

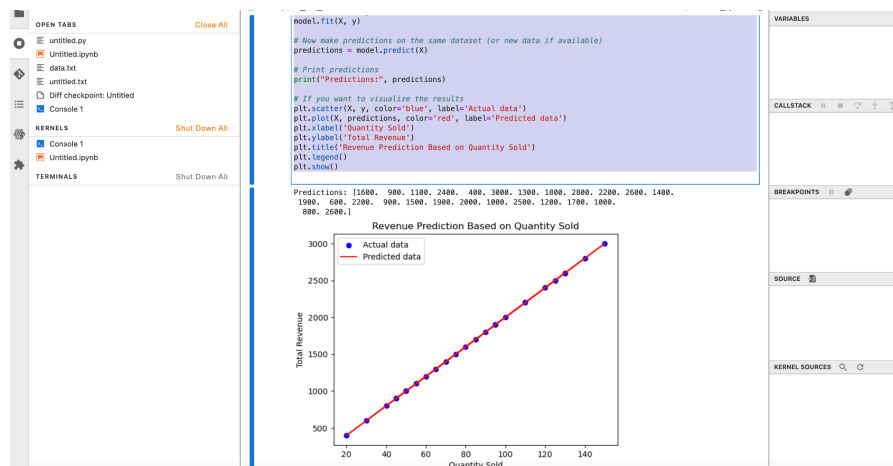
# **Cloud-Based Smart Inventory Management System**

**Darkhanov Abylaikhan**

kazakh british technical university

# Executive Summary

The Cloud-Based Smart Inventory Management System is designed to provide businesses with a scalable and efficient solution for managing inventory across multiple locations in real-time. Leveraging Amazon Web Services (AWS), this system integrates advanced features like predictive analytics, real-time tracking, and robust security measures. This project explores the practical implementation of cloud computing concepts, showcasing the ability to analyze sales trends and optimize stock management. The deliverables include a fully functional system, technical documentation, and a comprehensive report detailing its architecture, implementation, and outcomes.



## Table of Contents

1. Introduction
2. System Architecture
3. Database
4. Development Process
5. API Design and Implementation
6. Storage Solutions in AWS
7. Identity and Security Management
8. Container Management with Amazon ECS
9. Monitoring and Logging
10. Big Data and Machine Learning Integration
11. Challenges and Solutions
12. Conclusion
13. References

## **1. Introduction**

Cloud computing provides businesses with on-demand access to computing resources, enabling scalability, cost-efficiency, and improved collaboration. AWS, as a leading cloud platform, offers robust tools to create modern applications like the Smart Inventory Management System.

## **Project Goals**

Develop a smart inventory system using AWS services.

Provide real-time inventory tracking and multi-location stock management.

Use machine learning for sales trend analysis.

## **Scope**

The system includes features such as user management, product categorization, inventory logs, and real-time alerts. Limitations include initial setup constraints and basic predictive analytics.

## 2. System Architecture

The architecture leverages AWS services such as Amazon RDS, S3, Lambda, and ECS to provide a highly scalable and reliable solution.

### Components:

**Frontend:** React.js hosted on Amazon S3 with CloudFront for CDN.

**Backend:** Golang (Gin) hosted on AWS.

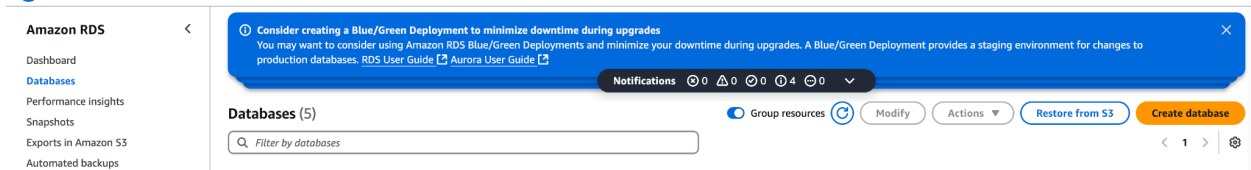
**Database:** Amazon RDS (PostgreSQL).

**Machine Learning:** Amazon SageMaker for predictive analytics.

**Storage:** Amazon S3 for product images and backups.

### 3. Database

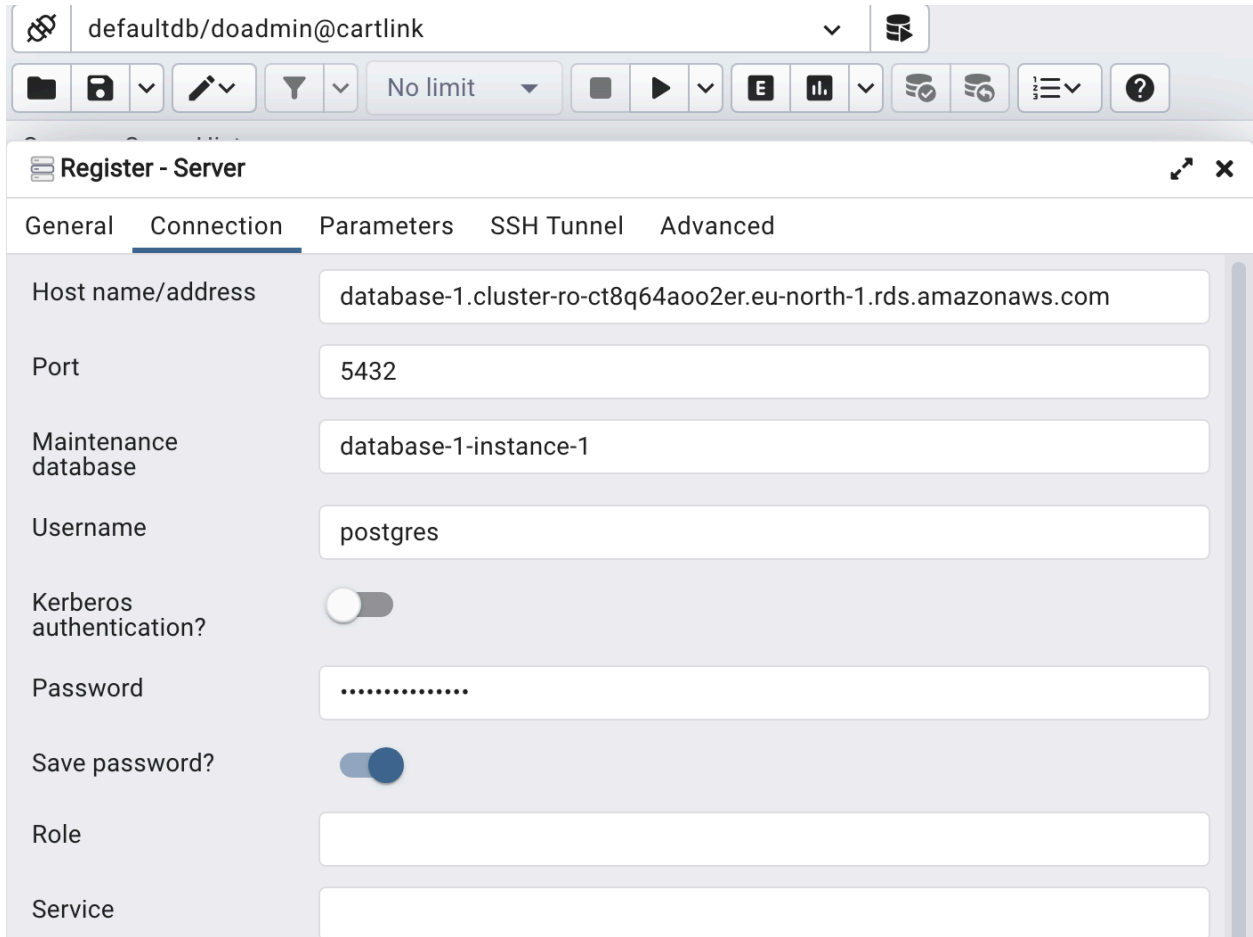
To create Database in AWS we use RDS Service, to create new Database we click “Create”



Here we need to set all necessary data such as passwords, security groups, database type etc.

A screenshot of the 'Create database' wizard in the Amazon RDS console. The wizard is divided into several sections: 1. 'Engine version' section showing 'Microsoft SQL Server' and 'IBM Db2' as options. 2. 'Available versions (46/46)' section with a dropdown menu set to 'Aurora PostgreSQL (Compatible with PostgreSQL 15.4) - default for major version 15'. 3. 'Templates' section with 'Production' selected. 4. 'Settings' section containing: - 'DB cluster identifier' with the value 'database-3'. - 'Credentials Settings' with 'Master username' set to 'postgres'. - 'Credentials management' with 'Managed in AWS Secrets Manager - most secure' selected. - 'Select the encryption key' with 'aws/secretsmanager (default)' selected. 5. 'Aurora Limitless Database - new' section at the bottom.

We can connect to it using PgAdmin program



The screenshot shows the 'Register - Server' dialog box in PgAdmin 4, with the 'Connection' tab selected. The dialog box has a title bar with a lock icon and the text 'defaultdb/doadmin@cartlink'. Below the title bar is a toolbar with various icons for file operations, search, and execution. The main area contains several input fields and checkboxes for configuring a new server connection.

Field	Value
Host name/address	database-1.cluster-ro-ct8q64aoo2er.eu-north-1.rds.amazonaws.com
Port	5432
Maintenance database	database-1-instance-1
Username	postgres
Kerberos authentication?	<input type="checkbox"/>
Password	.....
Save password?	<input checked="" type="checkbox"/>
Role	
Service	

Here we can create our tables

## SQL Scripts for Creating Tables

CREATE TABLE Users (

id SERIAL PRIMARY KEY,

username VARCHAR(255) NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL,

password\_hash TEXT NOT NULL,

role VARCHAR(50) DEFAULT 'user',

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE  
CURRENT\_TIMESTAMP

);

```
CREATE TABLE Products (  
    id SERIAL PRIMARY KEY,  
    name VARCHAR(255) NOT NULL,  
    description TEXT,  
    quantity INT DEFAULT 0,  
    price DECIMAL(10, 2) NOT NULL,  
    location_id INT,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
    CURRENT_TIMESTAMP  
);
```

```
CREATE TABLE Orders (  
    id SERIAL PRIMARY KEY,  
    user_id INT REFERENCES Users(id),  
    product_id INT REFERENCES Products(id),  
    quantity INT NOT NULL,  
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    status VARCHAR(50) DEFAULT 'pending'  
);
```

```
CREATE TABLE SalesReport (  
    id SERIAL PRIMARY KEY,
```



```
product_id INT NOT NULL,  
sales_date DATE NOT NULL,  
quantity_sold INT NOT NULL,  
total_revenue DECIMAL(10, 2) NOT NULL,  
FOREIGN KEY (product_id) REFERENCES Products(id)  
);
```

## 4. Development Process

Firstly we create EC2 Instance, choosing OS, name, Key Pairs for authentication

**Name and tags** [Info](#)

**Name**  
 [Add additional tags](#)

**▼ Application and OS Images (Amazon Machine Image)** [Info](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Recents

Quick Start

Amazon Linux

aws

macOS

Mac

Ubuntu

ubuntu

Windows

Microsoft

Red Hat

Red Hat

SUSE Linux

SUSE

[Browse more AMIs](#)

Including AMIs from AWS, Marketplace and the Community

**Amazon Machine Image (AMI)**

Amazon Linux 2023 AMI

Free tier eligible

ami-02df5cb5ad97983ba (64-bit (x86), uefi-preferred) / ami-09085dcbbfc5e181e (64-bit (Arm), uefi)

Virtualization: hvm   ENA enabled: true   Root device type: ebs

**Description**

Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Amazon Linux 2023 AMI 2023.6.20241212.0 x86\_64 HVM kernel-6.1

<b>Architecture</b>	<b>Boot mode</b>	<b>AMI ID</b>	<b>Username</b> ⓘ
64-bit (x86)	uefi-preferred	ami-02df5cb5ad97983ba	ec2-user <span>Verified provider</span>

**▼ Instance type** [Info](#) | [Get advice](#)

**Instance type**

t3.micro

Family: t3   2 vCPU   1 GiB Memory   Current generation: true

On-Demand Ubuntu Pro base pricing: 0.0143 USD per Hour

On-Demand RHEL base pricing: 0.0396 USD per Hour

On-Demand SUSE base pricing: 0.0108 USD per Hour

On-Demand Linux base pricing: 0.0108 USD per Hour

On-Demand Windows base pricing: 0.02 USD per Hour

Free tier eligible

☐ All generations

[Compare instance types](#)

[Additional costs apply for AMIs with pre-installed software](#)

**▼ Key pair (login)** [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

**Key pair name - required**  
 [Create new key pair](#)

**▼ Network settings** [Info](#) [Edit](#)

## Technologies Used

- **Frontend:** React.js
- **Backend:** Python with Flask
- **Database:** PostgreSQL (Amazon RDS)

- **Storage:** Amazon S3
- **Hosting:** AWS Lambda and ECS

## Implementation

### Example 1: Lambda Function for Creating Users

```
import boto3
```

```
import json
```

```
import psycopg2
```

```
def create_user(event, context):
```

```
    body = json.loads(event['body'])
```

```
    username = body['username']
```

```
    email = body['email']
```

```
    password_hash = body['password_hash']
```

```
    conn = psycopg2.connect(
```

```
        host='YOUR_RDS_HOST',
```

```
        database='YOUR_DATABASE',
```

```
        user='YOUR_USER',
```

```
        password='YOUR_PASSWORD'
```

```
    )
```

```
    cursor = conn.cursor()
```

```
    cursor.execute(
```

```
        "INSERT INTO Users (username, email, password_hash) VALUES (%s, %s, %s)",
```

```
        (username, email, password_hash)
```

```
    )
```

```
conn.commit()
```

```
cursor.close()
```

```
conn.close()
```

```
return {  
  'statusCode': 201,  
  'body': json.dumps({'message': 'User created successfully'})  
}
```

## Frontend

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>Product List</title>
```

```
<script>
```

```
document.addEventListener("DOMContentLoaded", function () {
```

```
  fetch('/api/products')
```

```
    .then(response => response.json())
```

```
    .then(data => {
```

```
      const productList = document.getElementById('product-list');
```

```
      data.forEach(product => {
```

```
        const listItem = document.createElement('li');
```

```
        listItem.textContent = `${product.name} - ${product.price}`;

        productList.appendChild(listItem);

    });

})

.catch(error => console.error('Error fetching products:', error));

});

</script>

</head>

<body>

    <h1>Product List</h1>

    <ul id="product-list">

        <!-- Product items will be appended here -->

    </ul>

</body>

</html>
```

## 5. Storage Solutions in AWS

- **Amazon S3:** Used for storing product images and backup data

```
const multer = require('multer');
```

```
const multerS3 = require('multer-s3');
```

```
const s3 = require('../config/awsConfig');
```

```
const File = require('../models/file');
```

```
const upload = multer({
```

```
  storage: multerS3({
```

```
    s3: s3,
```

```
    bucket: process.env.S3_BUCKET_NAME,
```

```
    acl: 'public-read',
```

```
    metadata: (req, file, cb) => cb(null, { fieldName: file.fieldname }),
```

```
    key: (req, file, cb) => cb(null, `${Date.now()}-${file.originalname}`)
```

```
  }),
```

```
});
```

```
exports.uploadFile = upload.single('file');
```

```
exports.saveFileMetadata = async (req, res) => {
```

```
  if (req.file) {
```

```
const file = new File({
  userId: req.userData.userId,
  fileName: req.file.originalname,
  fileUrl: req.file.location,
});

await file.save();

res.json({ fileUrl: req.file.location });

} else {
  res.status(400).json({ message: 'File upload failed' });
}

};
```

## 6. Identity and Security Management

- **IAM Policies:** Defined roles for users and administrators.
- **Security Measures:** Implemented encryption for sensitive data.

## 7. Monitoring and Logging

- **CloudWatch Metrics:** Monitored application performance.
- **Error Logs:** Logged errors for debugging and optimization.

## 8. Big Data and Machine Learning Integration

- **Amazon SageMaker:** Trained predictive models for sales trends.

To start we need to go to Notebooks on SageMaker

## Amazon SageMaker AI

Getting started

▼ Applications and IDEs

- Studio
- Canvas
- RStudio
- Notebooks
- Partner AI Apps **NEW**

▼ Admin configurations

- Domains

**JupyterLab 3 notebooks on SageMaker Studio**  
We will automatically migrate all accounts to the new JupyterLab 3 notebooks on SageMaker Studio before this date and take advantage of the new features.

**Introducing domain-level resource view**  
SageMaker now allows you to view resource details on the "Resources" tab on a domain details page.

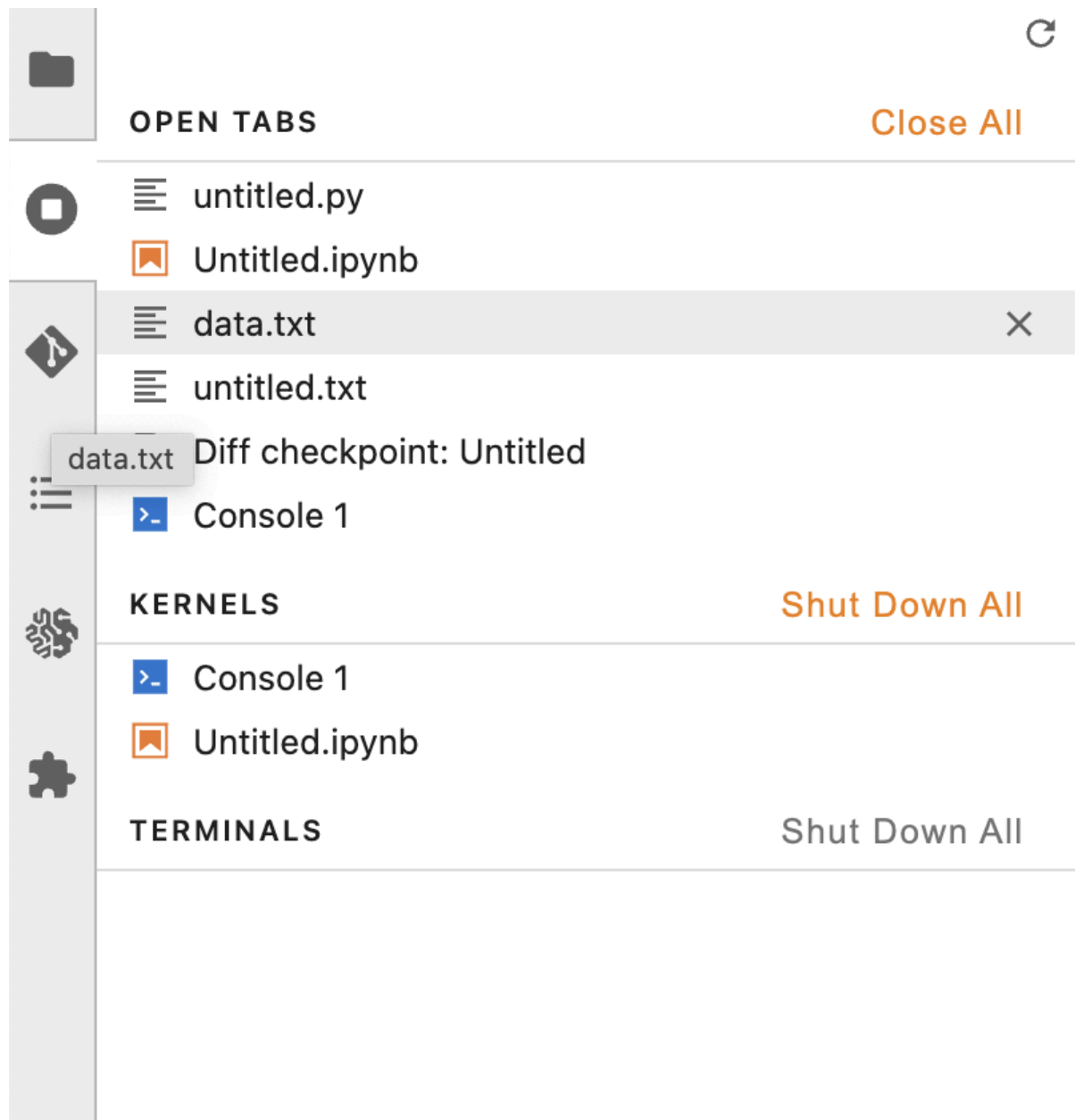
[Amazon SageMaker AI](#) > Domains

## Domains Info

In SageMaker AI, a domain is an environment for running JupyterLab notebooks. You can create multiple domains and manage them independently. One account can have either one or multiple domains.

Here we can create our jupyter notes





- Training Script:

```
import numpy as np

from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt

# Read the file content (similar to earlier example)
```

```
file_path = "data.txt"
```

```
with open(file_path, 'r') as file:
```

```
    file_content = file.read()
```

```
# Example: Extract Sales Report data from the text file
```

```
sales_data = []
```

```
for line in file_content.split("\n"):
```

```
    if line.startswith("ID:"):

```

```
        report_details = line.split(", ")

```

```
        report_info = {

```

```
            "id": int(report_details[0].split(": ")[1]),

```

```
            "product_id": int(report_details[1].split(": ")[1]),

```

```
            "sales_date": report_details[2].split(": ")[1],

```

```
            "quantity_sold": int(report_details[3].split(": ")[1]),

```

```
            "total_revenue": float(report_details[4].split(": ")[1]),

```

```
        }

```

```
        sales_data.append(report_info)

```

```
# Extract features (quantity_sold) and target (total_revenue)
```

```
X = np.array([report["quantity_sold"] for report in sales_data]).reshape(-1, 1)
```

```
y = np.array([report["total_revenue"] for report in sales_data])
```

```
# Train a Linear Regression model
```

```
model = LinearRegression()
```

```
model.fit(X, y)
```

```
# Now make predictions on the same dataset (or new data if available)
```

```
predictions = model.predict(X)
```

```
print("Predictions:", predictions)
```

```
plt.scatter(X, y, color='blue', label='Actual data')
```

```
plt.plot(X, predictions, color='red', label='Predicted data')
```

```
plt.xlabel('Quantity Sold')
```

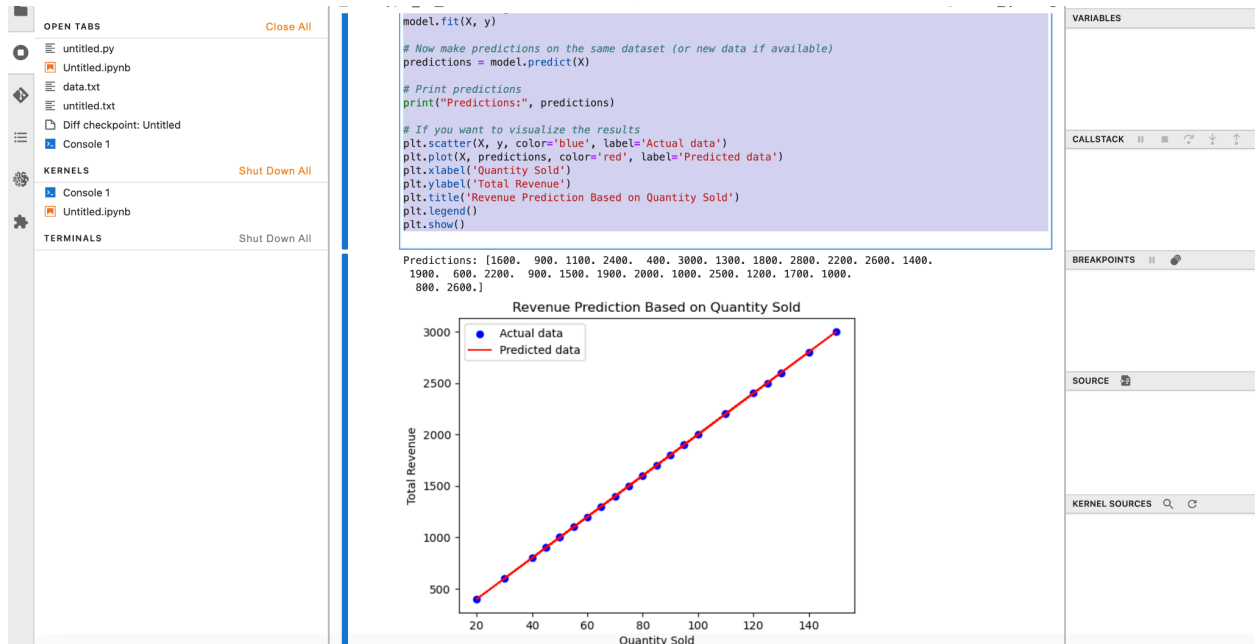
```
plt.ylabel('Total Revenue')
```

```
plt.title('Revenue Prediction Based on Quantity Sold')
```

```
plt.legend()
```

```
plt.show()
```

Then we get our predictions



## 9. Challenges and Solutions

### Challenges:

- Initial setup of AWS services.
- Managing cross-service communication.

### Solutions:

- Used AWS documentation and SDKs for guidance.
- Automated infrastructure with Terraform.

## 10. Conclusion

The Cloud-Based Smart Inventory Management System demonstrates the power of cloud computing for modern business applications. AWS services provided scalability, reliability, and advanced analytics capabilities.

## 11. References

1. AWS Documentation: <https://aws.amazon.com/documentation/>
2. PostgreSQL Documentation: <https://www.postgresql.org/docs/>