PHASE 3 CLASSIFICATION PROJECT by Benbellah Owino

1. Business Case

Client Mouz Bank is a rapidly expanding retail and commercial bank that prioritizes quick, ethical financing via online platforms. The bank is implementing data-driven algorithms to enhance credit choices, control risk, and guarantee equitable loan approvals in order to support its growing loan portfolio.

Business problem Our client Mouz bank is looking for a clever way to simplify and streamline the loan approval procedure while upholding ethical lending standards. In order to achieve this goal, we will create a prediction model that analyzes applicant data and more effectively evaluates risk in order to automate and assist credit decision-making. The following model will serve as the basis for increasing approval speed, decreasing manual labor, improving decision consistency, and supporting Mouz Bank's objective of providing quicker and more equitable loan outcomes.

Objectives

We want a model that satisfies the following thresholds

1. Minimum f1 score of 90%.
2. Recall f1 score of 90%.
3. Minimum precision score of 90%.

We have very high thresholds because the model will operate in a high risk high reward environment so we want to minimize the former while maximize the latter. Anything less than that is as good as useless. Also the model must be good in order not to affect customer satisfication negatively.

2. Imports And Data Loading

```
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 OneHotEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_squared_error, r2_score,
mean_absolute_error, accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import classification_report, roc_auc_score
```

```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv("./data/loan_approval_dataset.csv")
df.head()
```


```
{"columns":[{"name":"index","rawType":"int64","type":"integer"},
{"name":"loan_id","rawType":"int64","type":"integer"},{"name":"
no_of_dependents","rawType":"int64","type":"integer"},{"name":"
education","rawType":"object","type":"string"},{"name":"
self_employed","rawType":"object","type":"string"},{"name":"
income_annum","rawType":"int64","type":"integer"},{"name":"
loan_amount","rawType":"int64","type":"integer"},{"name":"
loan_term","rawType":"int64","type":"integer"},{"name":"
cibil_score","rawType":"int64","type":"integer"},{"name":"
residential_assets_value","rawType":"int64","type":"integer"},
{"name":"
commercial_assets_value","rawType":"int64","type":"integer"},{"name":"
luxury_assets_value","rawType":"int64","type":"integer"},{"name":"
bank_asset_value","rawType":"int64","type":"integer"},{"name":"
loan_status","rawType":"object","type":"string"}],"ref":"4aed7d46-
6f10-4af3-b00e-a3cc43aae4e2","rows":[["0","1","2"," Graduate","
No","9600000","29900000","12","778","2400000","17600000","22700000","8
000000"," Approved"],["1","2","0"," Not Graduate","
Yes","4100000","12200000","8","417","2700000","2200000","8800000","330
0000"," Rejected"],["2","3","3"," Graduate","
No","9100000","29700000","20","506","7100000","4500000","33300000","12
800000"," Rejected"],["3","4","3"," Graduate","
No","8200000","30700000","8","467","18200000","3300000","23300000","79
00000"," Rejected"],["4","5","5"," Not Graduate","
Yes","9800000","24200000","20","382","12400000","8200000","29400000","
5000000"," Rejected"]],"shape":{"columns":13,"rows":5}}
```


```
df = df.drop(columns=["loan_id"])
```

3. Data Understanding

About the Data The loan approval dataset is a collection of financial records and associated information used to determine the eligibility of individuals or organizations for obtaining loans from a lending institution. It includes various factors such as cibil score, income, employment status, loan term, loan amount, assets value, and loan status. This dataset is commonly used in machine learning and data analysis to develop models and algorithms that predict the likelihood of loan approval based on the given features.

Source https://www.kaggle.com/datasets/architsharma01/loan-approval-prediction-dataset

```
# Looking at the metadata of our dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4269 entries, 0 to 4268
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0    no_of_dependents          4269 non-null   int64
 1    education                 4269 non-null   object
 2    self_employed             4269 non-null   object
 3    income_annum              4269 non-null   int64
 4    loan_amount               4269 non-null   int64
 5    loan_term                 4269 non-null   int64
 6    cibil_score               4269 non-null   int64
 7    residential_assets_value  4269 non-null   int64
 8    commercial_assets_value   4269 non-null   int64
 9    luxury_assets_value       4269 non-null   int64
 10   bank_asset_value          4269 non-null   int64
 11   loan_status               4269 non-null   object
dtypes: int64(9), object(3)
memory usage: 400.3+ KB
```

From the results above we can see that the data has no null values. I can also identify some categorical columns that need to be one-hot encoded

```
# Looking at descriptive statistics of the numerical columns
df.describe()
```


```
{"columns":[{"name":"index","rawType":"object","type":"string"},
{"name":" no_of_dependents","rawType":"float64","type":"float"},
{"name":" income_annum","rawType":"float64","type":"float"},{"name":"
loan_amount","rawType":"float64","type":"float"},{"name":"
loan_term","rawType":"float64","type":"float"},{"name":"
cibil_score","rawType":"float64","type":"float"},{"name":"
residential_assets_value","rawType":"float64","type":"float"},
{"name":"
commercial_assets_value","rawType":"float64","type":"float"},{"name":"
luxury_assets_value","rawType":"float64","type":"float"},{"name":"
bank_asset_value","rawType":"float64","type":"float"}],"ref":"0b004671
-2b78-4e84-947a-b5c2bb585a80","rows":
[["count","4269.0","4269.0","4269.0","4269.0","4269.0","4269.0","4269.
0","4269.0","4269.0"],
["mean","2.4987116420707425","5059123.9166081045","15133450.456781447"
,"10.900445069102835","599.9360505973295","7472616.537830873","4973155
.3056922","15126305.926446475","4976692.433825252"],
["std","1.695910160711101","2806839.831818462","9043362.984842854","5.
7091872792452","172.43040073575904","6503636.587664101","4388966.08963
8461","9103753.665256497","3250185.3056957023"],
["min","0.0","200000.0","300000.0","2.0","300.0","-
100000.0","0.0","300000.0","0.0"],
["25%","1.0","2700000.0","7700000.0","6.0","453.0","2200000.0","130000
```

```
0.0","7500000.0","2300000.0"],
["50%","3.0","5100000.0","14500000.0","10.0","600.0","5600000.0","3700
000.0","14600000.0","4600000.0"],
["75%","4.0","7500000.0","21500000.0","16.0","748.0","11300000.0","760
0000.0","21700000.0","7100000.0"],
["max","5.0","9900000.0","39500000.0","20.0","900.0","29100000.0","194
00000.0","39200000.0","14700000.0"]],"shape":{"columns":9,"rows":8}}
```


4. Building our BASELINE model(Logistic Regression)

```
# Removing whitespaces from column names
df.columns = df.columns.str.strip()

df["loan_status"].value_counts()
```


```
{"columns":[{"name":"loan_status","rawType":"object","type":"string"},
{"name":"count","rawType":"int64","type":"integer"}],"ref":"adf6841c-
57a0-4e04-9cec-862f17123e24","rows":[[" Approved","2656"],["
Rejected","1613"]],"shape":{"columns":1,"rows":2}}
```


```
# Convert loan status to boolean where approved == True and rejected
== false
df['loan_status'] =  df['loan_status'].str.strip().map({'Approved':
True, 'Rejected': False})

# Splitting the data
df_mod = df.copy()
df_mod['loan_status'] =  df_mod['loan_status']
y = df_mod['loan_status']
X = df_mod.drop(columns=["loan_status"])
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

print(df.select_dtypes(include='number').columns)

Index(['no_of_dependents', 'income_annum', 'loan_amount', 'loan_term',
       'cibil_score', 'residential_assets_value',
'commercial_assets_value',
       'luxury_assets_value', 'bank_asset_value'],
      dtype='object')

df.select_dtypes(include=['object', 'category', 'bool']).columns

Index(['education', 'self_employed', 'loan_status'], dtype='object')
```

Below we define our continous numerical columns and categorical columns so that we can apply the appropriate preprocessing steps to both types of columns

```
num_cols = [ 'no_of_dependents', 'income_annum', 'loan_amount',
       'loan_term', 'cibil_score', 'residential_assets_value',
```

```
       'commercial_assets_value', 'luxury_assets_value',
'bank_asset_value']

cat_cols = ['education', 'self_employed']
```

Setting up pipelines

We set up a pipeline to put all our preprocessing steps together and prevent errors

```
# Setting up the preprocessing steps for our numerical and categorical
columns
# StandardScaling the numerical columns,
# One Hot encoding the categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), num_cols),
        ('cat', OneHotEncoder(drop="first"), cat_cols)
    ]
)

# Initializing our pipeline and providing our preprocessor and model
log_pipe = Pipeline(steps=[
    ("preprocess", preprocessor),
    ("logreg", LogisticRegression())
])

y_train
```


```
{"columns":[{"name":"index","rawType":"int64","type":"integer"},
{"name":"loan_status","rawType":"bool","type":"boolean"}],"ref":"8580c
544-c64b-4147-8bab-613e21d04a54","rows":[["3404","False"],
["781","True"],["3002","True"],["4047","False"],["3391","True"],
["4139","True"],["3214","False"],["2655","True"],["3558","False"],
["883","True"],["1091","True"],["382","True"],["3233","True"],
["3165","False"],["1674","False"],["2927","True"],["31","False"],
["2430","True"],["1912","False"],["4239","False"],["610","False"],
["570","False"],["343","False"],["3013","False"],["2389","False"],
["2290","True"],["3383","False"],["3639","True"],["2781","True"],
["2080","False"],["3996","False"],["3705","True"],["1135","False"],
["2344","True"],["3874","False"],["841","True"],["2805","True"],
["1563","True"],["965","True"],["528","True"],["176","True"],
["422","False"],["1684","True"],["3619","True"],["2267","True"],
["572","True"],["2567","True"],["288","True"],["1505","True"],
["2195","False"]],"shape":{"columns":1,"rows":2988}}
```


```
# Fitting our pipeline on the training data and predicting on the
testing data
log_pipe.fit(X_train, y_train)
y_pred = log_pipe.predict(X_test)
```

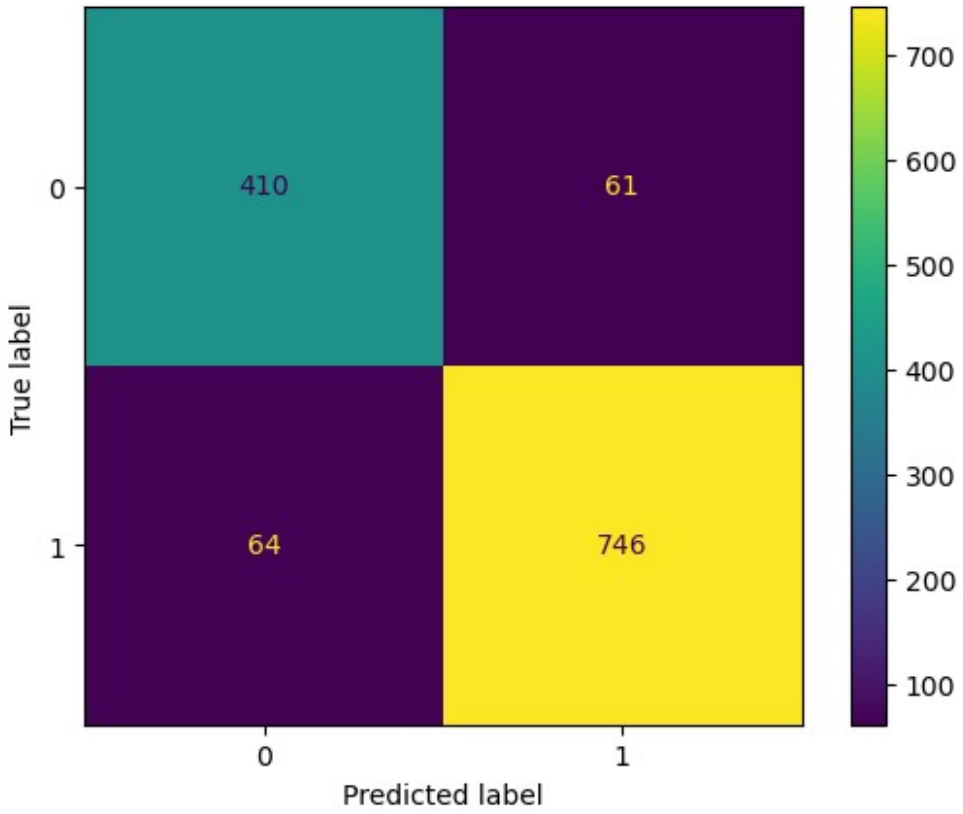
Model Evaluation

```
# Evaluate
def model_evaluate_pipe(model, X_train,X_test, y_test, y_pred):
    accuracy = model.score(X_test, y_test)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)  # Root mean squared error
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print("\nModel Performance Metrics:")
    print(f"R² Score: {r2:.4f}")  # Higher is better; 1.0 indicates
perfect prediction
    print(f"Root Mean Squared Error: {rmse:.2f}")  # Lower is better
    print(f"Mean Absolute Error: {mae:.2f}")  # Lower is better
    print("Accuracy:", accuracy )

    print("\n\n ========== Accuracy Scores=========")

    train_acc = accuracy_score(y_train, model.predict(X_train))
    test_acc  = accuracy_score(y_test, model.predict(X_test))

    print("Train accuracy:", train_acc)
    print("Test accuracy:", test_acc)


    print("\n\n ========== Error Scores=========")
    train_mae = mean_absolute_error(y_train, model.predict(X_train))
    test_mae  = mean_absolute_error(y_test,model.predict(X_test))

    print("Train MAE:", train_mae)
    print("Test MAE:", test_mae)

    print("\n\n ============================ Report
============================")
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    # Probabilities for each class if supported
    if hasattr(model, "predict_proba"):
        y_test_prob = model.predict_proba(X_test)[:, 1]
    else:
        y_test_prob = None

    print(classification_report(y_test, y_pred))
    print("ROC-AUC:", roc_auc_score(y_test, y_prob))

model_evaluate_pipe(model=log_pipe, X_train= X_train, X_test=X_test,
y_test=y_test, y_pred=y_pred)


Model Performance Metrics:
```

```
R² Score: 0.5803
Root Mean Squared Error: 0.31
Mean Absolute Error: 0.10
Accuracy: 0.9024199843871975


 ========== Accuracy Scores=========
Train accuracy: 0.9233601070950469
Test accuracy: 0.9024199843871975


 ========== Error Scores=========
Train MAE: 0.07663989290495314
Test MAE: 0.0975800156128025


 ============================ Report ============================
              precision    recall  f1-score   support

       False       0.86      0.87      0.87       471
        True       0.92      0.92      0.92       810

    accuracy                           0.90      1281
   macro avg       0.89      0.90      0.90      1281
weighted avg       0.90      0.90      0.90      1281

ROC-AUC: 0.9634924379439596
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)

ConfusionMatrixDisplay(confusion_matrix=cm).plot()
```

```
[[410  61]
 [ 64 746]]

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x23faad1a7b0>
```



Model Results

- The model has 746 TP, 410 TN, 61 FP and 64 FN.
- The model's test and training MAE's have a difference of 0.02 so the model generalizes well to new data. The model accuracys are also almost similar. The model is well fitted.
- The model is correct when it approves 93% of the time and when it rejects 87% of the time . This is good precision that doesn't significantly negatively affect customer satisfication.
- Recall for approval is 92% so it correctly approves good borrowers most of the time.
- Our ROC-AUC of 0.963 means the model separated the Approved vs Rejected extremely well.

This model fails to meet our f1 and recall goals especially for the negative case. Let's improve on that

5. Additional Models (Decision Tree Classifier)

Let's try to build a decision tree classifier and see if it improves our correcteness. We will start by building a Decision Tree with the default hyperparameters and a criterion of 1. I won't be using a pipeline for the next model. I'll do the preprocessing manually.

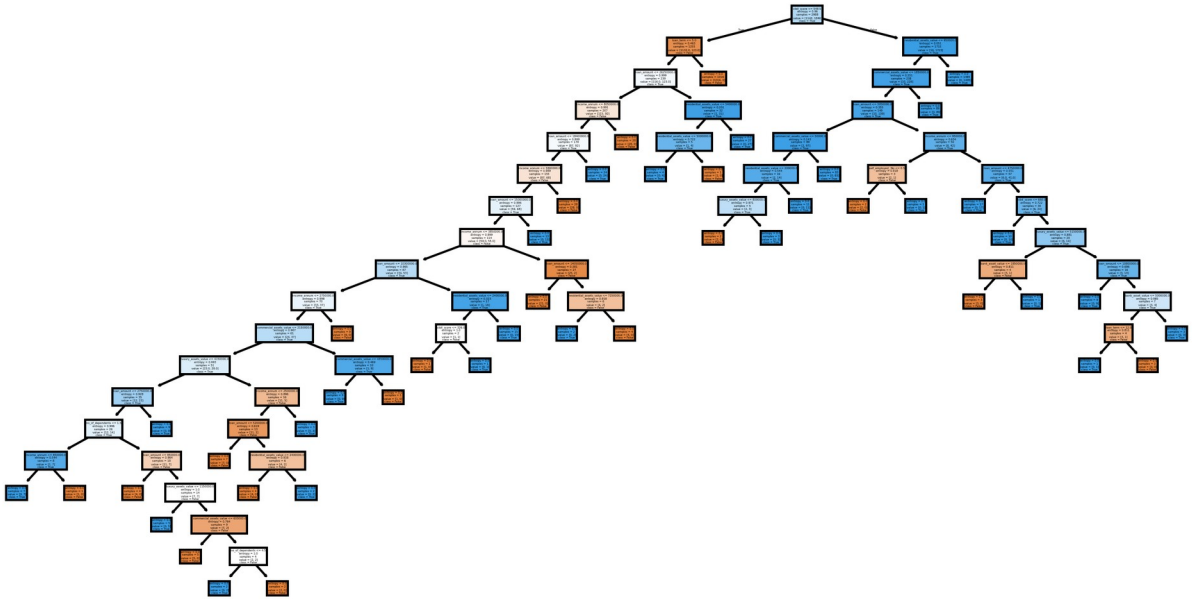
```
# Preparing the data
df_tree = df.copy()
yt = df_tree["loan_status"]
```

```
Xt = df_tree.drop(columns=["loan_status"])
Xt_train, Xt_test,yt_train, yt_test = train_test_split(Xt,
yt,test_size=0.3, random_state=42)

# One hot encoding the categorical variables
ohet = OneHotEncoder()

ohet.fit(Xt_train[cat_cols])
Xt_train_ohe = ohet.transform(Xt_train[cat_cols]).toarray()
ohe_df = pd.DataFrame(Xt_train_ohe,
columns=ohet.get_feature_names_out(cat_cols), index=Xt_train.index)

Xt_final = pd.concat(
    [Xt_train[num_cols], ohe_df],
    axis = 1
)
Xt_final.head()
```


```
{"columns":[{"name":"index","rawType":"int64","type":"integer"},
{"name":"no_of_dependents","rawType":"int64","type":"integer"},
{"name":"income_annum","rawType":"int64","type":"integer"},
{"name":"loan_amount","rawType":"int64","type":"integer"},
{"name":"loan_term","rawType":"int64","type":"integer"},
{"name":"cibil_score","rawType":"int64","type":"integer"},
{"name":"residential_assets_value","rawType":"int64","type":"integer"}
,
{"name":"commercial_assets_value","rawType":"int64","type":"integer"},
{"name":"luxury_assets_value","rawType":"int64","type":"integer"},
{"name":"bank_asset_value","rawType":"int64","type":"integer"},
{"name":"education_ Graduate","rawType":"float64","type":"float"},
{"name":"education_ Not Graduate","rawType":"float64","type":"float"},
{"name":"self_employed_ No","rawType":"float64","type":"float"},
{"name":"self_employed_
Yes","rawType":"float64","type":"float"}],"ref":"bf7a02ae-0982-4e2c-
b429-a3e01f767663","rows":
[["3404","4","2800000","8300000","14","381","3200000","1100000","91000
00","1500000","0.0","1.0","1.0","0.0"],
["781","3","9900000","20400000","4","865","26300000","3600000","365000
00","12800000","0.0","1.0","0.0","1.0"],
["3002","4","3500000","10700000","8","883","200000","2300000","8400000
","2000000","1.0","0.0","0.0","1.0"],
["4047","3","6400000","23000000","8","520","12100000","11400000","1600
0000","3800000","0.0","1.0","1.0","0.0"],
["3391","2","3300000","6700000","8","845","1300000","4600000","1230000
0","3200000","1.0","0.0","0.0","1.0"]],"shape":
{"columns":13,"rows":5}}
```


```
# Fitting the model
dtm1 = DecisionTreeClassifier(criterion="entropy")
dtm1.fit(Xt_final, yt_train)
```

```
DecisionTreeClassifier(criterion='entropy')

# Plot the decision tree
fig, axes = plt.subplots(nrows = 1, ncols=1, figsize = (10, 5),
dpi=300)
tree.plot_tree(dtm1,
               feature_names = Xt_final.columns,
               class_names=np.unique(y).astype('str'),
               filled = True)
plt.show()
```



The plot is complex and hard to read, but we have an idea of what the model looks like and its reasoning.

Model Evaluation

```
Xt_test_ohe = ohet.transform(Xt_test[cat_cols]).toarray()
ohe_ttest = pd.DataFrame(
    Xt_test_ohe,
    columns = ohet.get_feature_names_out(cat_cols),
    index = Xt_test.index
)

Xt_test_final = pd.concat(
    [Xt_test[num_cols], ohe_ttest],
    axis = 1
)

yt_preds = dtm1.predict(Xt_test_final)

print('Accuracy: ', accuracy_score(yt_test, yt_preds))
```

```
Accuracy:  0.9797033567525371
```

Xt_test_final


```
{"columns":[{"name":"index","rawType":"int64","type":"integer"},
{"name":"no_of_dependents","rawType":"int64","type":"integer"},
{"name":"income_annum","rawType":"int64","type":"integer"},
{"name":"loan_amount","rawType":"int64","type":"integer"},
{"name":"loan_term","rawType":"int64","type":"integer"},
{"name":"cibil_score","rawType":"int64","type":"integer"},
{"name":"residential_assets_value","rawType":"int64","type":"integer"}
,
{"name":"commercial_assets_value","rawType":"int64","type":"integer"},
{"name":"luxury_assets_value","rawType":"int64","type":"integer"},
{"name":"bank_asset_value","rawType":"int64","type":"integer"},
{"name":"education_ Graduate","rawType":"float64","type":"float"},
{"name":"education_ Not Graduate","rawType":"float64","type":"float"},
{"name":"self_employed_ No","rawType":"float64","type":"float"},
{"name":"self_employed_
Yes","rawType":"float64","type":"float"}],"ref":"18cc012a-528b-4da8-
a97a-d0929598c3f7","rows":
[["1703","5","5400000","19700000","20","423","6500000","10000000","157
00000","7300000","1.0","0.0","1.0","0.0"],
["1173","2","5900000","14000000","8","599","4700000","9500000","178000
00","6700000","1.0","0.0","1.0","0.0"],
["308","3","9600000","19900000","14","452","4200000","16200000","28500
000","6600000","1.0","0.0","1.0","0.0"],
["1322","2","6200000","23400000","8","605","10000000","10800000","2180
0000","9200000","1.0","0.0","1.0","0.0"],
["3271","3","5800000","14100000","12","738","11700000","4400000","1540
0000","8400000","0.0","1.0","0.0","1.0"],
["3539","4","4700000","12500000","8","678","13700000","200000","980000
0","7000000","1.0","0.0","0.0","1.0"],
["1522","4","3400000","13500000","12","705","10000000","1100000","8300
000","4900000","1.0","0.0","1.0","0.0"],
["3399","5","5100000","13700000","14","527","12100000","8900000","1940
0000","4700000","0.0","1.0","0.0","1.0"],
["1402","3","3300000","8500000","12","586","7500000","5100000","780000
0","3900000","1.0","0.0","0.0","1.0"],
["1829","1","3000000","6000000","16","518","3800000","1700000","116000
00","3900000","0.0","1.0","0.0","1.0"],
["296","5","8600000","28900000","8","516","9900000","1800000","3020000
0","6800000","0.0","1.0","0.0","1.0"],
["2700","5","2900000","6300000","6","719","7900000","1600000","8000000
","1600000","0.0","1.0","0.0","1.0"],
["471","4","3700000","11400000","12","728","4200000","5800000","730000
0","2300000","1.0","0.0","1.0","0.0"],
["3055","3","300000","500000","6","386","800000","200000","1200000","1
00000","0.0","1.0","0.0","1.0"],
["1123","0","9100000","18300000","14","458","6100000","9200000","34100
```

```
000","10000000","0.0","1.0","1.0","0.0"],
["2357","2","5200000","15400000","8","658","1200000","7500000","12800000","3200000","1.0","0.0","1.0","0.0"],
["4242","5","8200000","27100000","4","643","15400000","7700000","21300000","4100000","0.0","1.0","0.0","1.0"],
["3839","2","5100000","16800000","2","504","6400000","2700000","17100000","4400000","0.0","1.0","0.0","1.0"],
["1130","1","300000","1200000","4","342","800000","100000","1100000","200000","0.0","1.0","1.0","0.0"],
["3642","2","7600000","19300000","12","463","5400000","1500000","16000000","9600000","0.0","1.0","1.0","0.0"],
["862","2","4800000","10600000","8","588","3700000","5200000","15000000","4500000","0.0","1.0","1.0","0.0"],
["1338","0","1400000","3200000","14","384","3100000","800000","3400000","2000000","1.0","0.0","1.0","0.0"],
["3651","4","800000","2800000","12","453","1500000","300000","2900000","600000","0.0","1.0","1.0","0.0"],
["290","5","4300000","9300000","10","450","2100000","6700000","10000000","4100000","0.0","1.0","0.0","1.0"],
["3181","4","8000000","31500000","16","763","14900000","8800000","17600000","8900000","0.0","1.0","0.0","1.0"],
["3876","1","1600000","5100000","18","729","2500000","2000000","6200000","900000","0.0","1.0","1.0","0.0"],
["2411","5","3000000","8900000","20","886","2600000","1000000","10600000","2800000","0.0","1.0","1.0","0.0"],
["457","5","8400000","17500000","12","620","15300000","13600000","26000000","5800000","1.0","0.0","0.0","1.0"],
["555","5","6200000","22200000","20","819","13500000","6600000","21700000","8700000","0.0","1.0","0.0","1.0"],
["2905","5","1100000","2800000","14","473","100000","500000","4200000","700000","1.0","0.0","1.0","0.0"],
["1189","4","9600000","27500000","20","679","23400000","7500000","37800000","6300000","1.0","0.0","1.0","0.0"],
["1116","4","3800000","14900000","20","428","6500000","3500000","12200000","2500000","0.0","1.0","1.0","0.0"],
["3402","3","9200000","28900000","8","538","13900000","8300000","32400000","4900000","0.0","1.0","0.0","1.0"],
["817","0","5200000","18400000","14","600","7300000","9700000","16500000","3600000","0.0","1.0","1.0","0.0"],
["2107","4","6400000","18400000","16","320","17700000","3600000","13900000","8300000","0.0","1.0","0.0","1.0"],
["96","2","6600000","15000000","18","470","1200000","4800000","18200000","7000000","1.0","0.0","0.0","1.0"],
["2917","2","1700000","5700000","10","602","4900000","2700000","4500000","1800000","1.0","0.0","1.0","0.0"],
["4063","0","1000000","2800000","12","616","700000","700000","3900000","700000","1.0","0.0","0.0","1.0"],
["4000","0","7500000","25100000","14","877","21000000","11400000","16900000","10700000","1.0","0.0","1.0","0.0"],
```

["4119","4","6400000","19700000","20","400","2300000","6100000","20100000","6200000","0.0","1.0","1.0","0.0"],
["120","5","900000","2300000","16","545","100000","200000","2700000","800000","1.0","0.0","1.0","0.0"],
["2957","1","7200000","22800000","2","551","15000000","8700000","21300000","9500000","1.0","0.0","1.0","0.0"],
["2088","0","1400000","3700000","4","841","3000000","1000000","4600000","1000000","0.0","1.0","0.0","1.0"],
["1270","1","3500000","11600000","10","898","3100000","2700000","9000000","4500000","0.0","1.0","0.0","1.0"],
["3624","1","800000","2600000","2","439","600000","0","1800000","400000","0.0","1.0","0.0","1.0"],
["2409","0","7700000","24800000","10","642","16800000","10700000","27400000","4500000","1.0","0.0","1.0","0.0"],
["1051","0","1900000","6800000","4","782","100000","3400000","4900000","2400000","1.0","0.0","1.0","0.0"],
["166","4","5800000","12800000","2","656","500000","5000000","16000000","2800000","0.0","1.0","1.0","0.0"],
["1702","5","6100000","17800000","6","849","2900000","10900000","16100000","7800000","1.0","0.0","1.0","0.0"],
["2530","0","2400000","9100000","8","368","2200000","200000","5600000","3300000","0.0","1.0","0.0","1.0"]],"shape":{"columns":13,"rows":1281}}

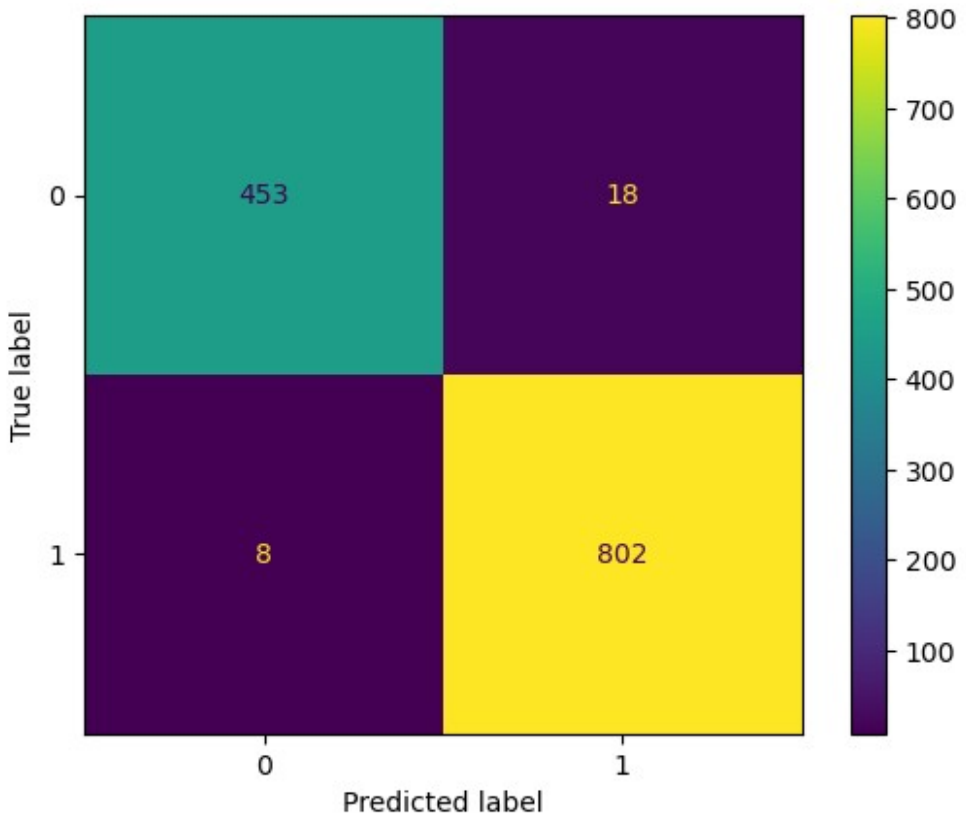
```
# Create a model evaluation function for non-pipeline models
# Evaluate
def model_evaluate_man(model,X_train,X_test, y_test, y_pred):
    # Predictions
    y_train_pred = model.predict(X_train)

    # Metrics
    accuracy = accuracy_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print("\nModel Performance Metrics:")
    print(f"R² Score: {r2:.4f}")
    print(f"Root Mean Squared Error: {rmse:.2f}")
    print(f"Mean Absolute Error: {mae:.2f}")
    print(f"Accuracy: {accuracy:.4f}")

    print("\n========== Accuracy Scores ==========")
    print("Train accuracy:", accuracy_score(y_train, y_train_pred))
    print("Test accuracy :", accuracy)

    print("\n========== Error Scores ==========")
    print("Train MAE:", mean_absolute_error(y_train, y_train_pred))
    print("Test MAE :", mae)
```

```
    print("\n============================== Report
============================")
    print(classification_report(y_test, y_pred))

    # Probabilities for each class if supported
    if hasattr(model, "predict_proba"):
        y_test_prob = model.predict_proba(X_test)[:, 1]
    else:
        y_test_prob = None

    if y_test_prob is not None:
        print("ROC-AUC:", roc_auc_score(y_test, y_test_prob))

model_evaluate_man(model = dtm1,X_train =
Xt_final,X_test=Xt_test_final, y_test=yt_test, y_pred=yt_preds)


Model Performance Metrics:
R² Score: 0.9127
Root Mean Squared Error: 0.14
Mean Absolute Error: 0.02
Accuracy: 0.9797

========== Accuracy Scores ==========
Train accuracy: 1.0
Test accuracy : 0.9797033567525371

========== Error Scores ==========
Train MAE: 0.0
Test MAE : 0.02029664324746292

============================== Report ============================
              precision    recall  f1-score   support

       False       0.98      0.96      0.97       471
        True       0.98      0.99      0.98       810

    accuracy                           0.98      1281
   macro avg       0.98      0.98      0.98      1281
weighted avg       0.98      0.98      0.98      1281

ROC-AUC: 0.9759534481402847
```

We see an overall increase in the perfomanc metrics. Let's look at the confusion matrix for this model.

```
cmt = confusion_matrix(yt_test, yt_preds)
print(cmt)
```

```
ConfusionMatrixDisplay(confusion_matrix=cmt).plot()

[[453  18]
 [  8 802]]

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x23fac3dda90>
```



Model Results

- The model has 800 TP, 452 TN, 19 FP and 10 FN.
- The model's test and training MAE's have a difference of 0.02 so the model generalizes well to new data. The model is really accurate to the training data to the point it seems overfitted but it also generalizes well to unseen data, hence their is a negligebal difference in both accuracy and test scores.
- The model is correct when it approves 98% of the time and rejects 98% of the time . This is good precision that doesn't significantly negatively affect customer satisfication.
- Recall for approval is 99% so it correctly approves actuall good borrowers most of the time and correctly rejects 96% of the time.
- Our ROC-AUC of 0.974 means the model separated the Approved vs Rejected extremely well.

This model meets our expectations and is good for deployment. However, it seems to be a little bit overfitted due to the 100% accuracy so let's try to improve it a little.

Final Model

In this section we will to hypertune the decision tree to try and squeeze more perfomance from it. We will use gini as our criterion and use cross validation to get the best values for our parameters

```
# Preparing data

dt_cv = df.copy()
yc = dt_cv["loan_status"]
Xc = dt_cv.drop(columns=["loan_status"])
Xc_train, Xc_test,yc_train, yc_test = train_test_split(Xc,
yc,test_size=0.3, random_state=42)


# One hot encoding the categorical variables for the training data
ohec = OneHotEncoder()

ohec.fit(Xc_train[cat_cols])
Xc_train_ohe = ohec.transform(Xc_train[cat_cols]).toarray()
ohe_df = pd.DataFrame(Xc_train_ohe,
columns=ohec.get_feature_names_out(cat_cols), index=Xc_train.index)

Xc_final = pd.concat(
    [Xc_train[num_cols], ohe_df],
    axis = 1
)


# One hot encoding the  test categorical columns
Xc_test_ohe = ohec.transform(Xc_test[cat_cols]).toarray()
ohe_ttest = pd.DataFrame(
    Xc_test_ohe,
    columns = ohec.get_feature_names_out(cat_cols),
    index = Xc_test.index
)

Xc_test_final = pd.concat(
    [Xc_test[num_cols], ohe_ttest],
    axis = 1
)

# Define Decision Tree
cvt = DecisionTreeClassifier(random_state=42,criterion="gini")

# Define hyperparameter grid
# param_grid = {
```

```
#     'max_depth': [2, 3, 4, 5],
#     'min_samples_split': [2, 5, 10],
#     'min_samples_leaf': [1, 2, 4]
# }

# param_grid = {
#     'max_depth': [4, 5, 6, 7, 8],
#     'min_samples_split': [7, 9, 10, 11, 12, 15],
#     'min_samples_leaf': [1, 2, 4]
# }

param_grid = {
    'criterion' : ['gini', 'entropy'],
    'max_depth': [ 12,13,14, 15, 16, 17, 18, 25],
    'min_samples_split': [18, 19, 20, 21, 22,25, 30],
    'min_samples_leaf': [6, 7, 8, 9, 15]
}
# Grid search with cross-validation
grid_search = GridSearchCV(cvt, param_grid, cv=5)
grid_search.fit(Xc_final, yc_train)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=42),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [12, 13, 14, 15, 16, 17, 18,
25],
                         'min_samples_leaf': [6, 7, 8, 9, 15],
                         'min_samples_split': [18, 19, 20, 21, 22, 25,
30]})
```

```
yc_predict = grid_search.predict(Xc_test_final)
```

```
model_evaluate_man(grid_search, X_train=Xc_final,
X_test=Xc_test_final, y_test = yc_test, y_pred=yc_predict)
```

```
Model Performance Metrics:
R² Score: 0.9093
Root Mean Squared Error: 0.15
Mean Absolute Error: 0.02
Accuracy: 0.9789

========== Accuracy Scores ==========
Train accuracy: 0.9906291834002677
Test accuracy : 0.9789227166276346

========== Error Scores ==========
Train MAE: 0.009370816599732263
Test MAE : 0.02107728337236534

============================== Report ============================
              precision    recall  f1-score   support
```

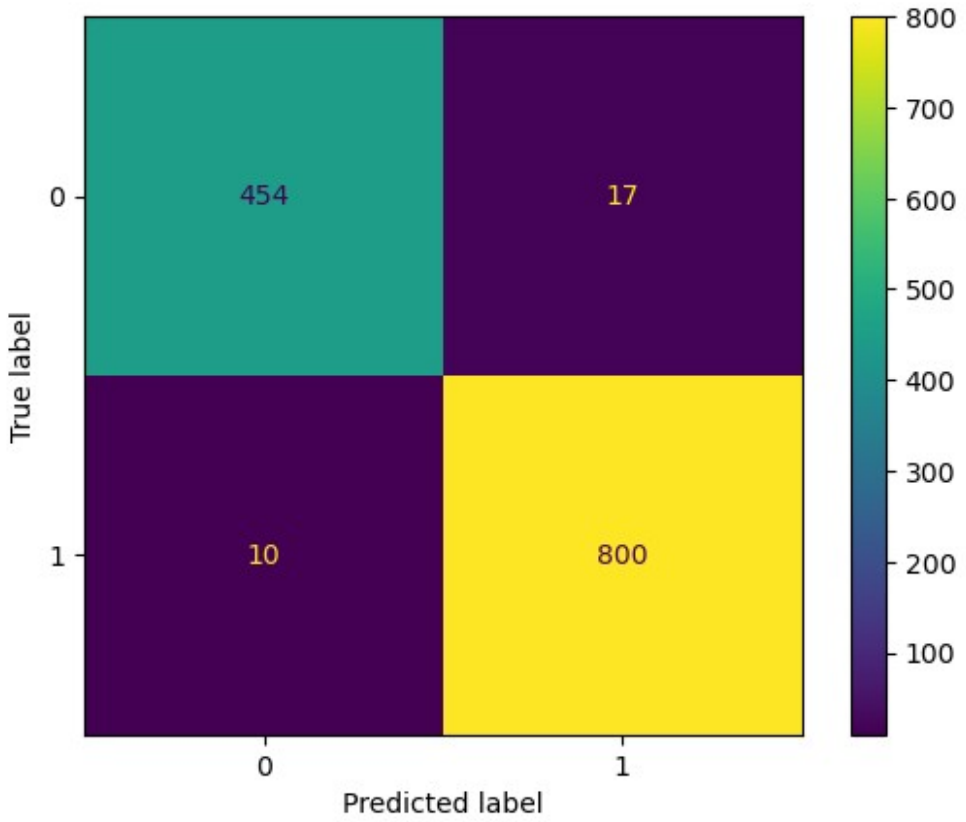
```
       False       0.98      0.96      0.97       471
        True       0.98      0.99      0.98       810

    accuracy                           0.98      1281
   macro avg       0.98      0.98      0.98      1281
weighted avg       0.98      0.98      0.98      1281

ROC-AUC: 0.9879282325496056
```

The model is more or less the same

```
cmv = confusion_matrix(yc_test, yc_predict)
print(cmv)

ConfusionMatrixDisplay(confusion_matrix=cmv).plot()
```

```
[[454  17]
 [ 10 800]]
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x23fac60a990>
```



Model Results

- The model has 800 TP, 454 TN, 17 FP and 10 FN.
- The model's test and training MAE's have a difference of 0.02 so the model generalizes well to new data. The model accuracys are also almost similar. The model is well fitted.
- The model is correct when it approves 99% of the time and reject 96% of the time . This is good precision that doesn't significantly negatively affect customer satisfication.
- Recall for approval is 99% so it approves good borrowers most of the time and rejection is 96% so it rejects risky loans most of the time so it reduces missing good opprtunities.
- Our ROC-AUC of 0.974 means the model separated the Approved vs Rejected extremely well.

This model is excellent for our goals. It's accuracy may be lower than the previous model but it actually perfoms slightly better in other metrics

6. Conclusion

- The third model performs slightly better than the second one and both greatly outperforms the base model according to the metrics.
- We will deploy the third model since its the best perfoming model.
- The model will do well to maximize revenue and customer satisfication due to the reason above.