Processo Seletivo Turing USP - 2023

Case Técnico

Pontuação Total: 100 pontos

Olá, candidate! Seja muito bem-vinde ao processo seletivo Turing USP 2023! Nessa fase do nosso processo seletivo, você terá que resolver um case técnico. São várias questões, incluindo dois desafios e um texto opcional em formato de Turing Talks.

Envios incompletos serão considerados, mas encorajamos que tente fazer o máximo que for possível. Algumas questões são propositalmente desafiadoras, então saiba o que atacar, padawan. O intuito aqui é entender o seu raciocínio e qualidade de código, bem como o seu desenvolvimento no processo.

Os desafios (assim como o Turing Talks) **valem uma pontuação extra** e você pode escolher se deseja fazê-los ou não. Mas **atenção**: você deverá escolher apenas um dos desafios para realizar. Caso entregue ambos corrigiremos apenas um, portanto veja qual faz mais sentido para você. Independentemente do desafio que escolher realizar (ou não), você sempre poderá entregar o seu texto no formato Turing Talks.

Lembrete: isso não é uma prova da USP, **sinta-se livre para procurar guias, vídeos e outros recursos na internet para te ajudar**.

Caso tenha alguma dúvida, é só perguntar para a gente no nosso Discord, onde temos canais específicos para cada parte do case. Se preferir, você pode também entrar em contato pelo nosso e-mail turing.usp@gmail.com, pela nossa página no instagram instagram.com/turing.usp ou mandar uma mensagem para algum dos membros que estão listados em cada parte do case.

Tentaremos responder o quanto antes! Além disso, **teremos monitorias para ajudá-los,** cujas datas serão divulgadas em nossas redes sociais, então prestem atneção! Elas ocorrerão de maneira **online em nosso Discord e presencial na USP**.

Estrutura e Orientações

O case é dividido em três partes:

1. **Lógica de programação e algoritmos**
   Essa parte deve ser entregue em python. Para essa parte, uma primeira etapa da correção será feita automaticamente por um programa de computador. Para evitar problemas durante a correção automática:
   - É **proibida a utilização de acentos nas palavras na hora de escrever no seu código**;
   - É obrigatório **utilizar os nomes das funções iguais aos nomes fornecidos nos enunciados.**
2. **Análise de dados e Inteligência artificial**
   Essa parte deve ser entregue em jupyter notebooks escritos em python.
3. **Texto opcional no formato Turing Talks**
   Essa parte deve ser entregue em formato PDF.

Também é importante lembrar que quanto mais bem escrito seus códigos melhor, pois facilitará o entendimento e a lógica por trás dos mesmos. Neste sentido, se esforce para dar bons nomes de variáveis, priorize lógicas simples e se preciso adicione comentários; códigos ilegíveis podem resultar em perda de pontuação.

Indicação de Materiais

Caso você não tenha o costume de programar em python ou precise revisar algum conceito da linguagem, recomendamos alguns de nossos posts no medium e vídeos no youtube como apoio para os exercícios:

- Tutorial de python:
  - Parte 1: Programação | Python — Parte 1 | by Lucas Fentanes Machado | Turing Talks
  - Parte 2: Programação | Python — Parte 2 | by Fernando Matsumoto | Turing Talks

- Cursos gratuitos:
  - Curso de Python Básico do Turing USP: Python Básico - YouTube
  - Curso em vídeo de Python Básico: Curso Python #01
  - Introdução à programação (Python): MAC2166 em Python - YouTube
  - Introdução à Ciência da Computação com Python: Coursera - Parte 1
  - Introdução à Ciência da Computação com Python: Coursera - Parte 2

- Textos e livros:
    - O Tutorial Python: python.org (Tutorial mais direto a sintaxe do python, recomendado para quem já tem uma base de lógica de programação)

Formato de Submissão

Quando terminar seu case, envie-o nesse forms. **Só aceitaremos uma única submissão**.

Você deve seguir o seguinte formato:

- As respostas dos exercícios de lógica de programação e algoritmos devem ser entregues em arquivos python (.py). Note que **esperamos um arquivo por questão/item**, com a questão devidamente identificada no nome do arquivo.
- As respostas dos exercícios de análise de dados e IA devem ser entregues em Jupyter Notebook (.ipynb), escritos em python (não será permitido o uso de R).
- O seu texto (Turing Talks) deve ser entregue em pdf.

Os nomes dos arquivos e pastas devem identificar as questões que eles resolvem, **conforme a estrutura abaixo** (só inclua as partes que você resolver, não precisam estar completas para mandá-las). A submissão deve ser feita por meio de um único arquivo zip com o nome "`nusp_nome_sobrenome.zip`".

```
12345678_alan_turing.zip
├ 1_logica
│ ├ q1a.py
│ ├ q1b.py
│ ├ q2a.py
│ ├ q2b.py
│ └ desafio.py
├ 2_analise-e-ia
│ └ analise_e_modelos.ipynb
└ 3_turing-talks
  └ turing-talks.pdf
```

Observações:

- **Recomendamos dar uma passada em todas as partes antes de começar a fazer o case para escolher o que você se sente mais confortável em fazer!** E não fique assustado com o tamanho dos enunciados, a maioria do conteúdo foi colocado justamente para ajudar na realização da tarefa.
- Como nas outras etapas, utilize nusp 0 (zero) se você for membro da USP, mas não tiver nusp.

Parte 1

Lógica de Programação e Algoritmos

Pontuação Total: 50 pontos

Aqui se inicia a parte de programação do case. Ela consiste em uma série de exercícios que abordam lógica de programação e algoritmos em um nível intermediário (ou seja, partimos do pressuposto que os candidates já possuem uma base sólida de programação em python), por isso alguns exercícios podem parecer desafiadores.

Caso tenha alguma dúvida nessa parte do case (seja de interpretação do enunciado ou alguma dúvida técnica), é só perguntar para a gente no nosso Discord (no canal **#dúvidas-lógica-2)**, no nosso e-mail turing.usp@gmail.com, no nosso instagram instagram.com/turing.usp ou mandar uma mensagem para algum dos membros da equipe abaixo:

João Pedro Menezes	+55 (11) 99352-9753
Guilherme Castelo	+55 (11) 91646-9899
João Feitosa	+55 (11) 96928-5220

Essa parte do case contém **2 questões** (Q1, Q2) de temas diferentes, cada uma com **itens A e B**, além de uma questão **desafio**. No final de cada item e do desafio são dados exemplos de entrada e saída para que você teste os seus programas.

Algumas orientações gerais:

- Antes de começar a programar, leia os exemplos. Eles podem ajudar na compreensão dos exercícios.
- Depois de escrever os seus programas, é importante testá-los com os exemplos dados, mas lembre-se de que você também pode escrever os seus próprios exemplos.

Observação técnica: nessa parte do case, estamos considerando o **python 3** (qualquer versão 3.x). Essa é a versão utilizada pelo repl.it, então caso esteja utilizando essa plataforma, você não precisa se preocupar com isso.

Exemplo de Resolução

Para nos auxiliar na correção dos cases, utilizamos um sistema de correção automática, que compara o gabarito com a sua solução. Por isso, se atente ao formato exato das saídas.

OBS: Fazemos também uma checagem manual do seu código e, em caso de erro, buscamos entender o que foi feito e dar notas parciais. Por isso não deixe de tentar e explicar sua linha de raciocínio!

OBS: Para evitar alguns problemas de compatibilidade entre softwares, **não utilizamos acentos nos programas**, preste atenção na hora de imprimir.

Item A (Exemplo de entrada/saída via função)

Escreva uma função que, dados dois números, **retorne a soma deles**.

Exemplo:

```
Input:  soma(1, 2)
Output: 3
```

Resolução:

```
def soma(a, b):
    return a + b
```

Q1. Situação de barril no Turing

Com a entrada dos membros no Turing, os membros decidiram organizar uma festa de boas-vindas; para isso, foi alugado o salão de festas do Sr. Alan. No entanto, o Sr. Alan é muito organizado e não gosta de bagunças; por isso, pediu uma lista com os emails de todos os membros do Turing, para que seus funcionários enviassem os termos de responsabilidade para o uso de sua propriedade. Todavia, o Sr. Alan é um homem muito antigo e por isso exigiu que a lista fosse entregue impressa.

Infelizmente, como os fundos do grupo foram todos destinados à festa, foi decidido utilizar o mínimo possível de tinta para imprimir a lista com emails. Um dos membros sugeriu a seguinte solução: alinhamos os e-mails à direita. A partir do segundo e-mail impresso, **os caracteres iniciais** do próximo e-mail a ser impresso que coincidirem com os do e-mail acima são omitidos, ficando apenas um espaço em branco.

Por exemplo para os emails `rian.fernandes@usp.br`, `rianzinho.2023@usp.br`, `rian_e_isah_sz@usp.br`, `davi.do.turing@usp.br` e `feitosah_do_rh@usp.br` a impressão ficará da seguinte forma:

```
rian.fernandes@usp.br
      zinho.2023@usp.br
      _e_isah_sz@usp.br
davi.do.turing@usp.br
feitosah_do_rh@usp.br
```

Pelo exemplo acima é possível perceber que foram economizadas 8 letras.

Item A

Você, entusiasmado por ter entrado no grupo, se propôs a fazer uma função ***conta_economia(emails)*** que recebe a lista de emails e retorna quantas letras foram economizadas. Neste item, você deve determinar quantos caracteres podem ser economizados **mantendo a ordem** dos emails.

A sua função deve receber uma lista com N emails e deve retornar um número indicando quantos caracteres podem ser economizados para o conjunto N de emails.

Exemplo:

```
Input:
conta_economia(["rian.fernandes@usp.br",
                "rianzinho.2023@usp.br",
                "davi.do.turing@usp.br"])

Output: 4

Input:
conta_economia(["davi.turing.cp@usp.br",
                "diva.turingusp@usp.br",
                "diva.estudausp@usp.br",
                "renatah.turing@usp.br",
                "reginaldo.fake@usp.br"])

Output: 8
```

OBS 1: Considere que todos os emails passados têm o mesmo tamanho.

OBS 2: As letras do domínio nunca serão economizadas, isto é, ao chegar no `@usp.br` você não deve considerar que tais letras podem ser omitidas. Considere também que os emails só possuem um "@" cada, sendo este aquele que inicia o domínio.

Referências:

- Vetores - IME-USP
- Ordenação de vetores

Item B

Enquanto os membros do Turing estavam imprimindo a lista de e-mails, eles perceberam que a impressora foi infectada por um vírus e está imprimindo de forma incorreta. Depois de olhar para várias páginas impressas por um tempo, você percebe que ele está imprimindo cada linha de dentro para fora. Em outras palavras, a metade esquerda de cada linha está sendo impressa a partir do meio da página até a margem esquerda. De maneira similar, a metade direita de cada linha está sendo impressa a partir da margem direita e prosseguindo em direção ao centro da página.

Por exemplo, o e-mail `rianzinhosz@usp.br` está sendo impresso como `ohniznairrb.psu@zs`. Isso ocorre pelo seguinte processo:

`rianzinhosz@usp.br` original

`rianzinho | sz@usp.br` orig dividido

`ohniznair | rb.psu@zs` div invertido

`ohniznairrb.psu@zs` invertido

Além disso, você percebeu que alguns e-mails também tiveram seus domínios embaralhados/errados após a impressão, inviabilizando o envio de tais e-mails ao professor. Um dos exemplos é o e-mail do membro Davi, que foi impresso como `irut.ivadrb.pus@gn` e após a desinverte-lo temos `davi.turing@sup.br`. Perceba que nesse caso o domínio `@usp.br` está errado.

Sua tarefa nesta questão é criar a função ***corrige_emails(emails)***, *que* recebe um array *emails* com N e-mails embaralhados e então retorna uma lista com os e-mails corretamente formatados ou `ERRO nas posições em` que o domínio esteja errado.

Exemplo:

```
Input:
corrige_emails(["id_atanerrb.psu@av",
                "t.alalalimacrb.repsu@ppo",
                ".orbmem_ovonrb.psu@gnirut"])

Output: ['renata_diva@usp.br','ERRO', 'novo_membro.turing@usp.br']
```

OBS: Caso o número de caracteres seja ímpar, considerar o piso da metade. Assim, a metade de 25 fica 12 ao invés de 12,5.

Q2. Estacionamento do Turing

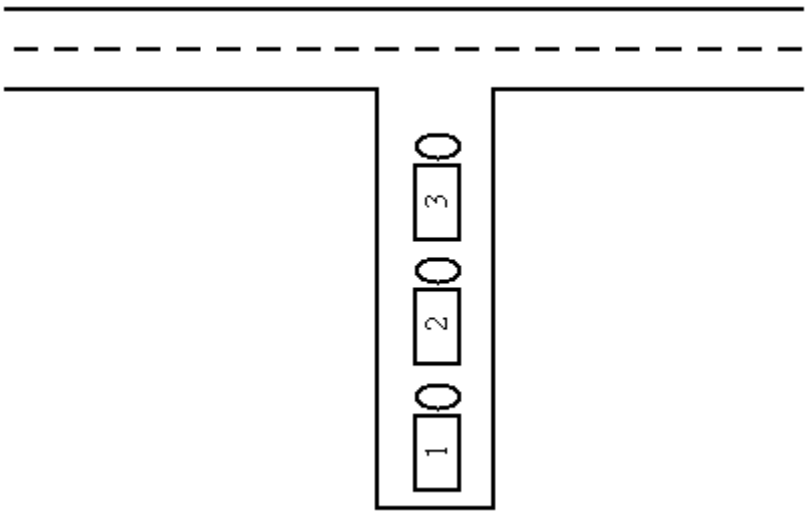
Para integrar melhor os membros do Turing, foi organizado um evento em uma chácara próxima no interior de São Paulo e, para que todos pudessem chegar com conforto, Feitosa organizou um sistema de caronas que foi um sucesso.

No entanto, no dia do evento, muitos carros começaram a chegar e estava ficando cada vez mais difícil estacioná-los. Então, para tentar resolver a situação, Davi

resolveu tentar aproveitar um corredor gigante na lateral da casa e fazê-lo de estacionamento.

O corredor comporta K carros um atrás do outro e tem apenas uma entrada pela parte frontal da casa.

Ao entrar com o primeiro carro, ele ocupa a posição próxima a parede (carro 1). O próximo carro fica à frente deste (carro 2) e assim por diante. A imagem a seguir demonstra um exemplo:



Obviamente, não é possível que um carro passe por cima de outro, portanto só é possível que um carro saia do estacionamento se ele for o último da fila (nesse exemplo o carro 3).

Referências:

- Vetores - IME-USP
- Pilha - IME-USP
- Pilha em Python - Pandas IME USP
- Pilhas em Python com listas

Item A

Como era muito difícil administrar como os carros estacionarem sem problemas em apenas um corredor, você, como um bom programador e um grande membro do Turing, decidiu ajudar Davi na organização. Dessa forma, você decide escrever uma função

estacionamento_ok(K, instantes) que verifica se é possível que **todos os carros** estacionem e saiam sem problemas.

A entrada de sua função é composta por um **inteiro K**, que indica quantos carros cabem no estacionamento, e por um **vetor no qual cada elemento é a representação do que ocorreu no instante correspondente ao índice desse elemento**. Cada elemento do vetor será um valor C. Se C for positivo, o carro indicado pelo número está entrando no estacionamento. Se C for negativo, o carro indicado pelo número está saindo do estacionamento. O elemento no índice 0 do vetor indica a ação ocorrida no instante 0, o elemento no índice 1, a ação ocorrida no instante 1 e assim por diante. **Veja o exemplo 1 para maiores explicações**.

Sua função deve retornar "`sim`", caso seja possível que todos os carros estacionem e saiam sem problemas e `nao`, caso contrário. Ou seja:

- Ao primeiro sinal de falta de espaço para que um carro entre ou saia do estacionamento, sua função deve retornar "`nao`";
- Caso um carro não consiga entrar no estacionamento por ele já estar lotado, seu programa deve retornar "`nao`";
- Todos os carros devem entrar **e** sair, portanto se o estacionamento não terminar vazio a sua função deve retornar "`nao`"

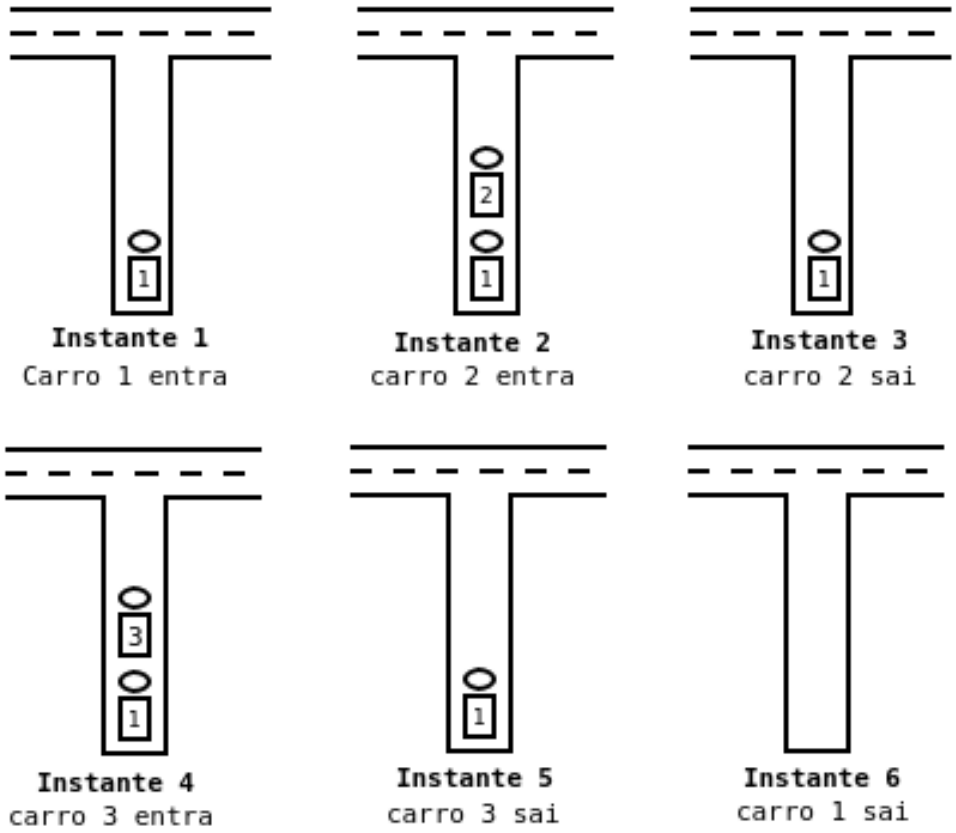
Dica: Utilize uma pilha para fazer a inserção e remoção dos carros no estacionamento.

Exemplo 1:

```
Input: estacionamento_ok(3,[1,2,-2,3,-3,-1])

Output: sim
```

No exemplo acima, quando temos `-3`, por exemplo, significa que o carro `3` está saindo do estacionamento. Veja:



Exemplo 2:

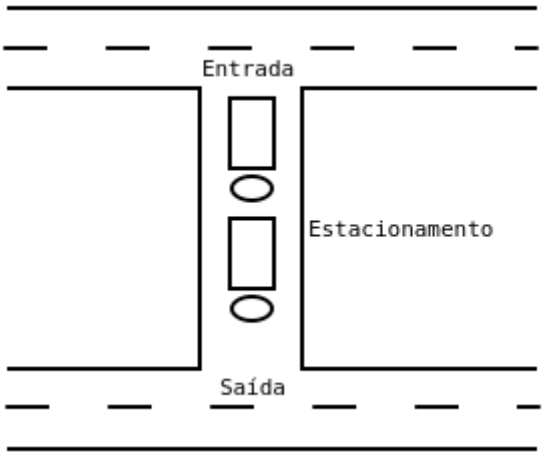
```
Input: estacionamento_ok(3,[1,2,-2,3,5,-3,-1,-5])

Output: nao
```

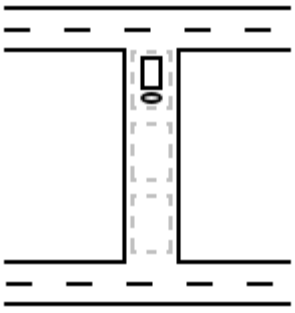
OBS: O usuário pode considerar que um carro que já esteja estacionado não tentará entrar novamente, a não ser que já tenha saído do estacionamento. De forma análoga, um carro que não está estacionado não tentará sair.

Item B

Ao voltar do evento, Rian decide se inspirar no código do item anterior para analisar como ele estacionava seu carro na rua em que mora. Antes, essa rua era sem saída em uma das extremidades, mas agora possui uma entrada e uma saída conforme a seguinte imagem mostra:

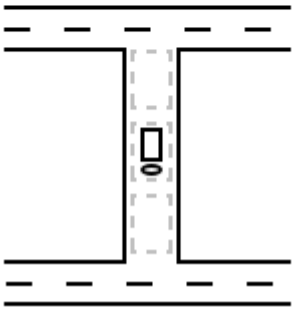


O estacionamento na rua de Rian funciona de uma maneira muito peculiar, pois a cada novo instante o carro avança uma posição dentro dele. Veja o exemplo para um estacionamento de tamanho K = 3 e um determinado carro:



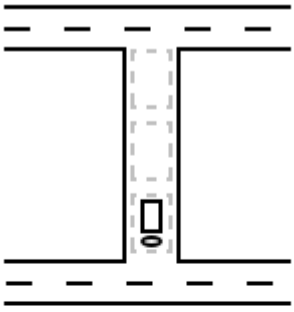
Instante 1:

O carro está no estacionamento



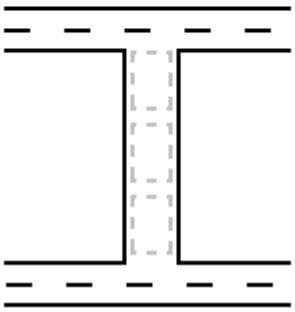
Instante 2:

O carro avança 1 posição no estacionamento



Instante 3:

O carro avança mais 1 posição no estacionamento



Instante 4:

O carro sai do estacionamento

Pelo exemplo: acima, percebemos que, uma vez que o carro entra no estacionamento, a cada instante ele avança uma posição até sair. Dito isso, você consegue ajudar Rian a descobrir como está o estacionamento em um determinado instante? Crie a função ***estado_atual(K, entradas, T)*** para resolver esse problema.

Sua entrada é composta por **K**, o tamanho do estacionamento; por um vetor ***entradas***, no qual cada elemento representa o horário de entrada E_n do carro associado àquela posição (a primeira posição do vetor representa o carro 1, a segunda posição o carro 2 e assim por diante); e por um instante **T**, no qual queremos saber status do estacionamento.

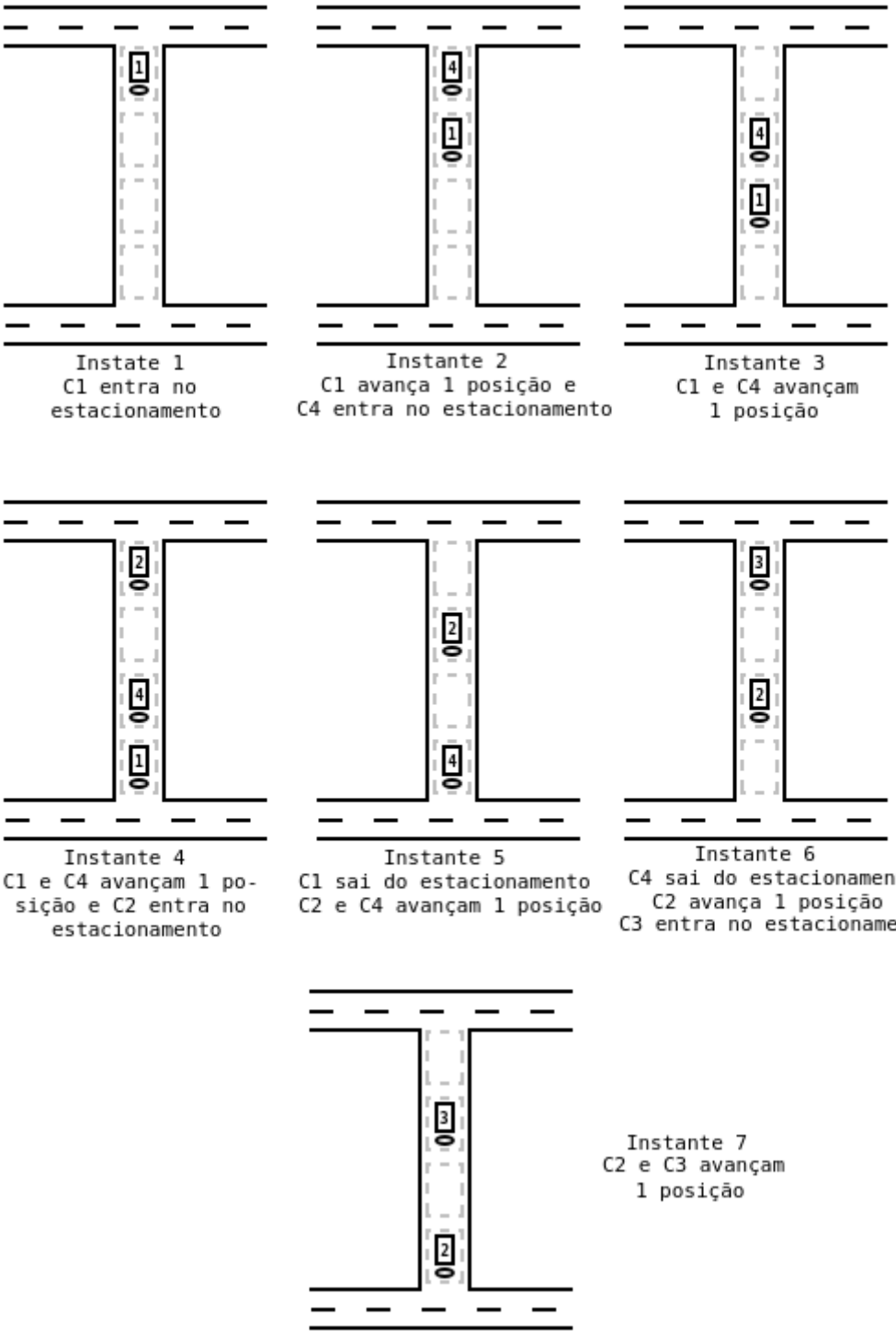
Sua saída deve ser um **vetor de K números**, que indica como está o estacionamento no instante T. Caso uma posição do estacionamento não possua nenhum carro, você deverá preencher aquela posição com `0`. Além disso, o primeiro número do vetor representa o carro na primeira posição do estacionamento, isto é, logo depois da entrada. Veja o exemplo a seguir para maiores explicações.

Exemplo 1:

```
Input: estado_atual(4,[1,4,6,2],7)

Output: [0, 3, 0, 2]
```

No exemplo acima perceba que K = 4 (tamanho do estacionamento). Além disso, sabemos que: o carro 1 entrou no instante 1, o carro 2 entrou no instante 4, o carro 3 entrou no instante 6 e o carro 4 entrou no instante 2. Por fim, o instante em que queremos saber como está o estacionamento é o 7.



Exemplo 2:

```
Input: estado_atual(5,[3,6,12,9,4,15,16],7)

Output: [0, 2, 0, 5, 1]
```

OBS: O usuário pode considerar que dois carros nunca entrarão no mesmo instante no estacionamento

OBS 2: Se o instante de interesse é X e um carro **entra** no estacionamento no mesmo instante X, considera-se que o carro **já está** no estacionamento. De maneira análoga, se um carro **sai** do estacionamento no mesmo instante X de interesse, considera-se que o carro **não está** no estacionamento.

QUESTÃO DESAFIO - Bônus

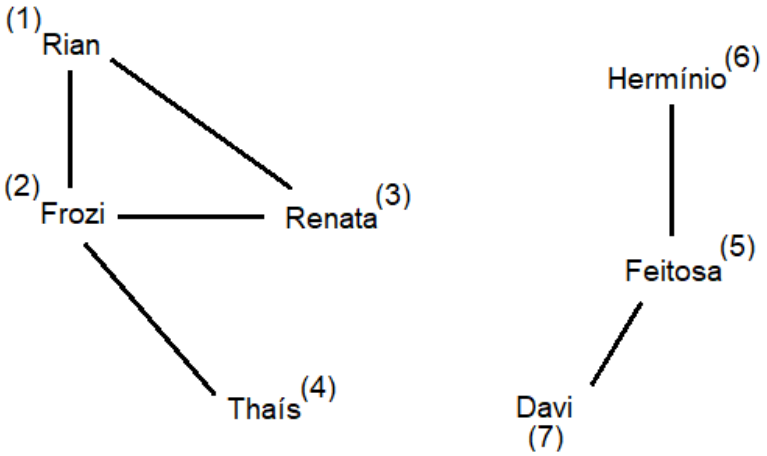
Atenção: esse case tem dois desafios: um de *análise de dados e IA* e outro de *lógica de programação e algoritmos*. Lembre-se de que:

- Os desafios são **opcionais** e valem uma pontuação extra (25 ptos);
- Os desafios não podem elevar a sua nota no case técnico acima dos 100 pontos; e
- Se optar por fazê-los, você deve escolher **no máximo um deles**. Em outras palavras, **não entregue ambos os desafios**.

Amizades no Turing

O Turing é formado por membros de vários institutos da USP, como IME, IF, FFLCH, FEA e POLI. Os membros da área de RH perceberam que micro grupos estão se formando dentro do Turing devido a proximidade de membros do mesmo instituto, o que atrapalha a integração geral do grupo.

Você, a fim de ajudar a área de RH do grupo a entender tal fenômeno, se comprometeu a criar uma função ***micro_grupos(membros, relacionamentos)*** que dada a relação de amizade entre os participantes do Turing, devolve quantos micro grupos existem atualmente no grupo. Por simplicidade, os relacionamentos podem ser representados como um **grafo**, como o abaixo:



Na imagem acima, cada nó é um membro, e cada aresta (linha) representa um relacionamentos, demonstrando que as duas pessoas envolvidas são amigas. Analisando o grafo, percebemos que o número de micro grupos é **2**.

A entrada de seu programa será composta de um inteiro *membros* , que representa o número de membros do Turing e de um vetor *relacionamentos*, composto por outros vetores de 2 elementos A e B, correspondendo a um relacionamento entre os membros A e B. A partir disso, seu programa deve retornar o número de micro grupos existentes demtro do Turing.

OBS: Caso um determinado subgrafo tenha apenas 1 membro (pessoa isolada), isto é, um subgrafo com apenas 1 nó, ele **deve** ser considerado como um micro grupo.

Exemplo:

```
Input: micro_grupos(7, [[1,2],[2,3],[3,1],[3,4],[6,5],[5,7]])

Output: 2
```

OBS: Nesse exemplo, que corresponde ao grafo mostrado no enunciado, o primeiro valor da entrada é o número de membros (7) e o segundo é o vetor de relacionamentos. O grafo correspondente é o mesmo que foi mostrado no início dessa questão.

Referências:

- Introdução à representação de grafos em Python
- Grafos

Parte 2

Análise de Dados e IA

Pontuação Total: 50 pontos

Caso tenha alguma dúvida nessa parte do case (seja de interpretação do enunciado ou alguma dúvida técnica), é só perguntar para a gente no nosso Discord (no canal **#dúvidas-análise-1)**, no nosso e-mail turing.usp@gmail.com, na nossa página no instagram instagram.com/turing.usp ou mandar uma mensagem para algum dos membros da equipe abaixo:

João Feitosa	+55 (11) 96928-5220
Davi Félix	+55 (11) 96889-0585
João Zangelmi	+55 (11) 99352-9753

- Nessa etapa, você deverá realizar uma análise de dados básica. Para isso, você pode consultar, além de outros recursos disponíveis na internet, os seguintes links:
- O exemplo analise_wine.ipynb, que mostra uma análise básica de um dataset de vinhos tintos. O objetivo é identificar como as características do vinho afetam a sua qualidade.
- Os nossos posts sobre visualização de dados no medium: https://medium.com/turing-talks/turing-talks-9-visualiza%C3%A7%C3%A3o-de-dados-93df670d479

  e

  https://medium.com/turing-talks/como-visualizar-e-analisar-dados-com-python-f209bfbae68e
- É recomendado que você utilize algumas bibliotecas de visualização de dados para fazer as análises solicitadas, dentre elas recomendamos fortemente:
  - Pandas - Python: https://pandas.pydata.org/docs/
  - Matplotlib - Python: https://matplotlib.org/stable/contents.html

- Seaborn - Python: https://seaborn.pydata.org/
- Plotly - Python: https://plotly.com/python/

Lembramos que uma análise de dados de qualidade não é feita apenas com gráficos, **é necessário interpretar as informações que o gráfico apresenta**, tirando conclusões e *insights*.

Plataforma

Essa parte do case deve ser feita **utilizando jupyter notebook.** Se você não possui o jupyter instalado no seu computador, pode usar o **Google Colab,** uma ferramenta do Google que permite rodar jupyter notebooks na nuvem.

Para criar um notebook no Colab, acesse o link:

https://colab.research.google.com/notebook#create=true.

Caso vá fazer o desafio, recomendamos que utilize um runtime com GPU. Para isso clique em *Ambiente de Execução > Alterar tipo de ambiente de execução > Acelerador de hardware > GPU.*

O Dataset

O seguro de carro é uma proteção essencial para os motoristas, pois garante a cobertura de eventuais danos que possam ocorrer com o veículo. Existem diversos tipos de coberturas disponíveis, desde as mais básicas até as mais completas, que incluem proteção contra roubo, colisão, incêndio, entre outros.

A reclamação de seguro (em inglês, “insurance claim”) consiste no processo de reivindicação de indenização após a ocorrência de uma dessas eventualidades. A seguradora, então, analisa a solicitação e avalia, com base em diversos fatores, se haverá ou não a liberação do pagamento.

Neste exercício, você deverá analisar uma base de dados que reúne informações a respeito de “insurance claims" nos EUA em um determinado período de tempo. São apresentados dados relativos aos segurados e seus veículos, além do desfecho

apresentado para cada claim (isto é, **caso houve ou não a liberação do pagamento pela seguradora**).

Seu objetivo principal será verificar a **relação entre as características de um segurado e o resultado de sua claim**. Como cientista de dados, você deverá realizar uma análise das features do dataset e apresentar graficamente os insights que você obtiver ao longo desse processo.

Além de se basear em nosso post do Medium e no notebook de exemplo, você pode também utilizar outras funções e métodos para extrair mais informações úteis.

OBS: Não se preocupe em tirar insights "milagrosos", apenas busque utilizar e procurar métodos interessantes e **comunique** suas descobertas em forma de comentários. Estamos testando mais o seu esforço do que seus resultados aparentes.

Link do dataset:
https://drive.google.com/file/d/1wgUnkmZ1H0ew-uBpQCJUKxrYOQCqyy2q/view?usp=share_link

Para orientar sua análise, segue uma breve descrição de cada uma das colunas do dataset:

1. **ID** - Identificação;
2. **AGE** - Idade;
3. **GENDER** - Gênero;
4. **RACE** - Etnia;
5. **DRIVING_EXPERIENCE** - Anos de experiência no volante;
6. **EDUCATION** - Nível de educação formal;
7. **INCOME** - Categoria de renda;
8. **CREDIT_SCORE** - Pontuação de crédito. Reflete o quão bom pagador o indivíduo é;
9. **VEHICLE_OWNERSHIP -** Posse do veículo. Se o segurado é proprietário do veículo que dirige ou não;
10. **VEHICLE_YEAR** - Idade do veículo. Se o carro é anterior ou posterior ao ano de 2015;
11. **MARRIED** - Estado Civil;
12. **CHILDREN** - Se o indivíduo possui filhos ou não;

13. **LOCALITY**- Localidade;
14. **ANNUAL_MILEAGE** - Quilometragem anual percorrida;
15. **VEHICLE_TYPE** - Tipo de automóvel;
16. **SPEEDING_VIOLATIONS** - Número de infrações cometidas pelo segurado por excesso de velocidade;
17. **DUIS** - Número de infrações cometidas pelo segurado por embriaguez ao volante;
18. **PAST_ACCIDENTS** - Número de acidentes passados com o segurado envolvido.
19. **OUTCOME** - Resultado da reclamação de seguro (isto é, se a insurance claim foi aprovada ou recusada).

Execução

Para realizar esta parte do case, vamos ajudá-los dando uma direção geral para a análise. Lembre-se, porém, que você deve buscar outras análises que possa considerar relevantes para justificar sua conclusão sobre o tema proposto. Orientamos que divida o seu notebook em duas partes: **análise de features** e **análise do outcome**.

Reforçamos que a forma como os gráficos e as análises são expostos influencia a avaliação. Portanto, recomendamos fortemente que tudo seja explicado de forma clara e justificada, além dos gráficos serem bem construídos.

Para a construção de gráficos melhores, incentivamos que vocês, além de lerem os Turing Talks, vejam as documentações da bibliotecas utilizadas (os links estão acima), para que entendam alguns parâmetros que podem ser alterados, como cores, tamanho do gráfico, entre outros.

À medida que avançar com a sua análise, reúna as referências que utilizar (vídeos, textos, links do Stack Overflow, etc...) na parte final de seu notebook. Suas referências vão nos ajudar a avaliar seu processo de aprendizagem e evolução ao longo da realização deste case.

Análise de Features (15 pontos)

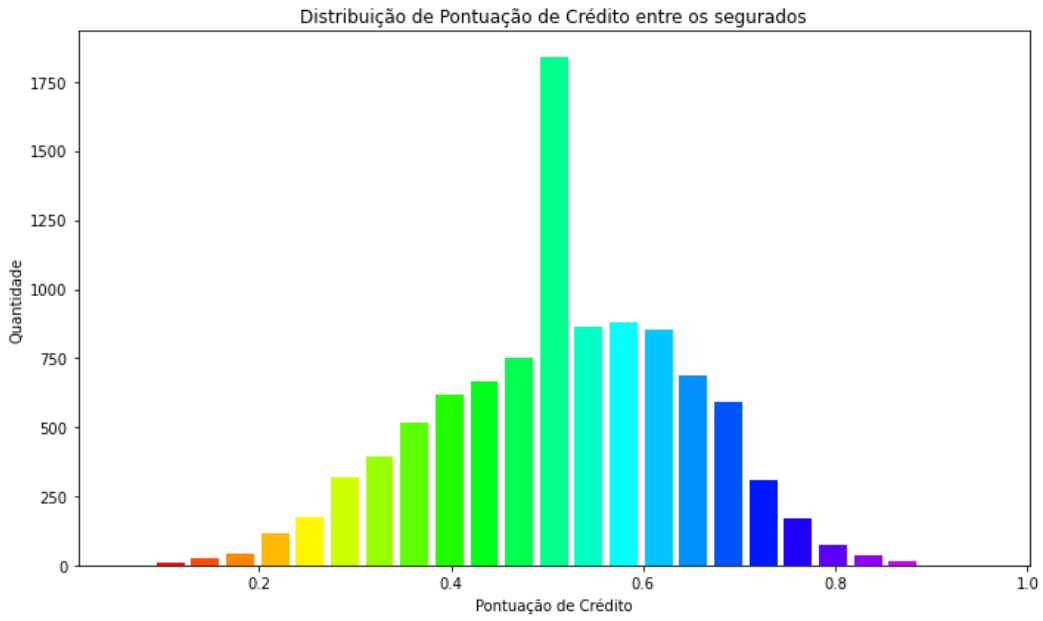
Nessa primeira parte da análise, você deve analisar as 18 primeiras colunas do dataset, deixando de fora a coluna "**outcome**". Você deve analisar como essas

features se comportam individualmente e como elas interagem entre si. Você pode seguir os exemplos abaixo, mas lembre-se de dar personalidade à sua análise indo além dessas propostas:

- Analisar (individualmente) as distribuições de idade, gênero, etnia, educação e renda.
- Analisar a relação entre renda dos segurados com suas pontuações de crédito. Como a distribuição de pontuações se comporta com o aumento de renda?
- Analisar a correlação entre escolaridade, renda e posse do veículo. Existe uma relação efetiva entre essas features?
- Analisar a correlação entre gênero e histórico de infrações de trânsito e acidentes. Homens costumam cometer mais infrações que mulheres?

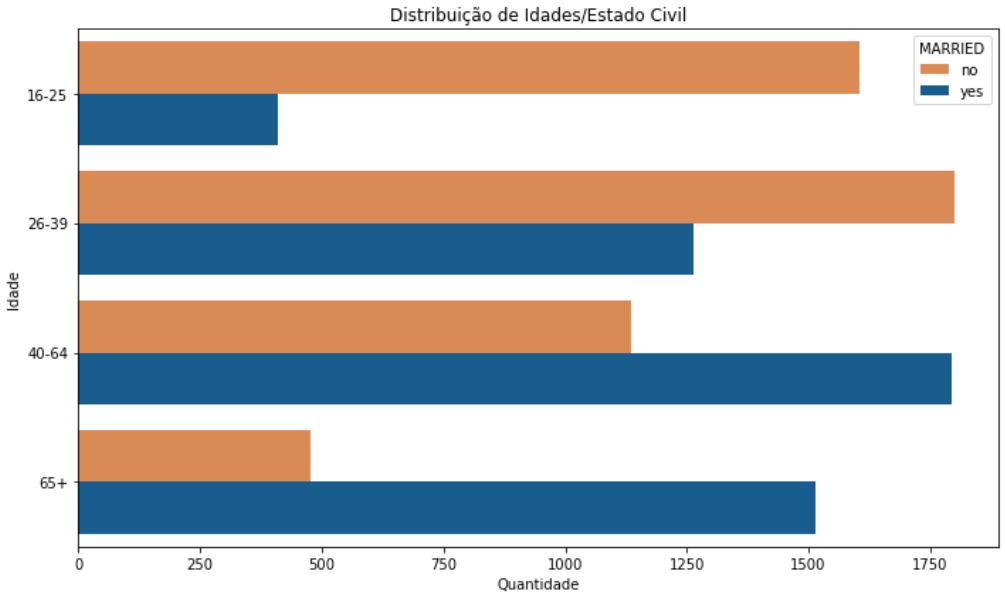
Obs: Os itens acima servem para guiar sua análise, mas ela não deve ser apenas uma série de respostas a estas perguntas. Também não é necessário cobrir todas as features em sua análise, você pode focar nas que achar mais interessantes.

Caso tenha dificuldade em pensar em gráficos, tome os a seguir como inspiração:



O histograma acima apresenta as informações de uma única coluna, com os valores para o CREDIT_SCORE no eixo x e as contagens de indivíduos com essa pontuação no eixo y.

O gráfico de barras abaixo, por sua vez, correlaciona 2 colunas, AGE e MARRIED, apresentando a distribuição de indivíduos entre as faixas de idade, bem como a distribuição de estado civil para cada uma dessas faixas.



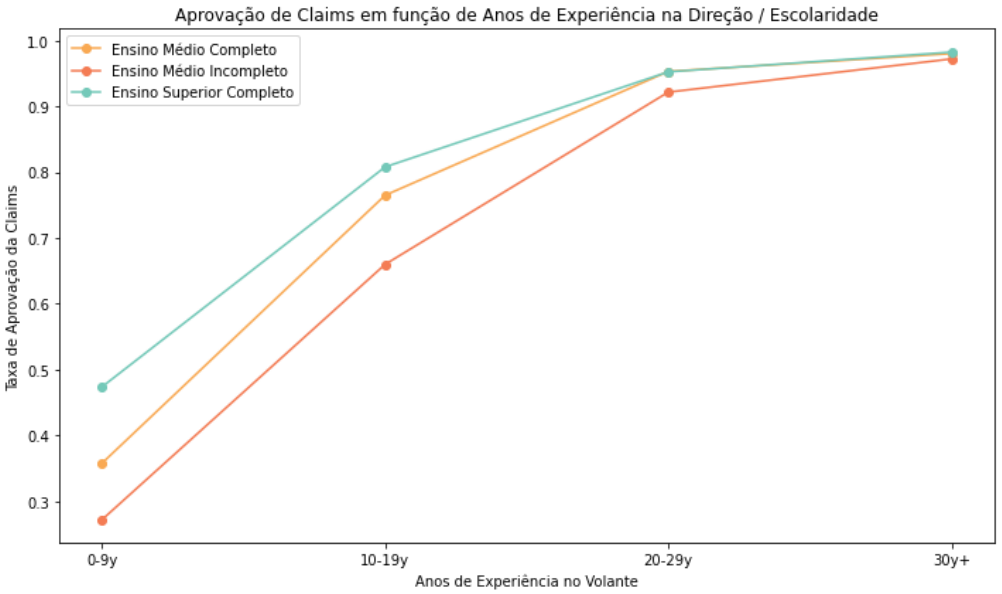
Análise do Outcome (20 pontos)

Agora que você conhece bem os dados com os quais está trabalhando, é hora de avançar para a parte principal de nossa análise. O foco agora deverá ser na coluna "**outcome**": você deverá analisar como as diferentes features do dataset influenciam na aprovação de "insurance claims ".

Novamente vamos passar alguns exemplos, mas recomendamos que você dê personalidade à sua análise indo além dessas propostas.

- Analisar como idade do motorista e tempo de experiência no volante influenciam na aprovação ou não de claims.
- Analisar como a taxa de aprovação se comporta conforme aumenta a pontuação de crédito dos segurados.
- Analisar como a idade do veículo e a quilometragem anual impactam na taxa de aprovação dos seguros.
- Analisar como a posse ou não de veículo pode determinar ou não o resultados de insurance claims.

O gráfico de linha abaixo apresenta a taxa de aprovação de insurance claims em função de 2 features distintas, EDUCATION e DRIVING_EXPERIENCE.



Note que uma análise multidimensional pode muitas vezes trazer uma riqueza adicional a seus gráficos. Busque sempre **enriquecer a sua análise**.

Conclusões (10 pontos)

Um cientista de dados, depois de promover as suas análises, deve chegar a conclusões acerca do assunto que tanto buscou entender. Você, portanto, deverá, da forma como desejar, apresentar, em uma conclusão, **os principais pontos a que conseguiu chegar**. Para isso, você pode resumir as análises que obteve e tentar relacioná-las.

Por que será que os resultados obtidos foram esses? Tente chegar a uma justificativa, seja por meio de suas análises, sugestões ou pesquisas. Sinta-se à vontade para concluir como achar melhor.

Referências (5 pontos)

Como constatado anteriormente, um dos principais pontos que queremos avaliar é o seu aprendizado e evolução ao longo da realização do case. Dessa forma, é importante

que registre suas referências à medida que avança em sua análise. A seguir segue um exemplo de como suas referências devem ser estruturadas.

Exemplo:

1. **Matplotlib** (Youtube) - Matplotlib Tutorial (Part 1): Creating and Customizing Our First Plots - YouTube
2. **Visualização de Dados** (Turing Talks) - Data Science | Visualização de Dados | by Fernando Matsumoto | Turing Talks |
3. **Como dropar colunas de um dataset** (Pandas) - pandas.DataFrame.drop — pandas 1.5.3 documentation (pydata.org)

OBS: Note que não é necessário incluir todas as referências utilizadas. Se atenha àquelas que você julgar de maior contribuição para o seu aprendizado.

Desafio: Modelo de Classificação - Bônus

Atenção: esse case tem dois desafios: um de *análise de dados e IA* e outro de *lógica de programação e algoritmos*. Lembre-se de que:

- Os desafios são **opcionais** e valem uma pontuação extra (25 ptos);
- Os desafios não podem elevar a sua nota no case técnico acima dos 100 pontos; e
- Se optar por fazê-los, você deve escolher **no máximo um deles**. Em outras palavras, **não entregue ambos os desafios**.

Após um cientista de dados ter uma noção consistente de seus dados, ele pode buscar aplicar um modelo de inteligência artificial sobre a base de dados para buscar predições. O modelo que vamos utilizar neste problema é a **regressão logística**, que é considerado um modelo simples de classificação. Nesta última etapa da análise, você deve aplicar este algoritmo nos dados anteriores, de forma a tentar prever se a claim de um segurado será aprovada ou não, com base nas suas características pessoais e as do veículo. Sinta-se livre para consultar os diversos recursos disponíveis na internet sobre o assunto. Para isso, recomendamos as seguintes informações:

- O nosso post sobre regressão logística no medium: https://medium.com/turing-talks/turing-talks-14-modelo-de-predi%C3%A7%C3%A3o-regress%C3%A3o-log%C3%ADstica-7b70a9098e43

- Não é necessário implementar a regressão logística do zero, utilize uma biblioteca como scikit-learn no python.
- Utilize métricas de avaliação para saber se seu modelo de regressão logística está razoável ou não.
- Sinta-se livre para tentar diferentes abordagens: fica a seu critério, por exemplo, usar uma única *feature*, várias ou todas.
- Uma parte importante será o tratamento dos dados categóricos, que não podem ser passados para um modelo como categorias. Para isso dê uma procurada em one-hot encoding e get dummies.

O guia aqui é alcançar uma regressão logística com uma acurácia maior que 0,7. Para saber mais sobre a acurácia e métricas de avaliação leia este texto: Como Avaliar Seu Modelo de Classificação | by Gustavo Korzune Gurgel | Turing Talks | Medium.

Parte 3 - Turing Talks (Opcional)

Pontuação Total: 10 pontos

O Turing USP possui três pilares principais que promovem a sustentação de todas as atividades desenvolvidas interna e externamente pelo grupo, são eles: Estudar, Aplicar e Disseminar conhecimentos de inteligência artificial e ciência de dados.

O pilar de disseminação de conhecimento é de extrema importância para o objetivo do grupo de se tornar uma referência no tema ao redor de São Paulo e do Brasil. Para tal, temos diversos meios que são importantes: fazemos workshops, rodas de conversas e outros eventos, buscamos a publicação de projetos desenvolvidos internamente, como, por exemplo, a publicação de um artigo sobre o projeto do Fernando Pessoa na revista Superinteressante.

Além disso, gravamos aulas e workshops para serem disponibilizados no canal do youtube do grupo. Em especial, uma das atividades mais importantes para o pilar de disseminação é a publicação dos Turing Talks, na plataforma Medium. Estes textos possuem um alcance muito grande por conta do caráter específico de divulgação científica da plataforma. Devido ao seu fácil acesso, linguagem descontraída, abordagem mais direta, periodicidade e por muitas vezes apresentar de forma consistente um tema dificilmente encontrado com a mesma simplicidade na língua portuguesa, o Turing Talks é um dos projetos recorrentes mais importantes do grupo.

Neste contexto da importância, você deve redigir um **Turing Talks** sobre **um dos temas listados abaixo** que aborda conceitos de inteligência artificial. Como referência para esse trabalho você tem a sua disposição um grande acervo em nossa página no Medium: Turing Talks, bem como qualquer outra fonte na internet. O objetivo é escrever um texto que seja informativo e conciso, não queremos textos puramente expositivos como uma página do wikipédia.

O texto não precisa seguir a norma culta, mas imagine que será um texto publicado com seu nome nele, para milhares de pessoas que buscam aquele conhecimento vão ler e julgar o conteúdo, por isso é esperado que não existam erros ortográficos e gramaticais graves.

Temas:

- Métricas para avaliação de modelos de machine learning
- Árvores de Decisão e Random Forest
- Feature Engineering (one-hot encoding, normalização de dados, etc)

Limite de tamanho: 4000 caracteres, sem contar espaços, referências e títulos. Recomendamos a fonte Arial ou Times New Roman 12.

Critérios de avaliação:

- Respeitar o limite máximo de caracteres.
- Texto coeso, com começo, meio e fim.
- Explicitação das referências usadas.
- O título e a estrutura estética em um Turing Talks são importantes, atente-se às imagens que você usa para construir o seu texto.
- Originalidade e criatividade do texto.