Protein Expression Patterns and Treatment Outcomes in Breast Cancer Patients

Final Report

Pooja Manikonda

Data Visualization Lead

INFO-B 512 SCIENTIC AND CLINICAL DATA MANAGEMENT

Instructor: Yan Zhuang, Ph.D.

15th December, 2024

Final Report

Question: Describe the database design used in your project and explain why it is effective for your specific role in the project. Provide examples to support your explanation.

Overview of the **'Database Design':**

The original 'Breast Cancer dataset' contains patient information related to breast cancer, including demographic data, protein expression levels, tumour characteristics, and surgical details. All data is presented in a single flat table (**1NF** and **2NF**)with a single-column primary key (Patient_ID). Although the given values for each attribute are atomic and there are no partial dependencies, this organization can lead to redundancy and potential insertion, deletion, or updation anomalies. To achieve **3NF**, the given dataset has been separated into multiple tables/entities:

1)The 'Patient table' with the attributes 'Patient_ID', 'Age', 'Gender', and 'Patient_Status' captures unique demographic information for each patient. By isolating patient data, we ensure that any changes to demographic information (like age or status) do not affect other records. Each patient can be uniquely identified by Patient_ID, which serves as the primary key.

2)The 'Diagnosis table' with the attributes 'Diagnosis_ID', 'Tumor_Stage', 'Histology' and 'Patient_ID' has been created so that we can manage multiple diagnoses for a single patient without redundancy. The use of Diagnosis_ID as a primary key allows us to uniquely identify each diagnosis record, while Patient_ID serves as a foreign key linking back to the Patient table.

3)The 'Visit table' with the attributes 'visit_ID', 'Date_of_Surgery', 'Date_of_Last_Visit' and 'Patient_ID' tracks visits and surgical dates associated with each patient. By creating a separate table for visits, we can maintain a history of visits without duplicating patient demographic data. This structure allows for easy tracking of multiple visits per patient.

4)The 'Surgery table' with the attributes
'Surgery_ID', 'Surgery_Type_ID', and 'Diagnosis_ID' records details about surgeries performed on patients, linking each surgery to a specific diagnosis and surgery type. This allows for flexibility in managing different types of surgeries and their associations with various diagnoses.

5)The ‘Lab Results table’ with the attributes ‘Lab_ID’, ‘Diagnosis_ID’, ‘ER_Status’, ‘PR_Status’ and ‘HER_Status’ captures laboratory results related to each diagnosis. By isolating lab results, we ensure that any updates or changes to lab data do not interfere with patient or diagnosis records. The use of Diagnosis_ID as a foreign key maintains the relationship between lab results and their corresponding diagnoses.

6)The ‘Protein Expression table’ with the attributes ‘Protein_ID’, ‘Protein1’, ‘Protein2’, ‘Protein3’, ‘Protein4’, ‘Diagnosis_ID’ focuses on protein expression levels measured in relation to each diagnosis. Here, we ensure that changes in protein levels are linked directly to the appropriate diagnosis with ‘Diagnosis_ID’ as the foreign key.

7)The ‘Surgery Type table’ with the attributes ‘Surgery Type _ID’, and ‘Surgery_Name’ defines different types of surgeries performed, allowing for consistent naming and categorization of surgical procedures across multiple records.

This structured database design not only supports effective data management but also enhances the ability to perform meaningful data visualizations that can lead to valuable insights in breast cancer research and treatment strategies due to the following:

a) **Clear Relationships**: The use of foreign keys (e.g., linking the Diagnosis table to the Patient table via Patient_ID) allows for straightforward joins when querying data for visualization. For example, visualizing the relationship between tumour stage and patient demographics can be easily accomplished by joining the Patient and Diagnosis tables.
b) The normalization of the data into separate tables, avoids duplication of information (e.g., patient demographics. This makes it easier to maintain and update data without affecting visualizations that rely on this information.
c) **Efficient Querying**: The structure allows for efficient querying of specific datasets needed for visualization. For instance, if we want to visualize lab results against different tumor stages, we can directly query the Lab Results and Diagnosis tables without sifting through irrelevant data.
d) **Enhanced Data Integrity**: Each piece of information is stored in its relevant context, reducing the risk of inconsistencies (e.g., ensuring that all lab results are accurately linked to their respective diagnoses). This integrity is crucial when creating reliable visualizations that inform clinical decisions.

Examples to support effectiveness of the database design:

1. **Patient Demographics Against Tumor Stages:** By querying the Patient and Diagnosis tables together, a bar chart/pie chart showing how different age groups are affected by various tumor stages is visualized.
2. **Comparing Protein Expressions Across Different Histologies**: By joining the Protein Expression and Diagnosis tables,a scatter plots that compares protein levels across different histological types of breast cancer, can be visualized thus, aiding in research and clinical insights.
3. **Analyzing Lab Results Trends Over Time**: Using the Visit and Lab Results tables, trends in lab results over time for individual or groups of patients can be visualized, to help identify patterns related to treatment effectiveness.

Data Visualization

I) Required question:

1. Detail your approach to data visualization and provide corresponding SQL queries

Approach to Data Visualization:

Post data-cleaning, creating and populating the database, the approach to data visualization involves using SQL queries to aggregate and summarize breast cancer patient data, followed by visualizing the results using Python libraries such as Pandas, Matplotlib, and Seaborn.

Step 1 **: Understanding the Dataset**

The dataset contains various attributes related to breast cancer patients, including demographic information (e.g., Patient_ID, Age, Gender), protein expression levels (Protein1, Protein2, Protein3, Protein4), tumor characteristics (Tumour_Stage, Histology), and patient outcomes (Patient_Status).

Step 2: Identifying Key Metrics

Key metrics for visualization include average protein levels by tumor stage, total patient counts by tumor stage, and survival rates segmented by tumor stage and patient status.

Step 3**:** ***SQL Queries for Data Extraction***

The following SQL queries are used to extract relevant data for visualization:

Step 4**:** ***Data Transformation***

The results from the queries are converted into Pandas 'DataFrames' for easier manipulation and visualization in Python

Step 5: Creating Visualization

Using libraries like Matplotlib and Seaborn, various plots (bar, stacked bar plots and heatmaps) are created based on the extracted data.

Query 1: Protein Summary by Tumor Stage and Histology

By joining the patient and diagnosis tables, this query

- Calculates average levels of four proteins (Protein1, Protein2, Protein3, Protein4) for each tumor stage and histology type.
- Rank tumor stages within each histology type based on the total number of patients in descending order.
- Display only the top-ranked tumor stage **(stage II)** for each histology type.

```
WITH ProteinSummary AS (
   SELECT
      d.Tumour_Stage,
      d.Histology,
      AVG(pr.Protein1) AS Avg_Protein1,
      AVG(pr.Protein2) AS Avg_Protein2,
      AVG(pr.Protein3) AS Avg_Protein3,
      AVG(pr.Protein4) AS Avg_Protein4,
      COUNT(*) AS Total_Patients
   FROM Diagnoses d
   JOIN Proteins pr ON d.Diagnosis_ID = pr.Diagnosis_ID
   GROUP BY d.Tumour_Stage, d.Histology
),
RankedTumorStages AS (
    SELECT
        Tumour_Stage,
        Histology,
        Avg_Protein1,
        Avg_Protein2,
        Avg_Protein3,
        Avg_Protein4,
        Total_Patients,
        RANK() OVER (PARTITION BY Histology ORDER BY Total_Patients DESC) AS
Stage_Rank
    FROM ProteinSummary
)
SELECT
    Tumour_Stage,
    Histology,
    Avg_Protein1,
    Avg_Protein2,
    Avg_Protein3,
    Avg_Protein4,
    Total_Patients,
    Stage_Rank
FROM RankedTumorStages
WHERE Stage_Rank = 1
ORDER BY Histology, Total_Patients DESC;
```

Visualization: The bar chart shows the total number of patients for each histology type (limited to Tumor Stage II, since its ranked 1).

Key Observations:

- Infiltrating Ductal Carcinoma has the highest patient count (~121). Therefore, it is the most prevalent histology type among Stage II BRCA patients.
- Infiltrating Lobular Carcinoma has a moderate patient count (~56) while Mucinous Carcinoma has a minimal patient count (~9). These 2 types highlight histological heterogeneity in BRCA patients.

Query 2: Protein averages and patient survival rates by Tumor Stage

The query calculates:

- Average levels of four proteins (Protein1 to Protein4) for each combination of tumor stage(d.Tumour_Stage) and patient survival status (p.Patient_Status: Alive or Dead).Uses Joins across 3 tables: Patients(survival status), Diagnoses(tumor stage) and Protein epression.
- Total number of patients for each combination (COUNT(*) AS Total_Patients).
- Percentage distribution of patients for each survival status within a specific tumor stage using (SUM OVER)
  ROUND((Total_Patients * 100.0) / SUM(Total_Patients) OVER (PARTITION BY Tumour_Stage), 2)
  Results are sorted by Tumour_Stage (ascending) and Patient_Status (descending, e.g., prioritizing 'Alive') and ties are further sorted by Total_Patients in descending order.

```
WITH ProteinSurvival AS (

  SELECT

    d.Tumour_Stage,

    p.Patient_Status,

    AVG(pr.Protein1) AS Avg_Protein1,

    AVG(pr.Protein2) AS Avg_Protein2,
    AVG(pr.Protein3) AS Avg_Protein3,
    AVG(pr.Protein4) AS Avg_Protein4,
    COUNT(*) AS Total_Patients
  FROM Patients p
  JOIN Diagnoses d ON p.Patient_ID = d.Patient_ID
  JOIN Proteins pr ON d.Diagnosis_ID = pr.Diagnosis_ID
  GROUP BY d.Tumour_Stage, p.Patient_Status
)
SELECT
  Tumour_Stage,
  Patient_Status,
  Avg_Protein1,
  Avg_Protein2,
  Avg_Protein3,
  Avg_Protein4,
  Total_Patients,
  ROUND((Total_Patients * 100.0) / SUM(Total_Patients) OVER (PARTITION BY
Tumour_Stage), 2) AS Percentage
FROM ProteinSurvival
ORDER BY Tumour_Stage, Patient_Status DESC, Total_Patients DESC;
```

Visualization: The bar chart shows the **survival rates** by tumor stage, segmented by **Patient Status** (Alive vs Dead) with 'Tumor Stages' (I, II, III) on x-axis, Percentage of patients (%) on y-axis and the hue d ifferentiates between **Alive** (orange) and **Dead** (blue) patient statuses.

Key Observations:

- **Stage I**: The survival rate (Alive) is the highest (~85%), and the mortality rate (Dead) is very low (~15%).
- **Stage II**: Survival rate drops slightly (~80%), while the mortality rate increases (~20%).
- **Stage III**: The survival rate decreases further (~75%), and the mortality rate increases to ~25%.

Therefore, As the **tumor stage progresses** from Stage I to Stage III, the **percentage of deceased patients increases**, indicating worsening survival outcomes with advanced tumor stages which implies that early detection (e.g., Stage I) is critical, as survival rates are significantly higher.

Query 3: Average Protein Levels by Tumor Stage

The **Diagnoses** table (d) and the **Proteins** table (pr) are joined using the Diagnosis_ID. For each **Tumor Stage**, the query calculates:

- Average levels of Protein1, Protein2, Protein3, and Protein4.
- Total number of patients (COUNT(*)).

The results are grouped by Tumour_Stage and the final output is sorted in ascending order of Tumour_Stage (I, II, III)

```
SELECT

   d.Tumour_Stage,
   AVG(pr.Protein1) AS Avg_Protein1,
   AVG(pr.Protein2) AS Avg_Protein2,
   AVG(pr.Protein3) AS Avg_Protein3,
   AVG(pr.Protein4) AS Avg_Protein4,
   COUNT(*) AS Total_Patients
FROM Diagnoses d
JOIN Proteins pr ON d.Diagnosis_ID = pr.Diagnosis_ID
GROUP BY d.Tumour_Stage
ORDER BY d.Tumour_Stage
cursor.execute(query)
results = cursor.fetchall()
columns = [desc[0] for desc in cursor.description]
```

```
df = pd.DataFrame(results, columns=columns)
print(df)
df.to_csv('Protein_Tumour_Stage.csv', index=False)
cursor.close() connection.close()
```

Prepare the data for a heatmap

```
protein_df = df[['Tumour_Stage', 'Avg_Protein1', 'Avg_Protein2', 'Avg_Protein3',
'Avg_Protein4']]

protein_df.set_index('Tumour_Stage', inplace=True)
```

Heatmap: Average Protein Levels by Tumour Stage

```
plt.figure(figsize=(10, 6))

sns.heatmap(protein_df, annot=True, cmap='coolwarm', cbar_kws={'label': 'Protein
Expression Levels'})

plt.title('Average Protein Levels by Tumour Stage')

plt.xlabel('Proteins')

plt.ylabel('Tumour Stage')

plt.show()
```

Visualization: The heatmap displays average protein expression levels (Protein1 to Protein4) for different tumor stages (I, II, and III).

Key Observations:

Avg_Protein2 Dominates Across All Tumor Stages:

- **Protein2** has the **highest expression levels** compared to the other proteins.
- Tumor Stage I: **1.0** (maximum possible expression).
- Tumor Stage II: **0.96**.
- Tumor Stage III: **0.86**.

The consistently high expression of **Protein2** across all tumor stages suggests it could serve as a **potential biomarker** for diagnosis or monitoring disease progression.

II) Elective questions 1: Discuss any challenges faced while visualizing large datasets and how you overcame them

a) **Performance Issues:** Slow query performance due to large size of the dataset.

- **Solution**: Implementing indexing on frequently queried fields (e.g., Tumour_Stage, Patient_Status) to speed up query execution.

b)**Data Quality Issues**: The dataset contains missing or inconsistent data entries that can skew results or lead to incorrect visualizations.

- **Solution**: **Data Cleaning**, checked for and removed null values from the dataset, formatted dates accordingly before normalization and designing the database schema.

c) **Data Overload**: Overwhelming users with the presentation of too much information. It was difficult to derive meaningful insights. For example, visualizing all protein levels for every patient across all tumor stages could result in cluttered and confusing plots.

- **Solution**: Aggregating data to show averages or counts rather than individual records helps simplify visualizations. For example, the following query aggregates protein levels by tumor stage, reducing complexity:

```
SELECT
   d.Tumour_Stage,
   AVG(pr.Protein1) AS Avg_Protein1,
   COUNT(*) AS Total_Patients
FROM Diagnoses d
JOIN Proteins pr ON d.Diagnosis_ID = pr.Diagnosis_ID
GROUP BY d.Tumour_Stage;
```

II) Elective questions 2: Illustrate how an optimized database design could enhance your visualization capabilities.

An optimized database design enhances visualization capabilities by:

1) Normalization: Structuring data into related tables minimizes redundancy and ensures efficient querying. In this case, the dataset is divided into several tables such as Patients, Diagnoses, Proteins and others.

SQL Query Example:

```
WITH ProteinSummary AS (
   SELECT
      d.Tumour_Stage,
      d.Histology,
      AVG(pr.Protein1) AS Avg_Protein1,
      AVG(pr.Protein2) AS Avg_Protein2,
      AVG(pr.Protein3) AS Avg_Protein3,
      AVG(pr.Protein4) AS Avg_Protein4,
      COUNT(*) AS Total_Patients
   FROM Diagnoses d
   JOIN Proteins pr ON d.Diagnosis_ID = pr.Diagnosis_ID
   GROUP BY d.Tumour_Stage, d.Histology
)
```

Impact: By normalizing the data into separate tables, the query can efficiently aggregate protein levels without redundant data entries, leading to faster execution and more meaningful outputs.

2) Indexing: Creating indexes on frequently queried fields improves performance for aggregations and joins, allowing for quicker access to data needed for visualizations. Indexes could be created on:

Patient_Status in the Patients table.

Diagnosis_ID in both the Diagnoses and Proteins tables.

Tumour_Stage in the Diagnoses table

SQL Query Example

In Query 2, if we were to filter by patient status dynamically, having an index on Patient_Status would significantly enhance performance:

```
SELECT
    d.Tumour_Stage,
    p.Patient_Status,
    AVG(pr.Protein1) AS Avg_Protein1,
    COUNT(*) AS Total_Patients
FROM Patients p
JOIN Diagnoses d ON p.Patient_ID = d.Patient_ID
JOIN Proteins pr ON d.Diagnosis_ID = pr.Diagnosis_ID
WHERE p.Patient_Status = 'Alive'
GROUP BY d.Tumour_Stage;
```

Impact: The index on Patient_Status allows for quicker access to relevant records, improving query performance when filtering by patient status.

3) Data Integrity: An optimized database design maintains data integrity through constraints and relationships, ensuring that visualizations reflect accurate information. Examples: Foreign key constraints between: Patients.Patient_ID and Diagnoses.Patient_ID; Diagnoses.Diagnosis_ID and Proteins.Diagnosis_ID

SQL Query Example

In Query 3, maintaining data integrity ensures that only valid relationships are queried:

```
SELECT
    d.Tumour_Stage,
    COUNT(*) AS Total_Patients
FROM Diagnoses d
JOIN Patients p ON d.Patient_ID = p.Patient_ID
GROUP BY d.Tumour_Stage;
```

Impact: The foreign key relationships ensure that every diagnosis corresponds to a valid patient, preventing orphaned records and ensuring accurate counts of patients per tumor stage.

Conclusion:

The transformation of a complex breast cancer dataset into meaningful visualizations involved addressing challenges associated with large dataset and writing strategic SQL queries for data extraction and visualization. By establishing direct connections to the database and implementing an optimized database design, the project successfully delivered insightful analytics within clinical contexts through effective visual representations.