

Problem Statement

The problem statement is to predict the likelihood of a loan being approved based on various factors such as applicant income, credit history, loan amount, and other demographic details.

Hypothesis

We hypothesize that the likelihood of being eligible for a loan is influenced by a combination of applicant income, credit history, loan amount, and demographic details. As higher income can indicate financial stability, we also hypothesize that higher income will have a positive correlation with loan approval probability.

Business goals

The goal is to develop a machine learning model that can accurately classify loan applications as either approved or rejected, thereby helping lenders make informed decisions.

Business Understanding: The business understanding of this problem is that lenders are looking for a way to reduce the risk of loan defaults by identifying applicants who are more likely to repay their loans. By analyzing various factors such as income, credit history, and loan amount, lenders can make more informed decisions about which applicants to approve and which to reject.

The business goals of this project are:

1. Risk Reduction: Reduce the risk of loan defaults by identifying applicants who are more likely to repay their loans.
2. Improved Decision Making: Develop a machine learning model that can accurately classify loan applications as either approved or rejected, thereby helping lenders make informed decisions.
3. Increased Efficiency: Automate the loan approval process by using a machine learning model, reducing the need for manual review and increasing the speed of the approval process.

The key performance indicators (KPIs) for this project will be:

1. Accuracy: The accuracy of the machine learning model in classifying loan applications as either approved or rejected.

2. Precision: The precision of the machine learning model in identifying applicants who are more likely to repay their loans.
3. Recall: The recall of the machine learning model in identifying applicants who are more likely to default on their loans.

By achieving these goals and KPIs, the project aims to improve the efficiency and effectiveness of the loan approval process, while also reducing the risk of loan defaults and increasing the overall profitability of the lending institution.

Getting the system ready and loading the data

Import libraries

```
In [ ]: # Suppress warnings
import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)

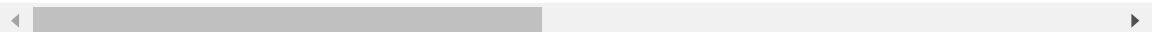
import pandas as pd
import sklearn as sk #for data profiling
```

Collect data

```
In [ ]: df = pd.read_csv('./data-for-project-1/raw_data.csv') #read the raw data from the
df.head() #show all features with first few rows of data
```

```
Out[ ]:
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000
3	LP001006	Male	Yes	0	Not Graduate	No	2583
4	LP001008	Male	No	0	Graduate	No	6000



We can see that the raw data consists of 12 features as well as the target attribute which is the loan status of each row entry. We notice that the data has a target attribute, Loan_Status, which informs us that we must apply classification machine learning models to predict outcomes. Therefore we can now plan and prepare our data accordingly.

Preprocessing and cleaning the data

Removing irrelevant features

The Loan_ID feature will be irrelevant to our models, thus we can remove it.

```
In [ ]: irrelevant_features = 'Loan_ID' #feature to be removed
#removing of feature
df.drop(
    columns=irrelevant_features,
    inplace=True
)
#display data without Loan_ID
df.head()
```

```
Out [ ]:
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
0	Male	No	0	Graduate	No	5849	
1	Male	Yes	1	Graduate	No	4583	
2	Male	Yes	0	Graduate	Yes	3000	
3	Male	Yes	0	Not Graduate	No	2583	
4	Male	No	0	Graduate	No	6000	

Change the target, Loan_Status, to int values

We want to convert the target data type from a string to an int value (0 and 1) to make data preparation and exploration easier.

```
In [ ]: df['Loan_Status'].replace({'Y': 1, 'N': 0}, inplace = True)
df.head()
```

```
Out [ ]:
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
0	Male	No	0	Graduate	No	5849	
1	Male	Yes	1	Graduate	No	4583	
2	Male	Yes	0	Graduate	Yes	3000	
3	Male	Yes	0	Not Graduate	No	2583	
4	Male	No	0	Graduate	No	6000	

Checking the cardinality of the features

```
In [ ]: # checking the cardinality of features
feature_cardinality = df.select_dtypes("object").nunique()
feature_cardinality
```

```
Out[ ]: Gender      2
Married    2
Dependents 4
Education  2
Self_Employed 2
Property_Area 3
dtype: int64
```

There is no need to handle the cardinality of the features as no features have very low cardinality or very high cardinality.

Understanding and profiling the data

Use the skimpy library to profile the data

```
In [ ]: sk.skim(df) #profile the dataframe of raw data
```

skimpy summary

Data Summary		Data Types	
dataframe	Values	Column Type	Count
Number of rows	614	string	6
Number of columns	12	float64	4
		int32	2

number

column_name	NA	NA %	mean	sd	p0	p25
ApplicantIncome	0	0	5400	6100	150	290
CoapplicantIncome	0	0	1600	2900	0	
LoanAmount	22	3.58	150	86	9	10
Loan_Amount_Term	14	2.28	340	65	12	36
Credit_History	50	8.14	0.84	0.36	0	
Loan_Status	0	0	0.69	0.46	0	

string

column_name	NA	NA %	words per row
Gender	13	2.12	
Married	3	0.49	
Dependents	15	2.44	
Education	0	0	
Self_Employed	32	5.21	
Property_Area	0	0	

End

According to the skimpy summary above, we notice that there are 3 numeric features and 9 categorical features (which we will be encoding for better data understanding). The features LoanAmount, Loan-Amount_Term, Credit_History, Gender, Married, Dependents,

and Self_Employed have missing values. Credit_History must also be changed to a string feature as it is categorical.

Handling missing values

```
In [ ]: df['LoanAmount'].fillna(  
        df['LoanAmount']  
        .dropna()  
        .mean(),  
        inplace=True  
    )  
  
col_cat = ['Gender', 'Married', 'Dependents', 'Self_Employed', 'Credit_History', 'Loa  
for col in col_cat:  
    df[col].fillna(  
        df[col]  
        .mode()[0],  
        inplace=True  
    )  
  
sk.skim(df)
```

skimpy summary

Data Summary

dataframe	Values
Number of rows	614
Number of columns	12

Data Types

Column Type	Count
string	6
float64	4
int32	2

number

column_name	NA	NA %	mean	sd	p0	p25
ApplicantIncome	0	0	5400	6100	150	290
CoapplicantIncome	0	0	1600	2900	0	
LoanAmount	0	0	150	84	9	10
Loan_Amount_Term	0	0	340	64	12	36
Credit_History	0	0	0.86	0.35	0	
Loan_Status	0	0	0.69	0.46	0	

string

column_name	NA	NA %	words per row
Gender	0	0	
Married	0	0	
Dependents	0	0	
Education	0	0	
Self_Employed	0	0	
Property_Area	0	0	

End

The missing values have been handled as we notice in the skimpy summary there are no NA values in any of the features.

Encoding categorical variables

We encode the categorical features to showcase the possible values of each feature as well as what categorical value each row entry has as an encoded variable.

```
In [ ]: from category_encoders import OneHotEncoder

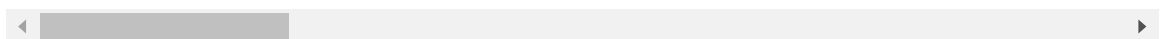
ohe = OneHotEncoder(
    use_cat_names=True,
    cols=['Gender', 'Married', 'Dependents', 'Self_Employed', 'Education', 'Property_
')

encoded_df = ohe.fit_transform(df)
encoded_df.head()
```

```
Out[ ]:
```

	Gender_Male	Gender_Female	Married_No	Married_Yes	Dependents_0	Dependents_
0	1	0	1	0	1	
1	1	0	0	1	0	
2	1	0	0	1	1	
3	1	0	0	1	1	
4	1	0	1	0	1	

5 rows × 31 columns



```
In [ ]: encoded_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 31 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Gender_Male                          614 non-null    int64
1   Gender_Female                        614 non-null    int64
2   Married_No                           614 non-null    int64
3   Married_Yes                          614 non-null    int64
4   Dependents_0                         614 non-null    int64
5   Dependents_1                         614 non-null    int64
6   Dependents_2                         614 non-null    int64
7   Dependents_3+                       614 non-null    int64
8   Education_Graduate                   614 non-null    int64
9   Education_Not Graduate                614 non-null    int64
10  Self_Employed_No                     614 non-null    int64
11  Self_Employed_Yes                    614 non-null    int64
12  ApplicantIncome                       614 non-null    int64
13  CoapplicantIncome                     614 non-null    float64
14  LoanAmount                           614 non-null    float64
15  Loan_Amount_Term_360.0                614 non-null    int64
16  Loan_Amount_Term_120.0                614 non-null    int64
17  Loan_Amount_Term_240.0                614 non-null    int64
18  Loan_Amount_Term_180.0                614 non-null    int64
19  Loan_Amount_Term_60.0                 614 non-null    int64
20  Loan_Amount_Term_300.0                614 non-null    int64
21  Loan_Amount_Term_480.0                614 non-null    int64
22  Loan_Amount_Term_36.0                 614 non-null    int64
23  Loan_Amount_Term_84.0                 614 non-null    int64
24  Loan_Amount_Term_12.0                 614 non-null    int64
25  Credit_History_1.0                    614 non-null    int64
26  Credit_History_0.0                    614 non-null    int64
27  Property_Area_Urban                   614 non-null    int64
28  Property_Area_Rural                   614 non-null    int64
29  Property_Area_Semiurban               614 non-null    int64
30  Loan_Status                           614 non-null    int64
dtypes: float64(2), int64(29)
memory usage: 148.8 KB
```

We can clearly see now that encoding the categorical features will return numeric values. We will make use of encoding when building our models to ensure all features fit the possible models we will be implementing.

Creating a prepare data function

We will combine our data preparation code into a function that we can call to prepare raw data for modelling. The function will return a dataframe of prepared data.

```
In [ ]: def prepare_data(path):
        prep_df = pd.read_csv(path) #read the raw data from the raw_data.csv file

        irrelevant_features = 'Loan_ID' #feature to be removed
        #removing of feature
        prep_df.drop(
            columns=irrelevant_features,
            inplace=True
        )
```

```

if 'Loan_Status' in prep_df.columns:
    prep_df['Loan_Status'].replace({'Y': 1, 'N': 0},inplace = True)#replace

#Handling of missing data
prep_df['LoanAmount'].fillna(
    prep_df['LoanAmount']
    .dropna()
    .mean(),
    inplace=True
)

col_cat = ['Gender','Married','Dependents','Self_Employed','Credit_History',
for col in col_cat:
    prep_df[col].fillna(
        prep_df[col]
        .mode()[0],
        inplace=True
    )

#clean column names
from skimpy import clean_columns
return clean_columns(prep_df)

```

Calling the prepare_data function and writing to a file

```

In [ ]: prepared_df = prepare_data('./data-for-project-1/raw_data.csv')
        prepared_df.to_csv('./data-for-project-1/prepared_data.csv')

```

Data exploration

```

In [ ]: # For Visualization
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sn

```

High Collinearity

```

In [ ]: corr_df = df.select_dtypes("number").corr() #checking the correlation between th
corr_df

```

```

Out[ ]:

```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_T
ApplicantIncome	1.000000	-0.116605	0.565620	-0.046531
CoapplicantIncome	-0.116605	1.000000	0.187828	-0.059383
LoanAmount	0.565620	0.187828	1.000000	0.036475
Loan_Amount_Term	-0.046531	-0.059383	0.036475	1.000000
Credit_History	-0.018615	0.011134	-0.001431	-0.004710
Loan_Status	-0.004710	-0.059187	-0.036416	-0.022000


```
In [ ]: fig = px.imshow(corr_df, color_continuous_scale='Spectral')
fig.update_layout(title='Heat Map: Correlation of Features', font=dict(size=12))
fig.show()
```

From the correlation matrix displayed, we can see that there is high multi-collinearity on Credit_History against the Loan_Status. As Loan_Status is our target, we could look to drop the Credit_History feature. We also notice high multi-collinearity on LoanAmount against ApplicantIncome. Since income is more important for loan eligibility, we could look to drop the LoanAmount feature.

Univariate analysis

```
In [ ]: # Prepare data to display
labels = (
    df['Loan_Status']
    .astype('str')
    .str.replace('0', 'No', regex=True)
    .str.replace('1', 'Yes', regex=True)
    .value_counts()
)

# Create figure using Plotly
fig = px.bar(
    data_frame=labels,
    x=labels.index,
    y=labels.values,
    title=f'Class Imbalance',
    color=labels.index
)

# Add titles & Display figure
fig.update_layout(xaxis_title='Loan Status', yaxis_title='Number of Customers')
fig.show()
```

The BC Finance company would like to decrease the number of customers who are not eligible for a loan, which is depicted by the orange bar labeled N in the graph above.

Bivariate/Multivariate analysis

We will now compare each feature against the target to gain insights on how features influence the target positively as well as negatively.

Analysis of numeric features

```
In [ ]: #Gaining insights on which features to use as numeric and categorical
df.select_dtypes('number').nunique()
```

```
Out[ ]: ApplicantIncome      505
        CoapplicantIncome    287
        LoanAmount           204
        Loan_Amount_Term      10
        Credit_History         2
        Loan_Status           2
        dtype: int64
```

```
In [ ]: # Select features to plot
        plot_cols = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']

        # Plot numeric features against target
        plt.figure(figsize=(3,4))
        for col in plot_cols:
            fig = px.box(data_frame=df[plot_cols], x=col, color=df['Loan_Status'], title=
            fig.update_layout(xaxis_title=f'{col} Feature')
            fig.show()
```

For all three numeric features we notice that they have a similar impact on both the positive and negative outcome of the target. There is however a considerable amount of outliers that we will be treating.

Visualization of numeric features without outliers

We will be looking at visualizations of the numeric features again but this time without the outliers.

```
In [ ]: mask_appincome = df['ApplicantIncome'] < 6000 #mask for filtering the ApplicantIncome
        df_mask1 = df[mask_appincome] #filtered dataframe from the applicant income mask
        mask_coincome = df['CoapplicantIncome'] < 2300 #mask for filtering the CoapplicantIncome
        df_mask2 = df[mask_coincome] #filtered dataframe from the coapplicant income mask
        mask_loanamount_upper = df['LoanAmount'] < 175 #mask for filtering the LoanAmount
        mask_loanamount_lower = df['LoanAmount'] > 35
        df_mask3 = df[mask_loanamount_upper & mask_loanamount_lower] #filtered dataframe

        plot_cols = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']

        plt.figure(figsize=(3,4))
        for col in plot_cols:
            if col == 'ApplicantIncome':
                fig = px.box(data_frame=df_mask1, x=col, color= df_mask1['Loan_Status'],
                title=f'BoxPlot for {col} Feature against the Target without outliers')
                fig.update_layout(xaxis_title=f'{col} Feature')
                fig.show()
            elif col == 'CoapplicantIncome':
                fig = px.box(data_frame=df_mask2, x=col, color= df_mask2['Loan_Status'],
                title=f'BoxPlot for {col} Feature against the Target without outliers')
                fig.update_layout(xaxis_title=f'{col} Feature')
                fig.show()
            else:
                fig = px.box(data_frame=df_mask3, x=col, color= df_mask3['Loan_Status'],
                title=f'BoxPlot for {col} Feature against the Target without outliers')
                fig.update_layout(xaxis_title=f'{col} Feature')
                fig.show()
```

By removing the outliers we now see that there is a slight difference in the effect of the features on the loan status. The most obvious one lies with the loan amount where higher loan amounts tend to result in more negative loan status outcomes.

Analysis of categorical features

We will now compare each categorical feature against the loan status.

```
In [ ]: cat_col = ['Loan_Amount_Term', 'Credit_History', 'Gender', 'Married', 'Dependents', '']
for col in cat_col:
    # Aggregate Category Feature
    new_df = pd.DataFrame(
        df[[col, 'Loan_Status']]
        .groupby(['Loan_Status'])
        .value_counts()
        .reset_index()
    )

    # Plot Category feature vs Label
    fig = px.bar(
        data_frame=new_df,
        x=col,
        y='count',
        facet_col='Loan_Status',
        color=new_df['Loan_Status'].astype(str), # convert it to string to avoid
        title=f'{col} vs Target'
    )

    fig.update_layout(xaxis_title=col, yaxis_title='Number of Customers')
    fig.show()
```

When looking at these visuals, we will focus on the effect of the features on the negative loan outcome as this is what is significant for the company. The loan amount term visual indicates an increase in denied loan as the term increases. Credit history shows more negative outcomes when credit history is evident. Gender shows that more males are denied loans than females. More married individuals are denied loans than single individuals. Having less dependents indicates an increased chance of being denied a loan. More customers who have graduated are denied loans than ungraduated customers. Customers who are not self employed are more likely to not get a loan. Customers from rural and urban areas are more likely to not receive a loan.

Evaluation metrics for this classification problem

Libraries to be used during classification evaluations:

```
In [ ]: import numpy as np
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```

from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from category_encoders import OneHotEncoder

# Metric variables to be used for output
accuracy_scores = []
precisions = []
f1_scores = []
recalls = []

```

Datasets to be used for the Logistic Regression and Decision Tree algorithms:

```

In [ ]: # Split loan status data
label = 'loan_status'
x = prepared_df.drop(columns=[label], inplace=False)
y = prepared_df[label]

x_Train, x_Val, y_Train, y_Val = train_test_split(x, y, test_size=0.4, random_st

print(
    f'Training dataset \
    \nx_Train: {x_Train.shape[0]/len(x)*100:.0f}% \ny_Train: {y_Train.shape[0]/1
    \n\nValidation dataset \
    \nx_Val: {x_Val.shape[0]/len(x)*100:.0f}% \ny_Val: {y_Val.shape[0]/len(x)*10
)

```

Training dataset

x_Train: 60%

y_Train: 60%

Validation dataset

x_Val: 40%

y_Val: 40%

Baseline accuracy

```

In [ ]: accuracy_Base = y_Train.value_counts(normalize=True).max()

print("Baseline Accuracy:", round(accuracy_Base, 2))

```

Baseline Accuracy: 0.71

Logistic Regression evaluation

```

In [ ]: # Model building and fitting
regression_Model = make_pipeline(
    OneHotEncoder(use_cat_names=True),
    LogisticRegression(max_iter=2500)
)
regression_Model.fit(x_Train, y_Train)

# Display accuracy scores
lr_train_acc = regression_Model.score(x_Train, y_Train)
lr_val_acc = regression_Model.score(x_Val, y_Val)
print("Logistic Regression training accuracy:", lr_train_acc)
print("Logistic Regression validation accuracy:", lr_val_acc)

```

```

y_val_pred_reg = regression_Model.predict(x_Val)

accuracy_scores.append(round(accuracy_score(y_Val, y_val_pred_reg),4)),
precisions.append(round(precision_score(y_Val, y_val_pred_reg),4)),
recalls.append(round(recall_score(y_Val, y_val_pred_reg),4)),
f1_scores.append(round(f1_score(y_Val, y_val_pred_reg),4))

```

Logistic Regression training accuracy: 0.8179347826086957

Logistic Regression validation accuracy: 0.7764227642276422

Decision Tree evaluation

```

In [ ]: depth_hyperpar = range(1, 8)

# List of scores per each set for visualization purpose
training_Scores = []
validation_Scores = []

for d in depth_hyperpar:
    # Model building and fitting
    tree_Model = make_pipeline(
        OneHotEncoder(use_cat_names=True),
        DecisionTreeClassifier(max_depth=d, random_state=42)
    )
    tree_Model.fit(x_Train, y_Train)

    # Calculate training accuracy score and append to `training_scores`
    training_Scores.append(tree_Model.score(x_Train, y_Train))

    # Calculate validation accuracy score and append to `validation_scores`
    validation_Scores.append(tree_Model.score(x_Val, y_Val))

tune_data = pd.DataFrame(
    data = {'Training': training_Scores, 'Validation': validation_Scores},
    index=depth_hyperpar
)

fig = px.line(
    data_frame=tune_data,
    x=depth_hyperpar,
    y=['Training', 'Validation'],
    title="Decision Tree Model training & validation curves"
)
fig.update_layout(xaxis_title="Maximum Depth", yaxis_title="Accuracy Score")
fig.show()

y_val_pred_tree = tree_Model.predict(x_Val)

accuracy_scores.append(round(accuracy_score(y_Val, y_val_pred_tree),4)),
precisions.append(round(precision_score(y_Val, y_val_pred_tree),4)),
recalls.append(round(recall_score(y_Val, y_val_pred_tree),4)),
f1_scores.append(round(f1_score(y_Val, y_val_pred_tree),4))

```

```

In [ ]: # Compare evaluation metrics for Logistic regression and Decision tree
metrics_1 = {
    'Accuracy': accuracy_scores,
    'Precision': precisions,
    'F1-Score': f1_scores,

```

```

        'Recall': recalls
    }

pd.DataFrame(
    data=metrics_1,
    index=['Logistic Regression', 'Decision Tree']
).sort_values(
    by='Accuracy',
    ascending=False
)

```

```
Out [ ]:
```

	Accuracy	Precision	F1-Score	Recall
Logistic Regression	0.7764	0.7524	0.8518	0.9814
Decision Tree	0.7195	0.7396	0.8045	0.8820

```

In [ ]: import joblib

# Saving first iteration as Logistic Regression model
joblib.dump(regression_Model, './artifacts/model_1.pkl')

```

```
Out [ ]: ['./artifacts/model_1.pkl']
```

Here we find that the Logistic Regression could be a better model to make use of, and it is saved as the first iteration.

However, the Deep Learning algorithm model is yet to be evaluated.

The building and evaluation of the Deep Learning algorithm model is done under Model Building Part 1 below, without feature engineering.

Model building part 1

```

In [ ]: # Columns to be encoded for deep Learning algorithm
categorical_columns = ['gender', 'married', 'dependents', 'education', 'self_employe

ohe = OneHotEncoder(cols=categorical_columns)

# Split Loan status encoded data for deep Learning
X_encoded = ohe.fit_transform(prepared_df)
y_deep = prepared_df[label]

X_train_deep, X_val_deep, y_train_deep, y_val_deep = train_test_split(X_encoded,

# Scaling features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_deep)
X_val_scaled = scaler.transform(X_val_deep)

# Building, compiling and training the deep Learning model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.5),

```

```

        Dense(1, activation='sigmoid')
    ])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.fit(X_train_scaled, y_train_deep, epochs=20)

# Evaluating deep Learning
y_pred_deep_probability = model.predict(X_val_scaled)
y_pred_deep = (y_pred_deep_probability > 0.5).astype(int)

accuracy_scores.append(round(accuracy_score(y_val_deep, y_pred_deep),4)),
precisions.append(round(precision_score(y_val_deep, y_pred_deep),4)),
recalls.append(round(recall_score(y_val_deep, y_pred_deep),4)),
f1_scores.append(round(f1_score(y_val_deep, y_pred_deep),4))

```

```

Epoch 1/20
12/12 ————— 1s 2ms/step - accuracy: 0.5126 - loss: 0.7704
Epoch 2/20
12/12 ————— 0s 1ms/step - accuracy: 0.6563 - loss: 0.6161
Epoch 3/20
12/12 ————— 0s 1ms/step - accuracy: 0.7068 - loss: 0.5740
Epoch 4/20
12/12 ————— 0s 2ms/step - accuracy: 0.7773 - loss: 0.5312
Epoch 5/20
12/12 ————— 0s 1ms/step - accuracy: 0.7280 - loss: 0.5426
Epoch 6/20
12/12 ————— 0s 1ms/step - accuracy: 0.7597 - loss: 0.5115
Epoch 7/20
12/12 ————— 0s 2ms/step - accuracy: 0.8086 - loss: 0.4317
Epoch 8/20
12/12 ————— 0s 1ms/step - accuracy: 0.8021 - loss: 0.4062
Epoch 9/20
12/12 ————— 0s 2ms/step - accuracy: 0.8681 - loss: 0.3948
Epoch 10/20
12/12 ————— 0s 1ms/step - accuracy: 0.8824 - loss: 0.3182
Epoch 11/20
12/12 ————— 0s 2ms/step - accuracy: 0.8915 - loss: 0.3069
Epoch 12/20
12/12 ————— 0s 1ms/step - accuracy: 0.8741 - loss: 0.3228
Epoch 13/20
12/12 ————— 0s 1ms/step - accuracy: 0.8952 - loss: 0.2787
Epoch 14/20
12/12 ————— 0s 2ms/step - accuracy: 0.9226 - loss: 0.2207
Epoch 15/20
12/12 ————— 0s 1ms/step - accuracy: 0.8837 - loss: 0.2857
Epoch 16/20
12/12 ————— 0s 1ms/step - accuracy: 0.9429 - loss: 0.2147
Epoch 17/20
12/12 ————— 0s 1ms/step - accuracy: 0.9452 - loss: 0.1890
Epoch 18/20
12/12 ————— 0s 1ms/step - accuracy: 0.9077 - loss: 0.2239
Epoch 19/20
12/12 ————— 0s 1ms/step - accuracy: 0.9630 - loss: 0.1395
Epoch 20/20
12/12 ————— 0s 1ms/step - accuracy: 0.9425 - loss: 0.1447
8/8 ————— 0s 5ms/step

```

```

In [ ]: # Comparing Deep Learning to previous algorithms
        metrics_2 = {

```

```

        'Accuracy': accuracy_scores,
        'Precision': precisions,
        'F1-Score': f1_scores,
        'Recall': recalls
    }

pd.DataFrame(
    data=metrics_2,
    index=['Logistic Regression', 'Decision Tree', 'Deep Learning 1']
).sort_values(
    by='Accuracy',
    ascending=False
)

```

```
Out[ ]:
```

	Accuracy	Precision	F1-Score	Recall
Deep Learning 1	0.9837	0.9816	0.9877	0.9938
Logistic Regression	0.7764	0.7524	0.8518	0.9814
Decision Tree	0.7195	0.7396	0.8045	0.8820

We find that the Deep Learning model is significantly more useful than the previous classification models

As such, next we save the Deep Learning model without the feature engineering as our second iteration.

```
In [ ]: # Saving second iteration of Deep Learning model
joblib.dump(model, './artifacts/model_2.pkl')
```

```
Out[ ]: ['./artifacts/model_2.pkl']
```

Feature engineering

```
In [ ]: # Total Income Feature
prepared_df['total_income'] = prepared_df['applicant_income'] + prepared_df['coapplicant_income']
prepared_df.drop(columns=['loan_amount', 'credit_history'], inplace=True)
prepared_df
```


Out[]:

	gender	married	dependents	education	self_employed	applicant_income	coapp
0	Male	No	0	Graduate	No	5849	
1	Male	Yes	1	Graduate	No	4583	
2	Male	Yes	0	Graduate	Yes	3000	
3	Male	Yes	0	Not Graduate	No	2583	
4	Male	No	0	Graduate	No	6000	
...
609	Female	No	0	Graduate	No	2900	
610	Male	Yes	3+	Graduate	No	4106	
611	Male	Yes	1	Graduate	No	8072	
612	Male	Yes	2	Graduate	No	7583	
613	Female	No	0	Graduate	Yes	4583	

614 rows × 11 columns



Model building part 2

```
In [ ]: # Columns to be encoded for featured deep Learning algorithm
categorical_columns = ['gender', 'married', 'dependents', 'education', 'self_employed']

ohe_2 = OneHotEncoder(cols=categorical_columns)

# Split loan status encoded data for featured deep Learning
X_encoded_2 = ohe_2.fit_transform(prepared_df)
y_deep_2 = prepared_df[label]

X_train_deep_2, X_val_deep_2, y_train_deep_2, y_val_deep_2 = train_test_split(X_encoded_2, y_deep_2,
                                                                              test_size=0.2,
                                                                              random_state=42)

# Scaling features
scaler_2 = StandardScaler()
X_train_scaled_2 = scaler_2.fit_transform(X_train_deep_2)
X_val_scaled_2 = scaler_2.transform(X_val_deep_2)















# Building, compiling and training the featured deep Learning model
model_2 = Sequential([
    Dense(64, activation='relu', input_shape=(X_train_scaled_2.shape[1],)),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model_2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model_2.fit(X_train_scaled_2, y_train_deep_2, epochs=20)
```

```
# Evaluating featured deep Learning
y_pred_deep_probability_2 = model_2.predict(X_val_scaled_2)
y_pred_deep_2 = (y_pred_deep_probability_2 > 0.5).astype(int)

accuracy_scores.append(round(accuracy_score(y_val_deep_2, y_pred_deep_2),4)),
precisions.append(round(precision_score(y_val_deep_2, y_pred_deep_2),4)),
recalls.append(round(recall_score(y_val_deep_2, y_pred_deep_2),4)),
f1_scores.append(round(f1_score(y_val_deep_2, y_pred_deep_2),4))
```

```
Epoch 1/20
12/12  1s 2ms/step - accuracy: 0.6646 - loss: 0.6498
Epoch 2/20
12/12  0s 2ms/step - accuracy: 0.6994 - loss: 0.5763
Epoch 3/20
12/12  0s 2ms/step - accuracy: 0.7528 - loss: 0.5743
Epoch 4/20
12/12  0s 2ms/step - accuracy: 0.7330 - loss: 0.5728
Epoch 5/20
12/12  0s 2ms/step - accuracy: 0.7490 - loss: 0.4962
Epoch 6/20
12/12  0s 1ms/step - accuracy: 0.8198 - loss: 0.4676
Epoch 7/20
12/12  0s 2ms/step - accuracy: 0.8259 - loss: 0.4497
Epoch 8/20
12/12  0s 1ms/step - accuracy: 0.8153 - loss: 0.3927
Epoch 9/20
12/12  0s 1ms/step - accuracy: 0.8553 - loss: 0.4469
Epoch 10/20
12/12  0s 2ms/step - accuracy: 0.8503 - loss: 0.3923
Epoch 11/20
12/12  0s 1ms/step - accuracy: 0.8890 - loss: 0.3149
Epoch 12/20
12/12  0s 1ms/step - accuracy: 0.8461 - loss: 0.3220
Epoch 13/20
12/12  0s 1ms/step - accuracy: 0.9046 - loss: 0.2816
Epoch 14/20
12/12  0s 1ms/step - accuracy: 0.9093 - loss: 0.2514
Epoch 15/20
12/12  0s 1ms/step - accuracy: 0.9299 - loss: 0.1888
Epoch 16/20
12/12  0s 1ms/step - accuracy: 0.9461 - loss: 0.1917
Epoch 17/20
12/12  0s 2ms/step - accuracy: 0.9632 - loss: 0.1750
Epoch 18/20
12/12  0s 1ms/step - accuracy: 0.9440 - loss: 0.1824
Epoch 19/20
12/12  0s 1ms/step - accuracy: 0.9369 - loss: 0.1458
Epoch 20/20
12/12  0s 2ms/step - accuracy: 0.9723 - loss: 0.1262
8/8  0s 5ms/step
```

The new Deep Learning model with feature engineering is built and trained and performs as follows:

```
In [ ]: # Comparing final Deep Learning to previous algorithms
metrics_3 = {
    'Accuracy': accuracy_scores,
    'Precision': precisions,
    'F1-Score': f1_scores,
    'Recall': recalls
```

```

    }

pd.DataFrame(
    data=metrics_3,
    index=['Logistic Regression', 'Decision Tree', 'Deep Learning 1', 'Deep Learning 2'],
).sort_values(
    by='Accuracy',
    ascending=False
)

```

Out []:

	Accuracy	Precision	F1-Score	Recall
Deep Learning 2	0.9959	0.9938	0.9969	1.0000
Deep Learning 1	0.9837	0.9816	0.9877	0.9938
Logistic Regression	0.7764	0.7524	0.8518	0.9814
Decision Tree	0.7195	0.7396	0.8045	0.8820

	Accuracy	Precision	F1-Score	Recall
Deep Learning 2	0.9959	0.9938	0.9969	1.0000
Deep Learning 1	0.9837	0.9816	0.9877	0.9938
Logistic Regression	0.7764	0.7524	0.8518	0.9814
Decision Tree	0.7195	0.7396	0.8045	0.8820

We can notice that the model performs better with the placed feature engineering and save it as our final model.

```

In [ ]: # Saving Final iteration of Deep Learning model
joblib.dump(model_2, './artifacts/final_model.pkl')

```

Out []: ['./artifacts/final_model.pkl']

Feature Importance

We will now take a look at the feature importances of our final model.

```

In [ ]: import shap

# Explain the model's predictions using SHAP
explainer = shap.Explainer(model_2, X_train_scaled_2)
shap_values = explainer.shap_values(X_val_scaled_2)

feature_list = ['Gender_Male', 'Gender_Female', 'Married_No', 'Married_Yes', 'Depend

# Plot SHAP values
shap.summary_plot(shap_values, X_val_scaled_2, plot_type="bar", feature_names=fe

# Write to relevant csv file
shap_df = pd.DataFrame(data=shap_values, columns=feature_list)
shap_df.to_csv('./artifacts/feature_importance')

```

PermutationExplainer explainer: 247it [00:10, 1.03s/it]



Making predictions on validation data

```
In [ ]: test_model = joblib.load('./artifacts/final_model.pkl')

test_data = prepare_data('./data-for-project-1/validation.csv')

test_data['total_income'] = test_data['applicant_income'] + test_data['coapplicant_income']
test_data.drop(columns=['loan_amount', 'credit_history'], inplace=True)

ohe_2_test = OneHotEncoder(handle_unknown='ignore', cols=['gender', 'married', 'dependents'])
ohe_2_test.fit(test_data)
# Encode categorical columns
X_test_encoded = ohe_2_test.transform(test_data)
if X_test_encoded.shape[1] < X_train_scaled_2.shape[1]:
    # Add missing columns with default values (e.g., 0)
    missing_columns = set(X_train_deep_2.columns) - set(test_data.columns)
    for col in missing_columns:
```

```

X_test_encoded[col] = 0

X_test_encoded = X_test_encoded[X_train_deep_2.columns]
X_test_scaled = scaler_2.transform(X_test_encoded)

predictions = test_model.predict(X_test_scaled)
binary_predictions = (predictions >= 0.5).astype(int)
binary_predictions_labels = ['yes' if pred == 1 else 'no' for pred in binary_pre

output_df = pd.DataFrame({'Predicted_Loan_Status': binary_predictions_labels})
output_df.to_csv('./artifacts/predictions.csv', index=False)

```

The ChatGPT template was used for the web application development. Bugs were addressed and the code is present in the web_application notebook in our [GitHub repository](#)

[Group J GitHub repository](#)

The web_application python file in the [repository](#), together with the requirements text file, were used to deploy the DASH web application on a web service.