Canser Classifier

Imports:

In [1]:
```
from src.homeworks.homework2.KNNClassifier import KNNClassifier
from src.homeworks.homework2.scalers import MinMaxScaler, MaxAbsScaler, S
from src.homeworks.homework2.score import MetricCalculator
from src.homeworks.homework2.train_test_split import train_test_split

import pandas as pd
import matplotlib.pyplot as plt
```

Read cancer.csv:

In [2]:
```
data = pd.read_csv("src/homeworks/homework2/notebooks/cancer.csv")
data.describe()
```

Out[2]:

	1	2	3	4	5	6	
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0

8 rows × 30 columns

Replace "M" to 1 and "B" to 0 in "label"

In [3]:
```
data["label"] = data["label"].replace({"M": 1, "B": 0})
data.describe()
```

```
/tmp/ipykernel_91621/2696528524.py:1: FutureWarning: Downcasting behavior
in `replace` is deprecated and will be removed in a future version. To ret
ain the old behavior, explicitly call `result.infer_objects(copy=False)`.
To opt-in to the future behavior, set `pd.set_option('future.no_silent_dow
ncasting', True)`
  data["label"] = data["label"].replace({"M": 1, "B": 0})
```

Out[3]:

	label	1	2	3	4	5	
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569
mean	0.372583	14.127292	19.289649	91.969033	654.889104	0.096360	0
std	0.483918	3.524049	4.301036	24.298981	351.914129	0.014064	0
min	0.000000	6.981000	9.710000	43.790000	143.500000	0.052630	0
25%	0.000000	11.700000	16.170000	75.170000	420.300000	0.086370	0
50%	0.000000	13.370000	18.840000	86.240000	551.100000	0.095870	0
75%	1.000000	15.780000	21.800000	104.100000	782.700000	0.105300	0
max	1.000000	28.110000	39.280000	188.500000	2501.000000	0.163400	0

8 rows × 31 columns

Divide the dataset into X and y:

In [4]:
```
y = data["label"].to_numpy()
X = data.drop(columns=['label']).to_numpy()
X
```

Out[4]:
```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]], shape=(569, 30))
```

Creat a function that performs normalization, splitting into train and test, and counts metrics.

In [5]:
```
def get_score(X, y, test_size: float = 0.15, shuffle: bool = True, random
    scaler = scaler()
    X_new = scaler.fit_transform(X)

    X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_si

    accuracy = []
    f1_score = []

    for k in range(1, 21):
        knn = KNNClassifier(k, 5)
        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_test)

        metric = MetricCalculator(y_pred, y_test)
        accuracy.append(metric.accuracy())
```

```
        f1_score.append(metric.f1_score())

    return accuracy, f1_score
```

MinMaxScaler

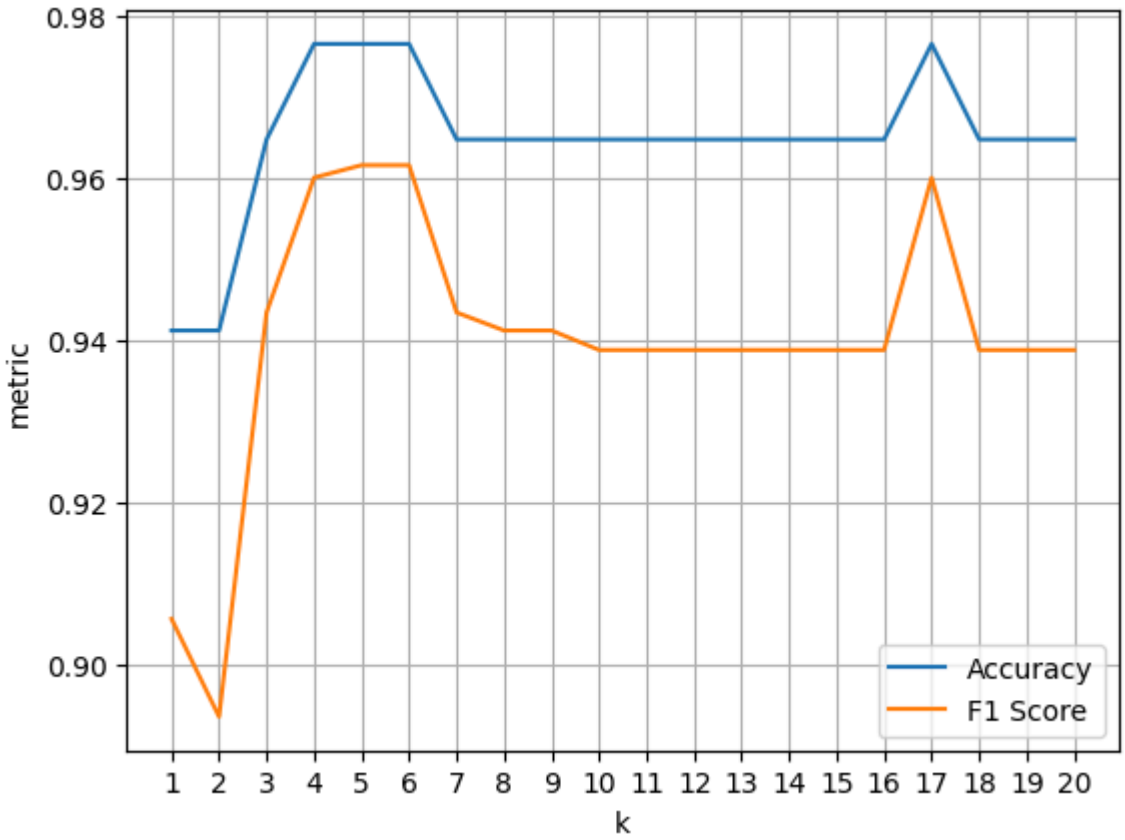
In [6]:
```
accuracy, f1_score = get_score(X, y, test_size=0.15, shuffle=True, random
```

In [7]:
```
k_values = range(1, 21)
plt.plot(k_values, accuracy, label='Accuracy', linestyle='-')
plt.plot(k_values, f1_score, label='F1 Score', linestyle='-')

plt.xlabel('k')
plt.ylabel('metric')
plt.xticks(k_values)

plt.grid(True)
plt.legend()

plt.show()
```



MaxAbsScaler

In [8]:
```
accuracy, f1_score = get_score(X, y, test_size=0.15, shuffle=True, random
```

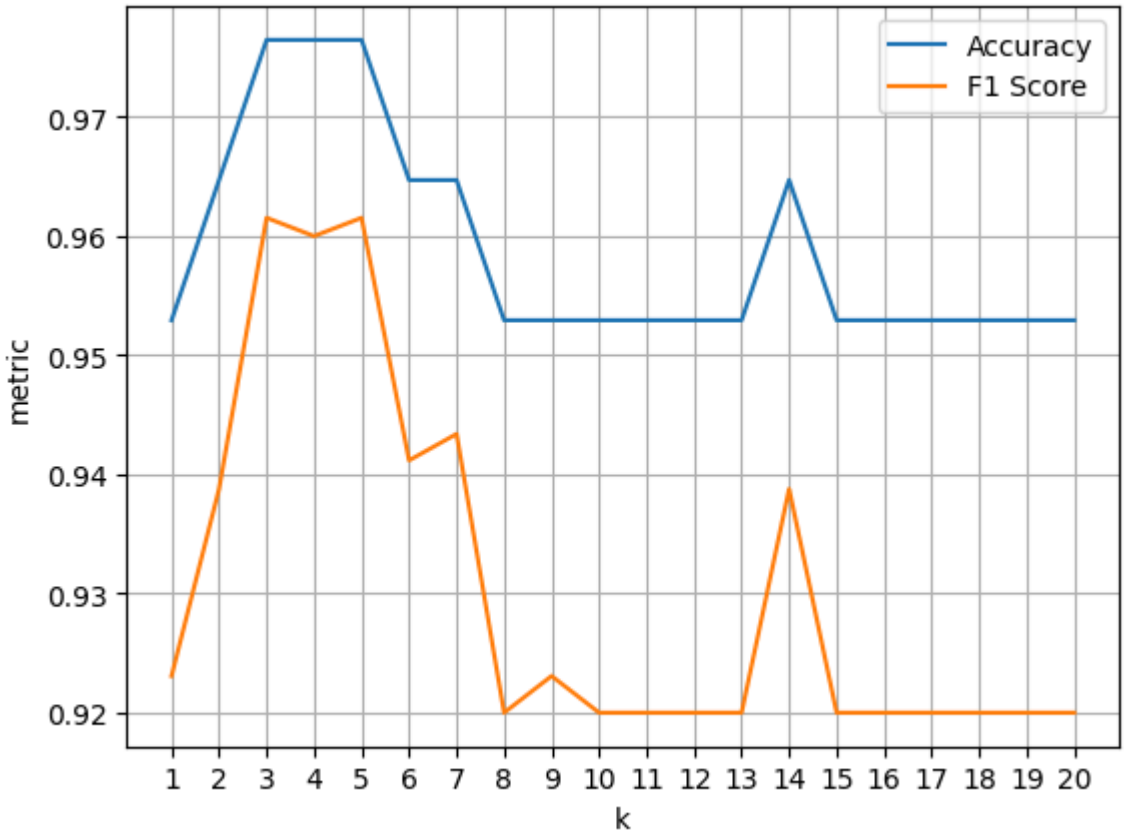
In [9]:
```
k_values = range(1, 21)
plt.plot(k_values, accuracy, label='Accuracy', linestyle='-')
plt.plot(k_values, f1_score, label='F1 Score', linestyle='-')

plt.xlabel('k')
```

```
plt.ylabel('metric')
plt.xticks(k_values)

plt.grid(True)
plt.legend()

plt.show()
```



StandardScaler

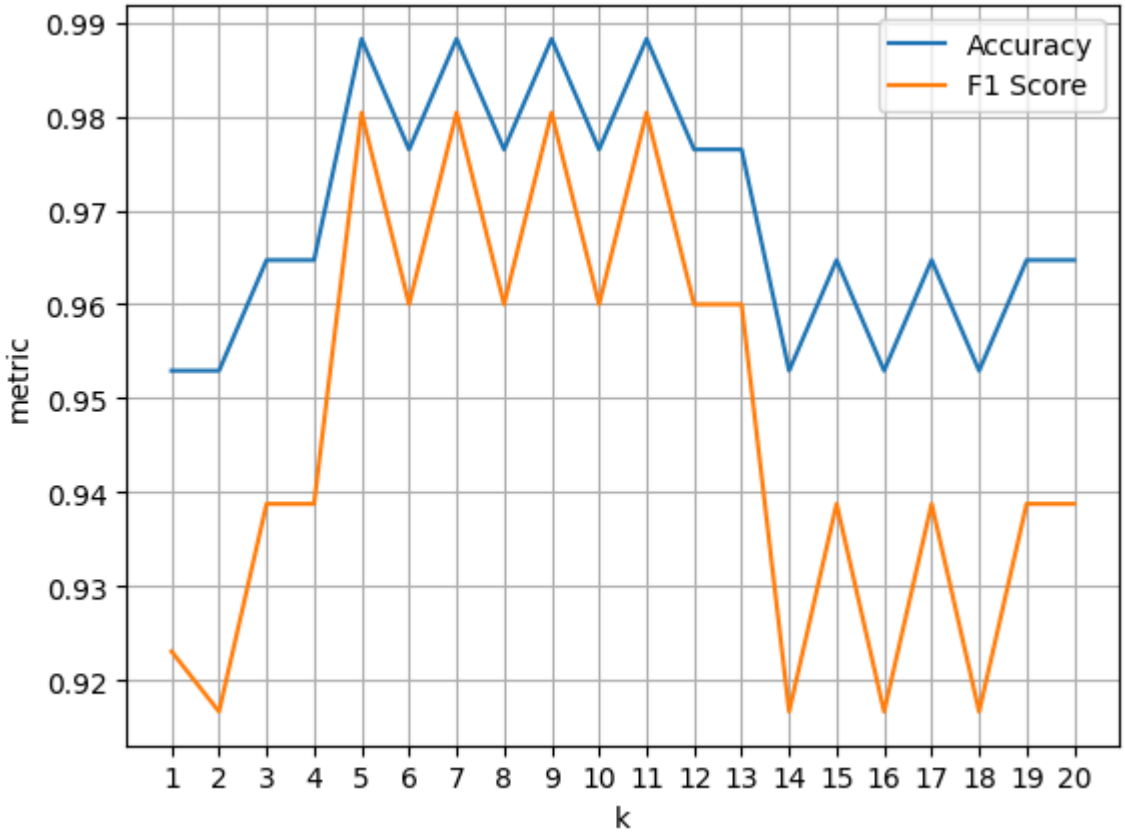
In [10]:
```
accuracy, f1_score = get_score(X, y, test_size=0.15, shuffle=True, random
```

In [11]:
```
k_values = range(1, 21)
plt.plot(k_values, accuracy, label='Accuracy', linestyle='-')
plt.plot(k_values, f1_score, label='F1 Score', linestyle='-')

plt.xlabel('k')
plt.ylabel('metric')
plt.xticks(k_values)

plt.grid(True)
plt.legend()

plt.show()
```


Accuracy
F1 Score
metric
k
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20