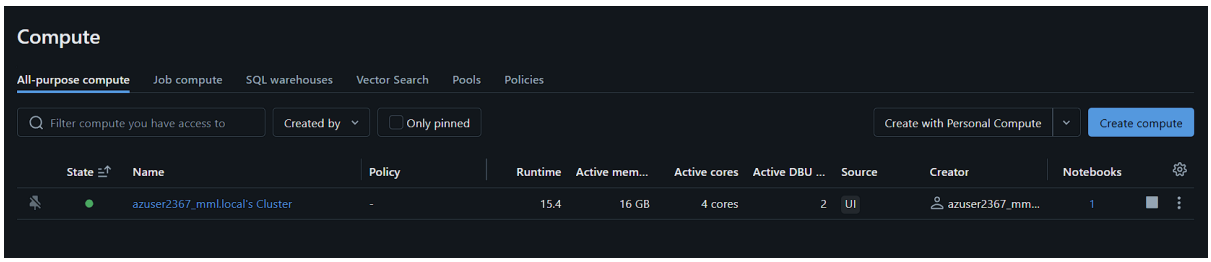
Azure Databricks Coding Challenge

J Jatin

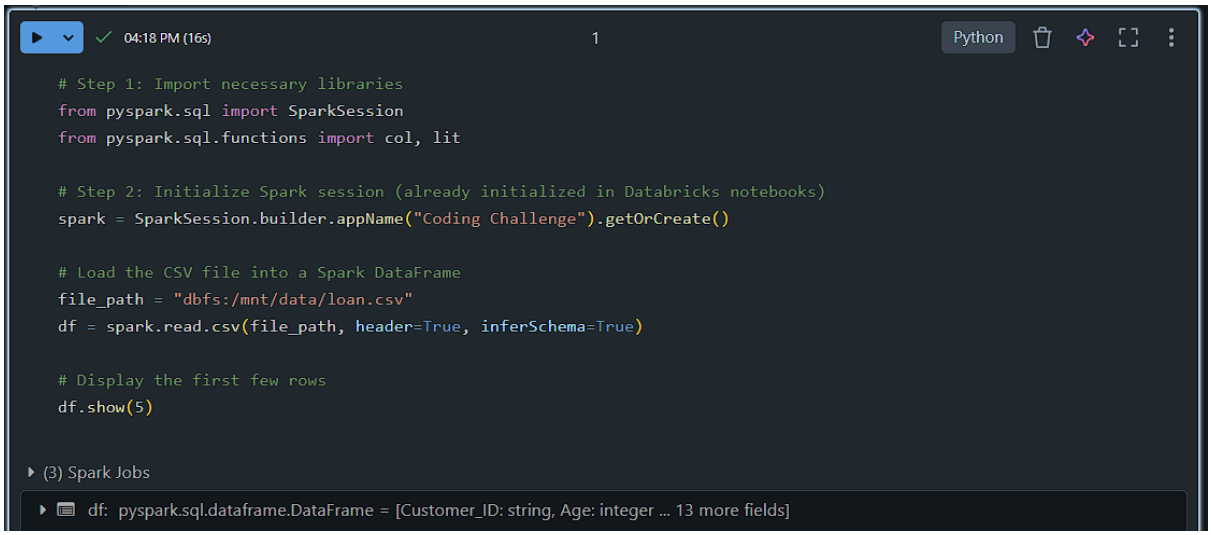
Data-Engineering

Create a cluster &attach the notebook to the cluster and run all commands in the notebook & creates a DataFrame from a Databricks dataset & Create a Visualizations in Databricks notebooks.

1) Creating Cluster in Databricks:



2) Loading Dataset for Transformations to be Performed:



The above code demonstrates how to load and inspect a dataset using PySpark in Databricks. First, the necessary libraries are imported, including SparkSession for managing Spark functionality and col and lit for DataFrame transformations. A Spark session is then initialized with the name "Coding Challenge" using SparkSession.builder.appName("Coding Challenge").getOrCreate(), which creates or retrieves an existing Spark session (though this is pre-initialized in Databricks). The dataset, located at "dbfs:/mnt/data/loan.csv", is loaded into a Spark DataFrame using

spark.read.csv(), with header=True to use the first row as column names and inferSchema=True to automatically determine column data types. Finally, the df.show(5) command displays the first five rows of the dataset, allowing for quick verification of the data's structure and content. This setup forms the foundation for further ETL (Extract, Transform, Load) processes, such as data cleaning, transformations, and visualization.

```
df: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+---------
--+-------+-----------+---------------+-----------------+
|Customer_ID|Age|Gender|  Occupation|Marital Status|Family Size|Income|Expenditure|Use Frequency|Loan Category|Loan Amou
nt|Overdue|Debt Record|Returned Cheque|Dishonour of Bill|
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+---------
--+-------+-----------+---------------+-----------------+
|    IB14001| 30|  MALE|BANK MANAGER|        SINGLE|          4| 50000|      22199|            6|      HOUSING|  10,00,0
00|      5|     42,898|              6|                9|
|    IB14008| 44|  MALE|   PROFESSOR|       MARRIED|          6| 51000|      19999|            4|     SHOPPING|     50,0
00|      3|     33,999|              1|                5|
|    IB14012| 30|FEMALE|     DENTIST|        SINGLE|          3| 58450|      27675|            5|   TRAVELLING|     75,0
00|      6|     20,876|              3|                1|
|    IB14018| 29|  MALE|     TEACHER|       MARRIED|          5| 45767|      12787|            3|    GOLD LOAN|   6,00,0
00|      7|     11,000|              0|                4|
|    IB14022| 34|  MALE|      POLICE|        SINGLE|          4| 43521|      11999|            3|   AUTOMOBILE|   2,00,0
00|      2|     43,898|              1|                2|
+-----------+---+------+------------+--------------+-----------+------+-----------+-------------+-------------+---------
--+-------+-----------+---------------+-----------------+
only showing top 5 rows
```

3) Exploratory Data Analysis:

```
04:24 PM (3s)                                   2                                   Python

# Show the schema
df.printSchema()

# Count the rows
print(f"Total records: {df.count()}")

# Display summary statistics
df.describe().show()

# Show column names
print("Columns:", df.columns)

(4) Spark Jobs
```

```
root
 |-- Customer_ID: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Occupation: string (nullable = true)
 |-- Marital Status: string (nullable = true)
 |-- Family Size: integer (nullable = true)
 |-- Income: integer (nullable = true)
 |-- Expenditure: integer (nullable = true)
 |-- Use Frequency: integer (nullable = true)
 |-- Loan Category: string (nullable = true)
 |-- Loan Amount: string (nullable = true)
 |-- Overdue: integer (nullable = true)
 |-- Debt Record: string (nullable = true)
 |-- Returned Cheque: integer (nullable = true)
 |-- Dishonour of Bill: integer (nullable = true)
```

```
Total records: 500
```

```
|summary|Customer_ID|              Age|Gender|     Occupation|Marital Status|       Family Size|           Income|
Expenditure|    Use Frequency|Loan Category|Loan Amount|          Overdue|       Debt Record|  Returned Cheque|Dishon
our of Bill|
+-------+-----------+-----------------+------+---------------+--------------+------------------+-----------------+---
---------------+-----------------+-------------+-----------+-----------------+------------------+-----------------+--
---------------+
|  count|        500|              500|   500|            500|           500|               500|              468|
481|             500|          500|        500|              500|               500|              500|
500|
|   mean|       NULL|           40.946|  NULL|           NULL|          NULL|              4.55|68339.49145299145|275
33.180873180874|            5.33|         NULL|       NULL|            4.974| 46357.55223880597|             4.12|
4.31|
| stddev|       NULL|10.192883485427213|  NULL|           NULL|          NULL|1.5428092950984222| 86796.4936775024|102
09.599414813823|2.048778902170746|         NULL|       NULL|2.4919356907781447|22291.178497846668|2.777233524949642|2.
616025071429688|
|    min|    1B14093|               21|FEMALE|ACCOUNT MANAGER|       MARRIED|                 2|            28366|
9000|               2|  AGRICULTURE|   1,00,000|                1|                 0|                0|
0|
|    max|    IBI4921|               60|  MALE|     TECHNICIAN|        SINGLE|                 7|           930000|
62541|               9|   TRAVELLING|    999,698|                9|            90,000|                9|
10|
```

The provided code performs fundamental exploratory data analysis (EDA) to understand the structure and content of the dataset. First, the df.printSchema() command displays the schema of the DataFrame, including column names, data types, and nullability, ensuring the data types are correctly inferred. Next, the total number of rows is calculated and printed using df.count(), providing insight into the dataset's size and confirming successful data loading. Summary statistics are generated with df.describe().show(), which includes metrics like count, mean, standard deviation, minimum, and maximum for numerical

columns, aiding in identifying patterns or anomalies. Finally, the df.columns command lists all column names, verifying the dataset's structure and checking for unexpected or missing columns. These steps are crucial for validating the data and preparing it for cleaning, transformation, and further analysis.

4) Applying Transformations:

```
04:24 PM (<1s)                                   3                                   Python

# Drop rows with missing values
loan_df_cleaned = df.dropna()
print(f"Rows after dropping nulls: {loan_df_cleaned.count()}")

(2) Spark Jobs
loan_df_cleaned: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]
Rows after dropping nulls: 449
```

The next step involves cleaning the dataset by removing rows with missing values using the dropna() method, which creates a new DataFrame (loan_df_cleaned) without any null entries. This is crucial for ensuring the integrity of downstream data processing and analysis, as missing values can interfere with computations or model training. The command loan_df_cleaned.count() calculates and prints the number of rows remaining after dropping null values, helping assess the impact of this cleaning step. This operation prepares the dataset for further transformations or modeling by ensuring it is free from incomplete records, thereby maintaining data quality.

04:27 PM (<1s) 4 Python

```python
# Filter loans with high Income (e.g., greater than 50,000)
high_loan_df = loan_df_cleaned.filter(loan_df_cleaned['Income'] > 50000)
high_loan_df.show(5)
```

(1) Spark Jobs

```
+-----------+---+------+-----------------+--------------+-----------+------+-----------+-------------+----------------
+-----------+-------+-----------+---------------+-----------------+
|Customer_ID|Age|Gender|        Occupation|Marital Status|Family Size|Income|Expenditure|Use Frequency|   Loan Category
|Loan Amount|Overdue|Debt Record|Returned Cheque|Dishonour of Bill|
+-----------+---+------+-----------------+--------------+-----------+------+-----------+-------------+----------------
+-----------+-------+-----------+---------------+-----------------+
|    IB14008| 44|  MALE|         PROFESSOR|       MARRIED|          6| 51000|      19999|            4|        SHOPPING
|     50,000|      3|     33,999|              1|               5|
|    IB14012| 30|FEMALE|           DENTIST|        SINGLE|          3| 58450|      27675|            5|      TRAVELLING
|     75,000|      6|     20,876|              3|               1|
|    IB14031| 37|FEMALE|  SOFTWARE ENGINEER|       MARRIED|          5| 55999|      23999|            5|      AUTOMOBILE
|     60,999|      2|          0|              5|               3|
|    IB14032| 24|  MALE|      DATA ANALYST|        SINGLE|          4| 60111|      28999|            6|      AUTOMOBILE
|     35,232|      5|     33,333|              1|               2|
|    IB14041| 59|FEMALE|ASSISTANT PROFESSOR|       MARRIED|          4| 50999|      22999|            5|EDUCATIONAL LOAN
|   5,99,934|      3|      9,000|              9|               9|
+-----------+---+------+-----------------+--------------+-----------+------+-----------+-------------+----------------
+-----------+-------+-----------+---------------+-----------------+
only showing top 5 rows
```

The code filters the cleaned dataset (loan_df_cleaned) to create a new DataFrame (high_loan_df) containing rows where the Income column value is greater than 50,000. This is achieved using the filter() method, with the condition loan_df_cleaned['Income'] > 50000 applied to the DataFrame. The filtered results are then displayed using high_loan_df.show(5), which outputs the first five rows of the high-income subset. This step focuses on analyzing loans associated with individuals earning higher incomes, providing insights into trends, loan statuses, or behaviors specific to this income group.

Just now (<1s) 5 Python

```python
# Filter based on age
lowage_loan_df = loan_df_cleaned.filter(loan_df_cleaned['Age'] < 24 )
lowage_loan_df.show(5)
```

(1) Spark Jobs

lowage_loan_df: pyspark.sql.dataframe.DataFrame = [Customer_ID: string, Age: integer ... 13 more fields]

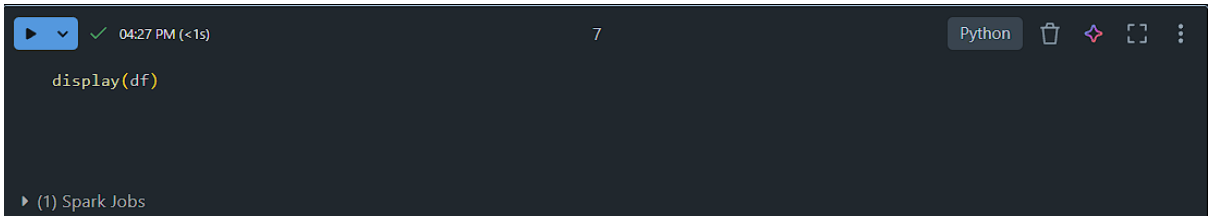
```
+-----------+---+------+----------+--------------+-----------+------+-----------+-------------+----------------+--------
---+-------+-----------+---------------+-----------------+
|Customer_ID|Age|Gender|Occupation|Marital Status|Family Size|Income|Expenditure|Use Frequency|   Loan Category|Loan Amo
unt|Overdue|Debt Record|Returned Cheque|Dishonour of Bill|
+-----------+---+------+----------+--------------+-----------+------+-----------+-------------+----------------+--------
---+-------+-----------+---------------+-----------------+
|    1B14093| 21|FEMALE|   MANAGER|        SINGLE|          3| 42516|      24567|            7|      AUTOMOBILE|  25,69,
874|      8|     89,652|              2|               3|
|    1B14312| 21|FEMALE|   MANAGER|        SINGLE|          3| 42516|      24567|            7|EDUCATIONAL LOAN|  25,69,
874|      8|     89,652|              2|               3|
+-----------+---+------+----------+--------------+-----------+------+-----------+-------------+----------------+--------
---+-------+-----------+---------------+-----------------+
```

The code filters the cleaned dataset (loan_df_cleaned) to create a new DataFrame (lowage_loan_df) containing only rows where the Age column value is less than 24. This is achieved using the filter() method, which applies the condition loan_df_cleaned['Age'] < 24 to the DataFrame. The resulting subset is displayed using lowage_loan_df.show(5), which outputs the first five rows of the filtered data. This step allows for targeted analysis of a specific demographic group—in this case, younger individuals—helping identify trends or patterns unique to this age segment.

5) Visualizations:

This will render the dataset in a tabular format, making it easy to view all the records in the notebook.

Visualization 1: Data Profile: The data profile provides an overview of key statistics such as mean, standard deviation, and range for each numerical column. Databricks offers automatic data profiling capabilities, but you can also create custom visualizations based on specific columns. To visualize the summary statistics for the entire dataset, you can use df.describe().show() in combination with visual tools.



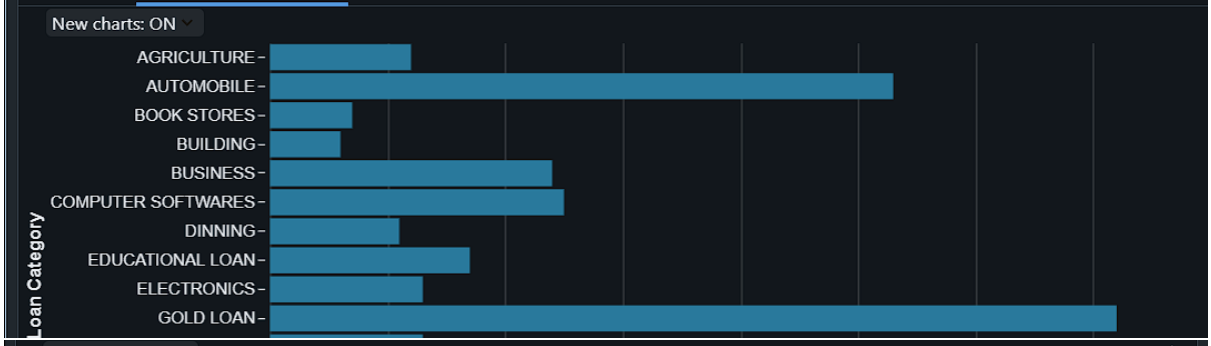
Visualization 2: Number of Loans Taken For Each Loan Category: This visualization helps you understand how many loans have been taken in each loan category (e.g., loan_status, loan_type). To create this visualization.

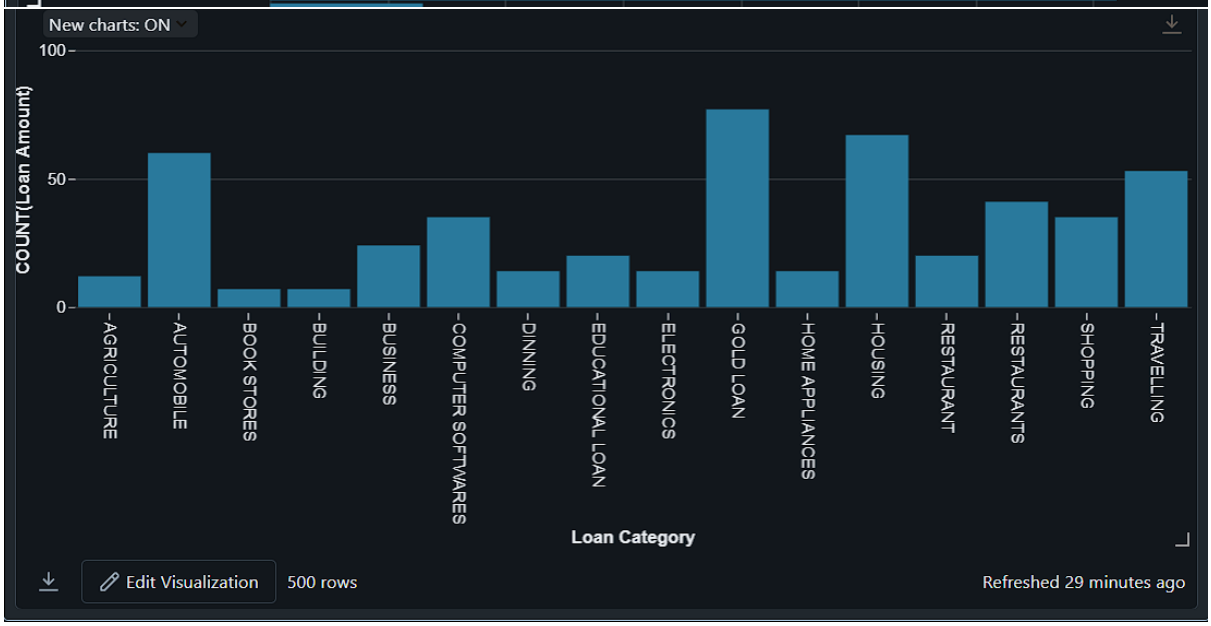
```
loan_count_by_category = loan_df_cleaned.groupBy("Loan Category").count()
display(loan_count_by_category)
```

(2) Spark Jobs

loan_count_by_category: pyspark.sql.dataframe.DataFrame = [Loan Category: string, count: long]

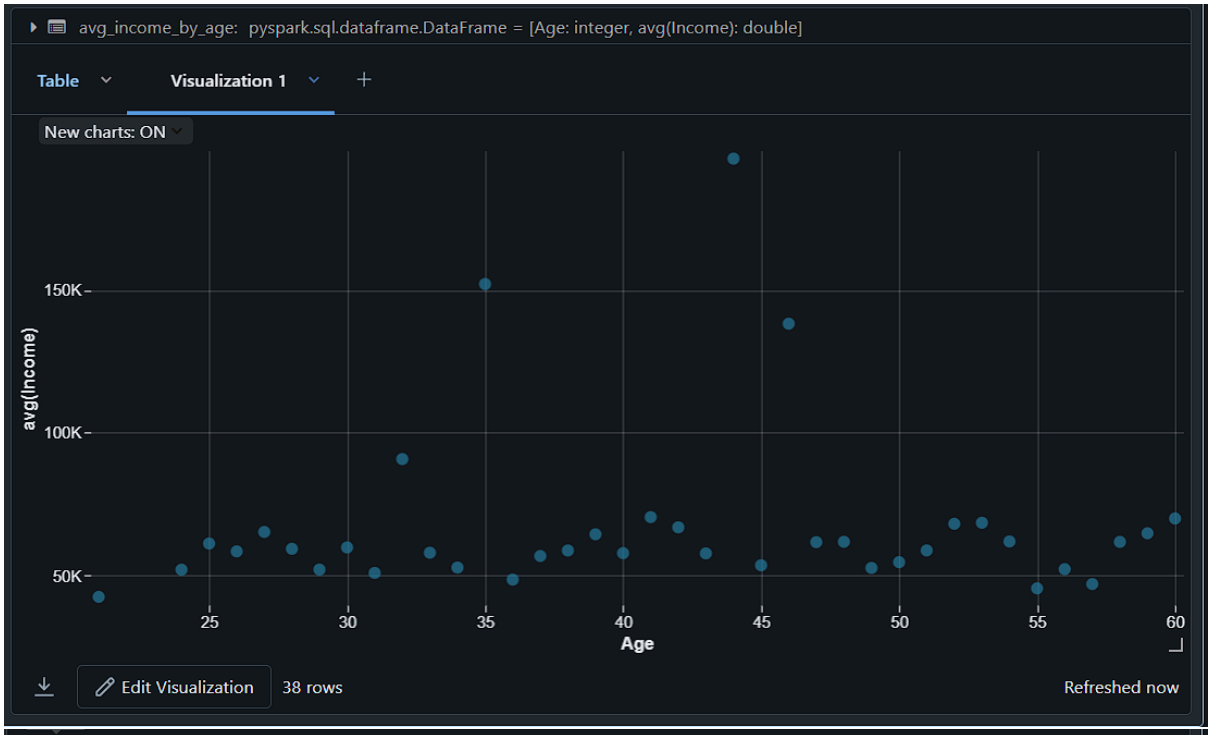
Table | Visualization 1





Visualization 3: Average Income by Age: To calculate the average income by age group, you can use a groupBy operation followed by aggregation

```
avg_income_by_age = loan_df_cleaned.groupBy("Age").agg({"Income": "avg"})
display(avg_income_by_age)
```


avg_income_by_age: pyspark.sql.dataframe.DataFrame = [Age: integer, avg(Income): double]
Table
Visualization 1
New charts: ON
avg(Income)
150K
100K
50K
25
30
35
40
45
50
55
60
Age
Edit Visualization
38 rows
Refreshed now

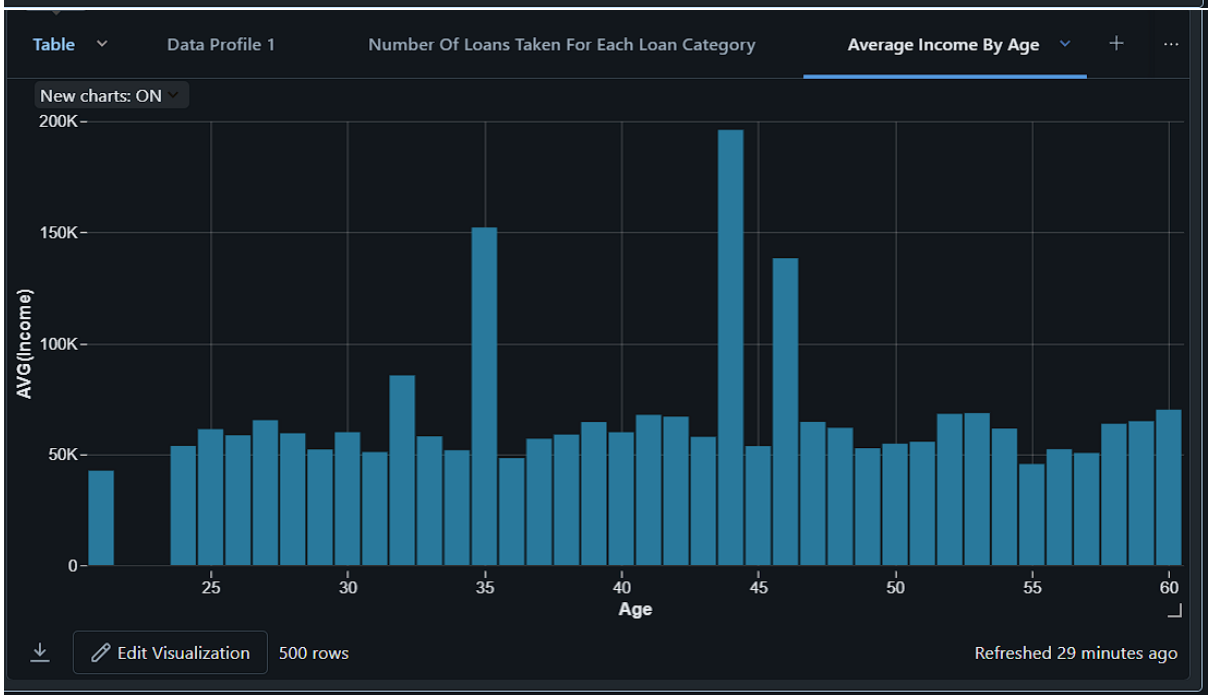


Table
Data Profile 1
Number Of Loans Taken For Each Loan Category
Average Income By Age
New charts: ON
AVG(Income)
200K
150K
100K
50K
0
25
30
35
40
45
50
55
60
Age
Edit Visualization
500 rows
Refreshed 29 minutes ago

Some Additional Visualizations: