Section 1: Databricks SQL

Target Audience

Main Target Audience Description

Databricks SQL is primarily designed for data analysts, SQL analysts, and business analysts. These professionals use Databricks SQL to query, analyze, and visualize data stored on the Lakehouse platform. The tool offers an integrated and scalable environment that facilitates the manipulation of large volumes of data, enabling the creation of valuable insights for organizations.

Secondary Target Audience Description

Although the primary focus is on data, SQL, and business analysts, data scientists and data engineers can also benefit from Databricks SQL. They can use the tool to perform advanced analyses, develop machine learning models, and implement robust data pipelines.

Benefits of Databricks SQL

Unified Platform

Databricks SQL offers a unified platform that seamlessly integrates data processing and analysis. This eliminates the need to move data between different systems, reducing latency and improving operational efficiency.

Scalability and Performance

With the ability to automatically scale as needed, Databricks SQL allows for efficient processing of large volumes of data. The use of elastic clusters ensures that computational resources are always available to meet workload demands without resource waste.

Integration with External Tools

Databricks SQL easily integrates with various data visualization and ingestion tools, such as Tableau, Power BI, Looker, and Fivetran. This provides a rich and multifunctional user experience, allowing data to be presented in a visually appealing and understandable manner.

Security and Governance

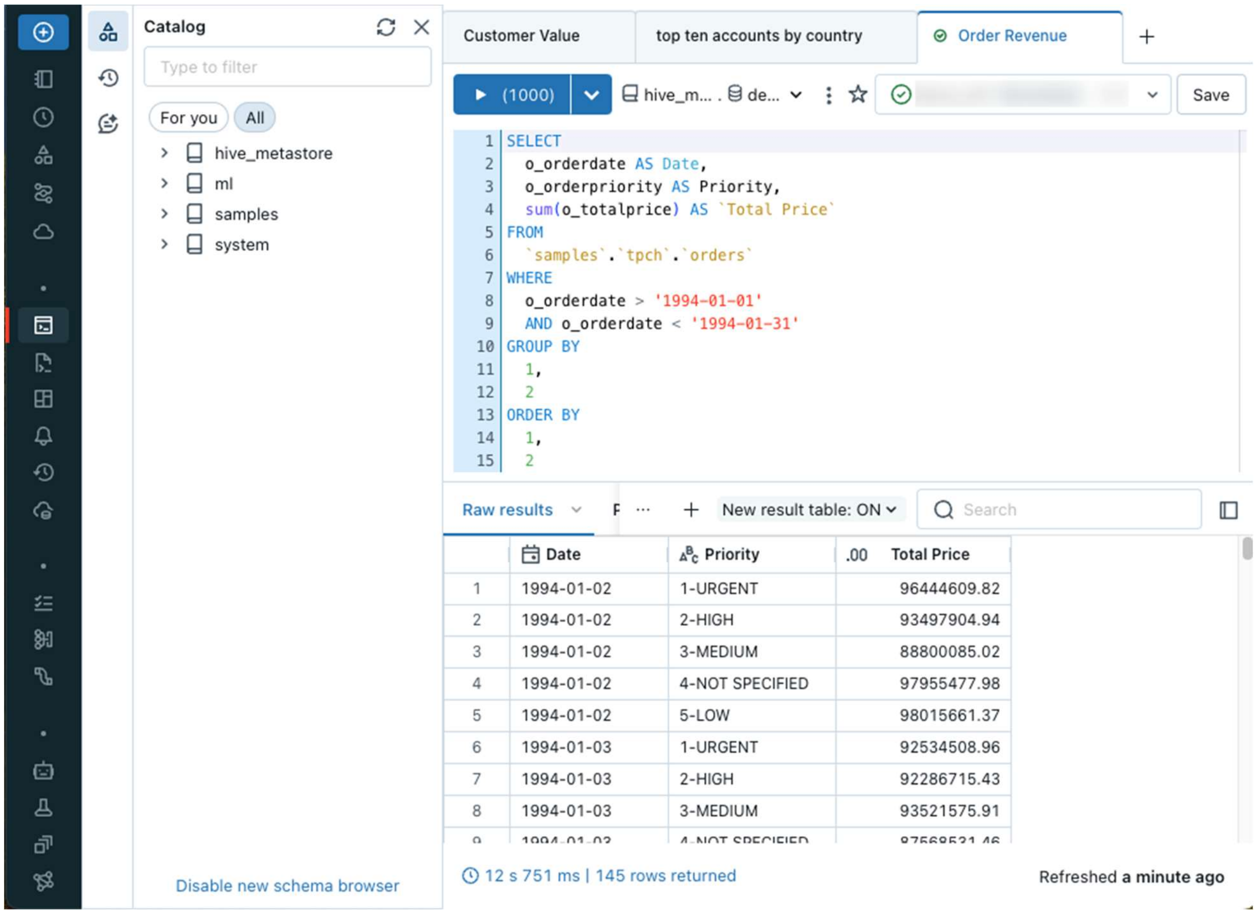
Databricks SQL includes robust security and governance features, allowing granular control over data access and compliance with privacy and security regulations. Access policies can be defined and managed to ensure that only authorized users can access sensitive information.

Buy the complete book

Performing Basic Queries in Databricks SQL

Access the Query Editor

In the Databricks Workspace, select the "SQL" option and click "New Query." This will open the query editor where you can write your SQL code.



```
SELECT
  o_orderdate AS Date,
  o_orderpriority AS Priority,
  sum(o_totalprice) AS `Total Price`
FROM
  `samples`.`tpch`.`orders`
WHERE
  o_orderdate > '1994-01-01'
  AND o_orderdate < '1994-01-31'
GROUP BY
  1,
  2
ORDER BY
  1,
  2
```

	Date	Priority	Total Price
1	1994-01-02	1-URGENT	96444609.82
2	1994-01-02	2-HIGH	93497904.94
3	1994-01-02	3-MEDIUM	88800085.02
4	1994-01-02	4-NOT SPECIFIED	97955477.98
5	1994-01-02	5-LOW	98015661.37
6	1994-01-03	1-URGENT	92534508.96
7	1994-01-03	2-HIGH	92286715.43
8	1994-01-03	3-MEDIUM	93521575.91

12 s 751 ms | 145 rows returned

Refreshed a minute ago

Write the SQL Query

Use the Query Editor text box to type the desired SQL query. For example:

```sql
SELECT * FROM example_table;
```

The Query Editor offers features such as autocomplete, syntax highlighting, and schema visualization to facilitate query construction.

Run the Query

Click the "Run" button to execute the query. The results will be displayed in the section below the editor, where you can analyze the returned data.

Schema Visualization

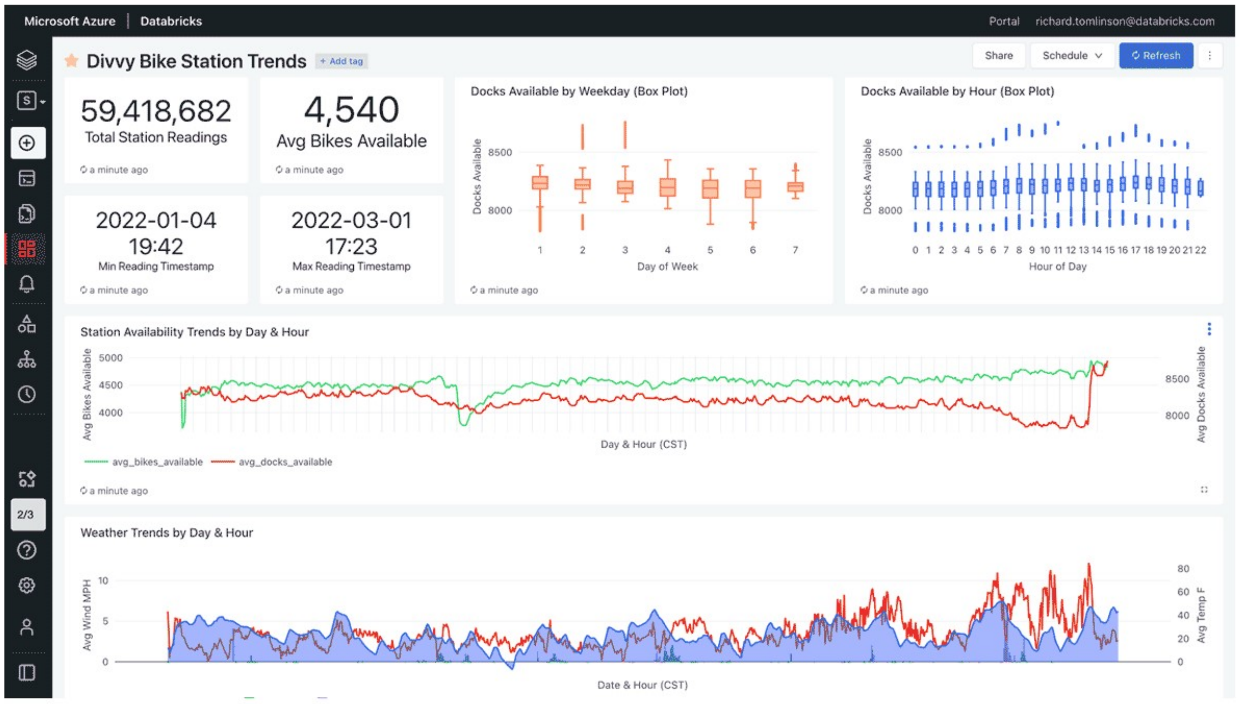
Use the schema browser on the sidebar to explore the available tables and columns. This helps identify fields and data structures you can use in your queries.

Dashboards in Databricks SQL

Creating Dashboards

Dashboards allow you to consolidate the results of multiple queries into a single visual interface. To create a dashboard:

1. **Create Visualizations**: Run queries in the Query Editor and save the resulting visualizations. Each visualization represents the output of a specific query.
2. **Group Visualizations in a Dashboard**: In the "Dashboards" menu, select "Create Dashboard" and add the desired visualizations. You can drag and drop the visualizations to organize the dashboard layout as needed.
3. **Automatic Refresh Configuration**: Configure the dashboard to refresh automatically at regular intervals, ensuring that the displayed data is always up-to-date. This is crucial to ensure users make decisions based on recent data.



Sharing Dashboards

Dashboards can be shared with other users within the organization, allowing for collaboration and dissemination of insights. You can set permissions to control who can view or edit the dashboard, ensuring data security and integrity.

Endpoints/Warehouses in Databricks SQL

Purpose of Endpoints

Endpoints (or warehouses) are computational resources dedicated to executing SQL queries and updating dashboards. They provide the necessary infrastructure to efficiently and high-performance process large volumes of data.

Serverless Endpoints

Serverless endpoints offer a quick and easy start option, eliminating the need for manual cluster management. This serverless approach is particularly useful for ad hoc or low-frequency workloads, providing a simplified and cost-effective user experience.

Cost Considerations

Cluster size and endpoint configuration directly influence operational costs. It is important to balance the need for performance with the available budget to optimize resources. Larger clusters offer greater processing capacity but also cost more. Adjusting the cluster size according to demand can help control costs.

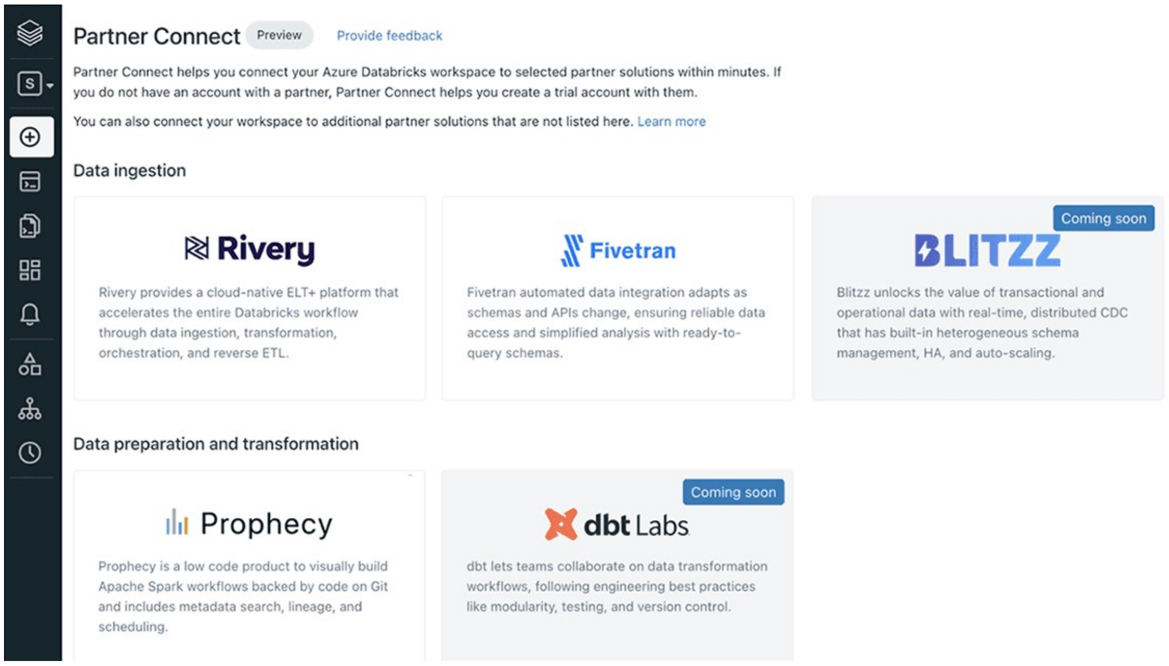
Integrations and Connectivity

Partner Connect

Partner Connect facilitates the integration of Databricks SQL with other tools and services, enabling quick and simplified data pipeline setup. To use Partner Connect:

1. **Select the Partner**: In the "Partner Connect" menu, choose the service or tool you want to integrate. Databricks offers a list of integrated partners, such as Fivetran, Tableau, Power BI, and Looker.
2. **Configure the Integration**: Follow the instructions to connect your Databricks environment to the selected tool. This may include generating authentication tokens, configuring endpoints, and defining access permissions.

Buy the complete book


Partner Connect Preview Provide feedback

Partner Connect helps you connect your Azure Databricks workspace to selected partner solutions within minutes. If you do not have an account with a partner, Partner Connect helps you create a trial account with them.

You can also connect your workspace to additional partner solutions that are not listed here. Learn more

Data ingestion

Rivery: Rivery provides a cloud-native ELT+ platform that accelerates the entire Databricks workflow through data ingestion, transformation, orchestration, and reverse ETL.

Fivetran: Fivetran automated data integration adapts as schemas and APIs change, ensuring reliable data access and simplified analysis with ready-to-query schemas.

BLITZZ (Coming soon): Blitzz unlocks the value of transactional and operational data with real-time, distributed CDC that has built-in heterogeneous schema management, HA, and auto-scaling.

Data preparation and transformation

Prophecy: Prophecy is a low code product to visually build Apache Spark workflows backed by code on Git and includes metadata search, lineage, and scheduling.

dbt Labs (Coming soon): dbt lets teams collaborate on data transformation workflows, following engineering best practices like modularity, testing, and version control.

Connection with Visualization Tools

Tools like Tableau, Power BI, and Looker can be connected to Databricks SQL to create advanced and interactive visualizations. To connect one of these tools:

1. **Obtain Connection Credentials**: In Databricks SQL, generate the necessary credentials (host, token, etc.). These credentials will be used to establish a secure connection between Databricks and the visualization tool.
2. **Configure the Connection in the Visualization Tool**: Use the obtained credentials to set up the connection in the chosen visualization tool. Each tool has its configuration process, usually found in the connection options or data source settings.

Medallion Architecture

Sequential Data Organization

The medallion architecture organizes data into sequential layers: bronze, silver, and gold. Each layer represents a stage of data cleaning and processing, providing a clear and organized structure for data analysis.

Bronze Layer

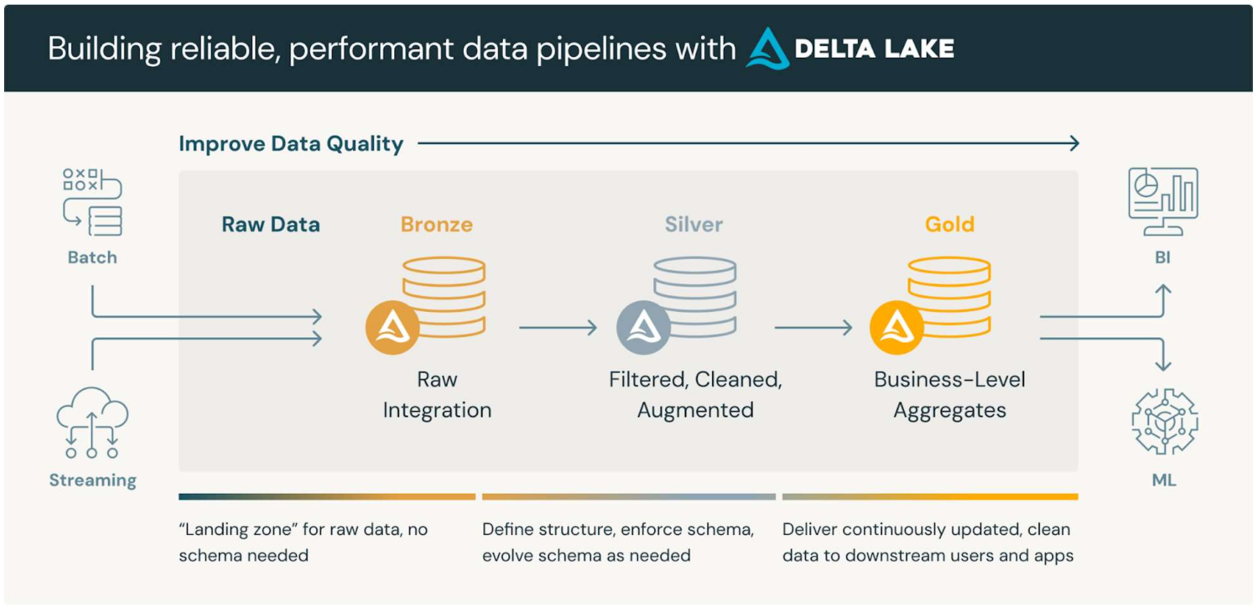
Raw data, directly ingested from external sources. This layer contains data in its original state, without any transformation or cleaning.

Silver Layer

Refined, cleaned, and structured data for analysis. In this layer, data has undergone cleaning and transformation processes, making it more consistent and usable.

Gold Layer

Highly processed and ready-for-consumption data for analysts and BI tools. The gold layer is the most valuable, containing high-quality data ready for advanced analyses and insights generation.


Building reliable, performant data pipelines with DELTA LAKE
Improve Data Quality
Batch
Streaming
Raw Data
Bronze
Raw Integration
Silver
Filtered, Cleaned, Augmented
Gold
Business-Level Aggregates
BI
ML
"Landing zone" for raw data, no schema needed
Define structure, enforce schema, evolve schema as needed
Deliver continuously updated, clean data to downstream users and apps

Benefits of the Gold Layer

The gold layer is the most used by data analysts, as it contains ready-to-analyze data with guaranteed quality and consistency. This facilitates the generation of accurate and actionable insights, enabling data-driven decisions.

Working with Streaming Data

Cautions and Benefits

Processing streaming data requires special considerations due to the continuous nature of the data. Databricks SQL offers robust support for streaming data, allowing ingestion and real-time analysis. This is particularly useful for use cases like system monitoring, real-time log analysis, and fraud detection.

Mixing Batch and Streaming Workloads

The Lakehouse platform allows the combination of batch and streaming workloads, providing unique flexibility for different data processing scenarios. This enables organizations to process both historical and real-time data in a unified environment, simplifying data architecture and improving operational efficiency.

Section 2: Data Management

Delta Lake

Description of Delta Lake

Delta Lake is a storage layer that brings reliability to data lakes, offering ACID (Atomicity, Consistency, Isolation, Durability) transactions, data versioning, and the ability to efficiently manipulate data. It manages the stored data files, ensuring integrity and performance.

Metadata Management

Delta Lake manages the metadata of tables, including schemas and change history. This facilitates auditing and change tracking, in addition to supporting time travel operations, where you can query data in a previous state.

Data History

Delta Lake tables maintain a history of changes, allowing you to query previous versions of the data. This is essential for audits, data recovery, and historical analyses.

Benefits of Delta Lake on the Lakehouse Platform

1. **Reliability**: ACID transactions ensure that all operations are completed successfully or reverted in case of failures.
2. **Performance**: Improved performance in reading and writing data due to file optimization and metadata management.
3. **Flexibility**: Support for various data formats and integration with big data tools like Apache Spark.

Persistence and Scope of Tables in Databricks

Managed vs. Unmanaged Tables

1. **Managed Tables**: Databricks manages the physical location of the data and its metadata. This simplifies management, as all operations (creation, deletion, modification) are controlled by Databricks.
2. **Unmanaged Tables**: You specify the physical location of the data. This provides more control over where the data is stored, but you are responsible for managing the underlying files.

Identifying Managed or Unmanaged Tables

You can identify if a table is managed or unmanaged by observing the table definition in Databricks. Managed tables do not specify a location path (LOCATION).

LOCATION Keyword

The LOCATION keyword allows you to specify a custom path to store table data. This is useful for organizing data in different directories or using external storage.

Creating, Using, and Dropping Databases, Tables, and Views

Creating Databases and Tables

```
CREATE DATABASE example_db;

CREATE TABLE example_db.example_table (

  id INT,

  name STRING

);
```

Using Databases and Tables

```
USE example_db;

SELECT * FROM example_table;
```

Dropping Databases and Tables

```
DROP TABLE example_db.example_table;

DROP DATABASE example_db;
```

Creating and Comparing Views and Temp Views

1. **Views**: Are reusable query definitions. They do not physically store data, only the query logic.

```
CREATE VIEW example_view AS
SELECT * FROM example_table WHERE id > 10;
```

2. **Temp Views**: Are similar to views but exist only in the current session and are discarded when the session ends.

```
CREATE TEMP VIEW example_temp_view AS

SELECT * FROM example_table WHERE id > 10;
```

Data Persistence in Views and Temp Views

Views are persistent, while temp views are temporary and exist only during the current session.

Exploration, Visualization, and Data Security using Data Explorer

Data Exploration

Data Explorer allows you to explore database schemas, view tables and columns, and inspect the data contained within them. This facilitates navigation and understanding of the data structure.

Data Visualization

You can view data samples directly in Data Explorer, helping verify table contents without writing SQL queries.

Data Security

Data Explorer also allows you to manage access permissions. You can change access rights to tables and identify the table owner, ensuring that only authorized users can access sensitive data.

Changing Access Rights to Tables

```
GRANT SELECT ON example_table TO example_user;

REVOKE SELECT ON example_table FROM example_user;
```

Considerations for PII (Personally Identifiable Information) Data

Responsibilities of the Table Owner

The table owner is responsible for managing access and ensuring compliance with data privacy and security regulations. This includes protecting PII data, such as names, addresses, and other personal information.

Organization-Specific Considerations

Each organization may have specific policies for handling PII data. This may include data encryption, anonymization, and data retention policies.

Section 3: SQL in the Lakehouse

SQL Queries

Retrieving Data with Specific Conditions

To retrieve data from a table with specific conditions, you can use the WHERE clause in your SQL queries. For example:

```
SELECT * FROM sales

WHERE sale_date >= '2023-01-01' AND sale_value > 1000;
```

This query returns all sales made from January 1, 2023, with a value greater than 1000.

Output of a SELECT Query

The SELECT query is used to retrieve data from one or more tables. The output of a SELECT query is a result set that matches the specified columns and conditions. For example:

```
SELECT name, age FROM customers

WHERE city = 'São Paulo';
```

This query returns the names and ages of customers residing in São Paulo.

Comparison of SQL Commands

MERGE INTO

The MERGE INTO command combines insert, update, and delete operations based on a match condition. It is useful for synchronizing tables. For example:

```
MERGE INTO target_customers AS dest

USING source_customers AS orig

ON dest.id = orig.id

WHEN MATCHED THEN
```

```
  UPDATE SET dest.name = orig.name, dest.age = orig.age

WHEN NOT MATCHED THEN

  INSERT (id, name, age) VALUES (orig.id, orig.name, orig.age);
```

INSERT INTO

The INSERT INTO command inserts new records into a table. For example:

```
INSERT INTO customers (id, name, age)

VALUES (1, 'John', 30);
```

COPY INTO

The COPY INTO command is used to load data from external files into a table in Databricks. For example:

```
COPY INTO target_table

FROM '/path/to/file.csv'

FILEFORMAT = CSV;
```

Subqueries

Simplifying Queries with Subqueries

Subqueries are queries nested within other queries. They can simplify logic and improve the readability of SQL code. For example:

```
SELECT name

FROM customers

WHERE id IN (SELECT customer_id FROM sales WHERE sale_value > 1000);
```

This query returns the names of customers who made sales with a value greater than 1000.

Joins

Comparison of Different Types of Joins

Joins are used to combine records from two or more tables based on a matching condition.

1. **INNER JOIN**: Returns only the records that have matches in both tables.

```sql
SELECT a.id, a.name, b.sale_value

FROM customers a

INNER JOIN sales b ON a.id = b.customer_id;
```

2. **LEFT JOIN**: Returns all records from the left table and the matching records from the right table. If no match, the result from the right side will be null.

```sql
SELECT a.id, a.name, b.sale_value

FROM customers a

LEFT JOIN sales b ON a.id = b.customer_id;
```

3. **RIGHT JOIN**: Returns all records from the right table and the matching records from the left table. If no match, the result from the left side will be null.

```sql
SELECT a.id, a.name, b.sale_value

FROM customers a

RIGHT JOIN sales b ON a.id = b.customer_id;
```

4. **FULL JOIN**: Returns all records when there is a match in either table.

```sql
SELECT a.id, a.name, b.sale_value

FROM customers a

FULL JOIN sales b ON a.id = b.customer_id;
```

5. **LEFT ANTI JOIN**: Returns only the rows from the left table that do not have a match in the right table. It is useful for finding records present in one table but not the other.

```
SELECT a.id, a.name

FROM customers a

LEFT ANTI JOIN sales b ON a.id = b.customer_id;
```

6. **LEFT SEMI JOIN**: Returns only the rows from the left table that have a match in the right table. It is useful for filtering records based on existence in another table.

```
SELECT a.id, a.name

FROM customers a

LEFT SEMI JOIN sales b ON a.id = b.customer_id;
```

Data Aggregation

Data Aggregation

Data aggregation allows summarizing large volumes of data to derive meaningful insights. Common aggregate functions include SUM, AVG, COUNT, MAX, and MIN. For example:

```
SELECT customer_id, SUM(sale_value) AS total_sales

FROM sales

GROUP BY customer_id;
```

This query returns the total sales per customer.

Handling Nested Data Formats

Databricks SQL allows handling nested and complex data formats, such as JSON and Parquet, within tables. This is useful for working with semi-structured data.

Roll-Up and Cube

Using Roll-Up and Cube

The ROLLUP and CUBE operations are used to create multidimensional aggregations, facilitating data analysis at different levels of detail.

Example of ROLLUP

Suppose we have a sales table with the following data:

department	product	sales
Electronics	TV	1000
Electronics	Mobile	1500
Furniture	Sofa	2000
Furniture	Table	500

The query for ROLLUP would be:

```
SELECT department, product, SUM(sales) AS total_sales

FROM sales

GROUP BY ROLLUP (department, product);
```

ROLLUP Result

department	product	total_sales
Electronics	TV	1000
Electronics	Mobile	1500
Electronics	NULL	2500
Furniture	Sofa	2000
Furniture	Table	500
Furniture	NULL	2500
NULL	NULL	5000

Explanation of ROLLUP

- For each combination of department and product, the sum of sales is calculated.
- Then, a subtotal for each department (with product as NULL) is calculated.
- Finally, a grand total (with both department and product as NULL) is calculated.

Example of CUBE

The query for CUBE would be:

```
SELECT department, product, SUM(sales) AS total_sales

FROM sales

GROUP BY CUBE (department, product);
```

CUBE Result

department	product	total_sales
Electronics	TV	1000
Electronics	Mobile	1500
Electronics	NULL	2500
Furniture	Sofa	2000
Furniture	Table	500
Furniture	NULL	2500
NULL	TV	1000
NULL	Mobile	1500
NULL	Sofa	2000
NULL	Table	500
NULL	NULL	5000

Explanation of CUBE

- Computes all possible combinations of department and product.
- Includes partial totals for each level of department and product.
- Includes a grand total (with both department and product as NULL).

Comparison between ROLLUP and CUBE

- **ROLLUP**: Focuses on grouping hierarchy, creating subtotals and a grand total.
- **CUBE**: Creates all possible combinations of groupings, providing a more comprehensive view of aggregations.

Temporal Data Aggregation

Using Temporal Windowing

Window functions allow performing aggregate calculations over specific data intervals. For example:

```
SELECT sale_date,

       SUM(sale_value) OVER (PARTITION BY customer_id ORDER BY sale_date)
AS cumulative_total

FROM sales;
```

This query calculates the cumulative total sales per customer, ordered by sale date.

ANSI SQL in the Lakehouse

Benefits of ANSI SQL Standard

Adopting the ANSI SQL standard in the Lakehouse ensures compatibility and consistency across different database management systems. This facilitates query and knowledge migration between platforms, promoting interoperability.

Access and Cleaning of Silver Level Data

Identification and Access to Silver Level Data

Silver-level data in Databricks represents cleaned and structured data, ready for analysis. You can access this data using SQL queries for further analysis.

Cleaning of Silver Level Data

Silver-level data is prepared through cleaning processes such as deduplication, handling missing values, and standardizing formats.

Query History and Caching

Using Query History

Query history allows you to review and reuse previous queries, saving development time and avoiding code repetition.

Query Caching

Query caching stores the results of frequently executed queries in memory, reducing latency and improving performance

Advanced Functions and UDFs

Performance Optimization with Advanced Spark SQL Functions

Advanced functions, such as ARRAY, MAP, STRUCT, and custom aggregation functions, can be used to optimize the performance of complex queries.

Creating and Applying UDFs

User-Defined Functions (UDFs) allow creating custom functions for specific scalability scenarios. For example, a UDF can be created to standardize address formats:

```python
from pyspark.sql.functions import udf

from pyspark.sql.types import StringType

@udf(StringType())

def standardize_address(address):

    # Implementation of the standardization function

    return address.lower().replace(" street ", " st. ")

# Applying the UDF

df = df.withColumn("standardized_address", standardize_address(df.address))
```

Section 4: Data Visualization and Dashboards

Creating Basic Visualizations

Schema-Specific Visualizations

In Databricks SQL, you can create basic schema-specific visualizations to represent data visually. This includes tables, bar charts, line charts, and more. Below is an example of creating a table visualization:

```
SELECT department, SUM(sales) AS total_sales

FROM sales

GROUP BY department;
```

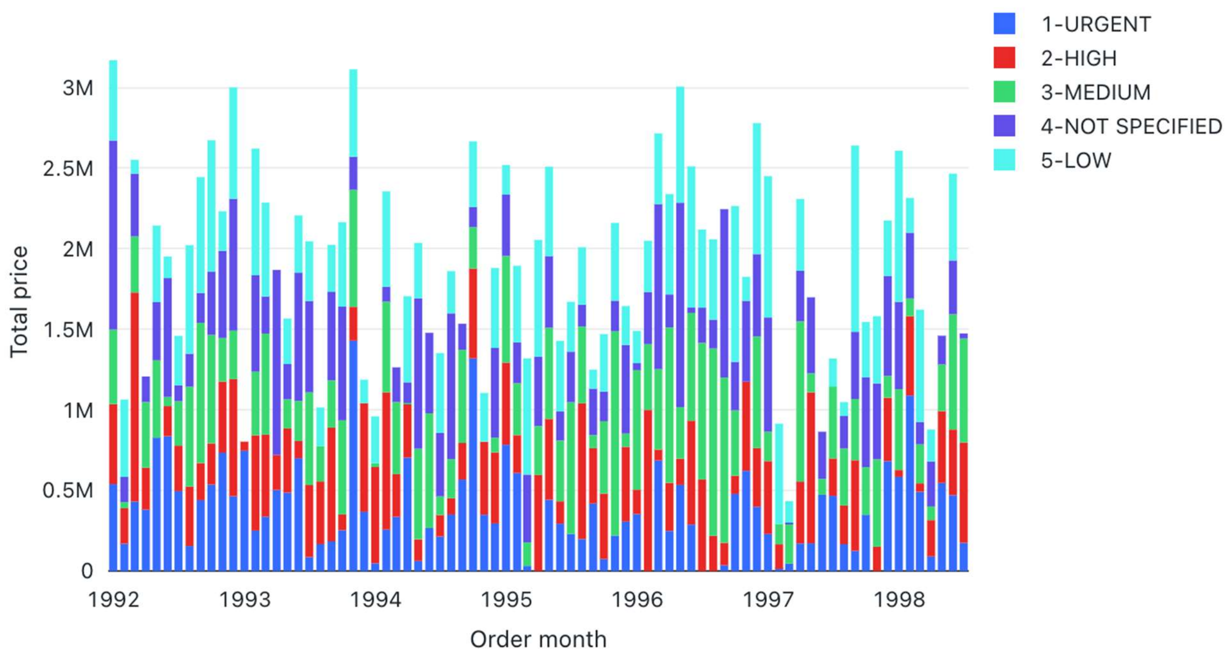
After executing the query, you can save the resulting visualization as a table.

Types of Visualizations in Databricks SQL

Available Visualization Types

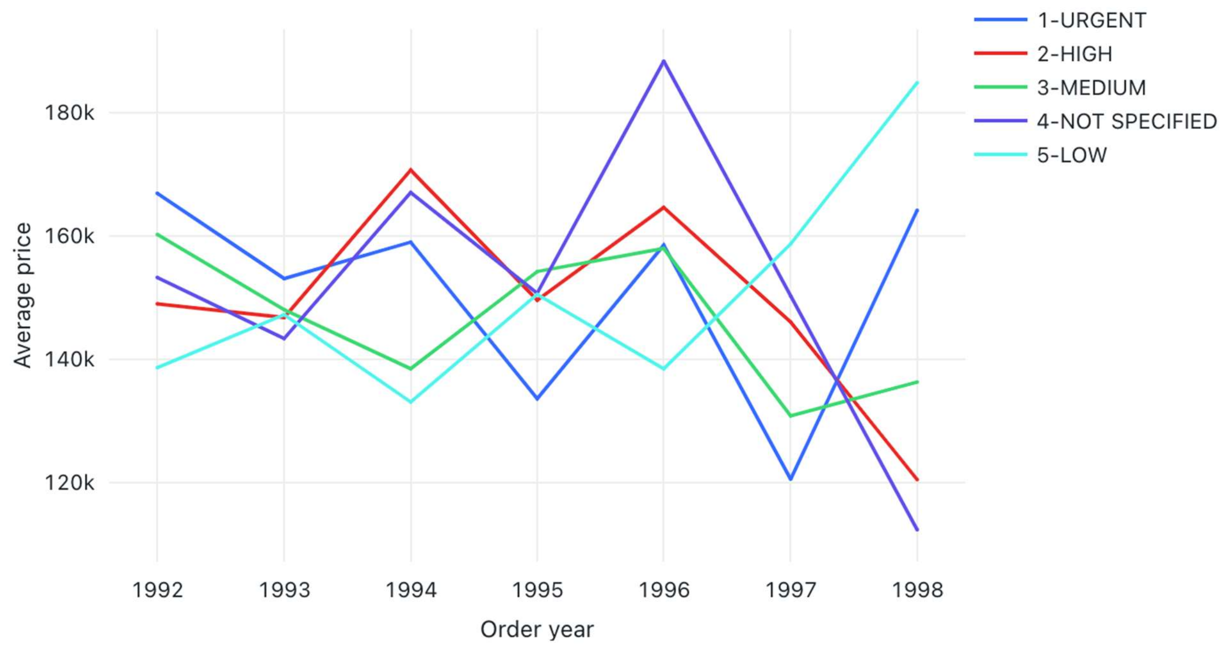
Databricks SQL offers various visualization types, each suitable for different types of analysis and storytelling with data:

1. **Table**: Display data in a tabular format for easy viewing of specific details.
2. **Bar Chart**: Compare values across different categories.



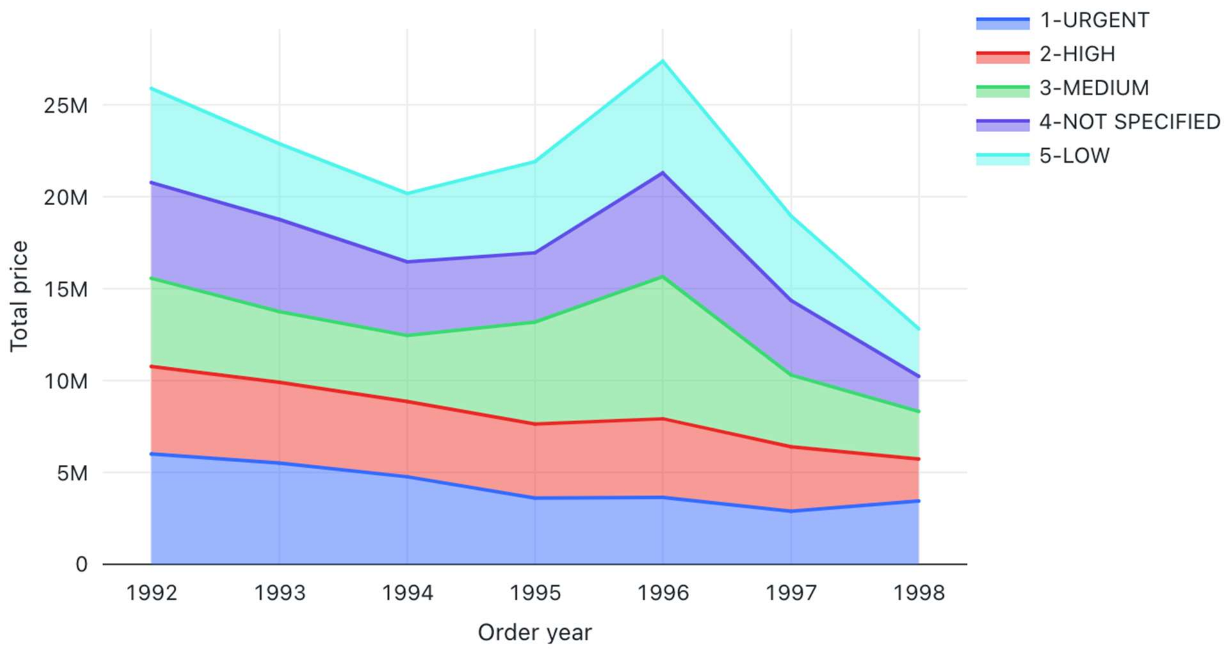
- Example: Sales by department.

3. **Line Chart**: Show trends over time.

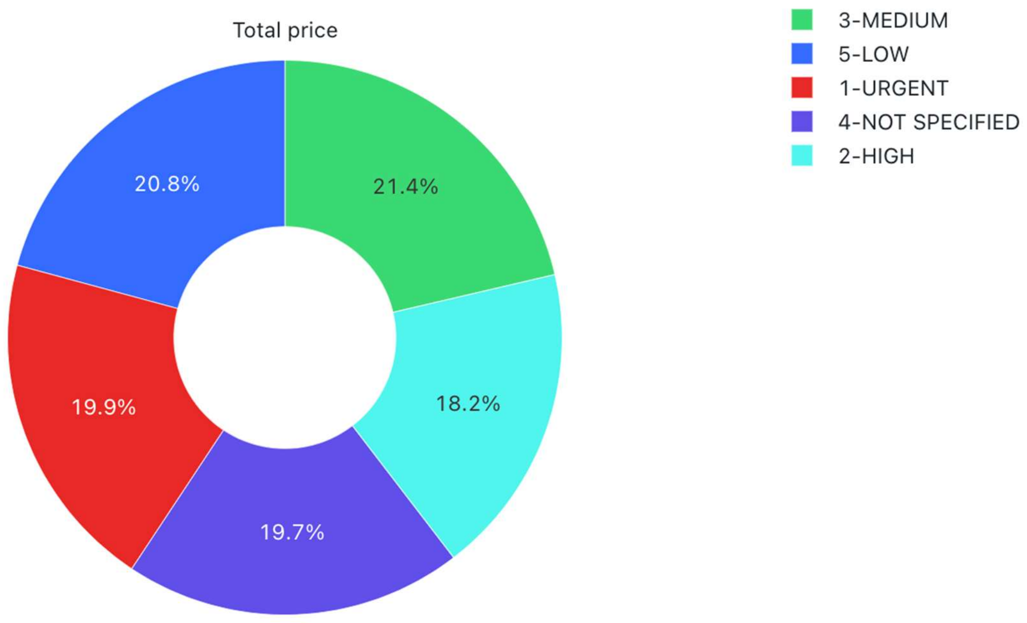


- Example: Monthly sales trend.

4. **Area Chart**: Area charts combine the line and bar chart to show how one or more groups’ numeric values change over the progression of a second variable, typically that of time. They are often used to show sales funnel changes through time.

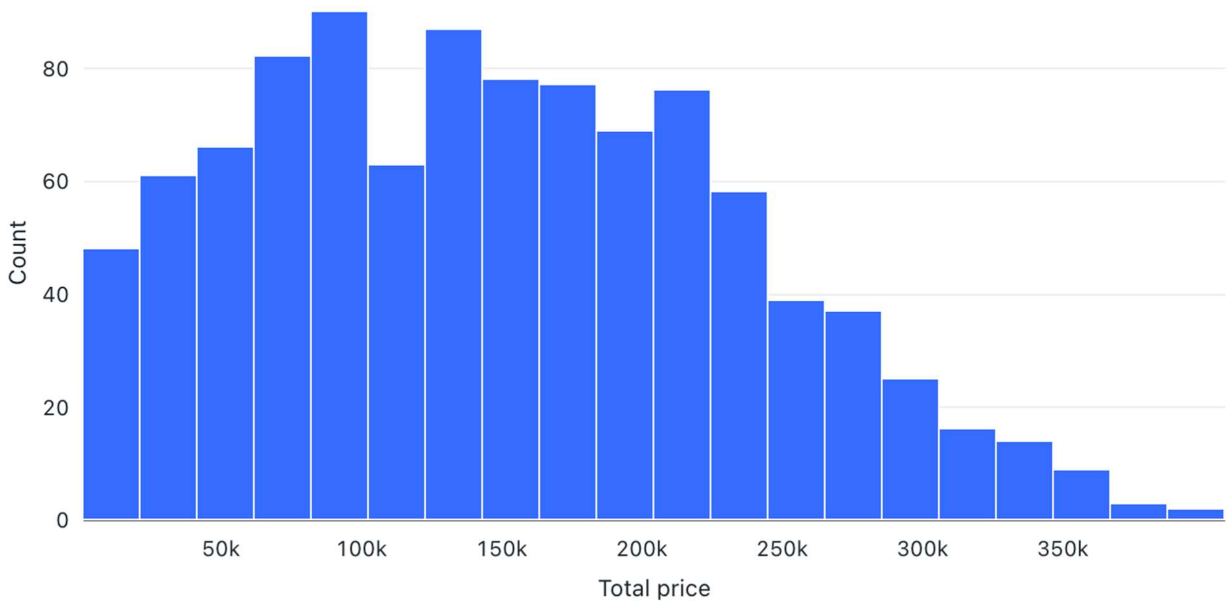


5. **Pie Charts**: Pie charts show proportionality between metrics. They are *not* meant for conveying time series data.

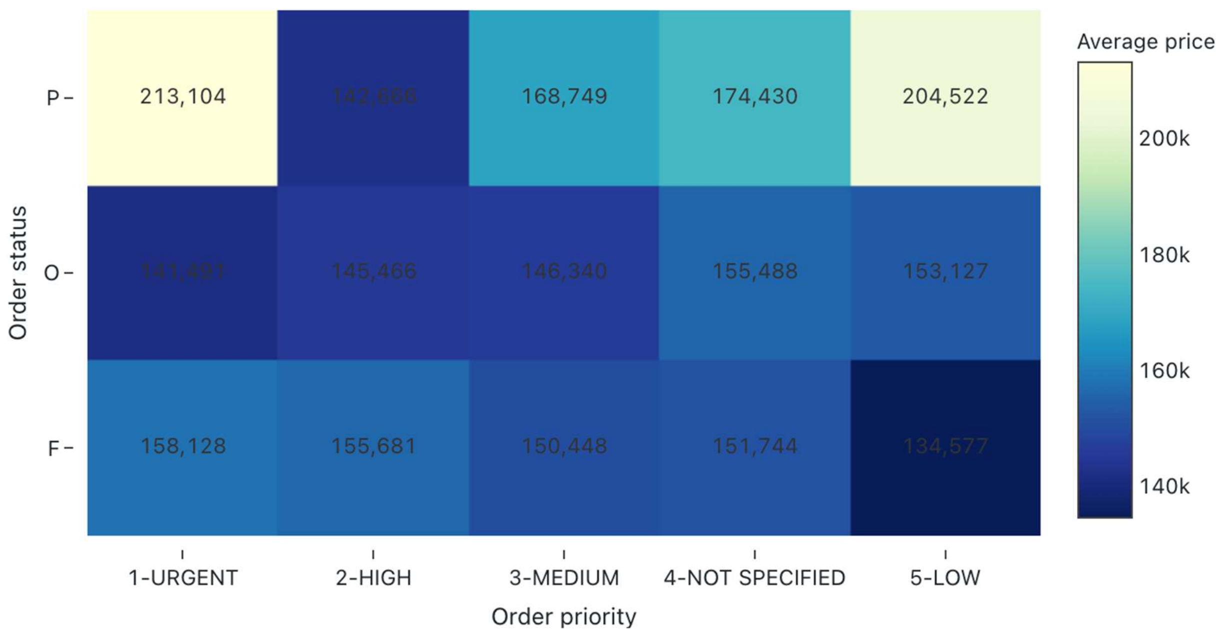


6. **Histogram charts**: A histogram plots the frequency that a given value occurs in a dataset. A histogram helps you to understand whether a dataset has values that are clustered around a small number of ranges or are more spread out. A histogram is displayed as a bar chart in which you control the number of distinct bars (also called bins).

Buy the complete book

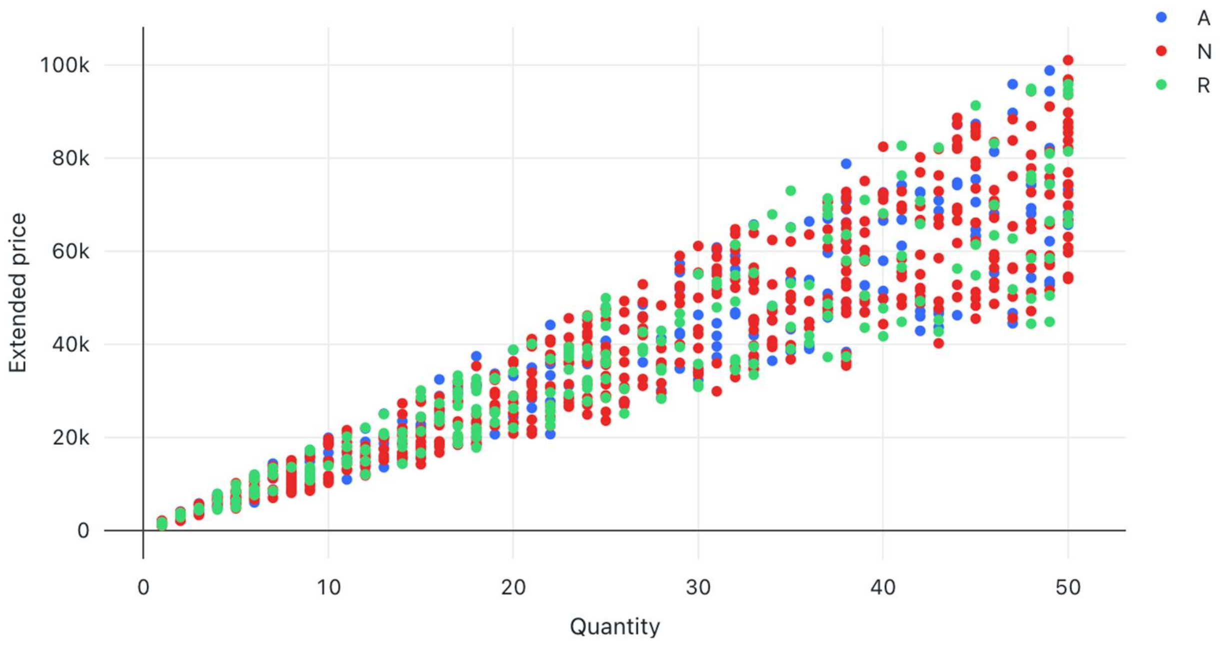


7. **Heatmap charts**: Heatmap charts blend features of bar charts, stacking, and bubble charts allowing you to visualize numerical data using colors. A common color palette for a heatmap shows the highest values using warmer colors, like orange or red, and the lowest values using cooler colors, like blue or purple.

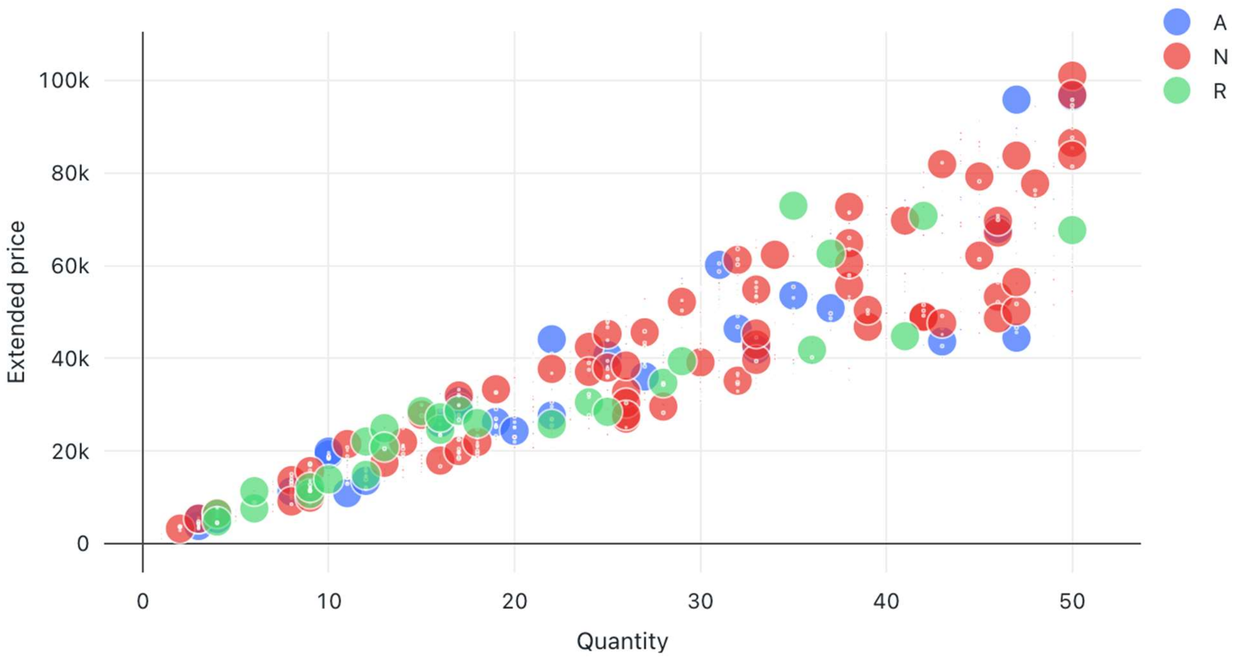


8. **Scatter chart**: Scatter visualizations are commonly used to show the relationship between two numerical variables. Additionally, a third dimension can be encoded with color to show how the numerical variables are different across groups.

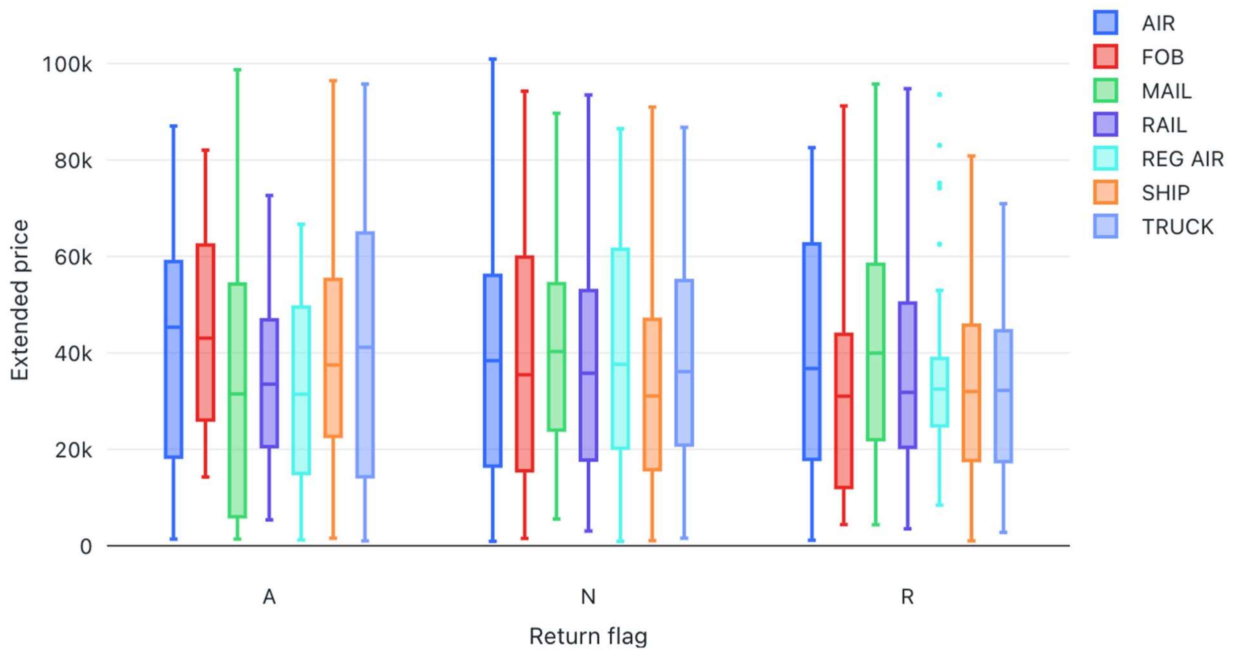
Buy the complete book



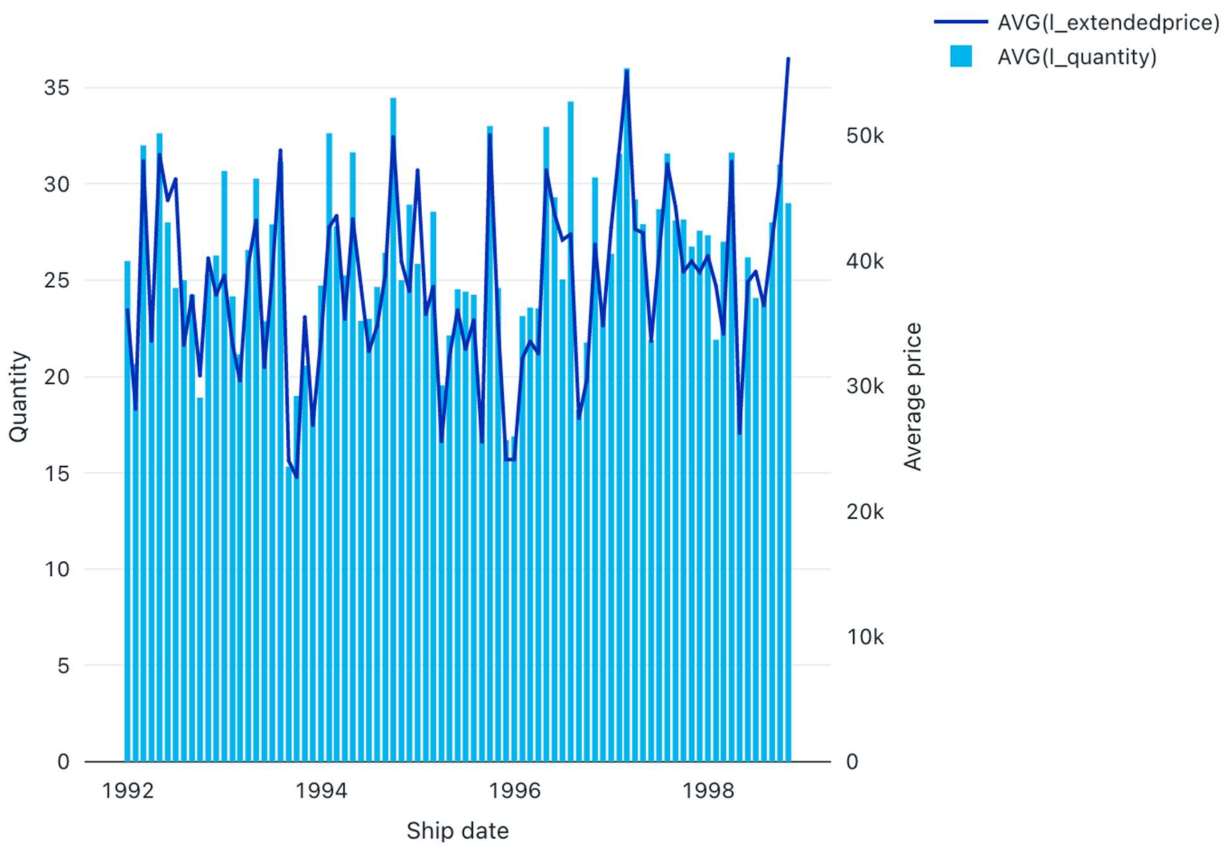
9. **Bubble Chart**: Bubble charts are scatter charts where the size of each point marker reflects a relevant metric.



10. **Box chart**: The box chart visualization shows the distribution summary of numerical data, optionally grouped by category. Using a box chart visualization, you can quickly compare the value ranges across categories and visualize the locality, spread and skewness groups of the values through their quartiles. In each box, the darker line shows the interquartile range.

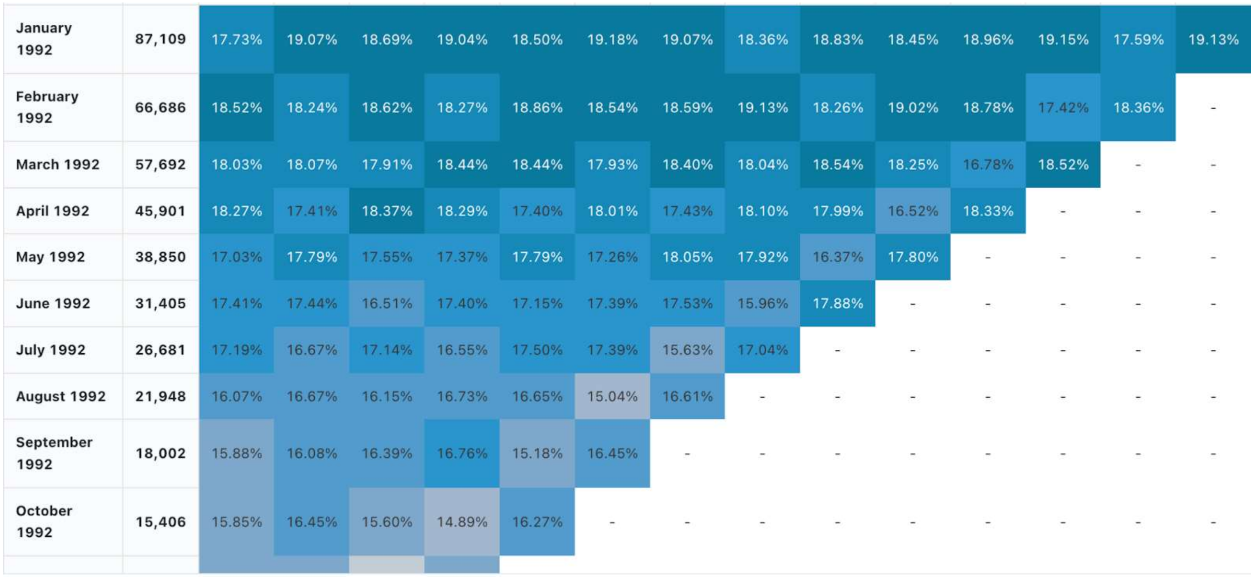


11. **Combo chart**: Combo charts combine line and bar charts to present the changes over time with proportionality.



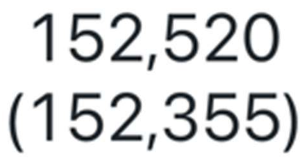
12. **Cohort analysis**: A cohort analysis examines the outcomes of predetermined groups, called cohorts, as they progress through a set of stages. The cohort visualization only

aggregates over dates (it allows for monthly aggregations). It does not do any other aggregations of data within the result set. All other aggregations are done within the query itself.



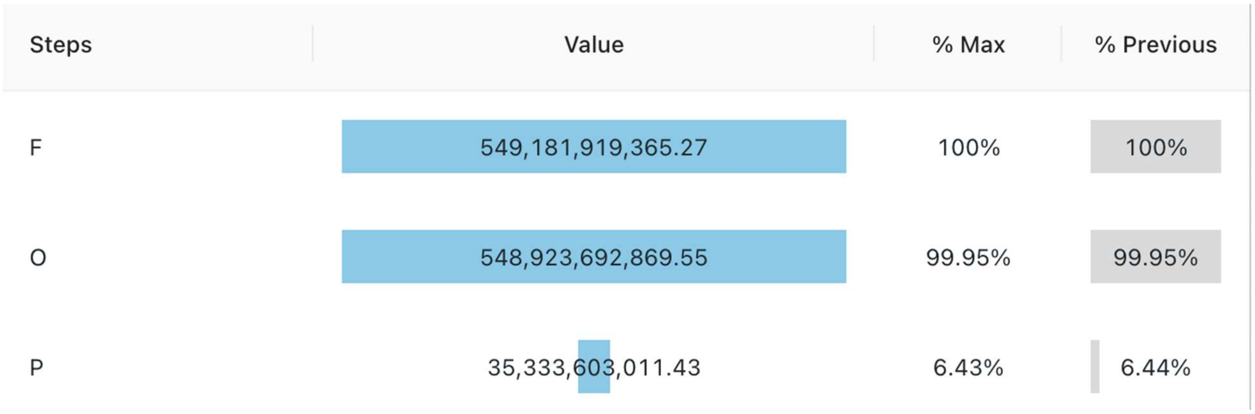
January 1992	87,109	17.73%	19.07%	18.69%	19.04%	18.50%	19.18%	19.07%	18.36%	18.83%	18.45%	18.96%	19.15%	17.59%	19.13%
February 1992	66,686	18.52%	18.24%	18.62%	18.27%	18.86%	18.54%	18.59%	19.13%	18.26%	19.02%	18.78%	17.42%	18.36%	-
March 1992	57,692	18.03%	18.07%	17.91%	18.44%	18.44%	17.93%	18.40%	18.04%	18.54%	18.25%	16.78%	18.52%	-	-
April 1992	45,901	18.27%	17.41%	18.37%	18.29%	17.40%	18.01%	17.43%	18.10%	17.99%	16.52%	18.33%	-	-	-
May 1992	38,850	17.03%	17.79%	17.55%	17.37%	17.79%	17.26%	18.05%	17.92%	16.37%	17.80%	-	-	-	-
June 1992	31,405	17.41%	17.44%	16.51%	17.40%	17.15%	17.39%	17.53%	15.96%	17.88%	-	-	-	-	-
July 1992	26,681	17.19%	16.67%	17.14%	16.55%	17.50%	17.39%	15.63%	17.04%	-	-	-	-	-	-
August 1992	21,948	16.07%	16.67%	16.15%	16.73%	16.65%	15.04%	16.61%	-	-	-	-	-	-	-
September 1992	18,002	15.88%	16.08%	16.39%	16.76%	15.18%	16.45%	-	-	-	-	-	-	-	-
October 1992	15,406	15.85%	16.45%	15.60%	14.89%	16.27%	-	-	-	-	-	-	-	-	-

13. **Counter display**: Counters display a single value prominently, with an option to compare them against a target value. To use counters, specify which row of data to display on the counter visualization for the Value Column and Target Column.



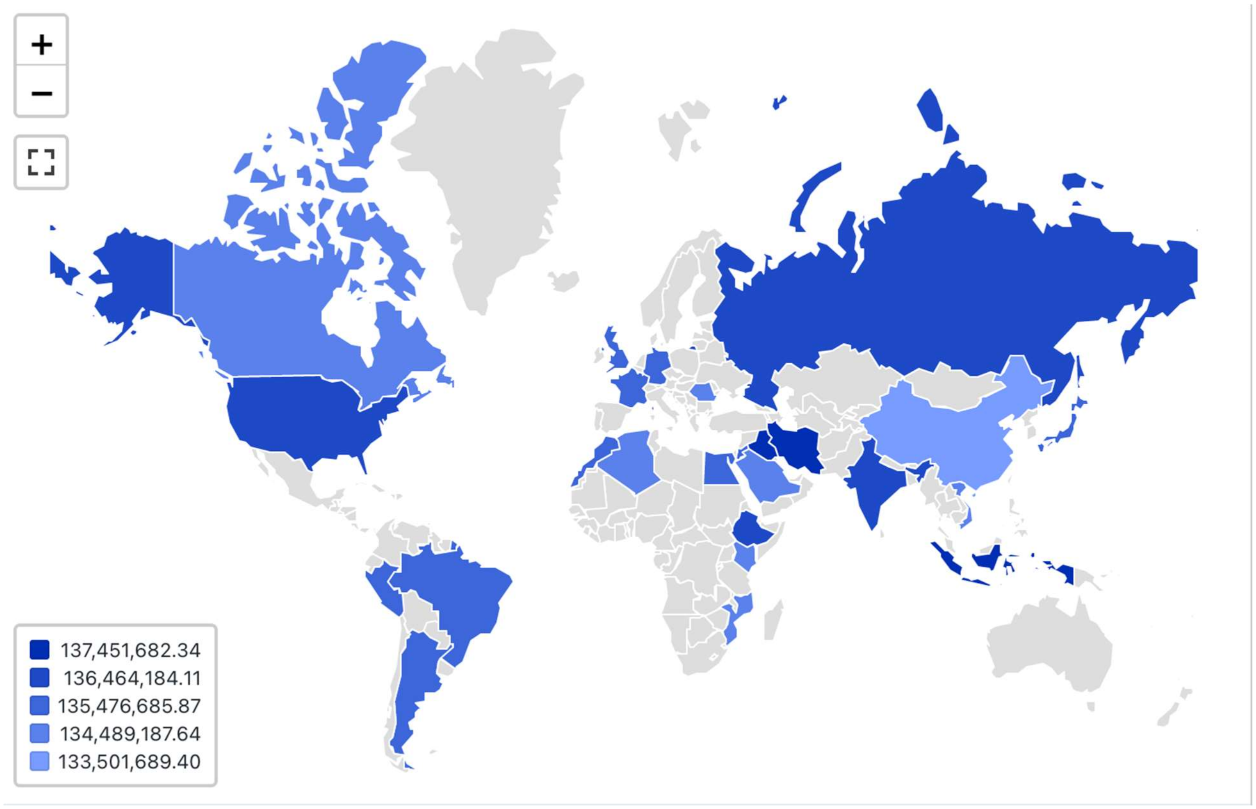
152,520
(152,355)

14. **Funnel visualization**: The funnel visualization helps analyze the change in a metric at different stages. To use the funnel, specify a step and a value column.

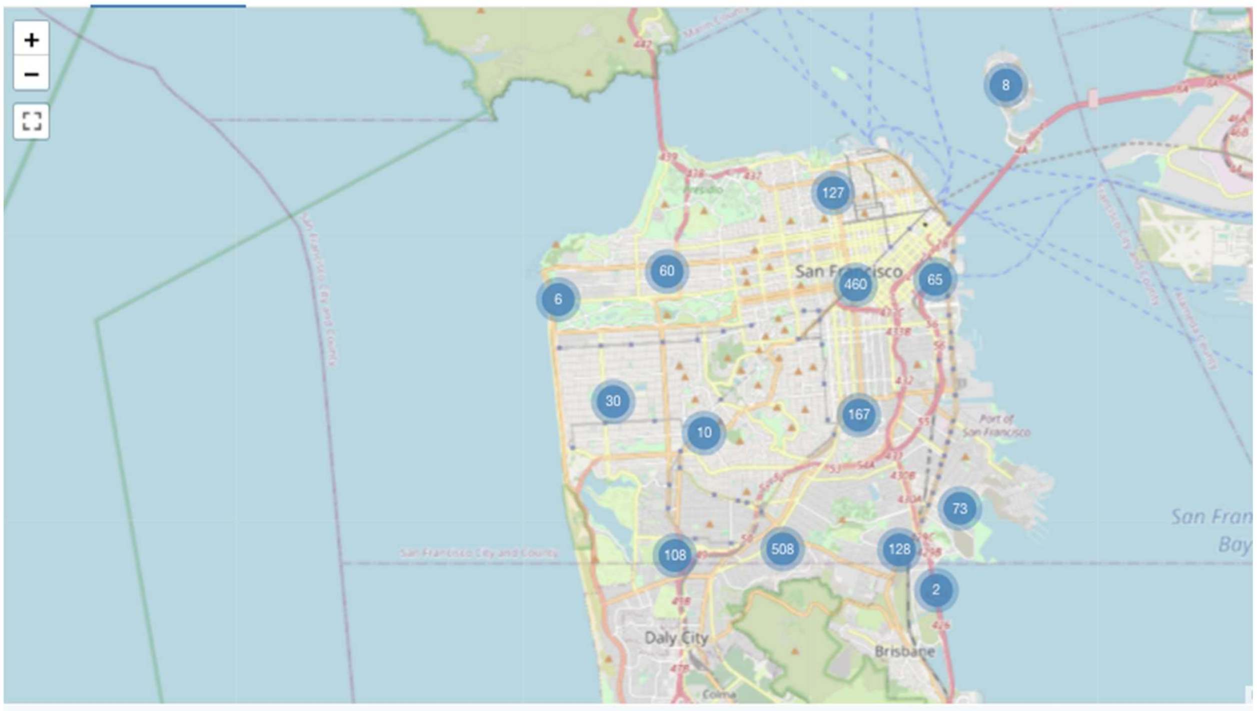


Steps	Value	% Max	% Previous
F	549,181,919,365.27	100%	100%
O	548,923,692,869.55	99.95%	99.95%
P	35,333,603,011.43	6.43%	6.44%

15. **Choropleth map visualization**: In choropleth visualizations, geographic localities, such as countries or states, are colored according to the aggregate values of each key column. The query must return geographic locations by name.



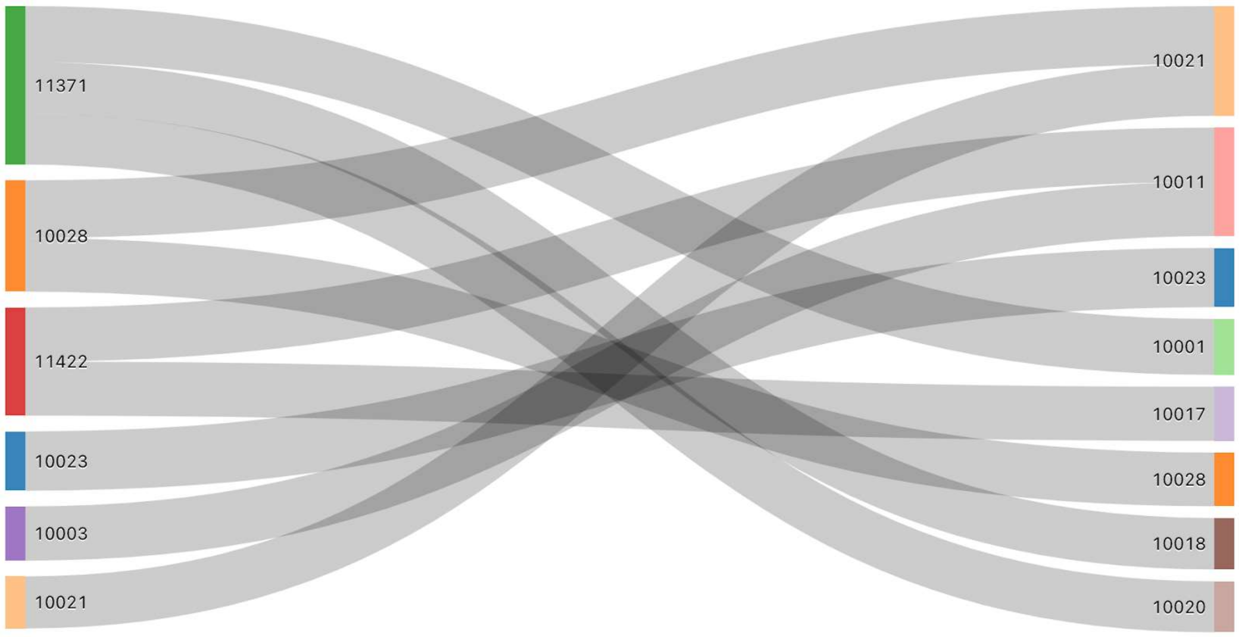
16. **Marker map visualization**: In marker visualizations, a marker is placed at a set of coordinates on the map. The query result must return latitude and longitude pairs.



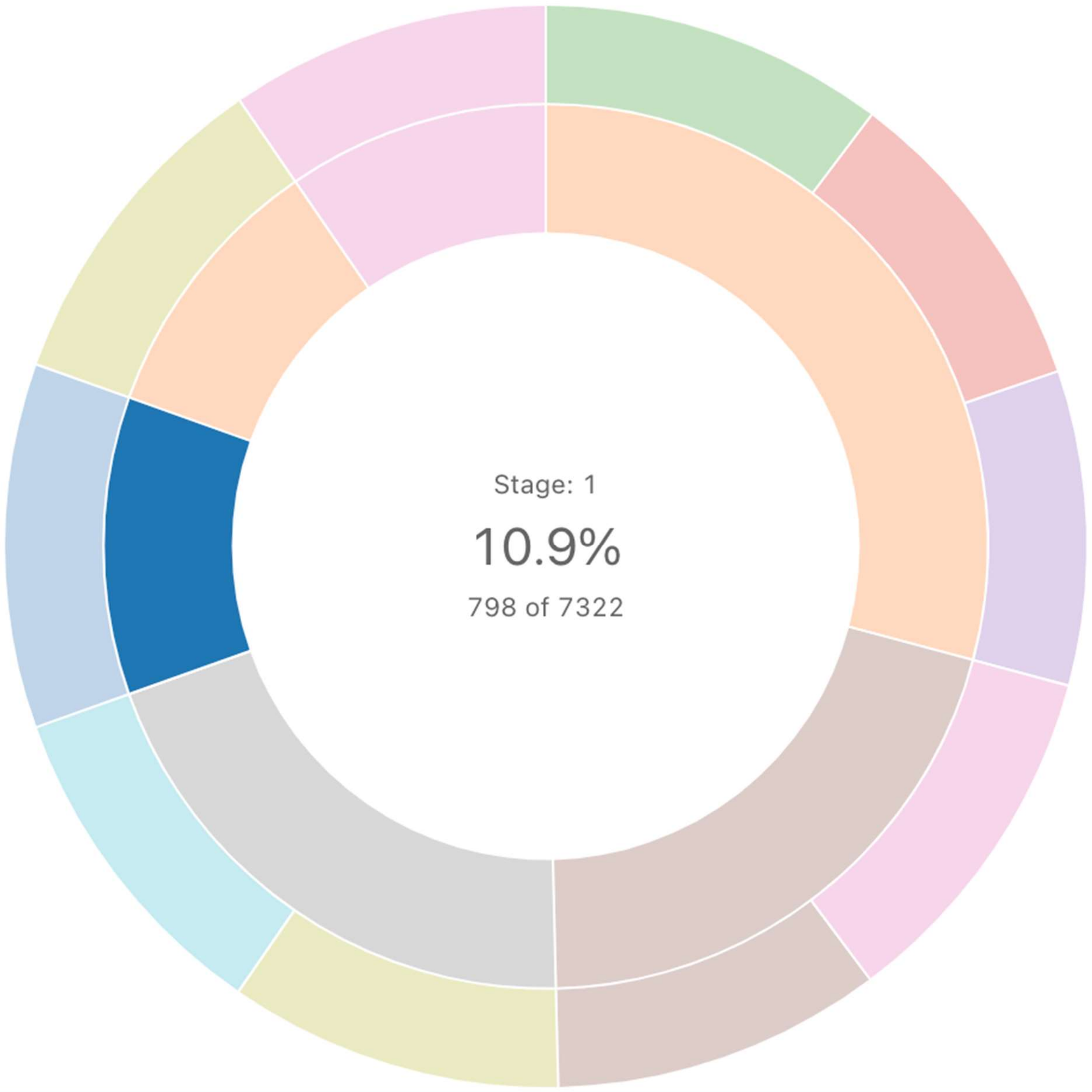
17. **Pivot table visualization**: A pivot table visualization aggregates records from a query result into a new tabular display. It’s similar to PIVOT or GROUP BY statements in SQL. You configure the pivot table visualization with drag-and-drop fields.

Buy the complete book

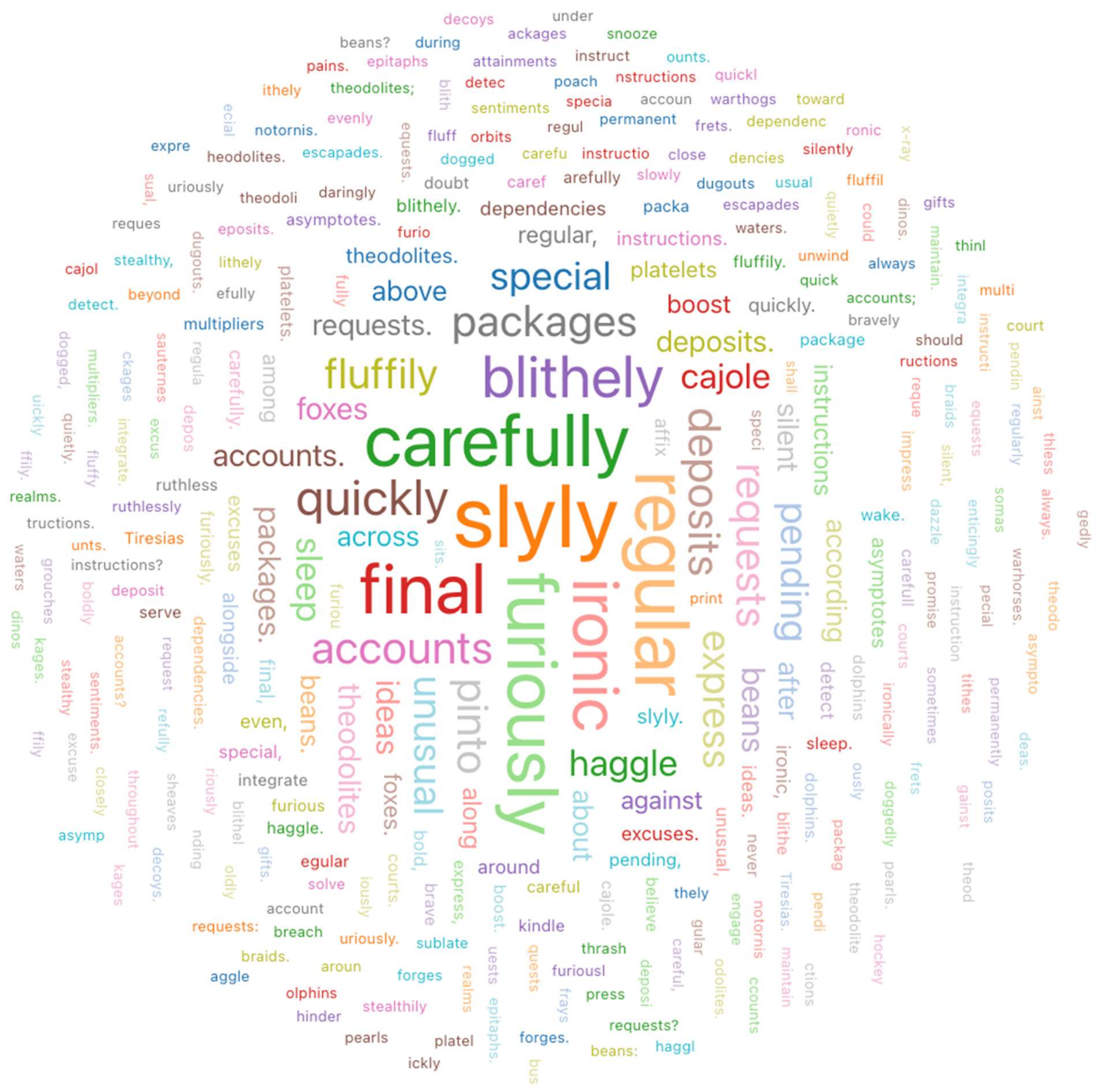
18. **Sankey**: A sankey diagram visualizes the flow from one set of values to another.



19. **Sunburst sequence**: A sunburst diagram helps visualize hierarchical data using concentric circles.



20. **Word cloud**: A word cloud visually represents the frequency a word occurs in the data.



Formatting Visualizations

Impact of Formatting on Visualization Reception

Proper formatting of visualizations can significantly improve the reception and understanding of the data. This includes using colors, clear labels, and organized layouts. For example, using consistent colors can help highlight patterns or anomalies in the data.

Adding Visual Appeal Through Formatting

Adding visual elements such as titles, legends, and axis labels can make visualizations more attractive and informative. For example:

- **Title**: Add a clear and descriptive title.
- **Axis Labels**: Ensure the axes are correctly labeled.
- **Colors**: Use a consistent color palette for readability.

Dashboards in Databricks SQL

Creating Dashboards

Dashboards allow you to combine multiple visualizations into a single interface. To create a dashboard in Databricks SQL:

1. **Create Visualizations**: Run your queries and save the desired visualizations.
2. **Group Visualizations in a Dashboard**: In the "Dashboards" menu, select "Create Dashboard" and add the visualizations. Drag and drop to organize the layout.
3. **Configuring Colors for Visualizations**: You can change the colors of all visualizations in a dashboard to maintain visual consistency.

Query Parameters in Dashboards

Changing Query Output with Parameters

Query parameters allow users to dynamically change the output of queries within a dashboard. This is useful for creating interactive dashboards that respond to different user inputs.

Behavior of a Dashboard Parameter

A dashboard parameter can be configured to filter data or change the visualization based on user-selected values. For example, you can add a date parameter to view sales over different periods.

Using "Query Based Dropdown List"

A "Query Based Dropdown List" allows creating a dropdown list of parameters based on the output of another query. For example:

```sql
SELECT DISTINCT department FROM sales;
```

This query can generate a list of departments to select in the dashboard.

Sharing and Updating Dashboards

Dashboard Sharing Methods

Dashboards can be shared in various ways, such as public links or restricted access to specific users. It is important to consider access permissions when sharing dashboards.

Updating Dashboards

Dashboards can be configured to update automatically at defined intervals. This ensures that the data is always up-to-date without manual intervention.

Permissions for Updating Dashboards

Users without permission for all queries, databases, and endpoints can easily update a dashboard using the owner's credentials.

Update Schedule Configuration

You can configure an update schedule to ensure the dashboard is updated regularly, such as hourly or daily.

Behavior of Update Rate Lower Than Warehouse "Auto Stop"

If the update rate is lower than the warehouse "Auto Stop," there may be delays in data updates. It is important to configure appropriately to avoid this issue.

Alerts and Notifications

Basic Alert Configuration and Troubleshooting

Alerts can be configured to monitor specific conditions and send notifications when these conditions are met. For example, you can set up an alert to notify when sales fall below a certain value.

Sending Notifications

Notifications are sent based on the alert configuration. You can define recipients and notification methods, such as email or Slack messages.

Section 5: Analytical Applications

Descriptive Statistics

Comparison Between Discrete and Continuous Statistics

1. **Discrete Statistics**: Used to describe data that can be counted in distinct and separate values. Examples include item count sold, number of customers, etc.
2. **Continuous Statistics**: Used to describe data that can take any value within a continuous range. Examples include height, weight, delivery time, etc.

Description of Descriptive Statistics

Descriptive statistics are used to summarize and describe the main characteristics of a dataset. They include measures of central tendency (mean, median, mode), measures of dispersion (variance, standard deviation, range), and shape measures (skewness, kurtosis).

Key Statistical Moments

1. **Mean**: Sum of all values divided by the number of values.
2. **Median**: Central value that divides the dataset into two equal halves.
3. **Mode**: Value that occurs most frequently in the dataset.
4. **Variance**: Average of the squared deviations of values from the mean.
5. **Standard Deviation**: Square root of the variance.
6. **Skewness**: Measure of the symmetry of the data distribution.
7. **Kurtosis**: Measure of the "peakedness" of the data distribution.

Data Enhancement

Common Data Enhancement Applications

Data enhancement involves cleaning, transforming, and enriching data to make it more useful and accurate for analysis. This can include value normalization, filling missing values, error correction, and adding contextual data.

Data Enhancement Scenario

A common example of data enhancement is standardizing addresses in a customer database. Addresses can be standardized to a consistent format, correcting abbreviations and spelling errors, to improve the accuracy of location analyses.

Last-Mile ETL Execution

Last-mile ETL involves performing project-specific transformations on the data before its final use. This can include combining data from different sources, data aggregation, and custom calculations.

Data Blending

Blending Data from Two Sources

Data blending involves combining data from different sources to create a more complete and useful dataset. This can be done through joins and aggregations.

Data Blending Scenario

An example of data blending is combining sales data from a physical store with online sales data to get a complete view of sales performance. This allows for more comprehensive analyses and better insights.

Last-Mile Application

Project-Specific ETL

In the context of a specific project, last-mile ETL may involve performing final transformations and preparing the data for analysis or visualization. This ensures that the data is in the correct format and ready for use.

Statistics and Measures

Comparison of Statistical Measures

Comparing different statistical measures helps to better understand the distribution and characteristics of the data. For example, comparing the mean and median can reveal the presence of extreme values (outliers).

Analytical Application Example

An example of an analytical application is customer churn analysis, where descriptive statistics are used to identify patterns and factors contributing to customer loss. This can include analyzing metrics such as churn rate, average retention time, and the profile of customers who most frequently cancel.

Data Blending

Blending Data from Sources

Data blending is a technique that allows combining data from different sources into a single cohesive dataset. This can be done through data integration tools or custom scripts.

Beneficial Data Blending Scenario

A scenario where data blending is beneficial is marketing campaign analysis, where data from different channels (email, social media, paid ads) is combined to get a unified view of campaign effectiveness.

Final ETL Application

Last-Mile ETL in Specific Projects

Last-mile ETL refers to the final transformations and preparation of data before it is used for analysis or reporting. This can include data aggregation, final cleaning, and specific formatting for visualization tools.