Emergency Room Visit Analysis: A Database Management Perspective

Info-B 512
Scientific and Clinical Data Management

Instructor

Yan Zhuang, Ph.D.

Authors

Sri Ramya Panja

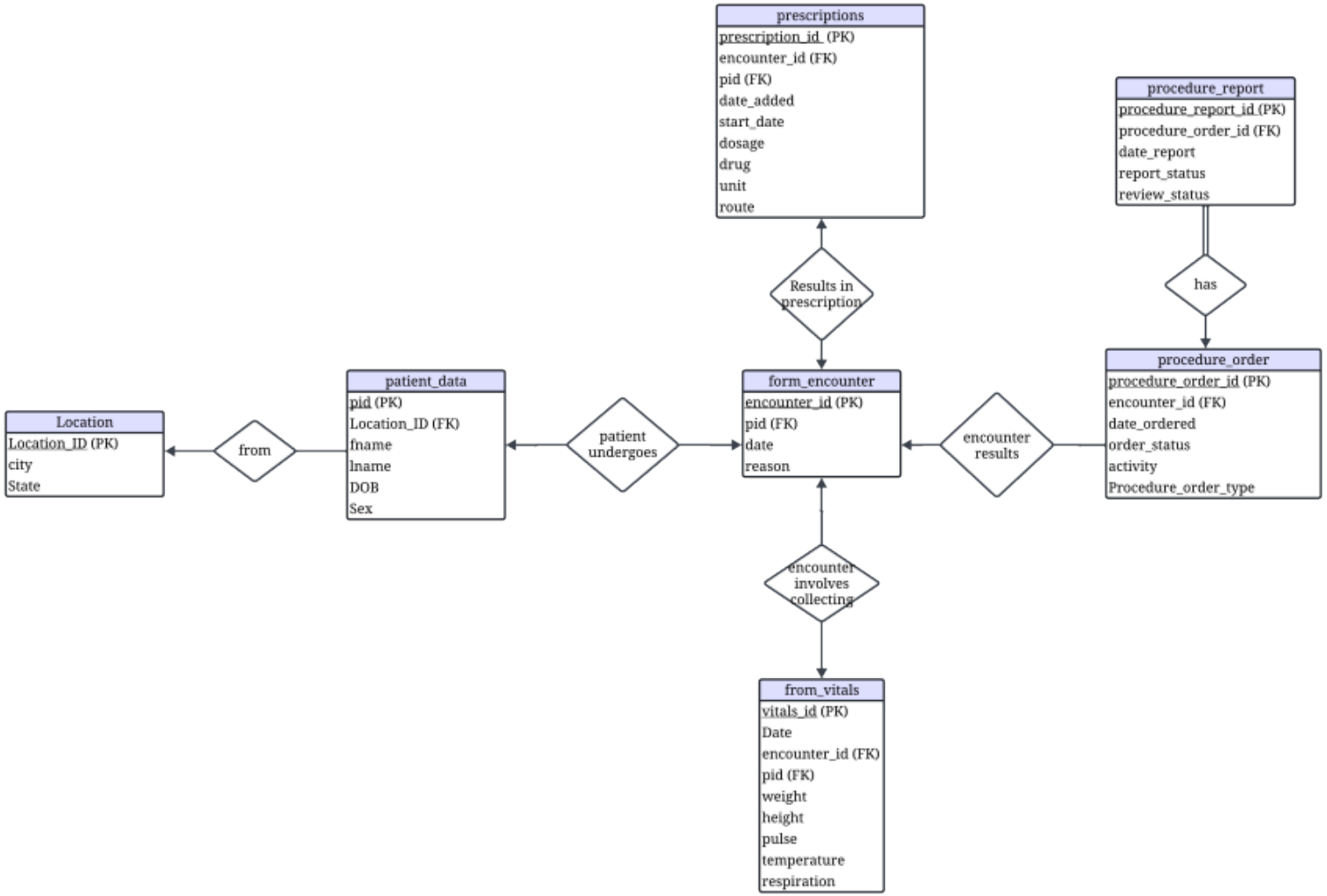
Srihari Myla Venkata

Introduction

Emergency rooms (ER) are the frontline for treating urgent and critical health conditions. Analyzing data from ER visits provides critical insights into patient health, medical interventions, and follow-up care. This analysis focuses on understanding the vital signs recorded during ER visits, the types of procedures performed, and the follow-up prescriptions given to patients after their emergency care.

Emergency room data analysis is crucial for improving patient outcomes, optimizing hospital resources, and ensuring effective care delivery by reviewing the key elements of each ER visit, such as patient demographics, vitals, procedures, and medications. Healthcare providers can better understand the effectiveness of ER processes and improve patient care quality, efficiency, and safety.

Updated ER-diagram



Methodology

Data cleaning

SQL connection

Table creation

Data insertion

Data analysis

Data Cleaning

Cleaned the extracted OpenEMR data using Python.

Patient Data:
•Kept pid, fname, lname, DOB, sex.
•Added LocationID (Foreign Key).
Location Data:
•Created location.csv with LocationID, city, and state.
Encounter Data:
•Kept encounter_id, pid, date, and reason.
Prescription and Vitals:
•Retained essential columns.
•Linked to encounters (one-to-one) based on patient ID.
•Dropped rows with missing vital signs.
Procedure Data:
•Linked orders and reports through procedure_order_id.

Cleaned Patient Data:

	pid	fname	lname	DOB	sex	LocationID
0	1	Abdul	Kuvalis	1995-04-23	Male	1
1	2	Adrienne	Graham	1953-06-19	Female	2
2	3	Ai	Gerlach	1996-07-10	Female	3
3	4	Aida	Marks	2016-03-20	Female	4
4	5	Althea	Schmeler	1974-07-28	Female	5

Location Data:

	LocationID	city	state
0	1	West Springfield	Massachusetts
1	2	Haverhill	Massachusetts
2	3	Worcester	Massachusetts
3	4	Haverhill	Massachusetts
4	5	Milford	Massachusetts

Cleaned Form Encounter Data:

	encounter_id	pid	date	reason
0	1	1	2002-05-07 05:44:10	Encounter for problem
1	2	1	2011-06-05 05:44:10	Well child visit (procedure)
2	3	1	2013-06-16 05:44:10	General examination of patient (procedure)
3	4	1	2014-06-22 05:44:10	General examination of patient (procedure)
4	5	1	2014-12-16 06:54:14	Encounter for problem

Fixed Cleaned Prescriptions:

	prescriptionID	pid	date_added	dosage	drug	unit	route	encounter_id
0	1	1	2015-05-29 23:44:10	1	Amoxicillin 250 MG / Clavulanate 125 MG Oral T...	10	16	1
1	2	1	2016-03-28 07:39:01	1	Acetaminophen 325 MG Oral Tablet	11	16	1
2	3	2	1959-02-17 15:26:14	1	diphenhydrAMINE Hydrochloride 25 MG Oral Tablet	12	16	14
3	4	2	1972-09-05 17:26:14	1	doxycycline hyclate 100 MG	13	16	14
4	5	2	2020-09-25 15:26:14	1	lisinopril 10 MG Oral Tablet	14	16	14

Fixed Cleaned Form Vitals:

	vital_id	pid	date	weight	height	temperature	pulse	respiration	encounter_id
4	5	5	2022-11-13 19:23:04	171.33	63.66	100.34	79.0	13.0	181
6	7	7	2022-12-07 12:40:03	127.69	63.98	99.28	96.0	13.0	252
25	26	28	2022-11-05 00:35:14	170.74	63.94	99.65	80.0	14.0	1163
49	50	56	2022-07-08 20:08:49	186.62	65.79	99.19	94.0	15.0	2313
55	56	62	2022-08-26 22:41:40	183.45	68.27	99.43	93.0	14.0	2663

Cleaned Procedure Order Data:

	procedure_order_id	encounter_id	date_ordered	order_status	activity	procedure_order_type
0	1	4	2013-06-16 05:44:10	completed	1	order
1	2	4	2013-06-16 06:23:23	completed	1	order
2	3	4	2013-06-16 06:37:55	completed	1	order
3	4	4	2013-06-16 06:59:34	completed	1	order
4	5	4	2013-06-16 07:12:54	completed	1	order

SQL connection

A connection between Python and MySQL was established using the mysql-connector-python library by providing the host, username, password, and database name. A cursor was created to execute SQL commands for creating tables and inserting cleaned data. After completing all operations, the cursor and connection were closed to maintain database integrity.

```
!pip install mysql-connector-python
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: mysql-connector-python in ./.local/lib/python3.10/site-packages (9.2.0)

[notice] A new release of pip is available: 23.2.1 -> 25.1
[notice] To update, run: python3 -m pip install --upgrade pip
```

```
#import package, if it does not work, restart the console and re-run this command.
import mysql.connector
```

```
#input the credentials
connection = mysql.connector.connect(
    host="127.0.0.1",  # Hostname
    user="srmyla",   # Username
    password="abodes hived malraux demeter",  # Password
    database="srmyla_db"  # Database name
)

cursor = connection.cursor()
```

Table Creation

Tables were created in MySQL to organize the cleaned data into a structured relational format. Each table was designed with appropriate primary keys to ensure uniqueness and foreign keys to maintain referential integrity between related tables. The Patient, Location, Form_Encounter, Prescription, Vitals, Procedure_Order, and Procedure_Report tables were created following a normalized schema. Data types were carefully assigned to match the nature of each field, and constraints were enforced to avoid duplication and maintain consistency across the database.

```
# Create Location table
cursor.execute('''
CREATE TABLE IF NOT EXISTS Location (
    LocationID INT PRIMARY KEY,
    city VARCHAR(100) NOT NULL,
    state VARCHAR(100) NOT NULL
);
''')

# Create Patient table
cursor.execute('''
CREATE TABLE IF NOT EXISTS Patient (
    pid INT PRIMARY KEY,
    fname VARCHAR(100) NOT NULL,
    lname VARCHAR(100) NOT NULL,
    DOB DATE NOT NULL,
    sex VARCHAR(10) NOT NULL,
    LocationID INT,
    FOREIGN KEY (LocationID) REFERENCES Location(LocationID)
);
''')

# Create Form_Encounter table
cursor.execute('''
CREATE TABLE IF NOT EXISTS Form_Encounter (
    encounter_id INT PRIMARY KEY,
    pid INT,
    date DATE,
    reason VARCHAR(255),
    FOREIGN KEY (pid) REFERENCES Patient(pid)
);
''')
```

```
# Create Procedure_Order table
cursor.execute('''
CREATE TABLE IF NOT EXISTS Procedure_Order (
    procedure_order_id INT PRIMARY KEY,
    encounter_id INT,
    date_ordered DATE,
    order_status VARCHAR(100),
    activity VARCHAR(100),
    procedure_order_type VARCHAR(100),
    FOREIGN KEY (encounter_id) REFERENCES Form_Encounter(encounter_id)
);
''')

# Create Procedure_Report table
cursor.execute('''
CREATE TABLE IF NOT EXISTS Procedure_Report (
    procedure_report_id INT PRIMARY KEY,
    procedure_order_id INT,
    date_report DATE,
    report_status VARCHAR(100),
    review_status VARCHAR(100),
    FOREIGN KEY (procedure_order_id) REFERENCES Procedure_Order(procedure_order_id)
);
''')
```

Data Insertion

Cleaned CSV files were loaded into the corresponding MySQL tables using Python. Each record was inserted carefully while maintaining primary and foreign key relationships to preserve database integrity. Insert operations were performed in the correct order — starting with independent tables like Location and Patient, followed by dependent tables like Form_Encounter, Prescription, Vitals, Procedure_Order, and Procedure_Report. This structured insertion ensured that all relational links were correctly established without violating any constraints.

```python
# Load CSV
location_df = pd.read_csv('location.csv')

# Insert into Location table using INSERT IGNORE
for _, row in location_df.iterrows():
    cursor.execute('''
        INSERT IGNORE INTO Location (LocationID, city, state)
        VALUES (%s, %s, %s)
    ''', (row['LocationID'], row['city'], row['state']))

# Commit changes (use your existing connection)
connection.commit()

print("✅ Location data inserted successfully without duplicates!")
```

```
✅ Location data inserted successfully without duplicates!
```

```python
# Load the cleaned patient data
patient_df = pd.read_csv('cleaned_patient_data.csv')

# Insert into Patient table safely
for _, row in patient_df.iterrows():
    cursor.execute('''
        INSERT IGNORE INTO Patient (pid, fname, lname, DOB, sex, LocationID)
        VALUES (%s, %s, %s, %s, %s, %s)
    ''', (row['pid'], row['fname'], row['lname'], row['DOB'], row['sex'], row['LocationID']))

# Commit changes
connection.commit()
```

```python
encounter_df = pd.read_csv('cleaned_form_encounter.csv')

for _, row in encounter_df.iterrows():
    cursor.execute('''
        INSERT IGNORE INTO Form_Encounter (encounter_id, pid, date, reason)
        VALUES (%s, %s, %s, %s)
    ''', (row['encounter_id'], row['pid'], row['date'], row['reason']))

connection.commit()
```

```python
prescription_df = pd.read_csv('cleaned_prescriptions.csv')

for _, row in prescription_df.iterrows():
    cursor.execute('''
        INSERT IGNORE INTO Prescription (prescriptionID, pid, date_added, dosage, drug, unit, route, encounter_id)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
    ''', (row['prescriptionID'], row['pid'], row['date_added'], row['dosage'], row['drug'], row['unit'], row['route'], row['encounter_id']))

connection.commit()
```

```python
vitals_df = pd.read_csv('cleaned_form_vitals.csv')

for _, row in vitals_df.iterrows():
    cursor.execute('''
        INSERT IGNORE INTO Vitals (vital_id, pid, date, weight, height, temperature, pulse, respiration, encounter_id)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
    ''', (row['vital_id'], row['pid'], row['date'], row['weight'], row['height'], row['temperature'], row['pulse'], row['respiration'], row['encounter_id

connection.commit()
```

```python
procedure_order_df = pd.read_csv('cleaned_procedure_order.csv')

for _, row in procedure_order_df.iterrows():
    cursor.execute('''
        INSERT IGNORE INTO Procedure_Order (procedure_order_id, encounter_id, date_ordered, order_status, activity, procedure_order_type)
        VALUES (%s, %s, %s, %s, %s, %s)
    ''', (row['procedure_order_id'], row['encounter_id'], row['date_ordered'], row['order_status'], row['activity'], row['procedure_order_type']))

connection.commit()
```

```python
procedure_report_df = pd.read_csv('cleaned_procedure_report.csv')

for _, row in procedure_report_df.iterrows():
    cursor.execute('''
        INSERT IGNORE INTO Procedure_Report (procedure_report_id, procedure_order_id, date_report, report_status, review_status)
        VALUES (%s, %s, %s, %s, %s)
    ''', (row['procedure_report_id'], row['procedure_order_id'], row['date_report'], row['report_status'], row['review_status']))

connection.commit()
```

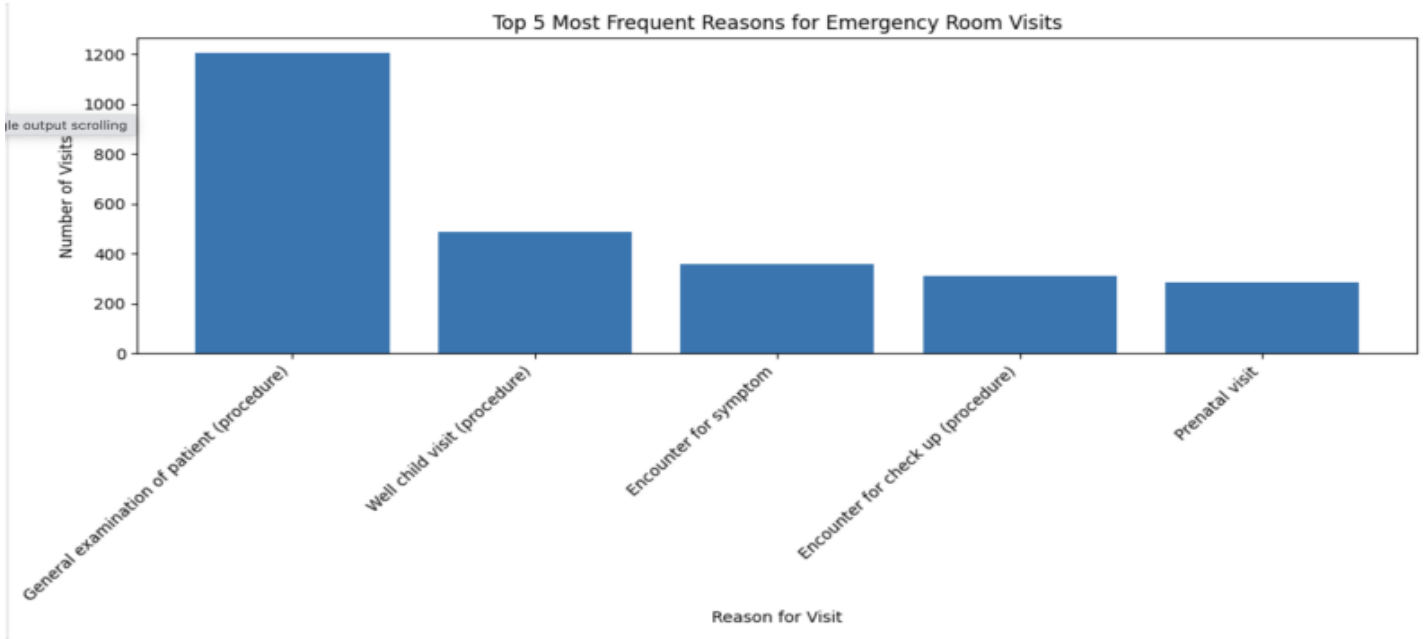
Complex Queries & Visualizations

What are the most frequent reasons for emergency room visits?

The most frequent reasons for emergency room visits were:

- General examinations
- Well-child visits
- Symptom encounters
- Check-ups
- Prenatal visits

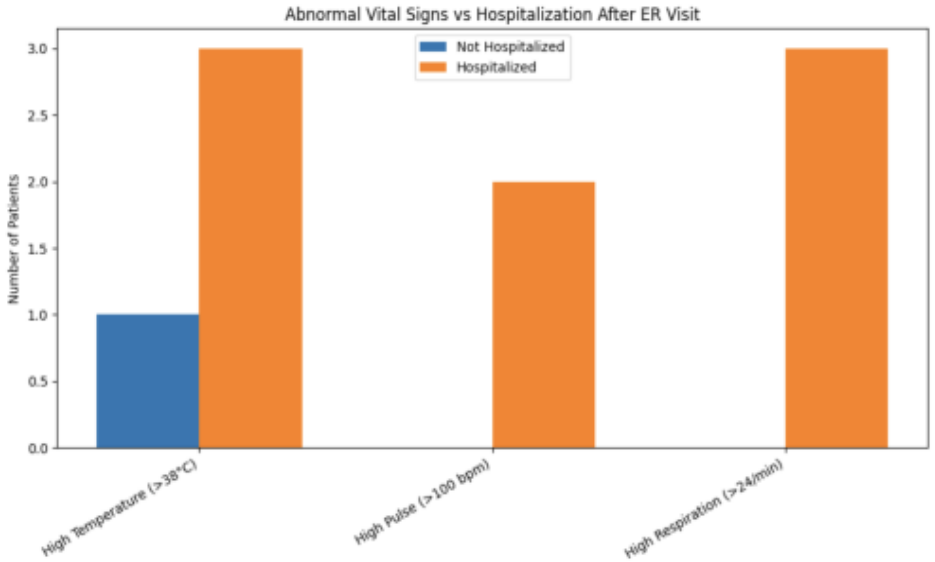
	Reason	Visit_Count
0	General examination of patient (procedure)	1203
1	Well child visit (procedure)	488
2	Encounter for symptom	359
3	Encounter for check up (procedure)	309
4	Prenatal visit	287




```
cursor.execute('''
    SELECT reason, COUNT(*) AS visit_count
    FROM Form_Encounter
    GROUP BY reason
    ORDER BY visit_count DESC
    LIMIT 5;
''')
```

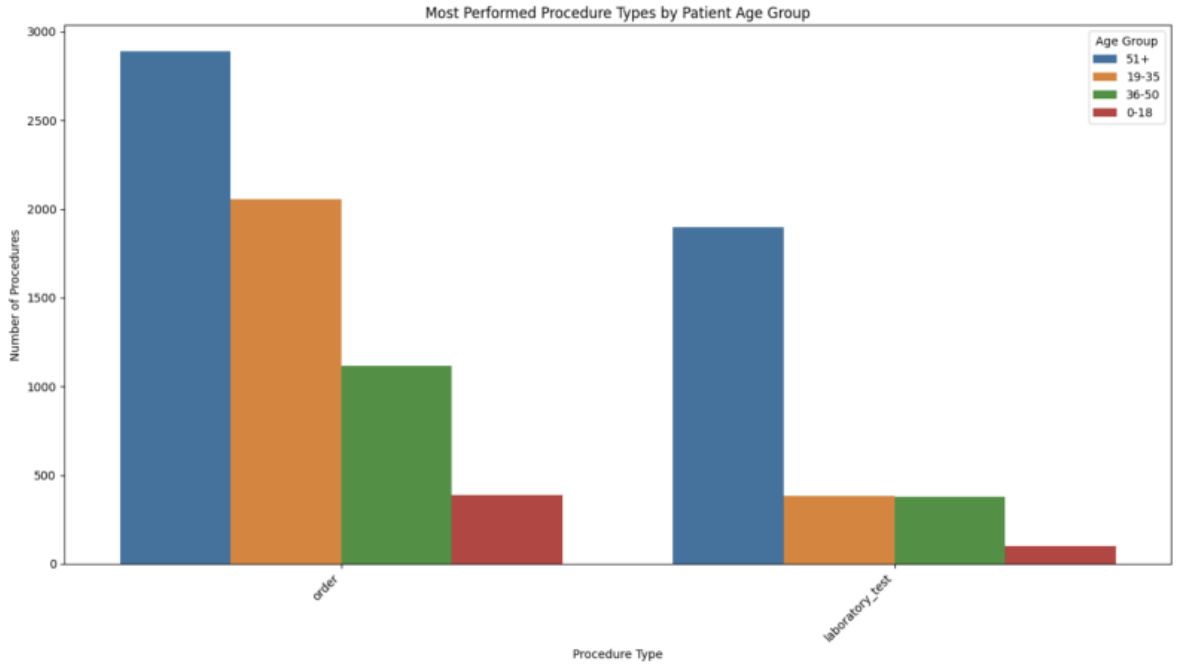
Are there patterns in patient vital signs that indicate a higher risk of hospitalization after an ER visit?

- The analysis considered abnormal values for temperature (>38°C), pulse (>100 bpm), and respiration rate (>24/min).
- More hospitalized patients showed abnormal vitals compared to non-hospitalized patients.
- Specifically, hospitalized patients had higher counts of elevated pulse and respiration.
- This suggests a possible association between abnormal vitals and hospitalization in this dataset, unlike the earlier result.




```python
# Create flags for abnormal vitals
merged_df['High_Temperature'] = merged_df['temperature'] > 38
merged_df['High_Pulse'] = merged_df['pulse'] > 100
merged_df['High_Respiration'] = merged_df['respiration'] > 24

# Flag hospitalization based on reason
merged_df['Hospitalized'] = merged_df['reason'].str.contains('Admission', case=False, na=False)

#Summarize abnormal vitals by hospitalization status
abnormal_vitals_summary = merged_df.groupby('Hospitalized')[['High_Temperature', 'High_Pulse', 'High_Respiration']].sum().reset_index()

#  Prepare for visualization
hospitalized = abnormal_vitals_summary.loc[abnormal_vitals_summary['Hospitalized'] == True]
not_hospitalized = abnormal_vitals_summary.loc[abnormal_vitals_summary['Hospitalized'] == False]

# Handle missing groups
if hospitalized.empty:
    hospitalized = pd.DataFrame({'High_Temperature': [0], 'High_Pulse': [0], 'High_Respiration': [0]})
if not_hospitalized.empty:
    not_hospitalized = pd.DataFrame({'High_Temperature': [0], 'High_Pulse': [0], 'High_Respiration': [0]})

abnormal_counts = {
    'High Temperature (>38°C)': [not_hospitalized['High_Temperature'].values[0], hospitalized['High_Temperature'].values[0]],
    'High Pulse (>100 bpm)': [not_hospitalized['High_Pulse'].values[0], hospitalized['High_Pulse'].values[0]],
    'High Respiration (>24/min)': [not_hospitalized['High_Respiration'].values[0], hospitalized['High_Respiration'].values[0]],
}

# Plotting
x = np.arange(len(abnormal_counts))
width = 0.35

fig, ax = plt.subplots(figsize=(10,6))
rects1 = ax.bar(x - width/2, [v[0] for v in abnormal_counts.values()], width, label='Not Hospitalized')
rects2 = ax.bar(x + width/2, [v[1] for v in abnormal_counts.values()], width, label='Hospitalized')

ax.set_ylabel('Number of Patients')
ax.set_title('Abnormal Vital Signs vs Hospitalization After ER Visit')
ax.set_xticks(x)
ax.set_xticklabels(abnormal_counts.keys(), rotation=30, ha='right')
ax.legend()

fig.tight_layout()
plt.show()
```

Which procedures are most performed in the emergency room, and how do they vary by patient age group?




```
cursor.execute('''
    SELECT
        po.procedure_order_type,
        CASE
            WHEN TIMESTAMPDIFF(YEAR, pa.DOB, CURDATE()) <= 18 THEN '0-18'
            WHEN TIMESTAMPDIFF(YEAR, pa.DOB, CURDATE()) <= 35 THEN '19-35'
            WHEN TIMESTAMPDIFF(YEAR, pa.DOB, CURDATE()) <= 50 THEN '36-50'
            ELSE '51+'
        END AS Age_Group,
        COUNT(*) AS Procedure_Count
    FROM
        Procedure_Order po
    JOIN
        Form_Encounter fe ON po.encounter_id = fe.encounter_id
    JOIN
        Patient pa ON fe.pid = pa.pid
    WHERE
        po.procedure_order_type IS NOT NULL
    GROUP BY
        po.procedure_order_type, Age_Group
    ORDER BY
        Procedure_Count DESC;
''')
```

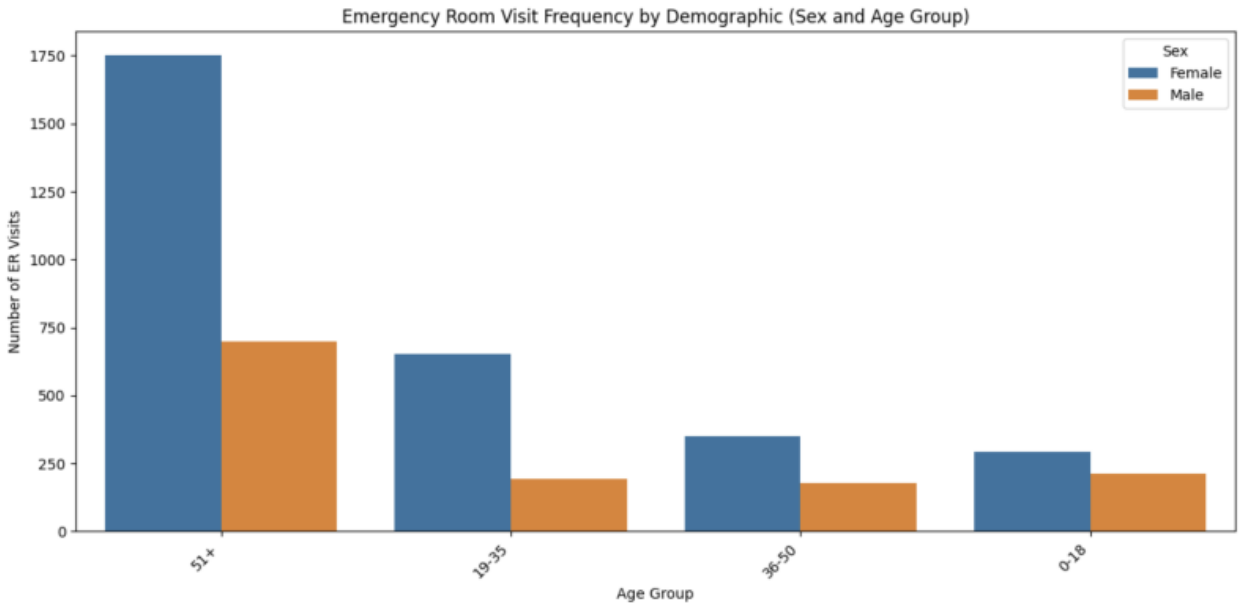
	Procedure_Order_Type	Age_Group	Procedure_Count
0	order	51+	2892
1	order	19-35	2054
2	laboratory_test	51+	1898
3	order	36-50	1115
4	order	0-18	386
5	laboratory_test	19-35	383
6	laboratory_test	36-50	376
7	laboratory_test	0-18	99

- The most performed procedures in the emergency room are orders followed by laboratory tests.
- These patterns are consistently seen across all patient age groups (0–18, 19–35, 36–50, and 51+).

Are there differences in emergency room visit frequency based on demographics?

- We considered the demographics, like **sex** and **age,** for this analysis.
- Females had a higher number of emergency room visits compared to **males** across all age groups.
- The **51+** age group had the highest ER visit frequency, with females significantly outnumbering males.
- In the younger age groups (0–18, 19–35, 36–50), females also consistently visited the ER more often than males.

	Sex	Age_Group	Visit_Count
0	Female	51+	1751
1	Male	51+	698
2	Female	19-35	652
3	Female	36-50	349
4	Female	0-18	291
5	Male	0-18	212
6	Male	19-35	194
7	Male	36-50	176



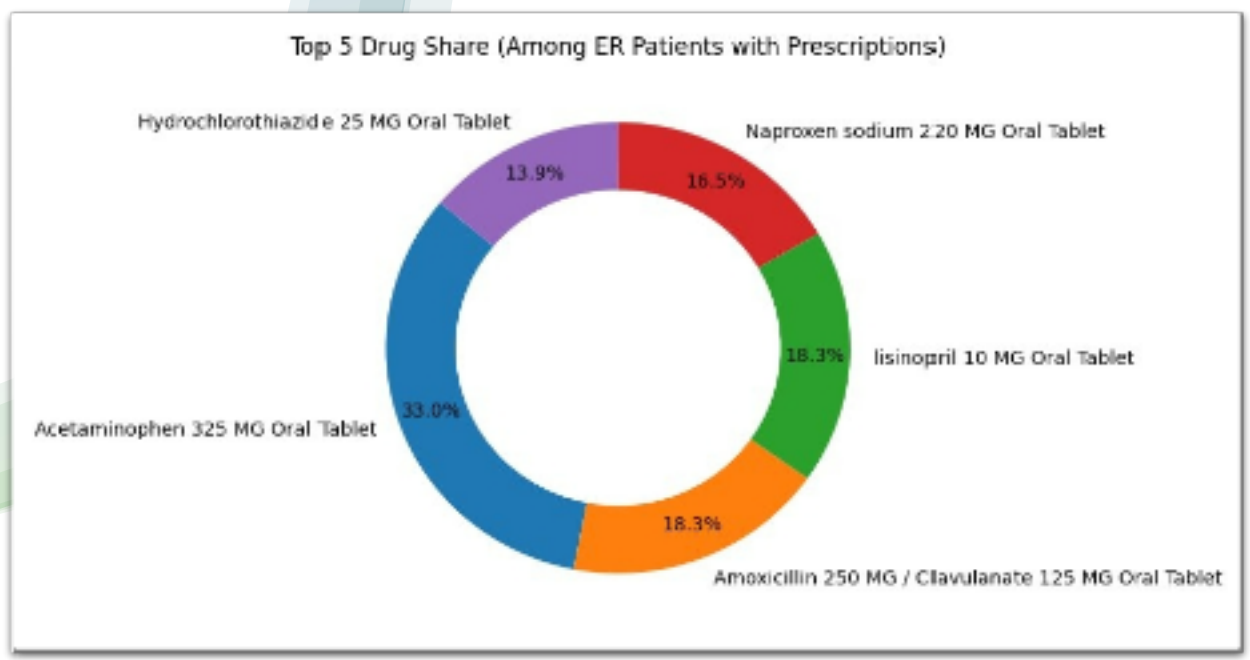

```
cursor.execute('''
    SELECT
        pa.sex,
        CASE
            WHEN TIMESTAMPDIFF(YEAR, pa.DOB, CURDATE()) <= 18 THEN '0-18'
            WHEN TIMESTAMPDIFF(YEAR, pa.DOB, CURDATE()) <= 35 THEN '19-35'
            WHEN TIMESTAMPDIFF(YEAR, pa.DOB, CURDATE()) <= 50 THEN '36-50'
            ELSE '51+'
        END AS Age_Group,
        COUNT(*) AS Visit_Count
    FROM
        Form_Encounter fe
    JOIN
        Patient pa ON fe.pid = pa.pid
    GROUP BY
        pa.sex, Age_Group
    ORDER BY
        Visit_Count DESC;
''')

#Fetch into DataFrame
result = cursor.fetchall()
demographics_df = pd.DataFrame(result, columns=['Sex', 'Age_Group', 'Visit_Count'])

#Visualization (Bar Chart)
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12,6))
sns.barplot(
    data=demographics_df,
    x='Age_Group',
    y='Visit_Count',
    hue='Sex'
```

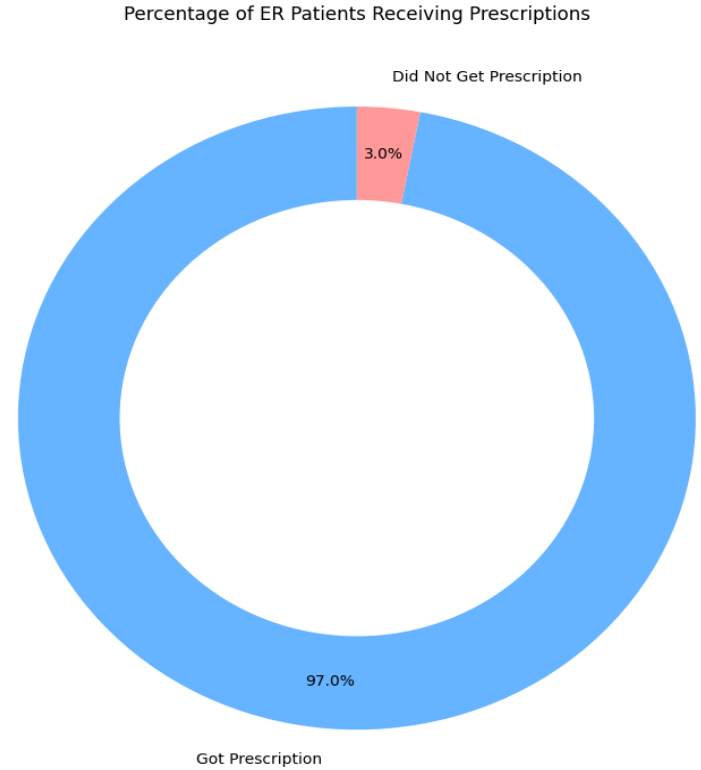
How often do emergency room patients require follow-up prescriptions, and what are the most prescribed medications?

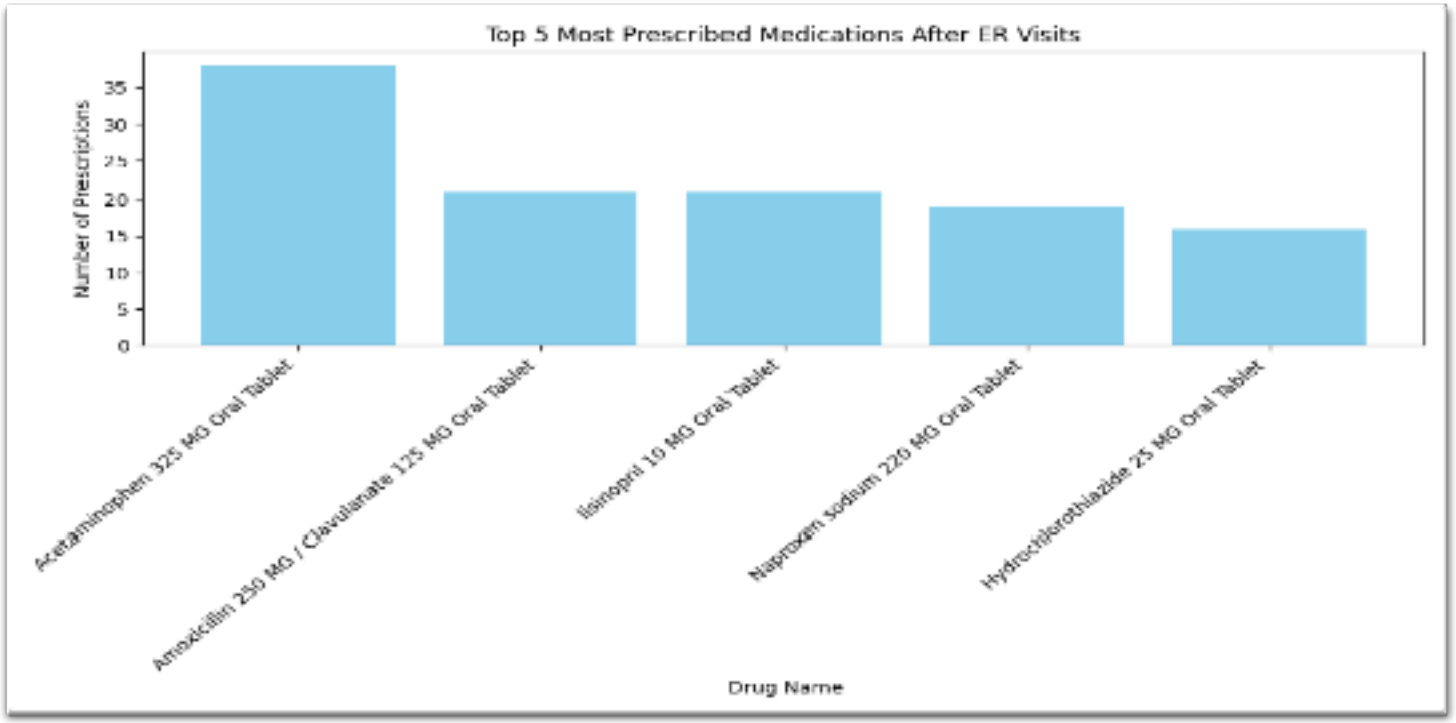
```python
# Q5: How often do emergency room patients require follow-up prescriptions, and what are the most prescribed medications?
#Step 1: Get total distinct patients from Form_Encounter

cursor.execute('''
    SELECT COUNT(DISTINCT pid) FROM Form_Encounter
''')
total_patients = cursor.fetchone()[0]

#Step 2: Get distinct patients who received at least one prescription
cursor.execute('''
    SELECT COUNT(DISTINCT fe.pid)
    FROM Prescription p
    JOIN Form_Encounter fe ON p.encounter_id = fe.encounter_id
''')
patients_with_prescription = cursor.fetchone()[0]

#Step 3: Calculate percentages
patients_without_prescription = total_patients - patients_with_prescription
percentages = [
    (patients_with_prescription / total_patients) * 100,
    (patients_without_prescription / total_patients) * 100
]

# Display
print("✅ ER Follow-up Prescription Statistics:")
print(f"Total ER Patients: {total_patients}")
print(f"Patients with Prescriptions: {patients_with_prescription}")
print(f"Patients without Prescriptions: {patients_without_prescription}")
print(f"Percentage of Patients Getting Prescriptions: {percentages[0]:.2f}%")
print(f"Percentage of Patients Not Getting Prescriptions: {percentages[1]:.2f}%")
```

```
✅ ER Follow-up Prescription Statistics:
Total ER Patients: 100
Patients with Prescriptions: 97
Patients without Prescriptions: 3
Percentage of Patients Getting Prescriptions: 97.00%
Percentage of Patients Not Getting Prescriptions: 3.00%
```

- **97%** of emergency room patients received a follow-up prescription after their visit.
- **3%** of emergency room patients did not require a prescription.
- The **top 5 most prescribed medications** were:
- **Acetaminophen 325 MG Oral Tablet** (38 prescriptions)
- **Amoxicillin 250 MG / Clavulanate 125 MG Oral Tablet** (21 prescriptions)
- **Lisinopril 10 MG Oral Tablet** (21 prescriptions)
- **Naproxen Sodium 220 MG Oral Tablet** (19 prescriptions)
- **Hydrochlorothiazide 25 MG Oral Tablet** (16 prescriptions)







✅ Top 5 Most Prescribed Medications (Among ER Patients with Prescriptions):

	Drug	Prescription_Count
0	Acetaminophen 325 MG Oral Tablet	38
1	Amoxicillin 250 MG / Clavulanate 125 MG Oral T...	21
2	lisinopril 10 MG Oral Tablet	21
3	Naproxen sodium 220 MG Oral Tablet	19
4	Hydrochlorothiazide 25 MG Oral Tablet	16

Findings

Common ER Visit Reasons

- General exams, well-child visits, symptoms, check-ups, and prenatal care.

Vital Signs & Hospitalization

- No link found between abnormal vitals and hospitalization.
- "Admission" cases showed normal temperature, pulse, and respiration.

Most Performed Procedures

- Orders and lab tests were the most common.
- Consistent across all age groups.

Demographic Trends

- Analysis based on sex and age.
- Females visited ER more than males in all age groups.
- Highest visits in 51+ age group, especially females.

.

ER Prescriptions

- 97% received prescriptions; only 3% did not.
- **Top 5 Medications:**
- Acetaminophen, Amoxicillin-Clavulanate, Lisinopril, Naproxen, Hydrochlorothiazide.

Conclusions

In our study, we transformed raw OpenEMR data into a structured MySQL database using Python, ensuring data integrity through normalization and proper key constraints. Our analysis of emergency room visits revealed that general exams and well-child visits were the most common reasons for visits, with orders and lab tests being the most frequently performed procedures across all age groups. We observed that hospitalized patients had higher instances of abnormal pulse and respiration, suggesting a possible association between abnormal vital signs and hospitalization. Additionally, 97% of patients received follow-up prescriptions, most commonly acetaminophen. These insights highlight opportunities to improve emergency care through data-driven decisions.