Consolidating Student Management Databases into a Unified Relational Database

John Chung, Daniela Alejandra Gonzalez, Noah Rae-Grant
GitHub Repository

1 Introduction

Provide a brief introduction to the project, including the background, objectives, and scope.

Our final project addresses a real-world challenge: creating a relational database for an educational institution that currently lacks a DBMS (database management system). The project is grounded in the specific needs and requests of an actual independent high school, referred to as "Academy X" in this report. Academy X aims to better understand trends in student outcomes by leveraging data from its Admissions Office and College Counseling Office.

This project explores the practical challenges of integrating siloed databases and demonstrates the use of SQL queries and visualizations to generate high-value insights for the institution.

2 Literature Review

Summarize the existing research relevant to the project. Discuss the methodologies, findings, and gaps in the literature.

Our project did not rely on existing research but instead utilized skills and methodologies taught directly in DS5110. Additionally, one group member, a subject matter expert with over a decade of experience at Academy X, provided critical insights. As a result, the SQL queries were designed to address the institution's actual needs and priorities.

3 Methodology

Describe the methods and techniques used in the project. Include details about data collection, preprocessing, and analysis.

This final project incorporates many of the skills taught throughout the semester in DS5110:

- Entity Relation Diagrams (ERD)
- Data preprocessing and cleaning techniques
- Building a relational database (RDB) in SQLite
- Designing SQL queries
- Data visualization

The data used for this project is based on existing data at Academy X but was modified for confidentiality purposes. The data fields in our RDB mirror the actual existing data fields at *Academy X* as shown in the ERD below, but the actual values needed to be modeled to simulate realistic student data (see section *3.1 Data Collection* for more).

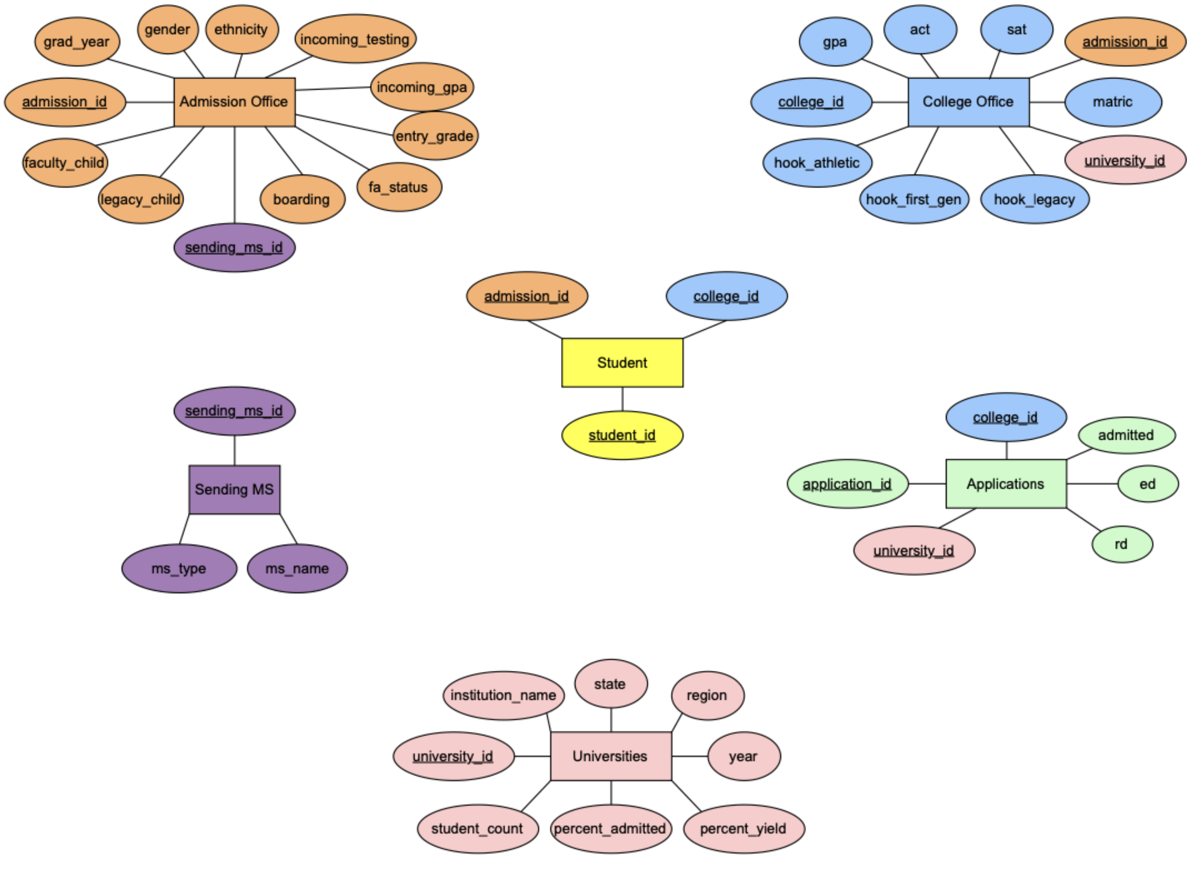


Fig 1. - ERD displaying entities and attributes. `STUDENT` provides the link between incoming (`ADMISSION`) and outgoing (`COLLEGE`) data.

3.1 Data Collection

Explain how the data was collected, including the sources and tools used.

The data for this project was derived from databases associated with Academy X's Admission and College Counseling offices. The attributes depicted in the ERD in Fig. 1 represent a subset of the actual data fields collected from these offices.

To comply with Academy X's data governance and privacy policies, real student data was not utilized. Instead, data was generated based on predetermined distribution levels, leveraging the expertise and insights of a group member with connections to Academy X.

For example, the `APPLICATIONS` table was designed according to the following specifications:

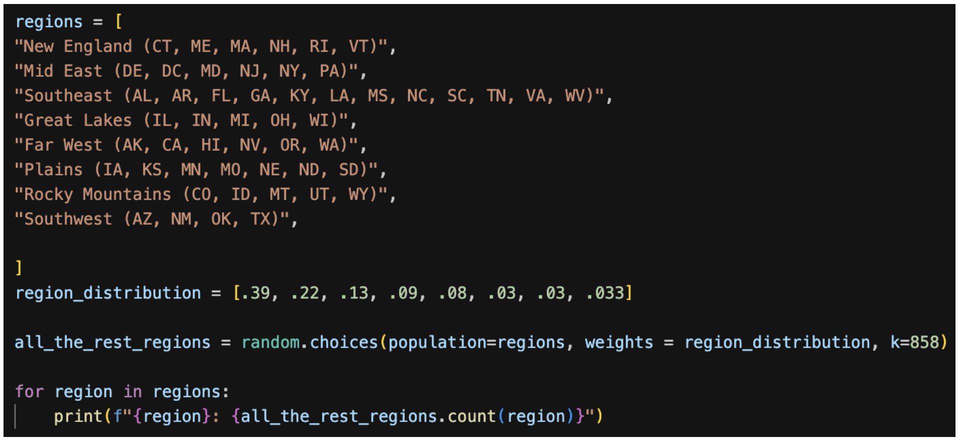
- Around 20% of the student body should be recruited athletes who only apply to one university.
- The remaining students submit applications to between 1 and 15 unique universities.

The number of applications submitted per student and universities applied to were structured to mimic realistic distributions. Code was developed to generate data satisfying these criteria, incorporating random seeding with preferential bias towards the most popular universities, which represent 78% of applications students submit from Academy X.

For example, the expected distribution of applications by `UNIVERSITIES.region` is outlined in the following table:

region	**percent**
New England (CT, ME, MA, NH, RI, VT)	39%
Mid East (DE, DC, MD, NJ, NY, PA)	22%
Southeast (AL, AR, FL, GA, KY, LA, MS, NC, SC, TN, VA, WV)	13%
Great Lakes (IL, IN, MI, OH, WI)	9%
Far West (AK, CA, HI, NV, OR, WA)	8%
Plains (IA, KS, MN, MO, NE, ND, SD)	3%
Rocky Mountains (CO, ID, MT, UT, WY)	3%
Southwest (AZ, NM, OK, TX)	3%

The screenshot below provides an excerpt of the corresponding code that adheres to the specified distribution:



```
regions = [
"New England (CT, ME, MA, NH, RI, VT)",
"Mid East (DE, DC, MD, NJ, NY, PA)",
"Southeast (AL, AR, FL, GA, KY, LA, MS, NC, SC, TN, VA, WV)",
"Great Lakes (IL, IN, MI, OH, WI)",
"Far West (AK, CA, HI, NV, OR, WA)",
"Plains (IA, KS, MN, MO, NE, ND, SD)",
"Rocky Mountains (CO, ID, MT, UT, WY)",
"Southwest (AZ, NM, OK, TX)",

]
region_distribution = [.39, .22, .13, .09, .08, .03, .03, .033]

all_the_rest_regions = random.choices(population=regions, weights = region_distribution, k=858)

for region in regions:
    print(f"{region}: {all_the_rest_regions.count(region)}")
```

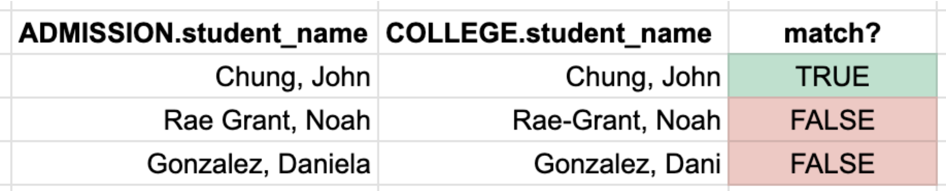
3.2 Data Preprocessing

Describe the steps taken to clean and preprocess the data.

Even though our database is ultimately comprised of generated data, the data preprocessing stage of this project highlighted important best practices in database design.

3.2.1 Real-life Challenges: Using `student_name` as Primary Key

The initial phase of this project involved attempting to merge the real databases from Academy X's Admission and College Offices. A significant issue encountered during this process was that both offices used `student_name` as a primary key in their spreadsheets. Neither office followed consistent criteria for formatting names, resulting in mismatches between `ADMISSION.student_name` and `COLLEGE.student_name`.

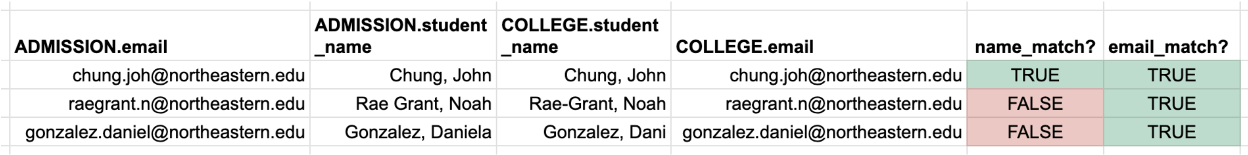


ADMISSION.student_name	COLLEGE.student_name	match?
Chung, John	Chung, John	TRUE
Rae Grant, Noah	Rae-Grant, Noah	FALSE
Gonzalez, Daniela	Gonzalez, Dani	FALSE

In real-life, both offices retained additional data fields that could be used to validate and reconcile unmatched records, such as:

- Email address
- Graduation year
- Demographics (gender, race/ethnicity, financial aid status)

While we ultimately generated our data for privacy reasons, this issue underscores an important best practice in database design: the use of integer-based primary keys to uniquely identify records. To address similar real-life challenges, leveraging secondary fields for data validation (ie, email and graduation year) is a practical workaround to improve data integrity and ensure successful integration across systems.

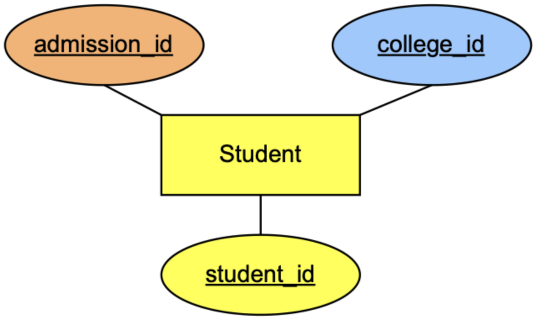


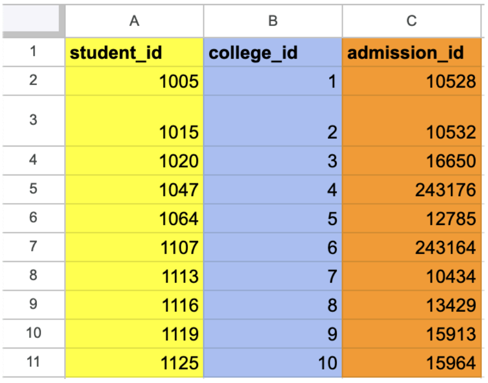
ADMISSION.email	ADMISSION.student_name	COLLEGE.student_name	COLLEGE.email	name_match?	email_match?
chung.joh@northeastern.edu	Chung, John	Chung, John	chung.joh@northeastern.edu	TRUE	TRUE
raegrant.n@northeastern.edu	Rae Grant, Noah	Rae-Grant, Noah	raegrant.n@northeastern.edu	FALSE	TRUE
gonzalez.daniel@northeastern.edu	Gonzalez, Daniela	Gonzalez, Dani	gonzalez.daniel@northeastern.edu	FALSE	TRUE

3.2.2 Creation of STUDENTS as Junction Table

In order to integrate Academy X’s data into a relational database, our group performed the following steps:

1. Added a column of unique integer primary keys into the `ADMISSION` and `COLLEGE` tables.
2. Created `STUDENTS` as a Junction Table that acts as the link between `ADMISSION` and `COLLEGE` tables
3. Validate identities of all students, ensuring that each `ADMISSION.admission_id` corresponds to the correct `COLLEGE.college_id`, removing any entries that do not have a match in both tables.
4. Generate `STUDENTS.student_id` once all student identities have been validated





	A	B	C
1	student_id	college_id	admission_id
2	1005	1	10528
3	1015	2	10532
4	1020	3	16650
5	1047	4	243176
6	1064	5	12785
7	1107	6	243164
8	1113	7	10434
9	1116	8	13429
10	1119	9	15913
11	1125	10	15964

3.2.1 Creation of Relational Database using SQLite

(include sample of code used for preprocessing and converting to RDB)

3.3 Analysis Techniques

Detail the analytical techniques and models used in the project.

4 Results

Present the results of the analysis. Use tables, figures, and charts to support the findings.

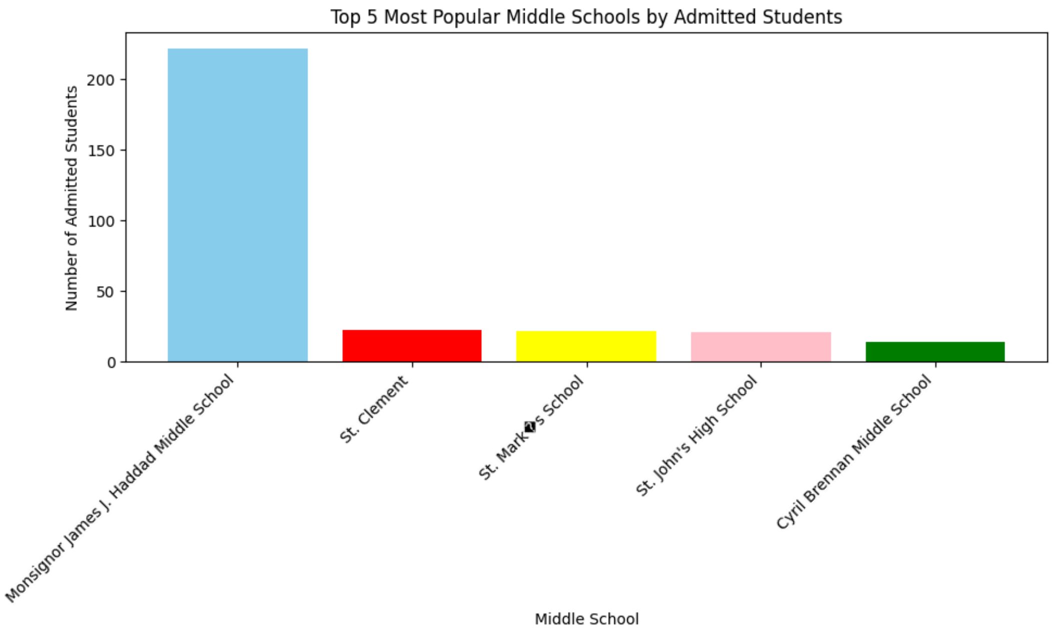
For certain queries that were multi-part, we've included only the output that resulted in a figure. The full results of those queries can be found in our GitHub repository.

Query 1: What are the top 5 most popular Middle Schools and how many admitted students did each send?

Query

```
SELECT Sending_ms.ms_name, COUNT(DISTINCT Admission.admission_id)
AS admitted_students_count
FROM Sending_ms
JOIN Admission ON Sending_ms.sending_ms_id =
Admission.sending_ms_id
JOIN College ON Admission.admission_id = College.admission_id
JOIN Students ON College.college_id = Students.college_id
JOIN Applications ON Students.student_id = Applications.student_id
WHERE Applications.admitted = 1
GROUP BY Sending_ms.ms_name
ORDER BY admitted_students_count DESC
LIMIT 5;
```

Result

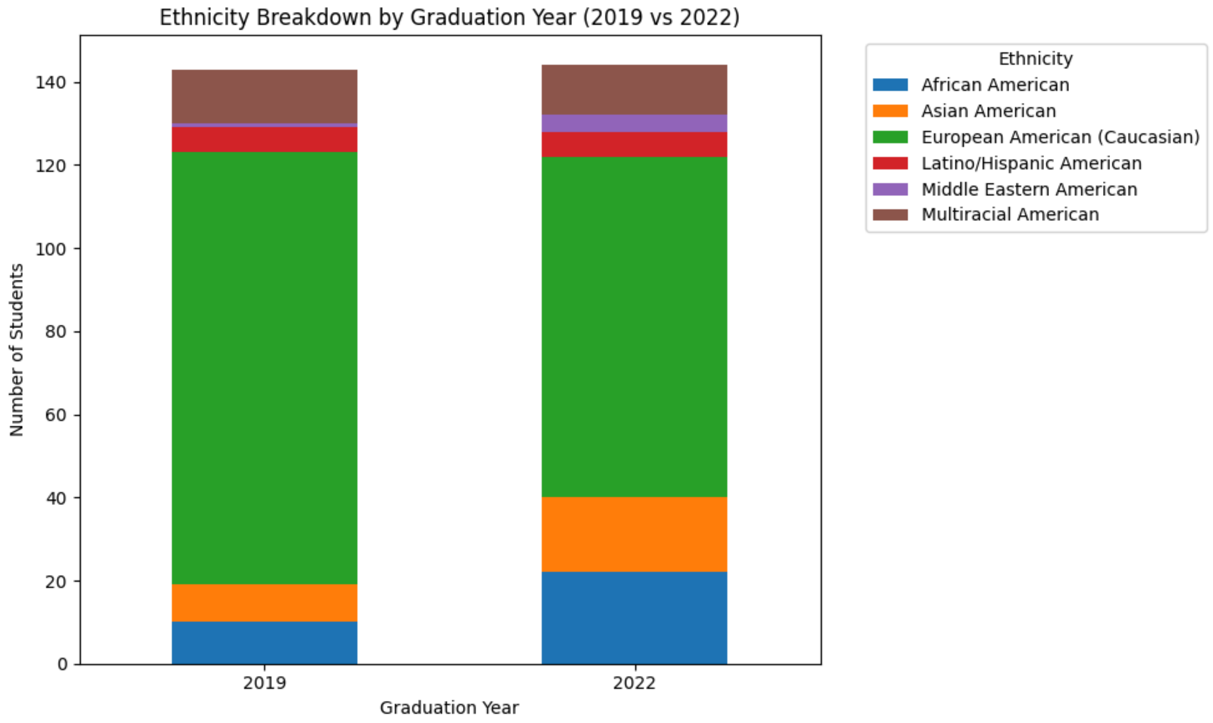


Query 2: What is the ethnicity breakdown of each graduating class? (Part of a multi-part question)

Query

```
SELECT
     Admission.grad_year,
     Admission.ethnicity,
     COUNT(Students.student_id) AS student_count
FROM
     Students
INNER JOIN
     Admission
ON
     Students.admission_id = Admission.admission_id
GROUP BY
     Admission.grad_year, Admission.ethnicity
ORDER BY
     Admission.grad_year, Admission.ethnicity;
```

Result (This query had multiple plots associated with it, but for brevity's sake we're only including the comparison chart between 2019 and 2022.)



Query 3: How does the ethnicity breakdown compare between students who are on 90%+ financial aid vs. students who are full pay? (Part of a multi-part question)

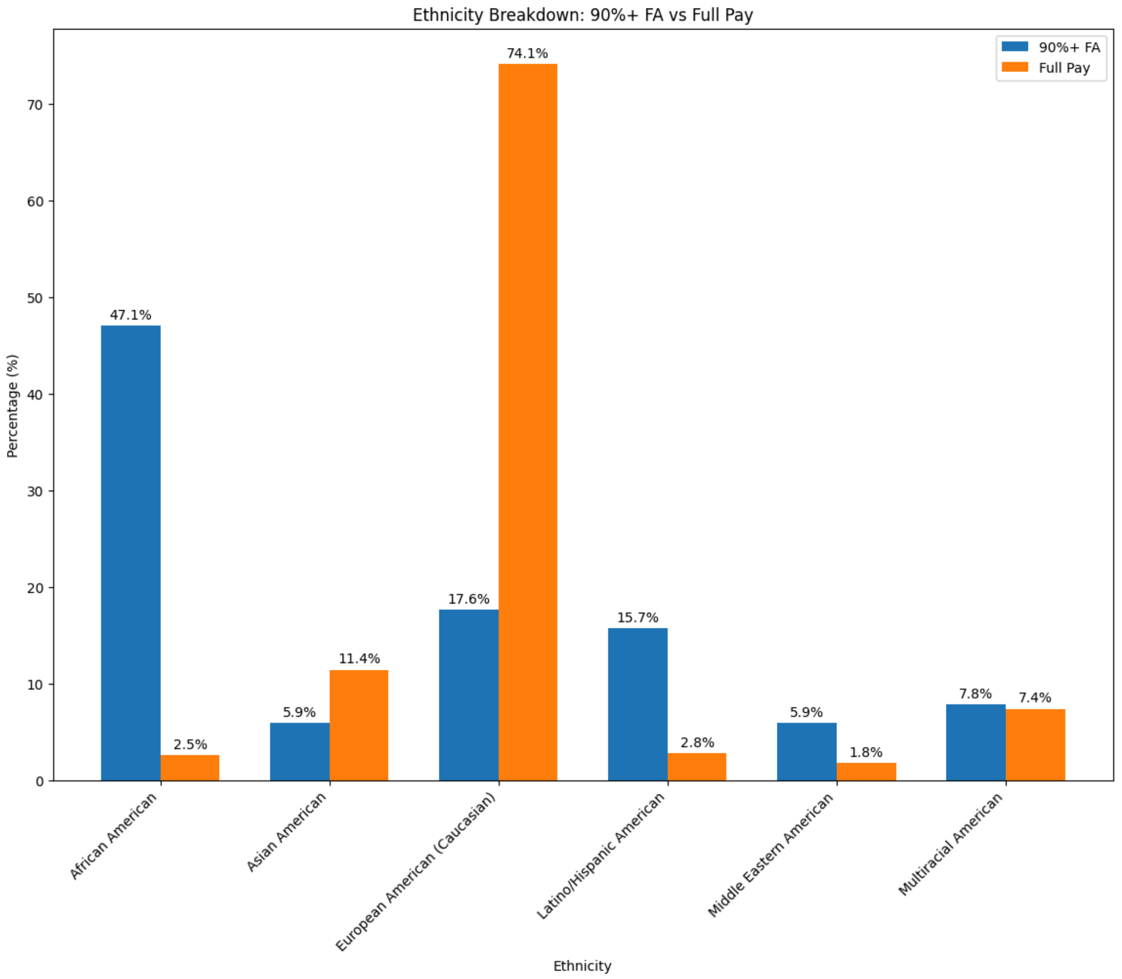
Query 1

```
SELECT Admission.ethnicity, COUNT(*) AS student_count
FROM Students
INNER JOIN Admission ON Students.admission_id = Admission.admission_id
WHERE Admission.FA_status = 2
GROUP BY Admission.ethnicity;
```

Query 2

```
SELECT Admission.ethnicity, COUNT(*) AS student_count
FROM Students
INNER JOIN Admission ON Students.admission_id = Admission.admission_id
WHERE Admission.FA_status = 0
GROUP BY Admission.ethnicity;
```

Result

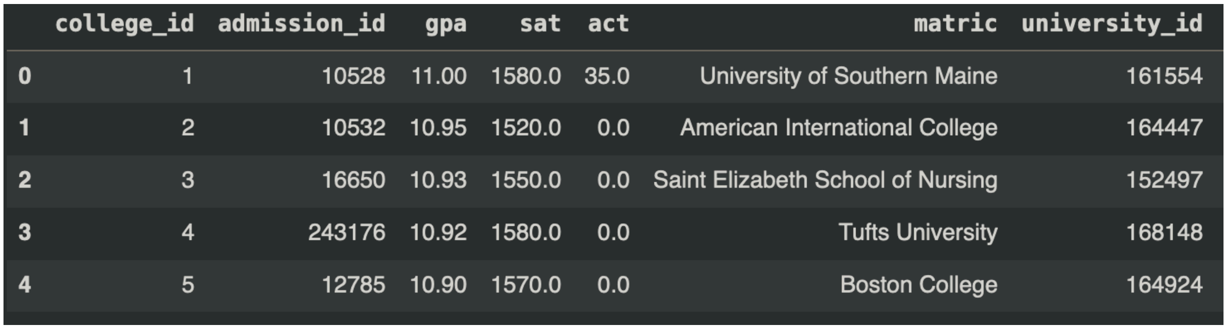


Query 4: Return a list of all students who earned an A- GPA or higher
Note: Academy X uses an 11-point GPA scale; a GPA of 10 or greater corresponds to an A- or higher.

Query

```
SELECT *
FROM College
WHERE gpa >= 10.0;
```

Result (Head of the dataframe; full results in our GitHub repository)



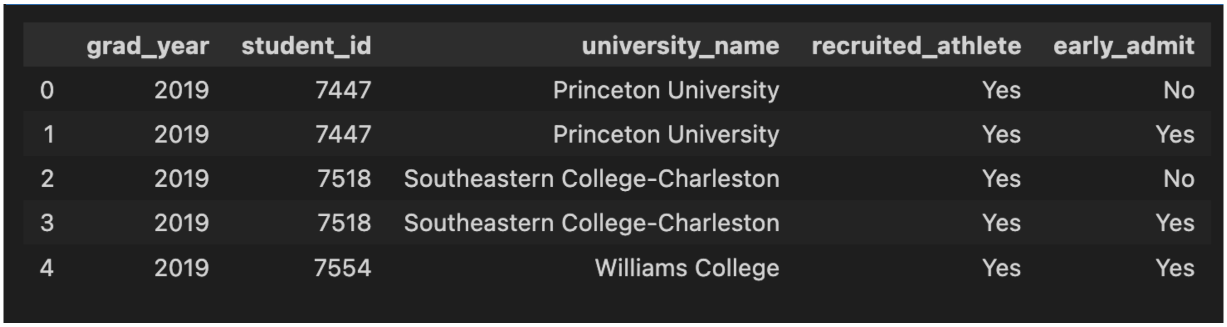
	college_id	admission_id	gpa	sat	act	matric	university_id
0	1	10528	11.00	1580.0	35.0	University of Southern Maine	161554
1	2	10532	10.95	1520.0	0.0	American International College	164447
2	3	16650	10.93	1550.0	0.0	Saint Elizabeth School of Nursing	152497
3	4	243176	10.92	1580.0	0.0	Tufts University	168148
4	5	12785	10.90	1570.0	0.0	Boston College	164924

Query 5: Return a list of all students who were recruited athletes grouped by graduation year. Include the university name that the student ended up matriculating at and whether the student was an early admit or not.

Query

```
SELECT
     Admission.grad_year,
     Students.student_id,
     Universities.institution_name AS university_name,
     CASE
     WHEN College.hook_athlete = 'True' THEN 'Yes'
     ELSE 'No'
     END AS recruited_athlete,
     CASE
     WHEN Applications.ed = 1 THEN 'Yes'
     ELSE 'No'
     END AS early_admit
FROM Students
INNER JOIN
     Admission ON Students.admission_id = Admission.admission_id
INNER JOIN
     College ON Students.college_id = College.college_id
INNER JOIN
     Universities ON College.university_id = Universities.university_id
INNER JOIN
     Applications ON Students.student_id = Applications.student_id
WHERE
     College.hook_athlete = 'True'
GROUP BY
     Admission.grad_year, Students.student_id,
Universities.institution_name, Applications.ed;
```

Result (Head of the dataframe; full results in our GitHub repository)



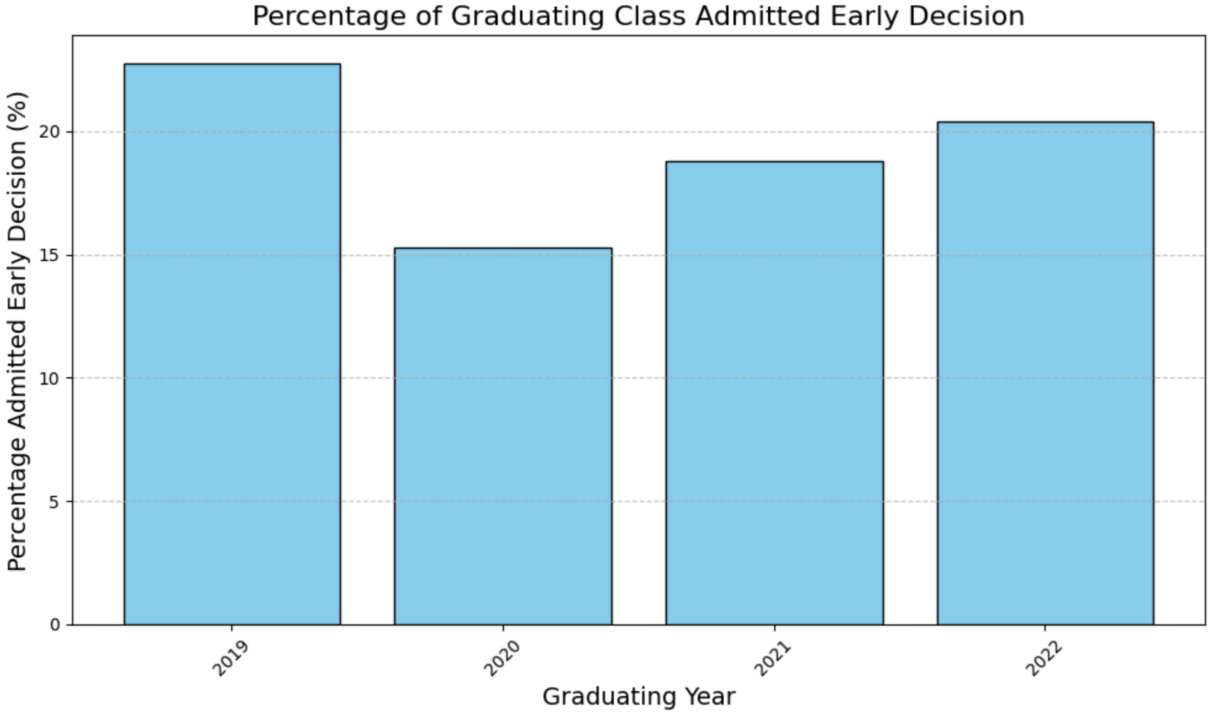
	grad_year	student_id	university_name	recruited_athlete	early_admit
0	2019	7447	Princeton University	Yes	No
1	2019	7447	Princeton University	Yes	Yes
2	2019	7518	Southeastern College-Charleston	Yes	No
3	2019	7518	Southeastern College-Charleston	Yes	Yes
4	2019	7554	Williams College	Yes	Yes

Query 6: What percentage of each graduating class was admitted early decision?

Query

```
SELECT
     Admission.grad_year AS graduating_year,
     COUNT(DISTINCT CASE
     WHEN Applications.ed = 1 AND Applications.admitted = 1 THEN
Applications.application_id
     END) * 1.0 /
     COUNT(DISTINCT CASE
     WHEN Applications.admitted = 1 THEN Applications.application_id
     END) * 100 AS percentage_early_decision_admitted
FROM
     Admission
JOIN
     Students ON Admission.admission_id = Students.admission_id
JOIN
     Applications ON Students.student_id = Applications.student_id
GROUP BY
     Admission.grad_year
ORDER BY
     Admission.grad_year
```

Result

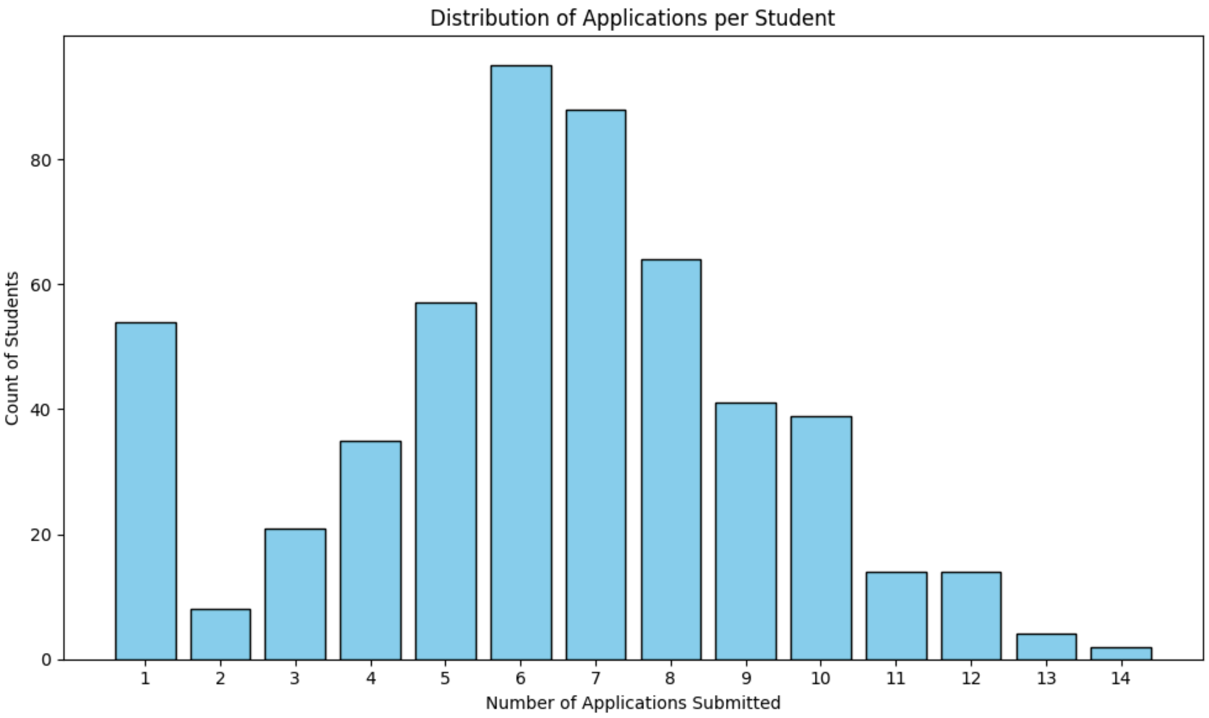


Query 7: Count the number of submitted applications per student. Create a bar graph of the distribution of the number of applications per student

Query

```
WITH temp_apps AS (
     SELECT
     DISTINCT Applications.student_id,
     COUNT(Applications.student_id) AS App_Count
     FROM Applications
     GROUP BY Applications.student_id)
SELECT DISTINCT App_Count, COUNT(App_Count) AS Num_Students
FROM temp_apps
GROUP BY App_Count;
```

Result

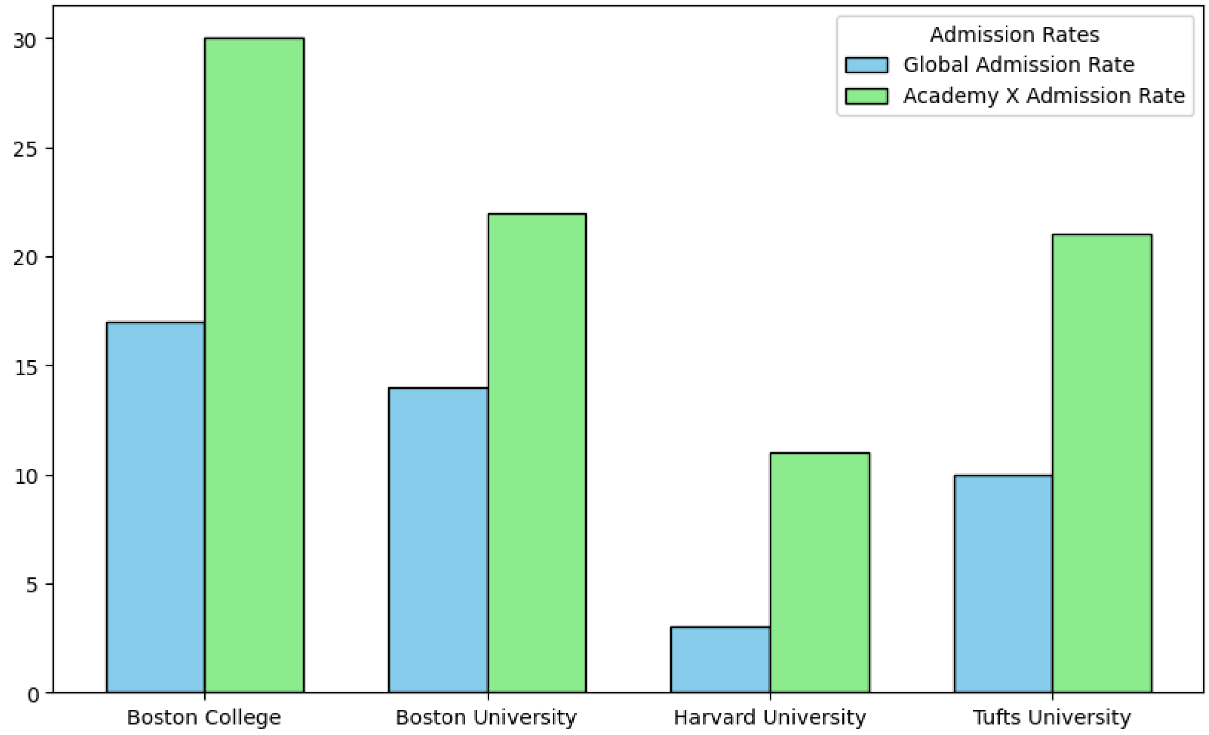


Query 8: Compare Academy X's admission rate into popular Boston Universities and compare this to each University's global admit rate

Query (Note: an "academy_percent_admitted" column was added with Pandas immediately after this query. This was calculated by dividing admitted by applications_count.)

```
SELECT
  Universities.institution_name,
  Universities.percent_admitted,
  COUNT(Applications.application_id) AS applications_count,
  COUNT(
     CASE WHEN Applications.admitted == '1' THEN 1
     END) AS admitted
FROM Applications
JOIN Universities ON Applications.university_id =
Universities.university_id
WHERE Universities.institution_name IN ('Harvard University', 'Boston
College', 'Tufts University', 'Boston University')
GROUP BY Universities.institution_name;
```

Result

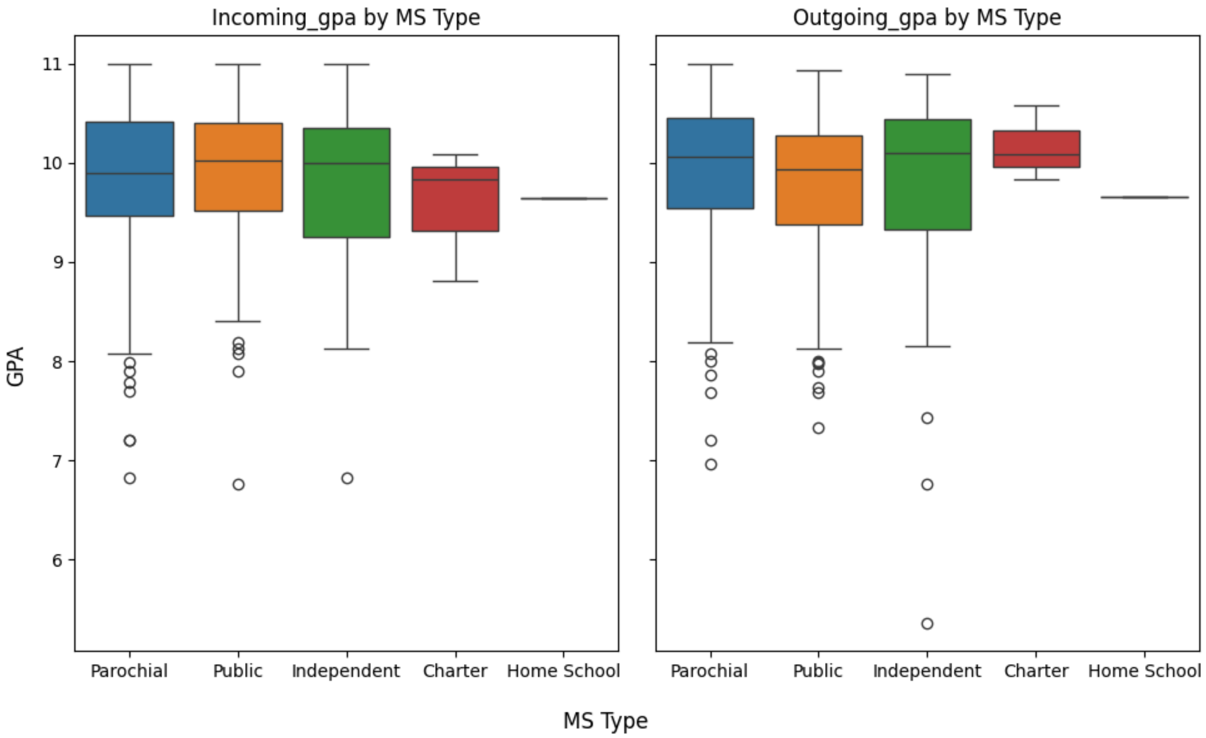


Query 9: Create a pair of boxplots comparing incoming and outgoing GPA grouped by MS type

Query

```
SELECT
  Admission.incoming_gpa,
  College.gpa AS outgoing_gpa,
  Sending_ms.ms_type
FROM Admission
JOIN College ON Admission.admission_id = College.admission_id
JOIN Sending_ms ON Admission.sending_ms_id = Sending_ms.sending_ms_id;
```

Result



5 Discussion

Interpret the results and discuss their implications. Compare the findings with the literature review and explain any discrepancies.

6 Conclusion

Summarize the key findings of the project. Discuss the limitations and suggest areas for future research.

7 References References

A Appendix A: Code

Include any relevant code used in the project. For example:

```
import pandas as pd
# Load data
df = pd.read_csv(’data.csv’)
# Preprocess data
df = df.dropna()
```

Listing 1: Example Python Code

B Appendix B: Additional Figures

Include any additional figures or tables that support the analysis.