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.

```
3 •  SELECT DISTINCT Mjob
4    FROM Student
5    WHERE sex = 'M' AND age < 20;
6
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Mjob
services
other
health
teacher
at_home

2) 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.

```
25
26 •  SELECT traveltime, COUNT(*) AS travel_count
27    FROM Student
28    WHERE address = 'R'
29    GROUP BY traveltime
30    ORDER BY travel_count DESC;
31
```

Result Grid | Filter Rows: | Export: | Wrap Cell C

traveltime	travel_count
1	35
2	34
3	14
4	5

```
25
26 •  SELECT traveltime AS "Frequent travel time"
27    FROM Student
28    WHERE address = 'R'
29    GROUP BY traveltime
30    LIMIT 1;
31
```

Result Grid | Filter Rows: | Export: | Wra

	Frequent travel time
▶	1

3) 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.

```
14 •  SELECT Pstatus, Fjob, COUNT(*) AS frequency
15    FROM Student
16    GROUP BY Pstatus,Fjob
17    ORDER BY frequency DESC;
18
```

Result Grid | Filter Rows: | Export: | Wrap

	Pstatus	Fjob	frequency
▶	together	other	194
	together	services	104
	together	teacher	24
	apart	other	23
	together	at_home	17
	together	health	15
	apart	services	7
	apart	teacher	5
	apart	health	3
	apart	at_home	3

```sql
WITH ranked_fjobs AS (
    SELECT
        Pstatus,
        Fjob,
        COUNT(*) AS frequency,
        ROW_NUMBER() OVER (PARTITION BY Pstatus ORDER BY COUNT(*) DESC) AS rank_fjob
    FROM Student
    GROUP BY Pstatus, Fjob
)
SELECT
    Pstatus,
    Fjob,
    frequency
FROM ranked_fjobs
WHERE rank_fjob <= 3
ORDER BY frequency DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Pstatus	Fjob	frequency
together	other	194
together	services	104
together	teacher	24
apart	other	23
apart	services	7
apart	teacher	5

4) Most frequent “class failures” label grouped by family sizes.

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

```sql
SELECT famsize, failures,COUNT(*)
FROM Student
GROUP BY famsize, failures
ORDER BY famsize, COUNT(*) DESC;
```

Result Grid | Filter Rows: | Export:

famsize	failures	COUNT(*)
greater	1	256
greater	2	13
greater	3	12
less	1	106
less	3	4
less	2	4

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.

```sql
SELECT famsize, failures AS most_frequent_failures, failure_count
FROM (
    SELECT famsize, failures, COUNT(*) AS failure_count,
           ROW_NUMBER() OVER (PARTITION BY famsize ORDER BY COUNT(*) DESC) AS rank_class
    FROM Student
    GROUP BY famsize, failures
) AS ranked_failures
WHERE rank_class = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

famsize	most_frequent_failures	failure_count
greater	1	256
less	1	106

```sql
SELECT famsize, failures AS most_frequent_failures
FROM (
    SELECT famsize, failures,
           ROW_NUMBER() OVER (PARTITION BY famsize ORDER BY COUNT(*) DESC) AS rank_class
    FROM Student
    GROUP BY famsize, failures
) AS ranked_failures
WHERE rank_class = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

famsize	most_frequent_failures
greater	1
less	1

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]

```
97 •  select
98        sex,
99        absences,
100       row_number() over(partition by sex order by absences) rn,
101       count(*) over(partition by sex) cnt
102     from student
103       WHERE famrel < 3
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

sex	absences	rn	cnt
F	0	1	14
F	0	2	14
F	0	3	14
F	2	4	14
F	2	5	14
F	2	6	14
F	5	7	14
F	6	8	14
F	10	9	14
F	12	10	14
F	12	11	14
F	14	12	14
F	14	13	14
F	15	14	14
M	0	1	12

```sql
select
    sex,
    round(avg(absences),2) as median_val
from (
select
    sex,
    absences,
    row_number() over(partition by sex order by absences) rn,
    count(*) over(partition by sex) cnt
  from student
    WHERE famrel < 3
) as dd
where rn in ( FLOOR((cnt + 1) / 2), FLOOR( (cnt + 2) / 2) )
group by sex;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

sex	median_val
F	5.50
M	5.50

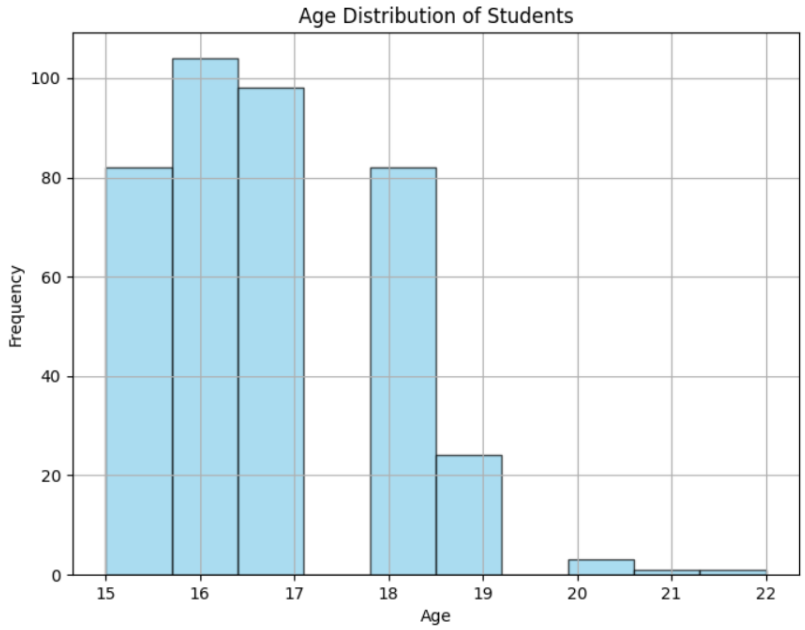
Additional Insights:

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

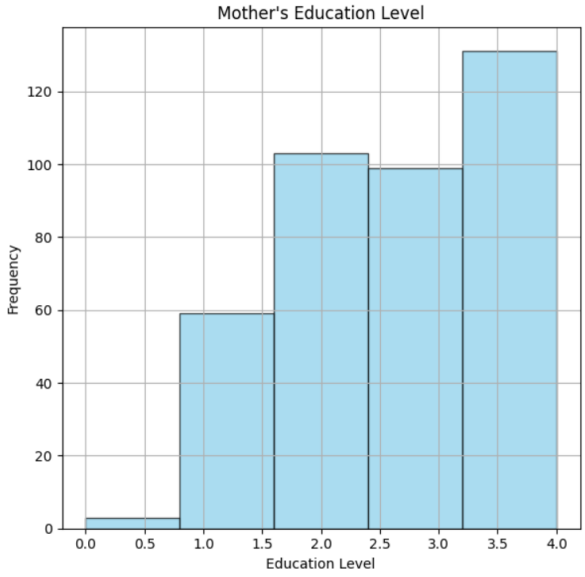
Data Visualization Dashboard

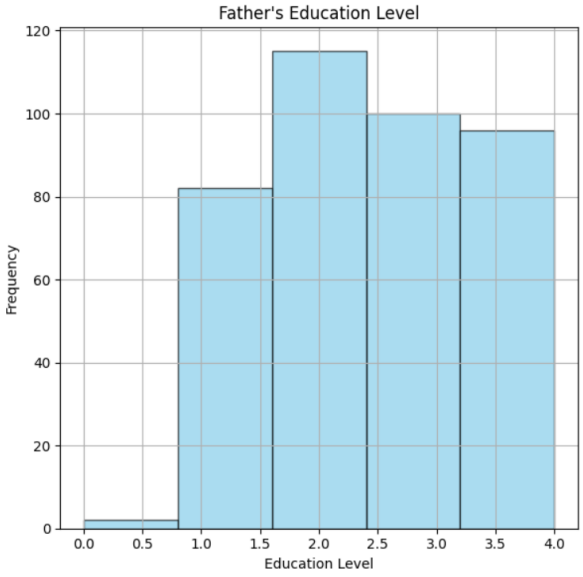
- Age Distribution
- Education Level of Parents
- Reasons for Choosing School
- Guardian Distribution
- Travel Time to School
- Study Time
- Family Relationship Quality
- Free Time After School
- Histogram of G1 Grades

Age Distribution of Students

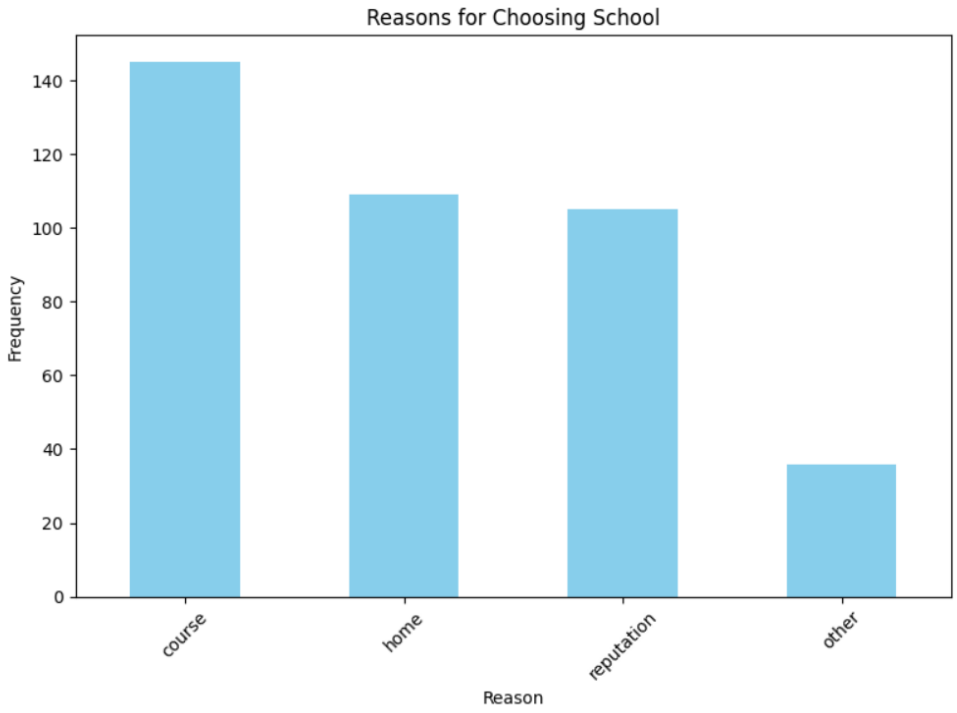


Education Level of Parents

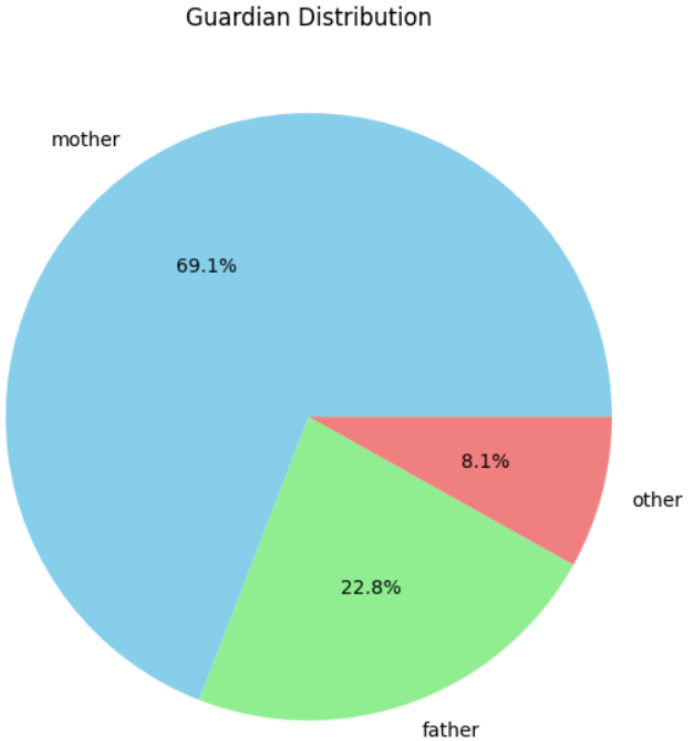




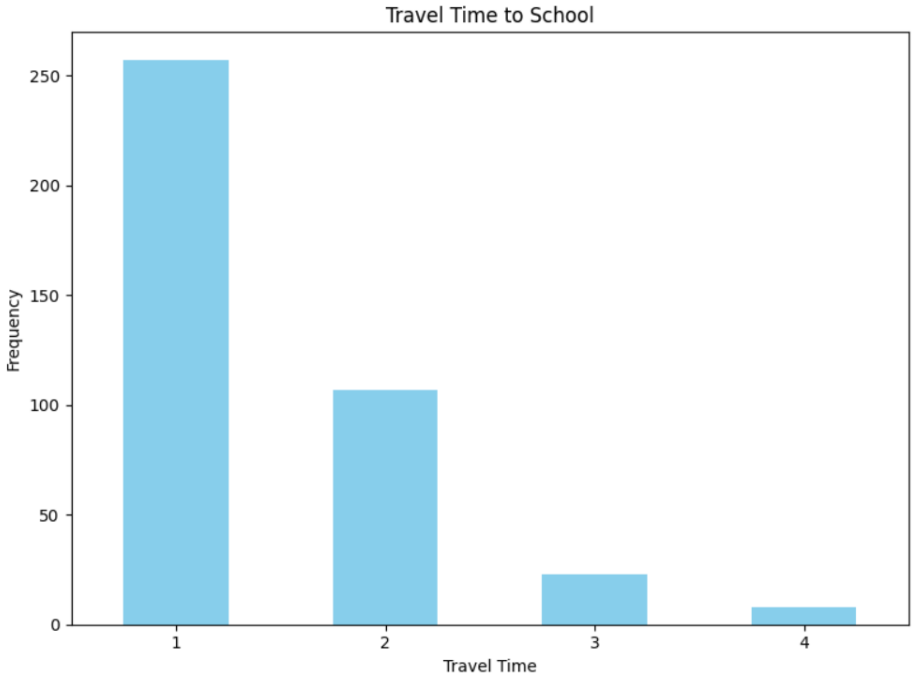
Reasons for Choosing School



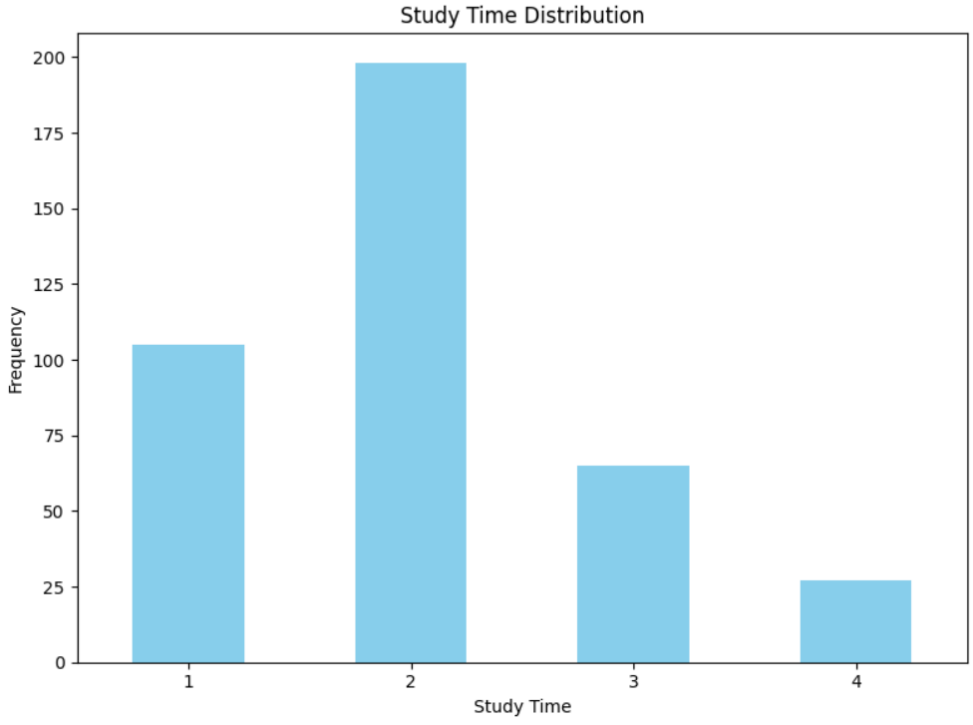
Guardian Distribution



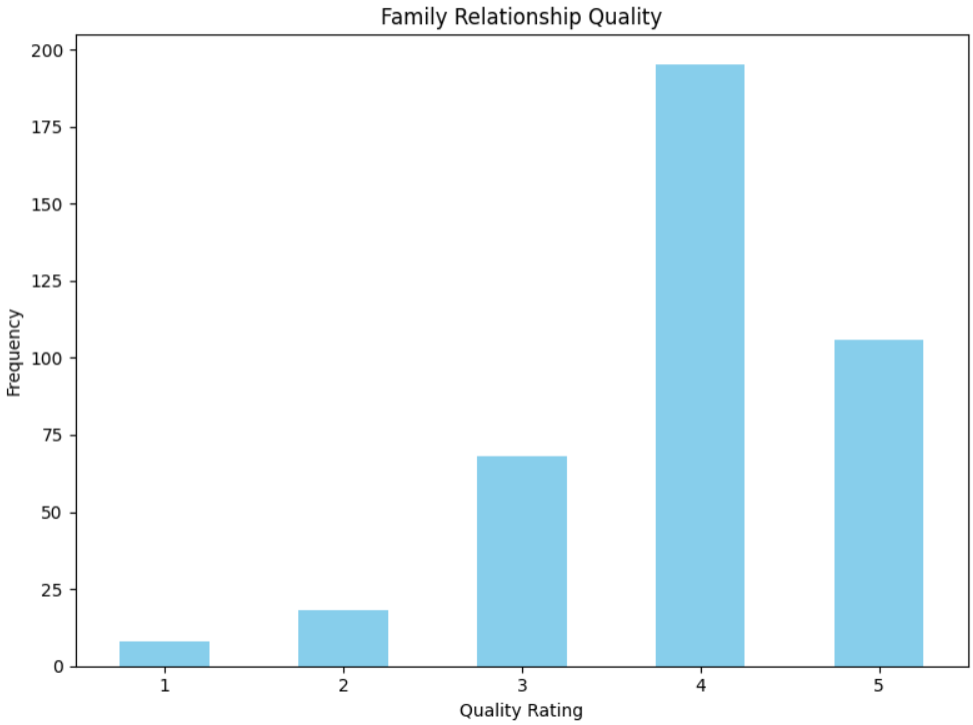
Travel time Distribution



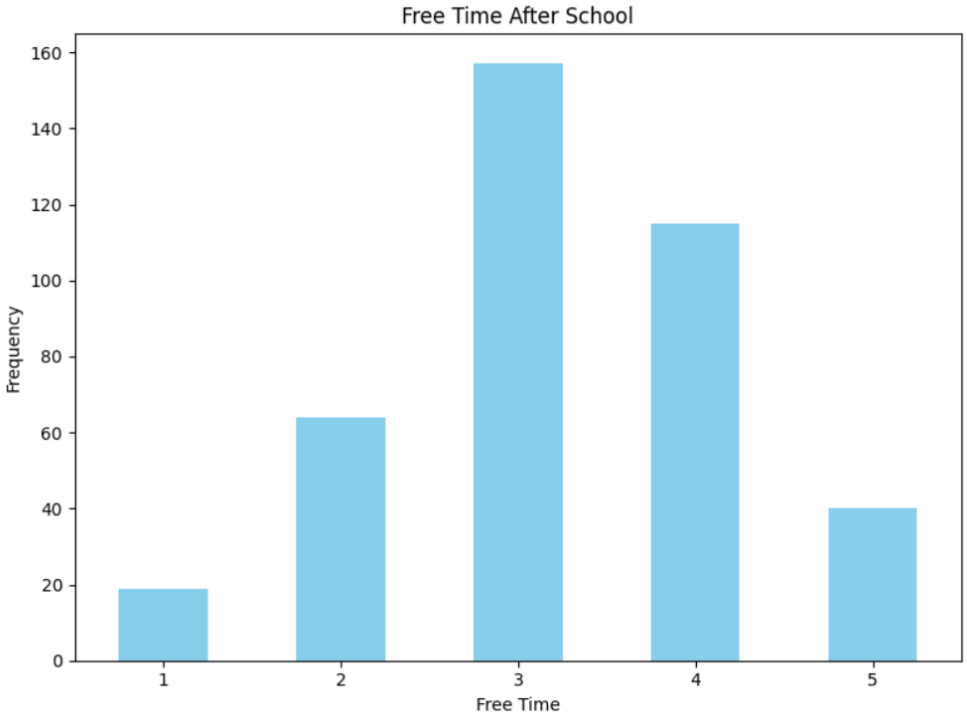
Study time Distribution

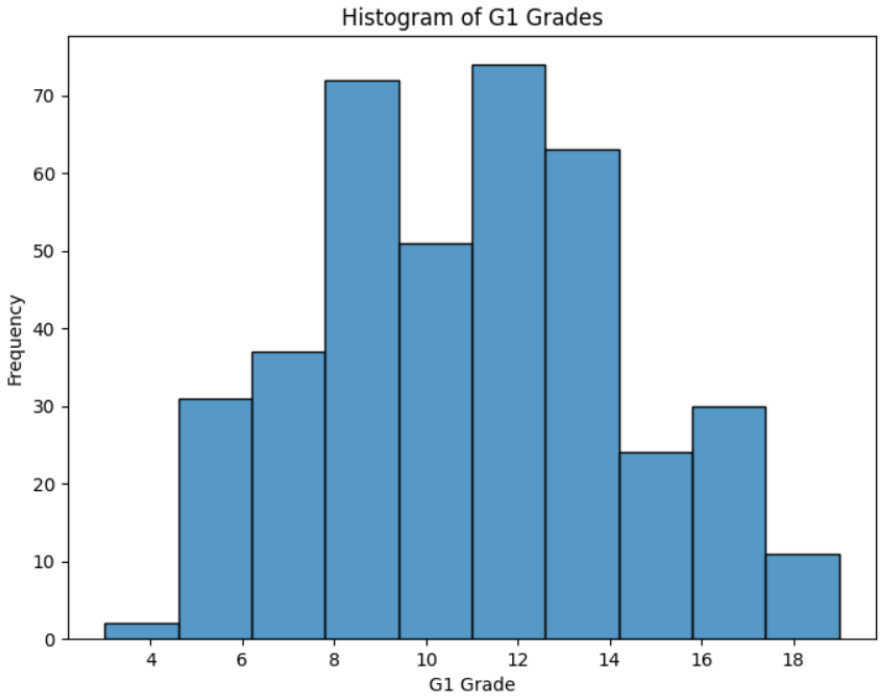


Family Relationship Distribution



Free Time Distribution

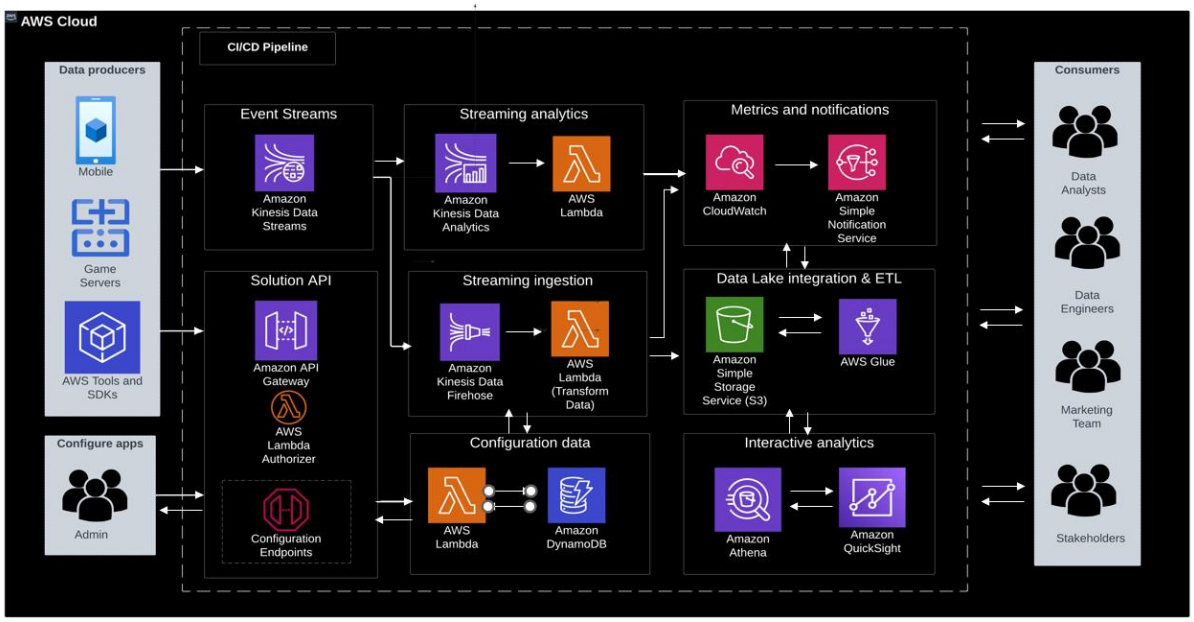




To run the app :

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