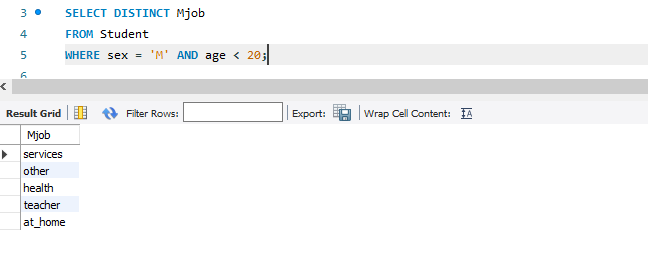
# Company: JustPlay

# Name: Shweta Sasidharan

**SQL Queries:**

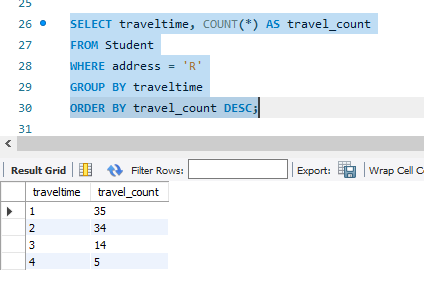
1. **List of unique “mother’s job” for male students younger than 20 years old.**

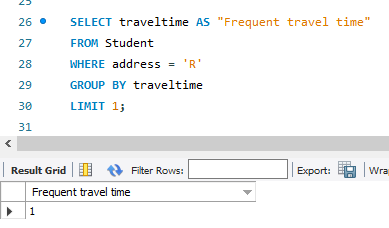
Since, the query is straightforward with some filters, this is a direct select statement.



1. **Most frequent “travel time” among students that live in rural areas**

I checked what is the frequency of the travel time among the students in the dataset. And then used limit in Descending order to choose the most frequent value.

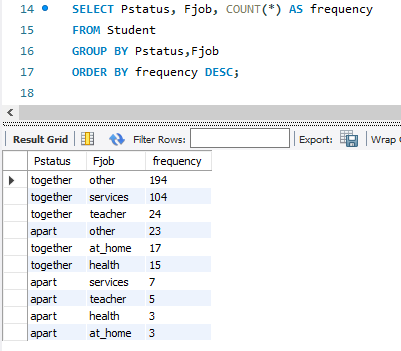


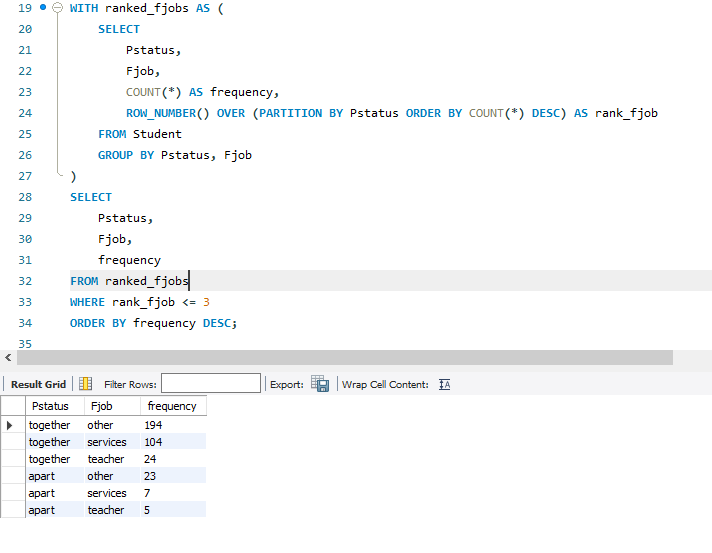


1. **Top 3 “father’s job” for students grouped by parent’s cohabitation status.**

My assumption for this query is that the Top 3 in each Cohabitation status had to be studied. So I initially calculated and queried for the frequency for the Father’s job status for each status.

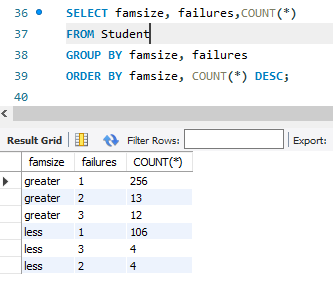
And then used a CTE query to rank the frequency and then limit to the Top 3 values.



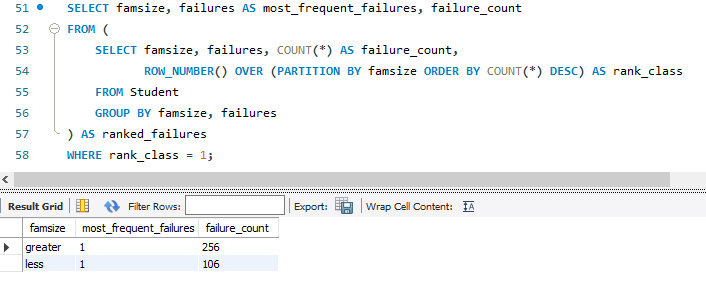


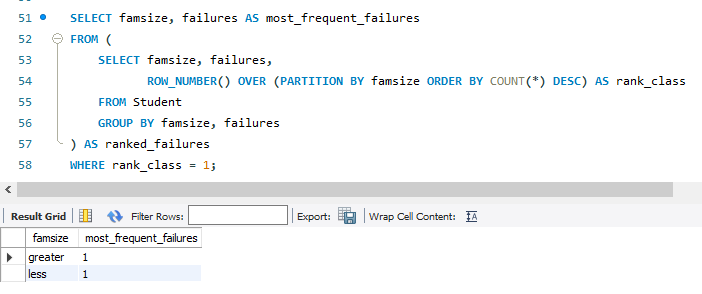
1. **Most frequent “class failures” label grouped by family sizes.**

First checked the total count of failure categories and famsize grouped together.



Then ranked it to have the most frequent Failure category in each Famsize value. And then finally displayed the frequent class in both family size categories which was the same as less than 3.

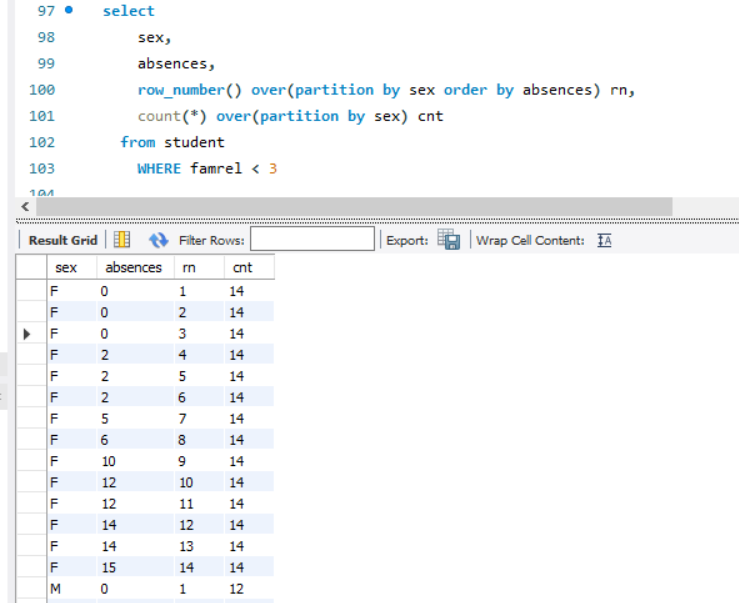


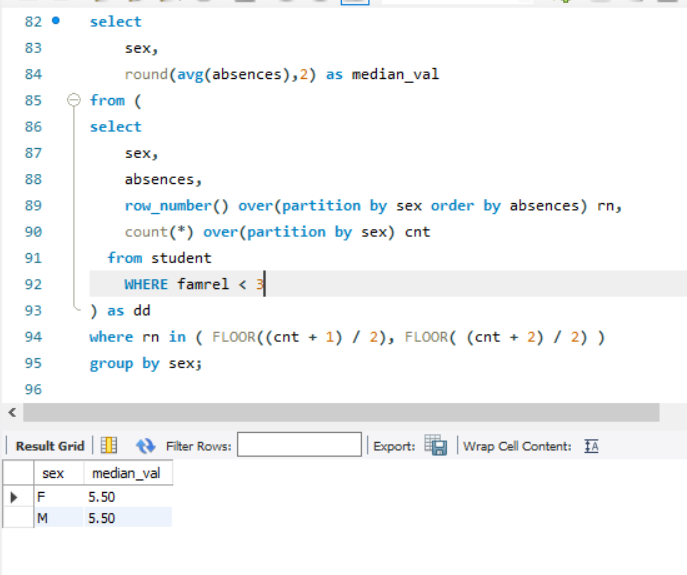


**5) Median “absences” for average and low family relationship qualities, group by sex.**

For Median calculations, the values need to be initially ordered in ascending order to calculate the median values. This was done using the row number functionality, which was ordered by absences column.

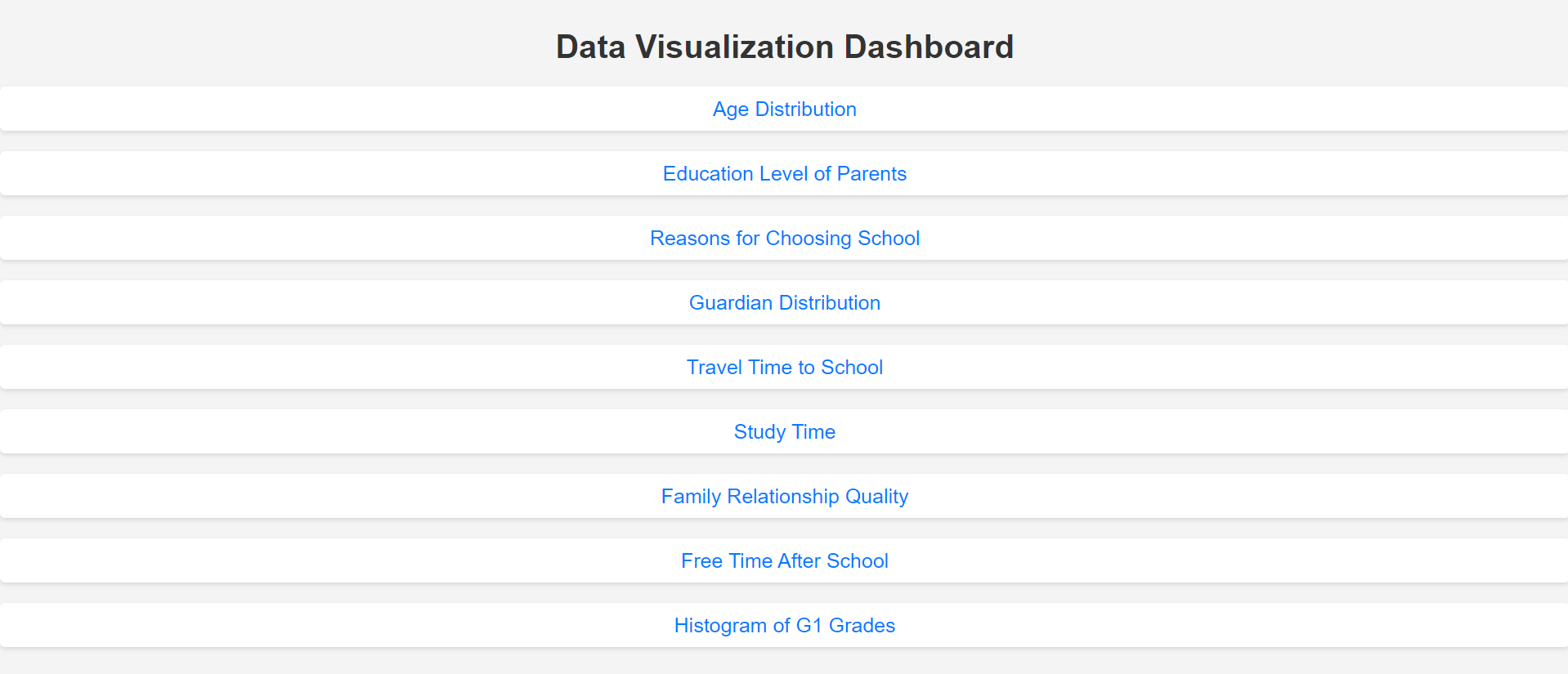
And then the mid values range, used the where condition to only get the average values for those rows. This was a reference from this blog post [1]

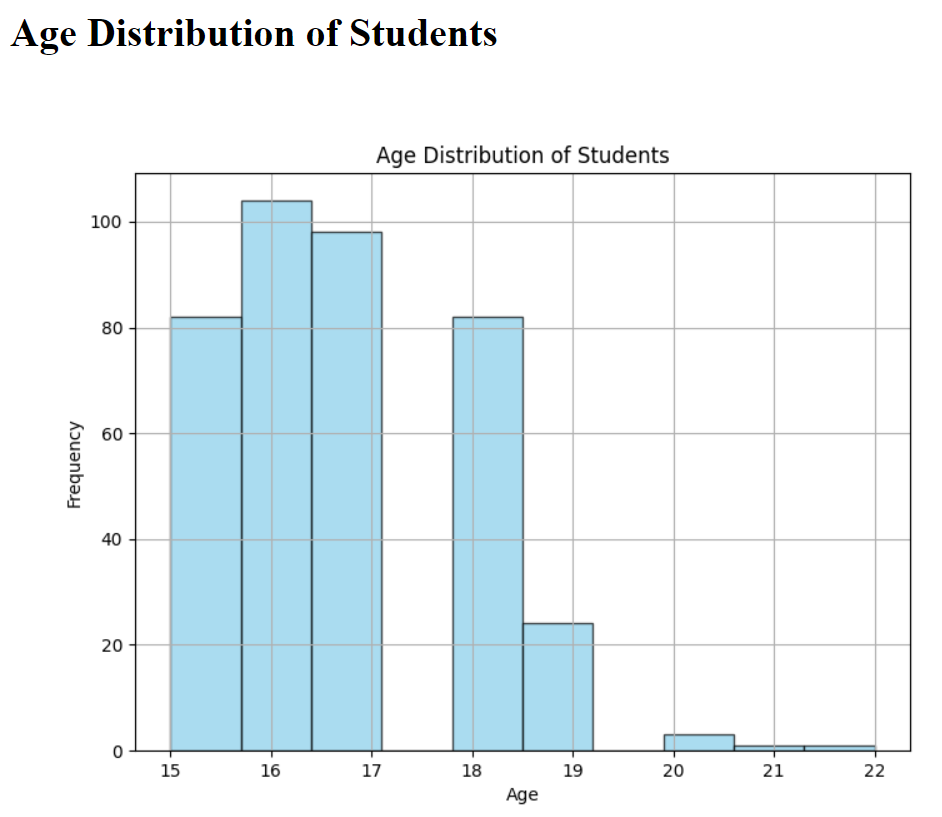


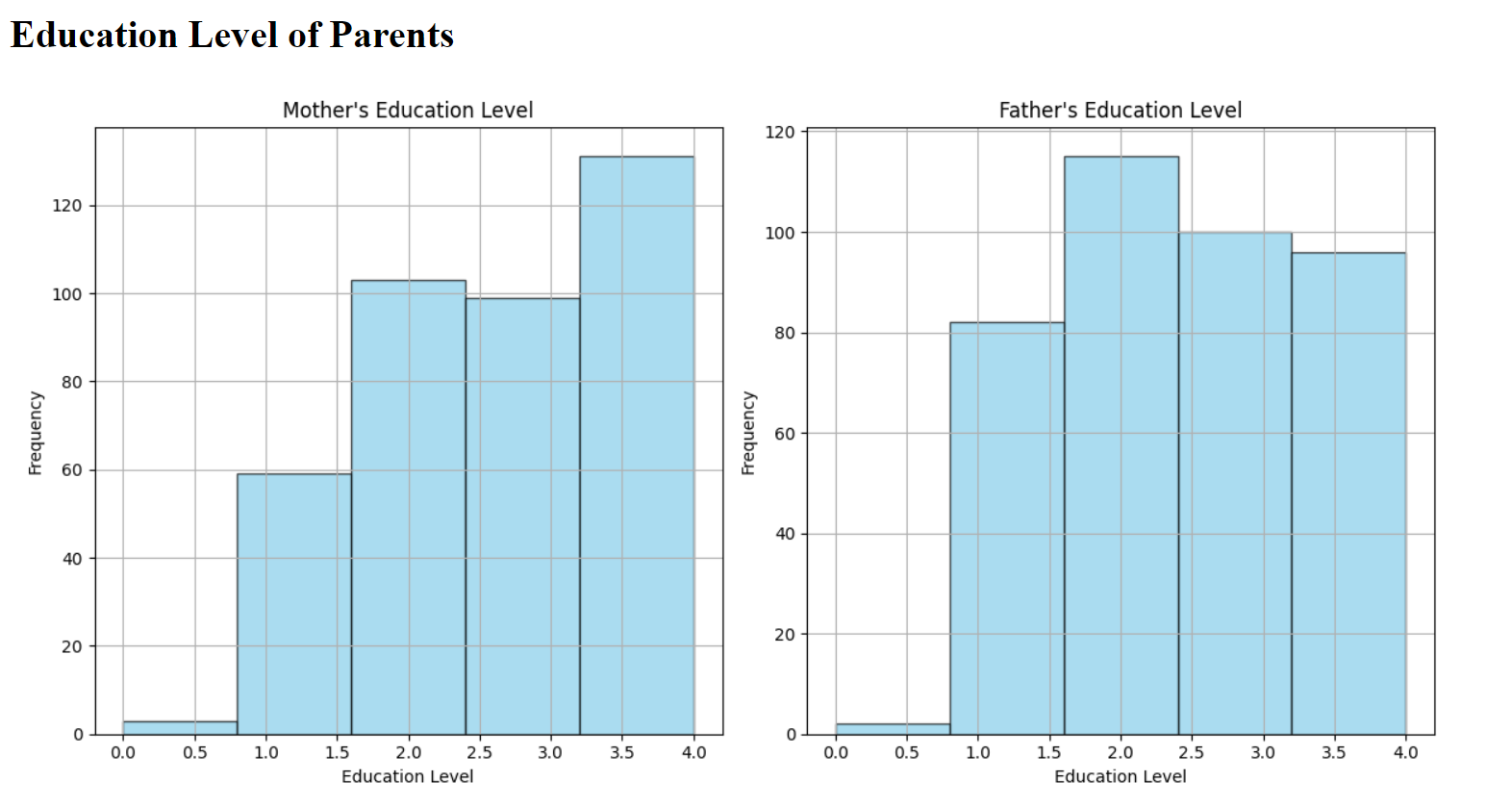


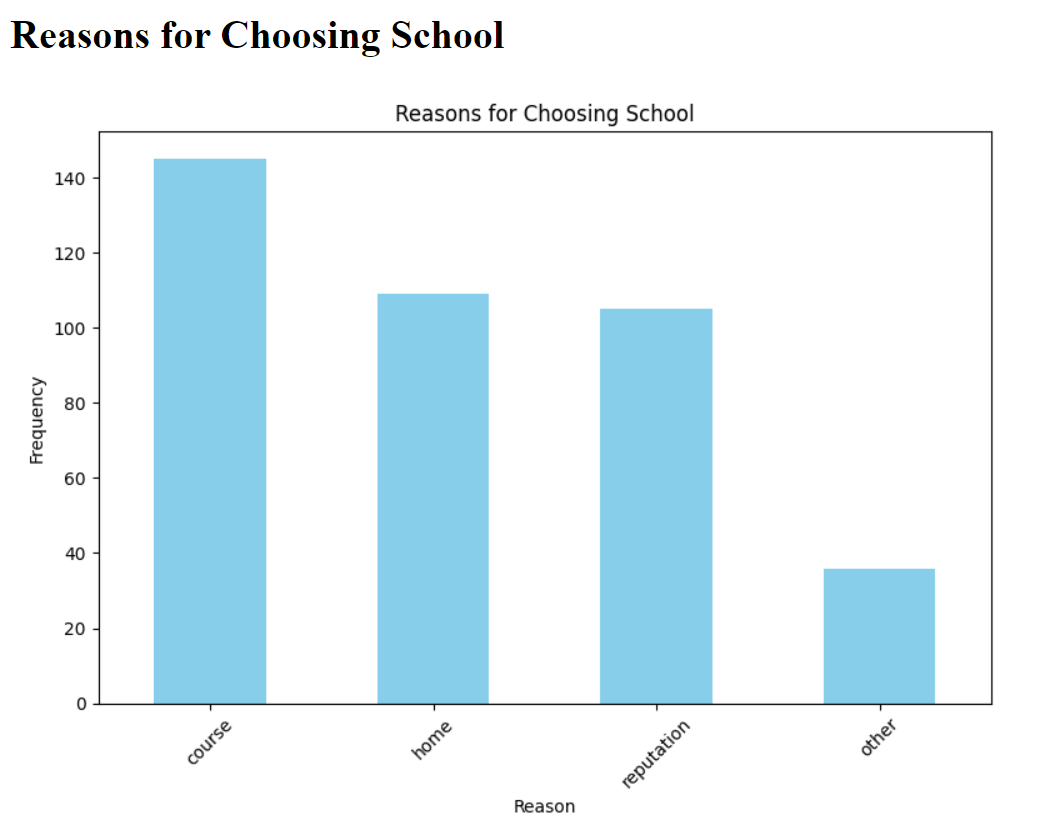
**Additional Insights:**

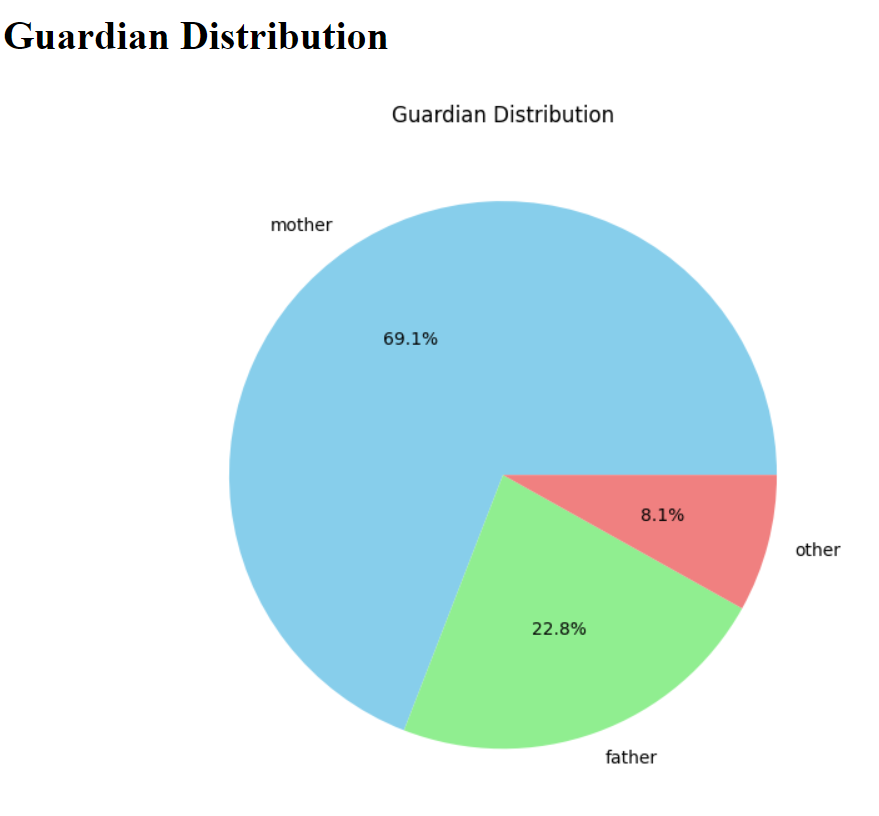
**Some of the further visualizations and observations from the data is displayed on the Flask app.**

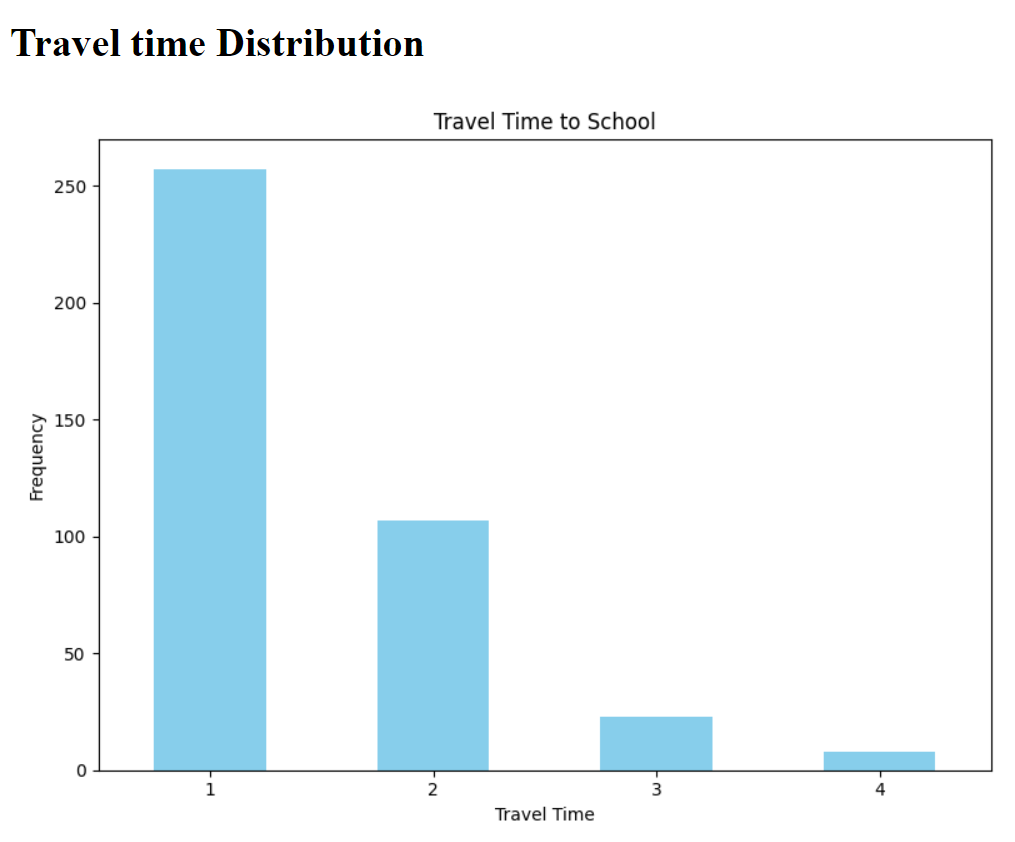


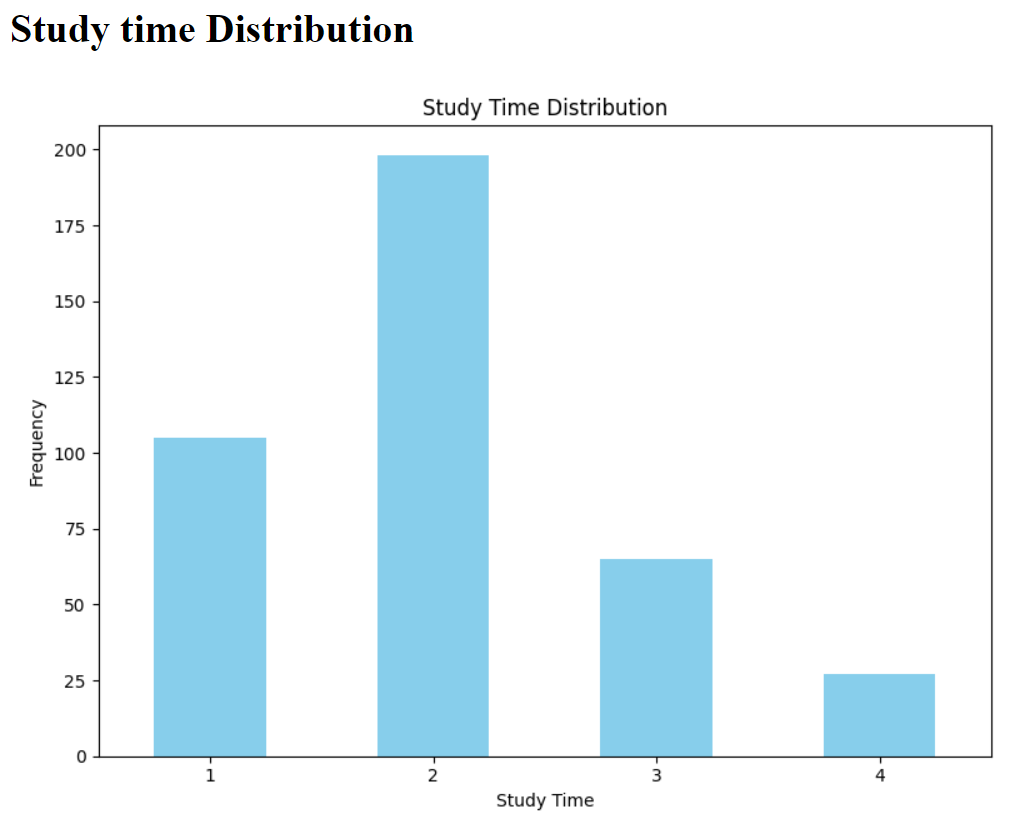


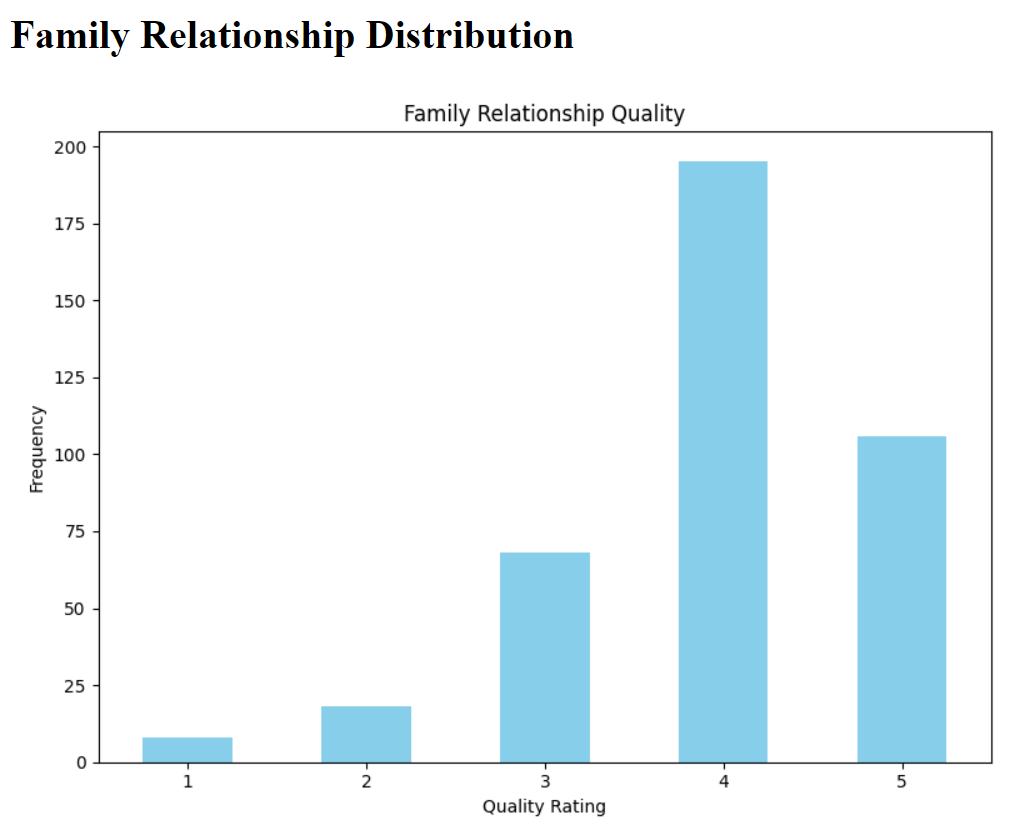


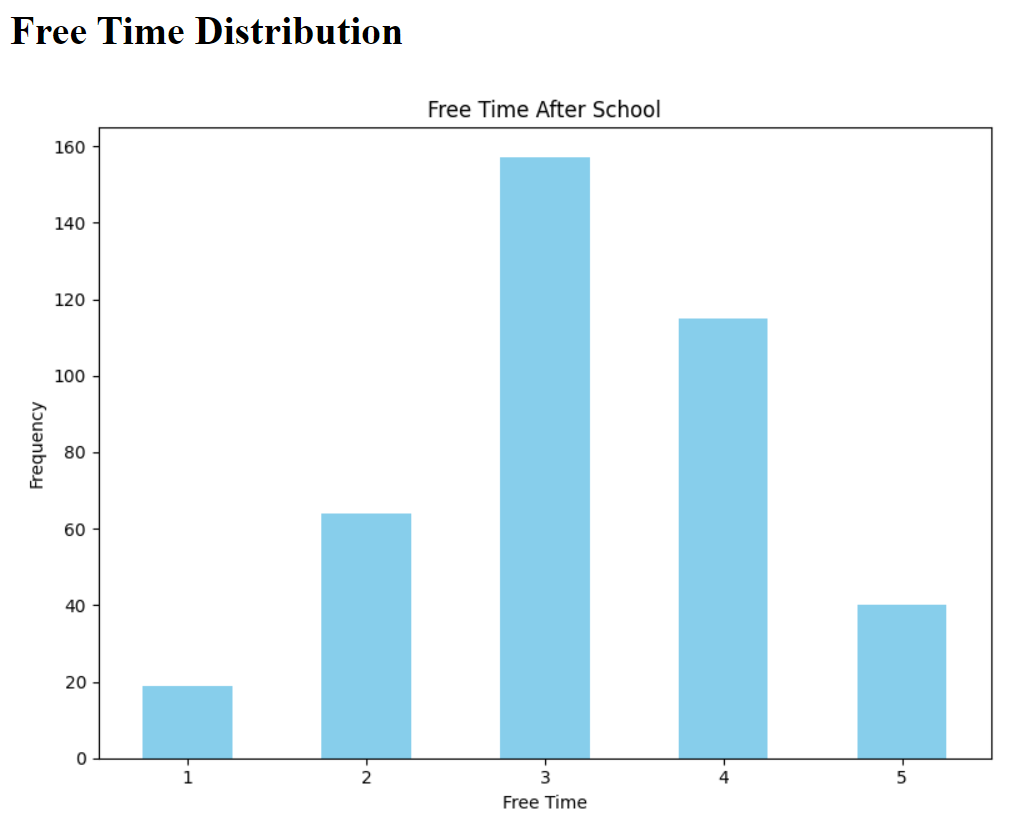


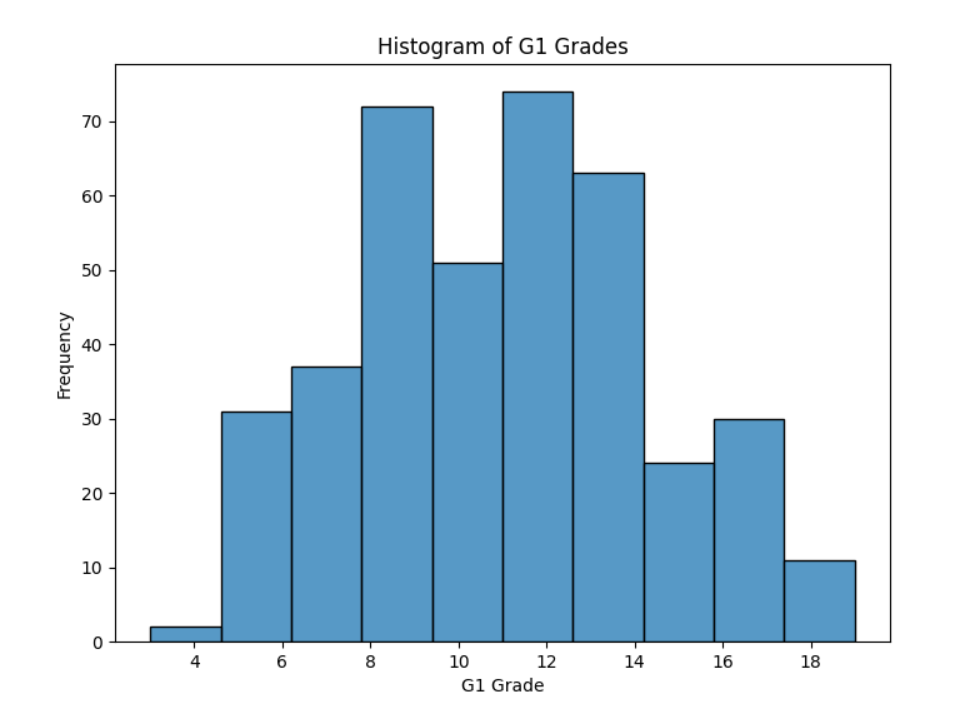








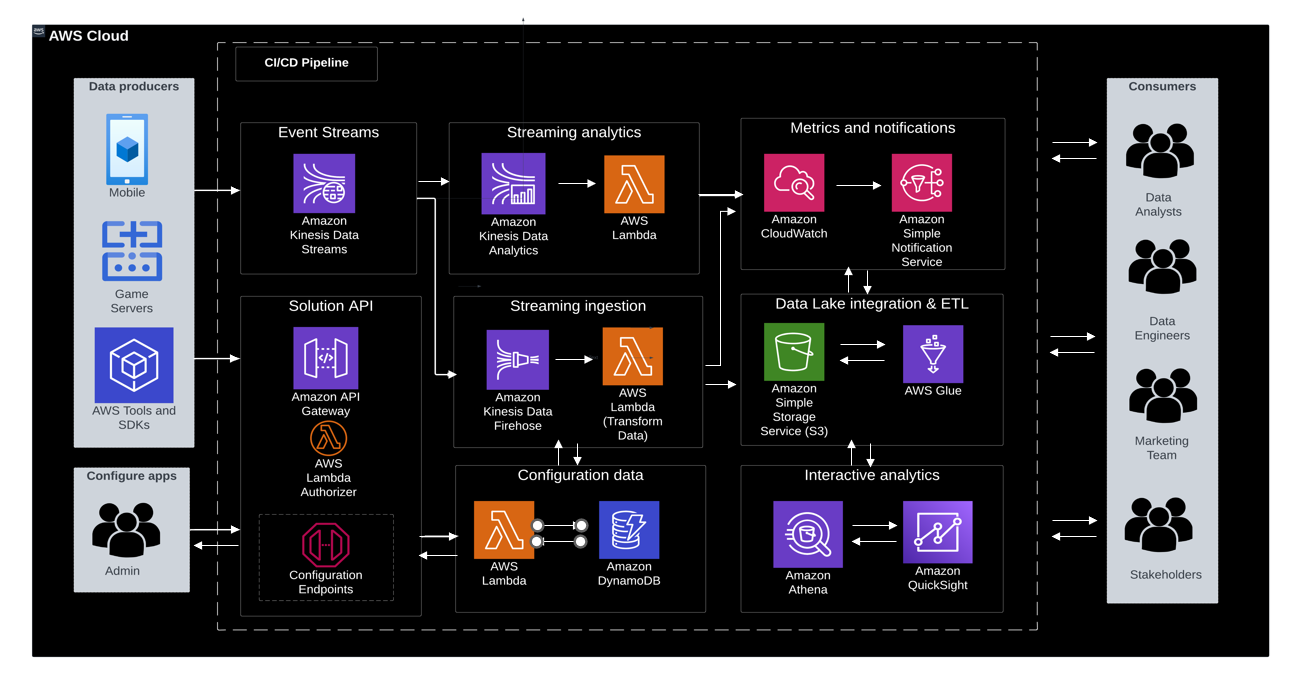




**To run the app :**

*python app.py*

**Architecture Design:**



The proposed architecture is with the AWS Cloud technologies. The following is the brief overview of the architecture:

**Solution API and configuration data:** The Amazon API Gateway offers REST API endpoints for registering game applications with the solution, ingesting game telemetry data, and sending events to Amazon Kinesis Data Streams (KDS). The game application configurations and API keys are stored in Amazon DynamoDB, which are used when sending events to the solution API.

**Event streaming**: KDS captures streaming data from the game, allowing real-time data processing through Amazon Kinesis Data Firehose and Amazon Kinesis Data Analytics.

**Streaming analytics**: Kinesis Data Analytics analyzes the streaming event data from KDS to generate customized metrics. These metrics are processed using AWS Lambda and published to Amazon CloudWatch.

**Metrics and notifications:** CloudWatch monitors, logs, and generates alarms for your AWS resources, creating an operational dashboard. It also stores the metrics generated by Kinesis Data Analytics. Amazon Simple Notification Service (Amazon SNS) delivers notifications to solution administrators and other data consumers when CloudWatch alarms are breached.

**Streaming ingestion:** Kinesis Data Firehose consumes data from KDS and invokes AWS Lambda with batches of events for serverless data processing and transformation before delivering the data to Amazon S3.

**Data lake integration and ETL:** Amazon S3 provides storage for both raw and processed data. AWS Glue handles the extract, transform, and load (ETL) processing workflows and metadata storage in the AWS Glue Data Catalog, which serves as the foundation for a data lake that can be integrated with various analytics tools.

**Interactive analytics:** Amazon Athena sample queries are deployed to enable analysis of game events. These queries can be easily integrated with Amazon QuickSight for reporting and visualization insights. The end users can then modify the data or work with the storage data to make transformations or query the data.

References:

[1] <https://sqlperformance.com/2012/08/t-sql-queries/median>