

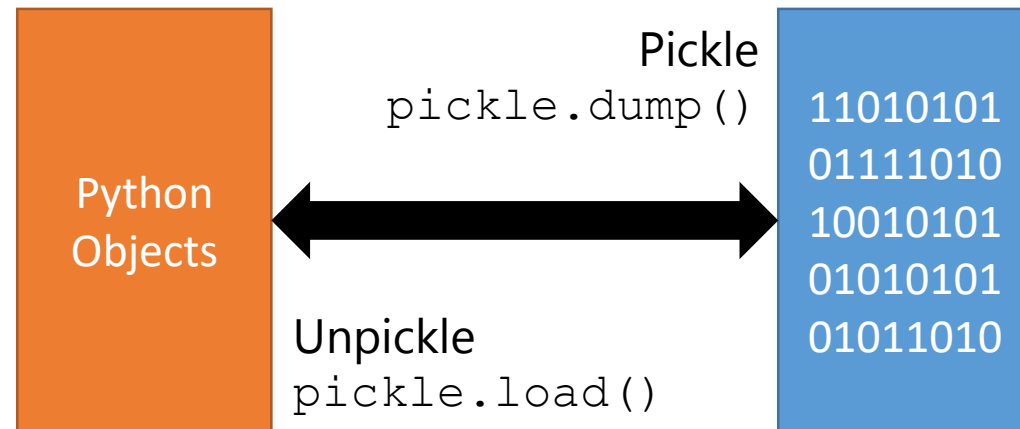
Model Deployment

Ratchainant Thammasudjartit, Ph.D.

-
- Pickle
 - API
 - Flask
 - Horeku

Python Pickle

- A module for (de)serialization
 - Storing a Python object into a file and later loading it back in another python script



Example

```
# Python3 program to illustrate store
# efficiently using pickle module
# Module translates an in-memory Python object
# into a serialized byte stream—a string of
# bytes that can be written to any file-like object.
```

```
import pickle
```

```
def storeData():
    # initializing data to be stored in db
    Omkar = {'key' : 'Omkar', 'name' : 'Omkar Pathak',
            'age' : 21, 'pay' : 40000}
    Jagdish = {'key' : 'Jagdish', 'name' : 'Jagdish Pathak',
            'age' : 50, 'pay' : 50000}

    # database
    db = {}
    db['Omkar'] = Omkar
    db['Jagdish'] = Jagdish

    # Its important to use binary mode
    dbfile = open('examplePickle', 'ab')

    # source, destination
    pickle.dump(db, dbfile)
    dbfile.close()
```

```
def loadData():
    # for reading also binary mode is important
    dbfile = open('examplePickle', 'rb')
    db = pickle.load(dbfile)
    for keys in db:
        print(keys, '=>', db[keys])
    dbfile.close()
```

```
if __name__ == '__main__':
    storeData()
    loadData()
```

• Output

```
➤ Omkar => {'key': 'Omkar', 'name': 'Omkar Pathak', 'age': 21, 'pay': 40000}
Jagdish => {'key': 'Jagdish', 'name': 'Jagdish Pathak', 'age': 50, 'pay': 50000}
```

Python Pickle

- Advantages
 - A recursive object because it contains reference to itself
 - Pickle tracks the serialized object by its reference
 - The same object won't be serialized again
 - Object sharing by referencing to the same object from different places
 - Similar to self-referencing objects
 - Pickle ensures that all other references point to the master copy
 - Shared objects remain shared, which can be very important for mutable objects
 - User-defined classes and their instances
 - Pickle can save and restore class instances transparently
 - The class definition must be importable and live in the same module as when the object was stored

Further reading: <https://docs.python.org/3/library/pickle.html>

Serialize your model

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

class ai:

    def __init__(self, data, features, target, test_size):
        self.X = data.loc[:, features].values
        self.y = data[target].values.ravel()
        self.model = self.learn(self.X, self.y, test_size)

    def learn(self, X, y, test_size):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, stratify=y)
        clf = LogisticRegression(max_iter=10000, penalty="L2")
        clf.fit(X_train, y_train)
        return clf
```

- Let's train a logistic regression model from heart disease data
- Write source code in the OOP style
 - medml.py represents the blueprint of your experiment
 - mymodel.py represents your data preparation and modeling

Serialize your model

```
import pandas as pd
import medml as ml
column_names = ["age", "sex", "cp", "trestbps", "chol", "fbs",
                "restecg", "thalach", "exang", "oldpeak", "slope",
                "ca", "thal", "num"]
df = pd.read_csv("../data/processed.cleveland.data", header=None, names=column_names)
df = df[df["thal"] != "?"].reset_index(drop=True)
df = df[df["ca"] != "?"].reset_index(drop=True)

df["Labels"] = df["num"].apply(lambda x: 1 if x > 0 else 0)

features = ["thal", "exang", "cp", "ca", "slope"]

# thal
thal = pd.get_dummies(df["thal"])
thal.columns = ["normal", "fixed defect", "reversable defect"]
df = pd.concat([df, thal], axis=1)

# cp
df["cp"] = df["cp"].map({1: "typical angina",
                        2: "atypical angina",
                        3: "non-anginal pain",
                        4: "asymptomatic"})
cp = pd.get_dummies(df["cp"])
df = pd.concat([df, cp], axis=1)

# slope
df["slope"] = df["slope"].map({1: "upsloping",
                              2: "flat",
                              3: "downsloping"})
slope = pd.get_dummies(df["slope"])
df = pd.concat([df, slope], axis=1)

data = df.loc[:, ["normal", "fixed defect", "reversable defect", "typical angina",
                  "atypical angina", "non-anginal pain", "asymptomatic",
                  "upsloping", "flat", "downsloping", "exang", "ca", "Labels"]]

features = data.columns.tolist()
features.remove("Labels")
heart = ml.ai(data=df, features=features, target="Labels", test_size=0.2)
```

- Let's train a logistic regression model from heart disease data
- Write source code in the OOP style
 - medml.py represents the blueprint of your experiment
 - mymodel.py represents your data preparation and modeling

Serialize your model

```
In [47]: heart.model.coef_  
Out[47]:  
array([[ -0.90843831, -0.10403861,  1.0125147 , -0.37344252, -0.04552741,  
        -0.56073491,  0.97974262, -0.87401177,  0.60989188,  0.26415767,  
         0.66072607,  1.40903355]])
```

- You have trained your model
- `.coef_` is the class attribute to store model parameters for logistic regression object

Serialize your model

```
import pickle  
pickle.dump(heart.model, open("Logregheart.pkl", "wb"))
```

- Dump you model

Serialize your model

```
In [79]: myheart = pickle.load(open("Logregheart.pkl", "rb"))  
  
In [80]: myheart.coef_  
Out[80]:  
array([[ -0.90843831, -0.10403861,  1.0125147 , -0.37344252, -0.04552741,  
        -0.56073491,  0.97974262, -0.87401177,  0.60989188,  0.26415767,  
         0.66072607,  1.40903355]])
```

- Yore model can be read back

Flask



- Flask is a fast, lightweight way to connect your Python scripts to a server
- Flask is a simple and robust framework to do
 - Small tasks (create a microblog, stand up a simple API) or
 - Complex tasks (Pinterest's API, create a twitter clone)

Hello World Example

```
import flask
app = flask.Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

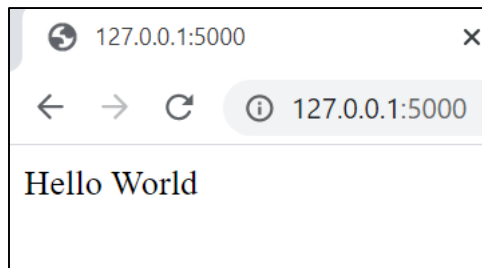
if __name__ == '__main__':
    app.run(debug=True)
```

- Create a new .py file, e.g. hello.py
- Flask constructor takes the name of current module (**__name__**) as argument
- The **route()** function of the Flask class is a decorator, which tells the application which URL should call the associated function
 - App routing is used to map the specific URL with the associated function
 - In this example, the URL ('/') is associated with the hello function that returns “Hello World!” displayed on the web page

ref: https://www.tutorialspoint.com/flask/flask_application.htm

Hello World Example

```
(cebrama) G:\My Drive\Classroom\RADS 602\Original Files\src>python hello.py
* Serving Flask app "hello" (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 windowsapi reloader
* Debugger is active!
* Debugger PIN: 193-478-798
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [26/Jan/2020 17:11:28] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [26/Jan/2020 17:11:28] "GET /favicon.ico HTTP/1.1" 404 -
```



- Run your hello.py
- Your app is running at default url:port at <http://127.0.0.1:5000>
- Access the above url via your web browser

Hello World Example

```
import flask

app = flask.Flask(__name__)

@app.route("/")
def hello_world():
    return "Hello World"

@app.route("/greet/<name>")
def greet(name):
    return "Hello, {}".format(name)

if __name__ == "__main__":
    app.run()
```

← → ↻ ⓘ localhost:5000/greet/Rucci

Hello, Rucci!

- Argument and Styling

- Add the new function under hello() function
- Go to url localhost:5000/greet/Rucci
- Data type can be specified <str:name>
 - String: Default
 - int: Use to convert string to integer
 - float: Use to convert string to float

- Note: the section

if __name__ == "__main__"

should be the last section of your code

Hello World Example

```
@app.route("/")
def hello():
    return '''
    <body>
    <h2> Hello World! <h2>
    </body>
    '''
```

- Decorator

- app.route(endpoint) tells Flask to listen to a URL, and what to do if requests are sent there
- Since the return statement sends text to an HTML page, you can style it with HTML tags
- The list of html tags can be found here

https://www.w3schools.com/tags/ref_byfunc.asp

Adding ML to Flask

```
import flask
app = flask.Flask(__name__)

#----- MODEL GOES HERE -----#

#----- ROUTES GO HERE -----#

if __name__ == '__main__':
    '''Connects to the server'''

    HOST = '127.0.0.1'
    PORT = '4000'
    app.run(HOST, PORT)
```

- Structure
- Create a new .py file inside your working directory named **app.py**

Adding ML to Flask

```
import numpy as np
import flask
import pickle

# app
app = flask.Flask(__name__)

# load model
heart = pickle.load(open("Logregheart.pkl", "rb"))

# routes
@app.route("/")
def home():
    return """
        <body>
        <h1>Heart Disease Prediction</h1>
        </body>"""
```

```
@app.route("/predict", methods=["GET"])
def predict():
    thal = flask.request.args["thal"]
    cp = flask.request.args["cp"]
    slope = flask.request.args["slope"]
    exang = flask.request.args["exang"]
    ca = flask.request.args["ca"]

    fmap = {"normal": [1, 0, 0],
            "fixed defect": [0, 1, 0],
            "reversible defect": [0, 0, 1],
            "typical angina": [1, 0, 0, 0],
            "atypical angina": [0, 1, 0, 0],
            "non anginal pain": [0, 0, 1, 0],
            "asymptomatic": [0, 0, 0, 1],
            "upsloping": [1, 0, 0],
            "flat": [0, 1, 0],
            "downsloping": [0, 0, 1]}

    # X_new = fmap[thal] + fmap[cp] + fmap[slope]

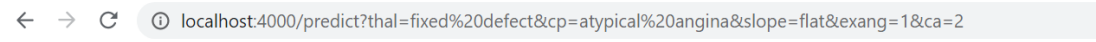
    X_new = np.array(fmap[thal] + fmap[cp] + fmap[slope] + [int(exang)] + [int(ca)]).reshape(1, -1)
    yhat = heart.predict(X_new)
    if yhat[0] == 1:
        outcome = "heart disease"
    else:
        outcome = "normal"
    prob = heart.predict_proba(X_new)

    return "This patient is diagnosed as " + outcome + " with probability " + str(round(prob[0][1], 2))

if __name__ == '__main__':
    """Connect to Server"""
    HOST = "127.0.0.1"
    PORT = "4000"
    app.run(HOST, PORT)
```

Adding ML to Flask

- Parsing your input variables
 - Example:
 - thal: fixed defect
 - cp: atypical angina
 - slope: flat
 - exang: 1
 - ca: 2



← → ↻ localhost:4000/predict?thal=fixed%20defect&cp=atypical%20angina&slope=flat&exang=1&ca=2

This patient is diagnosed as heart disease with probability 0.93

Applying ML to Flask

```
<html>
<head>
<<title> Heart Disease Prediction </title>
</head>
<body>
<h3> Patient Profile <h3>
<h5>
<form action="http://localhost:4000/result" method="POST">
<p>Thalassemia</p>
<input type="radio" name="thal" value="normal"> Normal
<input type="radio" name="thal" value="fixed defect"> Fixed defect
<input type="radio" name="thal" value="reversable defect"> Reversable Defect
<p>Chest Pain</p>
<input type="radio" name="cp" value="typical angina"> Typical Angina
<input type="radio" name="cp" value="atypical angina"> Atypical Angina
<input type="radio" name="cp" value="non anginal pain"> Non-Anginal Pain
<input type="radio" name="cp" value="asymptomatic"> Asymptomatic
<p>Slope</p>
<input type="radio" name="slope" value="upsloping"> Upsloping
<input type="radio" name="slope" value="flat"> Flat
<input type="radio" name="slope" value="downsloping"> Downsloping
<p>Exercise induced angina (exang)</p>
<input type="radio" name="exang" value=1> Yes
<input type="radio" name="exang" value=0> No
<p>Number of major vessels (0-3) colored by flourosopy</p>
<input type="radio" name="ca" value=0> 0
<input type="radio" name="ca" value=1> 1
<input type="radio" name="ca" value=2> 2
<input type="radio" name="ca" value=3> 3
<p><input type="submit" value="submit" /></p>
</form>
<h5>
</body>
</html>
```

```
@app.route("/page")
def page():
    with open("page.html", 'r') as viz_file:
        return viz_file.read()
```

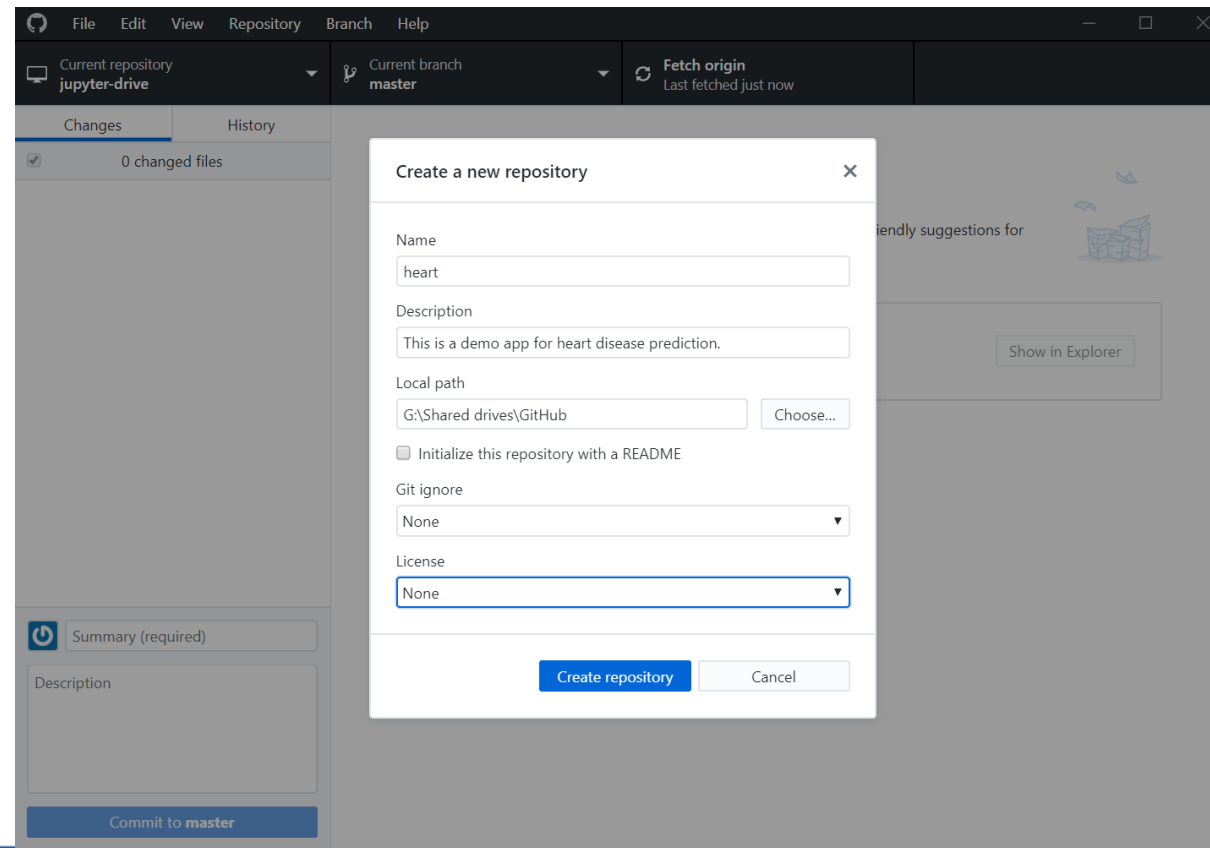
- Making the better one with web UI
 - Create a html file page.html
 - Add app.route("/page") in your app.py

Exercise

- Add a new method to print out the logit model

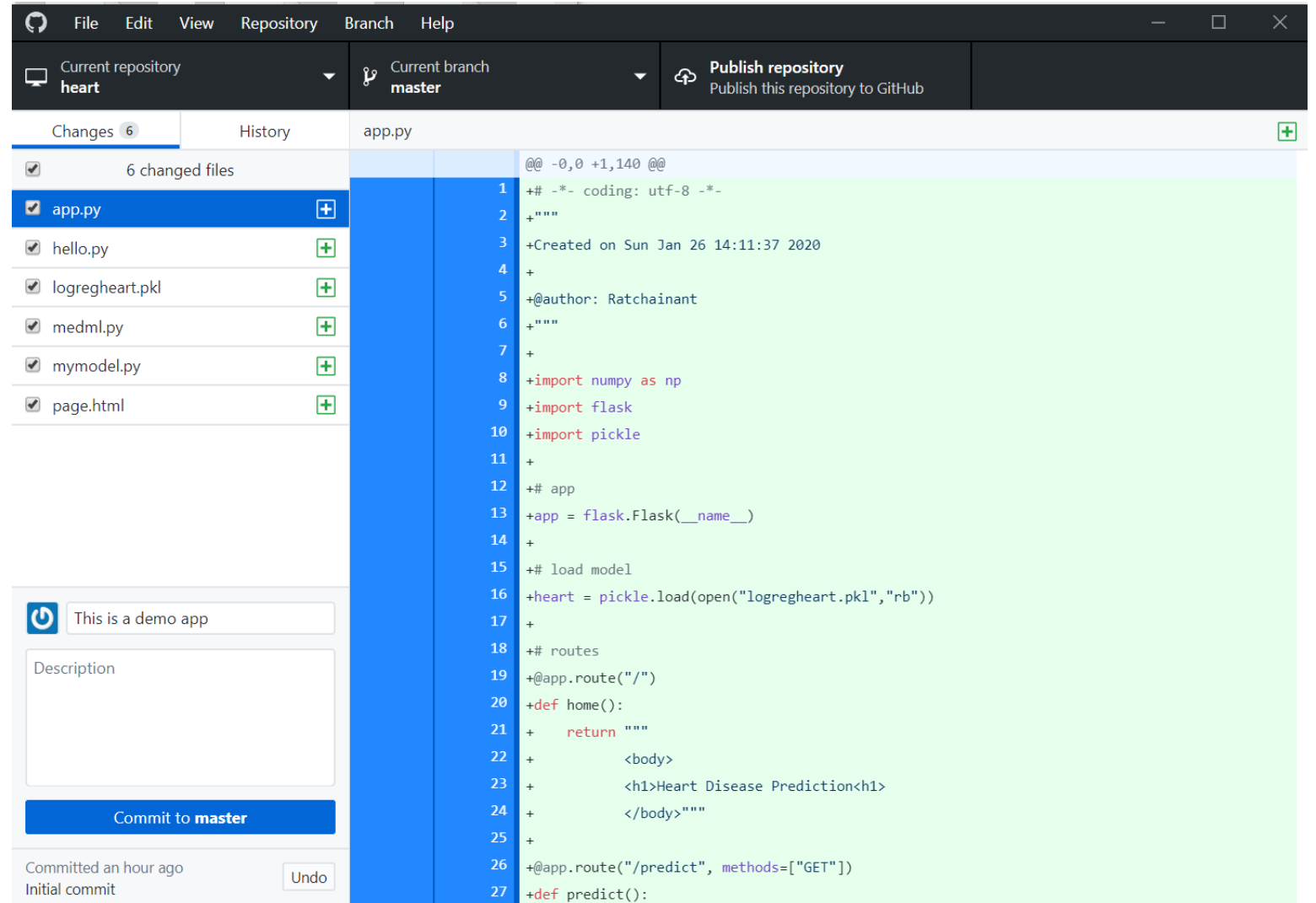
GitHub

- Create the new repository



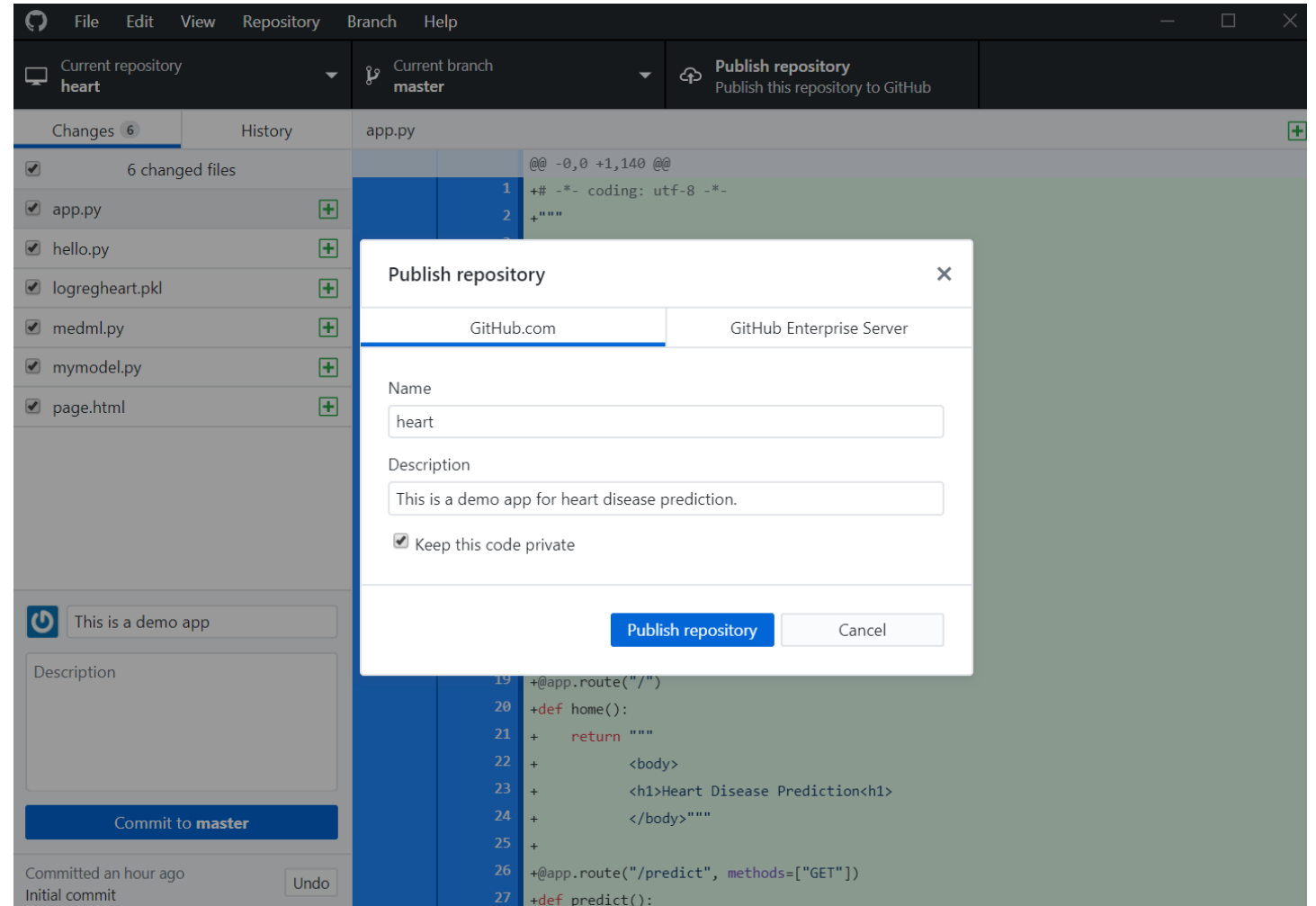
GitHub

- Add your files



GitHub

- Publish repository




GitHub









- Upload your files

Branch: master ▼

New pull request

 Ratchainant

Add files via upload

 .gitattributes	Initial commit
 README.md	Initial commit
 app.py	Add files via upload
 hello.py	Add files via upload
 logregheart.pkl	Add files via upload
 medml.py	Add files via upload
 mymodel.py	Add files via upload
 page.html	Add files via upload

Heroku

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

- Preparation
 - 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

Heroku

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

- Create a requirement.txt
- You may need to run this command at your working directory

```
$ pip install -r requirements.txt
```

- If you get error “access denied”, add --user after the above command

Heroku



Create a new app

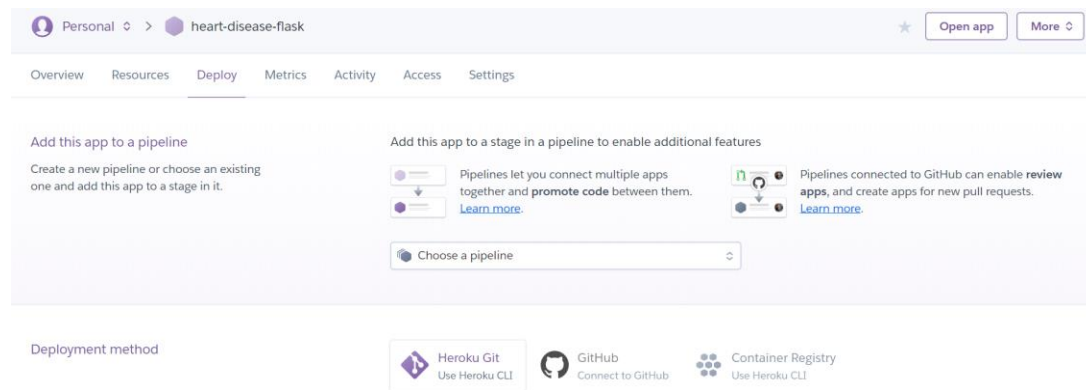
Create your first app and deploy
your code to a running dyno.

Create new app

- Create the new app

Heroku

- Deploy from your GitHub




Heroku

- Search for the correct repository and click connect

Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

Search for a repository to connect to

 Ratchainant

heart-disease

Search

Missing a GitHub organization? [Ensure Heroku Dashboard has team access.](#)

 Ratchainant/heart-disease

Connect

Heroku

- And then just scroll to the bottom of the page and click **Deploy Branch**

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