10. Implement KNN classification algorithm with an appropriate dataset and analyze the results.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# Load the Iris dataset
X, y = load_iris(return_X_y=True)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Train and predict with kNN
knn = KNeighborsClassifier(n_neighbors=3).fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Analyze the results
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}\n")
print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_pred)}\n")
print(f"Classification Report:\n{classification_report(y_test, y_pred)}")

Output:
Accuracy: 1.00

Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

8. Implement the Naive Bayesian classifier for a sample training data set stored as a .CSV file.

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report
from sklearn.naive_bayes import GaussianNB

data=load_breast_cancer()
x=data.data
y=data.target

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=
42)

model=GaussianNB()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)

print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))

Output:
0.9736842105263158
              precision    recall  f1-score   support

           0       1.00      0.93      0.96        43
           1       0.96      1.00      0.98        71

    accuracy                           0.97       114
   macro avg       0.98      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

7. Demonstrate the text classifier using Naive Bayes classifier algorithm.

```
#NAIVE BAYES CLASSIFIER
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_20newsgroups_vectorized
from sklearn.metrics import accuracy_score,classification_report

data=fetch_20newsgroups_vectorized()

x=data.data
y=data.target

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=
42)

clf=MultinomialNB()
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)


print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))

Output:
0.7445868316394167
              precision    recall  f1-score   support

           0       0.88      0.38      0.53        93
           1       0.82      0.62      0.71       118
           2       0.89      0.66      0.76       128
           3       0.62      0.77      0.69       120
           4       0.72      0.82      0.77       102
           5       0.88      0.73      0.80       124
           6       0.88      0.66      0.76       112
           7       0.65      0.95      0.77       112
           8       0.91      0.88      0.90       118
           9       0.97      0.93      0.95       125
          10       0.95      0.94      0.94       117
          11       0.52      0.97      0.68       120
          12       0.92      0.50      0.65       138
          13       0.87      0.90      0.88       118
          14       0.90      0.87      0.88       122
          15       0.38      0.98      0.55       120
          16       0.81      0.84      0.83       105
          17       0.95      0.85      0.90       115
          18       1.00      0.16      0.27        90
          19       1.00      0.02      0.03        66
```

```
    accuracy                           0.74      2263
   macro avg       0.83      0.72      0.71      2263
weighted avg       0.82      0.74      0.73      2263
```

6. Implement Random Forest classifier using python programming.

```
#RANDOM FOREST CLASSIFIER
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,classification_report


data=load_breast_cancer()

x=data.data
y=data.target

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=
42)


clf=RandomForestClassifier(n_estimators=500,max_leaf_nodes=16,n_jobs=-1)

clf.fit(x_train,y_train)

y_pred=clf.predict(x_test)


print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))

Output:
0.9649122807017544
              precision    recall  f1-score   support

           0       0.98      0.93      0.95        43
           1       0.96      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

5. Implement and demonstrate the working of the Decision Tree algorithm.

```
# DECISION TREE
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,classification_report


data=load_iris()

x=data.data
y=data.target


x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=
42)
clf=DecisionTreeClassifier().fit(x_train,y_train)
y_pred=clf.predict(x_test)

print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))

Output:
1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

4. Demonstrate the working of SVM classifier for a suitable dataset.

```
# SVM
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report


data=load_breast_cancer()
x=data.data
y=data.target

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=
42)

svm=SVC(kernel='linear').fit(x_train,y_train)
y_pred=svm.predict(x_test)


accuracy=accuracy_score(y_test,y_pred)
print("PREDICTED:", y_pred)
print("CONFUSION MATRIX: \n",confusion_matrix(y_test,y_pred))
print("ACCURACY:",accuracy)
print(classification_report(y_test,y_pred))

Output:
PREDICTED: [1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1
1 1 1 0
 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0
 1 1 1 1 1 1 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0
 1 0 0]
CONFUSION MATRIX:
 [[39  4]
 [ 1 70]]
ACCURACY: 0.956140350877193
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

1.Implement and demonstrate the Find-S algorithm for finding the most specific hypothesis.

```
import csv# Load data from CSV file
training_data = []
step=0
with open('enjoysport.csv', newline='') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        training_data.append(row)

# Initialize hypothesis with '?'
hypothesis = ['0'] * (len(training_data[0]) - 1)

# Apply Find-S algorithm
for example in training_data:
    if example[-1] == '1':
        for i in range(len(hypothesis)):
            if hypothesis[i] != example[i]:
                if hypothesis[i] == '0':
                    hypothesis[i] = example[i]
                else:
                    hypothesis[i] = '?'
    print('STEP',step,': ')
    print("Specific hypothesis:", hypothesis)
    step+=1
    print('\n')

print("Final hypothesis:", hypothesis)

Output:
STEP 0 :
Specific hypothesis: ['0', '0', '0', '0', '0', '0']


STEP 1 :
Specific hypothesis: ['0', '0', '0', '0', '0', '0']


STEP 2 :
Specific hypothesis: ['0', '0', '0', '0', '0', '0']


STEP 3 :
Specific hypothesis: ['0', '0', '0', '0', '0', '0']

STEP 4 :
Specific hypothesis: ['0', '0', '0', '0', '0', '0']

Final hypothesis: ['0', '0', '0', '0', '0', '0']
```

2.Implement and demonstrate the Candidate Elimination algorithm using a data set stored as a .CSV file.

```
import csv

# Load data from CSV file
training_data = []
step = 0
with open('enjoysport.csv', newline='') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        training_data.append(row)

# Initialize hypotheses
G = ['0'] * (len(training_data[0]) - 1)
S = []

# Apply CEA
for example in training_data:
    if example[-1] == '1':
        for i in range(len(G)):
            if G[i] != example[i]:
                G[i] = example[i] if G[i] == '0' else '?'
        S = [h for h in S if not any(h[j] != '?' and G[j] != h[j] for j in
range(len(G)))]
    else:
        S.extend(['?' if z != i else G[i] for z in range(len(G))] for i in
range(len(G)) if G[i] != example[i] and G[i] != '?')

    print(f'STEP {step}:')
    print("Specific hypothesis:", G)
    print("General hypothesis:", S)
    print('\n')
    step += 1

print("Final Specific hypothesis:", G)
print("Final General hypothesis:", S)

Output:
STEP 0:
Specific hypothesis: ['0', '0', '0', '0', '0', '0']
General hypothesis: [['0', '?', '?', '?', '?', '?'], ['?', '0', '?', '?', '?',
'?'], ['?', '?', '0', '?', '?', '?'], ['?', '?', '?', '0', '?', '?'], ['?',
'?', '?', '?', '0', '?'], ['?', '?', '?', '?', '?', '0']]



STEP 1:
Specific hypothesis: ['0', '0', '0', '0', '0', '0']
```

```
General hypothesis: [['0', '?', '?', '?', '?', '?'], ['?', '0', '?', '?', '?',
'?'], ['?', '?', '0', '?', '?', '?'], ['?', '?', '?', '0', '?', '?'], ['?',
'?', '?', '?', '0', '?'], ['?', '?', '?', '?', '?', '0'], ['0', '?', '?', '?',
'?', '?'], ['?', '0', '?', '?', '?', '?'], ['?', '?', '0', '?', '?', '?'],
['?', '?', '?', '0', '?', '?'], ['?', '?', '?', '?', '0', '?'], ['?', '?',
'?', '?', '?', '0']]


STEP 2:
Specific hypothesis: ['0', '0', '0', '0', '0', '0']
General hypothesis: [['0', '?', '?', '?', '?', '?'], ['?', '0', '?', '?', '?',
'?'], ['?', '?', '0', '?', '?', '?'], ['?', '?', '?', '0', '?', '?'], ['?',
'?', '?', '?', '0', '?'], ['?', '?', '?', '?', '?', '0'], ['0', '?', '?', '?',
'?', '?'], ['?', '0', '?', '?', '?', '?'], ['?', '?', '0', '?', '?', '?'],
['?', '?', '?', '0', '?', '?'], ['?', '?', '?', '?', '0', '?'], ['?', '?',
'?', '?', '?', '0'], ['0', '?', '?', '?', '?', '?'], ['?', '0', '?', '?', '?',
'?'], ['?', '?', '0', '?', '?', '?'], ['?', '?', '?', '0', '?', '?'], ['?',
'?', '?', '?', '0', '?'], ['?', '?', '?', '?', '?', '0']]


STEP 3:
Specific hypothesis: ['0', '0', '0', '0', '0', '0']
General hypothesis: [['0', '?', '?', '?', '?', '?'], ['?', '0', '?', '?', '?',
'?'], ['?', '?', '0', '?', '?', '?'], ['?', '?', '?', '0', '?', '?'], ['?',
'?', '?', '?', '0', '?'], ['?', '?', '?', '?', '?', '0'], ['0', '?', '?', '?',
'?', '?'], ['?', '0', '?', '?', '?', '?'], ['?', '?', '0', '?', '?', '?'],
['?', '?', '?', '0', '?', '?'], ['?', '?', '?', '?', '0', '?'], ['?', '?',
'?', '?', '?', '0'], ['0', '?', '?', '?', '?', '?'], ['?', '0', '?', '?', '?',
'?'], ['?', '?', '0', '?', '?', '?'], ['?', '?', '?', '0', '?', '?'], ['?',
'?', '?', '?', '0', '?'], ['?', '?', '?', '?', '?', '0'], ['0', '?', '?', '?',
'?', '?'], ['?', '0', '?', '?', '?', '?'], ['?', '?', '0', '?', '?', '?'],
['?', '?', '?', '0', '?', '?'], ['?', '?', '?', '?', '0', '?'], ['?', '?',
'?', '?', '?', '0']]


STEP 4:
Specific hypothesis: ['0', '0', '0', '0', '0', '0']
General hypothesis: [['0', '?', '?', '?', '?', '?'], ['?', '0', '?', '?', '?',
'?'], ['?', '?', '0', '?', '?', '?'], ['?', '?', '?', '0', '?', '?'], ['?',
'?', '?', '?', '0', '?'], ['?', '?', '?', '?', '?', '0'], ['0', '?', '?', '?',
'?', '?'], ['?', '0', '?', '?', '?', '?'], ['?', '?', '0', '?', '?', '?'],
['?', '?', '?', '0', '?', '?'], ['?', '?', '?', '?', '0', '?'], ['?', '?',
'?', '?', '?', '0'], ['0', '?', '?', '?', '?', '?'], ['?', '0', '?', '?', '?',
'?'], ['?', '?', '0', '?', '?', '?'], ['?', '?', '?', '0', '?', '?'], ['?',
'?', '?', '?', '0', '?'], ['?', '?', '?', '?', '?', '0'], ['0', '?', '?', '?',
'?', '?'], ['?', '0', '?', '?', '?', '?'], ['?', '?', '0', '?', '?', '?'],
['?', '?', '?', '0', '?', '?'], ['?', '?', '?', '?', '0', '?'], ['?', '?',
'?', '?', '?', '0'], ['0', '?', '?', '?', '?', '?'], ['?', '0', '?', '?', '?',
```

```
'?'], ['?', '?', '0', '?', '?', '?'], ['?', '?', '?', '0', '?', '?'], ['?',
'?', '?', '?', '0', '?'], ['?', '?', '?', '?', '?', '0']]


Final Specific hypothesis: ['0', '0', '0', '0', '0', '0']
Final General hypothesis: [['0', '?', '?', '?', '?', '?'], ['?', '0', '?',
'?', '?', '?'], ['?', '?', '0', '?', '?', '?'], ['?', '?', '?', '0', '?',
'?'], ['?', '?', '?', '?', '0', '?'], ['?', '?', '?', '?', '?', '0'], ['0',
'?', '?', '?', '?', '?'], ['?', '0', '?', '?', '?', '?'], ['?', '?', '0', '?',
'?', '?'], ['?', '?', '?', '0', '?', '?'], ['?', '?', '?', '?', '0', '?'],
['?', '?', '?', '?', '?', '0'], ['0', '?', '?', '?', '?', '?'], ['?', '0',
'?', '?', '?', '?'], ['?', '?', '0', '?', '?', '?'], ['?', '?', '?', '0', '?',
'?'], ['?', '?', '?', '?', '0', '?'], ['?', '?', '?', '?', '?', '0'], ['0',
'?', '?', '?', '?', '?'], ['?', '0', '?', '?', '?', '?'], ['?', '?', '0', '?',
'?', '?'], ['?', '?', '?', '0', '?', '?'], ['?', '?', '?', '?', '0', '?'],
['?', '?', '?', '?', '?', '0'], ['0', '?', '?', '?', '?', '?'], ['?', '0',
'?', '?', '?', '?'], ['?', '?', '0', '?', '?', '?'], ['?', '?', '?', '0', '?',
'?'], ['?', '?', '?', '?', '0', '?'], ['?', '?', '?', '?', '?', '0']]
```

3. Demonstrate data Preprocessing (Data Cleaning, Integration and Transformation) operations on a suitable data.

```python
# Import the necessary libraries
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder, LabelEncoder

# Load the dataset
tennis = pd.read_csv('iris.csv')

# Separate the features and target
X = tennis.iloc[:, 0:4]
y = tennis.iloc[:, 4:5]

print("FEATURES")
print(X)
print("TARGET")
print(y)

# Data Cleaning - Features (Ordinal Encoder) and Targets (Label Encoder)
ordinal_encoder = OrdinalEncoder()  # for cleaning the features
label_encoder = LabelEncoder()  # for cleaning the targets

X_ordinal_encoded = ordinal_encoder.fit_transform(X)
print("FEATURES\n", X_ordinal_encoded)

y_label_encoded = label_encoder.fit_transform(y.values.ravel())
print("TARGET\n", y_label_encoded)
```

```
Output:
FEATURES
     5.1  3.5  1.4  0.2
0    4.9  3.0  1.4  0.2
1    4.7  3.2  1.3  0.2
2    4.6  3.1  1.5  0.2
3    5.0  3.6  1.4  0.2
4    5.4  3.9  1.7  0.4
..   ...  ...  ...  ...
144  6.7  3.0  5.2  2.3
145  6.3  2.5  5.0  1.9
146  6.5  3.0  5.2  2.0
147  6.2  3.4  5.4  2.3
148  5.9  3.0  5.1  1.8

[149 rows x 4 columns]
TARGET
        Iris-setosa
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
144  Iris-virginica
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica

[149 rows x 1 columns]
FEATURES
 [[ 6.  9.  4.  1.]
 [ 4. 11.  3.  1.]
 [ 3. 10.  5.  1.]
 [ 7. 15.  4.  1.]
 [11. 18.  7.  3.]
 [ 3. 13.  4.  2.]
 [ 7. 13.  5.  1.]
 [ 1.  8.  4.  1.]
 [ 6. 10.  5.  0.]
 [11. 16.  5.  1.]
 [ 5. 13.  6.  1.]
 [ 5.  9.  4.  0.]
 [ 0.  9.  1.  0.]
 [15. 19.  2.  1.]
 [14. 22.  5.  3.]
 [11. 18.  3.  3.]
```

```
 [ 8. 14.  4.  2.]
 [14. 17.  7.  2.]
 [ 8. 17.  5.  2.]
 [11. 13.  7.  1.]
 [ 8. 16.  5.  3.]
 [ 3. 15.  0.  1.]
 [ 8. 12.  7.  4.]
 [ 5. 13.  8.  1.]
 [ 7.  9.  6.  1.]
 [ 7. 13.  6.  3.]
 [ 9. 14.  5.  1.]
 [ 9. 13.  4.  1.]
 [ 4. 11.  6.  1.]
 [ 5. 10.  6.  1.]
 [11. 13.  5.  3.]
 [ 9. 20.  5.  0.]
 [12. 21.  4.  1.]
 [ 6. 10.  5.  0.]
 [ 7. 11.  2.  1.]
 [12. 14.  3.  1.]
 [ 6. 10.  5.  0.]
 [ 1.  9.  3.  1.]
 [ 8. 13.  5.  1.]
 [ 7. 14.  3.  2.]
 [ 2.  2.  3.  2.]
 [ 1. 11.  3.  1.]
 [ 7. 14.  6.  5.]
 [ 8. 17.  8.  3.]
 [ 5.  9.  4.  2.]
 [ 8. 17.  6.  1.]
 [ 3. 11.  4.  1.]
 [10. 16.  5.  1.]
 [ 7. 12.  4.  1.]
 [27. 11. 23. 10.]
 [21. 11. 21. 11.]
 [26. 10. 25. 11.]
 [12.  2. 16.  9.]
 [22.  7. 22. 11.]
 [14.  7. 21.  9.]
 [20. 12. 23. 12.]
 [ 6.  3. 10.  6.]
 [23.  8. 22.  9.]
 [ 9.  6. 15. 10.]
 [ 7.  0. 11.  6.]
 [16.  9. 18. 11.]
 [17.  1. 16.  6.]
 [18.  8. 23. 10.]
 [13.  8. 12.  9.]
```

```
 [24. 10. 20. 10.]
 [13.  9. 21. 11.]
 [15.  6. 17.  6.]
 [19.  1. 21. 11.]
 [13.  4. 15.  7.]
 [16. 11. 24. 14.]
 [18.  7. 16.  9.]
 [20.  4. 25. 11.]
 [18.  7. 23.  8.]
 [21.  8. 19.  9.]
 [23.  9. 20. 10.]
 [25.  7. 24. 10.]
 [24.  9. 26. 13.]
 [17.  8. 21. 11.]
 [14.  5. 11.  6.]
 [12.  3. 14.  7.]
 [12.  3. 13.  6.]
 [15.  6. 15.  8.]
 [17.  6. 27. 12.]
 [11.  9. 21. 11.]
 [17. 13. 21. 12.]
 [24. 10. 23. 11.]
 [20.  2. 20.  9.]
 [13.  9. 17.  9.]
 [12.  4. 16.  9.]
 [12.  5. 20.  8.]
 [18.  9. 22. 10.]
 [15.  5. 16.  8.]
 [ 7.  2. 10.  6.]
 [13.  6. 18.  9.]
 [14.  9. 18.  8.]
 [14.  8. 18.  9.]
 [19.  8. 19.  9.]
 [ 8.  4.  9.  7.]
 [14.  7. 17.  9.]
 [20. 12. 36. 21.]
 [15.  6. 27. 15.]
 [28.  9. 35. 17.]
 [20.  8. 32. 14.]
 [22.  9. 34. 18.]
 [32.  9. 40. 17.]
 [ 6.  4. 21. 13.]
 [30.  8. 38. 14.]
 [24.  4. 34. 14.]
 [29. 15. 37. 21.]
 [22. 11. 27. 16.]
 [21.  6. 29. 15.]
 [25.  9. 31. 17.]
```

```
 [14.  4. 26. 16.]
 [15.  7. 27. 20.]
 [21. 11. 29. 19.]
 [22.  9. 31. 14.]
 [33. 17. 41. 18.]
 [33.  5. 42. 19.]
 [17.  1. 26. 11.]
 [26. 11. 33. 19.]
 [13.  7. 25. 16.]
 [33.  7. 41. 16.]
 [20.  6. 25. 14.]
 [24. 12. 33. 17.]
 [29. 11. 36. 14.]
 [19.  7. 24. 14.]
 [18.  9. 25. 14.]
 [21.  7. 32. 17.]
 [29.  9. 34. 12.]
 [31.  7. 37. 15.]
 [34. 17. 39. 16.]
 [21.  7. 32. 18.]
 [20.  7. 27. 11.]
 [18.  5. 32. 10.]
 [33.  9. 37. 19.]
 [20. 13. 32. 20.]
 [21. 10. 31. 14.]
 [17.  9. 24. 14.]
 [26. 10. 30. 17.]
 [24. 10. 32. 20.]
 [26. 10. 27. 19.]
 [15.  6. 27. 15.]
 [25. 11. 35. 19.]
 [24. 12. 33. 21.]
 [24.  9. 28. 19.]
 [20.  4. 26. 15.]
 [22.  9. 28. 16.]
 [19. 13. 30. 19.]
 [16.  9. 27. 14.]]
TARGET
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2]
```