Project Process

Automated Scales for real-time weight capture

Zhiyi Ding
Aug 8, 2017

Requirement

Input parameters:

1. Sensor data pulled from api
2. A latest batch date
3. Standard guide data

Output:

- Average weight of each data point in the last day/hours

Premex's Farm

Implementation

Main Technologies:

Python 2.7

Pandas 0.20

IPython

Beginning with version 6.0, IPython stopped supporting compatibility with Python versions lower than 3.3 including all versions of Python 2.7. [Link]

Steps:

1. Data preprocessing

2. Sensor processing

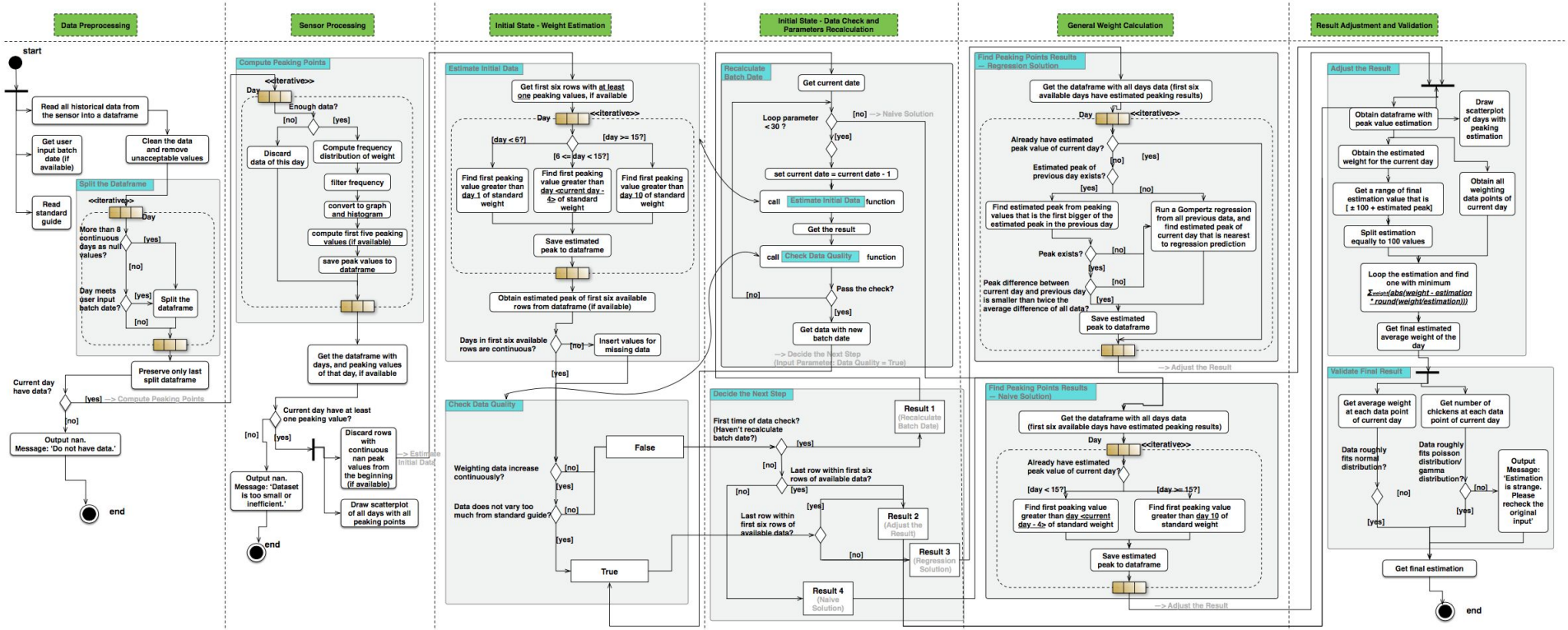
3. Weight estimation in initial state

4. Data check/Parameter recalculation

5. General weight calculation

6. Result adjustment and validation

Design



UML, Activity Diagram

Implementation & Verification

1. Data Preprocessing

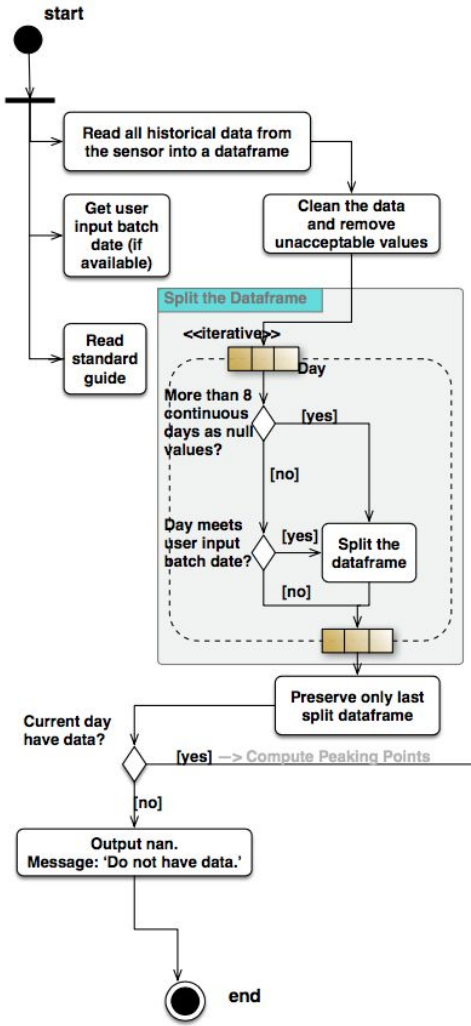
1) Read in all parameters

2) Clean the data

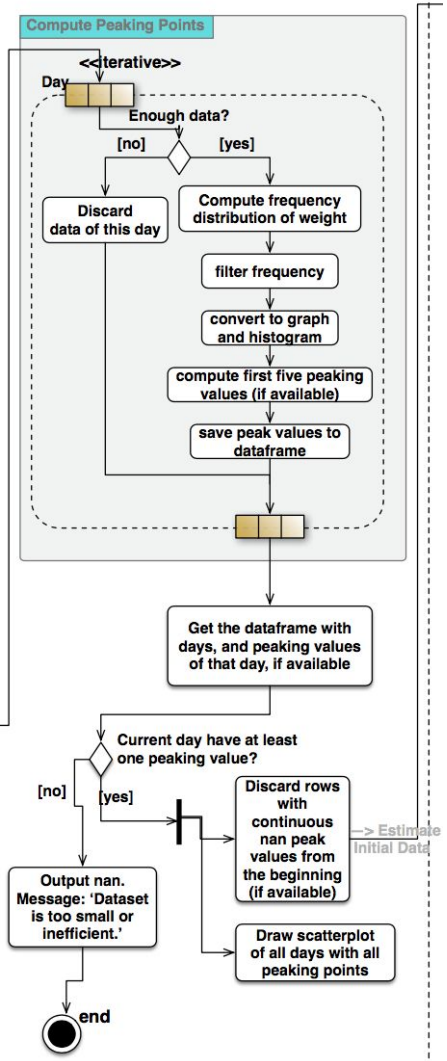
3) Split the dataframe

4) Preserve latest dataframe

Function: `df_split()`:
- split the dataframe



* In Pandas, DataFrame is a 2-dimensional labeled data structure with columns of potentially different types.

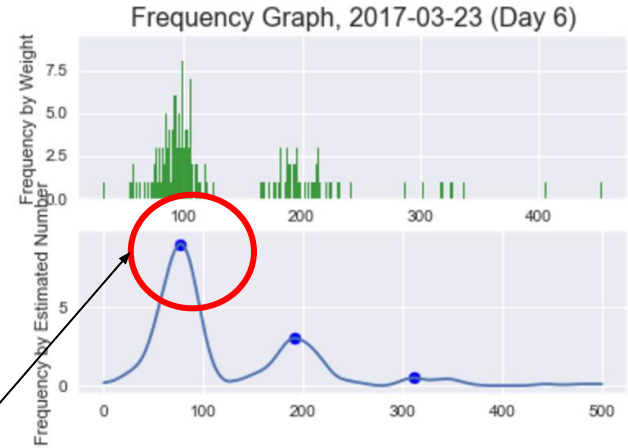


2. Sensor Processing

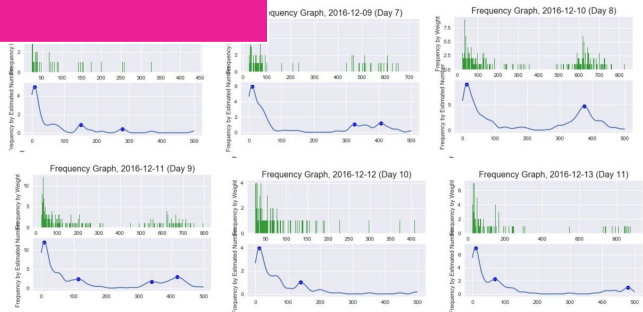
1) Process data **day by day**

2) For each day, compute frequency distribution and draw histogram

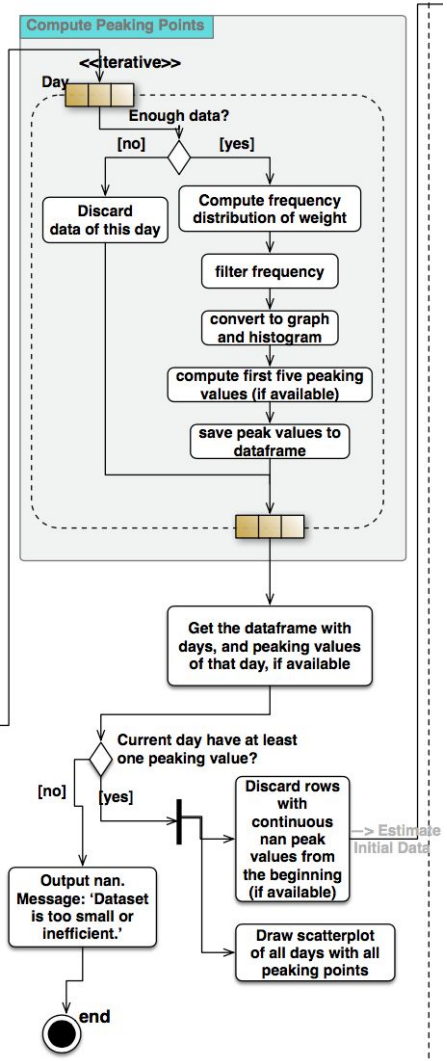
3) Find all peaking values



Biggest Problem 1:
How to find this peaking point???



Function: `find_peaking_point()`:
- For each day, find peaking points from histogram

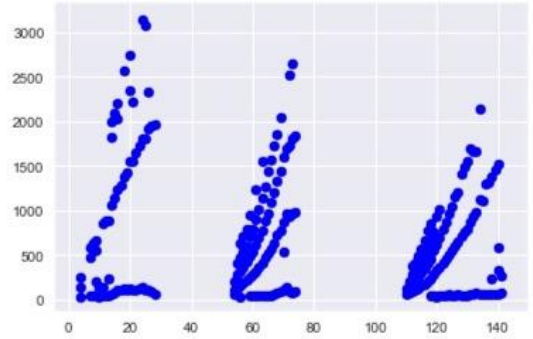


2. Sensor Processing (cont.)

Biggest Problem 2:
How to find the points in this line???

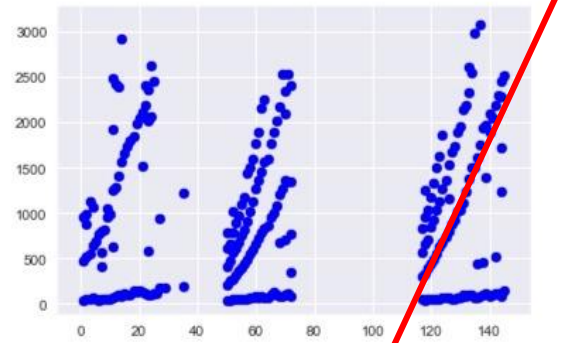
4) Get a scatterplot of all days' values

```
Begin processing Sensor1.
SUCCESS 584302380c51e0353c9d8e9c
Finished pulling data from api.
```




```
Finished peak finding and data prediction.
```

```
Begin processing Sensor2.
SUCCESS 584493520c51e008d3e88909
Finished pulling data from api.
```



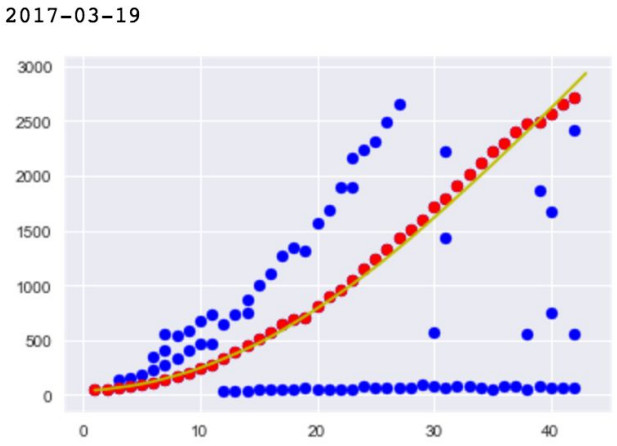

```
Finished peak finding and data prediction.
```

How to find the peaking point???

Discarded Approach 1:

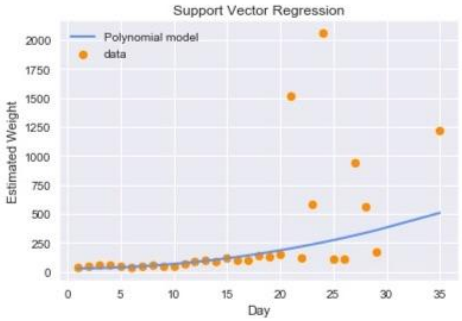
Using standard guide as compassion.

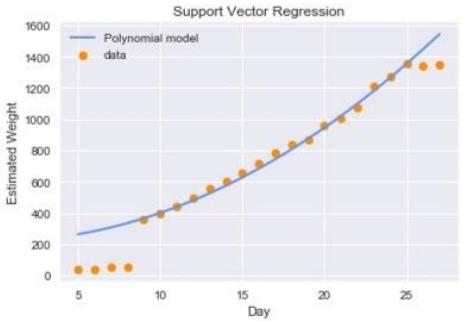
Find estimation based on standard data. Sometimes, the result can be very satisfying.



Discarded because:

1) Logic itself is dubious
2) Rely on batch date. Batch date may be unknown or inaccurate





Insights:
Standard guide is useful data. But it should be used more as data verification, instead of comparison.

How to find the peaking point???

Discarded Approach 2:

Do regression backwards.

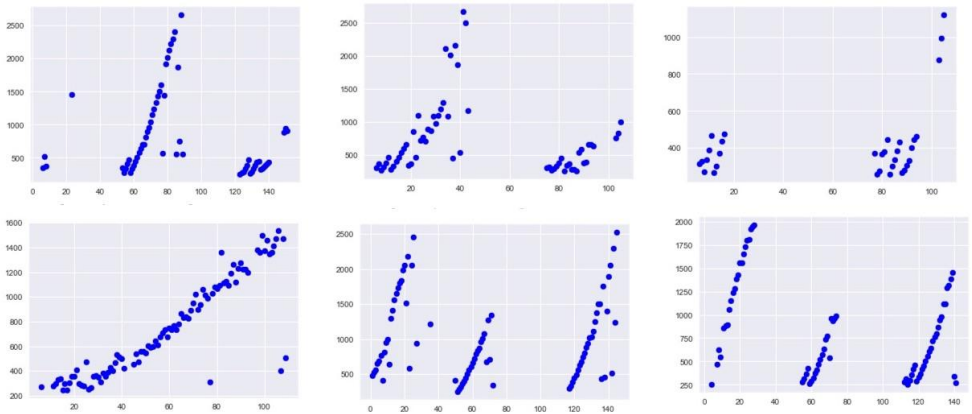
Discard noise value and data points less than 10 days weight. For remaining data points, choice the first data point as estimated weight. Then run regression to find data for those without an estimation (mainly data less than 10 days.)

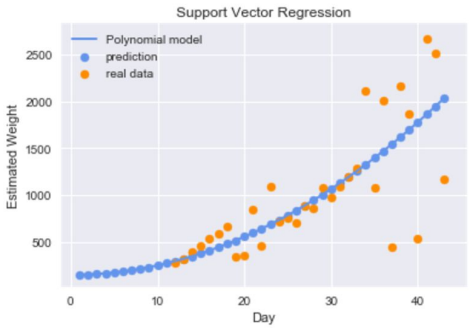
Discarded because:

- Data found might not be accurate.
- The system is automatic. We can only run regression forward, but we can not run it backwards.
- It relies on batch date, but sometimes batch date itself is inaccurate.

Insights:

This method should still be useful under some situations.





How to find the peaking point???

Discarded Approach 3:

Using a simple one-point method.

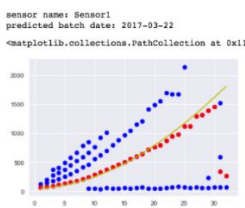
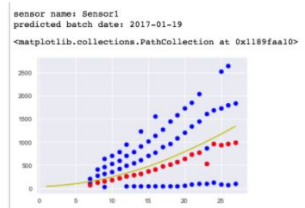
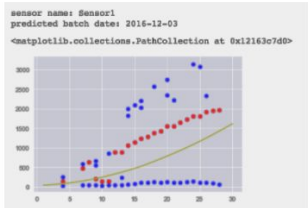
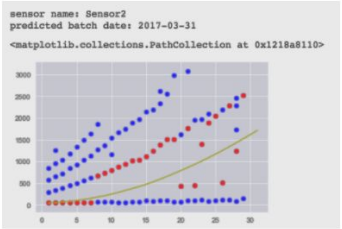
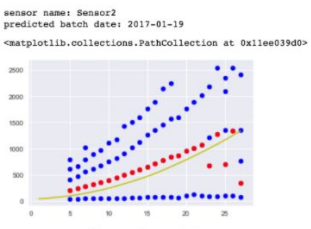
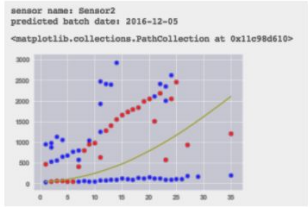
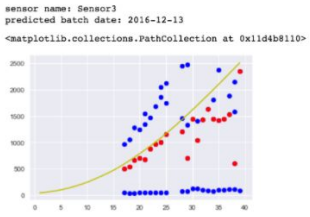
If **date < 10, we** discard weighting date less than standard (day 1); if **date > 10,** we discard weighting date less than standard (day 10). Then we can choose the smallest weight from remaining peaking point.

Discarded because:

- Estimation is not accurate.
- It relies on the batch date

Insights:

This might be a good approach. It just needs some adjustment for those estimation that is not accurate.

Name			
'Sensor 1'		 Given Batch Date	
'Sensor 2'		 Given Batch Date	
'Sensor 3'	 Given Batch Date		

Link: All results

How to find the peaking point???

Discarded Approach 4:

Using an iterative method.

Firstly, find all values of weight (both peaking point, and intervals), and find the range of the weight. Then loop the program within weighting data to find the best guess with minimum value:

$$\Sigma weight(abs(weight - estimation * round(weight/estimation)))$$

Discarded because:
Rely a lot on initial data quality. If the initial intervals are not correct, we cannot find the right estimation anyway.

Insights:
This method can be used to adjust the final estimation to make results more accurate.

How to find the peaking point???

Discarded Approach 5:

Putting all methods together.

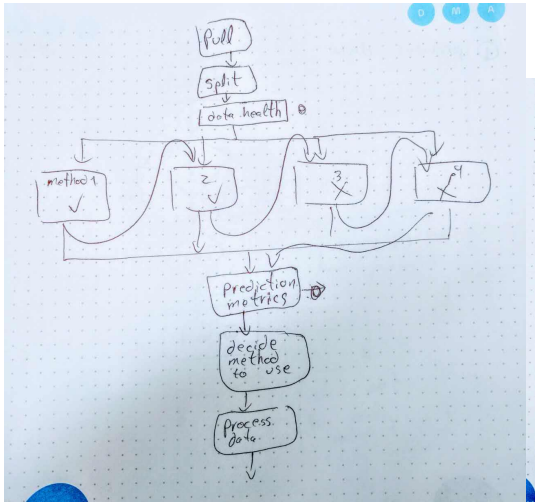
Method 1: One-point Method (Needn't assume batch da
Method 2: Standard Comparison Method (Assuming ba
date)
Method 3: MSE Method (Needn't assume batch date)
Method N: ...
Validation: if the output is compellent.
Regression: Regression Method (for day > 4)

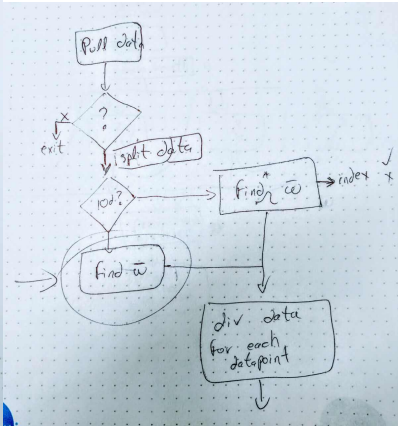
Discarded because:

- Sometimes we cannot find right estimation anyway using all methods.
- Validation cannot prove anything

Insights:
We can put all those methods together.

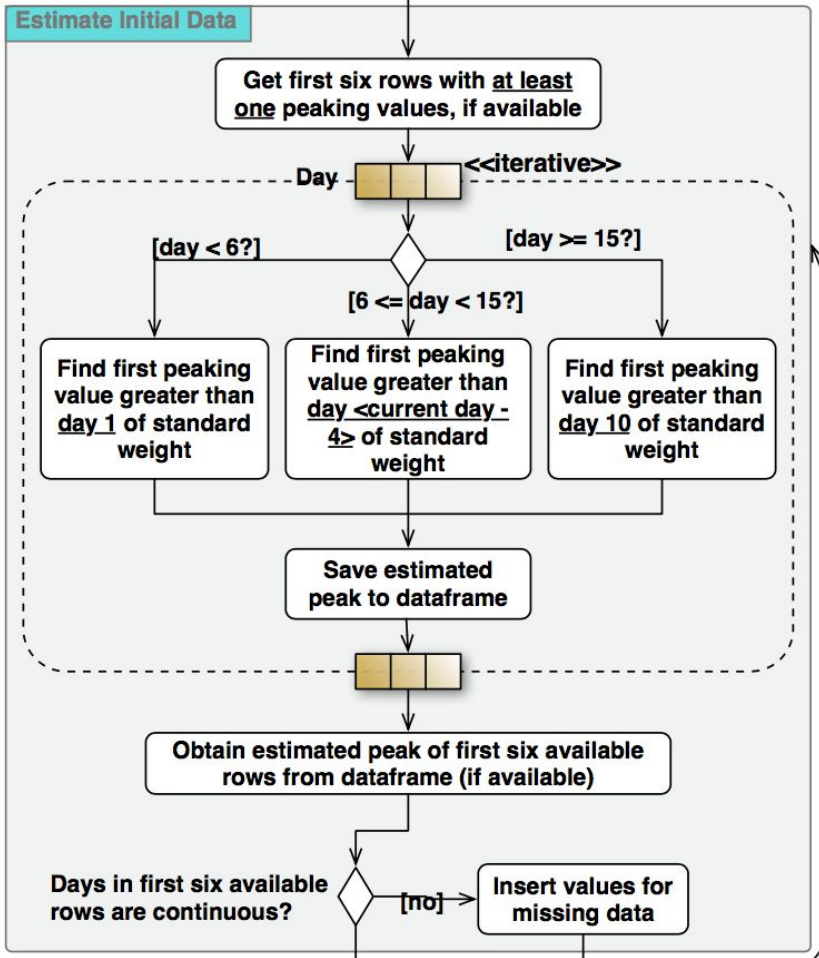
Try **method 1 + Regression**
| _____
|→| validate |---------→ pass **validation** → Output
| method 1 | |---→ reject **validation**
|→ If there's → **Try method 2 + Regression**
| a batch date | _____
|→ | validate |→ pass **validation** → Output
| method 2 | |→ reject **validation**
| |
| | ----------|
|
|--> If there's not → **Try method 3 + Regression**
a batch date | _____
|→| validate |→ pass **validation** → Output
| method 3 | |→ reject **validation**
|--> Output error message





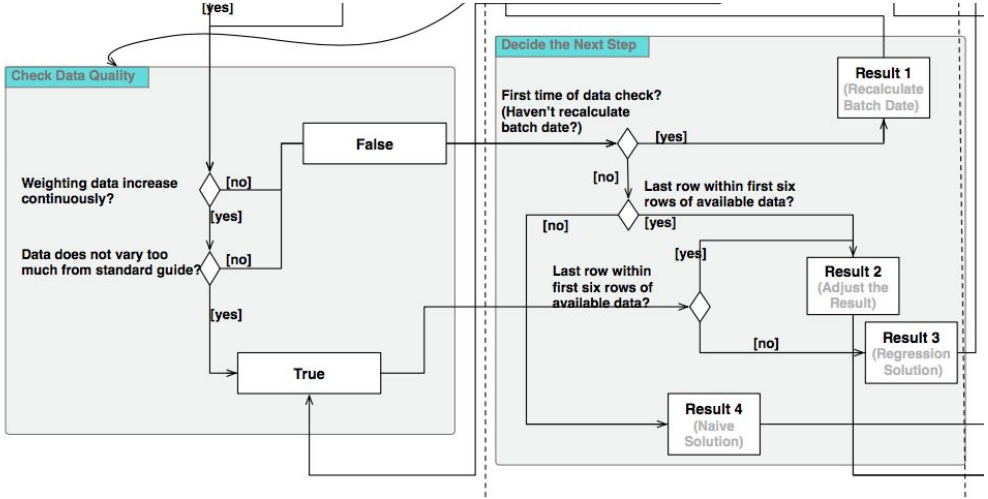
The final approach...

3. Weight Estimation in Initial State



Get **first six days of weight** estimation using the simple method, if these data exists. Insert data if they do not exist.

Function: `estimate_initial()`:
- Estimate the peaking point of first six days of data, if it exists.



4. Data Check/ Parameters Recalculation

1) Do data check

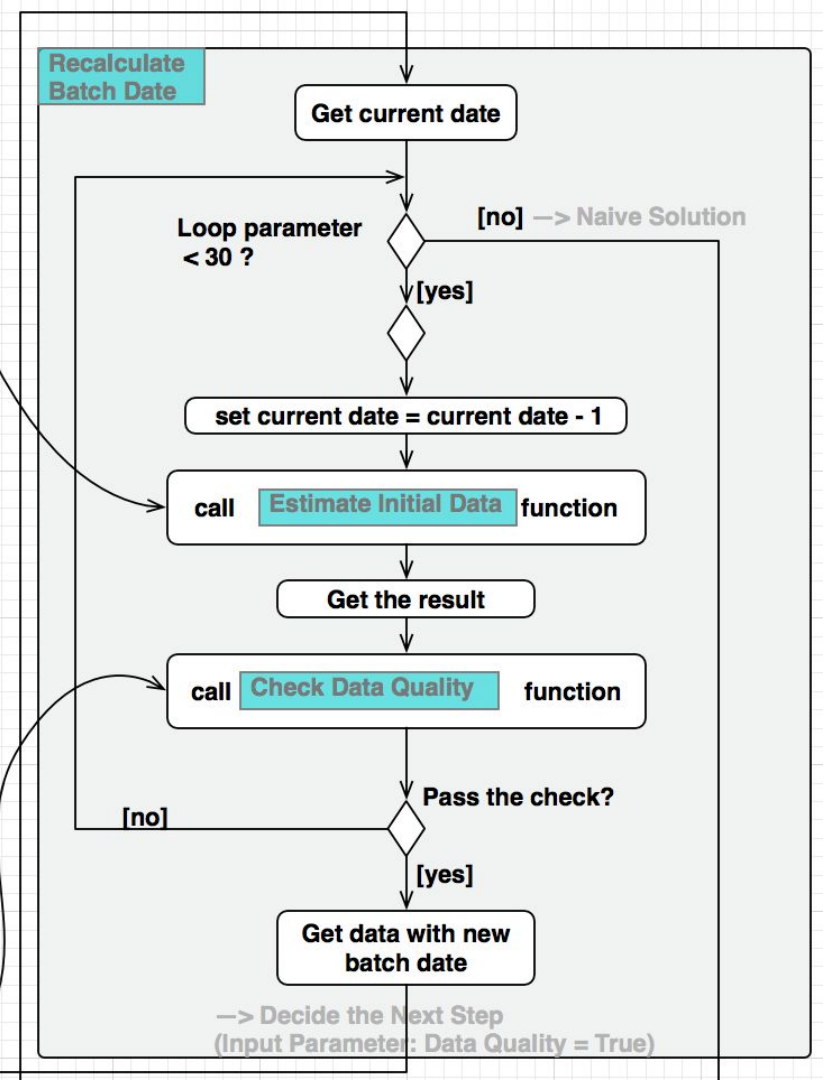
- **Check the quality** of first six days of data estimation
- Check if it is first time of batch date calculation
- Check if data belongs to first six days of data

Function:
`check_data_quality()`:
- Check if first six days of data estimation is acceptable

Function:
`decide_next_step()`:
- Decide which route the program should go on

2) Get four result choices:

- **Result 1:** Recalculate batch date ⇾ call `recalculate_batch_date()`
- **Result 2:** Adjust the result -⇾ call `adjust_result()`
- **Result 3:** Regression solution -⇾ call `find_peaking_point_regression()`
- **Result 4:** Naive solution -⇾ call `find_peaking_point_naive()`



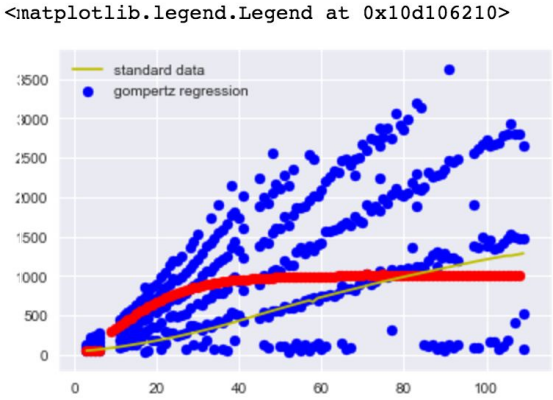
4. Data Check/ Parameters Recalculation (cont.)

Recalculate batch date.

Batch date should only be calculated once.

Reason:

- Almost all methods need the accuracy of batch date date
- We always get a bad estimation when batch date is wrong..



Function:

`recalculate_batch_date()`:

- Recalculate batch date

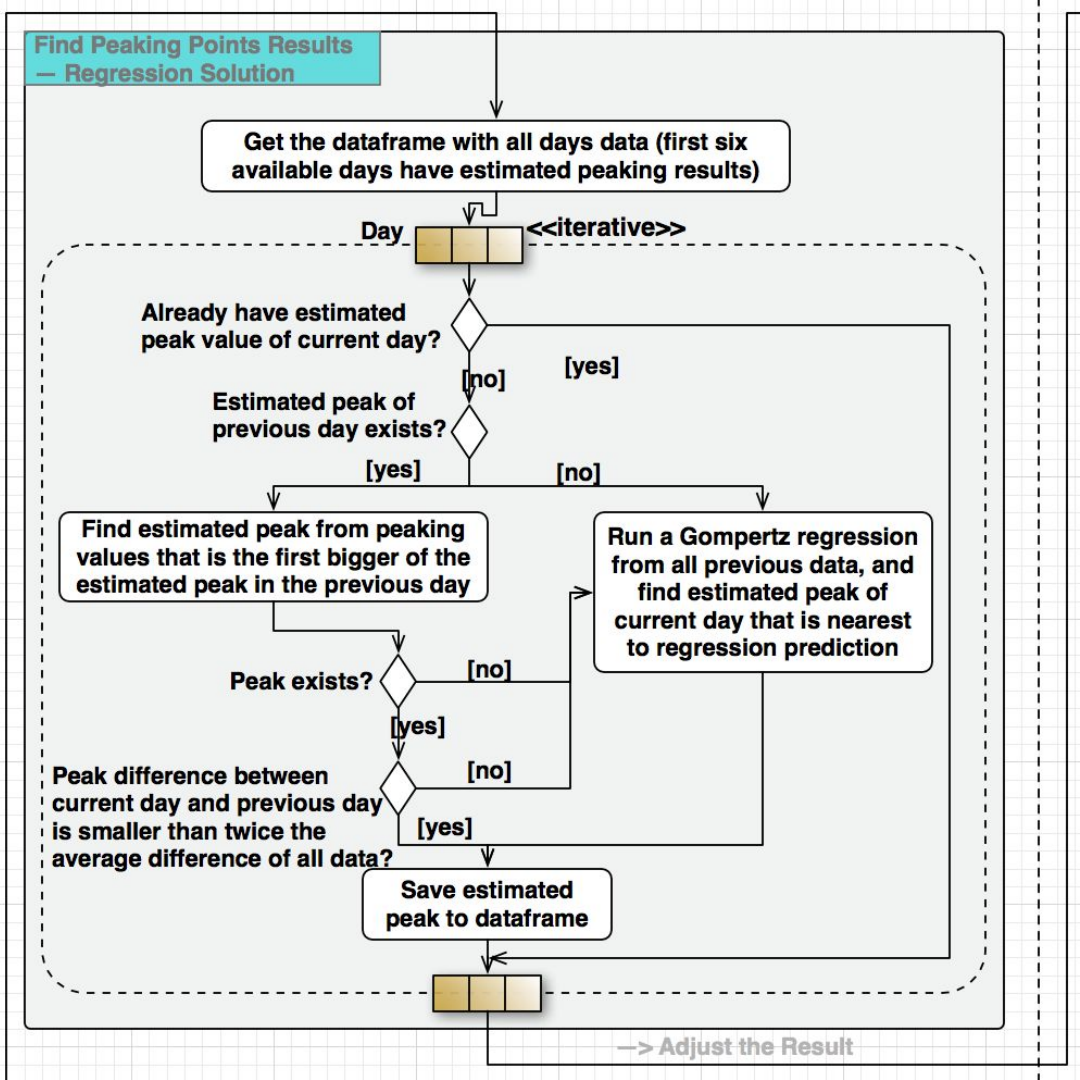
5. General Weight Calculation (1)

Using the **regression approach**.

1. Main plan: Self-created simple regression function
2. Backup Plan: Gompertz regression function

Do checking after regression.

Function:
`find_peaking_point_regression()`:
- Find peaking points using a regression approach

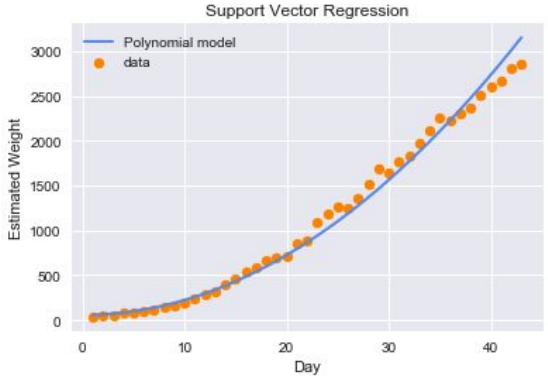


Which Regression??

Approach 1: SVR* Regression

Pros: Accurate
Cons: Very slow

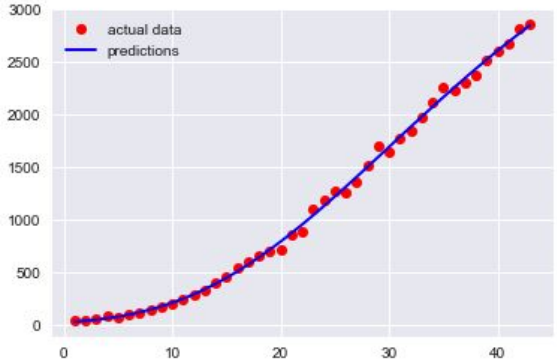
* SVR is short for Support Vector Regression. We use the library in sklearn: [sklearn.svm.SVR]



Approach 2: Gomperts Regression

Pros: Fast; accurate in most cases
Cons:

- Can not call the function for many times
- May encounter Overfitting problem;



Approach 3: Self-created Simple Regression

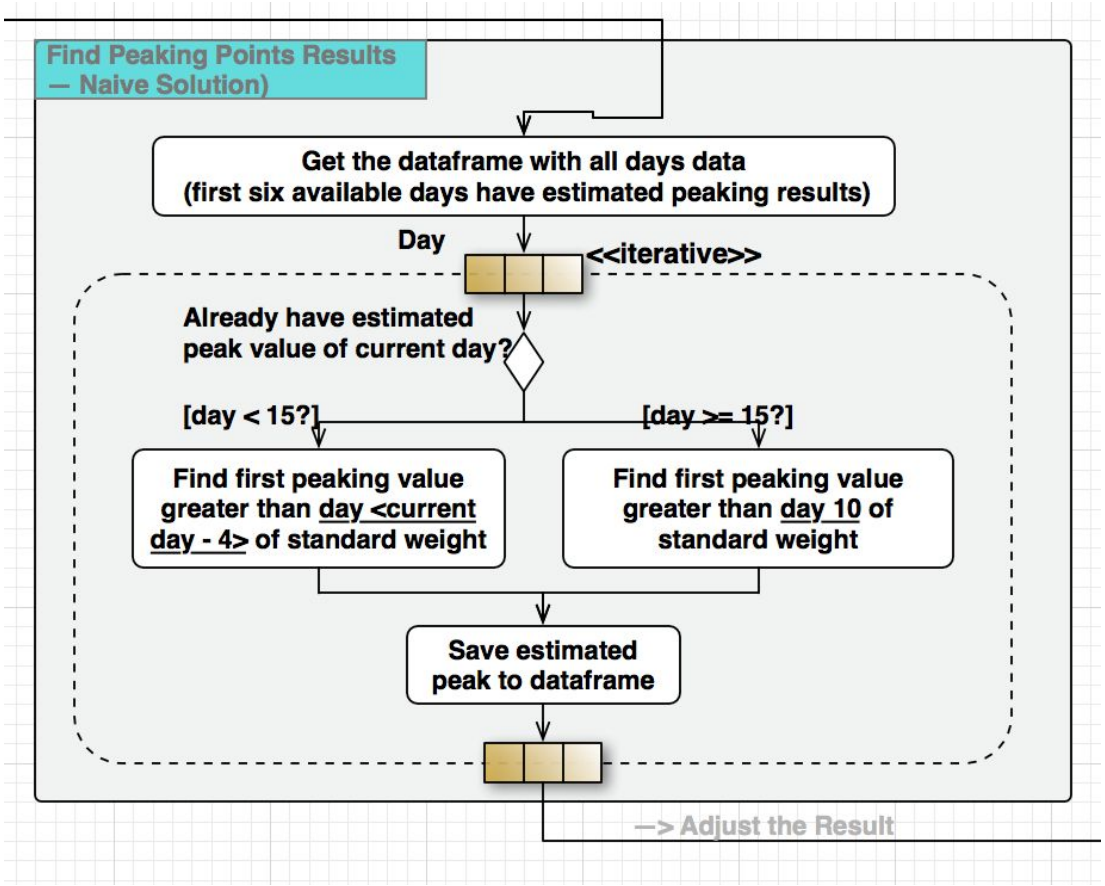
Pros: Easy and quick
Cons: Not so accurate

5. General Weight Calculation (2)

Using a **naive approach**.
(When we testify that data quality is not satisfiable and we can not use regression to get the result)

Do checking after regression.

Function:
`find_peaking_point_naive()`:
- Find peaking points using a naive approach

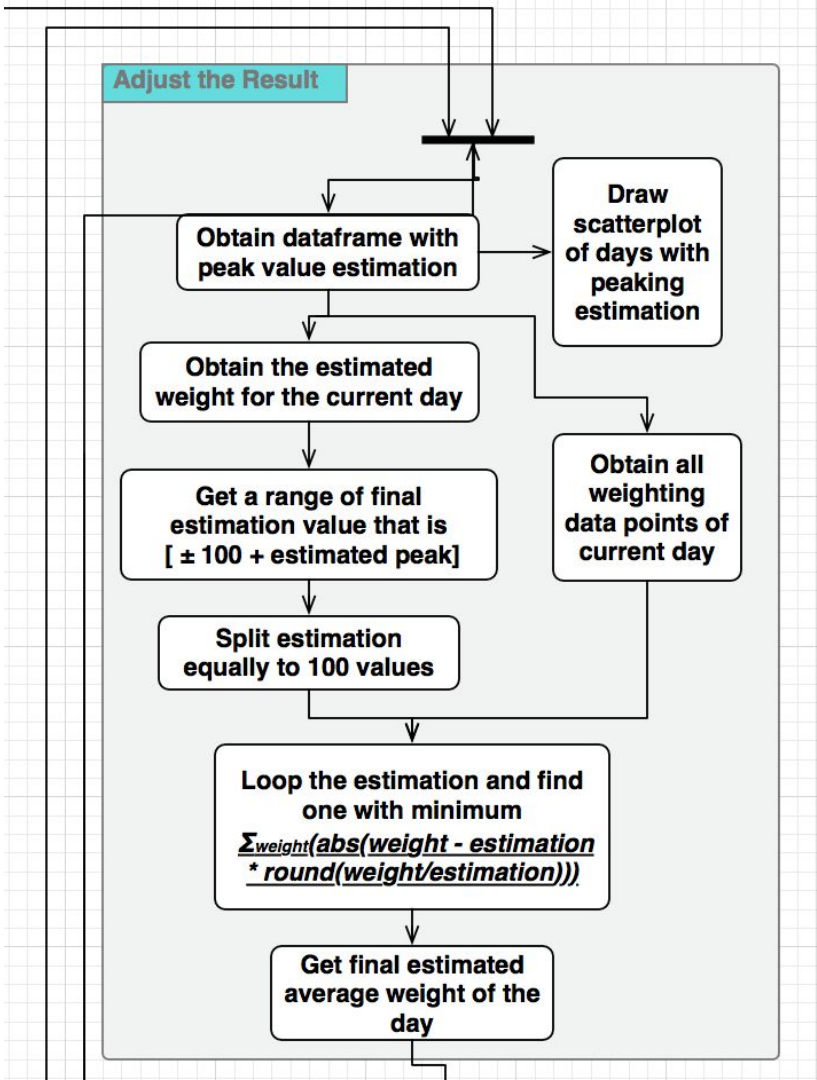


6. Result Adiustment and Validation

Adjust the result using an iterative brute force approach.
Using the formula:

$$\Sigma weight(abs(weight - estimation * round(weight/estimation)))$$

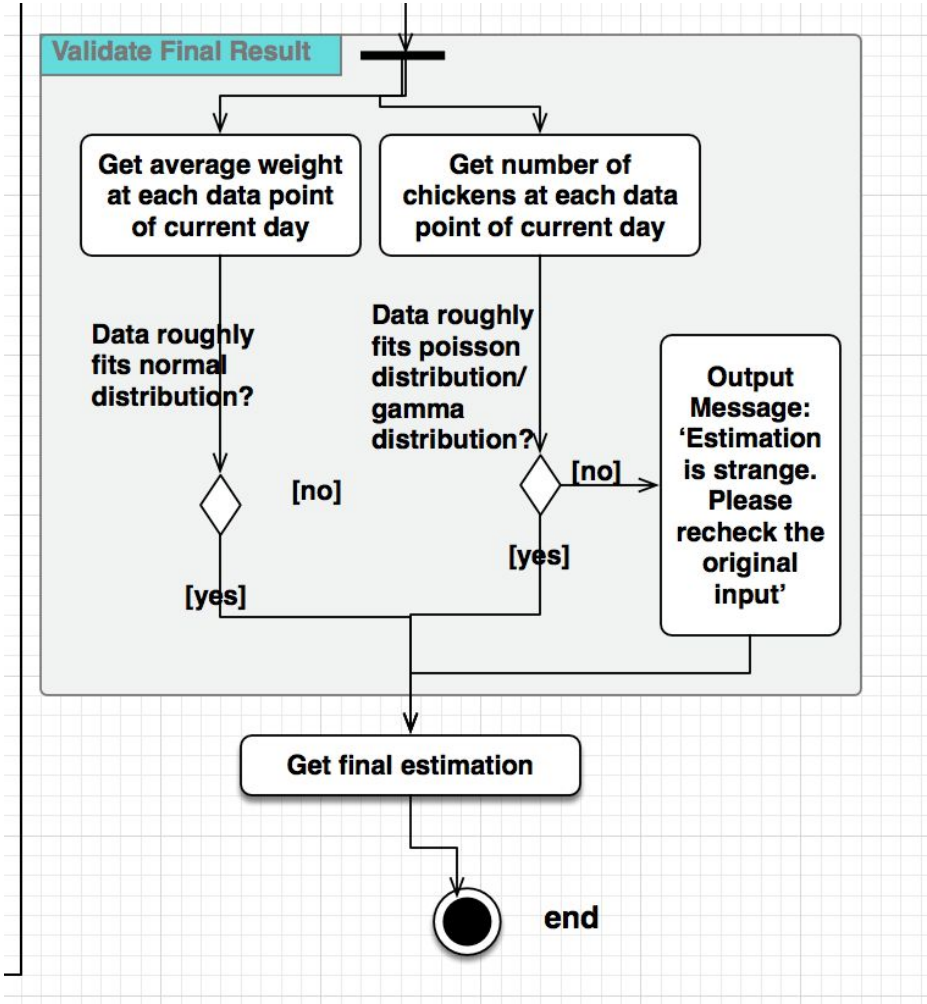
Function:
`adjust_result()`:
- adjust weight estimation to a more precise figure



6. Result Adiustment and Validation

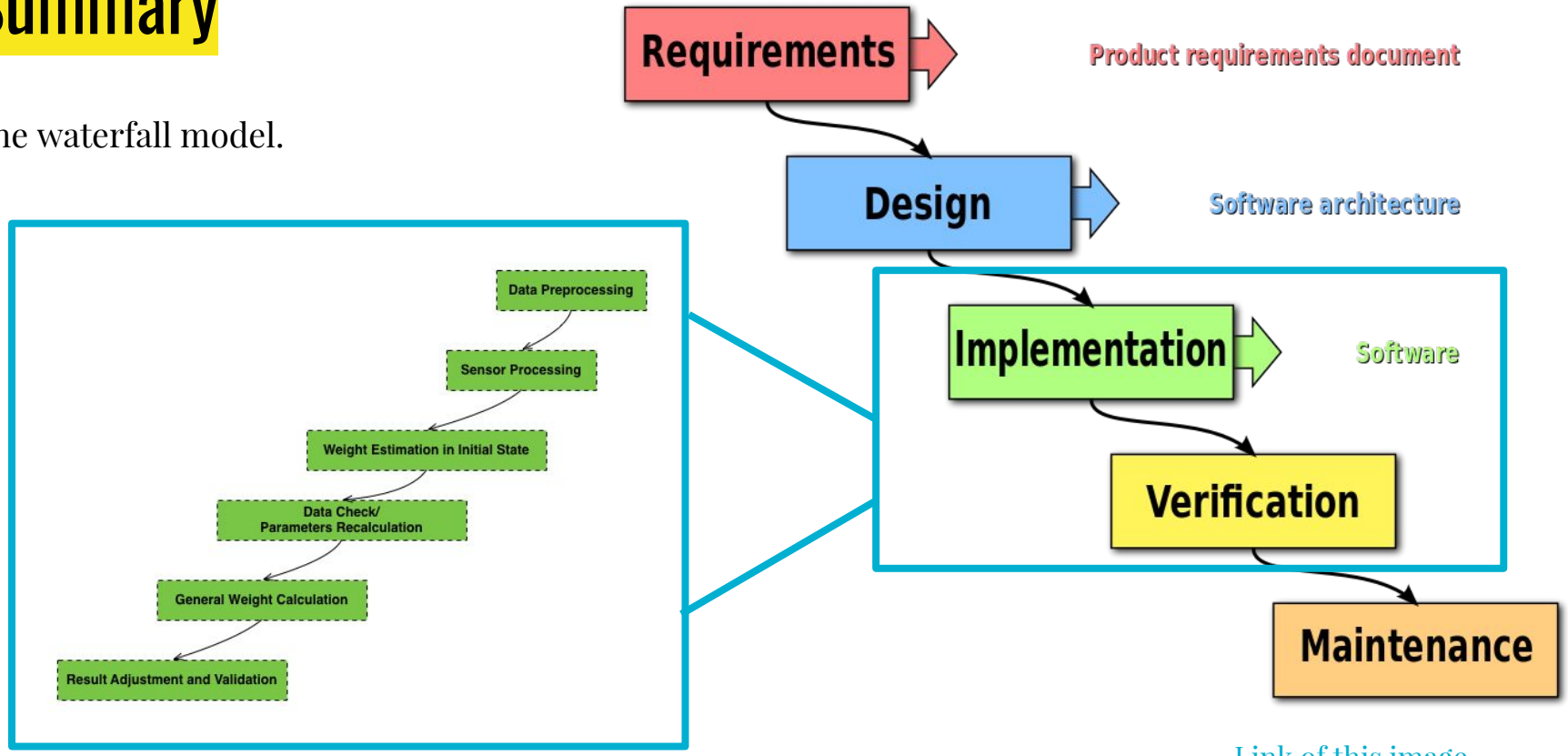
Validate final result.
Output message if anything is strange.

Function:
`validate_result()`:
- Validate final result



Summary

The waterfall model.



Link of this image

Reflection

Four major problems I encountered:

1. Control dev process
2. Design properly
3. Code readability
4. Process Continuity
5. Communication