CIS 325 Final Exam I
Submitted by: Abhishek Mote

Bank Telemarketing Classification Problem using Deep Learning

Introduction:

We are given the data of direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (target variable y). This case study is inspired by this research paper where the researchers have used a very similar dataset as the one we will be using throughout this case study for determining the success of Bank Telemarketing. The result shows the highest accuracy of 89.27%.

Methodology:

I have utilized a ANN (Artificial Neural Network) architecture in my telemarketing model. ANN is generally used for image analysis, segmentation, classification, medical image analysis, photo and video recognition, etc. In this project, **I used TensorFlow with Keras. I** first applied feature engineering on data and processing technique to transform the raw data into an efficient and useful format. Later, the ANN architecture was applied. Here, it has been decomposed into two parts:

- Feature Extraction
- Classification

The Convolution and the pooling layers perform the feature extraction of images that extract information from the input for decision making. Finally, a fully connected layer serves as the classification part.

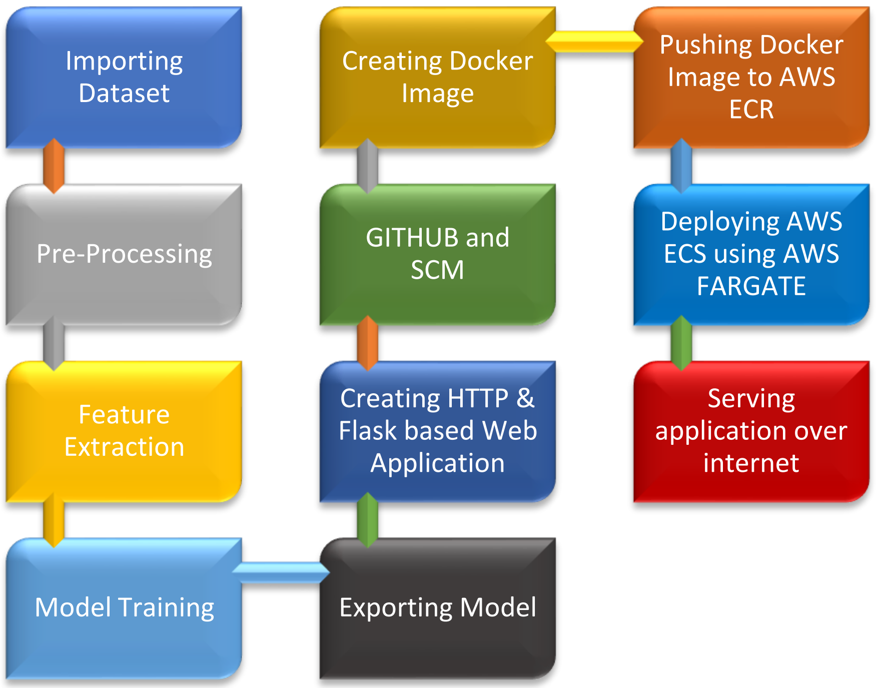
Dataset:

The dataset was Bank Telemarketing dataset with target label "y". The data is related with direct marketing campaigns of a Portuguese 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. https://archive.ics.uci.edu/ml/datasets/bank+marketing

Pre-Processing:

I checked for the null values in the data and found none. But interestingly the data was highly imbalance as there was more than 70% of the data which has "No" as a value for our response variable. Then I used balancer to make sure all the values in our response variable are equally balanced. I applied feature engineering to convert the categorical variable like age, month, year etc. and applied MinMaxScaler to make sure the data points are scaled for model.

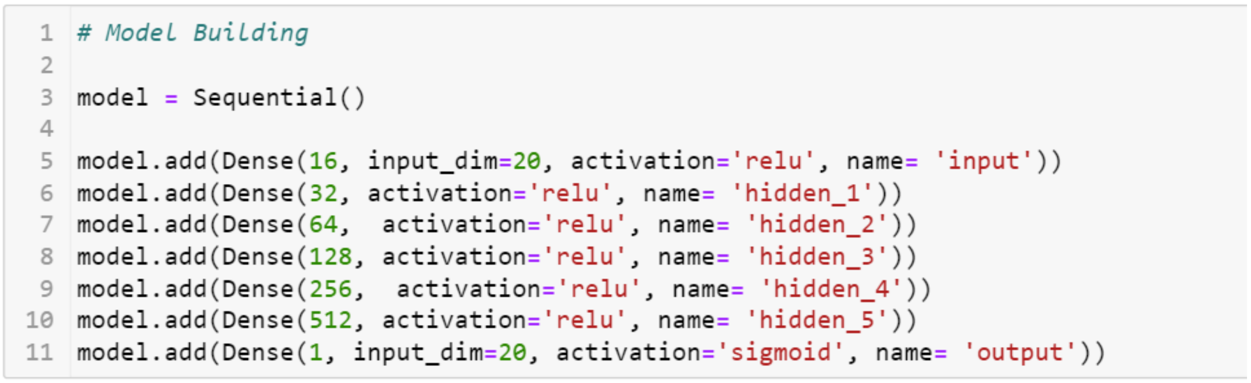
Project Workflow:


Importing Dataset
Pre-Processing
Feature Extraction
Model Training
Exporting Model
Creating HTTP & Flask based Web Application
GITHUB and SCM
Creating Docker Image
Pushing Docker Image to AWS ECR
Deploying AWS ECS using AWS FARGATE
Serving application over internet

TRAINING MODEL:

I used Python for creating this ML model. Packages used:

- Matplotlib,
- Keras
- 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.

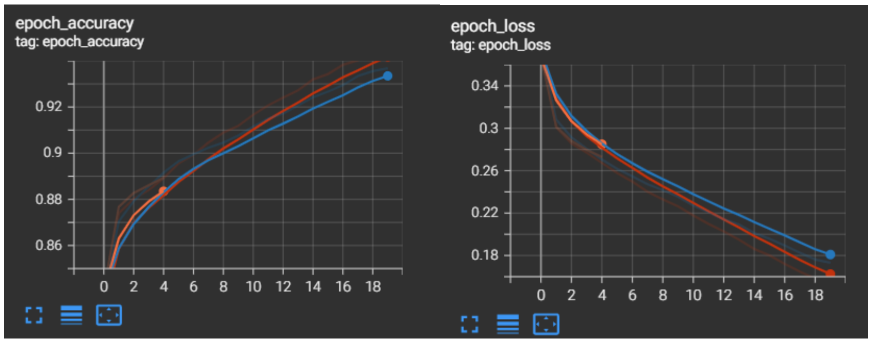


```
# Model Building

model = Sequential()

model.add(Dense(16, input_dim=20, activation='relu', name= 'input'))
model.add(Dense(32, activation='relu', name= 'hidden_1'))
model.add(Dense(64,  activation='relu', name= 'hidden_2'))
model.add(Dense(128, activation='relu', name= 'hidden_3'))
model.add(Dense(256,  activation='relu', name= 'hidden_4'))
model.add(Dense(512, activation='relu', name= 'hidden_5'))
model.add(Dense(1, input_dim=20, activation='sigmoid', name= 'output'))
```

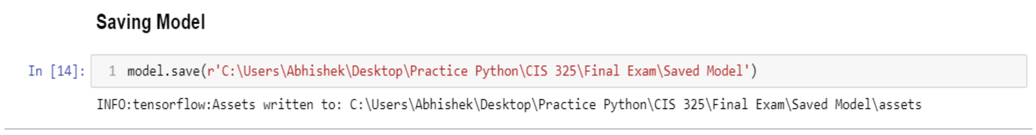
MODEL ACCURACY REPRESENTATION:



The above graph shows the accuracy of multiple models runs and how the accuracy has increased after each change in features, nodes, and variables.

EXPORTING MODEL:

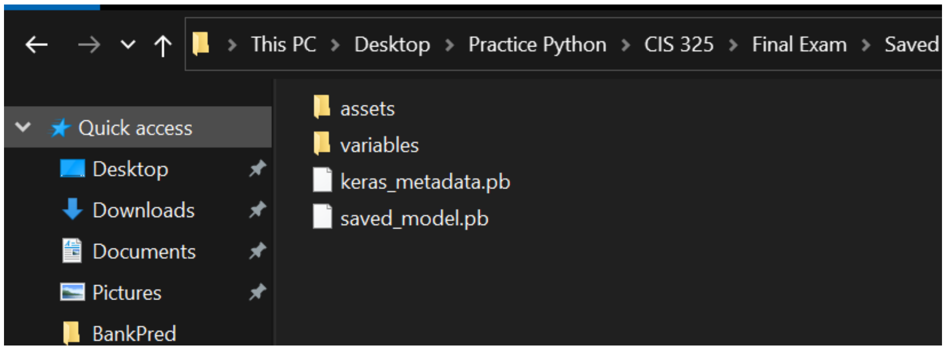
You must first export your trained models in TensorFlow SavedModel format before deploying them to AI Platform Prediction and using them to deliver predictions. TensorFlow's recommended format for exporting models is a SavedModel, and it is required for deploying trained TensorFlow models on AI Platform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



Saving Model

```
In [14]: model.save(r'C:\Users\Abhishek\Desktop\Practice Python\CIS 325\Final Exam\Saved Model')

INFO:tensorflow:Assets written to: C:\Users\Abhishek\Desktop\Practice Python\CIS 325\Final Exam\Saved Model\assets
```


This PC > Desktop > Practice Python > CIS 325 > Final Exam > Saved

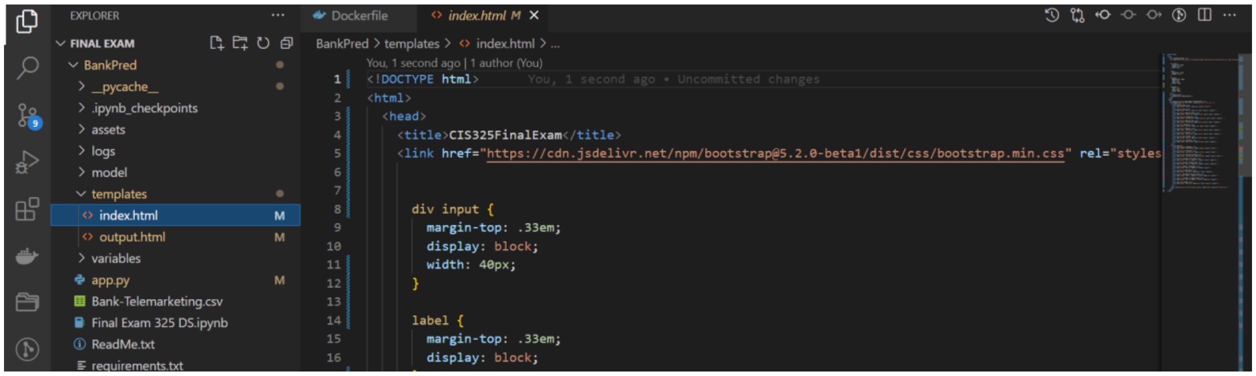
```
assets
variables
keras_metadata.pb
saved_model.pb
```

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

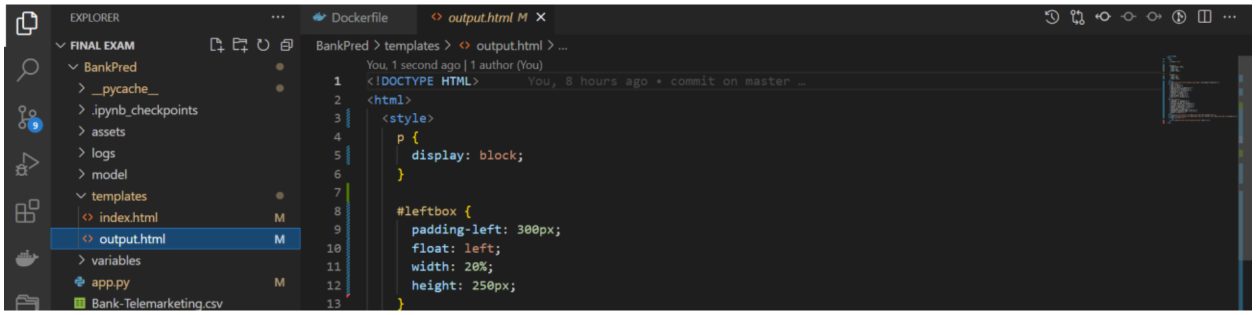
The front-end HTML acts as a medium to interact with and accept input from the user, who then receives predictions from the model. First, the POST request is received from the HTML. Then, when a request is received, the ML model is loaded, and the input file is pre-processed according to the steps described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


BankPred > templates > index.html > ...

```
<!DOCTYPE html>
<html>
  <head>
    <title>CIS325FinalExam</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/css/bootstrap.min.css" rel="styles


      div input {
        margin-top: .33em;
        display: block;
        width: 40px;
      }

      label {
        margin-top: .33em;
        display: block;
```


BankPred > templates > output.html > ...

```
<!DOCTYPE HTML>
<html>
  <style>
    p {
      display: block;
    }

    #leftbox {
      padding-left: 300px;
      float: left;
      width: 20%;
      height: 250px;
    }
```

Snippets of the flask backend engine and HTML front end.

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.

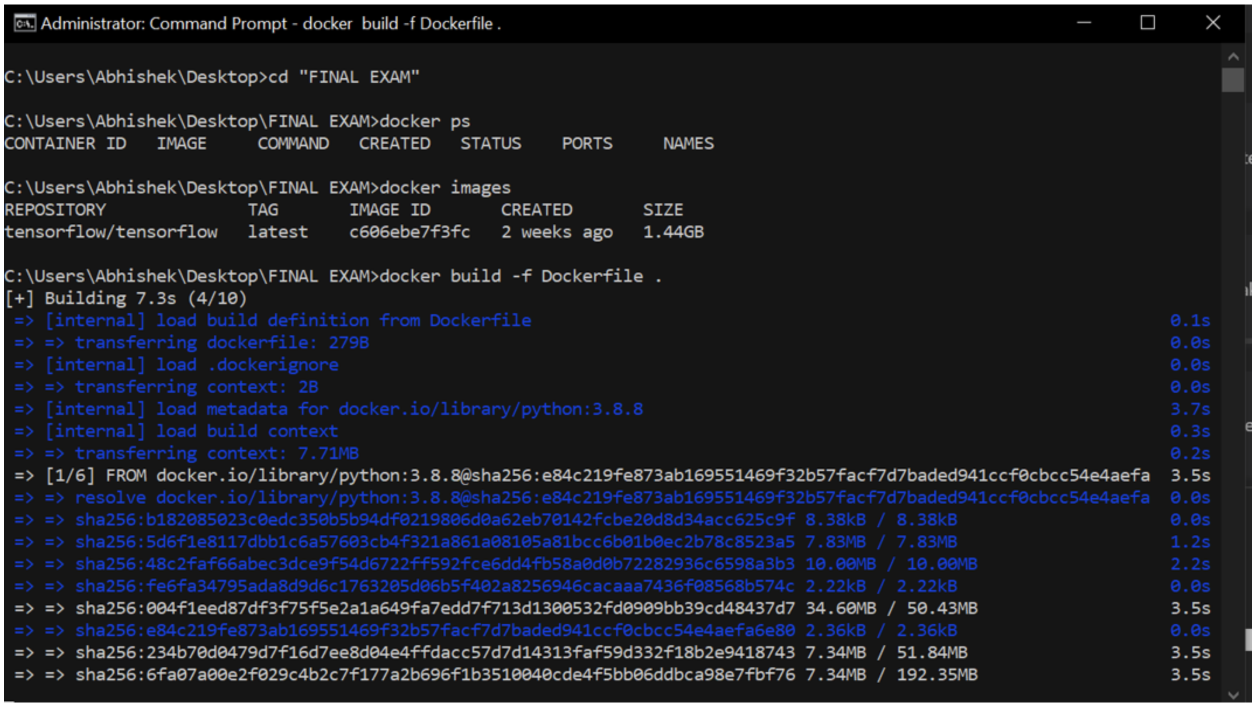
Visual Studio Code can be a valuable tool if you're working with a remote repository. You only need to add your remote repository to VS to be able to control the changes. Another option is to execute git commands from the terminal. Both of them can be seen in the snapshot below.

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 TERM PLAN <IMAGE_NAME: VERSION>* command.

The application can be seen running at *localhost:5000*.

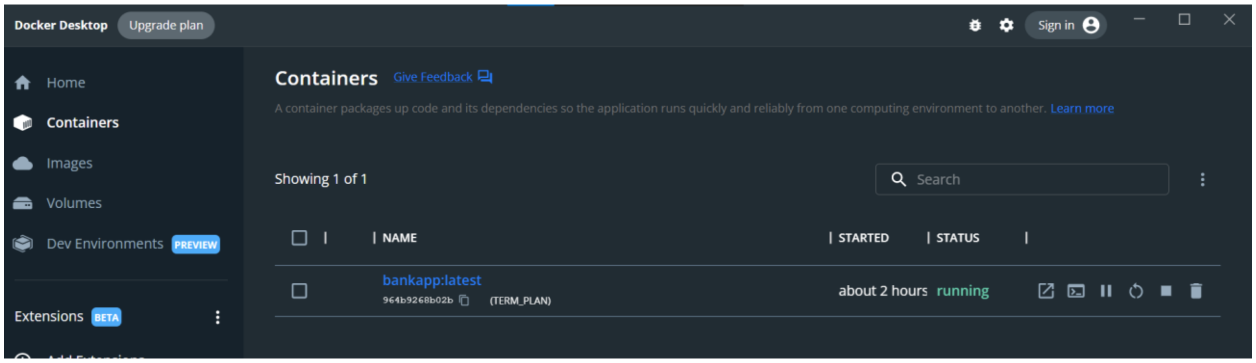

Administrator: Command Prompt - docker build -f Dockerfile .

```
C:\Users\Abhishek\Desktop>cd "FINAL EXAM"

C:\Users\Abhishek\Desktop\FINAL EXAM>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\Abhishek\Desktop\FINAL EXAM>docker images
REPOSITORY              TAG       IMAGE ID       CREATED       SIZE
tensorflow/tensorflow   latest    c606ebe7f3fc   2 weeks ago   1.44GB

C:\Users\Abhishek\Desktop\FINAL EXAM>docker build -f Dockerfile .
[+] Building 7.3s (4/10)
 => [internal] load build definition from Dockerfile                                                               0.1s
 => => transferring dockerfile: 279B                                                                               0.0s
 => [internal] load .dockerignore                                                                                  0.0s
 => => transferring context: 2B                                                                                    0.0s
 => [internal] load metadata for docker.io/library/python:3.8.8                                                    3.7s
 => [internal] load build context                                                                                  0.3s
 => => transferring context: 7.71MB                                                                                0.2s
 => [1/6] FROM docker.io/library/python:3.8.8@sha256:e84c219fe873ab169551469f32b57facf7d7baded941ccf0cbcc54e4aefa  3.5s
 => => resolve docker.io/library/python:3.8.8@sha256:e84c219fe873ab169551469f32b57facf7d7baded941ccf0cbcc54e4aefa  0.0s
 => => sha256:b182085023c0edc350b5b94df0219806d0a62eb70142fcbe20d8d34acc625c9f 8.38kB / 8.38kB                     0.0s
 => => sha256:5d6f1e8117dbb1c6a57603cb4f321a861a08105a81bcc6b01b0ec2b78c8523a5 7.83MB / 7.83MB                     1.2s
 => => sha256:48c2faf66abec3dce9f54d6722ff592fce6dd4fb58a0d0b72282936c6598a3b3 10.00MB / 10.00MB                   2.2s
 => => sha256:fe6fa34795ada8d9d6c1763205d06b5f402a8256946cacaaa7436f08568b574c 2.22kB / 2.22kB                     0.0s
 => => sha256:004f1eed87df3f75f5e2a1a649fa7edd7f713d1300532fd0909bb39cd48437d7 34.60MB / 50.43MB                   3.5s
 => => sha256:e84c219fe873ab169551469f32b57facf7d7baded941ccf0cbcc54e4aefa6e80 2.36kB / 2.36kB                     0.0s
 => => sha256:234b70d0479d7f16d7ee8d04e4ffdacc57d7d14313faf59d332f18b2e9418743 7.34MB / 51.84MB                    3.5s
 => => sha256:6fa07a00e2f029c4b2c7f177a2b696f1b3510040cde4f5bb06ddbca98e7fbf76 7.34MB / 192.35MB                   3.5s
```



Containers

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Showing 1 of 1

NAME	STARTED	STATUS
bankapp:latest 964b9268b02b (TERM_PLAN)	about 2 hours	running

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 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.

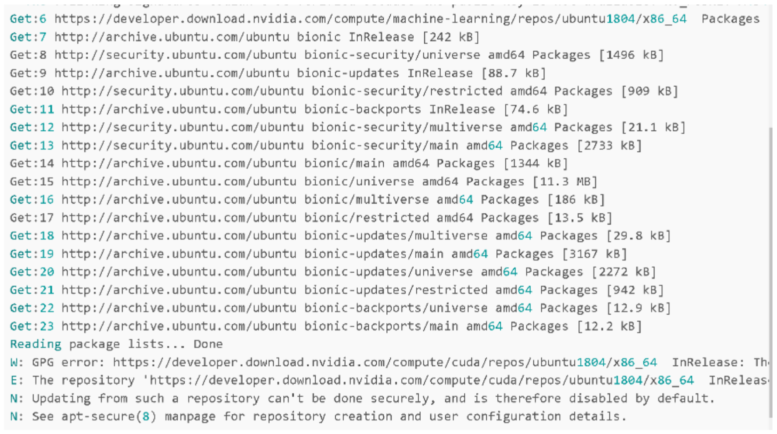
RESULTS:

Access this Application over Internet: http:// 54.87.120.37:5000/

Github Link: https://github.com/ABHISHEK9894/FINAL-EXAM-1.git

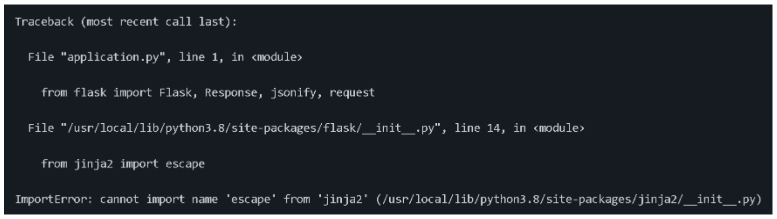
CHALLENGES / ERRORS ENCOUNTERED:

1. Invalid public key for CUDA apt repository



```
Get:6 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  Packages
Get:7 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Get:8 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [1496 kB]
Get:9 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:10 http://security.ubuntu.com/ubuntu bionic-security/restricted amd64 Packages [909 kB]
Get:11 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:12 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [21.1 kB]
Get:13 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [2733 kB]
Get:14 http://archive.ubuntu.com/ubuntu bionic/main amd64 Packages [1344 kB]
Get:15 http://archive.ubuntu.com/ubuntu bionic/universe amd64 Packages [11.3 MB]
Get:16 http://archive.ubuntu.com/ubuntu bionic/multiverse amd64 Packages [186 kB]
Get:17 http://archive.ubuntu.com/ubuntu bionic/restricted amd64 Packages [13.5 kB]
Get:18 http://archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 Packages [29.8 kB]
Get:19 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [3167 kB]
Get:20 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [2272 kB]
Get:21 http://archive.ubuntu.com/ubuntu bionic-updates/restricted amd64 Packages [942 kB]
Get:22 http://archive.ubuntu.com/ubuntu bionic-backports/universe amd64 Packages [12.9 kB]
Get:23 http://archive.ubuntu.com/ubuntu bionic-backports/main amd64 Packages [12.2 kB]
Reading package lists... Done
W: GPG error: https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  InRelease: Th
E: The repository 'https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  InReleas
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
```

2. Flask failing to startup due to Jinja2 breaking change



```
Traceback (most recent call last):

  File "application.py", line 1, in <module>

    from flask import Flask, Response, jsonify, request

  File "/usr/local/lib/python3.8/site-packages/flask/__init__.py", line 14, in <module>

    from jinja2 import escape

ImportError: cannot import name 'escape' from 'jinja2' (/usr/local/lib/python3.8/site-packages/jinja2/__init__.py)
```

3. Pip version error

```
WARNING: You are using pip version 20.1.1; however, version 20.2 is available. You
should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --
upgrade pip' command
```