Designing an Early Warning System for Clinical Deterioration

Step 0: Load the dataset

```
import pandas as pd
```

```
data = pd.read_csv('https://machine-learning-for-healthcare-applications-f276df.gitlab.io/labs/FinalProject/early_warning_longitudinal_
```

Step 1: Review longitudinal patient data

What is expected

Print the dataset structure to confirm patient IDs and timestamps.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   patient_id          3000 non-null   object 
 1   timestamp           3000 non-null   object 
 2   heart_rate          3000 non-null   float64
 3   systolic_bp         3000 non-null   float64
 4   deterioration_flag  3000 non-null   int64  
dtypes: float64(2), int64(1), object(2)
memory usage: 117.3+ KB
```

```
data.head()
```

	patient_id	timestamp	heart_rate	systolic_bp	deterioration_flag
0	P0001	2024-01-07 00:00:00	68.8	130.8	1
1	P0001	2024-01-07 04:00:00	72.2	127.6	1
2	P0001	2024-01-07 08:00:00	73.1	127.7	1
3	P0001	2024-01-07 12:00:00	72.7	125.7	1
4	P0001	2024-01-07 16:00:00	78.8	125.8	1

Step 2: Prepare data for temporal analysis

What is expected

Sort the dataset by patient and time.

```
data.sort_values(['patient_id', 'timestamp'])
```

	patient_id	timestamp	heart_rate	systolic_bp	deterioration_flag
0	P0001	2024-01-07 00:00:00	68.8	130.8	1
1	P0001	2024-01-07 04:00:00	72.2	127.6	1
2	P0001	2024-01-07 08:00:00	73.1	127.7	1
3	P0001	2024-01-07 12:00:00	72.7	125.7	1
4	P0001	2024-01-07 16:00:00	78.8	125.8	1
...	...	...	...	...	...
2995	P0300	2024-01-02 20:00:00	77.4	128.8	0
2996	P0300	2024-01-03 00:00:00	75.0	134.2	0
2997	P0300	2024-01-03 04:00:00	78.6	129.1	0
2998	P0300	2024-01-03 08:00:00	82.9	141.0	0
2999	P0300	2024-01-03 12:00:00	83.4	135.0	0

3000 rows × 5 columns

Step 3: Create recent-value temporal features

What is expected

Print the most recent clinical measurements per patient.

```
data.groupby('patient_id')[['heart_rate','systolic_bp']].last()
```

patient_id	heart_rate	systolic_bp
P0001	75.0	125.1
P0002	81.2	131.9
P0003	69.8	112.0
P0004	70.9	119.3
P0005	87.8	115.5
...	...	...
P0296	77.2	107.3
P0297	81.4	114.0
P0298	87.8	117.2
P0299	75.4	144.6
P0300	83.4	135.0

300 rows × 2 columns

Step 4: Engineer rolling temporal features

What is expected

Compute a rolling average heart rate feature.

```
data.groupby('patient_id')['heart_rate'].rolling(3).mean()
```

		heart_rate
patient_id		
P0001	**0**	NaN
	1	NaN
	2	71.366667
	3	72.666667
	4	74.866667
...	**...**	...
P0300	**2995**	79.200000
	2996	78.133333
	2997	77.000000
	2998	78.833333
	2999	81.633333

3000 rows × 1 columns

dtype: float64

Step 5: Engineer temporal trend features

What is expected

Compute change between consecutive heart rate measurements.

```
data.groupby('patient_id')['heart_rate'].diff()
```

	heart_rate
0	NaN
1	3.4
2	0.9
3	-0.4
4	6.1
...	...
2995	-4.6
2996	-2.4
2997	3.6
2998	4.3
2999	0.5

3000 rows × 1 columns

dtype: float64

Step 6: Aggregate temporal features at the patient level

What is expected

Aggregate temporal clinical features into one row per patient.

```
patient_temporal_features = data.groupby("patient_id").agg(
    latest_heart_rate=("heart_rate", "last"),
    latest_systolic_bp=("systolic_bp", "last")
)
patient_temporal_features
```

	latest_heart_rate	latest_systolic_bp
patient_id		
P0001	75.0	125.1
P0002	81.2	131.9
P0003	69.8	112.0
P0004	70.9	119.3
P0005	87.8	115.5
...	...	...
P0296	77.2	107.3
P0297	81.4	114.0
P0298	87.8	117.2
P0299	75.4	144.6
P0300	83.4	135.0

300 rows × 2 columns

Step 7: Visualize patient population using dimensionality reduction

What is expected

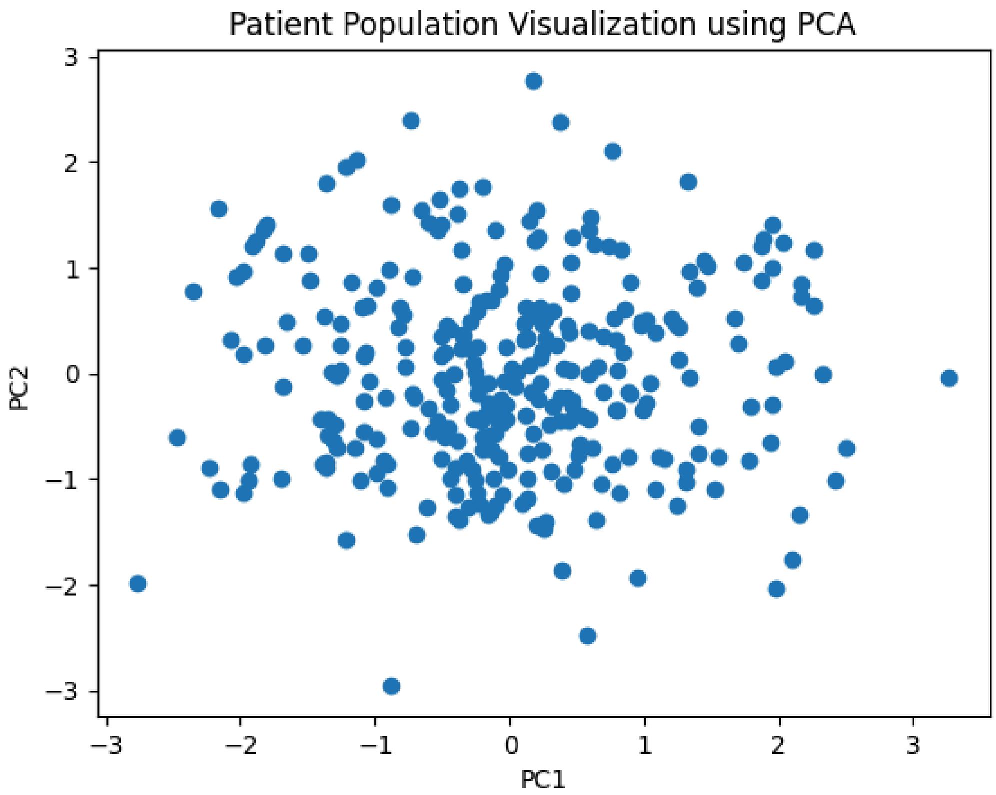
Apply PCA and visualize patients in 2D.

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

scaler = StandardScaler()
scaled_features = scaler.fit_transform(patient_temporal_features)

pca = PCA(n_components=2)
pca_components = pca.fit_transform(scaled_features)
```

```
plt.figure()
plt.scatter(pca_components[:,0], pca_components[:,1])
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("Patient Population Visualization using PCA")
plt.show()
```



Step 8: Build a baseline early warning classifier

What is expected

Train a logistic regression model using patient-level features.

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

X = data.drop(columns=['deterioration_flag','patient_id','timestamp'])
y = data['deterioration_flag']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

model = LogisticRegression()
model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

Step 9: Visualize model performance using a confusion matrix

What is expected

Visualize the confusion matrix using a Seaborn heatmap.

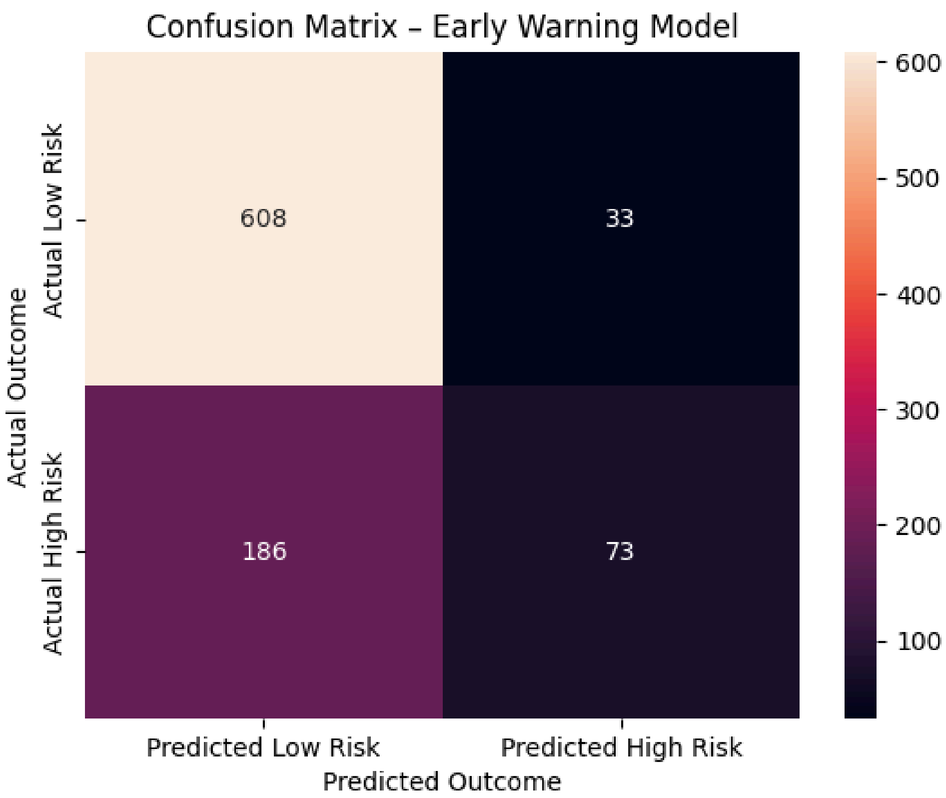
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

cm_df = pd.DataFrame(
    cm,
    index=["Actual Low Risk", "Actual High Risk"],
    columns=["Predicted Low Risk", "Predicted High Risk"]
)

plt.figure()
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title("Confusion Matrix – Early Warning Model")
```

```
plt.xlabel("Predicted Outcome")
plt.ylabel("Actual Outcome")
plt.show()
```



Step 10: Compute specificity manually

What is expected

Compute specificity from the confusion matrix.

```
tn, fp, fn, tp = cm.ravel()
specificity = tn / (tn + fp)
print(specificity)
```

```
0.9485179407176287
```