Objetivos

- Apresentar e utilizar o classificador k-nearest neighbours (kNN)
- Apresentar a técnica de separação de dados (treino e teste)
- Avaliar Aprendizagem do modelo

Começando

Vamos dar continuidade ao nosso estudo de aprendizagem de máquina, já vimos:

- Tudo começa, conhecendo os dados disponíveis.
- Como carregar um data frame
- Como visualizar os dados em gráficos (histograma, box plot, violin plot, matriz de confusão)
- Fizemos uma breve introdução sobre análise exploratória buscando correlacionar os dados para gerar informações.

  Hoje, vamos seguir nossa jornada e finalizar nosso estudo aplicando a técnica de KNN.

k-Nearest Neighbors

O KNN(K vizinhos mais próximos) é considerado um dos algoritmos mais simples dentro da categoria de ***aprendizagem supervisionada*** sendo muito utilizado para problemas de classificação, porém também pode ser utilizado em problemas de regressão.

Problemas de classificação = Vale lembrar que em problemas de classificação não estamos interessados em valores exatos, queremos apenas saber se um dado pertence ou não a uma dada classe.

Uma intuição sobre o método

Para realizar a classificação o KNN calcula a distância objeto desconhecido (target) para todos os outros elementos, encontra os mais K vizinhos mais próximos faz uma contagem dos rótulos e considera que o objeto desconhecido pertence ao rótulo de maior contagem.

A imagem abaixo exemplifica o funcionamento, mas se ficou um pouco complicado de entender, rode o script python ***iknn.py*** e faça algumas simulações para compreender.

Bora lá!!

Vamos juntos realizar nosso primeiro projeto, do começo ao fim, de aprendizagem de máquina.

Definição do problema

A primeira coisa que precisamos fazer é a definição do problema. Neste primeiro caso vamos trabalhar com o mesmo dataset da última aula, dataset iris. Vamos desenvolver um sistema de machine learning capaz de classificar sua espécie com base nos dimensionais da pétala.

São 150 exemplares de flor de íris, pertencentes a três espécies diferentes: **setosa**, **versicolor** e **virginica**, sendo 50 amostras de cada espécie. Os atributos de largura e comprimento de sépala e largura e comprimento de pétala de cada flor fooram medidos manualmente.

Desafio 1

Do ponto de vista de machine learning, que problema é esse:

```
Aprendizado supervisionado ou não-supervisionado?
```

R: Supervisionado, depende de acompanhamento humano

```
Classificação ou regressão?
```

R: Classificação

```
### R1: Supervisionado

## R2: De classificação
```

```
# Inicializção das bibliotecas
%matplotlib inline

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

```
# Caminho do arquivo
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
# Define o nome das colunas
header = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
# Lê e carrega o arquivo para a memória
df = pd.read_csv(url, header=None, names=header)
```

```
# Retorna um trecho com as 5 primeiras linhas do dataframe
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
df.tail()
```

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

```
# Mostra informações sobre o dataframe em si
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

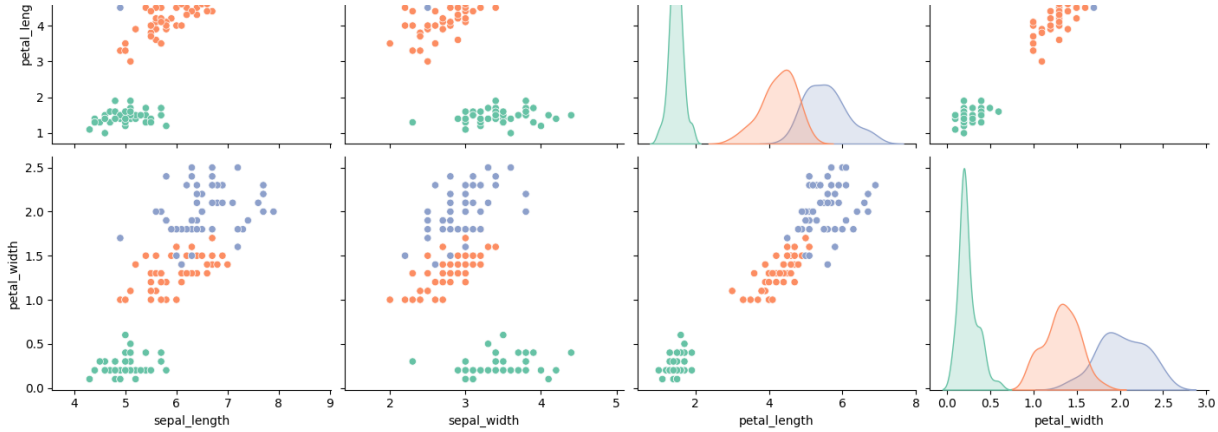
```
# class distribution
print(df.groupby('species').size())
```

```
species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

Desafio 2

Aplique os métodos que achar conveniente (vimos algumas opções na última aula) para visualizar os dados de forma gráfica.

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

# Caminho do arquivo
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Define o nome das colunas

header = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']

# Lê e carrega o arquivo para a memória

df = pd.read_csv(url, header=None, names=header)

# Pairplot para visualização multivariada
sns.pairplot(df, hue='species', height=3, aspect=1.2, palette='Set2')
plt.suptitle('Pairplot para Conjunto de Dados Iris', y=1.02)
plt.show()

# Heatmap de correlação
correlacao = df.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlacao, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Heatmap de Correlação para Conjunto de Dados Iris')
plt.show()

# Boxplot para cada variável em relação às espécies
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, palette='Set2')
plt.title('Boxplot para Conjunto de Dados Iris')
plt.xticks(rotation=45)
plt.show()
```




```
<ipython-input-3-af2f59944f58>:22: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future vers
  correlacao = df.corr()
```

PARE!!!

A análise feita no desafio 2 é uma das etapas mais importantes. Caso você tenha pulado essa etapa, volte e faça suas análises.

Com essa etapa concluída, vamos criar um sub-dataset com os atributos que serão utilizados.

```
# Selecionando um sub-dataframe com os campos petal_length e petal_width,
# e outro com a variável de classes
entradas = df[['petal_length', 'petal_width']]
classes = df['species']
print(f"Formato das tabelas de dados {entradas.shape} e classes {classes.shape}")
```

```
Formato das tabelas de dados (150, 2) e classes (150,)
```


Dividindo os dados em conjunto de treinamento e de testes

Dividir nosso dataset em dois conjuntos de dados.

```
Treinamento - Representa 80% das amostras do conjunto de dados original,
Teste - com 20% das amostras
```

Vamos escolher aleatoriamente algumas amostras do conjunto original. Isto pode ser feito com Scikit-Learn usando a função ***train_test_split()***

scikit-learn: pip3 install scikit-learn

```
# Separamos 20% para o teste
from sklearn.model_selection import train_test_split

entradas_treino, entradas_teste, classes_treino, classes_teste = train_test_split(entradas, classes, test_size=0.2)

print(f"Formato das tabelas de dados de treino {entradas_treino.shape} e teste {entradas_teste.shape}")
```

```
Formato das tabelas de dados de treino (120, 2) e teste (30, 2)
```

```
#Primeiras linhas do dataframe
entradas_treino.head()
```

	petal_length	petal_width
31	1.5	0.4
120	5.7	2.3
93	3.3	1.0
5	1.7	0.4
102	5.9	2.1

```
classes_treino.head()
```

```
31         Iris-setosa
120     Iris-virginica
93     Iris-versicolor
5          Iris-setosa
102     Iris-virginica
Name: species, dtype: object
```

Chegou a hora de aplicar o modelo preditivo

Treinar um modelo no python é simples se usar o Scikit-Learn. Treinar um modelo no Scikit-Learn é simples: basta criar o classificador, e chamar o método fit().

Uma observação sobre a sintaxe dos classificadores do `scikit-learn`

- O método `fit(X,Y)` recebe uma matriz ou dataframe X onde cada linha é uma amostra de aprendizado, e um array Y contendo as saídas esperadas do classificador, seja na forma de texto ou de inteiros
- O método `predict(X)` recebe uma matriz ou dataframe X onde cada linha é uma amostra de teste, retornando um array de classes

```
# Importa a biblioteca
from sklearn.neighbors import KNeighborsClassifier

# Cria o classificar KNN
k = 9
modelo = KNeighborsClassifier(n_neighbors=k)

# Cria o modelo de machine learning
modelo.fit(entradas_treino, classes_treino)
```

```
KNeighborsClassifier(n_neighbors=9)
```

Pronto!! bora testar se esta funcionando....

```
# Para obter as previsões, basta chamar o método predict()
classes_encontradas = modelo.predict(entradas_teste)
print("Predição: {}".format(classes_encontradas))
```

```
Predição: ['Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-versicolor' 'Iris-virginica'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa'
 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica' 'Iris-setosa'
 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor' 'Iris-versicolor'
```

```
       'Iris-setosa' 'Iris-virginica' 'Iris-versicolor' 'Iris-virginica'
       'Iris-versicolor' 'Iris-virginica']
```

```
# Para determinar a quantidade de acertos (acuracia)

from sklearn.metrics import accuracy_score
acertos = accuracy_score(classes_teste, classes_encontradas)
print("Acerto médio de classificação: ", acertos)
```

```
     Acerto médio de classificação:  0.9666666666666667
```

Utilizando o modelo treinado com amostras fora do dataset

Vamos colocar alguns valores e ver a predição do classificador.

```
# Criamos um modelo utilizando duas entradas e uma saida, logo temos que passar duas entradas para o modelo faça a predição.

modelo.predict([[3.3, 3.2]])
```

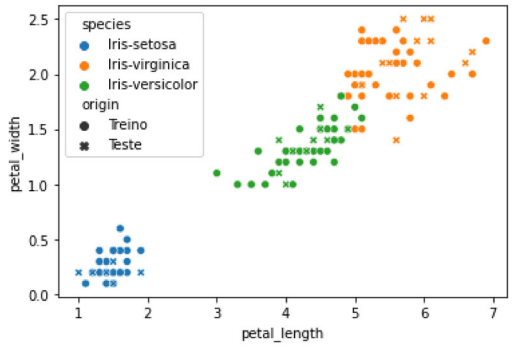
```
     array(['Iris-versicolor'], dtype=object)
```

Visualizando o modelo de forma gráfica

```
# Unificamos os dados de entrada e as classes de treino e teste em um daframe cada
df_treino = pd.concat((entradas_treino, classes_treino), axis=1)

novas_classes = pd.Series(classes_encontradas, name="species", index=entradas_teste.index)
df_teste = pd.concat((entradas_teste, novas_classes), axis=1)
```

```
import seaborn as sns
## Unificamos os dataframes de treinamento e teste em um novo DataFrame
# indicando a origem dos dados
novo_df = pd.concat((df_treino, df_teste), keys=['train', 'test'])
novo_df['origin'] = ''
novo_df.loc['train','origin'] = 'Treino'
novo_df.loc['test','origin'] = 'Teste'

# Usamos o scatterplot do seaborn, informando mudando o marcador de acordo com a origem do dado
sns.scatterplot('petal_length', 'petal_width', hue='species', style='origin', data=novo_df)

plt.show()
```

```
     c:\Users\junior\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following va
       warnings.warn(
```



Desafio 3

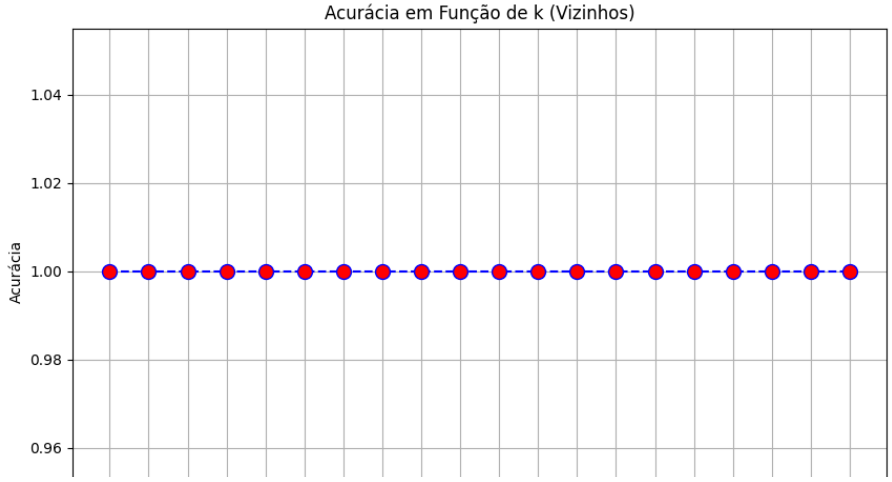
Fizemos o treinamento para k=3, mude o valor de k e análise a acurácia do modelo.

Dica: Faça um loop for que varre um range de k, a saída pode ser armazenada em uma lista. No final do loop exiba em um gráfico.

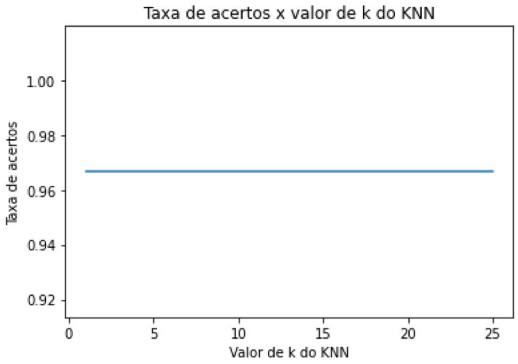
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Carregar os dados
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
header = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
df = pd.read_csv(url, header=None, names=header)

# Separar entradas e classes
entradas = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
classes = df['species']

# Dividir dados em conjuntos de treino e teste
entradas_treino, entradas_teste, classes_treino, classes_teste = train_test_split(entradas, classes, test_size=0.3, random_state=42)

# Criar o classificador KNN com k=3
k = 3
modelo = KNeighborsClassifier(n_neighbors=k)

# Treinar o modelo
modelo.fit(entradas_treino, classes_treino)

# Fazer uma predição com dados de teste
predicoes = modelo.predict(entradas_teste)

# Calcular a acurácia
acuracia = accuracy_score(classes_teste, predicoes)
print(f'Acurácia do modelo: {acuracia}')

# Scatterplot para visualização
novo_df = pd.concat((entradas_treino, classes_treino), keys=['train'])
novo_df['origin'] = 'Treino'

novas_classes = pd.Series(classes_teste, name="species", index=entradas_teste.index)
df_teste = pd.concat((entradas_teste, novas_classes), axis=1)
df_teste['origin'] = 'Teste'

novo_df = novo_df.append(df_teste.set_index('origin', append=True))

# Lista para armazenar as acurácias
acuracias = []

# Loop for para variar o valor de k de 1 a 20
for k in range(1, 21):
    # Criar o classificador KNN
    modelo = KNeighborsClassifier(n_neighbors=k)

    # Treinar o modelo
    modelo.fit(entradas_treino, classes_treino)

    # Fazer uma predição com dados de teste
    predicoes = modelo.predict(entradas_teste)

    # Calcular a acurácia e armazenar na lista
    acuracia = accuracy_score(classes_teste, predicoes)
    acuracias.append(acuracia)

# Scatterplot para visualizar acurácias em função de k
plt.figure(figsize=(10, 6))
plt.plot(range(1, 21), acuracias, marker='o', linestyle='dashed', color='b', markerfacecolor='r', markersize=10)
plt.title('Acurácia em Função de k (Vizinhos)')
plt.xlabel('Número de Vizinhos (k)')
plt.ylabel('Acurácia')
plt.xticks(range(1, 21))
plt.grid(True)
plt.show()
```

```
Acurácia do modelo: 1.0
<ipython-input-7-26797ef1b763>:42: FutureWarning: The frame.append method is deprecated
  novo_df = novo_df.append(df_teste.set_index('origin', append=True))
```




```
#### Resposta loop for para diferntes k
k_range = list(range(1,26))
acertos = []
for k in k_range:
    modelo = KNeighborsClassifier(n_neighbors=k)
    modelo.fit(entradas_treino, classes_treino)
    classes_encontradas = modelo.predict(entradas_teste)
    acertos.append(accuracy_score(classes_teste, classes_encontradas))


plt.plot(k_range, acertos)
plt.xlabel('Valor de k do KNN')
plt.ylabel('Taxa de acertos')
plt.title('Taxa de acertos x valor de k do KNN')
plt.show()
```

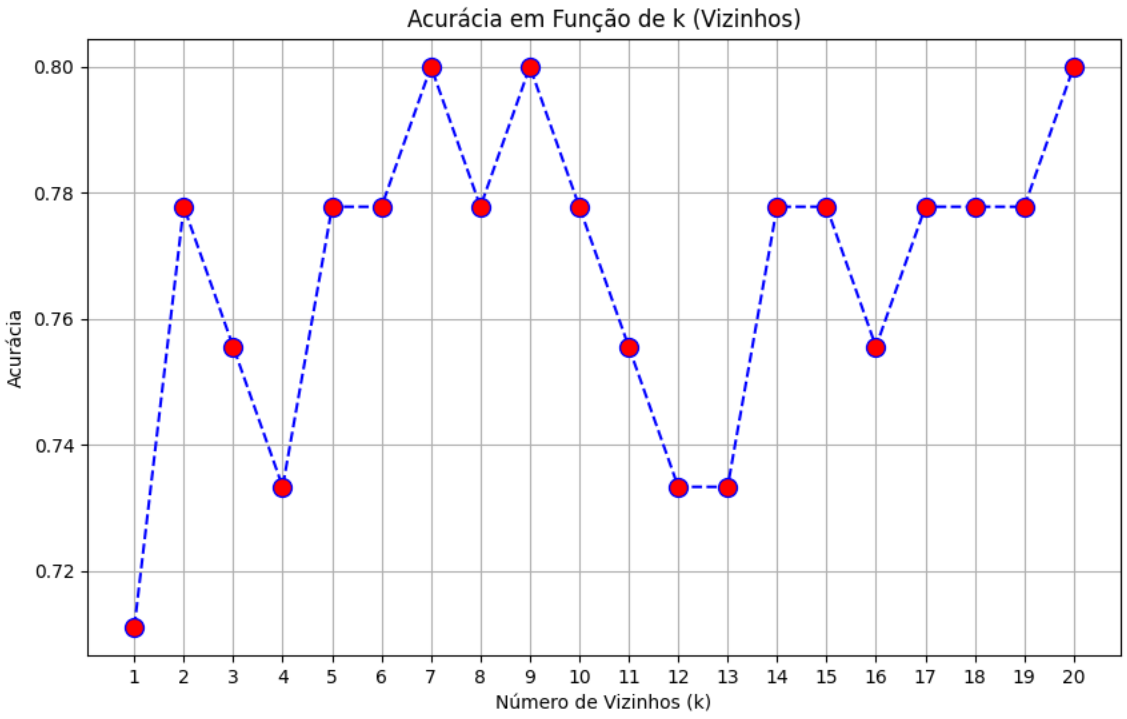


Desafio 4

Refaça os notebook substituindo as entradas (variaveis independentes) e analise se o modelo obtido ficou melhor ou pior.

```
# R. 4 Ao modificar as entradas, reduzindo-as somente para os dados das séplas, a acurácia do modelo classificatório diminui.

## implemente sua sua solução....
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Carregar os dados
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
header = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
df = pd.read_csv(url, header=None, names=header)

# Selecionar novas entradas (sepal_length e sepal_width)
```

```
novas_entradas = df[['sepal_length', 'sepal_width']]

# Separar as novas entradas e classes
entradas_treino, entradas_teste, classes_treino, classes_teste = train_test_split(novas_entradas, df['species'], test_size=0.3, random_state=

# Lista para armazenar as acurácias
acuracias = []

# Loop for para variar o valor de k de 1 a 20
for k in range(1, 21):
    # Criar o classificador KNN
    modelo = KNeighborsClassifier(n_neighbors=k)

    # Treinar o modelo
    modelo.fit(entradas_treino, classes_treino)

    # Fazer uma predição com dados de teste
    predicoes = modelo.predict(entradas_teste)

    # Calcular a acurácia e armazenar na lista
    acuracia = accuracy_score(classes_teste, predicoes)
    acuracias.append(acuracia)

# Scatterplot para visualizar acurácias em função de k
plt.figure(figsize=(10, 6))
plt.plot(range(1, 21), acuracias, marker='o', linestyle='dashed', color='b', markerfacecolor='r', markersize=10)
plt.title('Acurácia em Função de k (Vizinhos)')
plt.xlabel('Número de Vizinhos (k)')
plt.ylabel('Acurácia')
plt.xticks(range(1, 21))
plt.grid(True)
plt.show()
```



Desafio 5

Lembra o dataset 'breast_cancer', faça um modelo de predição que informa se o câncer é maligno ou não.

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```