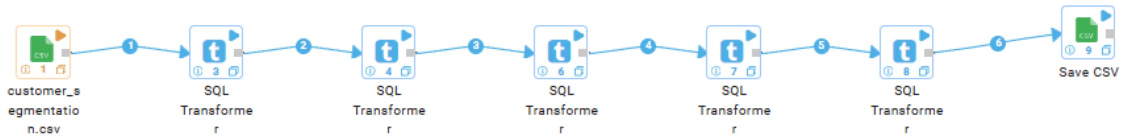
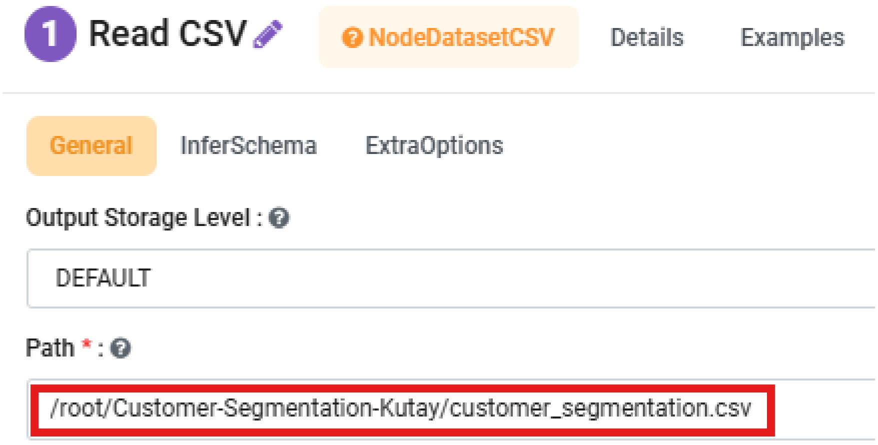
WORKFLOW 1 - FEATURE ENGINEERING



Node 1 - Read CSV (customer_segmentation.csv)



Output Storage Level :

DEFAULT

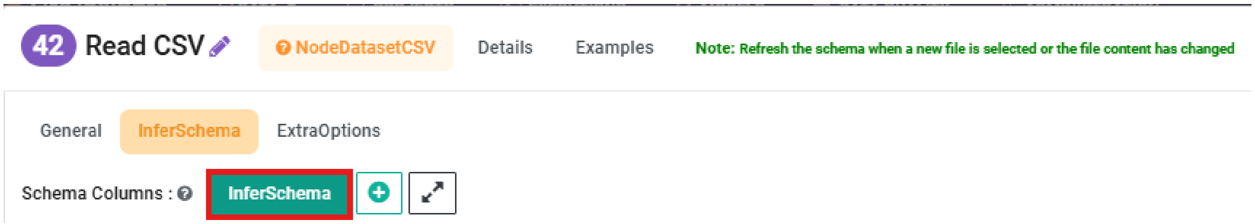
Path * :

/root/Customer-Segmentation-Kutay/customer_segmentation.csv

Path Selection

In the *General* tab, provide the full path to your dataset.

- Example used here: /root/fraud_in_credit_transaction/customer_segmentation.csv
- Make sure the file is correctly uploaded to this directory in Sparkflows.



Note: Refresh the schema when a new file is selected or the file content has changed

Schema Columns :

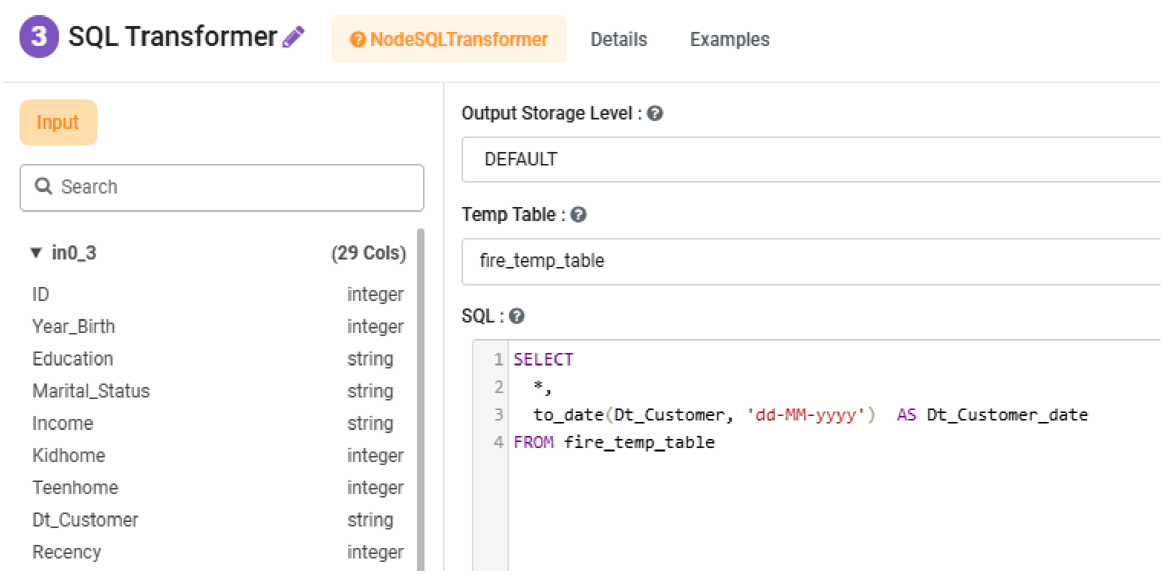
Schema Inference

After selecting the file path, switch to the **InferSchema** tab.

- Click the **green "Infer Schema" button** to automatically detect column names and data types.
- This ensures Sparkflows correctly interprets the dataset structure before running downstream nodes.

Tip: Always refresh the schema if you update the dataset or replace the file otherwise, mismatches can cause execution errors later.

Node 2- SQL Transformer



▾ in0_3 (29 Cols)

ID	integer
Year_Birth	integer
Education	string
Marital_Status	string
Income	string
Kidhome	integer
Teenhome	integer
Dt_Customer	string
Recency	integer

Output Storage Level : DEFAULT

Temp Table : fire_temp_table

SQL :

```
SELECT
  *,
  to_date(Dt_Customer, 'dd-MM-yyyy')  AS Dt_Customer_date
FROM fire_temp_table
```

Purpose

This node is used to transform the raw dataset by converting the **Dt_Customer** column from string format into a proper date format. Having a true date column allows for easier feature engineering later on.

Configuration

- **Temp Table: fire_temp_table**
- **Output Storage Level: DEFAULT**

Transformation Logic

1. **Preserve Existing Columns**
   - The **SELECT *** ensures all original columns remain unchanged in the output.

2. **Date Conversion**
    - **to_date(Dt_Customer, 'dd-MM-yyyy')** converts the **Dt_Customer** string into a standardized date object.

    - A new column **Dt_Customer_date** is created, making it possible to compute derived features like:

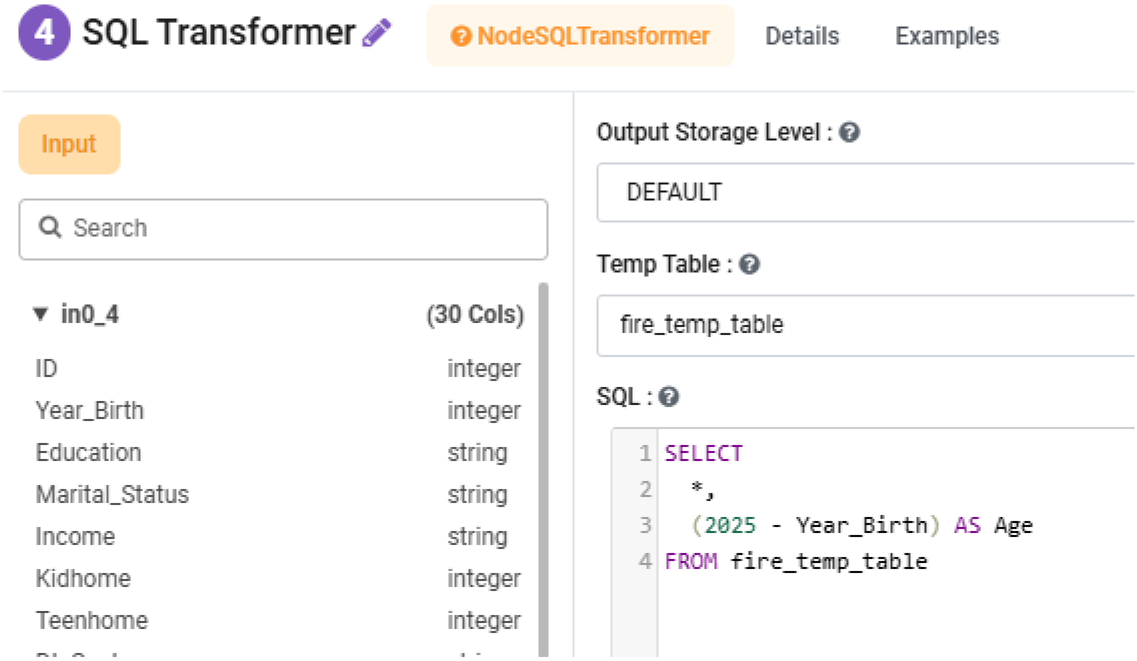
        - Customer tenure (e.g., how long since the customer joined).
        - Cohort analysis (grouping customers by join month/year).

3. **Output Schema**
    - After running the SQL, the schema inference will detect one additional column: **Dt_Customer_date** (data type: date).

Role in Workflow

This transformation prepares the dataset for deeper feature engineering in later steps. By standardizing dates, Sparkflows can handle temporal calculations, which are crucial for segmentation (e.g., segmenting “new vs loyal customers”).

Node 3 - SQL Transformer



▾ in0_4 (30 Cols)

ID	integer
Year_Birth	integer
Education	string
Marital_Status	string
Income	string
Kidhome	integer
Teenhome	integer

Output Storage Level : DEFAULT

Temp Table : fire_temp_table

SQL :

```sql
SELECT
  *,
  (2025 - Year_Birth) AS Age
FROM fire_temp_table
```

Purpose
This node is used to derive the Age of each customer from their year of birth. Instead of keeping **Year_Birth** as a raw feature, calculating age makes the dataset easier to interpret and allows clustering algorithms to use a more meaningful numerical attribute.

Configuration

- **Temp Table: fire_temp_table**
- **Output Storage Level: DEFAULT**

Transformation Logic

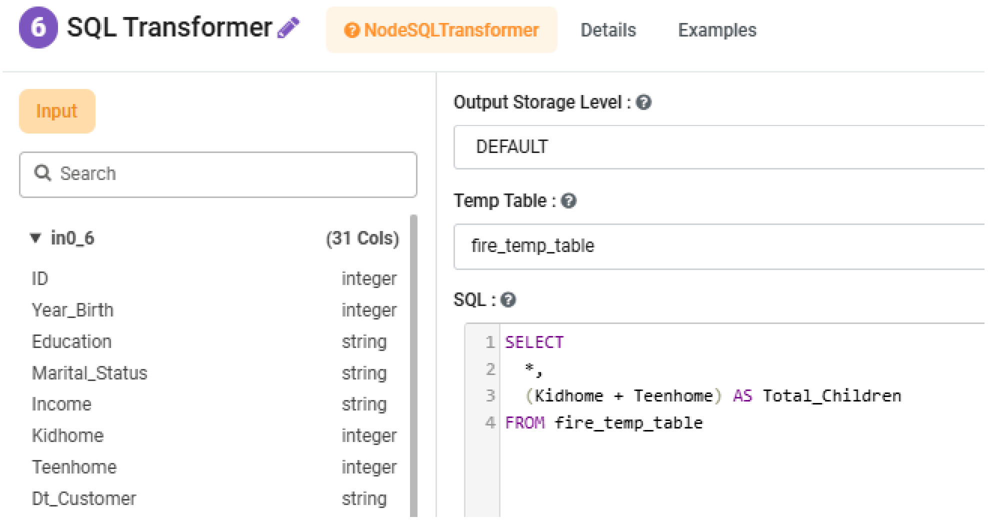
1. **Preserve Existing Columns**
    - The **SELECT *** ensures all existing fields remain available in the dataset.

2. **Age Calculation**
    - **(2025 - Year_Birth)** computes each customer's age, assuming the current analysis year is 2025.
    - The result is stored in a new column named **Age**.

3. **Output Schema**
    - After this transformation, a new integer column **Age** is added to the dataset.

Role in Workflow
This step converts a static feature (**Year_Birth**) into a dynamic, interpretable feature (**Age**). It supports segmentation by allowing the clustering process to group customers by demographic characteristics, such as younger vs. older customers, which can reveal distinct behavioral patterns.

Node 4 - SQL Transformer



▾ in0_6 (31 Cols)

ID	integer
Year_Birth	integer
Education	string
Marital_Status	string
Income	string
Kidhome	integer
Teenhome	integer
Dt_Customer	string

Output Storage Level : DEFAULT

Temp Table : fire_temp_table

SQL :

```sql
SELECT
  *,
  (Kidhome + Teenhome) AS Total_Children
FROM fire_temp_table
```

Purpose

This node creates a new feature called Total_Children, which combines the number of children living at home (**Kidhome**) and teenagers living at home (**Teenhome**). Instead of analyzing these columns separately, merging them provides a single, more interpretable feature that reflects the total number of dependents in the household.

Configuration

- **Temp Table: fire_temp_table**
- **Output Storage Level: DEFAULT**

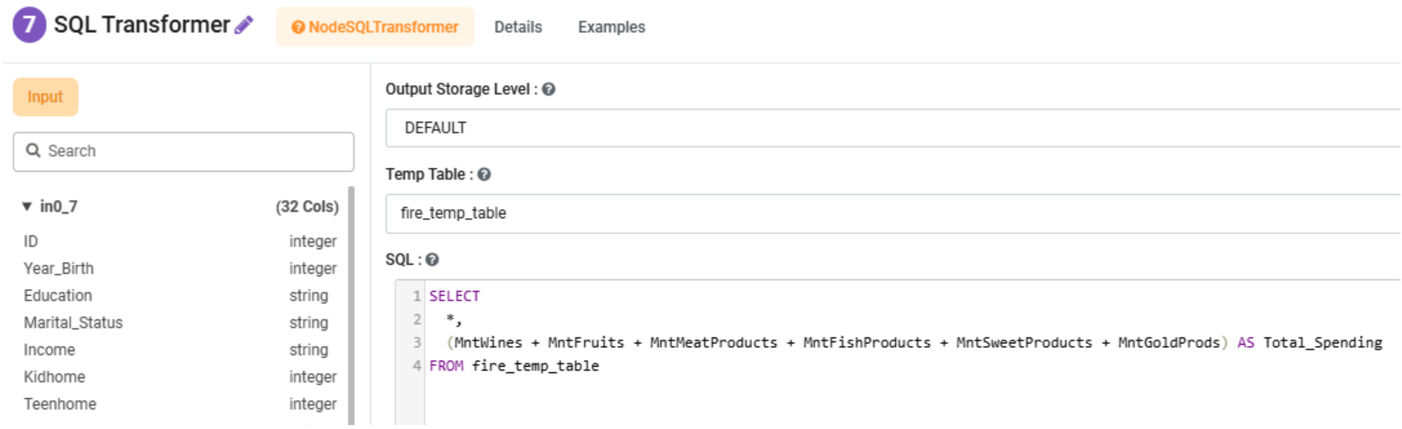
Transformation Logic

1. **Preserve Existing Columns**
   - The **SELECT *** keeps all original attributes intact.
2. **Feature Engineering**
   - **(Kidhome + Teenhome)** calculates the total number of children/teenagers in the household.
   - This new value is saved in the column Total_Children**.**
3. **Output Schema**
   - A new integer column Total_Children is added to the dataset.

Role in Workflow

Household size is an important factor in customer segmentation. By combining **Kidhome** and **Teenhome**, the dataset gains a consolidated feature that can highlight family-oriented customers vs. single/child-free customers. This distinction helps clustering algorithms detect lifestyle-based customer segments (e.g., families vs. singles).

Node 5 - SQL Transformer



▾ in0_7	(32 Cols)
ID	integer
Year_Birth	integer
Education	string
Marital_Status	string
Income	string
Kidhome	integer
Teenhome	integer

Output Storage Level : DEFAULT

Temp Table : fire_temp_table

SQL :

```
SELECT
  *,
  (MntWines + MntFruits + MntMeatProducts + MntFishProducts + MntSweetProducts + MntGoldProds) AS Total_Spending
FROM fire_temp_table
```

Purpose

This node creates a new aggregated feature called **Total_Spending**, which represents the sum of all product category expenditures (wine, fruits, meat, fish, sweets, and gold). Instead of analyzing each spending category separately, combining them provides a single metric to evaluate overall customer spending behavior.

Configuration

- **Temp Table: fire_temp_table**
- **Output Storage Level: DEFAULT**

Transformation Logic

1. **Preserve Existing Columns**
    - **SELECT *** keeps all other features available for future analysis.
2. **Aggregate Spending**
    - The expression adds up all monetary values across product categories:
        - **MntWines**
        - **MntFruits**
        - **MntMeatProducts**
        - **MntFishProducts**

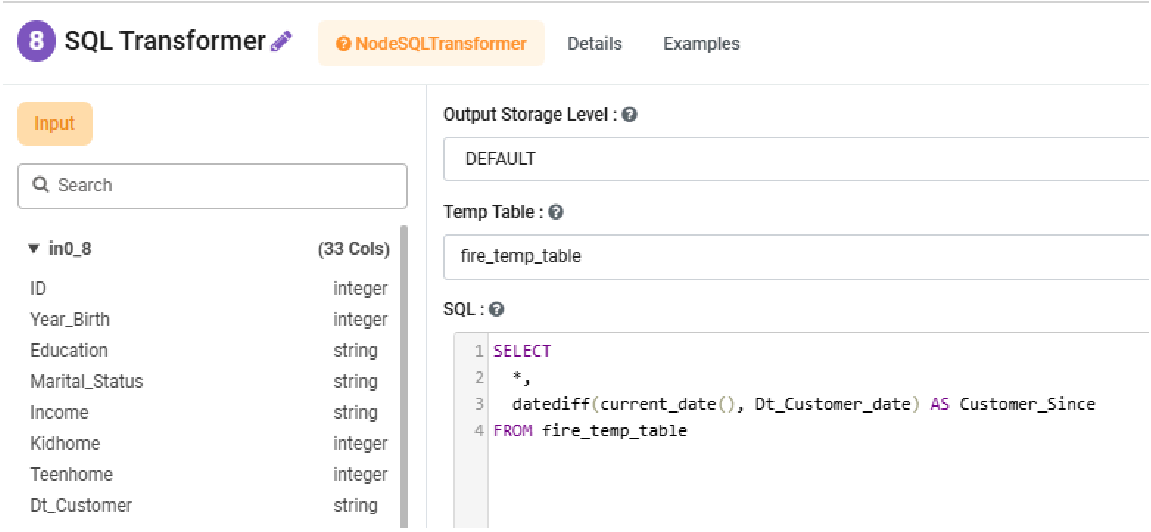
        - **MntSweetProducts**
        - **MntGoldProds**
    - This total value is stored in the column **Total_Spending**.

3. **Output Schema**
    - A new numeric column **Total_Spending** is added to the dataset.

Role in Workflow

Total spending is one of the **key behavioral features** for customer segmentation. It highlights high-value vs. low-value customers and allows clustering algorithms to distinguish between heavy spenders, moderate buyers, and low spenders. This feature is essential for marketing strategy, as it directly relates to customer lifetime value.

Node 6 - SQL Transformer



in0_8	(33 Cols)
ID	integer
Year_Birth	integer
Education	string
Marital_Status	string
Income	string
Kidhome	integer
Teenhome	integer
Dt_Customer	string

Output Storage Level : DEFAULT

Temp Table : fire_temp_table

SQL :

```sql
SELECT
  *,
  datediff(current_date(), Dt_Customer_date) AS Customer_Since
FROM fire_temp_table
```

Purpose

Compute **customer tenure** by measuring how long each customer has been with the company. This node creates **Customer_Since** as the day-level difference between **today** and each customer's join date (**Dt_Customer_date**).

Configuration

- **Temp Table: fire_temp_table**
- **Output Storage Level: DEFAULT**

Transformation Logic

1. **Keep all columns** with **SELECT ***.
2. **Tenure in days:**
   - **datediff(current_date(), Dt_Customer_date)** returns the number of **days** from the join date to today (Spark SQL **datediff** unit = days).
     Result saved as **Customer_Since**.
3. **Dynamic reference date:**
   - **current_date()** ensures the tenure updates automatically when the workflow runs on a later date.

Output Schema

- Adds an integer column: **Customer_Since** (days since first becoming a customer).

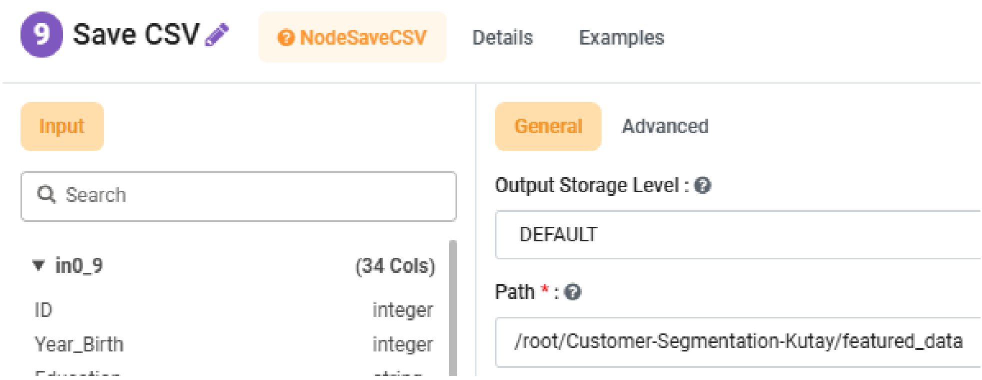
Role in Workflow

Tenure is a high-signal feature for segmentation (e.g., **new vs. loyal** customers). Clusters are often separated by recency/tenure along with spending and engagement features, enabling actions like onboarding campaigns for new users vs. retention offers for long-timers.

Notes / Good Practices

- If **Dt_Customer_date** can be null or malformed, add a filter or **CASE WHEN Dt_Customer_date IS NOT NULL** safeguard upstream.
- You can derive month/year versions later, e.g., `floor(Customer_Since/30)` for **months** or normalization for algorithms sensitive to scale.

Node 7 - Save CSV



in0_9 (34 Cols)

ID integer

Year_Birth integer

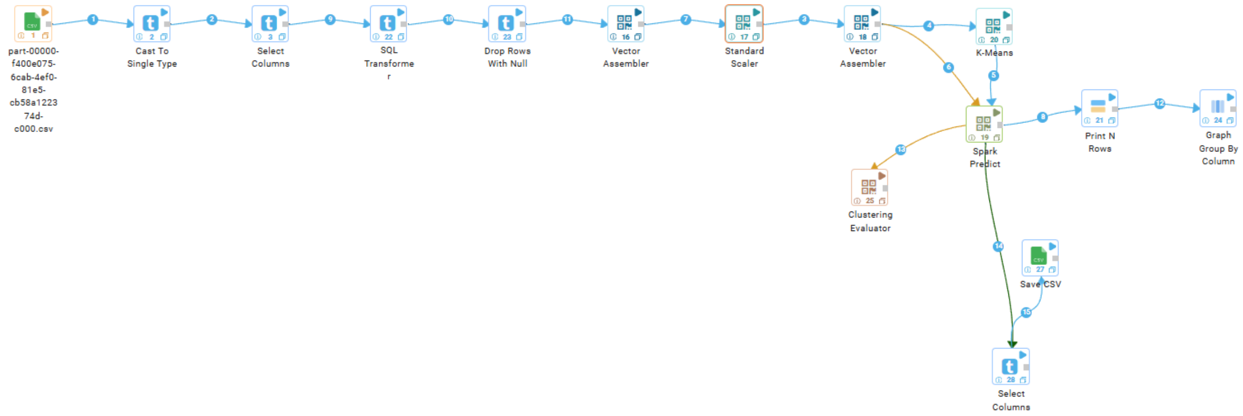
Output Storage Level : DEFAULT

Path * : /root/Customer-Segmentation-Kutay/featured_data

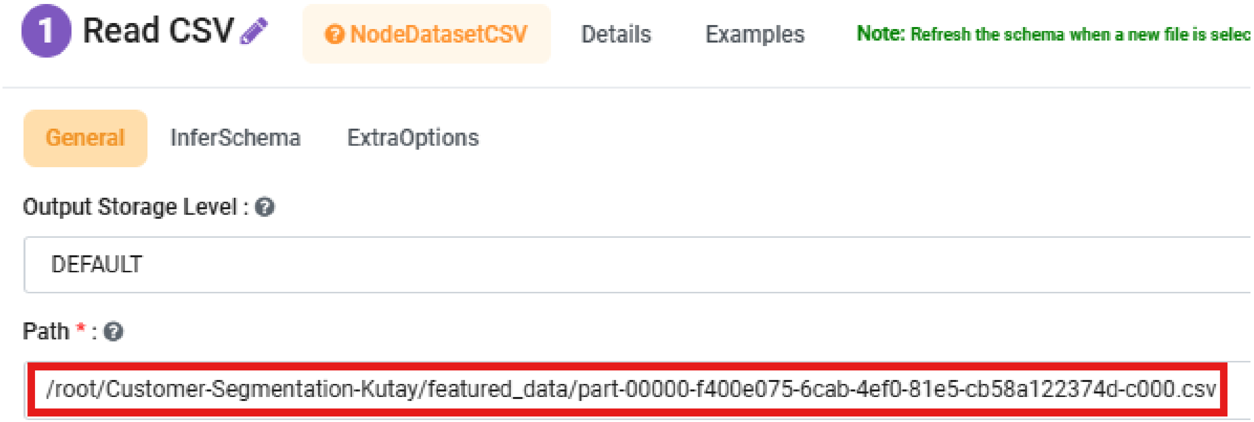
Purpose

This node saves the **final feature-engineered** dataset to a CSV file for later use. After applying all SQL transformations (date conversion, age calculation, total children, total spending, customer tenure), the enriched dataset is exported in a structured format that can be reused for clustering, reporting, or further analysis.

WORKFLOW 2 - KMeans



Node 1 - Read CSV (part-00000-f400e075-6cab-4ef0-81e5-cb58a122374d-c000.csv)



Note: Refresh the schema when a new file is selec

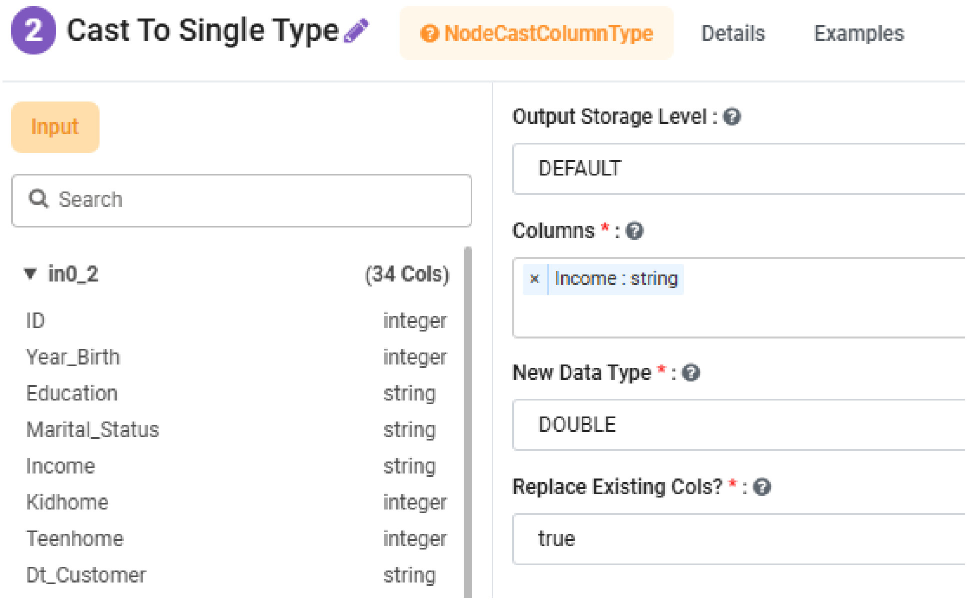
Output Storage Level : DEFAULT

Path * : /root/Customer-Segmentation-Kutay/featured_data/part-00000-f400e075-6cab-4ef0-81e5-cb58a122374d-c000.csv

Purpose
Load the **feature-engineered dataset** produced by the previous workflow so we can proceed with scaling and clustering (K-Means) without redoing transformations. Configuration is the **same as the first Read CSV node** we did.

,

Node 2 - Cast To Single Type



2 Cast To Single Type

▾ in0_2 (34 Cols)

ID	integer
Year_Birth	integer
Education	string
Marital_Status	string
Income	string
Kidhome	integer
Teenhome	integer
Dt_Customer	string

Output Storage Level : DEFAULT

Columns * : × Income : string

New Data Type * : DOUBLE

Replace Existing Cols? * : true

Purpose

Convert the **Income** column from **string** to **double**. This is required because clustering algorithms (like K-Means) only accept numeric input, and the **Vector Assembler** node cannot process string types.

Configuration

- **Output Storage Level: DEFAULT**
- **Columns: Income : string**
- **New Data Type: DOUBLE**
  Replace Existing Cols?: true (overwrites the old **Income** column with the new double-typed version)

Transformation Logic

1. The node scans the column **Income**, which was originally read as a string.
2. Converts each entry into a double precision number.
3. Ensures that the column remains with the same name (**Income**), since **Replace Existing Cols? = true**.

Output Schema

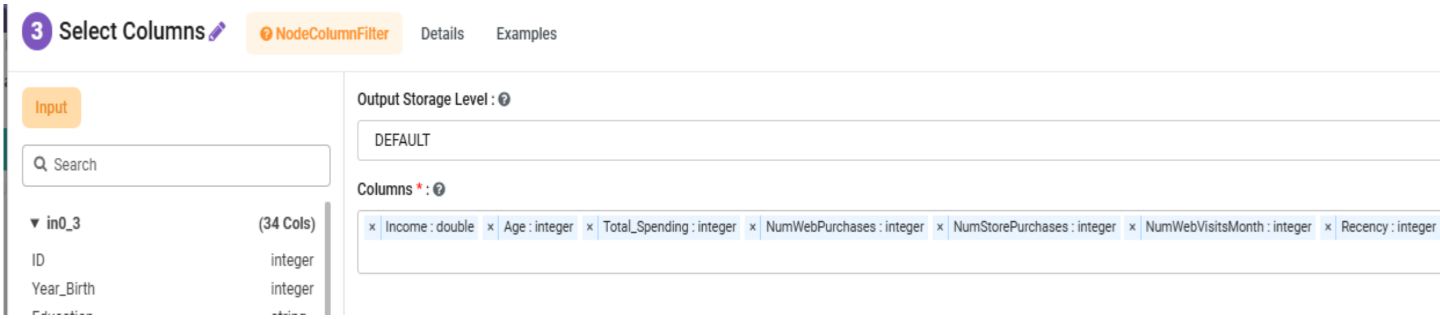
- **`Income`** now appears as a **double** (numeric) column.

- This guarantees compatibility with downstream nodes like **Vector Assembler** and **Standard Scaler**.

Role in Workflow

This step is critical for data preprocessing. Without it, the workflow would fail at the vectorization stage, since Sparkflows enforces numerical inputs for machine learning models. By casting early, we make sure the dataset is clean, consistent, and ready for feature assembly.

Node 3 - Select Columns



in0_3 (34 Cols)
ID integer
Year_Birth integer

Output Storage Level : DEFAULT

Columns * : × Income : double × Age : integer × Total_Spending : integer × NumWebPurchases : integer × NumStorePurchases : integer × NumWebVisitsMonth : integer × Recency : integer

Purpose

Choose only the **relevant numeric features** that will be used as input for the **K-Means clustering model**. This step filters out unnecessary columns (like IDs, categorical variables, or date fields) and ensures that only meaningful attributes are passed forward.

Configuration

- **Output Storage Level: DEFAULT**
- **Selected Columns:**
  - **Income** (double)
  - **Age** (integer)
  - **Total_Spending** (integer)
  - **NumWebPurchases** (integer)
  - **NumStorePurchases** (integer)
  - **NumWebVisitsMonth** (integer)
  - **Recency** (integer)

Transformation Logic

1. Removes non-essential columns (e.g., **ID**, **Education**, **Marital_Status**) that are not directly useful for clustering.

2. Keeps only numerical features that describe customer **demographics, spending, and behavior**.
3. Prepares a clean subset of columns for the **Vector Assembler** node.

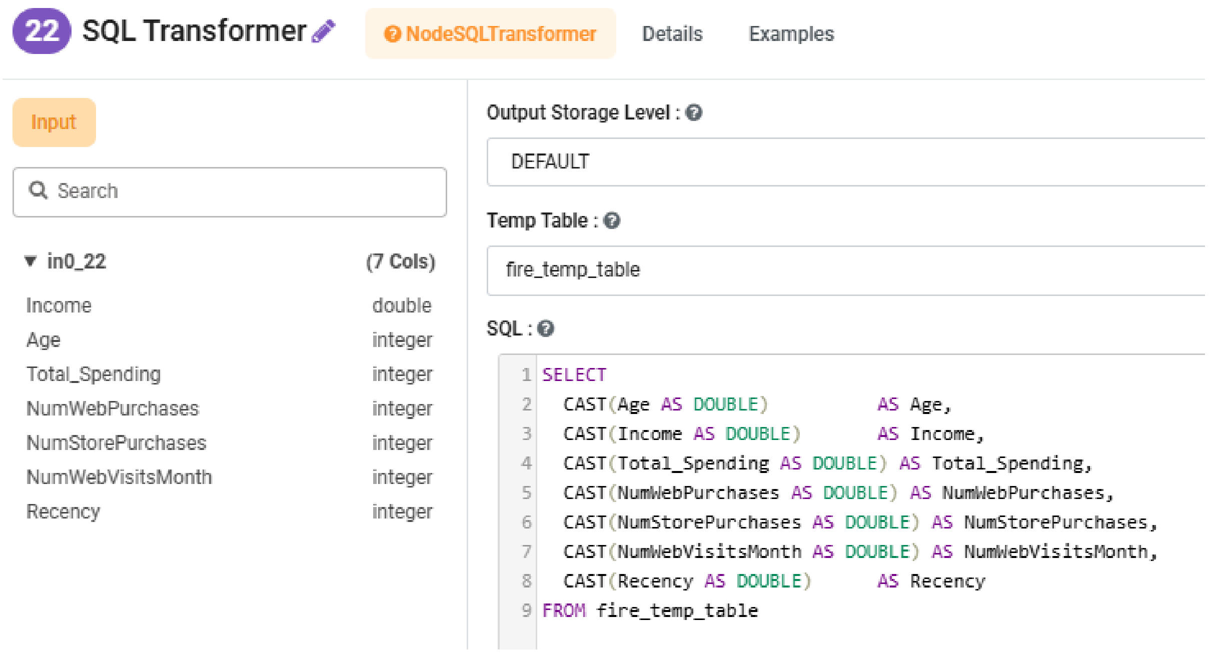
Output Schema

- A reduced dataset with only the **7 features** listed above, all in numeric format.

Role in Workflow

This step is critical for **dimensionality control**. Clustering models like K-Means rely on numerical distance metrics (e.g., Euclidean distance). Including irrelevant or categorical features would distort distances and reduce cluster quality. By selecting only carefully engineered features, this node ensures meaningful and interpretable clusters.

Node 4 - SQL Transformer



▼ in0_22 (7 Cols)

Income	double
Age	integer
Total_Spending	integer
NumWebPurchases	integer
NumStorePurchases	integer
NumWebVisitsMonth	integer
Recency	integer

Output Storage Level : DEFAULT

Temp Table : fire_temp_table

SQL :

```
SELECT
  CAST(Age AS DOUBLE)          AS Age,
  CAST(Income AS DOUBLE)       AS Income,
  CAST(Total_Spending AS DOUBLE) AS Total_Spending,
  CAST(NumWebPurchases AS DOUBLE) AS NumWebPurchases,
  CAST(NumStorePurchases AS DOUBLE) AS NumStorePurchases,
  CAST(NumWebVisitsMonth AS DOUBLE) AS NumWebVisitsMonth,
  CAST(Recency AS DOUBLE)      AS Recency
FROM fire_temp_table
```

Purpose

Standardize the data types of all selected features by casting them to **double**. Many machine learning algorithms (including **K-Means**) in Sparkflows require all input features to have consistent numeric types. This step ensures no type mismatch errors occur during vectorization or clustering.

Configuration

- **Temp Table: fire_temp_table**
- **Output Storage Level: DEFAULT**

Transformation Logic

1. **Casts All Columns**
   - Converts every chosen feature (**Age, Income, Total_Spending, NumWebPurchases, NumStorePurchases, NumWebVisitsMonth, Recency**) into **double** type.
2. **Renames Columns Back**
   - Keeps the same column names so downstream nodes don't break.
3. **Ensures Consistency**
   - Guarantees the Vector Assembler receives **only double-type features**, avoiding runtime type errors.

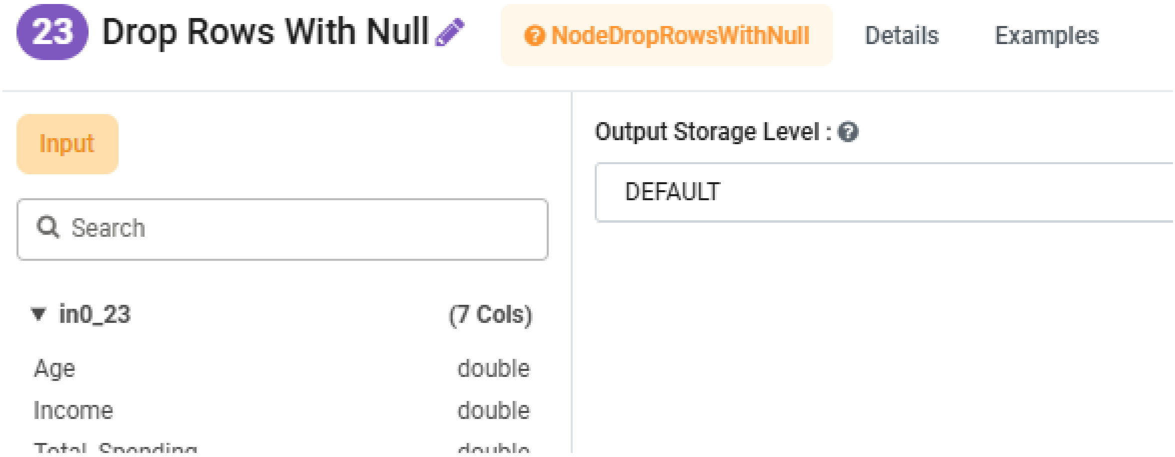
Output Schema

- Dataset with **7 features**, all of type **double**:
  - **Age, Income, Total_Spending, NumWebPurchases, NumStorePurchases, NumWebVisitsMonth, Recency.**

Role in Workflow

This step is a **safeguard transformation**. Even if some columns were already integers or doubles, explicitly casting ensures compatibility.

Node 5 - Drop Rows With Null



Output Storage Level : DEFAULT

in0_23 (7 Cols)
Age double
Income double
Total_Spending double

Purpose

Remove rows that contain **null or missing values** in any of the selected feature columns. This step is necessary because downstream stages like **Vector Assembler**, **Standard Scaler**, and **K-Means** cannot handle null values and would otherwise throw errors.

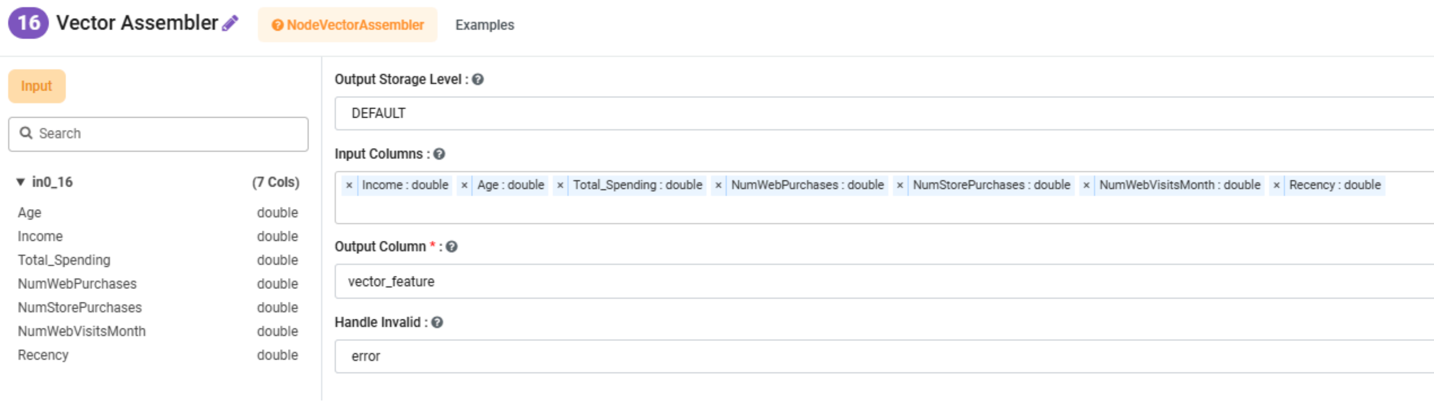
Transformation Logic

1. Scans every row for null values across the selected features.
2. Drops any row where one or more features are missing.
3. Keeps only **complete cases** to ensure a clean dataset.

Role in Workflow

This step ensures **data quality and compatibility**. If nulls are present, distance calculations in K-Means would break. By dropping incomplete rows, the dataset is ready for **Vector Assembler** → **Standard Scaler** → **K-Means clustering** without risk of null-related runtime errors.

Node 6 - Vector Assembler



in0_16 (7 Cols)

Age	double
Income	double
Total_Spending	double
NumWebPurchases	double
NumStorePurchases	double
NumWebVisitsMonth	double
Recency	double

Output Storage Level : DEFAULT

Input Columns : × Income : double × Age : double × Total_Spending : double × NumWebPurchases : double × NumStorePurchases : double × NumWebVisitsMonth : double × Recency : double

Output Column * : vector_feature

Handle Invalid : error

Purpose

Combine all selected **numeric feature columns** into a single vector column called **vector_feature**. This step is required because the **Standard Scaler** node (and most Spark MLlib models) can only process feature vectors, not separate numeric columns.

Configuration

- **Output Storage Level: DEFAULT**
- **Input Columns:**
  - **`Income`**
  - **`Age`**
  - **`Total_Spending`**
  - **`NumWebPurchases`**
  - **`NumStorePurchases`**
  - **`NumWebVisitsMonth`**

    - `Recency`

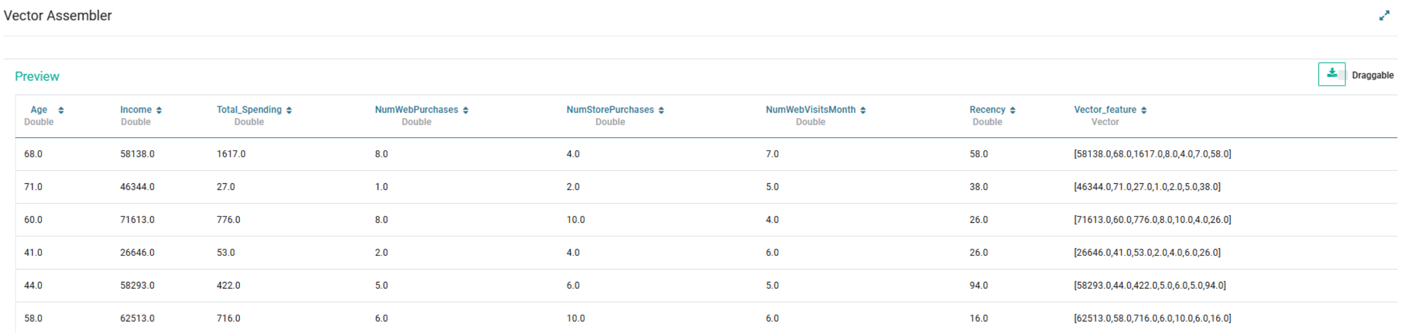
- **Output Column: `vector_feature`**
- **Handle Invalid: error** (workflow fails if invalid values exist, ensuring data integrity before scaling)

Transformation Logic

1. Collects all the chosen numeric features into a single **dense vector**.
2. Stores this vector in the new column **vector_feature**.
3. Prepares the dataset for **scaling**, ensuring each observation is represented as a single vector instead of multiple columns.

Output Schema

- Adds one new column: **vector_feature** (Vector type).
- Retains all the original numeric columns in the dataset as well.



Vector Assembler

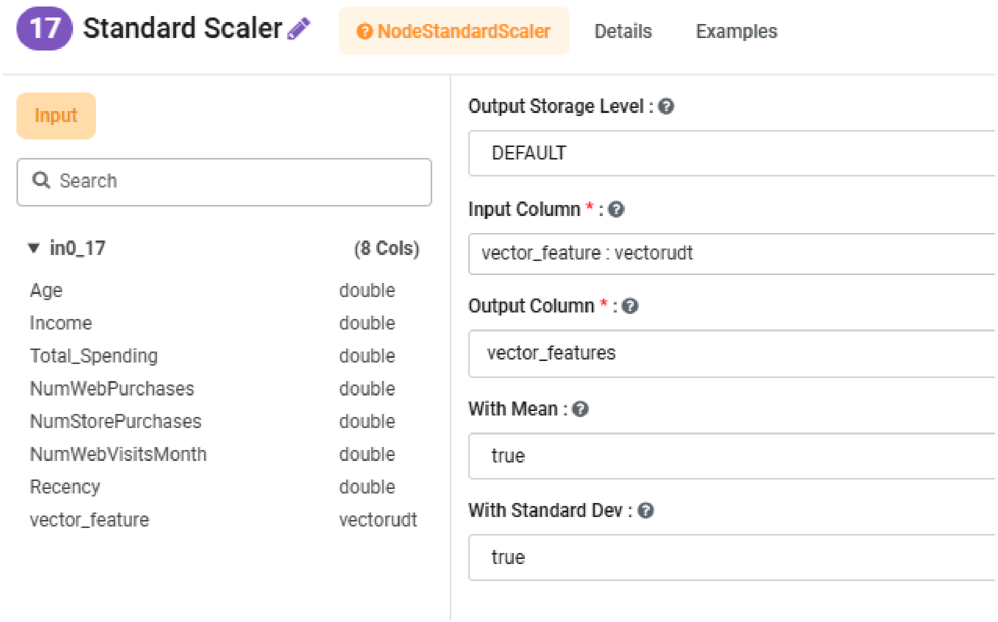
Preview

Age Double	Income Double	Total_Spending Double	NumWebPurchases Double	NumStorePurchases Double	NumWebVisitsMonth Double	Recency Double	Vector_feature Vector
68.0	58138.0	1617.0	8.0	4.0	7.0	58.0	[58138.0,68.0,1617.0,8.0,4.0,7.0,58.0]
71.0	46344.0	27.0	1.0	2.0	5.0	38.0	[46344.0,71.0,27.0,1.0,2.0,5.0,38.0]
60.0	71613.0	776.0	8.0	10.0	4.0	26.0	[71613.0,60.0,776.0,8.0,10.0,4.0,26.0]
41.0	26646.0	53.0	2.0	4.0	6.0	26.0	[26646.0,41.0,53.0,2.0,4.0,6.0,26.0]
44.0	58293.0	422.0	5.0	6.0	5.0	94.0	[58293.0,44.0,422.0,5.0,6.0,5.0,94.0]
58.0	62513.0	716.0	6.0	10.0	6.0	16.0	[62513.0,58.0,716.0,6.0,10.0,6.0,16.0]

Role in Workflow

This is the **bridge step** between raw numeric features and preprocessing/modeling. Without it, the **Standard Scaler** and node cannot accept the dataset.

Node 7 - Standard Scaler



17 Standard Scaler

▾ in0_17 (8 Cols)

Age	double
Income	double
Total_Spending	double
NumWebPurchases	double
NumStorePurchases	double
NumWebVisitsMonth	double
Recency	double
vector_feature	vectorudt

Output Storage Level : DEFAULT

Input Column * : vector_feature : vectorudt

Output Column * : vector_features

With Mean : true

With Standard Dev : true

Purpose

Normalize the feature vectors by applying **standardization**. This step ensures that all features contribute equally to distance calculations in K-Means, preventing features with larger numeric ranges (e.g., **`Income`**, **`Total_Spending`**) from dominating smaller-scaled features (e.g., **`NumWebPurchases`**).

Configuration

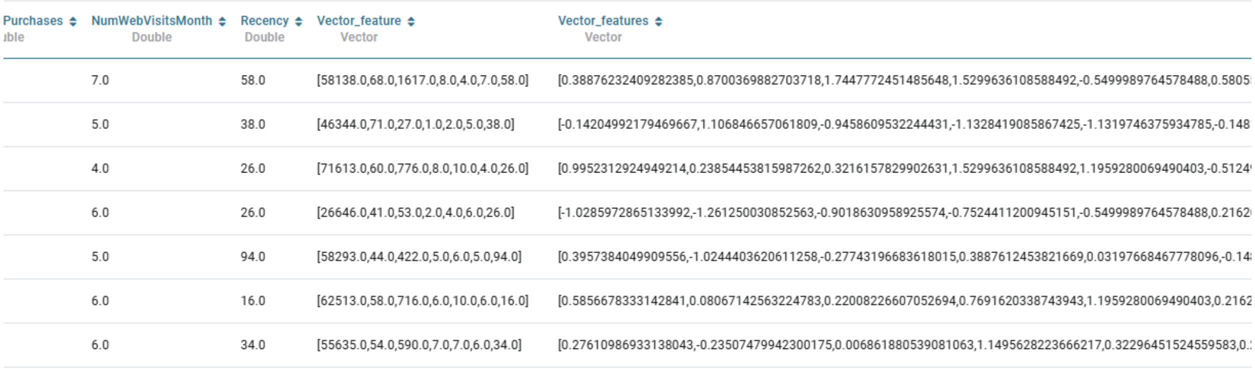
- **Output Storage Level: DEFAULT**
- **Input Column: vector_feature** (created by the Vector Assembler)
- **Output Column: vector_features**
- **With Mean: `true`** → Centers features by subtracting the mean.
- **With Standard Dev: `true`** → Scales features by dividing by standard deviation.

Transformation Logic

1. Takes the dense feature vector from the Vector Assembler (**vector_feature**).
2. Produces a new standardized feature vector stored in **`vector_features`**.

Output Schema

- Adds one new column: **vector_features** (standardized vector).
- Keeps the original raw features and the unscaled **`vector_feature`**.

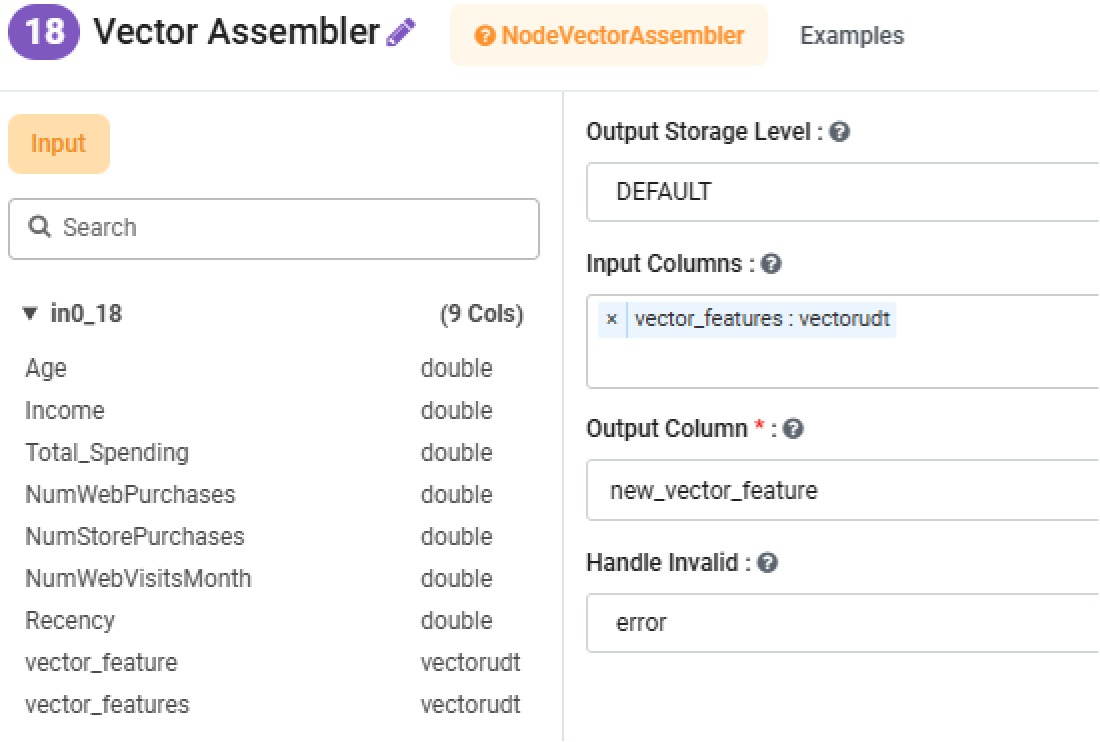


Purchases ⬍	NumWebVisitsMonth ⬍	Recency ⬍	Vector_feature ⬍	Vector_features ⬍
ıble	Double	Double	Vector	Vector
	7.0	58.0	[58138.0,68.0,1617.0,8.0,4.0,7.0,58.0]	[0.38876232409282385,0.8700369882703718,1.7447772451485648,1.5299636108588492,-0.5499989764578488,0.5805
	5.0	38.0	[46344.0,71.0,27.0,1.0,2.0,5.0,38.0]	[-0.14204992179469667,1.106846657061809,-0.9458609532244431,-1.1328419085867425,-1.1319746375934785,-0.148
	4.0	26.0	[71613.0,60.0,776.0,8.0,10.0,4.0,26.0]	[0.9952312924949214,0.23854453815987262,0.3216157829902631,1.5299636108588492,1.1959280069490403,-0.5124
	6.0	26.0	[26646.0,41.0,53.0,2.0,4.0,6.0,26.0]	[-1.0285972865133992,-1.261250030852563,-0.9018630958925574,-0.7524411200945151,-0.5499989764578488,0.2162
	5.0	94.0	[58293.0,44.0,422.0,5.0,6.0,5.0,94.0]	[0.3957384049909556,-1.0244403620611258,-0.27743196683618015,0.3887612453821669,0.03197668467778096,-0.14
	6.0	16.0	[62513.0,58.0,716.0,6.0,10.0,6.0,16.0]	[0.5856678333142841,0.08067142563224783,0.22008226607052694,0.7691620338743943,1.1959280069490403,0.2162
	6.0	34.0	[55635.0,54.0,590.0,7.0,7.0,6.0,34.0]	[0.27610986933138043,-0.23507479942300175,0.006861880539081063,1.1495628223666217,0.32296451524559583,0.

Role in Workflow

Scaling is critical in clustering because **K-Means uses Euclidean distance**. Without scaling, large-range variables (like Income) would overshadow smaller ones (like Recency). Standardization ensures that all features have equal weight in cluster formation, making results more balanced and interpretable.

Node 8 - Vector Assembler



18 Vector Assembler

▾ in0_18 (9 Cols)

Age	double
Income	double
Total_Spending	double
NumWebPurchases	double
NumStorePurchases	double
NumWebVisitsMonth	double
Recency	double
vector_feature	vectorudt
vector_features	vectorudt

Output Storage Level : DEFAULT

Input Columns : × vector_features : vectorudt

Output Column * : new_vector_feature

Handle Invalid : error

Purpose
Reassemble the **scaled feature vector** (**vector_features**) into a new column (**new_vector_feature**) that can be used by the **K-Means node**. This step is mainly a Sparkflows requirement, sometimes the Standard Scaler's output format is not directly compatible with modeling nodes, so re-assembling ensures the K-Means algorithm receives the correct vector input.

Configuration

- **Output Storage Level: DEFAULT**
- **Input Column:** vector_features (the standardized vector from the Standard Scaler)
- **Output Column: new_vector_feature**
- **Handle Invalid:** error (workflow stops if any invalid values are present)

Transformation Logic

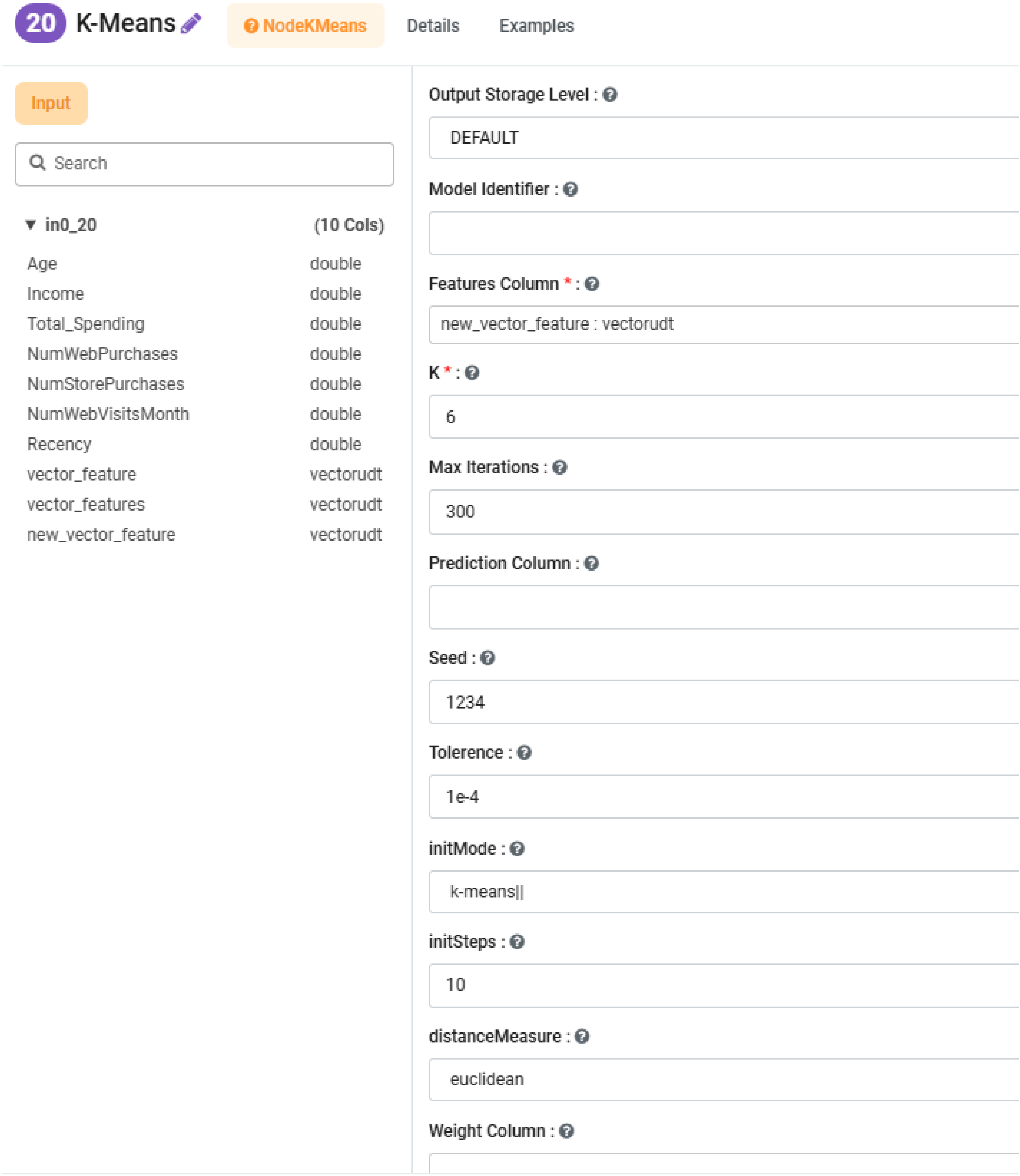
1. Takes the standardized feature vector (**vector_features**).
2. Wraps it into a new column **`new_vector_feature`**.
3. Ensures type compatibility with Sparkflows' K-Means implementation.

Output Schema

- Adds one new column: **new_vector_feature** (Vector type).
- Keeps original raw features and both previous vectors (**vector_feature, vector_features**) in the dataset.

Role in Workflow
This is essentially a **technical adjustment step**. While in pure Spark ML pipelines you usually pass the **`vector_features`** directly into K-Means, in Sparkflows it's sometimes necessary to run another Vector Assembler to avoid schema or type mismatches. By doing this, the dataset is guaranteed to be in the right format for clustering.

Node 9 - K-Means



in0_20	(10 Cols)
Age	double
Income	double
Total_Spending	double
NumWebPurchases	double
NumStorePurchases	double
NumWebVisitsMonth	double
Recency	double
vector_feature	vectorudt
vector_features	vectorudt
new_vector_feature	vectorudt

Output Storage Level : DEFAULT

Model Identifier :

Features Column * : new_vector_feature : vectorudt

K * : 6

Max Iterations : 300

Prediction Column :

Seed : 1234

Tolerence : 1e-4

initMode : k-means||

initSteps : 10

distanceMeasure : euclidean

Weight Column :

Purpose
Train a **K-Means clustering model** on the standardized feature vectors. This is the core modeling step of the workflow, where customers are grouped into clusters based on similarity across spending behavior, demographics, and recency.

Configuration

- **Output Storage Level: DEFAULT**
- **Features Column: new_vector_feature** (from the second Vector Assembler)
- **K (number of clusters): 6**
- **Max Iterations:** `300` (allows convergence even if clusters are complex)
- **Seed: 1234** (ensures reproducibility of results)
- **Tolerance: `1e-4`** (stopping criterion for convergence)
- **Init Mode: k-means||** (parallelized initialization, more efficient and stable than random)
- **Init Steps: 10** (number of initialization steps for better centroid placement)
- **Distance Measure: euclidean** (default, measures straight-line distance in feature space)
- **Prediction Column:** *(left blank)* → cluster assignments will instead be generated in the **Spark Predict node**.

Transformation Logic

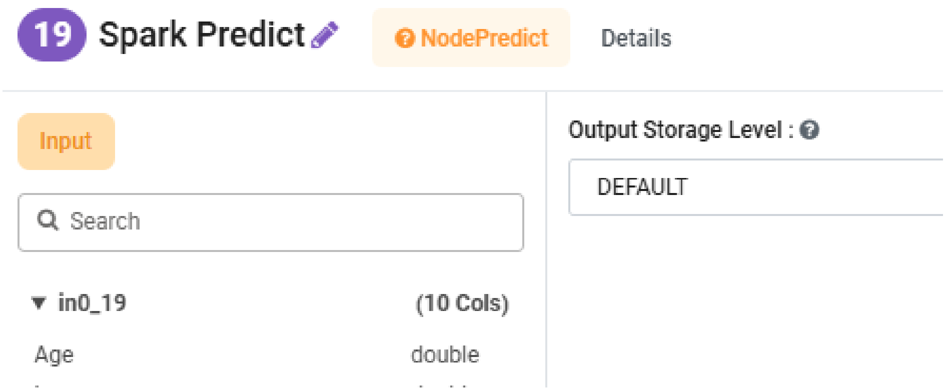
1. Reads the standardized feature vectors.
2. Uses **K-Means++ initialization** (via **`k-means||`**) to place initial centroids smartly.
3. Iteratively assigns points to the nearest centroid and recomputes centroids until convergence.
4. Produces a trained clustering model with **6 customer segments**.

Output Schema

- No new columns are added at this stage since the prediction column is left blank.
- Instead, the trained model object is stored, ready to be applied in the **Spark Predict node**.

Role in Workflow
This step is the **core machine learning model training**. It defines the cluster centers and learns the structure of customer groups. However, actual **cluster assignments** (predictions) are not created here, they are applied downstream in Spark Predict, keeping model training and scoring clearly separated.

Node 10 - Spark Predict



19 Spark Predict

NodePredict

Details

Input

Search

in0_19 (10 Cols)

Age double

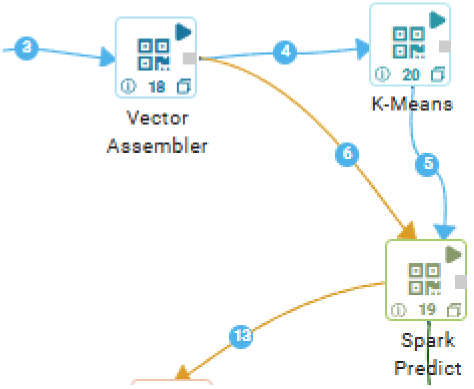
Output Storage Level :

DEFAULT

Purpose

Apply the **trained K-Means model** to the dataset and generate **cluster assignments** for every customer. This turns the unsupervised model into a labeled output by adding a **prediction** column (cluster ID).

Connections (as shown in the screenshot below)



- **Left/top input: `new_vector_feature`** from **Vector Assembler** → the feature vector to score.
- **Right/top input: K-Means model** → the trained clustering model.
  Sparkflows requires **both** the dataset (features) and the **model** to be connected into Spark Predict.

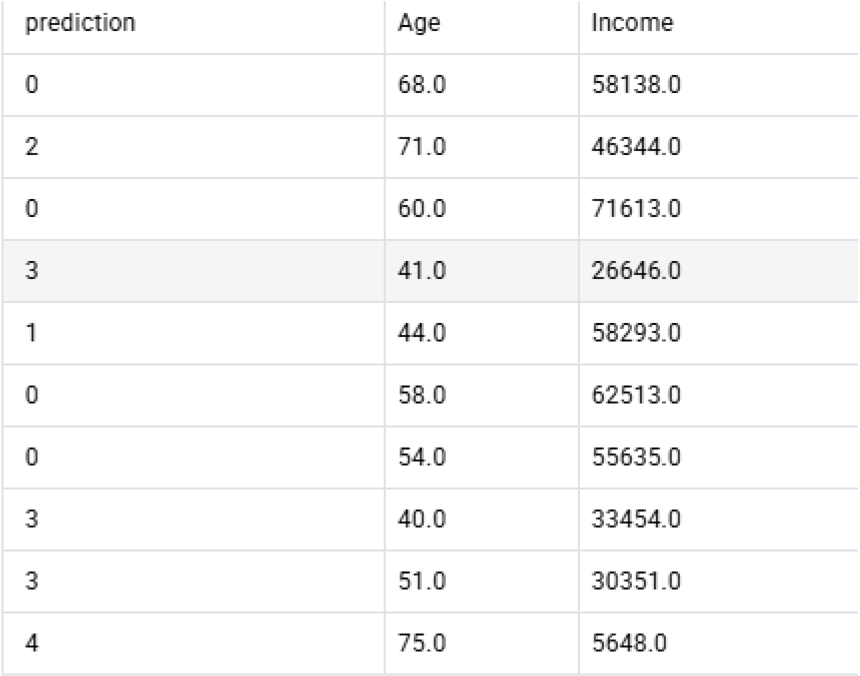
Configuration

- **Output Storage Level: DEFAULT**
- No extra params needed if the model and feature column names match what the model was trained with (**new_vector_feature**).

What it does

1. Reads the standardized feature vector (**new_vector_feature**) for each row.
2. Uses the trained K-Means centroids to find the **nearest cluster**.
3. Appends a new integer column **prediction** with values.

Output Schema

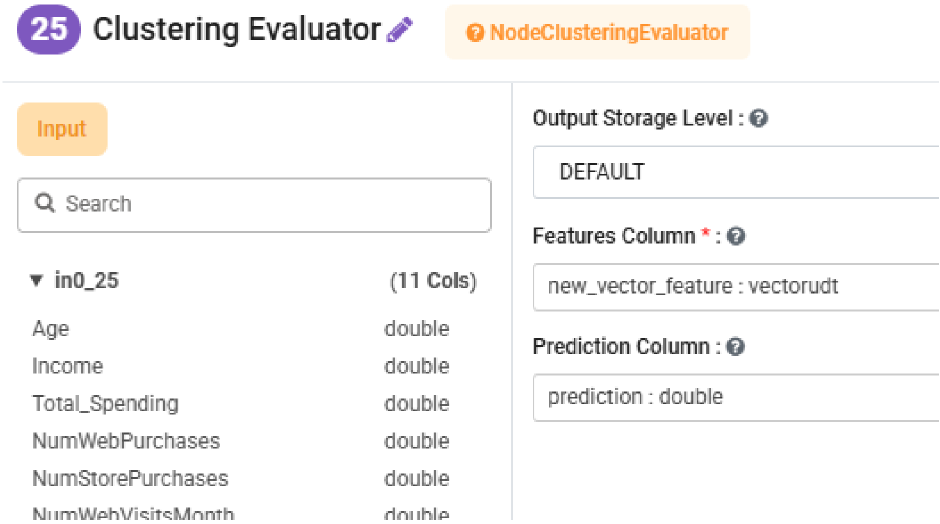


prediction	Age	Income
0	68.0	58138.0
2	71.0	46344.0
0	60.0	71613.0
3	41.0	26646.0
1	44.0	58293.0
0	58.0	62513.0
0	54.0	55635.0
3	40.0	33454.0
3	51.0	30351.0
4	75.0	5648.0

- All original columns **plus**:
  - **prediction** → the assigned **cluster ID** for each customer.

Role in Workflow
This is the **scoring step** that converts modeling artifacts into usable business output.

Node 11 - Clustering Evaluator



25 Clustering Evaluator

NodeClusteringEvaluator

▾ in0_25 (11 Cols)

Age	double
Income	double
Total_Spending	double
NumWebPurchases	double
NumStorePurchases	double
NumWebVisitsMonth	double

Output Storage Level : DEFAULT

Features Column * : new_vector_feature : vectorudt

Prediction Column : prediction : double

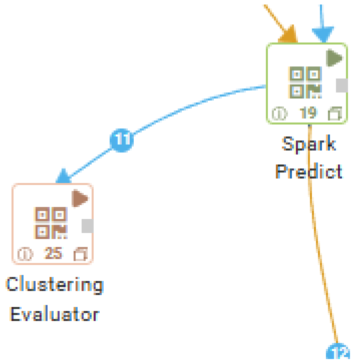
Purpose

Quantitatively evaluate the **quality of the K-Means clustering** by computing the **Silhouette Score** using the standardized feature vector and the predicted cluster labels.

Configuration

- **Output Storage Level: DEFAULT**
- **Features Column: new_vector_feature** (standardized vector used to fit K-Means)
- **Prediction Column: prediction** (from Spark Predict: cluster ID per row)

Connection

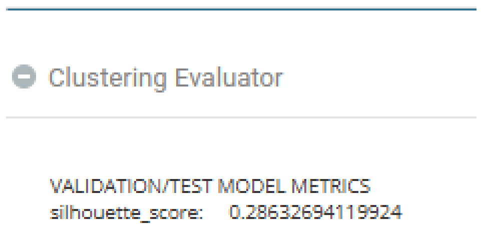


What it computes (Silhouette Score)

- **Definition:** For each sample *i*, Silhouette compares **cohesion** (how close *i* is to its own cluster) vs **separation** (how far *i* is from the nearest other cluster).
- **Range: −1** to **+1**

  - **~1.0:** very dense, well-separated clusters
  - **~0.5:** reasonable structure
  - **~0.0:** overlapping clusters / unclear structure
  - **< 0:** likely misassigned

Result



```
VALIDATION/TEST MODEL METRICS
silhouette_score:    0.28632694119924
```

- **silhouette_score: 0.2865**
  - Interpretation: **weak-to-moderate** cluster structure. Clusters are somewhat separable but there is overlap.
  - This can still be acceptable in **real customer data**, where behavior often blends; business interpretability matters alongside the metric.

Role in Workflow

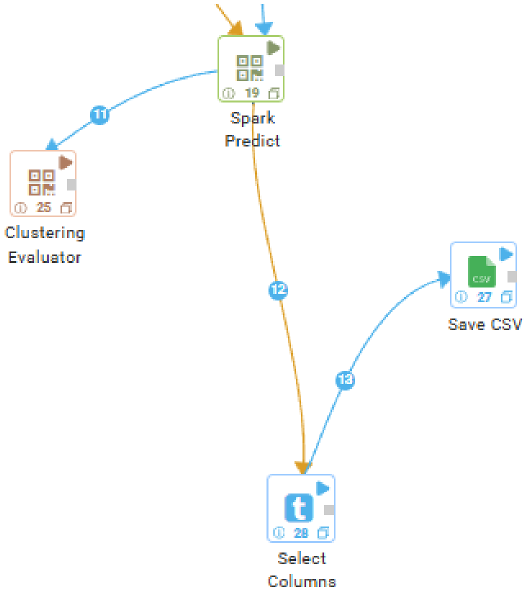
- Provides an **objective metric** to compare different **K** values (e.g., K=4,5,6).
- Helps validate preprocessing choices (scaling, feature set).
- Guides whether to iterate (tune K, try different features) before finalizing segments.

Notes / Good Practices

- **Don’t chase the max silhouette blindly.** Choose **K** that balances:
  - Interpretability of segments (clear business personas)
  - **Cluster size balance** (avoid tiny or huge clusters)
  - **Stability** across different seeds / samples
- Try quick iterations:
  - Re-run with **K** ∈ **{4,5,6,7}** and compare silhouette + business profiles.

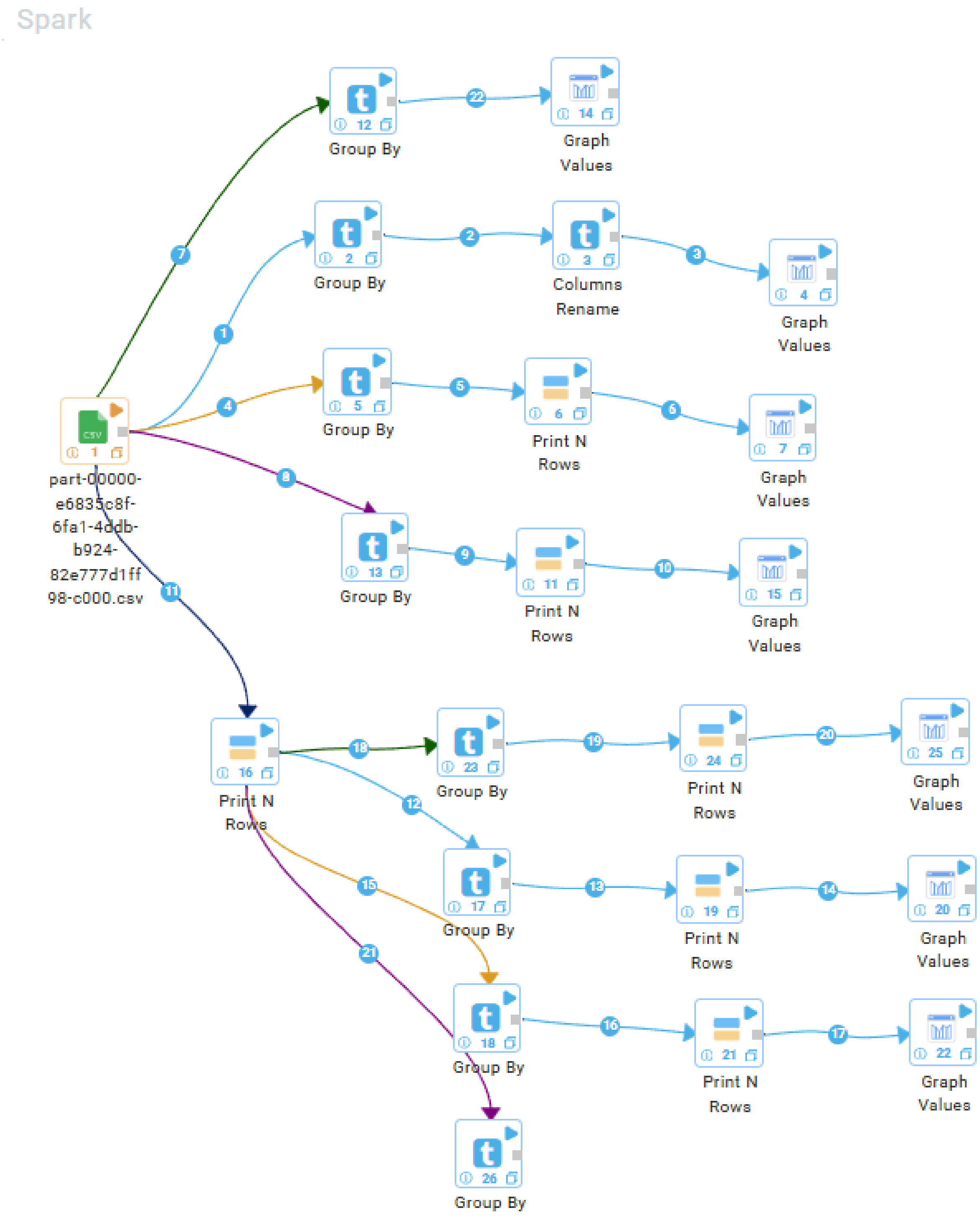
- ○ Revisit features (e.g., include/exclude **Recency**, split **Total_Spending** by category, or add tenure buckets).
- ○ Check **elbow/WCSS** alongside silhouette for a fuller picture.

Node 12 and 13 - Select Columns and Save CSV

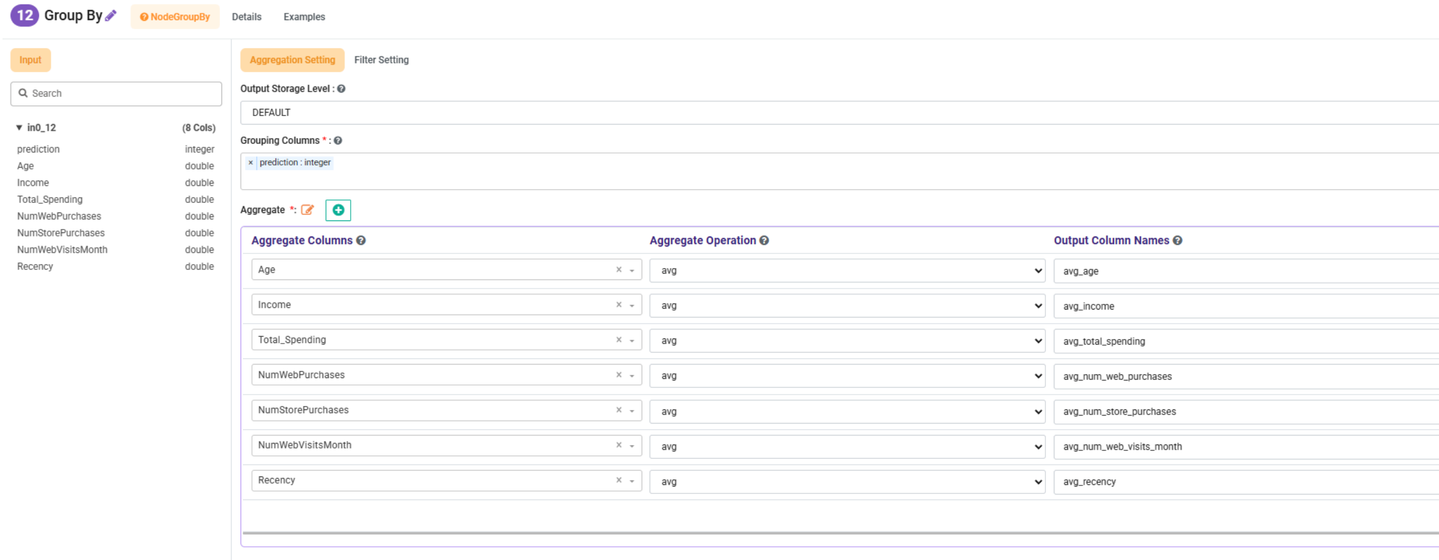


After **Spark Predict**, we pipe its output into a **Select Columns** node to keep only the fields we want in the deliverable (e.g., **ID** [optional], **Age**, **Income**, **Total_Spending**, **NumWebPurchases**, **NumStorePurchases**, **NumWebVisitsMonth**, **Recency**, and the **new prediction cluster label**). This trims intermediate vectors (**vector_feature**, **vector_features**, **new_vector_feature**) and any helper columns so the file is clean and business-friendly. That **Select Columns** output feeds directly into **Save CSV**, which writes the final, clustered dataset (**one row per customer + assigned prediction**) to the chosen path. This keeps training and scoring separate, produces a reproducible artifact for analysis/visuals, and makes it easy to compute per-cluster profiles in a notebook or BI tool.

WORKFLOW 3 - Cluster Analysis



Node 1 - Group By



12 Group By

NodeGroupBy Details Examples

Input

Search

in0_12 (8 Cols)

prediction	integer
Age	double
Income	double
Total_Spending	double
NumWebPurchases	double
NumStorePurchases	double
NumWebVisitsMonth	double
Recency	double

Aggregation Setting Filter Setting

Output Storage Level :

DEFAULT

Grouping Columns * :

× prediction : integer

Aggregate *:

Aggregate Columns	Aggregate Operation	Output Column Names
Age	avg	avg_age
Income	avg	avg_income
Total_Spending	avg	avg_total_spending
NumWebPurchases	avg	avg_num_web_purchases
NumStorePurchases	avg	avg_num_store_purchases
NumWebVisitsMonth	avg	avg_num_web_visits_month
Recency	avg	avg_recency

Purpose

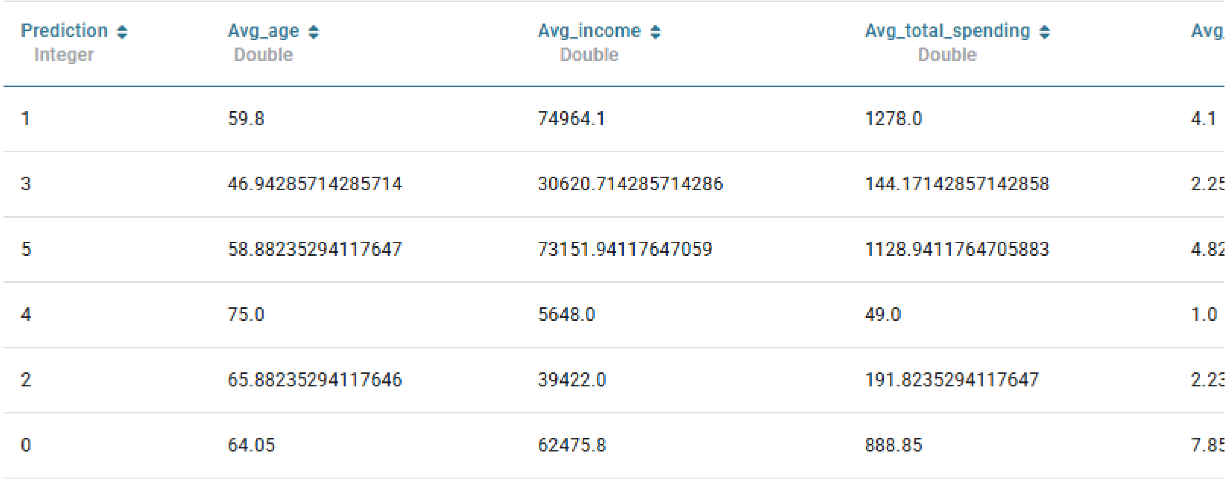
Compute **per-cluster averages** for the key features so we can profile segments and visualize differences across clusters.

Configuration

- **Grouping Columns: prediction** (cluster ID)
- **Aggregations:**
    - **avg(Age) → avg_age**
    - **avg(Income) → avg_income**
    - **avg(Total_Spending) → avg_total_spending**
    - **avg(NumWebPurchases) → avg_num_web_purchases**
    - **avg(NumStorePurchases)→ avg_num_store_purchases**
    - **avg(NumWebVisitsMonth)→ avg_num_web_visits_month**
    - **avg(Recency) → avg_recency**

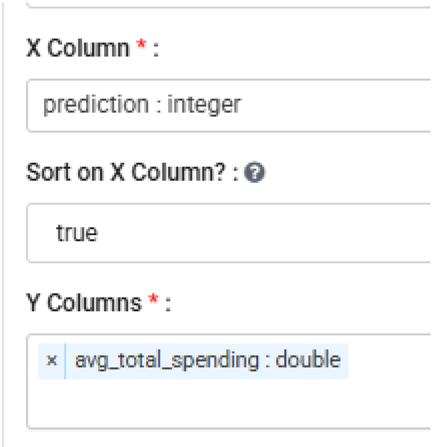
What it produces

One row per cluster (**prediction**) with the above KPI columns. This table is the basis for plots and the “cluster personas” summary.



Prediction	Avg_age	Avg_income	Avg_total_spending	Avg
Integer	Double	Double	Double	
1	59.8	74964.1	1278.0	4.1
3	46.94285714285714	30620.714285714286	144.17142857142858	2.25
5	58.88235294117647	73151.94117647059	1128.9411764705883	4.82
4	75.0	5648.0	49.0	1.0
2	65.88235294117646	39422.0	191.8235294117647	2.23
0	64.05	62475.8	888.85	7.85

Node 2 - Graph Values



X Column * :

prediction : integer

Sort on X Column? :

true

Y Columns * :

× avg_total_spending : double

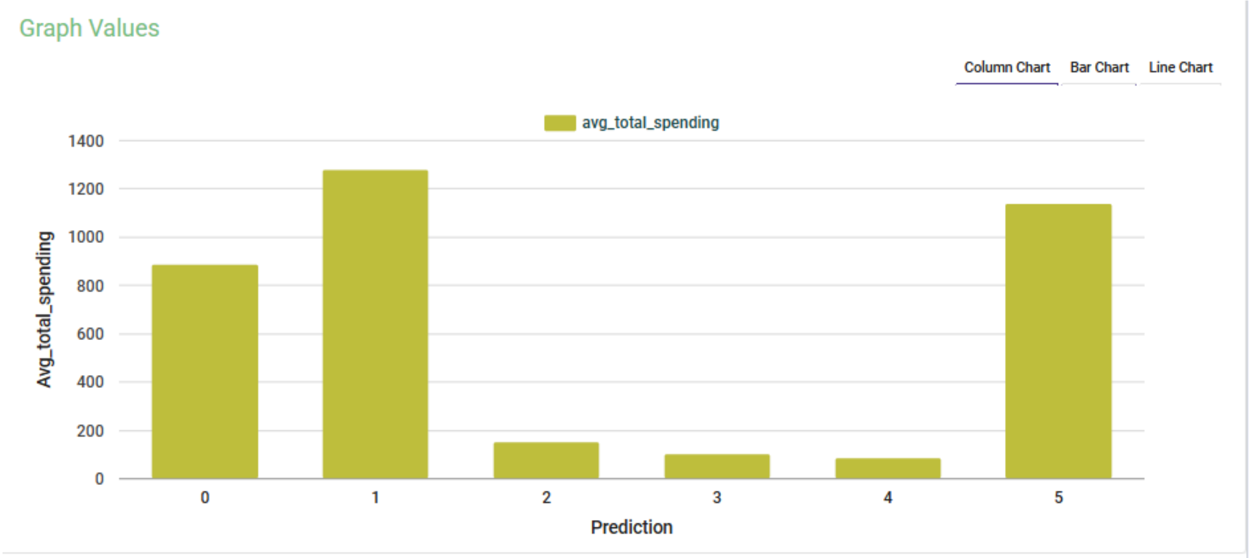
Purpose

Visualize **per-cluster KPIs** from the Group By output so differences between segments are easy to compare at a glance.

Configuration

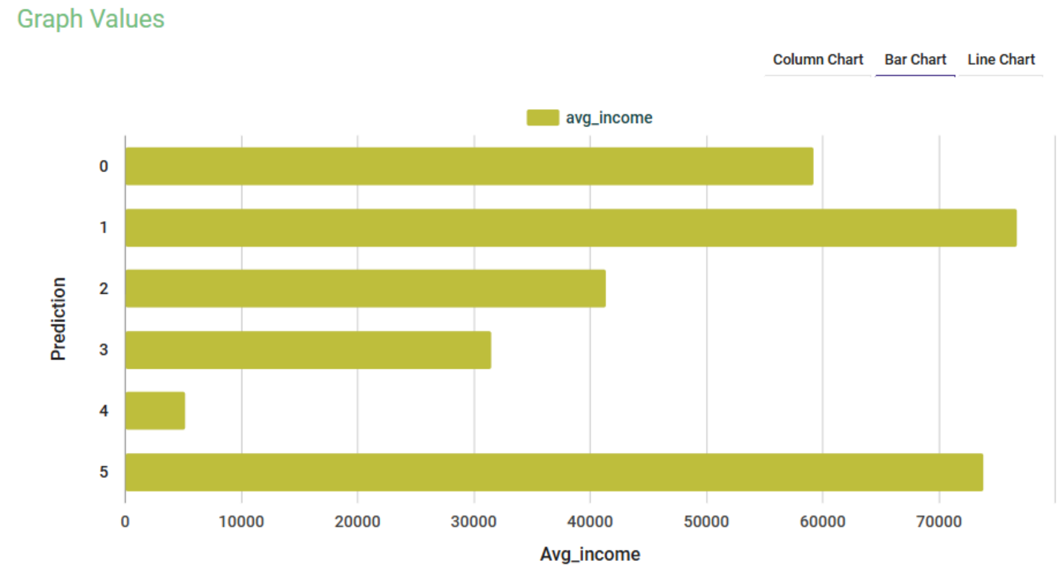
- **X Column: prediction** (cluster ID)
- **Sort on X Column: true** (keeps clusters ordered)
- **Y Columns: avg_total_spending**
- (Chart type: use **Bar**; one bar per cluster)

What it produces
A bar chart of **Average Total Spending by Cluster**. This immediately highlights high-value vs low-value segments and helps name personas (e.g., “Premium Spenders”, “Budget Segment”).

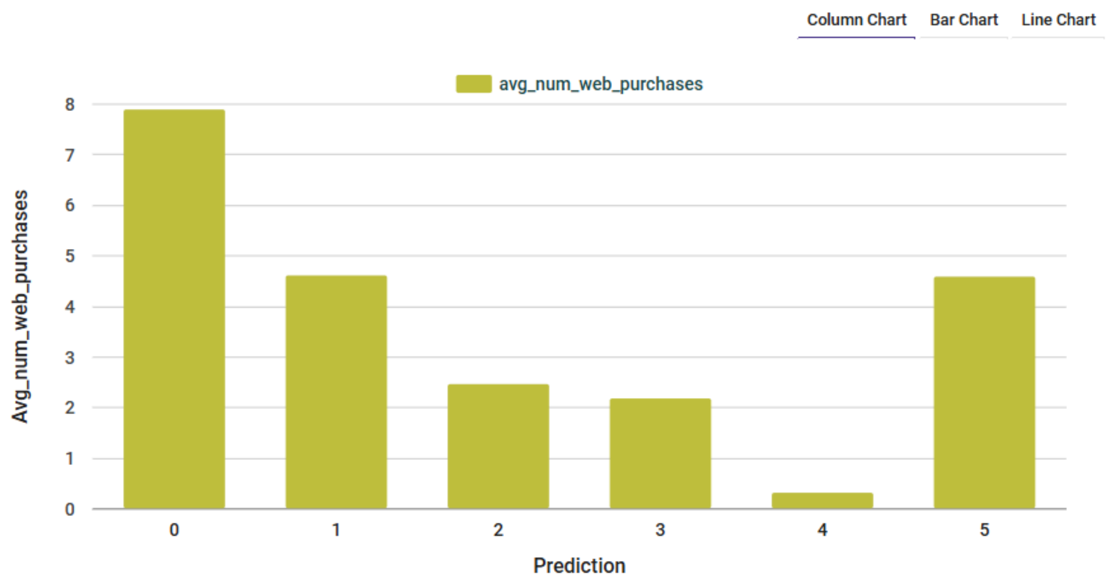


We used the same two-node pattern, **Group By** (group on **prediction**, compute **avg(...)**) → **Graph Values**, for each feature. Only the **Y column** in Graph Values changes per chart (e.g., **avg_total_spending, avg_income**, **avg_num_web_purchases, avg_num_store_purchases, avg_num_web_visits_month, avg_age, avg_recency**). The X column is always **prediction**, sorted ascending. The resulting visuals below compare clusters on each KPI.

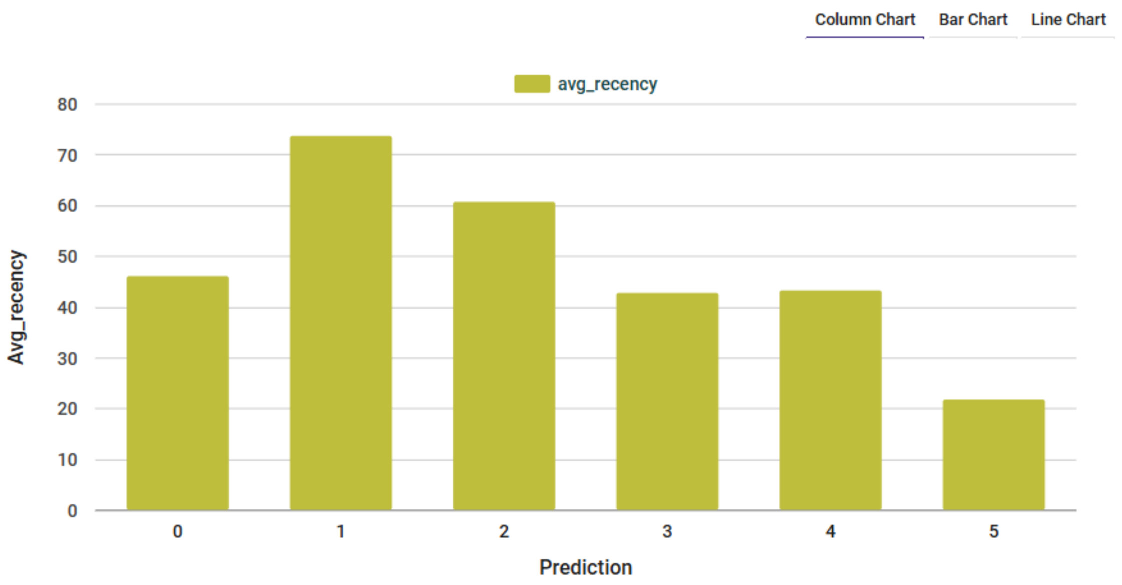
Here are those visuals:



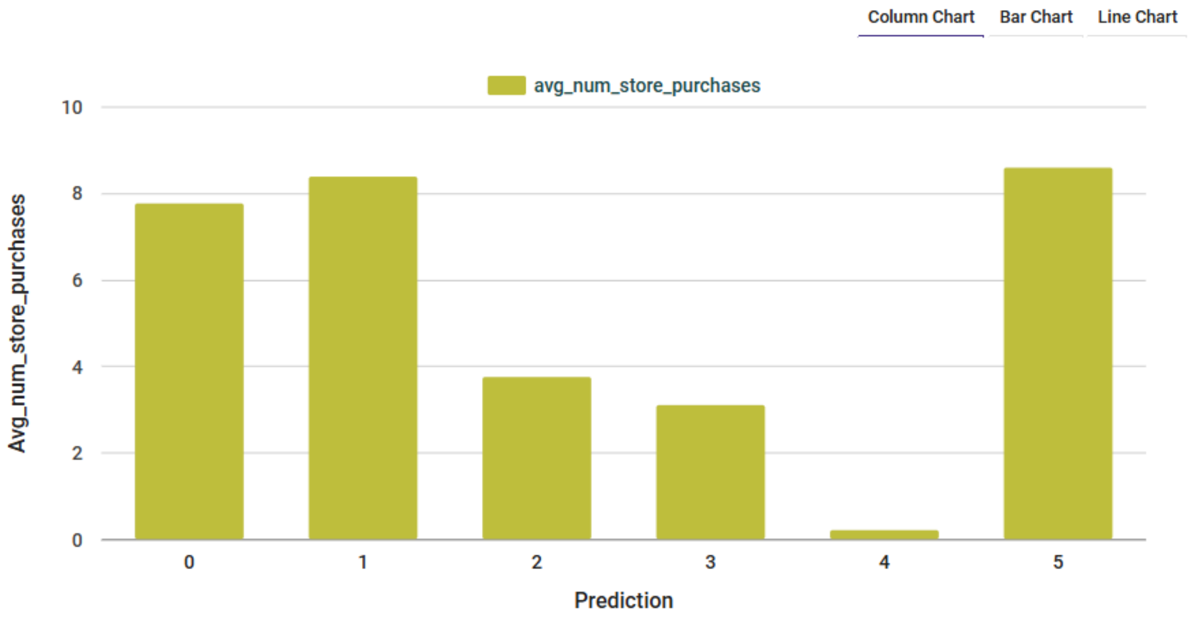
Graph Values



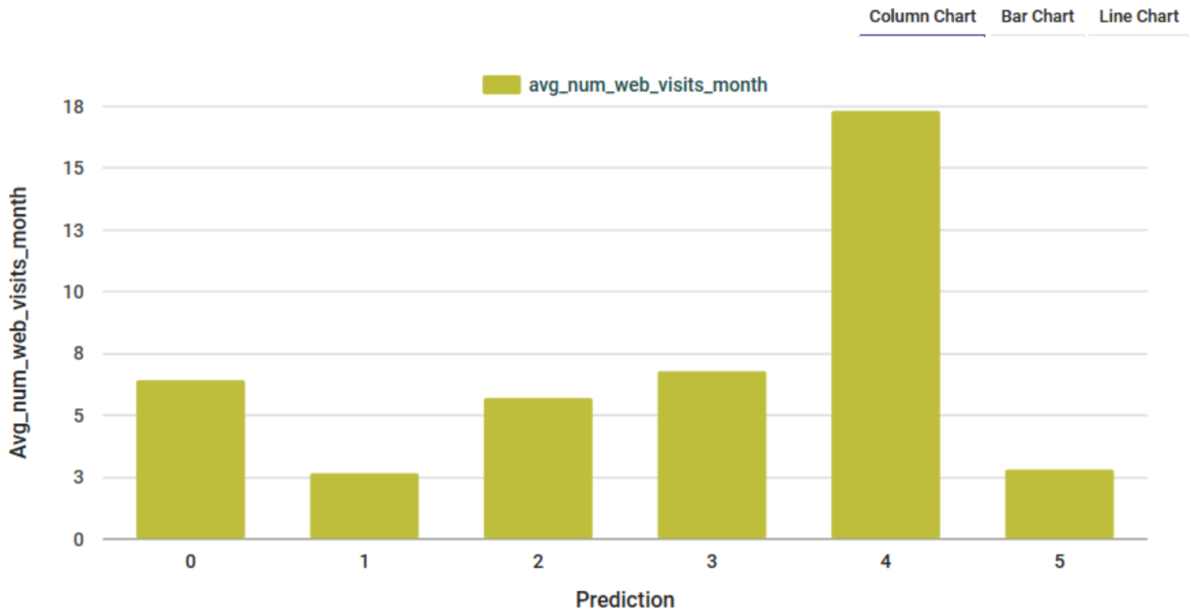
Graph Values



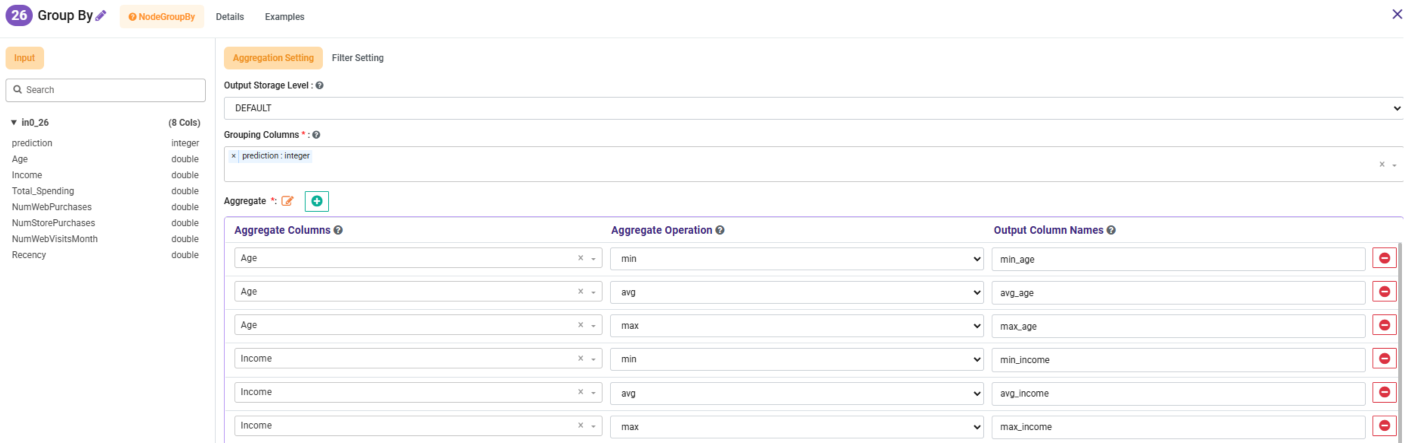
Graph Values



Graph Values



Node 3 - Group By



▾ in0_26 (8 Cols)

prediction	integer
Age	double
Income	double
Total_Spending	double
NumWebPurchases	double
NumStorePurchases	double
NumWebVisitsMonth	double
Recency	double

Output Storage Level : DEFAULT

Grouping Columns * : prediction : integer

Aggregate *:

Aggregate Columns	Aggregate Operation	Output Column Names
Age	min	min_age
Age	avg	avg_age
Age	max	max_age
Income	min	min_income
Income	avg	avg_income
Income	max	max_income

Purpose

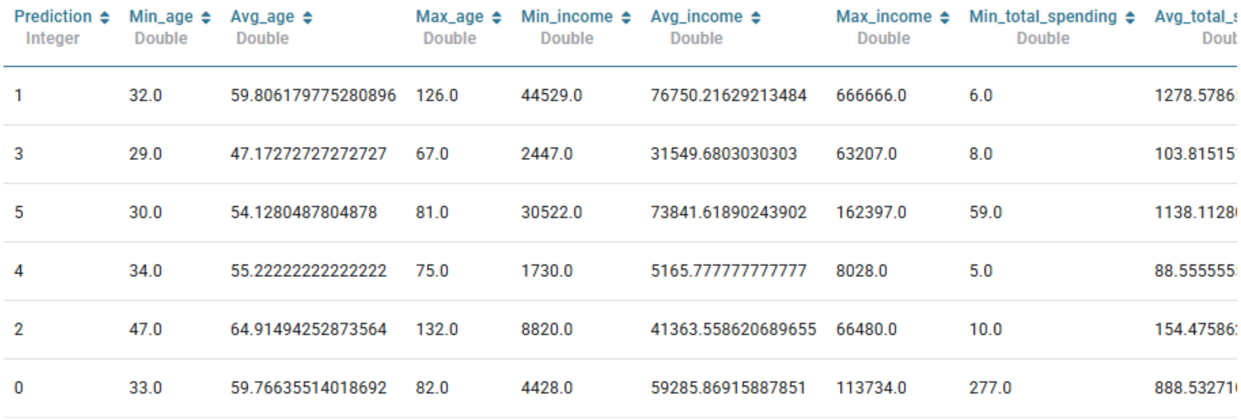
Produce a **summary table** for each cluster with **min / average / max** of all key features. This is the master table you'll use to write personas and business insights.

Configuration

- **Grouping Columns: prediction**
- **Aggregations (repeat pattern for every feature):**
  - **Age → min_age, avg_age, max_age**
  - **Income → min_income, avg_income, max_income**
  - **Total_Spending → min_total_spending, avg_total_spending, max_total_spending**
  - **NumWebPurchases → min_num_web_purchases, avg_num_web_purchases, max_num_web_purchases**
  - **NumStorePurchases → min_num_store_purchases, avg_num_store_purchases, max_num_store_purchases**
  - **NumWebVisitsMonth → min_num_web_visits_month, avg_num_web_visits_month, max_num_web_visits_month**
  - **Recency → min_recency, avg_recency, max_recency *(note: lower = more recent)***

What it produces

One row per **cluster (`prediction`)** with 21 KPI columns (3 stats × 7 features). This table is perfect for a single, dense "Cluster Profiles" view.



Prediction ⬍	Min_age ⬍	Avg_age ⬍	Max_age ⬍	Min_income ⬍	Avg_income ⬍	Max_income ⬍	Min_total_spending ⬍	Avg_total_
Integer	Double	Double	Double	Double	Double	Double	Double	Dout
1	32.0	59.806179775280896	126.0	44529.0	76750.21629213484	666666.0	6.0	1278.5786
3	29.0	47.17272727272727	67.0	2447.0	31549.6803030303	63207.0	8.0	103.81515
5	30.0	54.1280487804878	81.0	30522.0	73841.61890243902	162397.0	59.0	1138.1128
4	34.0	55.22222222222222	75.0	1730.0	5165.777777777777	8028.0	5.0	88.555555
2	47.0	64.91494252873564	132.0	8820.0	41363.558620689655	66480.0	10.0	154.47586
0	33.0	59.76635514018692	82.0	4428.0	59285.86915887851	113734.0	277.0	888.53271

Role in the workflow

- Drives **persona naming** (e.g., high income + high spend + recent = “Premium Active”).
- Enables **actionable insights** (e.g., high web visits but low orders → target conversion).

Segment personas & actions

Cluster 1 – High-Income Lapsed Spenders

Profile: Highest income (~76.8k), **highest spend** (~1,279), **store-leaning** (web_share ~35%), but **least recent** (recency ~74).
Opportunity: Win-back valuable customers before churn.
Actions:

- **Reactivation series:** limited-time coupons, “we miss you” + personalized recommendations.
- **Store-first incentives:** VIP in-store events, concierge service, exclusive bundles.
- **Trigger on inactivity:** if recency >60 days, escalate offer/touchpoint.

Cluster 5 – VIP Store Loyalists (Active Now)

Profile: High income (~73.8k), **very high spend** (~1,138), **store-heavy** (web_share ~35%), **most recent** (recency ~22).
Opportunity: Protect and grow LTV.
Actions:

- **Tiered VIP perks:** early access, priority checkout, invite-only drops.
- **High-AOV bundles** in store; add attach-items at POS.

- **White-glove follow-ups:** thank-you messages, concierge recommendations.

Cluster 0 – Digital Power Users (Balanced Buyers)

Profile: Upper-mid income (~59.3k), **mid-high spend** (~889), **highest web orders** (web_share ~50%), frequent web visits (~6.45), recent-ish (recency ~46).
Opportunity: Scalable **online** revenue growth.
Actions:

- **Personalized web promos** (A/B test banners/email) targeting categories tied to their browsing.
- **Cart/build-a-bundle** nudges and free shipping thresholds to lift **spend per order** (~57).
- **Loyalty multipliers** for web purchases to entrench digital behavior.

Cluster 3 – Mass Active Value Seekers *(largest segment)*

Profile: Lower income (~31.5k), **modest spend** (~104), **store-leaning** (web_share ~35%), **very recent** (recency ~22), orders per customer comparable to C1/C5.
Opportunity: Efficient **cross-sell** and basket-size growth at scale.
Actions:

- **Staple + accessory bundles** at entry price points.
- **POS cross-sell scripts** + coupons for next visit within 14–21 days.
- **Always-on promos**: "Buy X get Y," price-matched sets to keep them returning.

Cluster 2 – Low-Spend Occasional Shoppers

Profile: Mid-low income (~41.4k), **low spend** (~154), low orders (~5.3), moderate web visits (~6.8), recency ~43.
Opportunity: Nudge toward first/second repeat and raise conversion.
Actions:

- **First-timer/second-timer offers** (loyalty onboarding, small "welcome back" discounts).
- **Free returns / low-risk trials** to reduce friction.
- **Email/web personalization** that emphasizes value (price filters, bundles under X₺).

Cluster 4 – Heavy Browsers, Almost No Orders *(micro-segment; n=9)*

Profile: Very low income (~5.2k), **very high web visits** (~17.3) but **~0.56 orders**, inflated spend/order due to tiny denominator, recency ~43.

Interpretation: Likely **outliers** or a niche (bots, blocked payments, UX friction).
Actions:

- **Investigate individually:** payment declines, delivery constraints, UX bugs.
- If legit niche: **targeted trial offers** or cash-on-delivery where allowed.
- For reporting, **flag as unstable**; consider merging with nearest cluster in a K=5 test.

Channel & behavior insights (cross-cluster)

- **Store-heavy big spenders (C1, C5)** respond to **in-store experiences** and **bundles**; prioritize **retention (C5)** and **reactivation (C1)**.
- **Web-active segment (C0)** shows the best lever for **digital growth**; raise **spend/order** with bundles and thresholds.
- **C3's recency** is as good as C5's: they show up, but don't spend much → **entry bundles + cross-sell** can move the needle fast.
- **C2** needs **friction removal** and simple value messaging to convert browsing into orders.

Prioritized next steps

1. **Protect revenue:** launch VIP retention for **C5**; reactivation for **C1**.
2. **Scale growth:** digital A/B tests for **C0**; entry bundles for **C3**.
3. **Fix leaks:** diagnose **C4** (payment/UX); reduce friction for **C2** (returns, trials).
4. **Model iteration:** test **K=5** (merge C4), try **category-level spend** and **RFM** features, and compare silhouette + interpretability.