Title: The Ultimate Handbook for MediAI: AI-Powered Risk Assessment & Health Advisory

Introduction

Welcome to the **MediAI** project guide! This document will take you through **every phase of the project**, from setting up the environment to processing data for an AI-powered risk assessment system.

Since you are new to AI & ML, this guide is designed to be a **detailed learning resource**. It will explain:

- **Theories** behind AI, ML, and Data Science concepts encountered at each step.
- **Code explanations** with breakdowns of each function.
- **Expected outputs** and what to observe in them.
- **Why** we perform each step and **how** it fits into the project.

By the end of this guide, you'll have: ☑ A solid understanding of AI and ML fundamentals. ☑ Hands-on experience in building an AI-based medical risk assessment system.

Phase 1: Environment Setup

1.1 What is an Environment Setup?

Before coding, we need to **set up a proper environment** that ensures smooth development. This involves:

1. Installing necessary libraries.
2. Creating a virtual environment to **isolate dependencies**.
3. Configuring Jupyter Notebook to work in **VS Code**.

1.2 Understanding AI & Machine Learning

What is Artificial Intelligence (AI)?

Artificial Intelligence refers to the simulation of human intelligence in machines, allowing them to perform tasks like reasoning, learning, decision-making, and problem-solving. AI can be classified into two types:

1. **Narrow AI** – AI systems designed for specific tasks (e.g., disease prediction models).
2. **General AI** – Hypothetical AI that can perform any intellectual task a human can do.

What is Machine Learning (ML)?

Machine Learning is a subset of AI that allows computers to learn patterns from data without being explicitly programmed. It is broadly categorized into:

1. **Supervised Learning** – The model learns from labeled data (e.g., predicting disease risk levels based on patient data).
2. **Unsupervised Learning** – The model identifies patterns in unlabeled data (e.g., clustering patients based on symptoms).
3. **Reinforcement Learning** – The model learns by interacting with an environment and receiving feedback.

This project uses **Supervised Learning** for disease risk prediction.

1.3 Install Required Packages

```
pip install pandas numpy scikit-learn matplotlib seaborn streamlit pickle5 jupyter
notebook
```

Why are these libraries needed?

- `pandas`: For handling structured data (datasets).
- `numpy`: Supports mathematical operations (arrays, calculations).
- `scikit-learn`: Machine learning models and preprocessing utilities.
- `matplotlib & seaborn`: Data visualization.
- `streamlit`: For building an interactive web application.
- `pickle5`: To save and load trained ML models.
- `jupyter notebook`: Interactive Python environment for writing and running code.

1.4 Set Up a Virtual Environment

```
python -m venv ai_med_env
source ai_med_env/bin/activate  # macOS/Linux
ai_med_env\Scripts\activate  # Windows
```

Why use a virtual environment?

- Keeps your project dependencies separate from system-wide Python packages.
- Prevents conflicts between different projects that require different versions of libraries.

1.5 Install Jupyter in VS Code

- Install the **Jupyter extension** in VS Code (Ctrl+Shift+X → Search for "Jupyter").
- Run:

```
pip install ipykernel
```

- Create a new `.ipynb` notebook in VS Code and select the appropriate Python kernel.

Expected Output:

Once Jupyter is installed and launched, you should see an interface where you can create and run Python code in cells.

Phase 2: Data Handling & Preprocessing

2.1 What is Data Preprocessing?

Raw data is often **messy** (missing values, incorrect formats, outliers). Preprocessing ensures the data is:

- **Cleaned** (missing values handled, errors corrected).
- **Normalized** (data scaled properly for ML models).
- **Structured** (columns and labels well-defined for learning).

2.2 Load the Dataset

File: `notebooks/Data_Preprocessing.ipynb`

```python
import pandas as pd
df = pd.read_csv("datasets/diabetes.csv")
print(df.head())
print(df.info())
```

Explanation:

- `pd.read_csv()` loads the dataset from a CSV file.
- `df.head()` prints the first five rows for inspection.
- `df.info()` gives details about column types and missing values.

Expected Output:

- A printed table showing the first five rows of the dataset.
- Information about columns, data types, and missing values.

2.3 Handle Missing Values

```python
print(df.isnull().sum())
df.fillna(df.mean(), inplace=True)
```

Explanation:

- `df.isnull().sum()` checks for missing values.
- `df.fillna(df.mean(), inplace=True)` fills them with column mean values.

Expected Output:

- Before: Some columns may have missing values.
- After: All missing values are replaced, and `df.isnull().sum()` should show zeros.

2.4 Detect and Handle Outliers

```python
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.boxplot(data=df)
plt.xticks(rotation=45)
plt.show()
```

Explanation:

- A **boxplot** helps visualize extreme values (outliers) in each feature.
- If outliers exist, filtering them can prevent model distortion:

```python
df = df[(df["Glucose"] > 50) & (df["Glucose"] < 250)]
```

Expected Output:

- A boxplot will appear, highlighting potential outliers.

2.5 Normalize Data

```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df.iloc[:, :-1] = scaler.fit_transform(df.iloc[:, :-1])
```

Explanation:

- `MinMaxScaler()` scales values between 0 and 1, preventing large-value bias in ML models.

Expected Output:

- All numerical columns are transformed into a common scale, improving model efficiency.

2.6 Save Preprocessed Data

```python
df.to_csv("datasets/diabetes_preprocessed.csv", index=False)
```

Explanation:

- Saves the cleaned and processed dataset for model training.

Expected Output:

- A cleaned dataset is saved in the `datasets` directory.

Phase 3: Machine Learning Model Training

3.1 What is Machine Learning Model Training?

Machine Learning (ML) models learn from past data to make predictions on new data. In this project, we use a **supervised learning model**, meaning the dataset contains labeled outputs (e.g., risk levels). The model will analyze patterns in the training data and learn how different health parameters influence the risk levels.

3.2 Splitting the Dataset into Training and Testing Sets

Before training the model, we need to **split** the dataset into two parts:

1. **Training Set (80%)** – Used by the model to learn patterns.
2. **Testing Set (20%)** – Used to evaluate the model's performance on unseen data.

File: `notebooks/Model_Training.ipynb`

```
from sklearn.model_selection import train_test_split

# Separate features (X) and target variable (y)
X = df.drop(columns=["Risk_Level"])
y = df["Risk_Level"]

# Split into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)
```

Explanation:

- `train_test_split()` randomly splits data into training and testing sets.
- `stratify=y` ensures that all risk levels are proportionally represented in both training and testing sets.
- `random_state=42` ensures **reproducibility**, meaning the same split occurs each time the code is run.

Expected Output:

- Two sets of data: `X_train, y_train` (for model training) and `X_test, y_test` (for evaluation).
- The training set should contain **80%** of the data, and the testing set should contain **20%**.

3.3 Choosing a Machine Learning Model

There are multiple models we could use for classification. Some common ones include:

- **Logistic Regression** – Simple and interpretable but may not capture complex patterns.
- **Decision Trees** – Easy to interpret but prone to overfitting.
- **Random Forest** – A collection of decision trees that improves accuracy and reduces overfitting.
- **Support Vector Machine (SVM)** – Works well for high-dimensional data.
- **Neural Networks** – More advanced but requires large datasets and computational power.

For this project, we will use **Random Forest Classifier**, as it is **robust, interpretable, and handles missing data well**.

3.4 Training the Machine Learning Model

File: `notebooks/Model_Training.ipynb`

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize the model
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)
```

Explanation:

- `RandomForestClassifier(n_estimators=100)`: Creates a Random Forest model with **100 decision trees**.
- `model.fit(X_train, y_train)`: Trains the model using the training dataset.
- `model.predict(X_test)`: Uses the trained model to make predictions on the test set.

Expected Output:

- The model is now trained and can make predictions on new data.
- No visible output yet, but the model is ready for evaluation.

3.5 Evaluating Model Performance

Once trained, we need to assess **how well the model performs** using classification metrics.

File: `notebooks/Model_Training.ipynb`

```
# Evaluate the model
print("Model Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Explanation:

- `accuracy_score(y_test, y_pred)`: Measures how many predictions were correct.

- `classification_report(y_test, y_pred)`: Provides details on **precision, recall, and F1-score** for each risk level.

Expected Output:

- **Accuracy Score:** Displays a percentage indicating how well the model performs.
- **Classification Report:** Shows precision, recall, and F1-score for each category (Low, Moderate, High risk). Example:

```
Model Accuracy: 0.87

Classification Report:
                precision    recall  f1-score   support

        Low       0.85      0.89      0.87       150
    Moderate      0.88      0.84      0.86       130
       High       0.89      0.88      0.89       120
```

- **Precision**: How many predicted positives were actually correct?
- **Recall**: How many actual positives were correctly predicted?
- **F1-score**: The balance between precision and recall.

3.6 Saving the Trained Model

To use the model later (in the web app), we need to save it.

File: `models/diabetes_risk_model.pkl`

```python
import pickle

# Save the trained model
with open("models/diabetes_risk_model.pkl", "wb") as f:
    pickle.dump(model, f)
```

Explanation:

- `pickle.dump(model, f)`: Saves the trained model as a `.pkl` file.
- Later, this model can be loaded and used for making predictions.

Expected Output:

- The trained model is saved as `diabetes_risk_model.pkl` in the `models/` directory.

Phase 4: Implementing the Recommendation System

4.1 What is a Recommendation System?

A recommendation system provides personalized advice based on the predicted **risk level** from our ML model. Instead of just predicting whether someone has a disease, this system will guide users on what actions they should take next.

4.2 Defining the Recommendation Logic

We will implement a **rule-based recommendation system**, which provides predefined health advice based on the predicted risk category.

File: `src/recommendation.py`

```
def get_recommendation(risk_level):
    if risk_level == "Low":
        return "Maintain a healthy lifestyle with regular exercise and a balanced diet."
    elif risk_level == "Moderate":
        return "Increase physical activity and monitor diet. Regular health checkups recommended."
    else:
        return "High risk! Consult a doctor immediately and follow medical advice."
```

Explanation:

- This function takes `risk_level` as input and returns a **health recommendation**.
- It follows a **simple rule-based approach** to classify **Low, Moderate, and High** risk levels.
- The recommendations are general but can be expanded with more detailed guidance.

Expected Output:

If we test the function:

```
print(get_recommendation("Moderate"))
```

It should return:

```
"Increase physical activity and monitor diet. Regular health checkups recommended."
```

4.3 Integrating the Recommendation System with the ML Model

Once the ML model predicts a **risk level**, we will use our recommendation system to provide relevant advice.

File: `notebooks/Model_Training.ipynb`

```
import pickle
from src.recommendation import get_recommendation

# Load the trained model
model = pickle.load(open("models/diabetes_risk_model.pkl", "rb"))

# Sample patient data: Age, Glucose, BMI
sample_input = [[50, 140, 28]]

# Make prediction
predicted_risk = model.predict(sample_input)[0]
recommendation = get_recommendation(predicted_risk)

print(f"Predicted Risk Level: {predicted_risk}")
print(f"Health Recommendation: {recommendation}")
```

Explanation:

- The trained **Random Forest model** is loaded using `pickle.load()`.
- A **sample patient input** is provided for prediction.
- The model **predicts the risk level** based on the input features.
- The predicted risk level is passed to `get_recommendation()` to fetch the appropriate **health advice**.

Expected Output:

```
Predicted Risk Level: High
Health Recommendation: High risk! Consult a doctor immediately and follow medical
advice.
```

4.4 Expanding the Recommendation System

The current system is rule-based, but we can improve it using:

1. **Data-driven recommendations** – Analyzing historical data to provide better insights.
2. **Personalized advice** – Taking additional factors (diet, exercise, medications) into account.
3. **Deep Learning-based systems** – Implementing NLP models for advanced recommendations.

Phase 5: Developing the Web Application

5.1 Why Build a Web Application?

A machine learning model is useful, but it becomes **much more powerful** when users can interact with it. We will build a **web-based interface** using **Streamlit**, a Python framework for creating user-friendly apps with minimal code.

5.2 Setting Up the Web Application

We will create a **simple web app** that:

- Accepts user inputs for **age, glucose, and BMI**.
- Uses our trained ML model to **predict risk level**.
- Displays a **health recommendation** based on the prediction.

5.3 Creating the Web App

File: `web_app/app.py`

```python
import streamlit as st
import pickle
from src.recommendation import get_recommendation

# Load the trained model
model = pickle.load(open("models/diabetes_risk_model.pkl", "rb"))

# Web App Title
st.title("MediAI: Disease Risk Assessment")

# Collect user input
age = st.number_input("Enter your age:", min_value=1, max_value=120, step=1)
glucose = st.number_input("Enter your glucose level:")
bmi = st.number_input("Enter your BMI:")

# Make prediction
if st.button("Check Risk Level"):
    prediction = model.predict([[age, glucose, bmi]])[0]
    recommendation = get_recommendation(prediction)
    st.write(f"### Predicted Risk Level: {prediction}")
    st.write(f"### Health Recommendation: {recommendation}")
```

Explanation:

- The **trained model** is loaded using `pickle.load()`.
- `st.number_input()` collects **user input** for **age, glucose level, and BMI**.
- When the user clicks **Check Risk Level**, the model **predicts the risk level**.
- `get_recommendation()` provides **personalized health advice**.

Expected Output:

When the app runs, it should display a **form** where users input their details. After submitting, they will see:

```
### Predicted Risk Level: Moderate
### Health Recommendation: Increase physical activity and monitor diet. Regular health checkups recommended.
```

5.4 Running the Web App

After saving `app.py`, run the following command in the terminal:

```
streamlit run web_app/app.py
```

This will launch a **local web server**, and the app will open in your browser.

5.5 Enhancing the Web Application

To improve the app, we can:

1. **Add more health parameters** – Include blood pressure, cholesterol, etc.
2. **Improve UI/UX** – Use Streamlit widgets like sliders, dropdowns, and charts.
3. **Deploy Online** – Host the app using **Render, Heroku, or AWS**.

Next Steps:

✅ Phase 1 & 2: Environment Setup & Data Preprocessing ✅ Phase 3: Machine Learning Model Training ✅ Phase 4: Implementing the Recommendation System ✅ Phase 5: Developing the Web Application 🔜 Phase 6: Deploying the Web Application