Output and code:

```
#group:dania_mohammed
#      rinat_imad
#data:Loan Prediction Based on Customer Behavior
print("--------------------------------------------------------------------------
----------")
print("group:daniahilal 201911486 ")
print("      rinat imad 201911443")
print("data:Loan Prediction Based on Customer Behavior")
print("--------------------------------------------------------------------------
----------")
print("--------------------------------------------------------------------------
----------")


import imp
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

#loading my dataset
data=pd.read_csv(r"C:\Users\L\Desktop\sem3\data mining\مجلد جديد\Training Data.csv")

# missing handling by deletting
print("data before handling missing index",len(data))
data = data.dropna(axis=0,how='any')
print("data after handling missing index",len(data))
print("--------------------------------------------------------------------------
----------")


#remove noise using z-score
from scipy import stats
z_scores = stats.zscore(data)
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3).all(axis=1)
new_df = data[filtered_entries]
print('data after remove noise')
print(new_df)
print("--------------------------------------------------------------------------
----------")
```

```
#remove duplicated
dups = data.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
print("data before removing duplicated index",len(data))
data=data.drop_duplicates()
print("data after removing duplicated index",len(data))
print("--------------------------------------------------------------------------------
----------")



#Remove irrelevant attributes depend on relation with risk flag attribute
#first i Convert Categorical Variables to Numerical
#get all categorical columns
#numerical dataset
nudata=data
cat_columns = data.select_dtypes(['object']).columns
#convert all categorical columns to numeric
nudata[cat_columns] = nudata[cat_columns].apply(lambda x: pd.factorize(x)[0])
cor = nudata.corr()
cor_target = abs(cor["Risk_Flag"])
#detect features with less depend on risk flag"my class"
irrelevant_features = cor_target [cor_target < 0.01]
print ("#of irrr features",len(irrelevant_features))
print("irrr features",irrelevant_features.index)
#delete irrlevent
nudata=data.drop(labels=irrelevant_features.index, axis=1)
data=data.drop(labels=irrelevant_features.index, axis=1)
print ("columns after drop irrelevent ones",data.columns)
print("--------------------------------------------------------------------------------
----------")


#Remove correlated attributes.
cat_columns = data.select_dtypes(['object']).columns
#convert all categorical columns to numeric
nudata[cat_columns] = nudata[cat_columns].apply(lambda x: pd.factorize(x)[0])
print("befor drop corr",nudata.shape,nudata.columns)

correlated_features = set()
correlation_matrix = nudata.corr()
```

```
print(nudata.corr())
for i in range(len(correlation_matrix .columns)):
    for j in range(i) :
        if abs(correlation_matrix.iloc[i, j]) > 0.8:
            colname = correlation_matrix.columns[i]
            correlated_features.add(colname)
print("#of dependent featurs ",len(correlated_features))
nudata.drop(columns=correlated_features, axis=1, inplace=True)
data.drop(columns=correlated_features, axis=1, inplace=True)
print("data after corrolation drop",data.columns)
print("---------------------------------------------------------------------------
----------")
```

```
--------------------------------------------------------------------------------
group:daniahilal 201911486
      rinat imad 201911443
data:Loan Prediction Based on Customer Behavior
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
data before handling missing index 252000
data after handling missing index 252000
--------------------------------------------------------------------------------
Number of duplicate rows = 0
data before removing duplicated index 252000
data after removing duplicated index 252000
--------------------------------------------------------------------------------
#of irrr features 5
irrr features Index(['Income', 'Profession', 'CITY', 'STATE', 'CURRENT_HOUSE_YRS'], dtype='object')
columns after drop irrelevent ones Index(['Id', 'Age', 'Experience', 'Married/Single', 'House_Ownership',
       'Car_Ownership', 'CURRENT_JOB_YRS', 'Risk_Flag'],
      dtype='object')
--------------------------------------------------------------------------------
befor drop corr (252000, 8) Index(['Id', 'Age', 'Experience', 'Married/Single', 'House_Ownership',
       'Car_Ownership', 'CURRENT_JOB_YRS', 'Risk_Flag'],
      dtype='object')
                      Id       Age  Experience  Married/Single  \
Id              1.000000 -0.001816   -0.005810       -0.001134
Age            -0.001816  1.000000   -0.001118        0.005323
Experience     -0.005810 -0.001118    1.000000        0.001752
Married/Single -0.001134  0.005323    0.001752        1.000000
```

```
#discretization on numeric attributes
print (data)
num_colums = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
numerical_columns = list(data.select_dtypes(include=num_colums).columns)

for i in range(len(data[numerical_columns].columns)-1):
     bins = pd.qcut(data[numerical_columns[i+1]],2000,duplicates = 'drop')
     bins.value_counts(sort=False)
     print("data after discretization",bins.value_counts(sort=False))
print("---------------------------------------------------------------------------
----------")
```

```
#split the data into train 80% and test set20%
train,test = train_test_split(data, test_size=0.20, random_state=0)
```

```python
#save the data ino separate files
train.to_csv('training.csv',index=False)
test.to_csv('testing.csv',index=False)
print("finished first part")
print("----------------------------------------------------------------------------------")
print("----------------------------------------------------------------------------------")
```

```
befor drop corr (252000, 8) Index(['Id', 'Age', 'Experience', 'Married/Single', 'House_Ownership',
       'Car_Ownership', 'CURRENT_JOB_YRS', 'Risk_Flag'],
      dtype='object')
                       Id       Age  Experience  Married/Single  \
Id               1.000000 -0.001816   -0.005810       -0.001134
Age             -0.001816  1.000000   -0.001118        0.005323
Experience      -0.005810 -0.001118    1.000000        0.001752
Married/Single  -0.001134  0.005323    0.001752        1.000000
House_Ownership  0.002527  0.017551    0.013346       -0.026208
Car_Ownership   -0.004313  0.009395    0.007519       -0.001206
CURRENT_JOB_YRS -0.003250  0.002154    0.646098       -0.004251
Risk_Flag        0.032153 -0.021809   -0.034523       -0.021092

                 House_Ownership  Car_Ownership  CURRENT_JOB_YRS  Risk_Flag
Id                      0.002527      -0.004313        -0.003250   0.032153
Age                     0.017551       0.009395         0.002154  -0.021809
Experience              0.013346       0.007519         0.646098  -0.034523
Married/Single         -0.026208      -0.001206        -0.004251  -0.021092
House_Ownership         1.000000       0.002167         0.009390  -0.026661
Car_Ownership           0.002167       1.000000         0.011099  -0.024036
CURRENT_JOB_YRS         0.009390       0.011099         1.000000  -0.016942
Risk_Flag              -0.026661      -0.024036        -0.016942   1.000000
#of dependent featurs  0
data after corrolation drop Index(['Id', 'Age', 'Experience', 'Married/Single', 'House_Ownership',
       'Car_Ownership', 'CURRENT_JOB_YRS', 'Risk_Flag'],
      dtype='object')
------------------------------------------------------------------------------------
            Id  Age  Experience  Married/Single  House_Ownership  \
```

```
[252000 rows x 8 columns]
data after remove noise
            Id  Age  Experience  Married/Single  House_Ownership  \
0            1   23           3               0                0
1            2   40          10               0                0
2            3   66           4               1                0
3            4   41           2               0                0
4            5   47          11               0                0
...        ...  ...         ...             ...              ...
251995  251996   43          13               0                0
251996  251997   26          10               0                0
251997  251998   46           7               0                0
251998  251999   45           0               0                0
251999  252000   70          17               0                0

        Car_Ownership  CURRENT_JOB_YRS  Risk_Flag
0                   0                3          0
1                   0                9          0
2                   0                4          0
3                   1                2          1
4                   0                3          1
...               ...              ...        ...
251995              0                6          0
251996              0                6          0
251997              0                7          0
251998              0                0          0
```

```
data after discretization (20.999, 22.0]    8323
(22.0, 23.0]      4140
(23.0, 24.0]      4147
(24.0, 25.0]      4191
(25.0, 25.047]       0
(25.047, 26.0]    4017
(26.0, 27.0]      4920
(27.0, 28.0]      3801
(28.0, 29.0]      3959
(29.0, 30.0]      4152
(30.0, 31.0]      4079
(31.0, 32.0]      3576
(32.0, 33.0]      4740
(33.0, 34.0]      4135
(34.0, 35.0]      4501
(35.0, 36.0]      3256
(36.0, 37.0]      4022
(37.0, 38.0]      3620
(38.0, 39.0]      3546
(39.0, 40.0]      4018
(40.0, 41.0]      4586
(41.0, 42.0]      3849
(42.0, 43.0]      3916
(43.0, 44.0]      3479
(44.0, 45.0]      4632
(45.0, 46.0]      3481
(46.0, 47.0]      4293
(47.0, 48.0]      4713
```

```
(71.0, 72.0]      4244
(72.0, 73.0]      3238
(73.0, 74.0]      3825
(74.0, 75.0]      3479
(75.0, 76.0]      4264
(76.0, 77.0]      3710
(77.0, 78.0]      4137
(78.0, 79.0]      4300
Name: Age, dtype: int64
data after discretization (-0.001, 1.0]    21838
(1.0, 2.0]       10468
(2.0, 3.0]       11308
(3.0, 4.0]       11149
(4.0, 5.0]       11757
(5.0, 6.0]       12532
(6.0, 7.0]       10853
(7.0, 8.0]       10937
(8.0, 9.0]       12407
(9.0, 10.0]      11755
(10.0, 11.0]     10814
(11.0, 12.0]     11956
(12.0, 13.0]     11064
(13.0, 14.0]     11546
(14.0, 15.0]     11515
(15.0, 16.0]     11322
(16.0, 17.0]     11529
(17.0, 18.0]     11871
(18.0, 19.0]     11654
```

```
(16.0, 17.0]     11529
(17.0, 18.0]     11871
(18.0, 19.0]     11654
(19.0, 20.0]     10807
Name: Experience, dtype: int64
data after discretization (-0.001, 1.0]    239082
Name: Married/Single, dtype: int64
data after discretization (-0.001, 1.0]    239082
Name: House_Ownership, dtype: int64
data after discretization (-0.001, 1.0]    239082
Name: Car_Ownership, dtype: int64
data after discretization (-0.001, 1.0]    21838
(1.0, 2.0]       10468
(2.0, 3.0]       27379
(3.0, 4.0]       26623
(4.0, 5.0]       24069
(5.0, 6.0]       22339
(6.0, 7.0]       19252
(7.0, 8.0]       18037
(8.0, 9.0]       16488
(9.0, 10.0]      14837
(10.0, 11.0]     12678
(11.0, 12.0]      9955
(12.0, 13.0]      8716
(13.0, 14.0]      6403
Name: CURRENT_JOB_YRS, dtype: int64
data after discretization (-0.001, 1.0]    239082
Name: Risk_Flag, dtype: int64
```

```
(4.0, 5.0]       24069
(5.0, 6.0]       22339
(6.0, 7.0]       19252
(7.0, 8.0]       18037
(8.0, 9.0]       16488
(9.0, 10.0]      14837
(10.0, 11.0]     12678
(11.0, 12.0]      9955
(12.0, 13.0]      8716
(13.0, 14.0]      6403
Name: CURRENT_JOB_YRS, dtype: int64
data after discretization (-0.001, 1.0]    239082
Name: Risk_Flag, dtype: int64
---------------------------------------------------------------------------------
finished first part
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
```

```
#classification
# KNN Model classification algorithm 1
accuracies = {}
import matplotlib.pyplot as plt

data[cat_columns] = data[cat_columns].apply(lambda x: pd.factorize(x)[0])

y = data.Risk_Flag
x= data.drop(['Risk_Flag'], axis = 1)
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size =
0.2,random_state=0)
#transpose matrices
x_train = x_train.T
y_train = y_train.T
x_test = x_test.T
y_test = y_test.T
# KNN Model
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 2)  # n_neighbors means k
knn.fit(x_train.T, y_train.T)
prediction = knn.predict(x_test.T)

print("{} NN Score: {:.2f}%".format(2, knn.score(x_test.T, y_test.T)*100))


# try ro find best k value
scoreList = []
for i in range(1,20):
    knn2 = KNeighborsClassifier(n_neighbors = i)  # n_neighbors means k
    knn2.fit(x_train.T, y_train.T)
```

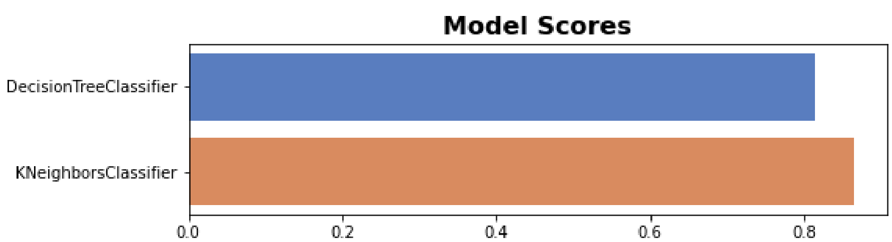
```python
    scoreList.append(knn2.score(x_test.T, y_test.T))


acc = max(scoreList)*100
accuracies['KNN'] = acc
print("Maximum KNN Score is {:.2f}%".format(acc))

print("--------------------")


#Naive Bayes classsification algorithm 2
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train.T, y_train.T)

acc = nb.score(x_test.T,y_test.T)*100
accuracies['Naive Bayes'] = acc
print("Accuracy of Naive Bayes: {:.2f}%".format(acc))


print("--------------------------------------------------------------------------
----------")
print("--------------------------------------------------------------------------
----------")
```

```
2 NN Score: 85.98%
Maximum KNN Score is 87.25%
--------------------
Accuracy of Naive Bayes: 87.26%
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
```

```python
#comparing
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25)
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score

from sklearn.metrics import accuracy_score
```

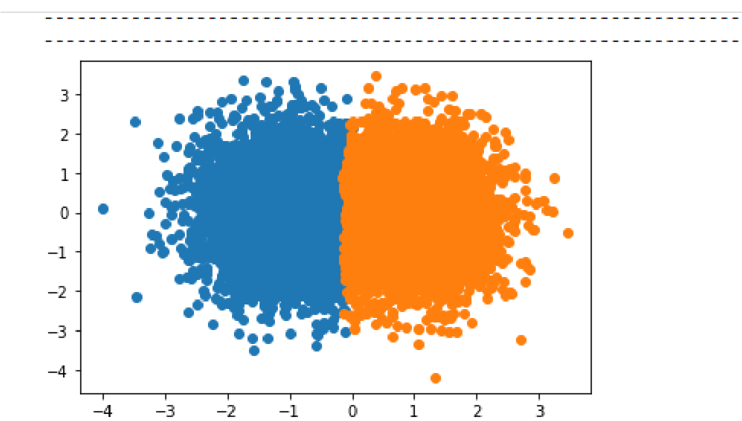
```
scaler = StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.transform(x_test)
dtc = DecisionTreeClassifier()
knc = KNeighborsClassifier()

keys = ['DecisionTreeClassifier','KNeighborsClassifier']
values = [DecisionTreeClassifier(), KNeighborsClassifier()]
scores = []
for value in values:
    model=value
    model.fit(x_train,y_train)
    predict = model.predict(x_test)
    acc = accuracy_score(y_test, predict)
    scores.append(acc)
plt.figure(figsize = (8,2))
sns.barplot(x = scores, y = keys, palette='muted')
plt.title("Model Scores", fontsize=16, fontweight="bold")


print("--------------------------------------------------------------------------
----------")
print("--------------------------------------------------------------------------
----------")
```

```
------------------------------------------------------------------------------
]: Text(0.5, 1.0, 'Model Scores')
```




```
#clustering
# k-means clustering algorithm 1
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans
```

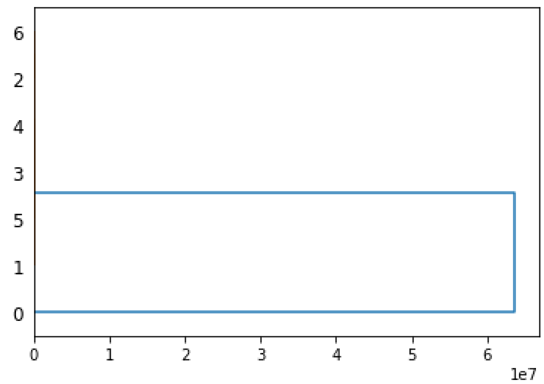
```
from matplotlib import pyplot
# define dataset
X, _ = make_classification(n_samples=10000, n_features=6, n_informative=2,
n_redundant=0, n_clusters_per_class=1, random_state=4)
# define the model
model = KMeans(n_clusters=2)
# fit the model
model.fit(X)
# assign a cluster to each example
yhat = model.predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
    # get row indexes for samples with this cluster
    row_ix = where(yhat == cluster)
    # create scatter of these samples
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()

print("----------------------------")
```




```
#Single Link (MIN) clustering algorithm 2
from scipy.cluster import hierarchy
import matplotlib.pyplot as plt
Z = hierarchy.linkage(x_train.to_numpy(), 'single')
dn = hierarchy.dendrogram(Z,orientation='right')


print("-----------------------------------------------------------------------------
----------")
```

```
print("-----------------------------------------------------------------------
----------")
```

```
------------------------------------------------------------
------------------------------------------------------------
```




```
# association rules
# get association rules by using FP-growth algorithm 1
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from mlxtend.preprocessing import TransactionEncoder


te = TransactionEncoder()

#Transform the transection dataset to binary 20 array

te_ary= te.fit(data).transform(data)

print(te_ary)
#covert the array of transaction data array into pandas DataFrame

df = pd.DataFrame(te_ary)

#get the frequent itemsets by using apriori algorithm

frequentItemsets = apriori(df, min_support=0.6, use_colnames=True)
print('Itemsets\n', frequentItemsets)

# get the association rules-

from mlxtend.frequent_patterns import association_rules
```

```python
rules= association_rules (frequentItemsets, min_threshold=0.7)
print('Rules\n', rules)


print("-------------------------------------------------------------------------
----------")
print("-------------------------------------------------------------------------
----------")

# get association rules algorithm 2
import pyfpgrowth
#use FP-growth to get patterns with minimum support = 3
patterns= pyfpgrowth.find_frequent_patterns(data,3)

#use FP-growth to get association rules with minimum confidence = 0.7

rules = pyfpgrowth.generate_association_rules(patterns, 0.7)
print("Rules\n", rules)

print("-------------------------------------------------------------------------
----------")
print("-------------------------------------------------------------------------
----------")
```

```
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
[[False False False ... False False False]
 [False  True False ... False False False]
 [False False False ... False False  True]
 ...
 [False False False ... False False False]
 [False False False ... False False False]
 [False False False ... False False False]]
Itemsets
 Empty DataFrame
Columns: [support, itemsets]
Index: []
```