Use PlanOut to Design an Online Factorial Experiment

Xiaotian Wang

Department of Statistics, University of Wisconsin Madison

Final Project Report for Course STAT 803

Abstract

PlanOut is a framework for online field experiments. We use PlanOut to design an online 2n factorial experiment. To figure out how do the users like the registration user interface, we choose font, font size and the button text as our factors to design the experiment. We design a web-based app to see user’s preference on registration interface. Then, more about PlanOut including PlanOut language, namespaces, strengths and weaknesses are discussed.

Keywords: PlanOut, A/B test, online experiment, factorial design, interface

Use PlanOut to Design an Online Factorial Experiment

In this project, we use PlanOut to design an online 2n factorial experiment to find out the preferred user interface when registering accounts for the users. More about the PlanOut platform is discussed.

# 1 Introduction

## 1. 1 PlanOut

PlanOut is a framework for online field experiment developed by Facebook. It was created to make it easy to run and iterate on sophisticated experiments in a statistically sound manner while satisfying the constraints of deployed Internet services.

PlanOut provides randomized parameter values to Internet services. Instead of using constants for user interface elements or switches controlling the rollout of a new feature or ranking model, with PlanOut, we only need to focus on the value of these parameters.

It is easy to use PlanOut to design simple A/B tests and other factorial or withing-subsects designs. Some more complex designs involving multiple types of units can also be handled by PlanOut.

PlanOut is written with portability and extensibility in mind and has been ported to all major Web programming languages, including Python, Java, PHP, JavaScript, Ruby, and Go.

The PlanOut framework includes:

* Extensible classes for implementing and testing experiments, which automatically log important data.
* A system for managing and deploying multiple mutually exclusive experiments, called namespaces.
* The PlanOut language, which lets you define, serialize, store, and execute experiment definitions in a platform-independent way.

(Note: most of the introductions to PlanOut are from the official website of PlanOut.)

## 1.2 Our Experiment

In this project, we have an online 23 factorial design to find out what are the better user interfaces (UI) when registering account for a website. We consider three factors that may be adjusted when designing the UI: font, font size and the button text. The levels are as the table below:

Table 1: Factors and levels

|  |  |  |
| --- | --- | --- |
| Factor | High Level | Low Level |
| Font | Verdana | Times |
| Font Size | 5 | 2 |
| Button text | Join Us | Sign Up |

We build the web-based app to form the registration interface and to test how the users like them. So, we have totally 23=8 different version of webs for the users to register their account to get our results. To know how the users like the interface, we also allow the users to rate our interface in these webs. The users can rate this interface by inputting a number 1(worst)-5(best). We will log the ratings from the users over time to see which kind of UI design is the best one, and how these factors will influence the results.

Our low-low-low level interface and the High-high-high level interface are as shown below：

![Graphical user interface, text, application, email

Description automatically generated]()

Figure 1: high-high-high level registration interface

![Graphical user interface, application

Description automatically generated]()

Figure 2: low-low-low level interface. (Note: for the second interface, the font size is much smaller than the first one, so the texts only occupy approximately a quarter of the whole interface.)

With the ratings from the users, we can analyze the results in a period of time to figure out how these factors will influence the attitude of the users to the registration interface.

# 2 Implementation

We first design the experiment using the PlanOut platform. In this platform, a class SimpleExperiment is provided to generate some experiments with simple functions. Here we wish the high- and low-level factors can be chosen by users randomly with equal probabilities, so we use the UniformChoice function from this planform to generate experiments meeting the requirements.



We use Flask package to generate the web-based app. Now we design the html page with the variables: font\_size, font, and button\_text to design 8 different pages. The code for the web-based app is attached in the appendix.

When a user clicks in the webpage, he/she will be allocated a unique user\_id. The PlanOut platform will automatically hash the id and allocate the corresponding interface to the user. Then the user can rate the interface by 1-5 level.

The HTML code of the rating and the interface are as below:



Figure 3:rating input box

All the actions from the users will be logged. They will be saved to a “.log” file. Analysis can be implemented by analyzing the data of the log file.



The data structure of the log file is as shown below:

When a user clicks in the webpage, the following information will be logged,



and when he/she finishes the rating(say he/she rates 3 score), the following information will also be recorded:



After collecting the data, the time, user\_id, parameters and the ratings are all provided. With these data from a period of time, we can implement the analysis to get some useful results.

# 3 More about PlanOut

## 3. 1 PlanOut Language

PlanOut language is a parsimonious, high-level language for thinking about and specifying complex experiments in a way that is useful to researchers as well as engineers building production systems. For researchers or engineers from different backgrounds in a same project, it is probable that they use different programming language for programming. With PlanOut Language, they can design or edit experiment with it because it is simple enough to use and understand. Additionally, PlanOut language is the key aspect to serialize the experiments. For example, when implementing large scale experiments, we may have many experiments with different designs. With PlanOut language, it is much easier to serialize the experiments and the administration will become easier.

The grammar of PlanOut language is similar to JavaScript. It can be compiled into JSON file and stored. With Interpreter API from PlanOut package, the stored JSON files (the experiments) can also be interpreted to be experiments and then be called directly.

In the example in this project, the structure of the experiment is not too complicated, so we do not use the PlanOut language. Instead, we only use the python-based PlanOut API.

However, if we use the PlanOut language to design the experiments in our project, the code will be like



If it is compiled to JSON file, for demonstration purpose, the first line of the code



will be compiled to be



This JSON file can be also interpreted to generate the experiments and then be called.

## 3. 2 Namespaces

Namespaces are used to manage related experiments that manipulate the same parameter. These experiments might be run sequentially (over time) or in parallel. Namespaces can be used to keep experiments “exclusive” or “non-overlapping”.

Namespaces objects are like the Experiments objects, but randomly assign primary units (e.g., users) to experiments. Generally, namespaces are used whenever there is a variable used to experiment with. However, the namespace class of PlanOut platform is useful only when there are more than one experiments needed to be administrated. For the experiments in our project, the structure is so simple that we only need to use SimpleExperiment class. If we want to use the namespace in our project, we can first define a namespace class, and then add experiments to the class.

# 4 Conclusion

## 4. 1 Strengths

The first strength of PlanOut is the randomization. It uses pseudo-random assignment through hashing, which make every experiment independent. Besides, it provides a “salt” method that help developer manually decide (if necessary) if some two experiments are needed to be correlated.

The second strength of PlanOut is that it is a way to serialize experiments. We know in production, there will be a large number of experiments working simultaneously. If we can generate many experiments different by several line of code, it will save us a lot of works.

Another strength is about the namespace class of the platform. With a large number o experiments, it is important to administrate them in a proper way and make them “exclusive”.

## 4. 2 Weakness

This platform does not provide any tools to analyze the data collected. So, for the developers, they need to design the analysis themselves. For the large-scale simultaneous experiments, automatic analyses are not easy. More future works on automatic analyses tools are needed, especially when some changes are made to the experiments. In fact, changes always happen when new knowledges are acquired.

According to the main page of PlanOut, the platform has been tested on Mac and Linux but not on Windows, and thus, some problems may occur when running the code from the official tutorial on Windows. For example, when compiling the PlanOut language to JSON, we can not use the node function in Linux shell scripts directly on Windows command line. Additionally, the tutorial is based on Python 2 which is outdated. Some problems may occur when running the code on Python 3.

# Reference

[1] Bakshy, E. and Eckles, D. and Bernstein, M. S., Designing and Deploying Online Field Experiments, Proceedings of the 23rd ACM conference on the World Wide Web, 2014

[2] https://facebook.github.io/planout/index.html

# Appendix: Code Used for the Experiment

1. **import** random
2. **from** uuid **import** uuid4
3. **from** flask **import** (
4. Flask,
5. session,
6. request,
7. redirect,
8. url\_for,
9. render\_template\_string
10. )
12. **from** planout.experiment **import** SimpleExperiment
13. **from** planout.ops.random **import** \*
15. app = Flask(\_\_name\_\_)
17. app.config.update(dict(
18. DEBUG=False,
19. SECRET\_KEY='3.14159', # shhhhh
20. ))



25. **class** MyExperiment(SimpleExperiment):
26. **def** setup(self):
27. self.set\_log\_file('register\_webapp.log')
29. **def** assign(self, params, userid):
30. params.font\_size = UniformChoice(choices =['2','5'],unit = userid)
31. params.font = UniformChoice(choices =['Times','Verdana'],unit=userid)
32. params.button\_text = UniformChoice(choices=['Sign Up','Join Us'],unit=userid)

35. @app.route('/')
36. **def** main():
37. # if no userid is defined make one up
38. **if** 'userid' **not** **in** session:
39. session['userid'] = str(uuid4())
41. register\_exp = MyExperiment(userid=session['userid'])
42. size = register\_exp.get('font\_size')
43. font = register\_exp.get('font')
44. button\_text = register\_exp.get('button\_text')
46. **return** render\_template\_string("""
47. <html>
48. <font size={{size}} face="{{font}}">
49. <h1>Join Us Now</h1>
50. <form action = "/register" method="GET">
51. <label for="email">Email:</label>
52. <input type="text" id="fname" name="fname"><br><br>
53. <label for="password">Password:</label>
54. <input type="text" id="lname" name="lname"><br><br>
55. <input type="submit" value= '{{button\_text}}'>
56. </form>
58. <form action="/rate" method="GET">
59. <p>How do you rate our register interface? please input 1(worst)~5(best)</p>
60. <input type="text" maxlength="1" name = "rate"></input>
61. <input type="submit" value = "rate"></input>
62. </form>
64. <br>
65. <p><a href="/">Reload without resetting my session ID. </a></p>
66. <p><a href="/reset">Reset my session ID so I get re-randomized into a new treatment.</a></p>
67. </body>
68. </html>
69. """, size=size, font=font, button\_text=button\_text)
71. @app.route('/reset')
72. **def** reset():
73. session.clear()
74. **return** redirect(url\_for('main'))
76. @app.route('/register')
77. **def** registered():
78. **return** render\_template\_string("""
79. <html>
80. <head>
81. <h1>Success! Welcome!</h1>
82. </head>
83. <body>
84. <title>Success! Welcome!</title>
85. <p><a href="/">Back</a></p>
86. </body>
87. </html>
88. """)


92. @app.route('/rate')
93. **def** rate():
94. rate\_string = request.args.get('rate')
95. **try**:
96. rate\_amount = int(rate\_string)
98. register\_exp = MyExperiment(userid=session['userid'])
99. register\_exp.log\_event('rate', {'Rating': rate\_amount})
101. **return** render\_template\_string("""
102. <html>
103. <head>
104. <title>Thank you!</title>
105. </head>
106. <body>
107. <head1>Thank you!</head1>
108. <p>Your rating is {{rate\_amount}}. </p>
109. <p><a href="/">Back</a></p>
110. </body>
111. </html>
112. """, rate\_amount=rate\_amount)
113. **except** ValueError:
114. **return** render\_template\_string("""
115. <html>
116. <head>
117. <title>Thank you!</title>
118. </head>
119. <body>
120. <head1>Thank you!</head1>
121. <p>But some error occured. You need to input a number.</p>
122. <p><a href="/">Back</a></p>
123. </body>
124. </html>
125. """)

128. **if** \_\_name\_\_ == '\_\_main\_\_':
129. app.run()
130. erid);