Technical Report for AdventureWorks2022 Analysis

DA-NAT4 - Group 2

This report and the code within has been compiled by:
Abdulrahman Rajjoub
Alice Venables
Dawit Atreso
Elsheikh Ibrahim
Hamza Ibrahim
Pearl Asare

Regional Sales in Best Performing Country

Objective:

The purpose of this analysis was to understand both which country has the best sales, and then what the regional sales in that country were.

Methodology

Step 1: Database Connection and Relevant Libraries

```
import pyodbc
import pandas as pd
from matplotlib import pyplot as plt
```

First we import the relevant libraries - in this case pyodbc, pandas, and pyplot. pyodbc will allow us to connect through to SQL, pandas will allow us to execute the query, and pyplot will allow us to plot the graph itself.

```
connection = pyodbc.connect(r'DRIVER={ODBC Driver 17 for SQL Server};'
        r'SERVER=PIRALOS\SQLEXPRESS;'
        r'DATABASE=AdventureWorks2022;'
        r'Trusted_Connection=yes;'
        r'TrustServerCertificate=yes;')
```

Next we run the connection, connecting through to the SQL database, which will allow us to execute the query.

Step 2: Query Execution

For this question, we use two SQL queries - one to find which country has the best regional sales, and one to find the regional sales within that country. Once these are done, we can execute these as two dataframes, in order to store the data.

```
query1 = """SELECT
	st.CountryRegionCode AS Country,
	SUM(soh.SubTotal) AS TotalSales
FROM
	Sales.SalesOrderHeader soh
JOIN
	Sales.Customer c ON soh.CustomerID = c.CustomerID
JOIN
	Sales.SalesTerritory st ON c.TerritoryID = st.TerritoryID
GROUP BY
	st.CountryRegionCode
ORDER BY
	SUM(soh.SubTotal) DESC;"""
```

This first query uses a series of joins to get the data we need, and filters the data to just the regional sales and the country code, to sort that data by country.

```
df1 = pd.read_sql(query1, connection)
df1.head()
df1['SalesMillions']=df1['TotalSales']/1000000
```

Then we will set up the dataframe to store this data, which we will use later to generate a bar chart, to clearly display which country has the best regional sales! (In this case, it is the US!) We also view the dataframe, and then create a new column within it, called SalesMillions, as the previous sales data was only visible via 1eN notation.

With this done, we need to set up the second query, to find the regional sales for the US.

```
query2 = """SELECT
	st.Name AS Region,
	SUM(soh.SubTotal) AS TotalSales
FROM
	Sales.SalesOrderHeader soh
JOIN
	Sales.Customer c ON soh.CustomerID = c.CustomerID
JOIN
	Sales.SalesTerritory st ON c.TerritoryID = st.TerritoryID
WHERE CountryRegionCode = 'US'
GROUP BY
	st.Name
```

```
ORDER BY
	SUM(soh.SubTotal) DESC;"""
```

Once again we have a series of joins to select the relevant tables, and then we simply sort the data by the region. Here we have an additional WHERE clause in order to limit it to just the relevant data - in this case, ones where the country is the US.

```
df2 = pd.read_sql(query2, connection)
df2.head()
df2['SalesMillions']=df2['TotalSales']/1000000
```

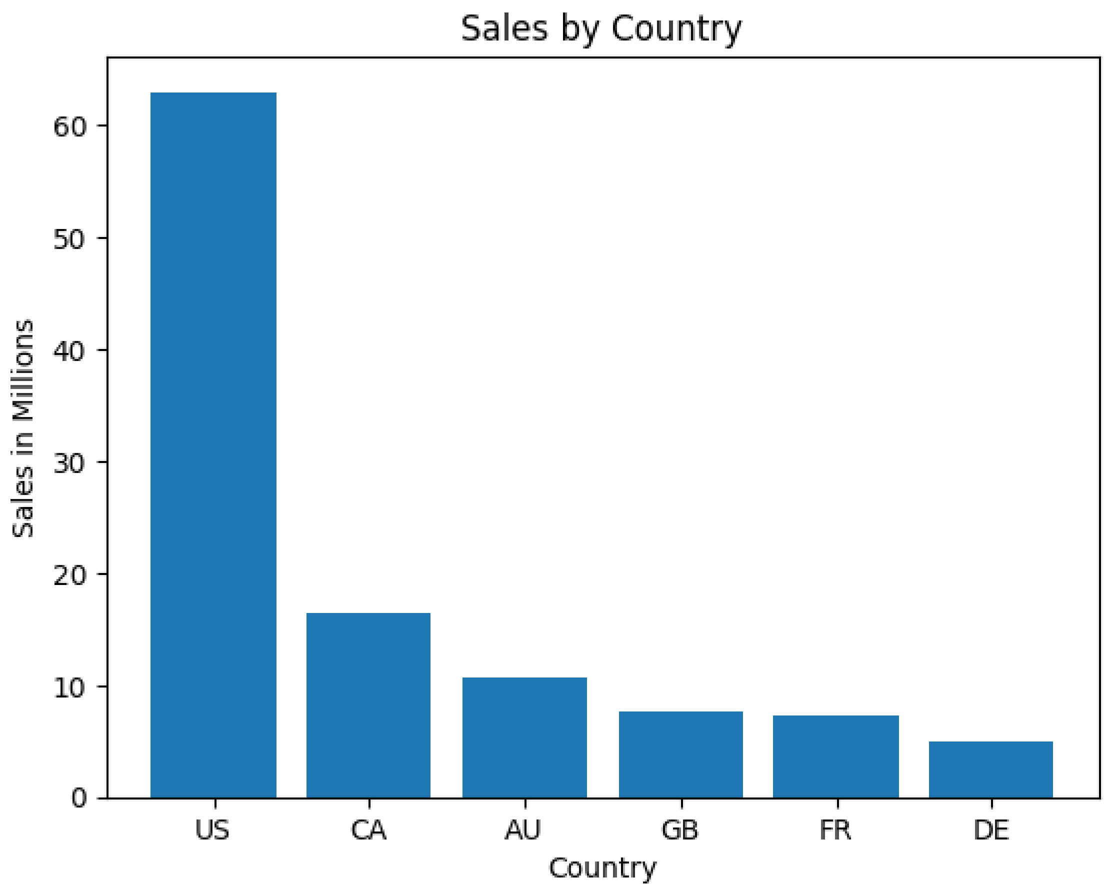
Once again, we set up the dataframe itself, view it, and divide it by one million once more, to create the SalesMillions column. This will allow us to have clearer bar charts once we set them up.

Step 3: Visualization

Here we create the plots, using our SalesMillions rows to have it presented in a cleaner, more effective manner. We plot them as bar charts for easier viewing, with the second being a horizontal one to make the region names easier to read.

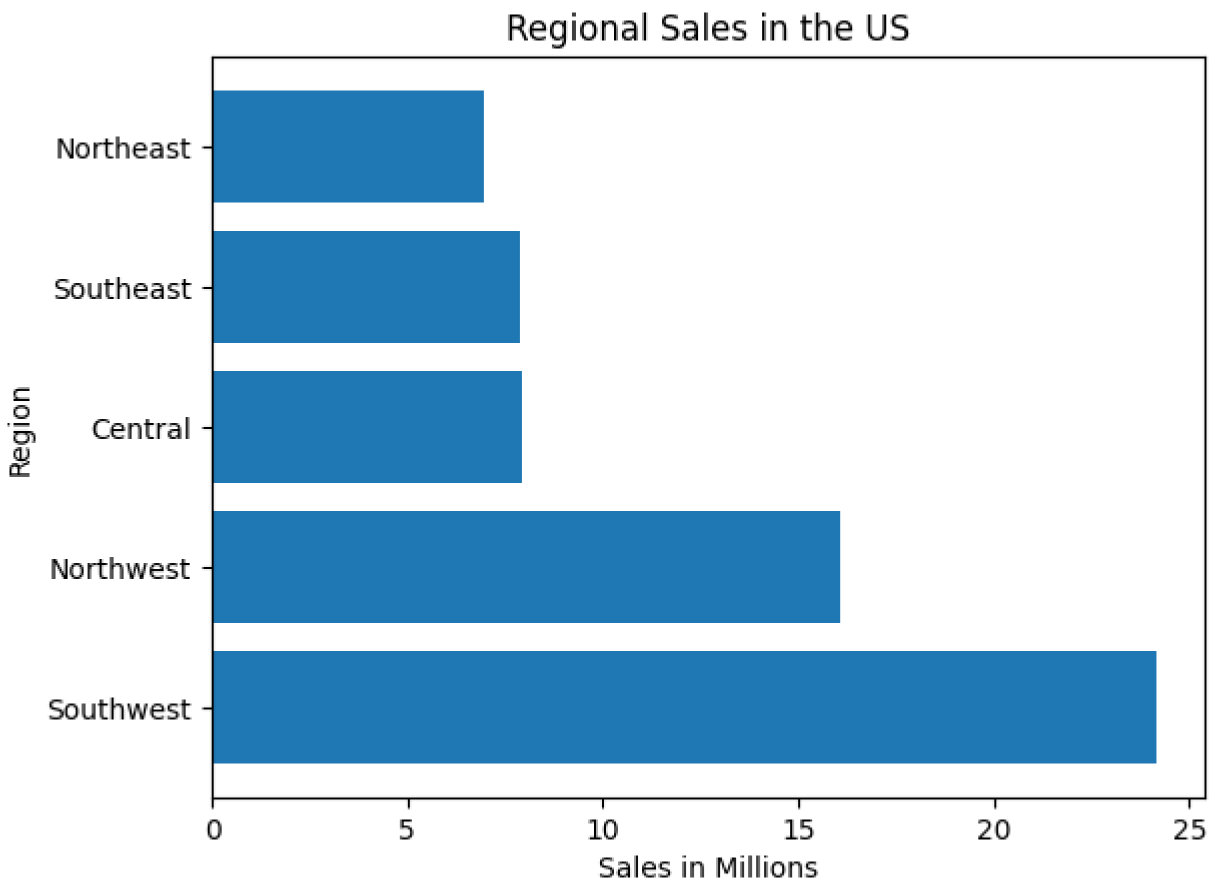
```
plt.bar(x=df1['Country'], height=df1['SalesMillions'])
plt.xlabel('Country')
plt.ylabel('Sales in Millions')
plt.title("Sales by Country")
plt.show()
```

This first bar chart shows the country with the greatest sales - in this case, it comes out as the US, which leads us to our second bar chart.



```
plt.barh(y=df2['Region'], width=df2['SalesMillions'])
plt.xlabel('Sales in Millions')
plt.ylabel('Region')
plt.title("Regional Sales in the US")
plt.show()
```

This one is a horizontal one as mentioned above, and displays the regional sales across the US.



Results

- The SQL query successfully combined joined tables together to determine the country with the greatest sales, and the regional sales within that country.
- A clear representation of which country had the most sales and the regional sales, resulting in exceedingly clear sales data.

Conclusion

This analysis provides some fascinating insights into the regional sales data - whilst it is perhaps predictable that the US would have the highest sales, given its population size and the stores seem to be primarily based there, the amount that it dwarfs other areas is surprising. Furthermore, Northwest having almost double the previous three area's sales is surprising, and South West's absolute dominance in sales by almost 10 million is incredibly shocking. This level of disparity is something that could be investigated further, as it may be an anomaly, but if it isn't it could indicate a need to further focus on this area.

Relationship between Annual Leave Taken and Bonus

Objective:

The purpose of this analysis was to understand the relationship between the amount of annual leave taken and the bonuses the employees receive, using data from the AdventureWorks2022 database.

Methodology

Step 1: Database Connection and Relevant Libraries

A connection to the database was established using the pyodbc library in Python. The connection used Windows Authentication for security and specified the SQL Server instance and database.

```
import pyodbc # first we import the relevant modules for this process
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np

connection = pyodbc.connect(
   r'DRIVER={ODBC Driver 17 for SQL Server};'
   r'SERVER=DESKTOP-P4JB2O7\SQLEXPRESS;'
   r'DATABASE=AdventureWorks2022;'
   r'Trusted_Connection=yes;'
   r'TrustServerCertificate=yes;' # then we set up the connection to the server
)
```

Here we also import numpy using its common alias, as we'll be using it later to chart the trend line for the scatter graph we're making.

Step 2: Query Execution

An SQL query was designed to calculate the total annual leave hours by employees, and their bonuses. We use an inner join to ensure that we only get the hours where there are bonuses to check them against. Then we execute the query to create a dataframe.

```
query = """
select sp.Bonus, sp.BusinessEntityID, (e.VacationHours + SickLeaveHours) as
total_annual_leave_Hours
from sales.SalesPerson as sp
Inner join HumanResources.Employee as e
```

```
ON sp.BusinessEntityID=e.BusinessEntityID""" # we run this query, selecting the relevant columns from SQL, and combine vacation hours and sick leave hours to find the total annual leave hours.


df = pd.read_sql_query(query, connection) # finally we set up the dataframe itself


connection.close() # Then we close the connection to preserve memory and security.

print(df) # We can now print the dataframe to ensure that it is all correct!
```

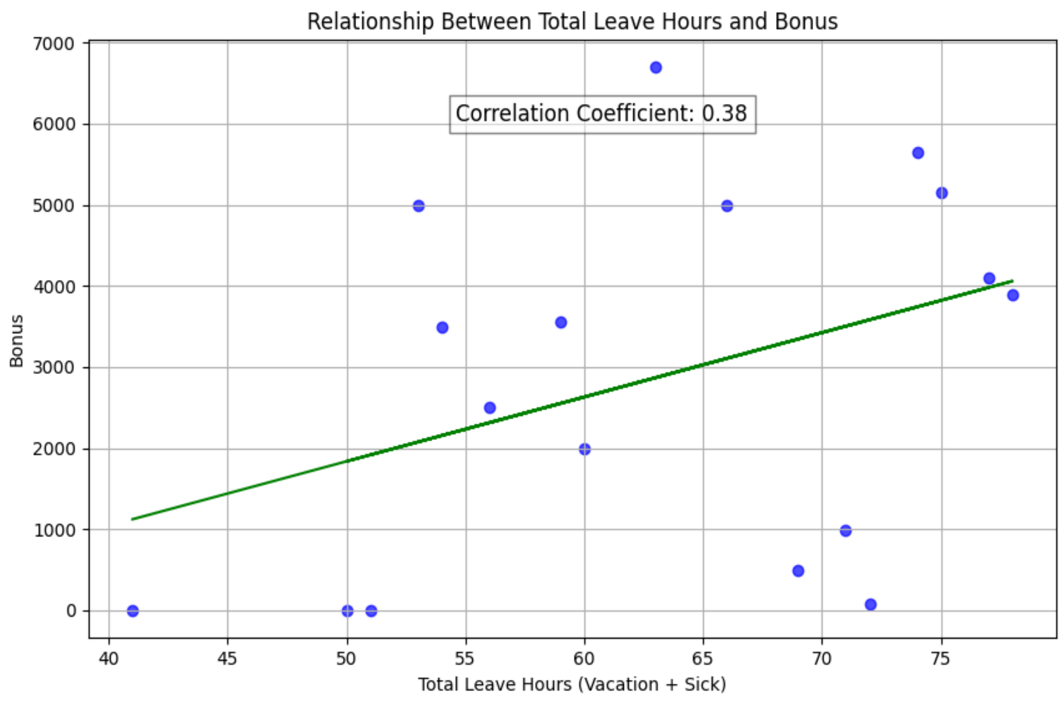
As the comments explain we print the dataframe to ensure everything is correct, and ensure that the SQL query was executed properly. After confirming this, we can move onto visualising the data that we have.

Step 3: Visualization

Here we set up the method to calculate the coefficient between the two values, and define what our x and y axes will be. Then, we set up a scatter plot to display these results, using our previously defined x and y, and use our values to calculate the trend line. Finally, we add a note displaying the correlation coefficient on the chart, to help make it clear the trend - in this case, that as the annual leave increases, as do the bonuses!

```
x = df['total_annual_leave_Hours']
y = df['Bonus']
m, b = np.polyfit(x, y, 1)

correlation_coefficient = np.corrcoef(x, y)[0, 1]
print(f"Correlation Coefficient: {correlation_coefficient:.2f}")


plt.figure(figsize=(10, 6))
plt.plot(x, m*x + b, color='green', label=f'Correlation Line: y = {m:.2f}x + {b:.2f}')
plt.scatter(x, y, alpha=0.7, color='blue')
plt.title('Relationship Between Total Leave Hours and Bonus')
plt.xlabel('Total Leave Hours (Vacation + Sick)')
plt.ylabel('Bonus')
plt.text(
    x.max() * 0.7, y.max() * 0.9,
    f'Correlation Coefficient: {correlation_coefficient:.2f}',
    fontsize=12, color='black', bbox=dict(facecolor='white', alpha=0.5)
)

plt.grid()
plt.show()
```

Various steps have been taken to make the graph more readable, including altering the font size and the transparency of the text, along with providing a trend line to make the overall trend easier to understand. The graph produced by this is below.



Results

- The SQL query successfully combined sick leave and vacation leave to calculate the annual leave, and filtered it to just those who had bonuses.
- A clear representation of how total annual leave matched with total vacation hours displayed as a scatter graph, with a trendline helping to display the correlation.

Conclusion

This analysis provides some fascinating insights into the relationship between annual leave and the bonuses that employees earn, as it indicates that higher rates of annual leave can actually result in more bonuses, potentially implying that employees who are given more breaks work better. However, we only have 17 data points to work from, and so this data is ultimately inconclusive, due to a small sample size to work with.

Relationship Between Country and Revenue

Objective

The purpose of this analysis was to understand the relationship between different countries and their respective revenues using data from the AdventureWorks2022 database.

Methodology

Step 1: Importing Libraries and Database Connection

A connection to the database was established using the pyodbc library in Python. The connection used Windows Authentication for security and specified the SQL Server instance and database.

```
import pyodbc
import pandas as pd
from matplotlib import pyplot as plt
# Establish connection to the AdventureWorks2022 database
connection = pyodbc.connect(
        r'DRIVER={ODBC Driver 17 for SQL Server};'
        r'SERVER=DESKTOP-G1HPGDT\SQLEXPRESS;'
        r'DATABASE=AdventureWorks2022;'
        r'Trusted_Connection=yes;'
        r'TrustServerCertificate=yes;'
)
```

For this query we only need pyodbc, pandas, and pyplot, as we will not be needing to create trend lines later on. Once we are connected through, we can execute the query.

Step 2: Query Execution

An SQL query was designed to calculate the total revenue for each country. The data was grouped by CountryRegionName and sorted in descending order by revenue.

```
# SQL Query to retrieve and aggregate revenue data
query = """SELECT SD.CountryRegionName as Country, SUM(SW.AnnualRevenue) AS
Revenue
FROM Sales.vStoreWithAddresses as SD
JOIN Sales.vStoreWithDemographics as SW
ON SD.BusinessEntityID=SW.BusinessEntityID
GROUP BY CountryRegionName
ORDER BY Revenue DESC;
"""
```

```
# Execute the query and load the results into a DataFrame
df = pd.read_sql(query, connection)

#  Create a new column to show the revenue in millions.
df['RevenueMill']=df['Revenue']/1000000


# Print the data to verify
print(df)
```

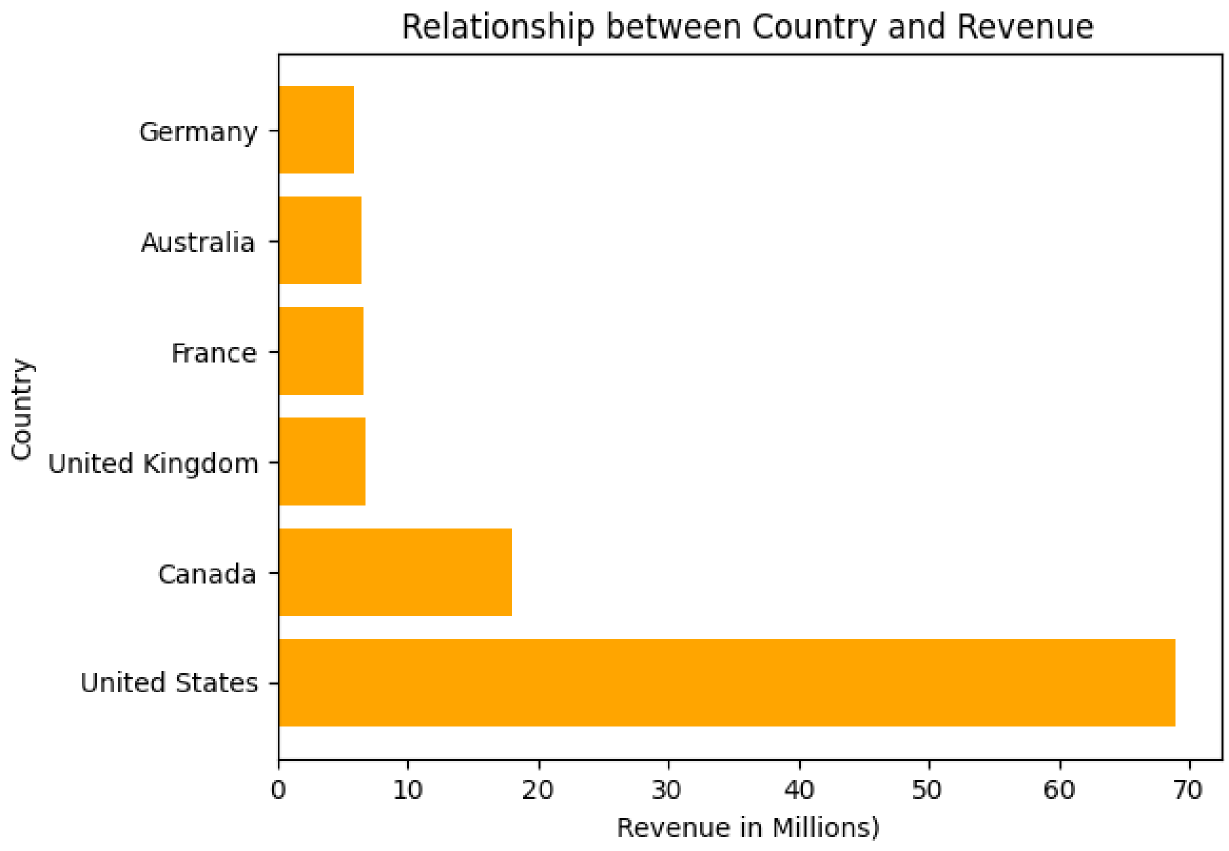
This query allows us to easily see the countries and their revenues, and makes use of some of the views found preloaded in the AdventureWorks2022 database to reduce the amount of joins needed. This reduction in joins in turn allows for quicker query execution, and more efficient code execution overall. Furthermore, we apply a final change to the dataframe, creating a new column which displays the revenue in millions, as otherwise it will display it in 1eN notation (eg 1e6, 1e7, etc).

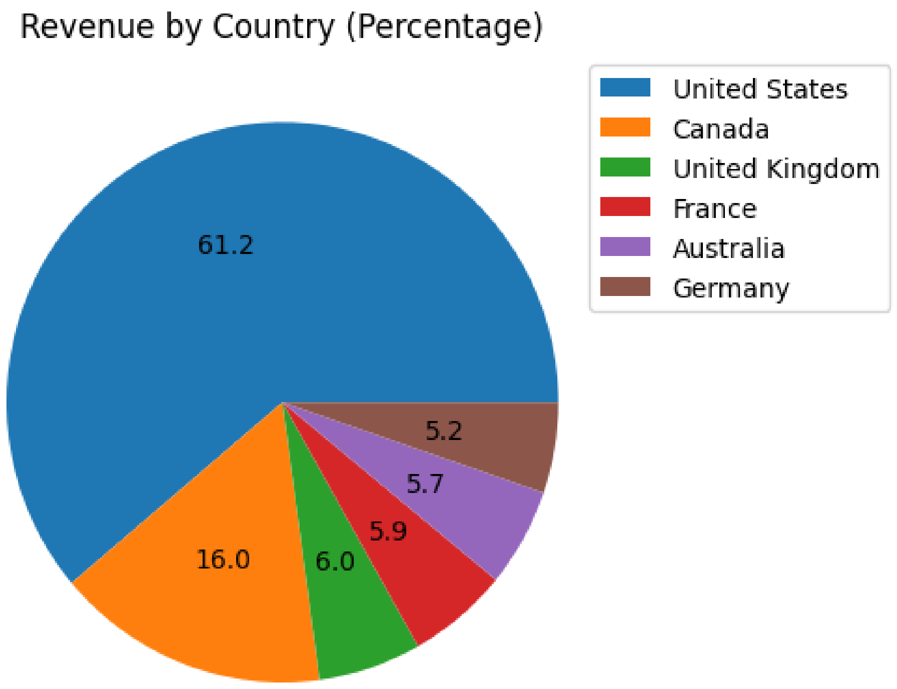
Step 3: Visualization

A horizontal bar chart was generated to visualize the relationship between countries and their revenues. This visualization helps identify the countries contributing the most to total revenue.

```
# Create a bar chart
plt.barh(df['Country'], df['RevenueMill'], color='orange')
plt.title("Relationship between Country and Revenue")
plt.ylabel("Country")
plt.xlabel("Revenue in Millions)")
plt.show()

# We also create a pie chart here for more understanding and greater visualisations of the data.
plt.pie(df['Revenue'], autopct='%0.1f')
# This allows us to set the values within the pie chart to display the revenue as a percentage of the total, making clear exactly how much we're looking at.
plt.legend(df['Country'], loc='upper right', bbox_to_anchor=(1.4, 1))
# This allows us to set a legend for the pie chart, rather than using the default labels, and sets the legend to appear in the upper right without overlapping the chart itself.
plt.title('Revenue by Country (Percentage)')
plt.show()
```

We choose a horizontal bar chart here as it allows us to more easily view the country names without the text becoming cramped, and without the need to rotate the text. This in turn allows us to easily visualise the data that we are working with. We also use a pie chart here, as whilst they are normally avoided, it makes for a very clear representation of how much the US accounts for revenue as a rough percentage - in this case, well over half! The charts produced here are displayed below.





Step 4: Closing the Connection

The connection to the database was closed to ensure proper resource management and security.

```
# Close the database connection

connection.close()
```

Results

- The SQL query successfully retrieved revenue data for all countries in the dataset.
- A clear visual representation was created showing countries and their respective revenues, with the highest-revenue country (The US) standing out.

Conclusion

This analysis highlights the significant differences in revenue contributions across countries. The horizontal bar chart provides an intuitive way to identify high-performing regions. Here it is overwhelmingly clear that the United States has vast amounts more revenue than any other country, accounting for over half of the revenue on its own. This is further evidenced by the pie chart, which makes clear the amount that the US contributes to the overall profits as a percentage. This information can guide business decisions related to market prioritization and resource allocation.

Relationship Between Sick Leave and Job Titles/Person Types

Objective

This analysis investigates the relationship between sick leave hours and the following:

1. **Job Titles**: Understanding which job roles accumulate the sickest leave.
2. **Person Types**: Exploring variations in sick leave based on employee classifications.

Methodology

Step 1: Database Connection

We connected to the AdventureWorks2022 database using the pyodbc library in Python. This connection allows secure and efficient querying of the database. Then we use pandas to read the sql query and compile the dataframe, and finally we use pyplot from the matplotlib library in order to plot the dataframe as a useful chart.

```
import pyodbc
import pandas as pd
from matplotlib import pyplot as plt

# Establish database connection
connection = pyodbc.connect(
        r'DRIVER={ODBC Driver 17 for SQL Server};'
        r'SERVER=DESKTOP-G1HPGDT\SQLEXPRESS;'
        r'DATABASE=AdventureWorks2022;'
        r'Trusted_Connection=yes;'
        r'TrustServerCertificate=yes;'
)
```

Step 2: Analysing Sick Leave by Job Title

We queried the database to calculate the total and average sick leave hours for each job title. The data was sorted in descending order by total sick leave hours. In this case we limit it to the top 20, as otherwise we end up with an overwhelming amount of data that doesn't fit onto the graph we are trying to make.

```
queryjobtitle = """
SELECT TOP 20
        e.JobTitle,
        AVG(e.SickLeaveHours) AS AvgSickLeaveHours,
        SUM(e.SickLeaveHours) AS TotalSickLeaveHours,
        COUNT(*) AS EmployeeCount
```

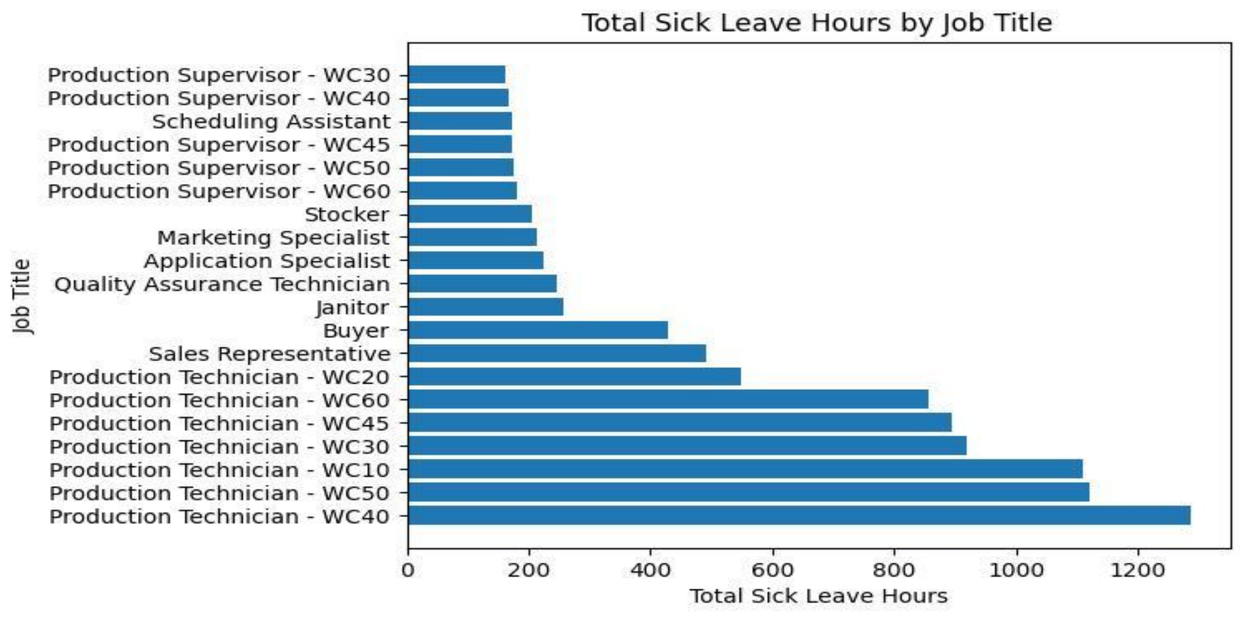
```
FROM
        HumanResources.Employee e
GROUP BY
        e.JobTitle
ORDER BY
        TotalSickLeaveHours DESC;"""

# Execute the query and load data into a DataFrame
dftitle = pd.read_sql(queryjobtitle, connection)

# Print the first 10 rows from data frame
print(dftitle.head(10))
```

Visualization

A horizontal bar chart was created to visualize the total sick leave hours by job title. We use a horizontal bar chart in order to better show the job titles in the graph, and then added labels to the x and y axes, and a title in order to make the data clearer!

```
# Plotting the data
dftitle_sorted = dftitle.sort_values('TotalSickLeaveHours', ascending=False)
plt.barh(dftitle_sorted['JobTitle'], dftitle_sorted['TotalSickLeaveHours'])
plt.xlabel('Total Sick Leave Hours')
plt.ylabel('Job Title')
plt.title("Total Sick Leave Hours by Job Title")
plt.show()
```

We sorted the data in order to have it display the graph in a single smooth curve, which allows for an easier time with the visualisation of the data itself. Once again, the chart is below.



Step 3: Analysing Sick Leave by Person Type

Another query explored the relationship between sick leave and employee classifications (PersonType). This analysis computed total and average sick leave hours for each classification. We take both the average sick leave and the total sick leave hours in order to provide a more comprehensive understanding of the data, as the numbers between the two different person types can vary greatly.

```
querypersontype = """
SELECT
        p.PersonType,
        AVG(e.SickLeaveHours) AS AvgSickLeaveHours,
        SUM(e.SickLeaveHours) AS TotalSickLeaveHours,
        COUNT(*) AS EmployeeCount
FROM
        HumanResources.Employee AS e
JOIN
        Person.Person AS p
        ON e.BusinessEntityID = p.BusinessEntityID
GROUP BY
        p.PersonType
ORDER BY
        TotalSickLeaveHours DESC;"""

# Execute the query and load data into a DataFrame
dfpersontype = pd.read_sql(querypersontype, connection)


# Print the first  5 rows from data frame
print(dfpersontype.head())
```

We print the head of the dataframe in order to better understand the data we are working with, and to ensure that in our visualisations we are able to use the right column names for the visualisations that are to follow.

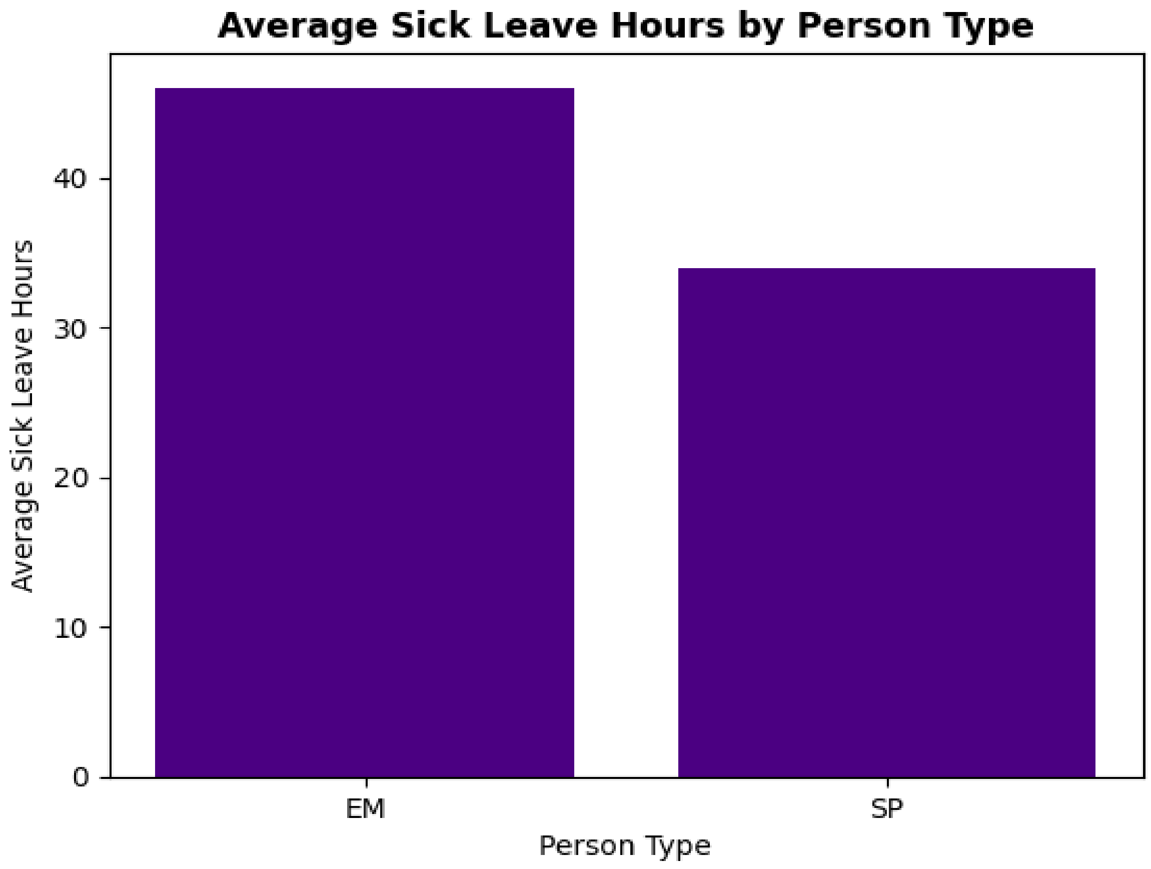
Visualizations

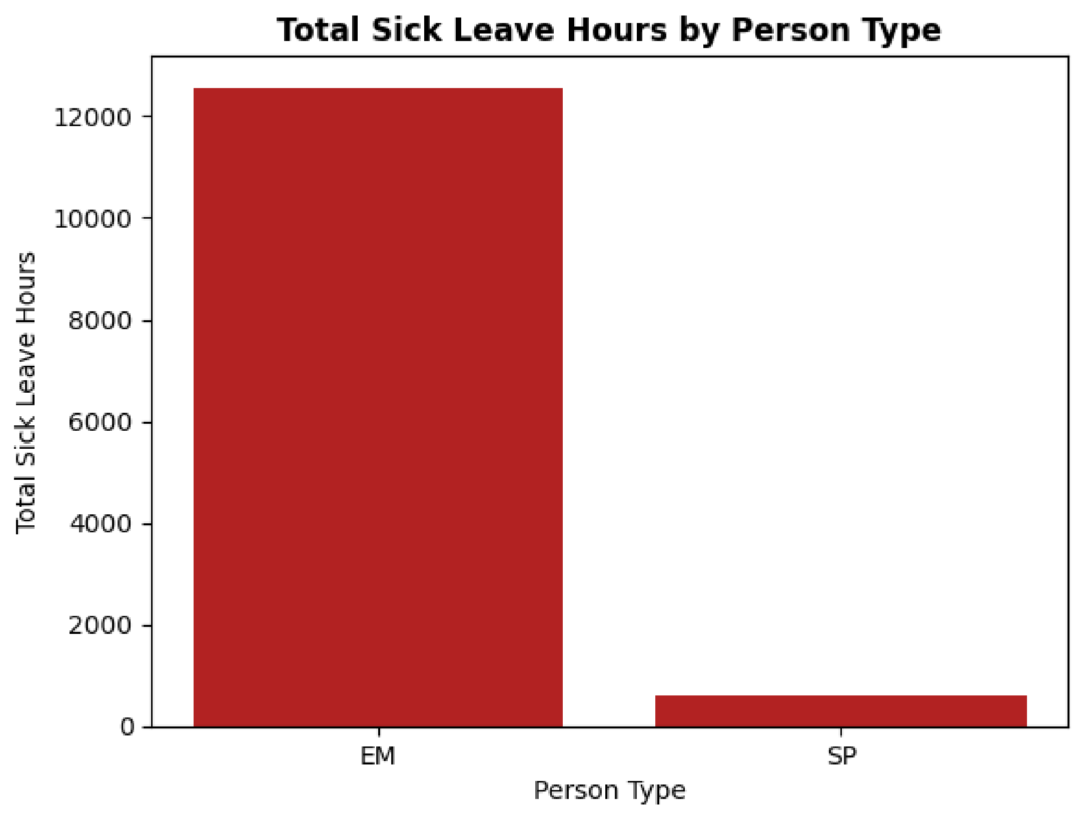
Two bar charts were generated:

1. Total sick leave hours by person type.
2. Average sick leave hours by person type.

```
# Total sick leave hours by person type
plt.bar(dfpersontype['PersonType'], dfpersontype['TotalSickLeaveHours'], color='blue')
plt.xlabel('Person Type')
plt.ylabel('Total Sick Leave Hours')
plt.title('Relationship between Person Type and Total Sick Leave Hours')
plt.show()

# Average sick leave hours by person type
plt.bar(dfpersontype['PersonType'], dfpersontype['AvgSickLeaveHours'], color='blue')
plt.xlabel('Person Type')
plt.ylabel('Average Sick Leave Hours')
plt.title('Relationship between Person Type and Average Sick Leave Hours')
plt.show()
```

For these we can just use vertical bar charts as the person type labels are rather short, allowing us to easily display them on vertical charts. We plot both the total and the average in order to better display the difference between the two of them, as the number of employees in each person type varied wildly. The charts are below.





Step 4: Closing the Connection

The database connection is then closed to free up resources and ensure security.

```
connection.close()
```

Results

- **Job Titles**:
  - Certain job roles, such as those with physical demands or high exposure, showed significantly higher sick leave hours.
- **Person Types**:
  - Variations in sick leave were observed based on employee classifications, indicating potential differences in work conditions or policies. However, whilst there is a large gap in the total hours taken as sick leave between the two person types, the average difference isn’t all that large, only totalling about 10 hours.

Conclusion

This analysis highlights key patterns in sick leave allocation, which could inform policy adjustments or targeted interventions to manage employee health and productivity effectively.

Relationship Between Store Trading Duration and Revenue

Objective
This analysis aims to display the relationship between how long a store has been open and the amount of revenue that that store generates, and investigates why that might be.

Methodology

Step 1: Database Connection

We connected to the AdventureWorks2022 database using the pyodbc library in Python. This connection allows secure and efficient querying of the database. Then we use pandas to read the sql query and compile the dataframe before using pyplot from the matplotlib library in order to plot the dataframe as a useful chart. Finally, we make use of numpy to plot the trend line for this data, in order to better analyse it.

```
# First we import the relevant python libraries.
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import pyodbc

connection = pyodbc.connect(r'DRIVER={ODBC Driver 17 for SQL Server};'
        r'SERVER=PIRALOS\SQLEXPRESS;'
        r'DATABASE=AdventureWorks2022;'
        r'Trusted_Connection=yes;'
        r'TrustServerCertificate=yes;')
```

This allows us to connect through to the server, accessing AdventureWorks2022, as per usual.

Step 2: Query Execution

For this question we only need one SQL query, in this case accessing the premade AdventureWorks view Sales.vStoreWithDemographics, as this houses all of the information we need. We take the average revenue, as this will give us a better understanding of any given store's performance, and group it by the year that the store opened to be able to view any overall trends based on how long the stores have been operating for. Then we simply compile the dataframe from the query, in order to work with it in python.

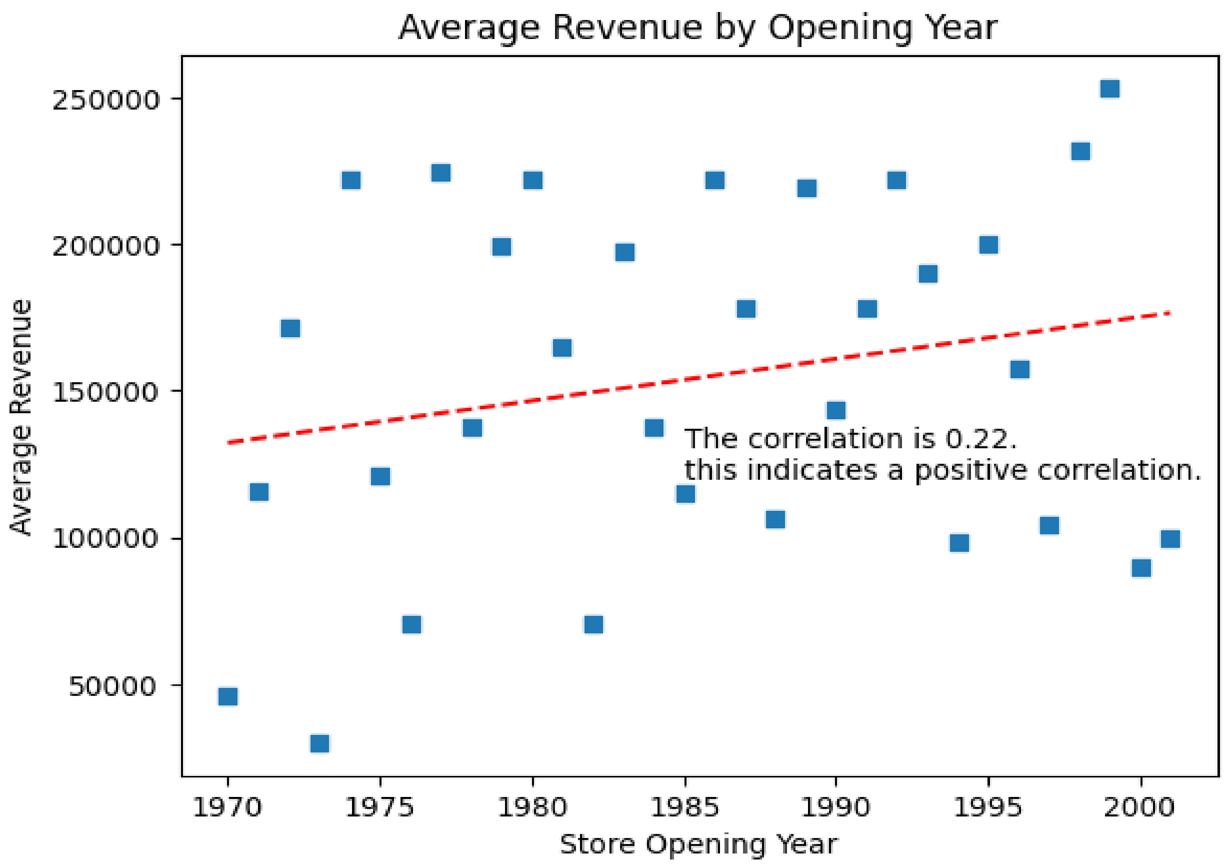
```
query5 = """SELECT YearOpened, AVG(AnnualRevenue) AS Average_Revenue
FROM Sales.vStoreWithDemographics
GROUP BY YearOpened
ORDER BY YearOpened"""

df5 = pd.read_sql(query5, connection)
```

Step 3: Visualization

For this question we utilise a scatter plot, as we will have multiple distinct variables that don't directly link to each other. As such, a scatter plot becomes our best option, in order to view the overall trends between the two values. Here we also use numpy to apply a trend line, and add axis labels and a title. Finally, we add text to display the correlation, and run a print function in order to display the correlation so we can confirm that it is correct. By using line styling and colour, we are able to make the trend line stand out more, allowing the insights gained from it to be easier to see.

```
# What is the relationship between store trading duration and revenue?
plt.scatter(x=df5['YearOpened'], y=df5['Average_Revenue'], marker='s')
# Once we have our scatterplot, we can add a trendline to it.
z = np.polyfit(df5['YearOpened'], df5['Average_Revenue'], 1)
p = np.poly1d(z)
plt.plot(df5['YearOpened'], p(df5['YearOpened']), linestyle='--', color='red')
plt.text(x=1985, y=120000, s="The correlation is 0.22. \nthis indicates a positive correlation.")
plt.xlabel("Store Opening Year")
plt.ylabel("Average Revenue")
plt.title("Average Revenue by Opening Year")
plt.show()

print(np.corrcoef(df5['YearOpened'], df5['Average_Revenue'])) # This gives us the correlation
between the two values.
```

As per usual, the visualisation created by the above code is below.



Step 4: Closing the Connection

Finally, as always, the database connection is then closed to free up resources and ensure security.

```
connection.close()
```

Results

- The SQL query successfully filters the data to select the average revenue for each size of store, and matches it to the relevant store sizes without the need for complex joins or multiple queries, by virtue of utilising already inbuilt views.
- This graph then provides a clear understanding of the overall trend, via the trendline. The data points themselves don't provide any clear correlation however, as they are too scattered for us to see it without assistance. However, after applying the trendline, the positive correlation becomes clear.

Conclusion

This analysis provides a fascinating insight into how the age of a store relates to the average revenue that it generates. Overall it seems that the revenue increases the more recently the store has opened - this could indicate that more recently opened stores are more profitable, however it could also be an indication that these stores have a larger size. What seems most likely to me, however, is that more modern stores are built in more profitable areas, and are better able to take advantage of things such as shopping centres in order to generate greater amounts of revenue. This does seem to indicate that overall stores are becoming more profitable, even with the initial investment in the creation of them.

Relationship Between The Size of the Store, Revenue, and Employee Count

Objective

This analysis aims to display the relationship between the size of the store, the revenue that the store generates, and the employee count of the store. As we're analysing three factors, we're going to have to present multiple charts, in order to fully understand each part.

Methodology

Step 1: Database Connection

We connected to the AdventureWorks2022 database using the pyodbc library in Python. This connection allows secure and efficient querying of the database. Then we use pandas to read the sql query and compile the dataframe before using pyplot from the matplotlib library in order to plot the dataframe as a useful chart. As we aren't using scatter plots, we don't need to make use of numpy for this.

```
# First we import the relevant python libraries.
import pandas as pd
from matplotlib import pyplot as plt
import pyodbc

connection = pyodbc.connect(r'DRIVER={ODBC Driver 17 for SQL Server};'
        r'SERVER=PIRALOS\SQLEXPRESS;'
        r'DATABASE=AdventureWorks2022;'
        r'Trusted_Connection=yes;'
        r'TrustServerCertificate=yes;')
```

Step 2: Query Execution

For this question we only need one SQL query, in this case accessing the premade AdventureWorks view Sales.vStoreWithDemographics, as this houses all of the information we need. We take the size of the store in square feet, and then take both the average annual revenue of the stores and the average number of employees, and match it to the size of the stores in square feet. This will allow us to utilise these columns to analyse these values once we have it in a dataframe.

```
# This opening section selects the square feet, the average annual revenue, and the average
number of employees per store.
query6 = """SELECT Squarefeet, AVG(AnnualRevenue) AS Average_Revenue,
AVG(NumberEmployees) AS Average_Employees
FROM Sales.vStoreWithDemographics
GROUP BY SquareFeet"""
```

```
df6 = pd.read_sql(query6, connection)
print(df6.head())
```

Then we compile the data into a dataframe, in order to be able to better analyse it. We print the head of the dataframe to confirm everything has gone through smoothly, and after confirming it has, we can move on to the visualisation of the data.

Step 3: Visualization

Here we are going to be using three different charts in order to properly display each relationship we wish to explore. We have to ensure that each axis is labelled and that the charts have titles - after which, we have added text in order to provide key insights into the data for any looking at them, without clogging up the charts too much.

```
plt.plot(df6['Squarefeet'], df6['Average_Revenue'], label='Size vs Average Revenue',
color='purple') # First we plot square feet vs revenue
plt.xlabel('Size in Square Feet') # We also need to label the axes to make it clear what each one
is
plt.ylabel('Average Revenue')
plt.title("Size vs Average Revenue") # And, of course, we need to add a title!
plt.text(x=30000, y=275000, s="Minimal returns past store \nsize 70000 square feet.") # Adding
this text will enable us to provide more insights into the data.
plt.show()
```

Our first chart plots the size in square feet vs revenue, and here we utilize a line plot over a scatter plot as with the data points we have, a line plot allows us to visualise it better. We have minimal data points and they tend to come clustered together with a somewhat linear increase, and so a line plot ends up visualising this rather well.

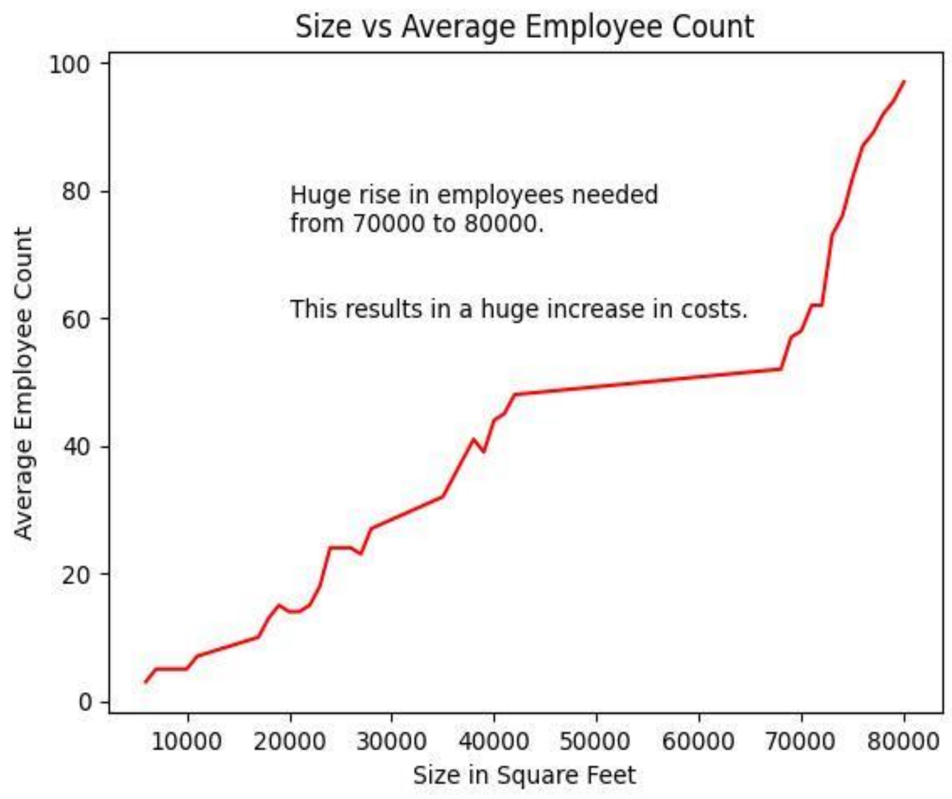
```
plt.plot(df6['Squarefeet'], df6['Average_Employees'], color='red') # We repeat the same process
as above twice more, for the other two possible combinations of how the data can look!
plt.xlabel('Size in Square Feet')
plt.ylabel('Average Employee Count')
plt.title("Size vs Average Employee Count")
plt.text(x=20000, y=60, s="Huge rise in employees needed \nfrom 70000 to 80000. \n\n\nThis
results in a huge increase in costs.")
plt.show()
```

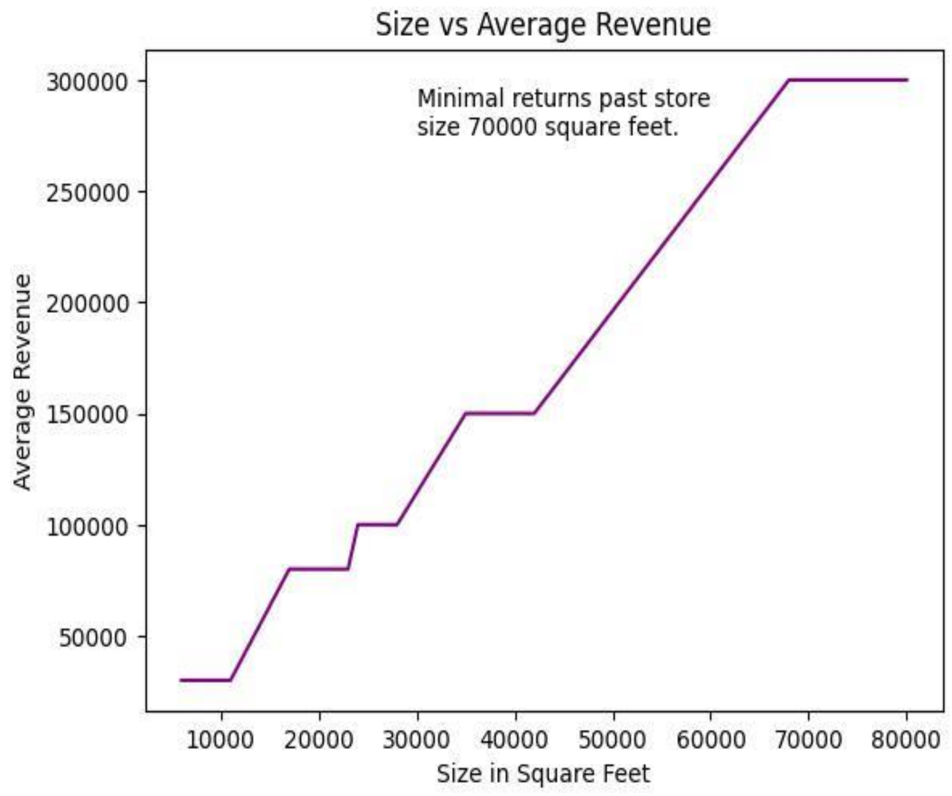
Our second chart plots square feet vs the average employee count, and here we once again use a line chart as it will allow us to clearly see the trends as the size increases. Once again the data points tend to come clustered together, due to the nature of the data, and so utilising a line chart here is helpful.

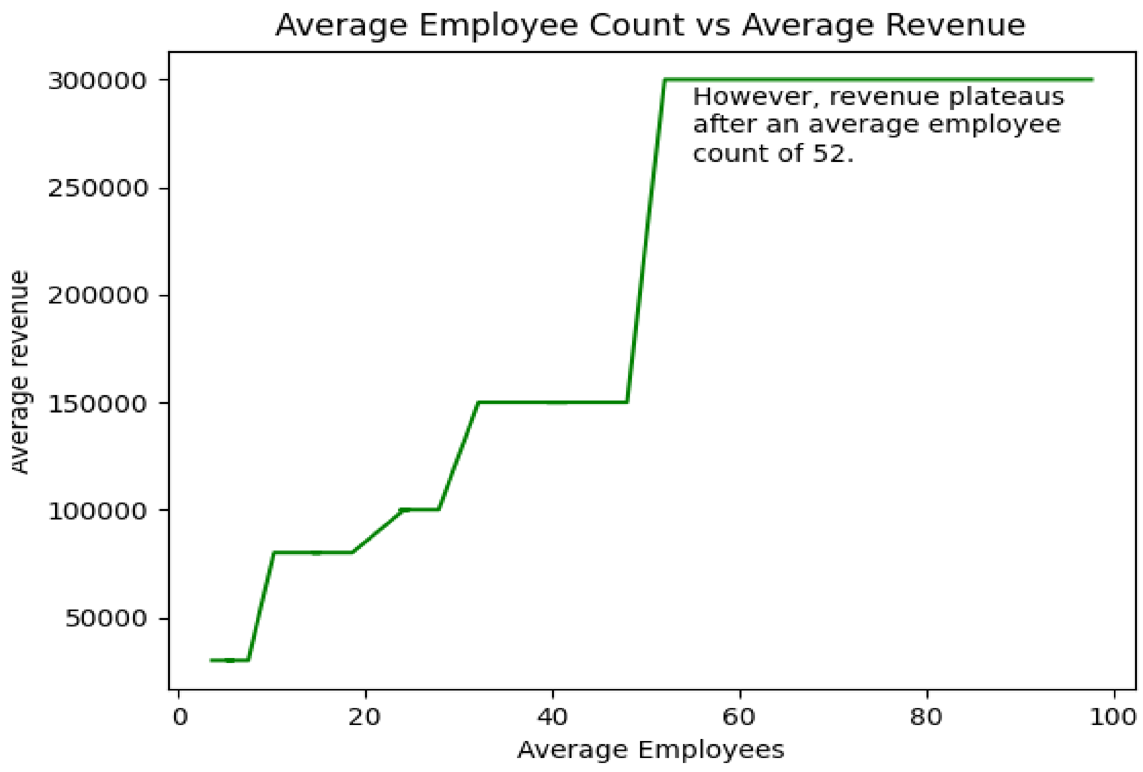
```
plt.plot(df6['Average_Employees'], df6['Average_Revenue'], color='green')
plt.xlabel('Average Employee Count')
plt.ylabel('Average Revenue')
plt.title("Average Employee Count vs Average Revenue")
plt.text(x=55, y=262000, s="However, revenue plateaus \nafter an average employee \ncount of 52.")
plt.show()
```

Finally, we plot the average employee count vs average revenue - as we saw in the last two, the data points tend to be rather clustered, and so a line chart works well.
The main reason that we don't plot all three of these lines on the same chart is that each of them requires different values on the axes, and so plotting them all on one chart would produce large amounts of chart junk and visual clutter, and make deriving insights from the charts harder, not easier. However, by plotting them on three separate charts, we are easily able to both see how they relate to each other, and the overall trends in these charts.

All three of the visualisations created are below.







Step 4: Closing the Connection

Finally, as always, the database connection is then closed to free up resources and ensure security.

```
connection.close()
```

Results

- The SQL query successfully filters the data to select all of the relevant columns - the size in square feet, the average of the employee count, and the average of the annual revenue. It does so without requiring multiple joins, as we are once again able to use a premade view to our advantage here, resulting in a clean, simple, and easy to execute SQL view.
- The graphs help us to see the overall trends very clearly, including a number of plateaus as various factors remain stable as others increase, indicating diminishing returns.

Conclusion

This analysis comes in three parts, each of them deeply fascinating. For example, increases in store size provide increases in profits as would be expected, however there is a dramatic increase when you jump from 4000 square feet to 7000 square feet in revenue. However, after 7000, the revenue seems to remain rather stable. This would likely be fine, however there is a stark jump in the number of employees required when going from 7000 to 8000 square feet of area. This represents a huge increase in costs, which would in turn result in diminishing true profits. This is something we see mirrored in the average employee count vs profit chart as well, where-in revenue hit a plateau after an average employee count of 52. This heavily indicates that the store size that provides the best return in revenue is likely around 7000 square feet, and that there is little reason to run stores above this size. Instead, it would likely be more profitable to open up new stores in different areas, potentially of a smaller size, to allow for those local communities to access it.

Correlation Heatmap Analysis of Sick Leave Data

Objective

This report aims to analyze the relationship between **Total Sick Leave Hours**, **Average Sick Leave Hours**, and **Employee Count** using a correlation heatmap. The analysis helps identify patterns and relationships that can provide insights into employee behavior and organizational trends. This is a continuation of the fourth query, in order to provide additional insights.

Dataset Overview

The data was queried from the **AdventureWorks2022** database, focusing on:

- **Job Title**
- **Total Sick Leave Hours** (Sum of sick leave hours across all employees in a job role)
- **Average Sick Leave Hours** (Average sick leave hours per employee in a job role)
- **Employee Count** (Number of employees in each job role)

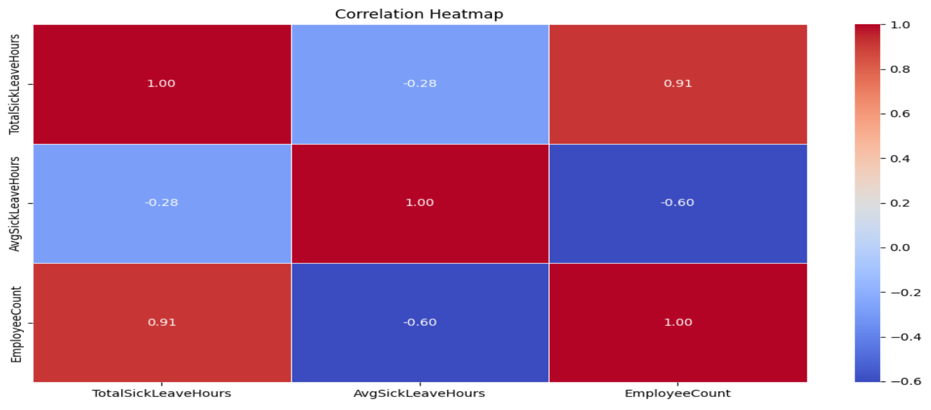
The top 20 job titles were included in the analysis, ranked by **Total Sick Leave Hours**.

Methodology

1. **Generate a Correlation Matrix** to compute the relationships between numeric variables.
2. **Create a Heatmap** using the correlation matrix to visualize these relationships.

```
# Compute the correlation matrix
correlation_matrix = dftitle[['TotalSickLeaveHours', 'AvgSickLeaveHours',
'EmployeeCount']].corr()
# Plot the correlation heatmap
plt.figure(figsize=(10, 8)) sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```

The visualisation is below.



Findings from the Correlation Heatmap

The heatmap displays the correlation coefficients between **Total Sick Leave Hours**, **Average Sick Leave Hours**, and **Employee Count**. These correlations are represented by values ranging from -1 to 1:

- **1**: Perfect positive correlation
- **-1**: Perfect negative correlation
- **0**: No correlation

Here's what the data reveals:

1. **Strong Positive Correlation Between Total Sick Leave Hours and Employee Count (0.91)**:
   - This indicates that as the number of employees increases, the total sick leave hours also increase. This trend is expected as larger teams contribute more cumulative sick hours.
2. **Moderate Negative Correlation Between Average Sick Leave Hours and Employee Count (-0.60)**:
   - This suggests that as the employee count increases, the average sick leave per employee tends to decrease. It could mean that in larger teams, workloads are more evenly distributed, reducing the likelihood of individual employees taking extended sick leave.
3. **Weak Negative Correlation Between Total Sick Leave Hours and Average Sick Leave Hours (-0.28)**:
   - There is a slight inverse relationship, indicating that higher total sick leave does not necessarily mean higher average sick leave hours per employee.

Insights and Recommendations

1. **Team Size and Efficiency**:
    - Teams with a higher number of employees show lower average sick leave per employee. This suggests that larger teams may be operating efficiently, with a more balanced workload.
2. **High Sick Leave Teams**:
    - Departments or roles with high **Total Sick Leave Hours** should be reviewed for potential issues such as workload imbalances, work environment concerns, or employee well-being programs.
3. **Actionable Steps**:
    - Focus on high-sick-leave departments to investigate causes and implement wellness programs.
    - Monitor trends in average sick leave hours as a potential efficiency indicator for teams.

Conclusion

This analysis provides actionable insights into how employee count, total sick leave hours, and average sick leave hours are interrelated. These findings can help improve workforce management and optimize team performance.

Total Revenue by Item Type per Country

Objective

The purpose of this analysis was to provide further insight into the revenue breakdown for each item type by country, in order to determine if certain countries had greater interest in one item type over another. This thus takes the form of multiple queries being ran in order to analyse the data.

Methodology

Step 1: Importing Libraries and Database Connection

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

# Step 1: Establish Connection to the Database
connection = pyodbc.connect(
        r'DRIVER={ODBC Driver 17 for SQL Server};'
        r'SERVER=DESKTOP-G1HPGDT\SQLEXPRESS;'
        r'DATABASE=AdventureWorks2022;'
        r'Trusted_Connection=yes;'
)
```

For this query we will be making use of pyodbc for the database connection, pandas to read and execute the SQL query, and pyplot from the matplotlib library in order to plot the charts themselves.

Step 2: First Query Execution - Accessories

Our first query will use a series of joins to select the relevant data, and then filter this selected data to limit it to just displaying the accessories.

```
# Step 2: Define the SQL Query to get Accessories by country and revenue
query = """
SELECT
        a.CountryRegionName AS Country,
        pc.Name AS Category,
        SUM(soh.TotalDue) AS Revenue
FROM
        Sales.SalesOrderHeader AS soh
JOIN
        Sales.SalesOrderDetail AS sod
        ON soh.SalesOrderID = sod.SalesOrderID
JOIN
```

```
        Production.Product AS p
        ON sod.ProductID = p.ProductID
JOIN
        Production.ProductSubcategory AS psc
        ON p.ProductSubcategoryID = psc.ProductSubcategoryID
JOIN
        Production.ProductCategory AS pc
        ON psc.ProductCategoryID = pc.ProductCategoryID
JOIN
        Sales.vStoreWithAddresses AS a
        ON soh.ShipToAddressID = a.BusinessEntityID  -- Corrected Join Condition
WHERE
        pc.Name = 'Accessories'
GROUP BY
        a.CountryRegionName, pc.Name
ORDER BY
        Revenue ASC;"""

# Execute the Query and Load Data into DataFrame
df = pd.read_sql(query, connection)

# Display the DataFrame
print(df)
```

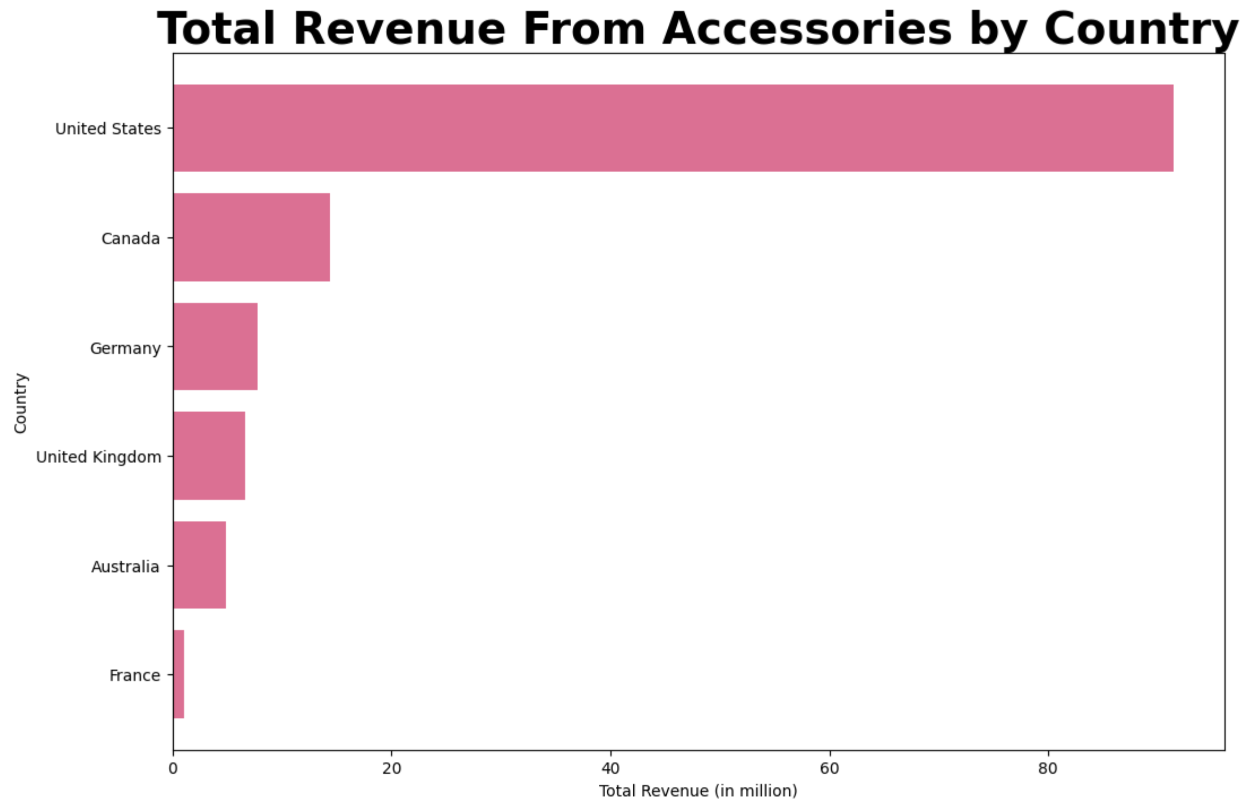
Through this series of joins we are able to select the data we need from SQL, which we will then be importing into a dataframe later. With this done, we then make the dataframe, and print it to display the data we have, to ensure that it is all correct.

Step 3: First Visualisation:

From here, we will plot our data, utilising the accessories dataframe we created above.

```
# Plot the horizontal bar chart
plt.figure(figsize=(12, 8))
plt.barh(df['Country'], df['Revenue']/1000000, color='palevioletred')

plt.xlabel('Total Revenue (in million)')
plt.ylabel('Country')
plt.title('Total Revenue From Accessories by Country', fontweight='bold', fontsize='28')
plt.show()
```

We display the revenue divided by one million in order to better represent the numbers, as otherwise it would come out in e1X notation. The resulting plot is as follows:



Step 3: Second Query Execution - Bikes

Our second query will use a series of joins to select the relevant data, and then filter this selected data to limit it to just displaying the bike sales. Finally, we update the dataframe variable in order to store this new query in a dataframe.

```
query = """
SELECT
        a.CountryRegionName AS Country,
        pc.Name AS Category,
        SUM(soh.TotalDue) AS Revenue
FROM
        Sales.SalesOrderHeader AS soh
JOIN
        Sales.SalesOrderDetail AS sod
        ON soh.SalesOrderID = sod.SalesOrderID
JOIN
        Production.Product AS p
        ON sod.ProductID = p.ProductID
JOIN
        Production.ProductSubcategory AS psc
```

```
    ON p.ProductSubcategoryID = psc.ProductSubcategoryID
JOIN
    Production.ProductCategory AS pc
    ON psc.ProductCategoryID = pc.ProductCategoryID
JOIN
    Sales.vStoreWithAddresses AS a
    ON soh.ShipToAddressID = a.BusinessEntityID  -- Corrected Join Condition
WHERE
    pc.Name = 'Bikes'
GROUP BY
    a.CountryRegionName, pc.Name
ORDER BY
    Revenue ASC;
"""
# Execute the Query and Load Data into DataFrame
df = pd.read_sql(query, connection)

# Display the DataFrame
print(df)
```

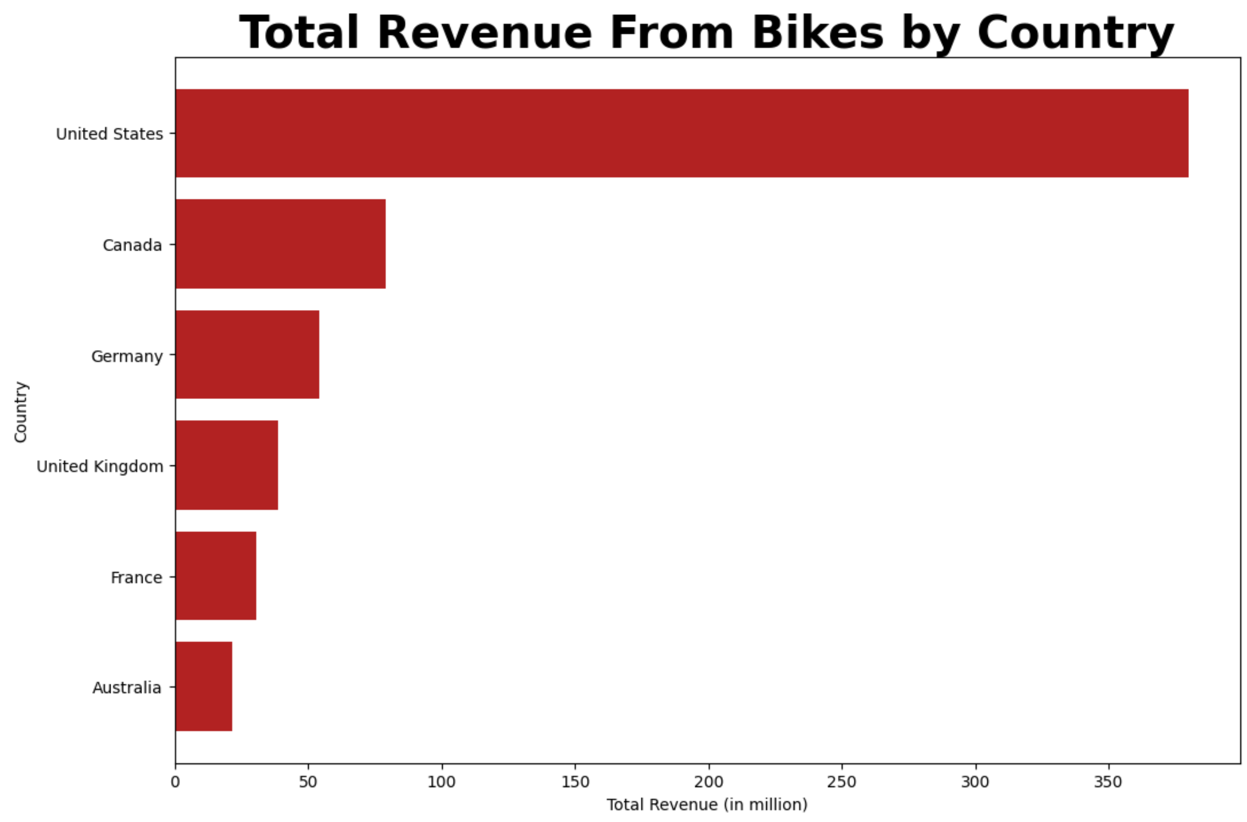
Once again we use a complex series of joins to select the relevant data, and then filter it to just be the data for bikes.

Step 4: Second Visualisation:

From here, we will plot our data, utilising the bikes dataframe we created above.

```
# Plot the horizontal bar chart
plt.figure(figsize=(12, 8))
plt.barh(df['Country'], df['Revenue']/1000000, color='firebrick')
plt.xlabel('Total Revenue (in million)')
plt.ylabel('Country')
plt.title('Total Revenue From Bikes by Country', fontweight='bold', fontsize='28')
plt.show()
```

Here we change the chart colour to be red in order to better contrast against our previous chart, but otherwise it is much of the same process. The chart is below.



Step 5: Third Query Execution - Clothing

Once again, we're going to use the same series of joins in order to select the relevant data, and then compile it into a dataframe. In this case, we're going to be limiting the data to just Clothing.

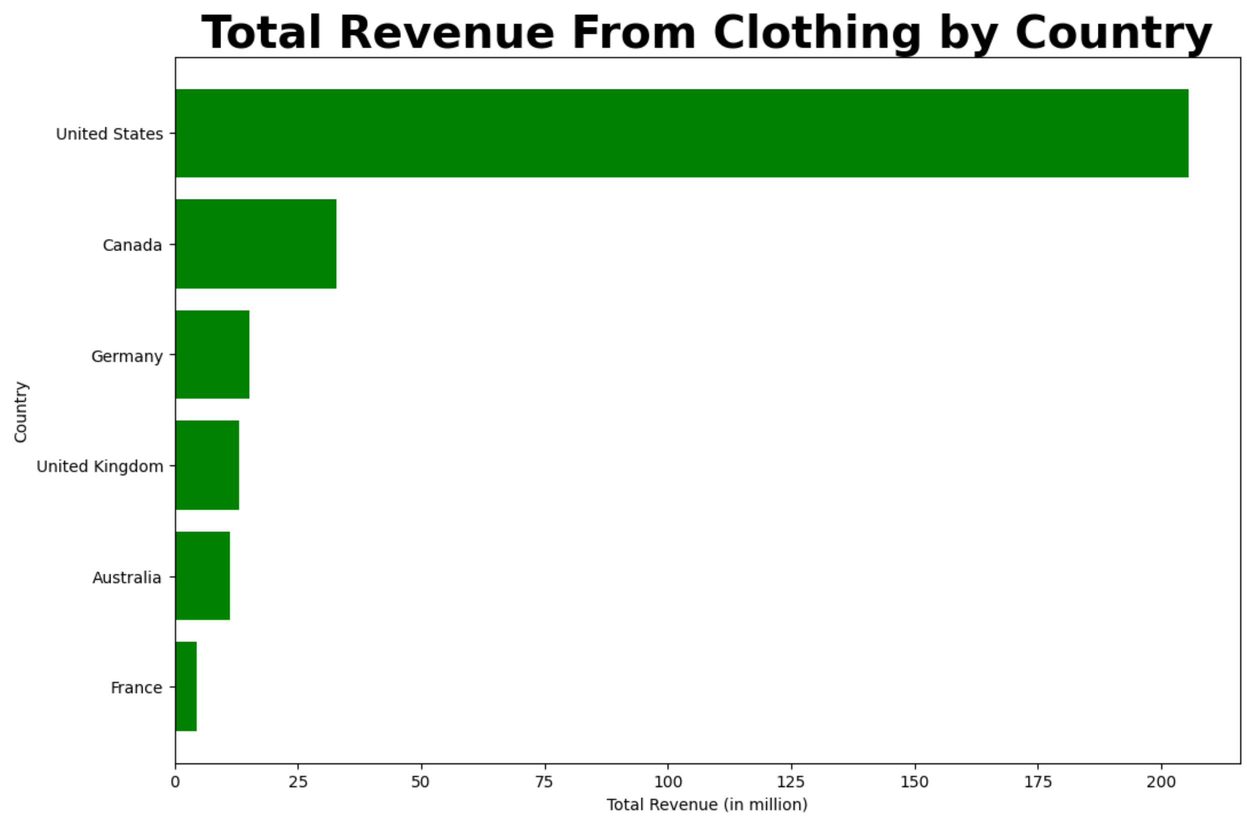
```
query = """
SELECT
        a.CountryRegionName AS Country,
        pc.Name AS Category,
        SUM(soh.TotalDue) AS Revenue
FROM
        Sales.SalesOrderHeader AS soh
JOIN
        Sales.SalesOrderDetail AS sod
        ON soh.SalesOrderID = sod.SalesOrderID
JOIN
        Production.Product AS p
        ON sod.ProductID = p.ProductID
JOIN
        Production.ProductSubcategory AS psc
        ON p.ProductSubcategoryID = psc.ProductSubcategoryID
JOIN
```

```
        Production.ProductCategory AS pc
        ON psc.ProductCategoryID = pc.ProductCategoryID
JOIN
        Sales.vStoreWithAddresses AS a
        ON soh.ShipToAddressID = a.BusinessEntityID  -- Corrected Join Condition
WHERE
        pc.Name = 'Clothing'
GROUP BY
        a.CountryRegionName, pc.Name
ORDER BY
        Revenue ASC;"""

# Execute the Query and Load Data into DataFrame
df = pd.read_sql(query, connection)

# Display the DataFrame
print(df)
```

Step 6: Third Visualisation:

Once again we're going to visualise our data, and the resulting graph will be below. In this case we change the colour to green, to once again provide better contrast to the other two charts already made.

```
plt.figure(figsize=(12, 8))
plt.barh(df['Country'], df['Revenue']/1000000, color='green')
plt.xlabel('Total Revenue (in million)')
plt.ylabel('Country')
plt.title('Total Revenue From Clothing by Country', fontweight='bold', fontsize='28')
plt.show()
```



Step 7: Fourth Query Execution - Clothing

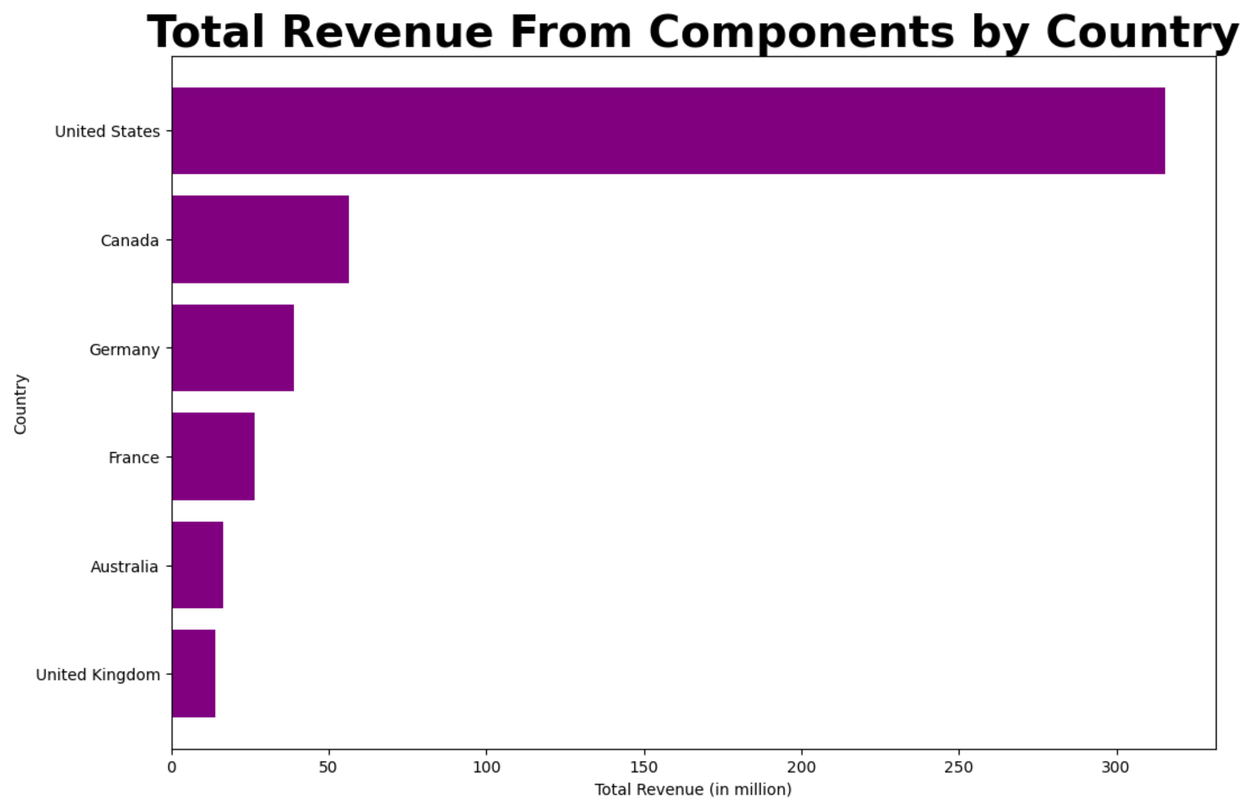
For one final time, we’re going to utilise this same series of joins, and instead filter it to select from components. With all our queries done, we can now also close the connection - this is more efficient than repeatedly opening and closing it, but it is important to close it for safety purposes.

```
query = """
SELECT
        a.CountryRegionName AS Country,
        pc.Name AS Category,
        SUM(soh.TotalDue) AS Revenue
FROM
        Sales.SalesOrderHeader AS soh
JOIN
        Sales.SalesOrderDetail AS sod
        ON soh.SalesOrderID = sod.SalesOrderID
JOIN
        Production.Product AS p
        ON sod.ProductID = p.ProductID
JOIN
        Production.ProductSubcategory AS psc
        ON p.ProductSubcategoryID = psc.ProductSubcategoryID
JOIN
        Production.ProductCategory AS pc
        ON psc.ProductCategoryID = pc.ProductCategoryID
JOIN
        Sales.vStoreWithAddresses AS a
        ON soh.ShipToAddressID = a.BusinessEntityID  -- Corrected Join Condition
WHERE
        pc.Name = 'Components'
GROUP BY
        a.CountryRegionName, pc.Name
ORDER BY
        Revenue ASC;"""

# Execute the Query and Load Data into DataFrame
df = pd.read_sql(query, connection)

#Close the connection
connection.close()

# Display the DataFrame
print(df)
```

Step 8: Fourth Visualisation:

For one final time, we will be visualising our data. In this case, we make the chart purple, in order to better contrast the output to the other charts.

```
# Plot the horizontal bar chart
plt.figure(figsize=(12, 8))
plt.barh(df['Country'], df['Revenue']/1000000, color='purple')
plt.xlabel('Total Revenue (in million)')
plt.ylabel('Country')
plt.title('Total Revenue From Components by Country', fontweight='bold', fontsize='28')
plt.show()
```



Results

- All four SQL queries were successfully executed, and by executing them in the same file we avoid the need to import the libraries or establish the connection multiple times.
- The charts clearly demonstrate the differences in each item type, including which items are more popular in which regions.

Conclusion

As can clearly be seen here, the US is consistently the lead on all sales types. However, the United Kingdom has a marked drop in interest for components compared to other regions, whilst France has a noticeable increase in this area. Furthermore, Germany is consistently third in terms of annual revenue, which is a striking difference to the amount of total revenue that it has contributed to - this could indicate that stores in Germany were only recently opened, but are proving to be incredibly profitable despite this recency of opening.

Bibliography:

In the course of our investigation, we have utilised the following open source python libraries:

- pyodbc - https://github.com/mkleehammer/pyodbc/wiki
- pandas - https://pandas.pydata.org/
- matplotlib, and specifically the pyplot functions, imported from it - https://matplotlib.org/
- numpy - https://numpy.org/

Furthermore, the coding has been performed using:

- SQL Server Management Studio 20
- Jupyter via Visual Studio Code

The Database used in these queries has been the OLTP AdventureWorks2022, an open source database curated and maintained by Microsoft. Various tables and views have been used from this database, primarily ones from the Sales and Production schemas, though a table from the Product schema and the Human Resources schema have also been used. The tables used were as follows:
From the Production Schema:

- Product
- ProductSubcategory

From the Sales Schema:

- SalesOrderHeader
- SalesOrderDetails
- CustomerSales
- SalesTerritoryID
- SalesPerson

From other schemas:

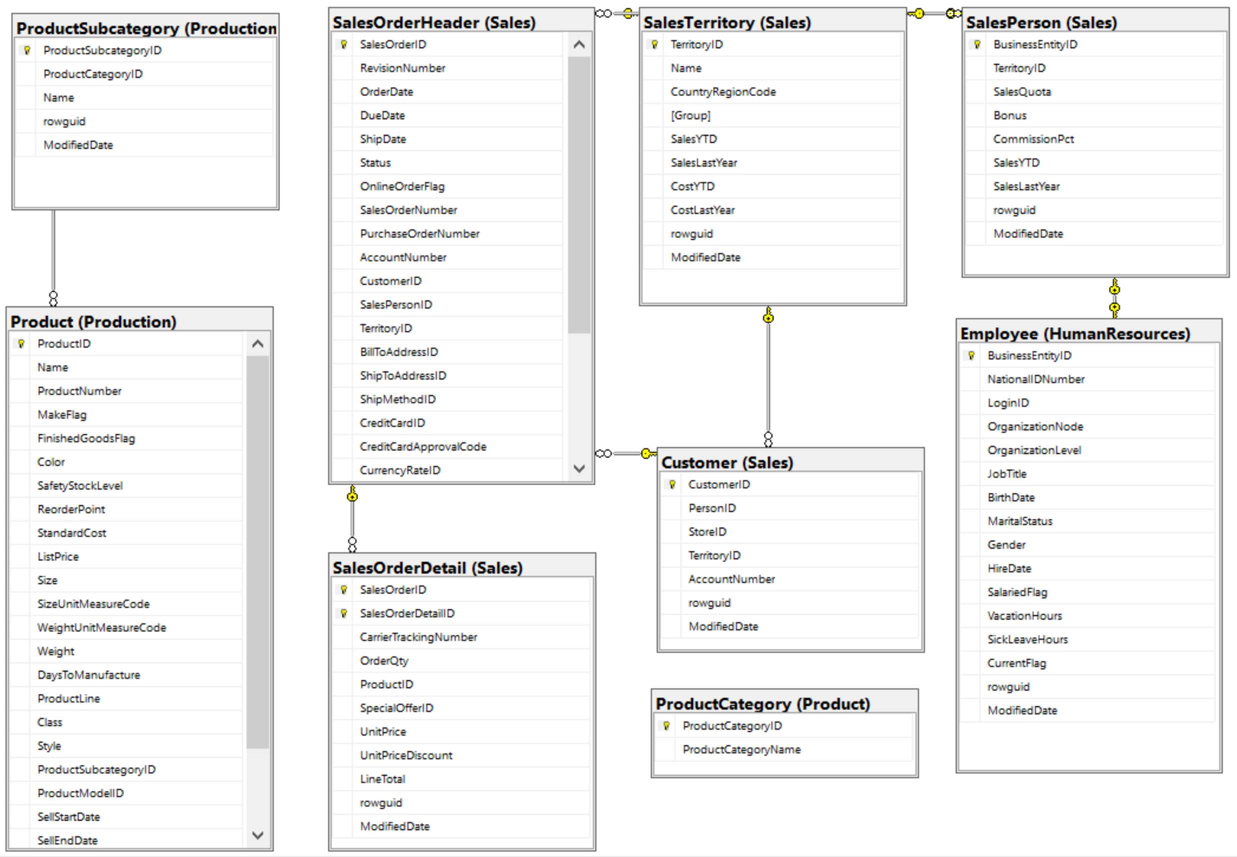
- Employee (HumanResources Schema)
- ProductCategory(Product Schema)

The Sales schemas are linked via SalesOrderID, TerritoryID, BusinessEntityID, and CustomerID.

The employee table is linked to SalesPerson from the sales schema by BusinessEntityID.

The two Production schema tables are linked by ProdutSubcategoryID.

Finally, the ProductCategory table isn't linked to any of the other tables that we used here. A visualisation of the database tables we used is below.



The specific files for the source code can be provided upon request from DA-NAT4 - Group 2, or by contacting the individuals mentioned at the top of this report.

The division of work was as follows:

Q1-2: Pearl, Elsheikh
Q3-4: Dawit, Hamza
Q5-6: Abdulrahman, Alice

If there are any further questions regarding anything on this project, please feel free to contact Dawit, as He was the lead on this project.

Thank you for your time.