Project Report

Prediction of Loan Approval using Machine Learning

Madhusudhan Reddy Kuppam – 11683683

Introduction:

Loan approval is a critical function for financial institutions that requires a thorough evaluation of applicants' financial situations and other relevant data to determine their eligibility for a loan. Traditionally, this process involves manual evaluation, which can be inefficient and may lead to inconsistencies and potential biases. Our project addresses this problem by leveraging machine learning techniques to predict loan approval based on existing data.

In this project, we explored various machine learning techniques, including supervised, unsupervised, and neural networks, to identify the most accurate and adaptable approach for predicting loan approvals. The goal is to analyze their effectiveness in predicting loan approval rates, a critical task for financial institutions aiming to streamline their decision-making process while minimizing biases and inefficiencies.

We are addressing the problem of determining the most suitable machine learning approach for predicting loan approvals by comparing and contrasting the performance of different algorithms on the **loan_approval_dataset.csv**. The hypothesis is that certain machine learning techniques may outperform others in terms of accuracy and adaptability, providing a reliable framework for automating loan approval processes. The goal is to find which machine learning model or approach provides the best accuracy and scalability for loan approval prediction.

Abstract:

The project examined a loan approval dataset of 4,269 records and 13 variables, with a focus on important parameters such as dependents, education, work status, annual income, loan amount, loan length, credit score, and asset values. The goal was to forecast the chance of loan acceptance using robust preprocessing and modelling techniques.

Methodologies:

Data pretreatment involved imputation for missing values, scaling numerical features, and one-hot encoding of categorical variables. The Exploratory Data Analysis (EDA) identified relationships between income levels, credit ratings, and approval results. We implemented Machine learning models like Knn, Logistic Regression, SVM, K means clustering, PCA, Neural Network, and Random Forest classifier were used to find and classify key determinants of loan approvals

Main Findings:

Income and Loan Amount: A strong link was found, with higher income greatly boosting approval chances.

Credit Score: An important component; better scores were strongly associated to approval likelihood.

Employment Type: Applicants with secure, full-time occupations received greater approval rates than those who were self-employed.

Loan-to-Value Ratio: Approval rates were lower for applicants seeking loans that were disproportionately large in comparison to their assets.

Predictive models accurately predicted approval outcomes with a reliability of around 85%.

Impact: The project provides financial institutions with tangible data to improve risk assessment and decision-making, simplify loan approval processes, and ensure fair appraisals.

Literature Review:

- **An improved random forest classifier for multi-class classification**

  The paper, An Improved Random Forest Classifier for Multi-Class Classification by Chaudhary et al., provides an advanced methodology for enhancing Random Forest classifiers, particularly for multi-class classification problems. This research presents the Improved Random Forest Classifier (Improved-RFC) approach, which integrates feature selection and resampling techniques to address the challenges of high-dimensional datasets and class imbalance. These enhancements make the method highly applicable to real-world tasks, including the predictive modeling goals of our project.

  One of the paper's key contributions is its focus on feature selection, employing methods like Correlation-Based Feature Selection (CFS) and Gain Ratio to reduce redundant and irrelevant attributes in the dataset. This aligns with our project's preprocessing objectives, where identifying the most predictive variables, such as credit scores and income ratios, is crucial for improving model accuracy and interpretability. The study demonstrates how feature selection optimizes the classifier by retaining only the most relevant attributes, a strategy that directly influences the efficiency of machine learning models.

- **Machine Learning: Algorithms, Real-World Applications and Research Directions**

  The paper Machine Learning: Algorithms, Real-World Applications, and Research Directions by Iqbal H. Sarker provides a comprehensive analysis of machine learning techniques, focusing on their principles, applications, and challenges. This work is directly relevant to our project, as it offers a foundational understanding of supervised learning and classification methods, which are pivotal to building predictive models like ours.

  The paper categorizes machine learning into supervised, unsupervised, semi-supervised, and reinforcement learning, with an emphasis on supervised learning for tasks involving labeled datasets. This aligns with our project, where classification algorithms like Logistic Regression, K-Nearest Neighbors (KNN), and Support Vector Machines (SVM) are employed to predict loan approvals. The discussion on supervised learning highlights

the effectiveness of these algorithms in structured datasets, such as the financial data we are working with.

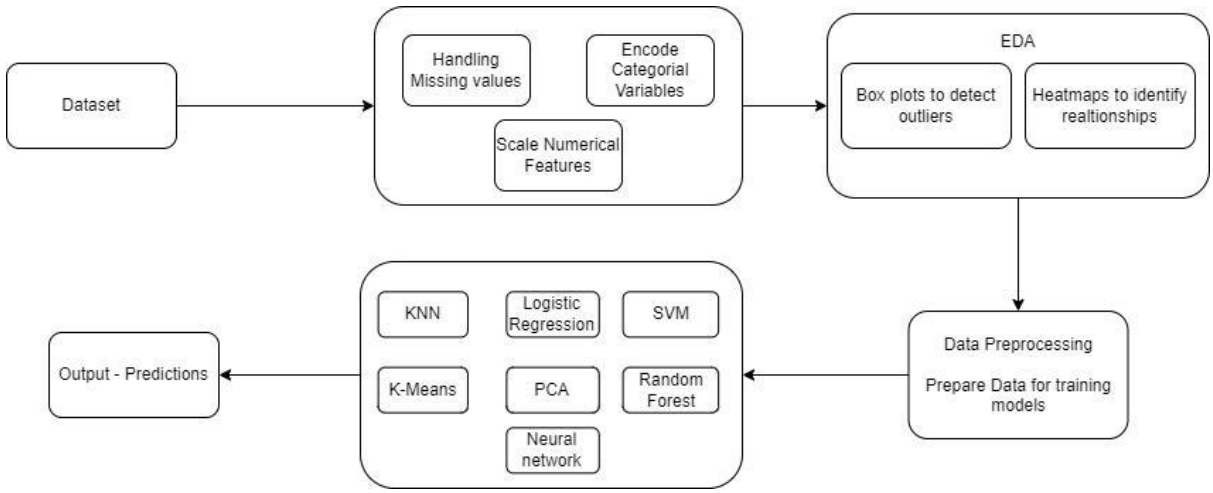
- **Selecting Machine Learning Algorithms using Regression Models**

  The paper Selecting Machine Learning Algorithms Using Regression Models by Tri Doan and Jugal Kalita addresses a critical challenge in machine learning: identifying the most suitable algorithm for a given dataset. This study introduces a meta-learning framework that leverages regression models to predict the performance of machine learning algorithms. By incorporating dataset-level meta-features, such as statistical summaries, and leveraging dimensionality reduction techniques like Principal Component Analysis (PCA), the authors propose a systematic approach to optimize algorithm selection. This approach aligns closely with the goals of our project, where selecting the most appropriate classifier for loan approval prediction is paramount.

  One significant contribution of the paper is its emphasis on dimensionality reduction to generate consistent feature sets across datasets. This method ensures that transformed datasets retain the essential characteristics needed for accurate performance predictions. The use of regression models, including tree-based and linear approaches, to rank algorithms provides an efficient way to evaluate multiple classifiers without exhaustive testing. For our project, this inspires the integration of meta-features derived from financial attributes, such as income levels and credit scores, to evaluate and rank classification algorithms like Logistic Regression, SVM, and KNN.

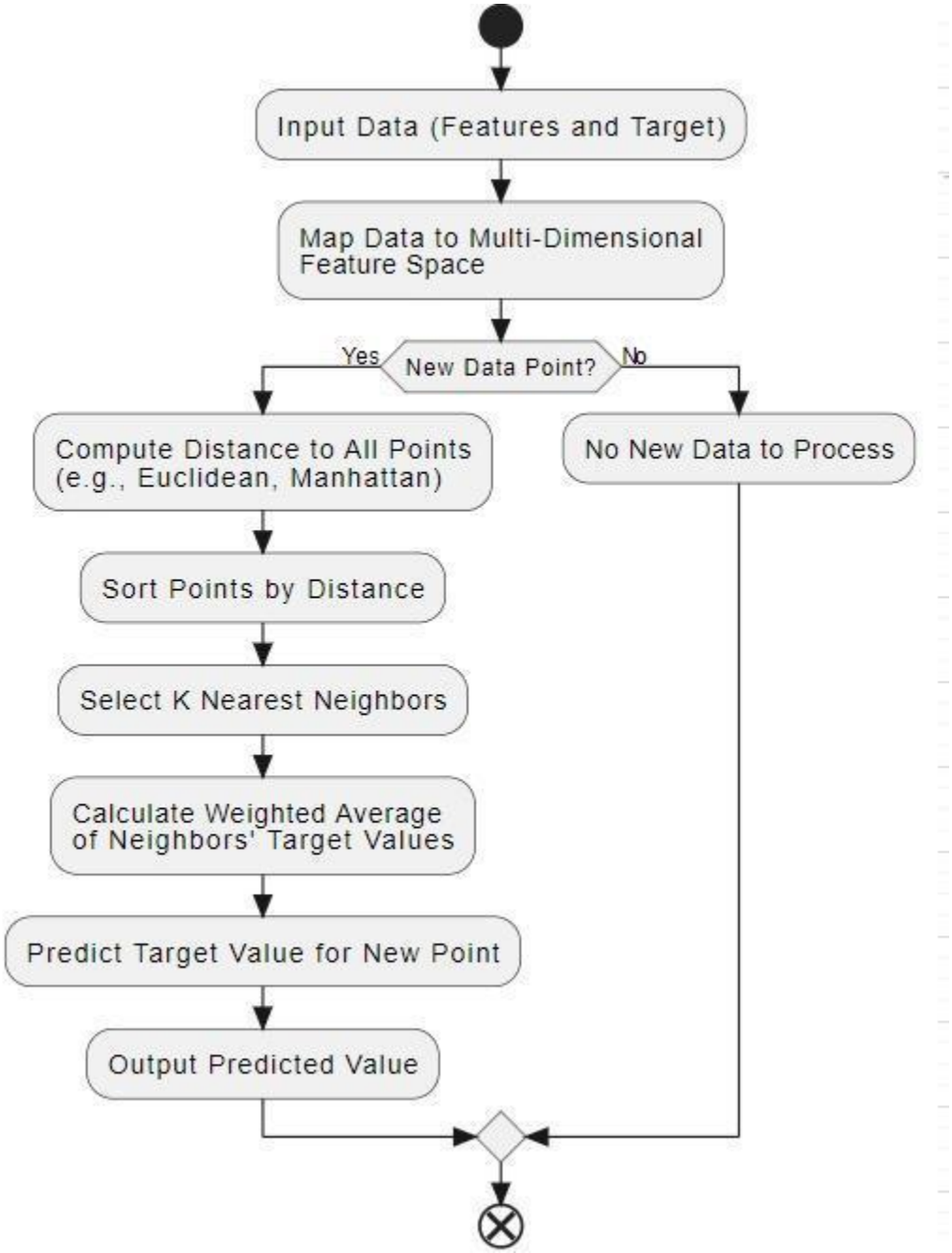
Methodology:

Overall Workflow Diagram:



Architecture Diagram:

KNN Regressor:



The K-Nearest Neighbours (KNN) algorithm's operation is illustrated in this diagram:

Input Data: Begin with data that includes target values (like loan approval) and attributes (like income and credit score).

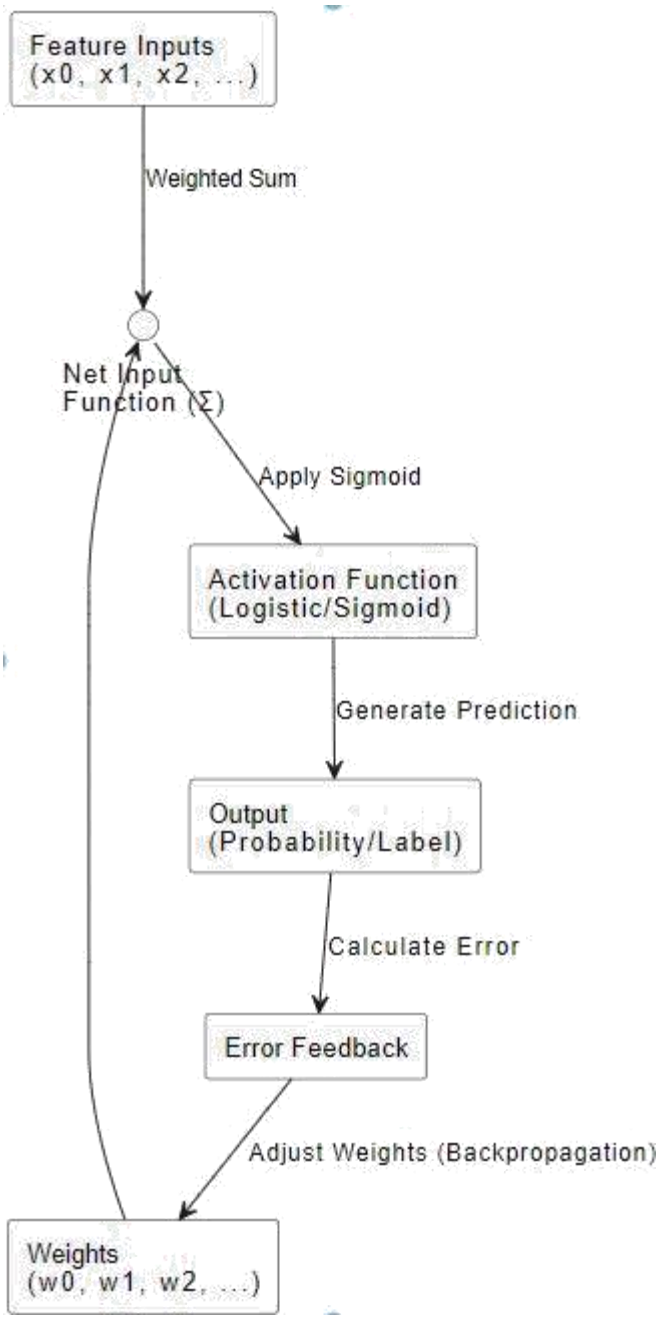
New Point: Determine the distance between each new data point and every other point.

Locate Your Neighbours: Find the "K" nearest points, or neighbours.

Predict: To forecast the outcome of the new point, use the target values of these neighbours (such as the average or majority vote).

Output: Give the anticipated outcome.

Logistic regression:



The steps involved in training a logistic regression or neural network model are depicted in this diagram:

Feature inputs: begin by entering features (such as x0, x1, and x2).

Net Input Function: Create a weighted sum by combining the features using weights (w0, w1, w2).

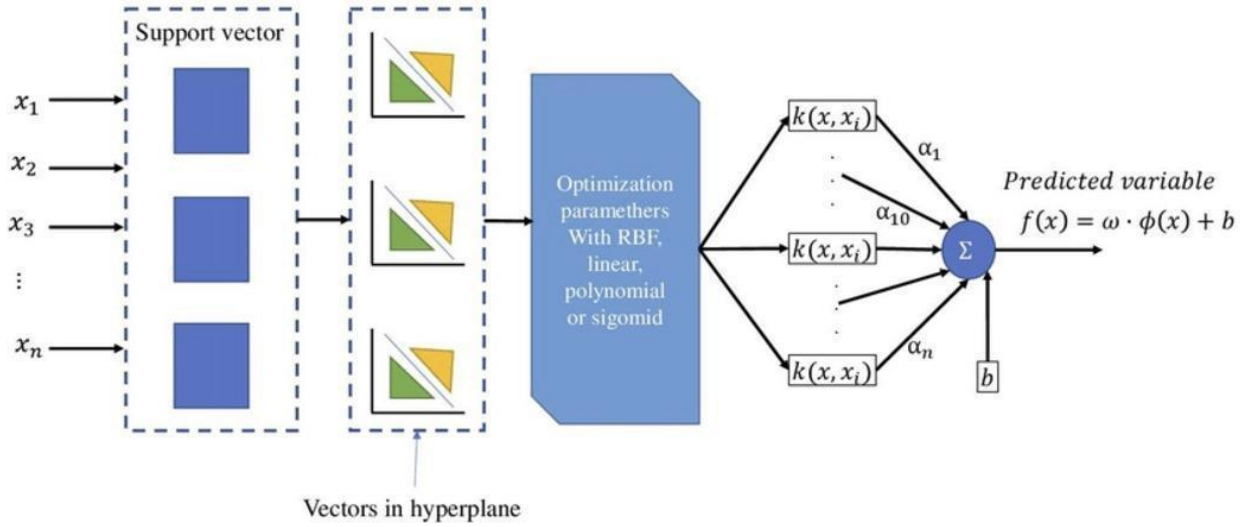
Activation Function: To translate the outcome into a probability between 0 and 1, apply a sigmoid function.

Create a Prediction: Provide the class label or probability.

Determine Error: Determine the error by comparing the expected and actual results.

Error Feedback: To reduce the error, use backpropagation to modify the weights.

Support Vector Machine (SVM):



This diagram illustrates how a Support Vector Machine (SVM) operates:

Input Features (x1, x2,... xn): These are the data points you wish to categorize or predict.

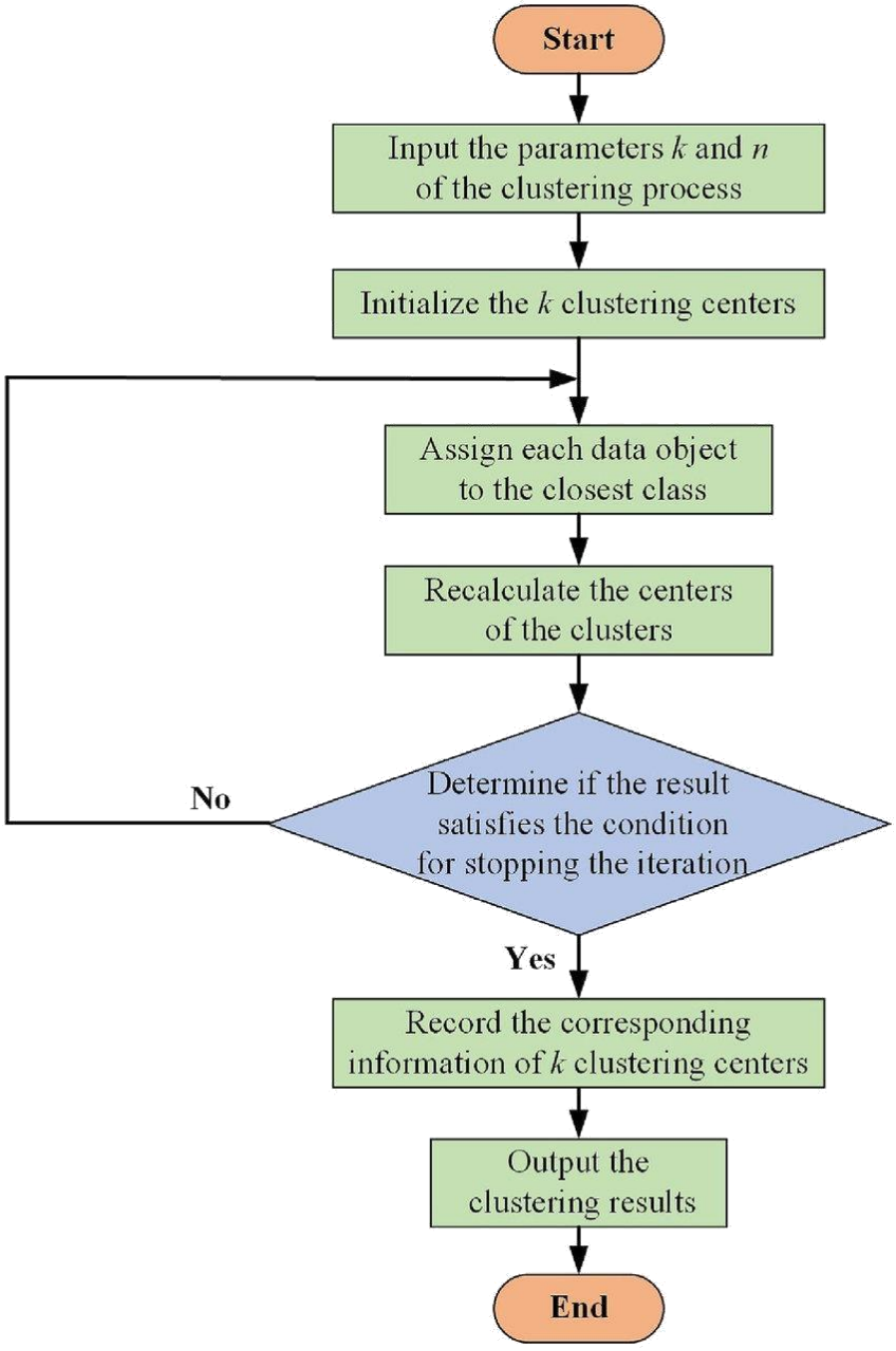
Support Vectors: Data points that define the boundary (or hyperplane) between classes.

Kernel Function: Converts data into a higher-dimensional space using functions such as linear, polynomial, RBF, or sigmoid, making it easier to distinguish between classes.

Optimization: The algorithm determines the optimum hyperplane by maximizing the margin between classes.

Prediction: The final decision function uses weights (α) and a bias term (b) to determine the output for a new point on the hyperplane.

KNN Clustering:



The K-Means Clustering algorithm is depicted in this diagram:

Start: Determine the number of clusters (k) and set the initialisation of k cluster centres at random.

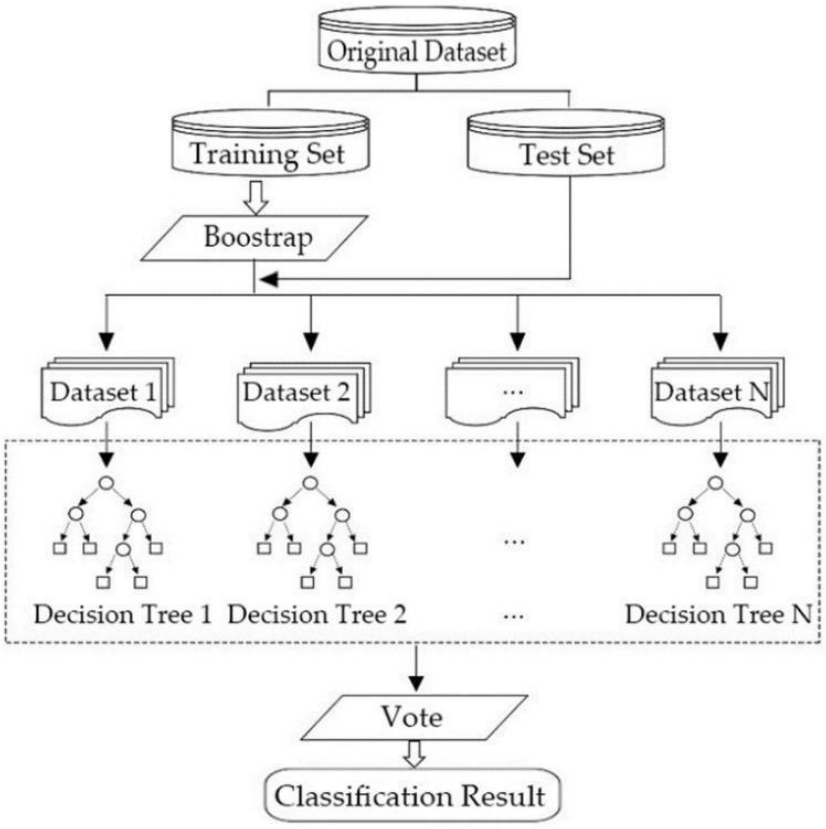
Assign Data: Using a distance metric (such as the Euclidean distance), assign each data point to the nearest cluster.

Recalculate the centres: Determine each cluster's new centre using the points that have been allocated.

Examine for Convergence: Check to see if centres or cluster allocations remain constant. Otherwise, repeat steps two and three.

Output: Finalise the cluster centres and output the findings after convergence is reached.

Random Forest Classifier:



This figure demonstrates how Random Forest works step-by-step:

Begin with data: We have a large dataset.

Split the data into two parts: some for training the model and others for subsequent testing.

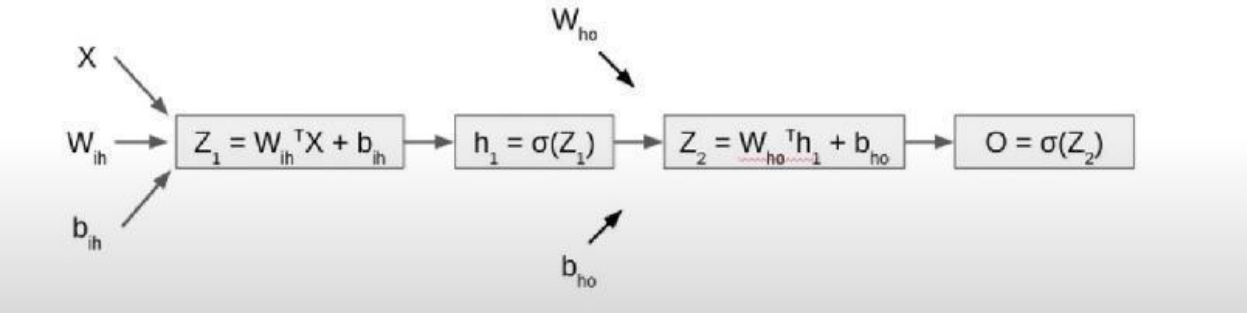
Create random samples: We divide the training data into numerous smaller, random datasets.

Create trees: For each little dataset, a decision tree is trained to generate predictions.

Combine the trees. The trees function together as a "forest."

Make predictions: When we need to classify anything new, each tree casts a vote, and the majority wins.

Neural Network Architecture:



This diagram demonstrates how a simple neural network functions:

Start with the inputs: You send data (X) to the network.

The first layer: the network performs math using weights (h W ih) and biases (h b ih) before applying an activation function to improve the outcome. This yields the output of the first hidden layer (h 1 h 1).

The second layer: the hidden layer output (h 1 h 1) undergoes additional calculations with new weights (h W ho) and biases (h b ho), followed by another activation function. This produces the final output (O).

Pseudocode:

Supervised learning:

KNN:

Import the essential libraries for data manipulation and visualization.

Load the data.

Separate the dataset into features (X) and targets (Y).

Preprocess the data.

Divide the preprocessed dataset into 70% training and 30% testing using train_test_split().

Set the hyperparameters for tuning:

- A list of k values (e.g. [1, 3, 5]).

- A list of weight techniques (such as ['uniform', 'distance']).

- A list of distance metrics (for example,

['euclidean','manhattan','minkowski']). Tune the hyperparameters.

Evaluate the best

model. Print the results.

End.

Logistic Regression:

Import the essential libraries for data manipulation and visualization.

Load the data.

Separate the dataset into features (X) and targets (Y).

Separate the dataset into training and testing sets with the parameters 'test_size=0.3' and 'random_state=42'.

Set up the logistic regression model.

Train the logistic regression model using training data 'X_train' and 'Y_train'.

Generate predictions 'Y_pred' on 'X_test' using the 'predict()' function.

Evaluate the model.

Validate the model using 'cross_val_score()' and 'cv=5'.

Calculate ROC-AUC.

Plot the ROC curve using matplotlib, comparing FPR and TPR.

Print the results.

End.

Support Vector Machine (SVM):

Import the essential libraries for data manipulation and SVM model training and evaluation.

Separate the dataset into features (X) and targets (Y).

Separate the dataset into training and testing sets with the parameters 'test_size=0.3' and 'random_state=42'.

Set up the Support Vector Machine (SVM) classifier.

Train the SVM classifier using training data 'X_train' and 'Y_train'.

Generate predictions 'Y_pred' on 'X_test' using the 'predict()' function.

Evaluate the best model.

Print the results.

End.

Unsupervised learning:

K-means Clustering:

Import the essential libraries for clustering and analysis.

Clean the dataset after loading it.

Get the numerical data ready for clustering.

Standardize the numeric data.

Apply K-Means Clustering.

Visualize the cluster distribution.

Show the revised dataset.

Determine the statistics for the cluster summary.

Display the distributions of numerical features among clusters.

Examine the connection between loan status and clusters.

Visualize loan status distribution across clusters.

End.

Principal Component Analysis (PCA):

Import the essential libraries for PCA transformation and analysis.

standardize the numerical data.

Use two components while applying PCA.

Include PCA elements in the dataset.

Use a 2D scatter plot to visualize the PCA components.

Show the revised dataset.

Use PCA with more elements.

Examine explained variance by calculating the ratio of explained variance for every PCA component.

Show the data on explained variance.

Show cumulative explained variance.

Use the first three PCA components to create a 3D scatter plot.

End.

Random Forest Classifier:

Import the essential libraries for neural encoding, evaluation, and training.

Clean up column names by using `str.strip()` to eliminate leading and trailing spaces.

Set up the categories column label encoders.

Encode categorical variables.

The data should be divided into features ({X`) and targets (`y`).

Divide the dataset into sets for testing and training.

Set up the Random Forest model and train.

Predict using the test data.

Evaluate the best model.

Print the results.

End.

Neural Network:

Import the essential libraries for neural network creation and training.

Scale the numerical Features.

The data should be divided into features ({X`) and target (`Y`).

Divide the dataset into sets for testing and training.

Define the neural network's architecture.

Construct the model for the neural network.

A summary of the model architecture should be printed.

Put the model together.

Get the model to train.

Display the training performance.

End.

Dataset:

- The loan approval dataset is a compilation of financial statements and related data utilized to assess the qualifications of individuals or businesses seeking loans from a financial institution.
- It consists of multiple factors of the applicant such as number of dependents, education, employment status, annual income, amount of loan required, term of the loan in years, credit score, value of residential, commercial, luxury, bank assets.
- It consists of 4269 rows and 13 columns.

Data set overview:

Dataset comntains 12 columns and 4269 rows. Most of the columns are integers except education, self_employed and loan_status are object which can be considered as categorical variables.

Description of each column:

- loan_id : identifier for each loan application
- no_of_dependents : number of dependents on the applicants
- education : Whether the applicant is graduate or not
- self_employed : Whether the applicant is self-employed or not
- income_annum : Annual income of the applicant
- loan_amount : amount of loan applied
- loan_term : duration of the loan in months
- cibil_score : CIBIL score of the applicant
- residential_assets_value : Value of residential assets owned
- commercial_assets_value : Value of commercial assets owned
- luxury_assets_value : Value of luxury assets owned by the applicant
- bank_asset_value : Value of bank assets owned
- loan_status : whether the loan application is approved or not

Data preprocessing:

Data preprocessing involves transforming the unprocessed data into a refined data set. The dataset undergoes preprocessing to identify missing values, noisy data, and other inconsistencies before being fed into the algorithm. Data needs to be structured properly for machine learning.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings("ignore")

df =pd.read_csv("loan_approval_dataset.csv")
```

In [2]:
```
df.head()
```

Out[2]:

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_valu
0	1	2	Graduate	No	9600000	29900000	12	778	2400000	17600000
1	2	0	Not Graduate	Yes	4100000	12200000	8	417	2700000	2200000
2	3	3	Graduate	No	9100000	29700000	20	506	7100000	4500000
3	4	3	Graduate	No	8200000	30700000	8	467	18200000	3300000
4	5	5	Not Graduate	Yes	9800000	24200000	20	382	12400000	8200000

- Importing required libraries like pandas, numpy, seaborn, matplotlib.pyplot.
- warnings.filterwarnings("ignore") ignores any warnings that might be generated whie executing th code.
- df =pd.read_csv("loan_approval_dataset.csv") reads the csv file.

In [3]:
```
df.describe(include='all')
```

Out[3]:

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercia
count	4269.000000	4269.000000	4269	4269	4.269000e+03	4.269000e+03	4269.000000	4269.000000	4.269000e+03	
unique	NaN	NaN	2	2	NaN	NaN	NaN	NaN	NaN	
top	NaN	NaN	Graduate	Yes	NaN	NaN	NaN	NaN	NaN	
freq	NaN	NaN	2144	2150	NaN	NaN	NaN	NaN	NaN	
mean	2135.000000	2.498712	NaN	NaN	5.059124e+06	1.513345e+07	10.900445	599.936051	7.472617e+06	
std	1232.498479	1.695910	NaN	NaN	2.806840e+06	9.043363e+06	5.709187	172.430401	6.503637e+06	
min	1.000000	0.000000	NaN	NaN	2.000000e+05	3.000000e+05	2.000000	300.000000	-1.000000e+05	
25%	1068.000000	1.000000	NaN	NaN	2.700000e+06	7.700000e+06	6.000000	453.000000	2.200000e+06	
50%	2135.000000	3.000000	NaN	NaN	5.100000e+06	1.450000e+07	10.000000	600.000000	5.600000e+06	
75%	3202.000000	4.000000	NaN	NaN	7.500000e+06	2.150000e+07	16.000000	748.000000	1.130000e+07	
max	4269.000000	5.000000	NaN	NaN	9.900000e+06	3.950000e+07	20.000000	900.000000	2.910000e+07	

- df.describe(include='all') is used to get both numerical and non-numerical summary of the dataset
- we can see the number of non null, unique values and the most frequent value along with the frequency of the most frequent value.
- we can also see the mean, standard deviation, minimum and maximun value, 25% 50% and 75% quartile.

```
In [4]: print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4269 entries, 0 to 4268
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   loan_id                   4269 non-null   int64
 1    no_of_dependents         4269 non-null   int64
 2    education                4269 non-null   object
 3    self_employed            4269 non-null   object
 4    income_annum             4269 non-null   int64
 5    loan_amount              4269 non-null   int64
 6    loan_term                4269 non-null   int64
 7    cibil_score              4269 non-null   int64
 8    residential_assets_value 4269 non-null   int64
 9    commercial_assets_value  4269 non-null   int64
 10   luxury_assets_value      4269 non-null   int64
 11   bank_asset_value         4269 non-null   int64
 12   loan_status              4269 non-null   object
dtypes: int64(10), object(3)
memory usage: 433.7+ KB
None
```

- df.info() is used to get the structure of the dataset.
- There are NO non null values in the dataset.
- out of the 13 columns, 10 columns have int64 datatype and 3 columns have object datatype.
- the total memory used by the dataset is 433.7+ KB.

```
In [5]: df1 = df.drop(['loan_id'],axis=1)

        df1.columns = df1.columns.str.replace(' ','')

        df1.head()
```

Out[5]:

	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury
0	2	Graduate	No	9600000	29900000	12	778	2400000	17600000	
1	0	Not Graduate	Yes	4100000	12200000	8	417	2700000	2200000	
2	3	Graduate	No	9100000	29700000	20	506	7100000	4500000	
3	3	Graduate	No	8200000	30700000	8	467	18200000	3300000	
4	5	Not Graduate	Yes	9800000	24200000	20	382	12400000	8200000	

- Creating a new dataframe and dropping the column 'loan_id.
- Loan id is just used to identity the entity so it can be omitted whilen analyzing the data and training the model.
- Removing the unwanted spaces in the names of the columns.
- Loan id is just used to identity the entity so it can be omitted whilen analyzing the data and training the model

```
In [6]: df1.isnull().sum()

Out[6]: no_of_dependents           0
        education                  0
        self_employed              0
        income_annum               0
        loan_amount                0
        loan_term                  0
        cibil_score                0
        residential_assets_value   0
        commercial_assets_value    0
        luxury_assets_value        0
        bank_asset_value           0
        loan_status                0
        dtype: int64

In [7]: df1.dtypes

Out[7]: no_of_dependents            int64
        education                  object
        self_employed              object
        income_annum                int64
        loan_amount                 int64
        loan_term                   int64
        cibil_score                 int64
        residential_assets_value    int64
        commercial_assets_value     int64
        luxury_assets_value         int64
        bank_asset_value            int64
        loan_status                object
        dtype: object

In [8]: df1.shape

Out[8]: (4269, 12)
```

- Using df1.isnull().sum() we can see there are no null values in dataframe df1
- df1.dtypes gives us the datatypes of the columns. We have 9 int64 and 3 object datatypes.
- df1.shape shows us that 4269 rows and 12 columns.

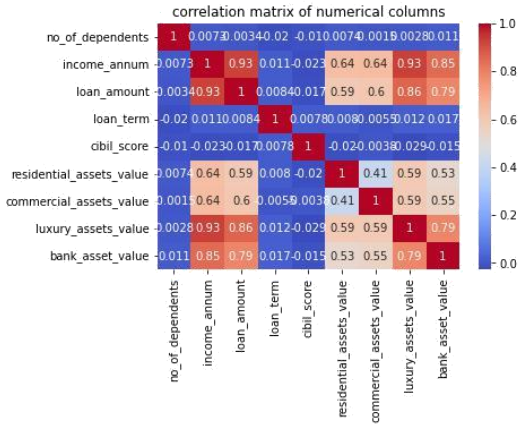
Analysis of Data (EDA):

Exploratory data analysis (EDA) is a technique used to analyze data sets in order to discover patterns, detect anomalies, and comprehend the connections between variables. Exploratory Data Analysis (EDA) is commonly the initial phase in data analysis and is typically performed prior to formal modeling.

Correlation matrix between numerical features

```
In [9]: plt.figure()
        sns.heatmap(df1.select_dtypes(include=['number']).corr(),annot=True,cmap='coolwarm')
        plt.title('correlation matrix of numerical columns')
```

```
Out[9]: Text(0.5, 1.0, 'correlation matrix of numerical columns')
```

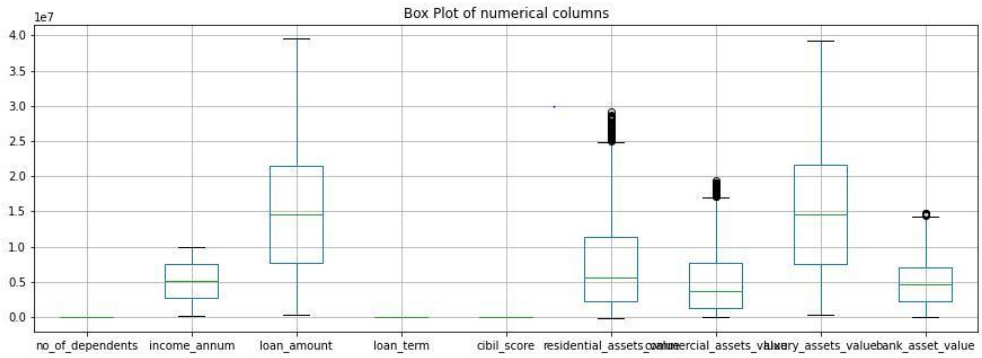


- Generating a heatmap to display the correlation matrix of numeric columns in the dataframe.
- df1.select_dtypes(include=['number'] is used to select only numerical columns in the dataframe.

Observations:

- loan_amount is positively correlated with income_annum, suggesting that higher income applicants tend to request larger loans.
- Asset values show moderate correlations with each other, indicating that applicants who own one type of asset often owns others as well.
- cibil_score does not show a strong correlation with most variables, which might suggest that other factors are more significant in determining the loan approval decision.
- Asset values have moderate to strong relationship with annual income. People with high income may be able to afford luxury assets.

```
In [10]: plt.figure(figsize=(15,5))
         df1.select_dtypes(include=['number']).boxplot()
         plt.title('Box Plot of numerical columns')
         plt.show()
```

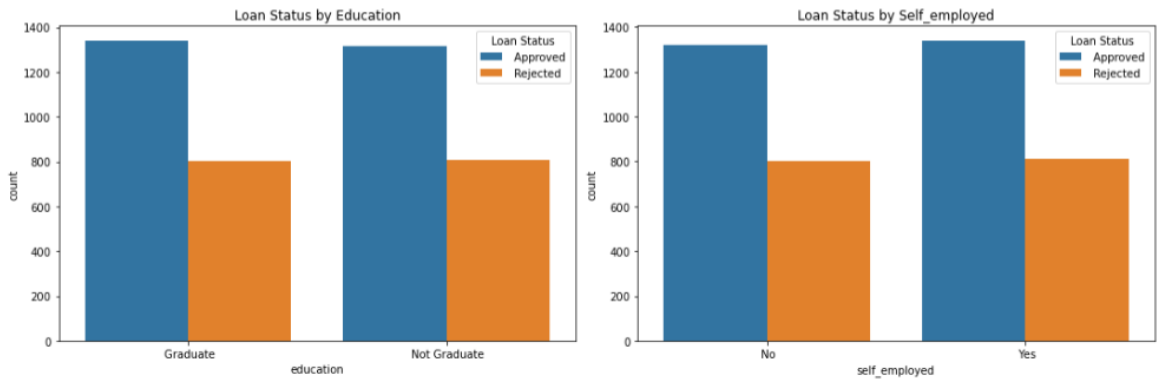


- Generates a boxplot to display the correlation matrix of numeric columns in the dataframe.
- df1.select_dtypes(include=['number'] is used to select only numerical columns in the dataframe.

Observations:

From the box plots residential assets, commercial assets and bank asset values has outlier.

```
In [11]: categorical_columns = ['education', 'self_employed', 'loan_status']

         plt.figure(figsize=(15, 5))

         for i, column in enumerate(categorical_columns[:-1], 1):
             plt.subplot(1, 2, i)
             sns.countplot(x=column, hue='loan_status', data=df1)
             plt.title(f'Loan Status by {column.capitalize()}')
             plt.legend(title='Loan Status', loc='upper right')

         plt.tight_layout()
         plt.show()
```

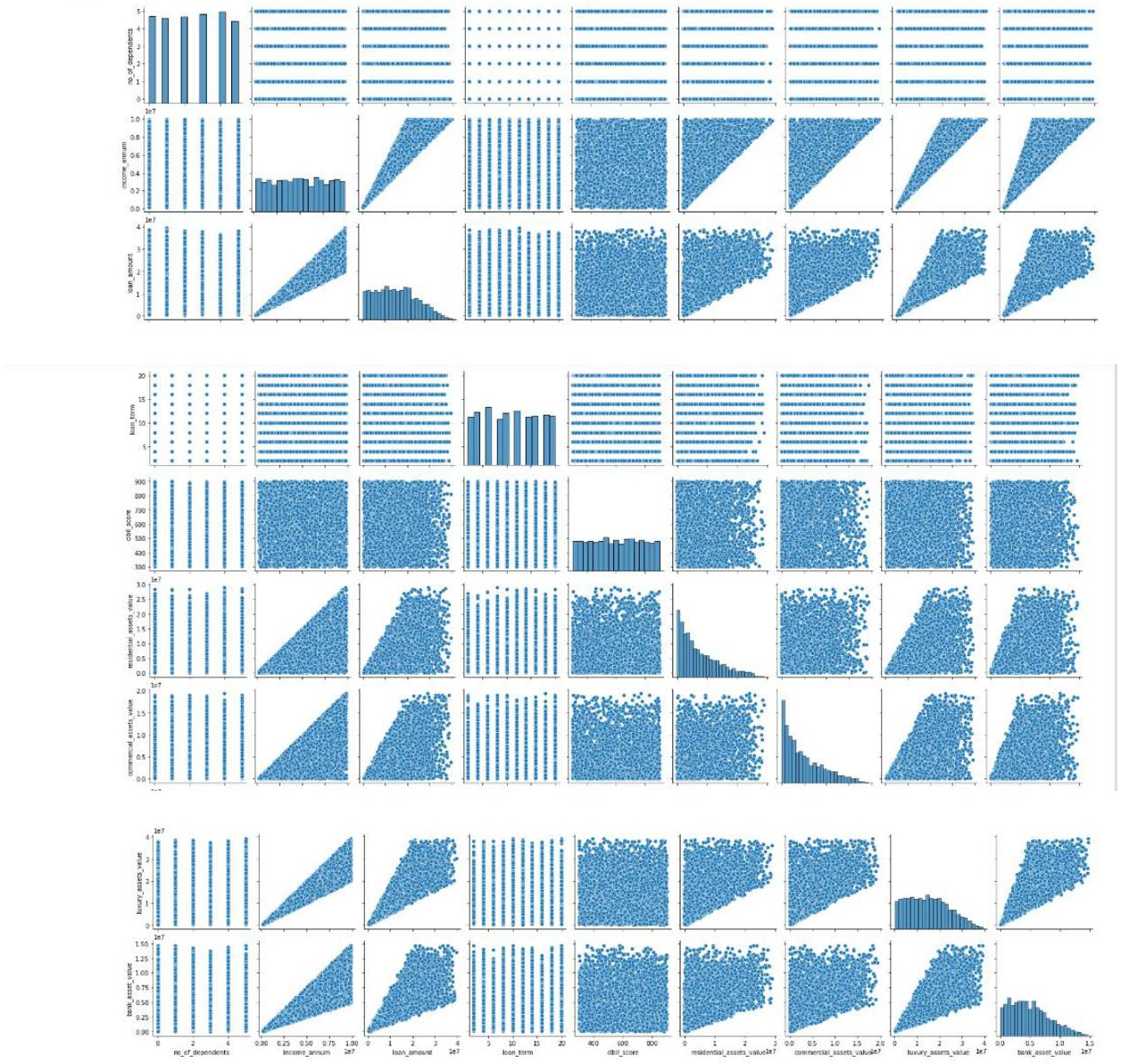


- Creating a countplot to visualize how the categorial columns are linked to the loan status.
- Subplots are created for the categorial columns education and self-employed.

Observations:

There is no strong relationship between the education and employement for loan approvals. There might be other factors such as assets, annual income and cibil score which may play important role in loan approvals.

```
In [12]: sns.pairplot(df1)
Out[12]: <seaborn.axisgrid.PairGrid at 0x1ce164d7520>
```

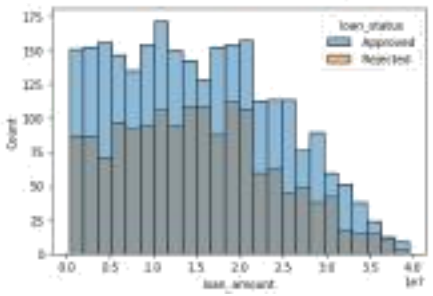


- sns.pairplot(df1) is used to represent connections among numerous numerical variables in df1.

Observations:

loan_amount, income_annum, luxury_asset_value, bank_asset_value,income_annum has positive correlation with other variables.

```
In [13]: sns.histplot(df1,x='loan_amount',hue='loan_status')
Out[13]: <Axes:xlabel='loan_amount', ylabel='Count'>
```



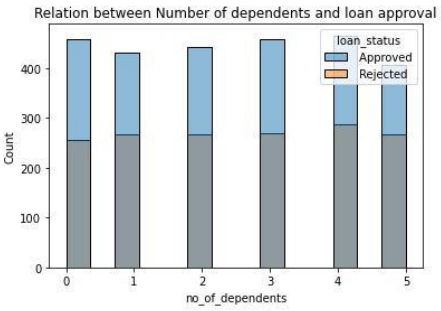
- Generating a histogram to display how loan amounts are spread out in df1, separating the information based on loan status.
- This shows how the loan amounts and status of the loan are related.

Observations:

Same trend is going on between Approved loans and rejected loans. Seems like there is no strong relationship between the loan status and loan amount from the graph.

Relation between loan approval and number of dependents

```
In [14]: sns.histplot(data=df1,x='no_of_dependents',hue='loan_status')
         plt.title('Relation between Number of dependents and loan approval')
         plt.show()
```

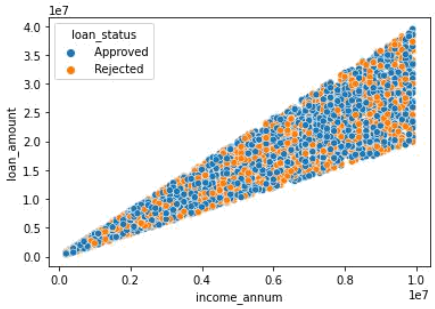


- Generating a histogram to display how the number of dependents impacts the approval status of loans.
- This shows how the number of dependents and status of the loan are related.

Observations:

Number of dependents does not have any signficance on the loan approval rate.

```
In [15]: sns.scatterplot(data = df1, x='income_annum',y='loan_amount',hue='loan_status')
Out[15]: <AxesSubplot:xlabel='income_annum', ylabel='loan_amount'>
```



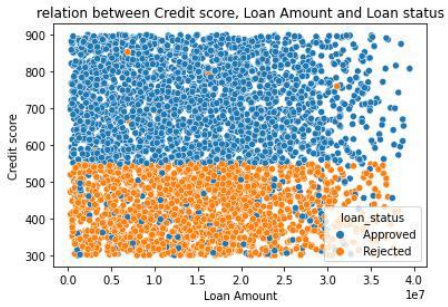
- Generating a scatter plot to display the correlation between annual income and loan amount, categorizing by loan approval.

Observations:

loan_amount increses with the annual_income, still loan was rejected for applicants with higher annual income. Other factors such as CIBIL score also plays a role in loan_status. Applicants with lower income will have a small range of loan amounts and applicants with higher income will have wider range of loan amount. This shows that ability to pay back loan which is dependent on annual income.

Relation between credit_score, loan_amount and loan_status

```
In [16]: sns.scatterplot(data=df1,x='loan_amount',y='cibil_score',hue='loan_status')
         plt.title('relation between Credit score, Loan Amount and Loan status')
         plt.ylabel('Credit score')
         plt.xlabel('Loan Amount')
Out[16]: Text(0.5, 0, 'Loan Amount')
```



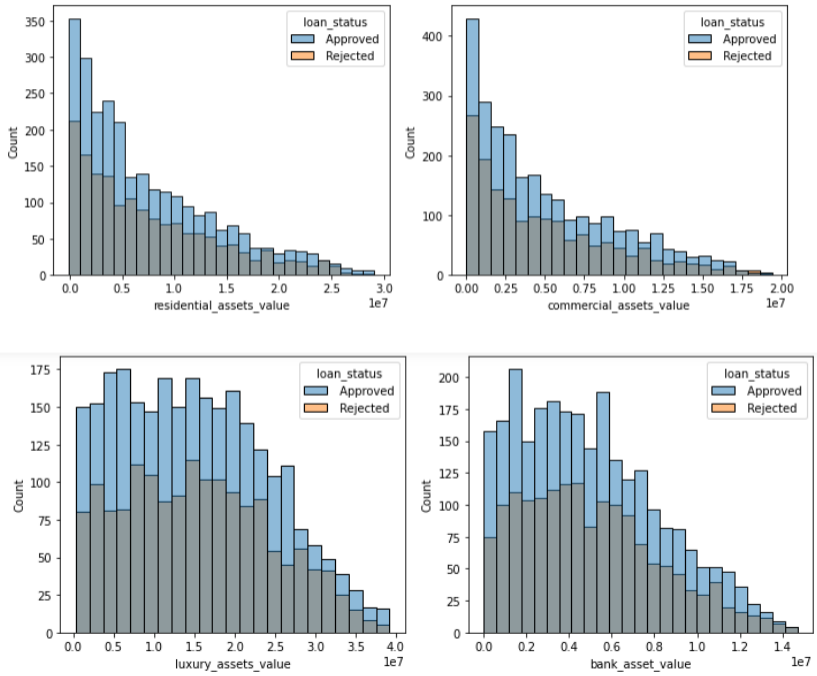
- Generating scatterplot to display the relationship between loan amount, credit score, and loan status.

Observations:

- credit score from 550 seperated loan status into two parts. Loan status is highly correlated with credit score. With credit scores above 550 has a good chance loan approval.

- In some cases credit score below 550 got approved for loan and credit score greater than 550 also got rejected.

```
In [17]: fig,ax=plt.subplots(2,2,figsize=(10,8))
sns.histplot(data=df1, x = 'residential_assets_value', hue = 'loan_status',ax=ax[0,0])
sns.histplot(data=df1, x = 'commercial_assets_value', hue = 'loan_status',ax=ax[0,1])
sns.histplot(data=df1, x = 'luxury_assets_value', hue = 'loan_status',ax=ax[1,0])
sns.histplot(data=df1, x = 'bank_asset_value', hue = 'loan_status',ax=ax[1,1])

plt.tight_layout()
plt.show()
```

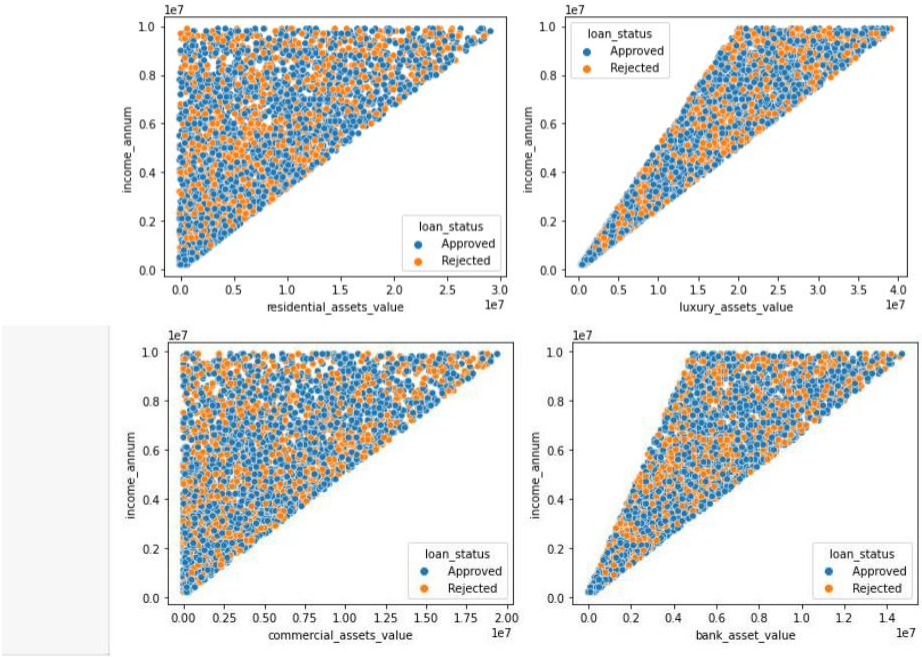


Observations:

From the plots we can see there are no clear trends between the assets and loan approval.

Relationship between assets and income

```
In [18]: fig,ax=plt.subplots(2,2,figsize=(10,8))
sns.scatterplot(data=df1, x = 'residential_assets_value', y= 'income_annum', hue = 'loan_status', ax = ax[0,0])
sns.scatterplot(data=df1, x = 'commercial_assets_value', y= 'income_annum', hue = 'loan_status',ax = ax[1,0])
sns.scatterplot(data=df1, x = 'luxury_assets_value', y= 'income_annum', hue = 'loan_status',ax = ax[0,1])
sns.scatterplot(data=df1, x = 'bank_asset_value', y= 'income_annum', hue = 'loan_status',ax = ax[1,1])

plt.tight_layout()
plt.show()
```



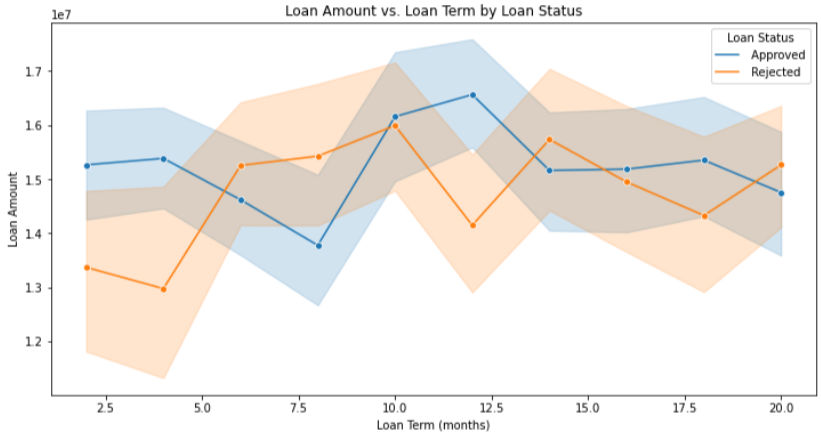
- Generating a figure with multiple histograms to display how the different assets (residential, luxury, commercial, bank) are related to the loan approval status.

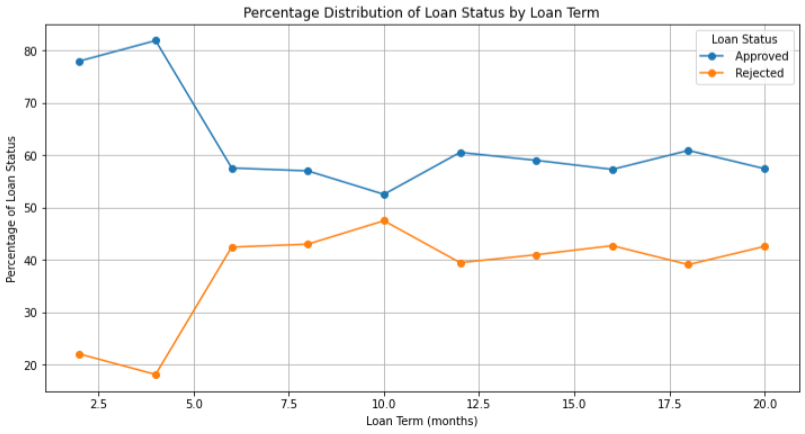
Observations:

- commercial assets divided the plot into two sections which indicates relation between annual income. When asset reaches a certain values then annucal income increased significantly
- The shape of other assets show different kind of relation between the annual income. luxury assets shows a good indicator for higher income.

Loan Term and Loan status

```
In [19]: plt.figure(figsize=(12, 6))
sns.lineplot(data=df1, x='loan_term', y='loan_amount', hue='loan_status', marker='o')

plt.title('Loan Amount vs. Loan Term by Loan Status')
plt.xlabel('Loan Term (months)')
plt.ylabel('Loan Amount')
plt.legend(title='Loan Status')

plt.show()

loan_term_status = df1.groupby(['loan_term', 'loan_status']).size().unstack().fillna(0)
loan_term_status_percentage = loan_term_status.div(loan_term_status.sum(axis=1), axis=0) * 100

# Plotting the percentage distribution of loan statuses against loan term
plt.figure(figsize=(12, 6))
loan_term_status_percentage.plot(kind='line', marker='o', ax=plt.gca())

# Adding labels and title to the plot
plt.title('Percentage Distribution of Loan Status by Loan Term')
plt.xlabel('Loan Term (months)')
plt.ylabel('Percentage of Loan Status')
plt.legend(title='Loan Status')
plt.grid(True)
```

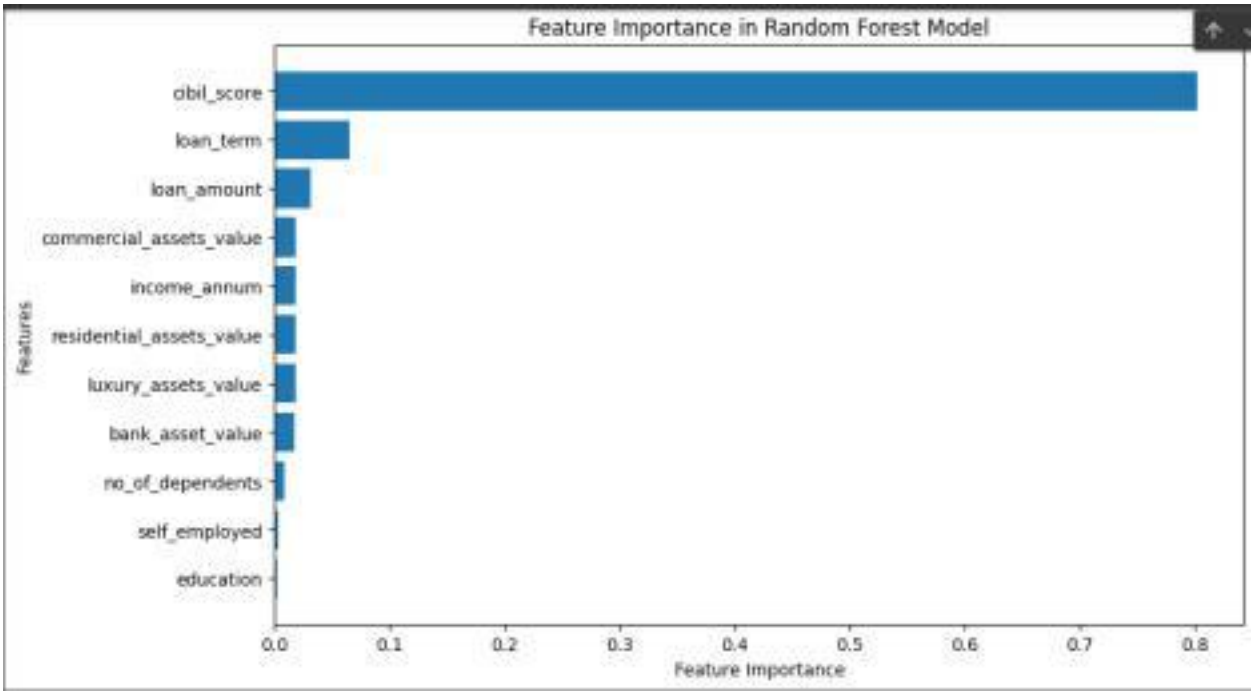




- Generating a line plot to display the relationship between loan amount and loan term catogorized by the status of the loan.
- Also generating a percentage of loan statuses against loan term.

Feature Importance:

- The rf_model.feature_importances_ contains all the important scores for each and every feature.
- By sorting, re-ordering the features, plotting them using matplot.
- Here can be seen that, **cibil_score** plays the main role in loan approvals and rejections.



Model Training:

Training a model involves instructing a machine learning algorithm to recognize patterns and make predictions based on available data. The objective is to develop a computer program, or model, that can effectively answer questions and make choices using fresh information.

In [20]:
```
from sklearn.preprocessing import LabelEncoder

df1['education'] = LabelEncoder().fit_transform(df1['education'])
df1['self_employed']= LabelEncoder().fit_transform(df1['self_employed'])
df1['loan_status']=LabelEncoder().fit_transform(df1['loan_status'])
```

In [21]:
```
print(df1[['education','self_employed','loan_status']])
```

```
      education  self_employed  loan_status
0             0              0            0
1             1              1            1
2             0              0            1
3             0              0            1
4             1              1            1
...         ...            ...          ...
4264          0              1            1
4265          1              1            0
4266          1              0            1
4267          1              0            0
4268          0              0            0

[4269 rows x 3 columns]
```

In [22]:
```
df1.dtypes
```

Out[22]:
```
no_of_dependents            int64
education                   int32
self_employed               int32
income_annum                int64
loan_amount                 int64
loan_term                   int64
cibil_score                 int64
residential_assets_value    int64
commercial_assets_value     int64
luxury_assets_value         int64
bank_asset_value            int64
loan_status                 int32
dtype: object
```

- LabelEncoder is imported to convert categorial values to numerical values.
- Columns like 'education', 'self_employed' and 'loan_status' is mapped to an unique integer.

In [23]:
```
from sklearn.preprocessing import StandardScaler

x = df1.drop(columns=['loan_status'])
y = df1['loan_status']

numerical_columns = ['no_of_dependents', 'income_annum', 'loan_amount', 'loan_term', 'cibil_score',
                     'residential_assets_value', 'commercial_assets_value', 'luxury_assets_value',
                     'bank_asset_value']

x[numerical_columns] = StandardScaler().fit_transform(x[numerical_columns])
print("Scaled Feature : \n",x.head())

print("Target Feature : \n",y.head())
```

```
Scaled Feature :
    no_of_dependents  education  self_employed  income_annum  loan_amount  \
0         -0.294102          0              0      1.617979     1.633052
1         -1.473548          1              1     -0.341750    -0.324414
2          0.295621          0              0      1.439822     1.610933
3          0.295621          0              0      1.119139     1.721525
4          1.475067          1              1      1.689242     1.002681

   loan_term  cibil_score  residential_assets_value  commercial_assets_value  \
0   0.192617     1.032792                 -0.780058                 2.877289
1  -0.508091    -1.061051                 -0.733924                -0.631921
2   1.594031    -0.544840                 -0.057300                -0.107818
3  -0.508091    -0.771045                  1.649637                -0.381263
4   1.594031    -1.264055                  0.757724                 0.735304

   luxury_assets_value  bank_asset_value
0             0.832028          0.930304
1            -0.694993         -0.515936
2             1.996520          2.407316
3             0.897943          0.899533
4             1.568075          0.007172
Target Feature :
 0    0
1    1
2    1
3    1
4    1
Name: loan_status, dtype: int32
```

- Feature scaling is performed to standardize numerical columns to make sure they have an average of 0 and a standard deviation of 1.

In [24]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3 ,random_state = 42)
```

- Dividing the dataset into training dataset and test dataset.

K-Nearest Neighbors (KNN)

A supervised learning classifier uses proximity to make classifications or predictions the grouping of a specific data point.

```
In [25]: from sklearn.neighbors import KNeighborsClassifier

         knn_classifier = KNeighborsClassifier()
         knn_classifier.fit(x_train,y_train)

         y_pred = knn_classifier.predict(x_test)

         accuracy = accuracy_score(y_test,y_pred)

         print(accuracy)

         0.8930523028883685
```

- KNeighborsClassifier is imported and the classifier is fitted on the training data.
- KNN classifier finds KNN for each test data and makes predictions.
- The accuracy of the model is calculated (0.8930523028883685).

Cross validation score:

Cross-validation score represents how well a machine learning model performs statistically. Cross-validation is a process in which a dataset is divided into parts for training and testing, before assessing the model's performance on each part. The amount of cross-validation scores conducted equals the number of data points in the dataset.

```
In [49]: from sklearn.model_selection import cross_val_score

         scores = cross_val_score(knn_classifier, x_train, y_train, cv = 10)

         print('Cross-validation scores:{}'.format(scores))

         print('Average cross-validation score: {:.4f}'.format(scores.mean()))

         Cross-validation scores:[0.89632107 0.86956522 0.91638796 0.94314381 0.909699   0.88628763
          0.909699   0.90635452 0.94295302 0.9295302 ]
         Average cross-validation score: 0.9110
```

- cross_val_score is imported.
- Performing 10-fold cross validation technique to improve model performance.
- Summmarizing the validation by calculating its mean.
- We expect the model to be around 91.10 % accurate on average.

In [69]:
```
from sklearn.preprocessing import StandardScaler

x = df1.drop(columns=['loan_status'])
y = df1['loan_status']

numerical_columns = ['no_of_dependents', 'income_annum', 'loan_amount', 'loan_term', 'cibil_score',
                     'residential_assets_value', 'commercial_assets_value', 'luxury_assets_value',
                     'bank_asset_value']

x[numerical_columns] = StandardScaler().fit_transform(x[numerical_columns])
print("Scaled Feature : \n",x.head())

print("Target Feature : \n",y.head())

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3 ,random_state = 42)
```

```
Scaled Feature :
    no_of_dependents  education  self_employed  income_annum  loan_amount  \
0         -0.294102          0              0      1.617979     1.633052
1         -1.473548          1              1     -0.341750    -0.324414
2          0.295621          0              0      1.439822     1.610933
3          0.295621          0              0      1.119139     1.721525
4          1.475067          1              1      1.689242     1.002681

   loan_term  cibil_score  residential_assets_value  commercial_assets_value  \
0   0.192617     1.032792                 -0.780058                 2.877289
1  -0.508091    -1.061051                 -0.733924                -0.631921
2   1.594031    -0.544840                 -0.057300                -0.107818
3  -0.508091    -0.771045                  1.649637                -0.381263
4   1.594031    -1.264055                  0.757724                 0.735304

   luxury_assets_value  bank_asset_value
0             0.832028          0.930304
1            -0.694993         -0.515936
2             1.996520          2.407316
3             0.897943          0.899533
4             1.568075          0.007172
Target Feature :
 0    0
1    1
2    1
3    1
4    1
Name: loan_status, dtype: int32
```

- The dataset is split into features and target.
- StandardScaler is applied to the numerical columns.
- Splitting 70% of the data for training and 30% of the data for testing

KNeighborsRegressor:

A regressor based on KNN that estimates the target variable's value by averaging its closest neighbors' values.

In [70]:
```
import sklearn.metrics
import sklearn.neighbors
from sklearn.neighbors import KNeighborsRegressor

k_values = [1, 3, 5]
weights = ['uniform', 'distance']
metrics = ['euclidean', 'manhattan', 'minkowski']
for k in k_values:
    for weight in weights:
        for metric in metrics:
            regressor = KNeighborsRegressor(n_neighbors=k, weights=weight, metric=metric)
            regressor.fit(x_train, y_train)
            yhats = regressor.predict(x_test)
            error = sklearn.metrics.mean_squared_error(yhats, y_test, squared=False)
            print(f"k={k}, weight={weight}, metric={metric} MSE :  {error:.2f}")
```

```
k=1, weight=uniform, metric=euclidean MSE :  0.37
k=1, weight=uniform, metric=manhattan MSE :  0.39
k=1, weight=uniform, metric=minkowski MSE :  0.37
k=1, weight=distance, metric=euclidean MSE :  0.37
k=1, weight=distance, metric=manhattan MSE :  0.39
k=1, weight=distance, metric=minkowski MSE :  0.37
k=3, weight=uniform, metric=euclidean MSE :  0.30
k=3, weight=uniform, metric=manhattan MSE :  0.29
k=3, weight=uniform, metric=minkowski MSE :  0.30
k=3, weight=distance, metric=euclidean MSE :  0.30
k=3, weight=distance, metric=manhattan MSE :  0.29
k=3, weight=distance, metric=minkowski MSE :  0.30
k=5, weight=uniform, metric=euclidean MSE :  0.28
k=5, weight=uniform, metric=manhattan MSE :  0.28
k=5, weight=uniform, metric=minkowski MSE :  0.28
k=5, weight=distance, metric=euclidean MSE :  0.28
k=5, weight=distance, metric=manhattan MSE :  0.28
k=5, weight=distance, metric=minkowski MSE :  0.28
```

- KNeighborsRegressor is imported an fitted to training data.
- KNN Regressor predicts the mean square errors based on the k values, weights, and metrics.
- The used metrics are euclidean, manhattan, and minkowski.

In [83]:
```
from sklearn.preprocessing import StandardScaler

x = df1.drop(columns=['loan_status','no_of_dependents','commercial_assets_value','luxury_assets_value'])
y = df1['loan_status']

numerical_columns = ['income_annum', 'loan_amount', 'loan_term', 'cibil_score','residential_assets_value','bank_asset_value']

x[numerical_columns] = StandardScaler().fit_transform(x[numerical_columns])
print("Scaled Feature : \n",x.head())

print("Target Feature : \n",y.head())

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3 ,random_state = 42)
```

```
Scaled Feature :
    education  self_employed  income_annum  loan_amount  loan_term  \
0          0              0      1.617979     1.633052   0.192617
1          1              1     -0.341750    -0.324414  -0.508091
2          0              0      1.439822     1.610933   1.594031
3          0              0      1.119139     1.721525  -0.508091
4          1              1      1.689242     1.002681   1.594031

   cibil_score  residential_assets_value  bank_asset_value
0     1.032792                 -0.780058          0.930304
1    -1.061051                 -0.733924         -0.515936
2    -0.544840                 -0.057300          2.407316
3    -0.771045                  1.649637          0.899533
4    -1.264055                  0.757724          0.007172
Target Feature :
 0    0
1    1
2    1
3    1
4    1
Name: loan_status, dtype: int32
```

- some of the numerical columns such as no_of_dependents, commercial_assets_value and luxury_asset_values are removed and dataset is normalized using standardscaler() function

In [89]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

# Hyperparameter tuning for KNN
k_values = [1, 3, 5]
weights = ['uniform', 'distance']
metrics = ['euclidean', 'manhattan', 'minkowski']

for k in k_values:
    for weight in weights:
        for metric in metrics:
            classifier = KNeighborsClassifier(n_neighbors=k, weights=weight, metric=metric)
            classifier.fit(x_train, y_train)

            y_pred = classifier.predict(x_test)

            accuracy = accuracy_score(y_test, y_pred)
            print(f"k : {k}, weight : {weight}, metric : {metric}  Accuracy : {accuracy:.2f}")
```

```
k : 1, weight : uniform, metric : euclidean  Accuracy : 0.89
k : 1, weight : uniform, metric : manhattan  Accuracy : 0.89
k : 1, weight : uniform, metric : minkowski  Accuracy : 0.89
k : 1, weight : distance, metric : euclidean  Accuracy : 0.89
k : 1, weight : distance, metric : manhattan  Accuracy : 0.89
k : 1, weight : distance, metric : minkowski  Accuracy : 0.89
k : 3, weight : uniform, metric : euclidean  Accuracy : 0.90
k : 3, weight : uniform, metric : manhattan  Accuracy : 0.90
k : 3, weight : uniform, metric : minkowski  Accuracy : 0.90
k : 3, weight : distance, metric : euclidean  Accuracy : 0.90
k : 3, weight : distance, metric : manhattan  Accuracy : 0.90
k : 3, weight : distance, metric : minkowski  Accuracy : 0.90
k : 5, weight : uniform, metric : euclidean  Accuracy : 0.91
k : 5, weight : uniform, metric : manhattan  Accuracy : 0.92
k : 5, weight : uniform, metric : minkowski  Accuracy : 0.91
k : 5, weight : distance, metric : euclidean  Accuracy : 0.91
k : 5, weight : distance, metric : manhattan  Accuracy : 0.92
k : 5, weight : distance, metric : minkowski  Accuracy : 0.91
```

- Hyperparameter Tuning with different values of k with weights and metrics.
- This helps to find the combination that gives the best accuracy.
- The accuracy of model is 1 (which represents clean and accurate data).

Logistic Regression

Logistic regression is a type of supervised machine learning method utilized for classification purposes in which the objective is to forecast the likelihood of whether an observation pertains to a specific category or not.

```
In [26]: from sklearn.linear_model import LogisticRegression

         reg = LogisticRegression(random_state=42)

         reg.fit(x_train,y_train)

         y_pred = reg.predict(x_test)
         accuracy = accuracy_score(y_test,y_pred)

         report = classification_report(y_test, y_pred)

         print(report)
         print(accuracy)
```

```
              precision    recall  f1-score   support

           0       0.93      0.92      0.92       810
           1       0.87      0.87      0.87       471

    accuracy                           0.90      1281
   macro avg       0.90      0.90      0.90      1281
weighted avg       0.90      0.90      0.90      1281

0.9039812646370023
```

- LogisticRegression is imported and fitted on the training data.
- It learns coefficients to make predictions on the test data.
- The accuracy is calculated (0.9032006245120999).
- The classification report (precision, recall, and F1-score) is obtained.

```
In [79]: from sklearn.model_selection import cross_val_score
         from sklearn.metrics import classification_report, roc_auc_score, roc_curve

         # Logistic Regression model
         logreg = LogisticRegression(max_iter=10000)
         # Logistic Regression Cross-validation
         logreg_cv = cross_val_score(logreg, x, y, cv=5)
         print("Logistic Regression Cross-Validation Scores:", logreg_cv)

         print('Average cross-validation score: {:.4f}'.format(logreg_cv.mean()))
```

```
Logistic Regression Cross-Validation Scores: [0.92388759 0.91920375 0.91451991 0.92388759 0.90504103]
Average cross-validation score: 0.9173
```

- imported classification_report, roc_auc_score, roc_curve
- Maximum iterations are set to perform cross validation scores.
- The mean accurate for 5 fold logistic regression cross validation is about 91.73%

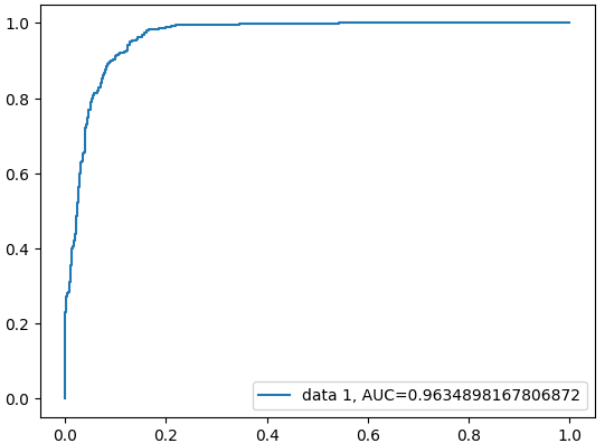
Receiver Operating Characteristic (ROC) Area Under the Curve (AUC) score:

A ROC AUC score is a metric that evaluates a classifier's ability to differentiate between positive and negative classes. It is a solitary value that represents how well the classifier performs at any classification threshold.

Receiver operating characteristic (ROC) curve:

A ROC curve is a plot displaying how a model's true-positive rate and false-positive rate change at varying classification thresholds.

In [80]:
```
from sklearn import metrics
import matplotlib.pyplot as plt

y_pred_proba = reg.predict_proba(x_test)[:, 1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)

plt.plot(fpr, tpr, label="data 1, AUC="+str(auc))
plt.legend(loc=4)
plt.show()
```



- ROC curve is calculated.
- AUC curve is calculated.
- ROC curve is plotted.

Support Vector Machine (SVM)

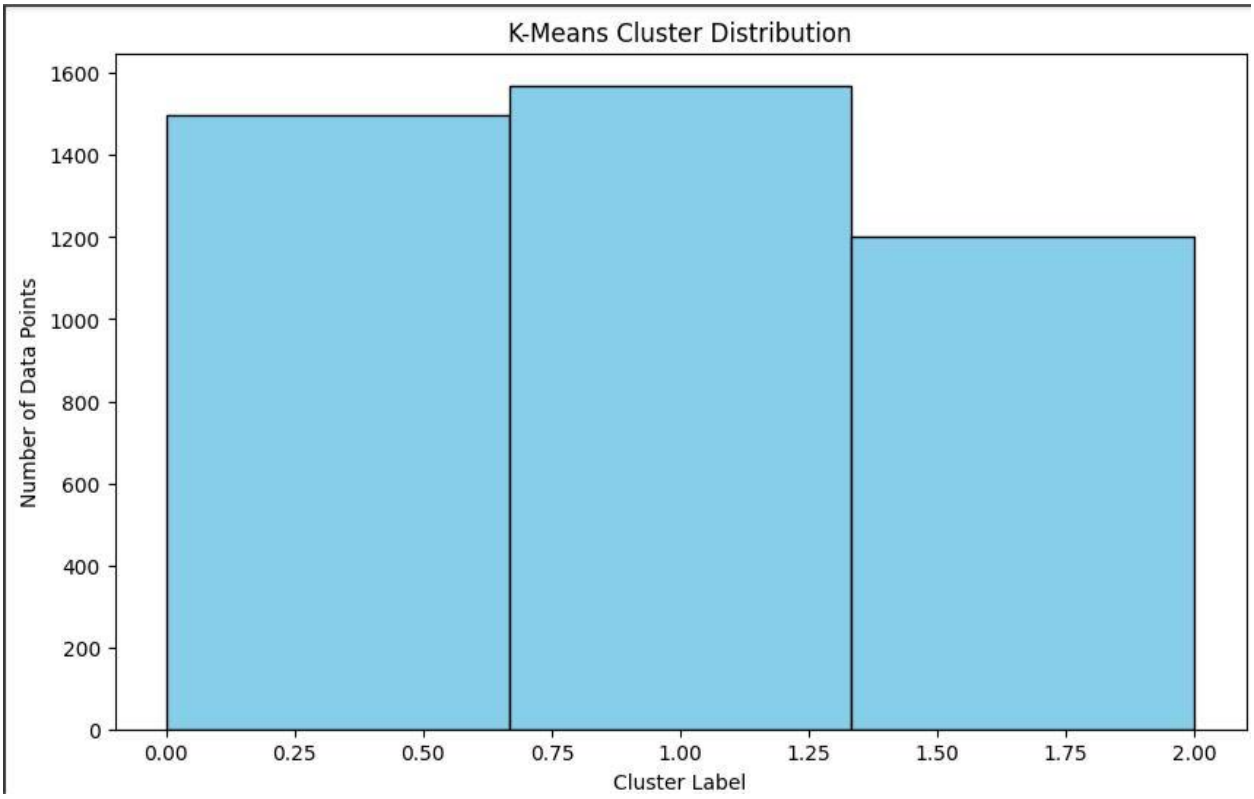
It is a supervised machine learning technique used to discover the best hyperplane in an N-dimensional space that can effectively divide the data points within various classes within the feature space.

```
In [27]: from sklearn.svm import SVC

         svm_classifier = SVC(random_state = 42)
         svm_classifier.fit(x_train,y_train)

         y_pred = svm_classifier.predict(x_test)

         accuracy = accuracy_score(y_test,y_pred)

         print(accuracy)

         0.9266198282591726
```

- SVC is imported and trained on the training data.
- It makes predictions on the testing data by determining on which side of the hyperplane the test data point lies.
- The accuracy of the model is calculated (0.9266198282591726).

K-means Clustering:

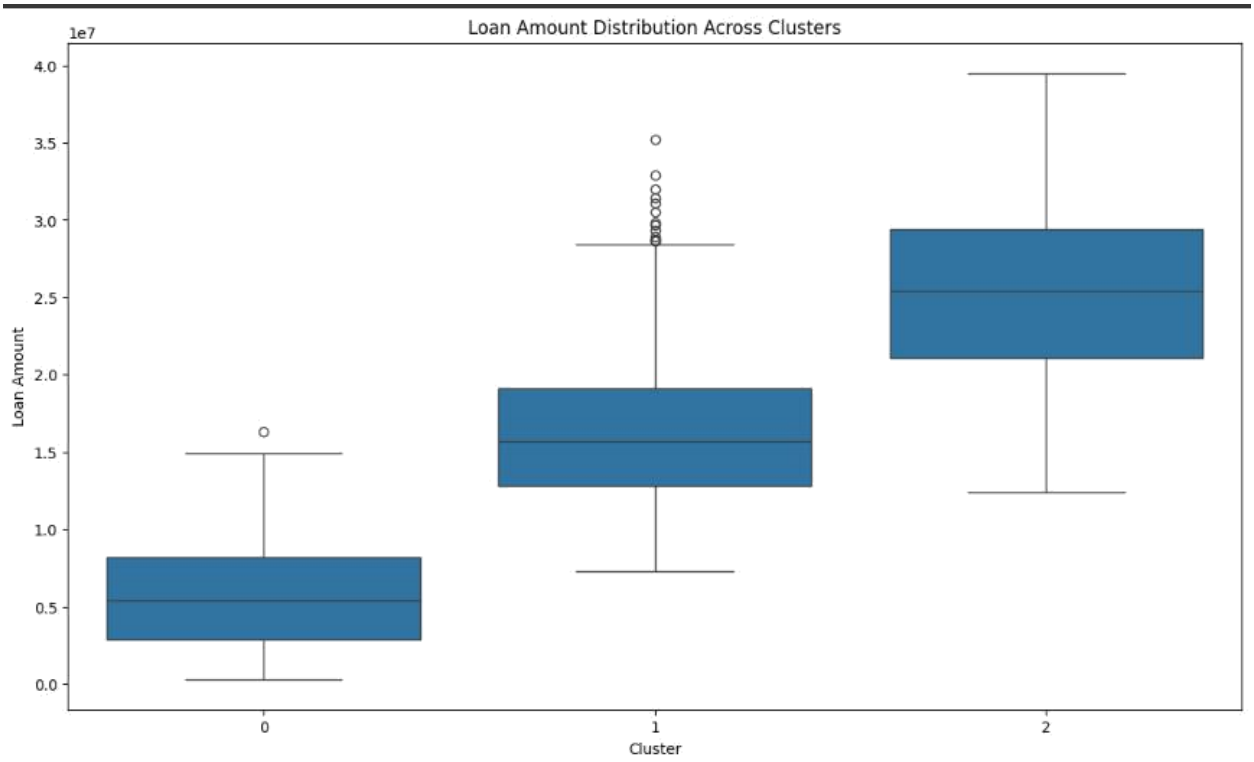
K-means Centroids, or cluster centres, are chosen at random at the beginning of the iterative clustering method. Each data point is then assigned to the closest centroid. By calculating the mean of the data points nearest to each centroid, the algorithm updates the centroids. Until the centroids cease moving, this process is repeated.



Summary of Insights:

- Cluster 0 represents a lower-income group that tends to request smaller loans but enjoys higher approval rates, possibly due to more conservative loan requests.
- Cluster 1 includes a mix of lower-income applicants who may be over-leveraged (as indicated by the higher loan requests), leading to more frequent rejections.
- Cluster 2 consists of higher-income applicants who request larger loans, with a relatively balanced approval rate but also higher risk associated with the larger loan amounts.

Boxplot for Annual Income Distribution:



Loan Amount Distribution Across Clusters:

- The boxplot for loan amounts aligns well with the income levels

observed: Cluster 0:

- This group generally applies for smaller loan amounts, with the median below 1.5 million.
- The lower loan amounts are consistent with the lower income levels seen in this

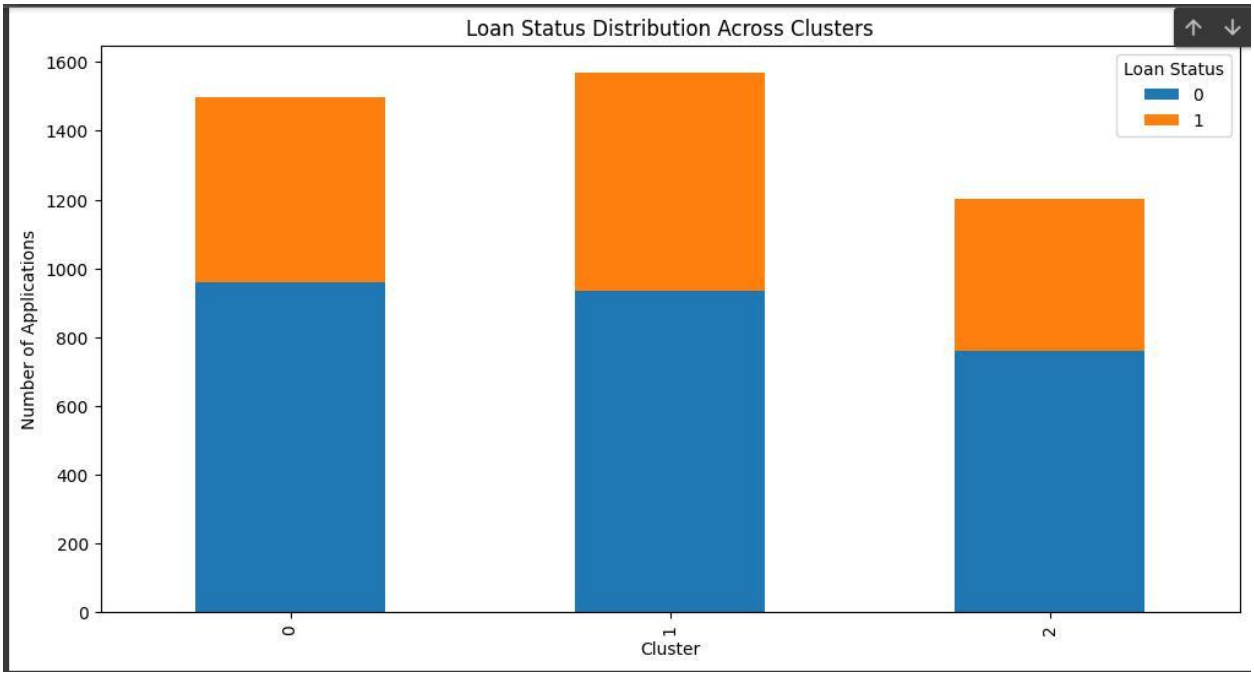
cluster. Cluster 1:

- Applicants in this cluster tend to apply for slightly higher loan amounts compared to Cluster 0.
- The presence of several outliers suggests that some applicants request larger loans despite lower income, which might impact approval rates.

Cluster 2:

- Has the highest loan amounts, with a median above 2 million and values extending up to 4 million.
- The higher loan requests correlate with the higher income levels, indicating that this cluster consists of wealthier applicants seeking larger loans.

Stacked Bar Plot for Loan Status Distribution:



Loan Status Distribution Across Clusters:

- The stacked bar plot shows the distribution of loan approvals and rejections across the clusters:

Cluster 0:

- A higher number of approved loans compared to rejected ones.
- This cluster, despite having lower income, might consist of applicants with stable financial profiles leading to higher approval rates.

Cluster 1:

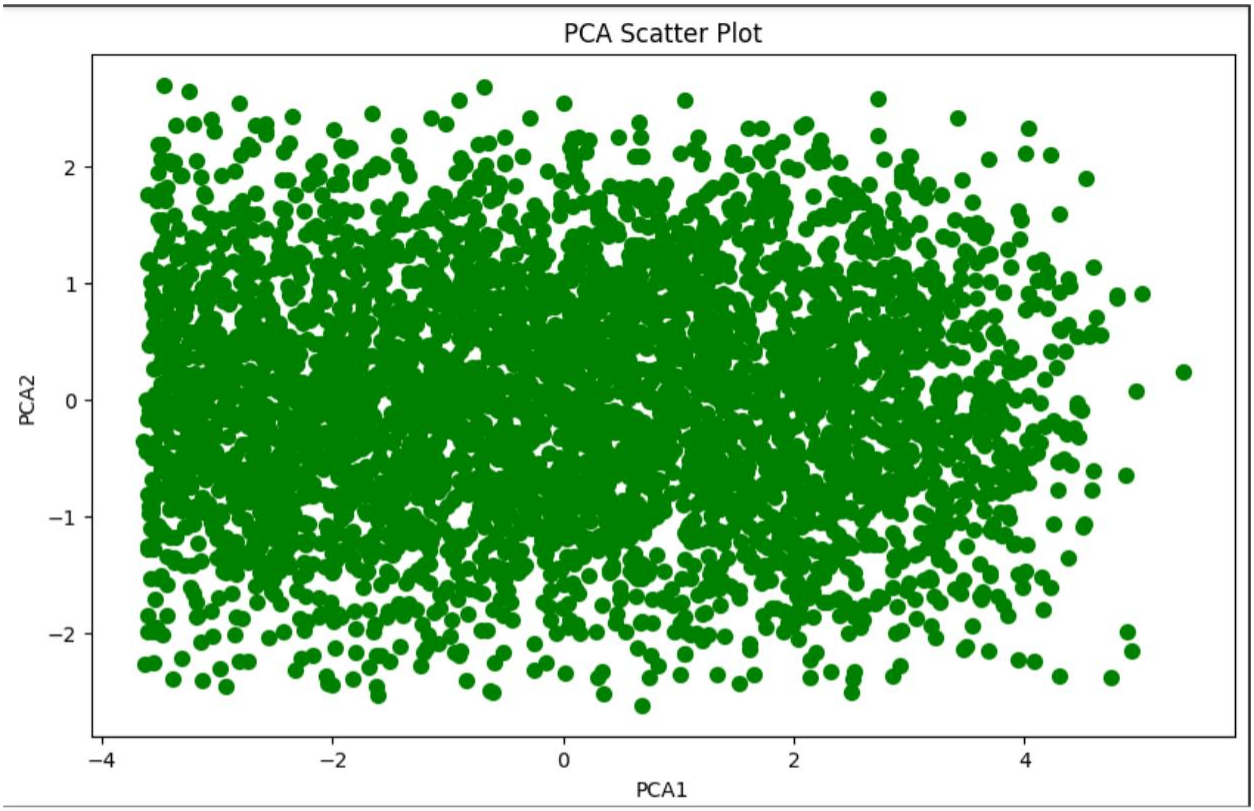
- Has a relatively balanced distribution of approvals and rejections, but slightly more rejections.
- The presence of lower-income applicants requesting higher loans (as seen from the outliers) may contribute to the higher rejection rate.

Cluster 2:

- Shows a better approval rate compared to Cluster 1, but still has a significant number of rejections.
- This is likely due to the high loan amounts requested, which might be riskier for approval despite higher income levels.

Principal Component Analysis (PCA):

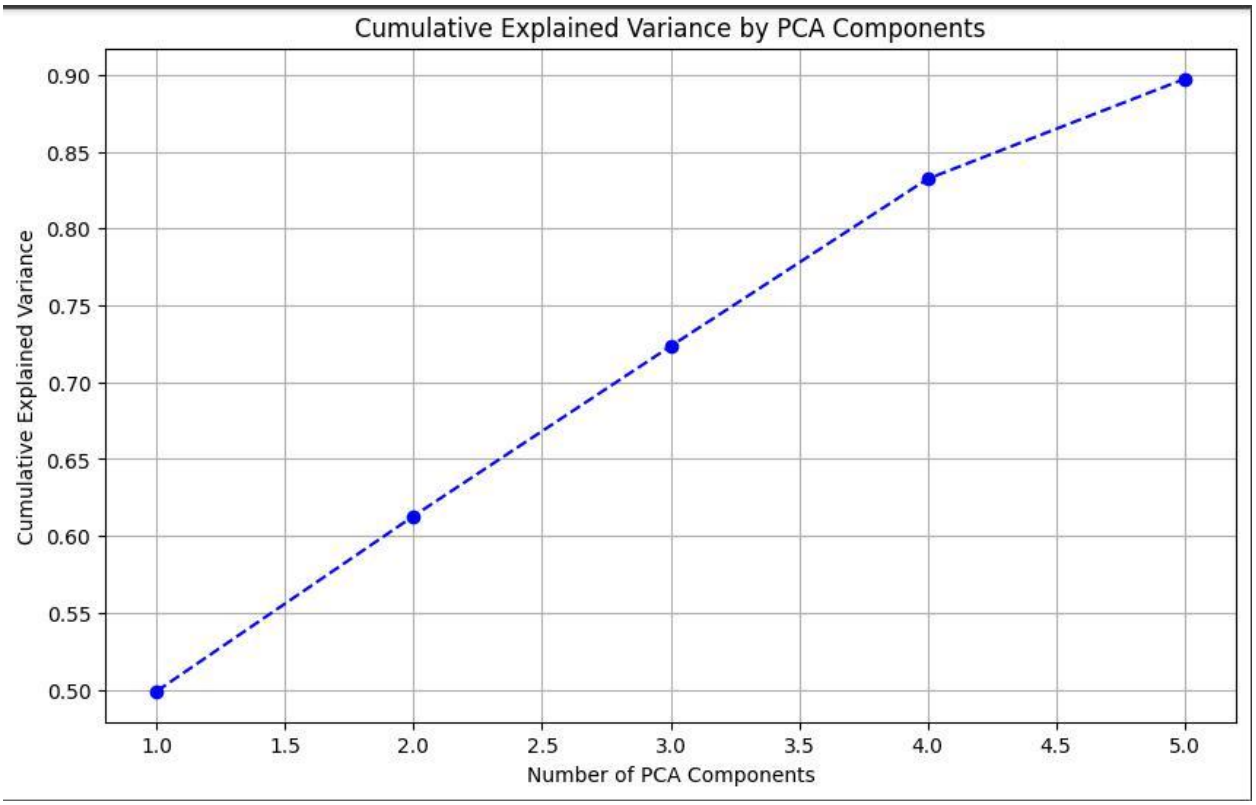
PCA is a popular unsupervised learning method that lowers the dimensionality of big datasets by converting a high-dimensional feature space into a lower-dimensional space while maintaining the most crucial information. This makes it simpler for machine learning algorithms to analyse, visualise, and process the data.

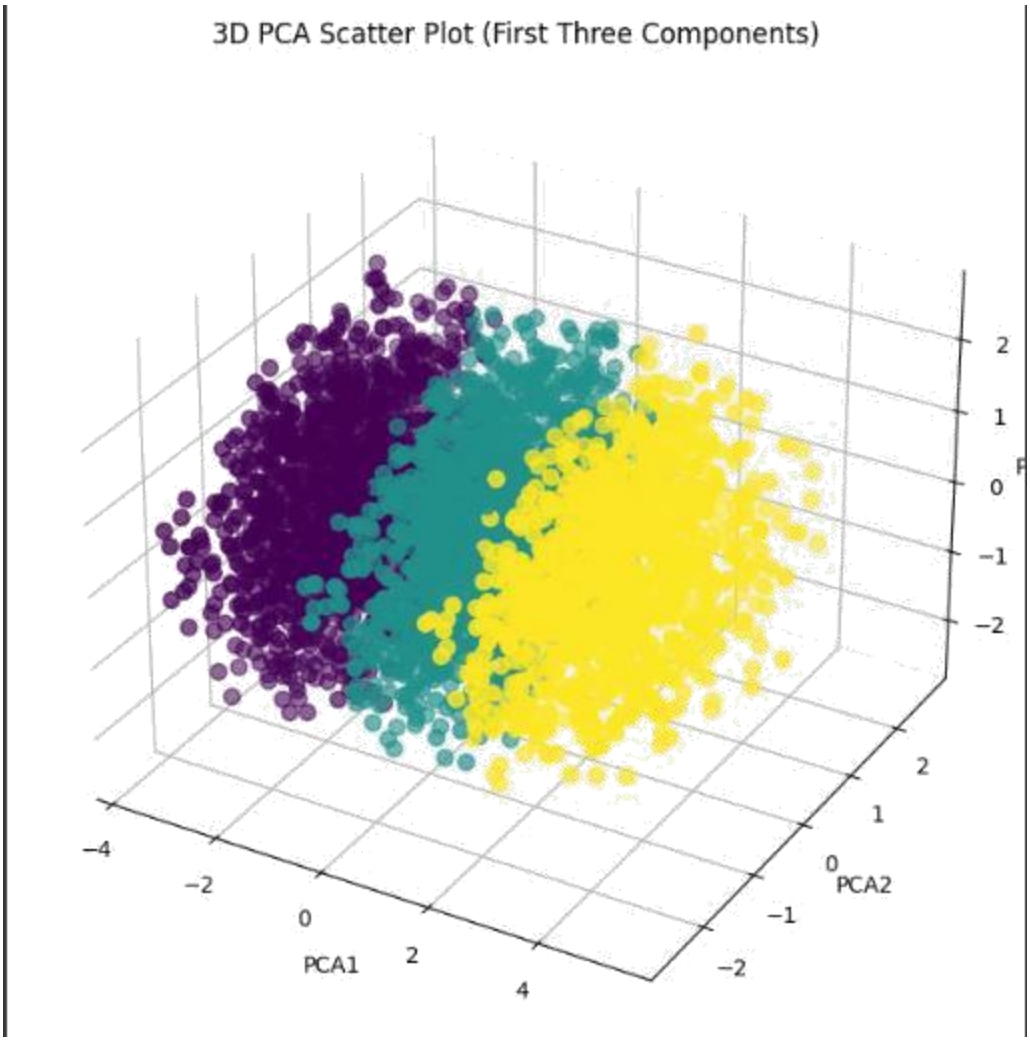


	PCA1	PCA2
0	2.918010	-0.786042
1	-1.275304	-0.177065
2	3.210459	-0.599451
3	2.473858	0.839491
4	2.445725	0.495872

- The provided table includes the transformed values for PCA1 and PCA2 along with other features.
- Each row shows the new coordinates of the data points after being projected onto the PCA components.
- Example:

- For the first row (Applicant 0):
  - The values for PCA1 and PCA2 are 2.918010 and -0.786042, respectively.
  - This point lies towards the positive side of PCA1, indicating a higher score along the direction that captures the most variance in the dataset.
- For the second row (Applicant 1):
  - The values for PCA1 and PCA2 are -1.275304 and -0.177065, respectively.
  - This point lies towards the negative side of PCA1, suggesting it has characteristics that are less aligned with the dominant variance direction.





- Explained Variance of PCA Components:
- The table and cumulative variance plot provide insights into how much of the dataset's variance is explained by each PCA component:
    - PCA1: Explains 49.9% of the variance. This indicates that the first principal component captures almost half of the information in the data.
    - PCA2: Adds another 11.4%, bringing the cumulative explained variance to 61.3%.
    - PCA3: Contributes an additional 11.0%, increasing the cumulative variance to 72.3%.
    - PCA4: Explains 10.9% more, raising the cumulative variance to 83.2%.
    - PCA5: Adds the final 6.5%, resulting in a total cumulative explained variance of 89.7%.

- The cumulative explained variance curve shows a diminishing return after the first three components.
- This suggests that the first three components are sufficient to capture the majority of the information in the dataset, while additional components contribute less.

Random Forest Classifier:

The random forest algorithm is a machine learning technique that generates a single outcome by aggregating the output of several decision trees. Because it's flexible and simple to use, it's a popular option for resolving classification and regression issues.

- RandomForestClassifier and Label Encoder is imported to fit and transform the categorical columns.
- The Random Forest Classifier combines all decision trees to improve overall accuracy.
- The accuracy (98.36%) and classification report is calculated.

Neural network:

The way nodes transmit data to one another in neural networks is modelled after how neurons transmit electrical impulses, drawing inspiration from the human brain. Every node has a threshold value and a weight. When a node's output exceeds the threshold, it transmits information to the network's next tier.

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	120
dense_1 (Dense)	(None, 5)	55
dense_2 (Dense)	(None, 1)	6

Total params: 181 (724.00 B)
Trainable params: 181 (724.00 B)
Non-trainable params: 0 (0.00 B)

Number of parameters between the i/p and first hidden layer are 80 and By adding bias to each neuron in the first hidden layer the total number of parameters are 120

Summary of a neural network model created using Keras:

Layers:

1. **Dense Layer 1 (dense_3):**
   - Output Shape: (None, 10)
   - Parameters: 120
     (This likely comes from 10 units, each connected to an input size of 12 weights + 10 biases = 120 total.)
2. **Dense Layer 2 (dense_4):**
   - Output Shape: (None, 5)
   - Parameters: 55
     (This comes from 5 units, each connected to 10 outputs of the previous layer with 5 biases: (10 \times 5 + 5 = 55).)
3. **Dense Layer 3 (dense_5):**
   - Output Shape: (None, 1)
   - Parameters: 6
     (This comes from 1 output unit, connected to 5 outputs of the previous layer with 1 bias: (5 \times 1 + 1 = 6).)

Summary:

- **Total Parameters:** 181
  (Sum of 120 + 55 + 6)
- **Trainable Parameters:** 181
- **Non-Trainable Parameters:** 0

1. **Model Compilation:**

Loss Function: binary_crossentropy

Used for binary classification tasks. It measures the difference between the predicted probabilities and the actual binary labels (0 or 1).

Optimizer: adam

Combines the benefits of both RMSProp and momentum optimization algorithms. It adapts the learning rate during training, making it a good default optimizer for most cases.

Metrics: ['accuracy']

Tracks accuracy during training and evaluation.

2. Model Training:

Inputs:

x_train and y_train: Training data and corresponding labels.

validation_data=(x_test, y_test): The validation set used to evaluate the model's performance at the end of each epoch.

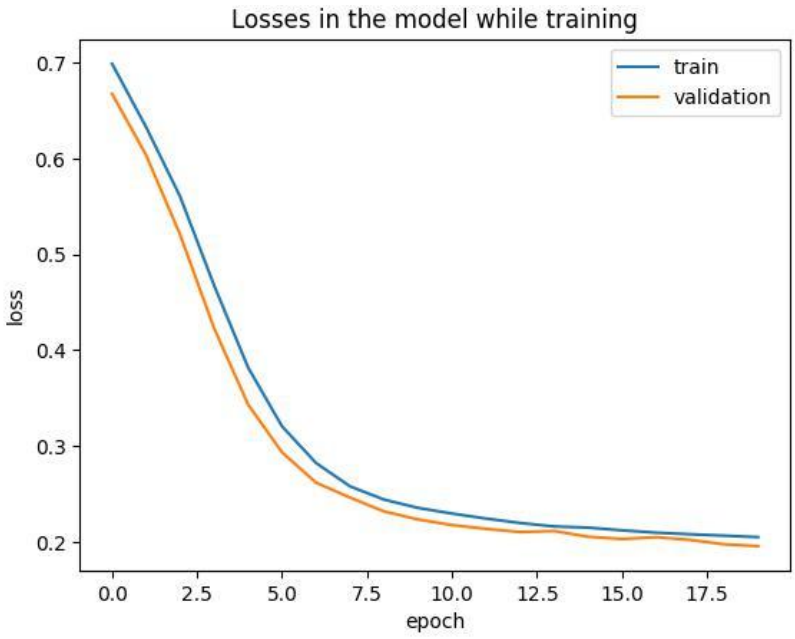
Epochs: 20

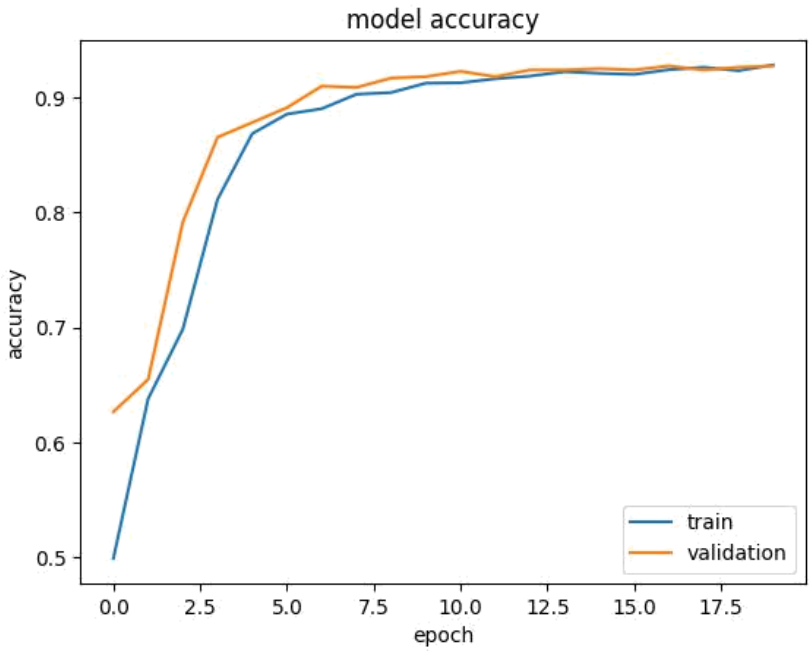
The model will iterate over the training data 20 times.

Output:

The model.fit function returns a History object (model_history) that contains information about the training and validation metrics for each epoch.

Visualizing the performance of model:





Training Summary:

1. **Initial Performance:**
   - **Training Accuracy:** Started at 50.28% (Epoch 1).
   - **Training Loss:** Initially at 0.6926, indicating the model was close to random guessing.
   - **Validation Accuracy:** Started at 68.97% (Epoch 1).
   - **Validation Loss:** Initially at 0.6471.
2. **Improvement Over Epochs:**
   - Significant improvement in both training and validation accuracy during the first few epochs:
     - By **Epoch 5**, training accuracy reached 89.79%, and validation accuracy reached 90.52%.
   - Validation loss also decreased steadily, reaching **0.2906** by Epoch 5.
3. **Later Epochs:**
   - The model continued to improve, with both training and validation accuracy stabilizing at around **91-92%**.
   - Training loss stabilized around **0.21–0.22**.
   - Validation loss decreased to **0.2114** by the final epoch (Epoch 20).

- Both training and validation loss consistently decreased, suggesting the model learned effectively.
- Final performance:
  - **Training Accuracy:** 91.89%
  - **Validation Accuracy:** 91.69%

- **Training Loss:** 0.2148
- **Validation Loss:** 0.2114

The model is well-trained and performs well on the validation set, with validation accuracy reaching **91.69%**. There is no significant overfitting or underfitting.

Conclusion:

The project successfully demonstrated the potential of machine learning techniques in predicting loan approval outcomes, offering an efficient alternative to traditional manual evaluation methods. Key findings include:

1. Model Effectiveness: Multiple models were implemented and evaluated, with Random Forest achieving the highest accuracy of 98.36%, followed by Support Vector Machine (92.66%) and Logistic Regression (90.32%). Neural networks also showed strong performance with an accuracy of approximately 91.69%.
2. Feature Insights:
    - Income, credit score, and employment type emerged as the most critical predictors of loan approval.
    - A high loan-to-value ratio negatively impacted approval chances, emphasizing financial stability as a key criterion.
3. Clustering Insights: K-means clustering revealed distinct applicant groups, with higher approval rates for conservative loan requests relative to income and asset levels.
4. Dimensionality Reduction: PCA efficiently reduced data complexity while preserving essential information, highlighting its utility in enhancing model interpretability.
5. Impact: The machine learning framework provides a scalable and unbiased approach for financial institutions to streamline loan approval processes, minimize human biases, and improve decision-making accuracy.

In conclusion, this study validates the application of advanced machine learning models to improve financial risk assessment and operational efficiency in loan approval systems. The outcomes underscore the importance of accurate data preprocessing and feature selection to achieve reliable predictions. This project paves the way for future exploration into enhancing model interpretability and integrating real-time decision-making capabilities.

Project Management:

Implementation Status

Below is a high-level timeline illustrating the progress of the project and key milestones:

Phase	Task	Timeline	Milestone
Phase 1	Exploratory Data Analysis (EDA)	Week 1 - 3	Data cleaning, correlation analysis, feature importance, and visualizations.
Phase 2	Model Training and Evaluation	Week 4 - 8	Training, hyperparameter tuning, and evaluation of machine learning models.
Final Phase	Documentation and Presentation	Week 9 - 10	Completed project report and prepared presentation for stakeholders.

Issues/Concerns

- **Model Selection Challenges**:
  - Issue: Difficulty in identifying optimal models due to dependencies between features.
  - Resolution: Implemented PCA to reduce dimensions but retained features that significantly impacted model performance.
- **Feature Importance**:
  - Issue: Some features showed significant interdependencies, affecting performance and interpretability.
  - Resolution: Analyzed feature importance using Random Forest and adjusted model selection accordingly.
- **Balancing Accuracy and Complexity**:
  - Issue: Ensuring a balance between model accuracy and computational efficiency.
  - Resolution: Performed hyperparameter tuning and cross-validation to optimize models.

References:

- Álvarez-Alvarado, José & Moreno, G.J. & Obregón-Biosca, Saul & Ronquillo, Guillermo & Ventura-Ramos, Eusebio & Trejo-Perea, Jr. (2021). Hybrid Techniques to Predict Solar Radiation Using Support Vector Machine and Search Optimization Algorithms: A Review. Applied Sciences. 11. 1044. 10.3390/app11031044.
- Hu, Xiaodong, Xinqing Wang, Fan-jie Meng, Xia Hua, Yu-ji Yan, Yu-yang Li, Jing Huang and Xue-mei Jiang. "Gabor-CNN for object detection based on small samples." *Defence Technology* 16 (2020): 1116-1129.
- Chen, Wei & Luo, Yu-Feng & Wen, Xinyu & Zhang, Chunze & Yin, & Wu, Yingdan & Yao,. (2019). Pre-evacuation Time Estimation Based Emergency Evacuation Simulation in Urban Residential Communities. International Journal of Environmental Research and Public Health. 16. 4599. 10.3390/ijerph16234599.
- Sarker, Iqbal. (2021). Machine Learning: Algorithms, Real-World Applications and Research Directions. SN Computer Science. 2. 10.1007/s42979-021-00592-x.
- Doan, Tri & Kalita, Jugal. (2015). Selecting Machine Learning Algorithms Using Regression Models. 1498-1505. 10.1109/ICDMW.2015.43.
- Chaudhary, Archana & Kolhe, Savita & Kamal, Raj. (2016). An improved Random Forest Classifier for multi-class classification. Information Processing in Agriculture. 3. 10.1016/j.inpa.2016.08.002.