ETL Report

Albert Prouty, Chris Kusha, Clara McGrath, Jassleen Bhullar

Introduction

Our group was curious about the rates of crime in Chicago, and the Chicago Data Portal includes a comprehensive dataset about reported crime in Chicago that updates daily. In the Portal, we also found datasets related to the state of education in Chicago Public Schools, and we were curious about any correlations between crime and education in Chicago. To answer these questions, we explored and transformed datasets from the Chicago Data Portal and the Chicago Public School website.

Data Sources

Crimes 2013-2023 City of Chicago Data Portal

https://data.cityofchicago.org/Public-Safety/Crimes-2013/a95h-gwzm
https://data.cityofchicago.org/Public-Safety/Crimes-2014/qnmj-8ku6
https://data.cityofchicago.org/Public-Safety/Crimes-2015/vwwp-7yr9
https://data.cityofchicago.org/Public-Safety/Crimes-2016/kf95-mnd6
https://data.cityofchicago.org/Public-Safety/Crimes-2017/d62x-nvdr
https://data.cityofchicago.org/Public-Safety/Crimes-2018/3i3m-jwuy
https://data.cityofchicago.org/Public-Safety/Crimes-2019/w98m-zvie
https://data.cityofchicago.org/Public-Safety/Crimes-2020/qzdf-xmn8
https://data.cityofchicago.org/Public-Safety/Crimes-2021/dwme-t96c
https://data.cityofchicago.org/Public-Safety/Crimes-2022/9hwr-2zxp
https://data.cityofchicago.org/Public-Safety/Crimes-2023/xguy-4ndq

Chicago Public Schools Profile Information - School Year 2022-2023

https://data.cityofchicago.org/Education/Chicago-Public-Schools-School-Profile-Information-/9a5f-2r4p

Chicago Public Schools - Progress Report Cards (2011-12, 2022-23).

https://data.cityofchicago.org/Education/Chicago-Public-Schools-Progress-Report-Cards-2011-/9xs2-f89t

https://data.cityofchicago.org/Education/Chicago-Public-Schools-School-Progress-Reports-SY2/d7as-muwj

Chicago Public Schools - Metrics

https://www.cps.edu/globalassets/cps-pages/about-cps/district-data/metrics/misconduct_report_police_and_expulsion_thru_eoy_2022_school_level.xlsx

https://www.cps.edu/globalassets/cps-pages/about-cps/district-data/metrics/misconduct_report_police_and_expulsion_thru_eoy_2022_district_level.xlsx

ETL Processes

Crime Database

Setting up SQL Database

1. Initialized crime table in SQL database using formatting from the Chicago crime API.

   Note: Code used for this and other SQL queries can be found in Code/crime_database.sql

2. Populated crime database with historical data:
   a. Download csv for 2023 Crimes from data portal
   b. Import into Azure Data Studio through flat file importer extension
   c. Check data types and import into a temporary table.
   d. Use INSERT INTO to add the entries into the main crime table
   e. Dropped temporary table
   f. Repeat process for each year 2022-2013

3. Using Query 1 we noticed a couple of columns have variations that should be condensed into single columns:
    a. ‘non-criminal’ and ‘non - criminal’
    b. ‘criminal sexual assault’ and ‘crim sexual assault’

Creating Pipeline for Streaming Data

1. Obtain API credentials from the API on any of the crime datasets through creating an account and requesting an app token. While technically optional, this step will allow for larger/more frequent data streaming without risk of throttling.
2. Create a Producer notebook in Databricks that will be responsible for extracting data from the API, and sending through the pipeline using Kafka messaging.
3. Install the confluent-kafka and sodapy packages.
4. Import packages:
    a. Pandas
    b. Numpy
    c. Socrata from sodapy
    d. Uuid
    e. Confluent_kafka as well as confluent_kafka.admin
    f. Json
    g. Time
5. Set up Configuration information with Data Lake credentials (storage container, SAS token) as well as API credentials obtained in step 1.
6. Initialize client with Socrata
7. Use client.get to make an API call to retrieve the 1000 latest crimes.

8. Put the results into a pandas dataframe using pd.DataFrame.from_records
9. Drop the location column (the information is contained elsewhere and the formatting is incompatible with the spark database)
10. Create a spark dataframe while filling null values (to avoid issues with json conversion later)
11. Initialize Kafka producer and credentials
12. Create topic for messages to be sent to
13. Transform spark dataframe entries to a list of json dictionaries for kafka messages:
    a. spark_df.toJSON() -> creates RDD object
    b. df.RDD.collect() -> creates list
    c. json.loads(row) for row in df_LIST -> transforms list into json dictionaries
14. Send messages to Kafka broker
15. Create a separate Databrick for the Consumer to read the Kafka messages and import the data into the SQL database.
16. Install confluence-kafka if needed, and import packages:
    a. Confluence_kafka
    b. Json
    c. Datetime and timedelta from datetime
    d. Col, min, and max from pyspark.sql.functions
17. Initialize consumer with credentials
18. Subscribe the consumer to the topic created in step 12.
19. Initialize an empty list for messages, using a while loop read in kafka messages and append to the message list while there are still messages to be read.

20. Create a DataFrame from the messages:
    a. Extract the column names from the keys of the first message
    b. Create an empty list for the data
    c. Looping through each message, extract the values and append them to the data list.
    d. Create spark dataframe using the data list and column names
21. Clean the data for import into SQL database:
    a. Drop any duplicates
    b. Drop the x_coordinate and y_coordinate columns
    c. Change data types:
        i. Id -> int
        ii. Beat -> int
        iii. District -> int
        iv. Date -> timestamp
        v. Latitude -> float
        vi. Longitude -> float
22. Read in the current SQL database through the Java Database Connectivity (jdbc) , selecting all ids between the min and max id and placing them into a Dataframe.
23. Use a left anti-join to get only the new entries that are not currently in the database.
24. Append the resulting data into the database using the jdbc write function.

Automating the Process

1. To automate this process we create a pipeline in Azure Data Factory.
2. Place the Producer and Consumer Databricks, and set up a connection link.

3. The Chicago database updates daily, so we set up a trigger to run this pipeline every 12 hours (giving leeway in case of a failed run).

Overall Results

As a result of this process, we have a database with 11 years of historical data with new data being added daily automatically. This data can now be queried for analysis, as well as streamed to dash for visualization with real time updates. Our ETL also involved static datasets:

Safety Scores

1. Download school profile information and progress report cards as CSV files from Chicago Data Portal.
2. Read in the profile information CSV file using Pandas in Jupyter Notebook after importing Pandas. (pd.read_csv('filename.csv'))
3. Check to see if the data was loaded correctly using the head() function in Pandas.
4. Filter to only keep columns that are needed. I used the subsetting method where I created a new dataframe and only included the columns I wanted from the previous column.
5. Split the new dataframe into 3 separate dataframes, depending on the type of schools: elementary, high, and middle schools.
6. Check for missing values in each new dataframe using isnull().sum().
7. Use .isnull().to_numpy().nonzero() to find the integer indices of the missing values in the columns that had the most missing values.
8. Examine the missing value and the row with the missing value.
9. Drop the row if the record is irrelevant without the missing data or keep the row if the record still presents sufficient information. For example, some schools were missing a

rating, but since they still had a safety score, I kept the record. Other schools with missing data were no longer operating, so I removed those records.

10. Concatenate the three data frames together after cleaning missing values using .concat().
11. Read in the progress report CSV file using Pandas in Jupyter Notebook (pd.read_csv('filename.csv')).
12. Filter for necessary columns by subsetting the data frame into a new dataframe.
13. Check for null values. Examine the missing values. Drop, if needed.
14. Merge the profile information and progress report data frames together using the .merge() function.
15. Change data types for columns where data type doesn't make sense using the .astype() function.
16. Remove records that don't have a safety score.
17. Replace the alphabetic scale with a numeric scale.
18. Create a column that has counts of each response using the .groupby() function.
19. Read in the school progress report (2011-12) CSV file to get ward information.
20. Filter out unnecessary columns. Rename columns in the new data frame so it matches the safety data frame. Change data types of records as needed.
21. Merge the wards dataframe with the previous safety data frame. Drop and rename columns as needed.
22. Check for correlations between variables and test theories of correlation using the .corr() function.
23. Import in matplotlib and seaborn. Use the two libraries to experiment with visuals for the final dashboard.

24. Export the full safety scores data frame as a CSV file using the .to_csv() function.

School Profiles

1. Download Chicago Public School Profiles CSV (School Year 2022-2023) from Chicago Data Portal.
2. Read CSV into Jupyter Notebook using Pandas.
3. Confirm data loaded properly, examine using describe(), shape, info() and/or head() function(s).
4. Drop all unnecessary columns from the dataframe to include only what is or might be needed.
5. Create subsets of the data frame to separate preschool, elementary, middle, and high school. Use “primary category” column to filter these, as some of the schools have many grades.
6. Check if the rows that are ‘False’ for the ‘Is_High_School’ column do not have values for the graduation rate and college enrollment rate columns as a sanity check.
7. Check the shape of each subset to confirm that changes have been made and subsets were created.
8. Check for nulls in the main dataframe and in subsets.
9. Create an aggregate column to calculate the percentages of each demographic in the student body (race/ethnicity, low income, etc.).
10. Check r values for graduation rate and college enrollment rates (exploratory, also sanity check).
11. Create exploratory visualizations using seaborn and PowerBI to perform further EDA.
12. Export the main dataframe as a CSV file.

13. In a new Jupyter Notebook, import Pandas, then read in the three CSVs: school profiles (cleaned version), school safety (cleaned and merged with ward information), and 2022 crime reports.
14. Merge school profiles and school safety dataframes on School ID using pd.merge().
15. Drop unnecessary or duplicate columns.
16. Create a new dataframe with only desired columns from the crime dataframe. Add a new column to the crime data frame that counts the number of crimes in each ward using groupby() and size().
17. Merge new crime dataframe with profile/safety data frame on ward column using pd.merge(). Confirm a new dataframe was created, check shape and head().
18. Perform sanity checks looking at schools, wards, and corresponding number of crimes in those wards.
19. Use corr() function in Pandas to look at correlations between columns of interest before creating visualizations.
20. Rename necessary columns for readability in the visualizations.
21. Create visualizations with seaborn and tailor them for readability.
22. Rewrite the code for the visualizations in plotly.express so that it is usable for final dashboard in Dash.

School Level and District Level

1. Downloaded SL and CW reports as Excel files from the Chicago Public Schools Metrics page.
2. Loaded both the SL and CW reports into Excel.
3. Removed the Overview tab from both files.

4. Removed footnotes from SL file as it created rows full of only NA values after saving as CSV.
5. Removed top header and used subheaders as headers to avoid reading issues with loading into DF.
6. After the above steps were taken, saved both files as a CSV with UTF-8 encoding with the following names: dl_2022 and sl_2022.
7. Loaded SL and CW reports into Visual Studio Code.
8. SL data types became objects; had to .astype() relevant columns to int.

Machine Learning

1. dl_2022 data was read in with the following constraint: 'Time Period1' == 'EOY'
2. The following columns were kept from the resulting dataframe: 'Category','School Year','# of Misconducts2','# of Group 1-2 (minor) Misconducts3','# of Group 3-4 (moderate) Misconducts4','# of Group 5-6 (major) Misconducts5', '# of Police Notifications', '% of Misconducts Resulting in Police Notification'
3. Only the following values of 'Category' were kept: 'Grade PK-2', 'Grade 3-5', 'Grade 6-8', 'Grade 9-12'
4. sl_2022 data was read in with the following constraint: 'Time Period' == 'EOY'
5. The following columns were kept from sl_2022: 'School ID', 'School Name','School Network','School Year','# of Misconducts','# of Group 1-2 Misconducts','# of Group 3-4 Misconducts', '# of Group 5-6 Misconducts', '# of Suspensions (includes ISS and OSS)','# of Police Notifications', '% of Misconducts Resulting in a Police Notification'

6. cps_safety.csv was read in and merged with the transformed sl_2022 data: how='inner', left_on ='School_ID', right_on ='School ID'
7. The following columns were dropped from the merged df: 'Unnamed: 0', 'School ID', 'Long_Name'
8. A new df (named x1) was created from the merged df with the following columns: 'Community Area Number', 'Ward', 'Police District', 'Primary_Category', 'Zip', 'Student_Count_Total', 'Student_Count_Low_Income', 'Student_Count_Special_Ed', 'Student_Count_Black', 'Student_Count_Hispanic', 'Student_Count_White', 'Student_Count_Asian', 'Student_Count_Native_American', 'Student_Count_Asian_Pacific_Islander', 'Student_Count_Hawaiian_Pacific_Islander', 'Overall_Rating', 'School_Survey_Safety', '# of Misconducts', '# of Group 1-2 Misconducts', '# of Group 3-4 Misconducts', '# of Group 5-6 Misconducts', '# of Suspensions (includes ISS and OSS)', '# of Police Notifications', '% of Misconducts Resulting in a Police Notification'
9. X1 was exported as a CSV for possible use for the dashboard visualizations.
10. Crime data (cc2) with year == 2022 was pulled in from the SQL database, and the following columns were kept: 'community_area', 'primary_type', 'domestic'
11. 'community_area' was converted to int type.
12. A new dataframe (x2) was created by merging x1 and cc2 using an inner join with left_on = 'Community Area Number' and right_on = 'community_area'
13. x2 was exported as hdf
14. x2 was then loaded into a new Visual Studio Code file to begin working on fitting an ML model.

Conclusion

All of the datasets that were used in the ETL process and transformed for visualizations are available on Github along with this report. This report should guide readers to reproduce Group 5's ETL, if needed.