RIGA TECHNICAL UNIVERSITY
Faculty of Computer Science, Information Technology and Energy

Report on the second practical assignment

Study course "Artificial Intelligence in Digital Humanities"

Student: **Zhen Pan (231AHG002)**

Teaching staff: **Alla Anohina-Naumeca**

Project link:
https://github.com/Grace128pan/AI_Assignment_2_Loan_Default_Prediction
Link to the dataset:
https://www.kaggle.com/datasets/msfasha/loan-default-prediction

Orange tool workflow

< a screenshot of the workflow created in the Orange tool>

Disclaimer： I decided to not use Orange Tool, because I have taken some Python courses this semester and the semester before and I really want to sharpen my skills again, and prepare myself for future career seeking.

The code is written with the help of Co-Pilot for automation and occasionally debugging with ChatGPT, because I was kind of running out of time.

However, **the whole design of the workflow, the structure of this assignment, and ideas are all mine.**

Co-Pilot and ChatGPT are merely tools for automation to reduce the workload and increase efficiency just so I can finish the assignment in time.

Part I

<this subsection should provide a general description of the dataset, accompanied by screenshots and references to the information sources used>

Description of the dataset

Dataset title: Loan Default Prediction

Dataset source: https://www.kaggle.com/datasets/msfasha/loan-default-prediction

Creator and/or owner of the dataset: MSFASHA

Description of the dataset problem domain: This dataset is designed for binary classification tasks for predicting whether a loan will default based on applicant info.

Dataset licensing conditions: MIT

Information about the method or procedure for collecting the dataset: Downloaded directly from Kaggle

Description of the dataset content

Number of data objects in the dataset: 1000 datapoints

Representation of features (attributes) of the dataset together with their roles in the Orange tool:

Input features: age, income, credit_score, dependent, home_owner

Target: loan_default

age	income	credit_score	dependents	home_owner	loan_default
64	100000	583	3	0	0
45	16803.24	685	4	0	0
69	15418.51	498	2	0	0
63	11474.46	533	4	0	0
46	11536	454	0	1	0
49	15359.96	580	2	0	0
53	32474.55	634	4	0	0
33	90129.23	464	4	0	0
39	24041.26	710	3	0	0
44	38996.2	333	4	1	1

Number of classes in the dataset: 2

Description of classes:

Class 1: Default, with value 1 in loan_default column, which are the datapoints with defaulted loans.
Class 2: Not Default, with value 0 in loan_default column, which are the datapoints with non-default loans.

Number of data objects belonging to each class:

<add rows to table as needed>

Class label	Number of data objects
Default	195
Not default	805

Description of features:

<add rows to table as needed>

Feature title	Explanation of the feature	Value type	Range of values
Age	Just age of the client	Integer	Min: 20, Max:69
Income	Income of the client	Float	Min: 4000.0, Max: 100000.0
Credit_score	Credit score of the client	Integer	Min: 300, Max: 849
Dependents	Number of dependents of the client	integer	Min: 0, Max: 4
Home_owner	Whether the client owns a home (1) or not (0)	Categorical	0 or 1
Loan_default	Whether the loan is defaulted (1) or not (0)	categorical	0 or 1

Data file structure:

<a screenshot showing all the columns in the data file and their values for at least some data objects>

age	income	credit_score	dependents	home_owner	loan_default
64	100000	583	3	0	0
45	16803.24	685	4	0	0
69	15418.51	498	2	0	0
63	11474.46	533	4	0	0
46	11536	454	0	1	0
49	15359.96	580	2	0	0
53	32474.55	634	4	0	0
33	90129.23	464	4	0	0
39	24041.26	710	3	0	0
44	38996.2	333	4	1	1

Information about missing values or outliers:

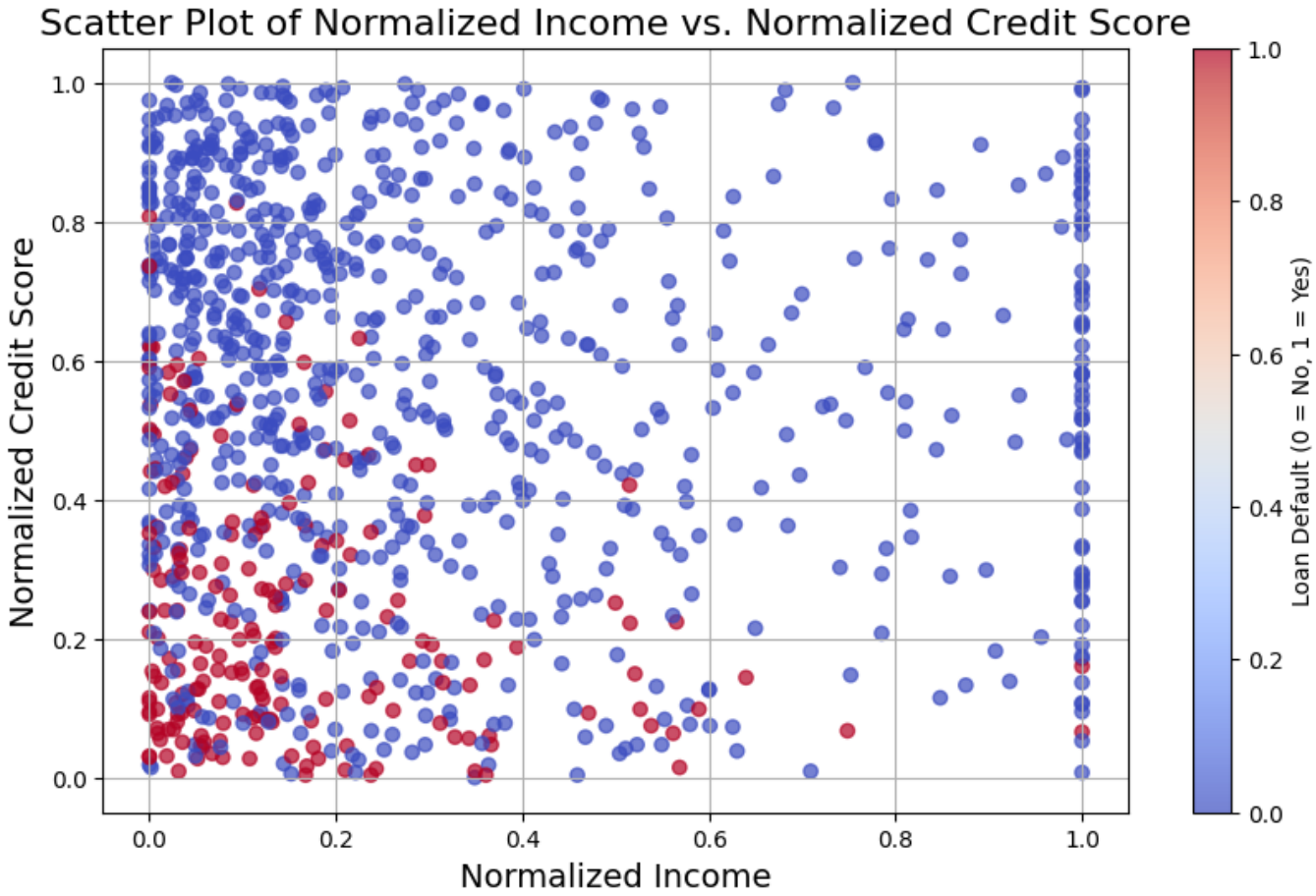
<a description of whether values of certain features (attributes) are missing in the dataset or whether there are outlier values and solutions that the students have used to solve the mentioned problems (demonstrated also by screenshots)>

There seems to be no missing values at all. The dataset is good enough.

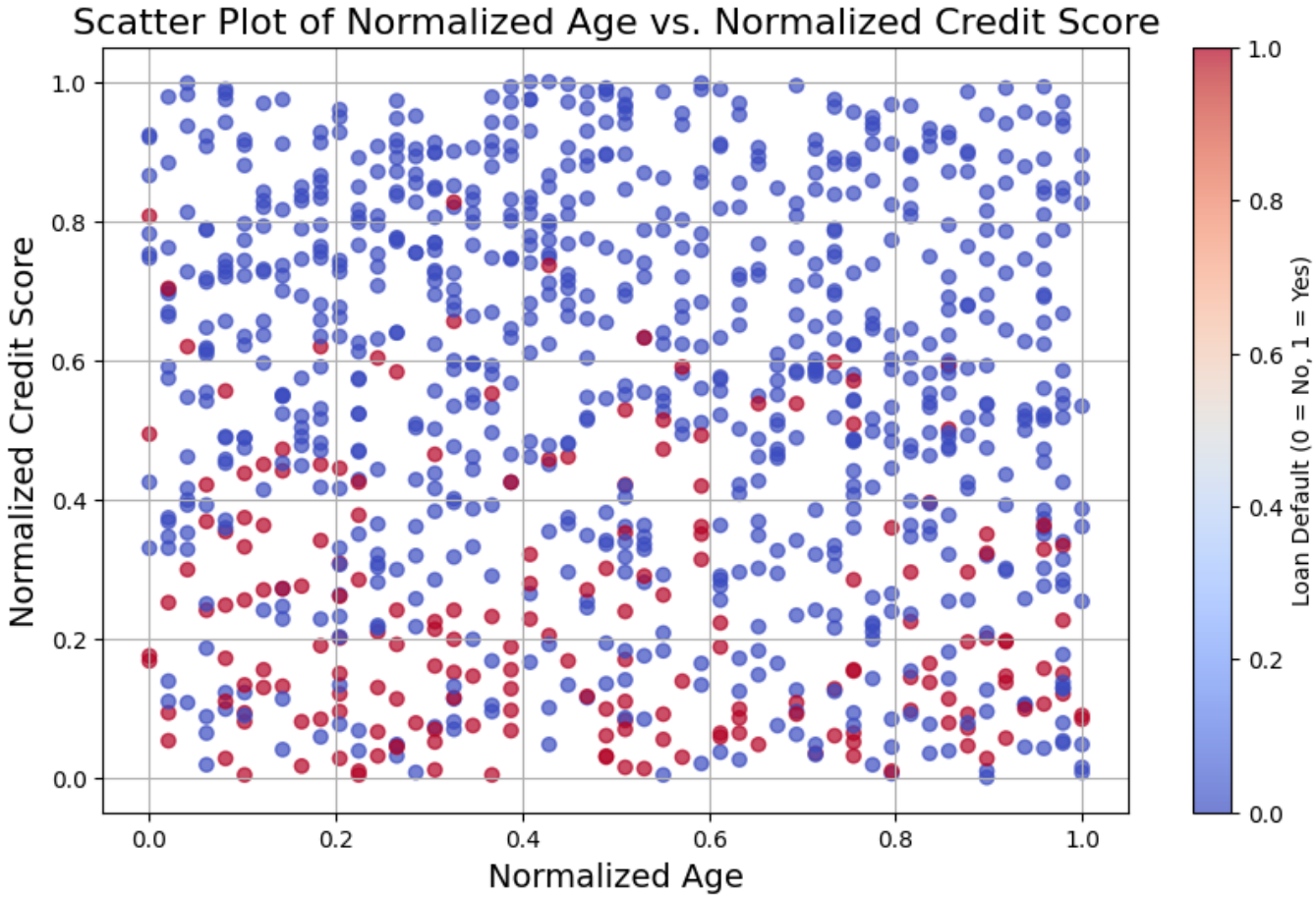
```
Number of NaN values before cleaning:
 age             0
income           0
credit_score     0
dependents       0
home_owner       0
loan_default     0
dtype: int64
Number of NaN values after cleaning:
 age             0
income           0
credit_score     0
dependents       0
home_owner       0
loan_default     0
dtype: int64
Data cleaning completed. Changes have been saved to the file.
```

Visual and statistical representation of the dataset

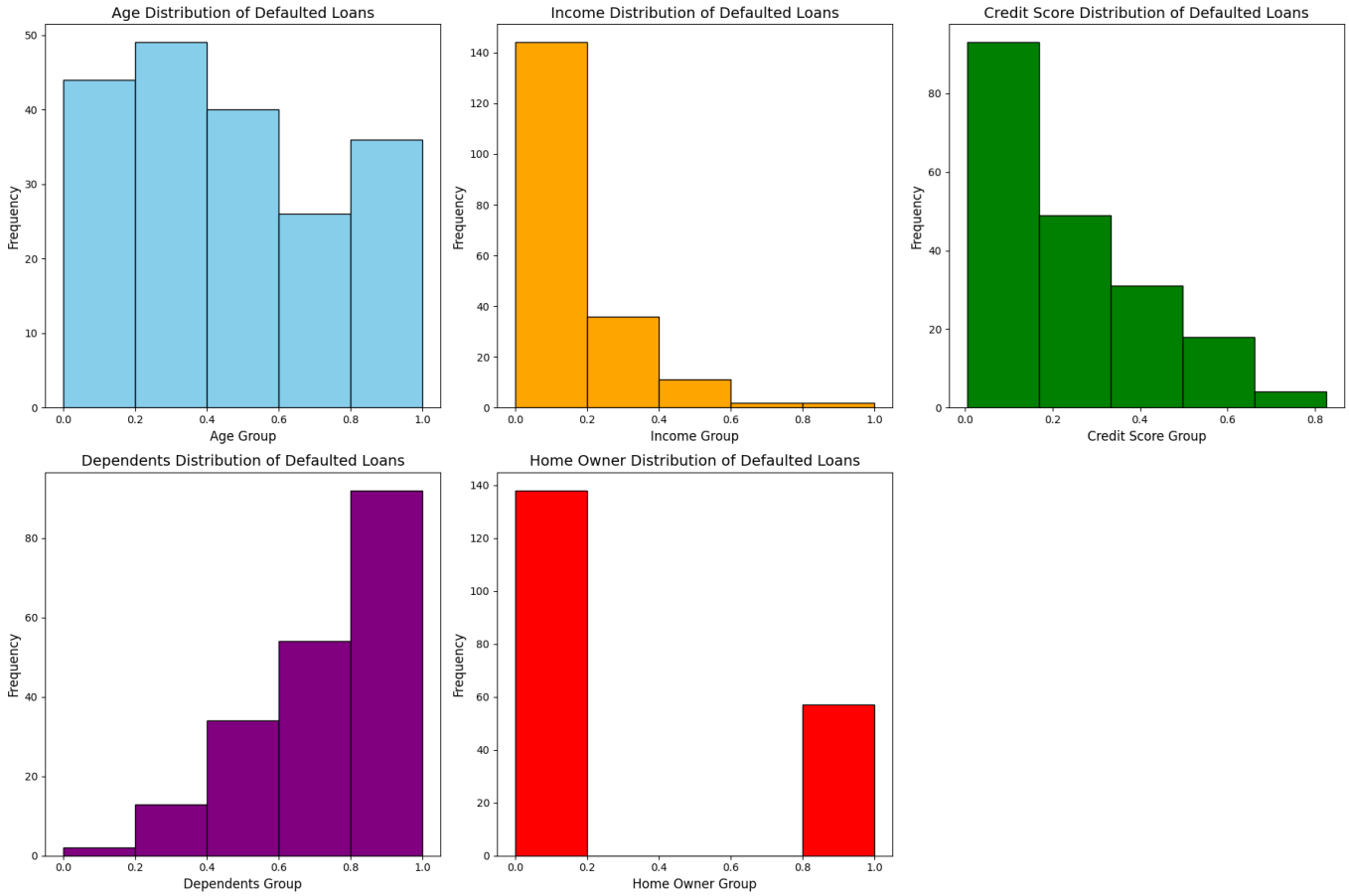
<a scatterplot screenshot>



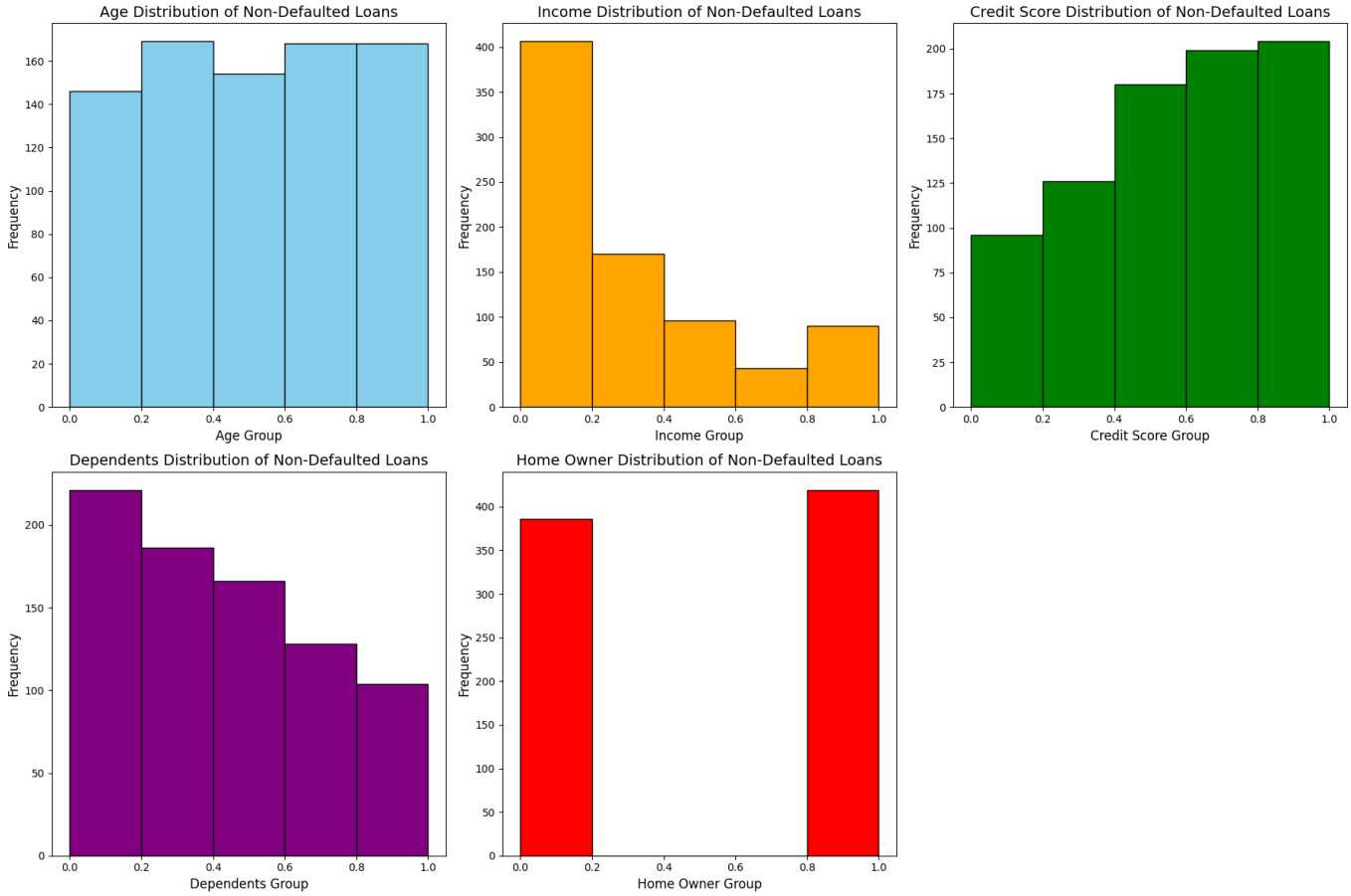
<a scatterplot screenshot>



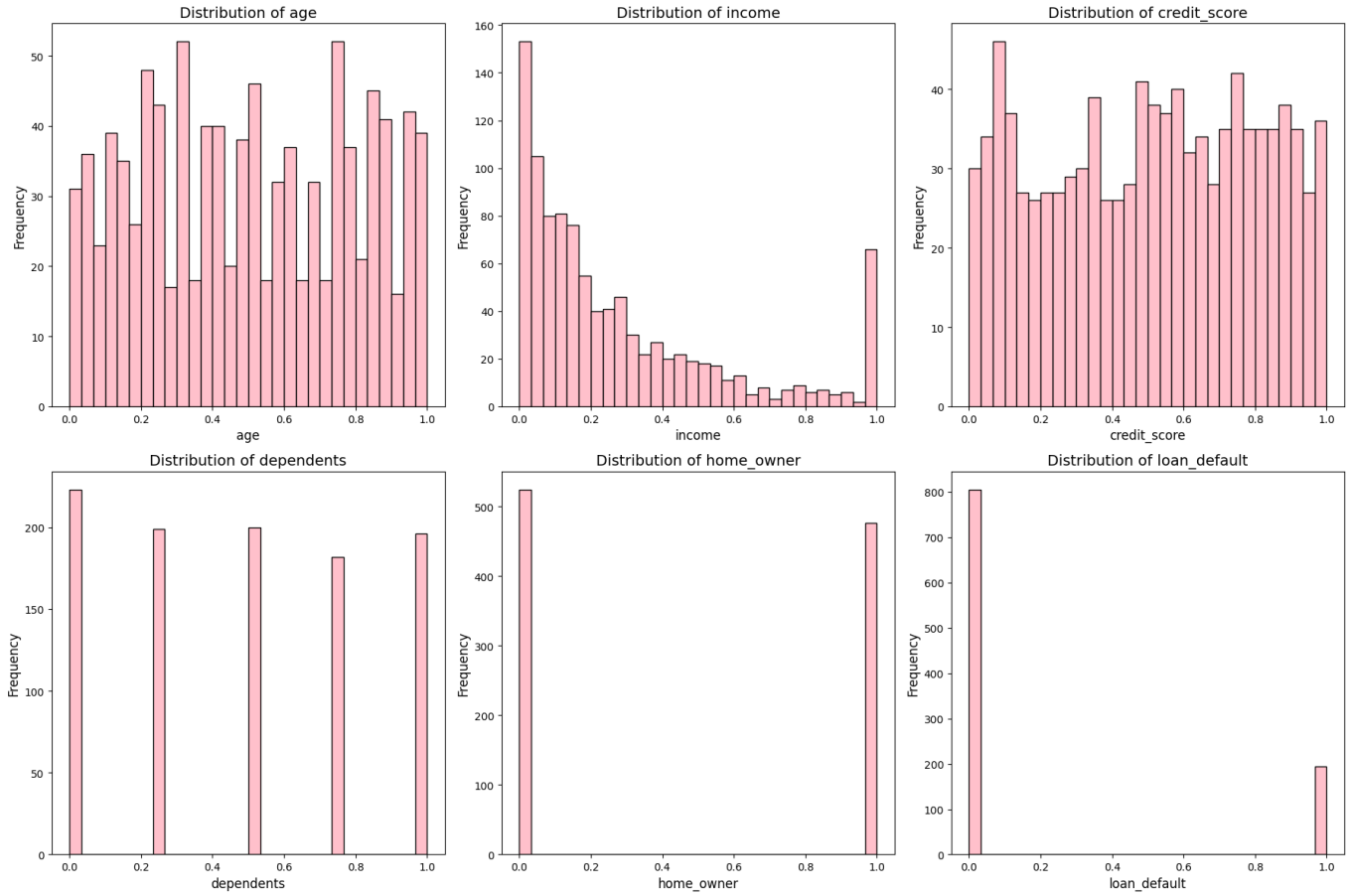
<a histogram screenshot>



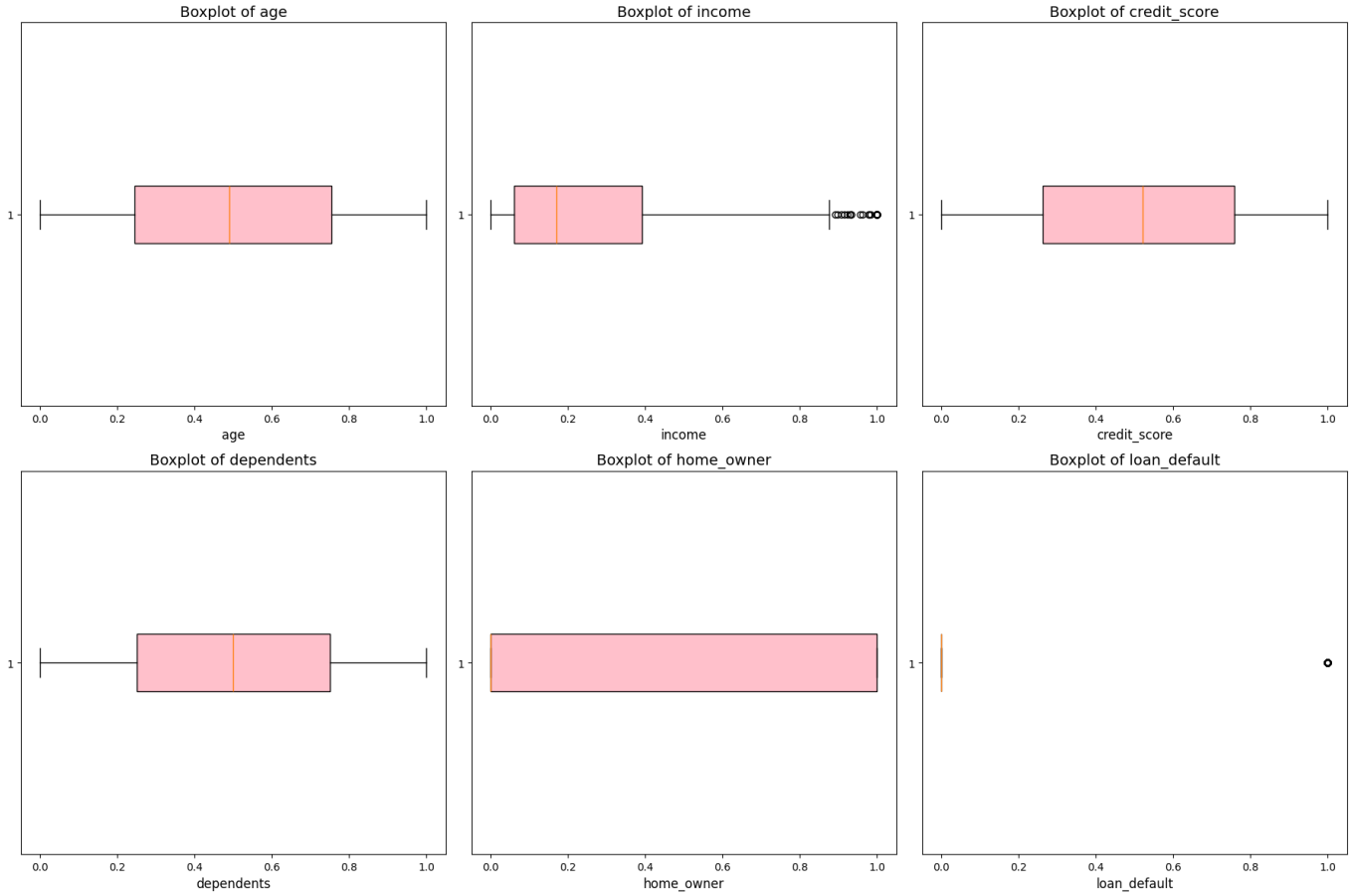
<a histogram screenshot>



<a screenshot of feature distribution>



<a screenshot of feature distribution>



<a screenshot with statistics>

	Data Type	Min Value	Max Value
age	int64	20.0	69.0
income	float64	4000.0	100000.0
credit_score	int64	300.0	849.0
dependents	int64	0.0	4.0
home_owner	int64	0.0	1.0
loan_default	int64	0.0	1.0

	Mean	Median (50th Percentile)	Standard Deviation
age	44.55200	44.00	14.338231
income	30496.36997	20284.82	27265.987560
credit_score	580.03800	586.50	158.897083
dependents	1.92900	2.00	1.433183
home_owner	0.47600	0.00	0.499674
loan_default	0.19500	0.00	0.396399

	Minimum	Maximum
age	20.0	69.0
income	4000.0	100000.0
credit_score	300.0	849.0
dependents	0.0	4.0
home_owner	0.0	1.0
loan_default	0.0	1.0

Answers to questions

<answers the questions below, referring to the screenshots above and providing an analysis of the results>

Are the classes in the dataset balanced, or does one class (or several classes) prevail?

No, there seems to be more datapoints in the Not-Default class than Default class.

Does the visual representation of the data allow you to see the structure of the data?

From the Income vs Credit Score Scatter Plot, we can see that:

There is not a clean separation, but there seems to be a pattern:
1. People with lower income and lower credit scores are more likely to default their loans;
2. People with lower income, but higher credit scores are less likely to default their loans;
3. People with higher income and lower credit scores rarely default their loans;
4. People with higher income and higher credit scores, never default their loans.

From the Age vs Credit Score Scatter Plot we can see:

1. There seems to be a rather even distribution of age among defaulted loans;
2. People with lower credit scores are more likely to default their loans;

Overall, the data has no apparent structures.

How many data groupings can be identified by studying the visual representation of the data?

There seems to be no clear lines between the plots.

Are the identified data groupings close to each other or far from each other?

They are merged together.

Conclusions arising from the analysis of statistical indicators

<analysis of statistical indicators by referencing specific values>
From the Data Summary of the Defaulted Loans, we can find that:

1. The Age: distribution for age of defaulted loans is fairly even, which indicates that it might not be the important factor of default.
2. The Income: Defaults are highly concentrated in the lowest income groups, indicating that this might be an important factor of default.
(But since the data contains more lower-income data points, so this might not be significant)
3. Credit Score: Defaults are highly concentrated in the groups with lowest credit scores, indicating that it is also an important factor.
4. Dependents: More dependents are associated with default, indicating it is an important factor.
5. Home Owners: Non-home owners are more likely to default.

From the Data Summary of the Non-Defaulted Loans we can notice that:

1. The Age: distribution of age of non-defaulted loans is fairly even, which indicates that it might not be the important factor of default.
2. The Income: Non-Defaults are highly concentrated in the lowest income groups, which is the same with the summary of default loans.
(This is because the data contains more people with lower income)
3. Credit Score: Non-Defaults are highly concentrated in the groups with highest credit scores, indicating that it is also an important factor.
4. Dependents: Fewer dependents are associated with non-default, indicating it is an important factor.
5. Home Owners: distribution is more or less the same.

This summary is in line with the scatter plots above.

Part II

<this subsection should describe the use of unsupervised machine learning algorithms, accompanied by screenshots and references to the information sources used>

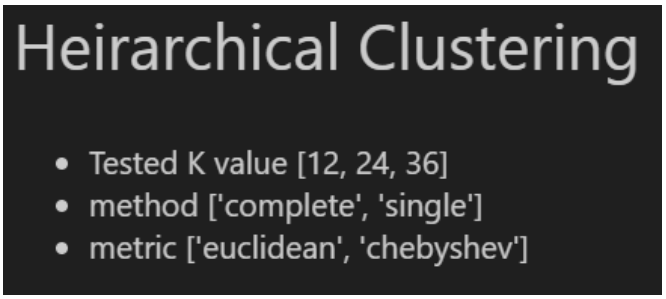
Hierarchical clustering

Hyperparameters available in the Orange tool:

<add rows to table as needed>

Hyperparameter	Description
Method of linkage	Different methods (single, complete, average, centroid and etc.) of clustering may affect the shape of the clustering distribution
Metrics	Different metric (Euclidean, Manhattan, Chebyshev, Minkowski, cosine and etc.) for calculating the distance will also affect the shape of the hierarchical clustering

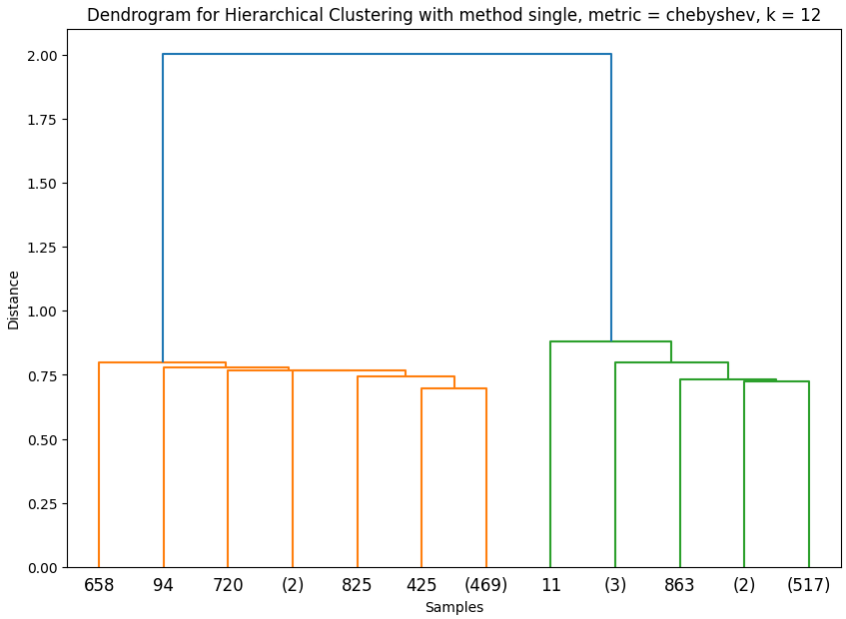
<a screenshot with hyperparameter values set for the algorithm>

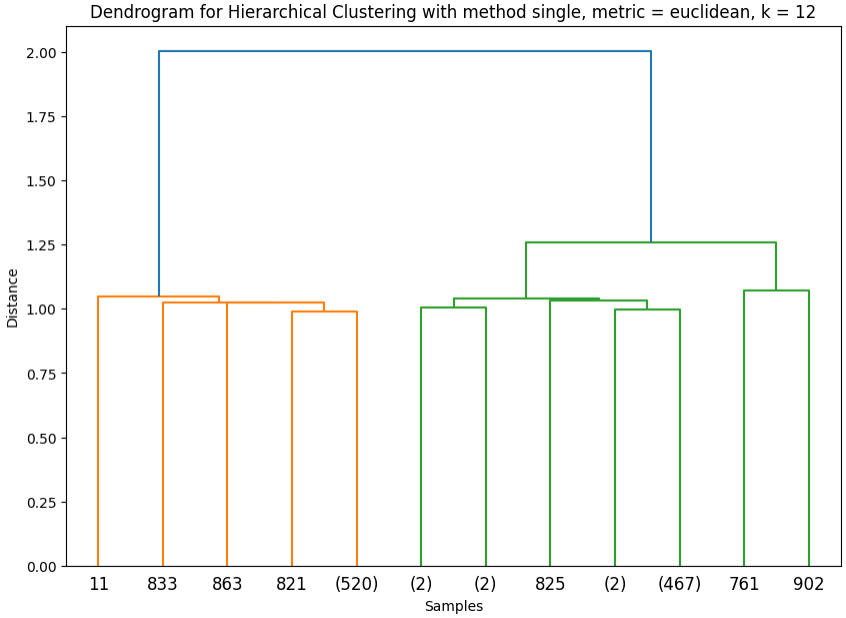


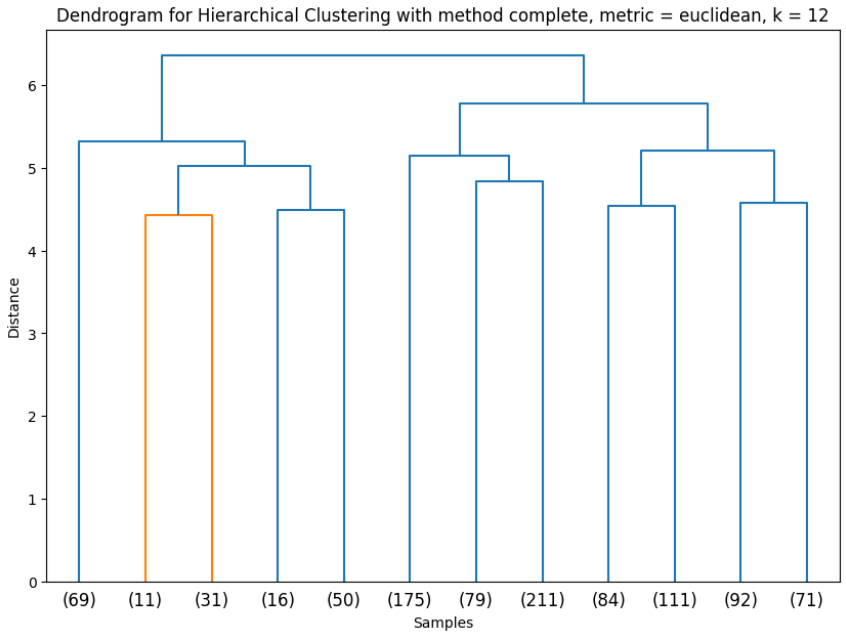

```
from sklearn.preprocessing import StandardScaler # type: ignore
from scipy.cluster.hierarchy import dendrogram, linkage # type: ignore
import matplotlib.pyplot as plt # type: ignore

# Step 1: Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_cleaned[['age', 'income', 'credit_score', 'dependents', 'home_owner']])

# Step 2: Perform hierarchical clustering
Z = linkage(data_scaled, method='single', metric='chebyshev')

# Step 3: Plot dendrograms with different cut-off values
plt.figure(figsize=(10, 7))
dendrogram(Z, truncate_mode='lastp', p=12)
plt.title('Dendrogram for Hierarchical Clustering with method single, metric = chebyshev, k = 12')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()
```

Description of experiments

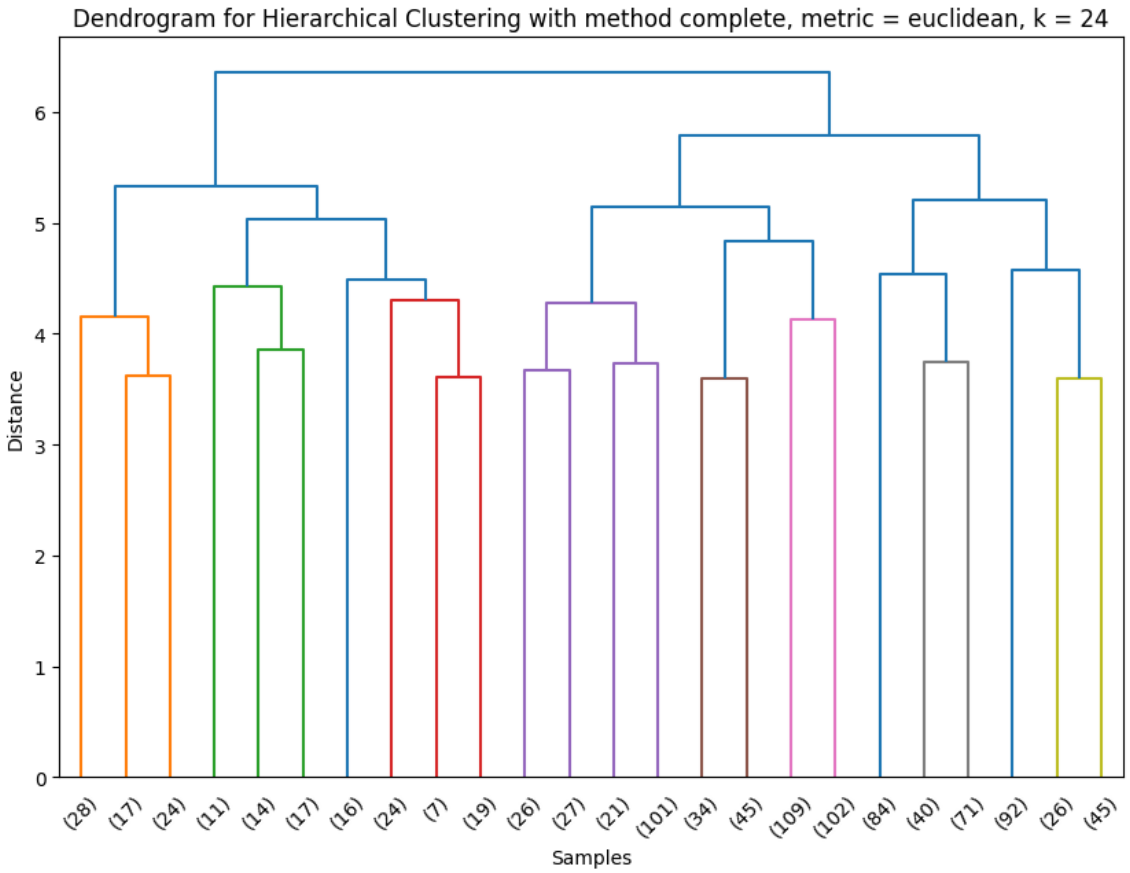
Different linkage methods and metrics with the same K:


Dendrogram for Hierarchical Clustering with method single, metric = chebyshev, k = 12
Distance
Samples
658 94 720 (2) 825 425 (469) 11 (3) 863 (2) (517)

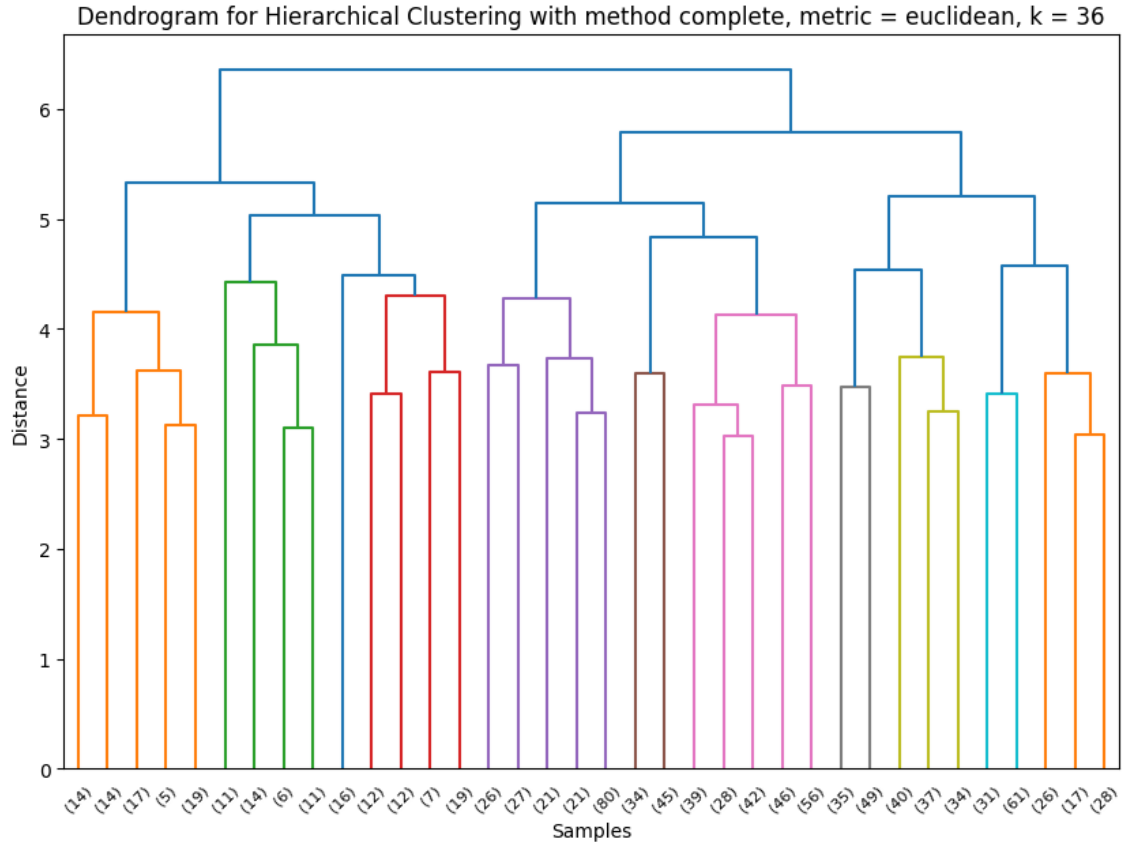

Dendrogram for Hierarchical Clustering with method single, metric = euclidean, k = 12
Distance
Samples
11 833 863 821 (520) (2) (2) 825 (2) (467) 761 902


Dendrogram for Hierarchical Clustering with method complete, metric = euclidean, k = 12
Distance
Samples
(69) (11) (31) (16) (50) (175) (79) (211) (84) (111) (92) (71)

<a screenshot for Experiment 1 with a certain position of the horizontal cut off line>



<a screenshot for experiment 2 with a certain position of the horizontal cut off line >



<a screenshot for Experiment 3 with a certain position of the horizontal cut off line >

Conclusions from experiments:

<a description of how the number and content of clusters change according to the position of the horizontal cut off line, and conclusions about whether class separation is achieved referring to and analysing the screenshots above>

For the Complete Linkage method (the last 3 graphs above)
With higher K, there would be more clusters.

At k = 12, the data forms 3 mother clusters:

* Cluster 1 (Orange): Samples 69, 11, 31, 16, 50,
* Cluster 2 (Green): Samples 175, 79, 211,
* Cluster 3 (Red): Samples 84, 111, 92, 71

At k = 24, the data forms still 3 mother clusters, but each cluster has more sub clusters and grand-sub clusters (totally 24). Some sub-clusters stay the same, while others split into two smaller groups.

Some of the sub clusters were separated into two, while other sub clusters remained the same.

* Cluster 1 (Orange): Samples 28, 17, 24, 11, 14, 17, 16, 24, 7, 19,
* Cluster 2 (Green): Samples 26, 27, 21, 101, 34, 45, 109, 102,
* Cluster 3 (Red): Samples 84, 40, 71, 92, 26, 45,

At k = 36, the data forms still 3 mother clusters, but each cluster has even more sub clusters and grand-sub clusters (totally 36). Some sub-clusters stay the same, while others split into two smaller groups.

* Cluster 1 (Orange): Samples 14, 14, 17, 5, 19, 11, 14, 6, 11, 16, 12, 12, 7, 19,
* Cluster 2 (Green): Samples 26, 27, 21, 21, 80, 34, 45, 39, 28, 42, 46, 56,
* Cluster 3 (Red): Samples 35, 49, 40, 37, 34, 31, 61, 26, 17, 28,

K-means algorithm

Hyperparameters available in the Orange tool:

<add rows to table as needed>

Hyperparameter	Description
K	In K-mean clustering, K controls how many clusters to produce. Higher K value would result in more clusters.

<a screenshot with hyperparameter values set for the algorithm>

K-Mean Clustering

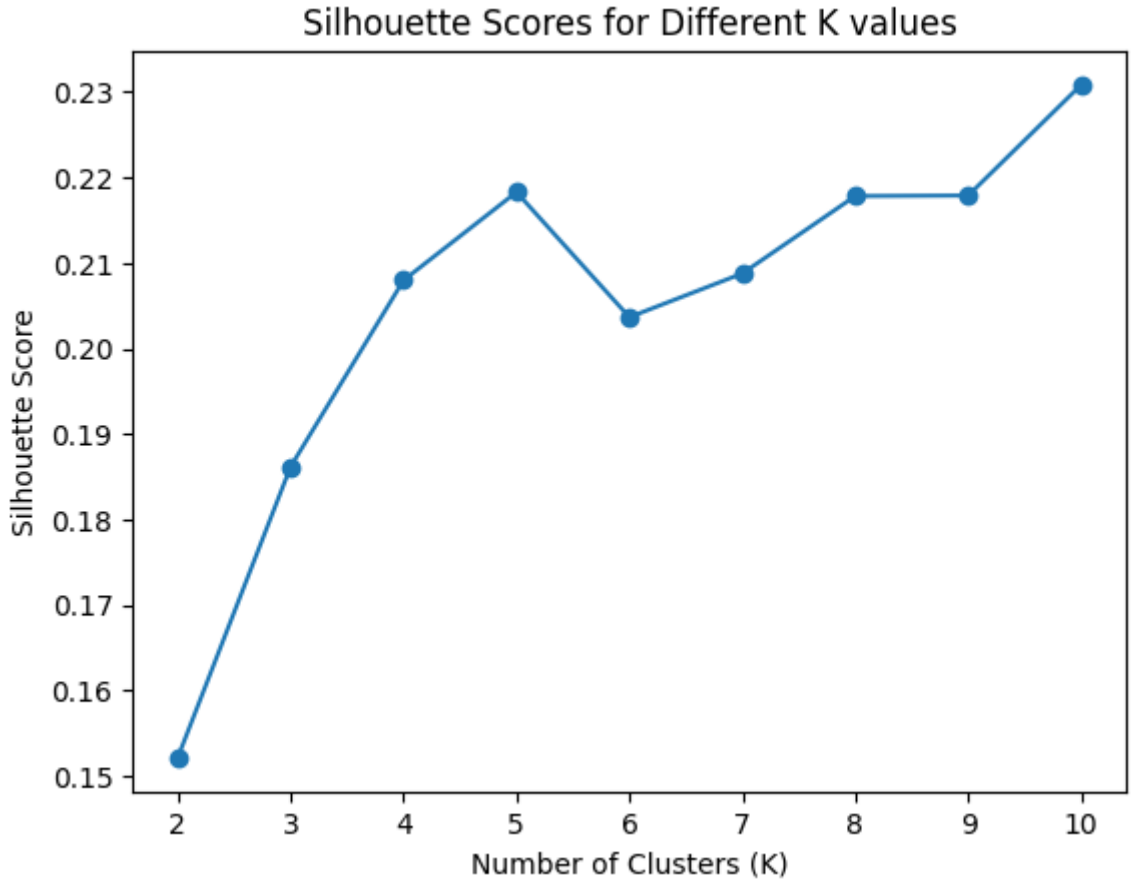
- K_range = [2, 3, 5, 8, 9, 10]

Experimented with:

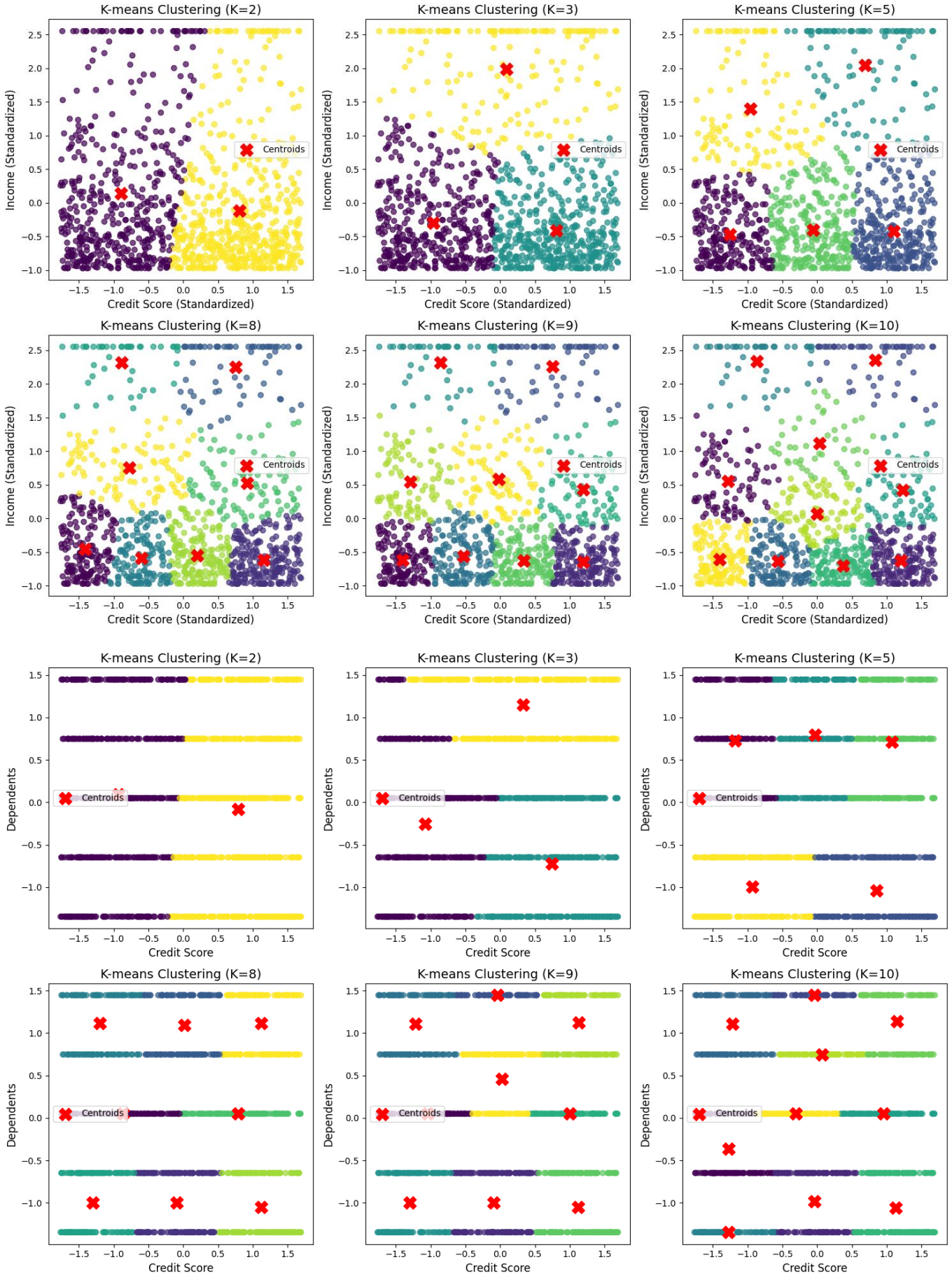
- Dependents vs Credit Score
- Income vs Credit Score
- All Features as Input Features

Description of experiments

<a screenshot of Silhouette coefficient with at least 5 different k values>



<a screenshot of the scatterplot according to the best value of the Silhouette coefficient>



Conclusions from experiments:

<a description of whether the classes are separable referring to and analysing the screenshots above>

From all the experiments above, I can see that:

1. The best features for clustering are: income, credit_score
2. When all features are put together, the clustering is not separated (perhaps I should try 3D visualization.)

Final conclusions

<conclusions whether the classes in the dataset are well or poorly separable based on the analysis of the performance of the two algorithms>

Based on the visualization, we can see that in K-mean clustering, there seems to be a very good boundary between different clusters.

Part III

<this subsection should describe the use of supervised machine learning algorithms, accompanied by screenshots and references to the information sources used>

Description of the selected algorithms

<a description of freely chosen algorithms and the rationale for their choice (except artificial neural network)>

Title of the first algorithm: Logistic Regression

Description of the first algorithm: Logistic Regression is very good for binary classification and therefore, it is chosen for this assignment.

Title of the second algorithm: Decision Tree

Description of the second algorithm: Decision tree is good for both regression tasks and classification, and is easy to train as well (it doesn't require TensorFlow, which I have trouble getting to work).

Description of hyperparameters

<a description of the hyperparameters available in the Orange tool should be given for each of the algorithms, adding rows to the table as necessary>

Hyperparameter	Description and values
Artificial Neural Networks - MLP	
Learning Rate	Learning rate controls how much the model adjusts its weights during training based on the error value. I tried with 0.00001, 0.001, 0.01
Number of layers and artificial neurons in each layer	Artificial neurons are responsible to "react" toward input and pass the output to the next neuron. Its weights would be adjusted through backpropagation. Increasing the number of layers and number of artificial neurons in each layer would allow the MLP to capture more details, but too many would cause the model to capture the noise instead of meaningful details. I tried with (32, 16), (16, 8), (128, 64), (512, 256), (64, 32, 16), (256, 128, 64, 32, 16)
Logistic Regression	

Tested with different values of c	C is responsible of controlling the strength of regularization. My understanding is that it controls the trade-off between risk of overfitting and accuracy of the prediction I tried with 0.0001, 0.001, 0.01, 0.1, and 1
Decision Tree	
Max depth	It controls the depth of the tree. I tried with 5, 10, 20.
Min sample split	It controls the minimum number of samples required to split internal nodes I tried with 10, 20, 40

Information about test and training datasets

<a screenshot of splitting dataset into test and training datasets>

```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Count the occurrences of each value in the target variable for the training set
train_counts = pd.Series(y_train).value_counts()

# Count the occurrences of each value in the target variable for the testing set
test_counts = pd.Series(y_test).value_counts()

# Display the counts
print("Training Set Counts:")
print(train_counts)

print("\nTesting Set Counts:")
print(test_counts)
```

✓ 0.0s

```
Training Set Counts:
loan_default
0    644
1    156
Name: count, dtype: int64

Testing Set Counts:
loan_default
0    161
1     39
Name: count, dtype: int64
```

Number of data objects in the training dataset:

% Proportion of data objects in the training dataset:

<add rows to table as needed>

Class label	Number of data objects in the training dataset	% Proportion of data objects in the training dataset
0	644	80.5%
1	156	19.5%

Number of data objects in the test dataset:

% Proportion of data objects in the test dataset:

<add rows to table as needed>

Class label	Number of data objects in the test dataset	% Proportion of data objects in the test dataset
0	161	80.5%
1	39	19.5%

Experiments with artificial neural network

<add rows to table as needed>

Experiment	Hyperparameter values
Experiment 1	hidden_layer_sizes = (32, 16), Learning rate 0.01
Experiment 2	hidden_layer_sizes = (32, 16), learning rate 0.001
Experiment 3	hidden_layer_sizes = (32, 16), learning rate 0.00001

<a screenshot of hyperparameter values for Experiment 1>

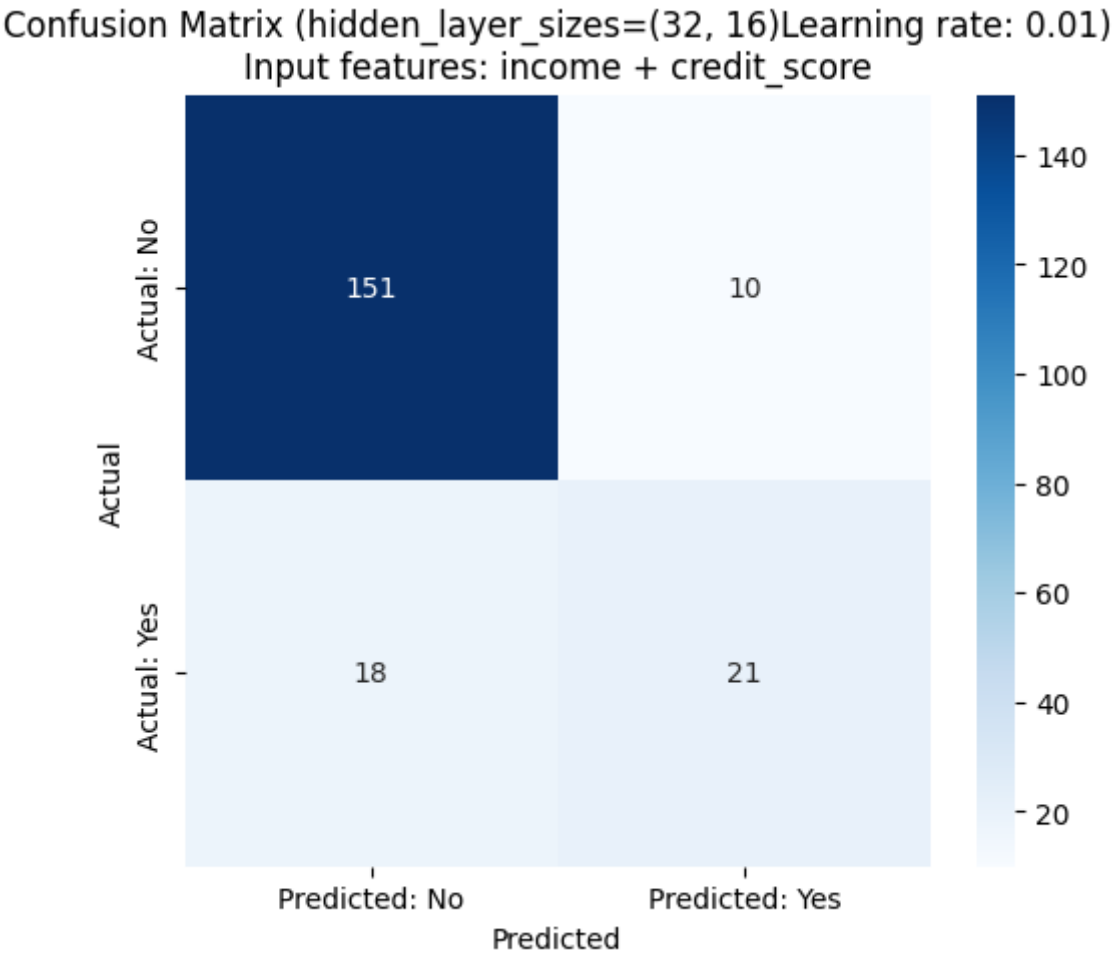
```
from sklearn.neural_network import MLPClassifier # type: ignore

# Initialize the MLPClassifier
ann = MLPClassifier(hidden_layer_sizes=(32, 16),  # Two hidden layers with 32 and 16 neurons
                    activation='relu',            # Activation function for hidden layers
                    solver='adam',                # Optimizer
                    learning_rate_init=0.01,          # Learning Rate
                    max_iter=1000,                # Maximum number of iterations
                    random_state=42)

# Train the neural network
ann.fit(X_train_scaled, y_train)
```

✓ 0.0s

```
MLPClassifier(hidden_layer_sizes=(32, 16), learning_rate_init=0.01,
              max_iter=1000, random_state=42)
```

<a screenshot of performance metrics for Experiment 1>

```
Accuracy: 0.86

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.94      0.92       161
           1       0.68      0.54      0.60        39

    accuracy                           0.86       200
   macro avg       0.79      0.74      0.76       200
weighted avg       0.85      0.86      0.85       200


Confusion Matrix:
 [[151  10]
 [ 18  21]]

ROC-AUC Score: 0.9058767319636885
```



<a screenshot of hyperparameter values for Experiment 2>

```python
from sklearn.neural_network import MLPClassifier

# Initialize the MLPClassifier
ann = MLPClassifier(hidden_layer_sizes=(32, 16),  # Two hidden layers with 32 and 16 neurons
                    activation='relu',            # Activation function for hidden layers
                    solver='adam',                # Optimizer
                    learning_rate_init=0.001,         # Learning Rate
                    max_iter=1000,                # Maximum number of iterations
                    random_state=42)

# Train the neural network
ann.fit(X_train_scaled, y_train)
```

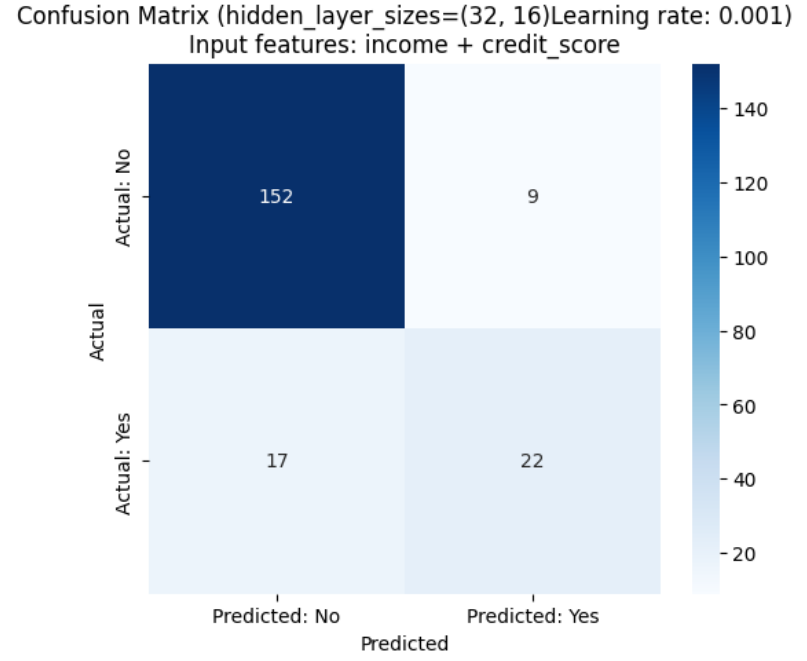
✓ 0.1s

MLPClassifier

```
MLPClassifier(hidden_layer_sizes=(32, 16), max_iter=1000, random_state=42)
```

<a screenshot of performance metrics for Experiment 2>

```
Accuracy: 0.87

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.94      0.92       161
           1       0.71      0.56      0.63        39

    accuracy                           0.87       200
   macro avg       0.80      0.75      0.77       200
weighted avg       0.86      0.87      0.86       200


Confusion Matrix:
 [[152   9]
 [ 17  22]]

ROC-AUC Score: 0.9065137760789934
```



<a screenshot of hyperparameter values for Experiment 3>

```python
from sklearn.neural_network import MLPClassifier # type: ignore

# Initialize the MLPClassifier
ann = MLPClassifier(hidden_layer_sizes=(32, 16),  # Two hidden layers with 32 and 16 neurons
                    activation='relu',            # Activation function for hidden layers
                    solver='adam',                # Optimizer
                    learning_rate_init=0.00001,         # Learning Rate
                    max_iter=1000,                # Maximum number of iterations
                    random_state=42)

# Train the neural network
ann.fit(X_train_scaled, y_train)
```

✓ 1.4s

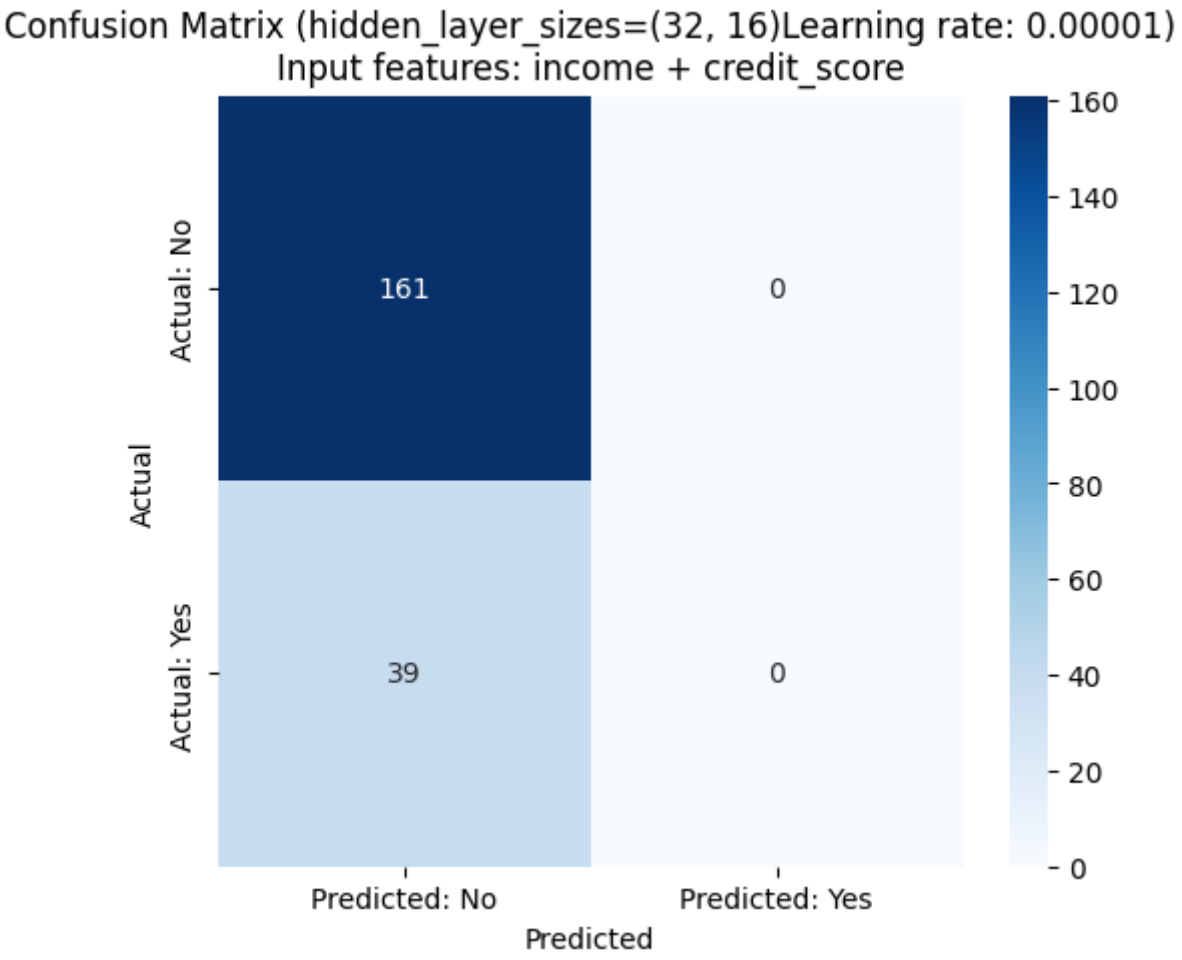
```
c:\SchoolWork\AI_For_Humanities_Second_Assignment\AI_Assignment_2_Loan_Default_Prediction\.venv\
  warnings.warn(
```

MLPClassifier

```
MLPClassifier(hidden_layer_sizes=(32, 16), learning_rate_init=1e-05,
              max_iter=1000, random_state=42)
```

<a screenshot of performance metrics for Experiment 3>

```
Accuracy: 0.805

Classification Report:
              precision    recall  f1-score   support

           0       0.81      1.00      0.89       161
           1       0.00      0.00      0.00        39

    accuracy                           0.81       200
   macro avg       0.40      0.50      0.45       200
weighted avg       0.65      0.81      0.72       200


Confusion Matrix:
 [[161   0]
 [ 39   0]]

ROC-AUC Score: 0.8913839783405
```



Conclusions from experiments:

<conclusions about the performance of the models in the conducted experiments referring to and analysing the screenshots above>

I have also tried with different input features, numbers of layers and neurons in each layer. You can see the results from my GitHub.

You can see that:
1. Reducing the number of neurons would reduce the accuracy.
2. Increasing the number of neurons would increase the accuracy in certain range, until it hits diminishing marginal utility.
(At least that is what we can see from the test above. But in reality, whether it would cause overfitting or not would be a question).
3. Increasing the number of neurons would increase training time.
4. Adding layers would increase the accuracy, at least that is what I found. But I assume it would also increase the noise captured.

Model selected for testing:

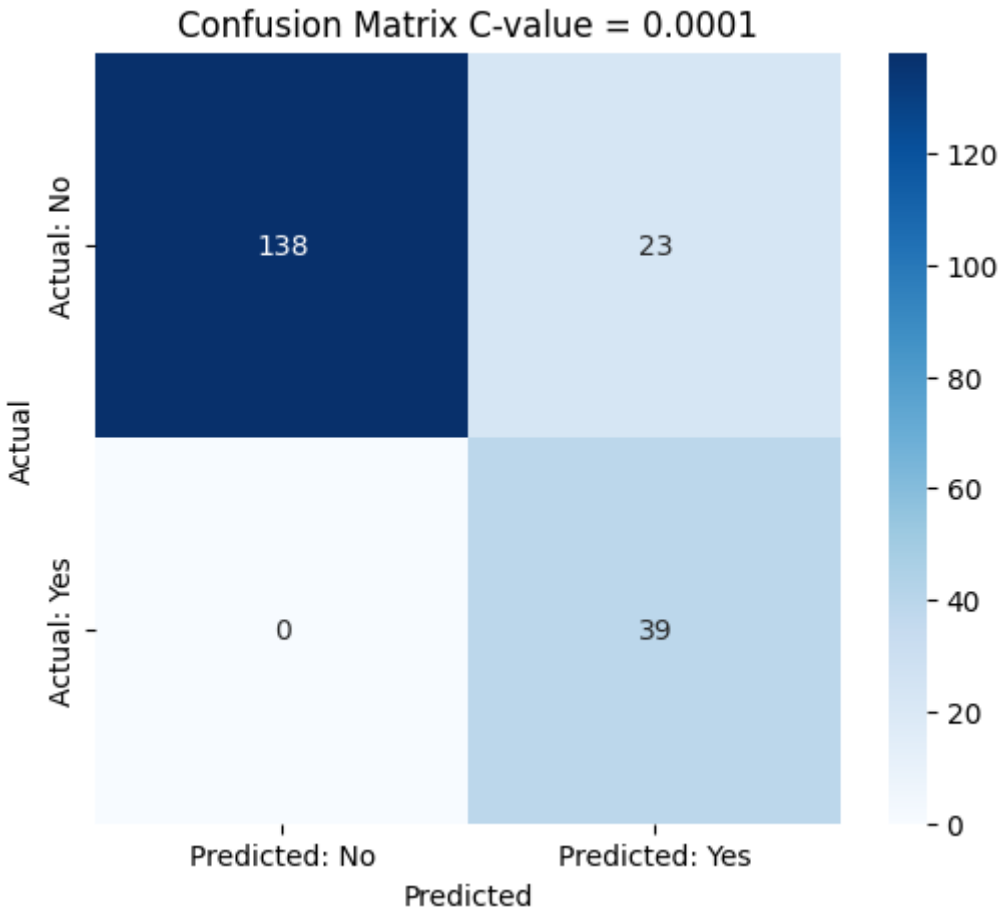
<indication of which experimental model is selected for the testing process>

Experiments with Logistic Regression

<add rows to table as needed>

Experiment	Hyperparameter values
Experiment 1	C = 0.0001
Experiment 2	C = 0.01
Experiment 3	

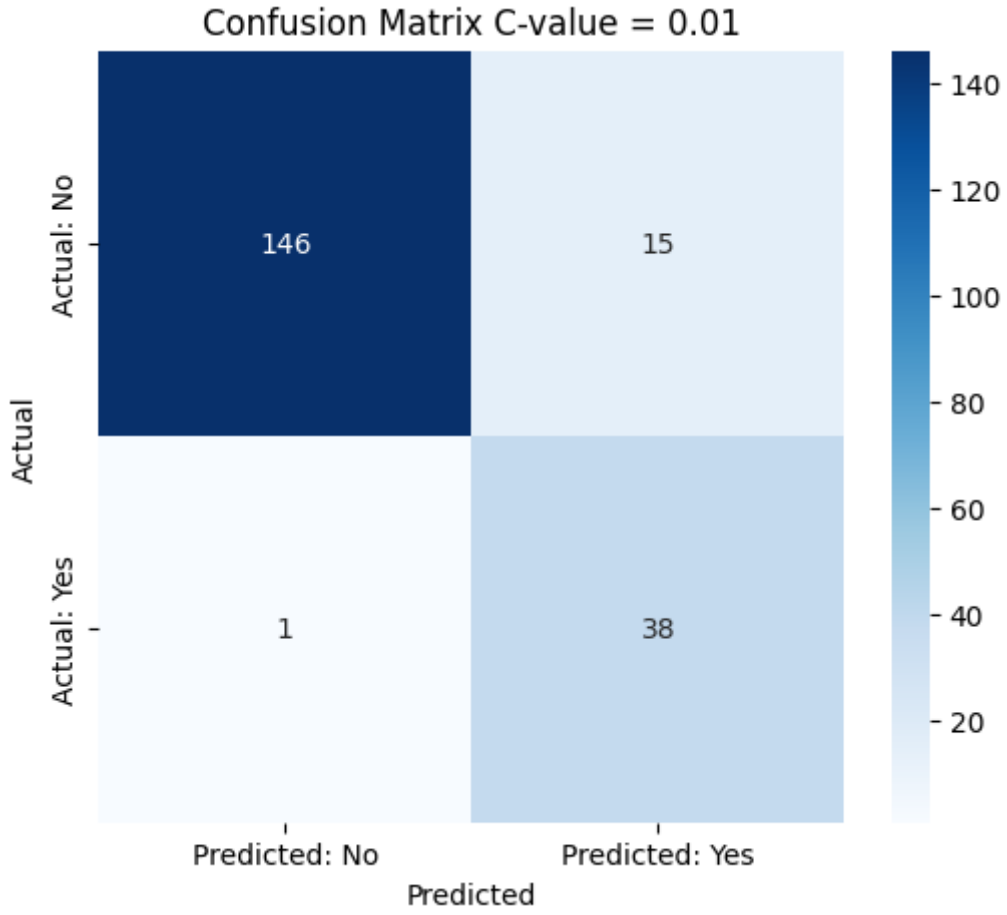
<a screenshot of hyperparameter values for Experiment 1>



<a screenshot of performance metrics for Experiment 1>

```
Accuracy: 0.885

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.86      0.92       161
           1       0.63      1.00      0.77        39

    accuracy                           0.89       200
   macro avg       0.81      0.93      0.85       200
weighted avg       0.93      0.89      0.89       200


Confusion Matrix:
 [[138  23]
 [  0  39]]

ROC-AUC Score: 0.9855072463768116
```

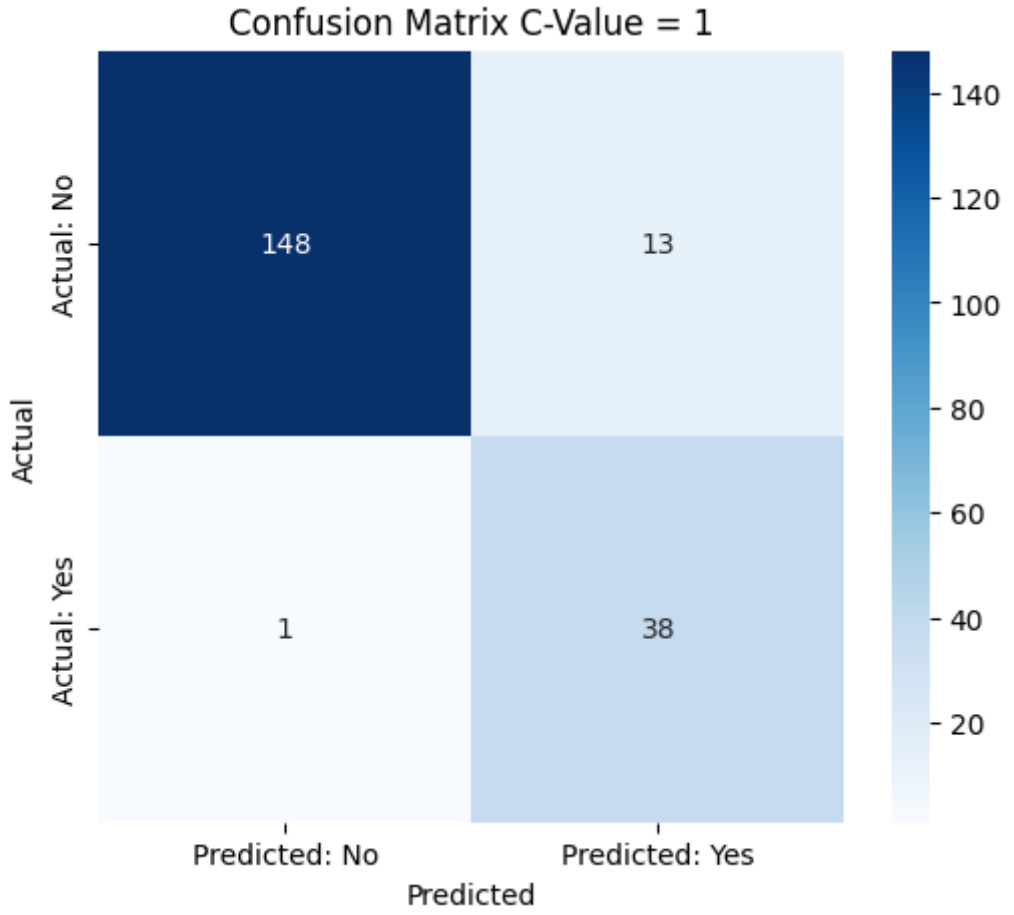
<a screenshot of hyperparameter values for Experiment 2>



<a screenshot of performance metrics for Experiment 2>

```
Tuned Logistic Regression Results
Accuracy: 0.92

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.91      0.95       161
           1       0.72      0.97      0.83        39

    accuracy                           0.92       200
   macro avg       0.86      0.94      0.89       200
weighted avg       0.94      0.92      0.92       200


Confusion Matrix:
 [[146  15]
 [  1  38]]

ROC-AUC Score: 0.9847109412326804
```

<a screenshot of hyperparameter values for Experiment 3>



<a screenshot of performance metrics for Experiment 3>

```
Tuned Logistic Regression Results
Accuracy: 0.93

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.92      0.95       161
           1       0.75      0.97      0.84        39

    accuracy                           0.93       200
   macro avg       0.87      0.95      0.90       200
weighted avg       0.94      0.93      0.93       200


Confusion Matrix:
 [[148  13]
 [  1  38]]

ROC-AUC Score: 0.9858257684344641
```

Conclusions from experiments:

<conclusions about the performance of the models in the conducted experiments referring to and analysing the screenshots above>

From the above we can see that:

1. increasing the C-value would increase the accuracy first, until it hits the optimal point.
2. when over the optimal point, and the accuracy would reduce again

Model selected for testing:

<indication of which experimental model is selected for the testing process>

Experiments with Decision Tree

<add rows to table as needed>

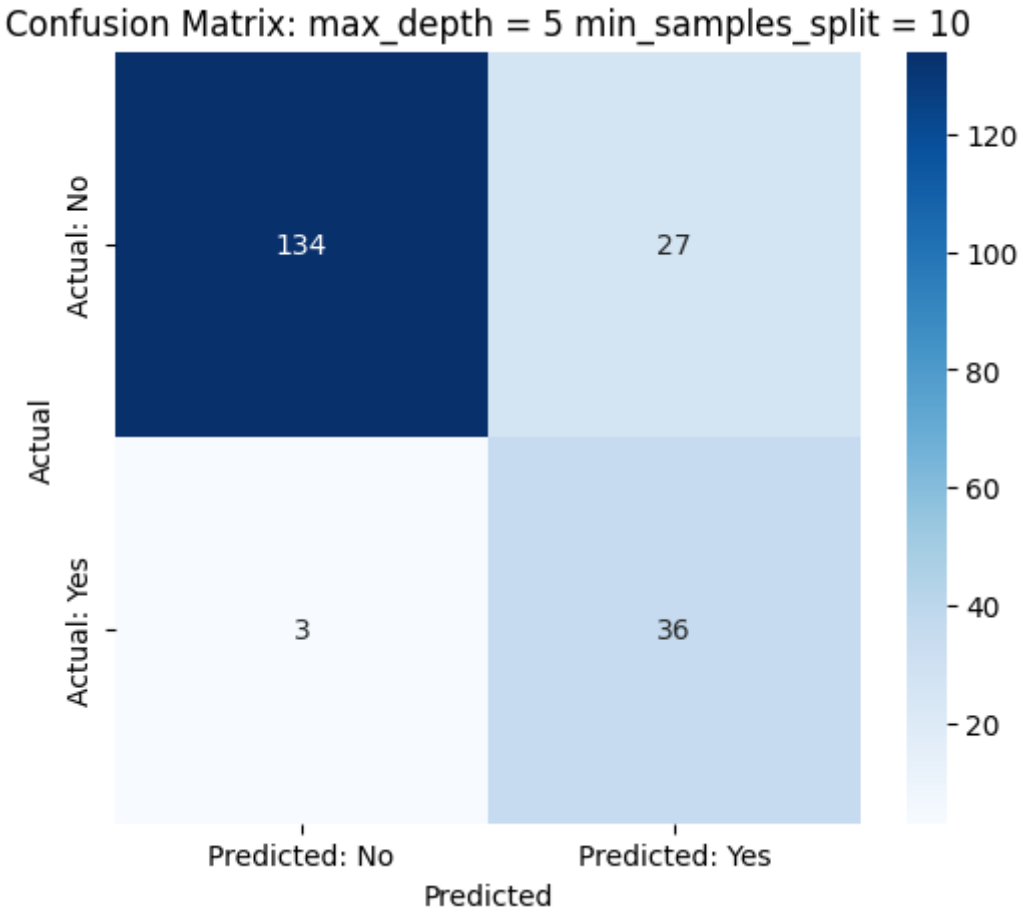
Experiment	Hyperparameter values
Experiment 1	Max depth = 5, min samples split = 10
Experiment 2	Max depth = 10, min samples split = 10
Experiment 3	Max depth = 10, min samples split = 40

<a screenshot of hyperparameter values for Experiment 1>

```python
# 3. Adjusting the hyperparameters
max_depth = 5  # Limiting the depth of the tree to avoid overfitting
min_samples_split = 10  # Minimum samples required to split a node
class_weight = 'balanced'  # Automatically adjust class weights to handle class imbalance
```

<a screenshot of performance metrics for Experiment 1>

```
Accuracy: 0.85

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.83      0.90       161
           1       0.57      0.92      0.71        39

    accuracy                           0.85       200
   macro avg       0.77      0.88      0.80       200
weighted avg       0.90      0.85      0.86       200


Confusion Matrix:
 [[134  27]
 [  3  36]]

ROC-AUC Score: 0.932712215320911
```

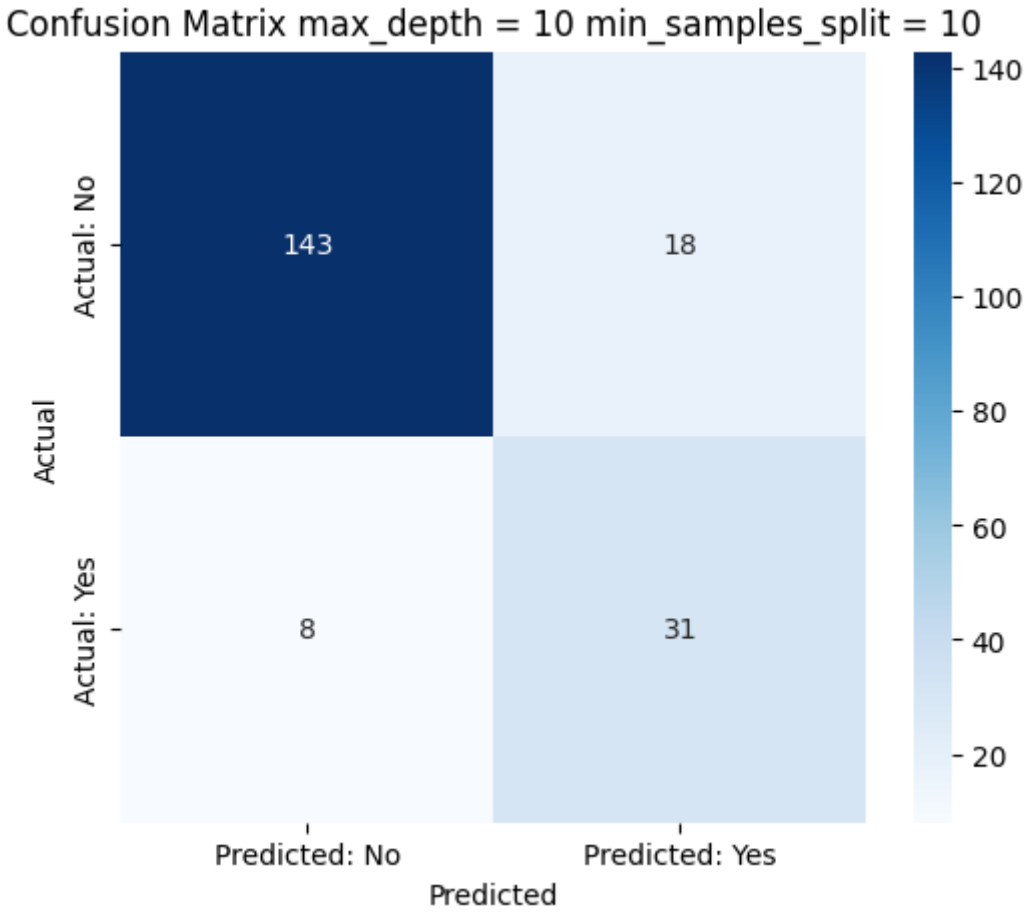


<a screenshot of hyperparameter values for Experiment 2>

```python
# 3. Adjusting the hyperparameters
max_depth = 10  # Limiting the depth of the tree to avoid overfitting
min_samples_split = 10  # Minimum samples required to split a node
class_weight = 'balanced'  # Automatically adjust class weights to handle class imbalance
```

<a screenshot of performance metrics for Experiment 2>

```
Accuracy: 0.87

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.89      0.92       161
           1       0.63      0.79      0.70        39

    accuracy                           0.87       200
   macro avg       0.79      0.84      0.81       200
weighted avg       0.89      0.87      0.88       200


Confusion Matrix:
 [[143  18]
 [  8  31]]

ROC-AUC Score: 0.8739448956840261
```

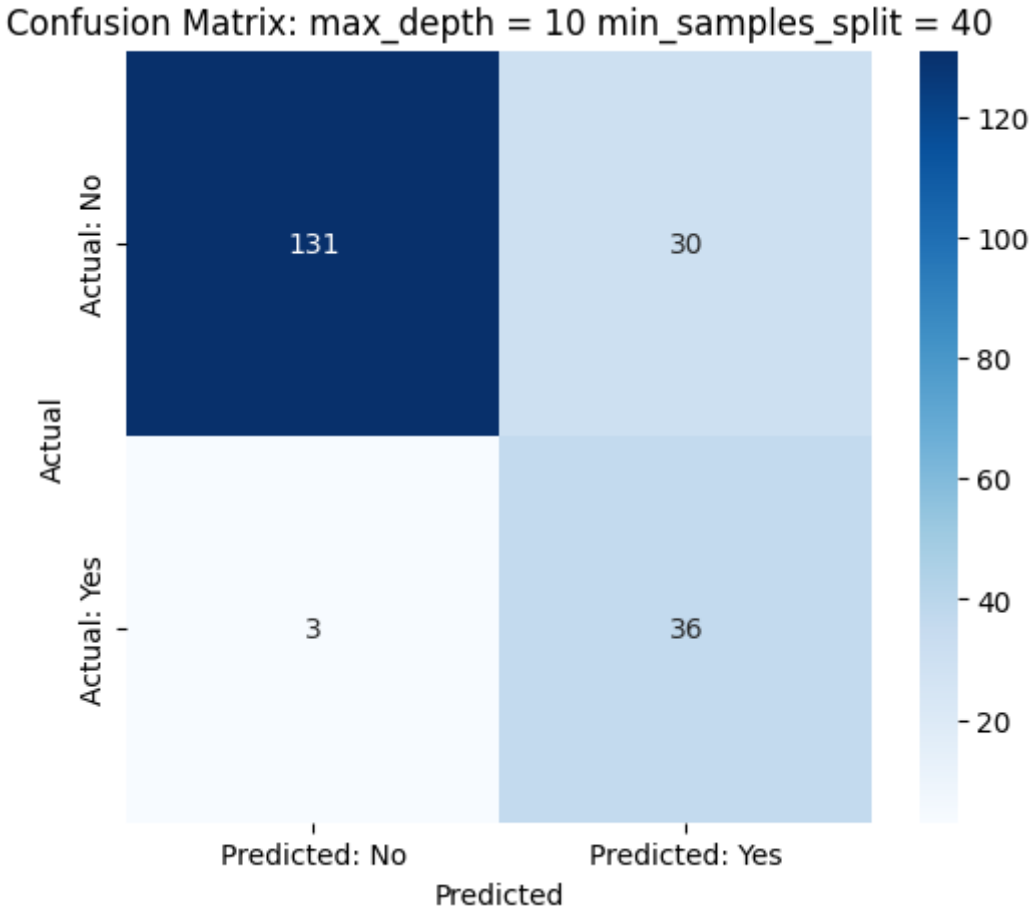


<a screenshot of hyperparameter values for Experiment 3>

```python
# 3. Adjusting the hyperparameters
max_depth = 10  # Limiting the depth of the tree to avoid overfitting
min_samples_split = 40  # Minimum samples required to split a node
class_weight = 'balanced'  # Automatically adjust class weights to handle class imbalance
```

<a screenshot of performance metrics for Experiment 3>

```
Accuracy: 0.835

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.81      0.89       161
           1       0.55      0.92      0.69        39

    accuracy                           0.83       200
   macro avg       0.76      0.87      0.79       200
weighted avg       0.89      0.83      0.85       200


Confusion Matrix:
 [[131  30]
 [  3  36]]

ROC-AUC Score: 0.9306418219461697
```



Conclusions from experiments:

<conclusions about the performance of the models in the conducted experiments referring to and analysing the screenshots above >

Model selected for testing:

<indication of which experimental model is selected for the testing process>

Testing results of the trained models

<a screenshot of performance metrics of models selected for testing>

It seems that I didn't separate the original data for final testing, therefore, if I use the existing data for testing, it would produce more or less the same results.

But since the whole report is done, if I separate the testing data for this part from the original data, and use the remaining for training and validation, it would require me to rewrite the whole report, which would lead to late submission.

I guess I'll have to think of some other ways to finish this part.

Meanwhile, I am looking for different datasets with the same features. And when I find one, I'll conduct this part.

Conclusions after testing:

<comparison of performance of the trained models in the testing process referring to and analysing the screenshot above >

Information sources

Project link: https://github.com/Grace128pan/AI_Assignment_2_Loan_Default_Prediction

Link to the dataset: https://www.kaggle.com/datasets/msfasha/loan-default-prediction