Hi, I’m Jesse. I have been in education and technology industry for several years. In this industry, I have tackled the issues between data science and engineering in AI / ML projects. Based on my experiences, I am going to talk about the process to productionize Research Oriented Code by Python.
Background

- Python engineers assign to AI / ML projects and frequently face the research oriented code.

- Understanding the process to productionize the research oriented code can help make AI / ML projects work more smoothly.
3 years ago, I made a 5 minutes lighting talk about this topic in Pycon US 2019. This was my first talk on the stage in my life. During the three years, I have made the talks about python and data science topics around the world, and published the book about them. Then I came back to pycon US today. So, this talk is aggregated based on my three year's experiences of the projects with data scientists, what I wrote in the book, and the past talks in pycon.
So, I abstracted the four steps to transform research oriented code into products. Understand the code, modularize the code, refactor them, and make them a product. That’s it. It’s very simple. I will explain each step one by one much deeper.
4 Step Transformation from Research Oriented code into Products

Understand the research oriented code

First, understand the research oriented code
What is Research Oriented Code?

In general, research oriented code is implemented for figuring out something new and summarize the results into paper.
Definition

Research Oriented Code in AI/ML projects is the code written mainly by data scientists or researchers for figuring out new knowledge.

Finding something new is the highest priority when writing the code.
This is the common lifecycle to write paper with code.

Only one person start to collect data, preprocess it, train or calculate the data, and then see what the results are telling you. If the results present new knowledge, write paper with it and publish.
Example of pre processing code I wrote

```
def insert_lcc_param(df_results):
    item_id = []
    resp = []

    num_lines = len(df_results.index)
    out_list = []
    n = 0
    a_first_row = True

    for index, row in df_results.iterrows():
        if item_id in row[0] and not is_first_row or (n == num_lines):
            out_list.append(call_lcc_param(exam_name, item_id, theta, resp))
            theta = []
            resp = []
            theta.appendrow(1)
            resp.appendrow(0)
        else:
            resp.appendrow(0)
            theta.appendrow(1)
            is_first_row = False
            item_id = row[0]
            exam_name = row[3]
            return out_list
```

This code is the pre processing code written during these process.
This code seems to be long function and use for-loop. But it tends to be visually traceable from the top to the bottom.

Also, it can be easily and quickly written. It’s not clean code but it’s enough to quickly get results.
Example of calculation code I wrote

```python
logreg = LogisticRegression(solver='lbfgs')
logger = None

def calc_ico_params(slope, intercept):
    a = slope / 1.701
    b = -(1) * intercept / slope
    return a, b

def calc_logreg(exam_name, item_id, theta, resp):
    theta = pd.DataFrame(theta)
    logreg.fit(theta, resp)
    slope = logreg.coef_[1]
    intercept = logreg.intercept_[0]
    a, b = calc_ico_params(slope, intercept)

out_row = []
out_row.append(exam_name)
out_row.append(item_id)
out_row.append(a[0])
out_row.append(b[0])
return out_row
```

This is the example of calculation code. This code use .append and use data-frame. It easily handles the input data and trace output data with data frame.
What is the code in production level?

So, on the other hand, what is the code in production level?
Quality definition of the production code

0 points: More of a research project than a productionized system
1-2 points: Not totally untested, but it is worth considering the possibility of serious holes in reliability.
3-4 points: There’s been first pass at basic productionization, but additional investment may be needed.
5-6 points: Reasonably tested, but it’s possible that more of those tests and procedures may be automated.
7-10 points: Strong levels of automated testing and monitoring, appropriate for mission critical systems.
12+ points: Exceptional levels of automated testing and monitoring.

According to the thesis about how we can evaluate ML production system based on a rubric, the thesis scores ML system from 0 to 12+. 0 points seems to be more like research oriented code before transformation. The ML system scoring from 7 to 12+ is the product quality a few years after deployed on the production environment. The target goal of the four steps I suggest in this talk is the product quality scoring from 5 to 6. That means reasonably tested, but it’s possible that more of those tests and procedures may be automated.
The product quality is improved through these kinds of the development cycle. I have had experiences to be only one python engineer in a company for a few years. But generally, engineers develop web application with team. They make architecture, implement features, test and review, and release a product. They receive feedback all the time unless they develop very perfect product with no bugs. So, they need to keep repeating release and fix the product. This means that the production code need high maintainability, scalability, and processing speed.
Example of both previous code I refactored

```python
def filtered_results(results):
    input_data = delete_name_score_per_item(replaced_data)
    ability_per_item_id = {
        row[1]: [row[2], row[3]]
        for row in input_data if row[1] != -1
    }

    item_ids = [value[0] for value in row if value in ability_per_item_id]
    X_trains = [value[1] for value in row if value in ability_per_item_id]
    y_trains = [value[2] for value in row if value in ability_per_item_id]

    return item_ids, X_trains, y_trains

def calc_logreg(X_train, y_train):
    logreg = LogisticRegression(solver='lbfgs')
    logreg.fit(X_train, y_train)
    intercept = logreg.intercept_[0]
    return slope, intercept

def calc_hyp(consts, slope, intercept):
    a = slope / intercept
    b = -1 / intercept / slope
    return a, b
```

This code builds the model in a faster and simpler way.

This is example of both previous code I refactored in pythonic way. This code seems to be shorter than previous code and there are list comprehension and set comprehension. There are two simple functions on the bottom. This code can build the model in a faster and simpler way.
I identified three differences between research oriented code and production code. There are different scopes, different characteristics of coding style, and different objectives of coding style.

Researchers focus more on writing pre-processing code and ML code. On the other hand, engineers have responsibility to write the whole part of code in production level.

The research oriented code seems to be easily-handled and visually traceable and on the other hand, production code need to be concerned about high calculation speed, high readability, and they are testable and modular.

This is because researchers focus on finding the most efficient and suitable machine learning model and on the other hand, engineers have responsibility to make the code work on the server correctly and reliably.
What are Python Engineers supposed to do for Research Oriented Code at first? What is first thing to do?
3 differences between research oriented code and production code

Read the code before write it.

Deep dive into the code. Read the code before write it.
This left picture is the jupiter note book which has long lines. I used to frequently receive the code on the jupiter note book from data scientists. After you received it, take notes for deep dive into the code as if you were scribbling.

There are three strategies to take notes to prepare for modularization. Just quickly take notes while reading the research oriented code.

First one is to write comments by using “#”

Second is add “TODO” comments.

These two are things everyone already does. Third one is the most important among these strategies.

Third is mob documentation. Mob documentation is reading the code with more than two team members.

Mob documentation is the very useful way to efficiently understand the code with team.

For example, if you are using vs code, install live share extension of vs code which enable to share code and edit together. After mob documentation, the team members who joined this mob can be on the same page and do not need extra wiki and documents about this code. Also, if you do with junior or new hire, it can be good for onboarding.
This is an example of a part of page of research oriented code with comments. While reading the code, add what you understand from the code with comments out or add TODO memo.
I know that many audiences already apply them on writing the code by yourself for remembering the code and todo, but not for understanding more about the code. The conscious and attitude about understanding and getting familiar with the code is going to make different in the way to take notes on the code, and easily lead to the next step.

Now, it’s ready to move to the next step.
Next thing to do is modularize the code by using labels
Categorize research oriented code from code documentation

Use category labels such as preparation code, pre/post processing code, and calculation code.

1. Preparation code
   # Get file name
   # Create new database "images.db"
   # TODO: set it config file
   # Create new connection to database object
   # TODO: Use OR mapper
   # Create cursor object to operate sqlite
   # Initialize database
   ...

2. Pre/Post processing code
   ...

3. Calculation code
   ...

Research oriented code has three types of scopes such the code to prepare, pre/post process, and calculate data. Based on the comments you wrote in previous step, categorize the code into preparation code, pre/post processing code, and calculation code.
Break them into functions and make them testable

Find duplicated code and delete or unified them, or fix small bugs

1. Preparation code
   # Get file name
   # Create new database "images.db"
   # TODO: set it config file
   # Create new connection to database object
   # TODO: Use OR mapper
   # Create cursor object to operate sqlite
   # Initialize database
   ...
2. Pre/Post processing code
   ...
3. Calculation code
   ...

1. Preparation code
   1. init_db()
   2. get_filename()
   3. load_config()

2. Pre/Post processing code
   ...
   ...
3. Calculation code
   ...
   ...

After grouping each code with labels, we can break them into functions and make them testable.
While grouping each code, you would find duplicated codes, unused lines or libraries, or small bugs. You can fix them in this step.
Then we can get modularization outcome like the table.

<table>
<thead>
<tr>
<th>Module name</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation code</td>
<td>- Access database</td>
</tr>
<tr>
<td></td>
<td>- Execute query</td>
</tr>
<tr>
<td></td>
<td>- Load input data</td>
</tr>
<tr>
<td>Preprocessing code</td>
<td>- Replace categorical data with discrete numbers</td>
</tr>
<tr>
<td></td>
<td>- Filter input data</td>
</tr>
<tr>
<td></td>
<td>- Rename columns</td>
</tr>
<tr>
<td>Calculation / execution code</td>
<td>- Calculate logistic regression</td>
</tr>
<tr>
<td></td>
<td>- Output results</td>
</tr>
</tbody>
</table>

We could get modules for each from one page of research oriented code.
We could get “preparation.py” which has functions to access database, execute query, and load input data.
We also could get “preprocessing.py” which has functions to replace categorical data with discrete numbers, filter input data, and rename columns.
Finally, we get “prediction.py” which has functions to calculate logistic regression and output results.
Modularization outcome

The research oriented code became loosely coupled.

As a result of that, the research oriented code became loosely coupled.
Mapping each module into directory

Now, you can make directory structure mapping each module. This is one of examples based on my experiences to develop API with Flask. I found that this kind of directory structure is very easy to work with.

The directory structure on the left box has the api directory and model directory as a main root directory. The __init__.py module located in api directory has routing lists.

Each logic for each endpoint is implemented in each module located in urls directory.
If you are working in a small team from 1 to 3 members and add new endpoint, you can just add it to new endpoint in urls directory. If the number of team members increase, you can split each module into each version and work together staying loosely coupled between modules.

I wrote the directory on the slide in the case of API with Flask and generally the way of mapping relies on the manner of each framework. But Fast API also can have similar directory structure.
Now, we reached to the step to refactor preparation code and pre processing code.
Before starting to refactor the code, we should write test code roughly, add docstrings such as reStructuredText style, Numpy style, and Google style, and execute code formatter such as black and autopep8, and check if code correctly work by using pytest, and check the consistency of coding style by using flake8. You can narrow down more about requirements of each code through these three processes.

Then we can finally start to refactor the code.
Now which part of the code should we refactor?

Just focus on preparation code and pre/post processing code.

In general, ml library is used in prediction module so the code does not have many lines to write and refactor. The optimization of CPU bounding processing can be done later after we could make sure if the code can work correctly on the server. Also, we have chances to figure out if the prediction is not validated yet though refactoring.
Now which part of code should we refactor?

Improve CPU bound processing but in this phase nothing to do.

Simplify I/O

Remove extra modules or replace library for application, not for analysis.

So, we are trying to refactor I/O. Preparation.py has a group of codes to access to database or object storage by using query and client library. Let me share a few examples I frequently have faced and how I simplified them.
preparation.py: simplify i/o

The two strategies to simplify I/O
1. Narrow down the data to extract from database -> faster and lower cost
2. Wrapping client library

```python
from xxxx import yyyy
client = yyyy.Client()
query = "SELECT * FROM 'data set name'"
query_job = client.query(query)
results = [list(row.values()) for row in query_job.result()]
```

```python
from xxxx import yyyy
client = yyyy.Client()
query = "SELECT column_1, column_2, column_3 FROM 'data set name' where column_1 is not NULL"
query_job = client.query(query)
results = [list(row.values()) for row in query_job.result()]
```

In exploratory analysis phase or data wrangling, data scientists are not sure of which data is specifically useful. If you received the first query with star, narrow down the data to extract from database. It could be much faster and lower cost like the second code.
In accessing to the object storage, sometimes you might make effort to write many lines just for loading csv file by using client library. One of my suggestion is that you wrap the code and make it simpler, then it can be re-usable and cost-effective.

That's because these kinds of code is the same code even though anyone write. Everyone in AI/ML projects such as data engineers, python engineers, and data scientists write the similar code. Not only your team members, but also the members in other team might be writing the same code just for loading the data. Also, it might be chances for python engineers to make a big impact internally.
Now which part of code should we refactor?

- Improve CPU bound processing but in this phase nothing to do.
- Simplify I/O
- Remove extra modules or replace library for application, not for analysis.

Next, let's look at pre-processing.py. The libraries for analysis use and development use are different. A group of the code to pre process or post process the data tend to include various kinds of libraries and data type is also mixed.
As one of the common cases, there are three ways to pre/post process the data like pandas, python only, and SQL query. The common operations are filtering, replace, de-duplicate, and delete the data for each three way. Most of research oriented code needs pandas code or sql query or both.
These three ways have different features. The code with pandas is iterative, python only code can be testable, and SQL query has high processing performance and simple grammar.

It depends on architecture and project scale but the first thing to think about is whether pandas and sql query is necessarily or not, and refactor them into python only code for making them more testable. If you could narrow down which data you calculated in preparing for data with SQL Query, you do not need to write much pre-processing code.
4 Step Transformation
from Research Oriented code into Products

1. Understand
2. Modularize
3. Refactor
4. Make them a product which is API

Last step is the step to make them product which is API.
-> 18m
What products can be generated from Research Oriented Code?

In fact, what products can be generated from Research Oriented Code?
Output is not paper. Its product which means the code working on the server and user can access to it.
The Flow Chart of Transformation from Research Oriented code into Products

The products can be web application or web api. This is drawing of the flow chart transformation from Research Oriented code into Products. There are two main flows to make the research oriented codes products. Integrate the code into web application, or integrated ready-made api into web application. On the other hand, implement web api from scratch based on the research oriented codes, or extracted ml processing code from web application and make it web api.
If each step of transformation from Research Oriented code into Products is mapped on this flow chart, it's going to be like this slide. In this talk, I will focus on making the code web api.
The Flow Chart of Transformation from Research Oriented code into WEB API

In order to transform Research Oriented code into WEB API, we have to implement how to route request, check request, check error, and routing points which are endpoints.
Let me share the tips to implement request routing and url.
Request routing: clarify input and output and define URI from data

@app.route("/v1/probabilities", methods=['GET'])
def probabilities():
    return calc_results(), 200
    return get_probs(), 200

There is the table of input data on the left side on this slide which is used for two parameters logistic regression. On the right side, there is the table of output data calculated by the model.

The input data is about whether students answer each question correctly or not. There are item_name which is the name of question, item_id, how difficult each question is, subject names, exam name, and correction which is binary data.

On the other hand, output data is about probabilities to answer questions correctly.

The role of the API is to get probabilities so we can directly label probabilities as the endpoint name. Based on best practices, function name should be verb or verb + noun. The role of this function is to calculate results or get probabilities. So we can label calc_result or get_probs as the function name. So, understanding what data is input and output, which means understanding what data the code calculates and makes, is considerably important step to make API.
Next is the implementation of request parameter check.
Request parameter check:
write decorators with JSON Schema

<table>
<thead>
<tr>
<th>Request curl command</th>
<th>JSON Schema File(make_name_grade.json)</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl <a href="http://localhost:5000/">http://localhost:5000/</a> -X POST -H &quot;Content-Type: application/json&quot; -d</td>
<td></td>
</tr>
<tr>
<td>&quot;{&quot;student_name&quot;: &quot;test_name&quot;, &quot;student_grade&quot;: &quot;forth-grade&quot;}&quot;</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;$schema&quot;: &quot;<a href="http://json-schema.org/draft-04/schema#">http://json-schema.org/draft-04/schema#</a>&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;student_name&quot;: {</td>
</tr>
<tr>
<td></td>
<td>&quot;type&quot;: &quot;string&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;required&quot;: &quot;True&quot;</td>
</tr>
<tr>
<td></td>
<td>},</td>
</tr>
<tr>
<td></td>
<td>&quot;student_grade&quot;: {</td>
</tr>
<tr>
<td></td>
<td>&quot;type&quot;: &quot;string&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;required&quot;: &quot;True&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;maximum&quot;: 120,</td>
</tr>
<tr>
<td></td>
<td>&quot;minimum&quot;: 1</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

My suggestion is to implement request parameter check with json schema.

This request curl command has student_name and student_grade as parameters.
This json file is sample json schema. In this file, you can define what data should be checked and how the data should be checked.
So, this make_name_grade.json file is defining the data type of student_name as `string` and it must be included as request parameters, and the data type of student_grade is also defined as `string` and it must be included as `true`, and the maximum length of characters is limited in 120 and minimum length is 1 character.
So, in this example, request parameters are not allowed to be empty.
Request parameter check:
write decorators with JSON Schema

Validate request body based on schema file

```python
def validate_json(f):
    @wraps(f)
    def wrapper(*args, **kw):
        try:
            request.json
        except BadRequest as e:
            msg = "This is an invalid json"
        return jsonify({"error": msg}), 400
        return f(*args, **kw)
    return wrapper

def validate_schema(schema_name):
    def decorator(f):
        @wraps(f)
        def wrapper(*args, **kw):
            try:
                validate(request.json, current_app.config[schema_name])
            except ValidationError as e:
                return jsonify({"error": e.message}), 400
            return f(*args, **kw)
    return wrapper
```

Add schema file to each endpoint

```python
@app.route('/', methods=['POST'])
@validate_json
@validate_schema('make_name_grade')
def index():
    if request.is_post:
        data = json.loads(request.data)
        print(data["student_name"])
        print(data["student_grade"])
        return "Hi! " + data["student_name"]
    else:
        return "Hi!"
```

Based on the json schema, the json_validator.py validates request parameters. The first validate_json function checks whether the json schema exists or not.
The second validate_schema function checks whether the request parameters are correct or not based on json schema defined in previous slide.
In order to execute these functions, write function name as a syntax sugar under the endpoint.

The code with json schema seems to be complicated but most of the code are like this on the slide. So, you can easily find a lot of the similar sample codes on the web resources. Or you can refer to this code. I will upload this slide on my twitter as well.
The Flow Chart Transformation from Research Oriented code into WEB API

Last implementation is error check.
Before implementing the code, we need to think if processing should be stopped or continue for each code. Whether the processing is stopped or not depends on service specification. For example, if you implement a recommendation system, sometimes you do not want to stop service and continue recommendation even when the preparation code fails to load certain data.
This is an example of error handler with Flask.
If you stop processing, you can use abort function and detect error. If you don’t want stop processing, you can just return the results with jsonify.
That’s it. It’s very easy to implement it but it’s more difficult to decide which part of the code processing should be stopped or not. So, I suggest to talk with product manager or QA team about the service specification before starting to implement error check.
Summarize 4 Step Transformation from Research Oriented code into Products

1. Understand the characteristics of the code and figure out how it is working by taking notes.
2. Modularize the code based on the code documentation by labeling the code as preparation, pre/post processing, and calculation.
3. Refactor the preparation code by simplifying I/O and the pre processing code by changing the coding style.
4. Make them a product which is an API composed of request routing, request parameter check, and error check.

Ok, so let me quickly summarize 4 step transformation from Research Oriented code into Products.

First thing is to understand the characteristics of the code and figure out how it is working by taking notes. Second, modularize the code based on the code documentation by labeling the code as preparation, pre/post processing, and calculation.

Thirdly, Refactor the preparation code by simplifying I/O and the pre processing code by changing the coding style.

Lastly, Make them a product which is an API composed of request routing, request parameter check, and error check.
After deployed the product, we need to optimize speed and stability by doing loading tests. There are useful loading test tools such as locust and vegeta. Locust is python based load test tool which can make it possible to manage loading test scenario by python code and monitor the loading on server by GUI. Vegeta is go-lang based HTTP load test tool. It can be used both as a command line utility and a library. If you are looking for the light load test tool, vegetal could be one of the options.

By using those kinds of load test tools, if we figure out that performance is not good enough, we do parameter tuning of web server and application server, and think about asynchronies or synchronies. Even if it is not improved, rethink about architecture of infrastructure or refactoring the code with different languages.

-> 22m
Just in case, again, 4 Step Transformation from Research Oriented code into Products is very simple. Understand the code, modularize the code, refactor them, and make them a product. After deployed the code, check the performance.

-> 28m
Thank you!

@JesseTetsuya
Request routing:
clarify input and output and define URI from formula

\[ p(\theta) = \frac{1}{1 + e^{-Da(\theta-b)}} \]

Item Response Theory: Two Parameters Logistic Regression
https://www.publichealth.columbia.edu/research/population-health-methods/item-response-theory

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>theta: ability</td>
<td>Probabilities to predict correction for item</td>
</tr>
<tr>
<td>a: discrimination parameter</td>
<td></td>
</tr>
<tr>
<td>b: difficulty parameter</td>
<td></td>
</tr>
</tbody>
</table>

If you are familiar with mathematics, formula could have an important role in understanding input and output, and how the code should behave. It