DAP

September 20, 2024

1 Data Analyst Professional Practical Exam Submission

You can use any tool that you want to do your analysis and create visualizations. Use this template to write up your summary for submission.

You can use any markdown formatting you wish. If you are not familiar with Markdown, read the Markdown Guide before you start.

[2]:

```
# Import dependencies

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:

```
#Get data

sales = pd.read_csv('https://s3.amazonaws.com/talent-assets.datacamp.com/
 ↪product_sales.csv')
```

[ ]:

```
```

2 PRODUCT SALES REPORT

2.1 Introduction

Sales report of the performances of three sales channels: 'Email', 'Email + Call' and 'Call'.

What does the spread of the revenue look like overall? And for each method?

In the time under consideration, Pens and Printers sold a total of 151,270 products to 15,000 customers, making a total revenue of 1,308,138.01. After Data validation, Total revenue was $ 1404261.01, Total sales is 151270.0 respectively. At an average revenue of 93.93 per customer.

[4]:

```
sales.head
```

[4]:

```
<bound method NDFrame.head of        week  sales_method
customer_id  nb_sold  \
0         2         Email  2e72d641-95ac-497b-bbf8-4861764a7097       10
1         6  Email + Call  3998a98d-70f5-44f7-942e-789bb8ad2fe7       15
```

```
2         5          Call  d1de9884-8059-4065-b10f-86eef57e4a44       11
3         4         Email  78aa75a4-ffeb-4817-b1d0-2f030783c5d7       11
4         3         Email  10e6d446-10a5-42e5-8210-1b5438f70922        9
…      …           …                                    …      …
14995     4          Call  17267b41-d048-4346-8b90-7f787690a836       10
14996     5          Call  09e10d6f-4508-4b27-895e-4db11ce8302b       10
14997     1          Call  839653cb-68c9-48cb-a097-0a5a3b2b298b        7
14998     6          Call  e4dad70a-b23b-407c-8bd3-e32ea00fae17       13
14999     5  Email + Call  4e077235-7c17-4054-9997-7a890336a214       13

      revenue  years_as_customer  nb_site_visits          state
0         NaN                  0              24        Arizona
1      225.47                  1              28         Kansas
2       52.55                  6              26      Wisconsin
3         NaN                  3              25        Indiana
4       90.49                  0              28       Illinois
…        …                …             …            …
14995   50.82                  0              22   Pennsylvania
14996   52.33                  1              27         Kansas
14997   34.87                  4              22  West Virginia
14998   64.90                  2              27     New Jersey
14999     NaN                  4              25       Illinois

[15000 rows x 8 columns]>
```

[5]:

```
sales.dtypes
```

[5]:

```
week                   int64
sales_method          object
customer_id           object
nb_sold                int64
revenue              float64
years_as_customer      int64
nb_site_visits         int64
state                 object
dtype: object
```

[6]:

```
# Total revenue  & total sales
total_rev = np.round(np.sum(sales['revenue']), 2)
print(f'Total revenue is ${total_rev}')
total_num = np.round(np.sum(sales['nb_sold']), 2)
print(f'Total sales is {total_num}')
```

```
Total revenue is $1308138.01
Total sales is 151270
```

3 Data Validation

[7]:

```
# Check for missing values and data types
data_info = sales.info()
missing_values = sales.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   week               15000 non-null  int64
 1   sales_method       15000 non-null  object
 2   customer_id        15000 non-null  object
 3   nb_sold            15000 non-null  int64
 4   revenue            13926 non-null  float64
 5   years_as_customer  15000 non-null  int64
 6   nb_site_visits     15000 non-null  int64
 7   state              15000 non-null  object
dtypes: float64(1), int64(4), object(3)
memory usage: 937.6+ KB
```

3.1 Missing values

There are 1074 missing values in the ‘revenue’. To fill these values,since all the customers bought an item, Impute missing revenue based on the median revenue for each sales method was exhibited. SimpleImputer strategy was used to fill the missing values.

[8]:

```
# Identify missing values in revenue and check for invalid entries
missing_revenue_rows = sales[sales['revenue'].isnull()]

# Convert 'revenue' to numeric if necessary
sales['revenue'] = pd.to_numeric(sales['revenue'], errors='coerce')

data_info, missing_values, missing_revenue_rows.head()
```

[8]:

```
(None,
 week                    0
 sales_method            0
 customer_id             0
 nb_sold                 0
 revenue              1074
 years_as_customer       0
 nb_site_visits          0
 state                   0
 dtype: int64,
     week  sales_method                           customer_id  nb_sold  \
 0      2         Email  2e72d641-95ac-497b-bbf8-4861764a7097       10
 3      4         Email  78aa75a4-ffeb-4817-b1d0-2f030783c5d7       11
 16     2         Email  0f744f79-1588-4e0c-8865-fdaecc7f6dd4       10
 17     6  Email + Call  d10690f0-6f63-409f-a1da-8ab0e5388390       15
 28     5         Email  f64f8fd5-e9b7-4326-9f5d-ef283f14d7ad       12

     revenue  years_as_customer  nb_site_visits         state
 0        NaN                  0              24       Arizona
 3        NaN                  3              25       Indiana
 16       NaN                  6              30  Pennsylvania
 17       NaN                  0              24     Wisconsin
 28       NaN                  4              32       Florida  )
```

[9]:
```python
from sklearn.impute import SimpleImputer

# Step 1: Impute missing revenue based on the median revenue for each sales␣
 ↪method
# Create an imputer object for the 'revenue' column
revenue_imputer = SimpleImputer(strategy='median')
```

[10]:
```python
# Apply the imputer to the 'revenue' column
sales['revenue'] = revenue_imputer.fit_transform(sales[['revenue']])

# Step 3: Verify that no missing values remain
print(sales.info())
print(sales.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   week               15000 non-null  int64
 1   sales_method       15000 non-null  object
 2   customer_id        15000 non-null  object
 3   nb_sold            15000 non-null  int64
 4   revenue            15000 non-null  float64
 5   years_as_customer  15000 non-null  int64
 6   nb_site_visits     15000 non-null  int64
 7   state              15000 non-null  object
dtypes: float64(1), int64(4), object(3)
memory usage: 937.6+ KB
None
week                 0
sales_method         0
customer_id          0
nb_sold              0
revenue              0
years_as_customer    0
```

```
nb_site_visits       0
state                0
dtype: int64
```

[11]:
```python
# Total revenue  & total sales
total_rev = np.round(np.sum(sales['revenue']), 2)
print(f'Total revenue is ${total_rev}')
total_num = np.round(np.sum(sales['nb_sold']), 2)
print(f'Total sales is {total_num}')
```

```
Total revenue is $1404261.01
Total sales is 151270
```

3.2 Inconsistencies in label category

The spelling inconsistencies in the 'sales_method' column was addressed by replacing them with title case labels

[12]:
```python
# Step 1: Check unique values in 'sales_method' column
sales_method_unique = sales['sales_method'].unique()
print("Unique values in 'sales_method':", sales_method_unique)
```

```
Unique values in 'sales_method': ['Email' 'Email + Call' 'Call' 'em + call'
'email']
```

[13]:
```python
# Step 2: Check unique values in 'state' column
state_unique = sales['state'].unique()
print("Unique values in 'state':", state_unique)
```

```
Unique values in 'state': ['Arizona' 'Kansas' 'Wisconsin' 'Indiana' 'Illinois'
'Mississippi'
 'Georgia' 'Oklahoma' 'Massachusetts' 'Missouri' 'Texas' 'New York'
 'Maryland' 'California' 'Tennessee' 'Pennsylvania' 'North Dakota'
 'Florida' 'Michigan' 'North Carolina' 'Hawaii' 'Colorado' 'Louisiana'
 'Virginia' 'New Mexico' 'Arkansas' 'Alaska' 'Oregon' 'New Hampshire'
 'Ohio' 'New Jersey' 'Connecticut' 'Iowa' 'Montana' 'Washington'
 'Kentucky' 'Alabama' 'Nebraska' 'South Carolina' 'Minnesota'
 'South Dakota' 'Delaware' 'Maine' 'Utah' 'West Virginia' 'Vermont'
 'Rhode Island' 'Nevada' 'Idaho' 'Wyoming']
```

[14]:
```python
# Update inconsistent labelling

sales['sales_method'] = sales['sales_method'].str.replace('email', 'Email')

# Update inconsistent labelling

sales['sales_method'] = sales['sales_method'].str.replace('em \+ call', 'Email␣
  ↪+ Call')
```

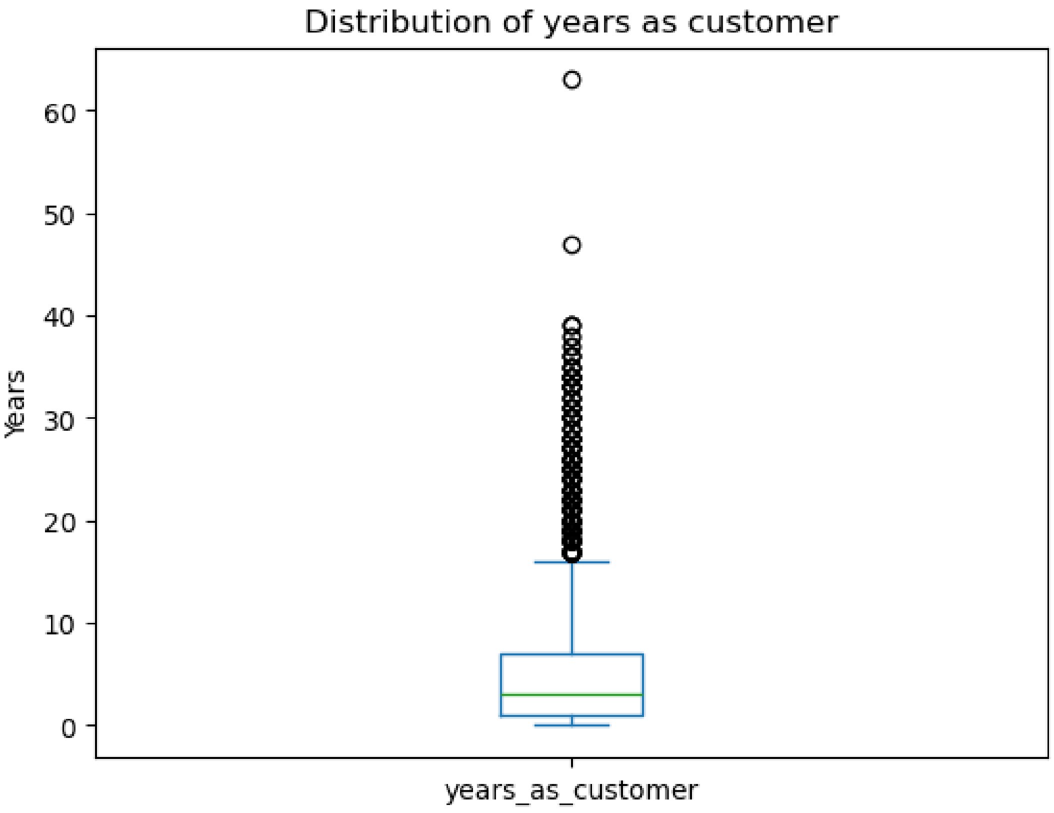
```
#Check
sales['sales_method'].unique()
```

```
[14]: array(['Email', 'Email + Call', 'Call', 'em + call'], dtype=object)
```

4 Outliers

The values in the 'years_as_customer' which were more than the years the company existed were replaced by the maximum years (40).

```
[15]: sales['years_as_customer'].plot(kind = 'box')

plt.ylabel('Years')
plt.title('Distribution of years as customer')
plt.show()
```



```
[16]: sales['years_as_customer'] = sales['years_as_customer'].apply(lambda x: 40 if x␣
↪> 40 else x)
```

5 Exploratory Analysis

Performance of Sales channel How many customers were there for each approach? 7466 customers were reached through the Email channels, 4962 recieved calls and only 2572 recieved calls and emails. On average of 93.62 of the total revenue, the email and call (170.88) combination performs better than other sales method which are email (96.57) and call (49.13) respetively.

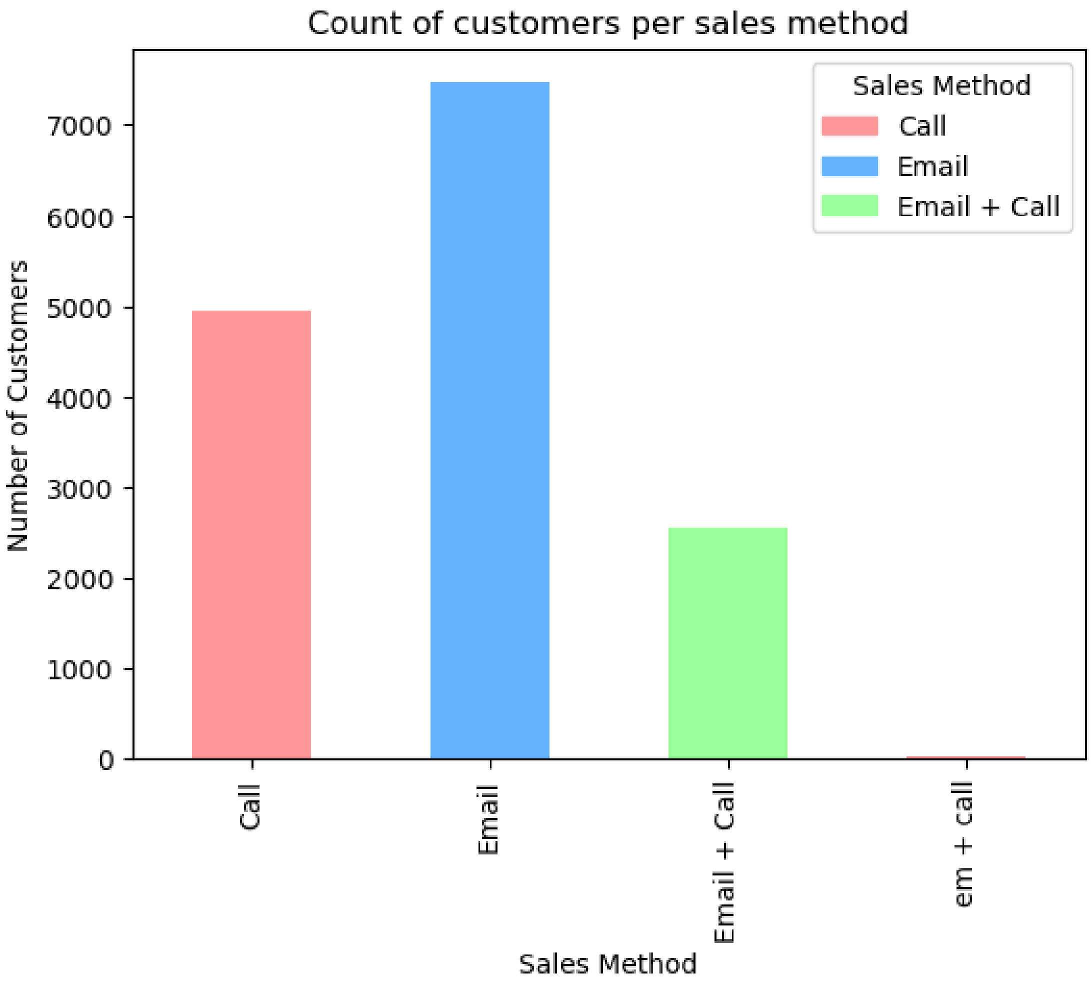
5.1 How many customers were there for each approach?

[17]:

```
# Count the unique customers for each sales method
customer_count = sales.groupby('sales_method')['customer_id'].nunique()
print(customer_count)



# Group and count
data = sales.groupby('sales_method')['customer_id'].count().reset_index()

# Define colors for each sales method
colors = ['#FF9999', '#66B3FF', '#99FF99']

# Create the bar plot without the label
ax = data.plot(
    x='sales_method',
    y='customer_id',
    kind='bar',
    color=colors,
    legend=False  # Disable the automatic legend
)

# Add title and labels
plt.title("Count of customers per sales method")
plt.xlabel('Sales Method')
plt.ylabel('Number of Customers')

# Create custom legend for sales methods only
handles = [plt.Rectangle((0,0),1,1, color=colors[i]) for i in␣
 ↪range(len(colors))]
labels = data['sales_method']
plt.legend(handles, labels, title='Sales Method')

# Display the chart
plt.show()
```

```
sales_method
Call            4962
Email           7466
Email + Call    2549
em + call         23
```

```
Name: customer_id, dtype: int64
```



5.2 What does the spread of the revenue look like overall? And for each method?

[18]:
```
# Summary statistics for overall revenue
overall_revenue_stats = sales['revenue'].describe()
print(overall_revenue_stats)

# Summary statistics for each sales method
method_revenue_stats = sales.groupby('sales_method')['revenue'].describe()
print(method_revenue_stats)
```

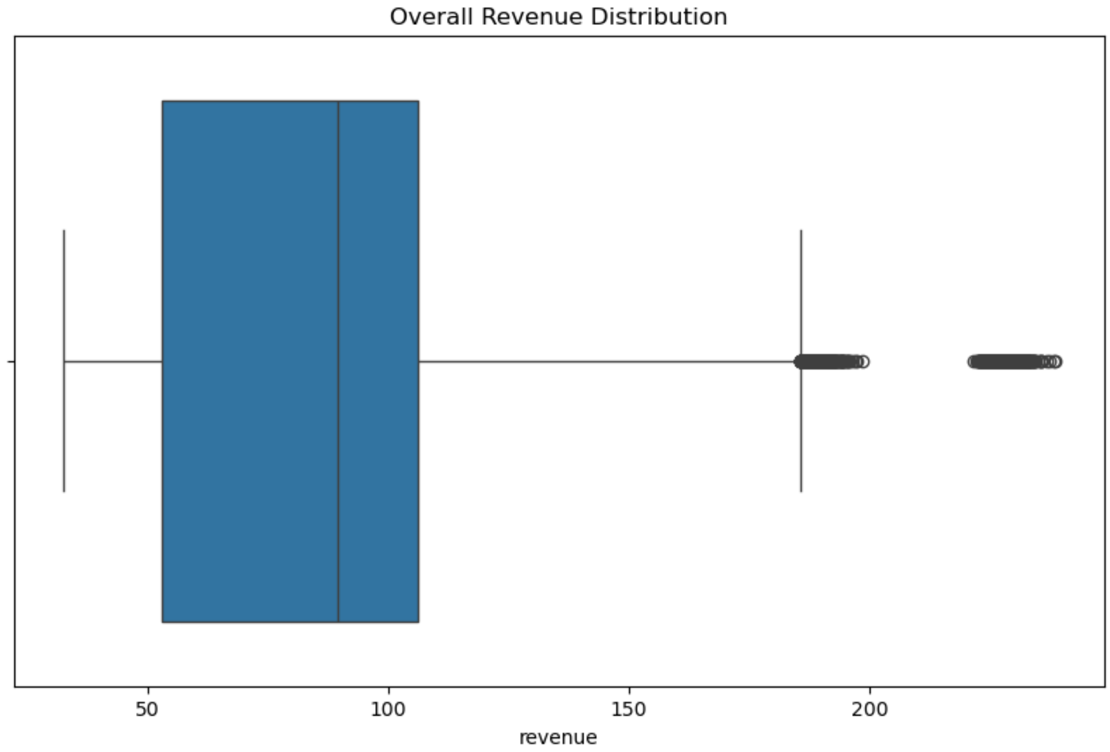
```
count    15000.000000
mean        93.617401
std         45.719775
min         32.540000
```

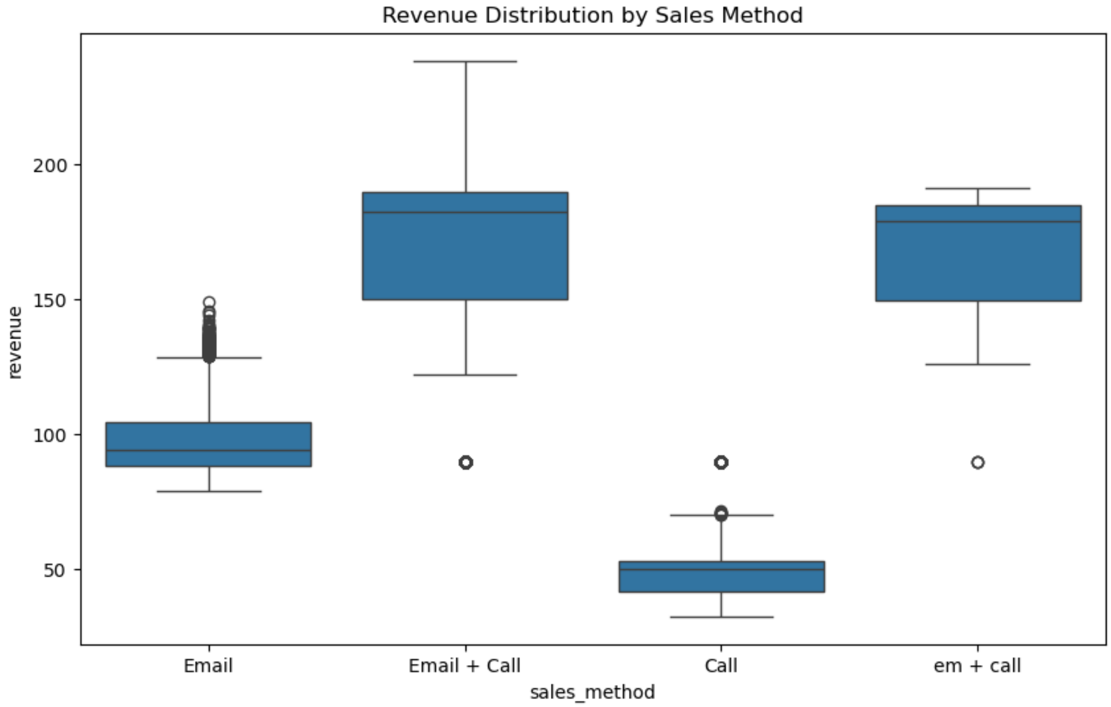
```
25%          53.040000
50%          89.500000
75%         106.070000
max         238.320000
Name: revenue, dtype: float64
               count        mean        std    min      25%      50%  \
sales_method
Call          4962.0   49.125955  11.539040  32.54   41.630   49.935
Email         7466.0   96.571903  10.974845  78.83   88.390   94.275
Email + Call  2549.0  170.951020  42.151660  89.50  149.840  182.160
em + call       23.0  162.523478  33.458694  89.50  149.425  178.720

                    75%    max
sales_method
Call           52.9775   89.50
Email         104.4600  148.97
Email + Call  189.5700  238.32
em + call     184.8450  190.90
```

5.3 Boxplot for Revenue Spread

[19]:

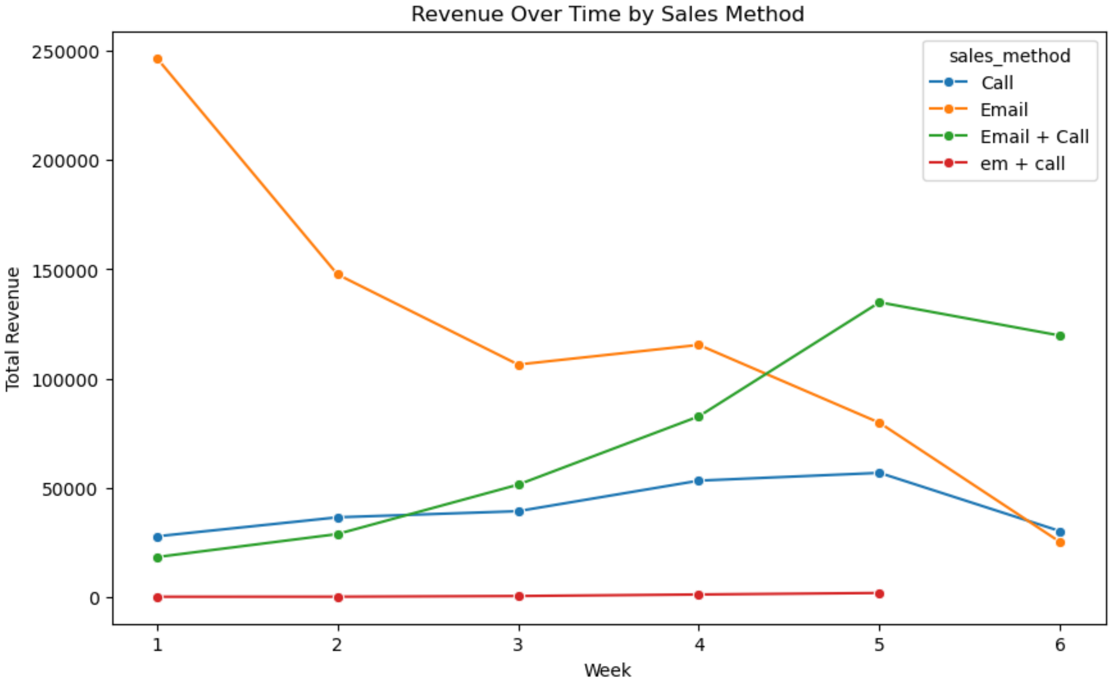
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Overall boxplot for revenue
plt.figure(figsize=(10, 6))
sns.boxplot(x='revenue', data=sales)
plt.title('Overall Revenue Distribution')
plt.show()

# Boxplot for each sales method
plt.figure(figsize=(10, 6))
sns.boxplot(x='sales_method', y='revenue', data=sales)
plt.title('Revenue Distribution by Sales Method')
plt.show()
```


Overall Revenue Distribution
50
100
150
200
revenue


Revenue Distribution by Sales Method
revenue
200
150
100
50
Email
Email + Call
Call
em + call
sales_method

Was there any difference in revenue over time for each of the methods? ## Grouping Revenue by Week and Sales Method

[20]:
```python
# Group revenue by week and sales method
revenue_over_time = sales.groupby(['week', 'sales_method'])['revenue'].sum().reset_index()

# Line plot to show revenue over time for each method
plt.figure(figsize=(10, 6))
sns.lineplot(x='week', y='revenue', hue='sales_method', data=revenue_over_time, marker='o')
plt.title('Revenue Over Time by Sales Method')
plt.xlabel('Week')
plt.ylabel('Total Revenue')
plt.show()
```



[21]:
```python
# Average revenue per customer for each method
avg_revenue_per_customer = sales.groupby('sales_method')['revenue'].mean()
print(avg_revenue_per_customer)
```

```
sales_method
Call             49.125955
Email            96.571903
Email + Call    170.951020
em + call       162.523478
```

```
Name: revenue, dtype: float64
```

5.4 Definition of a Metric for the Business to Monitor

To help Pens and Printers effectively monitor the performance of their sales methods for the new product line, a key metric can be defined based on efficiency and profitability. This metric should reflect both the revenue generated and the time/effort invested in each sales method.

5.5 How Should the Business Monitor This Metric?

5.5.1 Track Revenue and Time Spent Weekly:

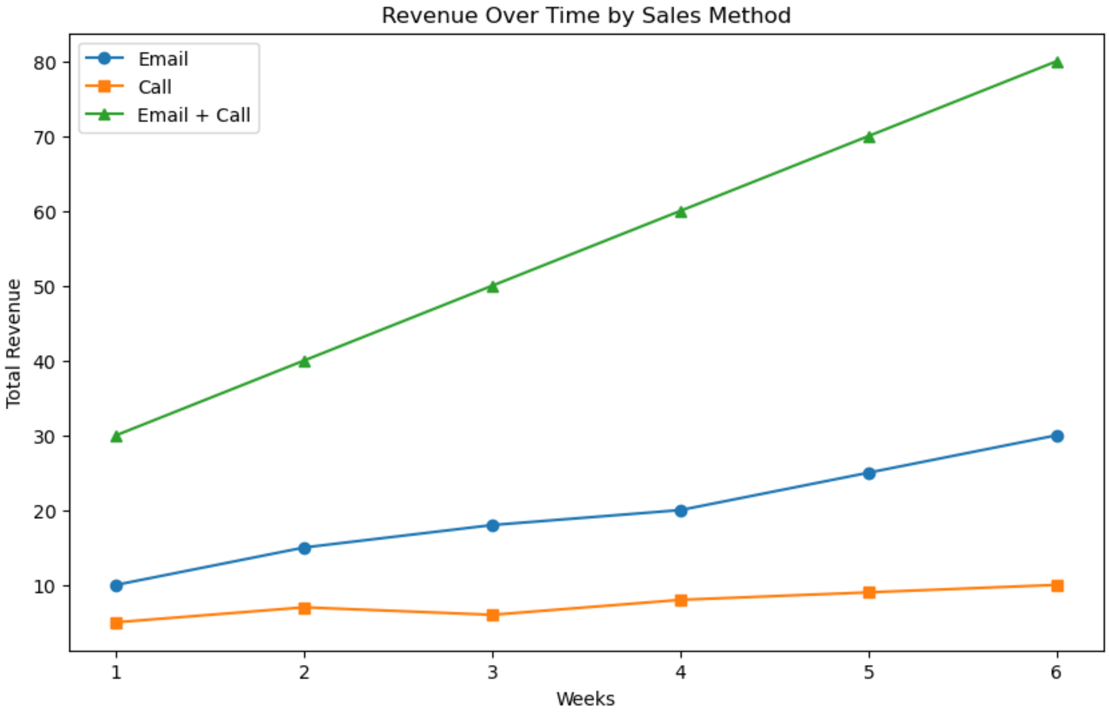
Weekly Monitoring: Track both the total revenue generated by each sales method and the total time spent by the sales team on that method.

Efficiency Comparison: Compare the revenue per minute across methods to see if more time-intensive methods yield proportionally higher returns.

5.6 Line chart for Revenue Over Time by Sales Method (Hypothetical weekly revenue for each method (for demonstration))

[22]:

```python
# Hypothetical weekly revenue for each method (for demonstration)
weeks = [1, 2, 3, 4, 5, 6]
email_revenue = [10, 15, 18, 20, 25, 30]
call_revenue = [5, 7, 6, 8, 9, 10]
email_call_revenue = [30, 40, 50, 60, 70, 80]

plt.figure(figsize=(10, 6))
plt.plot(weeks, email_revenue, label='Email', marker='o')
plt.plot(weeks, call_revenue, label='Call', marker='s')
plt.plot(weeks, email_call_revenue, label='Email + Call', marker='^')
plt.title('Revenue Over Time by Sales Method')
plt.xlabel('Weeks')
plt.ylabel('Total Revenue')
plt.legend()
plt.show()
```



5.7 Summary:

5.7.1 Metric: Revenue per minute of sales effort balances revenue with the time required for each method.

5.7.2 Monitoring: The business should track this weekly to assess the efficiency of each sales method and make data-driven decisions.

5.7.3 Initial Estimates: Based on current data, Email + Call seems to be the most efficient method in terms of time invested, but additional monitoring and adjustments may be required depending on evolving trends.

[ ]: