EECS 3401 – Introduction to AI and
Logic Programming
Instructor: Ruba Al Omari
Project

Loan Approval Prediction

Exploring and Predicting with a Dataset

Group Members:
1. Hasti Ghaneshirazi
2. Wenzhuo Li
3. Helia Ostadalipour

Abstract:

This program uses machine learning and data analysis to develop models and algorithms that predict the likelihood of loan approval based on the given features. Based on the features in the dataset, the prediction of a loan approval will be conducted.

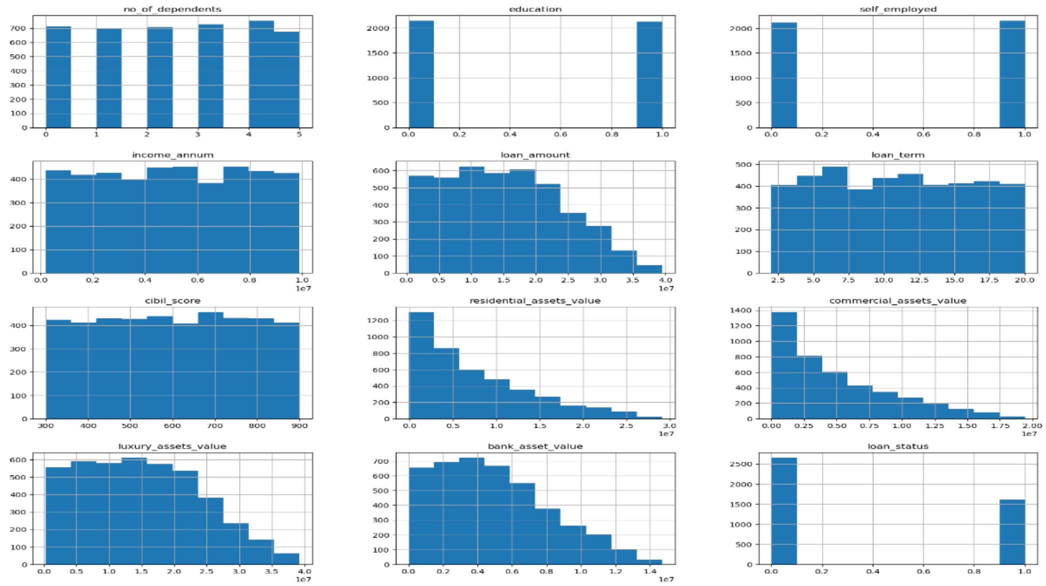
Task 1: Framing the problem and looking at the big picture.

- Supervised learning – We are using numerical values and the data has been labeled.
- Classification task – Predict the classes (labels) based on calculation and determining the approval of the loan (can be considered as 0 and 1)
- Batch learning
  - Small data set
  - No continuous flow of data coming into the system.
  - No need to adjust to changing data rapidly.

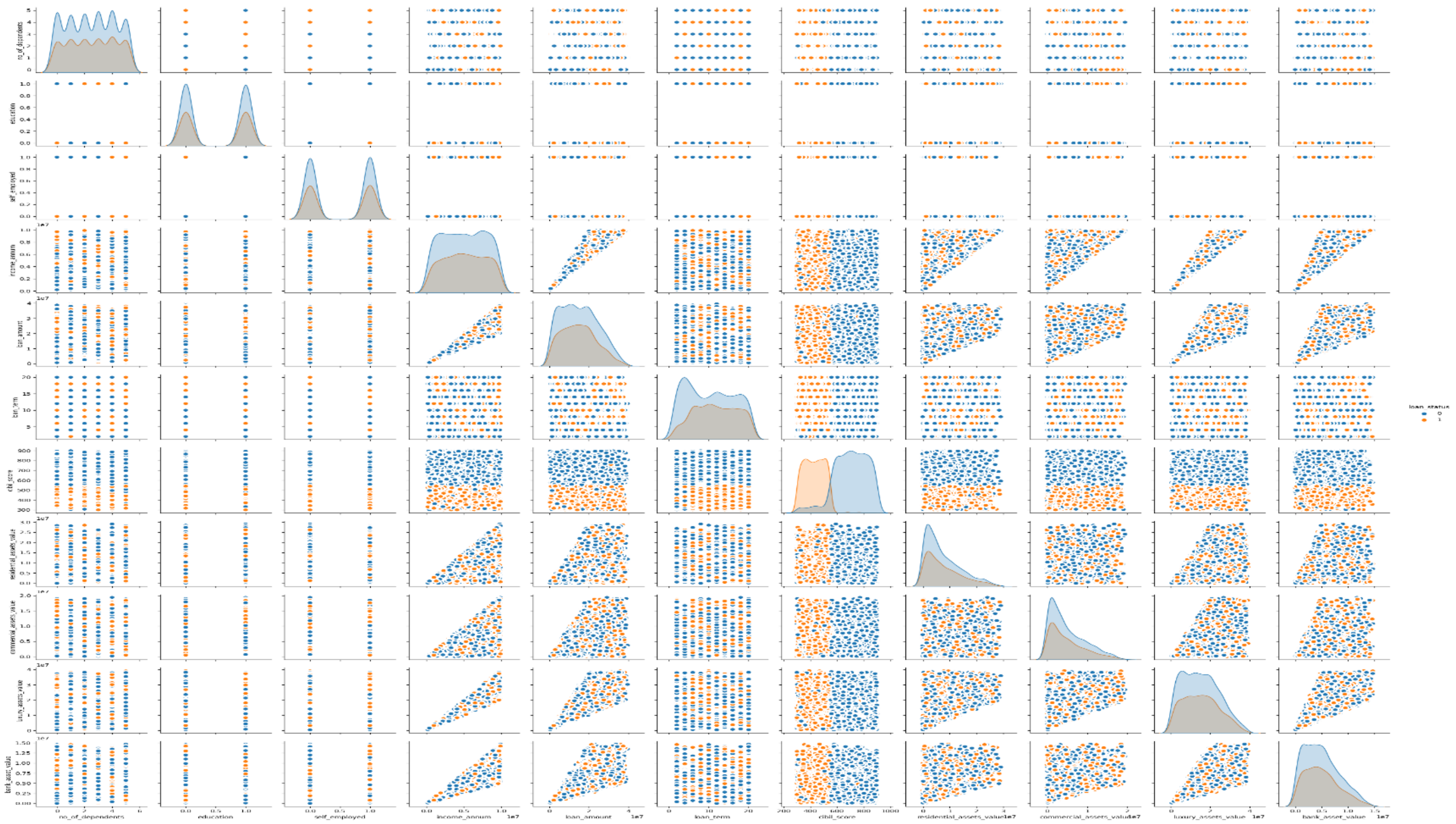
Task 2: A description of the dataset and 3 graphs of EDA.

In our EDAs (especially in the Correlation Matrix) we are showing that the features in our dataset don't have a strong correlation to each other. There are a few noticeable values. First is the relation between cibil_score and the loan_status (target) is the reverse relationship with a correlation of -0.77. There is no other significant correlation with the target value, however, we can observe that the amount of annual_income affects the loan_amount and the luxury_asset_value directly (logical relationship). This pattern is also visible in the other visuals. For example, in the pair plot, the most distinct separated values are the ones with the highest correlations.

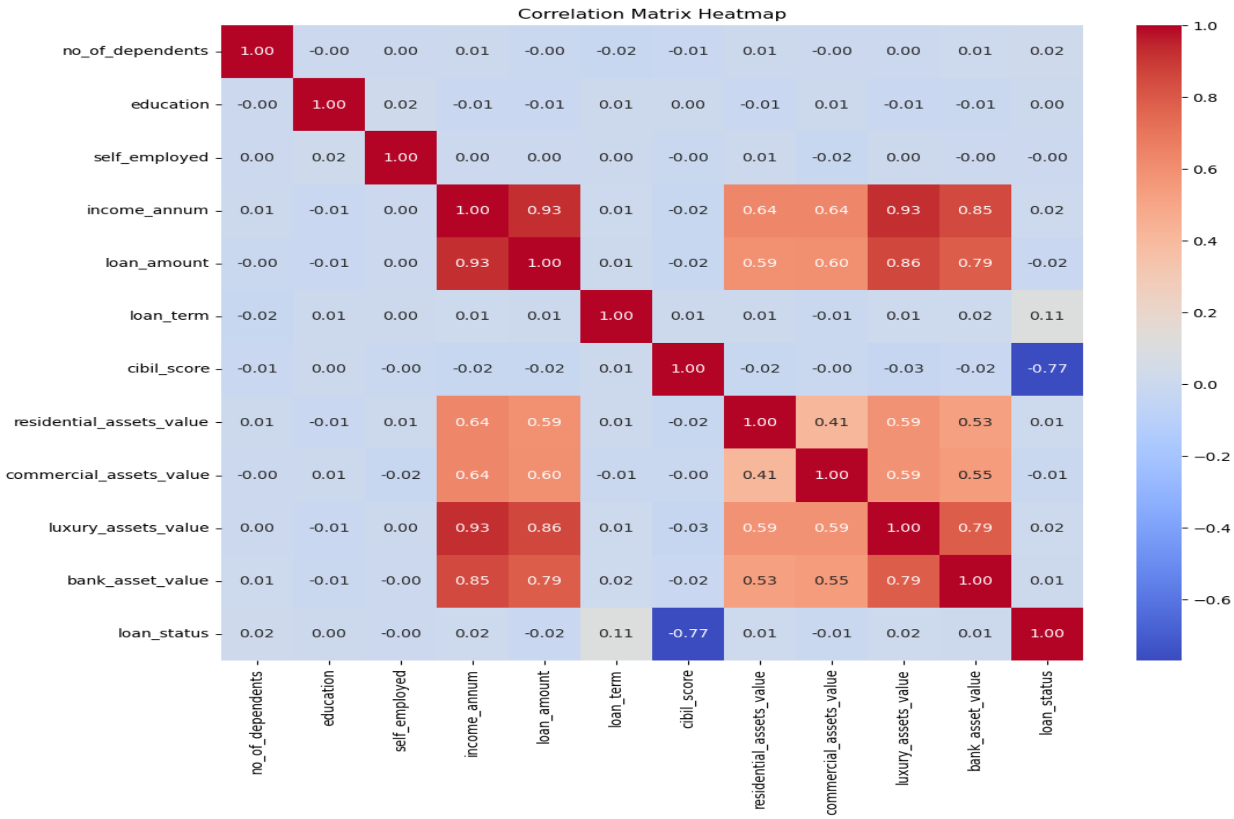
Histogram:



Pair Plot (scatter):



Correlation Matrix:



	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury_assets_value	bank_asset_value	loan_status
no_of_dependents	1.00	-0.00	0.00	0.01	-0.00	-0.02	-0.01	0.01	-0.00	0.00	0.01	0.02
education	-0.00	1.00	0.02	-0.01	-0.01	0.01	0.00	-0.01	0.01	-0.01	-0.01	0.00
self_employed	0.00	0.02	1.00	0.00	0.00	0.00	-0.00	0.01	-0.02	0.00	-0.00	-0.00
income_annum	0.01	-0.01	0.00	1.00	0.93	0.01	-0.02	0.64	0.64	0.93	0.85	0.02
loan_amount	-0.00	-0.01	0.00	0.93	1.00	0.01	-0.02	0.59	0.60	0.86	0.79	-0.02
loan_term	-0.02	0.01	0.00	0.01	0.01	1.00	0.01	0.01	-0.01	0.01	0.02	0.11
cibil_score	-0.01	0.00	-0.00	-0.02	-0.02	0.01	1.00	-0.02	-0.00	-0.03	-0.02	-0.77
residential_assets_value	0.01	-0.01	0.01	0.64	0.59	0.01	-0.02	1.00	0.41	0.59	0.53	0.01
commercial_assets_value	-0.00	0.01	-0.02	0.64	0.60	-0.01	-0.00	0.41	1.00	0.59	0.55	-0.01
luxury_assets_value	0.00	-0.01	0.00	0.93	0.86	0.01	-0.03	0.59	0.59	1.00	0.79	0.02
bank_asset_value	0.01	-0.01	-0.00	0.85	0.79	0.02	-0.02	0.53	0.55	0.79	1.00	0.01
loan_status	0.02	0.00	-0.00	0.02	-0.02	0.11	-0.77	0.01	-0.01	0.02	0.01	1.00

Task 3: Data cleaning and preprocessing.

In this step, we have cleaned the dataset so it can be trained with our chosen training algorithms. Below, are the steps on how the cleaning took order:

1- Based on our dataset and looking at our features, the first column ("loan_id") doesn't have a significant value in the dataset. Hence, we can drop the column.
2- Using label_encoder function from sklearn.preprocessing library to encode the categorial target column (loan_status being Approved/Rejected) to 0 indicating Approved (True), and 1 indicating Rejected (False).
3- We also used label_encoder to change the rest of the object type values to numerical for ease of processing and to find better correlation between the values

Task 4: Training and evaluation of three machine learning algorithms, analyze findings, and compare results.

Preprocessing:

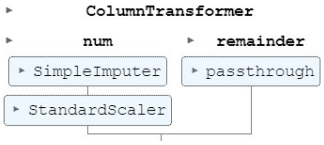
First, we created a pipeline using the numerical and categorical columns. In the Pipeline is:

1. Fill in the missing numerical values with the mean using a SimpleImputer

2. Scale the numerical columns using StandardScaler.

3. Fill in the missing categorical values with the most_frequent value using SimpleImputer

4. Encode the categorical columns using OneHotEncoder



The pipeline for the preprocessed dataset

After creating the preprocessed dataset, we will apply the pipeline to the dataset. After this step, the data is ready to be trained.

We have also divided our dataset into training, validation, and test sets (60-20-20)

*Our dataset is small and the number of instances in the validation and training sets is not much. In these cases, we can skip the validation set and do an 80-20 for the training and test sets. However, for the sake of following the common practice pattern, we also considered the validation set.

Training:

The training of our prediction can be done by regression models. For our prediction, we are training our dataset based on the logistic regression model, random forest model, and XGBoost model.

1- Logistic regression model: we chose this model because of the properties of our dataset. Our target can be considered a binary problem with two outcomes (in our case, approved or rejected). Moreover, logistic regression is efficient with small datasets.
2- Random Forest Model: Random Forest is commonly used for classification tasks, where the goal is to assign a label or category to input data. Unlike the regular decision trees that are prone to overfitting, random forests can handle noises or outliers.
3- XGBoost: which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree (GBDT). We can use this model since it's based on shallow decision trees and also doing gradient descent in the meantime to get the best result.

Analyzing the results:

In our problem accuracy is more important than the other metrics. We want to know the eligible people for a loan to get approved. Since we are doing classification, we would care about our types of errors. In addition to True Positives and True Negatives, we care about False positives more than False negatives. False Negative means, we are rejecting people who are eligible to get a loan and pay the loan back in due time. However, False Positive means that we are giving loans to people who are not eligible. If eligible people get rejected from a loan, they can try again, but if people who don't have the means to pay back the loan get approved, that will cause problems.

Hence, between our training models, we choose a model with the most accuracy (because less FP is more important to us) and compare the result with the performance of the classification. Between our training models, the Random Forest model had the best overall performance (by looking at the metrics and the confusion matrix). Here is a more specific analysis of the three training models:

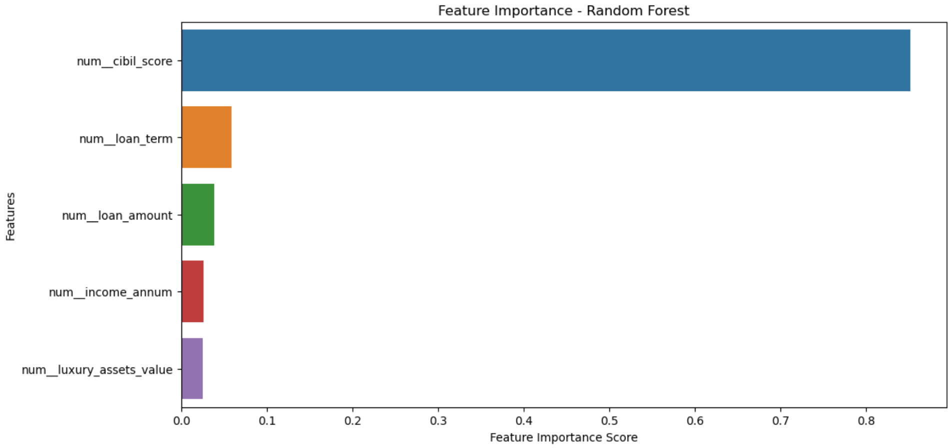
1- Logistic regression: the result of our accuracy and other metrics are worse than the other training models. This is mostly because in our problem there is no clear line for the decision boundary and different combinations of all features are determining the probability of loan approval. So, the is no clear distinction between the FP/FN from TP/TN.
2- Random forest: we chose this model as our best model because of the nature of the model and algorithm as well as consideration of the results. Since the correlation between the features is not that significant except for a few numbers, we can conduct random decision trees and form the forest. Our accuracy is very high in this model and as we wanted, the FP is the least.
3- XGBoost: This is also a good training model for our dataset. The results are also almost as good as random forest. However, even though XGBoost is a powerful model, for simpler and smaller datasets like ours it's taxing on the system. This model requires computational resources and it's more complex for hyperparameter tuning.

In conclusion, based on our dataset with the assumption that we are not adding external data or our data is not changing internally, random forest is the best model. However, for bigger datasets or in our case more important clients with more sensitive outcomes (like corporates or people with important status) it's better to use XGBoost since it can use Gradient boosting and will do the calculations in more detail.

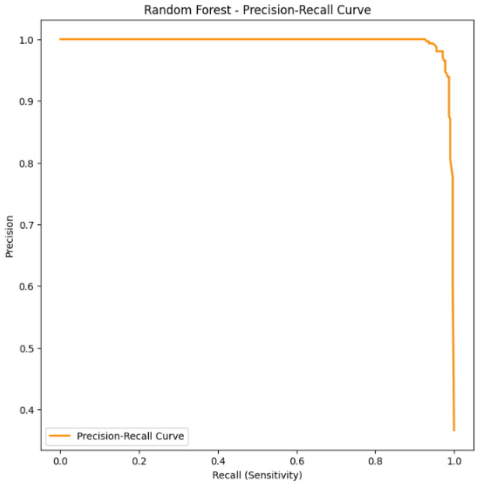
Task 5: 3 graphs for the best performing algorithm.

In the 3 graphs below, we show three different evaluations. First, we are seeing the feature importance of our model after the training which indicates that our assumptions and correlation were correct preprocessing. Second, The precision-recall curve shows the tradeoff between precision and recall for different threshold. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).. Last, we have the ROC curve which shows the TPR vs. FPR at different classification thresholds.

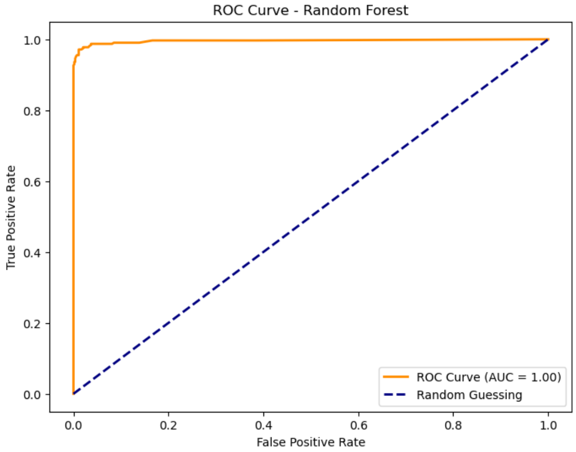
Feature Importance in descending order:



Precision-Recall Curve



ROC Curve:



Task 6: Any limitations you have run into.

1- One of our main limitations for determining the main features that have the biggest correlation to the target. After seeing the values in the correlation matrix, it was a hard task to choose the features that we wanted to work with.

2- Deciding the use label_encoding before the preprocessing to use correlation matrix or using OneHotEncode method within the preprocessing.

3- Right now that our dataset is small, we can use random forest over XGBoost. But, when our dataset gets bigger and more populated, the random forest has some shortcomings such as computational complexity for all the decision trees in the forest, bias toward dominant classes, and hyperparameter sensitivity.

4- Choosing appropriate plots for the result was a challenge since most of the graphs were for regression models or did not work well with a small dataset.

Appendix 1:

Loan Approval Prediction Notebook Using Logistic Regression, Random Forest, XGBoot

In [ ]:

```
#Incase the XGBoost library is not installed, please uncomment the below line.
#pip install xgboost
```

In [ ]:

```
#importing the necessary libraries
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import warnings; warnings.filterwarnings(action='once')
```

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
```

Importing the dataset.

In [ ]:

```
#Raw content URL for loan_approval_dataset.csv
url = "https://raw.githubusercontent.com/hastighsh/Credit-Card-Loan-Prediction/main

#Read the data into a DataFrame
df = pd.read_csv(url, delimiter=',')
df
```

In [ ]:

```
df.head()
```

In [ ]:

```
df.describe()
```

In [ ]:

```
#checking the dataframe information
df.info()
```

Performing Cleaning and Showing 3 Different EDAs.

1- Cleaning the dataset from redundant columns and make the values into integer.

In [ ]:

```
# dropping loan_id from the dataset since it has no value to the training
df.drop(columns="loan_id", inplace=True)

# removing the space before the column names in the rest of the dataset
df.rename(columns=lambda x: x.strip(), inplace=True)
df.info()
```

In [ ]:

```
# # Create a LabelEncoder instance
label_encoder = LabelEncoder()
#
# # Apply label encoding to the 'education' column
df['education'] = label_encoder.fit_transform(df['education'])
```

```
#
# # Apply label encoding to the 'self_employed' column
df['self_employed'] = label_encoder.fit_transform(df['self_employed'])
#
df['loan_status'] = label_encoder.fit_transform(df['loan_status'])
# #changing the data type of the column to int64 from int32
df['education'] = df['education'].astype('int64')
df['self_employed'] = df['self_employed'].astype('int64')
df['loan_status'] = df['loan_status'].astype('int64')
```

In [ ]:

```
df
```

In [ ]:

```
df.info()
```

2- Perform EDA on the dataset to understand the distribution of the data and identify any trends or patterns.

In [ ]:

```
# plotting a histogram of the data using hist()
df.hist(figsize=(20, 16))
plt.show()
```

In [ ]:

```
# Pairwise plot is a favorite in exploratory analysis to understand the relationshi
plt.figure(figsize=(10,8), dpi= 80)
sns.pairplot(df, kind="scatter", plot_kws=dict(s=80, edgecolor="white", linewidth=2
plt.show()
```

In [ ]:

```
# Plotting Correlation Matrix
correlation_matrix = df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix Heatmap")
plt.show()
```

3- Perform pipelining and preprocessing.

In [ ]:

```
# importing the libraries
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.impute import SimpleImputer
```

In [ ]:

```
# Define the numeric columns
X = df.drop('loan_status',axis=1)
y = df['loan_status']
num_cols = X.select_dtypes(include='number').columns.to_list()
# cat_cols = X.select_dtypes(exclude='number').columns.to_list()
```

In [ ]:

```
# Create pipelines for numeric columns
num_pipeline = make_pipeline(SimpleImputer(strategy='mean'), StandardScaler())
```

In [ ]:

```
#Preprocessing for numerical data
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, num_cols)],
#         ('cat', categorical_transformer, cat_cols)],
         remainder='passthrough'
)
```

In [ ]:

```
# Create and apply the preprocessing pipeline
data_prepared = preprocessor.fit_transform(X)
feature_names = preprocessor.get_feature_names_out()
data_prepared = pd.DataFrame(data=data_prepared, columns=feature_names)
```

In [ ]:

```
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import pandas as pd
import seaborn as sns
import xgboost as xgb
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

This cell below is for feature selection

In [ ]:

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

#inital the estimator as RandomForestClassifier
y = y.ravel()
estimator = RandomForestClassifier(n_estimators=100, random_state=42)
#use RFE to choose the attribute
selector = RFE(estimator, n_features_to_select=5, step=1)
selector = selector.fit(data_prepared, y)
#get attributes' name
selected_features = data_prepared.columns[selector.support_]
```

The cell below is a generic function to calculate accuracy of our training models to find the best training process

In [ ]:

```
#generic function to calculate accuracy
#inputs: model, model_name, y_test, X_test, X_train, y_train

def calculate_accuracy_classifier(model, model_name, y_test, X_test, X_train, y_tra
    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Calculate accuracy and other metrics
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Cross-validation
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    mean_score = scores.mean()

    print(f'{model_name} Accuracy: {accuracy}')
    print(f'{model_name} Mean Absolute Error: {mae}')
    print(f'{model_name} Mean Squared Error: {mse}')
    print(f'{model_name} R-squared: {r2}')
    print(f'{model_name} Confusion Matrix:\n{conf_matrix}')
    print(f'{model_name} Classification Report:\n{class_report}')
    print(f'{model_name} Cross-Validation Mean Accuracy: {mean_score}')

    # Plot a heatmap of the confusion matrix
    sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
                xticklabels=model.classes_, yticklabels=model.classes_)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title(f'Confusion Matrix - {model_name}')
    plt.show()
```

The cells below are for 3 different training which are: logistic regression, xgboost, random forest and calculating accuracy of each of them

In [ ]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Assuming 'data_prepared' is your feature matrix and 'y' is your target variable
# Replace 'selected_features' with the actual features you want to use for predicti
X = data_prepared[selected_features]
y = y.ravel()

# Split the data into training, validation, and testing sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_sta
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, rand

# Print the shapes of the datasets
print(f"Training set: {X_train.shape}, Validation set: {X_val.shape}, Test set: {X_
```

In [ ]:

```
#Train a logistic regression model and evaluate its performance on validation and t
#inputs: X_train, y_train: Training data and labels | X_val, y_val: Validation data

def logistic_regression(X_train, y_train, X_val, y_val):

    # Initialize the logistic regression model
    logreg_model = LogisticRegression()

    # Train the logistic regression model on the training set
    logreg_model.fit(X_train, y_train)
    # Calculate accuracy and other metrics on the validation set
    calculate_accuracy_classifier(logreg_model, "Logistic Regression (Validation)",
```

In [ ]:

```
#Train an XGBoost model and evaluate its performance on validation and test sets.
#inputs: X_train, y_train: Training data and labels | X_val, y_val: Validation data

def xgboost(X_train, y_train, X_val, y_val):

    # Initialize the XGBoost model
    xgb_model = xgb.XGBClassifier()

    # Train the XGBoost model on the training set
    xgb_model.fit(X_train, y_train)

    # Calculate accuracy and other metrics on the validation set
    calculate_accuracy_classifier(xgb_model, "XGBoost (Validation)", y_val, X_val,

    return xgb_model
```

In [ ]:

```
#Train a Random Forest model and evaluate its performance on validation and test se
#inputs: X_train, y_train: Training data and labels | X_val, y_val: Validation data

def random_forest(X_train, y_train, X_val, y_val):

    # Initialize the Random Forest model
    rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

    # Train the Random Forest model on the training set
    rf_model.fit(X_train, y_train)

    # Calculate accuracy and other metrics on the validation set
    calculate_accuracy_classifier(rf_model, "Random Forest (Validation)", y_val, X_

    return rf_model
```

In [ ]:

```
logistic_regression(X_train, y_train, X_val, y_val)
```

In [ ]:

```
xg_model = xgboost(X_train, y_train, X_val, y_val)
```

In [ ]:

```
rf_model = random_forest(X_train, y_train, X_val, y_val)
```

Plotting 3 Different Graphs for the Chosen Model (Random Forest).

In [ ]:

```
from sklearn.metrics import roc_curve, auc

# Assuming 'random_forest_model' is your trained Random Forest model
y_scores = rf_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f}
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend(loc='lower right')
plt.show()
```

In [ ]:

```
from sklearn.metrics import precision_recall_curve

y_probs = rf_model.predict_proba(X_test)[:, 1]
y_true = y_test

# Create a precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_true, y_probs)

# Plot the precision-recall curve
plt.figure(figsize=(8, 8))
plt.plot(recall, precision, color='darkorange', lw=2, label='Precision-Recall Curve
plt.xlabel('Recall (Sensitivity)')
plt.ylabel('Precision')
plt.title('Random Forest - Precision-Recall Curve')
plt.legend()
plt.show()
```

In [ ]:

```
importances = rf_model.feature_importances_
feature_names = X.columns
print(feature_names)

# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(12, 6))
sns.barplot(x=importances[indices], y=feature_names[indices])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title('Feature Importance - Random Forest')
plt.show()
```

Appendix 2:

- Loan-Approval-Prediction-Dataset
- Loan_Approval_Prediction.ipynb