

Challenge para puesto Data Engineer

Franco Balich

Repositorio de GitHub

<https://github.com/francobalich/DataEngineer-Meli-Challenge>

Reporte de Looker Studio

<https://lookerstudio.google.com/reporting/557dcca1-5d9e-4bf2-bc33-58e907be071d>

1. Descargar la data

El código para descargar los datos lo pueden encontrar en:

src\mercadolibre\get_products.py

Al mismo lo pueden probar y obtener los productos en un jsonl en:

data\origin\productos.jsonl

Para cargar datos hay que llamar 10 veces al endpoint /getproducts

Curl de ejemplo

```
curl --location 'http://127.0.0.1:5000/getproducts' \  
--header 'Content-Type: application/json' \  
--data '{  
    "product": "pc"  
}'
```

Esto puede ser con el mismo producto o con distintos.

2. Modelar

Para este punto decidí analizar la respuesta y pasar a un modelo relacional todos los datos que aportan valor al modelo, por esto no cree tablas para Dominio y Categoria, que si bien tenían IDs en el producto, no tenían contenido, por lo que deje los IDs de Dominio y Categoria en Producto por si alguna vez se requiere crecer el modelo o hacer alguna consulta para completar la información con otro endpoint.

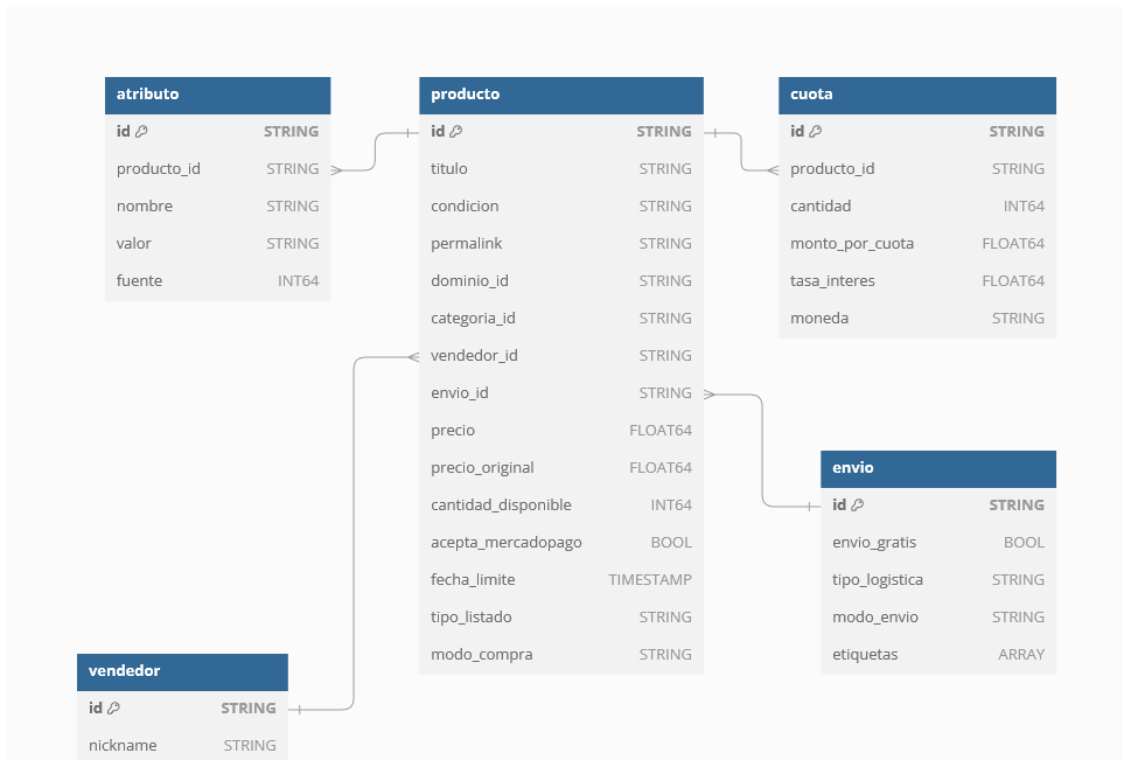
Por otro lado, no agregue otra tabla más sobre Promocion ya que no era algo recurrente en todos los productos, ya que varios no los tenían y prefería hacer mi análisis en otras características de los productos, siendo algunas de estas:

- Hacer una comparativa de precios entre productos
- Ver que vendedor tiene más productos
- Ver que producto tiene más atributos en sus publicaciones
- Ver que cantidad de cuotas es más recurrente
- Ver si hay más envíos gratis o pagos

El script para generar el modelo lo pueden ver en el path

data\model\dataset-gen.sql

Diagrama entidad relación



3.Parsear

El código del parseo lo pueden encontrar en `src\mercadolibre\product_processor.py` donde tomo los valores que me interesan el json `productos.jsonl` y genero los `jsonl` para cada tabla para luego importarlas en bigquery

Dichos archivos los pueden encontrar en los paths

- `data\conversions\atributo.jsonl`
- `data\conversions\cuota.jsonl`
- `data\conversions\envio.jsonl`
- `data\conversions\producto.jsonl`
- `data\conversions\vendedor.jsonl`

Para realizar la conversión deben ejecutar el siguiente curl:

```
curl --location --request POST 'http://localhost:5000/convertdata' \  
--data "
```

4.Looker

Para este punto el código que use para generar los views y los explorer en LookML se puede encontrar en el path

- `src\mercadolibre\lookml_generator.py`

y pueden ver los archivos convertidos en los paths:

- `data\lookml\atributo.view.lkml`
- `data\lookml\envio.view.lkml`
- `data\lookml\producto.view.lkml`

- data\lookml\vendedor.view.lkml
- data\lookml\producto.explore.lkml

Para realizar la conversión deben ejecutar el siguiente curl:

```
curl --location --request POST 'http://localhost:5000/genlookml' \
--data "
```

Programación orientada a objetos

Se creo una clase para cada funcionalidad que necesitaba con métodos para cada una.

Ejemplos de clases:

- getProducts
- LookMLGenerator
- ProductProcessor

Coverage

Agregue tests en la carpeta /tests

Los mismos se pueden ejecutar con

- `python -m unittest discover -s tests`

Además, genere un reporte del coverage en:

- `report\index.html`

Coverage report: 85%

Files Functions Classes

coverage.py v7.6.7, created at 2024-11-19 19:31 -0300

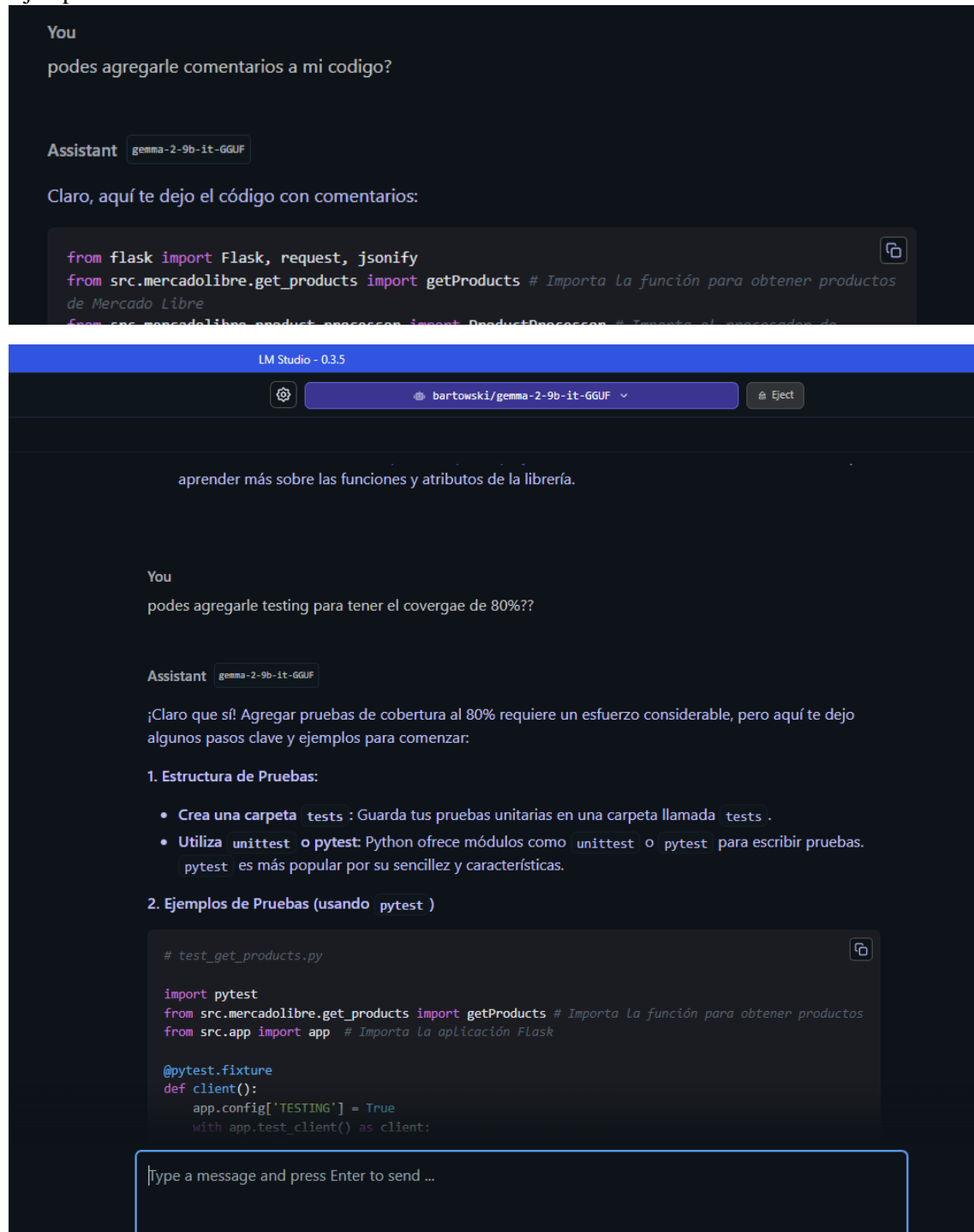
File ▲	statements	missing	excluded	coverage
src__init__.py	0	0	0	100%
src\mercadolibre__init__.py	0	0	0	100%
src\mercadolibre\api.py	15	3	0	80%
src\mercadolibre\lookml_generator.py	43	11	0	74%
src\mercadolibre\product_processor.py	61	16	0	74%
tests\test_get_products.py	23	0	0	100%
tests\test_lookml_generator.py	32	0	0	100%
tests\test_product_processor.py	28	0	0	100%
Total	202	30	0	85%

coverage.py v7.6.7, created at 2024-11-19 19:31 -0300

Uso de GenAI

Use un modelo open source de IA generativa para completar campos una vez que determine la estructura que quería de mis tablas, agregue comentarios de forma automatizada y la base de los testing para el coverage. Todo igual requirió revisiones porque lo que generaba no funcionaba a la primera, por ejemplo, había cambios de tipos que no debía ser así, test incompletos y errores en general que solucione para el proyecto.

Ejemplos de uso:



You

podes agregarle comentarios a mi codigo?

Assistant `gemma-2-9b-it-GGUF`

Claro, aquí te dejo el código con comentarios:

```
from flask import Flask, request, jsonify
from src.mercadolibre.get_products import getProducts # Importa la función para obtener productos de Mercado Libre
from src.mercadolibre.product_processor import ProductProcessor # Importa el procesador de productos
```

LM Studio - 0.3.5

`bartowski/gemma-2-9b-it-GGUF` `Eject`

aprender más sobre las funciones y atributos de la librería.

You

podes agregarle testing para tener el coverage de 80%??

Assistant `gemma-2-9b-it-GGUF`

¡Claro que sí! Agregar pruebas de cobertura al 80% requiere un esfuerzo considerable, pero aquí te dejo algunos pasos clave y ejemplos para comenzar:

- Estructura de Pruebas:**
 - **Crea una carpeta `tests`** : Guarda tus pruebas unitarias en una carpeta llamada `tests` .
 - **Utiliza `unittest` o `pytest`** : Python ofrece módulos como `unittest` o `pytest` para escribir pruebas. `pytest` es más popular por su sencillez y características.
- Ejemplos de Pruebas (usando `pytest`)**

```
# test_get_products.py

import pytest
from src.mercadolibre.get_products import getProducts # Importa la función para obtener productos
from src.app import app # Importa la aplicación Flask

@pytest.fixture
def client():
    app.config['TESTING'] = True
    with app.test_client() as client:
```

type a message and press Enter to send ...