Project Title: **MedCare AI**

Team Members:

-G NITHYALAKSHMI [Team Leader]
-SUBHEKSHANA THILAGA M
-UMADEVI M
-SIVALAKSHMI P
-SELVAPRIYA K

Abstract:

Healthcare organizations, like Apollo Hospitals Enterprise Limited, are leveraging Machine Learning Operations (MLOps) to segment patients based on disease conditions and severity. This project aims to enhance healthcare planning and operations by using advanced machine learning to understand and track patient disease conditions. It enables tailored treatments, efficient resource allocation, and improved patient outcomes. The project integrates data sources, builds robust models, and establishes an MLOps pipeline for continuous model refinement. This patient-centric, data-driven approach promises more effective and efficient healthcare services.

Project Overview:

- To address the challenge of patient disease condition segmentation for enhanced healthcare planning and operations at Apollo Hospitals Enterprise Limited, our approach revolves around the application of machine learning. We begin by collecting and integrating patient data from diverse sources, including electronic health records and medical reports. This data is then meticulously preprocessed, including **cleaning** and **feature engineering**, to prepare it for **modeling**. Subsequently, we develop machine learning models that best suit the data characteristics, involving techniques like **Linear Regression**, **K-Means Clustering**, and **Machine learning models**, which are then fine-tuned for optimal performance.
- Ensuring the models remain relevant, we implement an MLOps pipeline for continuous learning and refinement, which automates the retraining process as new data becomes available. Using these trained models, we categorize patients into groups based on their disease conditions and **illness severity**. Healthcare professionals are then provided with an intuitive dashboard, delivering actionable insights from the segmented patient data, enabling informed decisions regarding patient care, resource allocation, and treatment planning. Rigorous validation and evaluation ensure that the models offer clinically meaningful insights.
- Upon successful development, the machine learning system is seamlessly deployed and integrated into the hospital's existing infrastructure, with collaboration between our team and healthcare professionals to incorporate it into their workflow. Continuous monitoring and maintenance of the system's performance and data quality is emphasized to address issues proactively and to keep the models up-to-date as patient populations and disease conditions evolve. This comprehensive approach is designed to empower
- healthcare organizations like Apollo Hospitals Enterprise Limited with a powerful tool for improving healthcare planning, resource allocation, and, most importantly, patient care and outcomes.

Technologies Used:

-Programming Language: Python

-Machine Learning Libraries: Scikit Learn,Pandas,Matplotlib
-Frameworks: Stremlit
-Tool: Google Colaboratory

Data Collection and Preprocessing:

- Data collection and preprocessing for the MIMIC-III dataset, a widely used healthcare dataset, involve several critical steps to ensure data quality and usability.
- First, data collection from the MIMIC-III database requires gaining access to the dataset, which typically involves obtaining appropriate permissions and adhering to ethical and legal guidelines due to the sensitive and private nature of patient health information. Once access is granted, relevant patient data, such as electronic health records (EHRs), admission notes, lab results, imaging reports, and clinical narratives, are extracted from the database.
- Data preprocessing plays a vital role in preparing the MIMIC-III dataset for machine learning and analysis. This phase includes several key tasks:

1. **Data Cleaning:** Raw data from EHRs often contain errors, missing values, and inconsistencies. Cleaning involves identifying and correcting these issues to ensure data accuracy.
2. **Feature Selection:** Not all variables within the dataset may be relevant to the specific research or analysis objectives. Feature selection entails identifying the most pertinent variables and discarding irrelevant ones to streamline the dataset.
3. **Encoding Categorical Variables:** Categorical variables are typically encoded to numeric values for compatibility with machine learning algorithms.
4. **Time-Series Data Transformation**: For time-series data within MIMIC-III, temporal aspects may need to be considered, including aggregating data over time intervals, handling irregular time stamps, and creating meaningful time-related features.

- The successful completion of data preprocessing for the MIMIC-III dataset is crucial in ensuring that the data is clean, structured, and suitable for various analytical and machine learning tasks in healthcare research, including disease prediction, patient risk stratification, and treatment outcome analysis. It paves the way for meaningful insights and knowledge extraction from this rich healthcare resource.

Model Architecture:

Linear regression and k-means clustering are used for patient segmentation based on disease states and severity. They can be used to predict disease severity by gathering and preprocessing patient information, such as age, lab results, vital signs, and a severity score, and establishing a linear regression model with the score as the dependent variable.

Training Process:

- The training process for healthcare patient segmentation models involves collecting and preprocessing patient data, identifying informative features, labeling, splitting the dataset, selecting appropriate machine learning models, developing and training the model, evaluating its performance using relevant metrics, validating the model, and interpreting its predictions.
- The model is then deployed into the healthcare system, ensuring it can handle real-time or batch predictions and adheres to privacy and security standards. It is then integrated into electronic health records or decision support tools.
- The model's performance is continuously monitored and maintained, and ethical considerations are addressed. Open communication between the AI team, healthcare professionals, and stakeholders is maintained to ensure successful integration into clinical practice. This iterative and collaborative process improves the quality of care provided by healthcare organizations.

Evaluation Metrics:

Accuracy

Results and Discussion:

- The Streamlit application uses a logistic regression model to classify patient disease severity, providing valuable insights for healthcare professionals.
- K-Means clustering group patients based on severity and mortality, allowing for visualization of patient distribution. The data visualizations provide insights into disease severity, mortality, and their relationship with risk levels. Further interpretation and application of these results could potentially improve healthcare management.
- These visualizations include histograms, density plots, scatter plots, box plots, and a pair plot for exploring relationships between disease severity, mortality, and risk.

Deployment:

We used Streamlit to create a user-friendly healthcare patient segmentation model. We set up a robust deployment environment, prepared all necessary files, and configured security measures. Streamlit was chosen for its simplicity and effectiveness in showcasing data visualizations, analysis, and K-Means clustering outcomes. Its interactive widgets allowed for dynamic selections and presented key findings. We monitored the application's performance and security, ensuring it aligned with compliance and data privacy standards like HIPAA.

Instructions for Running the Project:

- Environment Setup:

Ensure you have Python installed.
Install required libraries using pip install streamlit pandas scikit-learn matplotlib seaborn.

- Download the Code:

Obtain the project code and data files.

- Launch the Application:

Open a terminal.
Navigate to the project directory.

- Run the Streamlit App:

Enter streamlit run app.py in the terminal.

- Access the App:

A browser window will open, providing access to the Streamlit web application.

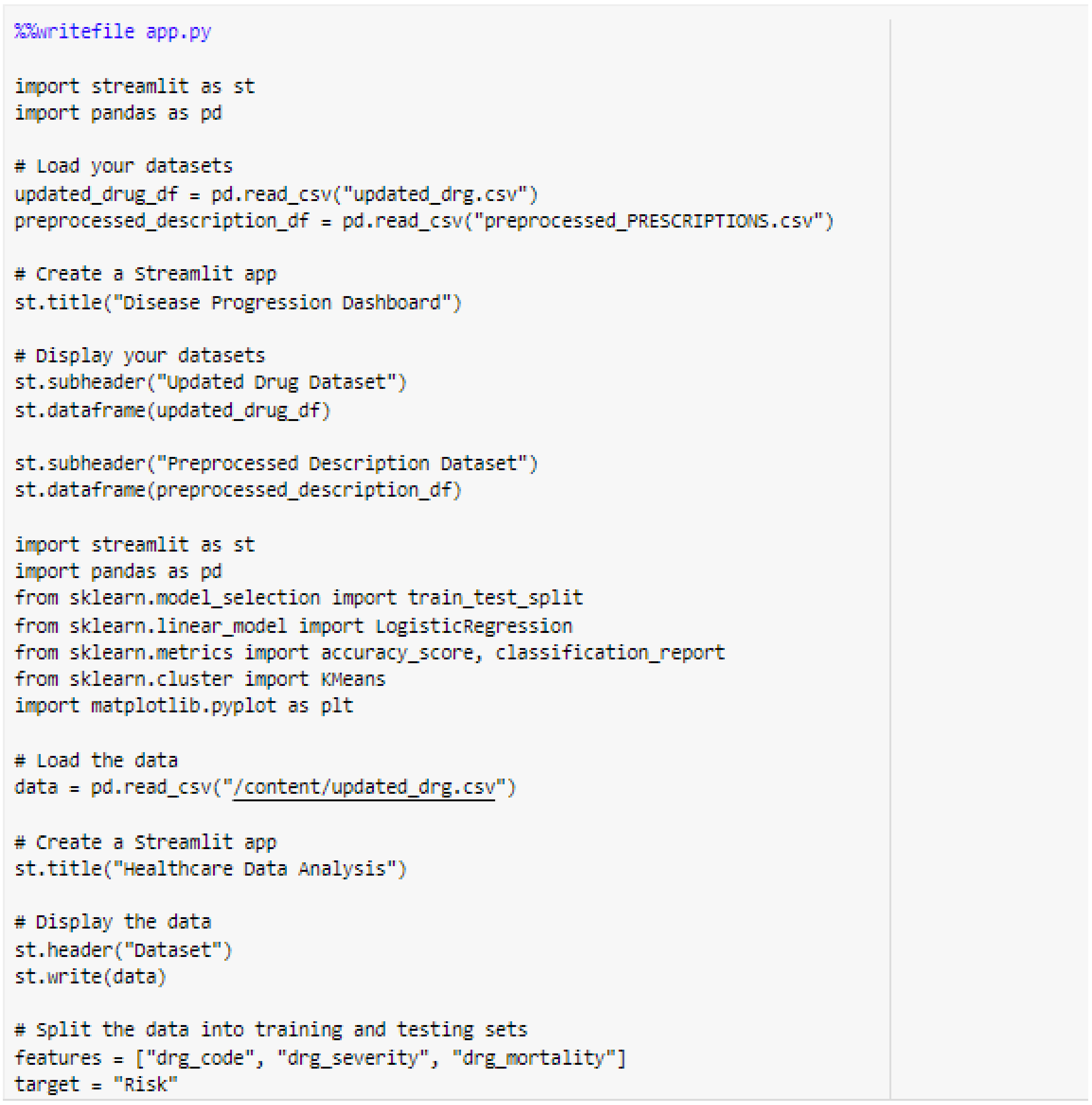
- Interact with the App:

Explore the healthcare data analysis, logistic regression results, K-Means clustering, and data visualizations using the int
interface.

- Close the App:

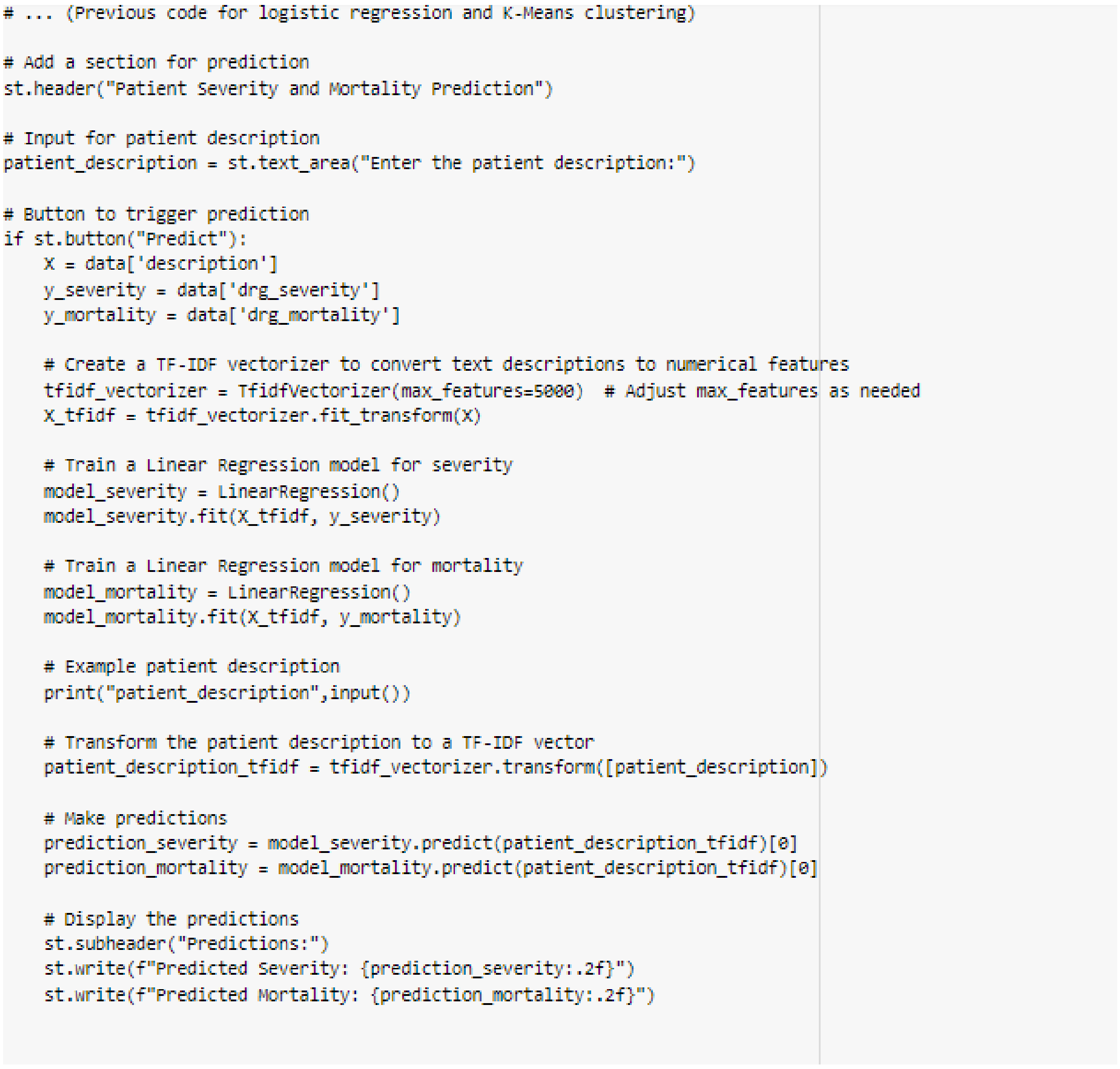
To stop the Streamlit app, return to the terminal and press Ctrl + C.

Code Snippets:



```
%%writefile app.py

import streamlit as st
import pandas as pd

# Load your datasets
updated_drug_df = pd.read_csv("updated_drg.csv")
preprocessed_description_df = pd.read_csv("preprocessed_PRESCRIPTIONS.csv")

# Create a Streamlit app
st.title("Disease Progression Dashboard")

# Display your datasets
st.subheader("Updated Drug Dataset")
st.dataframe(updated_drug_df)

st.subheader("Preprocessed Description Dataset")
st.dataframe(preprocessed_description_df)

import streamlit as st
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Load the data
data = pd.read_csv("/content/updated_drg.csv")

# Create a Streamlit app
st.title("Healthcare Data Analysis")

# Display the data
st.header("Dataset")
st.write(data)

# Split the data into training and testing sets
features = ["drg_code", "drg_severity", "drg_mortality"]
target = "Risk"
```

```
# Display the data
st.header("Dataset")
st.write(data)

# Split the data into training and testing sets
features = ["drg_code", "drg_severity", "drg_mortality"]
target = "Risk"
X_train, X_test, y_train, y_test = train_test_split(data[features], data[target], test_size=0.2, random_state=42)

# Train a logistic regression model
st.header("Logistic Regression Model")
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_report_text = classification_report(y_test, y_pred)

st.subheader("Model Evaluation")
st.write(f"Accuracy: {accuracy}")
st.write("Classification Report:")
st.code(classification_report_text)

# Select relevant features for clustering and segmentation
cluster_features = ["drg_severity", "drg_mortality"]

# Choose the number of clusters (K) based on your data and objectives
n_clusters = st.slider("Select the number of clusters", min_value=2, max_value=10, value=4)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=n_clusters)
data['cluster'] = kmeans.fit_predict(data[cluster_features])

# Visualize the clusters
st.header("K-Means Clustering for Severity of Illness")
fig, ax = plt.subplots()
scatter = ax.scatter(data["drg_severity"], data["drg_mortality"], c=data["cluster"], cmap='viridis')
ax.set_xlabel("DRG Severity")
ax.set_ylabel("DRG Mortality")
ax.set_title("K-Means Clustering for Severity of Illness")
st.pyplot(fig)
```



```
# ... (Previous code for logistic regression and K-Means clustering)

# Add a section for prediction
st.header("Patient Severity and Mortality Prediction")

# Input for patient description
patient_description = st.text_area("Enter the patient description:")

# Button to trigger prediction
if st.button("Predict"):
    X = data['description']
    y_severity = data['drg_severity']
    y_mortality = data['drg_mortality']

    # Create a TF-IDF vectorizer to convert text descriptions to numerical features
    tfidf_vectorizer = TfidfVectorizer(max_features=5000)  # Adjust max_features as needed
    X_tfidf = tfidf_vectorizer.fit_transform(X)

    # Train a Linear Regression model for severity
    model_severity = LinearRegression()
    model_severity.fit(X_tfidf, y_severity)

    # Train a Linear Regression model for mortality
    model_mortality = LinearRegression()
    model_mortality.fit(X_tfidf, y_mortality)

    # Example patient description
    print("patient_description",input())

    # Transform the patient description to a TF-IDF vector
    patient_description_tfidf = tfidf_vectorizer.transform([patient_description])

    # Make predictions
    prediction_severity = model_severity.predict(patient_description_tfidf)[0]
    prediction_mortality = model_mortality.predict(patient_description_tfidf)[0]

    # Display the predictions
    st.subheader("Predictions:")
    st.write(f"Predicted Severity: {prediction_severity:.2f}")
    st.write(f"Predicted Mortality: {prediction_mortality:.2f}")
```

Conclusion:

In conclusion, our project not only provides a valuable tool for patient risk assessment but also contributes to data-driven decision-making in healthcare. The intuitive and interactive Streamlit interface enhances the accessibility of the application for healthcare professionals and stakeholders. Moreover, the insights derived from the project can be leveraged to tailor treatments and interventions, ultimately improving patient care and healthcare management. Future work may focus on further model refinement and real-world clinical application of the findings to benefit both patients and healthcare providers.