Preprocesado, exploración y visualización de datos en Google Cloud Platform

Informática como Servicio

Diego Sánchez Lamas

15 de febrero de 2019

Índice

1. Introducción

La ciencia de los datos, popularmente conocida por su nombre inglés, ***Data Science***, se entiende como una forma más moderna de denominar a la clásica **estadística**, pero casi siempre enfocada bajo un contexto **tecnológico**. Es por esto último por lo que es un campo en completo auge, y es que la constante evolución de los procesadores permite trabajar con mayores y mayores cantidades de datos en cuestión de segundos, pudiendo realizar incluso análisis en tiempo real.

Ahí es donde entra en juego **Google Cloud Platform (GCP)**, una solución IaaS de Google. GCP dispone de varios servicios realmente útiles para cada una de las diferentes posibles etapas de Data Science [1]:

1. **Recolección de datos**: Cloud Pub/Sub, Stackdriver Logging, BigQuery, Cloud Storage o App Engine.
2. **Extracción y transformación de datos**: Vision API, Speech API, Translation API o Natural Language API.
3. **Limpieza y preprocesado de datos**: Cloud Dataprep, Cloud Dataflow o Cloud Dataproc.
4. **Exploración de datos**: BigQuery o Cloud DataLab.
5. **Visualización de datos y colaboración**: Cloud DataLab o Data Studio.
6. **Entrenamiento de máquinas a partir de los datos**: Cloud Machine Learning (ML) Engine.

Gracias a los servidores de GCP, cualquier persona puede procesar grandes cantidades de datos en cuestión de segundos sin disponer de una infraestructura capaz de ello.

En este documento se va a presentar el **preprocesado** (con **Cloud Dataflow**), la **exploración** (con **BigQuery**) y la **visualización** (con **Data Studio**) de un pequeño conjunto de datos de aproximadamente **1800 películas**.

1.1. Caso de uso: Test de Bechdel

Como se ha comentado, vamos a utilizar un conjunto de datos de 1794 películas disponible en `https://github.com/fivethirtyeight/data/blob/master/bechdel/movies.csv`.

Para cada película, se indica, entre otros atributos, su título, si supera el **Test de Bechdel**, el año de publicación, el presupuesto y los ingresos en dólares. En este ejemplo no se requiere

una gran capacidad de procesado y se podría analizar prácticamente en cualquier equipo, pero se utiliza a modo de ejemplo para no incurrir en gastos innecesarios en Google Cloud Platform.

El Test de Bechdel es una medida para comprobar la representación del género femenino en películas, series, novelas, comics o cualquier otro tipo de obras a través de tres requerimientos mínimos:

1. Aparecen al menos dos personajes femeninos.
2. Estos dos personajes se hablan uno al otro al menos una vez.
3. La conversación trata de algo distinto a un hombre.

A pesar de que parecen tres pautas muy sencillas, muchas películas (tanto antiguas como actuales) no superan este test. Los Vengadores, Avatar, La Sirenita, El Señor de los Anillos, Star Wars o la reciente Bohemian Rhapsody son algunos ejemplos de películas populares que no cumplen las tres reglas.

1.2. Prerrequisitos

Para poder seguir el caso de uso en Google Cloud Platform es necesario cumplir una serie de prerrequisitos que se detallan a continuación:

1. Disponer de una **cuenta de GCP** (se puede crear en `cloud.google.com`).
2. Desde la misma, crear o disponer de un **proyecto**:
   a) Entrar a la consola de GCP (`console.cloud.google.com`) y acceder a `New Project`.
   b) Establecer un nombre y un id para el mismo. En este ejemplo hemos establecido "Test de Bechdel" y "`test-bechdel`" respectivamente (ver figura 1).
3. Instalar **Python 2.7** (con `pip` y, recomendablemente, `virtualenv`)
4. Instalar **Google Cloud SDK** en nuestro ordenador e inicializarlo.

Además, tener cierta familiaridad con **Python** y **SQL** puede resultar útil a la hora de entender el ejemplo en su totalidad.

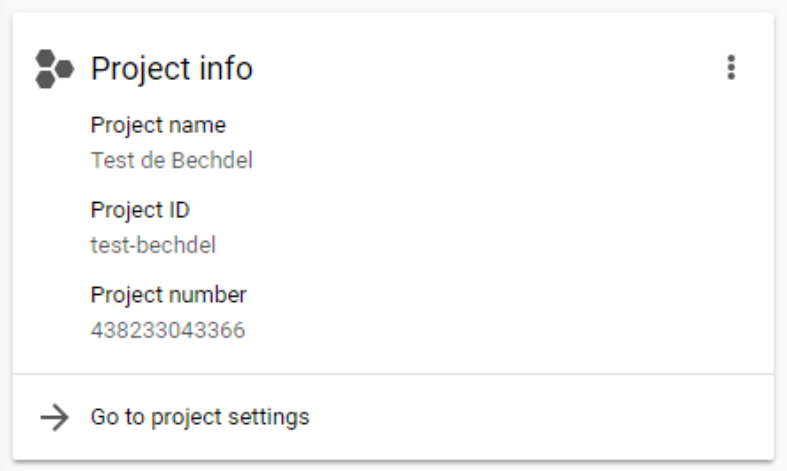

Figura 1: Proyecto creado para el caso de uso.

2. Preprocesado de datos

El **preprocesado de datos** en Data Science consiste en "arreglar" los datos que se han recopilado. Es un paso de vital importancia puesto que los datos pueden contener errores, omisiones o inconsistencias, con los que obtendríamos conclusiones muy poco fiables [3].

Las tareas más comunes dentro del preprocesado de datos incluyen la limpieza, la transformación y la reducción de los mismos [3]. En lugar de hacerlo a mano o ejecutando un proceso de limpieza separado, **Google Cloud Dataflow** nos ofrece la posibilidad de definir una serie de funciones que ejecutará a modo de ***pipeline***.

Una vez completado el preprocesado, se obtiene un conjunto de datos fiable a partir del cual se puede proseguir con el análisis de modo efectivo.

2.1. Cloud Dataflow

Google Cloud Dataflow es un servicio de Google Cloud Platform que nos permite definir funciones simples para preprocesar los datos en un pipeline. Este pipeline lo podemos agregar al pipeline de ingestión de datos para así conseguir un preprocesado automático [2].

2.2. Caso de uso: Test de Bechdel

En nuestro ejemplo, partimos de un conjunto de datos (*dataset*) muy bien formado, pero podríamos obtener uno más ajustado y limpio. Utilizaremos Cloud Dataflow con Python

como lenguaje de programación.

2.2.1. Crear pipeline

Para cada una de las tareas implementaremos una función que reciba un registro y lo devuelva (o no, si es descartado) tras ejecutar las operaciones pertinentes.

En primer lugar descartamos los registros que tienen "dudoso" como resultado. Si es dudoso interrumpimos la iteración, mientras que si no lo es devolvemos el registro para que continúe el preprocesado del mismo:

```
1 def discard_dubious(record):
2     """Discards dubious records."""
3     if record['test-clean'] == 'dubious':
4         raise StopIteration()
5     yield record
```

También queremos descartar aquellos que no dispongan de información presupuestaria, así que definimos otra función del mismo modo (también podríamos unificar los descartes en una misma función):

```
1 def filter_na(record):
2     """Filters records without budget info."""
3     if record['domgross'] == '#N/A':
4         raise StopIteration()
5     yield record
```

Por último, vamos a renombrar y descartar algunas de las columnas de las que disponemos y que son redundantes o resultan innecesarias para nuestro análisis:

```
1  def massage_rec(record):
2      """Massage columns."""
3      # Redundant columns
4      del record['test']
5      del record['code']
6      del record['budget']
7      del record['domgross']
8      del record['intgross']
9      del record['binary']
10     del record['period code']
11     del record['decade code']
12
```

```
    # Remove the 'clean' prefix
    record['test'] = record['clean-test']
    del record['clean-test']

    record['budget'] = record['budget_2013\$']
    del record['budget_2013\$']
    record['domgross'] = record['domgross_2013\$']
    del record['domgross_2013\$']
    record['intgross'] = record['intgross_2013\$']
    del record['intgross_2013\$']

    return record
```

Sería recomendable añadir alguna tarea más para comprobar y tratar de arreglar valores extraños en los presupuestos (beneficios desproporcionados) o en los años, pero para este caso vamos a dejarlo así. Podríamos definir, por ejemplo, una tarea que a los años anteriores al 1000 les añada un 1 al principio, asumiendo que ese sería el error y consecuentemente subsanándolo.

Tan sólo resta unirlo en un pipeline que podamos ejecutar en Cloud Dataflow, para ello añadimos las funciones para leer los datos del CSV y para escribir los resultados en BigQuery, desde donde los exploraremos en el siguiente paso.

Fichero completo [6]:

```
import apache_beam
from apache_beam.io import filebasedsource
from beam_utils.sources import CsvFileSource
import argparse
import re

BIGQUERY_TABLE_FORMAT = re.compile(r'[\w.:-]+:\w+\.\w+$')

def discard_dubious(record):
    """Discards dubious records."""
    if record['clean_test'] == 'dubious':
        raise StopIteration()
    yield record

def filter_na(record):
    """Filters records without budget info."""
    if record['domgross'] == '#N/A':
        raise StopIteration()
    yield record

def massage_rec(record):
```

```
22      """Massage columns."""
23      # Redundant columns
24      del record['test']
25      del record['code']
26      del record['budget']
27      del record['domgross']
28      del record['intgross']
29      del record['binary']
30      del record['period code']
31      del record['decade code']
32
33      # Remove the 'clean' prefix
34      record['test'] = record['clean_test']
35      del record['clean_test']
36
37      record['budget'] = record['budget_2013$']
38      if record['budget'] == '#N/A':
39          del record['budget']
40      del record['budget_2013$']
41      record['domgross'] = record['domgross_2013$']
42      if record['domgross'] == '#N/A':
43          del record['domgross']
44      del record['domgross_2013$']
45      record['intgross'] = record['intgross_2013$']
46      if record['intgross'] == '#N/A':
47          del record['intgross']
48      del record['intgross_2013$']
49
50      return record
51
52  def convert_types(record):
53      """Converts string values to their appropriate type."""
54
55      record['year'] =
56          int(record['year']) if 'year' in record
57          else None
58      record['budget'] =
59          int(record['budget']) if 'budget' in record
60          else None
61      record['domgross'] =
62          int(record['domgross']) if 'domgross' in record
63          else None
64      record['intgross'] =
65          int(record['intgross']) if 'intgross' in record
66          else None
67
68      return record
```

```
69
70  def main(src_path, dest_table, pipeline_args):
71      p = apache_beam.Pipeline(argv=pipeline_args)
72
73      value = p | 'Read CSV' >>
74          apache_beam.io.Read(CsvFileSource(src_path))
75
76      value |= (
77          'Remove dubious records' >>
78          apache_beam.FlatMap(discard_dubious))
79
80      value |= (
81          'Filter N/A data' >>
82          apache_beam.FlatMap(filter_na))
83
84      value |= (
85          'Massage columns' >>
86          apache_beam.Map(massage_rec))
87
88      value |= (
89          'Convert string values to their types' >>
90          apache_beam.Map(convert_types))
91
92      value |= (
93          'Dump data to BigQuery' >>
94          apache_beam.io.Write(apache_beam.io.BigQuerySink(
95              dest_table,
96              schema=', '.join([
97                  'year:INTEGER',
98                  'imdb:STRING',
99                  'title:STRING',
100                 'test:STRING',
101                 'budget:INTEGER',
102                 'domgross:INTEGER',
103                 'intgross:INTEGER',
104             ]),
105             create_disposition=(
106                 apache_beam.io
107                     .BigQueryDisposition.CREATE_IF_NEEDED),
108             write_disposition=(
109                 apache_beam.io
110                     .BigQueryDisposition.WRITE_TRUNCATE))))
111
112     p.run().wait_until_finish()
113
114 def bq_table(bigquery_table):
115     if not BIGQUERY_TABLE_FORMAT.match(bigquery_table):
```

```
116         raise argparse.ArgumentTypeError(
117             'bq_table must be of format
118                 PROJECT:DATASET.TABLE')
119     return bigquery_table
120
121
122 if __name__ == '__main__':
123     parser = argparse.ArgumentParser(
124         description=__doc__,
125         formatter_class=argparse.RawDescriptionHelpFormatter)
126     parser.add_argument('src_path', help=(
127         'The csv file with the data. This can be either a
128         local file, or a '
129         'Google Cloud Storage uri of the form
130         gs://bucket/object.'))
131     parser.add_argument(
132         'dest_table', type=bq_table, help=(
133             'A BigQuery table, of the form
134             project-id:dataset.table'))
135
136     args, pipeline_args = parser.parse_known_args()
137
138     main(args.src_path, args.dest_table,
139         pipeline_args=pipeline_args)
```

2.2.2. Ejecutar pipeline

Una gran ventaja de Apache Beam y Cloud Dataflow es que nos permite ejecutar el pipeline tanto de forma local como en Cloud Dataflow. Así, si el conjunto de datos es relativamente pequeño o queremos probar con una pequeña muestra, podremos hacerlo en nuestro ordenador, mientras que si el conjunto de datos es de mayor magnitud podremos aprovechar la gran capacidad de procesamiento de GCP.

De todas formas, lo primero que debemos hacer es **crear un *dataset* de BigQuery** en Google Cloud Platform, al que se derivarán los resultados del pipeline:

```
1 $ gcloud config set project test-bechdel
2 $ bq mk bechdel_dataset
```

Con el primer comando nos aseguramos de estar trabajando sobre el proyecto correcto, estableciendo el proyecto actual por su ID. El segundo comando nos sirve para crear el *dataset*, que en este caso le hemos llamado `bechdel_dataset`.

Para ejecutar el pipeline debemos instalar el paquete para Python de Google Cloud Dataflow. Es recomendable crear un entorno virtual de Python para ello (usando `virtualenv`), de modo que este paquete no interfiera en otros que se puedan tener instalados.

```
$ pip install google-cloud-dataflow
```

Como nuestros datos están en formato CSV, debemos instalar una dependencia más: Beam Utils, que nos permitirá tenerlos como entrada:

```
$ pip install beam_utils
```

Como se ha comentado anteriormente, disponemos de dos modos para ejecutar el pipeline que hemos creado, en local o en Cloud Dataflow, aprovechando la capacidad de procesamiento de GCP.

- **Ejecutar de forma local**:

```
$ python pipeline.py movies.csv
    test-bechdel:bechdel_dataset.cleansed
```

- **Ejecutar en Cloud Dataflow**:
  Para ejecutar en Cloud Dataflow esto mismo, el primer requisito es habilitar Dataflow API en nuestro proyecto, disponer de un *bucket* de Google Cloud Storage y subir al mismo el fichero CSV que queremos procesar.

```
gsutil mb gs://bechdel
gsutil cp movies.csv gs://bechdel
```

  Mediante estos dos comandos creamos un *bucket* de nombre `bechdel` y subimos al mismo el fichero `movies.csv`. A continuación, ejecutamos en la nube nuestro pipeline.

```
$ python pipeline.py gs://bechdel/movies.csv
    test-bechdel:bechdel_dataset.cleansed
    --project test-bechdel
    --job_name test-bechdel-movies
    --runner BlockingDataflowPipelineRunner
    --staging_location gs://bechdel/staging
    --temp_location gs://bechdel/temp
    --output gs://bechdel/output
```

Si la ejecución en GCP da un error de falta de autorización, puede ser necesario volver a loguearse con la cuenta de Google:

```
gcloud auth application-default login
```

Accediendo a la consola de Dataflow en Google Cloud Platform podemos observar la ejecución del pipeline paso por paso y toda la información de la propia ejecución:

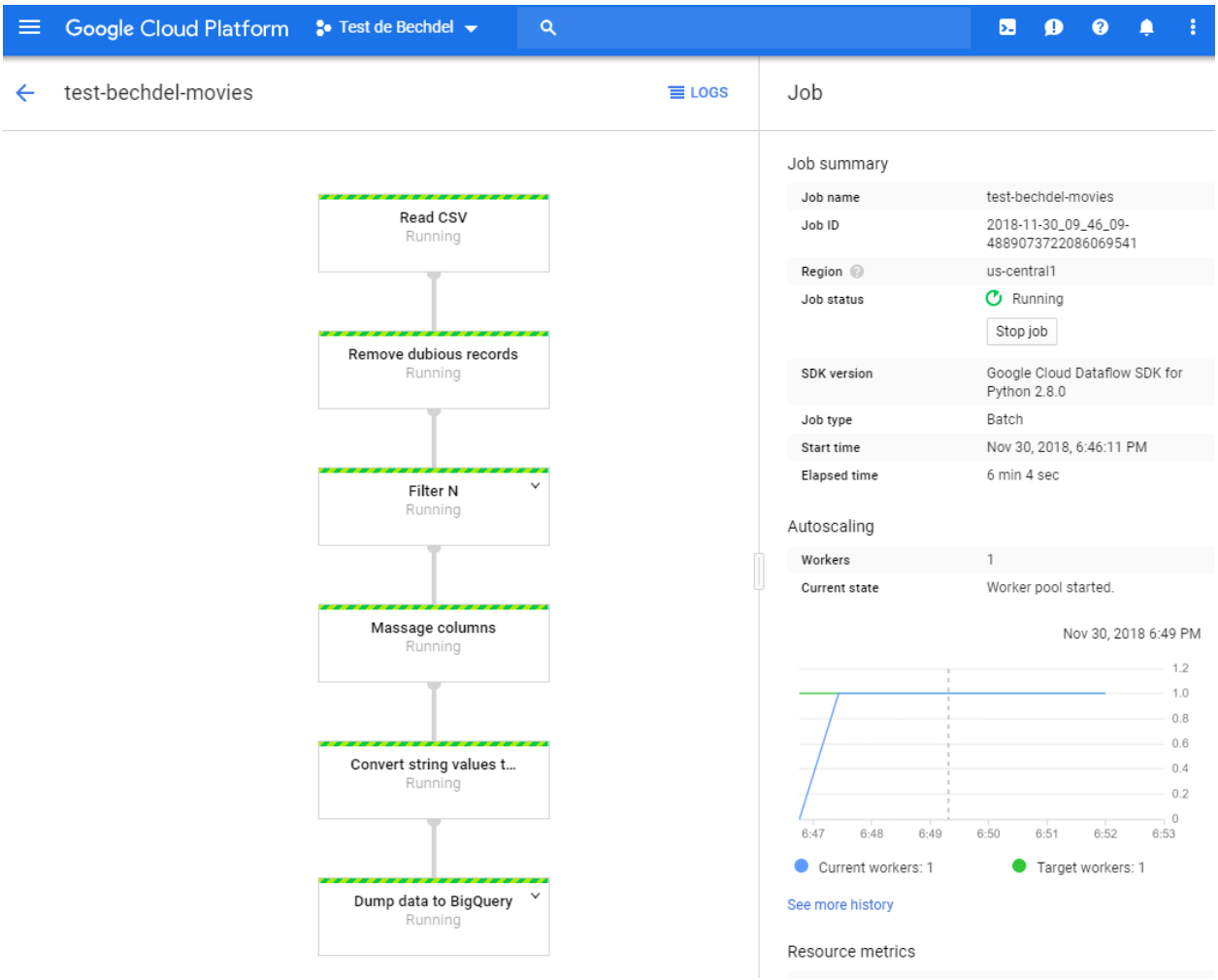

Figura 2: Información del *job* en Cloud Dataflow.

3. Exploración de datos

Una vez disponemos de los datos bien formados, el siguiente paso lógico en Data Science es **explorarlos**.

En este paso, además del conocimiento sobre el dominio sometido a estudio, entra en juego la creatividad y la intuición, para buscar posibles patrones y consultas interesantes que arrojen

información valiosa de cara a las respuestas que deseamos encontrar.

Por ejemplo, para una cadena de supermercados nos puede interesar saber cuál es el producto más vendido y si este tiene gran margen de ventas con el siguiente. Pero para no obtener resultados desvirtuados debemos tener en cuenta otras variables, como si algún producto se quedó fuera de stock. Además puede no resultar valioso si no se limita por regiones o zonas.

3.1. BigQuery

El servicio **BigQuery** de Google Cloud Platform se define en su página principal como *"un almacén de datos en la nube rápido, económico, de gran escalabilidad y totalmente administrado para realizar análisis con aprendizaje automático integrado"* [4].

Todas las características mencionadas nos resultan de gran utilidad para la exploración de datos, sobretodo en casos en los que disponemos de cantidades de datos del orden de terabytes o mayores.

Al ser totalmente administrado, podemos ejecutar nuestras consultas sin preocuparnos de la infraestructura subyacente y centrándonos así en el proceso de la ciencia de los datos que estemos considerando.

El gran rendimiento de BigQuery se puede comprobar con bases de datos públicas de las que dispone, como `bigquery-samples:wikipedia_benchmark`. En esta base de datos existe una tabla denominada `Wiki100B`, con 100 billones de filas y un tamaño de 7 TB.

En un artículo publicado en el blog de Google Cloud, *"Anatomy of a BigQuery Query"* [7], comprueban la velocidad de BigQuery ejecutando sobre dicha tabla la siguiente consulta:

```
1 SELECT language, SUM(views) as views
2     FROM [bigquery-samples:wikipedia_benchmark.Wiki100B]
3     WHERE REGEXP_MATCH(title, "G.*o.*o.*g")
4     GROUP BY language
5     ORDER BY views desc;
```

BigQuery completa la consulta en un tiempo de 24.7 segundos, procesando 4.06 TB de información.

De acuerdo al artículo, BigQuery, en el mejor de los casos, tuvo que realizar en ese lapso de tiempo todas estas tareas:

- Leer sobre 1 TB de información y descomprimirla a 4 TB (asumiendo una compresión 4 a 1).

- Ejecutar 100 billones de expresiones regulares con 3 *wildcards* cada una.
- Distribuir 1.25 TB de datos a través de la red (1 TB comprimido para la lectura inicial y 0.25 TB para la agregación).

Asumiendo un contexto muy optimista e ideal en el que las consultas son perfectamente paralelizables, para que un clúster distribuido hiciese la misma consulta en un tiempo de 30 segundos (recordemos que BigQuery lo hizo en 24.7) necesitaría, **como mínimo**, los siguientes recursos:

- Sobre 330 discos duros dedicados a 100 MB/seg para leer 1 TB de información.
- Una red de 330 Gigabits para distribuir los 1.25 TB de datos.
- 3330 núcleos para descomprimir 1 TB de información y procesar 100 billones de expresiones regulares a 1 microsegundo por expresión.

Nota: los recursos no coinciden exactamente con el artículo del blog de Google Cloud puesto que se han recalculado de modo ligeramente más realista.

3.2. Caso de uso: Test de Bechdel

Para nuestro caso de uso, una vez cargado nuestro dataset en BigQuery a través del pipeline que ejecutamos en el apartado anterior, vamos a probar el funcionamiento con algunas simples consultas.

Estas consultas pueden ejecutarse directamente en la interfaz web de BigQuery o pueden invocarse por línea de comandos mediante `bq query` [5]. Por comodidad vamos a utilizar la segunda opción, pero la interfaz web tiene algunas ventajas como información de la cantidad de datos que se van a procesar cuando se ejecute la *query* escrita, lo que puede resultar muy útil cuando se trabaja con grandes datasets y se quieren controlar los posibles gastos.

Vamos a empezar por ver cuántos registros tenemos:

```
1 $ bq query "select count(*) as TOTAL
2     from [test-bechdel:bechdel_dataset.cleansed]"
```

```
1 +-------+
2 | TOTAL |
3 +-------+
4 |  1636 |
5 +-------+
```

¿Cuántas de las películas registradas cumplen el test de Bechdel? ¿Cuántas fallan en según qué aspecto?

```
$ bq query "select test, count(*) as TOTAL
    from [test-bechdel:bechdel_dataset.cleansed]
    GROUP BY test"
```

```
+---------+-------+
|  test   | TOTAL |
+---------+-------+
| ok      |   794 |
| notalk  |   510 |
| men     |   193 |
| nowomen |   139 |
+---------+-------+
```

¿Qué ratio del total representan estas cifras?

```
$ bq query "select test, count(*) as TOTAL,
    RATIO_TO_REPORT(total) OVER () as RATIO
    from [test-bechdel:bechdel_dataset.cleansed]
    GROUP BY test"
```

```
+---------+-------+---------------------+
|  test   | TOTAL |        RATIO        |
+---------+-------+---------------------+
| ok      |   794 | 0.48533007334963324 |
| notalk  |   510 |  0.3117359413202934 |
| men     |   193 | 0.11797066014669927 |
| nowomen |   139 | 0.08496332518337409 |
+---------+-------+---------------------+
```

Como podemos ver, un 48.53 % de las películas pasa el test ante un 51.47 % que no lo consigue. Un 31.17 % falla porque los dos personajes femeninos no hablan entre sí, un 11.80 % falla porque la conversación que tienen estos dos personajes femeninos gira en torno a algún hombre y, finalmente, un 8.50 % falla porque no hay dos personajes femeninos.

Podríamos continuar nuestra exploración agrupando por años para analizar si el ratio ha cambiado con el paso del tiempo, pero dejaremos eso para el siguiente apartado, la visualización de datos.

4. Visualización de datos

Otro paso importante de Data Science es la **visualización de datos**. Tan importante como explorar la información es presentarla y cómo presentarla. Un modo u otro de presentarla puede hacer cambiar en gran medida el mensaje recibido por los lectores.

Las formas más comunes de visualización de análisis de datos incluyen gráficos de barras, tablas, gráficos de líneas, diagramas de cajas y bigotes, gráficos de velas, histogramas, etc.

4.1. Data Studio

Google Data Studio es una herramienta de Google que nos permite **crear visualizaciones** de datos de manera muy cómoda y personalizada. Se puede conectar con gran variedad de fuentes de datos e incluso puede ampliarlas definiendo nuevos campos calculados.

Otra gran ventaja de Google Data Studio es la posibilidad de **compartir** los *reports* con otros usuarios, de modo que se puede trabajar de manera conjunta.

Aunque al principio puede resultar un poco tedioso, rápidamente se descubren las bondades de esta herramienta y la facilidad que da a la hora de crear visualizaciones de datos profesionales [8].

4.2. Caso de uso: Test de Bechdel

Para nuestro caso de uso, desde la página principal de Google Data Studio, iniciamos un nuevo *report*.

El primer paso es seleccionar la fuente de los datos que se desean visualizar, en este caso creamos un nuevo *data source*, para el que seleccionamos y autorizamos el **conector de BigQuery**. Una vez autorizado, seleccionamos el proyecto, el dataset y la tabla en cuestión.

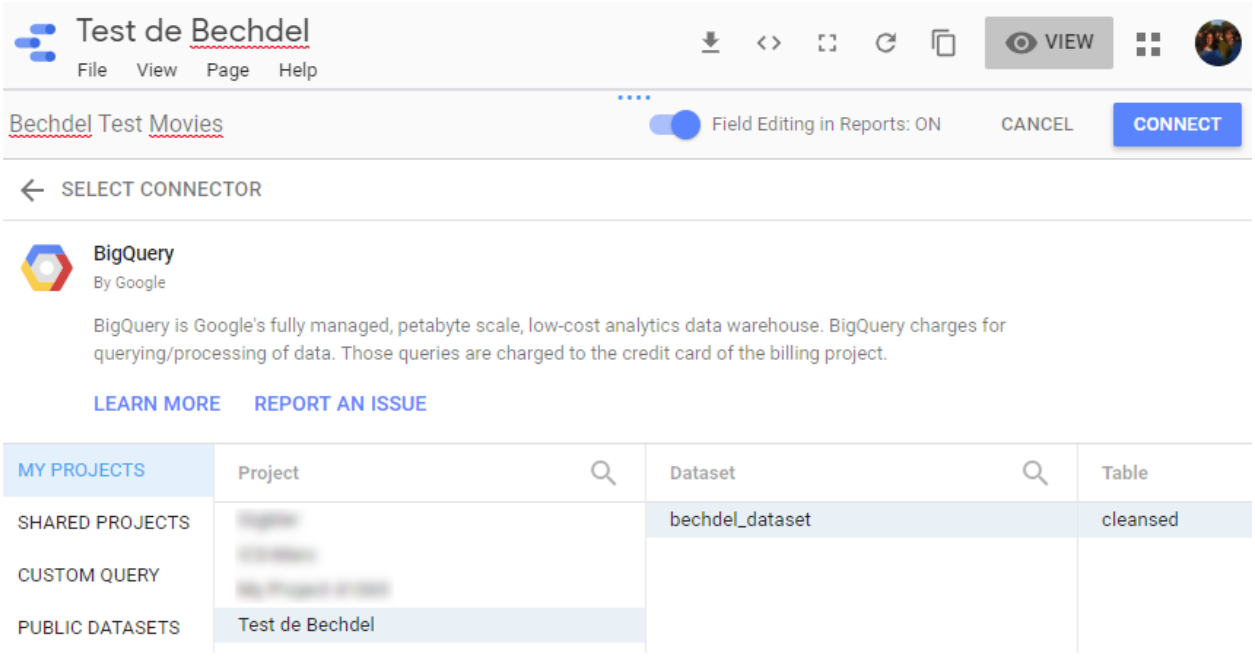

Figura 3: Fuente de datos para el *report* de Data Studio.

Para facilitar la visualización, podemos ajustar el tipo de dato de cada campo de la tabla o añadir columnas personalizadas. Por ejemplo, en este caso vamos a indicar que ciertos campos son de tipo monetario, concretamente dólares.

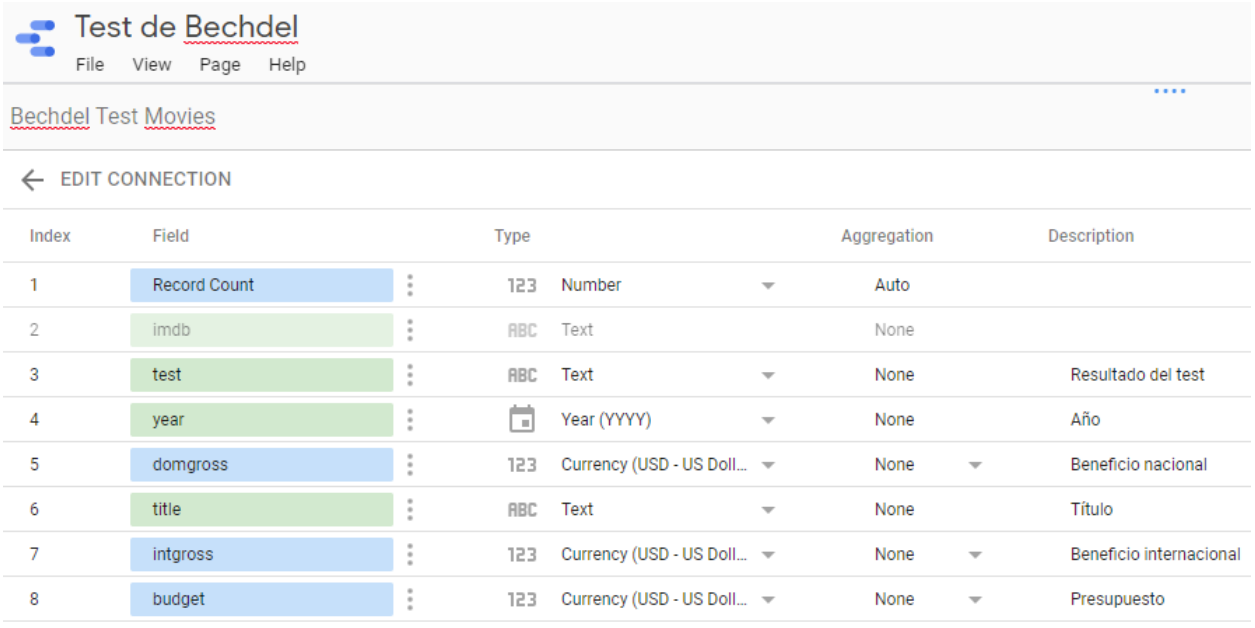

Figura 4: Mapeo de datos con el *datasource*.

En primer lugar vamos a hacer una de las gráficas más simples, un diagrama de barras que indica el número de registros (películas) que tenemos para cada año. Para ello tan solo tenemos que dibujar un rectángulo con la herramienta de diagrama de barras y marcar como dimensión el año (*year*) y como métrica el conteo de registros (*record count*).

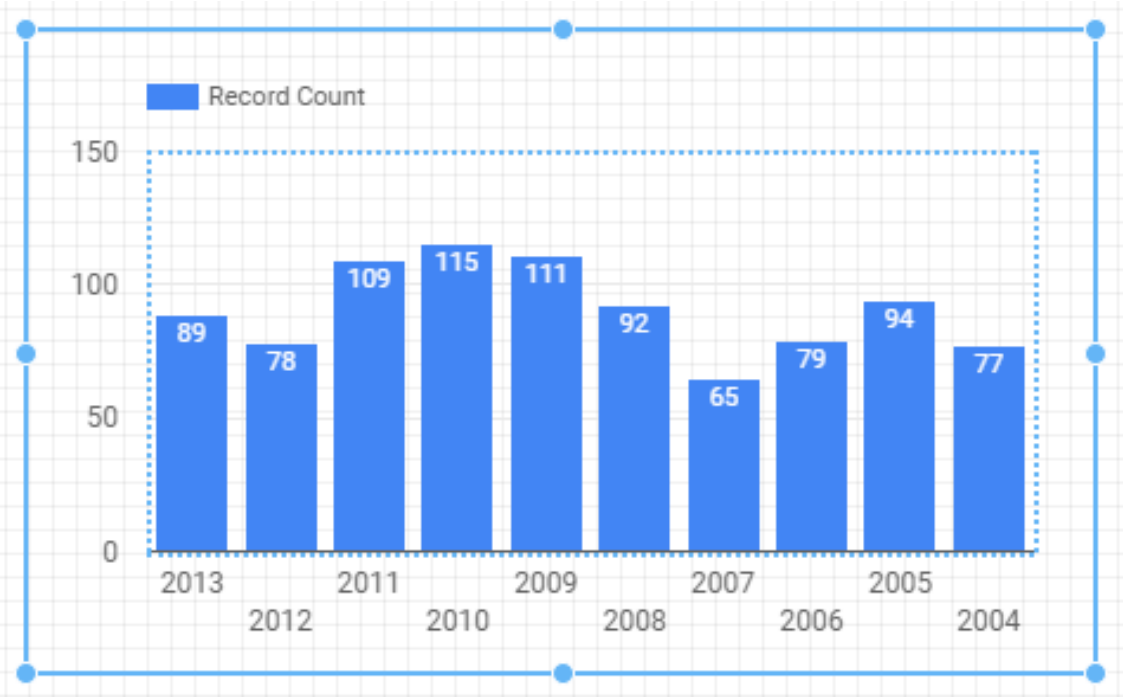

Figura 5: Número de películas por año, gráfica en Data Studio.

Este diagrama por sí solo no nos sirve de nada, vamos a añadirle la información del número de películas que pasan el Test de Bechdel.

Una posibilidad para añadir esta información es añadir a este diagrama una métrica más, utilizando esta fórmula, de modo que el campo será 1 cuando se pase el test y 0 cuando no se pase.

```
1 CASE
2     WHEN test = 'ok' THEN 1 ELSE 0
3 END
```

Ponemos la métrica de tipo númerico y con agregación sumatoria, aplicamos los cambios y ya tendremos una gráfica que aporta información mucho más valiosa:

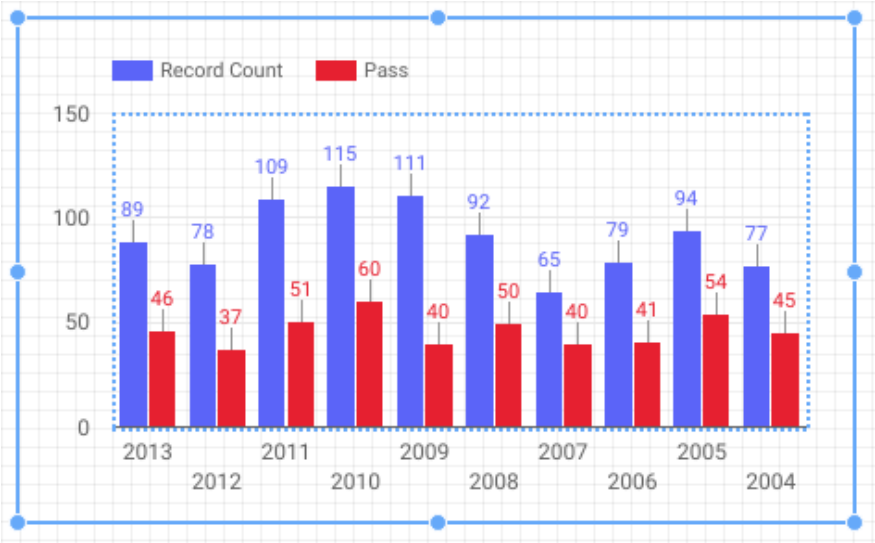

Figura 6: Número de películas por año enfrentadas a las que pasan el test.

Por último, vamos a probar otra de las posiblidades de Google Data Studio: **visualizaciones interactivas**. Vamos a permitir filtrar los registros a tener en cuenta según el presupuesto de los mismos.

Creamos un nuevo campo calculado en nuestro *datasource* (desde el propio Data Studio) con el que definimos los rangos de presupuesto deseados:

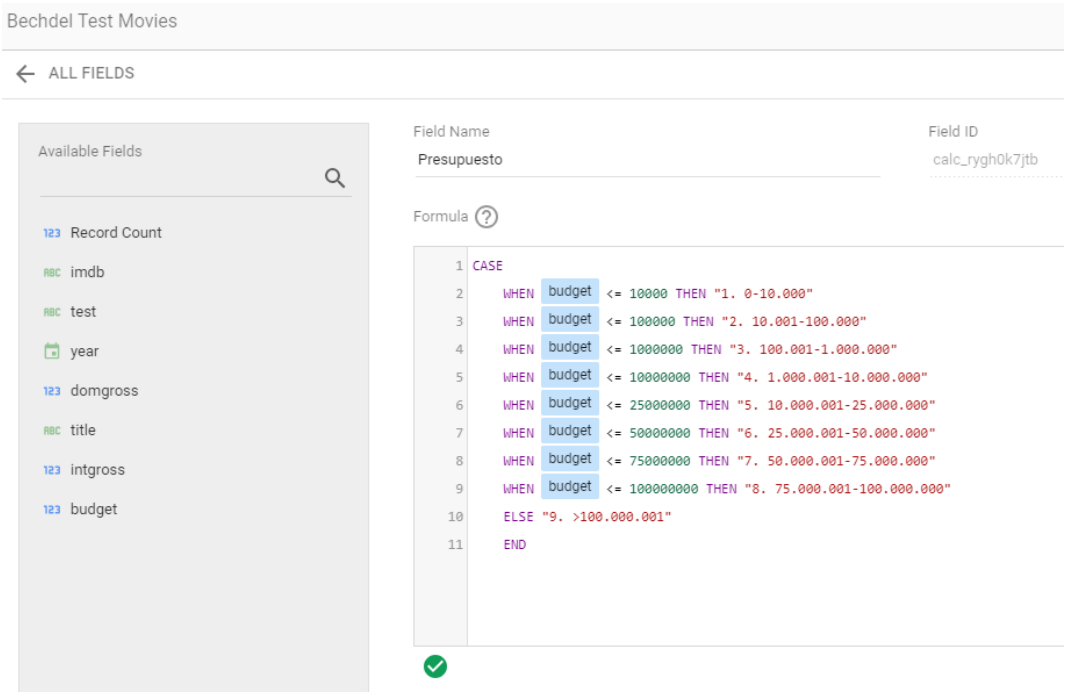

Figura 7: Campo de rangos de presupuestos añadido al *datasource*.

Añadimos a nuestro *report* un control de tipo filtro y escogemos como dimensión este nuevo campo de texto que hemos creado. Si escogemos una métrica como el contador de registros, al lado de cada opción del filtro aparecerá dicha métrica. Para comprobar el funcionamiento salimos del modo edición y podremos ver el diagrama interactivo:

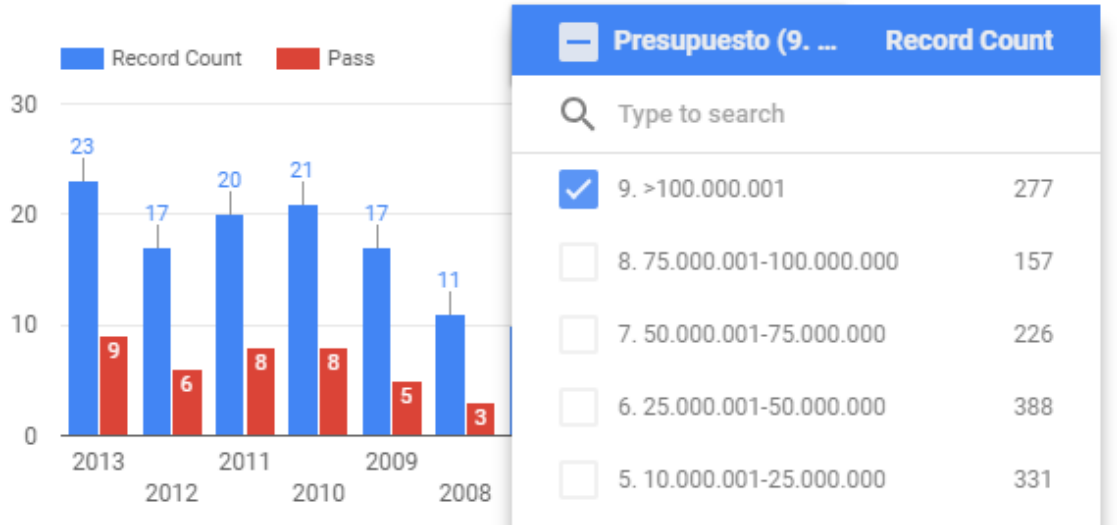

Figura 8: Gráfico de barras interactivo.

5. Conclusiones

Como hemos visto, Google Cloud Platform ofrece una serie de servicios que, utilizados de forma apropiada, nos sirven para llevar a cabo las fases más comunes de Data Science. Hemos partido de unos datos y, tras unas transformaciones como parte del preprocesado y unas exploraciones, hemos llegado a unas visualizaciones que arrojan información valiosa.

Este ha sido un ejemplo muy básico en el que apenas hemos puesto a prueba las capacidades de GCP, que como hemos comentado puede manejar grandes cantidades de datos y hacerlo incluso con una entrada por *streaming* en tiempo real.

En un caso práctico, se podría llegar más adelante y utilizar los datos para el entrenamiento de máquinas, popularmente conocido por su término inglés, *Machine Learning*, y para el que entraría en juego el servicio Google Cloud Machine Learning (ML) Engine.

A. Glosario

En orden alfabético:

- **Big Data**: Datasets de gran tamaño o complejidad.
- **Bucket**: En Google Cloud Storage, repositorio en el que se pueden almacenar ficheros planos.
- **Data Science**: Estadística enfocada bajo un contexto tecnológico.
- **Dataset**: Colección de datos.
- **Google BigQuery**: Solución en la nube para el análisis de Big Data.
- **Google Cloud Dataflow**: Solución de Google en la nube que permite implementar *pipelines* para procesar información.
- **Google Cloud Storage**: Solución de Google en la nube para el almacenamiento de ficheros.
- **Google Data Studio**: Solución de Google en la nube que permite crear y compartir visualizaciones de datos en forma de informes.
- **Pipeline**: Cadena de elementos de procesado, organizada de tal manera que la salida de un elemento es la entrada del siguiente.
- **Preprocesado de datos**: Proceso de preparación de los datos crudos para su exploración, eliminando posibles inconsistencias, omisiones o errores en los mismos.
- **Visualización de datos**: Presentación de la información en forma de infografías, diagramas, histogramas, etc.

Referencias

[1] *Productos y servicios de Google Cloud Platform* https://cloud.google.com/products/

[2] *Cloud Dataflow Documentation* https://cloud.google.com/dataflow/docs/

[3] Zdravko Markov, *Data preprocessing - Why preprocessing?* (CCSU, 2014) http://www.cs.ccsu.edu/~markov/ccsu_courses/DataMining-3.html

[4] *Página principal de Google BigQuery* https://cloud.google.com/bigquery/

[5] *Quickstart using the bq command-line tool* https://cloud.google.com/bigquery/docs/quickstarts/quickstart-command-line

[6] *Código completo del pipeline ejecutado en Dataflow* https://github.com/nulldiego/bechdel-pipeline-gcp/blob/master/pipeline.py

[7] T. Tereshko y J. Tigani, *Anatomy of a BigQuery Query* (Google Cloud Blog, 2016) https://cloud.google.com/blog/products/gcp/anatomy-of-a-bigquery-query

[8] B. Estes, *Google Data Studio: The Beginner's Tutorial* (Distilled, 2018) https://www.distilled.net/resources/google-data-studio-the-beginners-tutorial/