Build a Data-Driven Web App with Flask

@galuhsahid | galuh.me

Background

1. There are lots of cool codes and blog posts... but no way to try them unless you run the code yourself
2. The need to create a prototype/minimum viable product (MVP) that even non-technical users can use

```
$ git clone ...
```

```
$ jupyter notebook
```

From this:

Standardization and Transformation

In [35]:
```
scalerX = StandardScaler().fit(X_train)
X_train = scalerX.transform(X_train)
X_test = scalerX.transform(X_test)

y_train = np.log1p(y_train) # log transformation
```

In [47]:
```
len(X_test[0])
```

Out[47]: 17

Random Forest

In [36]:
```
rf = RandomForestRegressor(n_estimators=300)
rf.fit(X_train, y_train)
prediction = rf.predict(X = X_test)
prediction_ori_scale = np.expm1(prediction)
```

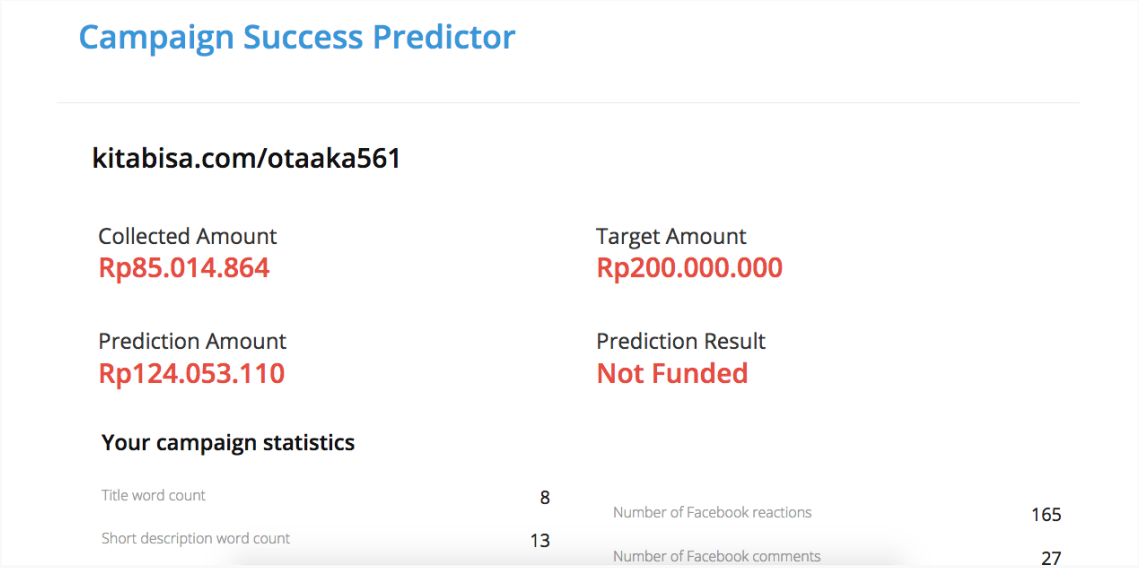
In [37]:
```
# RMSE
rmse = np.sqrt(np.mean((y_test-prediction_ori_scale)**2))
rmse
```

Out[37]: 14606034.523203889

In [38]:
```
# R^2
r2 = metrics.r2_score(np.array(y_test), np.array(prediction_ori_scale))
r2
```

Out[38]: 0.59164570215139789

To this:



Why Python?

1. Powerful data analytics libraries (pandas, numpy, scikit-learn...)
2. ... and web development frameworks (Flask, Django...)
3. Great community support

More time to focus on getting insights from your data and putting it out there

Flask

A microframework for Python

Great intro tutorials to Flask:

1. Flask
2. Flask Mega-Tutorial

... but why Flask?

Flask is simple and flexible:

```
from flask import Flask, render_template
from . import settings

app = Flask(__name__)
app.config.from_object(settings)

@app.route('/')
def index():
    return render_template('index.html')
```

However, it's always a good idea to assess what your application needs.

Feel free to follow along!

http://pyconid-demo.herokuapp.com

https://github.com/galuhsahid/pyconid2017

Based on "Estimating the Collected Funding Amount of the Social Project Campaigns in a Crowdfunding Platform"

Galuh Tunggadewi Sahid, Ivana Putri, Intan Sari Septiana, Rahmad Mahendra. ICACSIS 2017

Let's get started!

Input data

From user? External data?

Prediction

Done using the model we have built

Output

Display the result of our calculation and other stats

Tip #1: Think from the perspective of the users. What do they need?

/resources

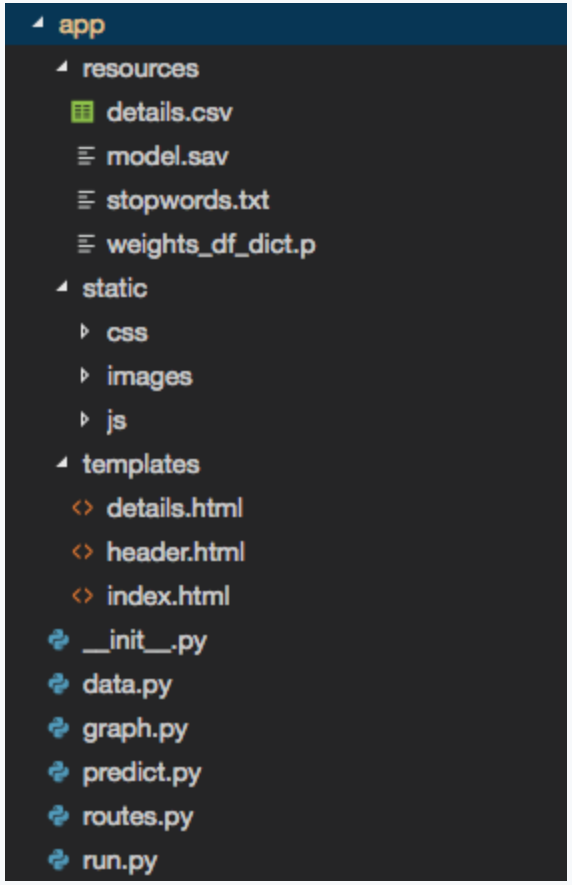
Your model and other files needed to process your data

/static

Where the stylesheet, JS files, and images go

/templates

Where the HTML files go



Input: data.py

Get data from campaign page and Facebook

Prediction: predict.py

Predict the amount of campaign donation based on scraped campaign data and Facebook data

Output: graph.py

Display a graph that explains how fundraisers can increase their predicted collected donation

Combining everything together: routes!

```python
@app.route('/')
@app.route('/index')
def index():
    return render_template('index.html')
```

```python
@app.route('/details')
def details():
    # 1. Get the campaign link
    # 2. Collect the data based on the given link
    # 3. Predict the donation
    # 4. Display the graph, result, stats, etc.

    ...

    return render_template('details.html',
                           campaign=campaign,
                           data=data,
                           fb_shares=fb_shares
                           )
```

Getting user input

templates/index.html

```
<form action="/details" method="GET">
 <div class="form-row">
   <span>kitabisa.com/</span> <input name="campaign" type="text"
   class="form-control" required/>
 </div>
 </form>
```

routes.py

```
@app.route('/details')
def details():
    # 1. Get the campaign link
    campaign = request.args.get('campaign', None)
```

Data Collection

Data from the campaign page

Web scraping using Beautiful Soup 4

Engagement data from Facebook

Facebook Graph API

routes.py

```
@app.route('/details')
def details():
    # 1. Get the campaign link
    campaign = request.args.get('campaign', None)

    response = get_data(campaign)
    data = json.loads(response)
```

Tip #2: Keep in mind the type of your data

Prediction

Libraries

We are using scikit-learn but depending on your app, you can use plenty others (LibSVM, Gensim, Tensor Flow...)

Make your model persistent

Pickle, joblib*

routes.py

```
# 2. Collect the data based on the given campaign link
# Remove collected_amt from our data
# because we don't need it for the model
df_input = pd.DataFrame([data]).drop(['collected_amt'], axis=1)
prediction = get_prediction(df_input)
```

predict.py

```
def get_prediction(df_input):

    # Load our saved model
    df_input = df_input.astype(float)
    model_file_name = './app/resources/model.sav'
    loaded_model = joblib.load(model_file_name)
    result = loaded_model.predict(df_input)
    result = np.expm1(result[0])

    return result
```

*) use with caution

Data Viz

Libraries

Matplotlib, Seaborn

How do we generate a plot dynamically and display it in a page?

routes.py

```
# 4. Display the graph & the result
target = int(data['donation_target_amt'])
fb_shares = display_fb_shares(target)
```

graph.py

```
import StringIO
import base64

...

fig, ax = plt.subplots(figsize=(8,4))
ax = sns.barplot(x='collected_amt', y='fb_share_count', data=df_binned, palette=colors)
ax.set(xlabel='Average Number of Facebook Shares', ylabel='Collected Amount')
ax.set_xticklabels(columns,rotation=30)

plt.tight_layout() # Give room for the x-axis labels

plt.savefig(img, format='png')
img.seek(0)

plot_url = base64.b64encode(img.getvalue())

return {'target_bin': target_bin, 'target_bin_avg': target_bin_avg, 'plot_url': plot_url}
```

Recap - what do we have so far?

from get_data()

```
data =
{
  "title_wc": title_wc,
  "short_wc": short_wc,
  "story_wc": story_wc,
  ...
  "prediction": prediction,
  ...
  "fb_reaction_count": fb_reaction_count,
  "fb_comment_count": fb_comment_count,
  "fb_share_count": fb_share_count,
  "collected_amt": collected_amt
}
```

Recap - what do we have so far?

from display_fb_shares()

```
fb_shares =
{
  "plot_url": plot_url,
  "target_bin": target_bin,
  "target_bin_avg": target_bin_avg
}
```

Display it to the user

routes.py

```
return render_template('details.html',
                       campaign=campaign,
                       data=data,
                       fb_shares=fb_shares
                       )
```

Display it to the user

Most of the time, everything is pretty straightforward

templates/details.html

```
<div class="card">
  <div class="inner-card">
    <h3>Your campaign statistics</h3>
    <br />
    <ul class="stats-list">
      <li>
        <div class="stats-name">Title word count</div>
        <div class="stats-value">{{ data["title_wc"] }}</div>
      </li>
      <li>
        <div class="stats-name">Short description word count</div>
        <div class="stats-value">{{ data["short_wc"] }}</div>
      </li>
      <li>
        <div class="stats-name">Story word count</div>
        <div class="stats-value">{{ data["story_wc"] }}</div>
      </li>

      ...
    </ul>
  </div>
</div>
```

Display it to the user

Displaying graph

RFC2397

templates/details.html

```
<h3>How can you improve your chances to reach your donation target?</h3>

<h4>Share your campaign link more often in Facebook</h4>

Campaigns that have successfully collected <strong>{{ fb_shares['target_bin'] }}</strong>

<div class="text-center">
  <img src=" {{ fb_shares['plot_url'] }}" />
</div>
```

Display it to the user

Conditional

templates/details.html

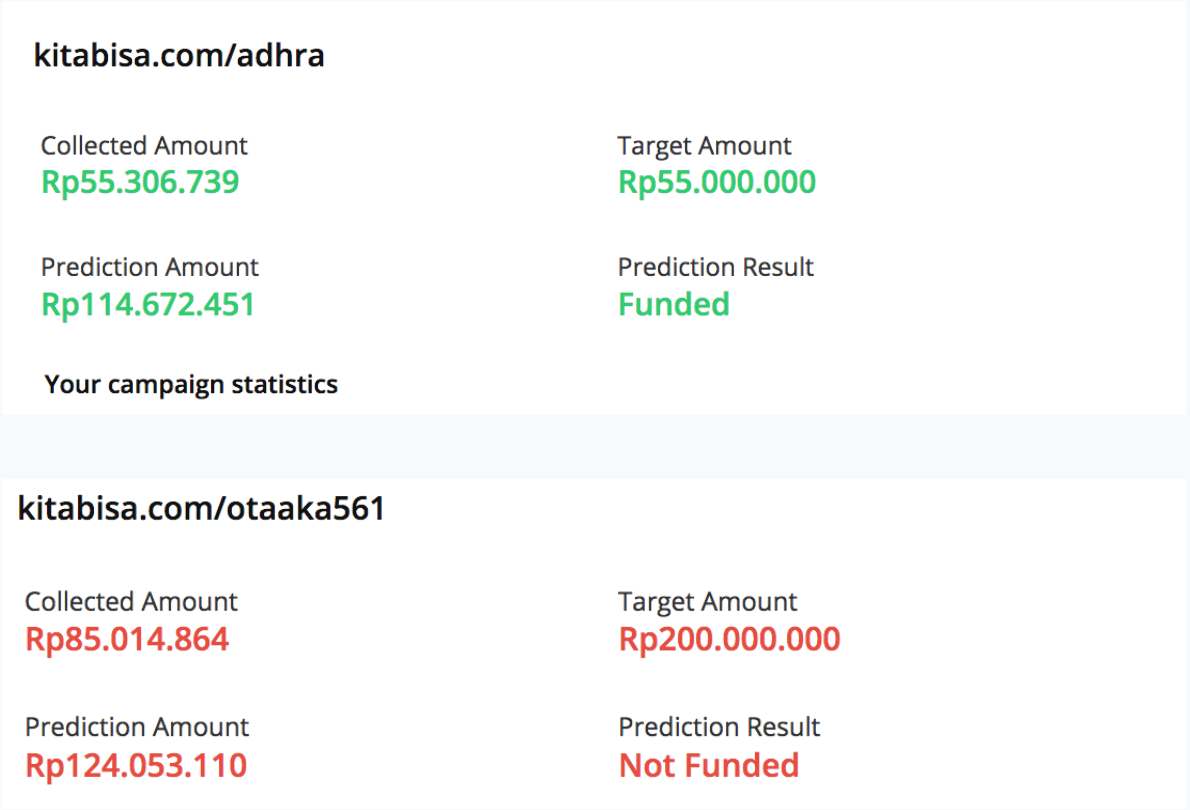
```
<h4>Collected Amount</h4>
</div>
<div class="stats-value {{'funded' if data['is_funded'] else 'not-funded'}}">
  {{ data["collected_amt"]|format_currency}}
</div>
```

static/style.css

```
.main-stats {
  padding-top: 24px;
  padding-bottom: 24px;
  background: #FFF; }
  .main-stats .stats-value {
    font-size: 30px;
    font-weight: 600;
    color: #3498db; }
    .main-stats .stats-value.not-funded {
      color: #e74c3c; }
    .main-stats .stats-value.funded {
      color: #2ecc71; }
```

Display it to the user

Conditional



Display it to the user

Custom filters

util/filters.py

```python
@app.template_filter('format_currency')
def format_currency(value):
    value = int(value)
    return "Rp{:,}".format(value).replace(",", ".")
```

templates/details.html

```
<h4>Collected Amount</h4>
</div>
<div class="stats-value {{'funded' if data['is_funded'] else 'not-funded'}}">
  {{ data["collected_amt"]|format_currency}}
</div>
```

Flask is capable of much more

Flask Admin

Flask Login

Flask Mail

... and so on!

That's it. 🎉 Thanks!