**Bank Telemarketing Classification Using Artificial Neural Network**

**Introduction**

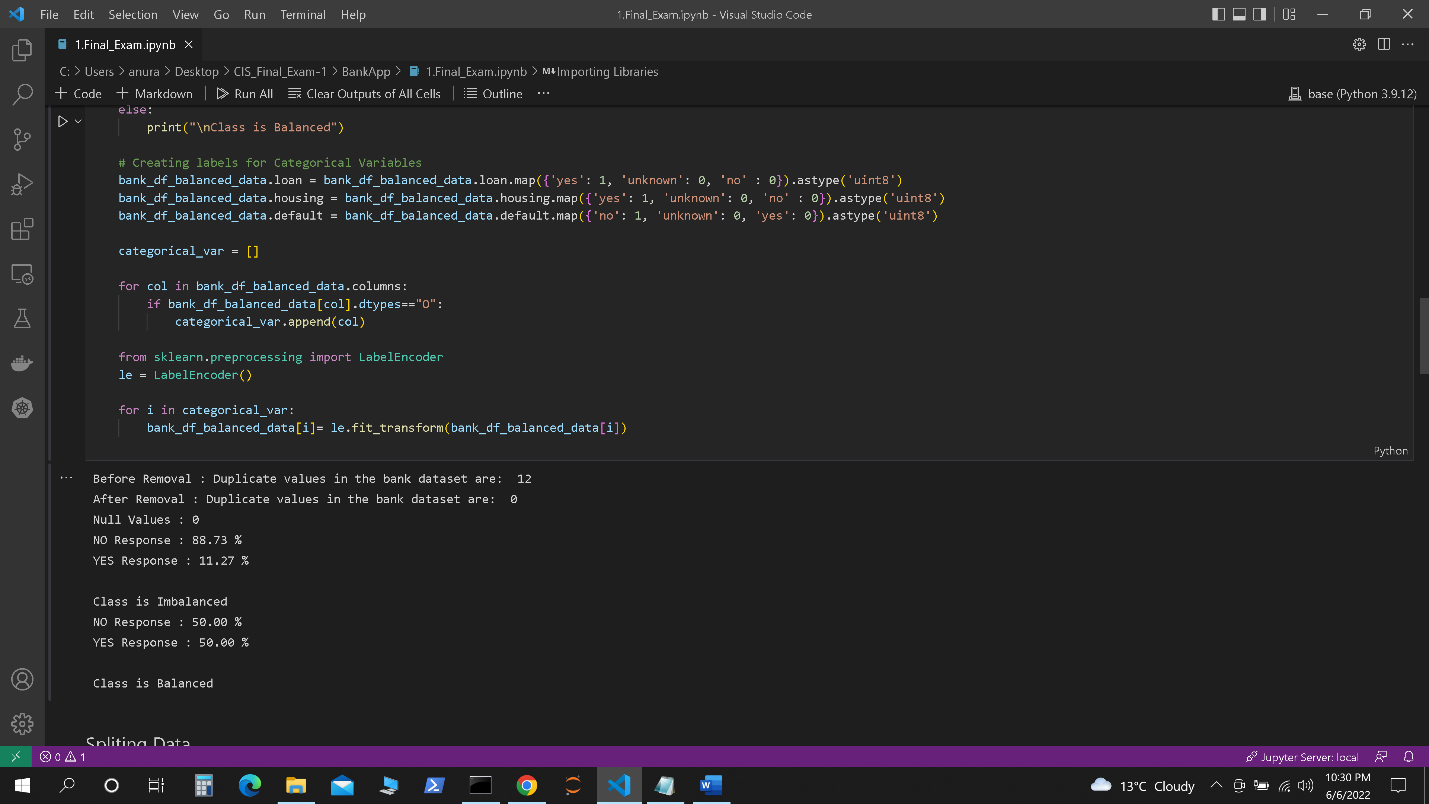
The data is related with direct marketing campaigns of a banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

Bank-Telemarketing.csv with all examples (41188) and 20 inputs, ordered by date (from May 2008 to November 2010). The classification goal is to predict if the client will subscribe (yes/no) a term deposit (variable y)

Download Link: <https://archive.ics.uci.edu/ml/datasets/bank+marketing>

**Pre-Processing**

The data that I used for this project was highly imbalanced and was dominated by NO responses, while only 11.27% responses were YES. This means that the data need to be processed first. Using **sklearn.utils.resample**, the data have been balanced.

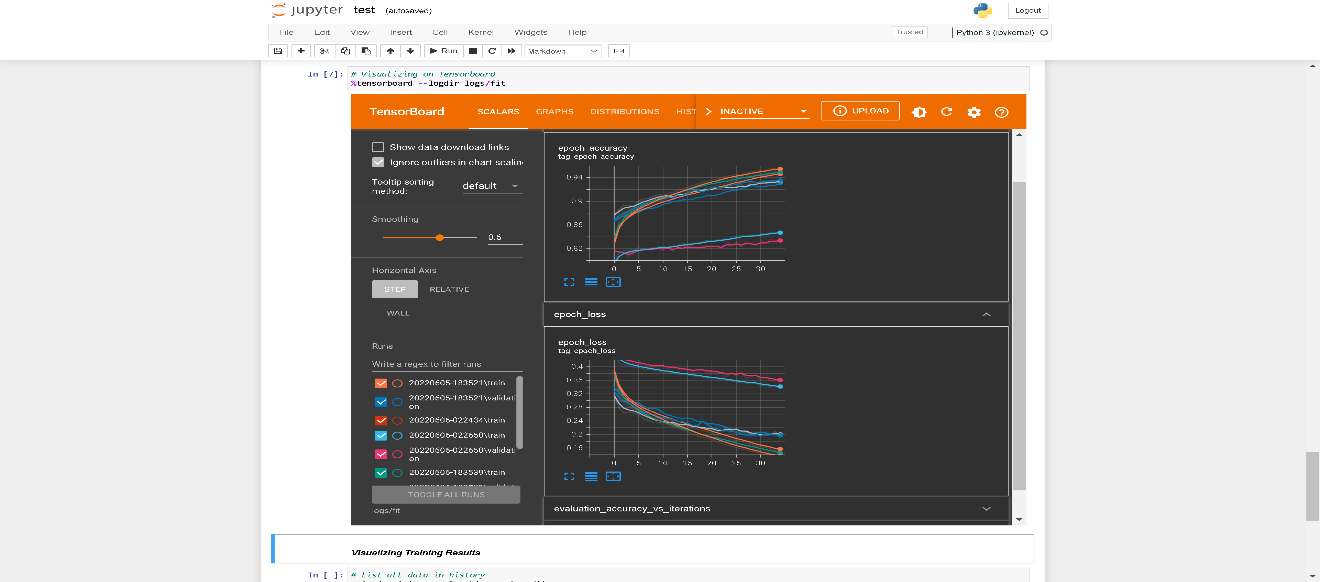


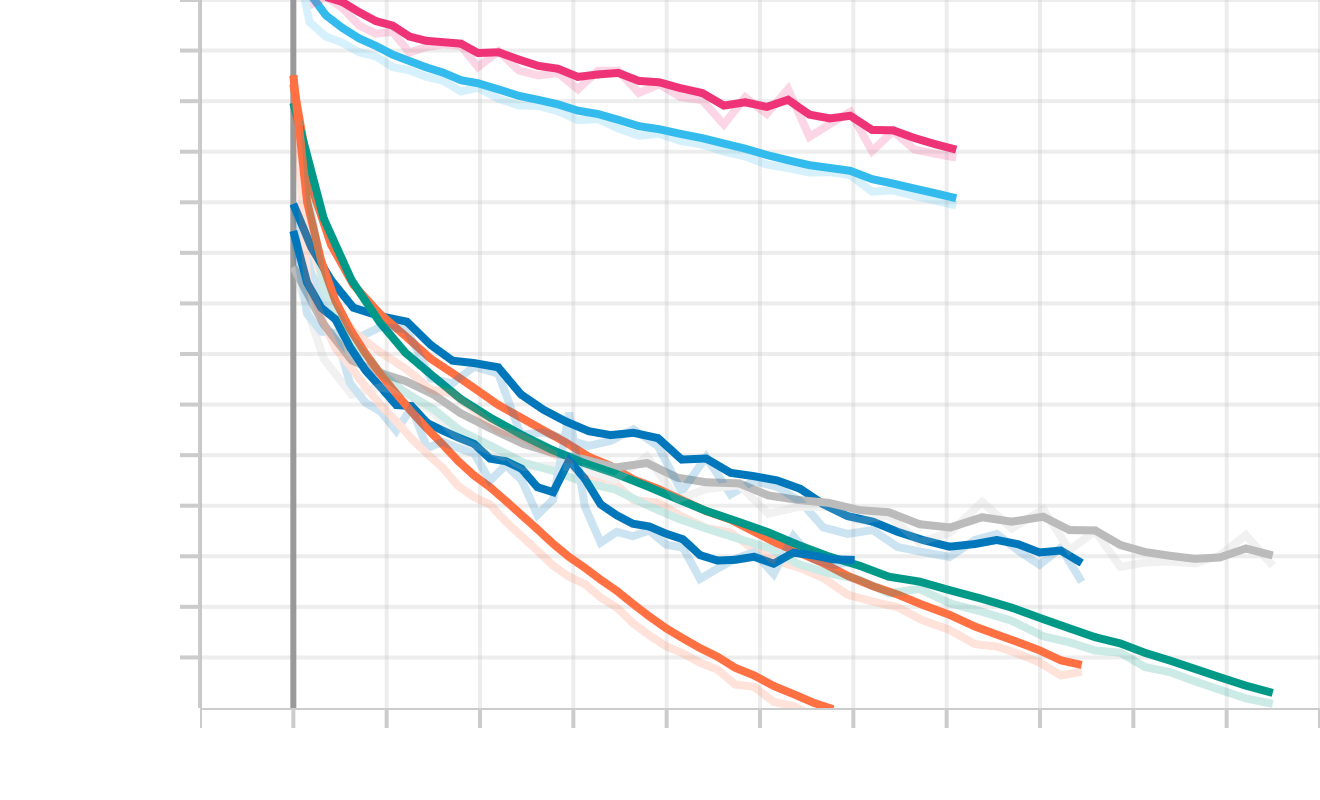
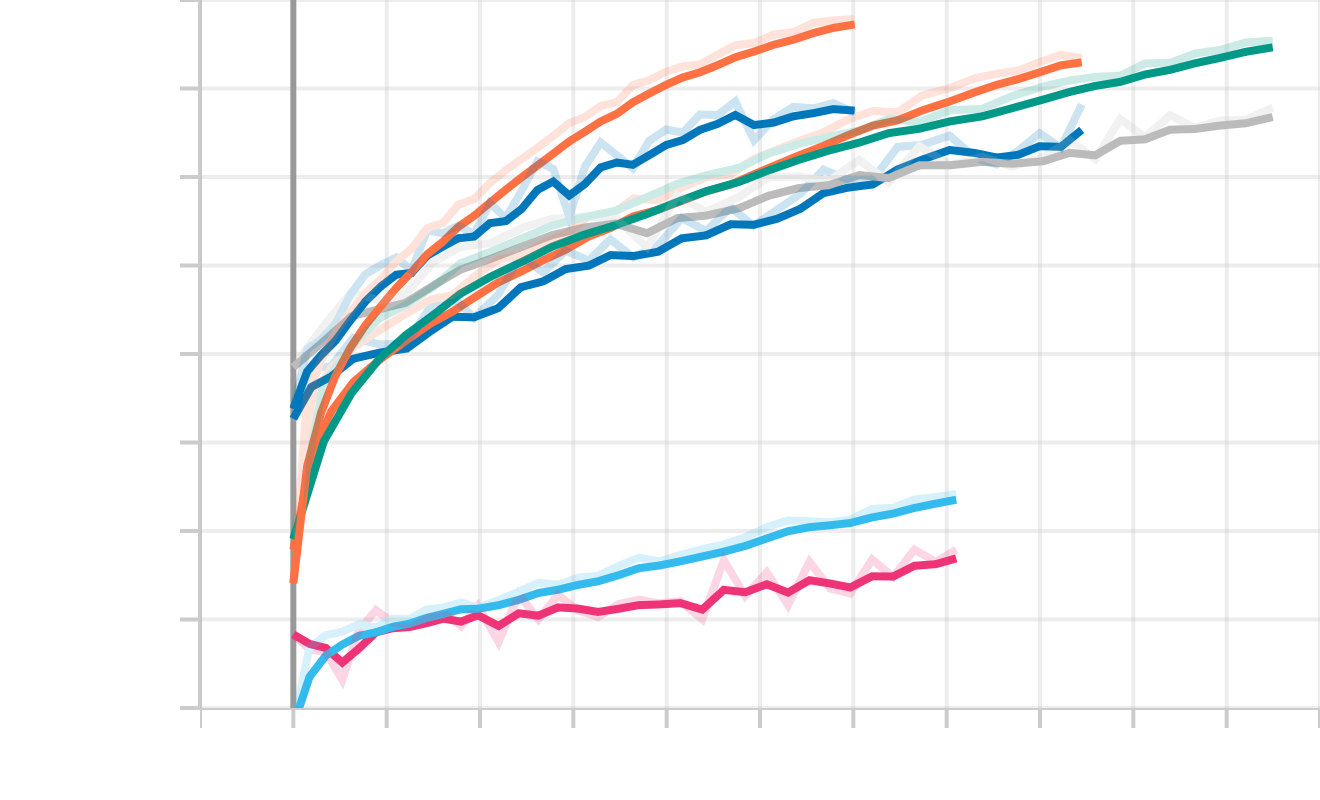
**Project Workflow**

**TRAINING MODEL**

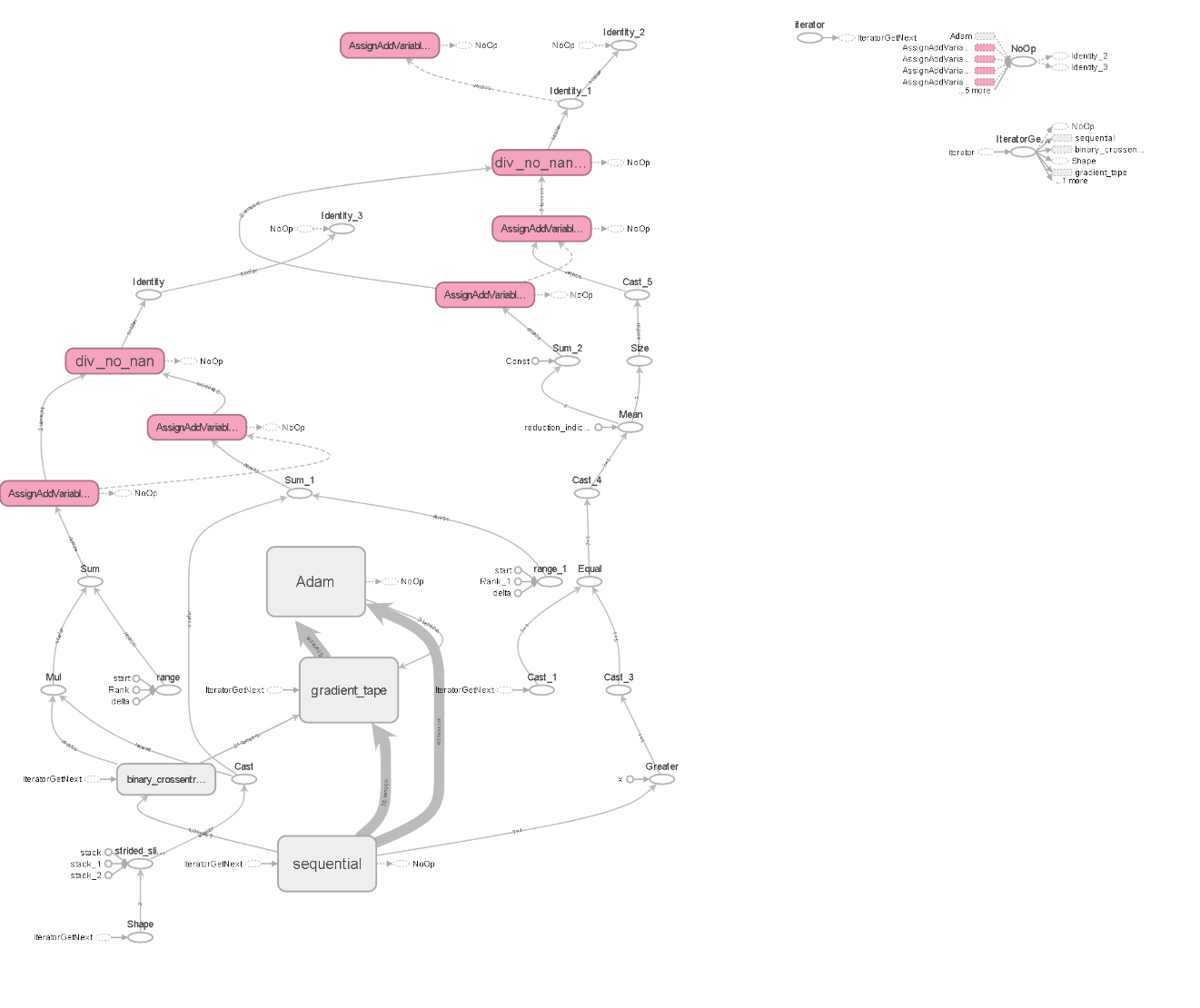
The Python programming language was used to create the model. In addition, the packages Matplotlib, Keras, and NumPy were utilized for system implementation. Keras provides built-in functions such as activation functions, optimizers, layers, etc. TensorFlow was also used as the system's back-end.

Furthermore, Tensor board was used for model tracking.



****

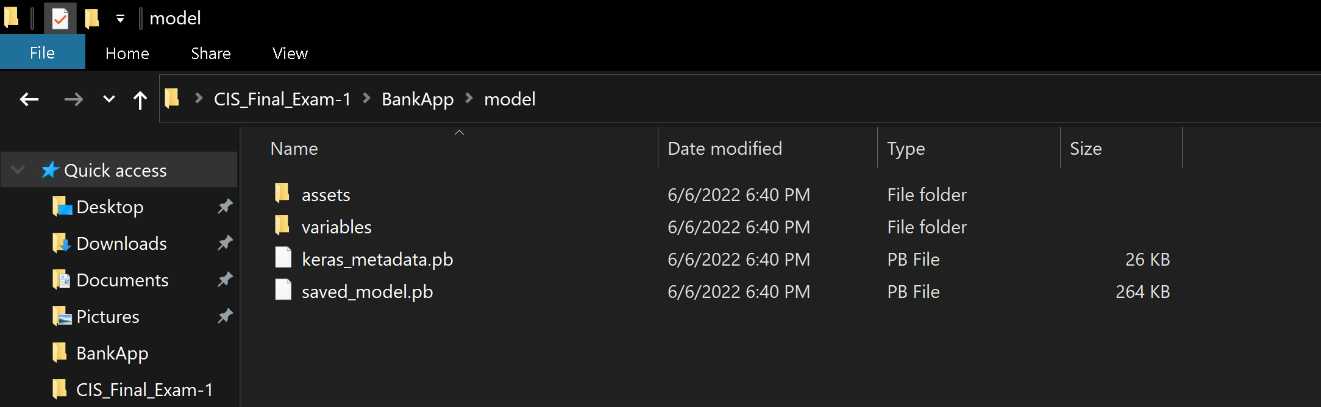
**Conceptual Graph Generated from Tensor Board**

****

**EXPORTING MODEL**

You must first export your trained models in TensorFlow Saved\_Model format before deploying them to AI Platform Prediction and using them to deliver predictions. TensorFlow's recommended format for exporting models is a Saved\_Model, and it is required for deploying trained TensorFlow models on AI Platform Prediction. When you export your trained model as a Saved\_Model, you save your training graph, including its assets, variables, and metadata, in a format that AI Platform Prediction can ingest and restore for predictions.





**CREATING HTTP & FLASK BASED WEB APPLICATION**

In order to create a Gender Detection web application, we have created two different files:

* Flask for the back-end engine - app.py
* HTML for the front-end - /templates/index.html & /output.html

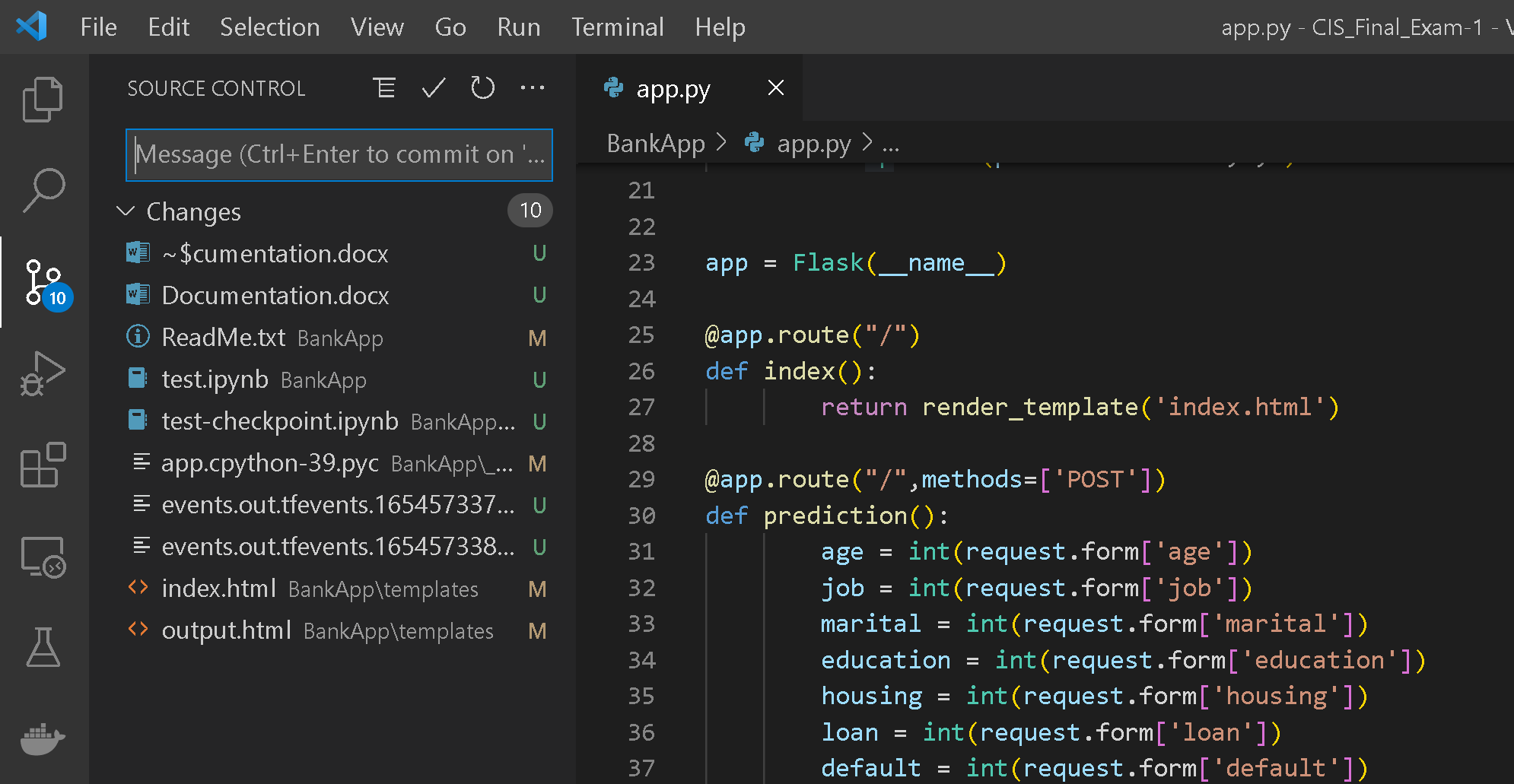
The front-end HTML acts as a medium to interact with and accept input from the user, who then receives predictions from the model. First, the POST request is received from the HTML. Then, when a request is received, the ML model is loaded, and the input file is pre-processed according to the steps described in the Flask back-end engine. Finally, the model generates the prediction, which is subsequently returned to the user via the *render\_template(output.html)* function.

**GITHUB & SCM**

Source code management (SCM) is a technique for tracking changes to a source code repository. SCM maintains a running history of modifications to a code base and aids in dispute resolution when integrating updates from various contributors.

Visual Studio Code can be a valuable tool if you're working with a remote repository. You only need to add your remote repository to VS to be able to control the changes. Another option is to execute git commands from the terminal. Both of them can be seen in the snapshot below.



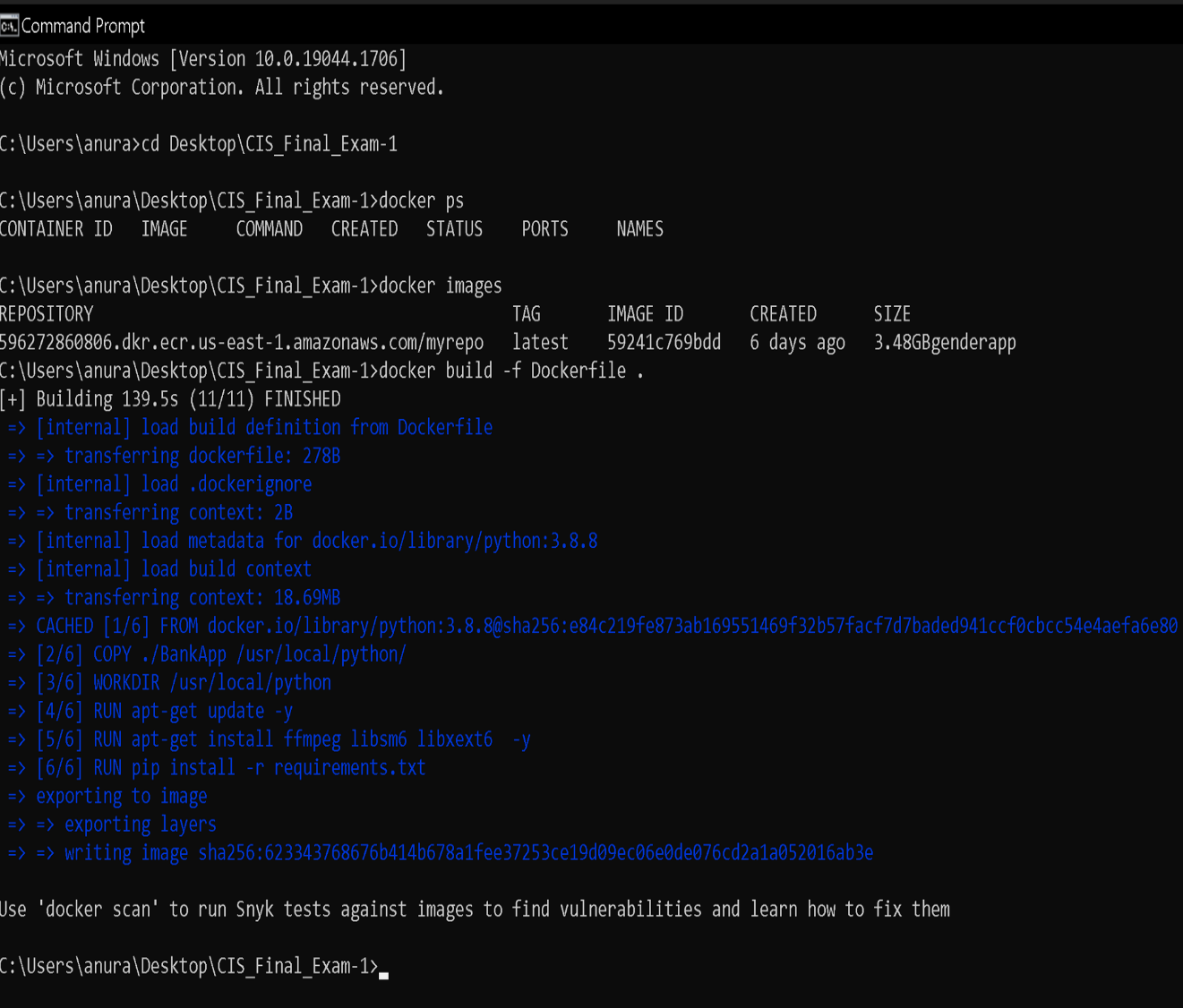


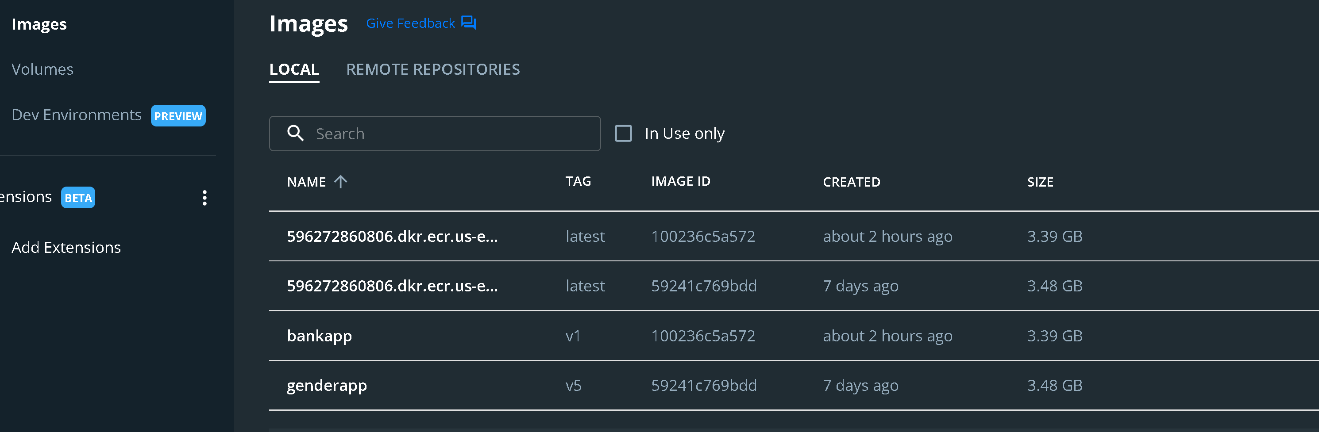
**CREATING A CUSTOM IMAGE FROM DOCKER**

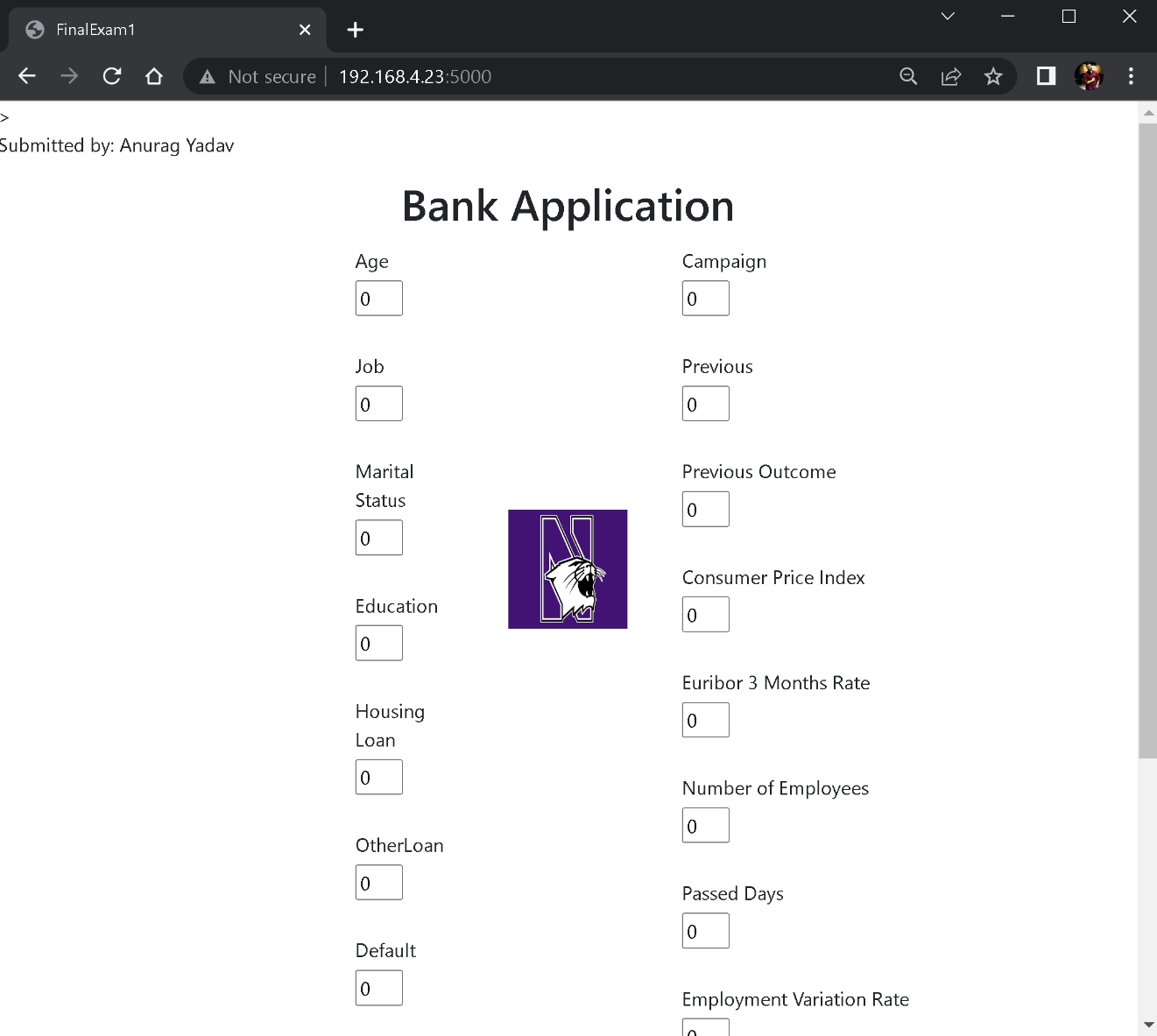
When we want to automate and operate a custom application, the easiest option to deploy your containerized application is to create your own docker image with the desired configuration. This custom configuration may be passed using the Dockerfile, and a container can then be created using this image.

Once the image is ready, it can be used to deploy containers in Docker. In order to test our application on Docker, we have created a container using *docker run -d -p 5000:5000 --name BankApplication <IMAGE\_NAME: VERSION>* command.

The application can be seen running at *localhost:5000* in the last image.

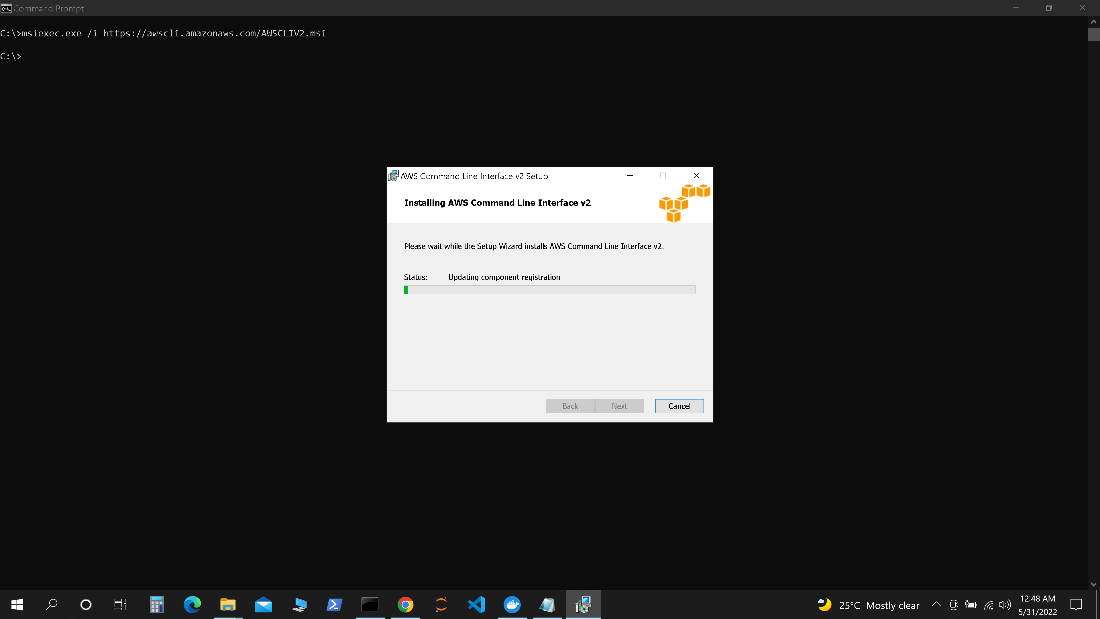


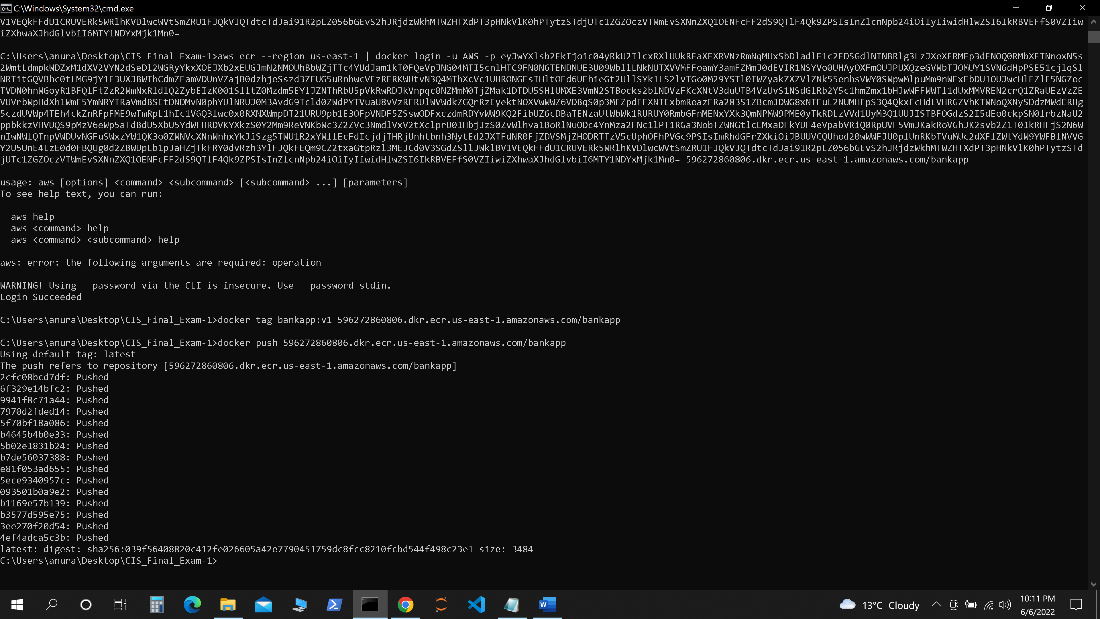




**PUSHING LOCAL DOCKER IMAGE TO AWS ELASTIC CONTAINER REGISTRY ECR**

To push a local docker image to AWS, we must first configure AWS CLI for the first time. This can be performed by using the instructions listed in the [ReadMe](https://github.com/anuragyadav16/CIS_Final_Exam-1/blob/main/BankApp/ReadMe.txt) file. Once the CLI is ready, we must use the terminal to create a repository and push the local docker image to AWS ECR.

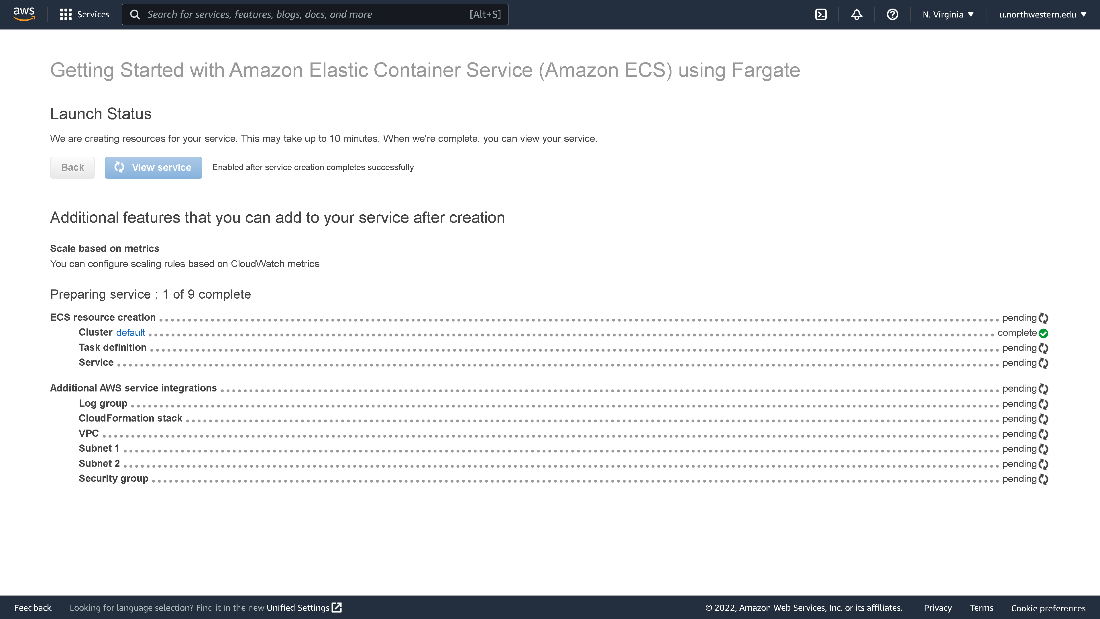


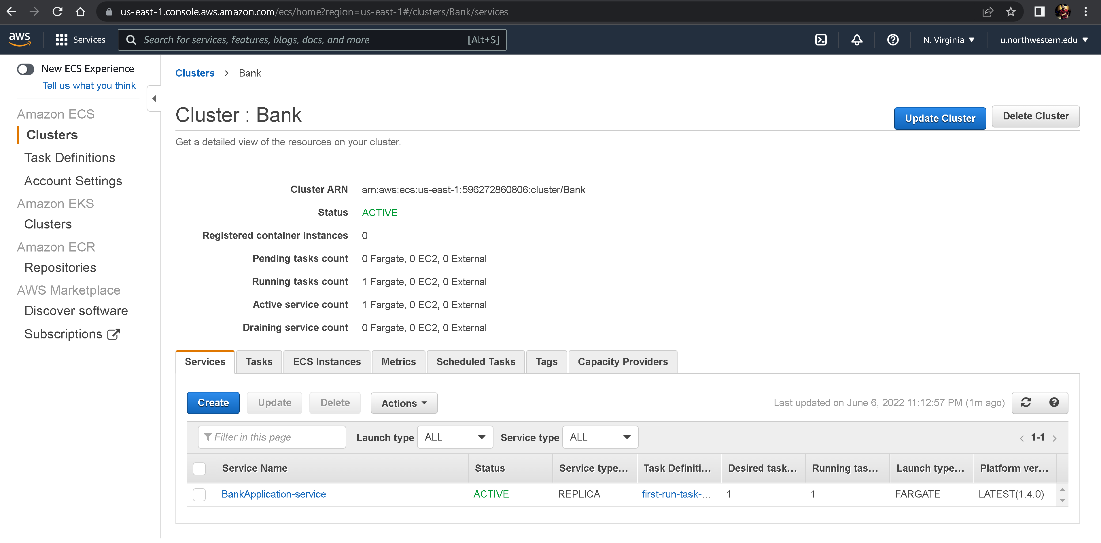


**DEPLOYING ELASTIC CONTAINER SERVICE ECS USING AWS FARGATE**

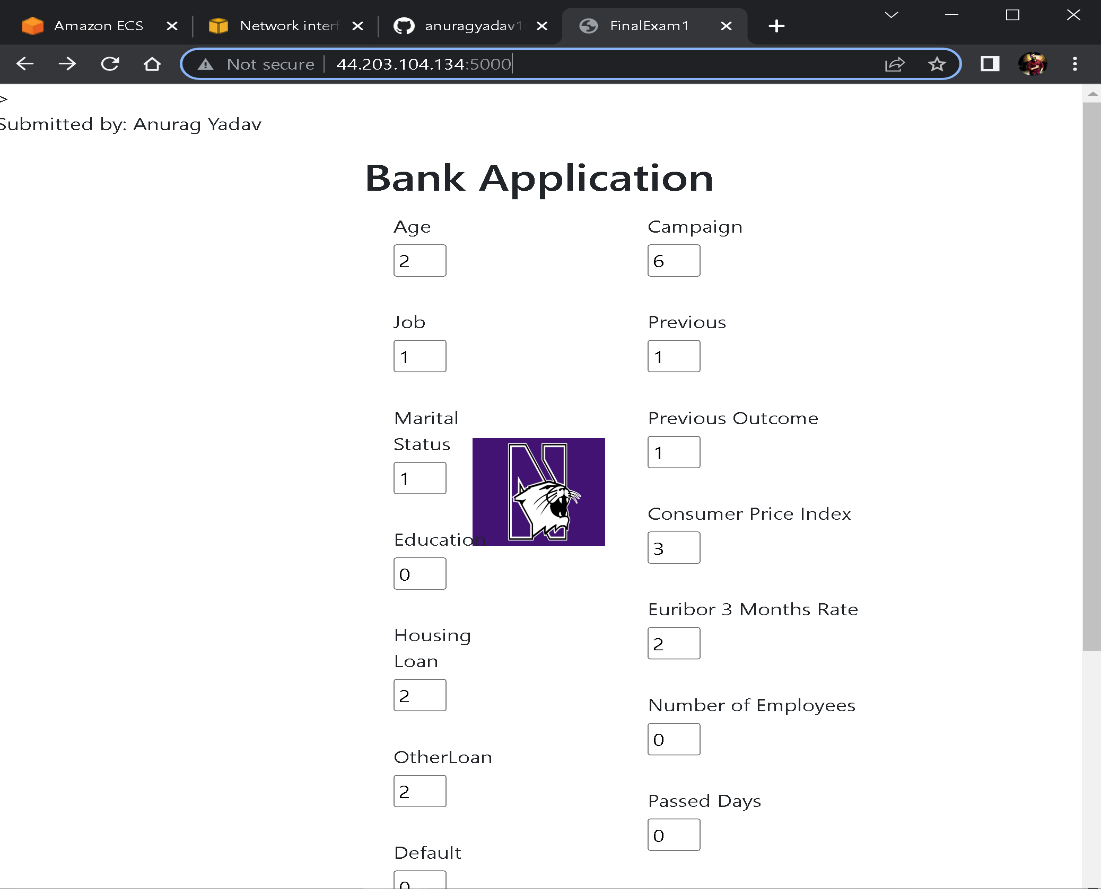
AWS FARGATE is a technology that you can use with Amazon ECS to run containers without having to manage servers or clusters of Amazon EC2 instances. With AWS FARGATE, you no longer have to provision, configure, or scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters or optimize cluster packing.

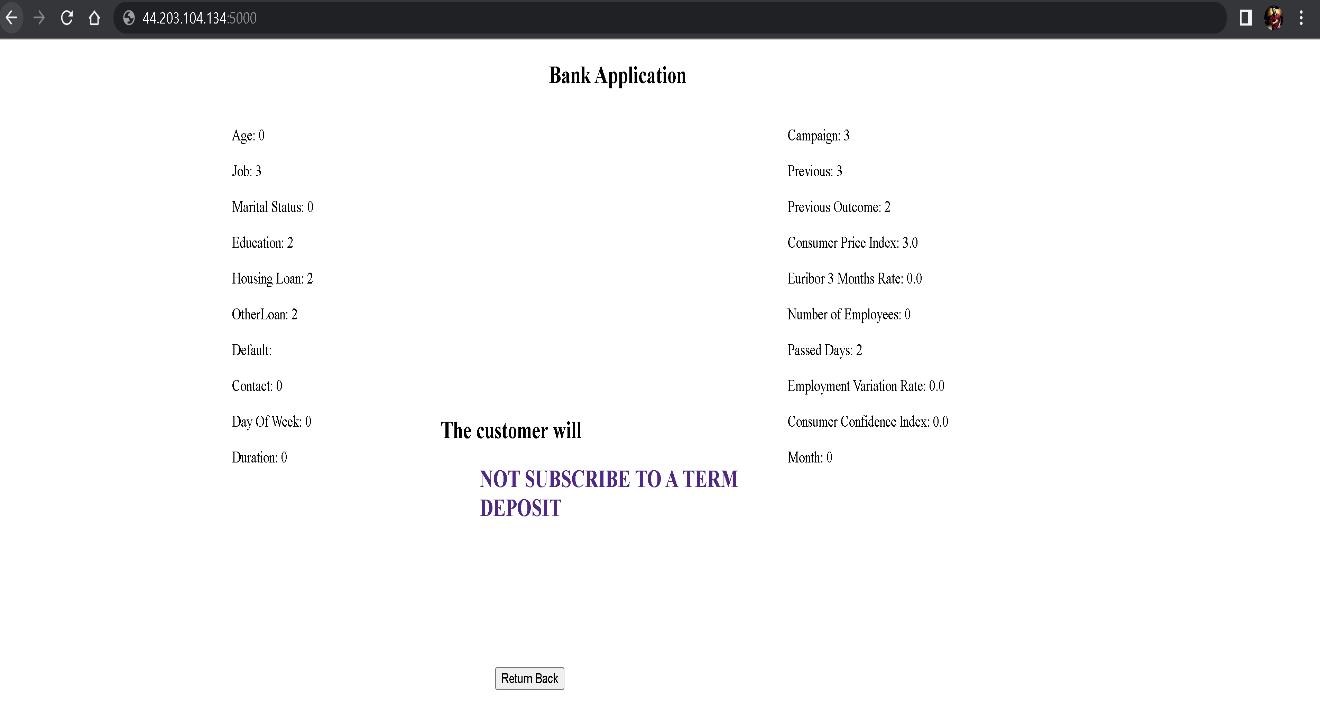
When running your tasks and services with the FARGATE launch type, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each FARGATE task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.

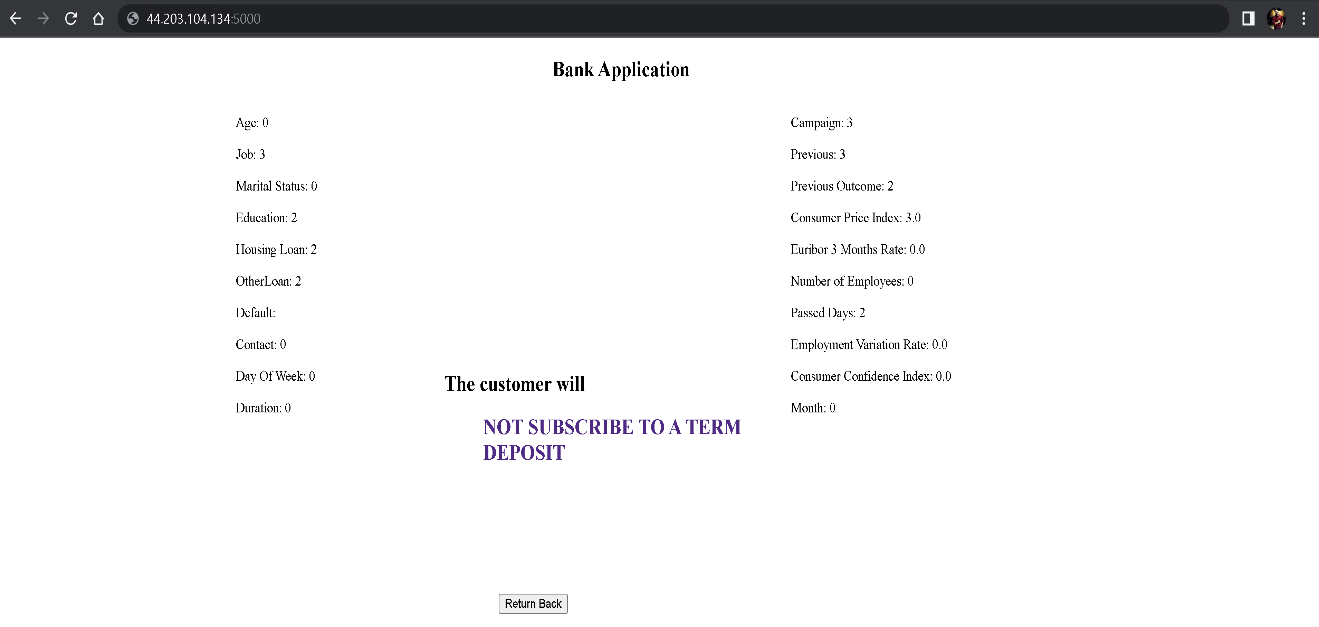


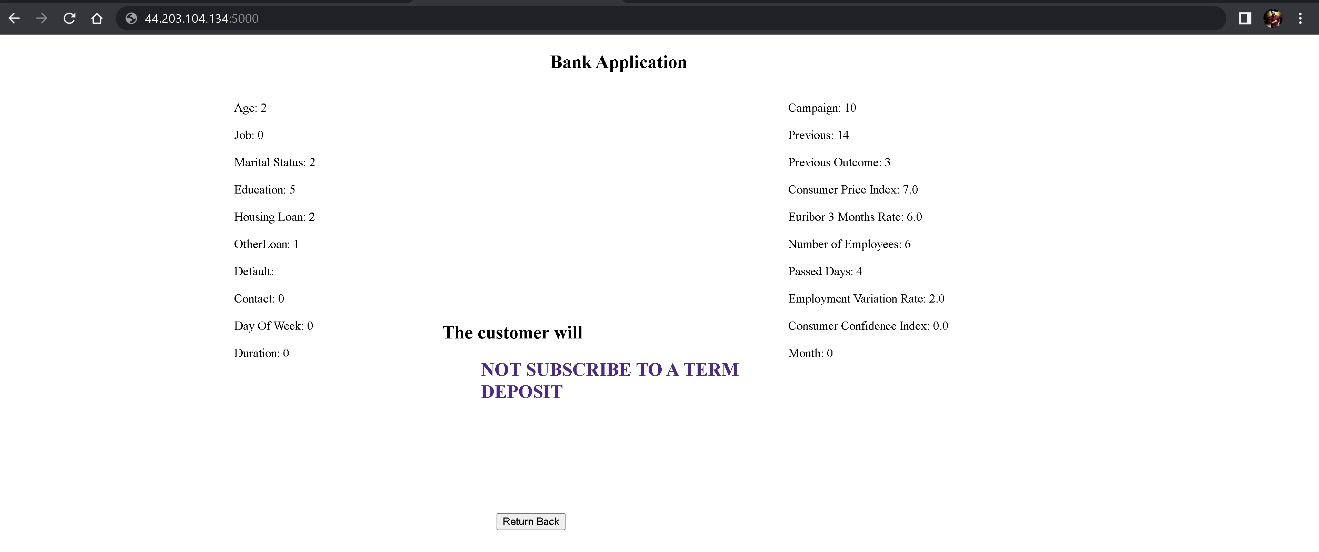


**RESULTS:**







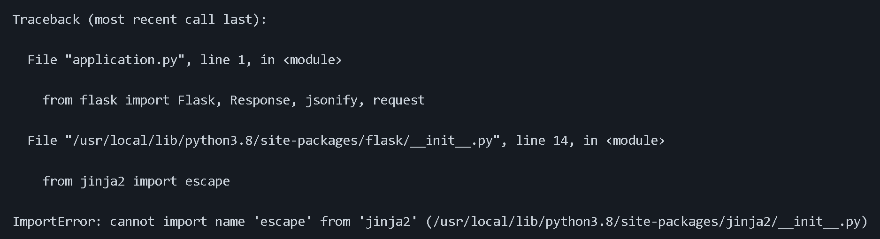


**CHALLENGES / ERRORS ENCOUNTERED:**

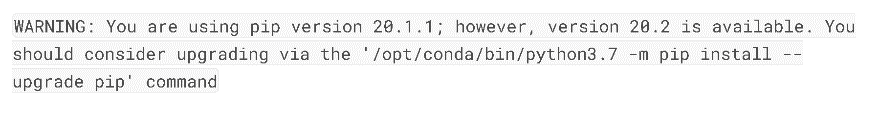
1. Invalid public key for CUDA apt repository

****

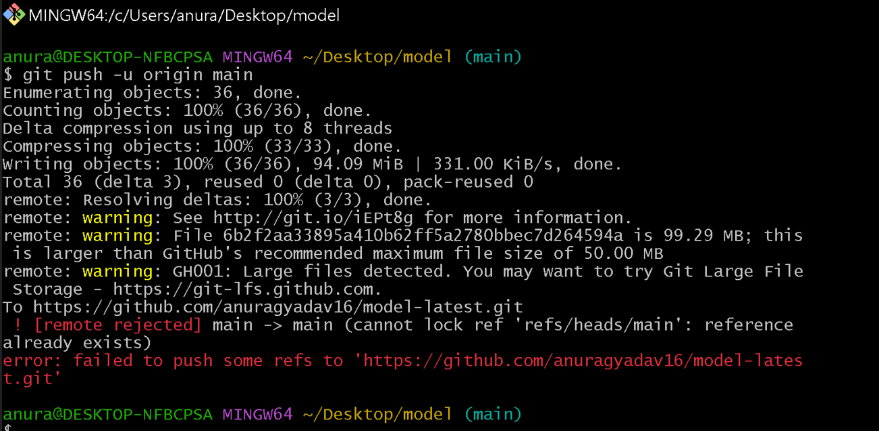
1. Flask failing to startup due to Jinja2 breaking change

****

1. Pip version error

****

1. Git size limit error

****