Studio
Shodwe

Loan Eligibility prediction using Data Mining Techniques

Group Members:

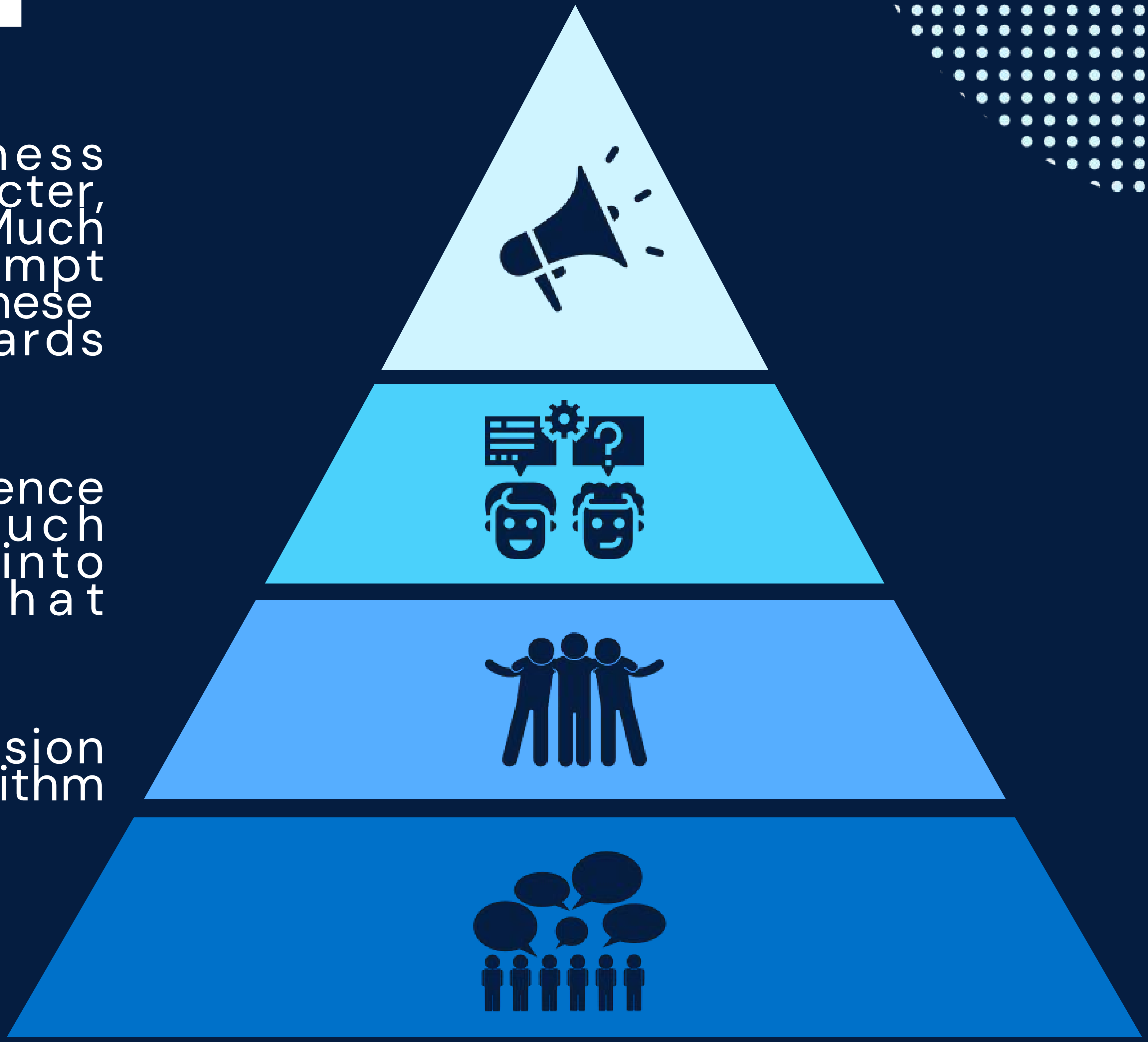
Aliyu Adenuga – 0255985
Sai Kiran Vaddadi – 0265024
Ridwan Mustapha – 0254138

INTRODUCTION

- Understanding Borrower Creditworthiness underscores the basic 5Cs of lending (character, Condition, Collateral, Capital and Capacity. Much more than the basics is required to pre-empt certain behaviors of borrowers. The import of these lending decisions serves as a guide towards successful risk management.

- The combination of technology and data science through machine learning has made such quantification of those characteristics into categorical and numerical data that eventually aid predictions

- The choice of decision tree vis-a-viz regression analysis was to compare a classification algorithm with a regression algorithm



DESCRIPTION OF ALGORITHM USED

Regression Analysis

- Regression analysis is a statistical technique used to analyze relationships between variables; to predict future outcomes based on historical data
- By examining past data on borrower attributes, financial activity, and economic factors, regression models forecast the likelihood of successful repayment or possible default. This underscores the significance of predictive analysis in credit risk management like other spheres of life

DESCRIPTION OF ALGORITHM USED

Decision Tree

- Decision trees like other classification algorithms such as Random Forest, SVM, Logistic Regression, etc; create prediction models by dividing data based on key criteria as depicted by the features considered in the model.
- In our Predictive analysis, this enables identification of borrower groups with varying levels of creditworthiness.
- Such segmentation facilitates focused risk management and optimization of loan portfolio performance.

IMPLEMENTATION DESCRIPTION

REGRESSION ANALYSIS

The regression leverages the intercept and the behavioral characteristics of the explanatory or independent variables to predict the outcome(s) of the dependent variable.

Preprocessing

```
# Defining a function to remove outliers using Z-score
def remove_outliers_zscore(df, columns, threshold=3):
    z_scores = np.abs((df[columns] - df[columns].mean()) / df[columns].std())
    df_cleaned = df[(z_scores < threshold).all(axis=1)]
    return df_cleaned
```

```
[48] # List of columns/features to check for outliers
     columns_to_check = [
         'emp_length', 'homeownership', 'annual_income', 'debt_to_income', 'annual_income_joint',
         'debt_to_income_joint', 'delinq_2y', 'months_since_last_delinq', 'earliest_credit_line',
         'inquiries_last_12m', 'total_credit_lines', 'open_credit_lines', 'total_credit_limit',
         'total_credit_utilized', 'num_collections_last_12m', 'num_historical_failed_to_pay',
         'months_since_90d_late', 'current_accounts_delinq', 'total_collection_amount_ever',
         'current_installment_accounts', 'accounts_opened_24m', 'months_since_last_credit_inquiry',
         'num_satisfactory_accounts', 'num_accounts_120d_past_due', 'num_accounts_30d_past_due',
         'num_active_debit_accounts', 'total_debit_limit', 'num_total_cc_accounts',
         'num_open_cc_accounts', 'num_cc_carrying_balance', 'num_mort_accounts',
         'account_never_delinq_percent', 'tax_liens', 'public_record_bankrupt', 'loan_purpose',
         'application_type', 'loan_amount', 'term', 'interest_rate', 'installment', 'grade',
         'sub_grade', 'balance', 'paid_total', 'paid_principal', 'paid_interest'
```

```
[50] # Remove outliers using Z-score method
     data_cleaned = remove_outliers_zscore(data, columns_to_check)
```

```
[51] # Print the shape of the cleaned DataFrame
     print("Original shape:", data.shape)
     print("Shape after removing outliers:", data_cleaned.shape)

     Original shape: (10000, 46)
     Shape after removing outliers: (0, 46)
```

```
[54] from sklearn.preprocessing import MinMaxScaler
```

```
[55] # Create a MinMaxScaler object
     scaler = MinMaxScaler()
```

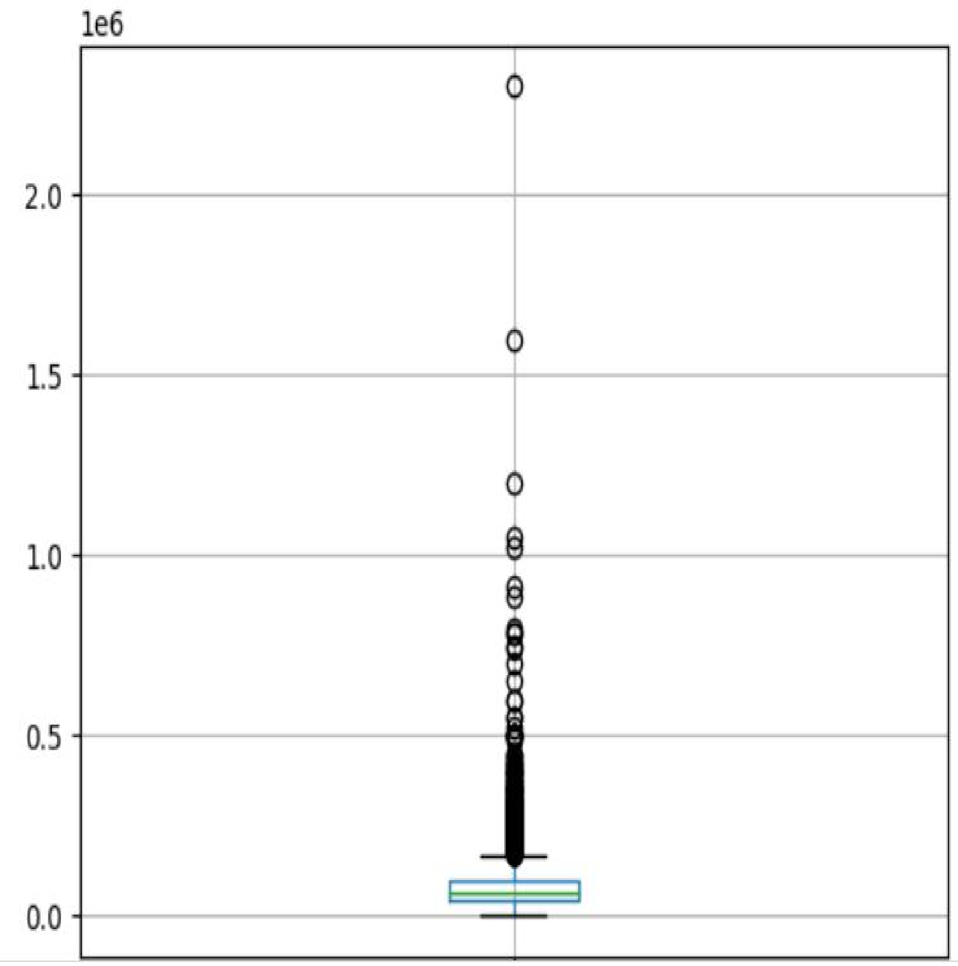
```
[57] # Fit and transform the scaler to normalize the feature values
     data_normalized = pd.DataFrame(scaler.fit_transform(data), columns=data.columns)
```

IMPLEMENTATION DESCRIPTION

Preprocessing

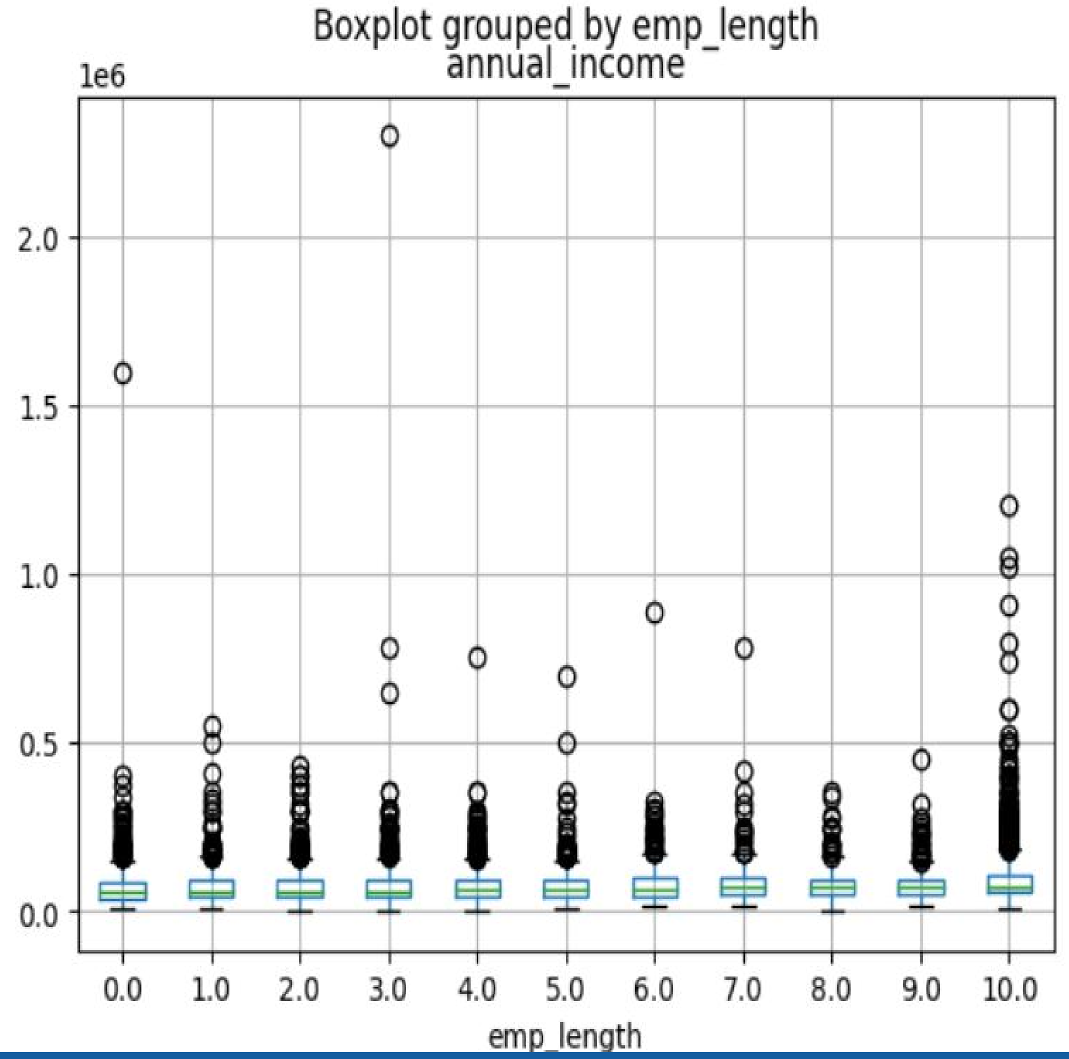
```
[ ] dataset.boxplot(column='annual_income')
```

```
<Axes: >
```

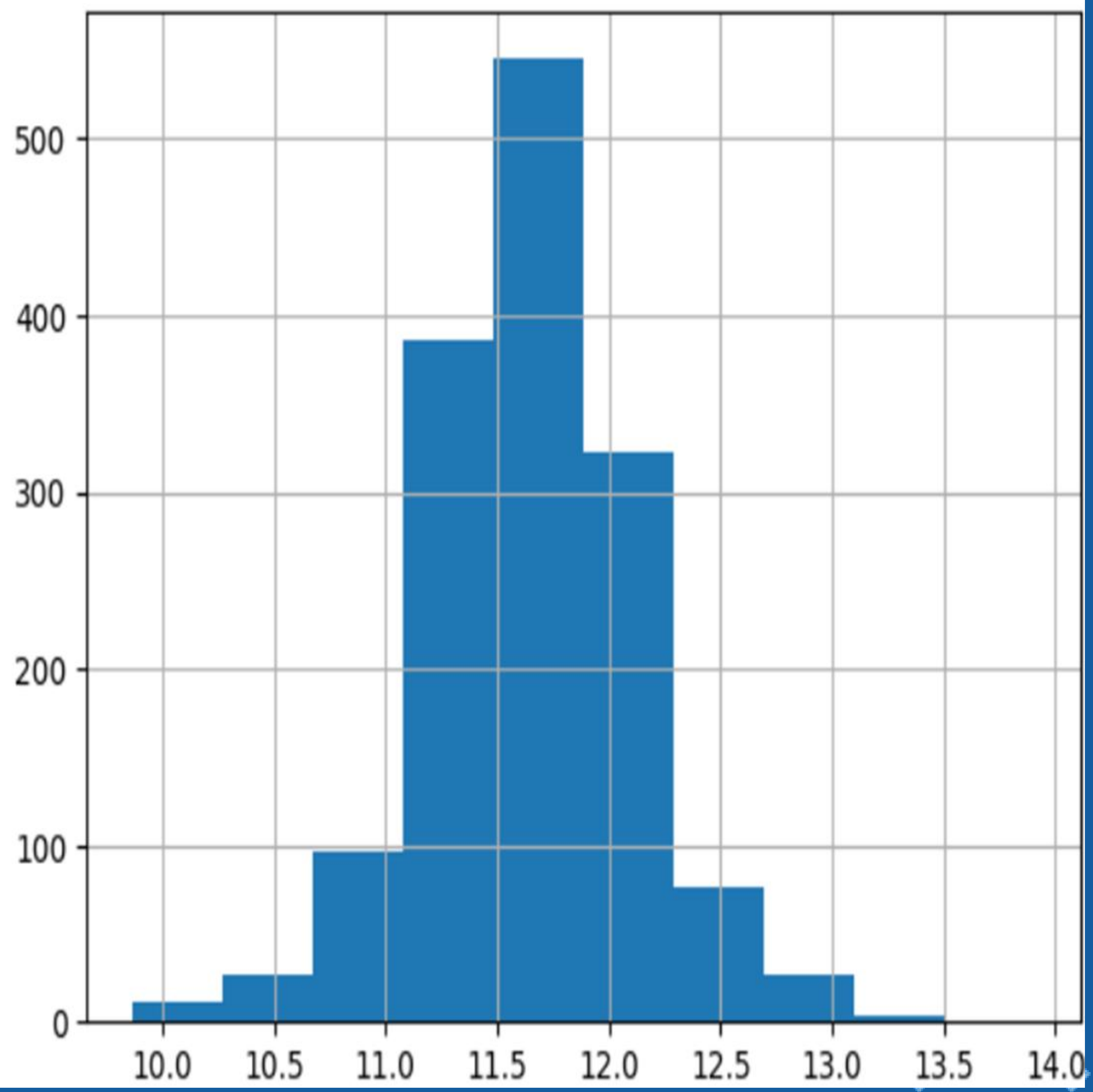



```
dataset.boxplot(column='annual_income', by= 'emp_length')
```

```
<Axes: title={'center': 'annual_income'}, xlabel='emp_length'>
```




```
<Axes: >
```



IMPLEMENTATION DESCRIPTION

Preprocessing

```
[59] # Print the first few rows of the normalized DataFrame
     print("Normalized DataFrame:")
     print(data_normalized.head())

Normalized DataFrame:
   emp_length  homeownership  annual_income  debt_to_income  \
0         0.3            0.0       0.039130        0.038393
1         1.0            1.0       0.017391        0.010744
2         0.3            1.0       0.017391        0.045087
3         0.1            1.0       0.013043        0.021659
4         1.0            1.0       0.015217        0.123558

   annual_income_joint  debt_to_income_joint  delinq_2y  \
0             0.100587              0.495696        0.0
1             0.100587              0.495696        0.0
2             0.100587              0.495696        0.0
3             0.100587              0.495696        0.0
4             0.034974              0.941503        0.0
```

```
[59]
   months_since_last_delinq  earliest_credit_line  inquiries_last_12m  ...
0                  0.316239              0.730769            0.206897  ...
1                  0.305647              0.634615            0.034483  ...
2                  0.230769              0.826923            0.137931  ...
3                  0.305647              0.846154            0.000000  ...
4                  0.305647              0.865385            0.241379  ...

   loan_amount  term  interest_rate  installment     grade  sub_grade  \
0     0.692308   1.0       0.341787     0.404847  0.333333   0.387097
1     0.102564   0.0       0.284822     0.089065  0.333333   0.322581
2     0.025641   0.0       0.459618     0.026468  0.500000   0.483871
3     0.528205   0.0       0.055014     0.412439  0.000000   0.064516
4     0.564103   0.0       0.341787     0.492317  0.333333   0.387097

    balance  paid_total  paid_principal  paid_interest
0  0.675397    0.048026        0.024604       0.240769
1  0.116284    0.011989        0.008716       0.035691
2  0.045616    0.006769        0.004384       0.025242
3  0.471331    0.079579        0.068668       0.134272
4  0.535754    0.055840        0.039246       0.179014

[5 rows x 46 columns]
```

IMPLEMENTATION DESCRIPTION

```
[ ] # Select the required features
    required_features = ['emp_length', 'homeownership', 'annual_income', 'debt_to_income', 'total_credit_limit', 'total_credit_utilized', 'num_accounts_120d_past_due', 'term', 'loan_amount', 'tax_liens']
```

```
[ ] # Create the new DataFrame
    new_df = df[required_features].copy()
    new_df.tail()
```

	emp_length	homeownership	annual_income	debt_to_income	total_credit_limit	total_credit_utilized	num_accounts_120d_past_due	term	loan_amount	tax_liens
9995	10.0	RENT	108000.0	22.28	199195	77963	0.0	36	24000	0
9996	8.0	MORTGAGE	121000.0	32.38	382061	101571	0.0	36	10000	0
9997	10.0	MORTGAGE	67000.0	45.26	346402	95421	0.0	36	30000	0
9998	1.0	MORTGAGE	80000.0	11.99	294475	27641	0.0	36	24000	0
9999	3.0	RENT	66000.0	20.82	91887	53413	0.0	36	12800	0

```
[ ] from sklearn.linear_model import LinearRegression
```

```
[ ] from sklearn.model_selection import train_test_split
```

```
[ ] from sklearn.metrics import r2_score, mean_squared_error
```

```
[ ] # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    y_test
```

```
8736    10000
1888     5000
4984    10000
811     15000
6885    20000
        ...
1074    28000
1396     5000
4010    12000
```

IMPLEMENTATION DESCRIPTION : DECISON TREE

For our predictive model, we have chosen Loan Status as our Dependent Variable with the possible outcome of either recommending the loan applicant as Eligible for approval or decline based on the relationship established with other features used as Independent Variables

IMPLEMENTATION DESCRIPTION

DECISION TREE

```
[1] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    %matplotlib inline
```

```
[2] from google.colab import files
    upload = files.upload()
```

```
Choose Files loans_full_schema.csv
• loans_full_schema.csv(text/csv) - 2536137 bytes, last modified: 2024-03-29 - 100% done
Saving loans_full_schema.csv to loans_full_schema.csv
```

```
[3] dataset = pd.read_csv('loans_full_schema.csv')
```

```
[4] # Select the required features
    required_features = ['emp_length', 'homeownership', 'annual_income', 'debt_to_income', 'grade', 'loan_amount', 'loan_status', 'tax_liens']
```

```
[5] # Create the new DataFrame
    new_dataset = dataset[required_features].copy()
```

```
[7] new_dataset.shape
```

```
(10000, 8)
```

```
[8] new_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   emp_length      9183 non-null   float64
 1   homeownership   10000 non-null  object
 2   annual_income   10000 non-null  float64
 3   debt_to_income  9976 non-null   float64
 4   grade           10000 non-null  object
 5   loan_amount     10000 non-null  int64
 6   loan_status     10000 non-null  object
 7   tax_liens       10000 non-null  int64
dtypes: float64(3), int64(2), object(3)
memory usage: 625.1+ KB
```

IMPLEMENTATION DESCRIPTION

DECISION TREE

```
[9] pd.crosstab(new_dataset['grade'], new_dataset['loan_status'], margins=True)
```

loan_status grade	Charged Off	Current	Fully Paid	In Grace Period	Late (16-30 days)	Late (31-120 days)	All
A	2	2341	99	10	3	4	2459
B	2	2896	108	11	7	13	3037
C	1	2467	134	19	12	20	2653
D	2	1323	74	18	10	19	1446
E	0	291	27	8	3	6	335
F	0	47	4	1	2	4	58
G	0	10	1	0	1	0	12
All	7	9375	447	67	38	66	10000

```
10] new_dataset.isnull().sum()

emp_length          817
homeownership         0
annual_income         0
debt_to_income       24
grade                 0
loan_amount           0
loan_status           0
tax_liens             0
dtype: int64
```

EXPERIMENTAL RESULT

REGRESSION ANALYSIS

```
[ ] # Train the regression model
    model = LinearRegression()
    model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
[ ] predictions = model.predict(X_test)
```

```
[ ] mse = mean_squared_error(y_test, predictions)
    print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 2.836734030105046e-23
```

DECISION TREE

```
[19] # Training the decision tree algorithm
     from sklearn.tree import DecisionTreeClassifier
```

```
[20] # Create a decision tree classifier with entropy as the criterion
     model = DecisionTreeClassifier(criterion='entropy', random_state=42)
     model.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=42)
```

```
[21] # Predicting on the test set
     y_pred = model.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, classification_report

accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

report = classification_report(y_test, y_pred)
print('Classification Report:\n', report)
```

```
Accuracy: 0.8671023965141612
Classification Report:
              precision    recall  f1-score   support

         0.0       0.05      0.05      0.05        96
         2.0       0.93      0.93      0.93      1709
         3.0       0.00      0.00      0.00        10
         4.0       0.00      0.00      0.00         8
         5.0       0.00      0.00      0.00        13

    accuracy                           0.87      1836
   macro avg       0.20      0.20      0.20      1836
weighted avg       0.87      0.87      0.87      1836
```

CONCLUSION

As compared to the outcome of Kumar (2016) which had 82% accuracy, ours reveals 86% accuracy which purports a relatively better predictive outcome

Studio
Shodwe

Thank's For your time!!!