PENS AND PRINTERS DATA ANALYSIS REPORT

Pens and Printers is a company that was founded in 1984 and provides high quality office products to large organizations. Pens and Printers is a trusted provider of everything from pens and notebooks to desk chairs and monitors. The purpose of this analysis is to determine the success of the various sales methods than were employed in selling of a product that was recently launched. We will begin with data cleaning and validation then we will analyse our data and use visualizations to explain findings in our data. We will then give our findings and recommendations regarding the various sales methods employed.

DATA VALIDATION

Data validation is the process of ensuring that data is accurate, consistent, and meets predefined criteria or rules, thereby maintaining data integrity and reliability. We will seek to validate our data column by column before we answer the business question. This is to ensure that we have accurate findings. Otherwise, if our data were to be inaccurate then our findings would be useless and misleading.

We will begin by importing our data to our workspace. In this analysis we will be using python programming language and we'll use pandas, numpy, matplotlib and seaborn libraries to do our analyses.

In [37]:
```
# importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [38]:
```
# importing our data
product_sales = pd.read_csv('product_sales.csv')
```

We will now view the first few records of our data in order to get a rough idea of how our data looks like.

In [39]:
```
# viewing first five rows of our data
print(product_sales.head())
```

```
   week  sales_method  ... nb_site_visits      state
0     2         Email  ...             24    Arizona
1     6  Email + Call  ...             28     Kansas
2     5          Call  ...             26  Wisconsin
3     4         Email  ...             25    Indiana
4     3         Email  ...             28   Illinois

[5 rows x 8 columns]
```

We will now look at the fields in our data. The fields in our dataframe are the columns. They represent the different variables in our dataset. We will also check the data types contained in the columns. This will enable us to see how best we can approach our analysis.

In [40]:
```
# viewing the columns in our dataframe
print(product_sales.columns)
```

```
Index(['week', 'sales_method', 'customer_id', 'nb_sold', 'revenue',
       'years_as_customer', 'nb_site_visits', 'state'],
      dtype='object')
```

In [41]:
```
# viewing the data types of the columns
print(product_sales.dtypes)
```

```
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
```

Now that we have seen our fields, next up we will check through our data to check whether there are any null values in our data. It is important to deal with null values because they affect our analysis and the quality of our findings.

In [42]:
```
# checking for null values
print(product_sales.isnull().sum())
# Skip a line
print('\n')
# checking the number of rows in our data
no_of_rows = product_sales.shape[0]
print("total number of rows is ", no_of_rows)
```

```
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


total number of rows is  15000
```

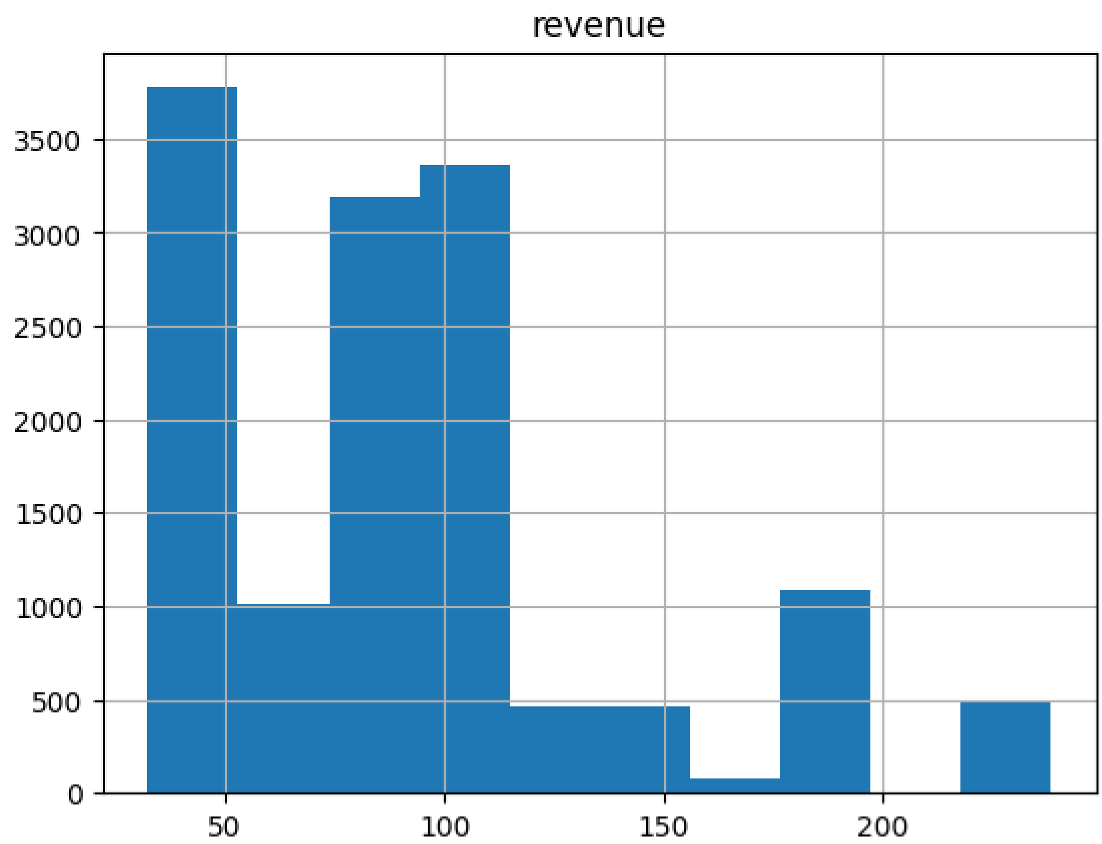
In [43]:
```
# calculating percentage of rows that have missing values
print((1074/no_of_rows)*100)
```

```
7.16
```

From the code above we see that the only column with missing values is the "revenue" column. We also see that rows missing values constitute 7.16% of our entire dataset. Because the missing values are relatively few, we will delete all the records that contain missing values.

In [44]:
```
# deleting the rows with missing values
product_sales.dropna(inplace = True)
# visualizing the revenue column using a histogram
product_sales.hist(column = 'revenue')
```

Out[44]:
```
array([[<AxesSubplot: title={'center': 'revenue'}>]], dtype=object)
```



As part of our data validation we are also going to check whether our "customer_id" column contains unique values only. It is important because this column holds the values that are unique identifiers for our records therefore it's paramount for the column to hold unique values only.

In [45]:
```
# checking whether the customer_id column contains unique values
unique_values = product_sales["customer_id"].duplicated().any()
print(unique_values)
```

```
False
```

The code above returns "False" meaning there are no duplicated values in the column.

We have dealt with the "revenue" column and the "customer_id" column. Next we'll look at the week column. Here we'll need to ensure the values here are non-negative and they are integers. This is because logically, we cannot have non negative weeks. We cannot also have weeks as fractions because the variable takes in the count of weeks since the product was launched and we know that counting is only done in whole numbers.

In [46]:
```
# checking whether the values are non-negative integers
non_negative_week = product_sales["week"] > 0
if non_negative_week.any() == "False":
    print("There exists a negative value")
else:
    print("All values are positive")
```

```
All values are positive
```

In [47]:
```
# checking whether the values are integers
integers = product_sales["week"].dtype == np.int64
print(integers)
```

```
True
```

From the lines of code above we see that the "week" column has non-negative integers only. Next, we'll move to the "sales_method" column. From our data information we know that we only have three sales methods: a) Email b) Call c) Email and Call So we'll check the data to see whether the unique values contained are these three.

In [48]:
```
# checking for unique values in the "sales_method" column
sales_method_unique = product_sales["sales_method"].unique()
sales_method_unique
```

Out[48]:
```
array(['Email + Call', 'Call', 'Email', 'em + call', 'email'],
      dtype=object)
```

From the result set above we see that the column has 5 unique values. We will replace 'email' with 'Email' and 'em + call' with 'Email + Call' and we will be good to go.

In [49]:
```
# replacing the values
product_sales["sales_method"] = product_sales["sales_method"].replace("email", "Em
product_sales["sales_method"] = product_sales["sales_method"].replace("em + call",
```

In [50]:
```
# checking whether we have replaced
product_sales["sales_method"].unique()
```

Out[50]:
```
array(['Email + Call', 'Call', 'Email'], dtype=object)
```

From the code above we can see that we have cleaned the "sales_method" column. Next we will move to the "nb_sold", "years_as_customer" and "nb_site_visits" columns. From our data information, we know that these columns should have non-negative integers as values because the are values that are counted.

In [51]:
```
# checking whether the columns are integers
nb_sold_int = product_sales["nb_sold"].dtype == np.int64
print(nb_sold_int)
years_as_customer_int = product_sales["years_as_customer"].dtype == np.int64
print(years_as_customer_int)
nb_site_visits_int = product_sales["nb_site_visits"].dtype == np.int64
print(nb_site_visits_int)
```

```
True
True
True
```

In [52]:
```
# checking whether the "nb_sold" column contains non-negative digits
product_sales["nb_sold"].any() < 0
```

Out[52]:
```
False
```

In [53]:
```
# checking whether the "nb_site_visits" column contains non-negative digits
product_sales["nb_site_visits"].any() < 0
```

Out[53]:
```
False
```

In [54]:
```
# checking whether the "years_as_customer" column contains non-negative digits
product_sales["years_as_customer"].any() < 0
```

Out[54]:
```
False
```

From the lines of code above we see that all these columns contain non-negative integers. The final column that we're going to validate is the "state" column. Here, we'll check whether all values have been inputed correctly and there are no unique values due to say, spelling errors or varying use of uppercase and lowercase letters.

In [55]:
```
# checking for uniqueness
unique_states = product_sales["state"].unique()
print(unique_states)
```

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

From the code above we see that there are no states that have been repeated. We therefore conclude that the column is okay and ready for analysis.

Seeing that we are done with validating and cleaning our data. We'll move to the next stage which is exploratory data analysis, commonly referred to as EDA. In this stage we will seek to answer the various questions raised by the sales rep. We will also describe our findings afterwards.

EXPLORATORY DATA ANALYSIS

EDA refers to the process of summarizing datasets in order to uncover insights and desribe the main characteristics of the data. Let's dive in.

1. Total number of customers for each sales approach

From the dataset above we know that the sales approach method that was employed is found in the "sales_method" column. We first determine the unique values in the column in order to get the different sales approaches.

In [56]:
```
# getting the different sales methods
unique_sales_methods = product_sales["sales_method"].unique()
print(unique_sales_methods)
```

```
['Email + Call' 'Call' 'Email']
```

We have seen that the 3 unique sales methods are:

1. Email
2. Email + Call
3. Call Now we will seek to determine the total number of customers that were approached using each of the three methods.

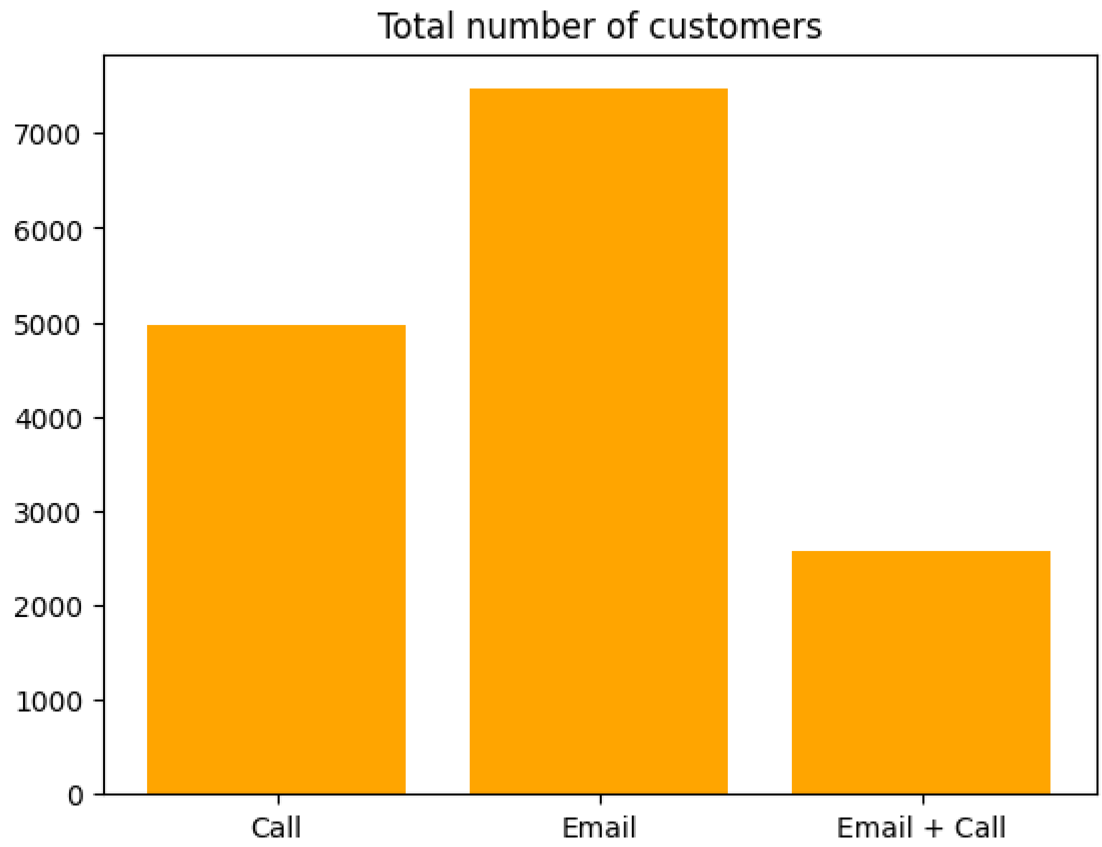
In [57]:
```
# determing total no of customers for each sales method
no_of_customers = product_sales.groupby("sales_method")["customer_id"].count()
```

```
print(no_of_customers)
```

```
sales_method
Call            4781
Email           6922
Email + Call    2223
Name: customer_id, dtype: int64
```

From the code above we see that the breakdown for each sales_method is: sales_method Call 4962 Email 7466 Email + Call 2572 We can use a bar graph to visualize the values

In [58]:
```
number_of_customers = [4962, 7466, 2572]
method_of_sales = ['Call', 'Email', 'Email + Call']
plt.bar(x=method_of_sales, height=number_of_customers, color='orange')
plt.title("Total number of customers")
plt.show()
```



2. Spread of revenue

The second question we are going to answer is what the spread of revenue looks like overall and for each method. Spread of revenue refers to a measure of how the revenues are dispersed in the dataset. It provides information about the variability or the extent to which the revenues deviate from the mean.

In [59]:
```
# variance of the revenue
revenue_var = product_sales["revenue"].var()
print("The variance of the revenue is ", revenue_var)
```

```
The variance of the revenue is  2250.1088478494826
```

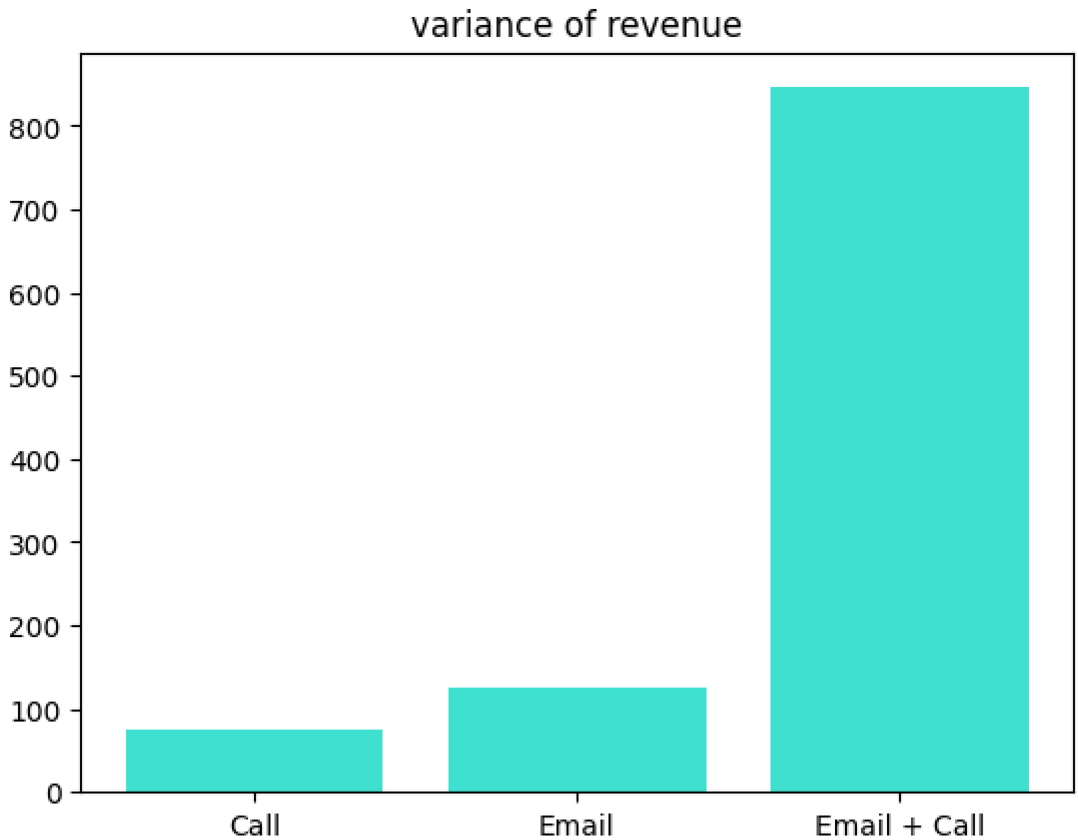
In [60]:
```
# variance of revenue for each sales method
var_rev_sales_method = product_sales.groupby("sales_method")["revenue"].var()
print(var_rev_sales_method)
```

```
sales_method
Call             74.130361
Email           125.674615
Email + Call    845.874652
Name: revenue, dtype: float64
```

From the code above we see that the variance of the revenue is 2250.1088478494826 And for each sales method we get: sales_method Call 74.130361 Email 125.674615 Email + Call 845.874652

In [61]:

```
var_sales_method = [74.13, 125.67, 845.87]
plt.bar(x=method_of_sales, height=var_sales_method, color='turquoise')
plt.title("variance of revenue")
plt.show()
```



We see that the "Email + Call" method has a very high variance whereas the "Call" method has the lowest variance.

3. Revenue over time for each method

Next up we're going to determine whether there was a difference in revenue over time for each of the 3 methods.

In [62]:

```
sales_call = product_sales[product_sales['sales_method'] == 'Call']
sales_email = product_sales[product_sales['sales_method'] == 'Email']
sales_call_email = product_sales[product_sales['sales_method'] == 'Email + Call']
rev_time_call = sales_call.groupby(['years_as_customer', 'sales_method'])['revenue
rev_time_email = sales_email.groupby(['years_as_customer', 'sales_method'])['reven
rev_time_call_email = sales_call_email.groupby(['years_as_customer', 'sales_method
rev_time_call = rev_time_call.droplevel(level=1)
rev_time_email = rev_time_email.droplevel(level=1)
rev_time_call_email = rev_time_call_email.droplevel(level=1)
```

```
rev_time_call.plot(label='call')
rev_time_email.plot(label='email')
rev_time_call_email.plot(label='call+email')
plt.title("Difference in Revenue over Time")
plt.ylabel("Revenue")
plt.legend()
plt.show()
```

From the code above and the graph we can see that "Call + Email" generates the highest average revenue whereas "Call" method generates the lowest revenue. However, we also see that over time the revenue from the "Call" method fluctuates slightly then begins to increase after about 38 years. For the "Email" method, the revenue also fluctuates slightly then becomes steady after about 38 years. For the "Call + Email" method, the revenue fluctuates with great varaibility then begins to steadily decrease after about 35 years.

Additional Findings

We will also seek to check and compare the revenue per state and also the revenue and products sold on a weekly basis after the products were launched.

In [63]:
```
# Revenue and products sold over time
products_sold = product_sales.groupby('week')[['revenue', 'nb_sold']].mean()
print(products_sold)
```

```
         revenue    nb_sold
week
1      78.012599   8.378038
2      85.260362   9.643564
3      81.425144   8.991582
4      98.734210  10.834939
5     107.650583  11.252747
6     148.824580  13.993613
```

In [64]:

```python
# revenue by state
state_revenue = product_sales.groupby('state')['revenue'].mean().sort_values(ascen
print(state_revenue)
```

```
state
South Dakota      104.755000
North Dakota      104.077200
Delaware          102.993333
Idaho             102.108305
Vermont           101.756667
Mississippi       100.372105
South Carolina    100.207371
West Virginia     100.025844
Utah               99.482609
Nevada             98.840722
Oregon             98.468832
Washington         98.376667
Nebraska           98.151395
New Mexico         97.937595
Connecticut        97.920060
Virginia           97.541532
Hawaii             97.397761
Alabama            96.949802
Rhode Island       96.295122
Oklahoma           96.204620
Wyoming            95.994062
Texas              95.847115
Michigan           95.332210
Louisiana          94.902207
Georgia            94.405239
Florida            94.009383
Kentucky           93.866980
Indiana            93.652202
Minnesota          93.651930
Wisconsin          93.572298
Massachusetts      93.428111
Pennsylvania       93.300814
Maryland           93.017633
California         92.605457
New York           92.594816
Kansas             92.573256
New Hampshire      92.451042
Ohio               92.328731
Arizona            92.207390
Colorado           92.195660
Arkansas           91.486271
Alaska             91.428286
Illinois           91.405122
New Jersey         90.863259
Iowa               90.767078
Missouri           90.417552
Tennessee          89.918734
North Carolina     89.344535
Maine              88.653167
Montana            83.379767
Name: revenue, dtype: float64
```

In [65]:

```
visits = product_sales.groupby('nb_site_visits')['revenue'].mean()
print(visits)
```

```
nb_site_visits
12     40.950000
13     51.330000
14     72.237143
15     69.407586
16     61.261899
17     64.007822
18     65.032885
19     70.344250
20     73.446878
21     76.645568
22     81.755383
23     84.758319
24     89.358714
25     91.802871
26     96.910413
27    102.403673
28    105.237561
29    113.684884
30    116.782135
31    123.935228
32    130.412564
33    126.551818
34    151.590250
35    162.301852
36    161.299231
37    186.245000
Name: revenue, dtype: float64
```

We shall include our conclusions in the findings below.

FINDINGS

1. Clients who were contacted via email and call brought in the highest revenue whereas clients contacted by call brought in the least amount of revenue.
2. Clients contacted by email and call demonstrated the largest variability in terms of revenue they brought in.
3. South Dakota recorded the highest average revenue whereas Montana had the lowest average revenue.
4. The revenue increased weekly since the products launch.
5. The quantity of products sold increased weekly since the products were launched.
6. The average revenue increases with an increased number of visits to the website by the customers.

Metric for business to monitor

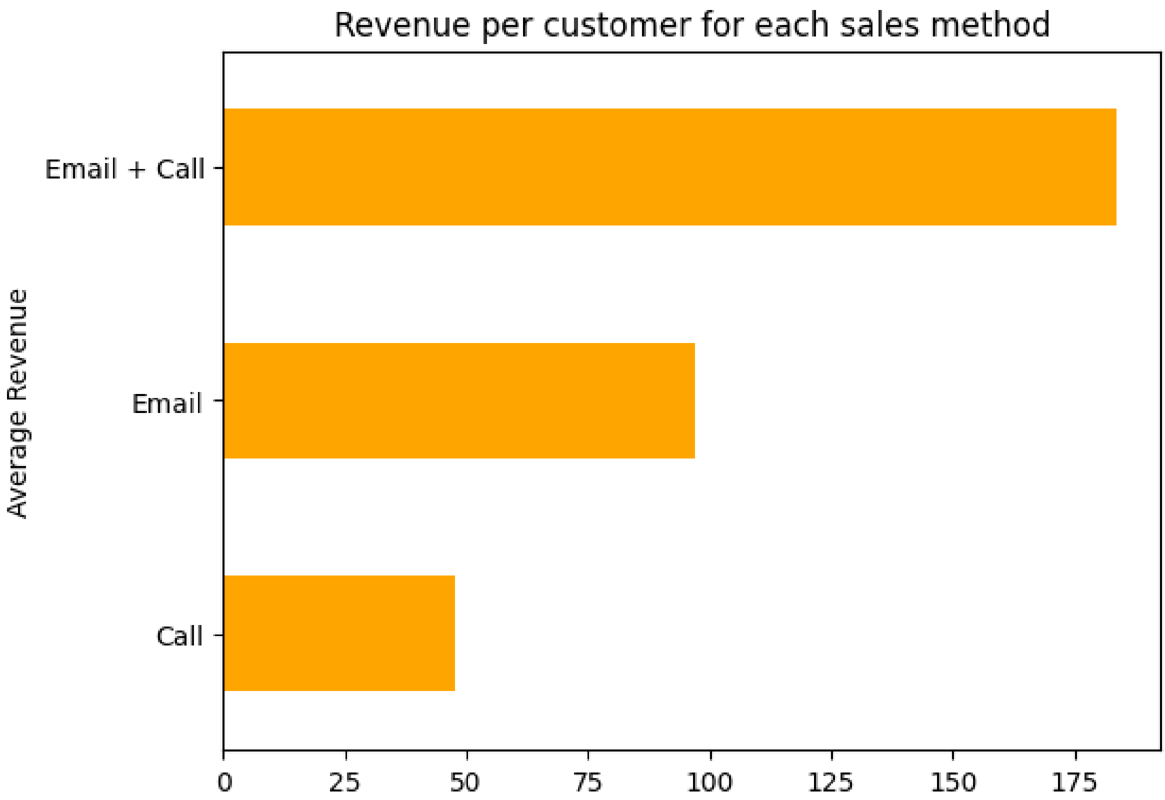
A metric is a quantifiable measurement used to assess and evaluate progress, performance or success in achieving specific business objectives or goals. The metric we will use is the average revenue per customer for each sales method. This metrics will seek to determine the average revenue generated by each customer in the context of the sales method that was used for them. This will also help us to monitor which sales method is the best and most effective. We will calculate it below.

In [66]:
```
total_revenue = product_sales.groupby('sales_method')['revenue'].sum()
print(total_revenue)
print(no_of_customers)
# revenue per customer for each sales method
rev_per_customer = total_revenue / no_of_customers
print(rev_per_customer)
```

```
sales_method
Call            227563.49
Email           672317.83
Email + Call    408256.69
Name: revenue, dtype: float64
sales_method
Call            4781
Email           6922
Email + Call    2223
Name: customer_id, dtype: int64
sales_method
Call             47.597467
Email            97.127684
Email + Call    183.651233
dtype: float64
```

From the code above we see that "Email + Call" has the highest revenue per customer, followed by "Email" and then "Call" method is last with nearly a quarter of the average revenue compared to "Email + Call"

In [67]:
```
# plotting a bar chart to compare the values
rev_per_customer.plot(kind = 'barh', color = 'orange')
plt.xticks(rotation = 0)
plt.title('Revenue per customer for each sales method')
plt.ylabel('Average Revenue')
plt.show()
```



The business should monitor this metric because it enables the business to see which sales method is most effective.

Recommendations that business should undertake

1. The company should focus more on emails and emails + calls and concentrate less on calls only. This is because emails + calls generate the largest revenue for the company and also, emails generate a moderate revenue with little variability.
2. The company should do further research in order to ascertain why sales are low in certain states like Maine, North Carolina, Montana and Tenessee. Could be due to competition or other reasons.
3. The company should seek to optimise their website to ensure maximum user utility seeing that the number of times customers visit the website in the last 6 months is positively correlated to the revenue accrued.