```
print("Hello World")
```

```
Hello World
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
```

```
cc_apps = pd.read_csv("cc_approvals.data", header = None)
cc_apps.head()
```

```
  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  +
```

```
print(cc_apps.describe())
print(cc_apps.info())
```

```
                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
```

```
cc_apps.tail(17)
```

```
    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
-
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
```

```
-
689  b  35.00   3.375  u  g   c   h  8.290  f  f   0  t  g  00000    0
-
```

```
cc_apps = cc_apps.replace('?',np.nan)
cc_apps.tail(17)
```

```
      0      1       2  3  4   5   6       7  8  9   10 11 12      13
14 15
673  NaN  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  -
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  -
689    b  35.00   3.375  u  g   c   h  8.290  f  f   0  t  g  00000
0  -
```

```
cc_apps.loc[[2,7,10,14]].fillna(np.mean,inplace=True)
print(cc_apps.isna().sum())
```

```
0     12
1     12
2      0
```

```
3      6
4      6
5      9
6      9
7      0
8      0
9      0
10     0
11     0
12     0
13    13
14     0
15     0
dtype: int64
```

```python
print(cc_apps[1].value_counts().index[0])
```

```
22.67
```

```python
for col in list(cc_apps):
    if cc_apps[col].dtypes == 'object':
        cc_apps = cc_apps.fillna(cc_apps[col].value_counts().index[0])

print(cc_apps.isna().sum())
```

```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
15    0
dtype: int64
```

```python
le = LabelEncoder()
for col in list(cc_apps):
    if cc_apps[col].dtypes == "object":
        cc_apps[col] = le.fit_transform(cc_apps[col])


print(cc_apps)
```

```
     0    1       2   3   4   5   6     7   8   9   10  11  12  13
14  15
0     1  156   0.000   2   1  13   8  1.25   1   1   1   0   0  68
0   0
1     0  328   4.460   2   1  11   4  3.04   1   1   6   0   0  11
560   0
2     0   89   0.500   2   1  11   4  1.50   1   0   0   0   0  96
824   0
3     1  125   1.540   2   1  13   8  3.75   1   1   5   1   0  31
3   0
4     1   43   5.625   2   1  13   8  1.71   1   0   0   0   2  37
0   0
..   ..  ...     ...  ..  ..  ..  ..   ...  ..  ..  ..  ..  ..  ..  ..
.  ..
685   1   52  10.085   3   3   5   4  1.25   0   0   0   0   0  90
0   1
686   0   71   0.750   2   1   2   8  2.00   0   1   2   1   0  67
394   1
687   0   97  13.500   3   3   6   3  2.00   0   1   1   1   0  67
1   1
688   1   20   0.205   2   1   0   8  0.04   0   0   0   0   0  96
750   1
689   1  197   3.375   2   1   2   4  8.29   0   0   0   1   0   0
0   1

[690 rows x 16 columns]
```

```
cc_apps = cc_apps.drop([11,13], axis=1)
cc_apps = cc_apps.values

X,y = cc_apps[:,0:12],cc_apps[:,13]
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.33,random_state=42)

print(X_train.shape,y_train.shape)
print(X_test.shape,y_test.shape)
```

```
(462, 12) (462,)
(228, 12) (228,)
```

```
scaler = MinMaxScaler(feature_range=(0,1))
rescaledX_train = scaler.fit_transform(X_train)
rescaledX_test = scaler.fit_transform(X_test)

print(rescaledX_train.shape)
print(rescaledX_test.shape)
```

```
(462, 12)
(228, 12)
```

```python
logreg = LogisticRegression(tol=0.01,max_iter=100)
logreg.fit(X_train,y_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
site-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: 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
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

LogisticRegression(tol=0.01)
```

```python
# Import confusion_matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

# 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: ",
accuracy_score(y_test,y_pred))

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

```
Accuracy of logistic regression classifier:  0.8421052631578947
[[95  8]
 [28 97]]
```

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

for i in range(1,21):
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(rescaledX_train,y_train.ravel())
    y_pred = model.predict(rescaledX_test)
    print(f"{i} neighbors: ",accuracy_score(y_test, y_pred))
```

```
1 neighbors:  0.9210526315789473
2 neighbors:  0.9078947368421053
3 neighbors:  0.9298245614035088
4 neighbors:  0.9254385964912281
5 neighbors:  0.9254385964912281
```

```
6 neighbors:  0.9210526315789473
7 neighbors:  0.9166666666666666
8 neighbors:  0.9122807017543859
9 neighbors:  0.9122807017543859
10 neighbors:  0.9166666666666666
11 neighbors:  0.9166666666666666
12 neighbors:  0.9166666666666666
13 neighbors:  0.9210526315789473
14 neighbors:  0.9254385964912281
15 neighbors:  0.9254385964912281
16 neighbors:  0.9298245614035088
17 neighbors:  0.9342105263157895
18 neighbors:  0.9342105263157895
19 neighbors:  0.9385964912280702
20 neighbors:  0.9298245614035088
```

```python
cc_apps = pd.read_csv("cc_approvals.data", header=None)
cc_apps.head()
```

```
   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  +
```

```python
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


    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
-
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
-
689  b  35.00   3.375  u  g   c   h  8.290  f  f   0  t  g  00000    0
-
```

```
from sklearn.model_selection import train_test_split

print(cc_apps.corr(numeric_only=True))

cc_apps = cc_apps.drop([11,13],axis = 1)

cc_apps_train, cc_apps_test =
train_test_split(cc_apps,test_size=0.33,random_state=42)
```

```
          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
```

```
# 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)

print(cc_apps_train)
```

```
      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   NaN  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  NaN  NaN  NaN  NaN   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]

# Impute the missing values with mean imputation
cc_apps_train.fillna(cc_apps_train.mean(numeric_only = True),
inplace=True)
cc_apps_test.fillna(cc_apps_train.mean(numeric_only=True),
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
```

```
for col in cc_apps_train.columns:
    if cc_apps_train[col].dtypes == 'object':
        cc_apps_train = 
cc_apps_train.fillna(cc_apps_train[col].value_counts().index[0])
        cc_apps_test = 
cc_apps_test.fillna(cc_apps_train[col].value_counts().index[0])

print(cc_apps_train.isnull().sum())
print(cc_apps_test.isnull().sum())
```

```
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
```

```
# Convert the categorical features in the train and test sets 
independently
print(cc_apps_train)
cc_apps_train = pd.get_dummies(cc_apps_train) # try with argument 
dtype = int
cc_apps_test = pd.get_dummies(cc_apps_test)
```

```python
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)
```

```
    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
\
382  2.500   4.500   0   456   True  False    False    False    False

137  2.750   4.250   6     0  False   True    False    False    False

346  1.500   0.250   0   122  False   True    False    False    False

326  1.085   0.040   0   179  False   True    False    False    False

33   5.125   5.000   0  4000   True  False    False    False    False

..     ...     ...  ..   ...    ...    ...      ...      ...      ...

71   4.000  12.500   0     0  False   True    False    False    False

106  1.165   0.500   0     0  False   True    False    False    False

270  0.000   0.000   0     0  False   True    False    False    False

435  0.000   0.000   4     1  False   True    False    False    False

102  5.000   0.375   2    38  False   True    False    False    False


     1_16.00  ...    6_z    8_f    8_t    9_f    9_t   12_g   12_p
12_s  \
382    False  ...  False   True  False   True  False   True  False
False
137    False  ...  False  False   True  False   True   True  False
False
346    False  ...  False   True  False   True  False   True  False
```

```
False
326    False  ...  False   True  False   True  False   True  False
False
33     False  ...  False  False   True   True  False   True  False
False
..       ...  ...    ...    ...    ...    ...    ...    ...    ...
...
71     False  ...  False  False   True   True  False   True  False
False
106    False  ...  False  False   True   True  False  False  False
True
270    False  ...  False   True  False   True  False  False   True
False
435    False  ...  False   True  False  False   True   True  False
False
102    False  ...  False  False   True  False   True   True  False
False

      15_+   15_-
382  False   True
137   True  False
346  False   True
326  False   True
33    True  False
..     ...    ...
71   False   True
106  False   True
270   True  False
435  False   True
102  False   True

[462 rows x 334 columns]
```

```python
# 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[:, [-1]].values
X_test, y_test = cc_apps_test.iloc[:, :-1].values,
cc_apps_test.iloc[:, [-1]].values

# 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
```

```python
# Instantiate a LogisticRegression classifier with default parameter
values
logreg = LogisticRegression(tol=0.1,max_iter=100)

# Fit logreg to the train set
logreg.fit(rescaledX_train,y_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
site-packages/sklearn/utils/validation.py:1408: 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().
  y = column_or_1d(y, warn=True)
```

```
LogisticRegression(tol=0.1)
```

```python
# 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_test,y_test))

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

```
Accuracy of logistic regression classifier:  0.9254385964912281
```

```
array([[ 95,   8],
       [  9, 116]])
```

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

for i in range(1,21):
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(rescaledX_train,y_train.ravel())
    y_pred = model.predict(rescaledX_test)
    print(f"{i} neighbors: ",accuracy_score(y_test, y_pred))
```

```
1 neighbors:  0.9210526315789473
2 neighbors:  0.9078947368421053
3 neighbors:  0.9298245614035088
4 neighbors:  0.9254385964912281
5 neighbors:  0.9254385964912281
6 neighbors:  0.9210526315789473
7 neighbors:  0.9166666666666666
8 neighbors:  0.9122807017543859
9 neighbors:  0.9122807017543859
```

```
10 neighbors:  0.9166666666666666
11 neighbors:  0.9166666666666666
12 neighbors:  0.9166666666666666
13 neighbors:  0.9210526315789473
14 neighbors:  0.9254385964912281
15 neighbors:  0.9254385964912281
16 neighbors:  0.9298245614035088
17 neighbors:  0.9342105263157895
18 neighbors:  0.9342105263157895
19 neighbors:  0.9385964912280702
20 neighbors:  0.9298245614035088
```