**Model Training Process**

**Step 1: Load Preprocessed Data**

* The Titanic dataset was preprocessed and saved as Data/preprocessed.csv.
* The cleaned dataset was loaded, and unnecessary columns like Age\_Group were removed to ensure compatibility.
* Features (X) and target labels (y) were separated.

**Step 2: Train Multiple Models**

* The dataset was split into **training (80%)** and **testing (20%)** sets.
* Several machine learning models were trained and evaluated:
  + **Logistic Regression**
  + **Decision Tree**
  + **Random Forest**
  + **K-Nearest Neighbors (KNN)**
  + **Support Vector Machine (SVM)**
  + **Gradient Boosting**
* Each model was tested, and its accuracy was recorded.

**Step 3: Model Selection**

* The model with the highest accuracy on the test data was selected.
* In your case, the best model was determined based on the accuracy scores.

**Step 4: Hyperparameter Tuning**

* **GridSearchCV** was used to fine-tune the best model’s hyperparameters.
* A cross-validation approach was applied to optimize parameters like:
  + **Number of estimators** for Random Forest/Gradient Boosting.
  + **Depth of decision trees** for Decision Tree models.
  + **Regularization strength** for Logistic Regression.
  + **Kernel type and C value** for SVM.
* The optimized model was retrained and validated.

**Step 5: Model Evaluation**

* The tuned model was tested on the test dataset.
* Metrics such as **accuracy, precision, recall, F1-score, and confusion matrix** were used to assess performance.

**Step 6: Saving the Model**

* The final trained model was saved as **best\_titanic\_model.pkl** using **Joblib**.
* This file was then used in the **inference phase** to make predictions on new data.

**Dockerization Process**

**Step 1: Creating a Docker Image**

* A **Dockerfile** was written to containerize the trained model.
* The Docker image contains:
  + A **Python environment** with the necessary dependencies.
  + The **model training script** and required files.
  + Instructions to **install dependencies** and execute the script.

**Step 2: Building the Docker Image**

* The Docker image was built from the **Dockerfile**.
* This ensures that all dependencies and configurations are packaged correctly.
* The image was tagged appropriately for version control.

**Step 3: Pushing to Docker Hub**

* The Docker image was pushed to **Docker Hub** to allow deployment in Kubernetes.
* This makes the image accessible across different environments.

**Kubernetes Deployment Process**

**Step 1: Writing the Deployment Configuration**

* A **deployment.yaml** file was created to define how the model would run inside **Kubernetes**.
* The configuration specifies:
  + The **Docker image** to use.
  + The number of **replicas** (to handle multiple requests).
  + The container **ports** for communication.

**Step 2: Deploying the Model in Kubernetes**

* The **deployment file** was applied using Kubernetes commands.
* The model container was scheduled to run in **Kubernetes pods**.

**Step 3: Exposing the Model as a Service**

* A **service.yaml** file was created to allow external access to the model.
* This enables communication between different components in the Kubernetes cluster.

**Step 4: Running and Monitoring the Model**

* Kubernetes commands were used to:
  + Check the status of running **pods**.
  + View real-time **logs** to ensure the model was functioning correctly.