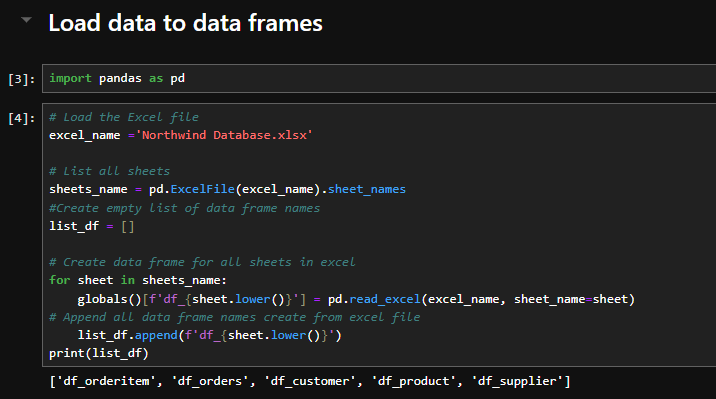
**Python**

**1. Load Data to data frames**

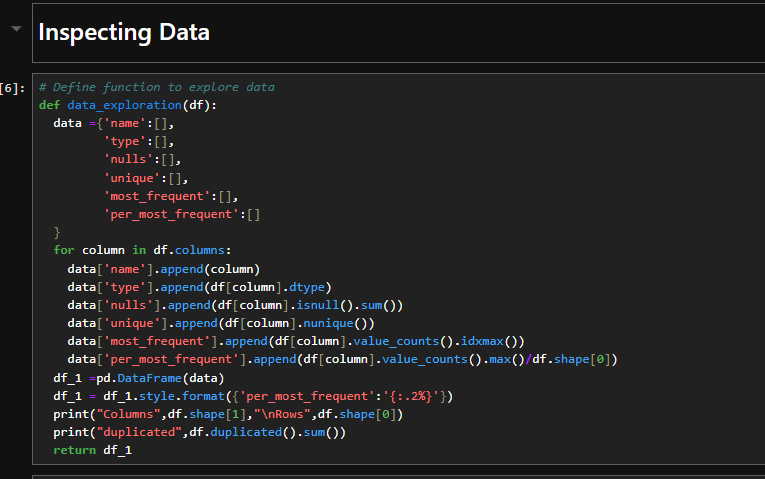
We began by importing the pandas library to facilitate data manipulation. A loop was implemented to load Excel file into separate data frames. In total, we created five data frames corresponding to different tables.



**2. Inspecting Data**

To understand the structure and quality of our data frames, we created a function to retrieve important metrics:

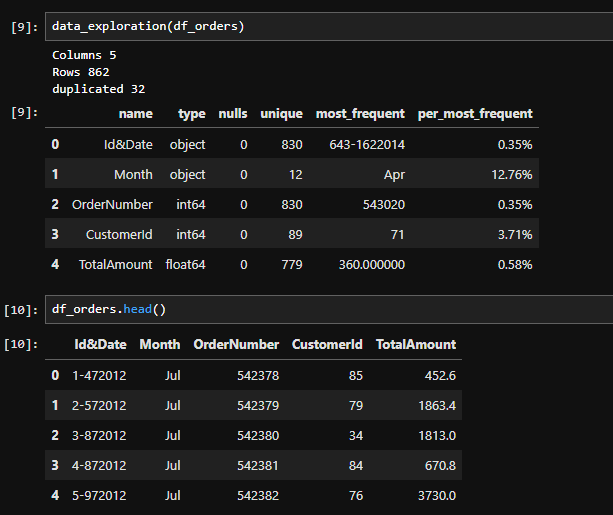
* Shape (rows and columns)
* Duplicates
* Column details: name, type, number of nulls, most frequent value, and percentage of the most frequent value.

****

**Function to Inspect Data**

**Findings**

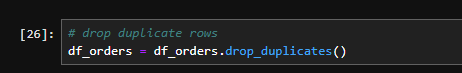
* **Duplications: We identified duplicates present in all data frames.**
* **“df\_orders” data frame: Notably, that data frame contained a column combining "Id" and "Date" which required splitting.**

****

**3. Data Cleaning**

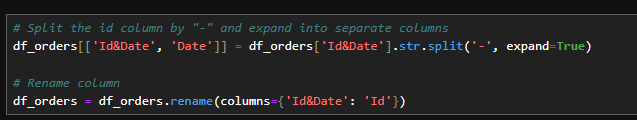
**Step 1: Remove Duplicates**

**We removed duplicates from all data frames to ensure data integrity.**

****

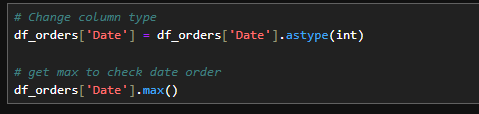
**Step 2: Split and Rename Columns in "df\_orders"**

**In the "df\_orders" data frame, we split the combined "Id&Date" column into two columns.**

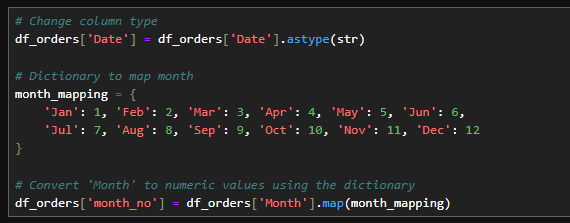
****

**Step 3: Date Manipulation**

1. **Change Data Type: The "Date" column's data type was changed to integer to get max to determine date format was "DMYYYY".**

****

1. **Create dictionary to map “Month” to number month in "month\_no".**

****

1. **Concatenate "month\_no” with the right 4 character in “Date” ( year) in new column "month\_year"**

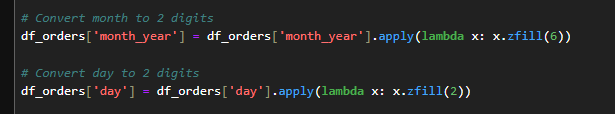
****

1. **Create a new column named "day," which will contain the count of characters in the "Date" column subtracted from the count of characters in the "month\_year" column.**

****

**Step 4: Formatting Dates**

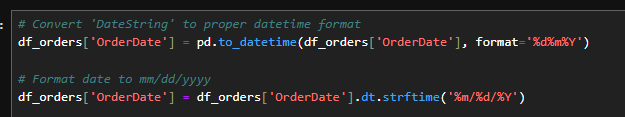
1. **Convert "month\_year" from "MYYYY" to "MMYYYY" and "day" from "D" to "DD". Finally, concatenate the "day" with "month\_year" in “OrderDate”.**

****

1. **Concatenate “day” to with “month\_year”.**



1. **Convert “OrderDate” string to proper datetime format.**



**Step 5: Drop Unnecessary Columns**

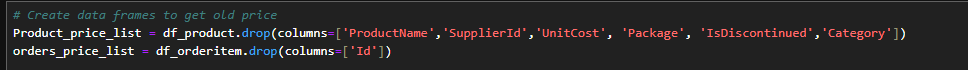
**We removed any columns that were not needed for further analysis.**

****

**4.** **Exploratory Data Analysis**

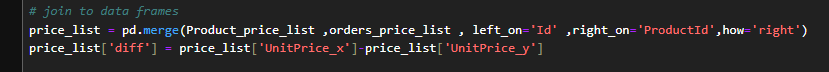
**Step 1: Data Preparation**

* **Initial Data Frames**:
  1. The code first creates two data frames:” Product\_price\_list” and “orders\_price\_list”.
  2. “Product\_price\_list” excludes non-essential columns such as “ProductName”, “SupplierId”, “UnitCost”, etc., while “orders\_price\_list” drops the Id column.



**Step 2: Data Merging**

* **Join Data Frames**:
  1. A right join is performed on the two data frames based on product IDs to create “price\_list”.
  2. The code calculates the price difference between the new and old prices.

****

**Step 3: Filtering and Cleaning Data**

* **Filtering Differences**:
  1. The code filters “price\_list” to keep only those records where there is a price difference.



* **Dropping Unnecessary Columns**:

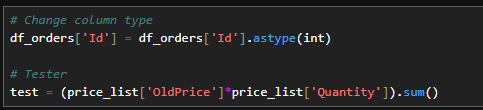
1. It drops columns that are no longer needed, including the price difference and duplicates.

A screenshot of a computer program

Description automatically generated

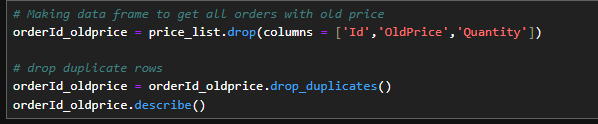
**Step 4: Type Conversion and Testing**

* **Type Conversion**:
  1. The Id column in “df\_orders” is converted to an integer type for consistency.
* **Testing Data Consistency**:
  1. A test is performed to ensure the computed sum of old prices multiplied by quantity matches the total amount for specific order IDs.



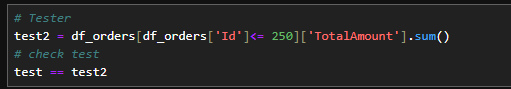
**Step 5: Further Data Manipulation**

* **Creating Additional Data Frames**:
  1. The code prepares a new data frame "orderId\_oldprice" for unique orders with old prices.



* **Calculating test2:**

1. You filter “df\_orders” to include only those rows where the 'Id' is less than or equal to 250.
2. You then sum the 'TotalAmount' for these filtered rows and store it in test2.
3. Comparing test and test2 to check if the value of test is equal to test2.

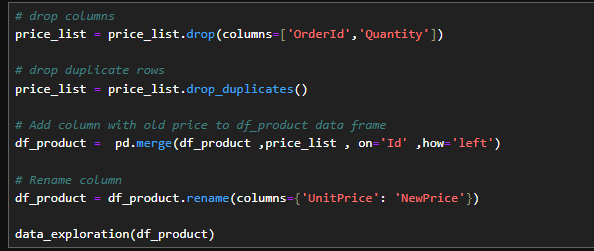


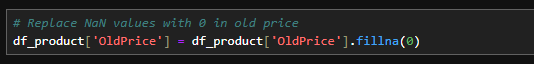
* **Adding Price Version**:
  1. A new column “PriceVersion” is added to categorize orders as 'Old' or 'New'.



**Step 6: Final Data Merging and Cleanup**

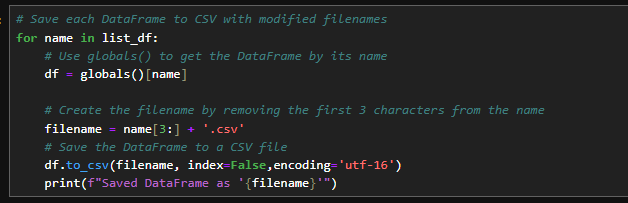
* **Merging with Original Product Data**:
  + The final step involves merging the “price\_list” with the original “df\_product” data frame to add the “OldPrice”.



****

**5. Save data frames to CSV**

**Finally, we saved all cleaned data frames to CSV format using UTF-16 encoding to ensure compatibility when uploading to SQL Server.**

****

**5. Summary**

**The steps taken to load, inspect, clean, and save data from Excel files into data frames. Each stage ensured that data quality was maintained, allowing for accurate further analysis. All data frames were successfully saved for future use.**

**SQL**

**1. Database Creation**

**Establish a new database for data storage and analysis.**

**2. Table Creation**

**Create tables within the database to store structured data.**

**3. Data Insertion from CSV Files**

**Import data from CSV files generated by Python scripts into the database.**

**4. Summary**

**The steps to create a database, establish a table, modify column names for better data representation, and import data from CSV files into the database. These actions will facilitate effective data modeling and enhance relationships within the dataset.**

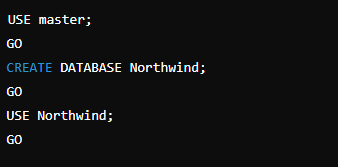
**SQL Report: Northwind Database Setup**

**Introduction**

**This report outlines the SQL code used to create and populate a database named Northwind. The database is structured to manage orders, customers, products, and suppliers.**

**Database Creation**

1. **Database Initialization**



**Table Definitions**

**1. OrderItem Table**

* **Purpose**: To store individual items in each order.
* **Columns**:
  + OrderItemid: Primary key.
  + OrderId: Foreign key referencing Orders.
  + ProductId: Foreign key referencing Product.
  + UnitPrice: Price of the product.
  + Quantity: Quantity of the product.

A screen shot of a computer code

Description automatically generated

**2. Orders Table**

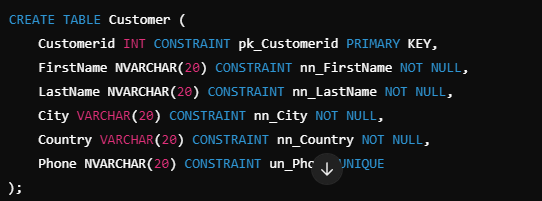
* **Purpose**: To store order details.
* **Columns**:
  + Orderid: Primary key.
  + OrderDate: Date of the order.
  + OrderNumber: Unique order number.
  + CustomerId: Foreign key referencing Customer.
  + TotalAmount: Total amount of the order.
  + PriceVersion: Version of the price.

A computer screen shot of a black screen

Description automatically generated

**3. Customer Table**

* **Purpose**: To store customer information.
* **Columns**:
  + Customerid: Primary key.
  + FirstName, LastName: Customer's names.
  + City, Country: Customer's location.
  + Phone: Unique phone number.



**4. Product Table**

* **Purpose**: To store product details.
* **Columns**:
  + Productid: Primary key.
  + ProductName: Name of the product.
  + Category: Category of the product.
  + SupplierId: Foreign key referencing Supplier.
  + NewPrice, UnitCost, OldPrice: Pricing details.
  + IsDiscontinued: Status of the product.

A computer screen with text on it

Description automatically generated

**5. Supplier Table**

* **Purpose**: To store supplier information.
* **Columns**:
  + Supplierid: Primary key.
  + CompanyName: Name of the company.
  + ContactName: Name of the contact person.
  + City, Country: Location of the supplier.
  + Phone: Contact number.

A screen shot of a computer code

Description automatically generated

**Data Relationships**

* Foreign keys were added to establish relationships between tables:
  + OrderItem references Orders and Product.
  + Orders references Customer.
  + Product references Supplier.

A screenshot of a computer program

Description automatically generated

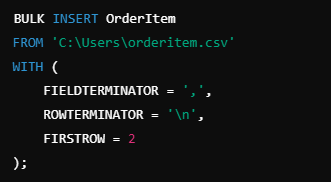
A screenshot of a computer screen

Description automatically generated

**Data Insertion**

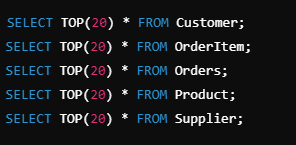
Data was inserted into each table using the BULK INSERT command. CSV files were used as the data source, with the first row skipped to avoid headers.

**Example of Data Insertion for OrderItem**

****

**Data Retrieval**

To verify the successful insertion of data, sample records were selected from each table:



**Conclusion**

The SQL code successfully creates the Northwind database, defining essential tables and relationships. Bulk data insertion allows for efficient population of the tables, facilitating further data analysis and operations. The structure supports the management of orders, customers, products, and suppliers in a coherent manner.

**Data Visualization**

**Introduction: Why We Chose Power BI**

We chose Power BI for this project because it offers powerful data visualization and reporting capabilities while allowing us to integrate and connect directly with SQL databases. By using Power BI, we could easily transform our cleaned and structured data into interactive dashboards. After using Python for data cleaning and SQL for data modeling, Power BI helped us ensure that the data relationships were accurately transferred before we focused on visualizing the insights.

**Step 1: Importing Data from SQL**

**For this project, we imported data directly from an SQL database into Power BI, allowing us to work with live and up-to-date data without manual file transfers.**

* Opened Power BI Desktop.
* Clicked "Get Data" from the Home ribbon and selected SQL Server as the data source.
* Entered the server and database details for the SQL connection.
* Used a Direct Query connection to ensure the data in Power BI remained updated in real time as it was refreshed from the SQL database.
* Selected the required tables from the database and clicked "Load" to import the data into Power BI.
* Once loaded, the data appeared in the Fields pane and was ready for use in creating visualizations.

**Step 2: Verifying Relationships in Power BI**

**Before creating visualizations, we ensured that the relationships between tables, created in SQL, were transferred correctly into Power BI.**

* Moved to the Model view to inspect the relationships between the tables.
* Checked that all relationships between tables were active and correct.
* Confirmed that the relationships mirrored the structure we had established in SQL, with all necessary links between tables in place.

**Step 3: Designing Visualizations**

In this step, we selected the most appropriate chart types for each data point to effectively convey insights and make the dashboard interactive and user-friendly.

To make our analysis comprehensive and user-friendly, we decided to analyze the data from four key perspectives: **Overview**, **Customers**, **Products**, and **Suppliers**. We implemented a navigator to organize the dashboard into these sections, allowing the client to switch between different views easily. Below is a breakdown of the visualizations within each perspective, explaining the chart types, why we chose them, and what insights they provide.

**1. KPI Cards – Total Orders, Total Revenue, and Total Profit**

* **Chart Type**: KPI (Key Performance Indicator) Cards
* **Why We Used It**: KPI cards summarize key metrics in a single view, providing at-a-glance information.
* **What It Displays**: These KPI cards display the total number of orders, total revenue, and total profit. The cards offer a quick summary of the overall performance metrics.

**2. Line Chart – Total Profit by Year**

* **Chart Type**: Line Chart
* **Why We Used It**: Line charts are effective for showing trends over time, making them perfect for displaying changes in profit year by year.
* A graph of a graph showing the growth of a company

  Description automatically generated with medium confidence**What It Displays**: The line chart tracks total profit from 2012 to 2014. The continuous line helps identify periods of growth or decline, making it easier to understand profit trends over multiple years.

**3.** **Donut Chart – Total Profit by Product Category**

* **Chart Type**: Donut Chart
* **Why We Used It**: Donut charts are useful for showing proportions of a whole. This chart visualizes how different product categories contribute to the overall profit.
* A blue and green pie chart

  Description automatically generated**What It Displays**: The chart breaks down total profit into categories such as Dairy Products, Beverages, Meat/Poultry, and Confections. The size of each slice indicates the percentage of total profit contributed by each product category, helping us quickly see which product lines are most profitable.

**4.** **Map Chart – Total Profit by Country**

* **Chart Type**: Map Chart
* **Why We Used It**: The map chart provides a geographic context for the data, allowing the client to see which countries are performing well and which are underperforming in terms of total profit.
* **A map of the world

  Description automatically generatedWhat It Displays**: By using color gradients, we can easily highlight significant differences in profit across regions, making it intuitive to identify high and low performers briefly. The client can hover over or click on specific countries to view detailed profit figures, enhancing the interactivity of the dashboard and providing deep

**5. Clustered Column Chart – Profit by Customer: Old vs. New Prices**

* **Chart Type**: Clustered Column Chart
* **Why We Used It**: Clustered column charts are great for side-by-side comparisons, making them ideal for showing profit under different pricing models for each customer.
* **What It Displays**: This chart compares the profit generated by individual customers under old and new pricing schemes. Each customer has two bars (old and new), allowing us to see how pricing changes have affected profitability for each customer.

A graph with numbers and text

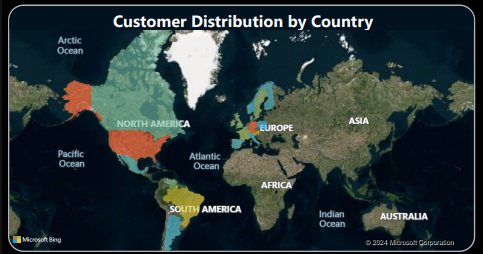
Description automatically generated with medium confidence

**6.** **Waterfall Chart – Revenue Across Time**

* **Chart Type**: Waterfall Chart
* **Why We Used It**: The waterfall chart effectively illustrates how revenue evolves over a specified period, highlighting both increases and decreases in revenue across different time intervals
* A graph showing a number of numbers and a graph

  Description automatically generated with medium confidence**What It Displays**: The chart visually connects each period, allowing the client to see how revenue is built up or reduced over time, providing a clear view of the journey from the initial revenue to the final figure.

**7. Map Chart – Customer Distribution by Country**

* **Chart Type**: Map Chart
* **Why We Used It**: Map charts are used to display geographic data, making them ideal for showing where customers are located around the world.
* **What It Displays**: This chart visualizes customer distribution by country. It provides a geographical perspective on customer bases, helping to identify regions with the highest concentration of customers and highlighting any underserved areas.

**8. Donut Chart – Total Cost by Product Category**

* **Chart Type**: Donut Chart
* **Why We Used It**: A Donut chart is well-suited for displaying proportions, showing how different product categories contribute to total costs.
* A chart with numbers and a circle

  Description automatically generated**What It Displays**: This chart breaks down the total costs by product category. It provides insight into which categories are driving the most expenses, helping to understand cost distribution relative to revenue and profit.

**9. Bar Chart – Total Cost by Country**

* **Chart Type**: Bar Chart
* **Why We Used It**:, A bar chart is effective for comparing costs across different countries.
* A graph of a number of countries/regions

  Description automatically generated**What It Displays**: This chart compares total costs incurred by country, making it easy to identify regions with the highest expenses. It helps to analyze cost distribution and identify areas where costs may need to be managed more efficiently.

**10. Combo Chart – Revenue and Cost by Product**

* **Chart Type**: Combo Chart (Bars for Cost, Line for Revenue)
* **Why We Used It**: A combo chart allows us to display two metrics on the same chart, making it easy to compare revenue and cost side-by-side for each product.
* **What It Displays**: This chart displays both total revenue and total cost for each product. It helps to quickly identify products that are profitable (higher revenue than cost) and those that might be underperforming (where cost exceeds revenue).

A graph of a company

Description automatically generated with medium confidence

**11. Line Chart – Total Profit by Product Name**

* **Chart Type**: Line Chart
* **Why We Used It**: This Line chart provides an effective way to display and compare profit by individual product.
* A graph with a line

  Description automatically generated**What It Displays**: This chart shows the total profit generated by each product. It allows us to identify the top-performing products, highlighting which ones are contributing most to overall profitability.

**12.** **Treemap Chart – Active Products by Product Category**

* **Chart Type**: Treemap Chart
* **Why We Used It**: The treemap chart effectively represents hierarchical data, allowing us to display active products within their respective product categories in a compact, organized manner.
* **A screen shot of a chart

  Description automatically generatedWhat It Displays**: The treemap chart displays each product category as a larger rectangle, subdivided into smaller rectangles that represent individual active products within that category. The size of each rectangle corresponds to the number of active products,

**Conclusion on Data Visualization and Navigator Setup:**

By organizing the data into four main sections (Overview, Customers, Products, and Suppliers), we were able to structure our analysis effectively. Using a navigator allowed us to make the dashboard more interactive and easier to navigate. Each section contains targeted visualizations to answer key business questions and provide actionable insights.