Final Project

GERMAN BANK LOAN

Tushar Shrivastava

Introduction

In the world of banking, predicting loan defaulters is a critical task for banks to manage risk and ensure profitability. The challenge the bank faces with the loan defaulters necessitates a deeper understanding of the factors contributing to loan defaults. By developing different machine learning models to predict whether a customer will default on a loan, the bank can minimize the risk factor and can help in significantly improve the decision making and minimizing the financial losses.

The primary objective of this project is to use historical data from a German bank to build a model that predict loan default. The project aims to identify key features within the dataset that are most indicative of defaulters and evaluate different machine learning models for their performance in prediction.

Through this project, we aim to provide answers to key research question: Which machine learning model provides the best accuracy? Additionally, we will evaluate the model's performance using relevant metrics and identify the most effective approaches for managing loan default risk.

Methods and Materials

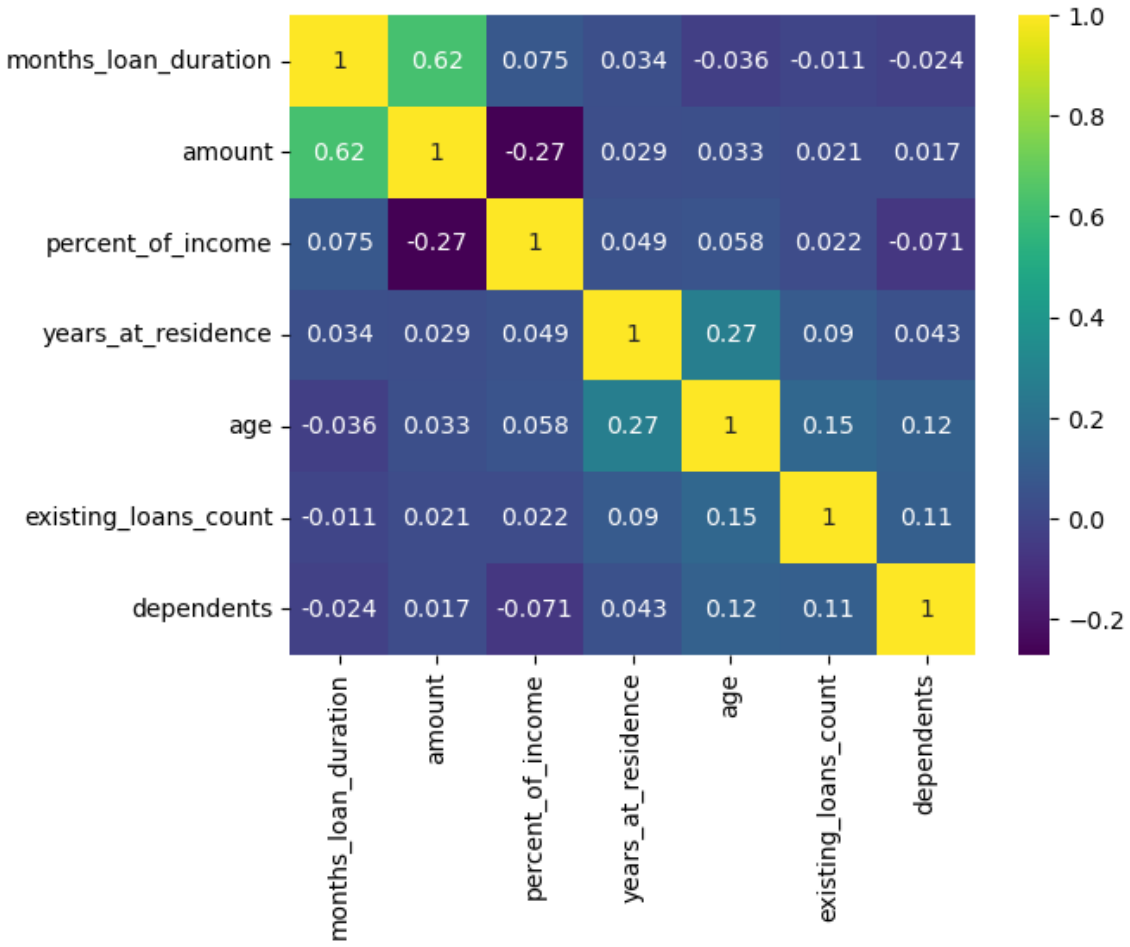
The dataset provided by the bank consists of 17 columns and 1000 rows, containing information on customers who have taken loans. The columns include features such as checking balance, loan duration, credit history, loan purpose, loan amount, savings balance, employment duration, percentage of income, years at residence, age, other credit, housing, existing loans count, job type, dependents, phone ownership, and default status.

To prepare the data for analysis, we conducted data preprocessing steps such as handling missing values, encoding categorical variables, and scaling numerical features as needed. This process ensured the data was clean and ready for modeling.

Exploratory data analysis (EDA) was carried out to understand the dataset's characteristics and identify patterns that may influence loan default prediction. Visualizations such as histograms, boxplots, and correlation heatmaps provided insights into feature distributions and relationships.

In terms of modeling, we used a variety of machine learning algorithms, including logistic regression, decision trees, random forests, Gradient Boosting, Support Vector Machine, and LGBM classfier. We assessed model performance using metrics such as accuracy, precision, recall, F1 score,. Hyperparameter tuning and cross-validation were conducted to optimize model performance.

Heatmap:-



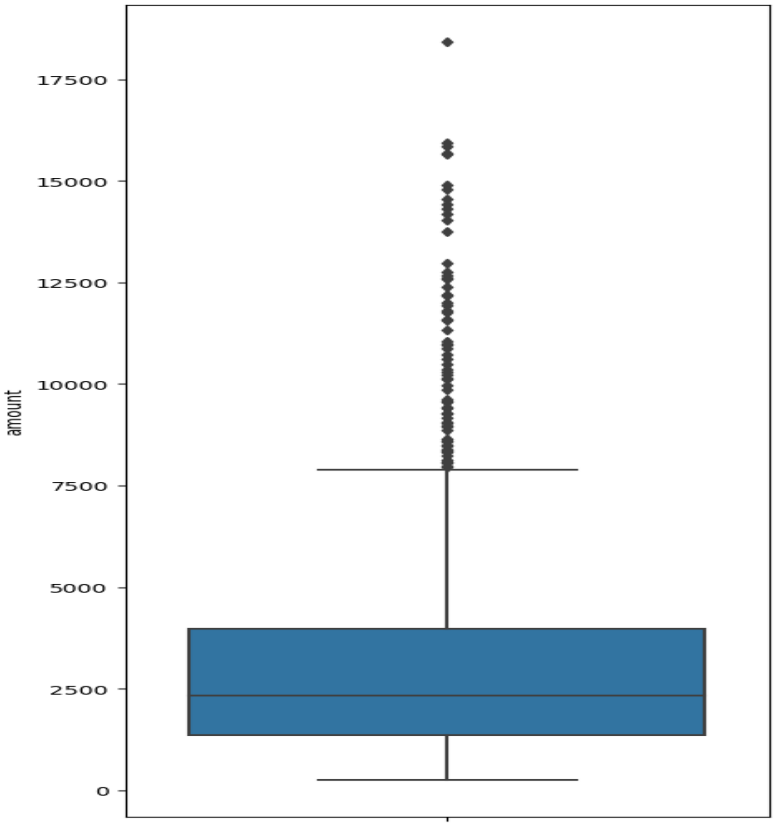
The above heatmap indicates that there is some strong positive correlation between predictors like 'Amount' and 'Months_loan_duration'. There is also some weak positive relation such as 'Years_at_residence' and 'Age', There is also depiction of some weak negative correlation such as 'Percent_of_income' and'Amount'.

Code snippet: -

```
# Checking the relationship of the various column in the dataset with each other
matrix = df.corr(numeric_only=True)
sns.heatmap(matrix,annot=True,cmap='viridis')
```

Boxplots: -

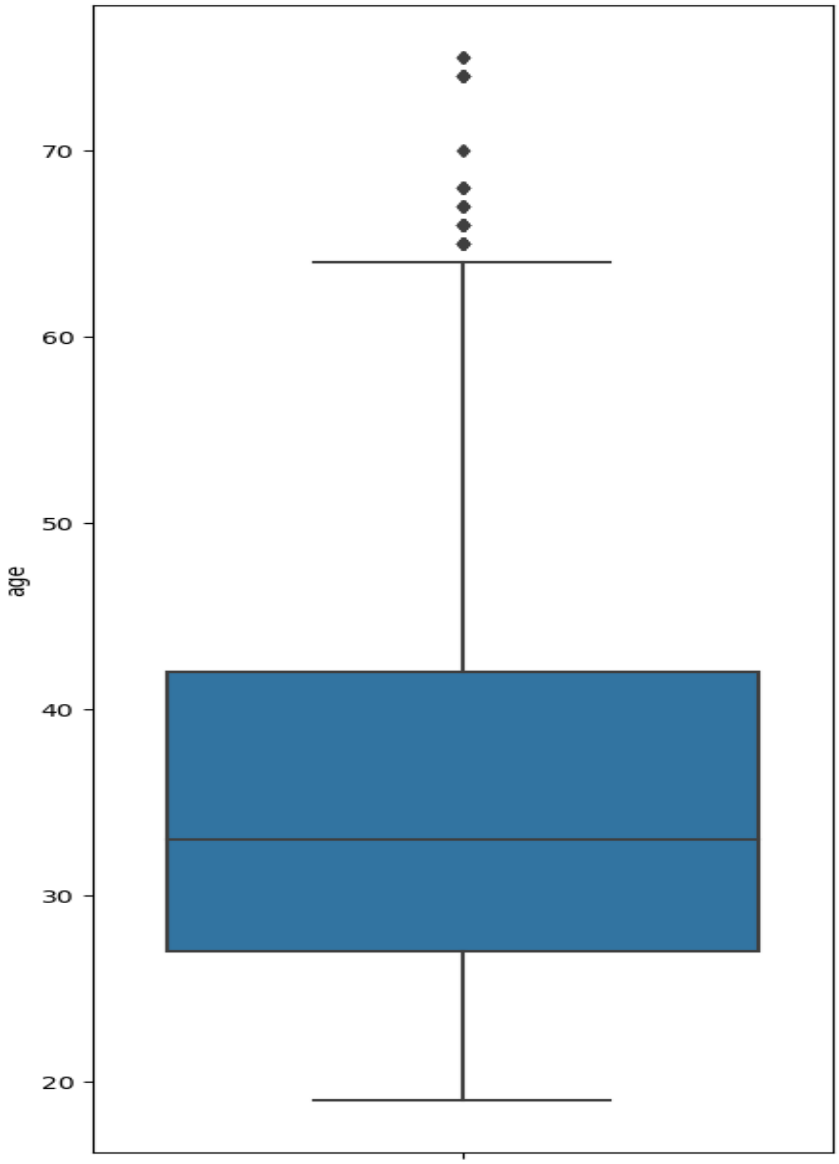
Boxplots were used in the numerical data('Age','Amount','Months_loan_duration') to identify Outliers present in the column.



The above graph is for column 'Amount' which depicts that there a lot of outliers present in the column.

Code snippet: -

```
# Checking Outliers in Numerical columns
plt.figure(figsize=(5,12))
sns.boxplot(y=df['amount'])
```

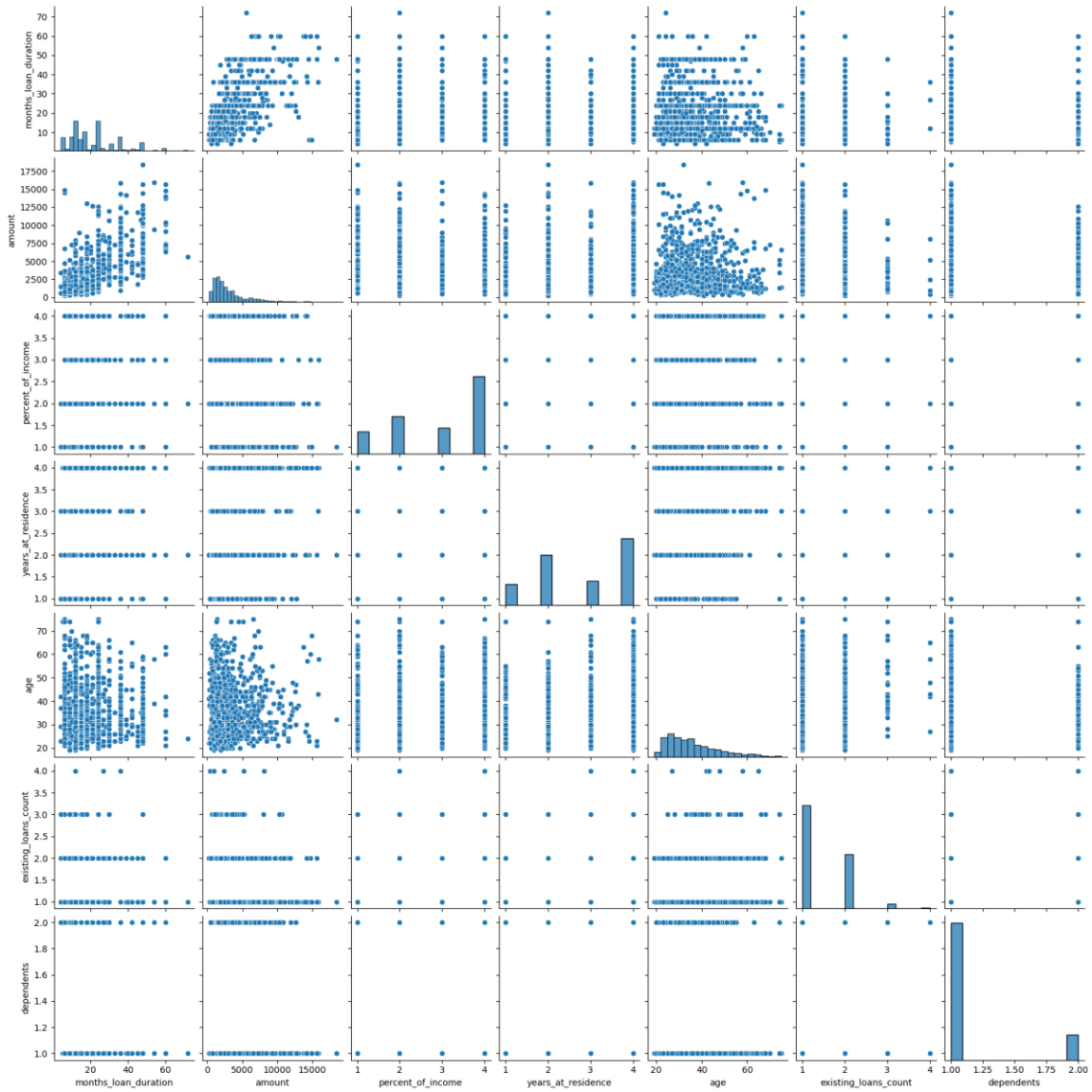


The above graph is for column 'Age' which depicts that there are outliers present in the column.

Code snippet: -

```
plt.figure(figsize=(5,12))
sns.boxplot(y=df['age'])
```

Pairplot: -



Pairplots are quickest and best way to plot out all the scatterplot amongst all columns to see relationships amongst each variable. The above plot shows all the depicted relationship between predictors and the histogram depicts the individual distribution of variables.

Code snippet: -

```
# Using pair plot from seaborn to determine the relationship
sns.pairplot(df)
```

Data Preprocessing: -

For columns which have binary categorical representation we changed it to numerical so that it can be used modeling as different model requires numerical data to fit the model

Code snippet: -

```
# Changing Binary categorical columns to numerical
df['phone'] = df['phone'].map({'yes':1,'no':0})
df['default'] = df['default'].map({'yes':1,'no':0})
df.head()
```

For Categorical columns, they were divided into Nominal and Ordinal columns as each requires different techniques to handle the data. We used One-hot encoding implemented by Pandas to handle Nominal values as shown below

Code snippet: -

```
# Classifying columns into Nominal and Ordinal categorical columns for preprocessing
ordinal_clmn = ['checking_balance','credit_history','savings_balance','employment_duration',]
nominal_clmn = ['purpose','other_credit','housing','job',]
```

```
# Using One-hot encoding for Nominal values
df_enc = pd.get_dummies(df,columns=nominal_clmn,dtype=int,drop_first=True)
df_enc
```

For Ordinal Values we use Label encoder from scikit library. As ordinal values have some order so it needs to be handle accordingly hence Label encoder assigns integers with specific order in order to maintain the ordinality of the data.

Code snippet: -

```
# Using Label encoding for Ordinal values
le = LabelEncoder()
```

```
for i in ordinal_clmn:
    df_enc[i] = le.fit_transform(df_enc[i])

df_enc.head()
```

Results: -

In terms of modeling, we used a variety of machine learning algorithms, including logistic regression, decision trees, random forests, and others. We assessed model performance using metrics such as accuracy, precision, recall, F1 score. Hyperparameter tuning and cross-validation were conducted to optimize model performance.

First the models were evaluated based on the average accuracy score from K-fold cross validation where k=5.

Code snippet: -

```
# Using K-fold cross validation to find best model
X = df_enc.drop(['default'],axis=1)
y = df_enc['default']
```

```
# Creating the instances for all the models
lomodel = LogisticRegression(random_state=42)
rmodel = RandomForestClassifier(random_state=42)
gmodel = GradientBoostingClassifier(random_state=42)
smodel = SVC(random_state=42)
dmodel = DecisionTreeClassifier(random_state=42)
lgmodel = LGBMClassifier(random_state=42)
```

```
kf = KFold(n_splits=5,shuffle=True,random_state=42)
```

Logistic Regression: -

```
# Logistic regression
score1 = cross_val_score(lomodel,X,y,cv=kf,scoring='accuracy')
mean_acc1 = np.mean(score1)
mean_acc1
```

```
0.7180000000000001
```

Random Forest Classifier: -

```
# Random Forest Classifier
score2 = cross_val_score(rmodel,X,y,cv=kf,scoring='accuracy')
mean_acc2 = np.mean(score2)
mean_acc2
```

```
0.761
```

Gradient Boosting Classifier: -

```python
# Gradient Boosting Classifier
score3 = cross_val_score(gmodel,X,y,cv=kf,scoring='accuracy')
mean_acc3 = np.mean(score3)
mean_acc3
```

```
0.7649999999999999
```

Support Vector Machine: -

```python
# Support Vector Machine
score4 = cross_val_score(smodel,X,y,cv=kf,scoring='accuracy')
mean_acc4 = np.mean(score4)
mean_acc4
```

```
0.7
```

Decision Tree classifier: -

```python
# Decision Tree classifier
score5 = cross_val_score(dmodel,X,y,cv=kf,scoring='accuracy')
mean_acc5 = np.mean(score5)
mean_acc5
```

```
0.699
```

LGBM classifier: -

```python
# LGBM classifier
score6 = cross_val_score(lgmodel,X,y,cv=kf,scoring='accuracy')
mean_acc6 = np.mean(score6)
mean_acc6
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

0.748
```

Then we used classical approach by evaluating through metrics such as accuracy, precision, recall, F1 score.

A function was created which can be used for each model to generate all the metrics which is application of Encapsulation which is key aspect of good coding structure.

We split the data with 80% train data and 20% test data and used stratified sampling so that the class will be balanced

Code snippet: -

```
# Using classic metrics like accuracy, recall, F1 score and confusion matrix
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,stratify=y)
```

```
# Defining a function to produce the metrics for the models
def metrics_evaluation(model):
    y_pred1 = model.predict(X=X_test)
    y_pred2 = model.predict(X=X_train)
    acc_score1 = accuracy_score(y_true=y_test,y_pred=y_pred1)
    matrix1 = confusion_matrix(y_true=y_test,y_pred=y_pred1)
    recall_score1 = recall_score(y_true=y_test,y_pred=y_pred1)
    f1_score1 = f1_score(y_true=y_test,y_pred=y_pred1)
    precision_score1 = precision_score(y_true=y_test,y_pred=y_pred1)
    acc_score2 = accuracy_score(y_true=y_train,y_pred=y_pred2)
    recall_score2 = recall_score(y_true=y_train,y_pred=y_pred2)
    f1_score2 = f1_score(y_true=y_train,y_pred=y_pred2)
    precision_score2 = precision_score(y_true=y_train,y_pred=y_pred2)

    print("Training performance:")
    print('The accuracy score is :-',acc_score2)
    print ('The recall_score is :-',recall_score2)
    print ('The f1_score is :-',f1_score2)
    print ('The precision score is :-',precision_score2)
    print("************************************")

    print("Test performance:")
    print('The accuracy score is :-',acc_score1)
    print ('The recall_score is :-',recall_score1)
    print ('The f1_score is :-',f1_score1)
    print ('The precision score is :-',precision_score1)
    print ('The confusion matrix is :-',matrix1)
```

Logistic Regression: -

```
# Logistic regression
lomodel.fit(X=X_train,y=y_train)
```

```
LogisticRegression(random_state=42)
```

```
metrics_evaluation(lomodel)
```

```
Training performance:
The accuracy score is :- 0.725
The recall_score is :- 0.30833333333333335
The f1_score is :- 0.40217391304347827
The precision score is :- 0.578125
************************************
Test performance:
The accuracy score is :- 0.765
The recall_score is :- 0.36666666666666664
The f1_score is :- 0.48351648351648346
The precision score is :- 0.7096774193548387
The confusion matrix is :- [[131   9]
 [ 38  22]]
```

Random Forest Classifier: -

```
# Random Forest Classifier
rmodel.fit(X=X_train,y=y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
metrics_evaluation(rmodel)
```

```
Training performance:
The accuracy score is :- 1.0
The recall_score is :- 1.0
The f1_score is :- 1.0
The precision score is :- 1.0
**************************************
Test performance:
The accuracy score is :- 0.78
The recall_score is :- 0.36666666666666664
The f1_score is :- 0.5
The precision score is :- 0.7857142857142857
The confusion matrix is :- [[134   6]
 [ 38  22]]
```

Gradient Boosting Classifier: -

```
# Gradient Boosting Classifier
gmodel.fit(X=X_train,y=y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

```
metrics_evaluation(gmodel)
```

```
Training performance:
The accuracy score is :- 0.89875
The recall_score is :- 0.7291666666666666
The f1_score is :- 0.8120649651972158
The precision score is :- 0.9162303664921466
**************************************
Test performance:
The accuracy score is :- 0.77
The recall_score is :- 0.4666666666666667
The f1_score is :- 0.5490196078431373
The precision score is :- 0.6666666666666666
The confusion matrix is :- [[126  14]
 [ 32  28]]
```

Support Vector Machine: -

```
# Support Vector Machine
smodel.fit(X=X_train,y=y_train)
```

```
SVC
SVC(random_state=42)
```

```
metrics_evaluation(smodel)
```

```
Training performance:
The accuracy score is :- 0.7
The recall_score is :- 0.0
The f1_score is :- 0.0
The precision score is :- 0.0
**************************************
Test performance:
The accuracy score is :- 0.7
The recall_score is :- 0.0
The f1_score is :- 0.0
The precision score is :- 0.0
The confusion matrix is :- [[140   0]
 [ 60   0]]
```

Decision Tree classifier: -

```
# Decision Tree Classifier
dmodel.fit(X=X_train,y=y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
metrics_evaluation(dmodel)
```

```
Training performance:
The accuracy score is :- 1.0
The recall_score is :- 1.0
The f1_score is :- 1.0
The precision score is :- 1.0
**************************************
Test performance:
The accuracy score is :- 0.625
The recall_score is :- 0.48333333333333334
The f1_score is :- 0.4360902255639098
The precision score is :- 0.3972602739726027
The confusion matrix is :- [[96 44]
 [31 29]]
```

LGBM classifier: -

```
# LGBM classifier
lgmodel.fit(X=X_train,y=y_train)
```

```
LGBMClassifier
LGBMClassifier(random_state=42)
```

```
metrics_evaluation(lgmodel)
```

```
Training performance:
The accuracy score is :- 0.99875
The recall_score is :- 1.0
The f1_score is :- 0.997920997920998
The precision score is :- 0.995850622406639
**************************************
Test performance:
The accuracy score is :- 0.74
The recall_score is :- 0.45
The f1_score is :- 0.5094339622641509
The precision score is :- 0.5869565217391305
The confusion matrix is :- [[121  19]
 [ 33  27]]
```

Logistic regression provided a solid baseline with good accuracy and interpretability. Decision trees offered more flexibility in capturing complex patterns but risked overfitting. Gradient Boosting demonstrated strong predictive power and generalization capabilities. The advanced technique like LGBM model also performed well and has comparable results with tree-based classifiers.

Discussion: -

The results suggest that the **Gradient Boosting** model offers the best performance in predicting loan defaults, with a decent level of accuracy and robustness. It has the most balanced Recall and f-1 score. The significance of features such as checking balance and loan amount aligns with existing literature on loan default prediction. The LGBM model also performed with great accuracy and has similar result as Gradient Boosting model.

The SVM model performed poorly, recall, F1-score, and precision are all 0.0, indicating that the model is unable to correctly identify any positive class instances (i.e., those belonging to the minority class). This may indicate that class dataset is highly imbalance which led to poor performance. The Tree based classifiers such as Decision Tree, Random Forest and Gradient Boosting are not susceptible to this and hence were effective to work with imbalance class.

One limitation of the study is the potential bias in the data, which may affect the model's generalizability. Additionally, model interpretability varies across algorithms, impacting the bank's ability to explain predictions to customers. Future research could explore the integration of additional data sources to enhance model performance and further improve risk management. Continued refinement of machine learning models and features can lead to even more accurate predictions.

Conclusion: -

In summary, this project successfully developed a machine learning model to predict loan defaults based on historical data from a German bank. The **Gradient Boosting** model showed the best performance, providing valuable insights for managing loan default risk. The findings have practical implications for the bank and contribute to the ongoing effort to optimize risk assessment and lending practices.