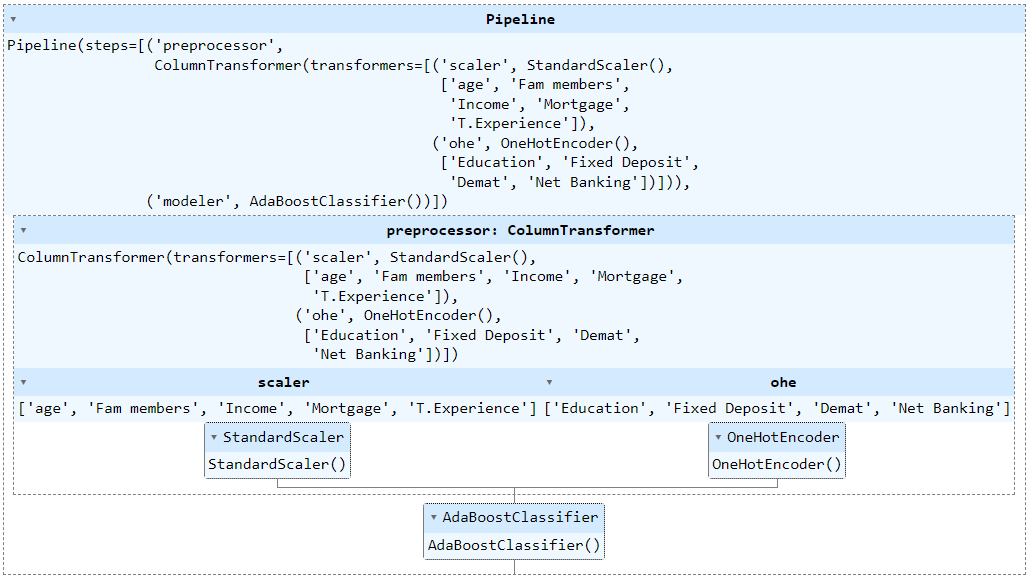
1. **Logistic Regression**

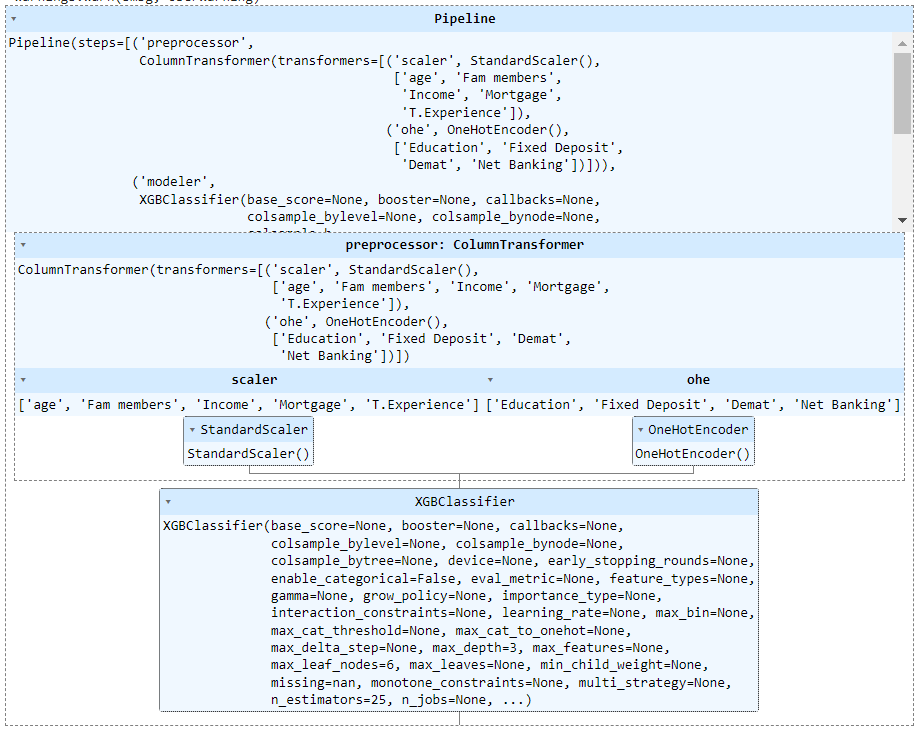
1. **Random Forest Classifier**
2. **Stochastic Gradient Descent (SGD)**



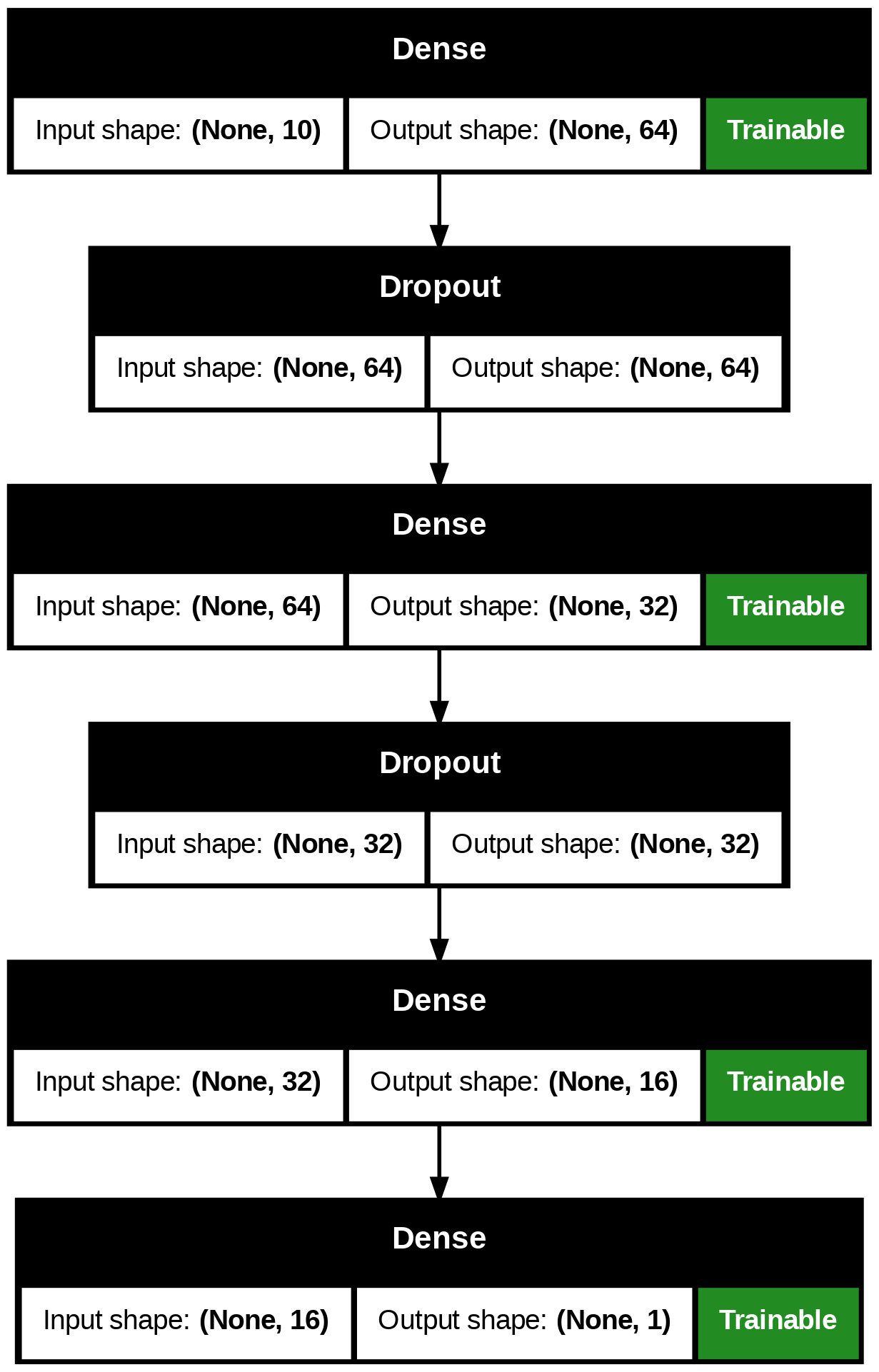
1. **Support Vector Machine (SVM)**
2. **ExtraTrees**



1. **Adaboost**
2. **XGBoost**



1. **Artificial Neural Network**



## 

# **Deployment**

### Initial Challenges

While performing the deployment of the web application the major challenges that we faced was choosing a good framework that works well without any lag in the web application.There are more than one model present in the web application for the user’s choice to select model of their own choice, hence the management of the models efficiently and effectively, is a bit tricky task and has to be carried out without any mistake.

The deployment environment might differ from the development environment, leading to issues like dependency conflicts, compatibility problems or unexpected behavior. Ensuring that the application can handle the increase in the user load without performance degradation, can be challenging. Performance optimization is another major challenge faced during the deployment of the application. Framework used should be platform independent so as to facilitate the management of the application efficiently and effectively.

Furthermore, when we decided to deploy more than one model, we sort of invited the added complexity problem. Though it is a bit mind numbing seeing the same error for the last 30 mins and not knowing which model is causing it is still an adventure of its own. But this added complexity, although it increased the complexity of the whole project but it also brought about interesting bugs . Still the major question in the readers’ mind must be,”What did you deploy your application and why?”.

## **Choosing Framework and Why**

Streamlit can be a highly effective framework for building web applications, particularly for those focused on data analysis and visualization.

### **1. Simplicity and Ease of Use**

* **Direct Python Development:** Streamlit allows you to create web applications directly using Python, without needing to learn additional web technologies like HTML, CSS, or JavaScript.
* **Minimal Setup:** With Streamlit, you can build functional web apps with just a few lines of code, making the development process straightforward.

### **2. Interactive Features**

* **Built-In Widgets:** Streamlit offers a range of built-in widgets such as sliders, buttons, and text inputs, which can be used to make your applications interactive.
* **Automatic Updates:** The app updates automatically as users interact with the widgets, streamlining the process of creating dynamic user interfaces.

### **3. Support for Visualizations**

* **Seamless Integration:** Streamlit works well with popular Python libraries for data visualization, such as Matplotlib, Plotly, and Altair.
* **Live Data Visualizations:** You can easily display and update visualizations in real time, making Streamlit suitable for data-driven applications.

### **4. Rapid Development**

* **Instant Feedback:** Streamlit allows for live previewing of changes, so you can see the results of your code as you write it.
* **Quick Prototyping:** It’s particularly useful for quickly building prototypes and iterating on ideas.

### **5. Community and Resources**

* **Active Community:** Streamlit has a growing community with plenty of tutorials, plugins, and components available to extend its functionality.
* **Open Source:** Streamlit is freely available, with an active community supporting and contributing to its development.

### **6. Easy Deployment**

* **Simple Deployment:** Streamlit apps can be easily deployed on platforms like Streamlit Sharing, Heroku, or Docker, making it easy to share your application with others.
* **Scalability:** While Streamlit is designed to be simple, it can be scaled for more complex applications by deploying on cloud platforms.

### **7. Ideal for Data Science Projects**

* **Designed for Data Projects:** Streamlit is particularly well-suited for projects that involve data analysis, visualization, or exploration, allowing for quick development and testing.

## Architecture Overview

* **Frontend**: A web-based user interface allowing users to input their details for personalized lending predictions.
* **Backend**: An API that processes user inputs, interacts with the machine learning model, and returns the predictions.
* **Database**: A storage solution for user data, input values, and prediction results.
* **Machine Learning Model**: A pre-trained model that processes user inputs and provides lending predictions.

**Technology Stack**

* **Frontend**:
  + **HTML/CSS**: Used for structuring and styling the user interface.
  + **JavaScript Framework**: Utilizes React, Angular, or Vue.js for creating dynamic and interactive user experiences.
* **Backend**:
  + **Node.js with Express** or **Python with Flask/Django**: For handling HTTP requests and managing API endpoints.
* **Database**:
  + **PostgreSQL** or **MongoDB**: For storing user details and prediction outcomes.
* **Machine Learning**:
  + **Python** with libraries such as **scikit-learn** or **TensorFlow**: Employed for developing the predictive model.
* **Deployment**:
  + **Docker**: For containerizing the application.
  + **Kubernetes**: For managing and orchestrating the containers (optional based on scale).
  + **CI/CD Tools**: Using Jenkins or GitHub Actions for continuous integration and deployment.

**Deployment Process**

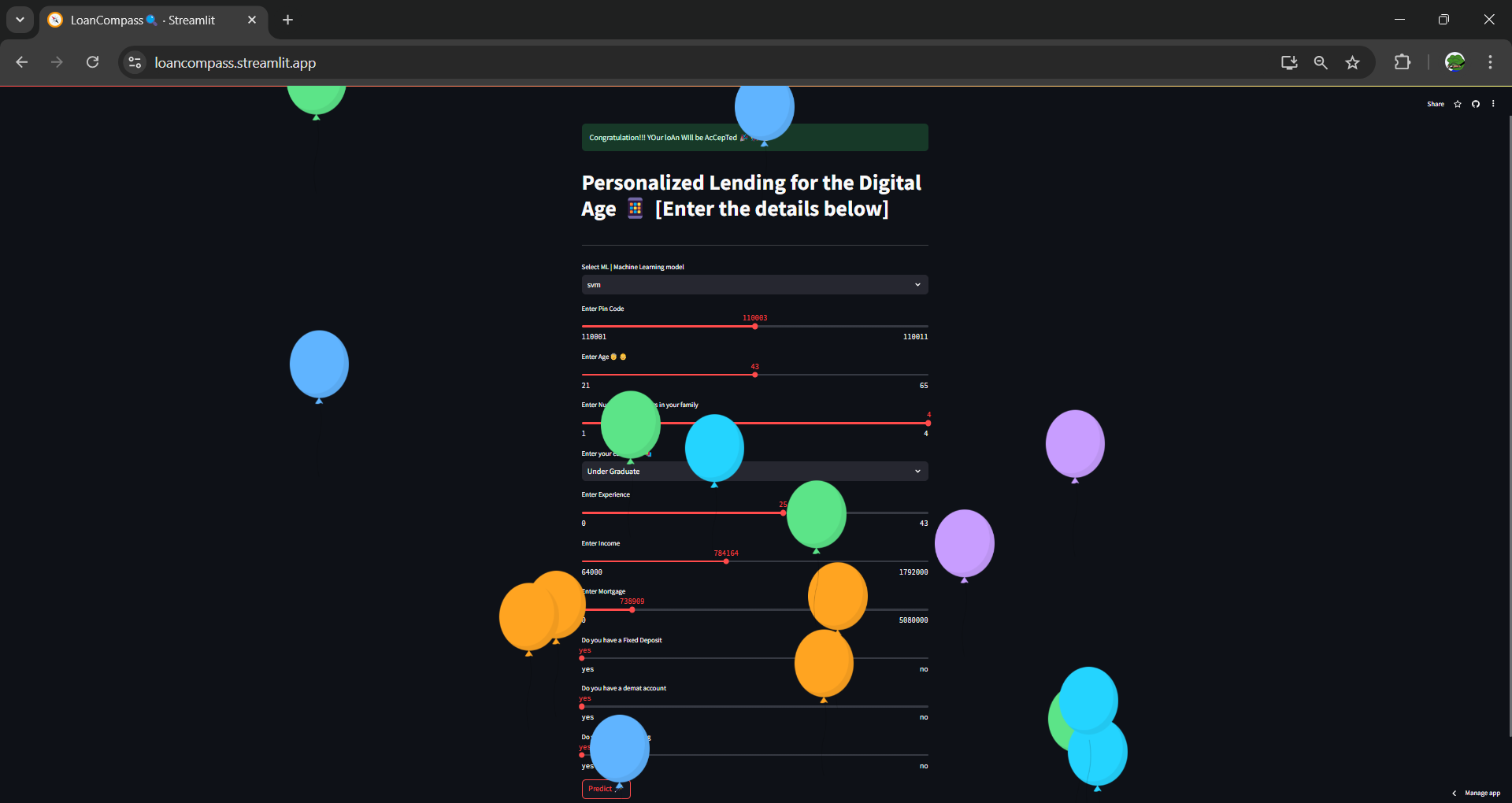
* **Step 1: Containerization**
  + Create Docker images for the frontend and backend.
  + Write Dockerfiles to define the build and runtime environment for each component.
* **Step 2: Cloud Setup**
  + Select a cloud provider (AWS, Azure, or Google Cloud).
  + Set up a Virtual Private Cloud (VPC) for enhanced security.
  + Configure networking components such as load balancers and security groups.
* **Step 3: Database Deployment**
  + Deploy the database service (e.g., AWS RDS for PostgreSQL).
  + Initialize the database schema to support user inputs and predictions.
* **Step 4: Backend Deployment**
  + Deploy the backend API using Kubernetes or a similar service.
  + Ensure the backend has connectivity with the database and the machine learning model.
* **Step 5: Frontend Deployment**
  + Host the frontend on a service like AWS S3, Azure Blob Storage, or Vercel.
  + Confirm that the frontend can successfully communicate with the backend API.
* **Step 6: Monitoring and Logging**
  + Implement monitoring tools like Prometheus and Grafana to track performance metrics.
  + Set up logging with tools like the ELK stack to capture and analyze application logs.

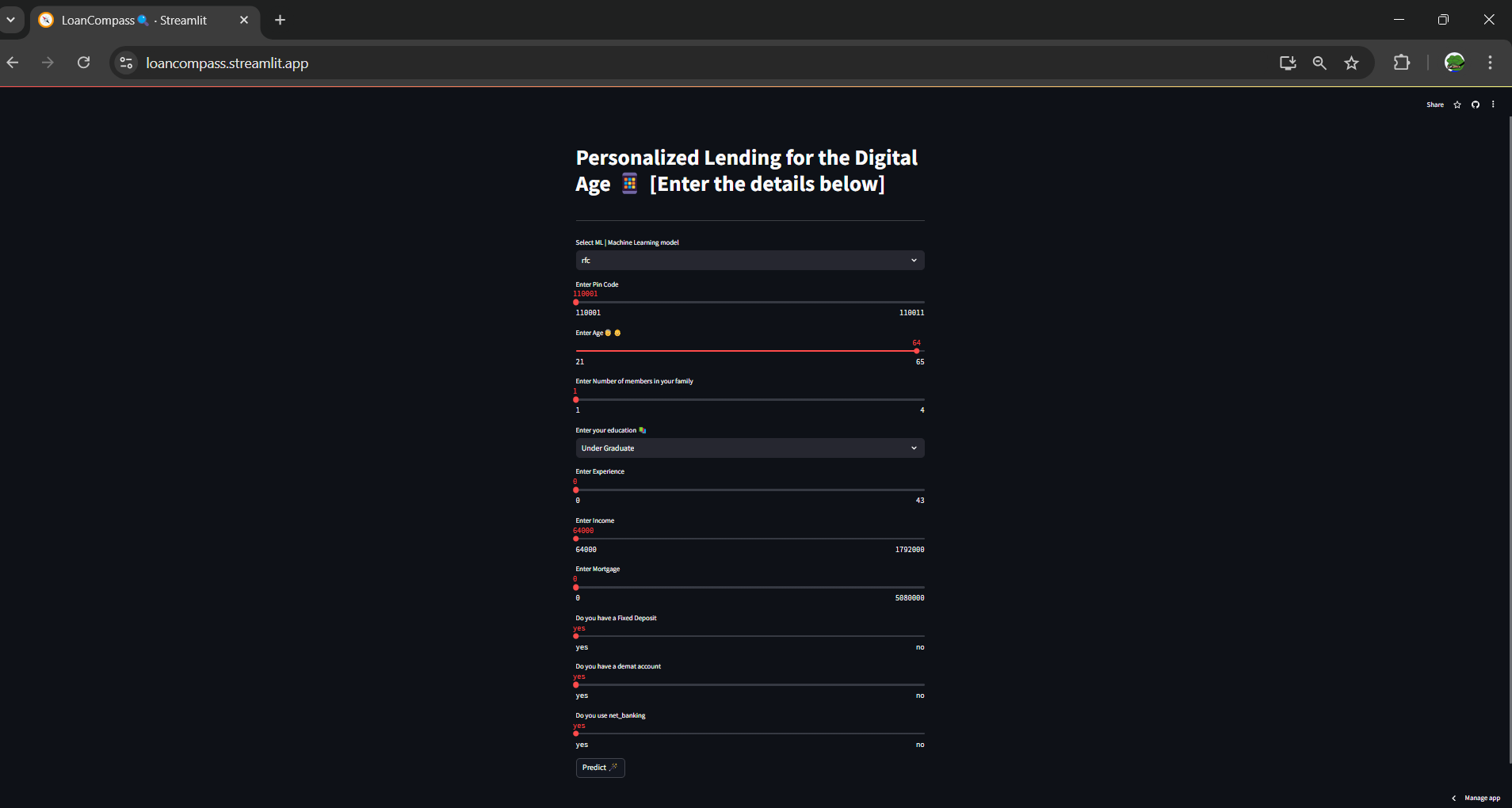
## 

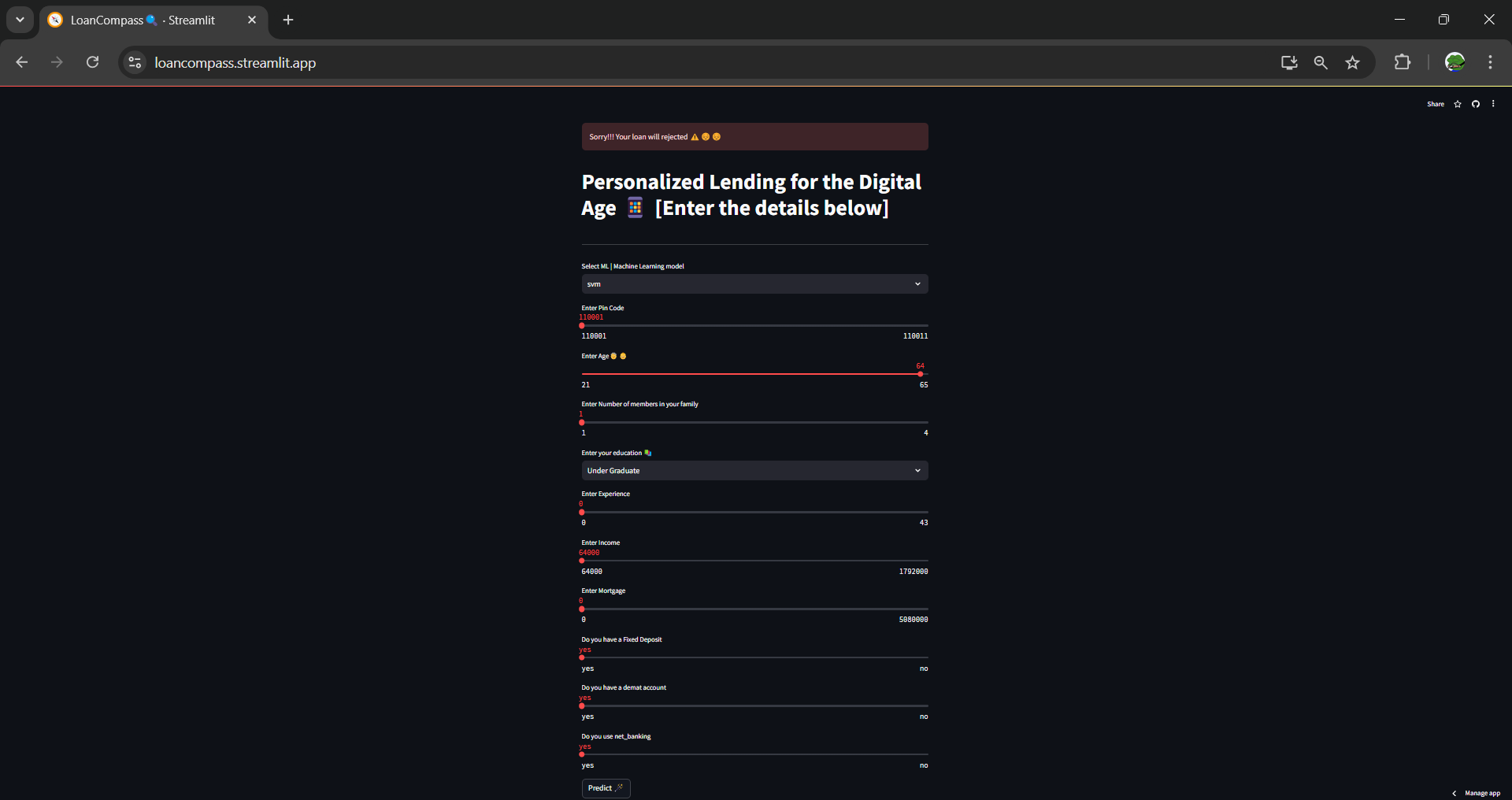
## 

## 

## **Final Deployment**







## Testing

* **Unit Testing**: Develop unit tests for individual components of the frontend and backend.
* **Integration Testing**: Test the interaction between the frontend and backend to ensure smooth data flow.
* **User Acceptance Testing (UAT)**: Conduct testing with end-users to gather feedback and make necessary adjustments.

## Monitoring and Maintenance

* **Performance Monitoring**: Continuously monitor the application's performance and user engagement metrics.
* **Error Tracking**: Use error tracking tools like Sentry to log and monitor errors in real time.
* **Updates and Maintenance**: Regularly update the application for security fixes and feature enhancements.

# **Useful Links**

## Github

* [Main Link](https://github.com/op-12/Pickl_AI)
* [PowerBI Dashboard](https://github.com/op-12/Pickl_AI/tree/main/Dashboards)
* [Models](https://github.com/op-12/Pickl_AI/tree/main/models)

# **Final Review of the Project**