US Household Income Exploratory Data Analysis in MySQL

By: Xavier Lancelot Veloso

Project Background

This project delves into the utilization of MySQL in the context of data transformation, exploratory analysis, and automation, focusing on its application in handling the US Household Income dataset to derive meaningful insights and identify general patterns in the data. These patterns include outliers and features of the data that might be unexpected.

My professional engagement within this project entails the utilization of MySQL as a pivotal tool for facilitating enhanced comprehension of the dataset among stakeholders. These stakeholders represent a broad audience, primarily individuals keen on understanding various facets of salary distributions, geographic trends across states and cities, and other pertinent insights.

With a keen awareness of stakeholder interests, my approach involves presenting an updated exploratory analysis of the dataset, highlighting states and cities characterized by the highest average incomes alongside other salient categories that offer valuable insights.

Summary & Key Insights

This project outlined the utilization of MySQL for data transformation, exploratory analysis, and automation for data cleaning. Data transformation involved cleaning inconsistencies, null values, and missing values. Exploratory data analysis unveiled income distribution across states, income types, and cities. Finally, an automated system using events, stored procedures and triggers was implemented to clean data at specific intervals and upon data insertion.

This project leveraged MySQL to explore a dataset of US household income and geographic locations. Key findings include Texas having the largest land area and Michigan boasting the most water area. Analyzing income data, we discovered the District of Columbia has the highest average income within the top ten states, but also the lowest median income. Boroughs hold the top spot for average income by type, though with a low count. Tracks, on the other hand, have a significantly higher count but a much lower average income, hinting at possible data errors. Further analysis revealed Delta Junction City with the highest average salary, but there's a chance the data has a cap of 300000 for median salaries.

Overall, this project showcased the potential of MySQL for data exploration and management, uncovering valuable insights from datasets. However, the project

acknowledges limitations in data quality and the need for potential further cleaning based on domain knowledge.

Dataset Overview

The dataset originally developed for real estate and business investment research. Income is a vital element when determining both quality and socioeconomic features of a given geographic location.

The database contains records on US Household Income Statistics & Geo Locations derived from over +36,000 files and covers 348,893 location records around the United States. The two datasets are Household & Geographic Statistics and Geographic Location. Household & Geographic Statistics have the statistical computations of the household incomes while the Geographic location have the geographic information.

Data Preparation

The datasets that were imported to MySQL are named as ushouseholdincome for the Geographic Location and ushouseholdincome_statistics for the Household & Geographic Statistics under the 'project' schema.

Household & Geographic Statistics:

- State Name (character)
- Mean Household Income (double)
- Median Household Income (double)
- Standard Deviation of Household Income (double)
- Number of Households (double)

Geographic Location:

- Longitude (double)
- Latitude (double)
- State Name (character)
- State abbreviated (character)
- State_Code (character)
- County Name (character)
- City Name (character)
- Name of city, town, village or CPD (character)
- Primary, Defines if the location is a track and block group.
- Zip Code (character)
- Area Code (character)
- Square area of land at location (double)
- Square area of water at location (double)

I conducted an initial exploration on the two datasets using the following queries, that will also be used to check the data after each query update.

```
# Data Exploration
SELECT *
FROM project.ushouseholdincome;
SELECT *
FROM project.ushouseholdincome_statistics;
```

Data Transformation

Before transforming the data, I made a backup copy of the dataset csv file in case of an error.

Upon checking the tables, one of the columns in ushousegoldincome_statistics.csv is in written as `ï»¿id`, so I wrote the following query to rename the column name to `id`.

```
#Renamed `ï»¿id` to `id`
ALTER TABLE project.ushouseholdincome_statistics RENAME COLUMN ï»¿id TO `id`;
```

id	State_Name	Mean	Median	Stdev	sum_w
1011000	Alabama	38773	30506	33101	1638.260513
1011010	Alabama	37725	19528	43789	258.0176847
1011020	Alabama	54606	31930	57348	926.0309998
1011030	Alabama	63919	52814	47707	378.1146191
1011040	Alabama	77948	67225	54270	282.3203278

After changing the column name

After checking the column names, I run a count on both tables to check for verification. The ushouseholdincome table returned with 32292 while ushouseholdincome_statistics table returned with 32526. Since the difference is only minimal, I chose to ignore the difference.

To check for duplicates, I used the following query on both tables to check the column id count that is more than one.

```
# To check for duplicates
SELECT id, count(id)
FROM project.ushouseholdincome
GROUP BY id
HAVING COUNT(id) > 1
;
SELECT id, count(id)
FROM project.ushouseholdincome_statistics
GROUP BY id
HAVING COUNT(id) > 1
;
```

id	count(id)
10226	2
60213229	2
60213239	2
60213249	2
24021897	2
36024654	2

Duplicate count

The ushousehold income returned 6 rows that are duplicates while the ushouseholdincome_statistics returned 0 duplicates. Seeing as there are duplicates in ushouseholdincome, I used the following query to give the duplicated rows a unique id using row_number() to delete them.

```
#To remove duplicates
DELETE FROM project.ushouseholdincome
WHERE row_id IN (
    SELECT row_id
    FROM (
        SELECT row_id,
        id,
        ROW_NUMBER() OVER(PARTITION BY id ORDER BY id) row_num
        FROM project.ushouseholdincome
        ) duplicates
    WHERE row_num > 1)
;
```

To check for inconsistencies in the data, I used the following format to select the column, group the column and get the count.

```
# To check for inconsistencies
SELECT column_name, COUNT(column_name)
FROM table_name
GROUP BY column_name
;
```

State_Name	COUNT(State_Name)
Connecticut	355
Delaware	88
District of Columbia	64
Florida	1658
Georgia	817
georia	1
Hawaii	136

Data inconsistency 'georia'

In the SELECT clause, replace 'column_name' with a column name in the dataset to check each column for inconsistencies. In the FROM clause, replace 'table_name' with the table you want to check. One of the inconsistencies I encountered is in the State_Name column where the field inputted is 'georia', so I queried the following to replace the state name to 'Georgia'. I did the same procedure for all other inconsistencies in the dataset.

```
# To replace the wrong names with the correct names and format
UPDATE project.ushouseholdincome
SET State_Name = 'Georgia'
WHERE STATE_Name = 'georia'
;
```

To check for missing values and null values, I used the following format to select the columns to be checked and filter the null and missing values in the columns.

```
# To check for missing values and nulls
SELECT *
FROM table_name
WHERE column_name = '' OR column_name IS NULL
;
```

row_id	id	State_Code	State_Name	State_ab	County	City	Place
32	102216	1	Alabama	AL	Autauga County	Vinemont	

Missing data in the Place column

In the WHERE clause, replace 'table_name' with a column name in the dataset to check each column for missing values and null values. In the FROM clause, replace 'table_name' with the table you want to check. I found a missing value in the column 'place' and from exploring and analyzing the dataset, I deduced to replace the value with 'Autaugaville', in line with the data values close to it. I used the following query to filter by county and city to accurately select the row with the missing value and replace it with 'Autaugaville'.

```
# To replace the missing values in place
UPDATE project.ushouseholdincome
SET place = 'Autaugaville'
WHERE County = 'Autauga County'
AND City = 'Vinemont'
;
```

There are some inconsistencies, null values, and missing values that can still be seen in the dataset, but I chose to just filter them out when doing the exploratory analysis. I prefer to not delete the null values and missing value with the idea being that the dataset could be updated in the future to fill in the missing data. I also lack the domain knowledge when it comes to fixing the remaining inconsistencies and I would need to consult someone who has the appropriate domain knowledge before I could proceed transforming the data further with confidence.

Exploratory Data Analysis

The goal of this project is to explore the data and derive any interesting insights and identify general patterns.

I wanted to start looking at the top ten largest land area and top ten largest water area by state.

```
# Top 10 Area of Land by State
SELECT State_Name, SUM(ALand) AS land_area, SUM(AWater) as water_area
FROM project.ushouseholdincome
GROUP BY State_name
ORDER BY land_area DESC
LIMIT 10;
# Top 10 Area of Water by State
SELECT State_Name, SUM(ALand) AS land_area, SUM(AWater) as water_area
FROM project.ushouseholdincome
GROUP BY State_name
ORDER BY water_area DESC
LIMIT 10;
```

State_Name	land_area
Texas	173222229898
California	90456155777
Missouri	80404645532
Minnesota	74395673850
Illinois	70794312509
Kansas	69752815156
Oklahoma	67144494658
Iowa	64928024047
Wisconsin	60376360719
Michigan	60028282240

State_Name	water_area
Michigan	13544227864
Texas	7984639571
Florida	7184634980
Minnesota	4311138060
Louisiana	4011517821
California	3865613533
Alaska	3630139204
North Carolina	3584120388
Washington	3288150796
Wisconsin	3063161381

Top ten land area and water area by state

As expected, Texas has the largest land area and Michigan has the largest water area.

To analyze the income data together with the geolocation data, I needed to join ushouseholdincome and ushouseholdincome_statistics. The primary key and foreign key used to join the two tables together are the 'id' column from both tables. I used inner join to join the two tables. Doing so automatically filter's out the missing rows that was mentioned previously when I did the count of all the rows for each table. To further filter out the missing values and null values, I added a WHERE clause.

```
# Joining the two tables
SELECT u.State_Name, County, Type, `Primary`, Mean, Median
FROM project.ushouseholdincome u
# Inner join to filter out missing data from the ushouseholdincome table
INNER JOIN project.ushouseholdincome_statistics us
    ON u.id = us.id
# To Filter out missing values from ushouseholdincome_statistics table
WHERE Mean <> 0 OR Mean IS NULL
;
```

Further analyzing the data, I wanted to explore the top ten average income by states.

```
# Top 10 Average Income by States
SELECT u.State_Name, ROUND(AVG(Mean),1) as avg, ROUND(AVG(Median),1) as median
FROM project.ushouseholdincome u
INNER JOIN project.ushouseholdincome_statistics us
    ON u.id = us.id
WHERE Mean <> 0
GROUP BY u.State_Name
ORDER BY avg DESC
LIMIT 10
;
```

State_Name	avg	median
District of Columbia	90668.4	93759.6
Connecticut	89732.8	121240.4
New Jersey	89565.4	126972.7
Maryland	88444.2	114969.2
Massachusetts	85645.7	119074.4
Hawaii	82992.1	102101.8
Alaska	81311.8	122295.7
Virginia	80426.3	105313.4
California	78742.0	101359.0
New York	77859.3	102731.8

Average income by state

The state with the highest average income is the District of Columbia, but the median income is the lowest out of all states in the top ten.

Next I analyzed the top ten average income by type, but I had to add an additional filter, the HAVING clause, to filter out types that had a low count. Without the filter, the query comes back with outliers skewing the results. I suspected that there could be data errors in the types but as mentioned before, I do not have the domain knowledge to be confident enough to make changes.

```
# Top 10 Average Income by Type
SELECT Type, COUNT(Type) as count, ROUND(AVG(Mean),1) as avg, ROUND(AVG(Median),1) as median
FROM project.ushouseholdincome u
INNER JOIN project.ushouseholdincome_statistics us
    ON u.id = us.id
WHERE Mean <> 0
GROUP BY Type
# To filter out outliers
HAVING COUNT(Type) > 100
ORDER BY avg DESC
LIMIT 20
;
```

Type	count	avg	median
Borough	129	68594.4	73384.0
Track	28939	68145.1	86925.3
CDP	962	64623.3	116376.6
Village	388	61548.6	72316.7
City	1055	58220.8	64850.4
Town	473	55194.1	63846.6

Average Income by type

Borough has the highest average income in terms of type, but it does have a low count. Track has a significant number of counts, and the average income is far from Borough's average income.

Analyzing the data further, I wrote a query to check the average salaries by city.

```
# To check the average salaries by City
SELECT u.State_Name, City, ROUND(AVG(mean),1) as avg, ROUND(AVG(Median),1) as median
FROM project.ushouseholdincome u
INNER JOIN project.ushouseholdincome_statistics us
    ON u.id = us.id
GROUP BY u.State_Name, City
HAVING avg <> 0 OR median <> 0
ORDER BY avg DESC
    ;
```

State_Name	City	avg	median
Alaska	Delta Junction	242857.0	300000.0
New Jersey	Short Hills	216503.0	300000.0
Pennsylvania	Narberth	194426.0	224616.0
Maryland	Chevy Chase	194157.5	259355.5
Connecticut	Darien	192882.0	300000.0
Virginia	Great Falls	192103.5	215673.5
New York	Pelham	190909.0	212250.0
Virginia	Chantilly	190865.0	300000.0
Virginia	Mclean	188414.0	246832.3
New York	Bronxville	188005.0	244249.5

Average salaries by city

Delta Junction City has the highest average salary with a median salary of 300000. This is an odd finding since other rows also show similar values. I assumed that there is a cap of 300000 for the median salaries in the data.

Data Cleaning Automation

In a scenario where the database gets updated, using a data cleaning automation helps with cleaning data at a specific period. Using the previous queries used in data transformation, I built an automation system using events and stored procedures. This would prove useful for dynamic data that is constantly changing.

I dropped the previously worked on ushouseholdincome dataset because it was already transformed and imported the original ushouseholdincome dataset from the backup copy to use to test the data cleaning automation.

Inside the stored procedure query, first, we create a new table called ushouseholdincome_cleaned. This table mirrors the structure of an existing table named ushouseholdincome. We copy data from the original table into the new one, ensuring that each row includes a timestamp indicating when the data was inserted. Next, we address data quality issues. We correct misspelled state names (e.g., changing “georia” to “Georgia”), convert text values to uppercase for consistency (county, city, place, and state names), and standardize the Type column values (e.g., “CPD” becomes “CDP” and “Boroughs” becomes “Borough”). Overall, this procedure enhances data consistency, removes duplicates, and improves data quality for further analysis.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS Copy_and_Clean_Data;
CREATE PROCEDURE Copy_and_Clean_Date()
BEGIN
#Creating the table
    CREATE TABLE IF NOT EXISTS `ushouseholdincome_cleaned` (
          `row_id` int DEFAULT NULL,
          `id` int DEFAULT NULL,
          `State_Code` int DEFAULT NULL,
          `State_Name` text,
          `State_ab` text,
          `County` text,
          `City` text,
          `Place` text,
          `Type` text,
          `Primary` text,
          `Zip_Code` int DEFAULT NULL,
          `Area_Code` int DEFAULT NULL,
          `ALand` int DEFAULT NULL,
          `AWater` int DEFAULT NULL,
          `Lat` double DEFAULT NULL,
          `Lon` double DEFAULT NULL,
          `TimeStamp` TIMESTAMP DEFAULT NULL
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
# Copy data to new table
    INSERT INTO project.ushouseholdincome_cleaned
    SELECT *, CURRENT_TIMESTAMP
    FROM project.ushouseholdincome;
# Remove Duplicates
DELETE FROM project.ushouseholdincome_cleaned
WHERE
    row_id IN (
    SELECT row_id
FROM (
    SELECT row_id, id,
        ROW_NUMBER() OVER (
            PARTITION BY id, `TimeStamp`
            ORDER BY id, `TimeStamp`) AS row_num
    FROM
        project.ushouseholdincome_cleaned
) duplicates
WHERE
    row_num > 1
);
# Fixing some data quality issues by fixing typos and general standardization
    UPDATE project.ushouseholdincome_cleaned
    SET State_Name = 'Georgia'
    WHERE State_Name = 'georia';

    UPDATE project.ushouseholdincome_cleaned
    SET County = UPPER(County);

    UPDATE project.ushouseholdincome_cleaned
    SET City = UPPER(City);

    UPDATE project.ushouseholdincome_cleaned
    SET Place = UPPER(Place);

    UPDATE project.ushouseholdincome_cleaned
    SET State_Name = UPPER(State_Name);

    UPDATE project.ushouseholdincome_cleaned
    SET `Type` = 'CDP'
    WHERE `Type` = 'CPD';

    UPDATE project.ushouseholdincome_cleaned
    SET `Type` = 'Borough'
    WHERE `Type` = 'Boroughs';

END $$
DELIMITER ;

CALL Copy_and_Clean_Date();
```

To run this stored procedure in a schedule, an event is needed named run_data_cleaning. This event is scheduled to run every 30 days. When triggered, it executes the stored procedure Copy_and_Clean_Data(). The purpose of this event is likely to automate data cleaning tasks at regular intervals. It testing, I set the schedule every 2 mins.

```
# Create Event
DROP EVENT run_data_cleaning;
CREATE EVENT run_data_cleaning
    ON SCHEDULE EVERY 2 MINUTE
    DO CALL Copy_and_Clean_Data();
```

A trigger is created named Transfer_clean_date. It fires after an insert operation (i.e., when new data is added) on the specified table. When triggered, it calls the same stored procedure, Copy_and_Clean_Data(). Essentially, this trigger ensures that data cleaning occurs whenever new records are inserted into the ushouseholdincome table.

```
# CREATE TRIGGER
DELIMITER $$
CREATE TRIGGER Transfer_clean_date
    AFTER INSERT ON project.ushouseholdincome
    FOR EACH ROW
BEGIN
    CALL Copy_and_and_Clean_Date();
END $$
DELIMITER ;
```

The provided SQL codes establishes an event and a trigger to maintain data quality and consistency in the ushouseholdincome dataset.

Tools Used

MySQL was used for the queries and code.