Diabetes Prediction using Logistic Regression: An End-to-End Solution

Batuhan Ayyıldız
Manisa Celal Bayar University

***Abstract*—In this study, a Logistic Regression model was developed to predict diabetes using the diabetes dataset obtained from Kaggle. The dataset underwent several stages including cleaning missing values, adding new features, applying various data visualization techniques, and comparing models. Based on the results, the Logistic Regression model was chosen for its ability to classify diabetes patients with the highest accuracy. This report comprehensively covers all processes from data preprocessing to model deployment.**

I. Introduction

Diabetes is a chronic disease that is increasingly prevalent worldwide and places a significant burden on healthcare systems. Early diagnosis of this disease is crucial for improving individual quality of life and reducing healthcare costs. Recent advancements in machine learning and data science offer revolutionary solutions for early diagnosis and management of diseases.

In this study, a Logistic Regression model was developed to predict diabetes using a dataset obtained from Kaggle. To enhance the model's performance, missing values in the dataset were corrected, new features were added, and various data visualization techniques were employed. Additionally, comparisons were made with different machine learning algorithms to evaluate model performance.

The Logistic Regression model was selected as the primary model for this study due to its high accuracy rate and interpretability. To make the model more accessible, FastAPI and Streamlit applications were developed and packaged within Docker containers. Docker's use increased the portability and compatibility of the applications, while deployment on AWS provided global access to the applications. This report demonstrates that we have provided an effective solution for diabetes prediction by thoroughly covering the process from data preprocessing to model deployment.

II. Understanding and Interpreting the Dataset

The diabetes dataset used in the data analysis and modeling process was obtained from the Kaggle platform. This dataset consists of 768 observations and 8 independent variables. The features and the content of the dataset are detailed below:

- **Pregnancies**: Represents the number of pregnancies a woman has had. It is used to understand the impact of pregnancy count on diabetes.
- **Glucose**: Measures blood glucose levels. It is considered an important biomarker for diabetes diagnosis.
- **BloodPressure**: Measures blood pressure. High blood pressure can be a factor associated with diabetes.
- **SkinThickness**: Measures skin thickness. Skin thickness may be related to insulin resistance and thus to diabetes risk.
- **Insulin**: Measures blood insulin levels. Insulin levels play a crucial role in diabetes diagnosis.
- **BMI**: Measures Body Mass Index. There may be a relationship between obesity and diabetes.
- **DiabetesPedigreeFunction**: Measures diabetes risk based on family history. It is used to understand the impact of genetic factors.
- **Age**: Indicates the age of the patients. Age can be an effective factor in diabetes risk.
- **Outcome**: As the target variable, indicates whether the person has diabetes (0: no diabetes, 1: diabetes present).

A. Feature Engineering

In addition to the existing features in the dataset, some new features were derived to improve model performance and better understand health status. This process is an important part of the data analysis and modeling process and was implemented to enhance the model's accuracy.

B. Glucose Classification

Glucose levels play a critical role in diabetes diagnosis. In this context, glucose levels have been categorized into three main categories:

- **Normal:** Values below 100 mg/dL are included in this category. These values generally reflect the glucose levels of healthy individuals.
- **Prediabetes:** Values between 100 mg/dL and 125 mg/dL are classified as prediabetes. This is considered a range where the risk of diabetes is increased.
- **Diabetes:** Values of 125 mg/dL and above are classified as diabetes. This indicates that individuals are diagnosed with diabetes.

This classification is an important step to better understand the relationship between glucose levels and diabetes risk and to enhance the model's performance.

C. Body Mass Index (BMI) Categories

Body Mass Index (BMI) shows the ratio of weight to height and is an important indicator in assessing health status. BMI values are categorized into the following four categories:

- **Underweight:** Individuals with a BMI value below 18.5 are included in this category. This may indicate malnutrition or health issues.
- **Healthy:** Individuals with a BMI value between 18.5 and 24.9 are considered to be in a healthy weight range.
- **Overweight:** Individuals with a BMI value between 25 and 29.9 are classified as overweight. This may indicate an increased risk of obesity.
- **Obese:** Individuals with a BMI value of 30 and above are classified as obese. This indicates a high risk of diabetes.

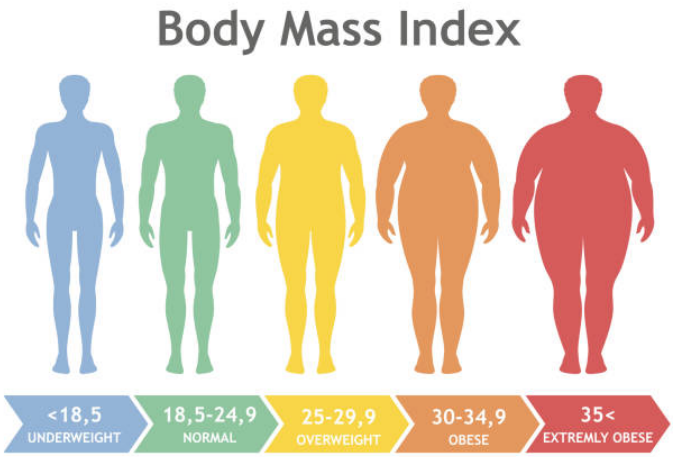

Fig. 1. BMI Categories.

The addition of these new features has led to a significant improvement in model accuracy. The glucose and BMI categories better classify the health status of individuals in the dataset and enhance the overall performance of the model.

D. Data Visualization

Various visualization techniques have been applied to better understand the analyses performed on the dataset. This process involved examining the distribution of the data and the effects of independent variables on diabetes. Two main visualization methods were used: histograms and correlation plots.

1) Histogram Visualizations: Histograms were used to understand the distribution of variables in the dataset. The following charts examined how the values of variables are distributed and whether any anomalies were detected.

Histograms, in particular, highlighted the following points:

- **Zero Values:** Zero values were found in some variables such as 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', and 'BMI'. These zero values may be due to data collection errors or could indicate that these measurements were not taken for some individuals. Therefore, analyzing zero values required either removing them from the dataset or processing them appropriately.

2) Correlation Plots: Correlation plots were used to examine the relationships between independent variables. This plot analyzed linear relationships and interactions between variables. The correlation plots revealed the following findings:

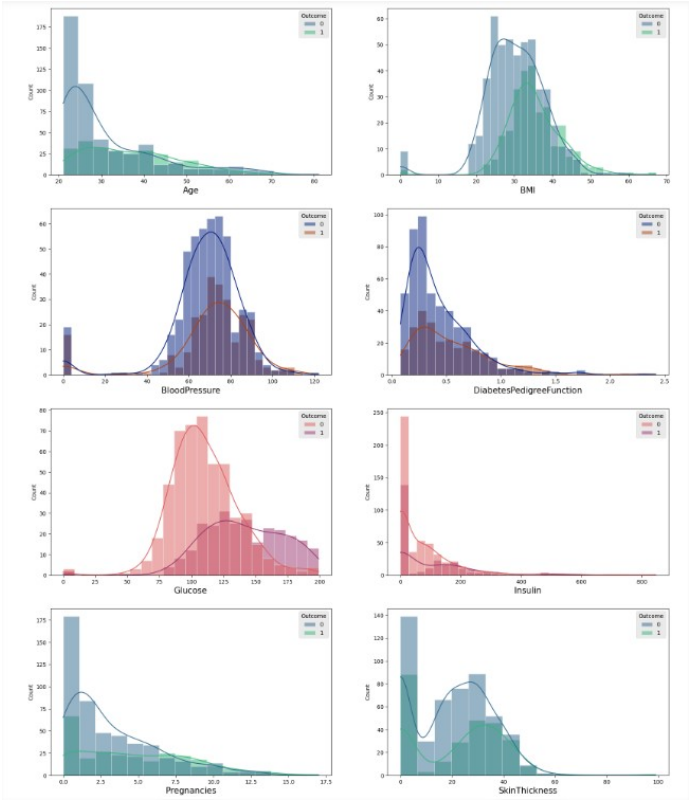

Fig. 2. Distribution of Age and its relation to Outcome.

- **Inter-Variable Relationships:** The correlation plot showed high correlations between variables such as 'Glucose', 'BMI', 'Insulin', and 'SkinThickness'. In particular, the high positive correlation between Glucose and Insulin suggests that these variables move together in determining diabetes risk.

  These visualizations helped in understanding anomalies and relationships in the dataset and contributed to guiding the data preprocessing steps. Findings about anomalies and relationships allowed for appropriate modifications in the dataset to enhance model accuracy and improve health analysis results.

E. Data Cleaning

The presence of zero values in some variables of the dataset indicates clinically impossible scenarios and suggests data collection errors. This situation can affect the accuracy of the model and reduce the reliability of analyses. Zero values were particularly found in the following variables:

- **Glucose:** Reporting a glucose level of zero generally indicates a data collection error or that measurement was not performed. Normally, a person's glucose level cannot be zero, so these values were removed from the dataset.
- **BloodPressure:** A zero blood pressure indicates a data collection error or that measurement was not performed. Clinically, blood pressure cannot be zero, so such observations were removed from the dataset.

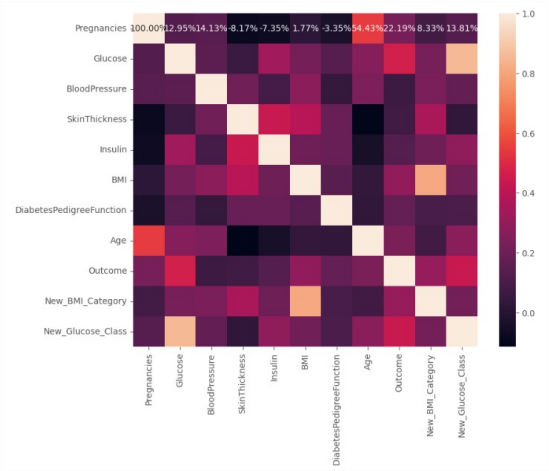

Fig. 3. Distribution of BMI and its relation to Outcome.

- **SkinThickness:** A zero skin thickness is clinically impossible and generally indicates a data collection error. Since zero skin thickness indicates incorrect data, these observations were removed from the dataset.
- **Insulin:** A zero insulin level indicates that the insulin measurement was not performed or there was a data collection error. Clinically, a zero insulin level is not normal, so these observations were removed from the dataset.
- **BMI:** A zero Body Mass Index indicates missing or incorrect height and weight measurements. Since a BMI of zero is impossible, such values were removed from the dataset.

Removing these zero values was done to improve data quality and the reliability of analyses. This step is crucial for ensuring the accuracy of the model and obtaining clinically meaningful results.

The cleaned dataset was analyzed by comparing it with both new and old datasets. These comparisons clearly demonstrated the improvements achieved through data cleaning.

III. Visualization of Variables with K-Means

The goal was to analyze in detail the relationships between variables in the model and the data of diabetic and non-diabetic individuals using the K-Means algorithm. In this analysis, various visualization techniques were employed to examine the distribution of the dataset and interactions between variables.

1) Data Distributions with K-Means: The K-Means algorithm classifies observations in the dataset into a specified number of clusters. In this context, graphs visualizing the relationships between two variables were created to better understand the distinction between diabetic and non-diabetic data.

These graphs detailed the relationship between two variables in the dataset. For example, graphs created between the 'Glucose' and 'BMI' variables show the distribution among clusters obtained using the K-Means algorithm. The graphs

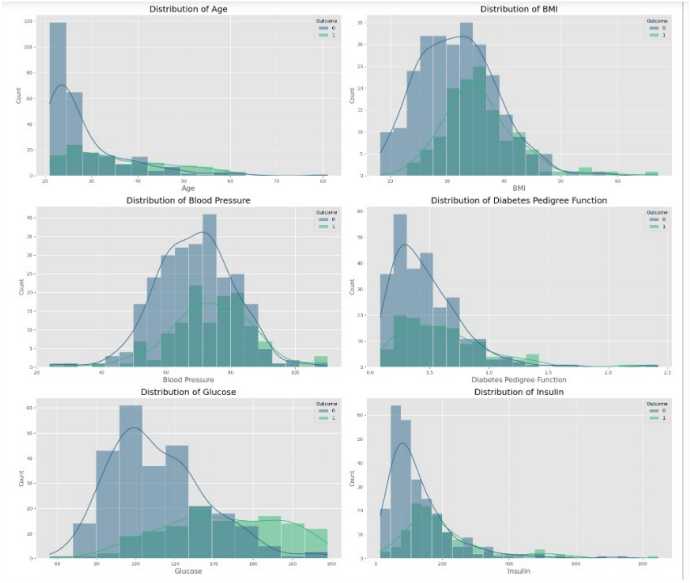

Fig. 4. Histogram plots of the cleaned dataset.

highlight the separation between diabetic and non-diabetic observations within specific variable ranges.

2) Dataset Distribution: The visualization of variables has provided a better understanding of some key features in the dataset. In particular, the distribution of diabetic and non-diabetic data across different variables has been analyzed. Below are the graphs used to show the distribution of various variables in the dataset and the differences between the two classes.

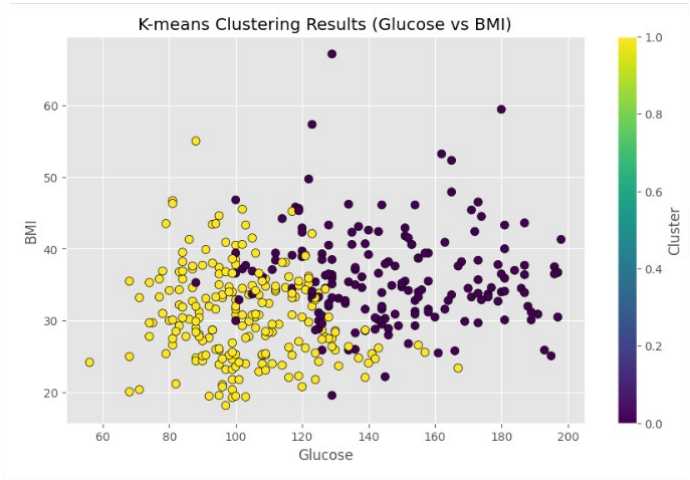

Fig. 5. Graph showing the distribution of the dataset with K-Means. Highlights the distribution differences between diabetic and non-diabetic data.

The graphs display the distributions of variables such as 'BloodPressure', 'SkinThickness', and 'Insulin'. These visualizations also reveal some anomalies and potential data cleaning requirements in the dataset. For example, zero values or abnormal distributions were observed in some variables,

indicating the need to remove or correct these values in the dataset.

Such visualizations help in better understanding the dataset during the modeling process and assist in identifying necessary steps to improve model performance.

IV. Model Development and Application Creation

Various machine learning models have been tested on the dataset. The performance of these models has been evaluated, particularly using metrics such as the F1 Score. The F1 Score is an important metric that measures the balance between precision and recall. Below is a comparison of the F1 Scores for the models used:

TABLE I
Comparison of F1 Scores

Model	F1 Score
Logistic Regression	0.69
K-Nearest Neighbors	0.54
Support Vector Machine	0.63
Gradient Boosting Classifier	0.66
XGBoost Classifier	0.62
Light GBM Classifier	0.57
Cat Boost Classifier	0.66
Decision Tree Classifier	0.60

Logistic Regression achieved the highest F1 Score of 0.69 compared to other models. This indicates that the model provides a good balance between precision and recall.

The initial choice of Logistic Regression was based on the decision to use it as the baseline model for the application. At the beginning of the project, we decided to design the application using Logistic Regression. We evaluated the performance of other models by comparing them with Logistic Regression. Based on the results obtained, Logistic Regression provided high accuracy and a good F1 Score for diabetes prediction. Therefore, Logistic Regression was chosen as the final model, and this model was used during the application development.

A. Logistic Regression

The comparisons have shown that the Logistic Regression model has the highest accuracy rate. The Logistic Regression model has demonstrated superior performance in terms of precision and accuracy for diabetes prediction compared to other models. Therefore, Logistic Regression was chosen as the final model and implemented in the application.

B. Logistic Regression Model Integration

The integration process of the Logistic Regression model involves preparing the model appropriately and integrating it with the application. This process includes the following steps:

- **Creating a Pipeline:** A pipeline containing the steps for data preprocessing, scaling, and prediction was created in the ‘logisticregression.py‘ file. The pipeline ensures that the dataset is processed according to the model’s requirements.
- **Storing the Model and Scaler:** After training the model, the trained model and the scaler used were stored in the ‘pipeline.pkl‘ file. This file allows for the reuse of the model and scaler for future predictions.
- **FastAPI Integration:** A web application created using the FastAPI framework integrates the Logistic Regression model. This integration allows users to enter data through a web interface to make predictions and presents the results in real-time.

C. Application Development

1) FastAPI Application: The FastAPI application includes the following HTML files to provide a user-friendly interface:

- **index.html:** This page allows users to enter the required data for the model. Based on the data provided by the user, the model makes predictions and displays the results to the user. The prediction results show the two conditions determined by the model: ”Diabetes” or ”No Diabetes.”
- **bmi.html:** This page allows users to enter their weight and height to calculate BMI. The BMI calculation functions compute the BMI value so that users can evaluate their health status and display the results to the user.

The deployment of the FastAPI application includes the following steps to provide a more efficient and scalable solution:

- **Local Running:** The application was run locally using the ‘uvicorn fastapi:app‘ command from the terminal. This method was used for testing the application during the development phase and was accessible on localhost:8000.
- **Using Docker:** The application was packaged as a Docker container for more effective deployment. Docker isolates the application’s dependencies and runtime environment, ensuring consistent performance across various systems.

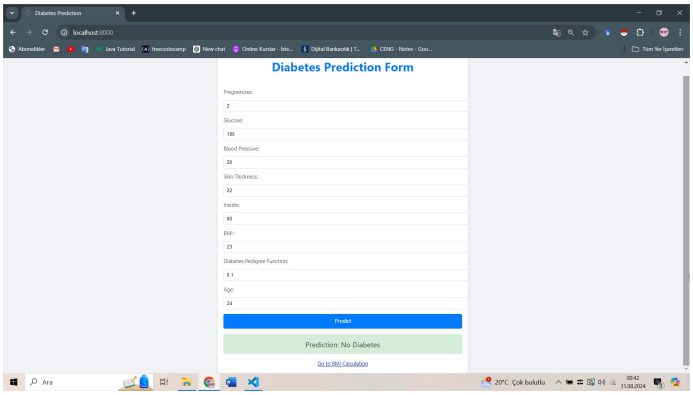

Fig. 6. This image shows the model making predictions with some random values, resulting in a No Diabetes outcome.

2) Streamlit Application: Additionally, a simpler and more user-friendly web interface has been developed using the Streamlit framework. This interface is designed solely for predicting the presence or absence of diabetes. Due to some challenges and limitations encountered during the design of

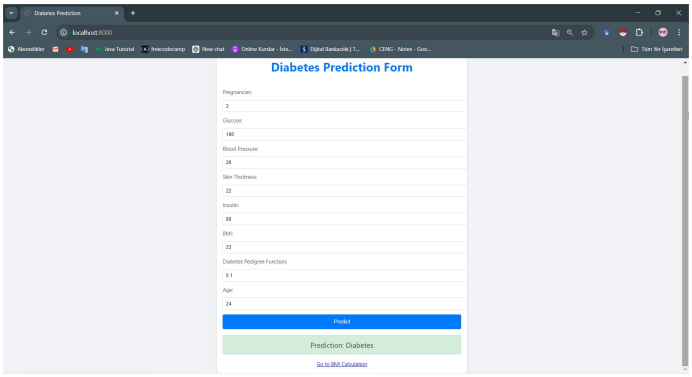

Fig. 7. This image shows the model making predictions with some random values, resulting in a Diabetes outcome.

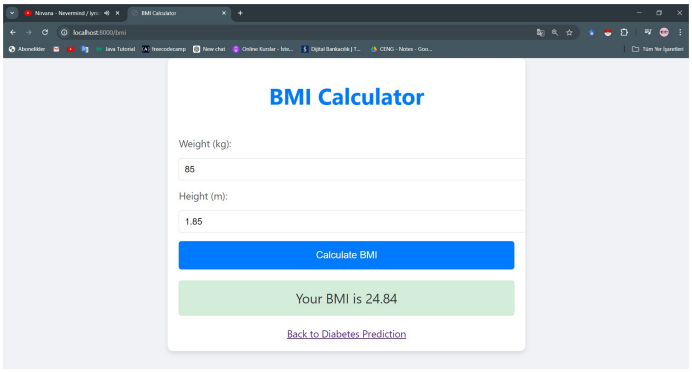

Fig. 8. This image shows BMI calculation being performed and additional functions tested.

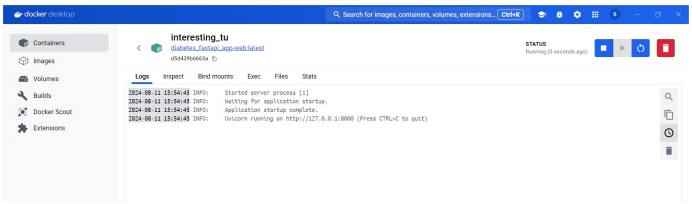

Fig. 9. This image demonstrates that FastAPI is running on Docker.

the HTML-based interface with FastAPI, an alternative interface was developed using Streamlit. Streamlit allows for the rapid and easy creation of user interfaces while providing an aesthetically pleasing experience.

In the Streamlit application, users can enter the necessary information to assess their diabetes risk. However, this application does not offer BMI calculation functionality; it only provides predictions based on the data entered by the user. This decision was made to enhance the focus of the application and keep the user experience simple and effective.

Packaging the application within a Docker container and deploying it to the AWS cloud environment has improved the performance and accessibility of the application. Docker isolates the application's dependencies and runtime environment, ensuring consistent performance across various systems. Deploying only the Streamlit application to AWS has facilitated the application's scalability and accessibility to a wider audience.

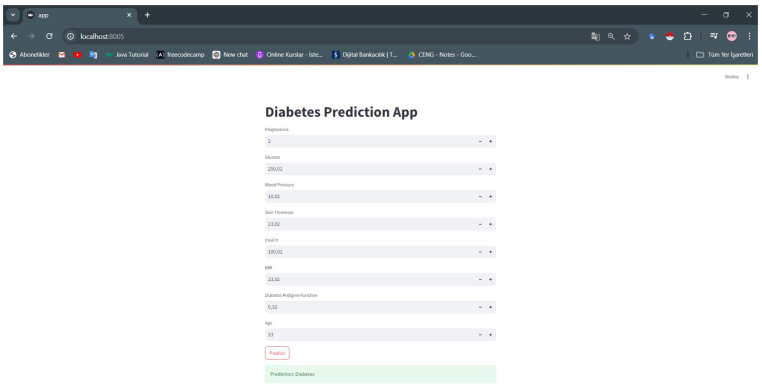

Fig. 10. Streamlit application's main page: This screen shows the application's main interface and the form fields required for users to make predictions, and the predicted result is Diabetes based on the filled form fields.

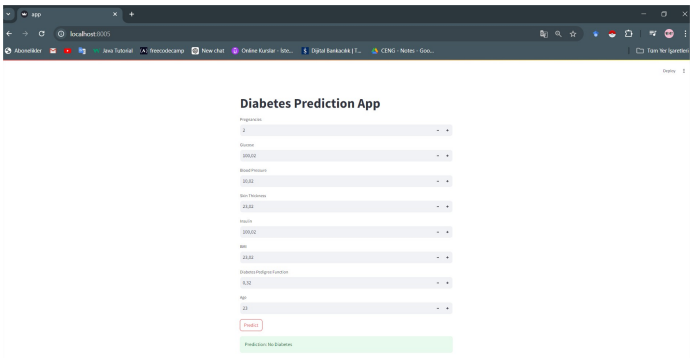

Fig. 11. This image shows the predicted result No Diabetes based on the data entered into the Streamlit application.

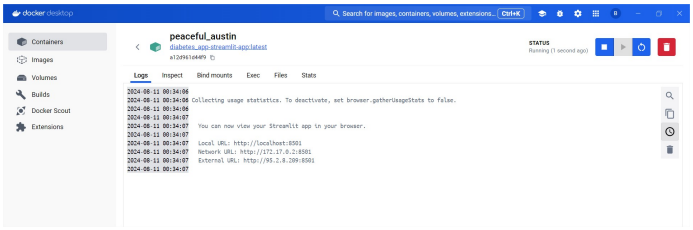

Fig. 12. This image demonstrates that the Streamlit application runs smoothly on Docker.

V. Application Deployment

A. What is Docker?

Docker is a containerization platform used to run applications in an isolated and portable environment. Docker allows applications and all their dependencies to be bundled into a single package known as a "container." This ensures that the application runs the same way regardless of the environment in which it is executed. Docker manages resources more

efficiently by using containers instead of virtual machines and enables faster application startup.

1) Basic Components of Docker:

- **Containers:** Docker containers include all the components required for an application to run: code, libraries, dependencies, and configuration files. Containers can be started quickly and are lightweight. Each container operates in its own isolated environment and is independent of other containers.
- **Docker Images:** Docker images represent a runnable version of an application. Images contain a snapshot of an application and all its dependencies. These images are used to create containers and provide version control.
- **Dockerfile:** A Dockerfile is a script file that defines how a Docker image is created. This file specifies which software to install, which configuration settings to apply, and which commands to execute within the image.
- **Docker Daemon:** The Docker daemon (dockerd) manages Docker containers and images. The daemon handles operations such as starting, stopping, and managing containers.
- **Docker CLI:** The Docker Command Line Interface (CLI) is a command-line tool used to interact with Docker. Through the CLI, you can create, run, stop, and manage containers.

2) Advantages of Docker: Docker offers several advantages in modern software development processes:

- **Portability:** Docker containers can be configured to run the same way in any environment. This ensures seamless migration of applications from developer machines to production environments. Since Docker containers include all the necessary components for running the application, compatibility issues are minimized.
- **Lightweight:** Docker containers consume far fewer resources compared to virtual machines. While virtual machines run separate operating systems, Docker containers run isolated within the same operating system. This allows for running more applications on the same hardware.
- **Consistency:** Docker isolates applications and their dependencies, eliminating inconsistencies that might arise from environmental differences. Using the same Docker image across development, testing, and production environments ensures that the application behaves consistently in every environment.
- **Fast Deployment:** Docker containers can be started and stopped quickly, accelerating development and testing processes. The rapid startup of containers facilitates the fast deployment of new features and updates.
- **Efficiency:** Docker uses system resources more efficiently, allowing more applications to run on the same hardware. Additionally, containers can be created and scaled more rapidly.

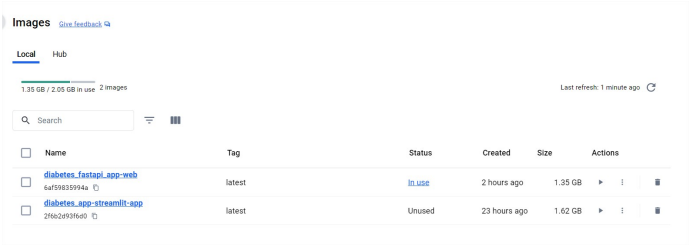

Fig. 13. This image shows that our applications are installed on Docker.

B. Application Deployment with Amazon Elastic Container Registry (ECR)

Amazon Elastic Container Registry (ECR) is a service used for securely storing and managing Docker container images. As part of the Amazon Web Services (AWS) ecosystem, ECR provides a private repository for storing Docker images and facilitates their easy use within AWS.

1) What is ECR?: Amazon Elastic Container Registry (ECR) is a managed service for storing Docker container images. ECR offers the following features:

- **Security:** ECR integrates with Amazon VPC, AWS Identity and Access Management (IAM), and AWS Key Management Service (KMS) to ensure the security of your images. Data in ECR is stored encrypted and is accessible only to authorized users.
- **Scalability:** ECR provides a high-performance and scalable service. Your images can scale automatically based on demand and are stored in AWS data centers around the world.
- **Integration:** ECR seamlessly integrates with other AWS services. It can be easily used with services such as AWS Elastic Container Service (ECS), AWS Elastic Kubernetes Service (EKS), and AWS Lambda.

2) Deploying Streamlit Application with ECR: In this project, we developed our Streamlit application using Docker and pushed the Docker image of this application to Amazon ECR. Here are the details of this process:

- **Creating an ECR Repository:** First, we created an ECR repository using the AWS Management Console. This repository serves as a private storage for our Docker images. By specifying the repository name and access permissions, we ensured the secure storage of the application.
- **Creating the Docker Image:** We created a Docker image for our Streamlit application. We used a Dockerfile to define all dependencies and the runtime environment for the application. The Dockerfile installs the necessary software and makes the application executable.
- **Pushing the Docker Image to ECR:** To push the Docker image to ECR, we followed these steps:
  - **Logging into ECR:** We ran the ‘aws ecr get-login-password‘ command using Docker CLI to log in to AWS ECR. This command generates an authentication token that allows Docker to access ECR.

  - **Tagging the Image:** We tagged the Docker image with an appropriate tag to match the ECR repository. This indicates which ECR repository the image belongs to.
  - **Pushing the Image:** Using Docker CLI, we pushed the image to ECR with the 'docker push' command. This action uploads the image to the ECR repository and makes it available on AWS.
- **ECR and AWS Integration:** ECR facilitates the integration of Docker containers with AWS services. Our Streamlit application was deployed using this image on Docker containers running on AWS EC2 instances. ECR simplifies the management, updating, and deployment of this image.

With the features provided by ECR, we can securely store our Docker images and deploy them quickly and effectively on AWS. This process enhances application management and deployment, thereby improving performance and availability.

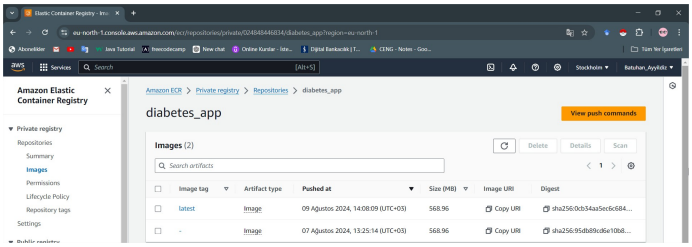

Fig. 14. This image shows that the Streamlit application has been pushed to AWS and the image is operational.

VI. Conclusion

In this study, a comprehensive solution for diabetes prediction was developed using the diabetes dataset obtained from Kaggle. Among various machine learning algorithms, the Logistic Regression model exhibited the highest performance and was thus selected as the final model. To enhance the model's success, missing values in the dataset were cleaned, new features were added, and the data was analyzed using various visualization techniques.

The developed Logistic Regression model was presented with a user-friendly interface. During this process, FastAPI was used to create API services for the model, and a dynamic web application was designed using Streamlit to facilitate user interaction with the model. Both applications were run in isolated and portable containers using Docker. With Docker, consistent operation of the application across different environments was ensured, and compatibility issues that could arise during development and production phases were minimized.

To increase the application's accessibility in the cloud environment, AWS services were utilized. Deploying the application on AWS ensured that the model reached a broad user base and provided benefits of high availability and scalability. This deployment process enabled the application to serve users seamlessly on a global scale.

In conclusion, the data analysis, preprocessing, modeling, and deployment stages conducted in this study provided an effective solution with high accuracy for diabetes prediction. The developed model and applications will contribute to improving healthcare services and supporting early diagnosis and management of diabetes patients.

References

[1] S. Meliana, T. Wahyuningrum, S. Khomsah, and S. Suyanto, "Diagnosing Diabetes with Machine Learning Techniques," *Hittite Journal of Science and Engineering*, vol. 9, no. 1, pp. 09–18, 2022. https://doi.org/10.17350/HJSE19030000250.

[2] S. Suyanto, S. Meliana, T. Wahyuningrum, and S. Khomsah, "A new nearest neighbor-based framework for diabetes detection," *Expert Systems with Applications*, vol. 199, pp. 116857, Aug. 2022. https://doi.org/10.1016/j.eswa.2022.116857.

[3] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer, 2009.

[4] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[5] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[6] M. Kuhn and K. Johnson, *Applied Predictive Modeling*. New York: Springer, 2022.

[7] J. Joubert, R. McKinney, and D. Williams, "Data-driven methods for diabetes prediction," *Journal of Biomedical Informatics*, vol. 117, pp. 103–117, Dec. 2021.

[8] S. Raschka, *Python Machine Learning*. Birmingham: Packt Publishing, 2018.

[9] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.