CLASSWORK

In [33]:
```
import pandas as pd

# Load the dataset
cc_apps = pd.read_csv("cc_approvals.data", header=None)
from sklearn.model_selection import train_test_split

print(cc_apps.corr())

#Drop the features 11 and 13
cc_apps = cc_apps.drop([11, 13], axis=1)

# Split into train and test sets
cc_apps_train, cc_apps_test = train_test_split(cc_apps, test_size=0.33, rand
# Import numpy
import numpy as np

# Replace the '?'s with NaN in the train and test sets
cc_apps_train = cc_apps_train.replace('?', np.NaN)
cc_apps_test = cc_apps_test.replace('?', np.NaN)
cc_apps_train.fillna(cc_apps_train.mean(), inplace=True)
cc_apps_test.fillna(cc_apps_train.mean(), inplace=True)
for col in cc_apps_train.columns: # Iterate over each column of cc_apps_trai

    if cc_apps_train[col].dtypes == 'object': # Check if the column is of ob
        # Impute with the most frequent value
        # The value_counts() function returns a Series that contain counts c
        # descending order so that its first element will be the most freque
        cc_apps_train = cc_apps_train.fillna(cc_apps_train[col].value_counts
        cc_apps_test = cc_apps_test.fillna(cc_apps_train[col].value_counts()

cc_apps_train = pd.get_dummies(cc_apps_train)
cc_apps_test = pd.get_dummies(cc_apps_test)
cc_apps_test = cc_apps_test.reindex(columns=cc_apps_train.columns, fill_valu
from sklearn.preprocessing import MinMaxScaler

# Segregate features and labels into separate variables
X_train, y_train = cc_apps_train.iloc[:, :-1].values, cc_apps_train.iloc[:,
X_test, y_test = cc_apps_test.iloc[:, :-1].values, cc_apps_test.iloc[:, [-1]

# Instantiate MinMaxScaler and use it to rescale X_train and X_test
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX_train = scaler.fit_transform(X_train)
rescaledX_test = scaler.transform(X_test)
# Import LogisticRegression
from sklearn.linear_model import LogisticRegression

# Instantiate a LogisticRegression classifier with default parameter values
logreg = LogisticRegression()

# Fit logreg to the train set
logreg.fit(rescaledX_train,y_train)
# Import confusion_matrix
from sklearn.metrics import confusion_matrix

# Use logreg to predict instances from the test set and store it
y_pred = logreg.predict(rescaledX_test)

# Get the accuracy score of logreg model and print it
```

```
print("Accuracy of logistic regression classifier: ", logreg.score(rescaledX

# Print the confusion matrix of the logreg model
confusion_matrix(y_test,y_pred)
```

```
          2         7         10        14
2   1.000000  0.298902  0.271207  0.123121
7   0.298902  1.000000  0.322330  0.051345
10  0.271207  0.322330  1.000000  0.063692
14  0.123121  0.051345  0.063692  1.000000
Accuracy of logistic regression classifier:  1.0
```

```
/var/folders/4j/bnvctt7152z6l5l6szd4m7wh0000gn/T/ipykernel_23995/1641907923.
py:20: FutureWarning: Dropping of nuisance columns in DataFrame reductions
(with 'numeric_only=None') is deprecated; in a future version this will rais
e TypeError.  Select only valid columns before calling the reduction.
  cc_apps_train.fillna(cc_apps_train.mean(), inplace=True)
/var/folders/4j/bnvctt7152z6l5l6szd4m7wh0000gn/T/ipykernel_23995/1641907923.
py:21: FutureWarning: Dropping of nuisance columns in DataFrame reductions
(with 'numeric_only=None') is deprecated; in a future version this will rais
e TypeError.  Select only valid columns before calling the reduction.
  cc_apps_test.fillna(cc_apps_train.mean(), inplace=True)
/Users/richard/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/valid
ation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples, ), for examp
le using ravel().
  y = column_or_1d(y, warn=True)
```

Out[33]:
```
array([[103,   0],
       [  0, 125]])
```

In [34]:
```
# # Import GridSearchCV
# from sklearn.model_selection import GridSearchCV

# # Define the grid of values for tol and max_iter
# tol = [0.01, 0.001 ,0.0001]
# max_iter = [100, 150, 200]

# # Create a dictionary where tol and max_iter are keys and the lists of the
# param_grid = dict(tol=tol, max_iter=max_iter)
# # Instantiate GridSearchCV with the required parameters
# grid_model = GridSearchCV(estimator=logreg, param_grid=param_grid, cv=5)

# # Fit data to grid_model
# grid_model_result = grid_model.fit(rescaledX, y)

# # Summarize results
# best_score, best_params = grid_model_result.best_score_, grid_model_result
# print("Best: %f using %s" % (best_score, best_params))
```

TASK 1

In [37]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

# Create a KNeighborsClassifier
knn = KNeighborsClassifier()

# Define a range of K values to try
param_grid = {'n_neighbors': range(1, 21)}  # Try K values from 1 to 20

# Create a grid search using cross-validation
```

```
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, scoring='ac
grid_search.fit(rescaledX_train, y_train.ravel())  # Using ravel() to conver

# Print accuracy for each K value
for params, accuracy in zip(grid_search.cv_results_['params'], grid_search.c
    k = params['n_neighbors']
    print(f"K = {k}: Accuracy = {accuracy}")

# Get the best K value and its corresponding accuracy
best_k = grid_search.best_params_['n_neighbors']
best_accuracy = grid_search.best_score_

print("\nBest K value:", best_k)
print("Best accuracy:", best_accuracy)

# Create a KNeighborsClassifier with the best K value
best_knn = KNeighborsClassifier(n_neighbors=best_k)
best_knn.fit(rescaledX_train, y_train.ravel())

# Use the best_knn to predict and calculate test accuracy
y_pred_test = best_knn.predict(rescaledX_test)
#test_accuracy = accuracy_score(y_test, y_pred_test)

#print("\nTest accuracy with best K:", test_accuracy)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_test))
```

```
K = 1: Accuracy = 0.9048153342683497
K = 2: Accuracy = 0.9090696587190275
K = 3: Accuracy = 0.9176718092566618
K = 4: Accuracy = 0.9111266947171576
K = 5: Accuracy = 0.9089294062646097
K = 6: Accuracy = 0.9089527816736792
K = 7: Accuracy = 0.9241701729780271
K = 8: Accuracy = 0.9285179990649837
K = 9: Accuracy = 0.9307152875175315
K = 10: Accuracy = 0.9307620383356708
K = 11: Accuracy = 0.926437587657784
K = 12: Accuracy = 0.9328892005610097
K = 13: Accuracy = 0.9329125759700794
K = 14: Accuracy = 0.9307386629266012
K = 15: Accuracy = 0.935063113604488
K = 16: Accuracy = 0.9328892005610099
K = 17: Accuracy = 0.9263674614305751
K = 18: Accuracy = 0.9242402992052361
K = 19: Accuracy = 0.9155913978494624
K = 20: Accuracy = 0.9112435717625058

Best K value: 15
Best accuracy: 0.935063113604488
Confusion Matrix:
 [[ 93  10]
 [  7 118]]
```

TASK 2

In [36]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
# Load the dataset
cc_apps = pd.read_csv("cc_approvals.data", header=None)
cc_apps = cc_apps.drop([11, 13], axis=1)
# Replace the '?'s with NaN
cc_apps = cc_apps.replace('?', np.NaN)
cc_apps.fillna(cc_apps.mean(), inplace=True)

# Impute missing values with the most frequent value for each column
for col in cc_apps.columns:
    if cc_apps[col].dtypes == 'object':
        cc_apps = cc_apps.fillna(cc_apps[col].value_counts().index[0])

# Perform one-hot encoding on categorical features
cc_apps = pd.get_dummies(cc_apps)

# Split into features (X) and target labels (y)
X = cc_apps.iloc[:, :-1].values  # Use all columns except the last one as fe
y = cc_apps.iloc[:, -1].values   # Use the last column as the target label

# Scale the features to a range between 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX = scaler.fit_transform(X)

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(rescaledX, y, test_size=

# Instantiate and train the Logistic Regression model
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Use the trained model to make predictions on the test set
y_pred = logreg.predict(X_test)

# Calculate and print the accuracy of the Logistic Regression model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of logistic regression classifier: ", accuracy)

# Print the confusion matrix of the Logistic Regression model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy of logistic regression classifier:  1.0
Confusion Matrix:
 [[103   0]
 [  0 125]]
```

```
/var/folders/4j/bnvctt7152z615l6szd4m7wh0000gn/T/ipykernel_23995/2160929429.
py:13: FutureWarning: Dropping of nuisance columns in DataFrame reductions
(with 'numeric_only=None') is deprecated; in a future version this will rais
e TypeError.  Select only valid columns before calling the reduction.
  cc_apps.fillna(cc_apps.mean(), inplace=True)
```

In [ ]:

In [ ]: