

Módulo 13: Almacenes de datos: Datalakes

Capstone XIII - Orquestación de ETLs en un Datalake

Javier Cózar

Máster en Ciencia de Datos e Ingeniería de Datos en la Nube
Universidad de Castilla-La Mancha

Marta Bellón Castro
Curso 2022-2023

NobelPrice

En este capstone vamos a trabajar con los datos de las celebraciones de premios Nobel y las personas y entidades laureadas. Para ello haremos uso de la API de nobelprize.org. Esta API tiene documentación en formato OpenAPI alojado en la plataforma swaggerhub, que es un estandar bastante extendido y como podréis comprobar es muy cómodo e interesante de explorar!

Vamos a crear un datalake usando **AWS S3**, a implementar nuestras ETLs en **AWS Lambda**, y a orquestar su ejecución a través de **Prefect**, ejecutando el flujo localmente. A continuación describimos qué ETLs implementar y los pasos a seguir para orquestar las ETLs con Prefect.

Descripción

Los premios Nobel se celebran anualmente (salvo algún problema, como ocurrió durante *la segunda guerra mundial*). Vamos a automatizar la extracción de los datos todos los **1 de enero de cada año** haciendo lo siguiente:

- Descargaremos los datos **del año anterior para todas las categorías** (`che` , `eco` , `lit` , `pea` , `phy` , `med`)
- Almacenaremos en el datalake (prefijo `raw`) estos datos en formato parquet
- Ejecutaremos las ETL implementadas (bronze y silver)

Finalmente, usaremos la libreta de jupyter para acceder a las tablas generadas por la ETL y **responder ciertas preguntas de analítica**.

Ejercicio 1. Crear el bucket de S3 que será nuestro datalake

Vamos a crear un bucket de S3 en nuestra cuenta de AWS Academy. Este bucket contendrá los prefijos:

- **raw**: contendrá los datos en crudo extraídos de la API.
- **bronze**: contendrá tablas que serán uniones de tablas con nombres de columnas posiblemente modificadas y tipos de columnas casteadas.
- **silver**: contendrá tablas de alto nivel que serán utilizadas para acceder a la información granularmente.

Ejercicio 2. Crear la layer para AWS Lambda

Vamos a utilizar `pandas` , `pyarrow` , `requests` y `s3fs` en todas nuestras lambdas. La consola tiene un límite de 50MB por zip, pero nuestra layer es mayor. Para poder crearla hay que subir el zip `layer.zip` a un bucket de S3 y crear la layer a partir de la S3 URI. La layer está creada para `python 3.8` y arquitectura `x86_64` .

NOTA: podéis ver las versiones de los paquetes utilizados en las layers para lambda en el fichero `requirements_lambda.txt`. Hay que usar la misma versión de python y de los paquetes para desarrollar en local y así evitar errores (podéis probar el código dentro de una imagen de docker si fuese necesario).

```
In [1]: """
!pip install urllib3==1.26.7
!pip install requests
!pip install numpy==1.24.3
!pip install pandas==1.5.3
!pip install fastparquet==2023.2.0
!pip install requests==2.28.2
!pip install botocore==1.29.76
!pip install aiobotocore==2.5.0
!pip install fsspec==2023.4.0
!pip install s3fs==2023.4.0
"""

#!pip list #Compruebo que tengo todos los paquetes instalados para poder hacer la práctica

Out[1]: '\n!pip install urllib3==1.26.7\n!pip install requests\n!pip install numpy==1.24.3\n!pip install pandas==1.5.3\n!pip
install fastparquet==2023.2.0\n!pip install requests==2.28.2\n!pip install botocore==1.29.76\n!pip install aiobotocor
e==2.5.0\n!pip install fsspec==2023.4.0\n!pip install s3fs==2023.4.0\n'
```

Ejercicio 3. Crear las AWS Lambdas de extracción

Las funciones deben extraer información de `nobelPrizes` y `laureates`, es decir, premios Nobel y laureados o personas premiadas. Para ello:

1. Usar el endpoint `http://api.nobelprize.org/2.1/nobelPrize/{category}/{year}` para obtener la información de todos los premios Nobel para una categoría y un año concreto.
2. Usar el endpoint `http://api.nobelprize.org/2.1/laureate/{laureateID}` para obtener la información de un laureado en base a su ID.

Los datos serán almacenados en el datalake en formato parquet, que permite leer cómodamente varios ficheros con una llamada `pd.read_parquet` y podríamos ejecutar procesos de compactación de datos cuando fuese necesario. Sin embargo, hay ciertas estructuras complejas que parquet no tiene implementado en su versión en python (algunas de ellas nos las encontramos en las respuestas de estas APIs). Por ello, en lugar de almacenar los datos directamente en crudo los procesaremos ligeramente con el fin de generar datos estructurados. Es importante destacar que esto se realiza **para simplificar el proceso**, ya que es importante almacenar los datos en crudo, tal cual los extraemos, para su posterior uso. Una práctica más realista sería almacenar el json tal cual como raw, y usar una lambda para crear un raw_processed que sería el parquet que estamos generando ahora.

Importante: selecciona la arquitectura `x86_64` y la versión de python 3.8. Finalmente, debes seleccionar el **rol de lam** existente llamado `LabRole`. También configuraremos el timeout de la lambda a 1 minuto y 2048 MB de RAM para asegurarnos que le da tiempo a ejecutarse correctamente.

```
In [2]: import boto3

# Configuro las credenciales para acceder a mi AWS
aws_access_key_id = 'ASIATLUAUS55WDDY7HUK'
aws_secret_access_key = 'YAqEm7xmMkfyiQ3+RVVkeFx1EGAnlZqPH6LHT6FF'
aws_session_token = 'FwoGZXIvYXZzEHIAaDLxhbZbVGm1xmifwNCK9AdhBvlgRjMx449oIiwVG6RsJypV9T3o2QSp0XuNI8hiMqzKv10sBLdBldno9i

# Cliente S3
s3 = boto3.client('s3', aws_access_key_id=aws_access_key_id, aws_secret_access_key=aws_secret_access_key, aws_session_

# Cliente Lambda
lambda_client = boto3.client('lambda', aws_access_key_id=aws_access_key_id, aws_secret_access_key=aws_secret_access_ke
```

Ejercicio 3.1 AWS Lambda `extractNobelPrizes`

La función debe recibir en el evento la categoría y el año a extraer y hacer uso del endpoint `http://api.nobelprize.org/2.1/nobelPrize/{category}/{year}` para extraer la información. Una vez obtenido, vamos a construir un DataFrame con pandas de una sola fila (manualmente a partir de una estructura de python). Este dataframe tendrá las siguientes columnas:

- **awardYear:** el campo `awardYear`. Si no existe dejar un NA
- **category:** el campo `en` dentro de `category`. Si no existe dejar un NA
- **categoryFullName:** el campo `en` dentro de `categoryFullName`. Si no existe dejar un NA

- **dateAwarded:** el campo `dateAwarded` . Si no existe dejar un NA
- **prizeAmount:** el campo `prizeAmount` . Si no existe dejar un NA
- **prizeAmountAdjusted:** el campo `prizeAmountAdjusted` . Si no existe dejar un NA
- **laureates_id:** una lista de laureate id's, correspondiente con el campo `id` de cada elemento de la lista de `laureates`
- **laureates_portion:** una lista de strings correspondientes con el valor interno `portion` de cada elemento de la lista de `laureates`

Posteriormente vamos a almacenar el DataFrame en el bucket del datalake en formato parquet (con el nombre `raw/nobelPrizes/{category}-{year}.parquet`).

Además la función debe **devolver una lista que contenga los IDs de los laureados** implicados en los premios Nobel obtenidos (lista de id's de la columna `laureates`).

Puedes ver si la lambda funciona usando uno de los siguientes códigos como categoría `che` , `eco` , `lit` , `pea` , `phy` , `med` y un año a tu elección (por ejemplo 2010).

```
In [3]: # Completar: introducir el código de la Lambda aquí
import io
import boto3
import fastparquet
import os
import json
import requests
import pandas as pd

def lambda_handler(event, context):
    category = event['category']
    year = event['year']

    endpoint = f"http://api.nobelprize.org/2.1/nobelPrize/{category}/{year}"
    response = requests.get(endpoint)
    data = response.json()

    prize = data[0] if isinstance(data, list) and len(data) > 0 else {}

    laureate_ids_list = [laureate['id'] for laureate in prize.get('laureates', [])]
    laureate_portions_list = [str(laureate['portion']) for laureate in prize.get('laureates', [])]

    # Crear el diccionario de columnas
    df_data = {
        'awardYear': prize.get('awardYear', 'NA'),
        'category': prize['category']['en'] if 'category' in prize else 'NA',
        'categoryFullName': prize['categoryFullName']['en'] if 'categoryFullName' in prize else 'NA',
        'dateAwarded': prize.get('dateAwarded', 'NA'),
        'prizeAmount': prize.get('prizeAmount', 'NA'),
        'prizeAmountAdjusted': prize.get('prizeAmountAdjusted', 'NA'),
        'laureates_id': laureate_ids_list,
        'laureates_portion': laureate_portions_list
    }

    # Crear el dataframe
    df_extractNobelPrizes = pd.DataFrame([df_data])

    # Para evitar el problema ValueError: I/O operation on closed file. se guarda el fichero temporalmente en disco
    temp_path = "tmp/filetemp.parquet"
    df_extractNobelPrizes.to_parquet(temp_path, engine='fastparquet')
    # Leemos el archivo temporal creado anteriormente
    with open(temp_path, 'rb') as temp_file:
        buffer = io.BytesIO(temp_file.read())

    # Almacenamos en S3
    s3 = boto3.client('s3', aws_access_key_id=aws_access_key_id, aws_secret_access_key=aws_secret_access_key, aws_session_token=aws_session_token)
    #s3 = boto3.client('s3', region_name='us-east-1') # En AWS usamos esto porque no hacen falta las credenciales
    bucket_name = 'capstone13mbc'
    s3_path = f"raw/nobelPrizes/{category}-{year}.parquet"
    s3.upload_fileobj(buffer, bucket_name, s3_path)

    #Eliminamos el fichero temporal
    os.remove(temp_path)
    display(df_extractNobelPrizes)

    return {
        'statusCode': 200,
```

```
'body': json.dumps({'message': f'Datos de {category}-{year} almacenados en S3', 'laureate_ids': laureate_ids_
```

```
In [4]: #Lambda test
event = {
    'category': 'eco',
    'year': 2010
}
result = lambda_handler(event, None)
print(result)
```

	awardYear	category	categoryFullName	dateAwarded	prizeAmount	prizeAmountAdjusted	laureates_id	laureates_portion
0	2010	Economic Sciences	The Sveriges Riksbank Prize in Economic Scienc...	2010-10-11	10000000	11015580	[856, 857, 858]	[1/3, 1/3, 1/3]

```
{'statusCode': 200, 'body': '{"message": "Datos de eco-2010 almacenados en S3", "laureate_ids": ["856", "857", "858"]}'}
```

Ejercicio 3.2 AWS Lambda `extractLaureate`

La función debe recibir en el evento el ID del laureado a extraer y hacer uso del endpoint

`http://api.nobelprize.org/2.1/laureate/{laureateID}` para extraer la información. Una vez obtenido, vamos a construir un DataFrame con pandas de una sola fila (manualmente a partir de una estructura de python). Este dataframe tendrá las siguientes columnas:

- **id**: el campo `id` . Si no existe dejar un NA
- **fullName**: el campo `en` dentro de `fullName` . Si no existe dejar un NA
- **fileName**: el campo `fileName` . Si no existe dejar un NA
- **gender**: el campo `gender` . Si no existe dejar un NA
- **birth**: el campo `date` dentro de `birth` . Si no existe dejar un NA

Posteriormente vamos a almacenar el DataFrame en el bucket del datalake en formato parquet (con el nombre `raw/laureates/{laureateID}.parquet`).

Puedes ver si la lambda funciona usando uno de los id's de la llamada anterior (por ejemplo 851).

Importante: selecciona la arquitectura `x86_64` y la versión de python 3.8. Finalmente, debes seleccionar el **rol de lam** existente llamado `LabRole` . También configuraremos el timeout de la lambda a 1 minuto y 2048 MB de RAM para asegurarnos que le da tiempo a ejecutarse correctamente.

```
In [5]: # Completar: introducir el código de La Lambda aquí
import io
import boto3
import fastparquet
import os
import json
import requests
import pandas as pd

def lambda_handler(event, context):
    laureate_id = event['laureate_id']

    endpoint = f"http://api.nobelprize.org/2.1/laureate/{laureate_id}"
    response = requests.get(endpoint)
    data_list = response.json()

    # Primer elemento de La Lista
    data = data_list[0] if data_list else {}

    # Diccionario de columnas del dataframe
    df_data = {
        'id': data.get('id', 'NA'),
        'fullName': data['fullName']['en'] if 'fullName' in data else 'NA',
        'fileName': data.get('fileName', 'NA'),
        'gender': data.get('gender', 'NA'),
        'birth': data['birth'].get('date', 'NA') if 'birth' in data else 'NA'
    }

    # Crear el dataframe
    df_laureates = pd.DataFrame([df_data])

    # Para evitar el problema ValueError: I/O operation on closed file. se guarda el fichero temporalmente en disco
```

```
temp_path = "tmp/filetemp.parquet"
df_laureates.to_parquet(temp_path, engine='fastparquet')
# Leemos el archivo temporal creado anteriormente
with open(temp_path, 'rb') as temp_file:
    buffer = io.BytesIO(temp_file.read())

# Almacenamos en S3
s3 = boto3.client('s3', aws_access_key_id=aws_access_key_id, aws_secret_access_key=aws_secret_access_key, aws_session_token=aws_session_token)
#s3 = boto3.client('s3', region_name='us-east-1') # En AWS usamos esto porque no hacen falta las credenciales
bucket_name = 'capstone13mbc'
s3_path = f"raw/laureates/{laureate_id}.parquet"

s3.upload_fileobj(buffer, bucket_name, s3_path)
os.remove(temp_path)

display(df_laureates)
return {
    'statusCode': 200,
    'body': json.dumps({'message': f'Datos del laureado {laureate_id} almacenados en S3'})
}
```

```
In [6]: #Lambda test
event = {
    'laureate_id': '856'
}

result = lambda_handler(event, None)
print(result)
event = {
    'laureate_id': '857'
}
result = lambda_handler(event, None)
print(result)
event = {
    'laureate_id': '858'
}
result = lambda_handler(event, None)
print(result)
```

id	fullName	fileName	gender	birth
0 856	Peter A. Diamond	diamond	male	1940-04-29

```
{'statusCode': 200, 'body': '{"message": "Datos del laureado 856 almacenados en S3"}'}
```

id	fullName	fileName	gender	birth
0 857	Dale T. Mortensen	mortensen	male	1939-02-02

```
{'statusCode': 200, 'body': '{"message": "Datos del laureado 857 almacenados en S3"}'}
```

id	fullName	fileName	gender	birth
0 858	Christopher A. Pissarides	pissarides	male	1948-02-20

```
{'statusCode': 200, 'body': '{"message": "Datos del laureado 858 almacenados en S3"}'}
```

Combinación de premios Nobel y laureados

Como los datos en `raw/nobelPrizes` y `raw/laureates` son ficheros parquet, podemos leer todos de una y crear un único `pd.DataFrame` usando `pd.read_parquet("s3://{bucket-datalake}/raw/nobelPrizes")` y `pd.read_parquet("s3://{bucket-datalake}/raw/laureates")` respectivamente.

A continuación vamos a implementar dos funciones AWS Lambda para generar una tabla `bronze` y una tabla `silver`.

NOTA: Para desarrollar localmente el código de estas dos ETL podéis usar los ficheros `nobelPrizes-1901-2019.parquet` y `laureates-1901-2019.parquet` disponibles en campus virtual, que contienen los datos exportados desde 1901 hasta 2019 (usando los endpoints de la API que dan todos los datos existentes paginados). Pero recordad que la lambda debe leer finalmente directamente del bucket de S3. En adelante nos referiremos a los datos de nobel prizes como `df_nobel_prizes` y a los datos de laureates como `df_laureates`.

Ejercicio 4. ETL bronze

En esta ETL vamos a analizar los datos en crudo que tenemos delante, limpiarlos, reestructurarlos y combinarlos.

Procesar el dataframe `df_nobel_prizes`

1. Vamos a explorar el contenido del dataframe y la completitud de los datos. ¿Hay nobels sin asignar? (es decir, `laureates` con valores perdidos). **Descartaremos esas filas en las que hay nobels sin asignar.**
2. Un mismo premio puede ser compartido por varios laureados, esto se indica a través de las columnas `laureates_id` y `laureates_portion`, que son listas. Utilizar `explode` para transformar el dataframe y tener un solo laureado por fila.
3. Vamos a castear la columna `laureates_id` para que sea un valor entero

Procesar el dataframe `df_laureates`

1. Hemos detectado que alguna de las columnas de interés no tiene un nombrado consistente. Renombrar las columnas:
 - `fileName` por `file_name`
2. Vamos a castear la columna `id` para que sea un valor entero

Combinar ambos dataframes

1. Combinar ambos dataframes por medio de la columna del id del laureado. Usar un inner join.

Implementar la AWS Lambda que ejecute la ETL bronze

1. La lambda debe y genere en el datalake un fichero parquet resultante de escribir el dataframe de pandas (path `bronze/bronze_laureates.parquet`).
2. Al igual que antes utiliza la layer. Selecciona la arquitectura `x86_64` y la versión de python 3.8. Finalmente, debes seleccionar el **rol de lam** existente llamado `LabRole`. También configuraremos el timeout de la lambda a 5 minutos y 2048 MB de RAM para asegurarnos que le da tiempo a ejecutarse correctamente.

Nota: el resultado de aplicar la ETL debería ser similar al que mostramos a continuación.

	awardYear	category	categoryFullName	dateAwarded	prizeAmount	prizeAmountAdjusted	laureates_id	laureates_portion	fullName
0	2010	Chemistry	The Nobel Prize in Chemistry	2010-10-06	10000000	11015580	851	1/3	Richard F. Heck
1	2010	Chemistry	The Nobel Prize in Chemistry	2010-10-06	10000000	11015580	851	1/3	Richard F. Heck

```
In [7]: # Completar: introducir el código de la Lambda aquí
import json
import requests
import pandas as pd
import io
import boto3
import fastparquet
import os
import s3fs

def lambda_handler(event, context):

    # Desde la Lambda no hace falta hacer mas que poner los path
    #df_nobel_prizes = pd.read_parquet("s3://capstone13mbc/raw/nobelPrizes")
    #df_laureates = pd.read_parquet("s3://capstone13mbc/raw/Laureates")

    # Para Leer el contenido de S3 desde Local
    ##### SOLO DESDE LOCAL INICIO #####
    # Especifica la ruta del archivo Parquet en S3
    #Cargo nobelprizes
    bucket_name = 'capstone13mbc'
    folder_key = 'raw/nobelPrizes'
    # Lista todos los objetos (archivos) en la carpeta en el bucket
    s3 = boto3.client('s3', aws_access_key_id=aws_access_key_id, aws_secret_access_key=aws_secret_access_key, aws_session_token=aws_session_token)

    # Lista todos los objetos (archivos) en la carpeta en el bucket
    objects = s3.list_objects_v2(Bucket=bucket_name, Prefix=folder_key)

    df_list = []

    for obj in objects.get('Contents', []):
        file_key = obj['Key']
        if file_key.endswith('.parquet'):
            response = s3.get_object(Bucket=bucket_name, Key=file_key)
```

```

        parquet_data = response['Body'].read()
        df = pd.read_parquet(io.BytesIO(parquet_data))
        df_list.append(df)

# Combina todos los DataFrames en uno solo
combined_df = pd.concat(df_list, ignore_index=True)

df_nobel_prizes = combined_df
display(df_nobel_prizes)

#Cargo Laureados
bucket_name = 'capstone13mbc'
folder_key = 'raw/laureates'
# Lista todos los objetos (archivos) en la carpeta en el bucket
objects = s3.list_objects_v2(Bucket=bucket_name, Prefix=folder_key)

df_list = []

for obj in objects.get('Contents', []):
    file_key = obj['Key']
    if file_key.endswith('.parquet'):
        response = s3.get_object(Bucket=bucket_name, Key=file_key)
        parquet_data = response['Body'].read()
        df = pd.read_parquet(io.BytesIO(parquet_data))
        df_list.append(df)

# Combina todos los DataFrames en uno solo
combined_df = pd.concat(df_list, ignore_index=True)

df_laureates = combined_df
display(df_laureates)

##### SOLO DESDE LOCAL FIN #####

"""
#Para hacerlo en local directamente
df_nobel_prizes = pd.read_parquet("raw_local/nobelPrizes")
df_laureates = pd.read_parquet("raw_local/laureates")
"""

df_nobel_prizes = df_nobel_prizes[df_nobel_prizes['laureates_id'].apply(lambda x: x is not None and len(x) > 0)]

#####

# Convertir las columnas de bytes a strings
df_nobel_prizes['laureates_id'] = df_nobel_prizes['laureates_id'].apply(lambda x: x.decode('utf-8'))
df_nobel_prizes['laureates_portion'] = df_nobel_prizes['laureates_portion'].apply(lambda x: x.decode('utf-8'))

# Convertir las columnas a listas
df_nobel_prizes['laureates_id'] = df_nobel_prizes['laureates_id'].apply(eval)
df_nobel_prizes['laureates_portion'] = df_nobel_prizes['laureates_portion'].apply(eval)

# Usar el método explode para duplicar las filas según las listas
df_nobel_prizes = df_nobel_prizes.explode(['laureates_id', 'laureates_portion'])

# Transformar el dataframe nobel_prizes usando explode
df_nobel_prizes_exp = df_nobel_prizes.explode('laureates_id').reset_index(drop=True)

# Convertir la columna 'laureates_id' a entero
df_nobel_prizes_exp['laureates_id'] = df_nobel_prizes_exp['laureates_id'].apply(lambda x: int(x))

# Convertir la columna 'id' a entero
df_laureates['id'] = df_laureates['id'].astype(int)

# Combinar ambos dataframes usando un inner join
df_bronze = pd.merge(df_nobel_prizes_exp, df_laureates, left_on='laureates_id', right_on='id', how='inner')

# Seleccionar columnas relevantes
columns = ['awardYear', 'category', 'categoryFullName', 'dateAwarded', 'prizeAmount', 'prizeAmountAdjusted',
           'laureates_id', 'laureates_portion', 'fullName', 'fileName', 'gender', 'birth']
df_bronze = df_bronze[columns]
display(df_bronze)

# Para evitar el problema ValueError: I/O operation on closed file. se guarda el fichero temporalmente en disco
temp_path = "tmp/filetemp.parquet"

df_bronze.to_parquet(temp_path, engine='fastparquet')
# Leemos el archivo temporal creado anteriormente
with open(temp_path, 'rb') as temp_file:
    buffer = io.BytesIO(temp_file.read())

```

```
# Almacenamos en S3
```

```
#s3 = boto3.client('s3', region_name='us-east-1') # En AWS usamos esto porque no hacen falta las credenciales
bucket_name = 'capstone13mbc'
s3_path = f"bronze/bronze_laureates.parquet"

s3.upload_fileobj(buffer, bucket_name, s3_path)
os.remove(temp_path)

return {
    'statusCode': 200,
    'body': 'ETL bronce cargado a S3'
}
```

```
In [8]: #Lambda test
event = {}
result = lambda_handler(event, None)
print(result)
```

awardYear	category	categoryFullName	dateAwarded	prizeAmount	prizeAmountAdjusted	laureates_id	laureates_portion
0	2010	Economic Sciences	The Sveriges Riksbank Prize in Economic Scienc...	2010-10-11	10000000	11015580	b'["856","857","858"]' b'["1/3","1/3","1/3"]'

	id	fullName	fileName	gender	birth
0	856	Peter A. Diamond	diamond	male	1940-04-29
1	857	Dale T. Mortensen	mortensen	male	1939-02-02
2	858	Christopher A. Pissarides	pissarides	male	1948-02-20

awardYear	category	categoryFullName	dateAwarded	prizeAmount	prizeAmountAdjusted	laureates_id	laureates_portion	fullName	
0	2010	Economic Sciences	The Sveriges Riksbank Prize in Economic Scienc...	2010-10-11	10000000	11015580	856	1/3	Peter A. Diamond
1	2010	Economic Sciences	The Sveriges Riksbank Prize in Economic Scienc...	2010-10-11	10000000	11015580	857	1/3	Dale T. Mortensen
2	2010	Economic Sciences	The Sveriges Riksbank Prize in Economic Scienc...	2010-10-11	10000000	11015580	858	1/3	Christopher A. Pissarides

{'statusCode': 200, 'body': 'ETL bronce cargado a S3'}

Ejercicio 5. ETL silver

En esta ETL vamos a procesar el dataframe bronze `df_bronze_laureates` y a construir nuevas variables.

Procesar el dataframe `df_bronze_laureates`

1. Tras analizar el dataframe bronze hemos evaluado el nombre de las columnas y algunas de ellas no nos parece un nombrado consistente. Renombrar las columnas para que tengan un formato uniforme:

- `prizeAmount` por `prize_amount`
- `awardYear` por `year`
- `fileName` por `file_name`
- `laureates_id` por `laureate_id`
- `laureates_portion` por `laureate_portion`
- `categoryFullName` por `category_full_name`
- `dateAwarded` por `date_awarded`
- `prizeAmountAdjusted` por `prize_amount_adjusted`
- `fullName` por `full_name`

2. Una de las columnas que hemos creado anteriormente, `laureate_portion`, es un string que representa una fracción. Nos interesa disponer del valor en formato numérico para crear una nueva columna `prize_amount_real` que represente el premio real recibido por cada laureado (al repartir el premio se divide económicamente en base a `laureate_portion`). Vamos a crear dos columnas más:

- `laureate_portion` : Vamos a sobrescribir esta columna, que es una fracción en formato string, para que sea un número decimal. Usar el código `from fractions import Fraction`, y aplicar un casting a `float` del `Fraction(texto)` para convertirlo a valor numérico

```
from fractions import Fraction
float(Fraction("1/3"))
```

- `prize_amount_real` : Esta columna representa el valor real ganado por cada laureado. Se calcula como `prize_amount` multiplicado por `laureate_portion`

Implementar la AWS Lambda que ejecute la ETL silver

1. La lambda debe y genere en el datalake un fichero parquet resultante de escribir el dataframe de pandas (path `silver/silver_laureates.parquet`).
2. Al igual que antes utiliza la layer. Selecciona la arquitectura `x86_64` y la versión de python 3.8. Finalmente, debes seleccionar el **rol de lam** existente llamado `LabRole`. También configuraremos el timeout de la lambda a 5 minutos y 2048 MB de RAM para asegurarnos que le da tiempo a ejecutarse correctamente.

Nota: el resultado de aplicar la ETL debería ser similar al que mostramos a continuación.

	year	category	categoryFullName	dateAwarded	prize_amount	prizeAmountAdjusted	laureate_id	laureate_portion	fullName	file_name
0	2010	Chemistry	The Nobel Prize in Chemistry	2010-10-06	10000000	11015580	851	0.333333	Richard F. Heck	heck
1	2010	Chemistry	The Nobel Prize in Chemistry	2010-10-06	10000000	11015580	851	0.333333	Richard F. Heck	heck

```
In [9]: # Completar: introducir el código de la Lambda aquí
import json
import requests
import pandas as pd
import io
import boto3
import fastparquet
import os
from fractions import Fraction

def lambda_handler(event, context):
    # Leer en S3 desde AWS
    df_bronze_laureates = pd.read_parquet("s3://capstone13mbc/bronze/bronze_laureates.parquet")

    """
    #En local
    df_bronze_laureates = pd.read_parquet("bronze/bronze_laureates.parquet")
    """

    ##### CARGAR DESDE LOCAL #####
    bucket_name = 'capstone13mbc'
    file_key = 'bronze/bronze_laureates.parquet'

    s3 = boto3.client('s3', aws_access_key_id=aws_access_key_id, aws_secret_access_key=aws_secret_access_key, aws_session_token=aws_session_token)

    # Descargar el contenido del archivo Parquet como un objeto BytesIO
    response = s3.get_object(Bucket=bucket_name, Key=file_key)
    parquet_data = response['Body'].read()
    df_bronze_laureates = pd.read_parquet(io.BytesIO(parquet_data))

    ##### CARGAR DESDE LOCAL #####

    # Nuevos nombres de columna
    renaming_dict = {
        'prizeAmount': 'prize_amount',
        'awardYear': 'year',
        'fileName': 'file_name',
        'laureates_id': 'laureate_id',
        'laureates_portion': 'laureate_portion',
        'categoryFullName': 'category_full_name',
        'dateAwarded': 'date_awarded',
        'prizeAmountAdjusted': 'prize_amount_adjusted',
        'fullName': 'full_name'
    }

    # Renombramos columnas
```

```
df_bronze_laureates.rename(columns=renaming_dict, inplace=True)

#Nuevas columnas con información
# Convertimos Laureate_portion a valores numéricos
df_bronze_laureates['laureate_portion'] = df_bronze_laureates['laureate_portion'].apply(lambda x: float(Fraction(x)))

# Calcular Los premios de cada Laureado
df_bronze_laureates['prize_amount_real'] = df_bronze_laureates['prize_amount'] * df_bronze_laureates['laureate_portion']

df_silver=df_bronze_laureates
display(df_silver)
# Para evitar el problema ValueError: I/O operation on closed file. se guarda el fichero temporalmente en disco
temp_path = "tmp/filetemp.parquet"

df_silver.to_parquet(temp_path, engine='fastparquet')
# Leemos el archivo temporal creado anteriormente
with open(temp_path, 'rb') as temp_file:
    buffer = io.BytesIO(temp_file.read())
# Almacenamos en S3
#s3 = boto3.client('s3', region_name='us-east-1') # En AWS usamos esto porque no hacen falta las credenciales
bucket_name = 'capstone13mbc'
s3_path = f"silver/silver_laureates.parquet"

s3.upload_fileobj(buffer, bucket_name, s3_path)
os.remove(temp_path)

return {
    'statusCode': 200,
    'body': 'ETL silver cargada a S3'
}
```

```
In [10]: #Lambda test
event = {}
result = lambda_handler(event, None)
print(result)
```

	year	category	category_full_name	date_awarded	prize_amount	prize_amount_adjusted	laureate_id	laureate_portion	full_name
index									
0	2010	Economic Sciences	The Sveriges Riksbank Prize in Economic Scienc...	2010-10-11	10000000	11015580	856	0.333333	Peter A. Diamond
1	2010	Economic Sciences	The Sveriges Riksbank Prize in Economic Scienc...	2010-10-11	10000000	11015580	857	0.333333	Dale T. Mortensen
2	2010	Economic Sciences	The Sveriges Riksbank Prize in Economic Scienc...	2010-10-11	10000000	11015580	858	0.333333	Christopher A. Pissarides

```
{'statusCode': 200, 'body': 'ETL silver cargada a S3'}
```

Ejercicio 6. Orquestrar con prefect core

Ahora vamos a escribir un flujo con prefect para que:

1. Exporte los datos del año anterior invocando la lambda `extractNobelPrizes` con un parámetro indicando la categoría a exportar (`che` , `eco` , `lit` , `pea` , `phy` o `med`). Es decir, se realizarán 6 invocaciones, una por categoría. Para obtener en el año anterior obtendremos del contexto la fecha actual (`prefect.runtime.flow_run.scheduled_start_time.year`), que es un objeto `pendulum.datetime.DateTime` , accederemos al campo `year` y le restaremos 1.
2. Por cada invocación de la lambda `extractNobelPrizes` se realizarán `N` invocaciones de la lambda `extractLaureate` , usando la funcionalidad de `map` para que por cada id de laureado devuelto por la lambda `extractNobelPrizes` (lista de ids de laureados) se ejecute la lambda `extractLaureate` con el parámetro `laureateId` correspondiente.
3. Tras extraer todos los datos del año anterior, ahora deberemos ejecutar la ETL bronze (lambda `bronze_laureates`).
4. Finalmente, deberemos ejecutar la ETL bronze (lambda `bronze_laureates`).

Nota: para invocar las AWS Lambda usaremos boto3. Podemos ver la documentación en este [link](#).

Tip: recuerda que puedes ejecutar tareas de prefect accediendo al atributo `fn` y los flujos ejecutándolos como si fueran funciones. Si necesitas establecer un contexto específico, como es el caso de la tarea `extract_nobel_prizes` con respecto a la fecha actual, puedes usar variables de entorno tal y como se indica en la [documentación oficial](#).

```
import os
os.environ["PREFECT__RUNTIME__FLOW_RUN__SCHEDULED_START_TIME"] = "2020-01-01 00:00:00"
# si queremos deshacer el cambio, basta con borrar esta variable de entorno
# del os.environ["PREFECT__RUNTIME__FLOW_RUN__SCHEDULED_START_TIME"]
```

En este caso el elemento `prefect.runtime.flow_run.scheduled_start_time` será un string. Podemos usar el siguiente código en nuestra tarea de `extract_nobel_prizes` para que si estamos mockeando el valor de `scheduled_start_time` y es un string convertirlo a `DateTime`, y si no dejarlo como está:

```
if isinstance(prefect.runtime.flow_run.scheduled_start_time, str):
    year = pendulum.from_format(prefect.runtime.flow_run.scheduled_start_time, "YYYY-MM-DD
HH:mm:ss").year
else:
    year = prefect.runtime.flow_run.scheduled_start_time.year
```

```
In [11]: #!/pip install pendulum
#!/pip install prefect
```

```
In [12]: # ETL
from prefect import flow, task
from io import BytesIO
import os
import pendulum
import prefect
import requests
import numpy as np
import json
import boto3
from concurrent.futures import ThreadPoolExecutor

@task
def extract():
    # Verificamos el formato de DateTime
    scheduled_start_time = os.environ["PREFECT__RUNTIME__FLOW_RUN__SCHEDULED_START_TIME"]

    if isinstance(prefect.runtime.flow_run.scheduled_start_time, str):
        year = pendulum.from_format(prefect.runtime.flow_run.scheduled_start_time, "YYYY-MM-DD HH:mm:ss").year
    else:
        year = prefect.runtime.flow_run.scheduled_start_time.year
    # Obtenemos el año anterior
    last_year=year-1
    print(last_year)
    def get_laureate_ids(category):
        nobel_payload = {
            "category": category,
            "year": last_year # Ajusta el año según sea necesario
        }

        nobel_response = lambda_client.invoke(
            FunctionName='caps13_extractNobelPrizes',
            InvocationType='RequestResponse',
            Payload=json.dumps(nobel_payload)
        )

        nobel_data = json.loads(nobel_response['Payload']).read().decode('utf-8'))

        if nobel_data.get("statusCode") == 200:
            return json.loads(nobel_data['body'])['laureate_ids']
        else:
            return []

    def process_laureate(laureate_id):
        laureate_payload = {
            "laureate_id": laureate_id
        }

        laureate_response = lambda_client.invoke(
            FunctionName='caps13_extractLaureate',
            InvocationType='RequestResponse',
            Payload=json.dumps(laureate_payload)
        )

        laureate_data = json.loads(laureate_response['Payload']).read().decode('utf-8'))
        print(laureate_data)

    # Categorías
    categories = ["che", "eco", "lit", "pea", "phy", "med"]
```

```

# mapeo de IDs de Laureados
with ThreadPoolExecutor() as executor:
    all_laureate_ids = list(executor.map(get_laureate_ids, categories))

# Procesamiento de Laureados y Llamada a Lambda_extractLaureateo
for laureate_ids in all_laureate_ids:
    with ThreadPoolExecutor() as executor:
        executor.map(process_laureate, laureate_ids)

@task
def transform_bronze():
    print("Transformación bronze")
    # Transformación bronze
    function_name = 'caps13_bronze'
    payload = {
    }
    response = lambda_client.invoke(
        FunctionName=function_name,
        InvocationType='RequestResponse',
        Payload=json.dumps(payload)
    )
    print(response)

@task
def transform_silver():
    print("Transformación silver")
    # Transformación silver
    function_name = 'caps13_silver'
    payload = {
    }
    response = lambda_client.invoke(
        FunctionName=function_name,
        InvocationType='RequestResponse',
        Payload=json.dumps(payload)
    )
    print(response)

@task
def load():
    print("Carga de datos finales (ya se han cargado en la transformación silver)")

@flow
def etl():
    e = extract()
    t1 = transform_bronze()
    t2 = transform_silver()
    l = load()

```

```

In [23]: # Establecemos el valor de la variable de entorno
os.environ["PREFECT__RUNTIME__FLOW_RUN__SCHEDULED_START_TIME"] = "2023-01-01 00:00:00"
result = etl()

```

```

22:14:35.160 | INFO      | prefect.engine - Created flow run 'laughing-cockatoo' for flow 'etl'
22:14:35.353 | INFO      | prefect.engine - Created task run 'extract-0' for task 'extract'
22:14:35.355 | INFO      | prefect.engine - Executing 'extract-0' immediately...

2022
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1016 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1015 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 743 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1023 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1021 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1022 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1017 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1020 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1018 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1019 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1014 almacenados en S3"}'}{'statusCode': 200, 'body': '{"message": "Datos del laureado 1013 almacenados en S3"}'}

{'statusCode': 200, 'body': '{"message": "Datos del laureado 1012 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1011 almacenados en S3"}'}
22:14:44.047 | INFO      | prefect.engine - Task run 'extract-0' - Finished in state Completed()
22:14:44.088 | INFO      | prefect.engine - Created task run 'transform_bronze-0' for task 'transform_bronze'
22:14:44.090 | INFO      | prefect.engine - Executing 'transform_bronze-0' immediately...

```

```

Transformación bronze
{'ResponseMetadata': {'RequestId': 'd5b1b1d5-970a-4c5d-b0d2-de908949f493', 'HTTPStatusCode': 200, 'HTTPHeaders': {'date': 'Sun, 13 Aug 2023 20:14:45 GMT', 'content-type': 'application/json', 'content-length': '54', 'connection': 'keep-alive', 'x-amzn-requestid': 'd5b1b1d5-970a-4c5d-b0d2-de908949f493', 'x-amzn-remapped-content-length': '0', 'x-amz-executed-version': '$LATEST', 'x-amzn-trace-id': 'root=1-64d939b4-2778ca29019ce3de3833c93c;sampld=0;lineage=ebec0023:0'}, 'RetryAttempts': 0}, 'StatusCode': 200, 'ExecutedVersion': '$LATEST', 'Payload': <botocore.response.StreamingBody object at 0x000001AA9E9C1820>}
22:14:45.544 | INFO | Task run 'transform_bronze-0' - Finished in state Completed()
22:14:45.583 | INFO | Flow run 'laughing-cockatoo' - Created task run 'transform_silver-0' for task 'transform_bronze-0'
22:14:45.585 | INFO | Flow run 'laughing-cockatoo' - Executing 'transform_silver-0' immediately...

Transformación silver
{'ResponseMetadata': {'RequestId': 'f72e4cc4-12d8-444b-8819-9b5e9312a499', 'HTTPStatusCode': 200, 'HTTPHeaders': {'date': 'Sun, 13 Aug 2023 20:14:45 GMT', 'content-type': 'application/json', 'content-length': '54', 'connection': 'keep-alive', 'x-amzn-requestid': 'f72e4cc4-12d8-444b-8819-9b5e9312a499', 'x-amzn-remapped-content-length': '0', 'x-amz-executed-version': '$LATEST', 'x-amzn-trace-id': 'root=1-64d939b5-45ce89cc5e5b0e8621326f84;sampld=0;lineage=be802fea:0'}, 'RetryAttempts': 0}, 'StatusCode': 200, 'ExecutedVersion': '$LATEST', 'Payload': <botocore.response.StreamingBody object at 0x000001AA9E9C2520>}
22:14:46.129 | INFO | Task run 'transform_silver-0' - Finished in state Completed()
22:14:46.167 | INFO | Flow run 'laughing-cockatoo' - Created task run 'load-0' for task 'transform_silver-0'
22:14:46.168 | INFO | Flow run 'laughing-cockatoo' - Executing 'load-0' immediately...

Carga de datos finales (ya se han cargado en la transformación silver)
22:14:46.300 | INFO | Task run 'load-0' - Finished in state Completed()
22:14:46.347 | INFO | Flow run 'laughing-cockatoo' - Finished in state Completed('All states completed.')

```

```

In [24]: bucket_name = 'capstone13mbc'
file_key = 'silver/silver_laureates.parquet'
# Descargar el contenido del archivo Parquet como un objeto BytesIO
response = s3.get_object(Bucket=bucket_name, Key=file_key)
parquet_data = response['Body'].read()
df_silver = pd.read_parquet(io.BytesIO(parquet_data))
display(df_silver)

```

	year	category	category_full_name	date_awarded	prize_amount	prize_amount_adjusted	laureate_id	laureate_portion	full_name
index									
0	2022	Chemistry	The Nobel Prize in Chemistry	2022-10-05	10000000	10000000	1015	0.333333	Carolyn R. Bertozzi
1	2022	Chemistry	The Nobel Prize in Chemistry	2022-10-05	10000000	10000000	1016	0.333333	Morten Meldal
2	2022	Chemistry	The Nobel Prize in Chemistry	2022-10-05	10000000	10000000	743	0.333333	K. Barry Sharpless
3	2022	Economic Sciences	The Sveriges Riksbank Prize in Economic Scienc...	2011-10-10	10000000	10000000	1021	0.333333	Ben S. Bernanke
4	2022	Economic Sciences	The Sveriges Riksbank Prize in Economic Scienc...	2011-10-10	10000000	10000000	1022	0.333333	Douglas W. Diamond
5	2022	Economic Sciences	The Sveriges Riksbank Prize in Economic Scienc...	2011-10-10	10000000	10000000	1023	0.333333	Philip H. Dybvig
6	2022	Literature	The Nobel Prize in Literature	2022-10-06	10000000	10000000	1017	1.000000	Annie Ernaux
7	2022	Physiology or Medicine	The Nobel Prize in Physiology or Medicine	2022-10-03	10000000	10000000	1011	1.000000	Svante Pääbo
8	2022	Peace	The Nobel Peace Prize	2022-10-07	10000000	10000000	1018	0.333333	Ales Bialiatski
9	2022	Peace	The Nobel Peace Prize	2022-10-07	10000000	10000000	1019	0.333333	NA
10	2022	Peace	The Nobel Peace Prize	2022-10-07	10000000	10000000	1020	0.333333	NA
11	2022	Physics	The Nobel Prize in Physics	2022-10-04	10000000	10000000	1012	0.333333	Alain Aspect
12	2022	Physics	The Nobel Prize in Physics	2022-10-04	10000000	10000000	1013	0.333333	John F. Clauser
13	2022	Physics	The Nobel Prize in Physics	2022-10-04	10000000	10000000	1014	0.333333	Anton Zeilinger

Ejercicio 7. Backfill

La forma ideal de exportar todos los datos desde 1901 hasta la fecha no es ejecutar nuestro flujo para cada año simulando su ejecución normal año a año, ya que supondría muchas llamadas API y sería un proceso muy lento y costoso. Lo ideal es hacer uso de los endpoints de la API para obtener todos los datos en bloque y hacer ese proceso de captura de los datos en bloque una sola vez, y a partir de ahí ya se encargaría nuestro flujo de actualizar los datos año a año, así como de ejecutar las ETLs correspondientes.

Los ficheros `nobelPrizes-1901-2019.parquet` y `laureates-1901-2019.parquet` (en el fichero `data.zip`) disponibles en campus virtual contienen los datos exportados desde 1901 hasta 2019 usando los endpoints de la API que dan todos los datos existentes paginados. Vamos a:

1. Asegurarnos de que nuestro bucket del datalake está vacío. Borrar todos los ficheros generados por las ejecuciones anteriores.
2. Subir los ficheros `nobelPrizes-1901-2019.parquet` y `laureates-1901-2019.parquet` al bucket del datalake a los prefijos `raw/nobelPrizes` y `raw/laureates` respectivamente.
3. Ejecutar el flujo manualmente simulando que es el 1 de enero desde el año 2021 hasta el año actual usando la variable de entorno descrita anteriormente (`2021-01-01` , `2022-01-01` , ...).

```
In [26]: # completar
import os
import pendulum

current_year = pendulum.now().year

for year in range(2021, current_year):
    print(f"Ejecutando ETL para año: {year-1}")
```

```
os.environ["PREFECT_RUNTIME_FLOW_RUN_SCHEDULED_START_TIME"] = f"{year}-01-01 00:00:00"
result = etl()
```

Ejecutando ETL para año: 2020

```
22:21:49.339 | INFO      | prefect.engine - Created flow run 'yellow-wildebeest' for flow 'etl'
22:21:49.528 | INFO      | prefect.engine - Created task run 'extract-0' for task 'extract'
22:21:49.531 | INFO      | prefect.engine - Executing 'extract-0' immediately...

2020
{'statusCode': 200, 'body': '{"message": "Datos del laureado 992 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 991 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 996 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 995 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 993 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 994 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 988 almacenados en S3"}'}{'statusCode': 200, 'body': '{"message": "Datos del laureado 989 almacenados en S3"}'}

{'statusCode': 200, 'body': '{"message": "Datos del laureado 990 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 985 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 987 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 986 almacenados en S3"}'}
22:22:02.353 | INFO      | prefect.engine - Task run 'extract-0' - Finished in state Completed()
22:22:02.392 | INFO      | prefect.engine - Created task run 'transform_bronze-0' for task 'transform_bronze'
22:22:02.395 | INFO      | prefect.engine - Executing 'transform_bronze-0' immediately...

Transformación bronce
{'ResponseMetadata': {'RequestId': '49332bdc-6ce0-455b-b5b8-92bc4ad75276', 'HTTPStatusCode': 200, 'HTTPHeaders': {'date': 'Sun, 13 Aug 2023 20:22:02 GMT', 'content-type': 'application/json', 'content-length': '878', 'connection': 'keep-alive', 'x-amzn-requestid': '49332bdc-6ce0-455b-b5b8-92bc4ad75276', 'x-amz-function-error': 'Unhandled', 'x-amzn-remapped-content-length': '0', 'x-amz-executed-version': '$LATEST', 'x-amzn-trace-id': 'root=1-64d93b6a-3011063e6b73cb4267403813;sampling=0;lineage=ebec0023:0'}, 'RetryAttempts': 0}, 'StatusCode': 200, 'FunctionError': 'Unhandled', 'ExecutedVersion': '$LATEST', 'Payload': <botocore.response.StreamingBody object at 0x000001AA9EB4E610>}
22:22:02.761 | INFO      | prefect.engine - Task run 'transform_bronze-0' - Finished in state Completed()
22:22:02.800 | INFO      | prefect.engine - Created task run 'transform_silver-0' for task 'transform_silver'
22:22:02.802 | INFO      | prefect.engine - Executing 'transform_silver-0' immediately...

Transformación silver
{'ResponseMetadata': {'RequestId': '018cc779-f31e-4f9a-84b5-a5ea1ec631e7', 'HTTPStatusCode': 200, 'HTTPHeaders': {'date': 'Sun, 13 Aug 2023 20:22:03 GMT', 'content-type': 'application/json', 'content-length': '54', 'connection': 'keep-alive', 'x-amzn-requestid': '018cc779-f31e-4f9a-84b5-a5ea1ec631e7', 'x-amzn-remapped-content-length': '0', 'x-amz-executed-version': '$LATEST', 'x-amzn-trace-id': 'root=1-64d93b6a-2ab07e2e756d80d3292e062f;sampling=0;lineage=be802fea:0'}, 'RetryAttempts': 0}, 'StatusCode': 200, 'ExecutedVersion': '$LATEST', 'Payload': <botocore.response.StreamingBody object at 0x000001AA9EB435B0>}
22:22:03.620 | INFO      | prefect.engine - Task run 'transform_silver-0' - Finished in state Completed()
22:22:03.667 | INFO      | prefect.engine - Created task run 'load-0' for task 'load'
22:22:03.669 | INFO      | prefect.engine - Executing 'load-0' immediately...

Carga de datos finales (ya se han cargado en la transformación silver)
22:22:03.791 | INFO      | prefect.engine - Task run 'load-0' - Finished in state Completed()
22:22:03.840 | INFO      | prefect.engine - Flow run 'yellow-wildebeest' - Finished in state Completed('All states completed.')
```

Ejecutando ETL para año: 2021

```
22:22:04.073 | INFO      | prefect.engine - Created flow run 'imposing-chicken' for flow 'etl'
22:22:04.261 | INFO      | prefect.engine - Created task run 'extract-0' for task 'extract'
22:22:04.264 | INFO      | prefect.engine - Executing 'extract-0' immediately...

2021
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1002 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1003 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1007 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1009 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1008 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1004 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1005 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1006 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1000 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 999 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 1001 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 997 almacenados en S3"}'}
{'statusCode': 200, 'body': '{"message": "Datos del laureado 998 almacenados en S3"}'}
22:22:23.153 | INFO      | prefect.engine - Task run 'extract-0' - Finished in state Completed()
22:22:23.192 | INFO      | prefect.engine - Created task run 'transform_bronze-0' for task 'transform_bronze'
22:22:23.193 | INFO      | prefect.engine - Executing 'transform_bronze-0' immediately...
```


Transformación bronze

```
{'ResponseMetadata': {'RequestId': 'f0831935-b2a5-47de-998b-d98dca0ec528', 'HTTPStatusCode': 200, 'HTTPHeaders': {'date': 'Sun, 13 Aug 2023 20:22:23 GMT', 'content-type': 'application/json', 'content-length': '878', 'connection': 'keep-alive', 'x-amzn-requestid': 'f0831935-b2a5-47de-998b-d98dca0ec528', 'x-amz-function-error': 'Unhandled', 'x-amzn-remapped-content-length': '0', 'x-amz-executed-version': '$LATEST', 'x-amzn-trace-id': 'root=1-64d93b7f-17ba474e21d05cb1256b4686;sampling=0;lineage=ebec0023:0'}, 'RetryAttempts': 0}, 'StatusCode': 200, 'FunctionError': 'Unhandled', 'ExecutedVersion': '$LATEST', 'Payload': <botocore.response.StreamingBody object at 0x000001AA9FC97AF0>}
```

22:22:23.573 | INFO | Task run 'transform_bronze-0' - Finished in state Completed()

22:22:23.613 | INFO | Flow run 'imposing-chicken' - Created task run 'transform_silver-0' for task 'transfo

22:22:23.615 | INFO | Flow run 'imposing-chicken' - Executing 'transform_silver-0' immediately...

Transformación silver

```
{'ResponseMetadata': {'RequestId': '597f0405-56e8-498c-a355-3c72444f8a3d', 'HTTPStatusCode': 200, 'HTTPHeaders': {'date': 'Sun, 13 Aug 2023 20:22:23 GMT', 'content-type': 'application/json', 'content-length': '54', 'connection': 'keep-alive', 'x-amzn-requestid': '597f0405-56e8-498c-a355-3c72444f8a3d', 'x-amzn-remapped-content-length': '0', 'x-amz-executed-version': '$LATEST', 'x-amzn-trace-id': 'root=1-64d93b7f-39e00a4910443f2c02d64b8f;sampling=0;lineage=be802fea:0'}, 'RetryAttempts': 0}, 'StatusCode': 200, 'FunctionError': 'Unhandled', 'ExecutedVersion': '$LATEST', 'Payload': <botocore.response.StreamingBody object at 0x000001AA9FC97B80>}
```

22:22:24.418 | INFO | Task run 'transform_silver-0' - Finished in state Completed()

22:22:24.462 | INFO | Flow run 'imposing-chicken' - Created task run 'load-0' for task 'load'

22:22:24.464 | INFO | Flow run 'imposing-chicken' - Executing 'load-0' immediately...

Carga de datos finales (ya se han cargado en la transformación silver)

22:22:24.594 | INFO | Task run 'load-0' - Finished in state Completed()

22:22:24.612 | INFO | Flow run 'imposing-chicken' - Finished in state Completed('All states completed')

```
In [28]: #Comprobamos el dataframe resultante
bucket_name = 'capstone13mbc'
file_key = 'silver/silver_laureates.parquet'
# Descargar el contenido del archivo Parquet como un objeto BytesIO
response = s3.get_object(Bucket=bucket_name, Key=file_key)
parquet_data = response['Body'].read()
df_silver = pd.read_parquet(io.BytesIO(parquet_data))
print("Dataframe completo hasta el año 2022")
display(df_silver)
```

Dataframe completo hasta el año 2022

	year	category	category_full_name	date_awarded	prize_amount	prize_amount_adjusted	laureate_id	laureate_portion	full_name
index									
0	2020	Chemistry	The Nobel Prize in Chemistry	2020-10-07	10000000	10000000	991	0.500000	Emmanuelle Charpentier
1	2020	Chemistry	The Nobel Prize in Chemistry	2020-10-07	10000000	10000000	992	0.500000	Jennifer A. Doudna
2	2021	Chemistry	The Nobel Prize in Chemistry	2021-10-06	10000000	10000000	1002	0.500000	Benjamin List
3	2021	Chemistry	The Nobel Prize in Chemistry	2021-10-06	10000000	10000000	1003	0.500000	David W.C. MacMillan
4	2022	Chemistry	The Nobel Prize in Chemistry	2022-10-05	10000000	10000000	1015	0.333333	Carolyn R. Bertozzi
...
1003	2021	Physics	The Nobel Prize in Physics	2021-10-05	10000000	10000000	1000	0.250000	Klaus Hasselmann
1004	2021	Physics	The Nobel Prize in Physics	2021-10-05	10000000	10000000	1001	0.500000	Giorgio Parisi
1005	2022	Physics	The Nobel Prize in Physics	2022-10-04	10000000	10000000	1012	0.333333	Alain Aspect
1006	2022	Physics	The Nobel Prize in Physics	2022-10-04	10000000	10000000	1013	0.333333	John F. Clauser
1007	2022	Physics	The Nobel Prize in Physics	2022-10-04	10000000	10000000	1014	0.333333	Anton Zeilinger

1008 rows × 13 columns

8. Responder a preguntas de analítica

Analítica de datos

Tenemos un conjunto de datos muy valioso. Vamos a leerlo localmente con pandas usando `pd.read_parquet("s3://{bucket-datalake}/silver/silver_laureates.parquet")` y a vamos a responder a las siguientes preguntas para ganar insight acerca de los datos que estamos manejando. Responder a cada pregunta usando celdas de **código** y/o **markdown** según se considere necesario.

1. ¿Quién es el laureado que más ha cobrado? (en caso de empate dar el más antiguo)
2. Generar una gráfica de línea en la que se muestre la inversión total por año (suma de `prize_amount_real`). ¿Cuál ha sido la tendencia?
3. ¿Hay algún laureado que haya recibido más de dos premios? Muestra el top 5.
4. ¿Qué categoría es la que más premios ha recibido? ¿Y que más dinero ha entregado?

```
In [37]: # Completar
#1
# Agrupamos laureados y calculamos la suma total
grouped = df_silver.groupby("laureate_id")["prize_amount_real"].sum()

max_paid_laureate_id = grouped.idxmax()

# Laureados con suma mas alta
max_paid_laureates = df_silver[df_silver["laureate_id"] == max_paid_laureate_id]

# Seleccionar al laureado más antiguo entre los de suma más alta
oldest_laureate = max_paid_laureates.sort_values("birth").iloc[0]

total_amount = grouped[max_paid_laureate_id]
formatted_amount = "{:,.2f}".format(total_amount)# Le doy un formato mas amigables para el dinero

print("\033[1;34mEl laureado que más ha cobrado es", oldest_laureate["full_name"], "con una cantidad total de", format
El laureado que más ha cobrado es K. Barry Sharpless con una cantidad total de 16,666,666.67 coronas suecas
```

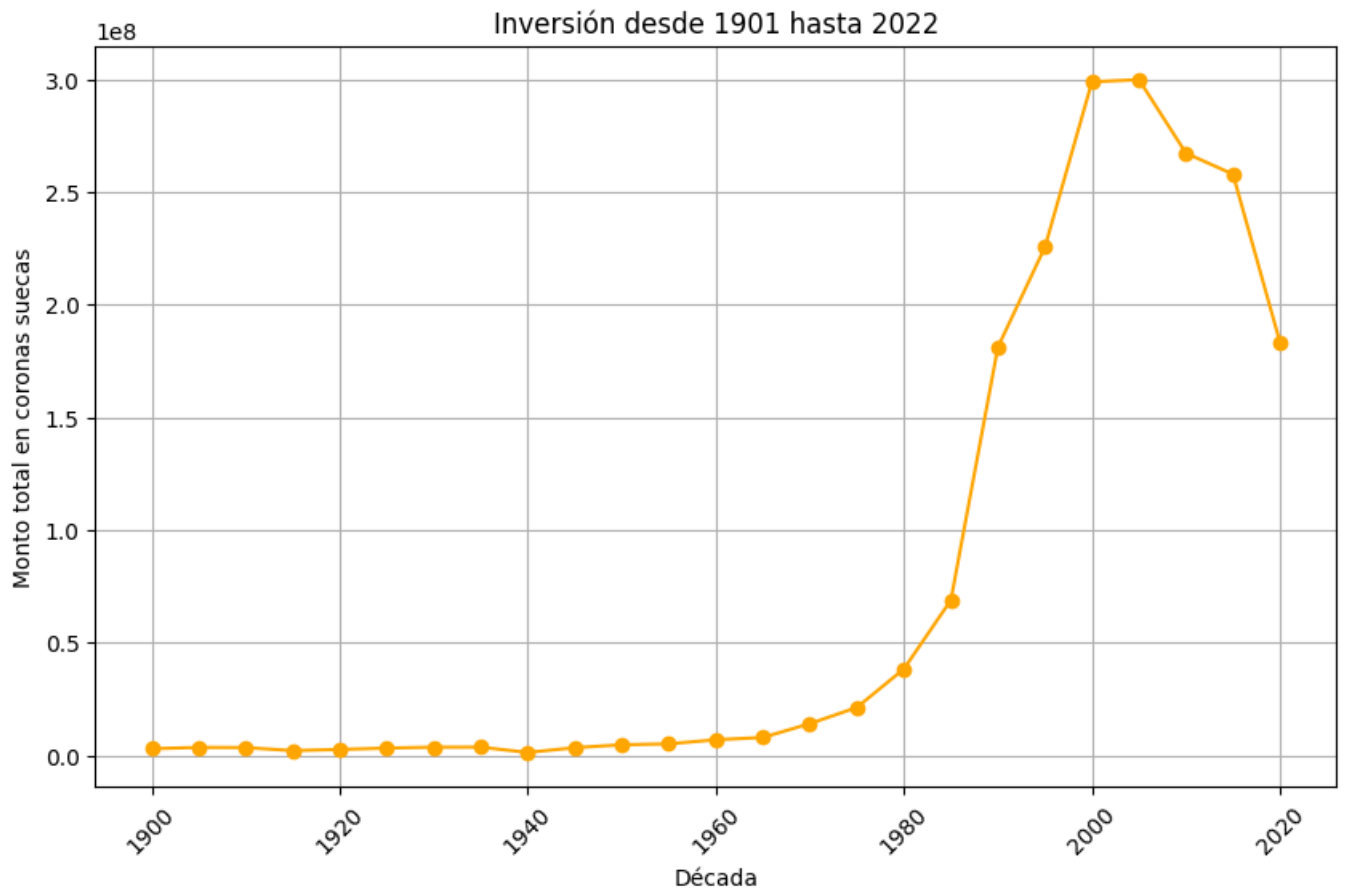
```
In [48]: #2
import matplotlib.pyplot as plt

# Convertir la columna 'year' a tipo numérico para cálculos
df_silver['year'] = pd.to_numeric(df_silver['year'])

# Agrupo por periodos de 5 años
grouped_by_decade = df_silver.groupby((df_silver['year'] // 5) * 5)['prize_amount_real'].sum()

# Crear la gráfica de puntos unidos por una línea
plt.figure(figsize=(10, 6))
plt.plot(grouped_by_decade.index, grouped_by_decade.values, marker='o', linestyle='-', color='orange')
plt.title("Inversión desde 1901 hasta 2022")
plt.xlabel("Década")
plt.ylabel("Monto total en coronas suecas")
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

print("\033[1;34mSe puede apreciar un claro aumento de la inversión a lo largo de los años, sobre todo desde los años
print("\033[1;34mPor otro lado, también vemos una caída desde el año 2005 hasta la actualidad, la cual llama la atenci
```



Se puede apreciar un claro aumento de la inversión a lo largo de los años, sobre todo desde los años 70. Por otro lado, también vemos una caída desde el año 2005 hasta la actualidad, la cual llama la atención en contraste con el crecimiento casi exponencial de décadas previas.

```
In [60]: #3
# Agrupamos y contamos la cantidad de premios para cada laureado, filtramos mayores de 2
laureate_counts = df_silver.groupby("laureate_id").size()
laureates_with_multiple_prizes = laureate_counts[laureate_counts > 2]

# Extraemos el top 5
top_5_laureates = laureates_with_multiple_prizes.nlargest(5)

# Mostrar Los resultados
print("Top 5 de laureados con más de dos premios:\n")
for laureate_id, count in top_5_laureates.items():
    laureate_data = df_silver[df_silver["laureate_id"] == laureate_id].iloc[0]
    print(laureate_data["full_name"], "(", laureate_data["file_name"], ")", "con ", count, " premios")

print("\n\033[1;34mLa Cruz Roja ha sido quien mas premios ha recibido, con un total de 9.")
```

Top 5 de laureados con más de dos premios:

```
None ( red-cross ) con 9 premios
Marie Curie, née Skłodowska ( marie-curie ) con 4 premios
John Bardeen ( bardeen ) con 4 premios
Linus Carl Pauling ( pauling ) con 4 premios
Frederick Sanger ( sanger ) con 4 premios
```

La Cruz Roja ha sido quien mas premios ha recibido, con un total de 9.

```
In [65]: #4

# Contamos Los premios por categoría y sumamos la cantidades
category_counts = df_silver['category'].value_counts()
category_total_amounts = df_silver.groupby('category')['prize_amount_real'].sum()

# Buscamos La categoría con mas premios
most_awarded_category = category_counts.idxmax()
most_money_category = category_total_amounts.idxmax()
most_money_amount = category_total_amounts[most_money_category]
formatted_amount = "{:,.2f}".format(most_money_amount) # Le doy un formato mas amigables para el dinero

# Mostrar Los resultados
print("Categoría con más premios es", most_awarded_category)
print("Categoría con más dinero entregado es ", most_money_category, " con un total de ", formatted_amount, " coronas ")
```

```
print("\n\033[1;34mLa categoría en la que se han otorgado más premios es medicina, aunque el campo que ha recibido la
```

Categoría con más premios es Physiology or Medicine

Categoría con más dinero entregado es Chemistry con un total de 330,463,899.67 coronas suecas.

La categoría en la que se han otorgado más premios es medicina, aunque el campo que ha recibido la mayor cantidad de dinero es química, con más de 330 millones de coronas suecas.