Dataset Validation and EDA

product_sales DataFrame as `df`

```
-- Explore the data in the table (SQL)
SELECT *
FROM 'product_sales.csv';
```

	week	sales_method	customer_id	nb_sold	revenue	years_as_customer	nb_site_visits	state
0	2	Email	2e72d641-95ac-497b-bbf8-4861764a7097	10	NA	0	24	Arizona
1	6	Email + Call	3998a98d-70f5-44f7-942e-789bb8ad2fe7	15	225.47	1	28	Kansas
2	5	Call	d1de9884-8059-4065-b10f-86eef57e4a44	11	52.55	6	26	Wisconsin
3	4	Email	78aa75a4-ffeb-4817-b1d0-2f030783c5d7	11	NA	3	25	Indiana
4	3	Email	10e6d446-10a5-42e5-8210-1b5438f70922	9	90.49	0	28	Illinois
5	6	Call	6489e678-40f2-4fed-a48e-d0dff9c09205	13	65.01	10	24	Mississippi
6	4	Email	eb6bd5f1-f115-4e4b-80a6-5e67fcfbfb94	11	113.38	9	28	Georgia
7	1	Email	047df079-071b-4380-9012-2bfe9bce45d5	10	99.94	1	22	Oklahoma
8	5	Email	771586bd-7b64-40be-87df-afe884d2af9e	11	108.34	10	31	Massachusetts
9	5	Call	56491dae-bbe7-49f0-a651-b823a01103d8	11	53.82	7	23	Missouri
10	3	Email	c40f2602-8a7c-429e-bf13-cb1ec9e5f92f	9	89.49	4	28	Texas
11	2	Call	c20ab049-cbac-4ba7-8868-310aa89e0549	9	45.42	2	23	New York
12	5	Call	0b026b91-fe12-4af0-86f9-387ba81c8fdb	11	53.42	2	30	Maryland
13	2	Email	6103bcac-9da6-4000-a0ce-fa2615cce846	10	101.54	1	28	California
14	5	Call	96c8b5b8-cb81-4c75-a284-0e0026a03be8	10	51.87	1	30	Tennessee
15	4	Email	189d4f1b-9e76-4f64-9e71-7bd9b133a2d1	10	104.22	2	23	Missouri

12,500 rows ⚠ truncated from 15,000 rows

```
# Display basic information about the dataframe
df.info()
```

```
<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
```

```
# A quick overview of the main statistical measures for numerical columns
df.describe()
```

	week	nb_sold	revenue	years_as_customer	nb_site_visits
count	15000	15000	13926	15000	15000
mean	3.0982666667	10.0846666667	93.9349425535	4.9659333333	24.9908666667
std	1.6564198071	1.8122133327	47.4353122457	5.0449515589	3.5009142152
min	1	7	32.54	0	12
25%	2	9	52.47	1	23
50%	3	10	89.5	3	25
75%	5	11	107.3275	7	27
max	6	16	238.32	63	41

8 rows

```
# Display the number of rows in the dataframe
df.shape[0]
```

```
15000
```

```
# Revenue change to float65
import pandas as pd

df['revenue'] = pd.to_numeric(df['revenue'], errors='coerce')
df['revenue'].head()
```

	revenue
0	null
1	225.47
2	52.55
3	null
4	90.49

5 rows

```
# Check for missing values
print(df.isna().sum())
```

```
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
```

```
na_count = df[df['revenue'] == 'null'].shape[0]
print(na_count)
```

```
0
```

```
# Calculate the total revenue
total_revenue = df['revenue'].sum()
total_revenue
```

```
1308138.01
```

```
# Stats for revenue

df['revenue'].describe()
```

	revenue
count	13926
mean	93.9349425535
std	47.4353122457
min	32.54
25%	52.47
50%	89.5
75%	107.3275
max	238.32

8 rows

```
# Check affect of dropping null values
df_cleaned = df.dropna(subset=['revenue'])
```

```
df_cleaned['revenue'].describe()
```

	revenue	
count	13926	
mean	93.9349425535	
std	47.4353122457	
min	32.54	
25%	52.47	
50%	89.5	
75%	107.3275	
max	238.32	

8 rows

```
# Address "null" values in revenue
# 1. summarize columns with null values

# Filter rows where revenue is null
null_revenue_rows = df[df['revenue'].isna()]

# Display the rows with null revenue values
print(null_revenue_rows)
```

```
       week  sales_method  ... nb_site_visits         state
0         2         Email  ...             24       Arizona
3         4         Email  ...             25       Indiana
16        2         Email  ...             30  Pennsylvania
17        6  Email + Call  ...             24     Wisconsin
28        5         Email  ...             32       Florida
...     ...           ...  ...            ...           ...
14951     5          Call  ...             25      Virginia
14957     1          Call  ...             23      Illinois
14970     4  Email + Call  ...             25    Washington
14992     5  Email + Call  ...             34      New York
14999     5  Email + Call  ...             25      Illinois

[1074 rows x 8 columns]
```

```
# Step 1: Filter rows where revenue is null (NaN)
null_revenue_rows = df[df['revenue'].isna()]

# count null revenue rows
total_null_revenue_count = null_revenue_rows.shape[0]

# Step 2: Group the null revenue rows by sales_method and count them
null_revenue_by_sales_method = null_revenue_rows.groupby('sales_method').size()

# Step 3: Calculate the total number of entries for each sales_method
total_sales_method_counts = df.groupby('sales_method').size()

# Step 4: Calculate the percentage of rows with null revenue for each sales_method
null_revenue_percentage_by_sales_method = (null_revenue_by_sales_method / total_null_revenue_count) * 100

# Display the result
print(total_null_revenue_count)
print(null_revenue_by_sales_method)
print(null_revenue_percentage_by_sales_method)
```

```
1074
sales_method
Call            181
Email           544
Email + Call    349
dtype: int64
sales_method
Call            16.852886
Email           50.651769
Email + Call    32.495345
dtype: float64
```

```
# Group the entire DataFrame by sales_method and count rows of revenue
revenue_count_by_sales_method = df.groupby('sales_method')['revenue'].count()

# Calculate the total number of rows in the dataset
total_rows = len(df)

# Calculate the percentage of rows with non-null revenue for each sales_method
revenue_percentage_by_sales_method = (revenue_count_by_sales_method / total_rows) * 100

# Display the result
print(revenue_count_by_sales_method)
print(revenue_percentage_by_sales_method)
```

```
sales_method
Call            4781
Email           6922
Email + Call    2223
Name: revenue, dtype: int64
sales_method
Call            31.873333
Email           46.146667
Email + Call    14.820000
Name: revenue, dtype: float64
```

```
# Display all distinct categories in the 'sales_method' column
unique_sales_methods = df['sales_method'].unique()
print(unique_sales_methods)
```

```
['Email' 'Email + Call' 'Call' 'em + call' 'email']
```

```
sales_method_mapping = {
    'Email': 'Email',
    'email': 'Email',          # Normalize different cases
    'Email + Call': 'Email + Call',
    'Call': 'Call',
    'em + call': 'Email + Call' # Merge 'em + call' into 'Email + Call'
}

# Replace categories in the 'sales_method' column based on the mapping
df['sales_method'] = df['sales_method'].replace(sales_method_mapping)

# Display all distinct categories in the 'sales_method' column
unique_sales_methods = df['sales_method'].unique()
print(unique_sales_methods)
```

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

```
df['sales_method'].count()
```

```
15000
```

```
# Count occurrences of each category
sales_method_counts = df['sales_method'].value_counts()

# Display the counts
print(sales_method_counts)
```

```
sales_method
Email           7466
Call            4962
Email + Call    2572
Name: count, dtype: int64
```

```
# Check for duplicate rows (entire rows)
duplicate_rows = df.duplicated().sum()

# Check for duplicate customer_id
duplicate_customer_ids = df['customer_id'].duplicated().sum()
```

```
print(f"Duplicate rows: {duplicate_rows}")
print(f"Duplicate customer_ids: {duplicate_customer_ids}")
```

```
Duplicate rows: 0
Duplicate customer_ids: 0
```

```
# Check for outliers or inconsistencies
# Descriptive statistics for numeric columns
print("Summary statistics for numeric columns:")
print(df.describe())

# For example, check if there are negative values in nb_sold or revenue
invalid_nb_sold = df[df['nb_sold'] < 0]
invalid_revenue = df[df['revenue'] < 0]

print(f"Rows with invalid nb_sold values:\n{invalid_nb_sold}")
print(f"Rows with invalid revenue values:\n{invalid_revenue}")
```

```
Summary statistics for numeric columns:
               week       nb_sold  ...  years_as_customer  nb_site_visits
count  15000.000000  15000.000000  ...       15000.000000    15000.000000
mean       3.098267     10.084667  ...           4.965933       24.990867
std        1.656420      1.812213  ...           5.044952        3.500914
min        1.000000      7.000000  ...           0.000000       12.000000
25%        2.000000      9.000000  ...           1.000000       23.000000
50%        3.000000     10.000000  ...           3.000000       25.000000
75%        5.000000     11.000000  ...           7.000000       27.000000
max        6.000000     16.000000  ...          63.000000       41.000000

[8 rows x 5 columns]
Rows with invalid nb_sold values:
Empty DataFrame
Columns: [week, sales_method, customer_id, nb_sold, revenue, years_as_customer, nb_site_visits, state]
Index: []
Rows with invalid revenue values:
Empty DataFrame
Columns: [week, sales_method, customer_id, nb_sold, revenue, years_as_customer, nb_site_visits, state]
Index: []
```

```
# Example: Check for customers with nb_sold > 0 but missing or zero revenue
inconsistent_revenue = df[(df['nb_sold'] > 0) & ((df['revenue'].isna()) | (df['revenue'] == 0))]

print("Rows with inconsistent nb_sold and revenue values:")
print(inconsistent_revenue)
```

```
Rows with inconsistent nb_sold and revenue values:
       week  sales_method  ... nb_site_visits         state
0         2         Email  ...             24       Arizona
3         4         Email  ...             25       Indiana
16        2         Email  ...             30  Pennsylvania
17        6  Email + Call  ...             24     Wisconsin
28        5         Email  ...             32       Florida
...     ...           ...  ...            ...           ...
14951     5          Call  ...             25      Virginia
14957     1          Call  ...             23      Illinois
14970     4  Email + Call  ...             25    Washington
14992     5  Email + Call  ...             34      New York
14999     5  Email + Call  ...             25      Illinois

[1074 rows x 8 columns]
```

```
# Categorical values validation

print("Unique state values:")
print(df['state'].unique())
print(df['state'].value_counts())
```

```
Unique state values:
['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']
state
California        1872
Texas             1187
New York           965
Florida            904
Illinois           617
Pennsylvania       598
Ohio               566
Michigan           498
Georgia            489
North Carolina     459
New Jersey         434
Virginia           372
Indiana            354
Tennessee          340
Washington         335
Arizona            321
Missouri           311
Massachusetts      294
```

```
unique_weeks = df['week'].unique()
print(unique_weeks)
```

```
[2 6 5 4 3 1]
```

```
weeks_count = df['week'].value_counts()
print(weeks_count)
```

```
week
1    3721
4    2575
5    2574
2    2491
3    2411
6    1228
Name: count, dtype: int64
```

```
# Calculate the total weeks counts
total_count = weeks_count.sum()
total_count
```

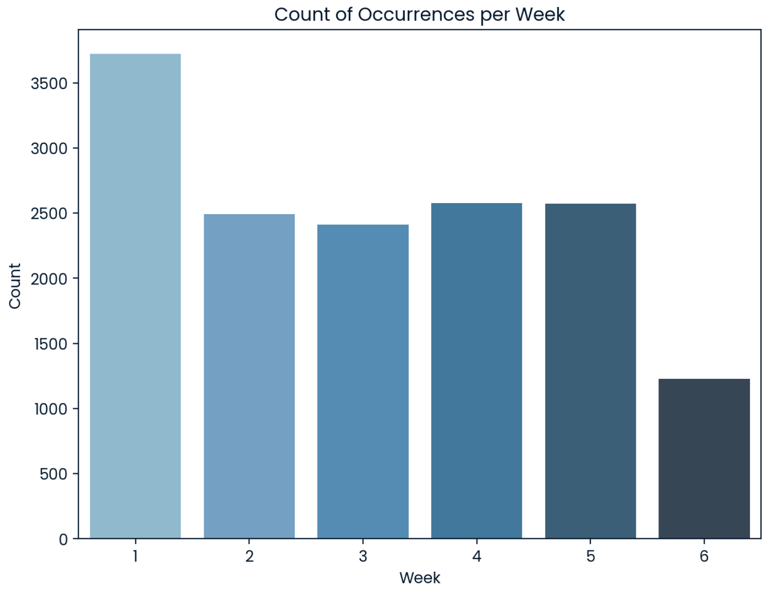
```
15000
```

```
# Calculate the percentage distribution
percentage_distribution = (weeks_count / total_count) * 100

# Print the original counts and the percentage distribution
print("Weeks Count:")
print(weeks_count)

print("\nPercentage Distribution:")
print(percentage_distribution)
```

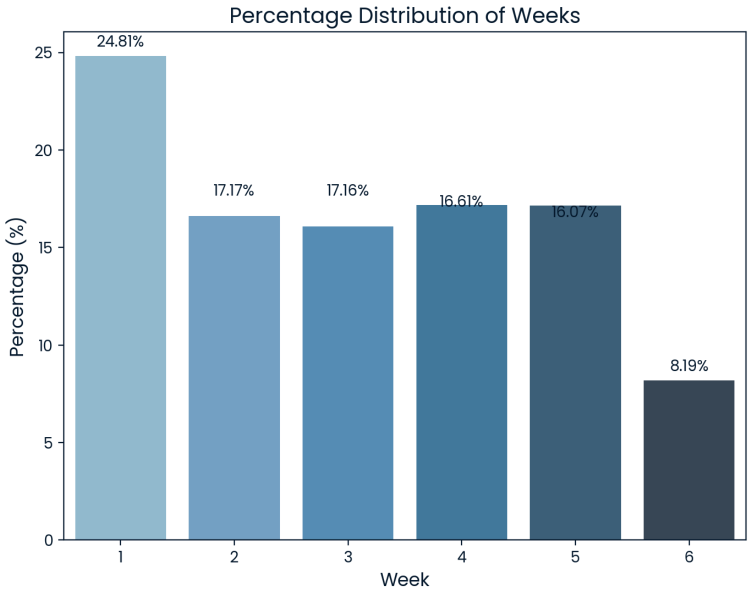
```
Weeks Count:
week
1    3721
4    2575
5    2574
2    2491
3    2411
6    1228
Name: count, dtype: int64

Percentage Distribution:
week
1    24.806667
4    17.166667
5    17.160000
2    16.606667
3    16.073333
6     8.186667
Name: count, dtype: float64
```

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

# Data (manually input based on your provided weeks_count)
weeks_count = {
    1: 3721,
    2: 2491,
    3: 2411,
    4: 2575,
    5: 2574,
    6: 1228
}

# Convert dictionary to two lists: weeks and counts
weeks = list(weeks_count.keys())
counts = list(weeks_count.values())

# Create the bar plot
plt.figure(figsize=(8, 6))  # Set figure size
sns.barplot(x=weeks, y=counts, palette='Blues_d')  # Create the barplot

# Add titles and labels
plt.title('Count of Occurrences per Week')
plt.xlabel('Week')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```

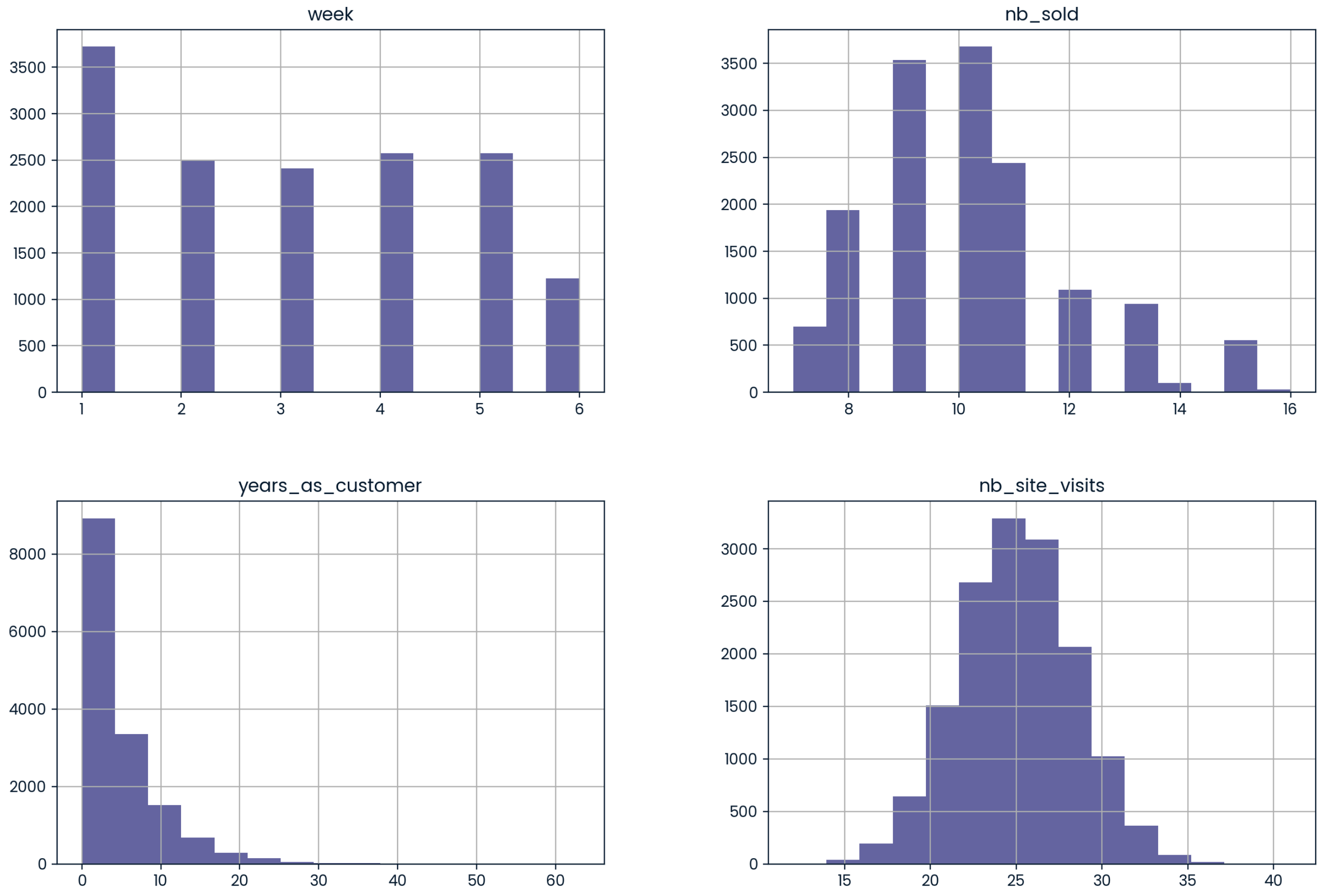


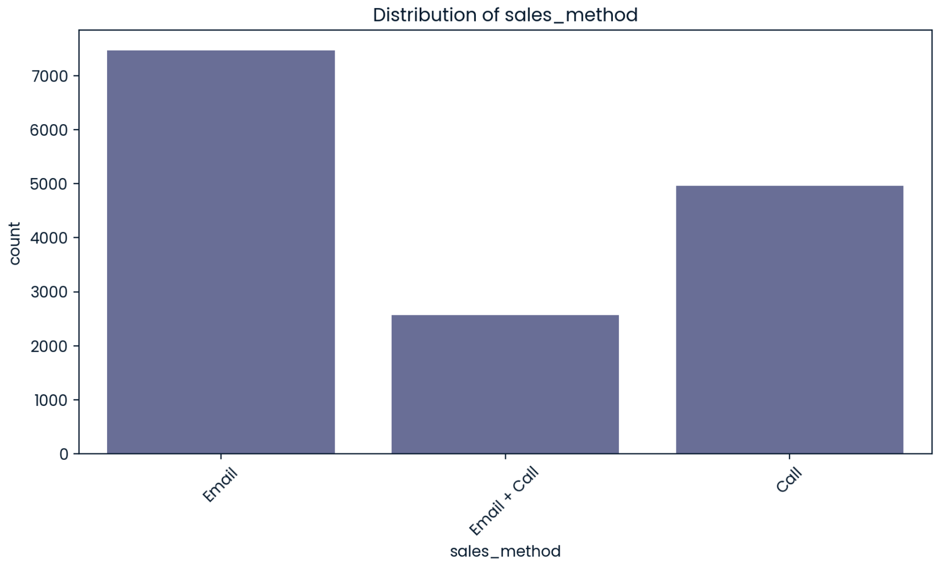
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Create the bar plot for percentage distribution
plt.figure(figsize=(8, 6))  # Set the figure size
sns.barplot(x=percentage_distribution.index, y=percentage_distribution.values, palette='Blues_d')

# Add titles and labels
plt.title('Percentage Distribution of Weeks', fontsize=14)
plt.xlabel('Week', fontsize=12)
plt.ylabel('Percentage (%)', fontsize=12)

# Show the percentage on top of the bars
for i, value in enumerate(percentage_distribution.values):
    plt.text(i, value + 0.5, f'{value:.2f}%', ha='center', fontsize=10)

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

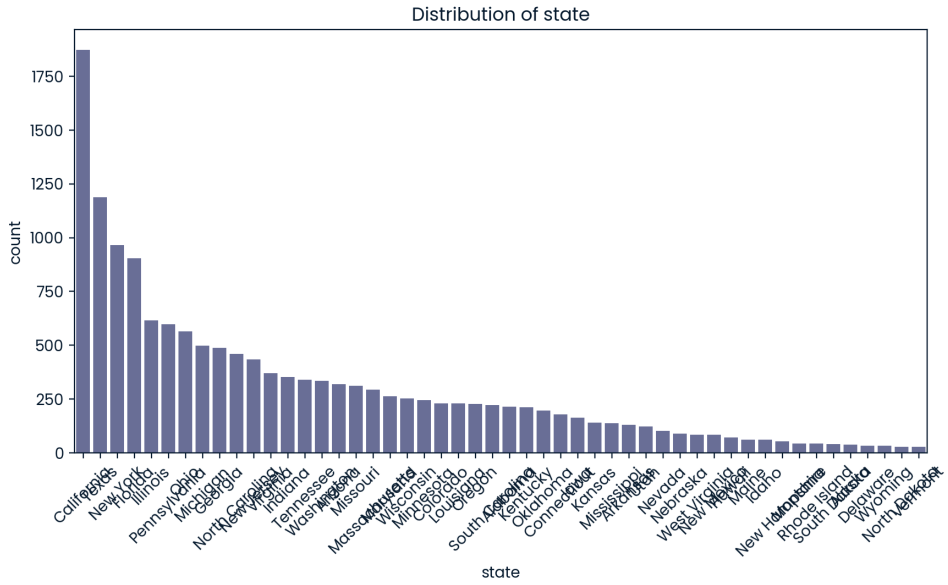


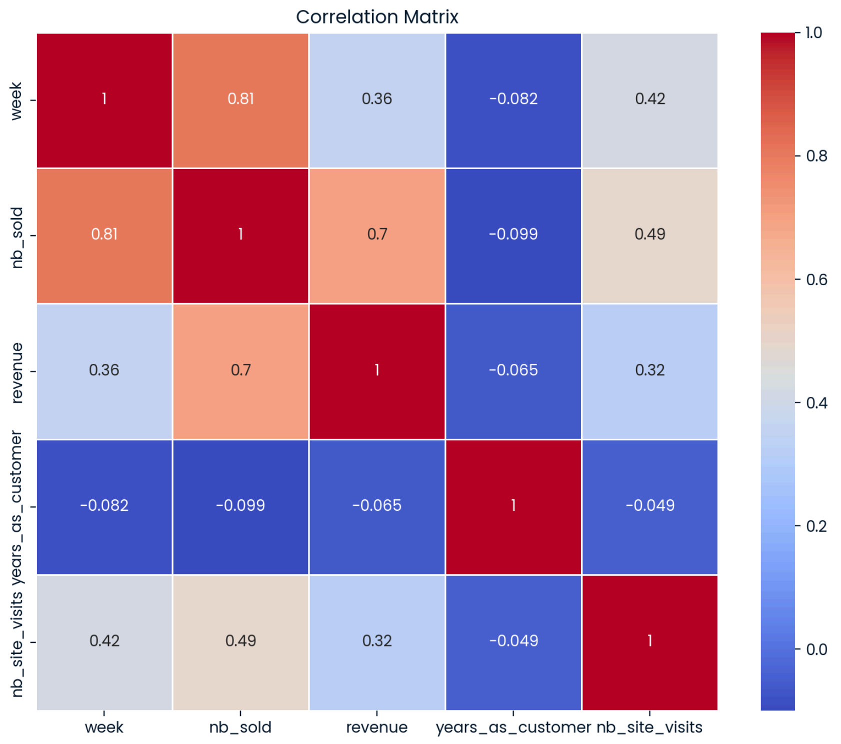
```python
# Re-display basic information about the dataframe
df.info()

# Display basic statistics of the dataframe
df.describe(include='all')

# Check for missing values
missing_values = df.isnull().sum()

# Display the first few rows of the dataframe
df.head()

# Plot the distribution of numerical columns
numerical_columns = ['week', 'nb_sold', 'years_as_customer', 'nb_site_visits']
df[numerical_columns].hist(bins=15, figsize=(15, 10), layout=(2, 2))

# Plot the distribution of categorical columns
categorical_columns = ['sales_method', 'state']
for column in categorical_columns:
    plt.figure(figsize=(10, 5))
    # Sort the state column based on the count (for 'state' column specifically)
    if column == 'state':
        order = df['state'].value_counts().index  # Sort states by count in descending order
        sns.countplot(data=df, x=column, order=order)
    else:
        sns.countplot(data=df, x=column)

    plt.title(f'Distribution of {column}')
    plt.xticks(rotation=45)
    plt.show()

# Plot the correlation matrix
plt.figure(figsize=(10, 8))
# Select only numerical columns for correlation matrix
numerical_df = df.select_dtypes(include=['number'])
correlation_matrix = numerical_df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

```
<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
```









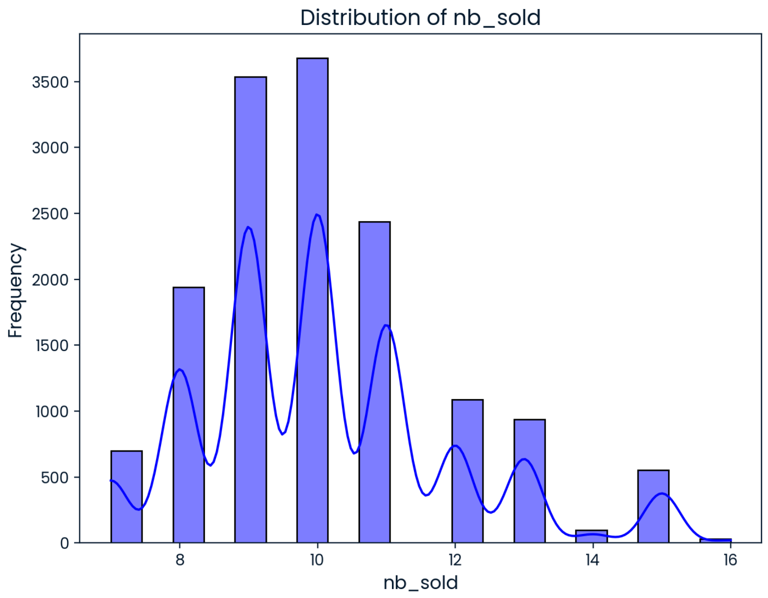
```
nb_sold_summary = df['nb_sold'].describe()
print(nb_sold_summary)
```

```
count    15000.000000
mean        10.084667
std          1.812213
min          7.000000
25%          9.000000
50%         10.000000
75%         11.000000
max         16.000000
Name: nb_sold, dtype: float64
```

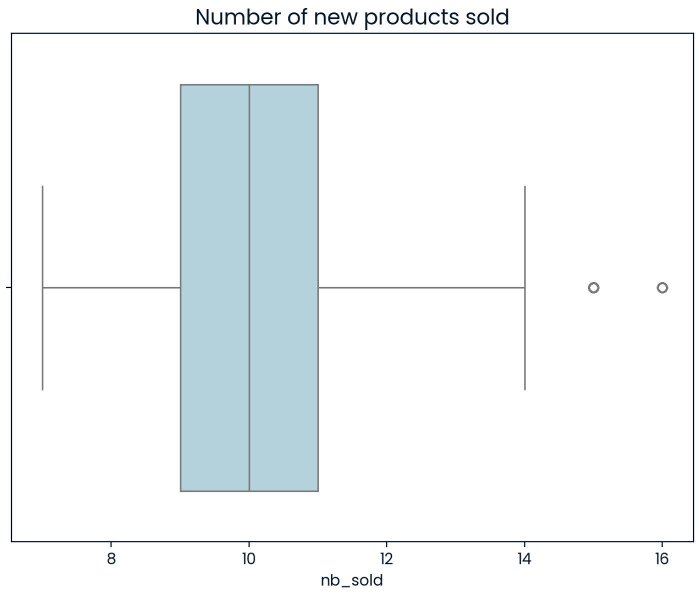
```
# Check for missing values
missing_nb_sold = df['nb_sold'].isnull().sum()
print(f"Missing values in nb_sold: {missing_nb_sold}")
```

```
Missing values in nb_sold: 0
```

```
# Plot the distribution of nb_sold using a histogram
plt.figure(figsize=(8, 6))
sns.histplot(df['nb_sold'], bins=20, kde=True, color='blue')

# Add titles and labels
plt.title('Distribution of nb_sold', fontsize=14)
plt.xlabel('nb_sold', fontsize=12)
plt.ylabel('Frequency', fontsize=12)

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



```
# Create a box plot for nb_sold
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['nb_sold'], color='lightblue')

# Add a title
plt.title('Number of new products sold', fontsize=14)

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



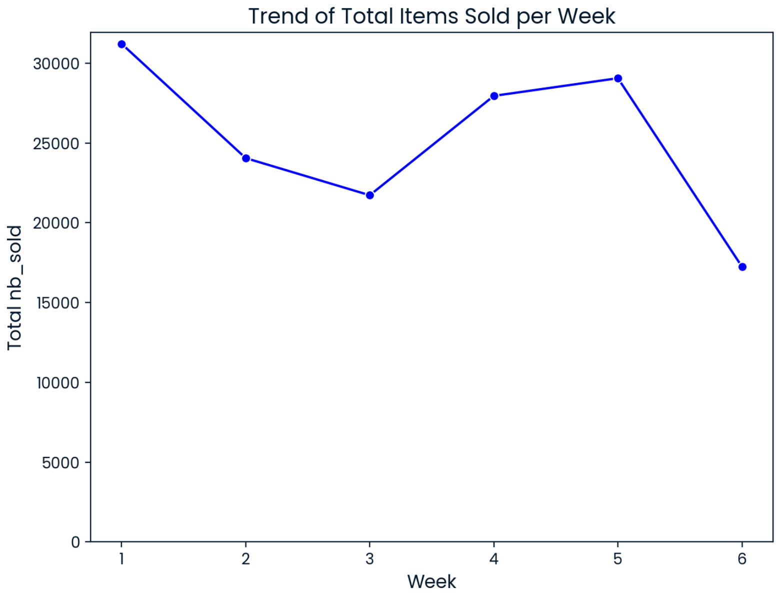
```
# Select only numeric columns from the DataFrame
numeric_df = df.select_dtypes(include='number')

# Calculate correlation between nb_sold and other numerical columns
if 'nb_sold' in numeric_df.columns:
    correlations = numeric_df.corr()['nb_sold'].sort_values(ascending=False)
    print(correlations)
else:
    print("The 'nb_sold' column is not found in the numeric columns.")
```

```
nb_sold              1.000000
week                 0.809887
revenue              0.696165
nb_site_visits       0.490718
years_as_customer   -0.099117
Name: nb_sold, dtype: float64
```

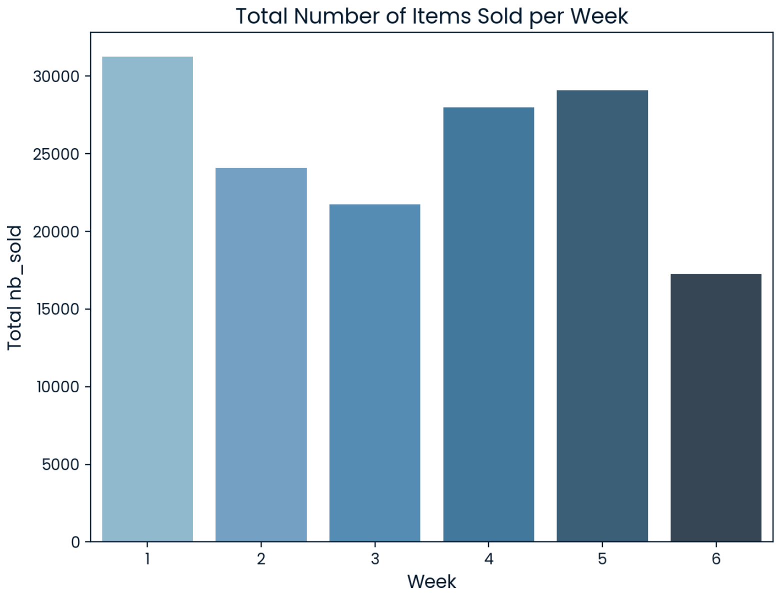
```
# Example: Group by week and calculate the sum of nb_sold for each week
nb_sold_by_week = df.groupby('week')['nb_sold'].sum()

print(nb_sold_by_week)
```

```
week
1    31220
2    24056
3    21728
4    27955
5    29063
6    17248
Name: nb_sold, dtype: int64
```

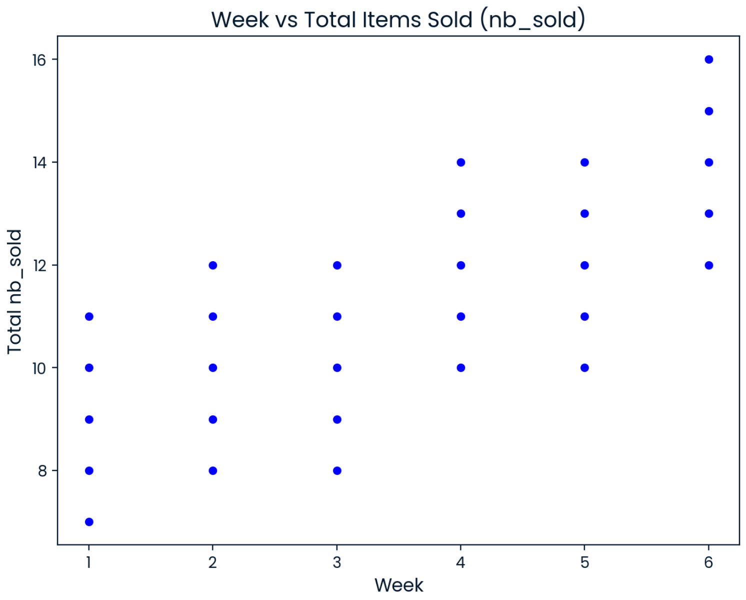
```
# Create a line plot for the sum of nb_sold by week
plt.figure(figsize=(8, 6))
sns.lineplot(x=nb_sold_by_week.index, y=nb_sold_by_week.values, marker='o', color='blue')

# Add titles and labels
plt.title('Trend of Total Items Sold per Week', fontsize=14)
plt.xlabel('Week', fontsize=12)
plt.ylabel('Total nb_sold', fontsize=12)

# Set the y-axis limit to start from 0
plt.ylim(0)

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

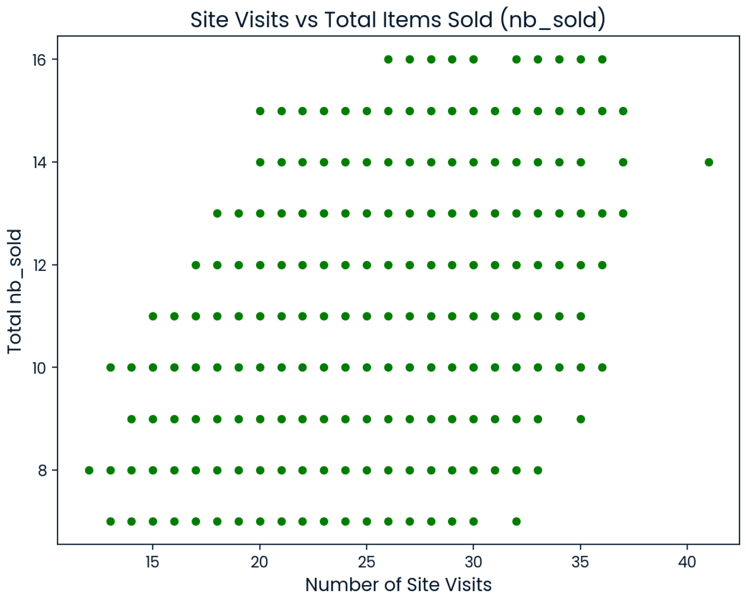


```
# Create a bar plot for the sum of nb_sold by week
plt.figure(figsize=(8, 6))
sns.barplot(x=nb_sold_by_week.index, y=nb_sold_by_week.values, palette='Blues_d')

# Add titles and labels
plt.title('Total Number of Items Sold per Week', fontsize=14)
plt.xlabel('Week', fontsize=12)
plt.ylabel('Total nb_sold', fontsize=12)

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



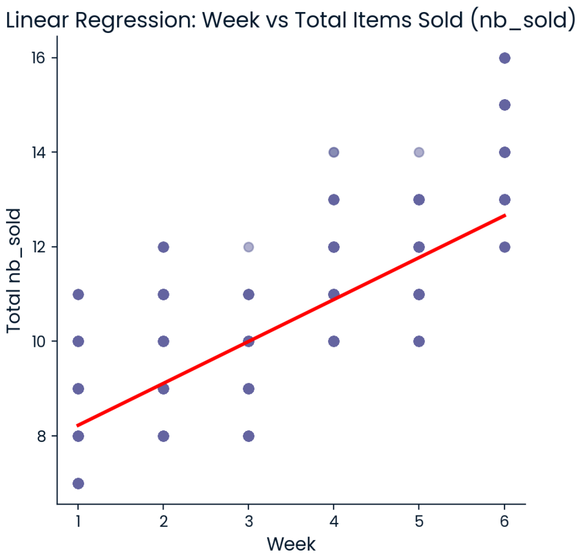
```
# Scatter plot for week vs nb_sold
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['week'], y=df['nb_sold'], color='blue')

# Add titles and labels
plt.title('Week vs Total Items Sold (nb_sold)', fontsize=14)
plt.xlabel('Week', fontsize=12)
plt.ylabel('Total nb_sold', fontsize=12)

# Show the plot
plt.show()

# Scatter plot for nb_site_visits vs nb_sold
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['nb_site_visits'], y=df['nb_sold'], color='green')

# Add titles and labels
plt.title('Site Visits vs Total Items Sold (nb_sold)', fontsize=14)
plt.xlabel('Number of Site Visits', fontsize=12)
plt.ylabel('Total nb_sold', fontsize=12)

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





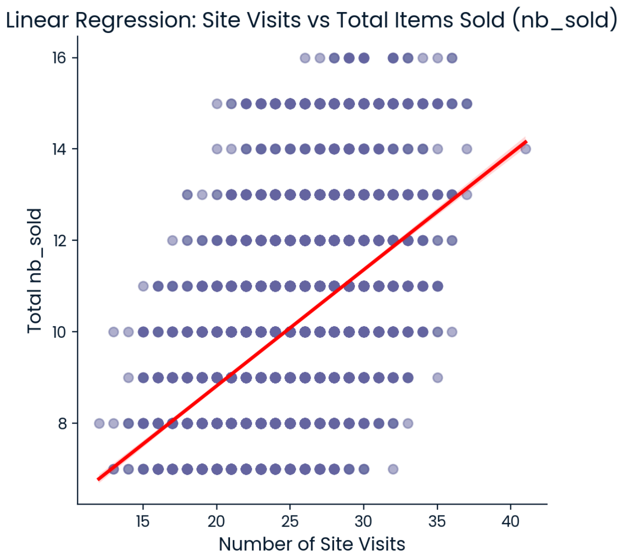
```
# Linear regression plot for week vs nb_sold
plt.figure(figsize=(8, 6))
sns.lmplot(x='week', y='nb_sold', data=df, line_kws={'color': 'red'}, scatter_kws={'alpha':0.5})

# Add titles and labels
plt.title('Linear Regression: Week vs Total Items Sold (nb_sold)', fontsize=14)
plt.xlabel('Week', fontsize=12)
plt.ylabel('Total nb_sold', fontsize=12)

plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```



```
# Linear regression plot for nb_site_visits vs nb_sold
plt.figure(figsize=(8, 6))
sns.lmplot(x='nb_site_visits', y='nb_sold', data=df, line_kws={'color': 'red'}, scatter_kws={'alpha':0.5})

# Add titles and labels
plt.title('Linear Regression: Site Visits vs Total Items Sold (nb_sold)', fontsize=14)
plt.xlabel('Number of Site Visits', fontsize=12)
plt.ylabel('Total nb_sold', fontsize=12)

plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```

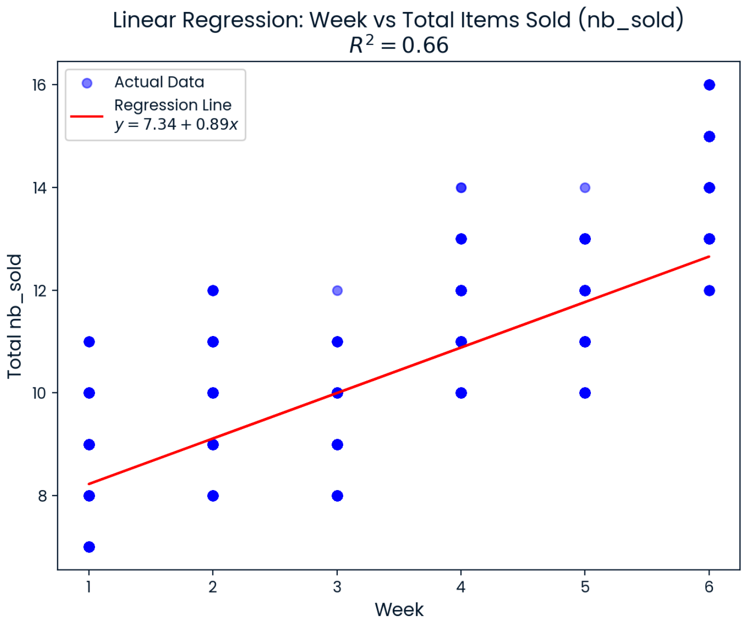


```
from sklearn.linear_model import LinearRegression

# Prepare data
X = df[['week']]  # Independent variable
y = df['nb_sold']  # Dependent variable

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X, y)
```

```
# Print the coefficients and intercept
print(f"Intercept: {model.intercept_}")
print(f"Coefficient (slope): {model.coef_[0]}")

# R-squared value to evaluate the model fit
r_squared = model.score(X, y)
print(f"R-squared: {r_squared}")
```

```
Intercept: 7.339413727962314
Coefficient (slope): 0.8860608959970155
R-squared: 0.6559176669278741
```

```
# Prepare the data
X = df[['week']]  # Independent variable (reshape as 2D array)
y = df['nb_sold']  # Dependent variable

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict values based on the model
y_pred = model.predict(X)

# Calculate regression coefficients
intercept = model.intercept_
slope = model.coef_[0]

# Calculate R-squared value
r_squared = model.score(X, y)

# Create the scatter plot and regression line
plt.figure(figsize=(8,6))
plt.scatter(df['week'], df['nb_sold'], color='blue', alpha=0.5, label='Actual Data')  # Scatter plot
plt.plot(df['week'], y_pred, color='red', label=f'Regression Line\n$y={intercept:.2f} + {slope:.2f}x$')  # Regression line

# Add titles and labels
plt.title(f'Linear Regression: Week vs Total Items Sold (nb_sold)\n$R^2={r_squared:.2f}$', fontsize=14)
plt.xlabel('Week', fontsize=12)
plt.ylabel('Total nb_sold', fontsize=12)

# Add legend
plt.legend()

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



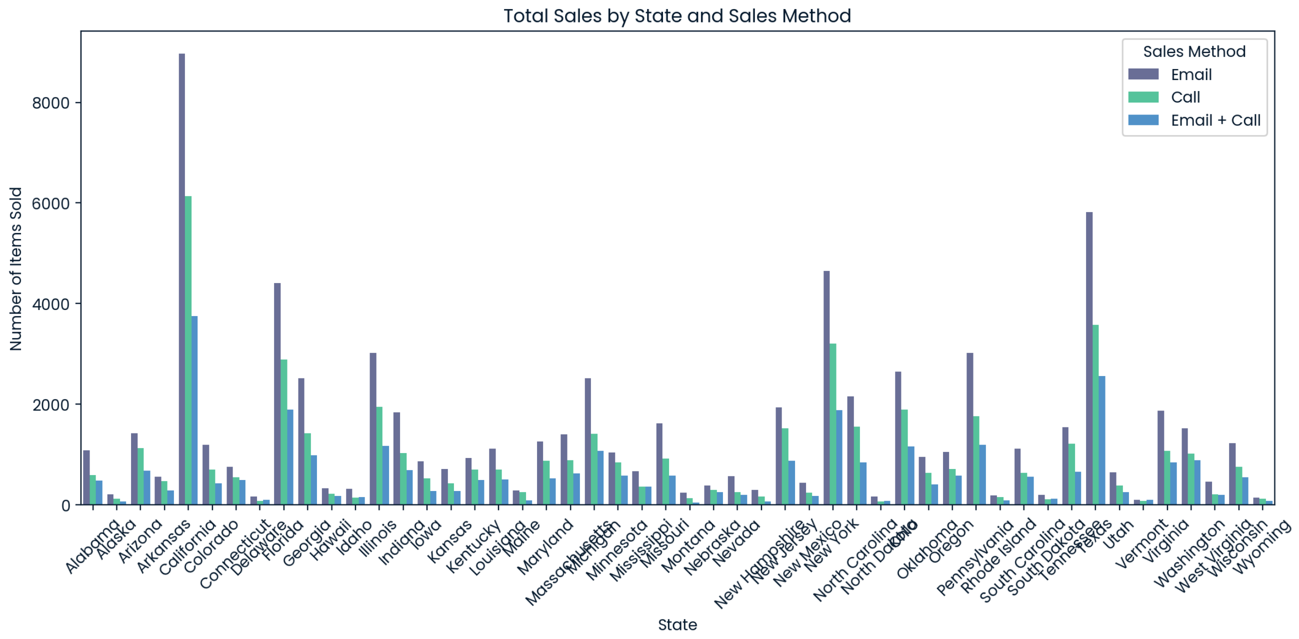
```
state_sales = df.groupby('state')['nb_sold'].sum().reset_index()
state_sales = state_sales.sort_values(by='nb_sold', ascending=False)
print(state_sales)

# Sort values to get the top 5 and bottom 5
# top_5_states = state_sales.sort_values(by='nb_sold', ascending=False).head(5)
# bottom_5_states = state_sales.sort_values(by='nb_sold', ascending=True).head(5)

# Display the results
# print("Top 5 States by Number of Items Sold:")
#print(top_5_states)

# print("\nBottom 5 States by Number of Items Sold:")
# print(bottom_5_states)
```

```
             state  nb_sold
4       California    18859
42           Texas    11957
31        New York     9734
8          Florida     9201
12        Illinois     6143
37    Pennsylvania     5979
34            Ohio     5699
21        Michigan     4998
9          Georgia     4930
32  North Carolina     4559
29      New Jersey     4338
45        Virginia     3790
13         Indiana     3558
46      Washington     3424
41       Tennessee     3414
2          Arizona     3238
24        Missouri     3122
20   Massachusetts     2913
19        Maryland     2669
48       Wisconsin     2528
22       Minnesota     2475
36          Oregon     2347
17       Louisiana     2325
5         Colorado     2322
39  South Carolina     2313
0          Alabama     2161
16        Kentucky     2131
35        Oklahoma     1998
```

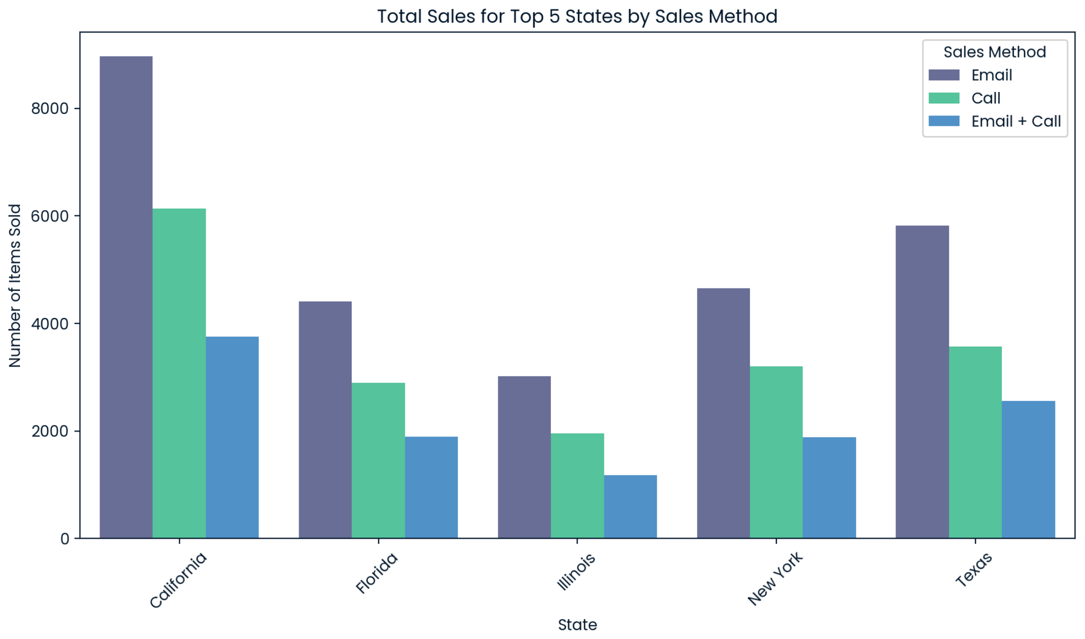
```
# Group by 'state' and 'sales_method', summing 'nb_sold'
sales_by_state_method = df.groupby(['state', 'sales_method'])['nb_sold'].sum().reset_index()

# Sort by 'state' and 'nb_sold' for better readability
sales_by_state_method = sales_by_state_method.sort_values(by=['state', 'nb_sold'], ascending=[True, False])

# Display the results
print("Total Sales by State and Sales Method:")
print(sales_by_state_method)
```

```
Total Sales by State and Sales Method:
         state  sales_method  nb_sold
1      Alabama         Email     1084
0      Alabama          Call      591
2      Alabama  Email + Call      486
4       Alaska         Email      211
3       Alaska          Call      128
..         ...           ...      ...
144  Wisconsin          Call      759
146  Wisconsin  Email + Call      545
148    Wyoming         Email      142
147    Wyoming          Call      119
149    Wyoming  Email + Call       79

[150 rows x 3 columns]
```

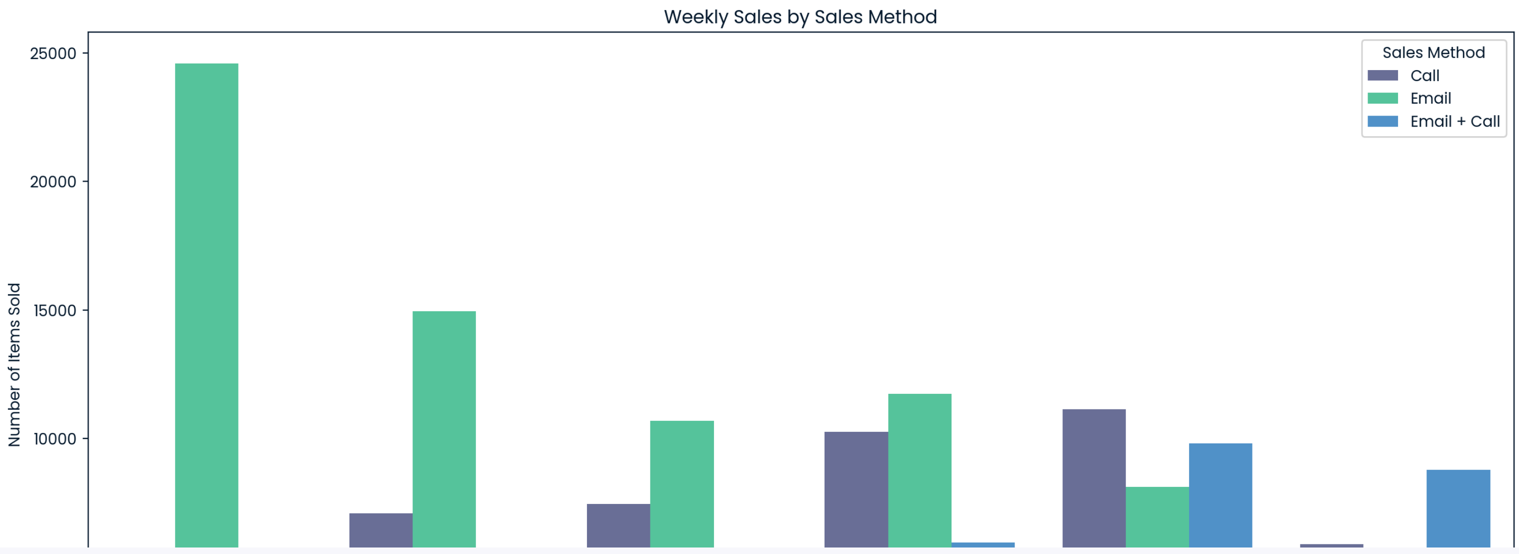
```
import matplotlib.pyplot as plt

# Create a bar plot
plt.figure(figsize=(12, 6))
sns.barplot(data=sales_by_state_method, x='state', y='nb_sold', hue='sales_method')

# Adding titles and labels
plt.title('Total Sales by State and Sales Method')
plt.xlabel('State')
plt.ylabel('Number of Items Sold')
plt.legend(title='Sales Method')
plt.xticks(rotation=45)
plt.tight_layout()

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



```
# Group by 'state' and 'sales_method', summing 'nb_sold'
sales_by_state_method = df.groupby(['state', 'sales_method'])['nb_sold'].sum().reset_index()

# Filter for the top five states
top_states = ['California', 'Texas', 'New York', 'Florida', 'Illinois']
sales_by_state_method = sales_by_state_method[sales_by_state_method['state'].isin(top_states)]

# Sort by 'state' and 'nb_sold' for better readability
sales_by_state_method = sales_by_state_method.sort_values(by=['state', 'nb_sold'], ascending=[True, False])

# Create a bar plot
plt.figure(figsize=(10, 6))
sns.barplot(data=sales_by_state_method, x='state', y='nb_sold', hue='sales_method')

# Adding titles and labels
plt.title('Total Sales for Top 5 States by Sales Method')
plt.xlabel('State')
plt.ylabel('Number of Items Sold')
plt.legend(title='Sales Method')
plt.xticks(rotation=45)
plt.tight_layout()

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



```
# Group by 'week' and 'sales_method', summing 'nb_sold'
weekly_sales = df.groupby(['week', 'sales_method'])['nb_sold'].sum().reset_index()

# Create a bar plot for visualization
plt.figure(figsize=(14, 7))
sns.barplot(data=weekly_sales, x='week', y='nb_sold', hue='sales_method')

# Adding titles and labels
plt.title('Weekly Sales by Sales Method')
plt.xlabel('Week')
plt.ylabel('Number of Items Sold')
plt.legend(title='Sales Method')
plt.xticks(rotation=45)
plt.tight_layout()

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


Weekly Sales by Sales Method
Number of Items Sold
25000
20000
15000
10000
Sales Method
Call
Email
Email + Call