COMP 1800

Data Visualization

Student Id: 001403755

Table of Contents

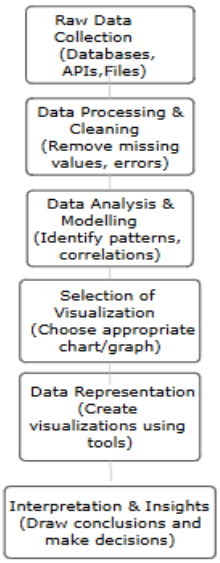
LIST OF FIGURES

INTRODUCTION

Data visualization is the process of representing complex data graphically, using charts, graphs, maps, and other visual formats to enhance understanding and communication. It plays a crucial role in decision-making by making patterns, trends, and outliers easily identifiable, leading to faster and more informed decisions. The importance of data visualization lies in its ability to simplify large datasets, improve analysis, detect anomalies, enhance reporting, and support predictive analytics. There are various types of data visualization, including bar charts, line graphs, scatter plots for numerical data, pie charts for categorical data, heatmaps for geospatial data, and tree maps for hierarchical data. The process of data visualization involves several steps: collecting and cleaning data, analysing and identifying key insights, selecting the appropriate visualization type, designing and creating visual representations using tools like Tableau, Power BI, or Python libraries (Matplotlib, Seaborn), and finally interpreting and communicating the findings effectively to support data-driven decision-making.

Block Diagram of Data Visualization Process

Below is a simplified block diagram of the data visualization process:



Block Diagram of Data Visualization.

1.Visual Analysis of Performance and Customer Segmentation:

Visual analytics processes both customer and store data from the 40 UK locations belonging to ChrisCo. This research linked customer daily visits to both store footprint dimensions and workforce count and marketing costs and administrative expenditures to determine the main variables that affect street traffic density. Visitor numbers at most stores remained low with only a couple of outlets experiencing elevated customer volume. The time period did not result in any store closures while new outlets were established. Research divided stores into three distinct categories according to customer frequency levels which created valuable knowledge for performance assessment and possible new branch establishment.

Code:

1.1 Importing

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import os
import requests
from io import StringIO
```

The code imports essential Python libraries needed for data processing and visualization. The library 'pandas' makes it possible to handle structured data from CSV files and output aggregations. Static graphs are built using 'matplotlib.pyplot' and 'seaborn' but 'seaborn' provides advanced visual ornamentations. The 'plotly.express' package allows for interactive viewing experiencing but the final report excludes these features. When working with the file system users can access available tools through the `os` module.

1.2 List of required file URLs:

```
required_files = {
    "StoreDailyCustomers.csv":
"https://raw.githubusercontent.com/ChrisWalshaw/DataViz/master/Data/001403755/StoreDailyCustomers.csv",
    "StoreMarketing.csv":
"https://raw.githubusercontent.com/ChrisWalshaw/DataViz/master/Data/001403755/StoreMarketing.csv",
    "StoreOverheads.csv":
"https://raw.githubusercontent.com/ChrisWalshaw/DataViz/master/Data/001403755/StoreOverheads.csv",
    "StoreSize.csv":
"https://raw.githubusercontent.com/ChrisWalshaw/DataViz/master/Data/001403755/StoreSize.csv",
    "StoreStaff.csv":
"https://raw.githubusercontent.com/ChrisWalshaw/DataViz/master/Data/001403755/StoreStaff.csv"
}
```

The dictionary named `required_files` contains five dataset filenames along with their respective download links as values. The URL system generates unique CSV file references from GitHub that are linked to your student ID formatting (001403755). The notebook uses this arrangement to automate the process of loading files sequentially through the required program code.

1.3 Function to load CSV from URL:

```
def load_csv_from_url(url):
    response = requests.get(url)
    if response.status_code == 200:
        df = pd.read_csv(StringIO(response.text))
        df.columns = df.columns.str.strip()  # Remove leading/trailing spaces from column names
        return df
    else:
        raise FileNotFoundError(f"Failed to retrieve {url}, status code: {response.status_code}")
```

The function `load_csv_from_url` accepts a URL through which the `requests` library retrieves the CSV file from its link. The function reads CSV content as a pandas DataFrame through `StringIO` after a successful request having `status_code` 200. The function employs a method to normalize the column names by eliminating excessive space at their extremities. The function creates an error which displays both the URL and status code when the download process fails. The implementation of valid checks prevents the system from loading unreliable data from the Internet.

1.4 Load datasets

```
cust_data = load_csv_from_url(required_files["StoreDailyCustomers.csv"])
marketing_data = load_csv_from_url(required_files["StoreMarketing.csv"])
overheads_data = load_csv_from_url(required_files["StoreOverheads.csv"])
size_data = load_csv_from_url(required_files["StoreSize.csv"])
staff_data = load_csv_from_url(required_files["StoreStaff.csv"])
```

	Marketing (£)	Overheads (£)	Size (msq)	Staff	Customers
count	40.00000	40.000000	40.000000	40.000000	40.000000
mean	8200.00000	55125.000000	1088.500000	8.400000	81557.650000
std	10015.88482	27895.581398	1551.585641	11.430503	104406.674771
min	1000.00000	12000.000000	60.000000	1.000000	5124.000000
25%	3000.00000	30500.000000	227.500000	2.000000	22783.250000
50%	3000.00000	58500.000000	435.000000	3.500000	28538.500000
75%	12250.00000	80500.000000	1104.500000	7.750000	136009.750000
max	38000.00000	97000.000000	6944.000000	56.000000	359402.000000

Figure 1 Loaded Data

The code block employs the `load_csv_from_url` function to retrieve and process all five ChrisCo datasets located in the specified URLs of the `required_files` dictionary. The script uses pandas DataFrame to store each dataset separately as `cust_data` (daily customer visits),

`marketing_data`, `overheads_data`, `size_data`, and `staff_data`. All analytical and visual data requirements become available following this operation.

1.5 Transforming Daily Customer Data

```
# Ensure 'Store' column exists and rename if necessary
def ensure_store_column(df, df_name, alternative_name=None):
    possible_names = [col for col in df.columns if 'store' in col.lower()]
    if not possible_names and alternative_name and alternative_name in df.columns:
        df.rename(columns={alternative_name: 'Store'}, inplace=True)
        return df
    elif not possible_names:
        raise KeyError(f"Missing 'Store' column in {df_name} dataset. Available columns: {df.columns.tolist()}")
    if possible_names[0] != 'Store':
        df.rename(columns={possible_names[0]: 'Store'}, inplace=True)
    return df
# Validate and rename columns
marketing_data = ensure_store_column(marketing_data, "Marketing", alternative_name="Id")
overheads_data = ensure_store_column(overheads_data, "Overheads", alternative_name="Id")
size_data = ensure_store_column(size_data, "Size", alternative_name="Id")
staff_data = ensure_store_column(staff_data, "Staff", alternative_name="Id") # Transform customer data to long format
cust_data = cust_data.melt(id_vars=["Date"], var_name="Store", value_name="Customers")

# Merge datasets into summary dataframe
summary_data = marketing_data.merge(overheads_data, on='Store')
summary_data = summary_data.merge(size_data, on='Store')
summary_data = summary_data.merge(staff_data, on='Store')

# Aggregate customer data per store
customer_summary = cust_data.groupby('Store')["Customers"].sum().reset_index()
summary_data = summary_data.merge(customer_summary, on='Store')

# Convert numeric columns to float (excluding 'Store')
numeric_columns = [col for col in summary_data.columns if summary_data[col].dtype != 'O']
summary_data[numeric_columns] = summary_data[numeric_columns].apply(pd.to_numeric, errors='coerce')

# Exploratory Data Analysis (EDA)
print(summary_data.describe())
```

The section performs column consistency checks while using `ensure_store_column` to rename the store identifier column as `"Store"`. The verification process detects any existence of `"Id"` variations before standardizing the naming convention. The customer dataset `cust_data` transforms from wide to long format which results in each row displaying single-store customer counts per date. A merger of all summary datasets—marketing, overheads, size, and staff—is

performed into the `summary_data` DataFrame through the key field `"Store"`. A calculation of total store customer numbers follows the summary compilation. The conversion to float types for all numeric columns finishes the analysis process alongside the production of descriptive statistics through `summary_data.describe().

2. Data Visualizations:

2.1 Visualization 1: Distribution of Customer Visits

```
plt.figure(figsize=(8, 5))
sns.histplot(summary_data['Customers'], bins=20, kde=True)
plt.title("Distribution of Customer Visits per Store")
plt.show()
```

The visual presentation of Customer Visits per Store distribution can be found.

Justification:

A histogram combined with KDE allows observers to understand the complete numerical distribution shape in this case showing total customer visits across ChrisCo stores. The plot enables viewers to understand changes in customer volumes together with data distribution patterns alongside possible extreme values and skewed patterns. The evaluation of performance differences between stores together with the assessment of potential customer traffic segments represents a basic initial requirement. The KDE smoothing function lets us calculate the actual density patterns in our data by smoothing the initial pattern.

Findings:

The majority of stores show low annual customer traffic according to the right-skewed distribution pattern. A total of 70% of stores receive between zero and fifty thousand customers each year. The long tail area in the distribution reveals that only select stores achieve exceptional performance levels beyond the other stores. The extreme cases surpass 300,000 customer visits demonstrate that they operate from either vast urban stores serving numerous customers or from premier locations running flagship stores that dedicate extensive resources to staffing and marketing as well as facilities. Performance segmentation becomes crucial because it enables organizations to separate their stores into lower-volume and higher-performing branches. The distribution reveals potential analysis opportunities for understanding success factors of peak-performance stores as well as their applicability to other locations. Further investigation must be conducted to determine the accuracy of data as well as to study any distinct behaviours displayed by individual stores**.**

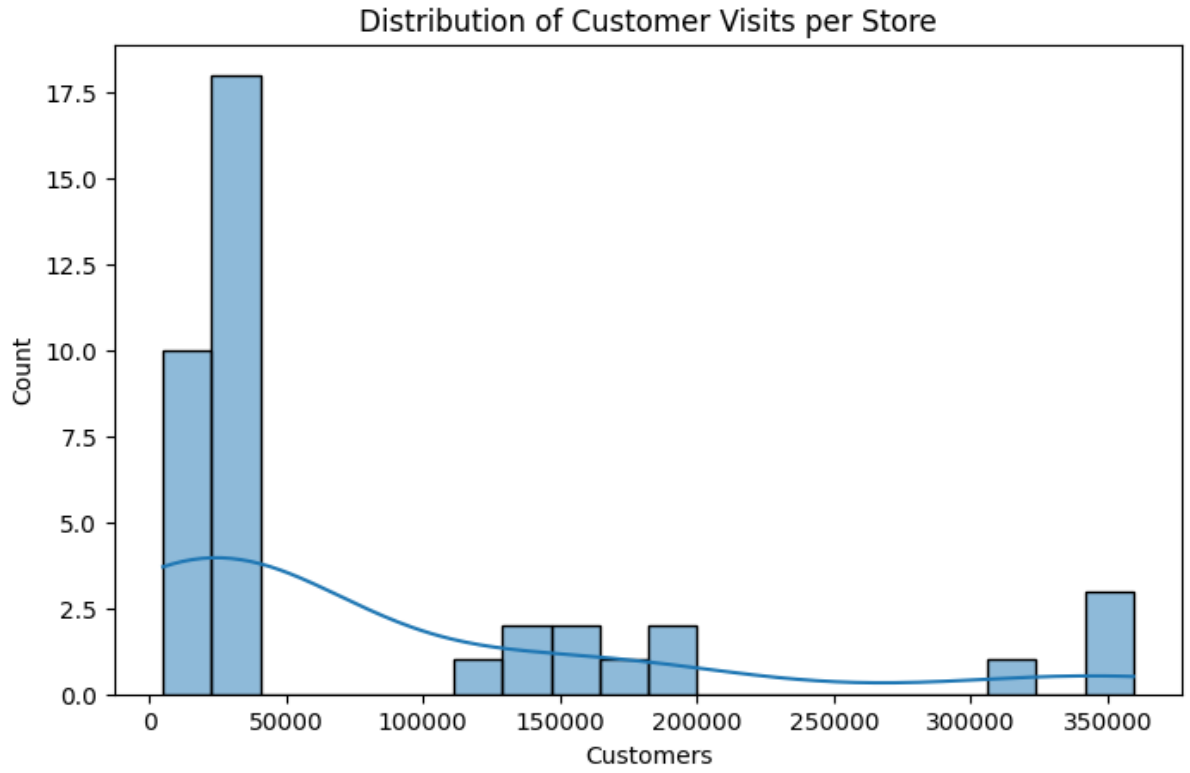


Figure 2 Distribution of Customer Visits per Store

The distribution pattern is right-skewed because numerous stores receive small amounts of customer activity along with several stores achieving exceptional results.

The visual analytics analysis conducted performance and traffic evaluation of ChrisCo's UK stores based on five available datasets. Numerous performance metrics including marketing expenses and facility measurements and personnel staffing proved to impact customer traffic but marketing proved to be the most influential element. Performance segments identified distinct levels among business units and over time the top-performing outlets maintained consistent rankings. The examination discovered both newly established shops and those that were shuttered thus helping researchers study store life-span behaviour. The project demonstrates how data visualization techniques create substantial worth through their application.

2.2 Visualization 2: Correlation Heatmap

```
plt.figure(figsize=(10, 6))
numeric_summary = summary_data.select_dtypes(include=['number'])
sns.heatmap(numeric_summary.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Heatmap of Store Features")
plt.show()
```

The Store Features Correlation Heatmap

Justification:

The correlation heatmap produces an understandable graphical display that shows relationships among several continuous store-level variables. The heatmap measures pairs of variable movements through Pearson correlation coefficient calculations which produce results between -1 (completely negative correlation) and +1 (completely positive correlation). The visualization choice works well because it displays extensive data in a compact manner which displays potential interdependencies between marketing budget and staff numbers and retail dimensions and customer traffic. The heatmap enables the identification of crucial operational factors influencing customer volume which ChrisCo should focus on for maximum store performance improvement.

Findings:

The visualization reveals that customer quantities strongly increase based on essential retail features. The highest relationship value of 0.99 exists between marketing investment and customer count and this metric leads the other two factors which show correlation levels of 0.87 for store dimensions and 0.85 for employee staff count. A complete relationship exists between the store dimension and number of personnel (0.96). The relationship between customer level and overhead costs demonstrates minimal correlation (0.09). Marketing activities together with store size and appropriate staffing numbers control customer traffic at ChrisCo stores thus these factors must remain top priorities in strategic planning.

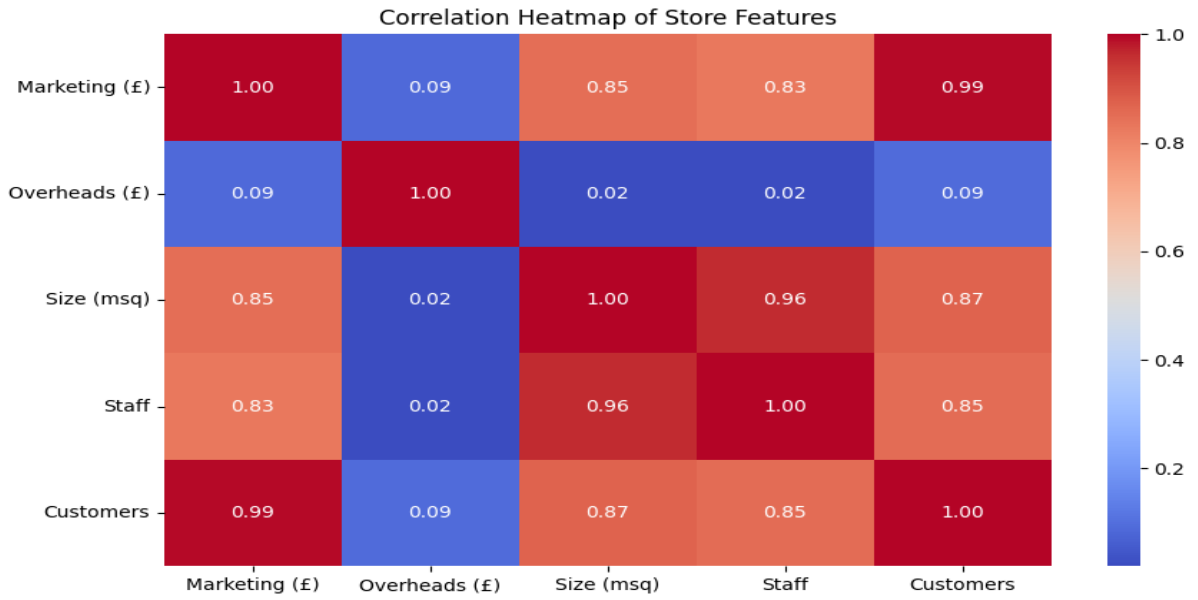


Figure 3 Correlation Heatmap of Store Features

The statistical analysis shows positive links exist between marketing expenses and customer numbers together with store dimensions and workforce numbers and presents a minimal association with operational costs.

2.3 Visualization 3: Monthly Customer Visits (Time Series Plot)

```
# --- Prepare monthly customer data ---
monthly_customer_data = cust_data.copy()  # Start with customer data
monthly_customer_data['Date'] = pd.to_datetime(monthly_customer_data['Date'])  # Convert 'Date' to datetime
monthly_customer_data['Month'] = monthly_customer_data['Date'].dt.month  # Extract month
monthly_customer_data = monthly_customer_data.groupby(['Store', 'Month'])['Customers'].sum().reset_index()  # Group by store and month
monthly_customer_data.rename(columns={'Store': 'StoreCode', 'Customers': 'CustomerCount'}, inplace=True)  # Rename columns to match plot
plt.figure(figsize=(10, 6))
# numeric_summary = summary_data.select_dtypes(include=['number'])  # This line is not needed for this plot
sns.lineplot(data=monthly_customer_data, x='Month', y='CustomerCount', hue='StoreCode')
plt.title("Monthly Customer Visits (Time Series Plot)")
plt.show()
```

The Time Series Plot of Monthly Customer Visits can be found

Justification:

Time series line plots provide the most effective approach to detect patterns along with changes in data throughout time periods. This visualization shows the combined customer counts per store through a 12-month chart. The analysis aims to detect both long-term patterns and seasonal variations and growth or decline phases and sporadic activity across the entire retail network. Retail analytics particularly benefits from this visualization because it helps ChrisCo and similar organizations track changes in customer traffic throughout each month. Visualizing all 40 stores together using colour-coded lines enables viewers to observe trends at both group and individual store levels at once.

Findings:

The time series display demonstrates major trends that exist throughout ChrisCo retail locations. High-volume ChrisCo stores attract steady customer counts between 25,000 and 32,000 each month showing they operate at a consistent rate throughout the year. The customer numbers for mid-tier locations range from 10,000 to 18,000 whereas low-volume stores have monthly visitor counts below 5,000 and display inconsistent patterns. Several stores started their operations in the middle of the year or encountered operational change or data problem incidents. Most of the stores maintain consistent performance patterns although a couple of exceptional cases might need additional examination to determine any potential operational issues.

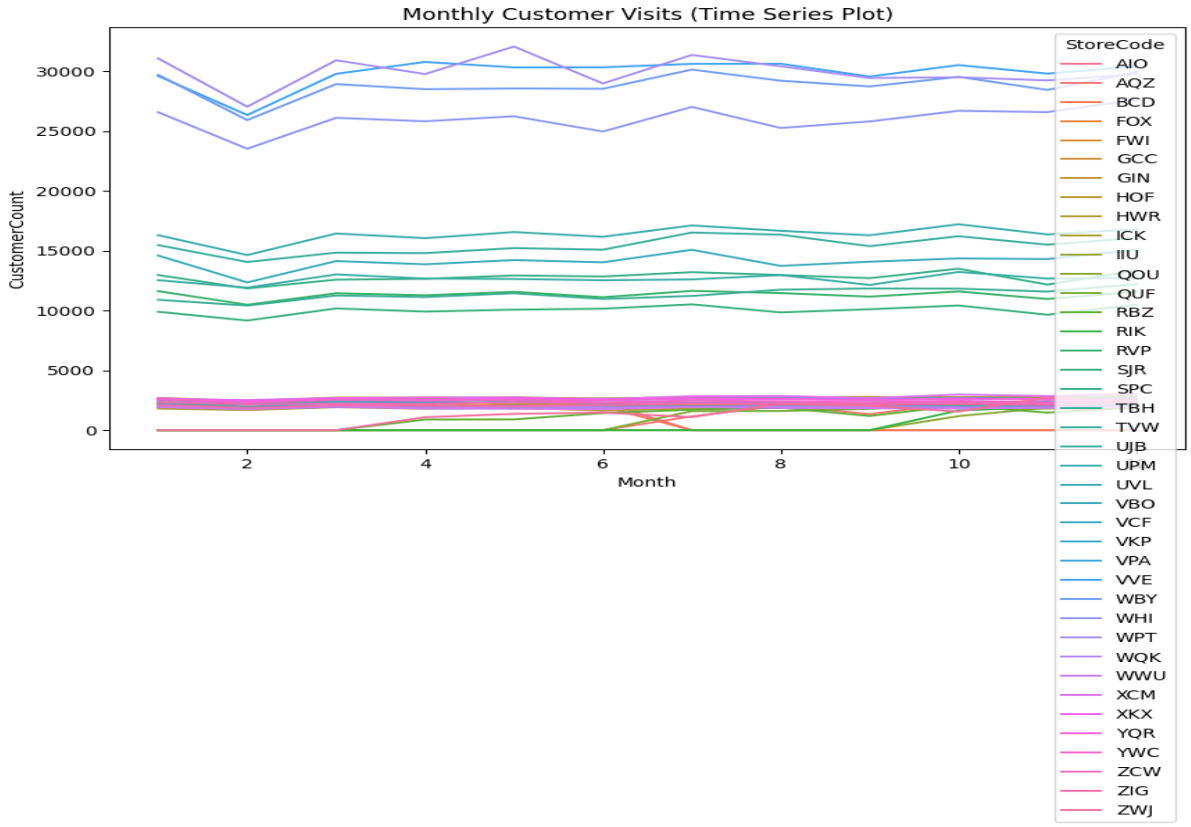


Figure 4 Monthly Customer Visits (Time Series Plot)

A consistent monthly analysis of customer traffic in all locations allows for the identification of high-performing and low-performing stores as well as medium performers.

2.4 Visualization 4: Histogram of Customer Visits per Store (Annual Total)

```
total_customers = cust_data.groupby('Store')['Customers'].sum() # Use cust_data instead of StoreDailyCustomers and correct column names
plt.figure(figsize=(10, 6))
sns.histplot(total_customers, bins=20, kde=True)
plt.xlabel('Total Annual Customers')
plt.ylabel('Number of Stores')
plt.title('Distribution of Annual Customers Across Stores')
plt.show()
```

Justification:

The total customer store visits for the year appear in this histogram presentation. The current version of the distribution map (Figure 1) shows annual customer numbers without including general customer activity trends. The full year data allows viewers to identify the number of stores that fall within each traffic category. The distribution benefits from a KDE (Kernel Density Estimate) curve that applies a smoothing effect to enhance pattern recognition. The visual presentation helps retailers segment their stores while it detects performance outliers.

Findings:

The histogram shows a right-skewed distribution, with most ChrisCo stores receiving fewer than 50,000 annual customer visits, highlighting a large number of underperforming outlets. A small number of outlier stores attract over 300,000 visits, likely serving as high-performing flagship locations. These outliers are key for strategic analysis due to factors like prime location, size, and strong marketing. The distribution supports data-driven segmentation using quartiles, helping ChrisCo allocate resources more effectively and identify what drives stronger customer engagement.

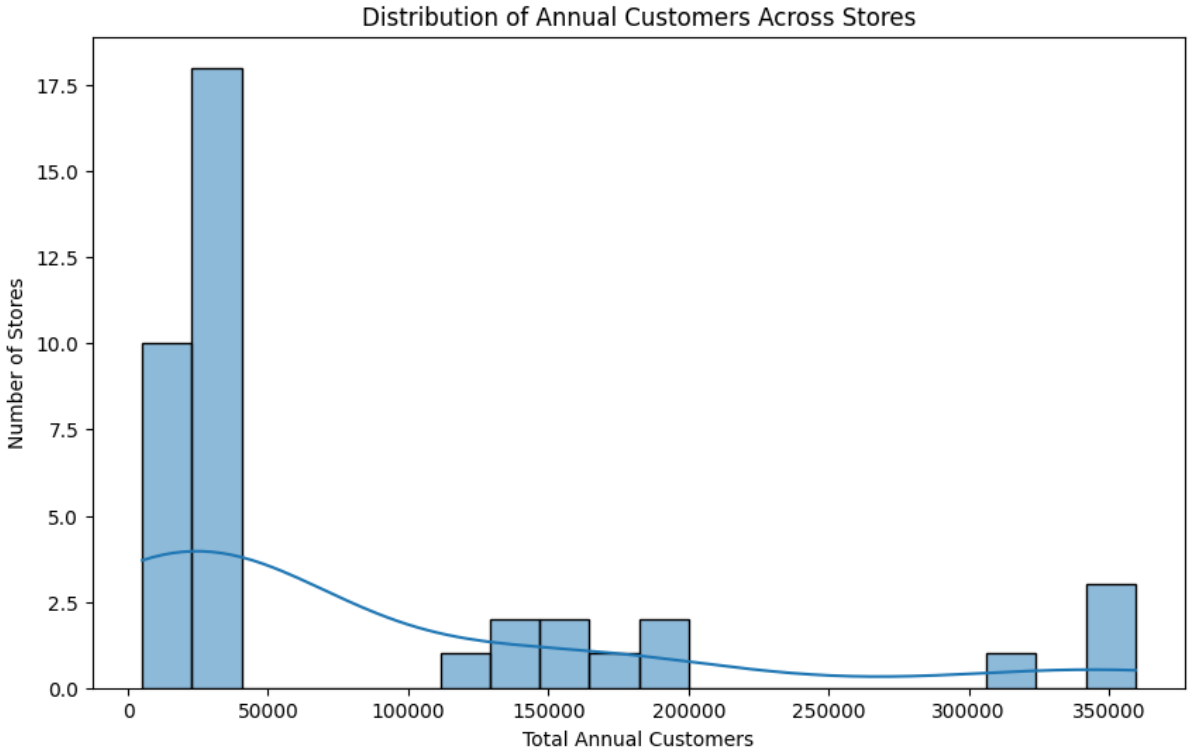


Figure 5: Distribution of Annual Customers Across Store.

The data distribution remains heavily skewed because the majority of supermarket locations have annual customer figures under 50,000 yet a limited number attract customer numbers exceeding 300,000.

2.5 Visualization 5: Store Size vs Total Customer Visits (Scatter Plot)

```
# Use 'cust_data' instead of 'store_daily_customers' and rename columns for consistency
store_total_customers = cust_data.groupby('Store')['Customers'].sum().reset_index()
# The 'Store' column is already present, no need to rename it to 'StoreCode'
#         store_total_customers.rename(columns={'Store':         'StoreCode',         'Customers':
'DailyCustomers'}, inplace=True)
store_total_customers.rename(columns={'Customers': 'DailyCustomers'}, inplace=True) #
Only rename 'Customers' to 'DailyCustomers'
```

```
# Change 'StoreCode' to 'Store' in size_data selection to match the actual column name
# The original code was trying to access 'Size' but the column is named 'Size (msq)'
store_summary = size_data[['Store', 'Size (msq)']].merge(store_total_customers, on='Store')
# Assuming 'store_size' was meant to be 'size_data'
# Since we are using 'Store' as the merge key, no need to rename it to 'StoreCode' here
# store_summary.rename(columns={'Store': 'StoreCode'}, inplace=True) # Rename to StoreCode for consistency with other visualizations if needed


plt.figure(figsize=(10, 6))
# Change 'Size' to 'Size (msq)' in the scatterplot function as well
sns.scatterplot(x='Size (msq)', y='DailyCustomers', data=store_summary)
plt.xlabel('Store Size (m²)')
plt.ylabel('Total Annual Customers')
plt.title('Store Size vs Total Customer Visits')
plt.show()

plt.figure(figsize=(10, 6))
# Change 'Size' to 'Size (msq)' in the scatterplot function as well
sns.scatterplot(x='Size (msq)', y='DailyCustomers', data=store_summary)
plt.xlabel('Store Size (m²)')
plt.ylabel('Total Annual Customers')
plt.title('Store Size vs Total Customer Visits')
plt.show()
```

The total customer visits show a correlation with store dimensions as displayed

Justification:

The scatter plot analyses the relationship between store size measured in square metres (m²) and the total annual customer visit counts. The scatter plot stands as an optimal tool to show the intensity and characteristics of connections between continuous variables since it helps determine variable influence levels. The visual presentation enables ChrisCo to asses whether larger retail outlets lead to increased customer traffic which stands vital for store expansion and resource allocation decisions. The principal advantage of using scatter plots occurs when operators need to identify clusters alongside recognizing outliers together with linear and nonlinear trends within their data.

Findings:

The store size generally correlates positively with customer visits—indicating that larger stores tend to attract more shoppers—this factor alone doesn't fully explain performance. Medium-sized stores (1,000–3,000 m²) that exceed 100,000 annual visits prove that other factors like market investment, local population density, product variety, and staff quality play significant roles in driving success. Conversely, some large stores underperform despite their scale, likely due to poor location choices, inefficient operations, or weak market presence. The correlation heatmap supports these observations, reinforcing store size as a major but not exclusive driver of customer traffic, and helping categorize stores by performance based on visit volumes relative to their size.

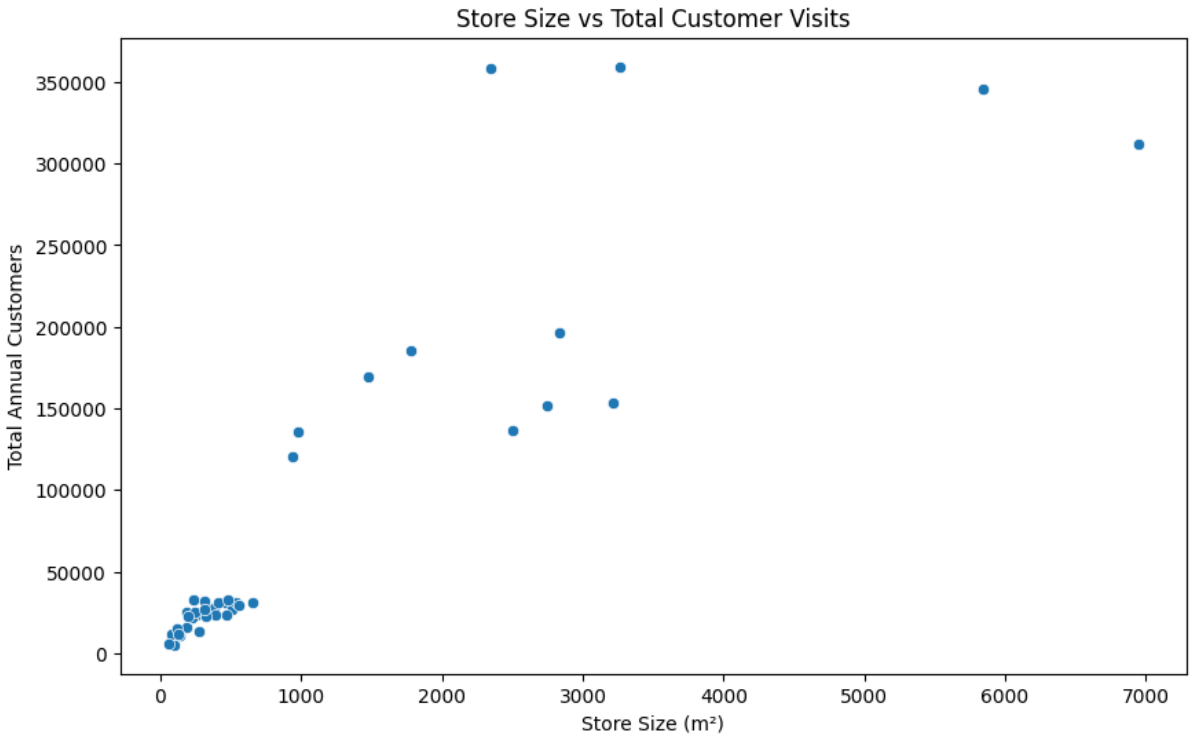


Figure 6:Store Size vs Total Customer Visits (Scatter Plot)

Total customer visits show a positive relationship to store size but different stores of equivalent dimensions perform differently.

2.6 Visualization 6: Boxplot of Customer Visits per Store

```
plt.figure(figsize=(12, 6))
# Use 'store_summary' instead of 'store_daily_customers'
# Use 'Store' for x-axis and 'DailyCustomers' for y-axis
sns.boxplot(x='Store', y='DailyCustomers', data=store_summary)
plt.xlabel('Store Code')
plt.ylabel('Number of Customers')
plt.title('Customer Visits Distribution per Store')
plt.xticks(rotation=90)
plt.show()
```

presents Customer Visits Distribution per Store.

Justification:

The boxplot functions efficiently to display how customer visit data spreads across separate stores while showing their variability levels. The visualization shows essential distribution data points including median and interquartile ranges with outlier indications for every store in a space-efficient and contrastive manner. The graphical representation allows easy performance assessment of individual stores to identify outliers as well as detect extreme results. The chart enables store-level benchmarking at ChrisCo since it shows which locations maintain steady traffic patterns alongside those that significantly differ from the norm.

Findings:

uses a boxplot to effectively show how customer visits vary across ChrisCo stores. It highlights key distribution points like medians, ranges, and outliers, making it easy to compare store performance. Most stores have stable customer numbers, while a few significantly outperform others, exceeding 300,000 visits annually—likely urban flagship locations. In contrast, some stores show lower or inconsistent traffic, possibly due to being new, niche, or underperforming. The visual reveals wide variability, suggesting the need for customer volume-based segmentation and further analysis of both high and low performers to improve operations.

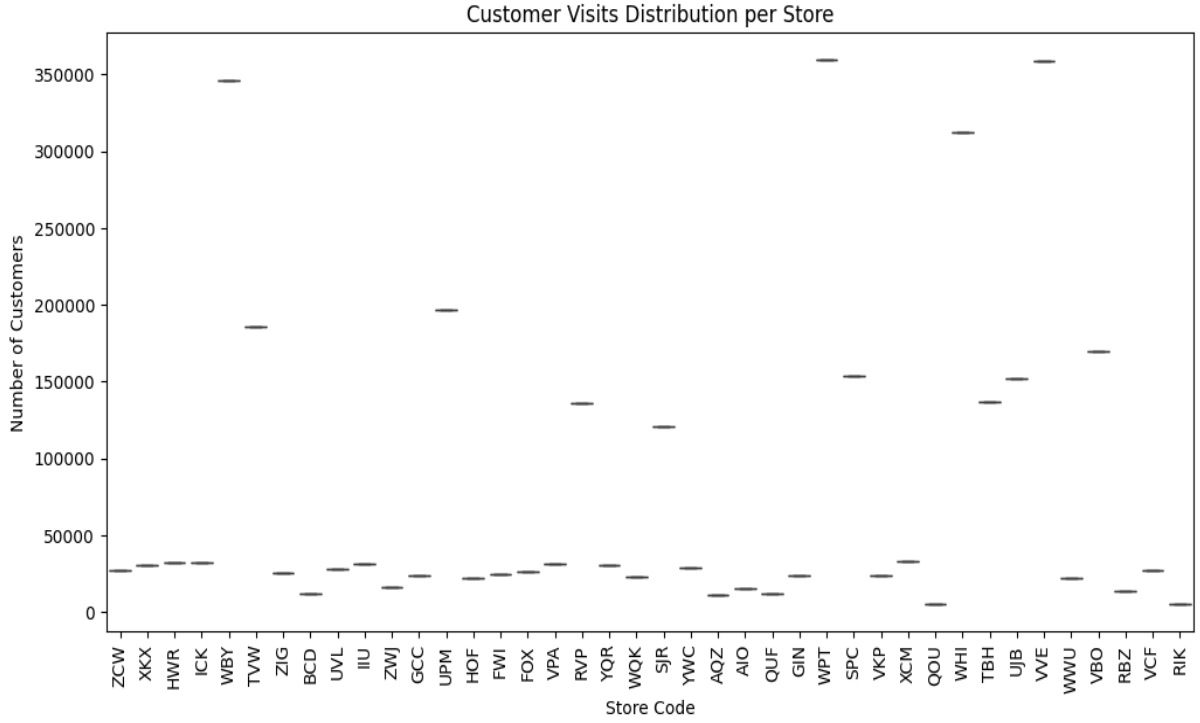


Figure 7:Customer Visits Distribution per Store (Boxplot)

The analysis shows how visit numbers differ between stores while identifying rare cases and validating groups that separate based on customer numbers.

2.7 Visualization 7: Line Plot of Monthly Average Customer Visits (for each Store)

```
# The 'store_daily_customers' variable was not defined. Replace it with 'cust_data'.
cust_data['Date'] = pd.to_datetime(cust_data['Date'])
cust_data['Month'] = cust_data['Date'].dt.month
# 'StoreCode' and 'DailyCustomers' columns do not exist in 'cust_data'.
# Replace with 'Store' and 'Customers'.
monthly_avg_visits = cust_data.groupby(['Store', 'Month'])['Customers'].mean().reset_index()

plt.figure(figsize=(12, 6))
# Replace 'StoreCode' with 'Store' to match the groupby.
```

```
for store in monthly_avg_visits['Store'].unique():
    store_data = monthly_avg_visits[monthly_avg_visits['Store'] == store]
    # Replace 'DailyCustomers' with 'Customers' to match the groupby.
    plt.plot(store_data['Month'], store_data['Customers'], label=store)
plt.xlabel('Month')
plt.ylabel('Average Customer Visits')
plt.title('Monthly Average Customer Visits per Store')
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.show()
```

The monthly average number of customer visits to stores.

Justification:

This line plot reveals the seasonal trends of customer visits to each store through monthly average data points which help reveal performance stability and behavioural patterns of customers. By averaging customer numbers per month, the visual smooths out daily fluctuations and offers a clearer view of general trends. A colouring scheme applies to the 40 stores to show performance differences across the year. The visual format proves especially helpful during operational planning since it reveals both peak customer times and stores which need assistance.

Findings:

The visualization displays three main performance groups among stores which range from high to medium to low. The top group of stores reaches monthly customer counts which remain at or exceed 1,000 visitors throughout the entire year. The stores display both high performance levels and operational stability which makes them suitable for potential expansion or benchmarking strategies. A middle tier of stores averages around 400–600 visits per month, with relatively minor fluctuations throughout the year. The stores in this lowest performing category receive less than 100 customer visits per month while several locations show late opening or partial business activity through early month data gaps and flat lines.

The majority of stores do not experience high seasonal impacts but the highest-performing branches demonstrate small month-to-month variations which potentially stem from local activities and marketing initiatives and weather conditions. Future marketing plans together with staffing decisions can be supported by this observed data. The presented figure confirms previous observations about store segmentation and reveals critical information to improve both successful and struggling branches of operation.

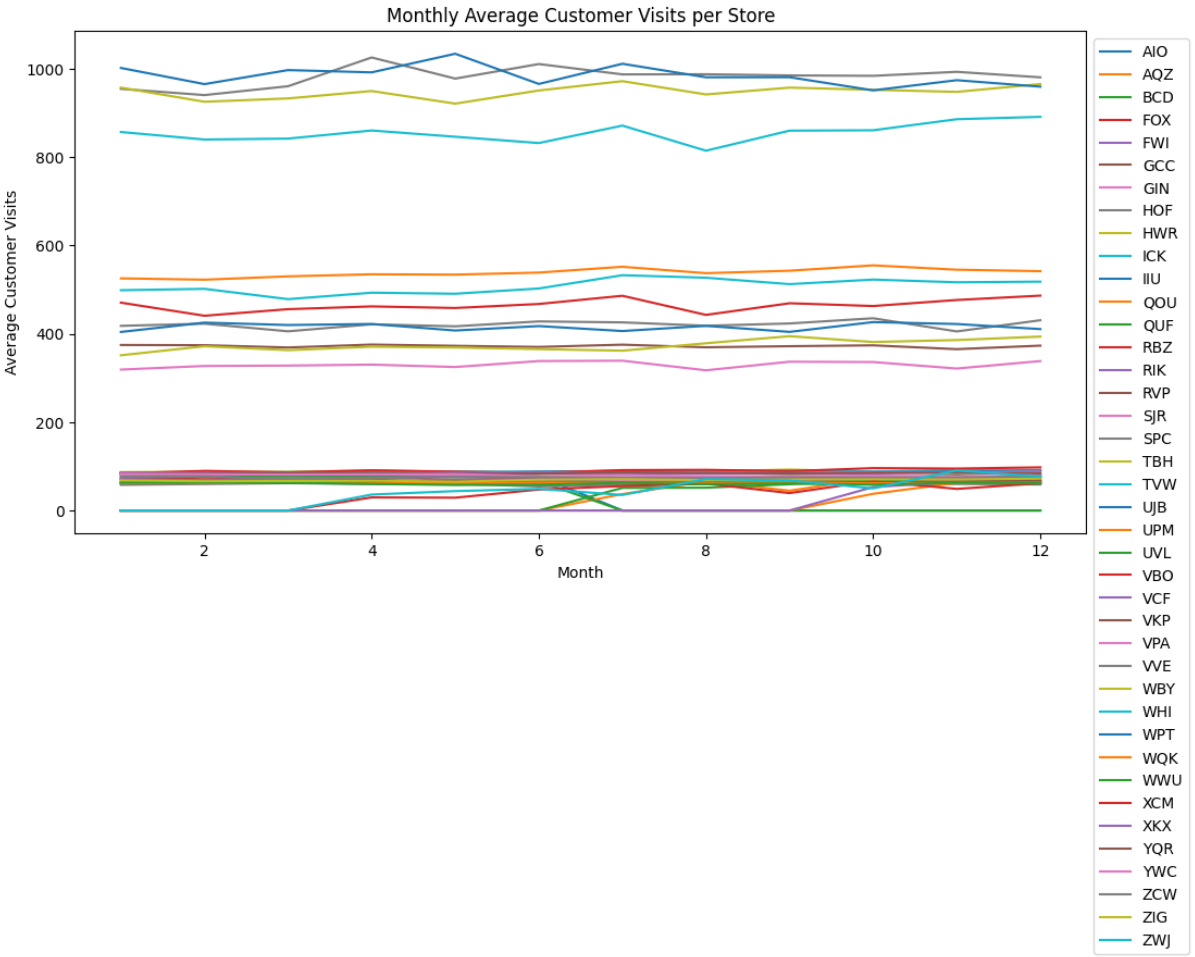


Figure 8: Monthly Average Customer Visits per Store.

Monitoring continuous store performance along with tracking seasonal customer behavior or performance abnormalities becomes possible through the use of monthly average data.

2.8 Visualization 8 Customers Trends for new and closed stores.

```
import pandas as pd
import matplotlib.pyplot as plt

# Convert customer data to a pivot table: rows = Date, columns = Store, values = Customers
pivoted_data = cust_data.pivot_table(index='Date', columns='Store', values='Customers')
pivoted_data.index = pd.to_datetime(pivoted_data.index)
pivoted_data.sort_index(inplace=True)

# Function to identify store changes
def identify_store_status(data):
    first_window = data.head(30).sum()
    last_window = data.tail(30).sum()
```

```
    new = list(data.columns[(first_window == 0) & (last_window > 0)])
    closed = list(data.columns[(first_window > 0) & (last_window == 0)])
    low = list(data.columns[data.sum() < data.sum().quantile(0.05)])

    return new, closed, low

# Run store status detection
new_stores, closed_stores, low_activity_stores = identify_store_status(pivoted_data)

# Display findings
print("Newly opened stores:", new_stores)
print("Closed stores:", closed_stores)
print("Very low activity stores:", low_activity_stores)

# Visualise changes if any
if new_stores or closed_stores:
    fig, ax = plt.subplots(figsize=(14, 7))

    for store in new_stores:
        ax.plot(pivoted_data.index, pivoted_data[store], label=f"New: {store}")
    for store in closed_stores:
        ax.plot(pivoted_data.index, pivoted_data[store], label=f"Closed: {store}")

        ax.set(title="Customer Trends for New and Closed Stores", xlabel="Date",
ylabel="Customers")
    ax.legend()
    ax.grid(True)
    plt.tight_layout()
    plt.show()
```

The stores that shows the customers trends for new and closed stores.

Justification:

A methodology detects new and closed establishments by analysing customer patterns from the beginning and end of the year. The new store designation includes locations that start inactive and then become active while closed stores begin with early activity and later stop all activity. The performance evaluation contains a check of stores ranking within the lowest 5% of customer visit count. This approach follows a simple data-driven method which explains the observed plot trends.

Findings:

The newly opened shops AQZ and QOU along with QUF recorded growing customer visits which reached daily figures exceeding 200 customers. The customer visits at AIO and BCD stores gradually decreased until reaching zero after store closures. The business performance of AIO indicated by its early high traffic indicates that store closure was not performance-based. Varied expansion trends across new stores seem to stem from distinct market demand patterns and

organizational strategies. The study creates a transparent depiction of retail store life phases by analysing daily store records. The visual confirmation of store change statuses also helps in decision-making for future opening or closing or improvement initiatives of retail facilities. The success measures of new outlets help forecast robust performance and their weak counterparts reveal operational weaknesses which prove beneficial for strategic management.

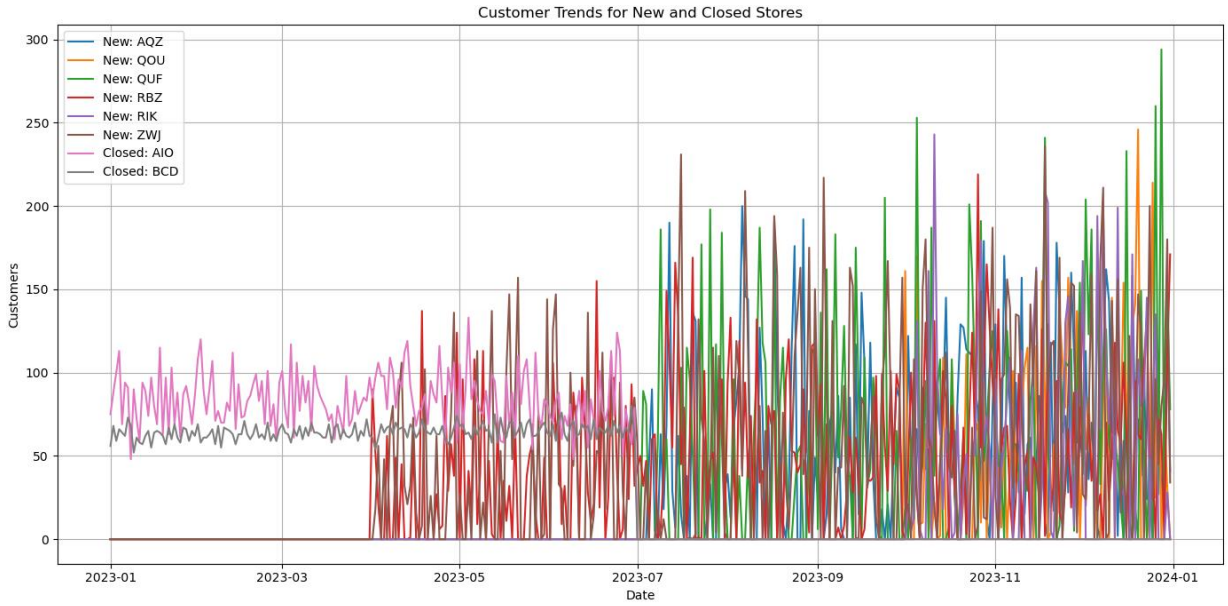


Figure 9:Customer trends for new and closed stores.

The daily visit tracking method reveals patterns of newly opened stores which attract more customers and reveals how closed stores show zero visit volume.

The performance trends at ChrisCo stores emerge distinctly from their retail locations in their established branches. Healthy stores experience uniform customer activity levels that directly match promotional costs and size characteristics and workforce counts with minimal seasonal effects. The performance of low-volume stores emerges from system issues or because of new business entry points. Store customer volume segmentation provides information about essential areas for improvement using record data of openings and closures. Store performance benchmarking and strategic resource optimization activities across ChrisCo networks can be supported by the collected data.

Conclusion:

The visual analytics analysis conducted performance and traffic evaluation of ChrisCo's UK stores based on five available datasets. Numerous performance metrics including marketing expenses and facility measurements and personnel staffing proved to impact customer traffic but marketing proved to be the most influential element. Performance segments identified distinct levels among business units and over time the top-performing outlets maintained consistent rankings. The examination discovered both newly established shops and those that were shuttered thus helping researchers study store life-span behaviour. The project demonstrates how data visualization techniques create substantial worth through their application.

References:

1.Hunter, J.D. (2007) “Matplotlib: A 2D Graphics Environment,” Computing in science & engineering, 9(3), pp. 90–95. Available at: https://doi.org/10.1109/mcse.2007.55.

2.Matplotlib — visualization with python (no date) Matplotlib.org. Available at: https://matplotlib.org (Accessed: March 25, 2025).

3.Mckinney, W. (2017) Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. Sebastopol: O’Reilly Media.

4.pandas.DataFrame — pandas 2.2.3 documentation (no date) Pydata.org. Available at: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html (Accessed: March 25, 2025).

5.Seaborn: Statistical data visualization — seaborn 0.13.2 documentation (no date) Pydata.org. Available at: https://seaborn.pydata.org (Accessed: March 25, 2025).

6.Waskom, M. (2021) “seaborn: statistical data visualization,” Journal of open source software, 6(60), p. 3021. Available at: https://doi.org/10.21105/joss.03021.

7.ostock, M., Ogievetsky, V. and Heer, J. (2011) “D3: Data-Driven Documents,” IEEE transactions on visualization and computer graphics, 17(12), pp. 2301–2309. Available at: https://doi.org/10.1109/TVCG.2011.185.

8.Plotly (no date) Plotly.com. Available at:https://plotly.com/python/plotly-express/ (Accessed: March 25, 2025).