**Real-Time Anomaly Detection System using Amazon SageMaker and Amazon EKS**

**Puneeth Reddy Golla**

### **Abstract**

The anomaly detection project focuses on building a scalable and efficient system to identify anomalous data points using machine learning techniques and deploying it on Amazon Web Services (AWS). The project leverages a range of AWS services, including SageMaker Studio for model training, Elastic Container Registry (ECR) for storing Docker images, and Elastic Kubernetes Service (EKS) for deployment. Python-based tools such as Flask were used to create an API, and the app was containerized using Docker to ensure portability and scalability. The integration of these technologies showcases a modern workflow for real-time anomaly detection. Challenges included configuring subnets for EKS, resolving compatibility issues between Python libraries, and managing AWS permissions.

### **Introduction**

In today’s data-driven world, detecting anomalies in datasets is a critical requirement for ensuring data quality, operational efficiency, and fraud prevention. Anomalies, often indicative of unusual or erroneous events, can disrupt processes if not identified promptly. This project aims to build a robust anomaly detection system that leverages machine learning for identifying such data points and deploys the solution on AWS for real-time processing and scalability.

The system was designed using Isolation Forest, a popular unsupervised learning algorithm, to identify anomalies in a credit card transactions dataset. The Flask framework was employed to create an API for the model, and the entire application was containerized with Docker for seamless deployment. The deployment was further extended to AWS EKS, demonstrating the use of Kubernetes for scaling the application in a production environment.

The project not only highlights the utility of various AWS services but also provides an example of integrating machine learning workflows with cloud platforms. The experience gained from this project serves as a steppingstone for deploying similar real-world applications.

### **Aim of the Project**

The primary objective of this project is to develop and deploy a scalable anomaly detection system capable of identifying irregularities in data, such as fraudulent transactions or operational anomalies. The project aims to demonstrate the integration of machine learning techniques with cloud services to create an end-to-end pipeline for real-time anomaly detection.

The specific goals include:

1. **Model Development**: Training a machine learning model to identify anomalies in a given dataset using AWS SageMaker.
2. **API Creation**: Building a Flask-based API to expose the model for real-time predictions.
3. **Containerization**: Packaging the API and model into a Docker container to ensure portability and scalability.
4. **Cloud Deployment**: Leveraging AWS services like Elastic Container Registry (ECR) and Elastic Kubernetes Service (EKS) for deploying and managing the application in a production-ready environment.

### **Dataset Selection**

For this project, we utilized a publicly available dataset of credit card transactions. The dataset includes a mix of normal and fraudulent transactions, making it ideal for anomaly detection tasks. It contains multiple features derived from principal component analysis (PCA), anonymized to ensure data privacy, and a Class column indicating whether a transaction is normal (0) or fraudulent (1).

Key characteristics of the dataset:

* **Number of Instances**: 284,807
* **Number of Features**: 30, including NormalizedAmount and Class.
* **Anomalies**: Approximately 0.17% of the transactions are fraudulent.

The dataset's high imbalance posed a challenge, addressed by using the Isolation Forest algorithm, which excels in detecting outliers in imbalanced datasets.

### **Methodology**

This project follows a systematic workflow to build and deploy a scalable anomaly detection system. Each step leverages a combination of Python-based tools and AWS services, ensuring a robust end-to-end solution. The methodology is detailed below:

#### **1. AWS SageMaker Studio Setup**

* **Purpose**: SageMaker Studio was used to train the Isolation Forest model for anomaly detection.
* **Steps**:
  1. A Jupyter Notebook instance was launched within SageMaker Studio.
  2. The credit card transactions dataset was uploaded to an S3 bucket for accessibility within SageMaker.
  3. The Isolation Forest algorithm was trained using the dataset to identify anomalous transactions.
  4. The trained model was saved as a .joblib file and stored back in the S3 bucket for further use.

#### **2. Python Code for Model Training**

The core model training was executed in SageMaker using Python. The key steps included:

* Splitting the dataset into training and testing sets.
* Configuring the Isolation Forest algorithm with parameters such as:
  + n\_estimators = 100
  + contamination = 0.01
* Training the model on the dataset and evaluating its performance.
* Saving the trained model using the joblib library.

#### **3. Flask App Development**

* **Purpose**: To create a REST API for the trained model, enabling real-time predictions.
* **Steps**:
  1. A Flask app was developed locally, with the following endpoints:
     + /: Welcome page.
     + /predict: Accepts input data and returns anomaly predictions.
  2. The app loads the trained model and uses it to predict whether incoming transactions are anomalous.
  3. The app was tested locally to ensure proper functioning before containerization.

#### **4. Dockerization**

* **Purpose**: To package the Flask app and model into a Docker container for portability and ease of deployment.
* **Steps**:
  1. A Dockerfile was created to define the application environment, including dependencies such as Flask and Scikit-learn.
  2. The Docker image was built and tested locally.
  3. The image was tagged for uploading to AWS Elastic Container Registry (ECR).

#### **5. AWS Elastic Container Registry (ECR)**

* **Purpose**: To store the Docker image for deployment in EKS.
* **Steps**:
  1. The Docker image was pushed to a private ECR repository.
  2. ECR was configured to allow access for EKS during deployment.

#### **6. AWS Elastic Kubernetes Service (EKS)**

* **Purpose**: To deploy the containerized application and enable scalability.
* **Steps**:
  1. An EKS cluster was created, ensuring proper availability zones were selected.
  2. Worker nodes were added to the cluster.
  3. The application was deployed using Kubernetes manifests:
     + A Deployment manifest to define the app’s replicas and container configuration.
     + A Service manifest to expose the app via a load balancer.
  4. The deployed app was tested using the external load balancer URL.

### **AWS Services Used**

This project leveraged several AWS services to streamline the development and deployment of the anomaly detection system:

1. **Amazon S3**: Used for storing the dataset and the trained machine learning model, ensuring seamless access for training and deployment.
2. **Amazon SageMaker**: Provided an integrated environment to preprocess data, train the Isolation Forest model, and export the results.
3. **Amazon Elastic Container Registry (ECR)**: A secure repository for storing the Docker image of the Flask application, enabling smooth integration with Kubernetes.
4. **Amazon Elastic Kubernetes Service (EKS)**: Managed Kubernetes clusters for deploying the Dockerized application and scaling it in a production environment.
5. **AWS IAM**: Ensured secure access to resources like S3, SageMaker, ECR, and EKS by managing roles and permissions effectively.
6. **AWS CloudShell**: Offered a CLI environment for managing AWS services and pushing the Docker image to ECR.
7. **AWS Elastic Load Balancer (ELB)**: Exposed the deployed application to the internet, providing a public endpoint for the anomaly detection API.

### **Challenges Faced**

During the development and deployment of the anomaly detection system, several challenges were encountered, each providing valuable learning experiences:

1. **SageMaker Model Compatibility**: Ensuring the trained model (IsolationForest) was compatible with Flask required careful attention to Python library versions and serialization formats. Mismatched versions initially caused deserialization errors.
2. **Docker Port Conflicts**: Running the Flask app in Docker led to port conflicts when multiple containers were launched. This was resolved by dynamically assigning ports and carefully managing container states.
3. **AWS EKS Subnet Configuration**: The EKS cluster creation initially failed due to unsupported availability zones. Adjusting the subnet configuration to span valid zones resolved the issue.
4. **Containerized App Testing**: Ensuring the Flask app functioned correctly inside a Docker container required multiple iterations of debugging, especially for API endpoints handling JSON data.
5. **Resource Access Permissions**: Configuring IAM roles and policies for secure resource access across AWS services was time-intensive but crucial for ensuring functionality and security.
6. **Dataset Imbalance**: The extreme imbalance in the dataset (only 0.17% anomalies) posed a challenge for effective model training. This was mitigated using the Isolation Forest algorithm, which is well-suited for such scenarios.

### **Conclusion**

This project successfully demonstrates the integration of machine learning and cloud-based technologies to develop and deploy a scalable anomaly detection system. By leveraging the Isolation Forest algorithm, the model effectively identified anomalies in a highly imbalanced credit card transactions dataset. AWS services such as SageMaker, ECR, and EKS provided the necessary infrastructure for training, containerization, and deployment, ensuring a streamlined and scalable workflow.

Despite encountering challenges like subnet configuration issues in EKS, library compatibility, and dataset imbalance, each hurdle provided an opportunity to refine the solution and enhance the project's robustness. The final deployment of the Flask-based API on Kubernetes not only showcased real-time anomaly detection but also illustrated the benefits of using cloud-native tools for modern data science workflows.

This project highlights the potential for deploying similar systems in real-world scenarios, such as fraud detection, operational monitoring, or predictive maintenance, emphasizing the value of combining machine learning techniques with scalable cloud services.

Appendix:

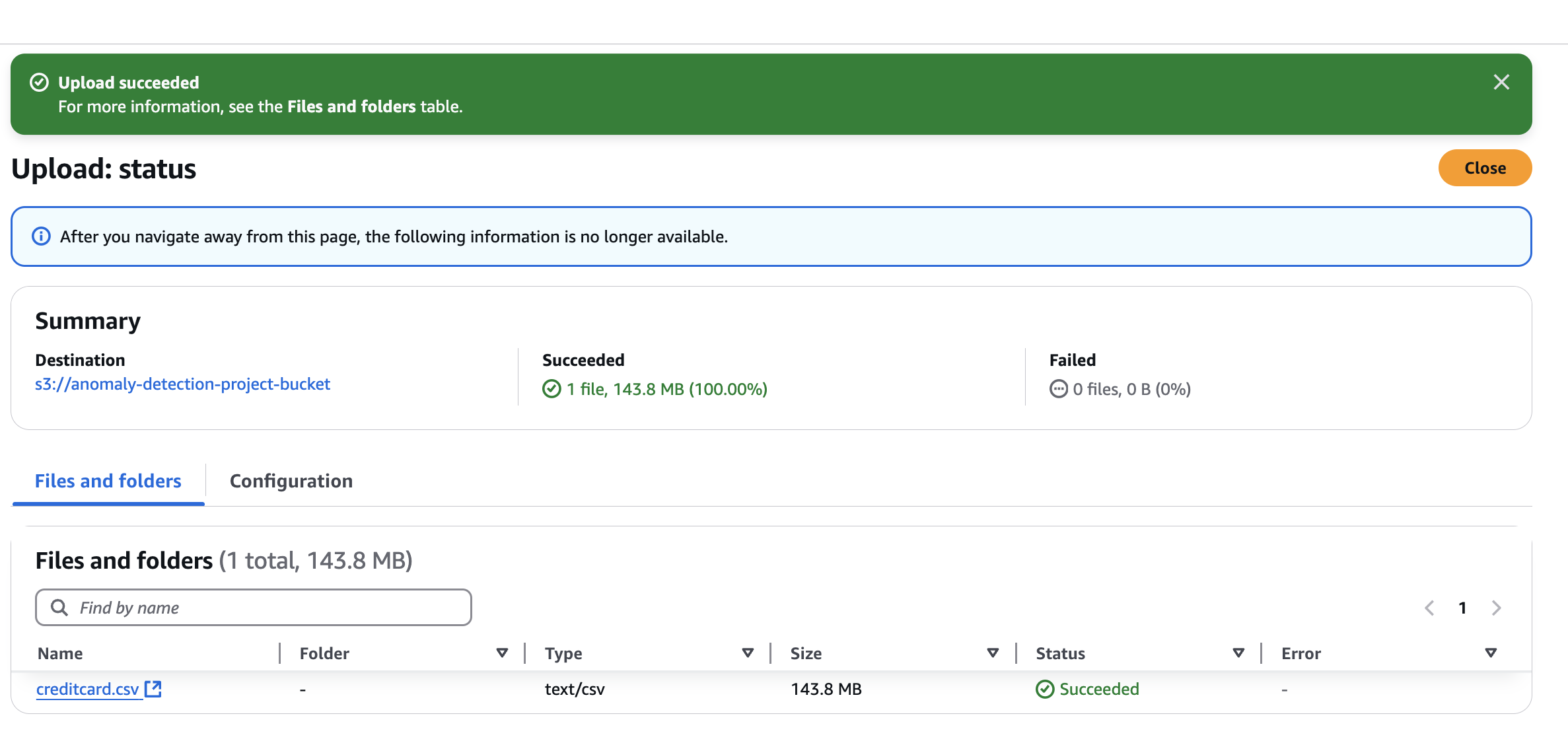
Step 1: Create and Configure an S3 Bucket

A screenshot of a computer

Description automatically generated

A screenshot of a computer

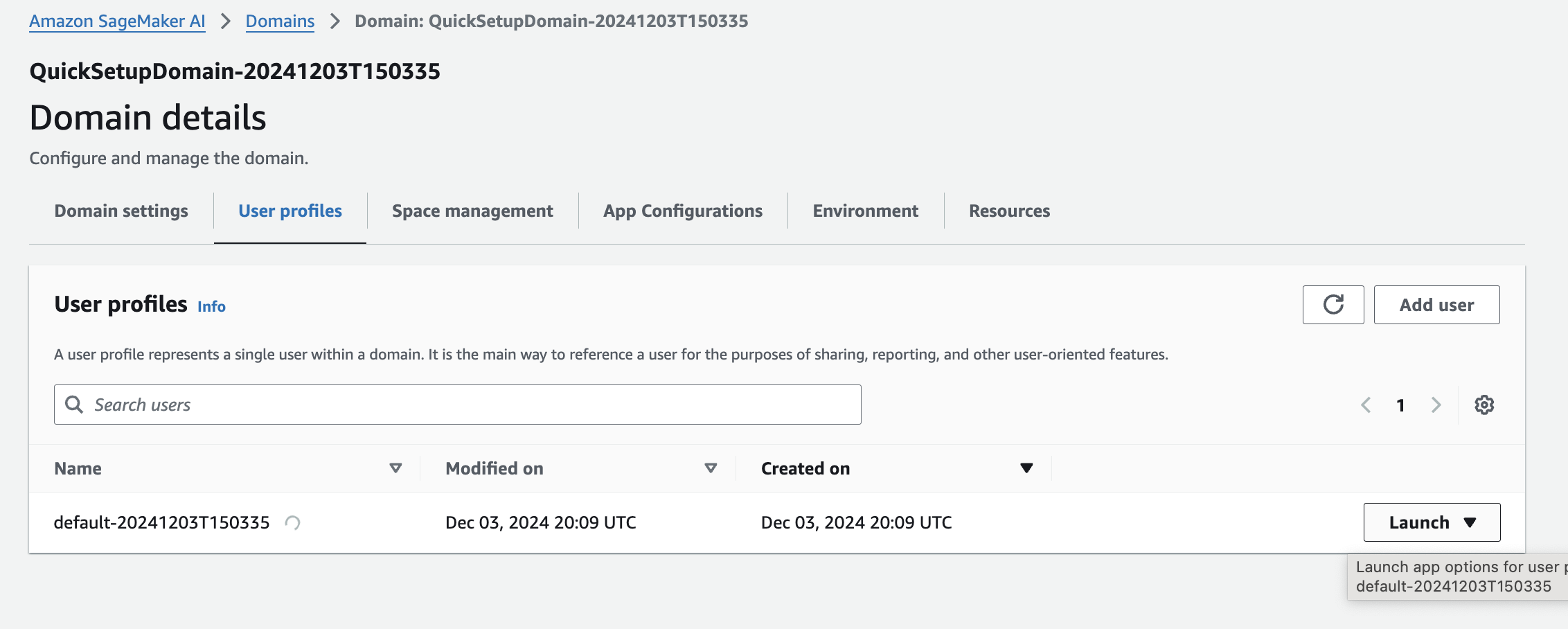
Description automatically generated



Step 2: Train the Model Using Amazon SageMaker

A screenshot of a computer

Description automatically generated

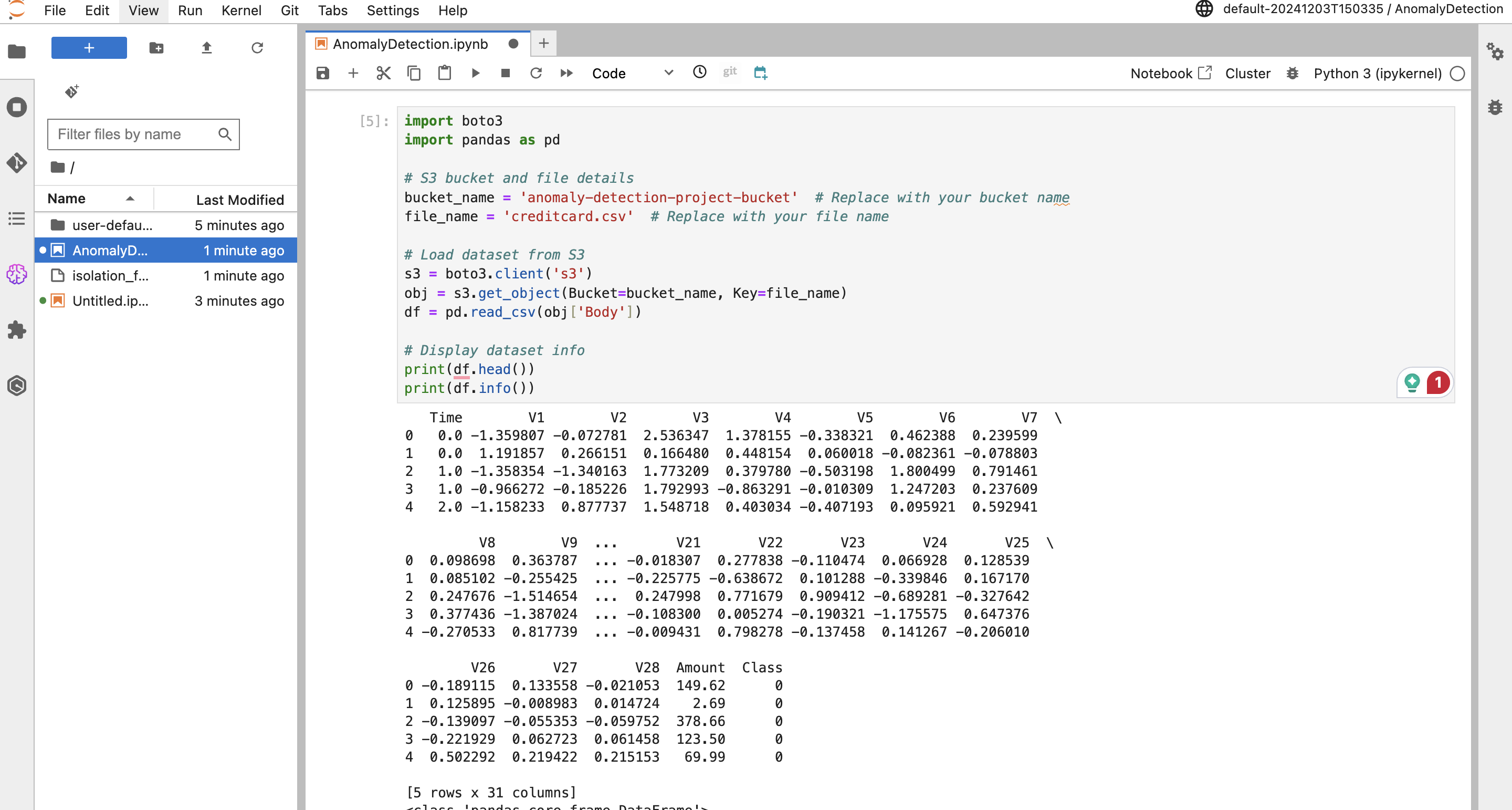


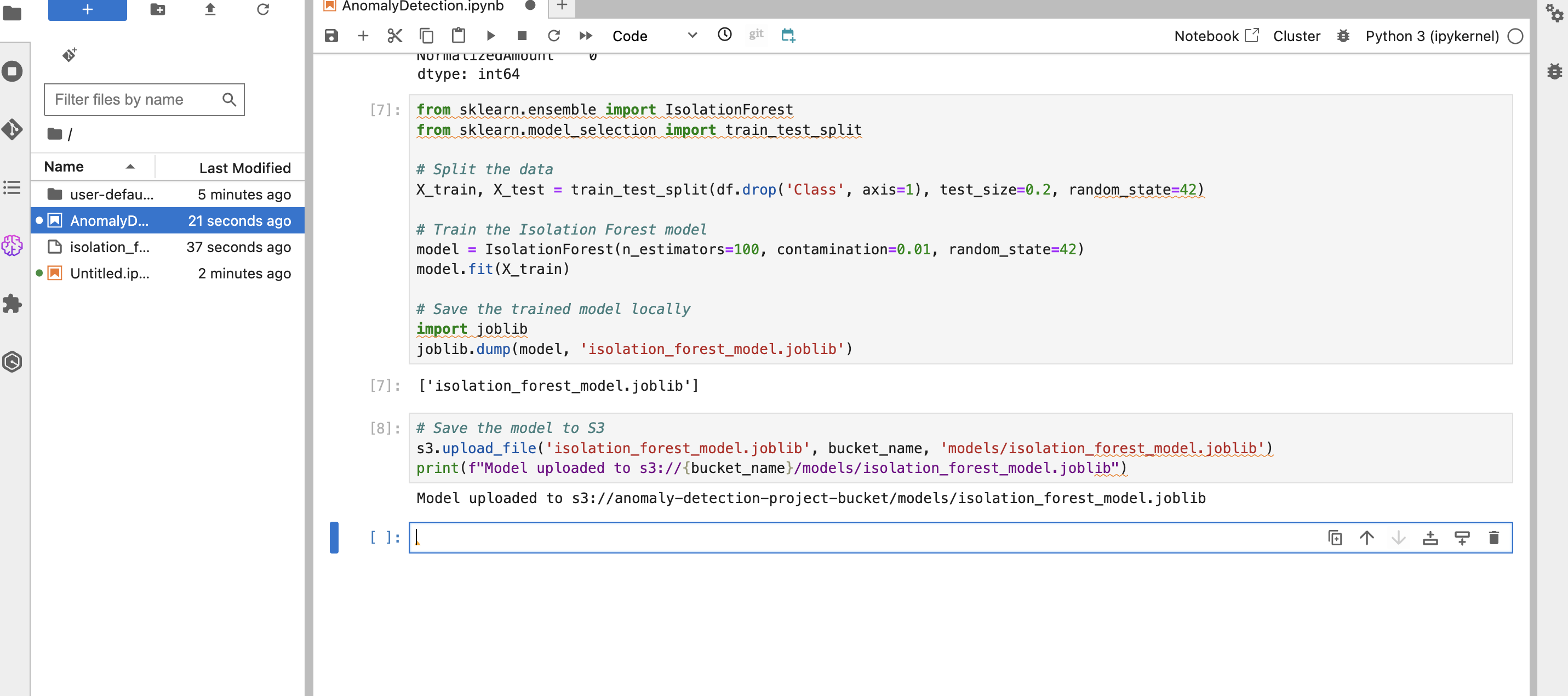
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated





**Step 3: Containerize the Model Using Docker**

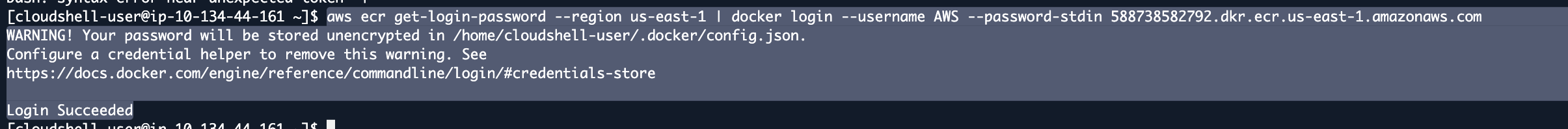


A screenshot of a phone

Description automatically generated

A screenshot of a computer program

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated