A/B Testing using Python

In [1]:
```
# Import the necessary libraries
import pandas as pd  # For data manipulation
import datetime  # For datetime-related functions
from datetime import date, timedelta  # Specific date and timedelta handling
import plotly.graph_objects as go  # For creating interactive graphs
import plotly.express as px  # A high-level interface for creating Plotly graphs
import plotly.io as pio  # For configuring Plotly templates

# Set the default Plotly template to "plotly_white" for a white background
pio.templates.default = "plotly_white"

# Load data from the "control_group.csv" and "test_group.csv" files
control_data = pd.read_csv("C:/Users/Kingsley Mills/Desktop/pretech/DS PROJECTS/AB Testing/control_group.csv", sep=";")
test_data = pd.read_csv("C:/Users/Kingsley Mills/Desktop/pretech/DS PROJECTS/AB Testing/test_group.csv", sep=";")
```

In [2]:
```
print(control_data.head())
```

```
      Campaign Name       Date  Spend [USD]  # of Impressions     Reach  \
0  Control Campaign  1.08.2019         2280           82702.0   56930.0
1  Control Campaign  2.08.2019         1757          121040.0  102513.0
2  Control Campaign  3.08.2019         2343          131711.0  110862.0
3  Control Campaign  4.08.2019         1940           72878.0   61235.0
4  Control Campaign  5.08.2019         1835               NaN       NaN

   # of Website Clicks  # of Searches  # of View Content  # of Add to Cart  \
0               7016.0         2290.0             2159.0            1819.0
1               8110.0         2033.0             1841.0            1219.0
2               6508.0         1737.0             1549.0            1134.0
3               3065.0         1042.0              982.0            1183.0
4                  NaN            NaN                NaN               NaN

   # of Purchase
0         618.0
1         511.0
2         372.0
3         340.0
4           NaN
```

In [3]:
```
print(test_data.head())
```

```
   Campaign Name       Date  Spend [USD]  # of Impressions  Reach  \
0  Test Campaign  1.08.2019         3008             39550  35820
1  Test Campaign  2.08.2019         2542            100719  91236
2  Test Campaign  3.08.2019         2365             70263  45198
3  Test Campaign  4.08.2019         2710             78451  25937
4  Test Campaign  5.08.2019         2297            114295  95138

   # of Website Clicks  # of Searches  # of View Content  # of Add to Cart  \
0                 3038           1946               1069               894
1                 4657           2359               1548               879
2                 7885           2572               2367              1268
3                 4216           2216               1437               566
4                 5863           2106                858               956

   # of Purchase
0            255
1            677
2            578
3            340
4            768
```

Data Preparation

In [4]:

```
# Rename the columns of the control_data DataFrame
control_data.columns = ["Campaign Name", "Date", "Amount Spent",
                        "Number of Impressions", "Reach", "Website Clicks",
                        "Searches Received", "Content Viewed", "Added to Cart",
                        "Purchases"]

# Rename the columns of the test_data DataFrame
test_data.columns = ["Campaign Name", "Date", "Amount Spent",
                        "Number of Impressions", "Reach", "Website Clicks",
                        "Searches Received", "Content Viewed", "Added to Cart",
                        "Purchases"]
```

In [5]:

```
# Check for missing values (NaN or null) in each column of the control_data DataFrame
missing_values_count = control_data.isnull().sum()

# Print the count of missing values for each column
print(missing_values_count)
```

```
Campaign Name           0
Date                    0
Amount Spent            0
Number of Impressions    1
```

```
Reach                    1
Website Clicks           1
Searches Received        1
Content Viewed           1
Added to Cart            1
Purchases                1
dtype: int64
```

In [6]:

```python
# Check for missing values (NaN or null) in each column of the test_data DataFrame
missing_values_count = test_data.isnull().sum()

# Print the count of missing values for each column
print(missing_values_count)
```

```
Campaign Name            0
Date                     0
Amount Spent             0
Number of Impressions    0
Reach                    0
Website Clicks           0
Searches Received        0
Content Viewed           0
Added to Cart            0
Purchases                0
dtype: int64
```

In [7]:

```python
# Fill missing values in the "Number of Impressions" column with the mean value of that column
control_data["Number of Impressions"].fillna(value=control_data["Number of Impressions"].mean(), inplace=True)

# Fill missing values in the "Reach" column with the mean value of that column
control_data["Reach"].fillna(value=control_data["Reach"].mean(), inplace=True)

# Fill missing values in the "Website Clicks" column with the mean value of that column
control_data["Website Clicks"].fillna(value=control_data["Website Clicks"].mean(), inplace=True)

# Fill missing values in the "Searches Received" column with the mean value of that column
control_data["Searches Received"].fillna(value=control_data["Searches Received"].mean(), inplace=True)

# Fill missing values in the "Content Viewed" column with the mean value of that column
control_data["Content Viewed"].fillna(value=control_data["Content Viewed"].mean(), inplace=True)

# Fill missing values in the "Added to Cart" column with the mean value of that column
control_data["Added to Cart"].fillna(value=control_data["Added to Cart"].mean(), inplace=True)

# Fill missing values in the "Purchases" column with the mean value of that column
control_data["Purchases"].fillna(value=control_data["Purchases"].mean(), inplace=True)
```

In [8]:
```
# Merge control_data and test_data DataFrames using an outer join on the "Date" column
ab_data = control_data.merge(test_data, how="outer").sort_values(["Date"])

# Reset the index of the merged DataFrame and drop the old index
ab_data = ab_data.reset_index(drop=True)

# Print the first few rows of the merged DataFrame
print(ab_data.head())
```

```
C:\Users\Kingsley Mills\anaconda3\lib\site-packages\pandas\core\reshape\merge.py:1203: UserWarning: You are merging on int and float columns where the float values are not equal to their int representation
  warnings.warn(
      Campaign Name        Date  Amount Spent  Number of Impressions    Reach  \
0  Control Campaign   1.08.2019          2280                82702.0  56930.0
1     Test Campaign   1.08.2019          3008                39550.0  35820.0
2     Test Campaign  10.08.2019          2790                95054.0  79632.0
3  Control Campaign  10.08.2019          2149               117624.0  91257.0
4     Test Campaign  11.08.2019          2420                83633.0  71286.0

   Website Clicks  Searches Received  Content Viewed  Added to Cart  Purchases
0          7016.0             2290.0          2159.0         1819.0      618.0
1          3038.0             1946.0          1069.0          894.0      255.0
2          8125.0             2312.0          1804.0          424.0      275.0
3          2277.0             2475.0          1984.0         1629.0      734.0
4          3750.0             2893.0          2617.0         1075.0      668.0
```

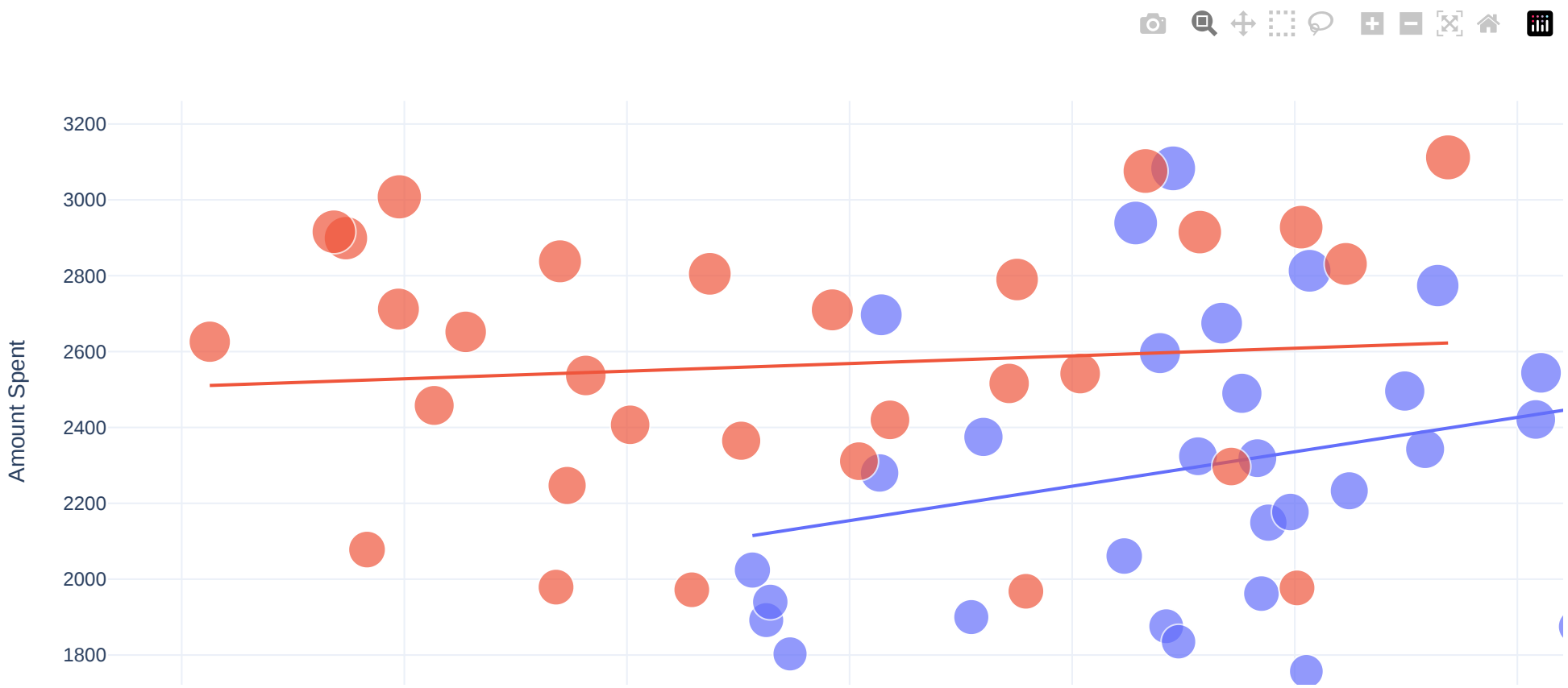
In [9]:
```
# Count the occurrences of each unique campaign name in the "Campaign Name" column
campaign_name_counts = ab_data["Campaign Name"].value_counts()

# Print the frequency of each campaign name
print(campaign_name_counts)
```

```
Control Campaign    30
Test Campaign       30
Name: Campaign Name, dtype: int64
```

I will begin by examining the relationship between the number of impressions received from both campaigns and the respective amounts spent on those campaigns.

In [10]:
```
# Create a scatter plot using Plotly Express
figure = px.scatter(data_frame=ab_data,  # The DataFrame to use for the plot
                    x="Number of Impressions",  # X-axis variable
                    y="Amount Spent",  # Y-axis variable
                    size="Amount Spent",  # Size of data points based on "Amount Spent"
                    color="Campaign Name",  # Color data points by "Campaign Name"
```

```
                    trendline="ols"  # Add an ordinary least squares (OLS) trendline
)

# Show the plot
figure.show()
```

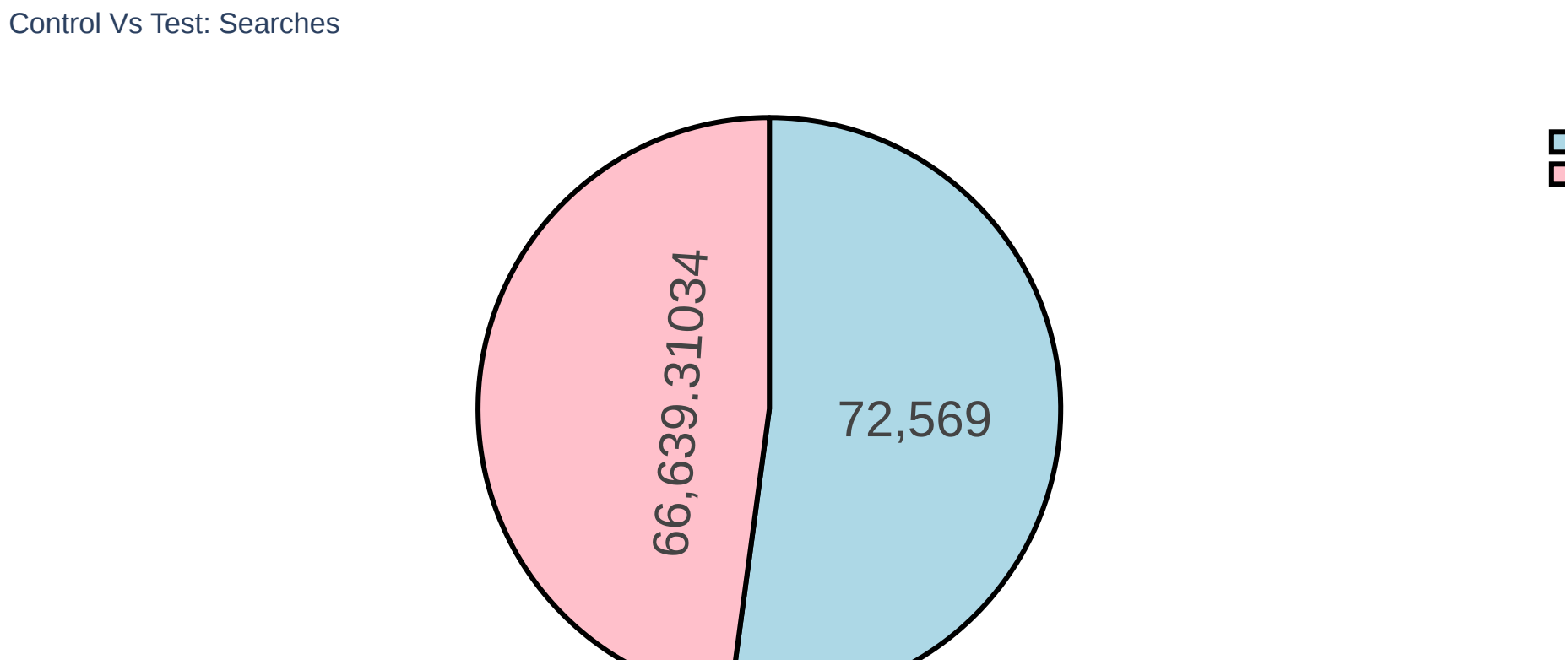


The control campaign yielded a higher number of impressions relative to the amount spent on both campaigns. Now, let's shift our focus to the number of searches conducted on the website from both campaigns.

In [11]:

```
# A pie chart to compare the total searches received in the control and test campaigns.

label = ["Total Searches from Control Campaign",
         "Total Searches from Test Campaign"]
counts = [sum(control_data["Searches Received"]),
```

```
          sum(test_data["Searches Received"])]
colors = ['pink','lightblue']
fig = go.Figure(data=[go.Pie(labels=label, values=counts)])
fig.update_layout(title_text='Control Vs Test: Searches')
fig.update_traces(hoverinfo='label+percent', textinfo='value',
                  textfont_size=30,
                  marker=dict(colors=colors,
                              line=dict(color='black', width=3)))
fig.show()
```

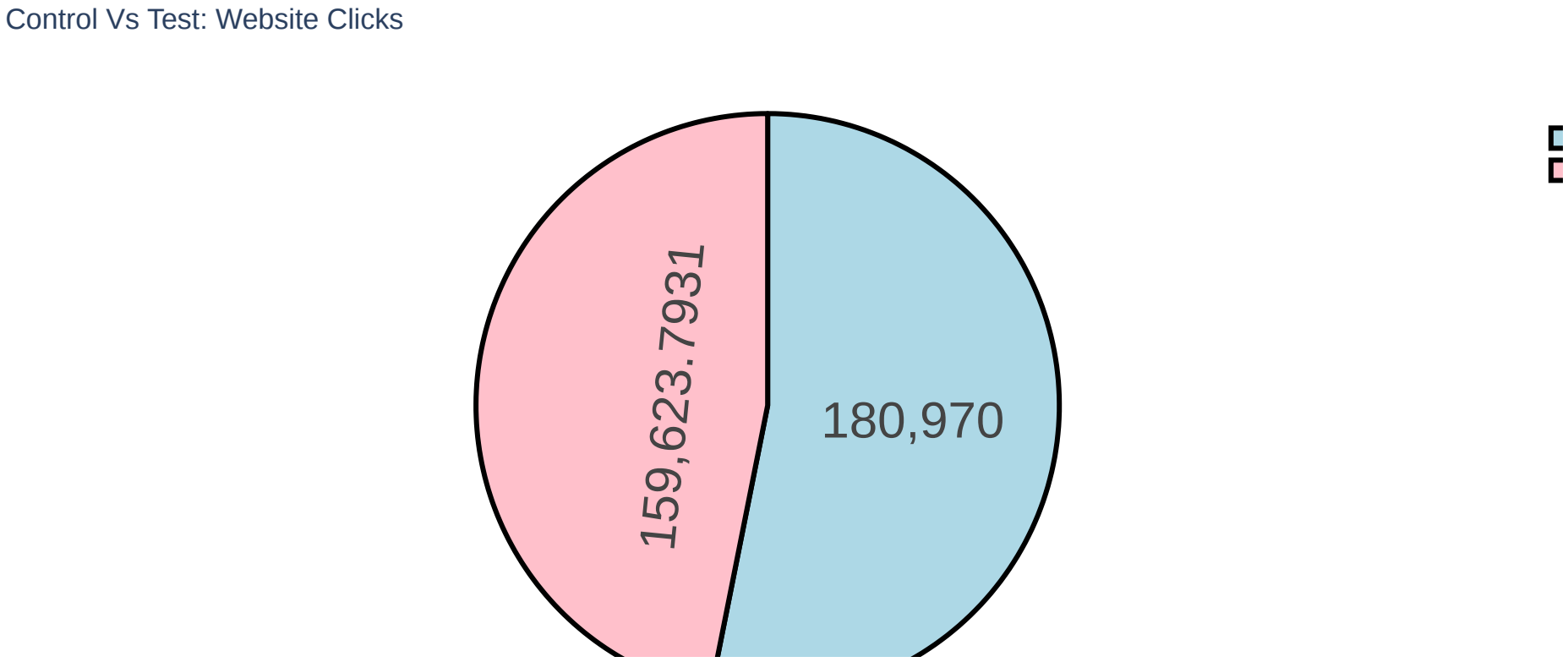


The test campaign led to a higher volume of searches on the website. Now, let's turn our attention to the number of website clicks from both campaigns.

In [12]:

```
# A pie chart to compare the total website clicks in the control and test campaigns.
```

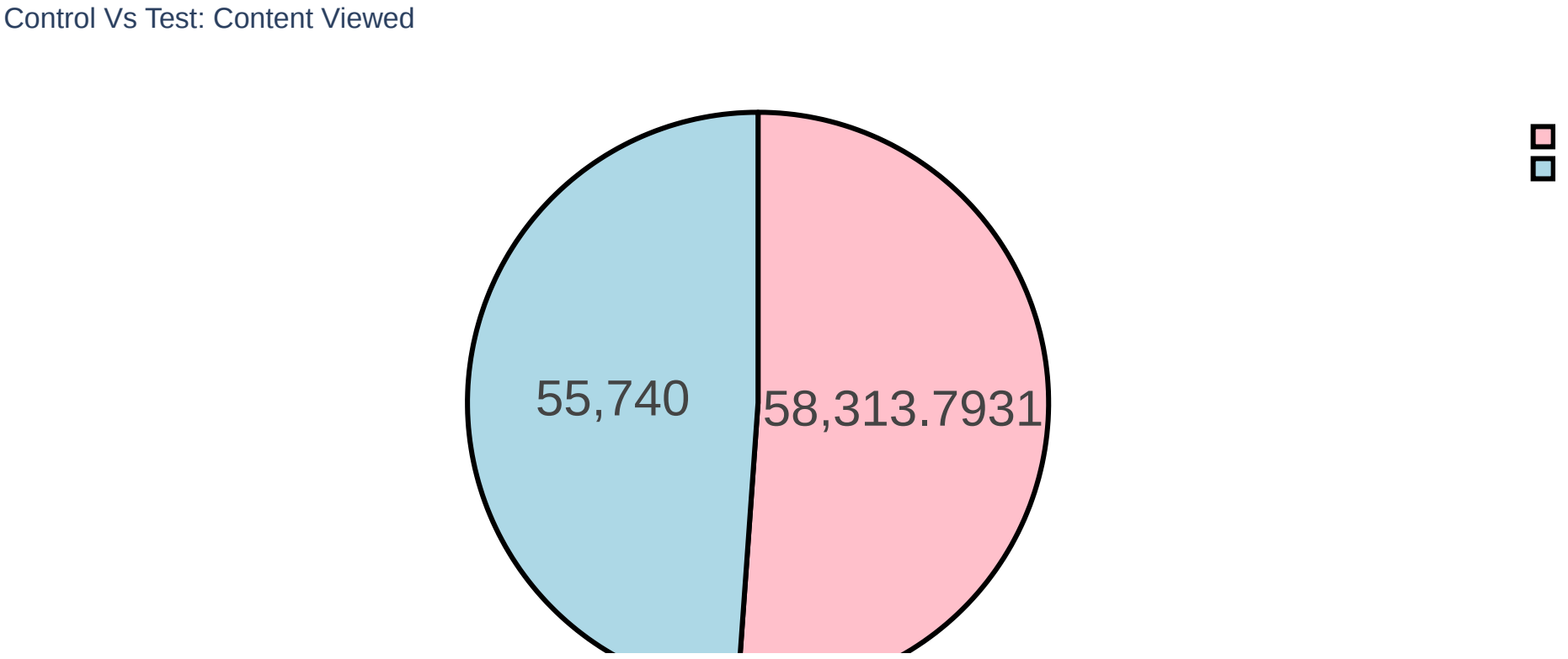
```
label = ["Website Clicks from Control Campaign",
         "Website Clicks from Test Campaign"]
counts = [sum(control_data["Website Clicks"]),
          sum(test_data["Website Clicks"])]
colors = ['pink','lightblue']
fig = go.Figure(data=[go.Pie(labels=label, values=counts)])
fig.update_layout(title_text='Control Vs Test: Website Clicks')
fig.update_traces(hoverinfo='label+percent', textinfo='value',
                  textfont_size=30,
                  marker=dict(colors=colors,
                              line=dict(color='black', width=3)))
fig.show()
```



The test campaign emerges as the winner in terms of the number of website clicks. Now, let's proceed to examine the amount of content viewed after visitors reached the website from both campaigns.

In [13]:

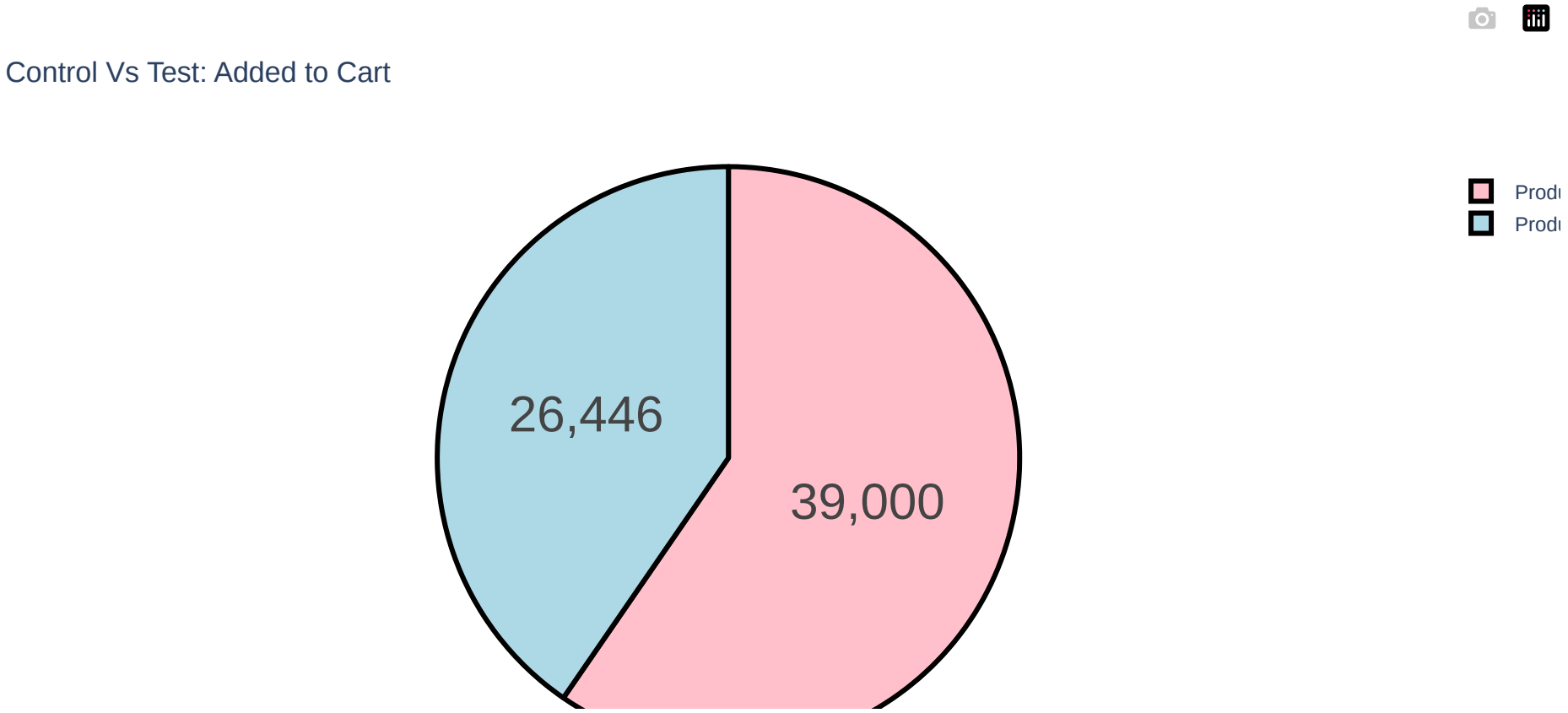
```
#  A pie chart to compare the total content viewed in the control and test campaigns.

label = ["Content Viewed from Control Campaign",
         "Content Viewed from Test Campaign"]
counts = [sum(control_data["Content Viewed"]),
          sum(test_data["Content Viewed"])]
colors = ['pink','lightblue']
fig = go.Figure(data=[go.Pie(labels=label, values=counts)])
fig.update_layout(title_text='Control Vs Test: Content Viewed')
fig.update_traces(hoverinfo='label+percent', textinfo='value',
                  textfont_size=30,
                  marker=dict(colors=colors,
                              line=dict(color='black', width=3)))
fig.show()
```



The audience of the control campaign observed a greater amount of content compared to the test campaign. Although the difference is not substantial, considering the relatively low website clicks in the control campaign, its engagement on the website surpasses that of the test campaign. Now, let's shift our focus to the number of products added to the cart from both campaigns.

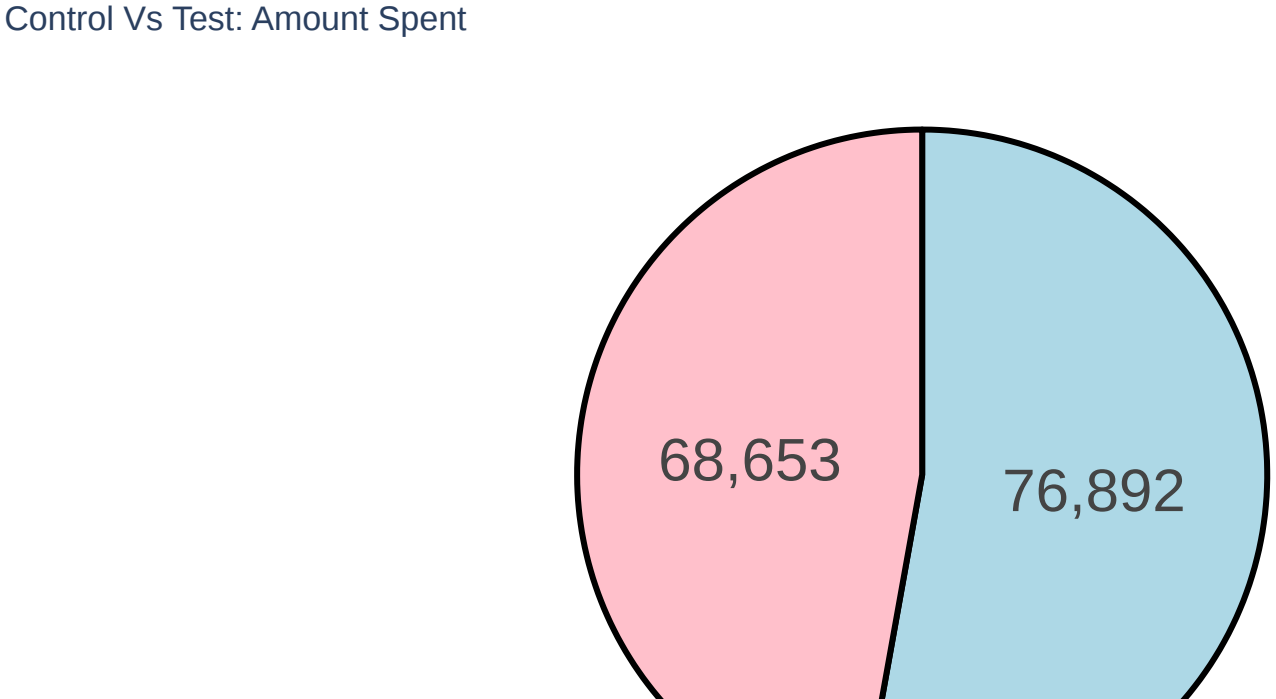
In [14]:

```
# A pie chart to compare the total products added to the cart in the control and test campaigns.

label = ["Products Added to Cart from Control Campaign",
         "Products Added to Cart from Test Campaign"]
counts = [sum(control_data["Added to Cart"]),
          sum(test_data["Added to Cart"])]
colors = ['pink','lightblue']
fig = go.Figure(data=[go.Pie(labels=label, values=counts)])
fig.update_layout(title_text='Control Vs Test: Added to Cart')
fig.update_traces(hoverinfo='label+percent', textinfo='value',
                  textfont_size=30,
                  marker=dict(colors=colors,
                              line=dict(color='black', width=3)))
fig.show()
```



Despite having fewer website clicks, the control campaign managed to accumulate more products added to the cart. Now, let's examine the amount spent on both campaigns.

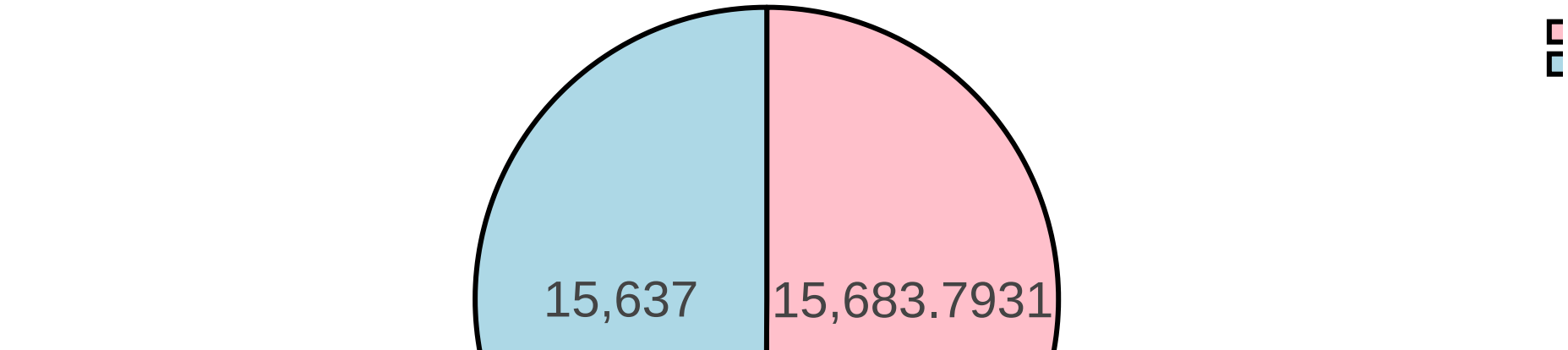
In [15]:
```
# A pie chart to compare the total amount spent in the control and test campaigns.

label = ["Amount Spent in Control Campaign",
         "Amount Spent in Test Campaign"]
counts = [sum(control_data["Amount Spent"]),
          sum(test_data["Amount Spent"])]
colors = ['pink','lightblue']
fig = go.Figure(data=[go.Pie(labels=label, values=counts)])
fig.update_layout(title_text='Control Vs Test: Amount Spent')
fig.update_traces(hoverinfo='label+percent', textinfo='value',
                  textfont_size=30,
                  marker=dict(colors=colors,
                              line=dict(color='black', width=3)))
fig.show()
```

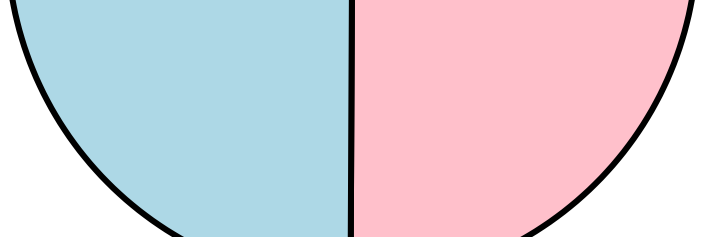


The amount spent on the test campaign surpasses that of the control campaign. However, considering that the control campaign generated more content views and more products in the cart, it appears that the control campaign is more efficient than the test campaign. Now, let's proceed to examine the purchases made by both campaigns.

In [16]:

```
# A pie chart to compare the total purchases made in the control and test campaigns.

label = ["Purchases Made by Control Campaign",
         "Purchases Made by Test Campaign"]
counts = [sum(control_data["Purchases"]),
          sum(test_data["Purchases"])]
colors = ['pink','lightblue']
fig = go.Figure(data=[go.Pie(labels=label, values=counts)])
fig.update_layout(title_text='Control Vs Test: Purchases')
fig.update_traces(hoverinfo='label+percent', textinfo='value',
                  textfont_size=30,
                  marker=dict(colors=colors,
                              line=dict(color='black', width=3)))
fig.show()
```



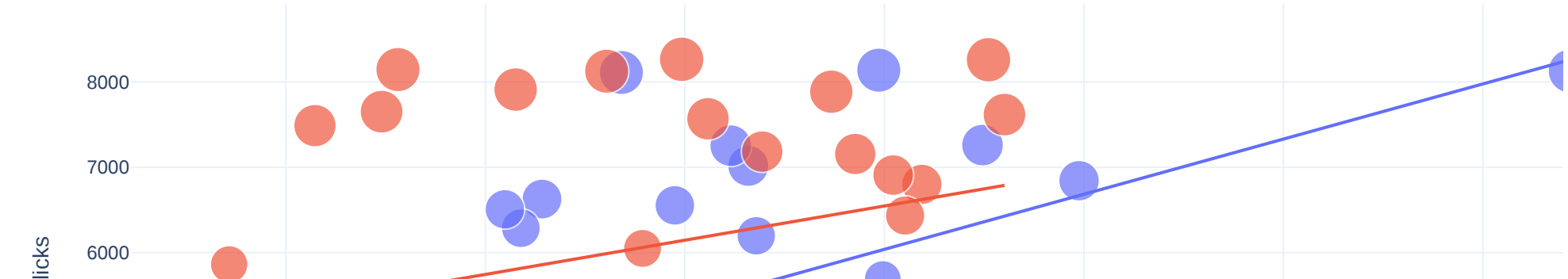


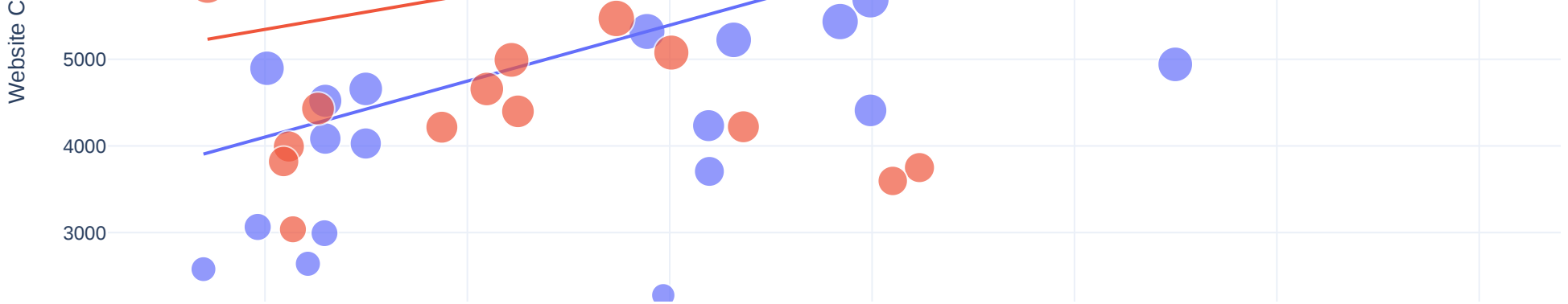
There's only a marginal difference of approximately 1% in the purchases made from both ad campaigns. Since the control campaign achieved more sales with a lower marketing expenditure, the control campaign emerges as the winner in this aspect.

Let's delve into the analysis of some key metrics to determine which ad campaign converts more effectively. We'll begin by examining the relationship between the number of website clicks and content viewed from both campaigns.

In [17]:

```
# Create a scatter plot using Plotly Express
figure = px.scatter(data_frame=ab_data,  # The DataFrame to use for the plot
                    x="Content Viewed",  # X-axis variable
                    y="Website Clicks",  # Y-axis variable
                    size="Website Clicks",  # Size of data points based on "Website Clicks"
                    color="Campaign Name",  # Color data points by "Campaign Name"
                    trendline="ols"  # Add an ordinary least squares (OLS) trendline
)

# Show the plot
figure.show()
```



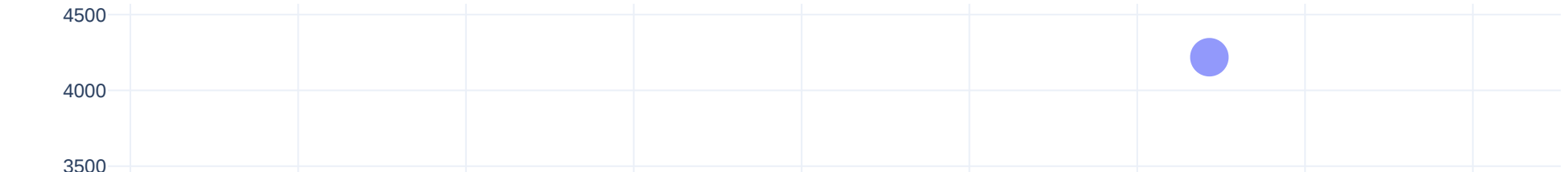


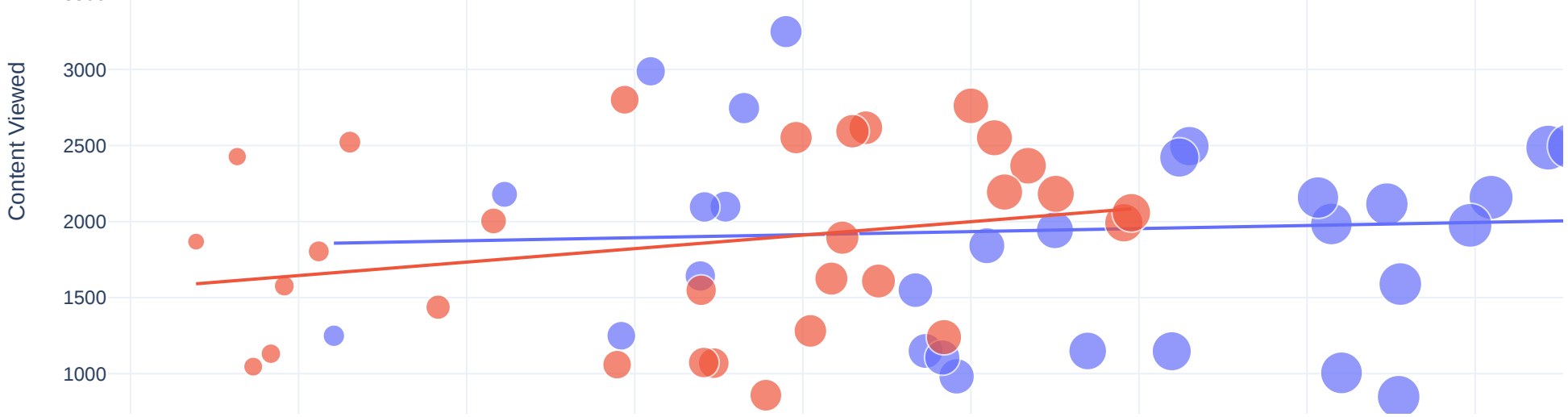
The test campaign records higher website clicks, but the engagement derived from these clicks is higher in the control campaign. Therefore, the control campaign secures a victory in this aspect.

Now, let's proceed with the analysis of the relationship between the amount of content viewed and the number of products added to the cart from both campaigns.

In [18]:

```
# Create a scatter plot using Plotly Express
figure = px.scatter(data_frame=ab_data,  # The DataFrame to use for the plot
                    x="Added to Cart",  # X-axis variable
                    y="Content Viewed",  # Y-axis variable
                    size="Added to Cart",  # Size of data points based on "Added to Cart"
                    color="Campaign Name",  # Color data points by "Campaign Name"
                    trendline="ols"  # Add an ordinary least squares (OLS) trendline
)

# Show the plot
figure.show()
```

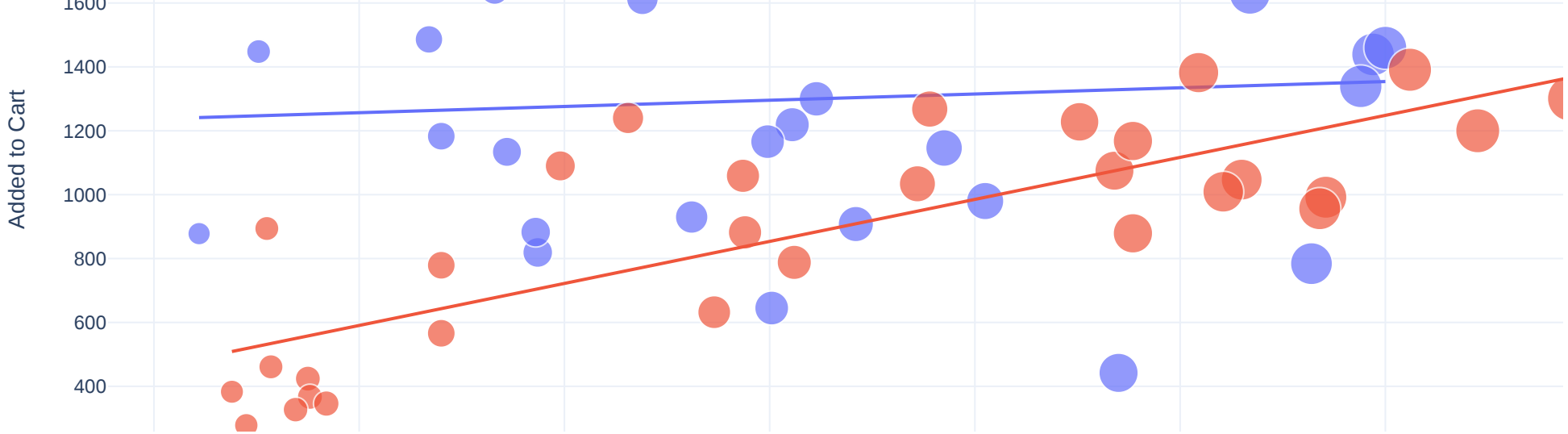




Once again, the control campaign emerges as the winner! Now, let's shift our focus to the relationship between the number of products added to the cart and the number of sales from both campaigns.

In [19]:

```
# Create a scatter plot using Plotly Express
figure = px.scatter(data_frame=ab_data,  # The DataFrame to use for the plot
                    x="Purchases",  # X-axis variable
                    y="Added to Cart",  # Y-axis variable
                    size="Purchases",  # Size of data points based on "Purchases"
                    color="Campaign Name",  # Color data points by "Campaign Name"
                    trendline="ols"  # Add an ordinary least squares (OLS) trendline
)

# Show the plot
figure.show()
```



Despite the control campaign generating more sales and having a higher number of products in the cart, it's worth noting that the conversion rate of the test campaign is higher.

Conclusion

From the above A/B tests, it's evident that the control campaign achieved more sales and higher visitor engagement. The control campaign received greater attention, with more products viewed, leading to a higher number of products in the cart and ultimately more sales. However, it's essential to note that the conversion rate of products in the cart is higher in the test campaign. The test campaign succeeded in generating more sales in relation to the products viewed and added to the cart. In summary, the test campaign can be effectively used to market a specific product to a targeted audience, while the control campaign is well-suited for promoting multiple products to a broader audience.

In [ ]: