**Deploying Machine Learning Models**

**Introduction**

Deployment of machine learning models is a step in the whole machine learning process where the focus is on how to make a good use of the model which was built. The users of the machine learning models need to be identified before a deployment method is chosen.

**Examples of Deployment of ML models**

1. **[Propensity of Home Loan]** Support we are working on a ML model which will predict the users (out of a bank’s existing customers) who are most likely to purchase a home loan from the bank. Let’s assume that model has been built and ready for deployment. In this case, we just generate predictions and share that with sales team. Or we can automate this procedure, so that every month this step is executed at an appropriate time.
2. **[Explicit Content Detection]** Now let us consider another very different problem. Suppose we are a video processing company just like YouTube, Vimeo Etc. We want to make sure that no explicit content is uploaded on our system. So, we built a computer vision system which automatically flags this type of content while the user is uploading the video. So, this type of ML model should be integrated with our existing web-based applications.
3. **[Fraud Detection]** Suppose we are a bank, and we want to filter those loan applications which are looking fraudulent. With this requirement, we have built a ML model for this. In this case, this model will be used by the banking front office personnel who are granting loan applications.

We can see in these examples that deciding who will use our model is of critical importance in deployment phase. This will decide what kind of deployment system will be built. These systems can be of following types:

1. Embedding ML models in a standalone web-based framework
2. Just sink the predictions in DWH
3. Embedding ML models in existing frameworks

In this chapter, we will look at some of these methods of deployment and address the challenges therein.

I hope you are excited. So, let us get started.

**Objective of this chapter**

In this chapter, we are going to look at various methods of deploying our Machine Learning Problems. We will look at how our model can be saved for later use, so that we don’t have to use it again and again. Also, we will look at how a model can be embedded in a web application (Flask based python web application). Then, we will look at the use of docker for packaging our application.

**NOTE FOR THIS CHAPTER**

***Further, this chapter is going to have different exercises than the previous chapters. In the previous 4 chapters, we had all the exercises where we gave you the notebook and asked you to follow the steps mentioned in the exercises. Now, there will be some exercises in this chapter, which will not follow the same exact patter of Jupyer notebook. For example, if we must create a docker container, we cannot do that in Notebook.***

Before we proceed further, open [**this**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD)link. This link will open an **online hosted Lab environment that** (ONLY FOR CERTAIN EXERCISES) can be used to run the codes discussed in this lab and try the exercises. You might have to wait a few minutes for it to fully load. This will have the labs, exercises, and project notebooks of all the chapters in this book.

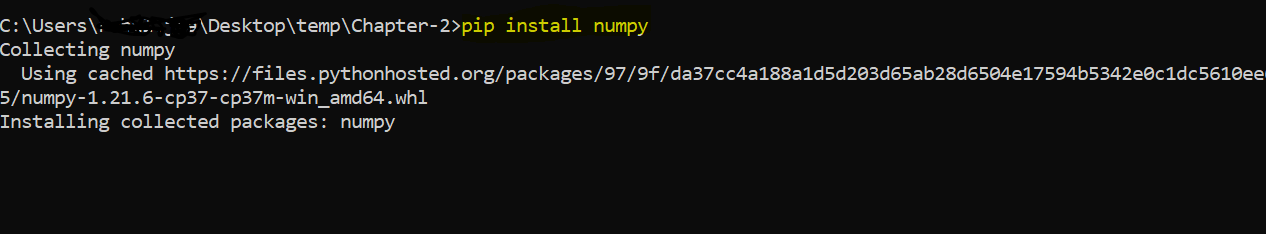
Link for online hosted environment: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

We will use the data used in the last chapter about the probability of a patient to get a stroke and the same can be found [**here**](https://www.kaggle.com/fedesoriano/stroke-prediction-dataset). It is also made available on the online hosted environment which we provided for you. So, you don’t need to download this data as everything is available on the online hosted lab.

Data Link: **https://www.kaggle.com/fedesoriano/stroke-prediction-dataset**

We will be using the Python programming language to do this guided project and the libraries you will need are “”, “” , “”, “” Etc. **You don’t need to install anything as we will be providing you with a hosted notebook on binder which you can run to execute codes as we move further into this chapter**.

But if you are working on your local system, then make sure that you have installed on your system. If it is not already installed, open a command prompt and enter the following command to install :



**Wikipedia content classification Project**

Though the objective of the chapter is to learn about **deploying machine learning models**, but for that we need to have **a machine learning service up and running to deploy it**.

So, in this first section, we are going to do an exercise where we will be building a Wikipedia content classification model. This model will take an arbitrary Wikipedia URL, grab its contents, and predict whether this URL belongs to any one of the following categories of topic:

* 1. Celebrity
  2. Physics
  3. Chemistry
  4. Economics
  5. Statistics
  6. Mathematics

We are not just going to learn about machine leaning models but also how to create our own Machine learning dataset with the help of web scraping.

**So, here is what all we will be looking at:**

1. Use web scraping to create a text data using web scraping
   1. We will grab names of celebrity from IMDB
   2. We will make a list of some common topics of the 5 subjects mentioned above.
   3. Then, we will grab the Wikipedia content of the topic names generated in the above 2 steps.
2. Once the data has been extracted; we are going to create some helper functions to clean the text generated from Wikipedia
3. Once the data has been cleaned, we are going to make a machine learning pipeline which will take those cleaned texts and create some machine learning features out of those texts.
4. Finally, we will train the model and make predictions for an arbitrary URL which was not available in the training dataset.

**In terms of python packages, we are going to learn the following packages:**

1. Pandas
2. Requests
3. NumPy
4. Beautiful Soup

Let us get started with the project in the following exercise.

**Exercise 5.1 Creating Wikipedia Content Classification app**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 5.1 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 5) and do the following:

1. Import the necessary modules:



1. Let us create the following helper function to grab the text of any Wikipedia page given its URL:



1. We will be creating our own training set. For that, we will grab a lot of Wikipedia pages and store it as text in python. Since, Wikipedia page URL follow the same pattern where you have wiki initials and then name is appended to the URL after the backward slash. So, we create a wiki initial variable



1. Note that, we had one of the categories as “Celebrity”. We can get a list of names of the celebrities from the IMDB. For that let us create a function to extract a list from the IMDB website:



1. Now, let us run this function on the actors of Hollywood:



We can now create a data frame from this and create a label “Celebrity”.



The above steps will be done many times since, we will have 6 categories, so let us create a helper function to do that. This will use the names list we got earlier and extract the data from Wikipedia using the .



Some URLs might not exist on Wikipedia, hence below we create a function to throw those data:



1. We have five other categories. We will create a list of names and then use the above function of all these five. The following is code for creating chemistry data:



1. Now, let us repeat the above steps 4 more times. For Physics, statistics, mathematics, and economics. You can find the list of subject names in a file called
2. Once these are prepared, we can run the following to combine the data



1. Let us also map the target as numbers. Also, we will create a pipeline to convert the text into features:



This creates a pipeline which can be used to make predictions.

**Predictions from the wiki model**

Once the model has been created, the idea was to use the model somehow. Let us start with **one of the simplest ways by which we can use the model**. Let us start by making predictions from the model in the following exercise.

**Exercise 5.2 Making predictions**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 5.2 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 5) and do the following:

1. Read the starter code from the previous exercise. Run the cells which are available and create them wiki-model.
2. Create the following helper function which is going to take a URL as its input and spit out the data which can then be used to make the prediction:



1. Let us now use this function to see what it produces:



1. Now, let us create one more function which will take the function we created in the previous step and use to produce the result:



1. Now, finally, let us use this function (by passing the URL to predict and the pipeline model we created in the last exercise):



It should predict the result of the “Machine Learning” Wikipedia page.

**Saving the Model as a Pickle File**

This model took us **some 5-10 minutes to created** (including the time spent scraping the Wikipedia data and training the model). Sometimes, in real world setting, when the data **dimension is in millions or billions, then this time can take up to hours or even days**.

So, every time when we are making a prediction, **we would not want this training to happen all over again**, **we might want to save the model’s important trained values** (which are also called as weights in machine learning), so that those weights can be used later while making the prediction.

Though it is possible to save the weights, but in python, there is a better way to just **save the python object itself in its current state** while saving all the values such as **model coefficients and weights** as it is and then use that object in a new python file.

The process that I just discussed is known as **“Pickling”** in python which means to save the python object in its current state with all the trained values.

**Let us look at this point using an example.**

Suppose I have a list of 10 values and if I pickle that (I.e to save that as binary form), then I can load that list in a new python file with exact same 10 values. In the same manner, we can even store the trained model pipeline. Then, the same model pipeline can be imported in a new python session.

**You may ask “What is a binary file anyway?”**

So, binary file is what is the default format of any file stored in a computer. Every file is stored in a binary form, but their interface may differ to make it easy to use by the user. Remember, everything is stored in a binary form in computer and binary form means 0s and 1s. So, we store our python object as 0s and 1s.

**Exercise 5.3 Using Pickle Format**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 5.3 file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 5) and do the following:

1. Read the starter code from the previous exercise. Run the cells which are available and create them wiki-model. It will also create the two helper functions which will help us in making predictions.
2. Import the pickle module:



1. We need to save the pipeline which we created in the Exercise 1. It can done using the following steps:
   1. Open a file in write-binary format ()
   2. Call the pickle module’s dump method to dump the pipeline python object to the file we just opened
   3. Close this file

All these steps can be done using the following two lines of code:



1. The same three method can be used to load that newly created python pickle file and use it in a different session. Now, we no longer need to run those modelling steps all over again.
   1. Open a file in write-binary format ()
   2. Call the pickle module’s dump method to dump the pipeline python object to the file we just opened
   3. Close this file



The only difference with the previous code is that this time the file is being opened with the “Rb” format which stands for read binary format.

1. Now, we have access to same pipeline object which we had in the earlier steps, Now, it can be used to make predictions.



Note here that we are passing the pipeline we just loaded from the pickle.

**Version management**

So far, we have an app which run on own system, and it also makes a prediction. We could also save the model as binary file so that we don’t have to run the models all over again. This is good but there is a limitation and **that is only we can run this app on our local system**.

We can’t **for example share this with someone else**. Because to run this application, they should have **python installation on their system**. Not only that, but they also need to have those packages (which our application is dependent on) installed on python. So, we can have two sort of dependencies for our app:

1. Python dependency
2. Packages dependency.

**Solving the packages dependencies problem**

To overcome the packages dependency, we can make a list of all packages our application is dependent on and share this list with the package code. Now, they call install those exact versions which were used by our model building process.

We call this list as requirements file and we will see it in action in the next exercise.

**Why same exact version is needed?**

The same version will be required to exactly reproduce the results which we got on our system. This is because as time passes, the package evolves, and it may be the situation where the new package’s code is not compatible with the old one. So, whoever install new packages for the old package code will be facing trouble running the code.

**How do we solve python dependency (or system dependency)?**

One possible method is to use **docker containers**. These are really wonderful way to package the app and its dependencies (including python) and make an image out of that. Image is just a fancy way to say that this contains all the dependencies and python. Anyone with the image and docker installed on their system will be able to run the image on the docker app. That image in turn will run our app. We can create images for all sort of things.

**What is system dependency?**

The system dependency just means that some packages and files only run on certain operating system and not on all. So, for these types of applications, we might need to be careful. Docker again helps to sort out the system requirements and system dependencies.

**What is a Docker Container?**

The docker container is just a image which is running. So, a running image is called a container because it temporarily installs the system dependency and package dependency on the host machine (a machine which is being used to run the image)

**What is a Docker Image?**

You can think of Image as a program which has instruction on which version of packages and python is being used.

**Why to install Docker?**

Now, one question might arise that why do we even need to install Docker? You can think of docker as middleman which is managing these dependencies exchanges.

**Can we also replace Docker?**

Yes, we can. We can deploy our app on some web application and run the web application on a given network. So, anyone in that network will be able to run the app by going to the IP address. We can go even further. We can deploy our app on the cloud and then share the URL to everyone. Then, everyone on this earth with the internet connection and the URL of the webpage will be able to access the web app.

**How do make a web app?**

To make a web app for our machine learning model, we can use flask (which is a micro web framework) in python. It can be used to speed up creating the servers and removes the hassle of creating these request and response handling manually.

**So, to summarise, we have discussed the following methods:**

1. Making package versions
2. Creating Docker contains
3. Making a web app and run it on network
4. Making a web app and deploy it on Cloud

**Further, to deploy your on Cloud can have several options such as:**

1. Microsoft Azure
2. Google Cloud Platform
3. Amazon Web services
4. Heroku
5. Digital Ocean
6. Etc.

In the next four exercises, we will discuss these four methods of deployment in detail.

**Exercise 5.4 Creating the requirements file (Package version management)**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

1. We could just run the following command from the terminal to create a requirements.txt file with a list of all the packages currently installed on our python system:



what this command will do is to install all the packages which are currently there on our python system. It will work **but is inefficient** because our app might just be dependent on a few packages such as , panadas, pickle etc.

This command on the other hand will just install all the packages currently there on our system. Let us look at how can we possibly solve this problem.

**Virtual Environments**

Virtual environment is a **separate mini python within our system** which is going to be **separate** from the original python installation. This way, we can install the packages required by our app **within this environment**. We can create unlimited number of virtual environments within our system and each of them will work on their own without being dependent on the main python installation or on other virtual environments in our system.

There are many packages which enable us to create these virtual environments. We will be using in the exercise. **But all these virtual environment packages should have**:

1. A method to **create** virtual environment
2. Once created, to **activate** that environment. This is required because once we run pip install or python run, we should make sure that these uses this virtual environment instead of the original python installation on our system.

**So, the ideal process of working with virtual environment is:**

1. Create the virtual environment
2. Activate the environment
3. Work in this environment
4. Install packages as and when required.
5. Once your work is done, deactivate the environment
6. If you wish to resume your work, re-activate the same environment
7. Once your project is completed, create a requirements.txt file to share with other.

Now, let us work with these virtual environments.

**Exercise 5.5 Virtual Environment**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

1. Open command prompt on your system.
2. Create a new folder where you wish to include all the project related files (suppose the name of the folder is “temp”)
3. Now, run the following command:

Text

Description automatically generated

Here, is a package inbuilt in python which is used for creating and manging environments. is the name of the environment

1. After entering, it will create the virtual environment with the name **.** We can activate this by running the following command**:**

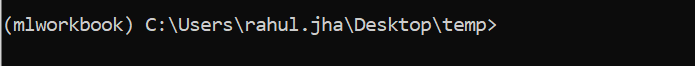
Text, logo

Description automatically generated

When you hit enter, it will activate the names environments. After this

activation, if you run pip install, it will install the packages within this environment.

To identify that your virtual environment is active, you can see on the left-hand side of your command prompt, there will be a name of your environment.



1. while this environment is active, we can run the same freeze command we ran earlier and now this time, it will only make a requiremnst.txt file with the packages of this environment.



1. To deactivate, we can just run the following command:

Text

Description automatically generated

**Creating Flask Web App**

Let us take the app (created in the first exercise) and turn that into a web app.

Before discussing any of the technical details, let us think and plan about what exactly we want to create here.

***We want a web app where there will be an input where we can enter the URL and then when we hit submit, it will give us a prediction of our web app.***

Let us briefly discuss Flask, HTML and CSS

**FLASK**

To learn about web application development, we need to know about the HTML and CSS which are two of the most important languages of web. Let us briefly talk about what is flask and how does it work.

To make a flask app, there are following steps:

1. Make an app variable using the **FLASK class**
2. Then define the routes for app. For each route, we need to declare a function. You will notice that we have made a function called before which we do something like **.** This is a python function decorator. It means that it tells python that the function which proceeds after this decorator is a special function and, in this case, it is a route function which will be executed once user tries to access our application
3. Then, we must decide what we want to show a user. The look and feel of the application are decided by **HTML** and **CSS**. We won’t cover these in detail as this is not the scope of this book.
4. Finally, we can call the run method of our app.

The code for the steps would look something like the following:



In this very simple and minimal application, once the server has started then the user will be greeted with a simple text "Hello, World". Now, we can put HTML code in the return statement but that would make the function very difficult to read, and this why we have clearly separated the string of the **HTML**.

**HTML**

HTML stands for **Hyper Text Markup Language** and it is the language of the web. With the help of HTML, we can define the structure of the web application that we want to build.

HTML decides the structure of a web page. Whatever you see on the web page is structured because of HTML. Now, everything on our Wikipedia content classification application` is also structured using `HTML`.

But note here, HTML only handles the structure of the application while the look and feel of your application is handled by another technique called CSS.

**What are tags in HTML?**

Tags in HTML are made up of opening and closing tags. Opening tags looks like

and closing tags looks like .

Let us take the example of our application. It has an input box, a submit button. The first is the header of the application which is handled by header tags. There can be various types of header tags.

1. H1 header tag
2. H2 header tag
3. And so on…

These all are used to place header for the HTML content. Each of them differs from each other because of the size.

Taking h1 as an example, the opening tag would look like <h1> and the closing tag would look like </h1>. The content will be placed in between.

Now, there are some types of tags which are self-closing which means for those tags, there are no closing tags. input tag is one of the examples self-closing tag.

**One special tag <style> </style>**

This tag holds all the CSS component of our app which is more of aesthetics. This is not important for us as our app is a simple HTML based app for now, obviously you are free to explore more and add more on your app (whatever you feel right).

**GET and POST**

The final application will be having two different states. One, where the user of the application will be shown the input box and some information about the application. Another state is where the user will submit the `URL` and tries to get the prediction that URL.

The user when visits our web app, **he is making a GET request** and when the user submits some form (in our example it would be the URL parameter, and hits the submit), the user will be shown the results of the web page, the result of our ML algorithm, **he is making a POST request**. Basically, he makes a POST request and then requests for predictions.

**Route**

Route is just a fancy word for saying URL. Now, the URL which the user is going to enter in the input box is different from this URL.

A web application is just like any other website you visit online. Now, when you visit any website, you enter some URL and that URL is what we are talking about here.

So, since we want our web application to run on our own system. We don't want to share it with world. So, we must create some way to open the web application on our local system.

Hence, we create route for this purpose.

So, before the function , we have a statement which starts with `@` which has a special name in python `decoraters`.

Inside this route we say that the URL is `'/'` and we will accept two types of methods `GET` and `POST`.

**METHODS**

Remember, when we talked about the states of the application. There were two states:

1. When user visits the web app for the first time (GET request)
2. When the user asks for prediction by posting the URL of the Wikipedia page and hitting the submit button.

Now, let us create a simple web app using the file which we created in earlier exercise.

**Exercise 5.6 Flask app (running locally)**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Create a python file and enter the codes given below. Please do it on your local system. As the flask app will run locally then

The exercise folder contains the which is the model we saved as a pickle file. This file will be present in the chapter 5 folder. Now, let us create the app:

1. Create an app instance:



1. Create a route and run the app:



1. Let us add HTML to render GET requests.



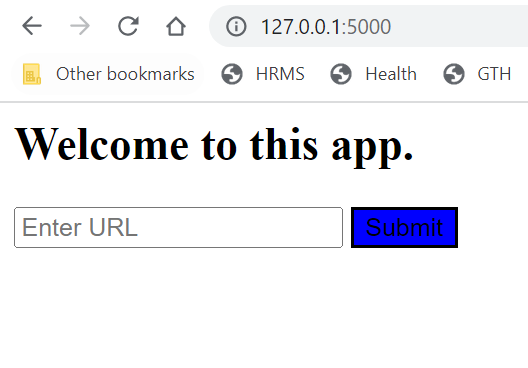
Here, we have two inputs, one for URL and one for submit.

Let us also add the POST case.

1. While making a post request, we will extract the URL and use the function we created earlier to predict the results and render them:



1. Now, if we run this app, we will see the following screen:

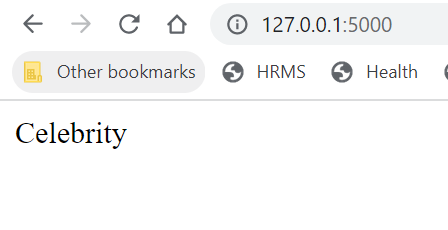


Let us test this for Jordan Peterson wiki page:

Graphical user interface, text, application, chat or text message

Description automatically generated

When we hit submit, it will give us the prediction:



Now, we have this app running on our local system. We can now create a docker container from this web app and share it with other without worrying about the package and system dependencies. In the next exercise, we will see how to take this app and create a docker container out of this app.

**Exercise 5.7 Docker Container**

Link: **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

In this exercise, we will take our Wikipedia application and create a docker container our of that app. In this way, going forward, we can share the image of our application, and everyone can run that image given that they have docker installed on their system (without worrying about other package requirements and system requirements).

1. Download docker desktop [**https://docs.docker.com/desktop/install/windows-install/**](https://docs.docker.com/desktop/install/windows-install/)
2. Download WSL (window subsystem for Linux). This is only required so that docker can run smoothly as docker depends on this.
3. Create a file named and have the following code in it:



* The first line grabs an image for python 3.7 (yes, our image can grad other images also, it grabs it from online docker hub) version and then created a working directory with the same wiki-app.
* Next, it copies the requirements.txt file which is present in the current directory to wiki-app.
* Then, it run pip install to install the packages present in the requirements file. It also happens in the new directory.
* It then copies all the contents of the directory (in which will be run, while creating image) to wiki-app.
* Final line refers to what **will be run when the image is going to get executed** by its user.

**What is ?**

In short, this file specifies two things:

* How the image is to be created? (Line 1 to 5, FROM to COPY)
* Once image is created, what exactly will be run once the image is run? (Last line CMD specifies this)

1. Once docker is installed and you have created a docker file, the next step is to create the image. To run the image, navigate to the app folder (this is the folder where the ) should be and run the following command:



This will start the build process. It will take some time. After it is finished, you can look at the list of images using the following command:

Text

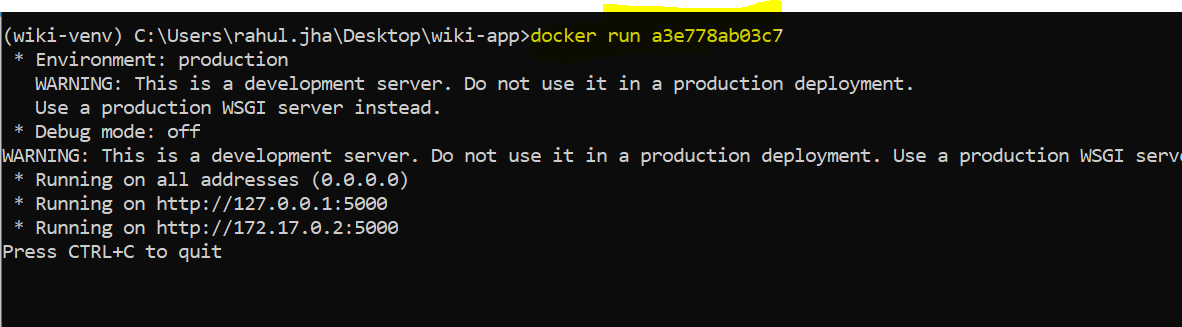
Description automatically generated

Since we did not provide any name for our image, it is showing <none> here. Also if you open the docker desktop, you might see something like the following:

Graphical user interface, application, website

Description automatically generated

1. You can share this image by uploading it to . Then, people can install this image by running
2. You can also run the image locally by running: (Note that we have provided the IMAGE\_ID which we got when we typed



This has started our images and now we can navigate to the URL to view our app.

**Deployment to Cloud**

So far whatever we have been doing has been restricted to our local machine, we never upload anything online, so it might have a potentiation for whole world to see it.

There are many cloud providers which offer python deployment options. Some of these are:

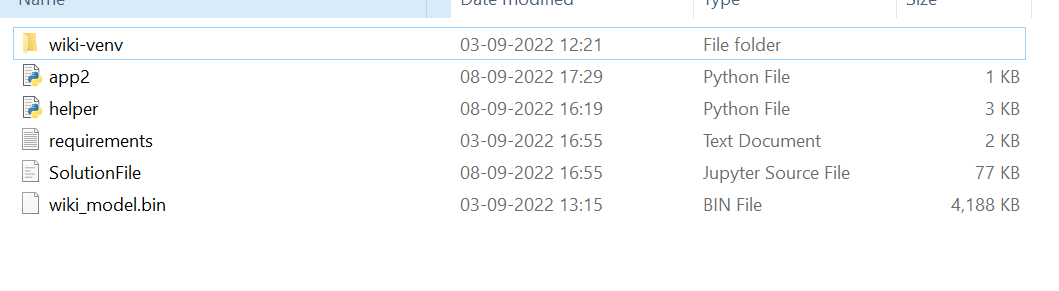
1. Microsoft Azure
2. Google Cloud Platform
3. Amazon Web services
4. Heroku
5. Digital Ocean
6. Etc.

We will look at uploading to few of them. These platforms make it easy for us to put our websites online. We have our flask web app ready. Let us start by seeing how to upload our website on Heroku.

**Exercise 5.8 Deployment to Cloud. 1 (Heroku)**

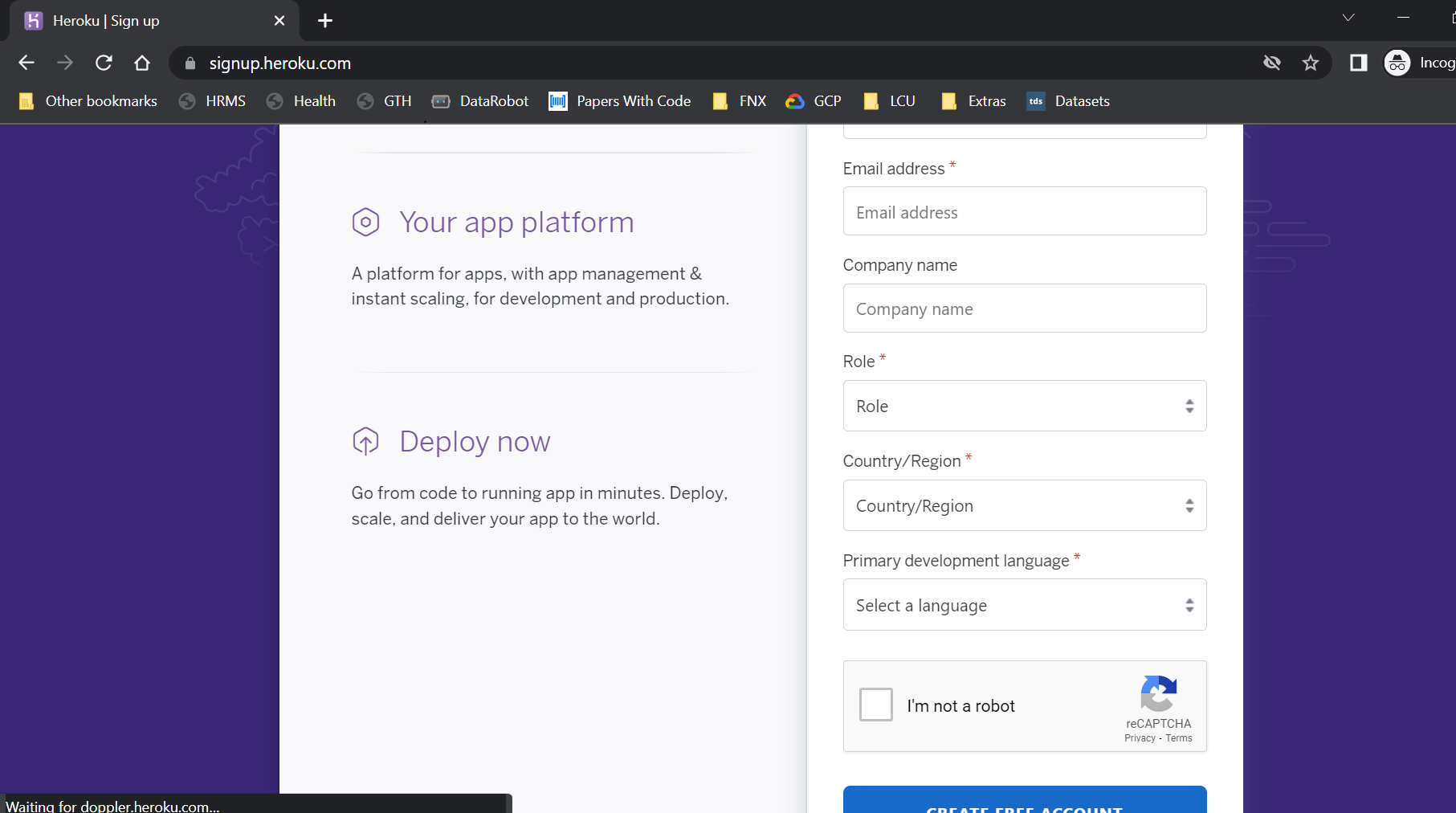
From the previous exercises, you should have a flask wiki app running on your local system. The idea is to push this app on Heroku servers so that everybody can access this website and play with the app which we created.

Before starting this exercise, you should make sure that the current folder state of the web app should look something similar to the following:



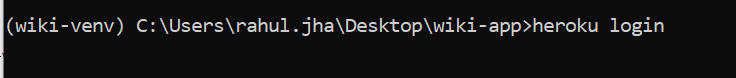
There is a requirements file, flask app python file and the virtual environment with the name -. Also, we have a helper python file which holds the necessary helper functions such as making prediction, loading pipeline from the binary file Etc.

1. The first step is to navigate to Heroku and create a free account. The sign in page will look like the following:



1. Once, you have created a free account, the next step is to download the Heroku client with which we can interact with our Heroku account and push our website to their server. Install it from [**https://devcenter.heroku.com/articles/heroku-cli#install-the-heroku-cli**](https://devcenter.heroku.com/articles/heroku-cli#install-the-heroku-cli)**.** Note that there is a prerequisite for this as git. Let us install that in the next step.
2. We will also need git installed on our systems. Install it from [**https://git-scm.com/book/en/v2/Getting-Started-Installing-Git**](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git)
3. Next, after the installation of the above two piece of software result in success, we need to login to Heroku from our command prompt. The idea is that we can enter command from the command prompt and it will open browser and then you just have to give your acceptance.

The command will look something like this:



When you hit enter, it will result in the following result:

Text

Description automatically generated

This will result in the following page on your website:

Graphical user interface, application, Teams

Description automatically generated

You just need to tap into that login button. It will ask you to return back to the command prompt, something like this:

Graphical user interface, application, Teams

Description automatically generated

And the resulting command prompt will look something like following:

Text

Description automatically generated

1. Once we have logged into Heroku, the next step is to create a for the Heroku. It just tells how to run the python file (app). Our app is named as app2, so we will create the following file. Note that should be installed in the virtual environment and it should be pushed to the requirements file.

Let us first create the with the following content in it:

Graphical user interface, text, application, Teams

Description automatically generated

Now, let us install in our virtual environment:

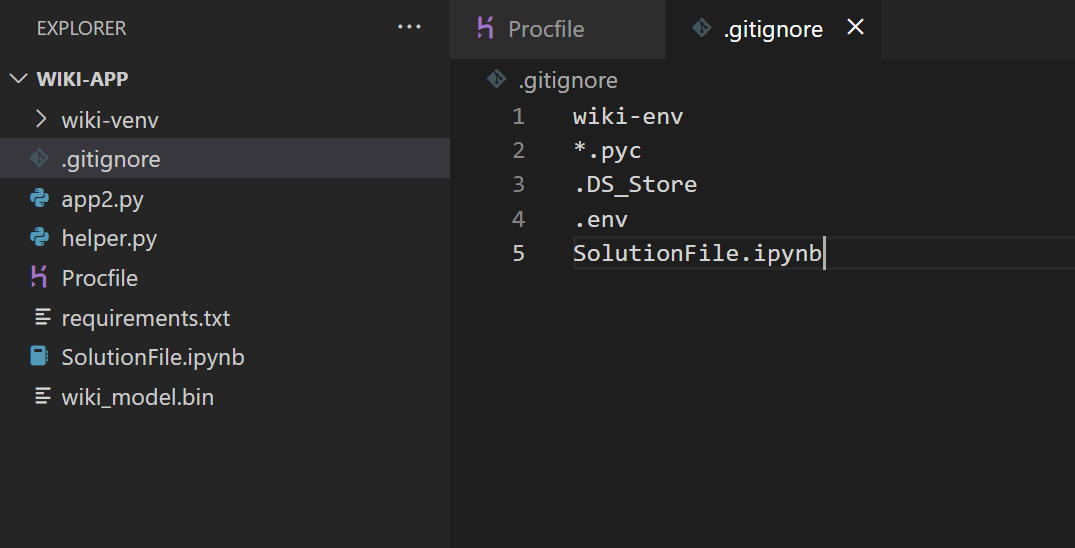


Finally, let us freeze all the requirements once again:

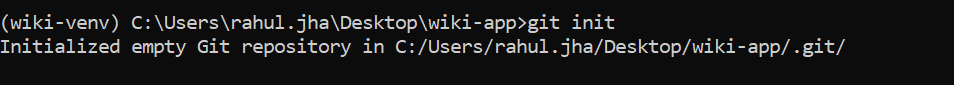


1. After the creation of the , the next step is to initialize a repository. But let us create a file where we list all the folders and files we need to ignore while initializing git.

Here, we have listed all the folders (such as virtual env) and notebook files.



Now, we can initialize the git from the command prompt and git will ignore these.



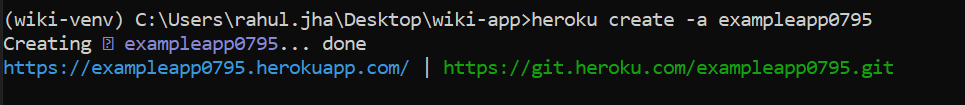
Now, let us add all the files (ignoring the files and folders from git ignore file):



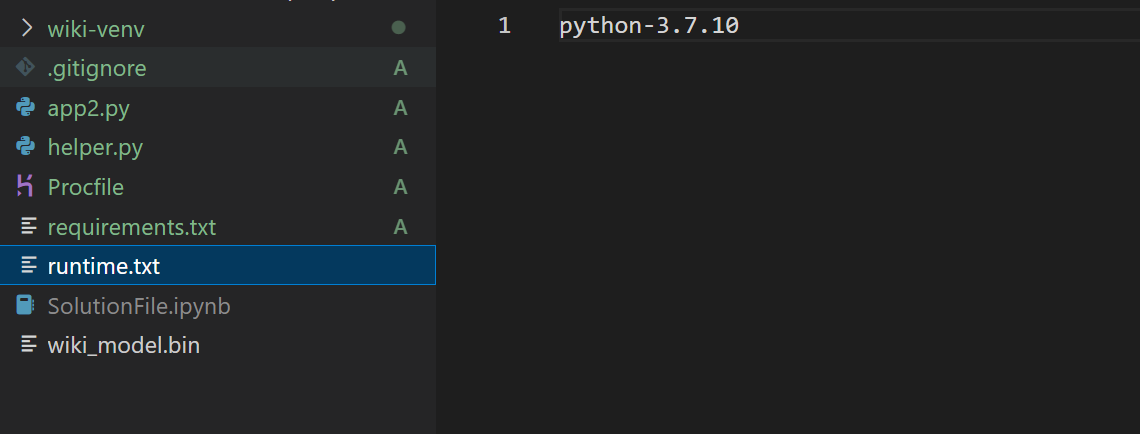
We can now commit these changes by:



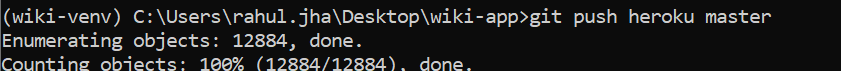
1. Let us create the app from command prompt using Heroku create:



Also create a new python file for specifying the runtime of the python.



1. Finally, let us push the changes we made in git:



This code pushes our code to the app we just created in the 8th step. It will take some time and after that you can navigate to the link, we got in step 7 to access and visit our app.

**Activity**

1. Create an image data by scraping 100 images of cat and 500 images of non-cat (any object will do; it must be a non-cat).
2. Convert the images into 28 by 28 shape using NumPy.
3. Create a data frame of shape 600 by 784.
4. Use dimensionality reduction techniques to reduce the number of columns to 20.
5. Use any machine learning classification algorithm to build a model to identify cat.
6. Save this model as pickle file
7. Create a machine learning flask app with following features:
   1. It should have an input element with an ability to upload images
   2. It should not accept any other file
   3. It should have a submit button.
   4. After submitting, it should tell the user the prediction of the model in terms of probability.

**Project**

Make a credit score classification web app using the Kaggle data available here, also try to have a docker container and share your project with your friends and make sure they can run it. [**https://www.kaggle.com/datasets/parisrohan/credit-score-classification**](https://www.kaggle.com/datasets/parisrohan/credit-score-classification)

**Summary**

In this chapter, we created a **Wikipedia content classification application** which takes an arbitrary URL and predicts **its possible category** out of the 6 categories which we manually created. We did that by creating our own ML data.

The main objective of the chapter was to learn about various deployment options for our machine learning models. We saw how we can make predictions using a saved binary version of our model and how we can export that to others using DOCKER.

We also learned how to manage our virtual environments to separate out concerns for specific web apps from other apps using .

We also looked at how we can create a flask web app for that machine learning model both locally and on internet.