

Exercise 8_1: Deploying ML Models to Production

In Module 6, we have learned how to train different ML models. In this Module, we will learn how to use these models in our applications. As discussed in the reader, there are different platforms that can be used for ML model deployment and different ways of deploying your ML models. In this exercise, we will see how to deploy models you trained in Module 6 on the cloud platform Heroku. We will use the Python library **flask** to deploy our applications.

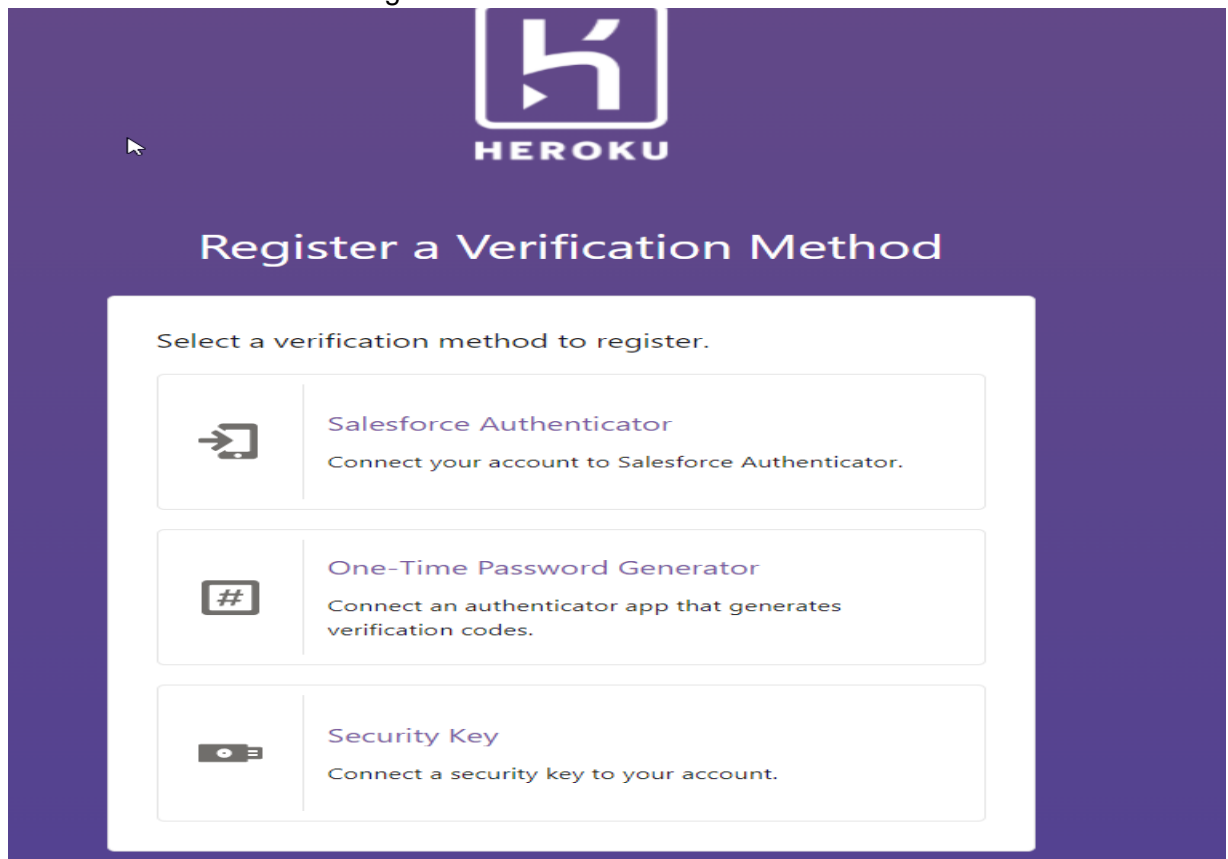
First, let's see how to set up your account on Heroku if you don't already have one.

What is Heroku?

Heroku is a PaaS (Platform-as-a-service) owned by Salesforce. Heroku is backed by AWS and all Heroku applications/services are hosted on AWS.

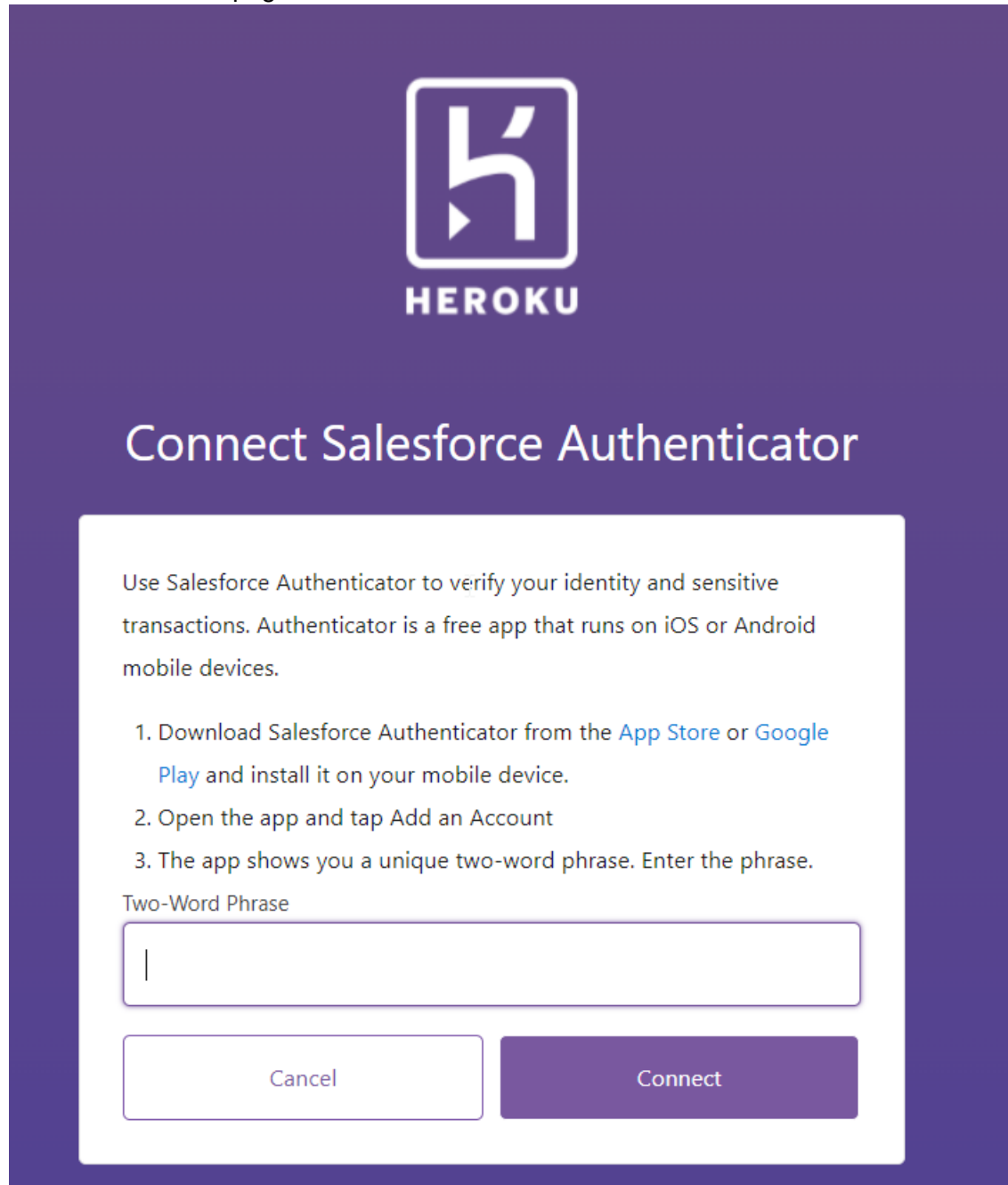
Creating a Heroku Account


1. Go to <https://www.heroku.com/> and click on Sign up.
2. You will be prompted to fill in different details. Complete the form and click "Create an Account".
3. You will be asked to confirm your email. Please do so by following the link Heroku sends to your email.
4. You will then see the following screen:



First, download the “Salesforce Authenticator” app on your mobile (either from App Store or Play Store). Then click on the Salesforce Authenticator option you see on the webpage.

5. You will now see a page that looks like below.




HEROKU

Connect Salesforce Authenticator

Use Salesforce Authenticator to verify your identity and sensitive transactions. Authenticator is a free app that runs on iOS or Android mobile devices.

1. Download Salesforce Authenticator from the [App Store](#) or [Google Play](#) and install it on your mobile device.
2. Open the app and tap Add an Account
3. The app shows you a unique two-word phrase. Enter the phrase.

Two-Word Phrase

Cancel

Connect

As you can see, you are prompted to enter a two-word phrase. Find the phrase from the “Salesforce Authenticator” application you downloaded, and write it on the webpage’s prompt. Then, click connect.

6. Now go to the “Salesforce Authenticator” app on your phone and verify the connection. You are now ready to use heroku.

Model Persistence

Model persistence is the ability to save and load the machine learning model. It is desirable to have a way to persist the model for future use without having to retrain. In many instances, model persistence is as simple as saving a copy of the model to a hard disk. Since we have been using Python in this training course we will explore and use Python tools for model persistence in this module.

To make the instructions simpler, we have given you a whittled-down version of Exercise 6_1 as Exercise 8_1. There is not much to do in the notebook except to run the cells and download the serialized model after running all cells.

Deploying Models on Heroku Using Flask

Deploying Models to Heroku can be completed in four easy steps.

1. Save (persist) the trained model
2. Create a Flask application that uses the trained model.
3. Create a Heroku App, and connect it to a GitHub repo of your application.
4. Configure the deployment environment and server on Heroku

Let's proceed with these steps using the instructions below for guidance:

1. Save (persist) the trained model

In this example, we will save the model we trained in Module 6_1.

- Go to:
<https://colab.research.google.com/drive/13D0dcl5DqSP4KlhBBh3kG2vkm3E2MIAL?usp=sharing> and download the notebook, then upload it to your drive.
- Once uploaded, run all the cells in the notebook. Notice the last cell of the notebook handles saving the model.
- After running the final cell, download the persisted model by going Files tab on the left pane and downloading the file `model.pkl`.

2. Create a Flask application that uses the trained model.

Here is what we will use Flask for.

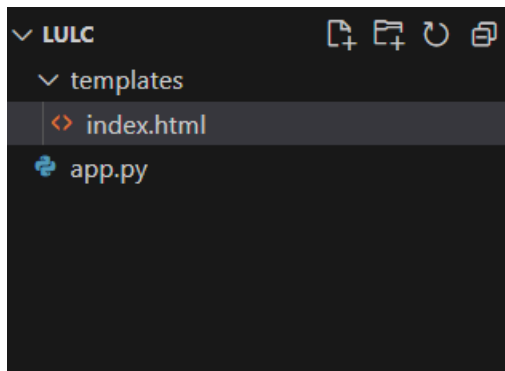
1. Display an HTML form with a form for the user to provide input.
2. Detect button-click by the user and send the entered data for processing by the backend.
3. Compute model input features from the user input.
4. Pass the features to the model and get the prediction.
5. Display the model outputs on the Webpage.

With Flask we can finish all the steps above by writing two files.

- First, create a folder named `lulc` on your computer.
- Then create files according to the following directory tree:

```
lulc
├── templates
│   └── index.html
├── app.py
└── model.pkl
```

The directory structure looks like this in VS Code:



- Open the folder with your favorite IDE or text editor.
- Copy the model you downloaded in Step 1 to the folder `lulc`.
- Copy the `.private-key.json` file you created previously (Module 5) into the folder `lulc`.
- Next, go to GitHub and create a repository named `lulc`.

If you are not sure how to create a **PRIVATE** repository on GitHub, please follow the instructions given at (<https://docs.github.com/en/github-ae@latest/get-started/quickstart/create-a-repo>).

If you are using a Windows machine, please also download Git Bash from <https://github.com/git-for-windows/git/releases/download/v2.40.1.windows.1/Git-2.40.1-64-bit.exe> and install it if you don't already have it.

- Open your terminal (or git bash if on Windows) and open the folder `lulc` using the following command:

```
cd path_to_lulc
```

Depending on your platform and where you placed `lulc` the path to the folder lulc can be different.

- Now go to https://github.com/<your_github_username>/lulc and make sure you replace <your_github_username> with your actual Github username.

From the page opened, you will find instructions that look like the ones below. (Notice the third command is modified).

- Copy and run the instructions below (on your terminal or git bash) one by one:

```
echo "# eodeploy" >> README.md
git init
git add .
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/<your_username>/lulc.git
git push -u origin main
```

Notice here that we are simply placing our `.private-key.json` on a GitHub repository. This is bad practice from a security standpoint. However, we tolerate this in this exercise for the purposes of simplification.

Let's now create a minimal Flask application by creating a web page that simply prints "Hello, Flask!"

- Copy the following content to the file `app.py`:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def home():
    return 'Hello, Flask!'
```

Let's discuss what each line does. The first line imports Flask. The second line creates an app, that hosts the application. The third line defines a route that calls a Python function. A route maps what you type in the browser (the URL) to a Python function. That is when a user types a certain URL on a website the request is handled by the function that the route is attached to. In the above example, the function ``home`` is called whenever there is a request from the homepage URL of the website. In the fourth line, the function ``home`` is defined. As you can see, the function returns text the browser can display.

3. Create a Heroku App, and connect it to a GitHub repo of your application.

We have already written the codes for our model deployment and pushed our code to the GitHub repo. Let's now connect Heroku to our GitHub repository.

- Login to Heroku at <https://heroku.com>
- Find Create App on the Heroku page. If it is your first time using Heroku you may be asked to enter a payment method. You should enter your credit card details here. More details on how to complete the payment will be provided on-site

The next page will prompt you to name your app. In the example below case, we have named the app `lulc`

App name

lulc

Choose a region

United States

Add to pipeline...

Create app Cancel

- Once you write the name click Create app.

Once the page opens you will get a chance to choose a deployment method as per the screenshot below:

Deployment method

Heroku Git Use Heroku CLI

GitHub Connect to GitHub

Container Registry Use Heroku CLI

- Click on GitHub.
- Then, click on “Connect to GitHub”. You will now be prompted to log in and approve the connection between Heroku and your repository.

After completing these steps, you will see dropdowns prompting you to choose which repository and branch to connect to as per the screenshot below.

- Carefully enter the repository and branch. The name of the repository should be lulc.
- Click search and you shall see the name of the GitHub repository of interest click on it. You have now connected GitHub and Heroku.

Deployment method

Heroku Git Use Heroku CLI

GitHub Connect to GitHub

Container Registry Use Heroku CLI

Connect to GitHub

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

Search for a repository to connect to

abrhamkg repo-name Search

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

4. Configure the deployment environment and server on Heroku

We just connected Heroku and the GitHub repository. Now, let's see how to configure the Heroku environment. First, we need to create a requirements.txt file that contains all the Python packages our application will use. In our case, the requirements file contents are shown below:

```
earthengine-api==0.1.354
flask==2.2.2
numpy==1.24.3
pandas==1.5.2
pip==23.0.1
scikit-learn==1.2.2
scipy==1.10.1
unicorn==19.9.0
itsdangerous==2.0
Jinja2==3.0
MarkupSafe==2.1.1
Werkzeug==2.2.2
```

Exercise 8.1:

Create a file named requirements.txt inside the `lulc` folder you created and copy the above contents into the file.

The second thing we need to do is create a Procfile. A Procfile is a file that specifies commands that are executed by Heroku on startup.

In our case, the content of our Procfile is just one line that starts a unicorn server (the server that runs our flask app). The command is given below:

```
web: unicorn app:app
```

Now that you have added all the necessary files to the repository let's deploy the application.

First, go to your terminal (or git bash) and open the folder `lulc` from the terminal (git bash). Then, run the following commands in order.

```
git add .
git commit -m "Ready for deployment"
git push origin main
```

In the above steps, we are merely updating our GitHub repository with the changes we made to our local files. Now everything is ready for application deployment.

For now, we will conduct a manual deployment. At the bottom of your page, you will notice a section called “Manual Deploy” as shown below.


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

 main

Deploy Branch

Click on “Deploy Branch”.

If this step completes successfully, you have now deployed your application successfully. You will see an output like shown below if this step completes successfully.

Receive code from GitHub

Build main 56ee65b4

Release phase

Deploy to Heroku

✓

✓

✓

✓

Your app was successfully deployed.

View

Click on View to test your deployed application. You should see something like the image below.

Hello, World!

This was a very simple example to show you how to deploy an application on Heroku. Let's now deploy the model as well and create an interface where a user can send requests to our model.

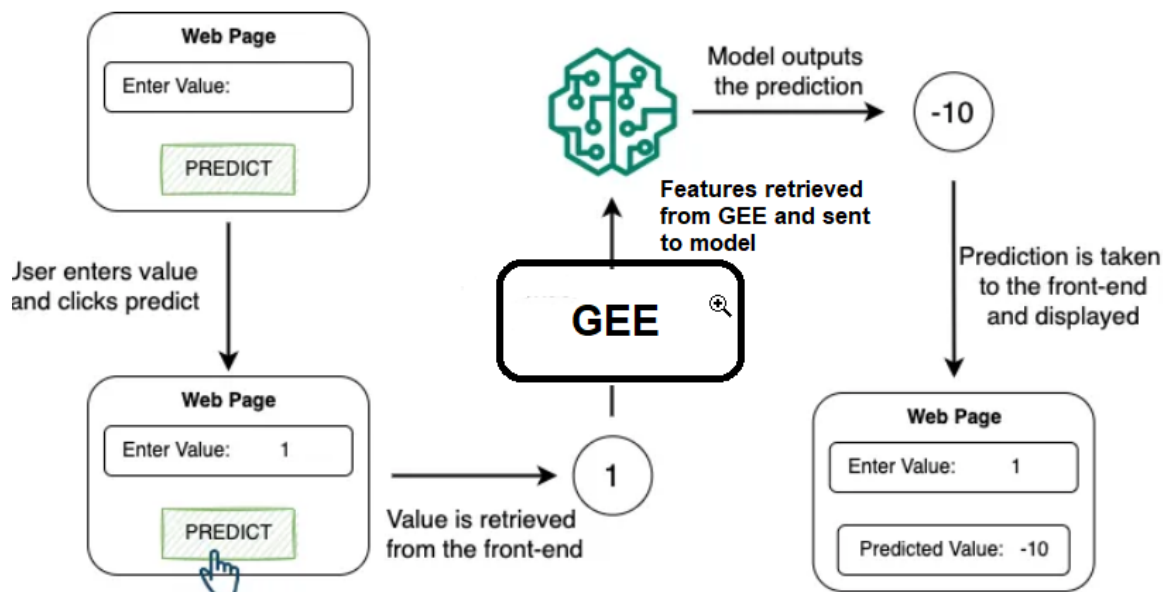
Deploying the Entire Application to Heroku

Let's first describe the application we are trying to build. We are interested in creating a web interface where a user can specify the coordinates of a location and send queries to the model. The desired interface is shown below.

Land Use Application

Longitude Latitude

Once a request is sent from this webpage our backend is expected to find the features that correspond to the location entered by a user, pre-process the features as necessary, pass the preprocessed features to the model, and return the output. A detailed workflow is shown below:



Let's now revisit our directory structure for the folder `lulc` we created.

```
lulc
├── templates
│   └── index.html
├── app.py
└── model.pkl
```

The first three steps in the workflow will be implemented in the `index.html` file. The remaining steps will be implemented inside the app.py file. For this exercise, all the content of the webpage will be provided.

Copy the following content and paste it inside your templates/index.html file:

```
<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">
  <title>ML Deployment using Heroku</title>
</head>

<body>
  <div class="prediction">
    <h1>Land Use Application</h1>
    <form action="{{ url_for('predict')}}" method="post">
      <label for="long">Longitude</label>
      <input type="text" id="longitude" name="longitude" required="required">
      <label for="lat">Latitude </label>
      <input type="text" id="latitude" name="latitude" required="required">
      <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
    </form>

    <br>
    <b>{{ prediction_text }}</b>
  </div>
</body>
</html>
```

The code above creates a web page that contains one form with two input fields and one submit button. The action attribute of the form must be set to the URL that the model listens at (we'll see how to set this up shortly).

Before we discuss what goes in app.py download the file app.py that contains the code skeleton for our application.

You will notice that the function contains four functions: *get_fused_data*, *get_features*, *home*, and *predict*.

The function *get_fused_data* will be used to preprocess, fuse and prepare our dataset when the application starts.

The function *home* is used to simply render the webpage with the form where the users can enter the location. You will notice that the function *home* is decorated with `@app.route('/')`. The decoration tells Flask to call the function *home* whenever the user visits the homepage URL on the website. The function *home* then renders the form we created in templates/index.html.

The function *predict* is used to process requests sent by the user. This function is supposed to convert the user inputs (coordinates) into a usable format, use the function *get_features* to get

the bands required by the model, pass the loaded features to the model, get the model output, and send a response to the user based on the model output. Like the function *home*, this function has a decoration. The decoration `@app.route('/predict', methods=['POST'])`. The decoration tells Flask to process POST requests sent to the URL/predict with the function *predict*.

The function *get_features* uses the coordinates provided by the user to fetch the features at a given location from the loaded dataset.

Exercise 8.2:

Complete the functions *get_fused_data*, *get_features*, and *predict* following the comments provided in the app.py file you downloaded.

Once you complete the above exercise, push your new code to GitHub using:

```
git add .  
git commit -m "Ready for deployment"  
git push origin main
```

and manually deploy the application to Heroku.

Testing the Application

Now that you have completed the code it is time to test your application. When sending a request for the location 30.06, -1.944 the model the following output is observed.

Land Use Application

Longitude Latitude

The area at 30.06, -1.944 location is built up land

A live demo of what is expected will also be shown on-site.

Exercise 8.3:

Display an error message showing that a user entered an out of bounds location whenever the location entered is outside the rectangular ROI we obtained our dataset from.