Project 1 Task 3

Hrishikesh Deshpande (hd11) - Task 3.2

Karthik Appana (kappana2) - Task 3.0

Siddharth Gummadapu (sg96) - Task 3.1

In [ ]:
```
# imports
import numpy as np
import scipy.io as sio
import scipy.stats as stat
import matplotlib.pyplot as plt
```

Patient 1

In [3]:
```
# load data from mat file
pat1 = sio.loadmat('data_patients/patient1.mat')

pat1Floor = np.floor(pat1['all_data'])

pat1Cutoff = int(len(pat1['all_data'][0]) * (2/3))

pat1TrainData = []
for i in range(7):
    pat1TrainData.append(pat1Floor[i][:pat1Cutoff])
pat1TrainLabels = pat1['all_labels'][0][:pat1Cutoff]

pat1TestData = []
for i in range(7):
    pat1TestData.append(pat1Floor[i][pat1Cutoff:])
pat1TestLabels = pat1['all_labels'][0][pat1Cutoff:]
```

Patient 2

In [4]:
```
# load data from mat file
pat2 = sio.loadmat('data_patients/patient2.mat')
```

```
pat2Floor = np.floor(pat2['all_data'])

pat2Cutoff = int(len(pat2['all_data'][0]) * (2/3))

pat2TrainData = []
for i in range(7):
    pat2TrainData.append(pat2Floor[i][:pat2Cutoff])
pat2TrainLabels = pat2['all_labels'][0][:pat2Cutoff]

pat2TestData = []
for i in range(7):
    pat2TestData.append(pat2Floor[i][pat2Cutoff:])
pat2TestLabels = pat2['all_labels'][0][pat2Cutoff:]
```

Patient 3

In [5]:
```
# load data from mat file
pat3 = sio.loadmat('data_patients/patient3.mat')

pat3Floor = np.floor(pat3['all_data'])

pat3Cutoff = int(len(pat3['all_data'][0]) * (2/3))

pat3TrainData = []
for i in range(7):
    pat3TrainData.append(pat3Floor[i][:pat3Cutoff])
pat3TrainLabels = pat3['all_labels'][0][:pat3Cutoff]

pat3TestData = []
for i in range(7):
    pat3TestData.append(pat3Floor[i][pat3Cutoff:])
pat3TestLabels = pat3['all_labels'][0][pat3Cutoff:]
```

Patient 4

In [6]:
```
pat4 = sio.loadmat('data_patients/patient4.mat')

pat4Floor = np.floor(pat4['all_data'])

pat4Cutoff = int(len(pat4['all_data'][0]) * (2/3))

pat4TrainData = []
```

```
for i in range(7):
    pat4TrainData.append(pat4Floor[i][:pat4Cutoff])
pat4TrainLabels = pat4['all_labels'][0][:pat4Cutoff]

pat4TestData = []
for i in range(7):
    pat4TestData.append(pat4Floor[i][pat4Cutoff:])
pat4TestLabels = pat4['all_labels'][0][pat4Cutoff:]
```

Patient 5

In [7]:
```
pat5 = sio.loadmat('data_patients/patient5.mat')

pat5Floor = np.floor(pat5['all_data'])

pat5Cutoff = int(len(pat5['all_data'][0]) * (2/3))

pat5TrainData = []
for i in range(7):
    pat5TrainData.append(pat5Floor[i][:pat5Cutoff])
pat5TrainLabels = pat5['all_labels'][0][:pat5Cutoff]

pat5TestData = []
for i in range(7):
    pat5TestData.append(pat5Floor[i][pat5Cutoff:])
pat5TestLabels = pat5['all_labels'][0][pat5Cutoff:]
```

Patient 6

In [8]:
```
pat6 = sio.loadmat('data_patients/patient6.mat')

pat6Floor = np.floor(pat6['all_data'])

pat6Cutoff = int(len(pat6['all_data'][0]) * (2/3))

pat6TrainData = []
for i in range(7):
    pat6TrainData.append(pat6Floor[i][:pat6Cutoff])
pat6TrainLabels = pat6['all_labels'][0][:pat6Cutoff]

pat6TestData = []
for i in range(7):
```

```
    pat6TestData.append(pat6Floor[i][pat6Cutoff:])
pat6TestLabels = pat6['all_labels'][0][pat6Cutoff:]
```

Patient 7

In [9]:
```
pat7 = sio.loadmat('data_patients/patient7.mat')

pat7Floor = np.floor(pat7['all_data'])

pat7Cutoff = int(len(pat7['all_data'][0]) * (2/3))

pat7TrainData = []
for i in range(7):
    pat7TrainData.append(pat7Floor[i][:pat7Cutoff])
pat7TrainLabels = pat7['all_labels'][0][:pat7Cutoff]

pat7TestData = []
for i in range(7):
    pat7TestData.append(pat7Floor[i][pat7Cutoff:])
pat7TestLabels = pat7['all_labels'][0][pat7Cutoff:]
```

Patient 8

In [10]:
```
pat8 = sio.loadmat('data_patients/patient8.mat')

pat8Floor = np.floor(pat8['all_data'])

pat8Cutoff = int(len(pat8['all_data'][0]) * (2/3))

pat8TrainData = []
for i in range(7):
    pat8TrainData.append(pat8Floor[i][:pat8Cutoff])
pat8TrainLabels = pat8['all_labels'][0][:pat8Cutoff]

pat8TestData = []
for i in range(7):
    pat8TestData.append(pat8Floor[i][pat8Cutoff:])
pat8TestLabels = pat8['all_labels'][0][pat8Cutoff:]
```

Patient 9

In [11]:
```
pat9 = sio.loadmat('data_patients/patient9.mat')

pat9Floor = np.floor(pat9['all_data'])

pat9Cutoff = int(len(pat9['all_data'][0]) * (2/3))

pat9TrainData = []
for i in range(7):
    pat9TrainData.append(pat9Floor[i][:pat9Cutoff])
pat9TrainLabels = pat9['all_labels'][0][:pat9Cutoff]

pat9TestData = []
for i in range(7):
    pat9TestData.append(pat9Floor[i][pat9Cutoff:])
pat9TestLabels = pat9['all_labels'][0][pat9Cutoff:]
```

Task 3.1a - Prior Probabilities

In [12]:
```
def getPriorVals(trainLabels):
    priorH0 = list(trainLabels).count(0) / len(trainLabels)
    priorH1 = list(trainLabels).count(1) / len(trainLabels)

    return round(priorH0, 2), round(priorH1, 2)
```

In [13]:
```
pat1PriorH0, pat1PriorH1 = getPriorVals(pat1TrainLabels)
pat2PriorH0, pat2PriorH1 = getPriorVals(pat2TrainLabels)
pat3PriorH0, pat3PriorH1 = getPriorVals(pat3TrainLabels)
pat4PriorH0, pat4PriorH1 = getPriorVals(pat4TrainLabels)
pat5PriorH0, pat5PriorH1 = getPriorVals(pat5TrainLabels)
pat6PriorH0, pat6PriorH1 = getPriorVals(pat6TrainLabels)
pat7PriorH0, pat7PriorH1 = getPriorVals(pat7TrainLabels)
pat8PriorH0, pat8PriorH1 = getPriorVals(pat8TrainLabels)
pat9PriorH0, pat9PriorH1 = getPriorVals(pat9TrainLabels)
```

Task 3.1b - Calculating Likelihood Matrices

In [14]:
```
def likelihoodMatrix(trainData, trainLabels):
    likelihoodMatrices = []

    for featureIdx in range(7):
        featureData = trainData[featureIdx]
        labels = trainLabels
```

```
        minVal = int(np.min(featureData))
        maxVal = int(np.max(featureData))
        possibleValues = range(minVal, maxVal + 1)

        countMatrix = np.zeros((2, len(possibleValues)))

        for i in range(len(featureData)):
            classIdx = int(labels[i])
            valueIdx = int(featureData[i]) - minVal
            countMatrix[classIdx][valueIdx] += 1

        h0Total = np.sum(countMatrix[0])
        h1Total = np.sum(countMatrix[1])

        likelihoodMatrix = np.zeros((2, len(possibleValues)))

        for valIdx in range(len(possibleValues)):
            likelihoodMatrix[0][valIdx] = countMatrix[0][valIdx] / h0Total if h0Total > 0 else 0
            likelihoodMatrix[1][valIdx] = countMatrix[1][valIdx] / h1Total if h1Total > 0 else 0

        likelihoodMatrices.append(likelihoodMatrix)

    return likelihoodMatrices
```

In [15]:
```
pat1LM = likelihoodMatrix(pat1TrainData, pat1TrainLabels)
pat2LM = likelihoodMatrix(pat2TrainData, pat2TrainLabels)
pat3LM = likelihoodMatrix(pat3TrainData, pat3TrainLabels)
pat4LM = likelihoodMatrix(pat4TrainData, pat4TrainLabels)
pat5LM = likelihoodMatrix(pat5TrainData, pat5TrainLabels)
pat6LM = likelihoodMatrix(pat6TrainData, pat6TrainLabels)
pat7LM = likelihoodMatrix(pat7TrainData, pat7TrainLabels)
pat8LM = likelihoodMatrix(pat8TrainData, pat8TrainLabels)
pat9LM = likelihoodMatrix(pat9TrainData, pat9TrainLabels)
```

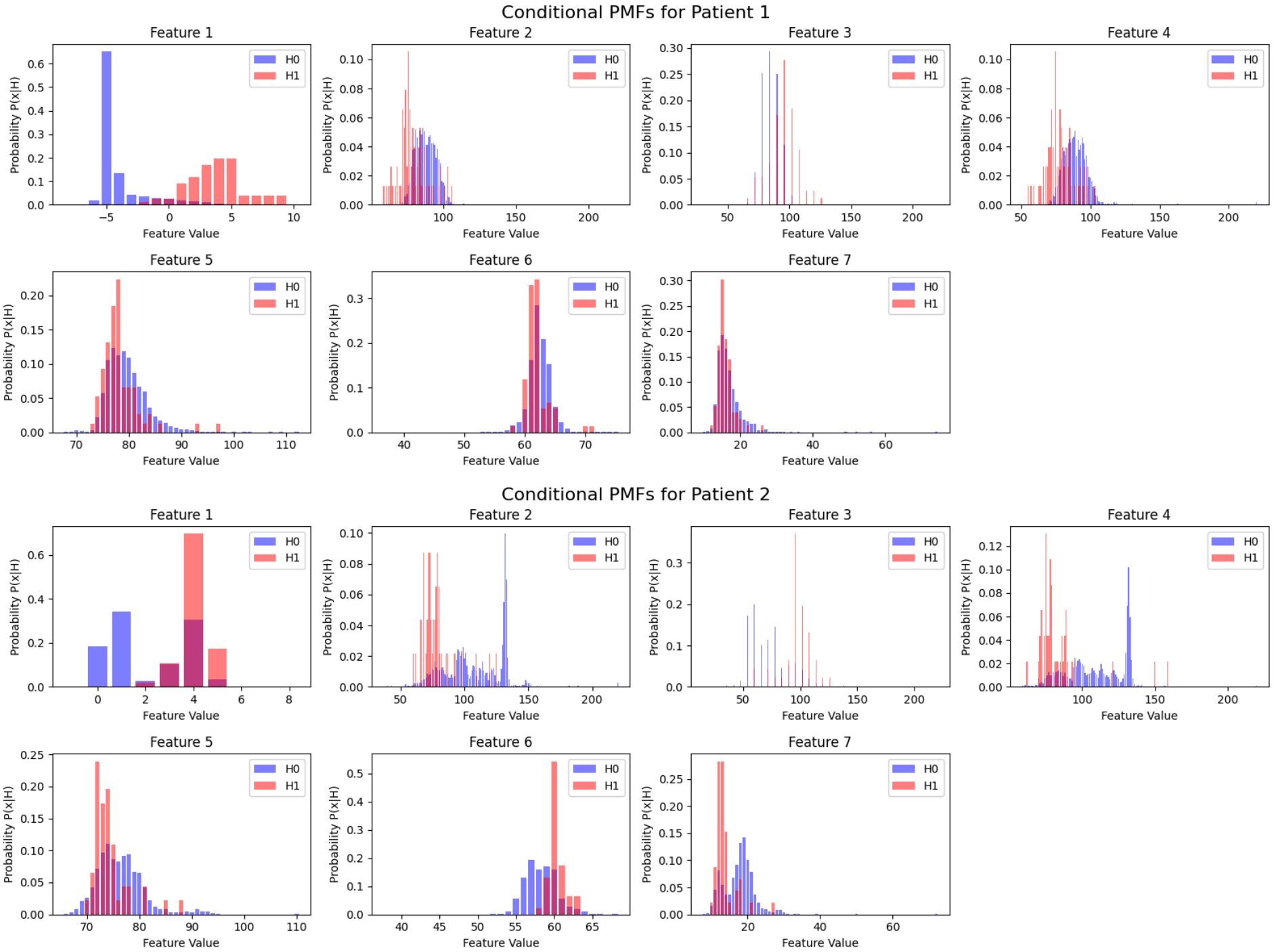
Task 3.1c - Patient Graphs

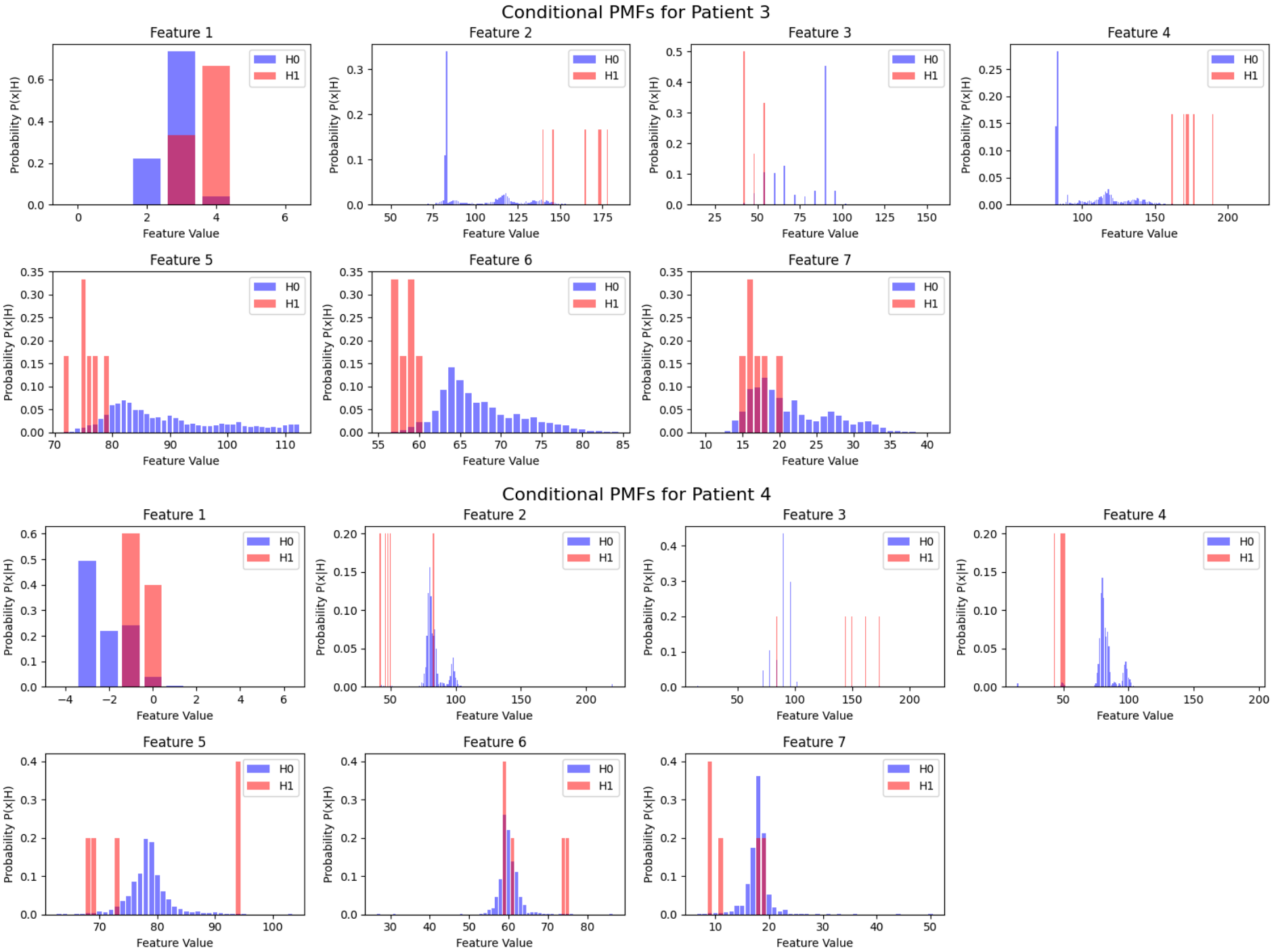
In [16]:
```
def plotData(trainData, likelihoodMatrices, patientName):

   fig, axes = plt.subplots(2, 4, figsize=(16, 6))
   fig.suptitle(f"Conditional PMFs for {patientName}", fontsize=16)

   axes = axes.flatten()
```

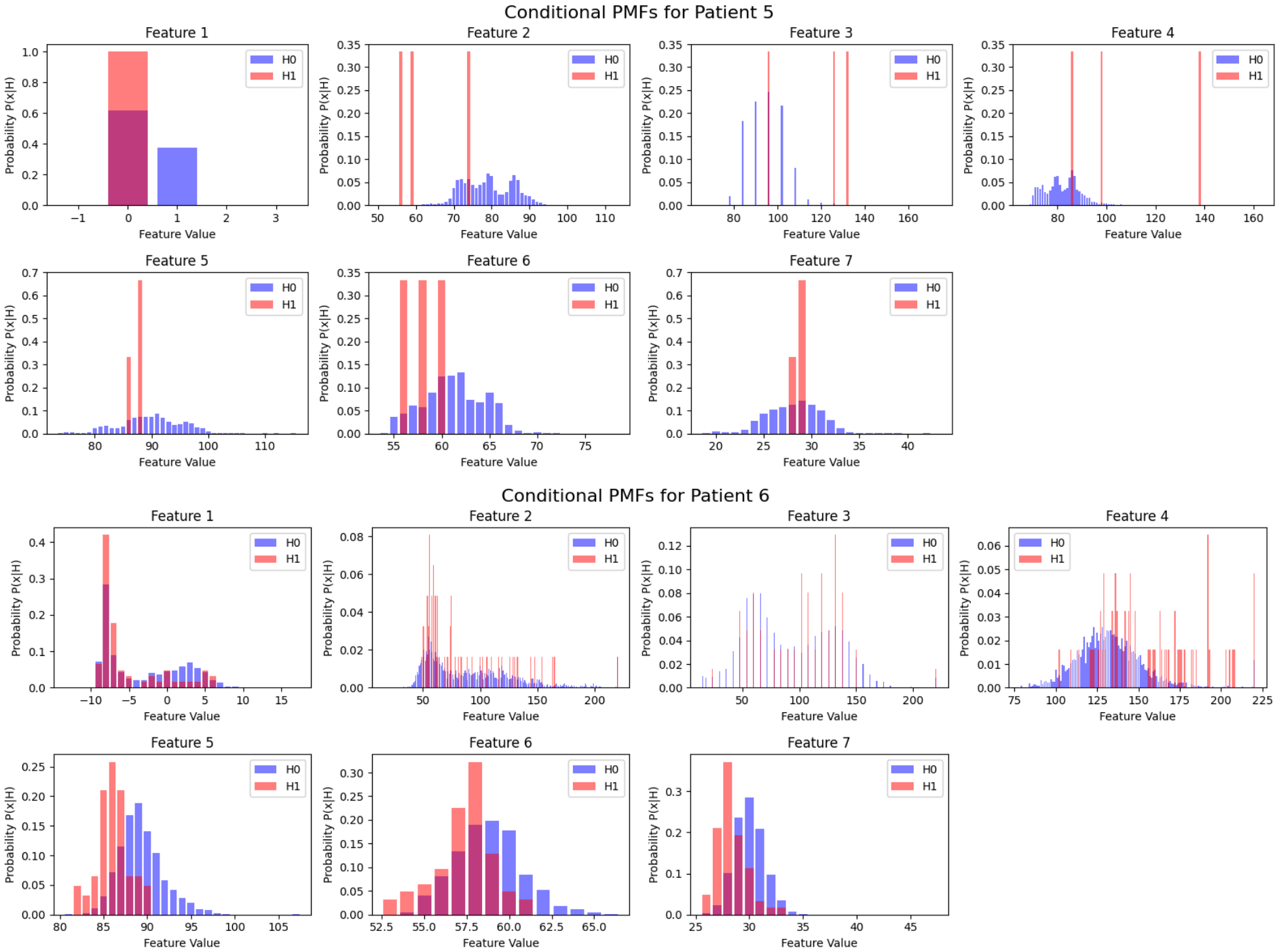
```
    for featureIdx in range(7):
        likelihoodMatrix = likelihoodMatrices[featureIdx]

        numValues = likelihoodMatrix.shape[1]

        minVal = int(np.min(trainData[featureIdx]))
        xValues = range(minVal, minVal + numValues)

        axes[featureIdx].bar(xValues, likelihoodMatrix[0], alpha=0.5, label='H0', color='blue')

        axes[featureIdx].bar(xValues, likelihoodMatrix[1], alpha=0.5, label='H1', color='red')

        axes[featureIdx].set_title(f"Feature {featureIdx+1}")
        axes[featureIdx].set_xlabel("Feature Value")
        axes[featureIdx].set_ylabel("Probability P(x|H)")
        axes[featureIdx].legend()

    if len(axes) > 7:
        axes[7].axis('off')

    plt.tight_layout()
    plt.subplots_adjust(top=0.9)
```

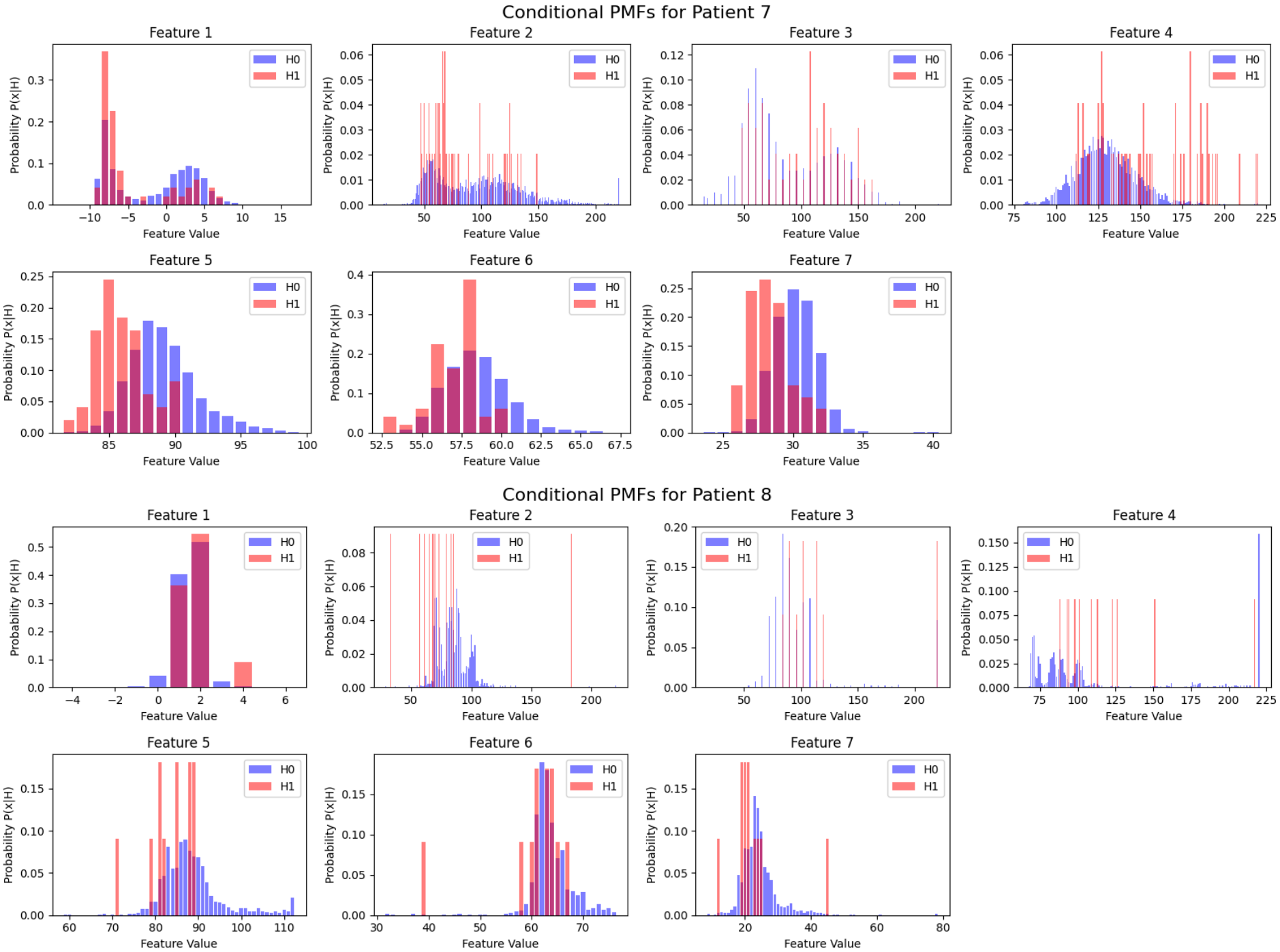
In [17]:

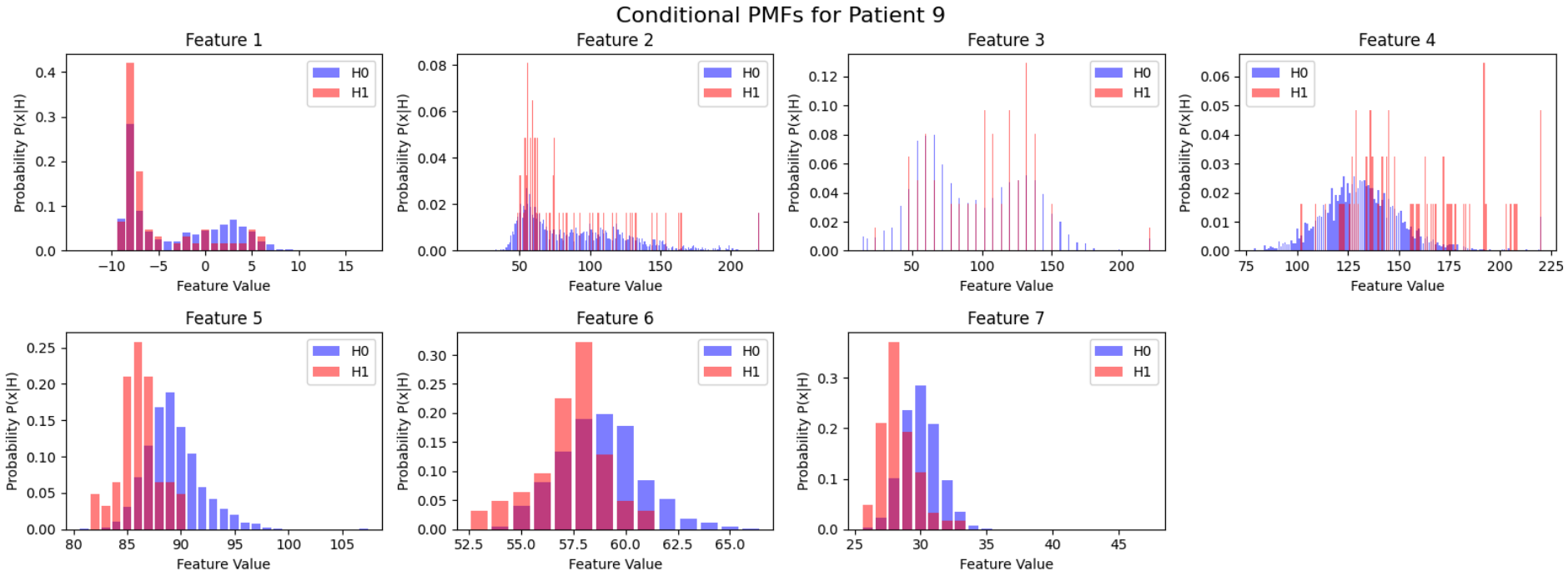
```
plotData(pat1TrainData, pat1LM, "Patient 1")
plotData(pat2TrainData, pat2LM, "Patient 2")
plotData(pat3TrainData, pat3LM, "Patient 3")
plotData(pat4TrainData, pat4LM, "Patient 4")
plotData(pat5TrainData, pat5LM, "Patient 5")
plotData(pat6TrainData, pat6LM, "Patient 6")
plotData(pat7TrainData, pat7LM, "Patient 7")
plotData(pat8TrainData, pat8LM, "Patient 8")
plotData(pat9TrainData, pat9LM, "Patient 9")
```


Conditional PMFs for Patient 1
Feature 1
Feature 2
Feature 3
Feature 4
Feature 5
Feature 6
Feature 7
H0
H1
Probability P(x|H)
Feature Value
Conditional PMFs for Patient 2
Feature 1
Feature 2
Feature 3
Feature 4
Feature 5
Feature 6
Feature 7
H0
H1
Probability P(x|H)
Feature Value


Conditional PMFs for Patient 3
Feature 1
Feature 2
Feature 3
Feature 4
Feature 5
Feature 6
Feature 7
Probability P(x|H)
Feature Value
H0
H1
Conditional PMFs for Patient 4
Feature 1
Feature 2
Feature 3
Feature 4
Feature 5
Feature 6
Feature 7
Probability P(x|H)
Feature Value
H0
H1


Conditional PMFs for Patient 5
Feature 1
Feature 2
Feature 3
Feature 4
Feature 5
Feature 6
Feature 7
Probability P(x|H)
Feature Value
H0
H1
Conditional PMFs for Patient 6
Feature 1
Feature 2
Feature 3
Feature 4
Feature 5
Feature 6
Feature 7
Probability P(x|H)
Feature Value
H0
H1


Conditional PMFs for Patient 7
Feature 1
Feature 2
Feature 3
Feature 4
Feature 5
Feature 6
Feature 7
H0
H1
Probability P(x|H)
Feature Value
Conditional PMFs for Patient 8
Feature 1
Feature 2
Feature 3
Feature 4
Feature 5
Feature 6
Feature 7
H0
H1
Probability P(x|H)
Feature Value



Task 3.1d - ML and MAP Decision Vectors

In [18]:

```python
def mlRule(likelihoodMatrices):
    MLArr = []
    for feature in likelihoodMatrices:
        featureMLArr = []
        for i in range(len(feature[0])):
            if feature[1][i] >= feature[0][i]:
                featureMLArr.append(1)
            else:
                featureMLArr.append(0)
        MLArr.append(featureMLArr)

    return MLArr


def mapRule(likelihoodMatrices, priorH0, priorH1):
    MAPArr = []
    for feature in likelihoodMatrices:
        featureMAPArr = []
        for i in range(len(feature[0])):
            if (priorH1 * feature[1][i]) >= (priorH0 * feature[0][i]):
                featureMAPArr.append(1)
```

```
            else:
                featureMAPArr.append(0)
        MAPArr.append(featureMAPArr)

    return MAPArr
```

Task 3.1e - Hypotheses Table

In [19]:
```
def hypTable(likelihoodMatrices, trainData, priorH0, priorH1):

    mainTable = []
    for featIdx in range(7):
        table = []
        featureData = trainData[featIdx]
        minVal = int(np.min(featureData))
        maxVal = int(np.max(featureData))
        possibleValues = range(minVal, maxVal + 1)
        table.append([i for i in possibleValues])
        table.append(likelihoodMatrices[featIdx][0])
        table.append(likelihoodMatrices[featIdx][1])
        mlArr = mlRule(likelihoodMatrices)
        table.append(mlArr[featIdx])
        mapArr = mapRule(likelihoodMatrices, priorH0, priorH1)
        table.append(mapArr[featIdx])
        npTable = np.array(table)
        mainTable.append(npTable.T)

    return mainTable
```

In [ ]:
```
pat1Table = hypTable(pat1LM, pat1TrainData, pat1PriorH0, pat1PriorH1)
pat2Table = hypTable(pat2LM, pat2TrainData, pat2PriorH0, pat2PriorH1)
pat3Table = hypTable(pat3LM, pat3TrainData, pat3PriorH0, pat3PriorH1)
pat4Table = hypTable(pat4LM, pat4TrainData, pat4PriorH0, pat4PriorH1)
pat5Table = hypTable(pat5LM, pat5TrainData, pat5PriorH0, pat5PriorH1)
pat6Table = hypTable(pat6LM, pat6TrainData, pat6PriorH0, pat6PriorH1)
pat7Table = hypTable(pat7LM, pat7TrainData, pat7PriorH0, pat7PriorH1)
pat8Table = hypTable(pat8LM, pat8TrainData, pat8PriorH0, pat8PriorH1)
pat9Table = hypTable(pat9LM, pat9TrainData, pat9PriorH0, pat9PriorH1)

finalHypothesisTable = [pat1Table, pat2Table, pat3Table, pat4Table,
                        pat5Table, pat6Table, pat7Table, pat8Table, pat9Table]
```

Task 3.2a - Alarm Predictions

In [28]:
```
def generatePreds(testData, hypeTable):
    allMLPreds = []
    allMAPPreds = []

    for i in range(7):
        mlPreds = []
        mapPreds = []

        featureValues = [arr[0] for arr in hypeTable[i]]

        for elem in testData[i]:
            foundMatch = False
            for arr in hypeTable[i]:
                if elem == arr[0]:
                    mlPreds.append(arr[3])
                    mapPreds.append(arr[4])
                    foundMatch = True
                    break

            if not foundMatch:
                closestIdx = np.argmin(np.abs(np.array(featureValues) - elem))
                mlPreds.append(hypeTable[i][closestIdx][3])
                mapPreds.append(hypeTable[i][closestIdx][4])

        allMLPreds.append(mlPreds)
        allMAPPreds.append(mapPreds)

    return allMLPreds, allMAPPreds
```

In [64]:
```
pat1MLPreds, pat1MAPPreds = generatePreds(pat1TestData, pat1Table)
pat2MLPreds, pat2MAPPreds = generatePreds(pat2TestData, pat2Table)
pat3MLPreds, pat3MAPPreds = generatePreds(pat3TestData, pat3Table)
pat4MLPreds, pat4MAPPreds = generatePreds(pat4TestData, pat4Table)
pat5MLPreds, pat5MAPPreds = generatePreds(pat5TestData, pat5Table)
pat6MLPreds, pat6MAPPreds = generatePreds(pat6TestData, pat6Table)
pat7MLPreds, pat7MAPPreds = generatePreds(pat7TestData, pat7Table)
pat8MLPreds, pat8MAPPreds = generatePreds(pat8TestData, pat8Table)
pat9MLPreds, pat9MAPPreds = generatePreds(pat9TestData, pat9Table)
```

Task 3.2b - Error Probabilities

In [30]:
```
def errorTable(predsML, predsMAP, testLabels):
    errorArr = []
    for i in range(7):
        falseAlarmsML = 0
        missDetectionsML = 0

        falseAlarmsMAP = 0
        missDetectionsMAP = 0

        for j in range(len(predsML[i])):
            if predsML[i][j] == 0 and testLabels[j] == 1:
                missDetectionsML += 1
            if predsML[i][j] == 1 and testLabels[j] == 0:
                falseAlarmsML += 1

        for j in range(len(predsMAP[i])):
            if predsMAP[i][j] == 0 and testLabels[j] == 1:
                missDetectionsMAP += 1
            if predsMAP[i][j] == 1 and testLabels[j] == 0:
                falseAlarmsMAP += 1


        falseAlarmsML /= len(predsML[i])
        falseAlarmsMAP /= len(predsMAP[i])

        missDetectionsML /= len(predsML[i])
        missDetectionsMAP /= len(predsMAP[i])


        errorML = 0.5 * falseAlarmsML + 0.5 * missDetectionsML
        errorMAP = pat1PriorH0 * falseAlarmsMAP + pat1PriorH1 * missDetectionsMAP

        errorArr.append([[falseAlarmsML, missDetectionsML, errorML], [falseAlarmsMAP, missDetectionsMAP, errorMAP]])

    return errorArr
```

In [ ]:
```
pat1ErrorTable = errorTable(pat1MLPreds, pat1MAPPreds, pat1TestLabels)
pat2ErrorTable = errorTable(pat2MLPreds, pat2MAPPreds, pat2TestLabels)
pat3ErrorTable = errorTable(pat3MLPreds, pat3MAPPreds, pat3TestLabels)
pat4ErrorTable = errorTable(pat4MLPreds, pat4MAPPreds, pat4TestLabels)
pat5ErrorTable = errorTable(pat5MLPreds, pat5MAPPreds, pat5TestLabels)
pat6ErrorTable = errorTable(pat6MLPreds, pat6MAPPreds, pat6TestLabels)
pat7ErrorTable = errorTable(pat7MLPreds, pat7MAPPreds, pat7TestLabels)
pat8ErrorTable = errorTable(pat8MLPreds, pat8MAPPreds, pat8TestLabels)
```

```
pat9ErrorTable = errorTable(pat9MLPreds, pat9MAPPreds, pat9TestLabels)

finalErrorTable = [pat1ErrorTable, pat2ErrorTable, pat3ErrorTable, pat4ErrorTable,
                   pat5ErrorTable, pat6ErrorTable, pat7ErrorTable, pat8ErrorTable, pat9ErrorTable]
```

Task 3.2 - Feature Pair Selection

Lowest Error Pairs Test

In [63]:
```
bestMLPairsArr = []
bestMAPPairsArr = []

for errTable in finalErrorTable:
    mlErrs = []
    mapErrs = []
    for errorArr in errTable:
        mlErrs.append(errorArr[0][2])
        mapErrs.append(errorArr[1][2])

    # print(mlErrs)
    # print(mapErrs)
    bestFeatPairML = np.argsort(mlErrs)[:2] + 1
    bestFeatPairMAP = np.argsort(mapErrs)[:2] + 1
    bestMLPairsArr.append(bestFeatPairML)
    bestMAPPairsArr.append(bestFeatPairMAP)

print("Best ML Rule Feature Pairs for each Patient: ", bestMLPairsArr)
print("Best MAP Rule Feature Pairs for each Patient: ", bestMAPPairsArr)
```

```
Best ML Rule Feature Pairs for each Patient:  [array([1, 3]), array([6, 4]), array([1, 5]), array([7, 5]), array([4,
1]), array([4, 6]), array([5, 6]), array([2, 3]), array([4, 6])]
Best MAP Rule Feature Pairs for each Patient:  [array([6, 3]), array([1, 5]), array([1, 5]), array([1, 7]), array([1,
7]), array([3, 6]), array([3, 5]), array([1, 3]), array([3, 6])]
```

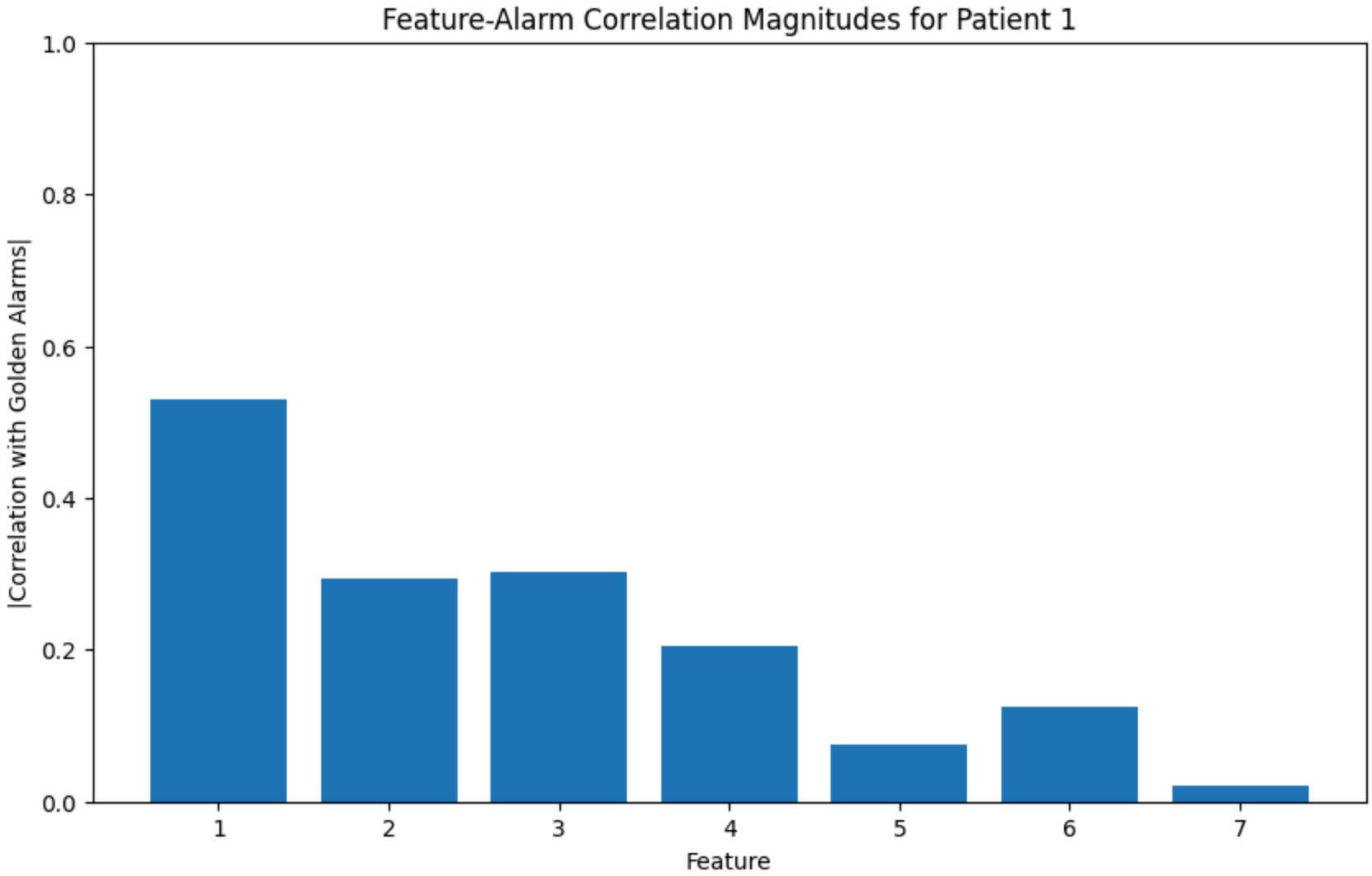
Golden Alarm Correlation Test

In [60]:
```
def analyzeFeatureAlarmCorrelation(patientTestData, patientTestLabels, patientName):
    correlations = []

    plt.figure(figsize=(15, 10))
    plt.suptitle(f'Feature vs Golden Alarms Correlation for {patientName}', fontsize=16)
```

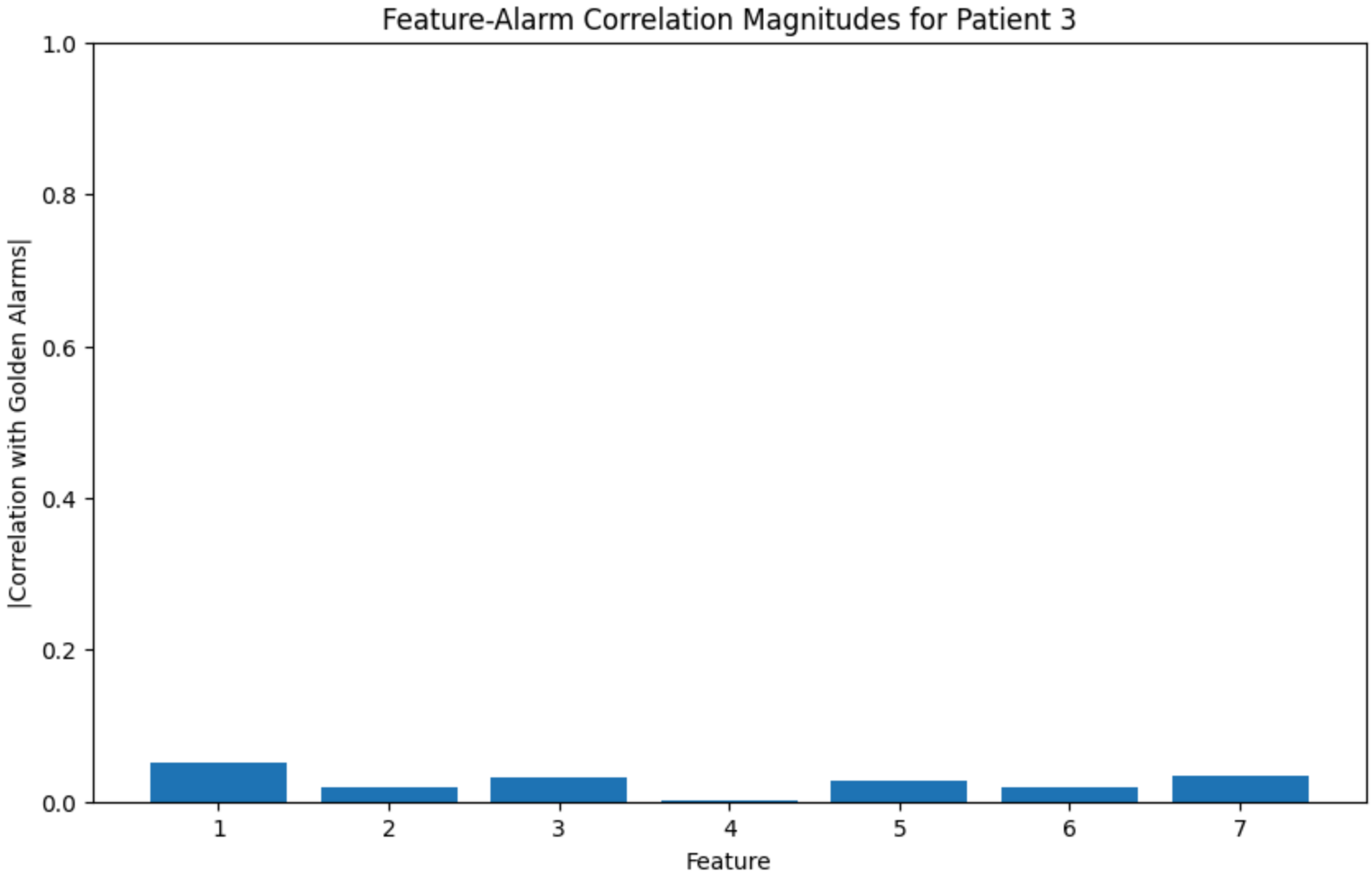
```
    for featureIdx in range(7):
        featureData = patientTestData[featureIdx]
        correlation = np.corrcoef(featureData, patientTestLabels)[0, 1]
        correlations.append(correlation)

    plt.figure(figsize=(10, 6))
    plt.bar(range(1, 8), np.abs(correlations))
    plt.xlabel('Feature')
    plt.ylabel('|Correlation with Golden Alarms|')
    plt.title(f'Feature-Alarm Correlation Magnitudes for {patientName}')
    plt.xticks(range(1, 8))
    plt.ylim(0, 1)

    return correlations
```

In [ ]:
```
pat1Correlations = analyzeFeatureAlarmCorrelation(pat1TestData, pat1TestLabels, "Patient 1")
pat3Correlations = analyzeFeatureAlarmCorrelation(pat3TestData, pat3TestLabels, "Patient 3")
pat4Correlations = analyzeFeatureAlarmCorrelation(pat4TestData, pat4TestLabels, "Patient 4")
pat5Correlations = analyzeFeatureAlarmCorrelation(pat5TestData, pat5TestLabels, "Patient 5")
pat6Correlations = analyzeFeatureAlarmCorrelation(pat6TestData, pat6TestLabels, "Patient 6")

for patientNum, correlations in [(1, pat1Correlations), (3, pat3Correlations),
                                 (4, pat4Correlations), (5, pat5Correlations), (6, pat6Correlations)]:
    absCorrelations = np.abs(correlations)
    topFeatures = np.argsort(absCorrelations)[-2:][::-1]
    print(f"Patient {patientNum} - Top 2 correlated features:
          {topFeatures + 1} with correlations: {[correlations[i] for i in topFeatures]}")
```

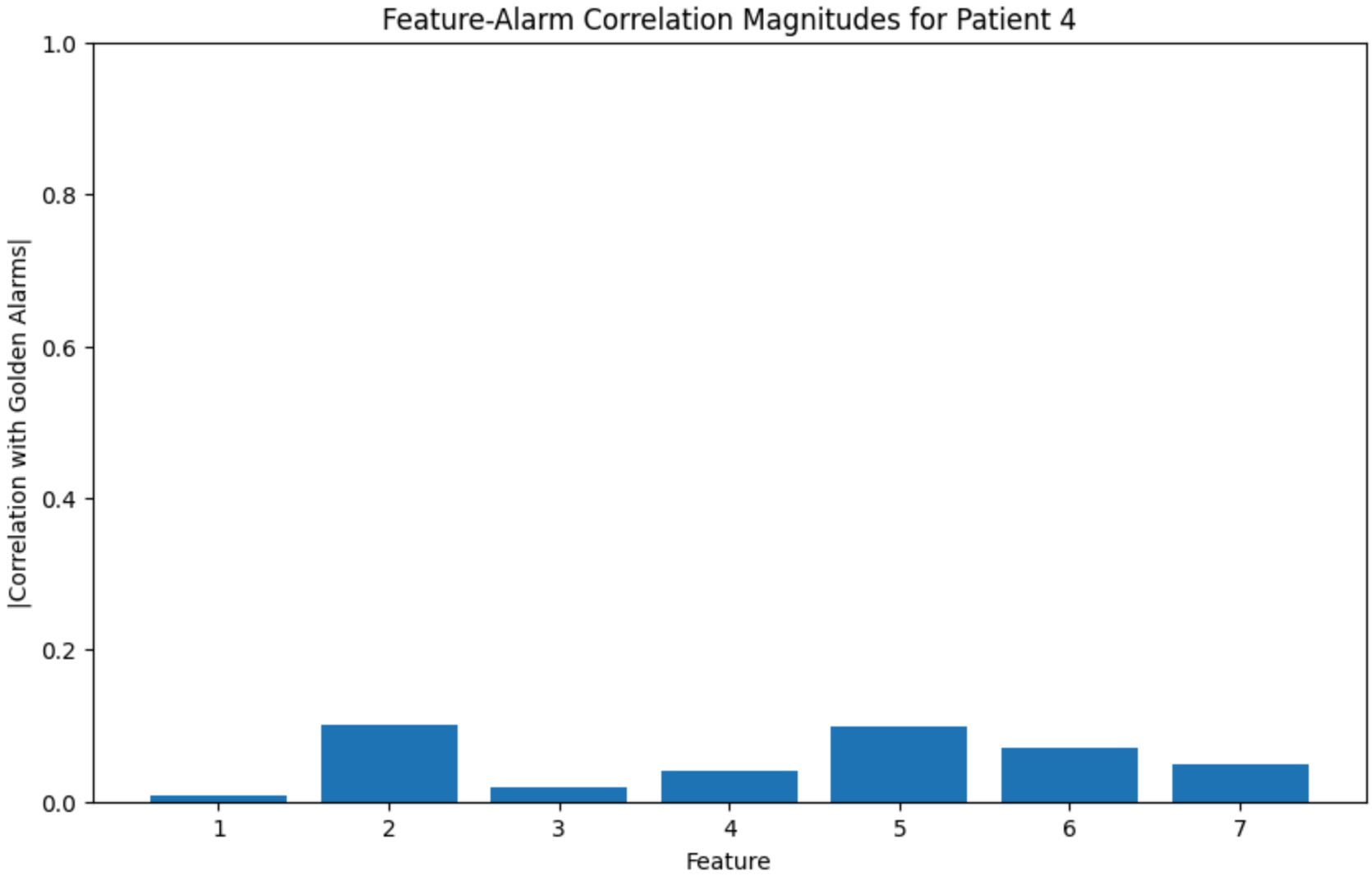
```
Patient 1 - Top 2 correlated features: [1 3] with correlations: [0.5287154422038963, 0.30151677386561837]
Patient 3 - Top 2 correlated features: [1 7] with correlations: [-0.05205556044446076, 0.03486218797046112]
Patient 4 - Top 2 correlated features: [2 5] with correlations: [0.10028935858094924, -0.09950230996429597]
Patient 5 - Top 2 correlated features: [3 4] with correlations: [0.16341836819567962, 0.14606344739898264]
Patient 6 - Top 2 correlated features: [4 6] with correlations: [0.1957787974996459, -0.1585817167459278]
<Figure size 1500x1000 with 0 Axes>
```



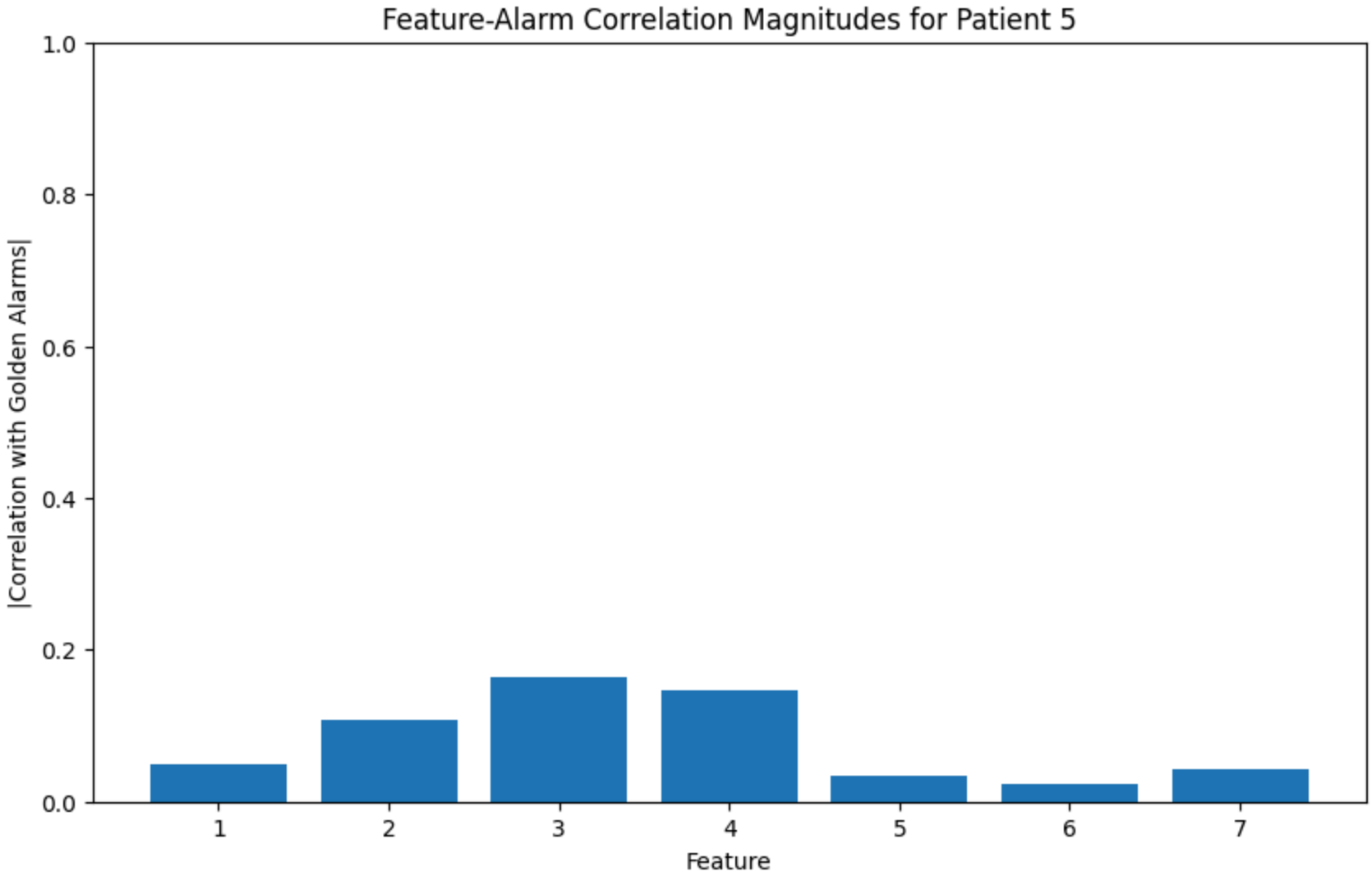
<Figure size 1500x1000 with 0 Axes>



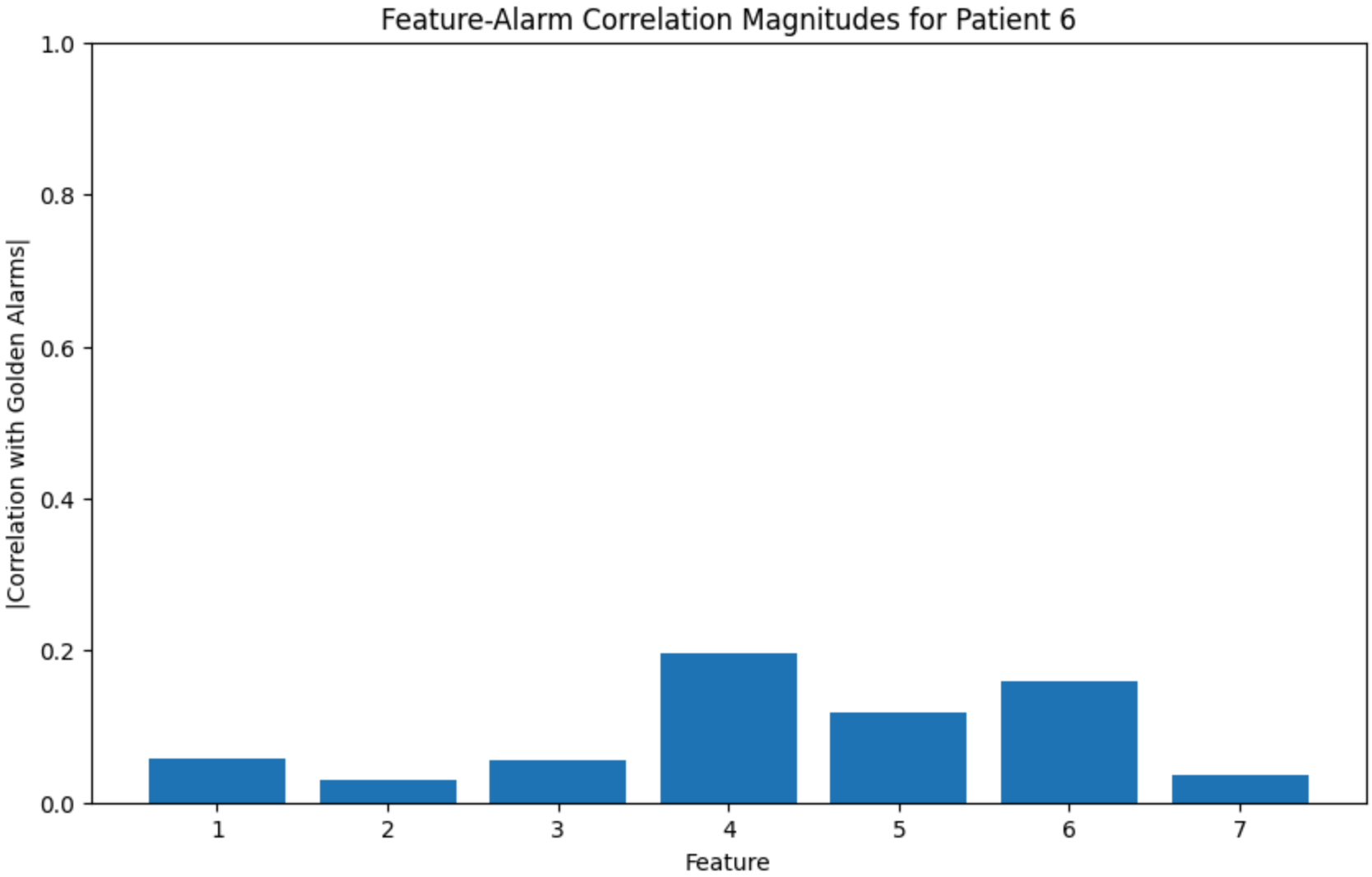
<Figure size 1500x1000 with 0 Axes>



<Figure size 1500x1000 with 0 Axes>



<Figure size 1500x1000 with 0 Axes>



Pair Decisions Based on Data

ML Rule: Patients 1, 3, 4

Patient 1 - Features 1 and 3:

In both the lowest error test and the golden alarm correlation test, both features 1 and 3 seemed to be the most accurate and prominent in determining the result for Patient 1.

Patient 3 - Features 1 and 7:

In both the lowest error test and the golden alarm correlation test, feature 1 seemed to be the most accurate and prominent in determining the result for Patient 3. The next step was to choose between lower error (feature 5) or higher impact (feature 7). In this case we decided to go with feature 7 as our second option because it also had a decently low error while providing a more significant impact on the patient's results.

Patient 4 - Features 2 and 5:

In both the lowest error test and the golden alarm correlation test, feature 5 seemed to be the most accurate and prominent in determining the result for Patient 4. The next step was to choose between lower error (feature 7) or higher impact (feature 2). In this case we decided to go with feature 2 as our second option because it had a decently low error while providing a much more significant impact on the patient's results as compared to feature 7.

MAP Rule: Patients 3, 5, 6

Patient 3 - Features 1 and 7:

In both the lowest error test and the golden alarm correlation test, feature 1 seemed to be the most accurate and prominent in determining the result for Patient 3. The next step was to choose between lower error (feature 5) or higher impact (feature 7). In this case we decided to go with feature 7 as our second option because it also had a decently low error while providing a more significant impact on the patient's results.

Patient 5 - Features 3 and 1:

For this patient, we got completely different pairs in both the lowest error test and the golden alarm correlation test. So we decided to choose the best of the 2 pairs and combine them into the pair of features for this patient. From the lower error test, we chose feature 1 since it had the lowest error, and from the golden alarm correlation test we chose feature 3 since it had the highest impact on determining the patient's results.

Patient 6 - Features 6 and 4:

In both the lowest error test and the golden alarm correlation test, feature 6 seemed to be the most accurate and prominent in determining the result for this patient. The next step was to choose between lower error (feature 3) or higher impact (feature 4). In this case we decided to go with feature 4 as our second option because it had a decently low error while providing a much more significant impact on the patient's results as compared to feature 3.