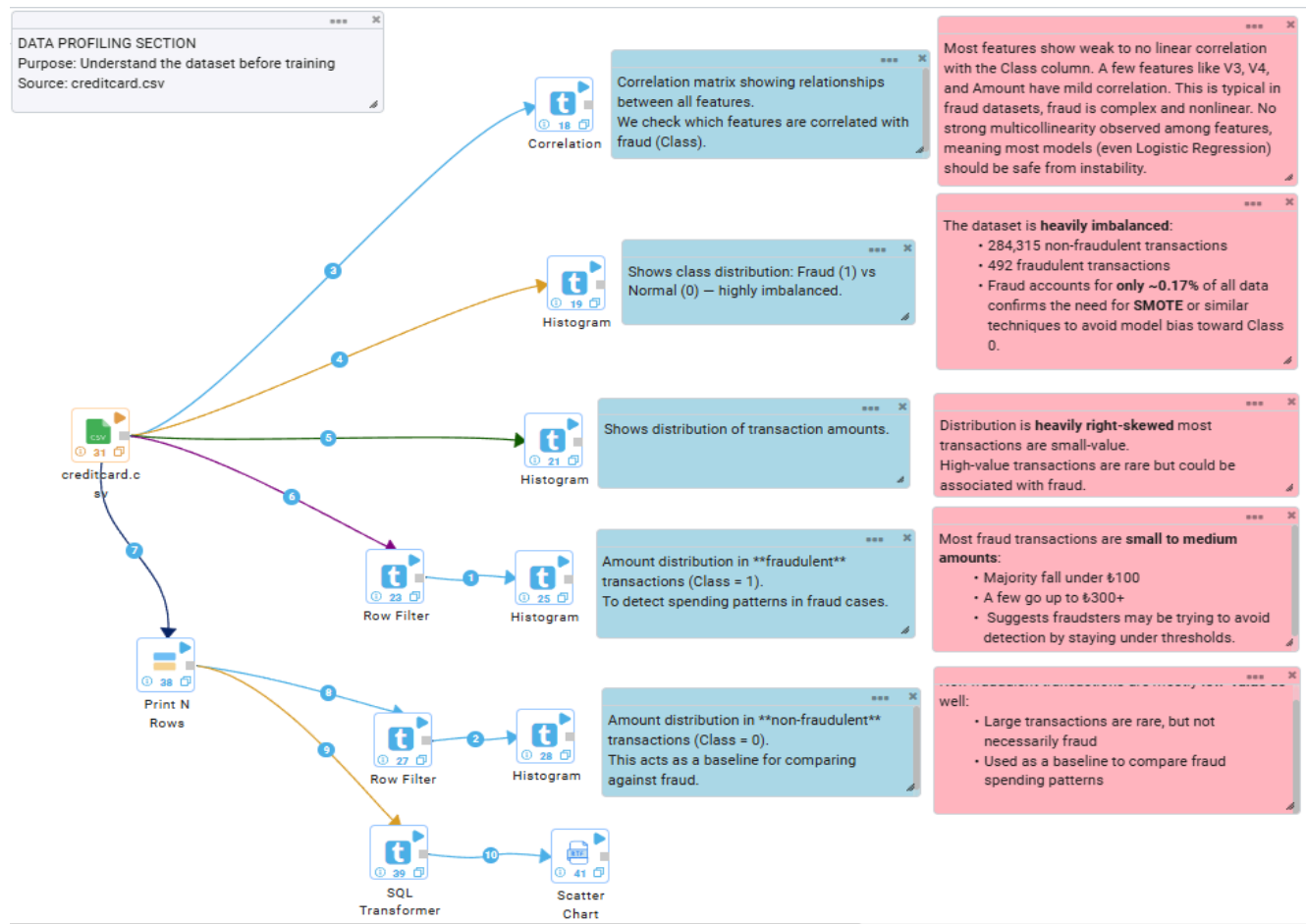


WORKFLOW 1 - DATA PROFILING



Node 1 - Read CSV (creditcard.csv)

42 Read CSV Details Examples **Note: Refresh the schema when a new file is selected or the file content has changed**

General InferSchema ExtraOptions

Output Storage Level :

DEFAULT

Path * :

/root/fraud_in_credit_transaction/creditcard.csv

Path Selection

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

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

42 Read CSV

NodeDatasetCSV

Details

Examples

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

General

InferSchema

ExtraOptions

Schema Columns :

InferSchema

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 - Correlation

18 Correlation

NodeCorrelation

Details

Examples

Input

▼ in0_18 (31 Cols)

Timeinteger

V1double

V2double

V3double

V4double

Output Storage Level :

DEFAULT

Title :

Correlation Matrix

Input Column for Correlation :

× Time : integer × V1 : double × V2 : double × V3 : double × V4 : double ×

× V17 : double × V18 : double × V19 : double × V20 : double × V21 : double

Purpose

The Correlation node computes the pairwise correlation between features and the target variable (Class). This helps identify which features are most strongly related to fraud detection.

General Settings

- **Output Storage Level:** Keep as DEFAULT unless you have memory optimization needs.
- **Title:** (Optional) You can give the chart a name, e.g. Correlation Matrix.

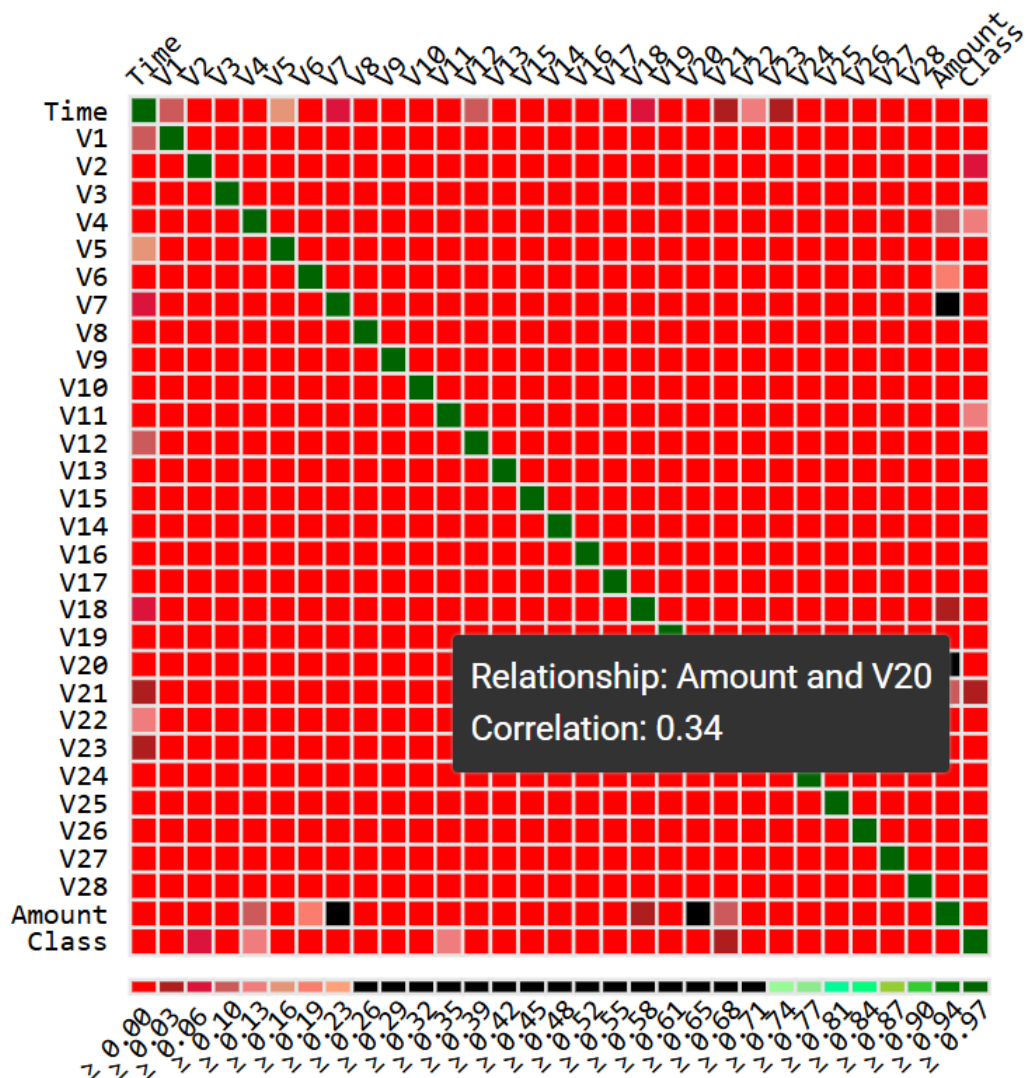
Input Column

- Select the numeric columns you want to include in the correlation analysis.
- Ensure the **Class** label is included so you can check correlation with the fraud outcome.

Execution

- When executed, Sparkflows generates a correlation matrix heatmap.
- This visualization helps to: Detect highly correlated variables (potential multicollinearity) and identify weak vs. strong predictors for fraud classification.

Output



You can move your mouse on it and see every correlation value.

Node 3 - Histogram

19

Histogram

NodeHistogramCal

Details

Examples

Input

Q Search

▼ in0_19

(31 Cols)

Time

integer

V1

double

V2

double

V3

double

V4

double

V5

double

V6

double

V7

double

V8

double

V9

double

V10

double

V11

double

V12

double

V13

double

V14

double

V15

double

V16

double

V17

double

V18

double

V19

double

V20

double

V21

double

V22

double

V23

double

V24

double

V25

double

V26

double

V27

double

V28

double

V29

double

V30

double

V31

double

Output Storage Level : ?

DEFAULT

Title : ?

Title Color :

#77C27F

Description :

Shows class distribution: Fraud (1) vs Normal (0) — highly imbalanced.

Description Color :

#808080

Chart Colors :

Column Name : ?

Class : integer

Number of Bins : ?

2

Purpose

The Histogram node is used here to **visualize the class distribution** between Fraud (1) and Non-Fraud (0). This is an essential first step in fraud detection projects because the dataset is highly imbalanced.

General Settings

- **Output Storage Level:** Keep as DEFAULT unless you have memory optimization needs.
- **Title:** (Optional) You can add a title like Fraud vs Non-Fraud Distribution.
- **Description:** Add a note such as: "Shows class distribution: Fraud (1) vs Normal (0) — highly imbalanced."

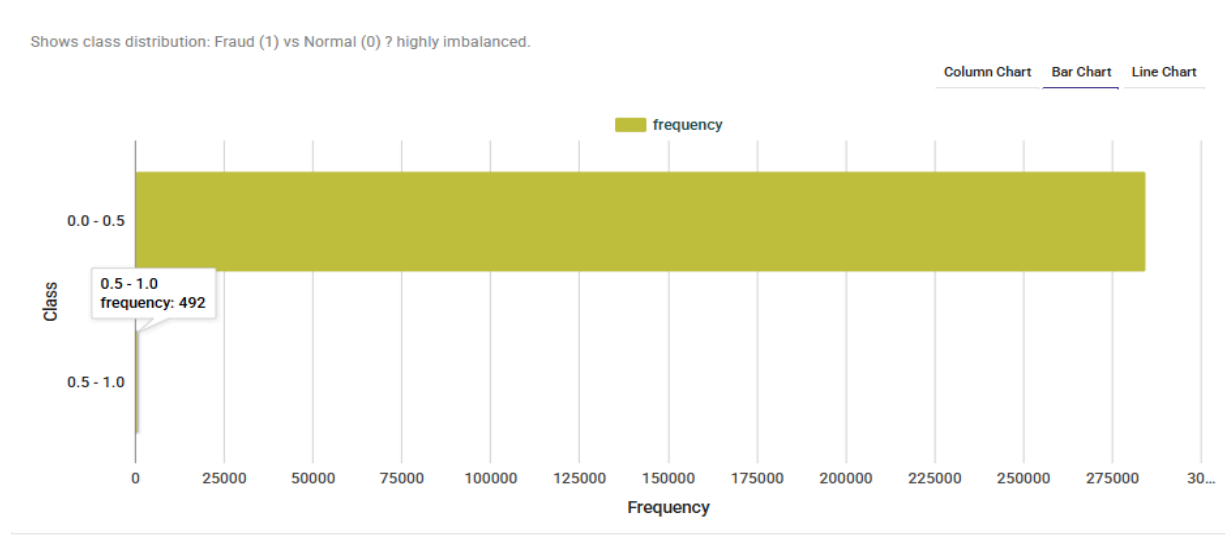
- **Title & Description Colors:** Customizable for better readability in reports.

Chart Settings

- **Column Name:** Set to Class (integer) → this is the fraud label column.
- **Number of Bins:** Set to 2 → one bin for Fraud (1), one bin for Non-Fraud (0).
- **Chart Colors:** You can customize to clearly differentiate the two classes.

Execution & Output

- Running this node generates a **bar chart** with two bars:
 Class 0 → representing legitimate transactions (majority).
 Class 1 → representing fraudulent transactions (minority).
- This confirms the imbalance problem (~0.17% fraud cases), which justifies using oversampling methods like SMOTE.



● Distribution of continuous values of class, min value is: 0.0 & max value is: 1.0




Distribution of continuous values of class, min value is: 0.0 & max value is: 1.0

■ Draggable

Class_range ↕	Count ↕
0.0 - 0.5	284315
0.5 - 1.0	492

Node 3 - Histogram

21 Histogram 

NodeHistogramCal Details Examples

Input

▼ in0_21 (31 Cols)

Time	integer
V1	double
V2	double
V3	double
V4	double
V5	double
V6	double
V7	double
V8	double
V9	double
V10	double
V11	double
V12	double
V13	double
V14	double
V15	double
V16	double
V17	double
V18	double

Output Storage Level : ?

DEFAULT

Title : ?

Title Color :

#77C27F

Description :

Description Color :

#808080

Chart Colors :

Column Name : ?

Amount : double

Number of Bins : ?

50

Purpose

This Histogram node is used to **visualize the distribution of transaction amounts** in the dataset. It helps us understand the spending patterns and detect outliers (e.g., extremely high-value transactions that might correlate with fraud).

General Settings

- **Output Storage Level:** Keep as DEFAULT unless you have memory optimization needs.
- **Title:** (Optional) (Optional) You can add something like Distribution of Transaction Amounts.
- **Description:** Add a note such as: "Shows how transaction amounts are distributed across the dataset. Most transactions are low-value, while a few are significantly higher."
- **Title & Description Colors:** Customizable for styling in reports.

Chart Settings

- **Column Name:** Set to Amount (double) → this is the transaction amount column.
- **Number of Bins:** Set to 50 → provides enough granularity to capture both small and large transactions.
- **Chart Colors:** Customize to differentiate better when presenting results.

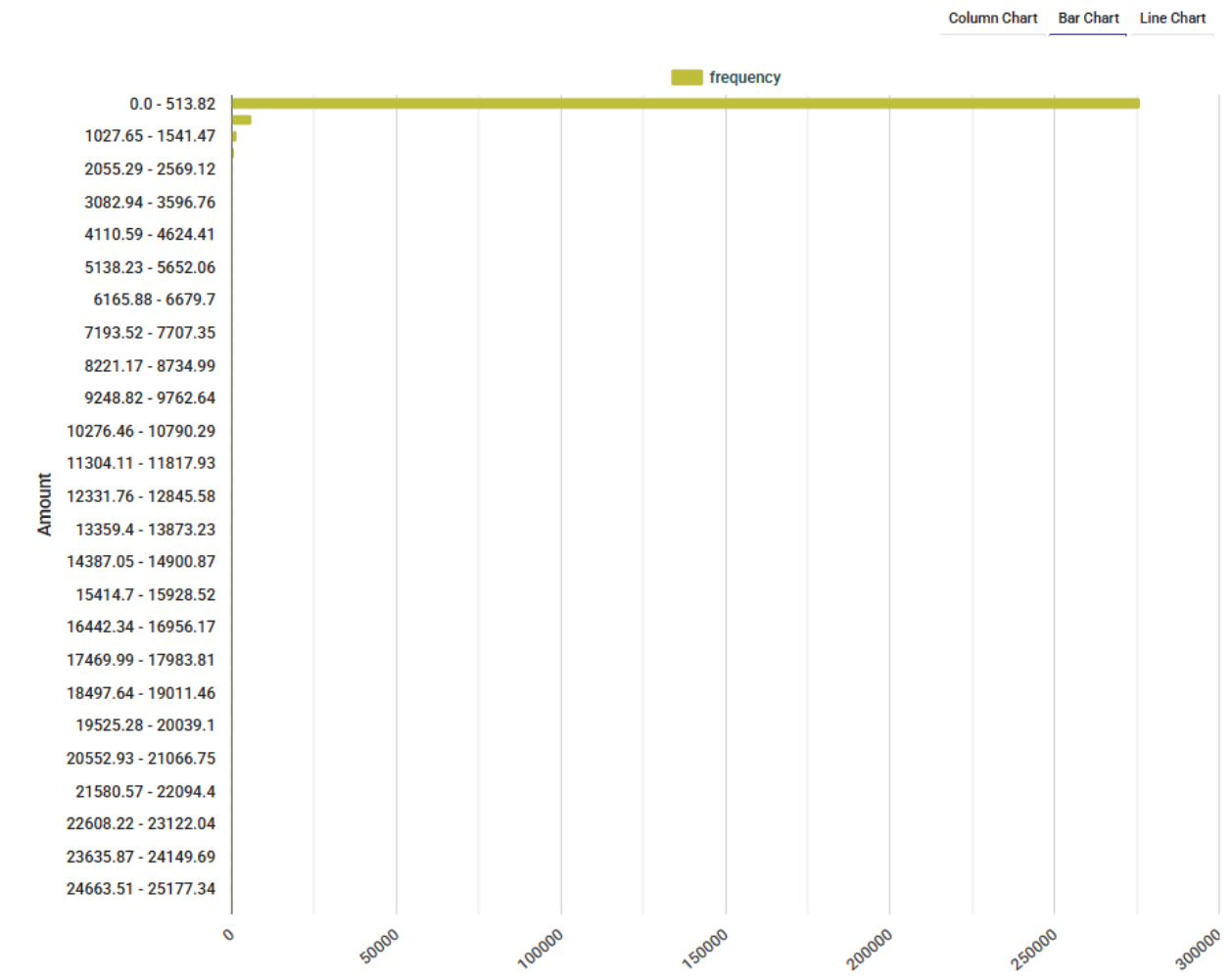
Execution & Output

- Running this node generates a **histogram** showing the frequency of transaction amounts.
- This imbalance in amount values is important because fraudulent cases often cluster at unusual transaction amounts.

Distribution of continuous values of amount, min value is: 0.0 & max value is: 25691.16

Draggable

Amount_range ↕	Count ↕
0.0 - 513.82	275992
513.82 - 1027.65	6010
1027.65 - 1541.47	1539
1541.47 - 2055.29	627
2055.29 - 2569.12	227
2569.12 - 3082.94	144
3082.94 - 3596.76	92
3596.76 - 4110.59	69
4110.59 - 4624.41	33
4624.41 - 5138.23	22
5138.23 - 5652.06	10
5652.06 - 6165.88	11
6165.88 - 6679.7	5



You can select the chart type from the top right corner.

Node 4 - Row Filter

23

Row Filter

NodeRowFilter

Details

Examples

Input

Search

▼ in0_23

(31 Cols)

Time

integer

V1

double

V2

double

V3

double

V4

double

V5

double

V6

double

Output Storage Level : ?

DEFAULT

Conditional Expression * : ?

```
1 Class == 1
```

Purpose

This node filters the dataset to only include fraudulent transactions.

Configuration

- **Conditional Expression:** Class == 1
- This ensures only rows where Class equals 1 (fraud cases) are passed forward.

Node 5 - Histogram (Connected to Row Filter)

25 Histogram 

 NodeHistoGramCal

Details

Examples

Input

▼ in0_25 (31 Cols)


Time	integer
V1	double
V2	double
V3	double
V4	double
V5	double
V6	double
V7	double
V8	double
V9	double
V10	double
V11	double
V12	double
V13	double
V14	double
V15	double
V16	double
V17	double
V18	double
V19	double


Output Storage Level : ?

DEFAULT

Title : ?


Title Color :



#77C27F 

Description :

Description Color :






#808080 

Chart Colors :





Column Name : ?

Amount : double

Number of Bins : ?

30

Purpose

Plots the distribution of transaction amounts for **fraudulent transactions only**.

Configuration

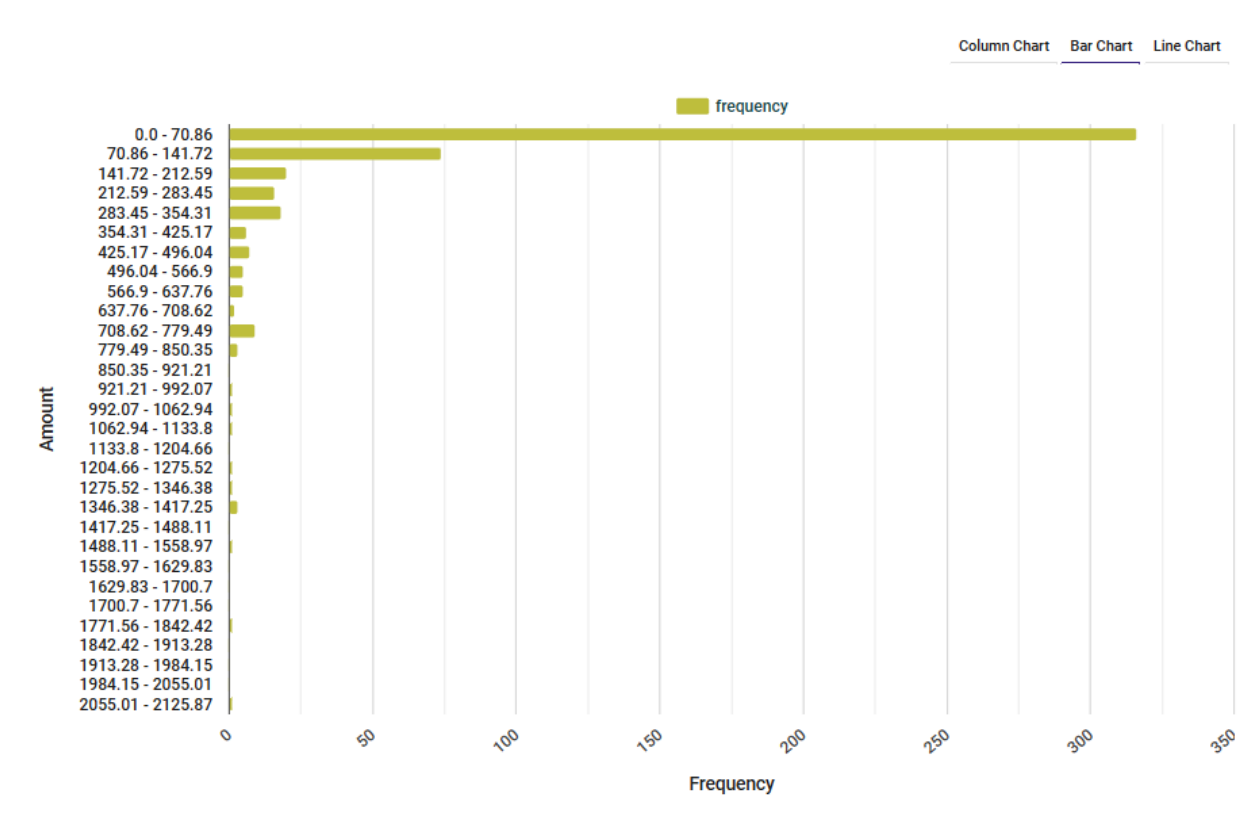
- **Column Name:** Amount : double
- **Number of Bins:** 30
- **Description:** "Shows the distribution of transaction amounts **only for fraud cases**, as filtered from the dataset. Useful to see whether fraudulent transactions cluster around certain amount ranges."

Output

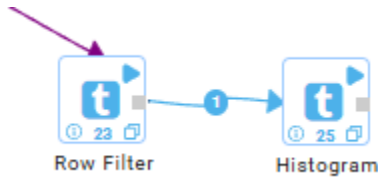
Distribution of continuous values of amount, min value is: 0.0 & max value is: 2125.87

Draggable

Amount_range ↕	Count ↕
0.0 - 70.86	316
70.86 - 141.72	74
141.72 - 212.59	20
212.59 - 283.45	16
283.45 - 354.31	18
354.31 - 425.17	6
425.17 - 496.04	7
496.04 - 566.9	5
566.9 - 637.76	5
637.76 - 708.62	2
708.62 - 779.49	9
779.49 - 850.35	3
850.35 - 921.21	0

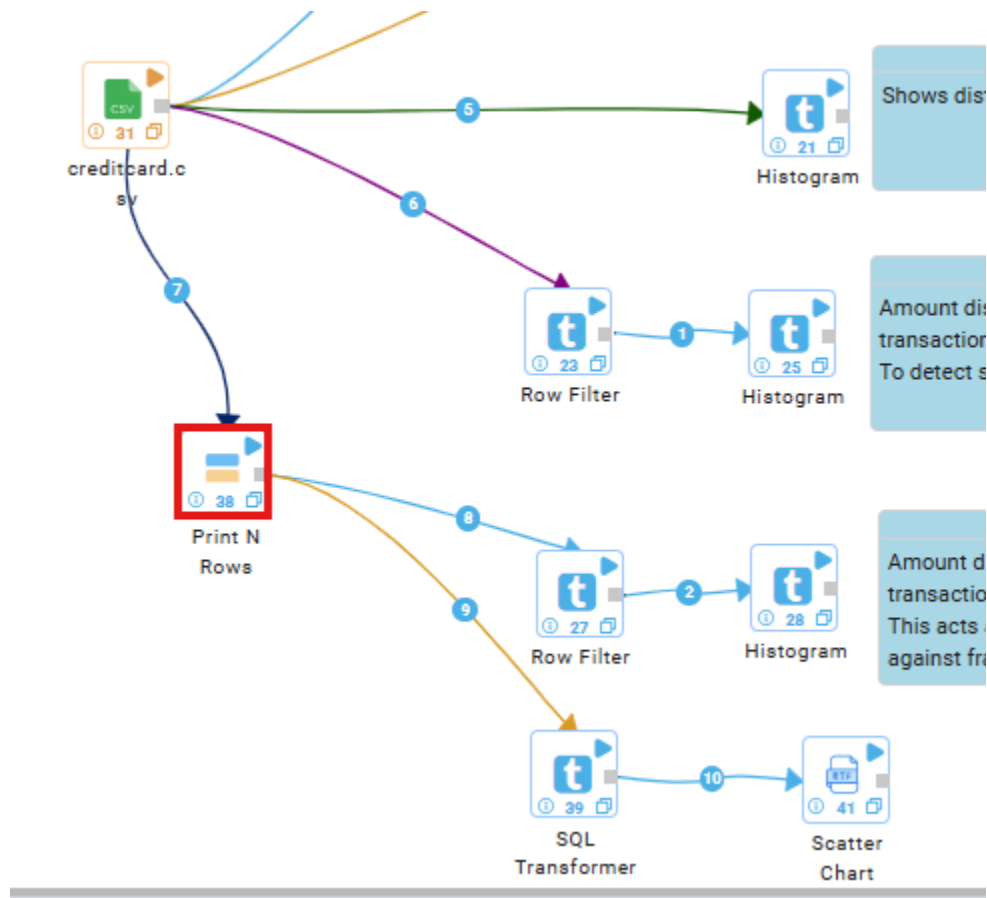


Connection Note:



Node 4 (Row Filter) outputs the fraud-only subset → Node 5 (Histogram) takes this filtered data as input to visualize the distribution of fraud transaction amounts.


Node 6 - Print N Rows



Purpose

Connected directly to the Read CSV node. Since only 5 arrows can come out of the Read CSV node, Print N Rows acts as a **hub** so we can branch out more analyses. It outputs the dataset unchanged but lets us create additional branches.

Node 7 - Row Filter

27 Row Filter 

NodeRowFilter

Details Examples

Input

▼ in0_27 (31 Cols)

Time	integer
V1	double
V2	double
V3	double
V4	double
V5	double

Output Storage Level : ?

DEFAULT

Conditional Expression * : ?

```
1 Class == 0
```


Purpose

To establish a baseline of what “normal” transaction amounts look like.

Configuration

- **Conditional Expression:** `Class == 0`
- This ensures only rows where Class equals 0 (not-fraud cases) are passed forward.

Node 8 - Histogram

28 Histogram 

NodeHistogramCal

Details

Examples

Input

▼ in0_28 (31 Cols)


Time	integer
V1	double
V2	double
V3	double
V4	double
V5	double
V6	double
V7	double
V8	double
V9	double
V10	double
V11	double
V12	double
V13	double
V14	double
V15	double
V16	double
V17	double
V18	double
V19	double


Output Storage Level : ?

DEFAULT

Title : ?


Title Color :



#77C27F 

Description :

Description Color :






#808080 

Chart Colors :





Column Name : ?

Amount : double

Number of Bins : ?

30

Purpose


Plots the distribution of transaction amounts for **non-fraudulent transactions only**.

Configuration

- **Column Name:** Amount : double
- **Number of Bins:** 30
- **Description:** A histogram showing how transaction amounts are distributed among non-fraud cases.

Output

Distribution of continuous values of amount, min value is: 0.0 & max value is: 25691.16

 Draggable

Amount_range ↕	Count ↕
0.0 - 856.37	280387
856.37 - 1712.74	2917
1712.74 - 2569.12	599
2569.12 - 3425.49	208
3425.49 - 4281.86	110
4281.86 - 5138.23	42
5138.23 - 5994.6	19
5994.6 - 6850.98	8
6850.98 - 7707.35	9
7707.35 - 8563.72	6
8563.72 - 9420.09	2
9420.09 - 10276.46	2
10276.46 - 11132.84	0

Node 9- SQL Transformer

39 SQL Transformer

NodeSQLTransformer

Details

Examples

Input

Q Search

▼ in0_39 (31 Cols)

Time integer

V1 double

V2 double

V3 double

V4 double

V5 double

V6 double

V7 double

V8 double

V9 double

V10 double

V11 double

V12 double

V13 double

V14 double

V15 double

V16 double

V17 double

V18 double

Output Storage Level : ?

DEFAULT

Temp Table : ?

t

SQL : ?

```
1 WITH with_id AS (  
2   SELECT  
3     ROW_NUMBER() OVER (ORDER BY Time) AS txn_id,  
4     Amount,  
5     CASE WHEN CAST(Class AS INT) = 1 THEN 'Fraud' ELSE 'Not Fraud' END AS label  
6   FROM t  
7   WHERE Amount IS NOT NULL  
8 ),  
9 sampled AS (  
10    SELECT * FROM with_id  
11    WHERE rand() < 0.6  
12    LIMIT 600  
13 )  
14 SELECT txn_id, Amount AS amount, label  
15 FROM sampled  
16 ORDER BY txn_id;
```

Schema Columns : ?

InferSchema

+

Purpose

Used to transform and sample the dataset with SQL queries. In this case, we assign unique IDs (txn_id), select transaction Amount, and label rows as Fraud or Not Fraud.

Configuration

- After pasting the query, click InferSchema so that the node detects the output columns (txn_id, amount, label).

SQL for Scatter Chart Preparation

This SQL query was used to generate the dataset for the scatter chart in the Data Profiling workflow. It works in three main steps:

1. Assign a Transaction ID

- ROW_NUMBER() OVER (ORDER BY Time) AS txn_id** creates a sequential ID for each transaction, ordered by time.
- This makes it easier to plot transactions along the x-axis.

2. Prepare Labels and Filter Data


- **CASE WHEN CAST(Class AS INT) = 1 THEN 'Fraud' ELSE 'Not Fraud' END AS label** converts the numeric class column into readable labels (“Fraud” / “Not Fraud”).
- The **WHERE Amount IS NOT NULL** ensures only valid transaction amounts are included.

3. Sampling for Visualization

- The subquery **WHERE rand() < 0.6 LIMIT 600** randomly selects about 60% of the data, capped at 600 rows.
- This keeps the scatter chart manageable and visually clear without overplotting.

Finally, the outer query selects only the transaction ID, amount, and label, ordering them by ID to produce a clean dataset for visualization.

Node 10- Scatter Chart Node (Connected to SQL Transformer Node)

41 Scatter Chart  **NodeScatterChart** [Details](#) [Examples](#)

Input

▼ in0_41 (3 Cols)

txn_id long

amount double

label string

Output Storage Level : ?

DEFAULT

Title :

Scatter Chart

Title Color :

#77C27F

Description :

Description Color :

#808080

X Label :

Y Label :

Max Values To Display * : ?

300000

X Column * :

txn_id : long

Y Columns * :

x amount : double

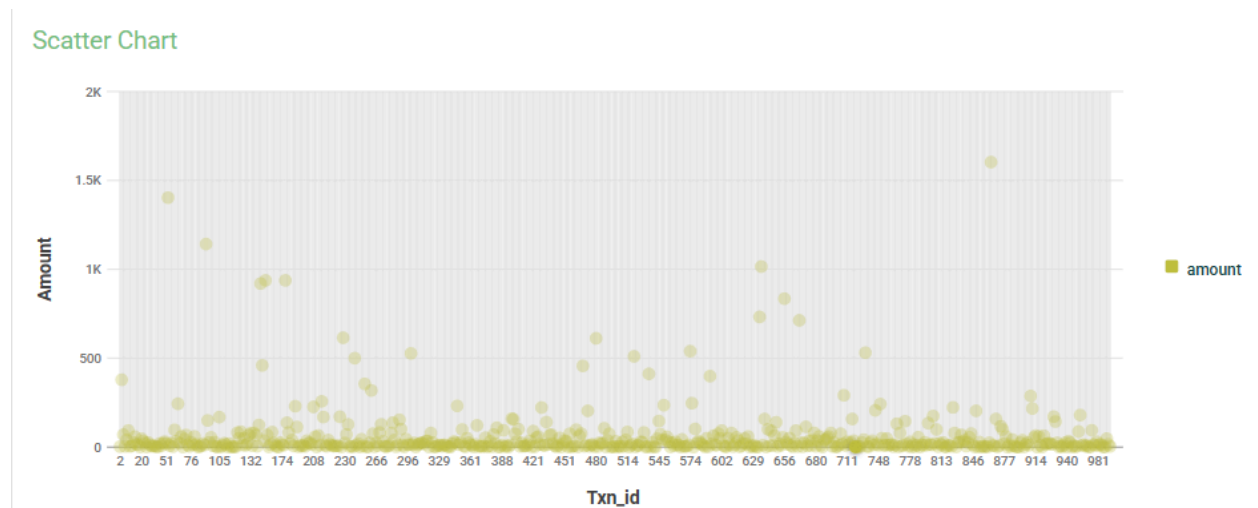
Purpose

To visualize the sampled transactions as a scatter plot.

Configuration

- X Column → txn_id
- Y Column → amount
- Max Values To Display → 300000

3. Output

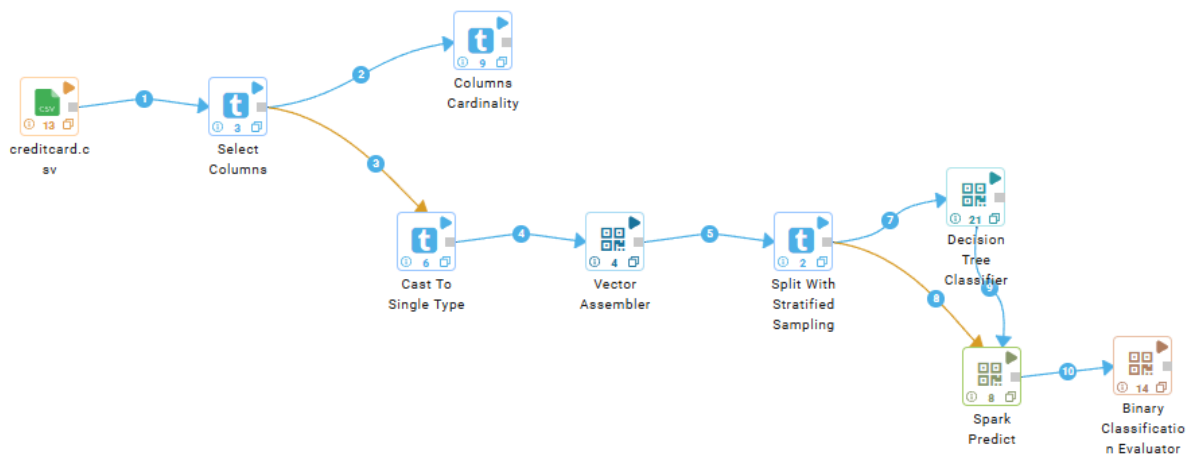


Connection Note:



To pass the transformed and sampled dataset (txn_id, amount, label) from the SQL Transformer node into the Scatter Chart node. This connection ensures the visual chart is based on the cleaned, labeled data produced by SQL.

WORKFLOW 2 - Model Training (Decision Tree Classifier, No SMOTE)



Node 1 - Read CSV (creditcard.csv)

General

InferSchema

ExtraOptions

Output Storage Level : ?

DEFAULT

Path * : ?

/root/fraud_in_credit_transaction/creditcard.csv

Purpose

Reads the dataset into Sparkflows for processing.

Configuration

Same as in Workflow 1 (CSV file path, delimiter, header).

Output

Table with raw dataset rows for downstream nodes

Node 2 - Select Columns

Input

 Search

▼ in0_3 (31 Cols)	
Time	integer
V1	double
V2	double
V3	double
...	...

Output Storage Level : ?

DEFAULT

Columns * : ?

× V1 : double	× V2 : double	× V3 : double	× V4 : double	×
× V18 : double	× V19 : double	× V20 : double	× V21 : double	

Purpose

To filter the dataset and keep only the relevant features for model training. Irrelevant or non-informative columns (like *Time*) are excluded to avoid noise.


Configuration

- **Selected Columns:** All feature columns (V1–V28) and the target column (*Class*).
- **Excluded Column:** *Time* (not useful for training).

Output

A DataFrame containing only the necessary features and target column, ready for downstream processing.

Node 3 - Columns Cardinality


9 Columns Cardinality 

NodeColumnsCardinality [Details](#) [Examples](#)


Input

▼ in0_9 (30 Cols)


V1	double
V2	double
V3	double
V4	double
V5	double
V6	double

Output Storage Level : 

DEFAULT

Max Values To Display * : 

100

Column Names * : 

×

Class : integer

Purpose

To check the distribution of the target column (*Class*) and identify any imbalance between fraud

(1) and non-fraud (0) transactions. This helps in deciding if techniques like SMOTE or re-sampling are needed.

Configuration

- **Column Names:** *Class*
- **Max Values to Display:** 100 (enough to cover both unique values: 0 and 1)

Output A summary of the *Class* column showing counts for each unique value. This confirms the dataset is highly imbalanced (majority = non-fraud).

Columns Unique Count

Columns Unique Count

Draggable

Columns Name	Count
Class	2

Class

Class

Draggable

Values	Count
0	284315
1	492

Node 4 - Cast To Single Type

6 Cast To Single Type

NodeCastColumnType

Details

Examples

Input

▼ in0_6 (30 Cols)

V1	double
V2	double
V3	double
V4	double
V5	double
V6	double
V7	double
V8	double
V9	double

Output Storage Level : ?

DEFAULT

Columns * : ?

×

Class : integer

New Data Type * : ?

DOUBLE

Replace Existing Cols? * : ?

true

Purpose

Convert the target column (*Class*) from integer to double. This is required because the Decision Tree Classifier in Sparkflows only accepts **double** data types for label columns.

Configuration

- **Columns:** *Class*
- **New Data Type:** DOUBLE
- **Replace Existing Cols?:** true (overwrite the original *Class* column)


Output

The *Class* column is successfully cast to type DOUBLE, making the dataset compatible with the classifier.

Tip: Always check the input requirements of your ML algorithm.

- Some algorithms (like Decision Trees, Random Forests, Logistic Regression) require labels to be of type **double**.
- Others may require categorical/string values or encoded features.

Node 5 - Vector Assembler

4 Vector Assembler  **NodeVectorAssembler** Examples

Input

▼ in0_4 (30 Cols)

V1	double
V2	double
V3	double
V4	double
V5	double
V6	double
V7	double
V8	double
V9	double
V10	double

Output Storage Level : ?

DEFAULT

Input Columns : ?

x

 V1 : double

x

 V2 : double

x

 V3 : double

x

 V4 : double

x

 V19 : double

x

 V18 : double

x

 V20 : double

x

 V21 : double

Output Column * : ?

features

Handle Invalid : ?

error

Purpose:

The Vector Assembler node combines multiple feature columns into a single vector column. This step is essential because machine learning algorithms in Sparkflows expect input features to be in a single vector column rather than spread across multiple columns.

Configuration:

- **Input Columns:** All numerical feature columns (V1–V28, plus Amount).
- **Output Column:** features → this is the column that will store the combined vector.
- **Handle Invalid:** Set to error, meaning if invalid data is encountered, the workflow will throw an error rather than ignoring it.

Output:

- A new DataFrame with a single vector column features that represents all selected input columns.
- This column is later used by the Decision Tree classifier for training.

Features

Vector

[-0.966271711572087,-0.185226008082898,1.79299333957872,-0.863291275036453,-

[-1.15823309349523,0.877736754848451,1.548717846511,0.403033933955121,-0.403

[-0.425965884412454,0.960523044882985,1.14110934232219,-0.168252079760302,0

[1.22965763450793,0.141003507049326,0.0453707735899449,1.20261273673594,0.1

[-0.644269442348146,1.41796354547385,1.0743803763556,-0.492199018495015,0.9

[-0.89428608220282,0.286157196276544,-0.113192212729871,-0.271526130088604,1

Tip:

Always ensure you exclude the target variable (Class) when assembling features. Only independent variables should go into the features vector.

Node 6 - Split With Stratified Sampling

2 Split With Stratified Sampling

 SplitWithStratifiedSampling

Details

Input

 Search

▼ in0_2 (31 Cols)	
V1	double
V2	double
V3	double
V4	double
V5	double
V6	double
V7	double
V8	double
V9	double

Output Storage Level : 

DEFAULT

Column Name * : 

Class : double

Fraction : 

0.8

Seed : 

42

Purpose:

This node splits the dataset into training and testing subsets while preserving the original class distribution.

Since our dataset is highly imbalanced (very few fraud cases compared to non-fraud), a stratified split ensures that both subsets have approximately the same class ratio.

Configuration:

- Column Name: Class→ used as the stratification key to preserve fraud/not-fraud balance.
- Fraction: 0.8→ 80% of the data goes into the training set, and 20% into the test set.
- Seed: 42→ ensures reproducibility of the split.

Output:

- Two datasets:
 - Training set (80%) with preserved class ratio.
 - Test set (20%) with preserved class ratio.

Tip:

Stratified sampling is crucial in fraud detection and other imbalanced classification problems. A simple random split might create a test set with very few fraud cases, making evaluation unreliable.

Node 7 - Decision Tree Classifier

21 Decision Tree Classifier

NodeDecisionTreeClassifier

Details

Examples

Input

▼ in0_21 (31 Cols)

V1	double
V2	double
V3	double
V4	double
V5	double
V6	double
V7	double
V8	double
V9	double
V10	double
V12	double
V11	double

General

Grid Search

Confusion Matrix

Confusion

Output Storage Level : ?

DEFAULT

Features Column : ?

features : vectorudt

Label Column : ?

Class : double

Prediction Column : ?

prediction

Path : ?

Purpose:

This node trains a **Decision Tree Classifier** on the training dataset to predict whether a transaction is fraud or not.

Configuration:

- **Features Column:** **features** → the vector of all input variables created by the Vector Assembler.
- **Label Column:** **Class** (double) → the target column (fraud vs. not fraud).
- **Prediction Column:** **prediction** → stores the model's predicted class.

Connections:

- From the **Split With Stratified Sampling** node, we get two outputs:
 - **First arrow (training set)** → connected to the Decision Tree Classifier for model training.
 - **(Later, the second arrow** → test set, will be used for evaluation).

Output:

A trained Decision Tree model that learns decision rules from the training data.

Note:

Decision Trees require the label column to be in double format. That's why earlier we converted Class from integer to double.

Node 8 and 9 - Spark Predict and Binary Classification Evaluator

8

Spark Predict

NodePredict

Details

Input

Q Search

▼ in0_8

(31 Cols)

V1double

V2double

Output Storage Level : ?

DEFAULT

14

Binary Classification Evaluator

NodeBinaryClassificationEvaluator

Details

Examples

Input

Q Search

▼ in0_14

(34 Cols)

V1double

V2double

V3double

V4double

V5double

V6double

V7double

V8double

V9double

V10double

General

Confusion Matrix

Confusion Matrix Description

ROC Curve

Output Storage Level : ?

DEFAULT

Label Column * : ?

Class : double

Prediction Column : ?

prediction : double

Path : ?

/root/fraud_in_credit_transaction/custom_metrics

Spark Predict Node

Purpose:

This node uses the trained model to make predictions on unseen data (usually the test split).

- **How it works:**
 - **Input** = DataFrame (features + labels) + the trained Decision Tree model.
 - **Output** = DataFrame with an extra column that contains the predicted class for each row.
- **Connection:**
 - The second arrow (test split) coming from Split With Stratified Sampling → goes into Spark Predict along with the trained model.

Binary Classification Evaluator

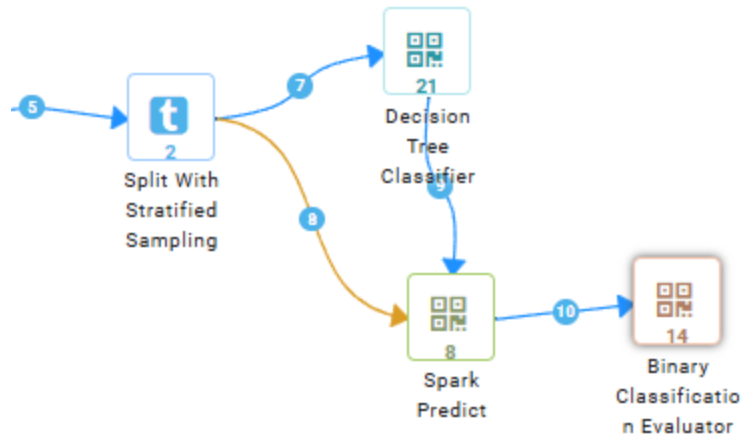
Purpose:

Evaluates how good the predictions are.

- **What it calculates:**
 - **Confusion Matrix (TP, FP, TN, FN)**
 - **Accuracy, AreaUnderROC , AreaUnderPR, Gini**
- **Configuration:**
 - **Label Column** : actual class (e.g., Class)
 - **Prediction Column**: predicted output (e.g., prediction)
 - **Optional**: path to save evaluation metrics.
- **Connection:**
 - The output of Spark Predict → connected to Binary Classification Evaluator.

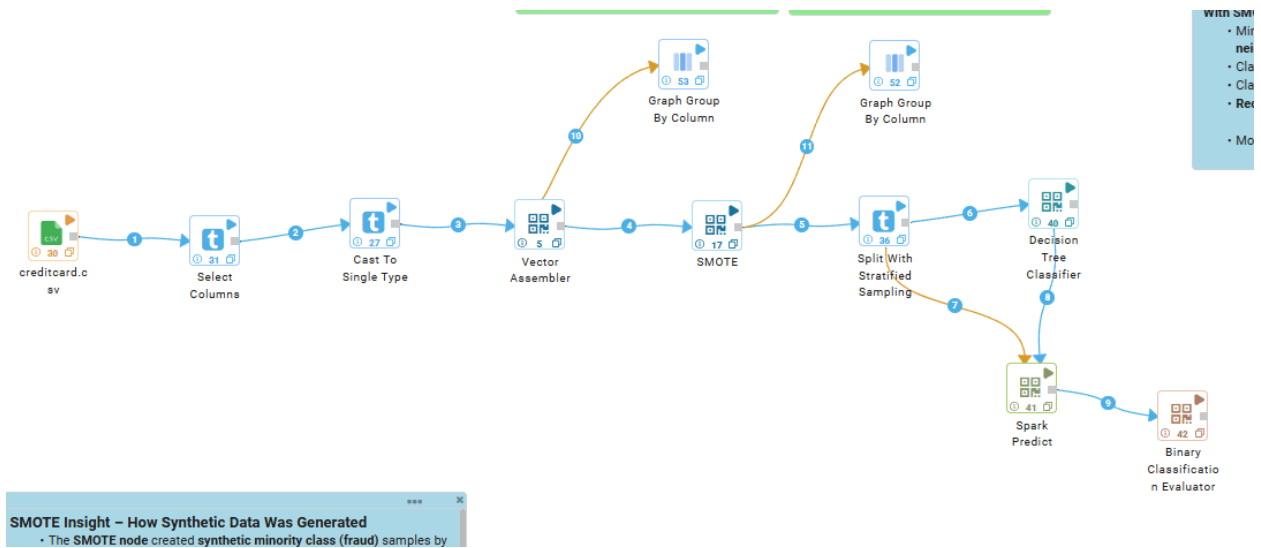
Note: The Classifier node defines the prediction column name, Spark Predict writes it, and the Evaluator reads it.

Connection Note:



As you can see, the first arrow from Split goes to our Classifier and second one goes to Spark Predict. And we connect Spark Predict to our evaluator.

WORKFLOW 3 - Model Training (SMOTE + Decision Tree Classifier)



The initial preprocessing steps (**Read CSV** → **Select Columns** → **Cast to Single Type** → **Vector Assembler**) are identical to the previous workflow and will not be repeated here.

Node 5 - Graph Group By Column

53 Graph Group By Column [NodeGraphGroupByColumn](#) [Details](#) [Examples](#)

Input

Q Search

▼ in0_53 (31 Cols)

V1	double
V2	double
V3	double
V4	double
V5	double
V6	double
V7	double
V8	double
V9	double
V10	double
V11	double
V12	double
V13	double
V14	double
V15	double
V16	double
V17	double
V18	double
V19	double
V20	double
V21	double
V22	double
V23	double
V24	double
V26	double
V25	double
V27	double

Output Storage Level : DEFAULT

Title : Graph of counts Grouped By Column

Title Color : #77C27F

Description : Column Cardinality node before SMOTE (Highly Imbalanced)

Description Color : #808080

X Label : Class

Y Label : Count

Max Values To Display : 2

Chart Colors : #808080

Group By Column * : Class : double

Chart Type : Bar Chart

Purpose

Plots the distribution of the Class column before applying SMOTE to highlight dataset imbalance.

Configuration

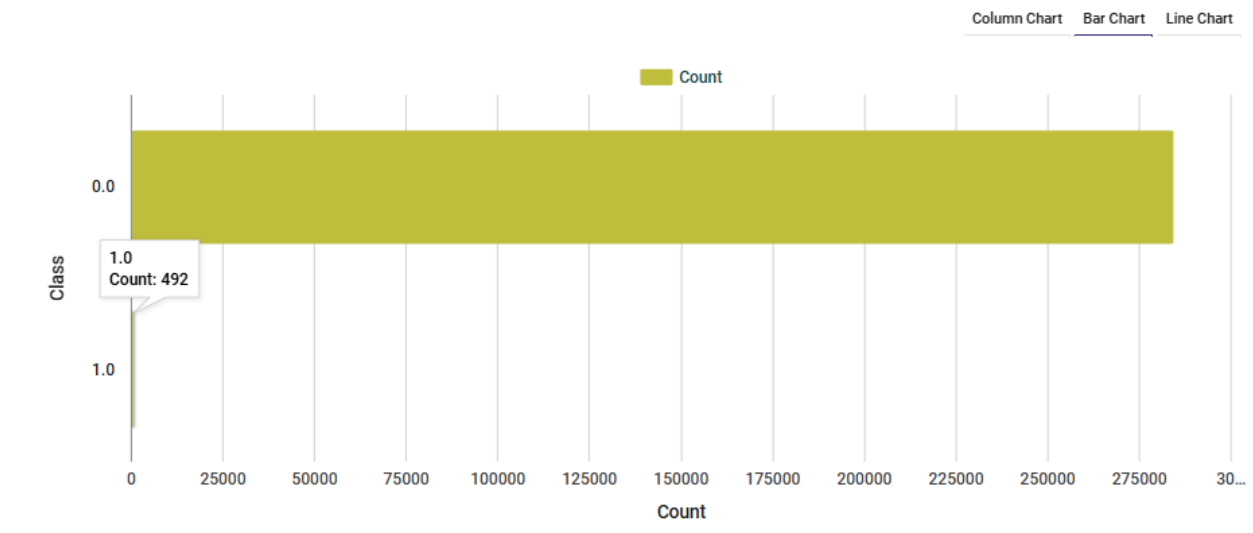
- **Group By Column:** Class : double
- **X Label:** Class
- **Y Label:** Count
- **Chart Type:** Bar Chart
- **Max Values to Display:** 2
- **Description:** Column cardinality visualization before SMOTE (highly imbalanced).

Output


A bar chart showing that the dataset is highly imbalanced. The majority of transactions belong to Class = 0 (non-fraud), while only a very small portion are Class = 1 (fraud). This visualization clearly demonstrates the need for balancing techniques like SMOTE.

Graph of counts Grouped By Column

Column Cardinality node before SMOTE



Node 6 - SMOTE

17 SMOTE 

NodeSMOTE

Details

Examples

Input

▼ in0_17

(31 Cols)

V1	double
V2	double
V3	double
V4	double
V5	double
V6	double
V7	double
V8	double

Output Storage Level : ?

DEFAULT

Label Column * : ?

Class : double

Feature Column * : ?

features : vectorudt

BucketLength : ?

5

Purpose

To address class imbalance in the dataset by creating synthetic samples of the minority class (Class = 1 , fraud). Without balancing, the model would be biased toward predicting the majority class (Class = 0 , non-fraud).

Configuration

- **Label Column:** Class : double
- **Feature Column:** features : vectorudt (from Vector Assembler)
- **Bucket Length:** 5 (used for approximate nearest-neighbor search to generate synthetic samples)

Output & Observations

- The Graph Group By Column node was applied both before and after SMOTE to visualize the class distributions.
- Before SMOTE: Fraud cases = **492**, Non-fraud cases = **275,190** (highly imbalanced).
- After SMOTE: Fraud cases increased to **43,455**, showing significant oversampling of minority cases.

- However, because the fraud class had far fewer samples than the non-fraud class, SMOTE could not perfectly balance the dataset. Still, the resulting dataset is **much more balanced** than the original.

Node 6 - Split With Stratified Sampling

Purpose

To split the dataset into **training** and **testing** sets while preserving the class distribution (stratification). This ensures both fraud and non-fraud cases appear proportionally in each set, which is especially important after SMOTE balancing.

Connections

- The **first output (train set)** from the Split node is connected to the Decision Tree Classifier.
- The **second output (test set)** from the Split node is connected to the Spark Predict node.
- The output of the **Decision Tree Classifier** is also connected to the **Spark Predict** node (to load the trained model).
- The **Spark Predict** node then feeds into the **Binary Classification Evaluator** to assess model performance.

Note

- This section follows the **same structure and configuration** as the workflow in *Model Training with Decision Tree (without SMOTE)*.
- The only difference is that here we used the **SMOTE-balanced dataset** before splitting and training.
- The documentation for the classifier, predictor, and evaluator can be referenced from the earlier workflow.

Output Of The Binary Classification Evaluator:

Confusion Matrix

Target ↕	PredictedLabel ↕	Count ↕
1.0	1.0	12034
0.0	1.0	149
1.0	0.0	998
0.0	0.0	82624

Test Metrics

Metrics ↕	Value ↕
Accuracy	0.9880277647304421
AreaUnderROC	0.8595909707184184
AreaUnderPR	0.8719422337362462
Gini	0.7191819414368368

2A — Decision Tree Classifier (No SMOTE)

- **Accuracy:** 99.923%
- **ROC-AUC:** 0.731
- **PR-AUC:** 0.596
- **Confusion Matrix (test):** TP = 76, FP = 17, FN = 25, TN = 54,556

Takeaways

- Very high accuracy is **misleading** due to severe class imbalance; most predictions default to class 0.
- PR-AUC is low (0.596) → the model struggles to balance precision/recall for fraud.
- **25 missed frauds (FN)** is risky in production.
- This baseline motivates using **SMOTE** (or other rebalancing) before training.

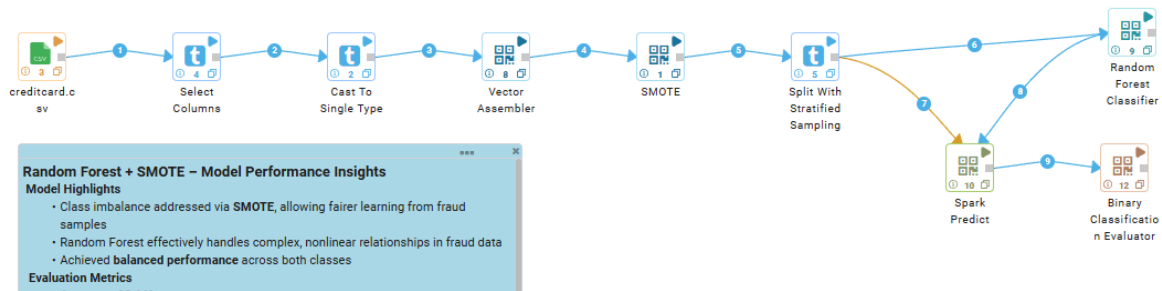
3 — SMOTE + Decision Tree Classifier

- **Accuracy:** 0.9880
- **ROC-AUC:** 0.8596
- **PR-AUC:** 0.8719
- **Confusion Matrix (test after SMOTE split):** TP = 12,034, FP = 149, FN = 998, TN = 82,624

Takeaways

- **PR-AUC jumps** markedly → far better minority-class detection.
- Many more frauds are recovered (higher recall); false alarms increase slightly (FP up), which is a typical trade-off in fraud detection.

WORKFLOW 4 - Model Training (Random Forest + SMOTE)



The only difference from the previous workflow is that we used a **Random Forest Classifier** instead of a Decision Tree. Random Forest is more robust since it trains multiple trees and averages their predictions, reducing overfitting. Below are the results.

Output Of The Binary Classification Evaluator:

Confusion Matrix

Target	PredictedLabel	Count
1.0	1.0	11826
0.0	1.0	72
1.0	0.0	1206
0.0	0.0	82701

Test Metrics

Metrics	Value
Accuracy	0.9866604039455144
AreaUnderROC	0.9908912935736431
AreaUnderPR	0.9779367700905898
Gini	0.9817825871472863

Results

- **Accuracy:** 98.66%
- **Area Under ROC (AUC-ROC):** 0.991 → Excellent overall classification performance
- **Area Under PR (AUC-PR):** 0.978 → Strong fraud detection capability
- **Gini:** 0.981 → High discriminative power

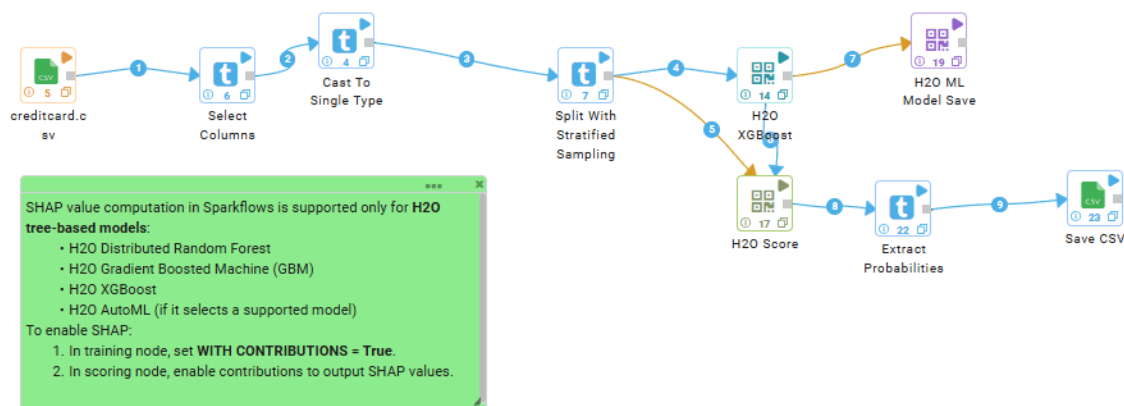
Confusion Matrix Observations:

- **Fraud (Class 1):** TP = 11,826 | FN = 1,206 → High recall, though some frauds remain missed
- **Normal (Class 0):** TN = 82,701 | FP = 72 → Very low false positives

Conclusion:


The combination of **SMOTE + Random Forest** provided the strongest model so far. It achieved excellent recall and precision on the fraud class, while maintaining minimal false positives, making it a well-balanced solution for credit card fraud detection.

WORKFLOW 5 - Model Training (H2O XGBoost)



The initial preprocessing pipeline (Read CSV → Select Columns → Cast To Single Type → Split With Stratified Sampling) is **identical to the previous workflows** and will not be repeated here.

Node 5 - H2O XGBoost

14 H2O XGBoost 

NodeH2OXGBoost Details

Input

▼ in0_14 (30 Cols)

V1	double
V2	double
V3	double
V4	double
V5	double
V6	double
V7	double
V8	double
V9	double
V10	double
...	...

With Contributions : ?

General Advanced Confusion Matrix ROC Curve

Output Storage Level : ?

Is Response Column Categorical : ?

Label Column * : ?

Feature Columns : ?

x V1 : double

x V2 : double

x V4 : double

x V3 : doub

x V18 : double

x V19 : double

x V20 : double

x V21 :

Purpose

Train a gradient-boosted tree model (XGBoost) using the H2O backend.

We use this model because **Sparkflows can compute SHAP values only for H2O tree-based models** (H2O DRF, H2O GBM, **H2O XGBoost**, and H2O AutoML when it picks one of these).

Enabling “contributions” lets us later extract per-feature SHAP contributions.

Configuration (key fields)

- **General tab**
 - **Is Response Column Categorical:** true
(our label Class is 0/1)
 - **Label Column:** Class : double
 - **Feature Columns:** V1 to V28 + Amount
 - **With Contributions:** true
(required to produce SHAP contributions during scoring)
 - Other XGBoost hyperparameters can stay at defaults for the baseline run.


3) Output

- Trained **H2O XGBoost Model** (in-memory).
- Because With Contributions = true, the model is capable of outputting **per-row feature contributions (SHAP)** when used by the H2O Score node.

Connection note

- **Training input** comes from the **training** output of *Split With Stratified Sampling*.
- The trained model is sent to **H2O ML Model Save** and to **H2O Score** for evaluation on the test split.

Node 6 - H2O Model Save

19 H2O ML Model Save 

NodeH2OMojoSave Details Examples

Input

▼ in0_19 (30 Cols)

V1	double
V2	double
V3	double
V4	double
V5	double

Output Storage Level : ?

DEFAULT

Path * : ?

/root/fraud_in_credit_transaction/models/h2o_model

Static path : ?

true

Purpose

This node saves the trained **H2O XGBoost model** to the file system so it can be reused later (e.g., in scoring workflows or deployment).

Configuration

- **Path:** `/root/fraud_in_credit_transaction/models/h2o_model`(directory where the model will be stored).
- **Static Path:** true
 - Ensures the model is always saved to the same path.
 - If a model already exists at that path, it will be **overwritten**.

3) Output

- A **H2O model file** in the given directory.

- This makes the model portable across workflows without retraining.

Node 7 - H2O Score

17
H2O Score
NodeH2OScore
Details
Examples

Input

Q Search

▼ in0_17 (30 Cols)

V1	double
V2	double
V3	double
V4	double
V5	double
V6	double
V7	double
V8	double
V9	double
V10	double
V11	double

Output Storage Level : ?

DEFAULT

is Test Data : ?

true

Compute Shapley Values : ?

true

Path : ?

/root/fraud_in_credit_transaction/h2o_model

Model Type : ?

Classification

Purpose

Score a dataset with a **previously trained H2O model** . The node loads the model, applies it to the incoming DataFrame, and returns predictions. It can also compute **SHAP (contribution) values** for per-row explainability.

Configuration

- **Path:** `/root/fraud_in_credit_transaction/h2o_model` (to save the **SHAPLEY values**)
- **Model Type:** Classification
(we're predicting fraud vs. non-fraud)
- **is Test Data:** true
(treats the input as held-out data; some UIs show evaluation panes accordingly)
- **Compute Shapley Values:** true
(requires *With Contributions* = *true* in the training node; outputs SHAP contributions per feature)

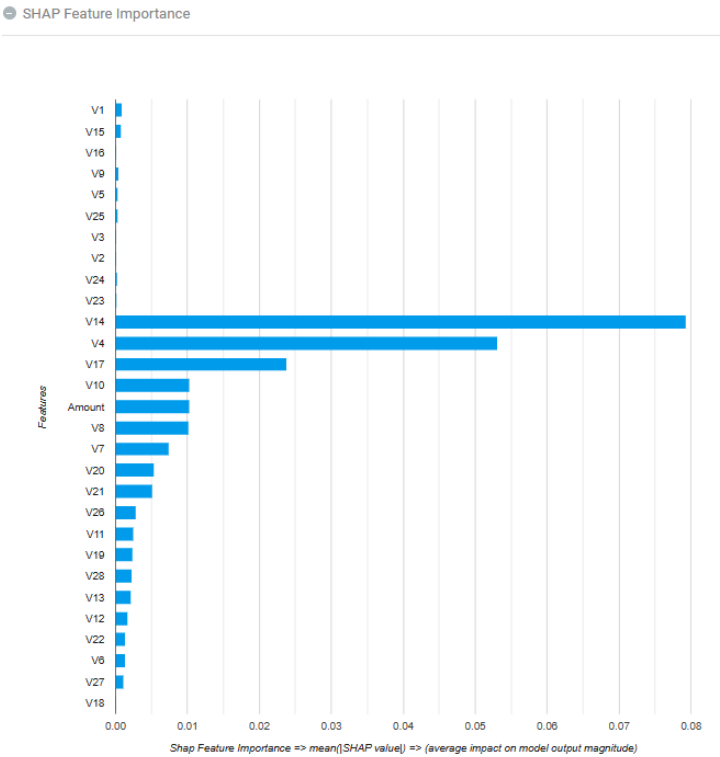
Output

A scored DataFrame that includes:

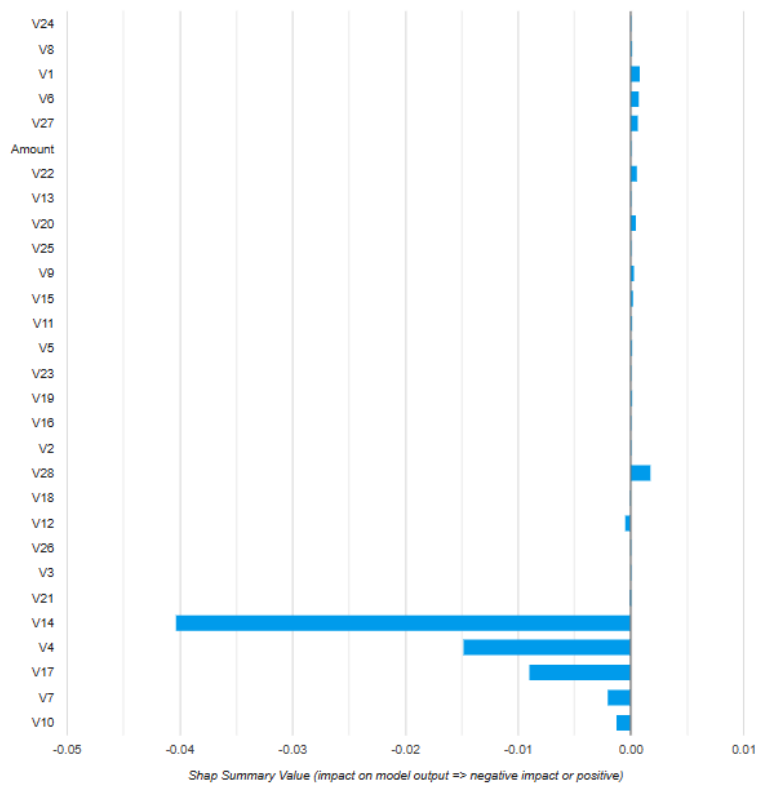
- Class probability

V27 Double	V28 Double	Amount Double	Class Double	Prediction String	Probability Struct
-3.222742325333	1.49782287600289	0.76	0.0	0.0	[0.9954894781112671,0.0045105163007974625]
1.47340829680905	0.244138563960931	4.79	0.0	0.0	[0.9939025044441223,0.006097521632909775]
0.620362430277848	-1.03217048814267	129.0	0.0	0.0	[0.9966862797737122,0.003313720226287842]
-0.232560161540913	-0.61108583011854	0.77	0.0	0.0	[0.9966862797737122,0.003313720226287842]
0.904606610119486	-0.121754915848334	192.6	0.0	0.0	[0.9966862797737122,0.003313720226287842]
-0.898081273839874	0.152504614099668	2547.24	0.0	0.0	[0.9946590662002563,0.005340932868421078]

- SHAP Feature Importance



- **SHAP Summary Plot**



- And calculated **SHAP values** are in the path where we specified (We will extract them in another workflow)

Connections

- **Input:** the post-split dataset you want to score (typically the **test** split).
- **Downstream:** feeds into **Extract Probabilities** node to get p0 and p1

Node 8 - Extract Probabilities

22 Extract Probabilities

NodeExtractProbabilities

Input

▼ in0_22 (32 Cols)

V1 double

V2 double

V3 double

Output Storage Level : ?

DEFAULT

Probability Column Names * : ?

prob_0,prob_1

Purpose

The probability column produced by H2O Score is stored as a Struct type, which cannot be saved directly into CSV. The Extract Probabilities node splits this Struct into separate numeric columns (e.g., prob_0, prob_1) so that downstream nodes (like Save CSV) can handle them properly.

Configuration

- **Probability Column Names:** `prob_0`, `prob_1`
(custom names for each probability column; must match the number of classes in the model output)
- **Output Storage Level:** DEFAULT

⚠ Schema note: Make sure the number of column names matches the number of classes (binary = 2, multi-class = number of classes).

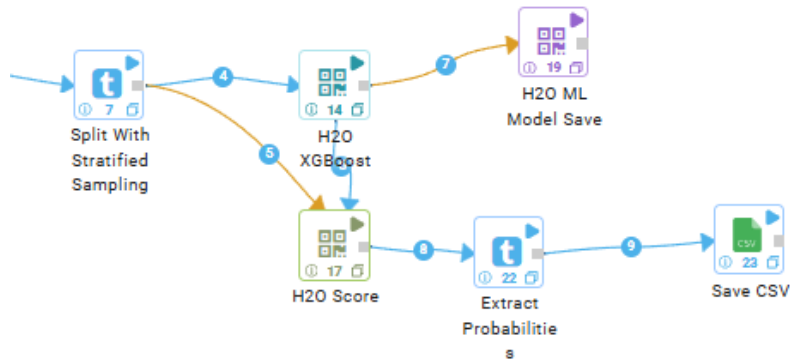
Output

A DataFrame with:

	V26	V25	V27	V28	Amount	Class	prediction	prob_0	prob_1
	-0.12144491541091	-0.16796320328991	-3.222742325333	1.49782287600289	0.76	0.0	0.0	0.9954894781112671	0.0045105163007974625
	0.5378226328911	1.03895780808724	1.47340829680905	0.244138563960931	4.79	0.0	0.0	0.9939025044441223	0.006097521632909775
	0.800676223338709	-0.491076944345672	0.620362430277848	-1.03217048814267	129.0	0.0	0.0	0.9966862797737122	0.003313720226287842
4	0.685621774069854	0.455713915898506	-0.232560161540913	-0.61108583011854	0.77	0.0	0.0	0.9966862797737122	0.003313720226287842
	-0.365926448989994	0.0249768997604201	0.904606610119486	-0.121754915848334	192.6	0.0	0.0	0.9966862797737122	0.003313720226287842
	-0.784874901172977	0.281938884013264	-0.898081273839874	0.152504614099668	2547.24	0.0	0.0	0.9946590662002563	0.005340932868421078
	-0.170472851168095	0.323166241615552	1.91316247413691	0.904414717990565	0.89	0.0	0.0	0.9965320825576782	0.0034679044038057327
	-0.0844723736674454	0.81425757781454	0.195109866380004	-0.695102790892654	80.75	0.0	0.0	0.9961485266685486	0.003851486835628748
	-0.482486956875396	0.120839769567113	-0.886986253069771	-0.444824558758533					

- Original feature columns
- Extracted probability columns (prob_0 = non-fraud probability, prob_1 = fraud probability)

Connections



- **Input:** Scored DataFrame from H2O Score (contains Struct probability column)
- **Downstream:** Feeds into **Save CSV** or any node that requires standard columnar format instead of Structs.

Node 9 - Save CSV

23 Save CSV

NodeSaveCSV

Details Examples

Input

Q Search

▼ in0_23 (33 Cols)

V1	double
V2	double
V3	double
V4	double
V5	double
V6	double
V7	double
V8	double
V9	double
V10	double
V11	double
V12	double
V13	double
V14	double
V15	double
V16	double
V17	double
V18	double

General Advanced

Output Storage Level : ?

DEFAULT

Path * : ?

/root/fraud_in_credit_transaction/predicted_data

Save Mode : ?

Overwrite

Header : ?

true

Encoding : ?

UTF-8

Quote : ?

.

Escape : ?

\

Purpose

The **Save CSV** node writes the final DataFrame (including extracted probabilities and predictions) to a CSV file so it can be stored, shared, or used for further analysis outside Sparkflows.

Configuration

- **Path:** /root/fraud_in_credit_transaction/predicted_data
(output location where the scored dataset will be saved)
- **Save Mode:** Overwrite
(replaces existing file if it already exists)
- **Header:** true
(writes column names as the first row)
- **Encoding:** UTF-8
- **Quote:** "
- **Escape:** \

Note: Because we first used **Extract Probabilities**, the probability column is no longer a Struct making it compatible with CSV export.

Output

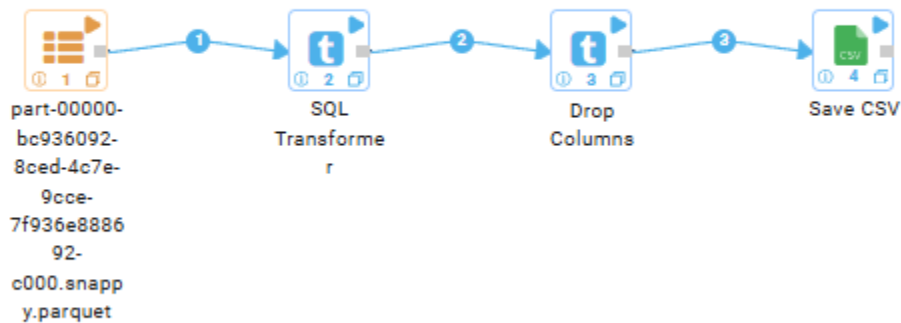
A CSV file containing:

- Original feature columns
- Prediction column (fraud = 1, non-fraud = 0)
- Extracted probability columns (\prob_0, prob_1)

Connections

- **Input:** DataFrame from **Extract Probabilities** (with clean probability columns)
- **Downstream:** External usage (analysis, reporting, sharing). This is typically the **final step** in the workflow.

WORKFLOW 6 - Shap Extract



Node 1 - Read Parquet

(part-00000-bc936092-8ced-4c7e-9cce-7f936e888692-c000.snappy.parquet)

1 Read Parquet

NodeDatasetParquet

Details

Examples

General

InferSchema

Output Storage Level : ?

DEFAULT

Path * : ?

/root/fraud_in_credit_transaction/h2o_model/part-00000-bc936092-8ced-4c7e-9cce-7f936e888692-c000.snappy.parquet

Add Input File Name : ?

false

Purpose

H2O Score outputs SHAP results in .parquet format, so we must read them back into Sparkflows for further transformation.

Configuration

- **Path:**
/root/fraud_in_credit_transaction/h2o_model/part-00000-bc936092-8ced-4c7e-9cce-7f936e888692-c000.snappy.parquet
- **Add Input File Name:** false (keeps only data columns, avoids adding extra filename field).

Output

V28 Double	Amount Double	Class Double	Detailed_prediction Struct
1.49782287600289	0.76	0.0	[0.0,[0.9954894781112671,0.0045105163007974625],[-0.006430813,1.4054009E-5,-0.040169127,-1.3968203E-5,-7.94276E-5,7.376564E-4,-0.011355,-0.0061112125,-0.0010669134,4.5752833E-5,0.11251585,-0.0015408528,6.914143E-4,-1.5085478E-5,-0.018946514,0.0,0.0022348566,0.115046255,0.0024913154,0.0011190779,0.002190277,3.499109E-5,-1.2926589E-4,-6.447236E-5,0.014065254,4,-5.5788035]]
0.244138563960931	4.79	0.0	[0.0,[0.9939025044441223,0.006097521632909775],[-9.100025E-5,-9.966322E-6,-0.1977551,-1.3968203E-5,1.5039174E-4,7.442445E-4,0.025771659,-0.024577878,0.004817604,-0.0067350497,0.0020251856,-8.5904985E-4,0.60903883,0.04582082,0.014229761,4.0193936E-5,-0.014065254,-0.0026828698,0.0024868236,0.011345137,-6.0172053E-5,1.2343357E-4,0.03191544,7.748702E-5,-0.0135248555,0.0020491162,-0.0027293952,

Example Row Breakdown

[0.0,[0.9954894781112671,0.0045105163007974625],[-0.006430813,1.4054009E-5, ..., -1.6597065E-4, -5.5788035]]

- **0.0** → Prediction = non-fraud.
- **0.99548** → Probability non-fraud.
- **0.00451** → Probability fraud.
- **Next values** → SHAP values for features

Node 2 - SQL Transformer

2 SQL Transformer

NodeSQLTransformer

Details

Examples

Input

Q Search

▼ in0_2 (32 Cols)

V1 double

V2 double

V3 double

V4 double

V5 double

V6 double

V7 double

V8 double

V9 double

V10 double

V11 double

V12 double

V13 double

V14 double

V15 double

V16 double

V17 double

Output Storage Level : ?

DEFAULT

Temp Table : ?

t

SQL : ?

1 SELECT

2 *

3 Detailed_prediction.label AS pred_label,

4 Detailed_prediction.probabilities['1.0'] AS prob_1,

5 Detailed_prediction.probabilities['0.0'] AS prob_0,

6 Detailed_prediction.contributions.V1 AS V1_shap,

7 Detailed_prediction.contributions.V2 AS V2_shap,

8 Detailed_prediction.contributions.V3 AS V3_shap,

9 Detailed_prediction.contributions.V4 AS V4_shap,

10 Detailed_prediction.contributions.V5 AS V5_shap,

11 Detailed_prediction.contributions.V6 AS V6_shap,

12 Detailed_prediction.contributions.V7 AS V7_shap,

13 Detailed_prediction.contributions.V8 AS V8_shap,

14 Detailed_prediction.contributions.V9 AS V9_shap,

15 Detailed_prediction.contributions.V10 AS V10_shap,

16 Detailed_prediction.contributions.V11 AS V11_shap,

Purpose

The SQL Transformer node is used to **unpack the Detailed_prediction struct** column into separate, human-readable columns. This makes the model outputs such as predicted label, class probabilities, and SHAP contribution values for each feature, available for further reporting and analysis.

Inputs

- **Temp table name to be used:** t
- **SQL** code to be run

Transformation Logic (SQL)

SELECT

*,

Detailed_prediction.label AS pred_label,

Detailed_prediction.proBABILITIES['1.0'] AS prob_1,

Detailed_prediction.proBABILITIES['0.0'] AS prob_0,

Detailed_prediction.contributions.V1 AS V1_shap,

Detailed_prediction.contributions.V2 AS V2_shap,

...

Detailed_prediction.contributions.V28 AS V28_shap,

Detailed_prediction.contributions.Amount AS Amount_shap,

Detailed_prediction.contributions.BiasTerm AS BiasTerm_shap

FROM t

Explanation of Columns

- * → Brings all original columns from the dataset (V1–V28, Amount, Class, etc.).
- pred_label → The predicted class label (0 = Not Fraud, 1 = Fraud).
- prob_1 → Probability of the transaction being classified as Fraud.
- prob_0 → Probability of the transaction being classified as Not Fraud.

- Vx_shap (e.g., $V1_shap$, $V2_shap$, ...) → SHAP contribution value of each feature ($V1$ - $V28$) to the model's prediction.
- $Amount_shap$ → SHAP contribution of the transaction amount feature.
- $BiasTerm_shap$ → Contribution from the model bias/intercept (baseline reference).

Output

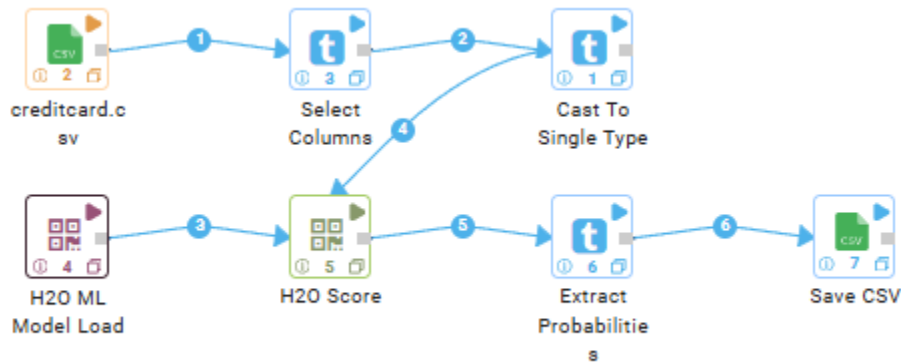
V28	Amount	Class	prediction	pred_label	prob_1	prob_0	V1_shap	V2_shap	V3_shap	V4_shap	V5_shap	V6_shap
1.49782287600289	0.76	0.0	0.0	0.0	0.0045105163007974625	0.9954894781112671	-0.006430813	1.4054009E-5	-1.3968203E-5	-0.040169127	-7.94276E-5	7.376564E-4
0.244138563960931	4.79	0.0	0.0	0.0	0.006097521632909775	0.9939025044441223	-9.100025E-5	-9.966322E-6	-1.3968203E-5	-0.1977551	1.5039174E-4	7.442445E-4
-1.03217048814267	129.0	0.0	0.0	0.0	0.003313720226287842	0.9966862797737122	-7.201598E-5	1.0663435E-5	-1.3968203E-5	-0.03586924	2.0424782E-4	0.001318454
-0.61108583011854	0.77	0.0	0.0	0.0	0.003313720226287842	0.9966862797737122	-7.201598E-5	-9.966322E-6	-1.3968203E-5	-0.03717856	1.6743578E-4	0.001566907
-0.121754915848334	192.6	0.0	0.0	0.0	0.003313720226287842	0.9966862797737122	-7.5177166E-5	9.939522E-6	-8.974287E-6	-0.05201159	2.6470757E-4	3.424147E-5

After extracting all necessary values from the Detailed_prediction struct using the SQL Transformer, we remove the original Detailed_prediction column with the **Drop Columns** node. This step is required because the column is of type **struct**, and downstream nodes such as **Save CSV** cannot process nested data types. Once the struct column is dropped, we use the **Save CSV** node to export the cleaned dataset (with predictions, probabilities, and SHAP contributions) into a flat CSV format for further analysis or reporting.


Connections



WORKFLOW 7 - Score



H2O ML Model Load

4 H2O ML Model Load  [NodeH2OMojoLoad](#) [Details](#) [Examples](#)

Output Storage Level : ?

DEFAULT

Path * : ?

/root/fraud_in_credit_transaction/models/h2o_model

This node is used to **bring in the previously trained and saved H2O model**. By providing the saved model path, the node loads the model back into the workflow so it can be reused for scoring on new datasets. This is the same model we trained and saved in the earlier workflow, and here it is connected to the H2O Score node to generate predictions and probabilities.

After scoring, probabilities are extracted and the final results are saved to CSV.