Minor Project - Classification model to predict whether credit risk is good or bad. ¶

In [3]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Reading File

In [4]:

```
df= pd.read_csv('credit_customers (1).csv')
df.head()
```

Out[4]:

	Unnamed: 0	checking_status	duration	credit_history	purpose	credit_amount	sa
0	0	<0	6.0	critical/other existing credit	radio/tv	1169.0	
1	1	0<=X<200	48.0	existing paid	radio/tv	5951.0	
2	2	no checking	12.0	critical/other existing credit	education	2096.0	
3	3	<0	42.0	existing paid	furniture/equipment	7882.0	
4	4	<0	24.0	delayed previously	new car	4870.0	

5 rows × 22 columns

Replacing string values with int/float

In [6]:

```
df['class']=df['class'].replace('good',1)
df.to_csv('data.csv', index=False)
```

In [7]:

```
df['class']=df['class'].replace('bad',0)
df.to_csv('data.csv', index=False)
```

In [8]:

```
df['checking_status'].value_counts()
```

Out[8]:

```
no checking    394
<0             274
0<=X<200       269
>=200           63
Name: checking_status, dtype: int64
```

In [9]:

```
df['checking_status']=df['checking_status'].replace('no checking',0)
df.to_csv('data.csv', index=False)

df['checking_status']=df['checking_status'].replace('<0',1)
df.to_csv('data.csv', index=False)

df['checking_status']=df['checking_status'].replace('0<=X<200',2)
df.to_csv('data.csv', index=False)

df['checking_status']=df['checking_status'].replace('>=200',3)
df.to_csv('data.csv', index=False)
```

In [10]:

```
df['credit_history'].value_counts()
```

Out[10]:

```
existing paid                     530
critical/other existing credit    293
delayed previously                 88
all paid                           49
no credits/all paid                40
Name: credit_history, dtype: int64
```

In [11]:

```
df['credit_history']=df['credit_history'].replace('existing paid',0)
df.to_csv('data.csv', index=False)

df['credit_history']=df['credit_history'].replace('critical/other existing credit',1)
df.to_csv('data.csv', index=False)

df['credit_history']=df['credit_history'].replace('delayed previously',2)
df.to_csv('data.csv', index=False)

df['credit_history']=df['credit_history'].replace('all paid',3)
df.to_csv('data.csv', index=False)

df['credit_history']=df['credit_history'].replace('no credits/all paid',4)
df.to_csv('data.csv', index=False)
```

In [12]:

```
df['purpose'].value_counts()
```

Out[12]:

```
radio/tv               280
new car                234
furniture/equipment    181
used car               103
business                97
education               50
repairs                 22
domestic appliance      12
other                   12
retraining               9
Name: purpose, dtype: int64
```

In [13]:

```
df['purpose']=df['purpose'].replace('radio/tv',0)
df.to_csv('data.csv', index=False)

df['purpose']=df['purpose'].replace('new car',1)
df.to_csv('data.csv', index=False)

df['purpose']=df['purpose'].replace('furniture/equipment',2)
df.to_csv('data.csv', index=False)

df['purpose']=df['purpose'].replace('used car',3)
df.to_csv('data.csv', index=False)

df['purpose']=df['purpose'].replace('business',4)
df.to_csv('data.csv', index=False)

df['purpose']=df['purpose'].replace('education',5)
df.to_csv('data.csv', index=False)

df['purpose']=df['purpose'].replace('repairs',6)
df.to_csv('data.csv', index=False)

df['purpose']=df['purpose'].replace('domestic appliance',7)
df.to_csv('data.csv', index=False)

df['purpose']=df['purpose'].replace('other',8)
df.to_csv('data.csv', index=False)

df['purpose']=df['purpose'].replace('retraining',9)
df.to_csv('data.csv', index=False)
```

In [14]:

```
df['savings_status'].value_counts()
```

Out[14]:

```
<100                603
no known savings    183
100<=X<500          103
500<=X<1000          63
>=1000               48
Name: savings_status, dtype: int64
```

In [15]:

```
df['savings_status']=df['savings_status'].replace('<100',0)
df.to_csv('data.csv', index=False)

df['savings_status']=df['savings_status'].replace('no known savings',1)
df.to_csv('data.csv', index=False)

df['savings_status']=df['savings_status'].replace('100<=X<500',2)
df.to_csv('data.csv', index=False)

df['savings_status']=df['savings_status'].replace('500<=X<1000',3)
df.to_csv('data.csv', index=False)

df['savings_status']=df['savings_status'].replace('>=1000',4)
df.to_csv('data.csv', index=False)
```

In [16]:

```
df['employment'].value_counts()
```

Out[16]:

```
1<=X<4        339
>=7           253
4<=X<7        174
<1            172
unemployed     62
Name: employment, dtype: int64
```

In [17]:

```
df['employment']=df['employment'].replace('1<=X<4',0)
df.to_csv('data.csv', index=False)

df['employment']=df['employment'].replace('>=7',1)
df.to_csv('data.csv', index=False)

df['employment']=df['employment'].replace('4<=X<7',2)
df.to_csv('data.csv', index=False)

df['employment']=df['employment'].replace('<1',3)
df.to_csv('data.csv', index=False)

df['employment']=df['employment'].replace('unemployed',4)
df.to_csv('data.csv', index=False)
```

In [18]:

```
df['personal_status'].value_counts()
```

Out[18]:

```
male single           548
female div/dep/mar    310
male mar/wid           92
male div/sep           50
Name: personal_status, dtype: int64
```

In [19]:

```
df['personal_status']=df['personal_status'].replace('male single',0)
df.to_csv('data.csv', index=False)

df['personal_status']=df['personal_status'].replace('female div/dep/mar',1)
df.to_csv('data.csv', index=False)

df['personal_status']=df['personal_status'].replace('male mar/wid',2)
df.to_csv('data.csv', index=False)

df['personal_status']=df['personal_status'].replace('male div/sep',3)
df.to_csv('data.csv', index=False)
```

In [20]:

```
df['other_parties'].value_counts()
```

Out[20]:

```
none            907
guarantor        52
co applicant     41
Name: other_parties, dtype: int64
```

In [21]:

```
df['other_parties']=df['other_parties'].replace('none',0)
df.to_csv('data.csv', index=False)

df['other_parties']=df['other_parties'].replace('guarantor',1)
df.to_csv('data.csv', index=False)

df['other_parties']=df['other_parties'].replace('co applicant',2)
df.to_csv('data.csv', index=False)
```

In [22]:

```
df['property_magnitude'].value_counts()
```

Out[22]:

```
car                    332
real estate            282
life insurance         232
no known property      154
Name: property_magnitude, dtype: int64
```

In [23]:

```
df['property_magnitude']=df['property_magnitude'].replace('car',0)
df.to_csv('data.csv', index=False)

df['property_magnitude']=df['property_magnitude'].replace('real estate',1)
df.to_csv('data.csv', index=False)

df['property_magnitude']=df['property_magnitude'].replace('life insurance',2)
df.to_csv('data.csv', index=False)

df['property_magnitude']=df['property_magnitude'].replace('no known property',3)
df.to_csv('data.csv', index=False)
```

In [24]:

```
df['other_payment_plans'].value_counts()
```

Out[24]:

```
none      814
bank      139
stores     47
Name: other_payment_plans, dtype: int64
```

In [25]:

```
df['other_payment_plans']=df['other_payment_plans'].replace('none',0)
df.to_csv('data.csv', index=False)

df['other_payment_plans']=df['other_payment_plans'].replace('bank',1)
df.to_csv('data.csv', index=False)

df['other_payment_plans']=df['other_payment_plans'].replace('stores',2)
df.to_csv('data.csv', index=False)
```

In [26]:

```
df['housing'].value_counts()
```

Out[26]:

```
own         713
rent        179
for free    108
Name: housing, dtype: int64
```

In [27]:

```
df['housing']=df['housing'].replace('own',0)
df.to_csv('data.csv', index=False)

df['housing']=df['housing'].replace('rent',1)
df.to_csv('data.csv', index=False)

df['housing']=df['housing'].replace('for free',2)
df.to_csv('data.csv', index=False)
```

In [28]:

```
df['job'].value_counts()
```

Out[28]:

```
skilled                      630
unskilled resident           200
high qualif/self emp/mgmt    148
unemp/unskilled non res       22
Name: job, dtype: int64
```

In [29]:

```
df['job']=df['job'].replace('skilled',0)
df.to_csv('data.csv', index=False)

df['job']=df['job'].replace('unskilled resident',1)
df.to_csv('data.csv', index=False)

df['job']=df['job'].replace('high qualif/self emp/mgmt',2)
df.to_csv('data.csv', index=False)

df['job']=df['job'].replace('unemp/unskilled non res',3)
df.to_csv('data.csv', index=False)
```

In [30]:

```
df['own_telephone'].value_counts()
```

Out[30]:

```
none    596
yes     404
Name: own_telephone, dtype: int64
```

In [31]:

```
df['own_telephone']=df['own_telephone'].replace('none',0)
df.to_csv('data.csv', index=False)

df['own_telephone']=df['own_telephone'].replace('yes',1)
df.to_csv('data.csv', index=False)
```

In [32]:

```
df['foreign_worker'].value_counts()
```

Out[32]:

```
yes    963
no      37
Name: foreign_worker, dtype: int64
```

In [33]:

```
df['foreign_worker']=df['foreign_worker'].replace('yes',0)
df.to_csv('data.csv', index=False)

df['foreign_worker']=df['foreign_worker'].replace('no',1)
df.to_csv('data.csv', index=False)
```

In [ ]:

```

```

Result after changing the values into int/float

In [34]:

```
df.head()
```

Out[34]:

	Unnamed: 0	checking_status	duration	credit_history	purpose	credit_amount	savings_sta
0	0	1	6.0	1	0	1169.0	
1	1	2	48.0	0	0	5951.0	
2	2	0	12.0	1	5	2096.0	
3	3	1	42.0	0	2	7882.0	
4	4	1	24.0	2	1	4870.0	

5 rows × 22 columns

In [ ]:

Checking null values, duplicated values and also value count for dependent variable

In [35]:

```
df.shape
```

Out[35]:

```
(1000, 22)
```

In [36]:

```
df.isnull().sum()
```

Out[36]:

```
Unnamed: 0                0
checking_status           0
duration                  0
credit_history            0
purpose                   0
credit_amount             0
savings_status            0
employment                0
installment_commitment    0
personal_status           0
other_parties             0
residence_since           0
property_magnitude        0
age                       0
other_payment_plans       0
housing                   0
existing_credits          0
job                       0
num_dependents            0
own_telephone             0
foreign_worker            0
class                     0
dtype: int64
```

In [37]:

```
df.duplicated().sum()
```

Out[37]:

```
0
```

In [38]:

```
df['class'].value_counts()
```

Out[38]:

```
1    700
0    300
Name: class, dtype: int64
```

In [ ]:

```

```

Initialising the value of dependent variable (y) and independent variable (x)

In [40]:

```
y= df['class']
x= df.drop('class', axis=1)
x= x.drop('Unnamed: 0', axis=1)

print(x.shape)
print(y.shape)
```

```
(1000, 20)
(1000,)
```

In [41]:

```
y.head()
```

Out[41]:

```
0    1
1    0
2    1
3    1
4    0
Name: class, dtype: int64
```

In [ ]:

```

```

Using train_test_split

In [42]:

```
from sklearn.model_selection import train_test_split
```

In [43]:

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(750, 20)
(250, 20)
(750,)
(250,)
```

In [ ]:

```

```

Now computing accuracy, mscore by different model

1) Logistic Regression

In [44]:

```
from sklearn.linear_model import LogisticRegression
```

In [45]:

```
m1 = LogisticRegression(max_iter=1000)
m1.fit(x_train,y_train)
```

Out[45]:

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

In [46]:

```
def eval_model(ytest,ypred):
 cm = confusion_matrix(ytest,ypred)
 print(cm)
 print(classification_report(ytest,ypred))

def mscore(model):
 print('Train Score',model.score(x_train,y_train))
 print('Test Score',model.score(x_test,y_test))
```

In [47]:

```
mscore(m1)
```

```
Train Score 0.7253333333333334
Test Score 0.7
```

In [48]:

```
from sklearn.metrics import confusion_matrix, classification_report
```

In [49]:

```
ypred_m1 = m1.predict(x_test)
eval_model(y_test,ypred_m1) #more accurate than KNN but less then SVM
```

```
[[ 23  49]
 [ 26 152]]
              precision    recall  f1-score   support

           0       0.47      0.32      0.38        72
           1       0.76      0.85      0.80       178

    accuracy                           0.70       250
   macro avg       0.61      0.59      0.59       250
weighted avg       0.67      0.70      0.68       250
```

In [51]:

```
# trying now with logistic regression

ypred_log = m1.predict([[1, 49.0, 0, 0, 5951, 1, 1, 3, 0, 0, 4, 1, 20, 0, 0, 1, 1, 0, 0,
print(ypred_log)
```

```
[0]

C:\Users\HP\anaconda3\anacondav3\lib\site-packages\sklearn\base.py:420: U
serWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

2) KNN

In [52]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [53]:

```
m2 = KNeighborsClassifier(n_neighbors=15)
m2.fit(x_train,y_train)
```

Out[53]:

```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=15)
```

In [54]:

```
mscore(m2)
```

```
Train Score 0.7213333333333334
Test Score 0.688
```

In [68]:

```
ypred_m2 = m2.predict(x_test)
eval_model(y_test,ypred_m2) #worst accuracy so we not use this!
```

```
[[ 10  62]
 [ 16 162]]
              precision    recall  f1-score   support

           0       0.38      0.14      0.20        72
           1       0.72      0.91      0.81       178

    accuracy                           0.69       250
   macro avg       0.55      0.52      0.51       250
weighted avg       0.63      0.69      0.63       250
```

3) SVM

In [56]:

```
from sklearn.svm import SVC
```

In [57]:

```
m3 = SVC(kernel='linear',C=10)
m3.fit(x_train,y_train)
```

Out[57]:

```
SVC
SVC(C=10, kernel='linear')
```

In [58]:

```
mscore(m3)
```

```
Train Score 0.712
Test Score 0.72
```

In [59]:

```
ypred_m3 = m3.predict(x_test)
eval_model(y_test,ypred_m3) #SVM model got the best accuracy
```

```
[[ 36  36]
 [ 34 144]]
              precision    recall  f1-score   support

           0       0.51      0.50      0.51        72
           1       0.80      0.81      0.80       178

    accuracy                           0.72       250
   macro avg       0.66      0.65      0.66       250
weighted avg       0.72      0.72      0.72       250
```

In [60]:

```
# trying with SVM Model
ypred_svm = m3.predict([[1, 49.0, 0, 0, 5951, 1, 1, 3, 0, 0, 4, 1, 20, 0, 0, 1, 1, 0, 0,
print(ypred_svm)
```

```
[0]

C:\Users\HP\anaconda3\anacondav3\lib\site-packages\sklearn\base.py:420: U
serWarning: X does not have valid feature names, but SVC was fitted with
feature names
  warnings.warn(
```

In [ ]:

```

```

Conclusion

```
Columns names with there values in string and integer

For Column= checking_status
no checking    0
<0             1
0<=X<200       2
>=200          3

For Column= credit_history
existing paid                  0
critical/other existing credit 1
delayed previously             2
all paid                       3
no credits/all paid            4

For Column= purpose
radio/tv              0
new car               1
furniture/equipment   2
used car              3
business              4
```

```
education              5
repairs                6
domestic appliance     7
other                  8
retraining             9

For Column= savings_status
<100                0
no known savings    1
100<=X<500          2
500<=X<1000         3
>=1000              4

For Column= employment
1<=X<4        0
>=7           1
4<=X<7        2
<1            3
unemployed    4

For Column= personal_status
male single           0
female div/dep/mar    1
male mar/wid          2
male div/sep          3

For Column= other_parties
none            0
guarantor       1
co applicant    2

For Column= property_magnitude
car                   0
real estate           1
life insurance        2
no known property     3

For Column= other_payment_plans
none      0
bank      1
stores    2

For Column= housing
own         0
rent        1
for free    2

For Column= job
skilled                      0
unskilled resident           1
high qualif/self emp/mgmt    2
unemp/unskilled non res      3

For Column= own_telephone
none    0
yes     1

For Column= foreign_worker
yes    0
no     1
```

```
AS PER ACCURACY
SVM > Logistic Regression > KNN
KNN has the worst accuracy so we don't use this
```

In [64]:

```python
def credit_score(value):
    if value == 1:
        print("Credit score is GOOD")
    elif value == 0:
        print("Credit score is BAD")
```

In [65]:

```python
# Result with logistic regression

ypred_logistic_regression = m1.predict([[1, 49.0, 0, 0, 5951, 1, 1, 3, 0, 0, 4, 1, 20, 0
print(ypred_logistic_regression)
credit_score(ypred_logistic_regression)
```

```
[0]
Credit score is BAD

C:\Users\HP\anaconda3\anacondav3\lib\site-packages\sklearn\base.py:420: U
serWarning: X does not have valid feature names, but LogisticRegression w
as fitted with feature names
  warnings.warn(
```

In [67]:

```python
# Result with SVM which has the best accuracy

ypred_SVM = m3.predict([[1, 49.0, 0, 0, 5951, 1, 1, 3, 0, 0, 4, 1, 20, 0, 0, 1, 1, 0, 0,
print(ypred_SVM)
credit_score(ypred_SVM)
```

```
[0]
Credit score is BAD

C:\Users\HP\anaconda3\anacondav3\lib\site-packages\sklearn\base.py:420: U
serWarning: X does not have valid feature names, but SVC was fitted with
feature names
  warnings.warn(
```

END - Thank You