Detailed Project Report (DPR)

1. Detailed Project Report (DPR)

1.1 Project Title: Mushroom Classification API

1.2 Abstract

This report details the development of a machine learning-powered REST API to classify mushrooms as either edible or poisonous. The project utilizes a Logistic Regression model trained on the UCI Mushroom dataset. The model is served via a Python-based Flask application, which is containerized using Docker for portability and deployed on the Google Cloud Run serverless platform for scalability and high availability. The resulting API provides real-time, accurate predictions based on a mushroom's physical attributes, demonstrating a complete end-to-end MLOps workflow from model training to cloud deployment.

1.3 Introduction

The primary objective of this project is to create a reliable and easily accessible tool for mushroom classification to mitigate the risks associated with consuming wild mushrooms. The project addresses the need for an automated system by building a machine learning model and deploying it as a public web service, following classical machine learning tasks from data exploration to model deployment.

1.4 Methodology

The project was executed following a structured machine learning workflow:

1. **Data Collection & Exploration:** The project used the "Mushroom" dataset from the Audubon Society Field Guide, containing 23 species of gilled mushrooms. The dataset was analyzed to understand feature distributions and relationships.
2. **Data Preprocessing:**
   - Missing values in the 'stalk-root' column, represented by '?', were imputed using the mode (the most frequent value).
   - The 'veil-type' column was dropped as it contained only a single, constant value and thus had no predictive power.
   - All categorical features were converted into a numerical format using one-hot encoding (pd.get_dummies).
   - The target variable 'class' was label-encoded ('e' -> 0, 'p' -> 1).
3. **Model Training & Evaluation:**
   - The preprocessed dataset was split into an 80% training set and a 20% testing set.
   - A Logistic Regression model from the Scikit-learn library was chosen for its simplicity and interpretability.
   - The model was trained on the training data.
   - The trained model's performance was evaluated on the test set, achieving high accuracy.
4. **Model Persistence:** The trained model object and the list of feature columns were serialized and saved to disk using the joblib library.

1.5 System Implementation

1. **API Development:** A lightweight REST API was developed using the Flask micro-framework in Python. A single /predict endpoint was created to handle POST requests containing the mushroom feature data in JSON format.
2. **Containerization:** The Flask application, along with the saved model files and all Python dependencies, was packaged into a Docker image using a Dockerfile. This ensures a consistent and reproducible runtime environment.
3. **Cloud Deployment:** The Docker image was pushed to Google Artifact Registry, a private container registry. From the registry, the image was deployed to Google Cloud Run, a serverless platform that automatically manages the infrastructure, scaling, and networking required to serve the API publicly and securely.

1.6 Results and Discussion

The final deployed API was tested using Postman. POST requests with valid JSON payloads to the public Cloud Run URL returned the correct JSON response format with a prediction and probabilities, confirming the success of the end-to-end deployment. The system demonstrated low latency and successfully translated the offline trained model into a real-time, scalable prediction service.

1.7 Conclusion and Future Scope

- **Conclusion:** This project successfully delivered a complete, end-to-end solution for mushroom classification. A machine learning model was trained, evaluated, and deployed as a robust, scalable, and publicly accessible REST API on a modern serverless platform.
- **Future Scope:**
  - **Model Improvement:** Experiment with more complex models like Gradient Boosting or Random Forests to potentially improve accuracy further.
  - **Front-End Interface:** Develop a simple web or mobile application that provides a user-friendly interface for interacting with the API.
  - **CI/CD Pipeline:** Implement a Continuous Integration/Continuous Deployment pipeline (e.g., using GitHub Actions) to automate the testing and deployment of new code or model versions.