MAJOR PROJECT: ANALYZE LOAN SANCTION DATASET

Sheetal Mishra

Data Science July 2024 Batch

Project Brief

You are provided with a loan sanction dataset where you will have to identify whether the loan of a particular person is approved or not depending on the information the individual has provided.

Steps for you to follow:

1. Import all the necessary libraries
2. Import the dataset provided
3. Understand the data
4. Deal with the missing values if any
5. Do some visualization if necessary
6. Divide the dataset into training and test datasets
7. Build the machine learning model which ever is suitable for the dataset
8. Fit the model on the training dataset
9. Test the model and find the accuracy of the model on the test and the training datasets
10. Create a confusion matrix

At last, draw conclusions based on the dataset provided and document the same on the jupyter/colab notebook.

Links

- Link to Colab file
- Link to Tableau dashboard
- Link to Github
- Link to the Google Drive folder

In the document

- Task 1 : Build a machine learning model : Code
- Task 2: Create a Dashboard on Tableau

Code

Dataset: Loan Sanction Dataset

Goal: Identify whether the loan of a particular person is approved or not depending on the information the individual has provided.

1. Import Libraries

```
# Load libraries

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

2. Load the Dataset

```
# Load the dataset

data = pd.read_excel('/content/drive/MyDrive/Colab Notebooks/Academor/Major Project/loan-predictionUC.csv.xlsx')
```

```
data.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

Dropping the Loan_ID column, because this column is not relevant for prediction.

```
# Dropping the first column - Loan_ID
data.drop('Loan_ID', axis=1, inplace=True)
```

```
data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

3. Understand the Data

```
data.shape
```

```
(614, 12)
```

We see that there are 614 individual's data with 12 columns.

```
# These are the column names
data.columns
```

```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

Column Description

Column Name	Description
Loan_ID	A unique identifier for each loan application.
Gender	The gender of the applicant. • Male • Female
Married	Indicates whether the applicant is married.
Dependents	The number of dependents the applicant has, including children and other family members. • 0 • 1 • 2 • 3+
Education	The educational qualification of the applicant. • Graduate • Non-graduate
Self_Employed	Indicates whether the applicant is self-employed.
ApplicantIncome	The monthly income of the applicant.
CoapplicantIncome	The monthly income of the co-applicant (if any).
LoanAmount	The loan amount applied for by the applicant.
Loan_Amount_Term	The term (duration) of the loan, typically measured in months. • 360 for 30 years • 180 for 15 years • 120 for 10 years
Credit_History	Indicates whether the applicant has a good credit history. • 1 : Good credit history • 0 : No or poor credit history
Property_Area	The area type where the property is located. • Urban • Semiurban • Rural
Loan_Status	Target Variable : Indicates whether the loan was approved or not.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Gender             601 non-null    object 
 1   Married            611 non-null    object 
 2   Dependents         599 non-null    object 
 3   Education          614 non-null    object 
 4   Self_Employed      582 non-null    object 
 5   ApplicantIncome    614 non-null    int64  
 6   CoapplicantIncome  614 non-null    float64
 7   LoanAmount         592 non-null    float64
 8   Loan_Amount_Term   600 non-null    float64
 9   Credit_History     564 non-null    float64
 10  Property_Area      614 non-null    object 
 11  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(7)
memory usage: 57.7+ KB
```

Convert Data Types

Here, we see that `Credit_History` is a `float` type variable, but we want it as an `object` type because it can only take 2 values:

- `1` : Good credit history
- `0` : No or poor credit history

```
# Convert all values in the 'Credit_History' column to string data type
data['Credit_History'] = data['Credit_History'].astype(str)
```

Correct Inconsistencies

```
# Get data types of individual values in 'Dependents` column
value_types = data['Dependents'].apply(lambda x: type(x))

print(value_types.unique())
```

```
[<class 'int'> <class 'str'> <class 'float'>]
```

We also observe see that although `Dependents` column is an `object` type, it contains `int`, `float` and `str` type of values that might create inconsistencies in our analysis. So let's convert the data types of all of its values to `str`.

```
# Convert all values in the 'Dependents' column to strings
data['Dependents'] = data['Dependents'].astype(str)
```

```
# Verify
value_types = data['Dependents'].apply(lambda x: type(x))

print(value_types.unique())
```

```
[<class 'str'>]
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Gender             601 non-null    object 
 1   Married            611 non-null    object 
 2   Dependents         614 non-null    object 
 3   Education          614 non-null    object 
 4   Self_Employed      582 non-null    object 
 5   ApplicantIncome    614 non-null    int64  
 6   CoapplicantIncome  614 non-null    float64
 7   LoanAmount         592 non-null    float64
 8   Loan_Amount_Term   600 non-null    float64
 9   Credit_History     614 non-null    object 
 10  Property_Area      614 non-null    object 
 11  Loan_Status        614 non-null    object 
dtypes: float64(3), int64(1), object(8)
memory usage: 57.7+ KB
```

There are 8 categorical columns (including target variable), and 4 numerical columns.

Separate Numerical and Categorical Columns

```
# Separating numerical and categorical columns
num_cols = list(data.select_dtypes(include=['float64', 'int64']).columns)
cat_cols = list(data.select_dtypes(include=['object']).columns)
```

```
num_cols
```

```
['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term']
```

```
cat_cols
```

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

Descriptive Statistics

```
data[num_cols].describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
count	614.000000	614.000000	592.000000	600.00000
mean	5403.459283	1621.245798	146.412162	342.00000
std	6109.041673	2926.248369	85.587325	65.12041
min	150.000000	0.000000	9.000000	12.00000
25%	2877.500000	0.000000	100.000000	360.00000
50%	3812.500000	1188.500000	128.000000	360.00000
75%	5795.000000	2297.250000	168.000000	360.00000
max	81000.000000	41667.000000	700.000000	480.00000

Here, we can observe various measures for all the numerical columns.

- Applicant income ranges from 150 to to $81{,}000$.
- Loan Amount has a range of $9k$ to $700k$, with an average of $146k$.
- Loan Amount Term has 360 for all its quartile values, meaning, most of the people have a loan duration of 30 years.

Analyze Categorical Variables

```
#Separating Target Variable from cat_cols
target = 'Loan_Status'
cat_cols.remove(target)
cat_cols
```

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

```
for col in cat_cols:
    print(f"{col}: {data[col].unique()}")
```

```
Gender: ['Male' 'Female' nan]
Married: ['No' 'Yes' nan]
Dependents: ['0' '1' '2' '3+' 'nan']
Education: ['Graduate' 'Not Graduate']
Self_Employed: ['No' 'Yes' nan]
Credit_History: ['1.0' '0.0' 'nan']
Property_Area: ['Urban' 'Rural' 'Semiurban' 'Semi-urban' 'semiurban']
```

Correct Inconsistencies

1. We observe that columns `Dependents` and `Credit_History` have `nan` filled as string values.
   - Fix it by replacing all the `nan` string values by `np.nan`.

2. Column `Property_Area` has three separate values `Semiurban`, `Semi-urban` and `semiurban` that basically indicate to the same thing.
   - Fix it by replacing these by `Semiurban` so as to have homogenous values.

```python
# Replace 'nan' with np.nan
data.replace(['nan'], np.nan, inplace=True)
```

```python
# Replace variations of 'semiurban' with a consistent value, e.g., 'semiurban'

data['Property_Area'] = data['Property_Area'].replace({
    'Semi-urban': 'Semiurban',
    'semiurban': 'Semiurban'
})
```

```python
# Verify
for col in cat_cols:
    print(f"{col}: {data[col].unique()}")
```

```
Gender: ['Male' 'Female' nan]
Married: ['No' 'Yes' nan]
Dependents: ['0' '1' '2' '3+' nan]
Education: ['Graduate' 'Not Graduate']
Self_Employed: ['No' 'Yes' nan]
Credit_History: ['1.0' '0.0' nan]
Property_Area: ['Urban' 'Rural' 'Semiurban']
```

```python
# Count the number of occurrences of each categorical variable
for col in cat_cols:
    print(f"{col}: {data[col].value_counts()}")
    print('\n')
```

```
Gender: Gender
Male      489
Female    112
Name: count, dtype: int64


Married: Married
Yes    398
No     213
Name: count, dtype: int64


Dependents: Dependents
0     345
1     102
2     101
3+     51
Name: count, dtype: int64


Education: Education
Graduate        480
Not Graduate    134
Name: count, dtype: int64


Self_Employed: Self_Employed
No     500
Yes     82
Name: count, dtype: int64


Credit_History: Credit_History
1.0    475
0.0     89
Name: count, dtype: int64


Property_Area: Property_Area
Semiurban    233
Urban        202
Rural        179
Name: count, dtype: int64
```

4. Data Preprocessing

Handle Missing Values

```
data.isnull().sum()
```

	0
Gender	13
Married	3
Dependents	15
Education	0
Self_Employed	32
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	22
Loan_Amount_Term	14
Credit_History	50
Property_Area	0
Loan_Status	0

dtype: int64

Columns with non-zero values indicate the number of missing values that column contains.

```
# Use SimpleImputer to fill missing values
from sklearn.impute import SimpleImputer
```

```
# Filling all the numerical values with mean
num_imputer = SimpleImputer(strategy='mean')
data[num_cols] = num_imputer.fit_transform(data[num_cols])

# Filling all the categorical values with mode
cat_imputer = SimpleImputer(strategy='most_frequent')
data[cat_cols] = cat_imputer.fit_transform(data[cat_cols])
```

Filling missing values with mean for numerical columns and mode for categorical columns.

```
data.isnull().sum()
```

	0
Gender	0
Married	0
Dependents	0
Education	0
Self_Employed	0
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
Loan_Amount_Term	0
Credit_History	0
Property_Area	0
Loan_Status	0

dtype: int64

We have handled the missing values succesfully.

Export the dataset for Tableau

The dataset is now complete with no missing values and the correct format.

It's time to export this dataset for Tableau visualizations.

```
# Export to excel
data.to_excel('/content/drive/MyDrive/Colab Notebooks/Academor/Major Project/Tableau_loan-predictionUC.xlsx', index=False)
```

Encoding Categorical Variables

We're label encoding the categorical variables to convert them into numerical form.

```
# Label Encoding for categorical variables
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

data[cat_cols] = data[cat_cols].apply(le.fit_transform)
```

```
data[cat_cols].head()
```

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area
0	1	0	0	0	0	1	2
1	1	1	1	0	0	1	0
2	1	1	0	0	1	1	2
3	1	1	0	1	0	1	2
4	1	0	0	0	0	1	2

Encoding Target Variable

Encoding Target variable as:

- `Y:1`
- `N:0`

```
# Convert target variable to numerical
data[target] = data[target].map({'Y': 1, 'N': 0})
```

```
# This is how the final pre-processed data looks like
data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	1	0	0	0	0	5849.0	0.0	146.412162	360.0	1	2	1
1	1	1	1	0	0	4583.0	1508.0	128.000000	360.0	1	0	0
2	1	1	0	0	1	3000.0	0.0	66.000000	360.0	1	2	1
3	1	1	0	1	0	2583.0	2358.0	120.000000	360.0	1	2	1
4	1	0	0	0	0	6000.0	0.0	141.000000	360.0	1	2	1

Analyze Correlations

```
data[num_cols].corr().round(2)
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
ApplicantIncome	1.00	-0.12	0.57	-0.05
CoapplicantIncome	-0.12	1.00	0.19	-0.06
LoanAmount	0.57	0.19	1.00	0.04
Loan_Amount_Term	-0.05	-0.06	0.04	1.00

There is a positive correlation between `ApplicantIncome` and `LoanAmount`.

5. Split the Data

Splitting the features and the target in the dataset into two variables X and y.

```
# Split the dataset into features matrix and target variable
X = data.drop(columns=['Loan_Status'])
y = data['Loan_Status']
```

Splitting the data into training set (75%) to train the model, and testing set (25%) to evaluate model performance.

```
from sklearn.model_selection import train_test_split

# Now split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)
```

Feature Scaling for Numerical Variables

Standardization

$$X' = \frac{X - \mu}{\sigma}$$

Standardizing numerical features to have a mean of 0 and standard deviation of 1 to ensure that all the values are in the same scale. This is done to prevent one feature to dominate others.

```
# Feature scaling for numerical variables
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])
```

6. Train the Logistic Regression Model on the Training Set

I explored multiple classification models, including Logistic Regression, Decision Tree, K-Nearest Neighbors (K-NN), Support Vector Machine (SVM), Kernel SVM, and Random Forest. After evaluating their performance on the test set, both **Logistic Regression** and **Kernel SVM** emerged as the top performers, achieving an accuracy of 83%.

I selected Logistic Regression for the final model.

```
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)

# Train the classifier model
classifier.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(random_state=0)
```

7. Test the Model

Accuracy of the model on the Training Set

Evaluating the model performance on the train dataset. This involves generating predictions for the train set and comparing them against the actual values in `y_train`.

```python
from sklearn.metrics import accuracy_score

# Accuracy on the training set
print(f"Accuracy: {accuracy_score(y_train, classifier.predict(X_train))}")
```

```
Accuracy: 0.8021739130434783
```

Accuracy of the model on the Test Set

Evaluating the model performance on the test dataset. This involves generating predictions for the test set, `y_pred`, and comparing them against the actual outcomes, `y_test`, to assess the model's accuracy.

```python
# Make prediction on the test set
y_pred = classifier.predict(X_test)

# Accuracy on the test set
accuracy = "{:.2f} %".format(accuracy_score(y_test, y_pred)*100)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 83.77 %
```

An accuracy of 83.77% on the test set is a strong result, indicating that the model has learned the underlying patterns in the data effectively and it correctly predicts the outcome in the majority of cases. This accuracy suggests that the model generalizes well to unseen data.

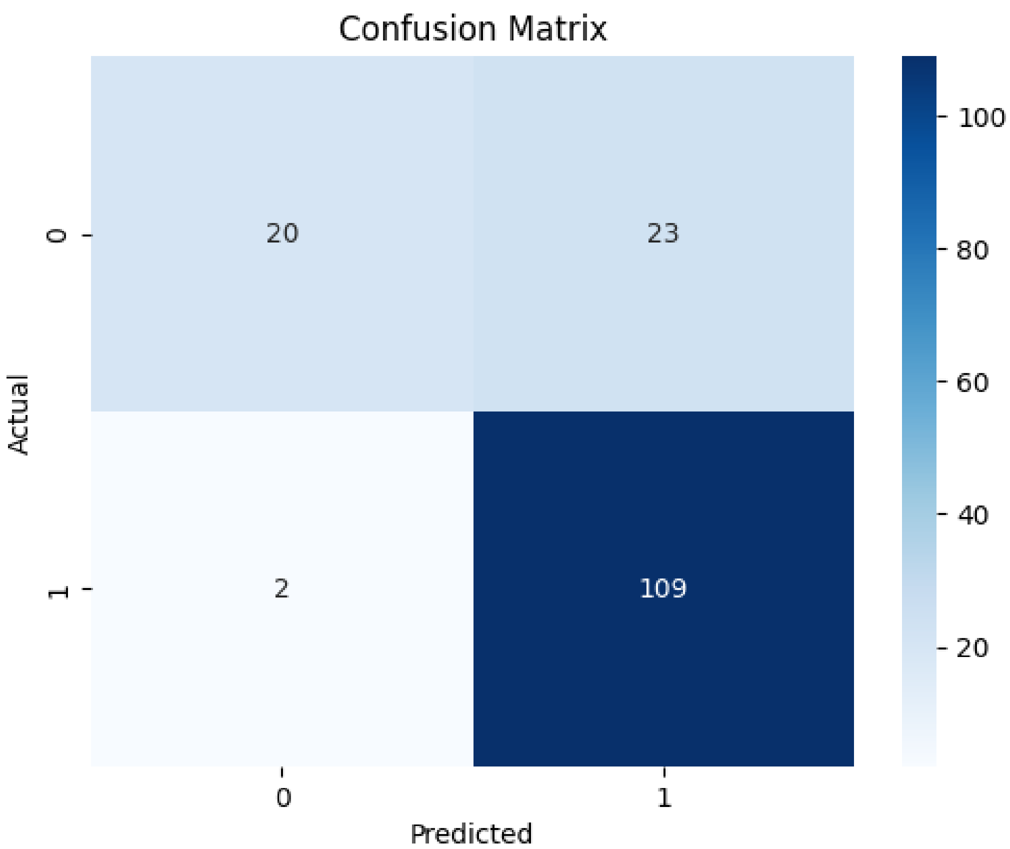
8. Create Confusion Matrix

The confusion matrix provides a breakdown of true positives, true negatives, false positives, and false negatives, enabling a more granular evaluation of the model's classification accuracy.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Number of Observations}}$$

```
from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Confusion Matrix Analysis:

- **True Positives (TP):** 109 cases where the model correctly predicted the positive class.
- **True Negatives (TN):** 20 cases where the model correctly predicted the negative class.
- **False Positives (FP):** 23 cases where the model incorrectly predicted the positive class when it was actually negative.

- **False Negatives (FN):** 2 cases where the model incorrectly predicted the negative class when it was actually positive.

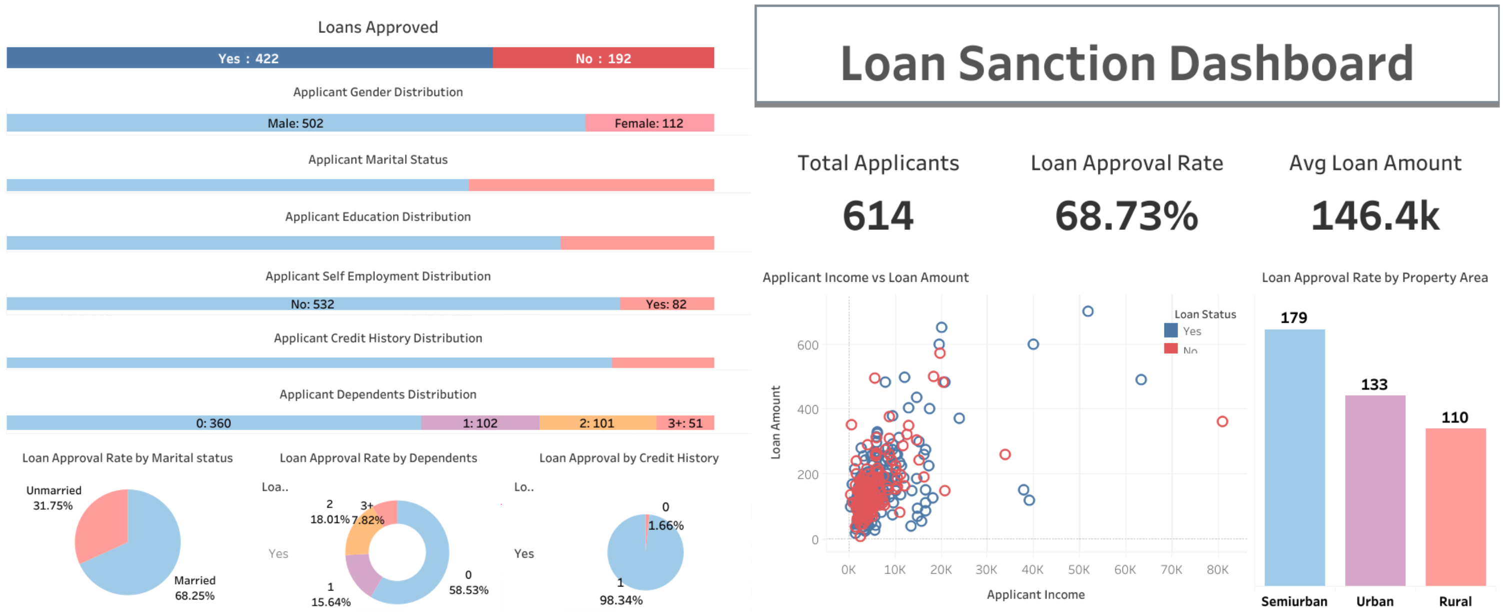
Conclusion:

Overall, the model does a good job of correctly predicting positive cases, which is great for situations where catching every positive case is important. But it has a moderate number of false positives, which means it sometimes predicts a positive when it shouldn't.

Task 2: On the same dataset draw conclusions from the dataset and create a Tableau dashboard for the same.

Link to the Dashboard.

And here, is the image of the dashboard:



Loans Approved

Applicant Gender Distribution

Applicant Marital Status

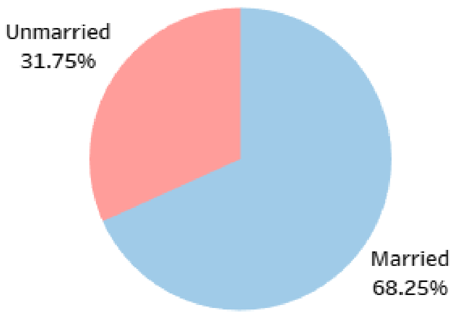
Applicant Education Distribution

Applicant Self Employment Distribution

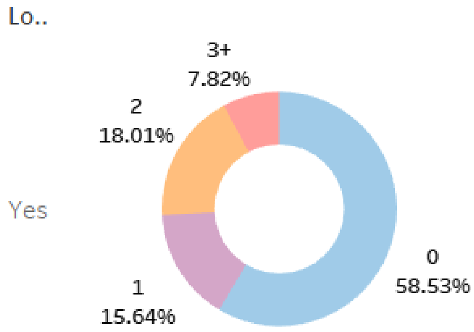
Applicant Credit History Distribution

Applicant Dependents Distribution

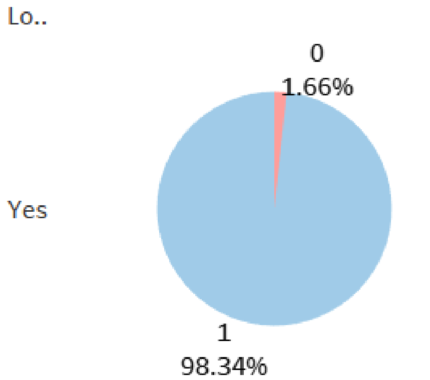
Loan Approval Rate by Marital status



Loan Approval Rate by Dependents



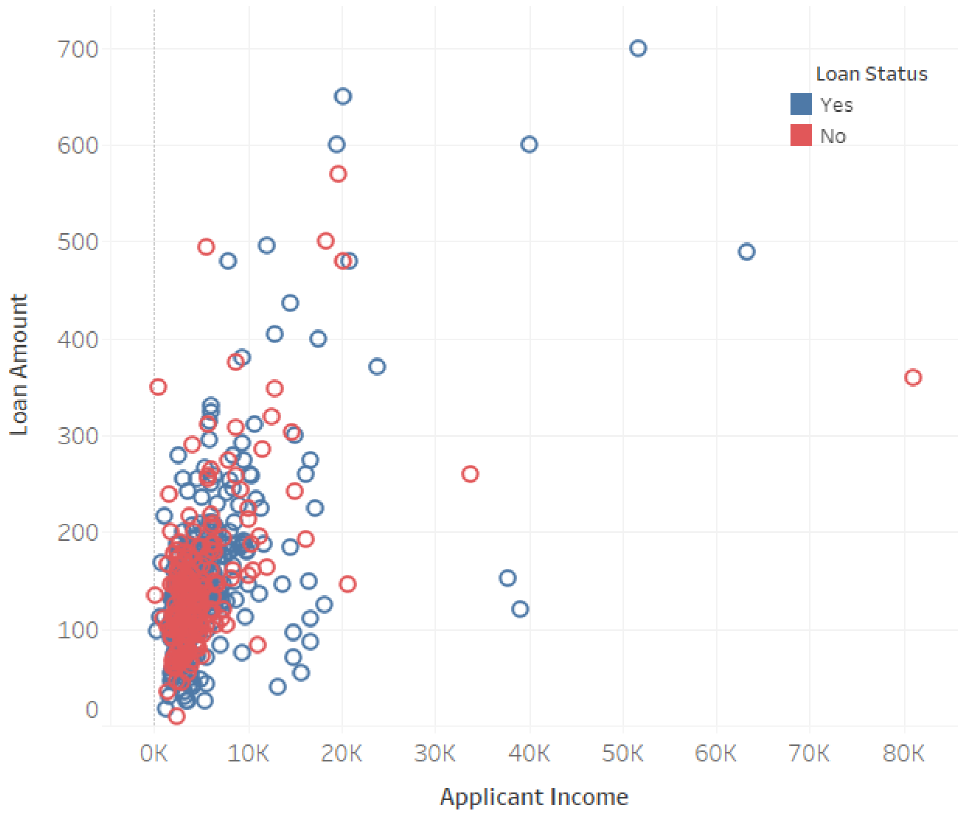
Loan Approval by Credit History



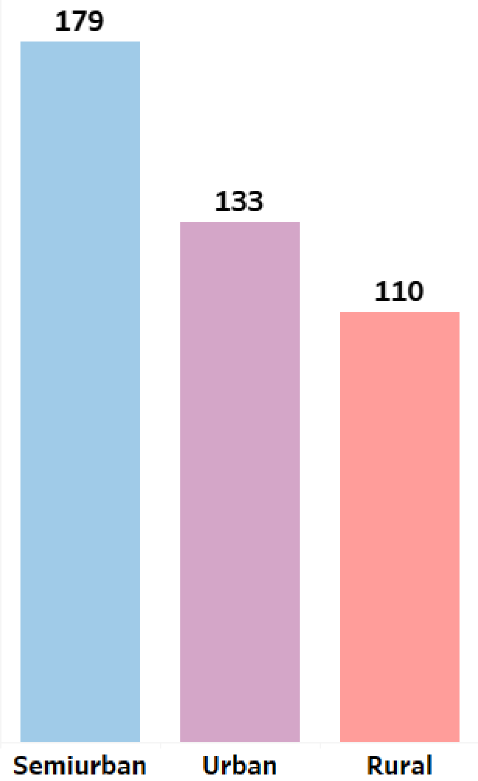
Loan Sanction Dashboard

Total Applicants	Loan Approval Rate	Avg Loan Amount
614	**68.73%**	**146.4k**

Applicant Income vs Loan Amount



Loan Approval Rate by Property Area



Loan Sanction Dashboard Overview

This dashboard provides a comprehensive analysis of the loan sanction dataset. The visualizations and metrics allow for a detailed exploration of the data, enabling us better understanding and decision-making.

Visualizations and Metrics Included and Observations:

- **Loans Approved:** Displays the overall distribution of loan approvals and rejections.
    - 2/3rds of the Loan Applications were approved.

- **Applicant Gender Distribution:** Shows the distribution of applicants by gender, helping us to identify gender-based.
    - Number of male applicants are significantly higher than female applicants.

- **Applicant Marital Status:** Illustrates the proportion of applicants based on marital status.
    - The number of Married applicants are almost double the number of unmarried applicants.

- **Applicant Education Distribution:** Depicts the educational background of applicants.
    - The number of graduate applicants are much higher than non-graduate applicants.

- **Applicant Self-Employment Distribution:** Highlights the employment status of applicants.
    - Majority of the applicants are not self-employed.

- **Applicant Credit History Distribution:** Shows the distribution of applicants based on their credit history, a critical factor in loan approval decisions.
    - More than 1/4th of the total applicants have a good Credit History (=1).

- **Applicant Dependents Distribution:** Provides the distribution of applicants based on the number of dependents.
    - More than half of the applicants have no dependents.

- **Loan Approval Rate by Marital Status Chart:** Compares the loan approval rate between married and unmarried applicants, revealing any patterns related to marital status.
    - The chances of getting one’s loan approved are slightly higher if they're married.

- **Loan Approval Rate by Dependents Chart:** Analyzes the loan approval rate based on the number of dependents, providing insights into how dependents impact loan decisions.
    - More than half of the approved loan applicants have no dependents.

- **Loan Approval Rate by Credit History:** Analyzes the loan approval rate based on the credit history, a critical factor in determining loan decisions.
    - Applicants with a good credit history (`Credit_History = 1`) have a much higher approval rate.

- **Loan Approval Rate by Property Area Chart:** Examines the approval rate across different property areas (Urban, Rural, Semi-urban), identifying geographical trends in loan approvals.
    - Loans are more frequently approved in `Semiurban` areas compared to `Rural` ones.

- **Applicant Income vs Loan Amount Scatter Plot:** Visualizes the relationship between applicant income and the loan amount requested, offering a perspective on the affordability and risk associated with different income levels.
    - Higher income generally correlates with higher loan amounts, but exceptions might occur based on other factors like credit history.

- **Total Applicants Metric:** Displays the total number of loan applicants in the dataset.

- **Loan Approval Metric:** Shows the total number of approved loans, giving a quick overview of the success rate.

- **Average Loan Amount Metric:** Highlights the average loan amount sanctioned, helping to understand the typical loan size granted.

This dashboard serves as a powerful tool for analyzing loan applications, enabling stakeholders to identify key factors that influence loan approval. The various demographic and financial metrics help us to uncover patterns and trends that may inform future lending strategies.
