# E-commerce | Shopping EDA

In [1]:
```
# Importing required packages to be used

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
from IPython.display import display
```

## Shopping Dataset

In [2]:
```
# Importing and Reading top 10 data

df_shopping=pd.read_csv('shopping.csv')
df_shopping.head(10)
```

Out[2]:

| | Administrative | Administrative_Duration | Informational | Informational_Duration | ProductRelated |
|---|---|---|---|---|---|
| **0** | 0 | 0.0 | 0 | 0.0 | 1 |
| **1** | 0 | 0.0 | 0 | 0.0 | 2 |
| **2** | 0 | 0.0 | 0 | 0.0 | 1 |
| **3** | 0 | 0.0 | 0 | 0.0 | 2 |
| **4** | 0 | 0.0 | 0 | 0.0 | 10 |
| **5** | 0 | 0.0 | 0 | 0.0 | 19 |
| **6** | 0 | 0.0 | 0 | 0.0 | 1 |
| **7** | 1 | 0.0 | 0 | 0.0 | 0 |
| **8** | 0 | 0.0 | 0 | 0.0 | 2 |
| **9** | 0 | 0.0 | 0 | 0.0 | 3 |

## General Analysis

In [3]:
```
# Defining the shape and dimension of data

a = df_shopping.shape
b = df_shopping.ndim
print("Shape-->", a,'\n',"Dimension-->",b)
```

```
Shape--> (12330, 18)
 Dimension--> 2
```

### Remarks: -

1. There are 12330 rows and 18 columns present in the dataset.
2. Its 2-d dataset by nature

In [4]:
```
# Checking general info of cols

df_shopping.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Administrative           12330 non-null  int64  
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64  
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64  
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object 
 11  OperatingSystems         12330 non-null  int64  
 12  Browser                  12330 non-null  int64  
 13  Region                   12330 non-null  int64  
 14  TrafficType              12330 non-null  int64  
 15  VisitorType              12330 non-null  object 
 16  Weekend                  12330 non-null  bool   
 17  Revenue                  12330 non-null  bool   
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

## Remarks: -

1. There are 2 string type columns (Month, VisitorType)
2. There are 2 bool type columns (Weekend, Revenue)
3. Rest all are integer and float types

In [5]:
```
# Overall stats of entire dataset

df_shopping.describe(include="all")
```

Out[5]:

| | Administrative | Administrative_Duration | Informational | Informational_Duration | ProductRel |
|---|---|---|---|---|---|
| **count** | 12330.000000 | 12330.000000 | 12330.000000 | 12330.000000 | 12330.00 |
| **unique** | NaN | NaN | NaN | NaN | |
| **top** | NaN | NaN | NaN | NaN | |
| **freq** | NaN | NaN | NaN | NaN | |
| **mean** | 2.315166 | 80.818611 | 0.503569 | 34.472398 | 31.73 |
| **std** | 3.321784 | 176.779107 | 1.270156 | 140.749294 | 44.47 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00 |
| **25%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 7.00 |
| **50%** | 1.000000 | 7.500000 | 0.000000 | 0.000000 | 18.00 |
| **75%** | 4.000000 | 93.256250 | 0.000000 | 0.000000 | 38.00 |
| **max** | 27.000000 | 3398.750000 | 24.000000 | 2549.375000 | 705.00 |

In [6]:
```
# Categorical Data Description

df_shopping.describe(include= object).T
```

Out[6]:

| | count | unique | top | freq |
|---|---|---|---|---|
| **Month** | 12330 | 10 | May | 3364 |
| **VisitorType** | 12330 | 3 | Returning_Visitor | 10551 |

## Remarks: -

1. This is solely targetting the object datatype columns overall statistics
2. There are 10 unique months and 3 unique VisitorType values present in entire dataset
3. The topmost consecutive value and its freq of each of the column values are present in the last 2 columns (in above result)

In [7]:
```
# Null/Empty values in dataset

df_shopping.isna().sum().sort_values(ascending=False)
```

Out[7]:
```
Administrative             0
Administrative_Duration    0
Weekend                    0
VisitorType                0
TrafficType                0
Region                     0
Browser                    0
OperatingSystems           0
Month                      0
SpecialDay                 0
PageValues                 0
ExitRates                  0
BounceRates                0
ProductRelated_Duration    0
ProductRelated             0
Informational_Duration     0
Informational              0
Revenue                    0
dtype: int64
```

## Remarks: -

1. This is a check on the number of nulls/empty values present in the entire dataset
2. There are no nulls/empty values present in the dataset for any of the column values

In [8]:
```
# Finding duplicate rows based on all columns

duplicate_rows = df_shopping[df_shopping.duplicated(keep=False)]

# Displaying the duplicate rows
print("Duplicate Rows:")
print(duplicate_rows)

# Counting the total number of duplicate rows
num_duplicates = len(duplicate_rows)

print(f"Total number of duplicate rows: {num_duplicates}")
```

```
Duplicate Rows:
       Administrative  Administrative_Duration  Informational  \
85                  0                      0.0              0
132                 0                      0.0              0
158                 0                      0.0              0
159                 0                      0.0              0
178                 0                      0.0              0
...               ...                      ...            ...
11934               0                      0.0              0
11938               0                      0.0              0
12159               0                      0.0              0
12180               0                      0.0              0
12185               0                      0.0              0

       Informational_Duration  ProductRelated  ProductRelated_Duration  \
85                        0.0               1                      0.0
132                       0.0               1                      0.0
158                       0.0               1                      0.0
159                       0.0               1                      0.0
178                       0.0               1                      0.0
...                       ...             ...                      ...
11934                     0.0               1                      0.0
11938                     0.0               1                      0.0
12159                     0.0               1                      0.0
12180                     0.0               1                      0.0
12185                     0.0               1                      0.0

       BounceRates  ExitRates  PageValues  SpecialDay Month  OperatingSystems
\
85             0.2        0.2         0.0         0.0   Feb                 1
132            0.2        0.2         0.0         0.0   Feb                 3
158            0.2        0.2         0.0         0.0   Feb                 1
159            0.2        0.2         0.0         0.0   Feb                 3
178            0.2        0.2         0.0         0.0   Feb                 3
...            ...        ...         ...         ...   ...               ...
11934          0.2        0.2         0.0         0.0   Dec                 1
11938          0.2        0.2         0.0         0.0   Dec                 1
12159          0.2        0.2         0.0         0.0   Dec                 1
12180          0.2        0.2         0.0         0.0   Dec                 1
12185          0.2        0.2         0.0         0.0   Dec                 8

       Browser  Region  TrafficType        VisitorType  Weekend  Revenue
85           1       1            3  Returning_Visitor    False    False
132          2       3            3  Returning_Visitor    False    False
158          1       1            3  Returning_Visitor    False    False
159          2       3            3  Returning_Visitor    False    False
178          2       3            3  Returning_Visitor    False    False
...        ...     ...          ...                ...      ...      ...
11934        1       1            2        New_Visitor    False    False
11938        1       4            1  Returning_Visitor     True    False
12159        1       1            3  Returning_Visitor    False    False
12180       13       9           20  Returning_Visitor    False    False
12185       13       9           20              Other    False    False

[201 rows x 18 columns]
Total number of duplicate rows: 201
```

## Remarks: -

1. There are 201 duplicated rows in entire dataset

In [9]:
```
# Removing duplicates

df_shopping_uniq = df_shopping.drop_duplicates()
```

In [10]:
```
# Cross verifying whether duplicate rows are removed or not

duplicate_rows = df_shopping_uniq[df_shopping_uniq.duplicated(keep=False)]

# Displaying the duplicate rows
print("Duplicate Rows:")
print(duplicate_rows)

# Counting the total number of duplicate rows
num_duplicates = len(duplicate_rows)

print(f"Total number of duplicate rows: {num_duplicates}")
```

```
Duplicate Rows:
Empty DataFrame
Columns: [Administrative, Administrative_Duration, Informational, Information
al_Duration, ProductRelated, ProductRelated_Duration, BounceRates, ExitRates,
PageValues, SpecialDay, Month, OperatingSystems, Browser, Region, TrafficTyp
e, VisitorType, Weekend, Revenue]
Index: []
Total number of duplicate rows: 0
```

In [11]:
```
# Checking count again and here the duplicates are handled
df_shopping_uniq.count()
```

Out[11]:
```
Administrative             12205
Administrative_Duration    12205
Informational              12205
Informational_Duration     12205
ProductRelated             12205
ProductRelated_Duration    12205
BounceRates                12205
ExitRates                  12205
PageValues                 12205
SpecialDay                 12205
Month                      12205
OperatingSystems           12205
Browser                    12205
Region                     12205
TrafficType                12205
VisitorType                12205
Weekend                    12205
Revenue                    12205
dtype: int64
```

In [12]:
```
# Checking for value ranges (e.g., numeric columns should not have negative va

numeric_columns = df_shopping.select_dtypes(include=['int', 'float']).columns
value_range_issues = (df_shopping[numeric_columns] < 0).any()

print("\nValue Range Issues:")
print(value_range_issues)
```

```
Value Range Issues:
Administrative             False
Administrative_Duration    False
Informational              False
Informational_Duration     False
ProductRelated             False
ProductRelated_Duration    False
BounceRates                False
ExitRates                  False
PageValues                 False
SpecialDay                 False
OperatingSystems           False
Browser                    False
Region                     False
TrafficType                False
dtype: bool
```

## Remarks: -

1. This is for checking if there is any oddity or unexpected data/anomalies present under the integer columns
2. There are no anomaly values and only numeric data is present across all the rows for these integer columns

# Non-Graphical Analysis

## Uniques and Value counts

In [13]:
```
# Unique values for Administrative

print("Unique values are: -")
print(df_shopping_uniq['Administrative'].unique())

# Value counts
print("\nValue counts are: -")
print(df_shopping_uniq['Administrative'].value_counts().reset_index().to_strin
```

```
Unique values are: -
[ 0  1  2  4 12  3 10  6  5  9  8 16 13 11  7 18 14 17 19 15 24 22 21 20
 23 27 26]

Value counts are: -
Administrative Count
             0  5643
             1  1354
             2  1114
             3   915
             4   765
             5   575
             6   432
             7   338
             8   287
             9   225
            10   153
            11   105
            12    86
            13    56
            14    44
            15    38
            16    24
            17    16
            18    12
            19     6
            24     4
            22     4
            23     3
            21     2
            20     2
            27     1
            26     1
```

In [14]:
```
# Unique values for Informational

print("Unique values are: -")
print(df_shopping_uniq['Informational'].unique())

# Value counts
print("\nValue counts are: -")
print(df_shopping_uniq['Informational'].value_counts().reset_index().to_string
```

```
Unique values are: -
[ 0  1  2  4 16  5  3 14  6 12  7  9 10  8 11 24 13]

Value counts are: -
Informational Count
            0  9574
            1  1041
            2   728
            3   380
            4   222
            5    99
            6    78
            7    36
            9    15
            8    14
           10     7
           12     5
           14     2
           16     1
           11     1
           24     1
           13     1
```

In [15]:
```
# Unique values for OperatingSystems

print("Unique values are: -")
print(df_shopping_uniq['OperatingSystems'].unique())

# Value counts
print("\nValue counts are: -")
print(df_shopping_uniq['OperatingSystems'].value_counts().reset_index().to_str
```

```
Unique values are: -
[1 2 4 3 7 6 8 5]

Value counts are: -
OperatingSystems Count
               2  6541
               1  2549
               3  2530
               4   478
               8    75
               6    19
               7     7
               5     6
```

In [16]:
```
# Unique values for Browser

print("Unique values are: -")
print(df_shopping_uniq['Browser'].unique())

# Value counts
print("\nValue counts are: -")
print(df_shopping_uniq['Browser'].value_counts().reset_index().to_string(index
```

```
Unique values are: -
[ 1  2  3  4  5  6  7 10  8  9 12 13 11]

Value counts are: -
Browser Count
      2  7883
      1  2427
      4   731
      5   465
      6   174
     10   163
      8   135
      3   105
     13    56
      7    49
     12    10
     11     6
      9     1
```

In [17]:
```
# Unique values for Region

print("Unique values are: -")
print(df_shopping_uniq['Region'].unique())

# Value counts
print("\nValue counts are: -")
print(df_shopping_uniq['Region'].value_counts().reset_index().to_string(index=
```

```
Unique values are: -
[1 9 2 3 4 5 6 7 8]

Value counts are: -
Region Count
     1  4714
     3  2379
     4  1171
     2  1128
     6   801
     7   758
     9   505
     8   431
     5   318
```

In [18]:
```
# Unique values for TrafficType

print("Unique values are: -")
print(df_shopping_uniq['TrafficType'].unique())

# Value counts
print("\nValue counts are: -")
print(df_shopping_uniq['TrafficType'].value_counts().reset_index().to_string(i
```

```
Unique values are: -
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 18 19 16 17 20]

Value counts are: -
TrafficType Count
          2  3911
          1  2388
          3  2013
          4  1066
         13   728
         10   450
          6   443
          8   343
          5   260
         11   247
         20   193
          9    41
          7    40
         15    37
         19    17
         14    13
         18    10
         16     3
         12     1
         17     1
```

In [19]:
```
# Unique values for VisitorType

print("Unique values are: -")
print(df_shopping_uniq['VisitorType'].unique())

# Value counts
print("\nValue counts are: -")
print(df_shopping_uniq['VisitorType'].value_counts().reset_index().to_string(i
```

```
Unique values are: -
['Returning_Visitor' 'New_Visitor' 'Other']

Value counts are: -
      VisitorType Count
Returning_Visitor 10431
      New_Visitor  1693
            Other    81
```

In [20]:
```
# Unique values for Weekend

print("Unique values are: -")
print(df_shopping_uniq['Weekend'].unique())

# Value counts
print("\nValue counts are: -")
print(df_shopping_uniq['Weekend'].value_counts().reset_index().to_string(index
```

```
Unique values are: -
[False  True]

Value counts are: -
Weekend Count
  False  9346
   True  2859
```

In [21]:
```
# Unique values for Revenue

print("Unique values are: -")
print(df_shopping_uniq['Revenue'].unique())

# Value counts
print("\nValue counts are: -")
print(df_shopping_uniq['Revenue'].value_counts().reset_index().to_string(index
```

```
Unique values are: -
[False  True]

Value counts are: -
Revenue Count
  False 10297
   True  1908
```

# Univariate Analysis

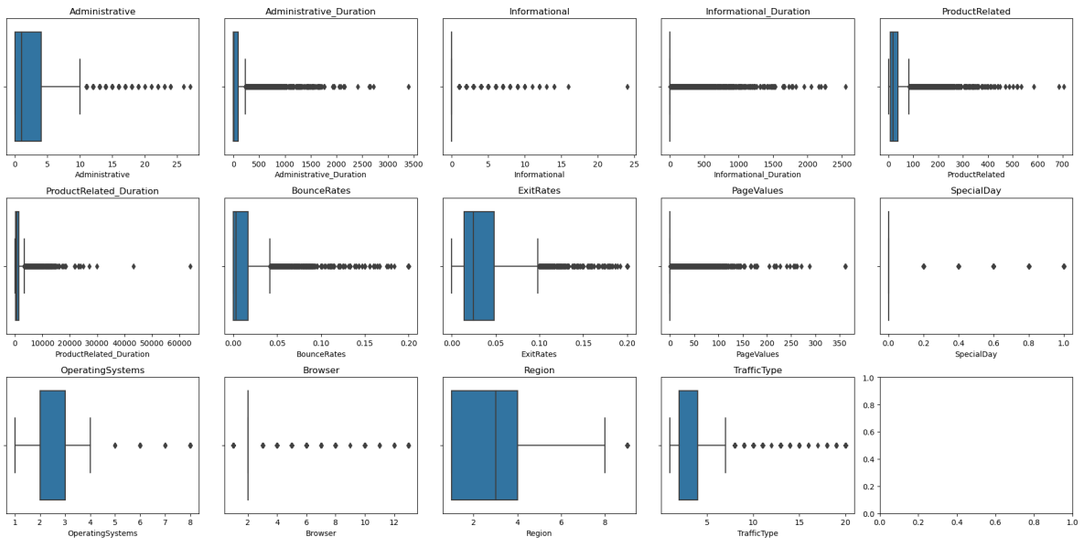
## Outlier Checks and Treatment

In [22]:
```
# Creating a backup of original database before handling outliers
df_shopping_uniq_copy=df_shopping_uniq.copy()
df_shopping_uniq_copy.head()
```

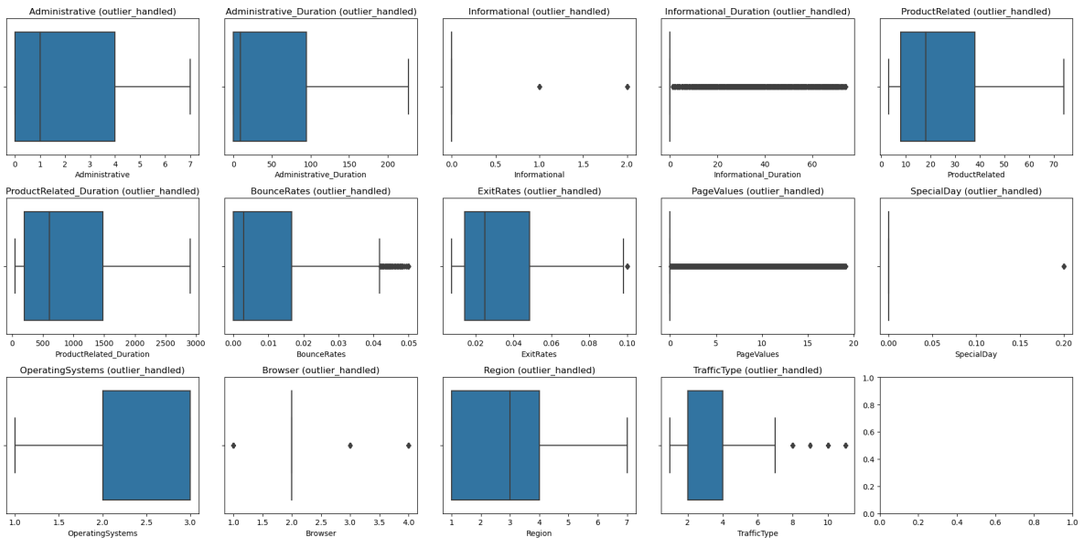
Out[22]:

| | Administrative | Administrative_Duration | Informational | Informational_Duration | ProductRelated |
|---|---|---|---|---|---|
| **0** | 0 | 0.0 | 0 | 0.0 | 1 |
| **1** | 0 | 0.0 | 0 | 0.0 | 2 |
| **2** | 0 | 0.0 | 0 | 0.0 | 1 |
| **3** | 0 | 0.0 | 0 | 0.0 | 2 |
| **4** | 0 | 0.0 | 0 | 0.0 | 10 |

In [23]:
```
# Checking outliers for all numeric columns

columns_of_interest = ["Administrative", "Administrative_Duration", "Informati

fig, axes = plt.subplots(3, 5, figsize=(20, 10))
axes = axes.flatten()

for i, column in enumerate(columns_of_interest):
    sns.boxplot(data=df_shopping_uniq, x=column, orient='h', ax=axes[i])
    axes[i].set_title(f"{column}")

plt.tight_layout()

plt.show()
```

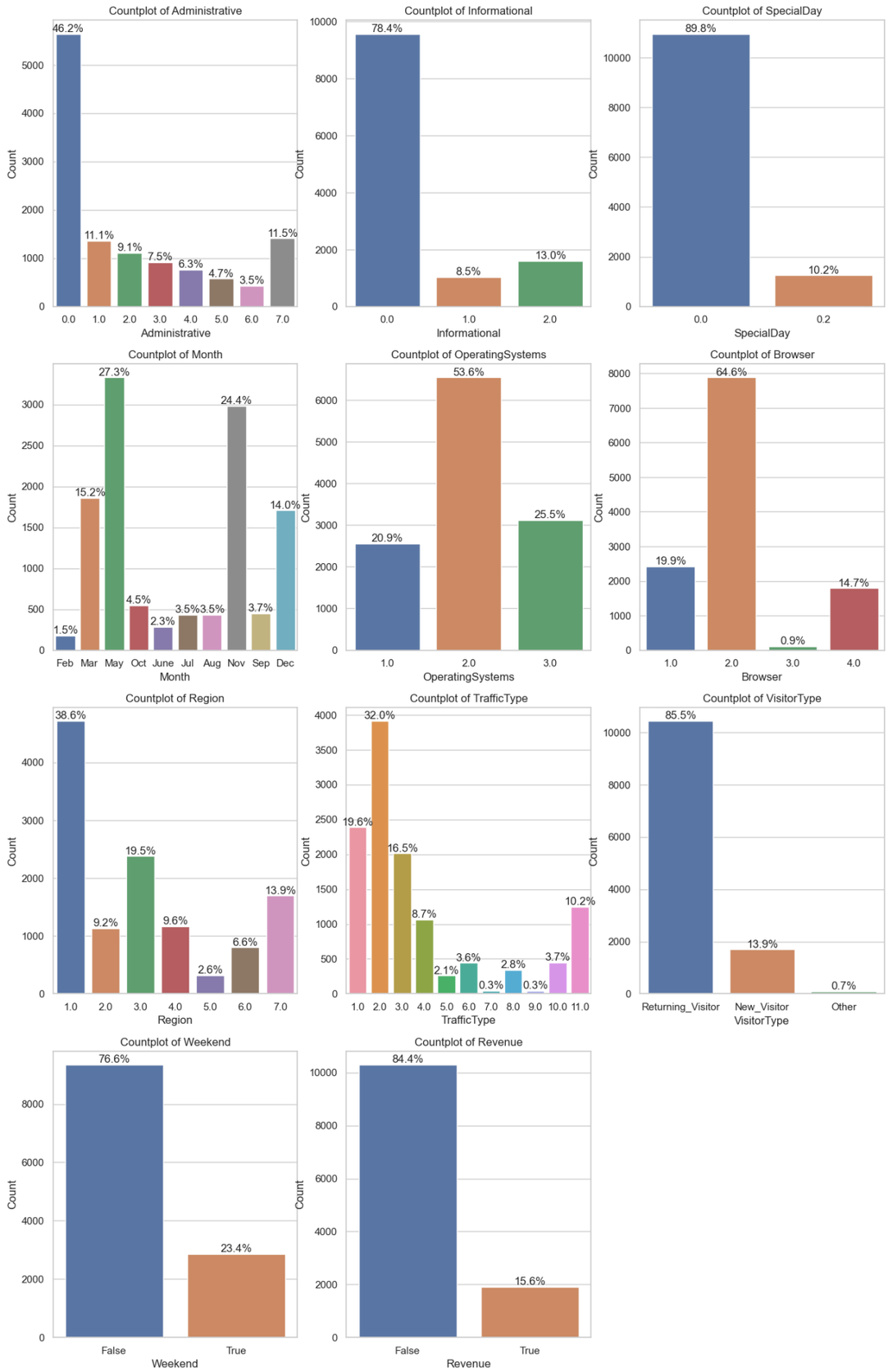


In [24]:
```
# Handling the outliers

columns_of_interest = ["Administrative", "Administrative_Duration", "Informati
fig, axes = plt.subplots(3, 5, figsize=(20,10))

axes = axes.flatten()

for i, column in enumerate(columns_of_interest):
    # Calculating the 10th and 90th percentiles
    tenth_percentile = df_shopping_uniq_copy[column].quantile(0.10)
    ninetieth_percentile = df_shopping_uniq_copy[column].quantile(0.90)

    # Applying quantile-based flooring and capping
    df_shopping_uniq_copy[column] = np.where(df_shopping_uniq_copy[column] < t
    df_shopping_uniq_copy[column] = np.where(df_shopping_uniq_copy[column] > n

    # Creating a boxplot of winsorized data
    sns.boxplot(data=df_shopping_uniq_copy, x=column, orient='h', ax=axes[i])
    axes[i].set_title(f"{column} (outlier_handled)")

plt.tight_layout()

plt.show()
```



# Graphical | Univariate analysis

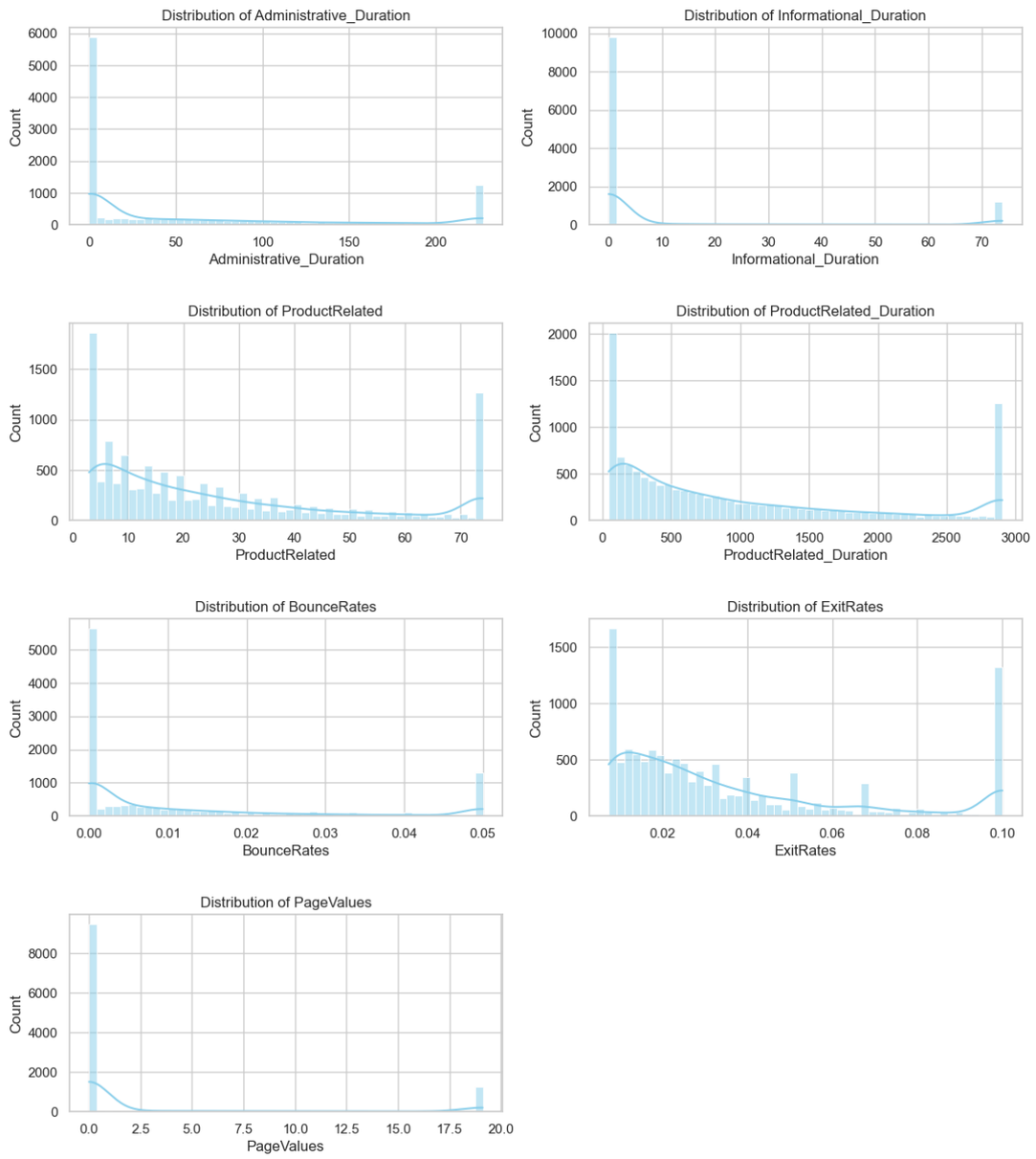
In [25]:
```
# Plotting counts and % distribution of columns present in the dataset

sns.set(style="whitegrid")

fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(16, 18))
fig.subplots_adjust(top=1.2)

columns = ["Administrative", "Informational", "SpecialDay", "Month",
           "OperatingSystems", "Browser", "Region", "TrafficType",
           "VisitorType", "Weekend", "Revenue"]

# Creating count plots
for i, column in enumerate(columns):
    row_index = i // 3
    col_index = i % 3

    plt_count = sns.countplot(data=df_shopping_uniq_copy, x=column, ax=axes[ro
    plt_count.set_title(f"Countplot of {column}")
    plt_count.set_xlabel(column)
    plt_count.set_ylabel("Count")

# Adding percentage labels on top of each bar for ease
    total = len(df_shopping_uniq_copy)
    for p in plt_count.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height() / total)
        x = p.get_x() + p.get_width() / 2
        y = p.get_height()
        plt_count.annotate(percentage, (x, y), ha='center', va='bottom')

# Removing the empty subplot
fig.delaxes(axes[3][2])

plt.show()
```




## Remarks: -

1. The highest count of administrative is for 0.0 followed by 7.0
2. 0.0 informational is the most used with overall usage % of 78.4% which means mostly non-informational pages are visited by customer
3. 89.8% is of 0.0 special day which means most transactions are done on non-special days.
4. The highest transactions is done in the month of May
5. The max operating system and browser belongs to 2.0 which is preferred/used by the user.
6. 38.6% of the users are located in 1.0 region which is the highest amongst all.
7. The highest traffic is observed from 2.0 followed by 1.0.
8. The maximum number of visitors/users belong to "returning visitor" type.
9. The maximum traffic/visits were observed on non-weekend days by various users.
10. There is a very less success rate seen as most of the users (84.4%) didnt complete the purchase.

In [26]:
```
# Distribution plots for certain columns having high data points

columns_to_plot = ['Administrative_Duration', 'Informational_Duration', 'Produ
sns.set(style="whitegrid")

fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(14, 16))
fig.subplots_adjust(hspace=0.5)

for i, column in enumerate(columns_to_plot):
    row = i // 2
    col = i % 2
    sns.histplot(df_shopping_uniq_copy[column], ax=axes[row, col], kde=True, c
    axes[row, col].set_title(f'Distribution of {column}')
    axes[row, col].set_xlabel(column)

# Removing the empty subplot
fig.delaxes(axes[3][1])

plt.show()
```

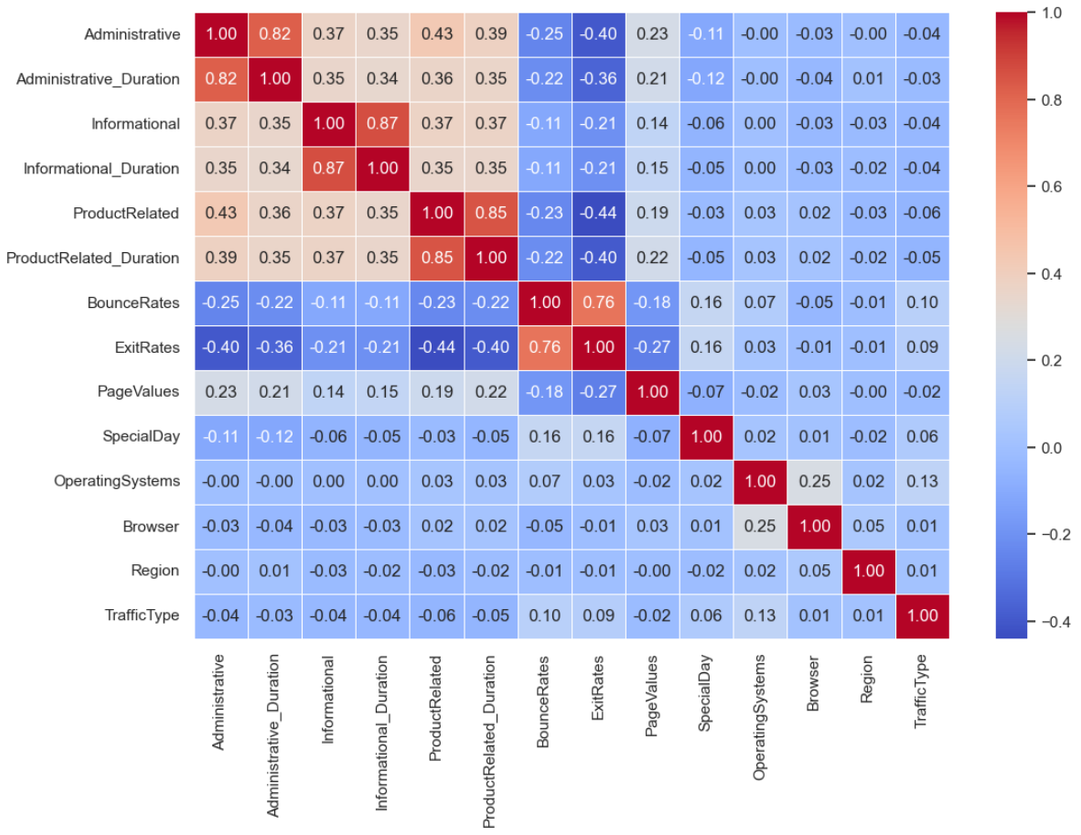



## Remarks: -

1. The max administrative_duration belong to 0.0 and the slope keeps on decreasing and at around 240 there is a rise seen.
2. The max informational_duration belong to 0.0 and the slope keeps on decreasing and at around 75 there is a rise seen.
3. The max number of similar pages the user visited lies max at 5 and the trend is un-uniform with 2nd highest at around 74.
4. The max amount of time spent in this category of pages is around 10.
5. The bounce and exit rates are max seen at 0 value. Same goes for page values.

# Correlation Analysis | Bivariate | Multivariate Analysis

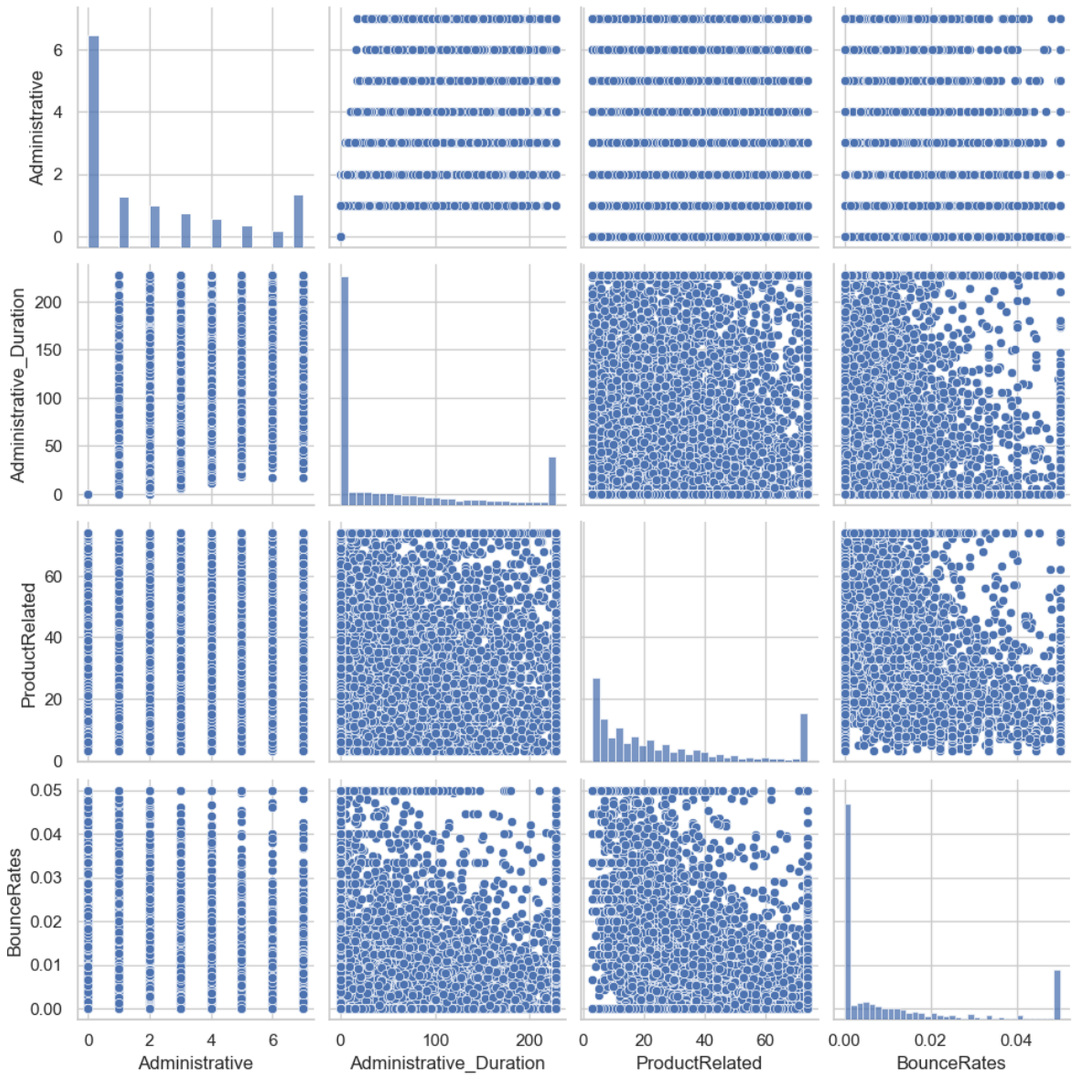
In [27]:
```
# Selecting numerical columns for correlation analysis
numerical_columns = df_shopping_uniq_copy.select_dtypes(include=['float64', 'i

correlation_matrix = df_shopping_uniq_copy[numerical_columns].corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewi
plt.show()
```



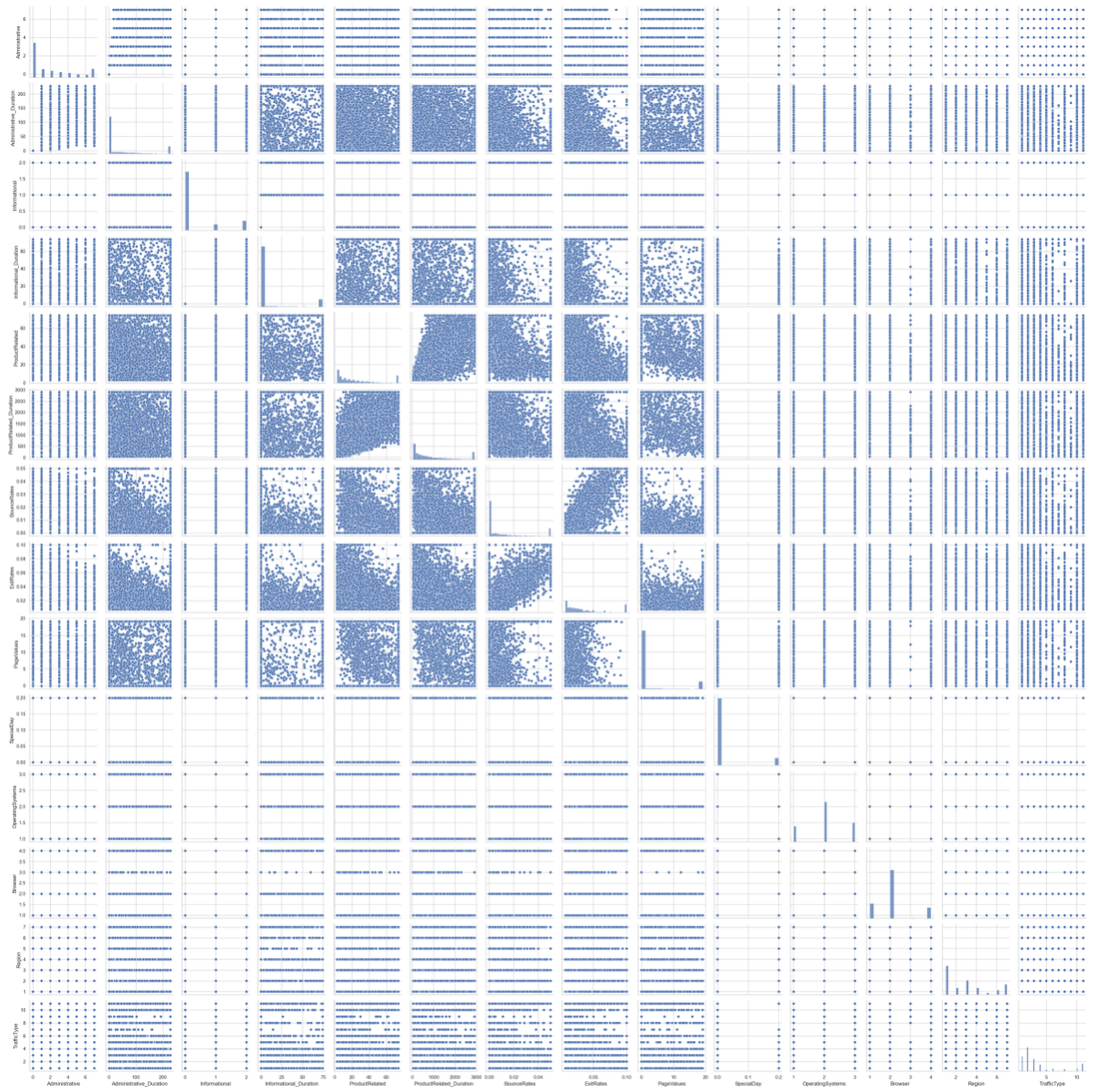

## Remarks

1. Informational and informational_duration are highly correlated
2. Highest negatively correlated is observed for productrelated and exit rates

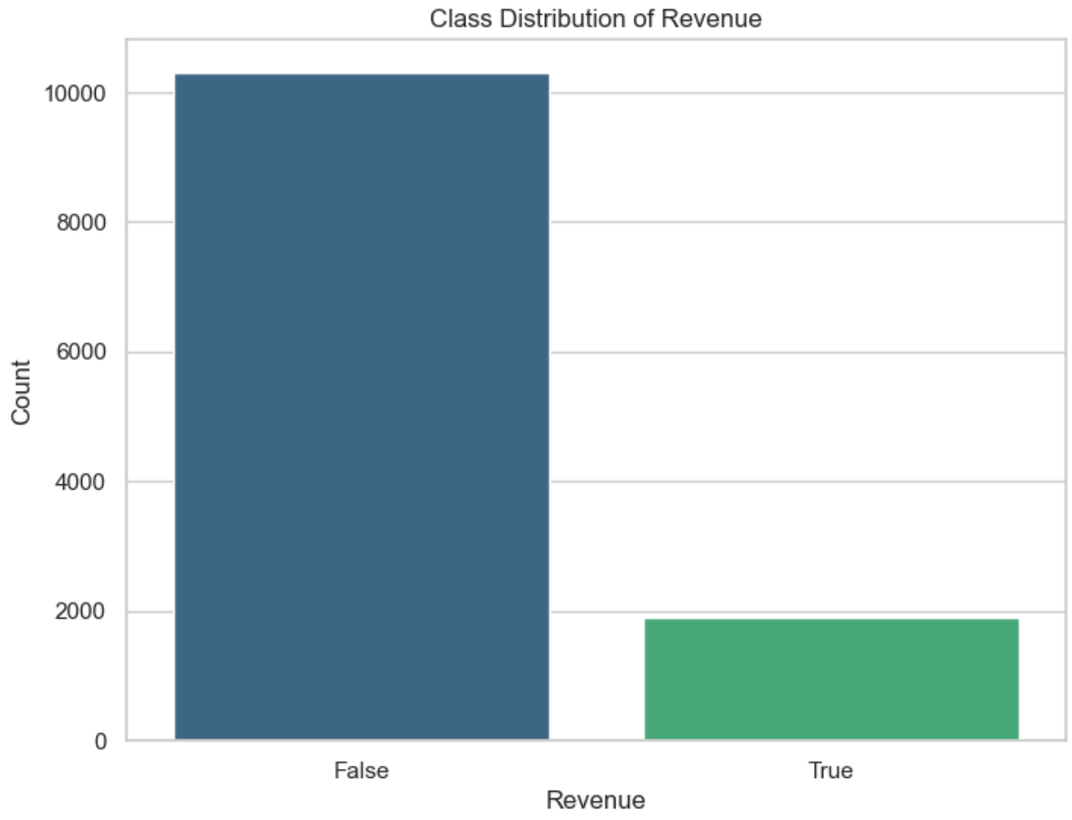
In [28]:
```
# Considering few columns for scatter plots
selected_columns = ['Administrative', 'Administrative_Duration', 'ProductRelat

sns.pairplot(df_shopping_uniq_copy[selected_columns])
plt.show()
```



In [29]:
```
# Pair plots for numerical realted columns
numerical_columns = df_shopping_uniq_copy.select_dtypes(include=['float64', 'i

sns.pairplot(df_shopping_uniq_copy[numerical_columns])
plt.show()
```



In [30]:
```
# Class Distribution: Check the distribution of the target variable ('Revenue'

plt.figure(figsize=(8, 6))
sns.countplot(x='Revenue', data=df_shopping_uniq_copy, palette='viridis')
plt.xlabel('Revenue')
plt.ylabel('Count')
plt.title('Class Distribution of Revenue')
plt.show()
```




In [31]:
```
# Ques-> Summarize page views, durations, and bounce/exit rates for each page

summary_df = df_shopping_uniq_copy.groupby('PageValues').agg({
    'ProductRelated': 'mean',
    'ProductRelated_Duration': 'mean',
    'BounceRates': 'mean',
    'ExitRates': 'mean'
}).reset_index()

print("Count of total rows:", summary_df['PageValues'].count())
summary_df.head()
```

```
Count of total rows: 1504
```

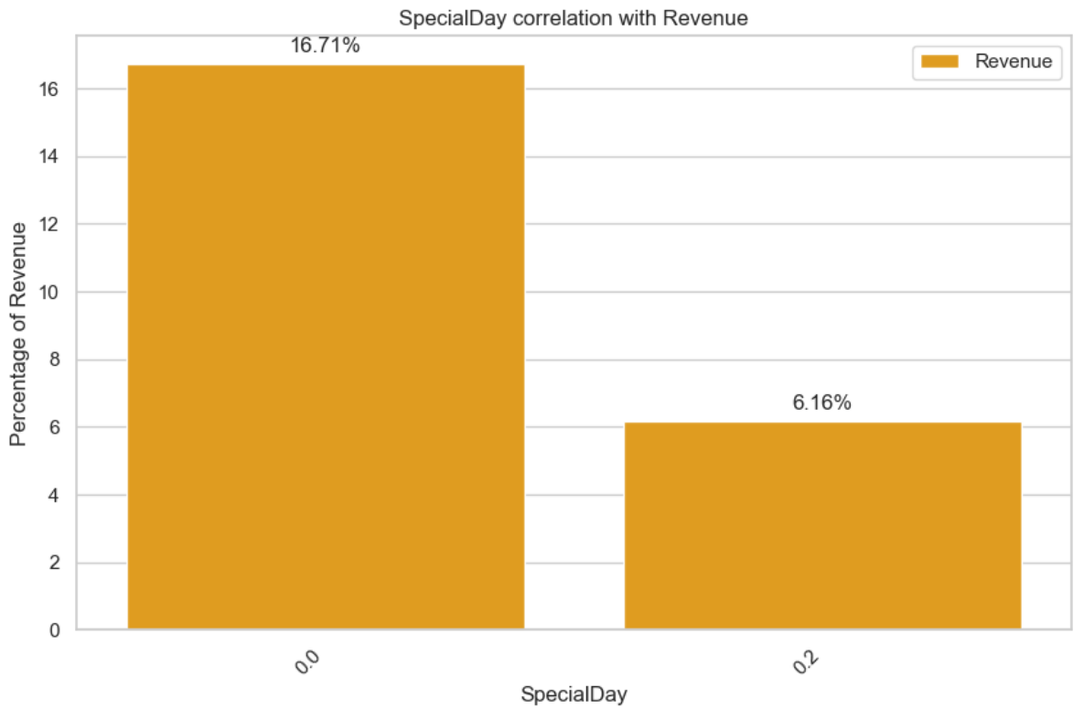
Out[31]:

| | PageValues | ProductRelated | ProductRelated_Duration | BounceRates | ExitRates |
|---|---|---|---|---|---|
| **0** | 0.000000 | 21.82058 | 780.774535 | 0.013185 | 0.040349 |
| **1** | 0.038035 | 74.00000 | 2904.459143 | 0.005995 | 0.020011 |
| **2** | 0.067050 | 74.00000 | 2904.459143 | 0.000000 | 0.012144 |
| **3** | 0.093547 | 69.00000 | 2269.732143 | 0.000000 | 0.007407 |
| **4** | 0.098621 | 74.00000 | 2904.459143 | 0.000039 | 0.007511 |

## Remarks

1. The total unique pagevalues formed are 1504 in number (grouped by).
2. The mean of each product oriented metrics corresponding to each unique pagevalue is present across the rows.

In [32]:
```
# Ques-> Analyze SpecialDay distribution and its correlation with Revenue.

percentage_df = df_shopping_uniq_copy.groupby('SpecialDay')['Revenue'].value_c

# Plotting a bar plot to analyze SpecialDay correlation with Revenue
plt.figure(figsize=(10, 6))
ax = sns.barplot(x=percentage_df.index, y=percentage_df[True], color='orange',
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')

# Displaying the percentage labels on each of the bars
for p in ax.patches:
    ax.annotate(f'{p.get_height():.2f}%', (p.get_x() + p.get_width() / 2., p.g
                ha='center', va='center', xytext=(0, 10), textcoords='offset p

plt.title('SpecialDay correlation with Revenue')
plt.xlabel('SpecialDay')
plt.ylabel('Percentage of Revenue')
plt.legend()
plt.show()
```




## Remarks

1. The highest % of revenue is earned/observed from non-special days

In [33]:
```
# Ques-> Generate a binary feature indicating whether the user visited all thr

# Creating a new column 'VisitedAllCategories' with binary values
df_shopping_uniq_copy['VisitedAllCategories'] = (
    (df_shopping_uniq_copy['Administrative'] > 0) &
    (df_shopping_uniq_copy['Informational'] > 0) &
    (df_shopping_uniq_copy['ProductRelated'] > 0)
).astype(int)


df_shopping_uniq_copy[['Administrative', 'Informational', 'ProductRelated', 'V
```

Out[33]:

| | Administrative | Informational | ProductRelated | VisitedAllCategories |
|---|---|---|---|---|
| **0** | 0.0 | 0.0 | 3.0 | 0 |
| **1** | 0.0 | 0.0 | 3.0 | 0 |
| **2** | 0.0 | 0.0 | 3.0 | 0 |
| **3** | 0.0 | 0.0 | 3.0 | 0 |
| **4** | 0.0 | 0.0 | 10.0 | 0 |

## Remarks

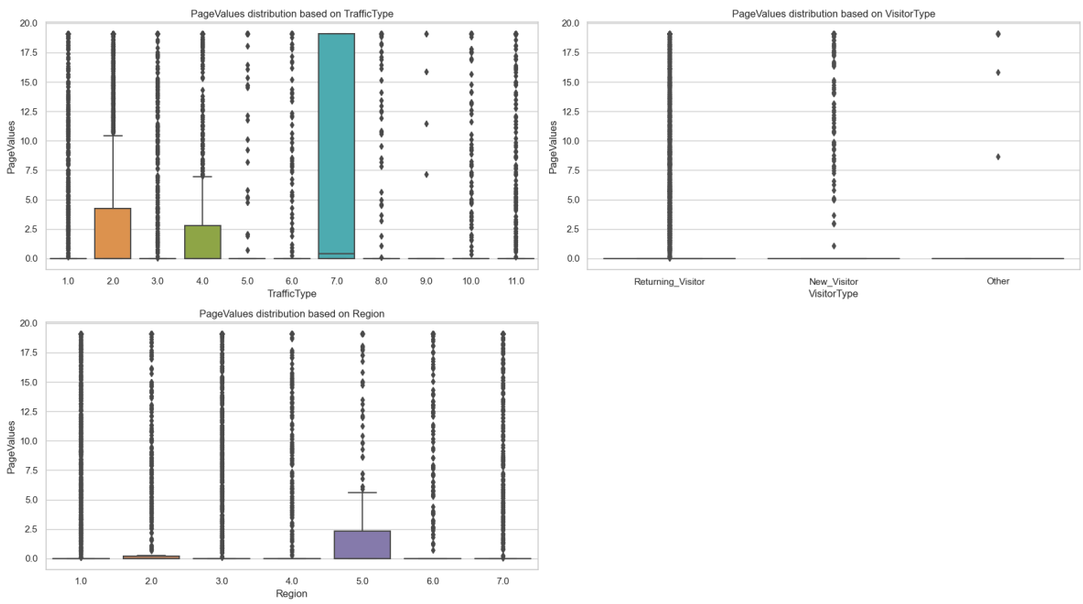
As per the question, the binary values have been asigned to all visited catrgories as "1" Here, the condition (df_shopping_uniq_copy['Administrative'] > 0) & (df_shopping_uniq_copy['Informational'] > 0) & (df_shopping_uniq_copy['ProductRelated'] > 0) checks if the user visited all three page categories by ensuring that the counts in each of the three columns are greater than zero. The resulting binary feature, 'VisitedAllCategories', will be 1 if the condition is met and 0 otherwise.

In [34]:
```
# Displaying few rows where VisitedAllCategories value = 1

filtered_df = df_shopping_uniq_copy[df_shopping_uniq_copy['VisitedAllCategorie
filtered_df[['Administrative', 'Informational', 'ProductRelated', 'VisitedAllC
```

Out[34]:

| | Administrative | Informational | ProductRelated | VisitedAllCategories |
|---|---|---|---|---|
| **29** | 1.0 | 1.0 | 45.0 | 1 |
| **57** | 4.0 | 2.0 | 36.0 | 1 |
| **103** | 2.0 | 1.0 | 36.0 | 1 |
| **109** | 6.0 | 2.0 | 74.0 | 1 |
| **161** | 2.0 | 2.0 | 31.0 | 1 |

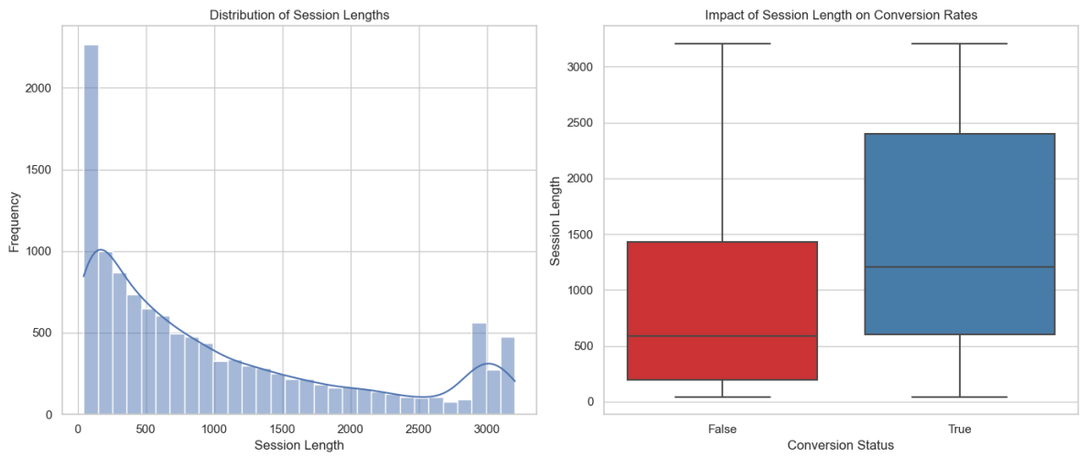
In [35]:
```
# Ques-> Explore PageValues distribution and its relationship with TrafficType

plt.figure(figsize=(18, 10))

# Plotting for PageValues distribution based on TrafficType
plt.subplot(2, 2, 1)
sns.boxplot(x='TrafficType', y='PageValues', data=df_shopping_uniq_copy)
plt.title('PageValues distribution based on TrafficType')

# Plotting for PageValues distribution based on VisitorType
plt.subplot(2, 2, 2)
sns.boxplot(x='VisitorType', y='PageValues', data=df_shopping_uniq_copy)
plt.title('PageValues distribution based on VisitorType')

# Plotting for PageValues distribution based on Region
plt.subplot(2, 2, 3)
sns.boxplot(x='Region', y='PageValues', data=df_shopping_uniq_copy)
plt.title('PageValues distribution based on Region')

plt.tight_layout()
plt.show()
```



## Remarks

1. The highest page value distribution w.r.t traffic type is observed for traffic type value = 7.0
2. More number of returning visitors is observed w.r.t relationship between visitor type and page value
3. The region 5.0 has highest users when compared w.r.t relationship with page values.

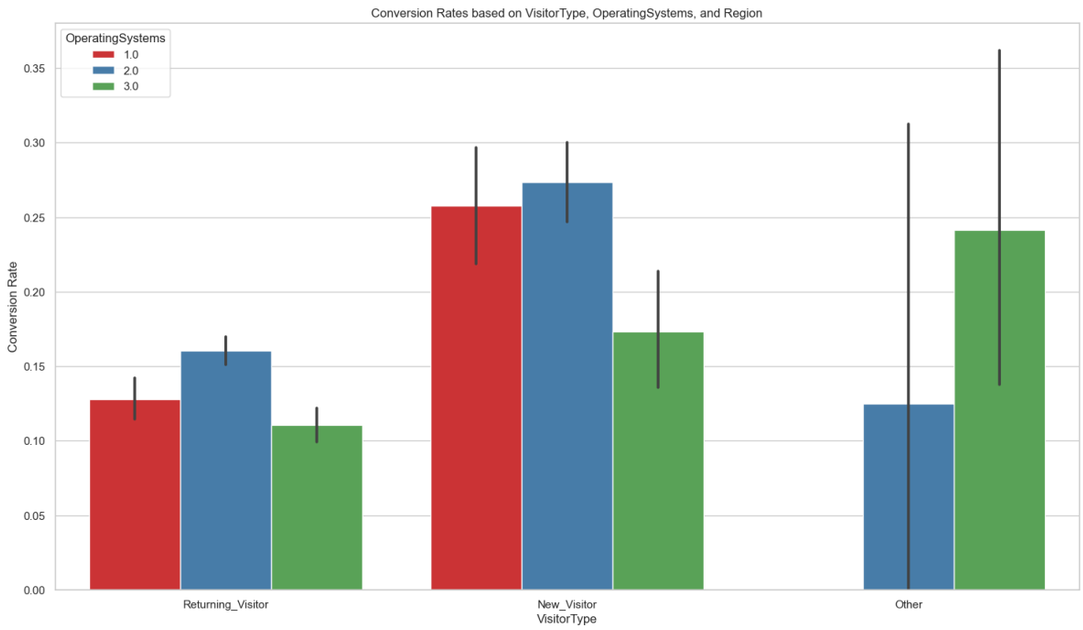
In [36]:
```
# Ques-> Investigate user session lengths and their impact on conversion rates

# Creating a new column for session length (sum of Administrative_Duration, In
df_shopping_uniq_copy['SessionLength'] = df_shopping_uniq_copy['Administrative
                                         df_shopping_uniq_copy['Informationa
                                         df_shopping_uniq_copy['ProductRelat

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
sns.histplot(df_shopping_uniq_copy['SessionLength'], bins=30, kde=True)
plt.title('Distribution of Session Lengths')
plt.xlabel('Session Length')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
sns.boxplot(x='Revenue', y='SessionLength', data=df_shopping_uniq_copy, palett
plt.title('Impact of Session Length on Conversion Rates')
plt.xlabel('Conversion Status')
plt.ylabel('Session Length')

plt.tight_layout()
plt.show()
```




## Remarks

1. The highest session length freq is observed around approx. 10mins
2. There is a positive relation where revenue increases with increase in session length with a median of around 1200.

In [37]:
```
# Ques-> Group users based on VisitorType, OperatingSystems, and Region to ide

plt.figure(figsize=(18, 10))

sns.barplot(x='VisitorType', y='Revenue', hue='OperatingSystems', data=df_shop
plt.title('Conversion Rates based on VisitorType, OperatingSystems, and Region
plt.xlabel('VisitorType')
plt.ylabel('Conversion Rate')

plt.show()
```

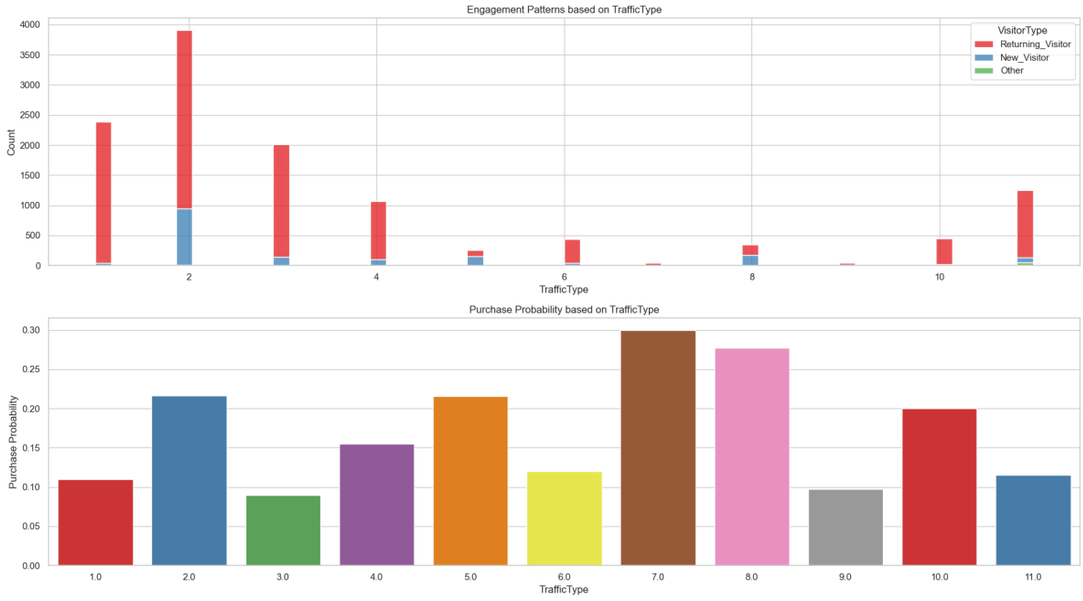



## Remarks

1. More number of 1.0 and 2.0 operating system user belong to new_visitor category
2. More number of 3.0 operating system user belong to other category

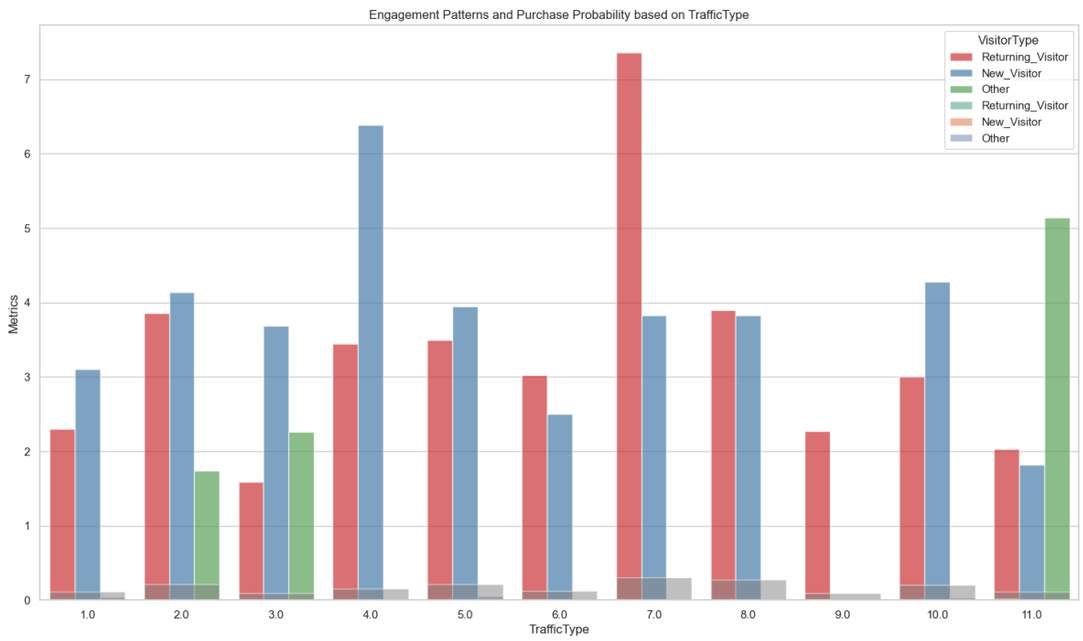
In [38]:
```
# Ques-> Segment users based on TrafficType and analyze their engagement patte

plt.figure(figsize=(18, 10))

plt.subplot(2, 1, 1)
sns.histplot(data=df_shopping_uniq_copy, x='TrafficType', hue='VisitorType', m
plt.title('Engagement Patterns based on TrafficType')
plt.xlabel('TrafficType')
plt.ylabel('Count')

plt.subplot(2, 1, 2)
sns.barplot(x='TrafficType', y='Revenue', data=df_shopping_uniq_copy, ci=None,
plt.title('Purchase Probability based on TrafficType')
plt.xlabel('TrafficType')
plt.ylabel('Purchase Probability')

plt.tight_layout()
plt.show()
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_19464\1770017106.py:12: FutureWarn
ing:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='TrafficType', y='Revenue', data=df_shopping_uniq_copy, ci=No
ne, palette='Set1')
```



In [39]:
```
plt.figure(figsize=(18, 10))

# Grouped bar plot for engagement patterns and purchase probability based on T
sns.barplot(x='TrafficType', y='PageValues', hue='VisitorType', data=df_shoppi
sns.barplot(x='TrafficType', y='BounceRates', hue='VisitorType', data=df_shopp
sns.barplot(x='TrafficType', y='Revenue', data=df_shopping_uniq_copy, ci=None,

# Display the plot
plt.title('Engagement Patterns and Purchase Probability based on TrafficType')
plt.xlabel('TrafficType')
plt.ylabel('Metrics')
plt.legend(title='VisitorType', loc='upper right')
plt.show()
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_19464\2660585155.py:4: FutureWarni
ng:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='TrafficType', y='PageValues', hue='VisitorType', data=df_sho
pping_uniq_copy, ci=None, palette='Set1', alpha=0.7)
C:\Users\Dell\AppData\Local\Temp\ipykernel_19464\2660585155.py:5: FutureWarni
ng:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='TrafficType', y='BounceRates', hue='VisitorType', data=df_sh
opping_uniq_copy, ci=None, palette='Set2', alpha=0.7)
C:\Users\Dell\AppData\Local\Temp\ipykernel_19464\2660585155.py:6: FutureWarni
ng:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='TrafficType', y='Revenue', data=df_shopping_uniq_copy, ci=No
ne, color='grey', alpha=0.5)
```




### Remarks

1. The above 2 graphs are 2 different ways to represent the usecase and the graphs plotted are clearly signifying that max users will be from 2.0 operating system having traffic type category as 7.0

## Recommendations

1. Bounce Rates and Exit Rates Analysis: High bounce rates might indicate that visitors are not finding what theyre looking for. Considering optimizing the landing pages and improving website navigation. Analyzing exit rates to identify pages where visitors commonly leave.

Optimization of these pages to encourage further engagement.

2. PageValues: High page values suggest that certain pages contribute more to conversions. Identifying these pages and investing in their optimization to enhance conversion rates.
3. Visitor Type: Understanding the behavior of returning visitors versus new visitors. Tailoring marketing strategies to engage both types effectively.
4. Weekend vs. Weekday Behavior: Analyzing how visitor behavior differs on weekends compared to weekdays. Adjusting the marketing campaigns and promotions based on this insight.
5. Month-wise Analysis: Identifying trends or seasonality in conversion rates across different months. Planning marketing campaigns accordingly to leverage peak periods.
6. Device and Browser Compatibility: Analyzing the data on different operating systems and browsers. Ensuring that the website is optimized for the most popular ones to provide a seamless user experience.
7. Geographical Insights (Region): Understanding user behavior based on geographical regions. Tailoring marketing messages and promotions to local preferences or cultural trends.
8. Traffic Sources (TrafficType): Analyzing the effectiveness of different traffic sources. Investment more in channels that drive high-quality traffic and conversions.
9. Special Days Impact: Assessing how special days impact user behavior. Planning marketing promotions and campaigns around these days to maximize engagement.
10. Revenue Analysis: Understanding the factors contributing to revenue generation. Then focussing on marketing efforts for products or services that have a higher impact on overall revenue.

EOF

In [1]:
```
# Importing required packages to be used

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
from datetime import datetime
from IPython.display import display
from scipy.stats import chi2_contingency
from scipy.stats import ttest_ind
from scipy.stats import pearsonr
from scipy.stats import f_oneway
```

In [2]:
```
# Importing and Reading top 10 data

df_campaign=pd.read_csv('campaign.csv')
df_campaign.head(10)
```

Out[2]:

| | ID | Year_Birth | Education | Marital_Status | Income | Kidhome | Teenhome | Dt_Customer | Recency | MntWines | ... | NumCatalogPurchases | NumStorePur |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1826 | 1970 | Graduation | Divorced | $84,835.00 | 0 | 0 | 6/16/14 | 0 | 189 | ... | 4 | |
| **1** | 1 | 1961 | Graduation | Single | $57,091.00 | 0 | 0 | 6/15/14 | 0 | 464 | ... | 3 | |
| **2** | 10476 | 1958 | Graduation | Married | $67,267.00 | 0 | 1 | 5/13/14 | 0 | 134 | ... | 2 | |
| **3** | 1386 | 1967 | Graduation | Together | $32,474.00 | 1 | 1 | 5/11/14 | 0 | 10 | ... | 0 | |
| **4** | 5371 | 1989 | Graduation | Single | $21,474.00 | 1 | 0 | 4/8/14 | 0 | 6 | ... | 1 | |
| **5** | 7348 | 1958 | PhD | Single | $71,691.00 | 0 | 0 | 3/17/14 | 0 | 336 | ... | 7 | |
| **6** | 4073 | 1954 | 2n Cycle | Married | $63,564.00 | 0 | 0 | 1/29/14 | 0 | 769 | ... | 10 | |
| **7** | 1991 | 1967 | Graduation | Together | $44,931.00 | 0 | 1 | 1/18/14 | 0 | 78 | ... | 1 | |
| **8** | 4047 | 1954 | PhD | Married | $65,324.00 | 0 | 1 | 1/11/14 | 0 | 384 | ... | 2 | |
| **9** | 9477 | 1954 | PhD | Married | $65,324.00 | 0 | 1 | 1/11/14 | 0 | 384 | ... | 2 | |

10 rows × 27 columns

## General Analysis

In [3]:
```
# Defining the shape and dimension of data

a = df_campaign.shape
b = df_campaign.ndim
print("Shape-->", a,'\n',"Dimension-->",b)
```

```
Shape--> (2239, 27)
 Dimension--> 2
```

### Remarks: -

There are 2239 rows and 27 columns present in the dataset. Its 2-d dataset by nature

In [4]:
```
# Checking general info of cols

df_campaign.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2239 entries, 0 to 2238
Data columns (total 27 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   2239 non-null   int64
 1   Year_Birth           2239 non-null   int64
 2   Education            2239 non-null   object
 3   Marital_Status       2239 non-null   object
 4   Income               2239 non-null   object
 5   Kidhome              2239 non-null   int64
 6   Teenhome             2239 non-null   int64
 7   Dt_Customer          2239 non-null   object
 8   Recency              2239 non-null   int64
 9   MntWines             2239 non-null   int64
 10  MntFruits            2239 non-null   int64
 11  MntMeatProducts      2239 non-null   int64
 12  MntFishProducts      2239 non-null   int64
 13  MntSweetProducts     2239 non-null   int64
 14  MntGoldProds         2239 non-null   int64
 15  NumDealsPurchases    2239 non-null   int64
 16  NumWebPurchases      2239 non-null   int64
 17  NumCatalogPurchases  2239 non-null   int64
 18  NumStorePurchases    2239 non-null   int64
 19  NumWebVisitsMonth    2239 non-null   int64
 20  AcceptedCmp3         2239 non-null   int64
 21  AcceptedCmp4         2239 non-null   int64
 22  AcceptedCmp5         2239 non-null   int64
 23  AcceptedCmp1         2239 non-null   int64
 24  AcceptedCmp2         2239 non-null   int64
 25  Complain             2239 non-null   int64
 26  Country              2239 non-null   object
dtypes: int64(22), object(5)
memory usage: 472.4+ KB
```

### Remarks: -

1. There are 5 string type columns
2. There are 22 int type columns

In [5]:
```
# Overall stats of entire dataset

df_campaign.describe(include="all")
```

Out[5]:

| | ID | Year_Birth | Education | Marital_Status | Income | Kidhome | Teenhome | Dt_Customer | Recency | MntWines | ... | NumCatalogPur |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 2239.000000 | 2239.000000 | 2239 | 2239 | 2239 | 2239.000000 | 2239.000000 | 2239 | 2239.000000 | 2239.000000 | ... | 2239 |
| **unique** | NaN | NaN | 5 | 8 | 1974 | NaN | NaN | 663 | NaN | NaN | ... | |
| **top** | NaN | NaN | Graduation | Married | $nan | NaN | NaN | 8/31/12 | NaN | NaN | ... | |
| **freq** | NaN | NaN | 1126 | 864 | 24 | NaN | NaN | 12 | NaN | NaN | ... | |
| **mean** | 5590.444841 | 1968.802144 | NaN | NaN | NaN | 0.443948 | 0.506476 | NaN | 49.121036 | 304.067441 | ... | 2 |
| **std** | 3246.372471 | 11.985494 | NaN | NaN | NaN | 0.538390 | 0.544555 | NaN | 28.963662 | 336.614830 | ... | 2 |
| **min** | 0.000000 | 1893.000000 | NaN | NaN | NaN | 0.000000 | 0.000000 | NaN | 0.000000 | 0.000000 | ... | 0 |
| **25%** | 2827.500000 | 1959.000000 | NaN | NaN | NaN | 0.000000 | 0.000000 | NaN | 24.000000 | 24.000000 | ... | 0 |
| **50%** | 5455.000000 | 1970.000000 | NaN | NaN | NaN | 0.000000 | 0.000000 | NaN | 49.000000 | 174.000000 | ... | 2 |
| **75%** | 8423.500000 | 1977.000000 | NaN | NaN | NaN | 1.000000 | 1.000000 | NaN | 74.000000 | 504.500000 | ... | 4 |
| **max** | 11191.000000 | 1996.000000 | NaN | NaN | NaN | 2.000000 | 2.000000 | NaN | 99.000000 | 1493.000000 | ... | 28 |

11 rows × 27 columns

In [6]:
```
df_campaign.describe(include= object).T
```

Out[6]:

| | count | unique | top | freq |
|---|---|---|---|---|
| **Education** | 2239 | 5 | Graduation | 1126 |
| **Marital_Status** | 2239 | 8 | Married | 864 |
| **Income** | 2239 | 1974 | $nan | 24 |
| **Dt_Customer** | 2239 | 663 | 8/31/12 | 12 |
| **Country** | 2239 | 8 | SP | 1095 |

### Remarks: -

1. This is solely targetting the object datatype columns overall statistics
2. The topmost consecutive value and its freq of each of the column values are present in the last 2 columns (in above result)

In [7]:
```
# Null/Empty values in dataset

df_campaign.isna().sum().sort_values(ascending=False)
```

Out[7]:
```
ID                   0
MntGoldProds         0
Complain             0
AcceptedCmp2         0
AcceptedCmp1         0
AcceptedCmp5         0
AcceptedCmp4         0
AcceptedCmp3         0
NumWebVisitsMonth    0
NumStorePurchases    0
NumCatalogPurchases  0
NumWebPurchases      0
NumDealsPurchases    0
MntSweetProducts     0
Year_Birth           0
MntFishProducts      0
MntMeatProducts      0
MntFruits            0
MntWines             0
Recency              0
Dt_Customer          0
Teenhome             0
Kidhome              0
Income               0
Marital_Status       0
Education            0
Country              0
dtype: int64
```

### Remarks: -

1. This is a check on the number of nulls/empty values present in the entire dataset
2. There are no nulls/empty values present in the dataset for any of the column values

In [8]:
```
# Finding duplicate rows based on all columns

duplicate_rows = df_campaign[df_campaign.duplicated(keep=False)]

# Displaying the duplicate rows
print("Duplicate Rows:")
print(duplicate_rows)

# Counting the total number of duplicate rows
num_duplicates = len(duplicate_rows)

print(f"Total number of duplicate rows: {num_duplicates}")
```

```
Duplicate Rows:
Empty DataFrame
Columns: [ID, Year_Birth, Education, Marital_Status, Income, Kidhome, Teenhome, Dt_Customer, Recency, MntWines, MntFruits, Mn
tMeatProducts, MntFishProducts, MntSweetProducts, MntGoldProds, NumDealsPurchases, NumWebPurchases, NumCatalogPurchases, NumS
torePurchases, NumWebVisitsMonth, AcceptedCmp3, AcceptedCmp4, AcceptedCmp5, AcceptedCmp1, AcceptedCmp2, Complain, Country]
Index: []

[0 rows x 27 columns]
Total number of duplicate rows: 0
```

### Remarks

There are 0 duplicates present

In [9]:
```
# Checking for value ranges (e.g., numeric columns should not have negative values)

numeric_columns = df_campaign.select_dtypes(include=['int', 'float']).columns
value_range_issues = (df_campaign[numeric_columns] < 0).any()

print("\nValue Range Issues:")
print(value_range_issues)
```

```
Value Range Issues:
ID                   False
Year_Birth           False
Kidhome              False
Teenhome             False
Recency              False
MntWines             False
MntFruits            False
MntMeatProducts      False
MntFishProducts      False
MntSweetProducts     False
MntGoldProds         False
NumDealsPurchases    False
NumWebPurchases      False
NumCatalogPurchases  False
NumStorePurchases    False
NumWebVisitsMonth    False
AcceptedCmp3         False
AcceptedCmp4         False
AcceptedCmp5         False
AcceptedCmp1         False
AcceptedCmp2         False
Complain             False
dtype: bool
```

### Remarks: -

1. This is for checking if there is any oddity or unexpected data/anomalies present under the integer columns
2. There are no anomaly values and only numeric data is present across all the rows for these integer columns

## Non-Graphical Analysis

### Uniques and Value counts

In [10]:
```
# Unique values for Year_Birth

print("Unique values are: -")
print(df_campaign['Year_Birth'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['Year_Birth'].value_counts().reset_index().to_string(index=False, header=['Year_Birth', 'Count']))
```

```
Unique values are: -
[1970 1961 1958 1967 1989 1954 1947 1979 1959 1981 1969 1977 1960 1966
 1976 1965 1956 1975 1971 1986 1972 1974 1990 1987 1984 1968 1955 1983
 1973 1978 1952 1962 1964 1982 1963 1957 1980 1945 1949 1948 1953 1946
 1985 1992 1944 1951 1988 1950 1994 1993 1991 1893 1996 1995 1899 1943
 1941 1940 1900]

Value counts are: -
Year_Birth Count
      1976    89
      1971    87
      1975    83
      1972    79
      1970    77
      1978    77
      1965    74
      1973    74
      1969    71
      1974    69
      1956    55
      1979    53
      1958    53
      1952    52
      1959    51
      1977    51
      1968    51
      1966    50
      1954    50
      1955    49
      1960    49
      1982    45
      1963    45
      1967    44
      1962    44
      1957    43
      1951    43
      1964    42
      1983    42
      1986    42
      1980    39
      1981    39
      1984    38
      1961    36
      1953    35
      1985    32
      1989    30
      1949    30
      1950    29
      1988    29
      1987    27
      1948    21
      1990    18
      1947    16
      1946    16
      1991    15
      1992    13
      1945     8
      1944     7
      1943     7
      1993     5
      1995     5
      1994     3
      1996     2
      1893     1
      1899     1
      1941     1
      1940     1
      1900     1
```

In [11]:
```
# Unique values for Education

print("Unique values are: -")
print(df_campaign['Education'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['Education'].value_counts().reset_index().to_string(index=False, header=['Education', 'Count']))
```

```
Unique values are: -
['Graduation' 'PhD' '2n Cycle' 'Master' 'Basic']

Value counts are: -
 Education Count
Graduation  1126
       PhD   486
    Master   370
  2n Cycle   203
     Basic    54
```

In [12]:
```
# Unique values for Marital_Status

print("Unique values are: -")
print(df_campaign['Marital_Status'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['Marital_Status'].value_counts().reset_index().to_string(index=False, header=['Marital_Status', 'Count']))
```

```
Unique values are: -
['Divorced' 'Single' 'Married' 'Together' 'Widow' 'YOLO' 'Alone' 'Absurd']

Value counts are: -
Marital_Status Count
       Married   864
      Together   579
        Single   480
      Divorced   232
         Widow    77
         Alone     3
          YOLO     2
        Absurd     2
```

In [13]:
```
# Unique values for Kidhome

print("Unique values are: -")
print(df_campaign['Kidhome'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['Kidhome'].value_counts().reset_index().to_string(index=False, header=['Kidhome', 'Count']))
```

```
Unique values are: -
[0 1 2]

Value counts are: -
Kidhome Count
      0  1293
      1   898
      2    48
```

In [14]:
```
# Unique values for Teenhome

print("Unique values are: -")
print(df_campaign['Teenhome'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['Teenhome'].value_counts().reset_index().to_string(index=False, header=['Teenhome', 'Count']))
```

```
Unique values are: -
[0 1 2]

Value counts are: -
Teenhome Count
       0  1157
       1  1030
       2    52
```

In [15]:
```
# Unique values for NumCatalogPurchases

print("Unique values are: -")
print(df_campaign['NumCatalogPurchases'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['NumCatalogPurchases'].value_counts().reset_index().to_string(index=False, header=['NumCatalogPurchases', 'C
```

```
Unique values are: -
[ 4  3  2  0  1  7 10  6  8  5  9 11 28 22]

Value counts are: -
NumCatalogPurchases Count
                  0   586
                  1   496
                  2   276
                  3   184
                  4   182
                  5   140
                  6   128
                  7    79
                  8    55
                 10    48
                  9    42
                 11    19
                 28     3
                 22     1
```

In [16]:
```
# Unique values for NumStorePurchases

print("Unique values are: -")
print(df_campaign['NumStorePurchases'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['NumStorePurchases'].value_counts().reset_index().to_string(index=False, header=['NumStorePurchases', 'Count
```

```
Unique values are: -
[ 6  7  5  2  3  9 10  0  8  4 13 12  1 11]

Value counts are: -
NumStorePurchases Count
                3   489
                4   323
                2   223
                5   212
                6   178
                8   149
                7   143
               10   125
                9   106
               12   105
               13    83
               11    81
                0    15
                1     7
```

In [17]:
```
# Unique values for NumWebVisitsMonth

print("Unique values are: -")
print(df_campaign['NumWebVisitsMonth'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['NumWebVisitsMonth'].value_counts().reset_index().to_string(index=False, header=['NumWebVisitsMonth', 'Count
```

```
Unique values are: -
[ 1  5  2  7  6  4  8  3  9  0 17 13 10 14 19 20]

Value counts are: -
NumWebVisitsMonth Count
                7   393
                8   342
                6   339
                5   281
                4   218
                3   205
                2   202
                1   153
                9    83
                0    11
               10     3
               20     3
               14     2
               19     2
               17     1
               13     1
```

In [18]:
```
# Unique values for AcceptedCmp3

print("Unique values are: -")
print(df_campaign['AcceptedCmp3'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['AcceptedCmp3'].value_counts().reset_index().to_string(index=False, header=['AcceptedCmp3', 'Count']))
```

```
Unique values are: -
[0 1]

Value counts are: -
AcceptedCmp3 Count
           0  2076
           1   163
```

In [19]:
```
# Unique values for AcceptedCmp5

print("Unique values are: -")
print(df_campaign['AcceptedCmp5'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['AcceptedCmp5'].value_counts().reset_index().to_string(index=False, header=['AcceptedCmp5', 'Count']))
```

```
Unique values are: -
[0 1]

Value counts are: -
AcceptedCmp5 Count
           0  2076
           1   163
```

In [20]:
```
# Unique values for AcceptedCmp1

print("Unique values are: -")
print(df_campaign['AcceptedCmp1'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['AcceptedCmp1'].value_counts().reset_index().to_string(index=False, header=['AcceptedCmp1', 'Count']))
```

```
Unique values are: -
[0 1]

Value counts are: -
AcceptedCmp1 Count
           0  2095
           1   144
```

In [21]:
```
# Unique values for AcceptedCmp2

print("Unique values are: -")
print(df_campaign['AcceptedCmp2'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['AcceptedCmp2'].value_counts().reset_index().to_string(index=False, header=['AcceptedCmp2', 'Count']))
```

```
Unique values are: -
[0 1]

Value counts are: -
AcceptedCmp2 Count
           0  2209
           1    30
```

In [22]:
```
# Unique values for Complain

print("Unique values are: -")
print(df_campaign['Complain'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['Complain'].value_counts().reset_index().to_string(index=False, header=['Complain', 'Count']))
```

```
Unique values are: -
[0 1]

Value counts are: -
Complain Count
       0  2218
       1    21
```

In [23]:
```
# Unique values for Country

print("Unique values are: -")
print(df_campaign['Country'].unique())

# Value counts
print("\nValue counts are: -")
print(df_campaign['Country'].value_counts().reset_index().to_string(index=False, header=['Country', 'Count']))
```

```
Unique values are: -
['SP' 'CA' 'US' 'AUS' 'GER' 'IND' 'SA' 'ME']

Value counts are: -
Country Count
     SP  1095
     SA   336
     CA   268
    AUS   160
    IND   148
    GER   120
     US   109
     ME     3
```

## Univariate Analysis

### Outlier Checks and Treatment

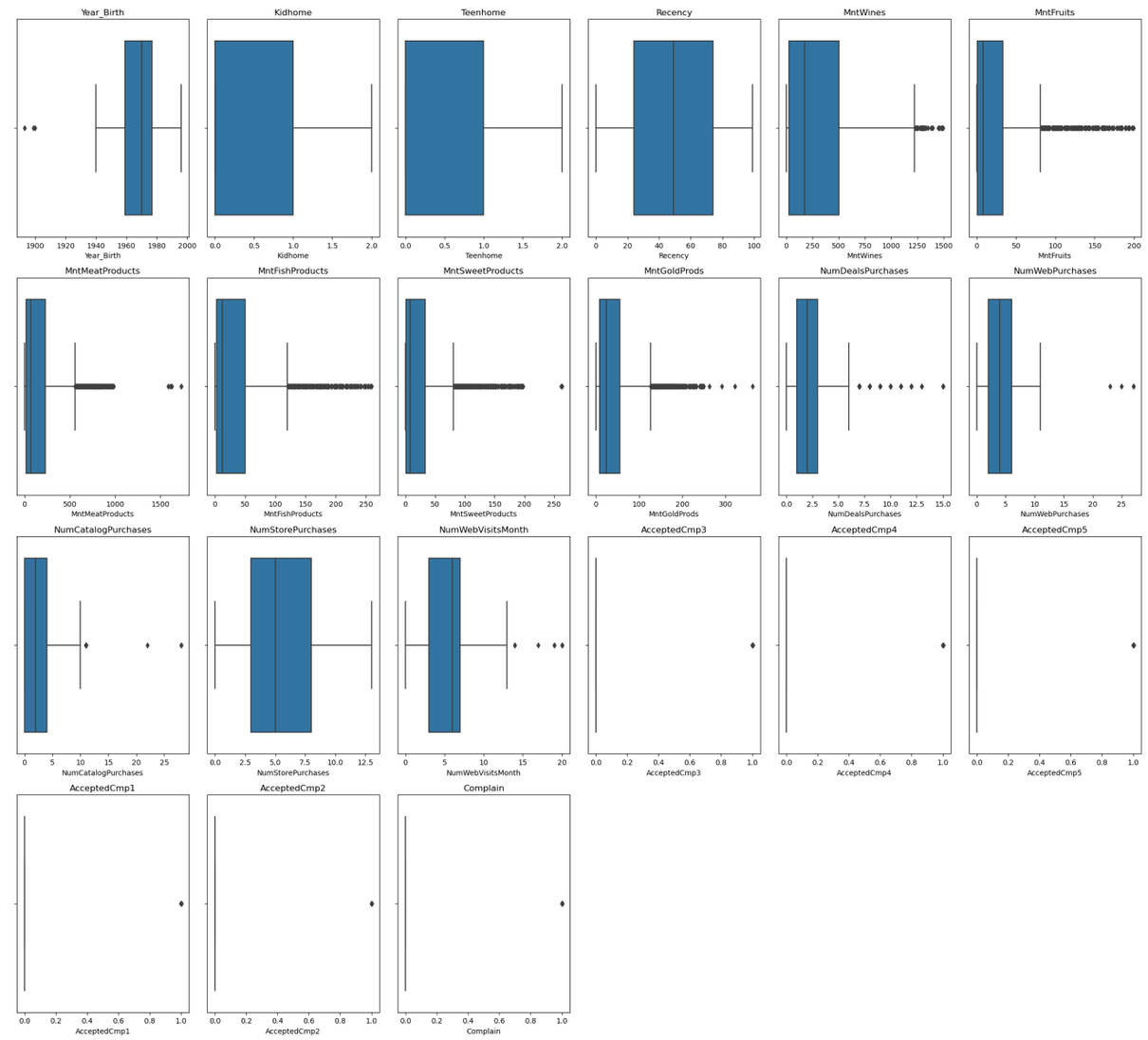
In [24]:
```
# Creating a backup of original database before removing outliers
df_campaign_copy=df_campaign.copy()
df_campaign_copy.head()
```

Out[24]:

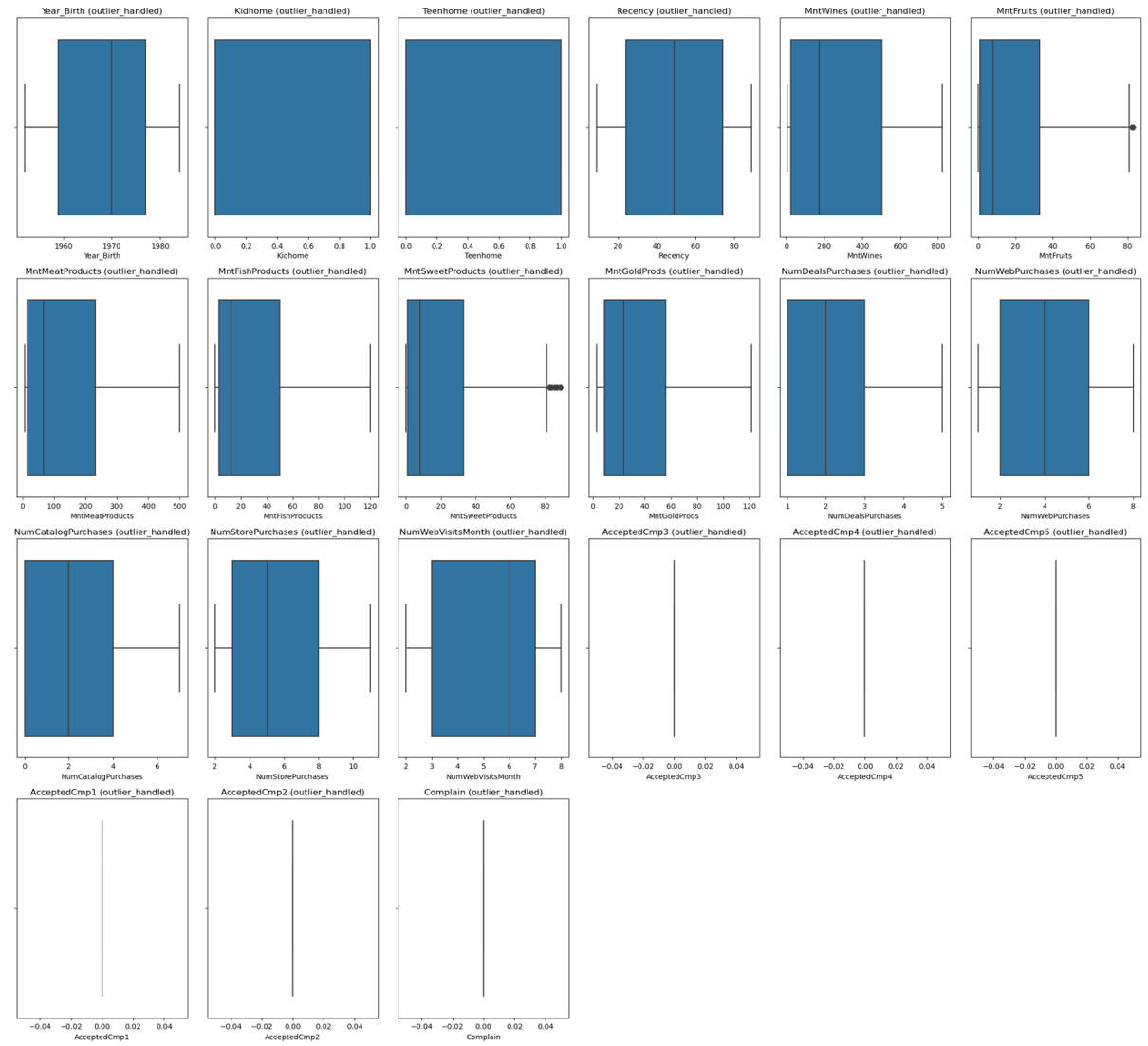
| | ID | Year_Birth | Education | Marital_Status | Income | Kidhome | Teenhome | Dt_Customer | Recency | MntWines | ... | NumCatalogPurchases | NumStorePur |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1826 | 1970 | Graduation | Divorced | $84,835.00 | 0 | 0 | 6/16/14 | 0 | 189 | ... | 4 | |
| **1** | 1 | 1961 | Graduation | Single | $57,091.00 | 0 | 0 | 6/15/14 | 0 | 464 | ... | 3 | |
| **2** | 10476 | 1958 | Graduation | Married | $67,267.00 | 0 | 1 | 5/13/14 | 0 | 134 | ... | 2 | |
| **3** | 1386 | 1967 | Graduation | Together | $32,474.00 | 1 | 1 | 5/11/14 | 0 | 10 | ... | 0 | |
| **4** | 5371 | 1989 | Graduation | Single | $21,474.00 | 1 | 0 | 4/8/14 | 0 | 6 | ... | 1 | |

5 rows × 27 columns

In [25]:
```
# Checking outliers for all the columns

columns_of_interest = ["Year_Birth","Kidhome","Teenhome","Recency","MntWines","MntFruits","MntMeatProducts","MntFishProducts",
fig, axes = plt.subplots(4, 6, figsize=(22, 20))
axes = axes.flatten()

for i, column in enumerate(columns_of_interest):
    sns.boxplot(data=df_campaign_copy, x=column, orient='h', ax=axes[i])
    axes[i].set_title(f"{column}")

# Removing the last 3 empty subplots
for i in range(len(columns_of_interest), len(axes)):
    fig.delaxes(axes[i])

plt.tight_layout()

plt.show()
```



In [26]:
```
# Handling the outliers

columns_of_interest = ["Year_Birth","Kidhome","Teenhome","Recency","MntWines","MntFruits","MntMeatProducts","MntFishProducts",

fig, axes = plt.subplots(4, 6, figsize=(22, 20))

axes = axes.flatten()

for i, column in enumerate(columns_of_interest):
    # Calculating the 10th and 90th percentiles
    tenth_percentile = df_campaign_copy[column].quantile(0.10)
    ninetieth_percentile = df_campaign_copy[column].quantile(0.90)

    # Applying quantile-based flooring and capping
    df_campaign_copy[column] = np.where(df_campaign_copy[column] < tenth_percentile, tenth_percentile, df_campaign_copy[column
    df_campaign_copy[column] = np.where(df_campaign_copy[column] > ninetieth_percentile, ninetieth_percentile, df_campaign_cop

    # Creating a boxplot of winsorized data
    sns.boxplot(data=df_campaign_copy, x=column, orient='h', ax=axes[i])
    axes[i].set_title(f"{column} (outlier_handled)")

# Removing the last 3 empty subplots
for i in range(len(columns_of_interest), len(axes)):
    fig.delaxes(axes[i])

plt.tight_layout()

plt.show()
```



## Graphical | Univariate analysis

In [27]:
```
# Feature engineering for the required analysis

# Assuming 'Income' column needs to be converted to numeric
df_campaign_copy['Income'] = df_campaign_copy['Income'].replace('[\$,]', '', regex=True).astype(float)

# Checking if 'Dt_Customer' column exists
if 'Dt_Customer' in df_campaign_copy.columns:
# Feature engineering for date-related columns
    df_campaign_copy['Dt_Customer'] = pd.to_datetime(df_campaign_copy['Dt_Customer'])
    df_campaign_copy['Year_Joined'] = df_campaign_copy['Dt_Customer'].dt.year
    df_campaign_copy['Month_Joined'] = df_campaign_copy['Dt_Customer'].dt.month

# Feature engineering for age
current_year = datetime.now().year
df_campaign_copy['Age'] = current_year - df_campaign_copy['Year_Birth']

# Display the modified dataframe
df_campaign_copy.head()
```

Out[27]:

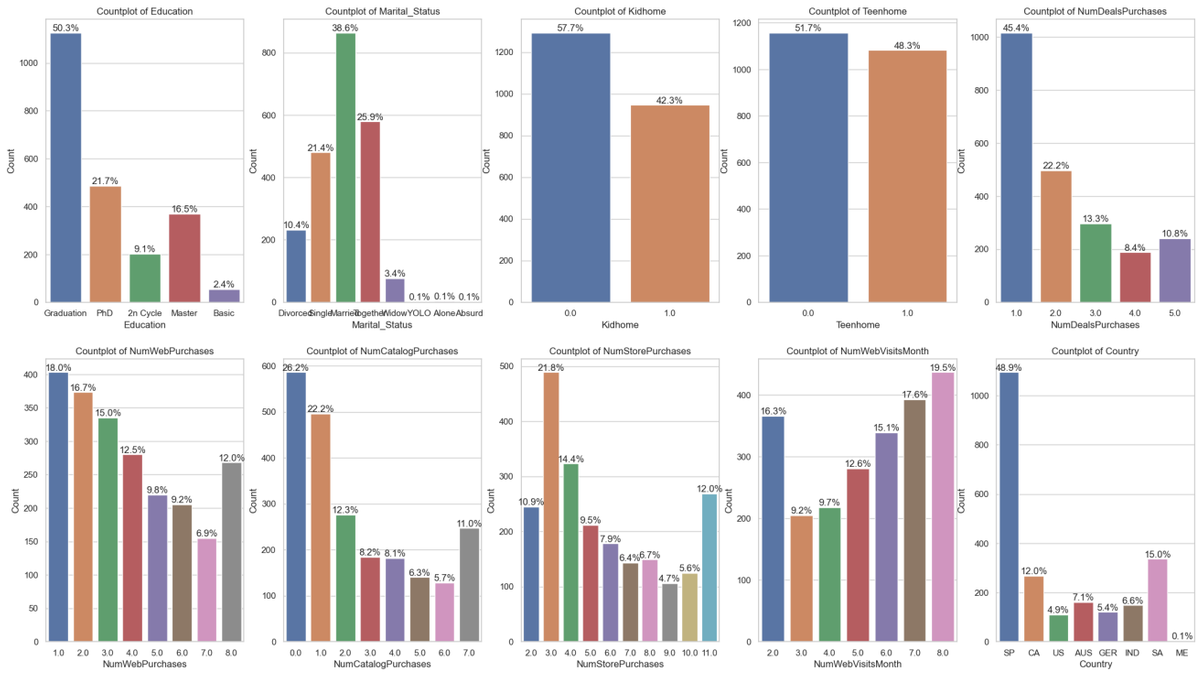
| | ID | Year_Birth | Education | Marital_Status | Income | Kidhome | Teenhome | Dt_Customer | Recency | MntWines | ... | AcceptedCmp3 | AcceptedCmp4 | Accep |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1826 | 1970.0 | Graduation | Divorced | 84835.0 | 0.0 | 0.0 | 2014-06-16 | 9.0 | 189.0 | ... | 0.0 | 0.0 | |
| **1** | 1 | 1961.0 | Graduation | Single | 57091.0 | 0.0 | 0.0 | 2014-06-15 | 9.0 | 464.0 | ... | 0.0 | 0.0 | |
| **2** | 10476 | 1958.0 | Graduation | Married | 67267.0 | 0.0 | 1.0 | 2014-05-13 | 9.0 | 134.0 | ... | 0.0 | 0.0 | |
| **3** | 1386 | 1967.0 | Graduation | Together | 32474.0 | 1.0 | 1.0 | 2014-05-11 | 9.0 | 10.0 | ... | 0.0 | 0.0 | |
| **4** | 5371 | 1984.0 | Graduation | Single | 21474.0 | 1.0 | 0.0 | 2014-04-08 | 9.0 | 6.0 | ... | 0.0 | 0.0 | |

5 rows × 30 columns

### Remarks

The feature engg for certain columns is done above and the data is now good for accurate analysis purpose

In [28]:
```
# Creating count plots for certain columns

sns.set(style="whitegrid")

fig, axes = plt.subplots(2, 5, figsize=(26, 10), squeeze=False)
fig.subplots_adjust(top=1.2)

columns_of_interest = ["Education", "Marital_Status", "Kidhome", "Teenhome", "NumDealsPurchases",
                       "NumWebPurchases", "NumCatalogPurchases", "NumStorePurchases", "NumWebVisitsMonth", "Country"]

columns_of_interest = columns_of_interest * (len(axes.flatten()) // len(columns_of_interest) + 1)

# Creating count plots with percentage labels
for i, column in enumerate(columns_of_interest):
    row_index = i // 5
    col_index = i % 5

    if row_index < 2:
        plt_count = sns.countplot(data=df_campaign_copy, x=column, ax=axes[row_index, col_index])
        plt_count.set_title(f"Countplot of {column}")
        plt_count.set_xlabel(column)
        plt_count.set_ylabel("Count")

# Adding percentage labels on top of each bar
        total = len(df_campaign_copy)
        for p in plt_count.patches:
            percentage = '{:.1f}%'.format(100 * p.get_height() / total)
            x = p.get_x() + p.get_width() / 2
            y = p.get_height()
            plt_count.annotate(percentage, (x, y), ha='center', va='bottom')

# Remove any remaining empty subplots
for i in range(len(columns_of_interest), len(axes.flatten())):
    fig.delaxes(axes.flatten()[i])

plt.show()
```

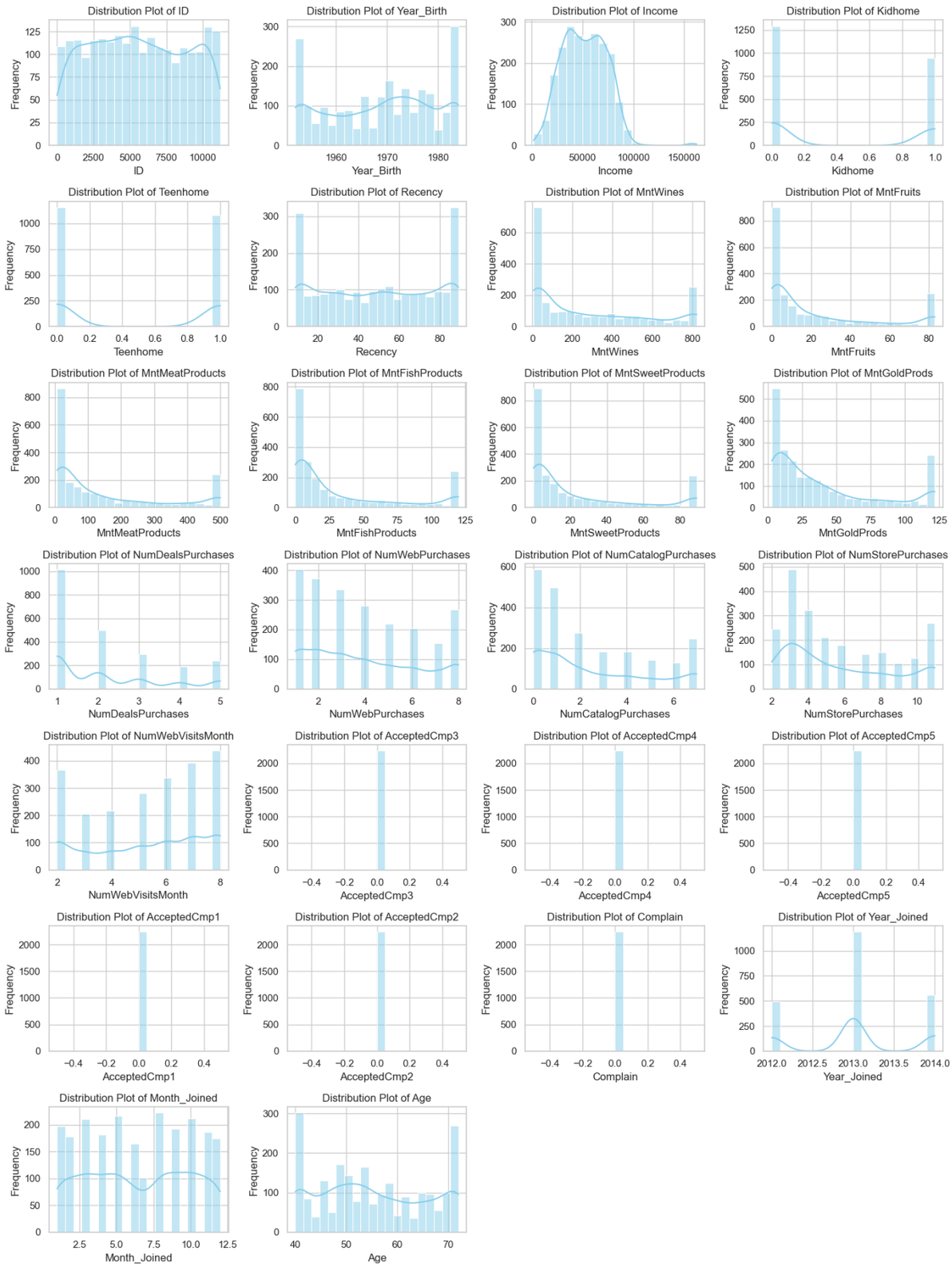



## Remarks

1. The highest count for education belongs to graduation with a % distribution of 50.3% across the entire dataset.
2. The max users are married
3. The max users w.r.t having kids at home is 0 , i.e., most users dont have kid at home
4. The max users w.r.t having teen at home is 0 , i.e., most users dont have teen members at home
5. Max users belong to country code SP
6. In last month max 8 times visit is observed to website by various users.
7. Max of 3 purchases is made directly through stores.

In [29]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Selecting only numeric columns
numeric_columns = df_campaign_copy.select_dtypes(include=['int64', 'float64']).columns

num_rows = len(numeric_columns) // 4 + (len(numeric_columns) % 4 > 0)
num_cols = 4

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 3 * num_rows))
fig.suptitle('Distribution Plots for Numeric Columns', y=1.02)

axes = axes.flatten()

# Creating distribution plots for each numeric column in subplots
for i, column in enumerate(numeric_columns):
    sns.histplot(df_campaign_copy[column], bins=20, kde=True, color='skyblue', ax=axes[i])
    axes[i].set_title(f'Distribution Plot of {column}')
    axes[i].set_xlabel(column)
    axes[i].set_ylabel('Frequency')

# Removing empty subplots
for i in range(len(numeric_columns), len(axes)):
    fig.delaxes(axes[i])

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

Distribution Plots for Numeric Columns




## Remarks

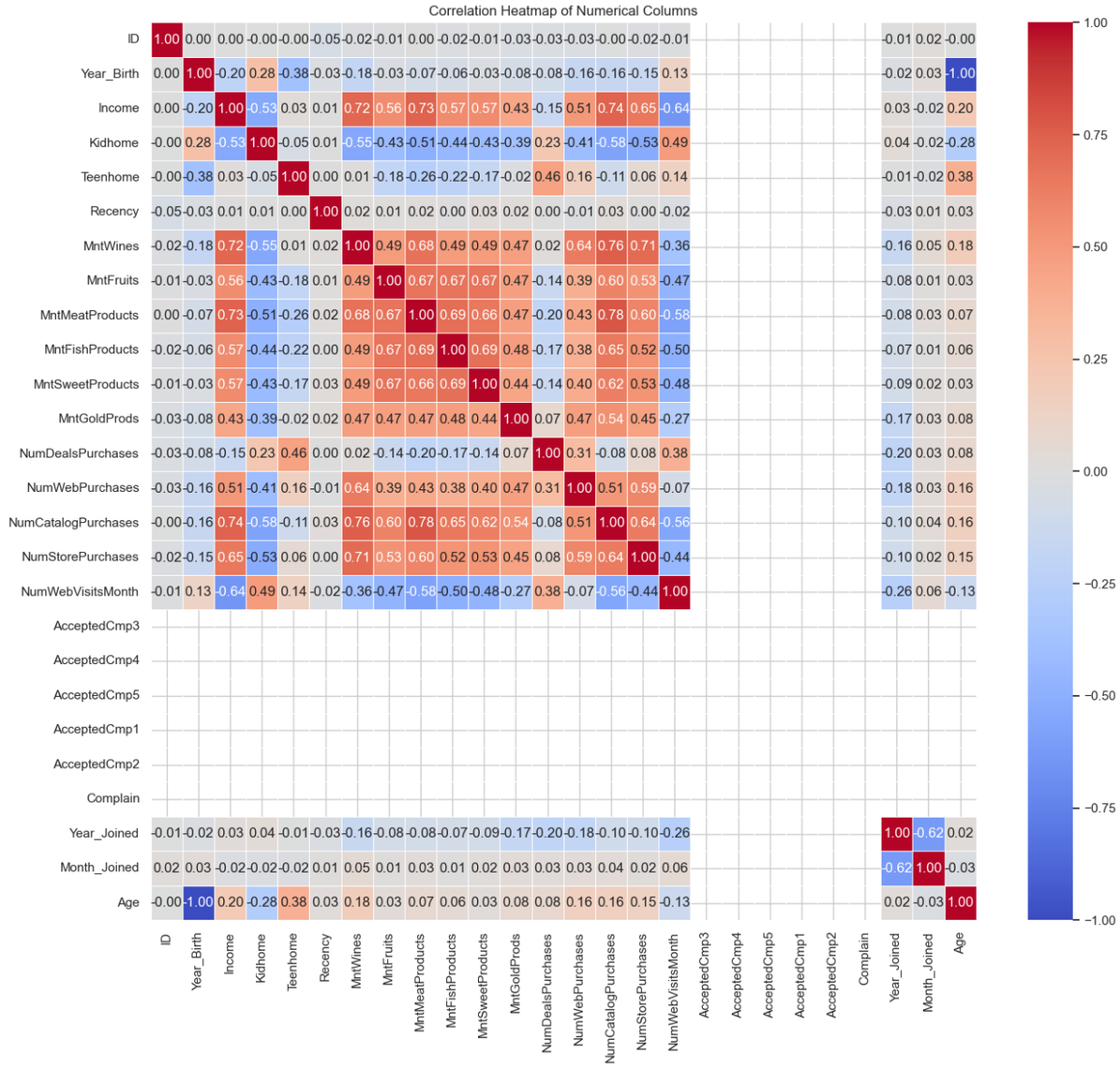
1. The max income of users lies in range of 40-50k
2. The highest recency observed is above 80 for users

## Correlation Analysis | Bivariate | Multivariate Analysis

In [30]:
```
# Selecting only numeric columns
numeric_columns = df_campaign_copy.select_dtypes(include=['int64', 'float64']).columns

# Creating a dataframe with only numerical columns
numeric_df = df_campaign_copy[numeric_columns]

# Calculating the correlation matrix
correlation_matrix = numeric_df.corr()

# Displaying the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)
```

```
AcceptedCmp2              NaN       NaN        NaN              NaN
Complain                  NaN       NaN        NaN              NaN
Year_Joined         -0.027097 -0.155306  -0.077982        -0.084878
Month_Joined         0.014694  0.049845   0.009744         0.026313
Age                  0.025589  0.175725   0.032657         0.071863

                 MntFishProducts  ...  NumWebVisitsMonth  AcceptedCmp3  \
ID                     -0.019228  ...          -0.014318           NaN
Year_Birth             -0.057697  ...           0.128855           NaN
Income                  0.567287  ...          -0.640733           NaN
Kidhome                -0.442251  ...           0.491094           NaN
Teenhome               -0.218483  ...           0.144098           NaN
Recency                 0.001614  ...          -0.022721           NaN
MntWines                0.490590  ...          -0.362638           NaN
MntFruits               0.671525  ...          -0.474640           NaN
MntMeatProducts         0.689721  ...          -0.579784           NaN
MntFishProducts         1.000000  ...          -0.495788           NaN
MntSweetProducts        0.686630  ...          -0.475315           NaN
MntGoldProds            0.484362  ...          -0.267195           NaN
```

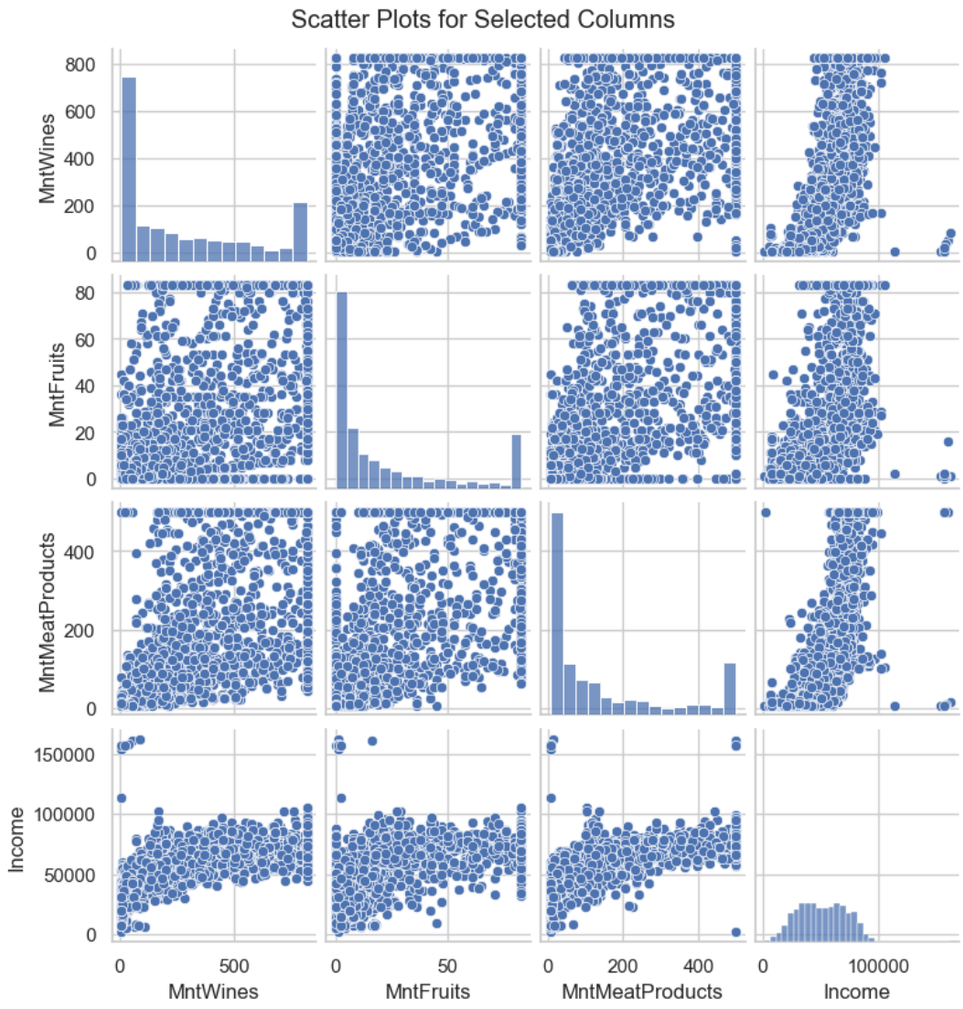
In [31]:
```
plt.figure(figsize=(16, 14))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=.5)
plt.title('Correlation Heatmap of Numerical Columns')
plt.show()
```



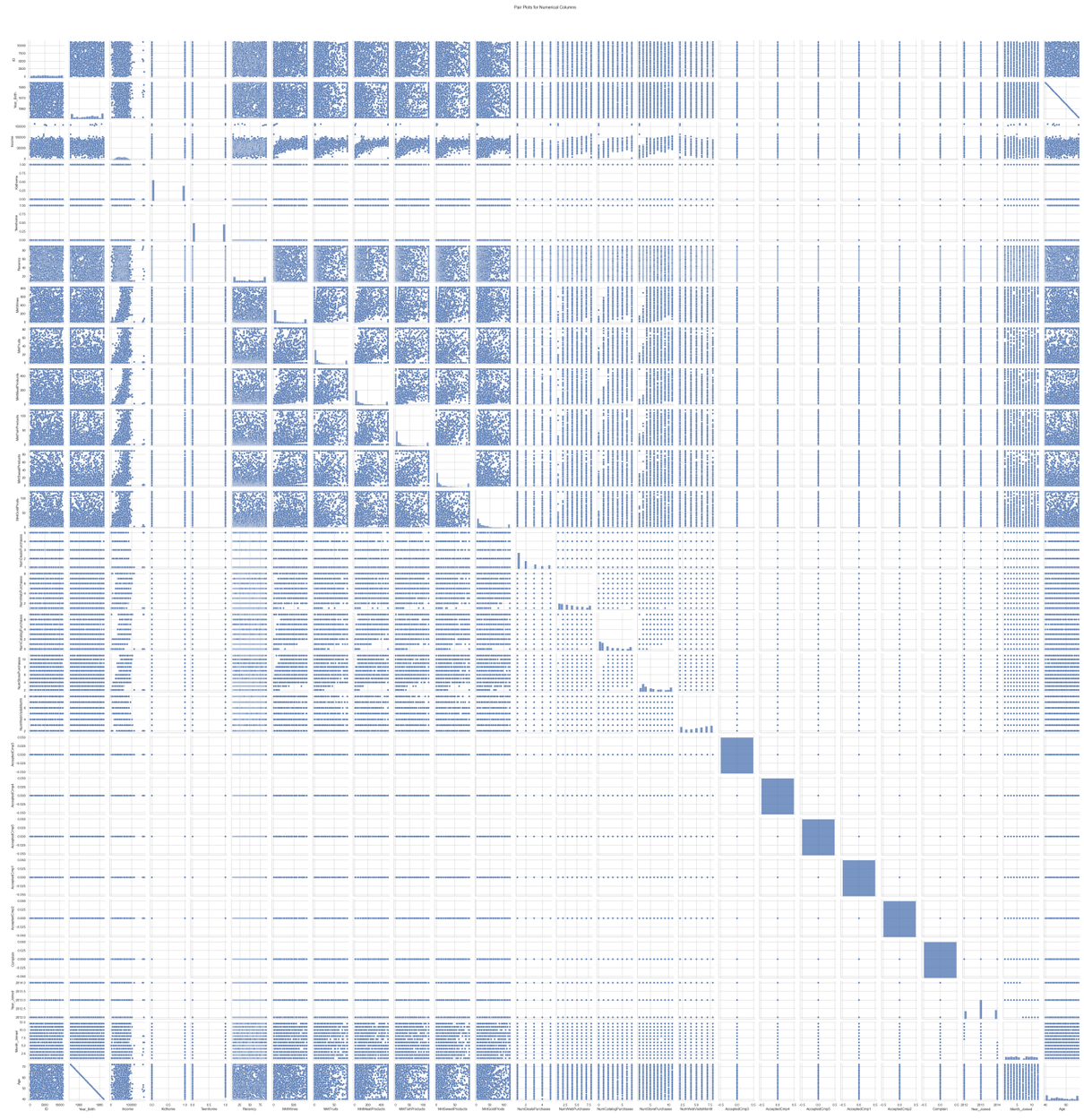

### Remarks

1. NumCatalogPurchases and MntMeatProducts are highly correlated which signifies that most users spent on meat products through catalog
2. NumWebVisitMonths and MntMeatProducts are high negatively correlated amongst all which signifies that users who visited last month didnt prefer meat products

In [32]:
```
# Selecting specific columns for scatter plots
selected_columns = ['MntWines', 'MntFruits', 'MntMeatProducts', 'Income']

# Creating a dataframe with only these selected columns
selected_df = df_campaign_copy[selected_columns]

# Plotting scatter plots
sns.set(style="whitegrid")
sns.pairplot(selected_df, height=2)
plt.suptitle('Scatter Plots for Selected Columns', y=1.02)
plt.show()
```

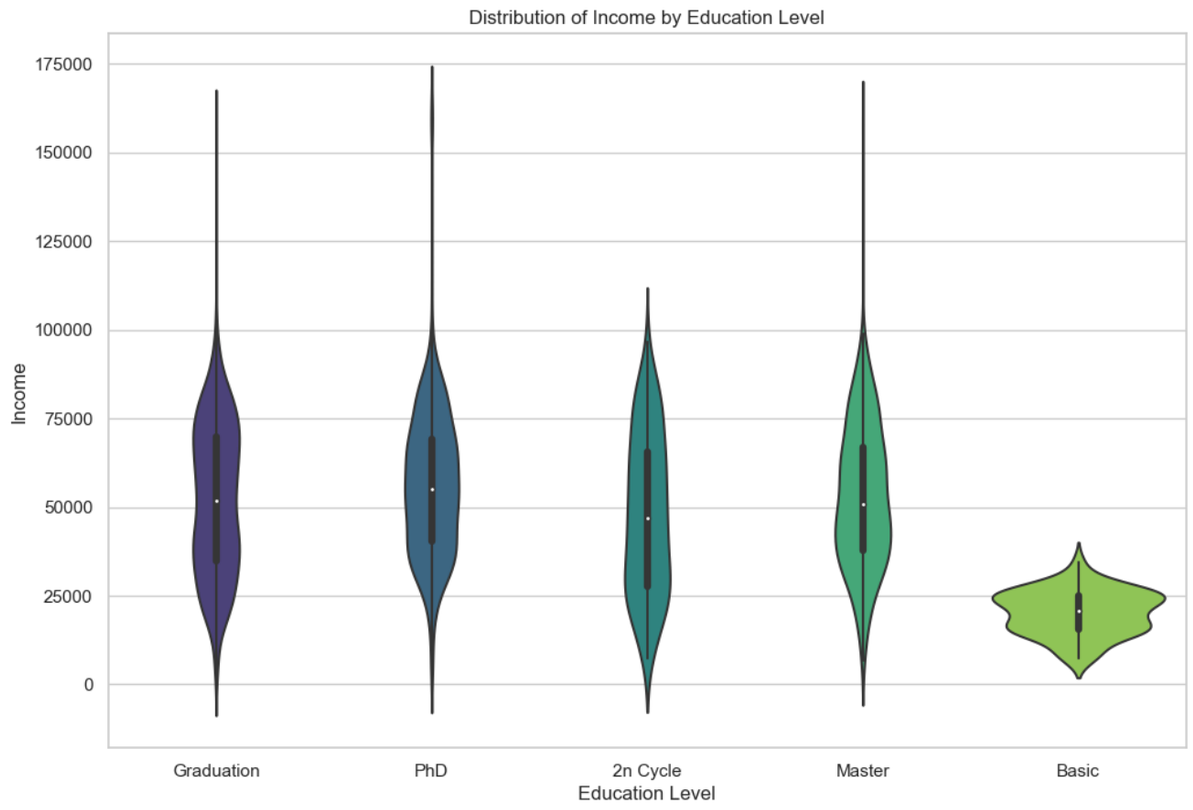


In [33]:
```
# Selecting only numeric columns
numeric_columns = df_campaign_copy.select_dtypes(include=['int64', 'float64']).columns

# Creating a dataframe with only numerical columns
numeric_df = df_campaign_copy[numeric_columns]

# Plotting pair plots for numerical columns
sns.set(style="whitegrid")
sns.pairplot(numeric_df, height=2)
plt.suptitle('Pair Plots for Numerical Columns', y=1.02)
plt.show()
```



## Hypothesis Testing

In [34]:
```
# Ques --> Is income of customers dependent on their education

# Create a contingency table
contingency_table = pd.crosstab(df_campaign_copy['Education'], pd.cut(df_campaign_copy['Income'], bins=[0, 50000, 100000, 1500

plt.figure(figsize=(12, 8))
sns.violinplot(x='Education', y='Income', data=df_campaign_copy, palette='viridis')

plt.title('Distribution of Income by Education Level')
plt.xlabel('Education Level')
plt.ylabel('Income')
plt.show()

# Performing chi-square test
chi2, p, _, _ = chi2_contingency(contingency_table)

# Setting the significance level
alpha = 0.05

print(f"Chi-square statistic: {chi2}")
print(f"P-value: {p}")

# Checking the significance
if p < alpha:
    print("Reject the null hypothesis: There is a significant association between education and income.")
else:
    print("Fail to reject the null hypothesis: There is no significant association between education and income.")
```




```
Chi-square statistic: 83.60510046098752
P-value: 8.433934834142125e-13
Reject the null hypothesis: There is a significant association between education and income.
```

### Remarks

1. The max income source for graduation users (highest amongst all users) have around range of 25k-100k max with median around 50k
2. The users with basic education level have a sparse distribution of income and lowest amongst rest of group
3. The hypothesis testing for above is done and result is printed above where it shows significant relation between both columns

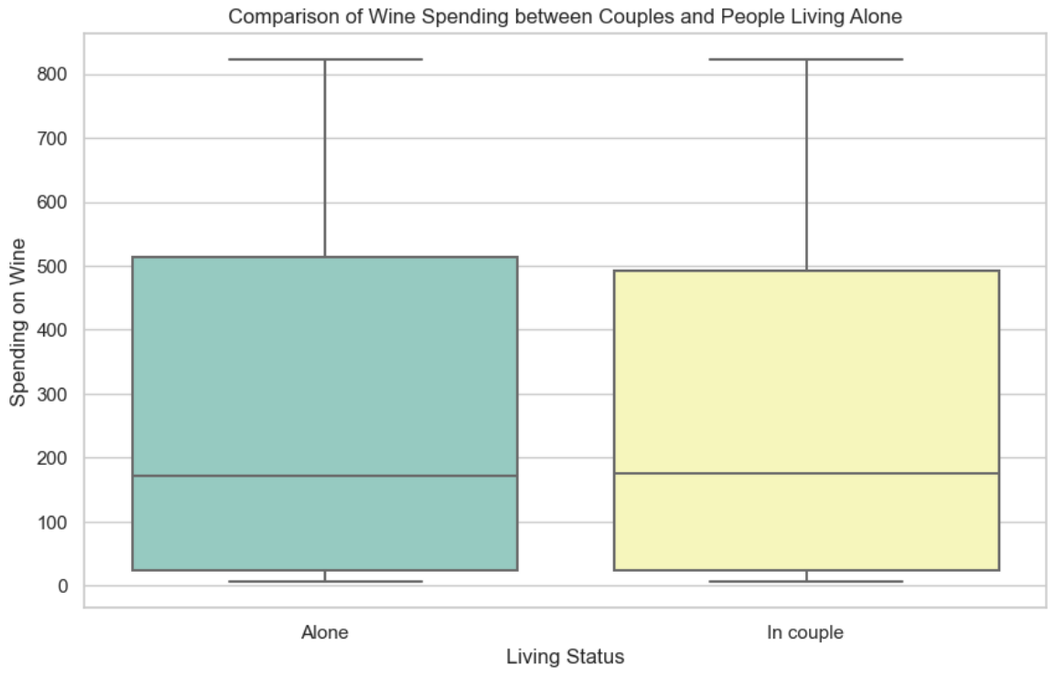
In [38]:
```
# Ques --> Do higher income people spend more (take in account spending in all categories together)

# Calculating the total spending across all categories
df_campaign_copy['Total_Spending'] = df_campaign_copy.loc[:, 'MntWines':'MntGoldProds'].sum(axis=1)

# Creating 1 more copy of the dataset for this testing
df_campaign_copy1=df_campaign_copy.copy()

plt.figure(figsize=(10, 6))
sns.regplot(x='Income', y='Total_Spending', data=df_campaign_copy, scatter_kws={'s': 20}, line_kws={'color': 'red'})
plt.title('Income vs Total Spending with Regression Line')
plt.xlabel('Income')
plt.ylabel('Total Spending')
plt.show()

# Dropping rows with missing or infinite values
df_campaign_copy1 = df_campaign_copy1.replace([np.inf, -np.inf], np.nan).dropna(subset=['Income', 'Total_Spending'])

# Performing Pearson correlation test
corr_coefficient, p_value = pearsonr(df_campaign_copy1['Income'], df_campaign_copy1['Total_Spending'])

# Setting the significance level
alpha = 0.05

print(f"Pearson Correlation Coefficient: {corr_coefficient}")
print(f"P-value: {p_value}")

# Checking the significance
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant correlation between income and total spending.")
else:
    print("Fail to reject the null hypothesis: No significant correlation between income and total spending.")
```


```
Pearson Correlation Coefficient: 0.7969599661090423
P-value: 0.0
Reject the null hypothesis: There is a significant correlation between income and total spending.
```

### Remarks

1. The max datapoints are around 500-1000 for income range between 40k-80k
2. The hypothesis testing for above is done and result is printed above where it shows significant relation between both columns

In [39]:
```
# Ques--> Do couples spend more or less money on wine than people living alone (set 'Married','Together':'
#         In couple' and 'Divorced','Single','Absurd','Widow','YOLO':'Alone')

# Mapping marital status to living status
living_status_mapping = {'Married': 'In couple', 'Together': 'In couple',
                         'Divorced': 'Alone', 'Single': 'Alone',
                         'Absurd': 'Alone', 'Widow': 'Alone', 'YOLO': 'Alone'}

df_campaign_copy['Living_Status'] = df_campaign_copy['Marital_Status'].map(living_status_mapping)

# Plotting boxplot to compare spending on wine
plt.figure(figsize=(10, 6))
sns.boxplot(x='Living_Status', y='MntWines', data=df_campaign_copy, palette='Set3')
plt.title('Comparison of Wine Spending between Couples and People Living Alone')
plt.xlabel('Living Status')
plt.ylabel('Spending on Wine')
plt.show()

# Extracting data for couples and people living alone
wine_spending_couples = df_campaign_copy[df_campaign_copy['Living_Status'] == 'In couple']['MntWines']
wine_spending_alone = df_campaign_copy[df_campaign_copy['Living_Status'] == 'Alone']['MntWines']

# Performing t-test
t_statistic, p_value = ttest_ind(wine_spending_couples, wine_spending_alone, equal_var=False)

# Setting the significance level
alpha = 0.05

print(f"T-statistic: {t_statistic}")
print(f"P-value: {p_value}")

# Checking the significance
if p_value < alpha:
    print("Reject the null hypothesis: Couples spend a different amount on wine than people living alone.")
else:
    print("Fail to reject the null hypothesis: No significant difference in wine spending between couples and people living al
```




```
T-statistic: -0.4003838083060327
P-value: 0.688926951725054
Fail to reject the null hypothesis: No significant difference in wine spending between couples and people living alone.
```

### Remarks

1. Single have spent more on wine as compared to married and the median is almost same for both around 160.
2. The hypothesis testing for above is done and result is printed above where it shows no significant relation between both columns

In [41]:
```
# Ques--> Are people with lower income are more attracted towards campaign or simply put accept more campaigns.
#         ( create two income brackets one below median , other above median income and create a column which tells
#         if they have ever accepted any campaign)

# Creating two income brackets: below median and above median
median_income = df_campaign_copy['Income'].median()
df_campaign_copy['Income_Bracket'] = pd.cut(df_campaign_copy['Income'], bins=[float('-inf'), median_income, float('inf')],
                                            labels=['Below Median', 'Above Median'])

# Creating a new column indicating whether a person has ever accepted any campaign
df_campaign_copy['Accepted_Any_Campaign'] = (df_campaign_copy[['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3',
                                                               'AcceptedCmp4', 'AcceptedCmp5']].sum(axis=1) > 0).astype(int)

# Performing chi-square test for independence
contingency_table = pd.crosstab(df_campaign_copy['Income_Bracket'], df_campaign_copy['Accepted_Any_Campaign'])
chi2, p, _, _ = chi2_contingency(contingency_table)

# Setting the significance level
alpha = 0.05

print(f"Chi-square statistic: {chi2}")
print(f"P-value: {p}")

# Checking the significance
if p < alpha:
    print("Reject the null hypothesis: There is a significant difference in campaign acceptance between income brackets.")
else:
    print("Fail to reject the null hypothesis: No significant difference in campaign acceptance between income brackets.")
```

```
Chi-square statistic: 0.0
P-value: 1.0
Fail to reject the null hypothesis: No significant difference in campaign acceptance between income brackets.
```

In [42]:
```
# Ques--> Is there a significant difference in spending on wines between customers from different countries?

# Performing one-way ANOVA test
result = f_oneway(df_campaign_copy[df_campaign_copy['Country'] == 'SP']['MntWines'],
                  df_campaign_copy[df_campaign_copy['Country'] == 'CA']['MntWines'],
                  df_campaign_copy[df_campaign_copy['Country'] == 'US']['MntWines'],
                  df_campaign_copy[df_campaign_copy['Country'] == 'AUS']['MntWines'],
                  df_campaign_copy[df_campaign_copy['Country'] == 'GER']['MntWines'])

# Setting the significance level
alpha = 0.05

print(f"One-way ANOVA F-statistic: {result.statistic}")
print(f"P-value: {result.pvalue}")

# Checking the significance
if result.pvalue < alpha:
    print("Reject the null hypothesis: There is a significant difference in wine spending between countries.")
else:
    print("Fail to reject the null hypothesis: No significant difference in wine spending between countries.")
```

```
One-way ANOVA F-statistic: 0.35228860349303065
P-value: 0.8425609397838358
Fail to reject the null hypothesis: No significant difference in wine spending between countries.
```

In [43]:
```
# Ques--> Is there a significant correlation between the number of web visits per month and the acceptance of Campaign 1?

# Performing Pearson correlation test
corr_coefficient, p_value = pearsonr(df_campaign_copy['NumWebVisitsMonth'], df_campaign_copy['AcceptedCmp1'])

# Setting the significance level
alpha = 0.05

print(f"Pearson Correlation Coefficient: {corr_coefficient}")
print(f"P-value: {p_value}")

# Checking the significance
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant correlation between web visits and acceptance of Campaign 1.")
else:
    print("Fail to reject the null hypothesis: No significant correlation between web visits and acceptance of Campaign 1.")
```

```
Pearson Correlation Coefficient: nan
P-value: nan
Fail to reject the null hypothesis: No significant correlation between web visits and acceptance of Campaign 1.

C:\Users\Dell\anaconda3\lib\site-packages\scipy\stats\_stats_py.py:4424: ConstantInputWarning: An input array is constant; th
e correlation coefficient is not defined.
  warnings.warn(stats.ConstantInputWarning(msg))
```

In [44]:
```
# Ques--> Is there a significant difference in the mean income between married and single customers?

# Extracting income data for married and single customers separately
income_married = df_campaign_copy[df_campaign_copy['Marital_Status'].isin(['Married', 'Together'])]['Income']
income_single = df_campaign_copy[df_campaign_copy['Marital_Status'].isin(['Single', 'Absurd', 'Widow', 'YOLO'])]['Income']

# Performing t-test
t_statistic, p_value = ttest_ind(income_married, income_single, equal_var=False)

# Setting the significance level
alpha = 0.05

print(f"T-statistic: {t_statistic}")
print(f"P-value: {p_value}")

# Checking the significance
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference in mean income between married and single customers."
else:
    print("Fail to reject the null hypothesis: No significant difference in mean income between married and single customers."
```

```
T-statistic: nan
P-value: nan
Fail to reject the null hypothesis: No significant difference in mean income between married and single customers.
```

In [45]:
```
# Ques--> Is there a significant association between education level and the acceptance of any campaign?

# Creating a contingency table
contingency_table = pd.crosstab(df_campaign_copy['Education'], df_campaign_copy['Accepted_Any_Campaign'])

# Performing chi-square test for independence
chi2, p, _, _ = chi2_contingency(contingency_table)

# Setting the significance level
alpha = 0.05

print(f"Chi-square statistic: {chi2}")
print(f"P-value: {p}")

# Checking the significance
if p < alpha:
    print("Reject the null hypothesis: There is a significant association between education level and the acceptance of any ca
else:
    print("Fail to reject the null hypothesis: No significant association between education level and the acceptance of any ca
```

```
Chi-square statistic: 0.0
P-value: 1.0
Fail to reject the null hypothesis: No significant association between education level and the acceptance of any campaign.
```

In [46]:
```
# Ques--> Is there a significant difference in the mean number of web visits per month between customers from different countr

# Performing one-way ANOVA test
result = f_oneway(df_campaign_copy[df_campaign_copy['Country'] == 'SP']['NumWebVisitsMonth'],
                  df_campaign_copy[df_campaign_copy['Country'] == 'CA']['NumWebVisitsMonth'],
                  df_campaign_copy[df_campaign_copy['Country'] == 'US']['NumWebVisitsMonth'],
                  df_campaign_copy[df_campaign_copy['Country'] == 'AUS']['NumWebVisitsMonth'],
                  df_campaign_copy[df_campaign_copy['Country'] == 'GER']['NumWebVisitsMonth'])

# Setting the significance level
alpha = 0.05

print(f"One-way ANOVA F-statistic: {result.statistic}")
print(f"P-value: {result.pvalue}")

# Checking the significance
if result.pvalue < alpha:
    print("Reject the null hypothesis: There is a significant difference in the mean number of web visits per month between cu
else:
    print("Fail to reject the null hypothesis: No significant difference in the mean number of web visits per month between cu
```

```
One-way ANOVA F-statistic: 0.5416708996002387
P-value: 0.7051521741364577
Fail to reject the null hypothesis: No significant difference in the mean number of web visits per month between customers fr
om different countries.
```

## Recommendations

1. Segmentation Strategy: Utilizing customer segmentation based on demographics, spending behavior, and other relevant factors to tailor marketing strategies for different customer groups.
2. Personalized Campaigns: Leveraging the insights from campaign acceptance analysis to design more personalized and targeted marketing campaigns. Tailoring offers to specific customer preferences can enhance campaign effectiveness.
3. Customer Retention: Analyzing the factors associated with customer complaints and take proactive measures to address customer concerns. Improving customer satisfaction can contribute to higher retention rates.
4. International Expansion: Exploring opportunities for international expansion by understanding the distribution of customers across different countries. Considering adapting marketing strategies to cater to the preferences of customers in specific regions.
5. Optimized Web Presence: Leveraging insights from web visits data to optimize the online presence. Enhancing the user experience on the website and strategically placing campaign information to attract and engage customers.
6. Campaign Timing: Considering the timing of marketing campaigns based on recency data. Analyzing when customers are most receptive to campaigns and optimizing the timing of future marketing initiatives.
7. Product Preferences: Identifying top-spending categories to align marketing efforts with customer preferences. Highlighting some promotions and offerings related to popular product categories to maximize customer engagement.
8. Education-Targeted Campaigns: Tailoring marketing messages and campaigns based on the education level of customers. For instance, designing campaigns that resonate with the preferences and aspirations of customers with different education backgrounds.
9. Cross-Sell and Up-Sell Opportunities: Identifying cross-selling and up-selling opportunities by analyzing spending patterns across different product categories. Creation of bundled offers or promotions to encourage customers to explore additional products.
10. Customer Loyalty Programs: Implementing customer loyalty programs to reward and retain high-spending customers. Design of loyalty initiatives that align with the preferences of the target customer segments.
11. Data Security and Privacy: Ensuring compliance with data security and privacy regulations. Maintaining the customer trust by implementing robust data protection measures and transparent communication regarding data usage.

12. Social Media Engagement: Leveraging insights from customer demographics to refine social media strategies. Tailoring content and engagement strategies to resonate with the age groups, education levels, and other demographic characteristics of the target audience.
13. Continuous Feedback Mechanism: Establishing a continuous feedback mechanism to gather insights from customers. Use of surveys, feedback forms, or customer service interactions to understand evolving customer preferences and expectations.
14. Competitor Benchmarking: Regularly benchmarking of marketing strategies against competitors. Staying informed about industry trends and competitive offerings to ensure the business remains competitive in the market.

EOF