

Training Guide

for   
PRE-MEDICARE VALUES

DATA MINING

Written By:

Swarna

1. Introduction

Although many people have the desire to retire early, there are additional complications involved in making this a reality. First of all, you will need to think about how to get health insurance in the meantime if you intend to retire before you become eligible for Medicare.

Experts advise setting aside more funds, or funds unrelated to a 401(k) account, as you get closer to early retirement to have on hand for unexpected medical expenses. (Experts usually advise maintaining an emergency fund with three to six months' worth of living expenses.)

A 401(k) plan is a company-sponsored retirement account to which employees can contribute income, while employers may match contributions.

Indeed, there are many factors to take into account when preparing for an early retirement, and pre-Medicare benefits are only one of them. A financial planner can assist you in developing a personal retirement savings plan that takes into account your expected spending, your saved amount, and your retirement goals. You should speak with a health insurance expert about your options for health insurance coverage, including short-term medical plans made specifically for early retirees.

Are you considering early retirement? While the prospect may seem daunting, it's essential to prepare for various aspects, including health insurance coverage. Even before you're eligible for Medicare, it's crucial to ensure you have adequate health insurance in place. Let's explore how understanding the different metal levels of Marketplace health insurance can be your first step towards securing suitable coverage for your early retirement years.

Thanks to the Affordable Care Act (ACA), Marketplace health insurance plans are categorized into Bronze, Silver, Gold, and Platinum metal tiers. Each tier offers different cost-sharing ratios between you and your insurance company, affecting your monthly premiums and out-of-pocket costs.

Before diving into the Marketplace, it's essential to understand which metal level aligns best with your early retirement needs. Here's a breakdown of the metal tiers:

**Bronze :**

Typically offers lower monthly premiums but higher deductibles. It may suit individuals with fewer anticipated medical expenses

**Silver :**

Strikes a balance between premiums and out-of-pocket costs. Suitable for those with moderate healthcare needs

**Gold :**

Comes with higher premiums but lower deductibles, making it ideal for individuals expecting higher medical expense

**Platinum :**

Offers the highest level of coverage with the highest premiums and lowest deductibles. Suitable for individuals with significant healthcare needs

Regardless of the metal tier you choose, all Marketplace plans must cover essential health benefits mandated by the ACA. These include ambulatory patient services, emergency care, hospitalization, maternity care, mental health services, prescription drugs, rehabilitation services, laboratory tests, preventive care, and pediatric services.

It is crucial to note that the metal tiers do not reflect differences in the quality of care received. Instead, they indicate how costs are distributed between you and your insurance provides.  
  
As you plan for early retirement, consulting with a health insurance expert can help you navigate the Marketplace and select the most suitable plan. Additionally, working with a financial planner can assist you in budgeting for healthcare expenses during your retirement years.  
  
In conclusion, understanding the metal levels of Marketplace health insurance is a crucial step towards ensuring comprehensive coverage during early retirement. Start exploring your options today to secure peace of mind for your future healthcare needs.

1. Methodology

**Postman for Data Extraction:**

Postman is a popular API development tool that allows users to send requests to APIs and receive responses.

In this context, Postman is being used to extract data from APIs by sending HTTP requests and capturing the responses.

Users can define request parameters, headers, and authentication methods within Postman to accurately retrieve the desired data from various APIs.

Postman provides features such as request history, collections, and environments, making it easy to organize and manage API requests.

**Dropbox for Data Collection:**

Dropbox is a cloud storage service that allows users to store and share files securely.

In the context of integrating with Postman, Dropbox can be used to establish a connection between Postman and Dropbox to collect the extracted data.

Users can configure Postman to automatically upload the extracted data to Dropbox after each request or at scheduled intervals.

This integration streamlines the data collection process and ensures that the extracted data is securely stored and accessible from anywhere.

**MySQL for Working with Raw Data:**

MySQL is a popular open-source relational database management system (RDBMS) that allows users to store, retrieve, and manipulate structured data.

After collecting raw data using Postman and storing it in Dropbox, MySQL can be used to further process and analyze the data.

Users can create database tables, import the raw data into MySQL, and perform various operations such as querying, filtering, and aggregating the data.

MySQL provides SQL (Structured Query Language) for interacting with the database, making it easy to perform complex data manipulation tasks.

**Excel with VB Script for Data Validation:**

Excel is a widely used spreadsheet application that allows users to organize, analyze, and visualize data.

In the context of data validation, users can leverage Excel's built-in features along with VB (Visual Basic) scripting capabilities to validate the collected data.

VB Script can be used to create custom validation rules and automate data validation tasks within Excel.

Users can define validation criteria, apply them to the collected data, and generate validation reports to identify any discrepancies or errors.

Excel's flexibility and VB Script's power enable users to ensure the accuracy and integrity of the collected data before further analysis or processing.

1. Postman Test Script

// Ensure the status code is 200

tests["Status code is 200"] **=** responseCode.code **===** 200;

// Parse the response body

var response **=** JSON.**parse**(responseBody);

// Log the entire response body for debugging

var recommendedPlans **=** response.recommendedPlans;

var numberOfPlans **=** response.numberOfPlans;

var planEnrollmentYear **=** response.planEnrollmentYear;

// Store plansCount in the environment variable

pm.environment.**set**("plansCount", numberOfPlans);

// Initialize the array to hold row data

var rowToAdd **=** [];

// Extract values outside recommendedPlans

var zipCode **=** response.UIPlanRequest.zip\_code;

var fipCode **=** response.UIPlanRequest.fip\_code;

var state **=** response.UIPlanRequest.state;

console.**log**(zipCode);

console.**log**(fipCode);

console.**log**(state);

// Loop through each plan in the array

**for** (var i **=** 0; i **<** numberOfPlans; i**++**) {

var currentPlan **=** recommendedPlans[i];

// Extracting values from the current plan

var healthcarePlanId **=** currentPlan.healthcarePlanId;

var carrierName **=** currentPlan.carrierName;

// Check and remove commas from carrierName

**if** (**typeof** carrierName **===** 'string' **&&** carrierName.includes(',')) {

carrierName **=** carrierName.**replace**(/,/**g**, '');

}

var healthcarePlanType **=** currentPlan.healthcarePlanType;

var healthcarePlanMetalLevel **=** currentPlan.healthcarePlanMetalLevel;

var hsaFsaFlag **=** currentPlan.hsaFsaFlag;

// var hsaFsaFlag = currentPlan.hsaFsaFlag === "None" ? false : true;

// Set to false if "None", true if "HSA"

var brokerOfReference **=** currentPlan.brokerOfReference;

**function** processValue(value) {

**if** (value **===** **null** **||** **typeof** value **!==** 'string') {

// Rule: If the value is null or not a string, return 0

**return** 0;

}

// Remove commas from the string

value **=** value.**replace**(/,/**g**, '');

// Check for different patterns

**if** (value.**toLowerCase**().includes('no charge')) {

// Rule: If the string contains "no charge", replace it with 0

**return** 0;

}

// Rule: Capture the value immediately after the first occurrence of the $ symbol and trim the rest

const matchResultDollar **=** value.**match**(/\$([0-9]**+**)/);

**if** (matchResultDollar) {

**return** **parseFloat**(matchResultDollar[1]);

}

// Rule: Capture the percentage value and trim the rest

const matchResultPercentage **=** value.**match**(/(\d**+**)% coinsurance/);

**if** (matchResultPercentage) {

**return** **parseFloat**(matchResultPercentage[1]);

}

// Rule: Capture the percentage value after deductible and trim the rest

const matchResultPercentageAfterDeductible **=** value.**match**(/(\d**+**)% after deductible/);

**if** (matchResultPercentageAfterDeductible) {

**return** **parseFloat**(matchResultPercentageAfterDeductible[1]);

}

// Rule: Capture the percentage value in the pattern "50% coinsurance after deductible" and trim the rest

const matchResult50PercentageAfterDeductible **=** value.**match**(/50% coinsurance after deductible/);

**if** (matchResult50PercentageAfterDeductible) {

**return** 50;

}

// Rule: Capture the percentage value in the pattern "60% coinsurance after deductible" and

//trim the rest

const matchResult60PercentageAfterDeductible **=** value.**match**(/60% coinsurance after deductible/);

**if** (matchResult60PercentageAfterDeductible) {

**return** 60;

}

// No specific rule matched, return 0

**return** 0;

}

var emergencyRoomVisit **=** processValue(currentPlan.emergencyRoomVisit);

var urgentCareVisit **=** processValue(currentPlan.urgentCareVisit);

var outOfNetworkCoinsurance **=** processValue(currentPlan.outOfNetworkCoinsurance);

var planLevelCoInsurance **=** processValue(currentPlan.planLevelCoInsurance);

var originalHealthcarePremium **=** currentPlan.originalHealthcarePremium;

var individualInNetworkDeductibleLimit **=** currentPlan.individualInNetworkDeductibleLimit;

var familyInNetworkDeductibleLimit **=** currentPlan.familyInNetworkDeductibleLimit;

var individualInNetworkOopLimit **=** currentPlan.individualInNetworkOopLimit;

var familyInNetworkOopLimit **=** currentPlan.familyInNetworkOopLimit;

var individualOutNetworkDeductibleLimit **=** currentPlan.individualOutNetworkDeductibleLimit;

var familyOutNetworkDeductibleLimit **=** currentPlan.familyOutNetworkDeductibleLimit;

var individualOutNetworkOopLimit **=** currentPlan.individualOutNetworkOopLimit;

var familyOutNetworkOopLimit **=** currentPlan.familyOutNetworkOopLimit;

// Create an array for the current plan's data

var planData **=** [

state,

zipCode,

fipCode,

numberOfPlans,

planEnrollmentYear,

healthcarePlanId,

carrierName,

healthcarePlanType,

healthcarePlanMetalLevel,

hsaFsaFlag,

brokerOfReference,

emergencyRoomVisit,

urgentCareVisit,

outOfNetworkCoinsurance,

planLevelCoInsurance,

originalHealthcarePremium,

individualInNetworkDeductibleLimit,

familyInNetworkDeductibleLimit,

individualInNetworkOopLimit,

familyInNetworkOopLimit,

individualOutNetworkDeductibleLimit,

familyOutNetworkDeductibleLimit,

individualOutNetworkOopLimit,

familyOutNetworkOopLimit

];

// Add the current plan's data to the rowToAdd array

rowToAdd.**push**(planData);

}

// Set the CSV content in the environment variable

pm.environment.**set**("csvContent", JSON.**stringify**(rowToAdd));

// Trigger the next request

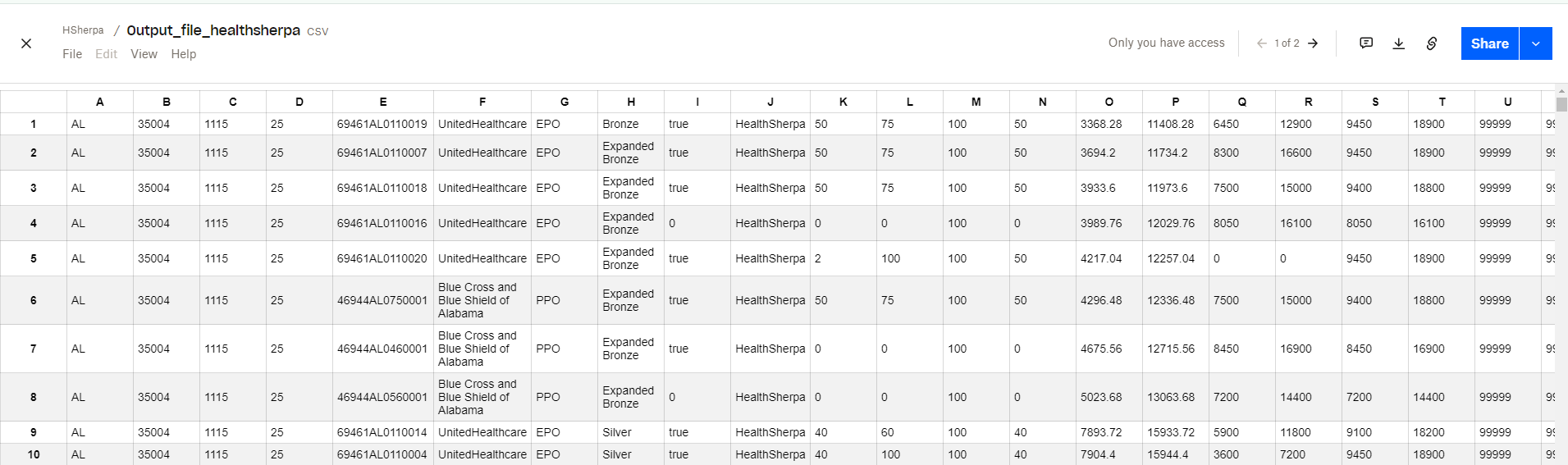
postman.setNextRequest("Custom\_CSV\_Content");

1. Data Collection

As part of the process to update the Finance application system, we initiated the data collection phase by obtaining necessary information from UI, a platform that provides healthcare enrollment solutions. Our goal was to improve the operation of the system by integrating pertinent data from UI.   
  
'zipcode.csv' was the name of the input file that needed to be created in order to begin the process of obtaining data from the UI API. Certain inputs were needed by the API, most notably zip codes and the names of the states they corresponded to. To fulfill this requirement, we leveraged the existing data stored in the SQL database.  
  
The dataset retrieved from the SQL database comprised a total of 3141 records, each containing a unique combination of zip codes and their respective state names. To ensure efficient processing and management of this substantial dataset, we opted to segment it into smaller, more manageable chunks based on state-wise divisions.

This segmentation process involved dividing the dataset into state-specific segments, with each segment containing approximately 100 to 120 records. By organizing the data in this manner, we aimed to streamline the subsequent steps of the data processing pipeline and facilitate smoother interactions.

Upon retrieving the data from UI via the API using Postman, the output was collected in Dropbox as CSV files. These CSV files served as the basis for further analysis and processing.

Subsequently, the CSV files were imported into MySQL for analysis and processing. Using MySQL, we conducted various operations such as data cleaning, filtering, and aggregation to extract actionable insights and prepare the data for integration into the Finance application system  
  


1. Sql

MySQL is a popular open-source relational database management system (RDBMS) that is widely used for building scalable and robust web-based applications. It is known for its reliability, performance, and ease of use. MySQL uses a structured query language (SQL) to manage and manipulate data stored in databases. SQL (Structured Query Language) is a standardized language used to interact with relational databases, allowing users to perform tasks such as querying data, updating records, and defining database structures.

SQLYog, on the other hand, is a graphical user interface (GUI) tool designed specifically for managing MySQL databases. It provides a user-friendly interface that allows developers and database administrators to interact with MySQL databases efficiently. SQLYog offers a range of features including database schema design, data modeling, query building, data synchronization, backup and restore, and server administration tasks. With SQLYog, users can easily perform common database operations without needing to write SQL queries manually. It simplifies database management tasks and enhances productivity for MySQL developers and administrators.

**MYSQL – Installation**

* <https://www.youtube.com/watch?v=YbOLy2LtsjM>

**SQLYOG – Installation**

* After Installation of MYSQL workbench we have to create an Local instance
* Remember the root password you will need for further to make connection between MYSQL workbench and SQLyog
* <https://webyog.com/product/sqlyog/>
* <https://www.youtube.com/watch?v=M-Dlly5v5lo>
* Click on File => newconnection( establishing connection between SQLYOG and Server)
* Provide the username and password for SERVER
* Click on Test Connection(if the test connect is successful) then click on Connect
* Click on File->newconnection(creating a local host )
* Name the Connection localhost123(You can give any name)
* Provide password as you gave it while downloading for MYSQl workbench

**Manage Your Database**

* Once connected, you'll see various options for managing your database:
  + **Object Explorer:** This panel displays the available databases, tables, views, etc. (circled in red below).
  + **Query Editor:** This area allows you to write and execute SQL queries (circled in blue below).
  + **Result Grid:** This section shows the results of your queries (circled in green below).
* **Browse Tables:** Expand the database in the Object Explorer and double-click a table to view its data.
* **Write and Execute Queries:** Use the Query Editor to write SQL statements. Click the "Execute" button (circled in red below) or press F9 to run the query and see the results in the Result Grid.

**Additional Features**

SQLyog offers various advanced features, including:

* **Visual Query Builder:** Helps build complex queries without writing code.
* **Data Import/Export:** Allows importing and exporting data from various formats.
* **User Management:** Create and manage database users and permissions.
* **Backup and Restore:** Backup and restore your databases for safekeeping.

**Note:** These are just basic steps. Explore the different menus and options within SQLyog to discover its full functionality.

**Remember:** Always refer to the official SQLyog documentation for detailed information and advanced features.

By following these steps and exploring the features, you can effectively use SQLyog to manage your MySQL databases

To establish the connection of the local instance into the Mysql => click on to the + button next to MYSQL Connection

Give the name of the local Instance created in the SQLYOG to get the local instance connected to the MYSQL

1. Sql queries

**Step 1:**

This step involves creating a table named combined\_rawdata to store raw data obtained from Postman sequencing.

The data from various state-specific tables (al\_ak\_rawdata, az\_ar\_rawdata, etc.) is combined into this table using the UNION ALL operator.

CREATE TABLE database\_UI.combined\_rawdata (

state VARCHAR(255),

zipCode VARCHAR(255),

fipCode VARCHAR(255),

numberOfPlans INT,

planEnrollmentYear INT,

healthcarePlanId VARCHAR(255),

carrierName VARCHAR(255),

healthcarePlanType VARCHAR(255),

healthcarePlanMetalLevel VARCHAR(255),

hsaFsaFlag VARCHAR(255),

brokerOfReference VARCHAR(255),

emergencyRoomVisit DECIMAL(10, 2),

urgentCareVisit DECIMAL(10, 2),

outOfNetworkCoinsurance DECIMAL(10, 2),

planLevelCoInsurance DECIMAL(10, 2),

originalHealthcarePremium DECIMAL(10, 2),

individualInNetworkDeductibleLimit DECIMAL(10, 2),

familyInNetworkDeductibleLimit DECIMAL(10, 2),

individualInNetworkOopLimit DECIMAL(10, 2),

familyInNetworkOopLimit DECIMAL(10, 2),

individualOutNetworkDeductibleLimit DECIMAL(10, 2),

familyOutNetworkDeductibleLimit DECIMAL(10, 2),

individualOutNetworkOopLimit DECIMAL(10, 2),

familyOutNetworkOopLimit DECIMAL(10, 2));

* INSERT INTO database\_UI.combined\_rawdata

SELECT \* FROM(

SELECT \* FROM database\_UI.al\_ak\_rawdata

UNION ALL

SELECT \* FROM database\_UI.az\_ar\_rawdata

UNION ALL

SELECT \* FROM database\_UI.ca\_co\_ct\_rawdata

UNION ALL

SELECT \* FROM database\_UI.de\_dc\_fl\_rawdata

UNION ALL

SELECT \* FROM database\_UI.ga\_rawdata

UNION ALL

SELECT \* FROM database\_UI.hi\_id\_il\_ia\_ks\_ky\_rawdata

UNION ALL

SELECT \* FROM database\_UI.in\_rawdata

UNION ALL

SELECT \* FROM database\_UI.la\_rawdata

UNION ALL

SELECT \* FROM database\_UI.mi\_rawdata

UNION ALL

SELECT \* FROM database\_UI.me\_md\_ma\_rawdata

UNION ALL

SELECT \* FROM database\_UI.mn\_rawdata

UNION ALL

SELECT \* FROM database\_UI.mo\_rawdata

UNION ALL

SELECT \* FROM database\_UI.ms\_rawdata

UNION ALL

SELECT \* FROM database\_UI.mt\_rawdata

UNION ALL

SELECT \* FROM database\_UI.nc\_rawdata

UNION ALL

SELECT \* FROM database\_UI.nd\_rawdata

UNION ALL

SELECT \* FROM database\_UI.ne\_rawdata

UNION ALL

SELECT \* FROM database\_UI.nh\_rawdata

UNION ALL

SELECT \* FROM database\_UI.nv\_rawdata

UNION ALL

SELECT \* FROM database\_UI.ny\_rawdata

UNION ALL

SELECT \* FROM database\_UI.oh\_rawdata

UNION ALL

SELECT \* FROM database\_UI.ok\_rawdata

UNION ALL

SELECT \* FROM database\_UI.or\_rawdata

UNION ALL

SELECT \* FROM database\_UI.pa\_rawdata

UNION ALL

SELECT \* FROM database\_UI.ri\_rawdata

UNION ALL

SELECT \* FROM database\_UI.sc\_rawdata

UNION ALL

SELECT \* FROM database\_UI.sd\_rawdata

UNION ALL

SELECT \* FROM database\_UI.tn\_rawdata

UNION ALL

SELECT \* FROM database\_UI.tx\_ut\_vt\_va\_wa\_wv\_wi\_wy\_rawdata

UNION ALL

SELECT \* FROM database\_UI.nj\_rawdata

UNION ALL

SELECT \* FROM database\_UI.nm\_rawdata) AS combined\_data;

select count(\*) from UI.combined\_rawdata; /\*154190\*/

**Step 2:**

This step creates a new table named processed\_data\_new1 by applying a formula to calculate the average of certain values from the combined\_rawdata table.

It rounds the averages to two decimal places for better presentation.

CREATE TABLE UI.processed\_data\_new1 AS

SELECT

state, zipCode, fipCode, planEnrollmentYear,

CONCAT(planEnrollmentYear, state, fipCode, healthcarePlanMetalLevel, healthcarePlanType, brokerOfReference) AS planId,

healthcarePlanMetalLevel, healthcarePlanType, brokerOfReference,

ROUND(AVG(emergencyRoomVisit), 2) AS avg\_emergencyRoomVisit,

ROUND(AVG(urgentCareVisit), 2) AS avg\_urgentCareVisit,

ROUND(AVG(outOfNetworkCoinsurance), 2) AS avg\_outOfNetworkCoinsurance,

ROUND(AVG(planLevelCoInsurance), 2) AS avg\_planLevelCoInsurance,

ROUND(AVG(originalHealthcarePremium), 2) AS avg\_originalHealthcarePremium,

ROUND(AVG(individualInNetworkDeductibleLimit), 2) AS avg\_individualInNetworkDeductibleLimit,

ROUND(AVG(familyInNetworkDeductibleLimit), 2) AS avg\_familyInNetworkDeductibleLimit,

ROUND(AVG(individualInNetworkOopLimit), 2) AS avg\_individualInNetworkOopLimit,

ROUND(AVG(familyInNetworkOopLimit), 2) AS avg\_familyInNetworkOopLimit,

ROUND(AVG(individualOutNetworkDeductibleLimit), 2) AS avg\_individualOutNetworkDeductibleLimit,

ROUND(AVG(familyOutNetworkDeductibleLimit), 2) AS avg\_familyOutNetworkDeductibleLimit,

ROUND(AVG(individualOutNetworkOopLimit), 2) AS avg\_individualOutNetworkOopLimit,

ROUND(AVG(familyOutNetworkOopLimit), 2) AS avg\_familyOutNetworkOopLimit

FROM

UI.combined\_rawdata

GROUP BY

state, zipCode, fipCode, planEnrollmentYear, brokerOfReference, healthcarePlanMetalLevel, healthcarePlanType

order by state,zipcode,fipCode,healthcarePlanMetalLevel, healthcarePlanType;

select count(\*) from UI. processed\_data\_new1;

/\*21778\*/

**Step 3:**

These queries check the count of distinct zip codes for each state in the input\_file, combined\_rawdata, and processed\_data\_new1 tables to ensure data consistency.

Additionally, there are SQL commands to resolve discrepancies in the data, such as deleting duplicate entries or updating incorrect values.  
  
select count(distinct state,zipCode,fipCode ) from UI.input\_file where state ='Alabama' ;

/\*67\*/

select count(distinct zipCode) from UI.combined\_rawdata where state ='Alabama';

/\*67\*/

select count(distinct zipcode ) from UI.processed\_data\_new1 where state = 'Alabama';

/\*67\*/

select count(distinct zipCode ) from UI.input\_file where state ='Alaska' ;

select count(distinct zipCode) from UI.combined\_rawdata where state ='Alaska';

select count(distinct zipcode ) from UI.processed\_data\_new1 where state = 'Alaska';

/\*29\*/

select count(distinct zipCode ) from UI.input\_file where state ='Arizona' ;

select count(distinct zipCode) from UI.combined\_rawdata where state ='Arizona';

select count(distinct zipcode ) from UI.processed\_data\_new1 where state = 'Arizona';

/\*15\*/

**Step 4:**

This step involves creating another table named validation\_report\_new1 to summarize data from the combined\_rawdata table for further validation.

It calculates the sum of certain columns and the count of distinct healthcarePlanId values for each state, zip code, and plan enrollment year.

CREATE TABLE UI.validation\_report\_new1 AS

SELECT

state,zipCode,fipCode,planEnrollmentYear,

CONCAT(planEnrollmentYear, state, fipCode, healthcarePlanMetalLevel, healthcarePlanType, brokerOfReference) AS planId,

healthcarePlanMetalLevel,

COUNT(DISTINCT healthcarePlanId) AS healthcarePlanIdCount,

healthcarePlanType,

sum(originalHealthcarePremium) AS sum\_originalHealthcarePremium\_rawdata,

sum(individualInNetworkDeductibleLimit) AS sum\_individualInNetworkDeductibleLimit\_rawdata,

sum(familyInNetworkDeductibleLimit) AS sum\_familyInNetworkDeductibleLimit\_rawdata,

sum(individualInNetworkOopLimit) AS sum\_individualInNetworkOopLimit\_rawdata,

ROUND(AVG(originalHealthcarePremium), 2) AS avg\_originalHealthcarePremium,

ROUND(AVG(individualInNetworkDeductibleLimit), 2) AS avg\_individualInNetworkDeductibleLimit,

ROUND(AVG(familyInNetworkDeductibleLimit), 2) AS avg\_familyInNetworkDeductibleLimit,

ROUND(AVG(individualInNetworkOopLimit), 2) AS avg\_individualInNetworkOopLimit,

ROUND(AVG(familyInNetworkOopLimit), 2) AS avg\_familyInNetworkOopLimit

FROM

UI.combined\_rawdata

GROUP BY

state, zipCode, fipCode, planEnrollmentYear, brokerOfReference, healthcarePlanMetalLevel, healthcarePlanType

order by state,zipcode,fipcode,healthcarePlanMetalLevel, healthcarePlanType;

1. Data validation

After creating the validation\_report\_new1 table, data is exported to a local disk and added to an existing Excel file (validation\_report.xlsx) for additional information.

Macros and VBA scripts are used to round up values and perform comparison and color-coding for validation.

Sub RoundColumnsValues()

Dim lastRow As Long

Dim i As Long

' Find the last row with data in column I

lastRow = Cells(Rows.Count, "I").End(xlUp).Row

' Loop through each row from 2 to the last row for column I

For i = 2 To lastRow

' Round the value in column I to the nearest whole number

Cells(i, "I").Value = Round(Cells(i, "I").Value, 0)

Next i

' Find the last row with data in column L

lastRow = Cells(Rows.Count, "L").End(xlUp).Row

' Loop through each row from 2 to the last row for column L

For i = 2 To lastRow

' Round the value in column L to the nearest whole number

Cells(i, "L").Value = Round(Cells(i, "L").Value, 0)

Next i

' Repeat the above process for columns O, R, and U

' Find the last row with data in column O

lastRow = Cells(Rows.Count, "O").End(xlUp).Row

' Loop through each row from 2 to the last row for column O

For i = 2 To lastRow

' Round the value in column O to the nearest whole number

Cells(i, "O").Value = Round(Cells(i, "O").Value, 0)

Next i

' Find the last row with data in column R

lastRow = Cells(Rows.Count, "R").End(xlUp).Row

' Loop through each row from 2 to the last row for column R

For i = 2 To lastRow

' Round the value in column R to the nearest whole number

Cells(i, "R").Value = Round(Cells(i, "R").Value, 0)

Next i

' Find the last row with data in column U

lastRow = Cells(Rows.Count, "U").End(xlUp).Row

' Loop through each row from 2 to the last row for column U

For i = 2 To lastRow

' Round the value in column U to the nearest whole number

Cells(i, "U").Value = Round(Cells(i, "U").Value, 0)

Next i

End Sub

