Importing and Analyzing Parquet Files with DuckDB and Shiny

This guide demonstrates how to create a dashboard using DuckDB to analyze Parquet files with Shiny.

> **Note**
>
> **Note! This is a translation from Python (and Streamlit) to R (and Shiny). The original was written by Chanukya Pekala for DataTribe.**

In this example, we'll

- Create sales and product data Parquet files
- Read them using DuckDB
- Perform SQL queries to calculate
  - total revenue
  - best-selling products
  - daily sales revenue
- Visualize the results using ggplot2 and Shiny

This setup doesn't need to have a distributed system. It rather uses in-memory processing to analyze and report the data.

Prerequisites

Install the required packages:

```
install.packages(c("arrow", "duckdb", "shiny", "tidyverse"))
```

Load the required packages:

```
library(arrow)      # Read and write columnar data with Apache Arrow
library(duckdb)     # Lightweight in-process SQL analytics database
library(shiny)      # Build interactive web applications
library(tidyverse)  # Core packages for data wrangling
```

Data Structure

This example uses two Parquet files with the following schemas:

1. sales.parquet:

```
sales_data = {
"sale_id":    int,      # Unique identifier for each sale
"product_id": int,      # References product_id in products table
"quantity":   int,      # Number of items sold
"price":      float,    # Price per unit
"sale_date":  datetime  # Date of sale
}
```

2. products.parquet:

```
products_data = {
"product_id":   int,    # Unique identifier for each product
"product_name": str,    # Name of the product
"category":     str,    # Product category
"price":        float   # Standard price
}
```

Example of creating sample data files:

```
# Create sales data
sales_data <- tibble(
  # Unique identifier for each sale
  sale_id    = 1:5,
  # References product_id in products table
  product_id = c(101, 102, 101, 103, 104),
  # Number of items sold
  quantity   = c(2, 1, 5, 3, 4),
  # Price per unit
  price      = c(20.5, 35.0, 20.5, 15.75, 40.0),
  # Date of sale
  sale_date  = ymd(
    c("2024-03-01", "2024-03-02", "2024-03-03", "2024-03-04", "2024-03-05")
  )
) %>%
  # Ensure ID and count fields are stored as integers
  mutate(across(c(sale_id, product_id, quantity), as.integer))
```

```r
# Create products data
products_data <- tibble(
  # Unique identifier for each product
  product_id   = c(101, 102, 103, 104),
  # Name of the product
  product_name = c("Laptop", "Smartphone", "Headphones", "Monitor"),
  # Product category
  category     = c("Electronics", "Electronics", "Accessories", "Electronics"),
  # Standard price
  price        = c(1000.0, 500.0, 150.0, 300.0)
) %>%
  # Ensure product IDs are stored as integers
  mutate(product_id = product_id %>% as.integer)

# Write to Parquet files
write_parquet(sales_data, "sales.parquet")
write_parquet(products_data, "products.parquet")
```

Code Implementation

Here's how to create a sales analytics dashboard using DuckDB and Shiny.

Let's start by getting the data from the Parquet files we created in the previous step:

```r
# Create DuckDB connection
con <- dbConnect(duckdb())

# Read Parquet files directly using DuckDB
products_tbl <- dbGetQuery(
  con,
  "SELECT * FROM read_parquet('duckdb-shiny/products.parquet')"
)

sales_tbl <- dbGetQuery(
  con,
  "SELECT * FROM read_parquet('duckdb-shiny/sales.parquet')"
)

# Register DataFrames as tables
duckdb_register(con, "products_table", products_tbl)
duckdb_register(con, "sales_table", sales_tbl)
```

Let's then create a simple Shiny dashboard.

The dashboard will open in your default web browser, displaying:

- Bar chart of revenue by product
- Best-selling product details
- Line chart of daily sales revenue

```
# Create the Shiny dashboard

## UI
ui <- fluidPage(
  titlePanel("Sales Dashboard (R + Shiny + DuckDB)"),

  tabsetPanel(
    tabPanel("Revenue Per Product", plotOutput("revenuePlot")),
    tabPanel("Best-Selling Product", tableOutput("bestsellerTable")),
    tabPanel("Daily Sales Revenue", plotOutput("dailyPlot"))
  )
)

## Server
server <- function(input, output, session) {
  ### Query 1: Total Revenue per Product
  revenue_data <- reactive({
    dbGetQuery(
      con,
      "
      SELECT
        p.product_name,
        SUM(s.quantity * s.price) AS total_revenue
      FROM sales_table s
      JOIN products_table p ON s.product_id = p.product_id
      GROUP BY p.product_name
      ORDER BY total_revenue DESC
    "
    )
  })

  ### Query 2: Best-Selling Product
  bestseller_data <- reactive({
    dbGetQuery(
      con,
```

```
    "
    SELECT
      p.product_name,
      SUM(s.quantity) AS total_quantity_sold
    FROM sales_table s
    JOIN products_table p ON s.product_id = p.product_id
    GROUP BY p.product_name
    ORDER BY total_quantity_sold DESC
    LIMIT 1
  "
  )
})

### Query 3: Daily Sales Revenue
daily_revenue_data <- reactive({
  dbGetQuery(
    con,
    "
    SELECT
      sale_date,
      SUM(quantity * price) AS daily_revenue
    FROM sales_table
    GROUP BY sale_date
    ORDER BY sale_date
  "
  ) %>%
    mutate(sale_date = as_date(sale_date))
})

### Output 1: Total Revenue per Product
output$revenuePlot <- renderPlot({
  ggplot(
    revenue_data(),
    aes(x = reorder(product_name, total_revenue), y = total_revenue)
  ) +
    geom_col(fill = "steelblue") +
    coord_flip() +
    scale_y_continuous(
      breaks = seq(0, 150, 50),
      expand = c(0, 0.5)
    ) +
    labs(x = NULL, y = NULL, title = NULL) +
```

```
      theme_bw()
  })

  ### Output 2: Best-Selling Product
  output$bestsellerTable <- renderTable({
    bestseller_data() %>%
      rename(
        "Product Name" = product_name,
        "Total Quantity Sold" = total_quantity_sold
      )
  })

  ### Output 3: Daily Sales Revenue
  output$dailyPlot <- renderPlot({
    ggplot(daily_revenue_data(), aes(x = sale_date, y = daily_revenue)) +
      geom_line(color = "steelblue", size = 2) +
      scale_x_date(
        date_labels = "%Y-%m-%d",
        expand = c(0, 0.1)
      ) +
      labs(x = NULL, y = NULL, title = NULL) +
      theme_bw()
  })
}

# Run the app
shinyApp(ui, server)
```

Key Features

1. Direct Parquet File Reading: DuckDB can read Parquet files directly without first loading them into memory
2. SQL Queries: Complex analytics using standard SQL syntax
3. Visualizations: ggplot2 provides simple but effective chart components for data visualization within Shiny
4. Real-time Analysis: Dashboard updates automatically when data changes

Shiny Dashboard

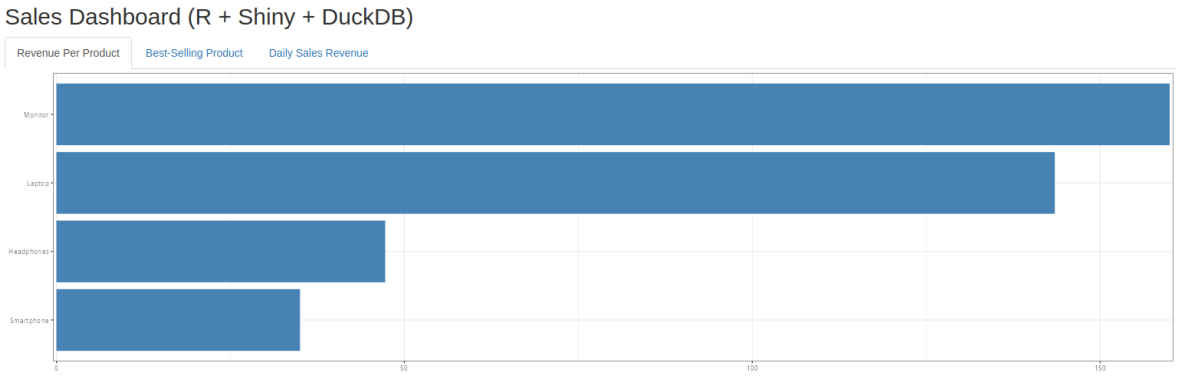

Figure 1: Revenue per product

Figure 2: Best selling product

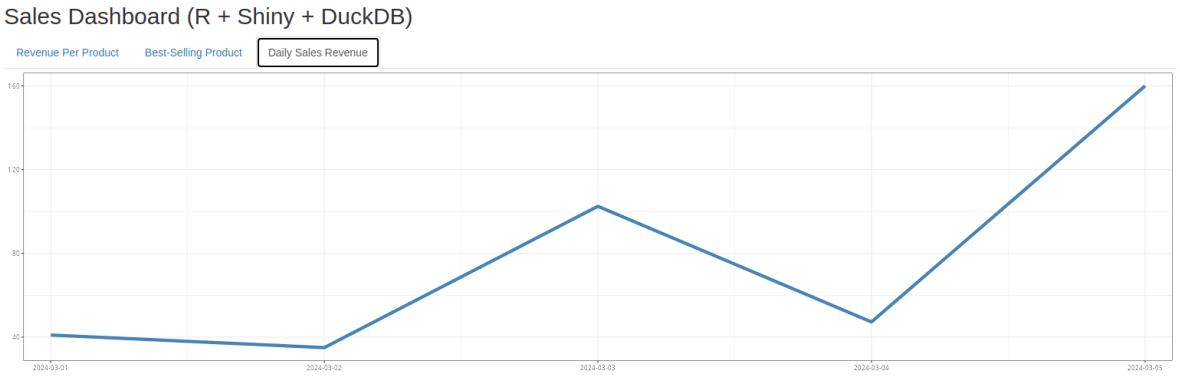

Figure 3: Daily sales revenue

Conclusion

This setup demonstrates the power of combining DuckDB with Shiny for local analytics:

- Fast and efficient SQL analytics without a distributed system
- Direct Parquet file processing with minimal memory overhead
- Simple, but effective visualizations through a user-friendly web interface
- Perfect for small to medium-sized datasets (up to several GBs)
- Zero-configuration analytics environment