1. Train, Evaluate, Predict Preterm Delivery Using Gradient Boosting, Process IoT Sensor Data, and Save Predictions to CSV

```
import pandas as pd
import random
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import joblib
import os

# Function to train and save the Gradient Boosting Model
def train_and_save_model(dataset_path, model_file):
    # Load the dataset
    data = pd.read_csv(dataset_path)

    # Define features and target variable
    features = [
        'Age_Of_Mother', 'weight_before_preg', 'wt_before_delivery',
        'Height(cm)', 'BMI', 'Hemoglobin', 'PCOS', 'Heartbeat_Rate',
        'Motion_of_Baby', 'Stress_Level'
    ]
    target = 'Preterm_Delivery'

    # Clean dataset
    data_cleaned = data.dropna(subset=[target])

    # Prepare features (X) and target (y)
    X = data_cleaned[features]
    y = data_cleaned[target]

    # Split into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Train the model
    gb_model = GradientBoostingClassifier(random_state=42)
    gb_model.fit(X_train, y_train)

    # Save the model
```

```
    joblib.dump(gb_model, model_file)
    print(f"Model trained and saved as '{model_file}'")

    # Evaluate the model
    y_pred = gb_model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Model Accuracy: {accuracy * 100:.2f}%")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

# Function to clean and process sensor files
def process_sensor_files(gsr_file, pulse_file, motion_file):
    """
    Reads and processes sensor data from three separate CSV files.
    Returns the best value for each sensor type after cleaning.
    """
    try:
        # Process Pulse (BPM) and clean any unwanted text like "BPM"
        pulse_data = pd.read_csv(pulse_file)
        pulse_data['BPM'] = pd.to_numeric(pulse_data['BPM'].replace(r'\D',
'', regex=True), errors='coerce')  # Remove non-numeric characters
        pulse_values = pulse_data['BPM'].dropna()

        if len(pulse_values) == 0:
            raise ValueError("No valid pulse data available.")

        # Format pulse values to retain full precision
        pulse_value_avg = pulse_values.mean()  # Average BPM with full
precision
        pulse_value_median = pulse_values.median()  # Median of BPM

        # Display pulse values with full precision for debugging
        print("Pulse values (with full precision):")
        print(pulse_values.head())


        # Process GSR and ensure it's numeric
        gsr_data = pd.read_csv(gsr_file)
        gsr_data['Voltage (V)'] = pd.to_numeric(gsr_data['Voltage (V)'],
errors='coerce')  # Force numeric conversion
```

```
        gsr_values = gsr_data['Voltage (V)'].dropna()

        if len(gsr_values) == 0:
            raise ValueError("No valid GSR data available.")

        gsr_value_avg = gsr_values.mean()  # Average GSR value
        gsr_value_mode = gsr_values.mode()[0]  # Mode for most frequent

        # Process Motion (Yes/No) and calculate the mode
        motion_data = pd.read_csv(motion_file)
        motion_data['Motion Detected'] = motion_data['Motion
Detected'].str.lower()
        motion_values = motion_data['Motion Detected'].dropna()

        if len(motion_values) == 0:
            raise ValueError("No valid motion data available.")

        motion_value = motion_values.mode()[0]  # Most frequent value
(Yes/No)
        motion_percentage = (motion_values == 'yes').mean() * 100  #
Percentage of "Yes"

        return {
            "Pulse": pulse_value_avg,  # You can replace with
pulse_value_median if you prefer
            "GSR": gsr_value_avg,      # Or use gsr_value_mode if you
prefer mode
            "Motion": motion_value,    # Most frequent motion value
(Yes/No)
            "Motion_Percentage": motion_percentage
        }

    except Exception as e:
        print(f"Error processing sensor files: {e}")
        return {"Pulse": None, "GSR": None, "Motion": None,
"Motion_Percentage": None}

# Function to make predictions
def predict_with_sensor_data(model, sensor_data):
    # Set default values for the other features
```

```
    default_values = {
        'Age_Of_Mother': 25,
        'weight_before_preg': 60,
        'wt_before_delivery': 70,
        'Height(cm)': 160,
        'BMI': 22.5,
        'Hemoglobin': 12,
        'PCOS': 0
    }

    # Combine sensor data with default values
    feature_values = default_values.copy()
    feature_values['Heartbeat_Rate'] = sensor_data['Pulse']
    feature_values['Motion_of_Baby'] = sensor_data['Motion']
    feature_values['Stress_Level'] = sensor_data['GSR']

    # Convert to DataFrame
    X_new = pd.DataFrame([feature_values])

    # Make prediction
    prediction = model.predict(X_new)[0]
    prediction_proba = model.predict_proba(X_new)[0][1] * 100  #
Probability of "Preterm"

    return prediction, prediction_proba

# Main execution
if __name__ == "__main__":
    # File paths
    dataset_path = "modified_dataset.csv"  # Replace with your dataset
file
    model_file = "gradient_boosting_model.pkl"
    gsr_file = "gsr_data.csv"
    pulse_file = "pulse_data.csv"
    motion_file = "motion_data.csv"

    # Train and save the model if not already saved
    if not os.path.exists(model_file):
        print("Model file not found. Training and saving the model...")
        train_and_save_model(dataset_path, model_file)
```

```
    # Load the model
    try:
        gb_model = joblib.load(model_file)
        print("Model loaded successfully.")
    except Exception as e:
        print(f"Error loading the model: {e}")
        exit()

    # Process sensor files and compute best values
    sensor_data = process_sensor_files(gsr_file, pulse_file, motion_file)
    print(f"Processed Sensor Values: {sensor_data}")

    # Make predictions with sensor data
    if None not in sensor_data.values():
        prediction, prediction_proba = predict_with_sensor_data(gb_model,
sensor_data)
        print(f"Prediction: {'Preterm' if prediction == 1 else 'Not
Preterm'}")
        print(f"Prediction Probability: {prediction_proba:.2f}%")

        # Prepare final output with additional columns
        prediction_data = {
            'Age_Of_Mother': 25,  # Replace with actual values or inputs
            'weight_before_preg': 60,  # Replace with actual values or
inputs
            'wt_before_delivery': 70,  # Replace with actual values or
inputs
            'Height(cm)': 160,  # Replace with actual values or inputs
            'BMI': 22.5,  # Replace with actual values or inputs
            'Hemoglobin': 12,  # Replace with actual values or inputs
            'PCOS': 0,  # Replace with actual values or inputs
            'Heartbeat_Rate': sensor_data['Pulse'],
            'Motion_of_Baby': sensor_data['Motion'],
            'Stress_Level': sensor_data['GSR'],
            'Preterm_Prediction': 'Preterm' if prediction == 1 else 'Not
Preterm',
            'Prediction_Probability': f"{prediction_proba:.8f}"  # Format
probability
        }
```

```
        # Save the prediction to a new CSV file with all columns
        prediction_df = pd.DataFrame([prediction_data])
        prediction_df.to_csv("predictions_database.csv", mode='a',
header=not os.path.exists("predictions_database.csv"), index=False)
        print("Prediction saved to 'predictions_database.csv'")
    else:
        print("Invalid or incomplete sensor data. Please check the input
files.")
```

Output :

```
Model file not found. Training and saving the model...
Model trained and saved as 'gradient_boosting_model.pkl'
Model Accuracy: 95.56%

Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      0.91      0.94        33
         1.0       0.95      0.98      0.97        57

    accuracy                           0.96        90
   macro avg       0.96      0.95      0.95        90
weighted avg       0.96      0.96      0.96        90

Model loaded successfully.
Pulse values (with full precision):
0     16.871601
1     13.308565
2     33.771823
3    101.090829
4      4.440879
Name: BPM, dtype: float64
Processed Sensor Values: {'Pulse': 38.97586230677411, 'GSR':
1.6685824771689497, 'Motion': '1', 'Motion_Percentage': 4.375}
Prediction: Not Preterm
Prediction Probability: 0.93%
Prediction saved to 'predictions_database.csv'
```

2. Manual Input Prediction :

2.1 Preterm Example :

```
import pandas as pd
import joblib

# Function to load the trained model
def load_model(model_file):
    try:
        gb_model = joblib.load(model_file)
        print("Model loaded successfully.")
        return gb_model
    except Exception as e:
        print(f"Error loading the model: {e}")
        return None

# Function to take user input for sensor data
def get_sensor_input():
    try:
        # Get GSR, pulse, and motion data from user input
        pulse = float(input("Enter Pulse (BPM): "))
        gsr = float(input("Enter GSR (Voltage in V): "))
        motion = input("Enter Motion ('yes' or 'no'): ").strip().lower()

        if motion not in ['yes', 'no']:
            raise ValueError("Motion value must be 'yes' or 'no'.")

        # Convert motion to numeric: 'yes' = 1, 'no' = 0
        motion_numeric = 1 if motion == 'yes' else 0

        return {
            "Pulse": pulse,
            "GSR": gsr,
            "Motion": motion_numeric
        }

    except ValueError as e:
        print(f"Invalid input: {e}")
        return None
```

```
# Function to make predictions with user input
def predict_with_user_input(model, sensor_data):
    # Set default values for the other features
    default_values = {
        'Age_Of_Mother': 25,
        'weight_before_preg': 60,
        'wt_before_delivery': 70,
        'Height(cm)': 160,
        'BMI': 22.5,
        'Hemoglobin': 12,
        'PCOS': 0
    }

    # Combine sensor data with default values
    feature_values = default_values.copy()
    feature_values['Heartbeat_Rate'] = sensor_data['Pulse']
    feature_values['Motion_of_Baby'] = sensor_data['Motion']
    feature_values['Stress_Level'] = sensor_data['GSR']

    # Convert to DataFrame
    X_new = pd.DataFrame([feature_values])

    # Make prediction
    prediction = model.predict(X_new)[0]
    prediction_proba = model.predict_proba(X_new)[0][1] * 100  #
Probability of "Preterm"

    return prediction, prediction_proba

# Main execution
if __name__ == "__main__":
    model_file = "gradient_boosting_model.pkl"  # Replace with the path to
your model file

    # Load the model
    model = load_model(model_file)

    if model:
        # Take input from the user for sensor data
```

```
        sensor_data = get_sensor_input()

        if sensor_data:
            # Make predictions with the provided sensor data
            prediction, prediction_proba = predict_with_user_input(model,
sensor_data)
            print(f"Prediction: {'Preterm' if prediction == 1 else 'Not
Preterm'}")
            print(f"Prediction Probability: {prediction_proba:.2f}%")
        else:
            print("Invalid input. Please enter valid sensor data.")
```

Output :

```
Model loaded successfully.
Enter Pulse (BPM): 110
Enter GSR (Voltage in V): 0.8
Enter Motion ('yes' or 'no'): yes
Prediction: Preterm
Prediction Probability: 99.26%
```

2.2 Not Preterm Example :

```
import pandas as pd
import joblib

# Function to load the trained model
def load_model(model_file):
    try:
        gb_model = joblib.load(model_file)
        print("Model loaded successfully.")
        return gb_model
    except Exception as e:
        print(f"Error loading the model: {e}")
        return None

# Function to take user input for sensor data
def get_sensor_input():
    try:
        # Get GSR, pulse, and motion data from user input
        pulse = float(input("Enter Pulse (BPM): "))
```

```
        gsr = float(input("Enter GSR (Voltage in V): "))
        motion = input("Enter Motion ('yes' or 'no'): ").strip().lower()

        if motion not in ['yes', 'no']:
            raise ValueError("Motion value must be 'yes' or 'no'.")

        # Convert motion to numeric: 'yes' = 1, 'no' = 0
        motion_numeric = 1 if motion == 'yes' else 0

        return {
            "Pulse": pulse,
            "GSR": gsr,
            "Motion": motion_numeric
        }

    except ValueError as e:
        print(f"Invalid input: {e}")
        return None

# Function to make predictions with user input
def predict_with_user_input(model, sensor_data):
    # Set default values for the other features
    default_values = {
        'Age_Of_Mother': 25,
        'weight_before_preg': 60,
        'wt_before_delivery': 70,
        'Height(cm)': 160,
        'BMI': 22.5,
        'Hemoglobin': 12,
        'PCOS': 0
    }

    # Combine sensor data with default values
    feature_values = default_values.copy()
    feature_values['Heartbeat_Rate'] = sensor_data['Pulse']
    feature_values['Motion_of_Baby'] = sensor_data['Motion']
    feature_values['Stress_Level'] = sensor_data['GSR']

    # Convert to DataFrame
    X_new = pd.DataFrame([feature_values])
```

```
    # Make prediction
    prediction = model.predict(X_new)[0]
    prediction_proba = model.predict_proba(X_new)[0][1] * 100  #
Probability of "Preterm"

    return prediction, prediction_proba

# Main execution
if __name__ == "__main__":
    model_file = "gradient_boosting_model.pkl"  # Replace with the path to
your model file

    # Load the model
    model = load_model(model_file)

    if model:
        # Take input from the user for sensor data
        sensor_data = get_sensor_input()

        if sensor_data:
            # Make predictions with the provided sensor data
            prediction, prediction_proba = predict_with_user_input(model,
sensor_data)
            print(f"Prediction: {'Preterm' if prediction == 1 else 'Not
Preterm'}")
            print(f"Prediction Probability: {prediction_proba:.2f}%")
        else:
            print("Invalid input. Please enter valid sensor data.")
```

Output :

```
Model loaded successfully.
Enter Pulse (BPM): 75
Enter GSR (Voltage in V): 0.3
Enter Motion ('yes' or 'no'): no
Prediction: Not Preterm
Prediction Probability: 0.02%
```