## Classification

6 : Predict Bank Credit Risk : Project Title Predict Bank Credit Risk using South German Credit Data Domain Banking

Problem Statement: Normally, most of the bank's wealth is obtained from providing credit loans so that a marketing bank must be able to reduce the risk of non-performing credit loans. The risk of providing loans can be minimized by studying patterns from existing lending data. One technique that you can use to solve this problem is to use data mining techniques. Data mining makes it possible to find hidden information from large data sets by way of classification. The goal of this project, you have to build a model to predict whether the person, described by the attributes of the dataset, is a good (1) or a bad (0) credit risk

Dataset Link : https://www.kaggle.com/competitions/south-german-credit-prediction/data (https://www.kaggle.com/competitions/south-german-credit-prediction/data)

Dataset Fields

The below list consists of a detailed breakdown of all the features in the Dataset. 'Kredit' is our target variable, the one whose value must be predicted. P.S. The feature names are in German to preserve the authenticity of the data. In the dataset, they can be replaced using the DataDescription.csv file and the Pandas library.

```
Id = Id of individual entries, for evaluation

laufkont = status
1 : no checking account
2 : … < 0 DM 3 : 0<= … < 200 DM 4 : … >= 200
DM / salary for at least 1 year

laufzeit = duration

moral = credit_history
0 : delay in paying off in the past
1 : critical account/other credits elsewhere
2 : no credits taken/all credits paid back du
ly
3 : existing credits paid back duly till now
4 : all credits at this bank paid back duly

verw = purpose
0 : others
1 : car (new)
2 : car (used)
3 : furniture/equipment
4 : radio/television
5 : domestic appliances
6 : repairs
7 : education
8 : vacation
9 : retraining
10 : business

hoehe = amount

sparkont = savings
1 : unknown/no savings account
```

```
2 : … < 100 DM 3 : 100 <= … < 500 DM 4 : 500
<= … < 1000 DM 5 : … >= 1000 DM

beszeit = employment_duration
1 : unemployed
2 : < 1 yr 3 : 1 <= … < 4 yrs 4 : 4 <= … < 7
yrs 5 : >= 7 yrs

rate = installment_rate
1 : >= 35
2 : 25 <= … < 35
3 : 20 <= … < 25
4 : < 20

famges = personal_status_sex
1 : male : divorced/separated
2 : female : non-single or male : single
3 : male : married/widowed
4 : female : single

buerge = other_debtors
1 : none
2 : co-applicant
3 : guarantor

wohnzeit = present_residence
1 : < 1 yr 2 : 1 <= … < 4 yrs 3 : 4 <= … < 7
yrs 4 : >= 7 yrs

verm = property
1 : unknown / no property
2 : car or other
3 : building soc. savings agr./life insurance
4 : real estate
```

```
alter = age

weitkred = other_installment_plans
1 : bank
2 : stores
3 : none

wohn = housing
1 : for free
2 : rent
3 : own

bishkred = number_credits
1 : 1
2 : 2-3
3 : 4-5
4 : >= 6

beruf = job
1 : unemployed/unskilled - non-resident
2 : unskilled - resident
3 : skilled employee/official
4 : manager/self-empl./highly qualif. employe
e

pers = people_liable
1 : 3 or more
2 : 0 to 2

telef = telephone
1 : no
2 : yes (under customer name)

gastarb = foreign worker
```

In [2]:

```
# importing Libraries
# importing Pandas Library as pd
import pandas as pd

# importing Numpy Library as np
import numpy as np

# importing matplotlib.pyplot as plt
import matplotlib.pyplot as plt

# imporing seaborn as sns
import seaborn as sns
```

In [3]:

```
# Loading the dataset using pandas module and ass
df = pd.read_csv('bankcredit.csv')

# Printing the dataset
df
```

Out[3]:

| | Id | laufkont | laufzeit | moral | verw | hoehe | spark |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 18 | 4 | 2 | 1049 | |
| **1** | 1 | 1 | 9 | 4 | 0 | 2799 | |
| **2** | 2 | 2 | 12 | 2 | 9 | 841 | |
| **3** | 3 | 1 | 12 | 4 | 0 | 2122 | |
| **4** | 5 | 1 | 10 | 4 | 0 | 2241 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **795** | 993 | 1 | 18 | 4 | 0 | 3966 | |
| **796** | 994 | 1 | 12 | 0 | 3 | 6199 | |
| **797** | 997 | 4 | 21 | 4 | 0 | 12680 | |

| | Id | laufkont | laufzeit | moral | verw | hoehe | spark |
|---|---|---|---|---|---|---|---|
| **798** | 998 | 2 | 12 | 2 | 3 | 6468 | |
| **799** | 999 | 1 | 30 | 2 | 2 | 6350 | |

In [4]:

```
df.columns = ['Id','status', 'duration', 'credit_
```

In [5]:

```
# By using tails we are getting last 5 values
df.tail(10)
```

Out[5]:

| | Id | status | duration | credit_history | purpose | am |
|---|---|---|---|---|---|---|
| **790** | 987 | 1 | 12 | 2 | 3 | |
| **791** | 989 | 2 | 24 | 2 | 0 | |
| **792** | 990 | 1 | 18 | 2 | 6 | |
| **793** | 991 | 2 | 24 | 2 | 1 | 1 |
| **794** | 992 | 1 | 18 | 2 | 1 | |
| **795** | 993 | 1 | 18 | 4 | 0 | |
| **796** | 994 | 1 | 12 | 0 | 3 | |
| **797** | 997 | 4 | 21 | 4 | 0 | 1 |
| **798** | 998 | 2 | 12 | 2 | 3 | |
| **799** | 999 | 1 | 30 | 2 | 2 | |

10 rows × 22 columns

In [6]:

```
# Information about the Dataset:
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 22 columns):
 #   Column                   Non-Null C
ount  Dtype
---  ------
--------------  -----
 0   Id                       800 non-nu
ll    int64
 1   status                   800 non-nu
ll    int64
 2   duration                 800 non-nu
ll    int64
 3   credit_history           800 non-nu
ll    int64
 4   purpose                  800 non-nu
ll    int64
 5   amount                   800 non-nu
ll    int64
 6   savings                  800 non-nu
ll    int64
 7   employment_duration      800 non-nu
ll    int64
 8   installment_rate         800 non-nu
ll    int64
 9   personal_status_sex      800 non-nu
ll    int64
 10  other_debtors            800 non-nu
ll    int64
 11  present_residence        800 non-nu
ll    int64
 12  property                 800 non-nu
ll    int64
 13  age                      800 non-nu
ll    int64
 14  other_installment_plans  800 non-nu
ll    int64
 15  housing                  800 non-nu
ll    int64
 16  number_credits           800 non-nu
ll    int64
```

```
 17  job                    800 non-nu
ll    int64
 18  people_liable          800 non-nu
ll    int64
 19  telephone              800 non-nu
ll    int64
 20  foreign_worker         800 non-nu
ll    int64
 21  credit_risk            800 non-nu
ll    int64
dtypes: int64(22)
memory usage: 137.6 KB
```

In [7]:

```
# isnull() method is used to check whether the da
# sum() is used to find count of null values
print(df.isnull().sum())

#getting size of the dataset(rows multiplied by c
print(df.size)

# getting rows and columnes in a dataset
print(df.shape)
```

```
Id                          0
status                      0
duration                    0
credit_history              0
purpose                     0
amount                      0
savings                     0
employment_duration         0
```

In [10]:

```
# getting all statctical values like mean,median,
# for numerical values
df.describe()
```

Out[10]:

| | Id | status | duration | credit_histo |
|---|---|---|---|---|
| **count** | 800.000000 | 800.000000 | 800.000000 | 800.0000 |
| **mean** | 478.101250 | 2.648750 | 20.496250 | 2.5825 |
| **std** | 278.883661 | 1.250931 | 12.006881 | 1.0998 |
| **min** | 0.000000 | 1.000000 | 4.000000 | 0.0000 |
| **25%** | 238.750000 | 1.750000 | 12.000000 | 2.0000 |
| **50%** | 472.000000 | 2.000000 | 18.000000 | 2.0000 |
| **75%** | 707.250000 | 4.000000 | 24.000000 | 4.0000 |
| **max** | 999.000000 | 4.000000 | 72.000000 | 4.0000 |

8 rows × 22 columns

In [11]:

```
df.columns
```

Out[11]:

```
Index(['Id', 'status', 'duration', 'cred
it_history', 'purpose', 'amount',
       'savings', 'employment_duration',
'installment_rate',
       'personal_status_sex', 'other_deb
tors', 'present_residence', 'property',
       'age', 'other_installment_plans',
'housing', 'number_credits', 'job'
```

In [12]:

```
df.drop(['Id'],inplace=True,axis=1)
```

In [53]:

```
df
```

Out[53]:

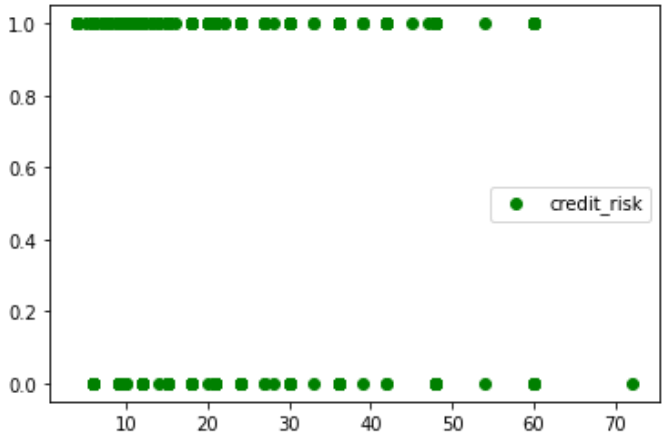
| | status | duration | credit_history | purpose | amount |
|---|---|---|---|---|---|
| **0** | 1 | 18 | 4 | 2 | 1049 |
| **1** | 1 | 9 | 4 | 0 | 2799 |
| **2** | 2 | 12 | 2 | 9 | 841 |
| **3** | 1 | 12 | 4 | 0 | 2122 |
| **4** | 1 | 10 | 4 | 0 | 2241 |
| **...** | ... | ... | ... | ... | ... |
| **795** | 1 | 18 | 4 | 0 | 3966 |
| **796** | 1 | 12 | 0 | 3 | 6199 |
| **797** | 4 | 21 | 4 | 0 | 12680 |
| **798** | 2 | 12 | 2 | 3 | 6468 |
| **799** | 1 | 30 | 2 | 2 | 6350 |

800 rows × 21 columns

## Data Visualization

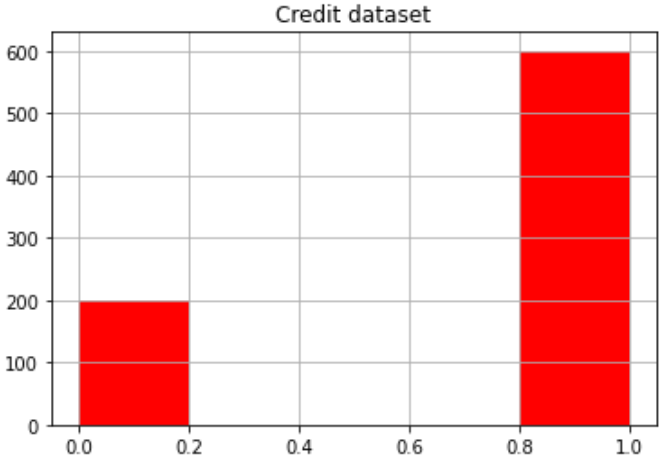
In [19]:

```
#Two column should be numerical
plt.scatter(x = 'duration',y = 'credit_risk',data

#Describe all the elements of a graph
plt.legend()

#show the graph
plt.show()
```



In [15]:

```
plt.hist(x = df["credit_risk"],color ='r',bins=5)
plt.title("Credit dataset")
# plot the grid:
plt.grid()
# display the plot:
plt.show()
```

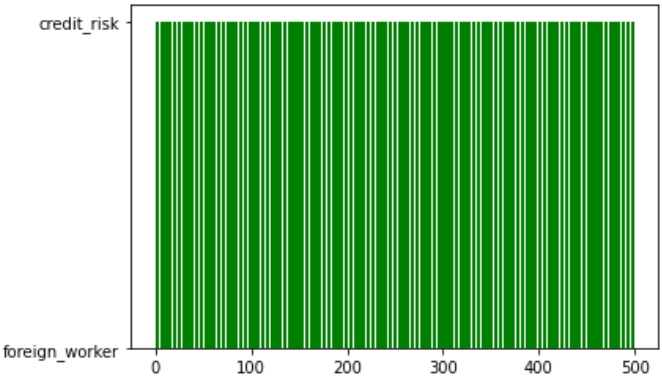


In [16]:

```
index = np.arange(500)
plt.bar(x = index,height = 'foreign_worker' ,colo
plt.bar(x = index,height = 'credit_risk',color =
```

Out[16]:

```
<BarContainer object of 500 artists>
```

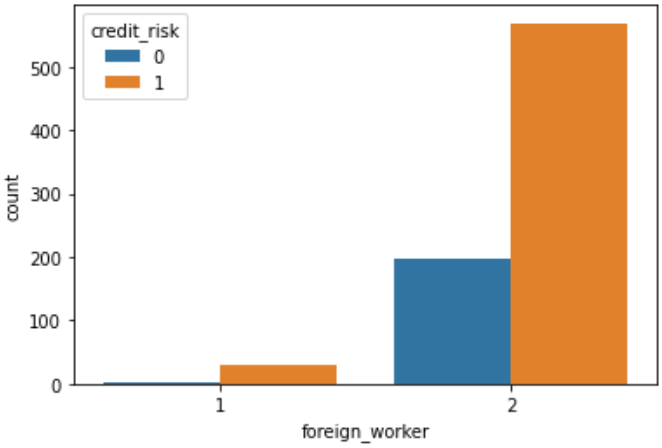


In [17]:

```
1 from warnings import filterwarnings
2 filterwarnings('ignore')
3 sns.countplot(df['foreign_worker'],hue=df['credit
```

Out[17]:

```
<AxesSubplot:xlabel='foreign_worker', yl
abel='count'>
```

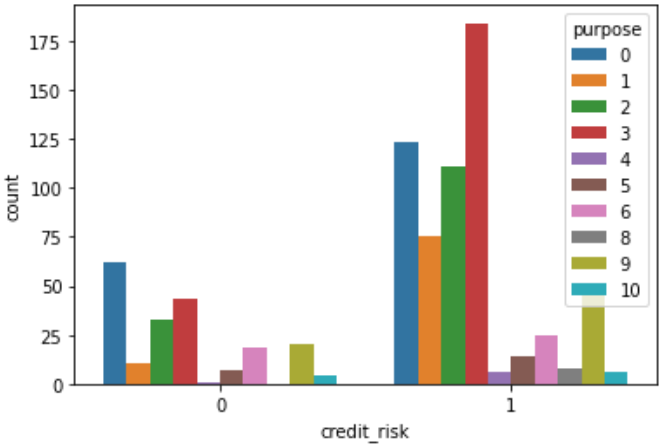


In [18]:

```
1 sns.countplot(df['credit_risk'],hue=df['purpose'])
```

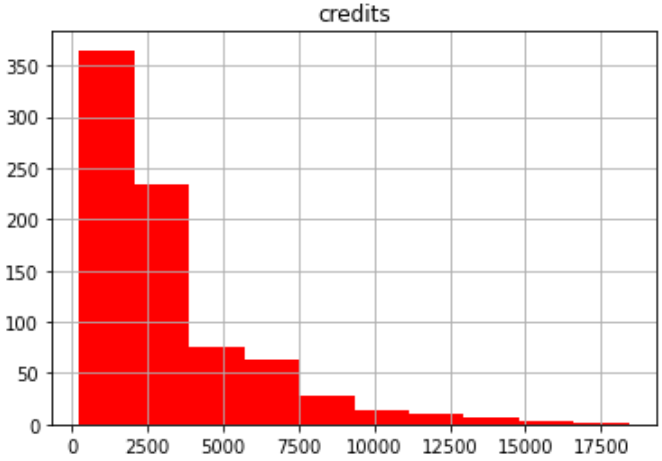
Out[18]:

```
<AxesSubplot:xlabel='credit_risk', ylabe
l='count'>
```



In [46]:

```
#Plot the Histogram with multiple bins  and add G

plt.hist(x = df['amount'],color = 'r')
plt.title('credits')

#Plot the Grid
plt.grid()
plt.show()
```




In [ ]:

```
1
```

In [20]:

```
# check the bad loans
df[df['credit_risk']==0]
```

Out[20]:

| | status | duration | credit_history | purpose | amount |
|---|---|---|---|---|---|
| **600** | 1 | 18 | 2 | 0 | 1216 |
| **601** | 4 | 18 | 4 | 6 | 1864 |

| | status | duration | credit_history | purpose | amount |
|---|---|---|---|---|---|
| **602** | 1 | 12 | 2 | 0 | 1228 |
| **603** | 2 | 12 | 2 | 0 | 685 |
| **604** | 3 | 9 | 2 | 3 | 745 |
| **...** | ... | ... | ... | ... | ... |
| **795** | 1 | 18 | 4 | 0 | 3966 |
| **796** | 1 | 12 | 0 | 3 | 6199 |
| **797** | 4 | 21 | 4 | 0 | 12680 |
| **798** | 2 | 12 | 2 | 3 | 6468 |
| **799** | 1 | 30 | 2 | 2 | 6350 |

In [21]:

```
1 # check the good loans
2 df[df['credit_risk']==1]
```

Out[21]:

| | status | duration | credit_history | purpose | amount |
|---|---|---|---|---|---|
| **0** | 1 | 18 | 4 | 2 | 1049 |
| **1** | 1 | 9 | 4 | 0 | 2799 |
| **2** | 2 | 12 | 2 | 9 | 841 |
| **3** | 1 | 12 | 4 | 0 | 2122 |
| **4** | 1 | 10 | 4 | 0 | 2241 |
| **...** | ... | ... | ... | ... | ... |
| **595** | 4 | 6 | 2 | 2 | 1766 |
| **596** | 2 | 24 | 2 | 1 | 2760 |
| **597** | 4 | 24 | 4 | 5 | 5507 |
| **598** | 3 | 24 | 2 | 2 | 2892 |
| **599** | 2 | 36 | 3 | 0 | 2862 |

600 rows × 21 columns

In [22]:

```
df["credit_risk"].value_counts()
```

Out[22]:

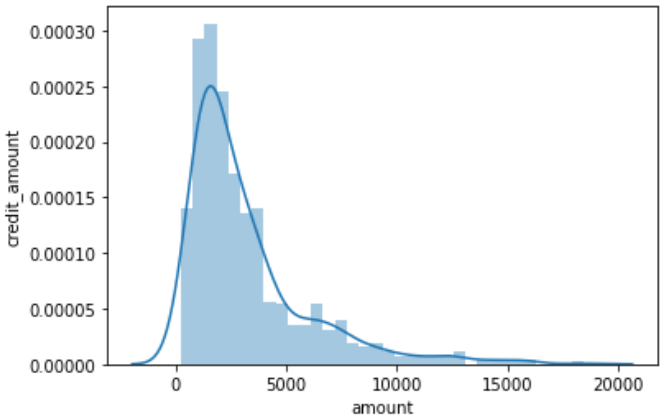
```
1    600
0    200
Name: credit_risk, dtype: int64
```

# Data Transformation

In [23]:

```
# Distribution of the Dispalcement column
sns.distplot(df['amount'])
plt.ylabel('credit_amount')
print('Skewness : ',df['amount'].skew())
plt.show()
```

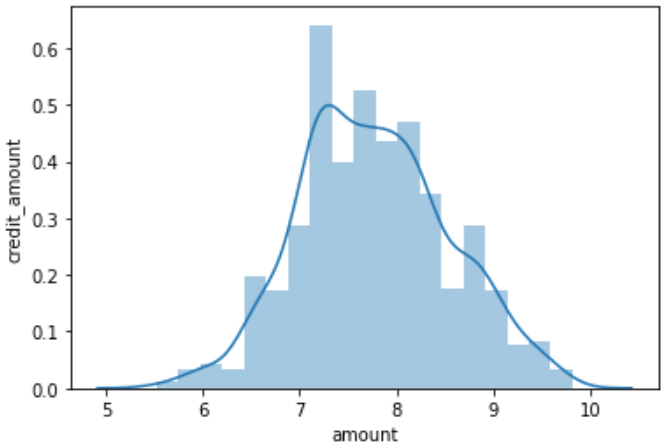
```
Skewness :  2.0472701844801806
```



In [24]:

```
# Apply natural log transformation for Displaceme
log_amount = np.log(df['amount'])
print('Skewness after log Transformtion : ',log_a

sns.distplot(log_amount)
plt.ylabel('credit_amount')
plt.show()
```

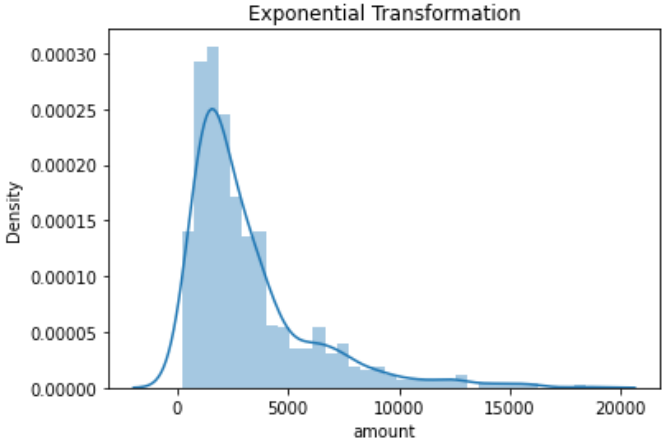
```
Skewness after log Transformtion :  0.13
83727959275712
```



In [25]:

```
# Anti - log or Exponential Transformation
displacement = np.exp(log_amount)
print('Skewness after log Transformtion : ',displ
# plot the Distribution
sns.distplot(displacement)
plt.ylabel('Density')
plt.title('Exponential Transformation')
plt.show()
```

```
Skewness after log Transformtion :  2.04
727018448018
```




In [26]:

```
df.drop(['age','duration'],axis=1,inplace=True)
df
```

Out[26]:

| | status | credit_history | purpose | amount | savings |
|---|---|---|---|---|---|
| **0** | 1 | 4 | 2 | 1049 | 1 |
| **1** | 1 | 4 | 0 | 2799 | 1 |
| **2** | 2 | 2 | 9 | 841 | 2 |

| | status | credit_history | purpose | amount | savings |
|---|---|---|---|---|---|
| **3** | 1 | 4 | 0 | 2122 | 1 |
| **4** | 1 | 4 | 0 | 2241 | 1 |
| **...** | ... | ... | ... | ... | ... |
| **795** | 1 | 4 | 0 | 3966 | 1 |
| **796** | 1 | 0 | 3 | 6199 | 1 |
| **797** | 4 | 4 | 0 | 12680 | 5 |
| **798** | 2 | 2 | 3 | 6468 | 5 |
| **799** | 1 | 2 | 2 | 6350 | 5 |

In [27]:

```
1 df.head()
```

Out[27]:

| | status | credit_history | purpose | amount | savings | ei |
|---|---|---|---|---|---|---|
| **0** | 1 | 4 | 2 | 1049 | 1 | |
| **1** | 1 | 4 | 0 | 2799 | 1 | |
| **2** | 2 | 2 | 9 | 841 | 2 | |
| **3** | 1 | 4 | 0 | 2122 | 1 | |
| **4** | 1 | 4 | 0 | 2241 | 1 | |

In [29]:

```
1 from sklearn.model_selection import train_test_sp
2 from sklearn.metrics import classification_report
```

In [30]:

```
1 xtrain, xtest, ytrain, ytest = train_test_split(d
```

In [31]:

```
1 xtrain.shape, xtest.shape
```

Out[31]:

```
((600, 18), (200, 18))
```

In [33]:

```
1 from sklearn.tree import DecisionTreeClassifier
2 from sklearn.tree import plot_tree
3 dtc = DecisionTreeClassifier()
```

In [34]:

```
1 dtc.fit(xtrain, ytrain)
```

Out[34]:

```
DecisionTreeClassifier()
```

In [35]:

```
1 plot_tree(dtc)
```

Out[35]:

```
[Text(220.88056640625, 211.4, 'X[0] <=
```

In [36]:

```
dtc.score(xtrain,ytrain)
```

Out[36]:

```
1.0
```

In [37]:

```
y_pred = dtc.predict(xtest)
```

In [38]:

```
print(classification_report(ytest, y_pred))
```

```
              precision    recall  f1-sc
ore   support

           0       0.42      0.52
0.46        44
           1       0.86      0.79
0.82       156

    accuracy
0.73       200
   macro avg       0.64      0.66
0.64       200
weighted avg       0.76      0.73
0.74       200
```

In [39]:

```
dtc.score(xtest, ytest)
```

Out[39]:

```
0.735
```

In [29]:

```
DecisionTreeClassifier?
```

In [41]:

```
from sklearn.model_selection import GridSearchCV
```

In [42]:

```
params = {
    'criterion' :  ["gini", "entropy", "log_loss"
    "splitter" : ["best", "random"],
    "max_features" : ["auto", "sqrt", "log2"],
    "min_samples_leaf" : [1,2,5,6]
}
```

In [43]:

```
grid = GridSearchCV(estimator = dtc, param_grid =
```

In [34]:

```
grid.fit(xtrain,ytrain)
```

/home/student/my_project_env/lib/python

In [44]:

```
dtc.score(xtrain,ytrain)
```

Out[44]:

1.0

In [45]:

```
dtc.score(xtest, ytest)
```

Out[45]:

0.735

In [46]:

```
dtc2 = DecisionTreeClassifier(
    criterion= 'gini',
    max_features = 'sqrt',
    splitter = 'random',
    min_samples_leaf = 6)
```

In [47]:

```
dtc2.fit(xtrain,ytrain)
```

Out[47]:

```
DecisionTreeClassifier(max_features='sqr
t', min_samples_leaf=6,
                       splitter='random
')
```

In [48]:

```
df.describe()
```

Out[48]:

| | status | credit_history | purpose | am |
|---|---|---|---|---|
| **count** | 800.000000 | 800.000000 | 800.000000 | 800.00 |

| | status | credit_history | purpose | am |
|---|---|---|---|---|
| **mean** | 2.648750 | 2.582500 | 2.785000 | 3210.29 |
| **std** | 1.250931 | 1.099866 | 2.680533 | 2792.84 |
| **min** | 1.000000 | 0.000000 | 0.000000 | 250.00 |
| **25%** | 1.750000 | 2.000000 | 1.000000 | 1364.00 |
| **50%** | 2.000000 | 2.000000 | 2.000000 | 2264.00 |
| **75%** | 4.000000 | 4.000000 | 3.000000 | 3907.25 |

In [53]:

```
from sklearn.preprocessing import MinMaxScaler
min = MinMaxScaler()
```

In [54]:

```
for column in df.columns:
  df[column] = min.fit_transform(df[[column]])
```

In [55]:

```
df.describe()
```

Out[55]:

| | status | credit_history | purpose | amou |
|---|---|---|---|---|
| **count** | 800.000000 | 800.000000 | 800.000000 | 800.0000 |
| **mean** | 0.549583 | 0.645625 | 0.278500 | 0.1628 |
| **std** | 0.416977 | 0.274967 | 0.268053 | 0.1536 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.0000 |
| **25%** | 0.250000 | 0.500000 | 0.100000 | 0.0612 |
| **50%** | 0.333333 | 0.500000 | 0.200000 | 0.1108 |
| **75%** | 1.000000 | 1.000000 | 0.300000 | 0.2012 |
| **max** | 1.000000 | 1.000000 | 1.000000 | 1.0000 |

In [60]:

```
from sklearn.preprocessing import StandardScaler
s = StandardScaler()
```

In [62]:

```
selected = pd.DataFrame(s.fit_transform(df.drop("
selected["ls"] = df["credit_risk"]
```

In [112]:

```
selected["ls"] = le.fit_transform(selected["ls"])
```

In [113]:

```
selected
```

Out[113]:

| | status | credit_history | purpose | amount | s |
|---|---|---|---|---|---|
| **0** | -1.318842 | 1.289599 | -0.278193 | -1.262349 | -0.7 |
| **1** | -1.318842 | 1.289599 | -1.099425 | 0.316359 | -0.7 |
| **2** | -0.518938 | -0.529941 | 2.185506 | -1.428529 | -0.( |
| **3** | -1.318842 | 1.289599 | -1.099425 | -0.140635 | -0.7 |
| **4** | -1.318842 | 1.289599 | -1.099425 | -0.052929 | -0.7 |
| **...** | ... | ... | ... | ... | |
| **795** | -1.318842 | 1.289599 | -1.099425 | 0.893372 | -0.7 |
| **796** | -1.318842 | -2.349482 | 0.132424 | 1.285741 | -0.7 |
| **797** | 1.080871 | 1.289599 | -1.099425 | 1.719655 | 1.7 |
| **798** | -0.518938 | -0.529941 | 0.132424 | 1.350367 | 1.7 |
| **799** | -1.318842 | -0.529941 | -0.278193 | 1.331902 | 1.7 |

800 rows × 19 columns

In [131]:

```
from sklearn.linear_model import LogisticRegressi
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
```

In [109]:

```
lr = LogisticRegression()
dtc = DecisionTreeClassifier()
sv = SVC()
gnb =GaussianNB()
```

In [110]:

```
g = GridSearchCV(estimator= dtc ,param_grid = {})
```

In [118]:

```
x = df.drop("credit_risk",axis=1)
y = df["credit_risk"]
```

In [119]:

```
svc = GaussianNB()
```

In [120]:

```
svc.fit(x,y)
```

Out[120]:

```
GaussianNB()
```

In [121]:

```
svc.score(x,y)
```

Out[121]:

```
0.76
```

In [122]:

```
1 x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 18 columns):
 #   Column                   Non-Null C
ount  Dtype
---  ------
--------------  -----
 0   status                   800 non-nu
ll    int64
 1   credit_history           800 non-nu
ll    int64
 2   purpose                  800 non-nu
ll    int64
 3   amount                   800 non-nu
ll    int64
 4   savings                  800 non-nu
ll    int64
 5   employment_duration      800 non-nu
ll    int64
 6   installment_rate         800 non-nu
ll    int64
 7   personal_status_sex      800 non-nu
ll    int64
 8   other_debtors            800 non-nu
ll    int64
 9   present_residence        800 non-nu
ll    int64
 10  property                 800 non-nu
ll    int64
 11  other_installment_plans  800 non-nu
ll    int64
 12  housing                  800 non-nu
ll    int64
 13  number_credits           800 non-nu
ll    int64
 14  job                      800 non-nu
ll    int64
```

```
 15  people_liable             800 non-nu
ll    int64
 16  telephone                 800 non-nu
ll    int64
 17  foreign_worker            800 non-nu
ll    int64
dtypes: int64(18)
memory usage: 112.6 KB
```

In [123]:

```
1 svc?
```

In [126]:

```
1 param_grid = [{'n_estimators': [3, 10, 30], 'max_
2               'learning_rate':[0.3,0.5,0.01,0.1]}
3
```

In [132]:

```
1 grid.fit(x,y)
```

Out[132]:

```
GridSearchCV(estimator=DecisionTreeClass
ifier(),
             param_grid={'criterion': ['
gini', 'entropy', 'log_loss'],
                         'max_features':
['auto', 'sqrt', 'log2'],
                         'min_samples_le
af': [1, 2, 5, 6],
                         'splitter': ['b
est', 'random']})
```

In [134]:

```
1 grid = GridSearchCV(estimator=svc, param_grid =
```

In [141]:

```
1 grid.best_params_
```

```
---------------------------------------
-----------------------------------
AttributeError
Traceback (most recent call last)
<ipython-input-141-28c2e4d7952c> in <mod
ule>
----> 1 grid.best_params_

AttributeError: 'GridSearchCV' object ha
s no attribute 'best_params_'
```

In [115]:

```
from sklearn.ensemble import RandomForestClassifi
rfc = RandomForestClassifier()
```

In [68]:

```
rfc.fit(xtrain,ytrain)

rfc.score(xtrain,ytrain)
```

Out[68]:

```
1.0
```

In [69]:

```
rfc?
```

In [142]:

```
pr = {
    'n_estimators' : [10,100,1000,600],
    'criterion':["log_loss", "entropy", "gini"]
}
```

In [143]:

```
g = GridSearchCV(estimator=rfc, param_grid = pr)
```

In [144]:

```
g.fit(x,y)
```

Out[144]:

```
GridSearchCV(estimator=RandomForestClass
ifier(),
             param_grid={'criterion': ['
log_loss', 'entropy', 'gini'],
                         'n_estimators':
[10, 100, 1000, 600]})
```

In [145]:

```
rfc2 = RandomForestClassifier(criterion = 'log_lo
```

In [148]:

```
rfc2.fit(x,y)

rfc2.score(x,y)
```

Out[148]:

```
1.0
```