Supercharge Your SQL Analysis with Python and DuckDB

Rethinking Data Analysis: A New Paradigm

Jesus LM

Nov, 2024

Abstract

This article explores the synergy between Python and DuckDB, a powerful in-memory database, to revolutionize SQL-based data analysis. By leveraging Python's extensive data science ecosystem and DuckDB's lightning-fast query execution, data professionals can significantly accelerate their workflows.

Table of Contents

Overview

In this post, we'll delve into the seamless integration of DuckDB with popular Python libraries, enabling efficient data ingestion, transformation, and analysis. Through practical examples, we demonstrate how to harness the full potential of DuckDB for complex SQL queries, real-time data processing, and exploratory data analysis. By the end of this post, readers will gain the knowledge and skills to supercharge their SQL analysis projects with Python and DuckDB.

Database Creation

Database: `retail_db`

Table: `retail_sales`

```
# Import libraries
import polars as pl
import duckdb as db
import plotly.express as px
```

```
# Create database
conn = db.connect('datasets/retail_db.db')
```

```
# Create table
conn.sql('''
    create table if not exists retail_sales (
        id INT,
        sale_date DATE,
        sale_time TIME,
        customer_id INT,
        gender VARCHAR(10),
        age INT,
        category VARCHAR(35),
        quantity INT,
        price_per_unit FLOAT,
        cogs FLOAT,
        total_sale FLOAT
        )
''')
```

Data Ingestion

```
# Insert data into table from csv file
conn.sql('''
    INSERT INTO retail_sales
    SELECT * FROM read_csv('datasets/sales.csv')
''')
```

Data Exploration and Cleaning

- Record Count: Determine the total number of records in the dataset.
- Customer Count: Find out how many unique customers are in the dataset.
- Category Count: Identify all unique product categories in the dataset.
- Null Value Check: Check for any null values in the dataset and delete records with missing data

```
# Show first 10 records
conn.sql('select * exclude(cogs) from retail_sales limit 10').pl()
```

id i32	sale_date date	sale_time time	customer_id i32	gender str	age i32	category str	quantity i32	price_per_unit f32	total_sale f32
1	2023-11-23	10:15:00	101	”Male”	35	”Electronics”	2	299.98999	599.97998
2	2023-11-23	10:30:00	102	”Female”	28	”Clothing”	3	49.990002	149.970001
3	2023-11-23	10:45:00	103	”Male”	42	”Books”	1	19.99	19.99
4	2023-11-23	11:00:00	104	”Female”	31	”Home Goods”	4	39.990002	159.960007
5	2023-11-23	11:15:00	105	”Male”	25	”Electronics”	1	999.98999	999.98999
6	2023-11-23	11:30:00	106	”Female”	45	”Clothing”	2	69.989998	139.979996
7	2023-11-23	11:45:00	107	”Male”	38	”Books”	3	14.99	44.970001
8	2023-11-23	12:00:00	108	”Female”	22	”Home Goods”	5	29.99	149.949997
9	2023-11-23	12:15:00	109	”Male”	33	”Electronics”	2	499.98999	999.97998
10	2023-11-22	12:30:00	110	”Female”	37	”Clothing”	1	99.989998	99.989998

Record count

```
conn.sql('select count(*) as records from retail_sales').pl()
```

records i64
448

Customer count

```
conn.sql('select count(distinct customer_id) customers from retail_sales').pl()
```

customers i64
55

Category count

```
conn.sql('select distinct category from retail_sales').pl()
```

category str
"Toys"
"Electronics"
"Books"
"Home Appliances"
"Groceries"
"Sports Equipment"
"Clothing"
"Home Goods"
"Beauty Products"

Null value check

```
conn.table('retail_sales').pl().null_count()
```

id u32	sale_date u32	sale_time u32	customer_id u32	gender u32	age u32	category u32	quantity u32	price_per_unit u32	cogs u32	total_sale u32
0	0	0	0	0	0	0	0	0	0	0

Data Analysis

Write a SQL query to retrieve all columns for sales made on '2023-11-23'

```
conn.sql('''
    select *
        exclude(cogs)
    from retail_sales
    where sale_date = '2023-11-23'
''').pl()
```

id i32	sale_date date	sale_time time	customer_id i32	gender str	age i32	category str	quantity i32	price_per_unit f32	total_sale f32
1	2023-11-23	10:15:00	101	"Male"	35	"Electronics"	2	299.98999	599.97998
2	2023-11-23	10:30:00	102	"Female"	28	"Clothing"	3	49.990002	149.970001
3	2023-11-23	10:45:00	103	"Male"	42	"Books"	1	19.99	19.99
4	2023-11-23	11:00:00	104	"Female"	31	"Home Goods"	4	39.990002	159.960007
5	2023-11-23	11:15:00	105	"Male"	25	"Electronics"	1	999.98999	999.98999
…	…	…	…	…	…	…	…	…	…
5	2023-11-23	11:15:00	105	"Male"	25	"Electronics"	1	999.98999	999.98999
6	2023-11-23	11:30:00	106	"Female"	45	"Clothing"	2	69.989998	139.979996
7	2023-11-23	11:45:00	107	"Male"	38	"Books"	3	14.99	44.970001
8	2023-11-23	12:00:00	108	"Female"	22	"Home Goods"	5	29.99	149.949997
9	2023-11-23	12:15:00	109	"Male"	33	"Electronics"	2	499.98999	999.97998

Write a SQL query to retrieve all transactions where the category is 'Clothing' and the quantity sold is more than 4 in the month of Nov-2022

```
conn.sql('''
    select *
        exclude(cogs)
    from retail_sales
    where category = 'Clothing'
        and extract('month' from sale_date) = '11'
        and quantity >= 2
''').pl()
```

id i32	sale_date date	sale_time time	customer_id i32	gender str	age i32	category str	quantity i32	price_per_unit f32	total_sale f32
2	2023-11-23	10:30:00	102	"Female"	28	"Clothing"	3	49.990002	149.970001
6	2023-11-23	11:30:00	106	"Female"	45	"Clothing"	2	69.989998	139.979996
14	2023-11-22	13:30:00	114	"Female"	27	"Clothing"	2	59.990002	119.980003
22	2023-11-20	15:30:00	122	"Female"	28	"Clothing"	2	49.990002	99.980003
26	2023-11-17	16:30:00	126	"Female"	36	"Clothing"	3	49.990002	149.970001
…	…	…	…	…	…	…	…	…	…

id i32	sale_date date	sale_time time	customer_id i32	gender str	age i32	category str	quantity i32	price_per_unit f32	total_sale f32
26	2023-11-17	16:30:00	126	”Female”	36	”Clothing”	3	49.990002	149.970001
30	2023-11-17	17:30:00	130	”Female”	42	”Clothing”	2	69.989998	139.979996
32	2023-11-16	14:15:00	456	”Female”	28	”Clothing”	3	49.990002	149.970001
40	2023-11-14	18:30:00	2859	”Female”	27	”Clothing”	4	39.990002	159.960007
48	2023-11-06	19:00:00	5443	”Female”	35	”Clothing”	2	59.990002	119.980003

Write a SQL query to calculate the total sales for each category

```
sales = conn.sql('''
    select
        category
        , round(sum(total_sale),2) as net_sale
        , count(*) as total_orders
    from retail_sales
    group by 1
    order by total_orders desc
''').pl()
```

```
sales
```

category str	net_sale f64	total_orders i64
”Electronics”	68878.32	96
”Clothing”	12557.76	96
”Books”	4133.04	80
”Home Goods”	8437.84	56
”Toys”	1639.04	24
”Beauty Products”	1359.44	24
”Home Appliances”	9599.76	24
”Groceries”	3069.84	24
”Sports Equipment”	3039.68	24

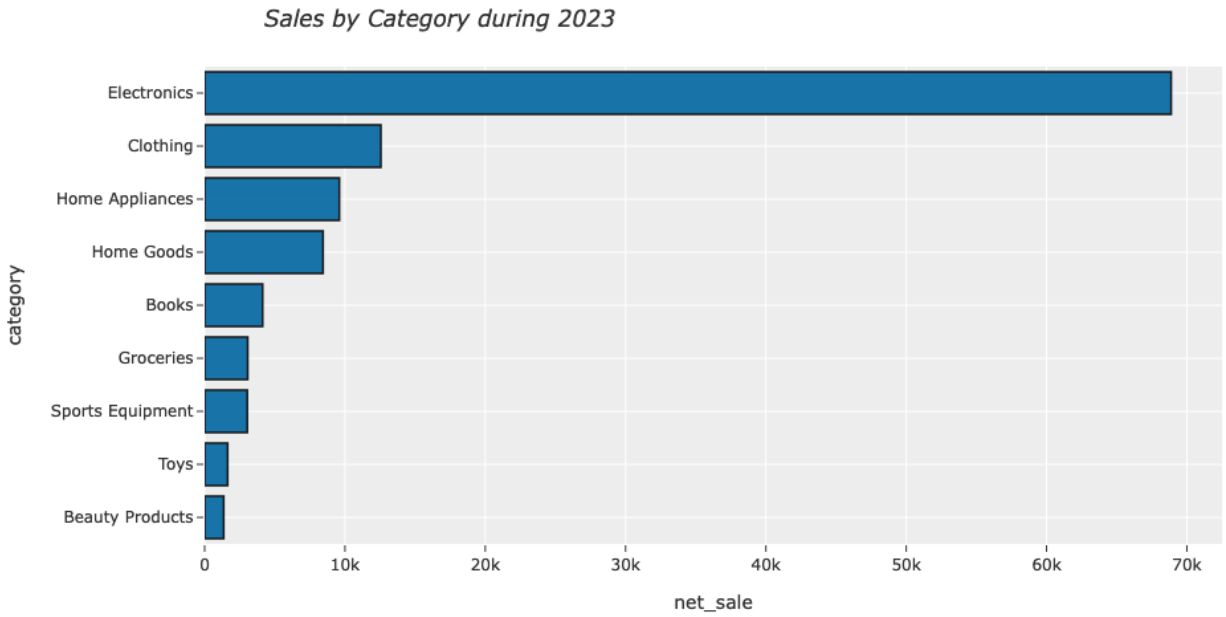
```
fig = px.bar(sales,
             x="net_sale",
             y="category",
             orientation='h',
             hover_data=['category','net_sale',],
            )

fig.update_traces(marker_color='#0066a1', marker_line_color='black',
                  marker_line_width=1.5, opacity=0.9)

fig.update_layout(width=850,
                  height=500,
                  title_text='<i>Sales by Category during 2023</i>',
                  title_x=0.2,
                  template="ggplot2",
                  yaxis={'categoryorder':'total ascending'}
                 )

fig.show()
```

```
Unable to display output for mime type(s): text/html
```



Write a SQL query to find the average age of customers who purchased items from the 'Clothing' category

```
conn.sql('''
    select
        round(avg(age), 2) as avg_age
    from retail_sales
    where category = 'Clothing'
''').pl()
```

avg_age
f64
33.25

Write a SQL query to find all transactions where the total_sale is greater than 1,000

```
conn.sql('''
    select *
        exclude(cogs)
    from retail_sales
    where total_sale > 999
''').pl()
```

id	sale_date	sale_time	customer_id	gender	age	category	quantity	price_per_unit	total_sale
i32	date	time	i32	str	i32	str	i32	f32	f32
5	2023-11-23	11:15:00	105	"Male"	25	"Electronics"	1	999.98999	999.98999
9	2023-11-23	12:15:00	109	"Male"	33	"Electronics"	2	499.98999	999.97998
29	2023-11-17	17:15:00	129	"Male"	27	"Electronics"	1	999.98999	999.98999
5	2023-11-23	11:15:00	105	"Male"	25	"Electronics"	1	999.98999	999.98999
9	2023-11-23	12:15:00	109	"Male"	33	"Electronics"	2	499.98999	999.97998
…	…	…	…	…	…	…	…	…	…
9	2023-11-23	12:15:00	109	"Male"	33	"Electronics"	2	499.98999	999.97998
29	2023-11-17	17:15:00	129	"Male"	27	"Electronics"	1	999.98999	999.98999
5	2023-11-23	11:15:00	105	"Male"	25	"Electronics"	1	999.98999	999.98999

id i32	sale_date date	sale_time time	customer_id i32	gender str	age i32	category str	quantity i32	price_per_unit f32	total_sale f32
9	2023-11-23	12:15:00	109	”Male”	33	”Electronics”	2	499.98999	999.97998
29	2023-11-17	17:15:00	129	”Male”	27	”Electronics”	1	999.98999	999.98999

Write a SQL query to find the total number of transactions made by each gender in each category

```
conn.sql('''
    select
        category
        , gender
        , count(*) as total_trans
    FROM retail_sales
    group by category
            , gender
    order by 2
''').pl()
```

category str	gender str	total_trans i64
”Clothing”	”Female”	96
”Home Goods”	”Female”	56
”Toys”	”Female”	24
”Home Appliances”	”Female”	24
”Beauty Products”	”Female”	24
”Groceries”	”Male”	24
”Electronics”	”Male”	96
”Sports Equipment”	”Male”	24
”Books”	”Male”	80

Write a SQL query to calculate the average sale for each month

```
conn.sql('''
    select
        year
        , month
        , avg_sale
    from
        (
        select
            extract(year from sale_date) as year
            , EXTRACT(month from sale_date) as month
            , round(avg(total_sale),2) as avg_sale
            , rank() over(partition by extract(year from sale_date)
            order by avg(total_sale) desc) as rank
        from retail_sales
        group by 1, 2
        ) as t1
''').pl()
```

year i64	month i64	avg_sale f64
2023	11	254.61
2023	10	198.3

year	month	avg_sale
i64	i64	f64

Write a SQL query to find the top 5 customers based on the highest total sales

```
conn.sql('''
    select customer_id
            , round(sum(total_sale),2) as total_sales
    from retail_sales
    group by 1
    order by 2 desc
    limit 5
''').pl()
```

customer_id	total_sales
i32	f64
129	7999.92
105	7999.92
109	7999.84
113	6399.92
117	6399.84

Write a SQL query to find the number of unique customers who purchased items from each category

```
customers = conn.sql('''
    select category
            , count(distinct customer_id) as count_unique
    from retail_sales
    group by category
    order by 2 desc
''').pl()
```

```
customers
```

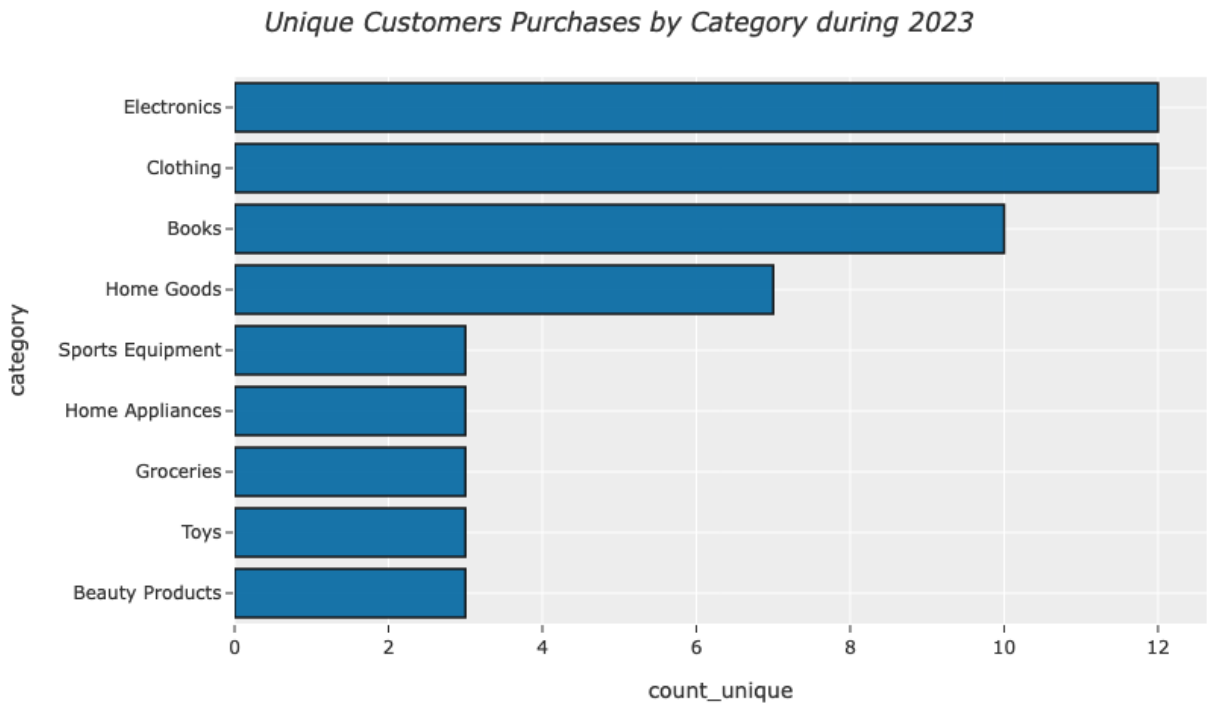
category	count_unique
str	i64
”Clothing”	12
”Electronics”	12
”Books”	10
”Home Goods”	7
”Beauty Products”	3
”Toys”	3
”Groceries”	3
”Home Appliances”	3
”Sports Equipment”	3

```
fig = px.bar(customers,
             x="count_unique",
             y="category",
             orientation='h',
             hover_data=['category','count_unique',],
            )
```

```
fig.update_traces(marker_color='#0066a1', marker_line_color='black',
                  marker_line_width=1.5, opacity=0.9)

fig.update_layout(width=850,
                  height=500,
                  title_text='<i>Unique Customers Purchases by Category during 2023</i>',
                  title_x=0.2,
                  template="ggplot2",
                  yaxis={'categoryorder':'total ascending'}
                 )

fig.show()
```



Write a SQL query to create each shift and number of orders (Example Morning <12, Afternoon Between 12 & 17, Evening >17)

```
conn.sql('''
    with hourly_sale as
        (
        select *
            , case
                when extract(hour from sale_time) <12 then 'Morning'
                when extract(hour from sale_time) between 12 and 17 then 'Afternoon'
            else 'Evening'
            end as shift
        from retail_sales
        )
    select
        shift
        , count(*) as total_orders
    from hourly_sale
    group by shift
```

```
''').pl()
```

shift str	total_orders i64
”Evening”	32
”Morning”	136
”Afternoon”	280

Close connection

```
# Close connection
conn.close()
```

Conclusions

This project demonstrates the power of combining Python and DuckDB for efficient and insightful SQL analysis. By mastering these tools, data analysts can streamline their workflows, uncover valuable insights, and make data-driven decisions that drive business success.

We've explored the fundamental concepts of SQL and its practical applications in data analysis. By leveraging the capabilities of Python and DuckDB, we've been able to efficiently query, clean, and analyze data. This knowledge and skillset can be applied to a wide range of data-driven tasks, from simple data exploration to complex predictive modeling. As we continue to delve deeper into the world of data, the combination of Python and DuckDB promises to be a powerful tool for data analysts.

These insights can be used to optimize marketing strategies, improve customer satisfaction, and drive revenue growth. As data continues to proliferate, the ability to harness its power through SQL analysis will become increasingly important for businesses to stay competitive.

Contact

Jesus LM *Economist & Data Scientist*

Medium | Linkedin | Twitter