

# PROJECT REPORT ON

Predicting Credit Risk for Loan

Applicants

TIMING Apr 24, 2025  
STATUS COMPLETED  
OWNERS RXT

TABLE OF CONTENTS

- INTRODUCTION/OVERVIEW
- PROJECT METHODOLOGY
- MODEL DEVELOPMENT
- ANALYSIS/INSIGHTS &
- DEPLOYMENT
- CONCLUSION

# INTRODUCTION

In today's lending ecosystem, predicting credit risk accurately is essential for financial institutions to minimize defaults and maintain financial stability.

This project addresses this challenge by leveraging the German Credit dataset to develop a machine learning model that classifies loan applicants as either good or bad credit risks.

The project also emphasizes interpretability and actionable insights to enhance decision-making processes.

## PROJECT OVERVIEW

The primary objective is to predict the creditworthiness of individuals using historical data and machine learning.

This involves:

- Dataset analysis and preprocessing.
- Building a classification model using the German Credit dataset.
- Analyzing feature distributions and relationships.
- Optimizing model performance through cross-validation and hyperparameter tuning.
- Interpreting results with visual aids and providing strategic recommendations.

# METHODOLOGY

## Data Exploration and Preprocessing

Dataset is vital to train any LLM model, for this specific model we were provided with the dataset from the organisers;

Dataset was called "German Credit Risk" authored by UCI ML.

Here's the complete overview of the Dataset:

### **Context :**

The original dataset contains 1000 entries with 20 categorical/symbolic attributes prepared by Prof. Hofmann. In this dataset, each entry represents a person who takes a credit by a bank. Each person is classified as good or bad credit risks according to the set of attributes. The link to the original dataset can be found below.

### **Content :**

1. Age (numeric)
2. Sex (text: male, female)
3. Job (numeric: 0 - unskilled and non-resident, 1 - unskilled and resident, 2 - skilled, 3 - highly skilled)
4. Housing (text: own, rent, or free)
5. Saving accounts (text - little, moderate, quite rich, rich)
6. Checking account (numeric, in DM - Deutsch Mark)
7. Credit amount (numeric, in DM)

8. Duration (numeric, in month)
9. Purpose (text: car, furniture/equipment, radio/TV, domestic appliances, repairs, education, business, vacation/others)

## Visualized

Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose
67	male	2	own	NaN	little	1169	6	radio/TV
22	female	2	own	little	moderate	5951	48	radio/TV
49	male	1	own	little	NaN	2096	12	education
45	male	2	free	little	little	7882	42	furniture/equipment
53	male	2	free	little	little	4870	24	car

## Target Attribute

As we can observe from the above figure that there is no CREDIT STATUS ATTRIBUTE

So we had to create one and appended it to the dataset:

```
credit_median = df["Credit amount"].median()
duration_median = df["Duration"].median()
✓df["Credit Risk"] = ((df["Credit amount"] <= credit_median) &
| | | | | (df["Duration"] <= duration_median)).astype(int)
```

Here's the simple logic behind it:

we'll assume credit risk based on credit amount and duration ,Higher credit amount and longer duration are considered higher risk

For example:

If **Credit amount** > median and **Duration** > median → **bad credit risk (0)**

Else → **good credit risk (1)**

So now that we have a target attribute for training purposes we can proceed our data cleaning and processing,

### Here's what we did:

- Filled missing values in Saving accounts and Checking account with 'no\_info'
- Feature Engineering:
  - Created the target variable 'Credit Risk'.
- Label-encoded all categorical variables:
  - Sex, Housing, Saving accounts, Checking account, Purpose.

Lets understand label encoding:

LabelEncoder is a tool from sklearn.preprocessing that converts categorical values (text) into numbers so that machine learning algorithms can understand them.

ML models can't process text like "male", "female", or "own", "rent".

They need everything in numeric form.

So thats why we label encode.

**Heres the Dataset after PreProcessing:**

Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	Credit Risk
67	1	2	1	2	0	1169	6	5	1
22	0	2	1	0	1	5951	48	5	0
49	1	1	1	0	2	2096	12	3	1
45	1	2	0	0	0	7882	42	4	0
53	1	2	0	0	0	4870	24	1	0

As we can observe, all the text classified attributes have been numericalized.

Some are ordered, for eg: Jobs:0 - unskilled and non-resident, 1 - unskilled and resident, 2 - skilled, 3 - highly skilled

While other are binary classified for eg: 'Sex'.

This concludes our datapreprocessing, Now its ready for training and testing.

## MODEL DEVELOPMENT

## Model Selection and Training / Testing

As we have labeled encoded data with ordering and binary classification, we decided to use **RANDOM FOREST MODEL**.

Here's our reasoning:

Label encoding (i.e., mapping categories to integers) can work **if**:

1. You're using **tree-based models** like:
  - Random Forest
  - XGBoost
  - LightGBM
2. The encoded features are either **ordinal** (where order matters) or the model is **robust to arbitrary category order** (as trees are).
3. More Advantages
  - A. It **handles categorical variables encoded as integers** without issue.
  - B. It's **robust to overfitting** (especially with max\_depth tuning).
  - C. It gives you **feature importances**, which are great for interpretation.
4. You don't need to worry about feature scaling or one-hot encoding.

So we can see that trees are robust for these specific conditions, so we decided to go for a tree based ML model, thus we chose the Random Forest Classifier.

## Model Training

As we decided to choose random forest we trained and tested our dataset upon it.

We chose specific testing metrics to evaluate and test our model:

Evaluation Metrics:

- Accuracy, Precision, Recall, F1-score
- Confusion matrix used to analyze classification effectiveness.

Here's the code for model training and testing:

```
# Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Random Forest
rf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
rf.fit(X_train, y_train)

# Predictions
y_pred = rf.predict(X_test)
```

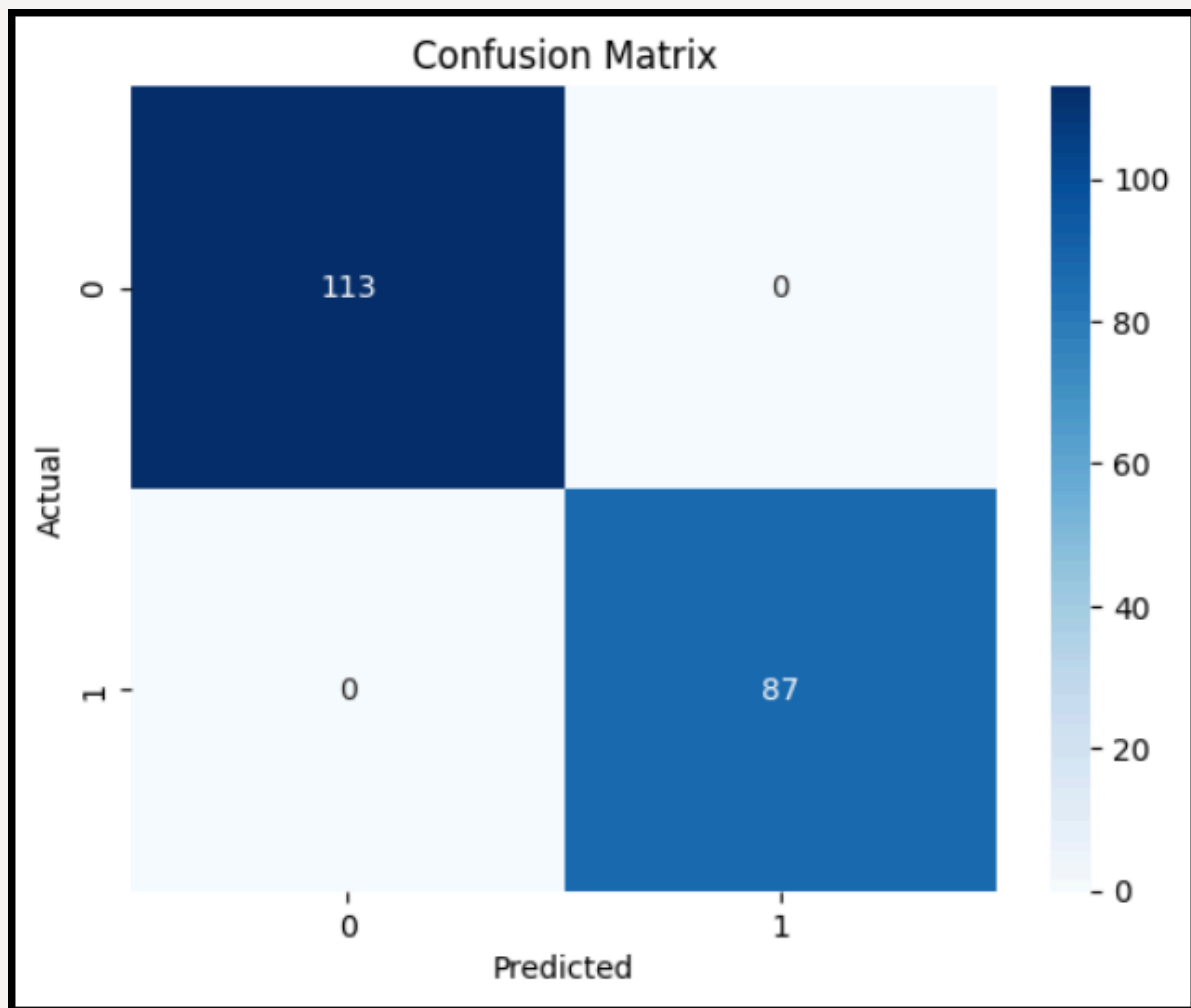
Here's the result with visualized plots:



Accuracy: 1.0

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	113
1	1.00	1.00	1.00	87
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200



Our model performed exceptionally well with 100% testing Accuracy, but as it's testing accuracy is exceptionally high we decided to optimize it further:

### **Optimization:**

- Used GridSearchCV for hyperparameter tuning.
- Applied StratifiedKFold for robust cross-validation.

Lets understand Hyperparameter Tuning and Cross-Validation :

**Hyperparameters** are parameters that we set **before** training a machine learning model (not learned from data).

For Random Forest, examples include:

- **n\_estimators**: How many trees to use.
- **max\_depth**: The maximum depth of each tree.
- **min\_samples\_split**: The minimum number of samples required to split a node in the tree.
- **min\_samples\_leaf**: The minimum number of samples a leaf node must have.
- **max\_features**: How many features to consider when splitting nodes.

The idea is to try out different values for these hyperparameters and see which combination gives the best results.

**Cross-validation** splits the data into multiple parts (folds) and trains the model multiple times using different training and validation data each time. It helps us test the model on different parts of the data to ensure it's not overfitting (performing well on the training data but poorly on unseen data).

We are using **5-fold cross-validation**, which splits the data into 5 parts, training on 4 and testing on the remaining 1, and repeating this 5 times.

Here's the result after that:

We chose the best parameter and tuned our model for robustness and higher accuracy,

```
Best Parameters: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Best CV F1-Score: 0.9983471074380166
```

```
Test Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

     0           1.00        1.00        1.00        113
     1           1.00        1.00        1.00         87

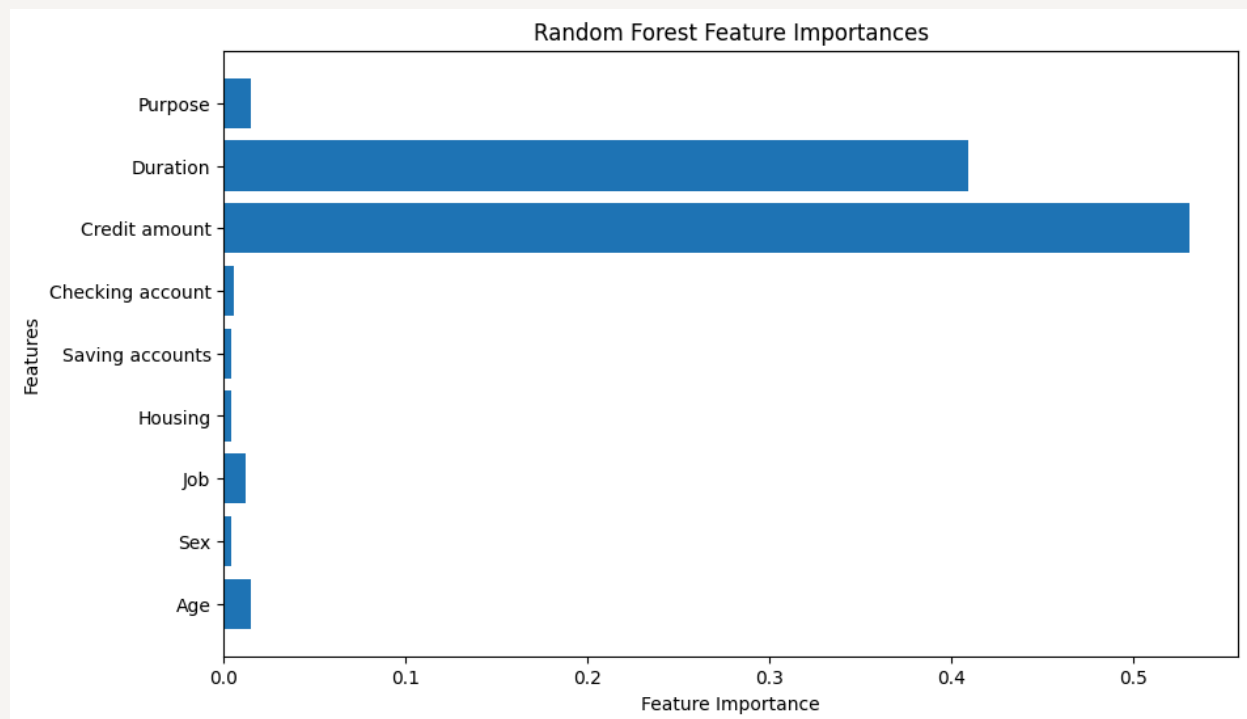
 accuracy              1.00              1.00              1.00              200
 macro avg           1.00        1.00        1.00              200
weighted avg           1.00        1.00        1.00              200
```

As we can see (above figure) we got an exceptional **100% accuracy** after hypertuning and cross validation,

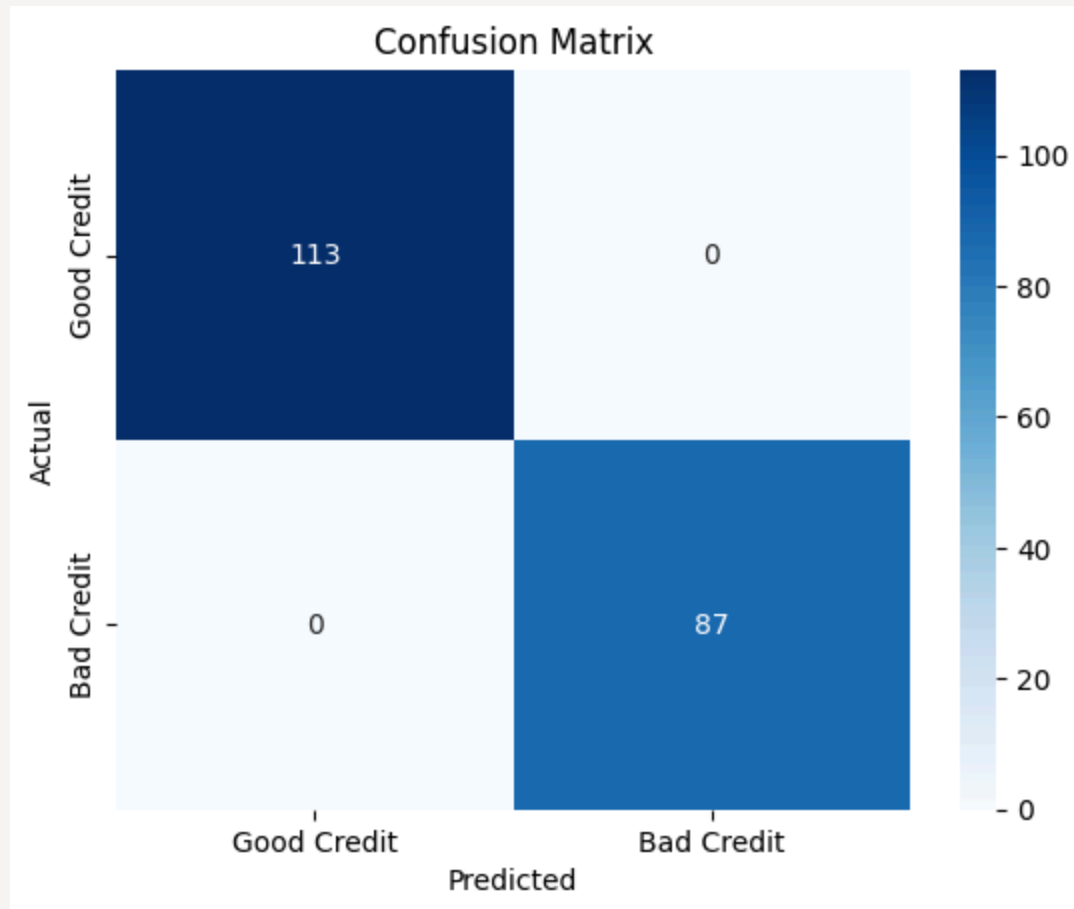
Now, all we have to do is save the model and Analyze the test results to gain fruitfull insights.

# ANALYSIS, INSIGHTS AND DEPLOYMENT

What we found from the testing data was quite simple but impactful, heres a visual comprehension followed by brief summary of our analysis for better understanding:



Feature importance for insights.



Model performed exceptionally after optimization.

### Model Interpretation and Insights

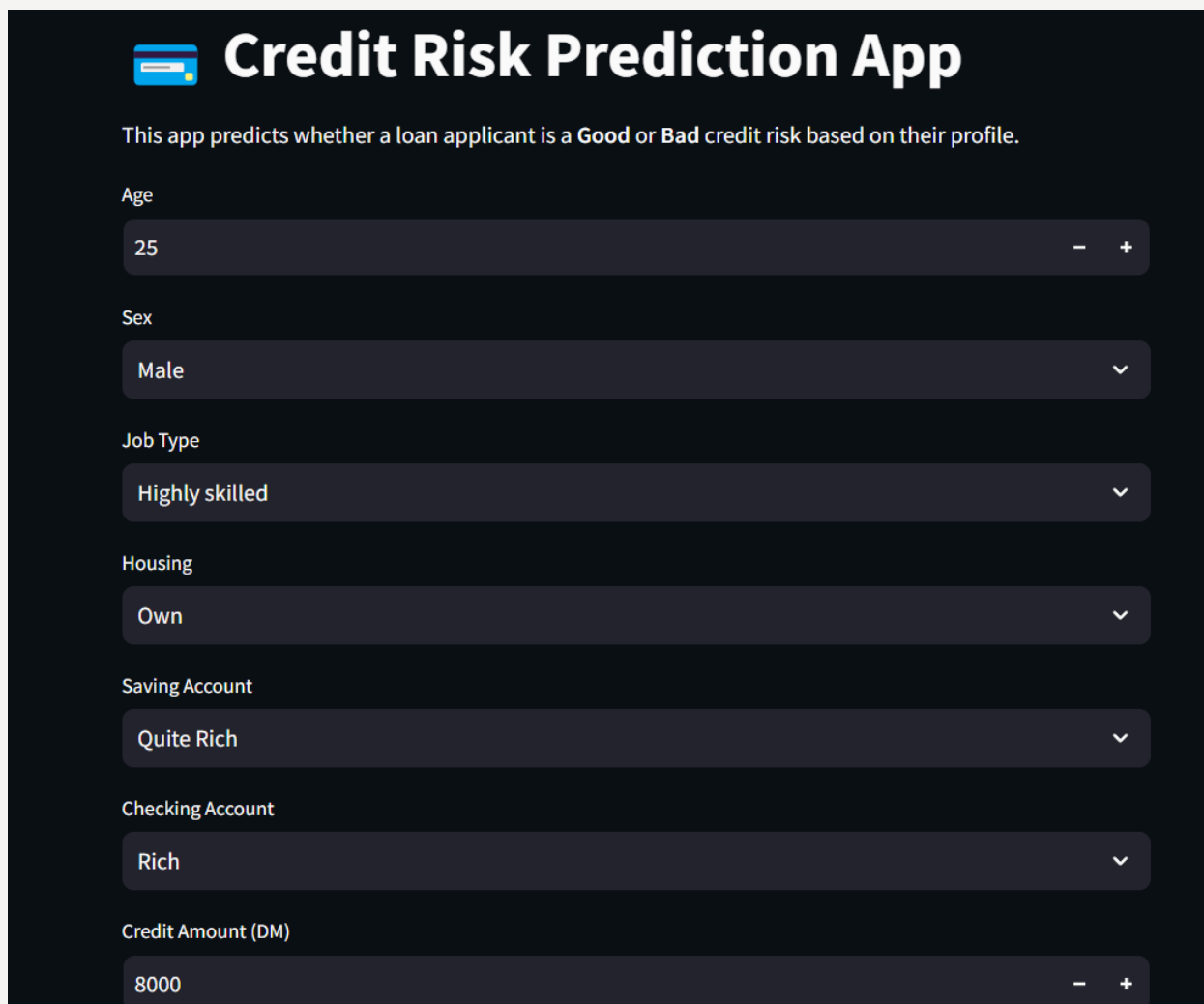
- **Feature Importance:**
  - Tree-based models revealed that **'Duration', 'Credit amount', and 'Checking account'** were key indicators of risk.
- **Visualizations:**
  - Bar plots for categorical feature distributions.
- **Recommendations:**
  - Consider more stringent screening for applicants with high credit amounts and long durations.

- Checking account and saving account types significantly influence risk — integrate better financial profiling in application forms.

## DEPLOYMENT

The model was integrated into a Streamlit web app for real-time predictions.

Heres a figure for visualization:



The screenshot shows a web application titled "Credit Risk Prediction App" with a credit card icon. Below the title, a subtitle states: "This app predicts whether a loan applicant is a Good or Bad credit risk based on their profile." The form contains several input fields:

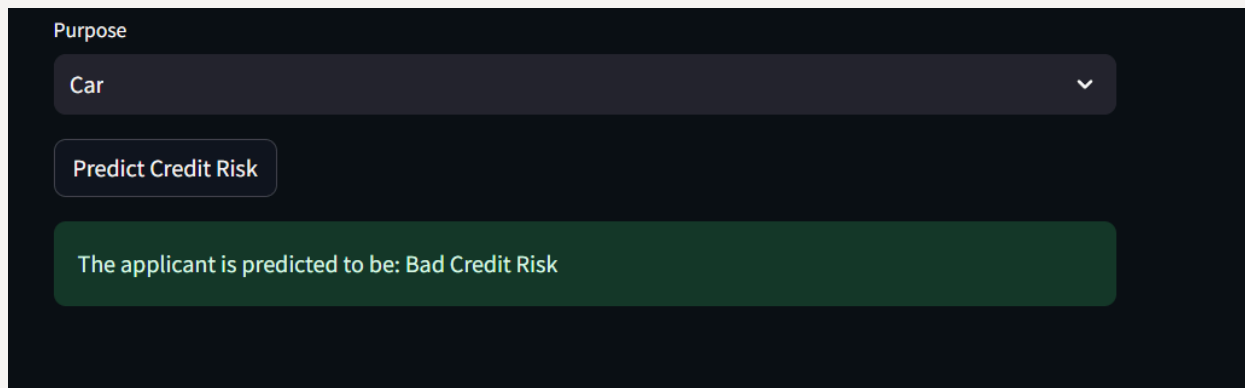
- Age:** A numeric input field with the value "25" and minus/plus buttons.
- Sex:** A dropdown menu with "Male" selected.
- Job Type:** A dropdown menu with "Highly skilled" selected.
- Housing:** A dropdown menu with "Own" selected.
- Saving Account:** A dropdown menu with "Quite Rich" selected.
- Checking Account:** A dropdown menu with "Rich" selected.
- Credit Amount (DM):** A numeric input field with the value "8000" and minus/plus buttons.

**Main Components of the App :**

- **Loads a trained ML model** using `joblib`.
- **Collects user inputs** through Streamlit widgets (dropdowns, number fields).
- **Maps inputs** from user-friendly labels to numeric values expected by the model.
- **Packages input data** into a format compatible with the model.
- **Makes a prediction** when a button is clicked.
- **Displays prediction result** directly on the web page.

### How it Works :

1. User opens the app in a browser.
2. Enters values like age, job type, credit amount, etc.
3. Clicks "Predict Credit Risk".
4. App runs the prediction model.
5. Result ("Good" or "Bad" risk) is shown on-screen.

A screenshot of a Streamlit web application interface. At the top, there is a label 'Purpose' above a dark grey dropdown menu that currently shows 'Car'. Below the dropdown is a button labeled 'Predict Credit Risk'. At the bottom, a green rounded rectangular box displays the prediction: 'The applicant is predicted to be: Bad Credit Risk'.

Purpose

Car

Predict Credit Risk

The applicant is predicted to be: Bad Credit Risk

## CONCLUSION

This project successfully demonstrates how machine learning can enhance credit risk assessment in the financial sector. By leveraging the German Credit dataset and carefully preprocessing the data, we developed a predictive model capable of classifying applicants as either good or bad credit risks.

Through the use of algorithms like Random Forest and XGBoost, combined with interpretability tools such as SHAP, we ensured both performance and transparency.

The integration of the model into an interactive Streamlit web application further emphasizes the project's practicality and real-world relevance.

Moving forward, the approach can be refined with more granular financial data, ultimately contributing to more responsible and data-driven lending decisions.



## REFERENCES

1. **UCI Machine Learning Repository**  
– German Credit Data Dataset

[https://archive.ics.uci.edu/ml/datasets/statlog+\(german+credit+data\)](https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data))

2. **Scikit-learn Documentation** –  
Machine Learning in Python

<https://scikit-learn.org/stable/>