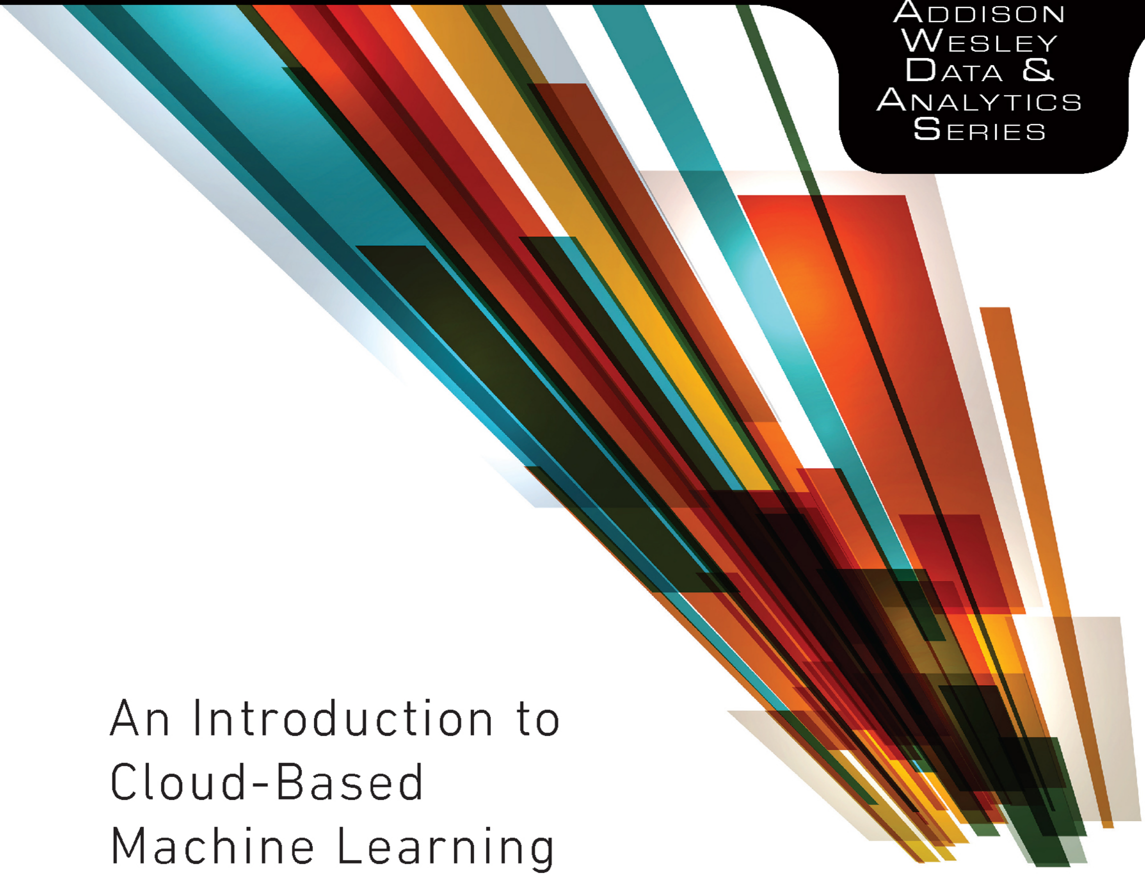
Addison Wesley Data & Analytics Series



An Introduction to Cloud-Based Machine Learning

Pragmatic AI

NOAH GIFT

Praise for *Pragmatic AI*

"[This is] a sweeping guide that bridges the gap between the promise of AI and solutions to the gritty problems that must be solved to deploy real-world projects. Clear and usable, *Pragmatic* AI covers much more than just Python and AI algorithms."

—Christopher Brousseau, founder and CEO of Surface Owl, the Enterprise AI platform

"A fantastic addition for any technology enthusiast! There is so much you can say about this book! Noah Gift really made this a practical guide for anyone involved with machine learning in the industry. Not only does it explain how one can apply machine learning on large data sets, it provides a valuable perspective on technology feedback loops. This book will benefit many data science and development teams so they can create and maintain their applications efficiently right from the beginning."

—Nivas Durairaj, technical account manager, AWS (Certified Professional Architect AWS)

"A great read if you want insights into building production-quality ML pipelines and tools that truly help your data engineering, data science, or data DevOps team. Even experienced developers often find themselves spinning their wheels on low-productivity tasks. Oftentimes, software books and university classes don't explain the steps needed to go into production. Noah has a gift for finding pragmatic approaches to software deployments that can really accelerate the development and delivery process. He has a focus and passion for enabling rapid software solutions that is very unique.

"The key to building production-quality ML pipelines is automation. Tasks and steps, which engineers may do manually during the research or prototype phase, must be automated and scaled in order to create a production system. This book is full of practical and fun exercises that will help any Python developer automate and extend their pipelines into the cloud.

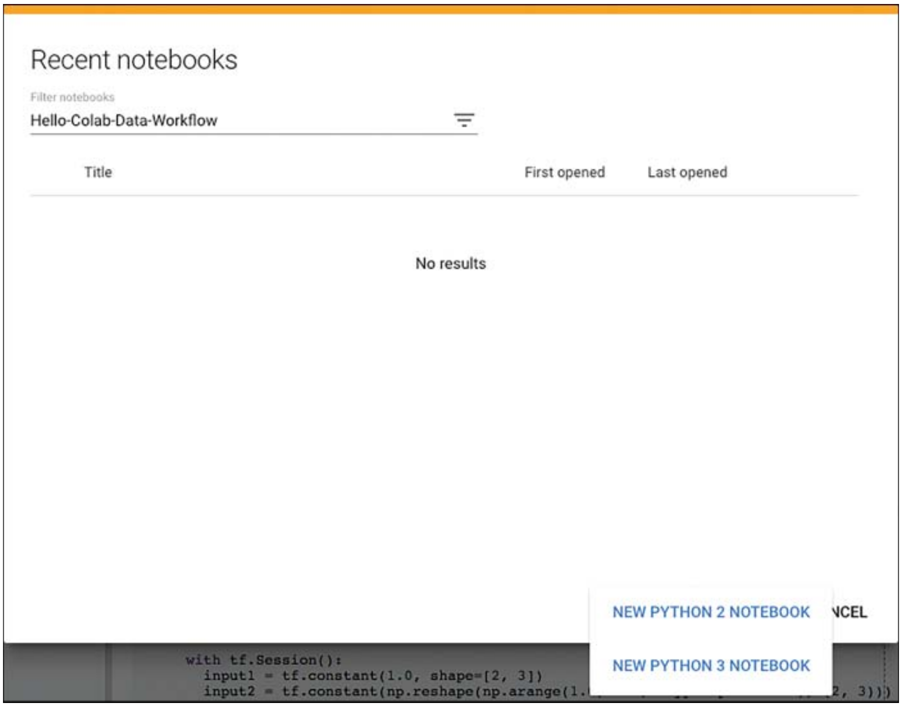
"I'm currently working with big data, ML pipelines, Python, AWS, Google cloud, and Azure at Roofstock.com, an online real estate company. Our analytics database is approaching 500 million rows! Within this book I found many practical tips and exercises that will immediately improve my own productivity. Recommended!"

—Michael Vierling, lead engineer, Roofstock

- Support for both Python 2 and 3
- Ability to upload notebooks
- Notebooks are stored in Google Drive and can piggyback on the same sharing features that make Google Drive documents easy to work with
- Ability for two users to edit the notebook at the same time

One of the more fascinating parts of Colaboratory is the ability to create a shared training lab for Jupyter Notebook–based projects. Several incredibly tricky problems are solved right away like sharing, dealing with data sets, and having libraries installed. An additional use case would be to programmatically create Colaboratory notebooks that have hooks into the AI pipeline of your organization. A directory of notebooks could be prepopulated with access to a BigQuery query or an ML model that is a template for a future project.

Here is a hello-world workflow. You can refer to this notebook in GitHub: https://github.com/noahgift/pragmaticai-gcp/blob/master/notebooks/dataflow_sheets_to_pandas.ipynb. In Figure 4.1, a new notebook is created.



```
with tf.Session():
  input1 = tf.constant(1.0, shape=[2, 3])
  input2 = tf.constant(np.reshape(np.arange(1.
```

Figure 4.1 Colaboratory Notebook Creation

Next, the gspread library is installed.

```
!pip install --upgrade -q gspread
```

Authentication is necessary to write to a spreadsheet as shown, and a "gc" object is created as a result.

```
from google.colab import auth
auth.authenticate_user()

import gspread
from oauth2client.client import GoogleCredentials

gc = gspread.authorize(GoogleCredentials.get_application_default())
```

The gc object is now used to create a spreadsheet that has one column populated with values from 1 to 10.

```
sh = gc.create('pramaticai-test')
worksheet = gc.open('pramaticai-test').sheet1
cell_list = worksheet.range('A1:A10')

import random
count = 0
for cell in cell_list:
  count +=1
  cell.value = count
worksheet.update_cells(cell_list)
```

Finally, the spreadsheet is converted to a Pandas DataFrame:

```
worksheet = gc.open('pramaticai-test').sheet1
rows = worksheet.get_all_values()
import pandas as pd
df = pd.DataFrame.from_records(rows)
```

Datalab

The next stop on the tour of GCP is Datalab (https://cloud.google.com/datalab/docs/quickstart). The entire gcloud ecosystem requires the software development kit (SDK) be installed from https://cloud.google.com/sdk/downloads. Another option to install is to use the terminal as follows.

```
curl https://sdk.cloud.google.com | bash
exec -l $SHELL
gcloud init
gcloud components install datalab
```

Once the gcloud environment is initialized, a Datalab instance can be launched. There are a few interesting things to note. Docker is an incredible technology for running isolated versions of Linux that will run the same on your laptop as it will in a data center, or some other laptop of a future collaborator.

Extending Datalab with Docker and Google Container Registry

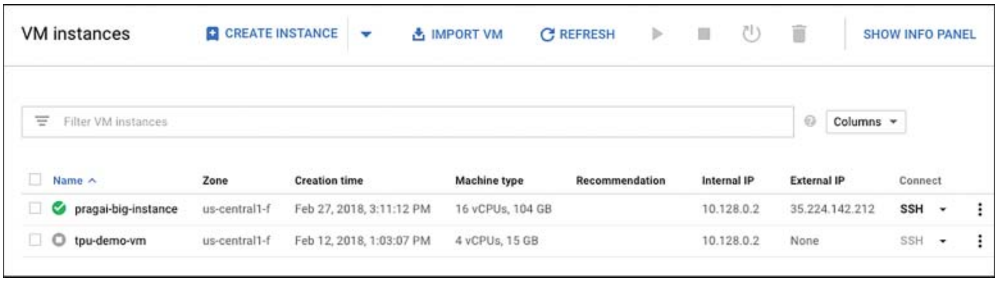
You can run Datalab locally in Docker by referring to the getting-started guide here: https://github.com/googledatalab/datalab/wiki/Getting-Started. Just having a local, friendly version of Datalab you can run free of charge is useful, but what is even more powerful is the ability to extend this Datalab base image, store it in your own Google Container Registry, then launch it with a more powerful instance than your workstation or laptop, like an n1-highmem-32, for example, which has 16 cores and 104GB of memory.

All of a sudden, problems that were just not addressable from your local laptop become pretty straightforward. The workflow to extend the Datalab Docker core image is covered in the guide mentioned in this section. The main takeaway is that after cloning the repository, a change will need to be made to Dockerfile.in.

Launching Powerful Machines with Datalab

To launch one of these mega instances of Jupyter Notebook, it looks like this, as seen in Figure 4.2.

```
➜  pragmaticai-gcp git:(master) datalab create\
 --machine-type n1-highmem-16 pragai-big-instance
Creating the instance pragai-big-instance
Created [https://www.googleapis.com/compute/v1
/projects/cloudai-194723/zones/us-central1-f/
instances/pragai-big-instance].
Connecting to pragai-big-instance.
This will create an SSH tunnel and may prompt you
to create an rsa key pair. To manage these keys, see
https://cloud.google.com/compute/docs/instances/\
adding-removing-ssh-keys
Waiting for Datalab to be reachable at http://localhost:8081/
Updating project ssh metadata...-
```

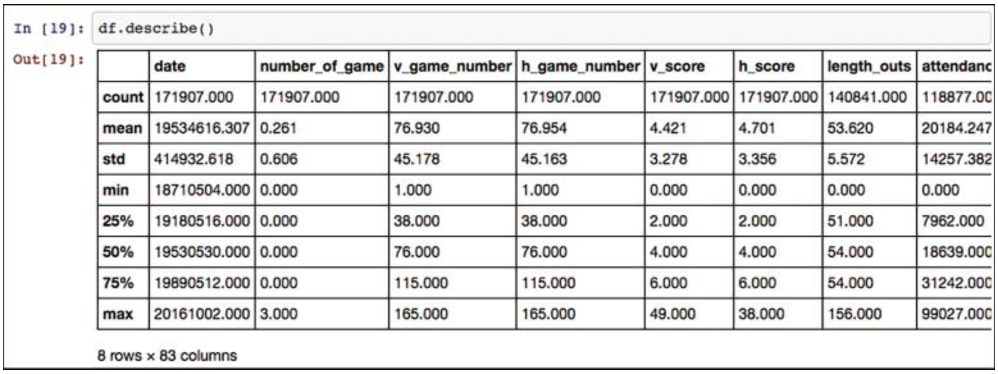


Name	Zone	Creation time	Machine type	Recommendation	Internal IP	External IP	Connect
pragai-big-instance	us-central1-f	Feb 27, 2018, 3:11:12 PM	16 vCPUs, 104 GB		10.128.0.2	35.224.142.212	SSH
tpu-demo-vm	us-central1-f	Feb 12, 2018, 1:03:07 PM	4 vCPUs, 15 GB		10.128.0.2	None	SSH

Figure 4.2 Datalab Instance Running in GCP Console

To make the instance do something useful, Major League Baseball game logs from 1871 to 2016 that were found at data.world (https://data.world/dataquest/mlb-game-logs) are synced to a bucket in GCP. Figure 4.3 shows that 171,000 rows were loaded into Pandas DataFrame via the `describe` command.

```
gsutil cp game_logs.csv gs://pragai-datalab-test
Copying file://game_logs.csv [Content-Type=text/csv]...
- [1 files][129.8 MiB/129.8 MiB]
 628.9 KiB/s
Operation completed over 1 objects/129.8 MiB.
```



```
In [19]: df.describe()
Out[19]:
```

	date	number_of_game	v_game_number	h_game_number	v_score	h_score	length_outs	attendanc
count	171907.000	171907.000	171907.000	171907.000	171907.000	171907.000	140841.000	118877.00
mean	19534616.307	0.261	76.930	76.954	4.421	4.701	53.620	20184.247
std	414932.618	0.606	45.178	45.163	3.278	3.356	5.572	14257.382
min	18710504.000	0.000	1.000	1.000	0.000	0.000	0.000	0.000
25%	19180516.000	0.000	38.000	38.000	2.000	2.000	51.000	7962.000
50%	19530530.000	0.000	76.000	76.000	4.000	4.000	54.000	18639.00
75%	19890512.000	0.000	115.000	115.000	6.000	6.000	54.000	31242.00
max	20161002.000	3.000	165.000	165.000	49.000	38.000	156.000	99027.00

8 rows × 83 columns

Figure 4.3 171,000 Rows in DataFrame from GCP Bucket

The entire notebook can be found in GitHub (https://github.com/noahgift/pragmaticai-gcp/blob/master/notebooks/pragai-big-instance.ipynb), and the commands are as follows. First, some imports.

```
Import pandas as pd
pd.set_option('display.float_format', lambda x: '%.3f' % x)
import seaborn as sns
from io import BytesIO
```

Next, a special Datalab command assigns the output to a variable called `game_logs`.

```
%gcs read --object gs://pragai-datalab-test/game_logs.csv\
          --variable game_logs
```

A new DataFrame is now created.

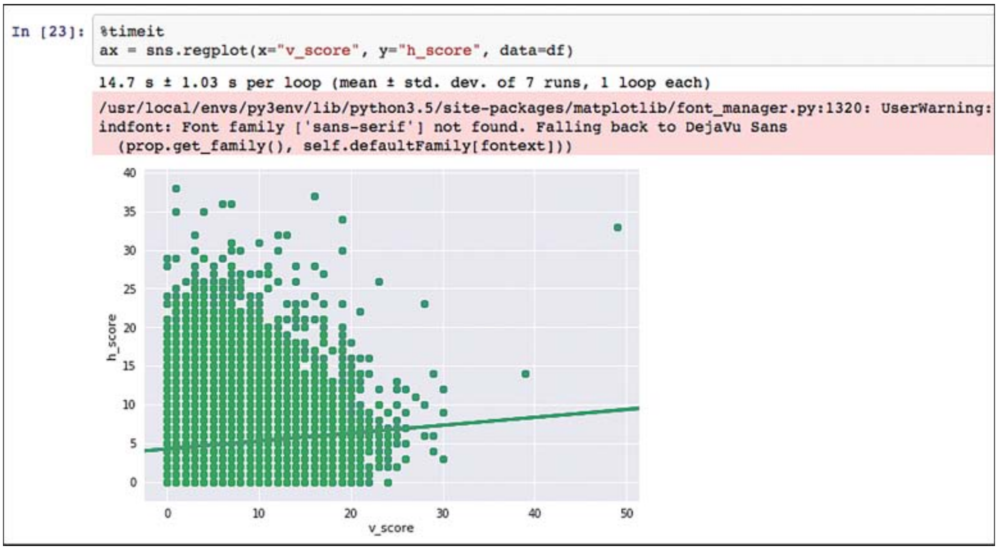
```
df = pd.read_csv(BytesIO(game_logs))
```

Finally, the DataFrame is plotted as shown in Figure 4.4.

```
%timeit
ax = sns.regplot(x="v_score", y="h_score", data=df)
```

The takeaway from this exercise is that using Datalab to spin up ridiculously powerful machines to do exploratory data analysis (EDA) is a good idea. Billing is by the second, and it can save hours to do heavy crunching and EDA. Another takeaway is that GCP is leading this section of cloud services versus AWS because the developer experience does a quite nice job in moving large sets of data into a notebook and using familiar "small data" tools like Seaborn and Pandas.

Another takeaway is that Datalab is a great foundation for building production ML pipelines. GCP buckets can be explored, BigQuery has straightforward integration, and other parts of the GCP ecosystem can be integrated like ML Engine, TPUs, and the container registry.



```
In [23]: %timeit
         ax = sns.regplot(x="v_score", y="h_score", data=df)

         14.7 s ± 1.03 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
         /usr/local/envs/py3env/lib/python3.5/site-packages/matplotlib/font_manager.py:1320: UserWarning:
         indfont: Font family ['sans-serif'] not found. Falling back to DejaVu Sans
           (prop.get_family(), self.defaultFamily[fontext]))
```

Figure 4.4 Timed Seaborn Plot Takes 17 seconds to Plot 171,000 Rows on 32 Core Machine with 100GB of Memory

BigQuery

BigQuery is one of crown jewels of the GCP ecosystem and an excellent service around which to build pragmatic production ML and AI pipelines. In many ways, this too has some developer usability advantages over AWS. While AWS is stronger on complete end-to-end solutions, GCP seems to cater to the tools and paradigms developers are already used to. It is trivial to move data in and out in a variety of different ways, from the command line on a local machine to GCP buckets to API calls.

Moving Data into BigQuery from the Command Line

Speaking of moving data into BigQuery, a straightforward way to do this is to use the bq command-line tool. Here is the recommended approach.

First, check whether the default project has any existing data sets.

```
➜  pragmaticai-gcp git:(master) bq ls
```

In this case, there isn't an existing data set in the default project, so a new data set will be created.

```
➜  pragmaticai-gcp git:(master) bq mk gamelogs
Dataset 'cloudai:gamelogs' successfully created.
```

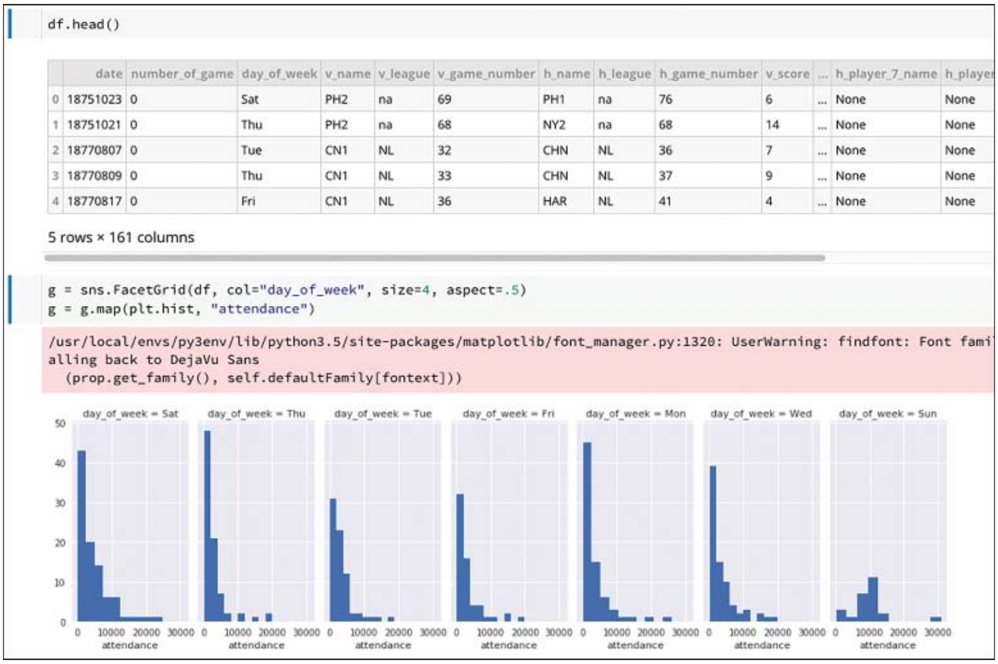
Next, bq ls is used to verify the data set was created.

```
➜  pragmaticai-gcp git:(master) bq ls
  datasetId
 -----------
  gamelogs
```

Next, a local CSV file that is 134MB and has 171,000 records is uploaded with the flag `-autodetect`. This flag enables a lazier way to upload data sets with many columns since you don't have to define the schema up front.

```
➜  pragmaticai-gcp git:(master) ✗ bq load\
 --autodetect gamelogs.records game_logs.csv
Upload complete.
Waiting on bqjob_r3f28bca3b4c7599e_00000161daddc035_1
        ... (86s) Current status: DONE
➜  pragmaticai-gcp git:(master) ✗ du -sh game_logs.csv
134M    game_logs.csv
```

Now that the data has been loaded, it can be queried easily from Datalab as shown in Figure 4.5.



```
df.head()
```

	date	number_of_game	day_of_week	v_name	v_league	v_game_number	h_name	h_league	h_game_number	v_score	...	h_player_7_name	h_player
0	18751023	0	Sat	PH2	na	69	PH1	na	76	6	...	None	None
1	18751021	0	Thu	PH2	na	68	NY2	na	68	14	...	None	None
2	18770807	0	Tue	CN1	NL	32	CHN	NL	36	7	...	None	None
3	18770809	0	Thu	CN1	NL	33	CHN	NL	37	9	...	None	None
4	18770817	0	Fri	CN1	NL	36	HAR	NL	41	4	...	None	None

5 rows × 161 columns

```
g = sns.FacetGrid(df, col="day_of_week", size=4, aspect=.5)
g = g.map(plt.hist, "attendance")
```

```
/usr/local/envs/py3env/lib/python3.5/site-packages/matplotlib/font_manager.py:1320: UserWarning: findfont: Font fami
alling back to DejaVu Sans
  (prop.get_family(), self.defaultFamily[fontext]))
```

Figure 4.5 BigQuery to Pandas to Seaborn Pipeline

First, imports are created. Note the google.datalab.bigquery import, which allows for easy access to BigQuery.

```
import pandas as pd
import google.datalab.bigquery as bq
pd.set_option('display.float_format', lambda x: '%.3f' % x)
import seaborn as sns
from io import BytesIO
```