Problem statement: Create a classification model to predict whether credit risk is good or bad.

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df=pd.read_csv('credit_customers (DS).csv')
```

In [3]:
```
df.head(5)
```

Out[3]:

	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	inst
0	<0	6.0	critical/other existing credit	radio/tv	1169.0	no known savings	>=7	
1	0<=X<200	48.0	existing paid	radio/tv	5951.0	<100	1<=X<4	
2	no checking	12.0	critical/other existing credit	education	2096.0	<100	4<=X<7	
3	<0	42.0	existing paid	furniture/equipment	7882.0	<100	4<=X<7	
4	<0	24.0	delayed previously	new car	4870.0	<100	1<=X<4	

5 rows × 21 columns

In [4]:
```
df.tail(5)
```

Out[4]:

	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	ir
995	no checking	12.0	existing paid	furniture/equipment	1736.0	<100	4<=X<7	
996	<0	30.0	existing paid	used car	3857.0	<100	1<=X<4	
997	no checking	12.0	existing paid	radio/tv	804.0	<100	>=7	
998	<0	45.0	existing paid	radio/tv	1845.0	<100	1<=X<4	
999	0<=X<200	45.0	critical/other existing credit	used car	4576.0	100<=X<500	unemployed	

5 rows × 21 columns

In [5]:
```
df.shape
```

Out[5]:
```
(1000, 21)
```

1. Remove handle null values (if any).

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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 [7]:
```
df.value_counts()
```

Out[7]:
```
checking_status  duration  credit_history  purpose              credit_amount  savings_
status    employment  installment_commitment  personal_status     other_parties  reside
nce_since  property_magnitude  age   other_payment_plans  housing   existing_credits  j
ob                        num_dependents  own_telephone  foreign_worker  class
0<=X<200        6.0       all paid        education            433.0          >=1000
<1          4.0                     female div/dep/mar  none            2.0
life insurance      24.0  bank                 rent      1.0                skilled
2.0             none            yes              bad       1
no checking      10.0      existing paid   new car              1364.0         <100
1<=X<4      2.0                     female div/dep/mar  none            4.0
car                 64.0  none                 own       1.0                skilled
1.0             yes             yes              good      1
                 9.0       existing paid   furniture/equipment  2301.0         100<=X<5
00        <1          2.0                     female div/dep/mar  none          4.0
life insurance      22.0  none                 rent      1.0                skilled
1.0             none            yes              good      1
                                           new car              2507.0         500<=X<1
000       >=7         2.0                     male single        none          4.0
no known property   51.0  none                 for free  1.0                unskilled re
```

In [8]:
```
df=df.dropna()
df.shape
```

Out[8]:
```
(1000, 21)
```

Data Preprocessing

Handling duplicate records

In [9]:
```
df.drop_duplicates(inplace=True)
```

Identify categorical columns and performing Label Enocoding

In [10]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

In [11]:
```
categorical_columms=['checking_status','property_magnitude','savings_status','employment',
label_encoder=LabelEncoder()
for col in categorical_columns:
    df[col]=label_encoder.fit_transform(df[col])
```

In [12]:
```
df['class'] = label_encoder.fit_transform(df['class'])
label_mapping = {0: 'good', 1: 'bad'}
df['class'] = df['class'].map(label_mapping)
```

In [13]:
```
df.head(5)
```

Out[13]:

	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	installment_c
0	1	6.0	1	6	1169.0	4	3	
1	0	48.0	3	6	5951.0	2	0	
2	3	12.0	1	2	2096.0	2	1	
3	1	42.0	3	3	7882.0	2	1	
4	1	24.0	2	4	4870.0	2	0	

5 rows × 21 columns

Split data into training and test data.

In [14]:
```
x=df.drop("class",axis=1)
y=df["class"]
print(type(x))
print(type(y))
print(x.shape)
print(y.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
(1000, 20)
(1000,)
```

In [15]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(800, 20)
(200, 20)
(800,)
(200,)
```

models a) Logistic Regression

In [16]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [17]:
```
logistic_regression = LogisticRegression()
```

In [18]:
```
logistic_regression.fit(x_train, y_train)
```

```
C:\Users\HP\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:458: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/
stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (http
s://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[18]:
```
▾ LogisticRegression
LogisticRegression()
```

In [19]:
```
logistic_regression_predictions = logistic_regression.predict(x_test)
```

In [20]:
```
logistic_regression_accuracy = accuracy_score(y_test, logistic_regression_predictions)
```

In [21]:
```
print("Logistic Regression Accuracy: ",logistic_regression_accuracy)
```

```
Logistic Regression Accuracy:  0.73
```

b) KNN Classification

In [22]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [23]:
```
knn_classifier=KNeighborsClassifier()
knn_classifier.fit(x_train,y_train)
knn_predictions = knn_classifier.predict(x_test)
```

In [24]:
```
knn_accuracy = accuracy_score(y_test, knn_predictions)
```

In [25]:
```
knn_accuracy
```

Out[25]:
```
0.685
```

In [26]:
```
print("KNN Accuracy:",knn_accuracy)
```

```
KNN Accuracy: 0.685
```

c) SVM Classifier with linear and rbf kernel

In [27]:
```
from sklearn.svm import SVC
```

In [28]:
```
svm_linear=SVC(kernel='linear')
svm_linear.fit(x_train,y_train)
svm_pred=svm_linear.predict(x_test)
```

In [29]:
```
svm_linear_accuracy=accuracy_score(y_test,svm_pred)
```

In [30]:
```
print("SVM (Linear Kernel) Accuracy:",svm_linear_accuracy)
```

```
SVM (Linear Kernel) Accuracy: 0.71
```

In [31]:
```
svm_rbf=SVC(kernel='rbf')
svm_rbf.fit(x_train,y_train)
svm_rbf_pred=svm_rbf.predict(x_test)
```

In [32]:
```
svm_rbf_accuracy=accuracy_score(y_test,svm_rbf_pred)
```

In [33]:
```
print("SVM (RBF Kernel) Accuracy:",svm_rbf_accuracy)
```

```
SVM (RBF Kernel) Accuracy: 0.715
```

Generate Confusion matrix and classification report for each of these models.

In [34]:
```
from sklearn.metrics import confusion_matrix,classification_report
```

In [35]:
```
best_accuracy = 0
best_model = ""
```

In [36]:
```
models = [
    ("Logistic Regression", logistic_regression_predictions),
    ("K-Nearest Neighbors", knn_predictions),
    ("SVM (Linear Kernel)", svm_pred),
    ("SVM (RBF Kernel)", svm_rbf_pred)
]
```

In [37]:
```
for model_name, predictions in models:
    print("Model: ",model_name)
    confusion = confusion_matrix(y_test, predictions)
    classification_rep = classification_report(y_test, predictions)
    accuracy = (confusion[0, 0] + confusion[1, 1]) / sum(sum(confusion))

    print("Confusion Matrix:")
    print(confusion)

    print("\nClassification Report:")
    print(classification_rep)

    print("Accuracy: ",accuracy,"\n")

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_model = model_name
```

```
Model:  Logistic Regression
Confusion Matrix:
[[128  13]
 [ 41  18]]

Classification Report:
              precision    recall  f1-score   support

         bad       0.76      0.91      0.83       141
        good       0.58      0.31      0.40        59

    accuracy                           0.73       200
   macro avg       0.67      0.61      0.61       200
weighted avg       0.71      0.73      0.70       200

Accuracy:  0.73

Model:  K-Nearest Neighbors
Confusion Matrix:
[[123  18]
 [ 45  14]]

Classification Report:
              precision    recall  f1-score   support

         bad       0.73      0.87      0.80       141
        good       0.44      0.24      0.31        59

    accuracy                           0.69       200
   macro avg       0.58      0.55      0.55       200
weighted avg       0.65      0.69      0.65       200

Accuracy:  0.685

Model:  SVM (Linear Kernel)
Confusion Matrix:
[[116  25]
 [ 33  26]]

Classification Report:
              precision    recall  f1-score   support

         bad       0.78      0.82      0.80       141
        good       0.51      0.44      0.47        59

    accuracy                           0.71       200
   macro avg       0.64      0.63      0.64       200
weighted avg       0.70      0.71      0.70       200

Accuracy:  0.71

Model:  SVM (RBF Kernel)
Confusion Matrix:
[[140   1]
 [ 56   3]]

Classification Report:
              precision    recall  f1-score   support

         bad       0.71      0.99      0.83       141
        good       0.75      0.05      0.10        59

    accuracy                           0.71       200
```

```
   macro avg       0.73      0.52      0.46       200
weighted avg       0.72      0.71      0.61       200

Accuracy:  0.715
```

In [38]:

```
print("The model with the best accuracy is",best_model,"with an accuracy of",best_accuracy
```

```
The model with the best accuracy is Logistic Regression with an accuracy of 0.73
```