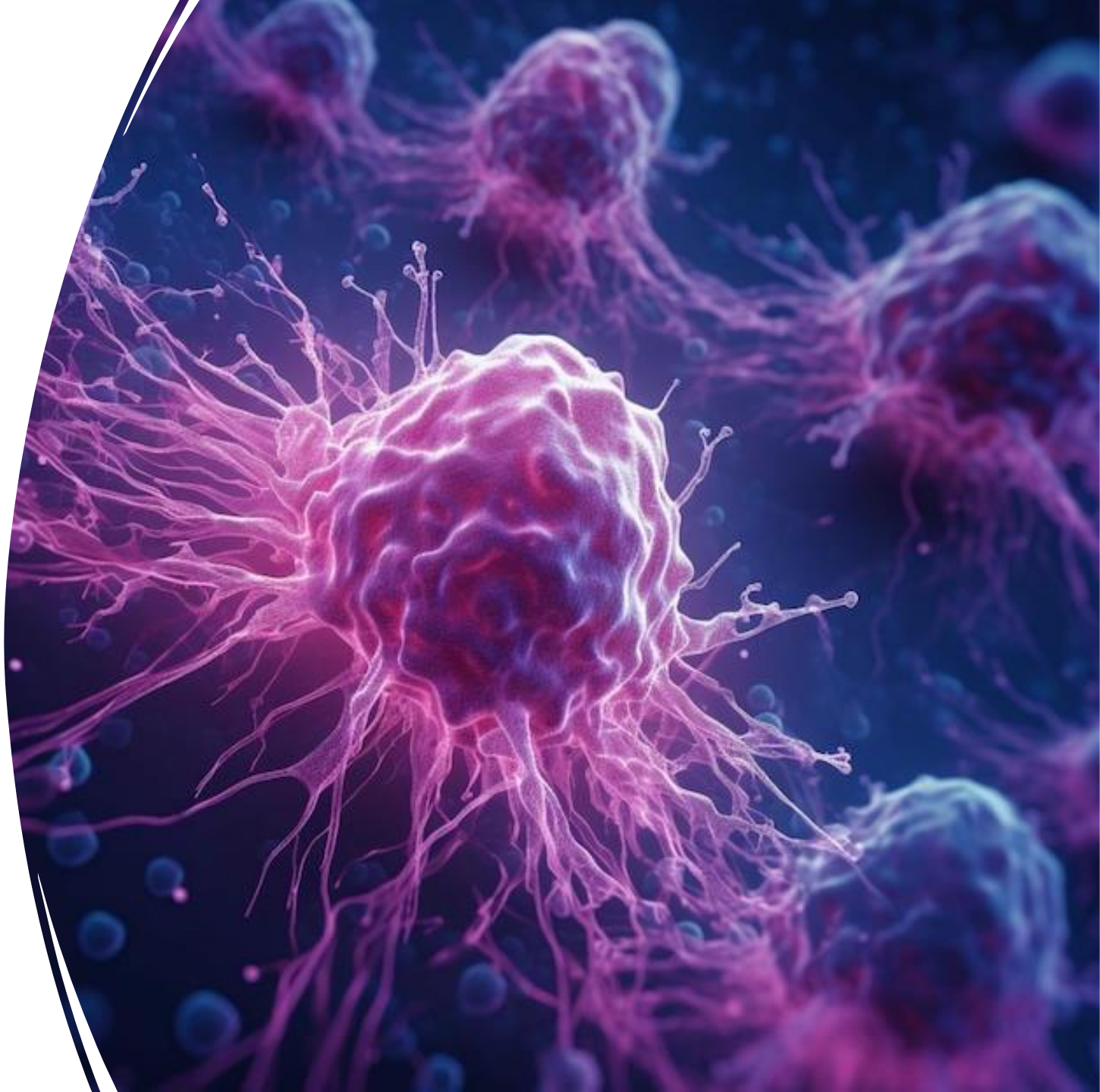
Protein Expression Patterns and Treatment Outcomes in Breast Cancer Patients

Lakshmi Menon

Lavanya Pragada

Pooja Manikonda



Background Motivation

- This project aims to investigate the relationship between protein expression levels, patient characteristics, and treatment outcomes in breast cancer. By analyzing a comprehensive dataset of breast cancer patients, we aim to identify potential biomarkers for prognosis and treatment response.
- This study is vital because understanding these relationships can lead to more effective and personalized treatment strategies for breast cancer patients.

Methods

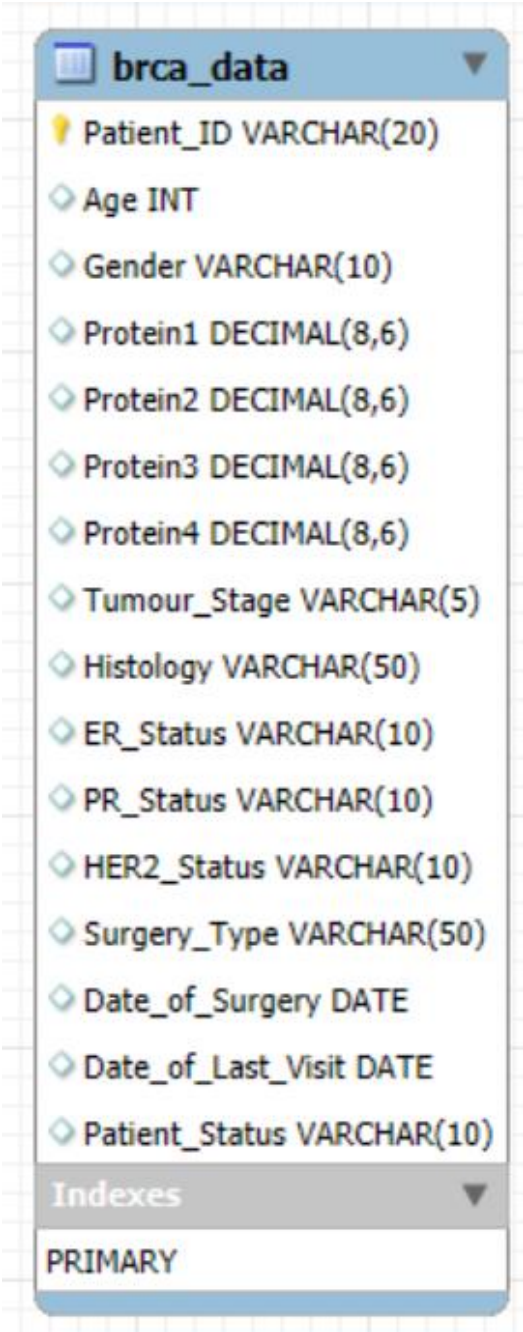
PhpMyAdmin	Database manipulation
MySQL Workbench	Database design
Pandas library	Data manipulation
Jupyter Notebook	Interactive development and result presentation
Matplotlib Seaborn	Data visualization

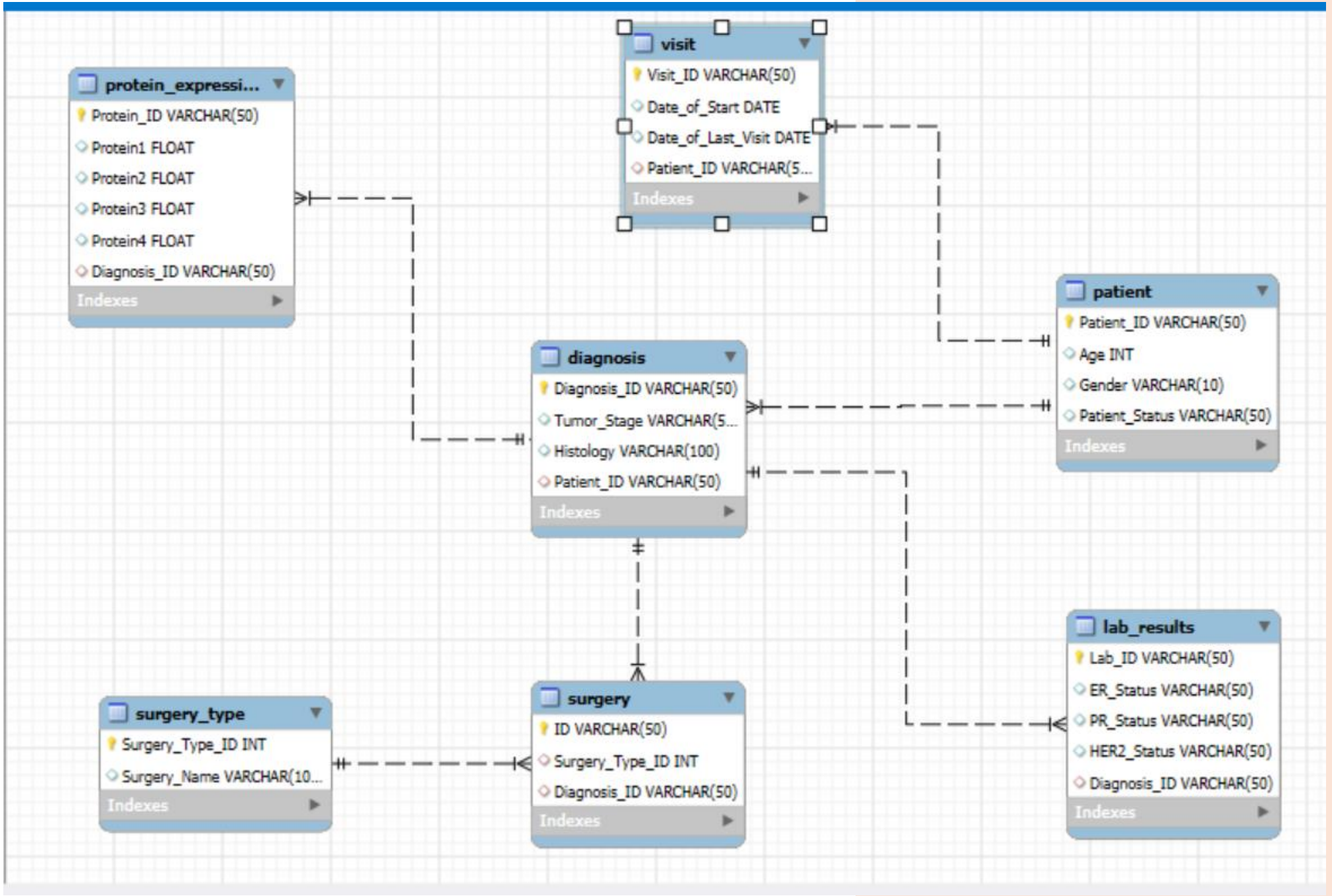
Entity-Relationships

1NF & 2NF

- The given ERD is in both 1NF and 2NF because:
- Since we have a single-column primary key (Patient_ID), there cannot be any partial dependencies. Partial dependencies can only occur with composite primary keys.

Primary Key: Patient_ID




visit
Visit_ID VARCHAR(50)
Date_of_Start DATE
Date_of_Last_Visit DATE
Patient_ID VARCHAR(5...
Indexes
protein_expressi...
Protein_ID VARCHAR(50)
Protein1 FLOAT
Protein2 FLOAT
Protein3 FLOAT
Protein4 FLOAT
Diagnosis_ID VARCHAR(50)
Indexes
patient
Patient_ID VARCHAR(50)
Age INT
Gender VARCHAR(10)
Patient_Status VARCHAR(50)
Indexes
diagnosis
Diagnosis_ID VARCHAR(50)
Tumor_Stage VARCHAR(5...
Histology VARCHAR(100)
Patient_ID VARCHAR(50)
Indexes
lab_results
Lab_ID VARCHAR(50)
ER_Status VARCHAR(50)
PR_Status VARCHAR(50)
HER2_Status VARCHAR(50)
Diagnosis_ID VARCHAR(50)
Indexes
surgery
ID VARCHAR(50)
Surgery_Type_ID INT
Diagnosis_ID VARCHAR(50)
Indexes
surgery_type
Surgery_Type_ID INT
Surgery_Name VARCHAR(10...
Indexes

3 NF

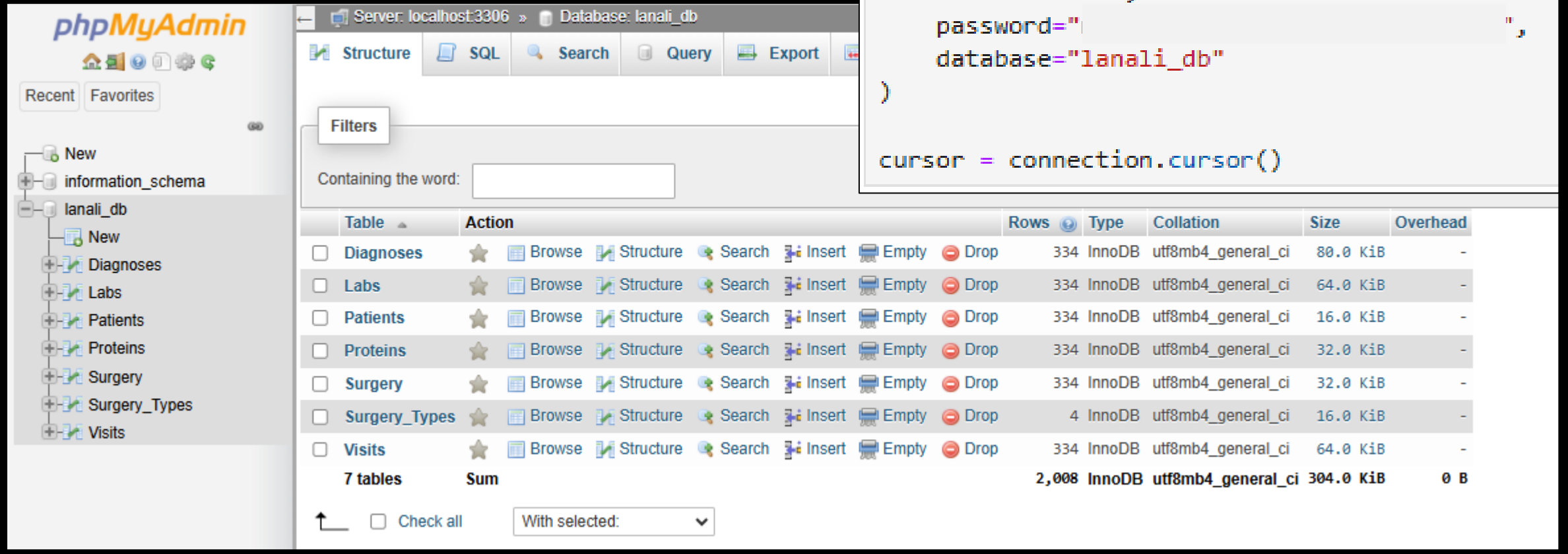
Cardinalities

One-to-One (1:1)	Many-to-One (N:1)
Patient to Visit Patient to Diagnosis Diagnosis to Surgery Diagnosis to Lab_Results Diagnosis to Protein_Expression	Surgery to Surgery_Type

TASK	TEAM MEMBER	CONTRIBUTION
DATABASE DESIGN	Lavanya Pragada	• Conceptual Design: identified main entities and established entity relationships • Logical Design: developed initial ER diagram applied normalization rules (1NF, 2NF, 3NF). • Reviewed feedback and made necessary changes to the ERD by refining relationships and cardinalities.
DATABASE IMPLEMENTATION	Lakshmi Menon	• Database schema is defined by writing SQL DDL statements. • Created database and tables; defined columns with appropriate data types and by implementing primary and foreign key constraints. • Populated the database ensuring data integrity.
DATA ANALYSIS & VISUALIZATION	Pooja Manikonda	• Established SQL database (phpMyAdmin) connection with python IDE (Jupyter Lab). • Performed data preprocessing and EDA. • Complex queries were written with results being demonstrated through visualizations.

Database Connection

```
import mysql.connector


connection = mysql.connector.connect(
    host="127.0.0.1",
    user="lanali",
    password="                    ",
    database="lanali_db"
)

cursor = connection.cursor()
```



Creation of Tables

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS Surgery (
    ID VARCHAR(10) PRIMARY KEY,
    Surgery_type_ID VARCHAR (100),
    Diagnosis_ID VARCHAR(10),
    FOREIGN KEY (Diagnosis_ID) REFERENCES Diagnoses(Diagnosis_ID)
);
""")
```

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS Visits (
    Visit_ID VARCHAR(20) PRIMARY KEY,
    Date_of_Surgery DATE,
    Date_of_Last_Visit DATE,
    Patient_ID VARCHAR(20),
    FOREIGN KEY (Patient_ID) REFERENCES Patients(Patient_ID)
);
""")
```

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS Proteins (
    ProteinID VARCHAR(20) PRIMARY KEY,
    Protein1 FLOAT,
    Protein2 FLOAT,
    Protein3 FLOAT,
    Protein4 FLOAT,
    Diagnosis_ID VARCHAR(10),
    FOREIGN KEY (Diagnosis_ID) REFERENCES Diagnoses(Diagnosis_ID)
);
""")
```

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS Patients (
    Patient_ID VARCHAR(20) PRIMARY KEY,
    Age INT,
    Gender ENUM('MALE', 'FEMALE'),
    Patient_Status ENUM('Alive', 'Dead')
);

""")
```

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS Diagnoses (
    Diagnosis_ID VARCHAR(10) PRIMARY KEY,
    Tumour_Stage VARCHAR(5),
    Histology VARCHAR(50),
    Patient_ID VARCHAR(20),
    FOREIGN KEY (Patient_ID) REFERENCES Patients(Patient_ID)
);
""")
```

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS Labs (
    Lab_ID VARCHAR(200) PRIMARY KEY,
    ER_status VARCHAR(10),
    PR_status VARCHAR(10),
    HER2_status VARCHAR(10),
    Diagnosis_ID VARCHAR(10),
    FOREIGN KEY (Diagnosis_ID) REFERENCES Diagnoses(Diagnosis_ID)
);
""")
```

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS Surgery_Types (
    Surgery_type_ID INT PRIMARY KEY,
    Surgery_name VARCHAR(100)
);
""")
```

Population of tables

```python
import csv

with open('Patients.csv', 'r') as f:
    csv_reader = csv.reader(f)
    next(csv_reader)  # Skip the header row
    for row in csv_reader:
        cursor.execute("INSERT INTO Patients (Patient_ID, Age, Gender, Patient_Status) VALUES (%s, %s, %s, %s)", row)

# Commit changes
connection.commit()
```

```python
with open('Labs.csv', 'r') as f:
    csv_reader = csv.reader(f)
    next(csv_reader)  # Skip the header row
    for row in csv_reader:
        cursor.execute("""
            INSERT INTO Labs (Lab_ID, ER_status, PR_status, HER2_status, Diagnosis_ID)
            VALUES (%s, %s, %s, %s, %s)
        """, row)

# Commit changes
connection.commit()
```

```python
with open('Proteins.csv', 'r') as f:
    csv_reader = csv.reader(f)
    next(csv_reader)  # Skip the header row
    for row in csv_reader:
        cursor.execute("""
            INSERT INTO Proteins (ProteinID, Protein1, Protein2, Protein3, Protein4, Diagnosis_ID)
            VALUES (%s, %s, %s, %s, %s, %s)
        """, row)

# Commit changes
connection.commit()
```

```python
# Open the CSV file and insert data
with open('Surgery.csv', 'r') as f:
    csv_reader = csv.reader(f)
    next(csv_reader)  # Skip the header row
    for row in csv_reader:
        # Ensure correct mapping of columns
        id_ = row[0].strip()  # ID (string, e.g., 'S1', 'S2')
        surgery_type_id = row[1].strip()  # Surgery_type_ID (string, e.g., '1', '2')
        diagnosis_id = row[2].strip()  # Diagnosis_ID (string, e.g., 'D1', 'D2')

        # Insert the data into the table
        cursor.execute("""
            INSERT INTO Surgery (ID, Surgery_type_ID, Diagnosis_ID)
            VALUES (%s, %s, %s)
        """, (id_, surgery_type_id, diagnosis_id))

# Commit the changes
connection.commit()
```

Population of tables

```
cursor = connection.cursor()

# Insert data into the Surgery_Types table
insert_queries = [
    "INSERT INTO Surgery_Types (Surgery_type_ID, Surgery_name) VALUES (1, 'Modified Radical Mastectomy')",
    "INSERT INTO Surgery_Types (Surgery_type_ID, Surgery_name) VALUES (2, 'Lumpectomy')",
    "INSERT INTO Surgery_Types (Surgery_type_ID, Surgery_name) VALUES (3, 'Simple Mastectomy')",
    "INSERT INTO Surgery_Types (Surgery_type_ID, Surgery_name) VALUES (4, 'Other')"
]

for query in insert_queries:
    cursor.execute(query)

# Commit changes
connection.commit()
```

```
with open('Diagnoses.csv', 'r') as f:
    csv_reader = csv.reader(f)
    next(csv_reader)  # Skip the header row
    for row in csv_reader:
        cursor.execute("""
            INSERT INTO Diagnoses (Diagnosis_ID, Tumour_Stage, Histology, Patient_ID)
            VALUES (%s, %s, %s, %s)
        """, row)

# Commit changes
connection.commit()
```

```
with open('Visits.csv', 'r') as f:
    csv_reader = csv.reader(f)
    next(csv_reader)  # Skip the header row
    for row in csv_reader:
        visit_id = row[0]

        # Handle missing or empty dates
        date_of_surgery = row[1].strip()  # Remove extra spaces
        date_of_surgery = (
            datetime.strptime(date_of_surgery, '%d-%m-%Y').strftime('%Y-%m-%d')
            if date_of_surgery else None
        )

        date_of_last_visit = row[2].strip()
        date_of_last_visit = (
            datetime.strptime(date_of_last_visit, '%d-%m-%Y').strftime('%Y-%m-%d')
            if date_of_last_visit else None
        )

        patient_id = row[3]

        # Properly indented cursor.execute block
        cursor.execute("""
            INSERT IGNORE INTO Visits (Visit_ID, Date_of_Surgery, Date_of_Last_Visit, Patient_ID)
            VALUES (%s, %s, %s, %s)
        """, (visit_id, date_of_surgery, date_of_last_visit, patient_id))

# Commit changes
connection.commit()
```

QUERIES

QUERY 1

```
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;
"""

# Execute the query and fetch the results
cursor.execute(query)
results = cursor.fetchall()

# Get the column names
columns = [desc[0] for desc in cursor.description]

# Create a pandas DataFrame
df = pd.DataFrame(results, columns=columns)

# Display the DataFrame
print(df)
```

Result:

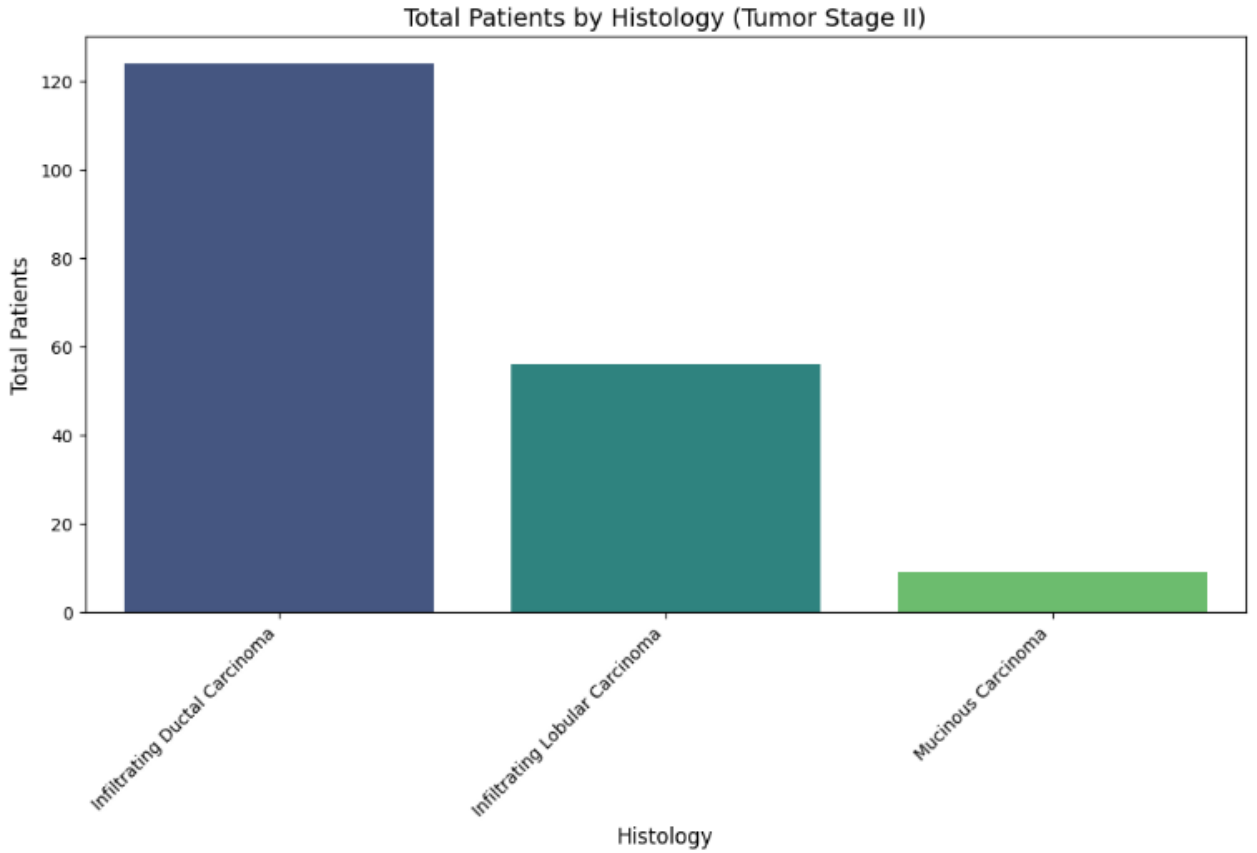
	Tumour_Stage	Histology	Avg_Protein1	Avg_Protein2	\
0	II	Infiltrating Ductal Carcinoma	-0.043145	0.941891	
1	II	Infiltrating Lobular Carcinoma	0.051042	1.073106	
2	II	Mucinous Carcinoma	0.114430	0.605754	

	Avg_Protein3	Avg_Protein4	Total_Patients	Stage_Rank
0	-0.041731	0.006162	124	1
1	-0.154017	0.018415	56	1
2	0.159706	0.178998	9	1

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

```
data = {
    "Tumour_Stage": ["II", "II", "II"],
    "Histology": ["Infiltrating Ductal Carcinoma", "Infiltrating Lobular Carcinoma", "Mucinous Carcinoma"],
    "Avg_Protein1": [-0.043145, 0.051042, 0.114430],
    "Avg_Protein2": [0.941891, 1.073106, 0.605754],
    "Avg_Protein3": [-0.041731, -0.154017, 0.159706],
    "Avg_Protein4": [0.006162, 0.018415, 0.178998],
    "Total_Patients": [124, 56, 9],
    "Stage_Rank": [1, 1, 1],
}
df = pd.DataFrame(data)

# Bar plot for Total Patients by Histology
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x="Histology", y="Total_Patients", palette="viridis")
plt.title("Total Patients by Histology (Tumor Stage II)", fontsize=14)
plt.xlabel("Histology", fontsize=12)
plt.ylabel("Total Patients", fontsize=12)
plt.xticks(rotation=45, ha="right")
plt.show()
```

Visualization:



QUERY 2

```python
# Indexing SQL commands
indexing_commands = [
    "CREATE INDEX idx_patient_status ON Patients (Patient_Status);",
    "CREATE INDEX idx_diagnosis_id ON Diagnoses (Diagnosis_ID);",
    "CREATE INDEX idx_tumour_stage ON Diagnoses (Tumour_Stage);",
    "CREATE INDEX idx_protein_diagnosis_id ON Proteins (Diagnosis_ID);"
]

# Execute each indexing command
for command in indexing_commands:
    try:
        cursor.execute(command)
        print(f"Index created successfully: {command}")
    except mysql.connector.Error as err:
        print(f"Error creating index: {err}")

# Commit the changes
connection.commit()
```

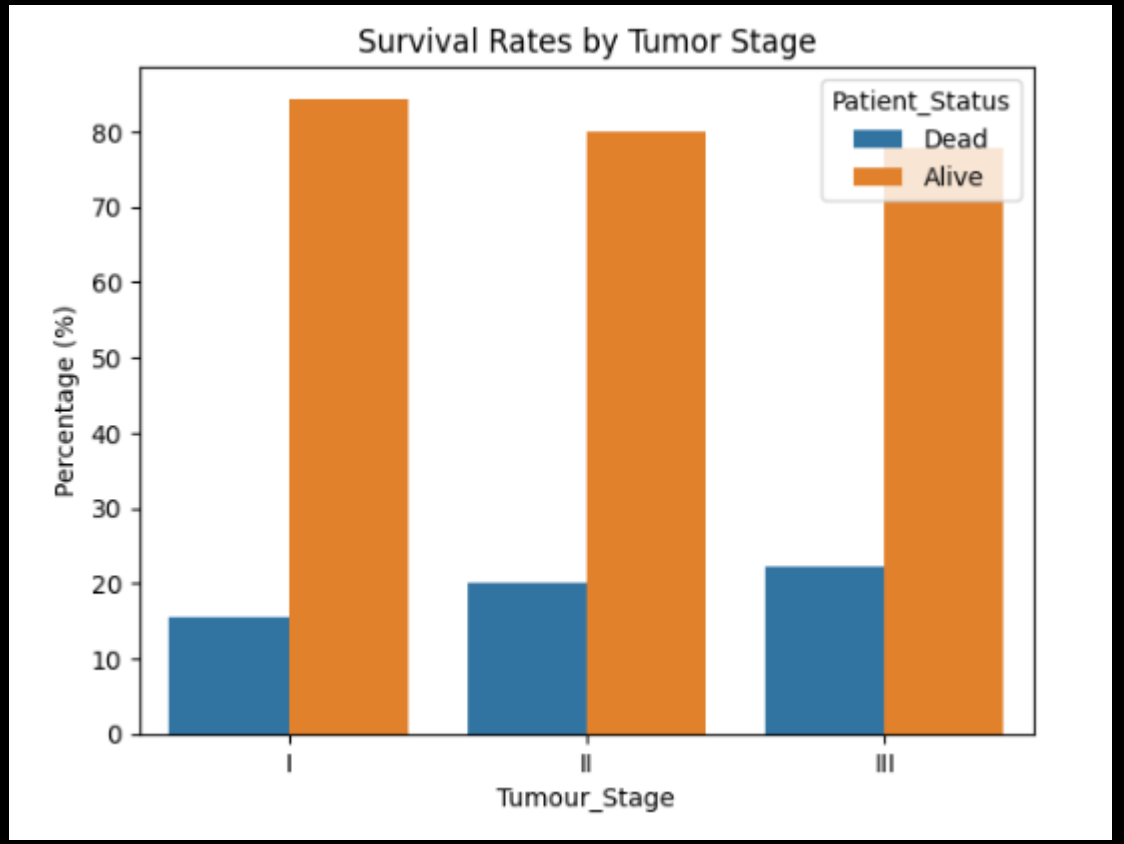
```python
query = """
WITH ProteinSurvival AS (
    SELECT
        d.Tumour_Stage,
        p.Patient_Status,
        AVG(pr.Protein1) AS Avg_Protein1,
        AVG(pr.Protein2) AS Avg_Protein2,
        AVG(pr.Protein3) AS Avg_Protein3,
        AVG(pr.Protein4) AS Avg_Protein4,
        COUNT(*) AS Total_Patients
    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;
"""
```

Result:

	Tumour_Stage	Patient_Status	Avg_Protein1	Avg_Protein2	Avg_Protein3	Avg_Protein4	Total_Patients	Percentage
0	I	Dead	0.009211	1.149777	-0.211212	-0.126490	10	15.63
1	I	Alive	-0.018808	0.973826	-0.156616	0.068257	54	84.38
2	II	Dead	-0.098867	1.139979	-0.017360	0.096950	38	20.11
3	II	Alive	0.015200	0.920669	-0.077501	-0.001839	151	79.89
4	III	Dead	0.042052	0.998399	-0.058217	0.254550	18	22.22
5	III	Alive	-0.133154	0.823295	-0.097596	-0.113168	63	77.78

Visualization:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Bar plot for survival rates by tumor stage
sns.barplot(x='Tumour_Stage', y='Percentage', hue='Patient_Status', data=df)
plt.title('Survival Rates by Tumor Stage')
plt.ylabel('Percentage (%)')
plt.xlabel('Tumour_Stage')
plt.show()
```



QUERY 3

```
query = """
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)

# Save to CSV (optional)
df.to_csv('Protein_Tumour_Stage.csv', index=False)
```

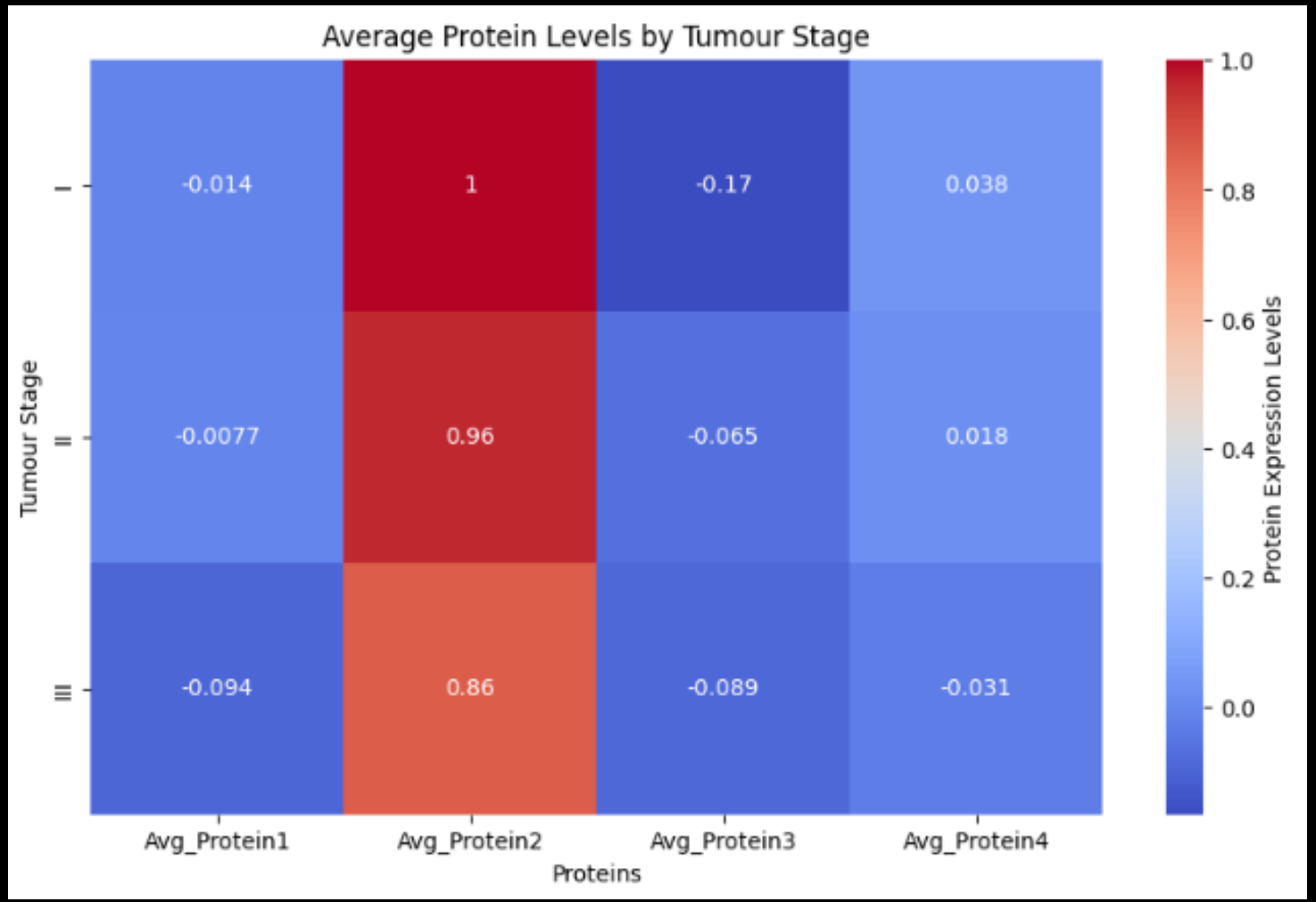
Result:

	Tumour_Stage	Avg_Protein1	Avg_Protein2	Avg_Protein3	Avg_Protein4	\
0	I	-0.014430	1.001318	-0.165147	0.037828	
1	II	-0.007734	0.964763	-0.065409	0.018023	
2	III	-0.094220	0.862207	-0.088845	-0.031453	

	Total_Patients
0	64
1	189
2	81

Visualization:

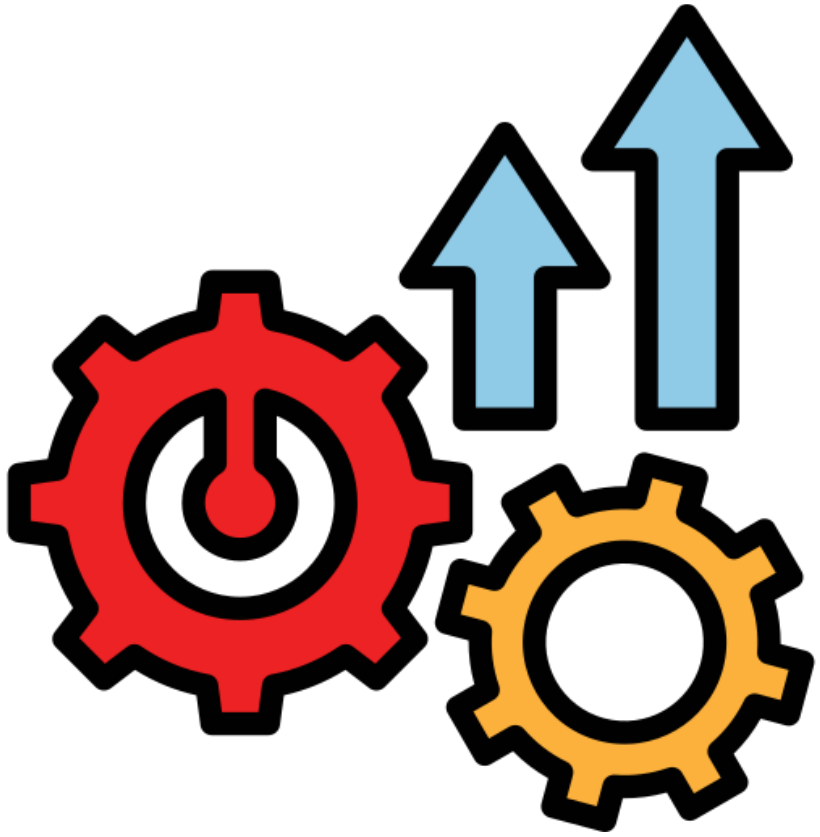
```
protein_df = df[['Tumour_Stage', 'Avg_Protein1', 'Avg_Protein2', 'Avg_Protein3', 'Avg_Protein4']]
protein_df.set_index('Tumour_Stage', inplace=True)

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()
```



Challenges Encountered

Dataset Limitation

- Initial dataset had only Patient_ID as the primary key.
- Insufficient for table decomposition; additional keys like Diagnosis_ID and Visit_ID were created.
- Schema redesign required extra effort but improved database structure.



Improvements

- Select datasets with richer attributes and pre-defined keys.
- Optimize schema design and relationships in the ERD.
- Normalize tables further (e.g., BCNF) to reduce redundancy.
- Add stricter data validation constraints (FOREIGN KEY, CHECK).
- Enhance query performance with indexing and optimization techniques.
- Introduce error-logging mechanisms for smoother debugging.
- Implement automated backup and recovery solutions.