

Scaling Data Analysis

Google Cloud Platform Fundamentals: Big Data and Machine Learning

Version #1.1



© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Notes:

75 minutes lecture + 45 minutes labs

In the previous chapter, we looked at ways to migrate your workloads to GCP. Those -- relational databases, Hadoop ecosystem -- are probably things you are doing already, and we wanted to make it easy to move those workloads to GCP. Hence, managed MySQL and Managed Hadoop. While those are good, as we discussed in Chapter 1, it is not taking full advantage of what the Cloud has to offer. In this chapter, we will look at more transformational use cases. These are going to be about **changing** how you compute with GCP. More transformational use cases. What kinds of transformational use cases?

Agenda



Notes:

1. Introduction

Overview of GCP as a whole, but with emphasis on the data-handling aspects of the platform

- GCP, GCP Big Data
- Usage scenarios
- Create an account on GCP

2. Foundation of GCP

Compute and Storage with a focus on their value in data ingest, storage, and federated analysis

- Compute Engine
- Cloud Storage
- Start GCE instance
- Upload data to GCS

3. Data analytics on the Cloud

Common use cases that Google manages for you and for which there is an easy migration path to the Cloud

- Cloud SQL
- Dataproc
- Import data into and query Cloud SQL
- Machine Learning with Dataproc

In the morning, we will complete Modules 1 and 2 and get halfway through Module 3.

4a. Scaling data analysis

Change how you compute, not just where you compute with GCP

- Datalab
- Datastore, Big Table
- BigQuery

5. TensorFlow

Change how you compute, not just where you compute with GCP

- TensorFlow
- Datalab instance
- BigQuery
- Demand forecasting with ML

6. Data processing architectures

Scaleable, reliable data processing on GCP

- Pub/Sub
- Dataflow

7. Summary

Course summary

- Resources

Please feel free to use the appendixes for self-study.

In the morning, we will get halfway through Module 3.

Please feel free to use the appendixes for self-study.

Agenda

Fast random access

Warehouse and interactively query petabytes

Interactive, iterative development + Lab

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.



Notes:

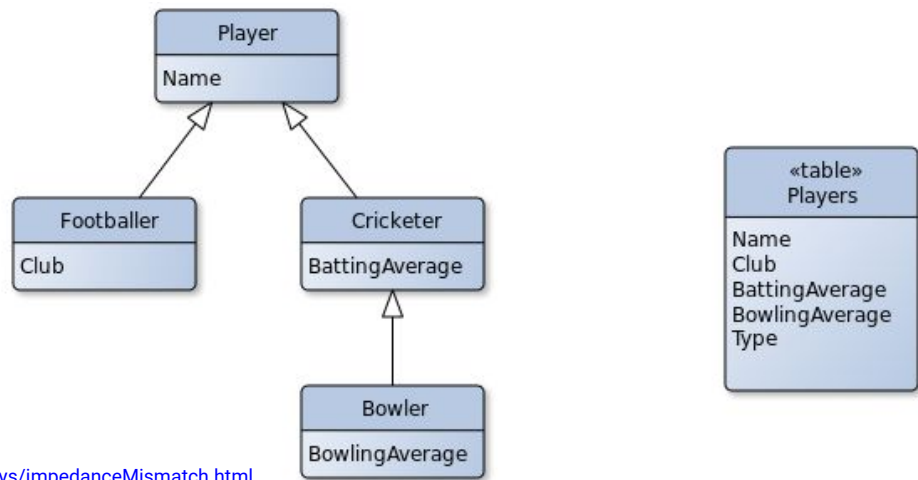
This chapter is going to be transformational use cases taking advantage of GCP. It's about database technologies, picking up from the previous chapter. Then we'll look at notebooks, large-scale querying of structured data and finally machine learning.

Now, why do you use a database? [ask, and most people will answer that it is to store data]

No, really, the reason you use a database rather than simply store the data on a tape drive is that you want fast random access to it. It's about fast queries.

Yet, the solution of using your typical relational database has certain drawbacks as we looked at in the review to module 3 – for example, if the data are not structured, it leads to a mismatch.

Relational tables are hard to use from object-oriented programs



<http://www.agiledata.org/essays/impedanceMismatch.html>

https://en.wikipedia.org/wiki/Object_database

Notes:

Image from https://de.wikipedia.org/wiki/Objektrelationale_Abbildung.

Every cricketer bats, so they all have a batting average, but only bowlers bowl. The relational table has to have both fields for everybody with NULLs, and so on.

Choose storage based on access pattern

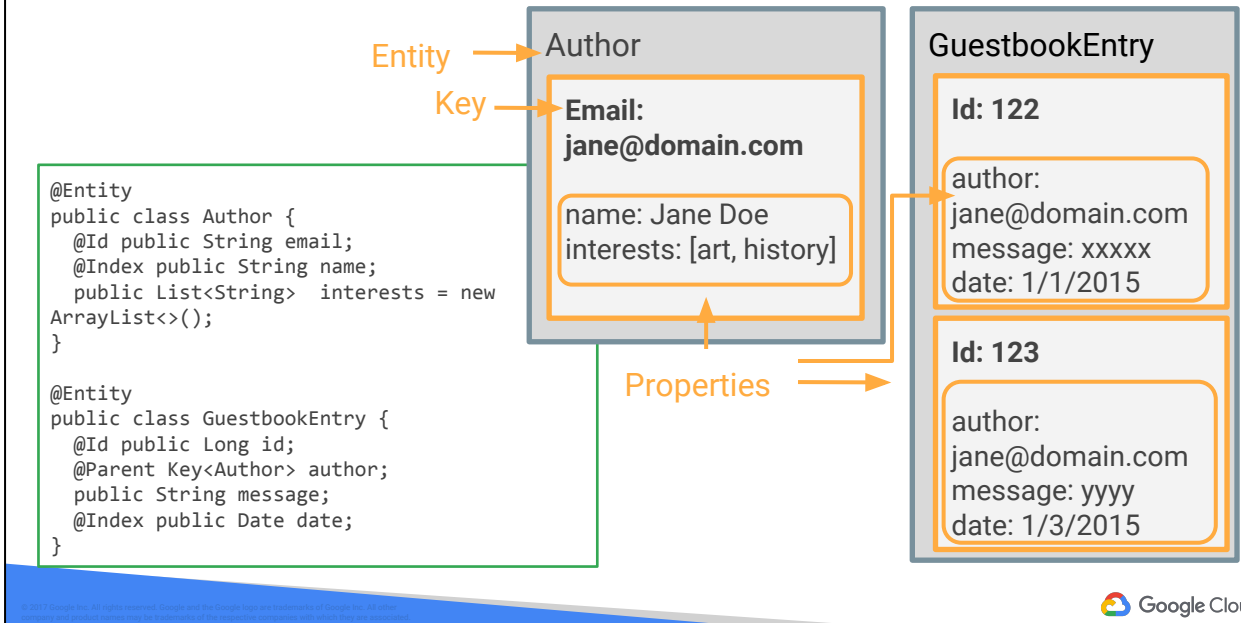
	Cloud Storage	Cloud SQL	Datastore	Bigtable	BigQuery
Capacity	Petabytes +	Gigabytes	Terabytes	Petabytes	Petabytes
Access metaphor	Like files in a file system	Relational database	Persistent Hashmap	Key-value(s), HBase API	Relational
Read	Have to copy to local disk	SELECT rows	filter objects on property	scan rows	SELECT rows
Write	One file	INSERT row	put object	put row	Batch/stream
Update granularity	An object (a "file")	Field	Attribute	Row	Field
Usage	Store blobs	No-ops SQL database on the cloud	Structured data from AppEngine apps	No-ops, high throughput, scalable, flattened data	Interactive SQL* querying fully managed warehouse

Notes:

Datastore/Bigtable/BigQuery are better suited to large data and the Cloud. We will look at all three of these in this chapter, starting with Cloud Datastore.

The point of this section is that if your primary use case is to access the data from OO programs, a better solution might be an object database, such as Datastore. We start this by motivating it in terms of the O-R impedance mismatch.

Datastore is like a persistent HashMap



Notes:

It's a persistent hashmap because keys identify objects. Because Datastore stores objects, we have no impedance mismatch issues.

Read <https://github.com/objectify/objectify/wiki> for the full metaphor.

The annotations here are "Objectify". You can also use JDO or JPA with Datastore, but Objectify provides a better level of abstraction.

Because of the Parent declaration, the two classes are part of an EntityGroup, and changes will be transactional. If instead, we had simply said: Key<Author> author then we'd be doing the same relationship, but without a transaction.

If it was Author author then the field values would have been flattened and written into the GuestbookEntry table. Ref<Author> author gives you access to the object itself (not just the key), so it's a combination of the two.

CRUD operations are easily implemented on Datastore

```
@Entity
public class Author {
    @Id public String email;
    @Index public String name;
    public List<String> interests =
        new ArrayList<>();
}
```

// CREATE

```
Author xjin = new Author("xjin@bu.edu", "Ha Jin");
xjin.interests.add("Misty Poetry");
ofy.save().entity(xjin);
```

// READ

```
Iterable<Author> authors =
    ofy().load().type(Author.class).filter("name", "Ha Jin");
Author jh = ofy().load().type(Author.class).id("xjin@bu.edu").now();
```

// UPDATE

```
jh.name = "Jīn Xuěfēi (金雪飞)";
ofy().save().entity(jh).now();
```

// DELETE

```
ofy().delete().entity(jh).now();
```

Notes:

Create, read, update, delete operations using the open-source Objectify API. Other APIs are also available in languages other than Java.

By indexing name, you make the filter() operation faster. Index as many properties as you want. Pricing remains unaffected by number of indices (since March 2016).

Ha Jin is the pen-name for Jin Xuefei, written in Chinese as 金雪飞.

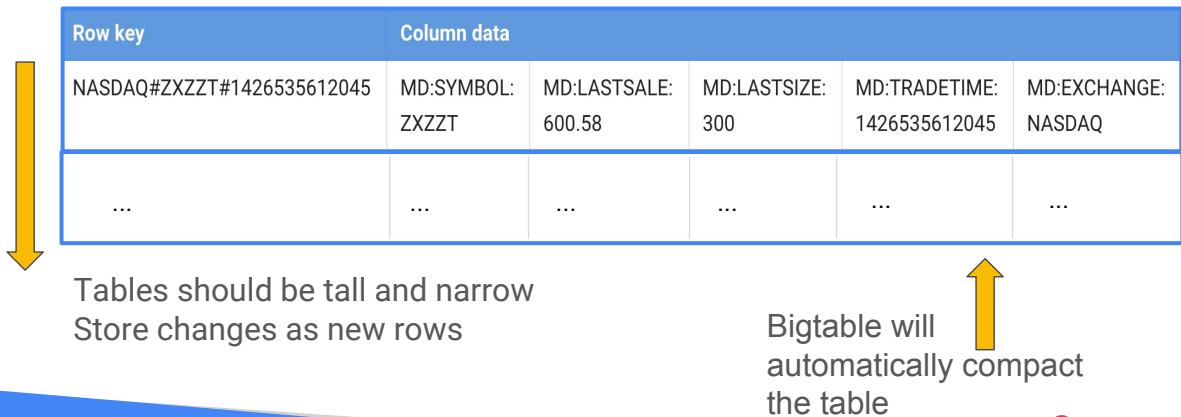
Choose storage product based on access pattern

	Cloud Storage	Cloud SQL	Datastore	Bigtable	BigQuery
Capacity	Petabytes +	Gigabytes	Terabytes	Petabytes	Petabytes
Access metaphor	Like files in a file system	Relational database	Persistent Hashmap	Key-value(s), HBase API	Relational
Read	Have to copy to local disk	SELECT rows	filter objects on property	scan rows	SELECT rows
Write	One file	INSERT row	put object	put row	Batch/stream
Update granularity	An object (a "file")	Field	Attribute	Row (write new row instead)	Field
Usage	Store blobs	No-ops SQL database on the cloud	Structured data from AppEngine apps	No-ops, high throughput, scalable, flattened data	Interactive SQL* querying fully managed warehouse

Notes:

With Bigtable, you are better off writing new rows than trying to update rows because such transactions are very slow and expensive. For high-throughput, it is better to allow clients to get all the rows and simply use the latest version. This is shown in the stock market data example on next slide. Instead of updating the price of a security, we add a new market-data trade message to the table.

Bigtable is meant for high throughput data where access is primarily for a range of Row Key prefixes



Row key	Column data				
NASDAQ#ZXZZT#1426535612045	MD:SYMBOL: ZXZZT	MD:LASTSALE: 600.58	MD:LASTSIZE: 300	MD:TRADETIME: 1426535612045	MD:EXCHANGE: NASDAQ
...

Tables should be tall and narrow
Store changes as new rows

Bigtable will automatically compact the table

Notes:

<https://cloud.google.com/bigtable/docs/schema-design-time-series>

- Tall and narrow: each trade is its own row. This will result in 100s of millions of rows per day. This is fine.
- Autobalanced: tables are broken up into “tablets” based on row-key. Data are stored in a durable way with redundancy. As entries get deleted, Bigtable will automatically compact the table to reduce overall storage.

Short meaningful column names reduce storage and RPC overhead

Design row key with most common query in mind

Column families is a quick way to get some hierarchy

Row key	Column data				
NASDAQ#ZXZZT#1426535612045	MD:SYMBOL: ZXZZT	MD:LASTSALE: 600.58	MD:LASTSIZE: 300	MD:TRADETIME: 1426535612045	MD:EXCHANGE: NASDAQ

Design row key to minimize hotspots

Use short column names
Designed for sparse tables

Google Cloud

Notes:

The MD here is a “column family”, just a cheap way of getting some hierarchy into Bigtable. It is supported by the APIs (see the next slide).

<https://cloud.google.com/bigtable/docs/schema-design-time-series>

- The most common query is by exchange and ticker for a given start and end time. So, you will need to use the values of EXCHANGE, SYMBOL, and QUOTETIME.
- Hotspot: Here, all Zs will be stored together in contiguous rows and probably on a different machine from the Ms. If this is not the case, you should consider randomizing and/or salting the keys (for example, by storing the hash of the key).
- Short column names because they contribute to storage. They are not just metadata. This is so that Bigtable can handle sparse tables more easily. So, for example, you can store boolean variables just by storing the column-name if the value is True and nothing if the value is False. This is useful when storing a binary matrix such as a social graph where any of your 1b users could follow any of the other 1b users. Instead of storing a 1b x 1b table, you simply store the row-ids of the users that anyone is following.

Can work with Bigtable using the HBase API

```
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.*;

byte[] CF = Bytes.toBytes("MD"); // column family
Connection connection = ConnectionFactory.createConnection(...)
Table table = null;
try {
    table = connection.getTable(TABLE_NAME);
    Put p = new Put(Bytes.toBytes("NASDAQ#GOOG #1234561234561"));
    p.addColumn(CF, Bytes.toBytes("SYMBOL"), Bytes.toBytes("GOOG"));
    p.addColumn(CF, Bytes.toBytes("LASTSALE"), Bytes.toBytes("742.03d"));
    ...
    table.put(p);
} finally {
    if (table != null) table.close();
}
```

Notes:

Besides the Java (HBase API) shown above, REST, Python, and Go are also available.

Agenda

Fast random access

Warehouse and interactively query petabytes

Interactive, iterative development + Lab

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.



Notes:

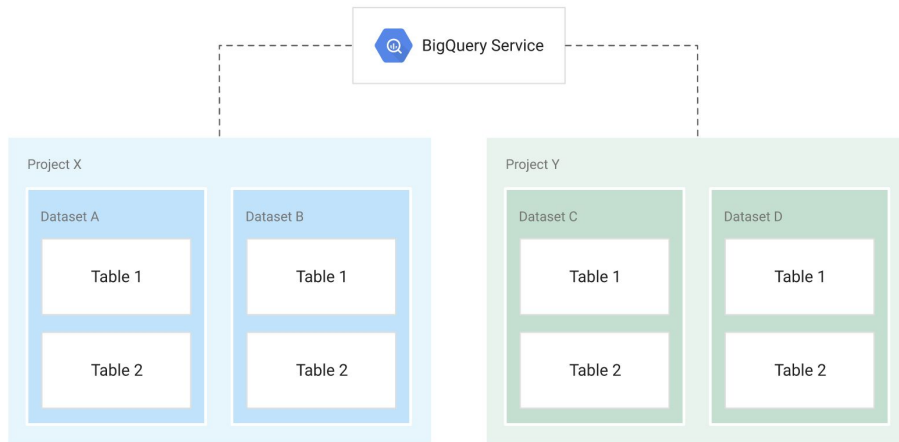
This chapter is going to be transformational use cases taking advantage of GCP. It's about database technologies, picking up from the previous chapter. Then we'll look at notebooks, large-scale querying of structured data and finally machine learning.

Now, why do you use a database? [ask, and most people will answer that it is to store data]

No, really, the reason you use a database rather than simply store the data on a tape drive is that you want fast random access to it. It's about fast queries.

Yet, the solution of using your typical relational database has certain drawbacks as we looked at in the review to module 3 – for example, if the data are not structured, it leads to a mismatch.

BigQuery is a fully managed data warehouse that lets you do ad-hoc SQL queries on massive volumes of data



© 2019 Google LLC. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other marks and names may be trademarks of their respective owners. All rights reserved.

 Google Cloud

<https://cloud.google.com/solutions/bigquery-data-warehouse>
<https://cloud.google.com/solutions/big-data/>

Another advantage is that you get to mash up data across projects.

In this slide we see the overall structure of BigQuery. Data tables are organized into units called datasets. Each data set is scoped to your project and this allows you to logically organize your data.

A demo of BigQuery on a 10 billion-row dataset shows what it is and what it can do

```
#standardsql
SELECT
  language, SUM(views) as views
FROM `bigquery-samples.wikipedia_benchmark.Wiki10B`
WHERE
  title like "%google%"
GROUP by language
ORDER by views DESC
```

Familiar, SQL 2011 query language

Interactive ad-hoc analysis of
petabyte-scale databases

No need to provision clusters

Notes:

<https://bigquery.cloud.google.com>

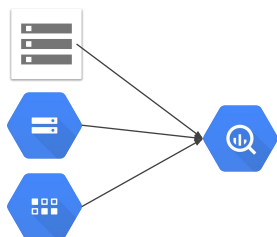
Let's start with a demo, so that you know what it is, before we get into the advantages.

This dataset is so large that this query would kill a relational database. Demonstrate the cost of this query.

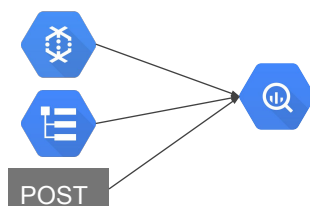
- Table size - 693 GB
- Number of rows 10,677,046,566 -- over 10 billion rows
- Processing time < 10 seconds

Three ways of loading data into BigQuery

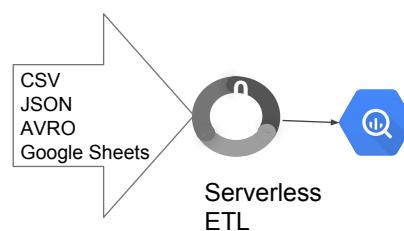
Files on disk or Cloud Storage



Stream Data



Federated data source



Notes:

1. From the web Console and files on disk, Cloud Storage or Cloud Datastore
2. Stream data with Cloud Dataflow, from Cloud Logging or with POST calls
3. Set up federated data source (serverless ETL!)
for CSV, JSON, Avro on GCS or Google Sheets

#1 is easy: just do it from Cloud Platform Console. It can also be done programmatically of course.

#3 is super-convenient. BigQuery will auto-detect CSV and JSON format! See <https://cloud.google.com/bigquery/federated-data-sources>.

With Federated data sources, you can directly query files on Cloud Storage, without having to ingest them into BigQuery

Create Table

Source Data ☒ Create from source ☐ Create empty table

Repeat job ?

Location ?

File format

Destination Table

Table name ?

Table type ☒ External table ?

Schema ☒ Automatically detect ?

Schema will be automatically generated. *Can also pass in a schema*

Options

Header rows to skip ?

Number of errors allowed ?



The key setting here is “External table”

Why is this cool? You have files on Cloud Storage and you are able to query them directly without having to provision or manage any servers! This is especially useful when your queries need a lot of massaging and UDFs.

How to do this from csv files in a gs bucket is shown in the diagram. First make the table definition. Query it like any native BigQuery table. The maximum data size for each federated table in a query is 10 GB; Files that are in CSV format and have quoted newlines or are compressed must be less than 1 GB each.

Agenda

Fast random access

Warehouse and interactively query petabytes

Interactive, iterative development + Lab

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.



Notes:

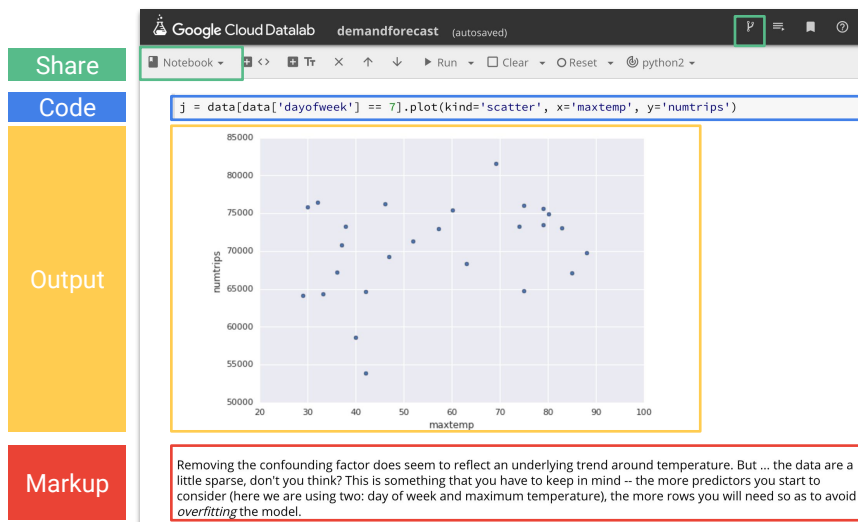
This chapter is going to be transformational use cases taking advantage of GCP. It's about database technologies, picking up from the previous chapter. Then we'll look at notebooks, large-scale querying of structured data and finally machine learning.

Now, why do you use a database? [ask, and most people will answer that it is to store data]

No, really, the reason you use a database rather than simply store the data on a tape drive is that you want fast random access to it. It's about fast queries.

Yet, the solution of using your typical relational database has certain drawbacks as we looked at in the review to module 3 – for example, if the data are not structured, it leads to a mismatch.

Increasingly, data analysis and machine learning are carried out in self-descriptive, shareable, executable notebooks



A typical notebook contains code, charts, and explanations

Image Source:
[Git Logo from Wikipedia](#)

Google Cloud

Who uses IPython notebooks today? Datalab is based on Jupyter and it's open source. Conceptually, they are the same thing. Code, output and markup, all together.

To share a notebook *within* a project, other users can simply "datalab connect" to the VM and work using the URL. Another way to share notebooks is through revision control systems (e.g. git).

This is what the interface to Datalab looks like. Notice how there are code sections interleaved with markup and output. This is what makes this style of computing so useful.

Increasingly, data analysis and machine learning are carried out in notebooks like this.

You can execute the code by either clicking the Run button or by typing Shift + Enter.

Notice that output here isn't just command line output; it's charts as well. The red section contains markup, so you can explain why you're doing what you're doing.

One of the green sections contains a button for exporting the notebook as a standalone file. The other green section is what you'd click if you wanted to commit your changes to a code repository, in GCP. In lab, we'll show you how to pull from and commit to code repositories.

Here's a demo:

[Do a brief demo and explore the link beforehand and find a notebook that works and which is visually appealing and talk about it for a few seconds.

<https://goo.gl/D8G1dX>

Here's one example:

The below notebook, which trains a Naive Bayes classifier to categorize news based on its title, shows a classic problem in ML, which is the imbalance of the classes with respect to label. Ask them to try and spot what it is. Tell them we'll review this in greater detail in on the third day.

http://nbviewer.jupyter.org/github/andressotov/News-Categorization-MNB/blob/master/News_Categorization_MNB.ipynb

Datalab is an open-source notebook built on Jupyter (IPython)

Analyze data in BigQuery,
Compute Engine or Cloud
Storage

Datalab is free—just pay for Google
Cloud resources

Use existing Python
packages



Notes:

Jupyter is an iPython notebook, so you can do all the things you can do in an iPython notebook, but you get four extra things:

- Reads from Cloud Storage or BigQuery (Google's data warehouse) directly
- Integrates in BigQuery, so you can do a BigQuery in SQL or JavaScript and transform to a DataFrame
- Can store notebooks that are checked into git

JavaScript is for user-defined functions in BigQuery.

You only pay for the cloud resources you use: the App Engine application, BigQuery, and any additional resources you decide to use, such as Cloud Storage.

Datalab notebooks are developed in an iterative, collaborative process



Similar to how you collaboratively edit Google Docs. When your Datalab notebook is hosted in the cloud, you can develop in a similar fashion.

And just as Google Docs are available even when your computer isn't on, so too are Datalab notebooks, when you run them in the Cloud.

To share a notebook *within* a project, other users can simply "datalab connect" to the VM and work using the URL. Another way to share notebooks is through revision control systems (e.g. git).

Datalab supports BigQuery

```
%bq query -n wxquery
SELECT EXTRACT(DAYOFYEAR FROM CAST(CONCAT(@YEAR,'-',mo,'-',da) AS TIMESTAMP))
      MIN(EXTRACT(DAYOFWEEK FROM CAST(CONCAT(@YEAR,'-',mo,'-',da) AS TIMEST.
      MIN(min) mintemp, MAX(max) maxtemp, MAX(IF(prcp=99.99,0,prcp)) rain
FROM `bigquery-public-data.noaa_gsod.gsod*`
WHERE stn='725030' AND _TABLE_SUFFIX = @YEAR
GROUP BY 1 ORDER BY daynumber DESC
```

```
query_parameters = [
  {
    'name': 'YEAR',
    'parameterType': {'type': 'STRING'},
    'parameterValue': [{'value': 2015}]
  }
]
weather = wxquery.execute(query_params=query_parameters).result().to_dataframe()
weather[:5]
```

	daynumber	dayofweek	mintemp	maxtemp	rain
0	365	5	46.0	48.2	0.17
1	364	4	34.0	48.0	0.13
2	363	3	33.8	46.9	0.37
3	362	2	39.0	62.1	0.02
4	361	1	46.0	62.6	0.14

To Pandas

Lab: Create ML dataset with BigQuery

Lab 2, Part 1: Create ML dataset with BigQuery

In this lab, we use BigQuery to create a dataset that we later use to build a taxi demand forecast system using Machine Learning.

- What kinds of things affect taxi demand?
- What are some ways to measure “demand”?



Notes:

Inputs: Weather, time of day, day of week, location, and so on. {The point is that these are things we can reasonably know on the day we want to predict demand for}.

Outputs: Total number of rides, total fares collected, total miles ridden in taxis, and so on. {The point is that these are aggregate measures, not related to a single taxi ride}.

Image from <https://pixabay.com/en/cab-yellow-cab-cars-street-taxi-453028/> (cc0).

Lab 2, Part 1: Create ML dataset with BigQuery

In this lab, we use BigQuery to create a dataset that we later use to build a taxi demand forecast system using Machine Learning.

1. Use BigQuery and Datalab to explore and visualize data
2. Build a Pandas dataframe that will be used as the training dataset for machine learning using TensorFlow

Notes:

Inputs: Weather, time of day, day of week, location, and so on. {The point is that these are things we can reasonably know on the day we want to predict demand for}.

Outputs: Total number of rides, total fares collected, total miles ridden in taxis, and so on. {The point is that these are aggregate measures, not related to a single taxi ride}.

Image from <https://pixabay.com/en/cab-yellow-cab-cars-street-taxi-453028/> (cc0).

Module Review

Module review

Match the use case on the left with the product on the right

Searching for objects by attribute value

1. Datalab

High-throughput writes of wide-column data

2. BigTable

Warehousing structured data

3. BigQuery

Develop Big Data algorithms interactively in Python

4. Datastore

Module review answer

Match the use case on the left with the product on the right

Searching for objects by attribute value (4)

High-throughput writes of wide-column data (2)

Warehousing structured data (3)

Develop Big Data algorithms interactively in Python (1)

1. Datalab

2. BigTable

3. BigQuery

4. Datastore



cloud.google.com