Data Science in Production

Production Ready Data Science
The 5 Biggest Challenges Companies Face when Becoming DataDriven.

How to Start a Data Science Project in Python
The Three Building Blocks for Every Data Science Project.

First Look at Facebook’s Prophet: Forecasting Stores Transactions
How Will This Open-Source Project Compete With Hand-Crafted Models?
Many organisations develop successful proof of concepts but then don’t manage to materialize the models beyond their laptops. Taking models into production requires a professional workflow, high quality standards, and scalable code and infrastructure. This magazine is dedicated to reaping benefit from data by taking data driven applications into production.

In this first issue of "Data Science in Production", Giovanni Lanzani discusses how organizations should avoid the Kaggle-curse in their journey to swiftly reach the "production" milestone. Henk Griffioen discusses the first step to get there: how to structure a (Python) project. Rodrigo Agundez deep dives into recently open-sourced Facebook Prophet, comparing it with his very own hand-crafted models (you will not believe what he found!).

At GoDataDriven, we firmly believe in open source. We use and love a variety of different libraries, tools, and programming languages that are open. Because we believe that the wave of open source software in the data landscape enables teams to work faster and be more productive.

Open source is also about giving back and that is what we try to do as much as we can: from the various Meetups we organize to the PyData Amsterdam conference. There is something, that is rarely emphasized: we contribute actively to the open source projects we love. In this first edition, we highlighted a few of our contributions to the open source world!

Hopefully, after reading this magazine, you are one step closer to becoming a Data Driven organization. Enjoy reading the articles and as always, we appreciate your feedback!
Production Ready Data Science

Becoming a data-driven organization is not an easy task. Off the top of my head, and in no particular order, these are the most frequent challenges a company faces: 1. Attracting, retaining, and training the right talents; 2. Collecting and making data available cross-silos; 3. Modernize their tech stack or increase the complexity of the IT landscape by adding new technologies; 4. Fear of the unknown i.e. many people are afraid about losing control, or their job, to data and data science; 5. Lack of vision\(^1\).

As I was thinking about this list, however, I felt there was something deeper about the troubles some organizations are facing. I know in fact about companies that have made significant progress in all five points, but that are still not reaping the fruits they were expecting. When I looked at these companies more closely, they were all not putting the models they developed into production. The reasons varied, from being content with report-driven decision making (either a one-off report, or periodic reporting) to simply struggling with all the pieces of the puzzle.

Being the curious type, I set out to investigate what was making the puzzle so difficult for them. Productionizing a model involves a series of (moving) pieces:

1. The data should be automatically available for the model, i.e. there should be no human intervention in the ETL phase;
2. The model should also run automatically, follow the DRY principles, should be battle-tested, and possibly be flexible in its sources/sinks, either by being used as a library, or by exposing accessible APIs (REST being one the most in vogue today);
3. Refreshing and/or re-training the model should not impact the front-end accessing it, or, in the easiest cases, should not impact it during business hours\(^3\).

The companies struggling with becoming data driven, are failing in one or more of the above points. What they are doing is a mix of the following:

1. They copy the data around manually; this makes it basically impossible to bring a model into production because, if the data does not flow automatically, the owners of the data (wherever they might be) are probably not even aware that their data is used in a model. If the data comes in automatically, instead, they know that systems are depends on them; on top of that you should get almost immediate alerts if the ingestion fails (if not: set it up);
2. They don’t test the code so software engineers basically refuse to touch the code; this problem is exacerbated when the engineers have to re-write the code because of performance or other reasons; with tests unavailable, modifying the model becomes daunting;
3. Related to the previous point, many data scientists are either ex data analysts that thought their job was more secure by changing their business title, or are coming from disciplines like Physics, Math, or Statistics, often with research experience. Coming myself from four years of research in Physics, I can attest that, except some unicorns, we (used to) write unreadable or very complicated code\(^5\). When PhDs end up doing software development, they quickly pick up the good practices. But in data science, exploration driven modelling can worsen the situation: you start poking around the data until it suddenly makes sense but you leave around all the steps you took, even the unused ones; coupling this to a lack of documentation, you can easily end up with thousands of lines of code that are basically acting as a scarecrow for your engineers;

---

1. A lack of vision is a much broader issue than 1-4 as it can bring even the largest and most flourishing corporations to the ground (a great read about this is *Good to Great*). I included it nonetheless as it will cut or make budget unavailable or prevent management buy-in of data-driven products. And lack of management buy-in is even worse that lack of budget. One of our first clients installed its first Hadoop cluster on dismissed machines, built a type-ahead and recommendation engine for their web shop, and see profits surge right after they put it into production. There was nothing a budget could do had management not agreed about “letting” the model into production.
2. Unless something breaks of course.
3. Whatever that means for you.
4. This is probably one of the post with the highest density of bullet points I’ve ever written. Apologies.
5. I still vividly remember when a professor suggested that using kk as a variable name was not a very wise choice, to which I replied that I was using k and kk for something else.
• Sometimes your data scientists code in a language that doesn’t nicely operate with the outside world; if you’re in charge, make this stop as there are no solutions other than complete lock-in;

• Many data scientists approach the problem at hand with a Kaggle-like mentality: delivering the best model in absolute terms, no matter what the practical implications are. It’s not the best model that we implement, but the one that combines quality and practicality. Take the Netflix (http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html) competition for example: the company made 1 million dollars available to the group that would improve its recommendation engine; the winning team found a combination of algorithms improving Netflix one by 8.43%. Netflix however never implemented it, as the method was built to handle 100 million ratings, much less than the 5 billion that Netflix had! Moreover, the algorithms were not built to adapt as members added more ratings. I am quoting here but think about it for a second: the winner reported more than 2000 hours of work to come up with the final combination of 107 algorithms that gave them this prize. They gave Netflix the source code. And yet they did not think how the algorithms were going to be used, that is daily updated as new users were rating additional movies. 2000 hours of work!

If you’ve paid attention to these points, you probably start seeing a pattern: data scientists usually suck at software quality, that is: reliability, usability, efficiency, portability, and maintainability. Because data-driven models are implemented through software, they suffer from bad software quality just as much as your typical application.

Let me be clear: this is not an easy task! To create a (great) model you need creativity, a scientific attitude, knowledge of various modeling techniques, etc. Getting data scientists able to create these models is one of the biggest challenges for an organization. But focusing on the modeling at the cost of software quality will produce something great and admirable that ends up not being used.

I imagine you now have the next burning question which is: what if the data scientists working at my organization are not good at it? What if someone left the company, implemented a great new method, but nobody can make sense of what this data scientist wrote?  

6 It is not my intention to denigrate their work. I often use the matrix factorization methods implemented in Spark to train my recommendation engines. I am merely stating that they set out to solve a problem without thinking about productionizing their work.

7 This is a subset of the ISO 9126 standard on software quality.

Training Calendar

“Never let school get in the way of your education!”

Vincent Warmerdam
Data Scientist

Download our complete training brochures here: https://godatadriven.com/training-brochure

May
01 Cloudera Data Analyst / Amsterdam
08 Spark Programming / Amsterdam
18 Data Science Accelerator Program / Amsterdam
22 Data Science with Spark / Amsterdam

June
06 From Excel to R / Amsterdam
08 Workshop Signal Processing for Data Science / Amsterdam
13 Cloudera Data Science at Scale using Spark and Hadoop / Amsterdam
19 Neo4J Masterclass / Amsterdam
21 Data Science with R / Amsterdam
26 Data Science with Python / Amsterdam

July
03 Cloudera Developer for Spark and Hadoop I / Amsterdam
How to Start a Data Science Project in Python

Many articles and tutorials online show complicated machine learning methods and cutting edge technologies, putting Data Scientists around the world in a constant state of fear of missing out. But what are the fundamentals? How should you structure your project? What is the minimum set of tools you need? This article gives a few pointers for setting up your projects so that you will reach Product Ready Data Science as described in the previous article as soon as possible.

- Your analyses should be reproducible and your structure should enable that.
- A project starts from raw data that should never be edited; consider raw data immutable and only edit derived sources.

I couldn’t help to invent my own project structure and my minimal structure looks something like this:

```
example_project/
  ├── data/        <-- The original, immutable data dump.
  │   └── __init__.py  <-- Make the folder a package.
  ├── figures/     <-- Figures saved by notebooks and scripts.
  │   └── process.py  <-- Example module.
  ├── notebooks/   <-- Jupyter notebooks.
  │   └── test_process.py  <-- Tests for process.py.
  ├── output/      <-- Processed data, models, logs, etc.
  │   └── exampleproject/  <-- Python package with source code.
  │       └── __init__.py  <-- Make the folder a package.
  ├── tests/       <-- Tests for your Python package.
  │   └── test_process.py  <-- Tests for process.py.
  └── environment.yml  <-- Virtual environment definition.
  └── README.md    <-- README with info of the project.
      └── setup.py  <-- Install and distribute your module.
```

You can find an example [here](https://github.com/hgrif/example-project).

It mostly follows the other structures:
- raw data is immutable and goes to data/;
- processed data and derived output goes to different folders such as figures/ and output/;
- notebooks go to notebooks/;
- project info goes in the README.md;
- and the project code goes to a separate folder.

I try to make a full-fledged Python package (plus tests) out of my project structure so that the step between prototyping and productionizing is as small as possible. The setup.py allows me to install the package in a virtual environment and use it in my notebooks (more on this in a later blog post).

It doesn’t really matter which structure you pick, as long as it fits your workflow and you stick with it for a while. Try to understand the philosophies of the projects and pick the structure that suits your needs.
Virtual Environment

Projects should be independent of each other: you don’t want your new experiments to mess up your older work. We do this partly by putting the files of different projects in different folders but you should also use separate Python environments.

Virtual environments are isolated environments that separate dependencies of different projects and avoid package conflicts. Each virtual environment has its own packages and its own package versions. Environment A can have numpy version 1.11 and pandas version 0.18 while environment B only has pandas version 0.17. I like conda virtual environments because they’re well suited for Data Science (read why here: https://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/).

Create a new conda virtual environment called example_project with Python 3.5:

$ conda install --name example_project python=3.5

Make sure your virtual environment is activated (leave out the source if you’re on Windows):

$ source activate example_project

... and you’re now ready to install your favourite packages!

$ conda install pandas numpy jupyter scikit-learn

When you’re switching to a different project, run a source deactivate and activate the project’s virtual environment.

Once you get the hang of the activate-deactivate-flow, you’ll find that a virtual environment is a lightweight tool to keep your Python environments separated. By exporting your environment definition file (i.e. all installed packages and their versions) your projects will also be easily reproducible. If you want a more detailed discussion, check Tim Hopper’s post (http://tdhopper.com/blog/2015/Nov/24/my-python-environment-workflow-with-conda/).

Tooling

You can get away of some of the repetitive tasks by using some tooling!

The Python package cookiecutter automatically creates project folders based on a template. You can use existing template such as the Cookiecutter Data Science (https://github.com/drivendata/cookiecutter-data-science) or mine (https://github.com/hgrif/cookiecutter-ds-python/tree/master/%7B%7C cookiecutter.repo_name%20%7D) or invent your own.

The easiest way to use virtual environments is to use an editor like PyCharm (https://www.jetbrains.com/pycharm/) that supports them. You can also use autoenv (https://github.com/kennethreitz/autoenv) or direnv (https://direnv.net/) to activate a virtual environment and set environment variables if you cd into a directory.

Add a .gitignore to your project directory so that you don’t accidentally add figures or data to your repository. I generally start with a .gitignore for Python (https://github.com/github/gitignore/blob/master/Python.gitignore) and add the folders data/ figures/ and output/ so that Git ignores these folders.

Now that Git is set up, you can git add and git commit to your heart’s content!

Git

Every project should have its own Git repository. Having a repo per project allows you to track the history of a project and maintain complex version dependencies between projects.

Alternatively, you can choose to have one repository with multiple projects, putting all the knowledge in a single place.

The downside is, however, that it often ends up with ugly merge conflicts: Data Scientists are generally not that fluent with Git. In addition to a lot of Git frustrations, it makes your projects less independent of each other.

The easiest way to set up Git is by creating a new git repository on your Git host (e.g. GitHub or GitLab) and cloning that:

$ git clone https://github.com/hgrif/example-project.git

You can then setup your project structure in this empty folder.

If you followed this guide and already created a folder with some files, first initialize a git repository on your machine:

$ git init

Then create a new git repository on your host, get its link and run:

$ git remote add origin
https://github.com/hgrif/example-project.git

This adds the remote repository with the link https://github.com/hgrif/example-project.git and names it origin ou probably should push your current master branch to origin:

$ git push --set-upstream origin master

Add a .gitignore to your project directory so that you don’t accidentally add figures or data to your repository. I generally start with a .gitignore for Python (https://github.com/github/gitignore/blob/master/Python.gitignore) and add the folders data/ figures/ and output/ so that Git ignores these folders.

Now that Git is set up, you can git add and git commit to your heart’s content!

Tooling

You can get away of some of the repetitive tasks by using some tooling!

The Python package cookiecutter automatically creates project folders based on a template. You can use existing template such as the Cookiecutter Data Science (https://github.com/drivendata/cookiecutter-data-science) or mine (https://github.com/hgrif/cookiecutter-ds-python/tree/master/%7B%7C cookiecutter.repo_name%20%7D) or invent your own.

The easiest way to use virtual environments is to use an editor like PyCharm (https://www.jetbrains.com/pycharm/) that supports them. You can also use autoenv (https://github.com/kennethreitz/autoenv) or direnv (https://direnv.net/) to activate a virtual environment and set environment variables if you cd into a directory.

Conclusion

Having a good setup for your Data Science projects makes it easier for other people to work on your projects and makes them more reproducible. A good structure, a virtual environment and a git repository are the building blocks for every Data Science project. ■
From June 6 - 9 the Dutch Data Science Week will take place across The Netherlands with the purpose to provide Data Scientists and organizations with a platform for innovation, to learn, inspire and network. Expect a wide range of activities, including conferences, hackathons, training classes, workshops and meetups.

- Grow your Data Science skills with a training class or workshop
- Hack away during one of the challenges
- Meet interesting organizations at the career café
- Be inspired during a meetup or conference

The Dutch Data Science Week is a joint initiative of SAS Netherlands and GoDataDriven in close collaboration with various partners and supporters.

www.dutchdatascienceweek.nl
Open Source Contributions

At GoDataDriven we have an Open Source First approach. Unless some shockingly good reasons exist, we always advise to use (and implement) open source solutions.

It is therefore only natural that we have the tendency to give back to the open source community. Some of these efforts have a good visibility (such as Divolte, www.divolte.io, our very own clickstream collector), while others remain in the shadow.

On a regular basis, we contribute to various open source projects, both old and new. To name a few: Fokko Driesprong contributed to 4 different projects: Druid, Docker-Druid, Airflow, and Flink.

- In Druid PR 3481 he fixed the INFO message logs and updated the documentation with PR 3973, regarding ingesting parquet format into a Druid cluster. Furthermore, he also fixed a faulty log line with PR 3970;
- In Docker-Druid PR 30 he expanded on the logging section in the README and fixed two issues in PR 33 and 34 by creating and setting correct directories and permissions;
- In Airflow PR 2038 he resolved session leakage;
- In Airflow PR 2042 (still open) he extended the spark-submit operator/hook by adding YARN integration;
- In Flink PR 3077 he implemented Stochastic Outlier Selection (!);
- In Flink PR 3081 he cleaned up the Flink Machine Learning library (!!);
- In Flink PR 3280 he fixed the documentation by setting a correct reference;
- In Airflow PR 2042 he further extended the spark-submit operator/hook by adding YARN integration;
Vincent Warmerdam created a whole new project called Kadro (https://github.com/koaning/kadro) which is a friendly pandas wrapper with a more composable grammar support. The goal of the library is to have a minimal wrapper that allows most of all dataframe operations to be more expressive by being chainable.

Bas Harenslak contributed to Prometheus JMX exporter PR 108 (still open) to add HBase example config.

Finally, Giovanni Lanzani contributed to NiFi PR 1467 (still open) to add the exception class to failing flow files coming out of InvokeHTTP. He also fixed a wrong description of the UnpackContent processor in NiFi PR 1558.

One project to highlight here is the open sourcing of a project to provision Google Cloud Engine instances to ease the classroom trainings deployment. At GoDataDriven, we are often faced with a lot of challenges when delivering training where Spark is involved:

• If we use virtual machines (VM), the users can’t never quite experience how powerful Spark is, as their machines are always so slow that it’s not even funny. As an added chore, we need to create, maintain, and distribute several GBs around as these VMs are not small;

• If we use local mode, installing Spark in all configurations is incredibly cumbersome, especially if you want HDFS support; the slowness still applies, albeit in a less severe form;

• If you create a cluster, it’s never nice to deploy it, install the packages, and make the keys available to everybody.

Since Google Cloud Engine makes it extremely easy to create clusters, the project kind of assumes that that’s what you’re using. That said, it should be easy enough to modify it. Giovanni is also working on getting Anaconda + JupyterHub integrated so that users don’t even need to have SSH access to the machine.

The workshop is suitable for basically any data scientist that is dealing with diverse data, because not only the amount of data is ever-increasing, but also the diversity. Think about speech data for speech recognition purposes, or time series in general. Signals from households and image data. We will see how you can do speech recognition, for example how Siri works. And we will answer the question “How you can do image recognition or image manipulation with deep convolutional networks?”

Author Ivo Everts

What Are Signals?

A signal is a function that takes a certain value at a certain time or a certain location in the case of images. That means that any mathematical operator can be applied to the function in any of the domains. Nowadays you have deep learning and convolutional neural networks, with which a lot of problems have been solved and the general performance has increased quite a lot. But still it is important to understand exactly what is happening inside the models and in the convolutional layers, so that you can apply it in the right way in your setting. Also, if you don’t have a lot of data, then deep learning usually doesn’t work so well. It is important to understand signal processing techniques, so that if you don’t have a lot of data, you can engineer the features yourself.

Program

The workshop starts with an overview of possible applications, in the domains of time series, speech recognition and image recognition. We will then cover some fundamental concepts of signal processing. Most notably we will cover the convolution so you will understand exactly what convolution is and how you can use it. Also, we will talk about Fourier analysis so that we can decompose a signal into its frequencies, and harmonics. Different sine and cosine components of a signal. Then we will apply those techniques to classify time series data. In this case, we have some time series of decease outbreaks in the US. We will also apply the techniques to extract features from speech data, so that we can identify a speaker, his age or gender. Finally, we will also apply the signal processing techniques and deep convolutional neural networks to image data for visual recognition. We will cover quite some theory of signal processing in the first part. In the second part, we will show a lot of practical examples in Python on how you can use those methods and hand it out afterwards so you can apply it yourself.
First Look at Facebook’s Prophet: Forecasting Stores Transactions

Recently Facebook’s Core Data Science team released an open source forecasting API (https://github.com/facebookincubator/prophet). I decided to give it a try during one of our famous GDD Fridays.

Using Prophet to Predict Product Sales

My goal is to see how Prophet’s forecasts behave using the same data we use in one of the models developed by Rogier van der Geer and me. The data belongs to a customer for which models are already in production, therefore I won’t be disclosing any details of it. The forecast of the number of transactions in a shop is used as a part of an ensemble to predict products sales. Since Prophet does not accept features, it would be unfair to make a comparison at that level since, for example, price is a very important factor.

Data: Transactions and Holidays

Our models make forecasts for different shops of this company. I took 2 shops, one which contains the easiest transactions to predict from all shops, and another with a somewhat more complicated history. The data consists of real transactions since 2014. Data is daily with the target being the number of transactions executed during a day. There are missing dates in the data when the shop closed, for example New Year’s Day and Christmas. The holidays provided to the API are the same I use in our model. They contain data from school vacations or large periods, to bank holidays like Christmas Eve. In total, the data contains 46 different holidays.

Code

If the data is in a nice format (this is a big if), Prophet provides a very easy to use API. Once I cleaned, aggregated and dumped the data, the calculation consisted of these two pieces of code:

```python
def predict(tseries, predict_date, holidays=None):
    model = Prophet(holidays=holidays)
    # train on data until 3 days before
    model.fit(tseries[tseries.ds < (predict_date - timedelta(days=2))])
    forecast = model.predict(model.make_future_dataframe(periods=5))
    return forecast.loc[forecast.ds == predict_date, ['ds', 'yhat']]

pred = []
pred_holidays = []
for date in pd.date_range('2016-1-1', '2016-12-31'):
    pred.append(predict(tseries_shop, date))
    pred_holidays.append(predict(tseries_shop, date, holidays))

predictions = pd.merge(pd.concat(pred), pd.concat(pred_holidays),
                        on='ds', how='inner', suffixes=('','_hol'))
```

Prophet offers a R and Python API, I used the Python API of course.
The forecast is done for 2016 with and without holiday data. Our production model gets trained daily via an Airflow (https://airflow.incubator.apache.org/) job. To make a fair comparison, I train a Prophet model for each date in 2016 using the data until 3 days before the date to be forecast. This is because the order for a product needs to be submitted 2 days before, which means it uses the data available until then.

Prophet leveraged the full capacity of my laptop using all 8 cores. The calculation took around 45 minutes per shop, which means a single day with or without holidays takes around 4 seconds.

**Metric**

The metric I used to measure the forecast performance is the coefficient of determination (https://en.wikipedia.org/wiki/Coefficient_of_determination) (R2 score). The R2 score gives the proportion of the variance in the data that is explained by the forecast. A perfect forecast will give 1.0 and a constant prediction for every day will give 0.0.

**Easy shop: Widushop**

Using Vincent Warmerdam’s awesome Pokemon name generator (http://tnaas.com/), I will call this shop Widushop. This is the transaction data for the 3 years.

Prophet produces a very accurate forecast, it scores 0.89 without using holidays and 0.94 using holidays. Below I show a comparison between the transactions (truth) and the forecast using holidays.

Pretty nice! Overall it produces very good results, for holidays seems to overestimate (look at Christmas Eve), nevertheless that can be tuned by the parameter holidays.prior.scale as stated in the documentation (https://facebookincubator.github.io/prophet/docs/holiday_effects.html).

**Difficult shop: Qumashop**

This time the shop name generated is Qumashop. The transaction history of Qumashop is more chaotic than the one for Widushop. Below I show the transaction history of Qumashop.

Holidays have a much greater impact. Look at that peak in the middle of July, this is a known event that draws a lot of people to the city (it is in the holidays data). Notice that transactions in 2016 are considerably higher than other years, especially from July until September.
Not catching this upward trend would mean losing a lot of potential sales.

This time the Prophet forecast is not as good as for Widushop giving 0.64 without holiday data and a solid 0.82 using holidays. Below I show a comparison between the transactions (truth) and the forecast using holidays for Qumashop.

Look at that, very nice. I am especially happy that it caught the mentioned trend between July and September. Moreover, the residuals on the week following the big peak in July, the second week in September and the two weeks at the end of October are too high. Remember that in practice this is just a model of an ensemble, is better to have a little overall bigger residual that can reduced by other models, than having weeks with such big errors. Perhaps the forecasts for the week after the big peak in July can improve by introducing a changepoint the last day of the peak holiday week.

Wrap-up
Prophet’s out of the box results were impressive. The quality of the forecasts is comparable with those from our current model in production for these 2 shops. Calculations were parallelized over all 8 cores of my machine. Training plus prediction time for each date was about 4 seconds. The API is ridiculously easy to use and the documentation seems sufficient.

For what I can read in the documentation, Prophet does not accept features. Nevertheless, Prophet’s forecasts can be part of an ensemble that produces predictions with a higher granularity. It would be interesting to make a comparison for every shop. I was surprised by the result on the difficult shop history.

There are also several hyperparameters that would be interesting to look at, among several, these:

- **cap**: the maximum possible value of the target.
- **changepoint**: indicate where do we expect an abrupt change in the time series.
- **changepoint_prior_scale**: related to how strongly should the model adjust to trends.
- **holidays_prior_scale**: adjust the importance of holiday effects.
- **interval_width**: sets the uncertainty interval to produce a confidence interval around the forecast. This could be very useful for monitoring the quality of the forecast. Defaults to 80%.

To anyone starting a project using time-series for forecasting, I really recommend taking a close look at this tool. Great work Prophet!
Eneco is an integrated energy group with more than 7,000 employees, offering comprehensive solutions for, and together with, its customers and partners. Eneco invests in well-maintained networks, onshore and offshore wind farms, solar energy projects and biomass plants. Eneco operates from bases in the Netherlands, Belgium, the UK, France and Germany.

**Benchmark of Data Science and Engineering Capabilities**
For some time, Eneco has been building up its Data Science practice. To find out where they stand in comparison to other organizations, Eneco asked GoDataDriven to perform a Data Science Audit of their Data Science and Engineering capabilities. More information about the Data Science Audit: https://godatadriven.com/data-science-audit

“The Data Science Audit provided us with fresh insights and concrete recommendations for our organization”

Ronald Root, senior Data Driven Business Developer

The goal of the quick scan is to highlight the areas where your company can grow to become truly data driven. The explosion in popularity of data science and big data has often spurred ad-hoc solutions and inefficient workflows. This, in turn, often prevents advanced models to be taken into production. Based on the outcome of this audit, Eneco has been able to improve their organization, IT, and skills.

“I've personally experienced the professionalism of the GDD Data Science Audit. The recommendations can be implemented directly to the organization, IT and development of skills Ronald Root.”

For more customer stories, go to godatadriven.com/customers
Import Partitioned Google Analytics Data in Hive Using Parquet

Recently, I was working on importing Google Analytics data into an Amazon EMR cluster. Google Analytics was offering files as Avro, but we wanted Parquet files partitioned by date (we literally have a field date in there). There are multiple reasons why you would choose one or another, but for us it came down to faster analytics thanks to the columnar format.

Using Spark for the ETL process makes this a piece of cake:

```scala
(spark.read.format('com.databricks.spark.avro').load('dataset.avro')
  .write.format('parquet')
  .partitionBy('date').saveAsTable
  ('tablename'))
```

Or does it? The first issue is that if one of your columns has a nested schema exceeding 4000 characters, the Hive metastore will not accept it.

If you look around, you’ll see this is a long standing issue (https://issues.apache.org/jira/browse/HIVE-12274) open since October 2015. Who’s to blame here is apparently Oracle (it’s always Oracle!).

The good news is that this limit can be changed in the metastore! I’m assuming you’re using a Postgres instance as the metastore, but the syntax is similar all across the board!

Once you’re logged in type:

```sql
ALTER TABLE "COLUMNS_V2"
ALTER COLUMN "TYPE_NAME"
TYPE VARCHAR(8000);

ALTER TABLE "TABLE_PARAMS"
ALTER COLUMN "PARAMS_VALUES"
TYPE VARCHAR(8000);

ALTER TABLE "SERDE_PARAMS"
ALTER COLUMN "PARAMS_VALUES"
TYPE VARCHAR(8000);

ALTER TABLE "SD_PARAMS"
ALTER COLUMN "PARAMS_VALUES"
TYPE VARCHAR(8000);
```

At this point you might re-execute the Spark command above. But you’d be surprised by what Spark tells you WARN CreateDataSourceTableUtils: Persisting partitioned data source relation 'tmp.'tmp1' into Hive metastore in Spark SQL specific format, which is NOT compatible with Hive.
This is another long standing issue where the workaround is to first create the table, and then do one of the following:

- Insert into it for a particular partition (this can be accomplished with `INSERT INTO test_partition PARTITION(date=2013) SELECT * FROM test` or:
- Write directly to disk and then create the partition in Hive manually (for example: `ALTER TABLE test_partition ADD PARTITION(date=2013)`;

“First create the table” is of course deceptively simple: you need to create a partitioned table which is basically equal to the initial one, save for the date field, that must be set as the partition column.

Here’s the steps I’ve used to get this done:

- Use Spark to read the Avro file and write it unpartitioned somewhere
  ```scala
  (spark.read.format('com.databricks.spark.avro')
  .load('dataset.avro')
  .write.format('parquet')
  .saveAsTable('test'))
  ```
- Use beeline to save the created schema:
  ```sql
  beeline-u {jdbc connection string} -e 'DESCRIBE test' > schema.sql
  ```
- Edit the `schema.sql` file by hand, remove `date` from the columns and add it as a partition
  ```sql
  CREATE TABLE test_partition (
  fullVisitorId STRING,
  visitorId INT,
  ...
  visitStartTime INT
  -- date STRING, note that this is commented out now
  totals STRUCT<...>,
  ...
  )
  PARTITION BY ('date' STRING)
  STORED AS PARQUET
  ```
- Now you should execute this query in beeline. However, as I could not restart the metastore, and the metastore was checking on an ORM level for fields not longer than 4000 characters, I could not do it. After a good hour of searching, I thought that I could just execute the query using `spark.sql("YOUR QUERY HERE")`. I totally forgot that Spark could bypass the ORM.

- Now, normally, you could just directly write using `partitionBy('date').mode('append').saveAsTable('test_partition'). However you cannot use `partitionBy` with `saveAsTable` if the table already exists. And if you remove the `partitionBy`, Spark assumes that `field4` (the one that was coming after `date` in the example above) was supposed to be partition column (this is of course not correct);
- At this point, what’s left is to use:
  ```python
  (df.write.format('parquet')
  .mode('append').partitionBy('date')
  .save('/user/hive/warehouse/database.db/test_partition'))
  ```
- Since we’ve manually written the files, we still need to tell Hive that new partitions are there. Doing that programmatically in Spark is simple (ugly interpolation, sorry!):
  ```python
  for dt in df.select('date').distinct().rdd.map(lambda row: row['date']).collect():
    spark.sql("ALTER TABLE test_partition ADD PARTITION ('date'='%s')" % dt)
  ```
- Done! You can now query the data. Note that only the last two steps are needed when new files come in. They can be automated easily enough with the workflow manager of your choice!