Introduction to Machine Learning — 2022/2023

Projeto final

André Guerra

Lucas Barrigó

Janeiro/2023

Conteúdo

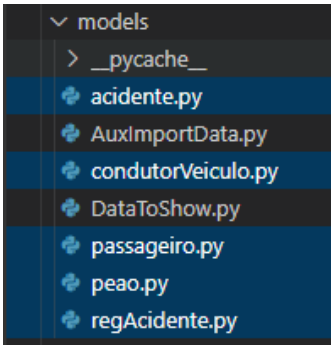
Introdução

Neste relatório irá ser explicado resumidamente as classes criadas, funcionalidade dos métodos criados e por fim a execução e discussão dos resultados dos exercícios.

Este trabalho tem como objetivo analisar os dados sobre acidentes rodoviários em Portugal, desde o ano 2010 a 2019. Os dados foram fornecidos em formato Excel, 1 por cada ano, dos quais foram retirados dados de forma aleatória (2000 acidentes por cada ano) guardados em formato JSON que depois foram filtrados e normalizados para formato CSV, formato utilizado pelos algoritmos dos exercícios.

Classes e métodos

Começando pela extração de dados, foram criados objetos para cada folha dentro dos exeis, que contém os mesmos atributos presentes nas mesmas:



O objeto “RegAcidente”, presente na pasta dos models, representa um acidente que contem o id, condutor do veiculo (condutoresVeiculos), passageiros e peões, caso o acidente tenha esses atributos nas diferentes folhas do respetivo Excel:

```
4   class RegAcidente(object):
5       def __init__(self, id, condutoresVeiculos, passageiros, acidentes, peoes):
6           self.id = id
7           self.condutoresVeiculos = condutoresVeiculos
8           self.passageiros = passageiros
9           self.acidentes = acidentes
10          self.peoes = peoes
```

O modelo “AuxImportData”, é um auxiliar para a extração dos dados, que irá ser explicada mais a frente:

```
1   class AuxImportData(object):
2       def __init__(self, id, rowIndex):
3           self.id = id
4           self.rowIndex = rowIndex
```

O modelo “DataToShow”, é um auxiliar para o exercício 3:

```
1   class DataToShow(object):
2       def __init__(self):
3           self.tipoAcidente = ""
4           self.countEstacoes = [0] * 4
5           self.verao = 0
6           self.outono = 0
7           self.inverno = 0
8           self.primavera = 0
```

Exercício 1

O exercício 1, presente no ficheiro exer1.py, consiste numa chamada do método que exporta aleatoriamente dados de todos os ficheiros para um formato de JSON, e mostra o tempo demorado da tarefa:

```
1  from Funcs import ImportData
2  import time
3
4  if __name__ == '__main__':
5
6      st = time.time()
7      ImportData()
8      et = time.time()
9      print(str(et - st))
```

A exportação dos dados utiliza multiprocessamento para efetuar a extração mais rapidamente, tendo em conta que temos 10 exeis para exportar, são criados 10 processos que correm em núcleos separados do CPU, caso o CPU tenho menos de 10 núcleos, o próximo Excel apenas começa a exportar quando um núcleo fica disponível. A nossa extração contém 2000 acidentes de cada Excel, um total de 20.000 acidentes, escolhidos aleatoriamente, e tendo em conta o processador usado (i7-12700H), demora cerca de 40 minutos a extração.

O método que inicia a exportação de dados "ImportData" está no ficheiro "Funcs.py", que também contem todos os outros métodos necessários para a exportação. Neste mesmo ficheiro começamos por declarar 2 variáveis que contem o número de acidentes e os anos a serem exportados:

```
19  maxRows = 3000
20  yearsToImport=["2010","2011","2012","2013","2014","2015","2016","2017","2018","2019"]
```

O método "ImportData" começa por percorrer o array com os anos a serem exportados e cria o processo que realiza a exportação, os processos partilham uma variável global "regAcidentes", que contem uma lista de objetos ("RegAcidente") extraídos. Por cada processo é chamado o método "ReadExcel" que lê e extrai os dados:

```
33  def ImportData():
34      regAcidentes = []
35      data_inputs = []
36      for year in yearsToImport:
37          data_inputs.append(year)
38      pool = Pool()
39      regAcidentes = pool.map(ReadExcel, data_inputs)
40      regAcidentes = [ent for sublist in regAcidentes for ent in sublist]
41      CreateJson(regAcidentes)
42      print("READER ENDED.")
```

O método "ReadExcel" começa por ler o ficheiro do ano que recebe, e as suas folhas. De seguida cria arrays com o objeto "AuxImportData" que contem o id e o índex da linha na folha, este array com o id e a posição da linha permite que a extração seja feita mais rapidamente (percorrer uma array de objetos pequenos ao invés das folhas no Excel):

```
44  def ReadExcel(year):
45      print("Excel " + str(year) + " started...")
46      regAcidentes = []
47      book = openpyxl.load_workbook(os.getcwd()  + "\excel\ISCTE_" + str(year) + "\ISCTE_" + str(year) + ".xlsx")
48      sheet0 = book.worksheets[0]
49      sheet1 = book.worksheets[1]
50      sheet2 = book.worksheets[2]
51      sheet3 = book.worksheets[3]
52      auxI = 0
```

```
    print("Building matrix's " + str(year) + " ...")
    arraySheet1 = []
    arraySheet2 = []
    arraySheet3 = []
    arraySheet4 = []
    auxRowIndex1 = 1
    auxRowIndex2 = 1
    auxRowIndex3 = 1
    auxRowIndex4 = 1
    for row in sheet0:
        aux = AuxImportData(
        row[0].value,
        auxRowIndex1)
        arraySheet1.append(aux)
        auxRowIndex1 += 1
    for row in sheet1:
        aux = AuxImportData(
        row[0].value,
        auxRowIndex2)
        arraySheet2.append(aux)
        auxRowIndex2 += 1
    for row in sheet2:
        aux = AuxImportData(
        row[0].value,
        auxRowIndex3)
        arraySheet3.append(aux)
        auxRowIndex3 += 1
    for row in sheet3:
        aux = AuxImportData(
        row[0].value,
        auxRowIndex4)
        arraySheet4.append(aux)
        auxRowIndex4 += 1
    arraySheet1.pop(0)
    arraySheet2.pop(0)
    arraySheet3.pop(0)
    arraySheet4.pop(0)
    print("Building matrix's finished " + str(year) + ".")
    print("len matrix " + str(year) + " to build => " + str(len(arraySheet3)))
```

Após os arrays estarem construídos, percorremos o array 3 (ou a folha do Excel 3 “30 Dias _ Acidentes”), até se atingir o número de dados pretendidos ou o array 3 chegar ao fim.

Esta extração pega numa posição aleatória do array, que escolhe o id do acidente a ser exportado, esse id é usado depois nos outros arrays, que depois de filtrados permite saber a posição do dado na folha de excel, essa exportação depois é feita nos métodos “GetCondutoresVeiculo”, “GetPassageiros”, “GetAcidentes” e “GetPeoes”, que cria o objeto “RegAcidente”. A cada acidente extraído, é removido dos arrays o acidente pelo id escolhido, impedindo a repetição do mesmo:

```
    while auxI < maxRows or len(arraySheet3) == 0:
        if len(arraySheet3) == 0:
            break
        print(str(year) + ": " + str(auxI/maxRows*100))
        #print(str(year) + " len matrix remaining => " + str(len(matrixSheet1)))
        rowIndexx = randrange(0, len(arraySheet3))
        row = sheet0[arraySheet3[rowIndexx].rowIndex]

        #print(str(year) + " adding id => " + str(row[0].value))
        reg = RegAcidente(
            row[0].value,
            GetCondutoresVeiculo(row[0].value, arraySheet1, sheet0),
            GetPassageiros(row[0].value, arraySheet2, sheet1),
            GetAcidentes(row[0].value, arraySheet3, sheet2),
            GetPeoes(row[0].value, arraySheet4, sheet3)
        )
        regAcidentes.append(reg)
        auxI += 1

        arraySheet1 = list(filter(lambda item: item.id != row[0].value, arraySheet1))
        arraySheet2 = list(filter(lambda item: item.id != row[0].value, arraySheet2))
        arraySheet3 = list(filter(lambda item: item.id != row[0].value, arraySheet3))
        arraySheet4 = list(filter(lambda item: item.id != row[0].value, arraySheet4))

    print("Excel " + str(year) + " finished.")
    return regAcidentes
```

Os 4 métodos que extraem o objeto de cada folha do Excel são todos semelhantes. Explicando o “GetPeoes”, este recebe o id do acidente, o array da folha de Excel que contem os id’s e o index da posição dado presente na folha peões “30 Dias_Peões”. O array depois de filtrado pelo id do acidente, é percorrido e extraída a informação da folha do Excel tendo em conta a posição do mesmo dado:

```
def GetPeoes(idAcidente, array, sheet):
    #print("GetPeoes started...")
    cvs = []
    arrayy = list(filter(lambda item: item.id == idAcidente, array))
    for m in arrayy:
        r = sheet[m.rowIndex]
        p = Peao(
            r[0].value,
            r[1].value,
            r[2].value,
            r[3].value,
            r[4].value,
            r[5].value,
            r[7].value,
            r[8].value,
            r[9].value,
            r[10].value,
            r[11].value,
            GetIdade(sheet[1], r, 30))
        cvs.append(p)
    return cvs
    #print("GetPeoes ended.")
```

O método “GetIdade” retorna a idade de forma mais prática de se trabalhar mais a frente nos exercícios, retornando apenas a idade a que pertence o dado, por exemplo “40-44”:

```
251    def GetIdade(header, row, numCols):
252        for c in range(numCols):
253            if header[c].value.__contains__("Gr.Etario"):
254                if row[c].value is not None and row[c].value > 0:
255                    return header[c].value.split("(")[1].split(")")[0]
```

Os métodos “ReadJson” e “CreateJson”, permitem ler e escrever em formato JSON os dados extraídos, estes dados estão presentes da pasta “data”:

```
22    def ReadJson():
23        with open( os.getcwd()  + "\src\data\Data3000.json", 'r') as file:
24            json_str = file.read()
25            return jsonpickle.decode(json_str)
26
27    def CreateJson(list):
28        with open(os.getcwd()  + "\src\data\Data3000.json", "w") as file:
29            jsonpickle.set_encoder_options('json', sort_keys=True, indent=4)
30            frozen = jsonpickle.encode(list, file)
31            file.write(frozen)
```

Exercício 2

Presente no ficheiro exer2.py, fazemos um tratamento dos dados exportados substituindo tudo por números e guardando tudo num ficheiro CSV, de forma a poder trabalhar com as bibliotecas:

```
def ConvertSexoToNum(sexoStr):
    if sexoStr == "Masculino":
        return 0
    elif sexoStr == "Feminino":
        return 1
    else:
        return "UNKOW"
```

Este tratamento torna-se bastante eficiente na escolha dos dados, pois caso não seja possível converter para inteiro este é ignorado. A escolha dos dados a serem usados também é bastante simples e permite escolher apenas os que iram ser usados ou relevantes para o exercício, criando um ficheiro CSV com as colunas pretendidas:

```
261    regAcidentes = ReadJson()
262    csvStr = "aNumFeridosgravesa30dias,aNumFeridosligeirosa30dia,aNumMortosa30dias,aFactoresAtmosféricos,aNatureza,aCaracterísticasTec
263    for dado in regAcidentes:
264        try:
265            x =  str(int(dado.acidentes[0].NumFeridosgravesa30dias)) + ","  + str(int(dado.acidentes[0].NumFeridosligeirosa30dias)) +
266            csvStr += x
267        except:
268            a=0
```

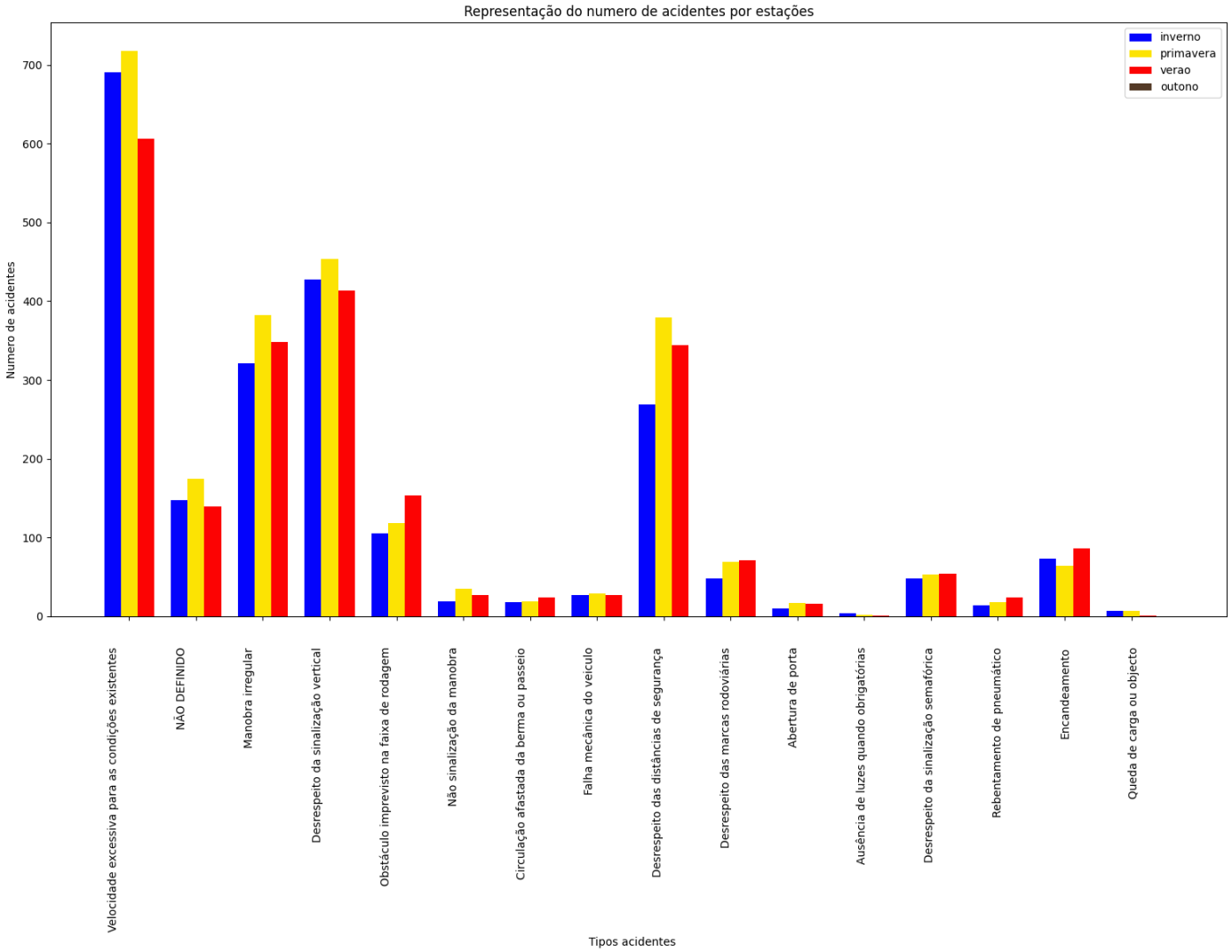
Exercício 3

No exercício 3, não é utilizado o CSV tratado anteriormente, mas sim o JSON criado na extração. Neste exercício, para cada tipo de acidente, verificamos a estação do ano em que ocorreu:

```
dataToGraphMain = []
for regAcidente in regAcidentes:
    for condutoresVeiculo in regAcidente.condutoresVeiculos:
        dataToGraphHasType = False

        for typeDataAux2 in dataToGraphMain:
            if typeDataAux2.tipoAcidente == condutoresVeiculo.InfCompaAcçõeseManobras:
                dataToGraphHasType = True

        if dataToGraphHasType == False:
            aux = DataToShow()
            aux.tipoAcidente = condutoresVeiculo.InfCompaAcçõeseManobras
            if get_season(datetime.strptime(condutoresVeiculo.Datahora, '%Y:%m:%d %H:%M:%S')) == "winter":
                aux.countEstacoes[0] = aux.countEstacoes[0] + 1
                aux.inverno += 1
            elif get_season(datetime.strptime(condutoresVeiculo.Datahora, '%Y:%m:%d %H:%M:%S')) == "spring":
                aux.countEstacoes[1] = aux.countEstacoes[1] + 1
                aux.primavera += 1
            elif get_season(datetime.strptime(condutoresVeiculo.Datahora, '%Y:%m:%d %H:%M:%S')) == "summer":
                aux.countEstacoes[2] = aux.countEstacoes[2] + 1
                aux.verao += 1
            elif get_season(datetime.strptime(condutoresVeiculo.Datahora, '%Y:%m:%d %H:%M:%S')) == "autumn":
                aux.countEstacoes[3] = aux.countEstacoes[3] + 1
                aux.outono += 1
            dataToGraphMain.append(aux)
```

E depois alteramos o formato dos dados contabilizados para o input do gráfico, o valor do tipo de acidente “não identificada” foi filtrado, pois tem um valor muito alto e irrelevante no exercício:

```
for dado in dataToGraphMain:
    if dado.tipoAcidente == "Não identificada":
        continue
    tiposAcidente.append(dado.tipoAcidente)

    inverno.append(dado.countEstacoes[0])
    primavera.append(dado.countEstacoes[1])
    verao.append(dado.countEstacoes[2])
    outono.append(dado.countEstacoes[3])

r1 = np.arange(len(tiposAcidente))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
r4 = [x + barWidth for x in r3]

plt.bar(r1, inverno, color="#0303fc", width=barWidth, label="inverno")
plt.bar(r2, primavera, color="#fce303", width=barWidth, label="primavera")
plt.bar(r3, verao, color="#fc0303", width=barWidth, label="verao")
plt.bar(r4, outono, color="#543a26", width=barWidth, label="outono")

plt.xlabel('Tipos acidentes')
plt.xticks([r + barWidth for r in range(len(tiposAcidente))], tiposAcidente,rotation='vertical')
plt.ylabel('Numero de acidentes')
plt.title('Representação do numero de acidentes por estações')
plt.legend()
plt.show()
```



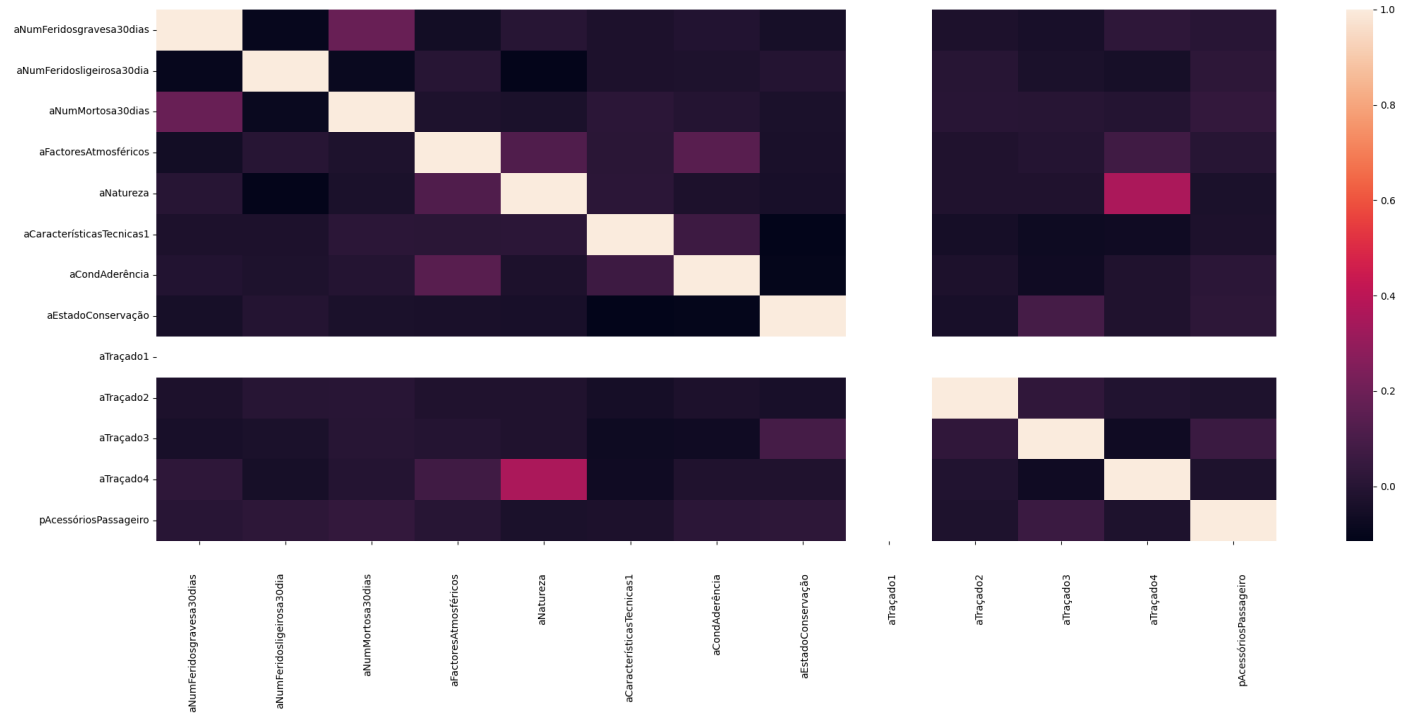
Relativamente aos resultados obtidos no gráfico, podemos verificar que não existem acidentes no Outono, possivelmente devido á biblioteca usada na extração de dados ter uma limitação no tamanho da leitura da folha de Excel.

Exercício 4

Tendo em conta o tratamento já feito no exercício 2, que filtra e trata os dados, este exercício 4 encontra-se incluído no anterior.

Exercício 5

Neste exercício, utilizando o CSV criado no exercício 2, ao correrms o algoritmo obtemos o seguinte gráfico:

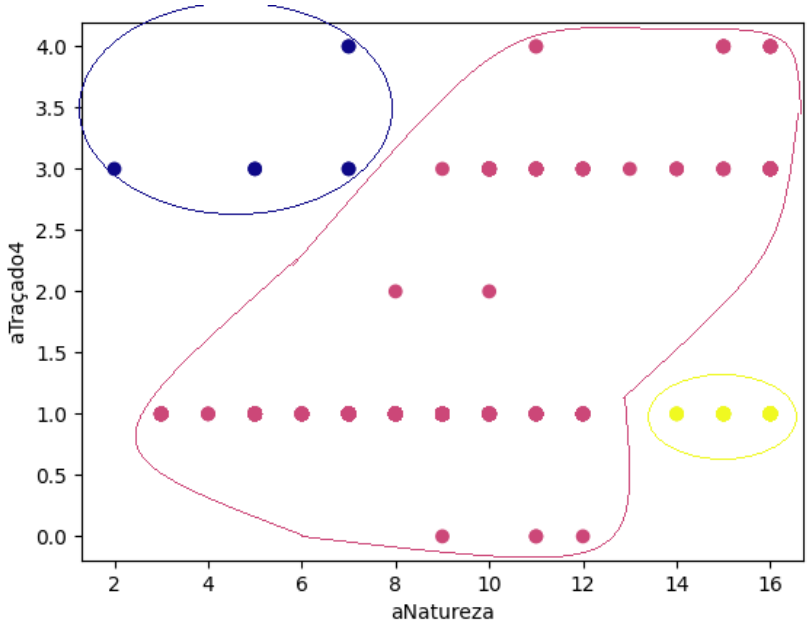


Podemos verificar que existe uma grande relação entre o número de mortos e feridos graves, pois esses feridos graves têm uma maior probabilidade de falecer que os feridos leves. Existe também uma alta relação entre as condições de aderência e os fatores atmosféricos, que podem ser explicados pela redução de aderência na estrada com piores condições atmosféricas. A relação entre o traçado 4 e a natureza, deve-se ao facto da maioria dos acidentes acontecer na via, fazendo com que exista uma alta correlação entre os dois fatores. A linha a branco no gráfico trata-se de um erro na biblioteca que produz o gráfico, visto que nos testes a alteração da ordem dos dados continua a mostrar essas linhas sempre nas mesmas posições, dando assim a entender que o bug não é devido aos parâmetros.

Exercício 6

Neste exercício, utilizando o csv criado anteriormente, utilizamos uma biblioteca para ambos os algoritmos. O DBScan que com base no número de vizinhos cria um conjunto de clusters, avaliamos as colunas natureza e traçado do acidente, tendo em conta os dados que temos usamos os seguintes parâmetros, o número de vizinhos 5 por cada ponto, o epson 1.1 visto visto que os nossos dados se distanciam entre eles no mínimo 1 e mínimo de dados para criar um kluster 8.

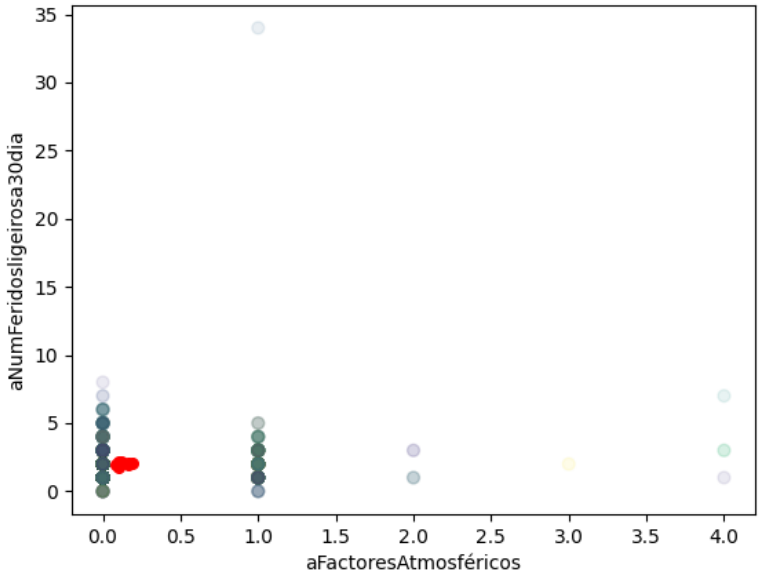
```
coluna1 = "aNatureza"
coluna2 = "aTraçado4"
df = pd.read_csv(os.getcwd()  + "\src\data\Data2000Exer6.csv")

### DBscan
x = df.loc[:, [coluna1, coluna2]].values
neighb = NearestNeighbors(n_neighbors=5) # creating an object of the NearestNeighbors class
nbrs=neighb.fit(x) # fitting the data to the object
distances,indices=nbrs.kneighbors(x) # finding the nearest neighbours
# Sort and plot the distances results
distances = np.sort(distances, axis = 0) # sorting the distances
distances = distances[:, 1] # taking the second column of the sorted distances
# cluster the data into five clusters
dbscan = DBSCAN(eps = 1.1, min_samples = 8).fit(x) # fitting the model
labels = dbscan.labels_ # getting the labels
# Plot the clusters
plt.scatter(x[:, 0], x[:,1], c = labels, cmap= "plasma") # plotting the clusters
plt.xlabel(coluna1) # X-axis label
plt.ylabel(coluna2) # Y-axis label
plt.show() # showing the plot
```



O algoritmo cria 3 kluster, a azul obtemos os acidentes “Na berma” e “No passeio”, nos quais a natureza é uma colisão ou atropelamento de peões, a vermelho obtemos todo o tipo de acidentes em todos os traçados, com exceção do tipo de acidente atropelamentos, a amarelo representa os acidentes com natureza de despiste “Em plena via”.

No algoritmo K-means, avaliamos os fatores atmosféricos e o número de feridos ligeiros, atribuímos o valor de 7 ao número de clusters, visto que são os diferentes tipos de fatores atmosféricos existentes.

```
### K Means
kmeans = KMeans(n_clusters=7).fit(df)
centroids = kmeans.cluster_centers_
print(centroids)
#select the 2 columns to work with
plt.scatter(df['aFactoresAtmosféricos'], df['aNumFeridosligeirosa30dia'], c= kmeans.labels_.astype(float), s=40, alpha=0.1)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=30)
plt.xlabel("aFactoresAtmosféricos") # X-axis label
plt.ylabel("aNumFeridosligeirosa30dia") # Y-axis label
plt.show()
```



Tendo em conta que o maior número de dados existente ocorre quando está chuva e bom tempo (0 e 1), os centroides dirigem-se para esses pontos.

Exercicio 7

No execicio 7, usamos um algoritmo chamado Adaboost, que calcula a percentagem de precisão, tendo em conta os dados escolhidos para chegar ao resultado de outro. Neste caso usamos as seguintes colunas: aNumFeridosgravesa30dias, aNumFeridosligeirosa30dia, aNumMortosa30dias, aFactoresAtmosféricos, aNatureza, aCaracterísticasTecnicas1, aCondAderência, aEstadoConservação, aTraçado1, aTraçado2, aTraçado3, aTraçado4, pAcessóriosPassageiro, cvSexo, cvIdade. Neste algoritmo podemos depois escolher qual a coluna que queremos prever.

```
### Adaboost
df = pd.read_csv(os.getcwd()  + "\src\data\Data2000Exer6.csv")
#split dataset in features and target variable
feature_cols = ['aNumFeridosgravesa30dias','aNumFeridosligeirosa30dia','aNumMortosa30dias','aFactoresAtmosf
X = df[feature_cols] # Features
y = df.cvSexo # Target variables => 'aNumFeridosgravesa30dias,aNumFeridosligeirosa30dia','aNumMortosa30dias
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and 30% test
# Create adaboost classifer object
abc = AdaBoostClassifier(n_estimators=1, learning_rate=2)
# Train Adaboost Classifer
model = abc.fit(X_train, y_train)
#Predict the response for test dataset
y_pred = model.predict(X_test)
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Escolhendo a coluna número de mortos, obtemos em média uma precisão de 99%, e conseguimos prever o sexo do condutor com uma precisão de 100%.

Já a natureza do acidente é mais difícil de prever, resultando numa precisão de 45%, como também a idade do condutor de 19% de acerto.

Exercício 8

No exercício 8, testamos com o cross validation e 2 data sets diferentes, a precisão de acerto.

```
15  df2 = pd.read_csv(os.getcwd()  + "\src\data\Data2000Exer6.csv")
16  df = pd.read_csv(os.getcwd()  + "\src\data\Data1102.csv")
17
18  X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, :-1], df.iloc[:,-1], test_size=0.3, random_state=1)
19  knn= KNeighborsClassifier(n_neighbors=2)
20  accuracies=cross_val_score(estimator=knn,X=X_train,y=y_train,cv=2)
21  print(accuracies)
22  print("average accuracy :",np.mean(accuracies))
23  knn.fit(X_train,y_train)
24  print("test accuracy :",knn.score(X_test,y_test))
25  print("train accuracy :",knn.score(X_train,y_train))
```

Usando o data set “Data1102” obtemos os seguintes resultados:

```
average accuracy : 0.8185320182961693
test accuracy : 0.8245614035087719
train accuracy : 0.8695652173913043
```

No data set “Data2000Exer6” obtemos uma precisão média mais alta, pois o data set “Data1102” tem no seu total 10 mil dados extraídos, comparado com os 20 mil do “Data2000Exer6”.

```
average accuracy : 0.1
test accuracy : 0.11944444444444445
train accuracy : 0.5345238095238095
```

Foi testado ainda outro data set “Data1101” que obteve resultados semelhantes ao “Data1102”:

```
average accuracy : 0.8382083389216057
test accuracy : 0.8222591362126246
train accuracy : 0.8688524590163934
```

Exercício 9

No exercício 9, implementamos 3 algoritmos, XGBoost, Multi Layer Perceptron (Neural Networks) e Decision Tree, para podermos averiguar a precisão média de cada um deles em 30 execuções. Também escolhemos várias colunas a serem prevista:

	Multi Layer Perceptron	Decision Tree	XGBoost
Natureza do acidente	87%	100%	Fail
Número de mortos	99%	99%	100%
Condições de aderência	88%	100%	100%
Traçado 4	99%	100%	100%
Número de feridos ligeiros	96%	100%	99%

Neste exercício podemos observar que os algoritmos Decision Tree e XGBoost são mais precisos que o Multi Layer Perceptron, e ainda que em termos não conseguir chegar ao resultado o XGBoost quando os dados de treino ou os de teste, têm um valor não presente nos 2 tipos de dados este falha.

Conclusão

Neste trabalho podemos concluir que a extração acabou por ser rápida, o que permitiu termos várias extrações para utilizarmos nos exercícios. O tratamento de dados é uma parte importante para que os exercícios corram de forma correta, o que pode influenciar os dados. Dos diferentes algoritmos implementados, o Decision Tree foi um dos mais precisos e eficazes.