1. **Ejercicios Python**

- **1.1** Imagina que en tu compañía llamada ImageDup se quiere probar el desempeño del lenguaje Python con la manipulación de imágenes y el performance en APIs. Todo esto porque en la empresa se ha trabajado previamente con lenguajes como C+, C++ y Java. Para esto tu technical lead te pide a ti como Data Engineer de la empresa que hagas una prueba de concepto (PoC) sobre la api **Imgflip**: https://imgflip.com/api . Esta api proporciona imágenes a través de un método GET (/get_memes) de memes aleatorios. Deberás obtener al menos una imagen y guardarla en formato .jpeg en tu computadora.

  Se te encomienda que generes un proceso sencillo corriendo un script de Python llamado **obtener_meme.py** y la imagen de salida debería llamarse **imagen_meme.jpeg.**

  Ayuda: Puedes hacer uso de la librería requests, PIL y io

❖ **Solución:**

obtener_meme.py

imagen_meme.jpeg [1] [2]

- **2.1** Recientemente en la startup donde trabajas se decidió comenzar a explorar las criptomonedas como posible mecanismo de inversión. Es por esto que el manager del equipo de data te sugiere a ti como Data Engineer de la empresa que investigues acerca del tema. Además menciona que existen muchas APIs gratuitas que proveen información en tiempo real del Bitcoin. Pero, él te sugiere que investigues acerca de la API Messari (https://data.messari.io/api/v1/assets/bitcoin/metrics) ya que es gratuita. Además te pide que para el final del día le muestres una implementación en Python de como extraer la información diariamente del precio del Bitcoin de forma recurrente. Las variables que se deberían capturar son fecha, volume_last_24_housr, open, close, low y high como se muestra a continuación:

```
{'fecha': '2023-07-10T19:21:23.001268478Z', 'volume_last_24_housr': 6482076059.128203, 'open': 30348.621341725313, 'low': 30348.621341725313, 'close': 30530.379477108152, 'high': 30530.379477108152}
```

❖ **Solución:**

El script de python debe llamarse **extraccion_bitcoin.py**

extraccion_bitcoin.py

2. Ejercicio SQL

1.1 En la última reunión el equipo de marketing de tu compañía levanto la idea de poder identificar si los ingenieros mayores de 30 años son los que realmente están comprando más productos en las tiendas, es por esto que te piden que con base en esta base de datos dada puedas dar respuesta a dos preguntas críticas para el negocio

a. Calcular las ventas totales y las llamadas totales realizadas a los clientes de la profesión de ingeniería

b. Generar otra consulta que calcule las mismas métricas para toda la base de clientes.

c. ¿Qué puedes concluir con respecto a la tasa de conversión entre los clientes de ingeniería frente a la base de clientes en general?

d. ¿Valdría la pena invertir en publicidad en el segmento de clientes que son ingenieros y mayores de 30 años?

❖ **Solución:**

Primero necesitamos crear las tablas de destino

```
-- CREACION DE TABLAS

CREATE TABLE customers(

  customerid INT primary key,

  name VARCHAR(50),

  occupation VARCHAR(50),

  email VARCHAR(50),
```

```
    company VARCHAR(50),
    phonenumber VARCHAR(20),
    age INT
);
CREATE TABLE agents(
    agentid INT primary key,
    name VARCHAR(50)
);
CREATE TABLE calls(
    callid INT primary key,
    agentid INT,
    customerid INT,
    pickedup SMALLINT,
    duration INT,
    productsold SMALLINT
);
```

Para los de ingeniería

```
SELECT SUM(ProductSold) AS TotalSales, COUNT(*) AS NCalls
FROM customers Cu
join calls Ca ON Ca.CustomerID = Cu.CustomerID
WHERE Occupation LIKE '%Engineer%'
```

Para los de NO ingeniería

```
SELECT SUM(ProductSold) AS TotalSales, COUNT(*) AS NCalls
FROM customers Cu
join calls Ca ON Ca.CustomerID = Cu.CustomerID
WHERE Occupation not LIKE '%Engineer%'
```

La tasa de conversión para ambos grupos es ~20.9%, lo que indica que los ingenieros no tienen más probabilidades de comprar nuestros productos que la población en general.

➢ **Apéndice: SQL**

- SELECT * FROM table_name *-- Seleccionar todas las columnas de la tabla*

- SELECT column_name(s) **FROM table_name** *-- Seleccionar algunas columnas de la tabla*

- SELECT DISTINCT column_name(s) **FROM table_name** *-- Seleccionar solo valores unicos*

- SELECT column_name(s) **FROM table_name** *-- Seleccionar data filtrada con la cláusula WHERE*

WHERE operador columna

AND operador columna

OR operador columna

- SELECT column_name(s) **FROM table_name** *-- Ordenar la data por múltiples columnas DESC para descendiente*

ORDER BY column_1, column_2 **DESC**, column_3 **ASC** *-- y ASC (opcional) para orden ascendente*

➢ **Operadores**

<	Menor que
>	Mayor que
<=	Menor o igual
>=	Mayor o igual
<>	Diferente

=	Igual
BETWEEN v1 AND v2	Entre un rango específico
LIKE	Patron de búsqueda. Usar '%' como wildcard. E.g., '%o%' hace match con o, bob, blob, etc.

➢ **Funciones de agregación**

- **AVG(column)** - Promedio de columna
- **COUNT(column)** - Número de filas sin valores nulos de una columna
- **MAX(column)** - Máximo de una columna
- **MIN(column)** - Mínimo de una columna

- **SUM(column)** - Suma de una columna

SELECT AVG(column_name), MIN(column_name), MAX(column_name) FROM table_name

Misc.

- **CASE-END** - Usado en **SELECT** para alterar variables E.g.

```
SELECT column_name
    CASE
    WHEN column_name >= 0 THEN 'POSITIVO'
    ELSE 'NEGATIVO'
```

END

FROM table

- **AS** - Usado para re-nombrar una variable E.g.

SELECT SUM(column_name) AS total_column_name **FROM table_name**

- **GROUP BY** - Usado para agrupar filas que tengan el mismo valor(es) en una columna (s). En la mayoría de los casos se usa con funciones de agregación
- **ORDER BY** - Determina el orden en el cual las filas se devuelven en la query