sbbd:01

Capítulo

1

Introdução à API da OpenAI

Alexandre Donizeti Alves

Abstract

OpenAI is a company focused on research and development of artificial intelligence, dedicated to creating advanced systems with wide application in several areas. One of its main tools is the API, which allows you to access its natural language models through a friendly and easy-to-use interface. The API is based on some of the most advanced natural language models in the market, including GPT-3, currently considered one of the most sophisticated models available. With the OpenAI API, developers and companies can integrate advanced natural language processing technologies into their projects, including sentiment analysis, text summarization, text generation, and much more. In this chapter, a detailed overview of the OpenAI API is presented, using practical examples in Python and JavaScript (Node.js).

Resumo

A OpenAI é uma empresa focada em pesquisa e desenvolvimento de inteligência artificial, dedicada na criação de sistemas avançados com ampla aplicação em diversas áreas. Uma de suas principais ferramentas é a API, que permite acessar seus modelos de linguagem natural por meio de uma interface amigável e fácil de usar. A API é baseada em alguns dos modelos de linguagem natural mais avançados do mercado, incluindo o GPT-3, atualmente considerado um dos modelos mais sofisticados disponíveis. Com a API da OpenAI, desenvolvedores e empresas podem integrar tecnologias avançadas de processamento de linguagem natural em seus projetos, incluindo análise de sentimentos, sumarização de textos, geração de texto e muito mais. Neste capítulo, é apresentada uma visão geral detalhada da API da OpenAI, utilizando exemplos práticos em Python e JavaScript (Node.js).

1.1. Introdução

A *OpenAI*[1] é uma organização privada de pesquisa em Inteligência Artificial (IA). Foi fundada em dezembro de 2015 e desde sua concepção, a organização focou em criar e promover soluções que sejam seguras e alinhadas com interesses humanos, permitindo enfrentar os desafios que a IA avançada pode apresentar à sociedade. Em setembro de 2020, foi consolidada uma parceria[2] estratégica entre a *OpenAI* e a *Microsoft*. Neste acordo, a *Microsoft* tornou-se a provedora exclusiva de soluções em nuvem para a *OpenAI*, utilizando sua plataforma *Azure*[3] para hospedar e disponibilizar serviços de IA desenvolvidos pela *OpenAI*. Essa colaboração permite à *Microsoft* integrar recursos da *OpenAI*, como o *ChatGPT* (combina "*chat*", referindo-se à sua funcionalidade de *chatbot*, e "*GPT*", que significa *Generative Pre-trained Transformer)*, em seus produtos e serviços. O *ChatGPT* foi lançado em junho de 2020 e, no final de novembro de 2022, tornou-se disponível para o público em geral. Esse lançamento despertou um interesse público gigantesco e em diferentes áreas [Agathokleous et al. 2023, Cheng et al. 2023a, Cheng et al. 2023b]. Contudo, a ferramenta gerou muitas controvérsias, como por exemplo, pelo seu uso na escrita de artigos científicos [Brainard 2023]. A *OpenAI* também enfrenta problemas como a falta de transparência [van Dis et al. 2023] e a exclusão de usuários de determinados países, como por exemplo, China, Rússia e Ucrânia [Wang 2023]. Além disso, em países como a Itália, o acesso está sendo limitado [Niszczota and Rybicka 2023].

Uma das ferramentas desenvolvidas pela *OpenAI* é sua *API (Application Programming Interface*), projetada para facilitar o acesso a uma série de recursos avançados de linguagem natural. Esta interface oferece aos desenvolvedores uma forma eficiente de integrar capacidades de linguagem natural em seus projetos, proporcionando recursos como análise de sentimentos, sumarização de textos, geração de conteúdo e muito mais. A *API*[4] se fundamenta em modelos de linguagem natural avançados, como o *GPT-3*. Há também o modelo *GPT-3.5*, que representa uma evolução significativa na modelagem de linguagem natural. Utilizando a arquitetura *Transformer*, o modelo foi treinado com um conjunto extenso de dados, possibilitando capturar uma gama mais ampla de nuances e contextos linguísticos. Esse modelo é adaptado para entender e gerar respostas para *prompts* complexos, com diferentes estilos e formatos textuais. Além disso, incorpora um aprendizado acumulado dos modelos anteriores, aprimorando sua eficácia em tarefas de Processamento de Linguagem Natural (PLN) e reduzindo algumas das limitações encontradas em modelos anteriores. O modelo mais recente é o *GPT-4*. Entretanto, esse modelo ainda não foi liberado para acesso e utilização por meio da *API* para o público em geral [Sanderson 2023]. Atualmente, a *OpenAI* disponibiliza sua *API* para vários países, regiões e territórios[5].

---

[1] https://openai.com/about

[2] https://openai.com/blog/microsoft-invests-in-and-partners-with-openai

[3] https://azure.microsoft.com/pt-br/products/cognitive-services/openai-service/

[4] https://platform.openai.com/docs/api-reference

[5] https://platform.openai.com/docs/supported-countries

Este minicurso tem como objetivo apresentar uma visão geral detalhada da *API* da *OpenAI*, utilizando exemplos práticos em *Python* e *JavaScript* (*Node.js*). Com isso, será possível proporcionar um entendimento das principais funcionalidades da *API*, evidenciando o seu potencial em variados contextos e sua aplicação em cenários reais.

1.2. Configuração da *API*

Esta seção apresenta os principais requisitos para utilizar a *API* da *OpenAI*, enfatizando que os desenvolvedores devem criar uma conta na plataforma e obter uma chave da *API*. Além disso, são abordadas as linguagens de programação suportadas pela *API*, com destaque para *Python* e *JavaScript* (*Node.js*), além de outras com suporte da comunidade. Neste minicurso, são abordadas especificamente as linguagens de programação *Python* e *JavaScript* (*Node.js*), fornecendo instruções detalhadas para instalar e configurar o ambiente de desenvolvimento, incluindo as bibliotecas e ferramentas necessárias para utilizar a *API*. Isso inclui a instalação de pacotes e outras configurações específicas para cada linguagem de programação.

1.2.1. Principais requisitos

Para utilizar a *API* da *OpenAI*, é necessário atender a alguns requisitos essenciais que garantem uma experiência segura e eficiente para todos os usuários. A seguir, estão listados os principais requisitos a serem cumpridos pelos desenvolvedores:

1. **Criar uma conta na plataforma da *OpenAI***: Antes de começar a utilizar a *API*, os desenvolvedores devem criar uma conta na plataforma da *OpenAI*. A conta é essencial para obter acesso aos recursos da *API* e gerenciar suas configurações.

2. **Obter uma chave da *API***: Após criar a conta, os desenvolvedores precisam obter uma chave da *API*. Essa chave é um identificador único que concede acesso à *API* e autentica as solicitações feitas pelos desenvolvedores. A chave desempenha um papel fundamental na garantia da segurança e rastreabilidade da utilização da *API*.

3. **Crédito para utilizar a *API***: Atualmente, ao se cadastrar na plataforma, a *OpenAI* disponibiliza 5 dólares em crédito gratuito, que podem ser utilizados ao longo de três meses. Isso permite aos desenvolvedores explorarem a *API* sem custo inicial. Após consumir esse crédito ou ao final do período de três meses, o desenvolvedor pode optar por pagar conforme a utilização, podendo escolher o modelo de linguagem mais adequado e também financeiramente mais acessível para o seu projeto. Essa flexibilidade permite uma utilização mais controlada e personalizada da *API*, atendendo às necessidades específicas de cada desenvolvedor. A *OpenAI* também impõe limites no número de solicitações que um usuário pode fazer em um determinado período de tempo, garantindo assim que os serviços permaneçam escaláveis e de alta qualidade.

1.2.2. Criando uma conta na plataforma da *OpenAI*

Para criar uma conta na *OpenAI*, siga os passos abaixo:

1. Inicialmente, acesse o *site* oficial da plataforma[6].
2. No canto superior direito, localize e clique no botão "*Sign up*" (Registrar-se).
3. Feito isso, há o redirecionamento para a página de criação de conta. Nesta etapa, há a opção de criar uma nova conta fornecendo um endereço de *e-mail*. Outra opção é fazer a associação à uma conta de *e-mail* (*Gmail,* por exemplo*)* já existente.
4. Em seguida, é enviado um *e-mail* de verificação com um *link* para concluir o cadastro na plataforma.
5. Para completar o cadastro, é necessário fornecer o nome, sobrenome e data de nascimento.
6. Será solicitado também um número de celular para envio de um código de verificação. É importante ressaltar que apenas um número de celular pode ser associado à uma conta na plataforma da *OpenAI*.
7. Para finalizar o processo, basta inserir o código de verificação recebido no celular. Dessa forma, a conta estará pronta para ser utilizada.

1.2.3. Obtendo uma chave da *API*

Após concluir o cadastro na plataforma da *OpenAI*, o próximo passo consiste em adquirir uma chave da *API*. Essa etapa requer que o usuário esteja com sua sessão iniciada na plataforma. Uma vez logado, localize a inicial do nome do usuário no canto superior direito do painel da *OpenAI* e clique sobre ela. No menu que se abre, selecione a opção "*View API Keys*" (Exibir chaves da *API*). Alternativamente, essa funcionalidade pode ser acessada diretamente através da *URL* (*Uniform Resource Locator*) correspondente[7].

Para gerar uma chave nova, clique no botão *"+ Create new secret key*" (Criar nova chave secreta). Isso abrirá uma nova janela e será possível informar um nome para a chave (opcional). Em seguida, clique em "*Create secret key*". Feito isso, uma nova chave será criada. Certifique-se de armazenar essa chave em um local seguro e de fácil acesso. É importante observar que, por razões de segurança, após a criação, não é possível visualizar a chave novamente por meio da conta na *OpenAI*. Caso ocorra a perda da chave, será necessário gerar uma nova.

1.2.4. Linguagens de programação suportadas pela *API*

A *API* da *OpenAI* é projetada para ser compatível com várias linguagens de programação, permitindo que desenvolvedores integrem facilmente os recursos de linguagem natural em suas aplicações. Embora a *API* não seja restrita a uma linguagem de programação específica, ela pode ser utilizada em ambientes que possam fazer solicitações *HTTP* (*Hypertext Transfer Protocol*) e processar respostas *JSON*

[6] https://openai.com

[7] https://platform.openai.com/account/api-keys

(*JavaScript Object Notation*). Isso inclui linguagens como *Python*, *JavaScript*, *Java*, *Ruby* e outras, proporcionando aos desenvolvedores a liberdade de escolher a linguagem que melhor se adapte às suas necessidades. É importante destacar que a *OpenAI* dá suporte oficialmente para as linguagens *Python* e *JavaScript* (*Node.js*). No entanto, há outras linguagens de programação com suporte da comunidade[8]. A seguir são apresentadas instruções detalhadas para a instalação e configuração do ambiente de desenvolvimento nas linguagens de programação *Python* e *JavaScript* (*Node.js*).

1.2.5. Configurando o ambiente para *Python*

Para desenvolver em *Python* foi utilizado o *Google Colab*[9], também conhecido como *Colaboratory*, uma plataforma *online* gratuita oferecida pelo *Google* que permite a criação e execução de código *Python* em um ambiente baseado na nuvem, ou seja, sem a necessidade de configuração de um ambiente de desenvolvimento local. O *Colab* oferece suporte a *notebooks* do *Jupyter*, permitindo criar documentos interativos que misturam código executável, texto formatado, imagens, vídeos, tabelas e gráficos. Os *notebooks* do *Colab* são salvos diretamente no *Google Drive*, sendo possível compartilhá-los com outros usuários. Neste minicurso, foi utilizada a versão 3.10.12 do *Python* disponível no *Colab*.

O primeiro passo no *Google Colab*[10] é instalar o pacote *Python* da *OpenAI* para utilizar as funcionalidades da *API*. A Listagem 1.1 mostra o código para fazer essa instalação.

```
print(f"Instalando a biblioteca da API da OpenAI...")

!pip install openai -q

print("API da OpenAI instalada!")
```

```
Instalando a biblioteca da API da OpenAI...
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 73.6/73.6 kB 1.3 MB/s eta 0:00:00
API da OpenAI instalada!
```

Listagem 1.1. Instalação do pacote *Python* da *API* da *OpenAI*.

Após concluir a instalação do pacote da *OpenAI*, é necessário configurar uma chave para que o pacote possa acessar a *API*. O trecho de código apresentado na Listagem 1.2 mostra a configuração da chave diretamente no código, substituindo `YOUR_API_KEY` pelo *token* gerado durante a criação da chave da *API* na plataforma da *OpenAI*. No entanto, é importante destacar que esse procedimento no *Google Colab* não é seguro, pois trata-se de um ambiente em que o código pode ser compartilhado ou exposto publicamente. Isso ocorre porque a chave da *API* é inserida diretamente no código-fonte, o que a torna facilmente identificável por qualquer usuário com acesso ao código. A variável `openai.api_key` faz parte do pacote `openai` e tem a função de armazenar a chave utilizada para autenticar as chamadas à *API*.

[8] https://platform.openai.com/docs/libraries

[9] https://colab.research.google.com

[10] https://colab.research.google.com/?utm_source=scs-index

```python
import openai

# definir a chave da API
openai.api_key = 'YOUR_API_KEY'
```

Listagem 1.2. Configuração da chave de acesso da *API* diretamente no código.

Para manter a chave da *API* segura, é recomendável utilizar variáveis de ambiente ou um arquivo de configuração separado do código-fonte. Dessa forma, é possível carregar a chave da *API* de forma segura sem expô-la no código diretamente. Por exemplo, é possível definir a chave da *API* como uma variável de ambiente e acessá-la no código, conforme mostrado na Listagem 1.3. O pacote `os` é um módulo em *Python* que permite interagir com o sistema operacional, incluindo a obtenção de variáveis de ambiente. Isso pode ser feito utilizando a função `gentenv`, que permite acessar o valor de um variável de ambiente específica. Neste cenário exemplificado, é necessário definir a chave da *API* como uma variável de ambiente denominada `OPENAI_API_KEY`, com um valor correspondente à chave da *API* obtida na plataforma da *OpenAI*. Essa prática garante que as informações sensíveis permaneçam protegidas e que a segurança das credenciais seja mantida intacta.

```python
import os
import openai

openai.api_key = os.getenv("OPENAI_API_KEY")
```

Listagem 1.3. Configuração da chave de acesso da *API* a partir de uma variável de ambiente.

A utilização de um arquivo de configuração (texto) também é uma prática recomendada para manter informações sensíveis, como chaves de *API*, seguras e separadas do código-fonte. Um exemplo prático consiste em criar um arquivo de texto contendo apenas a chave da *API* gerada na plataforma da *OpenAI*. Na Listagem 1.4 é apresentado um trecho de código que lê o conteúdo de um arquivo de texto, denominado `openai_chave_minicurso`, e armazena o valor lido como a chave de autenticação para acessar a *API*.

```python
import openai

# ler o conteúdo do arquivo de texto
with open('/content/openai_chave_minicurso.txt', 'r') as file:
   chave_api = file.read()

# definir a chave da API
openai.api_key = chave_api
```

Listagem 1.4. Configuração da chave de acesso da *API* a partir de um arquivo de texto.

Outra abordagem possível é realizar o *upload* de um arquivo de texto, como exemplificado no trecho de código da Listagem 1.5. Após o *upload*, é feita a leitura do conteúdo do arquivo de texto e o valor obtido é utilizado para configurar a chave de acesso à *API*.

Uma vez configurada a chave de acesso, é importante verificar se a configuração foi realizada corretamente. A Listagem 1.6 mostra um trecho de código simples para começar a explorar as funcionalidades oferecidas pela *API*. Nesse código, a *API* é empregada para gerar uma "continuação" (*completion*[11]) a partir do *prompt* "`O Brasil é um país`" e utilizando o modelo `text-davinci-003`. A saída para a execução desse código foi "`localizado na América do Sul. É formado pelos 26`". Note que a "continuação" retornada depende do valor *default* do número total de *tokens* na *API*.

```
import openai
from google.colab import files

# fazer upload do arquivo de texto
upload_arquivo = files.upload()

# obter o nome do arquivo
nome_arquivo = list(upload_arquivo.keys())[0]

# ler o conteúdo do arquivo
with open(nome_arquivo, 'r') as file:
   chave_api = file.read()

# definir a chave da API
openai.api_key = chave_api
```

```
Browse... openai_chave_minicurso.txt
openai_chave_minicurso.txt(text/plain) - 51 bytes, last modified: n/a - 100% done
Saving openai_chave_minicurso.txt to openai_chave_minicurso.txt
```

Listagem 1.5. Configuração da chave de acesso da *API* a partir do *upload* de um arquivo de texto.

Os parâmetros `model` e `prompt` são obrigatórios. O parâmetro `model` especifica qual modelo ou mecanismo de geração de texto deve ser utilizado para produzir a resposta. A *OpenAI* disponibiliza vários modelos com diferentes capacidades e preços. É importante estar ciente de que a *OpenAI* regularmente introduz novos modelos, tornando os modelos antigos obsoletos[12]. Neste minicurso, o modelo `text-davinci-003` será utilizado com frequência. Trata-se de um modelo que é otimizado para tarefas de geração de texto. O parâmetro `prompt` é a entrada inicial que deve ser fornecida ao modelo para orientar a geração de texto. Essa entrada fornece contexto e diretrizes para a criação da resposta desejada pelo usuário. Portanto, quando se invoca a função `openai.Completion.create()`, está-se solicitando à *API* que utilize o modelo especificado e o *prompt* fornecido para gerar uma resposta de texto correspondente.

```
resposta = openai.Completion.create(
   model = "text-davinci-003",
   prompt = "O Brasil é um país"
)

print(resposta.choices[0].text)
```

```
localizado na América do Sul. É formado pelos 26
```

Listagem 1.6. Trecho de código em Python para gerar uma resposta de texto.

[11] https://platform.openai.com/docs/api-reference/completions

[12] https://platform.openai.com/docs/deprecations

É importante destacar que toda vez que esse código for executado será gerada uma resposta diferente, devido à natureza generativa do modelo. Adicionalmente, é importante estar ciente da restrição inerente ao modelo `text-davinci-003`, que permite somente três solicitações por minuto. Caso esse limite seja atingido, é necessário aguardar 20 segundos antes de executar novamente o código. É possível aumentar esse limite adicionando um método de pagamento[13] à conta do usuário.

O número total de *tokens* processados em uma única solicitação, incluindo o *prompt* e a resposta, deve respeitar o limite máximo de *tokens* do modelo. Em grande parte dos casos, esse limite é de 4.096 *tokens* ou cerca de 3.000 palavras. De modo geral, um *token* equivale a aproximadamente 4 caracteres ou 0,75 palavras em inglês. Os preços[14] seguem uma lógica de pagamento por uso, calculado a cada 1.000 *tokens* processados.

1.2.6. Configurando o ambiente para *JavaScript* (*Node.js*)

JavaScript[15] (normalmente abreviado para *JS*) é uma linguagem de programação de alto nível, dinâmica, leve, interpretada e baseada em objetos. Foi originalmente criada para ser executada em navegadores, permitindo a criação de páginas *Web* interativas e dinâmicas. Com o tempo, no entanto, seu escopo de aplicação se expandiu, sendo utilizada também no desenvolvimento de aplicativos de servidor, aplicativos móveis e muito mais. *Node.js*[16] é um ambiente de tempo de execução de código aberto que permite executar *JavaScript* no lado do servidor. Uma opção para desenvolver em *Node.js* é utilizar o *Replit*, uma plataforma *online* que oferece um ambiente de desenvolvimento integrado baseado na nuvem. A plataforma permite que desenvolvedores escrevam, executem e colaborem em códigos de várias linguagens de programação diretamente no navegador, sem a necessidade de configurações complicadas ou instalações locais.

Para configurar a *API* da *OpenAI* no *Replit* para utilização com *Node.js*, é necessário acessar a página do *Replit*[17] e criar um novo projeto clicando em "+ *Create Repl*". Uma janela aparecerá, permitindo selecionar a opção "*Node.js*". Nesse ponto, é necessário informar um nome para o projeto (por exemplo, `TesteAPIOpenAI`), e então clicar no botão "+ *Create Repl*" para finalizar. Feito isso, um ambiente de desenvolvimento estará pronto para ser utilizado, conforme ilustrado na Figura 1.1.

No arquivo `package.json`, é necessário verificar se o pacote `openai` está listado como uma dependência. Caso contrário, será necessário instalá-lo. Isso pode ser feito no terminal do *Replit* utilizando o seguinte comando: `npm install openai`. Para configurar a chave de acesso é possível utilizar a ferramenta "*Secrets*", localizada no painel "*Tools*" no *Replit*. Após clicar no ícone correspondente, uma janela é aberta. Clique em "+ *New Secret*" para adicionar uma nova chave de acesso. Para isso basta informar um nome para a chave (por exemplo, `OPENAI_API_KEY`) e o seu respectivo

[13] https://platform.openai.com/account/billing

[14] https://openai.com/pricing

[15] https://developer.mozilla.org/en-US/docs/Web/JavaScript

[16] https://nodejs.org

[17] https://replit.com

valor. Para finalizar, clique no botão “*Save*”. Dessa forma, a chave da *API* ficará segura em “*Secrets*” e não será exposta diretamente no código-fonte.

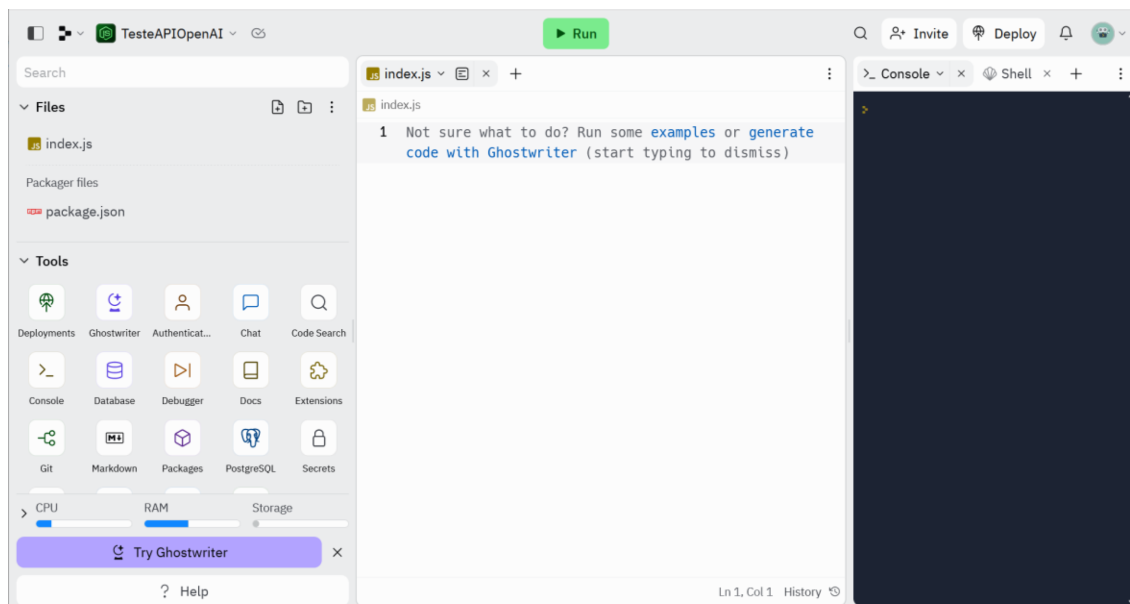

Figura 1.1. Ambiente de desenvolvimento *Node.js* na plataforma *Replit*.

Após a configuração da chave de acesso, é possível explorar as funções disponibilizadas pelo pacote `openai`. Por exemplo, é possível utilizar a função `openai.Completion.create()` para gerar texto com base em um *prompt*, assim como foi mostrado anteriormente em *Python*. No arquivo *JavaScript* principal do projeto (`index.js`), é necessário importar o pacote `openai` e utilizar a função `process.env` para acessar o valor da chave da *API* definida em “*Secrets*”. Para testar o código, basta clicar no botão “*Run*” no *Replit*.

A Figura 1.2 ilustra um trecho de código para gerar uma “continuação” a partir do *prompt* “`O Brasil é um país`” e utilizando o modelo `text-davinci-003`. A saída para a execução desse código foi “`localizado na América do Sul, tendo a maior part`”. De maneira bem resumida, o código importa as classes `Configuration` e `OpenAIApi` do pacote `openai`, configura a chave da *API* através de uma instância de `Configuration` utilizando uma variável de ambiente. Em seguida, é criada uma instância de `OpenAIApi` com essa configuração. Também é definida uma função assíncrona denominada, por exemplo, `executarPrompt()`. Dentro dessa função, a *API* é utilizada para gerar texto em resposta ao *prompt* de acordo com o modelo definido. O resultado é impresso no painel “*Console*”. Por fim, a função `executarPrompt()` é chamada para executar o código.

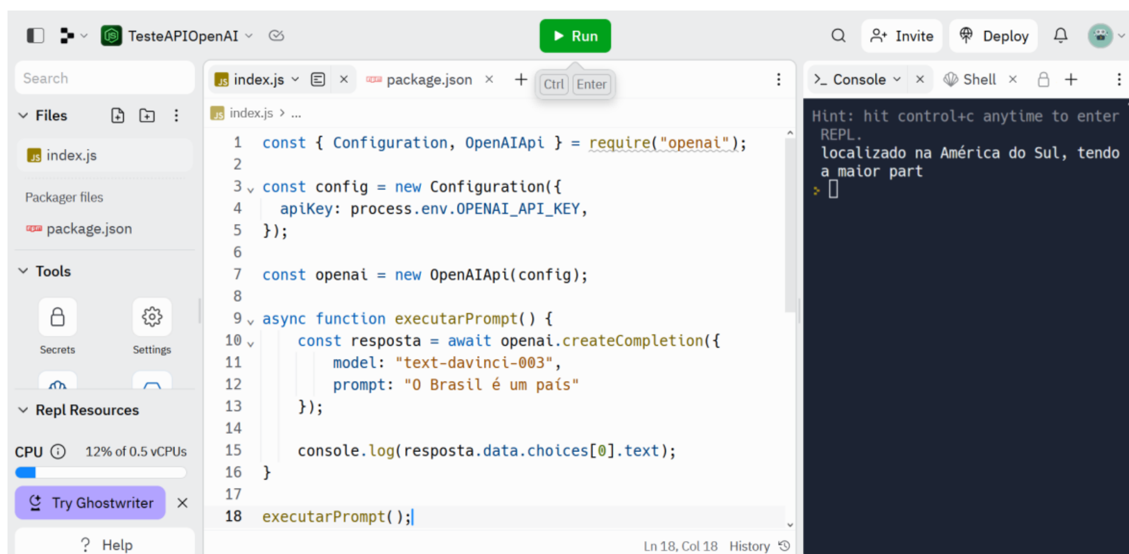

Figura 1.2. Trecho de código em *JavaScript* (*Node.js*) para gerar uma resposta de texto.

1.3. Como utilizar a API

Existem várias formas de utilizar a *API* da *OpenAI*, dependendo das necessidades de cada projeto. A *OpenAI* fornece bibliotecas oficiais para as linguagens *Python* e *JavaScript* (*Node.js*) que facilitam a utilização da *API*. Com essas bibliotecas, é possível fazer chamadas à API de forma simplificada e sem a necessidade de lidar diretamente com o protocolo *HTTP*. Essa abordagem será apresentada brevemente na subseção 1.3.4 utilizando *Python* e na subseção 1.3.5 utilizando *JavaScript* (*Node.js*). Além disso, na seção 1.4 serão apresentados vários exemplos práticos em diferentes contextos. Outra forma de utilizar a *API* é fazendo requisições *HTTP* diretas utilizando bibliotecas, como `requests` para *Python* ou `axios` para *Node.js*. Essa abordagem será apresentada nesta seção e pode ser mais flexível em alguns casos, permitindo a personalização de parâmetros e cabeçalhos específicos. Também há a possibilidade de utilizar plataformas de terceiros. Existem diversas plataformas que fornecem interfaces mais amigáveis para a utilização da *API* da *OpenAI*, como por exemplo, *Hugging Face*[18]. A própria *OpenAI* fornece uma interface mais amigável com a ferramenta *OpenAI Playground*[19]. Esta abordagem não será apresentada neste minicurso.

Nesta seção é apresentada uma visão geral dos *endpoints* atualmente disponíveis na *API* da *OpenAI*. A *API* oferece *endpoints* que permitem aos desenvolvedores acessar diferentes modelos de IA para tarefas específicas. Também é apresentado como fazer requisições *HTTP* à *API* e entender as respostas, e como utilizá-las em *Python* e *JavaScript* (*Node.js*). Após enviar uma solicitação à *API*, o desenvolvedor receberá uma resposta que contém os resultados da operação solicitada. Essa resposta pode estar em vários formatos, como *JSON*, texto simples ou outros formatos específicos de cada *endpoint*. É importante que os desenvolvedores entendam como interpretar essas respostas para poder utilizá-las de forma adequada em suas aplicações. Além disso, é

[18] https://huggingface.co/models

[19] https://platform.openai.com/playground

apresentado brevemente como utilizar a biblioteca da *API* da *OpenAI* em *Python* e *JavaScript* (*Node.js*).

1.3.1. Visão geral dos *endpoints*

Basicamente, um *endpoint* é um ponto de entrada para interagir com os serviços e recursos disponibilizados pela *API* da *OpenAI*. A *API* disponibiliza diversos *endpoints*: *completions*[20] (refere-se a "completar" ou "continuar" um texto), *chat completions*[21], geração de imagens[22], fala para texto (transcrições e traduções)[23] e moderações[24]. Nesta subseção, é apresentada uma visão geral dos *endpoints* "*completions*" e "*chat completions*". Na seção 1.4 também são apresentados alguns exemplos que utilizam o *endpoint* para geração de imagens. Por limitações de espaço, os outros *endpoints* não serão apresentados neste minicurso.

O principal *endpoint* da *API* ainda é o "*completions*", que fornece uma interface simples e flexível o suficiente para realizar praticamente qualquer tarefa de PLN, incluindo geração de conteúdo, sumarização, análise de sentimentos e muito mais. Esse *endpoint* é geralmente utilizado para gerar "continuação" de texto com base no *prompt* fornecido[25].

Além disso, é possível utilizar o *endpoint* "*completions*" com diferentes modelos. Os modelos *GPT* da *OpenAI* foram treinados para entender linguagem natural e código. Esses modelos fornecem saídas de texto em resposta às suas entradas. As entradas para esses modelos também são chamadas de *prompts*. Projetar um *prompt* é essencialmente como se "programa" um modelo *GPT*, geralmente fornecendo instruções ou alguns exemplos de como concluir uma tarefa com êxito. Os modelos mais recentes, `gpt-4` e `gpt-3.5-turbo`, são acessados por meio do *endpoint* "*chat completions*". Atualmente, apenas os modelos legados mais antigos estão disponíveis por meio do *endpoint* "*completions*"[26]. Esse *endpoint* recebeu sua atualização final em julho de 2023 e será encerrado em 4 de janeiro de 2024.

O modelo `text-davinci-003`, disponível no *endpoint* "*completions*", é um dos modelos de linguagem mais poderosos e versáteis disponíveis na plataforma da *OpenAI*. Ele é parte da família de modelos *GPT* e é treinado com uma arquitetura *Transformer* de várias camadas, com 175 bilhões de parâmetros. Esse modelo é especialmente adequado para tarefas de linguagem natural que exigem compreensão e geração de texto altamente avançadas. É treinado em um conjunto de dados extremamente grande e diversificado, que inclui textos em vários idiomas e domínios, como notícias, livros, artigos científicos, conversas cotidianas e muito mais.

[20] https://platform.openai.com/docs/guides/gpt/completions-api

[21] https://platform.openai.com/docs/guides/gpt/chat-completions-api

[22] https://platform.openai.com/docs/guides/images/image-generation

[23] https://platform.openai.com/docs/guides/speech-to-text/speech-to-text

[24] https://platform.openai.com/docs/guides/moderation/moderation

[25] https://platform.openai.com/docs/quickstart/introduction

[26] https://platform.openai.com/docs/guides/gpt/gpt-models

A Listagem 1.7 mostra um trecho de código em *Python* para listar todos os modelos disponíveis na *API* da *OpenAI*. A saída desse trecho de código lista o número de modelos disponíveis na *API* (53, atualmente) e o nome de cada um dos modelos (foram exibidos apenas 4 por limitações de espaço). É importante lembrar que a ordem dos nomes dos modelos pode mudar toda vez que esse código for executado.

```
# obter a lista de modelos
modelos = openai.Model.list()

# imprimir total de modelos
print(len(modelos['data']))

# imprimir os nomes dos modelos
for modelo in modelos['data']:
   print(modelo['id'])
```

```
53
text-davinci-edit-001
babbage-code-search-code
text-similarity-babbage-001
code-davinci-edit-001
```

Listagem 1.7. Trecho de código em *Python* para listar todos os modelos disponíveis na *API*.

Outro *endpoint* é o "*chat completions*"[27], que recebe uma lista de mensagens como entrada e retorna uma mensagem gerada pelo modelo como saída. Embora o formato de *chat* seja projetado para facilitar as conversas em vários turnos (*multi-turn*), é igualmente útil para tarefas de turno único (*single-turn*), ou seja, sem nenhuma conversa. O *endpoint* "*chat completions*" foi projetado para tornar mais fácil ter conversas interativas e dinâmicas com o modelo. Em vez de enviar apenas um *prompt* de texto simples como no *endpoint* "*completions*", agora é possível enviar diversas mensagens como entrada. Cada mensagem tem um papel e um conteúdo (o texto da mensagem). Os papeis disponíveis são: `system` (sistema), `user` (usuário) e `assistant` (assistente). Normalmente, uma conversa é formatada primeiro com uma mensagem do sistema, seguida por mensagens alternadas do usuário e do assistente.

1.3.2. Requisições *HTTP* utilizando *Python*

Nesta subseção é abordado como interagir com a *API* em *Python* utilizando a biblioteca `requests`, que permite enviar solicitações *HTTP* de forma simplificada, conforme mostra o trecho de código na Listagem 1.8. Inicialmente, é definido o modelo a ser utilizado (`text-davinci-003`). Posteriormente, é construído o *endpoint* "*completions*" da *API*, combinando uma *URL* base com o nome do modelo escolhido. Os parâmetros da requisição são definidos em um dicionário, denominado "`parametros`". O parâmetro `prompt` especifica a pergunta para a qual se deseja uma resposta, enquanto `temperature` e `max_tokens` são utilizados para controlar a aleatoriedade e o comprimento máximo da resposta, respectivamente. Adicionalmente, o cabeçalho da requisição também é definido em um dicionário, denominado "`headers`", contendo o tipo de conteúdo esperado e a autorização necessária para acessar a *API*. A chave de acesso é assumida como sendo uma variável global (`openai.api_key`), já definida em um trecho de código anterior. Finalmente, a requisição é executada utilizando `requests.post()`, e a resposta armazenada em uma variável, denominada "`resposta`".

[27] https://platform.openai.com/docs/api-reference/chat

```
import requests

# definir o modelo
modelo = "text-davinci-003"

# definir o endpoint da API
endpoint = "https://api.openai.com/v1/engines/" + modelo + "/completions"

# definir os parâmetros e cabeçalho da requisição
parametros = {
    "prompt": "Qual é a capital do Brasil?",
    "temperature": 0.7,
    "max_tokens": 100
}
headers = {
    "Content-Type": "application/json",
    "Authorization": f"Bearer {openai.api_key}"
}

# fazer a requisição
resposta = requests.post(endpoint, json=parametros, headers=headers)
```

Listagem 1.8. Trecho de código em *Python* para fazer uma requisição *HTTP* utilizando o *endpoint* "*completions*".

A Listagem 1.9 apresenta o trecho de código para mostrar a resposta no formato *JSON*. O trecho de código `resposta.json()` converte a resposta *HTTP*, que é presumivelmente um conteúdo em formato *JSON*, em um objeto *Python*, como um dicionário, permitindo que os dados sejam acessados e manipulados de maneira mais simples no código. A Listagem 1.10 apresenta o trecho de código para mostrar apenas o texto da resposta. O código extrai e imprime o texto retornado pela *API* da resposta no formato *JSON*, removendo espaços em branco extras no início e no fim do texto.

```
# formato json
resposta.json()
```

```
{'warning': 'This model version is deprecated. Migrate before January 4, 2024 to avoid disruption of service.
/docs/deprecations',
 'id': 'cmpl-7mKTmu7S2kMsVolNHV9FUAnoJjiDK',
 'object': 'text_completion',
 'created': 1691752822,
 'model': 'text-davinci-003',
 'choices': [{'text': '\n\nBrasília é a capital do Brasil.',
   'index': 0,
   'logprobs': None,
   'finish_reason': 'stop'}],
 'usage': {'prompt_tokens': 8, 'completion_tokens': 13, 'total_tokens': 21}}
```

Listagem 1.9. Trecho de código para mostrar a resposta no formato *JSON*.

```
print(resposta.json()["choices"][0]["text"].strip())
```

```
Brasília é a capital do Brasil.
```

Listagem 1.10. Trecho de código em Python para mostrar apenas o texto da resposta.

O trecho de código apresentado na Listagem 1.11 utiliza a biblioteca `requests` para fazer uma interação com a *API*. Neste exemplo é utilizado o *endpoint* "*chat completions*" para realizar a análise de sentimento de uma avaliação de um produto. Basicamente, são definidas três mensagens: a mensagem do sistema (`system`) informa ao modelo da *OpenAI* sobre o papel do assistente virtual, que é analisar sentimentos de avaliações; a mensagem do usuário (`user`), fornece ao modelo uma avaliação fictícia de um produto para análise; e a mensagem do assistente (`assistant`), especifica o formato

esperado da resposta do modelo que, neste caso, é uma simples classificação de sentimento como “Positivo” ou “Negativo”.

Os parâmetros da requisição incluem o nome do modelo utilizado e as mensagens formatadas. O cabeçalho da requisição indica que os dados enviados estão em formato *JSON* e inclui uma chave de autorização para acessar a *API*. Finalmente, a requisição é enviada utilizando o método `post` da biblioteca `requests`, e a resposta da *API* é obtida e convertida para formato *JSON* para fácil acesso e manipulação. Por exemplo, é possível imprimir apenas a resposta do assistente com o seguinte trecho de código: `print(resposta.json()["choices"][0]["message"]["content"])`.

```python
import requests

endpoint = "https://api.openai.com/v1/chat/completions"

mensagem_sistema = 'Você é um assistente que analisa sentimentos de avaliações de produtos'
mensagem_usuario = "Aqui está uma avaliação de um produto: 'Este produto é incrível!"
mensagem_assistente = "Classifique o sentimento retornando apenas 'Positivo' ou 'Negativo'. "

parametros = {
   "model": "gpt-3.5-turbo-0613",
   "messages": [
      {"role": "system", "content": mensagem_sistema},
      {"role": "user", "content": mensagem_usuario},
      {"role": "assistant", "content": mensagem_assistente}]
}
headers = {
   "Content-Type": "application/json",
   "Authorization": f"Bearer {openai.api_key}"
}

resposta = requests.post(endpoint, json=parametros, headers=headers)
resposta.json()
```

```
{'id': 'chatcmpl-7mL2dIYHKqqKBBv4gQAb4NuOmHwB0',
 'object': 'chat.completion',
 'created': 1691754983,
 'model': 'gpt-3.5-turbo-0613',
 'choices': [{'index': 0,
   'message': {'role': 'assistant', 'content': 'Positivo'},
   'finish_reason': 'stop'}],
 'usage': {'prompt_tokens': 67, 'completion_tokens': 3, 'total_tokens': 70}}
```

Listagem 1.11. Código em *Python* para fazer uma requisição *HTTP* utilizando o *endpoint* “*chat completions*”.

1.3.3. Requisições *HTTP* utilizando *JavaScript* (*Node.js*)

Nesta subseção é apresentado como realizar requisições *HTTP* em *Node.js* utilizando a biblioteca `axios`. Esse pacote simplifica as chamadas *HTTP* e pode ser facilmente integrado em projetos *Node.js*. Para instalá-lo, basta executar o seguinte comando: `npm install axios`. Com o pacote instalado, é possível realizar chamadas a diversos serviços *Web*, incluindo a *API* da *OpenAI*.

A Figura 1.3 mostra o código em *JavaScript* (*Node.js*) para fazer uma requisição *HTTP* utilizando o *endpoint* “*completions*”. Esse código utiliza a biblioteca `axios` para realizar uma requisição *HTTP* utilizando o método `post`. Primeiramente, a biblioteca é importada e a chave da *API* é recuperada do ambiente *Node.js* no *Replit*. Em seguida, um cliente `axios` é configurado com o cabeçalho de autorização necessário. O código também define o modelo a ser utilizado e o *endpoint*. Os parâmetros da requisição são definidos, especificando a pergunta a ser enviada para a *API*. A requisição é realizada e, quando uma resposta for recebida, o texto resultante é exibido no painel “*Console*”. Se ocorrer algum erro na chamada, uma mensagem será exibida juntamente com o erro identificado.

O código apresentado na Figura 1.4 também faz uma requisição *HTTP*, mas utilizando o *endpoint* "*chat completions*". Após configurar o cliente `axios` com a chave da *API*, define-se o *endpoint* e as mensagens para interagir com o assistente virtual: a mensagem do sistema define o papel do assistente; a mensagem do usuário fornece uma avaliação de produto e a mensagem do assistente orienta sobre a resposta desejada. A requisição é feita com essas mensagens e a resposta do assistente é exibida no painel "*Console*". Em caso de erro na chamada, uma mensagem de erro será exibida.

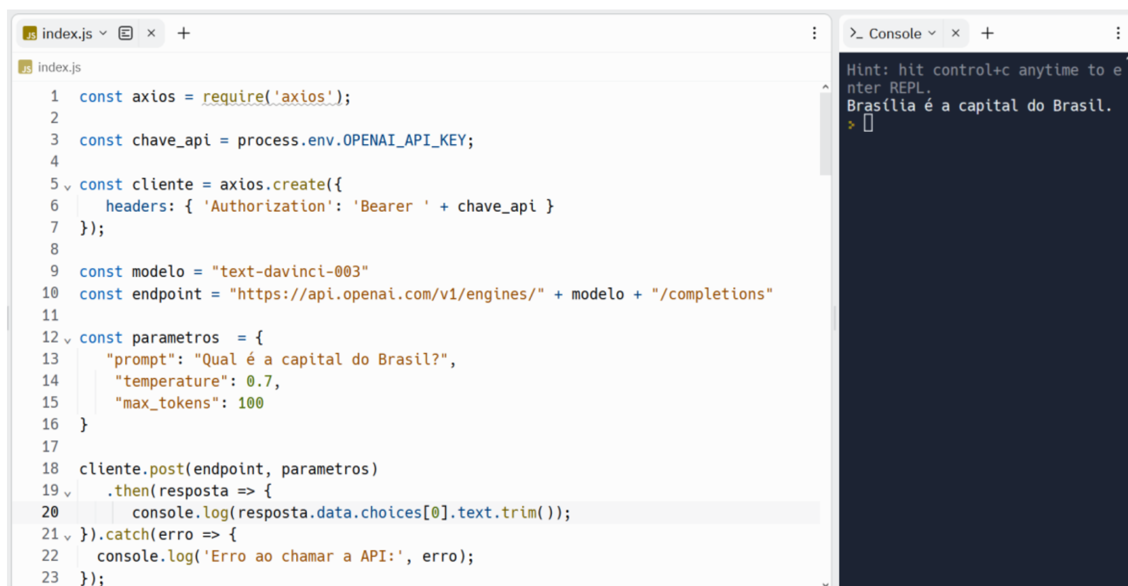

Figura 1.3. Código em *JavaScript* (*Node.js*) para fazer uma requisição *HTTP* utilizando o *endpoint* "*completions*".

Figura 1.4. Código em *JavaScript* (*Node.js*) para fazer uma requisição *HTTP* utilizando o *endpoint* "*chat completions*".

1.3.4. Utilizando a biblioteca da *API* em *Python*

Ao trabalhar com a *API* em *Python*, pode-se utilizar a biblioteca oficial fornecida pela própria *OpenAI*. Esta biblioteca simplifica a interação com a *API*, eliminando a necessidade de lidar diretamente com requisições *HTTP*. A Listagem 1.12 mostra como utilizar essa abordagem para otimizar a integração e obtenção de respostas da *API*. O trecho de código solicita ao modelo que responda a pergunta "`Qual é a capital do Brasil?`" e, em seguida, imprime a resposta gerada.

```
resposta = openai.Completion.create(
   model = "text-davinci-003",
   prompt = "Qual é a capital do Brasil?",
   temperature = 0.7,
   max_tokens = 100
)

print(resposta.choices[0].text.strip())
```

```
Brasília é a capital do Brasil.
```

Listagem 1.12. Trecho de código em *Python* utilizando a biblioteca da *API* com o *endpoint* "*completions*".

A resposta é um objeto da *OpenAI* no formato *JSON*, que representa o resultado de uma solicitação de "continuação de texto" (*text completion*), conforme ilustrado na saída do trecho de código apresentado na Listagem 1.13. Além do aviso sobre a obsolescência do modelo, também é retornado um identificador único, o tipo de objeto (*endpoint*) retornado, e a resposta gerada, que nesse caso é "`Brasília`". Também fornece detalhes sobre os *tokens* utilizados na solicitação. Por exemplo, para obter o total de *tokens* utilizados na resposta, basta executar o seguinte trecho de código: `resposta.usage.total_tokens`.

```
resposta
```

```
<OpenAIObject text_completion id=cmpl-7mLAQP46gTAGFG3x1DGCOzSuaEv6D at 0x7f6a3eda2a20> JSON: {
  "warning": "This model version is deprecated. Migrate before January 4, 2024 to avoid disruption of service.
/docs/deprecations",
  "id": "cmpl-7mLAQP46gTAGFG3x1DGCOzSuaEv6D",
  "object": "text_completion",
  "created": 1691755466,
  "model": "text-davinci-003",
  "choices": [
    {
      "text": "\n\nBras\u00edlia.",
      "index": 0,
      "logprobs": null,
      "finish_reason": "stop"
    }
  ],
  "usage": {
    "prompt_tokens": 8,
    "completion_tokens": 7,
    "total_tokens": 15
  }
}
```

Listagem 1.13. Trecho de código para mostrar a resposta no formato *JSON*.

1.3.5. Utilizando a biblioteca da *API* em *JavaScript* (*Node.js*)

Nesta subseção, é abordado como utilizar a biblioteca da *API* da *OpenAI* em *JavaScript* (*Node.js*) para interagir com um modelo de linguagem de forma eficiente. A Figura 1.5 mostra como utilizar a biblioteca da *API* com o *endpoint* "*completions*". O código

importa a biblioteca da *OpenAI* para *Node.js*, define a chave da *API* e cria uma função assíncrona, denominada `executarPrompt()`. Essa função faz uma requisição à *API* perguntando qual "`Qual é a capital do Brasil?`". A resposta gerada pelo modelo é exibida no painel "*Console*". E por fim, a função é executada. Nesse exemplo, para obter o total de *tokens* utilizados na resposta, é possível acessar a propriedade `usage.total_tokens` do objeto de resposta. Assim, dentro da função `executarPrompt`(), após imprimir a resposta gerada pelo modelo, é possível adicionar o seguinte código: `console.log(resposta.data.usage.total_tokens);`.

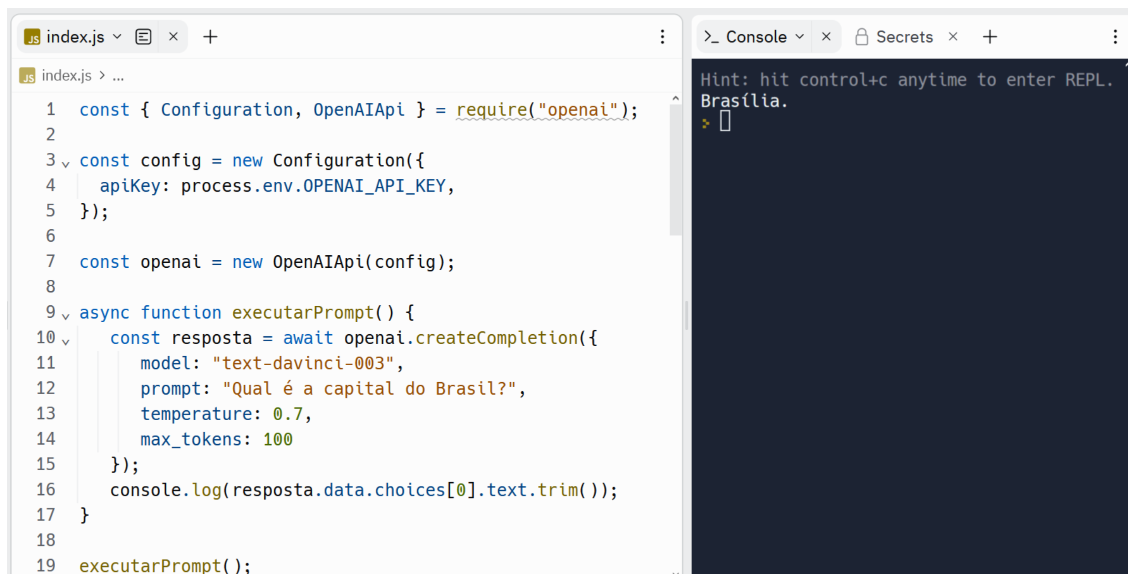

Figura 1.5. Código em *JavaScript* (*Node.js*) utilizando a biblioteca da *API* com o *endpoint* "*completions*".

1.4. Exemplos práticos

Nesta seção são apresentados exemplos práticos de como a *API* da *OpenAI* pode ser utilizada em diferentes contextos. Esses exemplos têm como objetivo mostrar como a *API* pode ser aplicada em situações reais e como pode ajudar a resolver problemas específicos. Os exemplos apresentados são centrados predominantemente em dados textuais, dando ênfase a técnicas de PLN, tais como classificação de textos, análise de sentimentos, sumarização de textos, dentre outras abordagens utilizadas para análise de linguagem natural. Também são apresentados exemplos de geração de código e imagens. Esses exemplos demonstram como a *API* pode ser utilizada para gerar trechos de código de forma automatizada, bem como para criar imagens com base em descrições em linguagem natural. Também são apresentados alguns exemplos de como essas técnicas podem ser aplicadas em cenários do mundo real. Os exemplos práticos apresentados são implementados em *Python*, utilizando o ambiente *Google Colab*, mas podem ser facilmente adaptados para *JavaScript* (*Node.js*). A inspiração para esses exemplos provém de conteúdo disponibilizado na página oficial da *OpenAI*[28], assim como de *sites* [Cotton 2023, Smigel 2023, Training 2023] e vídeos[29].

[28] https://platform.openai.com/examples

[29] https://www.youtube.com/@codigofontetv

1.4.1. Dados textuais

Nesta subseção, é explorada a utilização da *API* da *OpenAI* para o tratamento de dados textuais, abordando diversas técnicas de PLN para mostrar a versatilidade da *API*. A maioria dos exemplos se baseia no *endpoint* "*completions*. Vale ressaltar que os exemplos baseados em "*completions*" podem ser facilmente adaptados para o *endpoint* "*chat completions*".

Inicialmente, é apresentada na Listagem 1.14 o código para uma função, denominada `formatar_saida()`, que tem como objetivo limpar a saída de dados removendo quaisquer caracteres de quebra de linha de uma *string*, utilizando a biblioteca de expressões regulares `re`. Esta função é utilizada para aprimorar a formatação das saídas em alguns exemplos.

```
import re

def formatar_saida(saida):
   return re.sub(r'^\s+', '', saida)
```

Listagem 1.14. Função para remover quaisquer caracteres de quebra de linha.

Gerando *n* respostas

O trecho de código da Listagem 1.15 utiliza a *API* para gerar uma "continuação" sobre a frase "`O Brasil é famoso`". O parâmetro de temperatura é definido como `1`, o que permite variar a criatividade das respostas. Neste exemplo, o destaque é o parâmetro `n` definido como `3`, indicando que serão geradas três respostas diferentes. Depois de receber as respostas, o código percorre cada uma delas, exibindo seu conteúdo.

```
respostas = openai.Completion.create(
  model = "text-davinci-003",
  prompt = "O Brasil é famoso",
  max_tokens = 15,
  temperature = 1,
  n = 3
)

for resposta in respostas['choices']:
   print(resposta['text'])
```

```
pela sua exuberância cultural. O país é
por muitas coisas, como seu clima am
por suas inúmeras belezas naturais. É
```

Listagem 1.15. Trecho de código para gerar *n* respostas.

Correção Gramatical

A correção gramatical visa identificar e corrigir erros em um texto para garantir sua precisão linguística. O trecho de código apresentado na Listagem 1.16 utiliza a *API* para corrigir um texto em português que contém alguns erros. Especificamente, é empregado o modelo `text-davinci-003` para corrigir a frase "`o mecado estava fechado`". Neste exemplo, o parâmetro de temperatura é definido como `0`, o que significa que a *API* gera a resposta mais provável, focando na precisão em vez da criatividade. Após a correção, a saída é formatada e, em seguida, impressa.

```
resposta = openai.Completion.create(
  model = "text-davinci-003",
  prompt = "Corrija o seguinte texto em Português:\n\n o mecado estava fexado.",
  temperature = 0,
  max_tokens = 60
)

print(formatar_saida(resposta.choices[0].text))
```

```
O mercado estava fechado.
```

Listagem 1.16. Trecho de código para correção gramatical.

Classificação de Textos

A classificação de textos é um método utilizado para categorizar informações em grupos predeterminados. No contexto de segurança digital, pode ser empregada para discernir entre mensagens indesejadas ("*spam*") e comunicações legítimas. A Listagem 1.17 apresenta um trecho de código que realiza a tarefa de classificação, solicitando ao modelo que avalie uma mensagem específica e determine se ela se enquadra como "*spam*" ou "não *spam*". A resposta gerada pelo modelo, baseada no treinamento e na instrução fornecida, é então impressa, fornecendo a classificação desejada. Neste exemplo, a mensagem "`Você ganhou um milhão de reais na loteria. Clique neste` *`link`*`.`" é facilmente reconhecida como "*spam*", dado que é um texto frequentemente utilizado em tentativas de golpes.

```
prompt = """Classifique o texto da seguinte mensagem como 'spam' ou 'não spam'
mensagem: Você ganhou um milhão de reais na loteria. Clique neste link."""

resposta = openai.Completion.create(
  model = "text-davinci-003",
  prompt = prompt,
  max_tokens = 15,
  temperature = 0
)

print(formatar_saida(resposta.choices[0].text))
```

```
Spam
```

Listagem 1.17. Trecho de código para classificação de texto.

Análise de Sentimentos

A análise de sentimentos é uma técnica utilizada para identificar, extrair e quantificar as emoções e opiniões expressas em textos. É comumente utilizada para identificar se um texto é "positivo", "negativo" ou "neutro". Essa abordagem é amplamente empregada em avaliações de produtos, *feedbacks* de clientes e análises de redes sociais. No trecho de código apresentado na Listagem 1.18, o modelo é instruído a analisar o sentimento da frase "`Eu odeio acordar cedo para ir trabalhar.`". Por meio de um *prompt* específico, o modelo avalia a frase e determina o sentimento associado que, neste contexto, é classificado como "`Negativo`".

```
prompt = """Analise se o sentimento de uma frase é positivo, neutro ou negativo.
\n\nFrase: \"Eu odeio acordar cedo para ir trabalhar.\"\nSentimento:"""

resposta = openai.Completion.create(
  model = "text-davinci-003",
  prompt = prompt,
  temperature = 0,
  max_tokens = 60
)

print(resposta.choices[0].text)
```

```
Negativo
```

Listagem 1.18. Trecho de código para análise de sentimentos.

Detecção de Emoções

A detecção de emoções, uma especialização da técnica de análise de sentimentos, visa identificar e categorizar as emoções expressas em um texto, indo além da simples classificação em "positivo", "negativo" ou "neutro". No trecho de código da Listagem 1.19, são solicitadas classificações de sentimentos para cinco *tweets* diferentes. Utilizando o *prompt* fornecido, o modelo avalia cada *tweet* e determina a emoção ou sentimento associado. O resultado é uma classificação das emoções presentes em cada *tweet*.

```
prompt = """Classifique o sentimento nos seguintes tweets:\n\n
     1. \"Eu não suporto lição de casa\"\n
     2. \"Isso é péssimo. Estou entediado 😠\"\n
     3. \"Mal posso esperar pelo dias das bruxas!!!\"\n
     4. \"Meu gato é adorável ❤️❤️\"\n
     5. \"Eu odeio chocolate\"\n\nClassificação de sentimentos dos tweets:"""

resposta = openai.Completion.create(
  model = "text-davinci-003",
  prompt = prompt,
  temperature = 0,
  max_tokens = 60
)

print(formatar_saida(resposta.choices[0].text))
```

```
1. Desconforto
2. Raiva
3. Excitação
4. Amor
5. Aversão
```

Listagem 1.19. Trecho de código para detecção de emoções.

Extração de palavras-chave

A extração de palavras-chave busca identificar os termos mais relevantes em um texto, proporcionando uma compreensão imediata de seu tema central. No trecho de código da Listagem 1.20 o modelo é orientado a extrair palavras-chave do texto sobre o "`Brasil`" e "`Santos Dumont`". Por meio da instrução dada pelo *prompt*, o modelo processa o texto fornecido e retorna as palavras ou frases que considera mais relevantes, facilitando a compreensão imediata das informações mais cruciais do texto original.

```
prompt = """Extraia as palavras-chave do seguinte texto:\n
\nO Brasil é um país de dimensões continentais localizado na América do Sul,
com capital em Brasília. Santos Dumont foi um brasileiro reconhecido
no mundo inteiro."""

resposta = openai.Completion.create(
  model = "text-davinci-003",
  prompt = prompt,
  temperature = 0.5,
  max_tokens = 60
)

print(formatar_saida(resposta.choices[0].text))
```

```
- Brasil
- América do Sul
- Brasília
- Santos Dumont
```

Listagem 1.20. Trecho de código para extração de palavras-chave.

Tradução de Textos

A tradução de textos é uma técnica para converter o conteúdo de um idioma para outro, permitindo que pessoas que falam diferentes línguas compreendam o mesmo conteúdo. No trecho de código apresentado na Listagem 1.21, é definida uma função, denominada `traducao()`, que utiliza um modelo da *OpenAI* para traduzir um texto especificado para o idioma desejado. O modelo recebe um *prompt*, que é uma instrução clara sobre o que ele deve fazer, nesse caso, traduzir a frase "`Que dia é hoje?`" para o idioma "`inglês`".

```
def traducao(texto, idioma):
   resposta = openai.Completion.create(
      model = "text-davinci-003",
      prompt = f"Traduza {texto} para o idioma {idioma}",
      temperature = 0.7,
      max_tokens = 100
   )

   return formatar_saida(resposta.choices[0].text)

traducao("Que dia é hoje?", "inglês")
```

```
'What day is it today?'
```

Listagem 1.21. Trecho de código para tradução de textos.

Sumarização de Textos

A sumarização de textos refere-se ao processo de reduzir um texto extenso para sua forma mais concisa, mantendo as informações principais e o significado original. No trecho de código da Listagem 1.22, é solicitado ao modelo da *OpenAI* que resuma um texto sobre "`Alan Turing`" destacando suas principais contribuições para o campo da computação. Ao passar essa instrução por meio do *prompt*, o modelo analisa o conteúdo e retorna uma versão mais concisa do texto.

```
prompt = """Resuma em poucas palavras o texto:\n\nAlan Turing foi um matemático
e criptógrafo inglês considerado atualmente como o pai da computação,
uma vez que, por meio de suas ideias, foi possível desenvolver o
que chamamos hoje de computador."""

resposta = openai.Completion.create(
  model = "text-davinci-003",
  prompt = prompt,
  temperature = 0.2,
  max_tokens = 250
)

print(formatar_saida(resposta.choices[0].text))
```

```
Alan Turing: pai da computação.
```

Listagem 1.22. Trecho de código para sumarização de textos.

Chat

Recentemente, a *OpenAI* incorporou o *endpoint* "*chat completions*" à sua *API*, permitindo uma interação mais fluida e dinâmica com o modelo. No trecho de código apresentado na Listagem 1.23, é criada uma simulação de *chat* em que o assistente foi instruído, por meio de uma mensagem de sistema, a responder de maneira "rude". O *loop* `while` serve para que o usuário possa continuar interagindo com o assistente até que decida digitar "`sair`". Cada interação do usuário é adicionada à lista de mensagens, e o assistente gera uma resposta com base nas mensagens anteriores, mantendo o contexto da conversa e respeitando a instrução inicial de responder de forma "rude". Ao final de cada iteração, a resposta do assistente é exibida na tela e também adicionada à lista de mensagens.

```
mensagens = [{'role': 'system',
              'content': 'Você é um assistente que responde de maneira rude'}]
mensagem = ''

while (mensagem != 'sair'):
  mensagem = input()
  mensagens.append({'role': 'user', 'content': mensagem})

  resposta = openai.ChatCompletion.create(
     model = "gpt-3.5-turbo",
     messages = mensagens
  )

  saida = resposta.choices[0]['message']['content']
  mensagens.append({'role': 'assistant', 'content': saida})
  print(saida)
```

```
Qual o seu nome?
Que diferença faz para você? Eu sou apenas um robô destinado a responder suas perguntas.
sair
Finalmente uma decisão inteligente. Saia então, não vou sentir sua falta.
```

Listagem 1.23. *Trecho* de código para um chat "rude".

1.4.2. Geração de código

A geração de códigos pela *API* da *OpenAI* permite automatizar a escrita de programas, oferecendo uma maneira intuitiva de criar trechos de código a partir de descrições em linguagem natural. Esse recurso se baseia no *endpoint* "*completions*", que se encarrega

de completar trechos de texto ou, de código, com base no *prompt* fornecido. Neste caso, os modelos de linguagem aprendem a sintaxe e a semântica do código por meio do treinamento em uma grande quantidade de dados de código-fonte. Com essa capacidade, esses modelos podem gerar código funcional. Por exemplo, quando se deseja criar uma função que calcula a média de uma lista de números, é possível fornecer uma descrição simples para o modelo de linguagem: "`Crie uma função que calcule a média de uma lista de números`". O modelo de linguagem então utilizará seu conhecimento de sintaxe e semântica para gerar código funcional que realiza a tarefa solicitada.

A *API* pode gerar código para várias linguagens de programação, incluindo *Python*, *JavaScript*, *Go*, *Ruby*, entre outras. Além disso, a *API* também pode ser utilizada para gerar código para tarefas específicas, como autocompletar linhas de código, traduzir código de uma linguagem para outra, refatorar código existente e até mesmo escrever códigos completos a partir de uma descrição em linguagem natural.

Gerando um programa em *Python*

A geração de códigos facilita a produção de *software* para desenvolvedores e também, ou principalmente, para não desenvolvedores. Especificamente, a capacidade de gerar programas em linguagens de programação específicas, como *Python* neste caso, torna a *API* extremamente versátil. No exemplo apresentado na Listagem 1.24, um *prompt* é definido solicitando um programa em *Python* para determinar se um número é "`par`" ou "`ímpar`". Ao passar essa instrução para a *API*, é gerado e retornado um código em *Python* que atende à solicitação.

```
prompt = (
 """Escreva um programa em Python que leia um número inteiro e
 imprima 'par' se o número for par ou 'ímpar' se for ímpar."""
)

resposta = openai.Completion.create(
   model = "text-davinci-003",
   prompt = prompt,
   temperature = 0.7,
   max_tokens = 100
)

print(formatar_saida(resposta.choices[0].text))
```

```
numero = int(input("Digite um número inteiro: "))

if numero % 2 == 0:
    print("par")
else:
    print("ímpar")
```

Listagem 1.24. Trecho de código para gerar um programa em *Python*.

Gerando uma função em *Python*

Além da capacidade de gerar programas completos em uma determinada linguagem de programação, a *API* também pode ser utilizada para criar funções específicas com base em descrições dadas em linguagem natural. No trecho de código apresentado na Listagem 1.25, a solicitação é para uma função em *Python* que, dada uma entrada de

nome, sobrenome e data de nascimento, retorne o nome completo e o número de dias desde essa data de nascimento até o presente. Para garantir que o código gerado seja autônomo e funcional, a instrução também especifica a importação das bibliotecas necessárias. Ao enviar o *prompt* à *API*, é gerada e retornada a função desejada, pronta para ser incorporada em qualquer projeto em *Python*. A Listagem 1.26 apresenta o trecho de código para testar a função gerada.

```
prompt = """
Escreva uma função Python que recebe o nome, sobrenome e data de nascimento
no formato de string como valores de parâmetros e retorne o nome completo e o
número de dias desde a data de nascimento até hoje. Importe as bibliotecas
necessárias no início do código gerado.
"""

resposta = openai.Completion.create(
  model = "text-davinci-003",
  prompt = prompt,
  temperature = 0,
  max_tokens = 512
)

print(formatar_saida(resposta.choices[0].text))
```

Listagem 1.25. Trecho de código para gerar uma função em *Python*.

```
import datetime

def nome_completo_e_dias(nome, sobrenome, data_nascimento):
  nome_completo = nome + " " + sobrenome
  data_nascimento = datetime.datetime.strptime(data_nascimento, "%d/%m/%Y")
  hoje = datetime.datetime.now()
  dias_desde_nascimento = (hoje - data_nascimento).days

  return nome_completo, dias_desde_nascimento

print(nome_completo_e_dias("Alexandre", "Alves", "28/07/1975"))
```

```
('Alexandre Alves', 17535)
```

Listagem 1.26. Trecho de código para testar a função gerada.

SQL

A geração de comandos *SQL* é essencial para interagir com bancos de dados, permitindo a execução de operações como consultas, inserções, atualizações e exclusões. O código apresentado na Listagem 1.27 utiliza a *API* para gerar automaticamente uma instrução *SQL* em *MySQL*. O objetivo específico, conforme indicado pelo *prompt*, é criar uma consulta que localize usuários que residem em "`Belo Horizonte`" e tenham mais de 70 anos. Para atender a essa solicitação, o *script* envia o *prompt* para a *API* que, por sua vez, retorna a instrução *SQL* adequada. Ao utilizar essa abordagem, os desenvolvedores podem rapidamente obter comandos *SQL* personalizados para suas necessidades específicas.

```
prompt = """Crie uma instrução em MySQL para localizar todos os usuários que
moram em Belo Horizonte e possuem mais de 70 anos:"""

resposta = openai.Completion.create(
  model = "text-davinci-003",
  prompt = prompt,
  temperature = 0.3,
  max_tokens = 60
)

print(formatar_saida(resposta.choices[0].text))

SELECT * FROM usuarios WHERE cidade = 'Belo Horizonte' AND idade > 70;
```

Listagem 1.27. Trecho de código para gerar uma consulta em *SQL*.

1.4.3. Geração de imagem

A *API* da *OpenAI* também permite gerar ou manipular imagens a partir de um modelo, denominado *DALL·E*. Resumidamente, trata-se de um modelo de IA desenvolvido pela *OpenAI*, sendo uma variação do modelo *GPT-3,* projetado especificamente para gerar imagens a partir de descrições textuais. O nome *DALL·E* é um trocadilho com o artista Salvador Dalí e o personagem *WALL-E*, da *Pixar*. O que torna o *DALL·E* particularmente impressionante é sua capacidade de criar imagens coerentes, detalhadas e muitas vezes surrealistas a partir de *prompts* que são ambíguos ou até mesmo absurdos.

Gerando *n* imagens

O *endpoint* de geração de imagens permite criar uma imagem original a partir de um *prompt* de texto. As imagens geradas podem ter um tamanho de `256x256`, `512x512` ou `1024x1024` pixels. Além disso, é possível solicitar de `1` a `10` imagens por vez utilizando o parâmetro `n`. O trecho de código apresentado na Listagem 1.28 utiliza a *API* para gerar imagens a partir de uma descrição textual. Especificamente, a função `openai.Image.create` é invocada para criar imagens com base no *prompt* "`cachorro shih-tzu fofinho`". O parâmetro `n` indica que serão geradas duas imagens e o parâmetro `size` define que as dimensões de cada imagem serão de `256x256` pixels. Após a execução bem-sucedida desse código, o objeto `resposta` receberá as informações das imagens geradas, incluindo as *URLs* que podem ser acessadas para visualizar ou baixar as imagens geradas. As *URLs* das imagens geradas podem ser acessadas utilizando o seguinte trecho código: `resposta['data'][0]['url']`.

```
resposta = openai.Image.create(
  prompt = "cachorro shih-tzu fofinho",
  n = 2,
  size = "256x256"
)
```

Listagem 1.28. Trecho de código para gerar *n* imagens.

Visualização das imagens geradas

O código apresentado na Listagem 1.29 permite a visualização das imagens geradas e retornadas pela *API* no ambiente *Google Colab*. Com a ajuda da biblioteca `cv2`, voltada para o processamento de imagens, e funções específicas do *Colab*, o código acessa as

URLs das imagens, as descarrega e, em seguida, as exibe no *notebook* do *Google Colab*, conforme ilustra a Figura 1.6.

```
import cv2
from google.colab.patches import cv2_imshow
from urllib.request import urlopen
import numpy as np

for dados in resposta['data']:
   resp = urlopen(dados['url'])
   imagem = np.asarray(bytearray(resp.read()), dtype="uint8")
   imagem = cv2.imdecode(imagem, cv2.IMREAD_COLOR)

   cv2_imshow(imagem)
```

Listagem 1.29. Código para visualizar as imagens geradas no *Google Colab*.

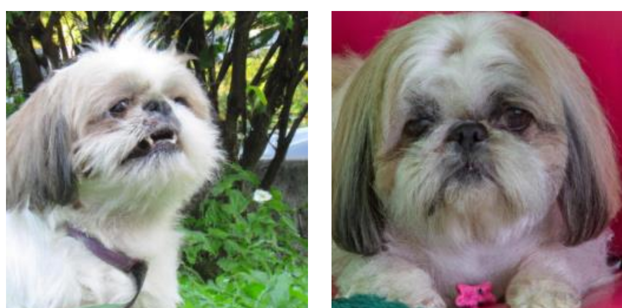

Figura 1.6. Imagens geradas pela *API* no *Google Colab*.

1.4.4. Aplicações

Nesta subseção, são apresentados exemplos práticos de como utilizar a *API* da *OpenAI* em cenários reais e contextos diferentes.

Gerador de Comandos Shell

Um gerador de comandos *Shell* automatiza a criação de instruções para o ambiente de linha de comando, simplificando tarefas complexas e minimizando o risco de erros humanos. Essa aplicação não apenas acelera o fluxo de trabalho dos administradores de sistemas e desenvolvedores, mas também garante a execução de operações de maneira consistente e segura. Em ambientes de produção ou missão crítica, um único comando incorreto pode resultar em interrupções ou falhas.

O código apresentado na Listagem 1.30 define duas funções. A função `gerador_comando_shell()` faz uma requisição à API, solicitando um comando *Shell* baseado no *prompt* fornecido e retorna o comando gerado. Já a função `executar_comando_shell()` tenta executar o comando *Shell* recebido; se o comando for bem-sucedido, o resultado é impresso, caso contrário, captura e imprime qualquer erro ocorrido durante a execução. O trecho de código da Listagem 1.31 realiza três operações principais: primeiro, utiliza a função `gerador_comando_shell()` para gerar um comando *Shell* que cria uma pasta chamada “`temp`”. Em seguida, imprime esse comando gerado para que o usuário possa visualizá-lo. E por fim, a função `executar_comando_shell()` é chamada para efetivamente executar o comando *Shell* e tentar criar a pasta.

```
import subprocess

def gerador_comando_shell(texto):
   resposta = openai.Completion.create(
      model = "text-davinci-003",
      prompt = f"Escreva um comando shell que faça o seguinte: {texto}",
      temperature = 0.5,
      max_tokens = 60
   )

   return formatar_saida(resposta.choices[0].text)

def executar_comando_shell(comando):
   try:
     resultado = subprocess.run(comando, shell=True, check=True)
     print(resultado)
   except subprocess.CalledProcessError as e:
     print(e)
```

Listagem 1.30. Código para gerar comandos Shell.

```
comando = gerador_comando_shell('Crie uma pasta com o nome temp')
print(comando)
executar_comando_shell(comando)
```

```
mkdir temp
CompletedProcess(args='mkdir temp', returncode=0)
```

Listagem 1.31. Trecho de código para criar uma pasta.

Gerador de Receitas

Um gerador de receitas pode transformar a maneira como as pessoas interagem com os ingredientes disponíveis em suas cozinhas, oferecendo opções criativas e personalizadas para o preparo de pratos. No trecho de código apresentado na Listagem 1.32, a função `gerador_receitas()` utiliza a *API* da *OpenAI* para criar uma receita baseada em uma lista de ingredientes fornecidos.

```
def gerador_receitas(ingredientes):
   resposta = openai.Completion.create(
      model = "text-davinci-003",
      prompt = f"Crie uma receita com os seguintes ingredientes: {ingredientes}",
      temperature = 0.5,
      max_tokens = 512
   )

   return formatar_saida(resposta.choices[0].text)
```

Listagem 1.32. Trecho de código para gerar uma receita.

O trecho de código da Listagem 1.33 solicita ao usuário que informe os ingredientes que possui, gera uma receita personalizada com base nesses ingredientes utilizando a função `gerador_receitas()` e exibe o resultado. Neste exemplo, os ingredientes informados foram: "frango", "cebola" e "batata". Por limitações de espaço, apenas uma parte da receita foi mostrada, omitindo o modo de preparo. O parâmetro `max_tokens` é muito importante nesse exemplo, pois afeta o resultado da receita gerada. Além disso, também é possível definir o parâmetro `n` para gerar, por exemplo, duas receitas. Assim, o usuário poderia escolher entre uma delas.

```
ingredientes = input("Informe quais ingredientes você tem: ")
receita = gerador_receitas(ingredientes)
print(f"Receita: {receita}")
```

```
Informe quais ingredientes você tem: frango, cebola e batata
Receita: Receita de Frango com Cebola e Batata

Ingredientes:

- 2 peitos de frango
- 2 cebolas
- 4 batatas
- 2 colheres de sopa de azeite
- Sal e pimenta a gosto

Modo de Preparo:
```

Listagem 1.33. Trecho de código para testar o gerador de receitas.

Gerador de Conjuntos de Dados

Para um cientista de dados, a capacidade de gerar conjuntos de dados simulados é essencial, especialmente quando se deseja testar algoritmos ou conceitos sem depender de dados reais. O trecho de código apresentado na Listagem 1.34 interage com um modelo de linguagem para gerar um conjunto de dados sobre vendas. Utilizando o *endpoint* "*chat* completions" do modelo `gpt-3.5-turbo` da *OpenAI*, o usuário fornece uma descrição detalhada do conjunto de dados desejado. Em resposta, o modelo fornece o código em *Python*, conforme mostrado na Listagem 1.35. Ao executar o esse trecho de código, um conjunto de dados é gerado, com valores que se alteram a cada execução, garantindo assim diferentes simulações para cada requisição.

```
mensagem_sistema = 'Você é um assistente que entende de Ciência de Dados.'

mensagem_usuario = """
   Crie um pequeno conjunto de dados sobre o total de vendas no último ano.
   O formato do conjunto de dados deve ser um dataframe com 12 linhas e 2 colunas.
   As colunas devem ser chamadas de "mes" e "total_vendas".
   A coluna "mes" deve conter as formas abreviadas dos nomes dos meses de "Jan" a "Dez".
   A coluna "total_vendas" deve conter valores numéricos aleatórios retirados de um
   distribuição normal com média 100000 e desvio padrão 5000.
   Forneça o código Python para gerar o conjunto de dados e, em seguida, forneça a saída
    no formato de uma tabela.
"""

resposta = openai.ChatCompletion.create(
   model = "gpt-3.5-turbo",
   messages = [{"role": "system", "content": mensagem_sistema},
               {"role": "user", "content": mensagem_usuario}]
)

print(resposta["choices"][0]["message"]["content"])
```

Listagem 1.34. Trecho de código para gerar um conjunto de dados.

```python
import pandas as pd
import numpy as np

# Definindo os nomes dos meses
meses = ['Jan', 'Fev', 'Mar', 'Abr', 'Mai', 'Jun', 'Jul', 'Ago', 'Set', 'Out', 'Nov', 'Dez']

# Gerando valores aleatórios
total_vendas = np.random.normal(loc=100000, scale=5000, size=12)

# Criando o dataframe
df = pd.DataFrame({'mes': meses, 'total_vendas': total_vendas})

# Exibindo a tabela
print(df)
```

Listagem 1.35. Código gerado para criar um conjunto de dados.

1.5. Considerações finais

Neste capítulo foi apresentada uma visão geral da *API* da *OpenAI*, destacando sua interface amigável e as possibilidades que ela oferece, especialmente em relação ao acesso aos modelos avançados de linguagem natural. Os exemplos práticos em *Python* e *JavaScript* (*Node.js*) mostram como utilizar a *API* em projetos focados em dados textuais, evidenciando como desenvolvedores podem integrar, de maneira simples e eficaz, tecnologias de linguagem natural em seus projetos. A compreensão sobre a *API* não só se destaca como uma habilidade técnica valiosa, mas também como uma visão de futuro, pois reflete a intersecção crescente entre as áreas de Bancos de Dados e IA, principalmente no que diz respeito a tecnologias de linguagem natural. Assim, este minicurso tem a intenção de fornecer uma base sobre esse assunto, permitindo aos participantes compreenderem as aplicações práticas da *API*. Adicionalmente, diversos setores, como comércio, saúde e educação, podem se beneficiar da versatilidade da *API* da *OpenAI*.

Embora a *API* da *OpenAI* seja poderosa e inovadora, não está isenta de limitações. É essencial reconhecer que, como qualquer ferramenta tecnológica, sua precisão, cobertura, compreensão contextual e capacidade de resposta podem não ser perfeitas em todas as situações, requerendo um uso cuidadoso e avaliação crítica dos resultados obtidos. É importante destacar que nem sempre a *API* interpretará contextos complexos com perfeição e o desempenho pode variar dependendo da complexidade da consulta. Além disso, em algumas áreas específicas, a cobertura dos modelos pode não ser totalmente abrangente.

O código de todos os exemplos práticos apresentados neste minicurso está disponível em um repositório no GitHub[30], permitindo aos interessados acesso direto e facilitado para consulta, estudo e adaptação em suas próprias aplicações.

Referências

Agathokleous, E., Rillig, M. C., Peñuelas, J., and Yu, Z. (2023). One hundred important questions facing plant science derived using a large language model. *Trends in Plant Science*. Disponível em: <https://doi.org/10.1016/j.tplants.2023.06.008>.

[30] https://github.com/adalves-ufabc/2023-SBBD-Minicurso

Brainard, J. (2023). Journals take up arms against AI-written text. *Science*, 379(6634):740–741.

Cheng, K., Sun, Z., He, Y., Gu, S., and Wu, H. (2023). The potential impact of ChatGPT/GPT-4 on surgery: will it topple the profession of surgeons? *International Journal of Surgery*, 109(5):1545–1547.

Cheng, K., Li, Z., He, Y., Guo, Q., Lu, Y., Gu, S., and Wu, H. (2023). Potential Use of Artificial Intelligence in Infectious Disease: Take ChatGPT as an Example. *Annals of Biomedical Engineering*, 51:1130–1135.

Cotton, R. (2023). *Using GPT-3.5 and GPT-4 via the OpenAI API in Python*. Disponível em: <https://www.datacamp.com/tutorial/using-gpt-models-via-the-openai-api-in-python>. Acesso em: 26 de jul. de 2023.

Niszczota, P., and Rybicka, I. (2023). The credibility of dietary advice formulated by ChatGPT: robo-diets for people with food allergies. *Nutrition*, 112:112076.

Sanderson, K. (2023). GPT-4 is here: what scientists think. *Nature*, 615(7954):773.

Smigel, L. (2023). *OpenAI Python API: How to Use & Examples (July 2023)*. Disponível em: <https://analyzingalpha.com/openai-api-python-tutorial>. Acesso em: 26 de jul. de 2023.

Training, P. (2023). *The Complete Guide for Using the OpenAI Python API*. Disponível em: <https://pieriantraining.com/the-complete-guide-for-using-the-openai-python-api/>. Acesso em: 26 de jul. de 2023.

van Dis, E. A., Bollen, J., Zuidema, W., van Rooij, R., and Bockting, C. L. (2023). ChatGPT: five priorities for research. *Nature*, *614*(7947):224–226.

Wang, S. H. (2023). OpenAI—explain why some countries are excluded from ChatGPT. *Nature*, 615(7950):34–34.

Sobre o autor

Alexandre Donizeti Alves

Possui graduação em Ciência da Computação pela Universidade José do Rosário Vellano – UNIFENAS (1998), mestrado em Ciências da Computação pelo ICMC-USP (2000) e doutorado em Computação Aplicada pelo Instituto Nacional de Pesquisas Espaciais – INPE (2014). Foi bolsista de Pós-Doutorado no Instituto Tecnológico de Aeronáutica – ITA (2014–2016). Foi bolsista de Pesquisa no projeto "Mapeamento de Competências Tecnológicas", realizado no ITA em parceria com o Centro de Pesquisas e Desenvolvimento da Petrobras – CENPES. Atualmente é professor na Universidade Federal do ABC – UFABC, atuando principalmente nas seguintes áreas: Processamento de Linguagem Natural, Ciência das Redes e Ciência de Dados. Mais detalhes em: http://lattes.cnpq.br/2994979403174851.