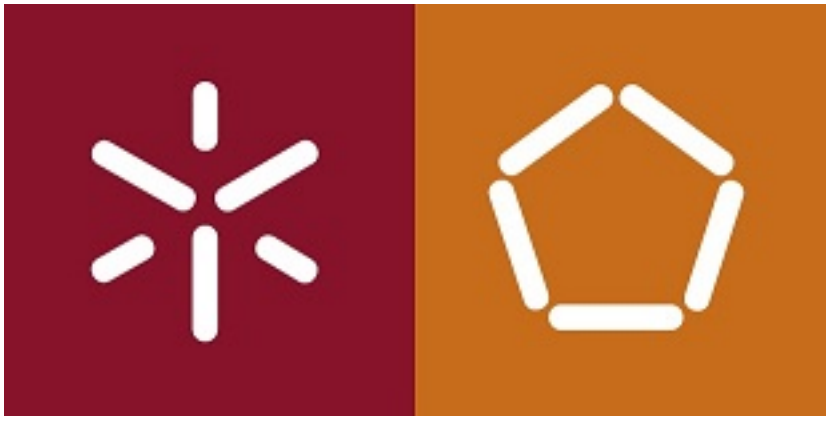

Universidade do Minho

Processador de registos de exames médicos desportivos (EMD)

Processamento de Linguagens

Group 04

[A93180] Diogo Rebelo - [A93313] Diogo Miguel - [A76603] Joel Araújo

Março, 2022

Python

Resumo

Serve o presente relatório como documento de apoio para o acompanhamento da resolução do enunciado relativo ao primeiro trabalho prático da cadeira de processamento de linguagens. O objetivo do mesmo é processar um ficheiro, utilizando como auxílio expressões regulares, e explorar a informação que este acarreta.

Índex

Índex de Figuras

Introdução

Face à quantidade de informação que nos é disponibilizada em qualquer website e/ou documento, seja qual for o seu tipo, o olho e a capacidade de atenção humanos não fazem jus, não conseguem acompanhar e filtrar a informação que realmente necessitam de maneira fácil, muito menos eficiente. Colmatando esta improficiência, surge-nos como ferramenta expressões regulares, um mecanismo flexível e sucinto com o qual se podem identificar palavras/padrões de caracteres com interesse para o utilizador.

Com este projeto, o objetivo é melhorar a nossa aptidão para a escrita deste tipo de expressões, utilizando a linguagem Python como ponte para esse efeito.

1 Descrição do problema

1.1 Enunciado

Dos enunciados que nos foram disponibilizados para escolha, aquele que escolhemos foi o referente a um processador de registos de exames médicos desportivos. O objetivo primário do mesmo é atuar sobre um dataset, previamente fornecido, no qual consta informação sobre indivíduos e os seus registos médicos respetivos, entre outros.

Após processar o mesmo, o trabalho foca-se numa apresentação, através de um website, de um conjunto de indicadores estatísticos, os quais apresentaremos de seguida. Destes, faz-se também acompanhar um conjunto de websites onde constam os dados dos registos que utilizamos para fazer os respetivos estudos.

1.2 Características

1.2.1 Tipo de ficheiros

Para desenvolver o nosso projeto, trabalhamos sobre um ficheiro de texto **CSV**, "emd.csv".

1.2.2 Indicadores estatísticos

a. Datas extremas dos registos no dataset;

b. Distribuição por género, em cada ano e no total;

c. Distribuição por modalidade, em cada ano e no total;

d. Distribuição por idade e género (utilizando 2 escalões: ¡ 35 anos e ¿= 35 anos);

e. Distribuição por morada;

f. Distribuição por estatuto de federado em cada ano;

g. Percentagem de aptos e não aptos, por ano.

2 Resolução de problema

2.1 Universo dos registos

Como ponto de partida, o primeiro foco foi em estudar e perceber o dataset providenciado, que tipo de informação ele continha e como era apresentada. Sendo um ficheiro csv, onde os registos estão delimitados por vírgulas, deparamo-nos com os seguintes campos:

1. id
2. index
3. dataEMD
4. nome/Primeiro
5. nome/último
6. idade
7. género
8. morada
9. modalidade
10. clube
11. email
12. federado
13. resultado

Sobre este conjunto de informação, e após atentar nos estudos que nos foram indicados para apresentar, decidimos que há campos com os quais nunca nos iríamos deparar, isto é, não seriam necessários para os nossos cálculos. Esses campos são o índex, visto que ele nem é único, pois itera de 0 a 99, voltando novamente ao 0, e o clube, já que se trata de um atributo meramente decorativo para os nossos propósitos, cujo valor não é de interesse.

Ainda sobre este tipo de escolhas, debatemo-nos sobre a necessidade de trabalhar com o id e o email. O id é um atributo importante, mas em nada influenciaria nos nossos cálculos, assumimos numa primeira discussão. Contudo, se nos depararmos com registos duplicados, isso influenciaria negativamente os dados produzidos, ao que concluímos que seria de valor continuar a trabalhar com o id. Por sua vez, o email também em nada influencia, mas tendo como objetivo publicar os dados utilizados para os cálculos, entendemos por bem apresentar também o email dos indivíduos, para declarar um certo realismo e coerência na informação processada, e não parecer que são informações que poderiam ter sido inventadas por qualquer pessoa.

Deste modo, apenas o índex e o clube foram excluídos do nosso tratamento.

2.2 Organização do projeto

Para facilitar o modo como desenvolvemos o projeto, assim para melhor controlar onde cada parte do mesmo se encontra, dividimo-lo em três módulos, que de seguida se identificam.

EMDsParser

Relativo a este módulo temos presente os ficheiros que tratam de definir uma classe emdRegister, que especificam o tokenizer que desenvolvemos e a respetiva leitura do ficheiro.

Queries

Nesta porção encontram-se ficheiros destinados a responder a cada uma das alíneas que nos foram propostas para resolução, onde exploram a estrutura de dados com os dados sobre os registos obtidos.

htmlGenerator

Por sua vez, este módulo acumula os ficheiros que estarão destinados a gerar os ficheiros html onde estarão apresentadas as nossas respostas

2.3 Tokenizer

Empregados de utilizar os módulos**re** e **ply** do Python, a nossa solução faz uso de um tokenizer para dividir cada linha do dataset em porções pequenas de informação com o formato pretendido, para cada campo do registo.

2.3.1 Tokens utilizados

Munidos de nomes esclarecedores, os tokens "id", "date", "gender" e email ficam encarregues de capturar esses mesmos campos de cada registo. Por sua vez, o token "word" trata de recolher os campos como os nomes, moradas, modalidade e clube, visto terem formatos idênticos, enquanto o boolean acarreta com os campos de federado e resultado do exame médico, visto representarem os valores "true" ou "false".

Figure 1: Tokens utilizados

2.3.2 Expressões regulares

- t_ID

Após estudar o ficheiro de texto com os registos, percebemos que os ids dos registos continham sempre um padrão onde o tamanho desse id era sempre de 24 caracteres e o alfabeto utilizado apenas continha letras minúsculas e números.

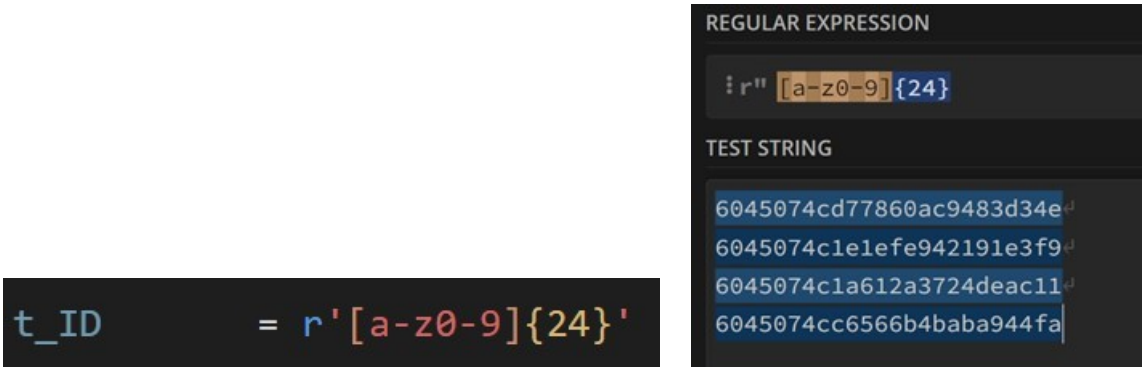

Figure 2: t_ID ER e exemplo de match

- t_DATE

O formato apresentado para o campo das datas corresponde a yyy-MM-dd.

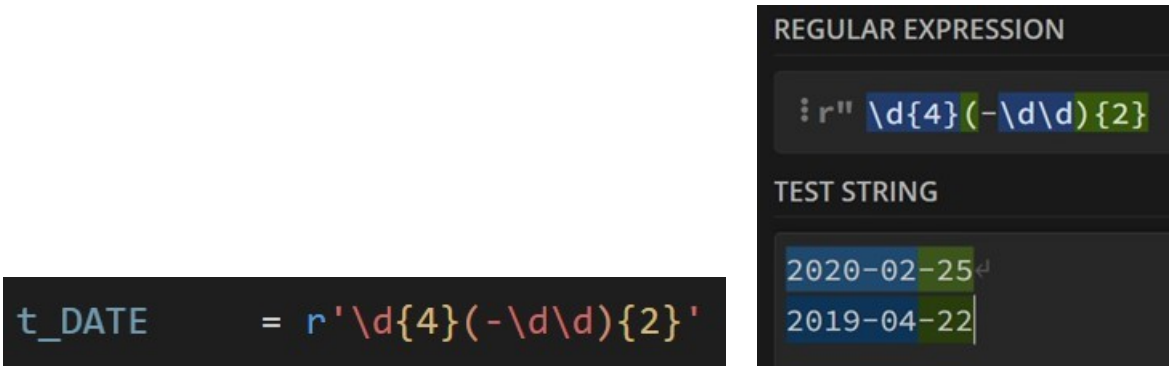

Figure 3: t_DATE ER e exemplo de match

- t_WORD

Para descrever o formato de palavras com acentos, como é o caso da língua portuguesa, poderíamos utilizar o atalho "\w" das expressões regulares, que consegue captar qualquer caractere de qualquer linguagem. Contudo, visto que as palavras que procuramos não contêm números nem "underscores", além de terem de começar por letra maiúscula, optamos por definir o nosso padrão com aquilo que não queríamos que ele contivesse. Por forma a captar siglas utilizadas para os nomes de algumas modalidades e/ou clubes, não limitamos o número de maiúsculas que pudessem aparecer.

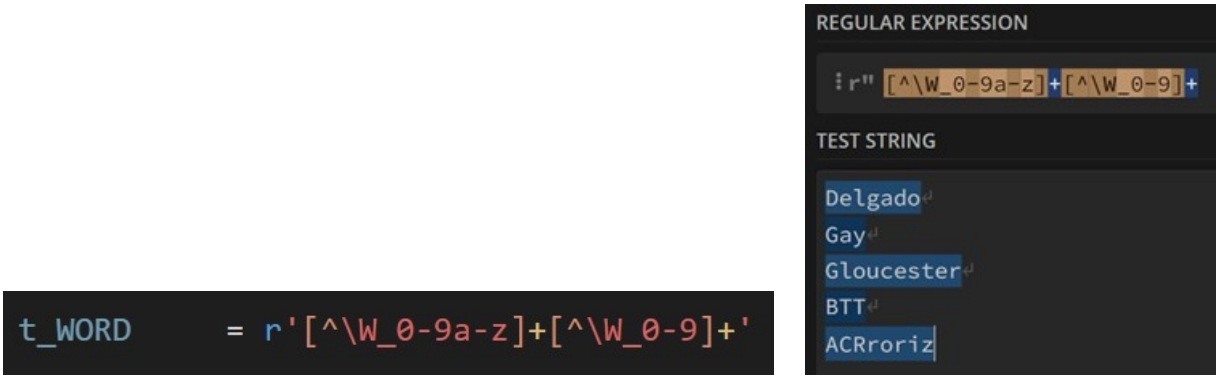

Figure 4: t_WORD ER e exemplo de match

- t_NUMBER

Por forma a não identificar números que estivessem contidos no campo do id, por exemplo, definimos o número com "word boundary", \b.

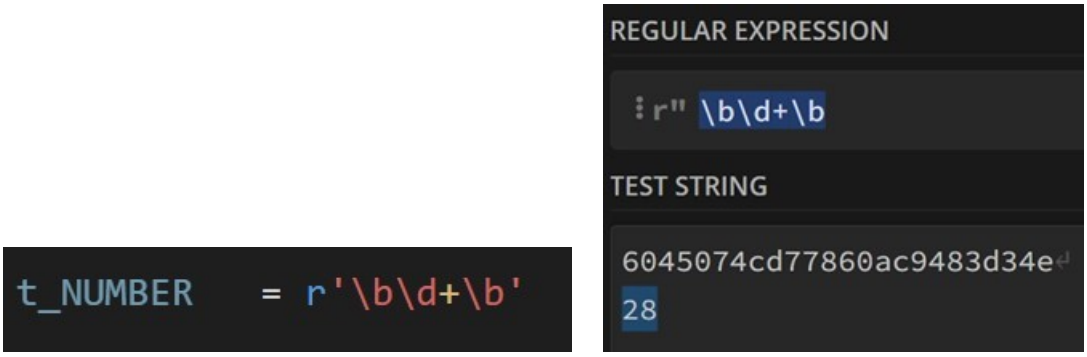

Figure 5: t_NUMBER ER e exemplo de match

- t_GENDER

O género é-nos apresentado no documento de texto com os caracteres maiúsculos 'M' e 'F'. Contudo, por forma a admitir que, por lapso, o género pudesse vir identificado com as respetivas letras minúsculas, decidimos abranger também esse caso, além de especificar que essas letras tinham de se encontrar isoladas de quaisquer outros caracteres.

Figure 6: t_GENDER ER utilizada

- t_BOOLEAN

Seguindo o mesmo conjunto de ideias do campo anterior, os campos de federado e de resultado médico podem ser identificados com os respetivos valores booleanos em minúsculo ou maiúsculo.

```
t_BOOLEAN  = r'\b(?i:true|false)\b'
```

Figure 7: t_BOOLEAN ER utilizada

- t_EMAIL

De modo a captar os emails de forma rigorosa é necessário o primeiro caracter estar contido no atalho "\w", sendo que após isso pode ter qualquer caracter desse atalho incluindo "." e "-", desde que não seja o caracter anterior e posterior ao "@". A próxima sequência de caracteres necessita de estar incluida no conjunto "\w" ou "-" até aparecer um ".", onde a partir daí necessita de ter 2 ou mais caracteres presentes no conjunto anterior (um *Top-Level Domain* de um email precisa de, no mínimo, 2 caracteres).

```
t_EMAIL    = r'\w(\w[\.\-]?)*\w@\w[\w\-]*(\.[\w\-]{2,})*'
```

Figure 8: t_EMAIL ER utilizada

- t_ignore

Por fim, atentamos que, numa primeira abordagem, consideramos as vírgulas delimitadoras dos campos de cada registo como um token a identificar. Contudo, visto que não as usaríamos, optamos por adicioná-la aos caracteres a serem ignorados pelo tokenizer, já que também não fazem parte de qualquer um dos tokens já definidos.

```
t_ignore = ', \t\n'
```

Figure 9: t_ignore ER utilizada

2.4 Estrutura de dados utilizada

A partir do momento que o objetivo seria explorar e produzir resultados sobre os dados que retiramos do ficheiro de texto utilizado, há necessidade de, de alguma forma, os organizar por forma a agilizar o respetivo processo. Numa primeira abordagem, para cada registo encontrado, criamos um objeto cuja classe se identifica como emdRecord.

```
class emdRecord:

    __slots__ = ["_id", "_date", "_name",
                 "_surname", "_age", "_gender",
                 "_address", "_modality", "_email",
                 "_federated", "_medicalResult"]
```

Figure 10: class emdRecord

Posteriormente, por forma a ajudar-nos nas respetivas alíneas, encontramos uma maneira de organizar estes registos que nos seria vantajosa nas posteriores pesquisas. De maneira geral, para cada ano onde haja, pelo menos, um registo, teríamos um dicionário que ligaria sempre quatro chaves, “femLT35” (feminine lower than 35), “femGET35” (feminine greater or equal to 35, “mascLT35”, “mascGET35”, cujos propósitos seriam filtrar imediatamente os registos por género e também por idade. Deste modo, para cada uma destas chaves está atribuída uma lista com todos os registos que nela se enquadram.

Exemplo:

{2022: {“femLT35”: [emd01, emd02], “femGET35”: [ ], “mascLT35”: [emd03], “mascGET35”: [ ] },
2019: {“femLT35”: [emd04], “femGET35”: [ ], “mascLT35”: [emd05], “mascGET35”: [emd06] } }

2.5 Leitura do ficheiro

De maneira a interpretar corretamente o conteúdo que o ficheiro de texto disponibilizado continha (caracteres com acento, por exemplo), o método de abertura do mesmo especifica a flag de encoding “utf-8”.

```
fHandler = open(file, "rt", encoding="utf-8")
```

Figure 11: especificações de leitura do ficheiro

Utilizando, por sua vez, os módulos anteriormente explicitados neste documento, o processo de leitura baseia-se em preparar a estrutura de dados criada e, percorrendo cada linha do texto, utilizar o tokenizer para preparar os chunks de informação selecionada para os fornecer a um novo objeto emdRegister. Deste modo, acedendo à data do registo escolhe-se qual é a primeira chave do dicionário a que pertence, e consultando ambas a data e a idade, encaminha-se novamente o registo para a chave do novo dicionário, adicionando-o à lista que esta contém.

```
for line in fHandler:

    # RECOLHA REGEX -----
    tknizer.input(str(line))
    tokenizerValues = [x.value for x in tknizer]

    try:
        emdR = emd.emdRecord(tokenizerValues)
    except:
        nWrongRegisters = nWrongRegisters + 1
        print("#> error: ", sys.exc_info()[0], " occorred")
    year = (emdR.date).split('-')[0]
```

```
if (emdR.gender == 'F' or emdR.gender == 'f') and int(emdR.age) < 35:       # 'F' && Age < 35
    ((_years[year])[fLT35]).append(emdR)
```

Figure 12: leitura com tokenizer

2.6 Queries

2.6.1 Datas extremas

Para identificar as datas extremas, isto é, a mais antiga e a mais recente, a cada linha processada pelo tokenizer retiramos o ano respetivo do registo e atualizamos duas variáveis que, respetivamente, representam as datas que pretendemos obter. Deste modo, o procedimento foi calcular sempre o mínimo e o máximo entre esses valores.

```
oldestDate = datetime.max
newestDate = datetime.min
```

```
auxDate = datetime.strptime(emdR.date, '%Y-%m-%d')

if oldestDate > auxDate:
    oldestDate = auxDate

elif newestDate < auxDate:
    newestDate = auxDate
```

Figure 13: datas extremas

2.6.2 Distribuição por género e idade

Após ter a estrutura de dados devidamente preenchida depois da leitura do ficheiro, resta-nos explorá-la de modo a obter a informação que pretendemos. Ora, o que fazemos é criar um dicionário que, para cada ano existente nos registos, prepara quatro chaves que remetem para o número de indivíduos do sexo masculino e feminino, divididos por 2 classes etárias, os que têm menos de 35 anos e os que têm 35 ou mais. Por forma a facilitar a posterior amostra dos indicadores utilizados para esta pesquisa, utilizamos também um outro dicionário que apresenta apenas essas mesmas quatro chaves e a elas fazem corresponder uma lista com todos os registos que nesse filtro se enquadrem.

```
perYear = {}
emdRegisters = {emdLDS.fLT35: [], emdLDS.fGET35: [], emdLDS.mLT35: [], emdLDS.mGET35: []}

for Year in dataset:                          ## por casa ano identificado

    if Year not in perYear:                   ## adicioná-lo
        perYear[Year] = [
                        {emdLDS.fLT35: 0, emdLDS.fGET35: 0},      ## Dicionário para o género 'feminino'
                        {emdLDS.mLT35: 0, emdLDS.mGET35: 0}       ## Dicionário para o género 'masculino'
                        ]
```

Figure 14: dicionário organizado por género e idade

2.6.3 Distribuição por género

Realizar uma procura apenas por género, depois de ter preparado uma procura por género e idade, torna-se mais fácil. Limitamo-nos a percorrer os dados já calculados e, por ano, juntamos o número de elementos de cada sexo, não atentando a restrição da idade. Desta forma, acabamos com um dicionário que, a cada ano, indica o número de indivíduos de cada sexo que têm registos efetuados, além de uma última chave que apresenta o mesmo conceito, mas para o total dos anos.

2.6.4 Distribuição por modalidade

De forma análoga aos temas anteriores, as modalidades são organizadas num dicionário que, a cada ano, faz corresponder um novo dicionário com as modalidades nesse ano registadas, às quais se relacionam listas com os devidos registos. Contudo, e por forma a agilizar os cálculos referentes a esta distribuição, voltamos a criar um outro dicionário que, a cada modalidade existente no universo dos registos, faz corresponder um dicionário com todos os anos que o dataset contém e o respetivo número de indivíduos registados.

```
modalitiesDict = {}
## { Modality : { Ano01 : #n, Ano02 : #n } }
```

Figure 15: dicionario de modalidades

2.6.5 Distribuição por Morada

As moradas foram organizadas por forma a obter um dicionário onde todas as suas chaves seriam as respetivas moradas e, a cada uma, fazia-se corresponder a lista com os registos respetivos. Desta forma, tanto os registos já se encontram devidamente subdivididos, como é possível obter as respetivas estatísticas, visto que se pode calcular o tamanho das listas para esse efeito.

2.6.6 Distribuição por estatuto de federado e resultado médico

De modo a conseguir identificar quais atletas possuem o estatuto de federado, a partir da informação presente na estrutura de dados principal, criou-se um dicionário que, a cada ano, separa os atletas federados dos atletas não federados, assim como o número de registos com essa característica. Visto que os resultados médicos também são representados como um boolean e de modo a diminuir o tempo de execução do programa, estes registos também são tratados ao mesmo tempo e guardados nesse dicionário sob uma key diferente.

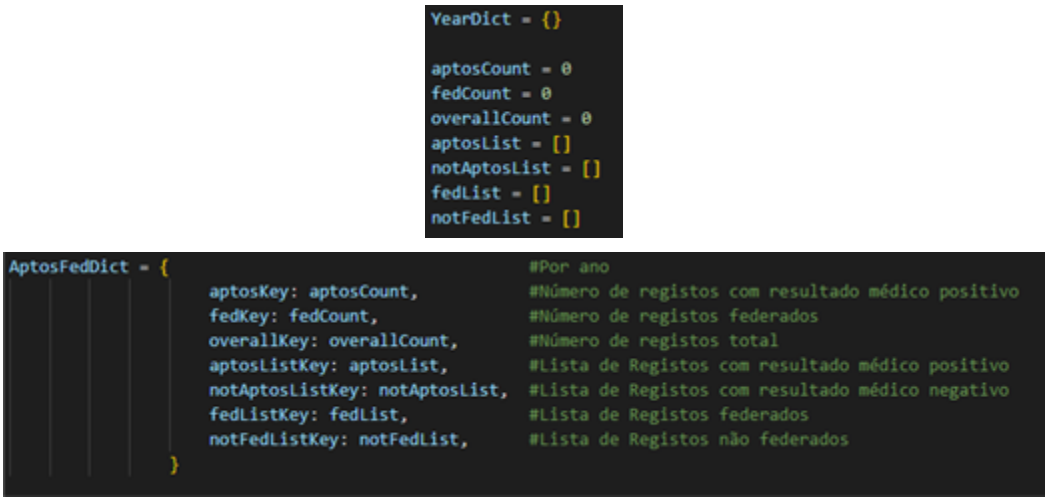

Figure 16: dicionário de federados e aptos

3 Resultados HTML

De modo a poder visualizar melhor a informação presente nos dados, cada alínea presente na página principal de html (index.html) está acompanhada de um gráfico com os dados relativos à mesma. Para este efeito, houve um recurso às bibliotecas de python matplotlib, numpy e pandas.

3.1 Datas extremas

São apresentadas todas as datas, por ordem, assim como o nome e email do indivíduo. Como extra, é ainda possível verificar-se a idade, género, se é federado e se se encontra apto.

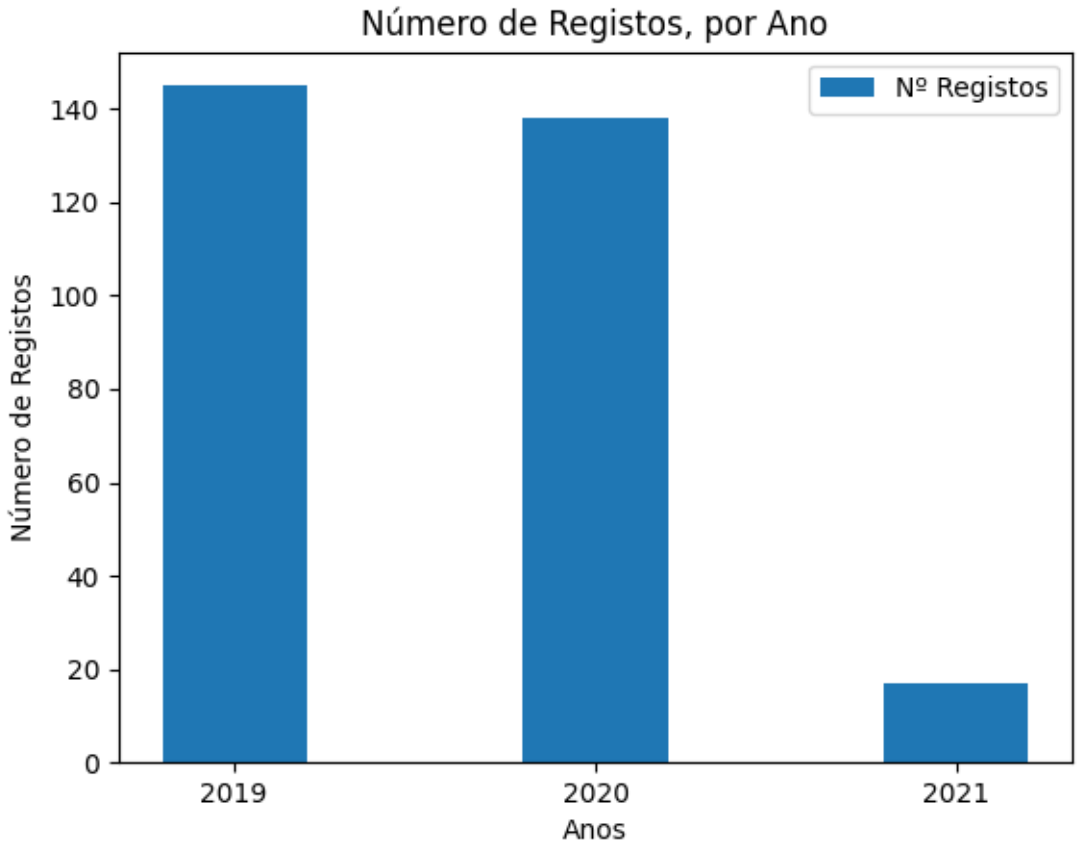

Figure 17: Distribuição por data

3.2 Distribuição por género

Por ano, dividimos os registos em feminino e masculino, os quais ordenamos por ordem alfabética ascendente do nome dos mesmos, apresentando, ainda, a idade e o email.

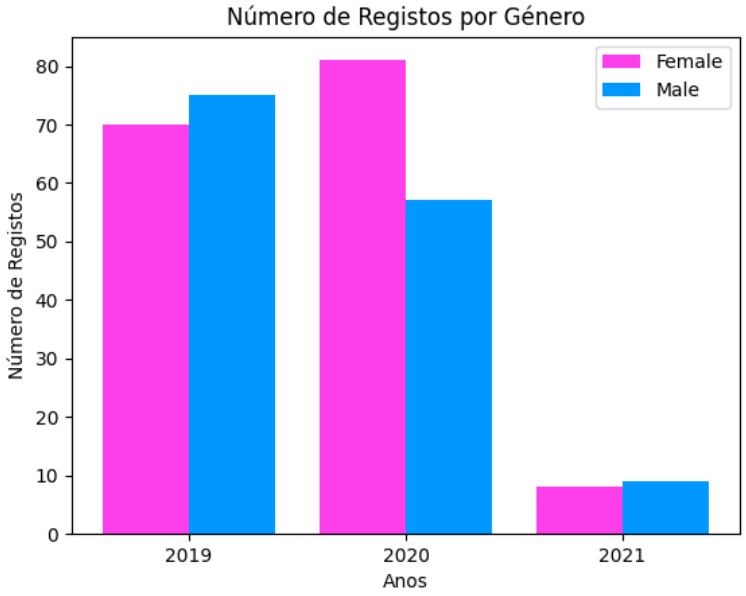

Figure 18: Distribuição por género por ano

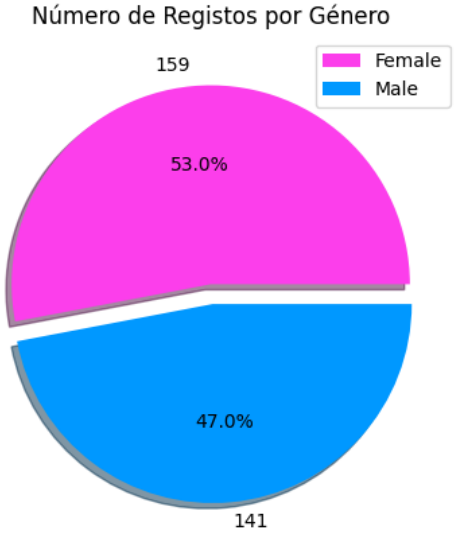

Figure 19: Distribuição por género no total

3.3 Distribuição por modalidade

Organizados por ordem alfabética ascendente do nome de cada modalidade presente nos registos, os mesmos são ainda apresentados por ordem alfabética do seu nome.

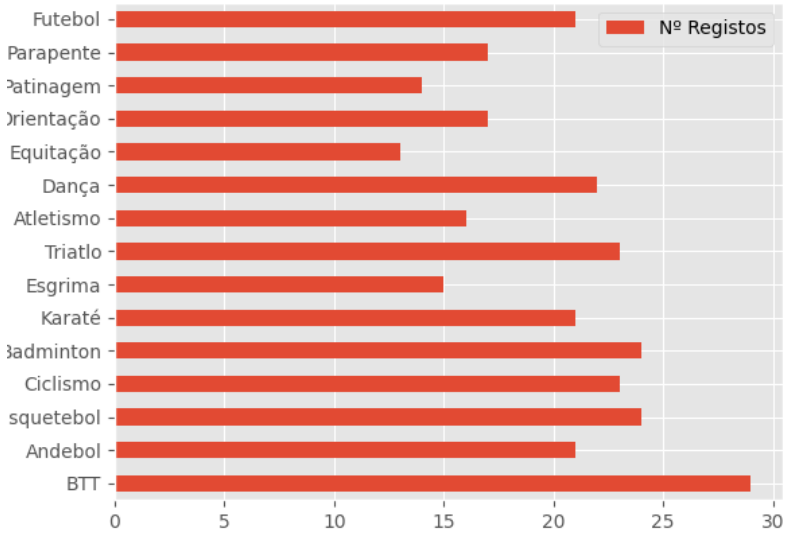

Figure 20: Distribuição por modalidade

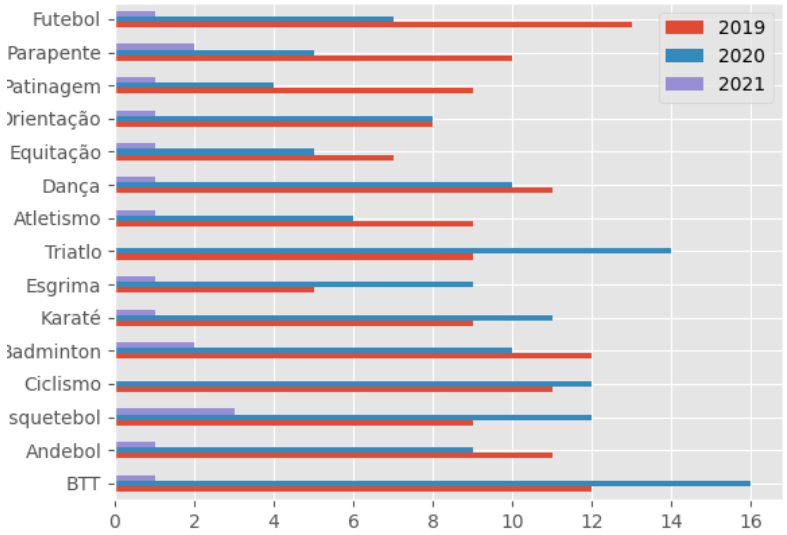

Figure 21: Distribuição por modalidade

3.4 Distribuição por idade e género

Respeitando as duas classificações etárias sobre as quais aplicaríamos a nossa procura, isto é, menores que 35 anos e maiores ou iguais a 35 anos, apenas aplicamos esse filtro aos respetivos géneros. Desta forma, e admitindo as quatro secções, os registos estão listados por forma a aparecerem ambos o nome, idade e género dos mesmos.

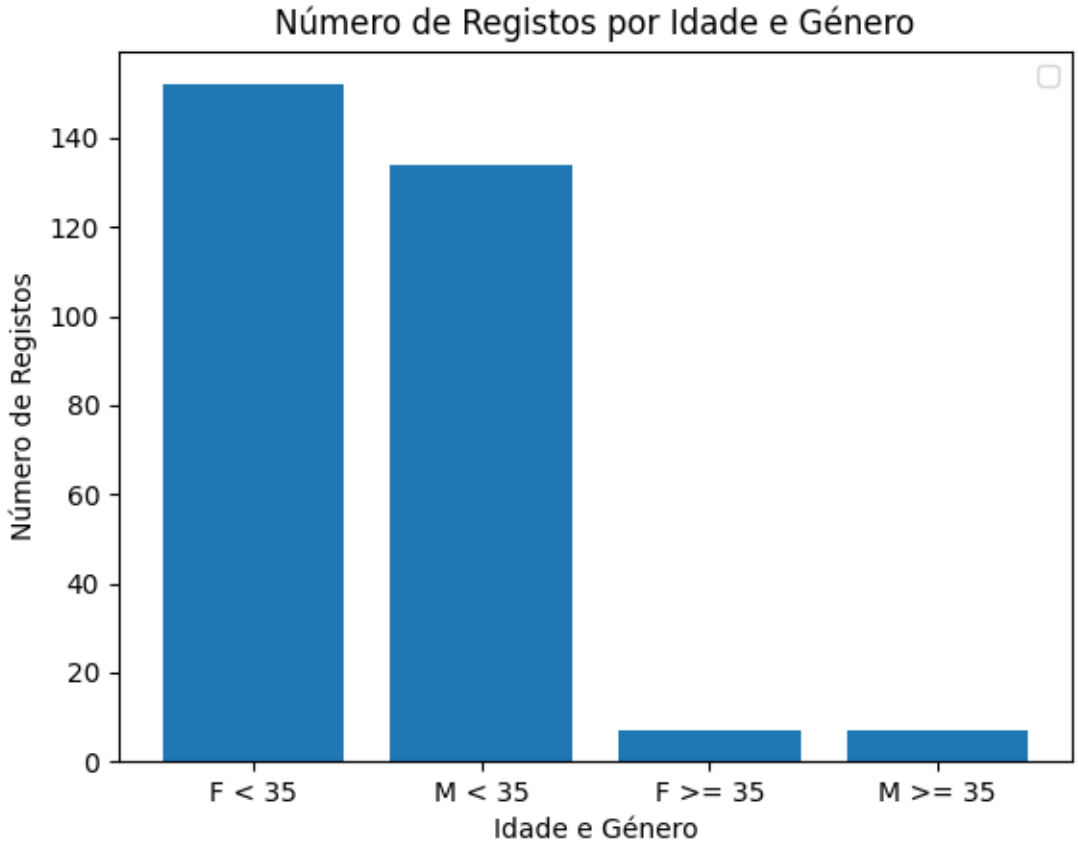

Figure 22: Distribuição por idade e género

3.5 Distribuição por morada

As moradas foram divididas sob as letras iniciais das mesmas, de maneira que, a cada morada, faz-se corresponder a lista de registos correspondente, disponibilizando ambos o nome e modalidade praticada pelo indivíduo em questão.

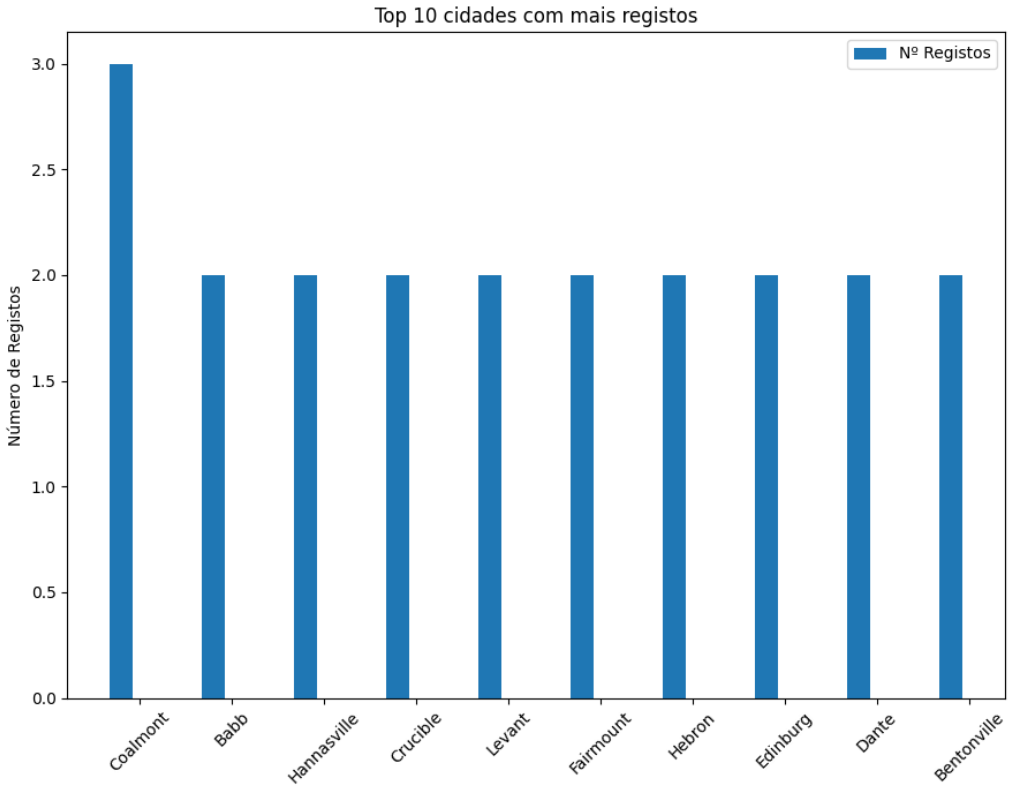

Figure 23: Representação das 10 moradas com mais registos

3.6 Distribuição por estatuto de federado

Aplicando uma separação por anos, temos duas divisões respetivas aos indivíduos que são federados e que não o são, apresentando-os por ordem alfabética do nome e disponibilizando ainda a idade, género e a modalidade que pratica.

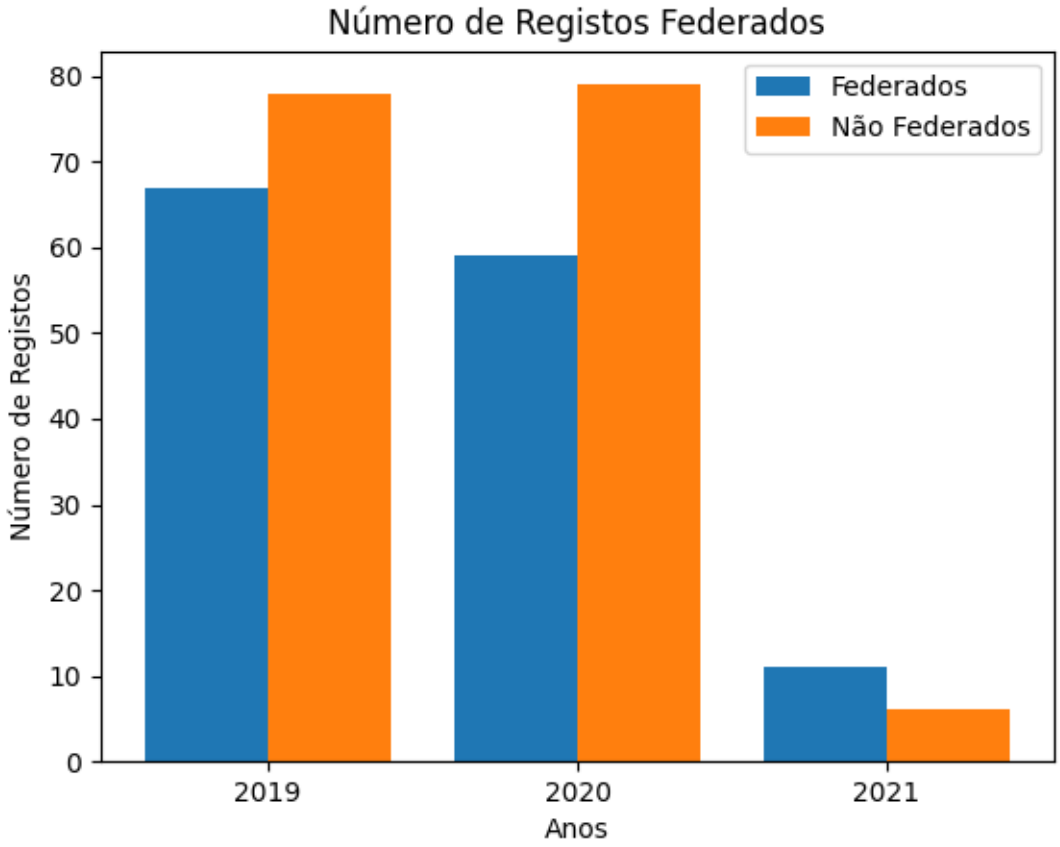

Figure 24: Distribuição por federados

3.7 Percentagem de aptos e não aptos

Com uma separação por anos, os registos são apresentados por ordem alfabética, divididos pelo resultado do exame médico, disponibilizando também idade, género e modalidade praticada.

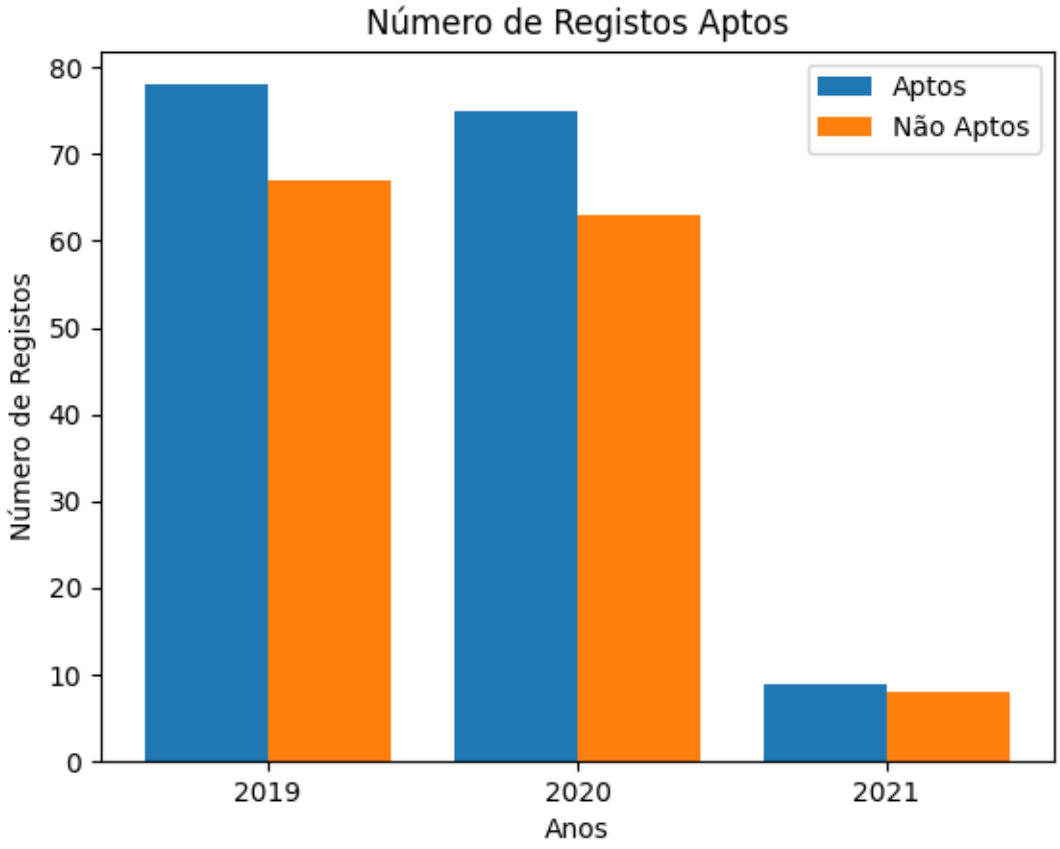

Figure 25: Distribuição por Resultados de exame médico

4 Conclusão

O projeto, em si, permitiu desenvolver as capacidades de lidar com python e aplicar os conhecimentos sobre expressões regulares para processar e lidar com o conteúdo de ficheiros de texto. No desenvolvimento do mesmo não sentimos grandes dificuldades, contudo, numa próxima versão gostávamos de tornar as nossas pesquisas mais eficientes.