Classificação de Variedades de Grãos de Trigo (Seeds Dataset) — CRISP-DM

Resumo: Notebook que segue a metodologia CRISP-DM para classificar três variedades de grãos de trigo (Kama, Rosa, Canadian) usando o *Seeds Dataset* (UCI). O notebook faz download automático dos dados da UCI, realiza EDA, pré-processamento, treina vários modelos, executa busca de hiperparâmetros e apresenta conclusões interpretáveis.

Como usar: execute todas as células (recomendado em Google Colab ou Jupyter local). O dataset será baixado automaticamente da UCI.

```
# Imports e configurações iniciais
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score, precision_score,
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

```
# 1) Carregar o dataset direto do UCI Repository
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00236/seeds_dataset.txt'

# O arquivo não contém cabeçalho; colunas conforme descrição UCI
colnames = ['area','perimeter','compactness','length_kernel','width_kernel','asymmetry_coeff','length_groove','c

df = pd.read_csv(url, sep='\s+', header=None, names=colnames)
print('Shape:', df.shape)
df.head()
```

```
Shape: (210, 8)
```

	area	perimeter	compactness	length_kernel	width_kernel	asymmetry_coeff	length_groove	class
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1

```
# Estatísticas descritivas
display(df.describe().T)

# Distribuição das classes
display(df['class'].value_counts().sort_index())
```

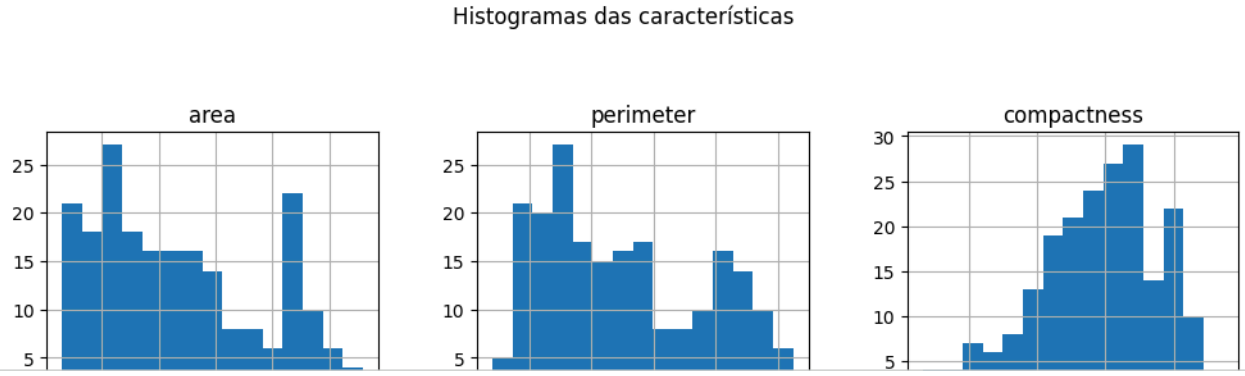
	count	mean	std	min	25%	50%	75%	max
area	210.0	14.847524	2.909699	10.5900	12.27000	14.35500	17.305000	21.1800
perimeter	210.0	14.559286	1.305959	12.4100	13.45000	14.32000	15.715000	17.2500
compactness	210.0	0.870999	0.023629	0.8081	0.85690	0.87345	0.887775	0.9183
length_kernel	210.0	5.628533	0.443063	4.8990	5.26225	5.52350	5.979750	6.6750
width_kernel	210.0	3.258605	0.377714	2.6300	2.94400	3.23700	3.561750	4.0330
asymmetry_coeff	210.0	3.700201	1.503557	0.7651	2.56150	3.59900	4.768750	8.4560
length_groove	210.0	5.408071	0.491480	4.5190	5.04500	5.22300	5.877000	6.5500
class	210.0	2.000000	0.818448	1.0000	1.00000	2.00000	3.000000	3.0000

class	count
1	70
2	70
3	70

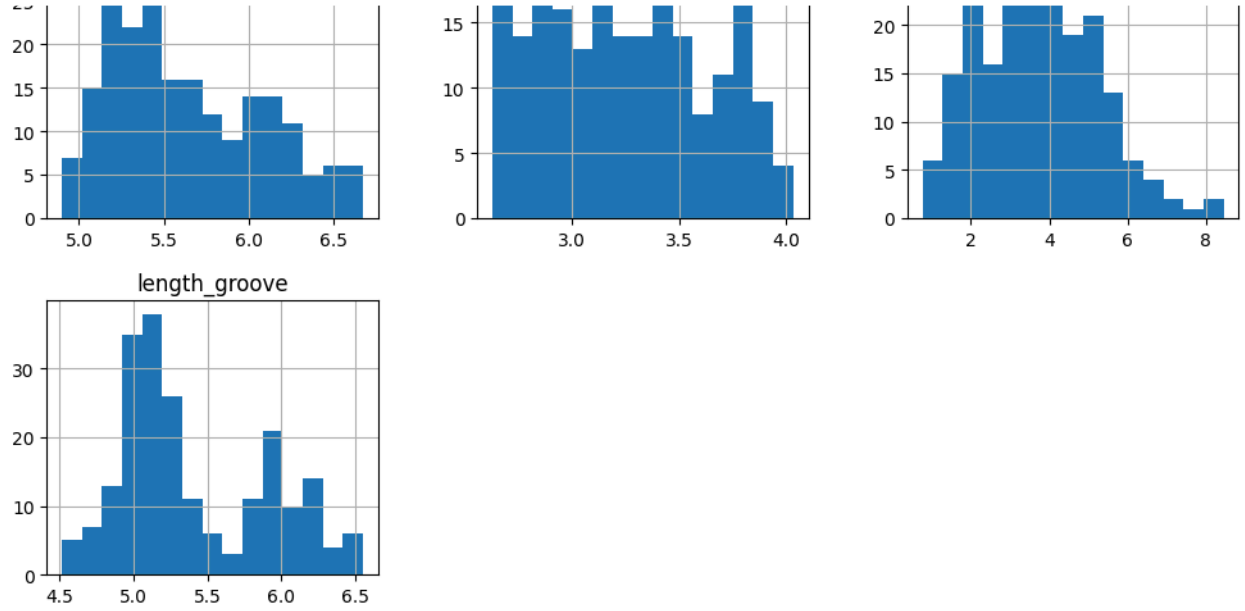
dtype: int64

```
# Visualizações: histogramas e boxplots
features = df.columns[:-1]

plt.figure(figsize=(12,8))
df[features].hist(bins=15, layout=(3,3), figsize=(12,10))
plt.suptitle('Histogramas das características')
plt.show()

plt.figure(figsize=(12,8))
df[features].plot(kind='box', subplots=True, layout=(3,3), figsize=(12,10))
plt.suptitle('Boxplots das características')
plt.show()
```

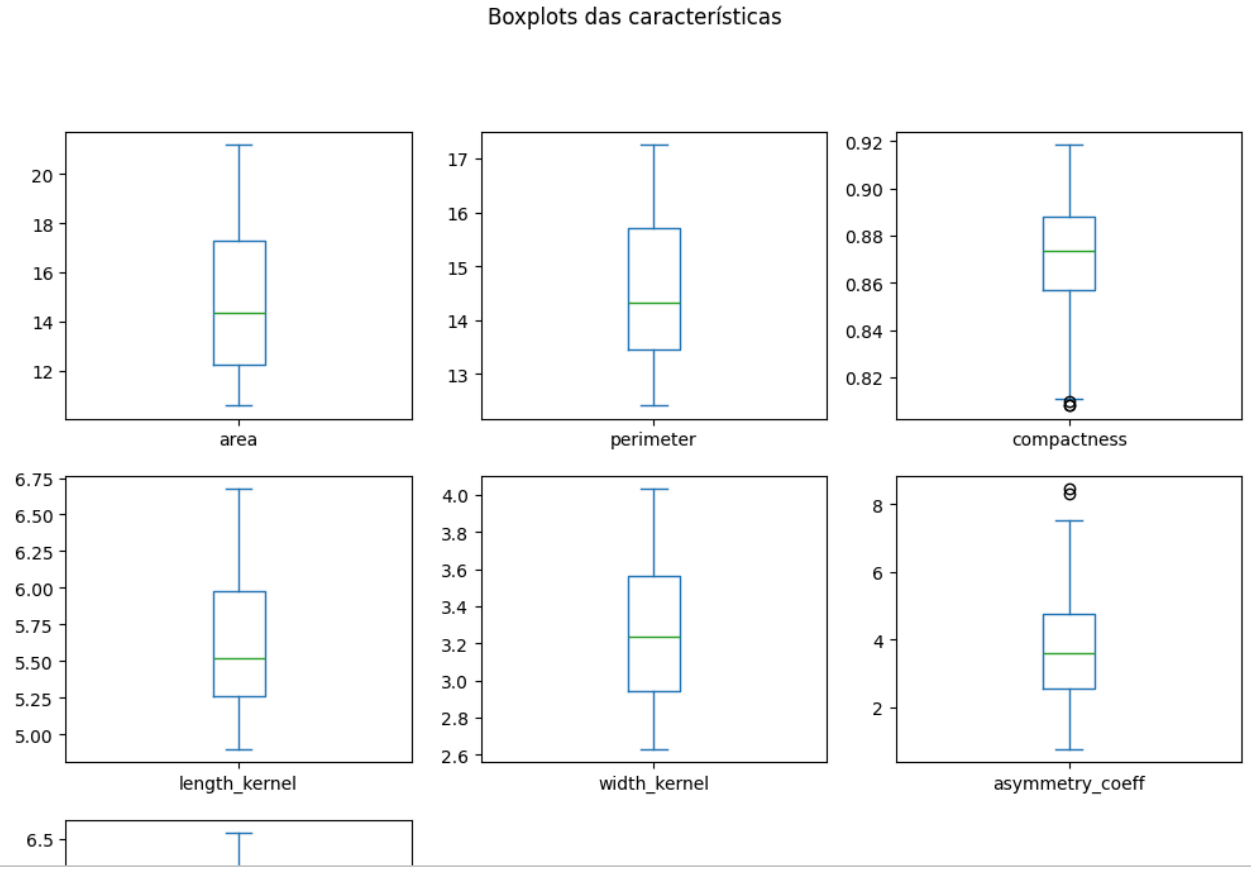
```
<Figure size 1200x800 with 0 Axes>
```

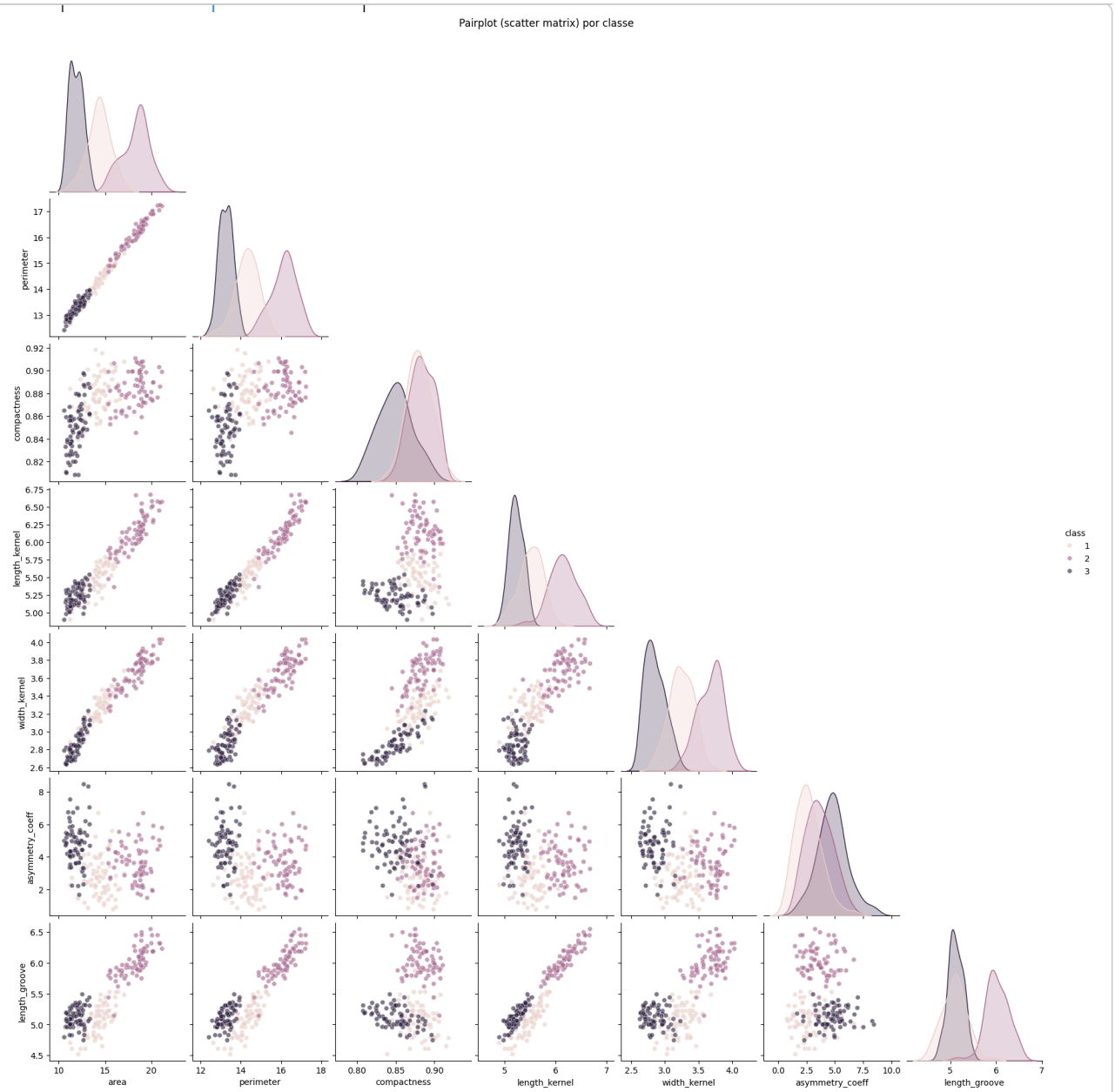



```
# Scatter matrix (pairplot) para ver relações entre atributos
sns.pairplot(df, vars=features, hue='class', corner=True, plot_kws={'alpha':0.6,'s':30})
plt.suptitle('Pairplot (scatter matrix) por classe', y=1.02)
plt.show()
```



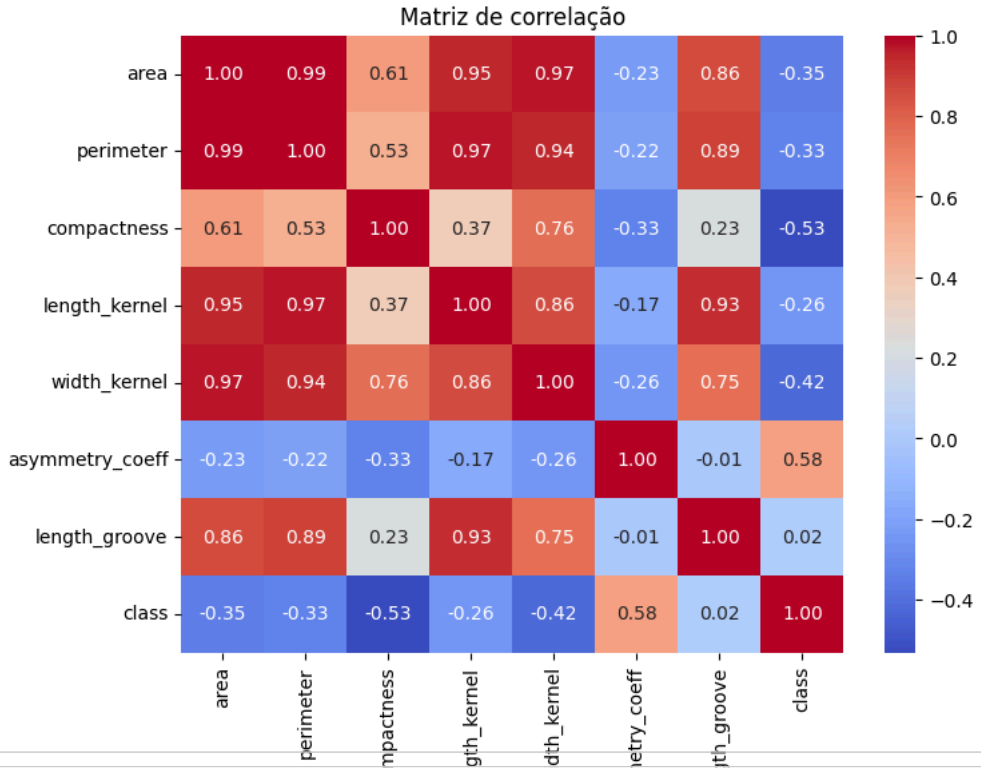

```
<Figure size 1200x800 with 0 Axes>
```






```
# Verificar valores ausentes
print('Missing values por coluna:\n', df.isnull().sum())

# Correlações
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(), annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Matriz de correlação')
plt.show()
```

```
Missing values por coluna:
 area               0
perimeter          0
compactness        0
length_kernel      0
width_kernel       0
asymmetry_coeff    0
length_groove      0
class              0
dtype: int64
```




```
# 2) Pré-processamento
X = df.drop('class', axis=1).values
y = df['class'].values

# Treino/teste (70/30)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42, stratify=y)

# Escalonamento
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print('Dataset split feito. X_train:', X_train.shape, 'X_test:', X_test.shape)
```

```
Dataset split feito. X_train: (147, 7) X_test: (63, 7)
```

```
# Função utilitária para treinar, prever e apresentar métricas
def evaluate_model(model, Xtr, Xte, ytr, yte, verbose=True):
    model.fit(Xtr, ytr)
    ypred = model.predict(Xte)
    acc = accuracy_score(yte, ypred)
    report = classification_report(yte, ypred, zero_division=0)
    cm = confusion_matrix(yte, ypred)
    if verbose:
        print(model.__class__.__name__)
        print('Accuracy: {:.4f}'.format(acc))
        print('Classification report:\n', report)
        print('Confusion matrix:\n', cm)
        print('-'*40)
    return {'model': model, 'accuracy': acc, 'report': report, 'confusion_matrix': cm, 'y_pred': ypred}
```

```
# Modelos iniciais (baseline)
models = [
    KNeighborsClassifier(),
    SVC(probability=True),
    RandomForestClassifier(random_state=42),
```

```
    GaussianNB(),
    LogisticRegression(max_iter=1000)
]

results = {}
for m in models:
    res = evaluate_model(m, X_train_scaled, X_test_scaled, y_train, y_test)
    results[m.__class__.__name__] = res
```

```
 [[16  2  3]
 [ 2 19  0]
 [ 1  0 20]]
----------------------------------------
RandomForestClassifier
Accuracy: 0.9206
Classification report:
              precision    recall  f1-score   support

           1       0.94      0.81      0.87        21
           2       0.95      0.95      0.95        21
           3       0.88      1.00      0.93        21

    accuracy                           0.92        63
   macro avg       0.92      0.92      0.92        63
weighted avg       0.92      0.92      0.92        63

Confusion matrix:
 [[17  1  3]
 [ 1 20  0]
 [ 0  0 21]]
----------------------------------------
GaussianNB
Accuracy: 0.8254
Classification report:
              precision    recall  f1-score   support

           1       0.73      0.76      0.74        21
           2       0.94      0.76      0.84        21
           3       0.83      0.95      0.89        21

    accuracy                           0.83        63
   macro avg       0.83      0.83      0.83        63
weighted avg       0.83      0.83      0.83        63

Confusion matrix:
 [[16  1  4]
 [ 5 16  0]
 [ 1  0 20]]
----------------------------------------
LogisticRegression
Accuracy: 0.8571
Classification report:
              precision    recall  f1-score   support

           1       0.83      0.71      0.77        21
           2       0.90      0.90      0.90        21
           3       0.83      0.95      0.89        21

    accuracy                           0.86        63
   macro avg       0.86      0.86      0.85        63
weighted avg       0.86      0.86      0.85        63

Confusion matrix:
 [[15  2  4]
 [ 2 19  0]
 [ 1  0 20]]
----------------------------------------
```

```
# Comparar acurácias
for name, r in results.items():
    print(f'{name:20s} -> Accuracy: {r["accuracy"]:.4f}')
```

```
KNeighborsClassifier -> Accuracy: 0.8730
SVC                  -> Accuracy: 0.8730
RandomForestClassifier -> Accuracy: 0.9206
GaussianNB           -> Accuracy: 0.8254
LogisticRegression   -> Accuracy: 0.8571
```

```
# 3) Otimização de hiperparâmetros (exemplos para KNN, SVM e RandomForest)
param_grids = {
    'KNN': {'n_neighbors': [1,3,5,7,9], 'weights': ['uniform','distance']},
    'SVC': {'C':[0.1,1,10,100], 'gamma': ['scale','auto'], 'kernel':['rbf','poly']},
    'RF': {'n_estimators':[50,100,200], 'max_depth':[None,5,10], 'min_samples_split':[2,5]}
}
```

```
}

# KNN GridSearch
knn = KNeighborsClassifier()
knn_gscv = GridSearchCV(knn, param_grids['KNN'], cv=5, scoring='accuracy', n_jobs=-1)
knn_gscv.fit(X_train_scaled, y_train)
print('Best KNN params:', knn_gscv.best_params_, 'Best CV score:', knn_gscv.best_score_)

# SVC GridSearch (may take longer)
svc = SVC(probability=True)
svc_gscv = GridSearchCV(svc, param_grids['SVC'], cv=5, scoring='accuracy', n_jobs=-1)
svc_gscv.fit(X_train_scaled, y_train)
print('Best SVC params:', svc_gscv.best_params_, 'Best CV score:', svc_gscv.best_score_)

# RandomForest GridSearch
rf = RandomForestClassifier(random_state=42)
rf_gscv = GridSearchCV(rf, param_grids['RF'], cv=5, scoring='accuracy', n_jobs=-1)
rf_gscv.fit(X_train_scaled, y_train)
print('Best RF params:', rf_gscv.best_params_, 'Best CV score:', rf_gscv.best_score_)
```

```
Best KNN params: {'n_neighbors': 1, 'weights': 'uniform'} Best CV score: 0.9393103448275861
Best SVC params: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'} Best CV score: 0.9593103448275861
Best RF params: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 50} Best CV score: 0.9055172413793103
```

```
# Avaliar modelos otimizados no conjunto de teste
best_models = {
    'KNN': knn_gscv.best_estimator_,
    'SVC': svc_gscv.best_estimator_,
    'RF': rf_gscv.best_estimator_
}

tuned_results = {}
for name, bm in best_models.items():
    tuned_results[name] = evaluate_model(bm, X_train_scaled, X_test_scaled, y_train, y_test)
```

```
KNeighborsClassifier
Accuracy: 0.9048
Classification report:
              precision    recall  f1-score   support

           1       0.89      0.81      0.85        21
           2       0.95      0.90      0.93        21
           3       0.88      1.00      0.93        21

    accuracy                           0.90        63
   macro avg       0.91      0.90      0.90        63
weighted avg       0.91      0.90      0.90        63

Confusion matrix:
 [[17  1  3]
 [ 2 19  0]
 [ 0  0 21]]
----------------------------------------
SVC
Accuracy: 0.8571
Classification report:
              precision    recall  f1-score   support

           1       0.83      0.71      0.77        21
           2       0.86      0.90      0.88        21
           3       0.87      0.95      0.91        21

    accuracy                           0.86        63
   macro avg       0.86      0.86      0.85        63
weighted avg       0.86      0.86      0.85        63

Confusion matrix:
 [[15  3  3]
 [ 2 19  0]
 [ 1  0 20]]
----------------------------------------
RandomForestClassifier
Accuracy: 0.8889
Classification report:
              precision    recall  f1-score   support
```