```
FILEPATH: C:\Users\andwe\Downloads\classification-
scrapers\run_local\run_local_scraper_amazon.py
import ssl
import string
from datetime import datetime, timedelta
from typing import Dict, List, Optional, Set, Tuple
from urllib.parse import unquote, urlparse

import nltk
import psycopg2
import psycopg2.extras
import requests
from bs4 import BeautifulSoup
from nltk.corpus import stopwords

try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    pass
else:
    ssl._create_default_https_context = _create_unverified_https_context

nltk.download("punkt")
nltk.download("stopwords")


def amazon_scraper(query) -> Optional[str]:
    SEARCH_PRODUCTS_URL = "https://www.amazon.com.br/s?k="

    proxy = {"http": "http://Eiprice-cc-
any:DQSXomtV7qri@gw.ntnt.io:5959"}

    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{SEARCH_PRODUCTS_URL}{query.replace(' ', '+')}"
    try:
        response = requests.get(url, headers=headers, proxies=proxy)
    except Exception as error:
        print("Erro ao fazer requisição:", error)
        return None

    soup = BeautifulSoup(response.content, "html.parser")
    product_list = soup.find_all(
        "div", class_="s-main-slot s-result-list s-search-results sg-row"
    )

    products = []

    for product_section in product_list:
```

```
        try:
            product = product_section.find(
                "div", class_="a-section aok-relative s-image-square-
aspect"
            )
            descricao = product.find("img", class_="s-image")["alt"]
            link_section = product_section.find(
                "a", class_="a-link-normal s-no-outline"
            )
            link = f"https://www.amazon.com.br{link_section['href']}"
            # img_link = product.find("img", class_="s-image")["src"]
            # sku = link.split("/dp/")[1].split("/")[0]

            products.append({"url": link, "description": descricao})

        except Exception as error:
            print("Erro ao buscar produto:", error)
            continue

    if most_similar_product := find_most_similar(query, products):
        print("Produto mais similar encontrado:")
        print(most_similar_product["url"])
        return most_similar_product["url"]
    else:
        print("Nenhum produto similar encontrado.")
        return None


def tokenize_and_lower(text: str) -> Set[str]:
    cleaned_text = remove_punctuation(text)
    tokens = nltk.word_tokenize(cleaned_text.lower())
    return set(tokens)


def remove_punctuation(text: str) -> str:
    """
    Remove punctuation from the given text.

    Args:
        text (str): The input text.

    Returns:
        str: The text without punctuation.
    """
    translator = str.maketrans("", "", string.punctuation)
    return text.translate(translator)


def find_most_similar(
    query: str, products: List[Dict[str, str]]
) -> Optional[Dict[str, str]]:
    query_tokens = tokenize_and_lower(query)
    max_intersection = 0
    most_similar_product = None
```

```
    min_intersection = calculate_min_intersection(query)

    for product in products:
        try:
            product_tokens = tokenize_and_lower(product["description"])
            intersection = len(query_tokens.intersection(product_tokens))

            if intersection >= min_intersection and intersection >
max_intersection:
                max_intersection = intersection
                most_similar_product = {
                    "url": product["url"],
                    "description": product["description"],
                }

        except Exception as error:
            print(f"Error processing product: {error}")
            continue

    return most_similar_product


def calculate_min_intersection(query: str, percentage: float = 50.0) ->
int:
    """
    Calculate the minimum intersection value based on the input query and
percentage.

    Args:
        query (str): The input query.
        percentage (float, optional): The percentage to calculate the
minimum intersection. Defaults to 50.0.

    Returns:
        int: The minimum intersection value.
    """
    tokens = nltk.word_tokenize(query)
    return int(len(tokens) * (percentage / 100))


def create_connection():
    return psycopg2.connect(
        dbname="eiprice-hml",
        user="postgres",
        password="GNrx+6bh)So<mU",
        host="35.184.21.249",
    )


def fetch_schemas(connection) -> List[Tuple[str]]:
    cursor = connection.cursor(cursor_factory=psycopg2.extras.DictCursor)
    cursor.execute(
```

```
        "SELECT schema_name FROM information_schema.schemata WHERE
schema_name NOT IN ('public', 'information_schema', 'pg_catalog',
'pg_toast', 'sku');"
    )
    return cursor.fetchall()


def fetch_products(connection, schema: str, id_concorrente) ->
List[Tuple]:
    cursor = connection.cursor(cursor_factory=psycopg2.extras.DictCursor)
    cursor.execute(
        f"SELECT * FROM {schema}.produto_novo WHERE id_concorrente =
{id_concorrente};"
    )
    return cursor.fetchall()


def insert_sku(connection, schema: str, data: Tuple):
    cursor = connection.cursor(cursor_factory=psycopg2.extras.DictCursor)
    cursor.execute(
        f"INSERT INTO {schema}.sku (ean, cod_ref, id_loja, slug) VALUES
(%s, %s, %s, %s)",
        data,
    )
    connection.commit()


def insert_product_not_found(produto: dict, schema: str):
    connection = create_connection()
    cursor = connection.cursor(cursor_factory=psycopg2.extras.DictCursor)
    cursor.execute(
        f"INSERT INTO {schema}.produto_nao_encontrado (ean, sku, produto,
departamento, categoria, marca, id_concorrente, nome_concorrente) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s)",
        (
            produto["ean"],
            produto["sku"],
            produto["produto"],
            produto["departamento"],
            produto["categoria"],
            produto["marca"],
            produto["id_concorrente"],
            produto["nome_concorrente"],
        ),
    )
    connection.commit()
    close_connection(connection)


def delete_sku(schema: str, id: str):
    connection = create_connection()
    cursor = connection.cursor(cursor_factory=psycopg2.extras.DictCursor)
    cursor.execute(f"DELETE FROM {schema}.produto_novo WHERE id = %s",
(id,))
```

```
    connection.commit()


def close_connection(connection):
    connection.close()


def extract_product_sku(url: str) -> str:
    parsed_url = urlparse(unquote(url))
    path_components = parsed_url.path.strip("/").split("/")
    if "dp" in path_components:
        dp_index = path_components.index("dp")
        if dp_index + 1 < len(path_components):
            return path_components[dp_index + 1]


def executar_consulta():
    connection = create_connection()

    try:
        schemas = fetch_schemas(connection)
        print(f"Schemas: {schemas}")

        for schema in schemas:
            produtos = fetch_products(connection, schema[0], 156)
            if not produtos:
                print("Nenhum resultado encontrado.")
                continue

            for produto in produtos:
                print("====================================")
                print(produto[3])
                slug = amazon_scraper(produto[3])
                print(slug)
                print("====================================")
                if slug is None:
                    insert_product_not_found(produto, schema[0])
                    delete_sku(schema[0], produto[0])
                else:
                    try:
                        sku = extract_product_sku(slug)
                        insert_sku(
                            connection,
                            schema[0],
                            (
                                produto[1],
                                sku,
                                produto[7],
                                slug,
                            ),
                        )
                        delete_sku(schema[0], produto[0])
                        print("SKU inserido com sucesso:", sku)
                        print("====================================")
```

```
                    except Exception as error:
                        print("Erro ao inserir sku:", error)
                        print(error)
                        print("====================================")
                        continue

    except Exception as error:
        print("Erro ao executar consulta:", error)

    finally:
        close_connection(connection)


executar_consulta()


FILEPATH: C:\Users\andwe\Downloads\classification-
scrapers\scrapers_template.py
import string
from datetime import datetime, timedelta
from typing import Dict, List, Optional

import nltk
import requests
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from unidecode import unidecode

nltk.download("punkt")
nltk.download("stopwords")

from airflow.models import Variable

default_args = {
    "owner": "airflow",
    "depends_on_past": False,
    "start_date": datetime(2023, 5, 2),
    "email": ["airflow@example.com"],
    "email_on_failure": False,
    "email_on_retry": False,
    "retries": 1,
    "retry_delay": timedelta(minutes=5),
}

dag = DAG(
    "scraper_X",
    default_args=default_args,
    description="Classificador de produtos X",
    schedule_interval="0 21 * * *",
)
```

```
def remove_punctuation(text: str) -> str:
    """
    Remove punctuation from the given text.

    Args:
        text (str): The input text.

    Returns:
        str: The text without punctuation.
    """
    translator = str.maketrans("", "", string.punctuation)
    return text.translate(translator)


def tokenize_and_lower(text: str) -> Set[str]:
    """
    Tokenize the input text and convert it to lowercase.

    Args:
        text (str): The input text.

    Returns:
        Set[str]: A set of unique lowercase tokens.
    """
    cleaned_text = remove_punctuation(text)
    tokens = nltk.word_tokenize(cleaned_text.lower())
    return set(tokens)


def calculate_min_intersection(query: str, percentage: float = 50.0) ->
int:
    """
    Calculate the minimum intersection value based on the input query and
percentage.

    Args:
        query (str): The input query.
        percentage (float, optional): The percentage to calculate the
minimum intersection. Defaults to 50.0.

    Returns:
        int: The minimum intersection value.
    """
    tokens = nltk.word_tokenize(query)
    return int(len(tokens) * (percentage / 100))


def find_most_similar(
    query: str, products: List[Dict[str, str]]
) -> Optional[Dict[str, str]]:
    query_tokens = tokenize_and_lower(query)
    max_intersection = 0
    most_similar_product = None
```

```
    min_intersection = calculate_min_intersection(query)

    for product in products:
        try:
            product_tokens = tokenize_and_lower(product["description"])
            intersection = len(query_tokens.intersection(product_tokens))

            if intersection >= min_intersection and intersection >
max_intersection:
                max_intersection = intersection
                most_similar_product = {
                    "url": product["url"],
                    "description": product["description"],
                }

        except Exception as error:
            print(f"Error processing product: {error}")
            continue

    return most_similar_product


def amazon_scraper(query: str) -> Optional[str]:
    query_tokens = tokenize_and_lower(query)
    SEARCH_PRODUCTS_URL = "https://SUAURL/SEUSPARAMENTROSDEBUSCA"

    # proxy = {"http": Variable.get('PROXY')} # usar ao subir para
airflow
    proxy = {
        "http": "http://Eiprice-cc-any:DQSXomtV7qri@gw.ntnt.io:5959"
    }  # usar local para testar

    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{SEARCH_PRODUCTS_URL}{query.replace(' ', '+')}"
    try:
        response = requests.get(url, headers=headers, proxies=proxy)
    except Exception as error:
        print("Erro ao fazer requisição:", error)
        return None

    soup = BeautifulSoup(response.content, "html.parser")
    product_list = soup.find_all(
        "div", class_="SUA CLASSE ONDE ESTA A GRADE DE PRODUTOS"
    )

    products = []

    for product_section in product_list:
```

```
        try:
            product = product_section.find(
                "div", class_="CLASSE DE CADA PRODUTO DA LISTA"
            )
            descricao = product.find("img", class_="CLASSE DA
IMAGEM")["alt"]
            link_section = product_section.find("a", class_="CLASSE DE
SEU LINK")
            link = f"https://www.amazon.com.br{link_section['href']}"
            img_link = product.find("img", class_="CLASSE DO LINK DA
IMAGEM")["src"]
            sku = link.split("/dp/")[1].split("/")[
                0
            ]  # GERALMENTE ENCONTRAMOS SKU NAS URLS, MAS EM ALGUNS SITES
PRECISAMOS ANALISAR CODIGO FONTE/CLASSES...

            products.append(
                {
                    "url": link,
                    "description": descricao,
                    "sku": sku,
                    "img_link": img_link,
                }
            )

        except Exception as error:
            print("Erro ao buscar produto:", error)
            continue

    if most_similar_product := find_most_similar(query_tokens, products):
        print("Produto mais similar encontrado:")
        print(most_similar_product["description"],
most_similar_product["url"])
        return most_similar_product["url"]
    else:
        print("Nenhum produto similar encontrado.")
        return None


def executar_consulta():
    # connection_hook =
PostgresHook(postgres_conn_id=Variable.get('POSTGRES_DB')) # usar ao
subir para airflow
    connection_hook = PostgresHook(postgres_conn_id="eiprice-dev")  #
usar local
    connection = connection_hook.get_conn()
    cursor = connection.cursor()

    try:
        cursor.execute(
            "SELECT ean, sku, id_loja, atributo, status, descricao FROM
kabum.produto_carga;"
        )
        resultados = cursor.fetchall()
```

```
        for resultado in resultados:
            ean, sku, id_loja, atributo, status, descricao = resultado
            print("====================================")
            print(resultado[5])
            slug = amazon_scraper(unidecode(resultado[5]))
            print(slug)
            print("====================================")

            try:
                cursor.execute(
                    "INSERT INTO kabum.sku (ean, cod_ref, id_loja,
atributo, status, slug) VALUES (%s, %s, %s, %s, %s, %s)",
                    (ean, sku, id_loja, atributo, status, slug),
                )
            except Exception as error:
                print("Erro ao inserir sku:", sku)
                print(error)
                print("====================================")
                continue

    except Exception as error:
        print("Erro ao executar consulta:", error)

    finally:
        connection.commit()
        connection.close()


run_query = PythonOperator(
    task_id="run_query",
    python_callable=executar_consulta,
    dag=dag,
)

run_query


def run_X_scraper_local(query: str) -> None:
    """
    Run the X scraper locally with the given query.

    Args:
        query (str): The input query.
    """
    print("Running X scraper locally...")
    print("Query:", query)

    SEARCH_PRODUCTS_URL = "https://www.X.com.br/"
    PROXY = "http://Eiprice-cc-any:DQSXomtV7qri@gw.ntnt.io:5959"
    HEADERS = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
```

```
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{SEARCH_PRODUCTS_URL}{query.replace(' ', '+')}"
    response = requests.get(url, headers=HEADERS, proxies={PROXY})

    soup = BeautifulSoup(response.content, "html.parser")
    product_list = soup.find_all(
        "div", class_="s-main-slot s-result-list s-search-results sg-row"
    )

    products = []

    for product_section in product_list:
        try:
            product = product_section.find(
                "div", class_="a-section aok-relative s-image-square-
aspect"
            )
            descricao = product.find("img", class_="s-image")["alt"]
            link_section = product_section.find(
                "a", class_="a-link-normal s-no-outline"
            )
            link = f"{SEARCH_PRODUCTS_URL}{link_section['href']}"

            products.append({"url": link, "description": descricao})

        except Exception as error:
            print("Erro ao buscar produto:", error)
            continue

    if most_similar_product := find_most_similar(query, products):
        print("Produto mais similar encontrado:")
        print(most_similar_product["url"])
    else:
        print("Nenhum produto similar encontrado.")


if __name__ == "__main__":
    sample_query = "COMPUTADOR GAMER 128GB RAM SSD HD 1TB NOVO"
    run_X_scraper_local(sample_query)


FILEPATH: C:\Users\andwe\Downloads\classification-
scrapers\scraper_amazon.py
import ssl
import string
from datetime import datetime, timedelta
from typing import Dict, List, Optional, Set, Tuple

import nltk
import requests
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
```

```
from airflow.providers.postgres.hooks.postgres import PostgresHook
from bs4 import BeautifulSoup
from nltk.corpus import stopwords

from urllib.parse import urlparse, unquote


nltk.download("punkt")
nltk.download("stopwords")

from airflow.models import Variable

try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    pass
else:
    ssl._create_default_https_context = _create_unverified_https_context

default_args = {
    "owner": "airflow",
    "depends_on_past": False,
    "start_date": datetime(2023, 5, 2),
    "email": ["airflow@example.com"],
    "email_on_failure": False,
    "email_on_retry": False,
    "retries": 1,
    "retry_delay": timedelta(minutes=5),
}

dag = DAG(
    "scraper_amazon",
    default_args=default_args,
    description="Classificador de produtos da Amazon",
    schedule_interval="0 21 * * *",
)


def remove_punctuation(text: str) -> str:
    """
    Remove punctuation from the given text.

    Args:
        text (str): The input text.

    Returns:
        str: The text without punctuation.
    """
    translator = str.maketrans("", "", string.punctuation)
    return text.translate(translator)



def tokenize_and_lower(text: str) -> Set[str]:
```

```
    """
    Tokenize the input text and convert it to lowercase.

    Args:
        text (str): The input text.

    Returns:
        Set[str]: A set of unique lowercase tokens.
    """
    cleaned_text = remove_punctuation(text)
    tokens = nltk.word_tokenize(cleaned_text.lower())
    return set(tokens)


def calculate_min_intersection(query: str, percentage: float = 50.0) ->
int:
    """
    Calculate the minimum intersection value based on the input query and
percentage.

    Args:
        query (str): The input query.
        percentage (float, optional): The percentage to calculate the
minimum intersection. Defaults to 50.0.

    Returns:
        int: The minimum intersection value.
    """
    tokens = nltk.word_tokenize(query)
    return int(len(tokens) * (percentage / 100))


def find_most_similar(
    query: str, products: List[Dict[str, str]]
) -> Optional[Dict[str, str]]:
    query_tokens = tokenize_and_lower(query)
    max_intersection = 0
    most_similar_product = None
    min_intersection = calculate_min_intersection(query)

    for product in products:
        try:
            product_tokens = tokenize_and_lower(product["description"])
            intersection = len(query_tokens.intersection(product_tokens))

            if intersection >= min_intersection and intersection >
max_intersection:
                max_intersection = intersection
                most_similar_product = {
                    "url": product["url"],
                    "description": product["description"],
                }

        except Exception as error:
```

```
            print(f"Error processing product: {error}")
            continue

    return most_similar_product


def amazon_scraper(query: str) -> Optional[str]:
    SEARCH_PRODUCTS_URL = "https://www.amazon.com.br/s?k="

    proxy = {"http": Variable.get("PROXY")}

    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{SEARCH_PRODUCTS_URL}{query.replace(' ', '+')}"
    try:
        response = requests.get(url, headers=headers, proxies=proxy)
    except Exception as error:
        print("Erro ao fazer requisição:", error)
        return None

    soup = BeautifulSoup(response.content, "html.parser")
    product_list = soup.find_all(
        "div", class_="s-main-slot s-result-list s-search-results sg-row"
    )

    products = []

    for product_section in product_list:
        try:
            product = product_section.find(
                "div", class_="a-section aok-relative s-image-square-
aspect"
            )
            descricao = product.find("img", class_="s-image")["alt"]
            link_section = product_section.find(
                "a", class_="a-link-normal s-no-outline"
            )
            link = f"https://www.amazon.com.br{link_section['href']}"
            # img_link = product.find("img", class_="s-image")["src"]
            # sku = link.split("/dp/")[1].split("/")[0]

            products.append({"url": link, "description": descricao})

        except Exception as error:
            print("Erro ao buscar produto:", error)
            continue

    if most_similar_product := find_most_similar(query, products):
        print("Produto mais similar encontrado:")
```

```
        print(most_similar_product["url"])
        return most_similar_product["url"]
    else:
        print("Nenhum produto similar encontrado.")
        return None


####
def get_connection():
    """
    Get the PostgresHook connection object.

    Returns:
        Connection object.
    """
    return
PostgresHook(postgres_conn_id=Variable.get("POSTGRES_DB")).get_conn()


def fetch_schemas() -> List[str]:
    connection = get_connection()
    cursor = connection.cursor()
    cursor.execute(
        "SELECT schema_name FROM information_schema.schemata WHERE
schema_name NOT IN ('public', 'information_schema', 'pg_catalog',
'pg_toast', 'sku');"
    )
    # Transforma a lista de tuplas em uma lista de strings
    return [result[0] for result in cursor.fetchall()]



def fetch_products(schema: str, id_concorrente) -> List[Tuple]:
    connection = get_connection()
    cursor = connection.cursor()
    cursor.execute(
        f"SELECT * FROM {schema}.produto_novo WHERE id_concorrente =
{id_concorrente};"
    )
    return cursor.fetchall()


def insert_sku(schema: str, data: Tuple):
    connection = get_connection()
    cursor = connection.cursor()
    cursor.execute(
        f"INSERT INTO {schema}.sku (ean, cod_ref, id_loja, slug) VALUES
(%s, %s, %s, %s)",
        data,
    )
    connection.commit()


def insert_product_not_found(produto: dict, schema: str):
```

```
    connection = get_connection()
    cursor = connection.cursor()
    cursor.execute(
        f"INSERT INTO {schema}.produto_nao_encontrado (ean, sku, produto,
departamento, categoria, marca, id_concorrente, nome_concorrente) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s)",
        (
            produto["ean"],
            produto["sku"],
            produto["produto"],
            produto["departamento"],
            produto["categoria"],
            produto["marca"],
            produto["id_concorrente"],
            produto["nome_concorrente"],
        ),
    )
    connection.commit()
    close_connection(connection)


def delete_sku(schema: str, id: str):
    connection = get_connection()
    cursor = connection.cursor()
    cursor.execute(f"DELETE FROM {schema}.produto_novo WHERE id = %s",
(id,))
    connection.commit()


def close_connection():
    connection = get_connection()
    connection.close()


def extract_product_sku(url: str) -> str:
    parsed_url = urlparse(unquote(url))
    path_components = parsed_url.path.strip("/").split("/")
    if "dp" in path_components:
        dp_index = path_components.index("dp")
        if dp_index + 1 < len(path_components):
            return path_components[dp_index + 1]


def executar_consulta():
    try:
        schemas = fetch_schemas()
        print(f"Schemas: {schemas}")

        for schema_name in schemas:
            try:
                produtos = fetch_products(schema_name, 156)
                if not produtos:
                    print("Nenhum resultado encontrado.")
                    continue
```

```
                for produto in produtos:
                    print("====================================")
                    print(produto[3])
                    slug = amazon_scraper(produto[3])
                    print(slug)
                    print("====================================")
                    if slug is None:
                        insert_product_not_found(produto, schema_name)
                        delete_sku(schema_name, produto[0])
                    try:
                        sku = extract_product_sku(slug)
                        insert_sku(
                            schema_name,
                            (
                                produto[1],
                                sku,
                                produto[7],
                                slug,
                            ),
                        )
                        delete_sku(schema_name, produto[0])
                        print("SKU inserido com sucesso:", sku)
                        print("====================================")
                    except Exception as error:
                        print("Erro ao inserir sku:", error)
                        print(error)
                        print("====================================")
            except Exception as schema_error:
                print(f"Erro ao processar o schema {schema_name}:",
schema_error)
                print("====================================")
    except Exception as error:
        print("Erro ao executar consulta:", error)


run_query = PythonOperator(
    task_id="run_query",
    python_callable=executar_consulta,
    dag=dag,
)

run_query


def run_amazon_scraper_local(query: str) -> None:
    """
    Run the Amazon scraper locally with the given query.

    Args:
        query (str): The input query.
    """
    print("Running Amazon scraper locally...")
    print("Query:", query)
```

```
    SEARCH_PRODUCTS_URL = "https://www.amazon.com.br/s?k="
    PROXY = "http://Eiprice-cc-any:DQSXomtV7qri@gw.ntnt.io:5959"
    HEADERS = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{SEARCH_PRODUCTS_URL}{query.replace(' ', '+')}"
    response = requests.get(url, headers=HEADERS, proxies={PROXY})

    soup = BeautifulSoup(response.content, "html.parser")
    product_list = soup.find_all(
        "div", class_="s-main-slot s-result-list s-search-results sg-row"
    )

    products = []

    for product_section in product_list:
        try:
            product = product_section.find(
                "div", class_="a-section aok-relative s-image-square-
aspect"
            )
            descricao = product.find("img", class_="s-image")["alt"]
            link_section = product_section.find(
                "a", class_="a-link-normal s-no-outline"
            )
            link = f"https://www.amazon.com.br{link_section['href']}"

            products.append({"url": link, "description": descricao})

        except Exception as error:
            print("Erro ao buscar produto:", error)
            continue

    if most_similar_product := find_most_similar(query, products):
        print("Produto mais similar encontrado:")
        print(most_similar_product["url"])
    else:
        print("Nenhum produto similar encontrado.")


if __name__ == "__main__":
    sample_query = "COMPUTADOR GAMER 128GB RAM SSD HD 1TB NOVO"
    run_amazon_scraper_local(sample_query)


FILEPATH: C:\Users\andwe\Downloads\classification-
scrapers\scraper_americanas.py
from datetime import datetime, timedelta
```

```
import nltk
import requests
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from bs4 import BeautifulSoup
from unidecode import unidecode

nltk.download("punkt")

default_args = {
    "owner": "airflow",
    "depends_on_past": False,
    "start_date": datetime(2023, 5, 2),
    "email": ["airflow@example.com"],
    "email_on_failure": False,
    "email_on_retry": False,
    "retries": 1,
    "retry_delay": timedelta(minutes=5),
}

dag = DAG(
    "scraper_americanas",
    default_args=default_args,
    description="Classificador de produtos Americanas",
    schedule_interval="0 21 * * *",
)

nltk.download("punkt")


def americanas_scraper(query):
    query_tokens = set(nltk.word_tokenize(query.lower()))
    max_intersection = 0
    most_similar_product = None
    SEARCH_PRODUCTS_URL = "https://www.americanas.com.br/busca/"
    MIN_INTERSECTION = 2

    proxy = {"http": "http://Eiprice-cc-
any:DQSXomtV7qri@gw.ntnt.io:5959"}

    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{SEARCH_PRODUCTS_URL}{query.replace(' ', '-')}"
    response = requests.get(url, headers=headers, proxies=proxy)

    base_url = "https://www.americanas.com.br"

    soup = BeautifulSoup(response.content, "html.parser")
```

```
    product_list = soup.find_all("div", class_="grid__StyledGrid-sc-
1man2hx-0")

    for produto in product_list:
        try:
            for a in soup.find_all("a", class_="inStockCard__Link-sc-
1ngt5zo-1 JOEpk"):
                link = a["href"].split(" ")[0]
                url = base_url + link
                descricao = a.find(
                    "h3", class_="product-name__Name-sc-1shovj0-0 gUjFDF"
                ).text

                product_tokens =
set(nltk.word_tokenize(descricao.lower()))
                intersection =
len(query_tokens.intersection(product_tokens))

                if intersection > max_intersection:
                    max_intersection = intersection
                    most_similar_product = {"url": url, "alt_text":
descricao}
        except Exception as error:
            print("====================================")
            print(f"Erro ao buscar produto: {produto}")
            print(error)
            print("====================================")
            continue

    if most_similar_product:
        print("Produto mais similar encontrado:")
        print(most_similar_product["url"])
        return most_similar_product["url"]
    else:
        print("Nenhum produto similar encontrado.")


def executar_consulta():
    connection_hook = PostgresHook(postgres_conn_id="eiprice-dev")
    connection = connection_hook.get_conn()
    cursor = connection.cursor()
    cursor.execute(
        "SELECT ean, sku, id_loja, atributo, status, descricao FROM
kabum.produto_carga;"
    )
    resultados = cursor.fetchall()
    for resultado in resultados:
        ean, sku, id_loja, atributo, status, descricao = resultado
        print("====================================")
        print(resultado[5])
        slug = americanas_scraper(unidecode(resultado[5]))
        print(slug)
        print("====================================")
        try:
```

```
            cursor.execute(
                "INSERT INTO kabum.sku (ean, cod_ref, id_loja, atributo,
status, slug) VALUES (%s, %s, %s, %s, %s, %s)",
                (ean, sku, id_loja, atributo, status, slug),
            )
        except Exception as error:
            print("====================================")
            print(f"Erro ao inserir sku: {sku}")
            print(error)
            print("====================================")
            continue
    connection.commit()
    connection.close()


run_query = PythonOperator(
    task_id="run_query",
    python_callable=executar_consulta,
    dag=dag,
)

run_query


FILEPATH: C:\Users\andwe\Downloads\classification-
scrapers\scraper_carrefour.py
import ssl
from datetime import datetime, timedelta
from urllib.parse import quote

import nltk
import requests
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from bs4 import BeautifulSoup
from unidecode import unidecode

nltk.download("punkt")

try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    pass
else:
    ssl._create_default_https_context = _create_unverified_https_context

default_args = {
    "owner": "airflow",
    "depends_on_past": False,
    "start_date": datetime(2023, 5, 2),
    "email": ["airflow@example.com"],
    "email_on_failure": False,
    "email_on_retry": False,
```

```
    "retries": 1,
    "retry_delay": timedelta(minutes=5),
}

dag = DAG(
    "scraper_carrefour",
    default_args=default_args,
    description="Classificador de produtos do Carrefour",
    schedule_interval="0 21 * * *",
)


def carrefour_scraper(query):
    query_encoded = quote(query, safe="")
    query_tokens = set(nltk.word_tokenize(query.lower()))
    max_intersection = 0
    most_similar_product = None
    SEARCH_PRODUCTS_URL = "https://www.carrefour.com.br/busca/"
    MIN_SIMILARITY = 0.6
    MIN_INTERSECTION = 2

    proxy = {"http": "http://Eiprice-cc-
any:DQSXomtV7qri@gw.ntnt.io:5959"}

    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{SEARCH_PRODUCTS_URL}{query_encoded}"
    response = requests.get(url, headers=headers, proxies=proxy)

    soup = BeautifulSoup(response.content, "html.parser")
    product_list = soup.find_all(
        "article",
        class_="vtex-product-summary-2-x-element pointer pt3 pb4 flex
flex-column h-100",
    )
    href_list = soup.find_all(
        "a", class_="vtex-product-summary-2-x-clearLink h-100 flex flex-
column"
    )

    for product_index, product in enumerate(product_list):
        descricao = product.find(
            "img",
            class_="vtex-product-summary-2-x-imageNormal vtex-product-
summary-2-x-image",
        )["alt"]
        product_tokens = set(nltk.word_tokenize(descricao.lower()))
        intersection = len(query_tokens.intersection(product_tokens))
```

```
        if intersection > max_intersection:
            max_intersection = intersection
            most_similar_product = {
                "url": href_list[product_index]["href"],
                "alt_text": descricao,
            }

    if most_similar_product:
        print("Produto mais similar encontrado:")
        print(most_similar_product)
        return most_similar_product["url"]
    else:
        print("Nenhum produto similar encontrado.")


def executar_consulta():
    connection_hook = PostgresHook(postgres_conn_id="eiprice-dev")
    connection = connection_hook.get_conn()
    cursor = connection.cursor()
    cursor.execute(
        "SELECT ean, sku, id_loja, atributo, status, descricao FROM
kabum.produto_carga;"
    )
    resultados = cursor.fetchall()
    for resultado in resultados:
        ean, sku, id_loja, atributo, status, descricao = resultado
        print("====================================")
        print(resultado[5])
        slug = carrefour_scraper(unidecode(resultado[5]))
        print(slug)
        print("====================================")
        try:
            cursor.execute(
                "INSERT INTO kabum.sku (ean, cod_ref, id_loja, atributo,
status, slug) VALUES (%s, %s, %s, %s, %s, %s)",
                (ean, sku, id_loja, atributo, status, slug),
            )
        except Exception as error:
            print("====================================")
            print(f"Erro ao inserir sku: {sku}")
            print(error)
            print("====================================")
            continue
    connection.commit()
    connection.close()


run_query = PythonOperator(
    task_id="run_query",
    python_callable=executar_consulta,
    dag=dag,
)

run_query
```

```
FILEPATH: C:\Users\andwe\Downloads\classification-
scrapers\scraper_casasbahia.py
from datetime import datetime, timedelta

import nltk
import requests
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from bs4 import BeautifulSoup
from unidecode import unidecode

nltk.download("punkt")

default_args = {
    "owner": "airflow",
    "depends_on_past": False,
    "start_date": datetime(2023, 5, 2),
    "email": ["airflow@example.com"],
    "email_on_failure": False,
    "email_on_retry": False,
    "retries": 1,
    "retry_delay": timedelta(minutes=5),
}

dag = DAG(
    "scraper_casasbahia",
    default_args=default_args,
    description="Classificador de produtos da Casas Bahia",
    schedule_interval="0 21 * * *",
)

BASE_URL = "https://www.casasbahia.com.br"
MIN_INTERSECTION = 2


def casas_bahia_scraper(query):
    query_tokens = set(nltk.word_tokenize(query.lower()))
    max_intersection = 0
    most_similar_product = None

    proxy = {"http": "http://Eiprice-cc-
any:DQSXomtV7qri@gw.ntnt.io:5959"}

    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{BASE_URL}/{query.replace(' ', '-')}/b"
```

```
    response = requests.get(url, headers=headers, proxies=proxy)

    soup = BeautifulSoup(response.content, "html.parser")
    product_list = soup.find_all("div", class_="sc-18eb4054-0 dPlWZd")

    for product in product_list:
        try:
            descricao = product.find("img", class_="sc-d2913f46-0
htwjrw")["alt"]
            link = product.find("a", class_="sc-2b5b888e-1
cflebu")["href"]

            product_tokens = set(nltk.word_tokenize(descricao.lower()))
            intersection = len(query_tokens.intersection(product_tokens))

            if intersection > max_intersection:
                max_intersection = intersection
                most_similar_product = {"url": link, "alt_text":
descricao}
        except Exception as error:
            print("====================================")
            print(f"Erro ao buscar produto: {product}")
            print(error)
            print("Nenhum produto similar encontrado.")
            print("====================================")
            continue

    if most_similar_product:
        print("Produto mais similar encontrado:")
        print(most_similar_product["url"])
        return most_similar_product["url"]
    else:
        print("Nenhum produto similar encontrado.")


def executar_consulta():
    connection_hook = PostgresHook(postgres_conn_id="eiprice-dev")
    connection = connection_hook.get_conn()
    cursor = connection.cursor()
    cursor.execute(
        "SELECT ean, sku, id_loja, atributo, status, descricao FROM
kabum.produto_carga;"
    )
    resultados = cursor.fetchall()
    for resultado in resultados:
        ean, sku, id_loja, atributo, status, descricao = resultado
        print("====================================")
        print(resultado[5])
        slug = casas_bahia_scraper(unidecode(resultado[5]))
        print(slug)
        print("====================================")
        try:
            cursor.execute(
```

```
                "INSERT INTO kabum.sku (ean, cod_ref, id_loja, atributo,
status, slug) VALUES (%s, %s, %s, %s, %s, %s)",
                (ean, sku, id_loja, atributo, status, slug),
            )
        except Exception as error:
            print("====================================")
            print(f"Erro ao inserir sku: {sku}")
            print(error)
            print("====================================")
            continue
    connection.commit()
    connection.close()


run_query = PythonOperator(
    task_id="run_query",
    python_callable=executar_consulta,
    dag=dag,
)

run_query


FILEPATH: C:\Users\andwe\Downloads\classification-scrapers\scraper_cec.py
from datetime import datetime, timedelta

import nltk
import requests
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from bs4 import BeautifulSoup
from unidecode import unidecode

nltk.download("punkt")

# DADOS DE QUANDO E COMO AIRFLOW VAI SER EXECUTADO
default_args = {
    "owner": "airflow",
    "depends_on_past": False,
    "start_date": datetime(2023, 5, 2),
    "email": ["airflow@example.com"],
    "email_on_failure": False,
    "email_on_retry": False,
    "retries": 1,
    "retry_delay": timedelta(minutes=5),
}

dag = DAG(
    "scraper_cec",
    default_args=default_args,
    description="Classificador de produtos do CEC",
    schedule_interval="0 21 * * *",
)
```

```
# SCRAPER
def cec_scraper(query):
    query_tokens = set(nltk.word_tokenize(query.lower()))
    max_intersection = 0
    most_similar_product = None
    SEARCH_PRODUCTS_URL = "https://www.cec.com.br"
    MIN_SIMILARITY = 0.6
    MIN_INTERSECTION = 2

    proxy = {"http": "http://Eiprice-cc-
any:onQoK56mACxA@gw.ntnt.io:5959"}

    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{SEARCH_PRODUCTS_URL}{'/busca?q='}{query.replace(' ',
'%20')}{'&ranking=2&topsearch=1'}"

    response = requests.get(url, headers=headers, proxies=proxy)

    soup = BeautifulSoup(response.content, "html.parser")
    product_list = soup.find_all(
        "div",
        class_="products",
    )
    for product in product_list:
        try:
            link =
f"{SEARCH_PRODUCTS_URL}{product.find('a',class_='photo',)['href']}"
            descricao = product.find(
                "a",
                class_="photo",
            )["title"]
            product_tokens = set(nltk.word_tokenize(descricao.lower()))
            intersection = len(query_tokens.intersection(product_tokens))

            if intersection > max_intersection:
                max_intersection = intersection
                most_similar_product = {"url": link, "alt_text":
descricao}
        except Exception as error:
            print("====================================")
            print(f"Erro ao buscar produto: {product}")
            print(error)
            print("====================================")
            continue

    if most_similar_product:
```

```
        print("Produto mais similar encontrado:")
        print(most_similar_product["url"])
        return most_similar_product["url"]
    else:
        print("Nenhum produto similar encontrado.")


def executar_consulta():
    connection_hook = PostgresHook(postgres_conn_id="eiprice-dev")
    connection = connection_hook.get_conn()
    cursor = connection.cursor()
    cursor.execute(
        "SELECT ean, sku, id_loja, atributo, status, descricao FROM
kabum.produto_carga;"
    )
    resultados = cursor.fetchall()
    for resultado in resultados:
        ean, sku, id_loja, atributo, status, descricao = resultado
        print("====================================")
        print(resultado[5])
        slug = cec_scraper(unidecode(resultado[5]))
        print(slug)
        print("====================================")
        try:
            cursor.execute(
                "INSERT INTO kabum.sku (ean, cod_ref, id_loja, atributo,
status, slug) VALUES (%s, %s, %s, %s, %s, %s)",
                (ean, sku, id_loja, atributo, status, slug),
            )
        except Exception as error:
            print("====================================")
            print(f"Erro ao inserir sku: {sku}")
            print(error)
            print("====================================")
            continue
    connection.commit()
    connection.close()


run_query = PythonOperator(
    task_id="run_query",
    python_callable=executar_consulta,
    dag=dag,
)

run_query


FILEPATH: C:\Users\andwe\Downloads\classification-
scrapers\scraper_leroy_merlin.py
from datetime import datetime, timedelta

import nltk
import requests
```

```
from airflow import DAG
from bs4 import BeautifulSoup

# from airflow.operators.python_operator import PythonOperator
# from airflow.providers.postgres.hooks.postgres import PostgresHook
from unidecode import unidecode

nltk.download("punkt")

# DADOS DE QUANDO E COMO AIRFLOW VAI SER EXECUTADO
# default_args = {
#     "owner": "airflow",
#     "depends_on_past": False,
#     "start_date": datetime(2023, 5, 2),
#     "email": ["airflow@example.com"],
#     "email_on_failure": False,
#     "email_on_retry": False,
#     "retries": 1,
#     "retry_delay": timedelta(minutes=5),
# }

# dag = DAG(
#     "scraper_cec",
#     default_args=default_args,
#     description="Classificador de produtos do CEC",
#     schedule_interval="0 21 * * *",
# )


# SCRAPER
def leroy_scraper(query):
    query_tokens = set(nltk.word_tokenize(query.lower()))
    max_intersection = 0
    most_similar_product = None
    SEARCH_PRODUCTS_URL = "https://www.leroymerlin.com.br/search?term="
    MIN_SIMILARITY = 0.6
    MIN_INTERSECTION = 2

    proxy = {"http": "http://Eiprice-cc-
any:onQoK56mACxA@gw.ntnt.io:5959"}

    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{SEARCH_PRODUCTS_URL}{'/busca?q='}{query.replace(' ',
'%20')}{'&searchType=default'}"
    print(url)
    response = requests.get(url, headers=headers, proxies=proxy)
    print(response.status_code)
    soup = BeautifulSoup(response.content, "html.parser")
```

```
    product = soup.find_all("script", {"type": "application/ld+json"})
    print(product)

    # product_list = soup.find_all(
    #     "div",
    #     class_="products",
    # )
    # for product in product_list:
    #     try:
    #         link =
f"{SEARCH_PRODUCTS_URL}{product.find('a',class_='photo',)['href']}"
    #         descricao = product.find(
    #             "a",
    #             class_="photo",
    #         )["title"]
    #         sku = link.split("produto=")[1]
    #         product_tokens = set(nltk.word_tokenize(descricao.lower()))
    #         intersection =
len(query_tokens.intersection(product_tokens))

    #         if intersection > max_intersection:
    #             max_intersection = intersection
    #             most_similar_product = {"url": link, "alt_text":
descricao}
    #     except Exception as error:
    #         print("====================================")
    #         print(f"Erro ao buscar produto: {product}")
    #         print(error)
    #         print("====================================")
    #         continue

    # if most_similar_product:
    #     print("Produto mais similar encontrado:")
    #     print(most_similar_product["url"])
    #     return most_similar_product["url"]
    # else:
    #     print("Nenhum produto similar encontrado.")


def executar_consulta():
    connection_hook = PostgresHook(postgres_conn_id="eiprice-dev")
    connection = connection_hook.get_conn()
    cursor = connection.cursor()
    cursor.execute(
        "SELECT ean, sku, id_loja, atributo, status, descricao FROM
kabum.produto_carga;"
    )
    resultados = cursor.fetchall()
    for resultado in resultados:
        ean, sku, id_loja, atributo, status, descricao = resultado
        print("====================================")
        print(resultado[5])
        slug = cec_scraper(unidecode(resultado[5]))
        print(slug)
```

```
        print("====================================")
        try:
            cursor.execute(
                "INSERT INTO kabum.sku (ean, cod_ref, id_loja, atributo,
status, slug) VALUES (%s, %s, %s, %s, %s, %s)",
                (ean, sku, id_loja, atributo, status, slug),
            )
        except Exception as error:
            print("====================================")
            print(f"Erro ao inserir sku: {sku}")
            print(error)
            print("====================================")
            continue
    connection.commit()
    connection.close()


leroy_scraper("PLACA DE VIDEO")
# run_query = PythonOperator(
#     task_id="run_query",
#     python_callable=executar_consulta,
#     dag=dag,
# )

# run_query


FILEPATH: C:\Users\andwe\Downloads\classification-
scrapers\scraper_magalu.py.py
from datetime import datetime, timedelta

import nltk
import requests
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from bs4 import BeautifulSoup
from unidecode import unidecode

nltk.download("punkt")

default_args = {
    "owner": "airflow",
    "depends_on_past": False,
    "start_date": datetime(2023, 5, 2),
    "email": ["airflow@example.com"],
    "email_on_failure": False,
    "email_on_retry": False,
    "retries": 1,
    "retry_delay": timedelta(minutes=5),
}

dag = DAG(
    "scraper_magalu",
```

```
    default_args=default_args,
    description="Classificador de produtos Magalu",
    schedule_interval="0 21 * * *",
)


def magalu_scraper(query):
    query_tokens = set(nltk.word_tokenize(query.lower()))
    max_intersection = 0
    most_similar_product = None
    SEARCH_PRODUCTS_URL = "https://www.magazineluiza.com.br/busca/"

    proxy = {"http": "http://Eiprice-cc-
any:DQSXomtV7qri@gw.ntnt.io:5959"}

    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{SEARCH_PRODUCTS_URL}{query.replace(' ', '+')}"

    try:
        response = requests.get(url, headers=headers, proxies=proxy)
        soup = BeautifulSoup(response.content, "html.parser")
        product_list = soup.find_all("div", class_="sc-eDvSVe koHJnT")

        for produto in product_list:
            for a in soup.find_all("li", class_="sc-ibdxON fwviCj"):
                link = a.find("a")["href"]
                descricao = a.find("a")["title"]

                product_tokens =
set(nltk.word_tokenize(descricao.lower()))
                intersection =
len(query_tokens.intersection(product_tokens))

                if intersection > max_intersection:
                    max_intersection = intersection
                    most_similar_product = {"url": link, "alt_text":
descricao}
    except Exception as error:
        print("Erro ao buscar produto:", error)

    if most_similar_product:
        print("Produto mais similar encontrado:")
        print(most_similar_product["url"])
        return most_similar_product["url"]
    else:
        print("Nenhum produto similar encontrado.")
```

```
def executar_consulta():
    connection_hook = PostgresHook(postgres_conn_id="eiprice-dev")
    connection = connection_hook.get_conn()
    cursor = connection.cursor()
    cursor.execute(
        "SELECT ean, sku, id_loja, atributo, status, descricao FROM
kabum.produto_carga;"
    )
    resultados = cursor.fetchall()
    for resultado in resultados:
        ean, sku, id_loja, atributo, status, descricao = resultado
        print("====================================")
        print(resultado[5])
        try:
            slug = magalu_scraper(unidecode(resultado[5]))
            print(slug)
            print("====================================")
            cursor.execute(
                "INSERT INTO kabum.sku (ean, cod_ref, id_loja, atributo,
status, slug) VALUES (%s, %s, %s, %s, %s, %s)",
                 (ean, sku, id_loja, atributo, status, slug),
            )
        except Exception as error:
            print("====================================")
            print(f"Erro ao inserir sku: {sku}")
            print(error)
            print("====================================")
            continue
    connection.commit()
    connection.close()


run_query = PythonOperator(
    task_id="run_query",
    python_callable=executar_consulta,
    dag=dag,
)

run_query


FILEPATH: C:\Users\andwe\Downloads\classification-
scrapers\scraper_mercadolivre.py
from datetime import datetime, timedelta

import nltk
import requests
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from bs4 import BeautifulSoup
from unidecode import unidecode

nltk.download("punkt")
```

```
default_args = {
    "owner": "airflow",
    "depends_on_past": False,
    "start_date": datetime(2023, 5, 2),
    "email": ["airflow@example.com"],
    "email_on_failure": False,
    "email_on_retry": False,
    "retries": 1,
    "retry_delay": timedelta(minutes=5),
}

dag = DAG(
    "scraper_mercadolivre",
    default_args=default_args,
    description="Classificador de produtos do Mercado Livre",
    schedule_interval="0 21 * * *",
)


def mercado_livre_scraper(query):
    query_tokens = set(nltk.word_tokenize(query.lower()))
    max_intersection = 0
    most_similar_product = None
    SEARCH_PRODUCTS_URL = "https://lista.mercadolivre.com.br/"
    MIN_SIMILARITY = 0.6
    MIN_INTERSECTION = 2

    proxy = {"http": "http://Eiprice-cc-
any:DQSXomtV7qri@gw.ntnt.io:5959"}

    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{SEARCH_PRODUCTS_URL}{query.replace(' ', '-')}"
    response = requests.get(url, headers=headers, proxies=proxy)

    soup = BeautifulSoup(response.content, "html.parser")
    product_list = soup.find_all(
        "section",
        class_="ui-search-results ui-search-results--without-disclaimer
shops__search-results",
    )

    for product in product_list:
        try:
            link = product.find(
                "a",
                class_="ui-search-item__group__element shops__items-
group-details ui-search-link",
```

```
            )["href"]
            descricao = product.find(
                "a",
                class_="ui-search-item__group__element shops__items-
group-details ui-search-link",
            )["title"]

            product_tokens = set(nltk.word_tokenize(descricao.lower()))
            intersection = len(query_tokens.intersection(product_tokens))

            if intersection > max_intersection:
                max_intersection = intersection
                most_similar_product = {"url": link, "alt_text":
descricao}
        except Exception as error:
            print("====================================")
            print(f"Erro ao buscar produto: {product}")
            print(error)
            print("====================================")
            continue

    if most_similar_product:
        print("Produto mais similar encontrado:")
        print(most_similar_product["url"])
        return most_similar_product["url"]
    else:
        print("Nenhum produto similar encontrado.")


def executar_consulta():
    connection_hook = PostgresHook(postgres_conn_id="eiprice-dev")
    connection = connection_hook.get_conn()
    cursor = connection.cursor()
    cursor.execute(
        "SELECT ean, sku, id_loja, atributo, status, descricao FROM
kabum.produto_carga;"
    )
    resultados = cursor.fetchall()
    for resultado in resultados:
        ean, sku, id_loja, atributo, status, descricao = resultado
        print("====================================")
        print(resultado[5])
        slug = mercado_livre_scraper(unidecode(resultado[5]))
        print(slug)
        print("====================================")
        try:
            cursor.execute(
                "INSERT INTO kabum.sku (ean, cod_ref, id_loja, atributo,
status, slug) VALUES (%s, %s, %s, %s, %s, %s)",
                (ean, sku, id_loja, atributo, status, slug),
            )
        except Exception as error:
            print("====================================")
            print(f"Erro ao inserir sku: {sku}")
```

```
            print(error)
            print("====================================")
            continue
    connection.commit()
    connection.close()


run_query = PythonOperator(
    task_id="run_query",
    python_callable=executar_consulta,
    dag=dag,
)

run_query
```