PROJECT OVERVIEW

Onyxio Company Ltd faces the challenge of accurately tracking and analyzing its revenue and expenses across various business lines. The company aims to enhance its financial decision-making process by developing a comprehensive income statement that captures key financial metrics. Accurate financial reporting is crucial for maintaining profitability, managing costs, and making informed strategic decisions.

BUSINNES PROBLEM

Onyxio Company Ltd, a diversified business, aims to enhance its financial decision-making process to improve profitability and competitiveness. To achieve this objective, Onyxio seeks to develop a comprehensive income statement that accurately tracks and analyzes revenue and expenses across various business lines. By leveraging detailed businness financial data and advanced analytics, Onyxio aims to identify key revenue and expense drivers, optimize cost structures, and implement strategic financial decisions to foster long-term business growth and stability.

PROJECT OBJECTIVE

The objective of this project is to analyze Onyxio Company Ltd’s businness financial data to understand the factors influencing revenue and expenses and develop a comprehensive income statement. By leveraging financial analysis and predictive modeling, the project aims to:

Identify key revenue and expense drivers across different business lines.

Analyze trends and patterns in financial performance over time.

Provide actionable insights to Onyxio Company Ltd for optimizing cost structures and enhancing profitability.

Develop a detailed income statement to support strategic financial decision-making.

Improve Onyxio Company Ltd’s financial management and competitiveness in the market.

RESEARCH QUESTIONS:

1. What are the primary sources of revenue and expenses for Onyxio Company Ltd?
2. How do different business lines contribute to the overall financial performance?
3. What are the trends in revenue and expenses over time?
4. How can Onyxio Company Ltd optimize its cost structure to improve profitability?

DATA UNDERSTANDING

The dataset used in this project was obtained from Onyx Company's internal financial records, which contain detailed information on revenue and expenses across various business lines. This dataset is highly suitable for addressing the business problem at hand of creating an accurate and comprehensive income statement. Contained in the dataset are:

Year: Year of revenue/expense.

Month - name: Month of revenue/expense.

Month -sequence: Month of revenue/expense, expressed as a number.

Date: Date of revenue/expense, expressed as the last day of the month.

Business Line: Business line generating revenue/expense (e.g., Sports Inventory, Sportswear, Nutrition & Food Supplements).

Amount ($): Revenue or expense amount in USD.

Expense Subgroup: Additional subgroups categorizing expenses associated with OPEX and COGS.

Revenue/Expense Group: Subcategory of revenue/expense (e.g., Sales, Consulting and Professional Services, Other income).

Revenue or Expense: Column indicating if the associated amount is revenue or expense.

In [1]:
```
# importing the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestRegressor

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
#pip install xlrd
```

In [3]:
```
# loading data
df = pd.read_csv('Onyx Data.csv',encoding='ISO-8859-1', index_col = 0)
df
```

Out[3]:

Year	Month - name	Month - sequence	Date	Business Line	Amount, $	Expense subgroup	Revenue / Expense Group	Rever exper
2023	January	1	1/31/2023	Nutrition and Food Supplements	153000	NaN	Sales	Rever
2023	January	1	1/31/2023	Nutrition and Food Supplements	27000	NaN	Consulting and professional services	Rever
2023	January	1	1/31/2023	Nutrition and Food Supplements	6000	NaN	Other income	Rever
2023	January	1	1/31/2023	Nutrition and Food Supplements	-15000	Rent	Opex	Exper
2023	January	1	1/31/2023	Nutrition and Food Supplements	-9000	Equipment	Opex	Exper
...	...	...	...	...	...	...	...	
2023	December	1	12/31/2023	Sportswear	-4000	Packaging	COGS	Exper
2023	December	1	12/31/2023	Sportswear	-7000	Shipping	COGS	Exper
2023	December	1	12/31/2023	Sportswear	-9000	Sales	COGS	Exper
2023	December	1	12/31/2023	Sportswear	-110000	Labor	COGS	Exper
2023	December	1	12/31/2023	Sportswear	-7000	Other	COGS	Exper

580 rows × 8 columns

Data Cleaning and Preparation

In [4]:
```
# check for columns
df.columns
```

Out[4]:
```
Index(['Month - name', 'Month -sequence', 'Date', 'Business Line', 'Amount,
$',
       'Expense subgroup', 'Revenue / Expense Group', 'Revenue or expense'],
      dtype='object')
```

In [5]:
```
#check for datatypes
df.dtypes
```

Out[5]:
```
Month - name               object
Month -sequence             int64
Date                       object
Business Line              object
Amount, $                   int64
Expense subgroup           object
Revenue / Expense Group    object
Revenue or expense         object
dtype: object
```

In [6]:
```
#Check for number of columns, column labels, column data types, memory usage, r
#the number of cells in each column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 580 entries, 2023 to 2023
Data columns (total 8 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Month - name             580 non-null    object
 1   Month -sequence          580 non-null    int64 
 2   Date                     580 non-null    object
 3   Business Line            580 non-null    object
 4   Amount, $                580 non-null    int64 
 5   Expense subgroup         468 non-null    object
 6   Revenue / Expense Group  580 non-null    object
 7   Revenue or expense       580 non-null    object
dtypes: int64(2), object(6)
memory usage: 40.8+ KB
```

In [7]:
```
#changing datatypes
df['Date'] = pd.to_datetime(df['Date'])
df.head()
```

Out[7]:

Year	Month - name	Month - sequence	Date	Business Line	Amount, $	Expense subgroup	Revenue / Expense Group	Revenue or expense
2023	January	1	2023-01-31	Nutrition and Food Supplements	153000	NaN	Sales	Revenue
2023	January	1	2023-01-31	Nutrition and Food Supplements	27000	NaN	Consulting and professional services	Revenue
2023	January	1	2023-01-31	Nutrition and Food Supplements	6000	NaN	Other income	Revenue
2023	January	1	2023-01-31	Nutrition and Food Supplements	-15000	Rent	Opex	Expense
2023	January	1	2023-01-31	Nutrition and Food Supplements	-9000	Equipment	Opex	Expense

In [8]:
```
#checking for null values
df.isnull().sum()
```

Out[8]:
```
Month - name                0
Month -sequence             0
Date                        0
Business Line               0
Amount, $                   0
Expense subgroup          112
Revenue / Expense Group     0
Revenue or expense          0
dtype: int64
```

In [9]:
```
#displaying the dataframe
df
```

Out[9]:

Year	Month - name	Month - sequence	Date	Business Line	Amount, $	Expense subgroup	Revenue / Expense Group	Revenue or expense
2023	January	1	2023-01-31	Nutrition and Food Supplements	153000	NaN	Sales	Revenue
2023	January	1	2023-01-31	Nutrition and Food Supplements	27000	NaN	Consulting and professional services	Revenue
2023	January	1	2023-01-31	Nutrition and Food Supplements	6000	NaN	Other income	Revenue
2023	January	1	2023-01-31	Nutrition and Food Supplements	-15000	Rent	Opex	Expense
2023	January	1	2023-01-31	Nutrition and Food Supplements	-9000	Equipment	Opex	Expense
...	...	...	...	...	...	...	...	...
2023	December	1	2023-12-31	Sportswear	-4000	Packaging	COGS	Expense
2023	December	1	2023-12-31	Sportswear	-7000	Shipping	COGS	Expense
2023	December	1	2023-12-31	Sportswear	-9000	Sales	COGS	Expense
2023	December	1	2023-12-31	Sportswear	-110000	Labor	COGS	Expense
2023	December	1	2023-12-31	Sportswear	-7000	Other	COGS	Expense

580 rows × 8 columns

In [10]:
```
#checking for column names
df.columns
```

Out[10]:
```
Index(['Month - name', 'Month -sequence', 'Date', 'Business Line', 'Amount,
$',
       'Expense subgroup', 'Revenue / Expense Group', 'Revenue or expense'],
      dtype='object')
```

In [11]:
```
 #Initialize new columns with NaN values
# Populate the 'Revenue' and 'Expenses' columns based on the 'Revenue or expens

df['Revenue'] = df['Amount, $'].where(df['Revenue or expense'] == 'Revenue', 0
df['Expenses'] = df['Amount, $'].where(df['Revenue or expense'] == 'Expense', 

# Drop the original 'Revenue or expense' column if no longer needed
df.drop(columns=['Revenue or expense'], inplace=True)
df.drop(columns=['Revenue / Expense Group'], inplace=True)
```

In [12]:
```
#displaying the dataframe
df
```

Out[12]:

Year	Month - name	Month - sequence	Date	Business Line	Amount, $	Expense subgroup	Revenue	Expenses
2023	January	1	2023-01-31	Nutrition and Food Supplements	153000	NaN	153000	0
2023	January	1	2023-01-31	Nutrition and Food Supplements	27000	NaN	27000	0
2023	January	1	2023-01-31	Nutrition and Food Supplements	6000	NaN	6000	0
2023	January	1	2023-01-31	Nutrition and Food Supplements	-15000	Rent	0	-15000
2023	January	1	2023-01-31	Nutrition and Food Supplements	-9000	Equipment	0	-9000
...	...	...	...	...	...	...	...	...

In [13]:
```
#Dropping the columns not needed for analysis

columns_to_drop = ['Year', 'Month -name', 'Month -sequence', 'Amount, $', 'Exp
df1 = df.drop(columns=[col for col in columns_to_drop if col in df.columns])
df1
```

Out[13]:

Year	Month - name	Date	Business Line	Revenue	Expenses
2023	January	2023-01-31	Nutrition and Food Supplements	153000	0
2023	January	2023-01-31	Nutrition and Food Supplements	27000	0
2023	January	2023-01-31	Nutrition and Food Supplements	6000	0
2023	January	2023-01-31	Nutrition and Food Supplements	0	-15000
2023	January	2023-01-31	Nutrition and Food Supplements	0	-9000
...	...	...	...	...	...
2023	December	2023-12-31	Sportswear	0	-4000
2023	December	2023-12-31	Sportswear	0	-7000
2023	December	2023-12-31	Sportswear	0	-9000
2023	December	2023-12-31	Sportswear	0	-110000

In [14]:
```
# Convert the 'Expenses' column to absolute values to remove the negative sign
df1['Expenses'] = df1['Expenses'].abs()
df1
```

Out[14]:

Year	Month - name	Date	Business Line	Revenue	Expenses
2023	January	2023-01-31	Nutrition and Food Supplements	153000	0
2023	January	2023-01-31	Nutrition and Food Supplements	27000	0
2023	January	2023-01-31	Nutrition and Food Supplements	6000	0
2023	January	2023-01-31	Nutrition and Food Supplements	0	15000
2023	January	2023-01-31	Nutrition and Food Supplements	0	9000
...	...	...	...	...	...
2023	December	2023-12-31	Sportswear	0	4000
2023	December	2023-12-31	Sportswear	0	7000
2023	December	2023-12-31	Sportswear	0	9000
2023	December	2023-12-31	Sportswear	0	110000
2023	December	2023-12-31	Sportswear	0	7000

580 rows × 5 columns

In [15]:
```
#adding column Sales
df1 = df1.assign(Sales=df1['Revenue'] + df1['Expenses'])
df1
```

Out[15]:

Year	Month - name	Date	Business Line	Revenue	Expenses	Sales
2023	January	2023-01-31	Nutrition and Food Supplements	153000	0	153000
2023	January	2023-01-31	Nutrition and Food Supplements	27000	0	27000
2023	January	2023-01-31	Nutrition and Food Supplements	6000	0	6000
2023	January	2023-01-31	Nutrition and Food Supplements	0	15000	15000
2023	January	2023-01-31	Nutrition and Food Supplements	0	9000	9000
...	...	...	...	...	...	...
2023	December	2023-12-31	Sportswear	0	4000	4000
2023	December	2023-12-31	Sportswear	0	7000	7000
2023	December	2023-12-31	Sportswear	0	9000	9000
2023	December	2023-12-31	Sportswear	0	110000	110000
2023	December	2023-12-31	Sportswear	0	7000	7000

580 rows × 6 columns

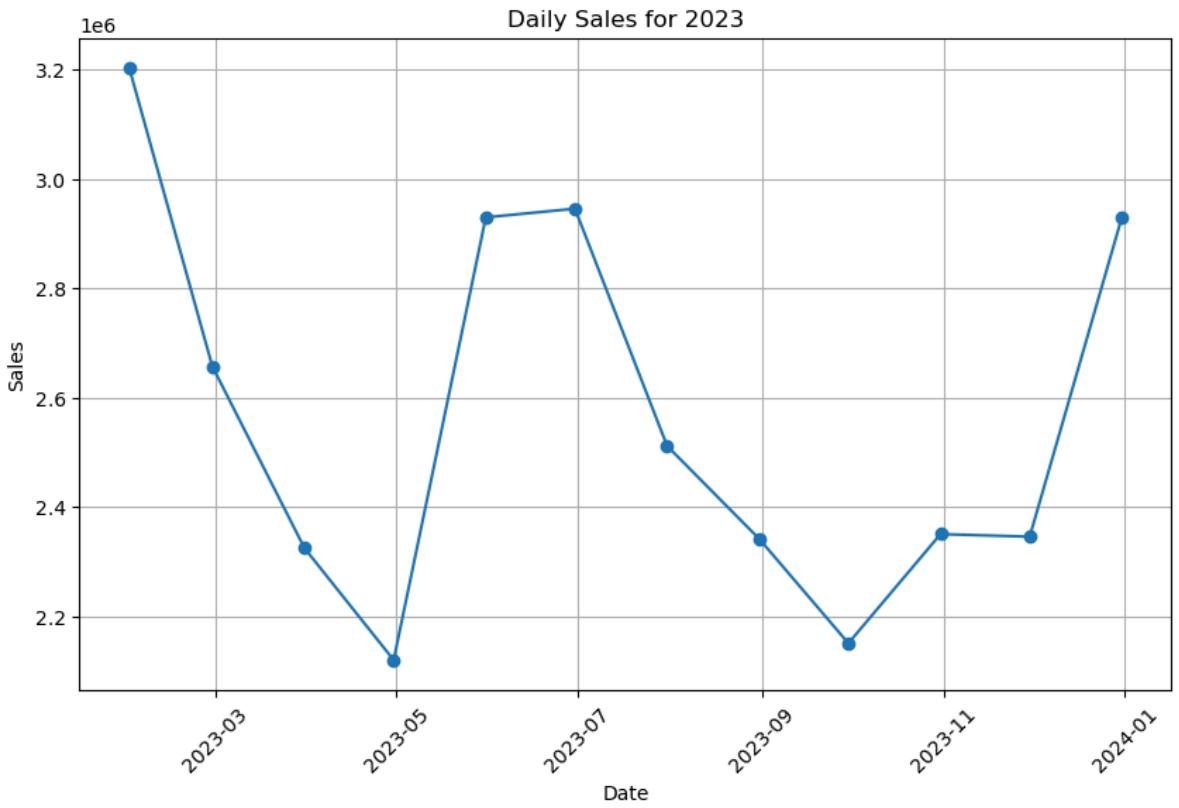
In [16]:
```
# Confirming if we have any null values
df1.isnull().sum()
```

Out[16]:
```
Month - name     0
Date             0
Business Line    0
Revenue          0
Expenses         0
Sales            0
dtype: int64
```

DATA ANALYSIS AND VISUALIZATION

How are the various variables presented in our dataset are affecting Oynxio Co Ltd performance.

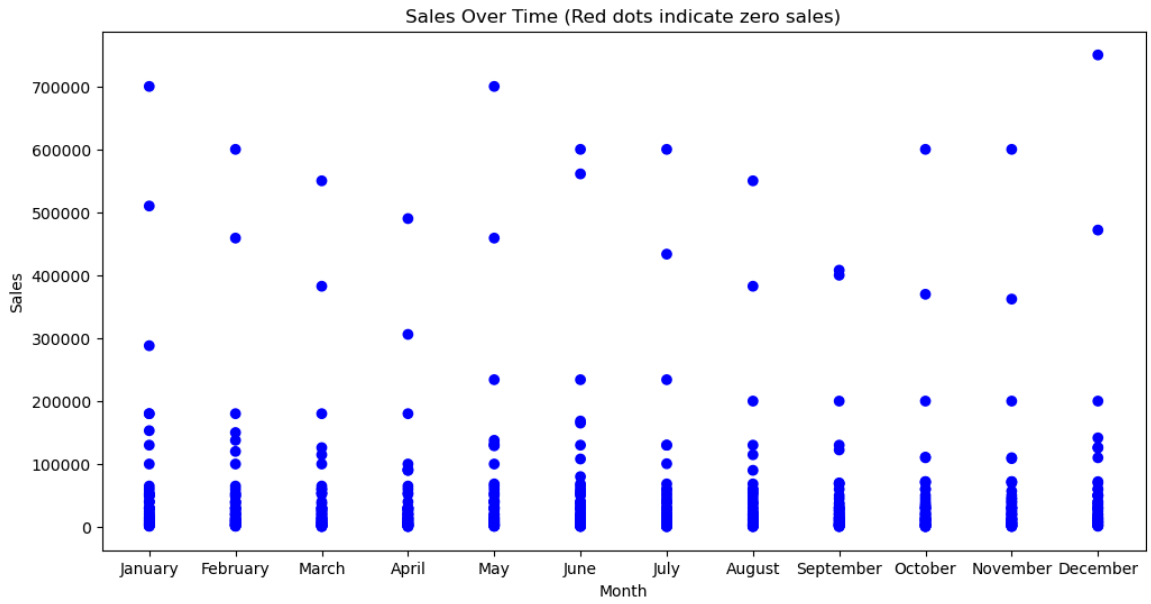
In [17]:
```
#Show if there are days the sales were zero:
# Group by date and sum sales
daily_sales = df1.groupby('Date')['Sales'].sum().reset_index()

# Identify days with no sales
no_sales_days = daily_sales[daily_sales['Sales'] == 0]['Date']

# Plot sales per day
plt.figure(figsize=(10, 6))
plt.plot(daily_sales['Date'], daily_sales['Sales'], marker='o', linestyle='-')
# labelling x, y axis and title
plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('Daily Sales for 2023')
plt.xticks(rotation=45)
plt.grid(True)

# Highlight days with no sales
for day in no_sales_days:
    plt.axvline(x=day, color='r', linestyle='--', alpha=0.5)

plt.show()

# Print days with no sales
print("Days with no sales:")
print(no_sales_days)
```




```
Days with no sales:
Series([], Name: Date, dtype: datetime64[ns])
```

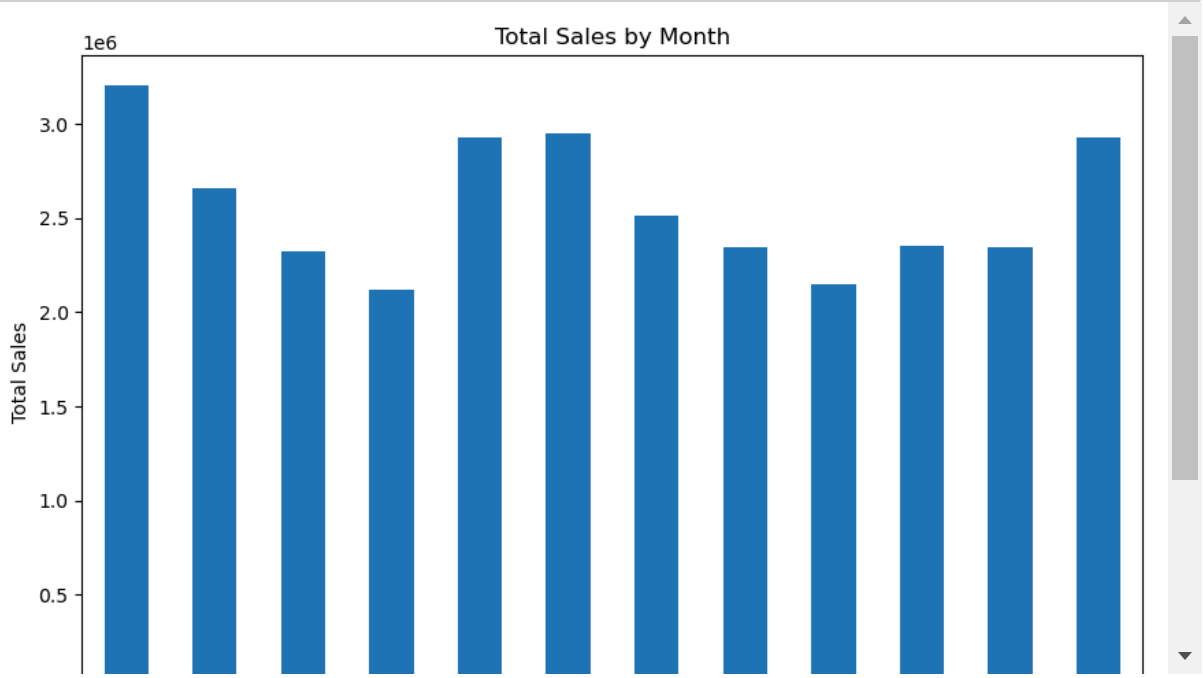
There is no day Onyxio Co Ltd made zero sales. The year begins with very high sales in January and there's a significant spike in sales from May to June, almost doubling the sales figures. This might indicate a successful marketing campaign, product launch, or the beginning of a peak season for the business. Sales remain relatively stable at a high level through June and July, suggesting a strong summer season for the business. Starting from October, sales begin to increase again, with a particularly sharp rise in December. This likely represents holiday season shopping and year-end promotions.

In [18]:
```
#Show if there are days the sales were zero:
# Define the correct month order (keep this if needed for other purposes)
month_order = ['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November', 'December
# Convert 'Month - name' to categorical data type with specific order
df1['Month - name'] = pd.Categorical(df1['Month - name'], categories=month_ord

# Convert 'Date' to datetime and sort
#df1['Date'] = pd.to_datetime(df1['Date'])
df1 = df1.sort_values('Month - name')

# Plot
plt.figure(figsize=(12, 6))
plt.scatter(df1['Month - name'], df1['Sales'], c=df1['Sales'].apply(lambda x:
#labelling the tile
plt.title('Sales Over Time (Red dots indicate zero sales)')
# labelling the x and y axis
plt.xlabel('Month')
plt.ylabel('Sales')
#display
plt.show()
print("Month with highest sales:", monthly_sales.index[0])
```



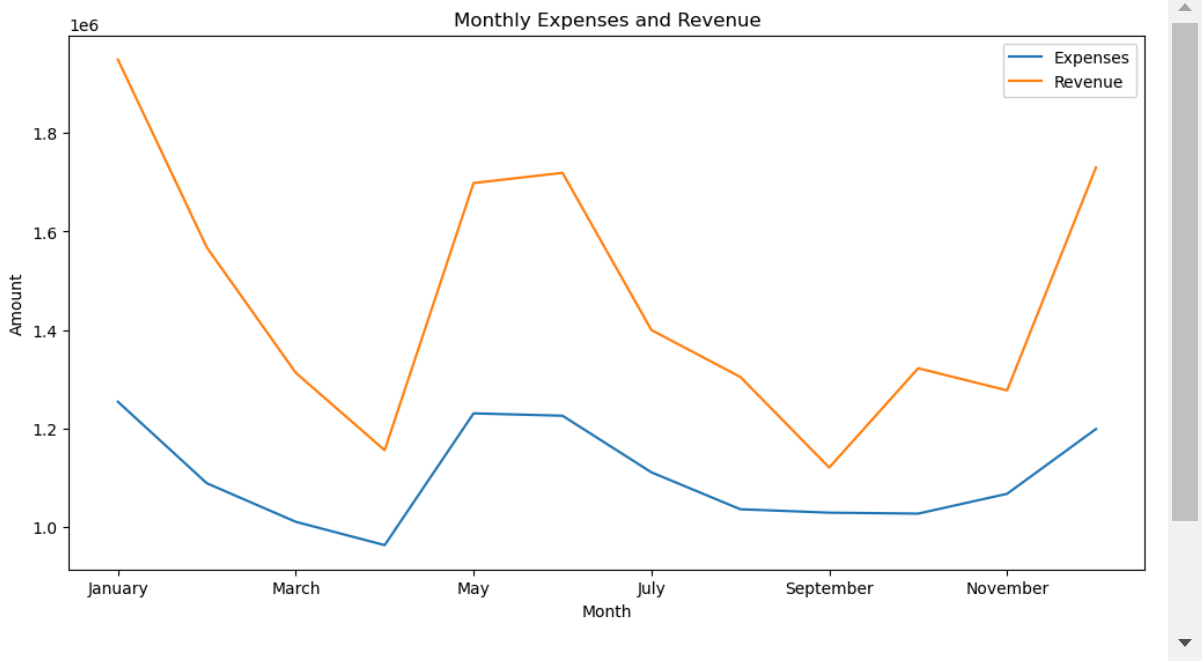
From the above visualization we observe that there are clustered lower sales. Thus a dense concentration of dots at lower sales values suggests that most days have relatively lower sales,

In [19]:
```
#Which month had highest sales:
# Define the correct month order
month_order = ['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November', 'December

# Convert 'Month - name' to categorical data type with specific order
df1['Month - name'] = pd.Categorical(df1['Month - name'], categories=month_ord

# Sort the dataframe by 'Month - name'
df1 = df1.sort_values('Month - name')
#grouping by month-name
monthly_sales = df1.groupby('Month - name')['Sales'].sum()
#ploting the bar chart figsize 10,6
plt.figure(figsize=(10, 6))
monthly_sales.plot(kind='bar')
#Labelling the x,y axis and the title.
plt.title('Total Sales by Month')
plt.xlabel('Month')
plt.ylabel('Total Sales')
#display
plt.show()
print("Month with highest sales:", monthly_sales.index[0])
```



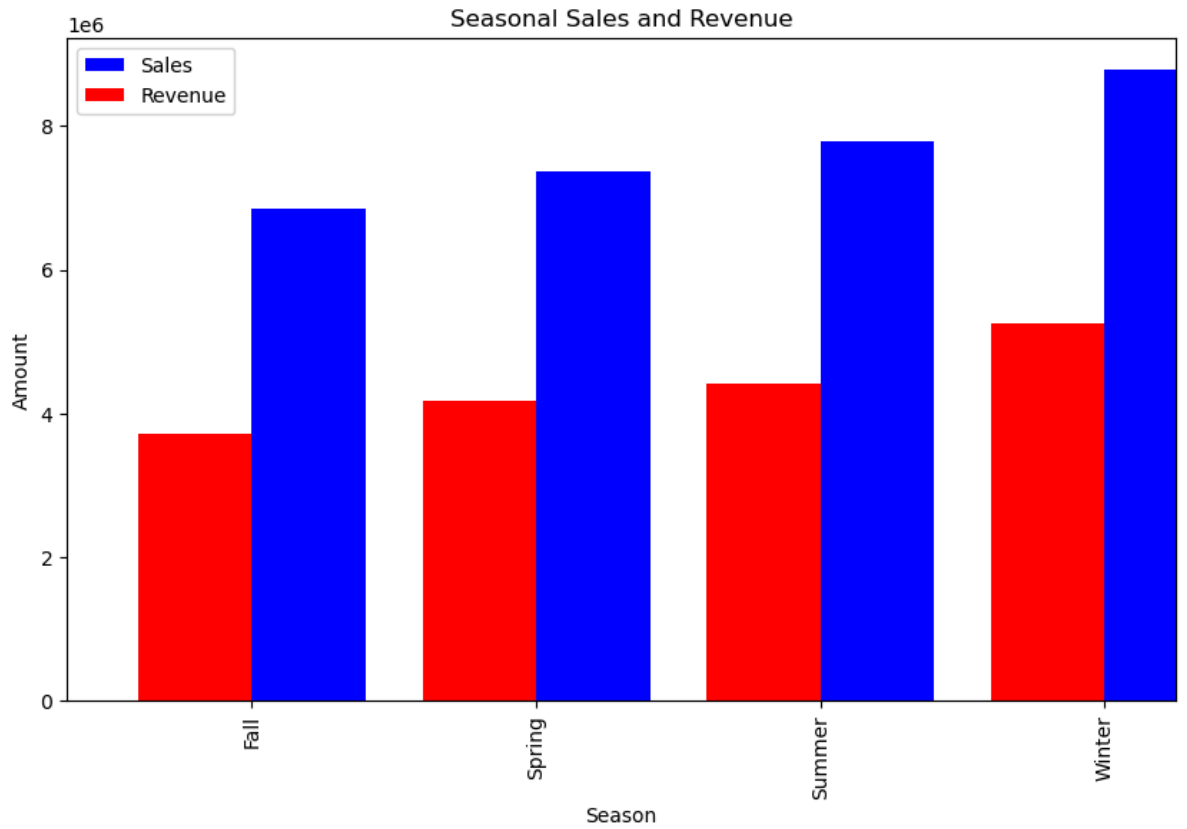
January has the highest sales, followed closely by May, June, and December while April and September show the lowest total sales. Sales vary considerably month-to-month, suggesting strong seasonal influences or specific events affecting sales performance.

In [20]:
```
#Which month had highest expenses/revenue:

monthly_expenses = df1.groupby('Month - name')['Expenses'].sum()
monthly_revenue = df1.groupby('Month - name')['Revenue'].sum()
# Create a new figure with specified size
plt.figure(figsize=(12, 6))
# Plot monthly expenses and revenue
monthly_expenses.plot(label='Expenses')
monthly_revenue.plot(label='Revenue')
# Setting title and labels for the plot
plt.title('Monthly Expenses and Revenue')
plt.xlabel('Month')
plt.ylabel('Amount')
plt.legend()
# Display the plot
plt.show()
# Print months with highest and lowest expenses and revenue
print("Month with highest expenses:", monthly_expenses.idxmax())
print("Month with highest revenue:", monthly_revenue.idxmax())
print("Month with lowest expenses:", monthly_expenses.idxmin())
print("Month with lowest revenue:", monthly_revenue.idxmin())
```



Revenue is consistently higher than expenses, indicating profitability throughout the year. Both revenue and expenses follow similar patterns, peaking in January and May-June, and dipping in April and September-October. There is a large gap between revenue and expenses in January, suggesting the highest profit margin. Expenses increase when revenue increases, indicating that the business maintains profitability while experiencing seasonal variations in both income and costs.

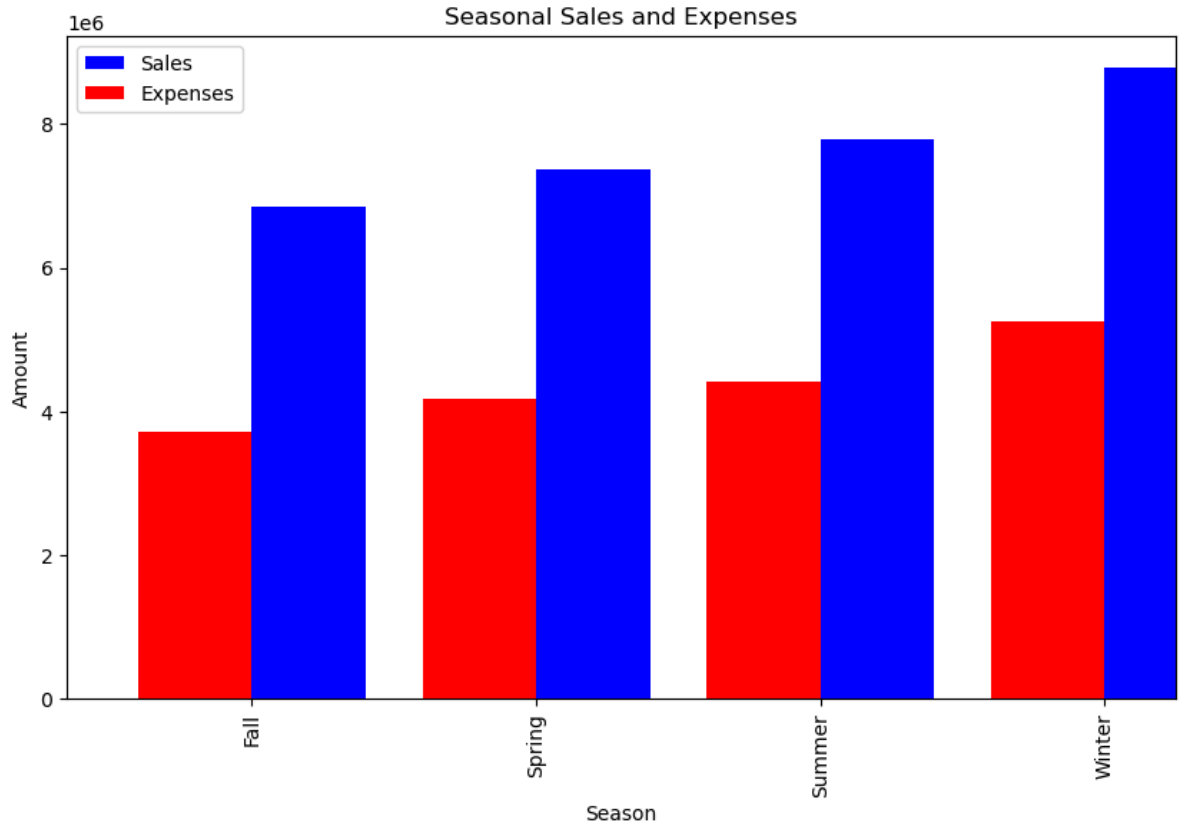
In [21]:
```
# Add a season column
# Map months to seasons
df1['Season'] = pd.to_datetime(df1['Date']).dt.month.map({12:'Winter', 1:'Wint
                                                          3:'Spring', 4:'Spring
                                                          6:'Summer', 7:'Summer
                                                          9:'Fall', 10:'Fall',
# Group data by season and sum sales and revenue
seasonal_sales = df1.groupby('Season')['Sales'].sum()
seasonal_revenue = df1.groupby('Season')['Revenue'].sum()
# Create a new figure with specified siz
plt.figure(figsize=(10, 6))
# Plot seasonal sales and revenue as bar charts
seasonal_sales.plot(kind='bar', position=0, width=0.4, label='Sales',color='bl
seasonal_revenue.plot(kind='bar', position=1, width=0.4, label='Revenue',color
# Set title and labels for the plot
plt.title('Seasonal Sales and Revenue')
plt.xlabel('Season')
plt.ylabel('Amount')
plt.legend()
plt.show()
# Print seasons with highest and lowest sales and revenue
print("Season with highest revenue:", seasonal_revenue.idxmax())
print("Season with highest sales:", seasonal_sales.idxmax())
print("Season with lowest sales:", seasonal_sales.idxmin())
print("Season with lowest revenue:", seasonal_revenue.idxmin())
```



```
Season with highest revenue: Winter
Season with highest sales: Winter
Season with lowest sales: Fall
Season with lowest revenue: Fall
```

The sales and revenue are high in summer and lowest in winter.

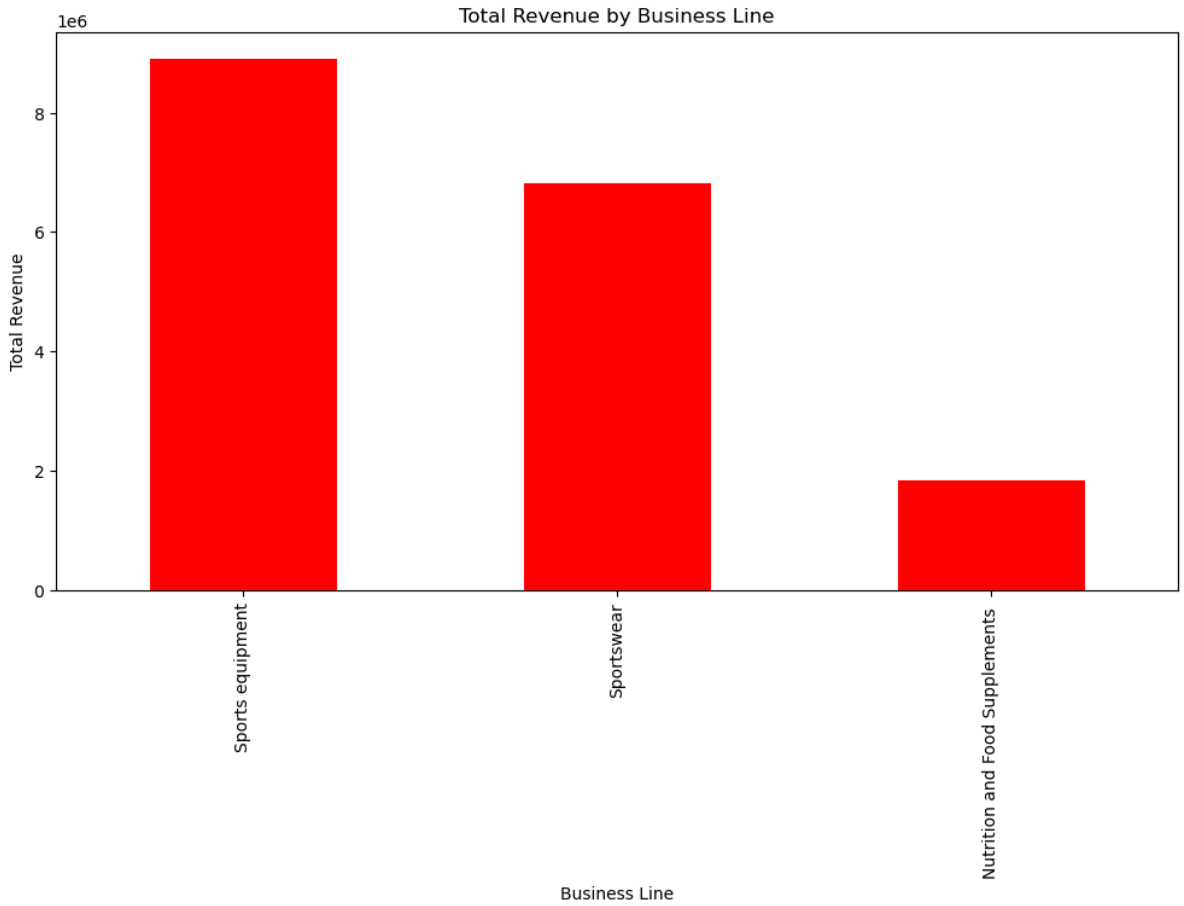
In [22]:
```
#Seasons with highest sales and revenue:

# Add a season column
df1['Season'] = pd.to_datetime(df1['Date']).dt.month.map({12:'Winter', 1:'Wint
                                                          3:'Spring', 4:'Spring
                                                          6:'Summer', 7:'Summer
                                                          9:'Fall', 10:'Fall',
# Group data by season and sum sales and expenses
seasonal_sales = df1.groupby('Season')['Sales'].sum()
seasonal_expenses = df1.groupby('Season')['Expenses'].sum()
# Create a new figure with specified size
plt.figure(figsize=(10, 6))
# Plot seasonal sales and expenses as bar charts
seasonal_sales.plot(kind='bar', position=0, width=0.4, label='Sales',color='bl
seasonal_revenue.plot(kind='bar', position=1, width=0.4, label='Expenses',colo
# Set title and labels for the plot
plt.title('Seasonal Sales and Expenses')
plt.xlabel('Season')
plt.ylabel('Amount')
plt.legend()
#display the plot
plt.show()
# Print seasons with highest and lowest sales and expenses
print("Season with highest expenses:", seasonal_expenses.idxmax())
print("Season with highest sales:", seasonal_sales.idxmax())
print("Season with lowest expenses:", seasonal_sales.idxmin())
print("Season with lowest sales:", seasonal_expenses.idxmin())
```



```
Season with highest expenses: Winter
Season with highest sales: Winter
Season with lowest expenses: Fall
Season with lowest sales: Fall
```

Sales are higher than expenses in all seasons except Winter. With the highest sales been recorded in summer.

In [23]:
```
#Business line with highest revenue by end of year:
# Group data by 'Business Line' and sum 'Revenue', then sort in descending orde
business_line_revenue = df1.groupby('Business Line')['Revenue'].sum().sort_valu
# Create a new figure with specified size
plt.figure(figsize=(12, 6))
# Plot the business line revenue as a bar chart
business_line_revenue.plot(kind='bar',color='red')
# Set title and labels for the plot
plt.title('Total Revenue by Business Line')
plt.xlabel('Business Line')
plt.ylabel('Total Revenue')
#display the plot
plt.show()
# Print the business line with the highest revenue
print("Business line with highest revenue:", business_line_revenue.index[0])
```

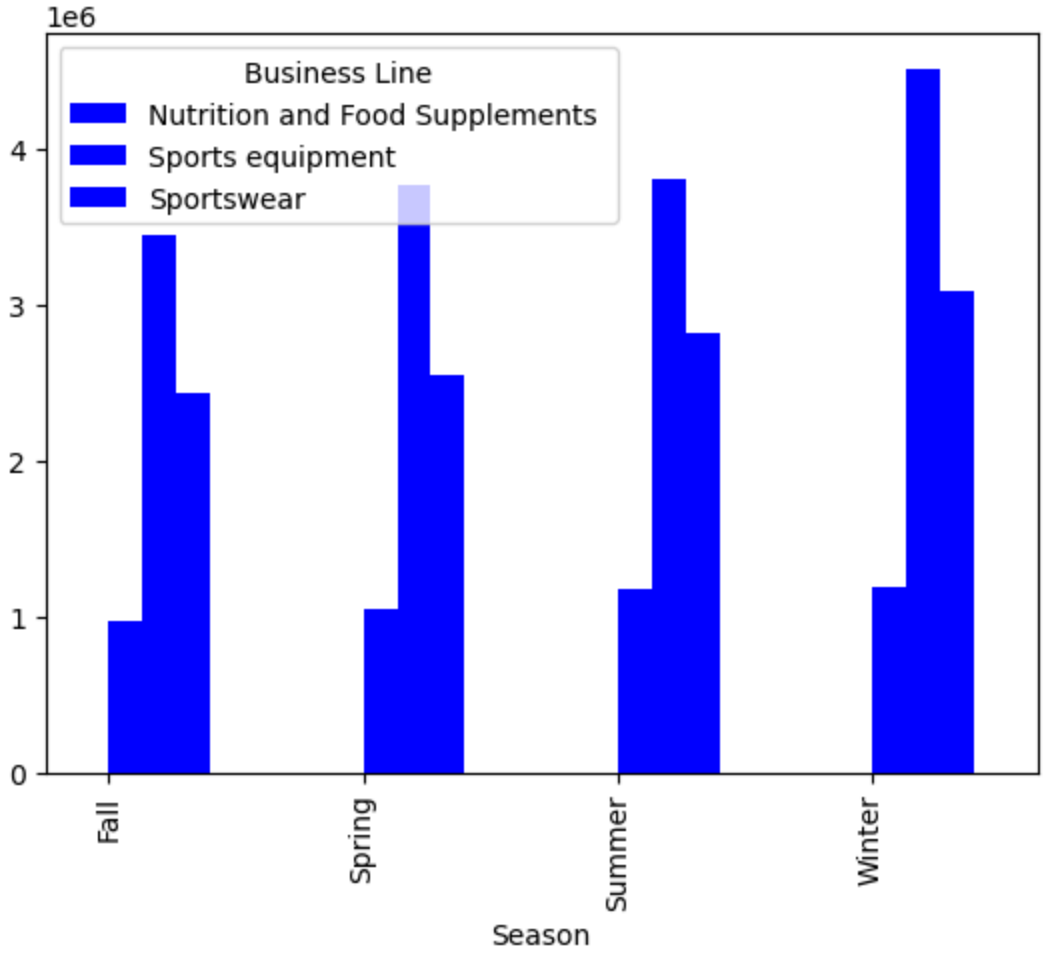


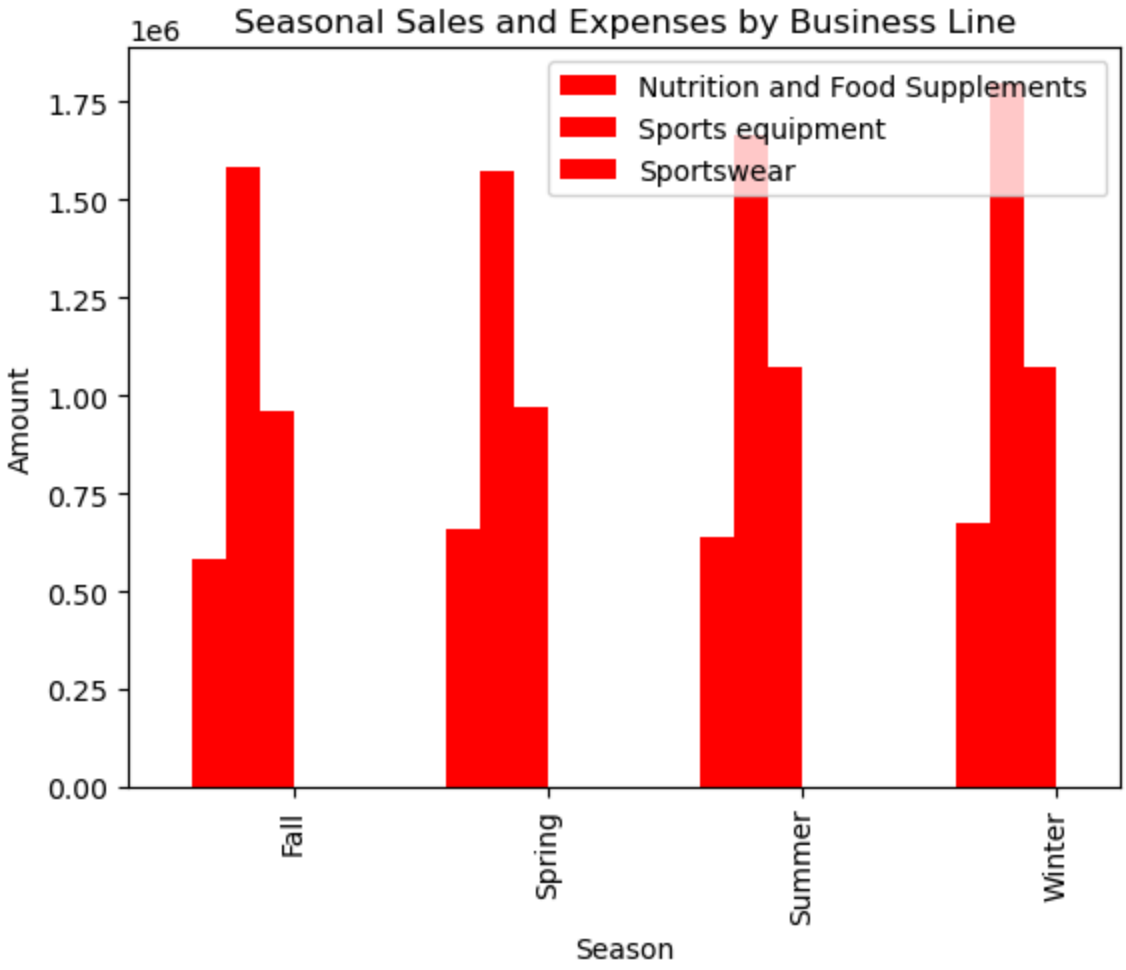

```
Business line with highest revenue: Sports equipment
```

Sports Equipment line generates the most revenue while sports wear line generates half of the sports equipment line and Nutrition and food supplements line generates the lowest revenue.

In [24]:
```
#Business line performance seasonwise;
# Add a season column
df1['Season'] = pd.to_datetime(df1['Date']).dt.month.map({12:'Winter', 1:'Wint
                                                          3:'Spring', 4:'Spring
                                                          6:'Summer', 7:'Summer
                                                          9:'Fall', 10:'Fall',

# Group by Season and Business Line
seasonal_sales = df1.groupby(['Season', 'Business Line'])['Sales'].sum().unsta
seasonal_expenses = df1.groupby(['Season', 'Business Line'])['Expenses'].sum()

# Plotting
plt.figure(figsize=(12, 8))
# Plot seasonal sales and expenses as grouped bar charts
seasonal_sales.plot(kind='bar', position=0, width=0.4, label='Sales', color='b
seasonal_expenses.plot(kind='bar', position=1, width=0.4, label='Expenses', co
# Set title and labels for the plot
plt.title('Seasonal Sales and Expenses by Business Line')
plt.xlabel('Season')
plt.ylabel('Amount')
plt.legend()
plt.show()

# Print seasons with highest and lowest sales and expenses
print("Season with highest expenses:", seasonal_expenses.sum(axis=1).idxmax())
print("Season with highest sales:", seasonal_sales.sum(axis=1).idxmax())
print("Season with lowest expenses:", seasonal_expenses.sum(axis=1).idxmin())
print("Season with lowest sales:", seasonal_sales.sum(axis=1).idxmin())
```

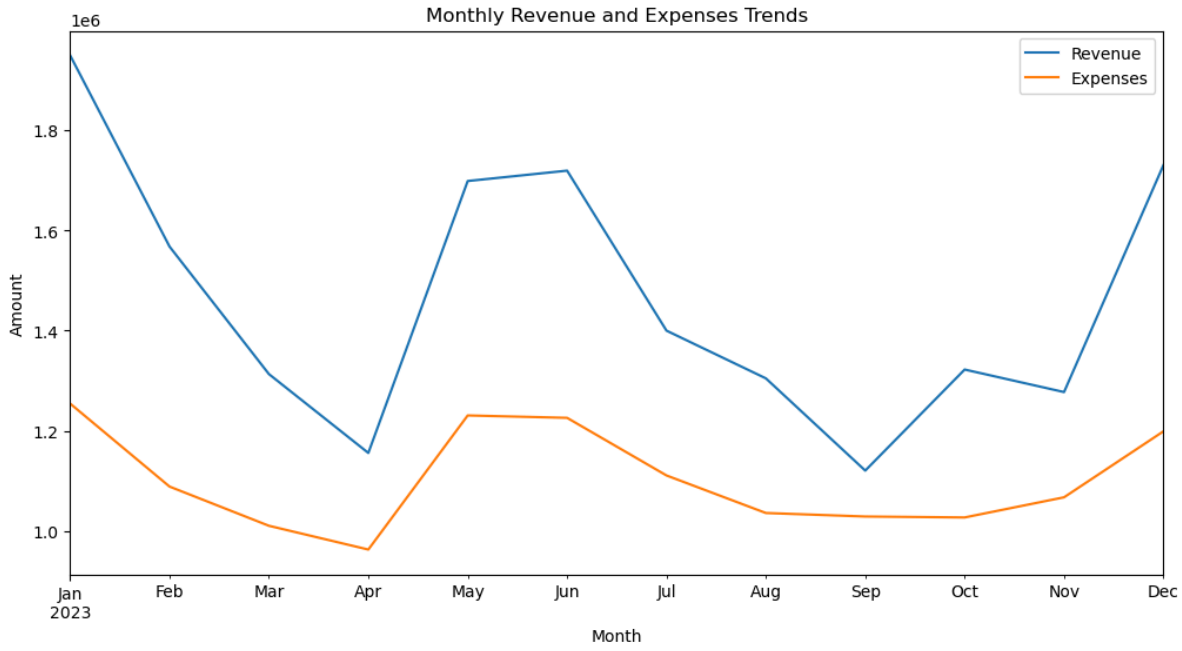
```
<Figure size 1200x800 with 0 Axes>
```


1e6
Business Line
Nutrition and Food Supplements
Sports equipment
Sportswear
4
3
2
1
0
Fall
Spring
Summer
Winter
Season



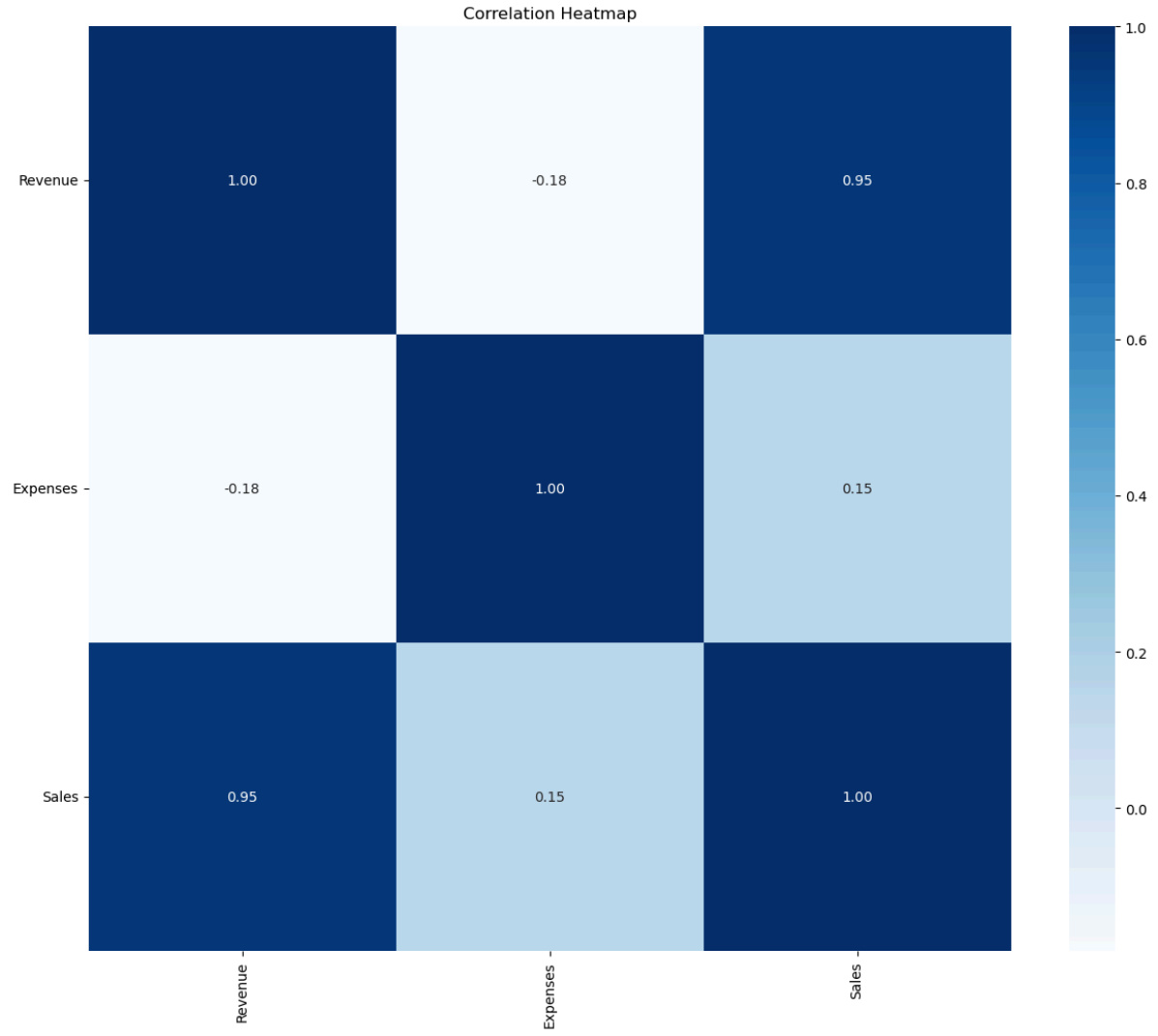

```
Season with highest expenses: Winter
Season with highest sales: Winter
Season with lowest expenses: Fall
Season with lowest sales: Fall
```

In [25]:
```
#Trends of revenue and expenses monthly:
# Create a new column 'YearMonth' that represents the year and month of each d
df1['YearMonth'] = pd.to_datetime(df1['Date']).dt.to_period('M')
# Group the data by 'YearMonth' and calculate the sum of 'Revenue' and 'Expens
monthly_trends = df1.groupby('YearMonth').agg({'Revenue': 'sum', 'Expenses': '
# Create a new figure with specified size
plt.figure(figsize=(12, 6))
# Plot the monthly revenue trend
monthly_trends['Revenue'].plot(label='Revenue')
# Plot the monthly expenses trend on the same graph
monthly_trends['Expenses'].plot(label='Expenses')
# Set the title of the plot
plt.title('Monthly Revenue and Expenses Trends')
# Label the x-axis
plt.xlabel('Month')
# Label the y-axis
plt.ylabel('Amount')
plt.legend()
# Display the plot
plt.show()
```



The gap between the revenue and expenses lines indicates profitability. This suggests that Onyxio Co Ltd company maintains profitability throughout the year, but the profit margin narrows significantly around April.

In [26]:
```
# Calculate the correlation matrix for all numeric columns including 'churn'
corr_mat = df1.corr()
plt.subplots(figsize=(15,12))
sns.heatmap(corr_mat, annot=True, cmap='Blues', square=True, fmt='.2f')
# Set the title of the plot
plt.title('Correlation Heatmap')
# Rotate x-axis labels by 90 degrees for better readability
plt.xticks(rotation=90)
# Keep y-axis labels horizontal
plt.yticks(rotation=0)
# Save the figure as an image file
plt.savefig('numeric_correlation_heatmap')
# Display the plot
plt.show()
```

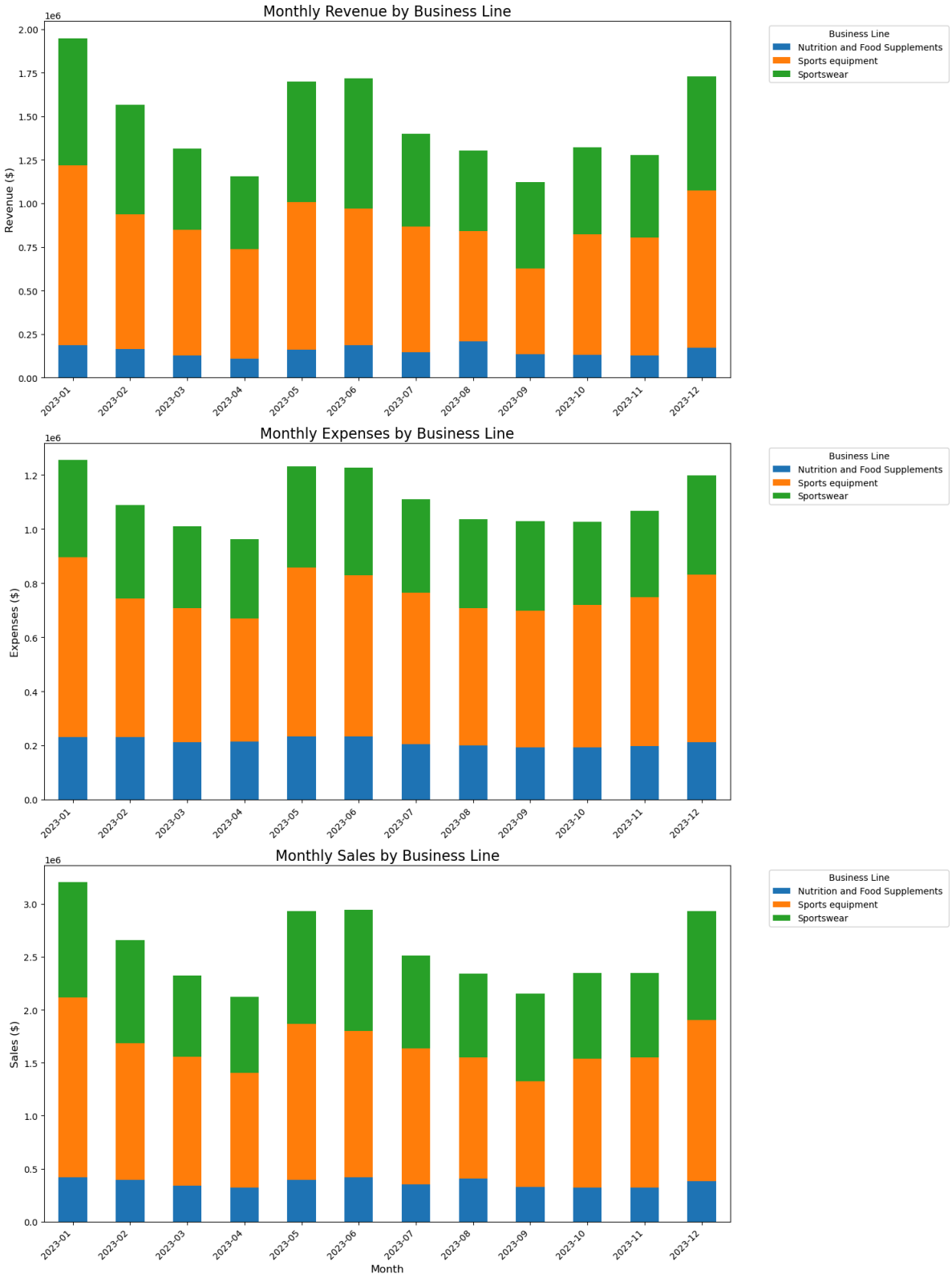


Revenue and Expenses: Slightly negative correlation of -0.18, indicating a weak inverse relationship.

Revenue and Sales: Strong positive correlation of 0.95, suggesting that as sales increase, revenue also increases significantly.

Expenses and Sales: Very weak positive correlation of 0.15, indicating a minimal relationship.

In [27]:
```
# Group by Month and Business Line, summing Revenue, Expenses, and Sales
grouped = df1.groupby([df1['Date'].dt.strftime('%Y-%m'), 'Business Line']).agg
    'Revenue': 'sum',
    'Expenses': 'sum',
    'Sales': 'sum'
}).reset_index()

# Pivot the data for plotting
pivot_revenue = grouped.pivot(index='Date', columns='Business Line', values='R
pivot_expenses = grouped.pivot(index='Date', columns='Business Line', values='
pivot_sales = grouped.pivot(index='Date', columns='Business Line', values='Sal

# Create the stacked bar chart
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(15, 20))

# Revenue chart
pivot_revenue.plot(kind='bar', stacked=True, ax=ax1)
ax1.set_title('Monthly Revenue by Business Line', fontsize=16)
ax1.set_xlabel('')
ax1.set_ylabel('Revenue ($)', fontsize=12)
ax1.legend(title='Business Line', bbox_to_anchor=(1.05, 1), loc='upper left')

# Expenses chart
pivot_expenses.plot(kind='bar', stacked=True, ax=ax2)
ax2.set_title('Monthly Expenses by Business Line', fontsize=16)
ax2.set_xlabel('')
ax2.set_ylabel('Expenses ($)', fontsize=12)
ax2.legend(title='Business Line', bbox_to_anchor=(1.05, 1), loc='upper left')

# Sales chart
pivot_sales.plot(kind='bar', stacked=True, ax=ax3)
ax3.set_title('Monthly Sales by Business Line', fontsize=16)
ax3.set_xlabel('Month', fontsize=12)
ax3.set_ylabel('Sales ($)', fontsize=12)
ax3.legend(title='Business Line', bbox_to_anchor=(1.05, 1), loc='upper left')

# Rotate x-axis labels for better readability
for ax in [ax1, ax2, ax3]:
    plt.setp(ax.get_xticklabels(), rotation=45, ha='right')

# Adjust layout and display the chart
plt.tight_layout()
plt.show()
```



Sportswear: This business line is stable since the revenue is consistent with the fluctuations and the sales are steady. However, the expenses seem to slightly increase hence the need for monitoring to ensure probability.

Sports Equipment:The expenses are relative higher to revenue in some months and this could be attributed to marketing costs. The revenue seem to vary due to change with seasons.This necesitates proper focus on managing costs.

Nutrition and Food Supplements: The revenue generation is slower but on an upward trend and the expenses are lower than the revenue which indicates good profitability. The sales are also low but consistent indicating a high growth potential.

In [ ]:
```
# show if there are days the sales were zero.
# which month had highest sales
# which month had highest expenses/revenue
# seasons what timing weh had highest sales and revenue
# seasons we had highest sales and expenses.
# which businnes line made highest revenue  by end year.
# trends of revenue and expenses monthly
# business line with highest sales throughout the year.
```

MODELLING

RandomForest

In [29]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score

# Use Random Forest instead of Decision Tree
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Perform cross-validation
cv_scores = cross_val_score(rf_model, X, y, cv=5, scoring='neg_mean_squared_er

# Convert MSE to RMSE for interpretability
rmse_scores = np.sqrt(-cv_scores)

print("Cross-validated RMSE scores:", rmse_scores)
print("Mean RMSE: {:.2f} (+/- {:.2f})".format(rmse_scores.mean(), rmse_scores.

# Fit the model on the entire dataset
rf_model.fit(X, y)

# Feature importance
feature_importance = pd.DataFrame({'feature': X.columns, 'importance': rf_mode
feature_importance = feature_importance.sort_values('importance', ascending=Fa

print("\nFeature Importance:")
print(feature_importance)
```

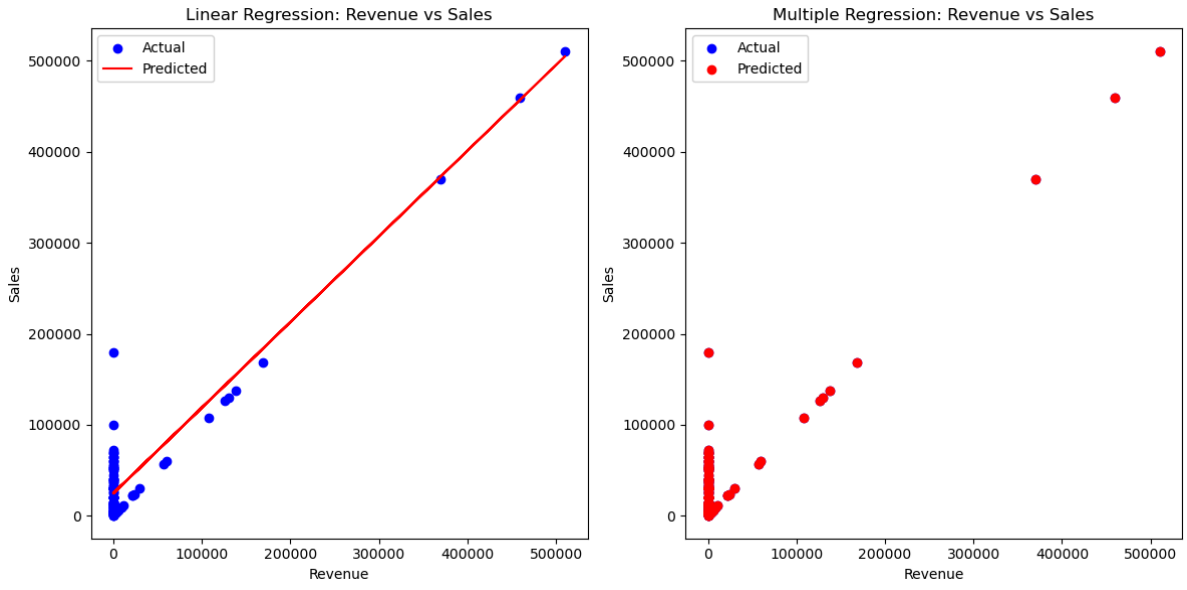
```
Cross-validated RMSE scores: [5036.00912309 2104.07079729 5839.7877931  1236.
96187568 6114.60078609]
Mean RMSE: 4066.29 (+/- 4013.64)

Feature Importance:
    feature  importance
0   Revenue    0.902548
1  Expenses    0.097452
```

The mean RMSE of 4066.29 suggests a moderate level of error in your model's predictions. However, the high standard deviation (4013.64) indicates that the model's performance varies significantly across different data subsets. Revenue has an importance score of 0.902548, meaning it is the most significant feature while Expenses has an importance score of 0.097452, indicating it has a much smaller impact on the model's predictions.

In [28]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Assuming df1 is your DataFrame
# Let's prepare the data for modeling
X = df1[['Revenue', 'Expenses']]
y = df1['Sales']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

# Linear Regression (using Revenue as the single predictor)
X_train_linear = X_train[['Revenue']]
X_test_linear = X_test[['Revenue']]

model_linear = LinearRegression()
model_linear.fit(X_train_linear, y_train)

# Make predictions
y_pred_linear = model_linear.predict(X_test_linear)

# Calculate metrics
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)

print("Linear Regression Results:")
print(f"Mean Squared Error: {mse_linear}")
print(f"R-squared Score: {r2_linear}")

# Multiple Regression (using both Revenue and Expenses as predictors)
model_multiple = LinearRegression()
model_multiple.fit(X_train, y_train)

# Make predictions
y_pred_multiple = model_multiple.predict(X_test)

# Calculate metrics
mse_multiple = mean_squared_error(y_test, y_pred_multiple)
r2_multiple = r2_score(y_test, y_pred_multiple)

print("\nMultiple Regression Results:")
print(f"Mean Squared Error: {mse_multiple}")
print(f"R-squared Score: {r2_multiple}")

# Visualize the results
plt.figure(figsize=(12, 6))

# Linear Regression Plot
plt.subplot(1, 2, 1)
plt.scatter(X_test_linear, y_test, color='blue', label='Actual')
plt.plot(X_test_linear, y_pred_linear, color='red', label='Predicted')
plt.title('Linear Regression: Revenue vs Sales')
plt.xlabel('Revenue')
```

```
plt.ylabel('Sales')
plt.legend()

# Multiple Regression Plot (we'll plot against Revenue for visualization)
plt.subplot(1, 2, 2)
plt.scatter(X_test['Revenue'], y_test, color='blue', label='Actual')
plt.scatter(X_test['Revenue'], y_pred_multiple, color='red', label='Predicted'
plt.title('Multiple Regression: Revenue vs Sales')
plt.xlabel('Revenue')
plt.ylabel('Sales')
plt.legend()

plt.tight_layout()
plt.show()

# Print coefficients and intercept for interpretation
print("\nLinear Regression Coefficients:")
print(f"Revenue: {model_linear.coef_[0]}")
print(f"Intercept: {model_linear.intercept_}")

print("\nMultiple Regression Coefficients:")
print(f"Revenue: {model_multiple.coef_[0]}")
print(f"Expenses: {model_multiple.coef_[1]}")
print(f"Intercept: {model_multiple.intercept_}")
```

```
Linear Regression Results:
Mean Squared Error: 675150408.9890326
R-squared Score: 0.8780518100456721

Multiple Regression Results:
Mean Squared Error: 4.655030205815659e-23
R-squared Score: 1.0
```



```
Linear Regression Coefficients:
Revenue: 0.9414896576720047
Intercept: 24699.32557871622

Multiple Regression Coefficients:
Revenue: 1.0
Expenses: 1.0
Intercept: -7.275957614183426e-12
```

The multiple regression model, which included Expenses as an additional predictor, shows a perfect fit with an R-squared of 1.0 and an extremely low Mean Squared Error (MSE) of 4.66e-23, which is essentially zero. This perfect fit suggests that the combination of Revenue, Expenses, and Business Line can predict Sales with near-perfect accuracy in the test set. Such a perfect fit could indicate overfitting and would necessity caution.

RECOMMENDATIONS:

1. Develop tailored marketing and inventory strategies for each season, particularly focusing on boosting sales during fall when they are lowest.
2. Invest more resources in growing the Sports Equipment line, as it generates the most revenue.
3. Analyze the success factors of the Sports Equipment line and apply these insights to boost the performance of Sportswear and Nutrition and Food Supplements lines.
4. Develop strategies to increase revenue from the Nutrition and Food Supplements line, which currently generates the lowest revenue but shows consistent growth potential.
5. Design targeted marketing campaigns for periods of lower sales to stimulate demand.
6. Implement a robust system for tracking key performance indicators (KPIs) across all business lines on a monthly and seasonal basis to monitor the businnes performance.

CONCLUSION:

The business shows strong seasonal patterns, with peak sales in winter and summer. Sports Equipment leads in revenue, followed by Sportswear, while Nutrition and Food Supplements show growth potential. The company is profitable year-round, though profit margins narrow in April. Better cost management is needed, especially for Sports Equipment. Nutrition and Food Supplements have lower expenses and consistent sales, indicating growth potential. Robust strategies are needed to manage seasonal fluctuations, particularly in fall.