

Relatório Trabalho 01 Inteligência Artificial

Alexandre Moraes Padilha das Neves

Questão 01

Para a base de dados Communities and Crime (disponível em <https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime>) faça:

- Faça as análises e alterações necessárias na base de dados para prever a variável ViolentCrimesPerPop usando regressão linear. Observe que essa base de dados possui valores faltantes. Explique as considerações e mudanças propostas.
- Divida aleatoriamente a base de dados em duas partes: treino, com 70% das amostras, e teste, com 30%. Use a parte de treino para estimar um modelo linear que melhor se ajuste aos dados. Obtenha os valores de RMSE e MAE sobre o conjunto de treino e teste.
- Aplique PCA sobre os dados de treino para reduzir os dados para 5 atributos. Realize análise gráfica sobre as variáveis e proponha alterações para melhorar o modelo de regressão linear (que poderá ser um modelo polinomial). Com esses atributos, obtenha os valores de RMSE e MAE sobre o conjunto de treino e teste. Compare com os resultados da letra b).

Link repositório Google Colab: [Communities and Crime.ipynb](#)

Instalação dos pacotes, conforme especificado no site (<https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime>) para download dos arquivos utilizados nesta questão

```
✓ 12s [1] #Instalação do pacote ucimlrepo
!pip install ucimlrepo
```

```
# Importa a função fetch_ucirepo da biblioteca ucimlrepo, usada para acessar datasets do UCI Machine Learning Repository.
from ucimlrepo import fetch_ucirepo

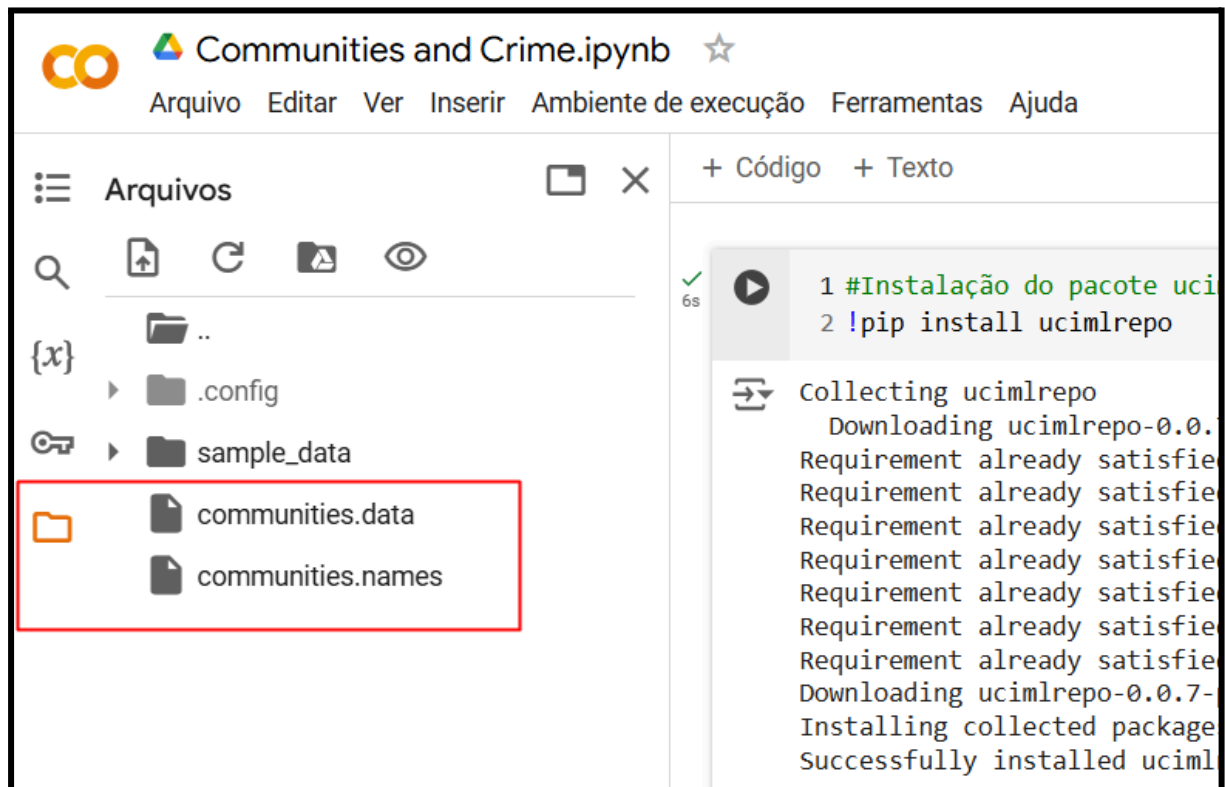
# fetch dataset
# Faz o download do dataset com o ID 183 do UCI Machine Learning Repository e armazena na variável communities_and_crime.
communities_and_crime = fetch_ucirepo(id=183)

# data (as pandas dataframes)
# Extrai as características (features) do dataset e as armazena na variável X, que será um DataFrame do pandas.
X = communities_and_crime.data.features
# Extrai os alvos (targets) do dataset e os armazena na variável y, que também será um DataFrame do pandas.
y = communities_and_crime.data.targets

# metadata
# mostra a metadata do dataset que contém informações gerais sobre o conjunto de dados, como descrição e o número de instâncias.
print(communities_and_crime.metadata)

# variable information
# mostra informações sobre as variáveis no dataset como nome, tipo e descrição de cada informação.
print(communities_and_crime.variables)
```

Em anexo ao Colab, para execução dos códigos utilizados no Colab, estão os arquivos *communities.data* e *communities.names* baixados através do site [‘archive’](#).



'import' de bibliotecas para exibir as tabelas com os dados

```
[3] 1 #importando bibliotecas para exibir as tabelas com os dados
      2 !pip3 install control
      3 import numpy as np
      4 import matplotlib.pyplot as plt
      5 import pandas as pd
```

```
[4] 1 #Import de bibliotecas
      2 from sklearn.model_selection import train_test_split
      3 from sklearn.linear_model import LinearRegression
      4 from sklearn.metrics import mean_squared_error
```

Fazendo a análise dos dados do arquivo 'communities.data'

```
[5] 1 #verificando data frame do arquivo communities.data
2 df_1 = pd.read_csv("communities.data")
3 df_1.head()
```

	8	?	?..1	Lakewoodcity	1	0.19	0.33	0.02	0.9	0.12	...	0.12.2	0.26.1	0.2.1	0.06.3	0.04.2	0.9.1	0.5.2	0.32.2	0.14.3	0.2.2
0	53	?	?	Tukwila	1	0.00	0.16	0.12	0.74	0.45	...	0.02	0.12	0.45	?	?	?	?	0.0	?	0.67
1	24	?	?	Aberdeen	1	0.00	0.42	0.49	0.56	0.17	...	0.01	0.21	0.02	?	?	?	?	0.0	?	0.43
2	34	5	81440	Willingboro	1	0.04	0.77	1.00	0.08	0.12	...	0.02	0.39	0.28	?	?	?	?	0.0	?	0.12
3	42	95	6096	Bethlehem	1	0.01	0.55	0.02	0.95	0.09	...	0.04	0.09	0.02	?	?	?	?	0.0	?	0.03
4	6	?	?	SouthPasadena	1	0.02	0.28	0.06	0.54	1.00	...	0.01	0.58	0.10	?	?	?	?	0.0	?	0.14

5 rows x 128 columns

Este trecho de código foi elaborado para fazer uma análise do arquivo 'communities.names' para verificar melhor suas informações e escolher nos próximos passos quais serão as variáveis importantes para elaborar o código sobre regressão linear.

```
✓ [30] 1 #Verificando texto do arquivo 'communities.names'
0s 2 with open("communities.names", "r") as file:
3     arquivoCommunities = file.read()
4
5 # Exibir o conteúdo do arquivo
6 print(arquivoCommunities)
```

➡ Title: Communities and Crime

Abstract: Communities within the United States. The data combines socio-economic data from the 1990 US Census, law enforcement data from the 1990 US LEMAS survey, and crime data from the 1995 FBI UCR.

Data Set Characteristics: Multivariate
Attribute Characteristics: Real
Associated Tasks: Regression
Number of Instances: 1994
Number of Attributes: 128
Missing Values? Yes

Em virtude do grande número de informações contendo no arquivo, foi feita uma análise das primeiras linhas do arquivo para entender a estrutura do mesmo.

```
1 #verificando os arquivos enviados para entender a estrutura dos dados
2 arquivoDados = 'communities.data'
3 arquivoNomes = 'communities.names'
4
5 #Verificando os primeiros 10 registros dos arquivos
6 with open(arquivoDados, 'r') as dadosArquivo:
7     conteudoDados = dadosArquivo.readlines()
8
9 with open(arquivoNomes, 'r') as nomes_file:
10     nomesConteudo = nomes_file.readlines()
11
12 # exibir as primeiras linhas do arquivo de dados e do arquivo de nomes para entender a estrutura
13 conteudoDados[:10], nomesConteudo[:10]
```

Fazendo uma análise das variáveis da base de dados, foi substituído os valores que continham '?' para 'NaN'. Irá ajudar nas próximas consultas a separar as variáveis relevantes para as amostras.

```

1 import pandas as pd
2 import numpy as np
3
4 #carregando os dados do arquivo communities.names e substituindo os valores faltantes que estão '?' para 'NaN'
5 nomesColunas = ['communityID', 'state', 'communityname', 'fold', 'population', 'householdsize', 'racepctblack', 'r
6                 'agePct12t29', 'agePct16t24', 'agePct65up', 'numUrban', 'pctUrban', 'medIncome', 'pctWage', 'pct
7                 'medFamInc', 'perCapInc', 'whitePerCap', 'blackPerCap', 'indianPerCap', 'AsianPerCap', 'OtherPerCa
8                 'PctNotHSGrad', 'PctBSorMore', 'PctUnemployed', 'PctEmploy', 'PctEmplManu', 'PctEmplProfServ', 'Pc
9                 'MalePctNevMarr', 'FemalePctDiv', 'TotalPctDiv', 'PersPerFam', 'PctFam2Par', 'PctKids2Par', 'PctYo
10                'PctWorkMom', 'NumIlleg', 'PctIlleg', 'NumImmig', 'PctImmigRecent', 'PctImmigRec5', 'PctImmigRec8'
11                'PctRecImmig5', 'PctRecImmig8', 'PctRecImmig10', 'PctSpeakEnglOnly', 'PctNotSpeakEnglWell', 'PctLa
12                'PersPerOccupHous', 'PersPerOwnOccHous', 'PersPerRentOccHous', 'PctPersOwnOccup', 'PctPersDenseHou
13                'HousVacant', 'PctHousOccup', 'PctHousOwnOcc', 'PctVacantBoarded', 'PctVacMore6Mos', 'MedYrHousBui
14                'OwnOccLowQuart', 'OwnOccMedQuart', 'OwnOccHiQuart', 'RentLowQ', 'RentMedian', 'RentHighQ', 'MedRe
15                'MedOwnCostPctIncNoMtg', 'NumInShelters', 'NumStreet', 'PctForeignBorn', 'PctBornSameState', 'PctS
16                'LemasSwornFT', 'LemasSwFTPerPop', 'LemasSwFTFieldOps', 'LemasSwFTFieldPerPop', 'LemasTotalReq', '
17                'RacialMatchCommPol', 'PctPolicWhite', 'PctPolicBlack', 'PctPolicHisp', 'PctPolicAsian', 'PctPolic
18                'PolicAveOTWorked', 'LandArea', 'PopDens', 'PctUsePubTrans', 'PolicCars', 'PolicOperBudg', 'LemasP
19                'LemasPctOfficDrugUn', 'PolicBudgPerPop', 'ViolentCrimesPerPop']

```

Aqui os detalhes de cada linha e operação estão comentados

```

[48] 1 #Carregar os dados para o data frame
2 df = pd.read_csv(arquivoDados, header=None, names=nomesColunas, na_values='?')
3
4 #Verificar a presença de valores faltantes e remover colunas irrelevantes
5 df = df.drop(['communityname', 'state'], axis=1)
6 missing_values = df.isna().sum()
7
8 #Exibir os primeiros registros e a contagem de valores faltantes
9 df.head(), missing_values

```

	communityID	fold	population	householdsize	racepctblack	racePctWhite	\
8	NaN	1	0.19	0.33	0.02	0.90	
53	NaN	1	0.00	0.16	0.12	0.74	
24	NaN	1	0.00	0.42	0.49	0.56	
34	5.0	1	0.04	0.77	1.00	0.08	
42	95.0	1	0.01	0.55	0.02	0.95	

Nesse trecho verifica-se as variáveis que contém valores faltantes 'NaN'. Observando estas colunas com estes valores faltantes, as mesmas já foram descartadas para as análises das amostras.

```
1 #verificando a presença de valores faltantes no DataFrame nas colunas
2 df.head()
```

agePct12t29	...	LandArea	PopDens	PctUsePubTrans	PolicCars	PolicOperBudg	LemasPctPolicOnPatr	LemasGangUnitDeploy
0.47	...	0.12	0.26	0.20	0.06	0.04	0.9	0.5
0.59	...	0.02	0.12	0.45	NaN	NaN	NaN	NaN
0.47	...	0.01	0.21	0.02	NaN	NaN	NaN	NaN
0.50	...	0.02	0.39	0.28	NaN	NaN	NaN	NaN
0.38	...	0.04	0.09	0.02	NaN	NaN	NaN	NaN

Conforme explicado na imagem abaixo, foram selecionadas apenas as variáveis relacionadas à raça, idade, população e tamanho da área de atuação. São características que não continham valores faltantes na base de dados e serão relevantes para fazer uma análise para a variável '*ViolentCrimesPerPop*'.

Removendo colunas

No arquivo foi notado muitas colunas com valores faltantes. Diante disso, nos próximos códigos, foi separada colunas relevantes para fazer as análises sobre a variável '*ViolentCrimesPerPop*'. O foco nesta análise foi para pegar dados referente a raça, idade, população e tamanho da área.

```
[53] 1 #lista de colunas a serem mantidas
2 colunasAManter = ['population', 'householdsize',
3                  'racePctblack', 'racePctWhite', 'racePctAsian', 'racePctHisp',
4                  'agePct12t21', 'agePct12t29', 'agePct16t24',
5                  'ViolentCrimesPerPop']
6
7 #selecionando apenas as colunas desejadas do dataframe
8 df = df[colunasAManter]
```

Ao executar novamente o código com `df.head()`, retorna-se o resultado apenas com as variáveis necessárias para continuar o trabalho.

```
[51] 1 #confirmando as colunas conforme alteração no código da janela anterior
2 df.head()
```

	population	householdsize	racepctblack	racePctWhite	racePctAsian	racePctHisp	agePct12t21	agePct12t29	agePct16t24	ViolentCrimesPerPop
8	0.19	0.33	0.02	0.90	0.12	0.17	0.34	0.47	0.29	0.20
53	0.00	0.16	0.12	0.74	0.45	0.07	0.26	0.59	0.35	0.67
24	0.00	0.42	0.49	0.56	0.17	0.04	0.39	0.47	0.28	0.43
34	0.04	0.77	1.00	0.08	0.12	0.10	0.51	0.50	0.34	0.12
42	0.01	0.55	0.02	0.95	0.09	0.05	0.38	0.38	0.23	0.03

Próximas etapas: [Gerar código com df](#) [Ver gráficos recomendados](#) [New interactive chart](#)

Seguindo com trabalho, vamos fazer o passo de selecionar as características e divisão de dados da variável escolhida para fazer as análises: *ViolentCrimesPerPop*. Os detalhes foram descritos no Colab:

Selecione as características e divisão dos dados da variável

'ViolentCrimesPerPop'

Este passo visa selecionar as características (ou variáveis) que serão usadas para treinar nosso modelo de regressão.

No caso deste trabalho:

Variáveis Independentes (X): Estas são todas as colunas de nosso conjunto de dados, exceto a coluna *ViolentCrimesPerPop*, como vimos outras colunas no código rodado anteriormente. Será usada essa característica como 'previsão' do número de crimes violentos por população

Variável Dependente (y): Esta é a coluna *ViolentCrimesPerPop*. É a variável que queremos prever com base em outras características que escolhemos para fazer a regressão linear.

```
[13] 1 # Seleção de características e divisão dos dados
      2 X = df.drop('ViolentCrimesPerPop', axis=1) # Usando todas as variáveis escolhidas anteriormente, exceto 'ViolentCrimesPerPop' como variáveis independentes
      3 y = df['ViolentCrimesPerPop'] # 'ViolentCrimesPerPop' é a variável dependente
```

Conforme ao que é solicitado na letra b do exercício, aqui foi dividida a base de dados em treino com 70% das amostras e de teste com 30%

Divisão dos Dados em treinamento e teste

A prática da divisão em dividir o conjunto de dados para treino e teste ajudará a avaliar como o modelo irá se comportar com dados que não viu antes.

Utilizamos a função `train_test_split` da biblioteca `sklearn` para realizar essa divisão. Conforme especificado na **letra b** deste exercício, foi dividida aleatoriamente a base de dados de treino (train) com 70% das amostras e de teste (test), com 30%.

```
[14] 1 #Treinamento de modelo com 70% das amostras e testes de performance com 30%.
      2 #Visualizar como o modelo irá se comportar com dados não vistos antes.
      3 #Random 42 garante que teremos a mesma divisão, útil para visualização dos resultados.
      4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
      5
```

Com isso, temos:

X_train e *y_train*: dados selecionados para a análise do modelo e crimes por população, respectivamente, para o conjunto de treinamento.

X_test e *y_test*: dados selecionados para a análise do modelo e crimes por população, respectivamente, para o conjunto de teste.

Continuando com a elaboração do código para fazer o modelo de regressão linear, aqui foi feito o treinamento do modelo. Os detalhes para os procedimentos estão especificados na imagem abaixo:

▼ Treinamento do Modelo

```
[15] 1 #Utilização da classe LinearRegression da biblioteca sklearn
      2 #Inicialização do modelo
      3 model = LinearRegression()
```

```
✓ [16] 1 #Utilização do método fit para treinar o modelo usando o conjunto de treinamento.
      2 #Ele ajusta o modelo aos dados, "aprendendo" a relação entre as características (X_train) e a variável alvo (y_train).
      3 model.fit(X_train, y_train)
```

↗

LinearRegression ⓘ ⓘ
LinearRegression()

Após essas etapas, o modelo está pronto para fazer previsões em novos dados. O algoritmo de regressão linear, neste ponto, já determinou os coeficientes (ou pesos) ideais para cada característica de forma a minimizar o erro entre as previsões do modelo e os valores reais.

A seguir está uma forma de visualização dos coeficientes das características e a interceptação.

```
✓ [17] 1 # Criando um DataFrame para visualizar melhor os coeficientes
      2 coeff_df = pd.DataFrame({
      3     'Feature': X.columns,
      4     'Coefficient': model.coef_
      5 })
      6 print(coeff_df)
      7
      8 # Mostrando também a interceptação
      9 print("\nInterceptação:", model.intercept_)
```

↗

	Feature	Coefficient
0	population	0.269737
1	householdsize	-0.274087
2	racepctblack	0.363469
3	racePctWhite	-0.253252
4	racePctAsian	-0.067165
5	racePctHisp	0.270572
6	agePct12t21	-0.081086
7	agePct12t29	-0.003617
8	agePct16t24	0.121236

Interceptação: 0.44492422476966

Realização e visualização das previsões

▼ Realização e visualização das previsões

Após o treinamento, é hora de testar o modelo em um conjunto de dados não vistos anteriormente, o conjunto de teste. Isso nos dá uma noção de como o modelo se comportará em dados novos e desconhecidos.

Realizando previsões: Usamos o método `predict` do modelo para prever os valores da variável alvo no conjunto de teste.

```
[18] 1 #Teste do modelo de conjunto de dados para verificar como o modelo se comportará em novos dados e desconhecidos.  
      2 #Y_pred tem a função de prever valores futuros, a partir de suas características  
      3  
      4 # Gerando previsões no conjunto de teste  
      5 y_pred = model.predict(X_test)  
      6  
      7 # Gerando previsões no conjunto de teste  
      8 y_pred_test = model.predict(X_test)
```

Ainda sobre a **letra b** desta questão do trabalho, a seguir foi se obtido os valores de RMSE e MAE sobre o conjunto de treino e teste:

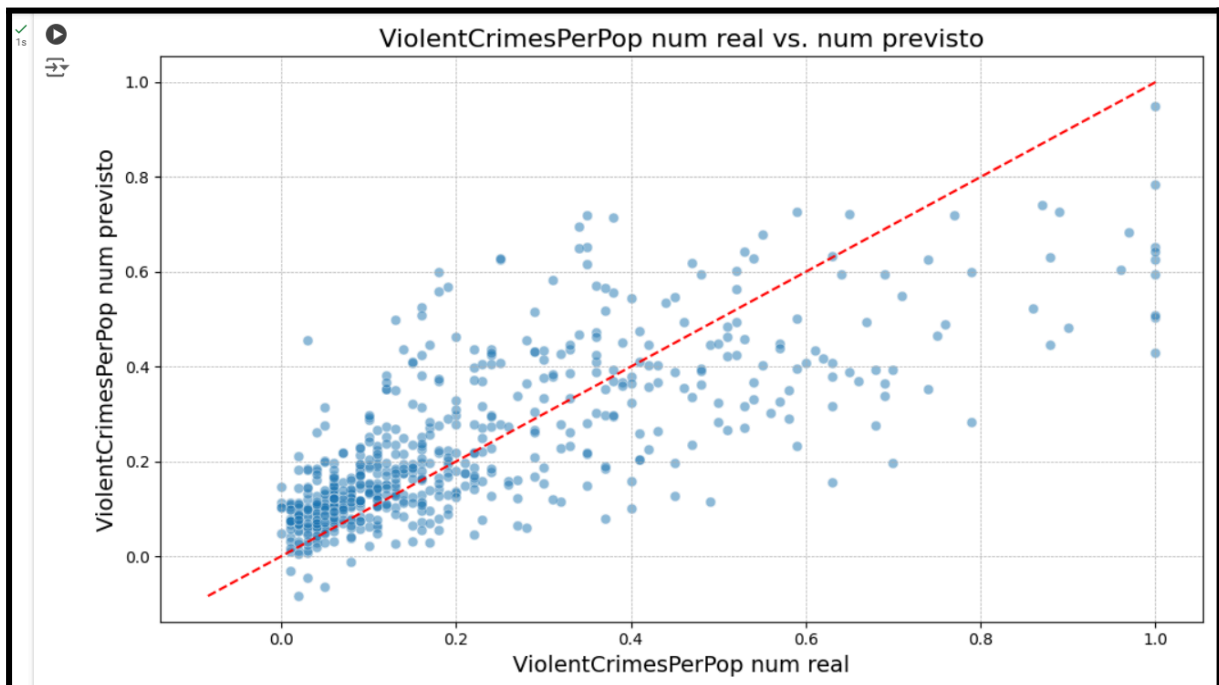
```
✓ 0s ▶ 1 # Importando os módulos necessários  
      2 from sklearn.metrics import mean_squared_error, mean_absolute_error  
      3  
      4 # Calculando RMSE e MAE para o conjunto de treino  
      5 # Usando y_test (os valores reais do conjunto de teste) para comparação  
      6 train_mse = mean_squared_error(y_test, y_pred)  
      7 train_mae = mean_absolute_error(y_test, y_pred)  
      8 train_rmse = np.sqrt(train_mse)  
  
✓ 0s [20] 1 # Calculando RMSE e MAE para o conjunto de teste  
         2 test_mse = mean_squared_error(y_test, y_pred_test)  
         3 test_mae = mean_absolute_error(y_test, y_pred_test)  
         4 test_rmse = np.sqrt(test_mse)  
         5  
         6 # Imprimindo os resultados  
         7 print('Treino - RMSE: %.4f' % train_rmse)  
         8 print('Treino - MAE: %.4f' % train_mae)  
         9 print('Teste - RMSE: %.4f' % test_rmse)  
        10 print('Teste - MAE: %.4f' % test_mae)  
  
⇒ Treino - RMSE: 0.1445  
   Treino - MAE: 0.1042  
   Teste - RMSE: 0.1445  
   Teste - MAE: 0.1042
```

Comparando Previsões e Valores Reais: Para avaliar a eficácia do modelo, é utilizado um gráfico de dispersão. No eixo X, são colocados os valores reais dos preços das casas, enquanto no eixo Y, temos os valores previstos pelo modelo.

Em um modelo ideal, todas as previsões cairiam exatamente sobre a linha vermelha pontilhada (representando a linha de previsão perfeita). Quanto mais os pontos estiverem próximos dessa linha, mais precisas são as previsões do modelo.

```
1 # Tamanho e resolução do gráfico
2 plt.figure(figsize=(10,6), dpi=100)
3
4 # Scatter plot
5 plt.scatter(y_test, y_pred, alpha=0.5, s=40, edgecolors="w", linewidth=0.5)
6
7 # Linha de previsão perfeita
8 min_val = min(min(y_test), min(y_pred))
9 max_val = max(max(y_test), max(y_pred))
10 plt.plot([min_val, max_val], [min_val, max_val], color='red', linestyle='--')
11
12 # Labels e título
13 plt.xlabel('ViolentCrimesPerPop num real', fontsize=14)
14 plt.ylabel('ViolentCrimesPerPop num previsto', fontsize=14)
15 plt.title('ViolentCrimesPerPop num real vs. num previsto', fontsize=16)
16
17 # Mostrar gráfico
18 plt.tight_layout()
19 plt.grid(True, which='both', linestyle='--', linewidth=0.5)
20 plt.show()
```

resultado do modelo:



Aqui é exibido o valor do coeficiente de determinação (R^2) do modelo.

```
✓ [22] 1 print("R quadrado = {}".format(model.score(X_test, y_test)))  
0s  
⇒ R quadrado = 0.5639719308946851
```

O valor do coeficiente indica que o modelo de regressão explica cerca de 56.4% da variabilidade dos dados de saída no conjunto de testes. Isso significa que o modelo está capturando um pouco mais da metade da variância presente nos dados, mas deixa uma grande parcela sem explicação.

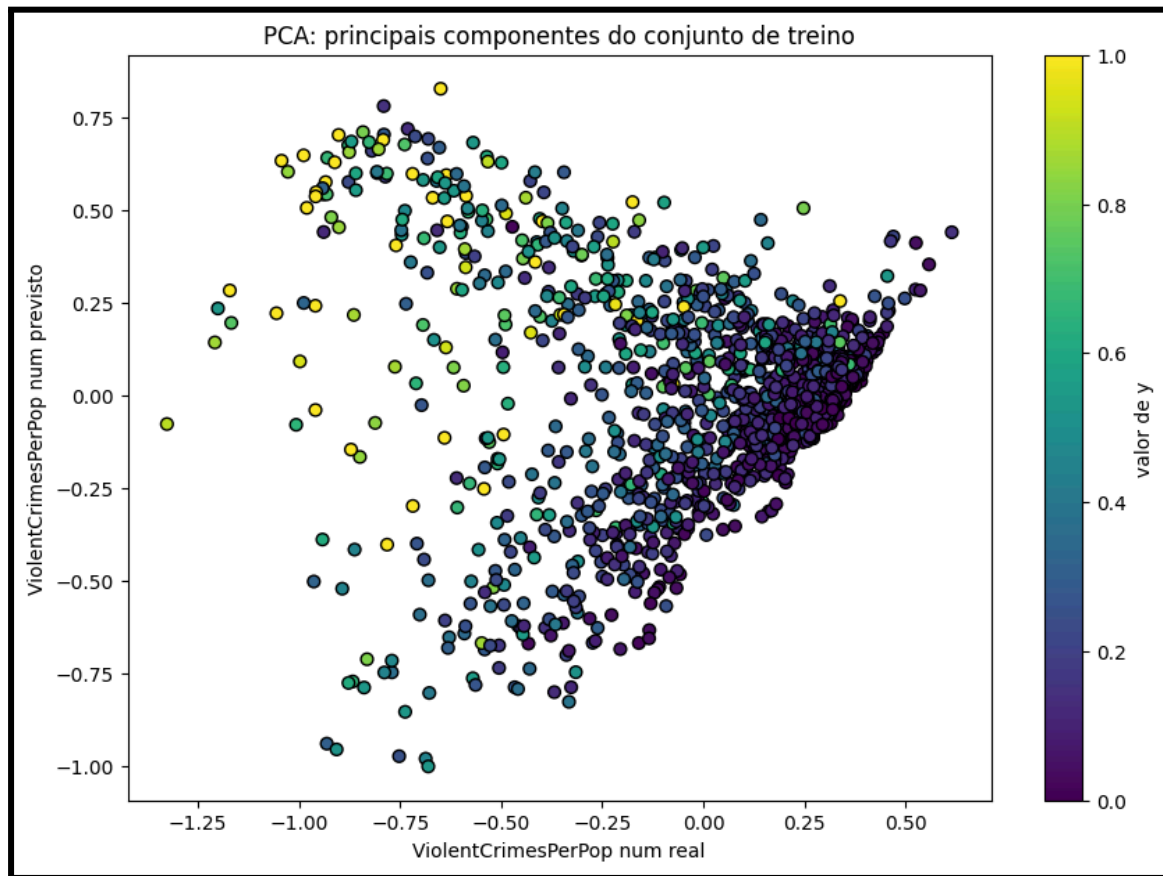
Já para a **letra c** da questão 1, neste passo houve a importação de bibliotecas necessárias para o decorrer do código. Aqui foi aplicado PCA para reduzir os dados para 5 atributos.

```
✓ [22] 1 # Importando bibliotecas necessárias para a #letra c da questão 1  
0s 2 from sklearn.decomposition import PCA  
3 from sklearn.preprocessing import PolynomialFeatures  
4 from sklearn.linear_model import LinearRegression  
5 from sklearn.metrics import mean_squared_error, mean_absolute_error  
6 import numpy as np  
7 import matplotlib.pyplot as plt  
8  
9 #aplicação do PCA para reduzir para 5 atributos  
10 pca = PCA(n_components=5) # PCA com 5 atributos principais  
11 X_train_pca = pca.fit_transform(X_train) # PCA de treino  
12 X_test_pca = pca.transform(X_test) #PCA de teste
```

Agora visualiza-se os principais componentes do conjunto de treino

```
✓ [22] 1 # Visualização dos componentes principais  
1s 2 plt.figure(figsize=(10, 7))  
3 plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap='viridis', edgecolor='k', s=40)  
4 plt.title('PCA: principais componentes do conjunto de treino')  
5 plt.xlabel('ViolentCrimesPerPop num real')  
6 plt.ylabel('ViolentCrimesPerPop num previsto')  
7 plt.colorbar(label='valor de y')  
8 plt.show()
```

Realizando a análise gráfica sobre as variáveis. Percebe-se uma grande quantidade de componentes entre 0,00 e 0,25



Iniciando o processo para melhoria do modelo de regressão linear, utilizando a *transformação polinomial*

```
1 # Transformação polinomial, como forma p melhorar o modelo
2 poliPCA = PolynomialFeatures(degree=2) # grau do polinômio em 2
3 X_train_poli = poliPCA.fit_transform(X_train_pca) #PCA poli treino
4 X_test_poli = poliPCA.transform(X_test_pca) #PCA poli teste
5
6 #utilizando o fit para treinar o modelo
7 model_poli = LinearRegression()
8 model_poli.fit(X_train_poli, y_train)
```

LinearRegression 1 2

LinearRegression()

Obtendo os valores de RMSE e MAE sobre o conjunto de treino e teste.

```
[64] 1 #previsões do modelo
      2 y_pred_train_poli = model_poli.predict(X_train_poli)
      3 y_pred_test_poli = model_poli.predict(X_test_poli)

1 #obtendo os valores de RMSE e MAE sobre o conjunto de treino
2 train_mse_poli = mean_squared_error(y_train, y_pred_train_poli)
3 train_mae_poli = mean_absolute_error(y_train, y_pred_train_poli)
4 train_rmse_poli = np.sqrt(train_mse_poli)
5
6 #obtendo os valores de RMSE e MAE sobre o conjunto de test
7 test_mse_poli = mean_squared_error(y_test, y_pred_test_poli)
8 test_mae_poli = mean_absolute_error(y_test, y_pred_test_poli)
9 test_rmse_poli = np.sqrt(test_mse_poli)
10 #print dos resultados
11 print('Treino (PCA + polinomial) - RMSE: %.4f' % train_rmse_poli)
12 print('Treino (PCA + polinomial) - MAE : %.4f' % train_mae_poli)
13 print('Teste(PCA + polinomial) - RMSE: %.4f' % test_rmse_poli)
14 print('Teste (PCA + polinomial) - MAE: %.4f' % test_mae_poli)
```

⇒ Treino (PCA + polinomial) - RMSE: 0.1531
Treino (PCA + polinomial) - MAE : 0.1065
Teste(PCA + polinomial) - RMSE: 0.1418
Teste (PCA + polinomial) - MAE: 0.1014

Agora a comparação entre resultados:

```
[68] 1 # Comparação com os resultados letra b e c
      2 print('Conjunto de treino e teste MAE e RSME sem poli')
      3 print('Treino - RMSE: %.4f' % train_rmse)
      4 print('Treino - MAE: %.4f' % train_mae)
      5 print('Teste - RMSE: %.4f' % test_rmse)
      6 print('Teste - MAE: %.4f' % test_mae)
      7 print('-----')
      8 print('Conjunto de treino e teste polinomial')
      9 print('Treino (PCA + polinomial) - RMSE: %.4f' % train_rmse_poli)
     10 print('Treino (PCA + polinomial) - MAE : %.4f' % train_mae_poli)
     11 print('Teste(PCA + polinomial) - RMSE: %.4f' % test_rmse_poli)
     12 print('Teste (PCA + polinomial) - MAE: %.4f' % test_mae_poli)
```

⇒ Conjunto de treino e teste MAE e RSME sem poli
Treino - RMSE: 0.1445
Treino - MAE: 0.1042
Teste - RMSE: 0.1445
Teste - MAE: 0.1042

Conjunto de treino e teste polinomial
Treino (PCA + polinomial) - RMSE: 0.1531
Treino (PCA + polinomial) - MAE : 0.1065
Teste(PCA + polinomial) - RMSE: 0.1418
Teste (PCA + polinomial) - MAE: 0.1014

Observa-se que o modelo sem polinômio mostra um ajuste um pouco melhor no conjunto de treino, mas um pouco menor no de teste. Com polinômio, mesmo com erro um pouco maior no conjunto de treino, apresenta um erro menor no conjunto de teste, tendo uma pequena vantagem nos números dos resultados. Essa vantagem pode ser preponderante pelo modelo polinomial do conjunto de teste ser um indicativo que ele consegue pegar algumas relações não lineares nos dados.

Questão 02

2. Realize a classificação da base de dados HTRU2 (disponível em <https://archive.ics.uci.edu/ml/datasets/HTRU2>) usando o esquema de validação holdout. Para cada execução, use 6000 amostras de treino selecionadas aleatoriamente e o restante para teste (normalmente o conjunto de treinamento é maior do que de teste, mas para reduzir o custo computacional ele foi reduzido aqui). Execute 5 vezes o treinamento e teste e retorne a acurácia, recall e precisão média para cada algoritmo. Faça a classificação usando:

- kNN com métrica de distância Euclidiana. Escolha 5 valores diferentes de k. Para selecionar o melhor valor de k divida a base de treinamento em duas partes iguais: uma para treinar e a outra para validar e encontrar o melhor valor de k;
- Compare os resultados, tempos de execução e número de protótipos usados por cada valor de k da letra a). Considerando a distribuição das classes, você considera o valor da acurácia média relevante? Por quê?

Link repositório Google Colab: [HTRU2.ipynb](#)

Instalação dos pacotes, conforme especificado no site (<https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime>) para download dos arquivos utilizados nesta questão

```
[1] 1 #Install the ucimlrepo package
    2 !pip install ucimlrepo

Collecting ucimlrepo
  Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ucimlrepo)
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.10/dist-packages (from ucimlrepo)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-packages (from ucimlrepo)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from ucimlrepo)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from ucimlrepo)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from ucimlrepo)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from ucimlrepo)
  Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7

1 from ucimlrepo import fetch_ucirepo
2
3 # fetch dataset
4 htru2 = fetch_ucirepo(id=372)
5
6 # data (as pandas dataframes)
7 X = htru2.data.features
8 y = htru2.data.targets
9
10 # metadata
11 print(htru2.metadata)
12
13 # variable information
14 print(htru2.variables)
```

Primeiro a importação de bibliotecas necessárias para resolução do exercício. Após isso, é preparado o código para executá-lo 5 vezes a etapa de treinamento e teste. Sendo assim retornará o resultado da acurácia, recall e precisão média para cada algoritmo.

```
✓ [15] 1 #importando bibliotecas de treino, acurácia, precisão e recall
0s      2 from sklearn.model_selection import train_test_split
      3 from sklearn.metrics import accuracy_score, precision_score, recall_score
      4 from sklearn.neighbors import KNeighborsClassifier
      5 import numpy as np
      6
      7 #função para calcular métricas médias apos as cinco execuções
      8 def calculaMetrica(model, X_train, X_test, y_train, y_test):
      9     model.fit(X_train, y_train)
     10     y_pred = model.predict(X_test)
     11     acuracia = accuracy_score(y_test, y_pred)
     12     precisao = precision_score(y_test, y_pred)
     13     recall = recall_score(y_test, y_pred)
     14     return acuracia, precisao, recall
```

Listando o modelo KNN para exibir o resultado dele nos próximos passos. Em seguida citada as variáveis para armazenamento dos resultados a serem calculados para acurácia, precisão e recall

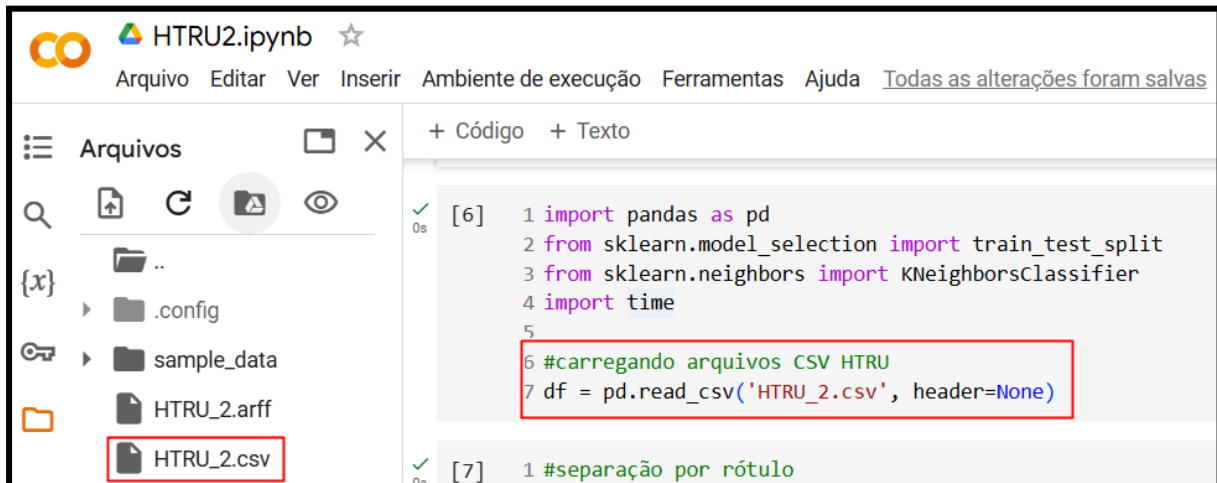
```
✓ [19] 1 #modelo KNN para armazenar exibir os resultados
0s      2 modelos = {
      3     "KNN": KNeighborsClassifier()
      4 }
      5
      6 # Variáveis para armazenar os resultados
      7 resultados = {nomeModelo: {"acurácia": [], "precisão": [], "recall": []} for nomeModelo in modelos.keys()}
```

Utilização de 6000 amostras de treino selecionadas aleatoriamente executando 5 vezes o treinamento e teste. Com isso exibindo os resultados da acurácia, recall e precisão média para cada algoritmo.

```
✓ [21] 1 #utilização do 'for' para repetir 5 vezes a divisão do esquema de validação holdout e o treinamento para cada modelo
3s      2 for _ in range(5):
      3     # Utilizando 6000 amostras para treino e teste selecionadas aleatoriamente
      4     X_train, X_test, y_train, y_test = train_test_split(X, y.values.ravel(), train_size=6000, random_state=None)
      5
      6     #calculando as métricas para cada modelo
      7     for nomeModelo, modelo in modelos.items():
      8         acuracia, precisao, recall = calculaMetrica(modelo, X_train, X_test, y_train, y_test)
      9         resultados[nomeModelo]["acurácia"].append(acuracia)
     10         resultados[nomeModelo]["precisão"].append(precisao)
     11         resultados[nomeModelo]["recall"].append(recall)
     12
     13 #calculando as médias das métricas para cada algoritmo
     14 averages = {nomeModelo: {metric: np.mean(scores) for metric, scores in metrics.items()} for nomeModelo, metrics in resultados.items()}
     15 averages
```

```
{'KNN': {'acurácia': 0.9720625315179022,
'precisão': 0.9042128236212413,
'recall': 0.7764405289986965}}
```

Seguindo com o exercício, neste momento foi executado arquivo CSV anexo ao Colab para prosseguir com a resolução da **letra a** desta segunda questão.



The screenshot shows a Jupyter Notebook titled 'HTRU2.ipynb'. The left sidebar contains a file explorer with a tree view showing folders like '.config' and 'sample_data', and files 'HTRU_2.arff' and 'HTRU_2.csv'. The 'HTRU_2.csv' file is highlighted with a red box. The main area shows two code cells. Cell [6] contains the following code:

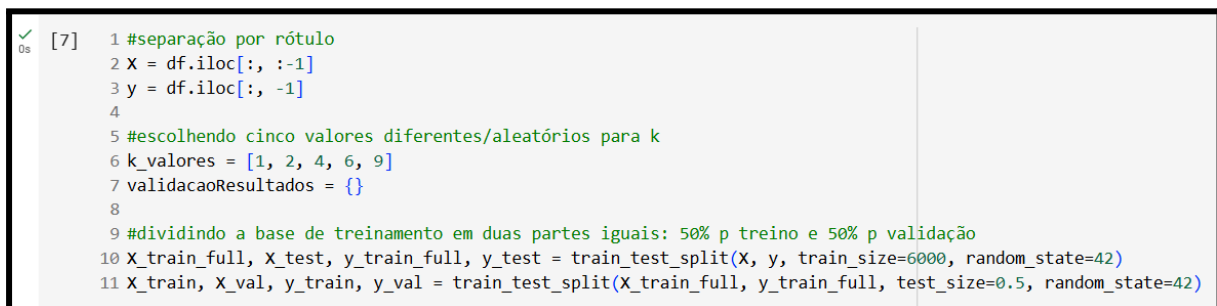
```
[6] 1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3 from sklearn.neighbors import KNeighborsClassifier
4 import time
5
6 #carregando arquivos CSV HTRU
7 df = pd.read_csv('HTRU_2.csv', header=None)
```

Cell [7] contains the following code:

```
[7] 1 #separação por rótulo
```

Conforme segue ao que é solicitado na **letra a** da questão 2:

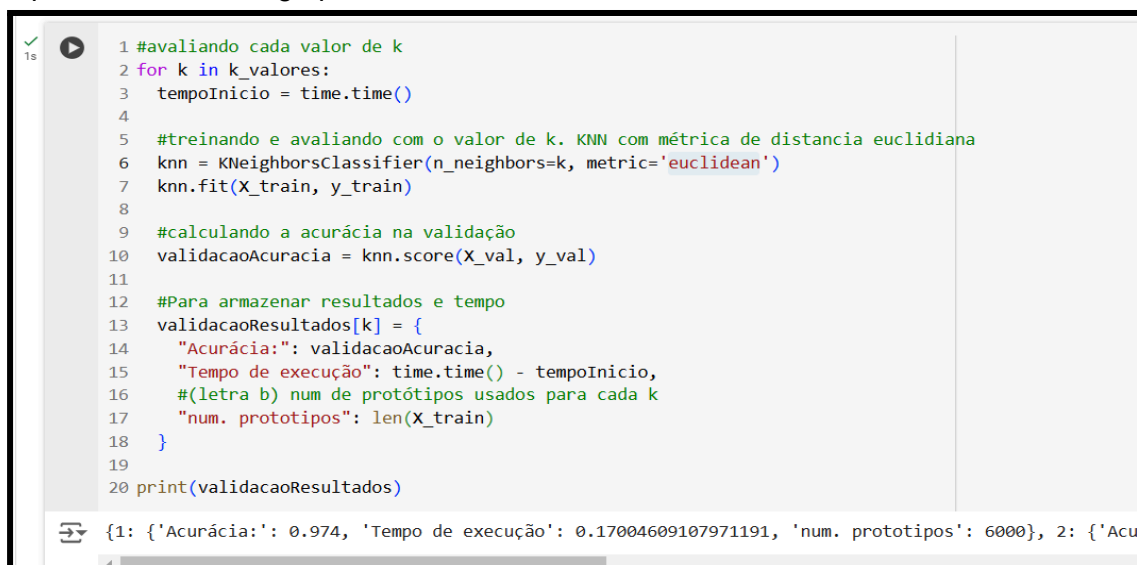
“kNN com métrica de distância Euclidiana. **Escolha 5 valores diferentes de k . Para selecionar o melhor valor de k divida a base de treinamento em duas partes iguais: uma para treinar e a outra para validar e encontrar o melhor valor de k ;**”



The screenshot shows a code cell [7] with the following code:

```
[7] 1 #separação por rótulo
2 X = df.iloc[:, :-1]
3 y = df.iloc[:, -1]
4
5 #escolhendo cinco valores diferentes/aleatórios para k
6 k_valores = [1, 2, 4, 6, 9]
7 validacaoResultados = {}
8
9 #dividindo a base de treinamento em duas partes iguais: 50% p treino e 50% p validação
10 X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, train_size=6000, random_state=42)
11 X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.5, random_state=42)
```

Aqui o trecho de código para calcular kNN com a métrica de distância Euclidiana.



The screenshot shows a code cell [15] with the following code:

```
[15] 1 #avaliando cada valor de k
2 for k in k_valores:
3     tempoInicio = time.time()
4
5     #treinando e avaliando com o valor de k. KNN com métrica de distancia euclidiana
6     knn = KNeighborsClassifier(n_neighbors=k, metric='euclidean')
7     knn.fit(X_train, y_train)
8
9     #calculando a acurácia na validação
10    validacaoAcuracia = knn.score(X_val, y_val)
11
12    #Para armazenar resultados e tempo
13    validacaoResultados[k] = {
14        "Acurácia": validacaoAcuracia,
15        "Tempo de execução": time.time() - tempoInicio,
16        #(letra b) num de protótipos usados para cada k
17        "num. prototipos": len(X_train)
18    }
19
20 print(validacaoResultados)
```

The output of the code is displayed at the bottom:

```
{1: {'Acurácia': 0.974, 'Tempo de execução': 0.17004609107971191, 'num. prototipos': 6000}, 2: {'Acurácia': 0.974, 'Tempo de execução': 0.17004609107971191, 'num. prototipos': 6000}, 4: {'Acurácia': 0.974, 'Tempo de execução': 0.17004609107971191, 'num. prototipos': 6000}, 6: {'Acurácia': 0.974, 'Tempo de execução': 0.17004609107971191, 'num. prototipos': 6000}, 9: {'Acurácia': 0.974, 'Tempo de execução': 0.17004609107971191, 'num. prototipos': 6000}}
```


Posteriormente o cálculo para acurácia e verificar se sua média será relevante.

Como não deu para mostrar no print os resultados do código, segue abaixo os resultados dos 5 valores diferentes de k escolhidos aleatoriamente:

```
{1: {'Acurácia': 0.974,  
'Tempo de execução': 0.17004609107971191,  
'num. protótipos': 6000},
```

```
2: {'Acurácia': 0.975,  
'Tempo de execução': 0.15946030616760254,  
'num. protótipos': 6000},
```

```
4: {'Acurácia': 0.9766666666666667,  
'Tempo de execução': 0.1669762134552002,  
'num. protótipos': 6000},
```

```
6: {'Acurácia': 0.9746666666666667,  
'Tempo de execução': 0.16246819496154785,  
'num. protótipos': 6000},
```



```
9: {'Acurácia': 0.9746666666666667,  
'Tempo de execução': 0.1692667007446289,  
'num. protótipos': 6000}}
```

Comparando os resultados do tempo de execução e número de protótipos em cada valor de k, os valores de acurácia obteve pouca diferença entre variações. Com isso sugere-se que o modelo esteja agindo em uma performance consistente. Assim como o tempo de execução obtém-se pouca diferença também entre um e outro.


Tendo a diferença de acurácia entre os valores de k não tão grande, a média acaba não sendo relevante, já que os resultados foram bem parecidos. Então, ao escolher um valor de k, vale só focar no equilíbrio entre um bom desempenho e um tempo de resposta rápido, ainda mais se o sistema precisar processar muitos dados.

Sendo assim, escolher o valor k4k é uma boa opção, pois oferece a melhor acurácia (0.9766666666666667) entre as alternativas e mantém o tempo de execução em um nível aceitável.

Verificando a proporção entre as classes

 0s 

```
1 #Verificando a proporção das classes
2 contasClasses = y.value_counts(normalize=True)
3 print("Proporção das classes:")
4 print(contasClasses)
```

 Proporção das classes:
8
0 0.908426
1 0.091574
Name: proportion, dtype: float64

Questão 03

Para a base Nursery (disponível em <https://archive.ics.uci.edu/dataset/76/nursery>):

- Construa uma árvore de decisão com dois níveis de nó de decisão (isto é, o primeiro nó de decisão (primeiro nível), os nós de decisão abaixo dele (segundo nível) e em seguida os nós folha) usando a medida de Ganho de Informação. Selecione aleatoriamente 10000 amostras dos dados para treinamento que serão usados para construir a árvore. Retorne a estrutura da árvore construída.
- Use os restantes dos dados para avaliação. Retorne a acurácia obtida.
- Tente obter as regras de decisão a partir da árvore construída.

Link repositório Google Colab:  Nursery.ipynb

Instalação de bibliotecas necessárias para execução do código

```
✓ [1] 1 #Install the ucimlrepo package
4s    2 !pip install ucimlrepo
    3

Collecting ucimlrepo
  Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/py
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/l
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/py
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/loc
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/pyt
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/p
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3
Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7

✓ [2] 1 from ucimlrepo import fetch_ucirepo
2s    2
    3 # fetch dataset
    4 nursery = fetch_ucirepo(id=76)
    5
    6 # data (as pandas dataframes)
    7 X = nursery.data.features
    8 y = nursery.data.targets
    9
   10 # metadata
   11 print(nursery.metadata)
   12
   13 # variable information
   14 print(nursery.variables)
```

Primeiro, carregar os dados da base *Nursery*, explorar o mesmo e verificar as características da base.

```
1 #import pandas para manipulação e análise de dados
2 import pandas as pd
3
4 #carregando a base de dados Nursery
5 df_nursery = pd.read_csv('/content/nursery.data', header=None)
6
7 #Exibindo as primeiras linhas e a distribuição das classes para análise
8 print("Formato do dataset:", df_nursery.shape)
9 print("Exemplo de dados:")
10 print(df_nursery.head())
```

Formato do dataset: (12960, 9)
Exemplo de dados:

	0	1	2	3	4	5	6	\
0	usual	proper	complete	1	convenient	convenient	nonprob	
1	usual	proper	complete	1	convenient	convenient	nonprob	
2	usual	proper	complete	1	convenient	convenient	nonprob	
3	usual	proper	complete	1	convenient	convenient	slightly_prob	
4	usual	proper	complete	1	convenient	convenient	slightly_prob	

	7	8
0	recommended	recommend
1	priority	priority
2	not_recom	not_recom
3	recommended	recommend
4	priority	priority

Agora as amostras serão divididas em 10.000 para treinamento e teste.

```
1 #Import da função para dividir os dados em conjuntos de treino e teste
2 from sklearn.model_selection import train_test_split
3
4 #dividindo os dados
5 X = df_nursery.iloc[:, :-1] #todas as colunas exceto a última
6 y = df_nursery.iloc[:, -1] #última coluna para ser a classe
7
8 #selecionando as 10.000 amostras para o treinamento
9 X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=10000, random_state=42)
10 print("Tamanho do conjunto de treinamento:", X_train.shape)
11 print("Tamanho do conjunto de teste:", X_test.shape)
```

Tamanho do conjunto de treinamento: (10000, 8)
Tamanho do conjunto de teste: (2960, 8)

Como os dados contêm variáveis sem considerar valores, foi utilizado o método *OneHotEncoder* para modificá-las antes de construir a árvore de decisão convertendo cada categoria em uma coluna binária.

A screenshot of a Jupyter Notebook cell. The code cell contains `1 df.head()`. Below it, a table displays the first five rows of the dataset. The table has 10 columns: `usual`, `proper`, `complete`, `1`, `convenient`, `convenient.1`, `nonprob`, `recommended`, and `recommend`. The data is as follows:

	usual	proper	complete	1	convenient	convenient.1	nonprob	recommended	recommend
0	usual	proper	complete	1	convenient	convenient	nonprob	priority	priority
1	usual	proper	complete	1	convenient	convenient	nonprob	not_recom	not_recom
2	usual	proper	complete	1	convenient	convenient	slightly_prob	recommended	recommend
3	usual	proper	complete	1	convenient	convenient	slightly_prob	priority	priority
4	usual	proper	complete	1	convenient	convenient	slightly_prob	not_recom	not_recom

Segue o código para transformar variáveis sem numeração

A screenshot of a Jupyter Notebook cell. The code cell contains the following Python code:

```
1 #import das funções para alteração em colunas binarias
2 from sklearn.preprocessing import OneHotEncoder
3 from sklearn.compose import ColumnTransformer
4
5 #aplicando o 'OneHotEncoding' as variáveis sem numeração
6 preprocessor = ColumnTransformer(transformers=[
7     ('cat', OneHotEncoder()), X.columns)
8 ])
9
10 X_train_enc = preprocessor.fit_transform(X_train)
11 X_test_enc = preprocessor.transform(X_test)
```

Agora seguindo o passo para construir a Árvore de Decisão com dois níveis.

Será usada a árvore de decisão com o critério *entropy* (ganho de informação) sendo limitada como os nós de decisão de primeiro e segundo níveis. O critério *entropy* define como a árvore de decisão mede a qualidade das divisões dos dados em cada nó. O *ganho de informação* calcula o quanto a incerteza do conjunto de dados é reduzida a cada divisão.

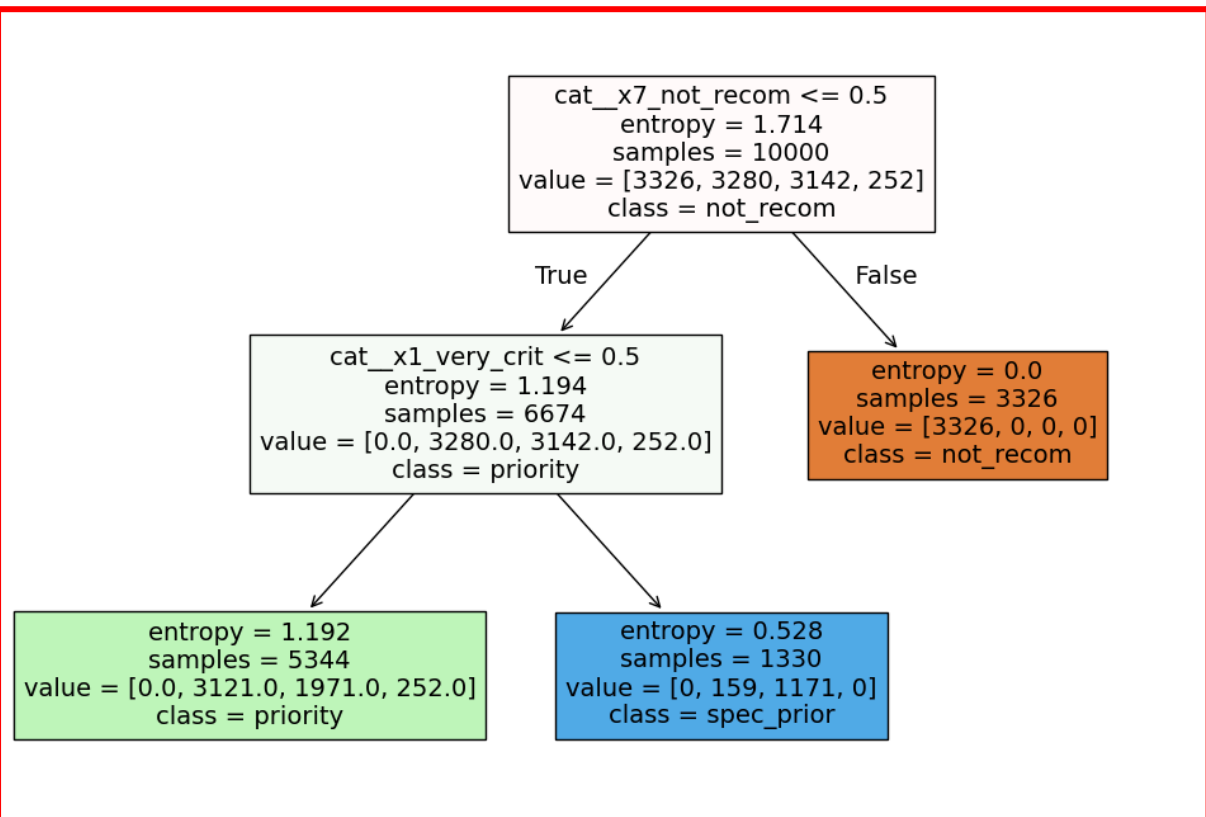
A screenshot of a Jupyter Notebook cell. The code cell contains the following Python code:

```
1 #import do classificador de árvore de decisão para criar e treinar o modelo
2 from sklearn.tree import DecisionTreeClassifier
3
4 #criando a árvore de decisão com critério de ganho de informação
5 clf = DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=42)
6 clf.fit(X_train_enc, y_train)
```

Below the code cell, the output of the last line of code is displayed in a box titled `DecisionTreeClassifier`. The output shows the instantiated object: `DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=42)`.

Feito isso, agora vamos visualizar a estrutura da árvore de decisão

```
1 #visualizando a estrutura da árvore
2 from sklearn import tree
3 import matplotlib.pyplot as plt
4
5 plt.figure(figsize=(12, 8))
6 tree.plot_tree(clf, feature_names=preprocessor.get_feature_names_out(), class_names=clf.classes_, filled=True)
7 plt.show()
```



Conforme solicitado na **letra b** desta questão, agora será avaliada a árvore com as amostras de teste. Como a árvore já foi treinada, será preciso calcular a acurácia usando o conjunto de teste.

```
1 #import para verificar a acurácia
2 from sklearn.metrics import accuracy_score
3
4 #predições para calcular a acurácia
5 y_pred = clf.predict(X_test_enc)
6 accuracy = accuracy_score(y_test, y_pred)
7 print("Acurácia da árvore de decisão no conjunto de teste:", accuracy)
```

Acurácia da árvore de decisão no conjunto de teste: 0.768918918918919

O próximo e último passo, conforme especificado na **letra c**, é *extrair* regras de decisão a partir da árvore construída. Para obtê-las, será necessário o método `export_text`, que permite descrever os caminhos na árvore. Este método pode transformar a estrutura de uma árvore de decisão treinada em um texto legível, por exemplo.

```
1 #Importando a função para exibir a árvore de decisão como texto, facilitando a visualização das regras de decisão
2 from sklearn.tree import export_text
3
4 #extraíndo e exibindo as regras de decisão
5 rules = export_text(clf, feature_names=list(preprocessor.get_feature_names_out()))
6 print("Regras de decisão da árvore:\n", rules)
```

```
Regras de decisão da árvore:
|--- cat_x7_not_recom <= 0.50
|   |--- cat_x1_very_crit <= 0.50
|   |   |--- class: priority
|   |   |--- cat_x1_very_crit > 0.50
|   |   |--- class: spec_prior
|--- cat_x7_not_recom > 0.50
|   |--- class: not_recom
```

Questão 04

Explique o dilema entre bias e variância e o seu relacionamento com underfitting e overfitting.

Pode-se levar em consideração que não há equilíbrio referente aos algoritmos no que tange ao Bias e a variância.

Bias (ou revés), em Machine Learning, é um modelo que não extrai todas as características da base de dados. Favorece algumas e outras não. Ou seja, não é um modelo preciso, correto, faltam informações necessárias para o algoritmo. Já ao contrário do Bias, a variância se refere a quanto cada valor da base de dados se distancia do valor médio. Ela pode ser dividida por variância alta e baixa onde a alta pode distribuir muitos dados, muitas informações, até mesmo desnecessárias para o usuário. Já a baixa sugere uma concentração maior de pontos em um mesmo intervalo.

O Underfitting, quando fazemos as variáveis de treino, ele não consegue entender e filtrar bem os dados ao relacionar variáveis X e Y por exemplo. Essa situação se corrobora se o modelo tiver poucas informações para pegar relações complexas de dados. Já o overfitting, em contrapartida, está ligado a uma situação em que o modelo de Machine Learning se adequou com tantas informações ao padrão de dados de treinamento que não consegue generalizar os dados para teste. Por um lado, o modelo de dados para treino é bom, mas para teste é ruim.

Questão 05

Comente sobre a veracidade das afirmações:

a. “Quanto mais variáveis de entrada forem usadas em um modelo de aprendizado de máquina, melhor será a qualidade do modelo”.

Na verdade não é bem assim. Conforme dito na questão anterior sobre variância onde há informação demais, até mesmo generalizada, com riqueza de detalhes pode trazer informações desnecessárias para o código. Podemos até pegar um “gancho” em comentar sobre esta questão se referindo ao exercício 1 deste trabalho onde foram retiradas informações de colunas ‘não relevantes para análise da variável ‘ViolentCrimesPerPop’, muitas colunas até mesmo sem informações preenchidas. Quando o modelo fica com muita informação para considerar, ele pode acabar “decorando” detalhes específicos dos dados de treino em vez de aprender de verdade, que é o overfitting.

b. “Independente da qualidade, quanto mais amostras forem obtidas para uma base de dados, maior a tendência de se obter modelos mais adequados”.

Até determinado ponto a afirmação faz sentido, porém não é tão garantido assim. Por um lado, se ter mais dados pode ajudar a ter mais informações, pode se aprender melhor os padrões gerais dos dados e pode evitar problemas quando se treina com pouca informação (overfitting). Por outro lado, temos que nos preocupar com a qualidade das amostras que é importante também. Se ‘rodar’ no modelo dados com erros, falta de informações (Nan, None ?) o algoritmo pode aprender coisas que não ajudam. Portanto mais amostras podem ajudar, desde que contenham informações concretas, boas e representam o modelo que vai encontrar. Percebe-se a diferença entre quantidade e qualidade.

c. “Às vezes com simples manipulações na base de dados (limpeza, conversão de valores, etc.) pode-se conseguir melhoras significativas nos resultados, sem fazer nenhuma alteração na técnica de aprendizado de máquina usada”.

Essa afirmação é condizente com a realidade. Antes de elaborarmos os códigos, fazer os trabalhos para aprendizagem de máquina, precisamos nos atentar a pequenas melhorias nos dados para trazer grandes resultados. Muitas das vezes nos preocupamos mais em fazer trabalhos muito elaborados, com muitos detalhes, muita riqueza de informação e esquecemos da parte da ‘limpeza’, etc. Essa limpeza ajuda o modelo a entender melhor as informações. Na base de dados por exemplo podemos ter uma coluna com informações faltantes serem preenchidas com a média, ou gênero que repassa para ‘0 ou 1’.

Questão 06

Em uma empresa é adotado um método de Aprendizado de Máquina para detectar defeito de fabricação de peças mecânicas, sendo que raramente acontece este tipo de problema na fábrica. Um funcionário anuncia empolgado que o sistema alcançou uma acurácia de 99%, porém seu gerente não achou o resultado tão relevante. Responda:

a. Por que o gerente não ficou empolgado com o resultado achado?

O gerente provavelmente percebeu que mesmo com a alta acurácia de 99% ela não significa necessariamente que o modelo está realmente identificando defeitos de forma eficaz. A acurácia faz o cálculo de todos os acertos (VP e VN) divididos por todos os acertos mais os erros (FP e FN). Entretanto, quando há uma classe desbalanceada, a acurácia não

é uma boa métrica a ser usada, pois o funcionário fazendo cálculo pela equação, os valores de classificados VN podem mascarar a classificações baixas de VP. Com isso pode acabar transmitindo uma sensação errada de que o modelo está fazendo a classificação correta e não está.

$$Acurácia = \frac{VP + VN}{VP + VN + FP + FN}$$

Equação 1: Acurácia

Em situações assim, o modelo pode “acertar” na maioria das vezes ao prever que não há defeito, pois isso já cobriria a maioria dos casos e resultaria em uma alta acurácia, mas sem realmente distinguir entre peças defeituosas e peças em bom estado.

b. O que o funcionário poderia fazer para confirmar se o método empregado é adequado para o problema?

Ele poderia verificar métricas como **precisão (precision)**, **revocação (recall)** que são métricas adequadas para resolver problemas desbalanceados. O recall, por exemplo, também conhecido como taxa de verdadeiros positivos, mede a capacidade do modelo em encontrar todos os exemplos positivos. Ela calcula a proporção de instâncias positivas previstas corretamente em relação ao total de instâncias positivas reais. Já a precisão métrica que se concentra na precisão das previsões positivas. Ela calcula a proporção de instâncias positivas previstas corretamente em relação ao total de instâncias previstas como positivas.

Uma outra opção seria ele utilizar a matriz de confusão que é uma tabela que indica os erros e acertos do seu modelo, comparando com o resultado esperado. Um exemplo prático seria um algoritmo que classifica imagens de animais como gatos e não gatos, gerando ao final a matriz de confusão abaixo:

		Classe esperada	
		Gato	Não é gato
Classe prevista	Gato	25 Verdadeiro Positivo	10 Falso Positivo
	Não é gato	25 Falso Negativo	40 Verdadeiro Negativo

Questão 07

Como pode ser usada uma árvore (de regressão ou de decisão) para avaliar uma amostra quando ela possui uma ou mais variáveis faltantes?

Citando alguns modelos como exemplo a seguir que podem ser usados nesta questão:

- CHAID (CHi-squared Automatic Interaction Detection) verifica, por exemplo, se a variável for categórica, cria um filho apenas para possibilidade dela estar faltando. Se ela for numérica, são criados 10 filhos (um para cada intervalo), e um 11º para possibilidade dela estar faltando.
- CART (*Classification and Regression Trees*). Utiliza-se *surrogate splits* (preenche o dado faltante utilizando a média dos dados presentes).
- C4.5: Apresenta tratamento de dados faltantes e *pruning*. O critério de divisão em cada nó utiliza uma métrica chamada *gain ratio* que é baseada em entropia.
- CRUISE (*Classification Rule with Unbiased Interaction Selection and Estimation*) utiliza a média/moda para imputar os valores faltantes.