

<https://www.kaggle.com/datasets/shreyasparaj1/loan-prediction-dataset>

```
from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import warnings
warnings.filterwarnings("ignore")

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

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest, chi2, RFE

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, classification_report
```

```
df = pd.read_csv('/content/drive/MyDrive/project1/loan_Dataset.csv')
```

df.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	

Next steps: [Generate code with df](#) [New interactive sheet](#)

df['Loan_Status'].value_counts()

	count
Loan_Status	
Y	422
N	192

dtype: int64

The target variable distribution was checked to understand the class balance.

df.isnull().sum()

	0
Loan_ID	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

```
df.drop('Loan_ID', axis=1, inplace=True)

df.fillna(df.mode().iloc[0], inplace=True)

df['Dependents'] = df['Dependents'].replace('3+', 3).astype(int)
```

df.isnull().sum()

0

Gender0

Married0

Dependents0

Education0

Self_Employed0

ApplicantIncome0

CoapplicantIncome0

LoanAmount0

Loan_Amount_Term0

Credit_History0

Property_Area0

Loan_Status0

dtype: int64

Missing values were handled using the most frequent values, and the unnecessary ID column was removed.

```
df.replace({
    'Gender': {'Male': 0, 'Female': 1},
    'Married': {'No': 0, 'Yes': 1},
    'Education': {'Not Graduate': 0, 'Graduate': 1},
    'Self_Employed': {'No': 0, 'Yes': 1},
    'Property_Area': {'Rural': 0, 'Semiurban': 1, 'Urban': 2},
    'Loan_Status': {'N': 0, 'Y': 1}}, inplace=True)
```

df.head()

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	0	0	0	1	0	5849	0.0	120.0	360.0	1.0	2	1
1	0	1	1	1	0	4583	1508.0	128.0	360.0	1.0	0	0
2	0	1	0	1	1	3000	0.0	66.0	360.0	1.0	2	1
3	0	1	0	0	0	2583	2358.0	120.0	360.0	1.0	2	1
4	0	0	0	1	0	6000	0.0	141.0	360.0	1.0	2	1

Next steps: [Generate code with df](#) [New interactive sheet](#)

Categorical features were converted into numerical values to make the dataset suitable for machine learning models.

df.dtypes

0

Genderint64

Marriedint64

Dependentsint64

Educationint64

Self_Employedint64

ApplicantIncomeint64

CoapplicantIncomefloat64

LoanAmountfloat64

Loan_Amount_Termfloat64

Credit_Historyfloat64

Property_Areaint64

Loan_Statusint64

dtype: object

df.describe()

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
count	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000
mean	0.182410	0.653094	0.744300	0.781759	0.133550	5403.459283	1621.245798	145.465798	342.410423	0.855049	1.037459	0.687296
std	0.386497	0.476373	1.009623	0.413389	0.340446	6109.041673	2926.248369	84.180967	64.428629	0.352339	0.787482	0.463973
min	0.000000	0.000000	0.000000	0.000000	0.000000	150.000000	0.000000	9.000000	12.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	1.000000	0.000000	2877.500000	0.000000	100.250000	360.000000	1.000000	0.000000	0.000000
50%	0.000000	1.000000	0.000000	1.000000	0.000000	3812.500000	1188.500000	125.000000	360.000000	1.000000	1.000000	1.000000
75%	0.000000	1.000000	1.000000	1.000000	0.000000	5795.000000	2297.250000	164.750000	360.000000	1.000000	2.000000	1.000000
max	1.000000	1.000000	3.000000	1.000000	1.000000	81000.000000	41667.000000	700.000000	480.000000	1.000000	2.000000	1.000000

X = df.drop('Loan_Status', axis=1)
y = df['Loan_Status']

The dataset was divided into input features X and target variable y.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

Data was split into training and testing sets.

```
models_params = {
    "Random Forest": {
        "model": RandomForestClassifier(random_state=42),
        "params": {
            "n_estimators": [100],
            "max_depth": [5, None]
        }
    },
    "Decision Tree": {
        "model": DecisionTreeClassifier(random_state=42),
        "params": {
```

```
        "max_depth": [5, None],
        "criterion": ["gini"]
    }
},
"SVM": {
    "model": SVC(),
    "params": {
        "C": [1],
        "kernel": ["rbf"]
    }}
}
```

Grid Search was prepared to find the best parameters for each machine learning model.

```
grid_results = []

for name, item in models_params.items():
    grid = GridSearchCV(
        item["model"],
        item["params"],
        cv=3,
        n_jobs=-1)

    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    y_pred = best_model.predict(X_test)

    print(f"\nClassification Report for {name}")
    print("Best Parameters:", grid.best_params_)
    print(classification_report(y_test, y_pred))

    grid_results.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred) })

grid_results_df = pd.DataFrame(grid_results)
grid_results_df
```

Classification Report for Random Forest
Best Parameters: {'max_depth': None, 'n_estimators': 100}

	precision	recall	f1-score	support
0	0.79	0.44	0.57	43
1	0.76	0.94	0.84	80
accuracy			0.76	123
macro avg	0.77	0.69	0.70	123
weighted avg	0.77	0.76	0.74	123

Classification Report for Decision Tree
Best Parameters: {'criterion': 'gini', 'max_depth': 5}

	precision	recall	f1-score	support
0	0.73	0.44	0.55	43
1	0.75	0.91	0.82	80
accuracy			0.75	123
macro avg	0.74	0.68	0.69	123
weighted avg	0.74	0.75	0.73	123

Classification Report for SVM
Best Parameters: {'C': 1, 'kernel': 'rbf'}

	precision	recall	f1-score	support
0	0.00	0.00	0.00	43
1	0.65	1.00	0.79	80
accuracy			0.65	123
macro avg	0.33	0.50	0.39	123
weighted avg	0.42	0.65	0.51	123

	Model	Accuracy	Precision	Recall	
0	Random Forest	0.764228	0.757576	0.9375	
1	Decision Tree	0.747967	0.752577	0.9125	
2	SVM	0.650407	0.650407	1.0000	

Next steps:

[Generate code with grid_results_df](#)

[New interactive sheet](#)

The models were trained using Grid Search, and their performance was evaluated using accuracy, precision, and recall.

```
selector = SelectKBest(score_func=chi2, k=5)
selector.fit(X, y)

chi_df = pd.DataFrame({
    'Feature': X.columns,
    'Score': selector.scores_
}).sort_values(by='Score', ascending=False)

chi_df
```

	Feature	Score	
6	CoapplicantIncome	11342.041603	
5	ApplicantIncome	93.904964	
7	LoanAmount	30.214108	
9	Credit_History	26.005877	
8	Loan_Amount_Term	3.778554	
1	Married	1.782425	
3	Education	0.988390	
10	Property_Area	0.377837	
0	Gender	0.162407	
2	Dependents	0.085953	
4	Self_Employed	0.007285	

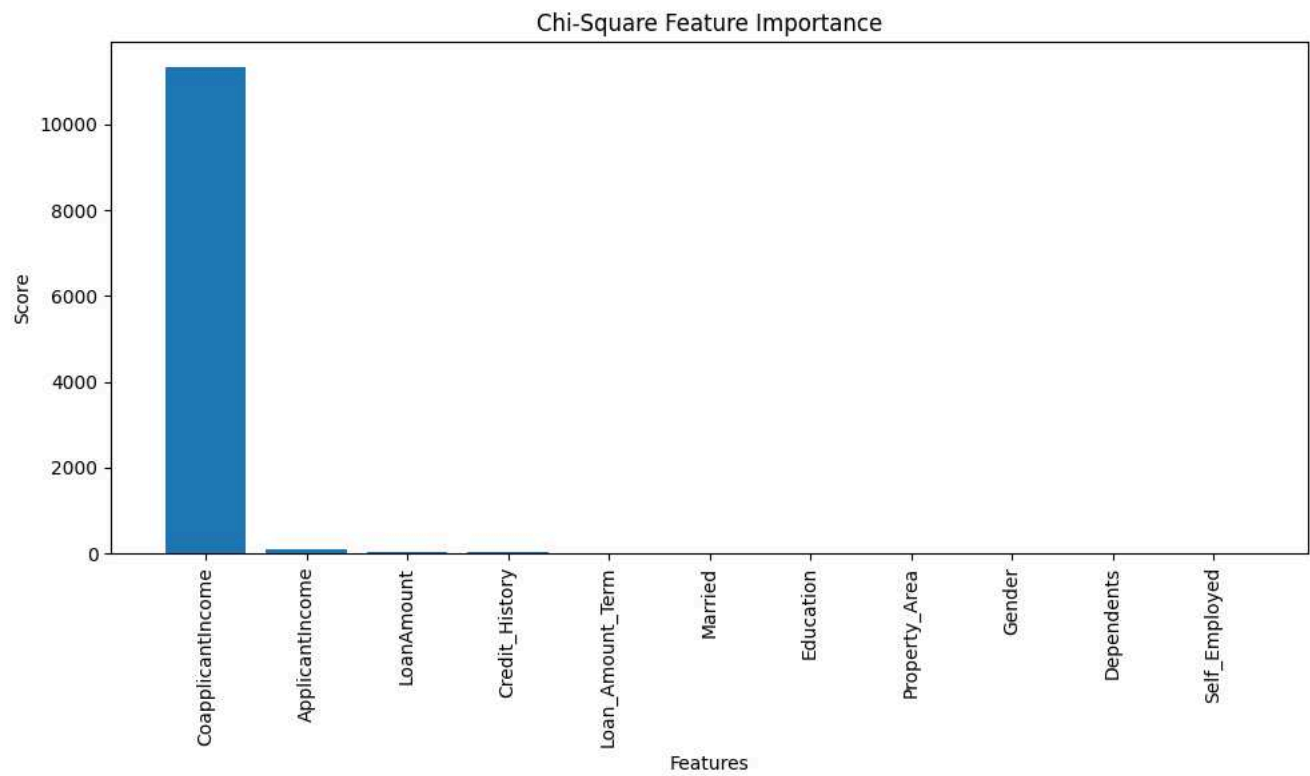
Next steps:

[Generate code with chi_df](#)

[New interactive sheet](#)

Chi-Square was used as the first feature selection method to identify the most important features.

```
plt.figure(figsize=(10, 6))
plt.bar(chi_df['Feature'], chi_df['Score'])
plt.xticks(rotation=90)
plt.title("Chi-Square Feature Importance")
plt.xlabel("Features")
plt.ylabel("Score")
plt.tight_layout()
plt.show()
```



The chart shows the importance score of each feature using the Chi-Square method.

```
chi_selected_features = X.columns[selector.get_support()]
X_chi = X[chi_selected_features]

print("Selected Features by Chi-Square:", list(chi_selected_features))
```

Selected Features by Chi-Square: ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History']

The most important features selected by Chi-Square were extracted.

```
X_train_chi, X_test_chi, y_train_chi, y_test_chi = train_test_split(
    X_chi, y, test_size=0.2, random_state=42
)
```

The dataset was split again using only the Chi-Square selected features.

```
models = {
    "Random Forest": RandomForestClassifier(random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC()
}

chi_results = []

for name, model in models.items():
    model.fit(X_train_chi, y_train_chi)
    y_pred = model.predict(X_test_chi)

    print(f"\nClassification Report for {name} using Chi-Square Features")
    print(classification_report(y_test_chi, y_pred))

    chi_results.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test_chi, y_pred),
        "Precision": precision_score(y_test_chi, y_pred),
        "Recall": recall_score(y_test_chi, y_pred)
    })

chi_results_df = pd.DataFrame(chi_results)
chi_results_df
```

```
Classification Report for Random Forest using Chi-Square Features
              precision    recall  f1-score   support

    0       0.80         0.47         0.59         43
    1       0.77         0.94         0.84         80

 accuracy         0.77         123
macro avg       0.78         0.70         0.72         123
weighted avg       0.78         0.77         0.75         123
```

```
Classification Report for Decision Tree using Chi-Square Features
              precision    recall  f1-score   support

    0       0.62         0.65         0.64         43
    1       0.81         0.79         0.80         80

 accuracy         0.74         123
macro avg       0.71         0.72         0.72         123
weighted avg       0.74         0.74         0.74         123
```

```
Classification Report for SVM using Chi-Square Features
              precision    recall  f1-score   support

    0       0.00         0.00         0.00         43
    1       0.65         1.00         0.79         80

 accuracy         0.65         123
macro avg       0.33         0.50         0.39         123
weighted avg       0.42         0.65         0.51         123
```

	Model	Accuracy	Precision	Recall	
0	Random Forest	0.772358	0.765306	0.9375	
1	Decision Tree	0.739837	0.807692	0.7875	
2	SVM	0.650407	0.650407	1.0000	

The models were trained using only the features selected by Chi-Square and evaluated again.

```
model = LogisticRegression(max_iter=500)

rfe = RFE(model, n_features_to_select=5)
rfe.fit(X, y)

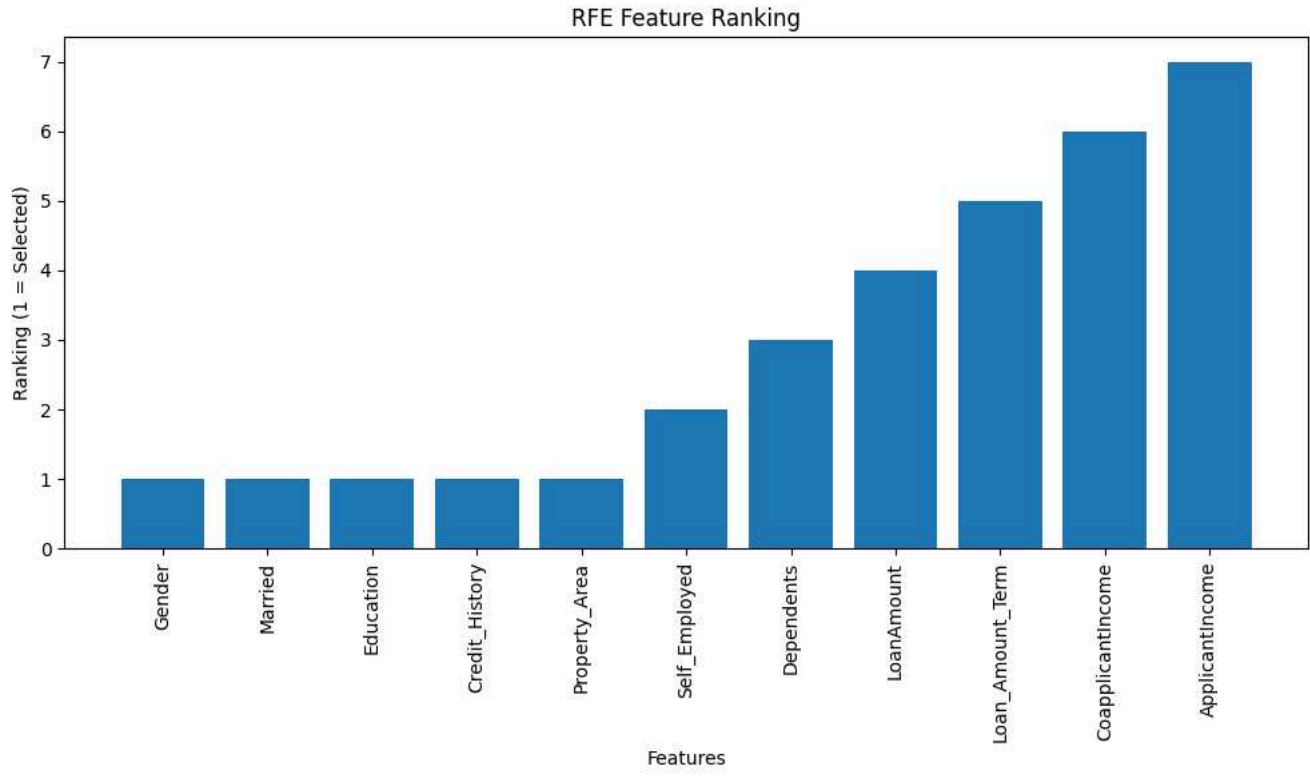
rfe_df = pd.DataFrame({
    'Feature': X.columns,
    'Ranking': rfe.ranking_
}).sort_values(by='Ranking')

rfe_df
```

	Feature	Ranking	
0	Gender	1	
1	Married	1	
3	Education	1	
9	Credit_History	1	
10	Property_Area	1	
4	Self_Employed	2	
2	Dependents	3	
7	LoanAmount	4	
8	Loan_Amount_Term	5	
6	CoapplicantIncome	6	
5	ApplicantIncome	7	

RFE was used as the second feature selection method to rank and select the most important features.

```
plt.figure(figsize=(10, 6))
plt.bar(rfe_df['Feature'], rfe_df['Ranking'])
plt.xticks(rotation=90)
plt.title("RFE Feature Ranking")
plt.xlabel("Features")
plt.ylabel("Ranking (1 = Selected)")
plt.tight_layout()
plt.show()
```



The chart shows the ranking of features using RFE, where rank 1 means the feature was selected.

```
rfe_selected_features = rfe_df[rfe_df['Ranking'] == 1]['Feature']
X_rfe = X[rfe_selected_features]

print("Selected Features by RFE:", list(rfe_selected_features))

Selected Features by RFE: ['Gender', 'Married', 'Education', 'Credit_History', 'Property_Area']
```

The most important features selected by RFE were extracted.

```
X_train_rfe, X_test_rfe, y_train_rfe, y_test_rfe = train_test_split(
    X_rfe, y, test_size=0.2, random_state=42
)
```

The dataset was split again using only the RFE selected features.

