

IST 615

Bryce Anthony

Shiva Kumar Godipally

Kumar Reddy

Final Project Paper - Deploying Artificial
Intelligence Solutions

Due: Friday May 3rd

Submitted: Friday May 3rd

Document Length: 25

Project Background

The primary driving force behind this project was the desire to gain experience with deploying machine learning models. Our frustration stemmed from developing AI solutions that would only be available to a narrow audience of tech savvy individuals. This led us to explore the deployment of AI solutions using cloud infrastructure. In addition to increasing the accessibility of locally developed AI solutions, mastering the deployment process promises to enhance our ability to craft superior solutions and collaborate effectively within teams in our careers. By gaining insights into the decision-making aspects that impact the end-users, we can refine our code and optimize its utility for cloud infrastructure. Effective deployment not only ensures the scalability and adaptability of machine learning solutions but also enables them to deliver actionable insights across various sectors. Ultimately, deploying machine learning models is a critical step in driving tangible outcomes, boosting productivity, and instigating changes across industries via artificial intelligence solutions. In this project we deployed three Machine Learning models that we had previously developed. An exam score predictor using traditional ML to generate score predictions for students using a variety of demographic information, a deep learning model delivering AI powered predictions for regular season NBA matches, and an AI powered web application that brings the power of computer vision and large language models to calorie tracking and nutritional guidance.

Project #1 Exam Score Predictor

Project Description:

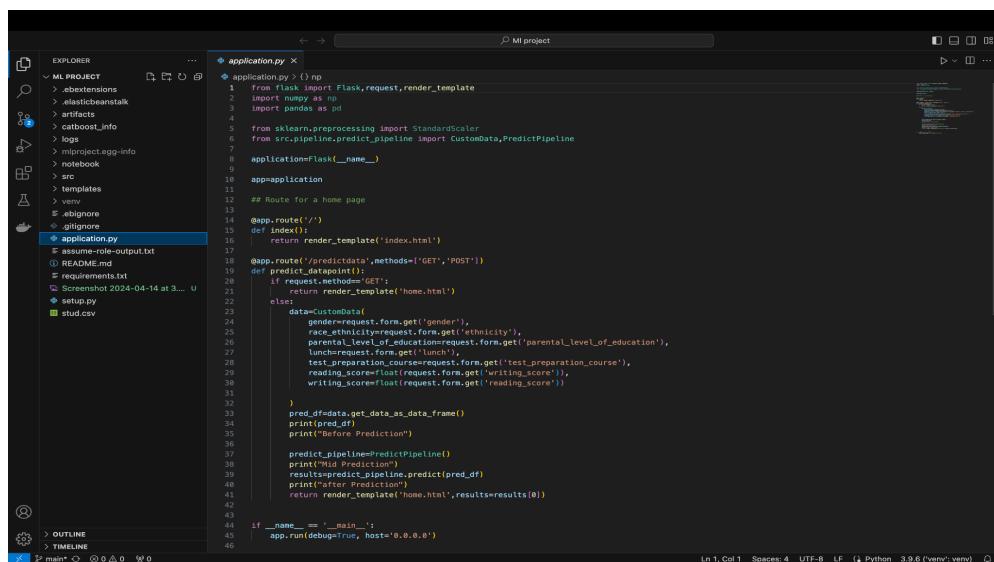
The Student Exam Performance Predictor is a machine learning-based web application designed to forecast student exam outcomes. Developed using Python and Flask, the project focuses on creating a robust and accurate predictive tool that considers multiple factors influencing student performance. This project showcases a commitment to transparent and collaborative development, being fully realized in VS Code and integrated with GitHub. The project serves as a valuable tool for educators and administrators, helping them identify areas for intervention and improvement. It also highlights the potential for technology to enhance educational outcomes through data-driven insights and predictive analytics.

Implementation Overview:

The implementation involved building a custom Python environment from scratch, providing a detailed understanding of the system configurations and dependencies involved in a machine learning workflow. Key components of the project include custom exception handlers and a detailed logging system, which enhance the application's robustness and debugging capabilities.

The web interface, developed using Flask, offers an intuitive user experience and integrates seamlessly with the machine learning model. The model has been trained on diverse datasets to ensure reliability and accuracy in predicting student exam performance. The project was successfully deployed on a host server, demonstrating its viability for real-world applications in educational settings.

Local Development Environment



A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows a file tree with a folder named 'ML PROJECT' containing files like '.ebextensions', '.elasticbeanstalk', 'artifacts', 'catboost_info', 'data', 'image', 'microsoft.egg-info', 'notebook', 'src', 'templates', 'venv', '.ebignore', '.gitignore', and 'application.py'. The 'application.py' file is currently selected and open in the main editor area. The code in the editor is as follows:

```
application.py X
application.py (1 np
1   from flask import Flask,request,render_template
2   import numpy as np
3   import pandas as pd
4
5   from sklearn.preprocessing import StandardScaler
6   from src.pipeline.predict_pipeline import CustomData,PredictPipeline
7
8   application=flask(__name__)
9
10  app=application
11
12  ## Route for a home page
13
14  @app.route('/')
15  def index():
16      return render_template('index.html')
17
18  @app.route('/predictdata',methods=['GET','POST'])
19  def predict_datapoint():
20      if request.method=='GET':
21          return render_template('home.html')
22      else:
23          data=CustomData()
24          gender=request.form.get('gender'),
25          race_ethnicity=request.form.get('ethnicity'),
26          parental_level_of_education=request.form.get('parental_level_of_education'),
27          lunch=request.form.get('lunch'),
28          test_preparation_course=request.form.get('test_preparation_course'),
29          reading_score=float(request.form.get('writing_score')),
30          writing_score=float(request.form.get('reading_score'))
31
32      )
33      pred_dffdata.get_data_as_data_frame()
34      print(pred_dff)
35      print("Before Prediction")
36
37      predict_pipeline=PredictPipeline()
38      print("After Prediction")
39      results=predict_pipeline.predict(pred_dff)
40      print("After Prediction")
41      return render_template('home.html',results=results[0])
42
43
44  if __name__ == '__main__':
45      app.run(debug=True, host='0.0.0.0')
```

The application was developed using Python, with Flask as the web framework, to create an intuitive interface for predicting student exam performance. We implemented key features such as data handling, custom prediction pipelines, and template rendering within the "application.py" file. The local development environment provided a sandbox for us to experiment with different approaches.

By structuring the project effectively and focusing on the essential components during this phase, we laid a solid foundation for subsequent deployment and scaling, ensuring the application met both functional and user experience goals.

EC2 Instance Creation

Instance summary for i-0f189e5363dc5455e (Studentapp-env)														
Instance ID	Public IPv4 address	Private IPv4 addresses												
i-0f189e5363dc5455e (Studentapp-env)	-	172.31.45.193												
IPv6 address	Instance state	Public IPv4 DNS												
-	Running	-												
Hostname type	Private IP DNS name (IPv4 only)	Elastic IP addresses												
IP name: ip-172-31-45-193.us-east-2.compute.internal	ip-172-31-45-193.us-east-2.compute.internal	-												
Answer private resource DNS name	Instance type	AWS Compute Optimizer finding												
-	t3.micro	Optimized View detail												
Auto-assigned IP address	VPC ID	Auto Scaling Group name												
-	vpc-off53e77836f932c0	awseb-e-6ptkk9uykg-stack-AWSEBAutoScalingGroup-nmDHkeOJGSSy												
IAM Role	Subnet ID													
MyElasticBeanstalkRole	subnet-08c86cece938967ee													
IMDSv2 Required														
Details Status and alarms Monitoring Security Networking Storage Tags														
Instance details <table border="1"> <tr> <td>Platform</td> <td>AMI ID</td> <td>Monitoring</td> </tr> <tr> <td>Linux/UNIX (Inferred)</td> <td>ami-0702e2b6d68a6b534</td> <td>disabled</td> </tr> <tr> <td>Platform details</td> <td>AMI name</td> <td>Termination protection</td> </tr> <tr> <td>Linux/UNIX</td> <td>-</td> <td>Disabled</td> </tr> </table>			Platform	AMI ID	Monitoring	Linux/UNIX (Inferred)	ami-0702e2b6d68a6b534	disabled	Platform details	AMI name	Termination protection	Linux/UNIX	-	Disabled
Platform	AMI ID	Monitoring												
Linux/UNIX (Inferred)	ami-0702e2b6d68a6b534	disabled												
Platform details	AMI name	Termination protection												
Linux/UNIX	-	Disabled												

For the deployment of the Student Exam Performance Predictor, we created an EC2 instance within the AWS Management Console. This t3.micro instance provides an optimal balance of computing power and cost-effectiveness for our application. The instance is managed under a specific VPC and subnet, aligning with our network architecture. We used the "MyElasticBeanstalkRole" IAM role to handle permissions, ensuring that the instance could interact securely with other AWS services. The instance state is "running," indicating that our application is currently active and operational.

In this setup, we leveraged AWS Compute Optimizer to ensure that our instance is appropriately configured for performance and cost efficiency. The optimized status suggests that our EC2

environment is well-suited for the workload of the predictive application. This EC2 instance creation was a crucial step in transitioning from local development to a cloud-based deployment, ensuring scalability and reliability for the Student Exam Performance Predictor.

Security Rules Updated

The screenshot shows the AWS EC2 Instances page. The instance details for 'flask-env' are displayed, including its IAM Role ('aws-elasticbeanstalk-ec2-role'), Owner ID ('058264440005'), and Launch time ('Fri May 03 2024 11:03:15 GMT-0400 (Eastern Daylight Time)'). Under the 'Security' tab, the 'Inbound rules' section shows two entries:

Name	Security group rule ID	Port range	Protocol	Source	Security group
-	sgr-0d799b5addc87cc9	22	TCP	0.0.0.0/0	awseb-e-hr
-	sgr-0a1bb60e275221c48	80	TCP	sg-0b24226cd1f5da277	awseb-e-hr

Under the 'Outbound rules' section, there is one entry:

Name	Security group rule ID	Port range	Protocol	Destination	Security group
-	sgr-07a5171fdf35761fd	All	All	0.0.0.0/0	awseb-e-hr

The security rules were updated to allow ssh traffic via port 22 and inbound/outbound http traffic on all ports

The Student Exam Performance Indicator was Deployed

The screenshot shows the deployed application at 'flask-env.eba-yahbfir.us-east-2.elasticbeanstalk.com'. The page title is 'Student Exam Performance Indicator'. Below it, the heading 'Student Exam Performance Prediction' is displayed. A form is present with various input fields: 'Gender' (dropdown), 'Race or Ethnicity' (dropdown), 'Parental Level of Education' (dropdown), 'Lunch Type' (dropdown), 'Test preparation Course' (dropdown), 'Writing Score out of 100' (input field), 'Reading Score out of 100' (input field), and 'Predict your Maths Score' (input field). At the bottom of the page, the text 'THE prediction is 94.5' is displayed.

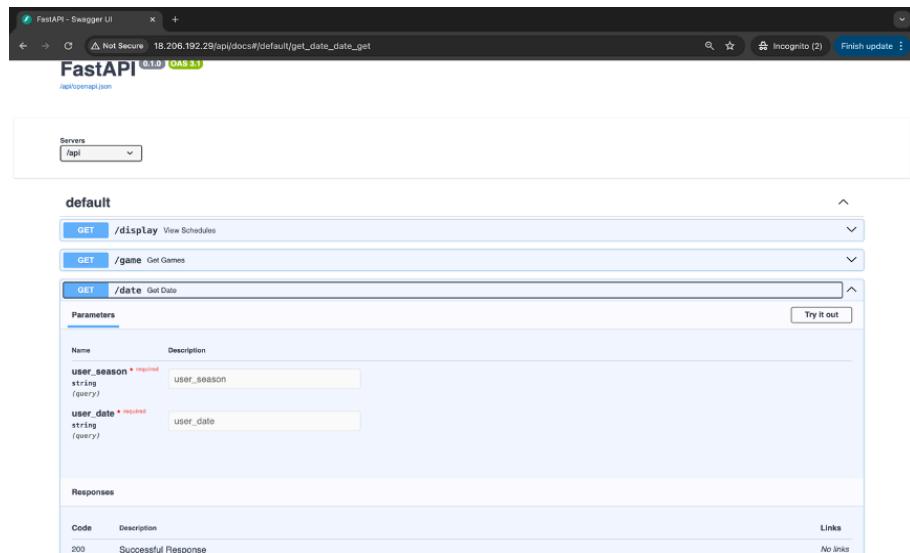
Discussion

AWS Elastic Beanstalk streamlined the deployment process, alleviating much of the heavy lifting. Using the Elastic beanstalk command line interface we were able to create our elastic beanstalk environment and a series of essential resources were initialized for us. This included; the instantiation of an EC2 instance, a load balancer for the EC2 instance, an auto scaling group, an S3 bucket, Amazon CloudWatch alarms, AWS CloudFormation, and security rules that allow the various resources to communicate with each other.

The security group allowed for the passage of HTTP traffic from the load balancer to the EC2 instance. The inclusion of a load balancer served as a critical component, adding an extra layer of security as we weren't directly exposing our instances to the internet. The load balancer security group was established to facilitate inbound traffic on port 80, enabling HTTP traffic from the internet to reach the load balancer securely. The auto-scaling group was integrated to dynamically manage the allocation of resources based on demand fluctuations. A domain name was configured to provide a user-friendly and accessible endpoint for our deployed application, culminating in a robust and scalable deployment architecture. The Amazon S3 bucket served as a central repository for our source code and project directory, facilitating seamless version control and deployment management. Additionally, Amazon CloudWatch alarms provided real-time monitoring capabilities, ensuring the health of our application and AWS CloudFormation orchestrated the connection of these services.

Ultimately this configuration facilitated by AWS Elastic Beanstalk not only streamlined our deployment process but also laid a solid foundation for a robust, scalable, and resilient deployment architecture, empowering us to focus on developing and enhancing our application with confidence.

Project #2 NBA Deep Learning match predictions API



The screenshot shows the FastAPI - Swagger UI interface. At the top, it displays the URL `18.206.192.29/api/docs#/default/get_date_date_get`. Below the header, there's a dropdown menu labeled "Servers" with "api" selected. The main content area is titled "default". It lists three GET endpoints: "/display", "/game", and "/date". The "/date" endpoint is expanded, showing its parameters: "user_season" (required, string, query) and "user_date" (required, string, query). Under "Responses", the 200 response is listed as "Successful Response". There are "Try it out" and "Links" buttons at the bottom right of the expanded endpoint section.

Project description

This Deep Learning model API was crafted with the aim of providing AI-driven insights into NBA game outcomes, accessible to all. Our conviction lies in empowering users with the predictive capabilities of machine learning, enriching the understanding of both dedicated fans and casual observers. With this tool at their disposal, enthusiasts, analysts, pundits, and casual viewers alike can gain deeper insights into individual games and their broader significance within the season. Equipped with such knowledge, they can make more informed predictions, whether for recreational enjoyment or strategic betting purposes. This model was developed locally for the Applied Machine Learning course and, like the other projects in this report, would benefit from being publicly accessible.

Implementation overview

Because there are a variety of use cases for the information being output by the model, we decided that this model would be best deployed as an API. Originally we planned to deploy it via aws lambda since it is serverless, internally handles load balancing and scaling, and only requires users to pay for active computing time, but due to the size of the web Server we decided the best way to deploy the web server would be through the Amazon Elastic Compute Cloud although this meant we would be responsible for the load balancing and scaling features available when using aws lambda.

Local Development environment

The screenshot shows a Jupyter Notebook environment with the following details:

- EXPLORER**: Shows files like `app.py`, `OneHotEncoder.py`, `requirements.txt`, and `NB-A-Pienv`.
- OPEN EDITORS**: Displays the contents of `app.py` and `OneHotEncoder.py`.
- CELLS**: Contains two code cells.
- OUTPUT**: Shows the execution results for both cells.
- TIMELINE**: Shows a history of operations.
- OUTLINE**: Shows the structure of the notebook.
- OPEN XML EXPLORER**: Shows file paths.

Code in `app.py`:

```
#!/usr/bin/env python
# encoding: utf-8
# In[1]:



# In[2]:



# In[3]:



# In[4]:



# In[5]:



# In[6]:



# In[7]:



# In[8]:



# In[9]:



# In[10]:



# In[11]:



# In[12]:



# In[13]:



# In[14]:



# In[15]:



# In[16]:



# In[17]:



# In[18]:



# In[19]:



# In[20]:



# In[21]:



# In[22]:



# In[23]:



# In[24]:



# In[25]:



# In[26]:



# In[27]:



# In[28]:



# In[29]:



# In[30]:



# In[31]:



# In[32]:



# In[33]:



# In[34]:



# In[35]:



# In[36]:



# In[37]:



# In[38]:



# In[39]:



# In[40]:



# In[41]:



# In[42]:
```

Code in `OneHotEncoder.py`:

```
#!/usr/bin/env python
# encoding: utf-8
# In[1]:



# In[2]:



# In[3]:



# In[4]:



# In[5]:



# In[6]:



# In[7]:



# In[8]:



# In[9]:



# In[10]:



# In[11]:



# In[12]:



# In[13]:



# In[14]:



# In[15]:



# In[16]:



# In[17]:



# In[18]:



# In[19]:



# In[20]:



# In[21]:



# In[22]:



# In[23]:



# In[24]:



# In[25]:



# In[26]:



# In[27]:



# In[28]:



# In[29]:



# In[30]:



# In[31]:



# In[32]:



# In[33]:



# In[34]:



# In[35]:



# In[36]:



# In[37]:



# In[38]:



# In[39]:



# In[40]:



# In[41]:



# In[42]:
```

Output of `app.py`:

```
INFO: 127.0.0.1:59225 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:59225 - "GET /openapi.json HTTP/1.1" 200 OK
INFO: 127.0.0.1:59225 - "GET /openapi.yaml HTTP/1.1" 200 OK
INFO: 127.0.0.1:59225 - "GET /health HTTP/1.1" 404 Not Found
[2023-07-10 10:45:11] "C:\Windows\system32\cmd.exe" -k -c shutdown -s -t 0
[2023-07-10 10:45:11] Application shutdown complete.
[2023-07-10 10:45:11] Application shutdown complete.
[2023-07-10 10:45:11] Process SpawnProcess-L
[2023-07-10 10:45:11] INFO: Stopping validate process [12252]
[2023-07-10 10:45:11] INFO: Stopping validate process [12258]
```

Output of `OneHotEncoder.py`:

```
[2023-07-10 10:45:11] "C:\Windows\system32\cmd.exe" -k -c shutdown -s -t 0
[2023-07-10 10:45:11] Application shutdown complete.
[2023-07-10 10:45:11] Application shutdown complete.
[2023-07-10 10:45:11] Process SpawnProcess-L
[2023-07-10 10:45:11] INFO: Stopping validate process [12252]
[2023-07-10 10:45:11] INFO: Stopping validate process [12258]
```

This deep learning model was developed locally and Fast api was used as the interface for the model. The model was created and trained using TensorFlow running on Python and data scraped from the basketball reference website. Although the functionality of the API was handled in python, for it to become a functional api it was important for it to be hosted in the cloud and be able to concurrently handle multiple requests.

EC2 Instance Creation

The screenshot shows the AWS EC2 Instance Details page. The instance is an Ubuntu (Inferred) AMI (ami-04b70fb74e45c3917). It has a Linux/UNIX platform and is using the ami-launch-index 0. The instance is stopped and protected from termination. The AMI location is amazon/ubuntu/images/hvm-ssd-gp3/ubuntu-noble-24.04-amd64-server-20240423. The instance auto-recovery is set to Default. The launch time is Thu May 02 2024 18:48:31 GMT-0400 (Eastern Daylight Time) [about 4 hours]. The lifecycle is normal. The key pair assigned at launch is NB-API-KEY. The instance has no kernel ID or RAM disk ID. The boot mode is left-preferred. The user RDN is guest and the OS hostname is disabled. The owner is S00638558931. The current instance boot mode is legacy-bios. The answer RDN DNS hostname IPv4 is enabled. The placement group is not specified. The placement group ID is r-05arc3957d973b3bd. The partition number is not specified.

We selected an Ubuntu EC2 instance with 8 gb of storage for this project, this was necessary to host our web server as it required around 3 gb of memory in addition to the 3 gb of underlying software necessary to run the web server.

EC2 Key pair creation

The screenshot shows the AWS EC2 Key Pairs page. There are two key pairs listed: NB-API-KEY (rsa, created 2024/05/02 18:47 GMT-4, fingerprint d7:44:75:93:46:e0:93:80:0d:08:9d:8f:7c:05:b9, ID key-0671ff51fc967c9) and aws-eb (rsa, created 2022/09/05 22:19 GMT-4, fingerprint 57:35:0ff:fb:f7:67:b4:8e:27:24:ec:b0:5a:158:b1, ID key-0178246b246ece776).

In order to authorize secure connections to the EC2 instance we needed a RSA Key pair and store it on our local machine. This key pair was then used to create secure and authorized connections to the EC2 instance.

Networking rules updated

The screenshot shows the AWS EC2 Instance Details page for an instance with IP address 18.206.192.29. The 'Security' tab is selected, displaying two tables of security group rules. The first table, 'Inbound rules', has two entries: one for port 80 (HTTP) and another for port 22 (SSH). The second table, 'Outbound rules', has one entry for port All (All). Both rules are associated with the security group 'launch-wizard-1'.

The Security rules of the EC2 instance were updated to allow in and outbound http connections from all ports as well as SSH connections in port 22.

SSH into EC2 Instance

```
NB-A-PI — ubuntu@ip-172-31-61-29: ~ ssh -i NB-API-KEY.pem ubuntu@ec2-18-206-192-29.compute-1.amazonaws.com — 119x43
(base) bryce@Bryces-MBP NB-A-PI % chmod 400 "NB-API-KEY.pem"
(base) bryce@Bryces-MBP NB-A-PI % ssh -i "NB-API-KEY.pem" ubuntu@ec2-18-206-192-29.compute-1.amazonaws.com
The authenticity of host 'ec2-18-206-192-29.compute-1.amazonaws.com (18.206.192.29)' can't be established.
ED25519 key fingerprint is SHA256:JaawXRV+bnGi11pdqL2aKQ9jYms90Bnj6yRNOGJ6fRg.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-18-206-192-29.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-1008-aws x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

System information as of Thu May 2 22:51:02 UTC 2024
System load: 0.12 Processes: 119
Usage of /: 23.2% of 6.71GB Users logged in: 0
Memory usage: 5% IPv4 address for enx0: 172.31.61.29
Swap usage: 0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

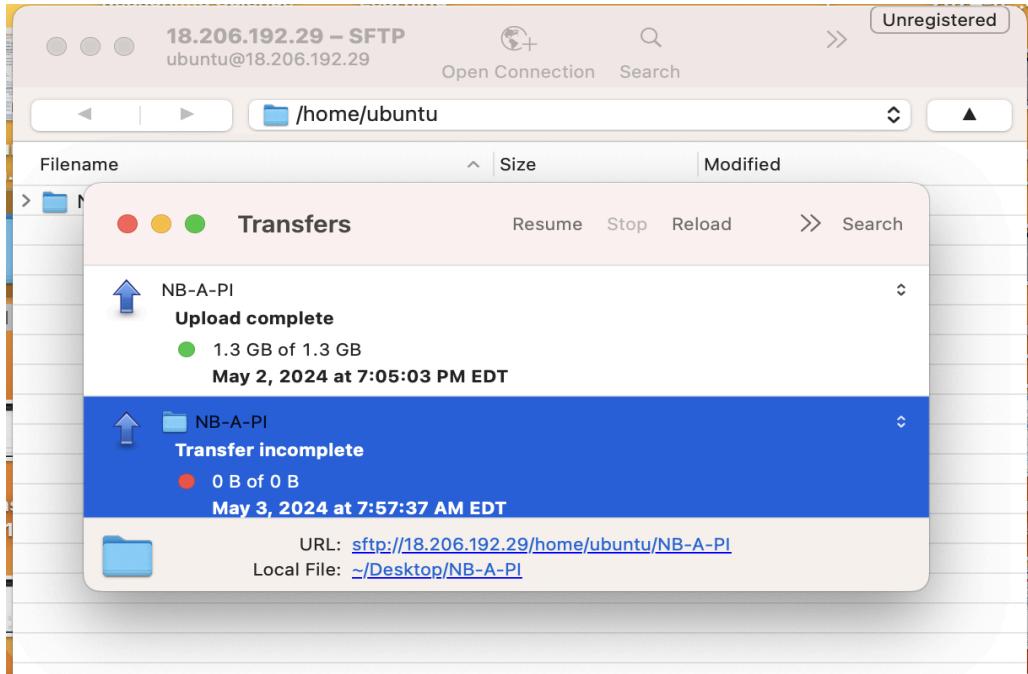
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-61-29:~$
```

Using the rsa key pair we were able to confirm our credentials by connecting to EC2 instances via ssh.

Cyber Duck SFTP



We were then able to use the same rsa key pair with cyberduck's Secure Shell File Transfer Protocol to transfer the local directory of the API to the EC2 instance to deploy it.

Set up EC2 Instance environment for web server

```
Ubuntu@ip-172-31-61-29:~/NB-A-PI$ ssh -i NB-API-KEY.pem ubuntu@ec2-18-206-192-29.compute-1.amazonaws.com
2024-05-02 23:19:03.581883: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-05-02 23:19:04.765888: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
INFO: Started server process [2881]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: 127.0.0.1:35916 - "GET /docs HTTP/1.1" 200 OK
^CINFO: Shutting down.
INFO: Waiting for application shutdown.
INFO: Application shutdown complete.
INFO: Finished server process [2881]

Aborted!
Ubuntu@ip-172-31-61-29:~/NB-A-PI$ pip install gunicorn
Collecting gunicorn
  Downloading gunicorn-22.0.0-py3-none-any.whl.metadata (4.4 kB)
Requirement already satisfied: packaging in ./env/lib/python3.12/site-packages (from gunicorn) (24.0)
  Downloading gunicorn-22.0.0-py3-none-any.whl (84 kB)           84.4/84.4 KB 3.6 MB/s eta 0:00:00
Installing collected packages: gunicorn
Successfully installed gunicorn-22.0.0
Ubuntu@ip-172-31-61-29:~/NB-A-PI$ sudo nano /etc/systemd/system/gunicorn.socket
Ubuntu@ip-172-31-61-29:~/NB-A-PI$ sudo nano /etc/systemd/system/gunicorn.service
Ubuntu@ip-172-31-61-29:~/NB-A-PI$ sudo systemctl start gunicorn.socket
Ubuntu@ip-172-31-61-29:~/NB-A-PI$ sudo systemctl enable gunicorn.socket
Created symlink /etc/systemd/system/sockets.target.wants/gunicorn.socket → /etc/systemd/system/gunicorn.socket.
Ubuntu@ip-172-31-61-29:~/NB-A-PI$ sudo nano /etc/nginx/sites-available/default
Ubuntu@ip-172-31-61-29:~/NB-A-PI$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
Ubuntu@ip-172-31-61-29:~/NB-A-PI$ sudo systemctl daemon-reload
Ubuntu@ip-172-31-61-29:~/NB-A-PI$ sudo systemctl restart gunicorn
Ubuntu@ip-172-31-61-29:~/NB-A-PI$ sudo systemctl restart nginx
```

To make the EC2 instance into a functional API we needed to install NGINX and Uvicorn. NGINX is open source software for web serving, reverse proxying, caching, and load balancing. NGINX was used to handle connections to the server while Uvicorn serves as the application

server running the backend of the API. This included re-configuring our application to accept incoming requests using NGINX as a proxy.

Deployment

```
curl -X 'GET' \
  'http://18.206.192.29/api/date?user_season=2023-24&user_date=2024-03-18' \
  -H 'Accept: application/json'
```

```
http://18.206.192.29/api/date?user_season=2023-24&user_date=2024-03-18
```

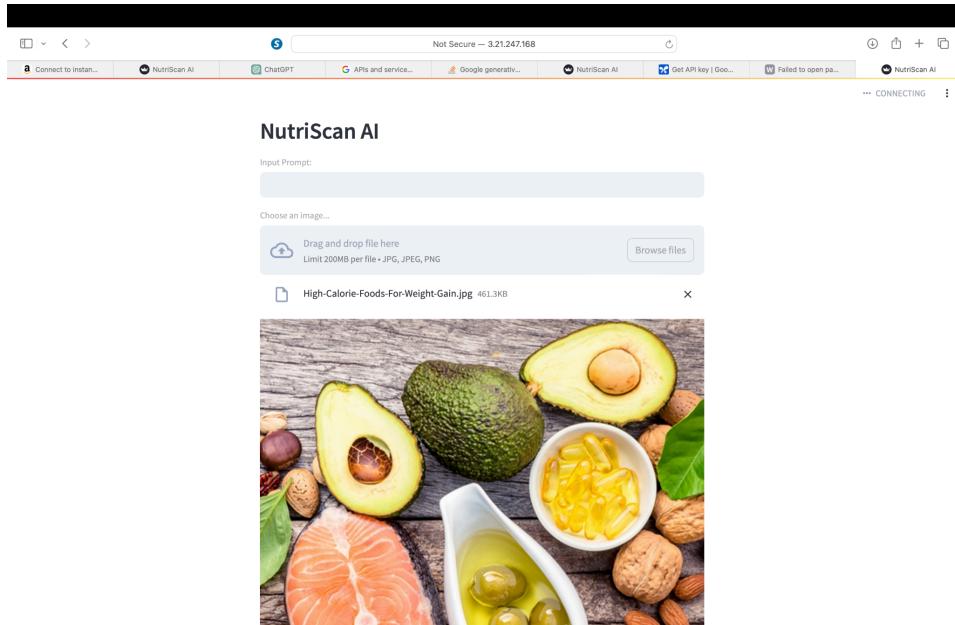
```
[{"date": 171072000000, "home_team": "SAC", "away_team": "MEM", "predicted_spread": 2, "predicted_outcome": "away_team_win"}, {"date": 171072000000, "home_team": "ATL", "away_team": "CLE", "predicted_spread": 2, "predicted_outcome": "away_team_win"}, {"date": 171072000000, "home_team": "IND", "away_team": "DET", "predicted_spread": 2, "predicted_outcome": "home_team_win"}, {"date": 171072000000, "home_team": "BOS", "away_team": "POR", "predicted_spread": 2, "predicted_outcome": "home_team_win"}, {"date": 171072000000, "home_team": "MIA", "away_team": "PHL", "predicted_spread": 2, "predicted_outcome": "home_team_win"}, {"date": 171072000000, "home_team": "GSM", "away_team": "NYK", "predicted_spread": 2, "predicted_outcome": "home_team_win"}]
```

After successfully launching our web server it was officially deployed and we were able to access the API using the public ip address of our instance.

Discussion

Due to the size of the application, other more pre-configured aws services were unavailable to us. Although the EC2 instance isn't super costly, there is no downtime so the costs of the api will steadily increase. While this isn't ideal for an API not expecting lots of web traffic, If our api was expecting lots of web traffic or was to suddenly see a sharp increase in web traffic this could be advantageous because we internally handle load balancing and scaling using NGINX. By managing these aspects internally, we have greater control over resource allocation and can optimize costs based on the specific needs of our application. This setup ensures that we only incur costs proportional to the actual usage and requirements of our API, making it one of the most economical options on AWS considering the capabilities of our application.

Project #3 Nutriscan AI



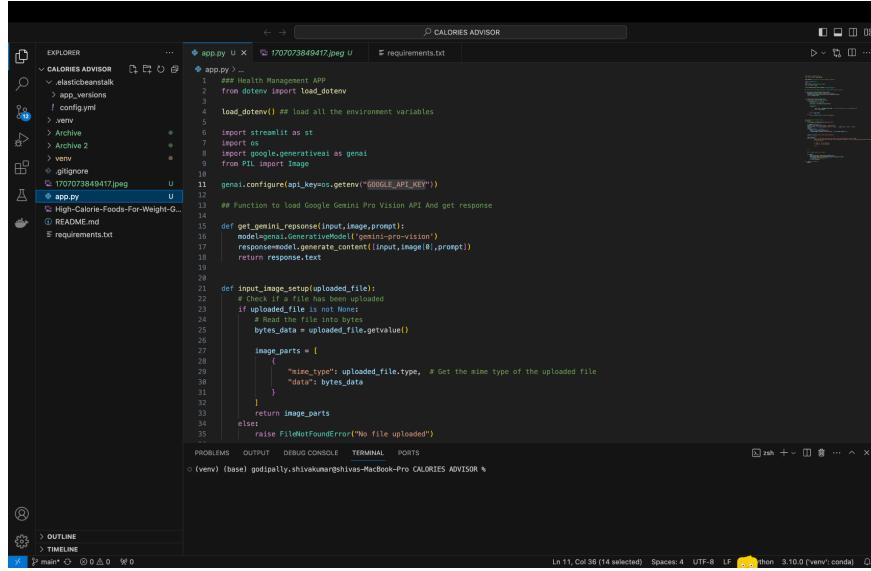
Project description

NutriScan AI is a cutting-edge calorie tracking application that uses advanced image recognition technology to analyze food and estimate calorie content. The app features a virtual dietary assistant powered by Google Gemini Pro, which provides personalized advice on meal optimization, catering to users' health goals. Designed for fitness enthusiasts, tech aficionados, and anyone interested in better managing their diet, NutriScan AI offers a user-friendly experience to analyze food images for calorie content and provide personalized dietary advice utilizing AI-powered insights and tailored guidance.

Implementation Overview

The project involved developing the app using Python, integrating it with Google Gemini Pro's API, and deploying it on an EC2 instance. Our efforts resulted in a holistic solution that bridges the gap between technology, nutrition, and health, enabling users to make informed dietary choices..

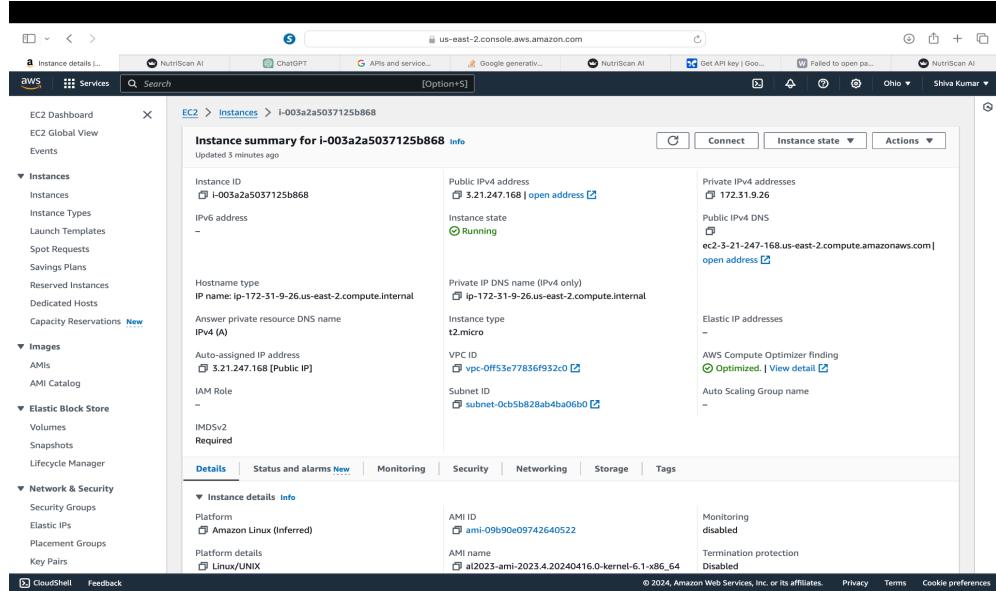
Local Development Environment



The screenshot shows a code editor interface with several files open. The main file is `app.py`, which contains Python code for a calorie advisor application. The code imports Streamlit, os, and PIL libraries, and uses the Google Gemini API via its Python client library. It defines functions for handling uploaded images and generating responses from the Gemini API. Other files visible include `requirements.txt`, `config.yml`, and `README.md`. The code editor has a dark theme and includes standard navigation and search tools.

We used Python for development, leveraging libraries like Streamlit to build an intuitive user interface. The application incorporated image recognition and AI technologies. We integrated Google Gemini Pro's API to power the virtual dietary assistant.

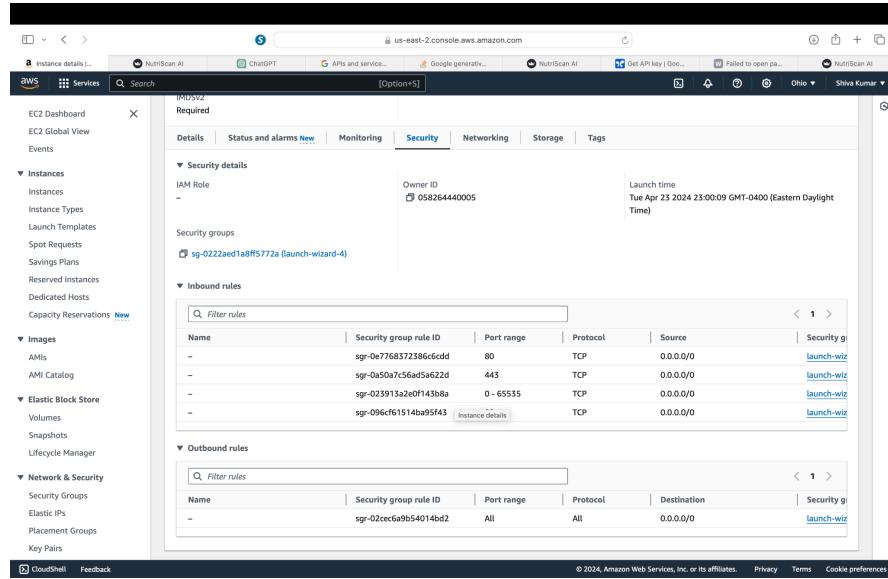
EC2 Instance Creation



The screenshot shows the AWS CloudWatch Metrics interface. On the left, there is a sidebar with navigation links for AWS services like EC2 Dashboard, EC2 Global View, Events, Instances, Images, Elastic Block Store, Network & Security, and more. The main content area displays the "Instance summary for i-003a2a5037125b868" for a t2.micro instance. The instance is currently running and has a public IPv4 address of 3.21.247.168. It was launched 3 minutes ago. The instance is associated with an Amazon Linux AMI and is part of a subnet. The "Details" tab is selected, showing information such as Platform (Amazon Linux Inferred), AMI ID (ami-09b90e09742640522), and AMI name (al2023-ami-2023.4.20240416.0-kernel-6.1-x86_64). The "Monitoring" tab indicates that monitoring is disabled. The bottom of the screen shows the AWS footer with links for CloudShell, Feedback, and various AWS services.

In setting up our cloud infrastructure, we created a t2.micro Amazon EC2 instance. This instance type was chosen for cost efficiency and functionality.

Security rules updated



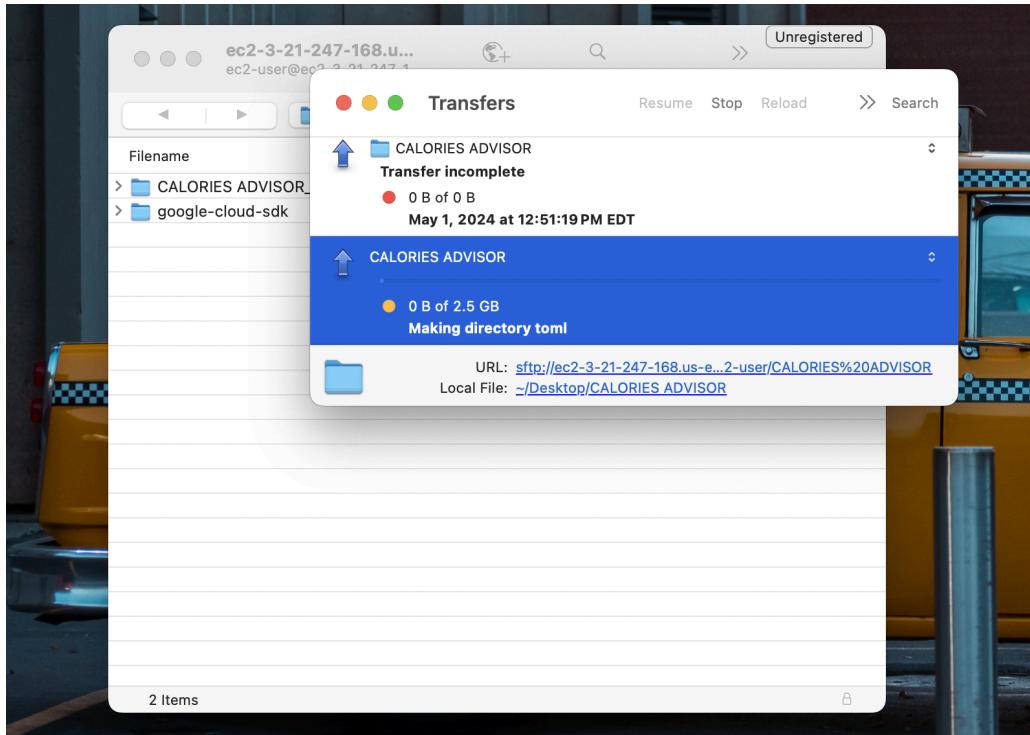
The security configuration for our EC2 instance involved updating both inbound and outbound rules to meet our specific needs. The instance was associated with a security group created during the setup wizard, and we defined multiple inbound rules to allow traffic over common web ports (80 and 443) as well as a broader range of TCP ports. This ensured our instance could function as a web server while also being flexible enough to accommodate other services. For outbound rules, we enabled unrestricted access, ensuring the instance could communicate with any necessary external services. These configurations provided the right balance between functionality and security, supporting our application's needs while protecting it from unauthorized access.

SSH into EC2 Instance

To further secure and manage our EC2 instance, we utilized SSH to establish a secure shell connection. This approach allowed us to remotely access the instance's command line,

facilitating direct control over configurations, software installations, and system maintenance. Using SSH, we authenticated with a private key, ensuring secure access to the instance. This setup provided us with the flexibility to manage the instance efficiently while maintaining strong security practices, essential for protecting sensitive data and ensuring the smooth operation of our cloud-based applications.

CyberDuck SFTP



In addition to using SSH, we leveraged Cyberduck for secure file transfer using SFTP. This allowed us to efficiently manage and transfer files between our local environment and the EC2 instance. With Cyberduck, we benefited from an intuitive graphical interface, simplifying the process of uploading and downloading files. Using this method, we securely managed project files, application assets, and logs, enhancing our workflow while maintaining robust security measures. This approach was vital for ensuring seamless deployment and management of our cloud-based applications.

Set up EC2 Instance environment for web server

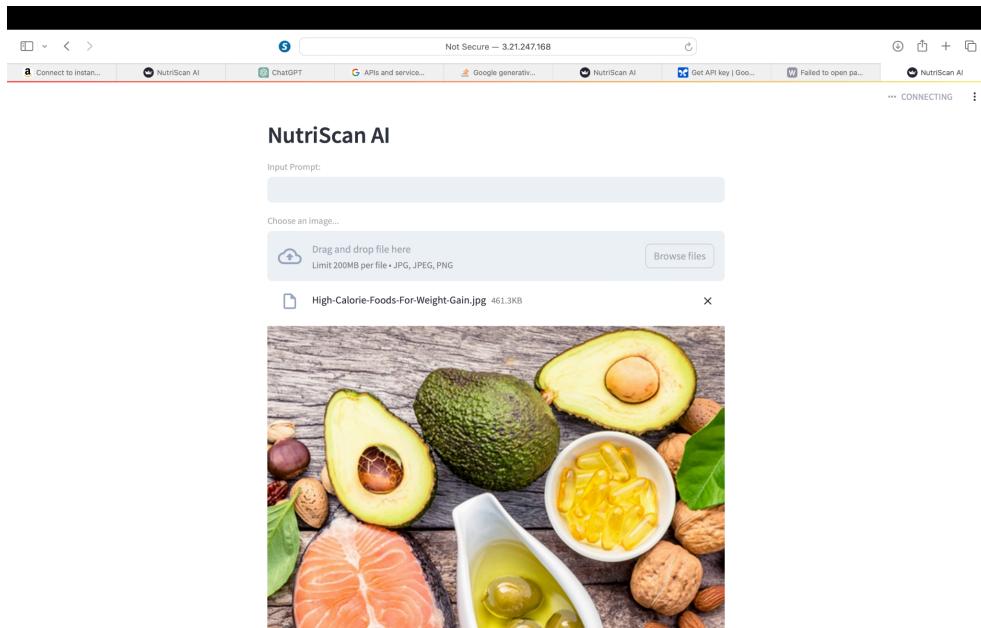
```
Requirement already satisfied: pyyaml==6.1.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 37)) (6.1.1)
Requirement already satisfied: pypi-modules==9.3.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 8)) (9.3.0)
Requirement already satisfied: pydeck==0.8.1b1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 39)) (0.8.1b1)
Requirement already satisfied: Pygments==2.17.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 40)) (2.17.2)
Requirement already satisfied: python-dateutil==2.8.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 41)) (2.8.2)
Requirement already satisfied: python-dotenv==1.0.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 42)) (1.0.1)
Requirement already satisfied: python-dotenv==1.0.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 43)) (2.1.1)
Requirement already satisfied: pytz==2024.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 44)) (2024.1)
Requirement already satisfied: requests==2.31.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 45)) (2.31.0)
Requirement already satisfied: requests==2.31.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 46)) (13.7.0)
Requirement already satisfied: rpsd-pys==0.18.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 47)) (0.18.0)
Requirement already satisfied: rsmq==4.9 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 48)) (4.9)
Requirement already satisfied: rich==13.7.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 49)) (1.4.2)
Requirement already satisfied: scipy==1.13.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 50)) (1.13.0)
Requirement already satisfied: six==1.16.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 51)) (1.16.0)
Requirement already satisfied: smmap==5.0.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 52)) (5.0.1)
Requirement already satisfied: streamlit==1.31.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 53)) (1.31.1)
Requirement already satisfied: tenacity==2.3.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 54)) (2.3.0)
Requirement already satisfied: tensorflow==3.4.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 55)) (3.4.0)
Requirement already satisfied: toml==0.10.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 56)) (0.10.2)
Requirement already satisfied: toolz==0.12.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 57)) (0.12.1)
Requirement already satisfied: tornado==6.4 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 58)) (6.4)
Requirement already satisfied: tqdm==4.66.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 59)) (4.66.2)
Requirement already satisfied: typeguard==3.10.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 60)) (3.10.2)
Requirement already satisfied: typeguard==3.10.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 61)) (2024.1)
Requirement already satisfied: tzlocal==2.2.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 62)) (2.2.0)
Requirement already satisfied: validators==0.22.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 64)) (0.22.0)
Requirement already satisfied: Werkzeug==2.0.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 65)) (2.0.2)
Requirement already satisfied: werkzeug==2.0.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 66)) (0.6.2)
Requirement already satisfied: zip==3.17.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 67)) (3.17.0)
Requirement already satisfied: watchdog==2.1.8 in /home/ec2-user/.local/lib/python3.9/site-packages (from streamlit==1.31.1->-r requirements.txt (line 53)) (4.0.0)
[ec2-user@ip-172-31-9-26 CALORIES ADVISOR\$ streamlit run app.py
Collecting usage statistics. To deactivate, set browser.gatherUsageStats to False.

You can now view your Streamlit app in your browser.

Network URL: http://172.31.9.26:8501
External URL: http://3.21.247.168:8501
2024-05-01 16:47:59.885 Uncought app exception
```

To set up the EC2 instance as a web server, we first ensured that the necessary software dependencies used in the development environment were installed. The screenshot shows a successful installation of Python packages from a requirements file, which was crucial for our application. Using the terminal, we installed and verified the presence of key packages such as streamlit, which serves as the backbone for our web application. Next, we ran the web server, using Streamlit to host our application. The output provided network URLs, including both internal and external addresses, allowing us to access the web application securely.

Deployment



The deployment of NutriScan AI on our EC2 instance involved configuring a secure environment for web hosting and integrating our machine learning model. The screenshot shows the user interface, where users can upload images of their meals to analyze calorie content.

Discussion

The development and deployment of NutriScan AI on an EC2 instance underscored the importance of integrating advanced technology with user-centric design in achieving impactful solutions. Throughout this project, we navigated through several critical phases, from setting up a secure cloud environment to integrating machine learning for dietary analysis and establishing a user-friendly interface using Streamlit.

Deploying this application as an EC2 instance allows for full control over server environments for customization and flexibility in software and configuration choices. Scalability is readily achievable, enabling seamless handling of fluctuating traffic demands while maintaining performance. Additionally, the pay-as-you-go pricing model enhances cost-effectiveness, particularly for applications with variable usage patterns. Managing EC2 instances entails technical expertise and ongoing maintenance tasks, potentially increasing operational overhead. Scalability configuration complexities may arise, and improper scaling can affect performance and cost-efficiency. Moreover, reliance on a single EC2 instance introduces a potential single point of failure, while ensuring security requires meticulous attention to access controls and firewall configurations to mitigate risks of unauthorized access and breaches.

Conclusion

This project emphasized the value of leveraging cloud technologies and machine learning to address a variety of real-world challenges. By focusing on both the technical and user experience aspects this project underscored the significance of utilizing cloud technologies and machine learning to tackle real-world challenges effectively. By addressing both technical intricacies and user experience considerations, we developed a comprehensive solution that not only delivers AI-powered solutions to consumers and diverse applications but also enhances our teamwork capabilities. Our focus on the deployment aspect of cloud technologies and machine learning provided invaluable insights into efficient deployment processes, allowing us to understand best practices and their impact on project success. This experience not only expanded our knowledge of cloud infrastructure but also equipped us with practical skills essential for deploying scalable and impactful AI-powered applications collaboratively within teams.