Jamie Colclough

Matrix Manufacturing and Retail – Sales Analysis

The Task

I have been requested to uncover a variety of insights about the sales of a bicycle company. Specifically, I need to analyse how bike sales are performing, how these sales differ by product, analyse the sales patterns of customers and use this information to recommend actions and communicate issues. Initially, I think some of the questions my project should answer include; what are the most/least popular products?, what products generate the most/least revenue?, which employees have generated the most/least revenue?, which customer have spent the most/least?, which customers have bought most/least frequently?, which period of time are the most lucrative?. I may uncover more questions and answers as the project unfolds, but these are the types of questions that should be the goal of my analysis.

In addition, I need to bear in mind the target audience of this project are the senior executives of the business, who will want advice on the actions that should be taken based on this data. This leads us to more insightful questions such as; which products should we produce more/less of?, what period of time should we have more/less stock?, which customers should we target our advertisements to?, which products are too cheap/expensive?. I believe success within this project will include an analysis that answers all of these questions and more, while using clear visualisations that are easy to understand to communicate the findings. This can then be used to support actionable recommendations supported by evidence.

The Plan

The deadline for this project is the 25th November, giving me 2 weeks to complete the analysis. I will structure my time in the following ways, while also allowing flexibility to explore interesting avenues.

- Day 1-2: Import and explore the dataset, looking for its scope and limitations. I will also clean the data, dealing with duplicates, missing values and any inconsistent formatting.
- Day 3-5: Begin the foundational data analysis, generating summary statistics, and creating visualizations for sales, products, customers, employees, and time trends. Some initial trends will be identified.
- Day 6-9: Look into some deeper analysis, answering the key questions. Explore more complicated queries and try to uncover trends in the revenue, whilst creating easy to understand visuals.
- Day 10: Translate the findings from the data into actionable insights, identifying areas for growth and where to focus marketing.
- Day 11-13: Prepare the report, using suitable visuals created during the analysis stage and clearly explaining the findings and recommendations. Leave time for proof reading and double checking the data and charts.

Customer Engagement

Here I will keep a list of questions I would ask the customer about the data, and, when necessary, the methods I have used when dealing with these inconsistencies/anomalies in the data, after meeting with the customer and discussing the problems.

Questions	Action Taken
Should the data only include a timespan of a single year?	Concluded that the data is representative.
Why does the customer data only include surnames beginning with Y? Do we have the full dataset?	Concluded that this is merely is an error in the surnames column, but all customers are represented in the data. When analysing customer loyalty, we can use first and, when

	possible, middle initials to identify them. The surnames column can also be repaired once the data supplier has fixed the problem on their end, and then can be applied to analysis I have found.
Are just the product names beginning with ‘Mountain-‘,’Touring-‘ or ‘Road-‘ the bike category products?	Yes, these are the bike products, but there are also bike frames, which are the most expensive part of a bike purchase when buying parts individually. I have chosen to include frames when comparing/analysing popularity of each type of bike. I then grouped all mountain, touring and road parts together for further analysis.
Should there be products listed as 0.00 for their price?	This is indeed an error so where appropriate, I’ve replaced the values with an average of the price for products with a similar name e.g. Flat Washer products with prices of 0.00 were replaced with the average from the products that were given prices. I will note down this approximation and label the products that have been dealt with in this way. This is a reasonable approach until the correct prices have been provided.
Are all the prices listed correct? E.g. there are a broad range of prices listed for different colours of paint which seems like it would be an error.	Until we can get confirmation on all the prices, we have to assume they are all correct. If any suspicious/alarming results come out of the analysis, I will note down that this may be due to pricing errors

Data Preparation

The data provided included four tables, which were Customers, Employees, Products and Sales. After some initial exploration of the data, I found that the Sales table linked the database together with three foreign keys contained within it. I used SQLiteStudio to query the data in order to get tables that could then be saved and exported to Python, where more in depth analysis was applied as well as creating the visualisations. I used JupyterLab for this analysis.

Once the data was loaded into SQL, the first process I applied was updating the prices of products that were given £0.00, which I interpreted as a mistake. To fix this I looked through the data and decided to price them based on the prices of similarly named products, specifically the mean of those products. Some products with zeroes were left unchanged, as I didn’t have enough information to give an estimate of the price.

UPDATING PRICES:

```
UPDATE tblProducts
SET Price = (SELECT AVG(Price) FROM tblProducts
WHERE Name LIKE 'Flat Washer%'
AND Price != 0)
WHERE Name LIKE 'Flat Washer%' AND Price = 0;
UPDATE tblProducts
SET Price = (SELECT AVG(Price) FROM tblProducts
WHERE Name LIKE '%End Caps'
AND Price != 0)
WHERE Name LIKE '%End Caps' AND Price = 0;
UPDATE tblProducts
SET Price = (SELECT AVG(Price) FROM tblProducts
WHERE Name LIKE 'External Lock Washer%'
AND Price != 0)
WHERE Name LIKE 'External Lock Washer%' AND Price =
0;

UPDATE tblProducts
```

```
UPDATE tblProducts
SET Price = (SELECT AVG(Price) FROM tblProducts
WHERE Name LIKE '%Grip Tape'
AND Price != 0)
WHERE Name LIKE '%Grip Tape' AND Price = 0;
UPDATE tblProducts
SET Price = (SELECT AVG(Price) FROM tblProducts
WHERE Name LIKE 'Hex Nut%'
AND Price != 0)
WHERE Name LIKE 'Hex Nut%' AND Price = 0;
UPDATE tblProducts
SET Price = (SELECT AVG(Price) FROM tblProducts
WHERE Name LIKE 'Internal Lock Washer%'
AND Price != 0)
WHERE Name LIKE 'Internal Lock Washer%' AND Price =
0;

UPDATE tblProducts
```

```
SET Price = (SELECT AVG(Price) FROM tblProducts
WHERE Name LIKE 'Lock Nut%'
AND Price != 0)
WHERE Name LIKE 'Lock Nut%' AND Price = 0;
UPDATE tblProducts
SET Price = (SELECT AVG(Price) FROM tblProducts
WHERE Name LIKE 'Metal Plate%'
AND Price != 0)
WHERE Name LIKE 'Metal Plate%' AND Price = 0;
```

```
SET Price = (SELECT AVG(Price) FROM tblProducts
WHERE Name LIKE 'Lock Washer%'
AND Price != 0)
WHERE Name LIKE 'Lock Washer %' AND Price = 0;
UPDATE tblProducts
SET Price = (SELECT AVG(Price) FROM tblProducts
WHERE Name LIKE '%Road Rim'
AND Price != 0)
WHERE Name LIKE '%Road Rim' AND Price = 0;
```

After having applied these updates, I later realised that I would be helpful to know which products had been approximated, and which hadn't. In hindsight, this could have been accomplished in the previous step, but in this scenario I had to write some more code in order to work out which rows had been altered. I did this by creating a new column and filling it with 'True' where the price was equal to the mean of the rest of that category. This is shown in the code below.

<u>LABELLING UPDATED ROWS</u>:

```
ALTER TABLE tblProducts
ADD PriceChanged VARCHAR(5);WHERE Name LIKE 'Flat
Washer%' AND Price = 0;


UPDATE tblProducts
SET PriceChanged = 'true'
WHERE Price = (SELECT ROUND(AVG(Price),2)
FROM tblProducts
WHERE Name LIKE '%Grip Tape');WHERE Name LIKE
'%End Caps' AND Price = 0;
UPDATE tblProducts
SET PriceChanged = 'true'
WHERE Price = (SELECT ROUND(AVG(Price),2)
FROM tblProducts
WHERE Name LIKE '%End Caps');
UPDATE tblProducts
SET PriceChanged = 'true'
WHERE Price = (SELECT ROUND(AVG(Price),2)
FROM tblProducts
WHERE Name LIKE 'External Lock Washer%');
UPDATE tblProducts
SET PriceChanged = 'true'
WHERE Price = (SELECT AVG(Price)
FROM tblProducts
WHERE Name LIKE 'Lock Nut%');
UPDATE tblProducts
SET PriceChanged = 'true'
WHERE Price = (SELECT ROUND(AVG(Price),2)
FROM tblProducts
WHERE Name LIKE 'Metal Plate%');
```

```
UPDATE tblProducts
SET PriceChanged = 'true'
WHERE Price = (SELECT ROUND(AVG(Price),2)
FROM tblProducts
WHERE Name LIKE 'Flat Washer%');
UPDATE tblProducts
SET PriceChanged = null
WHERE Name = 'LL Grip Tape'; /* realised there was only
one grip with a price originally, so even the product with a
price got updated to true. Corrected this.*/

UPDATE tblProducts
SET PriceChanged = 'true'
WHERE Price = (SELECT AVG(Price)
FROM tblProducts
WHERE Name LIKE 'Hex Nut%');
UPDATE tblProducts
SET PriceChanged = 'true'
WHERE Price = (SELECT ROUND(AVG(Price),2)
FROM tblProducts
WHERE Name LIKE 'Internal Lock Washer%');
UPDATE tblProducts
SET PriceChanged = 'true'
WHERE Price = (SELECT AVG(Price)
FROM tblProducts
WHERE Name LIKE 'Lock Washer %');
UPDATE tblProducts
SET PriceChanged = 'true'
WHERE Price = (SELECT ROUND(AVG(Price),2)
FROM tblProducts
WHERE Name LIKE '%Road Rim') AND Name != 'LL Road
Rim';
```

After this process I believed the data was ready for querying with SELECT statements. Further cleaning processes, such as changing data types or renaming columns, were completed within JupyterLab or when the tables were saved in Excel. This will be outlined in subsequent sections.

<u>1 – Employee Analysis</u>

Firstly I began to analyse the performances of each employee, by looking at metrics like sales and revenue, before looking into monthly contributions. To extract this data I used the following SQL query:

```
SELECT e.FirstName || ' ' || e.LastName AS 'Employee Name',SUM(s.Quantity) AS 'Sales',SUM(s.Quantity*p.Price) AS
'Revenue' FROM tblSales s
JOIN tblEmployees e ON s.SalesPersonID = e.EmployeeID JOIN tblProducts p ON p.ProductID = s.ProductID
```

```
GROUP BY s.SalesPersonID;
```

I combined first and last names into one column, summed total sales throughout the year, and multiped these sales by the prices of the products sold and summed to generate a total revenue. I then loaded the data and began to visualise some simple metrics using Python.

```
#loading sales data grouped by employee
revbyemp = pd.read_excel("Bicycle Shop Data.xlsx", sheet_name="Revenue By Employee")
revbyemp = revbyemp.sort_values('Revenue') #ordering by revenue
revbyemp.head()
#creating a visual that displays both the sales and revenue by each employee

x = revbyemp['Revenue']
x2 = revbyemp['Sales']
width = 0.4 #separation for the two columns
labels = revbyemp['Employee']
pos = np.arange(len(labels)) #positions of each employer on the y axis
fig, ax1 = plt.subplots(figsize=(12, 5))

ax1.barh(pos-width/2,x,width,color='#1f77b4')
ax1.set_xlabel("Total Revenue / £")
ax1.set_yticks(pos)
ax1.set_yticklabels(labels)

ax2 = ax1.twiny()
ax2.barh(pos+width/2,x2,width,color="#ff7f0e")
ax2.set_xlabel("Number of Sales")

#setting the color of both the axes to correspond to the relevant bars
ax1.spines["bottom"].set_color('#1f77b4')
ax1.tick_params(axis='x', colors='#1f77b4')
ax2.spines["top"].set_color("#ff7f0e")
ax2.tick_params(axis='x', colors='#ff7f0e')

plt.tight_layout()
```

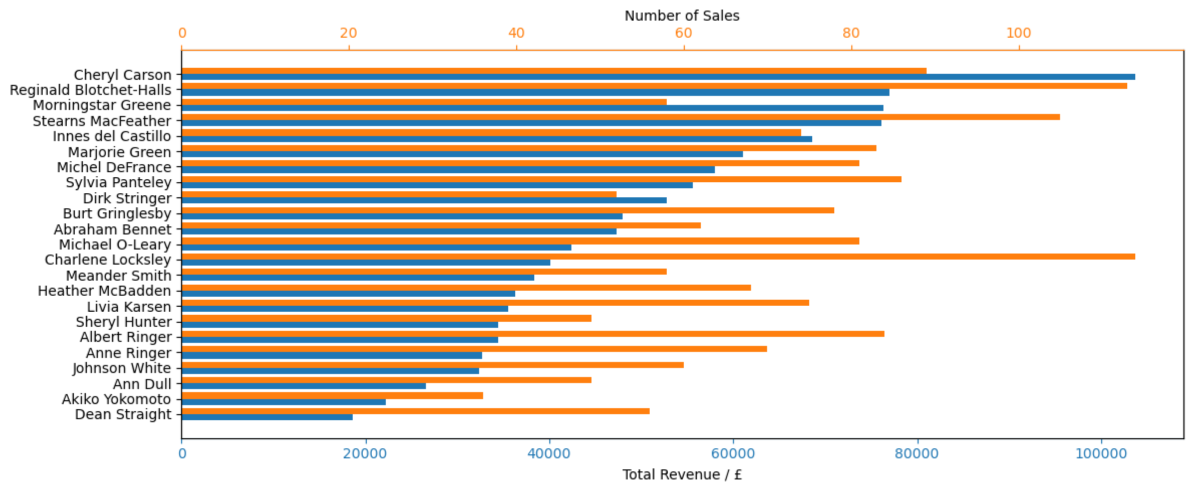


Figure 1: Bar Graph of Employee Sales and Revenue

Figure 1 give us an overview of the year in terms of how each employee performed overall in terms of sales and revenue. We can see that employees like Cheryl Carson or Reginald Blotchet-Halls are performing very well in this metric and may be eligible for bonuses, commissions, or recognition. Cheryl, in particular brought in a revenue that was 113% greater than the mean. These employees should be prioritised for future leadership or training roles. On the other hand, at the bottom of the graph, we have some underperformers. While Dean Straight made a decent amount of sales, just 23% shy of the mean, the revenue was the lowest overall, suggesting a high quantity of small value sales. Employees in bottom region of the graph may need support, training or possibly a role reassessment. There is a general correlation between sales and revenue, since the bars tend to fall together as we read down the chart, but it isn't a perfect relationship. For example, Charlene Locksley made the most sales out of everyone, but her total revenue lies close to the average. Some employees may take responsibility for smaller sales, so it's important to value both sales and revenue when it comes to evaluating employee performances.

However, this analysis fails to consider the amount of hours worked by each employee. Unfortunately we don't have access to this data currently, so this is something that could be addressed by a future analysis. One factor that could be considered was the effect of different months having a higher volume of sales and more revenue than others, meaning that employees who worked more during those months would have completed a higher number of sales. To start exploring this avenue I first investigated the sales data over time, extracting the information using the following SQL query:

```
SELECT s.SalesDate,s.Quantity,SUM(s.Quantity*p.Price) FROM tblSales s
JOIN tblProducts p ON s.ProductID = p.ProductID
GROUP BY s.SalesDate
ORDER BY s.SalesDate;
```

In Python, I then created a new 'Month' column in this table and created a simple line graph of the revenue over time.

```
#Loading sales grouped by date
revbydatedf = pd.read_excel("Bicycle Shop Data.xlsx", sheet_name="Revenue By Date")
#creating 'month' column
revbydatedf["Month"] = revbydatedf["Sales Date"].dt.to_period("M").astype(str)
revbydatedf.head()
#finding total sales and revenue each month
revbymonth = revbydatedf[['Month','Sales','Revenue']].groupby('Month').sum()
revbymonth
#displaying revenue by month as line graph
months=['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
plt.plot(months,revbymonth['Revenue'],color="#1f77b4")
plt.xlabel('Month')
plt.ylabel('Total Revenue / £')
```

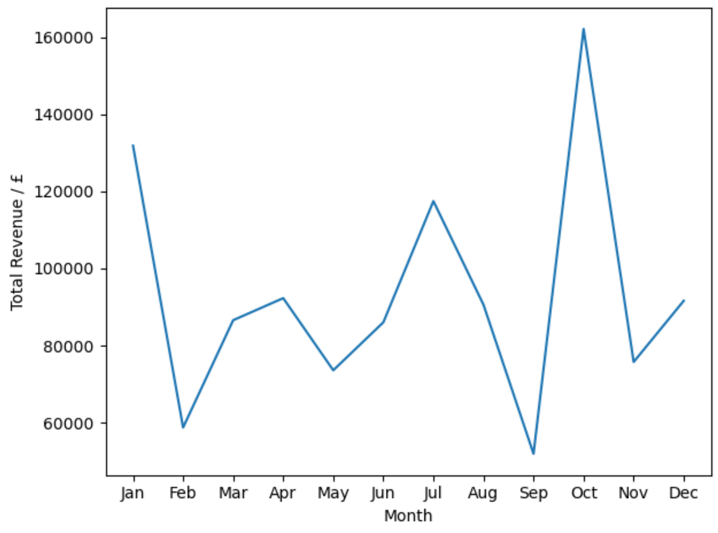


Figure 2: Line graph showing the total revenue each month

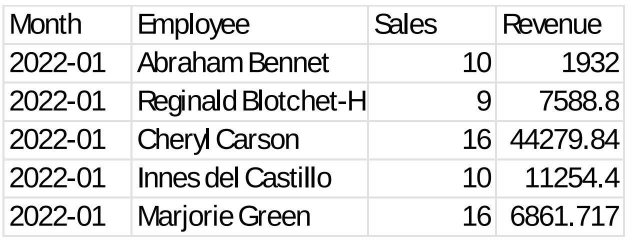
This graph shows there is quite a substantial different in revenues across months, with it perhaps surprisingly peaking in October. There is a 212% increase in revenue from September to October, which are the minimum and maximum values in the dataset, and shows that the months in which sales take place are a significant factor.

To consider this in our analysis, however, we need to go further and create a metric that evaluates the customers contributions each month. My solution was to work out each employees fractional contribution to the total each month, and then take the mean of these values to find out how consistently the employee makes significant contributions. This also had the added bonus of not including months where the employee made no sales, so therefore taking the assumption that they didn’t work during this period. This may be incorrect in some cases, but until we are provided the working hours data, this seems like a reasonable assumption.

Another SQL query was required so that we could group the employees sales data by both employee and month. This required creating a ‘month’ column in our query and grouping by this new column.

```
SELECT strftime('%Y-%m', s.SalesDate) AS Month,e.FirstName || ' ' || e.LastName AS 'Employee Name',SUM(s.Quantity)
AS 'Sales',SUM(s.Quantity*p.Price) AS 'Revenue' FROM tblSales s
JOIN tblEmployees e ON s.SalesPersonID = e.EmployeeID JOIN tblProducts p ON p.ProductID = s.ProductID
GROUP BY Month,s.SalesPersonID;
```

This data was then exported to Excel, resulting in a table where each employee’s sales and revenue are shown each month. This is shown in Figure 3 below:



Month	Employee	Sales	Revenue
2022-01	Abraham Bennet	10	1932
2022-01	Reginald Blotchet-H	9	7588.8
2022-01	Cheryl Carson	16	44279.84
2022-01	Innes del Castillo	10	11254.4
2022-01	Marjorie Green	16	6861.717

Figure 3: Snapshot of a table containing sales data grouped by employee and month

This was then analysed further in Python by using the previously acquired data of total revenue each month. The totals were mapped to each row of the Pandas DataFrame, such that the fractional contribution could then be found row by row, and the data was ordered by this contribution within each month. Then each employee’s mean fractional contribution was calculated.

```
# loading sales grouped by both month and employee
df = pd.read_excel("Bicycle Shop Data.xlsx", sheet_name="Revenue By Month and Employee")
df.head()
#Finding the fractional contribution of each employee each month
mapping = revbymonth["Revenue"].to_dict() #creating dictionary that has total revenue for each month
Monthlyrev = df['Month'].map(mapping) #mapping each row from the df to the corresponding months revenue
df['Fractional Contribution'] = df['Revenue']/Monthlyrev
df = df.sort_values(["Month", "Fractional Contribution"], ascending=[True, False])
df.head()
#finding the mean of each employees monthly fractional contribution
meancont = df.groupby('Employee')['Fractional Contribution'].mean().sort_values()
labels = meancont.index.tolist()
meancont
fig, ax1 = plt.subplots(figsize=(12, 5))

ax1.barh(labels,meancont,color='#1f77b4')
ax1.set_xlabel("Mean Monthly Contribution")
plt.title('Mean Fractional Monthly Contribution of Employees')
```

This code produced the following figure:

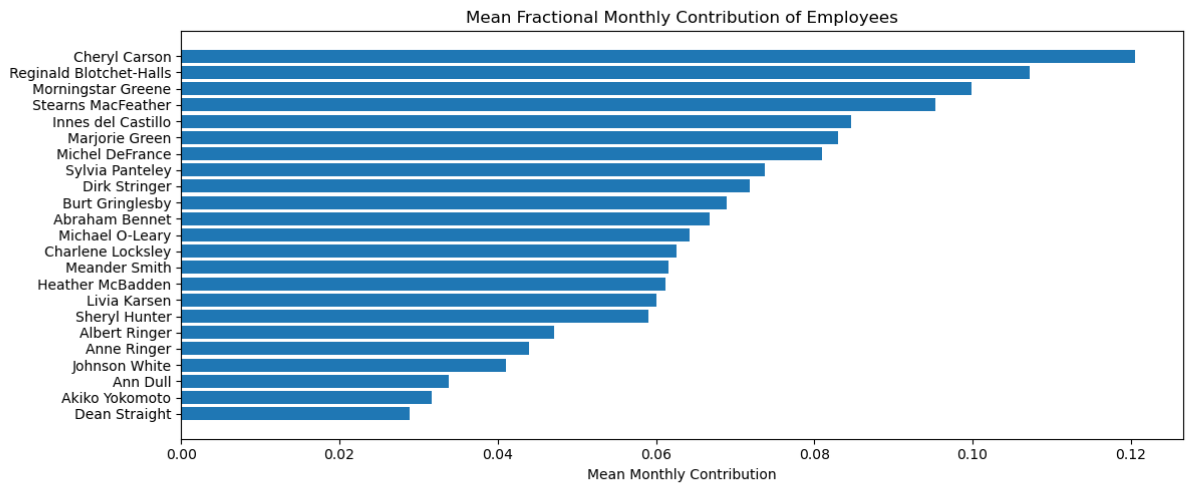


Figure 4: Bar graph ranking the employees by their mean monthly fractional contribution

This visual displays the information quite effectively, but I then decided to combine all this employee information together in one scatter graph to comprehensively show the good and bad performers.

```
mapping = meancont.to_dict() #creating dictionary that stores the mean frac_cont for each employee
revbyemp['Mean Fractional Contribution'] = revbyemp['Employee'].map(mapping) #mapping each row from the df to the
corresponding months revenue
revbyemp.head()
y = revbyemp['Revenue']
x = revbyemp['Sales']
labels = revbyemp['Employee']
c = revbyemp['Mean Fractional Contribution']

plt.figure(figsize=(10,6))
scatter = plt.scatter(x,y,c=c , cmap='inferno')
#creating colorbar that represents the mean fractional contribution
cbar = plt.colorbar(scatter, orientation='horizontal', pad=0.1,label='Mean Fractional Monthly Contribution')
# adding small labels
for xi, yi, label in zip(x, y, labels):
    if label == 'Reginald Blotchet-Halls' or label == 'Heather McBadden':
        plt.text(xi, yi+3000, label, fontsize=8, ha='left', va='top')
    else:
        plt.text(xi, yi, label, fontsize=8, ha='left', va='top')
plt.ylabel('Revenue / £')
plt.xlabel('Number of Sales')
plt.tight_layout()

plt.savefig('Employee Revenue vs Sales')
```

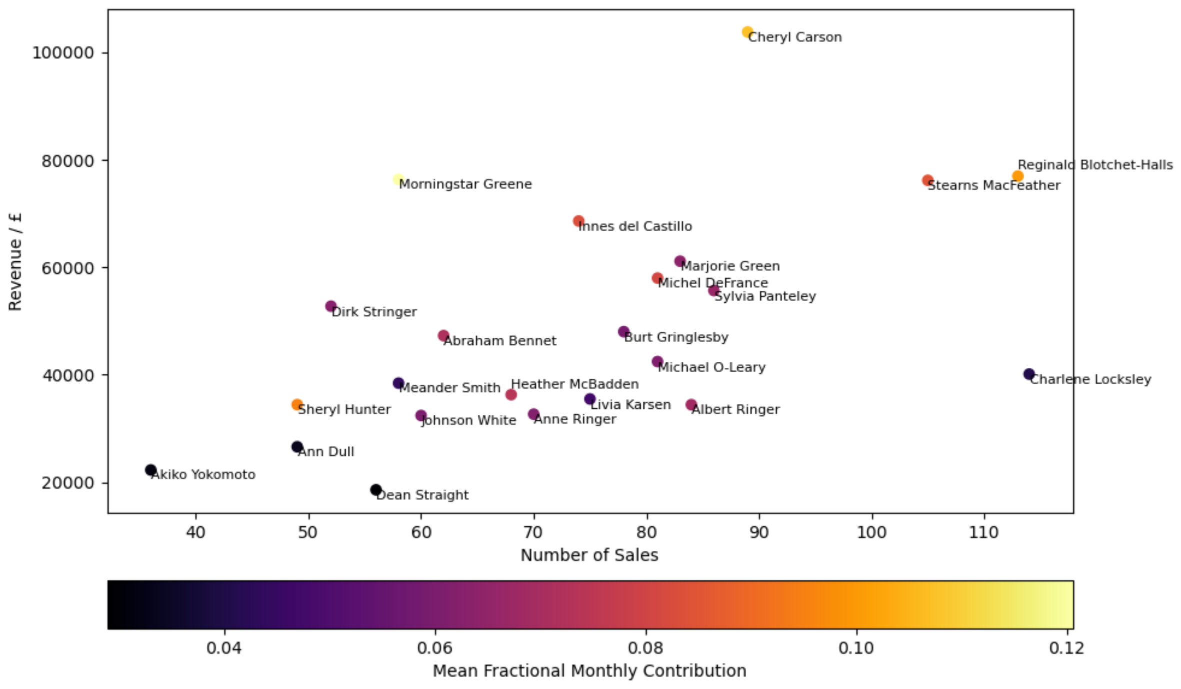


Figure 5: Employee Sales vs Revenue (Colour graded by Monthly Contributions)

Figure 5 brings together all the previous analysis, while highlighting some more potential insights. We again see that Cheryl Carson is an elite salesperson, with both high sales and revenue, and consistent monthly contributions. Additionally, we again can see that the weaker salespeople, such as Dean Straight, aren't substantially contributing each month. I can therefore repeat the strategies mentioned previously, about rewarding the strong employees and giving support and training to the weaker ones. However, this figure also shows that some employees, who may have otherwise gone unnoticed, are actually providing to a large part of their month's total earnings. For example, Sheryl Hunter is low on sales and revenue overall, but in the months that she did sell in, she was consistently contributing a significant percentage of the revenue. These people could perhaps be given more hours, in order to maximise their contributions. It is important to once again point out that a high volume of sales is just as important as a big revenue, so we shouldn't undervalue the contributions of employees like Charlene Locksley, just because the revenue is comparatively smaller than most of the staff.

2 – Product Analysis

I then began my analysis of the products, starting off by finding the products that provided the most and least revenue overall. The total sales and revenue data for each product was first extracted using the following SQL query;

```
SELECT p.Name,p.Price,SUM(s.Quantity) AS 'Units Sold',p.Price*SUM(s.Quantity) AS 'Total_Revenue',p.PriceChanged
FROM tblProducts p
JOIN tblSales s ON
p.ProductID = s.ProductID
GROUP BY Name
ORDER BY Total_Revenue DESC;
```

This data was then used to create a simple bar graph visualisation in Python, showing the best and worst products by revenue.

```
#loading the dataset of products with their total revenue and units sold
productdf = pd.read_excel("Bicycle Shop Data.xlsx", sheet_name="Products By Revenue")
#productdf = productdf.sort_values('Units_Sold',ascending=False)
```

```
productdf = productdf[productdf["Price"] != 0] #removing products where price = 0
productdf.tail()
#creating a bar chart showing the top 10 highest and lowest revenues for products
x = productdf['Units_Sold'][:10]
x2 = productdf['Total_Revenue'][:10]
labels = productdf['Name'][:10]

width = 0.4
pos = np.arange(len(labels))
fig, (ax1,ax3) = plt.subplots(1,2,figsize=(12, 5)) #creating a figure with two subplots, each subplot will have dual axes

ax1.barh(pos-width/2,x,width,color='#1f77b4')
ax1.set_xlabel("Number of Sales")
ax1.set_yticks(pos)
ax1.set_yticklabels(labels)

ax2 = ax1.twiny()
ax2.barh(pos+width/2,x2,width,color="#ff7f0e")
ax2.set_xlabel("Total Revenue / £")

#setting the color of both the axes to correspond to the relevant bars
ax1.spines["bottom"].set_color('#1f77b4')
ax1.tick_params(axis='x', colors='#1f77b4')
ax2.spines["top"].set_color("#ff7f0e")
ax2.tick_params(axis='x', colors='#ff7f0e')

x3 = productdf['Units_Sold'][-10:]
x4 = productdf['Total_Revenue'][-10:]
labels2 = productdf['Name'][-10:]

width2 = 0.4
pos2 = np.arange(len(labels2))

ax3.barh(pos2-width2/2,x3,width2,color='#1f77b4')
ax3.set_xlabel("Number of Sales")
ax3.set_yticks(pos2)
ax3.set_yticklabels(labels2)

ax4 = ax3.twiny()
ax4.barh(pos2+width2/2,x4,width2,color="#ff7f0e")
ax4.set_xlabel("Total Revenue / £")

#setting the color of both the axes to correspond to the relevant bars
ax3.spines["bottom"].set_color('#1f77b4')
ax3.tick_params(axis='x', colors='#1f77b4')
ax4.spines["top"].set_color("#ff7f0e")
ax4.tick_params(axis='x', colors='#ff7f0e')

plt.tight_layout()
plt.savefig('Top 10 Products by Highest and Lowest Revenues')
```

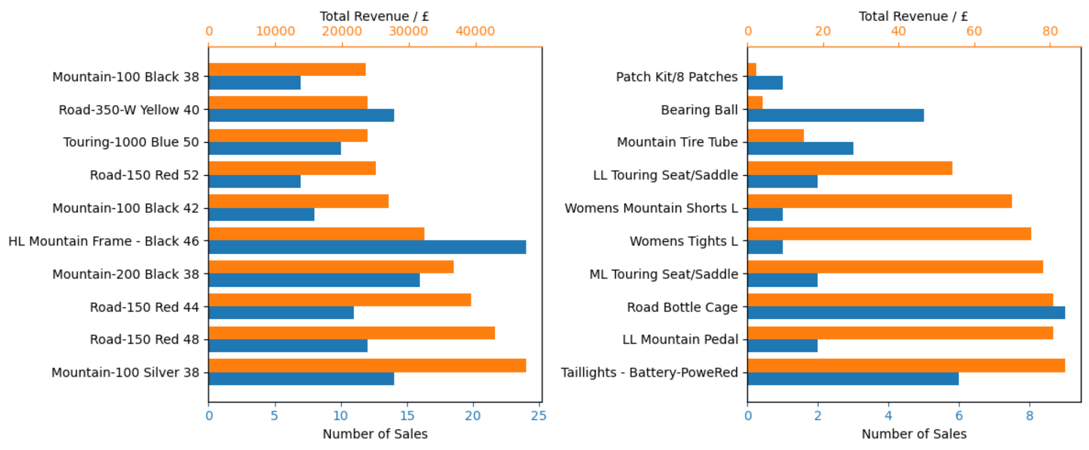


Figure 6: Bar Graph showing Top 10 Products by Highest and Lowest Revenues

Unsurprisingly, the most revenue is being provided by the bike and bike frame products, which are both the most expensive items and popular products. These specific products in the top 10 are candidates for an increase in stock or better display positioning. The least revenue is generated by cheaper products, however we can see that they don't necessarily have a smaller amount of sales. For example, the Road Bottle Cage has more sales that the Road-150 Red. These cheap items with low margin are still very important for a business, we just need to make sure that enough are being sold to justify being retained as a product. Products such as Women's Tights L, we can either stock less of or remove completely.

I then started to investigate the different types of bikes and bike frames, hoping to gain some insight on each type's popularity. I created a 'category' column in the DataFrame and separated the products using the following Python code:

```
#creating a category column,labelling bike and frame categories
productdf['Category']= None
productdf.loc[productdf['Name'].str.startswith('Mountain-'), 'Category'] = 'Mountain Bike'
productdf.loc[productdf['Name'].str.startswith('Road-'), 'Category'] = 'Road Bike'
productdf.loc[productdf['Name'].str.startswith('Touring-'), 'Category'] = 'Touring Bike'
productdf.loc[productdf['Name'].str.contains('Mountain Frame'), 'Category'] = 'Mountain Frame'
productdf.loc[productdf['Name'].str.contains('Road Frame'), 'Category'] = 'Road Frame'
productdf.loc[productdf['Name'].str.contains('Touring Frame'), 'Category'] = 'Touring Frame'

productdf.head()
```

I then created a pie chart to visualise the sales and revenue share of each of these product groups.

```
prodrev = productdf.groupby('Category')[['Total_Revenue','Units_Sold']].sum()
fig2,(ax1,ax2)=plt.subplots(1,2,figsize=(12, 5))
ax1.pie(prodrev['Total_Revenue'],labels=prodrev.index.tolist(),autopct='%1.1f%%')
ax2.pie(prodrev['Units_Sold'],labels=prodrev.index.tolist(),autopct='%1.1f%%')
ax1.set_title('Bikes/Frames Revenue Share')
ax2.set_title('Bikes/Frames Sales Share')
plt.tight_layout()
fig2.savefig('Bikes and Frames Sales')
```

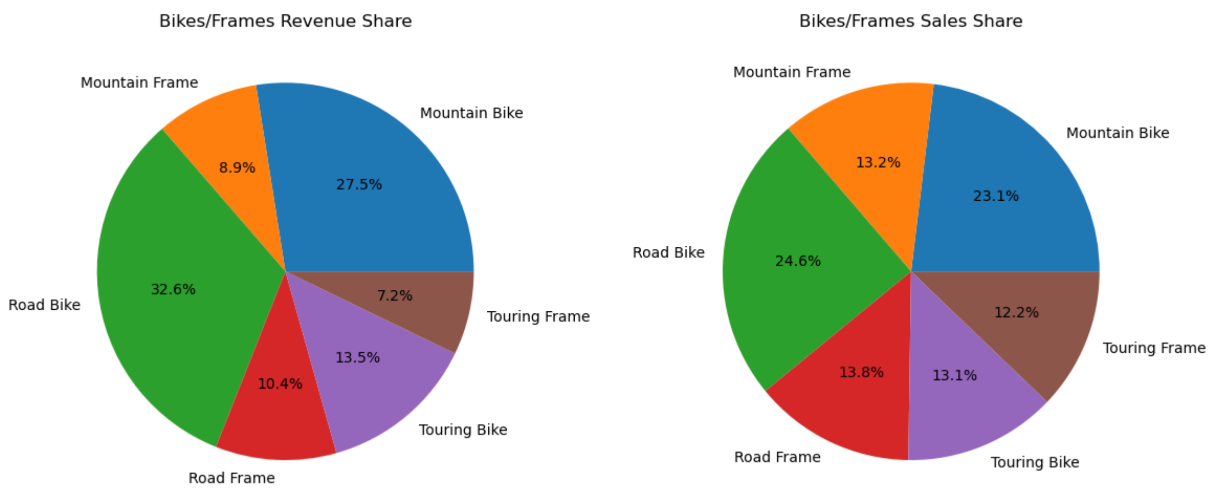


Figure 7: Pie Charts showing the share of sales and revenue from each bike category

Road bikes and road frames are our main revenue driver, accounting for 43% of the total revenue (where only bikes and frames are considered). Marketing these bikes more and increasing stock should be considered as they already provide the business with a healthy amount of revenue. Both the sales and revenue share pie charts show fairly similar proportions across bike categories. This indicates that each category mostly has a similar average selling price and that revenue contribution largely mirrors customer demand. The 8% difference from the Road bike's revenue share to sales share is the most notable change, indicating a slightly higher average selling price compared to other categories. However, we can see that these products are still able to attract customers who are willing to pay more. Overall, no category stands out as being too expensive or cheap, suggesting a consistent pricing strategy and balanced product offering. As a result, category performance is driven mainly by customer preference rather than pricing differences.

I then began to test the hypothesis that prices don't tend to play a significant role in the number of sales of that product. This process also allowed me to gain more insight into the individual products that are performing well, badly or in-between. I created a scatter plot of these bike products, colouring them by their category, and then drew divisions within this graph to create four quadrants that group these products by performance. This code is shown below:

```
#I created a scatter to see if there was any correlation between price and sales, and created four quadrants to distinguish
four groups of products
bikesdf = productdf[productdf['Category'].notna()]
x = bikesdf['Price']
y = bikesdf['Units_Sold']
labels = bikesdf['Name']
color_map = {
    'Mountain Bike': 'darkred',
    'Road Bike': 'darkblue',
    'Touring Bike': 'darkgreen',
    'Mountain Frame': 'red',
    'Road Frame': 'lightblue',
    'Touring Frame': 'lightgreen'
}
colors = bikesdf['Category'].map(color_map)

plt.figure(figsize=(10,6))
scatter = plt.scatter(x,y,c=colors)

# Adding Legend
handles = []
for category, color in color_map.items():
    handles.append(
        plt.Line2D(
            [0], [0],
            marker='o',
            color='w',
            label=category,
            markerfacecolor=color,
            markersize=10))

x_mid = x.mean()
y_mid = y.mean()
plt.axvline(x=x_mid, linestyle='--',color='grey')
plt.axhline(y=y_mid, linestyle='--',color='grey')

plt.text(x_mid*1.05, y_mid*1.05, "High Price, Selling Well",fontsize=8,style='italic')
plt.text(x_mid*0.05, y_mid*1.05, "Low Price, Selling Well",fontsize=8,style='italic')
plt.text(x_mid*0.05, y_mid*0.05, "Low Price, Selling Poorly",fontsize=8,style='italic')
plt.text(x_mid*1.05, y_mid*0.05, "High Price, Selling Poorly",fontsize=8,style='italic')

plt.legend(title="Category", handles=handles, bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

plt.ylabel('Units Sold')
plt.xlabel('Price / £')
plt.tight_layout()
plt.savefig('Bike Products Scatter')
```

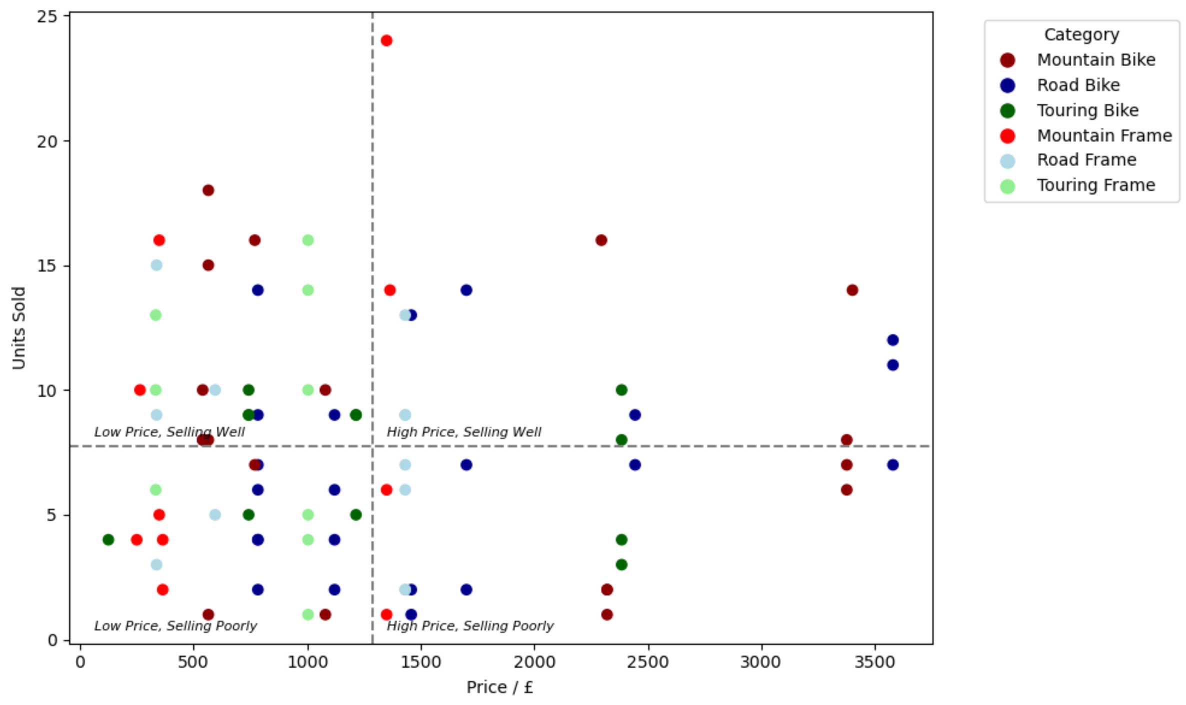


Figure 8: Scatter Graph of Price vs Units Sold for the Bike Products

Again, the lack of correlation in figure 8 shows us that the pricing isn't affecting the customer's purchasing habits, and there seems to be a fairly homogeneous spread of each category. The high price, selling well quadrant are our star performers, which are products we should consider stocking highly on and avoid discounting. We could even consider pricing some of these items up, especially the Mountain Frame product at the top of the Units Sold axis, as this could lead to more overall revenue even if it coincides with a reduction of sales. Our highly priced, low sales quadrant perhaps indicate a mistake in the pricing, as they could be overpriced relative to their perceived value by the customers. Prices could be adjusted or marketing strategies could be reviewed in order to make sure they are reaching the correct audiences. The low price, high sales quadrant are lower margin but high volume products that produce a reliable flow of income. These are popular products so should be advertised regularly to attract customers. Finally, we have our under-performing products in the lower quadrant. Even the low price isn't attracting enough customers to purchase them. These products are candidates for discontinuation or clearance sales.

I then looked at the company's general pricing strategy by creating two histograms to show the distribution of unique products in their pricing, and the distribution of sales.

```
#creating two histograms to compare price distribution of products and how many sales in each price range
x = productdf['Price']
sales = productdf['Units_Sold']
fig3,(ax1,ax2)=plt.subplots(1,2,figsize=(12, 5))
ax1.hist(x,bins=50)
ax1.set_xlabel('Price')
ax1.set_ylabel('Number of Products')
ax2.hist(x,bins=50,weights=sales)
ax2.set_xlabel('Price')
ax2.set_ylabel('Sales')
plt.tight_layout()
plt.savefig('Product Histograms')
```

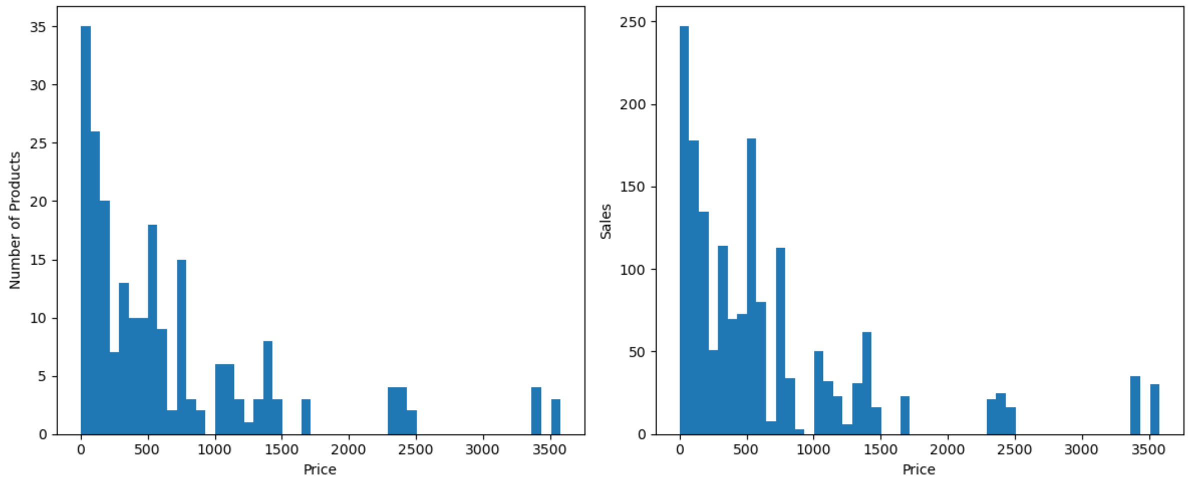


Figure 9: Histograms showing the unique products and sales within each price range

Both of the histograms shown in figure 9 exhibit a similar shape, showing that the volume of products in each price range broadly match the customer purchasing behaviour. The customers are buying products in proportion to how many exist in each price range, which tells us that the product mix is fairly well aligned with the market demand. One exception to this is the price range just above £500, specifically £501.65 to £573.20, where the sales seem to be much higher in comparison to the amount of products within that range. We could potentially try expanding the product range within this bracket as it could increase revenue and better meet customer demand.

Following on from this analysis, it was then time to include a broader range of products into consideration. Since there are far too many individual products to extract meaningful insights from, I decided to group the products further. I kept the bikes and frames separate, but decided to group the rest of the bike parts into their respective categories, Road, Mountain and Touring. This following code shows how I categorised all the products:

```
#creating additional categories to compare revenues
productdf['Category']= None
productdf.loc[productdf['Name'].str.contains('Ball Bearing'), 'Category'] = 'Ball Bearing'
productdf.loc[productdf['Name'].str.contains('Crankarm'), 'Category'] = 'Crankarm'
productdf.loc[productdf['Name'].str.contains('Mountain'), 'Category'] = 'Mountain Bike Part'
productdf.loc[productdf['Name'].str.contains('Road'), 'Category'] = 'Road Bike Part'
productdf.loc[productdf['Name'].str.contains('Touring'), 'Category'] = 'Touring Bike Part'
productdf.loc[productdf['Name'].str.contains('Washer'), 'Category'] = 'Washer'
productdf.loc[productdf['Name'].str.contains('Nut'), 'Category'] = 'Nuts'
productdf.loc[productdf['Name'].str.contains('Metal Sheet'), 'Category'] = 'Metal Sheet'
productdf.loc[productdf['Name'].str.contains('Paint'), 'Category'] = 'Paint'
productdf.loc[productdf['Name'].str.contains('Helmet'), 'Category'] = 'Helmet'
productdf.loc[productdf['Name'].str.contains('Jersey|Shorts|Tights|Gloves|Vest|Socks|Cap'), 'Category'] = 'Clothing'


productdf.loc[productdf['Name'].str.startswith('Mountain-'), 'Category'] = 'Mountain Bike'
productdf.loc[productdf['Name'].str.startswith('Road-'), 'Category'] = 'Road Bike'
productdf.loc[productdf['Name'].str.startswith('Touring-'), 'Category'] = 'Touring Bike'
productdf.loc[productdf['Name'].str.contains('Mountain Frame'), 'Category'] = 'Mountain Frame'
productdf.loc[productdf['Name'].str.contains('Road Frame'), 'Category'] = 'Road Frame'
productdf.loc[productdf['Name'].str.contains('Touring Frame'), 'Category'] = 'Touring Frame'
productdf
```

This information was then used to plot a scatter graph of all the categories by sales and revenue, colour graded by price.

```
prodrev = productdf.groupby('Category')[['Total_Revenue','Units_Sold']].sum()
x = prodrev['Units_Sold']
y = prodrev['Total_Revenue']
labels = prodrev.index.tolist()
avprice = productdf.groupby('Category')['Price'].mean()
```

```
plt.figure(figsize=(10,8))
scatter = plt.scatter(x,y,c=avprice,cmap='inferno')

#creating colorbar that represents the mean price
cbar = plt.colorbar(scatter, orientation='horizontal', pad=0.1,label='Average Price / £')
plt.xlabel('Units Sold')
plt.ylabel('Total Revenue / £')

for xi, yi, label in zip(x, y, labels):
    if label == 'Road Bike Part' or label == 'Paint':
        plt.text(xi, yi+10000, label, fontsize=8, ha='left', va='top')
    elif label == 'Crankarm':
        plt.text(xi, yi-4000, label, fontsize=8, ha='left', va='top')
    else:
        plt.text(xi, yi, label, fontsize=8, ha='left', va='top')

x_mid = x.mean()
y_mid = y.mean()
plt.axvline(x=x_mid, linestyle='--',color='grey')
plt.axhline(y=y_mid, linestyle='--',color='grey')

plt.text(x_mid*2, y_mid*1.05, "Strong Performers",fontsize=8,style='italic')
plt.text(x_mid*0.05, y_mid*1.05, "Low Volume, Strong Revenue",fontsize=8,style='italic')
plt.text(x_mid*0.05, y_mid*0.5, "Weak Performers",fontsize=8,style='italic')
plt.text(x_mid*2, y_mid*0.05, "High Volume, Low Revenue",fontsize=8,style='italic')
plt.tight_layout()
plt.savefig('All Products Scatter')
```

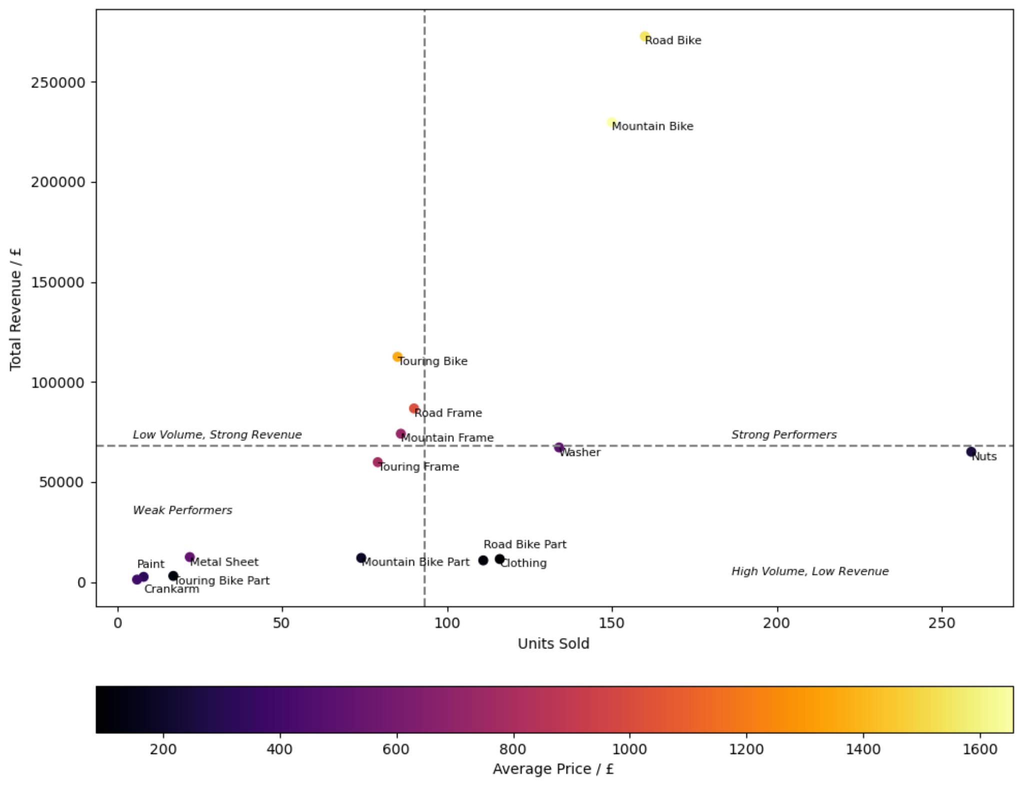


Figure 10: Scatter Graph of Units Sold by Total Revenue for Product Categories (Colour Graded by Price)

Figure 10 several important patterns and shows high priced items generate strong revenue and are selling well despite the premium price tag, while lower priced items rely more heavily on sales volume. There are few items solidly within the low, volume, strong revenue quarter, which would be products that are potentially too expensive. However, we can see be the pattern of the colours that the Touring Bikes looks out of place among these products, as despite their fairly high price, the revenue and sales is comparable to the cheaper Road and Mountain Frames. This could be down to overpricing of certain bikes within this category. In the lower right quadrant, we can see our cheaper but high volume products that bring consistent low margin returns. The ‘Nuts’ category is selling extremely well, so could be considered for a price rise as long as sales don’t take a significant hit. The weaker performers are again our cheaper products, but the cheap price isn’t matched by high customer purchases. These are the products in which we can reduce stock, or bundle with more popular items in order to drive their popularity. Once again we can see the Mountain and Road Bikes are our strongest selling points, and should be advertised well, and made sure that stock is plentiful. Overall, this visualisation helps determine optimal pricing, inventory prioritisation, and marketing focus across product categories.

The final analysis of the products I conducted was looking at the sales of the bikes as a function over time. This required another SQL query to retrieve the sales by day.

```
SELECT s.SalesDate,p.Name,p.Price,s.Quantity,p.Price*s.Quantity AS 'Cost' FROM tblSales s
JOIN tblProducts p ON s.ProductID = p.ProductID
ORDER BY s.SalesDate;
```

I then created a ‘Month’ column in the Pandas DataFrame, as well as categorising each type of bike, but this time I did not separate the frames and full bikes from each category. The purpose of this analysis was to see which type of bike performed well in which months and plot a line graph to visualise this information.

```
#loading the dataset of products sold on each day
salesdf = pd.read_excel("Bicycle Shop Data.xlsx", sheet_name="Sales By Date")
salesdf.head()
salesdf["Month"] = salesdf["Sales Date"].dt.to_period("M").astype(str)
salesdf['Category'] = None
salesdf.loc[salesdf['Product'].str.contains('Mountain'), 'Category'] = 'Mountain Bike'
salesdf.loc[salesdf['Product'].str.contains('Road'), 'Category'] = 'Road Bike'
salesdf.loc[salesdf['Product'].str.contains('Touring'), 'Category'] = 'Touring Bike'
salesdf.head()
y = salesdf.groupby(['Month', 'Category'])['Quantity'].sum().unstack()
y = y.fillna(0)
plt.plot(months,y['Mountain Bike'],label='Mountain Bike')
plt.plot(months,y['Road Bike'],label='Road Bike')
plt.plot(months,y['Touring Bike'],label='Touring Bike')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.legend()
plt.tight_layout()
plt.savefig('Bike Sales by Month')
```

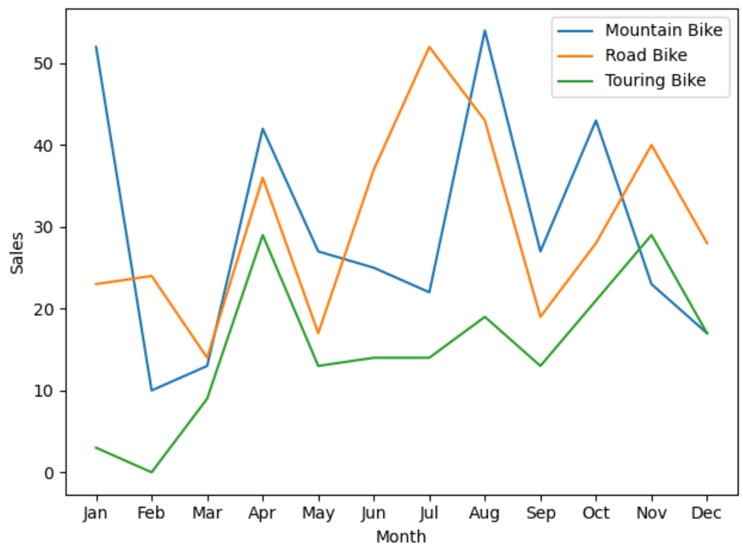


Figure 11: Line Graph Showing the Total Volume of Sales for Each Bike Category Every Month

This figure can be used to guide stock levels throughout the year, as this informs when customers are most likely to purchase each bike in the future. The sales seems to peak towards the end of Summer and start of Autumn, with a slight peak in April for all the bikes as well. Unexpectedly, Mountain Bikes are also very popular in January, unlike the other two products, so we can make sure to keep a high stock of those products then. However, we generally we find the sales tend to rise and fall together, and we can adjust total product levels accordingly.

3 – Customer Analysis

Finally, when it came it analysing customer behaviour, I chose a slightly different method to ones previously applied. I used a common business analytics practice, called RFM analysis, which scores customers based on their recency, frequency and monetary value. Each customer is assigned a value of 1-5 for each of these metrics with 1 being the worst and 5 being the best, and we can then segment those customers based on which type of customer we believe them to be. Our marketing strategies can then be adjusted for each set of customers in each segment.

While analysing the data in Python, I realised that my initial SQL SELECT queries had not included customers who had made no purchases. I corrected this and reuploaded the data. Below I have provided the SQL queries I used, but note that only the data found using the third query was used in the Python analysis. In order to quantify a recency score, I had to find the most recent purchase using a MAX aggregation. I also needed to use a COUNT aggregation rather than SUM such that I could find the number of separate transactions for the frequency score.

```
/*Finding the total purchases, money spent and most recent purchase from each customer*/
SELECT c.FirstName ||' '|| c.LastName AS 'Customer_Name',SUM(s.Quantity) AS 'Purchases',COUNT(s.Quantity) AS
'Separate Transactions',SUM(s.Quantity*p.Price) AS 'Total Spent',MAX(date(s.SalesDate)) AS 'Most Recent Purchase'
FROM tblSales s
JOIN tblCustomers c ON s.CustomerID = c.CustomerID JOIN tblProducts p ON p.ProductID = s.ProductID
GROUP BY Customer_Name
ORDER BY 4 DESC;
/* Checking if there's customers with 0 sales*/
SELECT c.customerid, c.firstname FROM tblcustomers c
LEFT JOIN tblsales s ON c.customerid = s.customerid
WHERE s.customerid IS NULL;
/* Including 0 sales customers in previous query and using COALESCE to turn Null values into zeroes for quantities
purchased*/
SELECT c.FirstName || ' ' || c.LastName AS Customer_Name,COALESCE(SUM(s.Quantity), 0) AS
Purchases,COALESCE(COUNT(s.Quantity), 0) AS "Separate Transactions",COALESCE(SUM(s.Quantity * p.Price), 0) AS
"Total Spent",
MAX(date(s.SalesDate)) AS "Most Recent Purchase"
FROM tblCustomers c
LEFT JOIN tblSales s ON c.CustomerID = s.CustomerID
LEFT JOIN tblProducts p ON s.ProductID = p.ProductID
GROUP BY Customer_Name
ORDER BY "Total Spent" DESC, "Purchases" DESC;
```

For recency, I decided to grade the customers such that each rating had an approximately equal number of people within it, except from the customers with a rating with 1, who are all the customers with no purchases registered in the data. I interpreted these people to be customers who have previously purchased products but not within the last year. Therefore, the 1 grouping had a disproportionately large number of customers compared to the others.

```
custdf = pd.read_excel('Bicycle Shop Data.xlsx',sheet_name='Customers By Purchases')
custdf.head()
#RFM Analysis brings together Recency, Frequency and Monetary values to evaluate customer loyalty, 5=best 1=worst
#first we'll evaluate recency, and assume the date now is 1st Jan 2023
today = pd.to_datetime('01-01-2023')
custdf['Most Recent Purchase'] = pd.to_datetime(custdf['Most Recent Purchase'])
recency = today - custdf['Most Recent Purchase'] #finding days since last purchase
#creating 4 groups of equal size based on recency with a grading 2-5
custdf['Recency Score']=pd.qcut(recency, 4, labels=[5,4,3,2]) #equally distributing ratings 5-2
custdf['Recency Score'] = custdf['Recency Score'].cat.add_categories([1])
custdf['Recency Score'].fillna(1, inplace=True) #customers with no purchases in the last year are assigned a 1 rating
custdf['Recency Score'].value_counts()
```

I chose not to equally distribute the frequency ratings once I realised that the number of separate purchases ranged from 0-4. This range of purchases made it easy to assign a rating, since there are five groups already laid out for us.

```
mapping = {0: 1,1: 2,2: 3,3: 4,4: 5}
custdf['Frequency Score']=custdf['Separate Sales'].map(mapping)
custdf.head()
```

The monetary score I grouped in a similar way to the recency scores, but it required a slightly different coding approach, as the 'Amount Spent' column were populated with zeroes not null values for the customers who didn't spend.

```
spend = custdf['Amount Spent']
zeroes = custdf['Amount Spent'] == 0 #finding the customers who spent nothing
custdf['Monetary Score'] = None #creating a monetary score column
custdf.loc[~zeroes, 'Monetary Score'] #using loc, I can index the rows of people who spent and didn't spend using ~ or no ~
custdf.loc[zeroes, 'Monetary Score'] = 1 #assigning a score of 1 to all 0 spenders
custdf.loc[~zeroes, 'Monetary Score'] = pd.qcut(custdf.loc[~zeroes, 'Amount Spent'],4,labels=[2, 3, 4, 5])
custdf.head()
```

Using these scores, I could then start creating visualisations for the customer data. I began with a heatmap showing the correlations between each score.

```
#creating a figure that shows the correlations between each score e.g. do recent purchases correlate with frequent
purchases
plt.figure(figsize=(10,6))
sns.heatmap(custdf[['Recency Score','Frequency Score','Monetary Score']].astype(int).corr(), annot = True,
        cmap='Blues')
plt.tight_layout()
plt.savefig('Customer RFM Heatmap')
```

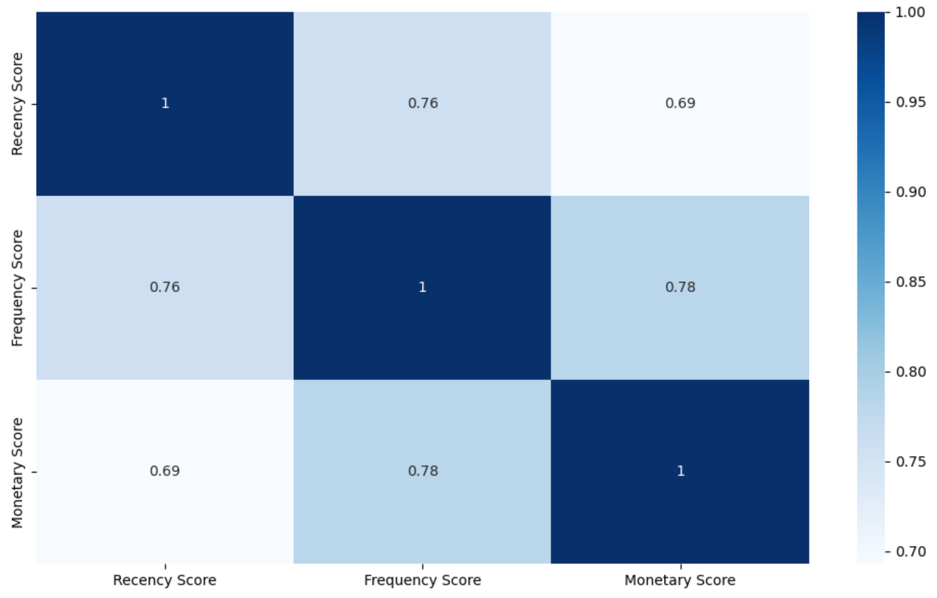


Figure 12: Heatmap Showing the Correlation Between the RFM Scores

The correlation heatmap shows strong positive relationships between all RFM dimensions. Recency and frequency correlate at 0.76, indicating that customers who have purchased recently also tend to purchase more frequently. Recency and monetary correlate at 0.69, the lowest correlation score though still positive, showing that recent customers tend to also be higher spenders, but this isn't necessarily true for all customers. There may be some high spenders in the data who haven't made a purchase recently. Frequency and monetary have the strongest correlation at 0.78, meaning that repeat customers are the highest contributors to revenue. These strong correlations suggest a consistent, loyal customer base where high value customers perform well across all dimensions.

I then segmented the customers using the following code:

```
#categorising into groups based on rfm score
custdf['Segment'] = None
for i in range(len(custdf)):
    r,f,m = custdf.loc[i,'Recency Score'],custdf.loc[i,'Frequency Score'],custdf.loc[i,'Monetary Score']
    if r == 5 and f >= 4 and m >= 4:
        custdf.loc[i,'Segment'] = 'Champions'
    elif f >= 4 and m >= 3: #frequent and spend well
        custdf.loc[i,'Segment'] = 'Loyal Customers'
    elif m == 5 and f >= 3:
        custdf.loc[i,'Segment'] = 'Big Spenders'
    elif r == 5 and f <= 2:
        custdf.loc[i,'Segment'] = 'Recent Customers'
    elif r >= 4 and f >= 2: #fairly recent and have moderate frequency
        custdf.loc[i,'Segment'] = 'Potential Loyalist'
    elif r <= 2 and f >= 3: #was frequent but hasn't spent in a while
        custdf.loc[i,'Segment'] = 'At Risk'
    elif r <= 2 and f <= 2 and m <=2:
        custdf.loc[i,'Segment'] = 'Hibernating'
    elif r >= 3 and f >= 2 and m >= 2:
        custdf.loc[i,'Segment'] = "Regular Customers"
    else:
        custdf.loc[i,'Segment'] = 'Other'
custdf.head()
plt.figure(figsize=(10,5))
sns.countplot(data=custdf, x='Segment', order=custdf['Segment'].value_counts().index)
plt.xticks(rotation=45, ha='right')
plt.xlabel('Customer Segment')
plt.ylabel('Count')
plt.tight_layout()
plt.savefig('Distribution of Customer Segments')
```

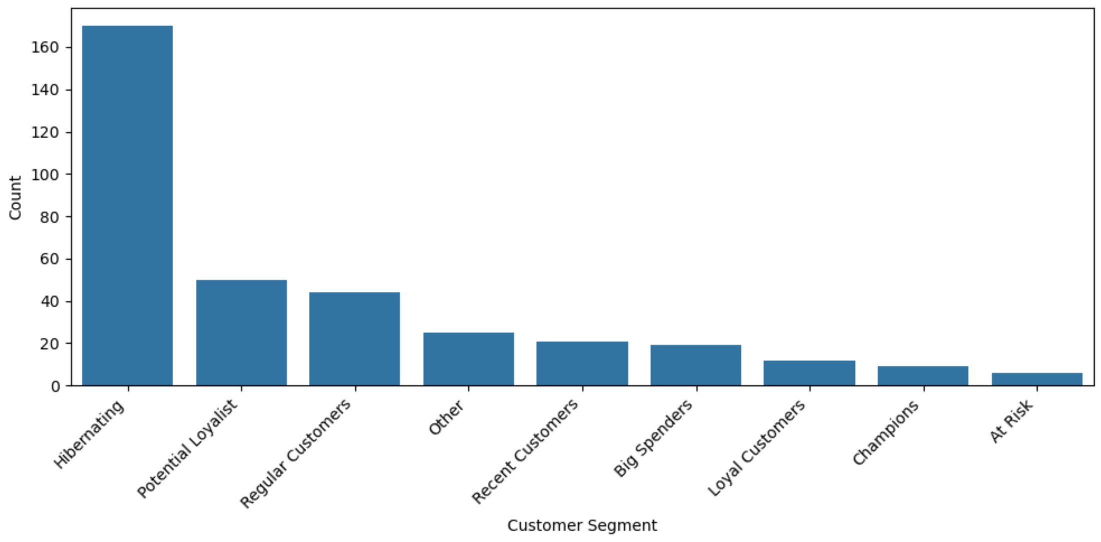


Figure 13: Bar Graph Showing the Total Customers in Each RFM Segment

This bar chart shows that we have a large number of hibernating customers within our database, who may be lost to us. Additional data would be useful to determine when they last purchased a product, and how much they spent, so that we can know whether it would be wise to run reactivation campaigns by sending emails with discounts or “we miss you” offers. For most of these customer, we should consider removing them from marketing spend to reduce cost. We also have a lot of potential loyalists, who have bought recently and more than once. We should encourage them to become higher value customers, like champions and loyal customers, by incentivising repeat purchases through loyalty points, reminder emails, or personalised recommendations based on purchase history.

A similar approach could be applied to regular customers, who score fairly average ratings across the board, but we could also additionally incentivise spending in this segment with new product updates. We need to try build relationships with recent customers, who have scored high in recency but low in frequency, as they are likely to become loyal in the future. We could encourage early engagement with new customer discount codes and follow up emails. The big spenders can provide a big part of our revenue, but at the moment they are not spending as frequently as we would desire. We can encourage more frequent purchases by recommending items similar to what they previously spent big on. The loyal and champions segments are very low comparatively, but this is mainly due to the strict range of values allowed for those segments. We should reward both of these group of customers and give them early access to new products. More importantly we should encourage referrals as champions are great advocates and can bring in even more loyal customers. Losing any of them has a big impact so we should monitor their activity closely. The smallest segment are out at risk customers who were previously highly valuable but now are at risk of hibernating. We should consider sending urgent messaging to win them back, using strong incentives. Furthermore, we need to identify why they left (price, competition, bad experience?) and address it directly, so we can reduce the risk of more customers having similar experiences. Although a small segment, these customers are a high priority, as they used to be frequent buyers.

Finally, I created a scatter graph displaying every customer color graded by their total RFM score, but removed the hibernating customers from the dataset as I thought they wouldn’t provide any useful information.

```
custdf['RFM_Total'] = (custdf['Recency Score'].astype(int) +custdf['Frequency Score'].astype(int) +custdf['Monetary
Score'].astype(int))
active_df = custdf[custdf['Segment'] != 'Hibernating']
labels = active_df['Customer Name']
x=active_df['Purchases']
y=active_df['Amount Spent']

plt.figure(figsize=(10,6))
scatter = plt.scatter(x,y,c=active_df['RFM_Total'],cmap='inferno')
plt.colorbar(scatter, label='RFM Score',orientation='horizontal',pad=0.1)

for xi, yi, label in zip(x, y, labels):
    if xi>15 or yi>20000:
        if label == 'Kate Yuan':
            plt.text(xi, yi+500, label, fontsize=8, ha='left', va='top')
        else:
            plt.text(xi, yi, label, fontsize=8, ha='left', va='top')
plt.xlabel('Number of Purchases')
plt.ylabel('Amount Spent / £')
plt.tight_layout()

plt.savefig('Purchases vs Amount Spent (Colour graded by RFM Score)')
```

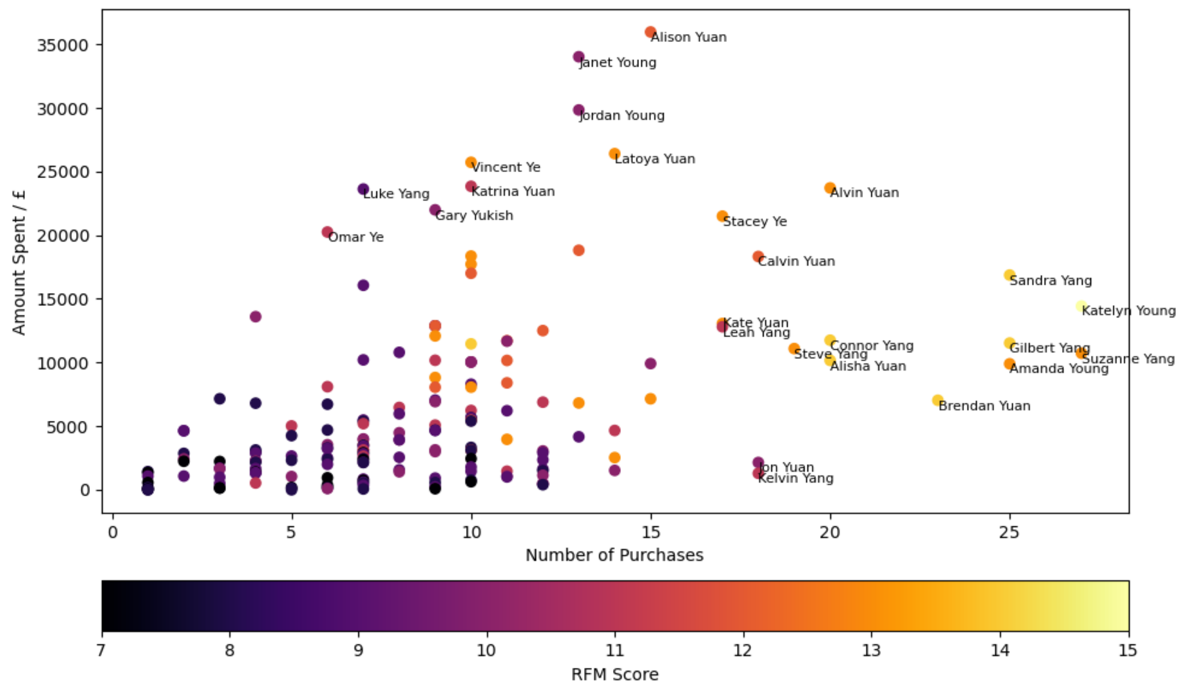


Figure 14: Scatter Graph Showing Purchases vs Amount Spent (Colour Graded by RFM Score)

This figure highlights high value customers, identifies low engagement groups, and validates our RFM scoring, as we can see a trend of similar colours clustering in areas of the graph. It also reveals outliers, and shows whether frequency or individual purchase value drives revenue. For example, we can see a lot of our high RFM scorers clustering towards the right side of the figure, showing their loyalty corresponds to a high amount of purchases. However, these customers aren’t our biggest revenue drivers. Customers like Alison and Janet score averagely in the RFM metric, but have made large enough purchases in one go to offset the lack of frequency. It is worthwhile to emphasise that the x axis on this figure represents total purchases, not separate purchases by date. Perhaps this tells us that the big spenders category are the segment of customers we should try to maximise and invest more time in marketing in order to cultivate. In addition, there are dense amount of data points in the lower left corner of our plot, which shows we have a large proportion of low value customers. There is potential for reactivation of these customers or for them to become loyal in the future, especially those with a lighter color as they may be newer or bought small and frequently. Overall, we can see the fairly strong relationship between the number of purchases and revenue brought in and we should try to encourage frequency where possible. This can be achieved by offering deals where cheaper products can be bundled together, or even buy one get one half price offers.