Data Products & Flask Web Apps

[Tetyana Kutsenko]

July 19, 2016

Objectives

At the end of the day, you'll be able to:

- Describe example data product workflows
- Implement simple webpages using HTML and Flask
- Describe the HTTP methods GET and POST & list the differences
- Build a cross-platform, modern website using the Bootstrap framework
- Embed plots in your website using the bokeh package

Data Products & Workflow

- What do you deliver as a data scientist? (Think about case studies)

Data Products & Workflow

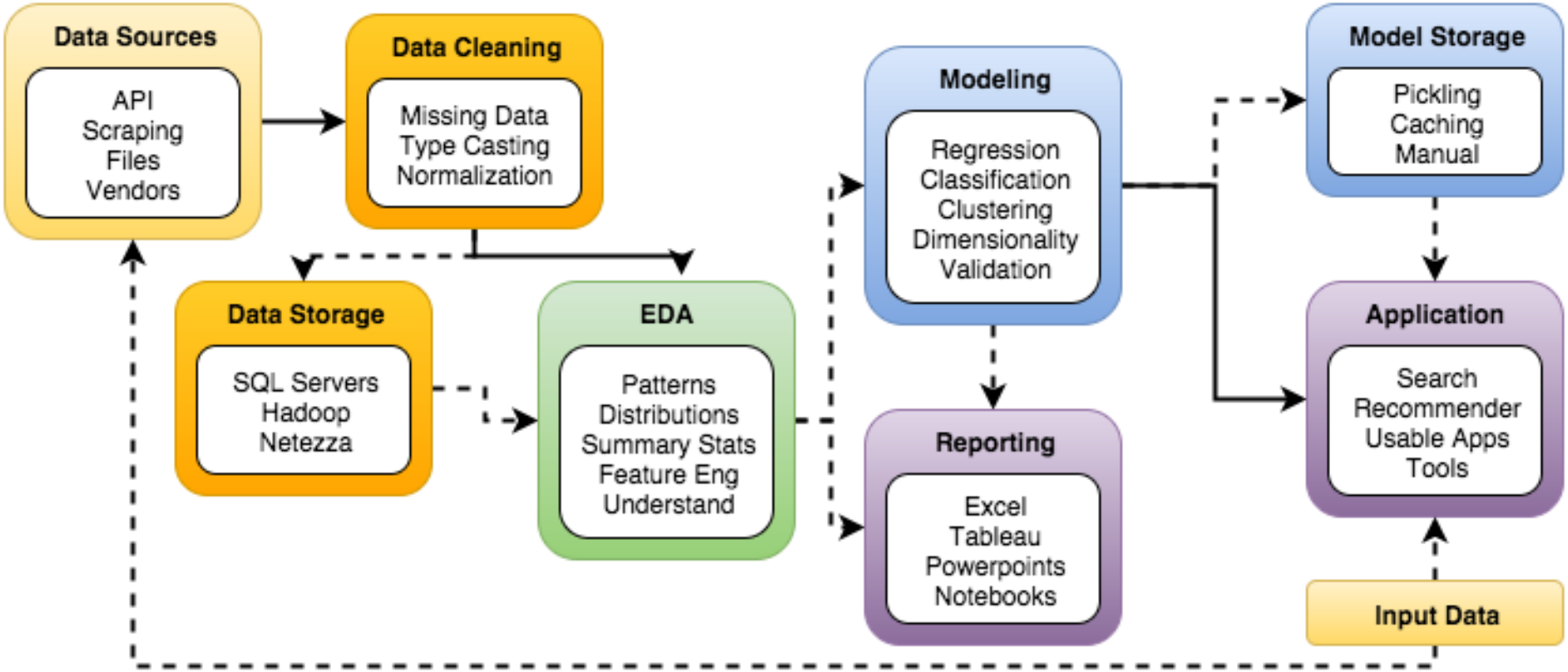


Figure 1:Data Products Flowchart

Image by Ming Huang

Why learn how to build web application

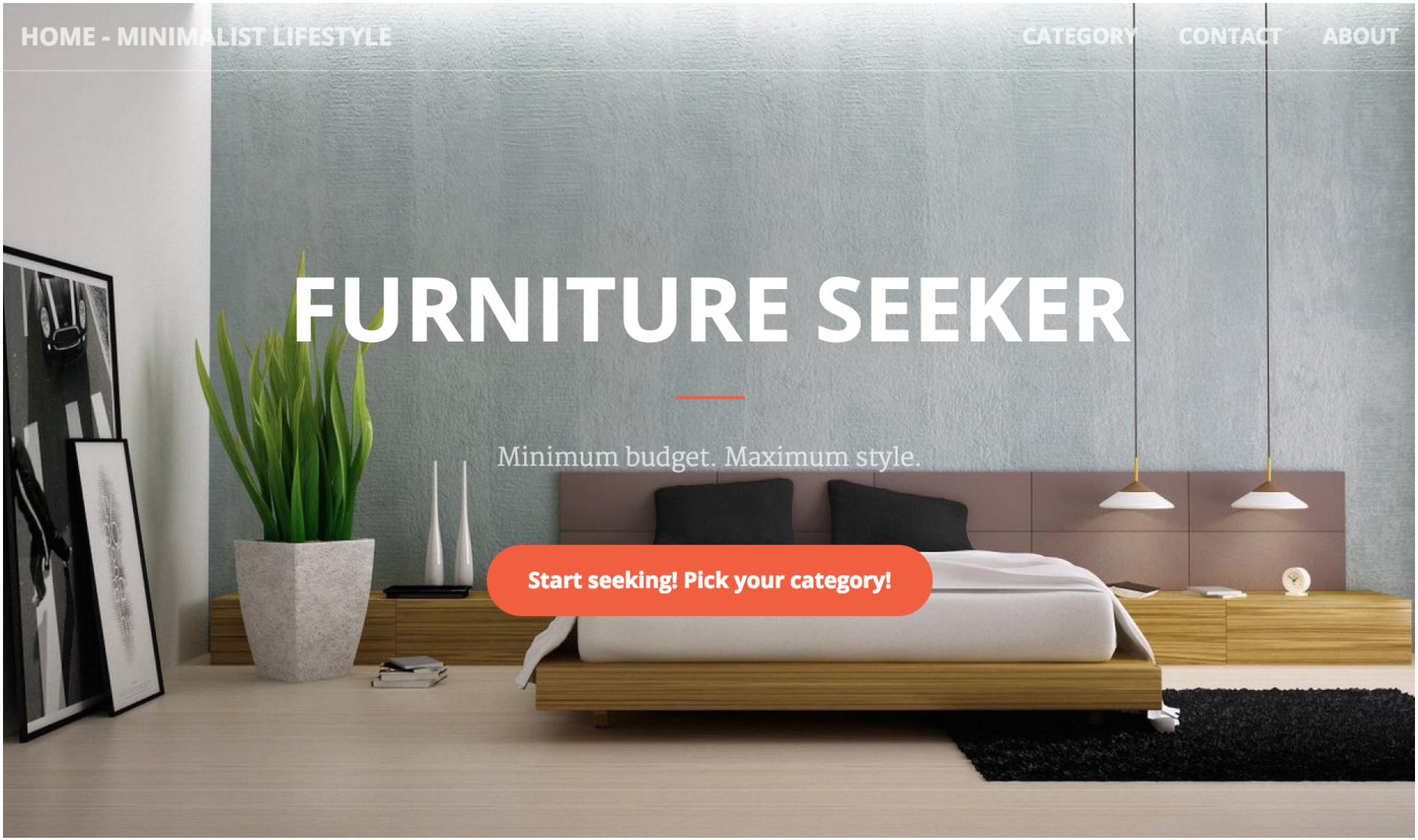


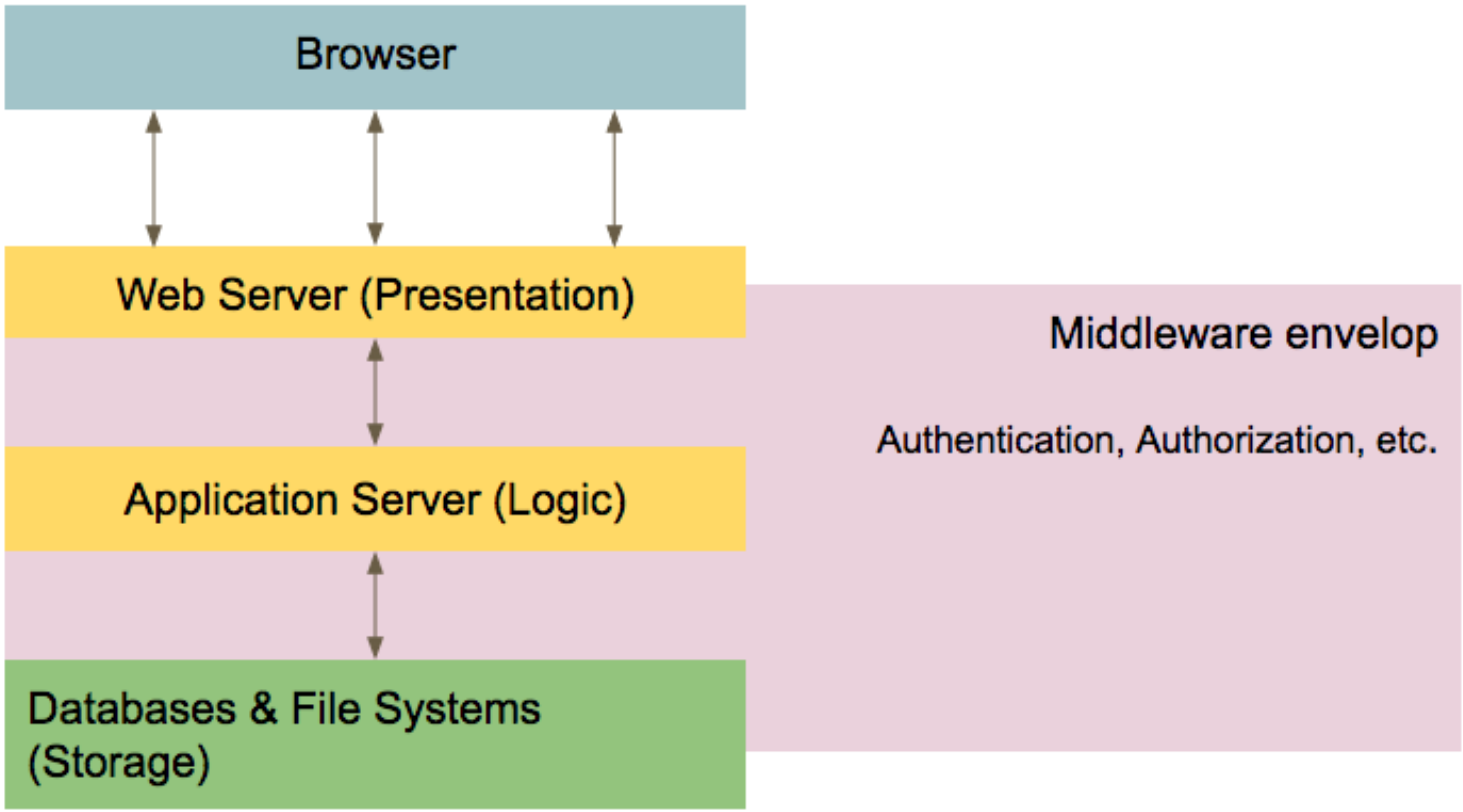
Figure 2:Furniture seeker

Why learn how to build web application

Figure 3:Furniture seeker

Web application

- Web application is a client–server software which is run in a web browser.
- Developing web application is simplified by frameworks such as Django, Ruby on Rails or Symfony.



HTTP methods: GET and POST

The Hypertext Transfer Protocol (HTTP) enables communications between clients and servers.

The two most common HTTP methods are:

- **GET**: Requests data from a server. (Default method in http & flask)
- **POST**: Submits data to server

Other HTTP Request Methods:

- PUT - Updates data on server
- DELETE - Deletes data on server

Important differences: see table at this w3 link

Review: HTML & CSS

Need basic HTML to build websites

- HTML (Hyper Text Markup Language)
  - Based on markup tags
  - Each tag describes different document entity
- CSS (Cascading Style Sheets)
  - Describes how HTML is displayed on screen
  - Assigns style properties to (sections of) your site
  - Can control the layout of multiple web pages all at once

Review: HTML & CSS

Your main reference today is W3 Schools. They cover:

- HTML
- CSS
- JavaScript
- Bootstrap

Example

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Page Title</title>
    </head>
    <body>
    <!-- page content -->
        <h1>My Page</h1>
        <p>
            All the things I want to say.
        </p>
        <p style="color: purple; text-align: right;">
            My right-aligned purple text.
        </p>
    </body>
</html>
```

Figure 4:HTML example

Flask

A Flask is a microframework for Python. "Micro" **does not mean**:

- Your whole web application has to fit into a single Python file (although it certainly can)
- Flask is lacking in functionality

It means:

- Flask aims to keep the core simple
- Flask won't make many decisions for you
- Decisions that it does make are easy to change

Installation

- **Install** using ‘pip install flask’

Jinja2 is a templating language for Python

- **Install** using ‘pip install Jinja2’

Flask Conventions

By convention

- templates - subdirectory for html template files
- static - subdirectory for files like css, js, font, images

Organize your files for flask (Reference)

Simple Flask application

```python
from flask import Flask
app = Flask(__name__)

# home page
@app.route('/')
def index():
    return '''
        <!DOCTYPE html>
        <html>
            <head>
                <meta charset="utf-8">
                <title>Page Title</title>
            </head>
          <body>
            <!-- page content -->
            <h1>My Page</h1>
            <p>
                All the things I want to say.
            </p>
          </body>
        </html>
        '''

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080, debug=True)

```

Figure 5:Flask application

Simple Flask application

Run ‘python example.py’

Open in browser ‘http://localhost:8080/’ or ‘http://0.0.0.0:8080/’

Routing

Routing is binding URLs to Python functions.

The route() call is used to bind a function to a URL.

```
4  # home page
5  @app.route('/')
6  def index():
7      return '''
8          <!DOCTYPE html>
9          <html>
```

Figure 6:Route

Routing

URL can contain variables (they are passed to bound function).

```
@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
    return 'User %s' % username

@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return 'Post %d' % post_id
```

Figure 7:Route variable

Routing

You can generate URL for route with url_for() function.

```
26
27  url_for('index')
28  url_for('login')
29  url_for('login', next='/')
30  url_for('profile', username='John Doe')
```

Figure 8:Route urls

Routing

By default, a route only answers to GET requests, but you can add the 'methods' argument to the route() call.

```
@app.route('/path', methods=['GET', 'POST'])
```

Figure 9:Route methods

Templates

- Generating HTML from within Python is not fun
- Template engine provides handy language to describe dynamic HTML
- Use render_template() from Jinja2 template engine

```
1  from flask import Flask, render_template
2  from random import random
3  app = Flask(__name__)
4
5
6  @app.route('/')
7  def index():
8      n = 100
9      x = range(n)
10     y = [random() for i in x]
11     return render_template('table.html', data=zip(x, y))
12
13
14 if __name__ == '__main__':
15     app.run(host='0.0.0.0', port=8080, debug=True)
16
```

Templates

```
1  <table border="1">
2    <thead>
3      <th>x</th>
4      <th>y</th>
5    </thead>
6    <tbody>
7      {% for x, y in data %}  <!--start for loop over variable data-->
8        <tr>                  <!-- on each row -->
9          <td>{{ x }}</td>    <!-- write variable x -->
10         <td>{{ y }}</td>    <!-- write variable y -->
11     </tr>                   <!-- end the row -->
12     {% endfor %}            <!-- end for loop -->
13 </tbody>
14 </table>
15
```

Figure 11:Flask application with template

Variables

Method *flask.render_template(template_name_or_list, context)* accepts context – the variables that should be available in the template.

- render_template('table.html', data=zip(x, y))
- render_template('hello.html', name=name)

From inside templates you can access *request* and *session* objects

- request.form['username']
- request.args.get('key', '')
- request.cookies.get('username')
- session['username']

Bootstrap: Introduction

Bootstrap is a popular front-end web framework combining HTML, CSS, & JavaScript.

- Easy way to develop modern web pages
- Cross-platform, including mobile
- Downloadable templates available at startbootstrap.com
- High quality results
- Free & open source

Bootstrap: Getting started

Start Bootstrap is resource with free Bootstrap themes and templates.

- Download a theme from startbootstrap.com & unzip
- You can start with bare template
- Match the file structure Flask:
  - Move the js, css, and fonts to 'static' folder
  - Move .html files to 'templates' folder
- Create flask application file .py
- Edit content in .html template files
- Run application

Bootstrap: Getting started

- Use the same .html template for all pages
- Don't forget to add routes and links to connect all new pages

Bokeh

Bokeh is a python library to create interactive plots.

- Display your data in a more pleasing way than a static image
- Update charts easily
- Users can interact with your charts

Installation

- **Install** using ‘conda install bokeh’ with all the dependencies that Bokeh needs
- If you have installed all dependencies you can use ‘pip install bokeh’. (It does not install the examples)

Use Bokeh in bootstrap/flask sites

Need to add the following two lines to the *.html template(s):

```
<link rel="stylesheet" href="http://cdn.pydata.org
/bokeh/release/bokeh-0.11.1.min.css" type="text/css" />
```

```
<script type="text/javascript" src="http://cdn.pydata.org
/bokeh/release/bokeh-0.11.1.min.js"></script>
```

Use Bokeh in bootstrap/flask sites

To add a bokeh plot to your site:

- Build figure in python app
  from bokeh.plotting import figure
  plot = figure(tools=TOOLS)
- Bokeh produces embeddable JavaScript that will render plot:
  from bokeh.embed import components
  script, div = components(plot)
  return render_template('dashboard.html', script=script, div=div)
- Add plot to template *{{ script | safe }} {{ div | safe }}*

The safe filter explicitly marks a string as "safe", i.e., it should not be automatically-escaped if auto-escaping is enabled.

Content

Don't steal content:

- Plenty of free-to-use images are available.
  - Google search options: filter images by usage rights
  - Flickr: license options in search
- Give your sources credit!

Post your website

Free options include:

- Python Anywhere
- Heroku

Paid options (free with credits) include:

- AWS