```
# -*- coding: utf-8 -*-
"""Aditya Chikte - Unified Mentor Data Analytics Internship [Heart Disease Diagnostic Analysis].ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1mdn_0wWHwyDjV4UnXm_nDkpAp9PArn1P

# **Install Required Libraries**
"""

# Install necessary libraries including Pandas, NumPy, Matplotlib, Plotly, and Dash.
!pip install pandas numpy matplotlib plotly dash gdown

"""# **Import Required Libraries**"""

# Import necessary libraries for data loading and analysis.
import pandas as pd
import gdown

"""# **Download and Load Dataset**"""

# Google Drive link to the dataset
url = "https://drive.google.com/uc?id=1ibxofEW5YmE-rl2dHN2QT69bv96h0Jv3"

# Download the dataset
gdown.download(url, 'heart_disease_data.csv', quiet=False)

# Load dataset into a Pandas DataFrame
df = pd.read_csv('heart_disease_data.csv')

"""# **Data Preprocessing**

**1. Viewing Dataset**
"""

# Display the first few rows of the dataset
print(df.head())

"""**2. Handle Missing Values**"""

# Check for missing values
missing_values = df.isnull().sum()
print("Missing Values:\n", missing_values)

# Fill missing values with median for numerical columns
numeric_cols = df.select_dtypes(include=['number']).columns
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].median())

# Fill missing values with mode for categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns
# Check for missing values in categorical columns
if not df[categorical_cols].empty:
    # Fill missing values with mode for categorical columns
    df[categorical_cols] = df[categorical_cols].fillna(df[categorical_cols].mode().iloc[0])

"""**3. Encode Categorical Variables**"""

# Encode categorical variables into numerical representations
df = pd.get_dummies(df, columns=['sex'])

"""**4. Split Dataset**"""

# Split the dataset into features (X) and target variable (y)
X = df.drop('target', axis=1)
y = df['target']

"""# **Exploratory Data Analysis (EDA)**

**1. Calculate summary statistics for the dataset.**
"""

summary_stats = df.describe()
print(summary_stats)

"""**2. Visualize Data**"""

# Create basic visualizations using Matplotlib and Plotly.
import seaborn as sns
import matplotlib.pyplot as plt

# Correlation matrix
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Scatter plot
sns.scatterplot(data=df, x='age', y='chol', hue='target')
plt.title('Scatter Plot: Age vs. Cholesterol')
```

```
plt.show()

# Pair plot
sns.pairplot(df, hue='target')
plt.title('Pair Plot')
plt.show()

import plotly.express as px

# Box plot of age vs. target
px.box(df, x='target', y='age', title='Age vs. Target')

"""**3. Identify Correlations**"""

# Explore correlations between features using correlation matrices or heatmaps.
correlation_matrix = df.corr()

# Visualize correlation matrix using heatmap
px.imshow(correlation_matrix)

"""# **Model Building**

**1. Train Machine Learning Model**
"""

# Train a classification model using scikit-learn.
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = RandomForestClassifier()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print('Accuracy:', accuracy)

"""**2. Evaluate Additional Metrics**"""

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Assuming you have your features in X and labels in y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Logistic Regression model
lr_model = LogisticRegression(max_iter=10000)
lr_model.fit(X_train, y_train)

from sklearn.metrics import precision_score, recall_score, f1_score

# Make predictions
lr_predictions = lr_model.predict(X_test)

# Calculate precision, recall, and F1-score
precision = precision_score(y_test, lr_predictions)
recall = recall_score(y_test, lr_predictions)
f1 = f1_score(y_test, lr_predictions)

print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)

# Placeholder data and graphs
fig_age_histogram = {}  # Placeholder for age histogram
fig_age_vs_target_boxplot = {}  # Placeholder for age vs target boxplot

"""# **Dashboard Creation with Plotly Dash**

**1. Import Dash Components**
"""

# Import necessary components from Dash
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd

"""**2. Initializing the app name**"""

# Initialize the Dash app
app = dash.Dash(__name__)

"""**3. Define App Layout**"""

# Define app layout with dropdown component, histogram, and box plot
app.layout = html.Div([
    html.H1('Heart Disease Diagnostic Analysis Dashboard'),
    dcc.Dropdown(
```

```
        id='dropdown',
        options=[
            {'label': col, 'value': col} for col in df.columns
        ],
        value='age',  # Default selected value
        style={'width': '50%'}  # Adjust width as needed
    ),
    dcc.Graph(id='age-histogram'),
    dcc.Graph(id='age-vs-target-boxplot'),
    dcc.RangeSlider(
        id='age-slider',
        min=df['age'].min(),
        max=df['age'].max(),
        step=1,
        marks={i: str(i) for i in range(df['age'].min(), df['age'].max() + 1, 5)},
        value=[df['age'].min(), df['age'].max()]
    )
])

"""**4. Add Callbacks**"""

# Implement callbacks to update graphs based on user interactions
@app.callback(
    [Output('age-histogram', 'figure'),
     Output('age-vs-target-boxplot', 'figure')],
    [Input('dropdown', 'value')]
)
def update_graph(selected_feature):
    # Update graphs based on selected feature
    fig_age_histogram = px.histogram(df, x=selected_feature, title=f'{selected_feature} Distribution', hover_data=df.columns)
    fig_age_vs_target_boxplot = px.box(df, x='target', y=selected_feature, title=f'{selected_feature} vs. Target', hover_data=df.columns)
    return fig_age_histogram, fig_age_vs_target_boxplot

"""# **Run Dashboard and Test**

**1. Run Dashboard**
"""

# Run the Dash app within Google Colab notebook
if __name__ == '__main__':
    app.run_server(debug=True, mode="inline")
```