Spark ETL and Retail Analytics Briefing

ETL Pipeline with Apache Spark: Briefing Document

This briefing document summarises the key themes, ideas, and facts extracted from the provided sources regarding an ETL (Extract, Transform, Load) pipeline implemented using Apache Spark, with a focus on retail transactional data analysis.

Core ETL Process with Apache Spark

Both sources demonstrate a clear, multi-step ETL pipeline using Apache Spark, highlighting two primary approaches: **PySpark DataFrames** and **Spark SQL**.

• **Extraction:** The initial step involves reading transactional data from an Excel file ("ETL Pipeline Dataset.xlsx").

◦ Using PySpark DataFrames: `df_pd = pd.read_excel("/content/ETL Pipeline Dataset.xlsx")` followed by `df = spark.createDataFrame(df_pd)`.

◦ Using Spark SQL: `df_pd = pd.read_excel("ETL Pipeline Dataset (1).xlsx")` followed by `df_spark = spark.createDataFrame(df_pd)` and then `df_spark.createOrReplaceTempView("retail_raw")`. This allows subsequent operations to be performed using SQL queries.

• **Transformation:** This phase focuses on cleaning and enriching the data.

◦ **Data Cleaning:** * Removing rows with null values in "InvoiceNo", "Description", and "CustomerID". * Filtering out transactions where "Quantity" is less than or equal to zero, as these are typically invalid or returns. * PySpark: `df_clean = df.dropna(subset=["InvoiceNo", "Description", "CustomerID"])`, `df_clean = df_clean.filter(df_clean["Quantity"] > 0)`. * Spark SQL: `CREATE OR REPLACE TEMP VIEW retail_clean AS SELECT * FROM retail_raw WHERE InvoiceNo IS NOT NULL AND Description IS NOT NULL AND CustomerID IS NOT NULL AND Quantity > 0`.

◦ **Feature Engineering/Data Enrichment:** * Calculating "TotalPrice" by multiplying "Quantity" and "UnitPrice". * Extracting "Year" and "Month" from the "InvoiceDate". * PySpark: `df_clean = df_clean.withColumn("TotalPrice", col("Quantity") * col("UnitPrice"))`, `df_clean = df_clean.withColumn("Year", year("InvoiceDate"))`, `df_clean = df_clean.withColumn("Month", month("InvoiceDate"))`. * Spark SQL: `CREATE OR REPLACE TEMP VIEW retail_enriched AS SELECT *, Quantity * UnitPrice AS TotalPrice, YEAR(InvoiceDate) AS Year, MONTH(InvoiceDate) AS Month FROM retail_clean`.

• **Loading:** The transformed data is prepared for further analysis or storage.

◦ The cleaned and enriched DataFrame can be converted to a Pandas DataFrame and saved as a CSV file: df_clean.toPandas().to_csv("Cleaned_ETL_Dataset.csv", index=False).

◦ In the Spark SQL example, the raw data is also saved as a table: df_spark.write.mode("overwrite").saveAsTable("retail_raw_table").

2. Key Analytical Insights and Data Visualisation

After the ETL process, the sources delve into various analytical tasks and visualisations to extract meaningful insights from the retail data.

Revenue Trends:

◦ **Monthly Revenue:** Both sources show a clear trend of monthly revenue, particularly for the United Kingdom. * PySpark output: Shows increasing revenue from December 2010 to November 2011, with a slight dip in December 2011. For instance, "2010| 12| 498661.85" and "2011| 11| 980645.74". * Spark SQL similarly visualises this trend with a line plot.

◦ **Daily Revenue Trend (Last 90 Days):** A line plot illustrates the daily revenue fluctuations, indicating short-term performance.

Product Performance:

◦ **Top Products by Quantity:** The most frequently purchased items are identified. * PySpark: "PAPER CRAFT , LIT..." (80995 units), "MEDIUM CERAMIC TO..." (76919 units), "WORLD WAR 2 GLIDE..." (49182 units) are among the top.

◦ **Top Products by Revenue:** The products generating the most sales are highlighted. * Spark SQL output for UK transactions lists "WHITE HANGING HEA..." as a top revenue generator.

Customer Behaviour:

◦ **Top Customers by Total Price:** Identifies high-value customers. * PySpark: CustomerID "18102.0" (259657.29), "17450.0" (194550.78), and "16446.0" (168472.5) are the top three.

◦ **Customer Spending Distribution:** A histogram visualises the distribution of total spending across customers, typically showing a long tail with a few high-spending customers.

◦ **Monthly Active Customers:** A line plot tracks the number of unique active customers each month, which is crucial for understanding customer engagement and growth.

◦ **Customer Lifetime Revenue Distribution:** A histogram or KDE plot showing the distribution of total revenue generated by each customer over their lifetime.

◦ **Customer Retention by Cohort:** A heatmap illustrating the percentage of customers retained over subsequent months based on their acquisition cohort, which is vital for understanding customer loyalty.

Geographical Analysis:

◦ **Revenue by Country:** A bar chart and a choropleth map demonstrate the revenue generated from different countries. * "United Kingdom" is consistently the highest revenue-generating country, with "7308391.55" in PySpark. "Netherlands" and "EIRE" follow.

◦ **Monthly Revenue by Country:** A heatmap visualises revenue across countries and months, offering insights into regional seasonality.

• Order and Transaction Analysis:

◦ **Invoice Size Distribution:** A histogram shows the distribution of total revenue per invoice, indicating typical transaction values.

◦ **Returns/Cancelled Orders:** The sources illustrate how to identify cancelled orders (those with "InvoiceNo" starting with 'C'). * The PySpark example shows a "Return Rate: 0.00%", which might indicate either no cancellations in the filtered dataset or an issue with the cancellation data itself, as the Spark SQL example confirms "No cancelled revenue data available" in its analysis.

◦ **Cancelled Revenue by Country:** A bar chart showing the countries with the highest cancelled revenue.

• Temporal Analysis (Day/Hour Heatmap):

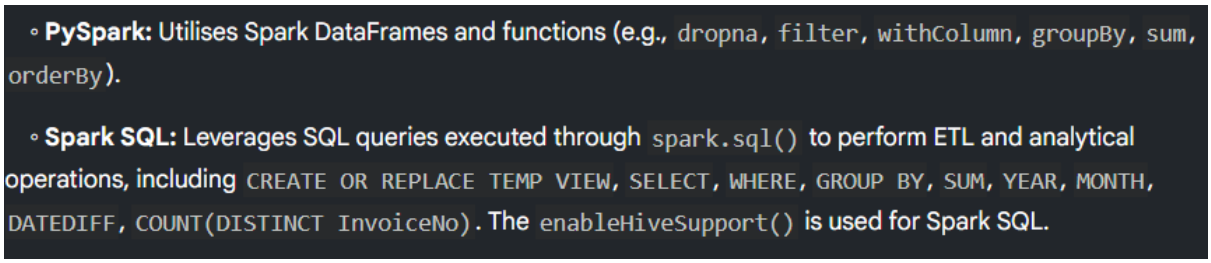
◦ A heatmap shows the "Revenue Heatmap: Day of Week vs Hour", revealing peak sales times and days. This can inform staffing and marketing strategies.

• Relationship between Quantity and Revenue:

◦ A scatter plot of "Revenue vs Quantity" helps understand if higher quantities always lead to proportionally higher revenue or if unit price plays a significant role.

3. Technical Implementation Details

• **Apache Spark:** Both sources use Apache Spark, highlighting its capabilities for distributed data processing.



• **Libraries:**

◦ **PySpark:** pyspark.sql for SparkSession and DataFrame operations.

◦ **Pandas:** pandas for reading Excel files and converting Spark DataFrames to Pandas for easier plotting (toPandas()).

◦ **Matplotlib and Seaborn:** matplotlib.pyplot and seaborn are extensively used for static data visualisation (line plots, bar charts, histograms, heatmaps, scatter plots).

◦ **Plotly Express:** plotly.express for interactive geographical visualisations (choropleth and scatter geo maps).

◦ **Statsmodels:** statsmodels.tsa.seasonal.seasonal_decompose for time series decomposition of daily revenue.

◦ **Scikit-learn:** sklearn.cluster.KMeans, sklearn.preprocessing.StandardScaler, sklearn.decomposition.PCA are mentioned for advanced customer segmentation (RFM and Clustering), although the full implementation is not explicitly shown in the provided excerpts for the clustering.

• **RFM Analysis (Recency, Frequency, Monetary):** This is mentioned as a technique for customer segmentation.

◦ Recency: Days since last purchase.

◦ Frequency: Number of distinct invoices.

◦ Monetary: Total revenue generated.

◦ Calculated using `DATEDIFF(MAX(InvoiceDate), MIN(InvoiceDate))` for Recency (though the PySpark example uses `(snapshot_date - x.max()).days`), `COUNT(DISTINCT InvoiceNo)` for Frequency, and `SUM(TotalPrice)` for Monetary. Heatmaps are used to visualise RFM metrics.

Important Facts and Observations

• The dataset contains columns such as "InvoiceNo", "StockCode", "Description", "Quantity", "InvoiceDate", "UnitPrice", "CustomerID", and "Country".

• The "InvoiceDate" column is crucial and is consistently converted to a datetime object for temporal analysis.

• The "Quantity" and "UnitPrice" are used to derive "TotalPrice", a fundamental metric for revenue analysis.

• The "United Kingdom" is the dominant country in terms of revenue contribution.

• Initial analysis on cancelled orders in both sources showed a "Return Rate: 0.00%" or "No cancelled revenue data available", suggesting that while the methodology for identifying them (InvoiceNo starting with 'C') is in place, the specific dataset used for the output might not contain such entries, or the Revenue for cancelled orders might be zero. This requires further investigation to confirm.

• The use of to_period('M') and dt.to_period('M') helps in monthly aggregation.

• spark.sparkContext.setLogLevel("ERROR") is used to suppress excessive output during Spark execution.

In conclusion, these sources provide a comprehensive guide to building an ETL pipeline and performing initial data analysis on retail transactional data using Apache Spark, demonstrating both DataFrame API and Spark SQL approaches, and leveraging powerful visualisation libraries for insights.