Homework 4: Big Data and Google Cloud

UIC CS 418, Spring 2019

*According to the **Academic Integrity Policy** of this course, all work submitted for grading must be done individually, unless otherwise specified. While we encourage you to talk to your peers and learn from them, this interaction must be superficial with regards to all work submitted for grading. This means you cannot work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. Academic dishonesty is unacceptable, and penalties range from failure to expulsion from the university; cases are handled via the official student conduct process described at https://dos.uic.edu/conductforstudents.shtml (https://dos.uic.edu/conductforstudents.shtml).*

This homework is an individual assignment for all graduate students. Undergraduate students are allowed to work in pairs and submit one homework assignment per pair. There will be no extra credit given to undergraduate students who choose to work alone. The pairs of students who choose to work together and submit one homework assignment together still need to abide by the Academic Integrity Policy and not share or receive help from others (except each other).

Due Date

This assignment is due at 11:59pm on April 25th, 2019.

Instructions

You need to complete all code and answer all questions denoted by **Q#** in this notebook. When you are done, you should export `hw4.ipynb` with your answers as a PDF file, upload that file `hw4.pdf` to *Homework 4 - written* on Gradescope, tagging each question.

Select your submission subdirectory `sub` and your completed Jupyter notebook (`hw4.ipynb` file) and zip it. As a result your `sub` subdirectory and `hw4.ipynb` file should be in the root of your zip file. Upload this zip file to *Homework 4 - code* on Gradescope.

For undergraduate students who work in a team of two, only one student needs to submit the homework and just tag the other student on Gradescope.

Keep an eye out for the following icons:

📝Specifies that you need to add something to the notebook in order to get credit.

💾Specifies that you need to save something to `sub` (submission) subdirectory or verify the existence of some file in `sub` to get credit.

⚠️Warnings to avoid common mistakes and pitfalls. Pay special attention to these.

Autograding

Q3 and Q4 will be graded based on your PDF submissions, the rest of the questions will be graded using an Autograder. All parts of Homework 4 are graded based on correctness, **not** based on completion.

Before We Start

Before we dive into the assignment, we need to install the following python packages:

- seaborn*
- matplotlib*
- nltk*
- sklearn*
- pandas*
- numpy*
- google-cloud
- google-cloud-storage
- google-cloud-bigquery[pandas]

Packages marked with * should be pre-installed in a conda environment. You may install missing packages one at a time using `pip` command or you can run the following shell command to install them all at once:

In [78]:
```
!pip3 install -r ./requirements.txt > pip-log.txt
```

⚠️If you are installing using the shell command listed above then pay attention to any warnings and check the logs `pip-log.txt` .

Now, we are ready to import all dependencies.

In [79]:
```
from google.cloud import storage
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA
import pandas as pd
import numpy as np
import os

%load_ext google.cloud.bigquery
```

```
The google.cloud.bigquery extension is already loaded. To reload it, use:
  %reload_ext google.cloud.bigquery
```

We also need to ensure that we have required nltk packages installed:

In [80]:
```
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Kruneet\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Kruneet\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Kruneet\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[80]:
```
True
```

Now that we have everything we need, we can dive into the homework itself.

Cloud Services

Most real world systems in the modern world generate copious amount of data (billions of records) and in order to process this data and to enable reasoning through statistical models, we need high computational power and memory which is not available in most personal machiens.

One solution is to build and maintain your own clusters of high-end machines which has a massive overhead and cost associated with it, which is definitely not suitable for individuals or small teams. Hence, most individuals and companies rely on cloud service providers for a range of services, including but not limited to storage, processing, querying and visualizing large datasets.

In this homework, we will use Google Cloud (https://cloud.google.com/) to query and visualize data and we will train and deploy a machine learning model in the cloud. The rest of the homework is divided in 4 parts.

Part 0: Setup: Initial configurations

Part 1: Big Query: A database for Big Data

Part 2: Data Studio: A visualization tool for Big Data

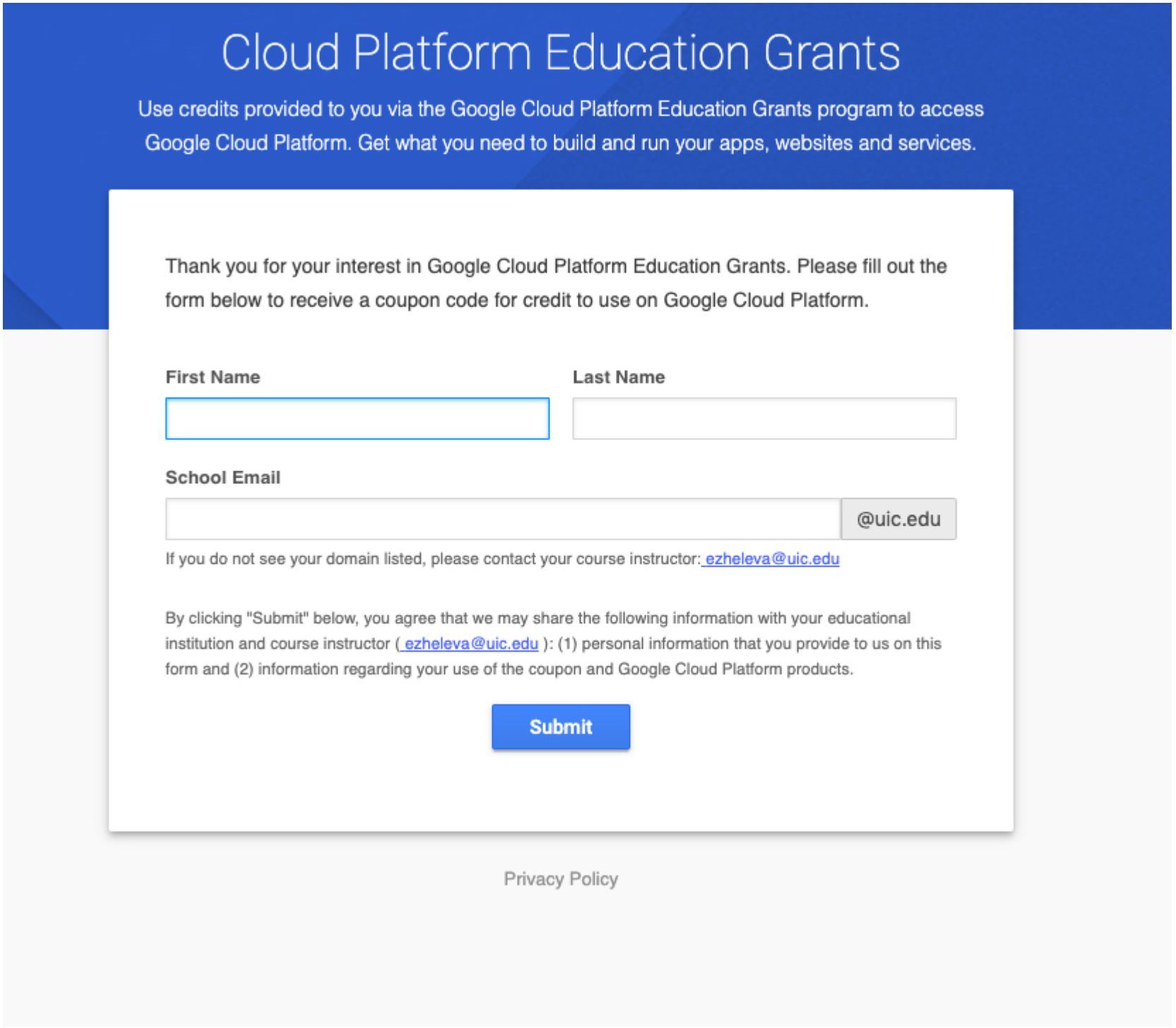
Part 3: Cloud ML: Machine Learning with Big Data

These services can save a lot of time and make your life easier when handling a large amount of data which cannot be processed using a single machine. So let's get started!

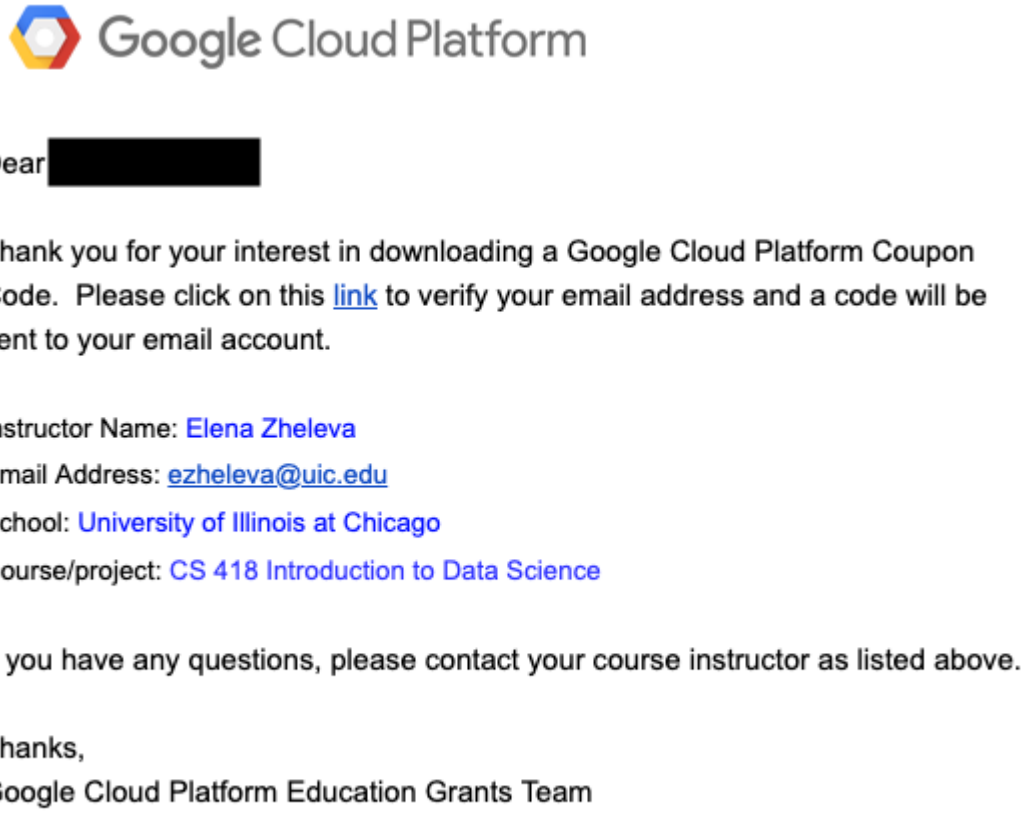
Part 0: Setup

Setup Billing

Follow this link (https://google.secure.force.com/GCPEDU?cid=GUXjYCCfVw4mzQIPG%2BXSNPhqmoOK9tztTUFuHYq08DR%2FLLuoWCWXg7YRDlMecFO2) to claim $50 credit using your @uic.edu email. This homework can be completed while utilizing less than $5. Fill up the details in following form and it will send you an email to verify your account.


Cloud Platform Education Grants
Use credits provided to you via the Google Cloud Platform Education Grants program to access Google Cloud Platform. Get what you need to build and run your apps, websites and services.
Thank you for your interest in Google Cloud Platform Education Grants. Please fill out the form below to receive a coupon code for credit to use on Google Cloud Platform.
First Name
Last Name
School Email
@uic.edu
If you do not see your domain listed, please contact your course instructor: ezheleva@uic.edu
By clicking "Submit" below, you agree that we may share the following information with your educational institution and course instructor (ezheleva@uic.edu): (1) personal information that you provide to us on this form and (2) information regarding your use of the coupon and Google Cloud Platform products.
Submit
Privacy Policy

After you have received your verification email (shown below), follow the link in email to request a coupon



Google Cloud Platform

Dear [redacted]

Thank you for your interest in downloading a Google Cloud Platform Coupon Code. Please click on this link to verify your email address and a code will be sent to your email account.

Instructor Name: Elena Zheleva
Email Address: ezheleva@uic.edu
School: University of Illinois at Chicago
Course/project: CS 418 Introduction to Data Science

If you have any questions, please contact your course instructor as listed above.

Thanks,
Google Cloud Platform Education Grants Team

You will receive a coupon in email as shown below, follow the link to redeem it.

Dear [redacted]

Here is your Google Cloud Platform Coupon Code: [redacted]-21TF-602U

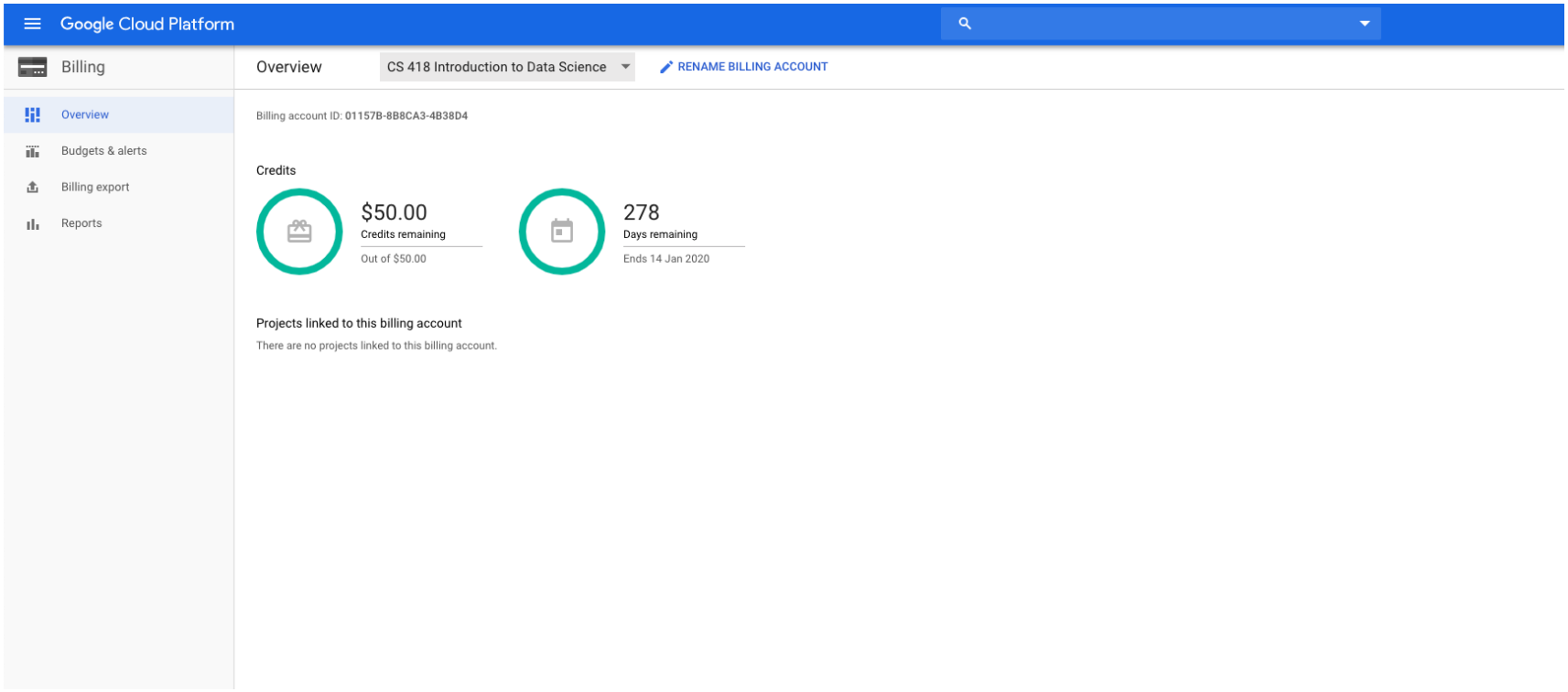
Click [here] to redeem.

Course/Project Information
Instructor Name: Elena Zheleva
Email Address: ezheleva@uic.edu
School: University of Illinois at Chicago
Course/project: CS 418 Introduction to Data Science
Activation Date: 14/01/2019
Redeem By: 14/05/2019
Coupon Valid Through: 14/01/2020

If you have any questions, please contact your course instructor as listed above.

Thanks,
Google Cloud Platform Education Grants Team

Once you accept and add credit to your account, you will be sent to the following page.

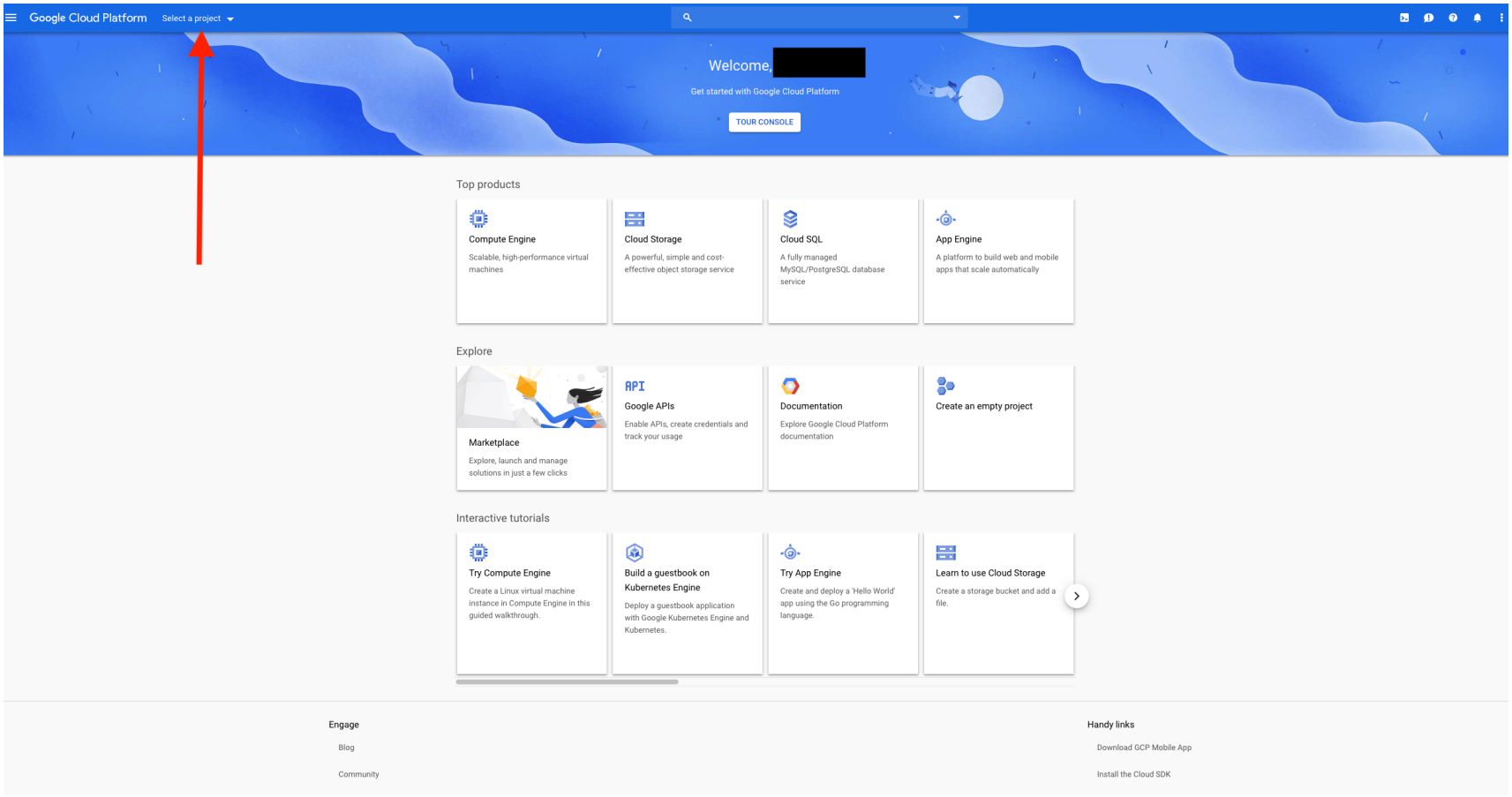


Verify that $50 have been added to your billing account "CS 418 Introduction to Data Science"

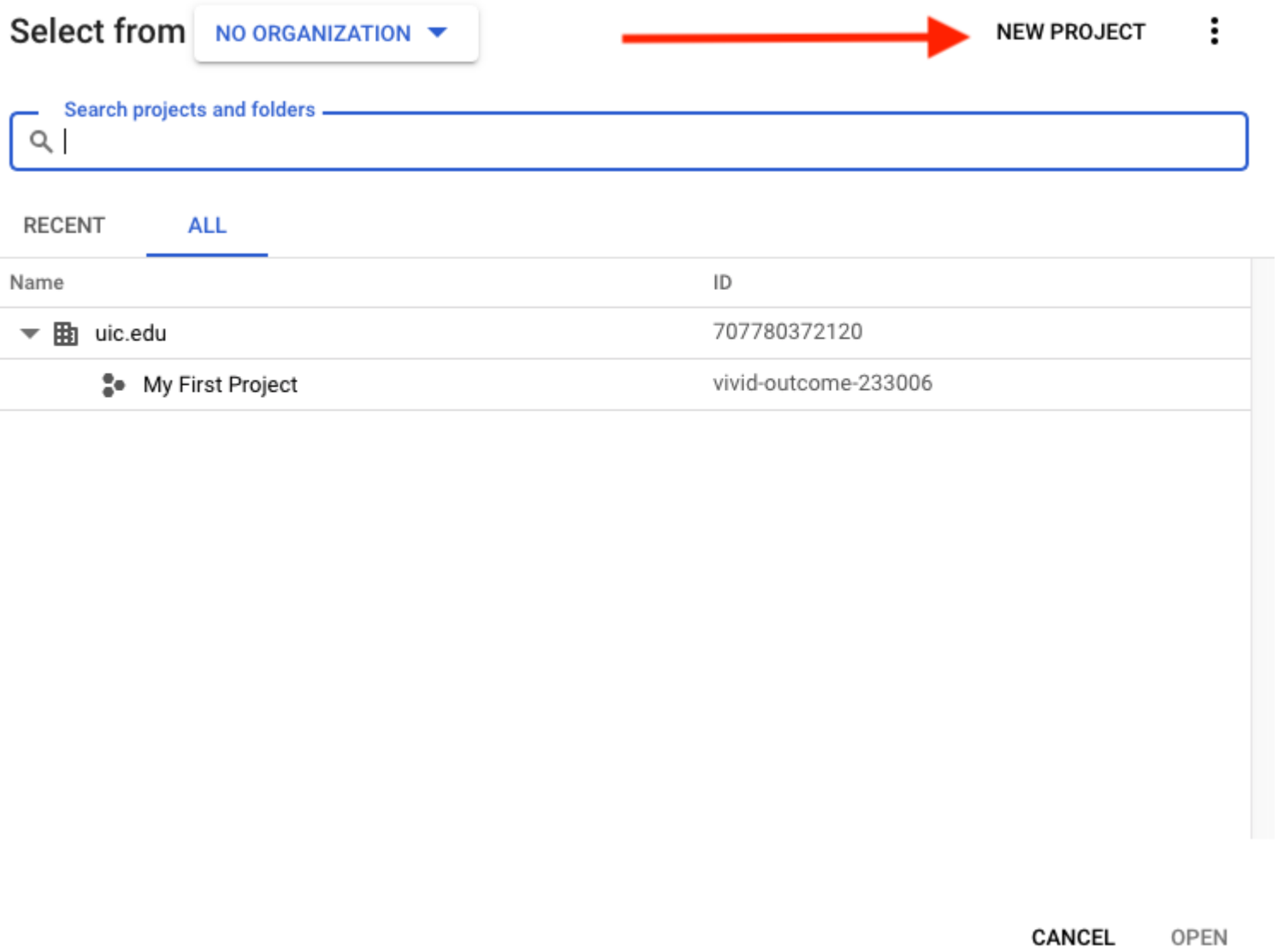
Create Project

Google Cloud provides a common interface to manage and monitor all projects, this is called Google Cloud Console. First, you need to log into Google Cloud Console (https://console.cloud.google.com/) using your university ID.

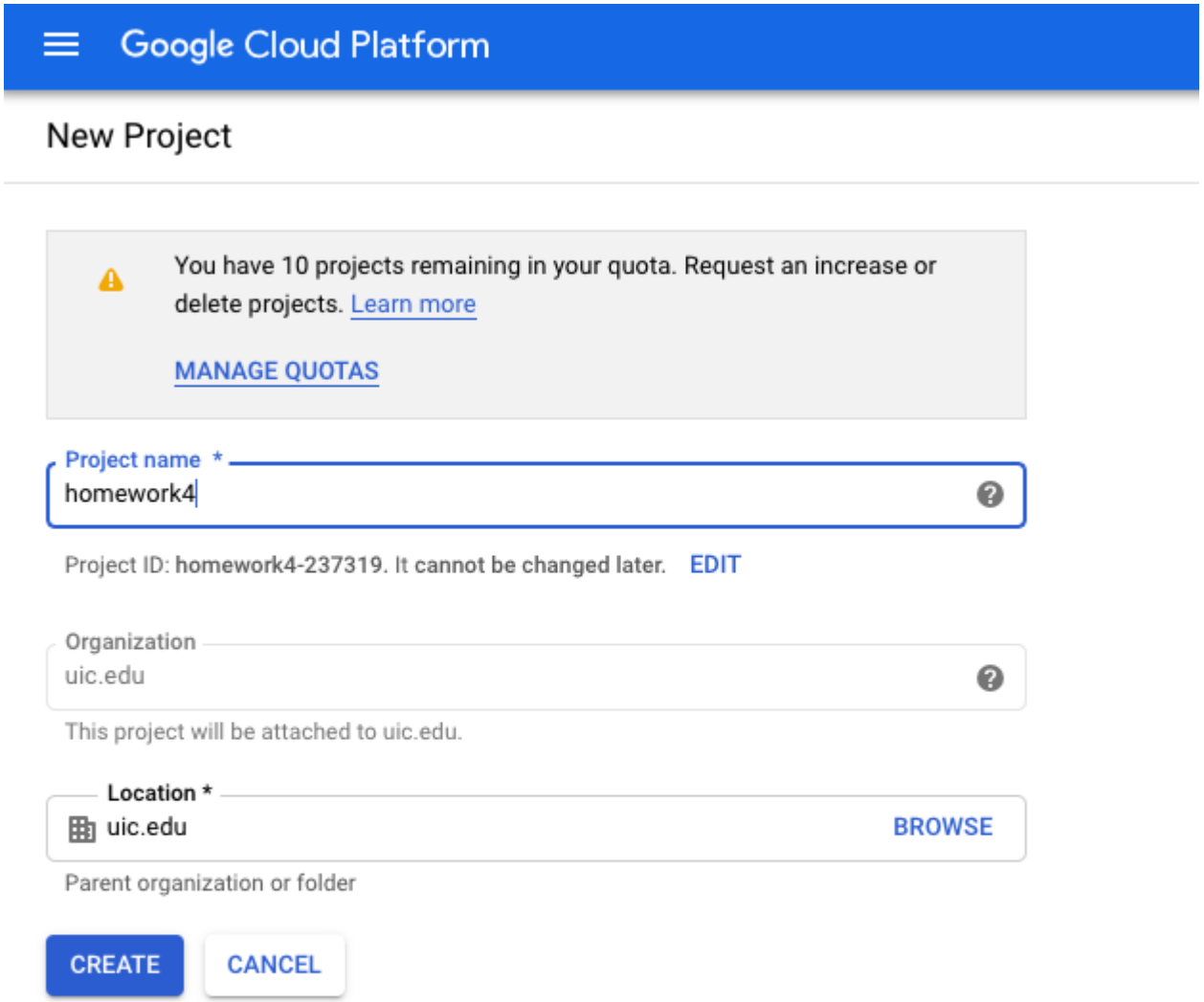
After logging in, you will land on a welcome page which, assuming you have not setup a project before, lists some popular services and tutorials shown below:



Click on the dropdown "Select a Project" indicated by the red arrow in the image above and you will be taken to this menu:

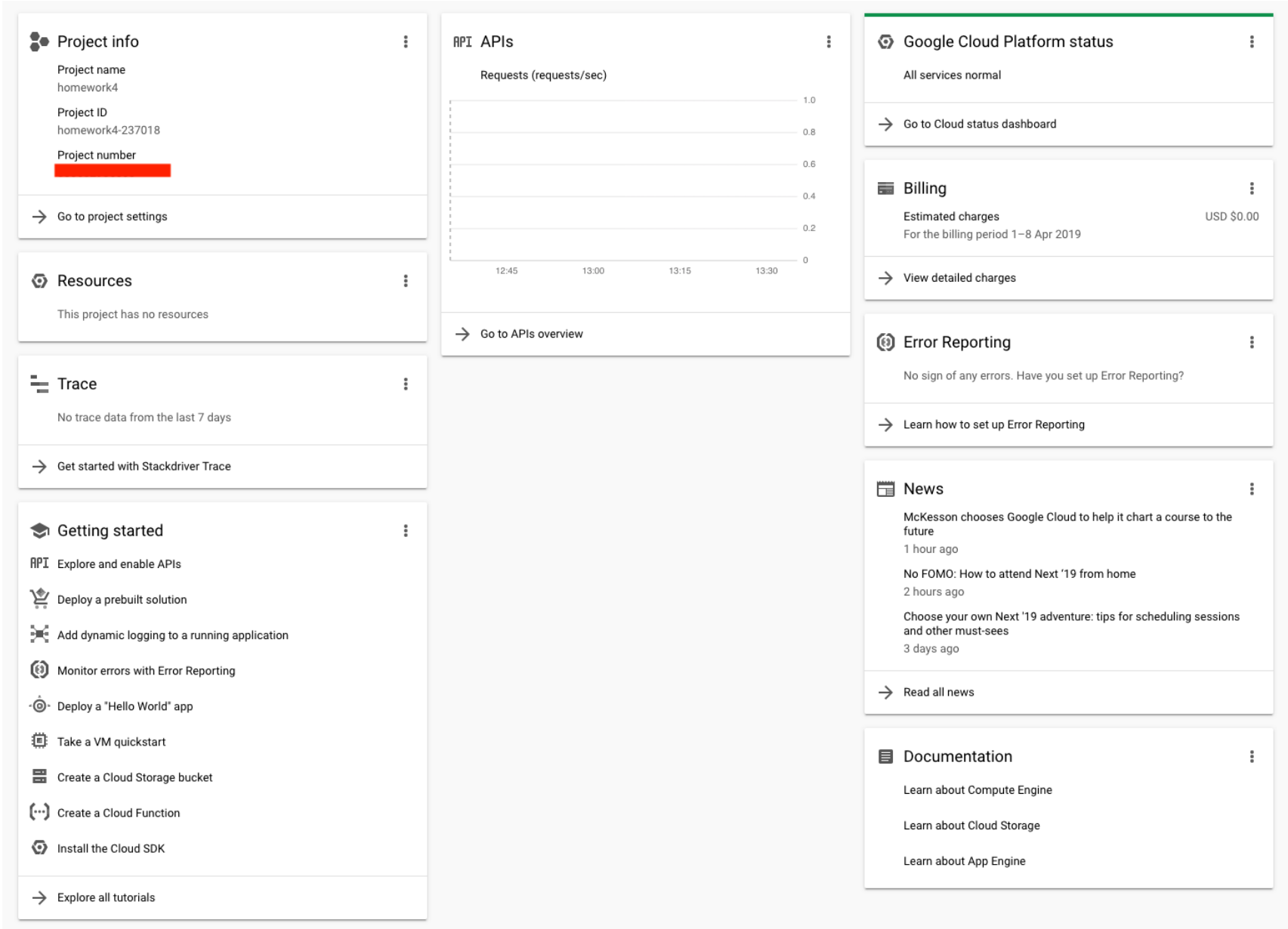


Click on the New Project button as shown above and it will take you to project form:



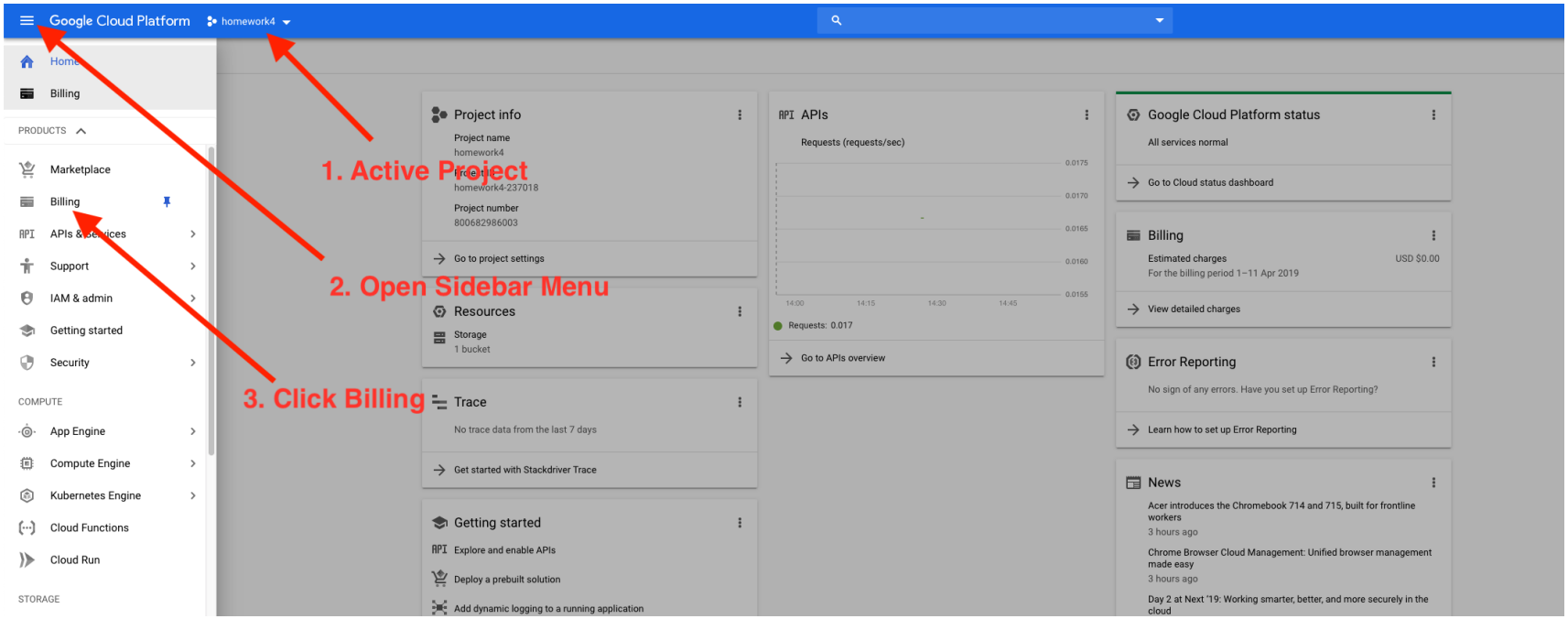
Name your project `homework4` and click *Create* button. And just as easily, your project is set up.

All the Google Cloud services are offered within the context of a project and once the project is created, you will be redirected to project dashboard (or Google Cloud Console) where you can add services to the project, monitor activity and manage billing etc.



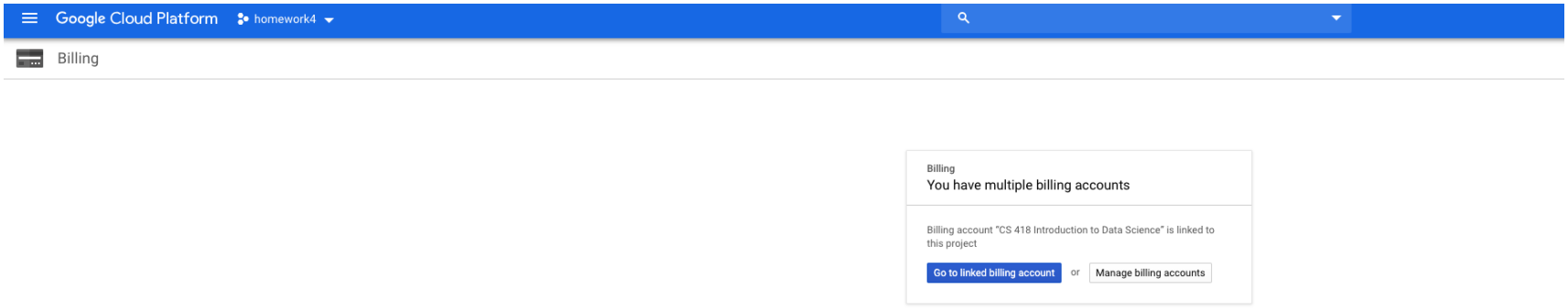
Link Billing Account

Before, we proceed, we need to link a billing account to our project or verify that the previously created billing account is linked to our project automatically. First, ensure that the correct project is selected, then open the left sidebar menu using the button on top-left and select Billing.



This will take you to a screen which specifies the billing account linked to your project. Verify that the linked account is "CS 418 Introduction to Data Science".

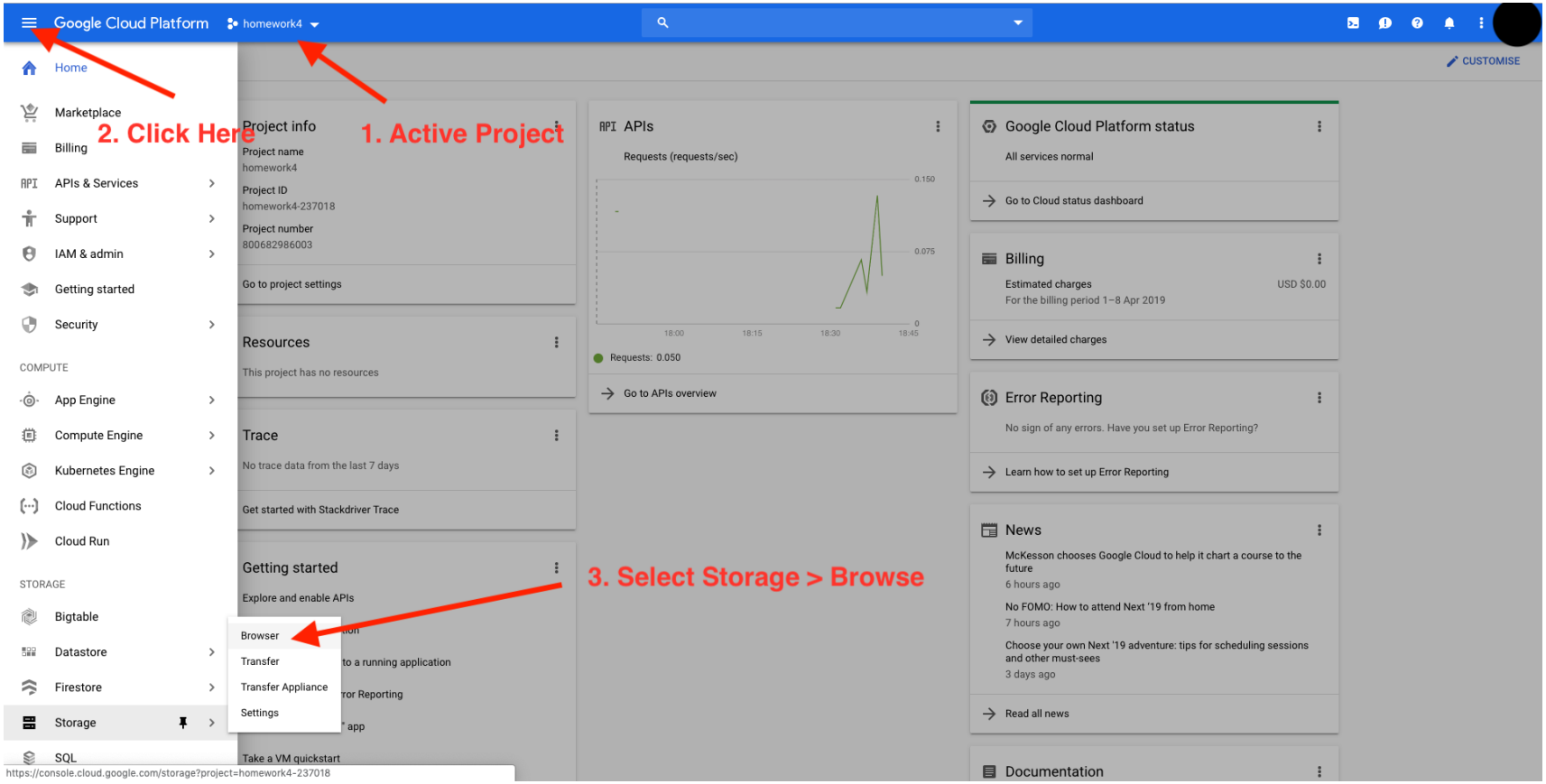
⚠️You might have multipled billing accounts, as shown in the image below, in that case, make sure that the correct account is linked to avoid any unnecessary cost.



Now that our project is setup, let's add a service to it!

Adding a Bucket

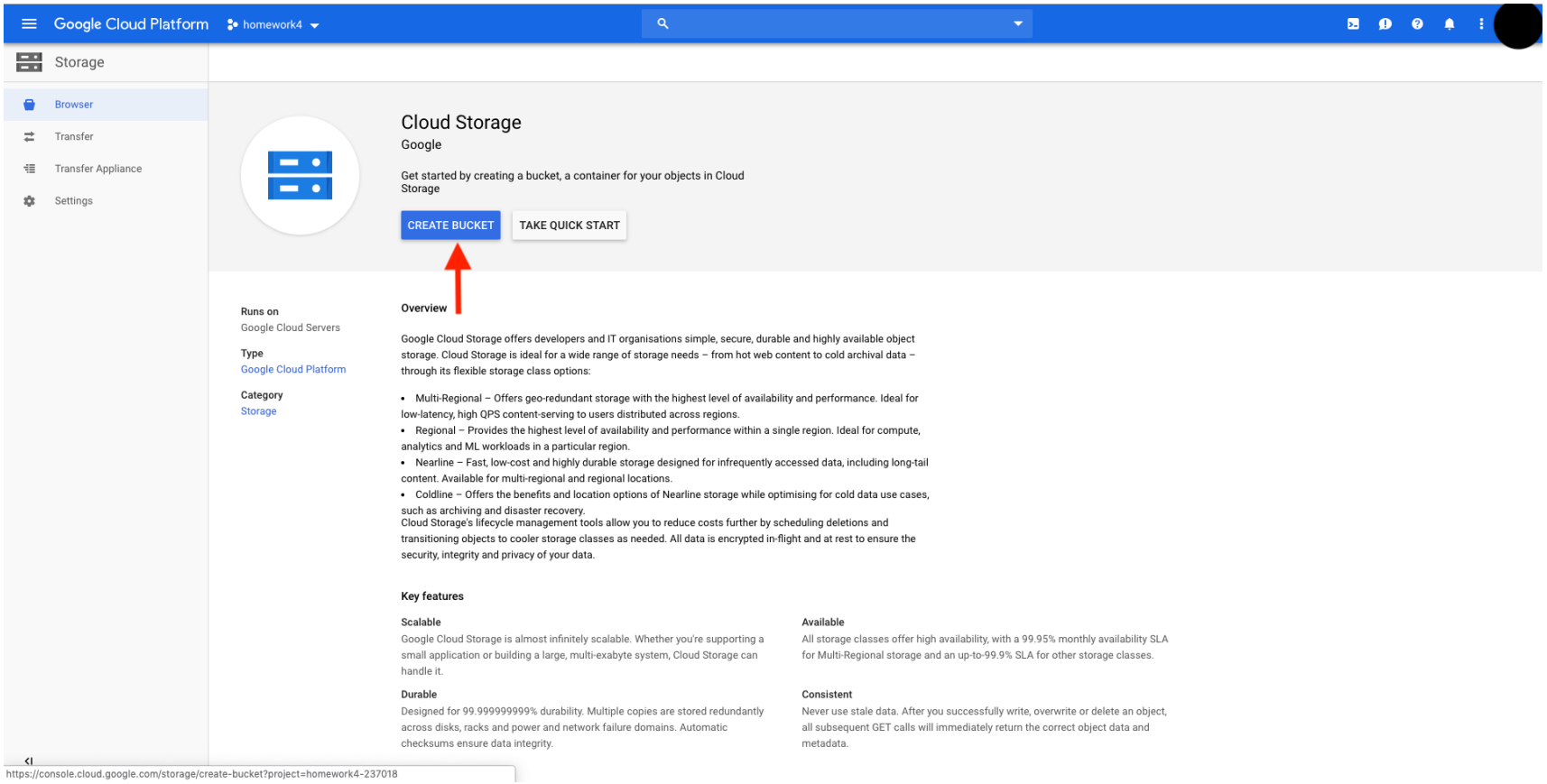
A bucket is just a storage container, it serves the same purpose as a cloud storage service (e.g. *Google Drive*) but it is more suitable for programmatic access and integration with other services as we will see later. Buckets are managed through a service *Cloud Storage* and we can add a bucket to our project through the sidebar (topleft menu) as shown in the following image:



Before you add any service, make sure that your active project is the one you just created. Then you can select the sidebar menu by clicking on the top-left button and add services from the menu.

⚠️Adding services may incur a high cost so only add what you need.

Selecting *Browse* will take you to the following screen:



Click on the *Create Bucket* button as shown above and you will be taken to the following form:

Name

Must be unique across Cloud Storage. If you're serving website content, enter the website domain as the name.

model-bucket-alpha

Default storage class

Objects added to this bucket are assigned the selected storage class by default. An object's storage class and bucket location affect its geo-redundancy, availability and costs. You can set storage classes for individual objects in gsutil. Learn more

Nearline and Coldline data in multi-regional locations is now stored geo-redundantly. New locations nam4 and eur4 (available in beta) enable co-location of compute and storage for high performance with geo-redundancy. Learn more

Dismiss

- ○ Multi-Regional
- ◉ Regional
- ○ Nearline
- ○ Coldline

Location

us-central1 (Iowa)

Compare storage classes

Storage cost	Retrieval cost	Class A operations	Class B operations
$0.02 per GB-month	Free	$0.005 per 1,000 ops	$0.0004 per 1,000 ops

Access control model

Choose how you'll control access to this bucket's objects. Learn more

- ○ Set permissions uniformly at bucket-level (Bucket Policy Only)
  Enforces the bucket's IAM policy without object ACLs. May help prevent unintended access. If selected, this option becomes permanent after 90 days.
- ◉ Set object-level and bucket-level permissions
  Enforces the IAM policy and object ACLs for more granular control of object access.

Show advanced settings

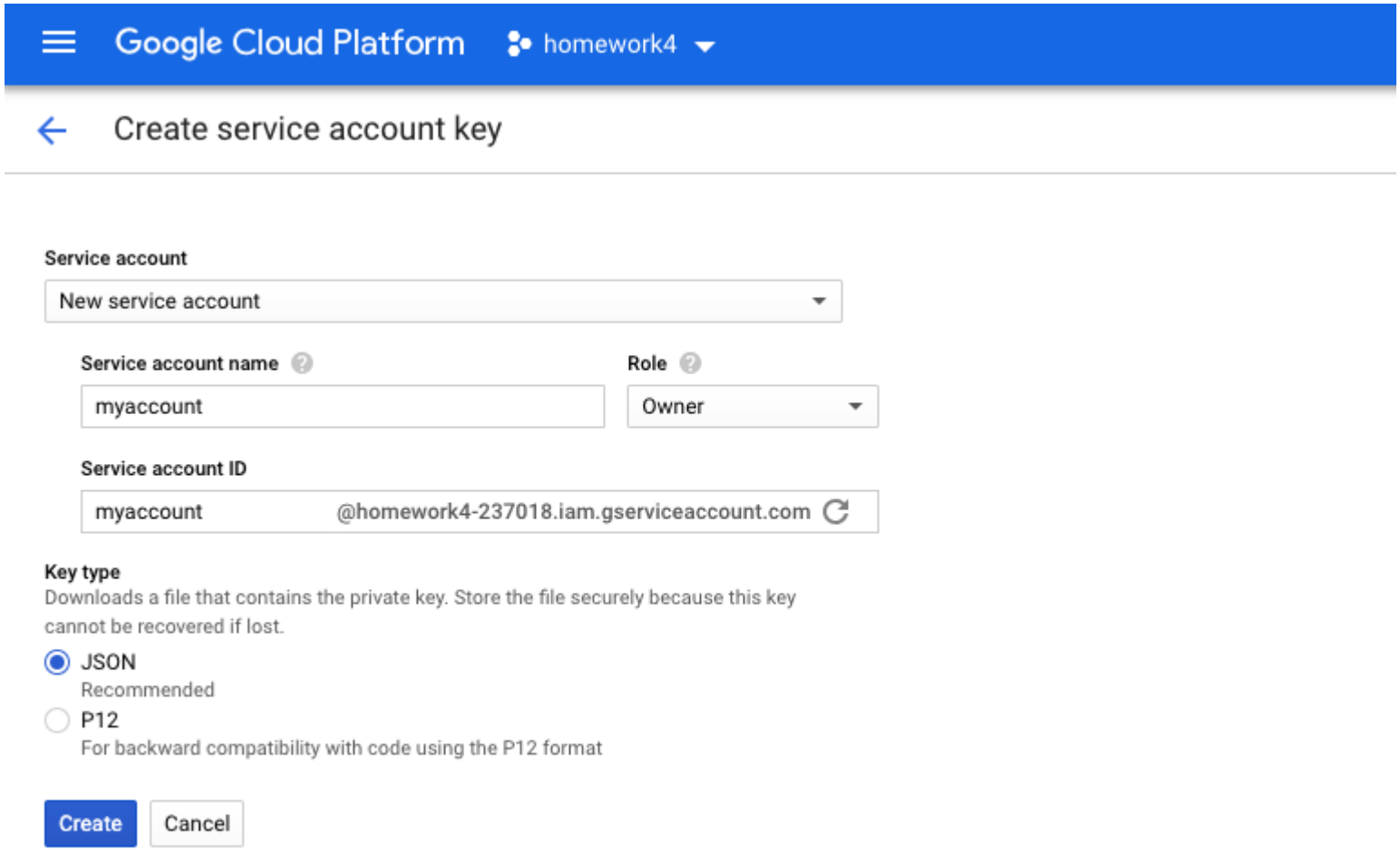
Create Cancel

Fill out the form as shown above (use the provided links to learn more about these fields), however you need to enter a unique bucket name and press create. Now you should be able to view your bucket in a file browser like environment. We will come back to this later, first, let's set up authentication first so that we can communicate with Google Cloud.

Setup Authentication

We will follow the simple steps listed in this tutorial (Authentication tutorial (https://cloud.google.com/docs/authentication/getting-started)) to setup authentication.

First, we need to setup a service account, open the tutorial and click on *GO TO THE CREATE SERVICE ACCOUNT KEY PAGE* button and fill up the form as shown below. When the account is created, you will be prompted to download a JSON key, save it in homework root directory as `homework4-key.json` .



We can now setup an enviornment variable `GOOGLE_APPLICATION_CREDENTIALS` using the following python command, this enables the google cloud API to use the JSON key while making requests.

In [81]:
```
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = 'homework4-key.json'
```

Now, we can test our setup by making an authenticated API call to list buckets.

In [82]:
```
def list_buckets():
    storage_client = storage.Client()
    # Make an authenticated API request
    buckets = list(storage_client.list_buckets())
    print(buckets)


list_buckets()
# The expected output of this command should contain the name of your bucket
# [<Bucket: [YOUR BUCKET NAME]>]
```

```
[<Bucket: model-bucket-kruneet>]
```

Using this JSON key, we can now communicate with server without having to explicitly log in.

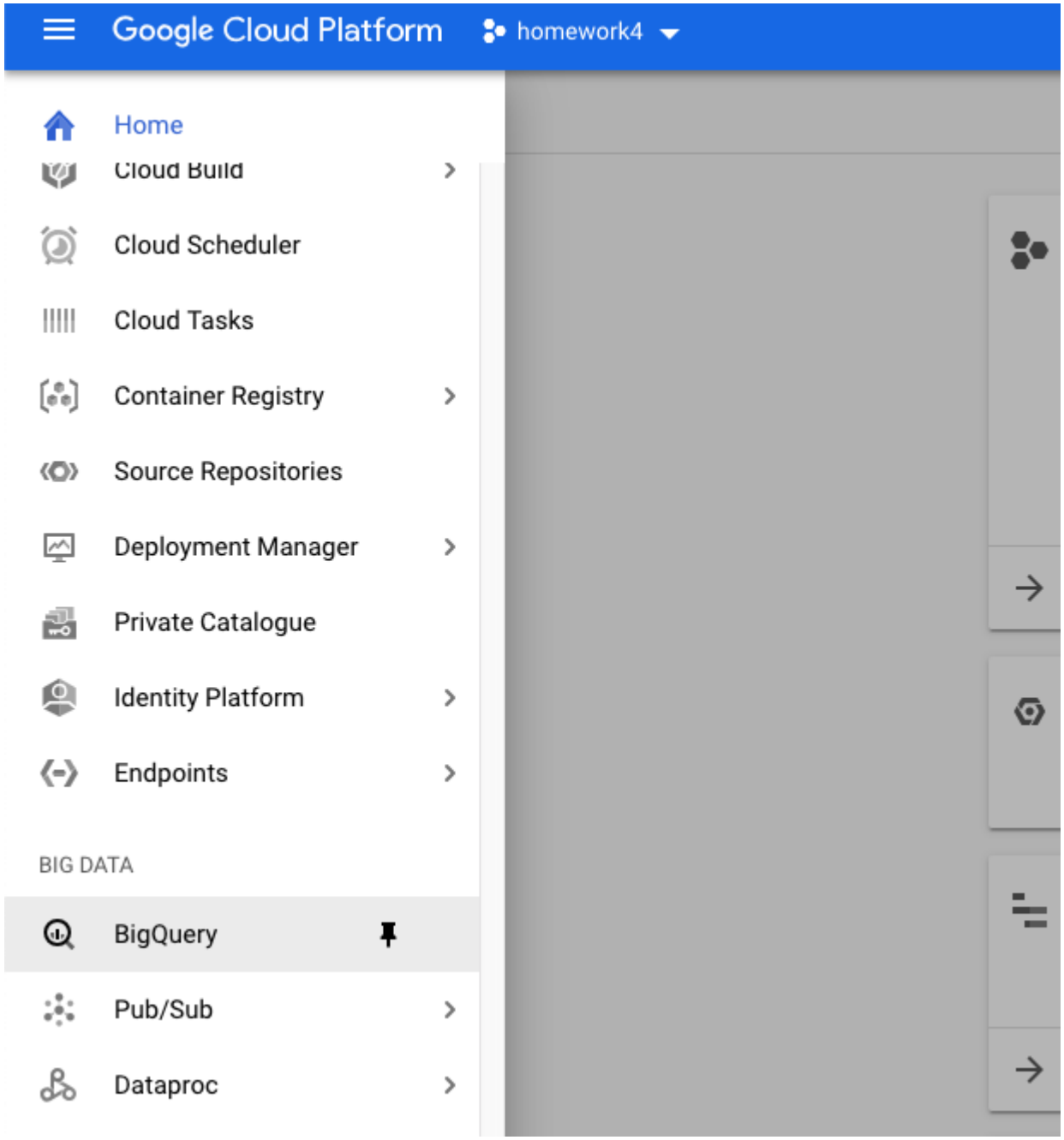
⚠️ `homework4-key.json` contains sensitive information that you should not share with anyone. Do not submit this file.

Part 1: BigQuery

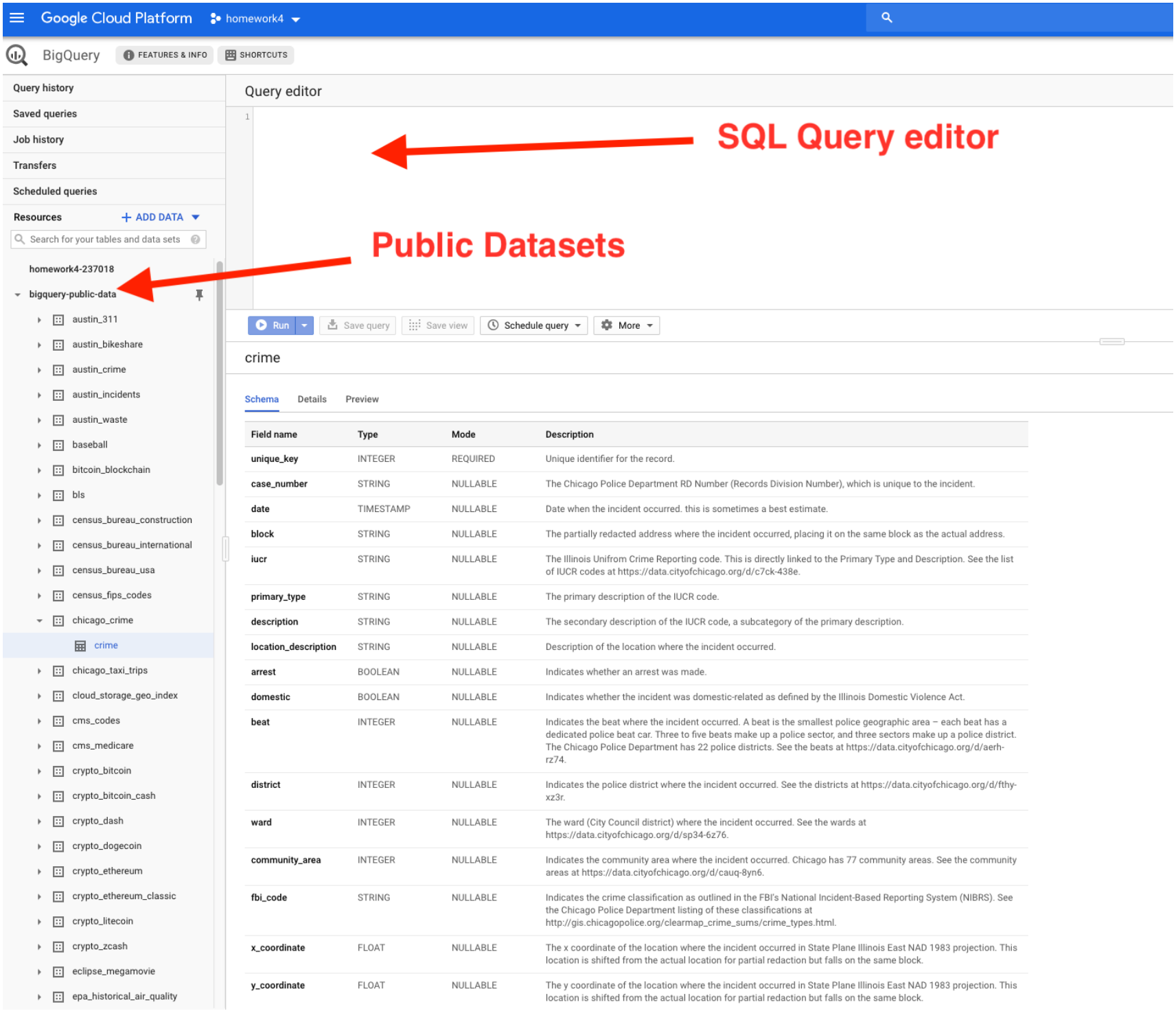
BigQuery is a large scale database which can process multiple GBs of data (e.g. all crime records in Chicago since 2001) within just a few seconds and provides the capacity to store and retrieve structured information which can either be used by your application or another cloud service. You can read more about it on the project page (https://cloud.google.com/bigquery/).

One of the key strenghts of BigQuery is that it supports Structured Query Language (SQL) which is the de facto query language for Relational Databases in which data is stored in tabular form (just like pandas! but not in-memory). This enables the developers to query large databases without having to learn a new language, however, if you are not familiar with SQL or a simple refresher course then you can get a quick introduction at W3 Schools (https://www.w3schools.com/sql/).

Let's first go to BigQuery interface and explore some large scale public datasets. From the project dashboard (https://console.cloud.google.com) select BigQuery from the sidebar:



This will take you to the BigQuery interface. For this homework, we will be using publicly available chicago crime data which encompasses crime records in Chicago since 2001. It is already structured, cleaned and maintained by Google and we can just directly query it, however we can also add our own datasets to BigQuery if needed.



Field name	Type	Mode	Description
unique_key	INTEGER	REQUIRED	Unique identifier for the record.
case_number	STRING	NULLABLE	The Chicago Police Department RD Number (Records Division Number), which is unique to the incident.
date	TIMESTAMP	NULLABLE	Date when the incident occurred. this is sometimes a best estimate.
block	STRING	NULLABLE	The partially redacted address where the incident occurred, placing it on the same block as the actual address.
iucr	STRING	NULLABLE	The Illinois Unifrom Crime Reporting code. This is directly linked to the Primary Type and Description. See the list of IUCR codes at https://data.cityofchicago.org/d/c7ck-438e.
primary_type	STRING	NULLABLE	The primary description of the IUCR code.
description	STRING	NULLABLE	The secondary description of the IUCR code, a subcategory of the primary description.
location_description	STRING	NULLABLE	Description of the location where the incident occurred.
arrest	BOOLEAN	NULLABLE	Indicates whether an arrest was made.
domestic	BOOLEAN	NULLABLE	Indicates whether the incident was domestic-related as defined by the Illinois Domestic Violence Act.
beat	INTEGER	NULLABLE	Indicates the beat where the incident occurred. A beat is the smallest police geographic area – each beat has a dedicated police beat car. Three to five beats make up a police sector, and three sectors make up a police district. The Chicago Police Department has 22 police districts. See the beats at https://data.cityofchicago.org/d/aerh-rz74.
district	INTEGER	NULLABLE	Indicates the police district where the incident occurred. See the districts at https://data.cityofchicago.org/d/fthy-xz3r.
ward	INTEGER	NULLABLE	The ward (City Council district) where the incident occurred. See the wards at https://data.cityofchicago.org/d/sp34-6z76.
community_area	INTEGER	NULLABLE	Indicates the community area where the incident occurred. Chicago has 77 community areas. See the community areas at https://data.cityofchicago.org/d/cauq-8yn6.
fbi_code	STRING	NULLABLE	Indicates the crime classification as outlined in the FBI's National Incident-Based Reporting System (NIBRS). See the Chicago Police Department listing of these classifications at http://gis.chicagopolice.org/clearmap_crime_sums/crime_types.html.
x_coordinate	FLOAT	NULLABLE	The x coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection. This location is shifted from the actual location for partial redaction but falls on the same block.
y_coordinate	FLOAT	NULLABLE	The y coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection. This location is shifted from the actual location for partial redaction but falls on the same block.

Browse and select the table from the sidebar as shown in the figure and you will be able to see all the fields in the table and their description. You can also select *Preview* to show a subset of the dataset.

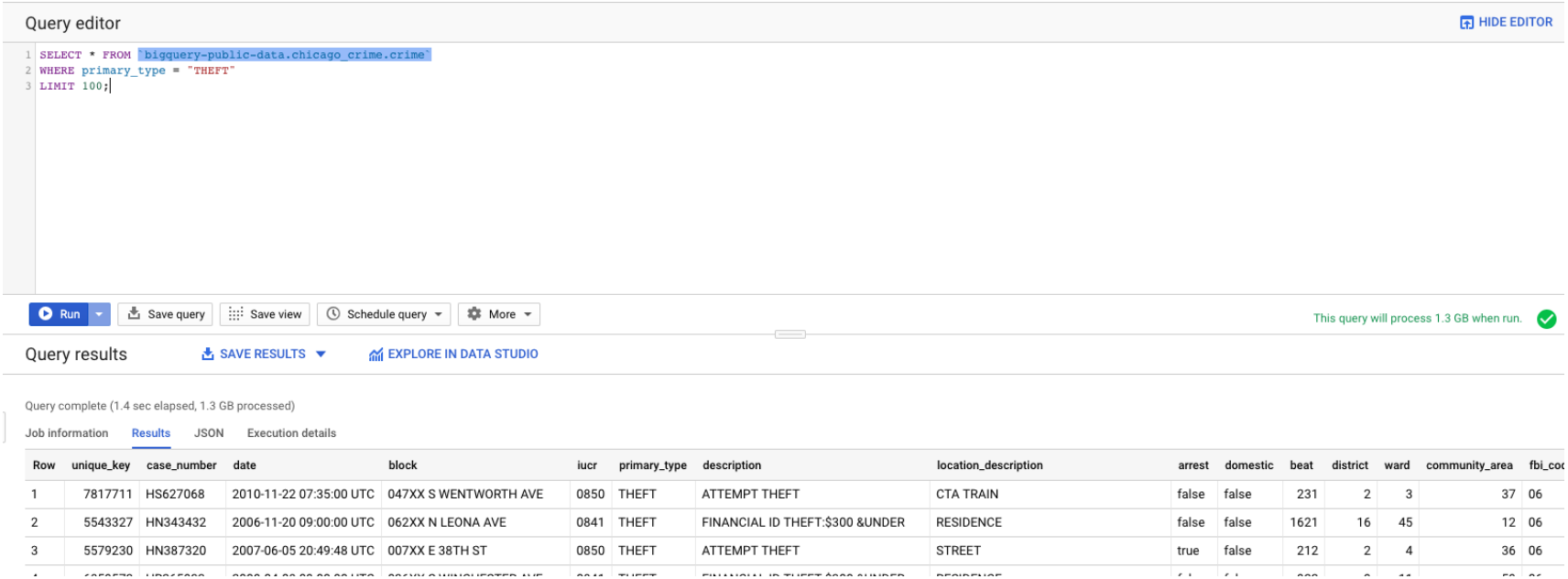
Let's run this simple query using the Query Editor

```
#standardSQL

SELECT * FROM `bigquery-public-data.chicago_crime.crime`
WHERE primary_type = "THEFT"
LIMIT 100;
```

This query will select the top 100 crime records of type theft form the crime dataset.

⚠ ` or backtick is a character which is not the same as an apostrophe or double quotes and is often used in SQL.

⚠ You always need to enter the prefix of the table or the dataset identifier *bigquery-public-data.chicago_crime* to access *crime* table.

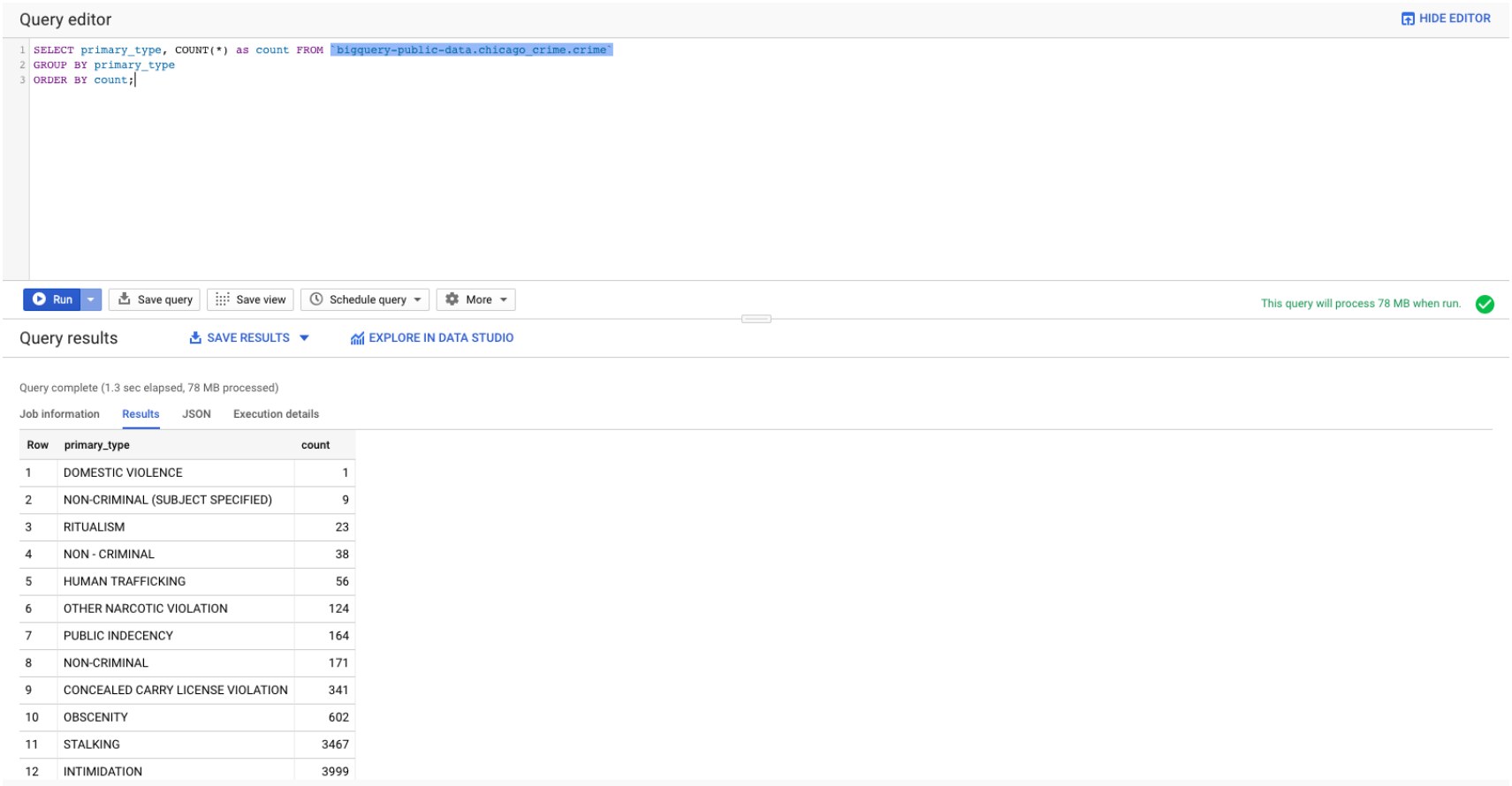


Row	unique_key	case_number	date	block	iucr	primary_type	description	location_description	arrest	domestic	beat	district	ward	community_area	fbi_cod
1	7817711	HS627068	2010-11-22 07:35:00 UTC	047XX S WENTWORTH AVE	0850	THEFT	ATTEMPT THEFT	CTA TRAIN	false	false	231	2	3	37	06
2	5543327	HN343432	2006-11-20 09:00:00 UTC	062XX N LEONA AVE	0841	THEFT	FINANCIAL ID THEFT:$300 &UNDER	RESIDENCE	false	false	1621	16	45	12	06
3	5579230	HN387320	2007-06-05 20:49:48 UTC	007XX E 38TH ST	0850	THEFT	ATTEMPT THEFT	STREET	true	false	212	2	4	36	06

In the figure above, you can see the results listed as a table. The Query editor shows you how much data it is going to process when the query runs so that you may estimate the cost. In this case, we are running this query on 1.3 GB of data within a few seconds. You may also export results in CSV format or explore them in Data Studio (a visualization portal that we will discuss later). Just like `pandas`, you may also perform a Group By operation and order the results using the following query:

```
SELECT primary_type, COUNT(*) as count FROM `bigquery-public-data.chicago_crime.crime`
GROUP BY primary_type
ORDER BY count;
```

The results are as follows:



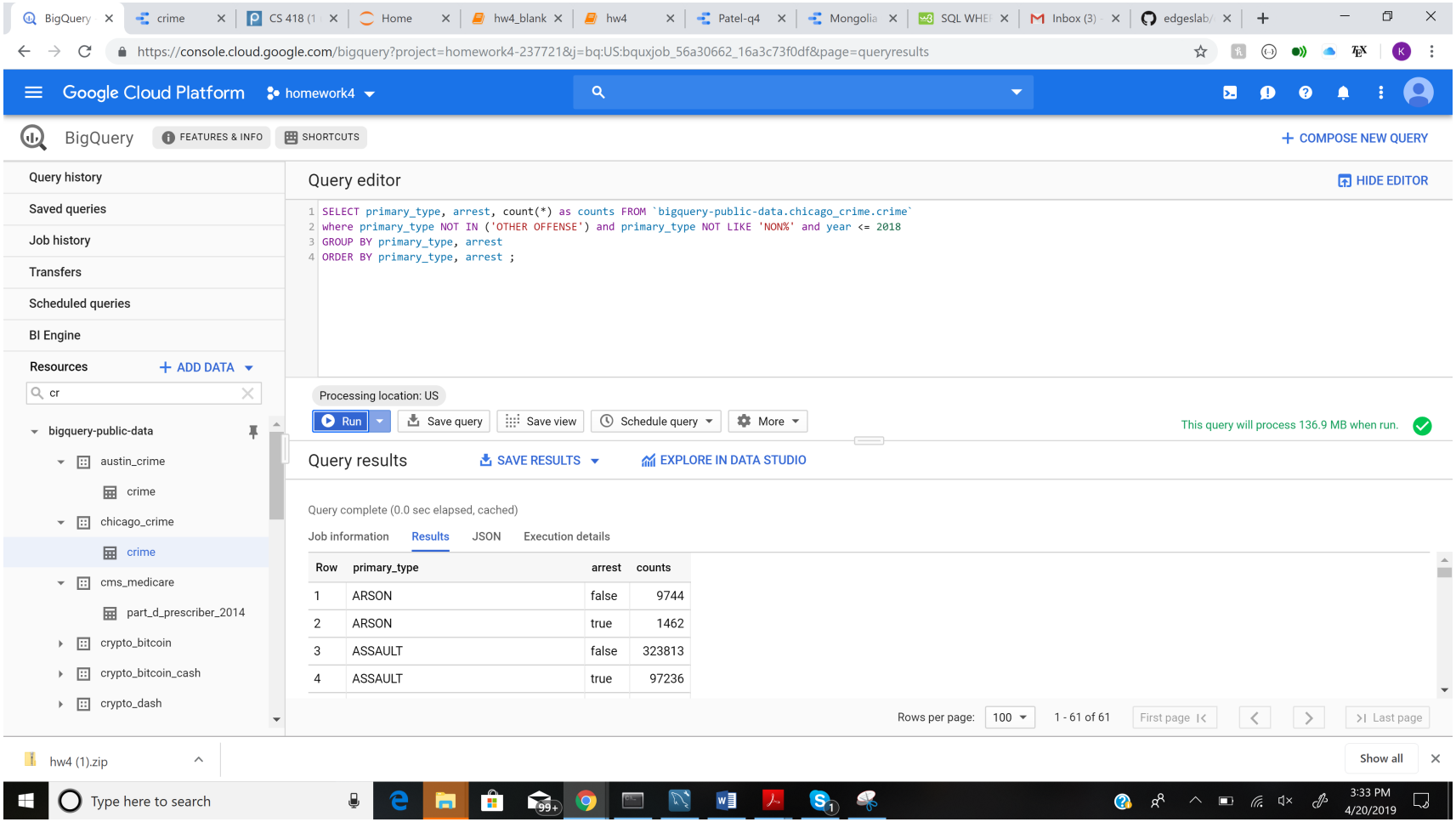
Row	primary_type	count
1	DOMESTIC VIOLENCE	1
2	NON-CRIMINAL (SUBJECT SPECIFIED)	9
3	RITUALISM	23
4	NON - CRIMINAL	38
5	HUMAN TRAFFICKING	56
6	OTHER NARCOTIC VIOLATION	124
7	PUBLIC INDECENCY	164
8	NON-CRIMINAL	171
9	CONCEALED CARRY LICENSE VIOLATION	341
10	OBSCENITY	602
11	STALKING	3467
12	INTIMIDATION	3999

Q1 (20%)

📝Write an SQL query to extract the number of arrests/no-arrests per primary type, exclude "OTHER OFFENSE" and all non-criminal types, only consider records until the end of year 2018 and sort the results in ascending order by primary type and arrest.

💾Take a screenshot (full screen, png format), rename it as `q1-results.png` and store it in `sub` subdirectory.

📝Include screenshot in the notebook by editing the following markdown (if needed). It shows as a broken image by default if screenshot is unavailable.



⚠️Refresh your browser if it does not appear right away.

Since `pandas` and SQL operate on tables and offer similar functionality, we can transfer SQL results to a `pandas` data frame for further processing. One arduous way to do it is to first export results from BigQuery interface as a CSV file and load it in pandas, but there is a much better and convenient method available through the `bigquery` magic command. The following statement will run a query on the server and store results in the specified `df` dataframe, while running it shows the time elapsed.

In [83]:
```
%%bigquery df

-- This is an SQL comment


SELECT * FROM `bigquery-public-data.chicago_crime.crime`
WHERE primary_type = "THEFT"
LIMIT 100;
```

In [84]:
```
# We can now print a subset of these results using this familiar method
df.head()
```

Out[84]:

	unique_key	case_number	date	block	iucr	primary_type	description	location_description	arrest	domestic	..
0	9853569	HX502222	2014-11-10 15:00:00+00:00	105XX S DRAKE AVE	0850	THEFT	ATTEMPT THEFT	RESIDENCE	True	False	..
1	9440430	HW584466	2013-12-26 12:45:00+00:00	074XX W TALCOTT AVE	0870	THEFT	POCKET-PICKING	HOSPITAL BUILDING/GROUNDS	False	False	..
2	9431030	HW574031	2013-12-16 16:00:00+00:00	103XX S WOODLAWN AVE	0870	THEFT	POCKET-PICKING	CTA BUS	False	False	..
3	9423166	HW566638	2013-12-11 10:45:00+00:00	035XX S LEAVITT ST	0870	THEFT	POCKET-PICKING	CTA TRAIN	False	False	..
4	11366975	JB332149	2018-07-02 13:30:00+00:00	063XX S CICERO AVE	0870	THEFT	POCKET-PICKING	CTA BUS	False	False	..

5 rows × 22 columns

📝Copy the query you wrote earlier to retrieve the number of arrests for each primary type in the following cell.

⚠️This is important!

In [85]:
```
%%bigquery df

-- Copy your SQL query below this line.

SELECT primary_type, arrest, count(*) as counts FROM `bigquery-public-data.chicago_crime.crime`
where primary_type NOT IN ('OTHER OFFENSE') and primary_type NOT LIKE 'NON%' and year <= 2018
GROUP BY primary_type, arrest
ORDER BY primary_type, arrest ;
```

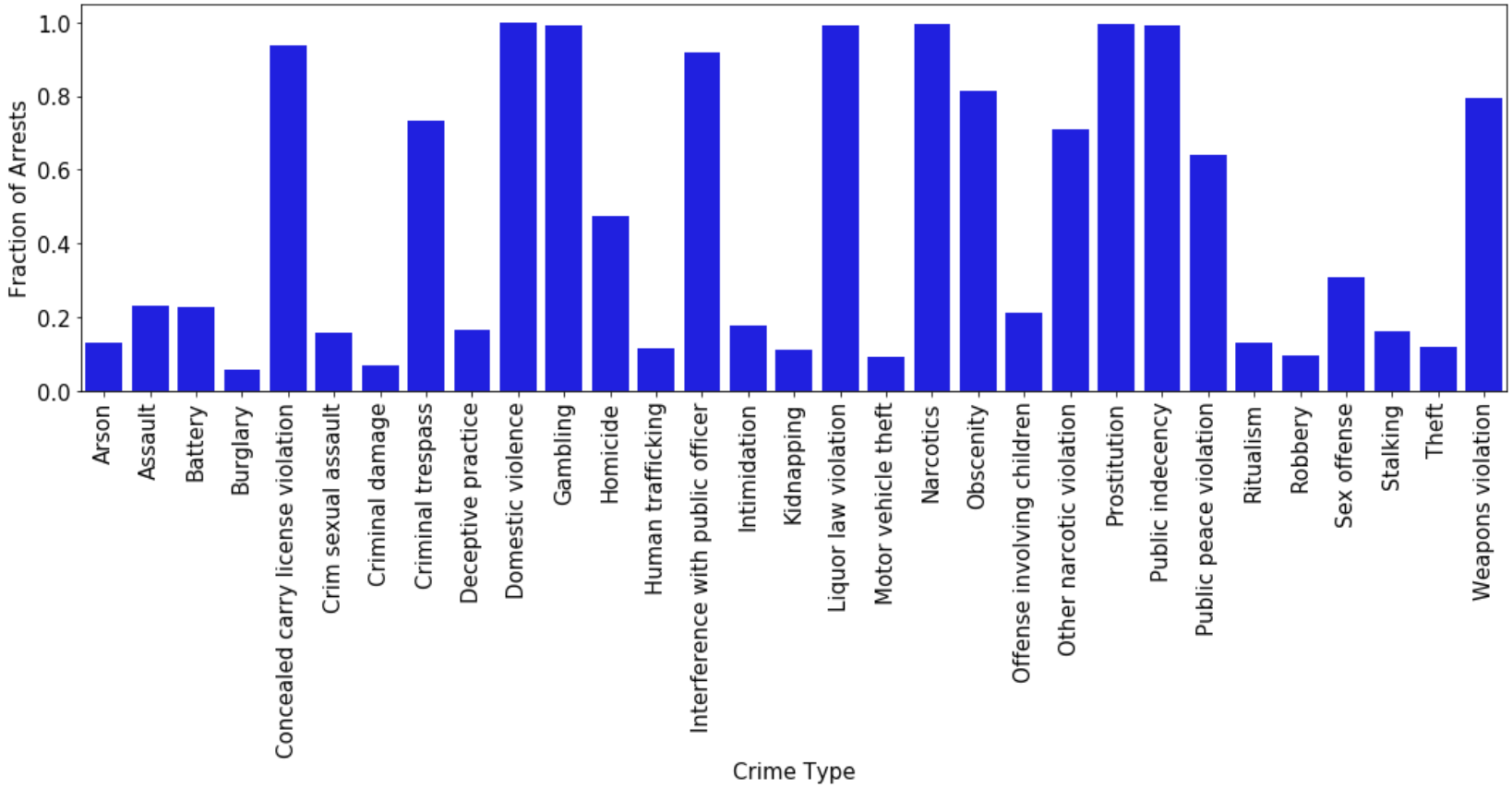
In [86]:
```
# We can now view a sample of our results
df.head()
```

Out[86]:

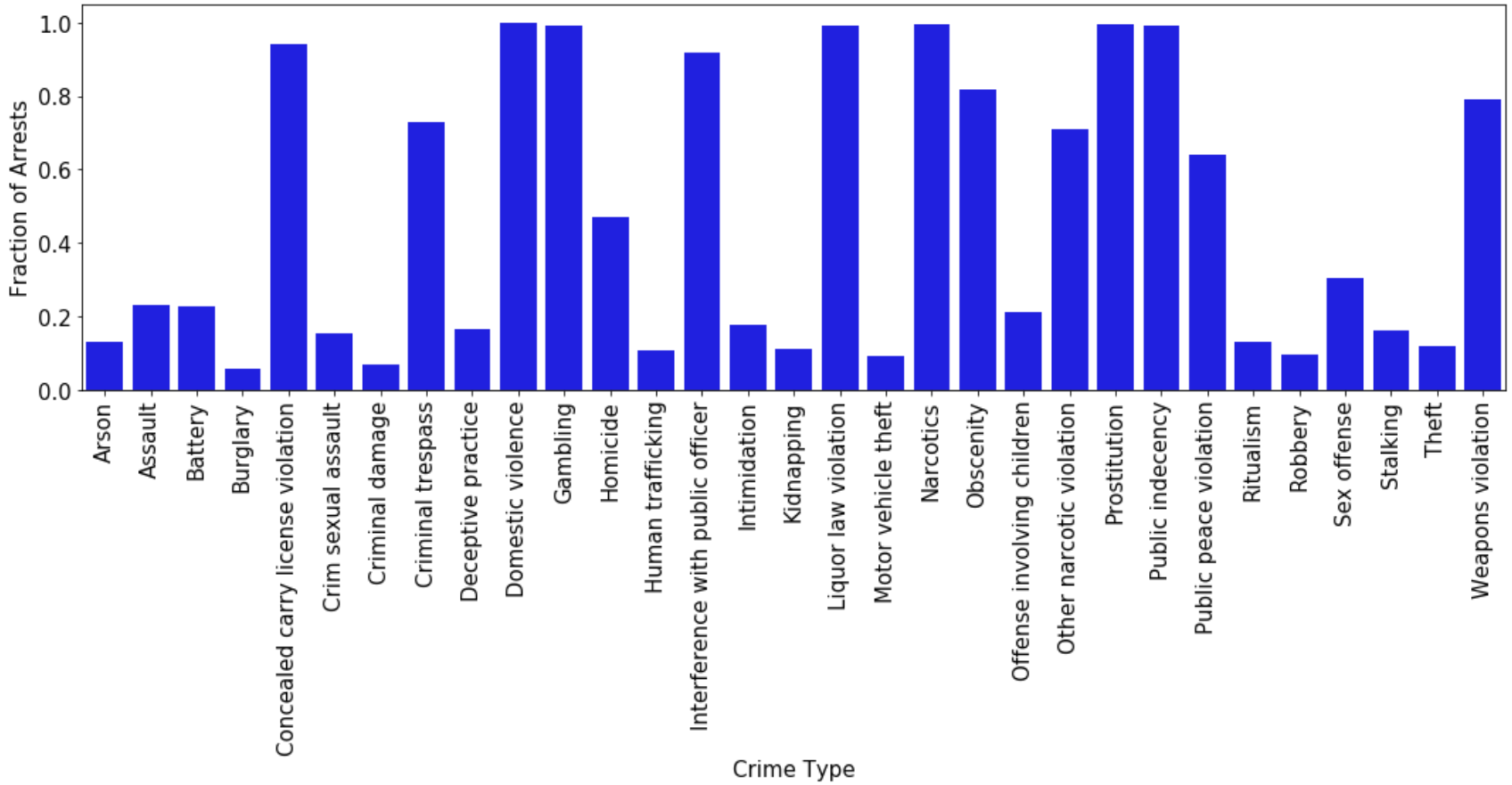
	primary_type	arrest	counts
0	ARSON	False	9743
1	ARSON	True	1463
2	ASSAULT	False	323815
3	ASSAULT	True	97235
4	BATTERY	False	956206

In [87]:

```
# Using the same dataframe, we can create a plot showing the fraction of arrests for each crime category
fig = plt.figure(figsize=(18, 5))

# Normalizing results
norm_df = df.assign(percentage=df.counts.divide(df.groupby('primary_type').counts.transform('sum')))
norm_df = norm_df[norm_df['arrest'] == True]
norm_df['primary_type'] = norm_df['primary_type'].apply(lambda x: x.capitalize())

# Standard sns barplot
ax = sns.barplot(x="primary_type", y="percentage", data=norm_df, color="b")

# Just putting on some makeup
plt.xticks(rotation=90, fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel("Crime Type", fontsize=15)
plt.ylabel("Fraction of Arrests", fontsize=15)
plt.show()
```



Your graph should look like the graph below:



In [88]:
```
# Now we can save this file in our submission subdirectory
out_filepath = os.path.join("sub", "crime_arrests.csv")
df.to_csv(out_filepath, index=False)
```

💾Check your `sub` (submission) subdirectory to verify that you have a file named `crime_arrests.csv`.

Q2 (20%)

📝For each block on `HALSTED ST` (North and South) find the most frequent crime type. Your final results (in the CSV) should have three columns `block`, `primary_type`, `count`. Where `block` is the block name, `primary_type` is the most frequent crime on that block and `count` is simply the number of records associated with that crime. Exclude "OTHER OFFENSE" and all non-criminal types, only

consider records until the end of year 2018 and sort results in ascending order by `block`.

It would be simpler to first extract relevant counts using SQL and then find the most frequent one for each block using `pandas` but some SQL experts might be able to do it in a single SQL query.

In [89]:
```
%%bigquery df

#standardSQL
-- Write your query below this line

SELECT block, primary_type, COUNT(*) as count FROM `bigquery-public-data.chicago_crime.crime`
WHERE block like '%HALSTED ST%' and primary_type NOT IN ('OTHER OFFENSE', 'NON - CRIMINAL') and primary_ty
GROUP BY block, primary_type
ORDER BY block, count DESC
```

In [91]:
```
df = df.sort_values(by='block')
```

In [92]:
```
# Write python code here for post-processing with pandas
df = df[df['count'] == df.groupby(['block'])['count'].transform(max)]
df = df.groupby(["block"]).apply(lambda x: x.sort_values(["primary_type"], ascending = True)).reset_index(
```

In [93]:
```
# Let us see a subset of results
df
```

Out[93]:

	block	primary_type	count
0	0000X N HALSTED ST	THEFT	878
1	0000X S HALSTED ST	THEFT	479
2	001XX N HALSTED ST	THEFT	73
3	001XX S HALSTED ST	THEFT	485
4	002XX N HALSTED ST	THEFT	66
5	002XX S HALSTED ST	BATTERY	164
6	003XX N HALSTED ST	THEFT	296
7	003XX S HALSTED ST	THEFT	97
8	004XX N HALSTED ST	THEFT	186
9	004XX S HALSTED ST	THEFT	78
10	005XX N HALSTED ST	THEFT	199
11	005XX S HALSTED ST	BATTERY	8
12	006XX N HALSTED ST	THEFT	34
13	006XX S HALSTED ST	THEFT	20
14	007XX N HALSTED ST	THEFT	61
15	007XX S HALSTED ST	THEFT	89
16	008XX N HALSTED ST	THEFT	37
17	008XX S HALSTED ST	THEFT	23
18	009XX N HALSTED ST	THEFT	48
19	009XX S HALSTED ST	THEFT	12
20	010XX N HALSTED ST	THEFT	54
21	010XX S HALSTED ST	THEFT	6
22	011XX N HALSTED ST	THEFT	30

	block	primary_type	count
23	011XX S HALSTED ST	THEFT	13
24	012XX N HALSTED ST	THEFT	125
25	012XX S HALSTED ST	THEFT	83
26	013XX N HALSTED ST	NARCOTICS	88
27	013XX S HALSTED ST	THEFT	98
28	014XX N HALSTED ST	THEFT	237
29	014XX S HALSTED ST	THEFT	80
...	...	...	...
143	101XX S HALSTED ST	THEFT	53
144	102XX S HALSTED ST	ROBBERY	55
145	103XX S HALSTED ST	THEFT	180
146	104XX S HALSTED ST	THEFT	46
147	105XX S HALSTED ST	NARCOTICS	64
148	106XX S HALSTED ST	THEFT	244
149	107XX S HALSTED ST	THEFT	162
150	108XX S HALSTED ST	THEFT	86
151	109XX S HALSTED ST	THEFT	148
152	110XX S HALSTED ST	THEFT	82
153	111XX S HALSTED ST	THEFT	230
154	112XX S HALSTED ST	THEFT	48
155	113XX S HALSTED ST	BATTERY	54
156	114XX S HALSTED ST	THEFT	634
157	115XX S HALSTED ST	THEFT	230
158	116XX S HALSTED ST	THEFT	132
159	117XX S HALSTED ST	DECEPTIVE PRACTICE	74
160	118XX S HALSTED ST	NARCOTICS	55
161	119XX S HALSTED ST	THEFT	110

	block	primary_type	count
162	120XX S HALSTED ST	BATTERY	71
163	121XX S HALSTED ST	BATTERY	52
164	122XX S HALSTED ST	NARCOTICS	139
165	123XX S HALSTED ST	NARCOTICS	278
166	123XX S HALSTED STREET	HOMICIDE	1
167	124XX S HALSTED ST	BATTERY	17
168	125XX S HALSTED ST	THEFT	19
169	126XX S HALSTED ST	THEFT	200
170	127XX S HALSTED ST	BATTERY	75
171	128XX S HALSTED ST	THEFT	53
172	129XX S HALSTED ST	CRIMINAL DAMAGE	8

172 rows × 3 columns

In [94]:
```
# Now we can save this file in our submission subdirectory
out_filepath = os.path.join("sub", "freq_crime.csv")
df.to_csv(out_filepath, index=False)
```

💾Check your `sub` (submission) subdirectory to verify that you have a file named `freq_crime.csv` following the format specified above.

Part 2: Google Data Studio

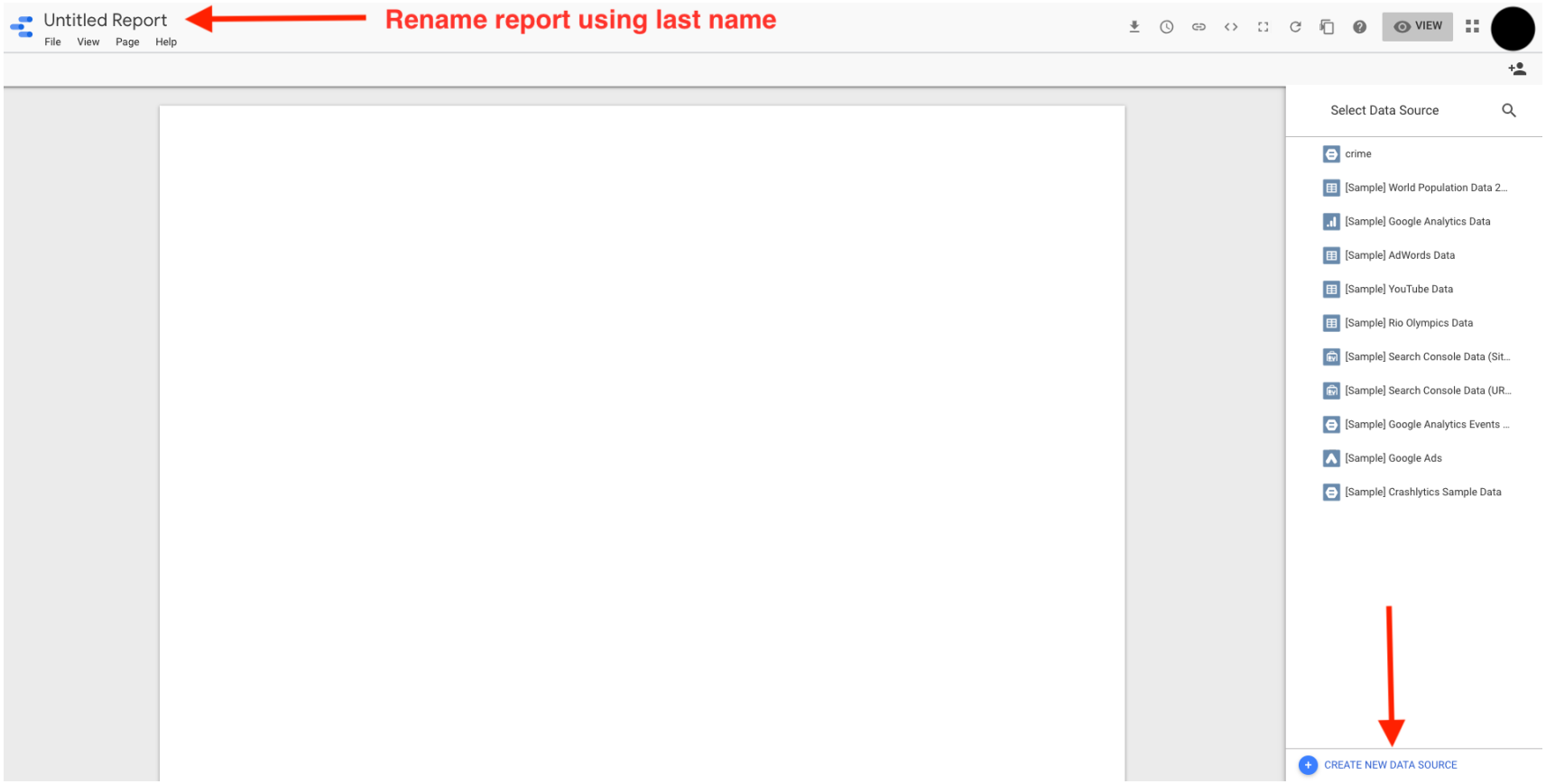
In this part, we will explore a tool called Google Data Studio (https://datastudio.google.com) which allows us to create visualizations using large scale datasets. This tool can retrieve data from many sources (including BigQuery tables!) and provide a simple interface to create dynamic and sophisticated visualizations. Let's get started!

Q3 (20%)

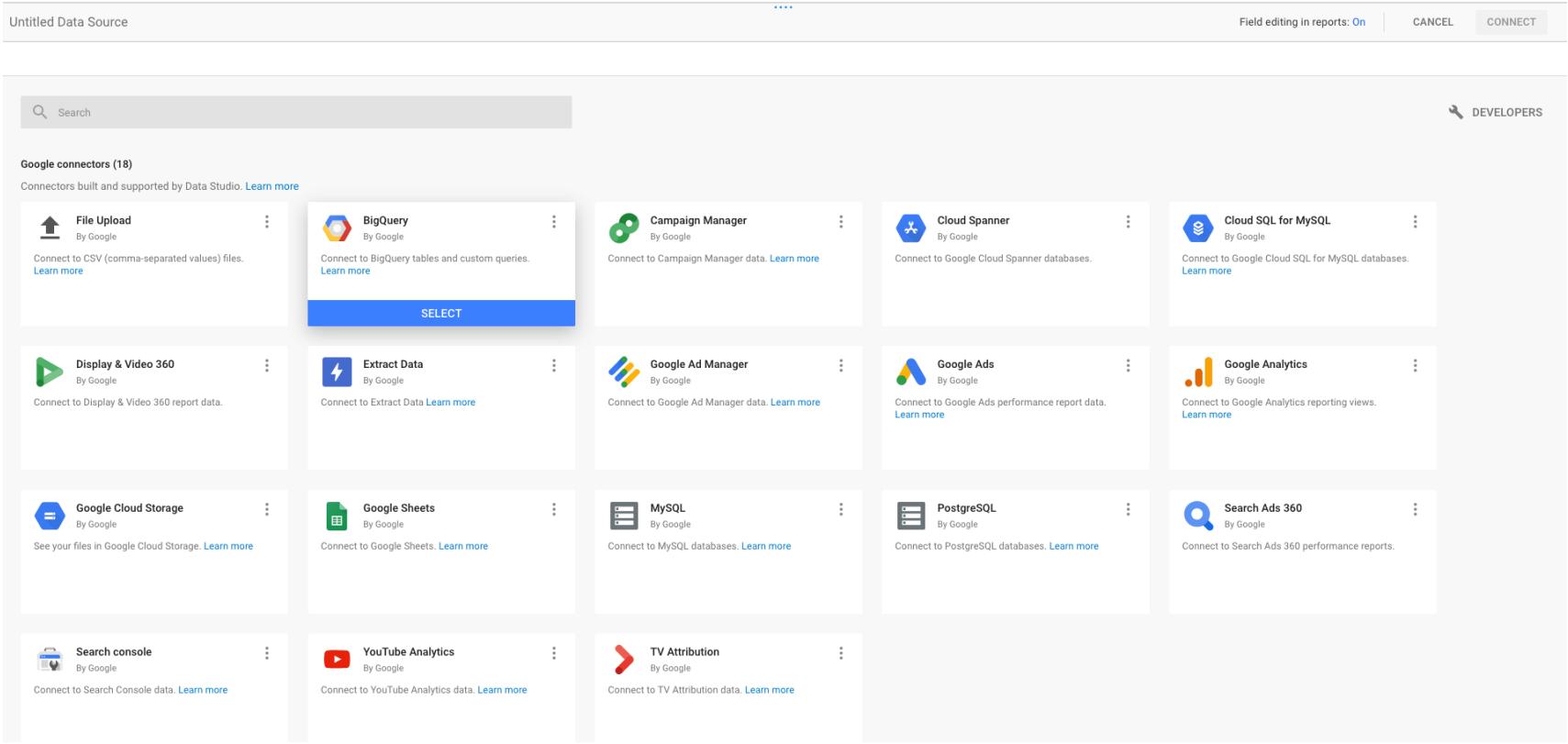
Credit will be awarded based on completion of this tutorial. We will try to replicate the graph we created earlier in **Q1**.

First, we need to create a new (blank) report, use your last name as the name of report with a suffix `-q3` , if your are working in group then the report name should be of the format `lastname1-lastname2-q3` .

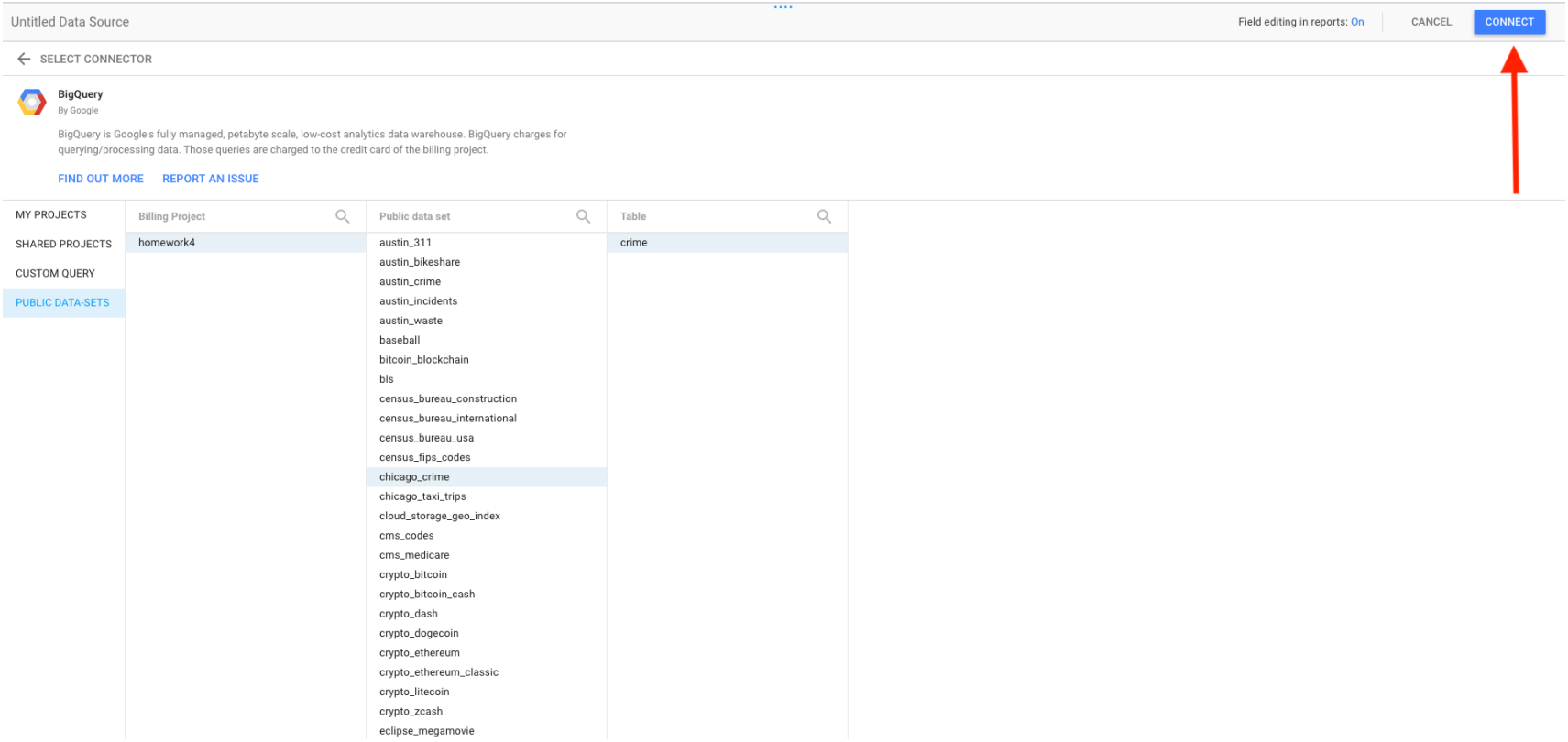
You will be directed to the following screen after report is created. Click *Create New Data Source* button in the sidebar.



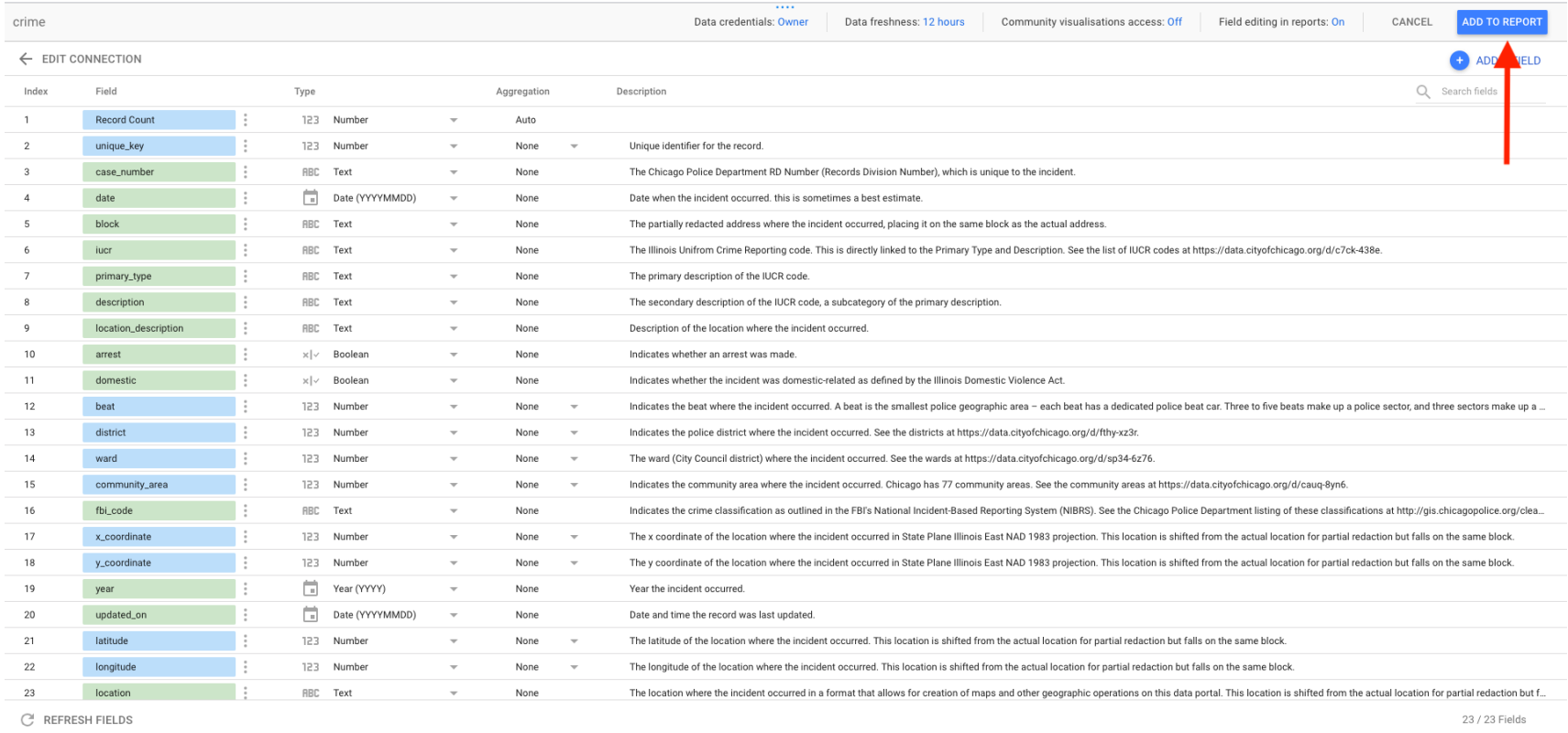
Select BigQuery as the data source



Select the crime table from chicago_crime public dataset and click *Connect* button.

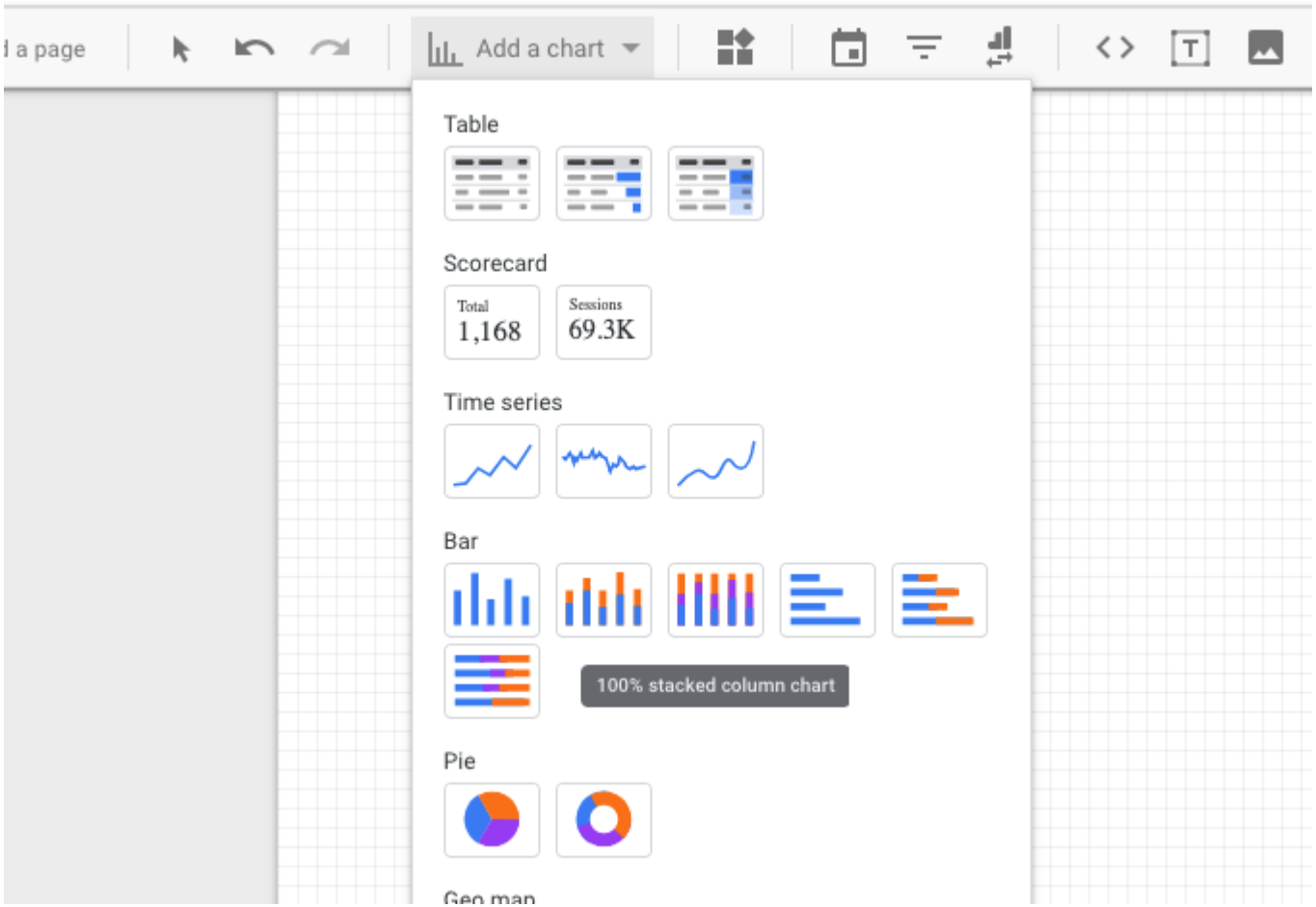


This will take you to add data interface, we have the option to modify fields here but it is not recommended to do it at this stage. Press *ADD TO REPORT* button to continue.

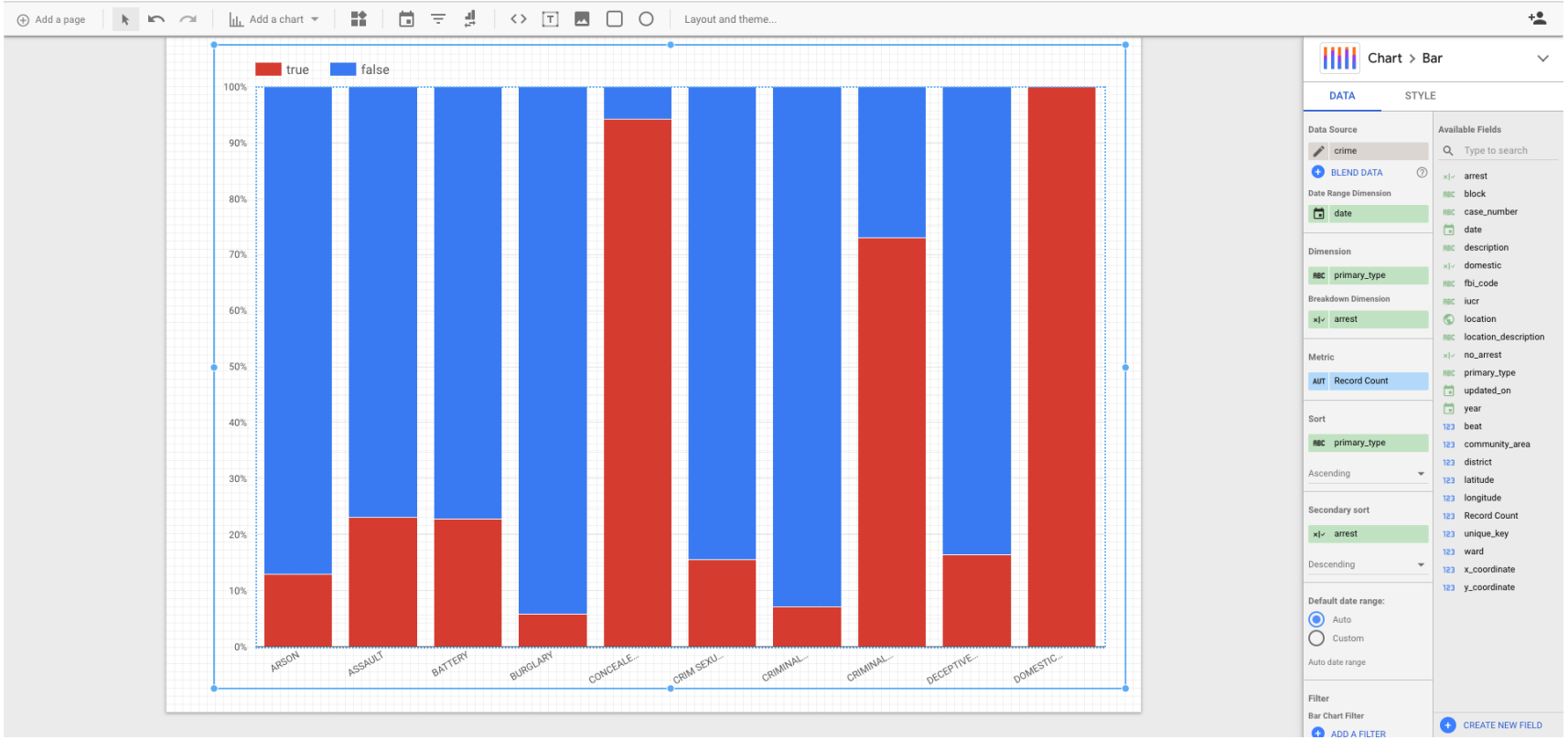


Index	Field	Type	Aggregation	Description
1	Record Count	Number	Auto	
2	unique_key	Number	None	Unique identifier for the record.
3	case_number	Text	None	The Chicago Police Department RD Number (Records Division Number), which is unique to the incident.
4	date	Date (YYYYMMDD)	None	Date when the incident occurred. this is sometimes a best estimate.
5	block	Text	None	The partially redacted address where the incident occurred, placing it on the same block as the actual address.
6	iucr	Text	None	The Illinois Unifrom Crime Reporting code. This is directly linked to the Primary Type and Description. See the list of IUCR codes at https://data.cityofchicago.org/d/c7ck-438e.
7	primary_type	Text	None	The primary description of the IUCR code.
8	description	Text	None	The secondary description of the IUCR code, a subcategory of the primary description.
9	location_description	Text	None	Description of the location where the incident occurred.
10	arrest	Boolean	None	Indicates whether an arrest was made.
11	domestic	Boolean	None	Indicates whether the incident was domestic-related as defined by the Illinois Domestic Violence Act.
12	beat	Number	None	Indicates the beat where the incident occurred. A beat is the smallest police geographic area – each beat has a dedicated police beat car. Three to five beats make up a police sector, and three sectors make up a ...
13	district	Number	None	Indicates the police district where the incident occurred. See the districts at https://data.cityofchicago.org/d/fthy-xz3r.
14	ward	Number	None	The ward (City Council district) where the incident occurred. See the wards at https://data.cityofchicago.org/d/sp34-6z76.
15	community_area	Number	None	Indicates the community area where the incident occurred. Chicago has 77 community areas. See the community areas at https://data.cityofchicago.org/d/cauq-8yn6.
16	fbi_code	Text	None	Indicates the crime classification as outlined in the FBI's National Incident-Based Reporting System (NIBRS). See the Chicago Police Department listing of these classifications at http://gis.chicagopolice.org/clea...
17	x_coordinate	Number	None	The x coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection. This location is shifted from the actual location for partial redaction but falls on the same block.
18	y_coordinate	Number	None	The y coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection. This location is shifted from the actual location for partial redaction but falls on the same block.
19	year	Year (YYYY)	None	Year the incident occurred.
20	updated_on	Date (YYYYMMDD)	None	Date and time the record was last updated.
21	latitude	Number	None	The latitude of the location where the incident occurred. This location is shifted from the actual location for partial redaction but falls on the same block.
22	longitude	Number	None	The longitude of the location where the incident occurred. This location is shifted from the actual location for partial redaction but falls on the same block.
23	location	Text	None	The location where the incident occurred in a format that allows for creation of maps and other geographic operations on this data portal. This location is shifted from the actual location for partial redaction but f...

Now that we have a data source added to the report, we can use it to create charts. Click *Add a chart* dropdown menu and select *100% stacked column chart* from it as shown below.

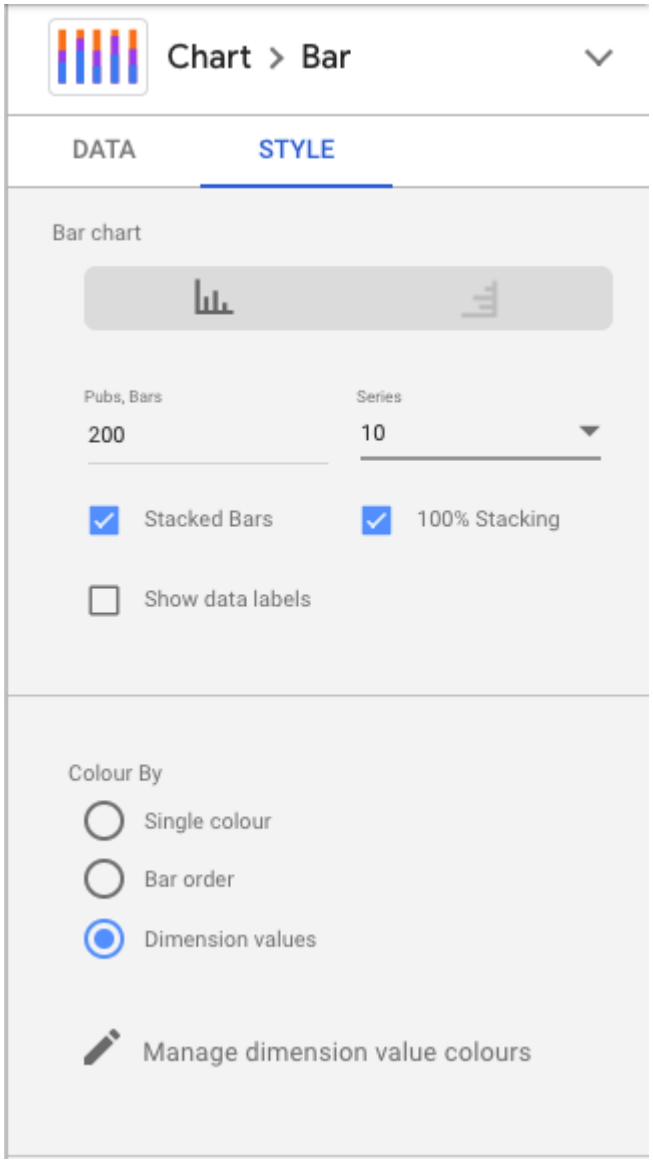


Once, the chart is added, you can click on the chart to view Chart configurations in the right sidebar. Under *DATA*, we can configure fields and under *STYLE*, we can configure appearances of the chart. Setup the fields as shown in the figure below:

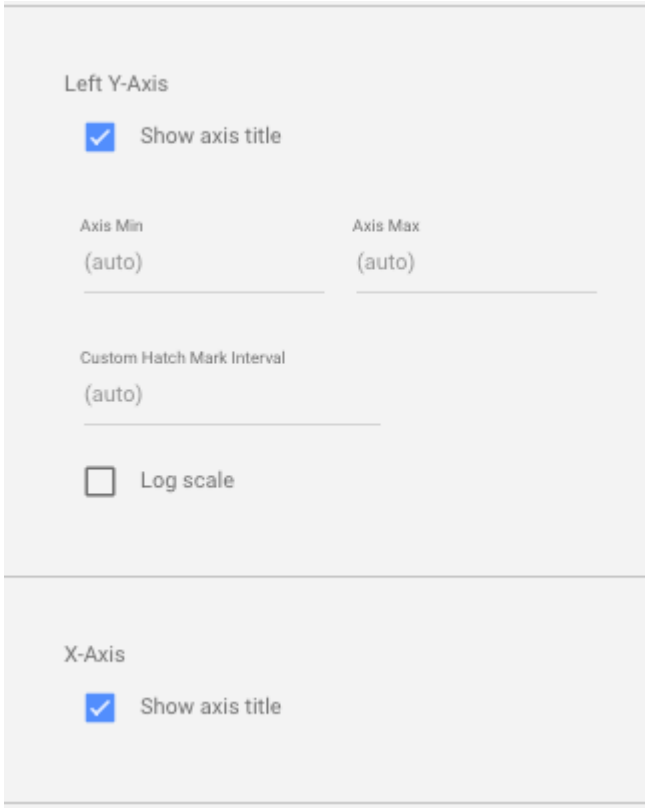


Here, *Dimension* is the variable on the x-axis, *Breakdown Dimension* is the same as *hue* in seaborn. *Metric* specifies how you are going to aggregate records and *Sort* is just sorting on the x-axis. Something seems missing in this graph (can you spot it?)

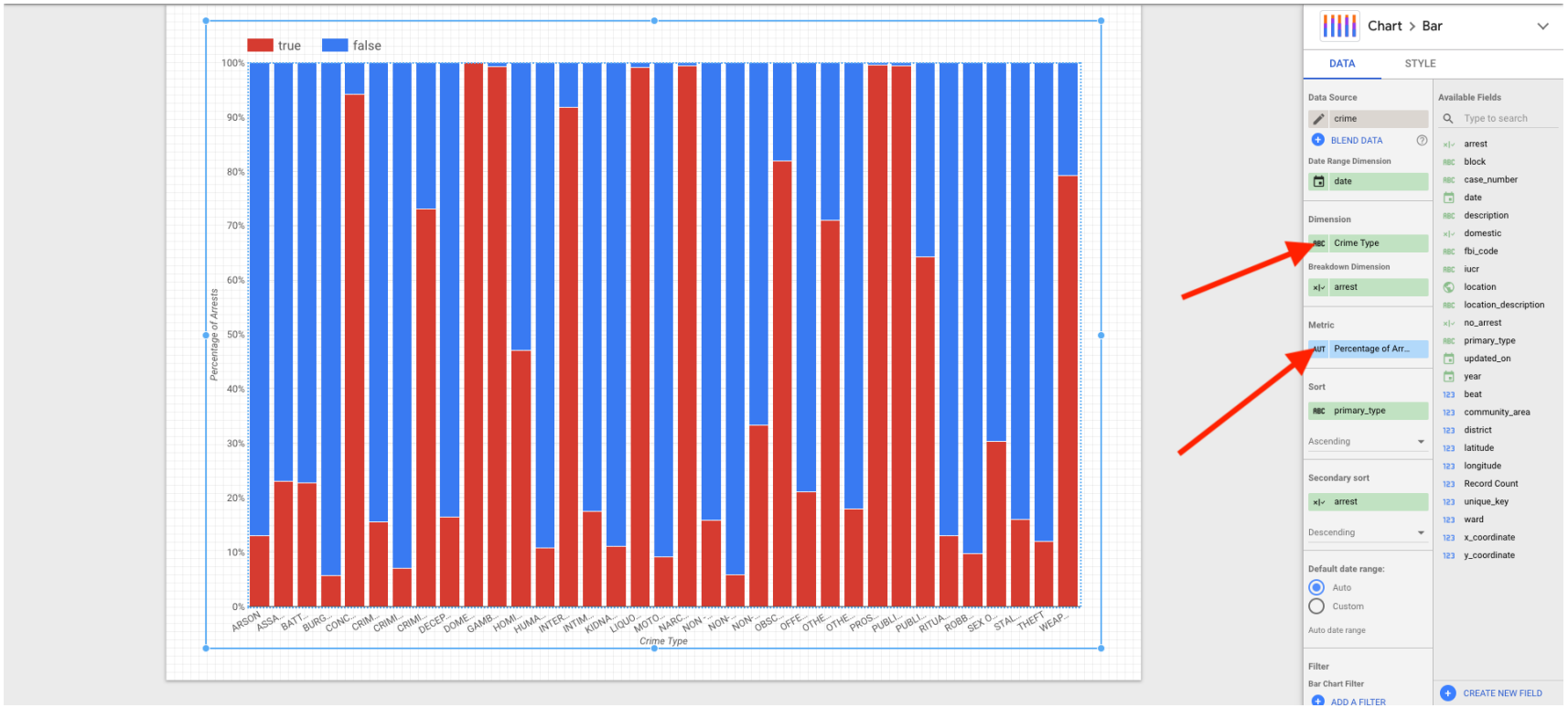
The graph we created earlier had a lot more bars than this one and this is because, by default, it only shows a subset of data. Let us fix that under the *STYLE* menu by setting *Pubs, Bars* to a large number (200).



That fixed the bars, but we also want to label our axes. We can also do that in the *STYLE* menu by checking these boxes:

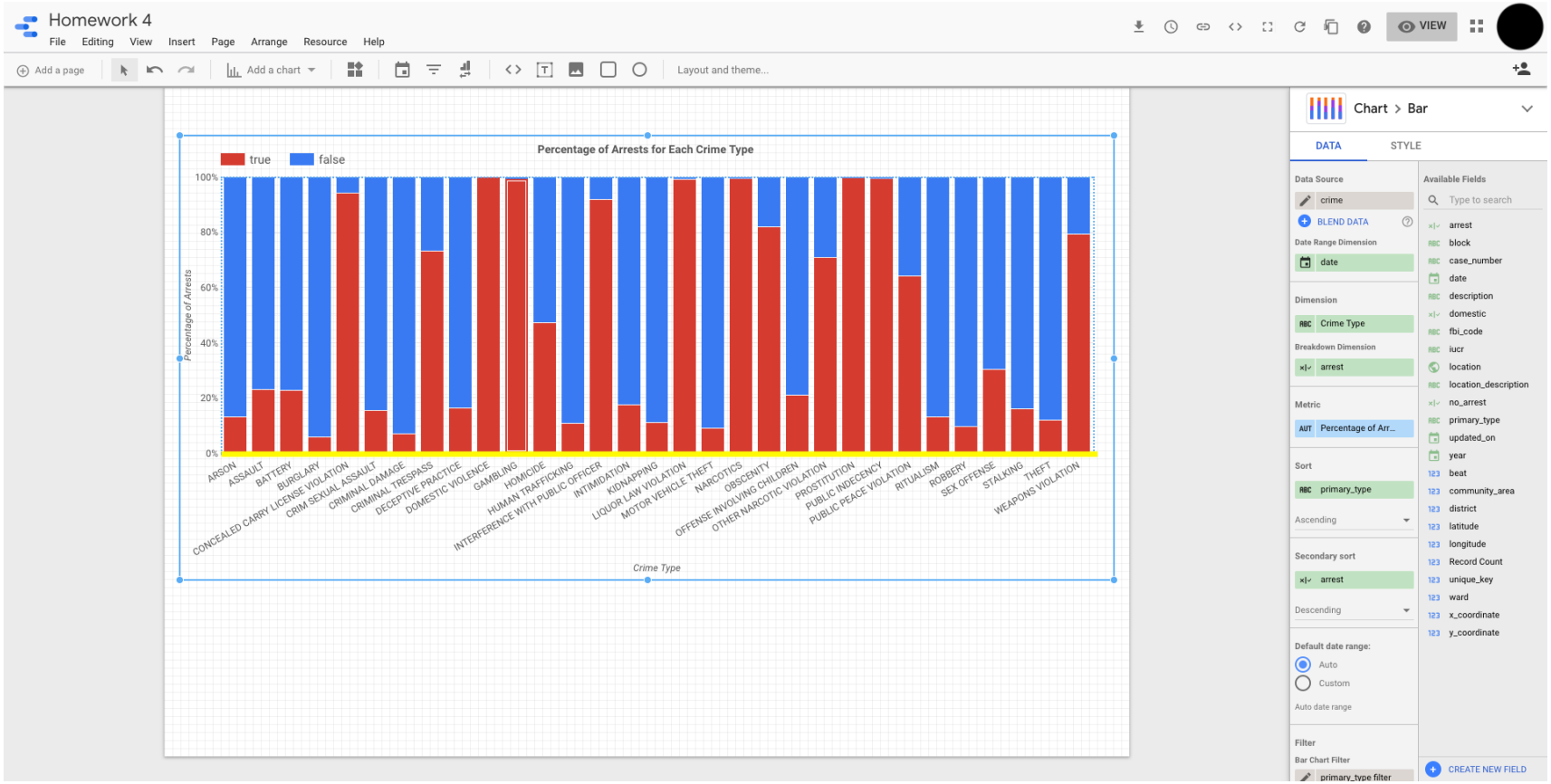


Now we can see all crime types and the axis labels but the labels are not very readable, we can change them by clicking the relevant areas highlighted by red arrows in the following figure.



Another problem in this figure is that, we can still see non-criminal crime types and "OTHER OFFENSE" on the x-axis which is just adding noise to the chart without providing any meaningful information so we need to filter these types out. Click on *ADD A FILTER* under *DATA* in the right sidebar and setup the appropriate filters as shown below:

Our graph is now almost ready, we can add title using *Text* tool from the toolbar and we can drag inner boundary (signified by yellow line in the figure below) to make our x-axis labels more visile. We can drag the external boundaries to stretch the graph and voila, with just a few clicks, we managed to create this graph from a large dataset.

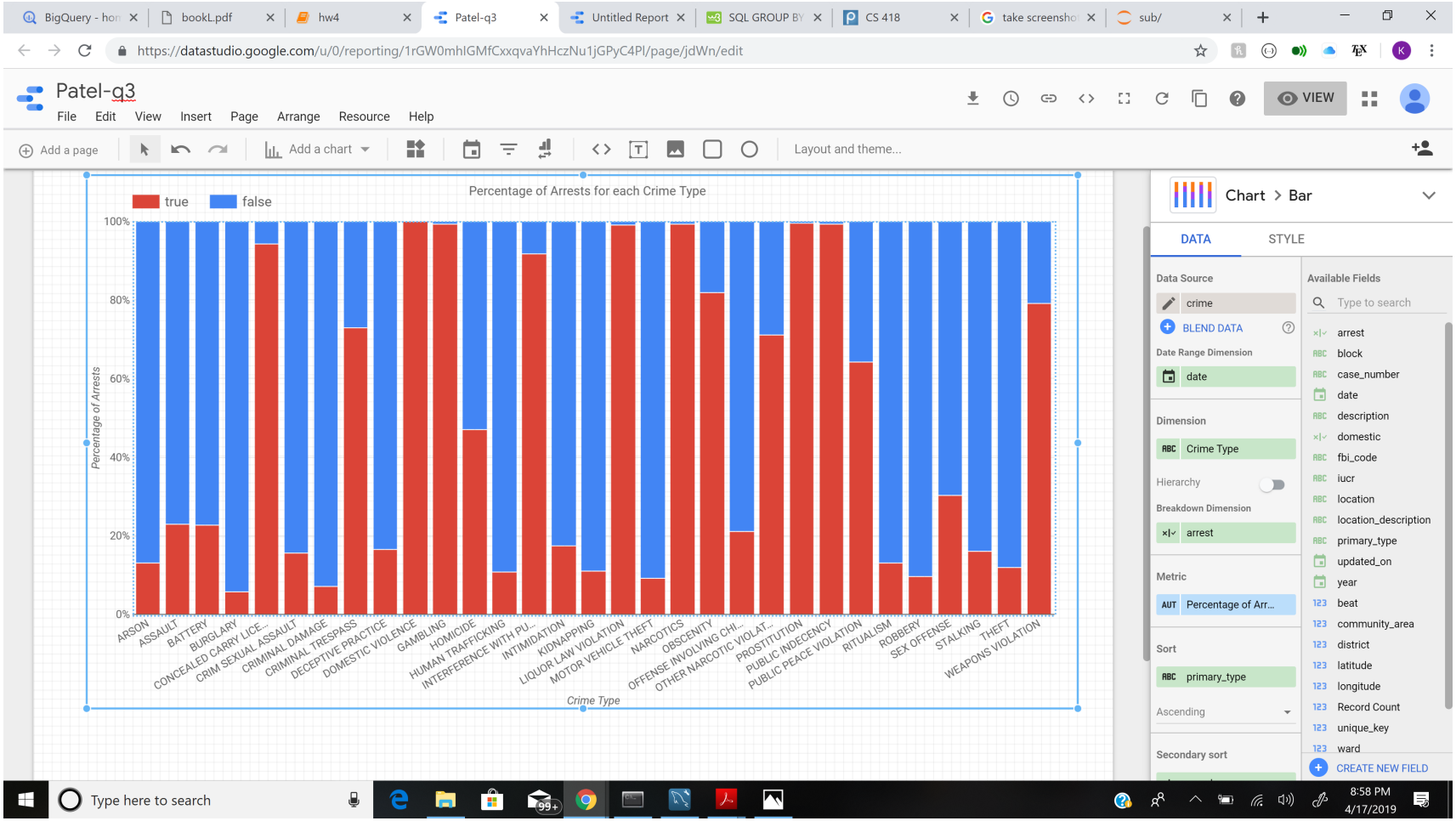


Submission

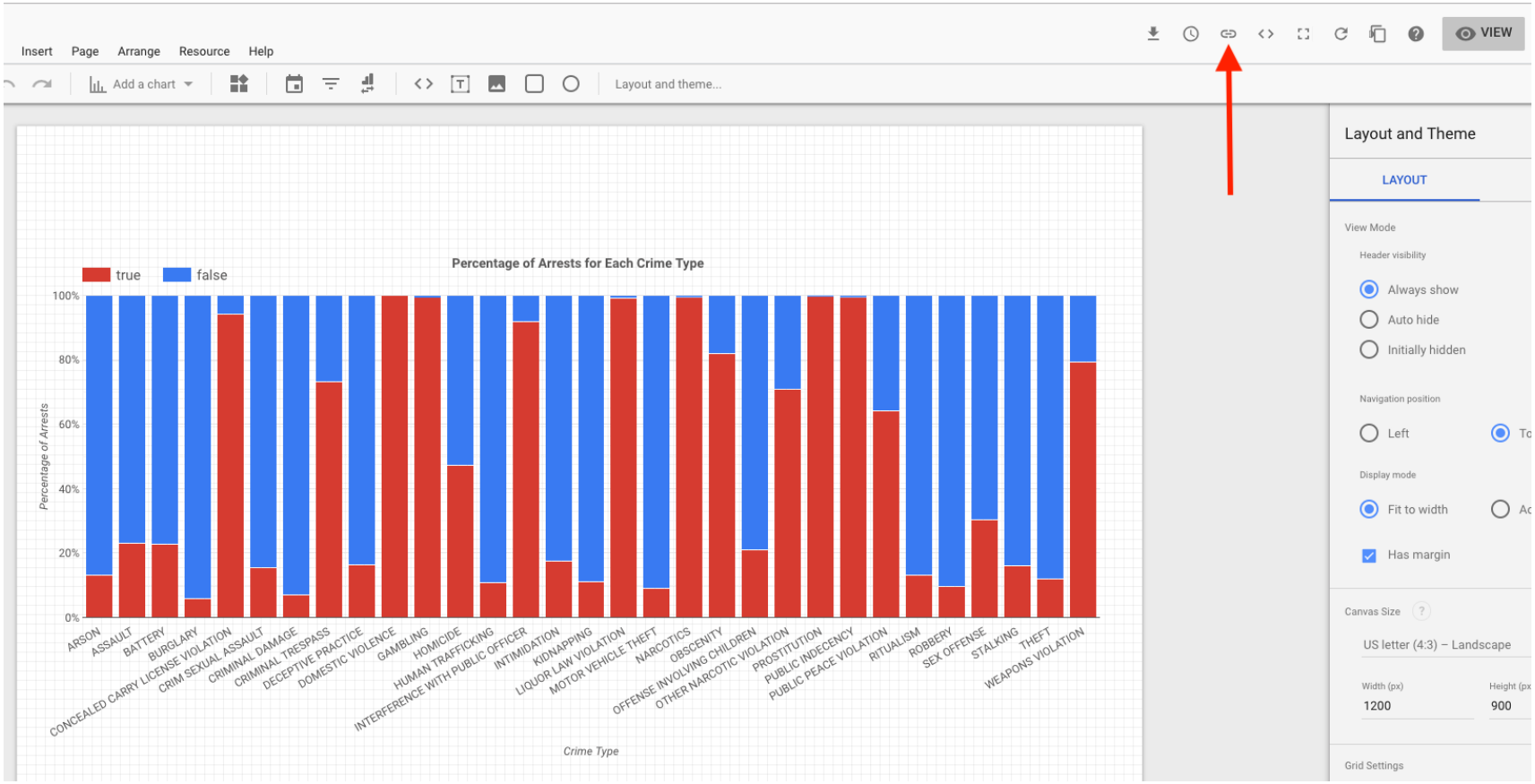
💾Take a screenshot (full screen, PNG format), rename it as `q3-tutorial.png` and store it in `sub` subdirectory.

⚠️Make sure that the report name and graph is visible in the screenshot, no credit will be awarded otherwise.

📝Include screenshot in the notebook by editing the following markdown (if needed). It shows as a broken image by default if screenshot is unavailable.

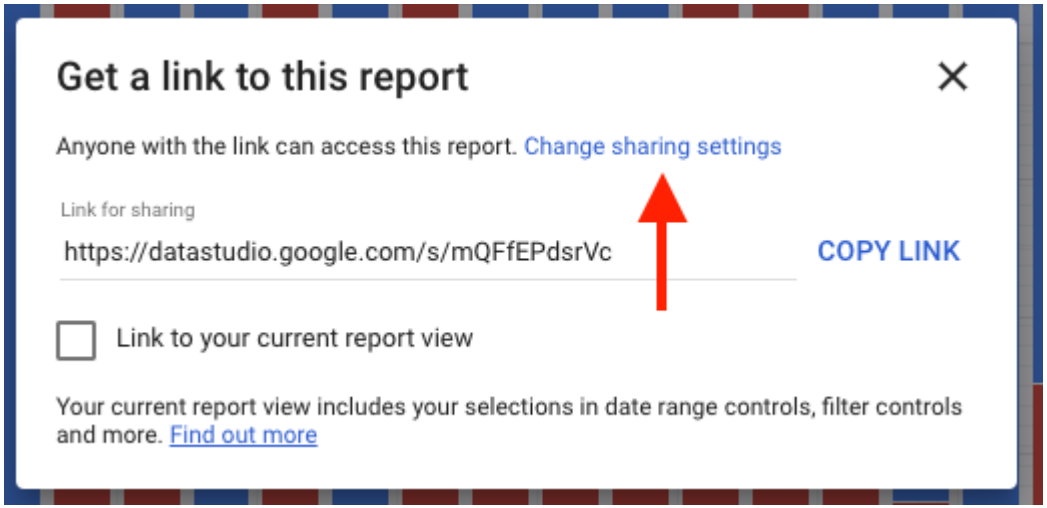


The report is now ready to be shared. Click the *Share* icon as shown in the following figure to initiate the process:

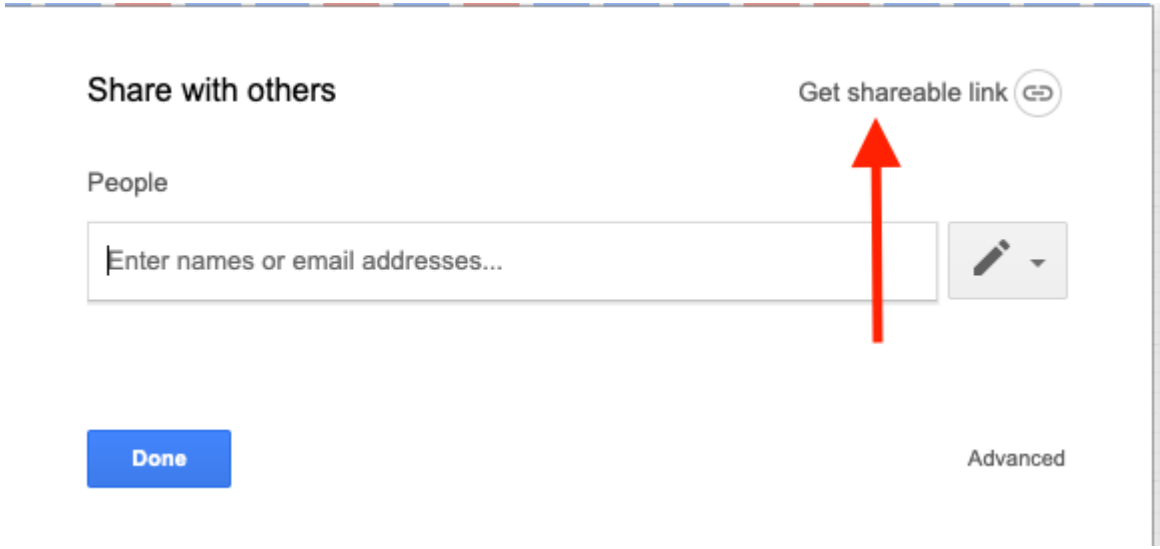


Click on *Change sharing settings* link as shown in the following figure

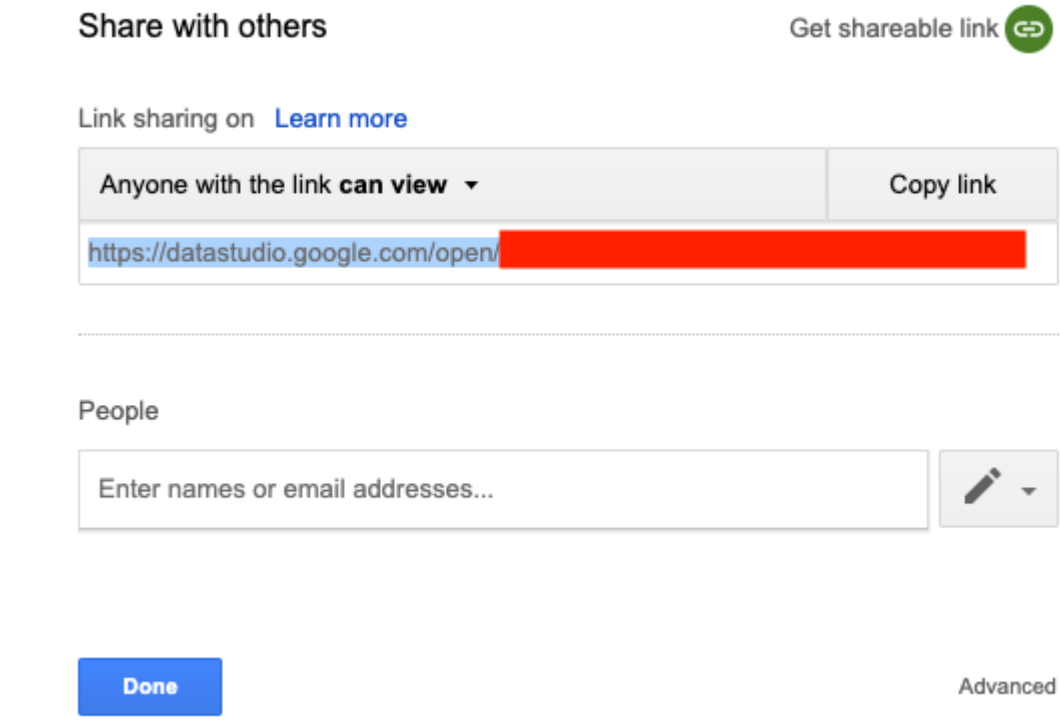
⚠Do not use/copy the link shown in the following figure.



Click on *Get shareable link*



Sharing should now be enabled using the link shown in the following screenshot.



📝Copy the sharable link to report from the previous step into the box below. Anyone should be able to view this report without having to explicity request access.

⚠️No credit will be awarded without this link. Test this link in incognito mode of the browser.

⚠️DO NOT change/delete the report after submission.

https://datastudio.google.com/open/1rGW0mhIGMfCxxqvaYhHczNu1jGPyC4Pl
(https://datastudio.google.com/open/1rGW0mhIGMfCxxqvaYhHczNu1jGPyC4Pl)

Q4 (20%)

This question is more open ended and similar to Q2.2-Q2.4 from *HW2*. It will be graded in the same way, based on the completeness of the report produced and the insights you gained from it. We will be using the publicly avaialable chicago crimes dataset for this task and you are only required to produce a single graph.

Be sure to consider transformations, subsets, correlations, reference markers, and lines/curves-of-best-fit (as covered in Chapter 6 of PTDS) to reveal the relationship that you are wanting to learn more about. Also be sure to make plots that are appropriate for the variable types. For completeness, be explicit about any assumptions you make in your analysis. An exemplary plot will have:

- A title
- Labelled and appropriately scaled axes
- A legend, if applicable
- A carefully selected color scheme
- A main point, accentuated through design choices

First, same as **Q3**, you need to create a new (blank) report, use your last name as the name of report with a suffix `-q4` , if your are working in group then the report name should be of the format `lastname1-lastname2-q4` .

⚠️No credit will be awarded if the report name is wrong.

⚠️Do not use the same report that you used in **Q3**

Submission

📝Write your main takeaway/hypothesis 5-15 words in the following cell:

The number of felonies have decreased over a span of last one year for all different types of Crime.

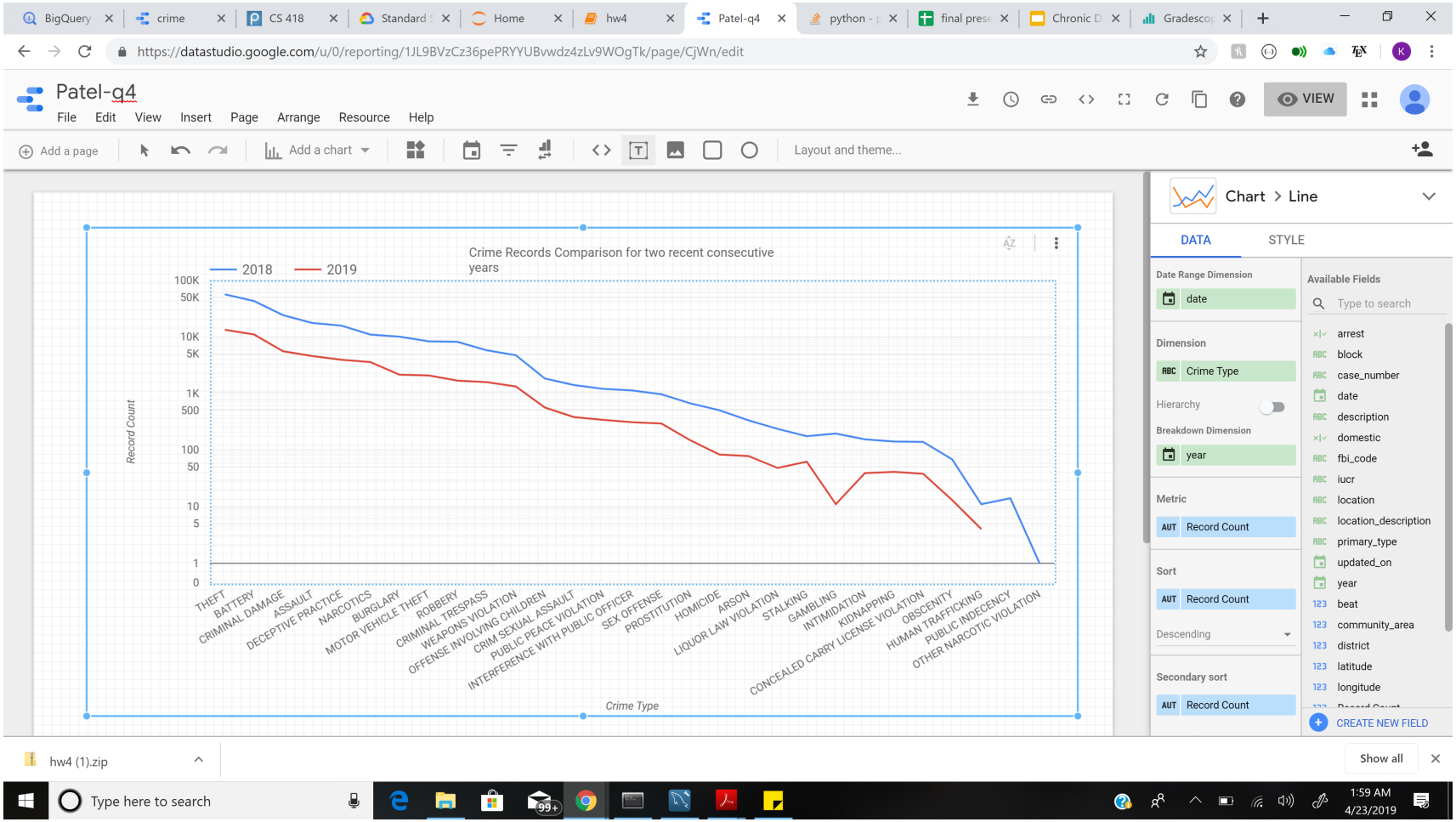
📝Write a description 100-150 words following cell explaining your assumptions and what you have found.

Line plots for the years 2018 and 2019 are almost parallel to each other, where the number of felonies for each crime type is a lot more in the year 2018 as compared to 2019. Hence, we can say that crime records for all different types of crimes have decreased by a same proportion from 2019 to 2018, except for Gambling as it has decreased substantially.

💾Take a screenshot (full screen, PNG format), rename it as `q4-report.png` and store it in `sub` subdirectory.

⚠️Make sure that the report name and graph is visible in the screenshot, no credit will be awarded otherwise.

Include screenshot in the notebook by editing the following markdown (if needed). It shows as a broken image by default if screenshot is unavailable.



Create a sharable link to this report using the process shown in **Q3** and add it in the next cell. Anyone should be able to view this report without having to explicity request access.

No credit will be awarded without this link. Test this link in incognito mode of the browser.

⚠DO NOT change/delete the report after submission.

https://datastudio.google.com/open/1JL9BVzCz36pePRYYUBvwdz4zLv9WOgTk
(https://datastudio.google.com/open/1JL9BVzCz36pePRYYUBvwdz4zLv9WOgTk)

Part 3: Google Cloud ML Engine

In this part, we will use Google Cloud ML Engine (https://cloud.google.com/ml-engine/) to train and deploy a simple machine machine learning model. Training models from large datasets is a particularly tedious task, especially if it involves hyper-parameter tuning and training may take days to complete and require high computational powers and multiple CPUs/GPUs/TPUs. Google Cloud ML Engine addressess all these problems. Let's get started!

Q5 (20%)

Credit will be awarded based on completion of this tutorial. Same, as before, we will train our model using tweets dataset from *HW3* and make predictions on the test set, however we will do all that using Google Cloud ML Engine.

Twitter data is extracted using this (https://dev.twitter.com/overview/api) api. The data contains tweets posted by the following six Twitter accounts: `realDonaldTrump, mike_pence, GOP, HillaryClinton, timkaine, TheDemocrats`

For every tweet, there are two pieces of information:

- `screen_name` : the Twitter handle of the user tweeting and
- `text` : the content of the tweet.

The tweets have been divided into two parts - train and test available to you in CSV files. For train, both the `screen_name` and `text` attributes were provided but for test, `screen_name` is hidden.

The overarching goal of the problem is to "predict" the political inclination (Republican/Democratic) of the Twitter user from one of his/her tweets. The ground truth (i.e., true class labels) is determined from the `screen_name` of the tweet as follows

- `realDonaldTrump, mike_pence, GOP` are Republicans
- `HillaryClinton, timkaine, TheDemocrats` are Democrats

Thus, this is a binary classification problem.

The code to create features from data and to create labels is provided below, you do not need to re-write it. Run the following cells to generate data in a format (https://cloud.google.com/ml-engine/docs/algorithms/preprocessing-data) that the ML Engine can understand.

⚠DO NOT change this code or use code form *HW3*

In [95]:
```
def create_labels(processed_tweets):
    return (~processed_tweets['screen_name'].isin(['realDonaldTrump', 'mike_pence', 'GOP'])).astype(int).v
```

In [96]:
```
def setup_traintest():
    # Setup training and testing paths
    train_path = os.path.join('data', 'tweets_train.csv')
    train_gcinput_path = os.path.join('data', 'train_gcinput.csv')

    test_path = os.path.join('data', 'tweets_test.csv')
    test_gcinput_path = os.path.join('data', 'test_gcinput.csv')

    # Setup vectorizer and PCA
    eng_stopwords = set(stopwords.words('english'))
    vec = TfidfVectorizer(stop_words=eng_stopwords, max_df=0.8, min_df=3, max_features=5000)
    pca = PCA(n_components=300)


    # Setup training data
    train_df = pd.read_csv(train_path)
    train_feat = vec.fit_transform(train_df['text'])
    train_feat = pca.fit_transform(train_feat.todense())
    train_lab = create_labels(train_df)
    df_feat = pd.DataFrame(train_feat)
    df_lab = pd.DataFrame(train_lab)
    df = pd.concat([df_lab, df_feat], axis=1)
    df.to_csv(train_gcinput_path, index=False, header=False)

    # # Setup testing data
    test_df = pd.read_csv(test_path)
    test_feat = vec.transform(test_df['text']).todense()
    test_feat = pca.transform(test_feat)
    df = pd.DataFrame(test_feat)
    df.to_csv(test_gcinput_path, index=False, header=False)
```

In [97]:
```
setup_traintest()
```

This code should produce two files `train_gcinput.csv` and `test_gcinput.csv` in the `data` subdirectory. `train_gcinput.csv` contains labels in first column and features in remaining columns for each record and `test_gcinput.csv` just contains the features for each record. There are 300 features for each record.

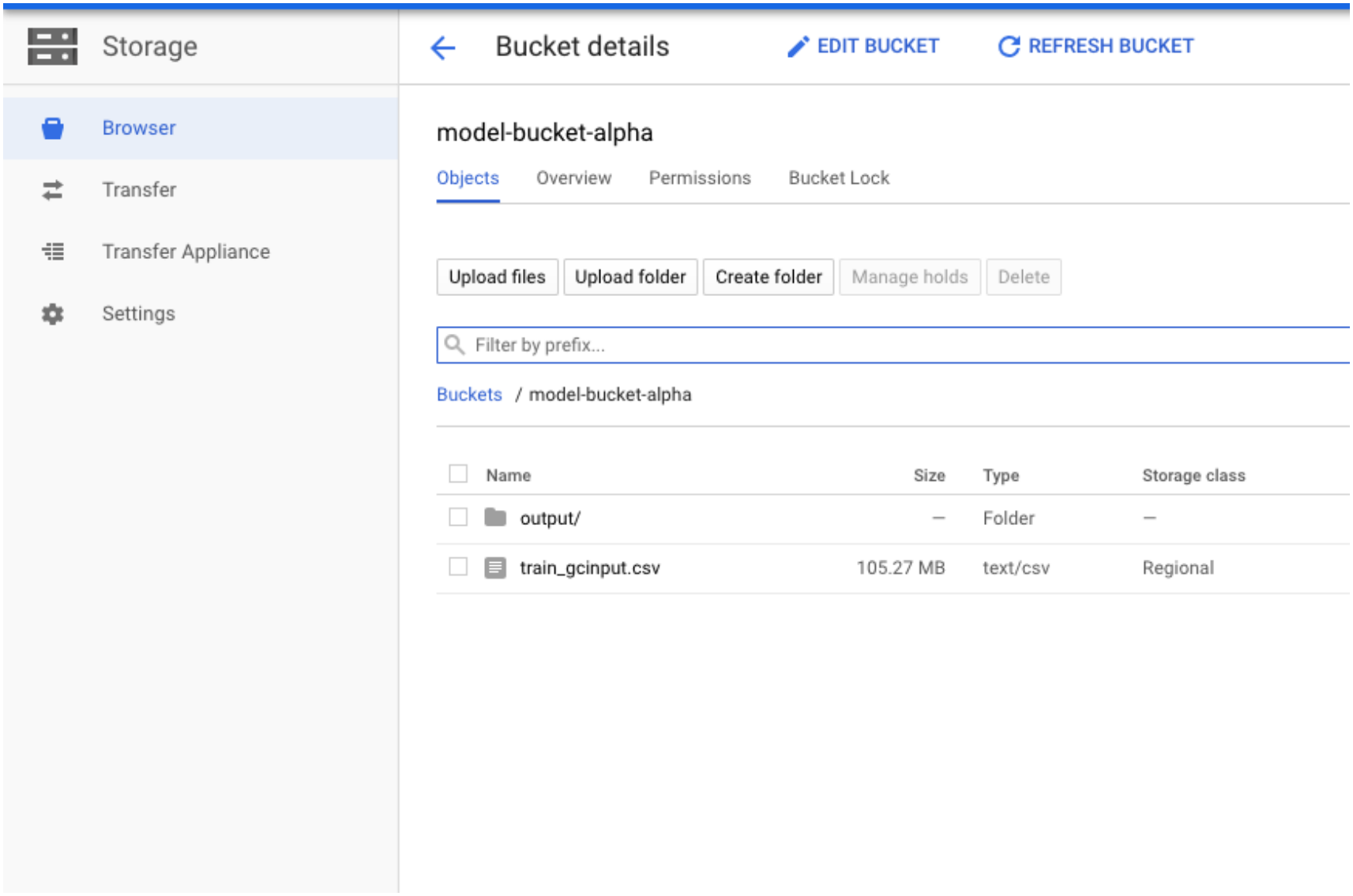
Side Note: PCA is a dimensionality reduction algorithm (https://medium.com/@mayur87545/dimensionality-reduction-1663f960293f) which makes our file size smaller and our computations simpler.

Uploading Data to Cloud

Before we can train our algorithm, we need to upload `train_gcinput.csv` to the cloud so that the ML engine can access it. But how do we do that?

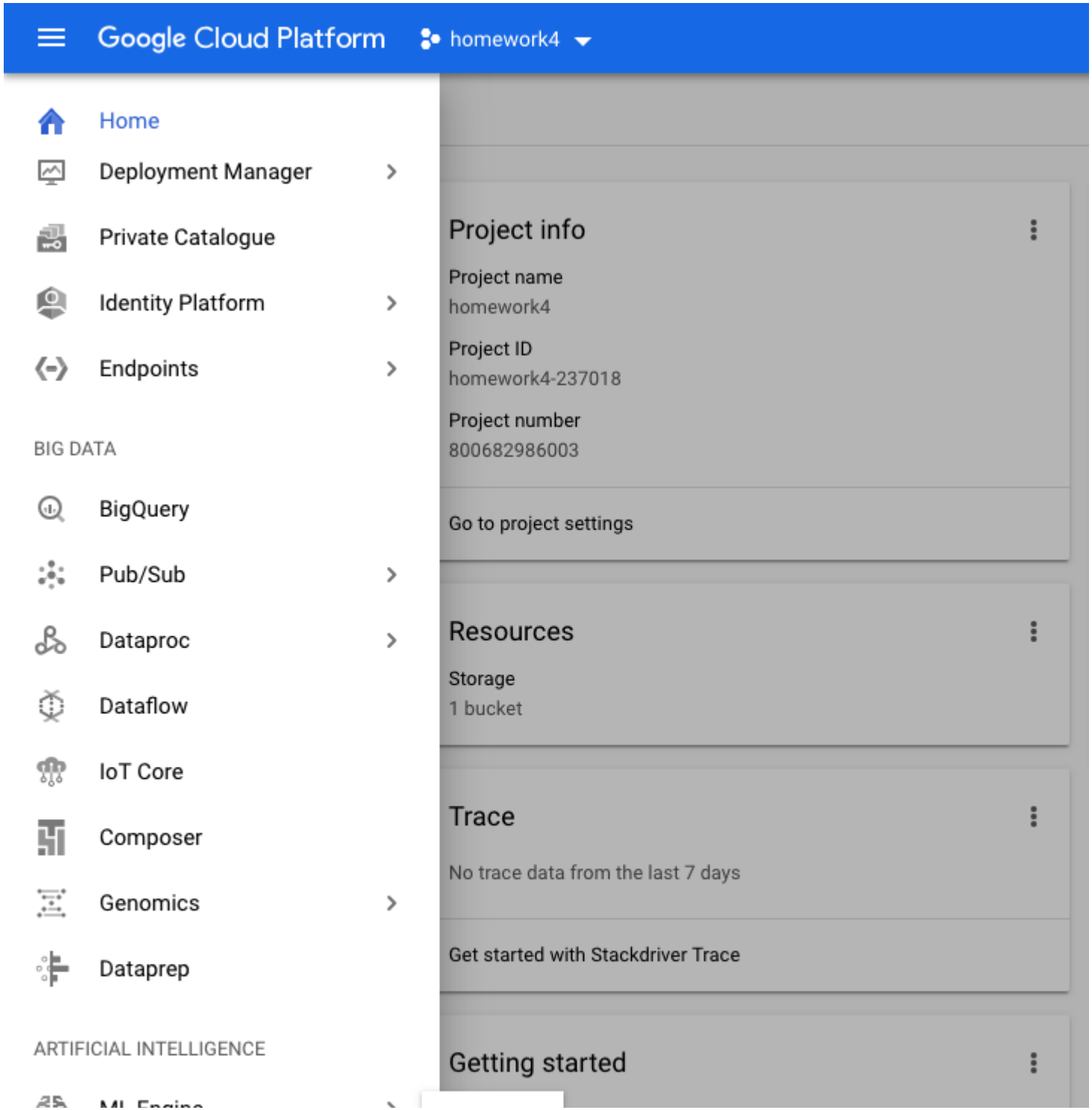
In *Part 0*, we created a bucket which makes it easier to store files on the cloud and enables different Google Cloud services to access those files. You can think of buckets as folders and all the services on cloud as applications which have access to these folders.

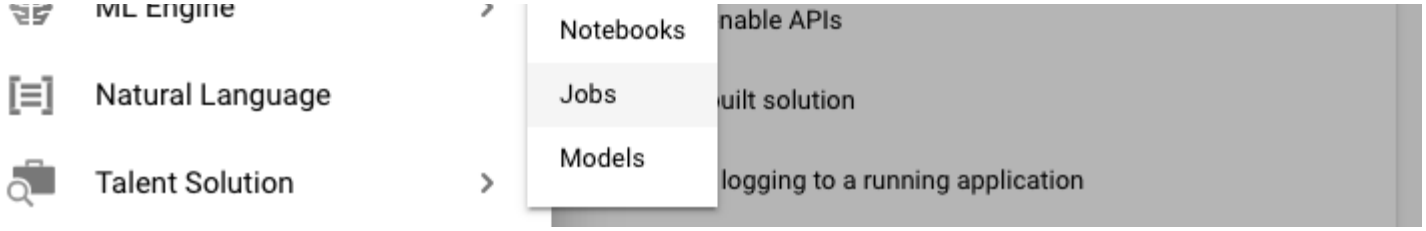
Go to Google Cloud Console (https://console.cloud.google.com/) and you should be able to see *Storage* under *Resources* card on project *Dashboard*. Click on it and it will take you to *Storage Browser*. Select your bucket and upload `train_gcinput.csv` in the root of the bucket. Creat a new subdirectory in the root of the bucket and name it as `output`, we will use it later to store logs and trained model. It should like the following figure:



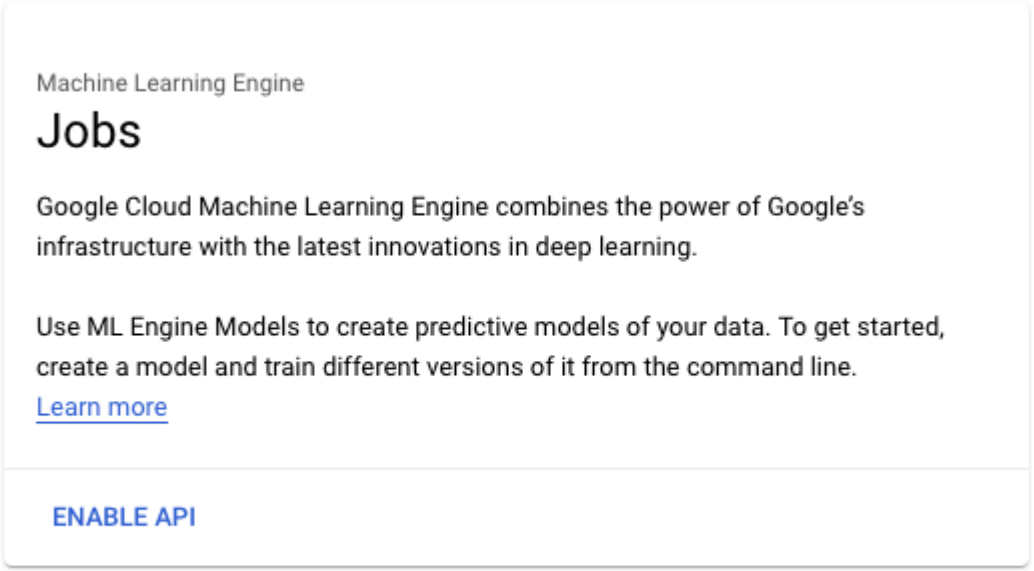
Training the Model

Now, we have all the files necessary to train our model. Go to Google Cloud Console (https://console.cloud.google.com) and from the sidebar, select *ML Engine* > *Jobs* as shown in the following figure.

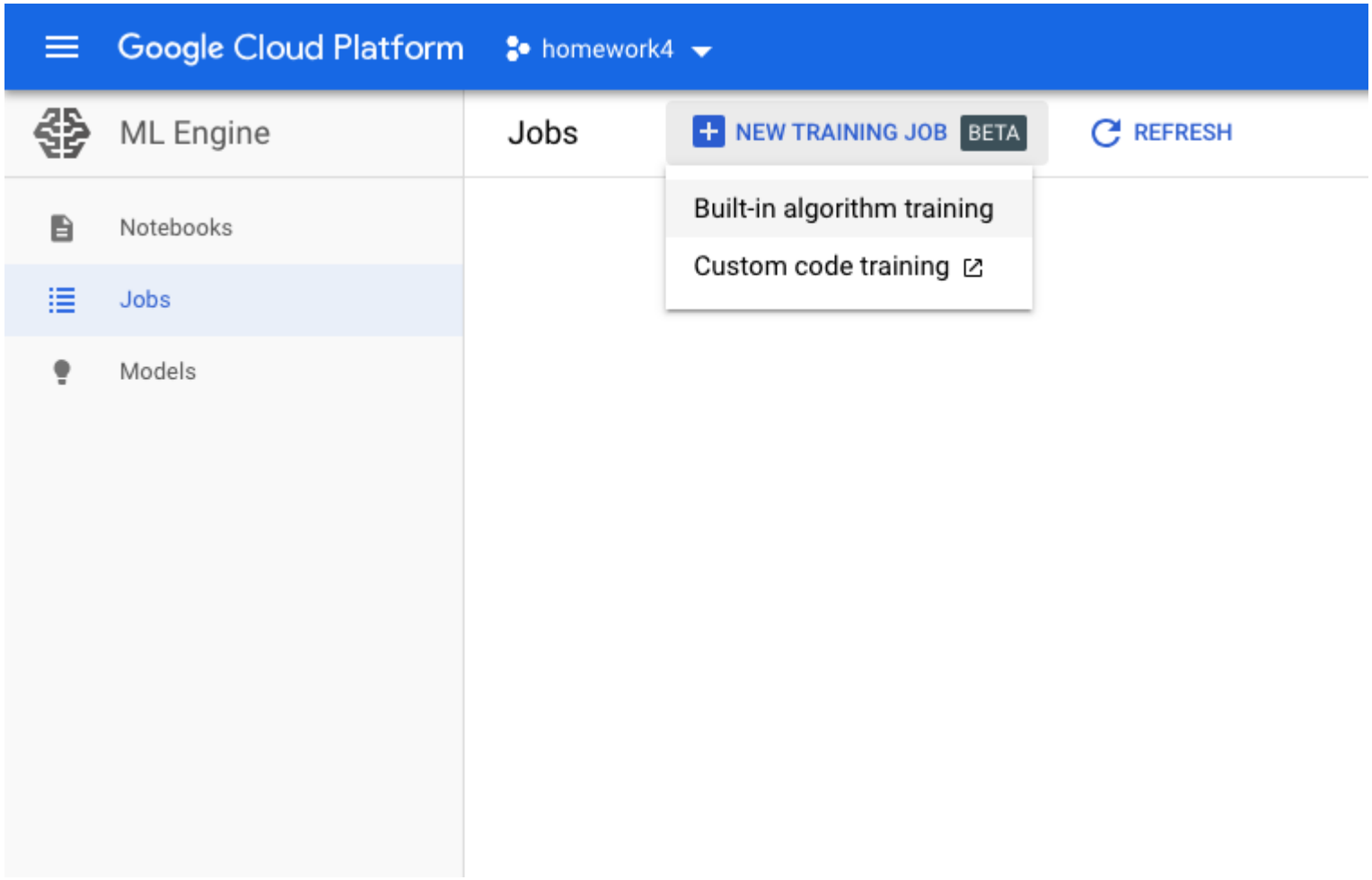

Google Cloud Platform
homework4
Home
Deployment Manager
Private Catalogue
Identity Platform
Endpoints
BIG DATA
BigQuery
Pub/Sub
Dataproc
Dataflow
IoT Core
Composer
Genomics
Dataprep
ARTIFICIAL INTELLIGENCE
Project info
Project name
homework4
Project ID
homework4-237018
Project number
800682986003
Go to project settings
Resources
Storage
1 bucket
Trace
No trace data from the last 7 days
Get started with Stackdriver Trace
Getting started



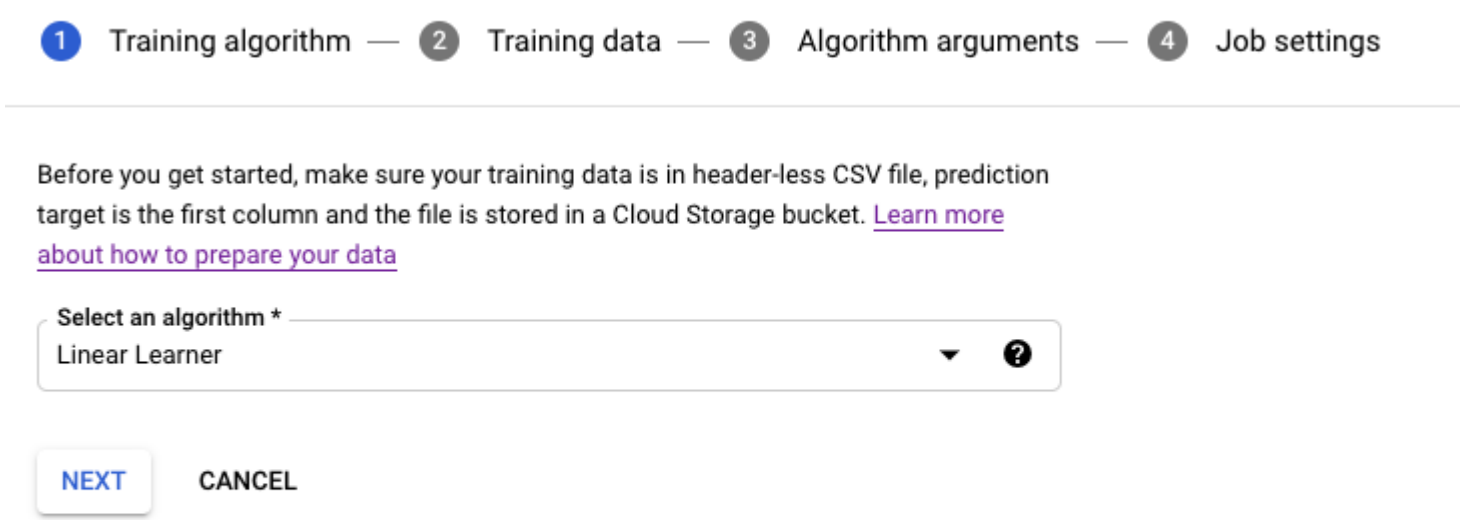
If you have not used *Jobs* before then you first need to Enable API to use this service. Click *ENABLE API* (it might take a while).



Google cloud provides some built-in models which can be trained on any dataset. Once, the API is enabled, click on *NEW TRAINING JOB* and select *Built-in algorithm training* from the dropdown as shown below:



We will use a built-in linear classifier for training. Select *Linear Learner* as your training algorithm and click *NEXT*



Click *BROWSE* and select `train_gcinput.csv` from the bucket under *Training data path* and setup the remaining arguments as shown in the following screenshot. The validation data is used while training the model for hyper-parameter tuning and we will also use 10% of the training data as Test data for model evaluation. Note that this is not the final test data that we will be using for predictions. Click *NEXT*.

Enable automatic data preprocessing

Make sure that your data is in a header-less CSV file and the prediction target is in the first column. Learn more

Training data path *

gs:// model-bucket-alpha/train_gcinput.csv BROWSE

The Cloud Storage path where the training data is stored

Validation data

Use a percentage of training data

10 %

Test data (optional)

Use a percentage of training data

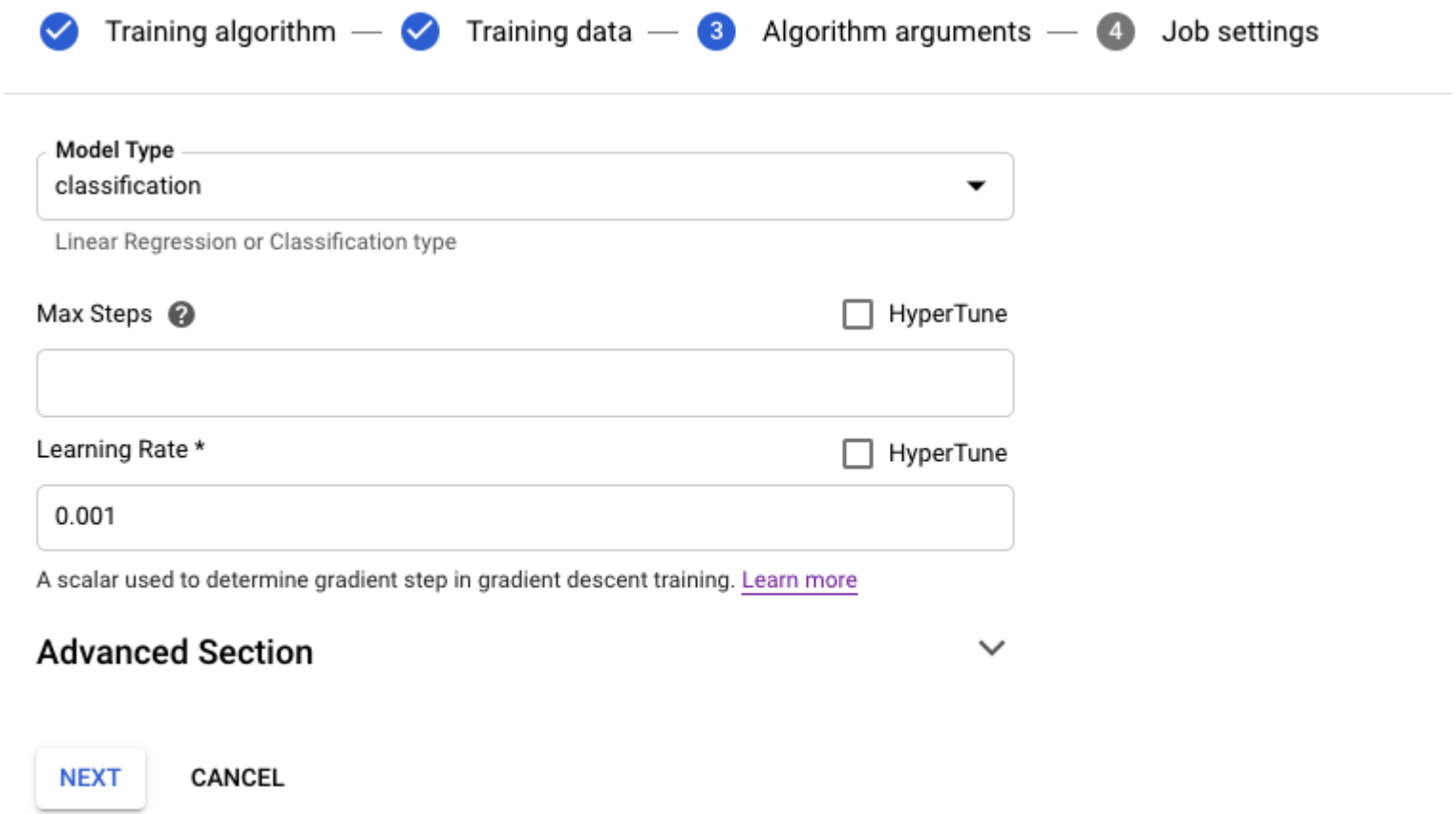
10 %

Output directory *

gs:// model-bucket-alpha/output/ BROWSE

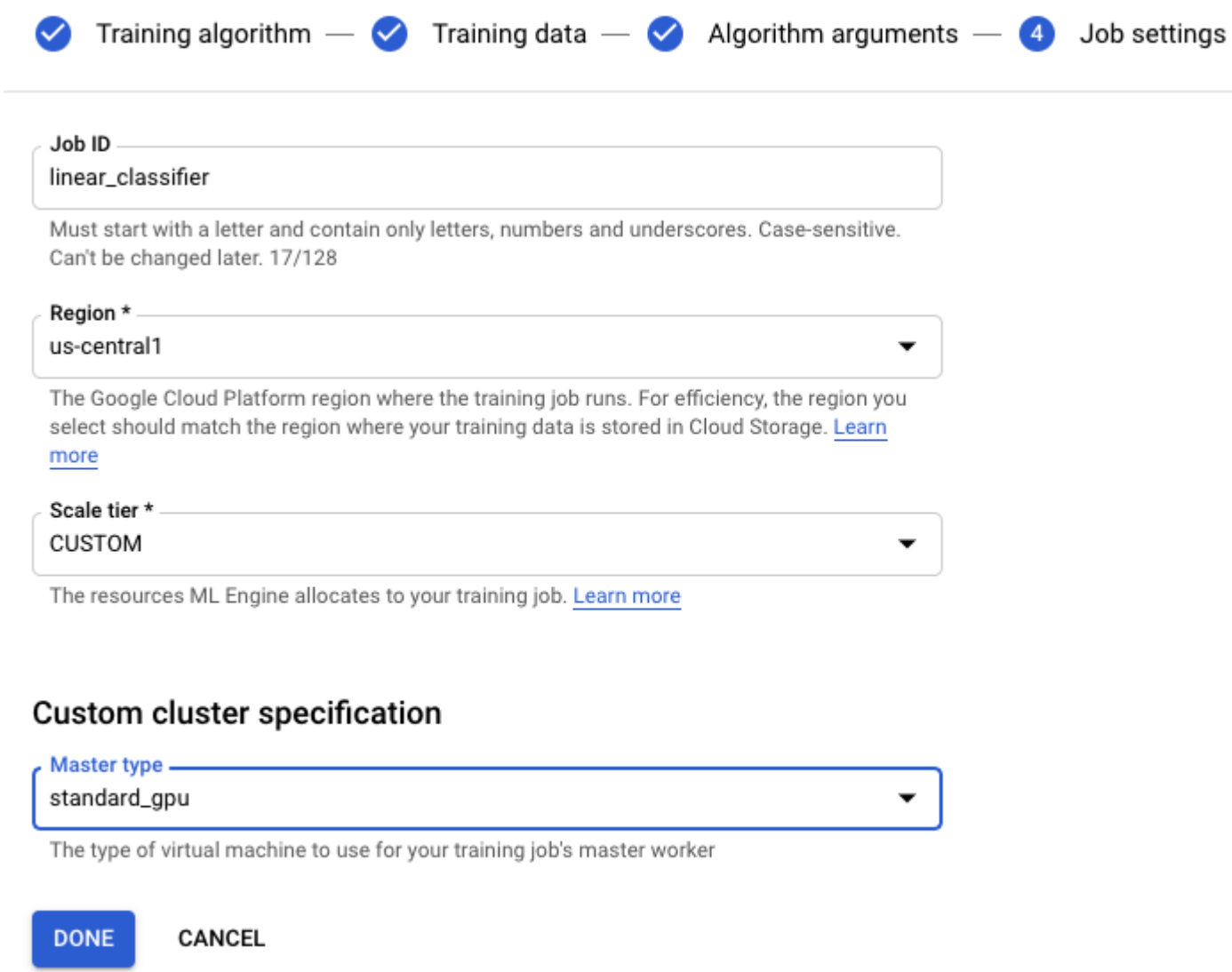
The path to the Google Cloud Storage location where you want the trained model and other training job output to be stored.

Select model type as classification and leave everything as is, click *NEXT*.

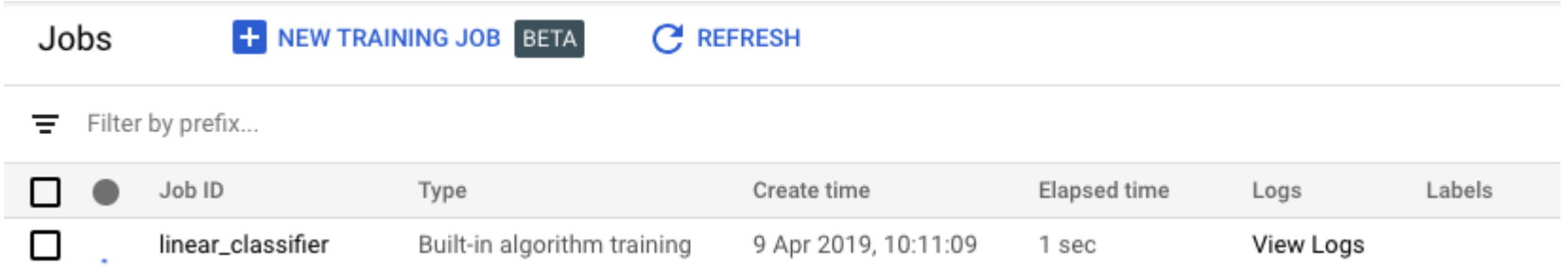


In the job setup, you can specify a *Job ID* and the type of machine (https://cloud.google.com/ml-engine/docs/tensorflow/machine-types) you want to run this on. We will run this job on a standard_gpu machine. Note that there are other high-end machines available as well. Click *DONE*.

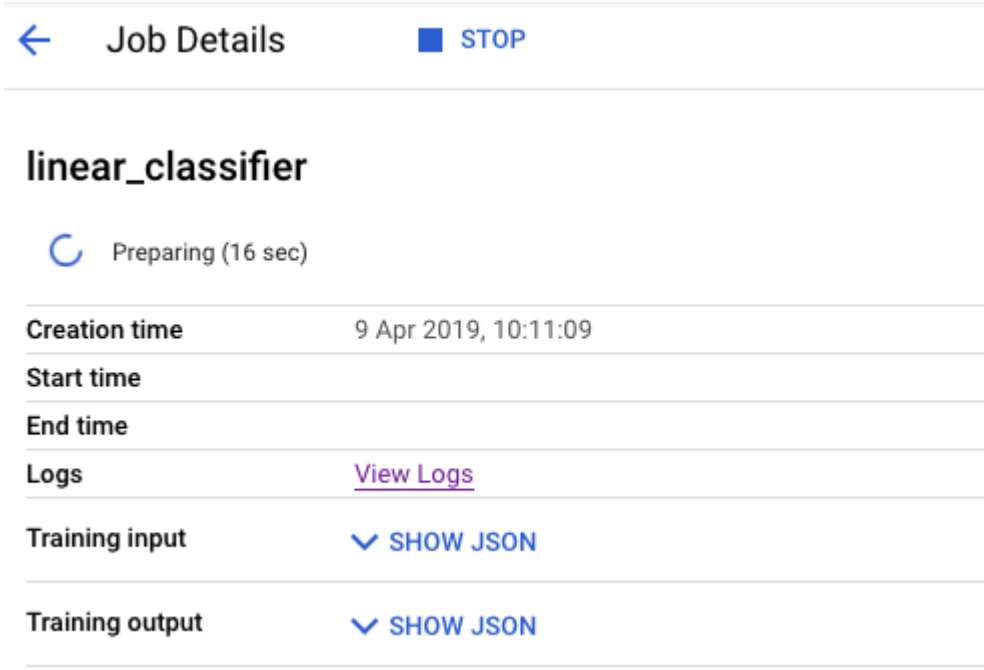
⚠High-end machines may incur high computational cost and built-in models do not have the capacity to fully utilize them. Details (https://cloud.google.com/ml-engine/docs/algorithms/linear-learner).



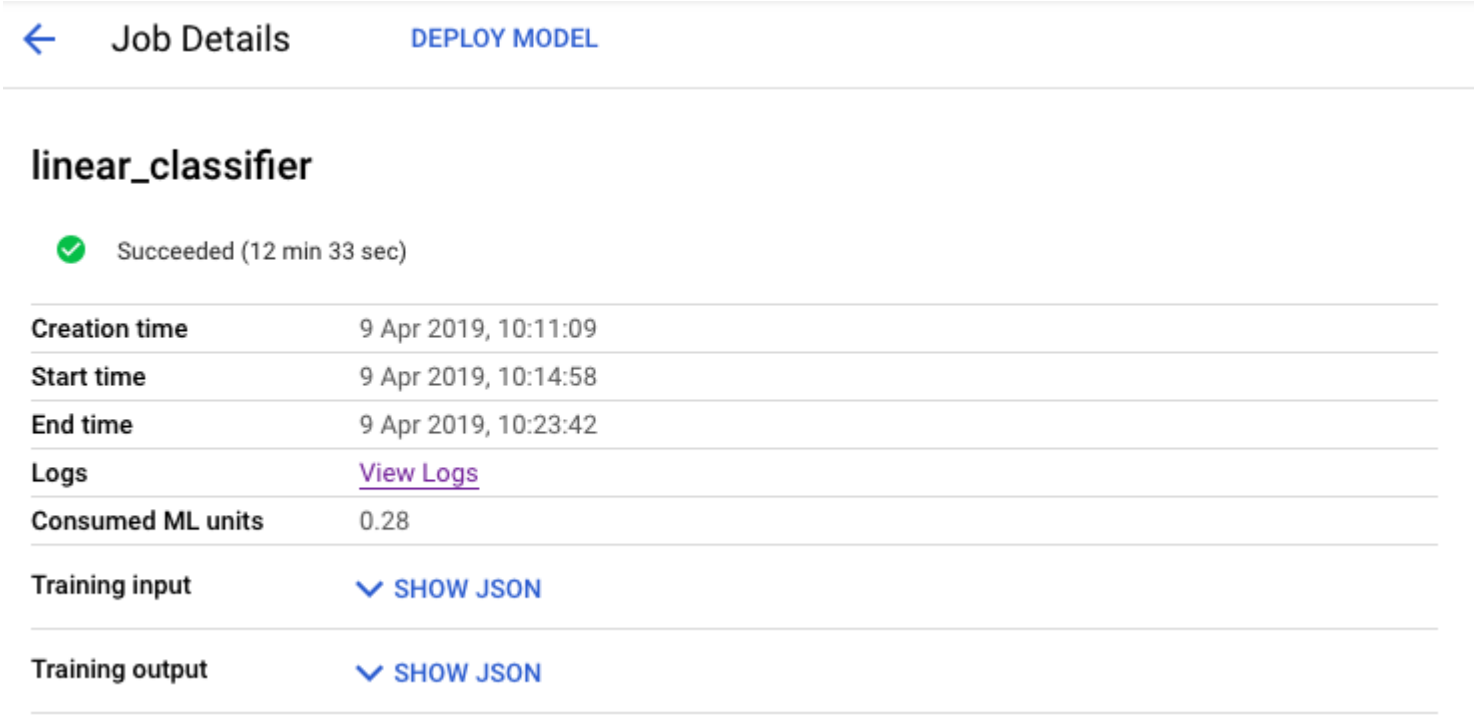
Your job should be up and running now and you should be able to see a list of running jobs:



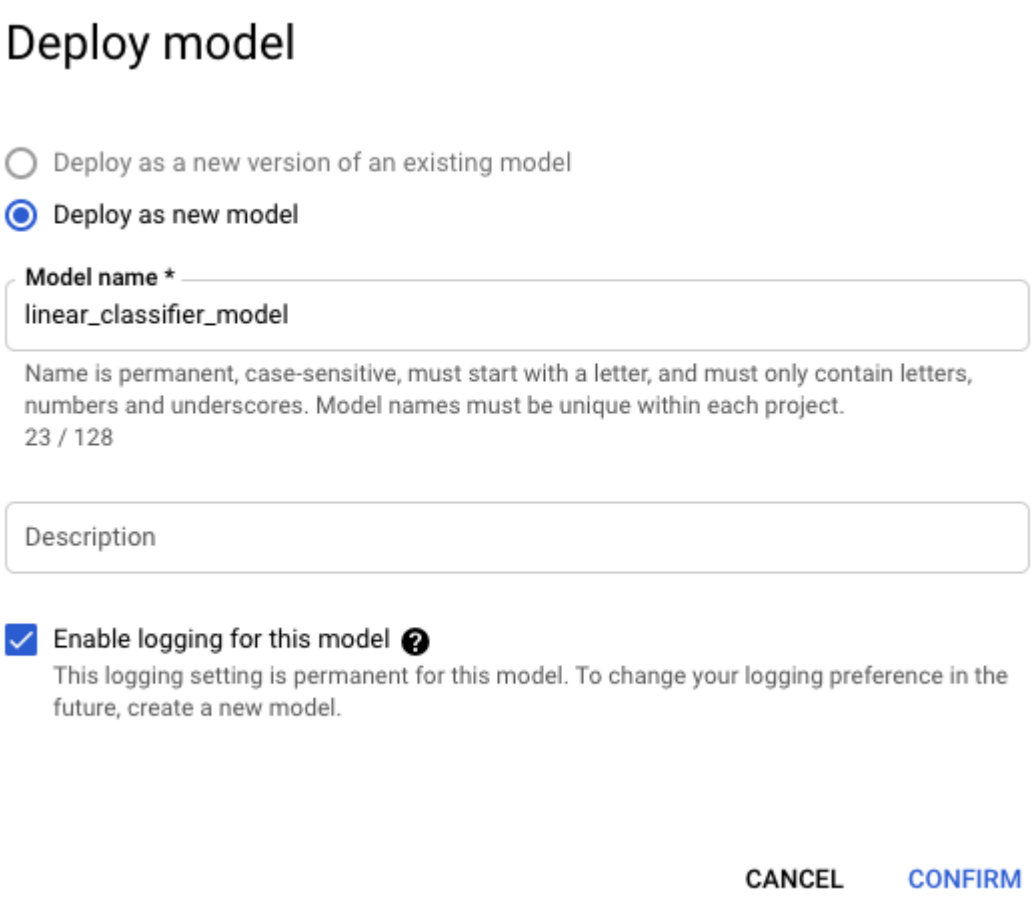
You can open a job by clicking on Job ID. This will take you to the job progress page (shown below) where you can mointor resources being used by the job, job logs, parameters and progress. Click on View Logs to view and stream the logs, if something goes wrong then you should be able to see it in the logs.



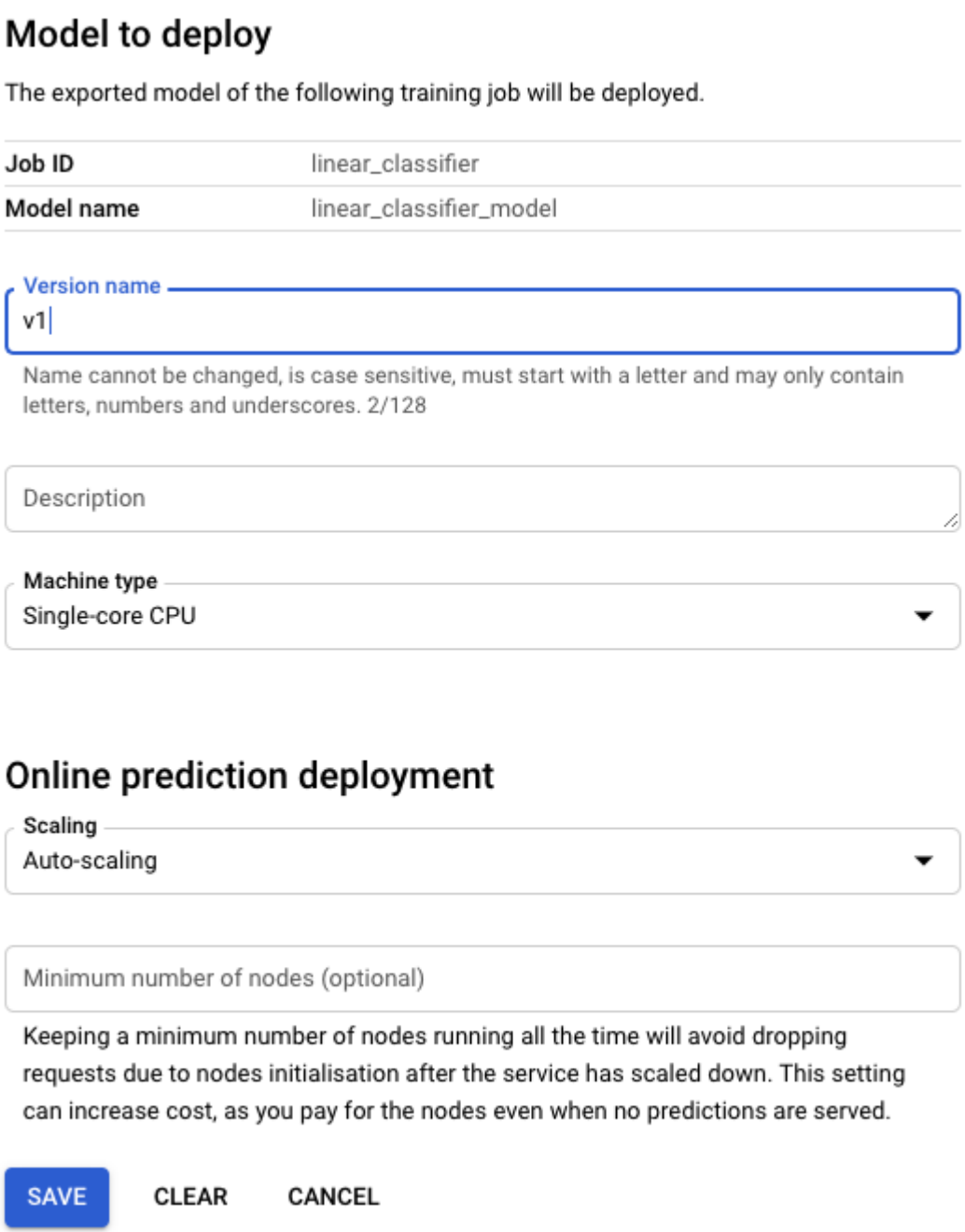
When the job is completed you should be able to see the following page. It takes about 12-15 minutes.



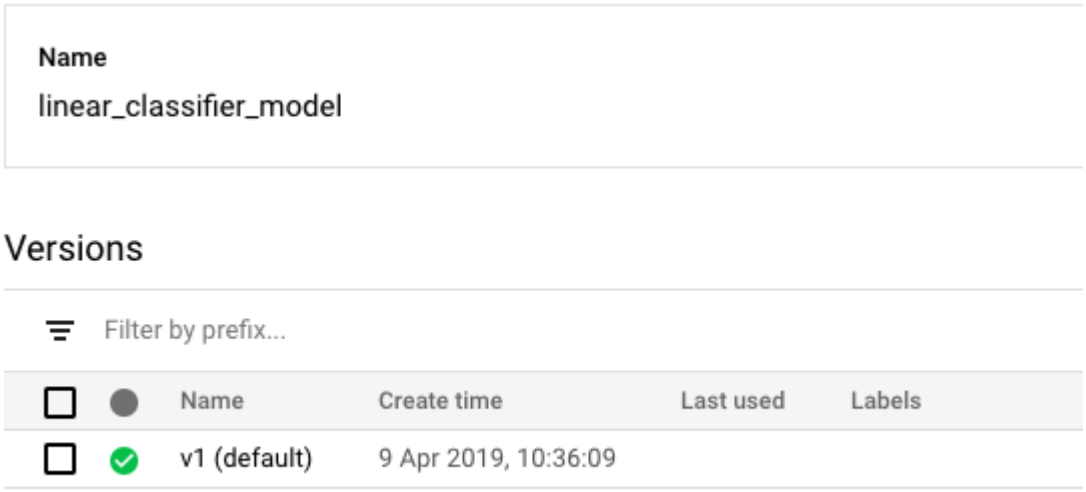
Now that our model is trained, we can deploy it and make predictions using this model on the cloud. Click *Deploy Model* button in the figure above. Deploy as a new model and specify a name, click *CONFIRM*



Specify a version (v1) and click *SAVE*.



It will take you to your model page, your model is now being deployed, wait for the green check to appear next to version before you proceed.

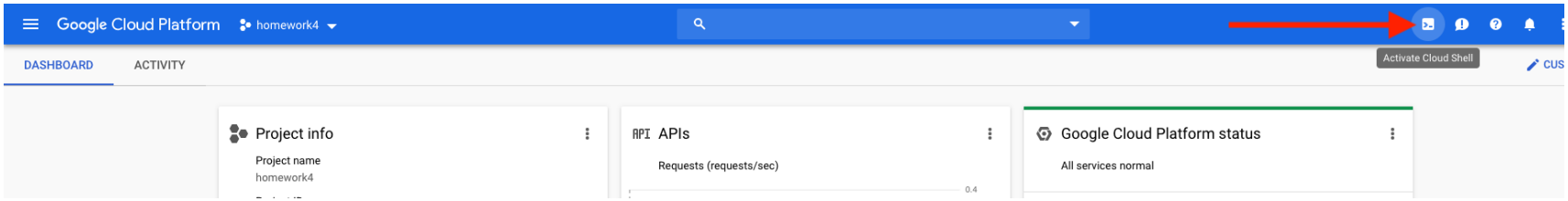


Your model is now deployed on the cloud and can be used to make predictions.

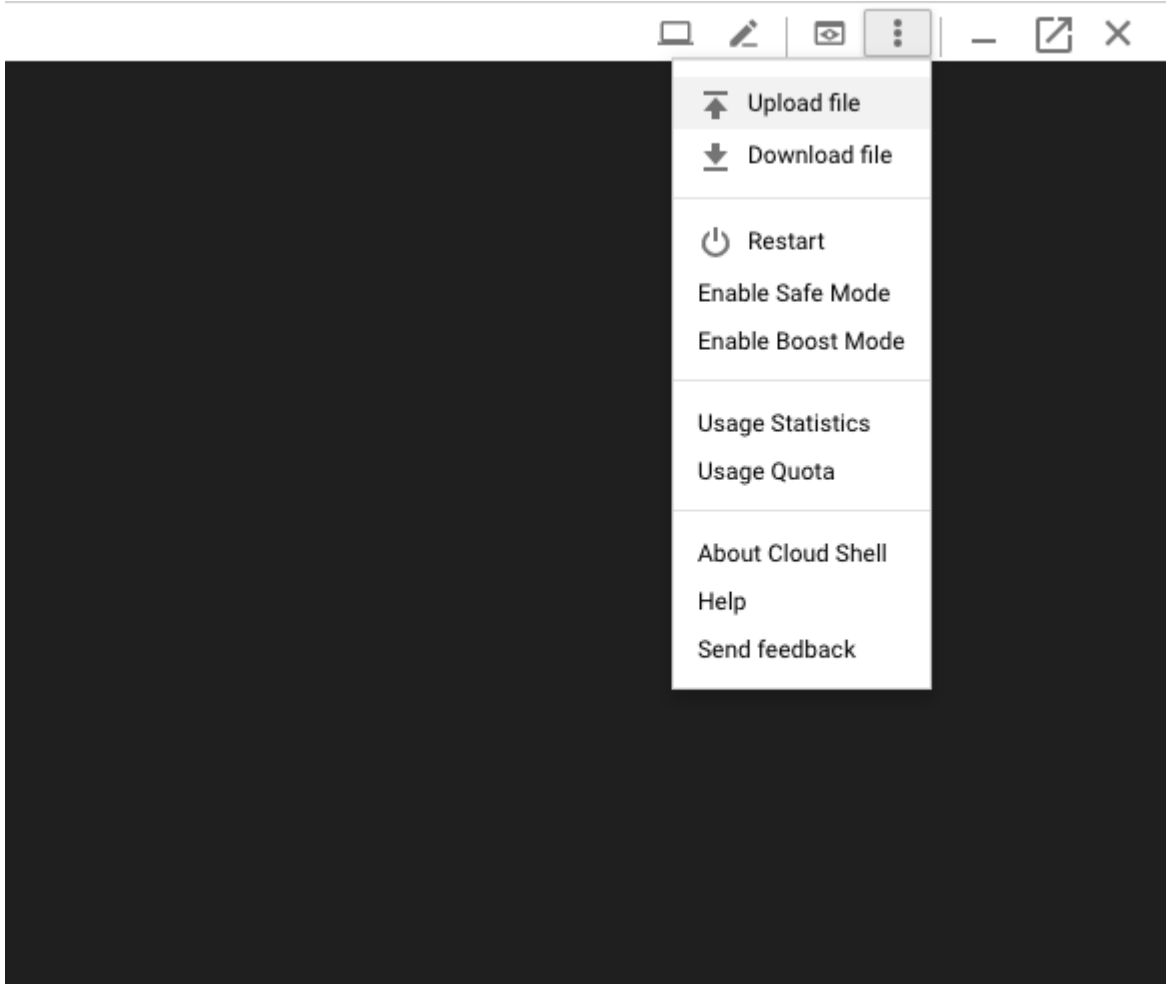
Making Predictions

We can either use our local machine to make calls to the Google Cloud and make predictions using gcloud command from Google Cloud SDK (https://cloud.google.com/sdk/docs/) or we can use Google Cloud Shell (https://cloud.google.com/shell/) which comes with Google Cloud SDK. We will use *Google Cloud Shell* in this tutorial.

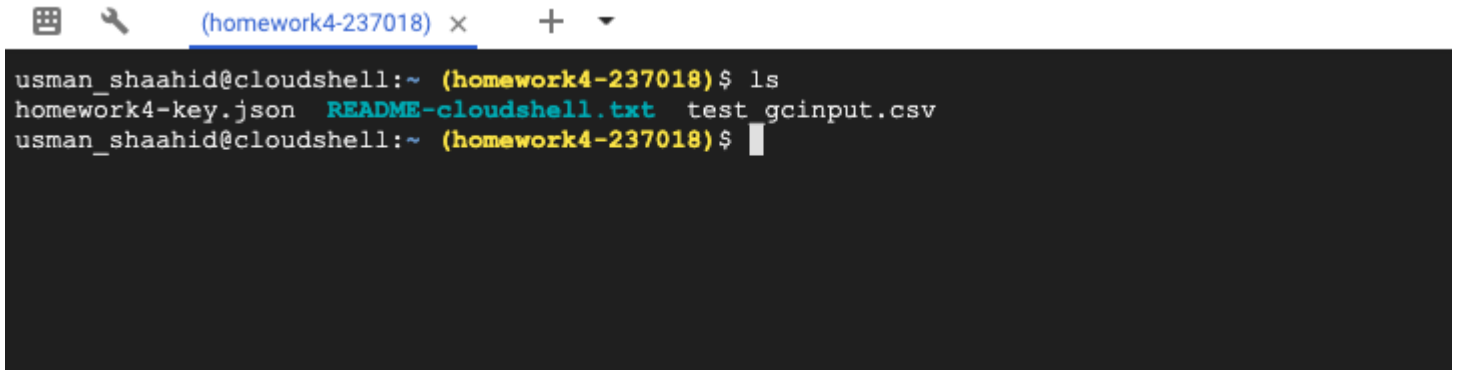
Visit Google Cloud Console (https://console.cloud.google.com) and click on *Activate Cloud Shell* icon in the top bar as shown in the following screenshot.



This should open a terminal in the browser and this is just like a regular linux terminal, you can give a command `ls` to see which files are there. Now, we can upload files to this terminal using the menu on top-right of the terminal window. We need to upload `homework4-key.json` file that we downloaded earlier and `test_gcinput.csv` file from `data` directory.



Once the files are uploaded you can verify success using `ls` command.



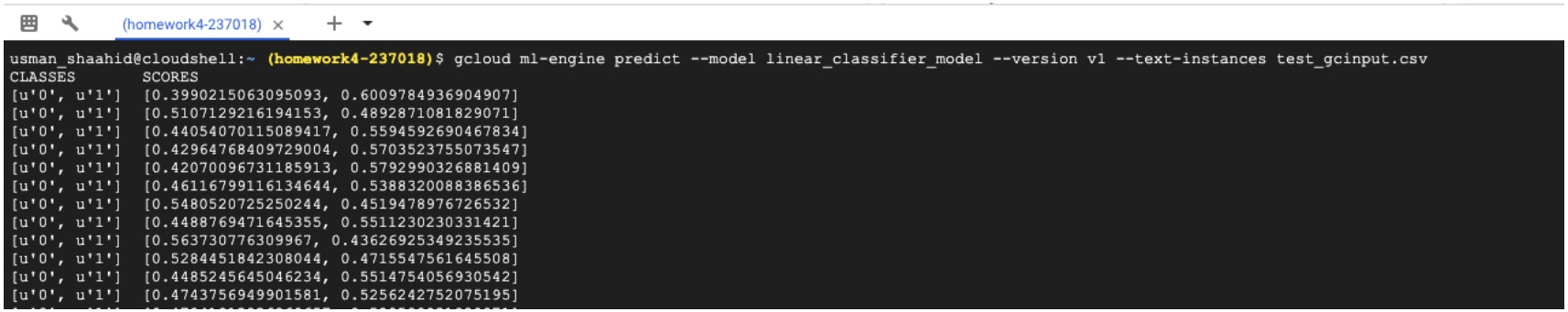
Now, you can run the following command to enable authentication using the key

```
export GOOGLE_APPLICATION_CREDENTIALS=homework4-key.json
```

Once, the authentication is enabled, you can use the following `gcloud` command to make predictions:

```
gcloud ml-engine predict --model linear_classifier_model --version v1 --text-instances test_gcinput.csv
```

If everything goes right, then you should be able to see the following output. Here, *CLASSES* are just binary class labels 0/1 and *SCORES* shows the probability of being in that class.



Let us save these results in JSON format using this command.

```
gcloud ml-engine predict --model linear_classifier_model --version v1 --text-instances test_gcinput.csv
--format=json > results.json
```

This should create a file `results.json` in current directory. You can view the produced JSON using this command:

```
cat results.json
```

💾Select *Download File* from the top-right menu and save `results.json` in your `sub` (submission) subdirectory and that's it!

Before Submitting

💾Make sure that you have all the required screenshots and data files in `sub` subdirectory signified by this icon.

📝Make sure that you have made all necessary additions to the notebook signified by this icon.

⚠️Make sure that you have read all the warnings.

Finally,

⚠Once the homework is fully graded, you may remove your project from Google cloud or stop any services to avoid running expenses. Most likely there will be no running expenses even if you do not stop/remove the project but it is usually not recommended to keep unecessary components running on Google Cloud as it may incur additional cost.

⚠Before submitting written part of the HW, make sure that all parts of your solution are completely visible, pay special attention to the images you have included.

Now, follow the instructions on top of this notebook to make submissions. Good luck!