Machine Learning

Course Code: MEAD-656

Lab Practical File

Submitted by

Shantanu Shukla

01211805424

in partial fulfillment for the award of the degree

of

Master of Technology (AI & DS)

batch of 2024 - 26 (2nd Semester)

Center for Development of Advanced Computing, Noida

Affiliated to Guru Gobind Singh Indraprastha University

INDEX

<u>Exercise-1</u>

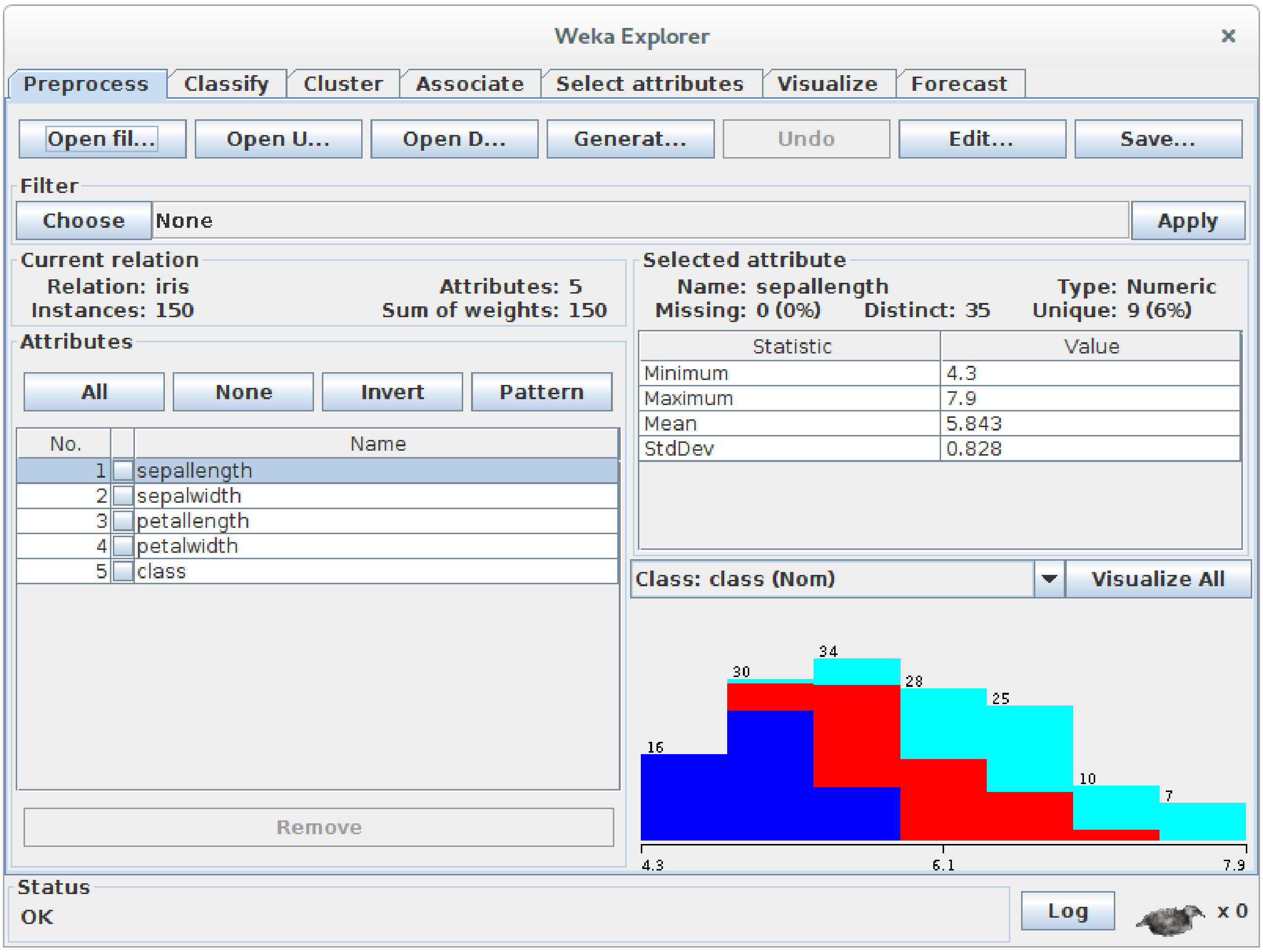
<u>Objective:</u> Decision Tree I.

Step1. Open the data/iris.arff Dataset

Click the "***Open file…***" button to open a data set and double click on the "***data***" directory.

Weka provides a number of small common machine learning datasets that you can use to practice on.

Select the "***iris.arff***" file to load the Iris dataset.



The Iris flower dataset is a famous dataset from statistics and is heavily borrowed by researchers in machine learning. It contains 150 instances (rows) and 4 attributes (columns) and a class attribute for the species of iris flower (one of setosa, versicolor, virginica).

Step2. Select and Run an Algorithm

Now that you have loaded a dataset, it's time to choose a machine learning algorithm to model the problem and make predictions.

Click the "***Classify***" tab. This is the area for running algorithms against a loaded dataset in Weka. You will note that the “***ZeroR***” algorithm is selected by default.

The ZeroR algorithm selects the majority class in the dataset (all three species of iris are equally present in the data, so it picks the first one: setosa) and uses that to make all predictions. This is the baseline for the dataset and the measure by which all algorithms can be compared. The result is **33%**, as expected (3 classes, each equally represented, assigning one of the three to each prediction results in 33% classification accuracy).

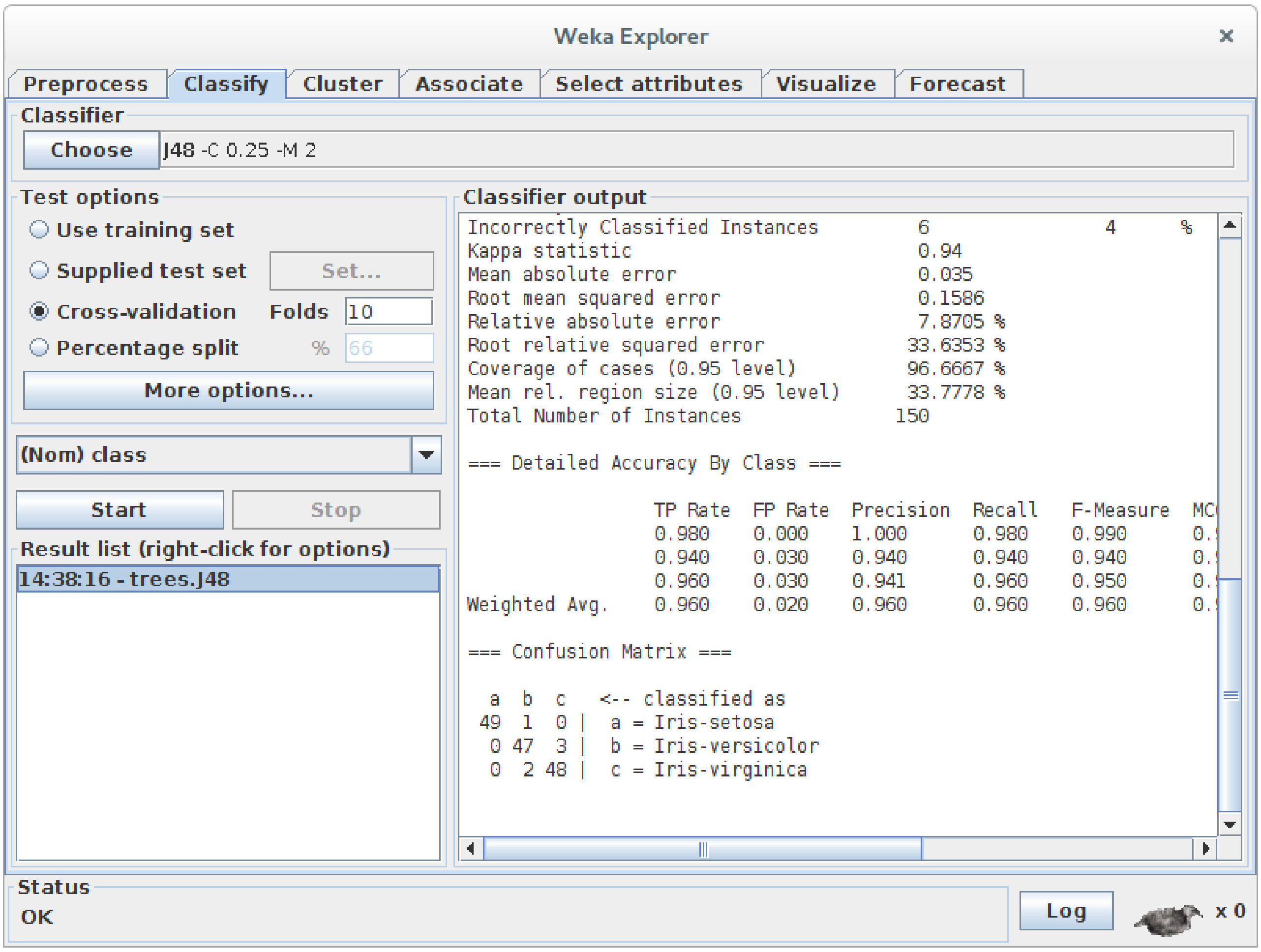
You will also note that the test options uses Cross Validation by default with 10 folds. This means that the dataset is split into 10 parts, the first 9 are used to train the algorithm, and the 10th is used to assess the algorithm. This process is repeated allowing each of the 10 parts of the split dataset a chance to be the held out test set.

Step3: Choose other Algorithm

Click the “***Choose***” button in the “Classifier” section and click on “***trees***” and click on the “***J48***” algorithm.

This is an implementation of the C4.8 algorithm in Java (“J” for Java, 48 for C4.8, hence the J48 name) and is a minor extension to the famous C4.5 algorithm.

Click the “***Start***” button to run the algorithm.



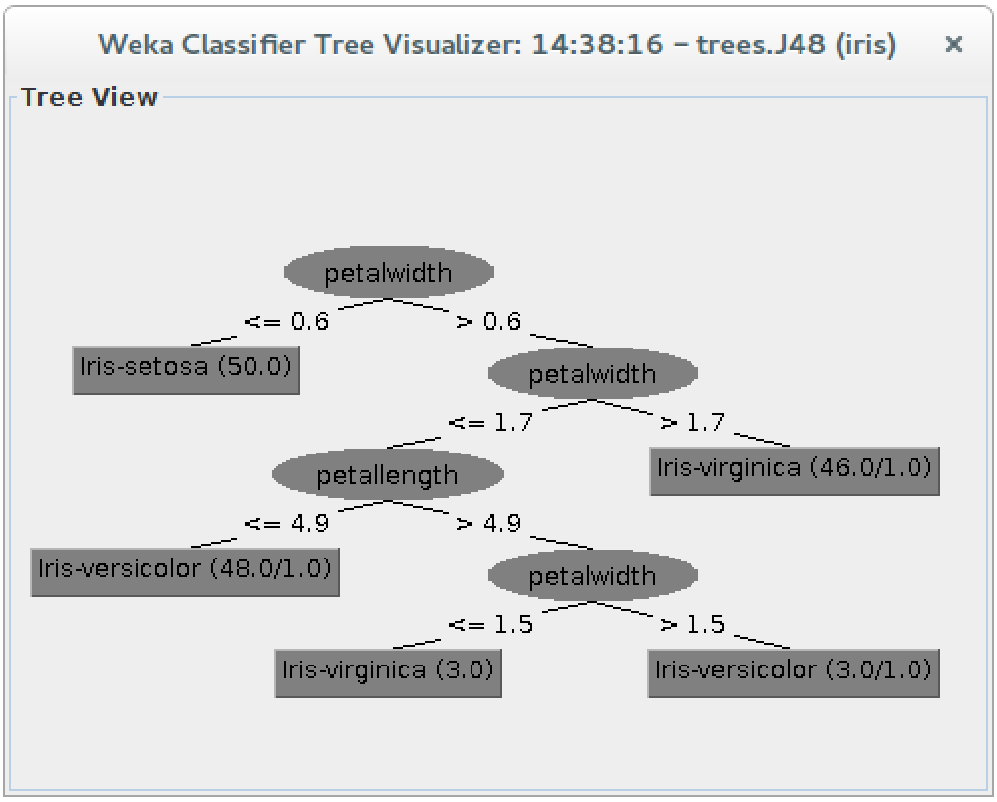
Step4: Review Results

After running the J48 algorithm, you can note the results in the “Classifier output” section.

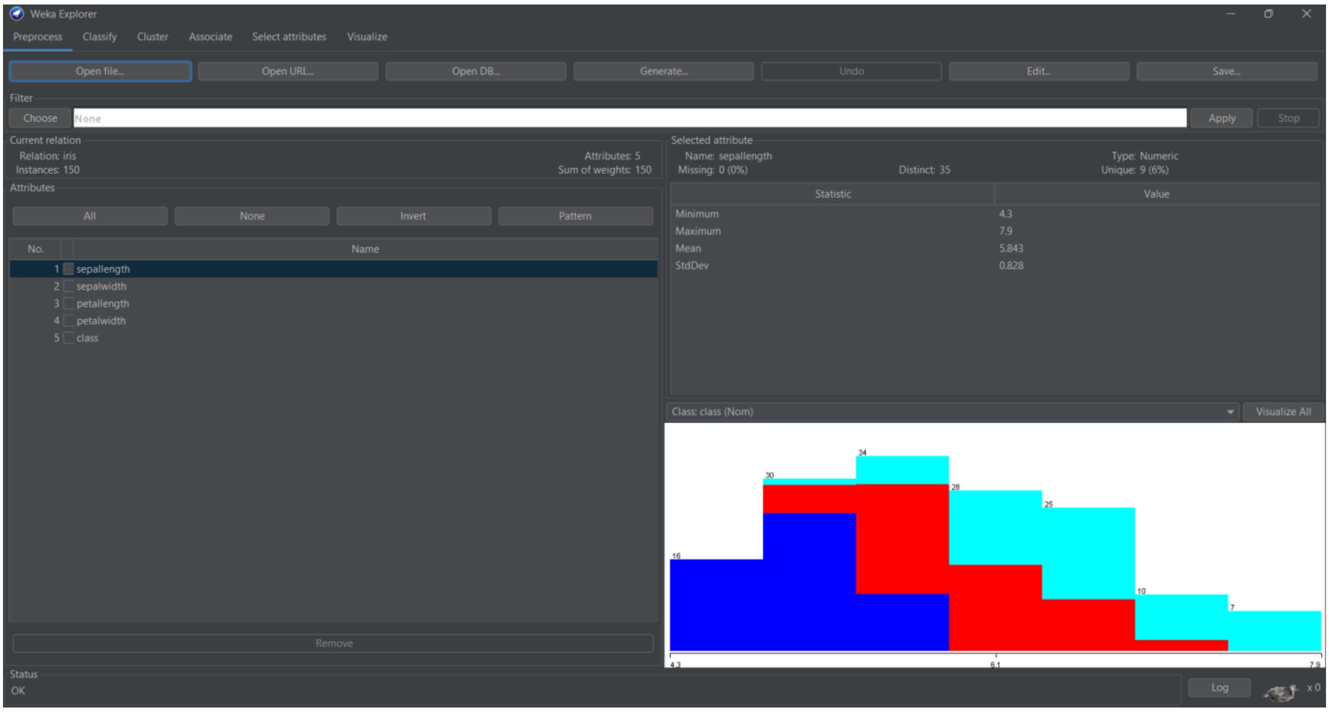
The algorithm was run with 10 fold cross validation, this means it was given an opportunity to make a prediction for each instance of the dataset (with different training folds) and the presented result is a summary of those predictions.

Firstly, note the Classification Accuracy. You can see that the model achieved a result of **144/150** correct or **96%**, which seems a lot better than the baseline of **33%**.

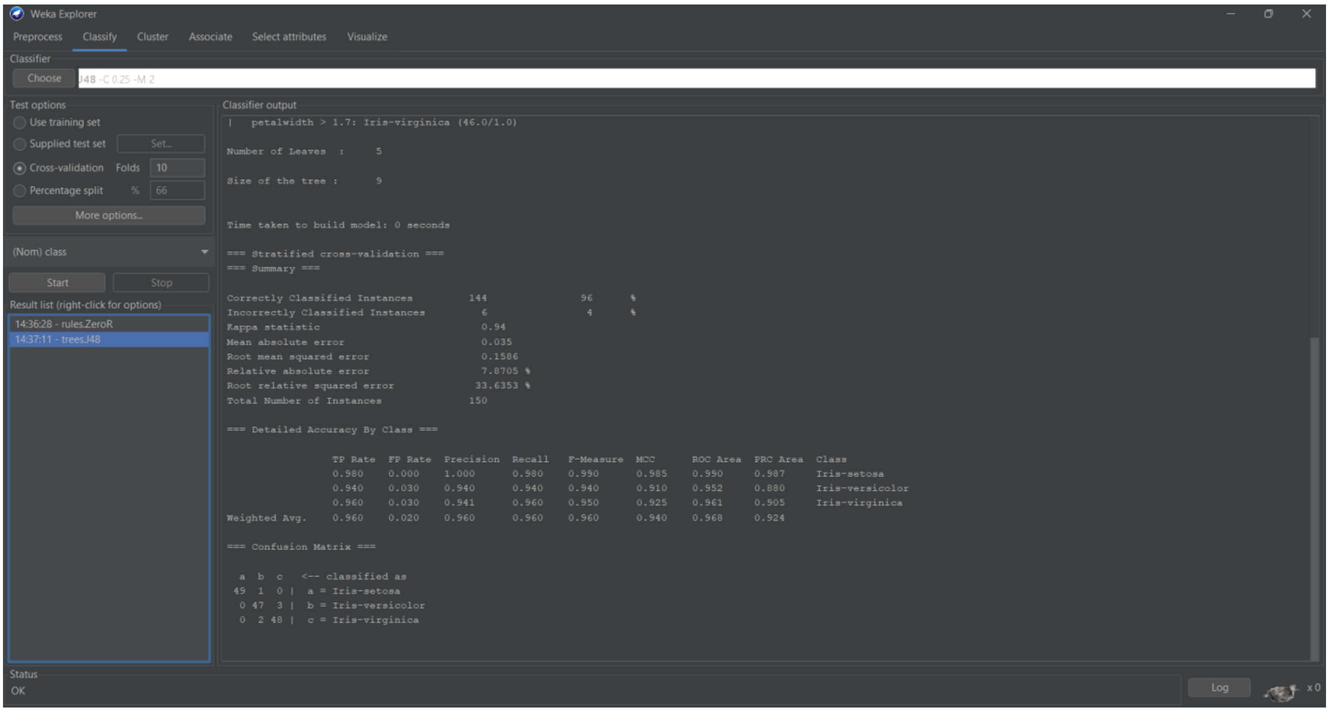
Secondly, look at the Confusion Matrix. You can see a table of actual classes compared predicted classes and you can see that was 1 error where a Iris-setosa was classified as a Iris-versicolor, 2 cases where Iris-virginica was classified as a Iris-versicolor and 3 cases where a Iris-versicolor was classified as a Iris-setosa (a total of 6 errors). This table can help to explain the accuracy achieved by the algorithm.
You can see the output given by J48 algorithm in a tree fashion by right click on the Results List and by choosing the option Visualize Tree.

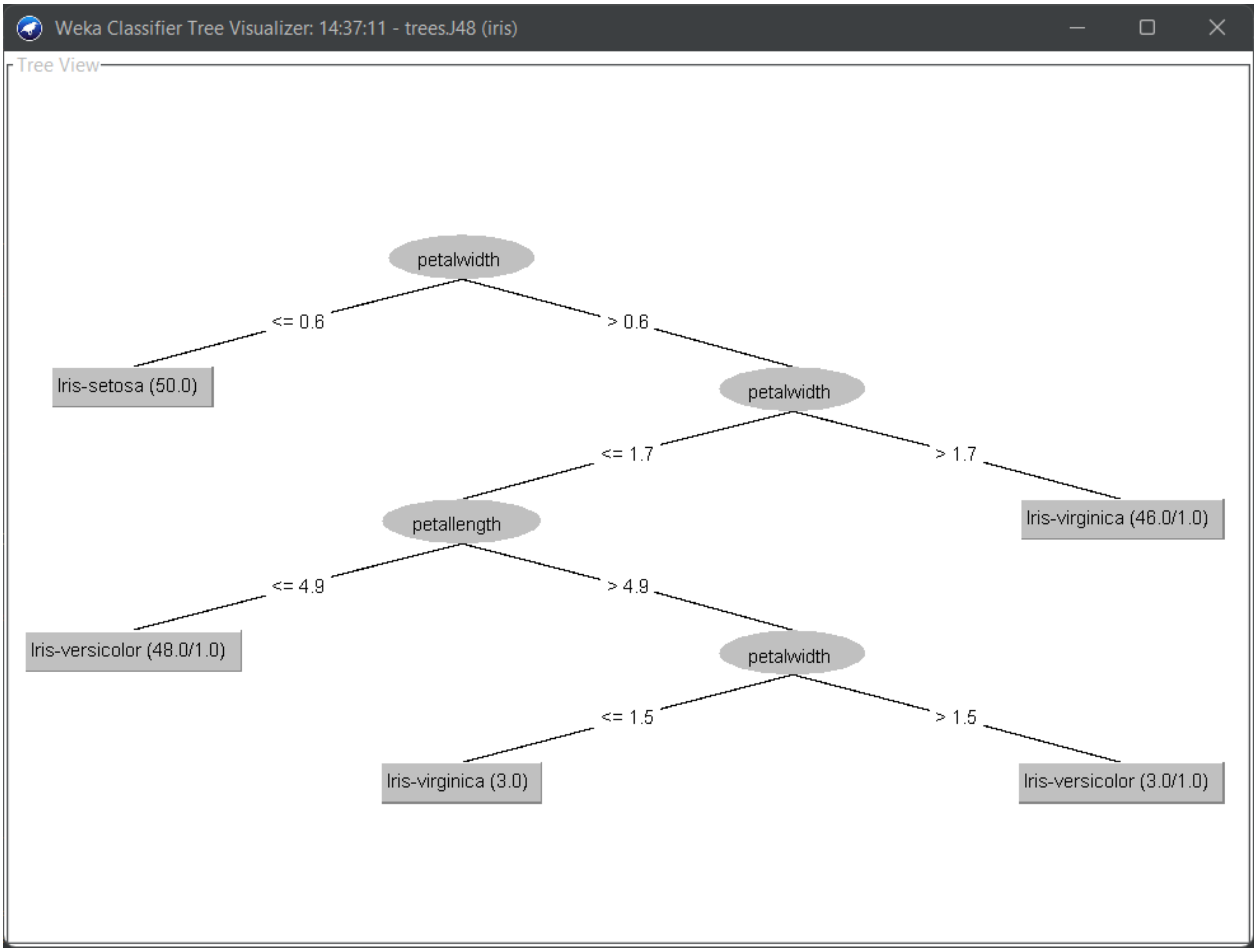


Results:


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
None
Apply
Stop
Current relation
Relation: iris
Instances: 150
Attributes: 5
Sum of weights: 150
Attributes
All
None
Invert
Pattern
No.
Name
1 sepallength
2 sepalwidth
3 petallength
4 petalwidth
5 class
Remove
Selected attribute
Name: sepallength
Missing: 0 (0%)
Distinct: 35
Type: Numeric
Unique: 9 (6%)
Statistic
Value
Minimum 4.3
Maximum 7.9
Mean 5.843
StdDev 0.828
Class: class (Nom)
Visualize All
34
30
28
25
16
10
7
4.3
6.1
7.9
Status
OK
Log
x 0


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
ZeroR
Test options
Use training set
Supplied test set
Set...
Cross-validation Folds 10
Percentage split % 66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
14:36:28 - rules.ZeroR
Classifier output
class
Test mode: 10-fold cross-validation
=== Classifier model (full training set) ===
ZeroR predicts class value: Iris-setosa
Time taken to build model: 0 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 50 33.3333 %
Incorrectly Classified Instances 100 66.6667 %
Kappa statistic 0
Mean absolute error 0.4444
Root mean squared error 0.4714
Relative absolute error 100 %
Root relative squared error 100 %
Total Number of Instances 150
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
1.000 1.000 0.333 1.000 0.500 ? 0.500 0.333 Iris-setosa
0.000 0.000 ? 0.000 ? ? 0.500 0.333 Iris-versicolor
0.000 0.000 ? 0.000 ? ? 0.500 0.333 Iris-virginica
Weighted Avg. 0.333 0.333 ? 0.333 ? ? 0.500 0.333
=== Confusion Matrix ===
a b c <-- classified as
50 0 0 | a = Iris-setosa
50 0 0 | b = Iris-versicolor
50 0 0 | c = Iris-virginica
Status
OK
Log
x 0


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
J48 -C 0.25 -M 2
Test options
Use training set
Supplied test set
Set...
Cross-validation Folds 10
Percentage split % 66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
14:36:28 - rules.ZeroR
14:37:11 - trees.J48
Classifier output
| petalwidth > 1.7: Iris-virginica (46.0/1.0)
Number of Leaves : 5
Size of the tree : 9
Time taken to build model: 0 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 144 96 %
Incorrectly Classified Instances 6 4 %
Kappa statistic 0.94
Mean absolute error 0.035
Root mean squared error 0.1586
Relative absolute error 7.8705 %
Root relative squared error 33.6353 %
Total Number of Instances 150
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
0.980 0.000 1.000 0.980 0.990 0.985 0.990 0.987 Iris-setosa
0.940 0.030 0.940 0.940 0.940 0.910 0.952 0.880 Iris-versicolor
0.960 0.030 0.941 0.960 0.950 0.925 0.961 0.905 Iris-virginica
Weighted Avg. 0.960 0.020 0.960 0.960 0.960 0.940 0.968 0.924
=== Confusion Matrix ===
a b c <-- classified as
49 1 0 | a = Iris-setosa
0 47 3 | b = Iris-versicolor
0 2 48 | c = Iris-virginica
Status
OK
Log
x 0


Weka Classifier Tree Visualizer: 14:37:11 - trees.J48 (iris)
Tree View
petalwidth
<= 0.6
> 0.6
Iris-setosa (50.0)
petalwidth
<= 1.7
> 1.7
petallength
Iris-virginica (46.0/1.0)
<= 4.9
> 4.9
Iris-versicolor (48.0/1.0)
petalwidth
<= 1.5
> 1.5
Iris-virginica (3.0)
Iris-versicolor (3.0/1.0)

<u>Exercise-2</u>

<u>Objective:</u> Decision Tree II.

In this assignment we use a data set of animals and their attributes. Using a decision tree classifier the computer learns to classify animals into different categories (mammals, fish, reptiles etc). Use the **zoo.xls** data set

1. Study the animals in the Excel document (<u>zoo.xls</u>). Without using a data mining tool, draw a decision tree of three to five levels deep that classifies animals into a mammal, bird, reptile, fish, amphibian, insect or invertebrate.

2. Open the zoo.arff data set in WEKA and Find out:
   a. How many animals are there in the data set?
   b. How many attributes are known of each animal?

3. Let us build some classifiers. Go to the classifier tab. We will use 66% of the animals to build the models, and the remaining 34% to evaluate the quality of the model, so select **percentage split – 66%.** First we will build a 'naïve' model that just predicts the most occurring class in the data set for each animal. This corresponds to a decision tree of depth 0. Click start to build a model.
   a. What % of animals is correctly classified?
   b. Into what category are all these animals classified and why?

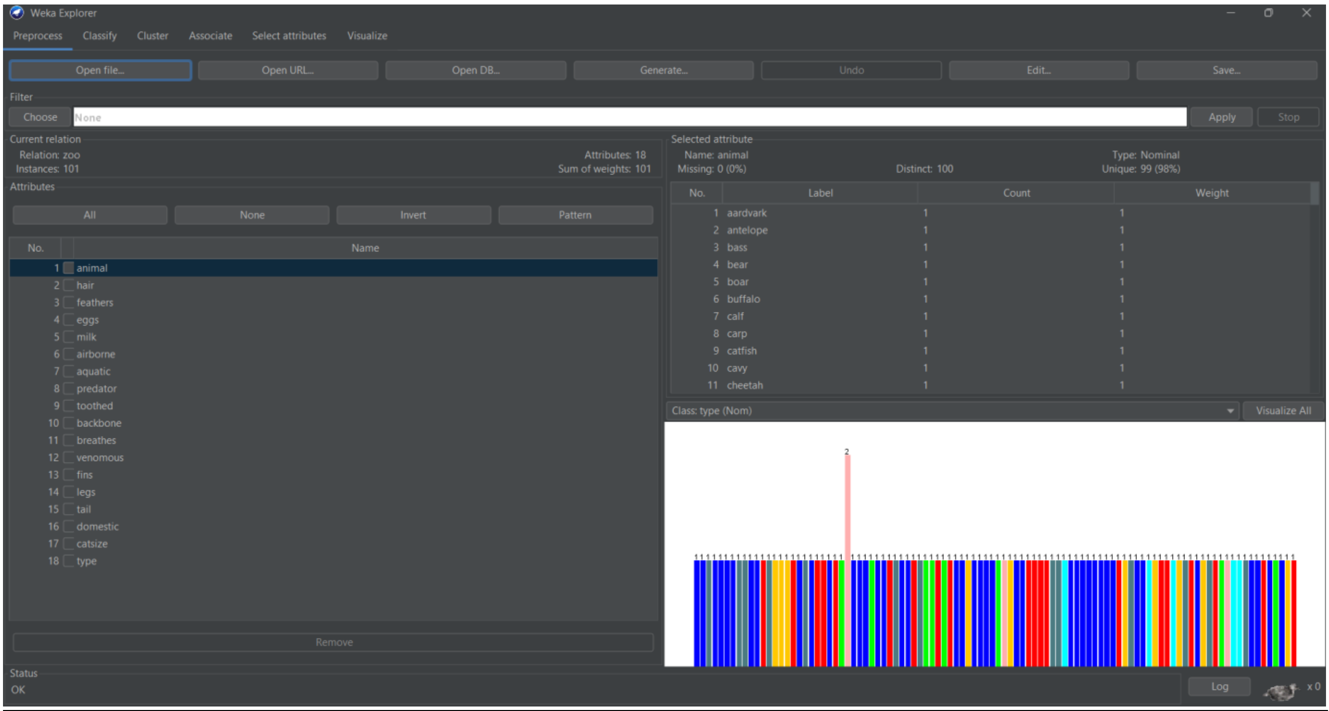
4. Now build a **decision tree of depth 1** (a.k.a. a decision stump - select choose – trees – **decision stump**).
   a. Draw the discovered decision tree.
   b. What % of animals is correctly classified?
   c. Give an example of an animal that would not be classified correctly by this model.

5. Now build a decision tree of **any depth** (a.k.a. a **J48 tree**).Go to the classifier tab and select the decision tree classifier j48. Click on the line behind the choose button. This shows you the parameters you can set and a button called 'More'.
   a. Which algorithm is implemented by j48?
   b. Draw the discovered decision tree.
   c. What percentage of instances is correctly classified by j48?
   d. Which families are mistaken for each other?

6. Again go to the parameter settings by clicking on the box after the 'Choose' button. Now change binarySplit to true and build a new decision tree.
    a. Draw the discovered decision tree.
    b. What is the difference?
    c. What % of animals is correctly classified?
    d. Give an example of an animal that would not be classified correctly by this model.

Results:

Dataset Overview:



Exercise-3

Objective: Data Preprocessing.

Step1. Open the data/bank-data.csv Dataset

Click the "***Open file...***" button to open a data set and double click on the "***data***" directory. Select the "bank-data.csv" file to load the bank dataset.

id	a unique identification number
age	age of customer in years (numeric)
sex	MALE / FEMALE
region	inner_city/rural/suburban/town
income	income of customer (numeric)
married	is the customer married (YES/NO)
children	number of children (numeric)
car	does the customer own a car (YES/NO)
save_acct	does the customer have a saving account (YES/NO)
current_acct	does the customer have a current account (YES/NO)
mortgage	does the customer have a mortgage (YES/NO)
pep	did the customer buy a PEP (Personal Equity Plan) after the last mailing (YES/NO)

Step2. Selecting or Filtering Attributes

In our sample data file, each record is uniquely identified by a customer id (the "id" attribute). We need to remove this attribute before the data mining step.

- We can do this by simply select the attribute and click on "Remove button"
- Using the Attribute filters in WEKA. In the "Filter" panel, click on the "Choose" button. This will show a popup window with a list available filters. Scroll down the list and select the "weka.filters.unsupervised.attribute.Remove" and click apply.
- Save the file with a name "bank-data-R1.arff"

Step3: Discretization

Some techniques, such as association rule mining, can only be performed on categorical data. This requires performing discretization on numeric or continuous attributes. There are 3 such attributes in this data set: "age", "income", and "children". In the case of the "children" attribute the range of possible values are only 0, 1, 2, and 3. In this case, we have opted for keeping all of these values in the data. This means we can simply discretize by removing the keyword "numeric" as the type for the "children" attribute in the ARFF file, and replacing it with the set of discrete values. We do this directly in our text editor. In this case, we have saved the resulting relation in a separate file "bank- data2.arff".

Load the bank-data2.arff dataset into weka.

If we select the "children" attribute in this new data set, we see that it is now a categorical attribute with four possible discrete values.

once again we choose the Filter dialog box, but this time, we will select Discretize from the list. Next, to change the defaults for this filter, click on the box immediately to the right of the "Choose" button. This will open the Discretize Filter dialog box. We enter the index for the attributes to be discretized. In this case we enter 1 corresponding to attribute "age". We also enter 3 as the number of bins (note that it is possible to discretize more than one attribute at the same time by using a list of attribute indices).

Save the file as bank-data3.arff and check the dataset in text editor.

Next, we apply the same process to discretize the "income" attribute into 3 bins. Again, Weka automatically performs the binning and replaces the values in the "income" column with the appropriate automatically generated labels. We save the new file into "bank-data3.arff", replacing the older version. Editing by text editor made bank-data-final.arff.

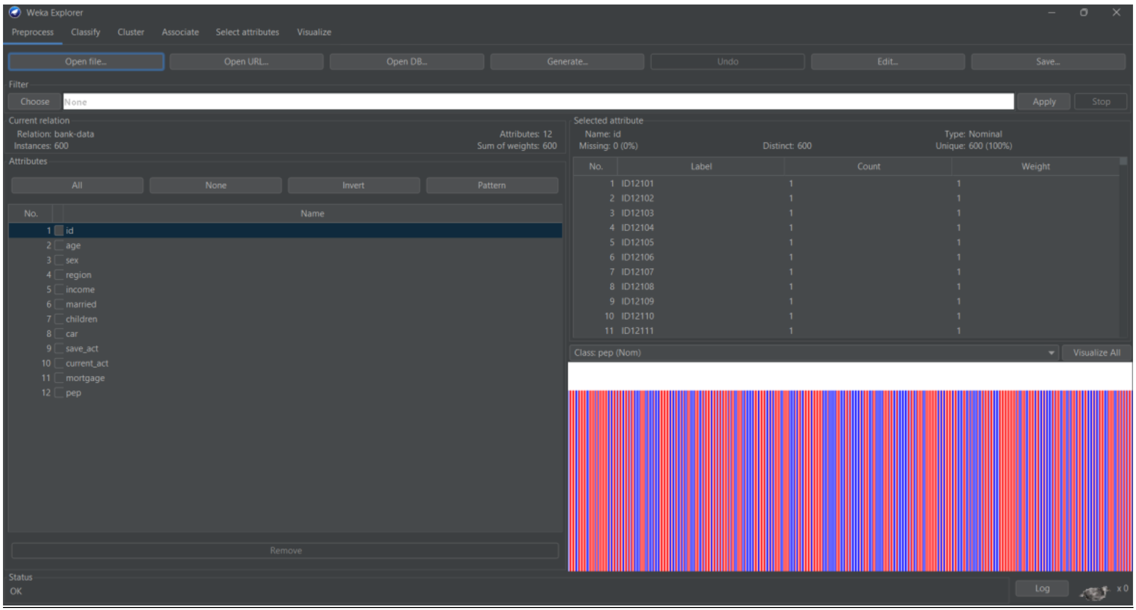
Step4: Missing Values

1. Open file "bank-data.arff"
2. Check if there is any missing values in any attribute.
3. Edit data to make some missing values.
4. Delete some data in "region"(Nominal) and "children"(Numeric) attributes. Click on "OK" button when finish.
5. Make note of Label that has Max Count in "region" and Mean of "children" attributes.
6. Choose "ReplaceMissingValues" filter

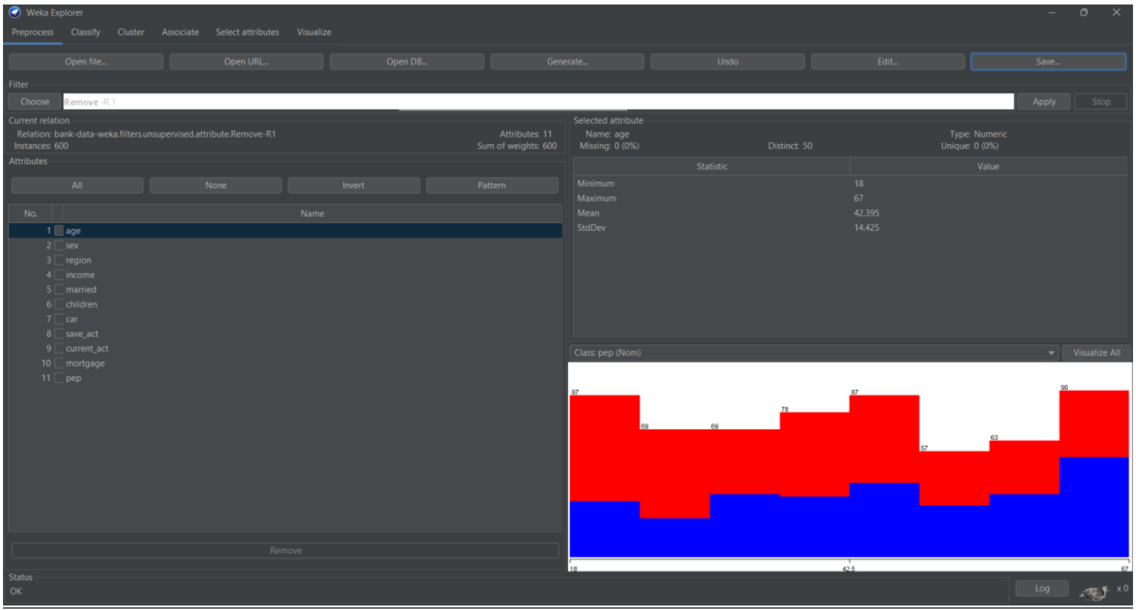
   (weka.filters.unsupervised.attribute.ReplaceMissingValues). Then, click on Apply button.
7. Look into the data. How did those missing values get replaced ?
8. Edit "bank-data.arff" with text editor. Make some data missing by replacing them with '?'. (Try with nominal and numeric attributes). Save to "bank-data-missing.arff".
9. Load "bank-data-missing.arff" into WEKA, observe the data and attribute information.
10. Replace missing values by the same procedure you had done before.

Results:

Step-1: Open and observe the “bank” dataset.

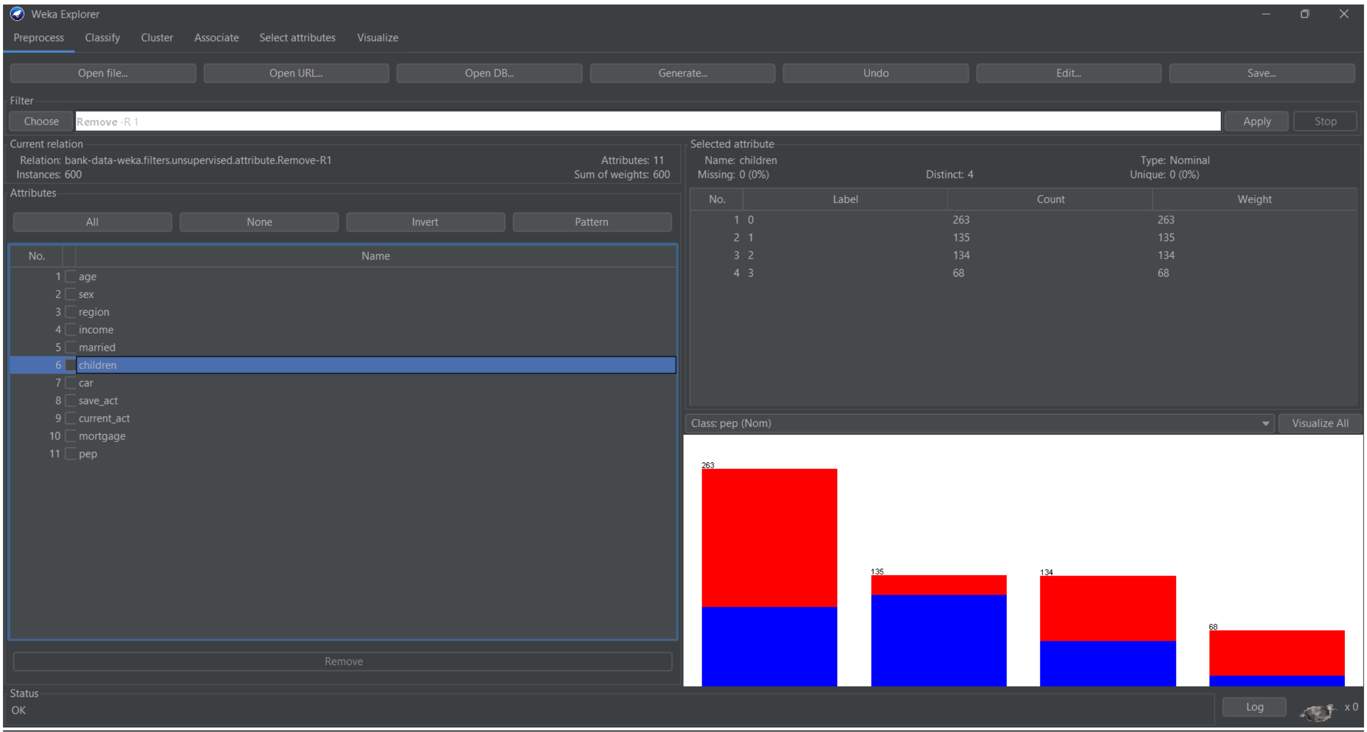


Step-2: Remove attribute “id

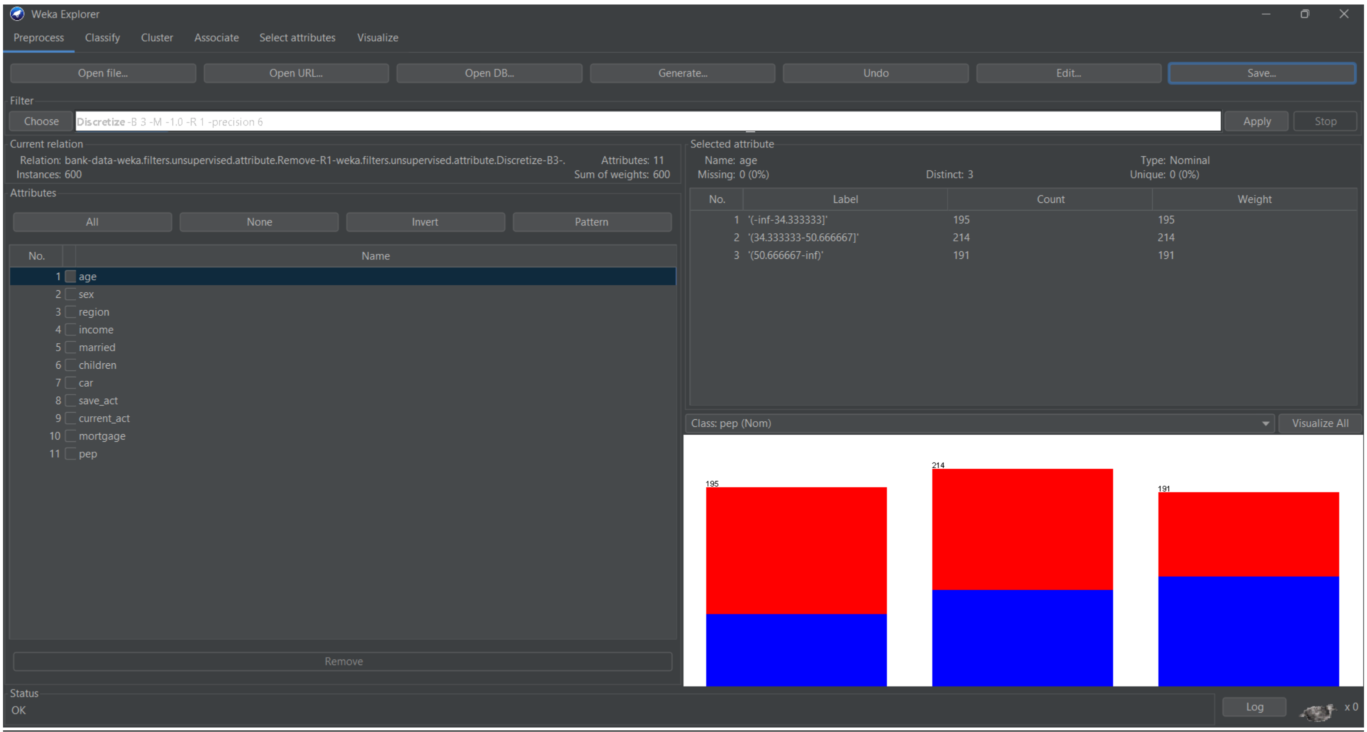


Step-3:

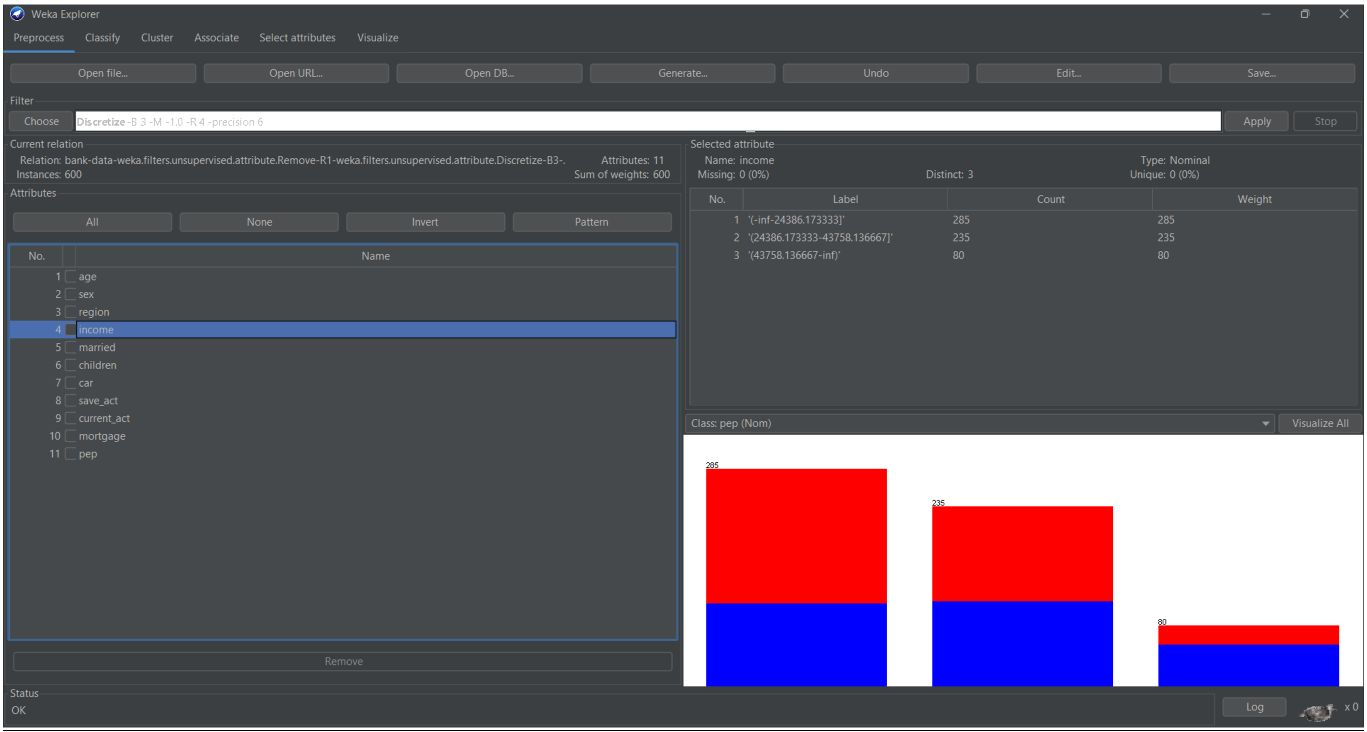
1. Discretize “children” attribute manually using text-editor.



2. Discretize “age” attribute in weka.

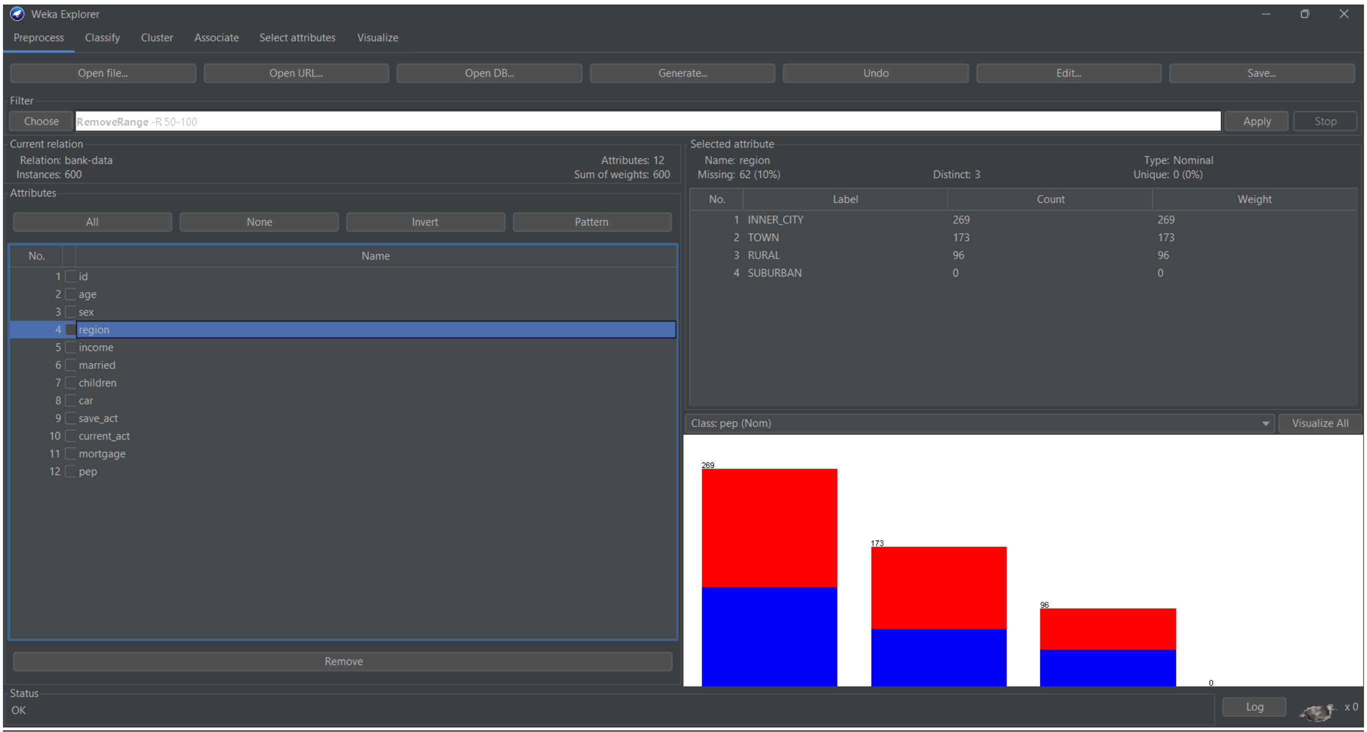


3. **Discretize "income" attribute in weka.**

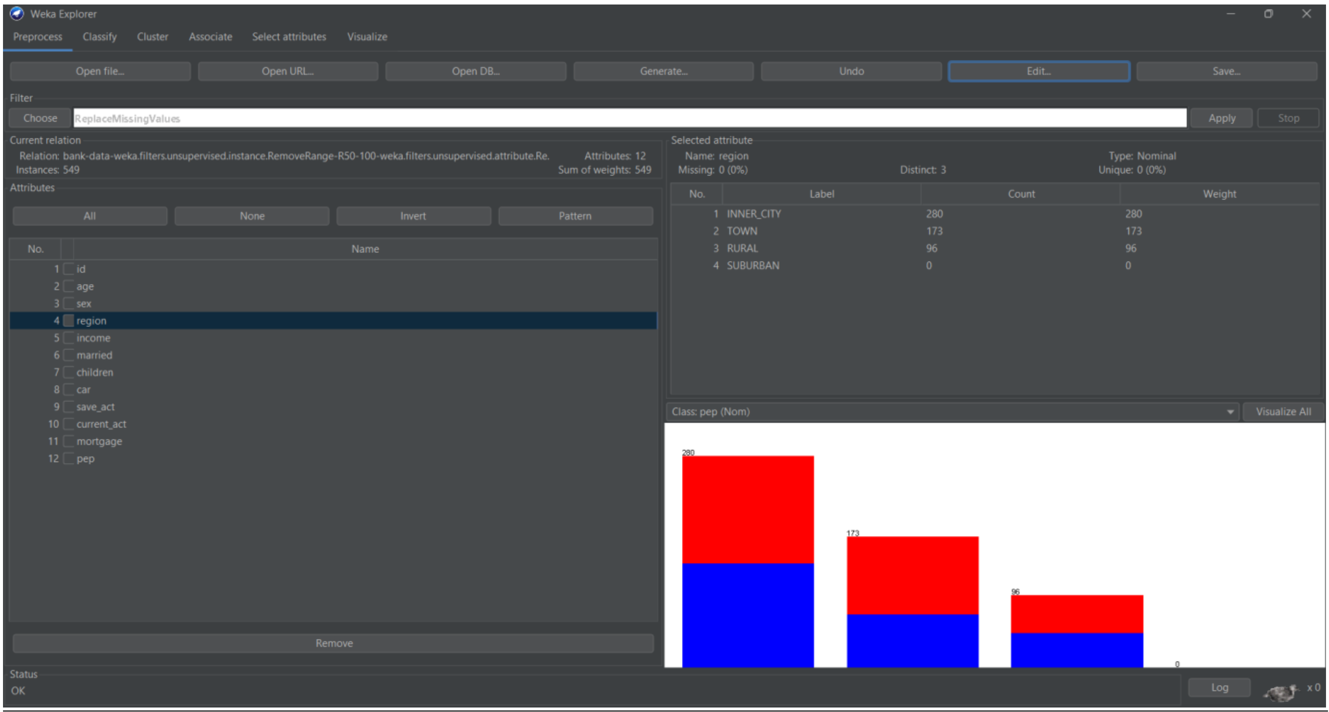


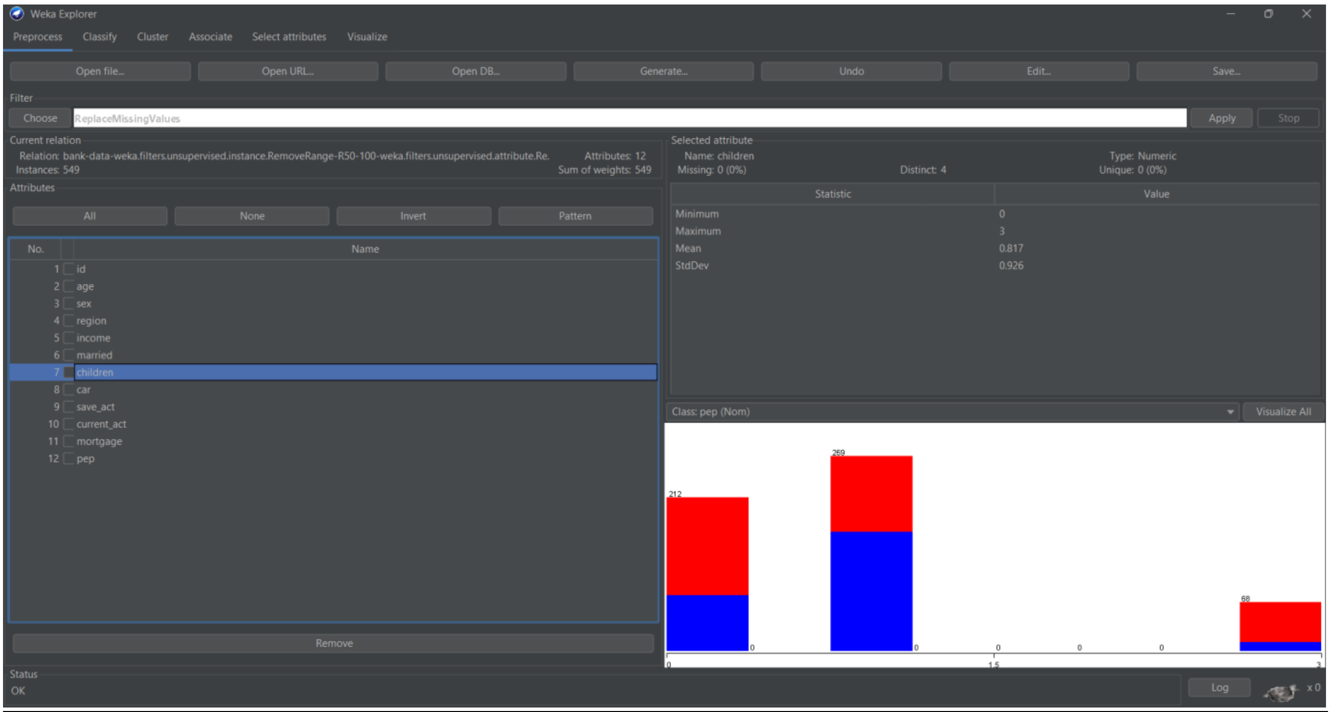
Step-4:

1. **Create missing values in "region" and "children" attributes.**



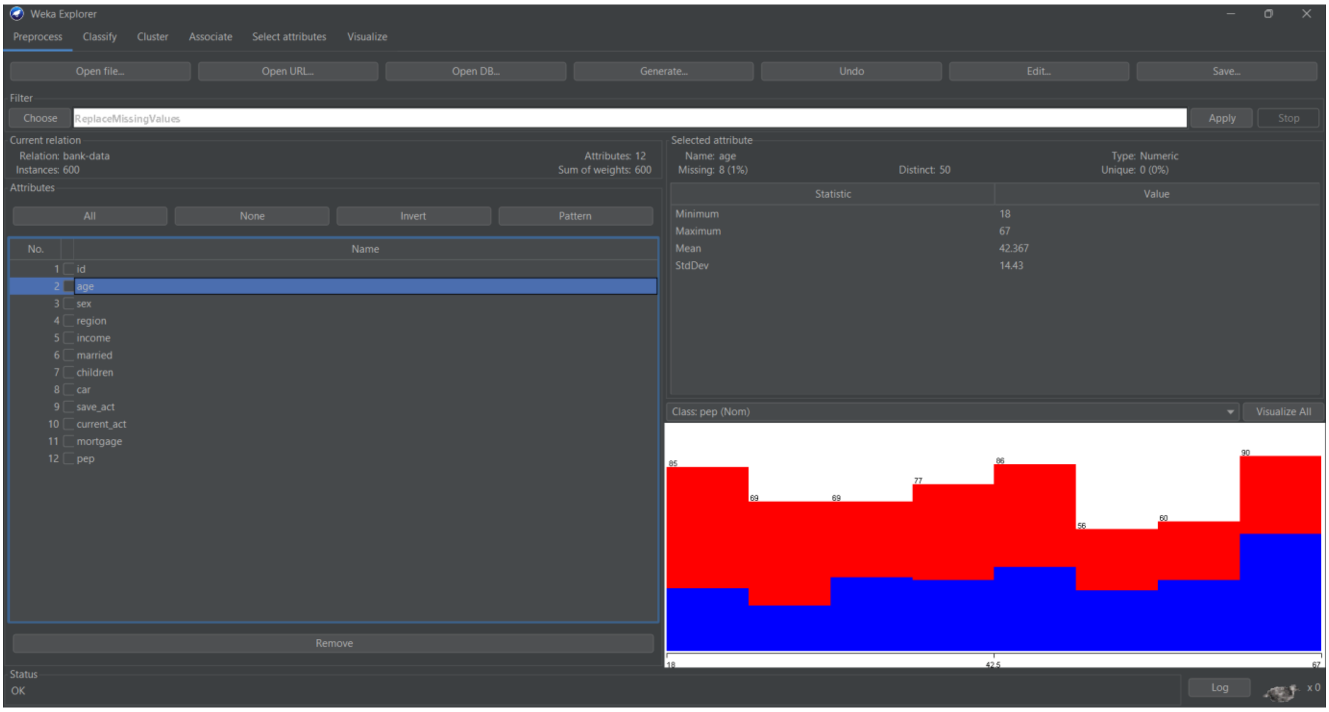
2. Replace above created missing values.



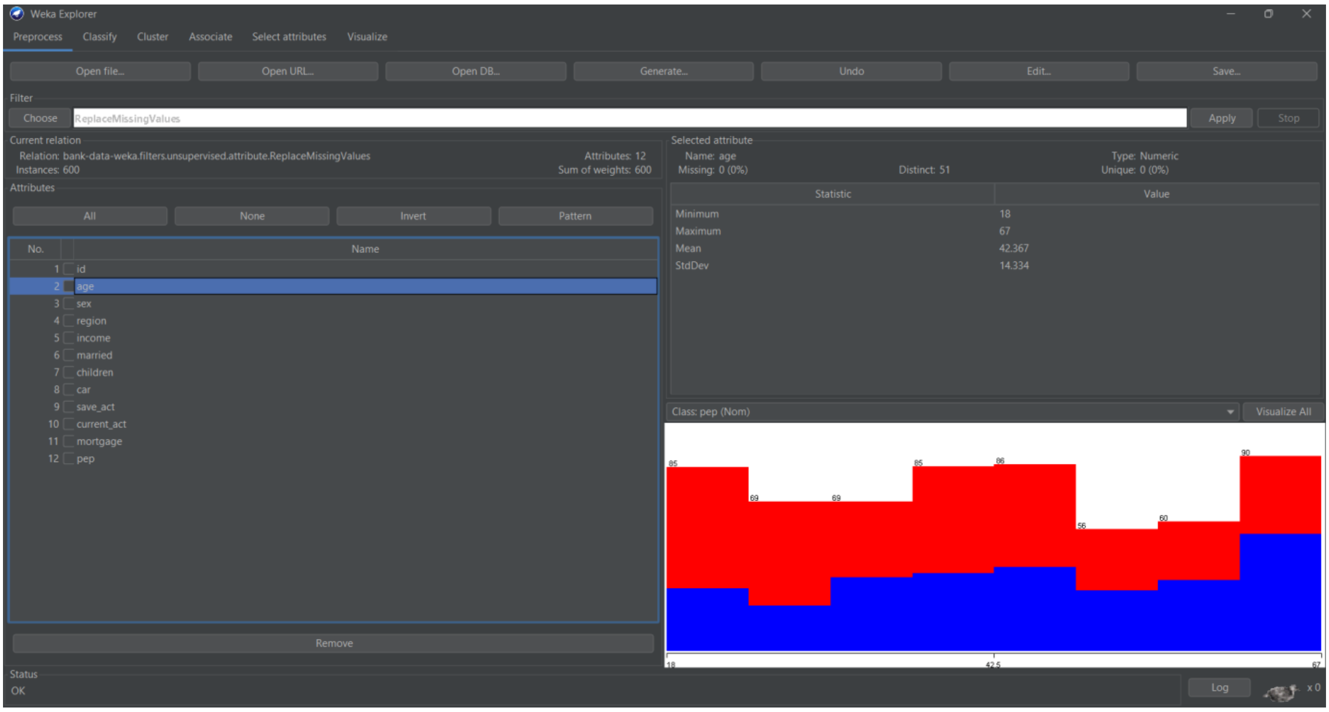


➔ The missing values were automatically replaced randomly with one of the existing values in other records in the dataset.

3. Make missing data using text-editor.



4. Replace above missing data.



Exercise-4

Objective: Decision Tree III.

1. You are required to work on the "diabetes.arff" data set. You have to apply the J48 classification algorithm with - split percentage as 80. This option will create a training data set with 80% of the points, and use the remaining points for testing.

   The focus of this exercise is to visualise the effect of the parameter called the 'Confidence Factor'. Once the tree has been built (nodes with fewer than minNumObj data points are not split), a further pass of pruning is performed by applying the confidence factor. The confidence factor may be viewed as how confident the algorithm is about the training data. (confidence factor represents a threshold of allowed inherent error in data while pruning the decision tree.) A low value leads to more pruning; a high value keeps the model close to the original tree built from the training data (the parameter is used in estimating error probabilities at leaves).

   Run experiments with the following values of the confidence factor (default 0.25, maximum 0.5), with the train/test split as specified.

   Confidence Factor: 0.002 , 0.008 , 0.02 , 0.08 , 0.1 , 0.2 , 0.3 , 0.4 , 0.5.

   For these values, record both the training and test accuracy.

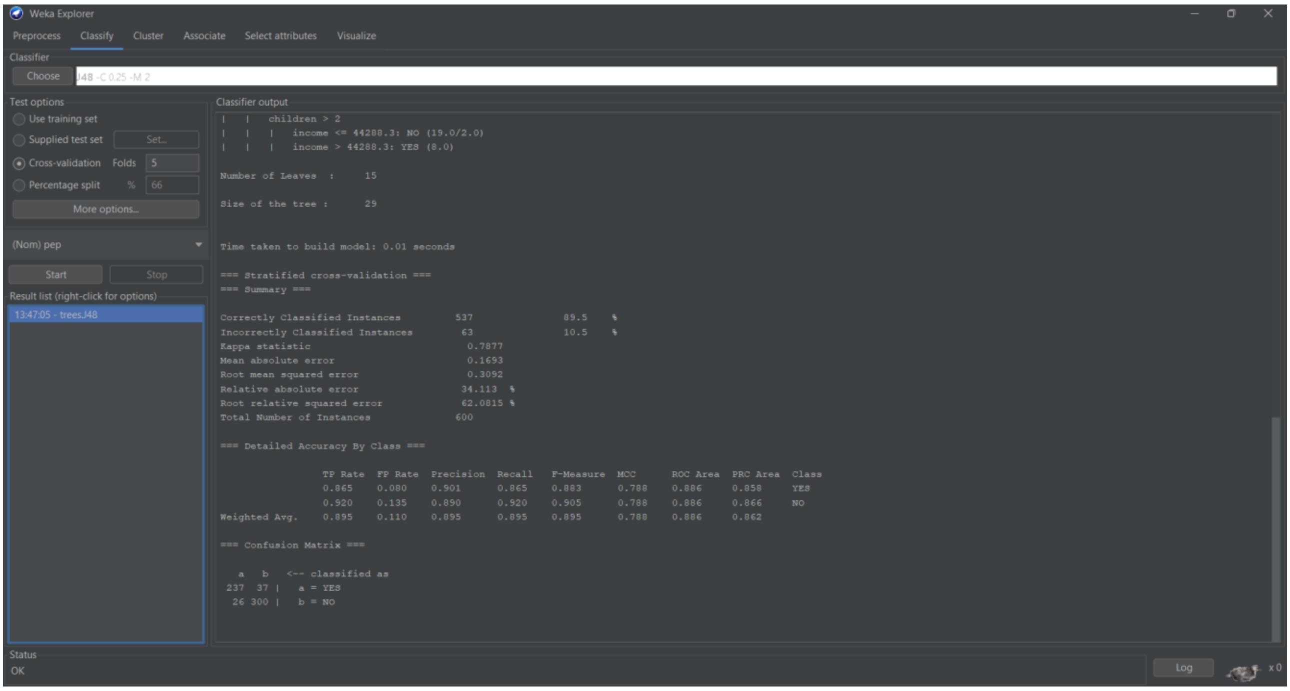
2. Comparing decision trees and neural networks

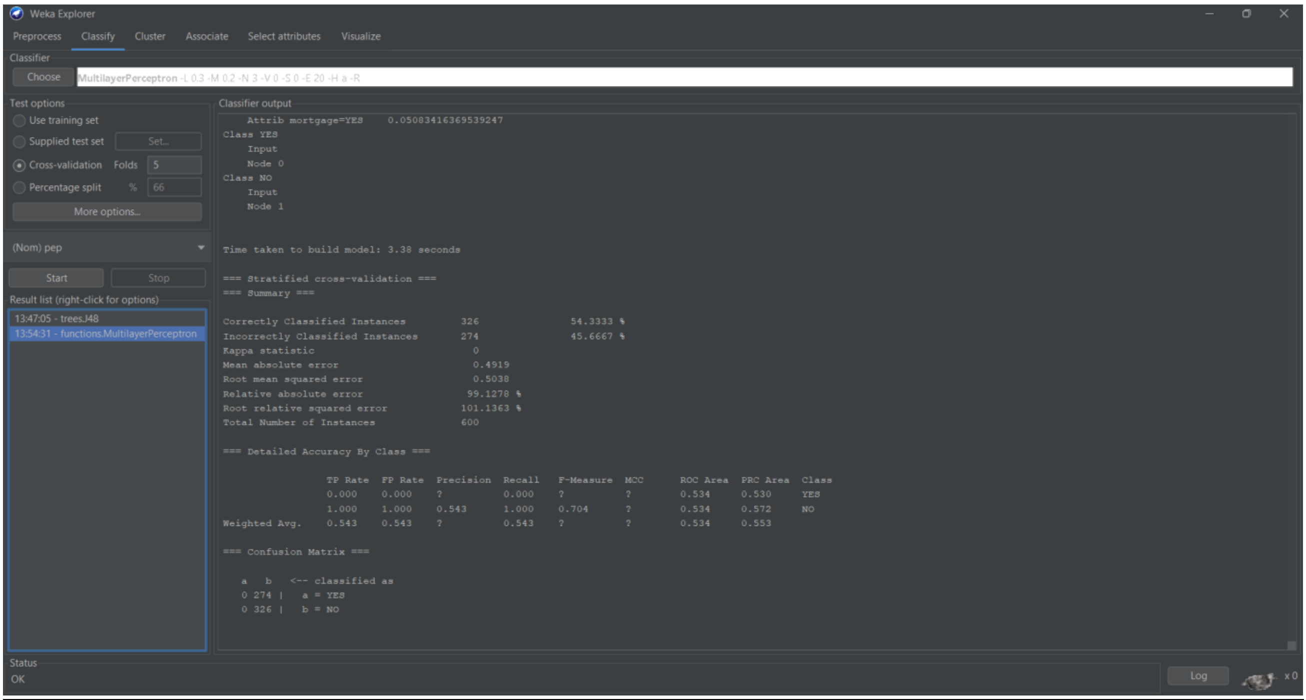
   Apply both J48 and the MultilayerPerceptron to the following data sets: "bank-data.arff". Note down the accuracy (5-fold cross-validation) while keeping the number of epochs for the Multilayer Perceptron as 3. Keep all the other parameters at their default settings.

   a) What accuracy do these methods achieve on "bank-data.arff" ?
   b) Can you think of reasons to explain their relative accuracy on these data sets?
   c) What are the properties of the data sets?
   d) Describe the decision tree model that is learned in each case.

Results:

2. Comparing decision trees and neural networks





Exercise-5

Objective: Artificial Neural Networks.

1. Run the classifier and observe the results shown in the “Classifier output” window. (Note that by default, WEKA use one hidden layer with the number of hidden neurons = (# of input attributes + # of classes) / 2.)

    1.1 How many units in the input layer, how many in the hidden layer, and how many in the output layer?

    1.2 What is the MAE (mean absolute error) made by the learned NN?

    1.3 What is the RMSE (root mean squared error) made by the learned NN?

    1.4 Visualize the errors made by the learned NN. In the plot, how can you see the detailed information of a test instance?

    1.5 Draw (on paper) the topology (together with the weights) of the learned NN.

2. To modify the value of a parameter, click on the “MultiplayerPerceptron” label. Set the value of the “momentum” parameter equal to 0.7. Run the classifier and observe the results shown in the “Classifier output” window.

    2.1 What is the MAE (mean absolute error) made by the learned NN?

    2.2 What is the RMSE (root mean squared error) made by the learned NN?

    2.3 Visualize the errors made by the learned NN. In the plot, how can you see the detailed information of a test instance?

3. Now, we shall modify the network structure. Click on the “MultiplayerPerceptron” label. Set the value of the “hiddenLayers” to “3, 2”. Run the classifier and observe the results shown in the “Classifier output” window.

    3.1 How many layers does the current NN have?

    3.2 How many units in each (input/hidden/output) layer?

    3.3 What is the MAE (mean absolute error) made by the learned NN?

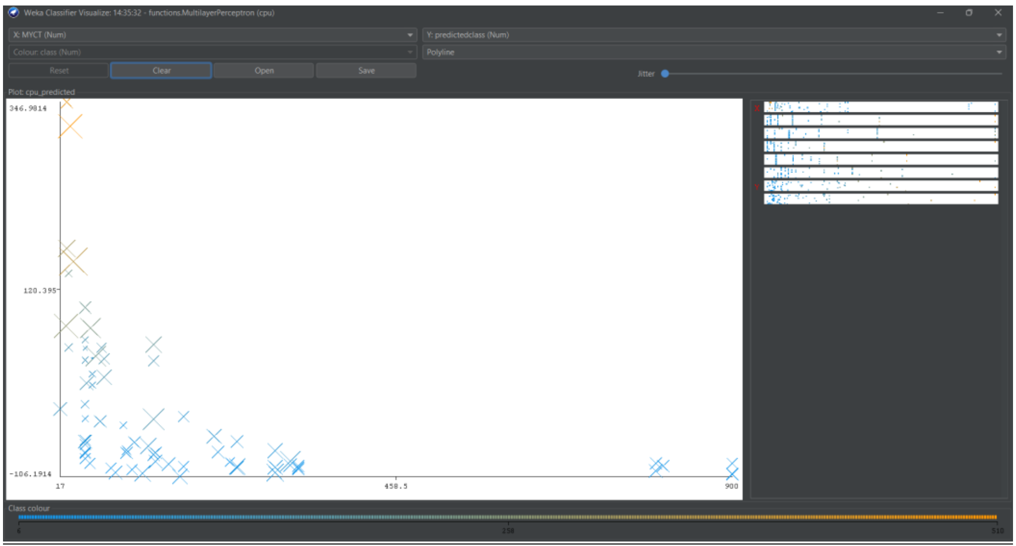
    3.4 What is the RMSE (root mean squared error) made by the learned NN?

    3.5 Visualize the errors made by the learned NN. In the plot, how can you see the detailed information of a test instance?

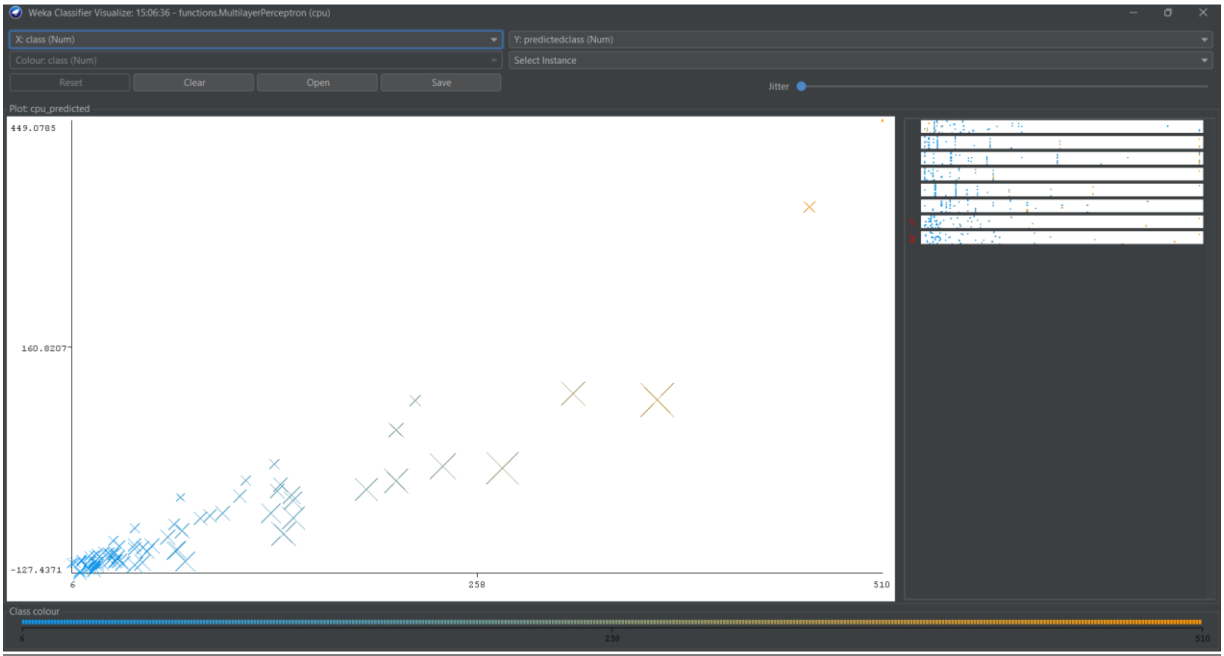
    3.6 Draw (on paper) the topology (together with the weights) of the learned NN.

Results:

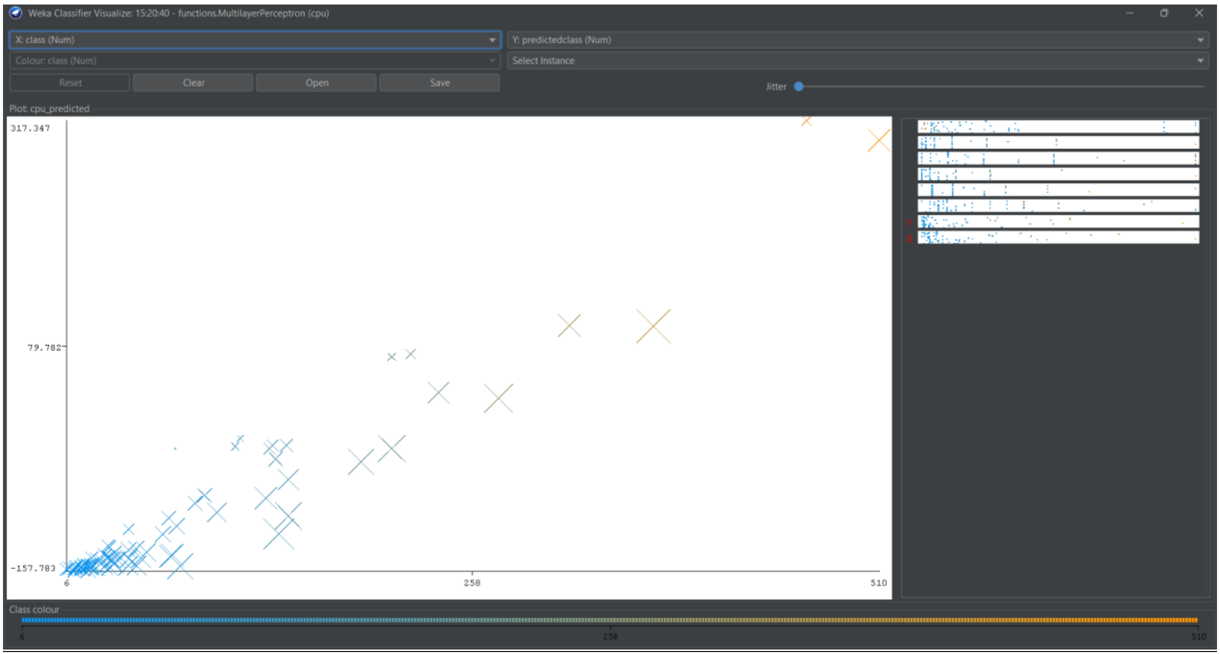
1.4.



2.3.


Weka Classifier Visualize: 15:06:36 - functions.MultilayerPerceptron (cpu)
X: class (Num)
Y: predictedclass (Num)
Colour: class (Num)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: cpu_predicted
449.0785
160.8207
-127.4371
6
258
510
Class colour
6
258
510

3.5.


Weka Classifier Visualize: 15:20:40 - functions.MultilayerPerceptron (cpu)
X: class (Num)
Y: predictedclass (Num)
Colour: class (Num)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: cpu_predicted
317.347
79.782
-157.783
6
258
510
Class colour
6
258
510

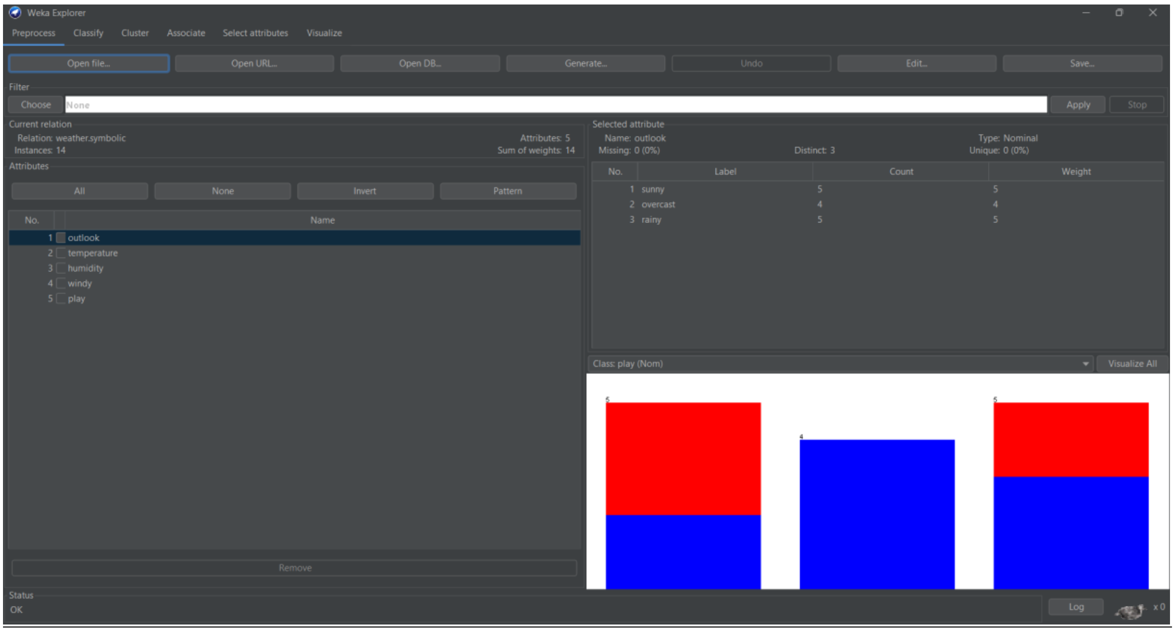
Exercise-6

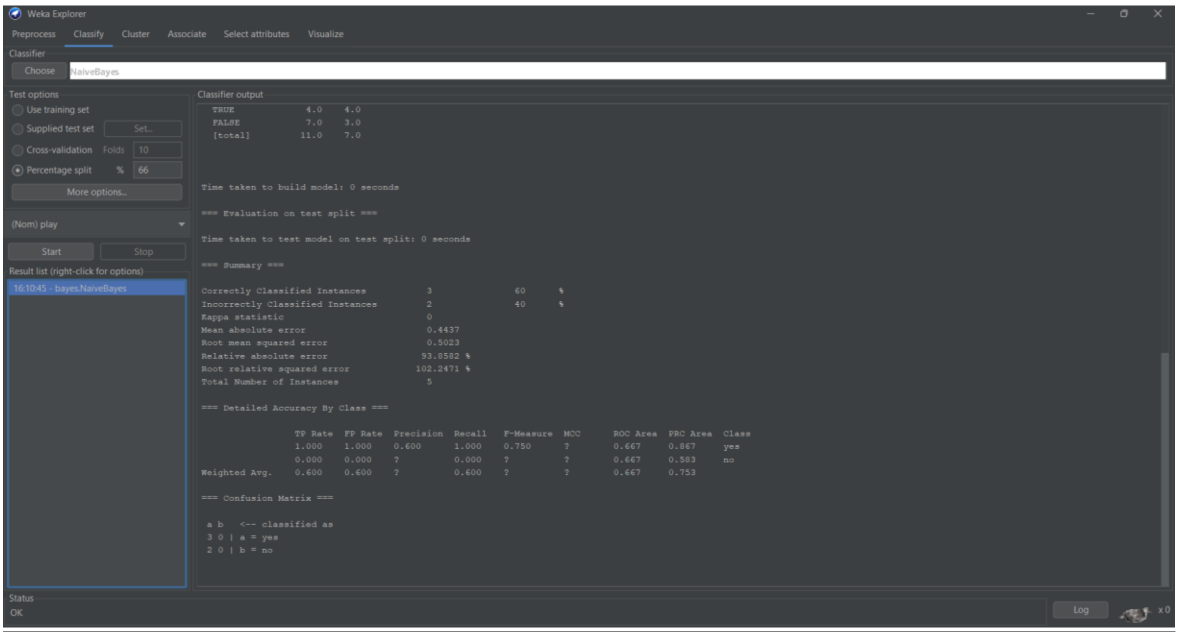
Objective: Naïve Bayesian Classifier.

The use of the Naive Bayesian classifier in Weka is demonstrated in this assignment. The “weather-nominal” data set used in this experiment is available in ARFF format. This assignment assumes that the data has been properly preprocessed.

The Bayes’ Theorem is used to build a set of classification algorithms known as Naive Bayes classifiers. It is a family of algorithms that share a common concept, namely that each pair of features being classified is independent of the others.

Results:





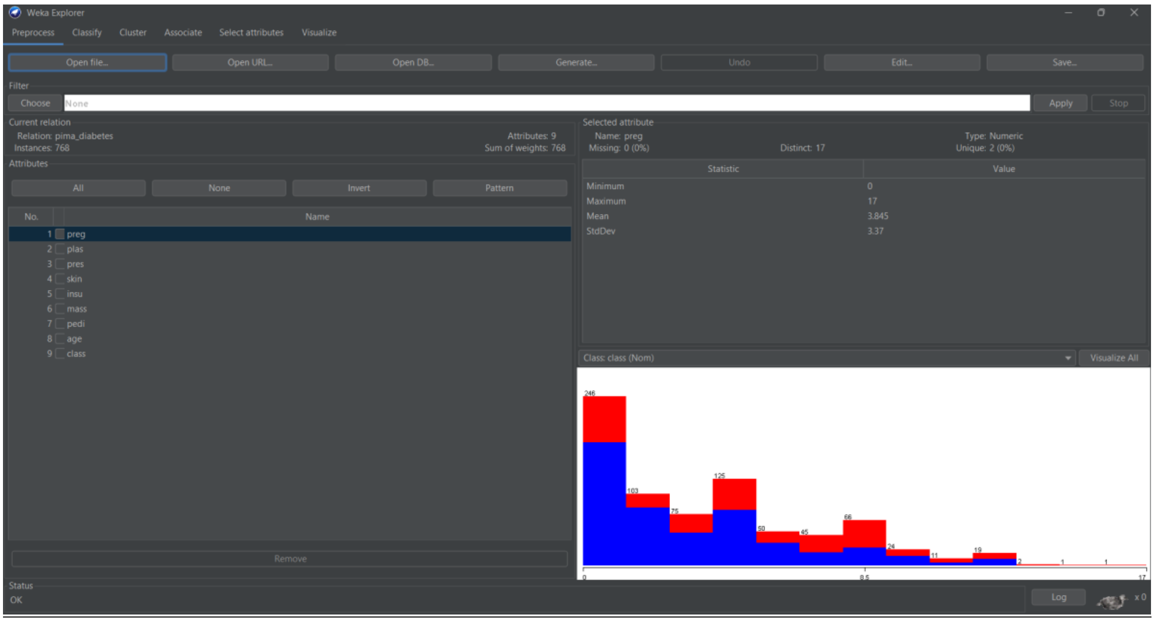
Exercise-7

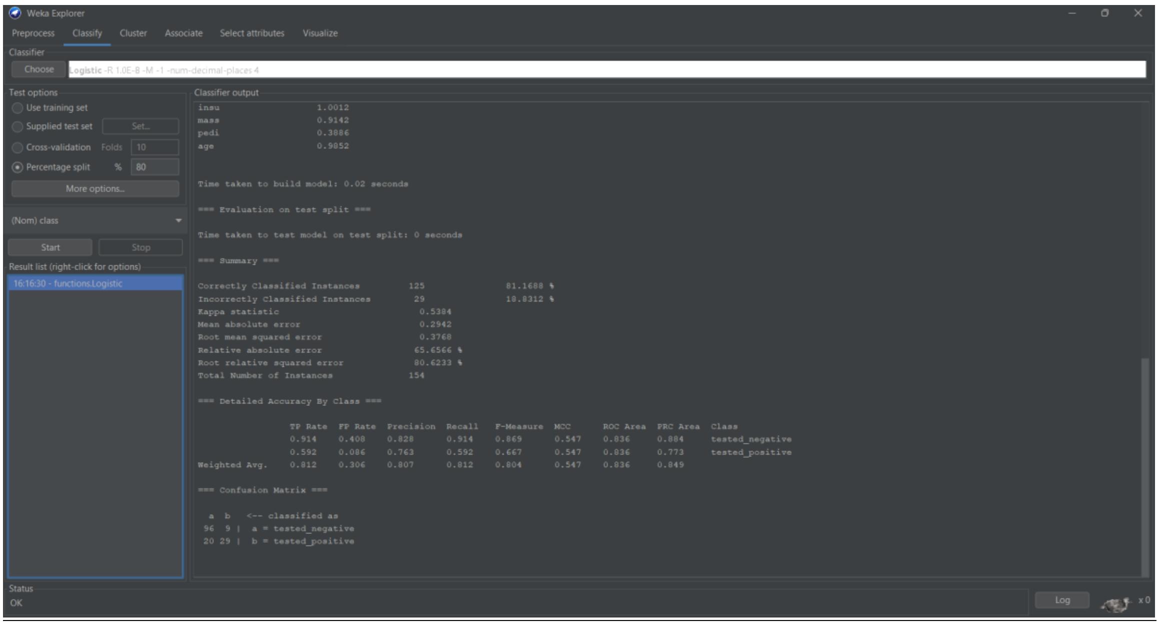
Objective: Logistic Regression.

Logistic regression is popular and powerful. It uses logit transform to predict probabilities directly.

1. Launch the WEKA tool, and activate the Explorer environment.
2. Open file diabetes.arff (pima_diabetes)(you can find this sample dataset in the folder \data).
3. Go to the Classify tab. Click on choose button. Open function folder and select Logistic.
4. Click on percentage split and change it to 80% and click start.
5. Discuss the results.

Results:



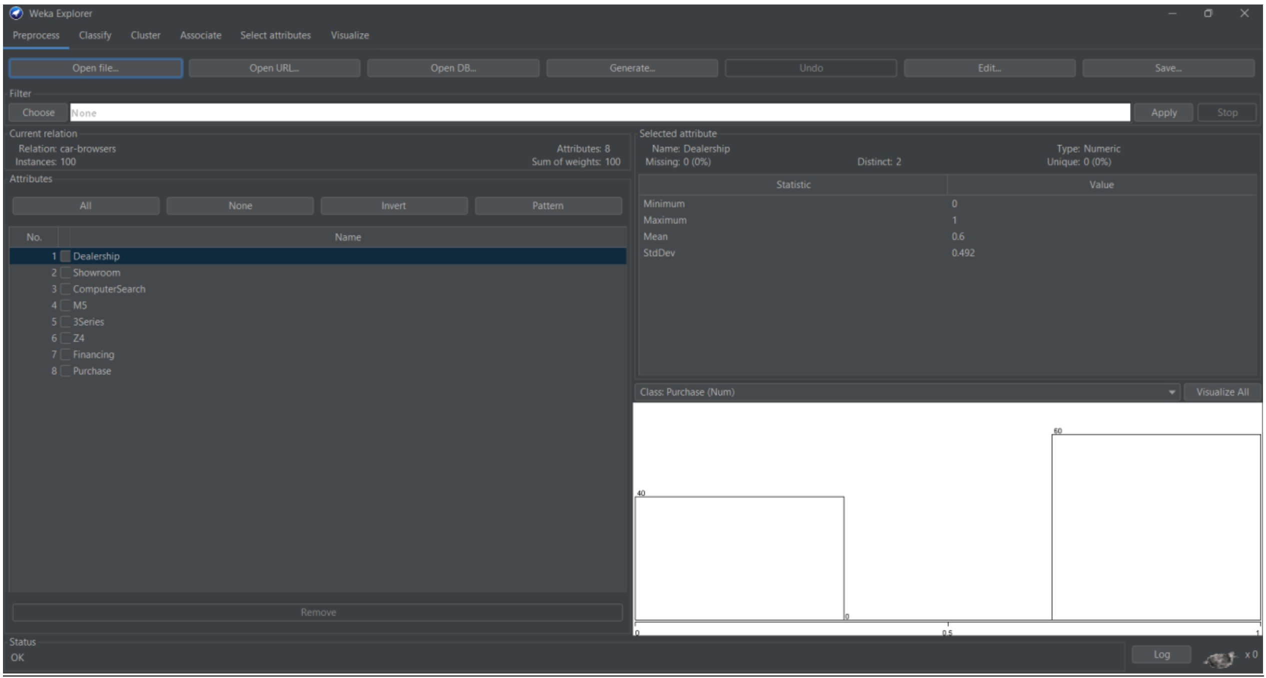


<u>Exercise-9</u>

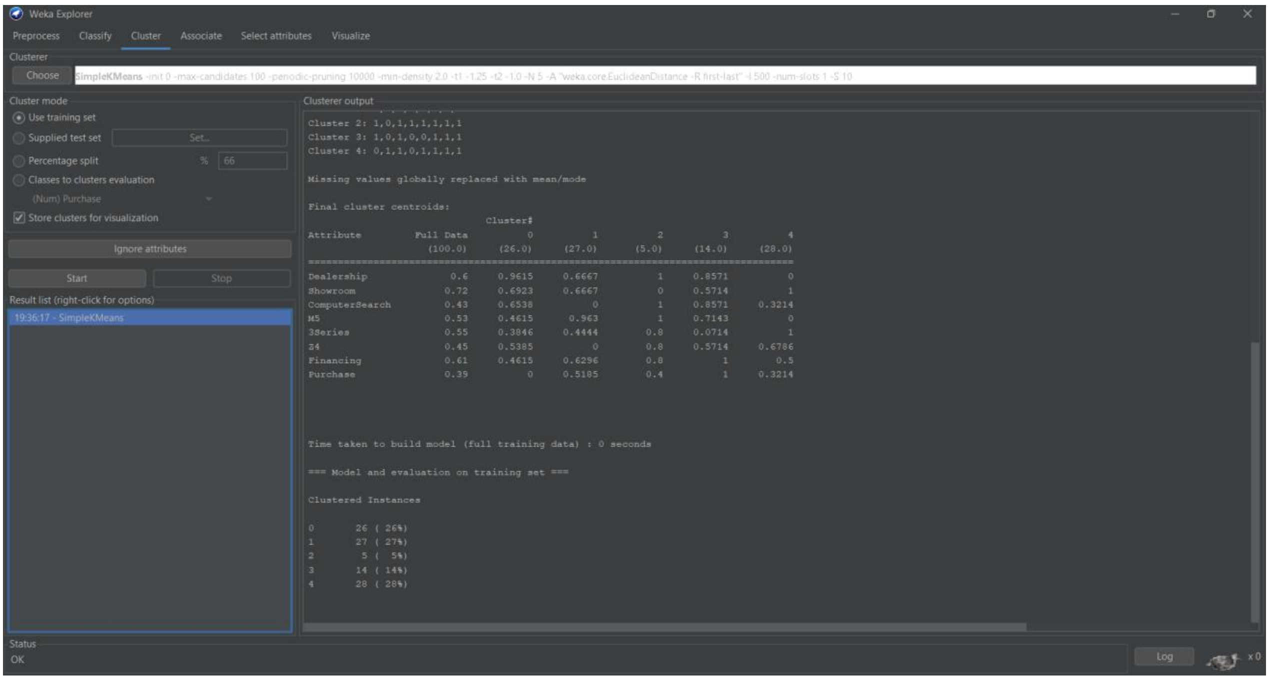
<u>Objective:</u> Clustering.

<u>Results:</u>

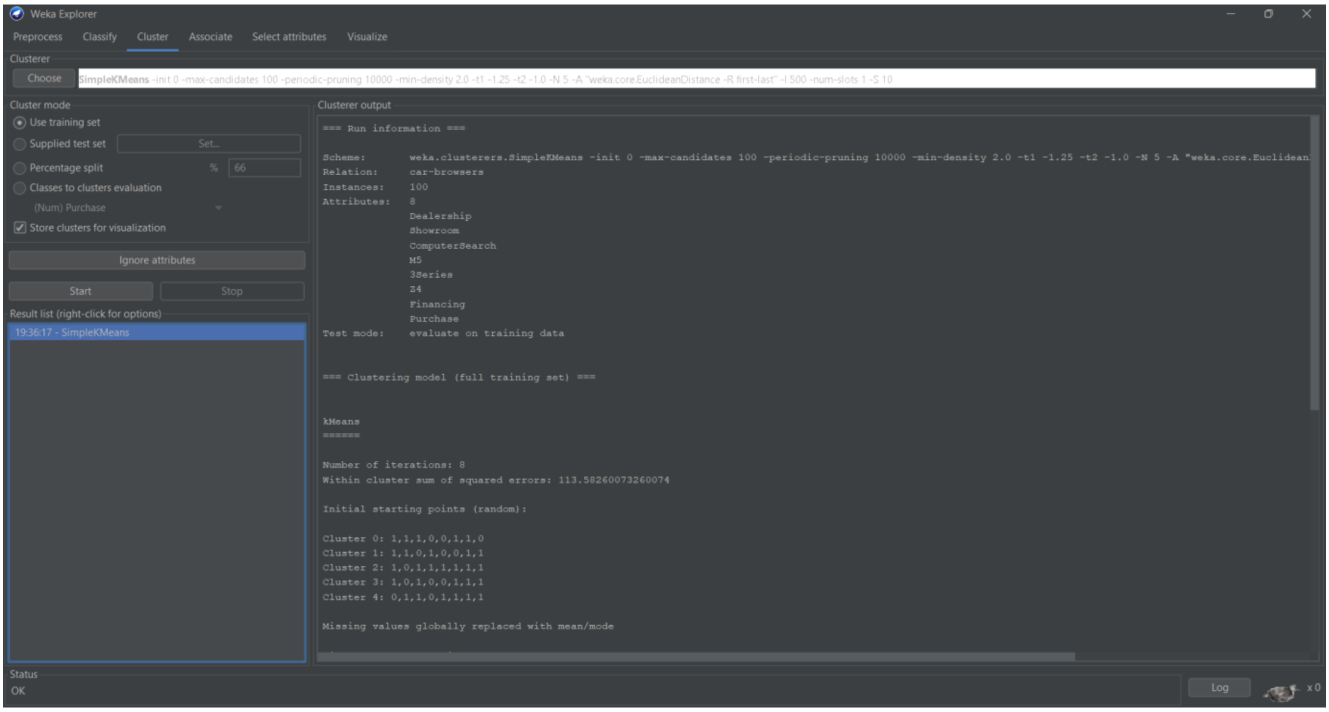
➔ Step1: Open the data/ bmw-browsers.arff Dataset



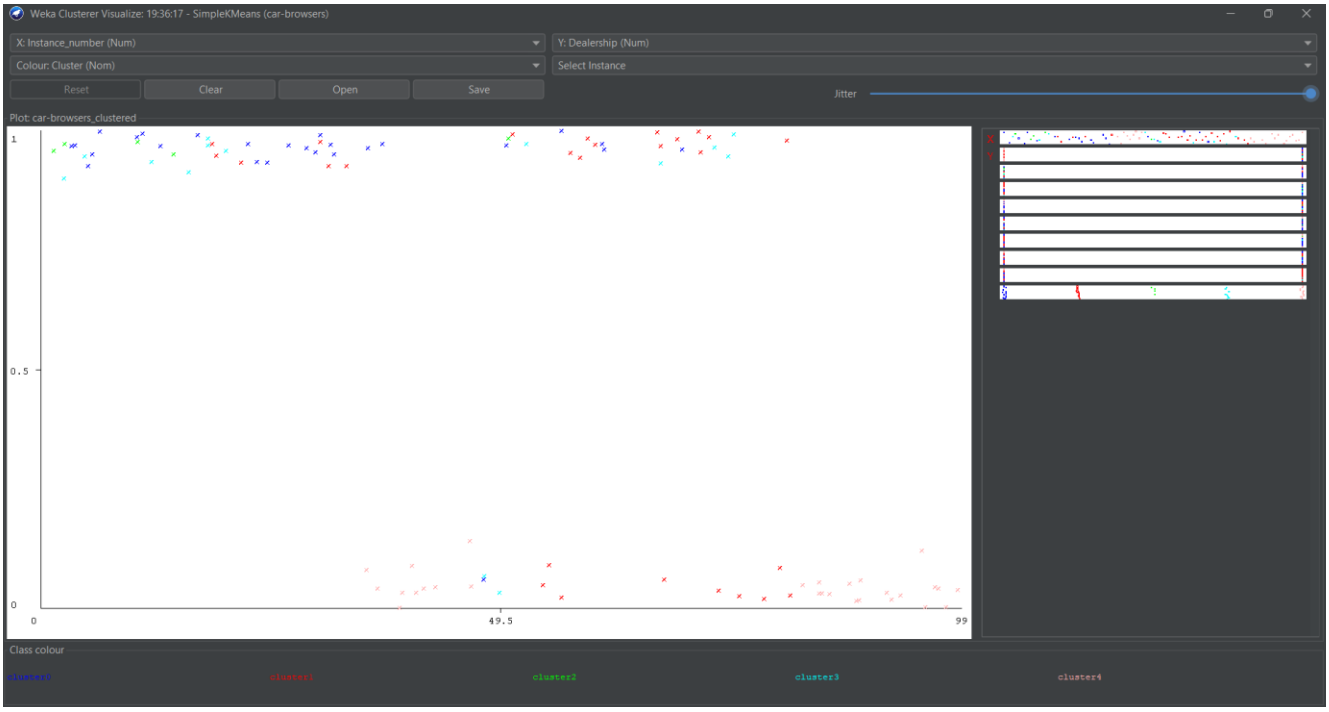
➔ Step2: Creating the Clusters with WEKA



➔ Step3: Interpreting the Clusters



➔ Step4: Cluster visual inspection



Exercise-10

Objective: K-means Clustering.

This assignment is based on analyzing clustering techniques. The objective of this assignment is to become familiar with **clustering algorithm** and evaluate or compare their performance using *Weka*.

1. Select the "**segment-test.arff**" dataset in Weka and run the Simple K-means algorithm on it using 2, 4, 8, and 16 clusters with 2 different distance functions: ***EuclideanDistance*** and ***ManhattanDistance*** respectively with a 44% percentage split (in total you would need to run it 8 times – 4 clusters with each distance function). You can keep the default values of the remaining parameters like maximum number of iterations, random seed, etc.

    1.1 How many clusters do you get?
    1.2 Visualize the clusters (graph) and take a screenshot of your results.

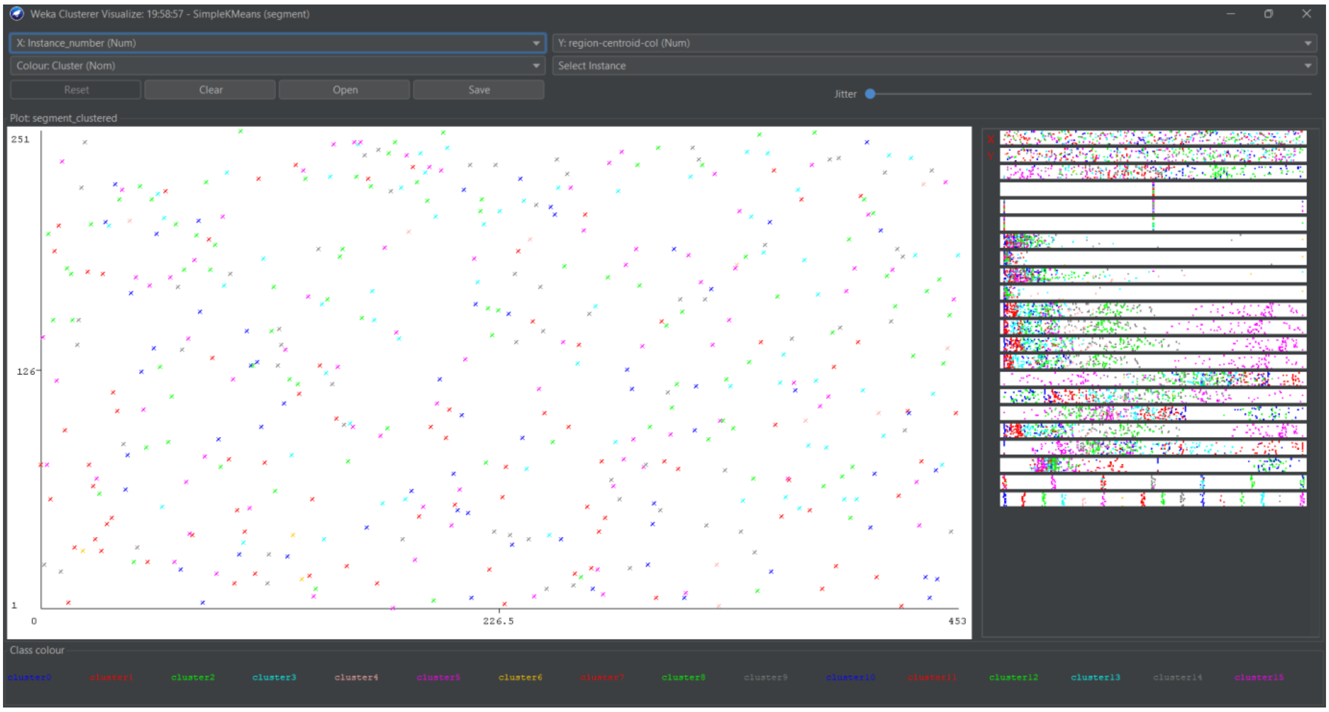
2. In the Explorer application, open the **cpu.arff** data file and do the following:

    2.1 Cluster the data (using the simple k-Means algorithm, with **k=3**) and report on the nature and composition of the extracted clusters.

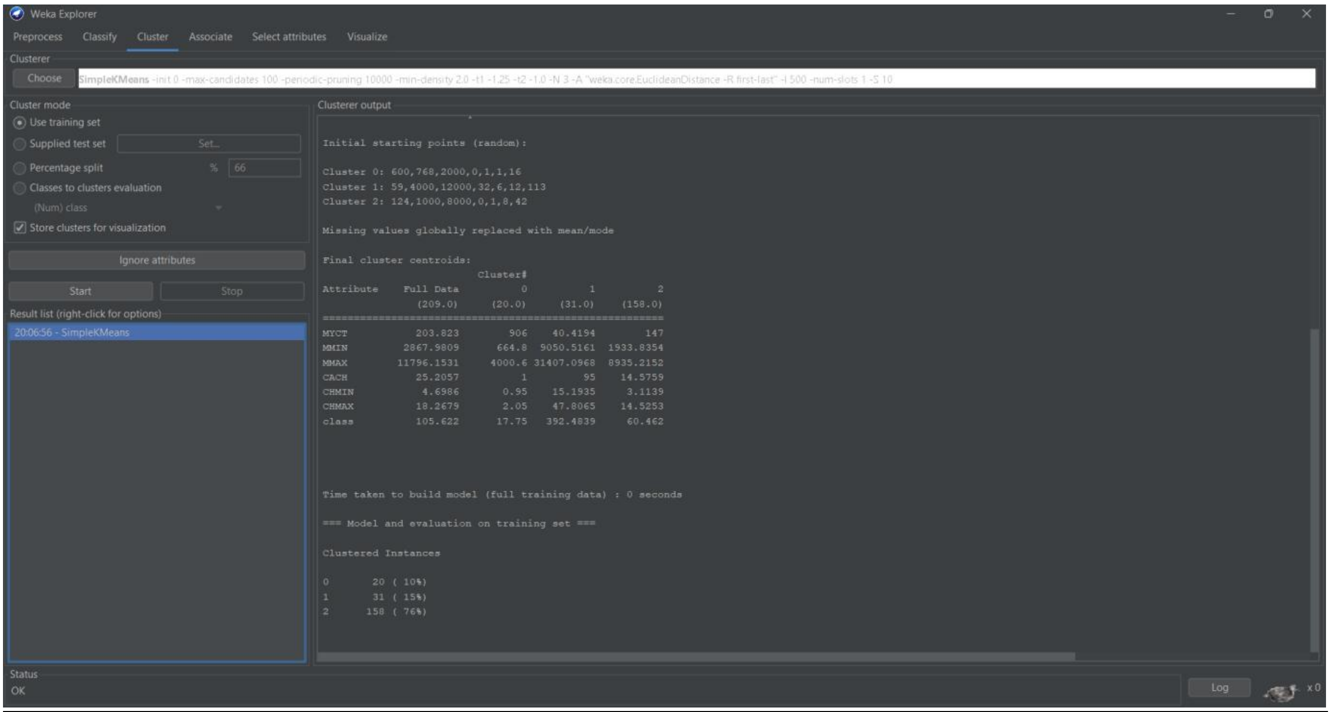
3. Load the 'weather.arff' data set and click on the 'Cluster' tab. Choose the 'SimpleKMeans' classifier with the default options. Under the 'Cluster mode' choose '**Classes to clusters evaluation'** to evaluate the clustering performance using the class labels provided in the data file. Click the 'Start' button.
    3.1 Explain the purpose of the menu under the option 'Classes to clusters evaluation' in **one sentence**.
    3.2 What is the number of clusters?
    3.3 What is the number of iterations?
    3.4 What class label is assigned to Cluster 0?
    3.5 What is the percentage of correctly clustered instances ?

Results:

1.2.



2.1.



Exercise-11

Objective: Credit card Approval detection using Machine Learning through financial data.

Credit Card Approval Detection Using Machine Learning Through Financial Data

Context

Credit score cards are a common risk control method in the financial industry. It uses personal information and data submitted by credit card applicants to predict the probability of future defaults and credit card borrowings. The bank is able to decide whether to issue a credit card to the applicant. Credit scores can objectively quantify the magnitude of risk.

Here we have various machine learning algorithms on the given financial data to draw comparisons between their performances.

Task

Build a machine learning model to predict if an applicant is 'good' or 'bad' client, different from other tasks, the definition of 'good' or 'bad' is not given. You should use some techique, such as vintage analysis to construct you label. Also, unbalance data problem is a big problem in this task.

Downloading Dataset

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("rikdifos/credit-card-approval-prediction")

print('Dataset download complete.')
```

```
Dataset download complete.
```

Imports

```
import warnings
import missingno
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Set display options to show all columns
pd.set_option('display.max_columns', None)          # Show all columns
pd.set_option('display.width', None)                # Adjust width dynamically
pd.set_option('display.expand_frame_repr', False)   # Prevent column wrapping
warnings.filterwarnings('ignore')                   # To avoid all warnings
```

Preprocessing

Firstly, we will be focusing on the application dataset.

```
application = pd.read_csv(path + "/application_record.csv")
credit_record = pd.read_csv(path + "/credit_record.csv")
```

```
print(f'application shape: {application.shape}')
print(f'credit_record shape: {credit_record.shape}')
```

```
application shape: (438557, 18)
credit_record shape: (1048575, 3)
```

```
application
```

```
             ID CODE_GENDER FLAG_OWN_CAR FLAG_OWN_REALTY  CNT_CHILDREN
AMT_INCOME_TOTAL      NAME_INCOME_TYPE            NAME_EDUCATION_TYPE
NAME_FAMILY_STATUS  NAME_HOUSING_TYPE  DAYS_BIRTH  DAYS_EMPLOYED  FLAG_MOBIL
FLAG_WORK_PHONE  FLAG_PHONE  FLAG_EMAIL OCCUPATION_TYPE  CNT_FAM_MEMBERS
0       5008804           M            Y               Y             0
427500.0              Working               Higher education        Civil
marriage   Rented apartment      -12005          -4542           1
1           0           0             NaN              2.0
1       5008805           M            Y               Y             0
427500.0              Working               Higher education        Civil
marriage   Rented apartment      -12005          -4542           1
1           0           0             NaN              2.0
2       5008806           M            Y               Y             0
112500.0              Working  Secondary / secondary special
Married   House / apartment      -21474          -1134           1
0           0           0  Security staff              2.0
3       5008808           F            N               Y             0
270000.0  Commercial associate  Secondary / secondary special  Single / not
married   House / apartment      -19110          -3051           1
0           1           1     Sales staff              1.0
4       5008809           F            N               Y             0
270000.0  Commercial associate  Secondary / secondary special  Single / not
married   House / apartment      -19110          -3051           1
0           1           1     Sales staff              1.0
...         ...         ...          ...             ...           ...
...                   ...                            ...                  ...
...         ...          ...          ...              ...          ...
...             ...               ...
438552  6840104           M            N               Y             0
135000.0             Pensioner  Secondary / secondary special
Separated  House / apartment       -22717         365243           1
0           0           0             NaN              1.0
438553  6840222           F            N               N             0
103500.0              Working  Secondary / secondary special  Single / not
married   House / apartment      -15939          -3007           1
0           0           0        Laborers              1.0
438554  6841878           F            N               N             0
54000.0  Commercial associate                Higher education  Single / not
married       With parents        -8169           -372           1
1           0           0     Sales staff              1.0
```

```
438555  6842765           F            N               Y              0
72000.0            Pensioner  Secondary / secondary special
Married  House / apartment      -21673        365243           1
0          0           0             NaN               2.0
438556  6842885           F            N               Y              0
121500.0              Working  Secondary / secondary special
Married  House / apartment      -18858         -1201           1
0          1           0     Sales staff               2.0

[438557 rows x 18 columns]
```

It appears that most of the null values in 'OCCUPATION_TYPE' feature has something in common, they all appear to be pensioners in the 'NAME_INCOME_TYPE' feature, which is logical if you think about it, that a retired person shouldn't be working.

```
application.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 438557 entries, 0 to 438556
Data columns (total 18 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   ID                   438557 non-null  int64  
 1   CODE_GENDER          438557 non-null  object 
 2   FLAG_OWN_CAR         438557 non-null  object 
 3   FLAG_OWN_REALTY      438557 non-null  object 
 4   CNT_CHILDREN         438557 non-null  int64  
 5   AMT_INCOME_TOTAL     438557 non-null  float64
 6   NAME_INCOME_TYPE     438557 non-null  object 
 7   NAME_EDUCATION_TYPE  438557 non-null  object 
 8   NAME_FAMILY_STATUS   438557 non-null  object 
 9   NAME_HOUSING_TYPE    438557 non-null  object 
 10  DAYS_BIRTH           438557 non-null  int64  
 11  DAYS_EMPLOYED        438557 non-null  int64  
 12  FLAG_MOBIL           438557 non-null  int64  
 13  FLAG_WORK_PHONE      438557 non-null  int64  
 14  FLAG_PHONE           438557 non-null  int64  
 15  FLAG_EMAIL           438557 non-null  int64  
 16  OCCUPATION_TYPE      304354 non-null  object 
 17  CNT_FAM_MEMBERS      438557 non-null  float64
dtypes: float64(2), int64(8), object(8)
memory usage: 60.2+ MB
```

- ID: Unique Id of the row
- CODE_GENDER: Gender of the applicant. M is male and F is female.
- FLAG_OWN_CAR: Is an applicant with a car. Y is Yes and N is NO.
- FLAG_OWN_REALTY: Is an applicant with realty. Y is Yes and N is No.
- CNT_CHILDREN: Count of children.
- AMT_INCOME_TOTAL: the amount of the income.
- NAME_INCOME_TYPE: The type of income (5 types in total).
- NAME_EDUCATION_TYPE: The type of education (5 types in total).
- NAME_FAMILY_STATUS: The type of family status (6 types in total).

- DAYS_BIRTH: The number of the days from birth (Negative values).
- DAYS_EMPLOYED: The number of the days from employed (Negative values). This column has error values.
- FLAG_MOBIL: Is an applicant with a mobile. 1 is True and 0 is False.
- FLAG_WORK_PHONE: Is an applicant with a work phone. 1 is True and 0 is False.
- FLAG_PHONE: Is an applicant with a phone. 1 is True and 0 is False.
- FLAG_EMAIL: Is an applicant with a email. 1 is True and 0 is False.
- OCCUPATION_TYPE: The type of occupation (19 types in total). This column has missing values.
- CNT_FAM_MEMBERS: The count of family members.

1. Duplicates

```
application.duplicated(subset='ID').value_counts()
```

```
False    438510
True         47
Name: count, dtype: int64
```

It appears that the application dataset has some duplicates, so we will be dropping them.

```
application.drop_duplicates(subset='ID', inplace=True)
```

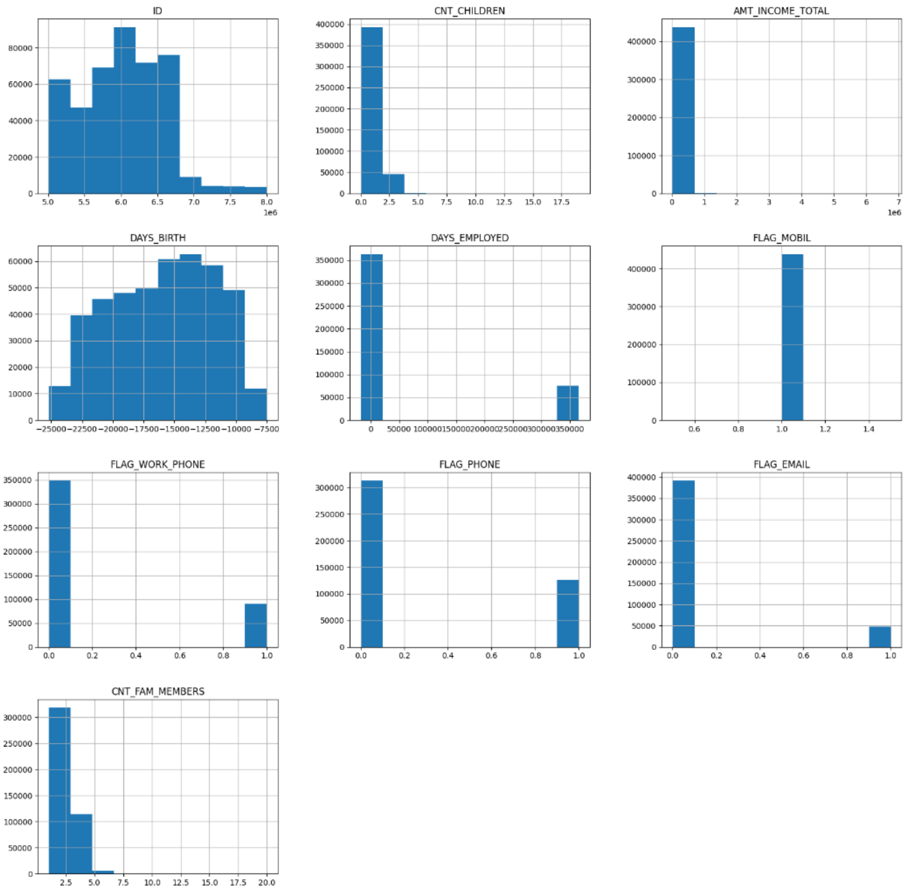
2. Unique Values

```
features = application.select_dtypes(include='object').columns.tolist()

for feature in features:
  print(f'{feature}: {application[feature].nunique()}')
  print(f'{application[feature].unique()}')
```

```
CODE_GENDER: 2
['M' 'F']
FLAG_OWN_CAR: 2
['Y' 'N']
FLAG_OWN_REALTY: 2
['Y' 'N']
NAME_INCOME_TYPE: 5
['Working' 'Commercial associate' 'Pensioner' 'State servant' 'Student']
NAME_EDUCATION_TYPE: 5
['Higher education' 'Secondary / secondary special' 'Incomplete higher'
 'Lower secondary' 'Academic degree']
NAME_FAMILY_STATUS: 5
['Civil marriage' 'Married' 'Single / not married' 'Separated' 'Widow']
NAME_HOUSING_TYPE: 6
['Rented apartment' 'House / apartment' 'Municipal apartment'
 'With parents' 'Co-op apartment' 'Office apartment']
OCCUPATION_TYPE: 18
[nan 'Security staff' 'Sales staff' 'Accountants' 'Laborers' 'Managers'
 'Drivers' 'Core staff' 'High skill tech staff' 'Cleaning staff'
 'Private service staff' 'Cooking staff' 'Low-skill Laborers'
 'Medicine staff' 'Secretaries' 'Waiters/barmen staff' 'HR staff'
 'Realty agents' 'IT staff']
```

```
application.hist(figsize=(20,20))
```

```
array([[<Axes: title={'center': 'ID'}>,
        <Axes: title={'center': 'CNT_CHILDREN'}>,
        <Axes: title={'center': 'AMT_INCOME_TOTAL'}>],
       [<Axes: title={'center': 'DAYS_BIRTH'}>,
        <Axes: title={'center': 'DAYS_EMPLOYED'}>,
        <Axes: title={'center': 'FLAG_MOBIL'}>],
       [<Axes: title={'center': 'FLAG_WORK_PHONE'}>,
        <Axes: title={'center': 'FLAG_PHONE'}>,
        <Axes: title={'center': 'FLAG_EMAIL'}>],
       [<Axes: title={'center': 'CNT_FAM_MEMBERS'}>, <Axes: >, <Axes: >]],
      dtype=object)
```



3. Null values

Like I stated before most of the null values are pensioners, so we will be further exploring that concept.

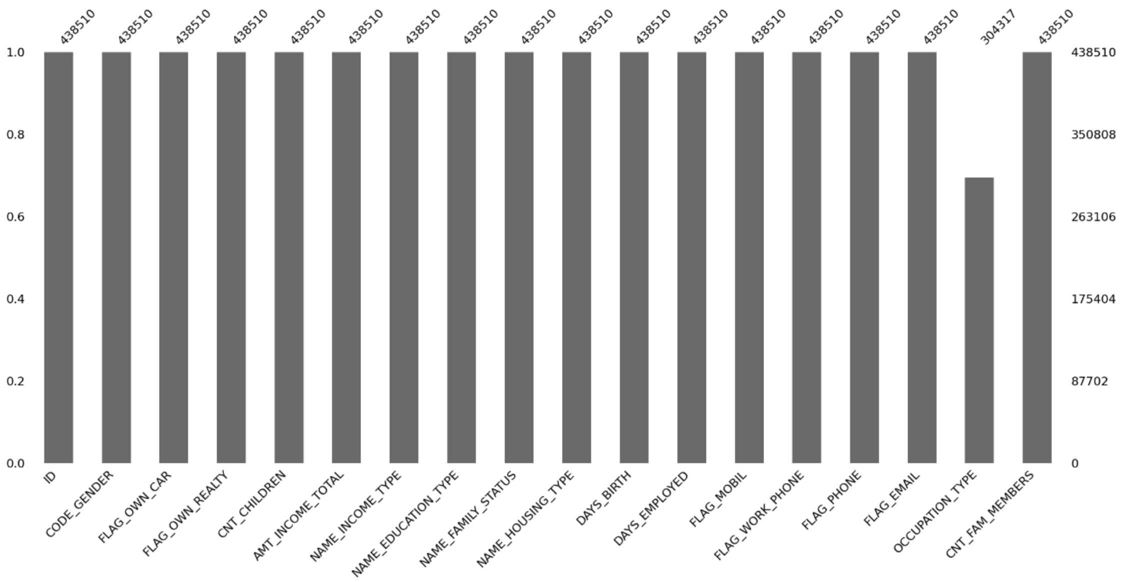
```
application.isnull().sum()
```

```
ID                            0
CODE_GENDER                   0
FLAG_OWN_CAR                  0
FLAG_OWN_REALTY               0
CNT_CHILDREN                  0
AMT_INCOME_TOTAL              0
NAME_INCOME_TYPE              0
```

```
NAME_EDUCATION_TYPE           0
NAME_FAMILY_STATUS            0
NAME_HOUSING_TYPE             0
DAYS_BIRTH                    0
DAYS_EMPLOYED                 0
FLAG_MOBIL                    0
FLAG_WORK_PHONE               0
FLAG_PHONE                    0
FLAG_EMAIL                    0
OCCUPATION_TYPE          134193
CNT_FAM_MEMBERS               0
dtype: int64
```

```
missingno.bar(application)
```

```
<Axes: >
```




```
print((application['OCCUPATION_TYPE'].isnull().sum() / application.shape[0]) *
100 ,'%')
```

```
30.602038722035985 %
```

It appears that nearly a third of 'OCCUPATION_TYPE' feature are nulls.

```
pensioners = application['OCCUPATION_TYPE'][application['NAME_INCOME_TYPE'] ==
'Pensioner']
others = application['OCCUPATION_TYPE'][application['NAME_INCOME_TYPE'] !=
'Pensioner']

print(f"Percentage of nulls that are pensioners: {(pensioners.isnull().sum() /
application['OCCUPATION_TYPE'].isnull().sum()) * 100} %")
print(f"Percentage of nulls that are NOT pensioners: {(others.isnull().sum() /
application['OCCUPATION_TYPE'].isnull().sum()) * 100} %")
```

```
Percentage of nulls that are pensioners: 56.15196023637596 %
Percentage of nulls that are NOT pensioners: 43.84803976362404 %
```

As we see that half of the null values are pensioners and the other 4 types are less than 50% .

According to the context of the data, there are 2 features that decides the 'OCCUPATION_TYPE' for the clients and they are 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE'.
So we will be checking the mode for every type of job and their academic degree.

```
df = pd.DataFrame(application[['NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
'OCCUPATION_TYPE']])
df.set_index(['NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE'], inplace=True)

df.dropna(inplace=True)

df
```

```
                                                           OCCUPATION_TYPE
NAME_INCOME_TYPE     NAME_EDUCATION_TYPE
Working              Secondary / secondary special  Security staff
Commercial associate Secondary / secondary special     Sales staff
                     Secondary / secondary special     Sales staff
                     Secondary / secondary special     Sales staff
                     Secondary / secondary special     Sales staff
...                                                            ...
Working              Higher education                     Laborers
                     Secondary / secondary special        Laborers
                     Secondary / secondary special        Laborers
Commercial associate Higher education                  Sales staff
Working              Secondary / secondary special     Sales staff

[304317 rows x 1 columns]
```

```
for job in application['NAME_INCOME_TYPE'].unique():
    for edu in application['NAME_EDUCATION_TYPE'].unique():
        if job == 'Student' and edu not in ['Higher education', 'Secondary /
secondary special']:
            continue
        if job == 'Pensioner' and edu not in ['Higher education', 'Secondary /
secondary special', 'Incomplete higher']:
            continue
        print(f"{job}, {edu}: {df.loc[job, edu]['OCCUPATION_TYPE'].mode()[0]}")
```

```
Working, Higher education: Core staff
Working, Secondary / secondary special: Laborers
Working, Incomplete higher: Laborers

Working, Lower secondary: Laborers
Working, Academic degree: Core staff
Commercial associate, Higher education: Managers
Commercial associate, Secondary / secondary special: Laborers
Commercial associate, Incomplete higher: Managers
Commercial associate, Lower secondary: Laborers
Commercial associate, Academic degree: Sales staff
Pensioner, Higher education: Core staff
Pensioner, Secondary / secondary special: Laborers
Pensioner, Incomplete higher: High skill tech staff
State servant, Higher education: Core staff
State servant, Secondary / secondary special: Core staff
State servant, Incomplete higher: Core staff
State servant, Lower secondary: Medicine staff
```

```
State servant, Academic degree: Managers
Student, Higher education: Core staff
Student, Secondary / secondary special: Laborers
```

So there are the modes for their respective position and academic degree.

We will impute the data accordingly.
Here is the percentage of nulls again before imputating any data.

```
print(f"{(application['OCCUPATION_TYPE'].isnull().sum() / application.shape[0]) *
100} %")
```

```
30.602038722035985 %
```

Now onto Data Imputation.

-> Data Imputation

```
# Working
mask = (application['NAME_INCOME_TYPE'] == 'Working') &
(application['NAME_EDUCATION_TYPE'].isin(['Higher education', 'Academic
degree']))
application.loc[mask, 'OCCUPATION_TYPE'] = application.loc[mask,
'OCCUPATION_TYPE'].fillna('Core staff')

mask = (application['NAME_INCOME_TYPE'] == 'Working') &
(application['NAME_EDUCATION_TYPE'].isin(['Secondary / secondary special',
'Incomplete higher', 'Lower secondary']))
application.loc[mask, 'OCCUPATION_TYPE'] = application.loc[mask,
'OCCUPATION_TYPE'].fillna('Laborers')

# Commercial associate
mask = (application['NAME_INCOME_TYPE'] == 'Commercial associate') &
(application['NAME_EDUCATION_TYPE'].isin(['Higher education', 'Incomplete
higher']))
application.loc[mask, 'OCCUPATION_TYPE'] = application.loc[mask,
'OCCUPATION_TYPE'].fillna('Managers')

mask = (application['NAME_INCOME_TYPE'] == 'Commercial associate') &
(application['NAME_EDUCATION_TYPE'].isin(['Secondary / secondary special', 'Lower
secondary']))
application.loc[mask, 'OCCUPATION_TYPE'] = application.loc[mask,
'OCCUPATION_TYPE'].fillna('Laborers')

mask = (application['NAME_INCOME_TYPE'] == 'Commercial associate') &
(application['NAME_EDUCATION_TYPE'].isin(['Academic degree']))
application.loc[mask, 'OCCUPATION_TYPE'] = application.loc[mask,
'OCCUPATION_TYPE'].fillna('Sales staff')

# State servant
mask = (application['NAME_INCOME_TYPE'] == 'State servant') &
(application['NAME_EDUCATION_TYPE'].isin(['Higher education', 'Secondary /
secondary special', 'Incomplete higher']))
application.loc[mask, 'OCCUPATION_TYPE'] = application.loc[mask,
'OCCUPATION_TYPE'].fillna('Core staff')

mask = (application['NAME_INCOME_TYPE'] == 'State servant') &
```

```
(application['NAME_EDUCATION_TYPE'].isin(['Lower secondary']))
application.loc[mask, 'OCCUPATION_TYPE'] = application.loc[mask,
'OCCUPATION_TYPE'].fillna('Medicine staff')

mask = (application['NAME_INCOME_TYPE'] == 'State servant') &
(application['NAME_EDUCATION_TYPE'].isin(['Academic degree']))
application.loc[mask, 'OCCUPATION_TYPE'] = application.loc[mask,
'OCCUPATION_TYPE'].fillna('Managers')

# Pensioner
mask = (application['NAME_INCOME_TYPE'] == 'Pensioner') &
(application['NAME_EDUCATION_TYPE'].isin(['Higher education']))
application.loc[mask, 'OCCUPATION_TYPE'] = application.loc[mask,
'OCCUPATION_TYPE'].fillna('Core staff')

mask = (application['NAME_INCOME_TYPE'] == 'Pensioner') &
(application['NAME_EDUCATION_TYPE'].isin(['Secondary / secondary special']))
application.loc[mask, 'OCCUPATION_TYPE'] = application.loc[mask,
'OCCUPATION_TYPE'].fillna('Laborers')

mask = (application['NAME_INCOME_TYPE'] == 'Pensioner') &
(application['NAME_EDUCATION_TYPE'].isin(['Incomplete higher']))
application.loc[mask, 'OCCUPATION_TYPE'] = application.loc[mask,
'OCCUPATION_TYPE'].fillna('High skill tech staff')

# Student
mask = (application['NAME_INCOME_TYPE'] == 'Student') &
(application['NAME_EDUCATION_TYPE'].isin(['Higher education']))
application.loc[mask, 'OCCUPATION_TYPE'] = application.loc[mask,
'OCCUPATION_TYPE'].fillna('Core staff')

mask = (application['NAME_INCOME_TYPE'] == 'Student') &
(application['NAME_EDUCATION_TYPE'].isin(['Secondary / secondary special']))
application.loc[mask, 'OCCUPATION_TYPE'] = application.loc[mask,
'OCCUPATION_TYPE'].fillna('Laborers')
```

Now to check the percentage of nulls.

```
print(f"{(application['OCCUPATION_TYPE'].isnull().sum() / application.shape[0]) *
100} %")
```

```
0.368520672276573 %
```

There is still a small percentage of nulls, which we will fill using the mode of the entire feature.

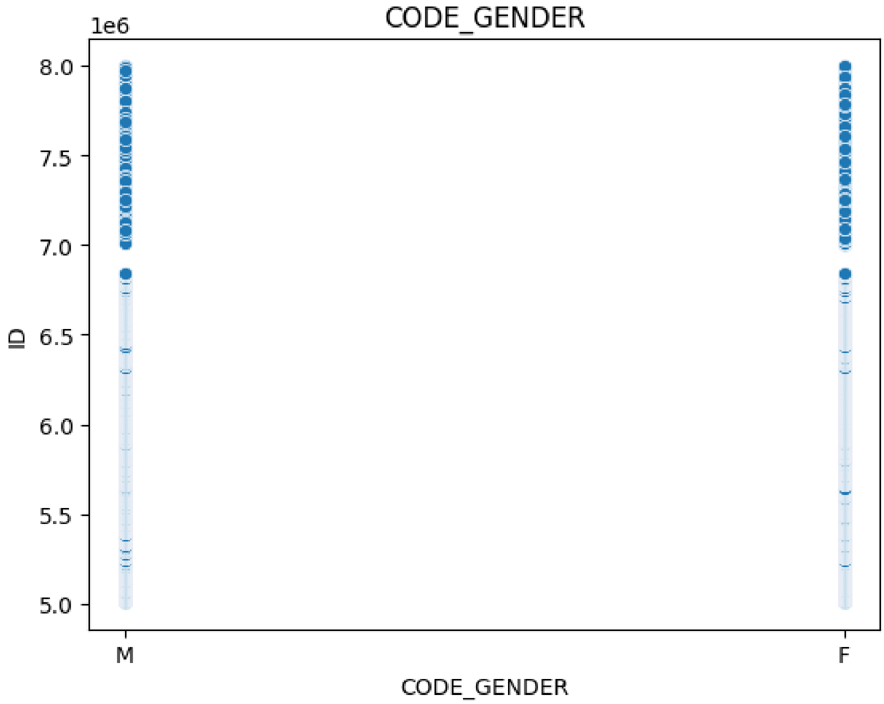
```
application['OCCUPATION_TYPE'].fillna(application['OCCUPATION_TYPE'].mode()[0],
inplace=True)

print(f"{(application['OCCUPATION_TYPE'].isnull().sum() / application.shape[0]) *
100} %")
```

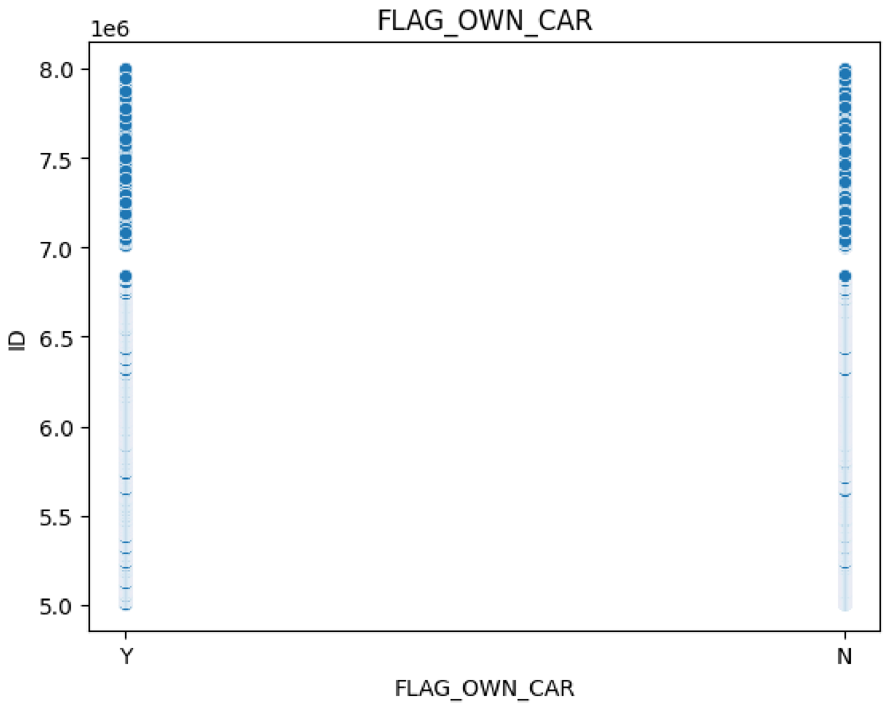
```
0.0 %
```

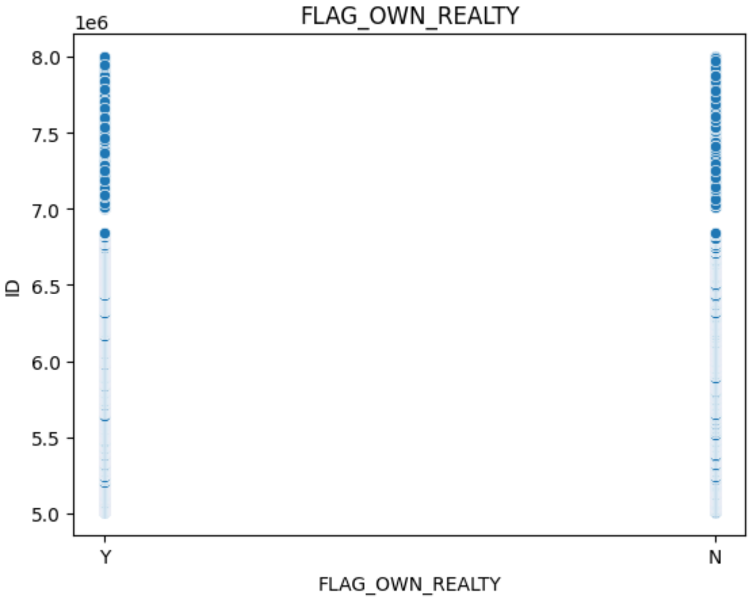
There are no more nulls.

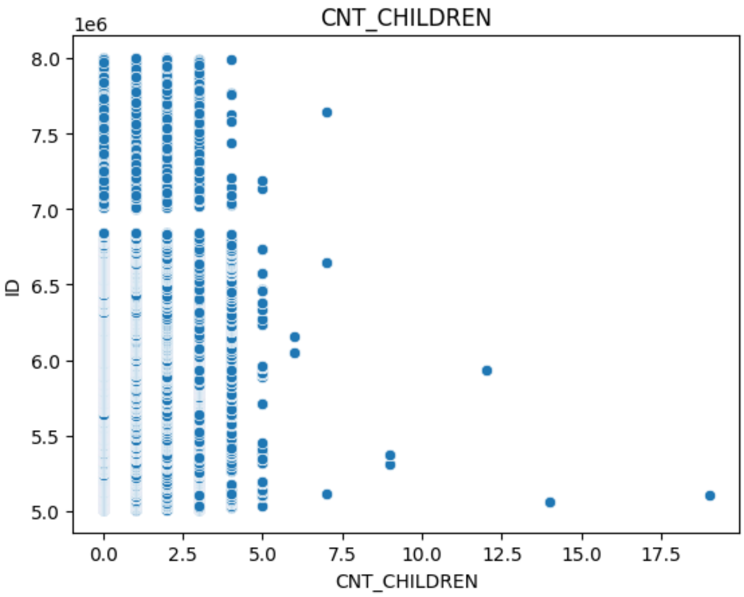
4. Outliers

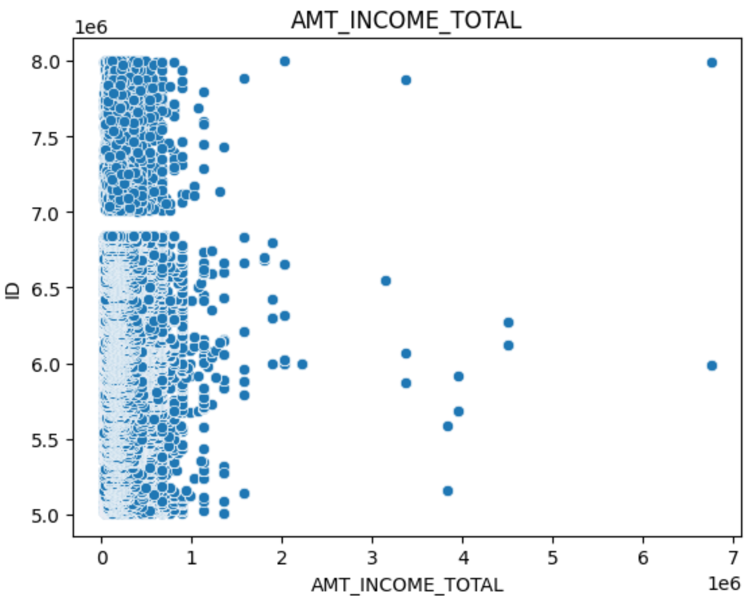
```
features = application.columns.tolist()

for feature in features:
    if feature == 'ID':
        continue
    sns.scatterplot(data=application, x=feature, y='ID')
    plt.title(feature)
    plt.show()
```

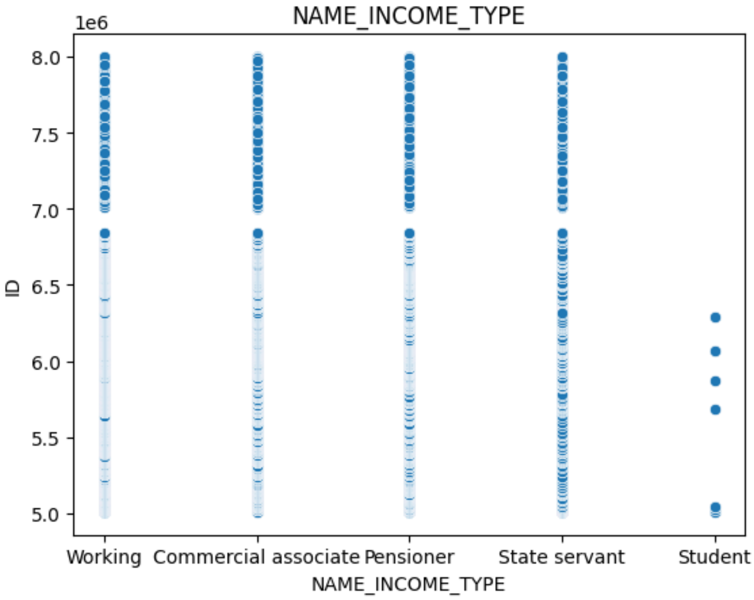


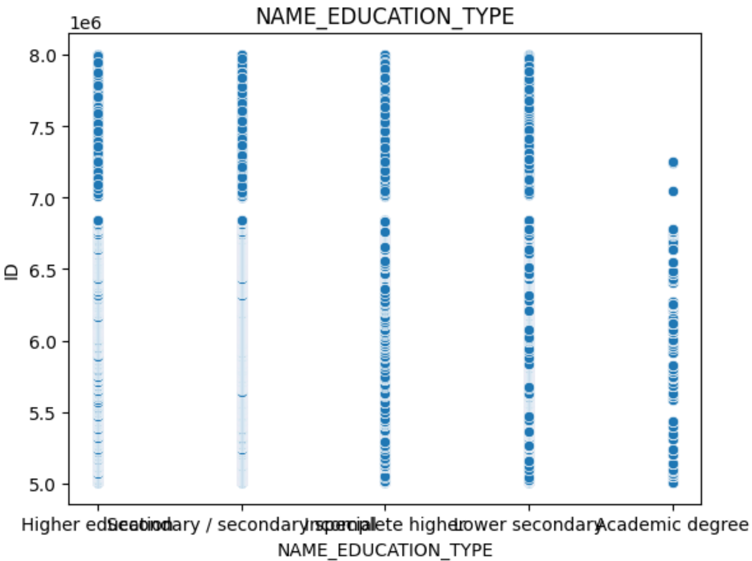


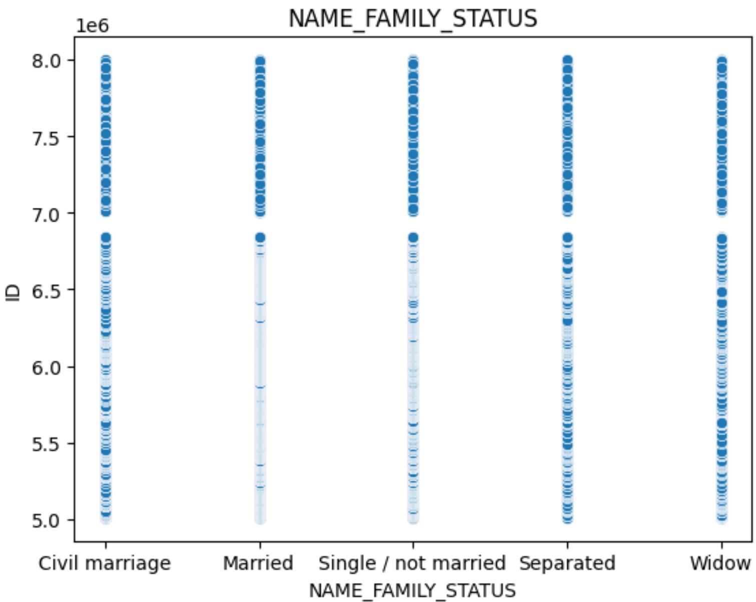

FLAG_OWN_REALTY
1e6
8.0
7.5
7.0
6.5
6.0
5.5
5.0
ID
Y
N
FLAG_OWN_REALTY

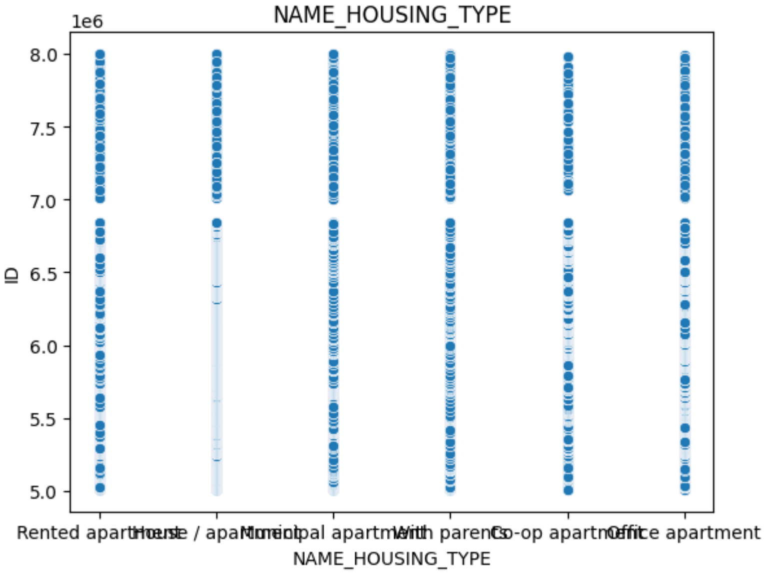

CNT_CHILDREN
1e6
8.0
7.5
7.0
6.5
6.0
5.5
5.0
ID
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
CNT_CHILDREN

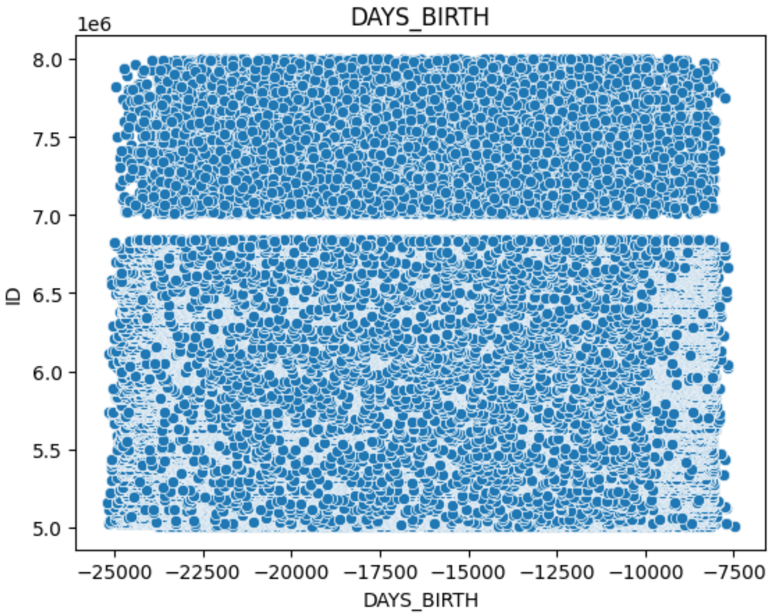

AMT_INCOME_TOTAL
1e6
8.0
7.5
7.0
6.5
6.0
5.5
5.0
ID
0
1
2
3
4
5
6
7
1e6
AMT_INCOME_TOTAL

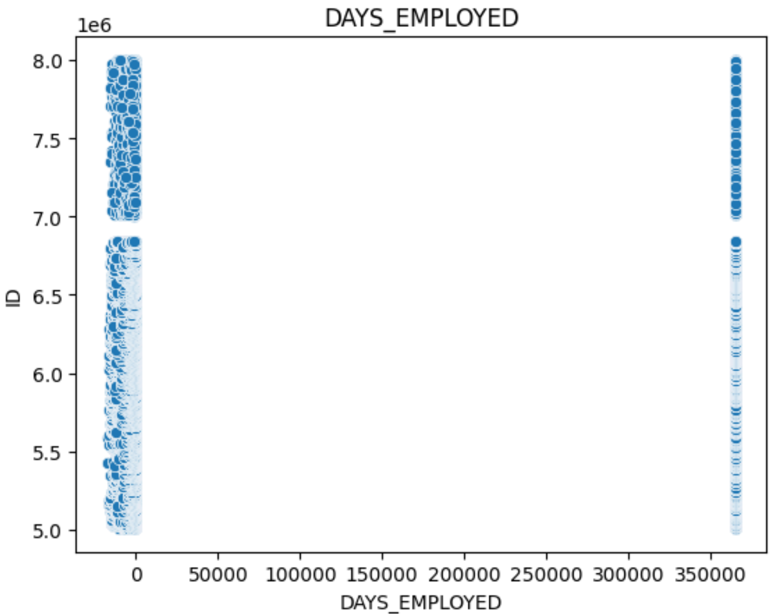

NAME_INCOME_TYPE
1e6
8.0
7.5
7.0
6.5
6.0
5.5
5.0
ID
Working
Commercial associate
Pensioner
State servant
Student
NAME_INCOME_TYPE

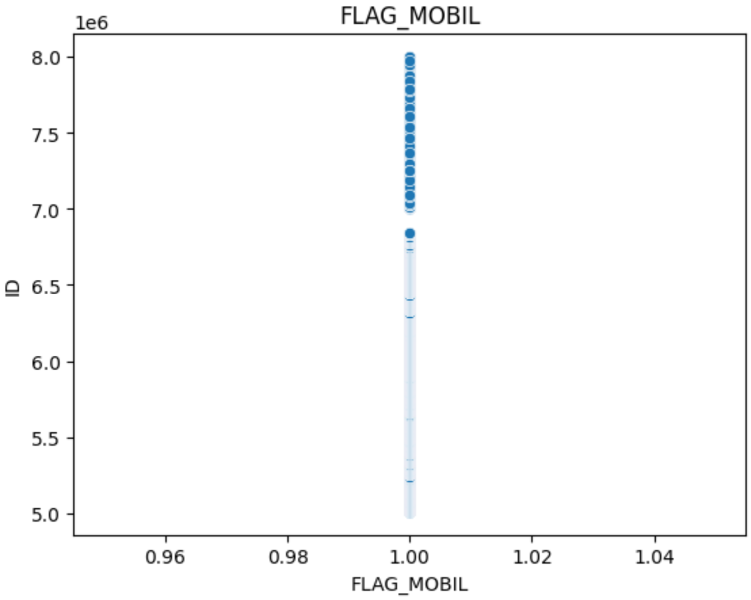

NAME_EDUCATION_TYPE
1e6
8.0
7.5
7.0
6.5
6.0
5.5
5.0
ID
Academic degree
NAME_EDUCATION_TYPE

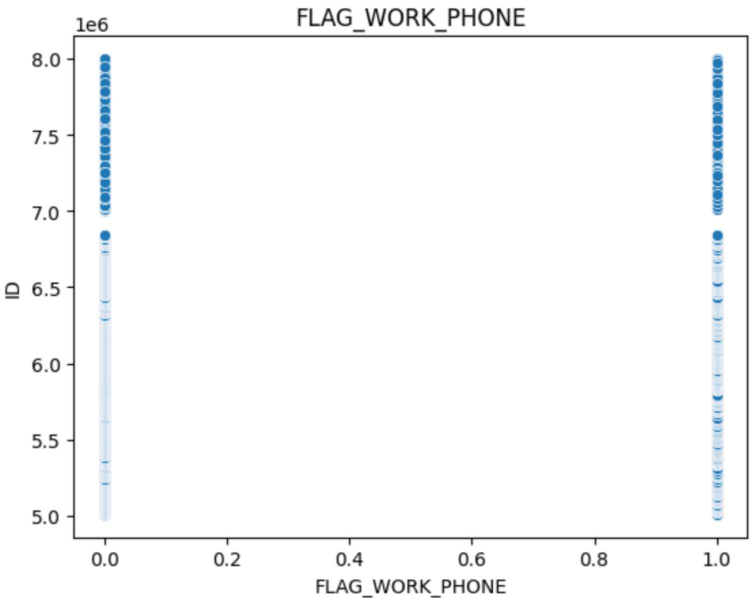

NAME_FAMILY_STATUS
1e6
8.0
7.5
7.0
6.5
6.0
5.5
5.0
ID
Civil marriage
Married
Single / not married
Separated
Widow
NAME_FAMILY_STATUS

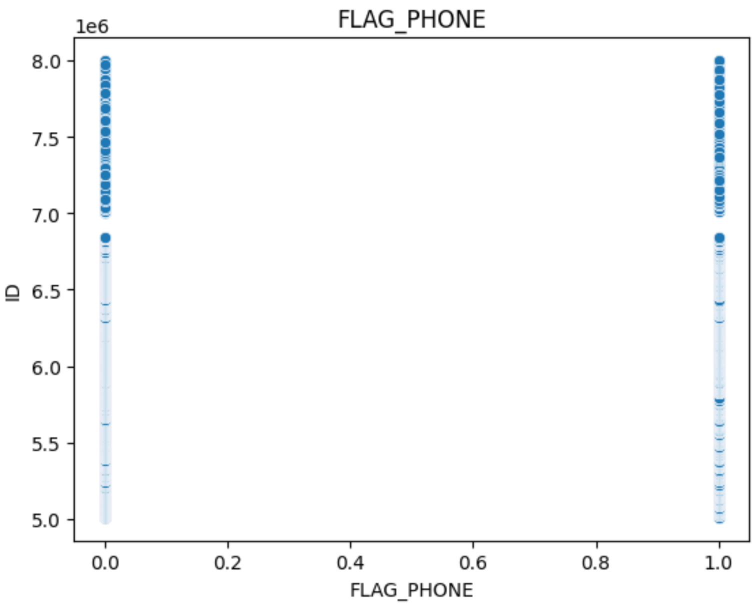

NAME_HOUSING_TYPE
1e6
8.0
7.5
7.0
6.5
6.0
5.5
5.0
ID
Rented apartment
House / apartment
Municipal apartment
With parents
Co-op apartment
Office apartment
NAME_HOUSING_TYPE

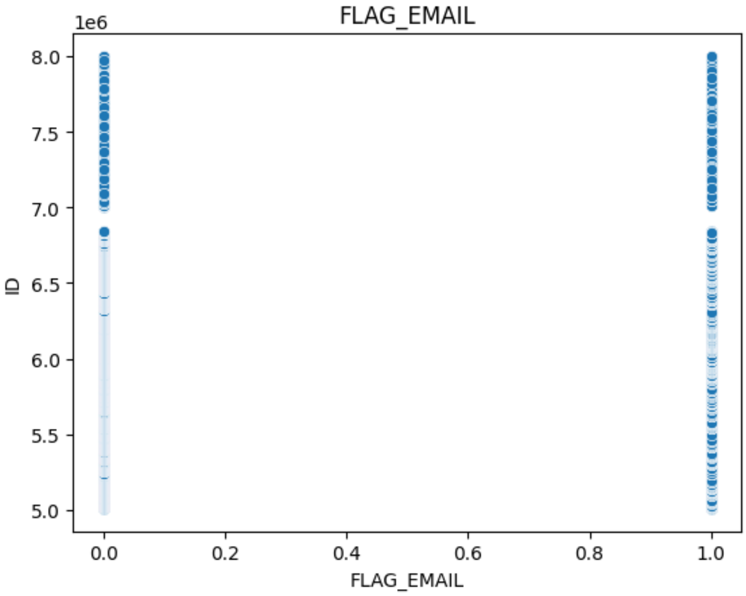

DAYS_BIRTH
1e6
8.0
7.5
7.0
6.5
6.0
5.5
5.0
ID
−25000
−22500
−20000
−17500
−15000
−12500
−10000
−7500
DAYS_BIRTH

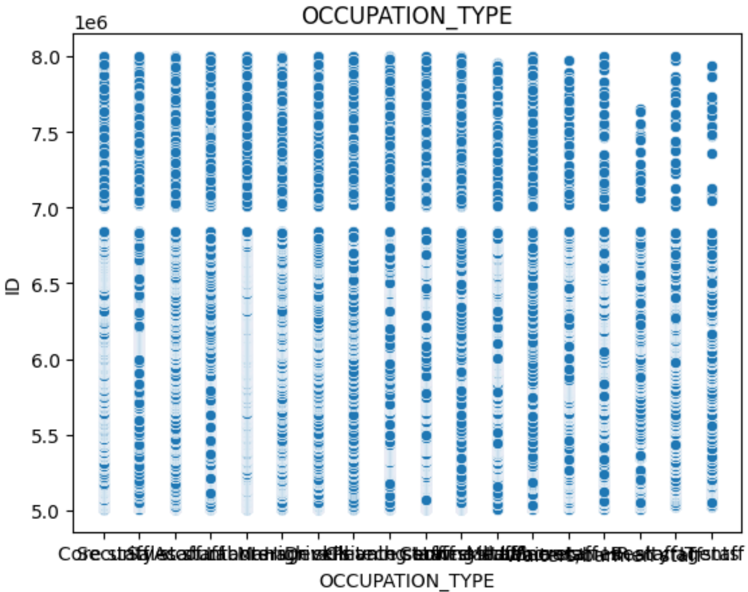

DAYS_EMPLOYED
1e6
8.0
7.5
7.0
6.5
6.0
5.5
5.0
ID
0
50000
100000
150000
200000
250000
300000
350000
DAYS_EMPLOYED

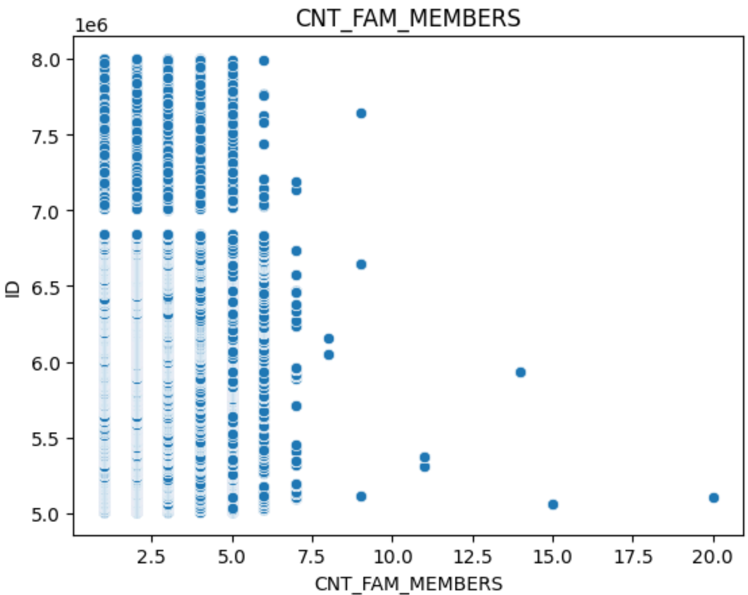

FLAG_MOBIL
1e6
8.0
7.5
7.0
6.5
6.0
5.5
5.0
ID
0.96
0.98
1.00
1.02
1.04
FLAG_MOBIL


FLAG_WORK_PHONE
1e6
8.0
7.5
7.0
6.5
6.0
5.5
5.0
ID
0.0
0.2
0.4
0.6
0.8
1.0
FLAG_WORK_PHONE


FLAG_PHONE
1e6
8.0
7.5
7.0
6.5
6.0
5.5
5.0
ID
0.0
0.2
0.4
0.6
0.8
1.0
FLAG_PHONE


FLAG_EMAIL
1e6
ID
FLAG_EMAIL


OCCUPATION_TYPE
1e6
ID
OCCUPATION_TYPE


CNT_FAM_MEMBERS
1e6
ID
CNT_FAM_MEMBERS

It appears there are outliers in the following columns:

- CNT_CHILDREN
- AMT_INCOME_TOTAL
- CNT_FAM_MEMBERS

we will remove those outliers.

```
# CNT_CHILDREN
q_hi = application['CNT_CHILDREN'].quantile(0.999)
q_low = application['CNT_CHILDREN'].quantile(0.001)
application = application[(application['CNT_CHILDREN'] > q_low) &
(application['CNT_CHILDREN'] < q_hi)]
```

```
# AMT_INCOME_TOTAL
q_hi = application['AMT_INCOME_TOTAL'].quantile(0.999)
q_low = application['AMT_INCOME_TOTAL'].quantile(0.001)
application = application[(application['AMT_INCOME_TOTAL'] > q_low) &
(application['AMT_INCOME_TOTAL'] < q_hi)]
```

```
# CNT_FAM_MEMBERS
q_hi = application['CNT_FAM_MEMBERS'].quantile(0.999)
q_low = application['CNT_FAM_MEMBERS'].quantile(0.001)
application = application[(application['CNT_FAM_MEMBERS'] > q_low) &
(application['CNT_FAM_MEMBERS'] < q_hi)]
```

5. Encoding

Encoding Categorical features.

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

object_features = application.select_dtypes(include='object').columns.tolist()
object_features
```

```
['CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'OCCUPATION_TYPE']
```

```
for feature in object_features:
    application[feature] = le.fit_transform(application[feature])
```

```
application
```

```
             ID  CODE_GENDER  FLAG_OWN_CAR  FLAG_OWN_REALTY  CNT_CHILDREN
AMT_INCOME_TOTAL  NAME_INCOME_TYPE  NAME_EDUCATION_TYPE  NAME_FAMILY_STATUS
NAME_HOUSING_TYPE  DAYS_BIRTH  DAYS_EMPLOYED  FLAG_MOBIL  FLAG_WORK_PHONE
FLAG_PHONE  FLAG_EMAIL  OCCUPATION_TYPE  CNT_FAM_MEMBERS
29      5008838            1             0                1             1
```

```
405000.0                 0                     1                    1
1      -11842          -2016          1                0            0            0
10             3.0
30     5008839           1             0                1             1
405000.0                 0                     1                    1
1      -11842          -2016          1                0            0            0
10             3.0
31     5008840           1             0                1             1
405000.0                 0                     1                    1
1      -11842          -2016          1                0            0            0
10             3.0
32     5008841           1             0                1             1
405000.0                 0                     1                    1
1      -11842          -2016          1                0            0            0
10             3.0
33     5008842           1             0                1             1
405000.0                 0                     1                    1
1      -11842          -2016          1                0            0            0
10             3.0
...         ...          ...           ...              ...           ...
...               ...                  ...                ...                ...
...               ...          ...              ...          ...          ...
...               ...
438536  6837264          0             0                0             2
90000.0                 2                     1                    3
1      -16062          -1275          1                0            0            0
3             4.0
438539  6837454          1             1                1             1
162000.0                 2                     4                    1
1      -10890          -2675          1                0            0            0
3             3.0
438542  6837905          1             1                1             1
355050.0                 4                     4                    1
1      -15904          -2614          1                0            0            0
8             3.0
438543  6837906          1             1                1             1
355050.0                 4                     4                    1
1      -15904          -2614          1                0            0            0
8             3.0
438548  6839936          1             1                1             1
135000.0                 4                     4                    1
1      -12569          -2095          1                0            0            0
8             3.0

[114188 rows x 18 columns]
```

-> Credit Dataset

- 0: 1-29 days past due
- 1: 30-59 days past due
- 2: 60-89 days overdue
- 3: 90-119 days overdue
- 4: 120-149 days overdue

- 5: Overdue or bad debts, write-offs for more than 150 days
- C: paid off that month
- X: No loan for the month

```
credit_record
```

```
              ID  MONTHS_BALANCE STATUS
0        5001711               0      X
1        5001711              -1      0
2        5001711              -2      0
3        5001711              -3      0
4        5001712               0      C
...          ...             ...    ...
1048570  5150487             -25      C
1048571  5150487             -26      C
1048572  5150487             -27      C
1048573  5150487             -28      C
1048574  5150487             -29      C

[1048575 rows x 3 columns]
```

It appears that the credit record dataset are set in a format which need to be grouped by the 'ID'.

The approach is as of following:

- Each 'ID' keeps the latest month (max(MONTHS_BALANCE)).
- Each 'ID' keeps the worst (highest) status after transformation.
- If any month had 'X' or 'C' (converted to 1), the final status is 1 (Good Client).
- If the user had only overdue payments, the final status is 0 (Bad Client).

Basically, we are hunting the bad clients, if 'STATUS' >= 2 represents serious overdue payments, this transformation marks those customers with 1 (high risk). If 'STATUS' < 2 means acceptable risk or good clients, they are marked with 0.

```
credit_record['STATUS'].replace({'C': 0, 'X' : 0}, inplace=True)
credit_record['STATUS'] = credit_record['STATUS'].astype('int')
credit_record['STATUS'] = credit_record['STATUS'].apply(lambda x:1 if x >= 2 else 0)

credit_record = credit_record.groupby('ID').agg(max).reset_index()

credit_record.drop('MONTHS_BALANCE', axis=1, inplace=True)

credit_record.head()
```

```
        ID  STATUS
0  5001711       0
1  5001712       0
2  5001713       0
3  5001714       0
4  5001715       0
```

Now it's fixed.

It appears that a new problem arises and that is the imbalance of target value samples.

```
credit_record['STATUS'].value_counts(normalize=True)
```

```
STATUS
0    0.985495
1    0.014505
Name: proportion, dtype: float64
```

We will address this problem in the Model Building section.

6. Merging

```
df = pd.merge(application, credit_record, on='ID', how='inner')
```

```
df.head()
```

```
        ID  CODE_GENDER  FLAG_OWN_CAR  FLAG_OWN_REALTY  CNT_CHILDREN
AMT_INCOME_TOTAL  NAME_INCOME_TYPE  NAME_EDUCATION_TYPE  NAME_FAMILY_STATUS
NAME_HOUSING_TYPE  DAYS_BIRTH  DAYS_EMPLOYED  FLAG_MOBIL  FLAG_WORK_PHONE
FLAG_PHONE  FLAG_EMAIL  OCCUPATION_TYPE  CNT_FAM_MEMBERS  STATUS
0  5008838           1             0                1             1
405000.0                 0                  1                  1
1       -11842          -2016          1               0          0            0
10              3.0       0
1  5008839           1             0                1             1
405000.0                 0                  1                  1
1       -11842          -2016          1               0          0            0
10              3.0       0
2  5008840           1             0                1             1
405000.0                 0                  1                  1
1       -11842          -2016          1               0          0            0
10              3.0       0
3  5008841           1             0                1             1
405000.0                 0                  1                  1
1       -11842          -2016          1               0          0            0
10              3.0       0
4  5008842           1             0                1             1
405000.0                 0                  1                  1
1       -11842          -2016          1               0          0            0
10              3.0       0
```

```
df.shape
```

```
(9516, 19)
```

There are 9516 rows ready for deployment.

Model Building

Splitting the data into X & y and further into train & test.

```
X = df.iloc[:,1:-1]
y = df.iloc[:,-1]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3)
```

Scaling the training data.

```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Addressing the oversampling problem by evening the target samples using SMOTE.

```python
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X_train, y_train = oversample.fit_resample(X_train, y_train)
```

Model Building:

1. Decision Tree

```python
from sklearn.tree import DecisionTreeClassifier

clf_entropy = DecisionTreeClassifier(
    criterion="entropy", random_state=100,
    max_depth=3, min_samples_leaf=5)
model_entropy = clf_entropy.fit(X_train, y_train)
prediction = model_entropy.predict(X_test)
```

```python
from sklearn.metrics import classification_report, accuracy_score
print(f"Accuracy: {accuracy_score(y_test, prediction) * 100:.2f}%")
print(classification_report(y_test, prediction))
```

```
Accuracy: 49.81%
              precision    recall  f1-score   support

           0       0.98      0.50      0.66      2809
           1       0.02      0.50      0.03        46

    accuracy                           0.50      2855
   macro avg       0.50      0.50      0.35      2855
weighted avg       0.97      0.50      0.65      2855
```

2. Random Forest

```python
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(
    n_estimators=100, random_state=100,
    max_depth=3, min_samples_leaf=5)
model = clf.fit(X_train, y_train)
prediction = model.predict(X_test)
```

```python
print(f"Accuracy: {accuracy_score(y_test, prediction) * 100:.2f}%")
print(classification_report(y_test, prediction))
```

```
Accuracy: 80.11%
              precision    recall  f1-score   support

           0       0.99      0.81      0.89      2809
           1       0.03      0.37      0.06        46
```

```
    accuracy                           0.80      2855
   macro avg       0.51      0.59      0.47      2855
weighted avg       0.97      0.80      0.88      2855
```

3. KNN

```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
prediction = knn.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, prediction) * 100:.2f}%")
print(classification_report(y_test, prediction))
```

```
Accuracy: 96.60%
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      2809
           1       0.23      0.46      0.30        46

    accuracy                           0.97      2855
   macro avg       0.61      0.72      0.64      2855
weighted avg       0.98      0.97      0.97      2855
```

4. XgBoost

```python
from xgboost import XGBClassifier
xgb = XGBClassifier()
model = xgb.fit(X_train, y_train)
prediction = xgb.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, prediction) * 100:.2f}%")
print(classification_report(y_test, prediction))
```

```
Accuracy: 98.21%
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2809
           1       0.42      0.28      0.34        46

    accuracy                           0.98      2855
   macro avg       0.70      0.64      0.66      2855
weighted avg       0.98      0.98      0.98      2855
```

5. SVM

```python
from sklearn.svm import SVC
svm = SVC(kernel='linear', C=1, random_state=100)
svm.fit(X_train, y_train)
prediction = svm.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, prediction) * 100:.2f}%")
print(classification_report(y_test, prediction))
```

```
Accuracy: 62.31%
              precision    recall  f1-score   support

           0       0.99      0.62      0.77      2809
           1       0.02      0.54      0.04        46

    accuracy                           0.62      2855
   macro avg       0.51      0.58      0.40      2855
weighted avg       0.97      0.62      0.75      2855
```

6. Neural Network

```python
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=1000,
random_state=100)
mlp.fit(X_train, y_train)
prediction = mlp.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, prediction) * 100:.2f}%")
print(classification_report(y_test, prediction))
```

```
Accuracy: 96.71%
              precision    recall  f1-score   support

           0       0.99      0.98      0.98      2809
           1       0.22      0.41      0.29        46

    accuracy                           0.97      2855
   macro avg       0.61      0.69      0.64      2855
weighted avg       0.98      0.97      0.97      2855
```