नक्कीच\! तुमच्या "मशरूम क्लासिफिकेशन" प्रोजेक्टसाठी आवश्यक असलेले HLD (High-Level Design), LLD (Low-Level Design), आणि DPR (Detailed Project Report) खाली तयार करून दिले आहेत.

हे डॉक्युमेंट्स तुमच्या प्रोजेक्टच्या रचनेचे आणि कार्यप्रणालीचे सविस्तर वर्णन करतात.

## \*\*१. हाय-लेव्हल डिझाइन (High-Level Design - HLD)\*\*

### \*\*१.१ प्रस्तावना (Introduction)\*\*

\* \*\*उद्दिष्ट (Objective):\*\* मशरूम खाण्यायोग्य आहे की विषारी, याचे वर्गीकरण करण्यासाठी एक मशीन लर्निंग मॉडेल तयार करणे आणि ते एका वेब API द्वारे उपलब्ध करून देणे.

\* \*\*समस्या विवरण (Problem Statement):\*\* मशरूमच्या भौतिक गुणधर्मांवर आधारित, तो खाण्यासाठी सुरक्षित आहे की नाही हे ओळखणे ही मुख्य समस्या आहे. चुकीच्या ओळखीमुळे गंभीर आरोग्य धोके निर्माण होऊ शकतात, त्यामुळे एक अचूक वर्गीकरण प्रणाली आवश्यक आहे.

\* \*\*व्याप्ती (Scope):\*\* ही प्रणाली JSON स्वरूपात इनपुट स्वीकारेल आणि मशरूम खाण्यायोग्य आहे की विषारी, हे भविष्यवाणी म्हणून परत करेल. प्रणालीला वेब इंटरफेस नसेल, ती फक्त बॅकएंड API म्हणून काम करेल.

### \*\*१.२ सिस्टम आर्किटेक्चर (System Architecture)\*\*

प्रणालीची रचना खालील प्रमुख घटकांमध्ये विभागलेली आहे:

1. \*\*क्लायंट (Client):\*\* कोणताही वापरकर्ता किंवा ॲप्लिकेशन (उदा. Postman, वेब ॲप) जो मशरूमच्या गुणधर्मांसह API ला रिक्वेस्ट पाठवतो.

2. \*\*API गेटवे/लोड बॅलेंसर:\*\* या प्रोजेक्टमध्ये, \*\*Google Cloud Run\*\* हा घटक हे काम करतो. तो येणाऱ्या रिक्वेस्ट स्वीकारतो आणि योग्य कंटेनरकडे पाठवतो.

3. \*\*प्रेडिक्शन सर्व्हिस (Prediction Service):\*\*

\* \*\*Flask Web Server:\*\* हा Python-आधारित सर्वर आहे जो HTTP रिक्वेस्ट हाताळतो.

\* \*\*ML मॉडेल:\*\* `joblib` वापरून लोड केलेला, आधीच प्रशिक्षित केलेला लॉजिस्टिक रिग्रेशन मॉडेल, जो भविष्यवाणी करतो.

\* \*\*डॉकर कंटेनर (Docker Container):\*\* Flask ॲप आणि मॉडेलला एका पोर्टेबल कंटेनरमध्ये पॅकेज केले जाते, जे Cloud Run वर चालते.

4. \*\*ट्रेनिंग पाइपलाइन (Offline Training Pipeline):\*\*

\* ही एक वेगळी प्रक्रिया आहे जिथे `mushrooms.csv` डेटासेटवर डेटा क्लीनिंग, प्रीप्रोसेसिंग आणि मॉडेल ट्रेनिंग केले जाते. यातून `mushroom\_model.joblib` आणि `model\_columns.joblib` या फाइल्स तयार होतात, ज्या प्रेडिक्शन सर्व्हिसमध्ये वापरल्या जातात.

### \*\*१.३ तंत्रज्ञान स्टॅक (Technology Stack)\*\*

\* \*\*प्रोग्रामिंग भाषा:\*\* Python 3.9

\* \*\*मशीन लर्निंग:\*\* Scikit-learn, Pandas

\* \*\*API फ्रेमवर्क:\*\* Flask

\* \*\*वेब सर्वर:\*\* Gunicorn

\* \*\*कंटेनरायझेशन:\*\* Docker

\* \*\*क्लाउड प्लॅटफॉर्म:\*\* Google Cloud Run

### \*\*१.४ प्रमुख मॉड्यूल्स (Major Modules)\*\*

1. \*\*डेटा प्रीप्रोसेसिंग आणि ट्रेनिंग मॉड्यूल:\*\* डेटासेट साफ करणे, एन्कोड करणे आणि मॉडेल ट्रेन करून सेव्ह करणे.

2. \*\*API मॉड्यूल (`app.py`):\*\* रिक्वेस्ट स्वीकारणे, इनपुट डेटावर प्रक्रिया करणे, मॉडेलकडून भविष्यवाणी घेणे आणि प्रतिसाद परत पाठवणे.

-----

-----

## \*\*२. लो-लेव्हल डिझाइन (Low-Level Design - LLD)\*\*

### \*\*२.१ प्रस्तावना (Introduction)\*\*

हा विभाग HLD मध्ये नमूद केलेल्या प्रत्येक मॉड्यूलच्या तांत्रिक आणि अंमलबजावणीच्या तपशिलावर लक्ष केंद्रित करतो.

### \*\*२.२ मॉड्यूल-नुसार तपशीलवार डिझाइन\*\*

#### \*\*API मॉड्यूल (`app.py`)\*\*

\* \*\*एंडपॉइंट (Endpoint):\*\*

\* \*\*URL:\*\* `/predict`

\* \*\*HTTP मेथड:\*\* `POST`

\* \*\*रिक्वेस्ट बॉडी (Request Body - JSON Format):\*\*

```json

{

"cap-shape": "x", "cap-surface": "s", "cap-color": "n",

"bruises": "t", "odor": "p", "gill-attachment": "f",

/\* ...इतर सर्व गुणधर्म... \*/

}

```

\* \*\*रिस्पॉन्स बॉडी (Response Body - JSON Format):\*\*

```json

{

"prediction": "Poisonous",

"probability": {

"edible": 0.0157,

"poisonous": 0.9843

}

}

```

\* \*\*कार्यप्रणाली (Logic Flow):\*\*

1. ॲप्लिकेशन सुरू झाल्यावर `mushroom\_model.joblib` आणि `model\_columns.joblib` फाइल्स मेमरीमध्ये लोड केल्या जातात.

2. `/predict` एंडपॉइंटवर `POST` रिक्वेस्ट आल्यावर, JSON डेटा स्वीकारला जातो.

3. JSON डेटाला Pandas DataFrame मध्ये रूपांतरित केले जाते.

4. `pd.get\_dummies()` वापरून आलेल्या डेटाचे वन-हॉट एन्कोडिंग केले जाते.

5. `reindex()` फंक्शन वापरून, इनपुट DataFrame चे कॉलम्स मूळ ट्रेनिंग डेटाच्या कॉलम्सनुसार जुळवले जातात.

6. लोड केलेल्या मॉडेलवर `.predict()` आणि `.predict\_proba()` फंक्शन्स चालवली जातात.

7. परिणाम JSON स्वरूपात तयार करून परत पाठवला जातो.

#### \*\*Dockerfile तपशील (Dockerfile Specification)\*\*

खालील `Dockerfile` ॲप्लिकेशनचा कंटेनर तयार करण्यासाठी वापरला जातो:

```dockerfile

# Python 3.9-slim ला बेस इमेज म्हणून वापरा

FROM python:3.9-slim

# वर्किंग डिरेक्टरी सेट करा

WORKDIR /app

# आवश्यक फाइल्स कंटेनरमध्ये कॉपी करा

COPY requirements.txt .

COPY app.py .

COPY mushroom\_model.joblib .

COPY model\_columns.joblib .

# requirements.txt मधील लायब्ररी इन्स्टॉल करा

RUN pip install --no-cache-dir -r requirements.txt

# Gunicorn वापरून ॲप 8080 पोर्टवर चालवण्यासाठी कमांड

CMD ["gunicorn", "-b", "0.0.0.0:8080", "app:app"]

```

-----

-----

## \*\*३. सविस्तर प्रोजेक्ट रिपोर्ट (Detailed Project Report - DPR)\*\*

### \*\*३.१ प्रोजेक्ट शीर्षक:\*\* मशरूम क्लासिफिकेशन (Mushroom Classification)

### \*\*३.२ सारांश (Abstract)\*\*

या प्रोजेक्टमध्ये, मशरूमच्या विविध भौतिक गुणधर्मांवर आधारित तो खाण्यायोग्य आहे की विषारी, हे वर्गीकृत करण्यासाठी एक मशीन लर्निंग मॉडेल विकसित केले आहे. लॉजिस्टिक रिग्रेशन अल्गोरिदम वापरून तयार केलेले हे मॉडेल एका Flask API द्वारे उपलब्ध करून देण्यात आले आहे, जे रिअल-टाइममध्ये भविष्यवाणी करू शकते. Docker आणि Google Cloud Run वापरून हे ॲप्लिकेशन स्केलेबल आणि पोर्टेबल बनवले आहे.

### \*\*३.३ प्रस्तावना (Introduction)\*\*

मशरूम ओळखण्यात होणारी चूक जीवघेणी ठरू शकते. त्यामुळे, मानवी चुका टाळण्यासाठी आणि अचूक वर्गीकरण करण्यासाठी स्वयंचलित प्रणालीची गरज आहे. या प्रोजेक्टचे मुख्य उद्दिष्ट एक असे मॉडेल तयार करणे आहे जे उच्च अचूकतेसह मशरूमचे वर्गीकरण करू शकेल आणि ते एका सोप्या API द्वारे वापरकर्त्यांसाठी उपलब्ध असेल.

### \*\*३.४ कार्यपद्धती (Methodology)\*\*

1. \*\*डेटा संकलन:\*\* 'UCI Machine Learning Repository' वरून 'mushrooms.csv' डेटासेट वापरण्यात आला.

2. \*\*डेटा प्रीप्रोसेसिंग:\*\*

\* 'stalk-root' कॉलममधील गहाळ मूल्ये ('?') मोड (सर्वात जास्त वेळा येणारे मूल्य) वापरून भरली.

\* 'veil-type' कॉलममध्ये कोणतेही वैविध्य नसल्याने तो वगळण्यात आला.

\* सर्व कॅटेगरीकल फीचर्सचे `pd.get\_dummies()` वापरून वन-हॉट एन्कोडिंग केले गेले.

3. \*\*मॉडेल ट्रेनिंग:\*\* डेटासेटला 80% ट्रेनिंग आणि 20% टेस्टिंगमध्ये विभागले. Scikit-learn लायब्ररीमधील `LogisticRegression` मॉडेलला ट्रेनिंग डेटावर प्रशिक्षित केले.

4. \*\*मॉडेल सेव्ह करणे:\*\* प्रशिक्षित मॉडेल आणि आवश्यक कॉलमची नावे `joblib` वापरून सेव्ह केली.

### \*\*३.५ सिस्टम अंमलबजावणी (System Implementation)\*\*

1. \*\*API डेव्हलपमेंट:\*\* Python च्या Flask फ्रेमवर्कचा वापर करून `/predict` नावाचा एक `POST` एंडपॉइंट तयार केला, जो JSON डेटा स्वीकारतो आणि भविष्यवाणी परत करतो.

2. \*\*कंटेनरायझेशन:\*\* ॲप्लिकेशन आणि त्याच्या सर्व गरजा (dependencies) एका Docker कंटेनरमध्ये पॅकेज केल्या, ज्यामुळे ते कोणत्याही वातावरणात सहज चालवता येते.

3. \*\*क्लाउड डिप्लॉयमेंट:\*\* Google Cloud Run च्या सर्व्हरलेस प्लॅटफॉर्मवर हा कंटेनर डिप्लॉय केला, ज्यामुळे स्केलेबिलिटी आणि व्यवस्थापनाची चिंता राहत नाही.

### \*\*३.६ परिणाम आणि चर्चा (Results and Discussion)\*\*

स्थानिक पातळीवर आणि क्लाउडवर डिप्लॉय केल्यानंतर API यशस्वीरित्या टेस्ट केले. Postman सारख्या टूलचा वापर करून सॅम्पल डेटा पाठवल्यावर, API ने योग्य JSON प्रतिसाद दिला, ज्यात भविष्यवाणी आणि त्याची संभाव्यता (probability) समाविष्ट होती. यावरून सिद्ध होते की सिस्टम अपेक्षेप्रमाणे काम करत आहे.

### \*\*३.७ निष्कर्ष आणि भविष्यातील व्याप्ती (Conclusion & Future Scope)\*\*

\* \*\*निष्कर्ष:\*\* हा प्रोजेक्ट मशरूमचे वर्गीकरण करण्यासाठी एक कार्यक्षम, स्केलेबल आणि अचूक मशीन लर्निंग API तयार करण्यात यशस्वी झाला आहे.

\* \*\*भविष्यातील व्याप्ती:\*\*

1. अधिक अचूकतेसाठी RandomForest किंवा XGBoost सारखे प्रगत मॉडेल वापरणे.

2. एक साधा वेब-आधारित यूजर इंटरफेस (UI) तयार करणे, जिथे वापरकर्ते फॉर्म भरून भविष्यवाणी मिळवू शकतील.

3. CI/CD पाइपलाइन सेट करणे, जेणेकरून कोडमधील बदल आपोआप डिप्लॉय होतील.