**Analyzing United States Household Income Data**

**Northwest Missouri State University**

**Maryville, MO**

**Section: 44517-01**

**Team Name: Walky Talkies**

**Team Members:**

MANOJ KUMAR GUDE (S546981)

CHENNU VENKATA SHASHANKAR (S546962)

NAVEEN KUMAR GUNDU (S546960)

ANUSHA POREDDY (S546655)

REETHU GOPIREDDY (S546954)

**Milestone 3:**

**Explanation for Implementation of Steps**

**Introduction to Dataset:**

The United States Household Income dataset contains data on the household income in every state and country collected form Kaggle. This data will be analyzed to identify patterns and insights that can help united states financial department to identify the households which have lowest income, and this analysis is also helpful for real estate and business investment research using PySpark and visualized using Tableau. In project by the visualization identifying the most vital income elements when determining both quality and quantity and socioeconomic features of a given geographic location.

**Metadata:**

The dataset includes the following variables:

Id - Type: Character, Description: The Id of the location of which you are analyzing.

State\_Code - Type: Character, Description: The state code reported by the U.S. Census Bureau for the specified geographic location.

State\_Name - Type: Character, Description: The state name reported by the U.S. Census Bureau for the specified geographic location.

State\_ab - Type: Character, Description: The abbreviated state name reported by the U.S. Census Bureau for the specified geographic location.

County - Type: Character, Description: The county name reported by the U.S. Census Bureau for the specified geographic location.

City - Type: Character, Description: The city name reported by the U.S. Census Bureau for the specified geographic location.

Place - Type: Character, Description: The place name reported by the U.S. Census Bureau for the specified geographic location.

Type - Type: Character, Description: The place Type reported by the U.S. Census Bureau for the specified geographic location.

Primary - Type: Character, Description: Defines whether the location is a geographic location or a track and block group.

Zip Code - Type: Character, Description: The zip code reported by the U.S. Census Bureau of the closest geographic location with a zip code.

Area Code - Type: Character, Description: The zip code reported by the U.S. Census Bureau of the closest geographic location with a zip code.

ALand - Type: Double, Description: The Square area of land at the geographic or track location.

AWater - Type: Double, Description: The Square area of water at the geographic or track location.

Lat - Type: Double, Description: The mean household income of the specified geographic location.

Lon - Type: Double, Description: The standard deviation of the household income for the specified geographic location.

Mean - Type: Double, Description: The mean household income of the specified geographic location.

Median - Type: Double, Description: The median household income of the specified geographic location.

Stdev - Type: Double, Description: The standard deviation of the household income for the specified geographic location.

Households - Type: Double, Description: The number of households used in the statistical calculations.

**Importing and Preprocessing the Data:**

1. Load the CSV file from the Kaggle.

2. Extract the csv file into the same directory as Jupyter Notebook.

3. Import the Spark Session.

from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()

4. Then, import the data into a PySpark dataframe and preprocess it to ensure that it is in a suitable format for analysis.

Household = spark.read.format('csv').option('header','true').load('filepath')

household.createOrReplaceTempView('ushousehold')

household.show()

display(household)

5. Dropping the duplicates in the dataframe.

if household.count() > household.dropDuplicates(["id"]).count():

raise ValueError('Data has duplicates')

6. Changing the datatypes into useable format.

from pyspark.sql.types import IntegerType,BooleanType,DateType

household = household.withColumn("id",household.id.cast(IntegerType()))

household = household.withColumn("State\_Code",household.State\_Code.cast(IntegerType()))

household = household.withColumn("ALand",household.ALand.cast(IntegerType()))

household = household.withColumn("AWater",household.AWater.cast(IntegerType()))

household = household.withColumn("Lat",household.Lat.cast(BooleanType()))

household = household.withColumn("Lon",household.Lon.cast(BooleanType()))

household = household.withColumn("Mean",household.Mean.cast(IntegerType()))

household = household.withColumn("Median",household.Median.cast(IntegerType()))

household = household.withColumn("Stdev",household.Stdev.cast(IntegerType()))

household = household.withColumn("sum\_w",household.sum\_w.cast(BooleanType()))

7. Analyze the goals for the Household Income in United States in Jupyter Notebook and visualize it in Tableau.

**Goal 1:**

query:spark.sql('select County ,City from ushousehold where mean <= 100 and County = "Apache County" ').show()

spark.sql('select County ,City from ushousehold where mean <= 100').show()

Explanation: The SQL queries provided aim to discover counties and cities in a dataset called "ushousehold" with a mean household income less than or equal to 100. The first query specifically filters for Apache County, which is a county in Arizona, USA. This query looks for Apache County counties and cities where the average family income is less than or equal to $100. The second query searches the full dataset for counties and cities with mean household incomes less than or equal to 100. This query will return a list of all counties and cities in the entire dataset that fulfill the condition of having a mean household income of less than or equal to a certain amount.

Map

Description automatically generated

Goal 2:

query: spark.sql('select avg(ALand), avg(AWater) from ushousehold').show()

Explanation: This query pulls the "ALand" and "AWater" columns from the "ushousehold" table and then employs the AVG() function to compute the average value for each column across all states in the dataset. The AVG() function computes the mean of a collection of numbers. The query returns a table with two columns: the average square area of land for all states, and the average square area of water for all states. The SHOW() function is used to display the query results. The output will display the average square area of land and water for each state in the dataset.

Chart, treemap chart

Description automatically generated

Goal 3:

query1:spark.sql('select City,Mean from ushousehold order by Mean Desc LIMIT 5').show()

Explanation: These SQL queries pull the top five and bottom five household incomes from the "ushousehold" dataset. Assuming the table has columns named "City" and "Mean" that represent the city name and the mean household income for each city, the queries work as follows: This query pulls the "City" and "Mean" columns from the "ushousehold" dataset and ranks the results descendingly by the "Mean" column, which reflects the average household income. The LIMIT 5 clause restricts the results to the top five cities with the greatest average household income. The query returns a table with the top 5 cities in each state with the highest mean family income.

query2:spark.sql('select City,Mean from ushousehold order by Mean ASC LIMIT 5').show()

Explanation: This query pulls the "City" and "Mean" columns from the "ushousehold" dataset and ranks the results ascending by the "Mean" column, which

reflects the average household income. The LIMIT 5 clause restricts the results to the top five cities with the lowest average family incomes. The query returns a table with the top five cities in each state with the lowest mean household earnings. The SHOW() function is used to display the query results. The result will be a table with the columns "City" and "Mean" and the accompanying rows that fulfill the provided criteria.

Chart, bar chart

Description automatically generated

Goal4:

query1: spark.sql('select Zip\_Code, Area\_Code from ushousehold order by Mean DESC LIMIT 5').show()

Explanation: This query pulls the "Zip\_Code" and "Area\_Code" columns from the "ushousehold" dataset and ranks the results descendingly by the "Mean" column, which indicates the average household income. The LIMIT 5 clause restricts the results to the top five zip codes with the highest average household incomes. The query yields a table containing the top five zip codes with the highest mean household earnings.

query2: spark.sql('select Zip\_Code, Area\_Code from ushousehold order by Median DESC LIMIT 5').show()

Explanation: This query pulls the "Zip\_Code" and "Area\_Code" columns from the "ushousehold" table and sorts the results by the "Median" column, which reflects the median household income. The LIMIT 5 clause restricts the results to the top 5 zip codes in terms of median household income. The query yields a table containing the top five zip codes with the highest median household incomes.The SHOW() function is used to display the query results. The result will include a table with the "Zip\_Code" and "Area\_Code" columns, as well as the rows that match the supplied criteria.

Chart, pie chart

Description automatically generated

Goal5:

query: spark.sql('select DISTINCT ALAND, AWATER from ushousehold where Type = "City"').show()

Explanation: This SQL query retrieves the unique values of the "ALAND" and "AWATER" columns from the "ushousehold" table where the "Type" column equals "City". Assuming that the "ALAND" and "AWATER" columns represent a city's land and water areas, respectively, and that the "Type" column denotes the type of area, the following query works: The DISTINCT keyword is used to pick only unique "ALAND" and "AWATER" combinations from the "ushousehold" table. Only entries with the "Type" column equal to "City" are filtered using the WHERE clause. This ensures that only land and water areas for cities are returned, rather than other types of locations (such as counties, towns, and so on). The query yields a table containing the unique combinations.

Graphical user interface, table

Description automatically generated

Goal6:

query: spark.sql('select City, Place from ushousehold order by mean DESC').show()

Explanation: This SQL query selects the "City" and "Place" columns from the "ushousehold" table, and then sorts the results in descending order by the "mean" column. Assuming that the "City" column represents the city name and the "Place" column represents the state or county name, this query works as follows: The SELECT statement selects the "City" and "Place" columns from the "ushousehold" table. The ORDER BY clause orders the results in descending order by the "mean" column, which represents the mean household income for each city or area. This means that the cities with the highest mean household income will appear first in the results.The query returns a table with the "City" and "Place" columns, and the corresponding rows sorted in descending order by the mean household income.The SHOW() function is used to display the results of the query. The output will show a table with the "City" and "Place" columns, and the corresponding rows that meet the specified criteria, sorted by the mean household income in descending order.

Chart, bubble chart

Description automatically generated

Goal7:

household = household.withColumn("ALand",household.ALand.cast(BooleanType()))

household = household.withColumn("AWater",household.AWater.cast(BooleanType()))

spark.sql('SELECT SUM(ALand) AS TotalLandArea, SUM(AWater) AS TotalWaterArea from ushousehold where State\_ab = "AL"').show()

query: This code snippet first converts the "ALand" and "AWater" columns of the "household" dataframe from whatever data type they are to BooleanType using the cast() function. This means that the values in these columns will be converted to either True or False depending on whether or not they are considered "truthy" in Python. For example, a non-zero number or a non-empty string will be considered "truthy" and will be converted to True. The SQL query SELECT SUM(ALand) AS TotalLandArea, SUM(AWater) AS TotalWaterArea from ushousehold where State\_ab = "AL" selects the sum of the "ALand" and "AWater" columns from the "ushousehold" table for the state of Alabama ("State\_ab" column is equal to "AL"). The SUM() function is an aggregate function that calculates the sum of all the values in a given column. The AS keyword is used to rename the resulting columns to "TotalLandArea" and "TotalWaterArea" respectively. The WHERE clause is used to filter the results to only include rows where the "State\_ab" column is equal to "AL", which represents the state of Alabama. The query returns a table with the sum of the "ALand" and "AWater" columns for the state of Alabama, under the columns "TotalLandArea" and "TotalWaterArea" respectively. Finally, the SHOW() function is used to display the results of the query. The output will show a table with the "TotalLandArea" and "TotalWaterArea" columns, and the corresponding values that meet the specified criteria.

Text

Description automatically generated

Goal8:

query: spark.sql('select county , AVG(Stdev) AS AvgStdev from ushousehold where State\_ab = "MO" GROUP BY County').show()

Explanation: The SELECT statement selects the "county" column and the average standard deviation of household incomes for each county. The AVG() function is an aggregate function that calculates the average of all the values in a given column. The AS keyword is used to rename the resulting column to "AvgStdev". The WHERE clause is used to filter the results to only include rows where the "State\_ab" column is equal to "MO", which represents the state of Missouri. The GROUP BY clause groups the results by county, which means that the average standard deviation will be calculated for each unique county in Missouri. The query returns a table with the "county" and "AvgStdev" columns, and the corresponding values that meet the specified criteria. Finally, the SHOW() function is used to display the results of the query. The output will show a table with the "county" and "AvgStdev" columns, and the corresponding values that meet the specified criteria, grouped by county.

Map

Description automatically generated

Goal9:

query:spark.sql('SELECT city, county, SUM(ALand) AS total\_land\_area FROM ushousehold WHERE median BETWEEN 40000 AND 60000 GROUP BY city, county').show()

Explanation: This SQL query selects the "city" and "county" columns and calculates the total land area ("ALand") for each city and county combination where the median household income ("median") is between 40000 and 60000. The SELECT statement selects the "city" and "county" columns and calculates the sum of the "ALand" column for each unique combination of "city" and "county". The AS keyword is used to rename the resulting column to "total\_land\_area". The WHERE clause is used to filter the results to only include rows where the "median" column is between 40000 and 60000, which represents the range of median household incomes. The GROUP BY clause groups the results by both "city" and "county", which means that the total land area will be calculated for each unique combination of city and county where the median household income falls within the specified range. The query returns a table with the "city", "county", and "total\_land\_area" columns, and the corresponding values that meet the specified criteria. Finally, the SHOW() function is used to display the results of the query. The output will show a table with the "city", "county", and "total\_land\_area" columns, and the corresponding values that meet the specified criteria, grouped by city and county.

Graphical user interface, text, application, Word

Description automatically generated

Goal10:

query: spark.sql('SELECT state\_name, AVG(median) AS avg\_median\_income FROM ushousehold GROUP BY state\_name ORDER BY avg\_median\_income DESC').show()

Explanation: This SQL query selects the "state\_name" column and calculates the average of the "median" column for each unique state in the dataset. The resulting table is then sorted in descending order based on the average median income. The SELECT statement selects the "state\_name" column and calculates the average of the "median" column for each unique state in the dataset. The AS keyword is used to rename the resulting column to "avg\_median\_income". The GROUP BY clause groups the results by "state\_name", which means that the average median income will be calculated for each unique state. The ORDER BY clause is used to sort the results by the "avg\_median\_income" column in descending order, which means that the states with the highest average median income will be listed first. The query returns a table with the "state\_name" and "avg\_median\_income" columns, and the corresponding values for each state in the dataset. Finally, the SHOW() function is used to display the results of the query. The output will show a table with the "state\_name" and "avg\_median\_income" columns, and the corresponding values for each state, sorted in descending order by average median income.

Text

Description automatically generated with low confidence

**Discussions around Relevant Metrics**

**Data Quality check on the Dataset**

The United States Household Income dataset is a widely used dataset on the household income for every state and county. However, like any other data set, it is important to evaluate its quality before using it for analysis.

Here are some aspects of data quality for this dataset:

**Completeness:** The dataset contains household income in specified locations. However, there are some missing values in the dataset, which may impact the analysis. It is important to assess the extent of the missing data and decide on the best way to handle it.

**Accuracy:** The accuracy of the dataset depends on the accuracy of the household income used to collect the data. The dataset includes calculations of mean, median and standard deviation, which should be accurate for reliable analysis.

**Consistency:** The dataset is consistent as the calculations are based on specified location and calculated the mean, median and standard deviation. But when it comes to household the dataset may face an inconsistency issue.

**Relevance:** The dataset should be relevant to the research question or problem being addressed. The dataset includes calculations of mean, median and standard deviation of houdehold income in United States, which may not be representative of other households or other countries. Therefore, it is important to assess the relevance of the dataset to the specific research question being addressed.

Overall, the United States Household Income dataset is a high-quality dataset for income of every state and county at the household level. However, as with any dataset, it is important to carefully evaluate its quality before using it for analysis.

**The 5 Vs for Household Income in United States**

**1. Volume:** The dataset contains a large volume of data, with over 32,000 of different cities in different counties in United States.

**2. Velocity:** The dataset was recorded at a high rate, with the mean and median household income in defined places calculated. This fast rate of data acquisition can make processing and evaluating the data difficult.

**3. Variety:** The dataset comprises a wide range of data types, including mean, median, and standard deviation estimates of type double and Strings for states, cities, and counties**.**

**4. Veracity:** Errors or irregularities in the data gathering process, as well as missing data points, may have an impact on the dataset's accuracy and reliability. Before using a dataset for analysis, it is critical to assess its authenticity and rectify any potential flaws.

**5. Value:** The dataset has significant value for studying mean, median and standard deviation of specified locations.

**Latency and Processing time**

The dataset's vast volume of data, with over 32,000 specified localities in the United States, can create processing and analysis issues.

Processing the dataset may necessitate large computational resources, and the processing time will be determined by the nature of the research. Running machine learning algorithms on the dataset, for example, may take significantly more processing time than performing basic descriptive statistics.

Pre-processing the dataset, such as aggregating the data to a lower time resolution or deleting unnecessary data points, may be necessary to reduce latency and processing time. Dacompression techniques may also be beneficial for lowering dataset size and improving processing time.

Overall, latency and processing time are important considerations when working with the United States Household Income dataset, and the specific approach to reducing latency and processing time will depend on the specific research question or problem being addressed.

**Resource Utilization, Security, and Cost**

**Resource Utilization:** The data set is analyzed using Python, Tableau, and Excel. Furthermore, there are numerous libraries and packages available for these tools that are specifically designed for time series analysis, machine learning, and data visualization.

**Security:** It is critical that any study or research undertaken using this dataset protects the privacy of the households. This can be accomplished by eliminating any identifiable information from the dataset.

**Cost:** The dataset is freely available and can be downloaded from the Kaggle website. However, depending on the size of the dataset and the complexity of the analysis, there may be some costs associated with processing and storing the data.

In conclusion, our project is to analyze the goals in PySpark to examine household income on specified locations. In order to find patterns and trends in the data, we collected data from Kaggle, cleaned and processed it, and then performed SQL queries. Our investigation showed that, depending on the specified locations, the mean, median and standard deviation is calculated. Additionally, calculations are done using the area of land and area of water in that geographic location. Overall, our initiative emphasizes the household income based on the cities in different states. We were able to process and analyze enormous amounts of data effectively and efficiently by utilizing PySpark and visualized in Tableau.