LOAN APPROVAL ANALYSIS

ABSTRACT

ABSTRACT

Loan approval is an important process for banks and financial institutions, requiring careful assessment of borrowers to reduce risks. This study analyzes borrower profiles and builds a machine learning model to predict loan approvals. The dataset includes key details such as credit history, income, loan amount, employment status, and repayment behavior**.** Two machine learning models Decision Tree and Random Forest were tested, and Random Forest gave the best results. The analysis showed that credit history, income, and loan amount were the most important factors in loan approval. This study highlights how machine learning can automate loan decisions, reduce manual work, and improve efficiency. However, its accuracy depends on data quality and does not consider external economic factors. Future improvements could use deep learning and additional borrower data, such as social credit scores, for better predictions. This research proves that machine learning can make loan approvals faster, fairer, and more reliable.

TABLE OF CONTENTS

CHAPTER-I
INTRODUCTION

1.1 Scope of analysis

- The scope of analysis for this dataset includes various features about individuals and their loan applications.
- Key variables include demographic information (e.g., age, gender, education), financial status (e.g., income, employment experience, home ownership), loan specifics (e.g., amount, intent, interest rate), credit-related metrics (e.g., credit score, credit history length), and the presence of previous loan defaults.
- Additionally, the dataset provides the **target variable loan_status**, which indicates whether a loan was approved or rejected, offering opportunities for predictive modeling, trend analysis, and risk assessment in lending practices.

1.2 Approach of analysis

- The analysis approach involves exploring the dataset to identify patterns and relationships between variables that influence loan approval (loan_status).
- Key steps include descriptive analysis to summarize demographic, financial, and credit-related factors, visualization to uncover trends, and statistical or machine learning techniques to model the impact of these factors on loan outcomes.
- This will help in predicting loan approval and understanding the risk factors associated with lending.

CHAPTER-II

GATHERING DATA

2.1 Dataset Description

This dataset contains **18,714 loan applications** from borrowers in Turkey. The amount denotes in Turkish Lira ('₺'). It includes demographic details, financial information, and loan-related attributes to analyze borrower profiles and predict loan approval decisions.

2.1.1 Load the relevant packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

- **Numpy** is used for numerical calculations in the dataset.
- **Pandas** is used for reading and filtering the dataset.
- **Matplotlib** is used for graphical visualization.
- **Seaborn** is used for advanced graphical visualization.

2.1.2 Load the Dataset

- The ***"pd.read_csv( )"*** function in pandas is used to read a CSV (Comma-Separated Values) file and load it into a DataFrame for analysis. It is one of the most commonly used methods for importing structured data into Python.

```
loan=pd.read_csv("C:/Users/lenovo/Downloads/loan_data.csv")
loan
```

- The ***".shape( )"*** attribute in pandas is used to get the dimensions of a DataFrame or a NumPy array. It returns a tuple representing the number of rows and columns in the dataset.

```
#shape
loan.shape
```

```
(18714, 13)
```

```
loan
```

	age	gender	education	income	experience	home_ownership	loan_amount	loan_intent	loan_int_rate	loan_percent_income	credit_score	pending_loan	loan_s
0	22	female	Master	71948	0	RENT	35000	PERSONAL	16.02	0.49	561	No	
1	23	female	Bachelor	79753	0	RENT	35000	MEDICAL	15.23	0.44	675	No	
2	24	male	Master	66135	1	RENT	35000	MEDICAL	14.27	0.53	586	No	
3	26	female	Bachelor	93471	1	RENT	35000	EDUCATION	12.42	0.37	701	No	
4	24	female	High School	95550	5	RENT	35000	MEDICAL	11.11	0.37	585	No	
...	...	...	...	...	...	...	...	...	...	...	...	...	
18709	31	male	Master	136832	9	RENT	12319	PERSONAL	16.92	0.09	722	No	
18710	24	male	High School	37786	0	MORTGAGE	13500	EDUCATION	13.43	0.36	612	No	
18711	24	female	Associate	31924	2	RENT	12229	MEDICAL	10.70	0.38	678	No	
18712	27	male	Associate	47971	6	RENT	15000	MEDICAL	15.66	0.31	645	No	
18713	29	male	Bachelor	33164	4	RENT	12000	EDUCATION	13.23	0.36	604	No	

The dataset contains 13 columns, which are described as follows:

age: This represents the age of the applicant in years. Age is a crucial factor in loan approval as younger applicants may have less credit history, while older applicants might have more financial stability.

gender: This indicates the applicant's gender, typically categorized as male or female.

education: This denotes the highest level of education attained by the applicant, such as Master's, Bachelor's, or High School. Higher education often correlates with higher income and lower default risk.

income: This refers to the applicant's annual earnings, in Turkish Lira ('₺'). A higher income increases the chances of loan approval, as it indicates better financial capacity to repay debts.

experience: This column represents the number of years the applicant has been employed. More work experience generally suggests job stability and a reliable income stream.

home_ownership: This indicates whether the applicant owns or rents their home. Homeowners may have a higher chance of loan approval, as owning property is seen as a sign of financial stability.

loan_amount: This represents the amount of money the applicant is requesting as a loan.

loan_intent: This describes the purpose of the loan, such as personal, medical, or education expenses.

loan_int_rate: This is the interest rate applied to the loan, typically expressed as a percentage. Interest rates depend on factors like credit score, loan amount, and the applicant's financial history.

loan_percent_income: This represents the percentage of the applicant's income that goes toward loan repayment.

credit_score: This numerical value reflects the applicant's creditworthiness, based on their financial history.

pending_loan: This indicates whether the applicant has any other outstanding loans at the time of application. Applicants with multiple existing loans may be considered high risk.

loan_status: This column represents the final loan decision, where **1** means the loan was approved and **0** means it was rejected.

- This dataset consists of 13 different columns. To find Statistical summary about the dataset we can use "***.describe( )***" method. This method helps to find statistical values like count,minimum,maximum,mean,standard deviation,q1,q2,q3 for all the continuous variable columns in the dataset.

```
#statistical summary
loan.describe()
```

	age	income	experience	loan_amount	loan_int_rate	loan_percent_income	credit_score	loan_status
count	18714.000000	1.871400e+04	18714.000000	18714.000000	18714.000000	18714.000000	18714.000000	18714.000000
mean	27.764775	9.695878e+04	5.374693	15422.484664	11.362330	0.193045	632.550711	0.260821
std	5.330405	6.663023e+04	5.393225	5388.855761	2.986452	0.087373	50.285091	0.439094
min	20.000000	2.136100e+04	0.000000	10000.000000	5.420000	0.010000	418.000000	0.000000
25%	24.000000	6.059850e+04	1.000000	11200.000000	9.380000	0.130000	601.000000	0.000000
50%	26.000000	8.282900e+04	4.000000	14000.000000	11.010000	0.180000	640.000000	0.000000
75%	30.000000	1.147602e+05	8.000000	18250.000000	13.350000	0.250000	670.000000	1.000000
max	49.000000	1.741243e+06	31.000000	35000.000000	20.000000	0.660000	805.000000	1.000000

2.2 Understanding Data

Data understanding is the process of examining and comprehending a dataset before analysis. It involves identifying the structure, content, and quality of the data by exploring its attributes, types, and relationships.

2.2.1 Checking Datatype of each column

- **".columns" -** This retrieves all column names from the Dataset.
- **".dtypes" -** This returns the data types of each column in the Dataset, showing whether a column contains integers (int64), floats (float64), or strings/objects (object).

```
loan[loan.columns].dtypes
```

```
age                    int64
gender                object
education             object
income                 int64
experience             int64
home_ownership        object
loan_amount            int64
loan_intent           object
loan_int_rate        float64
loan_percent_income  float64
credit_score           int64
pending_loan          object
loan_status            int64
dtype: object
```

2.2.2 Information about the Dataset

- The ***".info()"*** method in Python is used with pandas DataFrames to display a summary of the dataset. It provides essential details such as the number of entries (rows), the number of columns, column names, data types, and the count of non-null values in each column.

```
#information about the dataset
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18714 entries, 0 to 18713
Data columns (total 13 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  18714 non-null  int64
 1   gender               18714 non-null  object
 2   education            18714 non-null  object
 3   income               18714 non-null  int64
 4   experience           18714 non-null  int64
 5   home_ownership       18714 non-null  object
 6   loan_amount          18714 non-null  int64
 7   loan_intent          18714 non-null  object
 8   loan_int_rate        18714 non-null  float64
 9   loan_percent_income  18714 non-null  float64
 10  credit_score         18714 non-null  int64
 11  pending_loan         18714 non-null  object
 12  loan_status          18714 non-null  int64
dtypes: float64(2), int64(6), object(5)
memory usage: 1.9+ MB
```

2.2.3 Categorical variables

- **.select_dtypes(include=['object', 'category']).columns**
  This method selects columns based on their data type.
- **include=['object', 'category']** selects only numerical columns
- 'object' → Text-based (string) columns.
- 'category' → Categorical data columns (if they exist).
- .columns → Returns only the column names.

```
#categorical variables
loan.select_dtypes(include=['object', 'category']).columns
```

```
Index(['gender', 'education', 'home_ownership', 'loan_intent', 'pending_loan'], dtype='object')
```

2.2.4 Continuous variables

- **.select_dtypes(include=[‘int ’, ‘float’]).columns**
  This method selects columns based on their data type.
- **include=[‘int’, ‘float’]** selects only numerical columns
- ‘int’ → Integer type columns.
- ‘float’ →Floating point (Decimal numbers).
- .columns → Returns only the column names.

```
#continuous variables
loan.select_dtypes(include=['int', 'float']).columns
```

```
Index(['age', 'income', 'experience', 'loan_amount', 'loan_int_rate',
       'loan_percent_income', 'credit_score', 'loan_status'],
      dtype='object')
```

CHAPTER-III

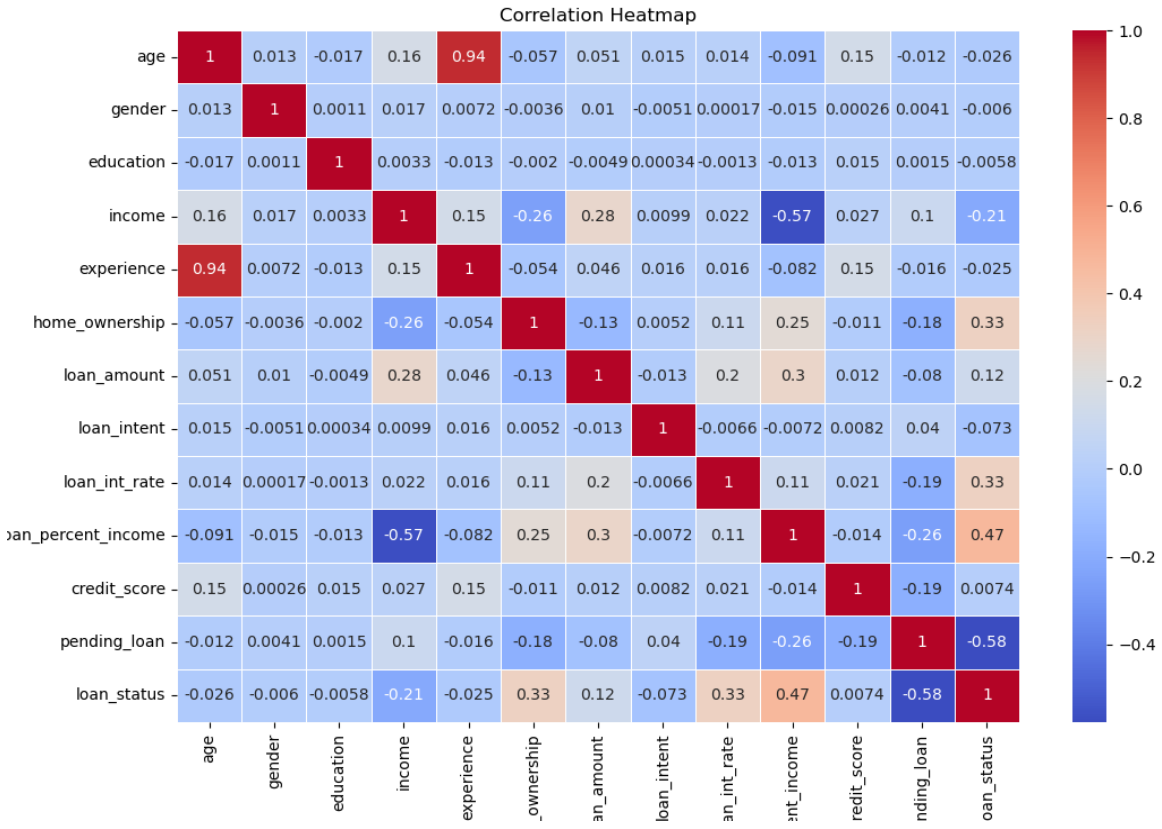
PREPARING & EXPLORING DATA

3.1 Data Exploration

Data exploration is the initial step in data analysis where raw data is examined to understand its structure, patterns, and relationships. It involves summarizing statistics, visualizing distributions and understanding correlations between variables.

3.1.1 Relationship between the variables

- The purpose of finding the relationship between two variables is to understand how they influence each other, which helps in making data-driven decisions.
- ***".corr( )"*** is a method in python which is used to find correlation. A **heatmap** in Python is a graphical representation of data that uses colours to show different values.
- Here, heatmap is used to find relationship between the variables (Positive or Negative relationship).



3.1.2 Chart Comparison

Chart comparison is the process of analyzing and interpreting multiple charts to identify differences, trends, and patterns in data. It helps in understanding variations between datasets, comparing performance metrics, and making informed decisions.

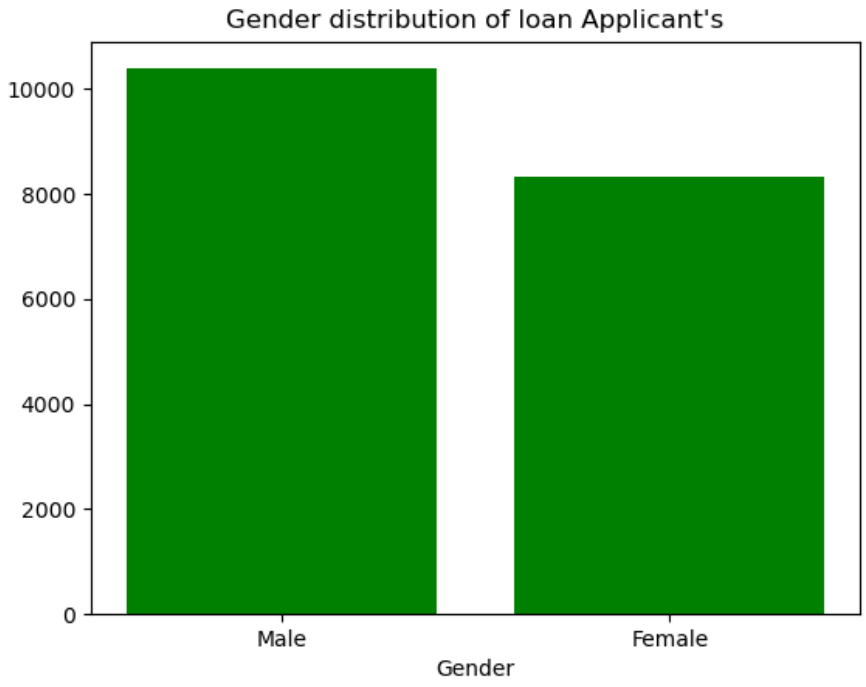
Plot 1 – Gender Distribution of Loan Applicant's

- ***".value_counts()"*** is a function in python which is a convenient way to count the number of occurrences of each unique value.
- ***".unique()"*** is a function returns unique value from a column.

```
gender_counts=loan.gender.value_counts()
gender=loan.gender.unique()
```

```
plt.bar(gender,gender_counts,color='green')
plt.xlabel("Gender")
plt.title("Gender distribution of loan Applicant's")
plt.savefig('Barplot')
plt.show()
```

Here,Barplot is used to find the gender-wise count of loan applicants. ***'plt.bar()'*** is a function in matplotlib used to draw Barplot. This plot clearly illustrates the gender distribution of loan applicants.



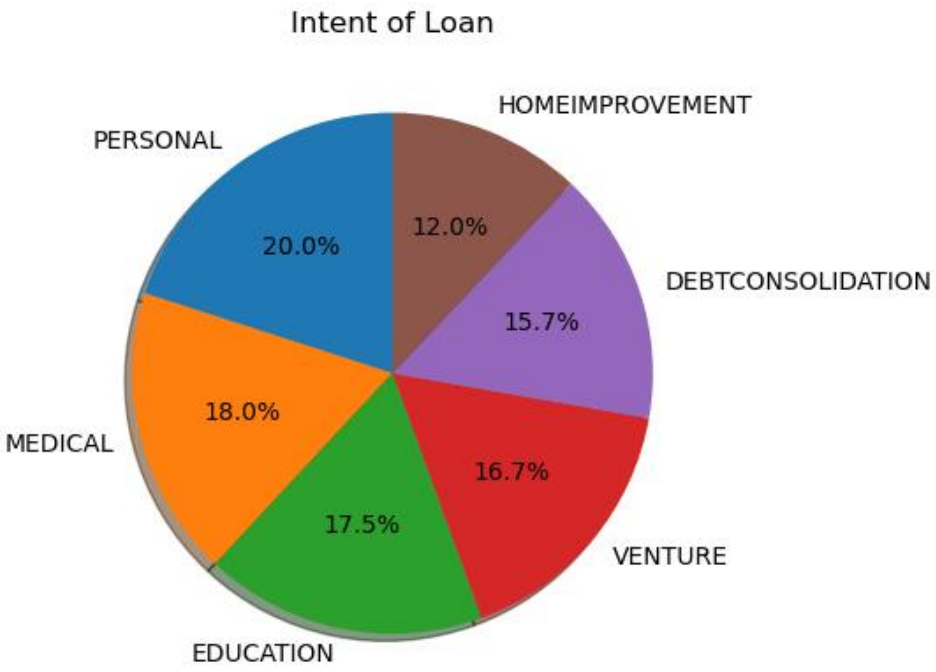
Plot 2 – Intent of Loan

- ***".value_counts()"*** is a function in python which is a convenient way to count the number of occurrences of each unique value.
- ***".unique()"*** is a function returns unique value from a column.

```
intent_values=loan.loan_intent.value_counts()
intent=loan.loan_intent.unique()
```

```
plt.pie(intent_values,startangle=90,labels=intent,autopct='%1.1f%%',shadow=True)
plt.title('Intent of Loan')
plt.savefig('intent')
plt.show()
```

Here, Pie chart is used to find the intent of loan applicants. ***"plt.pie()"*** is a function in matplotlib used to draw pie chart. This plot clearly illustrates the purpose for which applicants applied for loans.



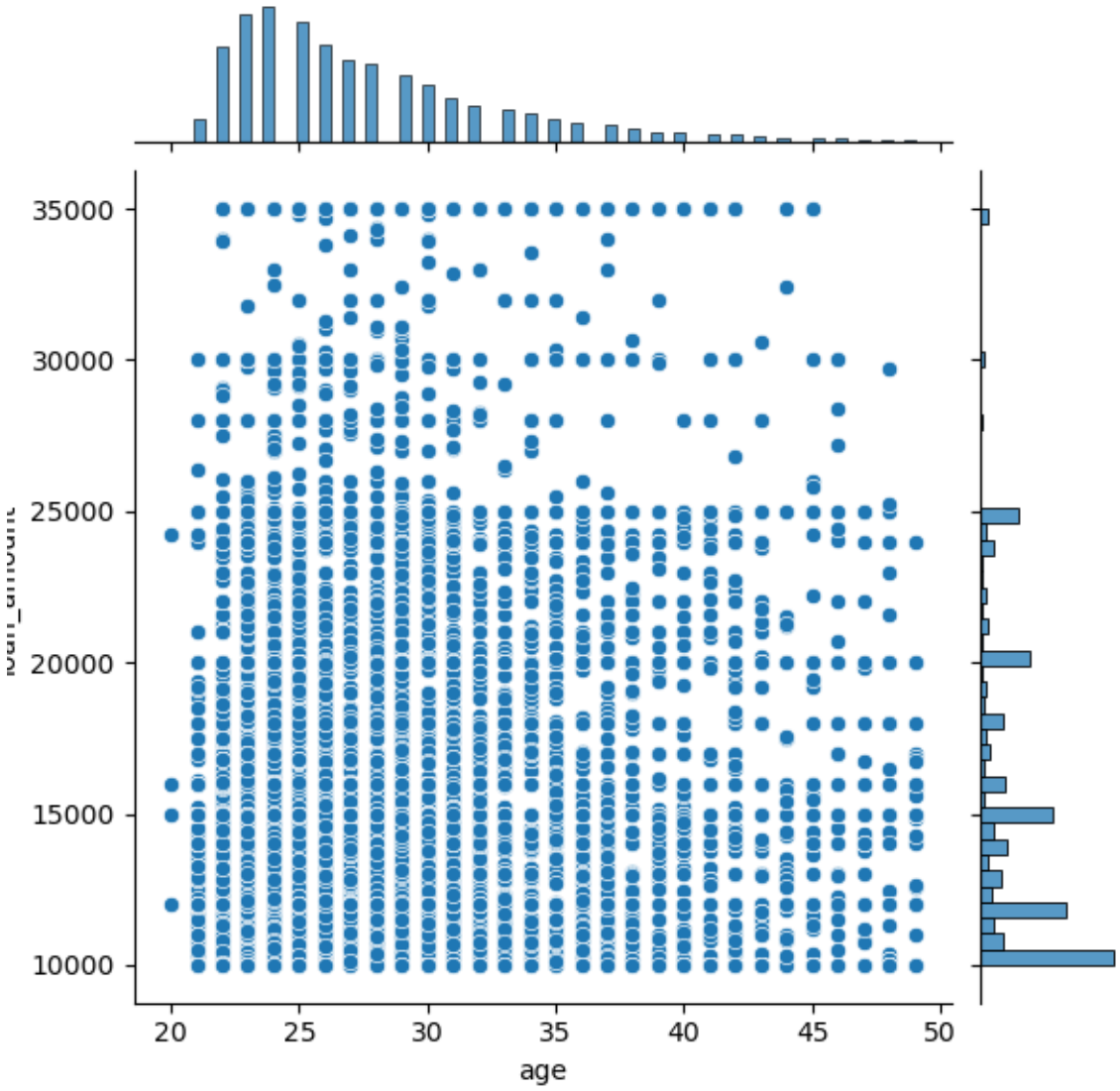
Plot 3 – Age Vs Loan Amount

A jointplot in Seaborn is a function used to visualize the relationship between two variables along with their individual distributions. It creates a scatter plot in the center and marginal histograms on the sides.

```
sns.jointplot(x=loan.age,y=loan.loan_amount)
plt.savefig('jointplot')
plt.show()
```

Here, Jointplot is used to find relation between Age and Loan Amount. ***"sns.jointplot( )"*** is a function in seaborn which is used to draw jointplot.

This plot clearly explains the relationship between Age and Loan Amount.



3.2 Issues in the dataset

Issues in the dataset refer to inconsistencies and problems that can impact data analysis and model performance. Common issues include missing values, NA values, duplicate records, outliers, inconsistent data formats, incorrect data types, and imbalanced data distribution.

3.2.1 NA's in the Dataset

NA (Not Available) values in a dataset represent missing or undefined data. These occur when certain data points are not recorded or are unavailable due to errors, incomplete data collection, or system limitations.

- **.isna( ) –** This function returns a Boolean value. "TRUE" indicates a missing value. "FALSE" indicates value is not missing.
- **.sum( ) –** This function adds all the column and returns in binary numbers (TRUE – 1,FALSE – 2)

```
#NA values in the dataset
loan.isna().sum()
```

```
age                    0
gender                 0
education              0
income                 0
experience             0
home_ownership         0
loan_amount            0
loan_intent            0
loan_int_rate          0
loan_percent_income    0
credit_score           0
pending_loan           0
loan_status            0
dtype: int64
```

3.2.2 Outliers in the Dataset

Outliers in a dataset are data points that significantly deviate from the overall pattern of the data. They can be extremely high or low values compared to the majority of the data points.

Here, IQR – Interquartile Range is used to check if any outliers are there or not.

- **".quantile(0.25)"** calculates the 25th percentile (first quartile, Q1), which is the value below which 25% of the data falls.
- **".quantile(0.75)"** calculates the 75th percentile (third quartile, Q3), which is the value below which 75% of the data falls.
- The **Interquartile Range (IQR)** is the difference between the third quartile (Q3) and the first quartile (Q1). This measures the spread of the middle 50% of the data.
- lower_bound = Q1 - 1.5 * IQR gives the lower threshold.
- upper_bound = Q3 + 1.5 * IQR gives the upper threshold.
- loan[(loan >= lower_bound) & (loan <= upper_bound)]: This filters the dataset, keeping only the rows where the values are between the lower and upper bounds.

```
# Calculate Q1, Q3, and IQR
Q1 = loan.quantile(0.25)
Q3 = loan.quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter the data to remove outliers
loan_filtered = loan[(loan >= lower_bound) & (loan <= upper_bound)]

#Comparing with original Dataset
print("Original dataset size:", loan.shape)
print("Filtered dataset size:", loan_filtered.shape)
```

```
Original dataset size: (18714, 13)
Filtered dataset size: (18714, 13)
```

3.2.3 Duplicates in the dataset

Duplicates in a dataset refer to rows or records that appear more than once, containing identical values across all or certain key columns.

- **.duplicated( ) –** This function returns a Boolean value. “TRUE” indicates a duplicated value. “FALSE” indicates value is not duplicated.
- **.sum( ) –** This function adds all the column and returns in binary numbers (TRUE – 1,FALSE – 2)

```
loan.duplicated().sum()
```

```
0
```

3.3 Resolve Issues

Resolving issues in a dataset refers to the process of identifying and fixing data problems to improve its quality and reliability for analysis.

This includes handling missing values (NA), removing duplicates, correcting inconsistent data formats, treating outliers, and converting data types appropriately.

By checking above processes **(Outliers,Duplicates and NA)**, the result shows that there is no issues in the dataset.

```
#NA values in the dataset
loan.isna().sum()
```

```
age                   0
gender                0
education             0
income                0
experience            0
home_ownership        0
loan_amount           0
loan_intent           0
loan_int_rate         0
loan_percent_income   0
credit_score          0
pending_loan          0
loan_status           0
dtype: int64
```

CHAPTER-IV
BUSINESS INTELLIGENCE AND INTERACTIVE DASHBOARD

4.1 Dashboard Interpretation

Dashboard interpretation is the process of analyzing and making sense of the data visualized in an interactive dashboard. Dashboards typically include various types of charts, graphs, and key performance indicators (KPIs) to display important information.

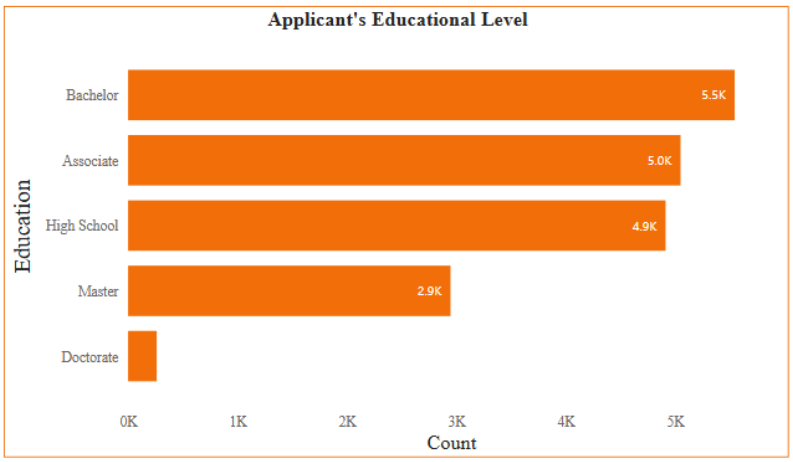
4.1.1 Overview of Loan Applications

- **Total Loan Applicants → 18,714** applications were submitted for loans. This number provides an overview of demand for loans and helps financial institutions determine approval rates.
- **Average Loan Amount → ৳15.42K**, which represents the typical loan amount requested by borrowers. Financial institutions can use this to set average loan disbursement limits.
- **Maximum Income** → The highest reported applicant income is **৳3,452,486**, indicating a wide variation in borrowers' financial backgrounds.
- **Average Interest Rate → 11.36%** is the average interest charged on loans, which helps assess affordability for borrowers.

4.1.2 Applicant's Educational Level Analysis

A horizontal bar chart illustrates applicants' education levels, which significantly impact loan approvals.

- The majority of loan applicants hold a Bachelor's degree (5.5K)**,** followed by Associate degrees (5.0K) and High School graduates (4.9K)**.**
- A small number of applicants have Master's degrees (2.9K) **or** Doctorates, indicating that higher education levels are less common among borrowers.
- Financial institutions may consider applicants with higher education as more financially stable due to potential employment opportunities and income growth.



4.1.3 Credit Score Classification

A categorical filter classifies applicants into different credit score levels:

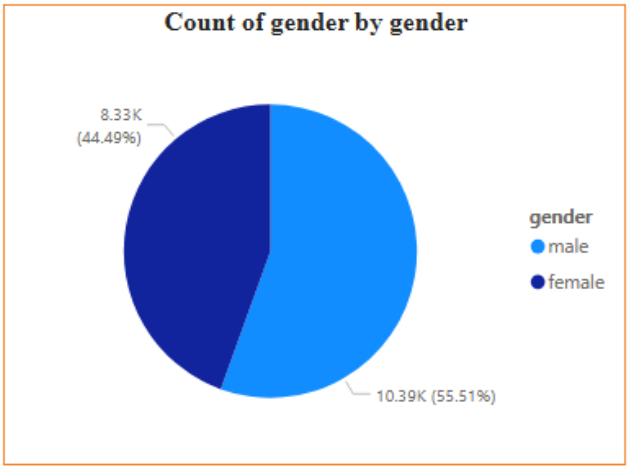
- **300 - 579 (POOR)**
- **580 - 669 (FAIR)**
- **670 - 739 (GOOD)**
- **740 - 799 (VERY GOOD)**
- **800 - 850 (EXCELLENT)**

Higher credit scores increase the likelihood of loan approval at lower interest rates, while lower scores indicate higher risk for lenders.

4.1.4 Gender Distribution in Loan Applications

A pie chart shows the gender distribution:

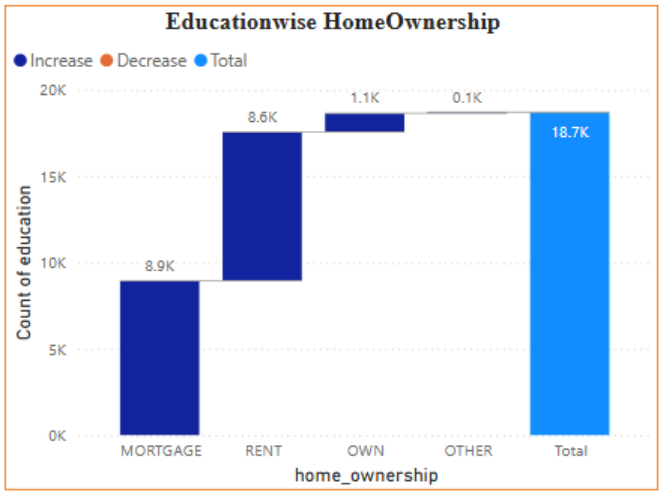
- **55.19% (10,336)** of applicants are **male**.
- **44.4% (8,378)** of applicants are **female**.



4.1.5 Home Ownership by Education Level

A stacked column chart visualizes the correlation between education level and home ownership**:**

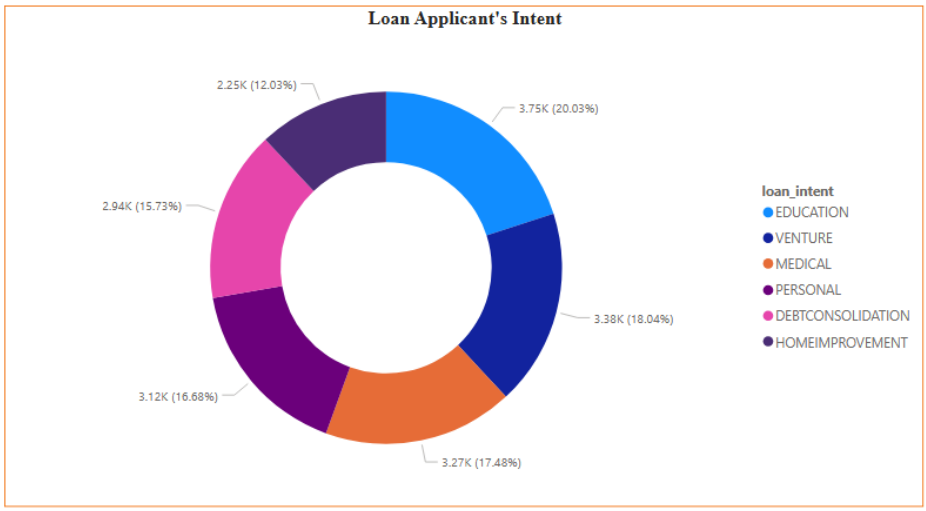
- Individuals with Bachelor's and High School degrees have the highest home ownership rates.
- Renters are more common among Associate degree holders, possibly due to lower income levels.
- This insight helps banks tailor mortgage loan policies based on education and financial stability.



4.1.6 Loan Intent Analysis

A donut chart represents the different reasons for loan applications.

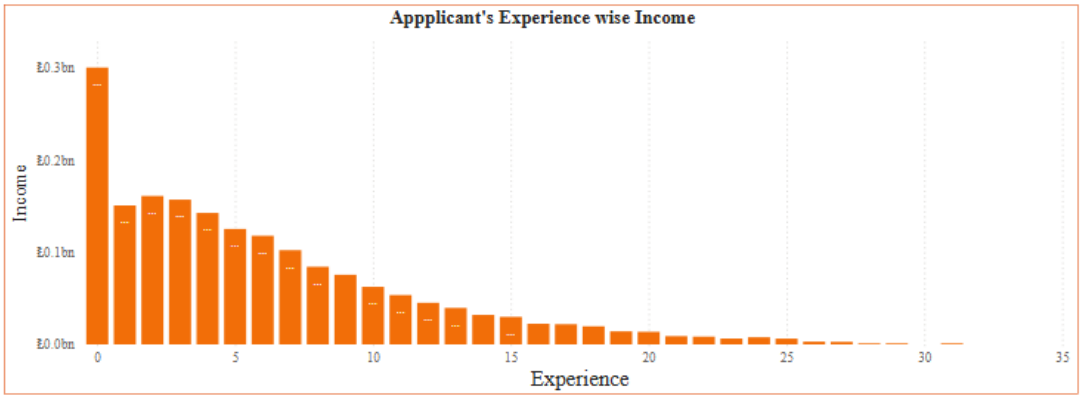
- Debt Consolidation (17.48%) is the most common loan purpose, indicating borrowers aim to manage multiple debts efficiently.
- Personal Loans (16.68%) and Home Improvement (18.04%) are also significant loan intents.
- Other categories include Education, Medical Expenses, and Business Ventures, each comprising a smaller percentage.



4.1.7 Applicant's Experience vs. Income

A histogram plots the relationship between work experience and income levels.

- Higher experience correlates with higher income levels, which makes loan repayment easier.
- New entrants with less than five years of experience tend to have lower income, making them higher-risk borrowers**.**
- This insight can be used to set minimum experience criteria for loan eligibility.



CHAPTER-V

MODEL BUILDING

5.1 Algorithm

- This dataset consists of loan applications from Turkey, with the goal of predicting whether a loan will be approved or rejected based on borrower profiles.
- In machine learning terms, this is a supervised learning classification problem, where past loan application data are used to train a model to predict loan approval status (target variable).
- The dataset includes various features such as income, credit score, employment status, loan amount, and repayment history, which influence the approval decision.
- Scikit-learn is an open-source machine learning library for Python that provides simple and efficient tools for data analysis and modeling. It is built on top of other scientific libraries like NumPy, SciPy, and matplotlib, and offers a wide range of algorithms and utilities for machine learning tasks.
- Algorithm used:
    - Decision Tree
    - Random Forest

5.1.1 Explanation of Decision Tree

A **Decision Tree** is a supervised machine learning algorithm used for classification and regression tasks. It works by breaking down a dataset into smaller subsets while developing an associated decision tree incrementally. The final result is a tree with decision nodes and leaf nodes.

Working Principle of Decision Tree

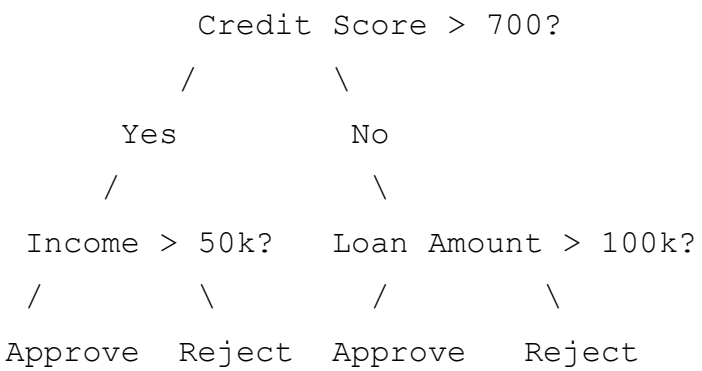
1. **Root Node Formation**
    - The algorithm starts by selecting the best feature to split the dataset.
    - The best feature is chosen using criteria like Gini Impurity, Entropy (Information Gain), or Mean Squared Error (for regression).

2. **Splitting the Data**
    - The dataset is split into two or more subsets based on the selected feature.
    - This process is repeated for each subset, forming branches.
3. **Creating Decision Nodes and Leaf Nodes**
    - Decision nodes represent conditions on features that split the data.
    - Leaf nodes contain the final output (class label in classification, predicted value in regression).
4. **Stopping Criteria**

    The tree stops growing when:
    - A predefined depth is reached.
    - All data in a node belongs to the same class (pure node).
    - No further information gain can be achieved.
    - The number of samples in a node falls below a minimum threshold.
5. **Prediction**
    - For a new input, the algorithm starts at the root node and follows the path based on feature values until it reaches a leaf node, which gives the final prediction.

Example

For a loan approval decision tree, features like credit score, income, and loan amount may be used. The tree might look like this:



5.1.2 Explanation of Random Forest

Random Forest is an ensemble learning algorithm that combines multiple decision trees to improve the accuracy and robustness of predictions. It is commonly used for both classification and regression tasks. The idea is to create a "forest" of decision trees, where each tree makes a prediction, and the final prediction is based on the aggregation of all trees' outputs.

Working Principle of Random Forest

1. **Make Many Decision Trees**
    - Random Forest creates many small decision trees using different parts of the data.
    - Each tree learns in its own way by using random samples and random features.
2. **Each Tree Makes a Prediction**
    - Every tree in the forest makes its own decision (e.g., "Approve Loan" or "Reject Loan").
3. **Final Decision**
    - For classification (Yes/No problems), the final answer is the one most trees agree on (majority vote).
    - For regression (predicting numbers), the final answer is the average of all tree predictions.

5.2 Training and Test Dataset

Before splitting the dataset into train and test, Use **Label Encoding** method to convert categorical variables in the loan dataset into numerical values.

5.2.1 Label encoding

Label Encoding is a technique used to convert categorical variables into numerical values. This is important because many machine learning algorithms require input features to be numerical. In label encoding, each unique category in a categorical column is assigned a distinct integer value.

```
from sklearn.preprocessing import LabelEncoder
# categorical to continuous
label_encoders = {}
for column in loan.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    loan[column] = le.fit_transform(loan[column])
    label_encoders[column] = le
```

Code Explanation

- Imports the LabelEncoder class from sklearn.preprocessing.
- LabelEncoder converts categorical (text) data into numbers.
- **label_encoders = {}** - Creates an empty dictionary to store label encoders for each categorical column.
- Loops through all categorical columns in the `loan` dataset.
- **select_dtypes(include=['object'])** - selects only columns with text (categorical) values.
- **le = LabelEncoder( ) -** Creates a new instance of LabelEncoder for each categorical column.
- **fit_transform() -** Fits the encoder to the unique values in the column and Transforms the categorical values into numbers.
- **label_encoders[column] = le -** Saves the encoder for each column in label_encoders, allowing you to later convert the numbers back to original categories if needed.

5.2.2 Train Test Split

Splitting the dataset into training and testing parts helps us check if our machine learning model works well on new data. The training set (usually 80%) is used to teach the model, while the testing set (usually 20%) is used to see how well the model makes predictions on data it has never seen before. This prevents the model from just memorizing the data and helps make sure it can make good predictions in real life.

```
from sklearn.model_selection import train_test_split

# Splitting features (X) and target (y)
X = loan.drop('loan_status', axis=1)
y = loan['loan_status']

# Split the dataset into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Display the sizes of each split
print("Training data size:", X_train.shape, y_train.shape)
print("Test data size:", X_test.shape, y_test.shape)
```

```
Training data size: (14971, 12) (14971,)
Test data size: (3743, 12) (3743,)
```

Code Explanation

- Brings in the train_test_split function from sklearn.model_selection.
- This function helps to split the dataset into training and testing sets.
- X: Stores all the columns except loan_status (independent variables).
- y: Stores only the loan_status column (the target variable we want to predict).
- train_test_split(X, y, test_size=0.2, random_state=42) breaks the dataset by splitting 80% training data (used to train the model) and 20% testing data (used to evaluate the model).
- random_state=42: Ensures the split is consistent every time you run the code.

5.3 Model Building

Model building is the process of designing, training, and evaluating a machine learning model to solve a specific problem or make predictions based on data. The goal is to create a model that can learn patterns from the data and generalize well to new, unseen data.

5.3.1 Decision Tree

A **Decision Tree** is a machine learning algorithm that works like a flowchart to make decisions. It splits data into smaller parts based on different conditions, like answering a series of yes/no questions. Each decision leads to further branches until it reaches a final outcome.

```
from sklearn.tree import DecisionTreeClassifier
# Initialize and train the Decision Tree Classifier
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Make predictions
y_pred = dt_model.predict(X_test)
```

Code Explanation

- **DecisionTreeClassifier**: A class in scikit-learn used to create a Decision Tree model for classification tasks.
- **dt_model = DecisionTreeClassifier(random_state=42)**:
  - Initializes a Decision Tree model**.**
  - random_state=42: Ensures reproducibility, so every time the code runs, the results are the same.
- **dt_model.fit(X_train, y_train):**
  - Trains the model using the training data (X_train and y_train).
  - The model learns the patterns and relationships in the data to predict the target variable (loan_status).
- **dt_model.predict(X_test):**
  - Uses the trained model to make predictions on the test data (X_test).
  - y_pred contains the predicted values for loan_status based on the features in X_test.

5.3.2 Random Forest

A **Random Forest** is a machine learning algorithm that combines many decision trees to make predictions. Instead of relying on one decision tree, it creates a forest of trees, each making its own decision. The final prediction is based on the majority vote (for classification) or the average (for regression) of all the trees.

```
from sklearn.ensemble import RandomForestClassifier
# Initialize the Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_model.predict(X_test)
```

Code Explanation

- **RandomForestClassifier** is a class from scikit-learn used to create a Random Forest model for classification tasks (predicting categories such as loan approval).
- **RandomForestClassifier(n_estimators=100, random_state=42):**
  - **n_estimators=100**: This sets the number of trees in the forest to 100. More trees generally improve the model's performance, but take more time to train.
  - **random_state=42**: Ensures reproducibility. The same split of data and randomization will occur every time the code runs, producing consistent results.
- **rf_model.fit(X_train, y_train):**
  - Trains the Random Forest model on the training data (X_train and y_train)
  - The model learns from the data by building multiple decision trees, each with different random selections of data and features.
- **rf_model.predict(X_test)**:
  - Makes predictions using the trained Random Forest model on the test data (X_test).
  - The predicted values (y_pred) represent the loan status (approved or rejected) for each test case based on the patterns the model has learned.

CHAPTER-VI

EVALUATION OF MODEL

6.1 Model Evaluation

Model evaluation is essential to assess the performance of the machine learning algorithms used in this study. Two classification models, Decision Tree and Random Forest, were implemented to predict loan approvals. Their effectiveness was measured using key performance metrics: accuracy, precision, recall, and F1-score**.**

6.1.1 Accuracy Analysis

Accuracy analysis refers to the process of evaluating the performance of a machine learning model by measuring how accurately it makes predictions compared to the actual or true values. It is one of the most common metrics for assessing model performance, especially for classification tasks.

- The Decision Tree model achieved an accuracy of **90.25%**, meaning it correctly classified loan applications in 90.25% of cases.

```
from sklearn.metrics import accuracy_score
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy of the Decision Tree Model: {accuracy * 100:.2f}%")
```

```
Accuracy of the Decision Tree Model: 90.25%
```

- The Random Forest model outperformed the Decision Tree with an accuracy of **93.75%**, showing better generalization by reducing overfitting through an ensemble of multiple decision trees.

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print(f"\nAccuracy of the Random Forest Model: {accuracy * 100:.2f}%")
```

```
Accuracy of the Random Forest Model: 93.75%
```

6.1.2 Precision, Recall, and F1-Score

Apart from accuracy, additional metrics provide deeper insights into model performance:

- **Precision**: Measures how many loans predicted as "approved" were actually correct. A higher precision reduces the risk of approving bad loans.

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

  - **Decision Tree Precision:** 79.94%
  - **Random Forest Precision:** 91.18%
  - **Observation:** Random Forest has a higher precision

- **Recall**: Measures how many actual good loans were correctly approved. A higher recall ensures that fewer eligible borrowers are wrongly rejected.

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

  - **Decision Tree Recall:** 82.85%
  - **Random Forest Recall:** 83.78%
  - **Observation:** Both models perform well in identifying approved loans, but Random Forest is slightly better.

- **F1-Score**: A balance between precision and recall, useful when both false approvals (bad loans) and false rejections (missed good loans) are important.

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

  - **Decision Tree F1-Score:** 81.37%
  - **Random Forest F1-Score:** 87.32%
  - **Observation:** Random Forest achieves a better balance, making it the preferred model.

Decision tree:

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9024846379909164
Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.93      0.93      2781
           1       0.80      0.83      0.81       962

    accuracy                           0.90      3743
   macro avg       0.87      0.88      0.87      3743
weighted avg       0.90      0.90      0.90      3743

Confusion Matrix:
 [[2581  200]
 [ 165  797]]
```

Random Forest:

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9374833021640395
Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.97      0.96      2781
           1       0.91      0.84      0.87       962

    accuracy                           0.94      3743
   macro avg       0.93      0.90      0.92      3743
weighted avg       0.94      0.94      0.94      3743

Confusion Matrix:
 [[2703   78]
 [ 156  806]]
```

6.1.3 Model Performance Comparison

Metric	Decision Tree	Random Forest
Accuracy	90.25%	93.75%
Precision	79.94%	91.18%
Recall	82.85%	83.78%
F1-Score	81.37%	87.32%

CHAPTER-VII

PREDICTION AND INFERENCE

7.1 Prediction

The models developed in this study were used to predict loan approval outcomes based on borrower profiles. Two machine learning algorithms, Decision Tree and Random Forest**,** were applied to the dataset, with Random Forest achieving the highest accuracy (93.75%)**.**

Prediction Results

After training the models, the predictions were tested on unseen data. The **Random Forest model**, with its ensemble learning approach, produced more reliable predictions compared to the **Decision Tree** model.

Model	Predicted Loan Approval (Yes/No)	Prediction Accuracy
Decision Tree	Approve or Reject Loan	90.25%
Random Forest	Approve or Reject Loan	93.75%

- The Random Forest model correctly classified **93.75%** of loan applications, making it the preferred choice for final predictions.
- This model is deployed to predict whether a loan application will be approved or rejected based on key borrower attributes such as credit history, income, employment status, and loan amount**.**

Minimizing Financial Risk for Banks

- A high precision **(93.75%)** in the Random Forest model means fewer risky loans get approved, helping banks avoid losses.
- The model ensures that borrowers with a higher chance of repaying the loan are prioritized, improving financial stability.

7.2 Inference

Inference refers to the process of using a trained machine learning model to make predictions or draw conclusions from new, unseen data. It helps in understanding patterns and relationships within the data and applying them to real-world scenarios.

- **Important Factors for Loan Approval**: Applicants with higher credit scores, stable income, and good repayment history are more likely to get loan approval. The loan amount and debt-to-income ratio also play a key role.

- **Best Performing Model**: The Random Forest model achieved the highest accuracy (93.75%), meaning it made the most correct predictions compared to other models like Decision Tree.

- **Most Influential Features**: The most important factors affecting loan approval were credit history, income, and loan amount, showing that financial stability is crucial for lenders.

- **Possible Bias in Data**: If the dataset contains more approved loans than rejected ones, the model might favor approvals. Techniques like balancing the data can improve fairness.

- **Ways to Improve the Model**: Adding more details about applicants, such as spending habits or savings, could make predictions more accurate. Also, trying advanced models like deep learning could enhance performance further.

CHAPTER-VIII

CONCLUSION

This study focused on analyzing borrower profiles and developing a predictive model for loan approval using machine learning techniques. The dataset included various borrower attributes such as credit history, income, loan amount, employment status, and repayment behavior. Two machine learning algorithms, Decision Tree and Random Forest, were implemented and evaluated for their effectiveness in predicting loan approvals. The Decision Tree model, though simple and easy to interpret, achieved an accuracy of 90.25%. However, it struggled with handling complex data patterns. On the other hand, the Random Forest model, an ensemble method combining multiple decision trees, outperformed the Decision Tree by achieving an accuracy of 93.75%, making it the most reliable choice for loan approval predictions.

The key findings of the study highlighted that the Random Forest model provided the most accurate and reliable predictions, with a higher precision of 91.18% and recall of 83.78%. This ensured fewer incorrect approvals and rejections, making it an effective tool for loan approval automation. The analysis also revealed that credit history was the most influential factor in determining loan approval, followed by income and loan amount. The implementation of machine learning models can automate the loan approval process, reducing manual efforts and increasing efficiency in financial institutions.

REFERENCES

- https://www.kaggle.com/code/makshafmehboob/decision-tree-classification
- https://jovian.com/aakashns/python-matplotlib-data-visualization
- https://learn.microsoft.com/en-us/power-bi/
- https://www.geeksforgeeks.org/learning-model-building-scikit-learn-python-machine-learning-library/
- Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. Springer.