

Azure Data Pipeline Documentation

Version 1.0

Date: 17 Oct 2024

Team Members:

- **Bassel Ashraf Ahmed** – Data Factory Implementation
 - **Omar Hussein Mohamed** – Merging and Querying SQL Tables
 - **Ahmed Tarek Mohamed** – Regression Analysis
 - **Mohamed Tarek Abdelsattar** – Power BI Dashboard Insights
-

This document outlines the design, implementation, and key insights from a data pipeline that ingests CSV data into an Azure SQL database, merges SQL tables for efficient querying, applies regression analysis for predictive insights, and visualizes the results using a Power BI dashboard.

Prerequisites

Before starting with the pipeline setup and data analysis, the following prerequisites must be met:

1. **Azure Subscription:** You must have an active Azure subscription with sufficient credits for provisioning SQL Database, Data Factory, and other necessary resources.
 2. **Power BI Account:** A Power BI account is required for creating the dashboards.
 3. **Python Environment:** For regression analysis, ensure that Python is installed with the following libraries:
 - `numpy`
 - `pandas`
 - `sklearn`
 - `matplotlib` (for visualizing results)
 4. **CSV Datasets:** The CSV files (`Customers.csv`, `Products.csv`, `Invoices.csv`, `InvoiceItems.csv`) should be available and accessible either locally or in a cloud storage account.
 5. **SQL Management Tools:** Tools like Azure Data Studio or SQL Server Management Studio (SSMS) to query and manage the Azure SQL Database.
-

1. Infrastructure Setup

The following Azure resources were provisioned:

- **Resource Group:** Created to group all related Azure services.
 - **SQL Server:** An Azure SQL Server instance was provisioned to host the target database.
 - **SQL Database:** A database named [database name not visible in screenshots] was created with the following tables:
 - **Customers:** Customer information (CustomerID, Country).
 - **Products:** Product details (StockCode, Description, UnitPrice).
 - **Invoices:** Invoice records (InvoiceNo, CustomerID, InvoiceDate).
 - **InvoiceItems:** Items in each invoice (InvoiceNo, StockCode, Quantity, TotalPrice).
-

2. Data Ingestion and Cleaning

2.1 Data Ingestion

In this step, we will load the dataset from a CSV file and display the first few rows to understand the structure of the data:

```
```import pandas as pd

Load the dataset from the CSV file

data = pd.read_csv("D:/Microsoft Data Engineer/Graduation Project/online_retail.csv")

Display the first few rows of the dataset

print(data.head())

```
```

2.2 Data Cleaning

2.2.1 Handling Missing Values

Missing values, especially in **CustomerID**, are crucial as they affect customer-level analysis.

- **Why:** Missing **CustomerID** leads to inaccurate insights like segmentation and personalization.
- **Strategy:** Drop rows where **CustomerID** is missing.

```
...
```

```
# Check for missing values
```

```
print(data.isnull().sum())
```

```
# Drop rows where CustomerID is missing
```

```
data_clean = data.dropna(subset=['CustomerID'])
```

```
# Confirm no missing CustomerID values exist
```

```
print(data_clean.isnull().sum())
```

```
...
```

2.2.2 Removing Duplicates

Duplicate records can skew transaction counts and total sales.

- **Why:** Duplicate entries inflate sales and key metrics.
- **Strategy:** Remove duplicate rows based on all columns.

```
...
```

```
# Check for duplicate rows
```

```
duplicates = data_clean.duplicated().sum()
```

```
print(f"Number of duplicate rows: {duplicates}")
```

```
# Drop duplicate rows
```

```
data_clean = data_clean.drop_duplicates()
```

```
# Confirm duplicates are removed
```

```
print(data_clean.duplicated().sum())  
...
```

2.2.3 Handling Incorrect or Negative Quantities

Negative quantities usually represent returns, which we filter out for standard sales analysis.

- **Why:** Negative quantities skew sales metrics like revenue.
- **Strategy:** Remove rows with negative quantities.

```
...  
  
# Check for negative quantities  
negative_quantities = data_clean[data_clean['Quantity'] < 0]  
  
print(f"Number of negative quantity transactions: {len(negative_quantities)}")  
  
# Remove rows where quantity is negative  
data_clean = data_clean[data_clean['Quantity'] > 0]  
  
# Confirm no negative quantities exist  
print(data_clean[data_clean['Quantity'] < 0])  
...
```

2.2.4 Handling Invalid Stock Codes

Invalid stock codes (e.g., non-product-related transactions) should be removed.

- **Why:** Invalid stock codes often correspond to charges, fees, or non-product transactions.
- **Strategy:** Filter out invalid stock codes.

```
...  
  
# Identify and remove invalid stock codes  
invalid_stock_codes = ['POST', 'D', 'M', 'DOT', 'BANK CHARGES', 'PADS', 'CRUK']  
  
data_clean = data_clean[~data_clean['StockCode'].isin(invalid_stock_codes)]
```

```
# Confirm invalid stock codes are removed

print(data_clean['StockCode'].isin(invalid_stock_codes).sum())

...
```

2.3 Ensuring Consistency and Final Checks

2.3.1 Fix Data Types

Ensure columns have the correct data types (e.g., dates should be in `datetime` format).

- **Why:** Correct data types ensure proper filtering, grouping, and aggregation.
- **Strategy:** Convert `InvoiceDate` to datetime, and ensure `CustomerID` is a string.

```
...

# Convert InvoiceDate to datetime format

data_clean['InvoiceDate'] = pd.to_datetime(data_clean['InvoiceDate'])

# Ensure CustomerID is a string

data_clean['CustomerID'] = data_clean['CustomerID'].astype(str)

# Confirm data types

print(data_clean.dtypes)

...
```

2.3.2 Recalculate Derived Columns

Calculate columns like `TotalPrice` to capture transaction metrics.

- **Why:** Helps with trend analysis and sales forecasting.
- **Strategy:** Derive `TotalPrice` from `Quantity` and `UnitPrice`.

```
...

# Calculate total price for each transaction
```

```
data_clean['TotalPrice'] = data_clean['Quantity'] * data_clean['UnitPrice']
```

```
# Preview the result
```

```
print(data_clean[['Quantity', 'UnitPrice', 'TotalPrice']].head())
```

```
...
```

2.3.3 Final Checks and Exporting Cleaned Data

Once the data is cleaned, export it for further use.

- **Why:** Clean data is ready for analysis, machine learning, or uploading to a data warehouse.
- **Strategy:** Save the cleaned dataset.

```
...
```

```
# Final check on the cleaned data
```

```
print(data_clean.info())
```

```
# Export cleaned data to CSV
```

```
data_clean.to_csv('D:/Microsoft Data Engineer/Graduation  
Project/cleaned_online_retail.csv', index=False)
```

```
...
```

3. Exploratory Data Analysis (EDA)

In this stage, we will perform exploratory data analysis to gain insights into the structure and key patterns of the dataset. Here's an overview of the main analyses performed:

1. Top 10 Countries by Number of Transactions

The first step is to explore which countries contribute the most transactions. We use a bar plot to visualize the top 10 countries in terms of the number of transactions.

```
...
```

```
# Calculate the top 10 countries with the most transactions
```

```

country_counts = data['Country'].value_counts().head(10)

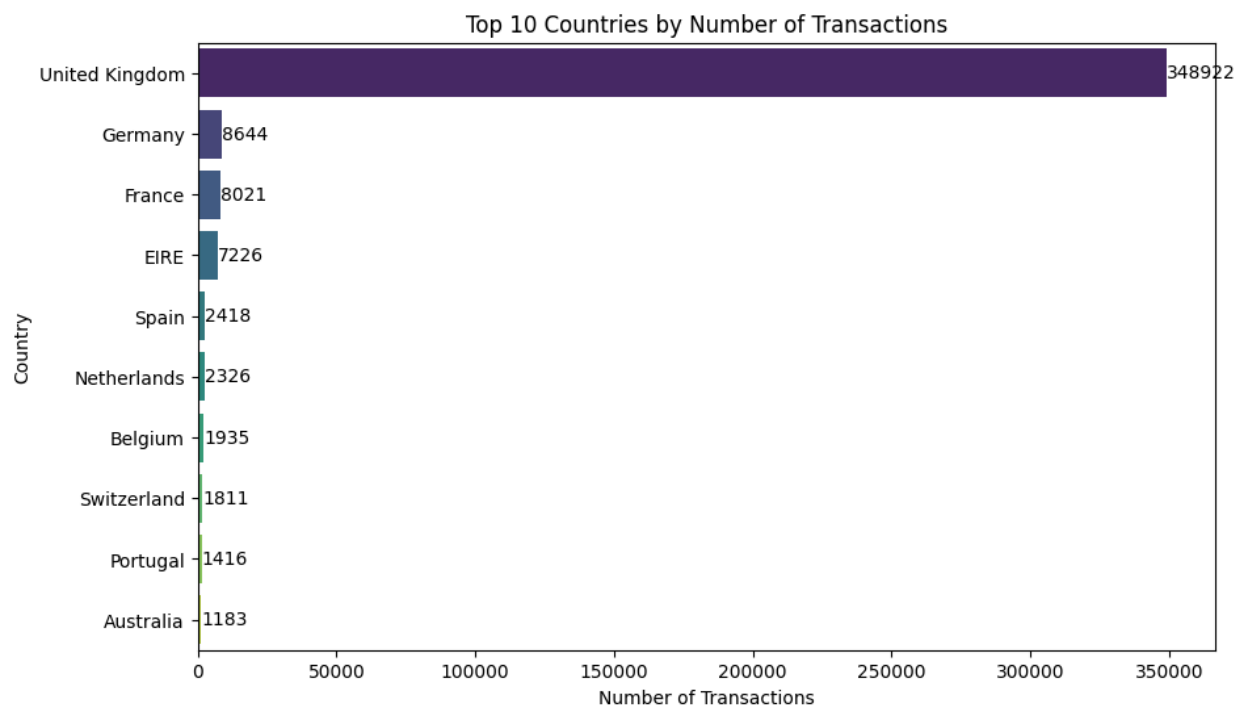
# Create a bar plot for the top 10 countries
plt.figure(figsize=(10, 6))
sns.barplot(x=country_counts.values, y=country_counts.index, palette='viridis')

# Annotate the bar plot with the actual values
for index, value in enumerate(country_counts.values):
    plt.text(value + 100, index, str(value), color='black', va="center")

# Set titles and labels
plt.title('Top 10 Countries by Number of Transactions')
plt.xlabel('Number of Transactions')
plt.ylabel('Country')
plt.show()

```

...



2. Top 10 Most Sold Products

This analysis identifies the most popular products by the quantity sold. A bar plot displays the top 10 products based on the total quantity sold.

...

```

# Group the data by product description and sum the quantities sold

```

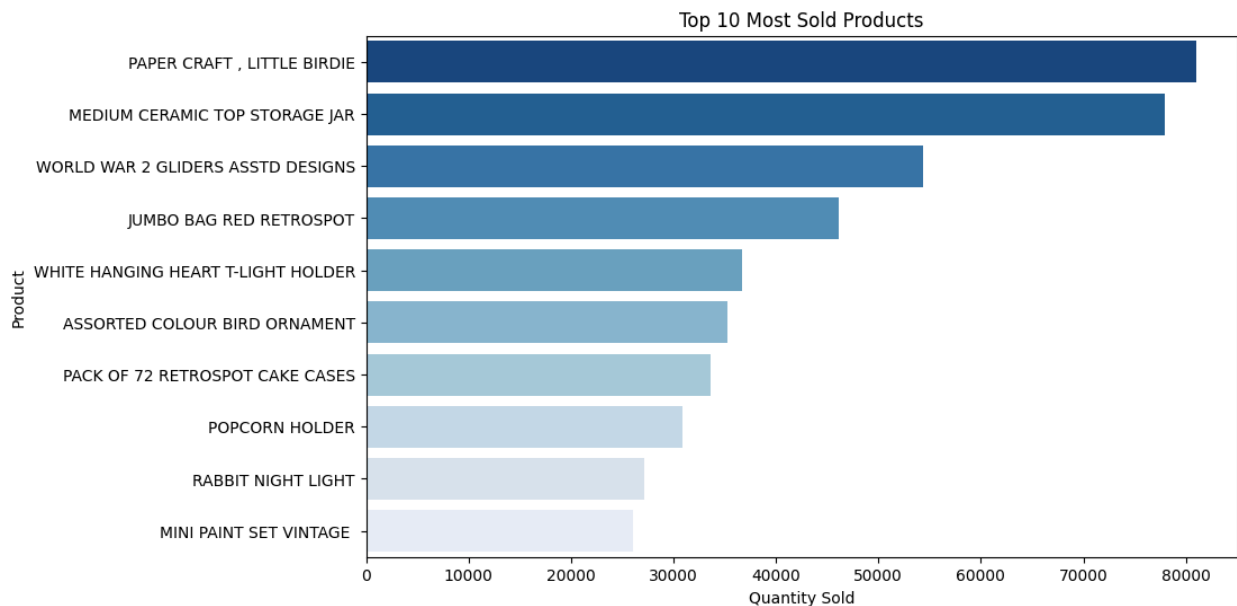
```

top_10_products =
data.groupby('Description')['Quantity'].sum().sort_values(ascending=False).head(10)

# Create a bar plot for the top 10 most sold products
plt.figure(figsize=(10, 6))
sns.barplot(x=top_10_products.values, y=top_10_products.index, palette='Blues_r')

# Set titles and labels
plt.title('Top 10 Most Sold Products')
plt.xlabel('Quantity Sold')
plt.ylabel('Product')
plt.show()

```



3. Monthly Sales Trend

To understand the sales performance over time, we group the data by month and calculate the total sales for each month. A line plot illustrates the monthly sales trend.

```

...

# Convert 'InvoiceDate' to a date-time object
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'], errors='coerce')

# Create a 'YearMonth' column
data['YearMonth'] = data['InvoiceDate'].dt.to_period('M')

# Group the data by 'YearMonth' and calculate total sales

```



```

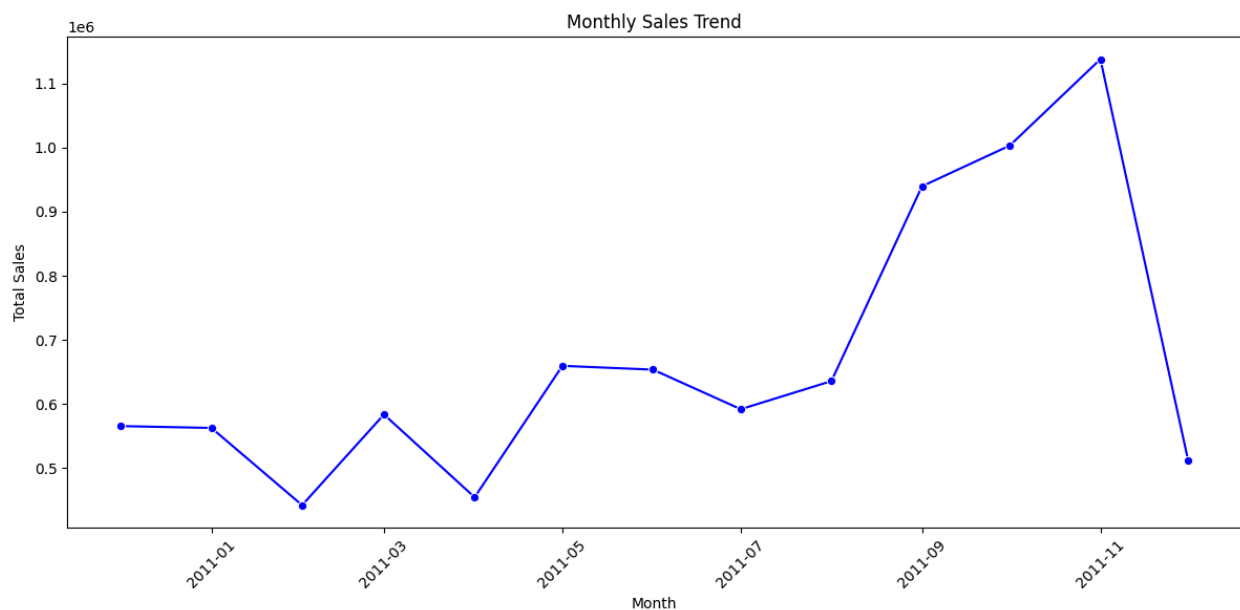
monthly_sales = data.groupby('YearMonth')['TotalPrice'].sum().reset_index()

# Convert 'YearMonth' back to a timestamp for plotting
monthly_sales['YearMonth'] = monthly_sales['YearMonth'].dt.to_timestamp()

# Plot the monthly sales trend
plt.figure(figsize=(12, 6))
sns.lineplot(x='YearMonth', y='TotalPrice', data=monthly_sales, marker='o', color='blue')

# Set titles and labels
plt.title('Monthly Sales Trend')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
'''

```



4. Distribution of Total Prices (Violin Plot)

Lastly, a violin plot is used to visualize the distribution of the total prices per transaction. This helps to understand the spread and density of the total sales amount.

```

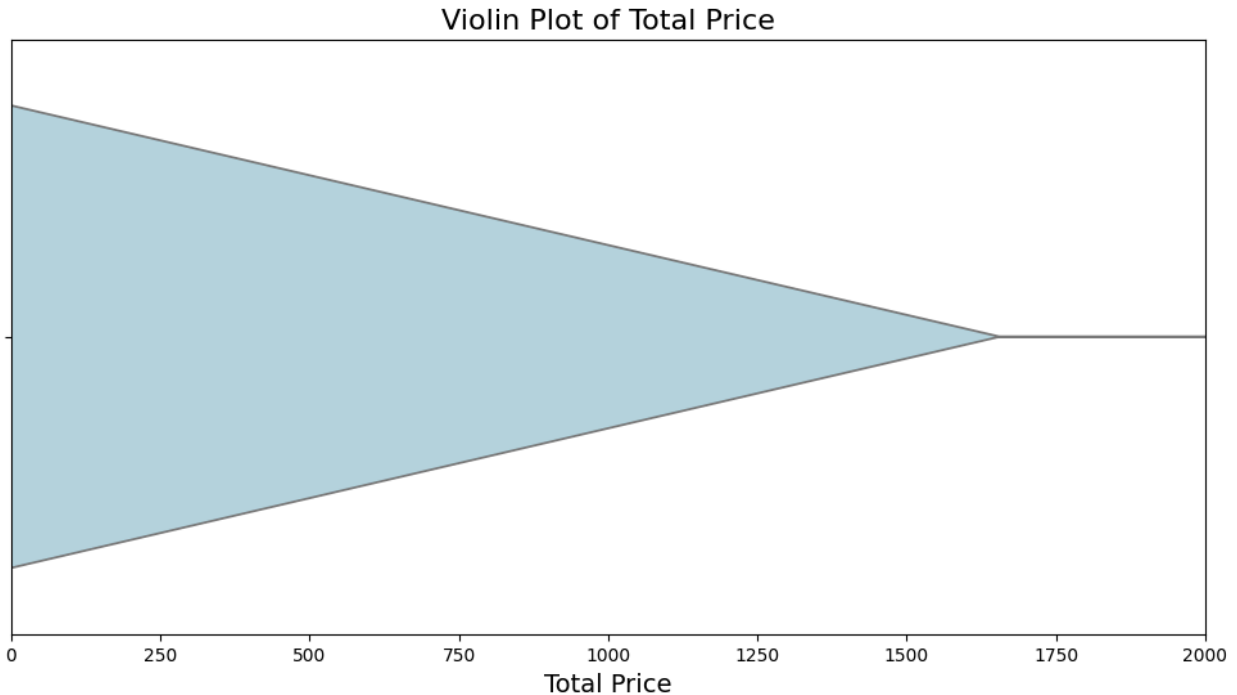
'''
# Create a violin plot for the distribution of total prices
plt.figure(figsize=(12, 6))
sns.violinplot(x=data['TotalPrice'], inner=None, color='lightblue')

```

```
# Limit the x-axis for better visualization
plt.xlim(0, 2000)

# Set titles and labels
plt.title('Violin Plot of Total Price', fontsize=16)
plt.xlabel('Total Price', fontsize=14)
plt.show()

...
```



Summary:

- **Top 10 countries:** The UK is the leading country by transaction count.
- **Top 10 most sold products:** The most popular products are dominated by decor and utility items.
- **Monthly sales trend:** Sales show seasonal fluctuations with a general upward trend.
- **Total price distribution:** Most transactions fall within a lower price range, with a few higher-value transactions.

4. Data Factory Implementation

The pipeline is built using Azure Data Factory, which orchestrates the entire data flow from CSV ingestion to SQL database insertion. The key components are:

- **Linked Services:** Connections to the source CSV files and the SQL database.
- **Datasets:** Datasets correspond to each CSV file (Customers, Products, Invoices, InvoiceItems) and the target SQL tables.
- **Data Flows:** Four data flows (`customers1`, `products1`, `invoices1`, `invoiceitems1`) handle the import of each dataset from CSV to the SQL database. Each data flow defines a source (CSV) and sink (SQL table).
- **Pipeline:** The pipeline (`graduationproject`) orchestrates the execution order of the data flows.

Data Transformation

- **Data Type Conversion:** Ensures that data types (e.g., `UnitPrice` as decimal, `Quantity` as integer) match between CSV and SQL.
- **Data Cleansing:** Handles nulls, invalid values, and inconsistent formats.
- **Data Mapping:** Maps CSV columns to SQL table columns during the data flow.

Execution Order

1. `customers1`: Loads the Customers table.
2. `products1`: Loads the Products table.
3. `invoices1`: Loads the Invoices table, referencing Customers.
4. `invoiceitems1`: Loads the InvoiceItems table, referencing Invoices and Products.

5. Merging and Querying SQL Tables

After data ingestion, four tables (Customers, InvoiceItems, Invoices, Products) are merged into a unified **SalesSummary** table for easier querying.

Table Relationships

- **Customers:** Links with Invoices through `CustomerID`.
- **Invoices:** Links with InvoiceItems through `InvoiceNo`.
- **InvoiceItems:** Links with Products through `StockCode`.

SQL Commands

...

```
CREATE TABLE SalesSummary (
```

```
    CustomerID INT,
```

```

Country VARCHAR(100),
InvoiceNo INT,
InvoiceDate DATETIME,
StockCode VARCHAR(100),
Description VARCHAR(255),
UnitPrice DECIMAL(10, 2),
Quantity INT,
TotalPrice DECIMAL(10, 2),
TotalItemPrice AS (Quantity * UnitPrice)
);

INSERT INTO SalesSummary (CustomerID, Country, InvoiceNo, InvoiceDate, StockCode,
Description, UnitPrice, Quantity, TotalPrice)

SELECT

c.CustomerID,
c.Country,
i.InvoiceNo,
i.InvoiceDate,
ii.StockCode,
p.Description,
p.UnitPrice,
ii.Quantity,
ii.TotalPrice

FROM dbo.Customers AS c

JOIN dbo.Invoices AS i ON c.CustomerID = i.CustomerID

```

```
JOIN dbo.InvoiceItems AS ii ON i.InvoiceNo = ii.InvoiceNo
```

```
JOIN dbo.Products AS p ON ii.StockCode = p.StockCode;
```

```
...
```

Data Example


```
``` SELECT TOP 10 * FROM dbo.SalesSummary; ```
```

 all

## Example Queries

Total Sales per Customer:


```
```SELECT CustomerID, Country, SUM(TotalItemPrice) AS TotalSales
FROM SalesSummary
GROUP BY CustomerID, Country;
```
```

 Total\_Sales\_per\_Customer

## Top-Selling Products:

```
...
```


```
SELECT StockCode, Description, SUM(Quantity) AS TotalQuantitySold
FROM SalesSummary
GROUP BY StockCode, Description
ORDER BY TotalQuantitySold DESC;
...
```

 Top-Selling-Products

## Invoices in a Date Range:

```
...
```

```
SELECT InvoiceNo, InvoiceDate, CustomerID, Country, SUM(TotalItemPrice) AS
TotalInvoiceAmount
FROM SalesSummary
WHERE InvoiceDate BETWEEN '2011-11-01' AND '2011-11-30'
GROUP BY InvoiceNo, InvoiceDate, CustomerID, Country;
...
```

 Invoices\_within\_a\_Specific\_Date Range

---

## 6. Regression Analysis

The regression analysis was performed using Python libraries (e.g., `numpy`, `pandas`, `sklearn.svm`) to predict sales trends.

### Key Steps

#### Data Preparation:

- Loaded the CSV data using `pandas`.
- Cleaned and fixed the `InvoiceDate` column by converting invalid hours into a correct format.
- Converted `InvoiceDate` to a datetime object and split it into day, month, and year.

#### Feature Definition:

- Defined features like product details, unit price, and sales trends.

#### Modeling:

- Standardized the data using `StandardScaler`.
- Trained a Support Vector Regression (SVR) model on the dataset.

#### Evaluation:

- Calculated Mean Squared Error (MSE) and  $R^2$  scores to evaluate the model.
  - Visualized predictions vs. actual results using line charts.
- 

## 7. Power BI Dashboard Insights

A Power BI dashboard was built to visualize key sales data.

### Features

- **Summary Cards:**
  - Number of Countries with customers.
  - Number of Customers who placed orders.
  - Total Sales.
  - Average Unit Price.
- **Donut Chart:** Shows the percentage of customers per country.

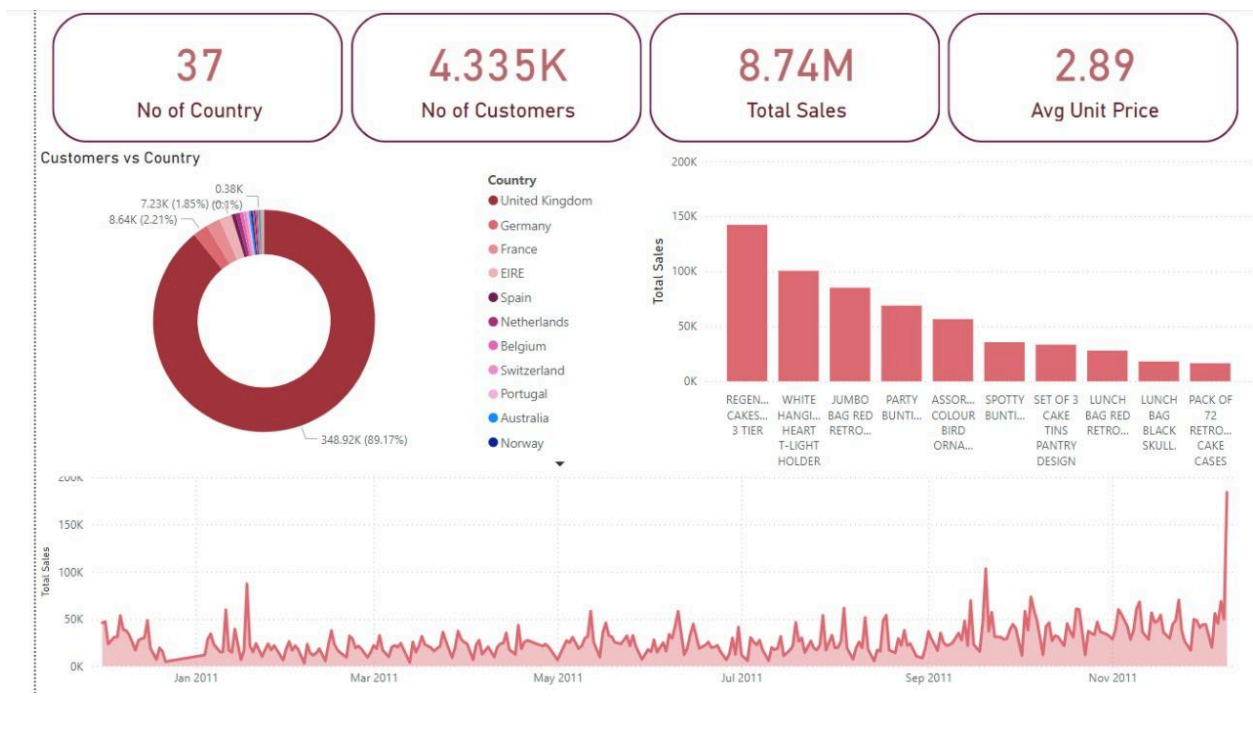
- **Top Products:** A column chart displaying the top 10 best-selling products.
- **Sales Trends:** An area chart representing sales over time (from January to December 2011).

## Insights

- **Country-wise Distribution:**
  - 89.17% of the customer base is from the United Kingdom, with smaller contributions from other countries such as Germany, France, and Spain.
- **Top Products:**
  - Popular products like "Regency Cakestand 3 Tier" and "White Hanging Heart T-light Holder" contribute significantly to sales.
- **Sales Peaks:**
  - Several sales peaks were observed during the year, indicating potential promotional events or seasonal campaigns.

## Challenges

- **Data Complexity:** Managing large datasets and maintaining table relationships was complex.
- **Visual Representation:** Choosing the right charts and ensuring the clarity of insights required careful consideration.
- **User Interaction:** Making the dashboard intuitive and user-friendly, with slicers for country and product filtering, was a key challenge.



## Conclusion

This project successfully ingested CSV data into an Azure SQL database using an Azure Data Factory pipeline, streamlined queries by merging SQL tables, applied regression analysis to predict trends, and provided actionable business insights through a Power BI dashboard. The combination of cloud-based data management, advanced analytics, and visualization tools demonstrated their effectiveness in driving data-informed decision-making.