In [40]:
```
import pandas as pd
cc_apps = pd.read_csv("cc_approvals.data", header = None)
cc_apps.head()
```

Out[40]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | b | 30.83 | 0.000 | u | g | w | v | 1.25 | t | t | 1 | f | g | 00202 | 0 | + |
| **1** | a | 58.67 | 4.460 | u | g | q | h | 3.04 | t | t | 6 | f | g | 00043 | 560 | + |
| **2** | a | 24.50 | 0.500 | u | g | q | h | 1.50 | t | f | 0 | f | g | 00280 | 824 | + |
| **3** | b | 27.83 | 1.540 | u | g | w | v | 3.75 | t | t | 5 | t | g | 00100 | 3 | + |
| **4** | b | 20.17 | 5.625 | u | g | w | v | 1.71 | t | f | 0 | f | s | 00120 | 0 | + |

In [41]:
```
print(cc_apps.describe())
print('\n')

print(cc_apps.info())
print('\n')

cc_apps.tail(17) # or cc_apps.sample()
```

```
                2           7         10             14
count  690.000000  690.000000  690.00000     690.000000
mean     4.758725    2.223406    2.40000    1017.385507
std      4.978163    3.346513    4.86294    5210.102598
min      0.000000    0.000000    0.00000       0.000000
25%      1.000000    0.165000    0.00000       0.000000
50%      2.750000    1.000000    0.00000       5.000000
75%      7.207500    2.625000    3.00000     395.500000
max     28.000000   28.500000   67.00000  100000.000000


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       690 non-null    object 
 1   1       690 non-null    object 
 2   2       690 non-null    float64
 3   3       690 non-null    object 
 4   4       690 non-null    object 
 5   5       690 non-null    object 
 6   6       690 non-null    object 
 7   7       690 non-null    float64
 8   8       690 non-null    object 
 9   9       690 non-null    object 
 10  10      690 non-null    int64  
 11  11      690 non-null    object 
 12  12      690 non-null    object 
 13  13      690 non-null    object 
 14  14      690 non-null    int64  
 15  15      690 non-null    object 
dtypes: float64(2), int64(2), object(12)
memory usage: 86.4+ KB
None
```

Out[41]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **673** | ? | 29.50 | 2.000 | y | p | e | h | 2.000 | f | f | 0 | f | g | 00256 | 17 | - |
| **674** | a | 37.33 | 2.500 | u | g | i | h | 0.210 | f | f | 0 | f | g | 00260 | 246 | - |
| **675** | a | 41.58 | 1.040 | u | g | aa | v | 0.665 | f | f | 0 | f | g | 00240 | 237 | - |
| **676** | a | 30.58 | 10.665 | u | g | q | h | 0.085 | f | t | 12 | t | g | 00129 | 3 | - |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **677** | b | 19.42 | 7.250 | u | g | m | v | 0.040 | f | t | 1 | f | g | 00100 | 1 | - |
| **678** | a | 17.92 | 10.210 | u | g | ff | ff | 0.000 | f | f | 0 | f | g | 00000 | 50 | - |
| **679** | a | 20.08 | 1.250 | u | g | c | v | 0.000 | f | f | 0 | f | g | 00000 | 0 | - |
| **680** | b | 19.50 | 0.290 | u | g | k | v | 0.290 | f | f | 0 | f | g | 00280 | 364 | - |
| **681** | b | 27.83 | 1.000 | y | p | d | h | 3.000 | f | f | 0 | f | g | 00176 | 537 | - |
| **682** | b | 17.08 | 3.290 | u | g | i | v | 0.335 | f | f | 0 | t | g | 00140 | 2 | - |
| **683** | b | 36.42 | 0.750 | y | p | d | v | 0.585 | f | f | 0 | f | g | 00240 | 3 | - |
| **684** | b | 40.58 | 3.290 | u | g | m | v | 3.500 | f | f | 0 | t | s | 00400 | 0 | - |
| **685** | b | 21.08 | 10.085 | y | p | e | h | 1.250 | f | f | 0 | f | g | 00260 | 0 | - |
| **686** | a | 22.67 | 0.750 | u | g | c | v | 2.000 | f | t | 2 | t | g | 00200 | 394 | - |
| **687** | a | 25.25 | 13.500 | y | p | ff | ff | 2.000 | f | t | 1 | t | g | 00200 | 1 | - |
| **688** | b | 17.92 | 0.205 | u | g | aa | v | 0.040 | f | f | 0 | f | g | 00280 | 750 | - |

In [42]:
```
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, random
```

```
           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
```

In [43]:
```
# 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)
```

In [44]:
```
# Impute the missing values with mean imputation
cc_apps_train.fillna(cc_apps_train.mean(), inplace=True)
cc_apps_test.fillna(cc_apps_train.mean(), inplace=True)

# Count the number of NaNs in the datasets and print the counts to verify
print(cc_apps_train.isnull().sum())
print(cc_apps_test.isnull().sum())
```

```
0     8
1     5
2     0
3     6
4     6
5     7
6     7
7     0
8     0
9     0
10    0
12    0
14    0
15    0
dtype: int64
0     4
1     7
2     0
3     0
4     0
5     2
6     2
7     0
8     0
9     0
10    0
12    0
14    0
15    0
dtype: int64

C:\Users\Lut Lat Aung\AppData\Local\Temp\ipykernel_29180\3580017964.py:2: Fut
ureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numer
ic_only=None') is deprecated; in a future version this will raise TypeError.
Select only valid columns before calling the reduction.
  cc_apps_train.fillna(cc_apps_train.mean(), inplace=True)
C:\Users\Lut Lat Aung\AppData\Local\Temp\ipykernel_29180\3580017964.py:3: Fut
ureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numer
ic_only=None') is deprecated; in a future version this will raise TypeError.
Select only valid columns before calling the reduction.
  cc_apps_test.fillna(cc_apps_train.mean(), inplace=True)
```

In [45]:
```
for col in cc_apps_train.columns: # Iterate over each column of cc_apps_train

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

# Count the number of NaNs in the dataset and print the counts to verify
print(cc_apps_train.isnull().sum())
print(cc_apps_test.isnull().sum())
# At this point, there is no missing values.
```

```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
12    0
14    0
15    0
dtype: int64
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
12    0
14    0
15    0
dtype: int64
```

In [46]:
```
# Convert the categorical features in the train and test sets independently
print(cc_apps_train)
cc_apps_train = pd.get_dummies(cc_apps_train)
cc_apps_test = pd.get_dummies(cc_apps_test)
print(cc_apps_train)
# Reindex the columns of the test set aligning with the train set
cc_apps_test = cc_apps_test.reindex(columns=cc_apps_train.columns, fill_value=
```

```
    0      1      2  3  4   5   6       7  8  9  10 12    14 15
382  a  24.33  2.500  y  p   i  bb   4.500  f  f   0  g   456  -
137  b  33.58  2.750  u  g   m   v   4.250  t  t   6  g     0  +
346  b  32.25  1.500  u  g   c   v   0.250  f  f   0  g   122  -
326  b  30.17  1.085  y  p   c   v   0.040  f  f   0  g   179  -
33   a  36.75  5.125  u  g   e   v   5.000  t  f   0  g  4000  +
..  ..    ...    ... .. ..  ..  ..     ... .. ..  .. ..   ... ..
71   b  34.83  4.000  u  g   d  bb  12.500  t  f   0  g     0  -
106  b  28.75  1.165  u  g   k   v   0.500  t  f   0  s     0  -
270  b  37.58  0.000  b  b   b   b   0.000  f  f   0  p     0  +
435  b  19.00  0.000  y  p  ff  ff   0.000  f  t   4  g     1  -
102  b  18.67  5.000  u  g   q   v   0.375  t  t   2  g    38  -

[462 rows x 14 columns]
         2       7  10    14  0_a  0_b  1_13.75  1_15.83  1_15.92  1_16.00  \
382  2.500   4.500   0   456    1    0        0        0        0        0
137  2.750   4.250   6     0    0    1        0        0        0        0
346  1.500   0.250   0   122    0    1        0        0        0        0
326  1.085   0.040   0   179    0    1        0        0        0        0
33   5.125   5.000   0  4000    1    0        0        0        0        0
..     ...     ...  ..   ...  ...  ...      ...      ...      ...      ...
71   4.000  12.500   0     0    0    1        0        0        0        0
106  1.165   0.500   0     0    0    1        0        0        0        0
270  0.000   0.000   0     0    0    1        0        0        0        0
435  0.000   0.000   4     1    0    1        0        0        0        0
102  5.000   0.375   2    38    0    1        0        0        0        0

     ...  6_z  8_f  8_t  9_f  9_t  12_g  12_p  12_s  15_+  15_-
382  ...    0    1    0    1    0     1     0     0     0     1
137  ...    0    0    1    0    1     1     0     0     1     0
346  ...    0    1    0    1    0     1     0     0     0     1
326  ...    0    1    0    1    0     1     0     0     0     1
33   ...    0    0    1    1    0     1     0     0     1     0
..   ...  ...  ...  ...  ...  ...   ...   ...   ...   ...   ...
71   ...    0    0    1    1    0     1     0     0     0     1
106  ...    0    0    1    1    0     0     0     1     0     1
270  ...    0    1    0    1    0     0     1     0     1     0
435  ...    0    1    0    0    1     1     0     0     0     1
102  ...    0    0    1    0    1     1     0     0     0     1

[462 rows x 334 columns]
```

In [47]:
```
# Import MinMaxScaler
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)
```

In [48]:
```
# 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)
```

```
C:\Users\Lut Lat Aung\anaconda3\lib\site-packages\sklearn\utils\validation.p
y:993: DataConversionWarning: A column-vector y was passed when a 1d array wa
s expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

Out[48]:
```
LogisticRegression()
```

In [49]:
```
# 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_t

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

```
Accuracy of logistic regression classifier:  1.0
```

Out[49]:
```
array([[103,   0],
       [  0, 125]], dtype=int64)
```

In [50]:

```
Accuracy of the best KNN model:  0.7368421052631579

C:\Users\Lut Lat Aung\anaconda3\lib\site-packages\sklearn\neighbors\_classifi
cation.py:198: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples,), for example
using ravel().
  return self._fit(X, y)
```

In [66]:
```
#Task 1

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler, LabelEncoder

# Load data
cc_apps = pd.read_csv("cc_approvals.data", header=None)

# Data Preprocessing
# Handling Missing Values (replace '?' with NaN)
cc_apps.replace('?', np.nan, inplace=True)
cc_apps.fillna(cc_apps.median(), inplace=True)  # Filling missing values with 

# Convert non-numeric columns to strings
for column in cc_apps.columns:
    if cc_apps[column].dtype == 'object':
        cc_apps[column] = cc_apps[column].astype(str)

# Label Encoding for non-numeric columns
label_encoders = {}
for column in cc_apps.columns:
    if cc_apps[column].dtype == 'object':
        label_encoders[column] = LabelEncoder()
        cc_apps[column] = label_encoders[column].fit_transform(cc_apps[column]

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

# Split data into features and target variable
X = cc_apps.drop(columns=[2])  # Replace 'target_column' with your actual targ
y = cc_apps[2]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, rand


# Find the best k using Grid Search for KNeighborsRegressor
param_grid = {'n_neighbors': range(1, 21)}  # Trying k values from 1 to 20
knn = KNeighborsRegressor()
grid_search = GridSearchCV(knn, param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)

# Print the best k value
print("Best K: ", grid_search.best_params_['n_neighbors'])

# Train the final KNN model with the best k value
best_k = grid_search.best_params_['n_neighbors']
knn = KNeighborsRegressor(n_neighbors=best_k)
knn.fit(X_train_scaled, y_train)
```

```
test_accuracy = knn.score(X_test, y_test)

print("Accuracy of the best KNN model: ", test_accuracy)
```

```
C:\Users\Lut Lat Aung\AppData\Local\Temp\ipykernel_29180\863855895.py:17: Fut
ureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numer
ic_only=None') is deprecated; in a future version this will raise TypeError.
Select only valid columns before calling the reduction.
  cc_apps.fillna(cc_apps.median(), inplace=True)  # Filling missing values wi
th median for numerical columns

Best K:  18
Accuracy of the best KNN model:  -0.43739902524631247
```

In [ ]:

In [ ]:

In [55]:
```
import pandas as pd
cc_apps = pd.read_csv("cc_approvals.data", header = None)
print(cc_apps.head())
print(cc_apps.describe())
print('\n')
print(cc_apps.info())
print('\n')
cc_apps.tail(17)
cc_apps.fillna(cc_apps.mean(), inplace=True)
print(cc_apps.isnull().sum())
import numpy as np
cc_apps = cc_apps.replace('?', np.NaN)
for col in cc_apps.columns:

    if cc_apps[col].dtypes == 'object':

        cc_apps = cc_apps.fillna(cc_apps[col].value_counts().index[0])
print(cc_apps.isnull().sum())
print(cc_apps.describe())
print('\n')
print(cc_apps.info())
print('\n')
cc_apps.tail(17)
cc_apps = pd.get_dummies(cc_apps)
print(cc_apps)

from sklearn.preprocessing import MinMaxScaler

X_train, y_train = cc_apps.iloc[:, :-1].values, cc_apps.iloc[:, [-1]].values
X_test, y_test = cc_apps.iloc[:, :-1].values, cc_apps.iloc[:, [-1]].values
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX_train = scaler.fit_transform(X_train)
rescaledX_test = scaler.transform(X_test)

from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(rescaledX_train,y_train)

from sklearn.model_selection import train_test_split

print(cc_apps.corr())
cc_apps_train, cc_apps_test = train_test_split(cc_apps, test_size=0.33, random

from sklearn.metrics import confusion_matrix

y_pred = logreg.predict(rescaledX_test)
print("Accuracy of logistic regression classifier: ", logreg.score(rescaledX_t
confusion_matrix(y_test,y_pred)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

for k in range(1, 10):
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
```

```
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'k = {k}: Accuracy = {accuracy:.2f}')
```

```
  0      1      2  3  4  5  6     7  8  9   10 11 12     13   14 15
0  b  30.83  0.000  u  g  w  v  1.25  t  t   1  f  g  00202    0  +
1  a  58.67  4.460  u  g  q  h  3.04  t  t   6  f  g  00043  560  +
2  a  24.50  0.500  u  g  q  h  1.50  t  f   0  f  g  00280  824  +
3  b  27.83  1.540  u  g  w  v  3.75  t  t   5  t  g  00100    3  +
4  b  20.17  5.625  u  g  w  v  1.71  t  f   0  f  s  00120    0  +
               2           7          10             14
count  690.000000  690.000000  690.00000     690.000000
mean     4.758725    2.223406    2.40000    1017.385507
std      4.978163    3.346513    4.86294    5210.102598
min      0.000000    0.000000    0.00000       0.000000
25%      1.000000    0.165000    0.00000       0.000000
50%      2.750000    1.000000    0.00000       5.000000
75%      7.207500    2.625000    3.00000     395.500000
max     28.000000   28.500000   67.00000  100000.000000


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
```

In [ ]: