Create & Access SQLite Database using Python

Description

This notebook demonstrates various operations using SQLite and Python. It covers creating a database, creating tables, inserting data, querying data, updating data, and retrieving data into a pandas DataFrame.

Task 1: Create Database using SQLite

python
Copy code
```
import sqlite3

# Connecting to sqlite
conn = sqlite3.connect('INSTRUCTOR.db')
```

Explanation: This code imports the sqlite3 library and establishes a connection to a SQLite database named INSTRUCTOR.db.

Task 2: Create a Table in the Database

python
Copy code
```
# cursor object
cursor_obj = conn.cursor()
```

Explanation: This creates a cursor object, which allows us to execute SQL commands.

python
Copy code
```
# Drop the table if already exists
cursor_obj.execute("DROP TABLE IF EXISTS INSTRUCTOR")
```

Explanation: Drops the INSTRUCTOR table if it already exists, ensuring a clean slate.

python
Copy code
```
# Creating table
table = """ create table IF NOT EXISTS INSTRUCTOR(
    ID INTEGER PRIMARY KEY NOT NULL,
    FNAME VARCHAR(20),
    LNAME VARCHAR(20),
    CITY VARCHAR(20),
    CCODE CHAR(2)
);"""
```

```
cursor_obj.execute(table)
print("Table is Ready")
```

Explanation: Defines and creates the INSTRUCTOR table with specified columns.

Task 3: Insert Data into the Table

python
Copy code

```
# Insert a single row
cursor_obj.execute('''insert into INSTRUCTOR values (1, 'Rav', 'Ahuja', 'TORONTO', 'CA')''')
```

Explanation: Inserts one row of data into the INSTRUCTOR table.

python
Copy code

```
# Insert multiple rows
cursor_obj.execute('''insert into INSTRUCTOR values
   (2, 'Raul', 'Chong', 'Markham', 'CA'),
   (3, 'Hima', 'Vasudevan', 'Chicago', 'US')''')
```

Explanation: Attempts to insert multiple rows with a single query (Note: This syntax is incorrect for SQLite, you should insert rows individually).

Task 4: Query Data in the Table

python
Copy code

```
# Retrieve all data
statement = '''SELECT * FROM INSTRUCTOR'''
cursor_obj.execute(statement)
output_all = cursor_obj.fetchall()
for row_all in output_all:
   print(row_all)
```

Explanation: Selects and prints all rows from the INSTRUCTOR table.

python
Copy code

```
# Retrieve a few rows
output_many = cursor_obj.fetchmany(2)
for row_many in output_many:
   print(row_many)
```

Explanation: Fetches and prints two rows from the INSTRUCTOR table.

python
Copy code
```
# Retrieve specific column
statement = '''SELECT FNAME FROM INSTRUCTOR'''
cursor_obj.execute(statement)
output_column = cursor_obj.fetchall()
for fetch in output_column:
    print(fetch)
```

Explanation: Selects and prints only the FNAME column from the INSTRUCTOR table.

Task 5: Update Data in the Table

python
Copy code
```
# Update Rav's CITY to MOOSETOWN
query_update='''update INSTRUCTOR set CITY='MOOSETOWN' where FNAME="Rav"'''
cursor_obj.execute(query_update)
```

Explanation: Updates the CITY of 'Rav' to 'MOOSETOWN'.

python
Copy code
```
# Verify the update
statement = '''SELECT * FROM INSTRUCTOR'''
cursor_obj.execute(statement)
output1 = cursor_obj.fetchmany(2)
for row in output1:
    print(row)
```

Explanation: Selects and prints two rows to verify the update.

Task 6: Retrieve Data into Pandas

python
Copy code
```
import pandas as pd

# Retrieve the query results into a pandas dataframe
df = pd.read_sql_query("select * from instructor;", conn)
print(df)
```

Explanation: Uses pandas.read_sql_query to load the INSTRUCTOR table into a DataFrame and prints it.

python

Copy code
```
# Print just the LNAME for the first row
df.LNAME[0]
```

Explanation: Prints the LNAME of the first row in the DataFrame.

Task 7: Close the Connection

python
Copy code
```
# Close the connection
conn.close()
```

Explanation: Closes the connection to the SQLite database.

Summary

In this notebook, you have created a SQLite database and table, inserted data, queried data, updated data, and retrieved data into a pandas DataFrame. This demonstrates how to perform basic database operations using SQLite and Python.

Accessing Databases with SQL Magic

Description

This notebook demonstrates how to use SQL magic with the ipython-sql extension in a JupyterLab environment to perform simplified database operations. It covers connecting to an SQLite database, creating tables, inserting data, running queries, and converting query results to Pandas DataFrames for further analysis.

Task 1: Loading the SQL Extension

python
Copy code
```
%load_ext sql
```

Description: Loads the ipython-sql extension to enable SQL magic commands in the notebook.

Task 2: Connecting to SQLite Database

python
Copy code
```
%sql sqlite:///SQLiteMagic.db
```

Description: Connects to an SQLite database named SQLiteMagic.db. If the database does not exist, it will be created.

Task 3: Creating a Table and Inserting Data

sql
Copy code

```
%%sql

CREATE TABLE INTERNATIONAL_STUDENT_TEST_SCORES (
   country VARCHAR(50),
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   test_score INT
);

INSERT INTO INTERNATIONAL_STUDENT_TEST_SCORES (country, first_name, last_name,
test_score)
VALUES
('United States', 'Marshall', 'Bernadot', 54),
-- Additional rows omitted for brevity
('Australia', 'Eduard', 'Leipelt', 53);
```

Description: Creates a table named INTERNATIONAL_STUDENT_TEST_SCORES with columns for country, first name, last name, and test score. Inserts multiple rows of test data into the table.

Task 4: Using Python Variables in SQL Statements

python
Copy code

```
country = "Canada"
%sql select * from INTERNATIONAL_STUDENT_TEST_SCORES where country = :country
```

Description: Demonstrates how to use a Python variable (country) in an SQL query to retrieve records of students from Canada.

Task 5: Assigning Query Results to a Python Variable

python
Copy code

```
test_score_distribution = %sql SELECT test_score as "Test_Score", count(*) as "Frequency"
from INTERNATIONAL_STUDENT_TEST_SCORES GROUP BY test_score;
test_score_distribution
```

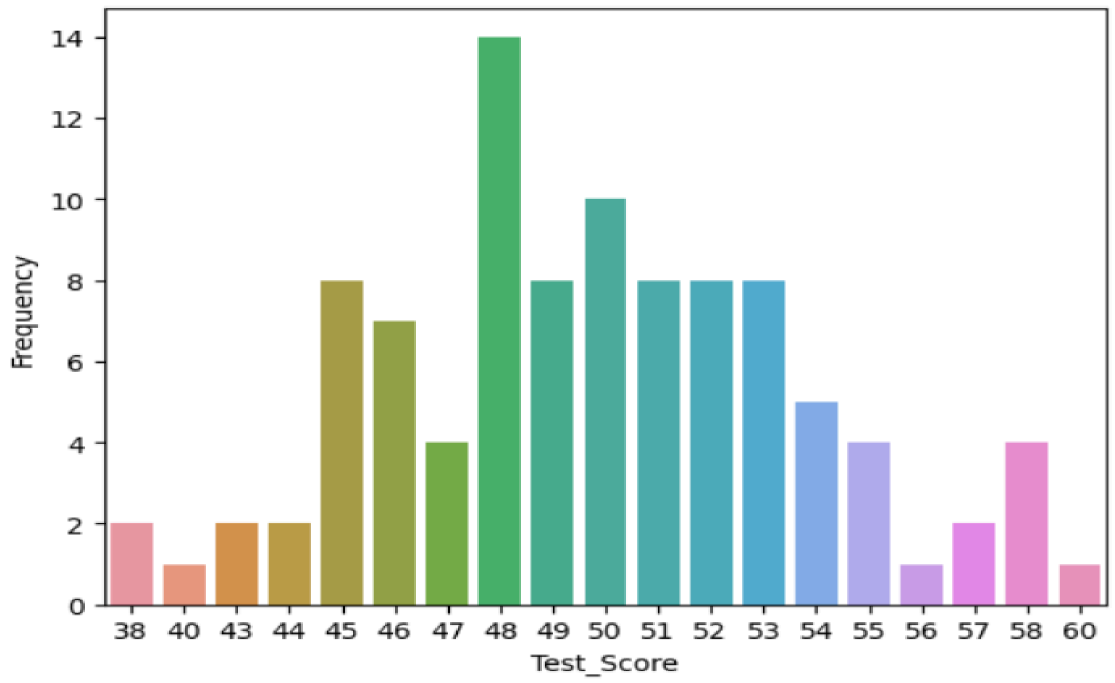
Description: Runs an SQL query to get the distribution of test scores and assigns the result to a Python variable test_score_distribution.

Task 6: Converting Query Results to a DataFrame

python
Copy code
dataframe = test_score_distribution.DataFrame()

%matplotlib inline
import seaborn

plot = seaborn.barplot(x='Test_Score', y='Frequency', data=dataframe)



Description: Converts the SQL query result to a Pandas DataFrame and uses Seaborn to plot a bar chart of the test score distribution.

Task 7: Experiment with the Data

sql
Copy code
%%sql

SELECT country, first_name, last_name, test_score FROM
INTERNATIONAL_STUDENT_TEST_SCORES;

Description: Runs an SQL query to select and display all records from the INTERNATIONAL_STUDENT_TEST_SCORES table for further experimentation.

Summary:
This repository contains Jupyter notebooks demonstrating various database operations using SQL magic with the ipython-sql extension. Each notebook covers different aspects of database interaction, including creating tables, inserting data, running queries, and visualizing query results. The notebooks are designed for educational purposes and to facilitate simplified database access within a JupyterLab environment.

Analyzing a Real World Data-Set with SQL and Python

Overview

The dataset of socioeconomic indicators in Chicago provides insights into various factors that affect public health and overall well-being across different community areas within the city. The data, collected by the city of Chicago, covers a selection of six key socioeconomic indicators along with a composite "hardship index" for the years 2008-2012. The purpose of this dataset is to facilitate the analysis of socioeconomic conditions and their impacts on community areas, helping in policy formulation and resource allocation.

In this notebook, we analyze a dataset of selected socioeconomic indicators in Chicago, focusing on various factors that may influence public health. By the end of this analysis, we aim to draw meaningful insights and conclusions from the data. The main objectives are to understand the dataset, store it in an SQLite database, and solve example problems using SQL. Additionally, we visualize certain aspects of the data to better understand correlations between variables.

Objective

The primary objective of this analysis is to:

- Understand the dataset and its variables.
- Store the dataset in an SQLite database for efficient querying.
- Perform SQL queries to extract meaningful insights.
- Visualize the data to uncover relationships and trends.
- Detailed Steps and Code Explanations

1. Connect to the Database

python
Copy code

```
%load_ext sql
import csv, sqlite3

con = sqlite3.connect("socioeconomic.db")
cur = con.cursor()
!pip install -q pandas==1.1.5
%sql sqlite:///socioeconomic.db
```

We start by loading the SQL extension and connecting to the SQLite database named socioeconomic.db.

2. Store the Dataset in a Table

python
Copy code

```
import pandas
df = pandas.read_csv('https://data.cityofchicago.org/resource/jcxq-k9xf.csv')
df.to_sql("chicago_socioeconomic_data", con, if_exists='replace', index=False, method="multi")
```

We read the dataset from a CSV file available online and store it in a table named chicago_socioeconomic_data within our SQLite database.

3. Verify Table Creation

sql
Copy code

```
%sql SELECT * FROM chicago_socioeconomic_data limit 5;
```

We run a basic SQL query to verify that the table has been created successfully and contains the expected data.

Problem Solving with SQL Queries

Problem 1: Count Rows in the Dataset

sql
Copy code

```
%sql SELECT COUNT(*) FROM chicago_socioeconomic_data;
```

We determine that there are 78 rows in the dataset.

Problem 2: Community Areas with Hardship Index Greater than 50.0

sql
Copy code

```
%sql SELECT COUNT(*) FROM chicago_socioeconomic_data WHERE hardship_index > 50.0;
```

We find that 38 community areas have a hardship index greater than 50.0.

Problem 3: Maximum Hardship Index

sql
Copy code

```
%sql SELECT MAX(hardship_index) FROM chicago_socioeconomic_data;
```

The maximum value of the hardship index in this dataset is 98.0.

Problem 4: Community Area with the Highest Hardship Index

sql
Copy code

```
%sql SELECT community_area_name FROM chicago_socioeconomic_data where
hardship_index=98.0;
```

Riverdale is identified as the community area with the highest hardship index.

Problem 5: Community Areas with Per-Capita Incomes Greater than $60,000

sql
Copy code

```
%sql SELECT community_area_name FROM chicago_socioeconomic_data WHERE
per_capita_income_ > 60000;
```

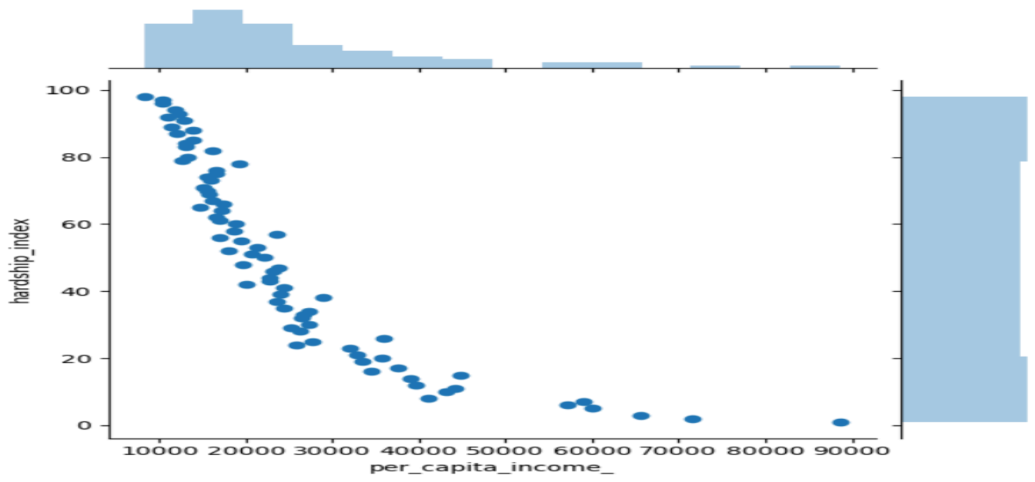
The community areas with per-capita incomes greater than $60,000 are Lake View, Lincoln Park, Near North Side, and Loop.

Problem 6: Scatter Plot of Per Capita Income vs. Hardship Index

python
Copy code

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

income_vs_hardship = %sql SELECT per_capita_income_, hardship_index FROM
chicago_socioeconomic_data;
plot = sns.jointplot(x='per_capita_income_', y='hardship_index',
data=income_vs_hardship.DataFrame())
```



We create a scatter plot to visualize the relationship between per capita income and hardship index. The plot shows an inverse correlation, indicating that areas with higher per capita income tend to have lower hardship indices.

Conclusion

In this notebook, we successfully:

- Stored a real-world dataset in an SQLite database.
- Performed various SQL queries to extract and analyze the data.
- Visualized the relationship between per capita income and hardship index.

Insights

A significant number of community areas in Chicago experience high levels of hardship. Riverdale has the highest hardship index, indicating severe socio economic challenges. Wealthier areas like Lake View, Lincoln Park, Near North Side, and Loop have higher per capita incomes and lower hardship indices, highlighting economic disparities across different regions of Chicago.

Summary

This notebook demonstrates how to leverage SQL and Python for exploratory data analysis of socioeconomic data, providing valuable insights into the socioeconomic landscape of Chicago. Through SQL queries and data visualization, we can better understand the distribution and impact of socioeconomic factors on public health.

Working with a real world data-set using SQL and Python

Objective

The goal of this analysis is to understand the performance of Chicago Public Schools during the 2011-2012 academic year. This involves examining various metrics related to school performance, safety, attendance, and other key indicators to derive insights that can aid in policy formulation and resource allocation.

Dataset Description

The dataset used for this analysis includes a wide range of metrics collected by the city of Chicago to create school report cards. These metrics encompass aspects such as safety scores, average student attendance, college enrollment, and other performance indicators for the 2011-2012 school year.

Methodology

The analysis is performed using SQL within a Jupyter notebook. The dataset is stored in an SQLite database, and various queries are executed to retrieve and analyze the data.

Step 1: Database Connection

First, we establish a connection to the SQLite database and load the necessary extensions:

python
Copy code

```
import csv, sqlite3
import pandas as pd

con = sqlite3.connect("RealWorldData.db")
cur = con.cursor()

!pip install -q pandas==1.1.5
%load_ext sql
```

```
%sql sqlite:///RealWorldData.db
```

Step 2: Loading Data into the Database

We load the CSV files into the SQLite database using Pandas:

python
Copy code

```
df =
pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloper
SkillsNetwork-DB0201EN-SkillsNetwork/labs/FinalModule_Coursera_V5/data/ChicagoCensusD
ata.csv")
df.to_sql("CENSUS_DATA", con, if_exists='replace', index=False, method="multi")

df =
pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloper
SkillsNetwork-DB0201EN-SkillsNetwork/labs/FinalModule_Coursera_V5/data/ChicagoCrimeDat
a.csv")
df.to_sql("CHICAGO_CRIME_DATA", con, if_exists='replace', index=False, method="multi")

df =
pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloper
SkillsNetwork-DB0201EN-SkillsNetwork/labs/FinalModule_Coursera_V5/data/ChicagoPublicSch
ools.csv")
df.to_sql("CHICAGO_PUBLIC_SCHOOLS_DATA", con, if_exists='replace', index=False,
method="multi")
```

Step 3: Retrieving Metadata

We verify the table creation and retrieve the column metadata:

sql
Copy code

```
%sql SELECT name FROM sqlite_master WHERE type='table'

%sql SELECT count(name) FROM
PRAGMA_TABLE_INFO('CHICAGO_PUBLIC_SCHOOLS_DATA');

%sql SELECT name, type, length(type) FROM
PRAGMA_TABLE_INFO('CHICAGO_PUBLIC_SCHOOLS_DATA');
```

Step 4: Data Analysis

Problem 1: Number of Elementary Schools

sql
Copy code

```
%sql SELECT count(*) FROM CHICAGO_PUBLIC_SCHOOLS_DATA WHERE "Elementary,
Middle, or High School"='ES';
```

Insight: There are 462 elementary schools in the dataset.

Problem 2: Highest Safety Score

sql
Copy code

```
%sql SELECT MAX(Safety_Score) AS MAX_SAFETY_SCORE FROM
CHICAGO_PUBLIC_SCHOOLS_DATA;
```

Insight: The highest safety score is 99.

Problem 3: Schools with Highest Safety Score

sql
Copy code

```
%sql SELECT Name_of_School, Safety_Score FROM CHICAGO_PUBLIC_SCHOOLS_DATA
WHERE Safety_Score = 99;
```

Insight: Several schools, including Abraham Lincoln Elementary School and Alexander Graham Bell Elementary School, have a safety score of 99.

Problem 4: Top 10 Schools by Average Student Attendance

sql
Copy code

```
%sql SELECT Name_of_School, Average_Student_Attendance FROM
CHICAGO_PUBLIC_SCHOOLS_DATA ORDER BY Average_Student_Attendance DESC
NULLS LAST LIMIT 10;
```

Insight: John Charles Haines Elementary School has the highest average student attendance at 98.40%.

Problem 5: Schools with Lowest Average Student Attendance

sql
Copy code

```
%sql SELECT Name_of_School, Average_Student_Attendance FROM
CHICAGO_PUBLIC_SCHOOLS_DATA ORDER BY Average_Student_Attendance LIMIT 5;
```

Insight: Velma F Thomas Early Childhood Center has no recorded attendance, while Richard T Crane Technical Preparatory High School has the lowest at 57.90%.

Problem 6: Remove '%' Sign from Average Student Attendance

sql
Copy code

```
%sql SELECT Name_of_School, REPLACE(Average_Student_Attendance, '%', '') FROM
CHICAGO_PUBLIC_SCHOOLS_DATA ORDER BY Average_Student_Attendance LIMIT 5;
```

Problem 7: Schools with Attendance Lower than 70%

sql
Copy code

```
%sql SELECT Name_of_School, Average_Student_Attendance FROM
CHICAGO_PUBLIC_SCHOOLS_DATA WHERE
CAST(REPLACE(Average_Student_Attendance, '%', '') AS DOUBLE) < 70 ORDER BY
Average_Student_Attendance;
```

Insight: The schools with Average Student Attendance lower than 70% are Richard T Crane Technical Preparatory High School, Barbara Vick Early Childhood & Family Center, Dyett High School, Wendell Phillips Academy High School, Orr Academy High School, Manley Career Academy High School, Chicago Vocational Career Academy High School, and Roberto Clemente Community Academy High School.

Problem 8: Total College Enrollment per Community Area

sql
Copy code

```
%sql SELECT Community_Area_Name, SUM(College_Enrollment) AS TOTAL_ENROLLMENT
FROM CHICAGO_PUBLIC_SCHOOLS_DATA GROUP BY Community_Area_Name;
```

Problem 9: Community Areas with Least Total College Enrollment

sql

Copy code

```
%sql SELECT Community_Area_Name, SUM(College_Enrollment) AS TOTAL_ENROLLMENT
FROM CHICAGO_PUBLIC_SCHOOLS_DATA GROUP BY Community_Area_Name ORDER
BY TOTAL_ENROLLMENT ASC LIMIT 5;
```

Insight: Oakland has the lowest total college enrollment at 140.

Problem 10: Schools with Lowest Safety Score

sql
Copy code

```
%sql SELECT Name_of_School, Safety_Score FROM CHICAGO_PUBLIC_SCHOOLS_DATA
WHERE Safety_Score != 'None' ORDER BY Safety_Score LIMIT 5;
```

Insight: The five schools with the lowest safety scores are Edmond Burke Elementary School, Luke O'Toole Elementary School, George W Tilton Elementary School, Foster Park Elementary School, and Emil G Hirsch Metropolitan High School.

Problem 11: Hardship Index for Community Area of a Specific School

sql
Copy code

```
%%sql
SELECT hardship_index FROM CENSUS_DATA CD, CHICAGO_PUBLIC_SCHOOLS_DATA
CPS WHERE CD.community_area_number = CPS.community_area_number AND
college_enrollment = 4368;
```

Insight: The hardship index for the community area of the school with a college enrollment of 4368 is 6.0.

Problem 12: Hardship Index for Community Area with Highest College Enrollment

sql
Copy code

```
%sql SELECT community_area_number, community_area_name, hardship_index FROM
CENSUS_DATA WHERE community_area_number IN (SELECT community_area_number
FROM CHICAGO_PUBLIC_SCHOOLS_DATA ORDER BY college_enrollment DESC LIMIT 1);
```

Insight: North Center has the highest college enrollment and a hardship index of 6.

Summary

In this notebook, we analyzed the Chicago Public Schools performance dataset for 2011-2012. By executing SQL queries, we gained insights into school safety scores, average student attendance, college enrollment, and community hardship indices. This analysis provides valuable information for educational policymakers and stakeholders to understand and improve school performance in Chicago.

Final Assignment-Database Querying using SQLite

In this notebook, I worked with three datasets from the city of Chicago's Data Portal to understand various socioeconomic, educational, and crime data. The goal was to understand and load these datasets into an SQLite database and execute SQL queries to answer specific questions. Below, I provide a brief description of each dataset, the code used to manipulate and analyze them, and the insights gained from the queries.

Datasets Used

Socioeconomic Indicators in Chicago: This dataset contains six socioeconomic indicators and a hardship index for each Chicago community area for the years 2008-2012. It helps in understanding the public health and economic conditions of different community areas.

Chicago Public Schools: This dataset provides school-level performance data used to create CPS School Report Cards for the 2011-2012 school year. It includes various metrics like safety scores, college enrollment, and other performance indicators.

Chicago Crime Data: This dataset includes reported incidents of crime in Chicago from 2001 to present (excluding the most recent seven days). It provides detailed information about each crime, including its type, location, and whether an arrest was made.

Code Summary

Loading Libraries and Data:

Pandas and SQLite3 libraries were used.
Data was read from provided URLs and loaded into dataframes.
Dataframes were then loaded into an SQLite database (FinalDB.db).

Connecting to Database:

Established a connection to the SQLite database using SQL magic module.

SQL Queries and Insights:

Problem 1: Found the total number of crimes recorded. (533 crimes)

Problem 2: Listed community areas with per capita income less than $11,000. (West Garfield Park, South Lawndale, Fuller Park, Riverdale)

Problem 3: Retrieved case numbers for crimes involving minors. (HL266884, HK238408)

Problem 4: Identified kidnapping crimes involving children. (One case on 050XX W VAN BUREN ST)

Problem 5: Listed unique types of crimes recorded at schools. (Battery, Criminal Damage, Narcotics, Assault, Criminal Trespass, Public Peace Violation)

Problem 6: Computed the average safety score for each type of school. (Elementary: 49.52, High: 49.62, Middle: 48.0)

Problem 7: Listed 5 community areas with the highest percentage of households below the poverty line. (Riverdale, Fuller Park, Englewood, North Lawndale, East Garfield Park)

Problem 8: Identified the most crime-prone community area. (Area number 25)

Problem 9: Found the community area with the highest hardship index. (Riverdale with a hardship index of 98)

Problem 10: Determined the community area with the most crimes. (Austin)
Insights

From the queries and answers, I gained several insights about the datasets:

Crime Data: Certain community areas like Austin and the area with number 25 are more prone to crimes. Schools are hotspots for various types of crimes.

Socioeconomic Data: Areas like Riverdale and Fuller Park are economically distressed, as indicated by low per capita incomes and high percentages of households below the poverty line. Riverdale also has the highest hardship index, reflecting significant socioeconomic challenges.

Education Data: The safety scores across elementary, middle, and high schools are relatively consistent, with middle schools having slightly lower scores on average.

This analysis provided a deeper understanding of the interconnectedness of socioeconomic conditions, educational environments, and crime patterns in Chicago.