Machine Learning Model Deployment

Deploying Flask Apps

Using Heroku

Steps:

- Create and Pickle a Model Using Titanic Data
- Create Flask App
- Test Flask App in Production (optional)
- Create GitHub Repository (optional)
- Deploy to Heroku
- Test Working App

Creating and Training a Model

Titanic Data set

#serializing our model to a file called model.pkl

```
import pickle
pickle.dump(dt_clf_gini, open(".../Logistic_Regression_model.pkl","wb"))
```

The pickle module implements a fundamental, but powerful algorithm for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” 1 or “flattening”, however, to avoid confusion, the terms used here are “pickling” and “unpickling”.

As a data scientist, you will use sets of data in the form of dictionaries, DataFrames, or any other data type. When working with those, you might want to save them to a file, so you can use them later on or send them to someone else. This is what Python's pickle module is for: it serializes objects so they can be saved to a file, and loaded in a program again later on.

Creating a Simple Web Application using Flask

- Flask is a python based microframework used for developing small scale websites.
- It can create a REST API that allows you to send data, and receive a prediction as a response.
- Similar to what Django REST framework .

Creating a Simple HTML Form for WebApp

Index.html

Titanic Prediction Form

Age

Sex Male

Fare

Pclass 1

Submit

Installing Flask

!pip install flask

Create a project directory i.e flask-app

Put Logistic_Regression_model.pkl in project directory **flask-app**

mkdir **templates** under directory flask-app

Put index.html in templates directory

Create Logistic_Regression_model.py file and put under Titanic_Flask_Prediction directory

```
import os
import numpy as np
import flask
import pickle
from flask import Flask, render_template, request

#creating instance of the class
app=Flask(__name__)

#to tell flask what url should trigger the function index()
@app.route('/')
@app.route('/index')
def index():
    return flask.render_template('index.html')

if __name__ == '__main__':
    app.run(port = 5000, debug=True)
```

- app=Flask(__name__) create an instance of flask.
- @app.route('/') is used to tell flask what url should trigger the function index()
- In the function index we use render_template('index.html') to display the script *index.html* in the browser.

Now run the application from jupyter terminal

python Logistic_Regression_model.py

This should run the application and launch a simple server. Open http://127.0.0.1:5000/ to see the html form.

```
PS C:\Users\DELL> python Logistic_Regression_model.py
 * Serving Flask app "Logistic_Regression_model" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 229-278-700
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Putting the Prediction part in file

Logistic_Regression_model.py

```
#prediction function
def ValuePredictor(to_predict_list):
    to_predict = np.array(to_predict_list).reshape(1,4)
    loaded_model =
pickle.load(open("Logistic_Regression_model.pkl","rb"))
    result = loaded_model.predict(to_predict)
    return result[0]


@app.route('/result',methods = ['POST'])
def result():
    if request.method == 'POST':
        to_predict_list = request.form.to_dict()
        to_predict_list=list(to_predict_list.values())
        to_predict_list = list(map(int, to_predict_list))
        result = ValuePredictor(to_predict_list)
        if int(result)==1:
            prediction='Pessenger survived'
        else:
            prediction='Pessenger NOT survived'
        return

    render_template("result.html",prediction=prediction)
```

- Here after the form is submitted, the form values are stored in variable *to_predict_list* in the form of dictionary.
- We convert it into a list of the dictionary's values and pass it as an argument to *ValuePredictor()* function.
- In this function, we load the Logistic_Regression_model.pkl file and predict the new values and return the result.
- This result/prediction is then passed as an argument to the template engine with the html page to be displayed.

Create the following *result.html* file and add it to templates directory.

```
<!doctype html>
<html>
  <body>
    <h1> {{ prediction }}</h1>
  </body>
</html>
```

- Run the application again
- python Logistic_Regression_model.py
- It should predict the income after submitting the form.

This is how our project layout looks like

```
/flask-app
├── templates
│   └── index.html
│   └── result.html
├── Logistic_Regression_model.py
├── Logistic_Regression_model.pkl
```

Successfully created the Webapp
Now it’s time to use heroku to deploy it

HEROKU

Heroku is a platform as a service (PaaS) that enables developers to build, run, and operate applications entirely in the cloud.

Install gunicorn

```
!pip install gunicorn
```

Gunicorn handles requests and takes care of complicated things like threading very easily in real production.

Three common building blocks when deploying a Python web application to production are:

- A web server (like nginx)
- A WSGI (Web Server Gateway Interface) application server (like Gunicorn)
- Your actual application (written using a developer-friendly framework like Django)

- The web server accepts requests, takes care of general domain logic and takes care of handling https connections. Only requests which are meant to arrive at the application are passed on toward the application server and the application itself. The application code does not care about anything except being able to process single requests.

- Gunicorn takes care of everything which happens in-between the web server and your web application. This way, when coding up your a Django application you don’t need to find your own solutions for:
    - communicating with multiple web servers
    - reacting to lots of web requests at once and distributing the load
    - keepiung multiple processes of the web application running

Creating a **requirements.txt** file and put under project directory

```
pip freeze > requirements.txt
```

- The requirements.txt file will contain all of the dependencies for the flask app.
- In our local machine, we have installed a lot of libraries and other important files like flask, gunicorn, sklearn etc. We need to tell heroku that our project requires all these libraries to successfully run the application. This is done by creating a **requirements.txt** file.

Creating **Procfile** and put under project directory

- A Procfile specifies the commands that are executed by a Heroku app on startup.
- To create one, open up a new file named Procfile (no extension) in the working directory and paste the following.

```
web: gunicorn app:app
```

This is how our project layout looks like now

```
/flask-app
├── templates
│   ├── index.html
│   └── result.html
├── Logistic_Regression_model.py
├── Logistic_Regression_model.pkl
├── Procfile
└── requirements.txt
```

Create New GitHub Repository

Owner **Repository name** *

arachauhan / flask-app

Great repository names are short and memorable. Need inspiration? How about **miniature-garbanzo**?

Description (optional)

Public
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer.

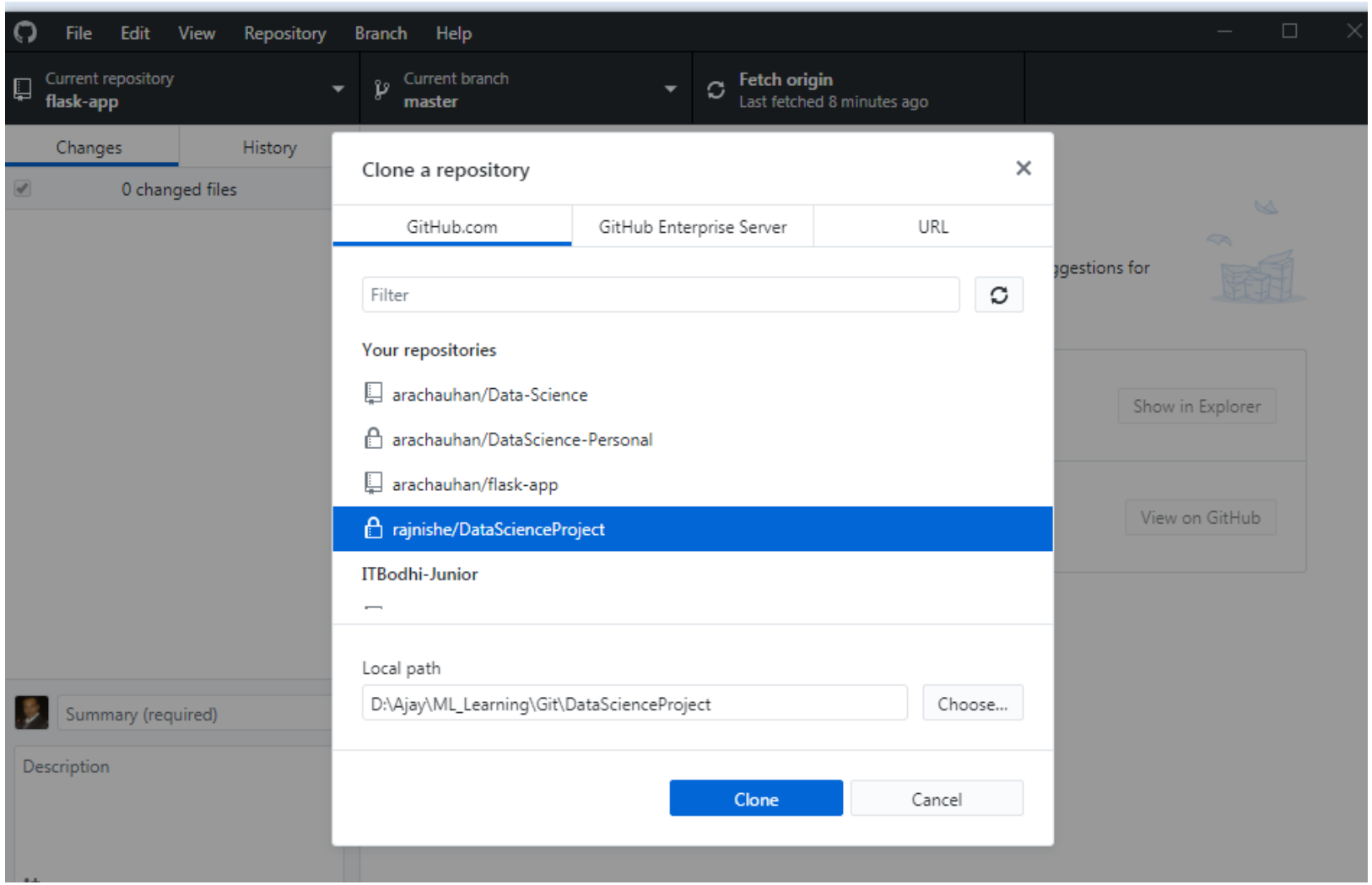
Add .gitignore: **None** | Add a license: **None**

Create repository

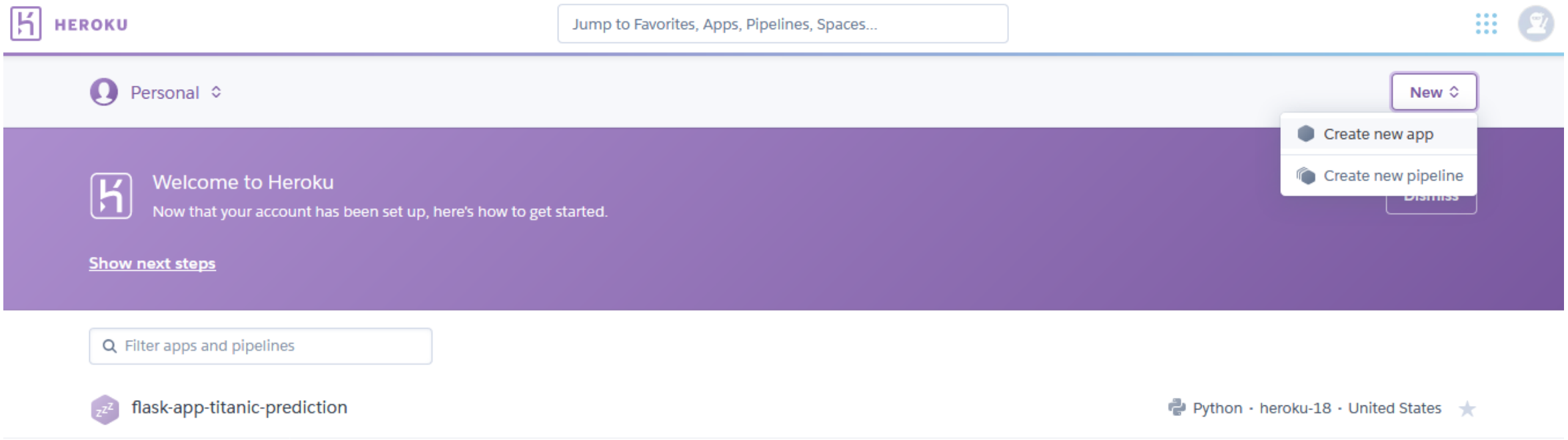
Using GitHUb Desktop for Rich Experience

- Download and install it
- We will be COMMIT complete project directory to GitHub and Connect Heroku to GitHub for WebApp deployment.
- You can also use heroku git that uses command line to deploy models.

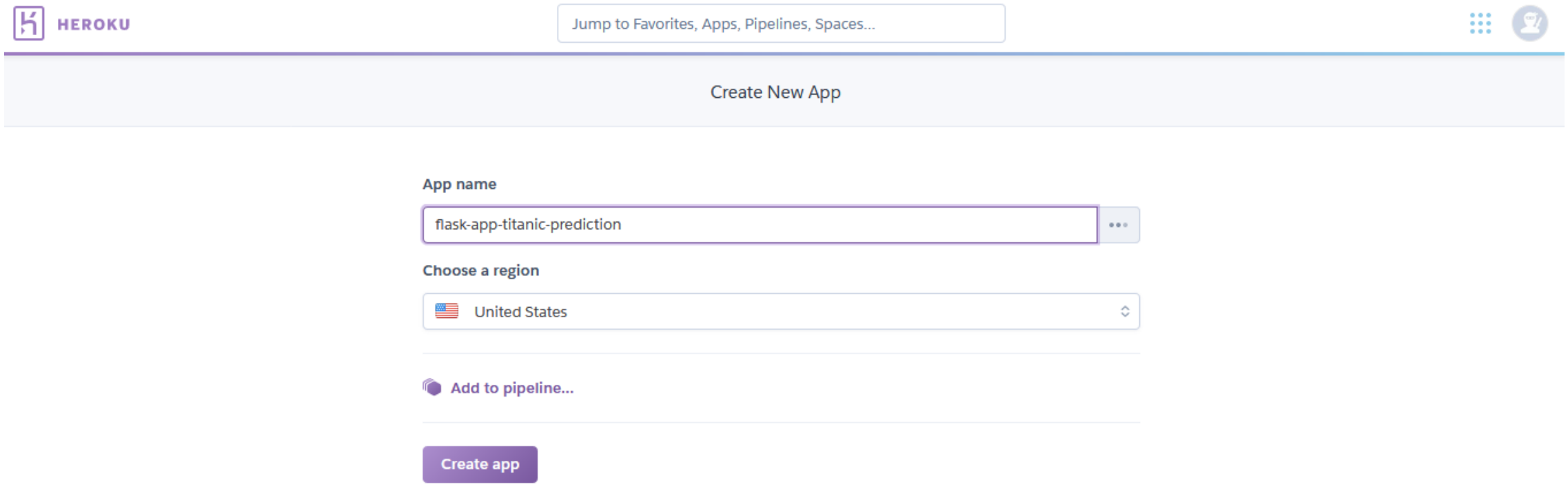
Clone GitHub repository to local project directory



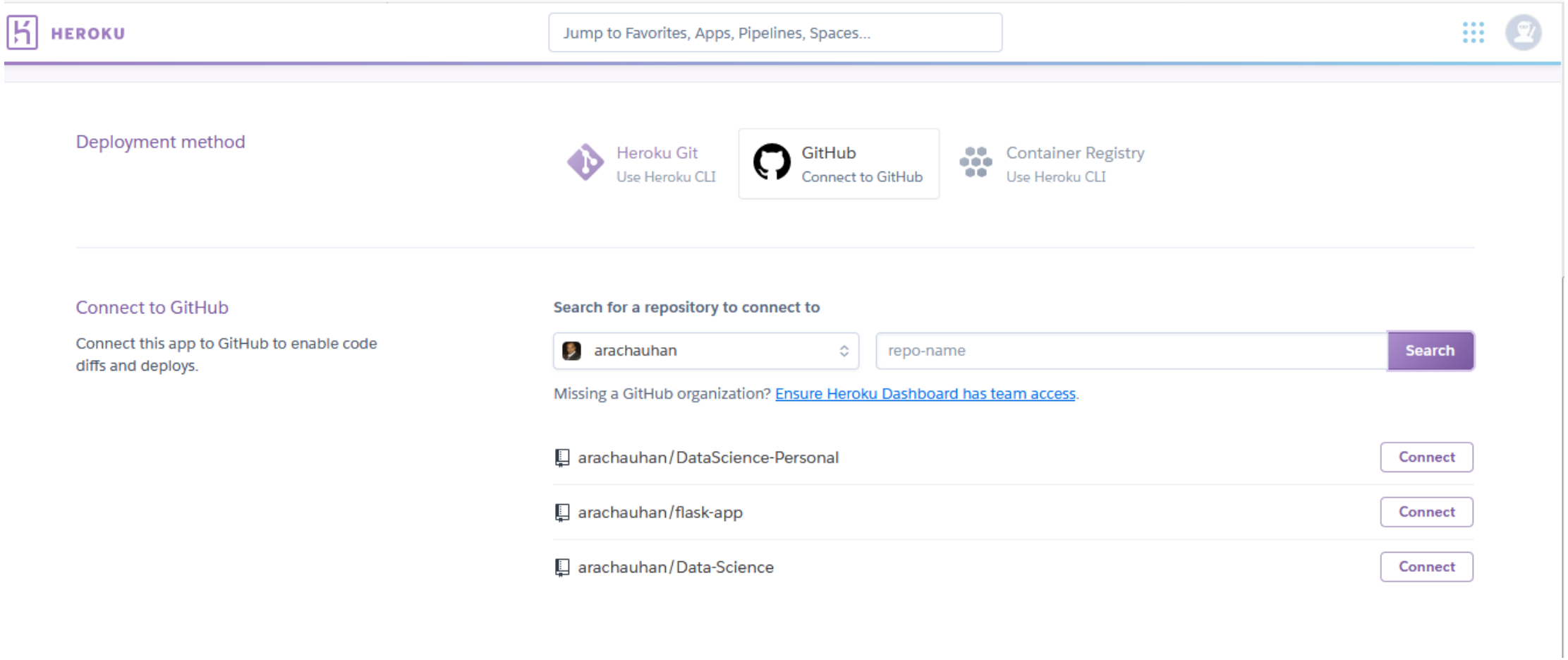
- **Create Fee account at www.heroku.com**
- Create a new app simply by choosing a name and clicking “create app”. This name doesn’t matter but it does have to be unique.



- **Create Fee account at** www.heroku.com
- Create a new app simply by choosing a name and clicking “create app”. This name doesn’t matter but it does have to be unique.



- You have a few options for the way you can deploy the app.
- Heroku CLI and GitHub
- Select Github
- Search repo-name as flask-app and connect



Just scroll to the bottom of the page and click “Deploy Branch”

HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch is always in a deployable state and any tests have passed before you push. Learn more.

Choose a branch to deploy

master

Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Enable Automatic Deploys

Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. Learn more.

Choose a branch to deploy

master

Deploy Branch

If everything Goes well, You should see below message

Your app was successfully deployed.

View

If something went wrong, check your requirements.txt, delete and dependencies that are giving you problems, and try again

Deploying with Heroku CLI

- In the Heroku CLI section, you will see these instructions to follow for deployment.
- Paste each command into your terminal and follow any prompts like logging in.
- Pay attention to any commands you will need to modify, such as cd my-project/ — where my-project/ should actually be your project directory.
- The git remote should be set to the app name from Heroku EXACTLY.

Install the Heroku CLI

Download and install the Heroku CLI.

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

```
$ heroku login
```

Create a new Git repository

Initialize a git repository in a new or existing directory

```
$ cd my-project/
$ git init
$ heroku git:remote -a  flak-app
```

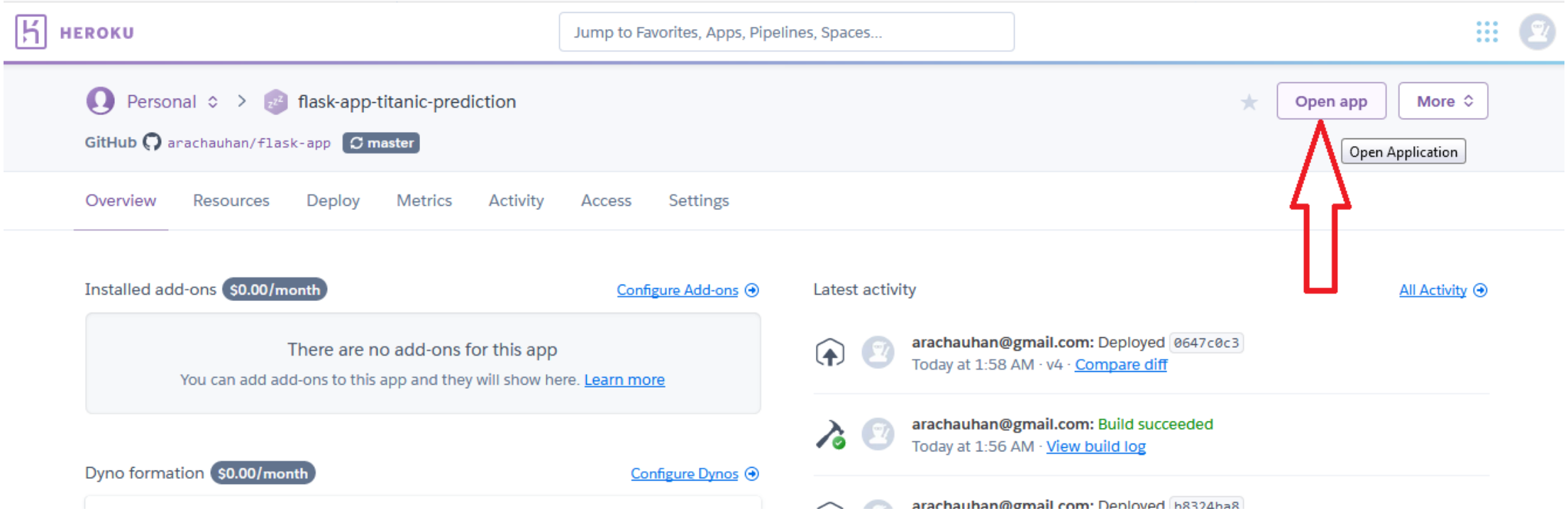
Deploy your application

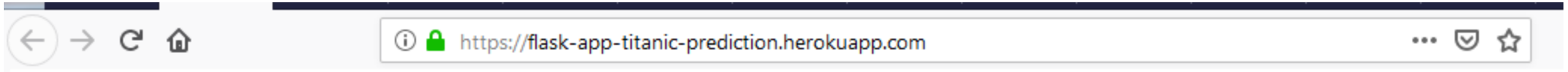
Commit your code to the repository and deploy it to Heroku using Git.

```
$ git add .
$ git commit -am "make it better"
$ git push heroku master
```

Test the Deployed Model & Generate Prediction

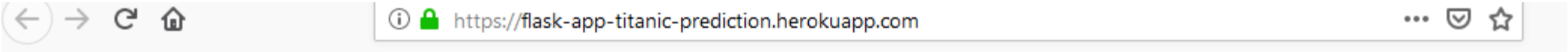
Click on Open App





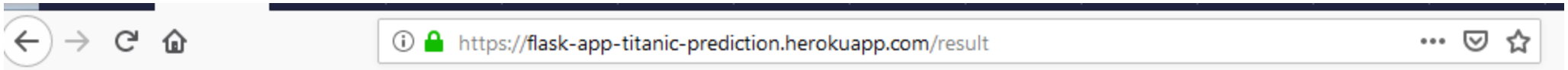
Titanic Prediction Form

Age
Sex Male
Fare
Pclass 1
Submit



Titanic Prediction Form

Age 34
Sex Male
Fare 56
Pclass 2
Submit



Pessenger NOT survived