STEP - 1 : Importing All The Libraries

In [1]:
```
import numpy as np
import pandas as pd
import json
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from scipy import stats
from scipy.stats.mstats import winsorize
from scipy.stats import ttest_ind
from scipy.stats import chi2_contingency
```

STEP - 2 : Data Quality and Check (Task 1)

1. Create a consolidated view of data by joining the data present in three files.

In [2]:
```
# Opening the text file.
with open (r'C:\Users\shikh\OneDrive\Desktop\Capstone Project\Retail_Sales_ABADS\demo
        columns = file.readline().strip().split('\t')
        data = []
        for line in file:
            row = line.strip().split('\t')
            data.append(row)
```

In [3]:
```
df = pd.DataFrame(data, columns=columns)
```

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

Out[4]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Country
0	1826	1970	Graduation	Divorced	$84,835.00	0	0	6/16/14	SP
1	1	1961	Graduation	Single	$57,091.00	0	0	6/15/14	CA
2	10476	1958	Graduation	Married	$67,267.00	0	1	5/13/14	US
3	1386	1967	Graduation	Together	$32,474.00	1	1	5/11/14	AUS
4	5371	1989	Graduation	Single	$21,474.00	1	0	4/8/14	SP

In [5]:
```
df.shape
```

Out[5]: (2240, 9)

In [6]:
```
# Opening behaviour json file
with open(r"C:\Users\shikh\OneDrive\Desktop\Capstone Project\Retail_Sales_ABADS\behav
    data = json.load(f)

# Initialize an empty list to store dictionaries
rows = []

# Iterate over each dictionary in the list
for item in data:
    # Extract the ID and attributes from the dictionary
    id_ = list(item.keys())[0]
    attributes = item[id_]

    # Create a dictionary with ID as a key and attributes
    row = {'ID': id_}
    row.update(attributes)
    rows.append(row)

# Create a DataFrame from the list of dictionaries
df1 = pd.DataFrame(rows)
```

In [7]:
```
df1.shape
```

Out[7]: (2240, 13)

In [8]:
```
df1.head()
```

Out[8]:

	ID	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweetProducts	Mnt(
0	ID_1826	0	189	104	379	111	189	
1	ID_1	0	464	5	64	7	0	
2	ID_10476	0	134	11	59	15	2	
3	ID_1386	0	10	0	1	0	0	
4	ID_5371	0	6	16	24	11	0	

In [9]:
```python
# Opening Campaign json file.
with open(r"C:\Users\shikh\OneDrive\Desktop\Capstone Project\Retail_Sales_ABADS\campa
    data = json.load(f)

# Initialize an empty list to store dictionaries
rows = []

# Iterate over each dictionary in the list
for item in data:
    # Extract the ID and attributes from the dictionary
    id_ = list(item.keys())[0]
    attributes = item[id_]

    # Create a dictionary with ID as a key and attributes
    row = {'ID': id_}
    row.update(attributes)
    rows.append(row)

# Create a DataFrame from the list of dictionaries
df2 = pd.DataFrame(rows)
```

In [10]:
```python
df2.shape
```

Out[10]: (2240, 8)

In [11]:
```python
df2.head()
```

Out[11]:

	ID	AcceptedCmp1	AcceptedCmp2	AcceptedCmp3	AcceptedCmp4	AcceptedCmp5	Response
0	ID_1826	0	0	0	0	0	1
1	ID_1	0	1	0	0	0	1
2	ID_10476	0	0	0	0	0	0
3	ID_1386	0	0	0	0	0	0
4	ID_5371	0	0	1	0	0	1

In [12]:
```python
# Merging df1 & df2 files on the basis of ID column.
df3 = pd.merge(df1, df2, on='ID')
```

In [13]:
```python
# Removing ID text and underscore from ID column
df3['ID'] = df3['ID'].str.replace('ID_', '')
```

In [14]:
```python
df3.head()
```

Out[14]:

	ID	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweetProducts	MntGol
0	1826	0	189	104	379	111	189	
1	1	0	464	5	64	7	0	
2	10476	0	134	11	59	15	2	
3	1386	0	10	0	1	0	0	
4	5371	0	6	16	24	11	0	

In [15]:
```python
# Merging df3 with text file df on ID column
final_df=pd.merge(df, df3, on='ID')
```

In [16]:
```python
final_df.columns
```

Out[16]:
```
Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', ' Income ',
       'Kidhome', 'Teenhome', 'Dt_Customer', 'Country', 'Recency', 'MntWines',
       'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
       'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
       'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth',
       'AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'AcceptedCmp4',
       'AcceptedCmp5', 'Response', 'Complain'],
      dtype='object')
```

2. Are there any variables where you will need to clean the raw data, what kind of cleaning will be needed?

In [17]:
```python
final_df.rename(columns = {' Income ':'Income'}, inplace = True)
```

In [18]:
```python
final_df.columns
```

Out[18]:
```
Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome',
       'Teenhome', 'Dt_Customer', 'Country', 'Recency', 'MntWines',
       'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
       'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
       'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth',
       'AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'AcceptedCmp4',
       'AcceptedCmp5', 'Response', 'Complain'],
      dtype='object')
```

In [19]:
```python
final_df['Income'] = final_df['Income'].str.replace('$', '')
```

In [20]:
```python
final_df['Income'] = final_df['Income'].str.replace(',', '')
```

In [21]:
```python
final_df.head()
```

Out[21]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Country
0	1826	1970	Graduation	Divorced	84835.00	0	0	6/16/14	SP
1	1	1961	Graduation	Single	57091.00	0	0	6/15/14	CA
2	10476	1958	Graduation	Married	67267.00	0	1	5/13/14	US
3	1386	1967	Graduation	Together	32474.00	1	1	5/11/14	AUS
4	5371	1989	Graduation	Single	21474.00	1	0	4/8/14	SP

5 rows × 28 columns

In [22]:
```python
final_df['Income'] = final_df['Income'].apply(lambda x: x.replace(',', '') if isinsta
final_df['Income'] = pd.to_numeric(final_df['Income'], errors='coerce')  # Coerce err
```

In [23]:
```
final_df.head()
```

Out[23]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Country	R
0	1826	1970	Graduation	Divorced	84835.0	0	0	6/16/14	SP	
1	1	1961	Graduation	Single	57091.0	0	0	6/15/14	CA	
2	10476	1958	Graduation	Married	67267.0	0	1	5/13/14	US	
3	1386	1967	Graduation	Together	32474.0	1	1	5/11/14	AUS	
4	5371	1989	Graduation	Single	21474.0	1	0	4/8/14	SP	

5 rows × 28 columns

Fixing The Response Column

In [24]:
```
# Filter rows where AcceptedCmp1 to 5 are all 0 and Response is 1
filtered_rows = final_df[(final_df['AcceptedCmp1'] == 0) &
                         (final_df['AcceptedCmp2'] == 0) &
                         (final_df['AcceptedCmp3'] == 0) &
                         (final_df['AcceptedCmp4'] == 0) &
                         (final_df['AcceptedCmp5'] == 0) &
                         (final_df['Response'] == 1)]

# Count the number of rows
num_rows = len(filtered_rows)

print("\033[1;38;5;208mNumber of rows where AcceptedCmp1 to 5 are 0 and Response is 1
```

```
Number of rows where AcceptedCmp1 to 5 are 0 and Response is 1: 146
```

In [25]:
```
final_df.loc[(final_df['AcceptedCmp1'] == 0) &
             (final_df['AcceptedCmp2'] == 0) &
             (final_df['AcceptedCmp3'] == 0) &
             (final_df['AcceptedCmp4'] == 0) &
             (final_df['AcceptedCmp5'] == 0) &
             (final_df['Response'] == 1), 'Response'] = 0
```

In [26]:
```
# Filter rows where any one column out of AcceptedCmp1 to 5 is 1 and Response is 0
filtered_rows = final_df[((final_df['AcceptedCmp1'] == 1) |
                          (final_df['AcceptedCmp2'] == 1) |
                          (final_df['AcceptedCmp3'] == 1) |
                          (final_df['AcceptedCmp4'] == 1) |
                          (final_df['AcceptedCmp5'] == 1)) &
                         (final_df['Response'] == 0)]

# Count the number of rows
num_rows = len(filtered_rows)

print("\033[38;5;208m\033[1mNumber of rows where any one column out of AcceptedCmp1 t
```

```
Number of rows where any one column out of AcceptedCmp1 to 5 is 1 and Response is 0:
275
```

In [27]:
```
final_df.loc[((final_df['AcceptedCmp1'] == 1) |
              (final_df['AcceptedCmp2'] == 1) |
              (final_df['AcceptedCmp3'] == 1) |
              (final_df['AcceptedCmp4'] == 1) |
              (final_df['AcceptedCmp5'] == 1)) &
             (final_df['Response'] == 0), 'Response'] = 1
```

In [28]:
```
final_df.head()
```

Out[28]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Country	R
0	1826	1970	Graduation	Divorced	84835.0	0	0	6/16/14	SP	
1	1	1961	Graduation	Single	57091.0	0	0	6/15/14	CA	
2	10476	1958	Graduation	Married	67267.0	0	1	5/13/14	US	
3	1386	1967	Graduation	Together	32474.0	1	1	5/11/14	AUS	
4	5371	1989	Graduation	Single	21474.0	1	0	4/8/14	SP	

5 rows × 28 columns

Adding the Age Column

In [29]:
```
final_df['Year_Birth'] = final_df['Year_Birth'].astype(int)

# Calculate current year
current_year = 2024

# Calculate age
final_df['Age'] = current_year - final_df['Year_Birth']

# Display the DataFrame with the new 'Age' column
final_df.head()
```

Out[29]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Country	R
0	1826	1970	Graduation	Divorced	84835.0	0	0	6/16/14	SP	
1	1	1961	Graduation	Single	57091.0	0	0	6/15/14	CA	
2	10476	1958	Graduation	Married	67267.0	0	1	5/13/14	US	
3	1386	1967	Graduation	Together	32474.0	1	1	5/11/14	AUS	
4	5371	1989	Graduation	Single	21474.0	1	0	4/8/14	SP	

5 rows × 29 columns

Adding the total amount spent columns

In [30]:
```
# Calculating total amount spent.
# Select the specified columns
spending_columns = ['MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'M

# Calculate the total amount spent and assign it to a new column 'Total_Amount_Spent'
final_df['Total_Amount_Spent'] = final_df[spending_columns].sum(axis=1)

# Display the DataFrame with the new column
final_df.head()
```

Out[30]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Country	R
0	1826	1970	Graduation	Divorced	84835.0	0	0	6/16/14	SP	
1	1	1961	Graduation	Single	57091.0	0	0	6/15/14	CA	
2	10476	1958	Graduation	Married	67267.0	0	1	5/13/14	US	
3	1386	1967	Graduation	Together	32474.0	1	1	5/11/14	AUS	
4	5371	1989	Graduation	Single	21474.0	1	0	4/8/14	SP	

5 rows × 30 columns

Adding the total purchases made columns

In [31]:
```
## Finding out total purchases
purchase_columns = ['NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases', 'N

# Calculate the total of these columns and assign it to a new column 'Total_Purchases
final_df['Total_Purchases'] = final_df[purchase_columns].sum(axis=1)

# Display the DataFrame with the new column
final_df.head()
```

Out[31]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Country	R
0	1826	1970	Graduation	Divorced	84835.0	0	0	6/16/14	SP	
1	1	1961	Graduation	Single	57091.0	0	0	6/15/14	CA	
2	10476	1958	Graduation	Married	67267.0	0	1	5/13/14	US	
3	1386	1967	Graduation	Together	32474.0	1	1	5/11/14	AUS	
4	5371	1989	Graduation	Single	21474.0	1	0	4/8/14	SP	

5 rows × 31 columns

In [32]:
```
final_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   object 
 1   Year_Birth           2240 non-null   int32  
 2   Education            2240 non-null   object 
 3   Marital_Status       2240 non-null   object 
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   object 
 6   Teenhome             2240 non-null   object 
 7   Dt_Customer          2240 non-null   object 
 8   Country              2240 non-null   object 
 9   Recency              2240 non-null   int64  
 10  MntWines             2240 non-null   int64  
 11  MntFruits            2240 non-null   int64  
 12  MntMeatProducts      2240 non-null   int64  
 13  MntFishProducts      2240 non-null   int64  
 14  MntSweetProducts     2240 non-null   int64  
 15  MntGoldProds         2240 non-null   int64  
 16  NumDealsPurchases    2240 non-null   int64  
 17  NumWebPurchases      2240 non-null   int64  
 18  NumCatalogPurchases  2240 non-null   int64  
 19  NumStorePurchases    2240 non-null   int64  
 20  NumWebVisitsMonth    2240 non-null   int64  
 21  AcceptedCmp1         2240 non-null   int64  
 22  AcceptedCmp2         2240 non-null   int64  
 23  AcceptedCmp3         2240 non-null   int64  
 24  AcceptedCmp4         2240 non-null   int64  
 25  AcceptedCmp5         2240 non-null   int64  
 26  Response             2240 non-null   int64  
 27  Complain             2240 non-null   int64  
 28  Age                  2240 non-null   int32  
 29  Total_Amount_Spent   2240 non-null   int64  
 30  Total_Purchases      2240 non-null   int64  
dtypes: float64(1), int32(2), int64(21), object(7)
memory usage: 525.1+ KB
```

3.1. Doing univariates for continuous variables (compute: percentage of missing values, percentage of terms which are zero, mean, 25th, 50th, 75th, 90th and 95th percentile, min and max)

In [33]:
```
final_df.describe()
```

Out[33]:

	Year_Birth	Income	Recency	MntWines	MntFruits	MntMeatProducts	MntFishPro
count	2240.000000	2216.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.0
mean	1968.805804	52247.251354	49.109375	303.935714	26.302232	166.950000	37.5
std	11.984069	25173.076661	28.962453	336.597393	39.773434	225.715373	54.6
min	1893.000000	1730.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	1959.000000	35303.000000	24.000000	23.750000	1.000000	16.000000	3.0
50%	1970.000000	51381.500000	49.000000	173.500000	8.000000	67.000000	12.0
75%	1977.000000	68522.000000	74.000000	504.250000	33.000000	232.000000	50.0
max	1996.000000	666666.000000	99.000000	1493.000000	199.000000	1725.000000	259.0

8 rows × 24 columns

In [34]:
```
missing_percentage = final_df.isnull().mean() * 100

# Compute the percentage of zero values for each column
zero_percentage = (final_df == 0).mean() * 100

# Compute descriptive statistics
description = final_df.describe(percentiles=[.25, .50, .75, .90, .95])

# Add missing and zero percentages to the description DataFrame
description.loc['missing_percentage'] = missing_percentage
description.loc['zero_percentage'] = zero_percentage
```

In [35]:
```
# Transpose the DataFrame for better visualization
description = description.T

# Convert the DataFrame to a new DataFrame
summary_df = pd.DataFrame({
    'mean': description['mean'],
    'std': description['std'],
    'min': description['min'],
    '25th percentile': description['25%'],
    'median': description['50%'],
    '75th percentile': description['75%'],
    '90th percentile': description['90%'],
    '95th percentile': description['95%'],
    'max': description['max'],
    'missing_percentage': description['missing_percentage'],
    'zero_percentage': description['zero_percentage']
})
```

In [36]:
```python
summary_df.head(31)
```

Out[36]:

	mean	std	min	25th percentile	median	75th percentile	90th percentile	pe
Year_Birth	1968.805804	11.984069	1893.0	1959.00	1970.0	1977.00	1984.0	
Income	52247.251354	25173.076661	1730.0	35303.00	51381.5	68522.00	79844.0	84
Recency	49.109375	28.962453	0.0	24.00	49.0	74.00	89.0	
MntWines	303.935714	336.597393	0.0	23.75	173.5	504.25	822.1	
MntFruits	26.302232	39.773434	0.0	1.00	8.0	33.00	83.0	
MntMeatProducts	166.950000	225.715373	0.0	16.00	67.0	232.00	499.0	
MntFishProducts	37.525446	54.628979	0.0	3.00	12.0	50.00	120.0	
MntSweetProducts	27.062946	41.280498	0.0	1.00	8.0	33.00	89.0	
MntGoldProds	44.021875	52.167439	0.0	9.00	24.0	56.00	122.0	
NumDealsPurchases	2.325000	1.932238	0.0	1.00	2.0	3.00	5.0	
NumWebPurchases	4.084821	2.778714	0.0	2.00	4.0	6.00	8.0	
NumCatalogPurchases	2.662054	2.923101	0.0	0.00	2.0	4.00	7.0	
NumStorePurchases	5.790179	3.250958	0.0	3.00	5.0	8.00	11.0	
NumWebVisitsMonth	5.316518	2.426645	0.0	3.00	6.0	7.00	8.0	
AcceptedCmp1	0.064286	0.245316	0.0	0.00	0.0	0.00	0.0	
AcceptedCmp2	0.013393	0.114976	0.0	0.00	0.0	0.00	0.0	
AcceptedCmp3	0.072768	0.259813	0.0	0.00	0.0	0.00	0.0	
AcceptedCmp4	0.074554	0.262728	0.0	0.00	0.0	0.00	0.0	
AcceptedCmp5	0.072768	0.259813	0.0	0.00	0.0	0.00	0.0	
Response	0.206696	0.405026	0.0	0.00	0.0	0.00	1.0	
Complain	0.009375	0.096391	0.0	0.00	0.0	0.00	0.0	
Age	55.194196	11.984069	28.0	47.00	54.0	65.00	72.0	
Total_Amount_Spent	605.798214	602.249288	5.0	68.75	396.0	1045.50	1536.2	
Total_Purchases	14.862054	7.677173	0.0	8.00	15.0	21.00	25.0	

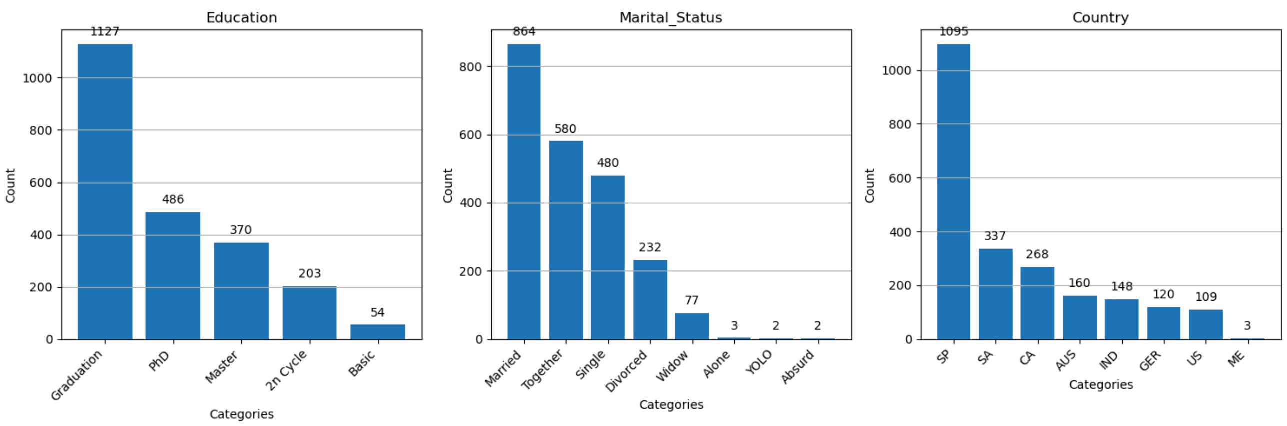
3.2. Doing univariates for categorical variables (compute:percentage of missing values, number of unique values)

In [37]:
```python
# Calculate percentage of missing values and number of unique values for each categor
categorical_variables = ['Education', 'Marital_Status', 'Country','Kidhome','Teenhome

missing_values_percentage = {}
unique_values_count = {}

for var in categorical_variables:
    missing_values_percentage[var] = (final_df[var].isnull().mean()) * 100
    unique_values_count[var] = final_df[var].nunique()

# Create a DataFrame to store the results
results_df = pd.DataFrame({'Missing Values (%)': missing_values_percentage,
                           'Unique Values Count': unique_values_count})

# Display the results
print("Univariate Analysis for Categorical Variables:")
print(results_df)
```

```
Univariate Analysis for Categorical Variables:
                Missing Values (%)  Unique Values Count
Education                      0.0                    5
Marital_Status                 0.0                    8
Country                        0.0                    8
Kidhome                        0.0                    3
Teenhome                       0.0                    3
```

In [38]:
```python
# List of categorical variables
categorical_variables = ['Education', 'Marital_Status', 'Country']  # Example list of

# Create subplots for each categorical variable
fig, axes = plt.subplots(nrows=1, ncols=len(categorical_variables), figsize=(15, 5))

for i, var in enumerate(categorical_variables):
    # Count the frequency of unique values
    value_counts = final_df[var].value_counts()

    # Plot bar chart
    ax = axes[i]
    ax.bar(value_counts.index, value_counts)
    ax.set_title(var)
    ax.set_xlabel('Categories')
    ax.set_ylabel('Count')
    ax.grid(axis='y')

    # Rotate category names
    ax.set_xticklabels(value_counts.index, rotation=45, ha='right')  # Adjust rotatic

    # Add annotations
    for bar in ax.patches:
        ax.annotate(f'{bar.get_height()}',
                    (bar.get_x() + bar.get_width() / 2, bar.get_height()),
                    ha='center', va='bottom', xytext=(0, 5),
                    textcoords='offset points')

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

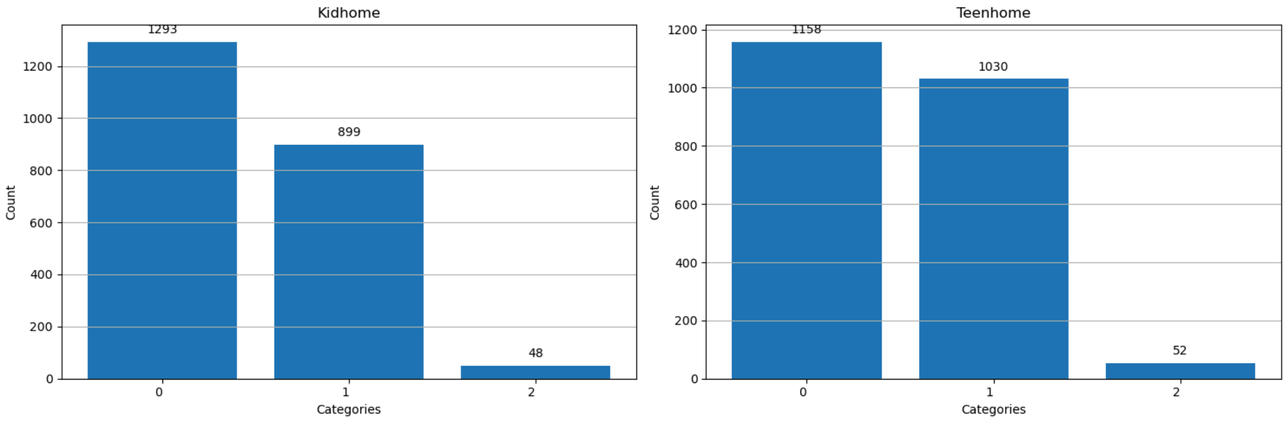
```
C:\Users\shikh\AppData\Local\Temp\ipykernel_2336\3951115988.py:20: UserWarning: Fixe
dFormatter should only be used together with FixedLocator
  ax.set_xticklabels(value_counts.index, rotation=45, ha='right')  # Adjust rotation
angle as needed
C:\Users\shikh\AppData\Local\Temp\ipykernel_2336\3951115988.py:20: UserWarning: Fixe
dFormatter should only be used together with FixedLocator
  ax.set_xticklabels(value_counts.index, rotation=45, ha='right')  # Adjust rotation
angle as needed
C:\Users\shikh\AppData\Local\Temp\ipykernel_2336\3951115988.py:20: UserWarning: Fixe
dFormatter should only be used together with FixedLocator
  ax.set_xticklabels(value_counts.index, rotation=45, ha='right')  # Adjust rotation
angle as needed
```



In [39]:

```
# List of categorical variables
categorical_variables = ['Kidhome', 'Teenhome']  # Example list of categorical variab

# Create subplots for each categorical variable
fig, axes = plt.subplots(nrows=1, ncols=len(categorical_variables), figsize=(15, 5))

for i, var in enumerate(categorical_variables):
    # Count the frequency of unique values
    value_counts = final_df[var].value_counts()

    # Plot bar chart
    ax = axes[i]
    ax.bar(value_counts.index, value_counts)
    ax.set_title(var)
    ax.set_xlabel('Categories')
    ax.set_ylabel('Count')
    ax.grid(axis='y')

    # Rotate category names
    ax.set_xticklabels(value_counts.index, ha='right')  # Adjust rotation angle as ne

    # Add annotations
    for bar in ax.patches:
        ax.annotate(f'{bar.get_height()}',
                    (bar.get_x() + bar.get_width() / 2, bar.get_height()),
                    ha='center', va='bottom', xytext=(0, 5),
                    textcoords='offset points')

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

```
C:\Users\shikh\AppData\Local\Temp\ipykernel_2336\2547910356.py:20: UserWarning: Fixe
dFormatter should only be used together with FixedLocator
  ax.set_xticklabels(value_counts.index, ha='right')  # Adjust rotation angle as nee
ded
C:\Users\shikh\AppData\Local\Temp\ipykernel_2336\2547910356.py:20: UserWarning: Fixe
dFormatter should only be used together with FixedLocator
  ax.set_xticklabels(value_counts.index, ha='right')  # Adjust rotation angle as nee
ded
```

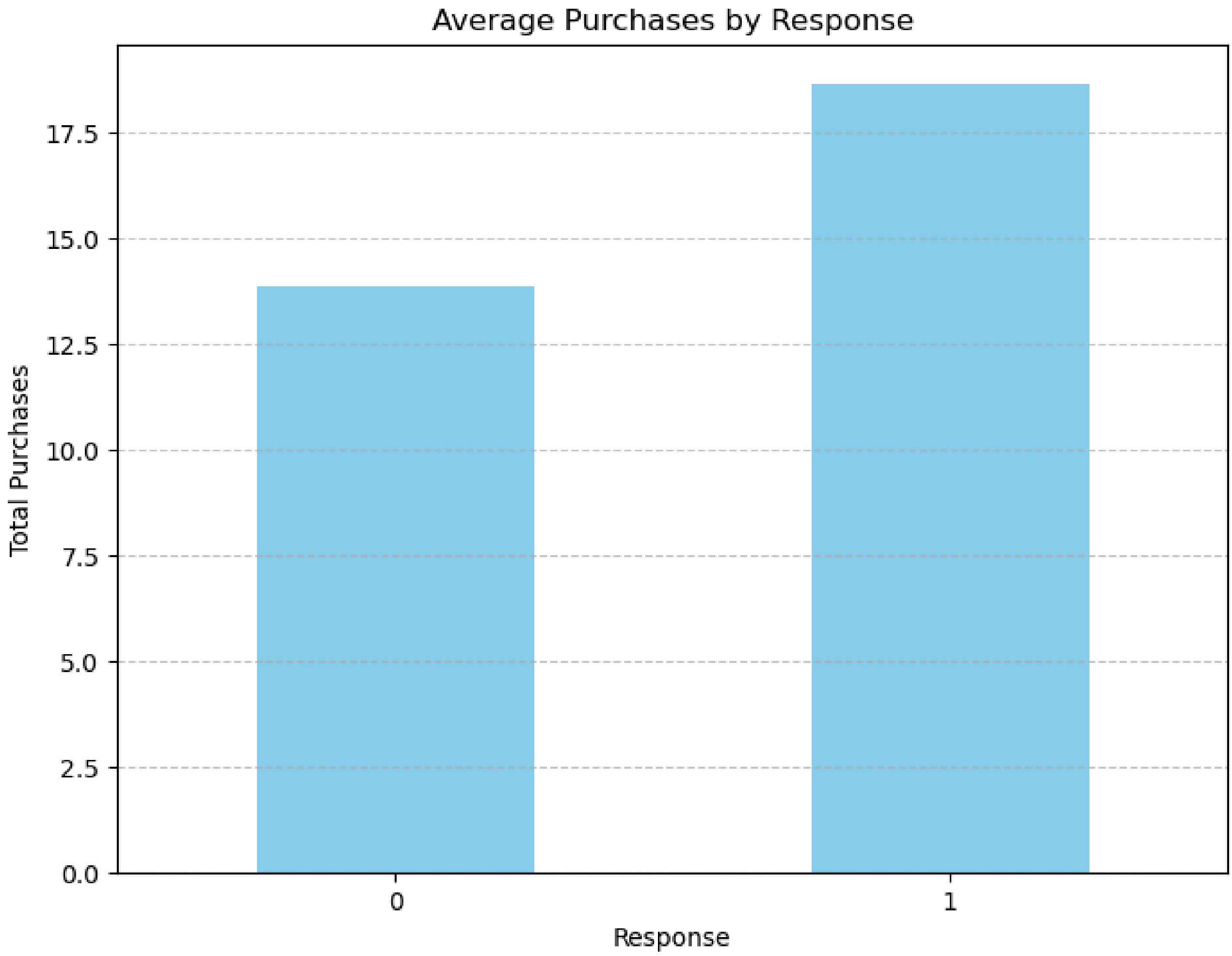


Understanding the distribution of numerical columns with respect to Response

Income v/s Total Purchases

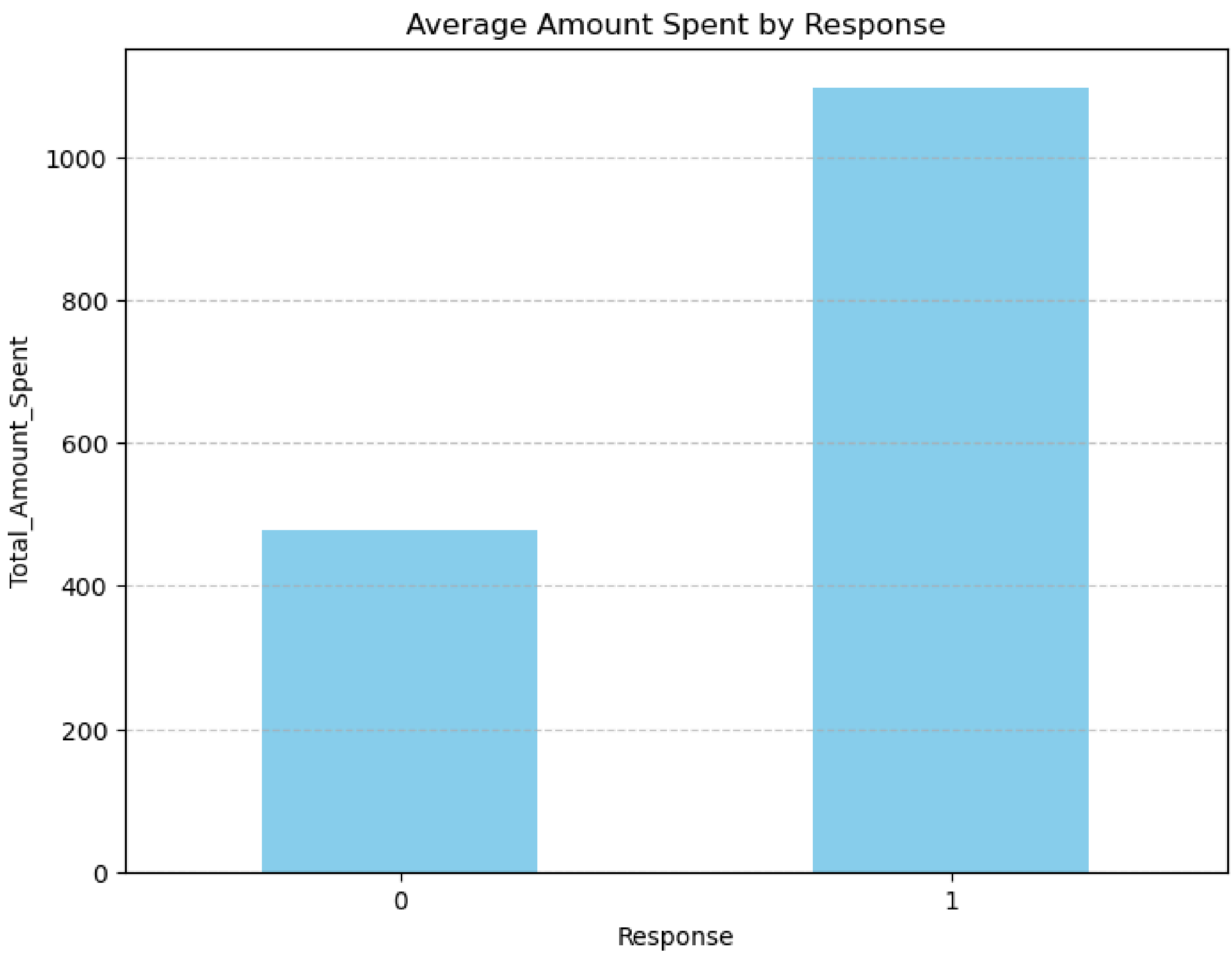
In [65]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=final_df, x='Income', y='Total_Purchases', hue='Response')
plt.title('Scatter Plot of Income vs Total Purchases with Response')
plt.xlabel('Income')
plt.ylabel('Total Purchases')
plt.legend(title='Response')
plt.grid(True)
plt.show()
```

In [40]:
```python
# Calculate the sum total purchases for each response category
sum_total_purchases = final_df.groupby('Response')['Total_Purchases'].mean()

# Plot the bar plot
plt.figure(figsize=(8, 6))
sum_total_purchases.plot(kind='bar', color='skyblue')
plt.title('Average Purchases by Response')
plt.xlabel('Response')
plt.ylabel('Total Purchases')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



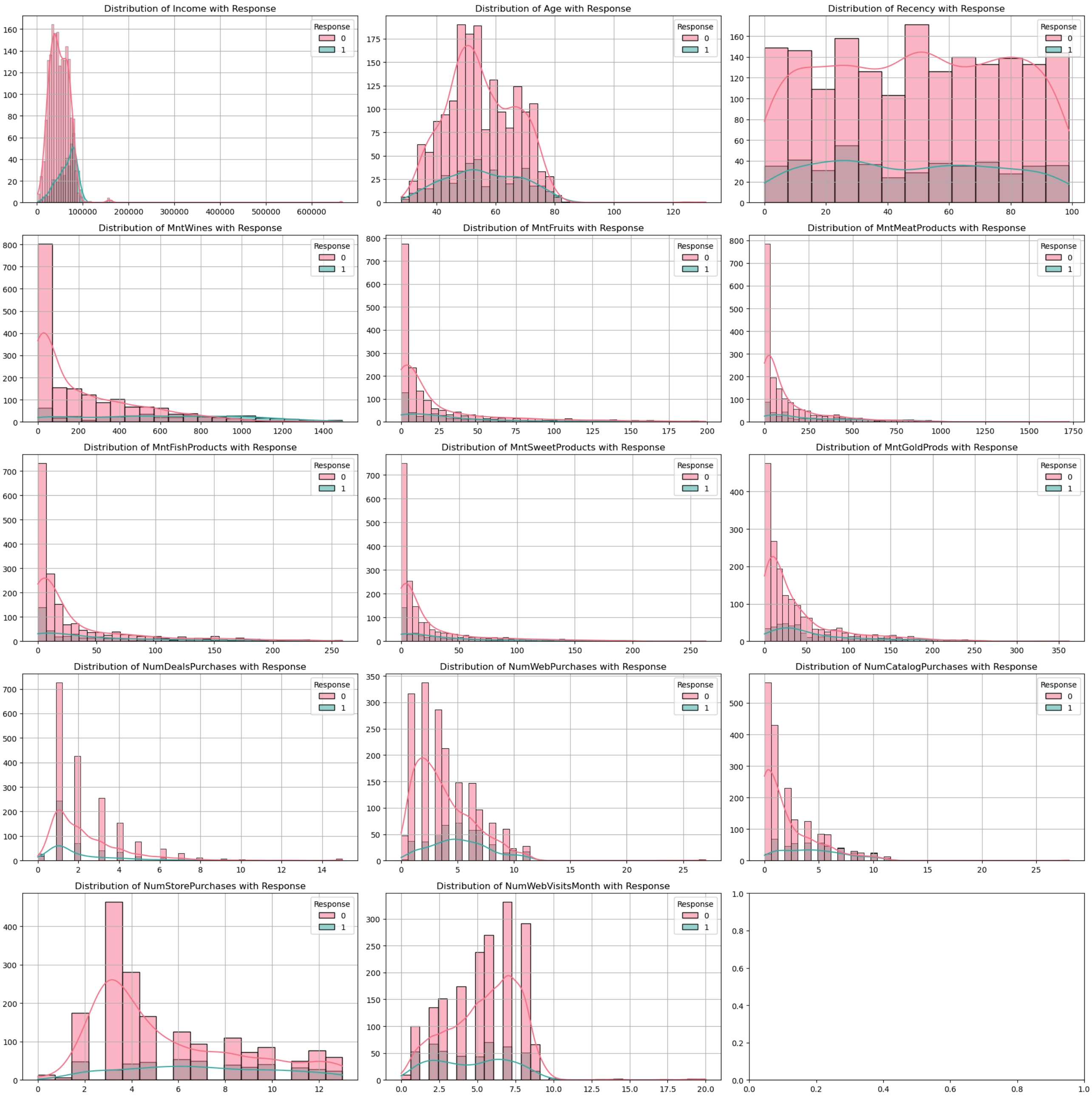
In [41]:

```python
# Calculate the sum total purchases for each response category
sum_total_purchases = final_df.groupby('Response')['Total_Amount_Spent'].mean()

# Plot the bar plot
plt.figure(figsize=(8, 6))
sum_total_purchases.plot(kind='bar', color='skyblue')
plt.title('Average Amount Spent by Response')
plt.xlabel('Response')
plt.ylabel('Total_Amount_Spent')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



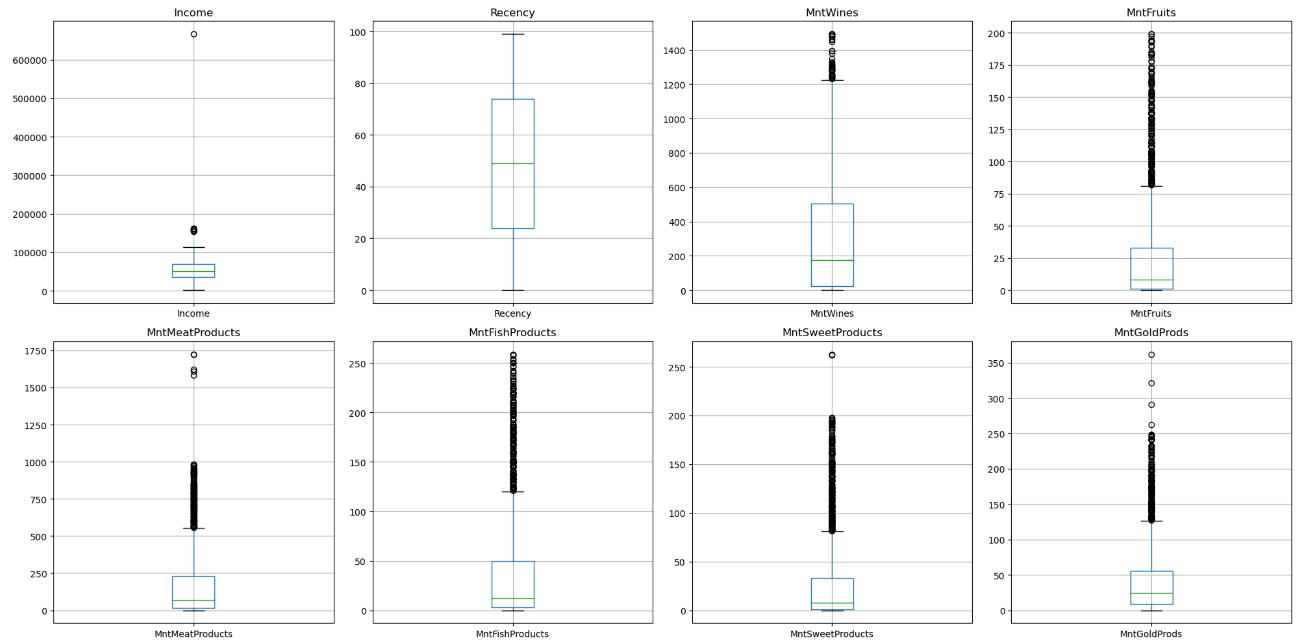
In [42]:

```
# Define the numerical columns
numerical_columns = ['Income','Age', 'Recency', 'MntWines', 'MntFruits', 'MntMeatProd
                     'MntFishProducts', 'MntSweetProducts', 'MntGoldProds',
                     'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases',
                     'NumStorePurchases', 'NumWebVisitsMonth']

# Calculate the number of rows and columns for the subplots
num_rows = 5
num_cols = 3

# Set the height of each subplot
subplot_height = 4  # Adjust this value as needed

# Calculate the total height of the figure
total_height = subplot_height * num_rows

# Create subplots with reduced height
fig, axs = plt.subplots(num_rows, num_cols, figsize=(20, total_height))

# Flatten the axs array for easier iteration
axs = axs.flatten()

# Loop through numerical columns and create a histogram for each
for i, col in enumerate(numerical_columns):
    ax = axs[i]
    sns.histplot(final_df, x=col, hue='Response', ax=ax, kde=True, palette='husl')
    ax.set_title(f'Distribution of {col} with Response')
    ax.set_xlabel('')
    ax.set_ylabel('')
    ax.grid(True)

# Adjust layout
plt.tight_layout()
```



4. Are there any extreme values of variables representing income, amount of money spent on various categories, recency of purchase?

In [43]:
```
# Select specific columns for box plot
columns_of_interest = ['Income', 'Recency', 'MntWines', 'MntFruits', 'MntMeatProducts
                       'MntFishProducts', 'MntSweetProducts', 'MntGoldProds']

# Plotting box plots
fig, axes = plt.subplots(2, 4, figsize=(20, 10))
for i, col in enumerate(columns_of_interest):
    ax = axes[i // 4, i % 4]
    final_df.boxplot(column=col, ax=ax)
    ax.set_title(col)

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



STEP - 3 : Business Analysis and Hypothesis (Task 2)

5. Generate and check hypothesis around Amount Spent on different categories and response rate in different marketing campaigns.

Formulating Hypotheses:
--> Null Hypothesis (H0): There is no significant difference in the mean amount spent on different categories between customers who responded positively (response = 1) and those who did not respond (response = 0) across different marketing campaigns.

Alternative Hypothesis (H1): There is a significant difference in the mean amount spent on different categories between customers who responded positively and those who did not respond across different marketing campaigns.

In [44]:
```python
# Define the categories of amount spent
categories = ['MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSwee

# Initialize lists to store results
results = []
categories_col = []
t_statistic_col = []
p_value_col = []

# Iterate over each category
for category in categories:
    # Perform t-test for the category between response groups
    response_0_data = final_df[final_df['Response'] == 0][category]
    response_1_data = final_df[final_df['Response'] == 1][category]

    t_statistic, p_value = stats.ttest_ind(response_0_data, response_1_data)

    # Append results to lists
    categories_col.append(category)
    t_statistic_col.append(t_statistic)
    p_value_col.append(p_value)

# Create DataFrame to store results
results_df = pd.DataFrame({
    'Category': categories_col,
    'T-Statistic': t_statistic_col,
    'P-Value': p_value_col
})

# Print results
print(results_df)
```

```
           Category  T-Statistic        P-Value
0          MntWines   -24.928932  2.956191e-121
1         MntFruits    -6.322610   3.095000e-10
2   MntMeatProducts   -13.778360   1.620541e-41
3   MntFishProducts    -7.872955   5.351005e-15
4  MntSweetProducts    -7.687954   2.222422e-14
5      MntGoldProds    -9.154464   1.196196e-19
```

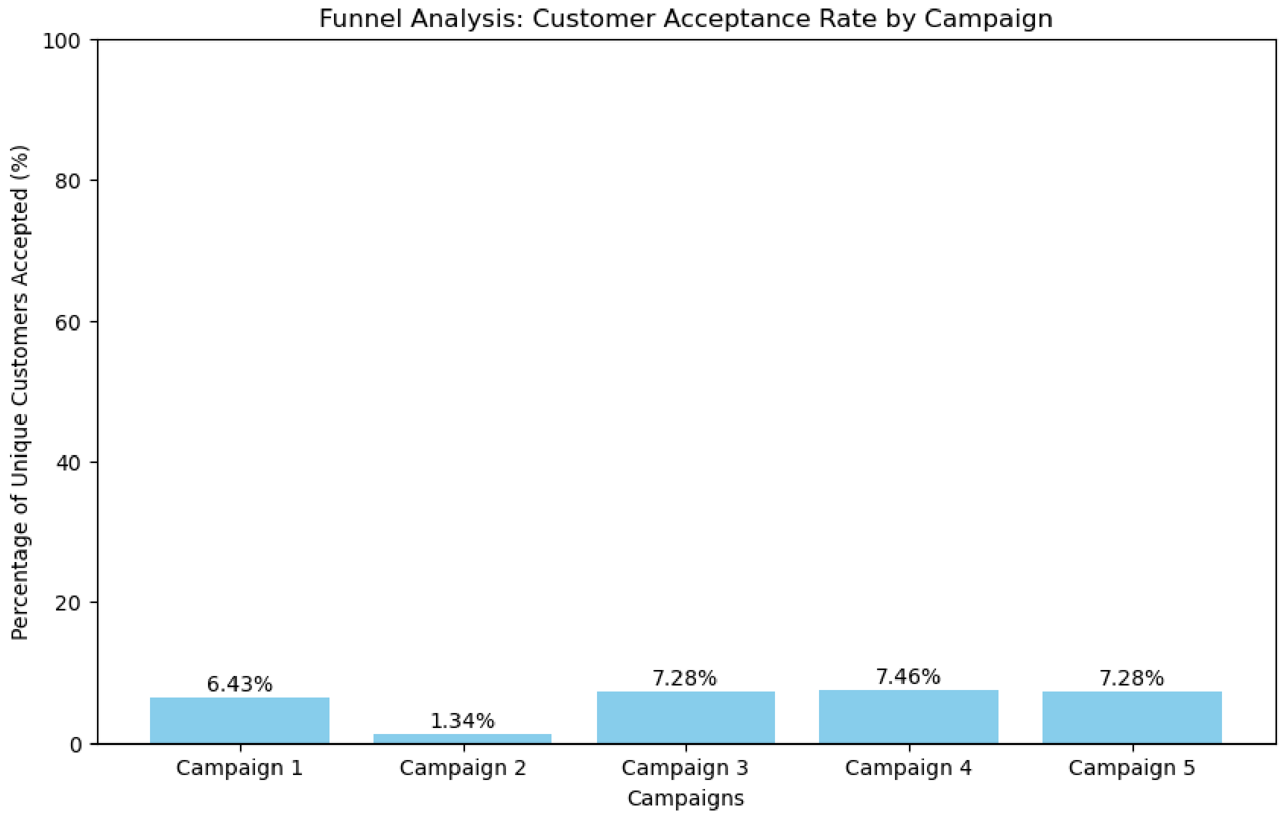
Interpreting The Results From Above Table

The p-values for all categories are significantly smaller than the chosen significance level (e.g., 0.05),

Therefore we conclude that alternate hypothesis is true and there is a significant difference in the mean amount spent on different categories between customers who responded positively and those who did not respond across different marketing campaigns.

6. Create a funnel analysis showing what percentage of unique customers accept campaign 1,2, 3,..etc

In [45]:
```
campaign_acceptance_counts = []
total_customers = len(final_df['ID'].unique())

for i in range(1, 6):
    campaign_acceptance_counts.append(len(final_df[final_df[f'AcceptedCmp{i}'] == 1][

# Calculate the percentage of unique customers who accepted each campaign
campaign_acceptance_percentages = [count / total_customers * 100 for count in campaig

# Campaign names
campaigns = ['Campaign 1', 'Campaign 2', 'Campaign 3', 'Campaign 4', 'Campaign 5']

# Plot the funnel analysis
plt.figure(figsize=(10, 6))
plt.bar(campaigns, campaign_acceptance_percentages, color='skyblue')
plt.xlabel('Campaigns')
plt.ylabel('Percentage of Unique Customers Accepted (%)')
plt.title('Funnel Analysis: Customer Acceptance Rate by Campaign')
plt.ylim(0, 100)

# Annotate the bars with their values
for i, percentage in enumerate(campaign_acceptance_percentages):
    plt.text(i, percentage + 1, f'{percentage:.2f}%', ha='center')

plt.show()
```



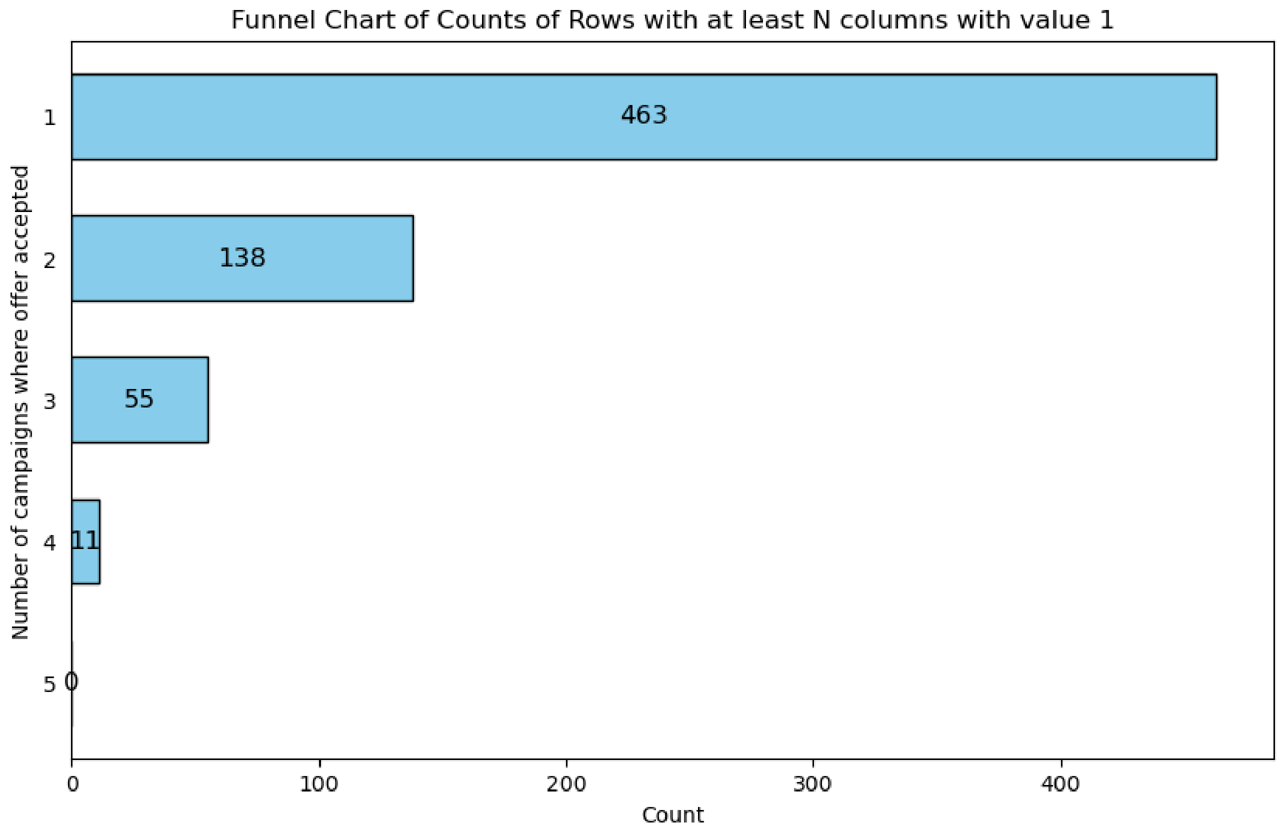
In [46]:
```
binary_columns = final_df[['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'AcceptedC
```

In [47]:
```python
# Create an empty DataFrame to store the summary
summary_table = pd.DataFrame()

# Iterate over each column and calculate value counts
for column in binary_columns.columns:
    value_counts = binary_columns[column].value_counts()
    summary_table[column] = value_counts

# Transpose the summary table for better readability
summary_table = summary_table.T

# Rename the index for better clarity
summary_table.index.name = 'Column'

# Rename the columns for better clarity
summary_table.columns = ['0', '1']

# Print the summary table
print(summary_table)
```

```
                 0    1
Column
AcceptedCmp1  2096  144
AcceptedCmp2  2210   30
AcceptedCmp3  2077  163
AcceptedCmp4  2073  167
AcceptedCmp5  2077  163
Response      1777  463
Complain      2219   21
```

In [48]:

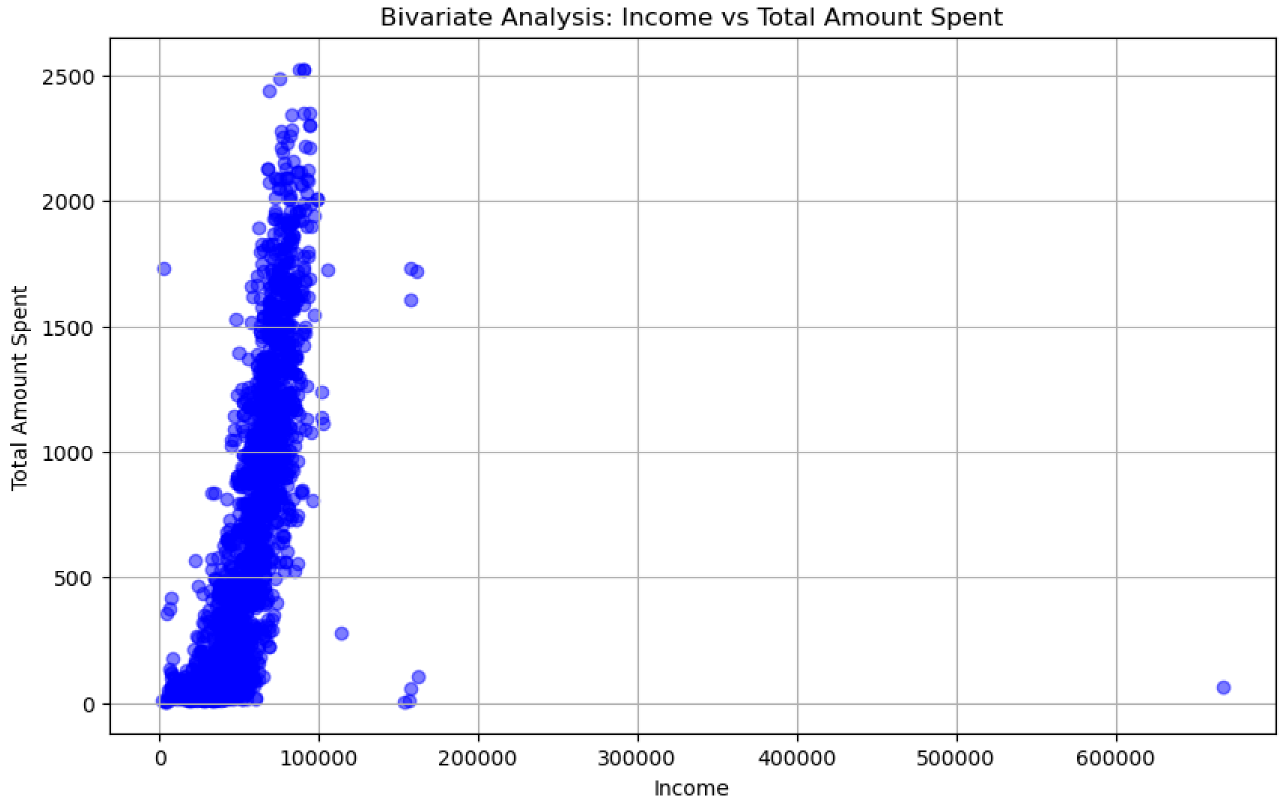
```
# Select the specified columns
binary_columns = final_df[['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'AcceptedC

# Define the thresholds
thresholds = [1, 2, 3, 4, 5]

# Create a dictionary to store the counts for each threshold
counts = {}

# Iterate over each threshold
for threshold in thresholds:
    # Count the number of rows where the sum of values is greater than or equal to th
    count = (binary_columns.sum(axis=1) >= threshold).sum()
    counts[threshold] = count

# Create a DataFrame from the counts dictionary
counts_df = pd.DataFrame.from_dict(counts, orient='index', columns=['Count'])

# Reverse the order of thresholds and counts
thresholds.reverse()
counts_df = counts_df.reindex(thresholds)

# Plot the counts as a funnel chart with larger bars at the top
plt.figure(figsize=(10, 6))  # Adjust figure size if needed

# Calculate the width of each section of the funnel
total_count = counts_df['Count'].sum()
section_widths = [counts_df['Count'][threshold] / total_count for threshold in thresh

# Plot each section of the funnel
for i, threshold in enumerate(thresholds):
    plt.barh([i], counts_df.loc[threshold, 'Count'], color='skyblue', height=0.6, edg

# Add annotations to the bars
for i, threshold in enumerate(thresholds):
    plt.text(counts_df.loc[threshold, 'Count'] / 2, i, str(int(counts_df.loc[threshol
             ha='center', va='center', color='black', fontsize=12)

# Add title and labels
plt.title('Funnel Chart of Counts of Rows with at least N columns with value 1')
plt.xlabel('Count')
plt.ylabel('Number of campaigns where offer accepted')
plt.yticks(range(len(thresholds)), thresholds)

# Remove y-axis ticks
plt.tick_params(axis='y', which='both', left=False)

# Show plot
plt.show()
```

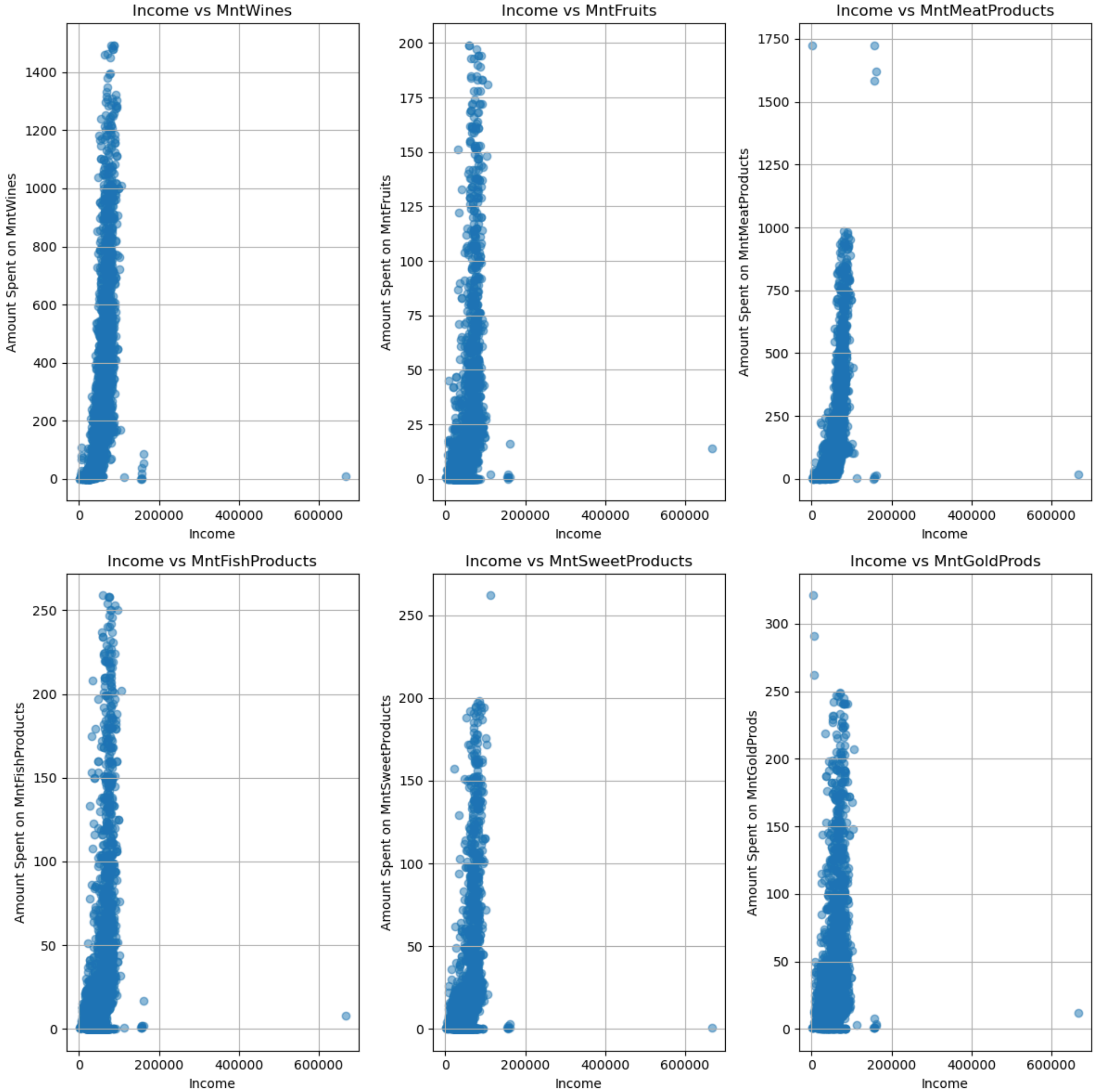


7. Find out how income impacts the amount spent on - Wine - Meat Products - Gold Products - Fish Products

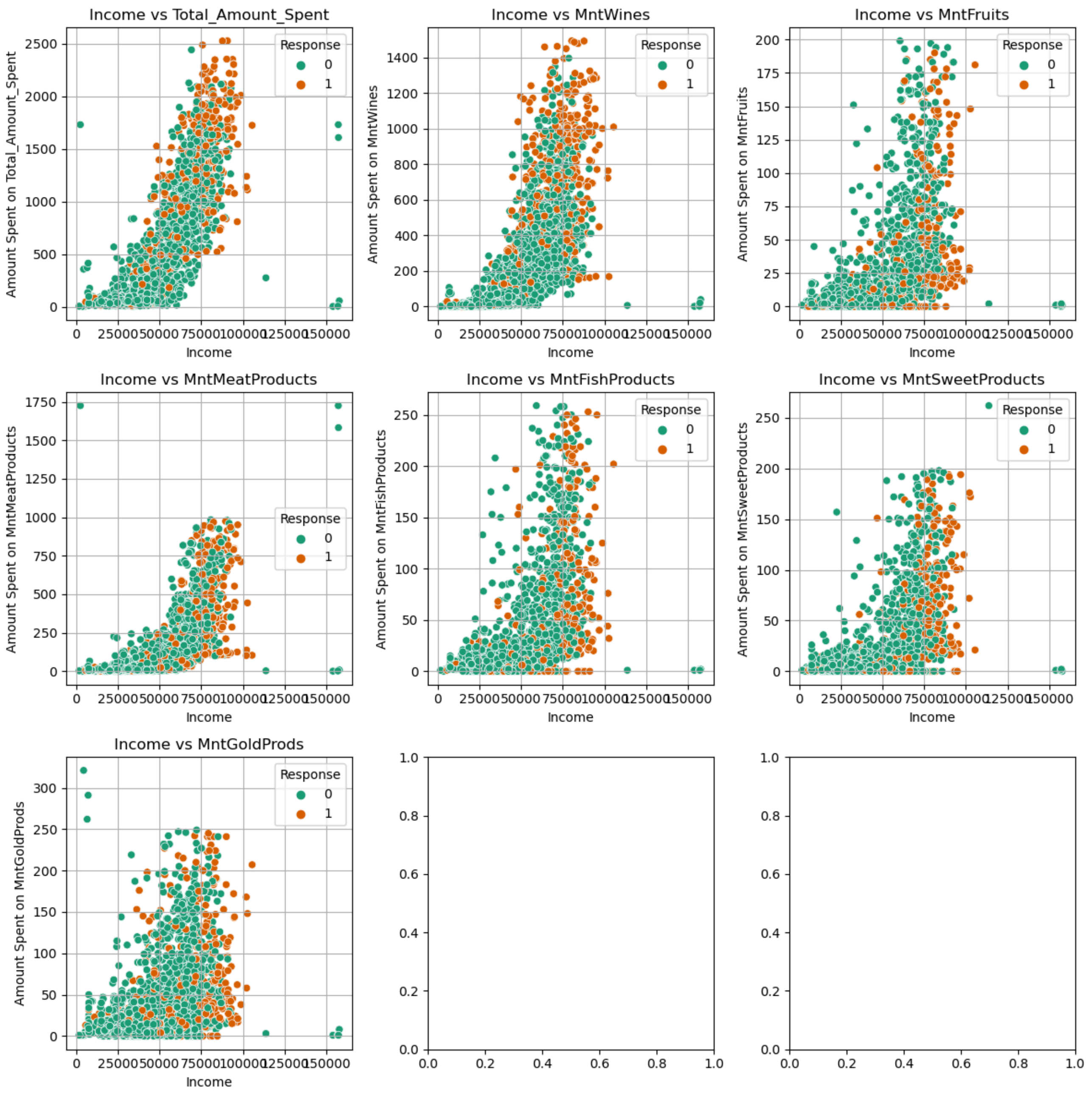
In [49]:

```python
plt.figure(figsize=(10, 6))
plt.scatter(final_df['Income'], final_df['Total_Amount_Spent'], alpha=0.5, color='blu
plt.title('Bivariate Analysis: Income vs Total Amount Spent')
plt.xlabel('Income')
plt.ylabel('Total Amount Spent')
plt.grid(True)
plt.show()

# Define the columns to be analyzed
columns_to_analyze = ['Income', 'MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishP

# Create a figure and subplots
fig, axs = plt.subplots(2, 3, figsize=(12, 12))  # 3 rows, 2 columns

# Flatten the axes array for easier iteration
axs = axs.flatten()

# Iterate over each product column (excluding 'Income')
for i, column in enumerate(columns_to_analyze[1:], start=1):
    # Scatter plot
    axs[i-1].scatter(final_df['Income'], final_df[column], alpha=0.5)
    axs[i-1].set_title(f'Income vs {column}')
    axs[i-1].set_xlabel('Income')
    axs[i-1].set_ylabel(f'Amount Spent on {column}')
    axs[i-1].grid(True)

# Adjust layout to prevent overlap
plt.tight_layout()

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




Income vs MntWines
Amount Spent on MntWines
Income
Income vs MntFruits
Amount Spent on MntFruits
Income
Income vs MntMeatProducts
Amount Spent on MntMeatProducts
Income
Income vs MntFishProducts
Amount Spent on MntFishProducts
Income
Income vs MntSweetProducts
Amount Spent on MntSweetProducts
Income
Income vs MntGoldProds
Amount Spent on MntGoldProds
Income

In [50]:
```
columns_to_analyze = ['Income', 'Total_Amount_Spent','MntWines', 'MntFruits', 'MntMea

# Filter the DataFrame for incomes less than or equal to 160000
filtered_df = final_df[final_df['Income'] <= 160000]

# Create a figure and subplots
fig, axs = plt.subplots(3, 3, figsize=(12, 12))  # 3 rows, 2 columns

# Flatten the axes array for easier iteration
axs = axs.flatten()

# Iterate over each product column (excluding 'Income')
for i, column in enumerate(columns_to_analyze[1:], start=1):
    # Scatter plot with hue
    sns.scatterplot(data=filtered_df, x='Income', y=column, hue='Response', ax=axs[i-
    axs[i-1].set_title(f'Income vs {column}')
    axs[i-1].set_xlabel('Income')
    axs[i-1].set_ylabel(f'Amount Spent on {column}')
    axs[i-1].grid(True)

# Adjust layout to prevent overlap
plt.tight_layout()

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



8. Can you test the hypothesis that recent customers complain less in general compared to older customers?

In [51]:

```python
# Convert 'Dt_Customer' column to datetime format
final_df['Dt_Customer'] = pd.to_datetime(final_df['Dt_Customer'])

# Find the oldest and most recent dates
oldest_date = final_df['Dt_Customer'].min()
most_recent_date = final_df['Dt_Customer'].max()

print("Oldest Date:", oldest_date)
print("Most Recent Date:", most_recent_date)
```

```
C:\Users\shikh\AppData\Local\Temp\ipykernel_2336\328040276.py:2: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  final_df['Dt_Customer'] = pd.to_datetime(final_df['Dt_Customer'])

Oldest Date: 2012-07-30 00:00:00
Most Recent Date: 2014-06-29 00:00:00
```

In [52]:
```python
# Define recent and older customers based on a threshold (e.g., join date)
threshold_date = pd.to_datetime('2014-01-01')  # Example threshold date
recent_customers = final_df[final_df['Dt_Customer'] >= threshold_date]
older_customers = final_df[final_df['Dt_Customer'] < threshold_date]

# Count complaints for recent and older customers
recent_complaints = recent_customers['Complain'].sum()
older_complaints = older_customers['Complain'].sum()

# Total number of recent and older customers
total_recent_customers = recent_customers.shape[0]
total_older_customers = older_customers.shape[0]

# Create contingency table for chi-square test
contingency_table = [[recent_complaints, total_recent_customers - recent_complaints],
                     [older_complaints, total_older_customers - older_complaints]]

# Perform chi-square test
chi2_stat, p_value, _, _ = chi2_contingency(contingency_table)

# Print results
print("Chi-square test results:")
print("Chi-square statistic:", chi2_stat)
print("p-value:", p_value)
```

```
Chi-square test results:
Chi-square statistic: 0.7628363648756273
p-value: 0.3824423445637971
```

Interpreting the Results:

--> If the p-value is less than the chosen significance level (e.g., 0.05), reject the null hypothesis and conclude that recent customers complain less in general compared to older customers.
--> If the p-value is greater than the significance level, fail to reject the null hypothesis, indicating no significant difference in complaint rates between recent and older customers.

9. Do people who accept the offer in the first campaign also accept in any other campaign?

In [53]:
```
# Select relevant columns
campaign_data = final_df[['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'AcceptedCm

# Create a contingency table
contingency_table = pd.crosstab(campaign_data['AcceptedCmp1'],
                                [campaign_data['AcceptedCmp2'], campaign_data['Accept
                                 campaign_data['AcceptedCmp4'], campaign_data['Accept
                                rownames=['AcceptedCmp1'], colnames=['AcceptedCmp2',

# Perform chi-square test of independence
chi2_stat, p_value, _, _ = chi2_contingency(contingency_table)

# Print results
print("Chi-square test results:")
print("Chi-square statistic:", chi2_stat)
print("p-value:", p_value)
```

```
Chi-square test results:
Chi-square statistic: 463.55584709180073
p-value: 2.6775097340913834e-93
```

In [54]:
```
print(contingency_table)
```

```
AcceptedCmp2     0                       1
AcceptedCmp3     0              1        0        1
AcceptedCmp4     0       1      0        0  1     0
AcceptedCmp5     0   1   0   1  0    1   0  0  1  0  1
AcceptedCmp1
0             1777  59  84  24  129   6  1  8  4  2  2
1               52  21  14  23    8  13  0  2  8  0  3
```

Interpreting the Results:

--> Since the p-value (2.6775097340913834e-93) is significantly less than the chosen significance level (e.g., 0.05), we reject the null hypothesis.
--> Therefore, we conclude that there is a significant association between accepting offers in the first campaign and accepting offers in other campaigns.

10. Profile of people who respond vs. who don’t.

In [55]:
```
## Define bins for age
age_bins = [20, 40, 60, 80, 100, float('inf')]

## Create bins for age
final_df['Age_Bin'] = pd.cut(final_df['Age'], bins=age_bins, labels=['20-39', '40-59'

## Display the DataFrame with the new age bins
print(final_df[['Age', 'Age_Bin']].head())
```

```
   Age  Age_Bin
0   54    40-59
1   63    60-79
2   66    60-79
3   57    40-59
4   35    20-39
```

In [56]:
```python
# Define bins for income
income_bins = [0, 100000, 200000, 300000, 400000, 500000, float('inf')]

# Create bins for income
final_df['Income_Bin'] = pd.cut(final_df['Income'], bins=income_bins, labels=['0-99k'

# Display the DataFrame with the new income bins
print(final_df[['Income', 'Income_Bin']].head())
```

```
    Income Income_Bin
0  84835.0      0-99k
1  57091.0      0-99k
2  67267.0      0-99k
3  32474.0      0-99k
4  21474.0      0-99k
```

In [57]:
```python
final_df.head()
```

Out[57]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Country	R
0	1826	1970	Graduation	Divorced	84835.0	0	0	2014-06-16	SP	
1	1	1961	Graduation	Single	57091.0	0	0	2014-06-15	CA	
2	10476	1958	Graduation	Married	67267.0	0	1	2014-05-13	US	
3	1386	1967	Graduation	Together	32474.0	1	1	2014-05-11	AUS	
4	5371	1989	Graduation	Single	21474.0	1	0	2014-04-08	SP	

5 rows × 33 columns

In [58]:
```python
final_df['Income_Bin'].value_counts()
```

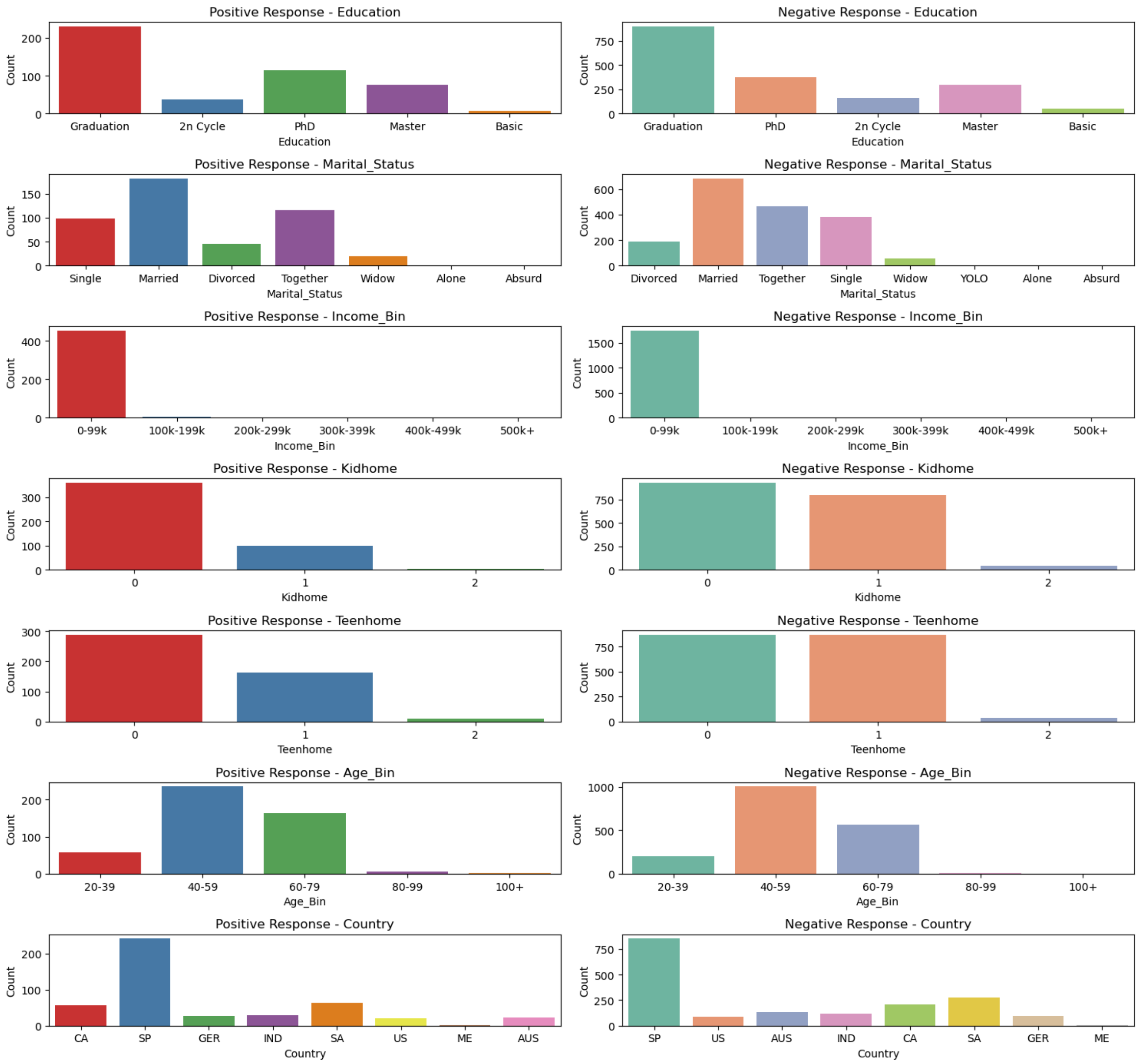
Out[58]:
```
Income_Bin
0-99k        2203
100k-199k      12
500k+           1
200k-299k       0
300k-399k       0
400k-499k       0
Name: count, dtype: int64
```

In [59]:
```python
final_df['Age_Bin'].value_counts()
```

Out[59]:
```
Age_Bin
40-59    1237
60-79     732
20-39     259
80-99       9
100+        3
Name: count, dtype: int64
```

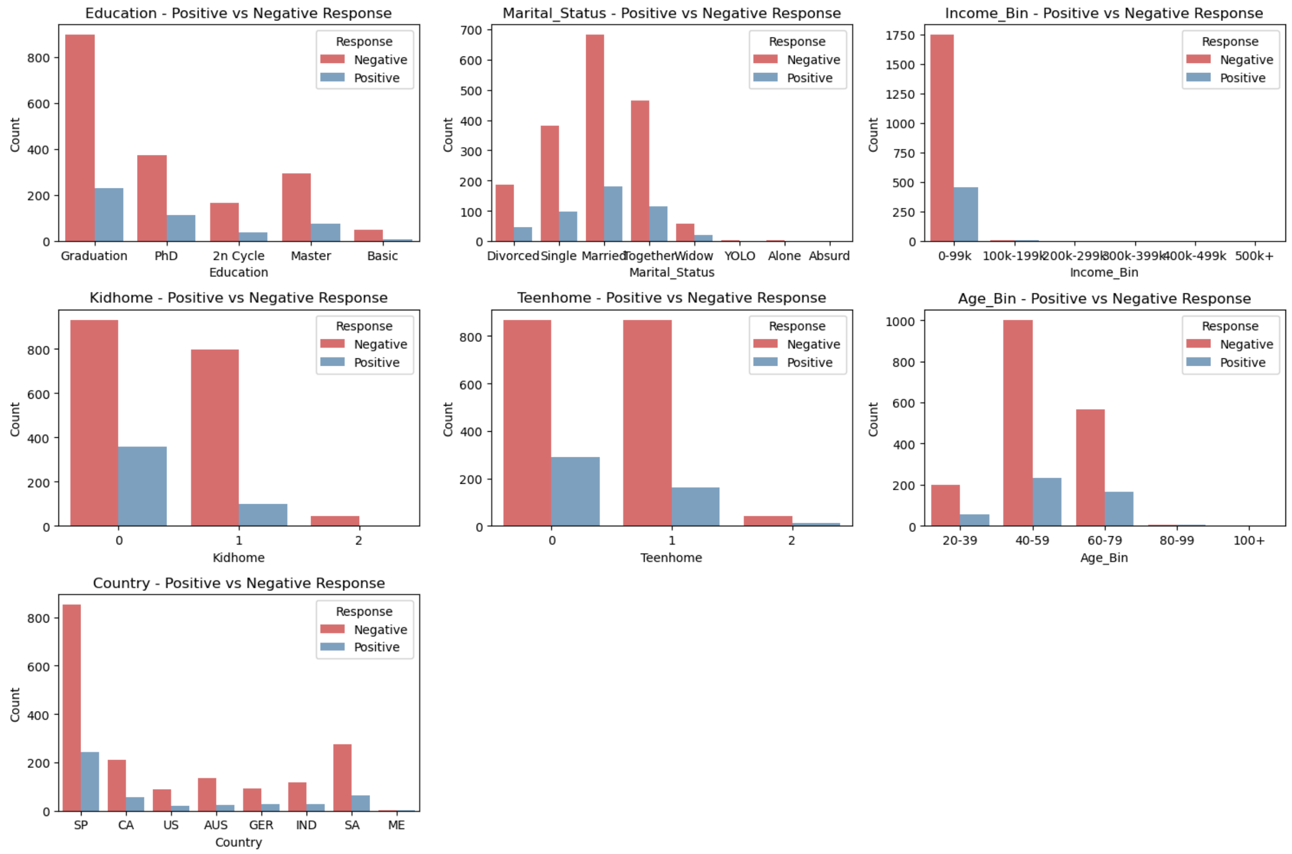
In [60]:

```
# Filter the DataFrame for positive response (Response == 1)
positive_response_df = final_df[final_df['Response'] == 1]

# Filter the DataFrame for negative response (Response == 0)
negative_response_df = final_df[final_df['Response'] == 0]

# Select columns for comparison
columns_to_compare = ['Education', 'Marital_Status', 'Income_Bin', 'Kidhome', 'Teenhc

# Plot side-by-side comparisons
fig, axes = plt.subplots(len(columns_to_compare), 2, figsize=(15, 2*len(columns_to_cc

for i, column in enumerate(columns_to_compare):
    # Plot for positive response
    sns.countplot(data=positive_response_df, x=column, ax=axes[i, 0], palette='Set1')
    axes[i, 0].set_title(f'Positive Response - {column}')
    axes[i, 0].set_xlabel(column)
    axes[i, 0].set_ylabel('Count')

    # Plot for negative response
    sns.countplot(data=negative_response_df, x=column, ax=axes[i, 1], palette='Set2')
    axes[i, 1].set_title(f'Negative Response - {column}')
    axes[i, 1].set_xlabel(column)
    axes[i, 1].set_ylabel('Count')

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

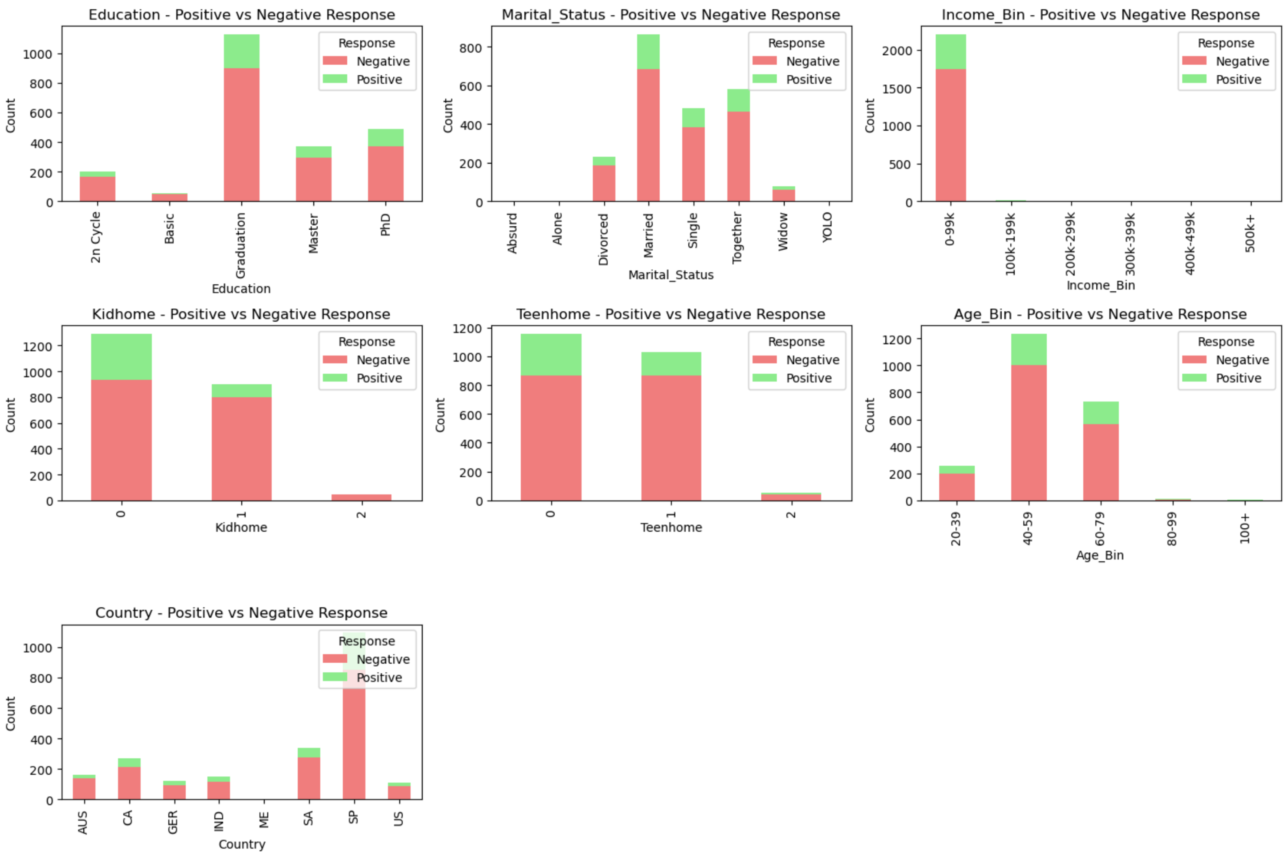

Positive Response - Education
Count
0
100
200
Graduation
2n Cycle
PhD
Master
Basic
Education
Negative Response - Education
Count
0
250
500
750
Graduation
PhD
2n Cycle
Master
Basic
Education
Positive Response - Marital_Status
Count
0
50
100
150
Single
Married
Divorced
Together
Widow
Alone
Absurd
Marital_Status
Negative Response - Marital_Status
Count
0
200
400
600
Divorced
Married
Together
Single
Widow
YOLO
Alone
Absurd
Marital_Status
Positive Response - Income_Bin
Count
0
200
400
0-99k
100k-199k
200k-299k
300k-399k
400k-499k
500k+
Income_Bin
Negative Response - Income_Bin
Count
0
500
1000
1500
0-99k
100k-199k
200k-299k
300k-399k
400k-499k
500k+
Income_Bin
Positive Response - Kidhome
Count
0
100
200
300
0
1
2
Kidhome
Negative Response - Kidhome
Count
0
250
500
750
0
1
2
Kidhome
Positive Response - Teenhome
Count
0
100
200
300
0
1
2
Teenhome
Negative Response - Teenhome
Count
0
250
500
750
0
1
2
Teenhome
Positive Response - Age_Bin
Count
0
100
200
20-39
40-59
60-79
80-99
100+
Age_Bin
Negative Response - Age_Bin
Count
0
500
1000
20-39
40-59
60-79
80-99
100+
Age_Bin
Positive Response - Country
Count
0
100
200
CA
SP
GER
IND
SA
US
ME
AUS
Country
Negative Response - Country
Count
0
250
500
750
SP
US
AUS
IND
CA
SA
GER
ME
Country

In [61]:

```
# Select columns for comparison
columns_to_compare = ['Education', 'Marital_Status', 'Income_Bin', 'Kidhome', 'Teenhc

# Set up the matplotlib figure
plt.figure(figsize=(15, 10))

# Loop through each column and create a countplot
for i, column in enumerate(columns_to_compare, start=1):
    plt.subplot(3, 3, i)
    sns.countplot(data=final_df, x=column, hue='Response', palette='Set1', alpha=0.7)
    plt.title(f'{column} - Positive vs Negative Response')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.legend(title='Response', labels=['Negative', 'Positive'], loc='upper right')

# Adjust layout
plt.tight_layout()

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

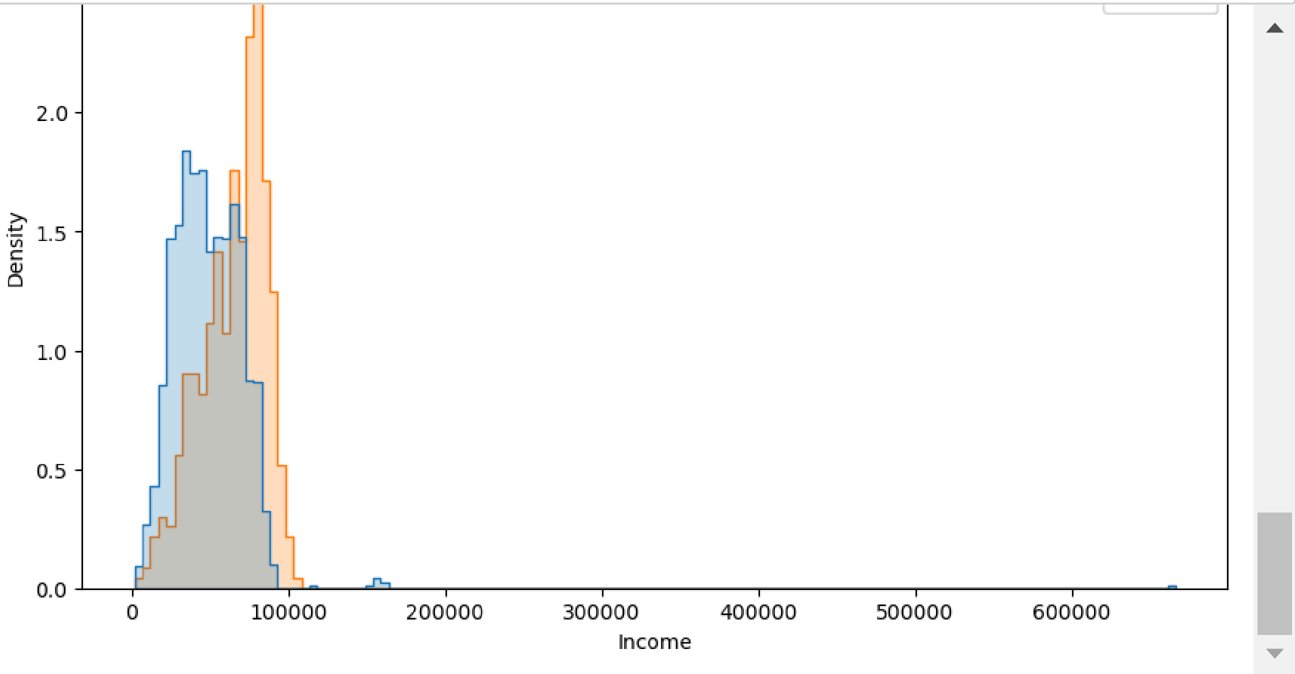


```python
In [62]: columns_to_compare = ['Education', 'Marital_Status', 'Income_Bin', 'Kidhome', 'Teenhc

         # Set up the matplotlib figure
         plt.figure(figsize=(15, 10))

         # Loop through each column and create a stacked bar chart
         for i, column in enumerate(columns_to_compare, start=1):
             plt.subplot(3, 3, i)

             # Aggregate data for the current column
             data = final_df.groupby([column, 'Response']).size().unstack()

             # Plot stacked bar chart
             data.plot(kind='bar', stacked=True, color=['lightcoral', 'lightgreen'], ax=plt.gc

             plt.title(f'{column} - Positive vs Negative Response')
             plt.xlabel(column)
             plt.ylabel('Count')
             plt.legend(title='Response', labels=['Negative', 'Positive'], loc='upper right')

         # Adjust layout
         plt.tight_layout()

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



In [63]:

```python
# Splitting the DataFrame into responders and non-responders
responders = final_df[final_df['Response'] == 1]
non_responders = final_df[final_df['Response'] == 0]

# Example attributes: 'age', 'income', you can add more relevant attributes
attributes = ['Year_Birth', 'Income']

# Descriptive Statistics
for attribute in attributes:
    print(f"Statistics for {attribute}:")
    print("Responders:")
    print(responders[attribute].describe())
    print("Non-Responders:")
    print(non_responders[attribute].describe())
    print("\n")

# Data Visualization
for attribute in attributes:
    plt.figure(figsize=(10, 6))
    sns.histplot(data=final_df, x=attribute, hue='Response', element='step', stat='de
    plt.title(f"Distribution of {attribute} for Responders vs Non-Responders")
    plt.show()
```



Hypothesis 1: Customers who spend more on wines are more likely to respond positively to marketing campaigns promoting wine-related products.

In [64]:
```
# Define the variables
category = 'MntWines'  # Amount spent on wines
response_column = 'Response'  # Response column in your dataset

# Split data into groups based on response (0 = not responded, 1 = responded)
responded = final_df[final_df[response_column] == 1][category]
not_responded = final_df[final_df[response_column] == 0][category]

# Perform t-test to compare means of amount spent on wines between responded and not
t_stat, p_value = stats.ttest_ind(responded, not_responded)

# Define significance level
alpha = 0.05

# Visualization - Box plot of amount spent on wines by response status
plt.figure(figsize=(8, 6))
data.boxplot(column=category, by=response_column, grid=False)
plt.title("Box plot of amount spent on wines by response status")
plt.ylabel("Amount spent on wines")
plt.xlabel("Response status")
plt.xticks([1, 2], ['Not responded', 'Responded'])

# Annotate the p-value on the plot
if p_value < alpha:
    plt.text(1.1, 200, f'P-value = {p_value:.4f}\nSignificant', fontsize=12, color='r
else:
    plt.text(1.1, 200, f'P-value = {p_value:.4f}\nNot significant', fontsize=12, colo


# Print p value
print(f'the p value is {p_value}')

# Print the result
if p_value < alpha:
    print("The amount spent on wines significantly differs between customers who resp
else:
    print("There is no significant difference in the amount spent on wines between cu


plt.figure(figsize=(15,15))
plt.show()
plt.tight_layout()
```

```
    930 if grouper is None:
--> 931     grouper, exclusions, obj = get_grouper(
    932         obj,
    933         keys,
    934         axis=axis,
    935         level=level,
    936         sort=sort,
    937         observed=observed,
    938         dropna=self.dropna,
    939     )
    941 self.obj = obj
    942 self.axis = obj._get_axis_number(axis)

File ~\anaconda3\Lib\site-packages\pandas\core\groupby\grouper.py:985, in get_grouper(obj, key, axis, level, sort, observed, validate, dropna)
    983         in_axis, level, gpr = False, gpr, None
    984     else:
--> 985         raise KeyError(gpr)
    986 elif isinstance(gpr, Grouper) and gpr.key is not None:
    987     # Add key to exclusions
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: