Health Informatics Standards and Terminologies Final Project

GROUP 8
Subina Ghimire
Likhitha Kantipudi
Sreshta Reddy Nomula
Sahrash Fatima
Sai Datta Prashanth Vadala

Link to Project GitHub Repo

https://github.iu.edu/snomula/final_project

INTRODUCTION

FHIR (Fast Healthcare Interoperability Resources) APIs simplify healthcare data exchange by using web standards like REST and HTTP. They treat healthcare data, such as patient records or medications, as separate resources, allowing for targeted and efficient data sharing.

These APIs enhance system interoperability, integrate seamlessly with EHRs to improve clinical workflows, and enable patients to access their health information through apps. They also support large-scale data transfers, benefiting research and public health initiatives.

Although REST does not inherently address security, FHIR includes authentication and authorization measures to protect sensitive information and ensure regulatory compliance.

Overall, FHIR APIs promote secure, real-time data sharing and help build a connected, patient-focused healthcare system.

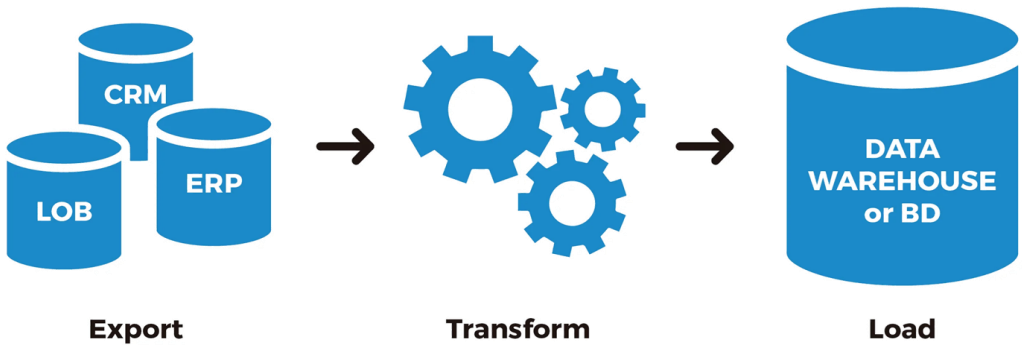
Objective of the Project:

Developing an ETL Pipeline: The primary goal was to design and implement an Extract, Transform, Load (ETL) pipeline to facilitate seamless interaction with healthcare data using FHIR (Fast Healthcare Interoperability Resources) standards.

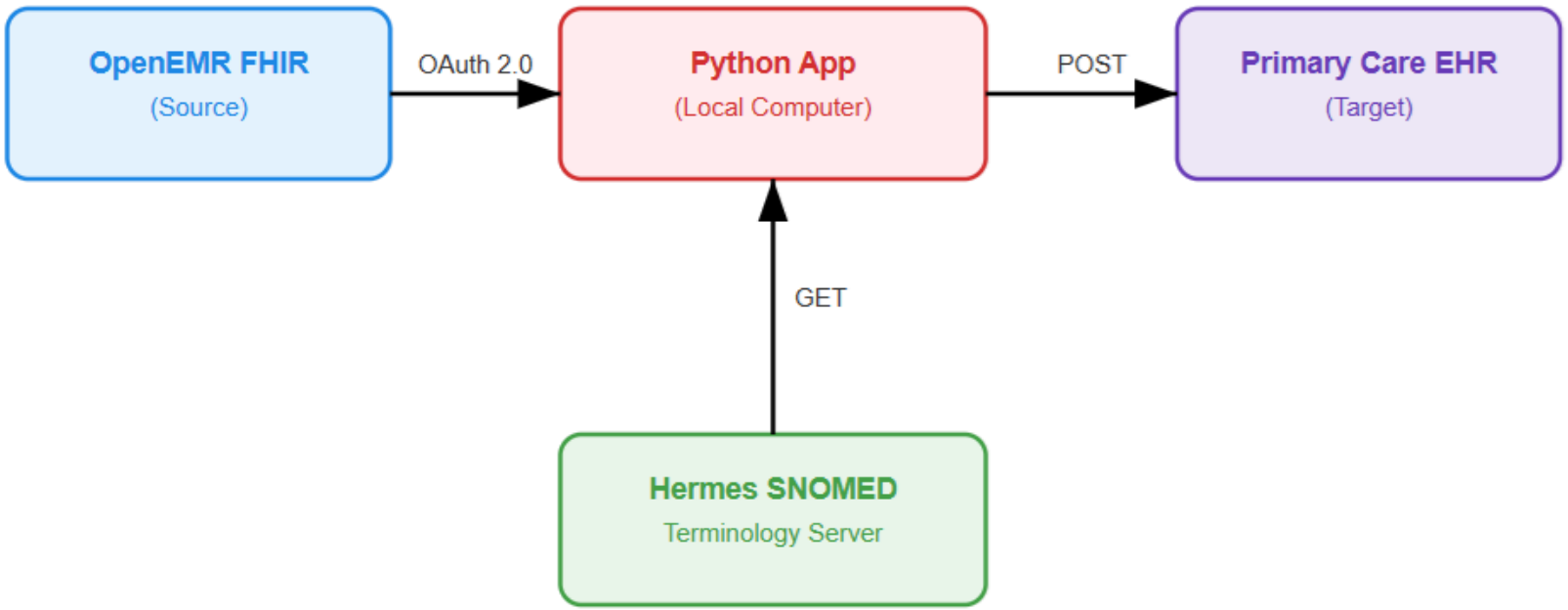
Data Interoperability: Enable efficient data exchange between OpenEMR and Primary Care EHR systems

FHIR API Integration: Use FHIR APIs to extract patient data, process and transform it into a structured format, and load it into target EHR systems.

Enhancing Healthcare Data Insights: Provide visualizations and analyses of patient demographics, conditions, and other healthcare indicators to aid in decision-making.



ETL Pipeline Overview



Extraction

- Retrieved patient data, conditions, and demographics from the OpenEMR FHIR API.
- Accessed hierarchical SNOMED codes (parent/child terms) using the Hermes SNOMED API.
- Used OAuth 2.0 for secure API interactions, with robust error handling.

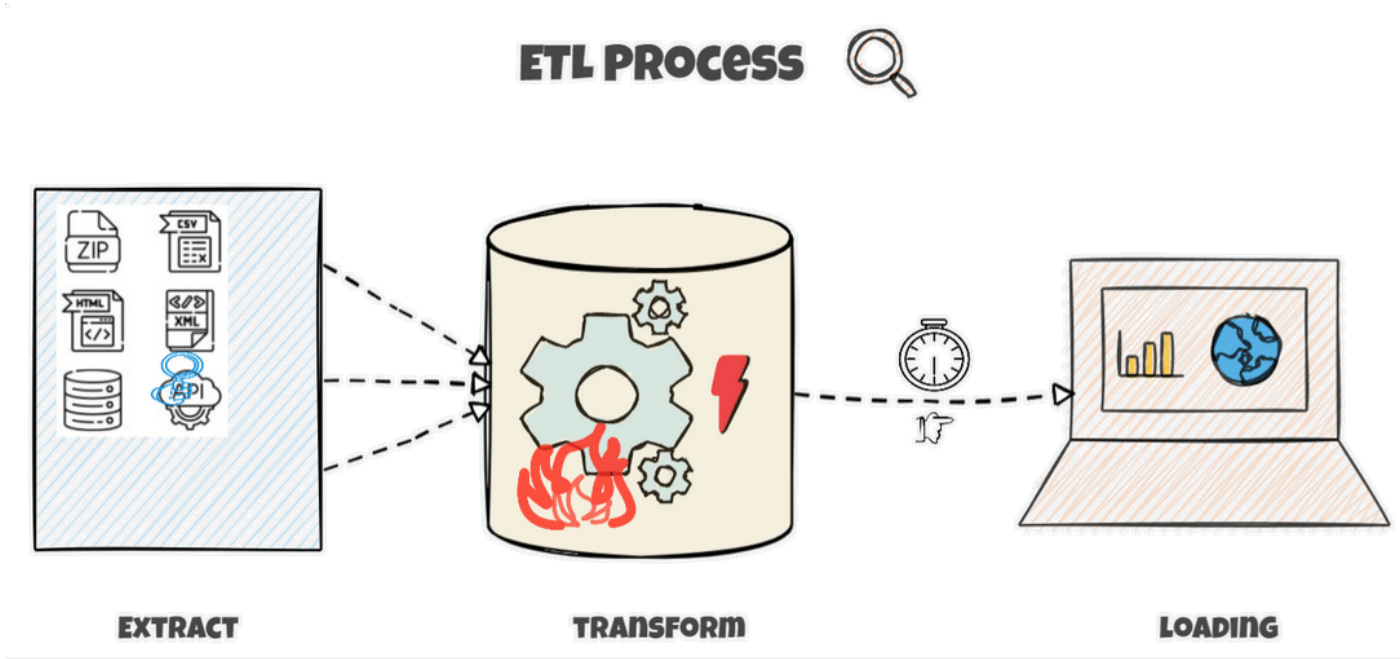
Transformation

- Raw data was cleaned and transformed into JS ON that complies with FHIR.
- Improved the level of de tail of clinical data by in cluding hierarchical SN OMED concepts.
- Validated data to ensure compatibility with the Primary Care EHR system.

Loading

- Posted patient, condition, observation, and procedure data to the Primary Care EHR.
- Ensured all four resources (Patient, Condition, Observation, Procedure) were correctly linked to the same patient.

ETL Pipeline Demonstration



Task 1- Parent Term

Connecting To FHIR API:

- The Python script uses the requests library to connect to the FHIR API for data retrieval. Authentication is handled using OAuth 2.0 tokens stored in a local JSON file, which are dynamically added to request headers.

Error Handling:

- Missing or expired access tokens are identified, and meaningful error messages are printed.
  - Missing or inconsistent fields, such as null addresses or incomplete condition codes, were handled using fallback mechanisms.

Task Execution:

- The script is executed from the command line using Python – task_1.py

Retrieving Patient and Condition Data:

- Using fetch_patient_details(), The patient "Jayson Denesik" was extracted using the /Patient endpoint. Key details such as patient ID, name, and address were retrieved and logged.
- The condition details were extracted using the get_one_condition function. The condition associated with the patient, Aortic valve regurgitation (disorder), was fetched from the /Condition endpoint. The retrieved details include:
  - Condition Code: 60234000
  - Display Term: "Aortic valve regurgitation (disorder)"
  - Terminology Source: SNOMED-CT

Transforming Data:

- SNOMED-CT terminology was utilized to structure the condition data into a hierarchy:
  - Parent Term: "Heart valve regurgitation"
  - Code: 40445007
- Used the generate_patient_json_from_fhir function to transform the patient data into a simplified FHIR-compliant JSON structure.
- Added randomized identifiers using the generate_random_identifier function.
- Combined address components into a single formatted string using the format_combined_address function.

Data Loading:

- Posted the transformed condition JSON to the Primary Care EHR server using the post_condition_to_primary_care function.
- Validated the response to ensure the condition was successfully recorded.

Outcome:

- Successfully integrated Jayson Denesik into the Primary Care EHR along with their condition (Heart valve regurgitation). This demonstrated seamless data interoperability using FHIR and SNOMED-CT standards.

Task 2- Child Term

Task 2 – Overview and Implementation

Connecting To FHIR API:

- Similar to task 1, The Python script utilizes the requests library to connect to the FHIR API, authenticating requests with an OAuth 2.0 token. The token is dynamically retrieved from a secure local file and incorporated into the headers for all API interactions.

Error Handling:

- Missing or expired access tokens are identified, and meaningful error messages are printed.
  - Missing or inconsistent fields, such as null addresses or incomplete condition codes, were handled using fallback mechanisms.

Task Execution:

- The script is executed from the command line using Python – task_2.py

Data Extraction:

- Extracted a condition associated with the patient using the get_one_condition function.
- The retrieved condition was:
  - Condition: Aortic valve regurgitation (disorder)
  - Code: 60234000
  - System: SNOMED-CT

Transformation

- Queried the Hermes SNOMED API to identify the direct child SNOMED term for the condition.
- Retrieved child term details were:
- Child Term: Aortic cusp regurgitation
- Code: 25097600
- Generated a JSON representation of the child condition using the generate_condition_json function.

Data Loading:

- Posted the transformed condition JSON to the Primary Care EHR server using the post_condition_to_primary_care function.
- Validated the response to ensure the condition was successfully recorded.

Outcome

- Successfully posted the transformed condition, Aortic cusp regurgitation, to the Primary Care EHR server, demonstrating effective use of the FHIR and SNOMED-CT standards for healthcare data interoperability.

Task 3- Observation

Task 3 – Overview and Implementation

Receiving Patient ID:

- Retrieved patient information using the patient ID stored in primary_care_patient_resource_id.txt.
- Used the read_patient_id_from_file() function to ensure consistent access to the correct patient ID for creating observations.

Generating JSON

- Created the generate_observation_json() function to generate FHIR-compliant JSON for the Observation resource. The observation included critical details such as:
  - **Body Site**: Mapped to SNOMED CT code 368208006 with the display term "Left arm."
  - **Systolic Blood Pressure**: Mapped to LOINC code 8480-6 with a value of 140 mmHg and marked as "High."
  - **Diastolic Blood Pressure**: Mapped to LOINC code 8462-4 with a value of 92 mmHg and marked as "Above High Normal."

Generating JSON

- Created the generate_observation_json() function to generate FHIR-compliant JSON for the Observation resource. The observation included critical details such as:
  - **Body Site**: Mapped to SNOMED CT code 368208006 with the display term "Left arm."
  - **Systolic Blood Pressure**: Mapped to LOINC code 8480-6 with a value of 140 mmHg and marked as "High."
  - **Diastolic Blood Pressure**: Mapped to LOINC code 8462-4 with a value of 92 mmHg and marked as "Above High Normal."

Posting Observation to Primary Care

- Utilized the post_observation_to_primary_care() function to send the generated observation JSON to the OBSERVATION_URL.

Generating JSON

- Created the generate_observation_json() function to generate FHIR-compliant JSON for the Observation resource. The observation included critical details such as:
  - **Body Site**: Mapped to SNOMED CT code 368208006 with the display term "Left arm."
  - **Systolic Blood Pressure**: Mapped to LOINC code 8480-6 with a value of 140 mmHg and marked as "High."
  - **Diastolic Blood Pressure**: Mapped to LOINC code 8462-4 with a value of 92 mmHg and marked as "Above High Normal."

Posting Observation to Primary Care

- Utilized the post_observation_to_primary_care() function to send the generated observation JSON to the OBSERVATION_URL.

Task 4 - Procedure

Task 4 – Overview and Implementation

Receiving Patient Data:

- Similar to Task 3, we Retrieved patient information using the patient ID stored in primary_care_patient_resource_id.txt.

Posting Observation to Primary Care

Created the generate_procedure_json() function to generate FHIR-compliant JSON for the Procedure resource. The procedure included critical details such as:

- Code and Display: Mapped to SNOMED CT code 1155885007 with the display term "Aortic Valve Repair (Procedure)."

Recorder and Performer: Referenced "Dr. Adam Careful" (Practitioner ID 4) as the procedure recorder and performer.

Follow-up: Included follow-up details, such as "ROS 7 days - 2024-04-10."

Notes: Added contextual information, "Routine Aortic Valve Repair. Valve was repaired successfully."

Generating JSON

- Utilized the post_observation_to_primary_care() function to send the generated observation JSON to the OBSERVATION_URL.

Insights

- Using FHIR APIs for data interoperability, as demonstrated in Tasks 1 and 2, minimized manual errors and ensured seamless healthcare data exchange.
- Automating resource creation (e.g., observations and procedures) in Tasks 3 and 4 highlighted the robustness of structured healthcare data.
- Implementing fallback mechanisms ensured pipeline integrity despite incomplete data, as seen in Task 1 and 2.

Challenges

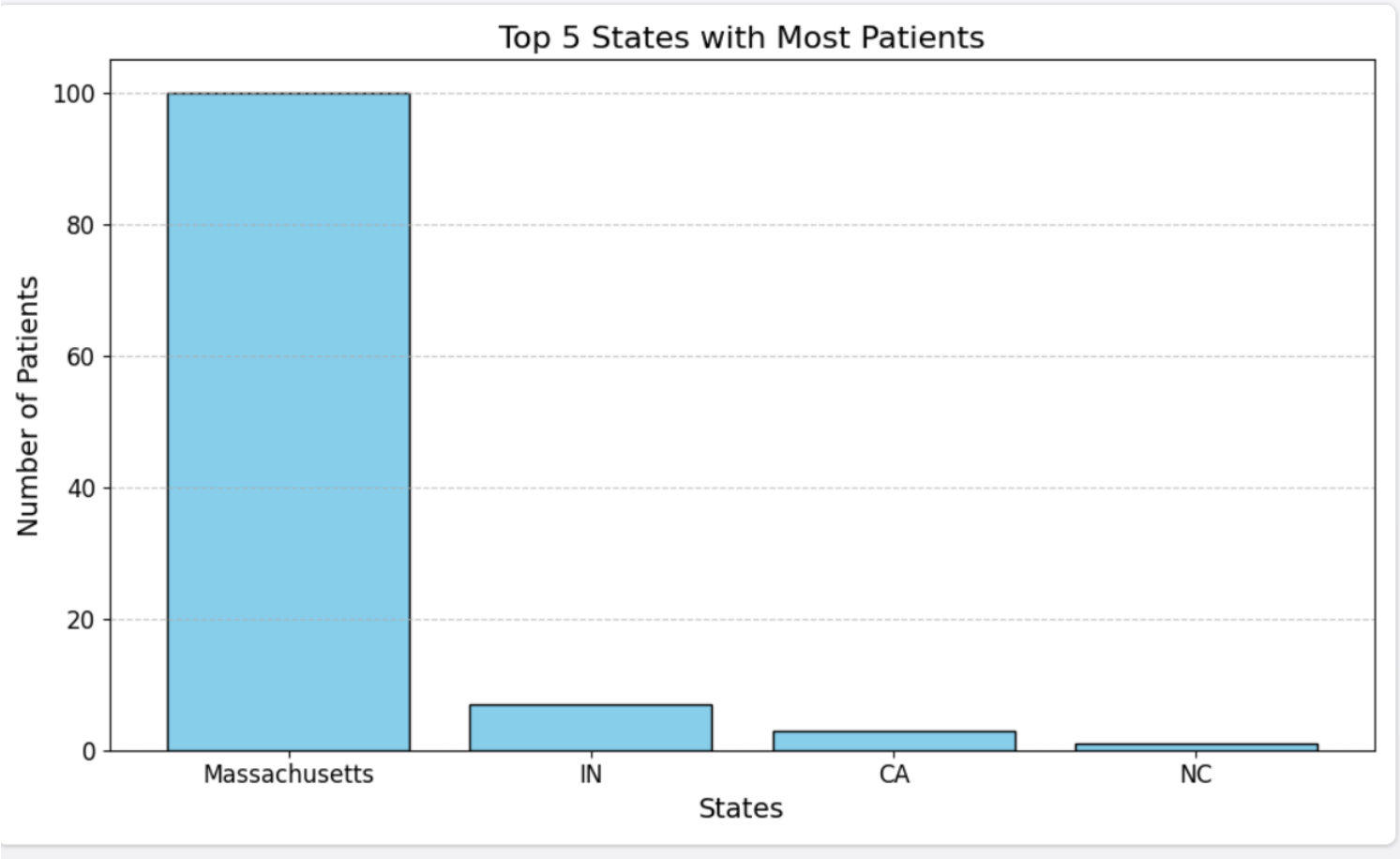
- API Complexities: Authentication hurdles required securely managing access tokens.
- For example, retrieving patient details in Task 1 demanded precise token handling due to strict expiration policies.
- Handling varying data structures between OpenEMR and Primary Care EHR APIs required advanced mapping techniques and custom JSON formats for resources.
- Data Inconsistencies: Missing or incomplete patient data required implementing fallback mechanisms. For instance, Task 1 and Task 2 handled missing body site data while creating observations.

Lessons Learned

- One of the key lessons learned was the importance of understanding FHIR standards.
- Designing effective ETL (Extract, Transform, Load) pipelines required a deep understanding of FHIR resource structures, such as Patient, Condition, Observation, and Procedure.
- This knowledge was crucial in ensuring that the data could be properly mapped and utilized.
- Modular and reusable code streamlined the process for different tasks, while robust error logging helped identify issues during API interaction.

Visualization of OpenEMR Data

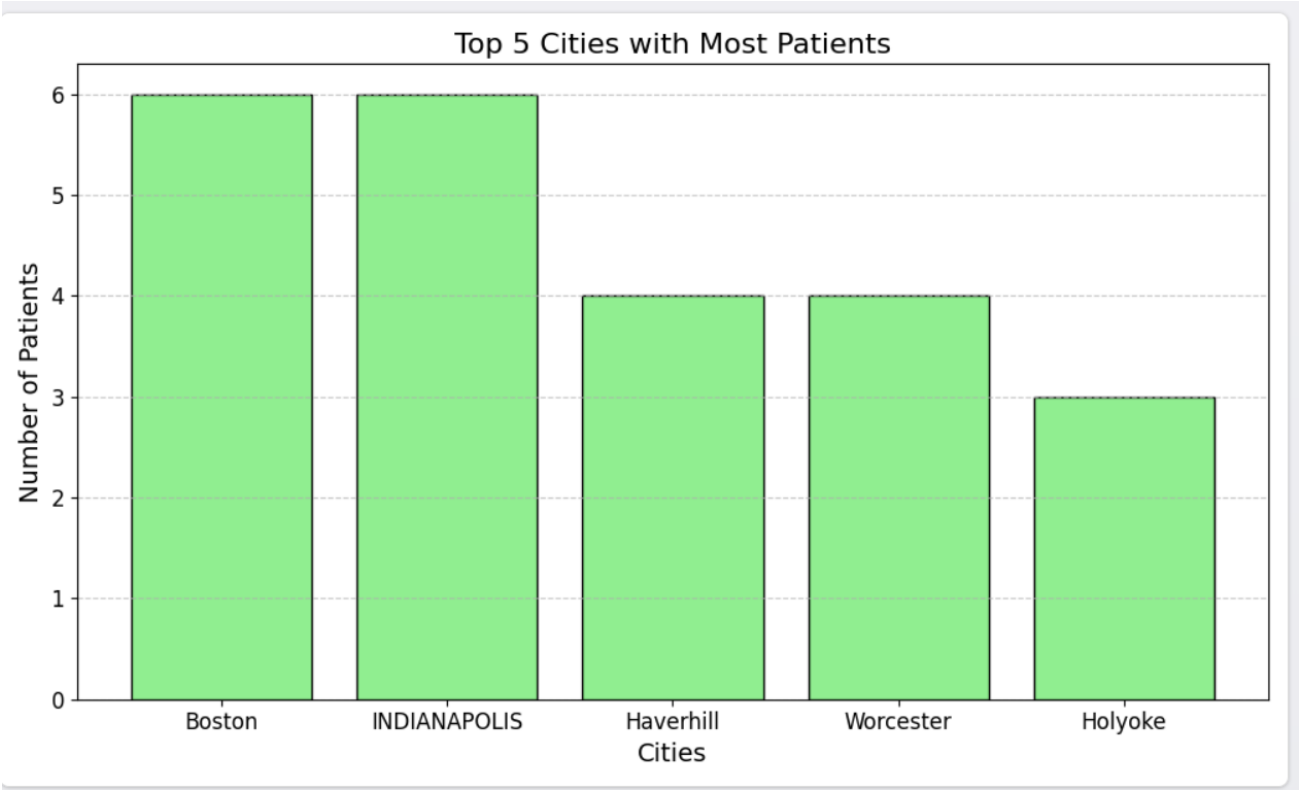
To gain insights from the OpenEMR data, we visualized patient distribution across states. The top 5 states with the highest patient counts were identified and presented in the chart below:



This visualization highlights that Massachusetts has the highest concentration of patients, followed by other states with significantly lower counts.

Visualization of OpenEMR Data

Also, we Visualized the top 5 states with the highest patient counts using a bar chart. Visualization provided actionable insights into patient concentration, useful for healthcare planning.



Boston and Indianapolis have the highest patient counts, followed by Haverhill, Worcester, and Holyoke.

Conclusion

- The ETL (Extract, Transform, Load) pipeline is an invaluable tool for our healthcare organization as it ensures seamless and efficient handling of data across various systems.
- By extracting data from diverse sources like electronic health records (EHRs), lab systems, and patient portals, the ETL pipeline gathers crucial information into one place.
- The transformation step standardizes and cleans this data, ensuring accuracy, consistency, and compliance with regulatory standards like HIPAA.
- Finally, the loading process integrates this prepared data into centralized repositories or analytics platforms, making it readily accessible for reporting and decision-making.
- This pipeline enhances operational efficiency by automating data workflows, reducing manual errors, and saving time.

Thank You!