

✓ 1. Carregamento e Visualização Inicial

Carregar os dados e explorar sua estrutura:

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

# Definir nomes das colunas
columns = ['Área', 'Perímetro', 'Compacidade', 'Comprimento_Núcleo', 'Largura_Núcleo',
           'Coeficiente_Assimetria', 'Comprimento_Sulco', 'Variedade']

# Ler o arquivo tratando múltiplos espaços como delimitadores
df = pd.read_csv("seeds_dataset.txt", sep="\s+", names=columns)

# Exibir as primeiras linhas
print(df.head())

# Estatísticas descritivas
print(df.describe())

# Verificar valores ausentes
print(df.isnull().sum())

# Visualizar distribuições dos atributos
df.hist(figsize=(12, 8))
plt.show()
```



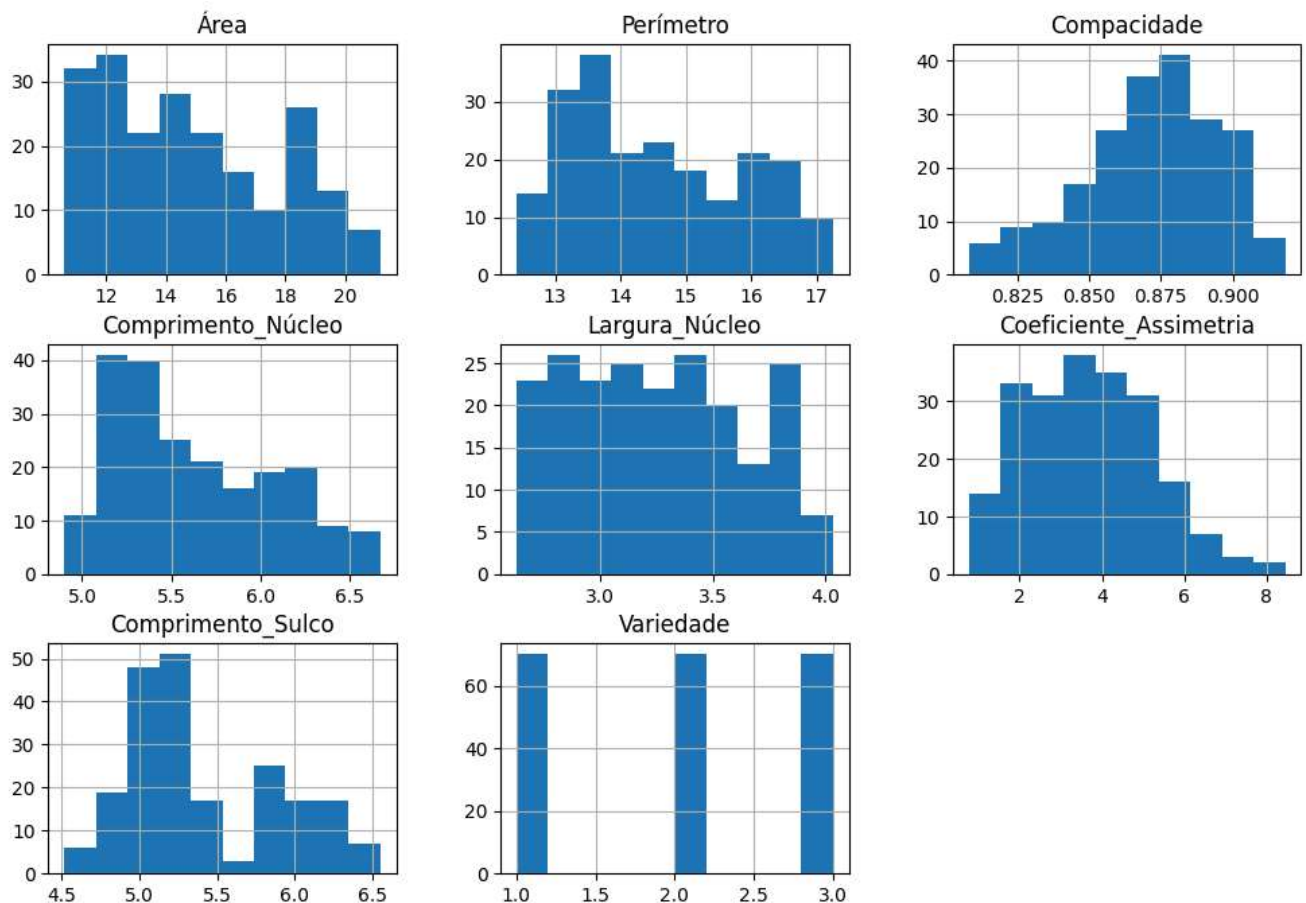
	Área	Perímetro	Compacidade	Comprimento_Núcleo	Largura_Núcleo	\
0	15.26	14.84	0.8710	5.763	3.312	
1	14.88	14.57	0.8811	5.554	3.333	
2	14.29	14.09	0.9050	5.291	3.337	
3	13.84	13.94	0.8955	5.324	3.379	
4	16.14	14.99	0.9034	5.658	3.562	

	Coeficiente_Assimetria	Comprimento_Sulco	Variedade
0	2.221	5.220	1
1	1.018	4.956	1
2	2.699	4.825	1
3	2.259	4.805	1
4	1.355	5.175	1

	Área	Perímetro	Compacidade	Comprimento_Núcleo	\
count	210.000000	210.000000	210.000000	210.000000	
mean	14.847524	14.559286	0.870999	5.628533	
std	2.909699	1.305959	0.023629	0.443063	
min	10.590000	12.410000	0.808100	4.899000	
25%	12.270000	13.450000	0.856900	5.262250	
50%	14.355000	14.320000	0.873450	5.523500	
75%	17.305000	15.715000	0.887775	5.979750	
max	21.180000	17.250000	0.918300	6.675000	

	Largura_Núcleo	Coeficiente_Assimetria	Comprimento_Sulco	Variedade
count	210.000000	210.000000	210.000000	210.000000
mean	3.258605	3.700201	5.408071	2.000000
std	0.377714	1.503557	0.491480	0.818448
min	2.630000	0.765100	4.519000	1.000000
25%	2.944000	2.561500	5.045000	1.000000
50%	3.237000	3.599000	5.223000	2.000000
75%	3.561750	4.768750	5.877000	3.000000
max	4.033000	8.456000	6.550000	3.000000

Área 0
 Perímetro 0
 Compacidade 0
 Comprimento_Núcleo 0
 Largura_Núcleo 0
 Coeficiente_Assimetria 0
 Comprimento_Sulco 0
 Variedade 0
 dtype: int64



✓ 2. Normalização e Padronização dos Dados

Utilizar MinMaxScaler ou StandardScaler, para que as escalas dos atributos não influenciem o modelo.

Aplicar StandardScaler para padronizar os dados, pois os dados possuem magnitudes muito diferentes e deve padronizá-los antes da modelagem.

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Min-Max Scaling
scaler_minmax = MinMaxScaler()
df_minmax = df.copy()
df_minmax.iloc[:, :-1] = scaler_minmax.fit_transform(df.iloc[:, :-1])

# StandardScaler (Z-score)
scaler_standard = StandardScaler()
df_standard = df.copy()
df_standard.iloc[:, :-1] = scaler_standard.fit_transform(df.iloc[:, :-1])

# Aplicar padronização nos dados
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Treinar o modelo logistic regression com dados escalados
logistic_model = LogisticRegression()
logistic_model.fit(X_train_scaled, y_train)
```



✓ 3. Separação em Conjuntos de Treinamento e Teste

Dividir os dados para treinar e testar os modelos:

```
from sklearn.model_selection import train_test_split

# Definir X (variáveis preditoras) e y (variável alvo)
X = df.iloc[:, :-1]
y = df['Variedade']

# Divisão dos dados
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

✓ 4. Implementação e Comparação dos Algoritmos de Classificação

Treinar diferentes modelos de classificação e avaliar seu desempenho:

```
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

# Modelos de classificação
models = {
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "SVM": SVC(kernel='linear'),
    "Logistic Regression": LogisticRegression(max_iter=500),
    "Naive Bayes": GaussianNB(),
}

# Treinar e avaliar cada modelo
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"\nModelo: {name}")
    print(classification_report(y_test, y_pred))
    print("Matriz de Confusão:")
    print(confusion_matrix(y_test, y_pred))

# Comparação de acurácia
print("\nComparação de Acurácia:")
for model, acc in results.items():
    print(f"{model}: {acc:.4f}")
```

```
[[16  3  2]
 [ 2 19  0]
 [ 1  0 20]]
```

Modelo: SVM

[1 0 20]]

Modelo: Naive Bayes

	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

Matriz de Confusão:

```
[[16  1  4]
 [ 5 16  0]
 [ 1  0 20]]
```

Comparação de Acurácia:

Random Forest: 0.9206

KNN: 0.8730

SVM: 0.8571

Logistic Regression: 0.8571

Naive Baves: 0.8254

✓ 5. Otimização dos Modelos

Utilizar Grid Search para encontrar melhores hiperparâmetros:

```
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler

# Otimização do SVM
param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
grid = GridSearchCV(SVC(), param_grid, cv=5)
grid.fit(X_train, y_train)

# Melhor combinação de parâmetros SVM
print("Melhores parâmetros para SVM:", grid.best_params_)

# Otimização do K-Nearest Neighbors (KNN)
# Testar diferentes valores para n_neighbors e o tipo de métrica utilizada:
param_grid_knn = {'n_neighbors': [3, 5, 7, 9], 'weights': ['uniform', 'distance']}
grid_knn = GridSearchCV(KNeighborsClassifier(), param_grid_knn, cv=5)
grid_knn.fit(X_train, y_train)

# Melhor combinação de parâmetros KNN
print("Melhores parâmetros para KNN:", grid_knn.best_params_)

# Otimização do Random Forest
# Explorar o número de árvores na floresta (n_estimators),
# profundidade (max_depth) e critérios de divisão (criterion):
param_grid_rf = {'n_estimators': [50, 100, 150], 'max_depth': [None, 10, 20], 'criterion': ['gini', 'entropy']}
grid_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf, cv=5)
grid_rf.fit(X_train, y_train)

# Melhor combinação de parâmetros Random Forest
print("Melhores parâmetros para Random Forest:", grid_rf.best_params_)

# Otimização do Naive Bayes
# O algoritmo GaussianNB não tem muitos hiperparâmetros para ajustar, mas
# podemos otimizar var_smoothing para melhor regularização:
param_grid_nb = {'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6]}
grid_nb = GridSearchCV(GaussianNB(), param_grid_nb, cv=5)
grid_nb.fit(X_train, y_train)

# Melhor combinação de parâmetros Naive Bayes
print("Melhores parâmetros para Naive Bayes:", grid_nb.best_params_)
```

```
# Otimização da Regressão Logística

# Padronização dos dados
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Ajustar o solver (solver) e o parâmetro de regularização (C):
param_grid_lr = {'C': [0.01, 0.1, 1, 10], 'solver': ['newton-cg']}
grid_lr = GridSearchCV(LogisticRegression(max_iter=2000), param_grid_lr, cv=5)
grid_lr.fit(X_train_scaled, y_train_scaled)
```