



Open in app

Get started



Published in Towards Data Science

You have **1** free member-only story left this month. [Sign up for Medium and get an extra one](#)



Frank Andrade

Follow

Apr 6 · 9 min read ★ · Listen



Save



# How to Easily Build Your First Machine Learning Web App in Python

A step-by-step guide to creating an ML web app with Flask in Python

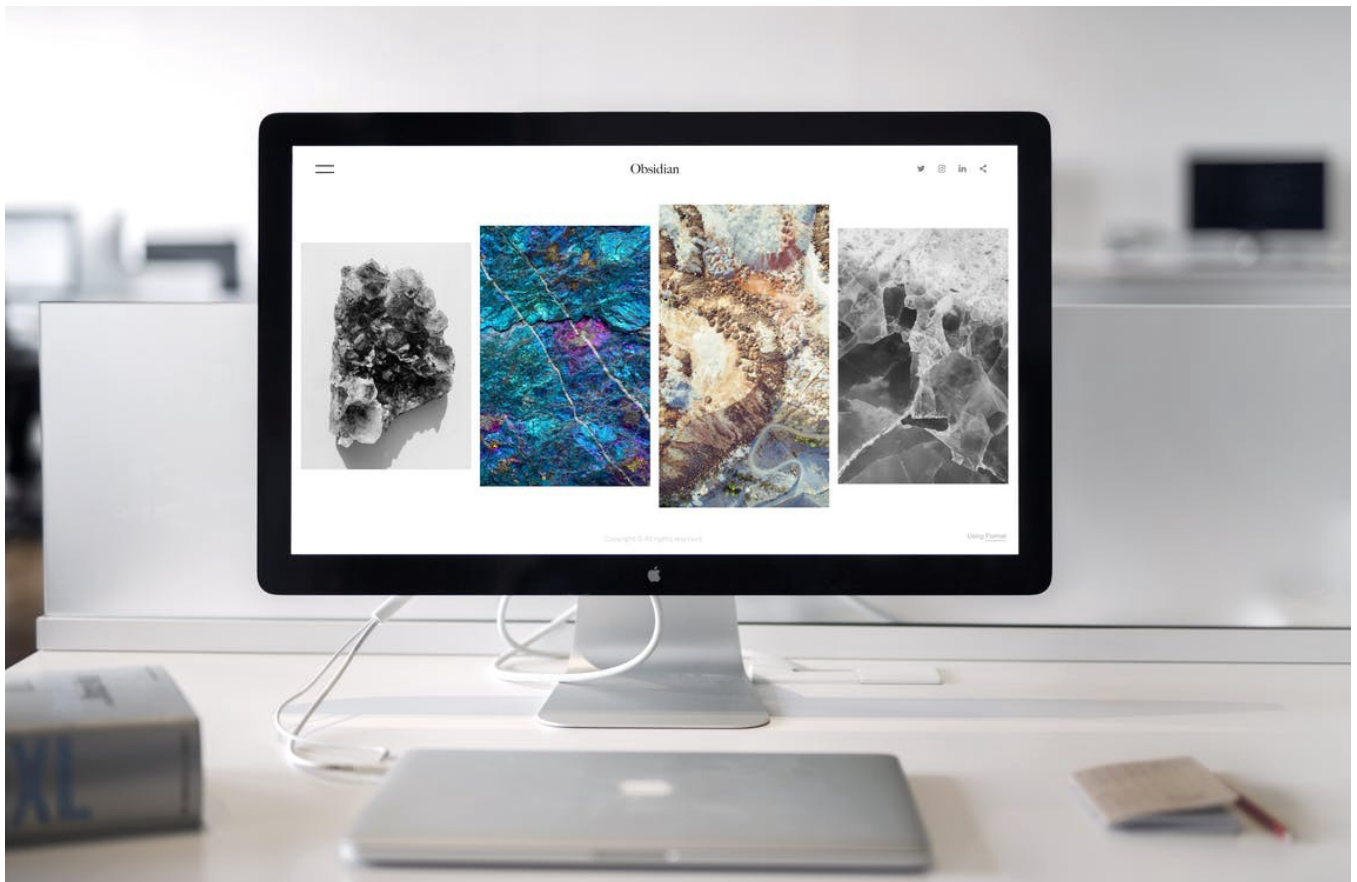


Photo by [Format](#) on [Pexels](#)



[Open in app](#)[Get started](#)

No matter how many models you've built, if they stay offline, only very few people will be able to see what you've accomplished. This is why we should deploy our models, so anyone can play with them through a nice UI.

In this article, we'll deploy a linear regression model with Flask from scratch. Flask is a Python framework that lets us develop web applications easily. After following this guide, you'll be able to play with a simple machine learning model in your browser as shown in the gif below.

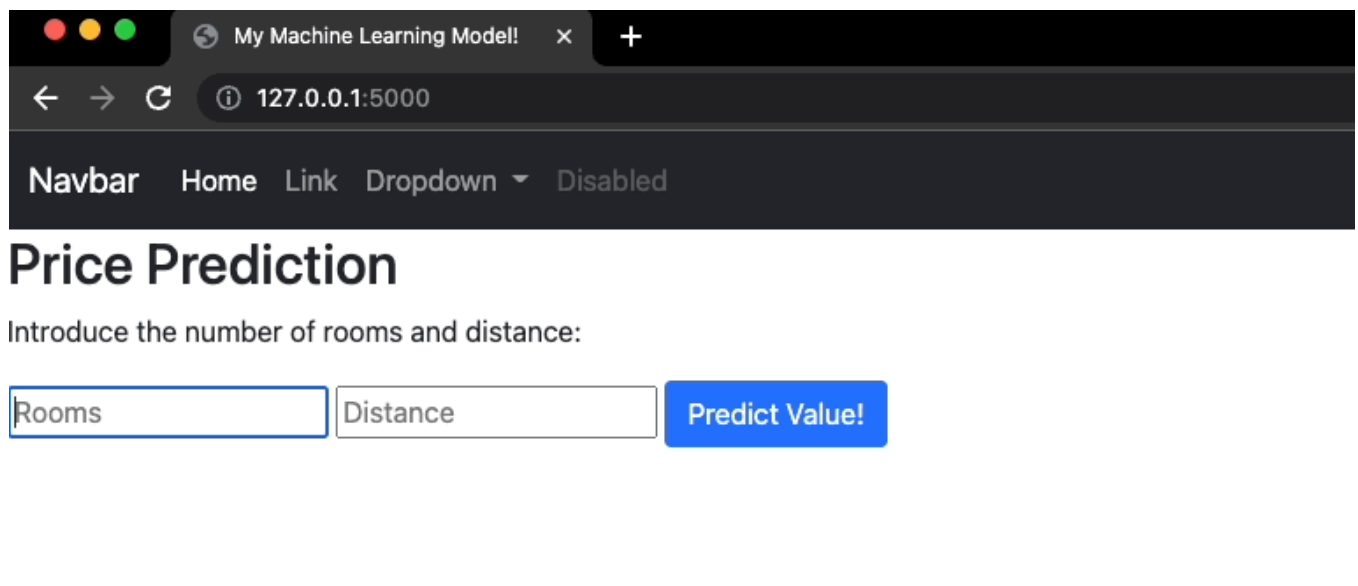


Image by author

This guide is split into two sections. In section 1, we'll quickly build our model and, in section 2, we'll build the web app from scratch. The main purpose of this guide is to build an ML web app, so you could go to my [Github](#) and download the model created in section 1 (model.pkl) and jump to section 2.

## Table of Contents

1. [Building a Simple Machine Learning Model](#)
  - [First things first – Libraries and Dataset](#)
  - [The dependent and independent variable](#)
  - [Fitting and saving the model](#)
2. [Building The Web App with Flask](#)
  - [Setting up a new virtual environment](#)
  - [Installing Flask and quick setup](#)
  - [Loading the model, building the home function and front end](#)
  - [Building the form and predict function](#)



[Open in app](#)[Get started](#)

## 1. Building a Simple Machine Learning Model

One of the simplest machine learning algorithms is linear regression. In this section, we'll build a multiple linear regression that predicts home values. At the end of this section, our model will be saved with pickle and, in the next section, we'll deploy our machine learning model with Flask.

### First things first — Libraries and Dataset

For this section, we have to install scikit-learn and pandas:

```
pip install scikit-learn  
pip install pandas
```

Also, we'll use a dataset that contains 3 columns: value (in \$1000's), rooms (number of rooms per dwelling), and distance (weighted distances to employment centers). You can find this random data generated with Excel on [Github](#) or on [Google Drive](#).

Let's have a look at this dataset using Pandas.



[Open in app](#)[Get started](#)

Rooms	Distance	Value
5.966	6.8185	16.0
5.613	1.7572	15.7
5.924	4.3996	15.6
5.757	2.3460	15.6
5.468	1.4118	15.6

Image by author

The goal of our linear regression model will be to predict home values based on the number of rooms a home has and the distance to employment centers.

### The dependent and independent variable

To build our linear regression model we have to define the dependent (outcomes) and independent variables (predictors).

Here are our dependent and independent variables.

- Dependent variable: “Value”
- Independent variables: “Rooms” and “Distance”

Here's the code to set both variables

```
y = df['Value']  
X = df[['Rooms', 'Distance']]
```

### Fitting and saving the model

Usually, you'd need to wrangle the data, train, and validate the results, but to make things simple, we'll focus on fitting and deploying the model only.

To build our regression model, we need to import `linear_model` from `sklearn`.



[Open in app](#)[Get started](#)

```
lm = linear_model.LinearRegression()  
lm.fit(X, y)
```

Once we built the `lm` model, we can predict the value given rooms and distance .

```
lm.predict([[15, 61]])
```

If you run the code above, the predicted value will be printed.

That OK! But now we want to deploy all of this on a website, so anyone can play with our model. We'll send everything from the backend to the frontend with Flask in the next section, but before we do that let's save our `lm` model with pickle.

```
import pickle  
pickle.dump(lm, open('model.pkl', 'wb'))
```

Note; As shown in the previous example, `lm.predict()` accepts inputs in dataframe format `[[[]]]` . Keep this in mind in the next section.

## 2. Building The Web App with Flask

Python offers different options for building web apps. Flask is one of the simplest options out there.

For this section, it's recommended to create a new environment. Working with different environments will help us better manage the different resources required for building a model and deploying a web app.

### Setting up a new virtual environment

For this section, I'll create a virtual environment called `deploy-mlmodel-env` running the following command on the terminal.



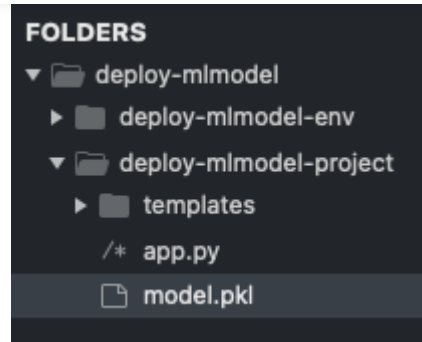
[Open in app](#)[Get started](#)

Image by author

Feel free to organize your directory as you want, but make sure that the `model.pkl` we saved in the previous section is in the same folder where the web app (we're about to build) is located.

### Installing Flask and quick setup

To install Flask, run the following command on the terminal.

```
pip install flask
```

Now to check out everything's working fine follow these steps.

Create a new file — I'm naming mine `app.py` — and paste the following code (that's a minimal app example extracted from the [Flask documentation](#))

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"

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

Open up a terminal, use `cd` to go to the working directory where `app.py` is located, and

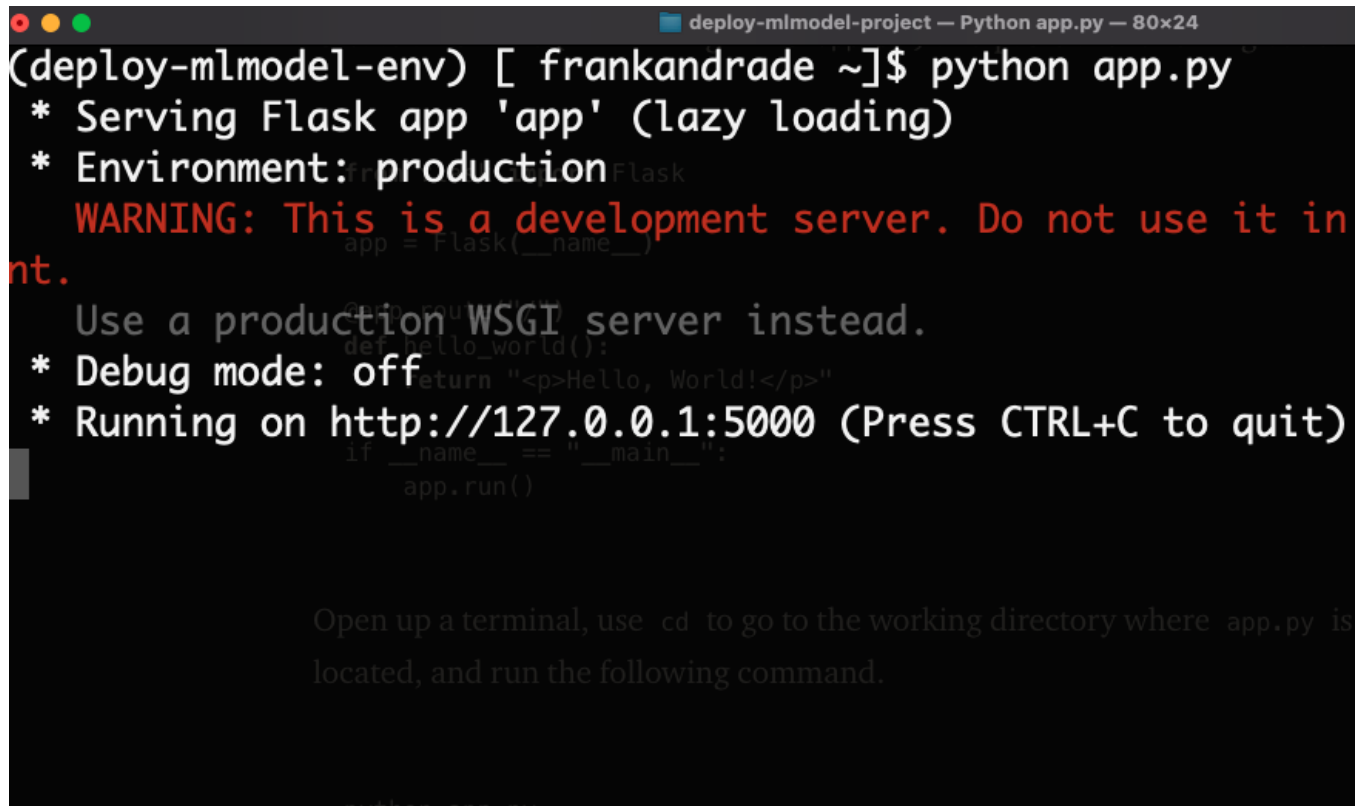




Open in app

Get started

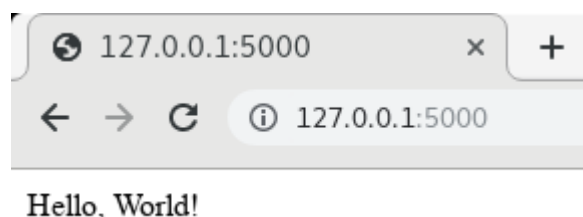
The following message should pop up.



```
(deploy-mlmodel-env) [frankandrade ~]$ python app.py
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in
  production.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```

Open up a terminal, use `cd` to go to the working directory where `app.py` is located, and run the following command.

The last line has a link with the format `127.0.0.1:port`. That indicates where our app is running. Copy the link (mine is `http://127.0.0.1:5000`) and paste it into your web browser. If, after pressing enter, you see the message “Hello, World!” everything was set up correctly.



Source: Documentation

Now let me explain to you what that little script does.

- `app = Flask(__name__):` Creates an instance of the Flask class. `__name__` is a variable that represents the name of the application’s module (this helps Flask know where to look for resources like “templates” which we’ll use later)



[Open in app](#)[Get started](#)

- `def hello_world()`: This function returns a message that will be displayed in the browser.

### Loading the model, building the home function and front end

Now it's time to start building our web app in the `app.py` file we created before. First, we import `Flask`, `request`, and `render_template`.

Then, we load the linear regression model `model.pkl` that we previously saved with pickle.

Also, we'll rename the `hello_world` function to `home`. This function will render an HTML file named `index.html`, every time we visit the homepage (`'/'`).



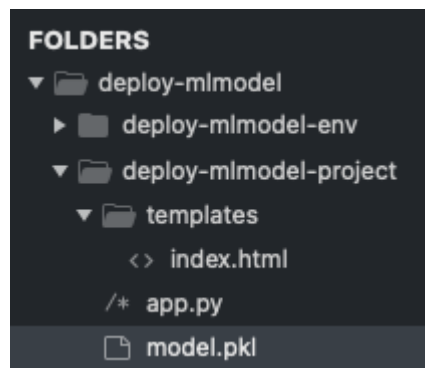


[Open in app](#)[Get started](#)

The `index.html` file will contain the front end of the website (navigation bar, buttons, images, etc.). Building the front end would usually require some fair knowledge of JavaScript and CSS. That said, we can still build cool web apps using only Bootstrap and basic HTML.

To build the front end, first, create a folder named “templates” (you can’t use any other name) in the folder where your app is located. Inside this “templates” folder create an HTML file. I named this file `index.html`, but you can name it whatever you want.

Your directory should look something like this.



Open the HTML file and write either `doc` or `html` and then press tab. If you’re in luck, your IDE will generate a basic HTML template. If not, copy and paste the following template.



[Open in app](#)[Get started](#)

Now to add bootstrap, go to this [website](#) and copy/paste the code inside the CSS section into the `<head>` section of `index.html`. This will load their CSS.

If you couldn't find it, here's the code to copy/paste:

```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
```

We'll also add a navigation bar for fun (we don't actually need it, but it'll help you get used to bootstrap), so go to the [navbar](#) section of the website and copy the code and paste it into the `<head>` section (right below the CSS)

Our `index.html` file should look like this (I know it looks scary, but it's only code we copied!)



[Open in app](#)[Get started](#)

Before running the script, I'll change the navbar to dark color by changing the `light` elements to `dark` in the first line of code (if you leave it as is, it'll have a light color).

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
```

Also, in the body section, I'll add a "Hello World!":

```
<body>  
  <h1>"Hello World!"</h1>  
</body>
```

Now to see all these changes, first, press CTRL + C to quit the current process running on the terminal. After this run again the command `python app.py`.

If you go to <http://127.0.0.1:5000>, you should see the following.





Open in app

Get started



# "Hello World!"

Image by author

Great! There's the title, the navbar, and the message. Now let's build the actual app that takes care of the prediction.

## Building the form and predict function

The predict function will use our machine learning model to make predictions. To do so, first, the end-user needs to introduce two inputs (rooms and distance) in a form.

We'll build this form inside the `<body>` section of the `index.html` file.

First, we create a `<form>` tag. This will contain the inputs the end-user needs to introduce.

```
<form action="{{url_for('predict')}}" method="post">

</form>
```

The URL inside the `action` parameter indicates where to send the form-data when the form is submitted



[Open in app](#)[Get started](#)

Then, inside the form section, we create 2 `<input>` tags and 1 `<button>`. The code behind them is straightforward, so I won't dive too much into it.

The important thing to know is that the input tags contain the room and distance variables, and to make the prediction the user needs to click on the "Predict Value!" button

Here's the code we have so far.



[Open in app](#)[Get started](#)

Inside the `app.py` file, we create a decorator with URL “/predict” that triggers a predict function and renders the `index.html` file.

```
@app.route('/predict',methods=['POST'])
def predict():

    return render_template('index.html')
```

Now we need to import `request` from `flask` to get access to the values the end-user introduced in the form.

The values inside the form are returned with a dictionary shape, so we have to use square brackets to obtain each value. Once we have the values we put them inside double square brackets `[ ]` to make the predictions (this is the format our `model.pkl` accepts)

Here's the predict function we're building.



[Open in app](#)[Get started](#)

Note that the `render_template` method has a new parameter I named `prediction_text`. This parameter contains a message that will pop up after the user clicks on the predict button.

This message is now in the back end. To send it to the front end, add the `prediction_text` variable inside the `<body>` of the `index.html` file.

```
<body>
  <div class="login">
    ...
    <b> {{ prediction_text }} </b>
  </div>
</body>
```

The final `index.html` looks like this:



[Open in app](#)[Get started](#)

And the final `app.py` looks like this:





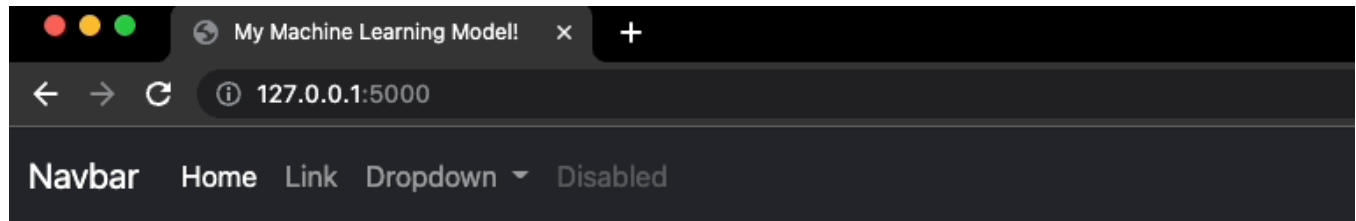


Open in app

Get started

That's it! Now quit the current process with CTRL+C, run the command `python app.py` in the terminal and go to <http://127.0.0.1:5000>.

You should be able to play with your model in your browser now!



## Price Prediction

Introduce the number of rooms and distance:

<input type="text" value="Rooms"/>	<input type="text" value="Distance"/>	<button>Predict Value!</button>
------------------------------------	---------------------------------------	---------------------------------

Image by author

**[Join my email list with 10k+ people to get my Python for Data Science Cheat Sheet I use in all my tutorials \(Free PDF\)](#)**

If you enjoy reading stories like these and want to support me as a writer, consider signing up to become a Medium member. It's \$5 a month, giving you unlimited access to thousands of Python guides and Data science articles. If you sign up using [my link](#), I'll earn a small commission with no extra cost to you.

### Join Medium with my referral link — Frank Andrade

As a Medium member, a portion of your membership fee goes to writers you read, and you get full access to every story...

[frank-andrade.medium.com](https://frank-andrade.medium.com)



[Open in app](#)[Get started](#)

## Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

[Get this newsletter](#)