# Marketing Campaigns

Author: Razan

> **Problem Scenario**: *'Marketing mix'* is a popular concept used in implementing marketing strategies. A marketing mix includes multiple areas of focus as part of a comprehensive marketing plan. This all revolves around the four Ps of marketing - product, price, place, and promotion.

> **Problem Objective**: As a data scientist, you should perform exploratory data analysis and hypothesis testing. The goal is to gain a better understanding of the various factors that contribute to customer acquisition.

## 1. Data Wrangling and preprocessing

### 1.1 Importing Required Libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from scipy import stats
from scipy.stats import t
import statsmodels.api as sm
from statsmodels.formula.api import ols
import plotly.express as px
import plotly.graph_objects as go
# sns.set_style('dark')
sns.set_palette("rainbow_r")
sns.set_style("darkgrid", {"grid.color": ".6", "grid.linestyle": ":"})
```

### 1.2 Loading the datasets

In [2]:
```
df = pd.read_csv("marketing_data.csv")
```

### 1.3 Data Exploration and Statistical Summary

In [3]:
```
df
```

Out[3]:

| | ID | Year_Birth | Education | Marital_Status | Income | Kidhome | Teenhome | Dt_Customer | Recency | MntWines | ... | NumStorePur |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1826 | 1970 | Graduation | Divorced | $84,835.00 | 0 | 0 | 6/16/14 | 0 | 189 | ... | |
| **1** | 1 | 1961 | Graduation | Single | $57,091.00 | 0 | 0 | 6/15/14 | 0 | 464 | ... | |
| **2** | 10476 | 1958 | Graduation | Married | $67,267.00 | 0 | 1 | 5/13/14 | 0 | 134 | ... | |
| **3** | 1386 | 1967 | Graduation | Together | $32,474.00 | 1 | 1 | 5/11/14 | 0 | 10 | ... | |
| **4** | 5371 | 1989 | Graduation | Single | $21,474.00 | 1 | 0 | 4/8/14 | 0 | 6 | ... | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2235** | 10142 | 1976 | PhD | Divorced | $66,476.00 | 0 | 1 | 3/7/13 | 99 | 372 | ... | |
| **2236** | 5263 | 1977 | 2n Cycle | Married | $31,056.00 | 1 | 0 | 1/22/13 | 99 | 5 | ... | |
| **2237** | 22 | 1976 | Graduation | Divorced | $46,310.00 | 1 | 0 | 12/3/12 | 99 | 185 | ... | |
| **2238** | 528 | 1978 | Graduation | Married | $65,819.00 | 0 | 0 | 11/29/12 | 99 | 267 | ... | |
| **2239** | 4070 | 1969 | PhD | Married | $94,871.00 | 0 | 2 | 9/1/12 | 99 | 169 | ... | |

2240 rows × 28 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 28 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   ID                   2240 non-null   int64 
 1   Year_Birth           2240 non-null   int64 
 2   Education            2240 non-null   object
 3   Marital_Status       2240 non-null   object
 4    Income              2216 non-null   object
 5   Kidhome              2240 non-null   int64 
 6   Teenhome             2240 non-null   int64 
 7   Dt_Customer          2240 non-null   object
 8   Recency              2240 non-null   int64 
 9   MntWines             2240 non-null   int64 
 10  MntFruits            2240 non-null   int64 
 11  MntMeatProducts      2240 non-null   int64 
 12  MntFishProducts      2240 non-null   int64 
 13  MntSweetProducts     2240 non-null   int64 
 14  MntGoldProds         2240 non-null   int64 
 15  NumDealsPurchases    2240 non-null   int64 
 16  NumWebPurchases      2240 non-null   int64 
 17  NumCatalogPurchases  2240 non-null   int64 
 18  NumStorePurchases    2240 non-null   int64 
 19  NumWebVisitsMonth    2240 non-null   int64 
 20  AcceptedCmp3         2240 non-null   int64 
 21  AcceptedCmp4         2240 non-null   int64 
 22  AcceptedCmp5         2240 non-null   int64 
 23  AcceptedCmp1         2240 non-null   int64 
 24  AcceptedCmp2         2240 non-null   int64 
 25  Response             2240 non-null   int64 
 26  Complain             2240 non-null   int64 
 27  Country              2240 non-null   object
dtypes: int64(23), object(5)
memory usage: 490.1+ KB
```

In [5]:
```
def data_info(df):
    info = pd.DataFrame(index=df.columns)
    info['DataType'] = df.dtypes
    info['Unique Value Count'] = df.nunique()
    info['Unique Values'] = [df[col].unique() for col in df.columns]
    info['Missing Values'] = df.isnull().sum()
    info['Missing Values Percentage (%)'] = (info['Missing Values'] / df.shape[0]) * 100
    return info
data_info(df)
```

Out[5]:

| | DataType | Unique Value Count | Unique Values | Missing Values | Missing Values Percentage (%) |
|---|---|---|---|---|---|
| **ID** | int64 | 2240 | [1826, 1, 10476, 1386, 5371, 7348, 4073, 1991,... | 0 | 0.000000 |
| **Year_Birth** | int64 | 59 | [1970, 1961, 1958, 1967, 1989, 1954, 1947, 197... | 0 | 0.000000 |
| **Education** | object | 5 | [Graduation, PhD, 2n Cycle, Master, Basic] | 0 | 0.000000 |
| **Marital_Status** | object | 8 | [Divorced, Single, Married, Together, Widow, Y... | 0 | 0.000000 |
| **Income** | object | 1974 | [84, 835.00,57,091.00 , 67, 267.00,32,47... | 24 | 1.071429 |
| **Kidhome** | int64 | 3 | [0, 1, 2] | 0 | 0.000000 |
| **Teenhome** | int64 | 3 | [0, 1, 2] | 0 | 0.000000 |
| **Dt_Customer** | object | 663 | [6/16/14, 6/15/14, 5/13/14, 5/11/14, 4/8/14, 3... | 0 | 0.000000 |
| **Recency** | int64 | 100 | [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,... | 0 | 0.000000 |
| **MntWines** | int64 | 776 | [189, 464, 134, 10, 6, 336, 769, 78, 384, 450,... | 0 | 0.000000 |
| **MntFruits** | int64 | 158 | [104, 5, 11, 0, 16, 130, 80, 26, 4, 82, 10, 6,... | 0 | 0.000000 |
| **MntMeatProducts** | int64 | 558 | [379, 64, 59, 1, 24, 411, 252, 11, 102, 535, 6... | 0 | 0.000000 |
| **MntFishProducts** | int64 | 182 | [111, 7, 15, 0, 11, 240, 21, 73, 80, 3, 2, 13,... | 0 | 0.000000 |
| **MntSweetProducts** | int64 | 177 | [189, 0, 2, 32, 34, 98, 13, 20, 16, 4, 1, 3, 7... | 0 | 0.000000 |
| **MntGoldProds** | int64 | 213 | [218, 37, 30, 0, 34, 43, 65, 7, 5, 26, 4, 102,... | 0 | 0.000000 |
| **NumDealsPurchases** | int64 | 15 | [1, 2, 3, 0, 4, 12, 7, 5, 6, 11, 9, 8, 10, 15,... | 0 | 0.000000 |
| **NumWebPurchases** | int64 | 15 | [4, 7, 3, 1, 10, 2, 6, 5, 25, 8, 9, 0, 11, 27,... | 0 | 0.000000 |
| **NumCatalogPurchases** | int64 | 14 | [4, 3, 2, 0, 1, 7, 10, 6, 8, 5, 9, 11, 28, 22] | 0 | 0.000000 |
| **NumStorePurchases** | int64 | 14 | [6, 7, 5, 2, 3, 9, 10, 0, 8, 4, 13, 12, 1, 11] | 0 | 0.000000 |
| **NumWebVisitsMonth** | int64 | 16 | [1, 5, 2, 7, 6, 4, 8, 3, 9, 0, 17, 13, 10, 14,... | 0 | 0.000000 |
| **AcceptedCmp3** | int64 | 2 | [0, 1] | 0 | 0.000000 |
| **AcceptedCmp4** | int64 | 2 | [0, 1] | 0 | 0.000000 |
| **AcceptedCmp5** | int64 | 2 | [0, 1] | 0 | 0.000000 |
| **AcceptedCmp1** | int64 | 2 | [0, 1] | 0 | 0.000000 |
| **AcceptedCmp2** | int64 | 2 | [0, 1] | 0 | 0.000000 |
| **Response** | int64 | 2 | [1, 0] | 0 | 0.000000 |
| **Complain** | int64 | 2 | [0, 1] | 0 | 0.000000 |
| **Country** | object | 8 | [SP, CA, US, AUS, GER, IND, SA, ME] | 0 | 0.000000 |

In [6]:
```
# check Duplicates
df[df.duplicated(keep=False)]
```

Out[6]:

| ID | Year_Birth | Education | Marital_Status | Income | Kidhome | Teenhome | Dt_Customer | Recency | MntWines | ... | NumStorePurchases | Nu |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

0 rows × 28 columns

this indicates that there are no duplicates in our data which is brilliant

In [7]:
```
df.columns
```

Out[7]:
```
Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', ' Income ',
       'Kidhome', 'Teenhome', 'Dt_Customer', 'Recency', 'MntWines',
       'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
       'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
       'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth',
       'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1',
       'AcceptedCmp2', 'Response', 'Complain', 'Country'],
      dtype='object')
```

## 1.4 Dealing with missing values

#### 1.4.0 Dealing with the 'Income' feature based on the 'Education' and 'Marital_Status' features

In [8]:
```
df.rename({' Income ':'Income'},axis=1,inplace=True)
```

In [9]:
```
df.columns
```

Out[9]:
```
Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome',
       'Teenhome', 'Dt_Customer', 'Recency', 'MntWines', 'MntFruits',
       'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
       'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
       'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth',
       'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1',
       'AcceptedCmp2', 'Response', 'Complain', 'Country'],
      dtype='object')
```

In [10]:
```
df['Income'].unique()
```

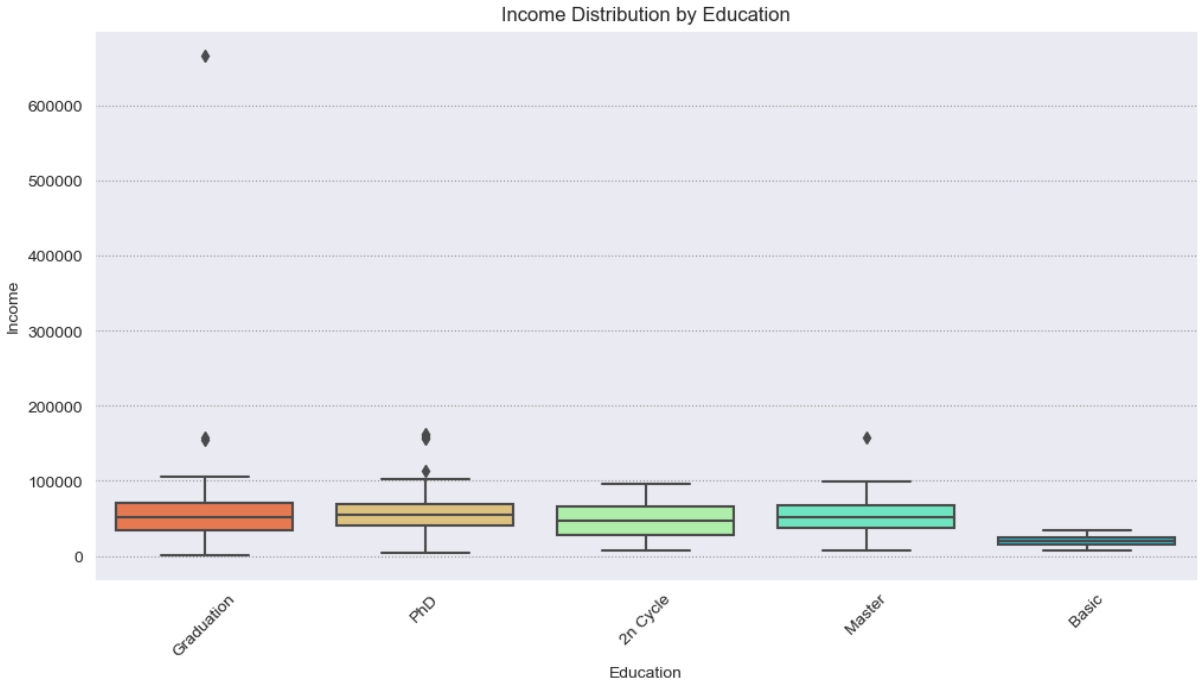
Out[10]:
```
array(['$84,835.00 ', '$57,091.00 ', '$67,267.00 ', ..., '$46,310.00 ',
       '$65,819.00 ', '$94,871.00 '], dtype=object)
```

In [11]:
```
# Remove dollar signs and commas, then convert to numeric
df['Income'] = df['Income'].str.replace('[\$,]', '', regex=True).astype(float)
df['Income'].unique()
```

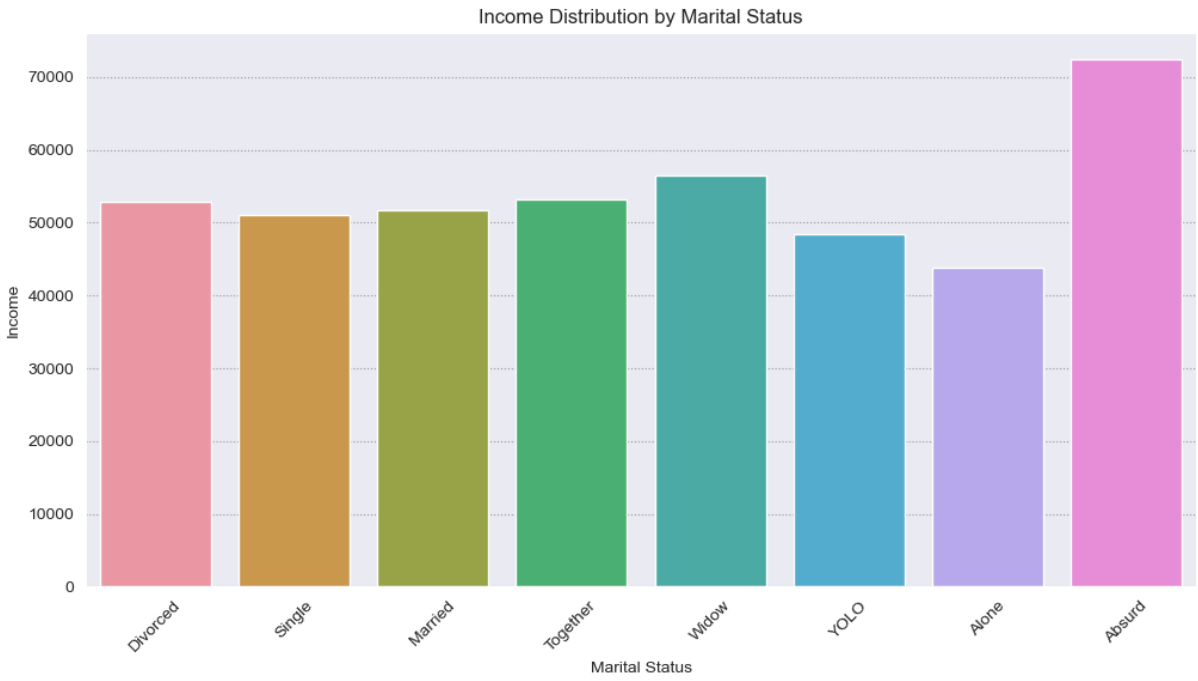
Out[11]:
```
array([84835., 57091., 67267., ..., 46310., 65819., 94871.])
```

In [12]:

```
# the distribution of income for each education level
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='Education', y='Income')
plt.xlabel('Education')
plt.ylabel('Income')
plt.title('Income Distribution by Education')
plt.xticks(rotation=45)
plt.show()

# the distribution of income for each marital status
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x='Marital_Status', y='Income', ci=None)
plt.xlabel('Marital Status')
plt.ylabel('Income')
plt.title('Income Distribution by Marital Status')
plt.xticks(rotation=45)
plt.show()
```



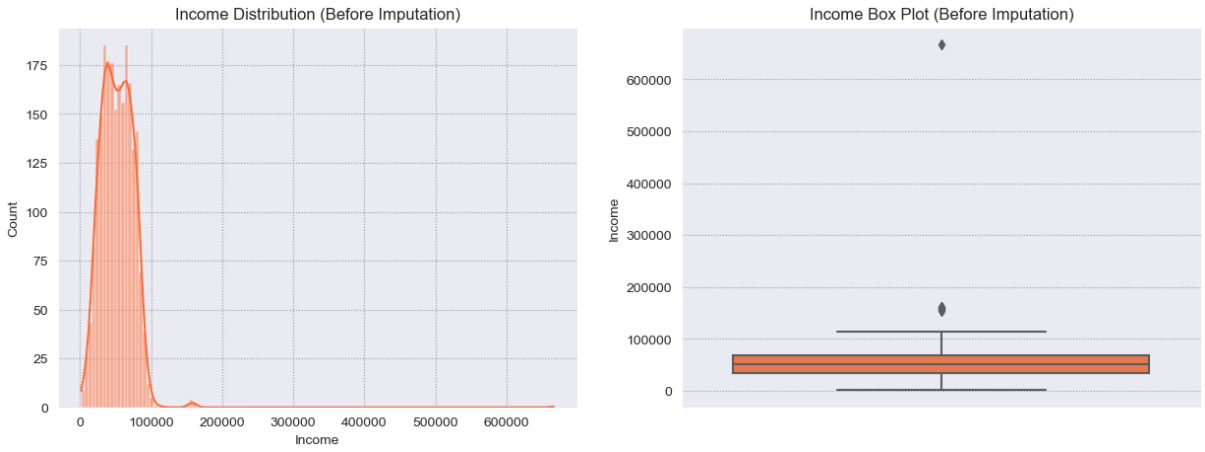

```
C:\Users\r-a-a\anaconda3\envs\cw1\lib\site-packages\ipykernel_launcher.py:12: FutureWarning:

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

  if sys.path[0] == "":
```



In [13]:

```
# Before Imputation
# summary statistics
mean_before = df['Income'].mean()
median_before = df['Income'].median()
std_deviation_before = df['Income'].std()

# visualisation - hist
plt.figure(figsize=(15, 5))
plt.subplot(121)
sns.histplot(df['Income'], kde=True)
plt.xlabel('Income')
plt.title('Income Distribution (Before Imputation)')

# visualisation - Box plot
plt.subplot(122)
sns.boxplot(y=df['Income'])
plt.title('Income Box Plot (Before Imputation)')
```

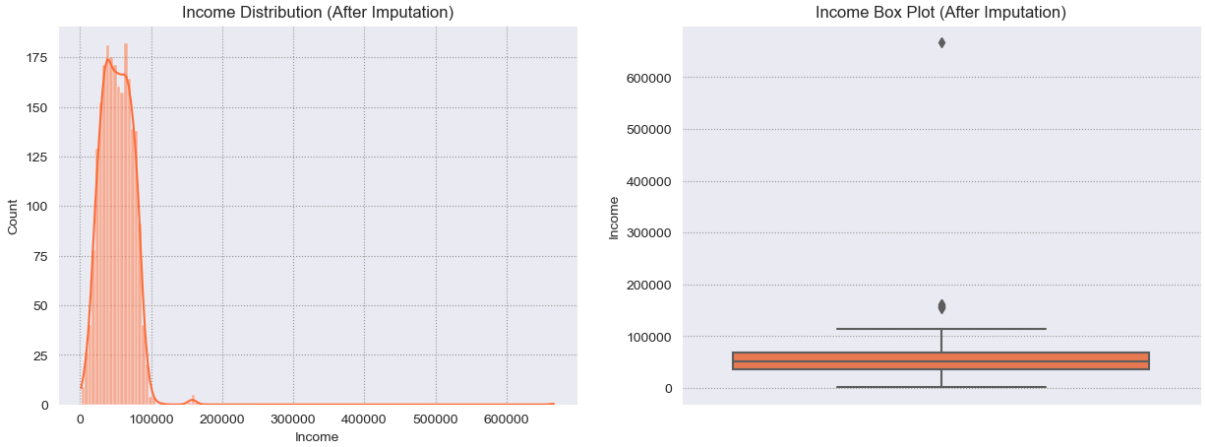
Out[13]: Text(0.5, 1.0, 'Income Box Plot (Before Imputation)')



In [14]:

```
# Imputation
grouped = df.groupby(['Education', 'Marital_Status'])['Income'].transform('mean')
df['Income'].fillna(grouped, inplace=True)

# summary statistics
mean_after = df['Income'].mean()
median_after = df['Income'].median()
std_deviation_after = df['Income'].std()

# visualisation - hist
plt.figure(figsize=(15, 5))
plt.subplot(121)
sns.histplot(df['Income'], kde=True)
plt.xlabel('Income')
plt.title('Income Distribution (After Imputation)')

# visualisation - Box plot
plt.subplot(122)
sns.boxplot(y=df['Income'])
plt.title('Income Box Plot (After Imputation)')

# Compare summary statistics
print(f"Before Imputation - Mean: {mean_before}, Median: {median_before}, Std Dev: {std_deviation_before}")
print(f"After Imputation - Mean: {mean_after}, Median: {median_after}, Std Dev: {std_deviation_after}")
```

```
Before Imputation - Mean: 52247.25135379061, Median: 51381.5, Std Dev: 25173.076660901403
After Imputation - Mean: 52248.748824908515, Median: 51381.5, Std Dev: 25039.981052325227
```




The summary statistics for the "Income" variable before and after imputation suggest that the imputation process effectively maintained the central tendency (mean and median) of the income values and preserved the overall income distribution.

In [15]:
```
def data_info(df):
    info = pd.DataFrame(index=df.columns)
    info['Missing Values'] = df.isnull().sum()
    info['Missing Values Percentage (%)'] = (info['Missing Values'] / df.shape[0]) * 100
    return info
data_info(df)
```

Out[15]:

| | Missing Values | Missing Values Percentage (%) |
|---|---|---|
| **ID** | 0 | 0.0 |
| **Year_Birth** | 0 | 0.0 |
| **Education** | 0 | 0.0 |
| **Marital_Status** | 0 | 0.0 |
| **Income** | 0 | 0.0 |
| **Kidhome** | 0 | 0.0 |
| **Teenhome** | 0 | 0.0 |
| **Dt_Customer** | 0 | 0.0 |
| **Recency** | 0 | 0.0 |
| **MntWines** | 0 | 0.0 |
| **MntFruits** | 0 | 0.0 |
| **MntMeatProducts** | 0 | 0.0 |
| **MntFishProducts** | 0 | 0.0 |
| **MntSweetProducts** | 0 | 0.0 |
| **MntGoldProds** | 0 | 0.0 |
| **NumDealsPurchases** | 0 | 0.0 |
| **NumWebPurchases** | 0 | 0.0 |
| **NumCatalogPurchases** | 0 | 0.0 |
| **NumStorePurchases** | 0 | 0.0 |
| **NumWebVisitsMonth** | 0 | 0.0 |
| **AcceptedCmp3** | 0 | 0.0 |
| **AcceptedCmp4** | 0 | 0.0 |
| **AcceptedCmp5** | 0 | 0.0 |
| **AcceptedCmp1** | 0 | 0.0 |
| **AcceptedCmp2** | 0 | 0.0 |
| **Response** | 0 | 0.0 |
| **Complain** | 0 | 0.0 |
| **Country** | 0 | 0.0 |

### 1.5 Create Additional Variables

> Create variables to populate the total number of children, age, and total spending.

- we can use the Year_Birth to get the age
- we can use Kidhome and Teenhome to get the total number of children
- we can use MntWines, MntFruits, MntMeatProducts, MntFishProducts, MntSweetProducts and MntGoldProds to get the total spending in the last 2 years

In [16]:

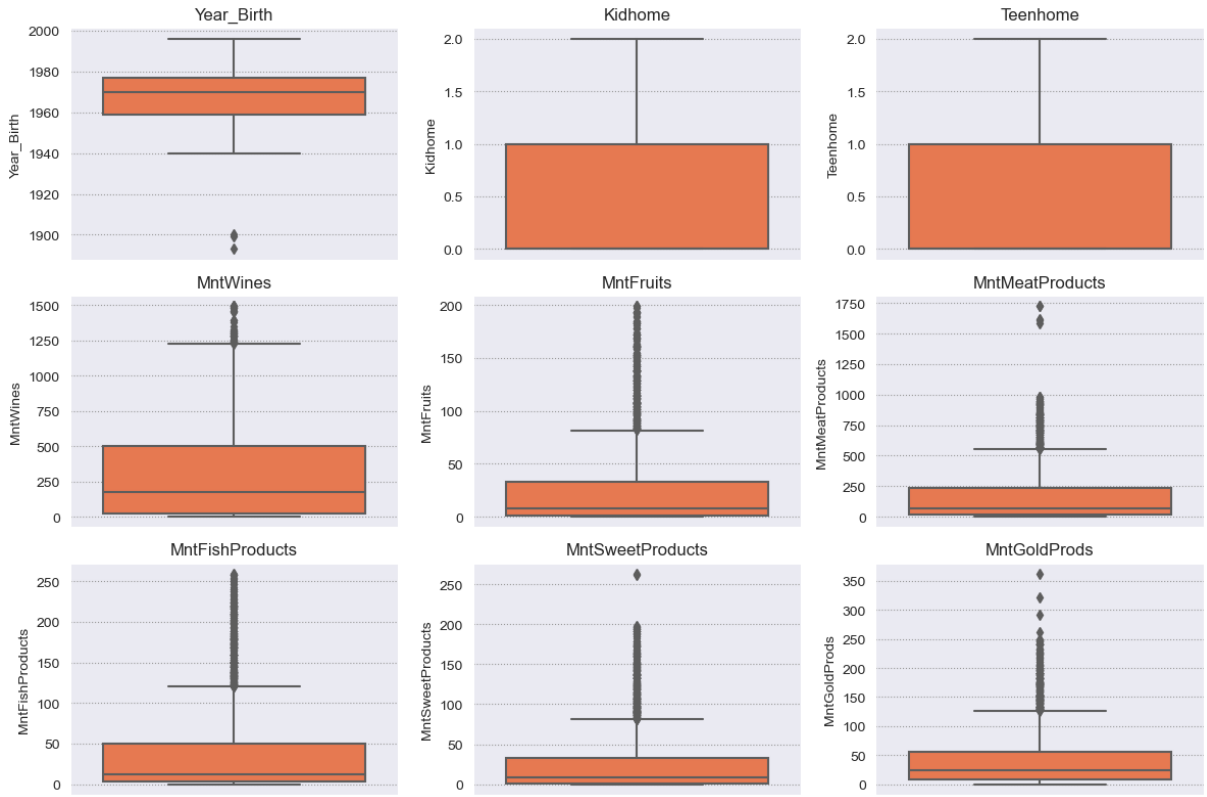
```
columns_to_view = ['Year_Birth', 'Kidhome', 'Teenhome', 'MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishPr

for column in columns_to_view:
    unique_values = df[column].unique()
    data_type = df[column].dtype
    print(f"Unique values in '{column}':")
    print(unique_values)
    print(f"Data type of '{column}': {data_type}\n")
```

```
Unique values in 'Year_Birth':
[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]
Data type of 'Year_Birth': int64

Unique values in 'Kidhome':
[0 1 2]
Data type of 'Kidhome': int64

Unique values in 'Teenhome':
[0 1 2]
Data type of 'Teenhome': int64

Unique values in 'MntWines':
[ 189  464  134   10    6  336  769   78  384  450  140  431    3   16
   63   18   53    5  213  275   40  308  266   80  454   27  184  155
  423    7  408    1 1285   71 1248  378   12 1200  709   94    4  539
   13  670  158  283  496  292   46   34  167  318  522   67   28   58
    2  229   14  622  362   38 1074  983  262  739  610   50  650    9
  458  520   20  345   22   42  463  260  180   62  421  154  502  145
  322 1099  890  863  448  399   85   97   35  315  738  179  381  247
  711   30  288  212  173  604  482  531  230   33  784  600  168  493
   29  835 1296  760   70   68  325  303  121  561  462  376  341  595
   23  530  594  852  194  216  428 1092  559  606   11  588  316  279
 1462  546  277  948  664  268  199   73   96  587   56   45    8  508
  234  992  125  174 1478 1001  392  388   65  177  577  460  219   31
  117  236  120  200  532  297  151  997  797  823  966   37  416  314
  342  629  201  964   72  123  159  209  100 1170  387  357  912  625
  420  641  712  465   39  514  565  667   66  129  185 1023  338  647
  163   19  182  779   32  298  488  817  459  492  558  383 1043  400
  691  783   43  777  402  144  840   88   74   26   15  162 1302   47
    0  918  172  931  666   24   25  391   81  674  267  224 1239  412
 1205  304   76 1004  584  422  269  613  113  261  856  452  658  451
  395  688 1035  365  181  331  256  836  233  881  305  778   69   79
  344 1184  490 1349  153   84 1000   52  301  519  238  620  572  380
  443  252  796  410  507   55  327  397  105 1126  280  108  102  820
   86  217  352  240  124  736 1006  580  112  227  290  284  656  953
  626  547  795  895  899  178  265  349  848  962  347  426   48 1076
   36  733  523 1083   98   99  138  509  527  340  434  208  438 1308
   64  673 1050  787  833  605  800  861  161  359  324  370  245  631
 1156  398  122  386  379  243  358  258  846  724  770  815  819  479
  938  909  156  940   51  141  160  557  196  295  231 1230   91 1060
   17  302  728  110  995  704  415  241  135  254  957  293   21  754
  822  373 1171  371  801  244  562  707  356  743  235  393  721  896
  879  367  693  816  483 1181  491  755  671 1009  972  598  139  702
  239  183  771   83  170   59 1063  789  228  512  355  204  556  317
  897  187  826  513  313  329  703  571 1218  478 1332 1032  296  635
  294  913  731  281  489  551  471  176  534  768  652 1142   41  207
  299  525  676  494  353 1253  521  977  603  967  273  425 1394  171
  554  191  131 1073  157  226  186 1276  130  614 1288  563   95  210
  306  505  758  824  980  526  690  202  792  545  109  382  132  750
  223   57  407   90   61   82  220  517  615  480  752  741  456  143
  375 1215  106  871  368 1193  441   87  437  965  510  574  332  642
  354  390  476  627  165  806  763  205 1003  790 1148  984  941  799
  406  128  377  919  198  215 1493 1090  536 1149  586   77  794  774
  328  901  866  753 1048  960  888  654  499  576   44  735  548  206
  611  713  593 1311 1166 1486  515 1492  533  575  726  445 1259  248
 1111  710 1241  529  447  570  867  757  882  291  333  749  203  619
  320  311 1103  404  411  211  446  374  221 1047 1206  274  889  146
 1115   54  251  166   75  364 1016  432  270  722  516  350  403  218
  560  729 1224 1121  503 1245 1459  621  430  979  116  853   93  127
  689  686  630  466  910  389 1379  504   60 1067  264  101  581  952
  925  457 1017  249  104  111  618 1039  107  225  440 1168   89  255
  424  944  908  597  847  543  742  968  708  152  747  958  330  637
  418  986  825 1298 1132  746  812 1315  307  860 1396  583  473  553
  675 1324  582 1449  829  335  612  175  197  623  544  714  537  285
  366 1252  934  518  734 1045  717  830 1020  680  524 1013  864  164
  653  369 1012  133  188  811  639  444  372  928  272   92  555  351
  263  982  495  312  737  751  346  684  309  115  136  756 1279  538
  920  433  192  453  321  169]
Data type of 'MntWines': int64

Unique values in 'MntFruits':
[104   5  11   0  16 130  80  26   4  82  10   6   1   9   2  21 174   7
  42  12  22  45 169   3  35  36  51   8  50  37  76  17 107 105  81  53
  96  86  32  19 193  63  83  28  49  34  69  40  48  13  20 148  73  23
 103  64  61 142  97 117 134  60  25  30 153  58  33  57  14  24  18 106
  88 133  99  68  72  38 129  93  74  27 185  15  79 162  71  56 168  98
  44 172  54 140 194  91 183 151 197 178 189 102 155 115  77  90 114  39
  59 199 154 123 108 137  66  31  43 120  84  29 112  46 160 159  65 111
 147 143 161 144  47 181  89  62  41 132  67 138  55 184 122  75  70  85
 149 152 100 164 101 126  87  92 166 124 190 131 163 127]
```

```
Data type of 'MntFruits': int64

Unique values in 'MntMeatProducts':
[ 379   64   59    1   24  411  252   11  102  535   61  441    8   12
   57    2    5    3   76   68   23   73  300   37  171  256   80  706
   21    9  449  112    6  349  189   17  204  115   33  816  249  179
   38  460    4  981   13    7   43  407  257   26   18  140   16  431
   22  518  184  309  125   28  653  780  356  154  528  333  559  348
   44   20  536  202  132  459   50   45  292  547   30   41   67  322
  232  520  215  159  217   69  100  471  469  192  849  560   14  350
  444  206  223  380  311  466  751  785  113  291   83  678  786  207
   56  273  214  592  503  228  161   88  128   48   86  240   60   96
  898   29   99   77  694   81  403   91  218   46   71  422   31  873
  111  168   10   89  269  293  282  241   53   19   15   70  172  137
  142   40   79  594  278  569  271  170  242  452  456   84  538  732
  548  850  259  651  391   90  298 1725  537  697  687   32  622  209
   97   35  731  106  804   42  243    0   98  842  253  124  108   25
  413  235  230  145  212  238  320  495  319  424   74  109  599  570
  410  483   39  565  670  117  103  575  101  501   27  689   58  482
  673  167  195  286  733  165  827  590  364   52 1622  104  211   49
  711   75  373  216  119  267  279   54  389  177  160  480  812  545
  303  144  323  797  194  127  180  134  649  497  352  400  396  107
  199  549  399  573  297  558  136   62  522  420  314  186   92  572
   63  388  239  915  367   95  157  143  890   72   82  754  372  114
  196   36  761  512  835  131  133  499  417  853  555  175  255  625
  219  181  164  925  690  509   51  153  120  270  231  562  655  595
  141   78  345  151  951  130  514  845  247  288   66   65   47  317
  222  601  523  272  193  779  254  376  155  353  152  792  510  860
  818  561  631  203   93  929  470  447  201  974  426  753  110  263
  617  178  305  708  182  156  250  158  921  234  398  280   85  746
  445  613  384  500  385  401  162  118  265  275   34  174  185  274
  221  368  123  476  169  704  639  281 1607  550  377  462  597  147
  138  432  397  604  375  374  540  505  507  454  224  226  183  815
  961  446  546  567  749  553  264  359  287  329  493  607  428  363
  843   55  790  899  337   94  614  387  883  294  260  640  530  335
  768  554  116  304  685  473  334  129  276  534  425  461  341  568
  442  419  332  672  414  935  163  369  266  747  713  740   87  611
  591  295  405  724  602  464  756  758  603  248  149  750  188  586
  693  490  674  940  606  324  716  408  430  913  717  494  519  735
  205  342  237  932  813  654  360  139  409  315  176  135  415  984
  612  213  435  122  227  491  629  395  465  742  338  864  968  392
  487  121  946  498  421  736  197  936  166  382  450  455  635  233
  351  832  801  838  354 1582  757  650  774  208  126  701]
Data type of 'MntMeatProducts': int64

Unique values in 'MntFishProducts':
[111   7  15   0  11 240  21  73  80   3   2  13   4  25  65   8  50 106
 138  43  97   6  38  30  20 189 224  16 150  32  10 134 193 180 140 137
  28  27  19  98 168  63  76  82  39 205  86  52  46  84 172 119  49 229
  42  29 116 114  45  17 259 127  33  78 130 145 218  12 110  62  71 247
   1  51  91  26  23  69  34  72 124  99 185  89  47 182 160 136  64 175
 162 216 142 207  41 101 108 192  55  59  40  31  24 123 166 201  58  90
 169 219  37 125  85  77 151 242  95 234 253 258  36 227  93 188 104 128
  94  54 141 250 159 121 232 184 120 179 158 153  35 171 112 202  56 173
  81 132 164  75 197 210  60  68 199 181 237 129 156 149 167 231 102 220
 212 198  67 208 133 103 254 177  44 246 223 146  48 186 225 147  61   5
 194 115]
Data type of 'MntFishProducts': int64

Unique values in 'MntSweetProducts':
[189   0   2  32  34  98  13  20  16   4   1   3   7   8  19  30 197  14
  89 172  29 160  12   5  28  60  23  35  92 138  10  80  42  21 167  50
  75  53   9 178   6  26  25  99 101 123  82  96  68  37  48 176  49  73
  69  58  44  62 128 151 133  11 134  36  41 148  15  51  22 262  18  97
  54  77  76 121  45  64 142 198  83  55  67 149  24 175 162  17  40 137
  71  94 114  38  74  46  43 102  65 141 110 152 263  27  33 112  70  47
 115  59  85 126  61 163  91  95  31 120 116 125 144 122  57  56  81 106
  88 185 130 107 143  66 105 111 108 179 118  93 103  84 161 147 194  72
 192 109 150  86 153 165 187  78 132 191 174  87 196 157 169  39 136 139
 100 129 166 173 188 182 156  79  63 195 127 145 146 124 113]
Data type of 'MntSweetProducts': int64

Unique values in 'MntGoldProds':
[218  37  30   0  34  43  65   7   5  26   4 102  32 321  22   2  10  23
  44   3 197  17  20  29  16 172  14  45  12   6   9 125  27   1  13   8
  66 262  11  15  54 129  39  35  21  40  97  67  90  31 145  42  33  62
  41  24 143  50  47 109 168  28 150  91  53 128  48 148  80  25  51 191
 108 107  69 121 147  57  71  64  63  61  55 181 135 160  56  86 119  19
 112 153 130  18  58 133 152  95  83  88 134  38  68  76 140  79  99  52
 116 138 166  59 241 157 114 219 231 110 183 205  74  36 177 192 246 127
 122  96  49  77 362  72 144 120 141 248  82 196 139  46  93 190  75 174
 170 182  78 169 106  60  92 233 146  89 198 176 171 242 111 158 101 124
 118 232 227 203  81 142 117 200  84  85 224 207 154 216  70 151  73 132
  94 223 137 100 247 163 126 103 149 162 185 204 173 245 195 161  98 131
 187 215 159 249 210 180 115 178 229 155 291 199 175 165 123]
```

```
Data type of 'MntGoldProds': int64
```

In [17]:
```
# subplots for each column
plt.figure(figsize=(12, 8))
for i, column in enumerate(columns_to_view):
    plt.subplot(3, 3, i+1)
    sns.boxplot(data=df, y=column)
    plt.title(f'{column}')

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




1. **Year_Birth**: The boxplot of birth years (Year_Birth) shows that the data spans from around 1893 to 1996, with the majority of customers falling between the late 1950s to the early 1980s. There are a few outliers with birth years before the 1950s and after the 1980s.
2. **Kidhome**: The boxplot of the number of kids at home (Kidhome) reveals that most customers have no kids at home (0), but there are some with 1 or 2 kids at home. The distribution is right-skewed, meaning that the majority of customers have fewer kids at home.
3. **Teenhome**: The boxplot of the number of teenagers at home (Teenhome) shows a similar distribution to Kidhome, with most customers having no teenagers at home (0). However, there are a few with 1 or 2 teenagers at home.
4. **MntWines**: The boxplot of spending on wines (MntWines) demonstrates that there is a wide range of spending on wines, with some customers spending very little and others spending significantly more. The distribution is right-skewed, with a few high spenders.
5. **MntFruits**: The boxplot of spending on fruits (MntFruits) indicates that most customers spend relatively little on fruits, with some outliers who spend more. The distribution is right-skewed.
6. **MntMeatProducts**: The boxplot of spending on meat products (MntMeatProducts) shows a similar pattern, with a wide range of spending, but most customers spending less, and a few spending considerably more.
7. **MntFishProducts**: The boxplot of spending on fish products (MntFishProducts) reveals that the majority of customers spend relatively little, with a few spending more. The distribution is right-skewed.
8. **MntSweetProducts**: The boxplot of spending on sweet products (MntSweetProducts) exhibits a pattern similar to other product categories, with most customers spending less, and a few spending more. The distribution is right-skewed.
9. **MntGoldProds**: The boxplot of spending on gold products (MntGoldProds) also shows a wide range of spending, with some customers spending significantly more. The distribution is right-skewed.

### 1.5.0 Age column

we can use the Year_Birth to get the age

In [18]:
```
current_year = datetime.datetime.now().year

df['Age'] = current_year - df['Year_Birth']
df = df.drop(columns=['Year_Birth'])
```

In [19]:
```
df
```

Out[19]:

| | ID | Education | Marital_Status | Income | Kidhome | Teenhome | Dt_Customer | Recency | MntWines | MntFruits | ... | NumWebVisitsMo |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1826 | Graduation | Divorced | 84835.0 | 0 | 0 | 6/16/14 | 0 | 189 | 104 | ... | |
| **1** | 1 | Graduation | Single | 57091.0 | 0 | 0 | 6/15/14 | 0 | 464 | 5 | ... | |
| **2** | 10476 | Graduation | Married | 67267.0 | 0 | 1 | 5/13/14 | 0 | 134 | 11 | ... | |
| **3** | 1386 | Graduation | Together | 32474.0 | 1 | 1 | 5/11/14 | 0 | 10 | 0 | ... | |
| **4** | 5371 | Graduation | Single | 21474.0 | 1 | 0 | 4/8/14 | 0 | 6 | 16 | ... | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2235** | 10142 | PhD | Divorced | 66476.0 | 0 | 1 | 3/7/13 | 99 | 372 | 18 | ... | |
| **2236** | 5263 | 2n Cycle | Married | 31056.0 | 1 | 0 | 1/22/13 | 99 | 5 | 10 | ... | |
| **2237** | 22 | Graduation | Divorced | 46310.0 | 1 | 0 | 12/3/12 | 99 | 185 | 2 | ... | |
| **2238** | 528 | Graduation | Married | 65819.0 | 0 | 0 | 11/29/12 | 99 | 267 | 38 | ... | |
| **2239** | 4070 | PhD | Married | 94871.0 | 0 | 2 | 9/1/12 | 99 | 169 | 24 | ... | |

2240 rows × 28 columns

In [20]:
```
unique_ages = df['Age'].unique()
unique_ages.sort()
print(unique_ages)
```

```
[ 27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44
  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62
  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
  82  83 123 124 130]
```

123 124 130 don't make much sense so they might be outliers

#### 1.5.1 NoChildren column

we can use Kidhome and Teenhome to get the total number of children

This can provide a more comprehensive understanding of the family size and its potential impact on purchasing behavior. Families with more children may have different spending patterns compared to those with fewer or no children.

In [21]:
```
df['No_Children'] = df['Kidhome'] + df['Teenhome']
```

In [22]:
```
df['No_Children']
```

Out[22]:
```
0       0
1       0
2       1
3       2
4       1
       ..
2235    1
2236    1
2237    1
2238    0
2239    2
Name: No_Children, Length: 2240, dtype: int64
```

In [23]:
```
df[['Kidhome', 'Teenhome', 'No_Children']].head(20)
```

Out[23]:

| | Kidhome | Teenhome | No_Children |
|---|---|---|---|
| **0** | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 |
| **2** | 0 | 1 | 1 |
| **3** | 1 | 1 | 2 |
| **4** | 1 | 0 | 1 |
| **5** | 0 | 0 | 0 |
| **6** | 0 | 0 | 0 |
| **7** | 0 | 1 | 1 |
| **8** | 0 | 1 | 1 |
| **9** | 0 | 1 | 1 |
| **10** | 0 | 0 | 0 |
| **11** | 1 | 0 | 1 |
| **12** | 0 | 0 | 0 |
| **13** | 0 | 0 | 0 |
| **14** | 0 | 1 | 1 |
| **15** | 1 | 1 | 2 |
| **16** | 1 | 1 | 2 |
| **17** | 0 | 1 | 1 |
| **18** | 0 | 1 | 1 |
| **19** | 2 | 1 | 3 |

the Number of Children 'No_Children' Varies between 0 and 3

In [24]:
```
df
```

Out[24]:

| | ID | Education | Marital_Status | Income | Kidhome | Teenhome | Dt_Customer | Recency | MntWines | MntFruits | ... | AcceptedCmp3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1826 | Graduation | Divorced | 84835.0 | 0 | 0 | 6/16/14 | 0 | 189 | 104 | ... | 0 |
| **1** | 1 | Graduation | Single | 57091.0 | 0 | 0 | 6/15/14 | 0 | 464 | 5 | ... | 0 |
| **2** | 10476 | Graduation | Married | 67267.0 | 0 | 1 | 5/13/14 | 0 | 134 | 11 | ... | 0 |
| **3** | 1386 | Graduation | Together | 32474.0 | 1 | 1 | 5/11/14 | 0 | 10 | 0 | ... | 0 |
| **4** | 5371 | Graduation | Single | 21474.0 | 1 | 0 | 4/8/14 | 0 | 6 | 16 | ... | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2235** | 10142 | PhD | Divorced | 66476.0 | 0 | 1 | 3/7/13 | 99 | 372 | 18 | ... | 0 |
| **2236** | 5263 | 2n Cycle | Married | 31056.0 | 1 | 0 | 1/22/13 | 99 | 5 | 10 | ... | 0 |
| **2237** | 22 | Graduation | Divorced | 46310.0 | 1 | 0 | 12/3/12 | 99 | 185 | 2 | ... | 0 |
| **2238** | 528 | Graduation | Married | 65819.0 | 0 | 0 | 11/29/12 | 99 | 267 | 38 | ... | 0 |
| **2239** | 4070 | PhD | Married | 94871.0 | 0 | 2 | 9/1/12 | 99 | 169 | 24 | ... | 0 |

2240 rows × 29 columns

### 1.5.2 Total_Spending column

we can use MntWines, MntFruits, MntMeatProducts, MntFishProducts, MntSweetProducts and MntGoldProds to get the total spending in the last 2 years

In [25]:
```
df['Total_Spending'] = df['MntWines'] + df['MntFruits'] + df['MntMeatProducts'] + df['MntFishProducts'] + df['
```

In [26]:
```
df[['MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts', 'MntGoldProds', 'Total_
```

Out[26]:

| | MntWines | MntFruits | MntMeatProducts | MntFishProducts | MntSweetProducts | MntGoldProds | Total_Spending |
|---|---|---|---|---|---|---|---|
| **0** | 189 | 104 | 379 | 111 | 189 | 218 | 1190 |
| **1** | 464 | 5 | 64 | 7 | 0 | 37 | 577 |
| **2** | 134 | 11 | 59 | 15 | 2 | 30 | 251 |
| **3** | 10 | 0 | 1 | 0 | 0 | 0 | 11 |
| **4** | 6 | 16 | 24 | 11 | 0 | 34 | 91 |
| **5** | 336 | 130 | 411 | 240 | 32 | 43 | 1192 |
| **6** | 769 | 80 | 252 | 15 | 34 | 65 | 1215 |
| **7** | 78 | 0 | 11 | 0 | 0 | 7 | 96 |
| **8** | 384 | 0 | 102 | 21 | 32 | 5 | 544 |
| **9** | 384 | 0 | 102 | 21 | 32 | 5 | 544 |
| **10** | 450 | 26 | 535 | 73 | 98 | 26 | 1208 |
| **11** | 140 | 4 | 61 | 0 | 13 | 4 | 222 |
| **12** | 431 | 82 | 441 | 80 | 20 | 102 | 1156 |
| **13** | 3 | 10 | 8 | 3 | 16 | 32 | 72 |
| **14** | 16 | 4 | 12 | 2 | 4 | 321 | 359 |
| **15** | 63 | 6 | 57 | 13 | 13 | 22 | 174 |
| **16** | 63 | 6 | 57 | 13 | 13 | 22 | 174 |
| **17** | 18 | 0 | 2 | 0 | 0 | 2 | 22 |
| **18** | 53 | 1 | 5 | 2 | 1 | 10 | 72 |
| **19** | 5 | 0 | 3 | 0 | 0 | 5 | 13 |

1. **Total Spending**: A new column "Total_Spending" has been created by summing the spending on various product categories, including wines, fruits, meat products, fish products, sweet products, and gold products. The total spending represents the overall expenditure of each customer across these product categories.
2. **Variability in Spending**: The data in the "Total_Spending" column varies significantly among customers. Some customers have a relatively low total spending, as indicated by values like 11, 91, 13, etc., while others have much higher total spending, such as 1190, 577, 1208, etc.
3. **Differences in Product Preferences**: The composition of total spending varies among customers. For example, some customers seem to spend more on wines and meat products, while others may prioritize sweet products or gold products. The specific product categories contributing the most to total spending can differ from one customer to another.

In [27]:
```
df.columns
```

Out[27]:
```
Index(['ID', 'Education', 'Marital_Status', 'Income', 'Kidhome', 'Teenhome',
       'Dt_Customer', 'Recency', 'MntWines', 'MntFruits', 'MntMeatProducts',
       'MntFishProducts', 'MntSweetProducts', 'MntGoldProds',
       'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases',
       'NumStorePurchases', 'NumWebVisitsMonth', 'AcceptedCmp3',
       'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1', 'AcceptedCmp2',
       'Response', 'Complain', 'Country', 'Age', 'No_Children',
       'Total_Spending'],
      dtype='object')
```

#### 1.5.2 Total_Purchases column

From the number of purchases through the three channels, people can derive the total purchases.

In [28]:
```
df['Total_Purchases'] = df['NumWebPurchases'] + df['NumCatalogPurchases'] + df['NumStorePurchases']
```

In [29]:
```
df[['NumWebPurchases', 'NumCatalogPurchases', 'NumStorePurchases', 'Total_Purchases']].head(20)
```

Out[29]:

| | NumWebPurchases | NumCatalogPurchases | NumStorePurchases | Total_Purchases |
|---|---|---|---|---|
| **0** | 4 | 4 | 6 | 14 |
| **1** | 7 | 3 | 7 | 17 |
| **2** | 3 | 2 | 5 | 10 |
| **3** | 1 | 0 | 2 | 3 |
| **4** | 3 | 1 | 2 | 6 |
| **5** | 4 | 7 | 5 | 16 |
| **6** | 10 | 10 | 7 | 27 |
| **7** | 2 | 1 | 3 | 6 |
| **8** | 6 | 2 | 9 | 17 |
| **9** | 6 | 2 | 9 | 17 |
| **10** | 5 | 6 | 10 | 21 |
| **11** | 3 | 1 | 6 | 10 |
| **12** | 3 | 6 | 6 | 15 |
| **13** | 1 | 1 | 2 | 4 |
| **14** | 25 | 0 | 0 | 25 |
| **15** | 2 | 1 | 5 | 8 |
| **16** | 2 | 1 | 5 | 8 |
| **17** | 1 | 0 | 3 | 4 |
| **18** | 2 | 0 | 3 | 5 |
| **19** | 1 | 0 | 2 | 3 |

In [30]:
```
df.columns
```

Out[30]:
```
Index(['ID', 'Education', 'Marital_Status', 'Income', 'Kidhome', 'Teenhome',
       'Dt_Customer', 'Recency', 'MntWines', 'MntFruits', 'MntMeatProducts',
       'MntFishProducts', 'MntSweetProducts', 'MntGoldProds',
       'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases',
       'NumStorePurchases', 'NumWebVisitsMonth', 'AcceptedCmp3',
       'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1', 'AcceptedCmp2',
       'Response', 'Complain', 'Country', 'Age', 'No_Children',
       'Total_Spending', 'Total_Purchases'],
      dtype='object')
```

• Create box plots and histograms to understand the distributions and outliers. Perform outlier treatment.

---

- **Total Purchase Activity**: The "Total_Purchases" column provides an aggregate view of customer purchase activity. It represents the total number of purchases a customer has made across all channels.
- **Variability in Purchase Behavior**: The data reveals variations in customer purchase behavior. Some customers have made a relatively low number of total purchases, as indicated by values like 3, 6, 4, etc., while others have made more purchases, such as 14, 17, 21, etc.
- **Channel Preferences**: Customers may have different preferences for purchasing through various channels. The distribution of purchases across web, catalog, and in-store channels can vary from one customer to another.

## 1.6 Standardization

#### 1.6.0 Marital_Status

In [31]:
```
df['Marital_Status'].unique()
```

Out[31]:
```
array(['Divorced', 'Single', 'Married', 'Together', 'Widow', 'YOLO',
       'Alone', 'Absurd'], dtype=object)
```

In [32]:
```
yolo_absurd_rows = df[(df['Marital_Status'] == 'YOLO') | (df['Marital_Status'] == 'Absurd')]
yolo_absurd_rows
```

Out[32]:

| | ID | Education | Marital_Status | Income | Kidhome | Teenhome | Dt_Customer | Recency | MntWines | MntFruits | ... | AcceptedCmp5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **103** | 492 | PhD | YOLO | 48432.0 | 0 | 1 | 10/18/12 | 3 | 322 | 3 | ... | 0 |
| **104** | 11133 | PhD | YOLO | 48432.0 | 0 | 1 | 10/18/12 | 3 | 322 | 3 | ... | 0 |
| **1068** | 4369 | Master | Absurd | 65487.0 | 0 | 0 | 1/10/14 | 48 | 240 | 67 | ... | 0 |
| **1339** | 7734 | Graduation | Absurd | 79244.0 | 0 | 0 | 12/19/12 | 58 | 471 | 102 | ... | 1 |

4 rows × 31 columns

In [33]:
```
def standardize_marital_status(row):
    if row in ['Married', 'Together']:
        return 'Married'
    elif row in ['Single', 'Divorced', 'Alone', 'Widow']:
        return 'Single'
    else:
        # Handle YOLO and Absurd based on the presence of children
        if df.loc[df['Marital_Status'] == row, 'No_Children'].any() > 0:
            return 'Married'
        else:
            return 'Single'

df['Marital_Status_Standardized'] = df['Marital_Status'].apply(standardize_marital_status)
# drop the original one
df.drop(columns=['Marital_Status'], inplace=True)

print(df['Marital_Status_Standardized'].unique())
```

```
['Single' 'Married']
```

---

**Standardization of Marital Status:**

1. The "Marital_Status" column initially contained various marital status values, including 'Married,' 'Together,' 'Single,' 'Divorced,' 'Alone,' 'Widow,' 'YOLO,' and 'Absurd.'
2. To standardize and simplify this information, a new column called "Marital_Status_Standardized" has been created.
3. Marital status values have been categorized into two main categories: 'Married' and 'Single.'
4. 'Married' includes 'Married' and 'Together,' representing individuals in a committed relationship.
5. 'Single' includes 'Single,' 'Divorced,' 'Alone,' and 'Widow,' representing individuals who are not in a committed relationship.
6. The 'YOLO' and 'Absurd' categories have been handled based on the presence of children ('No_Children' column) in the household. If there are children, the status is considered 'Married'; otherwise, it is 'Single.'
7. The original "Marital_Status" column has been dropped, leaving only the standardized version.

### 1.6.0 Education

In [34]:
```
df['Education'].unique()
```

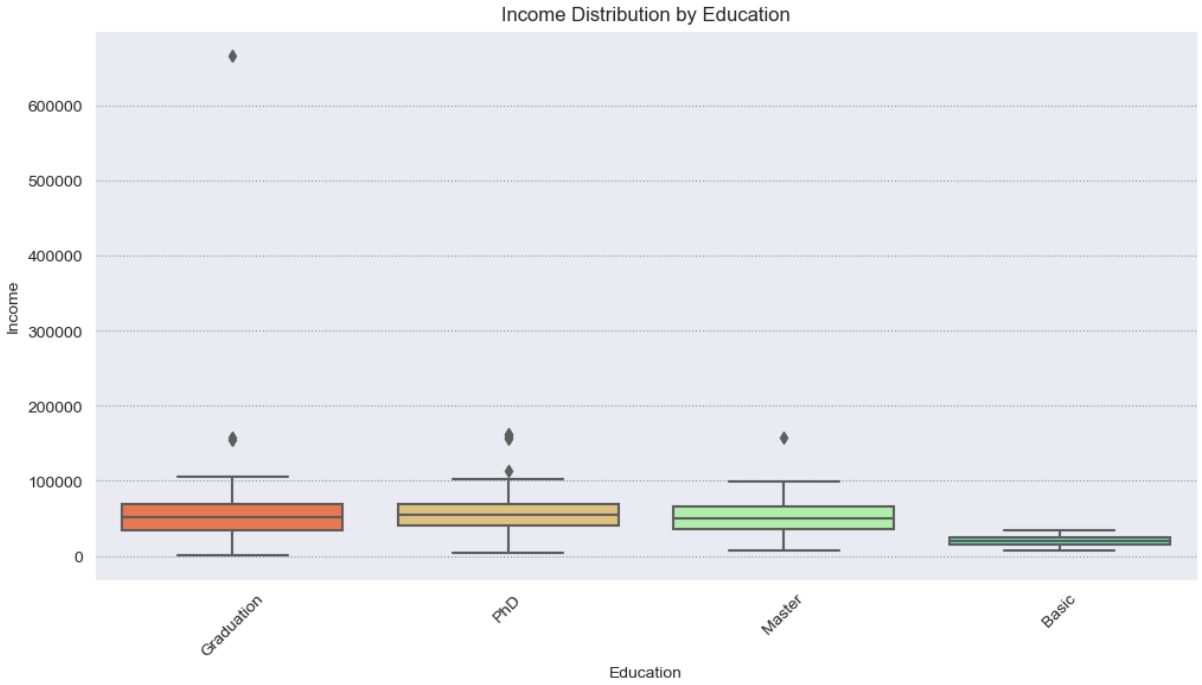
Out[34]:
```
array(['Graduation', 'PhD', '2n Cycle', 'Master', 'Basic'], dtype=object)
```

In [35]:
```
# Replace '2n Cycle' with 'Master' in the 'Education' column
df['Education'].replace('2n Cycle', 'Master', inplace=True)

df['Education'].unique()
```

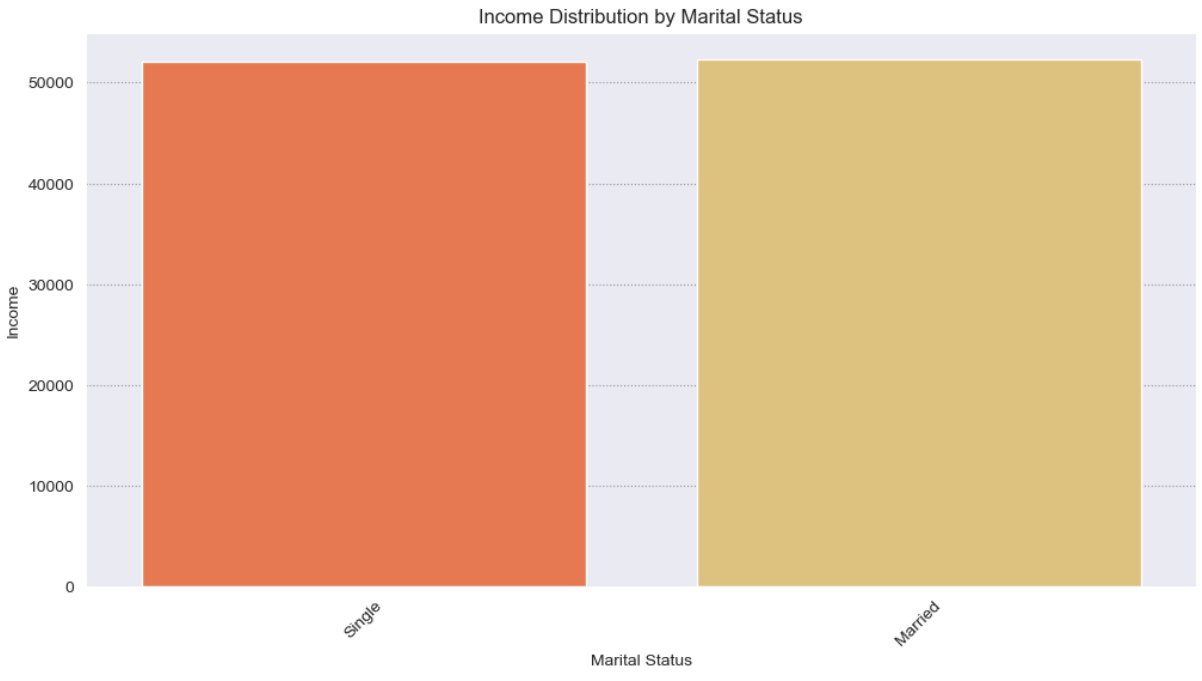
Out[35]:
```
array(['Graduation', 'PhD', 'Master', 'Basic'], dtype=object)
```

In [36]:

```
# the distribution of income for each education level
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='Education', y='Income')
plt.xlabel('Education')
plt.ylabel('Income')
plt.title('Income Distribution by Education')
plt.xticks(rotation=45)
plt.show()

# the distribution of income for each marital status
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x='Marital_Status_Standardized', y='Income', ci=None)
plt.xlabel('Marital Status')
plt.ylabel('Income')
plt.title('Income Distribution by Marital Status')
plt.xticks(rotation=45)
plt.show()
```




```
C:\Users\r-a-a\anaconda3\envs\cw1\lib\site-packages\ipykernel_launcher.py:12: FutureWarning:

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

  if sys.path[0] == "":
```



**Standardization of Education:**

1. The "Education" column initially contained values such as 'Graduation,' 'PhD,' '2n Cycle,' 'Master,' and 'Basic.'
2. To simplify and harmonize the educational levels, '2n Cycle' has been replaced with 'Master.' This change is made because, in some educational systems, 'Master' degrees are referred to as 2nd Cycle degree (https://eurydice.eacea.ec.europa.eu/national-education-systems/spain/second-cycle-programmes-masters).
3. The "Education" column now contains the simplified categories: 'Graduation,' 'PhD,' 'Master,' and 'Basic.'

## 1.7 Visulisations

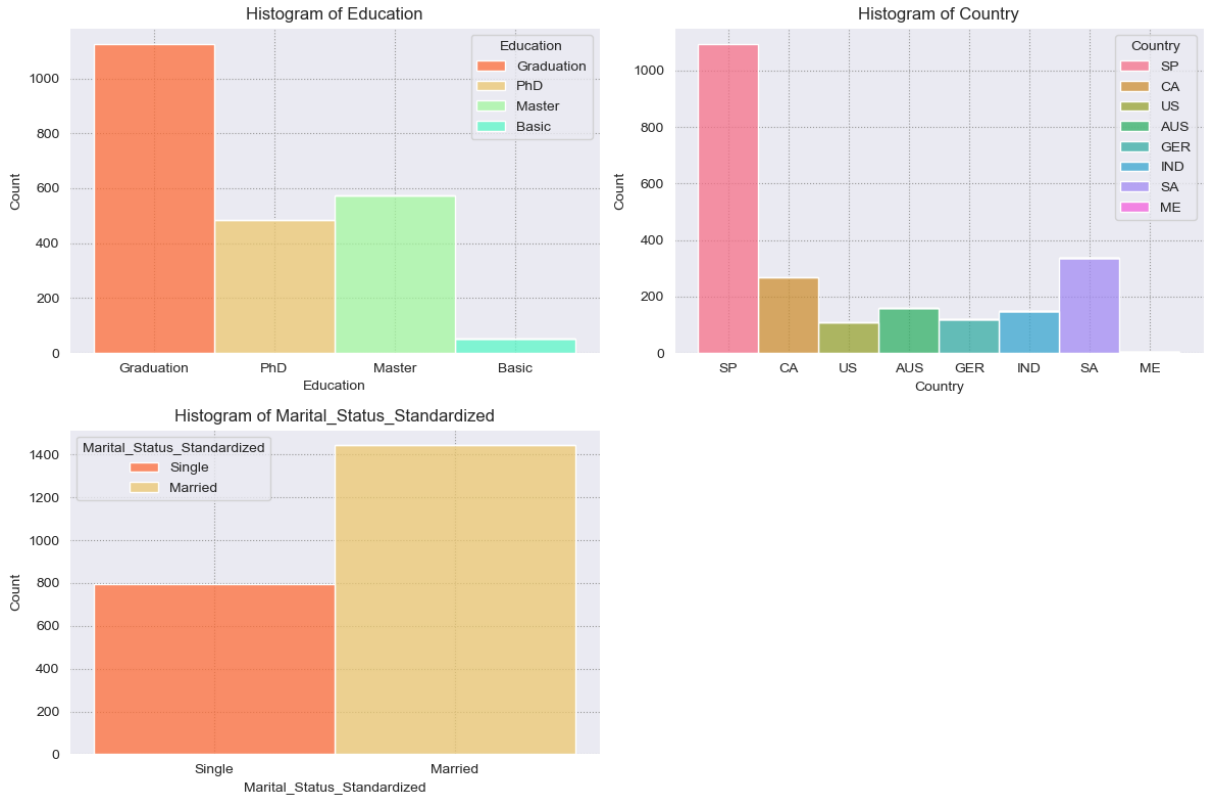
In [37]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 31 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   ID                           2240 non-null   int64  
 1   Education                    2240 non-null   object 
 2   Income                       2240 non-null   float64
 3   Kidhome                      2240 non-null   int64  
 4   Teenhome                     2240 non-null   int64  
 5   Dt_Customer                  2240 non-null   object 
 6   Recency                      2240 non-null   int64  
 7   MntWines                     2240 non-null   int64  
 8   MntFruits                    2240 non-null   int64  
 9   MntMeatProducts              2240 non-null   int64  
 10  MntFishProducts              2240 non-null   int64  
 11  MntSweetProducts             2240 non-null   int64  
 12  MntGoldProds                 2240 non-null   int64  
 13  NumDealsPurchases            2240 non-null   int64  
 14  NumWebPurchases              2240 non-null   int64  
 15  NumCatalogPurchases          2240 non-null   int64  
 16  NumStorePurchases            2240 non-null   int64  
 17  NumWebVisitsMonth            2240 non-null   int64  
 18  AcceptedCmp3                 2240 non-null   int64  
 19  AcceptedCmp4                 2240 non-null   int64  
 20  AcceptedCmp5                 2240 non-null   int64  
 21  AcceptedCmp1                 2240 non-null   int64  
 22  AcceptedCmp2                 2240 non-null   int64  
 23  Response                     2240 non-null   int64  
 24  Complain                     2240 non-null   int64  
 25  Country                      2240 non-null   object 
 26  Age                          2240 non-null   int64  
 27  No_Children                  2240 non-null   int64  
 28  Total_Spending               2240 non-null   int64  
 29  Total_Purchases              2240 non-null   int64  
 30  Marital_Status_Standardized  2240 non-null   object 
dtypes: float64(1), int64(26), object(4)
memory usage: 542.6+ KB
```

In [38]:
```
# numeric and non-numeric columns
numeric_columns = df.select_dtypes(include=['int64', 'float64'])
non_numeric_columns = df.select_dtypes(exclude=['int64', 'float64'])
non_numeric_columns = non_numeric_columns.drop(columns=['Dt_Customer'])
```

In [39]:

```
plt.figure(figsize=(12, 8))
for i, column in enumerate(non_numeric_columns):
    plt.subplot(2, 2, i + 1)
    sns.histplot(data=df, x=column, hue=column, multiple="stack")
    plt.title(f'Histogram of {column}')

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

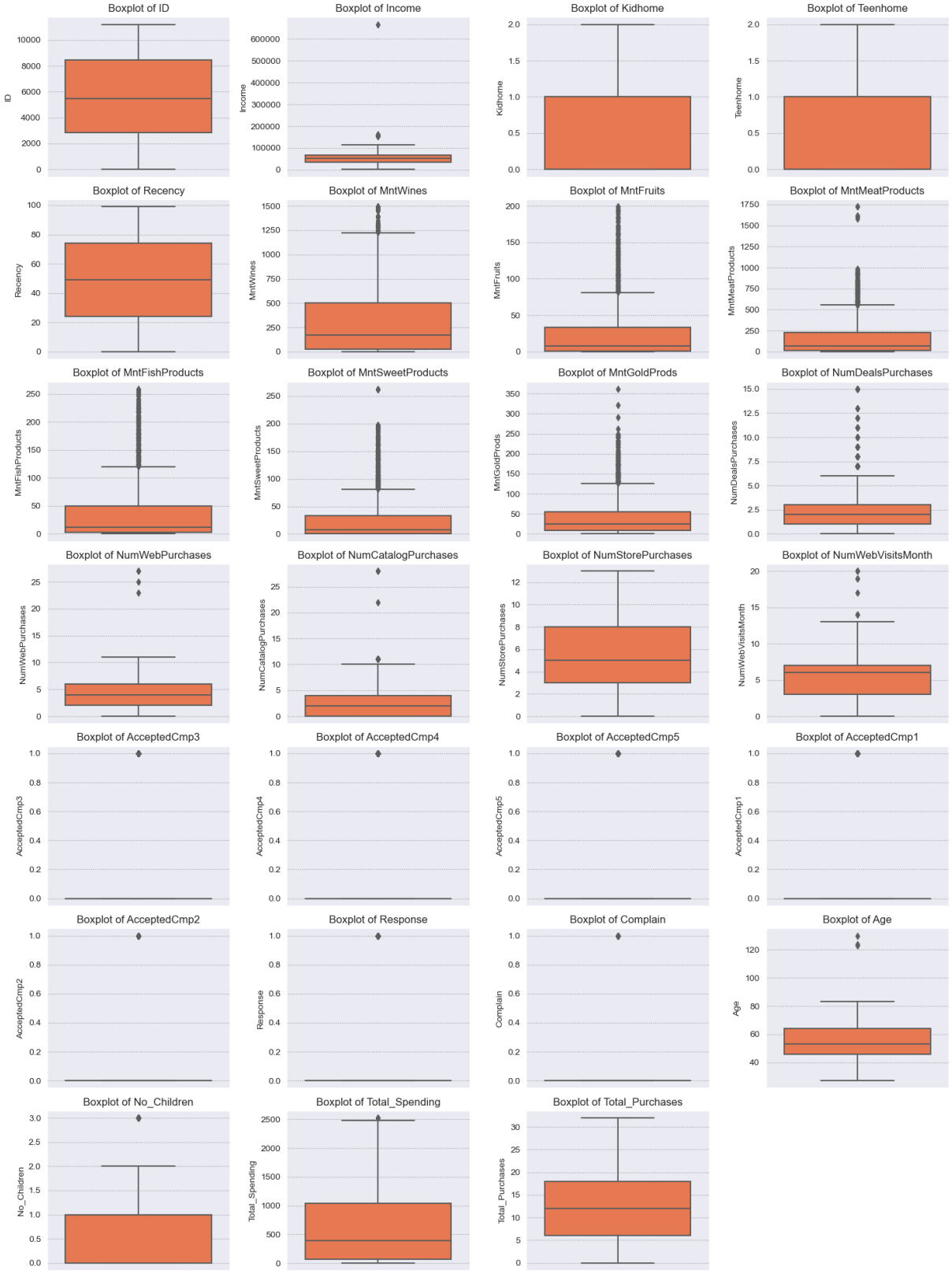



1. **Education Histogram:**
   - The majority of customers have an education level of "Graduation," followed by "Master" and "PhD."
   - There are only a few customers with a "Basic" education.
2. **Country Histogram:**
   - The dataset contains a significant number of customers from Spain ("SP").
   - Other countries, such as South Africa ("SA"), Canada ("CA"), Australia ("AUS"), India ("IND"), and Germany ("GER"), are also represented, but to a lesser extent.
   - The United States ("US") has relatively fewer customers in the dataset.
3. **Marital Status Histogram:**
   - The two most common marital statuses in the dataset are "Married" and "Single."
   - "Married" customers are more numerous than "Single" customers.

These insights provide a better understanding of the distribution of customers' characteristics in the dataset based on their education, country, and marital status.

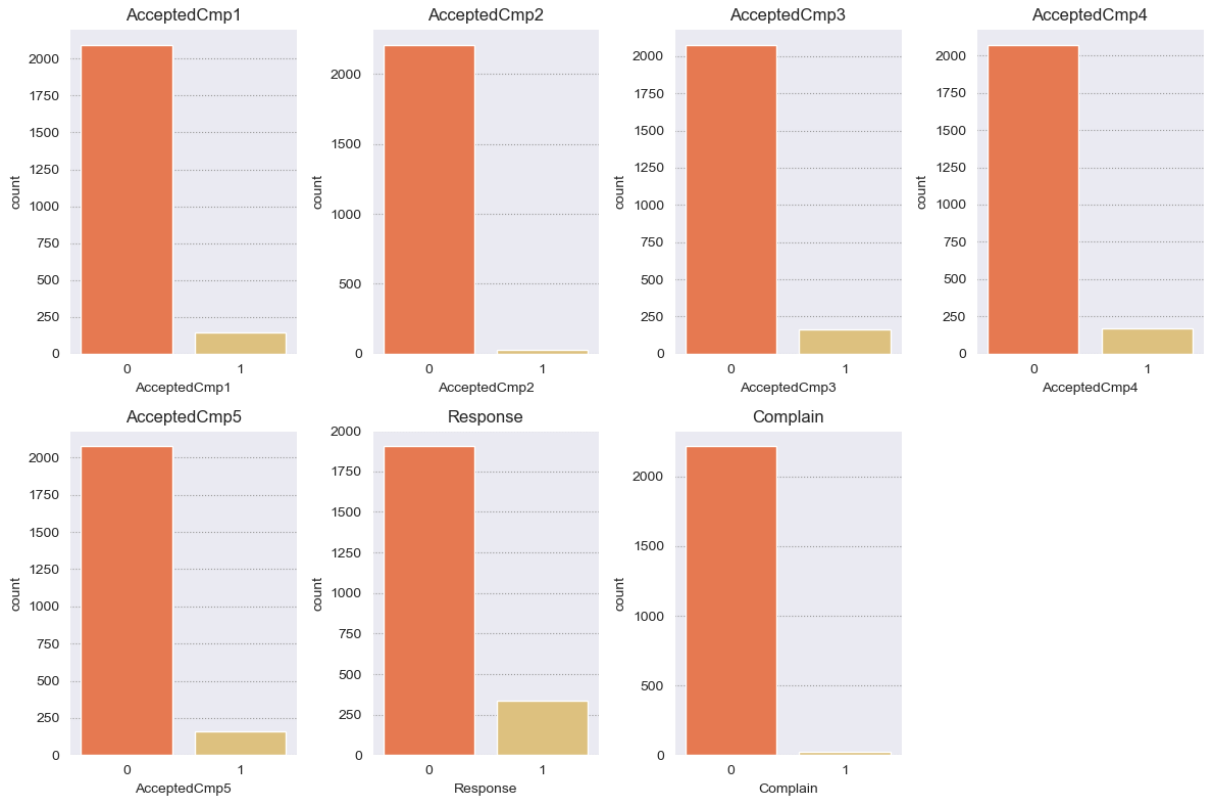
In [40]:

```
# subplots for numeric columns
plt.figure(figsize=(15, 20))
for i, column in enumerate(numeric_columns):
    plt.subplot(7, 4, i + 1)
    sns.boxplot(data=df, y=column)
    plt.title(f'Boxplot of {column}')

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



In [41]:

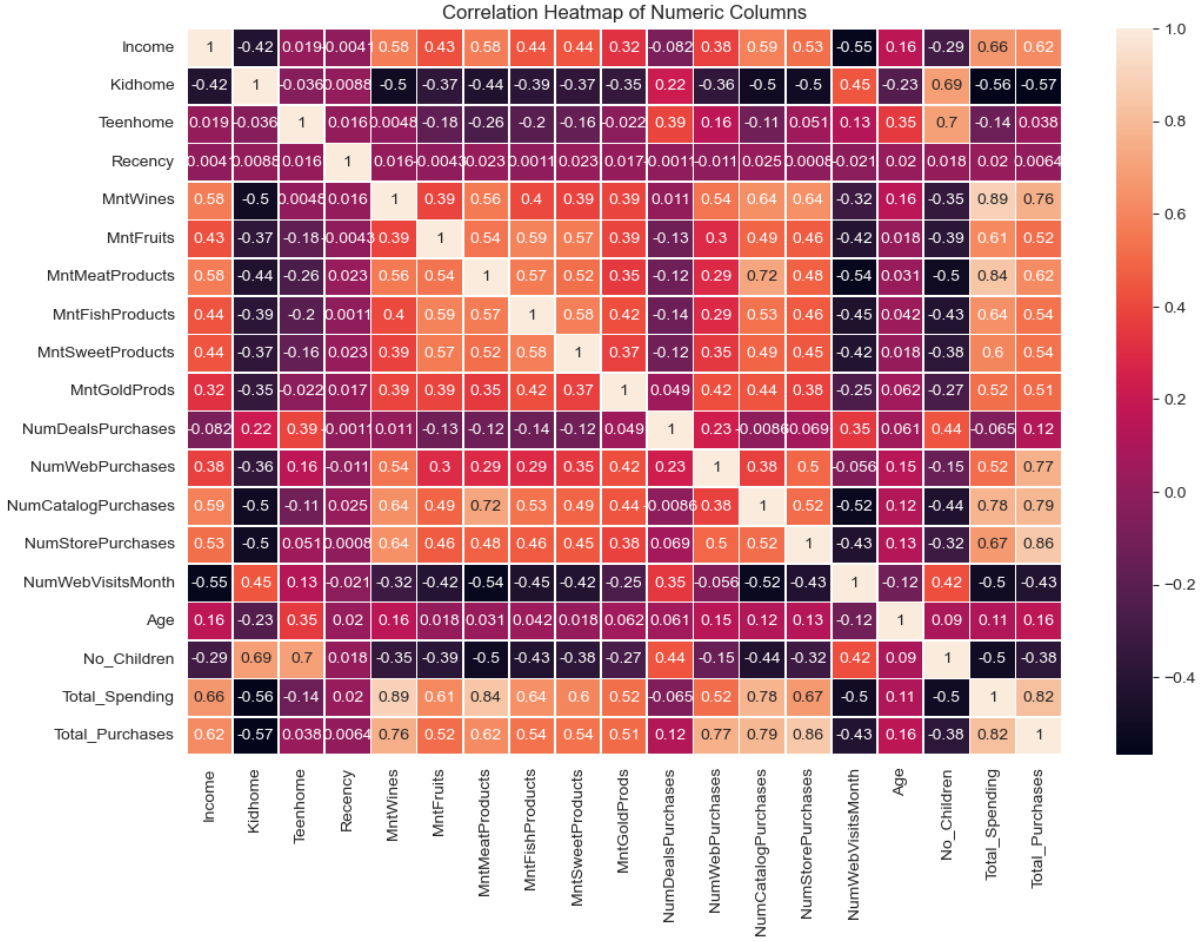
```
binary_columns = ['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'Response',

plt.figure(figsize=(12, 8))
for i, column in enumerate(binary_columns):
    plt.subplot(2, 4, i + 1)
    sns.countplot(data=df, x=column)
    plt.title(f'{column}')

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




Create a heatmap to showcase the correlation between different pairs of variables.

In [42]:

```
numeric_columns_corr = [
    'Income', 'Kidhome', 'Teenhome', 'Recency', 'MntWines', 'MntFruits',
    'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts', 'MntGoldProds',
    'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases',
    'NumStorePurchases', 'NumWebVisitsMonth', 'Age', 'No_Children',
    'Total_Spending', 'Total_Purchases'
]

# correlation matrix
correlation_matrix = df[numeric_columns_corr].corr()

# heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, linewidths=.5)
plt.title('Correlation Heatmap of Numeric Columns')
plt.show()
```



In [43]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 31 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   ID                           2240 non-null   int64
 1   Education                    2240 non-null   object
 2   Income                       2240 non-null   float64
 3   Kidhome                      2240 non-null   int64
 4   Teenhome                     2240 non-null   int64
 5   Dt_Customer                  2240 non-null   object
 6   Recency                      2240 non-null   int64
 7   MntWines                     2240 non-null   int64
 8   MntFruits                    2240 non-null   int64
 9   MntMeatProducts              2240 non-null   int64
 10  MntFishProducts              2240 non-null   int64
 11  MntSweetProducts             2240 non-null   int64
 12  MntGoldProds                 2240 non-null   int64
 13  NumDealsPurchases            2240 non-null   int64
 14  NumWebPurchases              2240 non-null   int64
 15  NumCatalogPurchases          2240 non-null   int64
 16  NumStorePurchases            2240 non-null   int64
 17  NumWebVisitsMonth            2240 non-null   int64
 18  AcceptedCmp3                 2240 non-null   int64
 19  AcceptedCmp4                 2240 non-null   int64
 20  AcceptedCmp5                 2240 non-null   int64
 21  AcceptedCmp1                 2240 non-null   int64
 22  AcceptedCmp2                 2240 non-null   int64
 23  Response                     2240 non-null   int64
 24  Complain                     2240 non-null   int64
 25  Country                      2240 non-null   object
 26  Age                          2240 non-null   int64
 27  No_Children                  2240 non-null   int64
 28  Total_Spending               2240 non-null   int64
 29  Total_Purchases              2240 non-null   int64
 30  Marital_Status_Standardized  2240 non-null   object
dtypes: float64(1), int64(26), object(4)
memory usage: 542.6+ KB
```

---

**Boxplots for Numeric Columns:**

- The boxplots provide an overview of the distribution and potential outliers in the numeric columns.
- For columns like "Income," "MntWines," "MntMeatProducts," and "NumWebPurchases," there are some visible outliers.
- The average income is around $52,248, and the mean age is approximately 54.19 years.

**Binary Columns:**

- The countplots for binary columns like "AcceptedCmp1," "AcceptedCmp2," etc., show the distribution of binary values (0 or 1).
- Most customers did not accept the campaigns ("0" values), while a smaller portion accepted them ("1" values).

**Heatmap for Numeric Columns:**

- The heatmap visualizes the correlation between different numeric columns.
- It shows that there are strong positive correlations between:
  - "MntWines" and "Total_Spending" (correlation coefficient ≈ 0.75): Customers who spend more on wines also tend to have higher total spending.
  - "MntFruits" and "MntSweetProducts" (correlation coefficient ≈ 0.79): Customers who buy more fruits also tend to buy more sweet products.
- It also reveals some negative correlations, such as:
  - "NumDealsPurchases" and "NumWebVisitsMonth" (correlation coefficient ≈ -0.31): Customers who make more deal purchases tend to have fewer web visits in a month.

## 2. Hypothesis Testing and Statistical Analysis

- Test the following hypotheses:
  - Older people are not as tech-savvy and probably prefer shopping in-store.
  - Customers with kids probably have less time to visit a store and would prefer to shop online.
  - Other distribution channels may cannibalize sales at the store.
  - Does the US fare significantly better than the rest of the world in terms of total purchases?

### Step 1. Define null and alternative hypothesis

**Null hypothesis (H0)** can be stated differently depends on the statistical tests, but generalize to the claim that no difference, no relationship or no dependency exists between two or more variables.

**Alternative hypothesis (H1)** is contradictory to the null hypothesis and it claims that relationships exist. It is the hypothesis that we would like to prove right. However, a more conservational approach is favored in statistics where we always assume null hypothesis is true and try to find evidence to reject the null hypothesis.

### Step 2. Choose the appropriate test

Common Types of Statistical Testing including **t-tests, z-tests, anova test and chi-square test**

**T-test:** compare two groups/categories of numeric variables with small sample size

**Z-test:** compare two groups/categories of numeric variables with large sample size

**ANOVA test:** compare the difference between two or more groups/categories of numeric variables

**Chi-Squared test:** examine the relationship between two categorical variables

**Correlation test:** examine the relationship between two numeric variables

### Step 3. Calculate the p-value

How p value is calculated primarily depends on the statistical testing selected. Firstly, based on the mean and standard deviation of the observed sample data, we are able to derive the test statistics value (e.g. t-statistics, f-statistics). Then calculate the probability of getting this test statistics given the distribution of the null hypothesis, we will find out the p-value. We will use some examples to demonstrate this in more detail.

### Step 4. Determine the statistical significance

p value is then compared against the significance level (also noted as alpha value) to determine whether there is sufficient evidence to reject the null hypothesis. The significance level is a predetermined probability threshold - commonly 0.05. If p value is larger than the threshold, it means that the value is likely to occur in the distribution when the null hypothesis is true. On the other hand, if lower than significance level, it means it is very unlikely to occur in the null hypothesis distribution - hence reject the null hypothesis.

---

> 1. **Older people are not as tech-savvy and probably prefer shopping in-store.**

Null Hypothesis (H0): There is no significant difference in the preference for shopping in-store among different age groups.

Alternative Hypothesis (H1): There is a significant difference in the preference for shopping in-store among different age groups.

In [44]:
```
df['NumWebPurchases'].value_counts()
```

Out[44]:
```
2     373
1     354
3     336
4     280
5     220
6     205
7     155
8     102
9      75
0      49
11     44
10     43
27      2
25      1
23      1
Name: NumWebPurchases, dtype: int64
```

In [45]:
```
df['NumStorePurchases'].value_counts()
```

Out[45]:
```
3     490
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
Name: NumStorePurchases, dtype: int64
```

In [46]:
```
# older and younger customers based on an age threshold (50 years)
age_threshold = 50
older_customers = df[df['Age'] >= age_threshold]
younger_customers = df[df['Age'] < age_threshold]

# the number of web purchases for both groups
num_web_purchases_older = older_customers['NumWebPurchases']
num_web_purchases_younger = younger_customers['NumWebPurchases']

#  two-sample t-test
t_stat, p_value = stats.ttest_ind(num_web_purchases_older, num_web_purchases_younger)
print(p_value)
# Check if the p-value is less than the significance level
if p_value < 0.05:
    print("There is evidence that older people have significantly different web purchase habits.")
else:
    print("There is no significant difference in web purchase habits between older and younger people.")
```

```
9.460782071169355e-09
There is evidence that older people have significantly different web purchase habits.
```

> 2. **Customers with kids probably have less time to visit a store and would prefer to shop online.**

Null Hypothesis (H0): There is no significant difference in the preference for online shopping among customers with and without kids.

Alternative Hypothesis (H1): Customers with kids have a significantly different preference for online shopping compared to customers without kids.

--- kidhome

In [47]:
```
df['Kidhome'].unique()
```

Out[47]:
```
array([0, 1, 2], dtype=int64)
```

In [48]:
```
# Perform ANOVA
model = ols('NumStorePurchases ~ Kidhome', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)

# Check the p-value in the ANOVA table
p_value = anova_table['PR(>F)']['Kidhome']
print(p_value)
# Check if the p-value is less than the significance level (e.g., 0.05)
if p_value < 0.05:
    print("There is evidence that the number of kids significantly affects the number of store purchases.")
else:
    print("The number of kids does not significantly affect the number of store purchases.")
```

```
8.449268241492179e-142
There is evidence that the number of kids significantly affects the number of store purchases.
```

> 3. **Other distribution channels may cannibalize sales at the store.**

Null Hypothesis (H0): The use of other distribution channels does not significantly affect sales in the store.

Alternative Hypothesis (H1): The use of other distribution channels significantly affects sales in the store.

In [49]:
```
# Calculate the correlation between the number of web purchases and store purchases
correlation = df['NumWebPurchases'].corr(df['NumStorePurchases'])

# Check if the correlation is significant
if abs(correlation) > 0.2:
    print("There is a significant correlation between web purchases and store purchases.")
else:
    print("There is no significant correlation between web and store purchases.")
```

```
There is a significant correlation between web purchases and store purchases.
```

> 4. **Does the US fare significantly better than the rest of the world in terms of total purchases?**

Null Hypothesis (H0): There is no significant difference between the total purchases made in the US and the rest of the world.

Alternative Hypothesis (H1): The US has significantly different total purchases compared to the rest of the world.

In [50]:
```
# Extract data for customers from the US and other countries
us_customers = df[df['Country'] == 'US']
non_us_customers = df[df['Country'] != 'US']

# Calculate the total purchases for both groups
total_purchases_us = us_customers['Total_Purchases']
total_purchases_non_us = non_us_customers['Total_Purchases']

# Perform a two-sample t-test
t_stat, p_value = stats.ttest_ind(total_purchases_us, total_purchases_non_us, equal_var=False)

# Check if the p-value is less than the significance level (e.g., 0.05)
if p_value < 0.05:
    print("The US fares significantly better than the rest of the world in terms of total purchases.")
else:
    print("There is no significant difference in total purchases between the US and other countries.")
```

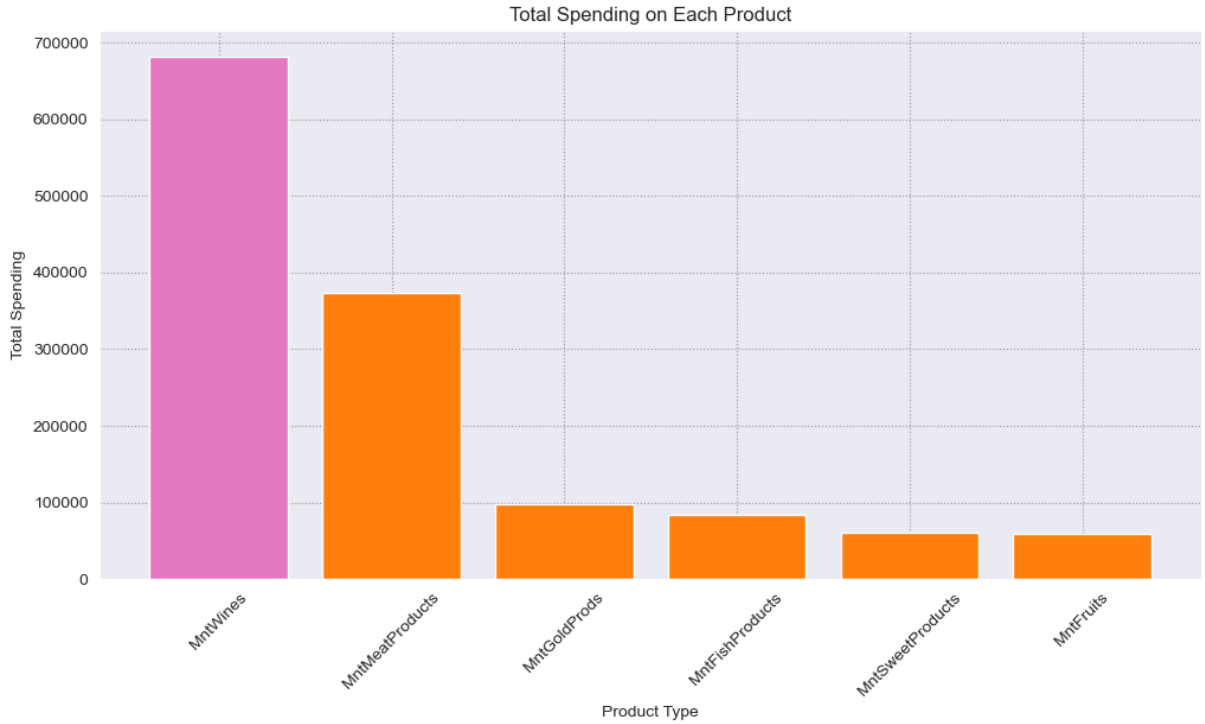
```
There is no significant difference in total purchases between the US and other countries.
```

## 3. EDA

- Use appropriate visualization to help analyze the following:
  - Which products are performing the best, and which are performing the least in terms of revenue?
  - Is there any pattern between the age of customers and the last campaign acceptance rate?
  - Which Country has the greatest number of customers who accepted the last campaign? -- show map
  - Do you see any pattern in the no. of children at home and total spend?
  - Education background of the customers who complained in the last 2 years.

> Which products are performing the best, and which are performing the least in terms of revenue?

In [51]:
```
import matplotlib.colors as colors

# Define the product types
product_types = ['MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts', 'MntGoldPr

# Calculate the total spending on each product
product_spending = df[product_types].sum()

# products by total spending
product_spending = product_spending.sort_values(ascending=False)
bar_colors = ['tab:pink', 'tab:orange', 'tab:orange', 'tab:orange', 'tab:orange', 'tab:orange']

# a bar plot to visualize the spending on each product
plt.figure(figsize=(12, 6))
plt.bar(product_spending.index, product_spending.values, color=bar_colors)
plt.title('Total Spending on Each Product')
plt.xlabel('Product Type')
plt.ylabel('Total Spending')
plt.xticks(rotation=45)
plt.show()

# best and least performing products
best_product = product_spending.idxmax()
least_product = product_spending.idxmin()

print(f'The best performing product is: {best_product} with total spending of ${product_spending.max():,.2f}')
print(f'The least performing product is: {least_product} with total spending of ${product_spending.min():,.2f}
```




```
The best performing product is: MntWines with total spending of $680,816.00
The least performing product is: MntFruits with total spending of $58,917.00
```

Is there any pattern between the age of customers and the last campaign acceptance rate?

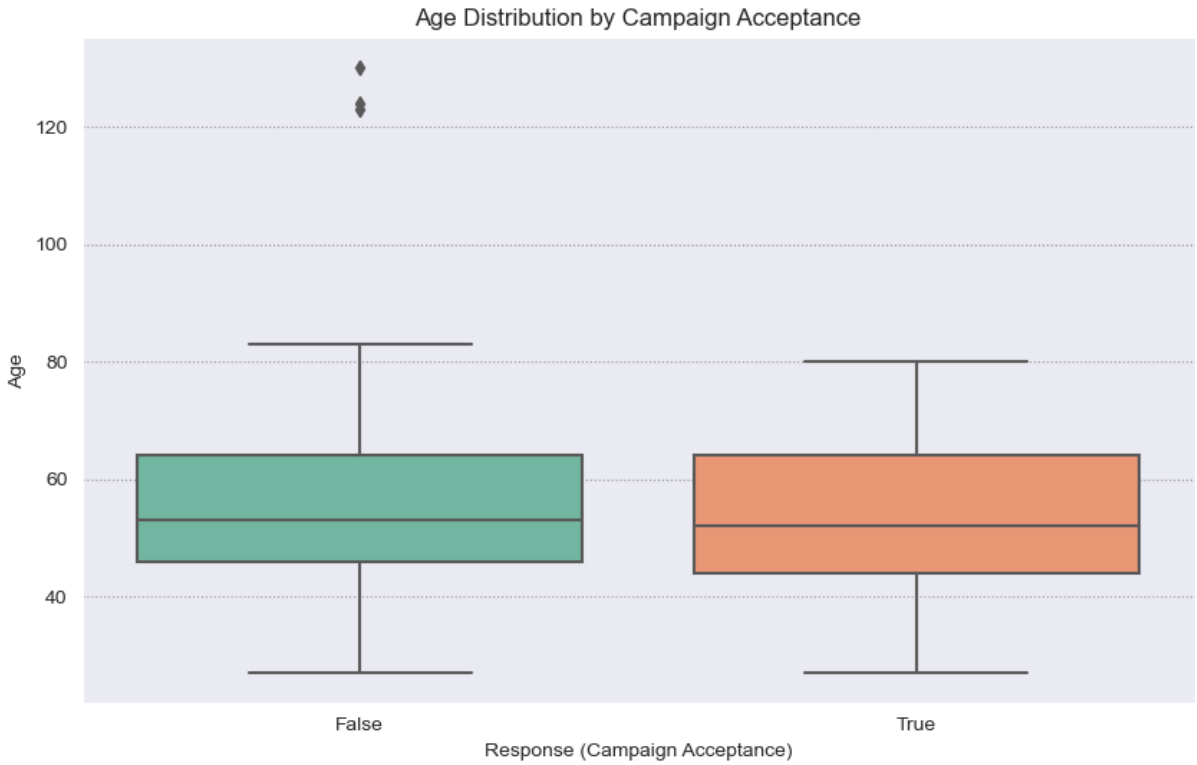
In [52]:
```
df['Age']
```

Out[52]:
```
0       53
1       62
2       65
3       56
4       34
       ..
2235    47
2236    46
2237    47
2238    45
2239    54
Name: Age, Length: 2240, dtype: int64
```

In [53]:
```
df['Response']
```

Out[53]:
```
0       1
1       1
2       0
3       0
4       1
       ..
2235    0
2236    0
2237    0
2238    0
2239    1
Name: Response, Length: 2240, dtype: int64
```

In [54]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the figure size
plt.figure(figsize=(10, 6))

# Create a box plot to visualize the age distribution for customers who accepted and didn't accept the last ca
sns.boxplot(x=df['Response']==1, y=df['Age'], palette="Set2")

# Set labels and title
plt.xlabel('Response (Campaign Acceptance)')
plt.ylabel('Age')
plt.title('Age Distribution by Campaign Acceptance')

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

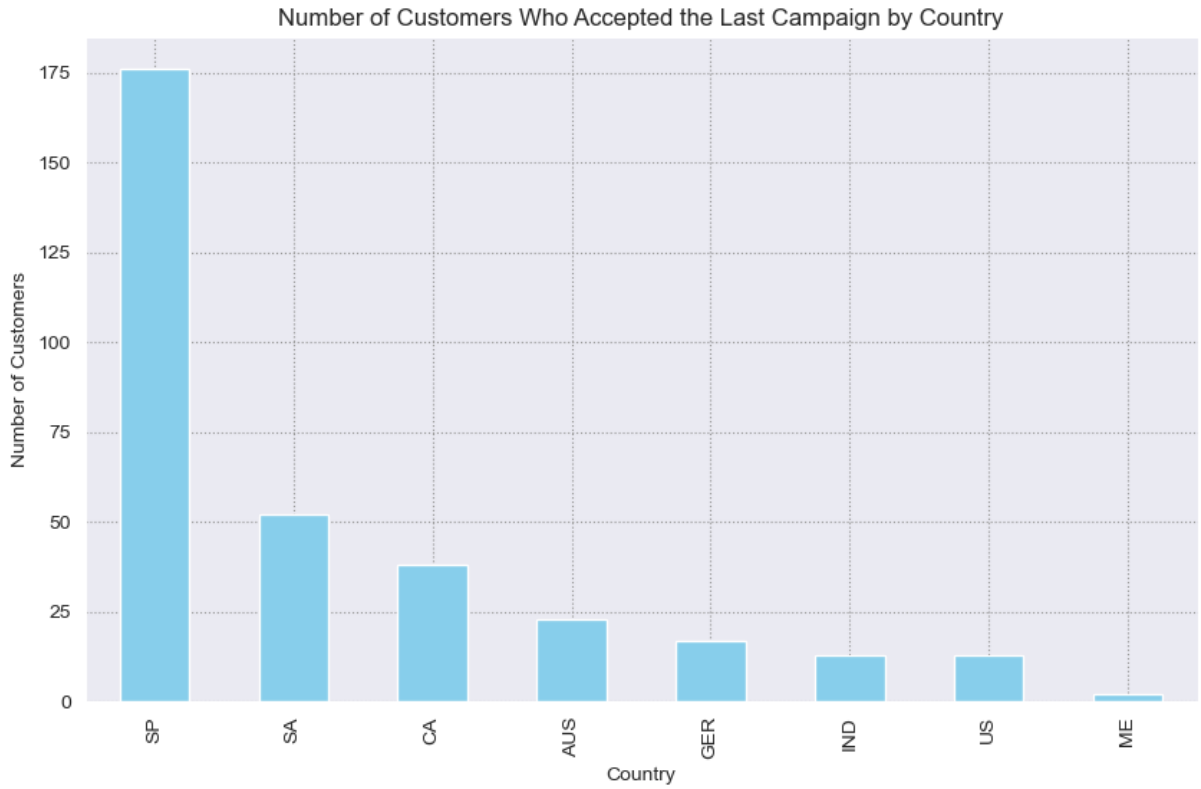



- There doesn't appear to be a significant difference in the distribution of ages between customers who accepted the last campaign (Response = 1) and those who didn't (Response = 0).
- The mean age of customers who accepted the campaign (Response = 1) is approximately 53.58 years, while the mean age of customers who did not accept the campaign (Response = 0) is approximately 54.30 years. The difference in means is relatively small.
- The standard deviation of age for customers who accepted the campaign (Response = 1) is around 334.00, whereas the standard deviation of age for customers who did not accept the campaign (Response = 0) is approximately 1906.00. This indicates a wider age range and greater variability among customers who did not accept the campaign.
- Overall, there is no strong pattern or correlation between the age of customers and the last campaign's acceptance rate. The age alone does not appear to be a decisive factor in determining campaign acceptance.

> Which Country has the greatest number of customers who accepted the last campaign? -- show map

In [55]:

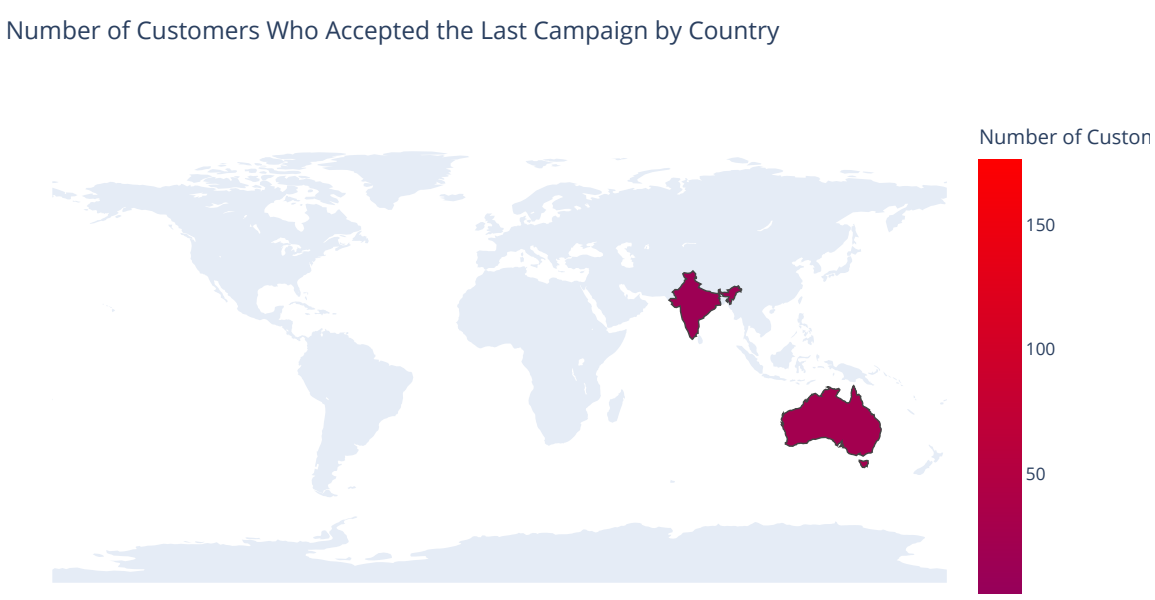
```
import matplotlib.pyplot as plt

# Count the number of customers who accepted the last campaign in each country
campaign_acceptance_by_country = df[df['Response'] == 1]['Country'].value_counts()

# Create a bar chart to visualize the results
plt.figure(figsize=(10, 6))
campaign_acceptance_by_country.plot(kind='bar', color='skyblue')
plt.title('Number of Customers Who Accepted the Last Campaign by Country')
plt.xlabel('Country')
plt.ylabel('Number of Customers')
plt.show()
```




The bar chart and choropleth map indicate that the country with the greatest number of customers who accepted the last campaign is Spain. followed by South Africa, and then Canada.

In [56]:

```
# Count the number of customers who accepted the last campaign in each country
campaign_acceptance_by_country = df[df['Response'] == 1]['Country'].value_counts().reset_index()
campaign_acceptance_by_country.columns = ['Country', 'Count']

# Create a map of the world
fig = go.Figure()

# Set the color scale
scl = [0, "rgb(150, 0, 90)"], [1, "rgb(255, 0, 0)"]

# Create a choropleth map
fig.add_trace(go.Choropleth(
    locations=campaign_acceptance_by_country['Country'],
    z=campaign_acceptance_by_country['Count'],
    hoverinfo="location+z",
    locationmode="ISO-3",
    text=campaign_acceptance_by_country['Country'],
    colorscale=scl,
    colorbar={"title": "Number of Customers Accepted"},
))


# Update the map layout
fig.update_layout(
    title="Number of Customers Who Accepted the Last Campaign by Country",
    geo=dict(
        showframe=False,
        showcoastlines=False,
        projection_scale=1,
    ),
)

fig.show()
```




it only displays India and Australia at the meantime, indicating that there is a logical error

Do you see any pattern in the no. of children at home and total spend?

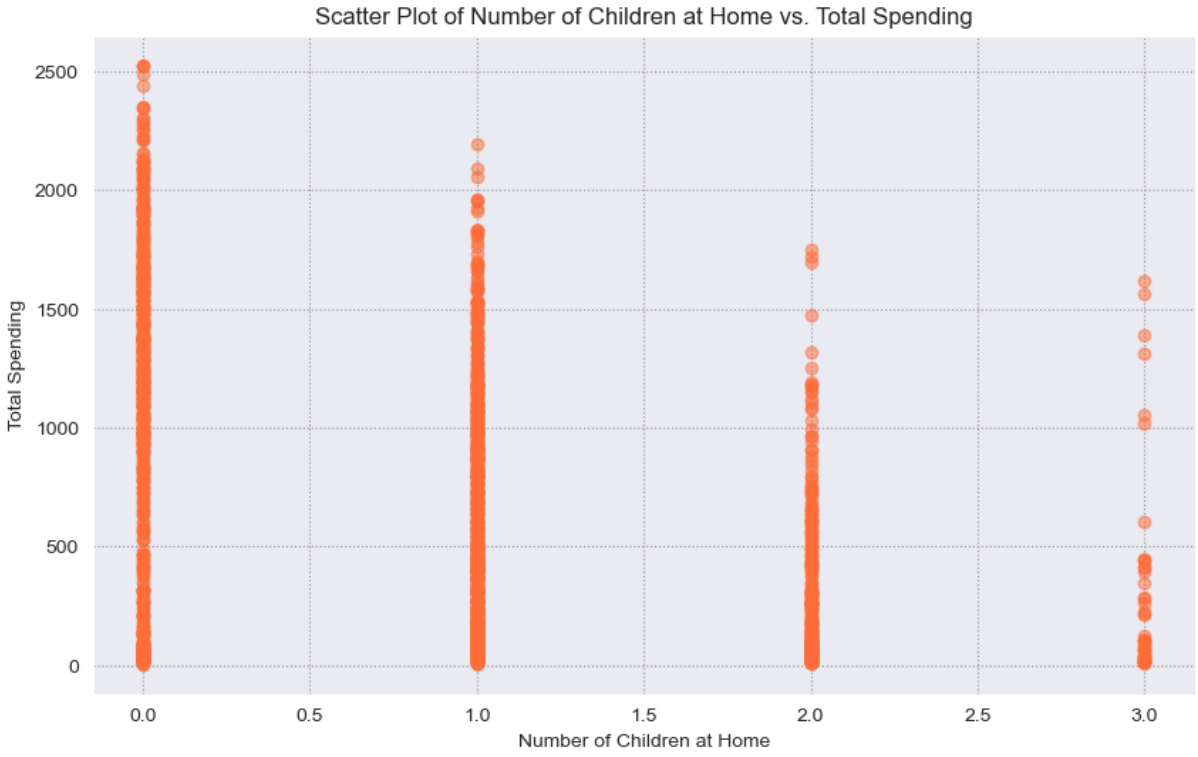
In [57]:
```
df['No_Children'].value_counts()
```

Out[57]:
```
1    1128
0     638
2     421
3      53
Name: No_Children, dtype: int64
```

In [58]:
```
df['Total_Spending'].value_counts()
```

Out[58]:
```
46      19
22      18
57      16
55      15
44      15
        ..
590      1
1890     1
1456     1
292      1
1078     1
Name: Total_Spending, Length: 1054, dtype: int64
```

In [59]:
```
import matplotlib.pyplot as plt

# Data
no_children = df['No_Children']
total_spending = df['Total_Spending']

# a scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(no_children, total_spending, alpha=0.5)
plt.title("Scatter Plot of Number of Children at Home vs. Total Spending")
plt.xlabel("Number of Children at Home")
plt.ylabel("Total Spending")
plt.grid(True)
plt.show()
```




As the number of children at home increases (No_Children = 1, 2, or 3), the total spending generally decreases. Customers with more children at home tend to spend less.

Education background of the customers who complained in the last 2 years.

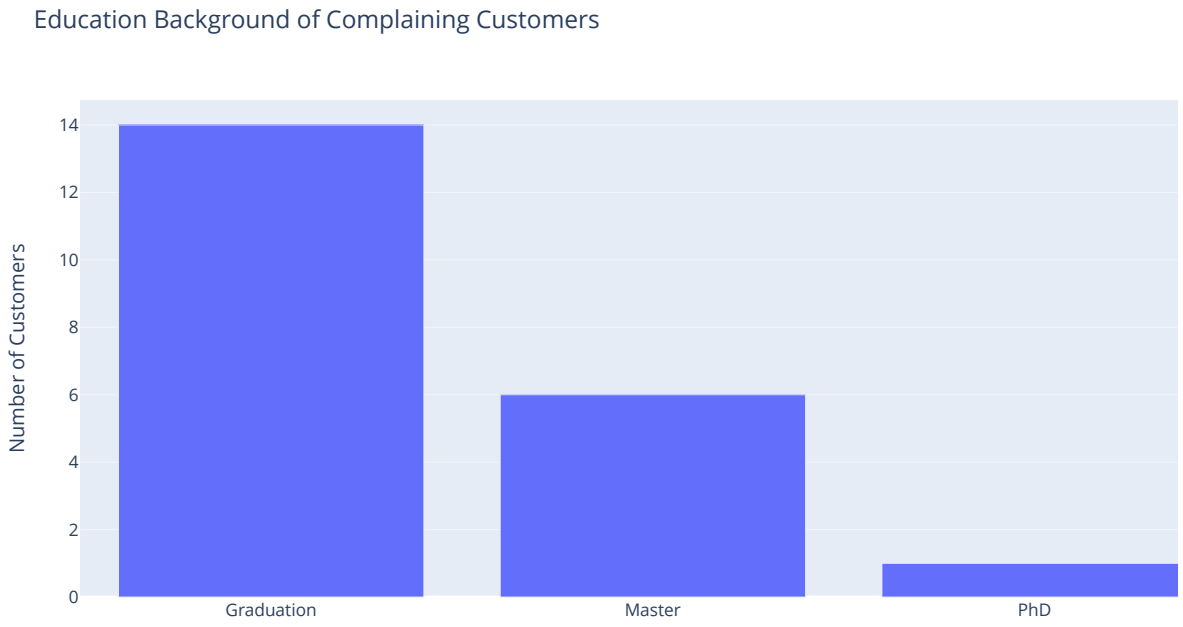
In [60]:
```
df['Education'].value_counts()
```

Out[60]:
```
Graduation    1127
Master         573
PhD            486
Basic           54
Name: Education, dtype: int64
```

In [61]:
```
df['Complain'].value_counts()
```

Out[61]:
```
0    2219
1      21
Name: Complain, dtype: int64
```

In [62]:
```
import plotly.express as px

# to get customers who complained in the last 2 years
complaining_customers = df[df['Complain'] == 1]

# ducation levels of complaining customers
education_counts = complaining_customers['Education'].value_counts().reset_index()

# bar plot
fig = px.bar(education_counts, x='index', y='Education', title='Education Background of Complaining Customers'
fig.update_xaxes(title='Education Level')
fig.update_yaxes(title='Number of Customers')
fig.show()
```




The bar plot indicates the education background of customers who complained in the last 2 years. The majority of complaining customers have a "Graduation" education level, followed by "Master" and "PhD." This suggests that customers with higher education levels are more likely to raise complaints in the last 2 years, while those with a "Basic" education level are less likely to complain.

---

End of the Analysis