Loan Prediction Web Application

BASANI ANANTH

Step - 1:Prototype Selection

Problem Statement:

The loan approval process often presents challenges such as uncertainty for applicants regarding eligibility and the need for efficient decision-making by financial institutions managing numerous applications. To address these issues, the Loan Prediction Web Application leverages machine learning to predict loan approval outcomes based on user input and historical data. This innovative tool aims to streamline the application process, reduce delays, and support financial institutions in making more informed lending decisions, thereby enhancing the overall experience for both applicants and lenders.

Market/Customer/Business Need Assessment

Market/Customer/Business Need Assessment is a crucial step in any ML project as it helps to understand the problem from different perspectives and identify the key challenges, risks and growth opportunities. For a Loan Prediction Web Application project, the assessment can be as follows:

Market Need Assessment

- **Growing Demand for Loans:** With increasing consumer demand for personal, auto, and home loans, there is a pressing need for efficient loan processing systems. Individuals often seek quick and easy methods to ascertain their eligibility before applying.
- **Increasing Loan Applications:** With a rising number of individuals seeking loans for personal, educational, and business purposes, there is a growing demand for streamlined application processes. Customers need quick insights into their loan eligibility to make informed financial decisions.
- **Digital Transformation:** The financial services sector is undergoing a digital transformation, with more consumers preferring online solutions. This trend underscores the need for web-based applications that provide instant feedback on loan applications.

Customer Need Assessment

- **Instant Gratification:** Customers expect immediate responses when they apply for loans. Traditional loan approval processes can take days or weeks, leading to frustration. The Loan Prediction Web Application addresses this need by providing real-time predictions based on user input.
- **Informed Decision-Making:** Customers need a reliable method to understand their likelihood of loan approval, which can aid them in preparing their applications or exploring alternative options.
- **Transparency:** Consumers are increasingly seeking transparency in the lending process. They want to understand the factors influencing their loan approval chances. The application can demystify this process by using data-driven insights to explain the prediction outcomes.
- **User-Friendly Experience:** A simple and intuitive interface is essential to attract users, particularly those who may not be tech-savvy. The Loan Prediction Web Application caters to this need by offering an accessible platform for all users.

Business Need Assessment

- **Efficiency for Financial Institutions:** Banks and lending agencies are inundated with applications, which can overwhelm traditional processing systems. The application aids in reducing workload by quickly filtering potential candidates based on predictive analytics, allowing institutions to focus on more complex cases.
- **Enhancing Customer Experience:** Providing a tool that quickly assesses loan eligibility helps attract new customers. A positive initial experience can lead to higher customer retention rates as satisfied users are more likely to return for future financial needs.
- **Building Trust:** Transparency in the loan approval process fosters trust between lenders and customers. By offering a reliable prediction tool, lenders can enhance their credibility and strengthen customer relationships.
- **Competitive Advantage:** As the lending market becomes increasingly competitive, financial institutions must leverage technology to maintain their edge. The Loan Prediction Web Application offers a modern solution that aligns with industry trends, helping businesses stand out.
- **Responding to Economic Changes:** In a fluctuating economy, lenders need the ability to adapt quickly to changing market conditions. The Loan Prediction Web Application can provide insights into current lending trends, enabling institutions to make timely adjustments to their loan offerings.

Target Specifications and Characterization

Target specifications and characterization depends on the specific business and industry, but generally, the following characteristics can be considered for identifying and predicting Loan's:

- Age: Typically 18-65 years old, with a focus on young adults (20-35) and mid career professionals (35-50).

- Income Level: Varied income levels, but generally middle-income to upper middle-income individuals seeking personal, home, or auto loans.
- Education: Primarily educated individuals, including college students, graduates, and working professionals.
- Technological Proficiency: Comfortable using online tools and applications, with varying levels of tech-savviness.
- Types of Institutions: Banks, credit unions, online lenders, and fintech companies.
- Decision-Makers: Loan officers, risk assessment teams, and financial product managers looking for innovative solutions to enhance their lending processes.
- Need for Quick Solutions: Individuals looking for immediate feedback on their loan applications to make informed decisions.
- Desire for Transparency: Customers who value clarity regarding their loan eligibility and the factors influencing approval.
- Interest in Customization: Users seeking tailored loan options that align with their financial situations and goals.
- Variety of Loan Types: Interest in different types of loans, including personal loans, mortgages, and auto loans.

Targeting the right customers is crucial for the success of any financial application, such as the Loan Prediction Web Application. By utilizing data-driven insights and predictive analytics, businesses can identify and focus on individuals who are most likely to benefit from their services. This strategic approach not only optimizes resource allocation but also enhances customer satisfaction by providing personalized solutions. Ultimately, effective customer targeting fosters stronger relationships, drives engagement, and contributes to achieving long-term business objectives.

External Search (online information sources/references/links):

Dataset:

https://www.kaggle.com/datasets/altruistdelhite04/loan-prediction-problem-dataset

Applicable Constraints:

There are several constraints that should be taken into consideration when conducting a loan prediction analysis, some of which include:

Space Constraints

- **Cloud Infrastructure:** The web application will require hosting on cloud platforms like AWS, Azure, or Google Cloud. These services offer scalable storage solutions, but space usage can escalate based on traffic and data storage needs.
- Storage for Machine Learning Models and User Data: As the application collects user inputs, stores model predictions, and integrates machine learning models, you need adequate storage. Consider constraints around,

- Data Retention: How long user data needs to be stored (for repeat users or performance tracking).
- Size of the Machine Learning Models: Depending on the model's complexity, storage for models serialized with Joblib or similar tools may also increase.
- Database Space: If you're storing user data, prediction logs, or financial data, database solutions (e.g., PostgreSQL, MongoDB) will require sufficient space. The need for real-time performance might require using caching systems (e.g., Redis) that also have space constraints.

Budget Constraints

- Development Tools and Frameworks: Although frameworks like Node.js, Express.js, Python, and EJS are open-source and free to use, there will be costs associated with integrating third-party services (e.g., for cloud hosting, payment gateways).
- Developer Costs: You will need experienced developers for frontend (UI/UX) and backend (model integration, API design). Depending on the expertise level required, developer salaries or freelance rates will be a significant part of the budget.
- Design: Creating a user-friendly interface may require hiring UI/UX designers, which could be an additional cost.

Infrastructure and Hosting

- Cloud Hosting: Budget for the cloud hosting service. Costs will depend on traffic, data storage, and computational resources required for running the machine learning models. As your user base scales, hosting costs can increase.
- Data Security and Encryption Services: To comply with regulations, you may need third-party security services (encryption, firewalls, DDoS protection) which will add to the budget.
- Licensing Costs: While most libraries (Pandas, scikit-learn, Joblib) are open-source, integrating proprietary financial APIs (credit bureaus, payment gateways) could have licensing fees.

Time Constraints

- Development Timeline: Depending on the complexity of the project, timelines for the initial build and testing phase could range from several weeks to a few months. A tight deadline may constrain the development scope, forcing you to prioritize certain features over others.
- Model Training and Tuning: Machine learning model training can be time-intensive, especially if you're using large datasets. Tuning hyperparameters and testing models for optimal performance may take additional time.
- Compliance and Legal Review: Ensuring compliance with data privacy and financial regulations could slow down the launch if you need to consult legal advisors, especially with evolving financial laws in India.

Business Model

Subscriptions

Basic Subscription: Charge monthly or annual subscription fees for access to the loan prediction platform.

Tiered Subscription Plans:

- Basic Plan: Access to basic loan prediction features with limited usage or a fixed number of predictions per month.
- Premium Plan: Includes advanced features like higher prediction accuracy, custom model integration, and enhanced support.
- Enterprise Plan: Custom pricing for large financial institutions with extensive use of predictions and API access.
- Value Proposition: Financial institutions gain access to a scalable loan prediction solution that improves decision-making and increases loan approval efficiency.

Partnerships/Commissions(Partnership with Lenders)

Target Users: Banks, credit unions, and lending companies.

Monetization Approach: Partner with financial institutions and lenders to provide loan applicants who meet their criteria through your prediction system. Charge a commission or referral fee for each successful loan application that converts through your platform. You could also partner with multiple lenders, giving users the ability to compare loan products and terms. Charge a fee for each lead sent to lenders, regardless of loan approval status.

Value Proposition: Lenders benefit from receiving pre-qualified leads, improving their approval rates and reducing the cost of customer acquisition.

Users benefit by receiving loan offers from multiple lenders based on their predicted eligibility.

Concept Generation

The concept generation process for the **Loan Prediction Web Application** begins with identifying the problem: the need for a more efficient and reliable loan approval prediction tool. Manual loan processing is time-consuming, prone to errors, and often uses outdated scoring models that don't consider modern financial behaviors. The goal is to create a web-based tool that leverages machine learning to predict loan approval, offering a faster and more accurate decision-making process for both users and lenders. Researching the market revealed that existing solutions like loan calculators and credit scoring platforms lack personalization, scalability, and advanced data-driven insights, leaving room for a more sophisticated product.

Brainstorming ideas led to key features such as a user-friendly interface for easy input, a customizable machine learning model for lenders, and real-time predictions based on user data like income, employment, and credit score. Additional features considered include loan product comparison, personalized financial insights, and partnership with lenders to offer pre-approved loans. The backend of the application would use Node.js and Express.js, integrating Python-based machine learning models with scikit-learn and Pandas for data handling. Frontend development would focus on creating an intuitive user experience with dynamic content rendering using EJS, and cloud services would be employed for scalability.

After generating several concepts, including a basic prediction tool, an AI-powered loan marketplace, and a B2B platform for lenders, the final concept was refined based on feasibility. The chosen concept is a loan eligibility prediction tool that connects users to personalized loan offers, with a freemium model for individual users and a commission based partnership model with financial institutions. This concept balances technical feasibility, market demand, and profitability, providing a comprehensive solution for both individuals seeking loans and financial institutions looking to improve their 14 decision-making processes recommendations, subscription plans, incentives & discounts, and more, readers can enjoy the economic and environmental benefits of renting, all while indulging in the nostalgic pleasure of library books and supporting local libraries.

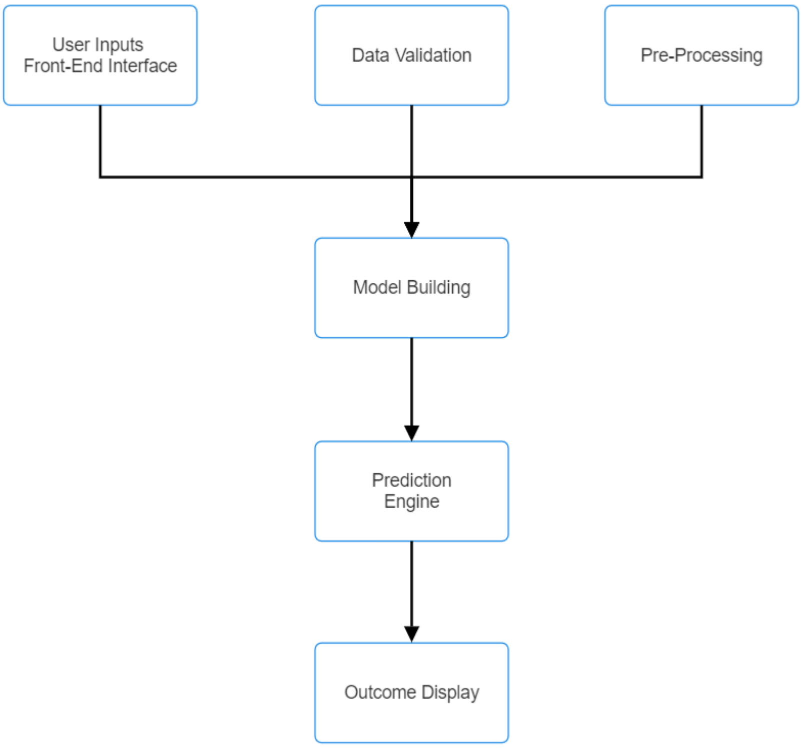
Concept Development

The Loan Prediction Web Application is a web-based tool designed to predict the likelihood of loan approval based on user-provided data such as income, employment status, credit score, and other relevant factors. The application will utilize a machine learning model trained on historical loan data to generate accurate predictions, helping both individuals and financial institutions make informed decisions regarding loan approvals. The primary goal of the application is to provide users with a fast, user friendly, and data-driven method to assess their loan eligibility, while also offering financial institutions a scalable, customizable tool for streamlining the loan decision process.

Key features of the application include a user-friendly interface for seamless data entry, real-time loan prediction results, and the ability to offer personalized financial insights to help users improve their chances of loan approval. The backend will be powered by Node.js and Express.js, while the prediction model will be integrated using Python, Pandas, and scikit-learn, serialized through Joblib for efficient performance. The frontend will utilize EJS for dynamic rendering, ensuring an intuitive experience for users. Scalability and performance will be ensured through the use of cloud services, making the application capable of handling a large number of users simultaneously.

Additionally, the web application will incorporate a business model that allows for monetization through a freemium subscription plan for individual users and a commission-based model for partnering with financial institutions. Lenders will also have the option to integrate custom loan prediction models into the platform, creating a valuable tool for the broader financial ecosystem. The Loan Prediction Web Application is designed to provide a comprehensive, scalable solution to both users and financial institutions, transforming the loan approval process into a more efficient, data-driven experience.

Schematic Diagram:



Product Details:

Data Soruces:

Kaggle:https://www.kaggle.com/datasets/altruistdelhite04/loan-prediction-problem-dataset

Algorithms, Frameworks, Software

1. Frontend:
   - HTML/CSS: For structuring and styling the user interface.
   - JavaScript: For interactivity and dynamic content handling.
   - EJS: Embedded JavaScript templating for rendering HTML.
2. Backend:
   - Node.js: For building the server-side application.
   - Express.js: A web application framework for Node.js, facilitating routing and middleware functions.
3. Machine Learning:
   - Python: Primary language for data analysis and model development.
   - Pandas: For data manipulation and analysis.
   - Scikit-learn: For building and training machine learning models.
   - Joblib: For saving and loading the trained model.

Team required to develop:

1. **Project Manager**
   - **Responsibilities**: Oversee the entire project lifecycle, manage timelines, coordinate communication among team members, and ensure the project stays within budget.
   - **Skills**: Strong leadership, organizational, and communication skills; experience in managing tech projects; knowledge of Agile methodologies.
2. **Frontend Developer**
   - **Responsibilities:** Design and implement the user interface, ensuring a responsive and user-friendly experience. This includes creating forms for user input and displaying prediction results.
   - **Skills:** Proficiency in HTML, CSS, JavaScript, and frameworks like React or Vue.js; experience with EJS for server-side rendering; understanding of UI/UX principles.
3. **Backend Developer**
   - **Responsibilities:** Develop the server-side logic, manage database interactions, handle API requests, and implement data validation processes.
   - **Skills:** Expertise in Node.js and Express.js; experience with databases like MongoDB or PostgreSQL; familiarity with RESTful API design.
4. **Data Scientist/Machine Learning Engineer**
   - **Responsibilities**: Analyze historical loan data, develop and train the machine learning model, perform data preprocessing, and ensure model accuracy and performance.
   - **Skills**: Strong knowledge of Python, Pandas, and Scikit-learn; experience in model deployment and evaluation; understanding of statistical analysis and data visualization.
5. **UI/UX Designer**
   - **Responsibilities**: Create intuitive and visually appealing designs for the application, conduct user research, and ensure that the user experience is seamless.

- **Skills**: Proficiency in design tools like Figma, Sketch, or Adobe XD; knowledge of user-centered design principles; experience in conducting usability testing.

6. **Quality Assurance (QA) Engineer**
   - **Responsibilities**: Test the application for bugs, usability issues, and overall functionality. Develop test cases and automate testing processes when possible.
   - **Skills**: Experience in manual and automated testing; knowledge of testing frameworks (e.g., Selenium, JUnit); attention to detail and problem-solving skills.
7. **DevOps Engineer (Optional)**
   - **Responsibilities**: Manage the deployment pipeline, ensure application scalability and reliability, and monitor performance in production environments.
   - **Skills**: Familiarity with cloud services (e.g., AWS, Azure), CI/CD tools, containerization technologies (e.g., Docker), and infrastructure as code (e.g., Terraform).

By bringing together these skills, the team can develop a comprehensive loan prediction web application analysis that can help the business improve customer requriments.

Step 2: Prototype Development

Code Implementation:

Import packages

[3]:
```python
# Dataframe manipulation
import pandas as pd

# Linear algebra
import numpy as np

# Data visualization with plotnine
from plotnine import *
import plotnine

# Data visualization with matplotlib
import matplotlib.pyplot as plt

# Data partitioning
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

# Grid-search
from sklearn.model_selection import GridSearchCV

# Evaluation metrics
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from sklearn.metrics import make_scorer

# XGBoost model
import xgboost as xgb

# Save the model
import joblib
```

[5]:
```python
# Ignore warnings
import warnings
warnings.filterwarnings('ignore', category = FutureWarning)
```

Import data set

After importing the data set into Python, the `df_train` is now our data frame. The data frame has a lot of functions and methods that will create spesific outputs about the characteristic of data frame. The method of `columns` will print out all the column names.

Training set

*[181]:
```python
# Import the training set
df_train = pd.read_csv(
    filepath_or_buffer = 'C:/Users/Ananth/Downloads/loan_train.csv',
    usecols = [i for i in range(1, 14)]
)
```

[12]:
```python
# Data dimension
print('Data dimension: {} rows and {} columns'.format(len(df_train), len(df_train.columns)))
df_train.head()
```

```
Data dimension: 491 rows and 13 columns
```

[12]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property
0	LP002305	Female	No	0	Graduate	No	4547	0.0	115.0	360.0	1.0	Semi
1	LP001715	Male	Yes	3+	Not Graduate	Yes	5703	0.0	130.0	360.0	1.0	
2	LP002086	Female	Yes	0	Graduate	No	4333	2451.0	110.0	360.0	1.0	
3	LP001136	Male	Yes	0	Not Graduate	Yes	4695	0.0	96.0	NaN	1.0	
4	LP002529	Male	Yes	2	Graduate	No	6700	1750.0	230.0	300.0	1.0	Semi

Testing data

[175]:
```python
# Import the testing set
df_test = pd.read_csv(
    filepath_or_buffer = "C:/Users/Ananth/Downloads/loan_test.csv"
)
```

[17]:
```python
# Data dimension
print('Data dimension: {} rows and {} columns'.format(len(df_test), len(df_test.columns)))
df_test.head()
```

```
Data dimension: 123 rows and 12 columns
```

[17]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property
0	LP001116	Male	No	0	Not Graduate	No	3748	1668.0	110.0	360.0	1.0	Semi
1	LP001488	Male	Yes	3+	Graduate	No	4000	7750.0	290.0	360.0	1.0	Semi
2	LP002138	Male	Yes	0	Graduate	No	2625	6250.0	187.0	360.0	1.0	
3	LP002284	Male	No	0	Not Graduate	No	3902	1666.0	109.0	360.0	1.0	
4	LP002328	Male	Yes	0	Not Graduate	No	6096	0.0	218.0	360.0	0.0	

Data preprocessing

Training data

Scale measurement

The method of `info` will show us the metadata or information about the columns in a data frame. It undirectly specifies the scale measurement of a given columns in a data frame. However, it can be misleading. So, we must modify the scale measurement or column types based on column characteristic.

[23]:
```python
# Data frame metadata
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 491 entries, 0 to 490
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            491 non-null    object 
 1   Gender             481 non-null    object 
 2   Married            490 non-null    object 
 3   Dependents         482 non-null    object 
 4   Education          491 non-null    object 
 5   Self_Employed      462 non-null    object 
 6   ApplicantIncome    491 non-null    int64  
 7   CoapplicantIncome  491 non-null    float64
 8   LoanAmount         475 non-null    float64
 9   Loan_Amount_Term   478 non-null    float64
 10  Credit_History     448 non-null    float64
 11  Property_Area      491 non-null    object 
 12  Loan_Status        491 non-null    int64  
dtypes: float64(4), int64(2), object(7)
memory usage: 50.0+ KB
```

[25]:
```python
# Change column types
df_train = df_train.astype({'Credit_History': object, 'Loan_Status': int})
df_train.select_dtypes(include = ['object']).dtypes
```

[25]:
```
Loan_ID           object
Gender            object
Married           object
Dependents        object
Education         object
Self_Employed     object
Credit_History    object
Property_Area     object
dtype: object
```

```
[27]: # Summary statistics of categorical columns
for i in df_train.select_dtypes('object').columns:
    print(df_train[i].value_counts(),'\n')
```

```
Loan_ID
LP002305    1
LP001806    1
LP002543    1
LP001669    1
LP002272    1
           ..
LP001138    1
LP002813    1
LP001213    1
LP002738    1
LP002777    1
Name: count, Length: 491, dtype: int64

Gender
Male      393
Female     88
Name: count, dtype: int64
```

Handle missing values ¶

```
[30]: # Check missing values
df_train.isna().sum()
```

```
[30]: Loan_ID               0
Gender               10
Married               1
Dependents            9
Education             0
Self_Employed        29
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           16
Loan_Amount_Term     13
Credit_History       43
Property_Area         0
Loan_Status           0
dtype: int64
```

Note: Consideration to remove missing values is based on a business logic. The concept of *garbage in garbage out* applies. Without any relevant domain knowledges of loan problem, the interpolation will lead to the biased result.

Instead of dropping the missing values brutally, we try to inspect the relevant variables in the data in order to suggest the consideration for the next analysis

`Dependents` ¶

```
[34]: print('Number of missing dependents is about {} rows'.format(df_train['Dependents'].isna().sum()))
```

```
Number of missing dependents is about 9 rows
```

```
[36]: # Replace missing valuess with "0"
df_train['Dependents'].fillna(value = '0', inplace = True)
```

`Self_Employed`

```
[39]: print('Number of missing Self_Employed is about {} rows'.format(df_train['Self_Employed'].isna().sum()))
```

```
Number of missing Self_Employed is about 29 rows
```

```
[41]: # Replace missing values with "No"
df_train['Self_Employed'].fillna(value = 'No', inplace = True)
```

`Loan_Amount_Term`

```
[44]: df_train[['Loan_Amount_Term', 'Loan_Status']].groupby('Loan_Status').describe()
```

[44]:

	Loan_Amount_Term							
	count	mean	std	min	25%	50%	75%	max
Loan_Status								
0	143.0	341.790210	73.018891	36.0	360.0	360.0	360.0	480.0
1	335.0	341.086567	64.320411	12.0	360.0	360.0	360.0	480.0

[46]:
```
print('Percentile 20th: {}'.format(df_train['Loan_Amount_Term'].quantile(q = 0.2)))
```

```
Percentile 20th: 360.0
```

[48]:
```
# Replace missing values with "360"
df_train['Loan_Amount_Term'].fillna(value = 360, inplace = True)
```

`Credit_History`

[51]:
```
# Cross tabulation of credit history and loan status
df_cred_hist = pd.crosstab(df_train['Credit_History'], df_train['Loan_Status'], margins = True).reset_index()
# Remove index name
df_cred_hist.columns.name = None
# Remove last row for total column attribute
df_cred_hist = df_cred_hist.drop([len(df_cred_hist) - 1], axis = 0)
df_cred_hist.rename(columns = {'Credit_History':'Credit History', 0:'No', 1:'Yes'}, inplace = True)
df_cred_hist
```

[51]:

	Credit History	No	Yes	All
0	0.0	62	6	68
1	1.0	74	306	380

[53]:
```
# Slice the data frame based on loan status
pos_cred_hist0 = df_train[(df_train['Credit_History'].isna()) & (df_train['Loan_Status'] == 0)]
pos_cred_hist1 = df_train[(df_train['Credit_History'].isna()) & (df_train['Loan_Status'] == 1)]
print('Number of rows with Loan_Status is No but Credit_History is NaN  : {}'.format(len(pos_cred_hist0)))
print('Number of rows with Loan_Status is Yes but Credit_History is NaN : {}'.format(len(pos_cred_hist1)))
```

```
Number of rows with Loan_Status is No but Credit_History is NaN  : 12
Number of rows with Loan_Status is Yes but Credit_History is NaN : 31
```

[55]:
```
# Replace the missing values with a specific condition
credit_loan = zip(df_train['Credit_History'], df_train['Loan_Status'])
df_train['Credit_History'] = [
                                0.0 if np.isnan(credit) and status == 0 else
                                1.0 if np.isnan(credit) and status == 1 else
                                credit for credit, status in credit_loan
                             ]
```

`Gender` and `Loan Amount`

[58]:
```
# Drop missing values
df_train.dropna(axis = 0, how = 'any', inplace = True)
```

[60]:
```
# Check missing value
df_train.isna().sum()
```

[60]:
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

Testing data

Scale measurement

[64]:
```
# Data frame metadata
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 123 entries, 0 to 122
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            123 non-null    object
 1   Gender             120 non-null    object
 2   Married            121 non-null    object
 3   Dependents         117 non-null    object
 4   Education          123 non-null    object
 5   Self_Employed      120 non-null    object
 6   ApplicantIncome    123 non-null    int64
 7   CoapplicantIncome  123 non-null    float64
 8   LoanAmount         117 non-null    float64
 9   Loan_Amount_Term   122 non-null    float64
 10  Credit_History     116 non-null    float64
 11  Property_Area      123 non-null    object
dtypes: float64(4), int64(1), object(7)
memory usage: 11.7+ KB
```

[66]:
```python
# Change column types
df_test = df_test.astype({'Credit_History': object})
df_test.select_dtypes(include = ['object']).dtypes
```

[66]:
```
Loan_ID           object
Gender            object
Married           object
Dependents        object
Education         object
Self_Employed     object
Credit_History    object
Property_Area     object
dtype: object
```

[68]:
```python
# Summary statistics of categorical columns
for i in df_test.select_dtypes('object').columns:
    print(df_test[i].value_counts(),'\n')
```

```
Loan_ID
LP001116    1
LP002262    1
LP001047    1
LP001844    1
LP001938    1
           ..
LP001917    1
LP001940    1
LP001316    1
LP001266    1
LP001616    1
Name: count, Length: 123, dtype: int64

Gender
Male      96
Female    24
Name: count, dtype: int64
```

Handle missing values

[71]:
```python
# Check missing values
df_test.isna().sum()
```

[71]:
```
Loan_ID              0
Gender               3
Married              2
Dependents           6
Education            0
Self_Employed        3
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           6
Loan_Amount_Term     1
Credit_History       7
Property_Area        0
dtype: int64
```

`Dependents`

[74]:
```python
print('Number of missing values in Dependents is about {} rows'.format(df_test['Dependents'].isna().sum()))
```

```
Number of missing values in Dependents is about 6 rows
```

[76]:
```python
# Replace missing values with "0"
df_test['Dependents'].fillna(value = '0', inplace = True)
```

`Self_Employed`

[79]:
```python
print('Number of missing values in Self_Employed is about {} rows'.format(df_test['Self_Employed'].isna().sum()))
```

```
Number of missing values in Self_Employed is about 3 rows
```

[81]:
```python
# Replace missing values with "No"
df_test['Self_Employed'].fillna(value = 'No', inplace = True)
```

`Loan_Amount_Term`

[84]:
```python
# Replace missing values with "360"
df_test['Loan_Amount_Term'].fillna(value = 360, inplace = True)
```

`Gender`, `Married`, `LoanAmount` and `Credit_History`

[87]:
```python
# Drop missing values
df_test.dropna(axis = 0, how = 'any', inplace = True)
```

[89]:
```python
# Check missing values
df_test.isna().sum()
```

[89]:
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
dtype: int64
```

Exploratory Data Analysis

The composition of default and not default customers

[93]:
```python
# Data aggregation between default and not default customers
df_viz_1 = df_train.groupby(['Loan_Status'])['Loan_ID'].count().reset_index(name = 'Total')
# Map the loan status
df_viz_1['Loan_Status'] = df_viz_1['Loan_Status'].map(
    {
        0: 'Not default',
        1: 'Default'
    }
)
```

[95]:
```python
# Show the data
df_viz_1
```

[95]:

	Loan_Status	Total
0	Not default	134
1	Default	330

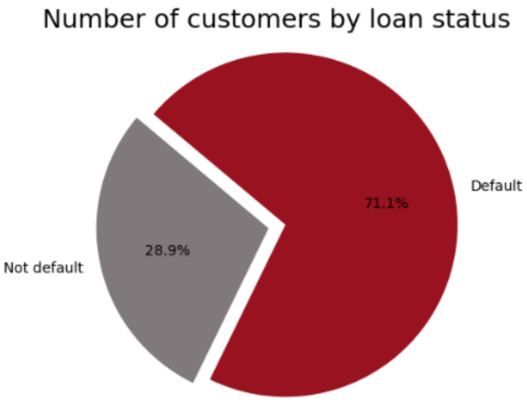
[97]:
```python
# Figure size
plt.figure(figsize = (6.4,4.8))

# Customize colors and other settings
colors = ['#80797c','#981220']

# Explode 1st slice
explode = (0.1, 0)

# Create a pie chart
plt.pie(
    x = 'Total',
    labels = 'Loan_Status',
    data = df_viz_1,
    explode = explode,
    colors = colors,
    autopct = '%1.1f%%',
    shadow = False,
    startangle = 140
)

# Title and axis
plt.title('Number of customers by loan status', fontsize = 18)
plt.axis('equal')
plt.show()
```



The composition of loan status by the dependents

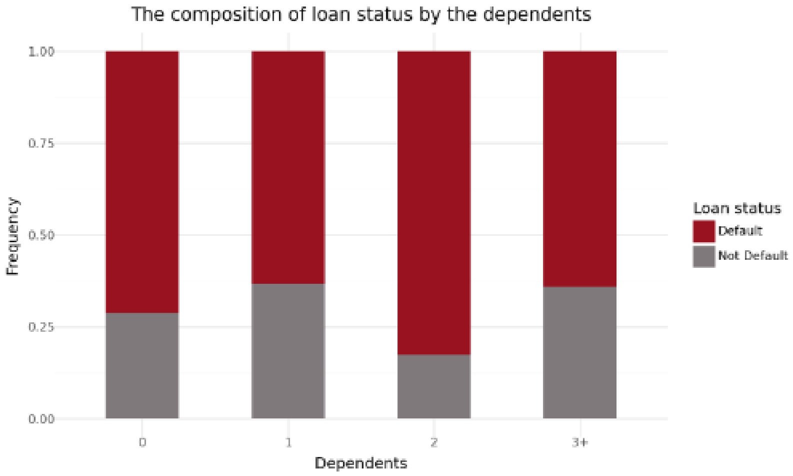
[100]:
```python
# Data aggregation between loan status and dependents
df_viz_2 = df_train.groupby(['Loan_Status', 'Dependents'])['Loan_ID'].count().reset_index(name = 'Total')
# Map the loan status
df_viz_2['Loan_Status'] = df_viz_2['Loan_Status'].map(
    {
        0: 'Not default',
        1: 'Default'
    }
)
```

[102]:
```python
# Show the data
df_viz_2
```

[102]:

	Loan_Status	Dependents	Total
0	Not default	0	77
1	Not default	1	30
2	Not default	2	13
3	Not default	3+	14
4	Default	0	191
5	Default	1	52
6	Default	2	62
7	Default	3+	25

```
[104]: plotnine.options.figure_size = (8, 4.8)
(
    ggplot(
        data = df_viz_2
    )+
    geom_bar(
        aes(
            x = 'Dependents',
            y = 'Total',
            fill = 'Loan_Status'
        ),
        stat = 'identity',
        position = 'fill',
        width = 0.5
    )+
    labs(
        title = 'The composition of loan status by the dependents',
        fill = 'Loan status'
    )+
    xlab(
        'Dependents'
    )+
    ylab(
        'Frequency'
    )+
    scale_x_discrete(
        limits = ['0', '1', '2', '3+']
    )+
    scale_fill_manual(
        values = ['#981220','#80797c'],
        labels = ['Default', 'Not Default']
    )+
    theme_minimal()
)
```



The composition of default customer by the educations

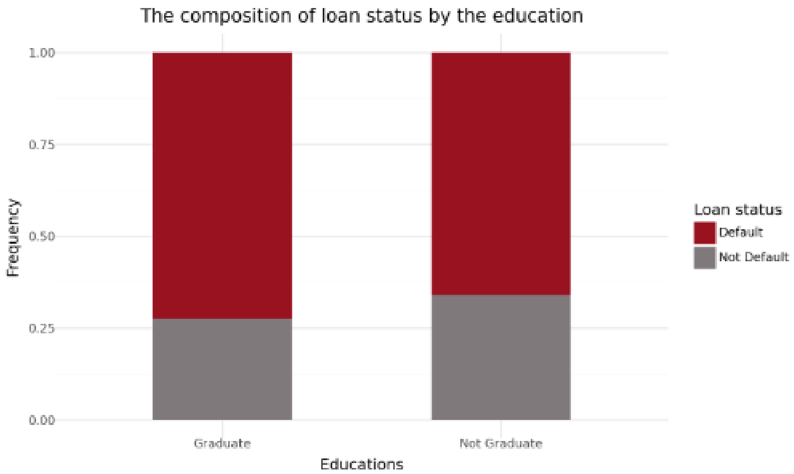
```
[107]: # Data aggregation between loan status and dependents
df_viz_3 = df_train.groupby(['Loan_Status', 'Education'])['Loan_ID'].count().reset_index(name = 'Total')
# Map the loan status
df_viz_3['Loan_Status'] = df_viz_3['Loan_Status'].map(
    {
        0: 'Not default',
        1: 'Default'
    }
)
```

```
[109]: # Show the data
df_viz_3
```

[109]:

	Loan_Status	Education	Total
0	Not default	Graduate	101
1	Not default	Not Graduate	33
2	Default	Graduate	266
3	Default	Not Graduate	64

[111]:
```
plotnine.options.figure_size = (8, 4.8)
(
    ggplot(
        data = df_viz_3
    )+
    geom_bar(
        aes(
            x = 'Education',
            y = 'Total',
            fill = 'Loan_Status'
        ),
        stat = 'identity',
        position = 'fill',
        width = 0.5
    )+
    labs(
        title = 'The composition of loan status by the education',
        fill = 'Loan status'
    )+
    xlab(
        'Educations'
    )+
    ylab(
        'Frequency'
    )+
    scale_x_discrete(
        limits = ['Graduate', 'Not Graduate']
    )+
    scale_fill_manual(
        values = ['#981220','#80797c'],
        labels = ['Default', 'Not Default']
    )+
    theme_minimal()
)
```



The distribution of applicant incomes by loan status

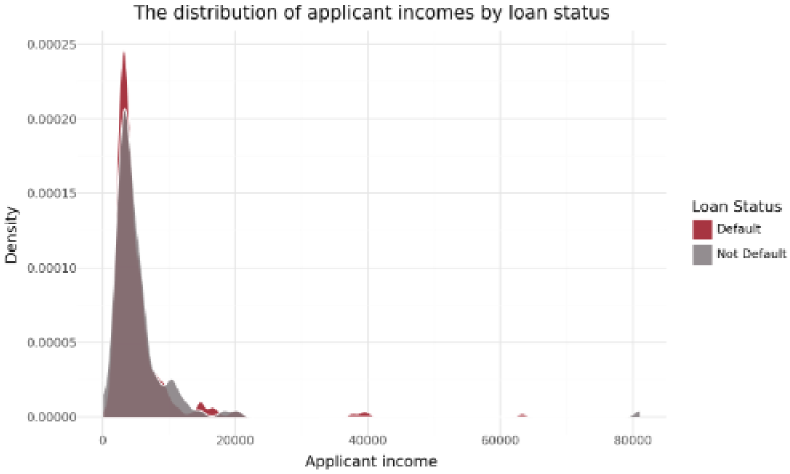
[114]:
```
# Slice the columns
df_viz_4 = df_train[['ApplicantIncome', 'Loan_Status']].reset_index(drop = True)
# Map the loan status
df_viz_4['Loan_Status'] = df_viz_4['Loan_Status'].map(
    {
        0: 'Not default',
        1: 'Default'
    }
)
```

[116]:
```
# Show the data
df_viz_4.head()
```

[116]:

	ApplicantIncome	Loan_Status
0	4547	Default
1	5703	Default
2	4333	Not default
3	4695	Default
4	6700	Default

[118]:
```python
plotnine.options.figure_size = (8, 4.8)
(
    ggplot(
        data = df_viz_4
    )+
    geom_density(
        aes(
            x = 'ApplicantIncome',
            fill = 'Loan_Status'
        ),
        color = 'white',
        alpha = 0.85
    )+
    labs(
        title = 'The distribution of applicant incomes by loan status'
    )+
    scale_fill_manual(
        name = 'Loan Status',
        values = ['#981220','#80797c'],
        labels = ['Default', 'Not Default']
    )+
    xlab(
        'Applicant income'
    )+
    ylab(
        'Density'
    )+
    theme_minimal()
)
```



The distribution of loan amount by loan status

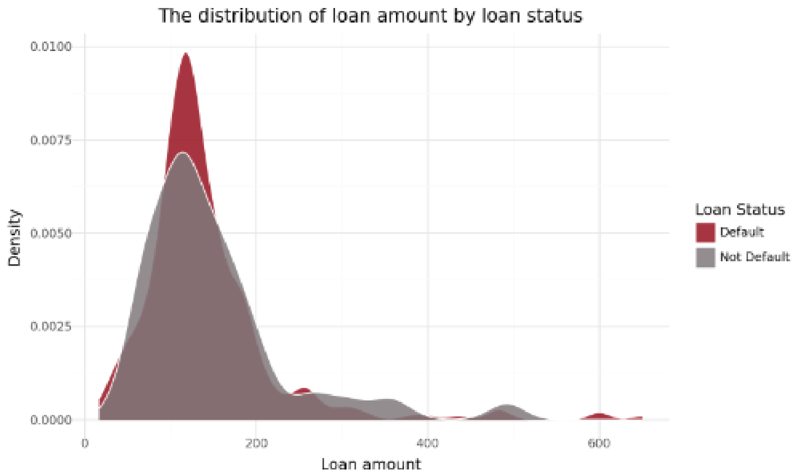
[120]:
```python
# Slice the columns
df_viz_5 = df_train[['LoanAmount', 'Loan_Status']].reset_index(drop = True)
# Map the loan status
df_viz_5['Loan_Status'] = df_viz_5['Loan_Status'].map(
    {
        0: 'Not default',
        1: 'Default'
    }
)
```

[121]:
```python
# Show the data
df_viz_5.head()
```

[121]:

	LoanAmount	Loan_Status
0	115.0	Default
1	130.0	Default
2	110.0	Not default
3	96.0	Default
4	230.0	Default

[122]:
```
plotnine.options.figure_size = (8, 4.8)
(
    ggplot(
        data = df_viz_5
    )+
    geom_density(
        aes(
            x = 'LoanAmount',
            fill = 'Loan_Status'
        ),
        color = 'white',
        alpha = 0.85
    )+
    labs(
        title = 'The distribution of loan amount by loan status'
    )+
    scale_fill_manual(
        name = 'Loan Status',
        values = ['#981220','#80797c'],
        labels = ['Default', 'Not Default']
    )+
    xlab(
        'Loan amount'
    )+
    ylab(
        'Density'
    )+
    theme_minimal()
)
```



One-hot encoder

[124]:
```
# Add new column of Loan_Status with 999 in testing data
df_test['Loan_Status'] = 999
# Concat the training and testing data
df_concat = pd.concat(objs = [df_train , df_test], axis = 0)
```

[128]:
```
# Drop the column of Loan_ID
df_concat.drop(columns = ['Loan_ID'], inplace = True)
```

[129]:
```
# Categorical columns
cols_obj_train = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Credit_History', 'Property_Area']
print(cols_obj_train)
```

```
['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Credit_History', 'Property_Area']
```

[132]:
```
# One-hot encoding
df_concat = pd.get_dummies(data = df_concat, columns = cols_obj_train, drop_first = True)
print('Dimension data: {} rows and {} columns'.format(len(df_concat), len(df_concat.columns)))
df_concat.head()
```

```
Dimension data: 570 rows and 15 columns
```

[132]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Loan_Status	Gender_Male	Married_Yes	Dependents_1	Dependents_2	Dependents_3+	Educ
0	4547	0.0	115.0	360.0	1	False	False	False	False	False	
1	5703	0.0	130.0	360.0	1	True	True	False	False	True	
2	4333	2451.0	110.0	360.0	0	False	True	False	False	False	
3	4695	0.0	96.0	360.0	1	True	True	False	False	False	
4	6700	1750.0	230.0	300.0	1	True	True	False	True	False	

Data partitioning

[137]:
```
# Unique values of Loan_Status
df_concat['Loan_Status'].value_counts()
```

[137]:
```
Loan_Status
1      330
0      134
999    106
Name: count, dtype: int64
```

[139]:
```
# Training set
df_train = df_concat[df_concat['Loan_Status'].isin([0, 1])].reset_index(drop = True)
print('Dimension data: {} rows and {} columns'.format(len(df_train), len(df_train.columns)))
df_train.head()
```

```
Dimension data: 464 rows and 15 columns
```

[139]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Loan_Status	Gender_Male	Married_Yes	Dependents_1	Dependents_2	Dependents_3+	Educ
0	4547	0.0	115.0	360.0	1	False	False	False	False	False	
1	5703	0.0	130.0	360.0	1	True	True	False	False	True	
2	4333	2451.0	110.0	360.0	0	False	True	False	False	False	
3	4695	0.0	96.0	360.0	1	True	True	False	False	False	
4	6700	1750.0	230.0	300.0	1	True	True	False	True	False	

[141]:
```
# Testing set
df_test = df_concat[df_concat['Loan_Status'].isin([999])].reset_index(drop = True)
print('Data dimension: {} rows and {} columns'.format(len(df_test), len(df_test.columns)))
df_test.head()
```

```
Data dimension: 106 rows and 15 columns
```

[141]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Loan_Status	Gender_Male	Married_Yes	Dependents_1	Dependents_2	Dependents_3+	Educ
0	3748	1668.0	110.0	360.0	999	True	False	False	False	False	
1	4000	7750.0	290.0	360.0	999	True	True	False	False	True	
2	2625	6250.0	187.0	360.0	999	True	True	False	False	False	
3	3902	1666.0	109.0	360.0	999	True	False	False	False	False	
4	6096	0.0	218.0	360.0	999	True	True	False	False	False	

[143]:
```
# Data partitioning >>> training set into training and validation
df_train_final = df_train.reset_index(drop = True)
X = df_train_final[df_train_final.columns[~df_train_final.columns.isin(['Loan_Status'])]]
y = df_train_final['Loan_Status']

# Training = 70% and validation = 30%
X_train, X_val, y_train, y_val = train_test_split(X , y, test_size = 0.3, random_state = 42)
print('Data dimension of training set   :', X_train.shape)
print('Data dimension of validation set :', X_val.shape)

# Testing set
X_test = df_test[df_test.columns[~df_test.columns.isin(['Loan_Status'])]]
print('Data dimension of testing set    :', X_test.shape)
```

```
Data dimension of training set   : (324, 14)
Data dimension of validation set : (140, 14)
Data dimension of testing set    : (106, 14)
```

Machine learning model development

[146]:
```
# XGBoost model
xgb_model = xgb.XGBClassifier(
    objective = 'binary:logistic',
    use_label_encoder = False
)
```

[148]:
```
# Define parameter range
params = {
    'eta': np.arange(0.1, 0.26, 0.05),
    'min_child_weight': np.arange(1, 5, 0.5).tolist(),
    'gamma': [5],
    'subsample': np.arange(0.5, 1.0, 0.11).tolist(),
    'colsample_bytree': np.arange(0.5, 1.0, 0.11).tolist()
}
```

[150]:
```
# Make a scorer from a performance metric or loss function
scorers = {
    'f1_score': make_scorer(f1_score),
    'precision_score': make_scorer(precision_score),
    'recall_score': make_scorer(recall_score),
    'accuracy_score': make_scorer(accuracy_score)
}
```

[152]:
```
# k-fold cross validation
skf = KFold(n_splits = 10, shuffle = True)
```

[154]:
```python
# Set up the grid search CV
grid = GridSearchCV(
    estimator = xgb_model,
    param_grid = params,
    scoring = scorers,
    n_jobs = -1,
    cv = skf.split(X_train, np.array(y_train)),
    refit = 'accuracy_score'
)
```

[156]:
```python
# Fit the model
grid.fit(X = X_train, y = y_train)
```

[156]:
```
▸ GridSearchCV
▸ estimator: XGBClassifier
  ▸ XGBClassifier
```

[157]:
```python
# Best parameters
grid.best_params_
```

[157]:
```
{'colsample_bytree': 0.83,
 'eta': 0.25000000000000006,
 'gamma': 5,
 'min_child_weight': 1.5,
 'subsample': 0.61}
```

[158]:
```python
# Create a prediction of training
predicted = grid.predict(X_val)
```

[159]:
```python
# Model evaluation - training data
accuracy_baseline = accuracy_score(predicted, np.array(y_val))
recall_baseline = recall_score(predicted, np.array(y_val))
precision_baseline = precision_score(predicted, np.array(y_val))
f1_baseline = f1_score(predicted, np.array(y_val))

print('Accuracy for baseline   :{}'.format(round(accuracy_baseline, 5)))
print('Recall for baseline     :{}'.format(round(recall_baseline, 5)))
print('Precision for baseline  :{}'.format(round(precision_baseline, 5)))
print('F1 Score for baseline   :{}'.format(round(f1_baseline, 5)))
```

```
Accuracy for baseline   :0.87143
Recall for baseline     :0.84615
Precision for baseline  :1.0
F1 Score for baseline   :0.91667
```

Store the ML model

[173]:
```python
# Store the model into a pickle file
filename = 'xgboostModel.pkl'
joblib.dump(grid.best_estimator_, filename)
```

[173]:
```
['xgboostModel.pkl']
```

Conclusion:

This Web Application is a transformative tool designed to streamline the loan approval process by leveraging machine learning algorithms and providing users with real-time insights into their loan eligibility. By integrating a user-friendly interface with robust backend functionality, the application empowers individuals to easily input their financial data and receive immediate predictions based on historical loan patterns. The collaborative efforts of a diverse development team ensure the application is scalable, adaptable, and accessible to a broad audience. With an estimated cost that reflects both feasibility and potential return on investment, the project is well-positioned to meet the growing demand for digital financial solutions. As it approaches deployment, ongoing user feedback and iterative enhancements will be essential to its success in an increasingly competitive market.

Step-3 : Business Modelling

A Loan Prediction Web Application serves as both a customer-facing tool and a backend efficiency enhancer for financial institutions. Below is a detailed business model:

1. Problem Identification
   - Customers: Face uncertainty about loan eligibility, leading to delays and frustration.
   - Financial Institutions: Struggle with processing numerous applications, increasing operational costs and potential errors.

2. Value Proposition
   - For Customers: Instant loan eligibility predictions based on minimal input, reducing uncertainty and enhancing user experience.
   - For Financial Institutions: Streamlined and automated decision-making to reduce processing time, improve accuracy, and lower costs.

3. Target Customers
   - Primary Users: Individuals seeking personal, home, or business loans.
   - Secondary Users: Financial institutions, banks, and NBFCs aiming to optimize loan processing.

4. Revenue Streams
   - Direct Revenue:
     - Subscription-based pricing for institutions using the platform.
     - Pay-per-use model for institutions with lower application volumes.

   - Indirect Revenue:
     - Lead generation for partner banks and lenders.
     - Data monetization (aggregated, anonymized insights).

5. Key Resources
   - Technical Resources: A robust machine learning model trained on diverse loan datasets, cloud hosting for scalability, and an intuitive user interface.
   - Human Resources: Data scientists, developers, and financial domain experts.
   - Partnerships: Collaborations with banks and financial institutions for data access and application integration.

6. Channels
   - Digital Platform: The web application itself, accessible via browsers.
   - Mobile Integration: A mobile app for broader accessibility.
   - API Services: Allowing institutions to integrate prediction models into their systems.

7. Cost Structure
    - Development Costs: Building the machine learning model and web platform.
    - Maintenance Costs: Server hosting, regular updates, and model retraining.
    - Marketing Costs: Digital marketing campaigns to attract users and institutions.

8. Key Metrics
    - For Customers: Time saved, accuracy of predictions, and satisfaction levels.
    - For Institutions: Reduced application processing time, cost savings, and increase in approved loans.

9. Scalability and Expansion
    - Expand to offer credit score checks, financial literacy tools, and multi-language support.
    - Broaden partnerships with more banks and financial services providers.
    - Explore global markets, adapting to regional financial regulations.

This business model ensures that the Loan Prediction Web Application meets the needs of both customers and financial institutions while maintaining profitability and scalability.

Step-4 : Financial Modelling (Equation) with Machine Learning & Data Analysis:

1. PROFIT EARNED BY THE MODEL:

Market profit = (Number of Customers * Average Revenue per Customer * Subscription Duration) - (Acquisition Cost * Number of New Customers + expenditure cost)

Number of Customers is the total number of customers in each period.

Average Revenue per Customer is the average revenue generated per customer in a given period.

Subscription duration is the duration for which customer has taken the subscription. Acquisition Cost is the cost of acquiring a new customer.

Number of New Customers is the number of new customers acquired during the given period.

Expenditure cost is the cost spent in miscellaneous like salary, maintenance etc. Now, let us assume that number of customers be 1000, average revenue per customer be Rs. 10000 per month, acquisition cost be 4000 with new customers be 300 and expenditure cost be 3000 and let T be 1 year.

Market profit = (1000*10000* T) – (4000*300 + 3000)

Market profit= (10000000*T)- 1203000

Market Profit= (10000000*1)-1203000

Market profit= 8,797,000.

2. PROFIT EARNED BY INDUSTRY USING MODEL:

The profit earned by the industry using a loan prediction model can be estimated based on its operational impact and cost-efficiency. The model directly contributes to revenue generation, cost savings, and improved customer retention. Below is the detailed calculation:

Formula for Industry Profit:

Profit Earned by Industry = Revenue Generated from Loans Approved - Loan Defaults + Cost Savings from Efficiency Improvements - Model Development and Maintenance Cost

Breakdown of Components:

1. **Revenue Generated from Loans Approved:**
   - Loan Approval Rate: Increased approvals due to accurate predictions.
   - Loan Amount: Average loan amount approved per customer.
   - Interest Revenue: Revenue generated from interest rates applied to loans.

   ### Formula:

   Revenue from Loans = Number of Approved Loans x Average Loan Amount x Average Interest Rate

2. **Loan Defaults Reduced:**
   - Default Rate Reduction: The model reduces risky approvals, lowering default rates.

   ### Formula:

   Saved Losses from Reduced Defaults = Baseline Default Rate - Model Default Rate x Total Loan Amount

3. **Cost Savings from Efficiency Improvements:**
   - Operational Cost Reduction: Reduced time and resources spent on manual evaluations.
   - Staffing Efficiency: Lower need for manual underwriters or reviewers.

   Formula:

   Cost Savings = Baseline Cost per Application - Model Cost per Application x Number of Applications Processed

4. **Model Development and Maintenance Costs:**
   - Includes costs for:
     - Initial development.
     - Training and retraining the model.
     - Hosting and infrastructure costs.
     - Ongoing maintenance.

Industry Profit Estimation:

By combining these components, the total profit for the industry can be estimated as:

Profit Earned by Industry = Revenue from Loans - Losses from Defaults + Cost Savings - Model Costs

Final Profit:

Profit = Revenue from Loans + Saved Losses from Defaults + Cost Savings) - Model Costs

This approach illustrates how the loan prediction model significantly enhances profitability by boosting loan revenues, reducing defaults, and cutting operational costs.