Vantage 101

Introduction

Dear colleague,

Welcome to Vantage AI! You have joined a community of ML Engineers / Data Scientists / Data Engineers that is characterized by being very eager to learn, and believes that learning is facilitated by sharing with each other. In this spirit, the Vantage 101 course serves as a list of topics and assignments that are more or less essential for a highly skilled data practitioner. Although these subjects are often learned on the job, it is often the case that learning them comes with a realization of how to do things right, and leaves you thinking "I wish I had known this before". To ease this transition, this course will present you with the know-how which will certainly prove useful in the near future, so that you can work better and faster. Since you already have experience working on ML projects, some of these topics will not be new for you, and the time it can take to complete the whole course can vary greatly, so don't be too concerned about the time it takes and rather make sure you will eventually understand the purpose and implementation of the tools presented here.

The structure of the course is as follows: a topic is introduced and relevant material is linked, to which the reader should invest time enough until the subject is understood. Then, a hands-on assignment is outlined. These assignments are based on the code you used in your interview in Vantage AI. The assignments are an incremental effort, improving the code at every step. The end result will be production ready code, with all the tools needed to improve it, as well as a model deployed in the cloud.

We would greatly appreciate your feedback in this document, or to build upon it, so make to do so!

Good luck and enjoy!

ML Engineer and ML teams

To start, what is a ML Engineer? Here our own Lieke sheds light on the current terminology of data-related jobs, how they can be related to the previous all-encompassing terms of data scientist and data engineer, and what are the important things to focus on (spoiler: it's not the terms). Also, read here to get an overview of a ML team composition.

IDE

If you have not already, download and use an IDE. At the time of writing, VS Code and PyCharm are the top contenders. If you must know, the writer of this document prefers VS

Code (it's free!), but he also believes that the preferences of each person are usually skewed in favor of the first IDE that person has encountered first.

Git

The git manual serves as a complete tour for the functionality of git, especially good if the commands are followed along. The fundamental commands to develop with git can be found on sections Developing with Git and Sharing development with others, while the sections Repositories and Branches and Exploring Git history will give you a sense of how to explore your git repo. It is important to note that the manual explains some commands that are less popular and only available on recent versions of git. A better walkthrough of git was done by our former colleague Guido and can be found here. What is fundamental to understand are the commands: clone, add, commit, push, pull (so fetch and merge) and checkout. In most cases, these commands will be enough to get you around.

After understanding the main building blocks of git, it is useful to know preferred workflows for it. Here you can find a popular workflow for git that is often used for Data Science.

Here you can find an interactive way to learn Git.

Assignment:

Create a develop branch, commit the changes you have made in your repo (from the previous assignment) then observe the changes in a log. Create a new feature branch from develop, and add changes in a way that you expect to have a merge conflict, merge them, and solve conflicts. Finally explore commits in the past. You can try to use the git functionality that your IDE provides, and observe if it does what you expect it to do.

Github

Follow some tutorials on Github Learning Lab, depending on your level. At the very least it is important to understand pull requests, conflicts, and how to clone/push to and from a remote repo

Assignment:

Push your repo in your github account. From now on you can always make a commit at the end of every assignment and push it to your remote!

OOP

Make sure you understand the concepts and benefits of object oriented in python, which are explained here (and in the subsequent videos of the same series). Here you can find an example of how OOP is used in data science products, but there are many other scenarios when OOP makes your life easier in your projects.

Assignment: Make sure that (part of) the functionality of your code (which should be in src) is in the form of classes, for example, make a parsing class that deals with all the parsing part of the data, and a model class that encapsulate all the modeling part. Since the parsing and the modeling part would deal with different components of your project, it may be a good idea to have them in separate files in your src folder. If you do not have a function/class for each of these 3 parts, write it (based on the code you already have)

Atomic functions

Read about Atomic functions

Assignment: Ensure the functions in your classes are atomic.

Virtual environments and making a package out of your code

Another key part of setting up your project is to create a virtual environment for it. In your virtual environment, you will often need to make and install your own package, because of the functionality it will provide, and the increase in code quality, see this blog for reference.

Assignment:

- Copy paste the assignment you did to join Vantage AI into the created repo from the previous assignment (if it is all in notebooks, paste the notebook on the notebooks folder)
- Migrate all the logic from the notebooks to the src folder, that is, place all the functions into the src folder (in order to do this, you must have all the logic in functions and classes, not in the notebook). Place your functions/classes in the subfolder that makes the most sense (by name)
- Adapt the requirements file so that your environment also has the libraries your project needs and create a virtual environment from the requirements file. Make sure this environment has installed the package from your own repo
- Make sure you can use your own environment (using ipykernel, as shown here) and package in your notebooks, that is, that you can import on your notebook the code you have on src, so that you can use your functions in your notebook.
- Make sure that all the logic of your repo is now migrated to the source code. Therefore, the notebook serves only as an interface that shows how your package could be used.
- https://godatadriven.com/blog/a-practical-guide-to-using-setup-py/

Repo structure - cookiecutter

When embarking on a DS project, it is essential to have an effective code structure and version control for a number of reasons. It is best to set this up at the beginning of a project, in order to get the benefits of them sooner rather than later. Cookiecutter helps with setting up the structure of your repo. One (of many) templates for code structure can be found here. You will notice that the template has some elements in it that will come back later in this document, so if you don't understand right away why a Makefile is created for example and what it does, that is fine! When choosing a template, make sure you understand the ideas and workflow proposed in it.

Assignment:

Setup a folder structure using cookiecutter and track your project using git, which should be linked to a remote, likely on your github (if you are not familiar with git, you can leave this part for Assignment 3).

It is a good habit to begin projects in such a way.

Modern Pandas

Read about Modern Pandas, especially the first 5 parts (the last 3 are optional), some highlights of this are:

- Use `.loc` and `.iloc`, never use `][` (and definitely not `.ix`), eg.

  Bad:

  ```
  f[f['a'] <= 3]['b'] = f[f['a'] <= 3 ]['b'] / 10
  ```

  Good:

  ```
  f.loc[f['a'] <= 3, 'b'] = f.loc[f['a'] <= 3, 'b'] / 10
  ```

  Better:

  ```
  f.loc[lambda df: df['a'] <= 3, 'b'] = f.loc[lambda df:
  df['a'] <= 3, 'b'] / 10
  ```

- Use Method Chaining, therefore use `.assign` and `.pipe`, and forget about `inplace = True`. Consider decorators for making your life easier when debugging such chains.

- Be aware of the index in your dataframe. All binary operations (add, multiply, etc...) between Series/DataFrames first align the index and then proceed. Also, the index type can determine functionality, eg. Datetime index allows resampling.

- Make sure to vectorize when possible, but also think twice before you jump into optimizing the runtime of your code: how necessary is it? How much time will it take you as opposed to the benefit, and does it make any difference to the business? (would the business really be affected otherwise?)

- Use list comprehensions, for example, when you're loading many files with (allegedly) the same format, you can do:

```
weather_dfs = [pd.read_csv(fp, names=columns) for fp in files]

weather = pd.concat(weather_dfs)
```

  And not a for loop that iterates through the files, reads them, and concatenates them.

- Convert dtypes early on
- Avoid `.apply`, especially with `axis = 1`
- Make sure your data is tidy, understand the difference between long format and short format, and see how the operations `.melt .pivot_table .stack` and `.unstack` can help you switch between them. Notice whenever the analysis you are going to do with the data expect data on long or wide format. As an example, see here for the format expected in seaborn.

Assignment: Rewrite your code so that it follows the modern pandas principles

----

PEP-8

Your code will always be more readable (and therefore more useful to your team and yourself) if you follow some conventions. See this video to learn about PEP-8 and make sure your code follows these conventions.

Code formatting and linting

Code formatting and linting are clearly explained here:

Linting highlights syntactical and stylistic problems in your Python source code, which oftentimes helps you identify and correct subtle programming errors or unconventional coding practices that can lead to errors. For example, linting detects use of an uninitialized or undefined variable, calls to undefined functions, missing parentheses, and even more subtle issues such as attempting to redefine built-in types or functions. Linting is thus distinct from Formatting because linting analyzes how the code runs and detects errors whereas formatting only restructures how code appears.

Assignment:

Set up a code formatter such as Black, and a linter such as pylint in your project. Find a way to use them in a convenient way, eg. to code format every time you save your project in your IDE.

Testing

Read about Unit testing and how to follow the AAA pattern.

This video is a great introduction to Pytest

Assignment: set up tests in your code for the most important functions. Check for the way in which tests can be run/debugged from the IDE of your choice (and while you're at it, make sure you know how you can work with the debugger of your IDE! Your debugger is your best friend).

Pre-commit and makefiles

Learn about pre-commit

TL;DR: the quick start for pre-commit should already get you going. You can use the example .pre-commit-config.yaml file to also include the git hook ‘Black’ during pre-commits.

Learn about Makefiles

Read how to glue it all together here!

Assignment:

Find a way to conveniently run tests formatting and linting via Makefile and pre-commit.

ML algorithms

ML is a constantly evolving field where the best performing algorithms can constantly change. At the time of writing, gradient boosting algorithms such as LGBM or CATBoost are the favorite of many for its great performance and ease of use. Read here to learn more about these kinds of algorithms, implement them and compare its performance with the one you have in your repo. Also check here to learn how to best tune the hyperparameters. Tip: When tackling a new problem, it never hurts to check Kaggle to see how similar problems are currently tackled!

Serve as a REST API

Investigate ways in which you could serve your model, for example, like a REST API. You can find inspiration here.

Assignment: Serve your model as a REST API locally.

Docker

If you are a beginner to Docker, you can start a tutorial here.

Note: at the time of writing, the files in the Readme of section 2.0 are outdated. The correct files are found here.

Also consider this blog.

Deploy to GCP

Assignment: Deploy your Machine Learning model to Google Cloud! Learn here how to. If you make an account with your @vantage-ai.com address, you will get 300 dollars of free credits! If you would like to know more about GCP and/or cloud computing in general before you start, you can check here courses for using the cloud aimed at audiences at different levels. Pick the one (or more) that suits you best and give it a go!

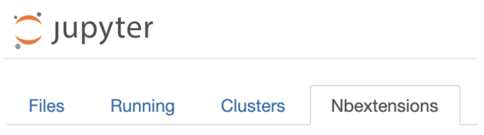
Conclusion

After having done all the assignments above, you have created high quality code that is set up in production, and deployed a model in the cloud. Congrats! Make sure to check the rest of the knowledge from your colleagues, for example, you can find here some inspiration in how you could monitor your model. Also, make sure to take a look at the articles from BDR.

Additional topics:

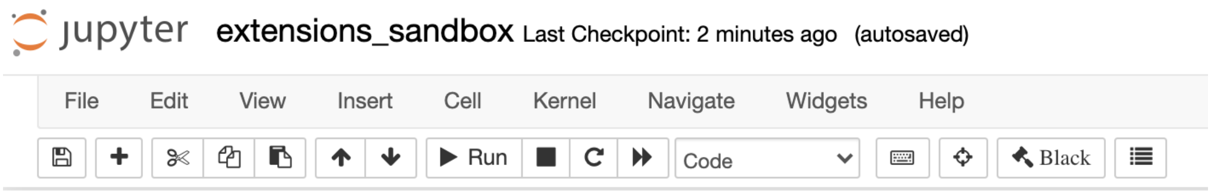
SQL (in the cloud)
In the near future you will likely find yourself doing SQL in the cloud. Here you can find a tutorial on how to use BigQuery (SQL on GCP), and if you could use some training with your SQL skills, make sure to check this full beginners course.

Jupyter notebook extensions
Jupyter notebooks are ideal for conducting experiments or creating PoC's, but lack many of the advantages that your IDE has. To some extent, additional functionality can be added to your notebooks by enabling Jupyter notebook extensions. I usually like to also have the configurator for those extensions installed, so that you can enable and disable extensions in a separate tab:



In that tab, you can explore the extensions with their descriptions. Some of my personal favorites are: ExecuteTime, Variable Inspector and Table of Contents. Note that not all possible extensions are listed there. Did you like the automatic code formatting that "Black" does for you? You can enable that functionality in your notebook separately here: https://github.com/drillan/jupyter-black.

After having enabled the extensions, when you open your notebook, you will note that new icons are part of the toolbar:



iTerm2
As an alternative to the standard terminal you can install https://iterm2.com/.

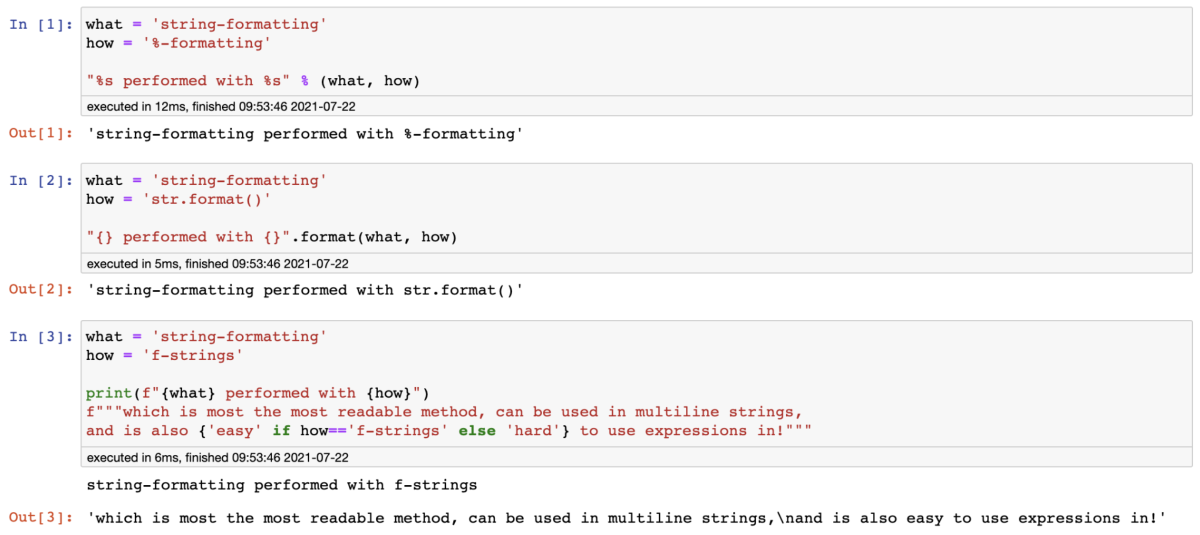
Oh My Zsh (for MacOS or linux users)
Make your boring plain terminal look sexier and also more readable with Oh My Zsh! They have a bunch of different themes, but the default one already gives you standard clarity on which git branch you are at all times!

If you face the problem that the `conda` command does not work anymore, use the following guide to fix this issue:
https://stackoverflow.com/questions/31615322/zsh-conda-pip-installs-command-not-found
.

Oh My Zsh theme and plugins
To make Oh My Zsh even more comfortable you can use a theme such as https://github.com/romkatv/powerlevel10k.

Some other plugins that are recommended are:
- https://github.com/zsh-users/zsh-autosuggestions
- https://github.com/zsh-users/zsh-syntax-highlighting
- https://github.com/zsh-users/zsh-history-substring-search

String formatting
Still doing string formatting using `%s` or `str.format()`? Since python 3.6, formatted literal strings, or "f-strings" are introduced, which will make your life a lot easier! See the example below comparing the methods:



```
In [1]: what = 'string-formatting'
        how = '%-formatting'

        "%s performed with %s" % (what, how)
        executed in 12ms, finished 09:53:46 2021-07-22
Out[1]: 'string-formatting performed with %-formatting'

In [2]: what = 'string-formatting'
        how = 'str.format()'

        "{} performed with {}".format(what, how)
        executed in 5ms, finished 09:53:46 2021-07-22
Out[2]: 'string-formatting performed with str.format()'

In [3]: what = 'string-formatting'
        how = 'f-strings'

        print(f"{what} performed with {how}")
        f"""which is most the most readable method, can be used in multiline strings,
        and is also {'easy' if how=='f-strings' else 'hard'} to use expressions in!"""
        executed in 6ms, finished 09:53:46 2021-07-22
        string-formatting performed with f-strings
Out[3]: 'which is most the most readable method, can be used in multiline strings,\nand is also easy to use expressions in!'
```

Type hinting

https://realpython.com/lessons/type-hinting/

Hyperparameter tuning

Grid search/ random search
optuna/ hyperopt

https://neptune.ai/blog/hyperparameter-tuning-in-python-a-complete-guide-2020

Hyperparameter tuning, especially grid searches, can be extremely resource-intensive. Sklearn's built-in grid- and randomized search aren't the most advanced methods, although they can be a good option for less computationally intensive models. For models that have longer training times, e.g. neural networks, more advanced optimization methods are better suited. Have a look at this blog to get an idea of different hyperparameter tuning methods.

Agile

Most teams nowadays work with the agile project management paradigm, so it's worth knowing the basics. Some popular agile methods are Scrum, Kanban and SAFe.
Here is a general guide to agile:
https://www.tutorialspoint.com/agile/index.htm
And here a list of more in depth resources:
https://medium.com/javarevisited/7-best-agile-and-scrum-online-training-courses-3b191e9b65eb

Knowing how to translate a business problem into a data science solution is a valuable skill. Our colleagues wrote a white paper describing this process: