Primeiramente, demos import nas bibliotecas necessárias para a realização dos testes

In [ ]:
```
#import das bibliotecas necessárias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Segundamente, carreguei os dados do dataset que escolhemos, no caso, o da Iris, e em seguida, carreguei as colunas para uma visualização primária.

In [ ]:
```
# Carregar os dados do arquivo CSV
iris = pd.read_csv('iris_extended.csv')
# Visualizar as colunas do dataset
print(iris.columns)
```

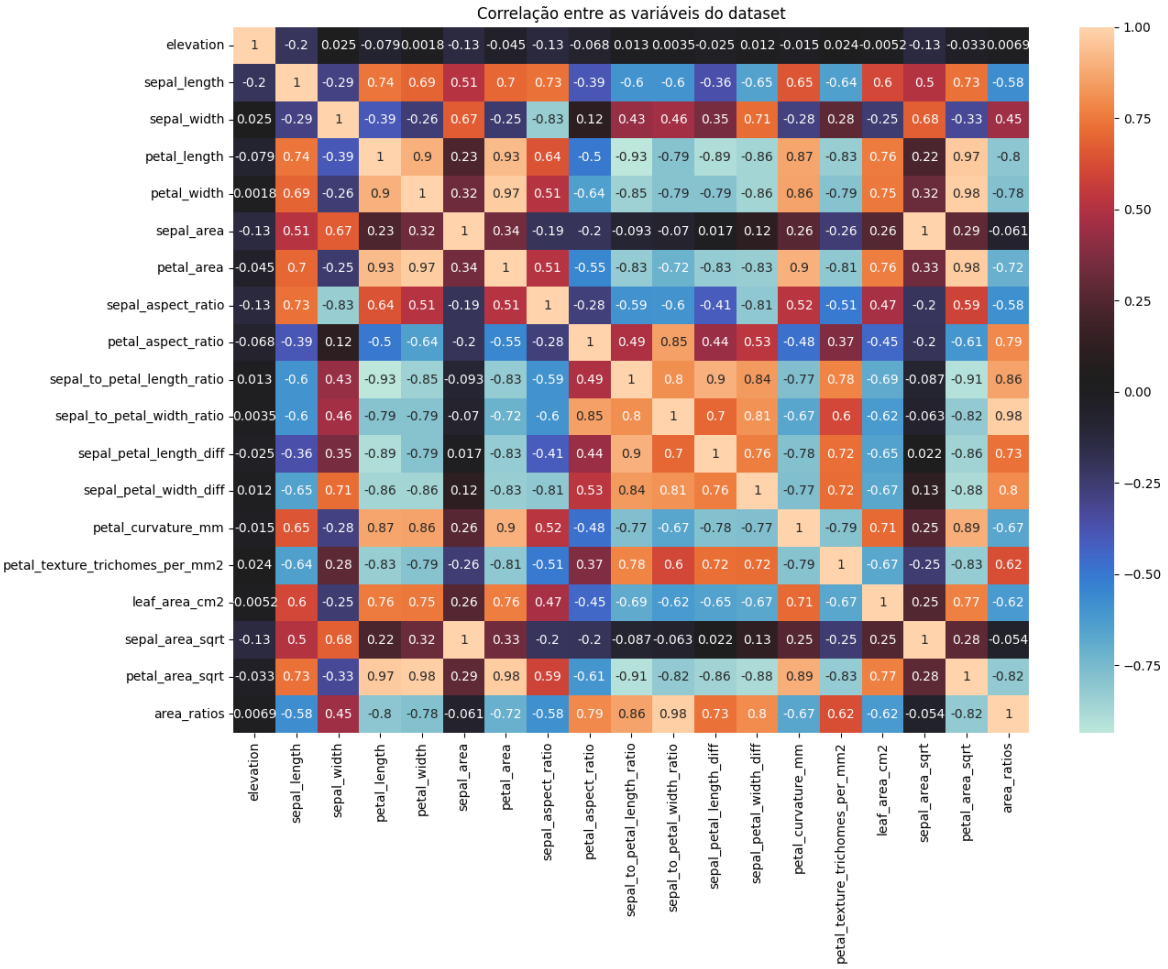
```
Index(['species', 'elevation', 'soil_type', 'sepal_length', 'sepal_width',
       'petal_length', 'petal_width', 'sepal_area', 'petal_area',
       'sepal_aspect_ratio', 'petal_aspect_ratio',
       'sepal_to_petal_length_ratio', 'sepal_to_petal_width_ratio',
       'sepal_petal_length_diff', 'sepal_petal_width_diff',
       'petal_curvature_mm', 'petal_texture_trichomes_per_mm2',
       'leaf_area_cm2', 'sepal_area_sqrt', 'petal_area_sqrt', 'area_ratios'],
      dtype='object')
```

Logo em seguida, há um tratamento de dados, no qual se exclui as colunas não númericas afim de plotar o gráfico de calor do dataset. E no fim, há o gráfico para a visualização das correlações entre as variáveis,no qual as cores mais claras, indicam uma forte correlação positiva entre as variáveis, o que significa, que quando uma variável aumenta, a outra também tende a aumentar. Diferentemente das cores mais escuras, indicam uma forte correlação negativa entre as variáveis, significando que, quando uma variável aumenta, a outra tende a diminuir.

In [ ]:
```
#exclue a coluna target(não númericas) do dataset
numeric_iris = iris.select_dtypes(exclude=['object'])

# Visualizar a correlação apenas para as variáveis numéricas
plt.figure()
plt.figure(figsize=(14, 10))
sns.heatmap(numeric_iris.corr(), annot=True, cmap='icefire').set_title('Correlaç
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



Nessa parte do código, iniciamos a divisão dos dados para iniciar o treinamento da IA, as variáveis X, são as colunas no qual quero fazer as comparações, a fim de obter o "alvo", que é a coluna Y. As características, ou features, são as variáveis que o modelo usará para fazer previsões. Neste caso, estamos escolhendo várias medidas relacionadas às pétalas e sépalas das flores iris. O alvo é a variável que o modelo tentará prever. Neste caso, 'species'-há 3 espécies- representa a classe da flor iris.

In [ ]:
```
# Dividir os dados em recursos (X_val) e alvo (Y_val)
# Selecionar as colunas desejadas para os recursos (X_val)
X_val = iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
              'sepal_area', 'petal_area', 'sepal_aspect_ratio', 'petal_aspect_ra
              'sepal_to_petal_length_ratio', 'sepal_to_petal_width_ratio',
              'sepal_petal_length_diff', 'sepal_petal_width_diff', 'petal_curvat
              'petal_texture_trichomes_per_mm2', 'leaf_area_cm2', 'sepal_area_sq
              'petal_area_sqrt', 'area_ratios']].values
# Selecionar a coluna 'species' como o alvo (Y_val)
Y_val = iris['species'].values
```

Nesse momento, há a divisão dos dados de teste e dados de treinamento, no qual 40% dos dados são destinados aos testes, e 60% dos dados para o treinamento. Também, definimos arrays my_neighbords, contendo valores de 1 a 9, para avaliar os dados as amostras dedos e sua próximidade a amostra de consulta.

In [ ]:
```
# Dividir os dados em conjuntos de treinamento e teste
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_val, Y_val, test_size=0.4,
                                                    random_state=42)
```

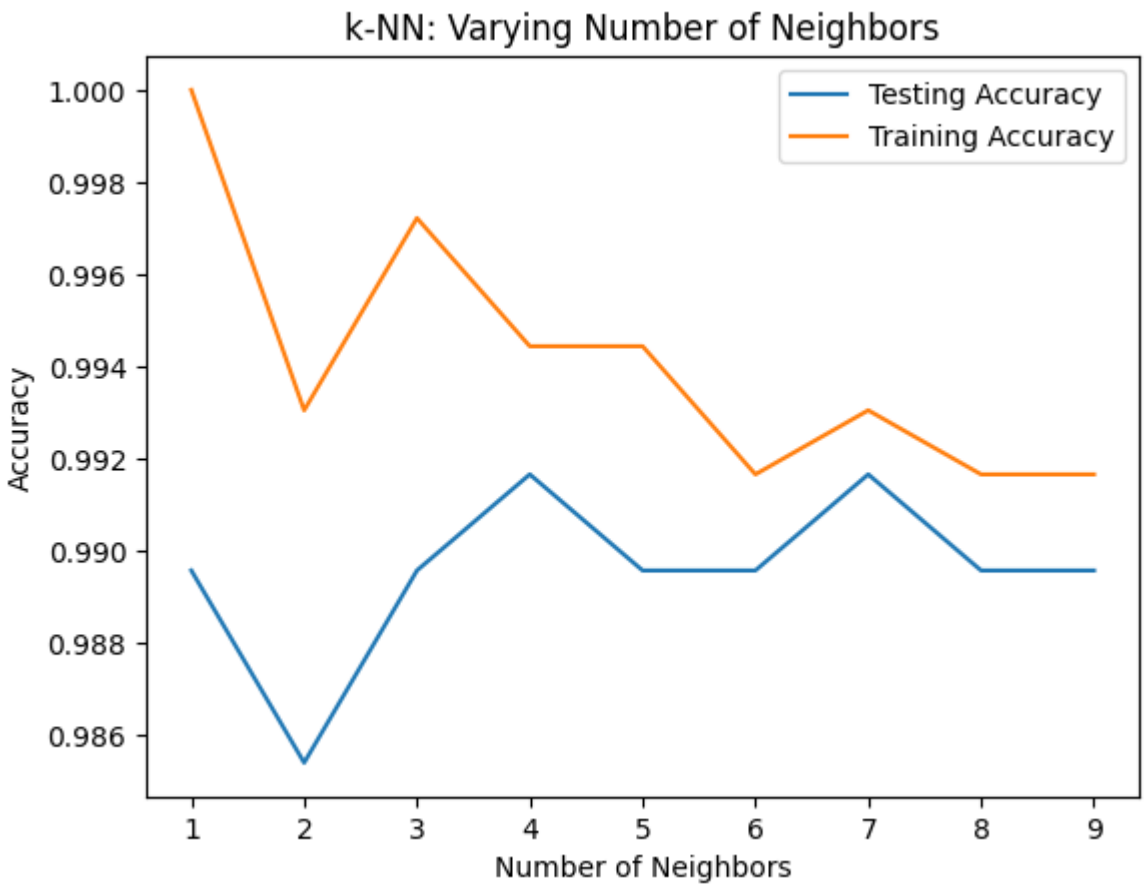
```
my_neighbors = np.arange(1, 10)
train_accuracy = np.empty(len(my_neighbors))
test_accuracy = np.empty(len(my_neighbors))
```

Aqui, usa-se os variados valores dos vizinhos para conseguir posteriormente enxergar quando as amostras são mais próximas, testando assim a sua precisão.

In [ ]:
```
#treinar o modelo com diferentes valores de k e testar a precisão
for i, k in enumerate(my_neighbors):
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(X_train, Y_train)
    train_accuracy[i] = knn_model.score(X_train, Y_train)
    test_accuracy[i] = knn_model.score(X_test, Y_test)
```

E então, o algoritmo abaixo, é destinado a plotar os valores para cada vizinho, tornando possível a avaliação gráfica da precisão.

In [ ]:
```
# Plotar os resultados
plt.title('k-NN: Varying Number of Neighbors')
plt.plot(my_neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(my_neighbors, train_accuracy, label='Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
```



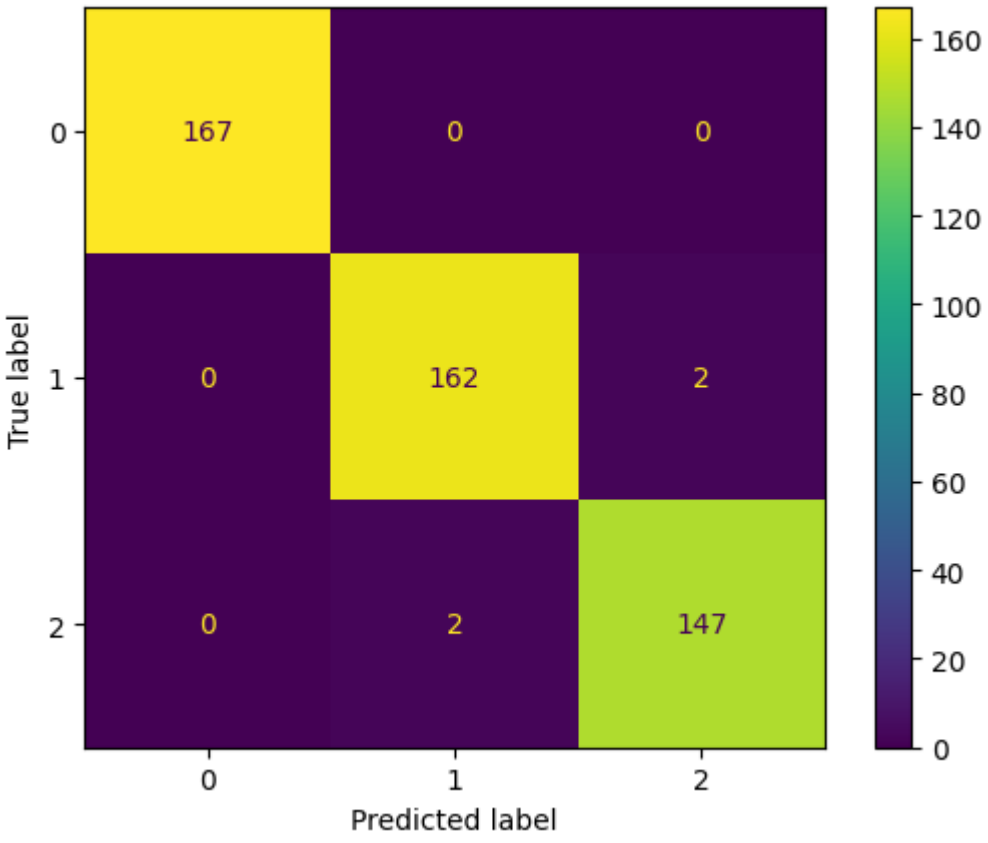
Logo em seguida, escolhemos o número do vizinho (7), no qual a acurácia dos testes se aproxima mais do ponto ideal(maior pico), e menor distância referente ao gráfico da acurácia do treinamento. E o output se refere ao score da acurácia do modelo em questão. O valor 0.99 refere a porcentagem de dados classficados corretamente.

In [ ]:
```
# Treinar o modelo com k=7
knn_model = KNeighborsClassifier(n_neighbors=7)
knn_model.fit(X_train, Y_train)
knn_model.score(X_test, Y_test)
```

Out[ ]:
```
0.9916666666666667
```

Abaixo, há a construção da matriz de confusão, na qual serve para verificar o desempenho do modelo em termos de acerto e erros. Pode se notar, que em várias instâncias o erro (Falso positivo e Falso negativo), é pequena, tendo sua maxima frequência de aparição em 2, já a maioria dos dados, estão no conjunto(Verdadeiro positivo e Verdadeiro negativo), evidenciado pelos números centrais de cor amarelada.

In [ ]:
```
# Matriz de confusão
Y_pred = knn_model.predict(X_test)
cm = confusion_matrix(Y_test, Y_pred)
ConfusionMatrixDisplay(cm).plot()
plt.show()
```



Por fim, concluimos o código com um algoritmo no qual conseguimos classificar a partir de um relatório a eficiência do modelo, elencando a precisão, revocação,que indica a proporção de instâncias positivas corretamente classficadas, F1-score, que é a média harmônica entre as duas medidas anteriores,support, número real de ocorrências de cada classe no conjunto de teste, acurácia, que indica a proporção de todas as previsões corretas, para cada variavél.

In [ ]:
```
# Relatório de classificação
from sklearn.metrics import classification_report
```

```
print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00       167
  versicolor       0.99      0.99      0.99       164
   virginica       0.99      0.99      0.99       149

    accuracy                           0.99       480
   macro avg       0.99      0.99      0.99       480
weighted avg       0.99      0.99      0.99       480
```