In [27]:
```
from sklearn.model_selection import train_test_split
```

In [28]:
```
# We need to predict Survived column, so we take it as y
# and drop it from X
y = titanic_df['Survived']
# Took everything except Survived
X = titanic_df.drop('Survived', axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [29]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd
```

In [30]:
```
# Decision Tree
tree = DecisionTreeClassifier(random_state=42)
tree.fit(X_train, y_train)
y_pred_tree = tree.predict(X_test)
acc_tree = accuracy_score(y_test, y_pred_tree)
print("Decision Tree Accuracy:", acc_tree)
print(classification_report(y_test, y_pred_tree))
```

```
Decision Tree Accuracy: 0.7932960893854749
              precision    recall  f1-score   support

           0       0.81      0.84      0.83       105
           1       0.76      0.73      0.74        74

    accuracy                           0.79       179
   macro avg       0.79      0.78      0.79       179
weighted avg       0.79      0.79      0.79       179
```

In [31]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
acc_knn = accuracy_score(y_test, y_pred_knn)
print("KNN Accuracy:", acc_knn)
print(classification_report(y_test, y_pred_knn))
```

```
KNN Accuracy: 0.7150837988826816
              precision    recall  f1-score   support

           0       0.73      0.82      0.77       105
           1       0.69      0.57      0.62        74

    accuracy                           0.72       179
   macro avg       0.71      0.69      0.70       179
weighted avg       0.71      0.72      0.71       179
```

In [32]:
```
#Baseline
dummy = DummyClassifier(strategy="most_frequent")
dummy.fit(X_train, y_train)
```

```
y_pred_dummy = dummy.predict(X_test)

acc_dummy = accuracy_score(y_test, y_pred_dummy)
print("DummyClassifier Accuracy:", acc_dummy)
print(classification_report(y_test, y_pred_dummy))
```

```
DummyClassifier Accuracy: 0.5865921787709497
              precision    recall  f1-score   support

           0       0.59      1.00      0.74       105
           1       0.00      0.00      0.00        74

    accuracy                           0.59       179
   macro avg       0.29      0.50      0.37       179
weighted avg       0.34      0.59      0.43       179
```

```
c:\Users\sofiy\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\met
rics\_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and b
eing set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\sofiy\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\met
rics\_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and b
eing set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\sofiy\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\met
rics\_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and b
eing set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In [33]:

```
results_df = pd.DataFrame({
    'Model': ['Decision Tree', 'KNN', 'DummyClassifier'],
    'Accuracy': [acc_tree, acc_knn, acc_dummy]
})

print("\nCompare:")
print(results_df)
```

```
Compare:
             Model  Accuracy
0    Decision Tree  0.793296
1              KNN  0.715084
2  DummyClassifier  0.586592
```

W przeprowadzonych testach najlepszy wynik uzyskało drzewo decyzyjne (ok. 79%), wyprzedzając KNN (ok. 71%). Obie metody radzą sobie znacznie lepiej od prostego baseline (ok. 59%), co pokazuje, że modele faktycznie coś się nauczyły z danych.

(Baseline zawsze wybierał klasę 0, bo była najczęstsza w danych testowych (~58%), dlatego jego wynik jest niski i służy tylko jako punkt odniesienia dla oceny prawdziwych modeli.)

In [35]:

```
#  hiperparametry Decision Tree
max_depth_values = range(1, 21)
```

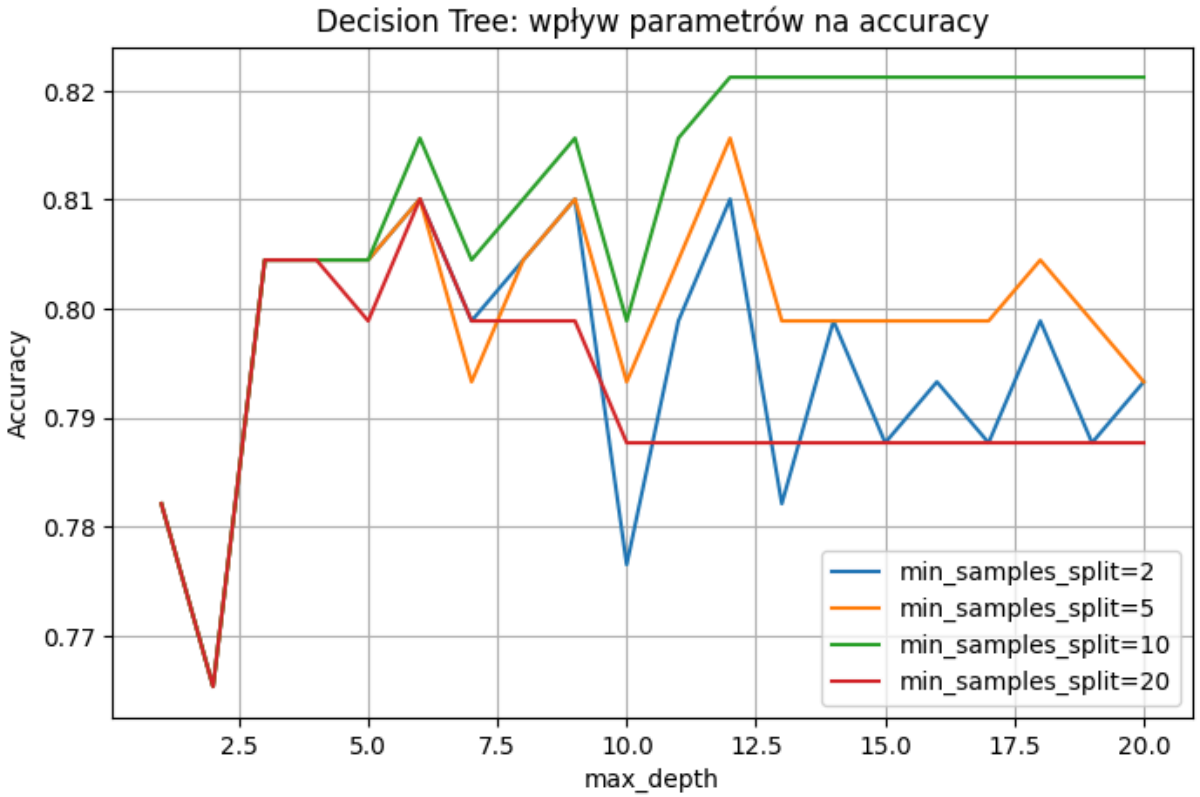
```
min_samples_split_values = [2, 5, 10, 20]

results_tree = []
for depth in max_depth_values:
    for split in min_samples_split_values:
        model = DecisionTreeClassifier(max_depth=depth, min_samples_split=split, ra
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results_tree.append((depth, split, acc))

df_tree = pd.DataFrame(results_tree, columns=["max_depth", "min_samples_split", "ac

plt.figure(figsize=(8, 5))
for split in min_samples_split_values:
    subset = df_tree[df_tree["min_samples_split"] == split]
    plt.plot(subset["max_depth"], subset["accuracy"], label=f"min_samples_split={sp

plt.xlabel("max_depth")
plt.ylabel("Accuracy")
plt.title("Decision Tree: wpływ parametrów na accuracy")
plt.legend()
plt.grid(True)
plt.show()
```

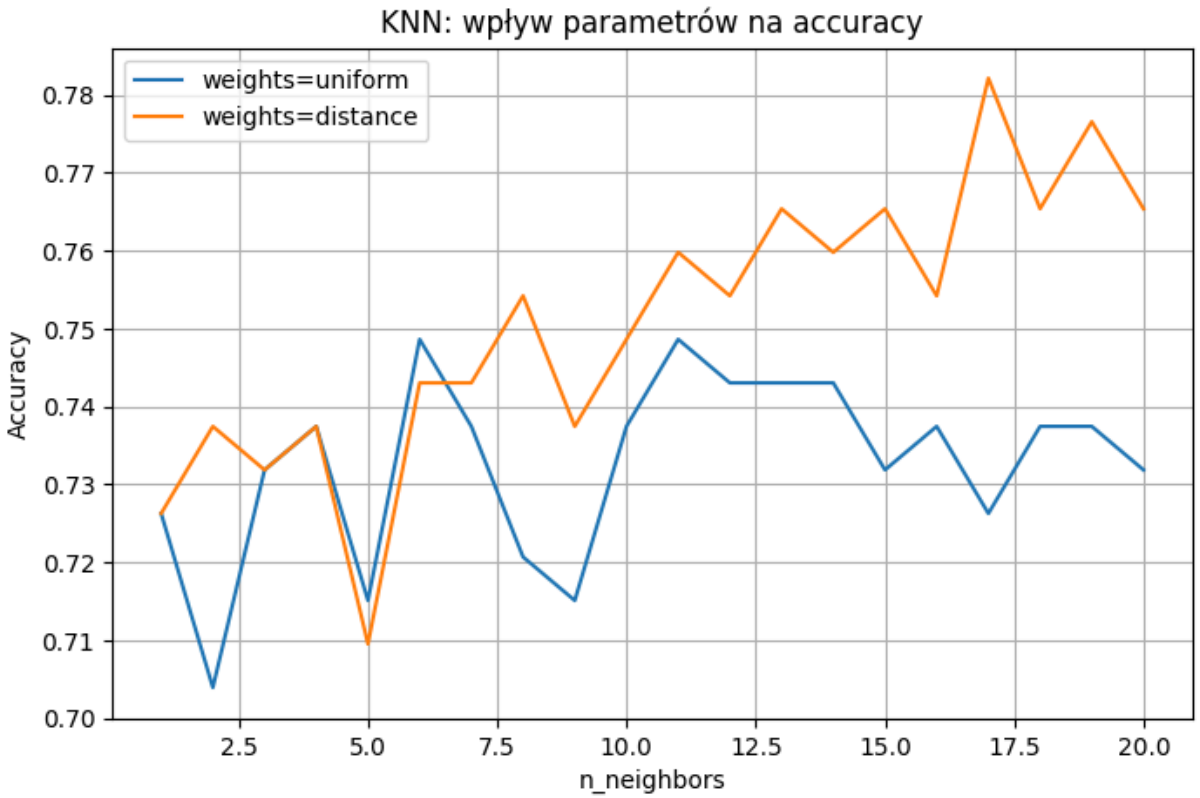


In [36]:
```
n_neighbors_values = range(1, 21)
weights_values = ['uniform', 'distance']

results_knn = []
for n in n_neighbors_values:
    for w in weights_values:
```

```
        model = KNeighborsClassifier(n_neighbors=n, weights=w)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results_knn.append((n, w, acc))
df_knn = pd.DataFrame(results_knn, columns=["n_neighbors", "weights", "accuracy"])
plt.figure(figsize=(8, 5))
for w in weights_values:
    subset = df_knn[df_knn["weights"] == w]
    plt.plot(subset["n_neighbors"], subset["accuracy"], label=f"weights={w}")

plt.xlabel("n_neighbors")
plt.ylabel("Accuracy")
plt.title("KNN: wpływ parametrów na accuracy")
plt.legend()
plt.grid(True)
plt.show()
```



Decision Tree Najwyższe accuracy (~82,2%) uzyskano przy max_depth=12 oraz min_samples_split=10. Widać, że zbyt małe głębokości drzewa lub zbyt duże wartości min_samples_split mogą ograniczać skuteczność modelu.

KNN Najlepszy wynik (~78,2%) pojawił się dla n_neighbors=17 oraz weights='distance'. Ogólnie widać, że metoda ważenia „distance" wypada lepiej od „uniform" dla większości wartości n_neighbors.

(n_neighbors określa, ilu najbliższych sąsiadów jest branych pod uwagę przy klasyfikacji (większe wartości oznaczają bardziej uśrednione decyzje))