Marketing Campaign Customer Segmentation

Jessica Chipera

Massachusetts Institute of Technology - Data Science and Machine Learning

Imports

In [2]:
```
!pip install scikit-learn-extra
```

```
Collecting scikit-learn-extra
  Downloading scikit_learn_extra-0.2.0-cp39-cp39-win_amd64.whl (380 kB)
Requirement already satisfied: numpy>=1.13.3 in c:\users\jessica\anaconda3\lib\site-p
ackages (from scikit-learn-extra) (1.20.3)
Requirement already satisfied: scipy>=0.19.1 in c:\users\jessica\anaconda3\lib\site-p
ackages (from scikit-learn-extra) (1.7.1)
Requirement already satisfied: scikit-learn>=0.23.0 in c:\users\jessica\anaconda3\li
b\site-packages (from scikit-learn-extra) (0.24.2)
Requirement already satisfied: joblib>=0.11 in c:\users\jessica\anaconda3\lib\site-pa
ckages (from scikit-learn>=0.23.0->scikit-learn-extra) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\jessica\anaconda3\li
b\site-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (2.2.0)
Installing collected packages: scikit-learn-extra
Successfully installed scikit-learn-extra-0.2.0
```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pylab as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler as ss
from scipy.linalg import eigh
from scipy.spatial.distance import cdist, pdist
from sklearn.decomposition import PCA
from sklearn.manifold import MDS
from sklearn.manifold import TSNE
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.mixture import GaussianMixture
from sklearn_extra.cluster import KMedoids
import warnings
warnings.filterwarnings("ignore")

from statsmodels.formula.api import ols
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import r2_score,mean_absolute_percentage_error,mean_absolute_er
```

Previewing the data

In [2]:
```
df=pd.read_csv("C:/Users/Jessica/Dropbox/MIT/marketing_campaign.csv")
df.head()
```

Out[2]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2012	58
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2014	38
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013	26
3	6182	1984	Graduation	Together	26646.0	1	0	10-02-2014	26
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2014	94

5 rows × 27 columns

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 27 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   float64
 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  Complain             2240 non-null   int64  
 26  Response             2240 non-null   int64  
dtypes: float64(1), int64(23), object(3)
memory usage: 472.6+ KB
```

Observations:

- There are 27 columns and 240 rows.
- All columns contain 2240 non-null values, meaning that there are no NaN values.
- Most variables are integers. However there are three objects and one "float" variable

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

Out[4]:
```
ID                   2240
Year_Birth             59
Education               5
Marital_Status          8
Income               1974
Kidhome                 3
Teenhome                3
Dt_Customer           663
Recency               100
MntWines              776
MntFruits             158
```

```
MntMeatProducts         558
MntFishProducts         182
MntSweetProducts        177
MntGoldProds            213
NumDealsPurchases        15
NumWebPurchases          15
NumCatalogPurchases      14
NumStorePurchases        14
NumWebVisitsMonth        16
AcceptedCmp3              2
AcceptedCmp4              2
AcceptedCmp5              2
AcceptedCmp1              2
AcceptedCmp2              2
Complain                  2
Response                  2
dtype: int64
```

Data Processing and Exploratory Data Analysis

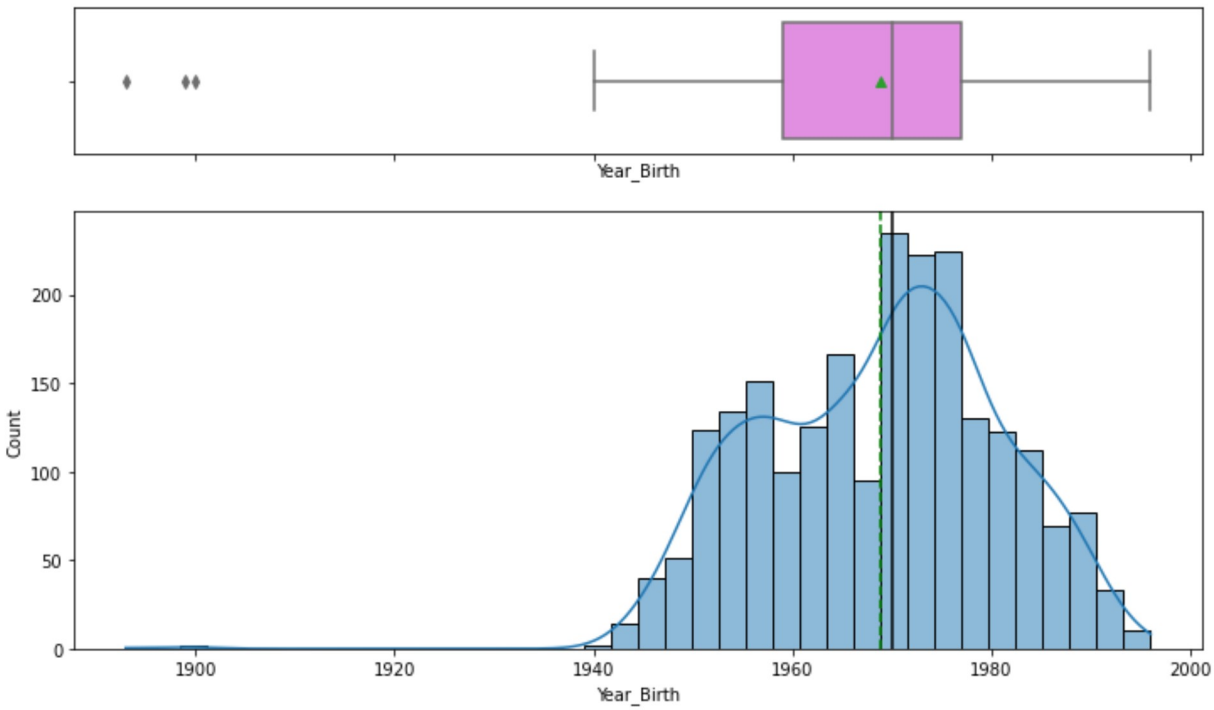
Discrete, Integer Data

- First, all of the integer data can be examined for distributions.

In [5]:

```
def histogram_boxplot(df, feature, figsize = (12, 7), kde = True, bins = None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows = 2,  # Number of rows of the subplot grid = 2
        sharex = True,  # x-axis will be shared among all subplots
        gridspec_kw = {"height_ratios": (0.25, 0.75)},
        figsize = figsize,
    )  # Creating the 2 subplots
    sns.boxplot(
        data = df, x = feature, ax = ax_box2, showmeans = True, color = "violet"
    )  # Boxplot will be created and a star will indicate the mean value of the colu
    sns.histplot(
        data = df, x = feature, kde = kde, ax = ax_hist2, bins = bins, palette = "wi
    ) if bins else sns.histplot(
        data = df, x = feature, kde = kde, ax = ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        df[feature].mean(), color = "green", linestyle = "--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        df[feature].median(), color = "black", linestyle = "-"
    )  # Add median to the histogram
```

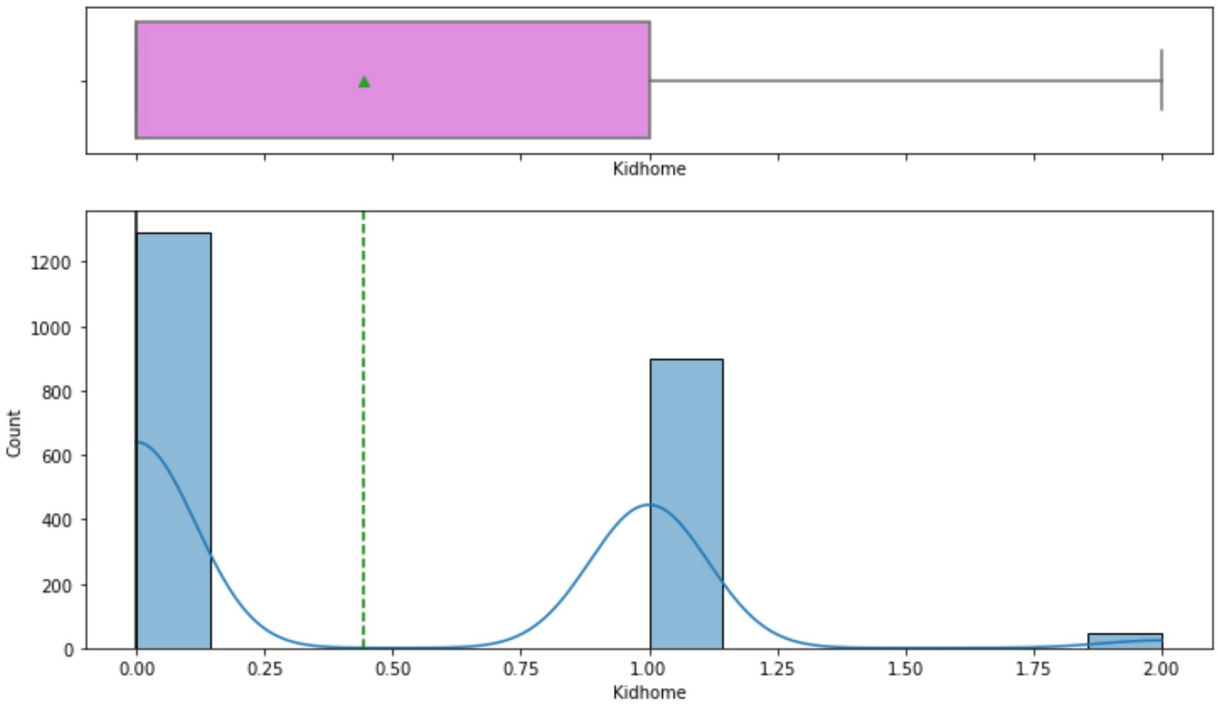
In [6]:
```
histogram_boxplot(df,"Year_Birth")
```



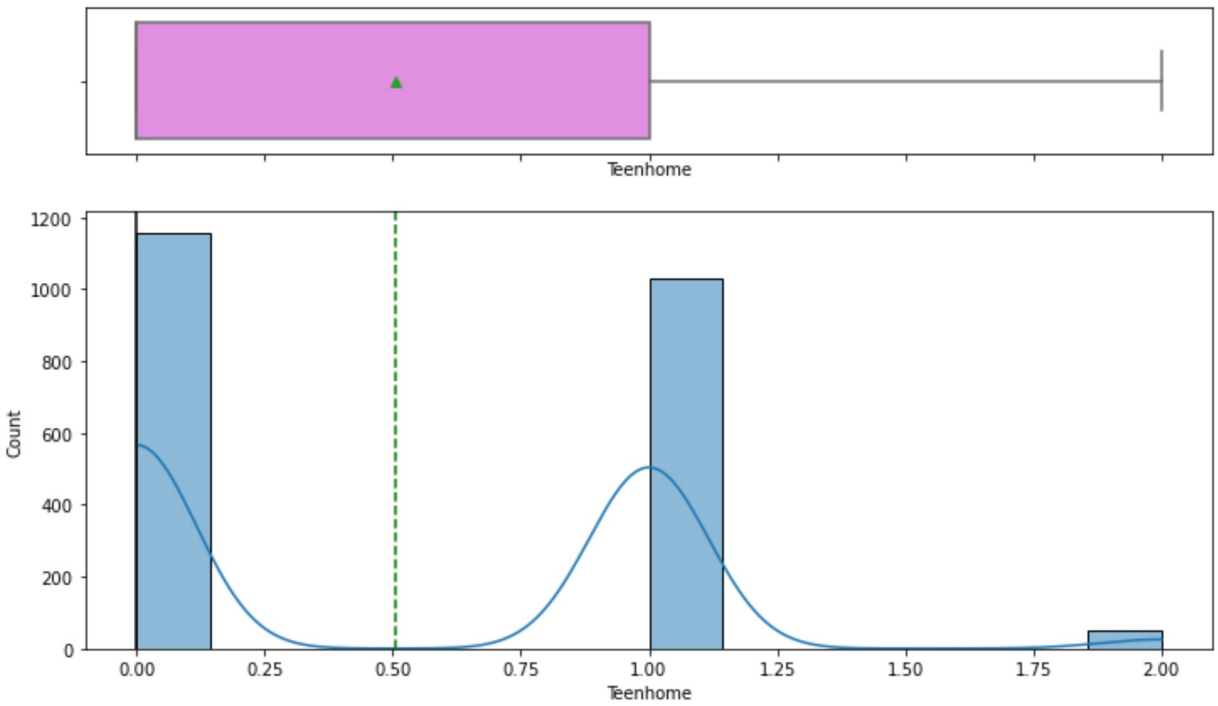
Observations:

- It seems unlikely that someone born in 1900 or before would be alive today. Those are probably typos / other bad data, and can be deleted.
- Otherwise, the data seems mostly normal in nature.

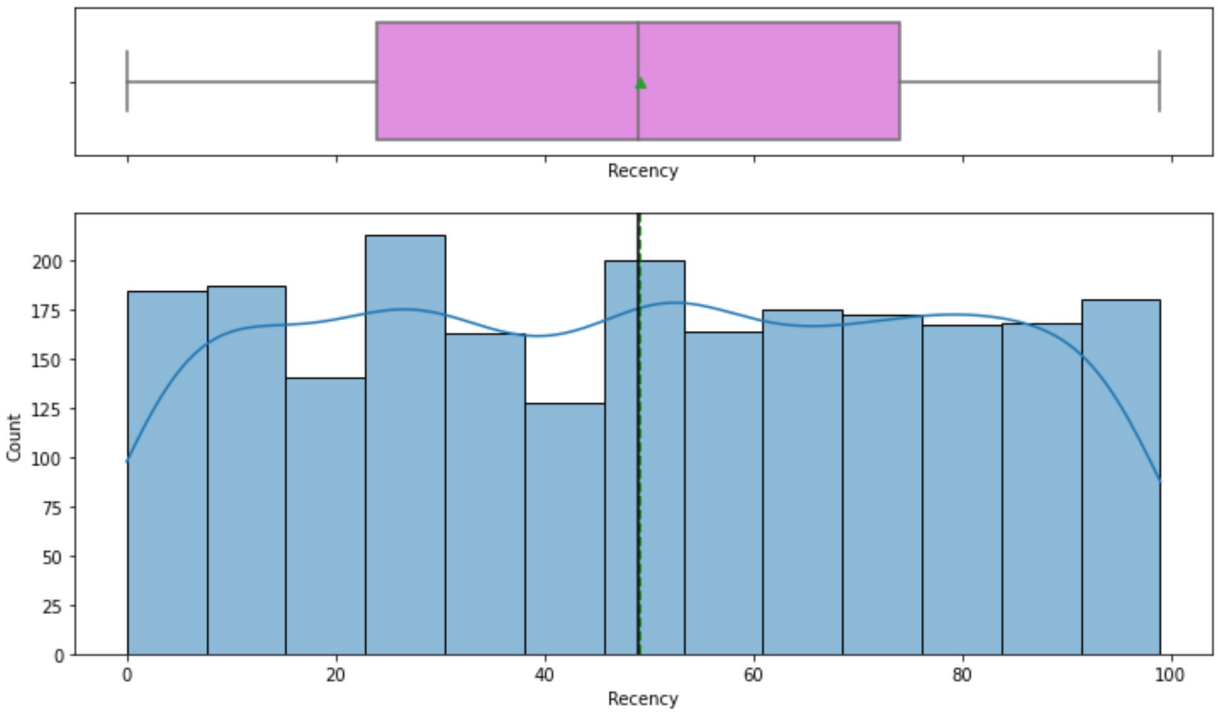
In [7]:
```
histogram_boxplot(df,"Kidhome")
```



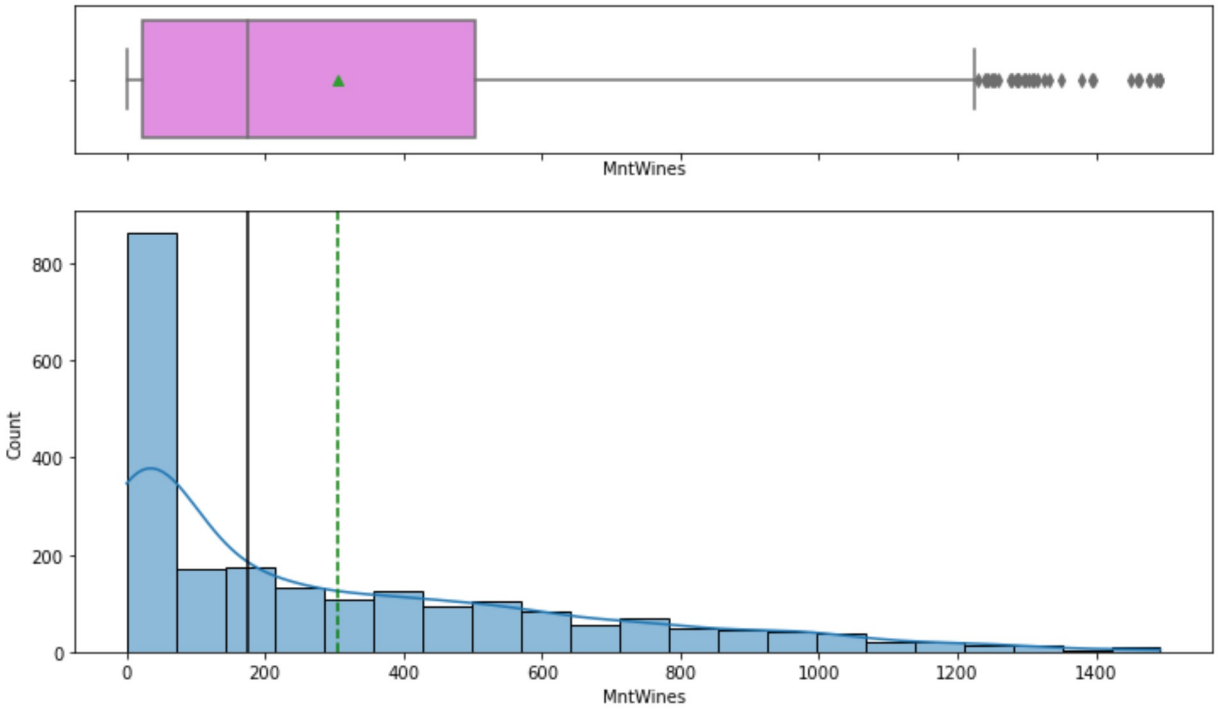
In [8]:
```
histogram_boxplot(df,"Teenhome")
```



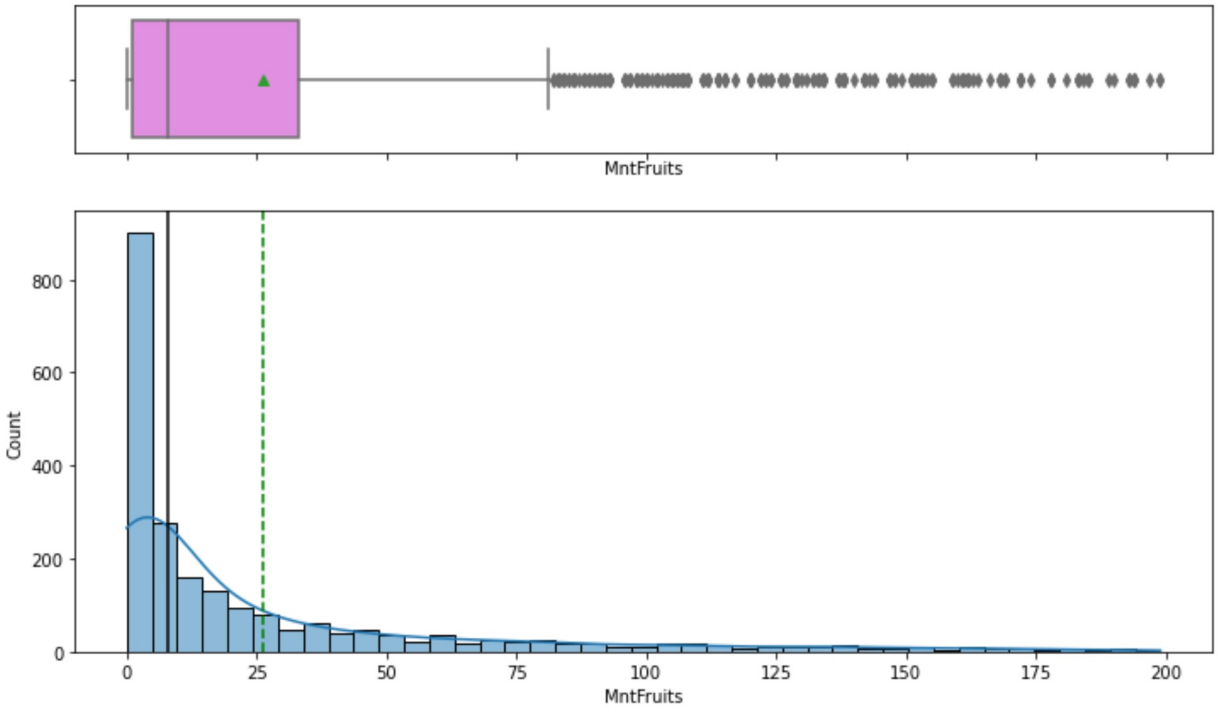
In [9]:
```
histogram_boxplot(df,"Recency")
```



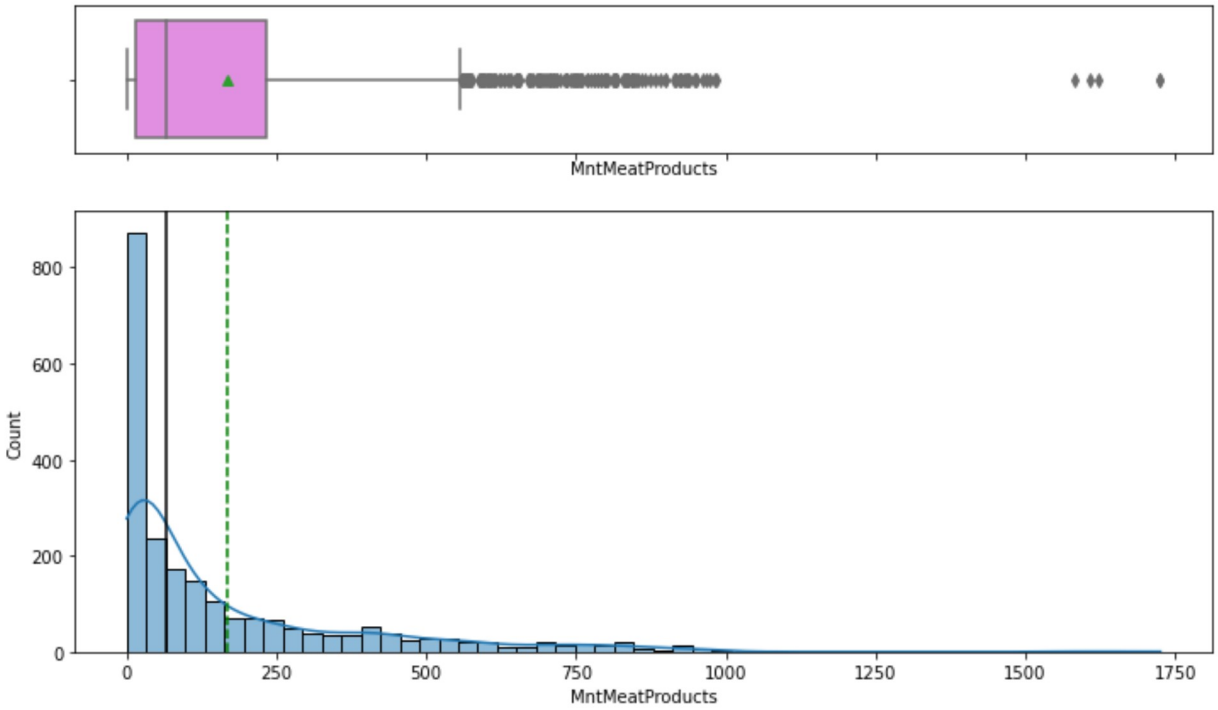
In [10]:
```
histogram_boxplot(df,"MntWines")
```



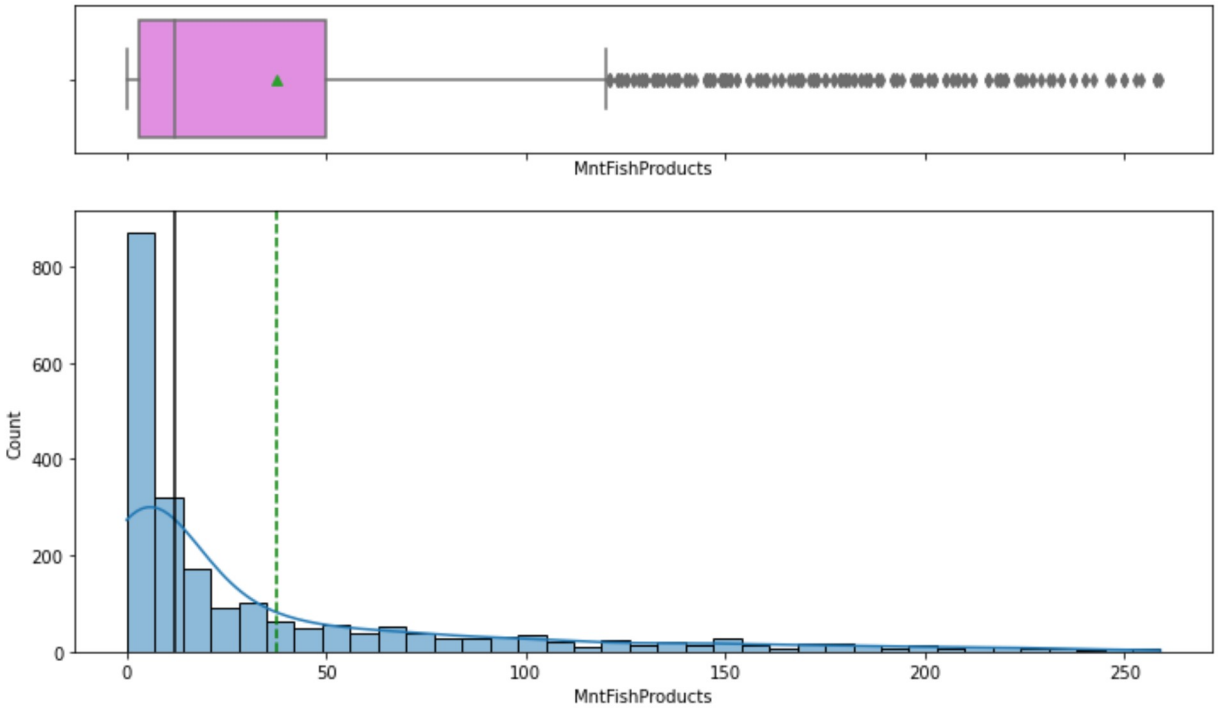
In [11]:
```
histogram_boxplot(df,"MntFruits")
```



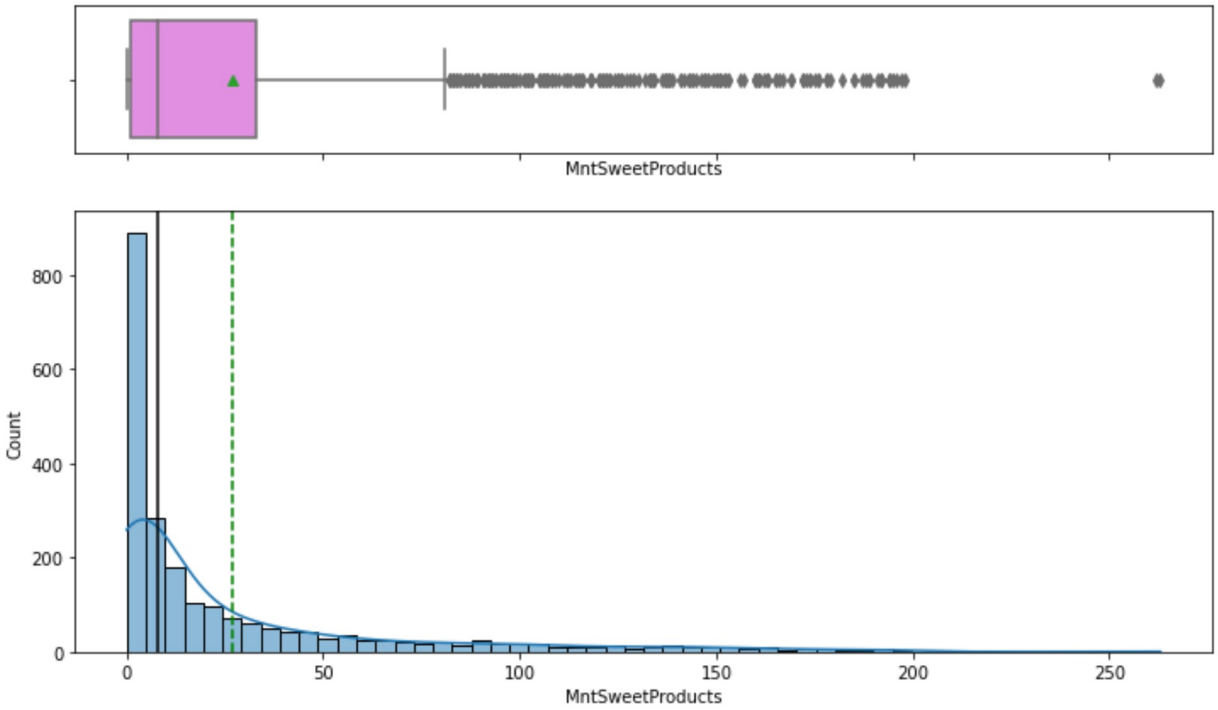
In [12]:
```
histogram_boxplot(df,"MntMeatProducts")
```



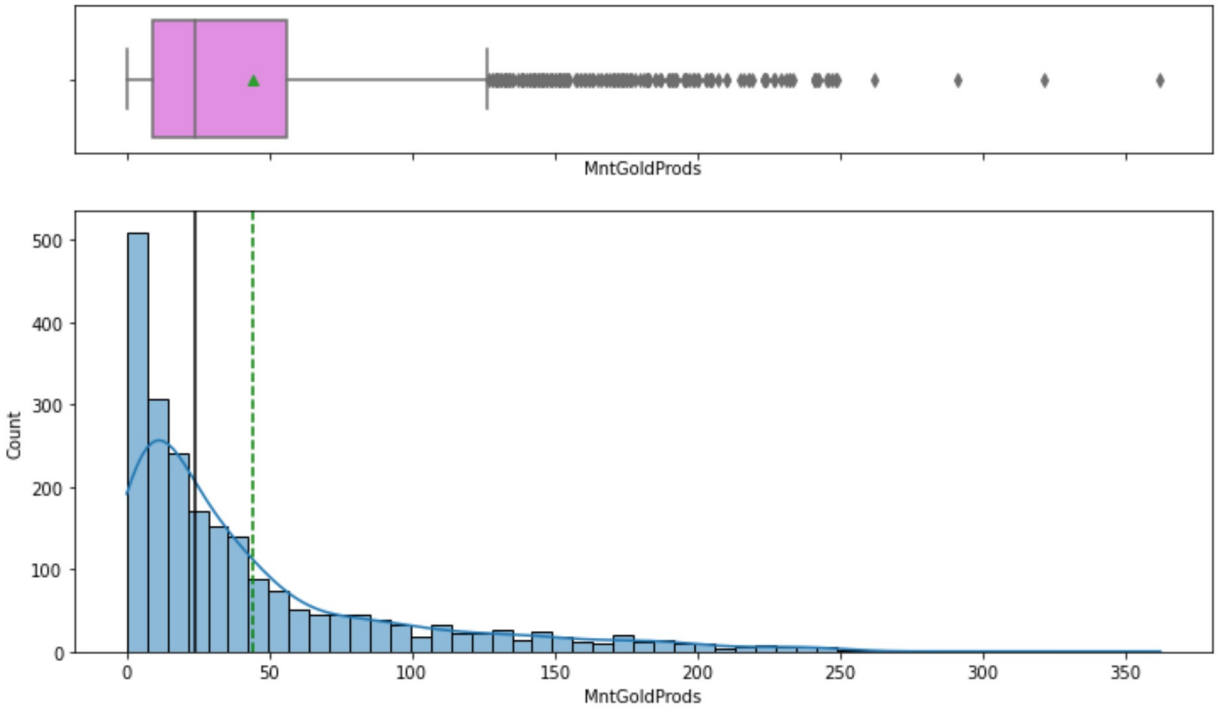
In [13]:
```
histogram_boxplot(df,"MntFishProducts")
```



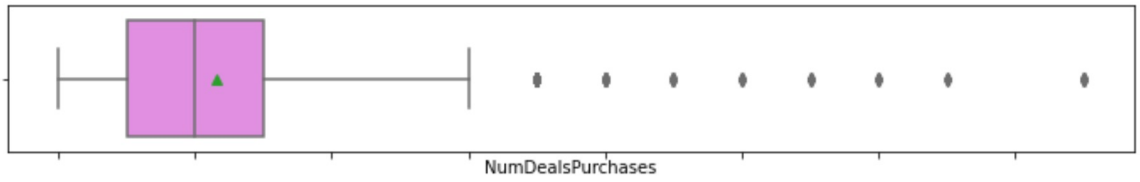
In [14]:
```
histogram_boxplot(df,"MntSweetProducts")
```

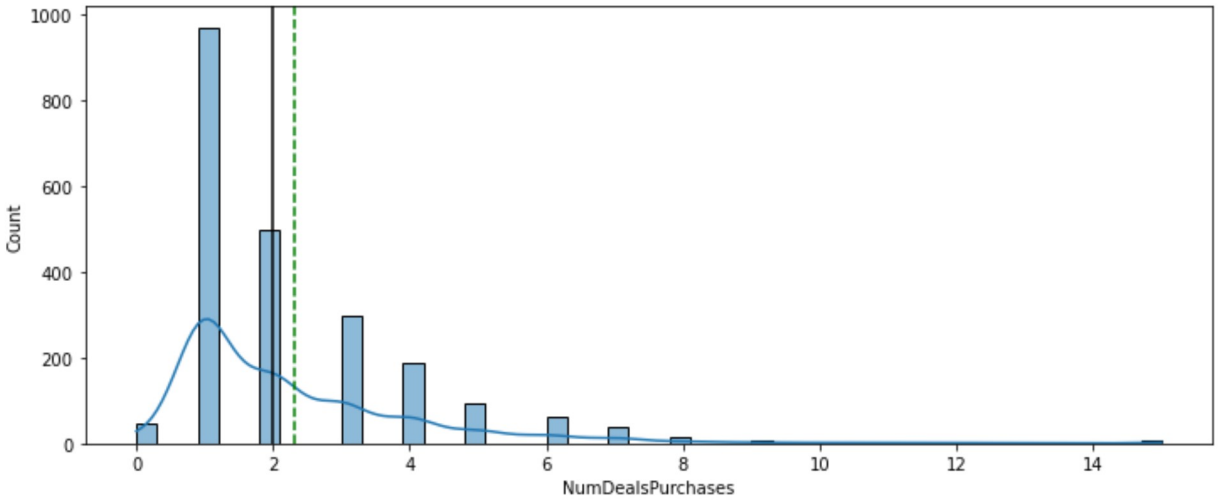


In [15]:
```
histogram_boxplot(df,"MntGoldProds")
```

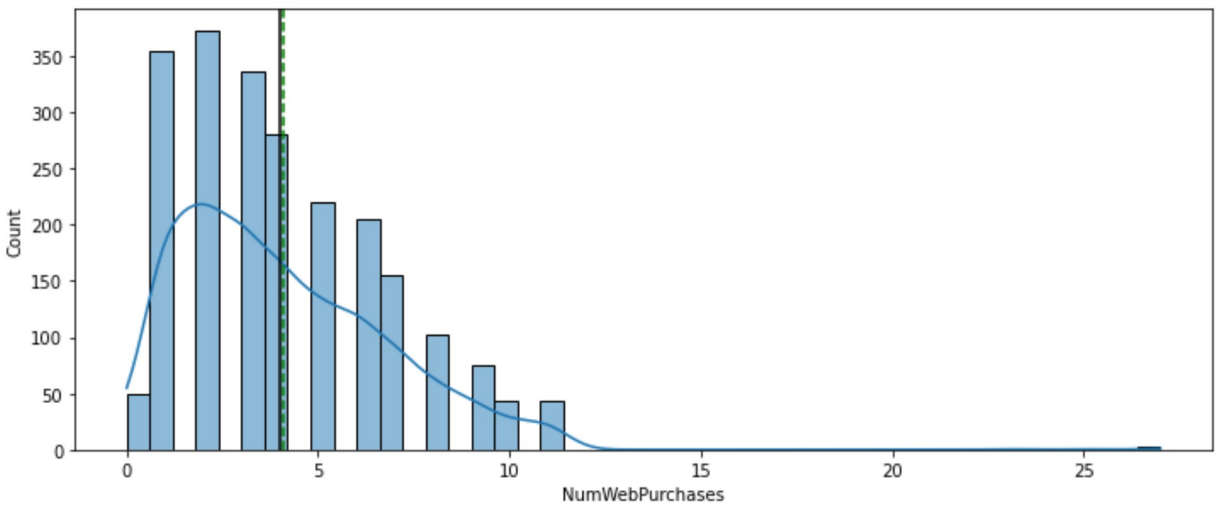


In [16]:
```
histogram_boxplot(df,"NumDealsPurchases")
```

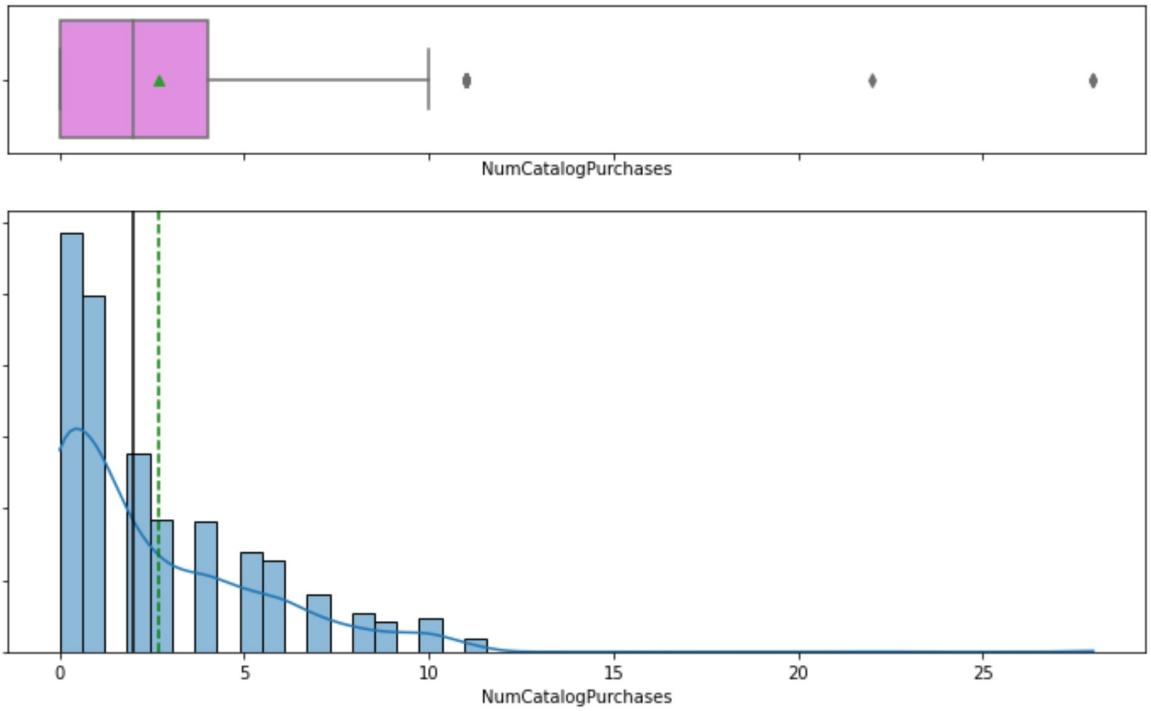




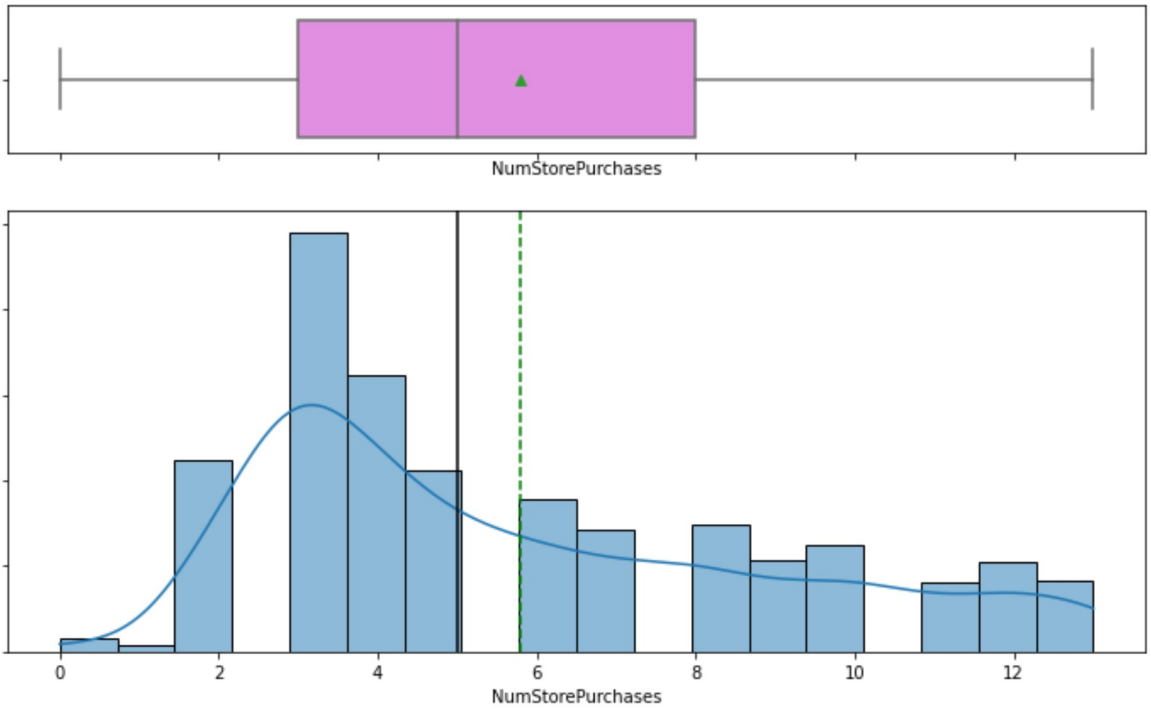
In [17]:
```
histogram_boxplot(df,"NumWebPurchases")
```



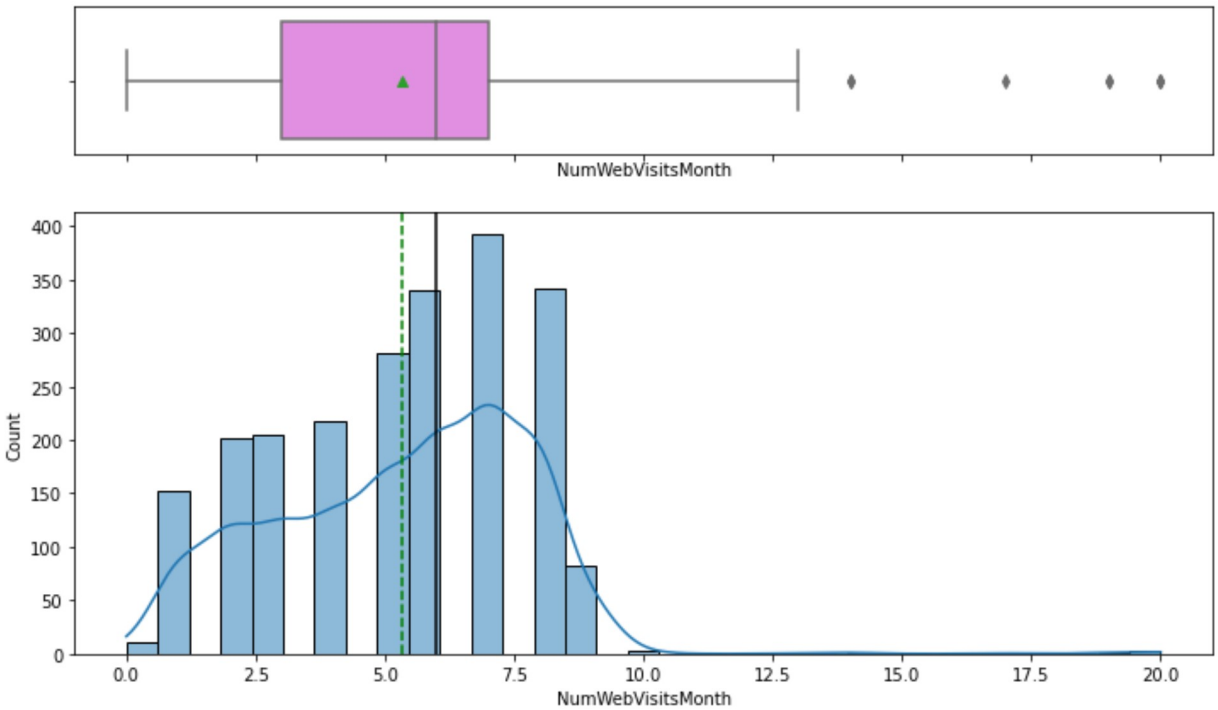
In [18]:
```
histogram_boxplot(df,"NumCatalogPurchases")
```



In [19]:
```
histogram_boxplot(df,"NumStorePurchases")
```



In [20]:
```
histogram_boxplot(df,"NumWebVisitsMonth")
```



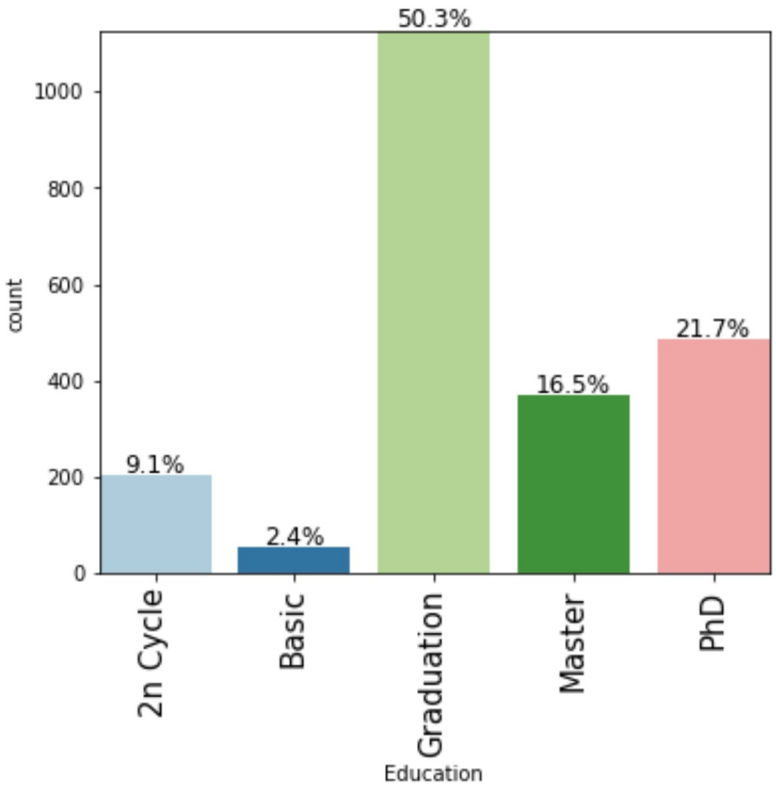
Next, Categorical Data

In [21]:

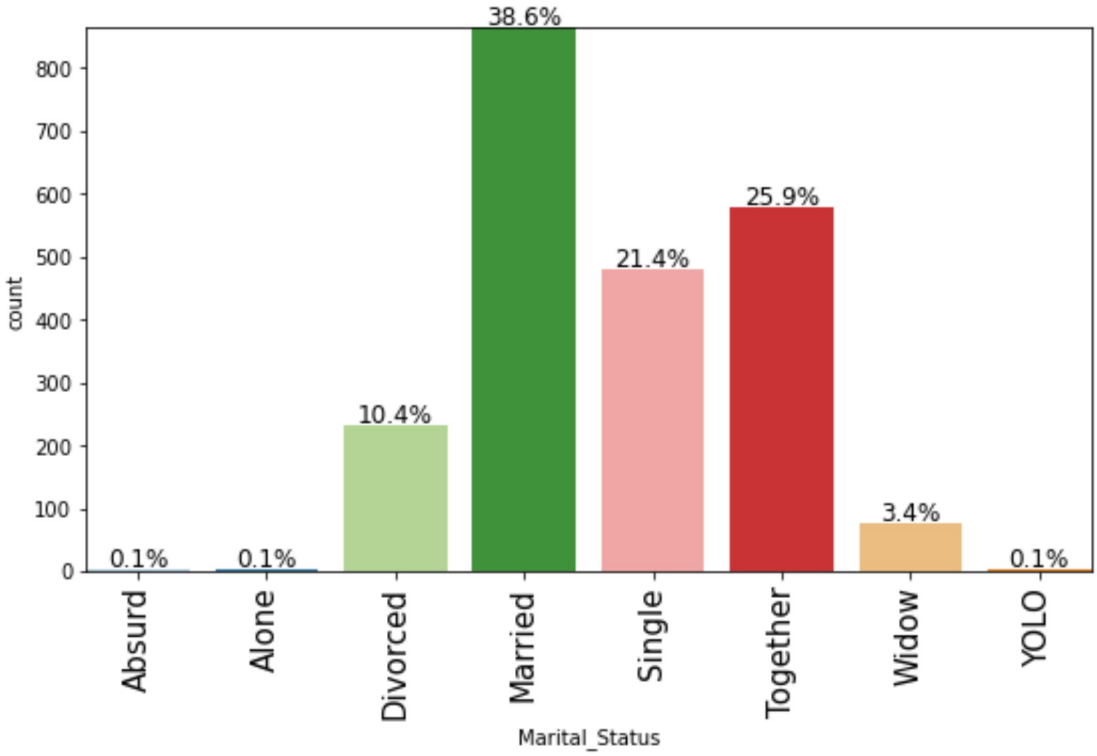
```
def labeled_barplot(data, feature, perc = False, n = None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # Length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize = (count + 1, 5))
    else:
        plt.figure(figsize = (n + 1, 5))

    plt.xticks(rotation = 90, fontsize = 15)
    ax = sns.countplot(
        data = data,
        x = feature,
        palette = "Paired",
        order = data[feature].value_counts().index[:n].sort_values(),
    )

    ax.margins(0)

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # Percentage of each class of the category
        else:
            label = p.get_height()  # Count of each level of the category

        x = p.get_x() + p.get_width() / 2  # Width of the plot
        y = p.get_height()  # Height of the plot

        ax.annotate(
            label,
            (x, y),
            ha = "center",
            va = "center",
            size = 12,
            xytext = (0, 5),
            textcoords = "offset points",
        )  # Annotate the percentage

    plt.show()
```

In [22]:

```
labeled_barplot(df,"Education",perc=True)
```



In [23]:
```
labeled_barplot(df,"Marital_Status",perc=True)
```



In [24]:
```
labeled_barplot(df,"Dt_Customer",perc=True)
```

In [25]:
```
df.describe().T
```

Out[25]:

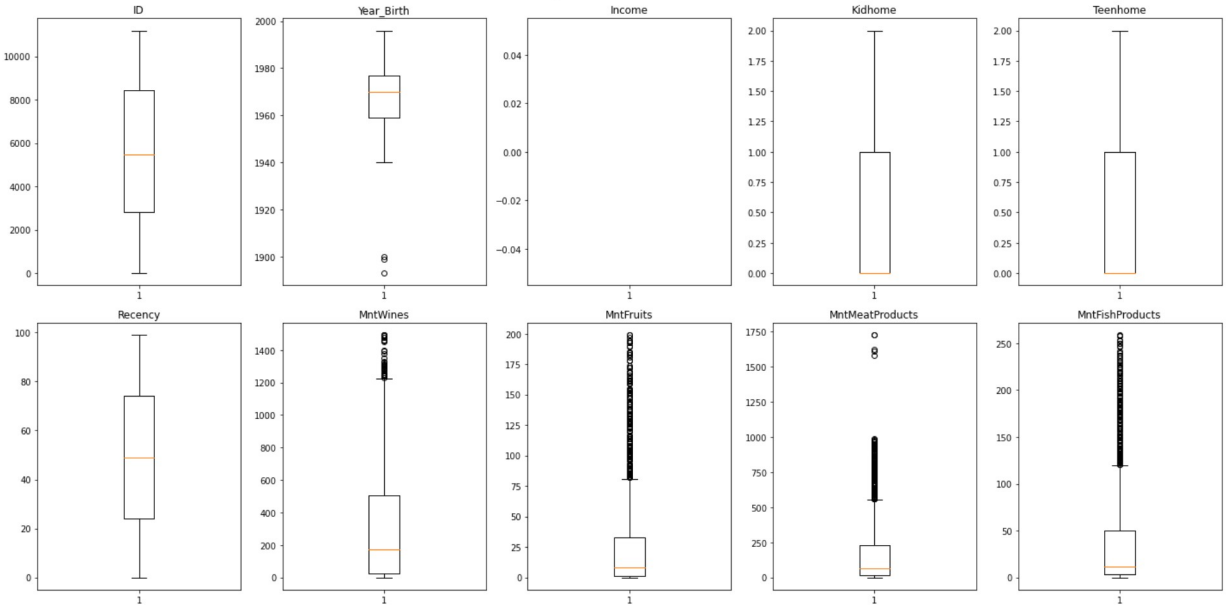
	count	mean	std	min	25%	50%	75%	m
ID	2240.0	5592.159821	3246.662198	0.0	2828.25	5458.5	8427.75	1119
Year_Birth	2240.0	1968.805804	11.984069	1893.0	1959.00	1970.0	1977.00	199
Income	2216.0	52247.251354	25173.076661	1730.0	35303.00	51381.5	68522.00	66666
Kidhome	2240.0	0.444196	0.538398	0.0	0.00	0.0	1.00	
Teenhome	2240.0	0.506250	0.544538	0.0	0.00	0.0	1.00	
Recency	2240.0	49.109375	28.962453	0.0	24.00	49.0	74.00	9
MntWines	2240.0	303.935714	336.597393	0.0	23.75	173.5	504.25	149
MntFruits	2240.0	26.302232	39.773434	0.0	1.00	8.0	33.00	19
MntMeatProducts	2240.0	166.950000	225.715373	0.0	16.00	67.0	232.00	172
MntFishProducts	2240.0	37.525446	54.628979	0.0	3.00	12.0	50.00	25
MntSweetProducts	2240.0	27.062946	41.280498	0.0	1.00	8.0	33.00	26
MntGoldProds	2240.0	44.021875	52.167439	0.0	9.00	24.0	56.00	36
NumDealsPurchases	2240.0	2.325000	1.932238	0.0	1.00	2.0	3.00	1
NumWebPurchases	2240.0	4.084821	2.778714	0.0	2.00	4.0	6.00	2
NumCatalogPurchases	2240.0	2.662054	2.923101	0.0	0.00	2.0	4.00	2
NumStorePurchases	2240.0	5.790179	3.250958	0.0	3.00	5.0	8.00	1
NumWebVisitsMonth	2240.0	5.316518	2.426645	0.0	3.00	6.0	7.00	2
AcceptedCmp3	2240.0	0.072768	0.259813	0.0	0.00	0.0	0.00	
AcceptedCmp4	2240.0	0.074554	0.262728	0.0	0.00	0.0	0.00	
AcceptedCmp5	2240.0	0.072768	0.259813	0.0	0.00	0.0	0.00	
AcceptedCmp1	2240.0	0.064286	0.245316	0.0	0.00	0.0	0.00	
AcceptedCmp2	2240.0	0.012946	0.113069	0.0	0.00	0.0	0.00	
Complain	2240.0	0.009375	0.096391	0.0	0.00	0.0	0.00	
Response	2240.0	0.149107	0.356274	0.0	0.00	0.0	0.00	

Outlier Check

In [26]:

```
plt.figure(figsize=(20,10))
numeric_columns=df.select_dtypes(include=np.number).columns.tolist()
for i, variable in enumerate(numeric_columns):
    plt.subplot(2, 5, i + 1)
    plt.boxplot(df[variable], whis = 1.5)
    plt.tight_layout()
    plt.title(variable)
plt.show()
```

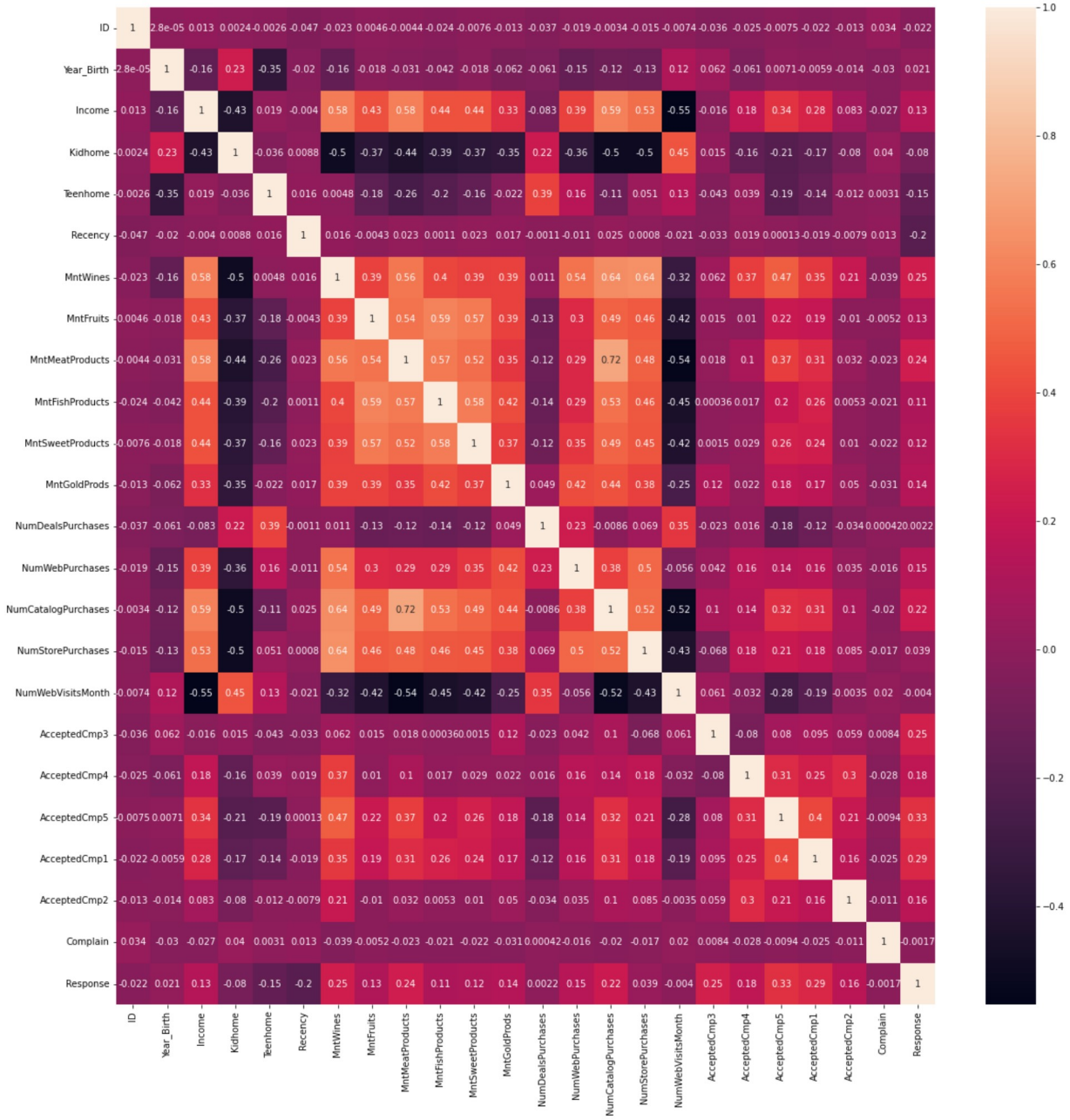
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_20656/2278909642.py in <module>
      2 numeric_columns=df.select_dtypes(include=np.number).columns.tolist()
      3 for i, variable in enumerate(numeric_columns):
----> 4     plt.subplot(2, 5, i + 1)
      5     plt.boxplot(df[variable], whis = 1.5)
      6     plt.tight_layout()

~\anaconda3\lib\site-packages\matplotlib\pyplot.py in subplot(*args, **kwargs)
   1270
   1271     # First, search for an existing subplot with a matching spec.
-> 1272     key = SubplotSpec._from_subplot_args(fig, args)
   1273
   1274     for ax in fig.axes:

~\anaconda3\lib\site-packages\matplotlib\gridspec.py in _from_subplot_args(figure, ar
gs)
    651                     num = int(num)
    652                 if num < 1 or num > rows*cols:
--> 653                     raise ValueError(
    654                         f"num must be 1 <= num <= {rows*cols}, not {num}")
    655                 i = j = num

ValueError: num must be 1 <= num <= 10, not 11
```



Bivariate Analysis

In [27]:

```
plt.figure(figsize = (20,20))
sns.heatmap(df.corr(), annot = True)
plt.show()
```



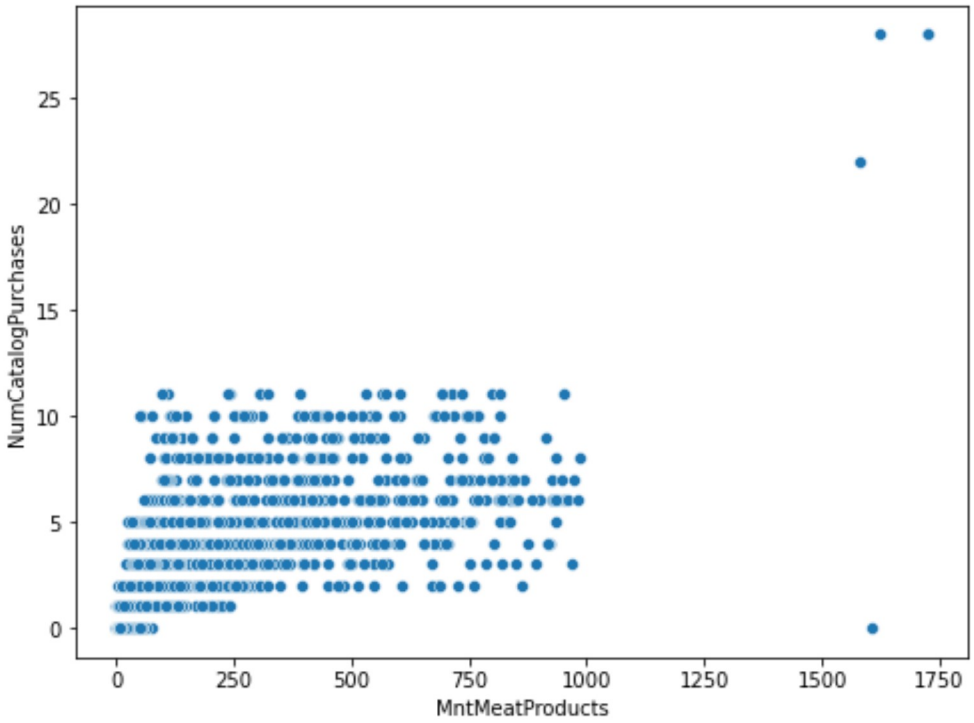
Observations:

- People are likely to buy meat and wine on the same visit
- People are also likely to buy fruit and sweets on the same visit.
- There is a high correlation between catalog shopping and meat and wine purchases. If the store is advertising meats and wines in a catalong, that's working.
- In addition, the store's website seems to be hurting them. There is a high negative correlation between number of web visits and purchases of fruits, meats, fish, sweets, catalog purchases and store purchases.

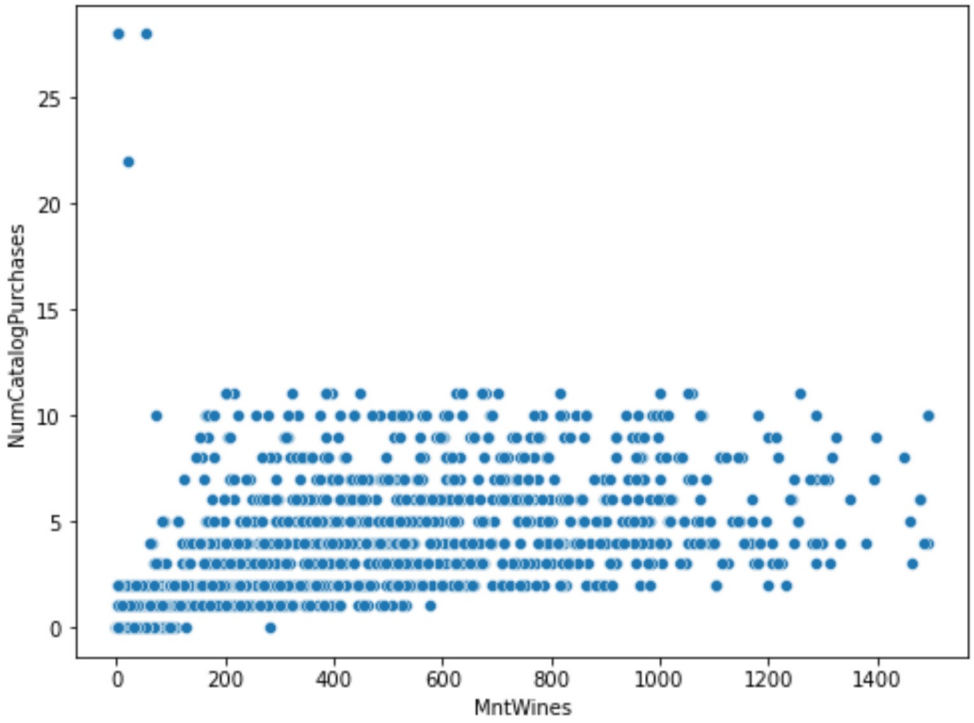
- There is a weak correlation between the website visits and purchases of deals. This shows that their online shoppers are probably looking for the cheapest products.

Comparing Variables with a High Positive Correlation

In [28]:
```
plt.figure(figsize=[8, 6])
sns.scatterplot(x=df.MntMeatProducts,y=df.NumCatalogPurchases)
plt.show()
```

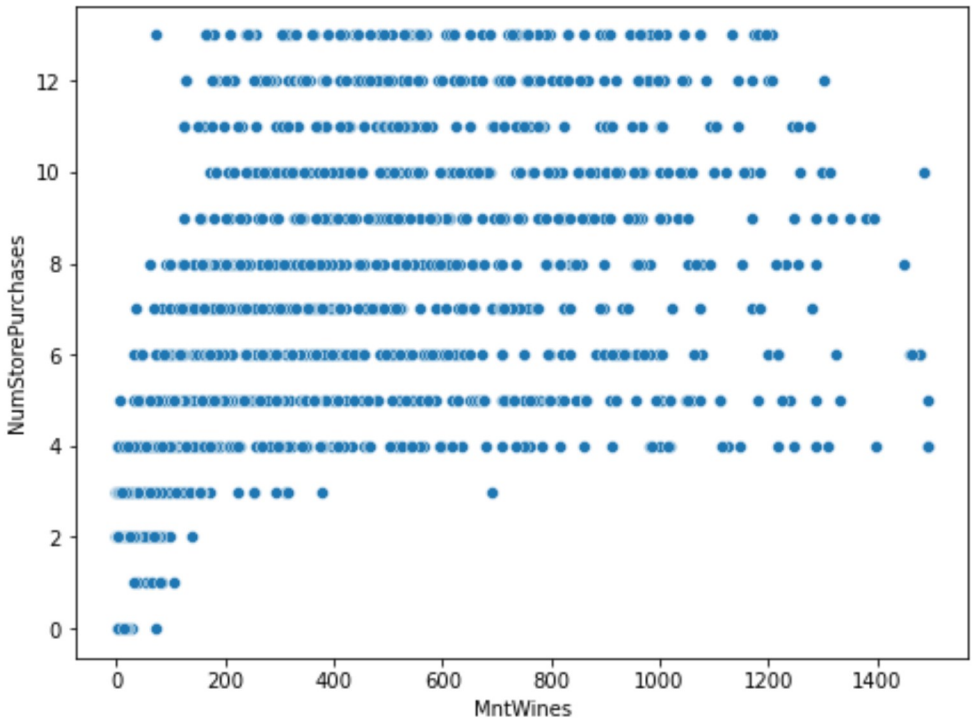


In [29]:
```
plt.figure(figsize=[8, 6])
sns.scatterplot(x=df.MntWines,y=df.NumCatalogPurchases)
plt.show()
```



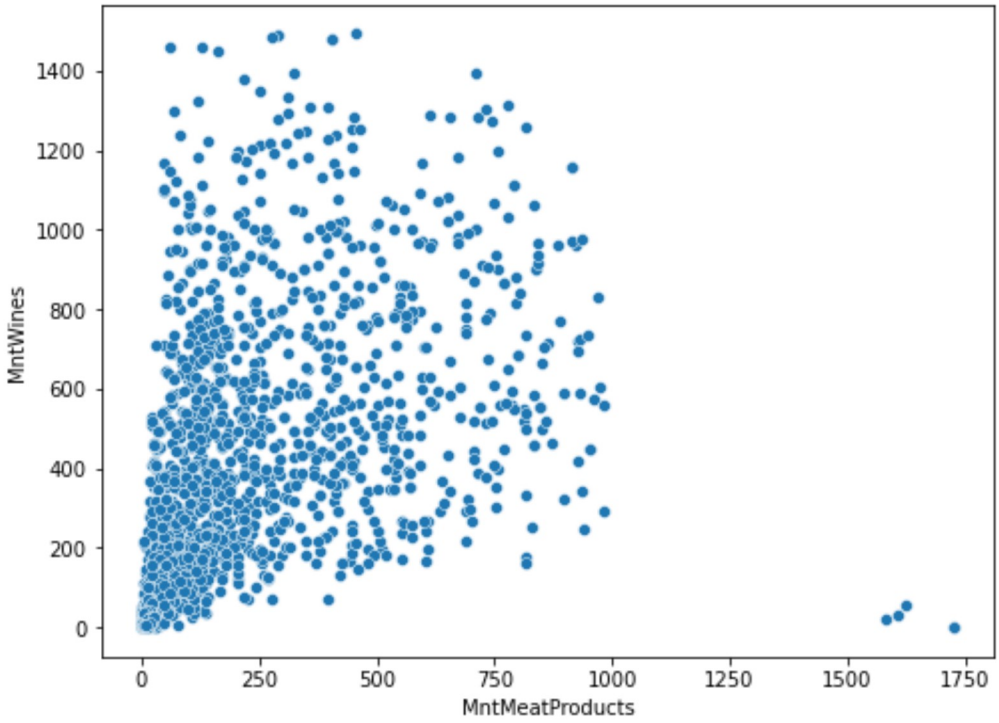
In [30]:

```
plt.figure(figsize=[8, 6])
sns.scatterplot(x=df.MntWines,y=df.NumStorePurchases)
plt.show()
```

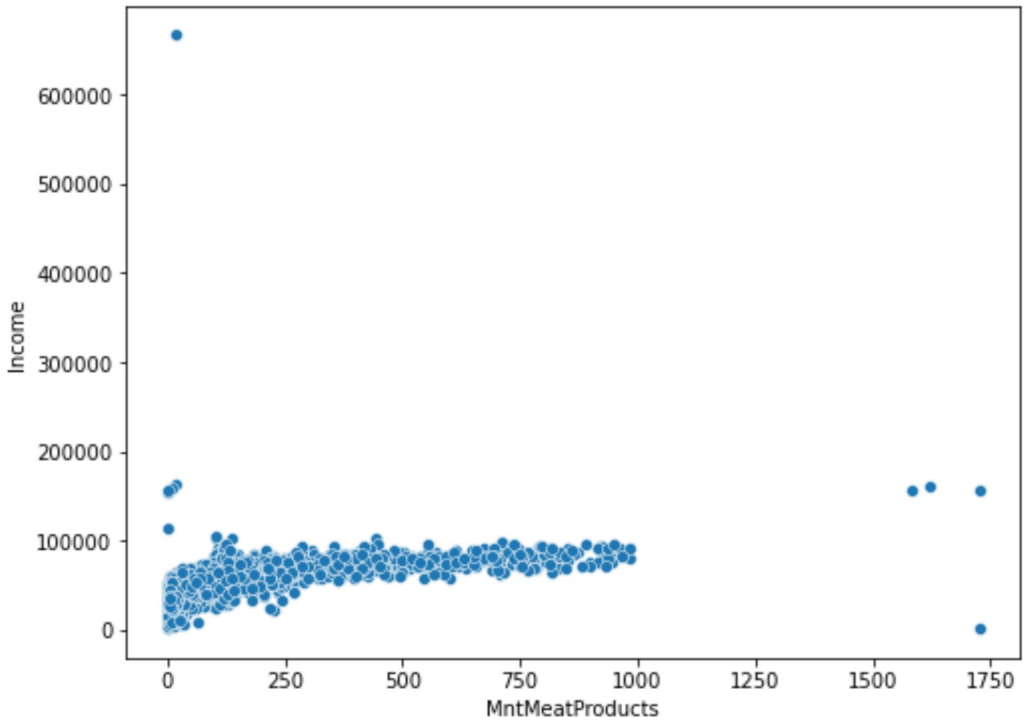


In [31]:

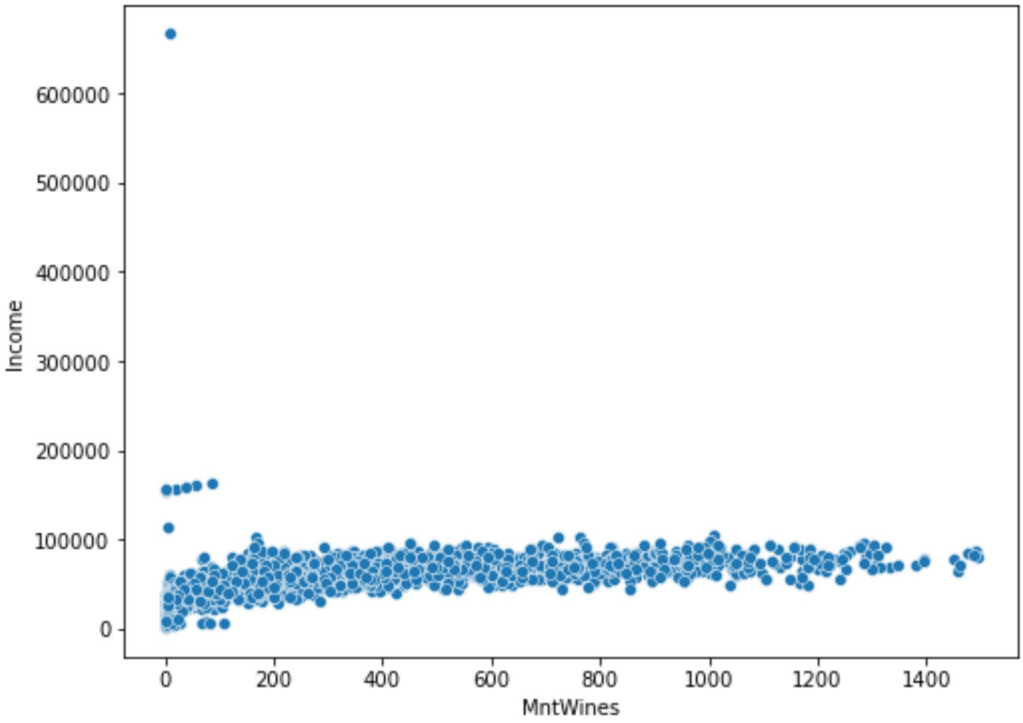
```
plt.figure(figsize=[8, 6])
sns.scatterplot(x=df.MntMeatProducts,y=df.MntWines)
plt.show()
```



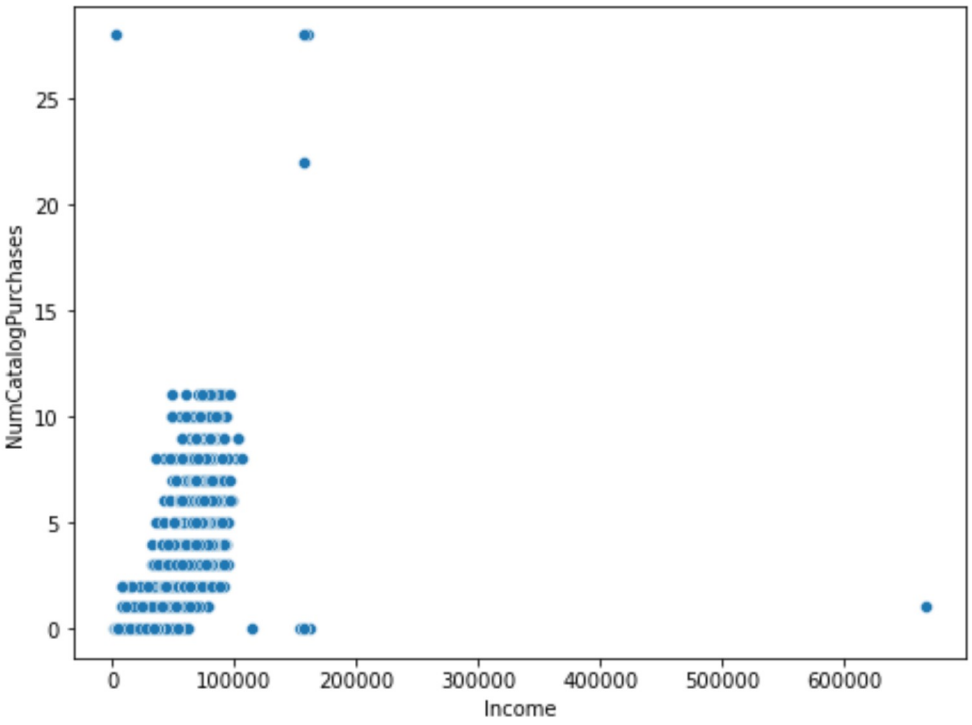
In [32]:
```
plt.figure(figsize=[8, 6])
sns.scatterplot(x=df.MntMeatProducts,y=df.Income)
plt.show()
```



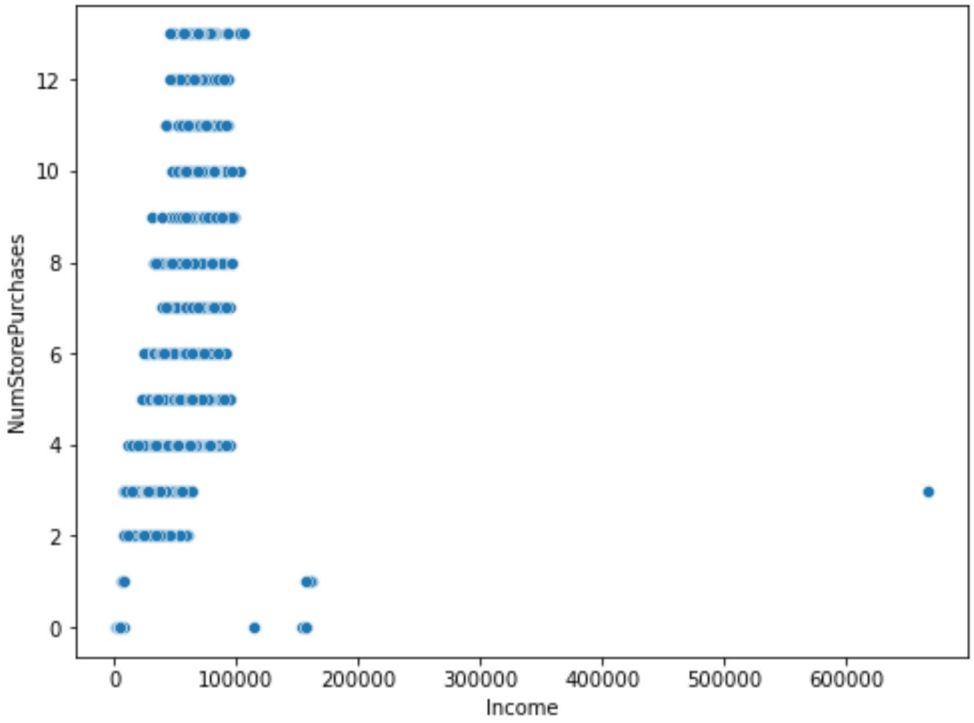
In [33]:
```
plt.figure(figsize=[8, 6])
sns.scatterplot(x=df.MntWines,y=df.Income)
plt.show()
```



In [34]:
```
plt.figure(figsize=[8, 6])
sns.scatterplot(x=df.Income,y=df.NumCatalogPurchases)
plt.show()
```

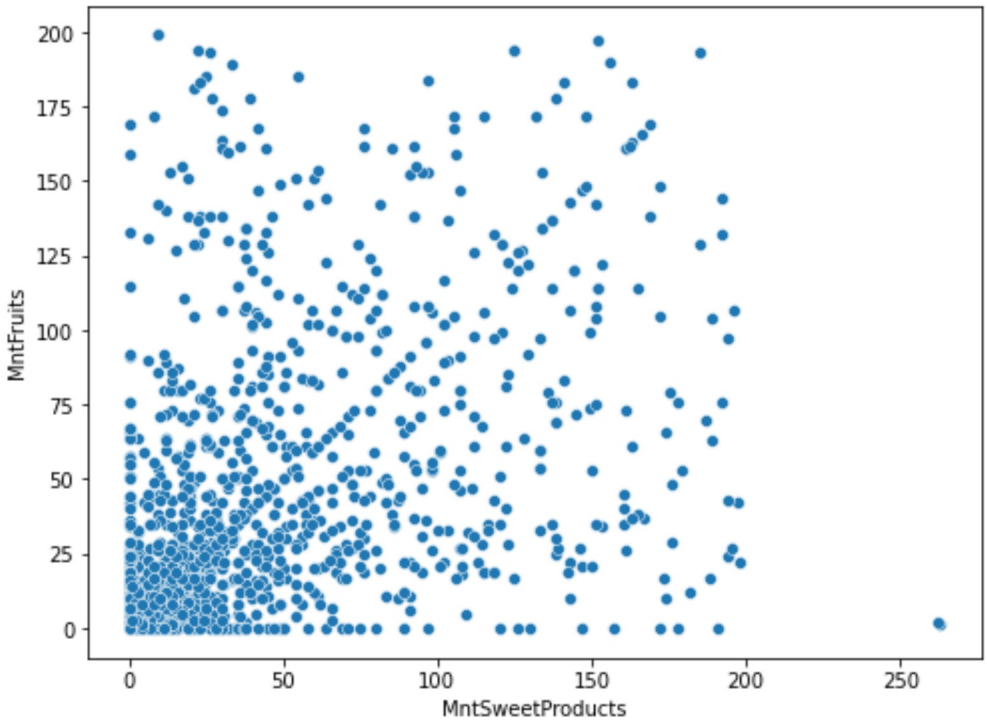


In [35]:
```
plt.figure(figsize=[8, 6])
sns.scatterplot(x=df.Income,y=df.NumStorePurchases)
plt.show()
```



In [36]:

```
plt.figure(figsize=[8, 6])
sns.scatterplot(x=df.MntSweetProducts,y=df.MntFruits)
plt.show()
```



Prepare Data for Algorithm

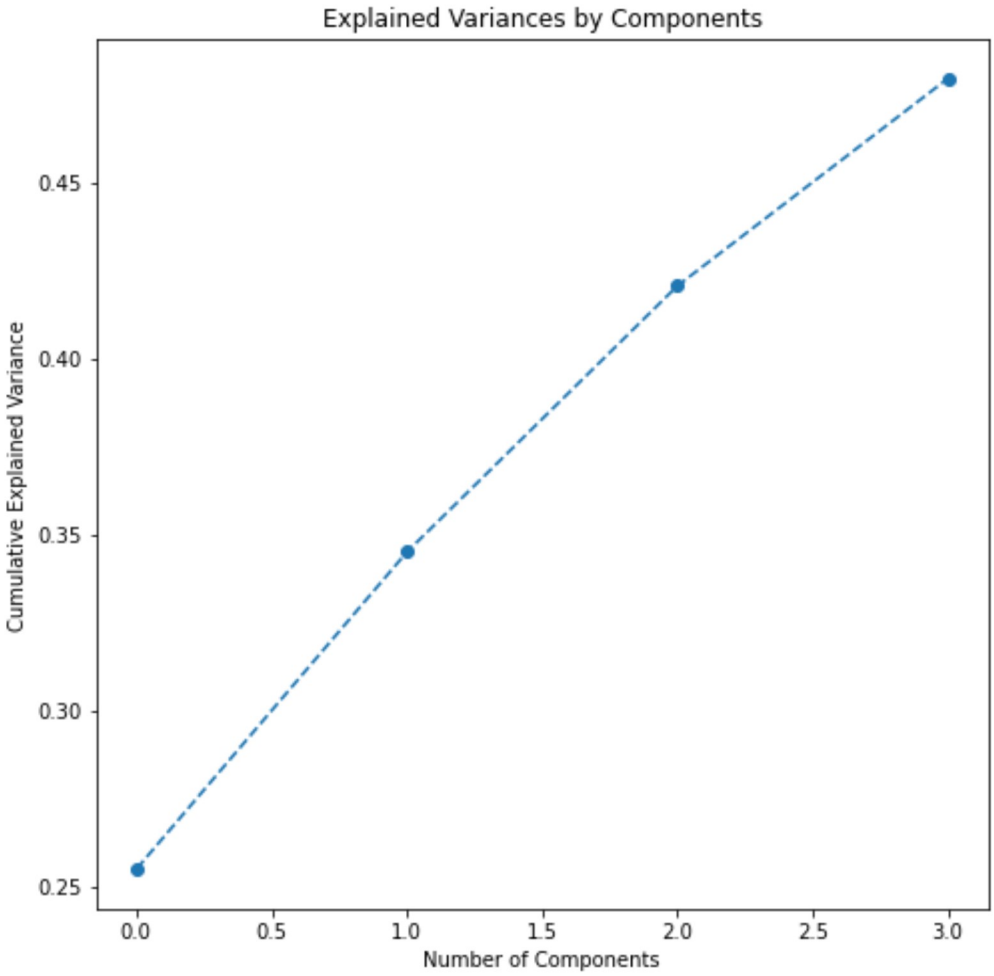
In [37]:
```
df2=df.drop(["Marital_Status", "Income", "Dt_Customer"],axis=1)
df2['Education']=df2['Education'].map({'2n Cycle':0,'Basic':1,'Graduation':2,'Master
```

In [38]:
```
X=df2[['Education']]
```

Principal Component Analysis

In [39]:
```
scaler=ss()
subset=df2.iloc[:, 3:].copy()
subset_scaled=scaler.fit_transform(subset)
subset_scaled_df=pd.DataFrame(subset_scaled, columns=subset.columns)

pca=PCA(n_components=4,random_state=1)

# Fit and transform the pca function on scaled data
df2_pca1=df2.iloc[:,0:23].values
df2_pca2=df2.iloc[:,23].values
xtrain,xtest,ytrain,ytest=train_test_split(df2_pca1,df2_pca2,test_size=0.2,random_sta
ss=ss()
xtrain=ss.fit_transform(xtrain)
xtest=ss.transform(xtest)
xtrain=pca.fit_transform(xtrain)
xtest=pca.transform(xtest)

# The percentage of variance explained by each principal component
exp_var=pca.explained_variance_ratio_
```

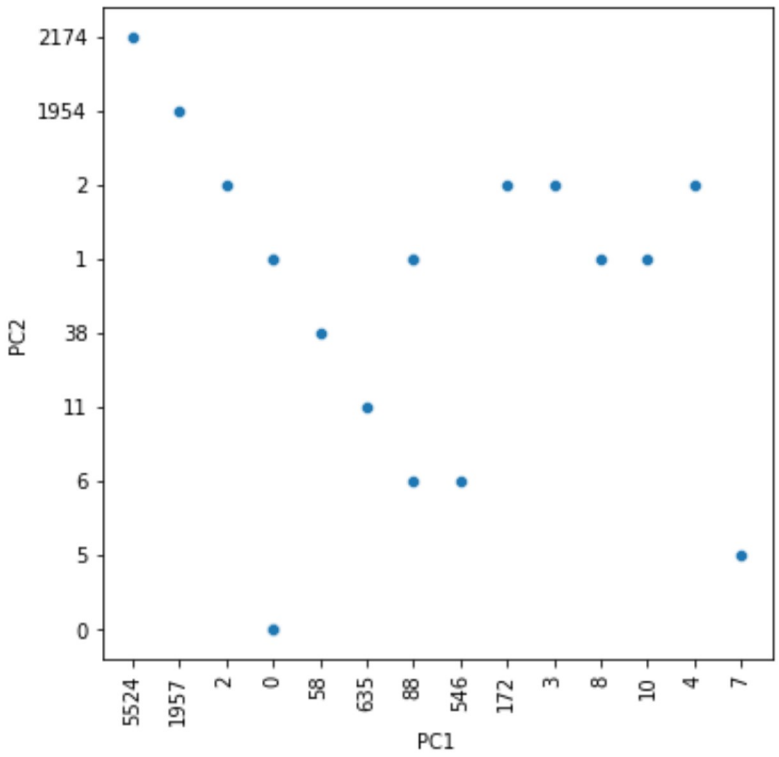
In [40]:
```
plt.figure(figsize=(8,8))
plt.plot(range(0,4),exp_var.cumsum(),marker='o',linestyle='--')
plt.title("Explained Variances by Components")
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.show()
```



In [41]:
```
print(pca.explained_variance_ratio_)
```

```
[0.25489781 0.09013492 0.07565379 0.05896879]
```

In [42]:
```
plt.figure(figsize=(6, 6))
sns.scatterplot(x=df2_pca1[0].astype(str), y=df2_pca1[1].astype(str))
plt.xlabel("PC1")
plt.ylabel("PC2")
locs, labels = plt.xticks()
plt.setp(labels, rotation=90)
plt.show()
```



Observations

- The scree plot shows that PC 1 and PC 2 only contain 34% of the data.
- Therefore, PCA will not work on this data because too much is lost if it is flattened into 2 dimensions.

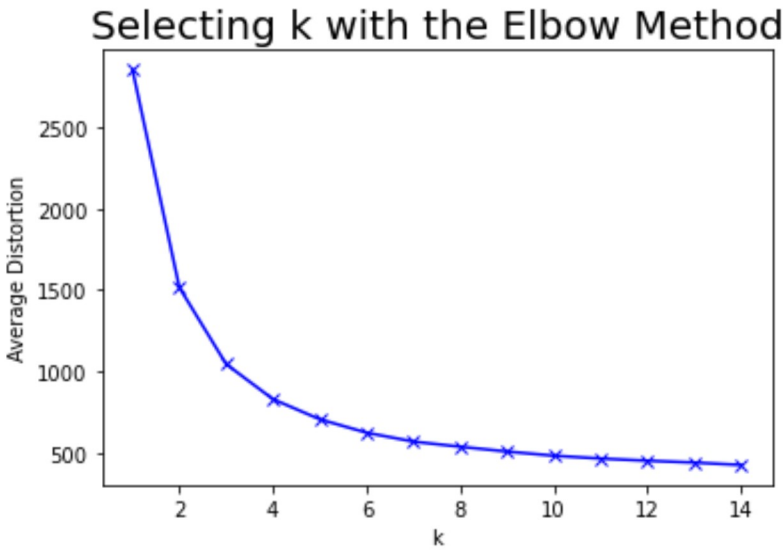
K Means Testing

In [43]:
```
k_means_df=pd.DataFrame(df2_pca1.copy())
```

In [44]:

```
clusters=range(1,15)
meanDistortions=[]

for k in clusters:
    model=KMeans(n_clusters=k,random_state=1)
    model.fit(df2_pca1)
    prediction=model.predict(k_means_df)
    distortion=(
        sum(np.min(cdist(k_means_df,model.cluster_centers_,"euclidean"),axis=1))
        /k_means_df.shape[0]
    )

    meanDistortions.append(distortion)

    print("Number of Clusters:", k, "\tAverage Distortion:", distortion)

plt.plot(clusters, meanDistortions, "bx-")
plt.xlabel("k")
plt.ylabel("Average Distortion")
plt.title("Selecting k with the Elbow Method", fontsize = 20)
plt.show()
```

```
Number of Clusters: 1   Average Distortion: 2856.1896474741325
Number of Clusters: 2   Average Distortion: 1516.5628673030299
Number of Clusters: 3   Average Distortion: 1045.0136669616297
Number of Clusters: 4   Average Distortion: 829.5470501180541
Number of Clusters: 5   Average Distortion: 705.7455721738778
Number of Clusters: 6   Average Distortion: 623.2481545064891
Number of Clusters: 7   Average Distortion: 568.5852155625796
Number of Clusters: 8   Average Distortion: 536.8686151327623
Number of Clusters: 9   Average Distortion: 507.278533664838
Number of Clusters: 10  Average Distortion: 481.79223249114875
Number of Clusters: 11  Average Distortion: 464.2575589786305
Number of Clusters: 12  Average Distortion: 450.85992227524
Number of Clusters: 13  Average Distortion: 439.52380605824203
Number of Clusters: 14  Average Distortion: 424.5994711184819
```



In [45]:
```
kmeans = KMeans(n_clusters = 3, random_state = 1)
kmeans.fit(k_means_df)
```

Out[45]:
```
KMeans(n_clusters=3, random_state=1)
```

In [46]:
```
# Creating a copy of the original data from pca1
df3=df2.copy()

# Adding K-Means cluster labels to the KMeans and original dataframes
k_means_df=k_means_df.assign(KM_segments=kmeans.labels_)
df3["KM_segments"]=kmeans.labels_
```

Cluster Profiles

In [47]:
```
km_cluster_profile=df3.groupby("KM_segments").mean()
```

In [48]:
```
# Creating the "count_in_each_segment" feature in K-Means cluster profile

km_cluster_profile["count_in_each_segment"] = (
    df3.groupby("KM_segments")["MntMeatProducts"].count().values
)
```

In [49]:
```
km_cluster_profile.style.highlight_max(color="lightgreen", axis=0)
```

Out[49]:

KM_segments	ID	Year_Birth	Education	Kidhome	Teenhome	Recency	MntWines	Mn
0	5654.113665	1968.203065	2.429119	0.458493	0.499361	50.220945	309.839080	25.4
1	1906.018494	1969.211361	2.383091	0.439894	0.511229	49.990753	311.075297	26.7
2	9509.158571	1969.041429	2.365714	0.432857	0.508571	46.912857	289.611429	26.6

In [50]:
```
# Let's see the names of the elements in each cluster
for cl in df3["KM_segments"].unique():
    print("In cluster {}, the following elements are present:".format(cl))
    print(df3[df3["KM_segments"] == cl]["MntWines"].unique())
    print()
```

```
In cluster 0, the following elements are present:
[ 635  426   11  173  520   76   14   28   84 1012    1  384    5    8
  110   96  702   81   13   48  826  650  984  712  546  398  247  258
    2 1032   25  112   17  292  445  505  207  386  144  425 1332   12
   34  722  220    0   73   42  295  888   18  277    7   35  356  783
  466   10  656  344  448  613  224 1076  897  131  297  231  983  789
    3   24  167   38   16    4  248   39  777  395  801   36  513  675
  509  768  792   21   22  797  378  561  629  269  736  333  833   43
  314  571  140 1200   85  483  966  822   19 1285  235  438   99   77
  141  507  164  606  143  639  451   26  738  502  504 1047  815  493
  325  238 1168  227 1379  881 1000  456  464  252 1478   15    6  189
   98  145  203  270    9   37   46  264   88   55  162  390 1001   41
```

```
  129  387  586  163 1009  721  757   30  410  100  111 1004 1132  565
  260  459   67  909   23  537  670  524   54  432  138  948  307  769
  518  359  799  895  245  647  587  107  680  324  521   31  479  411
  161   68  642 1492  752  191  563  362  724  547  556  408  899  889
  205  316  756  194  243  151   29  604  960  315  471  517  267 1111
   79   20  115   97  186  230  729  674  284 1394  393  223  157 1230
   45  986   61  380   80  463 1315  347  192  631 1099  473 1050  918
  652  336  105  574  614  211   53  391 1224   74  165  957  176  454
  213  457  241  210 1156  108   27  964  572  341  309  172  919  618
  595  293  982 1142  489  420  113  545  266 1074  771  371  209  171
  774  262  135   86  488  689  622  201  204  279  972  714   56  703
  458   91  443  901  846   72  169  406  123  758  216  331  910 1171
   33   51   71  928 1006  233  255  158  199  739 1083  342  365  313
  109  298  527  593 1181  308  494  621  612  861  533  560  312  178
   40   64   93 1073  402  686  383  536  128  581  584  641 1020  272
  829 1017  953 1193  340  154  710  338  605  499  381  913  673   63
   58   62   83 1276  239  182  106  753  555  240  848  202  433  367
  958  375  532  835  229  529  816 1311  654  496  576 1013  570  254
  116  611  671  580  368  908]

In cluster 1, the following elements are present:
[  11  235    5    6  194    3 1006    4  867  270  173  684  112  437
   13  123   12  510  452  523   10  856  496   14  721  577   64  163
  130  135   27   53   41  295   76  527  490  547   47  162   70  534
  231  796   17  962  614  212  536   15   85   81   59    1   23   96
  763   22   54 1170   72  410  239  171  182  824   30  377  787  792
  562   18  901 1218   45  918  138  554  177  483  778  167  625  216
  196  204 1241   80  154   25  861  185   24   44  738  412  424    9
   40  755  189  495  656   78   19  789    7  977   38 1043   73  375
  102  919    2   65  416  583 1076  551  378 1248  515    0  398  208
  430  116  522  563  746  478   20  213  317  529  388   34  284 1288
 1035  982   32    8  141  206  619  308 1492  249  200 1050  399   31
  526  215  352  121  965  707 1063  305 1200  125  219  111  509  275
  595   55  466   67  434  712  664  299  345  294  349   43  794   39
   26  229  584  627  330   28 1039  623  393  464  860  421   91  104
  265  374  600  243   48  379  387  852   35  181   63  156  372  871
  354 1253  371  588 1166  261  711  263   69  492 1206  292  587   66
  384  350   68  403  538  743  124  530 1215  444  251  172  152  938
  120  760  199  516  709   88  508  381  146  110  815  717 1067  320
   46  735  342  258  356  441  364  184  373 1148  816   87   57  140
  291  544  972  157  279  408   51   62  952   56  303   95  133  151
  736  737  254  817  462  283   52  558  183  779   97  139 1099  714
  823  557  368  543  347  635  422   33  129   16   29  966  153  460
  556  751  267 1259  896  244  539 1181  750  306   60  675  332  784
  674 1000  293   50  532  411  268  733  255  754  202  322 1016  100
  158   86  159  629 1459  340 1003  446 1184  390  726  690  667  691
  493  582 1060  979  704  505  136 1493  166  358   21   71  376  453
  777  362  367  198  941  389 1121  174  666  407  488 1045  175  864
  957  273  122  637 1324  353  652  217  256  448 1462  631  879  280
   90  575  315  561   94  908  228  995  676  450  264  520  178 1486
 1115  934  830  728  188  997  890  224  531  626 1074  741  519  297
  366  269  302  594]

In cluster 2, the following elements are present:
[ 233   53   86    6   36  482   40   55  421    9    3  245  328  295
   11  447    1  688   25  565  350  230    7   15   51  562  217  454
  378  320  450  352  492  120  236  738  399   32  598    2  346  997
    8   42 1349 1001   20  125 1149  847   29  152  620   47  519  265
  587  155    0  415  382   14  559  925  864  960   16  168  650   77
```

```
  33   68  297   84   26  304  285  395   65   37  525   91  731  909
 317  856  456 1103   30  452  546 1048  512  404  159   57  866  594
 554  610   67  370  407  604  800   31  357  400   10  398  161  112
1239    4  335  371  281  138  983  691 1396   13   28  225  127  179
 172   99   63  198  416  458   90  557  465  129  174  747  117   56
1060  817    5  101  389   27  625  215 1181  603  514  315  418  327
 187  365  116  372  899  760   92  273  321  212 1184  749  440  597
 292  412 1126  967  283  709  301  180  392  522  184  183   24   19
 606  376  840 1279  351  336  605  224  384  121 1308  515   22   48
 255  437  825  171  494  464  102  953  770  741 1478   38  812  375
 819  480  135  267  197  693  434  829  318 1023  311 1296  218  795
 244  158 1285  349  397  201   18  806  576   39   73   75  241  736
 288  234  162  160   70   12  166  714  229  897  658  476  863  277
 134 1298  221  684   35  132   62  790  882  615  313  630 1090  968
 556  202  170  491  173 1205   82  358   66  274  641  794  182  280
  95  992  100  642   88  329  296 1218 1302  508  290  386  944  526
 553 1449  247  931  811  423  252  104  431  448   46  483   34 1252
 368  548  836  238   98  595  269  980   50  205  493  532  387  293
 199  940  520   87  194 1073  275  303  708  324  347   71  920  653
 153   52  734  572  438  531  503  369   79  209  239  539  712  707
 823  284  185  355  279   45  505  344  145   69  261  111  228 1092
  85 1017  226  820  208  713  724  757 1245  124  471  189  742  750
  23  912  141  154   89  675  516  530  240  266  322  332  853  428]
```

In [51]:
```
df3.groupby(["KM_segments", "MntWines"])['MntMeatProducts'].count()
```

Out[51]:
```
KM_segments  MntWines
0            0           3
             1          12
             2          17
             3          11
             4           9
                        ..
2            1308        1
             1349        1
             1396        1
             1449        1
             1478        1
Name: MntMeatProducts, Length: 1228, dtype: int64
```

Check for Multicollinearity with Variance Inflation Factor (VIF)

In [52]:
```
xtrain=pd.DataFrame(xtrain)

# Function to check VIF
def checking_vif(train):
    vif=pd.DataFrame()
    vif["MntWines"]=train.columns

    # Calculating VIF for each feature
    vif["VIF"]=[
        variance_inflation_factor(train.values,i) for i in range(len(train.columns))
    ]
    return vif

print(checking_vif(xtrain))
```

```
   MntWines  VIF
0         0  1.0
1         1  1.0
2         2  1.0
3         3  1.0
```

In [53]:
```
xtrain=pd.DataFrame(xtrain)

# Function to check VIF
def checking_vif(train):
    vif=pd.DataFrame()
    vif["MntMeats"]=train.columns

    # Calculating VIF for each feature
    vif["VIF"]=[
        variance_inflation_factor(train.values,i) for i in range(len(train.columns))
    ]
    return vif

print(checking_vif(xtrain))
```

```
   MntMeats  VIF
0         0  1.0
1         1  1.0
2         2  1.0
3         3  1.0
```

Build Model

In [54]:

```
# Model Performance on test, train data
def model_pref(olsmodel,x_train,x_test,y_test,y_train):

    # Insample Prediction
    y_pred_train=olsmodel.predict(x_train)
    y_observed_train=y_train

    # Prediction on test data
    y_pred_test=olsmodel.predict(x_test)
    y_observed_test=y_test

    print(
        pd.DataFrame(
            {
                "Data": ["Train", "Test"],
                "RMSE": [
                    np.sqrt(mean_squared_error(y_pred_train, y_observed_train)),
                    np.sqrt(mean_squared_error(y_pred_test, y_observed_test)),
                ],
                "MAE": [
                    mean_absolute_error(y_pred_train, y_observed_train),
                    mean_absolute_error(y_pred_test, y_observed_test),
                ],

                "r2": [
                    r2_score(y_pred_train, y_observed_train),
                    r2_score(y_pred_test, y_observed_test),
                ],
            }
        )
    )
```

In [55]:

```
# Create the model
model1=sm.OLS(ytrain, xtrain).fit()
model1.summary()
```

Out[55]:

OLS Regression Results

Dep. Variable:	y	**R-squared (uncentered):**	0.156
Model:	OLS	**Adj. R-squared (uncentered):**	0.154
Method:	Least Squares	**F-statistic:**	82.44
Date:	Sat, 29 Apr 2023	**Prob (F-statistic):**	2.68e-64
Time:	01:39:22	**Log-Likelihood:**	-681.87
No. Observations:	1792	**AIC:**	1372.
Df Residuals:	1788	**BIC:**	1394.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**

0	0.0363	0.003	10.506	0.000	0.030	0.043
1	-0.0021	0.006	-0.355	0.723	-0.013	0.009
2	0.0762	0.006	12.003	0.000	0.064	0.089
3	0.0623	0.007	8.672	0.000	0.048	0.076

Omnibus:	545.962	**Durbin-Watson:**	1.628
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1290.552
Skew:	1.697	**Prob(JB):**	5.76e-281
Kurtosis:	5.400	**Cond. No.**	2.08

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.