EDA example - Marketing analytics

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

We'll work with a small marketing analytics dataset, taken from iFood

Based on the example here

In [2]:
```
url = 'https://raw.githubusercontent.com/nlihin/data-analytics/main/datasets/marketing_data.csv'
mrkt_df = pd.read_csv(url)
mrkt_df.head()
```

Out[2]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	NumStorePurchases
0	1826	1970	Graduation	Divorced	$84,835.00	0	0	6/16/14	0	189	...	6
1	1	1961	Graduation	Single	$57,091.00	0	0	6/15/14	0	464	...	7
2	10476	1958	Graduation	Married	$67,267.00	0	1	5/13/14	0	134	...	5
3	1386	1967	Graduation	Together	$32,474.00	1	1	5/11/14	0	10	...	2
4	5371	1989	Graduation	Single	$21,474.00	1	0	4/8/14	0	6	...	2

5 rows × 28 columns

Data description in an image, and also here

---

1. Wrangling data

1.1 Fix columns

You may already notice something is strange with the Income column, is it aligned to the right??

Let's look at the types of data

In [3]:
```
mrkt_df.dtypes
```

Out[3]:
```
ID                      int64
Year_Birth              int64
Education              object
Marital_Status         object
 Income                object
Kidhome                 int64
Teenhome                int64
Dt_Customer            object
Recency                 int64
MntWines                int64
MntFruits               int64
MntMeatProducts         int64
MntFishProducts         int64
MntSweetProducts        int64
MntGoldProds            int64
NumDealsPurchases       int64
NumWebPurchases         int64
NumCatalogPurchases     int64
NumStorePurchases       int64
NumWebVisitsMonth       int64
AcceptedCmp3            int64
AcceptedCmp4            int64
AcceptedCmp5            int64
AcceptedCmp1            int64
AcceptedCmp2            int64
Response                int64
Complain                int64
Country                object
dtype: object
```

So here is the problem: the 'Income' column contains extra whitespace, clean it:

In [4]:
```
mrkt_df.columns = mrkt_df.columns.str.replace(' ', '')
```

And: the 'Income' column should be turned to numeric (float is better)

In [5]:
```
# transform Income column to a numerical
mrkt_df['Income'] = mrkt_df['Income'].str.replace('$', '')
mrkt_df['Income'] = mrkt_df['Income'].str.replace(',', '').astype('float')
```

Check that our changes worked:

In [6]:
```
mrkt_df.dtypes['Income']
```

Out[6]:
```
dtype('float64')
```

Change the date column to a date type

In [7]:
```
mrkt_df['Dt_Customer'] = pd.to_datetime(mrkt_df['Dt_Customer'])
```

In [8]:
```
mrkt_df.dtypes
```

Out[8]:
```
ID                              int64
Year_Birth                      int64
Education                      object
Marital_Status                 object
Income                        float64
Kidhome                         int64
Teenhome                        int64
Dt_Customer            datetime64[ns]
Recency                         int64
MntWines                        int64
MntFruits                       int64
MntMeatProducts                 int64
MntFishProducts                 int64
MntSweetProducts                int64
MntGoldProds                    int64
NumDealsPurchases               int64
NumWebPurchases                 int64
NumCatalogPurchases             int64
NumStorePurchases               int64
NumWebVisitsMonth               int64
AcceptedCmp3                    int64
AcceptedCmp4                    int64
AcceptedCmp5                    int64
AcceptedCmp1                    int64
AcceptedCmp2                    int64
Response                        int64
Complain                        int64
Country                        object
dtype: object
```

1.2 Missing values

In [9]:
```
mrkt_df.isnull().sum().sort_values(ascending=False)
```

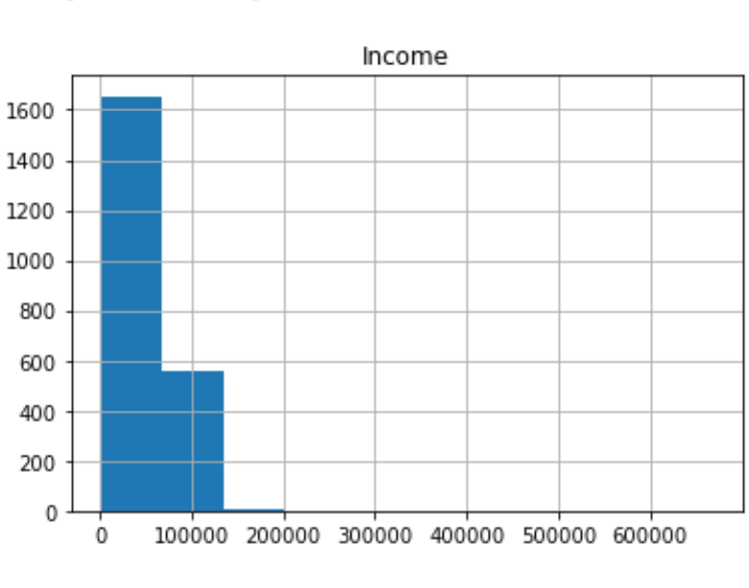
Out[9]:
```
Income                 24
Country                 0
Complain                0
Year_Birth              0
Education               0
Marital_Status          0
Kidhome                 0
Teenhome                0
Dt_Customer             0
Recency                 0
MntWines                0
MntFruits               0
MntMeatProducts         0
MntFishProducts         0
MntSweetProducts        0
MntGoldProds            0
NumDealsPurchases       0
NumWebPurchases         0
NumCatalogPurchases     0
NumStorePurchases       0
NumWebVisitsMonth       0
AcceptedCmp3            0
AcceptedCmp4            0
AcceptedCmp5            0
AcceptedCmp1            0
AcceptedCmp2            0
Response                0
ID                      0
dtype: int64
```

The feature Income contains 24 null values

Plot this feature to identify best strategy for imputation

In [10]:
```
mrkt_df.hist('Income')
```

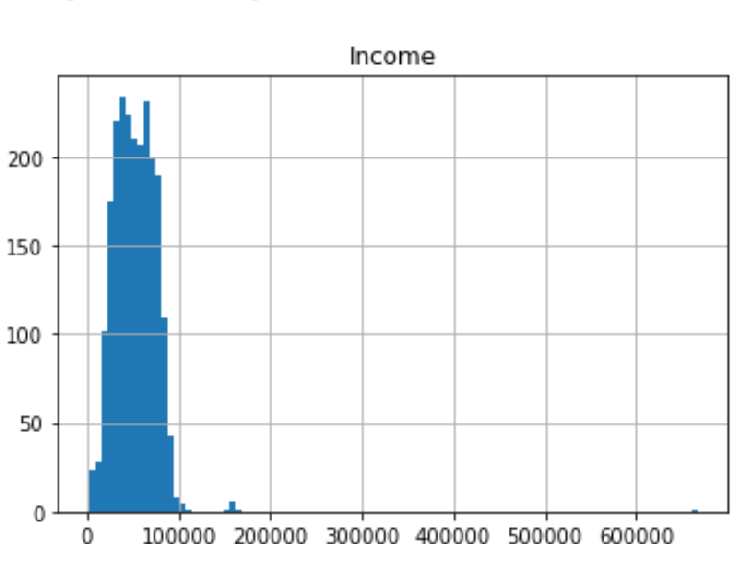
Out[10]:
```
array([[<AxesSubplot:title={'center':'Income'}>]], dtype=object)
```



Try more bins to get a better idea:

In [11]:
```
mrkt_df.hist('Income',bins=100)
```

Out[11]:
```
array([[<AxesSubplot:title={'center':'Income'}>]], dtype=object)
```

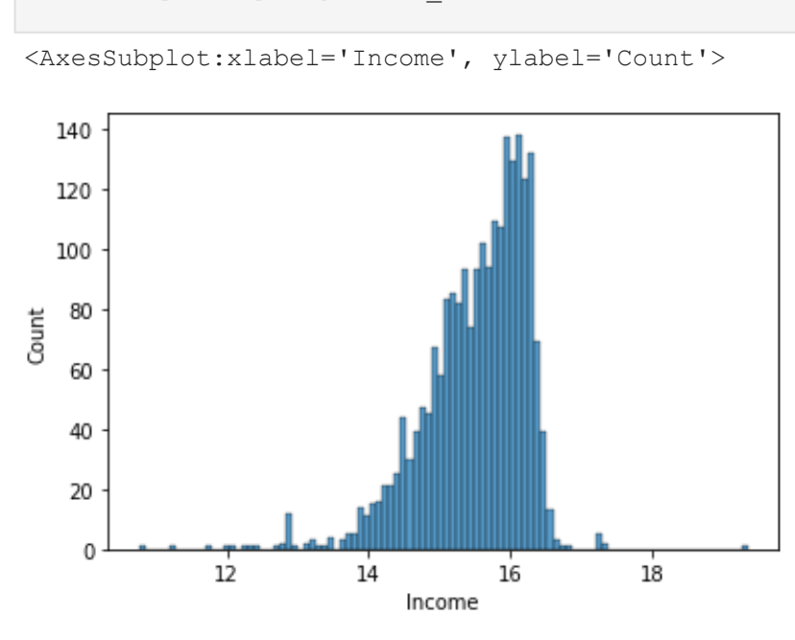


In [12]:
```
np.log(mrkt_df.Income)
```

Out[12]:
```
0       11.348463
1       10.952402
2       11.116425
3       10.388195
4        9.974598
          ...
2235    11.104596
2236    10.343547
2237    10.743113
2238    11.094664
2239    11.460273
Name: Income, Length: 2240, dtype: float64
```

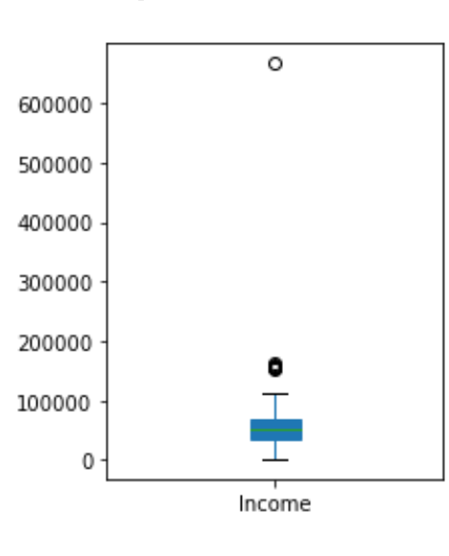
In [13]:
```
import seaborn as sns
sns.histplot(np.log2(mrkt_df.Income),bins=100)
```

Out[13]:
```
<AxesSubplot:xlabel='Income', ylabel='Count'>
```



In [14]:
```
mrkt_df['Income'].plot(kind='box', figsize=(3,4), patch_artist=True)
```

Out[14]:
```
<AxesSubplot:>
```



Findings:

Most incomes are distributed between $0-$100,000, with a few outliers

Replace null values with median value, to avoid effects of outliers on imputation value

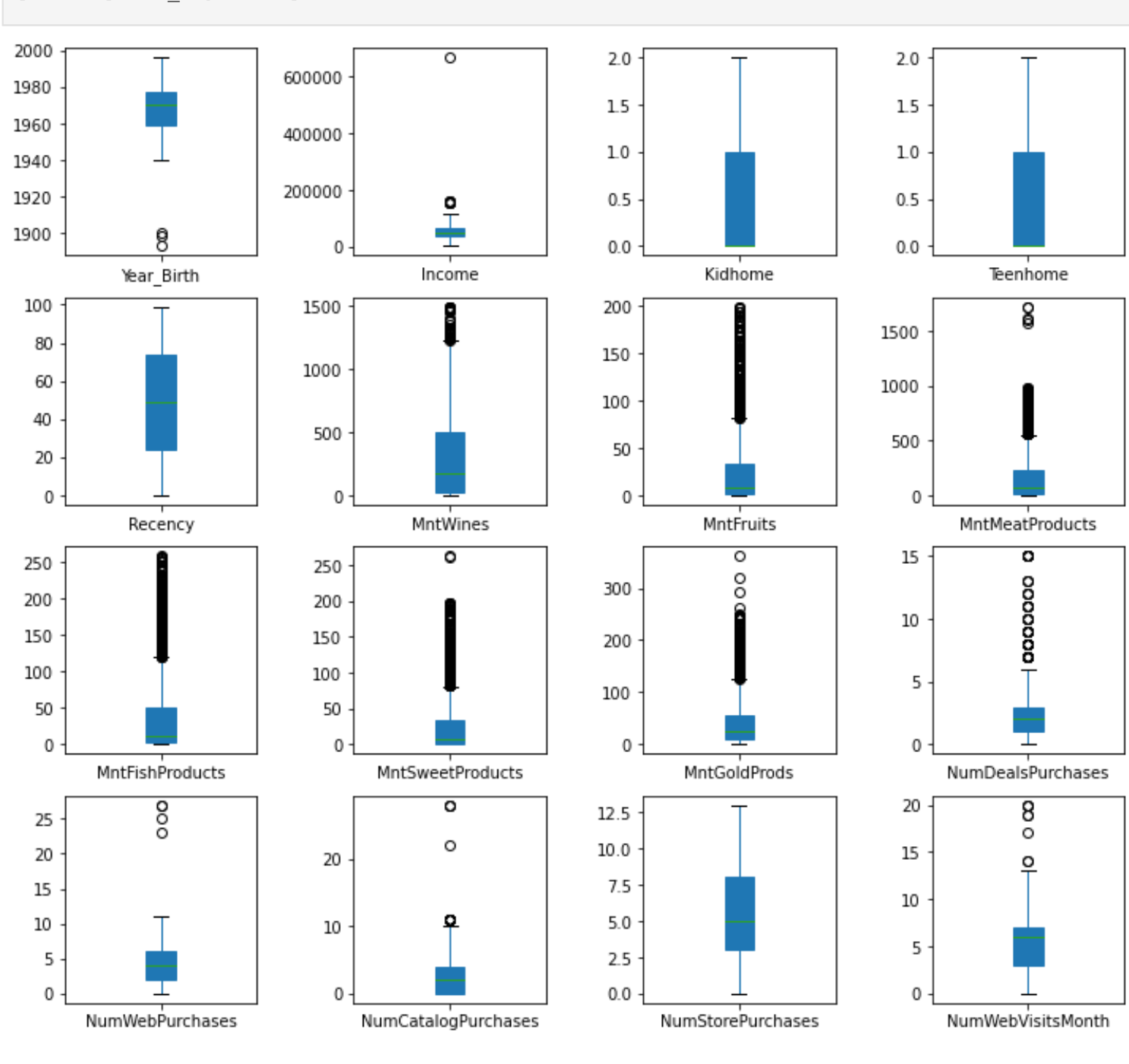
In [15]:
```
mrkt_df['Income'] = mrkt_df['Income'].fillna(mrkt_df['Income'].median())
```

1.3 Noisy data - outliers

Plot only numeric columns, that are not binary or an ID:

In [16]:
```
#remove binary and ID
df_to_plot = mrkt_df.drop(columns=['ID', 'AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'AcceptedCmp4', 'Accep

#select only numeric columns
df_to_plot = df_to_plot.select_dtypes(include=np.number)
```

In [17]:
```
# subplots
df_to_plot.plot(subplots=True, layout=(5,4), kind='box', figsize=(12,14), patch_artist=True)
plt.subplots_adjust(wspace=0.5);
```



Findings:

Multiple features contain outliers (see boxplots), but the only that likely indicate data entry errors are Year_Birth <= 1900

How many are there?

In [18]:
```
mrkt_df[mrkt_df['Year_Birth'] < 1901]
```

Out[18]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	NumStorePurchases
513	11004	1893	2n Cycle	Single	60182.0	0	1	2014-05-17	23	8	...	2
827	1150	1899	PhD	Together	83532.0	0	0	2013-09-26	36	755	...	4
2233	7829	1900	2n Cycle	Divorced	36640.0	1	0	2013-09-26	99	15	...	2

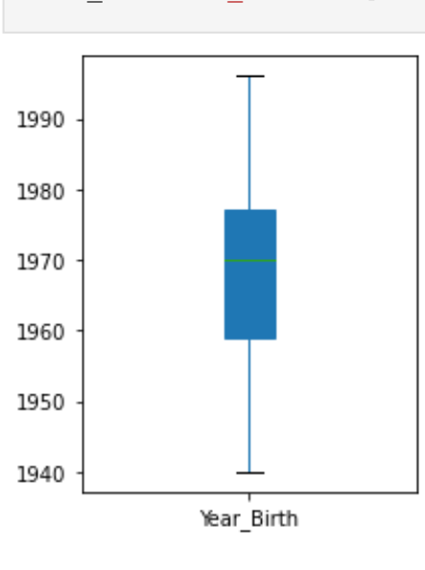
3 rows × 28 columns

Remove people born before 1900:

In [19]:
```
mrkt_df = mrkt_df[mrkt_df['Year_Birth'] > 1900].reset_index(drop=True)
```

Check (patch_artist fills the boxplot with color)

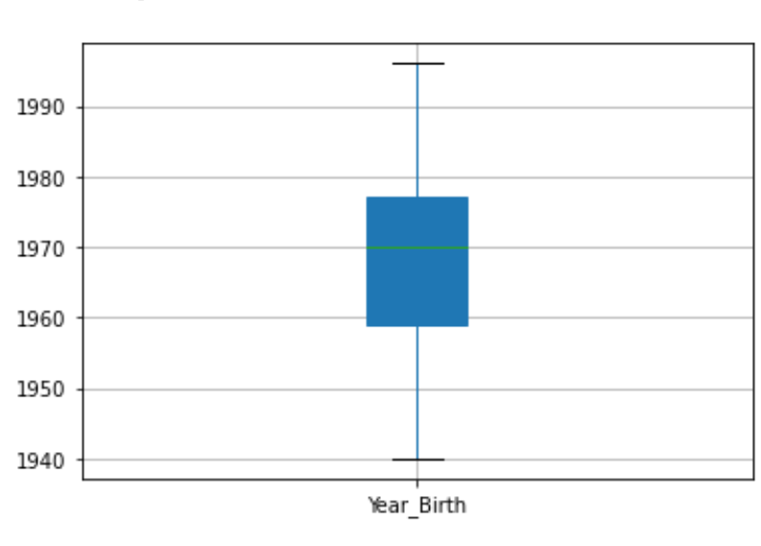
In [20]:
```
plt.figure(figsize=(3,4))
mrkt_df['Year_Birth'].plot(kind='box', patch_artist=True);
```



You can also plot via pandas:

In [21]:
```
mrkt_df.boxplot('Year_Birth', patch_artist=True)
```

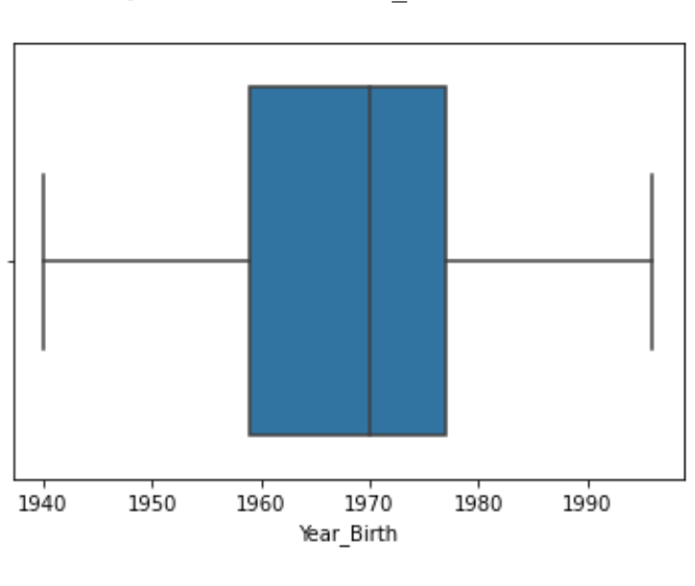
Out[21]:
```
<AxesSubplot:>
```



another option is to use seaborn.

In [22]:
```
sns.boxplot( x = 'Year_Birth', data = mrkt_df)
```

Out[22]:
```
<AxesSubplot:xlabel='Year_Birth'>
```



check what happens if you change x = 'Year_Birth' to y= 'Year_Birth'

1.4 Transformations

We already did this.. can do it here instead (if didn't do it before)

In [23]:
```
mrkt_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2237 entries, 0 to 2236
Data columns (total 28 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   2237 non-null   int64
 1   Year_Birth           2237 non-null   int64
 2   Education            2237 non-null   object
 3   Marital_Status       2237 non-null   object
 4   Income               2237 non-null   float64
 5   Kidhome              2237 non-null   int64
 6   Teenhome             2237 non-null   int64
 7   Dt_Customer          2237 non-null   datetime64[ns]
 8   Recency              2237 non-null   int64
 9   MntWines             2237 non-null   int64
 10  MntFruits            2237 non-null   int64
 11  MntMeatProducts      2237 non-null   int64
 12  MntFishProducts      2237 non-null   int64
 13  MntSweetProducts     2237 non-null   int64
 14  MntGoldProds         2237 non-null   int64
 15  NumDealsPurchases    2237 non-null   int64
 16  NumWebPurchases      2237 non-null   int64
 17  NumCatalogPurchases  2237 non-null   int64
 18  NumStorePurchases    2237 non-null   int64
 19  NumWebVisitsMonth    2237 non-null   int64
 20  AcceptedCmp3         2237 non-null   int64
 21  AcceptedCmp4         2237 non-null   int64
 22  AcceptedCmp5         2237 non-null   int64
 23  AcceptedCmp1         2237 non-null   int64
 24  AcceptedCmp2         2237 non-null   int64
 25  Response             2237 non-null   int64
 26  Complain             2237 non-null   int64
 27  Country              2237 non-null   object
dtypes: datetime64[ns](1), float64(1), int64(23), object(3)
memory usage: 489.5+ KB
```

Transform the date to a datetime object:

In [24]:
```
mrkt_df['Dt_Customer'] = pd.to_datetime(mrkt_df['Dt_Customer'])
```

2. Understand the data

The total number of dependents in the home ('Dependents') can be engineered from the sum of 'Kidhome' and 'Teenhome'

In [25]:
```
mrkt_df['Dependents'] = mrkt_df['Kidhome'] + mrkt_df['Teenhome']
```

The year of becoming a customer ('Year_Customer') can be engineered from 'Dt_Customer'

In [26]:
```
mrkt_df['Year_Customer'] = pd.DatetimeIndex(mrkt_df['Dt_Customer']).year
```

The total amount spent ('TotalMnt') can be engineered from the sum of all features containing the keyword 'Mnt'

We will use list comprehension to do that

A list comprehension consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses.

In [27]:
```
squares = []
for x in range(10):
    squares.append(x**2)
    squares
squares
```

Out[27]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [28]:
```
squares = [x**2 for x in range(10)]
squares
```

Out[28]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [29]:
```
mnt_cols = [col for col in mrkt_df.columns if 'Mnt' in col]
mnt_cols
```

Out[29]:
```
['MntWines',
 'MntFruits',
 'MntMeatProducts',
 'MntFishProducts',
 'MntSweetProducts',
 'MntGoldProds']
```

We have used `.sum()` to sum by columns. Now we want to sum rows. So `axis=1`

In [30]:
```
mrkt_df['TotalMnt'] = mrkt_df[mnt_cols].sum(axis=1)
```

The total purchases ('TotalPurchases') can be engineered from the sum of all features containing the keyword 'Purchases'

In [31]:
```
purchases_cols = [col for col in mrkt_df.columns if 'Purchases' in col]
mrkt_df['TotalPurchases'] = mrkt_df[purchases_cols].sum(axis=1)
```

The total number of campains accepted ('TotalCampaignsAcc') can be engineered from the sum of all features containing the keywords 'Cmp' and 'Response' (the latest campaign)

In [32]:
```
campaigns_cols = [col for col in mrkt_df.columns if 'Cmp' in col] + ['Response'] # 'Response' is for the latest
mrkt_df['TotalCampaignsAcc'] = mrkt_df[campaigns_cols].sum(axis=1)
```

Look at our new columns

In [33]:
```
mrkt_df[['ID', 'Dependents', 'Year_Customer', 'TotalMnt', 'TotalPurchases', 'TotalCampaignsAcc', 'NumDealsPurch
```

Out[33]:

	ID	Dependents	Year_Customer	TotalMnt	TotalPurchases	TotalCampaignsAcc	NumDealsPurchases
0	1826	0	2014	1190	15	1	1
1	1	0	2014	577	18	2	1
2	10476	1	2014	251	11	0	1
3	1386	2	2014	11	4	0	1
4	5371	1	2014	91	8	2	2
5	7348	0	2014	1192	17	1	1
6	4073	0	2014	1215	28	2	1
7	1991	1	2014	96	7	0	1
8	4047	1	2014	544	20	0	3
9	9477	1	2014	544	20	0	3

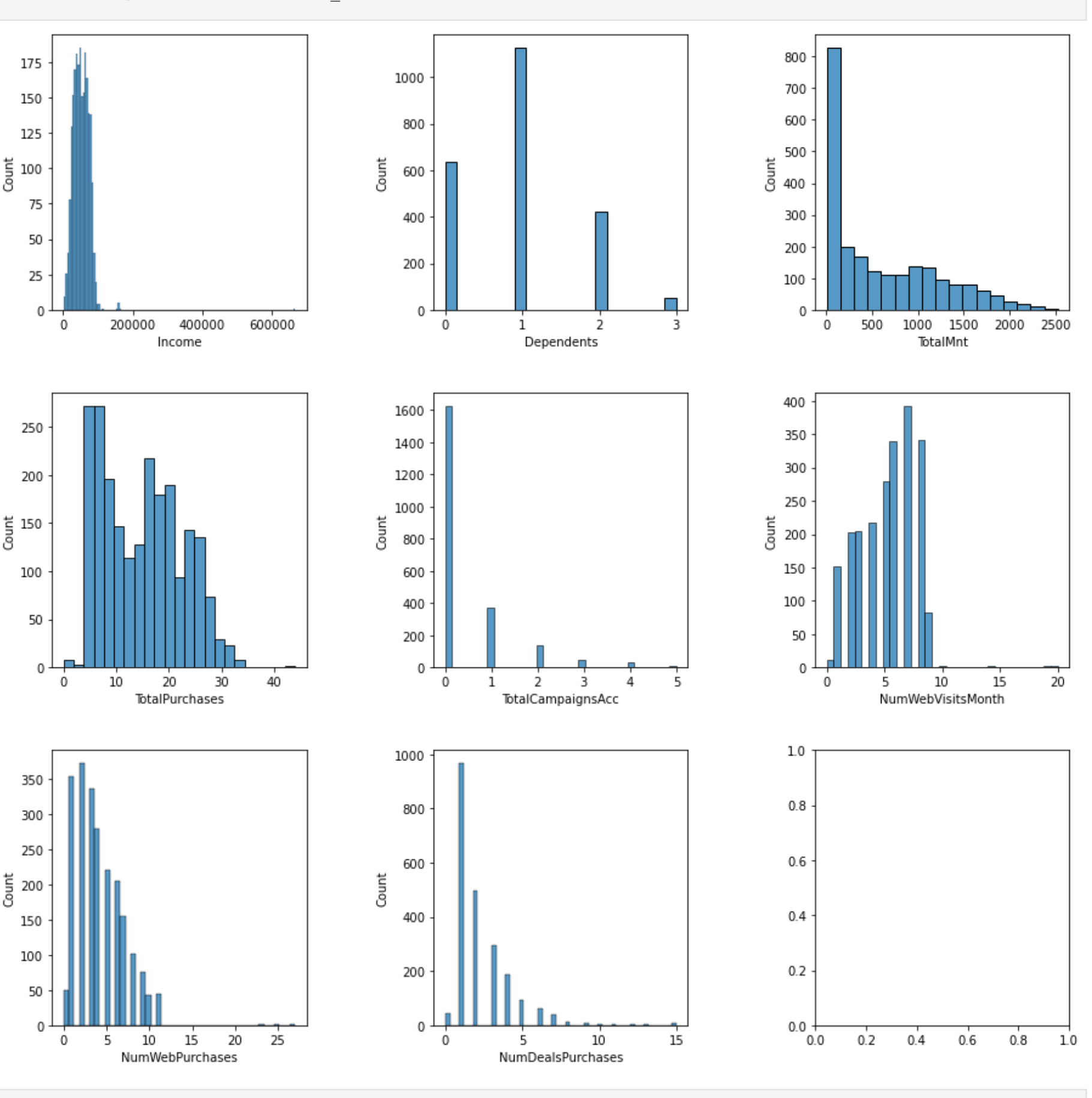
3. Correlations - patterns in the data

To find patterns in the data, we need to find correlations in the data

To understand what type of correlation, plot the histograms

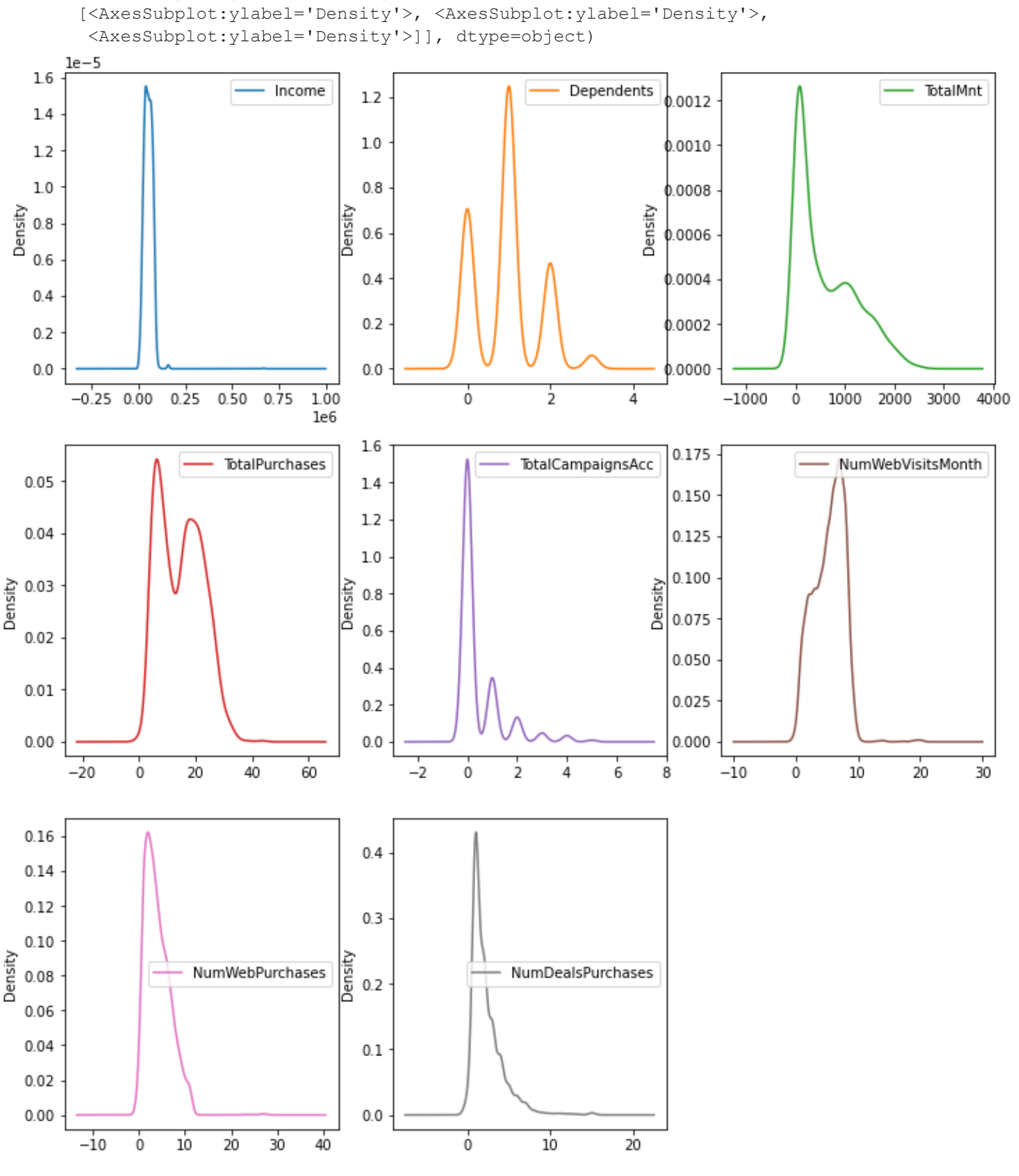
> Needed functions:
>
> * `plt.subplots(nrow,ncol,figsize=(x,y))` - creates a a grid for plots with nrow rows and ncol columns. The size is set by figsize. Returns a figure (fig) and a grid.
>
> * `plt.subplots_adjust(wspace, hspace)` - adjust width and height between plots
>
> # * `.flatten` - collapses the array (in this case, the array of subplots) into one dimension

In [34]:
```
features = ['Income', 'Dependents','TotalMnt','TotalPurchases', 'TotalCampaignsAcc', 'NumWebVisitsMonth', 'NumW
fig, axes = plt.subplots(3,3,figsize=(15,15))
plt.subplots_adjust(wspace=0.5, hspace = 0.3)
axes = axes.flatten()
for i,att in enumerate(features):
    sns.histplot(x=att, data=mrkt_df,  ax=axes[i])
```



In [35]:
```
mrkt_df[features].plot.density(subplots=True, sharex=False, sharey=False, layout=(3, 3), figsize=(12,14))
```

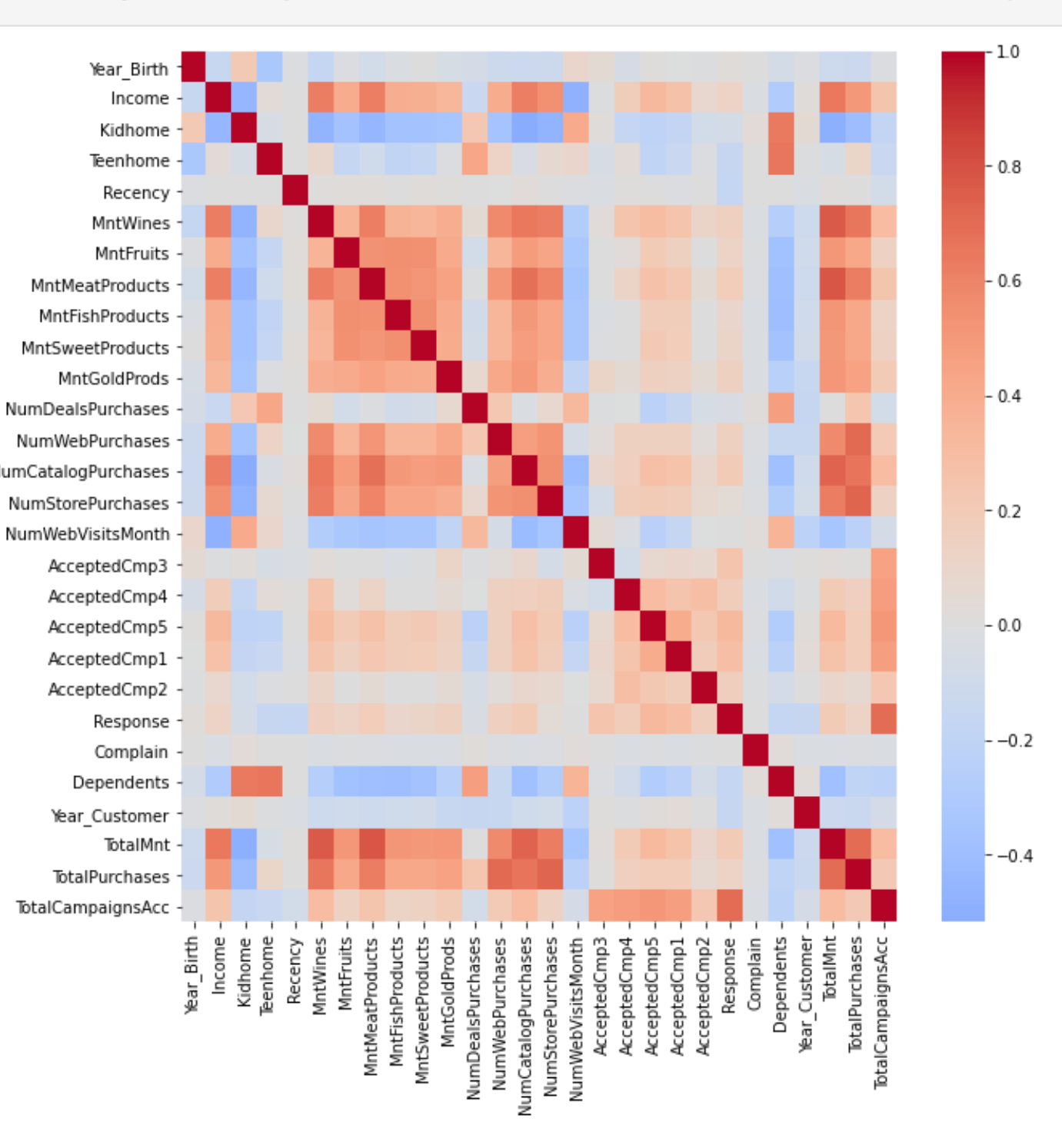
Out[35]:
```
array([[<AxesSubplot:ylabel='Density'>, <AxesSubplot:ylabel='Density'>,
        <AxesSubplot:ylabel='Density'>],
       [<AxesSubplot:ylabel='Density'>, <AxesSubplot:ylabel='Density'>,
        <AxesSubplot:ylabel='Density'>],
       [<AxesSubplot:ylabel='Density'>, <AxesSubplot:ylabel='Density'>,
        <AxesSubplot:ylabel='Density'>]], dtype=object)
```



This is not even close to a normal distribution, so use kendall-tau

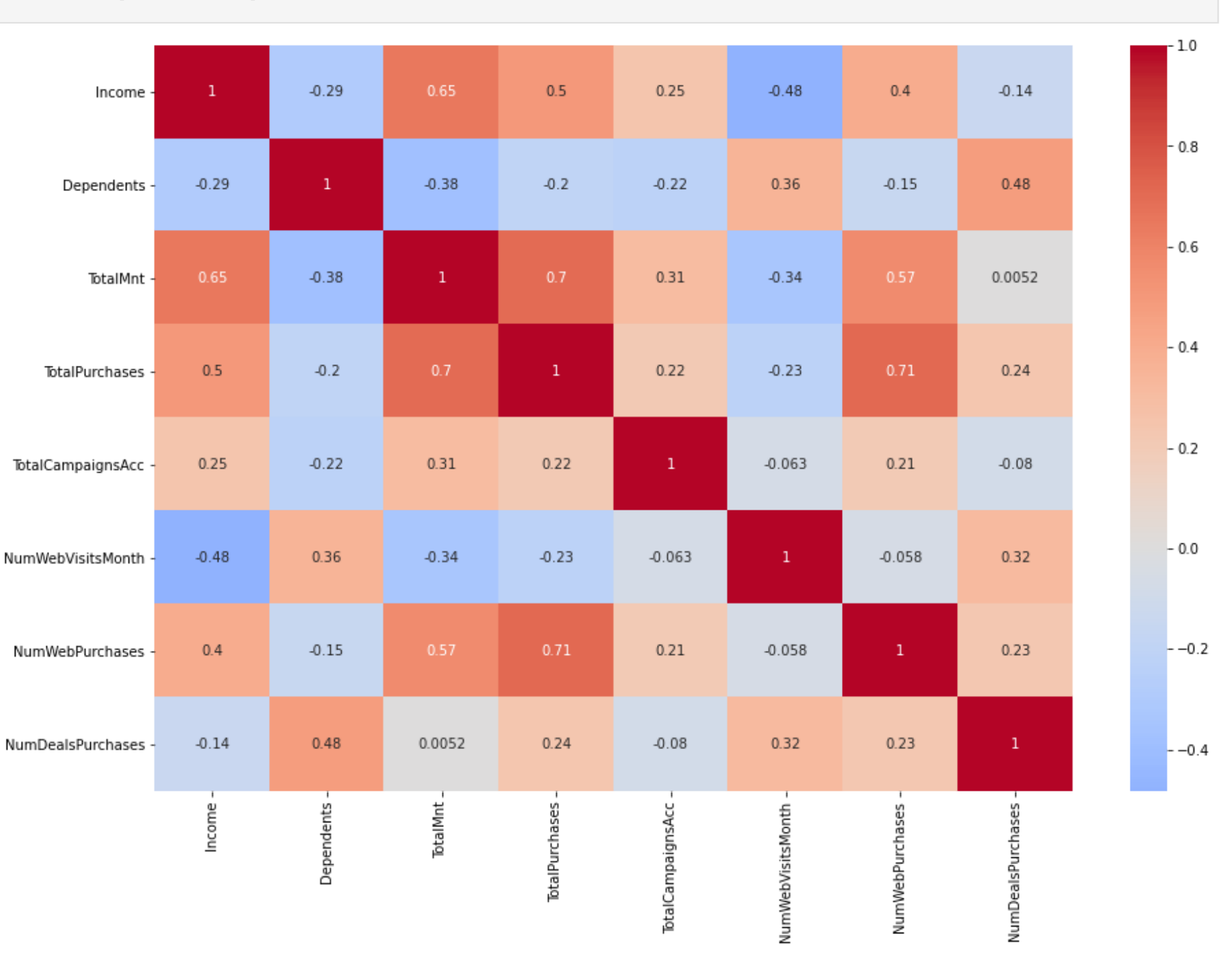
Use all numeric columns except the 'ID' column

In [36]:
```
corrs = mrkt_df.drop(columns='ID').corr(method = 'kendall')

plt.figure(figsize=(10,10))  #figure size
sns.heatmap(corrs,  cmap='coolwarm', center =0 );  #centeral color of the colormap (cmap)
```



Try again, less features:

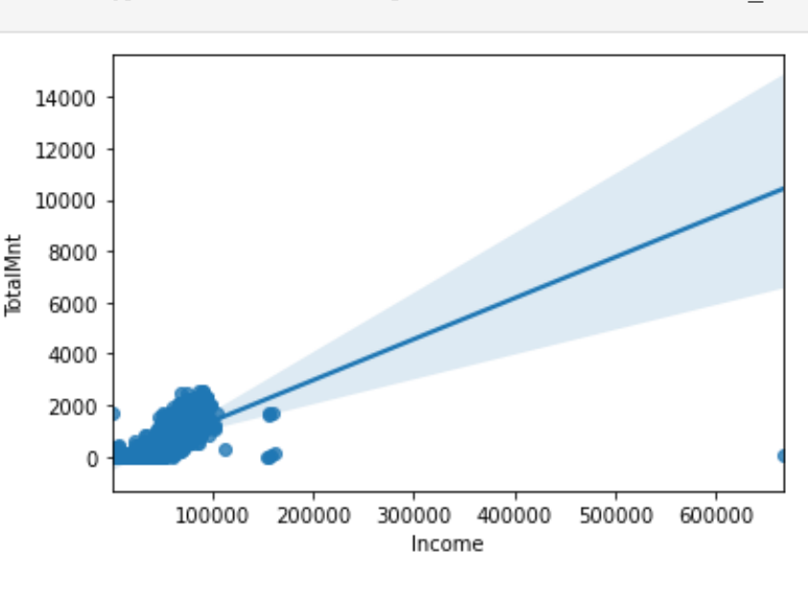
In [37]:
```
corrs = mrkt_df[features].corr(method = 'kendall')

plt.figure(figsize=(15,10))  #figure size
sns.heatmap(corrs, cmap='coolwarm', center=0, annot = True);
```



We found some correlations :-)

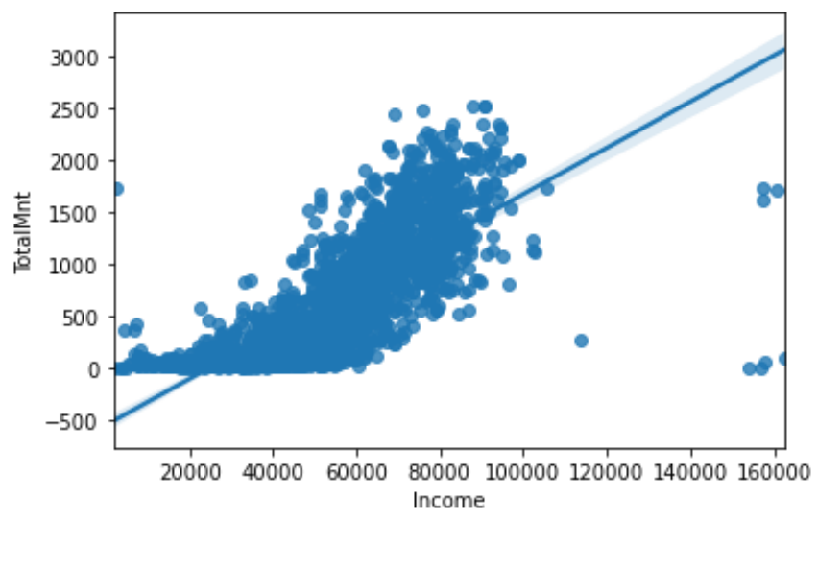
Let's begin with income and spendings

In [38]:
```
sns.regplot(x='Income', y='TotalMnt', data=mrkt_df);
```



the same, but remove the outliers:

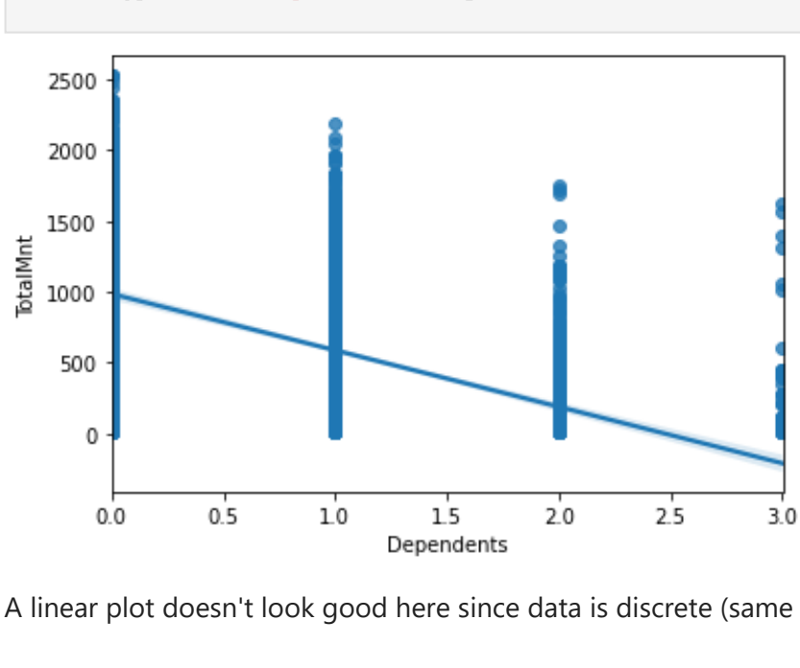
In [39]:
```
sns.regplot(x='Income', y='TotalMnt', data=mrkt_df[mrkt_df['Income'] < 200000]);
```



> **Observation 1:**
>
> The higher the income is, the more you spend
>
> # (a bit trivial)

A plot illustrating negative effect of having dependents (kids & teens) on spending:

In [40]:
```
sns.regplot(x='Dependents', y='TotalMnt', data=mrkt_df);
```

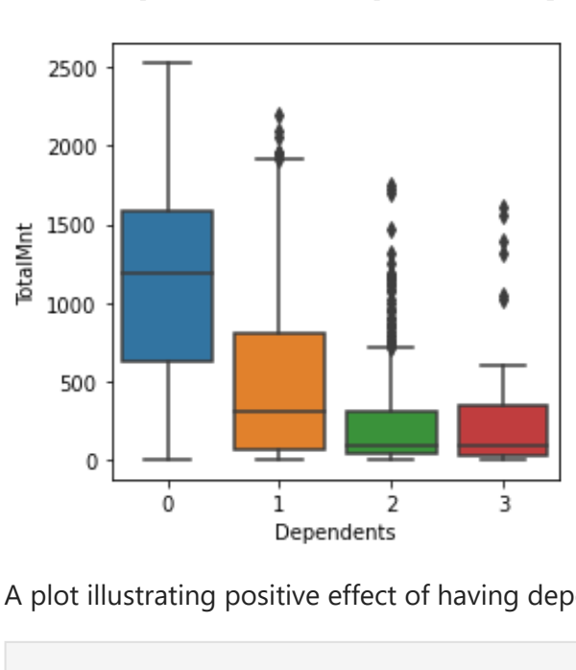


A linear plot doesn't look good here since data is discrete (same for ordinal data - e.g. - none, few, many)

Let's try a boxplot instead

In [41]:
```
plt.figure(figsize=(4,4))
sns.boxplot(x='Dependents', y='TotalMnt', data=mrkt_df)
```

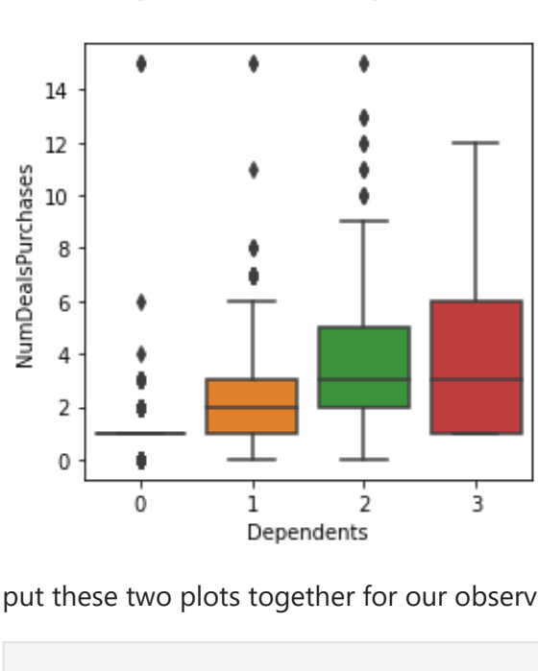
Out[41]:
```
<AxesSubplot:xlabel='Dependents', ylabel='TotalMnt'>
```



A plot illustrating positive effect of having dependents (kids & teens) on number of deals purchased:

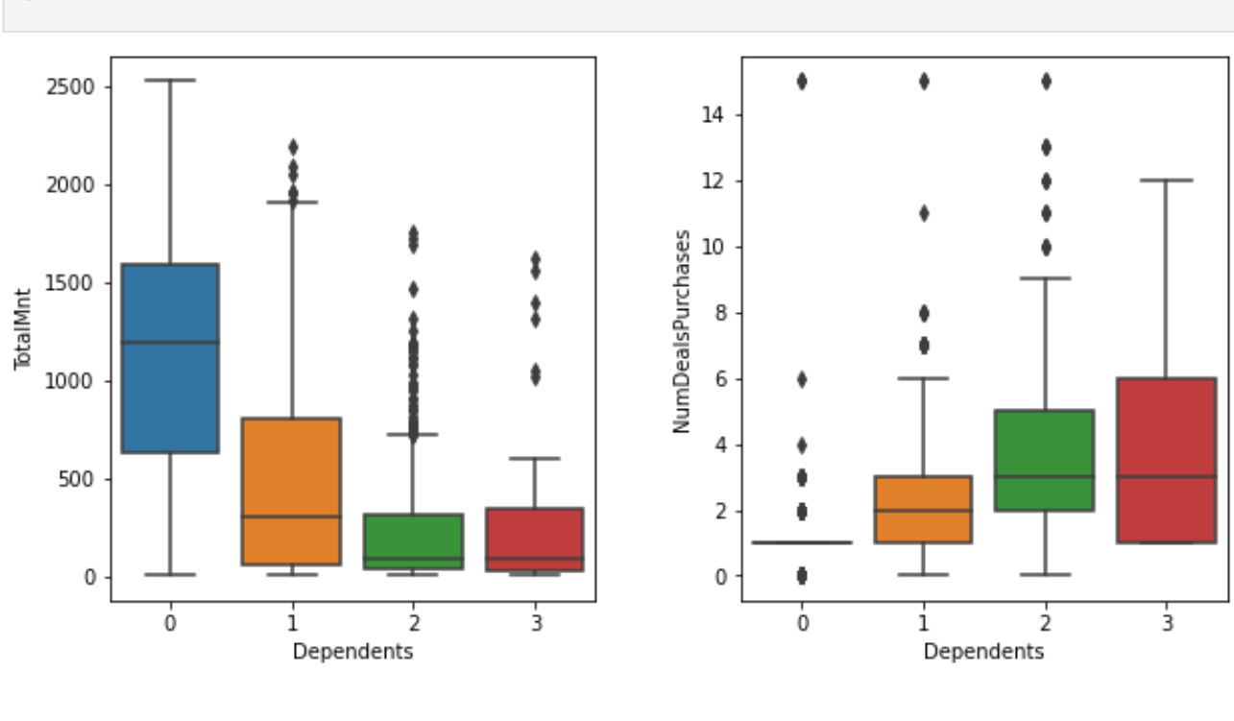
In [42]:
```
plt.figure(figsize=(4,4))
sns.boxplot(x='Dependents', y='NumDealsPurchases', data=mrkt_df)
```

Out[42]:
```
<AxesSubplot:xlabel='Dependents', ylabel='NumDealsPurchases'>
```



put these two plots together for our observation:

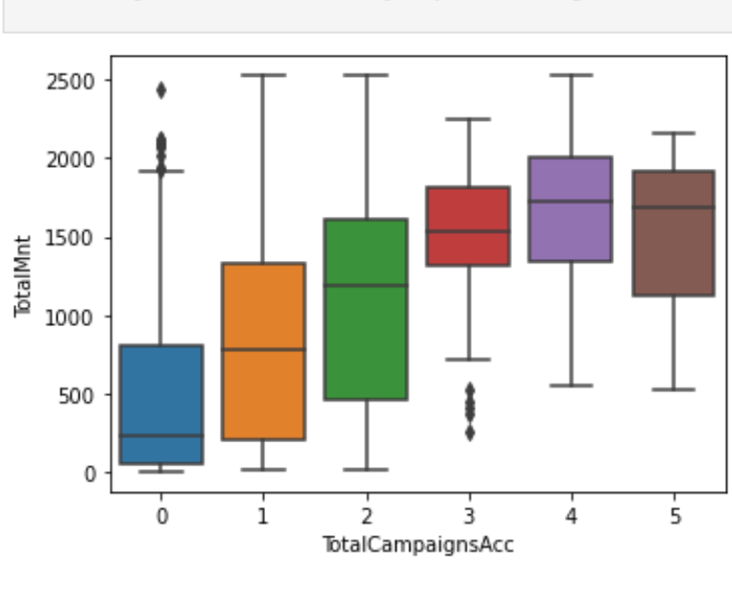
In [43]:
```
fig, ax = plt.subplots(1,2, figsize = (10,5))
plt.subplots_adjust(wspace = 0.3)
sns.boxplot(x='Dependents', y='TotalMnt', data=mrkt_df, ax = ax[0])
sns.boxplot(x='Dependents', y='NumDealsPurchases', data=mrkt_df, ax = ax[1])
plt.show()
```



> **Observation 2:**
>
> People with more kids spend less
>
> People with more kids buy more deals

Plots illustrating the positive effect of campaigns

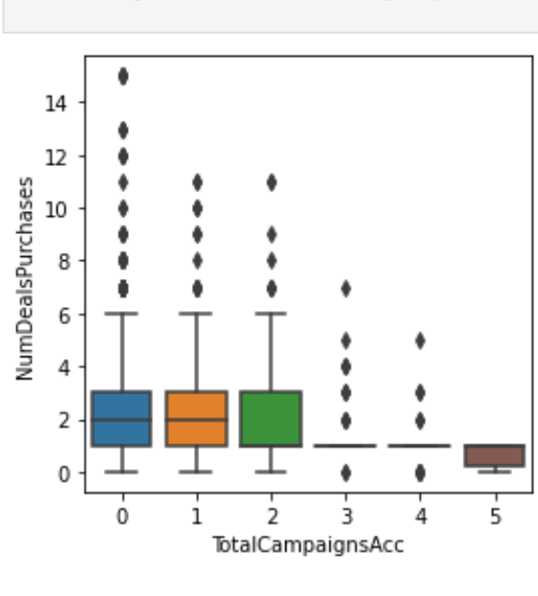
In [44]:
```
plt.figure(figsize=(5.5,4))
sns.boxplot(x='TotalCampaignsAcc', y='TotalMnt', data=mrkt_df);
```



> **Observation 3:**
>
> Campaigns seem to be working

Do people who access campaigns also buy more deals?

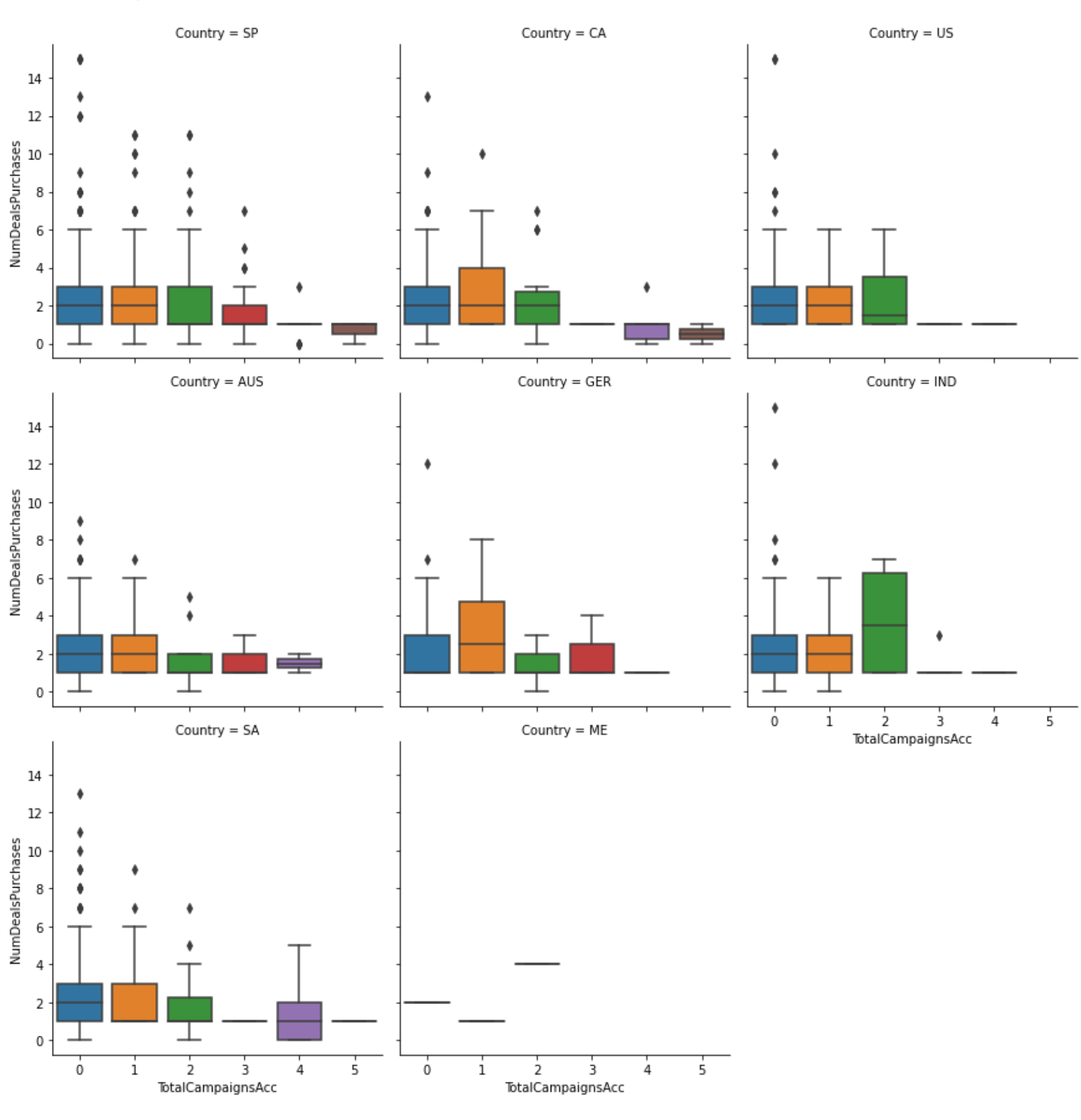
In [45]:
```
plt.figure(figsize=(4,4))
sns.boxplot(x='TotalCampaignsAcc', y='NumDealsPurchases', data=mrkt_df);
```



Doesn't seem so. But maybe we need to look at different countries?

In [46]:
```
sns.catplot(
    data=mrkt_df, x='TotalCampaignsAcc', y='NumDealsPurchases',
    col='Country', kind='box', col_wrap=3, height=4
)
```

Out[46]:
```
<seaborn.axisgrid.FacetGrid at 0x21d76cf4d90>
```

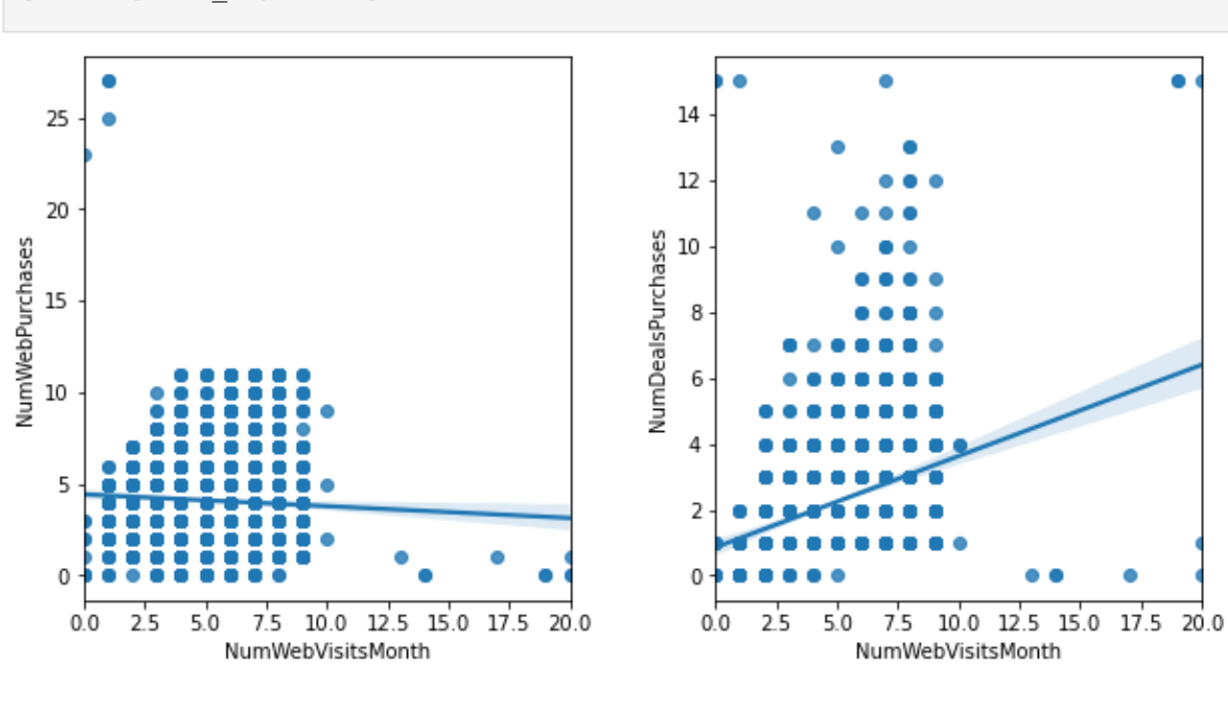


Note: maybe we need to try: different years? number of dependants? maratial status?

> **Observation??**
>
> Campigns don't seem to corrolate with deals, even for different countries (this may be a good thing)
>
> ## This is not really an observation, since it doesn't focus on finding a new connection. It is only an observation if the company currently believes that there is a connection and you're proving them wrong.

What about web visits and web purchases?

In [47]:
```
fig, ax = plt.subplots(1,2,figsize=(10, 5))
sns.regplot(x='NumWebVisitsMonth', y='NumWebPurchases', data=mrkt_df, ax=ax[0])
sns.regplot(x='NumWebVisitsMonth', y='NumDealsPurchases', data=mrkt_df, ax = ax[1])
plt.subplots_adjust(wspace = 0.3)
```



> **Observation 4:**
>
> Number of web visits in the last month is not positively correlated with number of web purchases
>
> Instead, it is positively correlated with the number of deals purchased, suggesting that deals are an effective way of stimulating purchases on the website

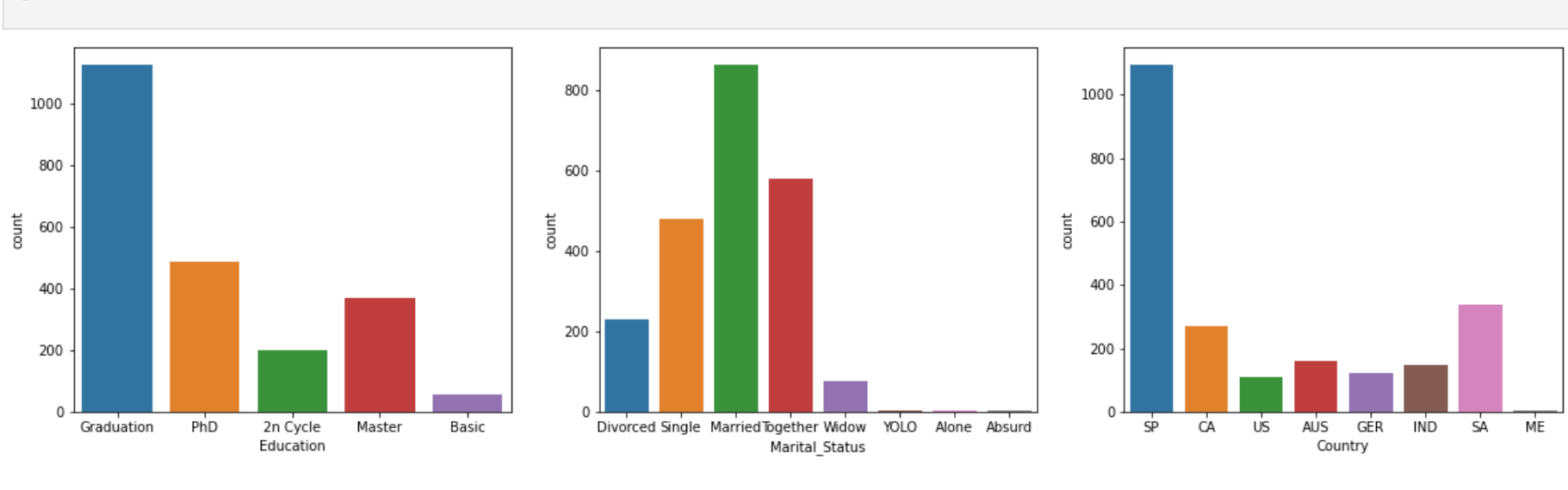
---

4. Categorical data

We have some categorical data.

Plot it in one row, or on seperate rows. Here's one row:

In [48]:
```
fig, axes = plt.subplots(figsize=(20, 5), ncols=3)
sns.countplot(ax=axes[0], x="Education", data=mrkt_df)
sns.countplot(ax=axes[1], x="Marital_Status", data=mrkt_df)
sns.countplot(ax=axes[2], x="Country", data=mrkt_df)
plt.show()
```



Let's look at the marital status:

In [49]:
```
mrkt_df.groupby('Marital_Status')[['ID']].count()
```

Out[49]:

	ID
Marital_Status	
Absurd	2
Alone	3
Divorced	231
Married	864
Single	479
Together	579
Widow	77
YOLO	2

Is there a correlation between marital status and spendings?

Create a df that holds only the status's we're interested in - the four biggest categories

In [50]:
```
status_mrkt = mrkt_df.loc[(mrkt_df.Marital_Status  == 'Single')|(mrkt_df.Marital_Status == 'Married')|(mrkt_df.
```

Create one-hot encodings for the categorical variables:

In [51]:
```
features2 = ['Income', 'Dependents','TotalMnt','TotalPurchases', 'TotalCampaignsAcc', 'NumDealsPurchases', 'Nun

status_mrkt = pd.get_dummies(status_mrkt[features2])
```

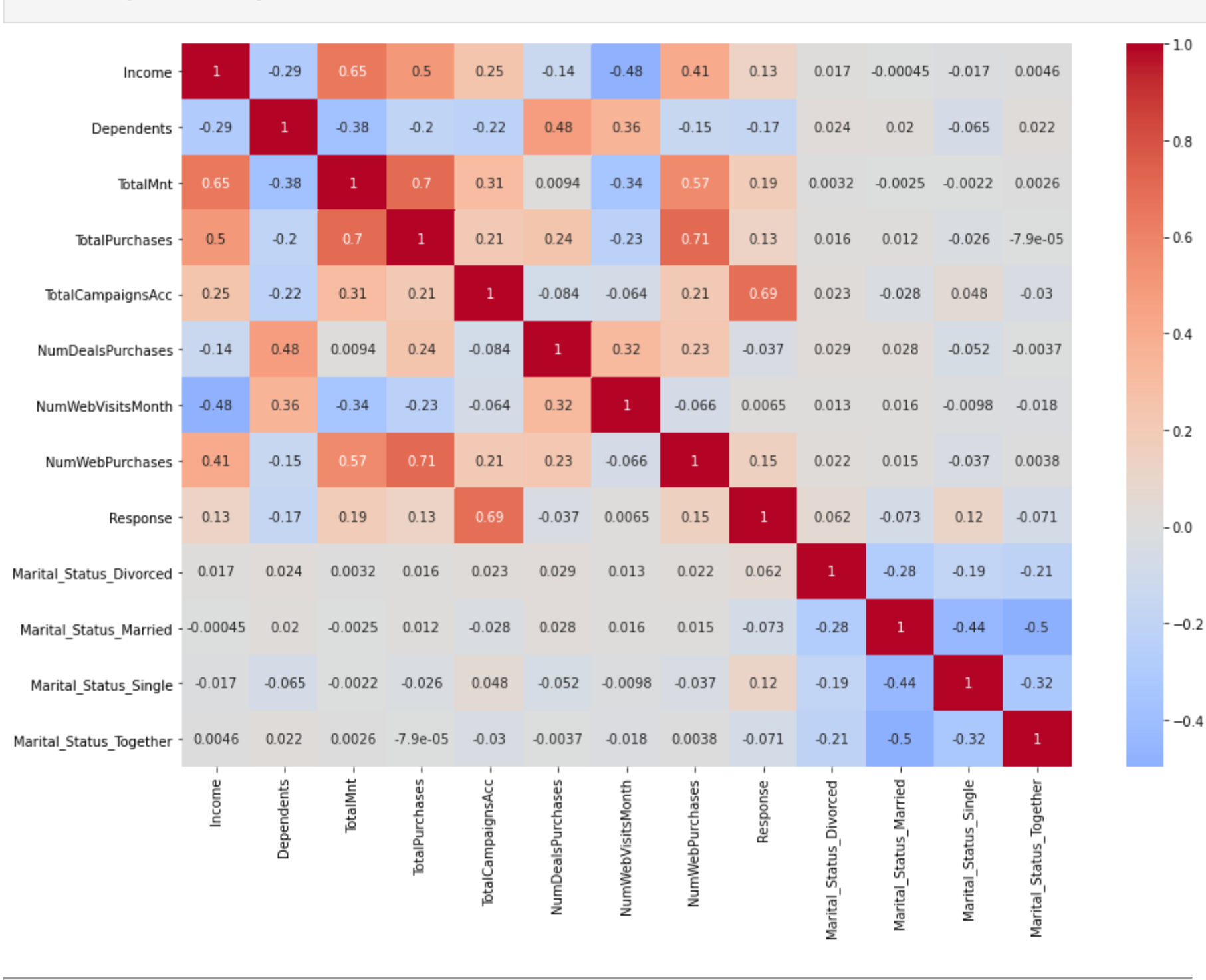
One-hot encoding doesn't affect variables that are not categorical

In [52]:
```
status_mrkt.head()
```

Out[52]:

	Income	Dependents	TotalMnt	TotalPurchases	TotalCampaignsAcc	NumDealsPurchases	NumWebVisitsMonth	NumWebPurchases
0	84835.0	0	1190	15	1	1	1	4
1	57091.0	0	577	18	2	1	5	7
2	67267.0	1	251	11	0	1	2	3
3	32474.0	2	11	4	0	1	7	1
4	21474.0	1	91	8	2	2	7	3

In [53]:
```
corrs = status_mrkt.corr(method = 'kendall')

plt.figure(figsize=(15,10))  #figure size
sns.heatmap(corrs, cmap='coolwarm', center=0, annot = True);
```



> **Observation??**
>
> There isn't any correlation between martial staus and spendings
>
> **Observations should focus on what there IS, not what there isn't**
>
> Let's try to look at the response to campgains

Is there a connection between the status, dependents and response?

In [54]:
```
grouped_mrkt = mrkt_df.groupby(['Marital_Status', 'Dependents'])['Response' ].aggregate('mean').reset_index()
grouped_mrkt
```

Out[54]:

	Marital_Status	Dependents	Response
0	Absurd	0	0.500000
1	Alone	1	0.000000
2	Alone	2	0.500000
3	Divorced	0	0.448276
4	Divorced	1	0.109244
5	Divorced	2	0.183673
6	Divorced	3	0.000000
7	Married	0	0.182609
8	Married	1	0.091314
9	Married	2	0.089286
10	Married	3	0.000000
11	Single	0	0.389222
12	Single	1	0.130045
13	Single	2	0.150685
14	Single	3	0.062500
15	Together	0	0.163399
16	Together	1	0.086957
17	Together	2	0.071429
18	Together	3	0.066667
19	Widow	0	0.370370
20	Widow	1	0.181818
21	Widow	2	0.176471
22	YOLO	1	0.500000

If we want a table, we can groupby, and then unstack:

In [55]:
```
table_mrkt = mrkt_df.groupby(['Marital_Status', 'Dependents'])['Response' ].aggregate('mean').unstack()
table_mrkt
```

Out[55]:

Dependents	0	1	2	3
Marital_Status				
Absurd	0.500000	NaN	NaN	NaN
Alone	NaN	0.000000	0.500000	NaN
Divorced	0.448276	0.109244	0.183673	0.000000
Married	0.182609	0.091314	0.089286	0.000000
Single	0.389222	0.130045	0.150685	0.062500
Together	0.163399	0.086957	0.071429	0.066667
Widow	0.370370	0.181818	0.176471	NaN
YOLO	NaN	0.500000	NaN	NaN

Or we can use a pivot table to obtain the same results:

In [56]:
```
mrkt_df.pivot_table('Response', index='Marital_Status', columns='Dependents', aggfunc='mean')  #aggfunc = 'mear
```

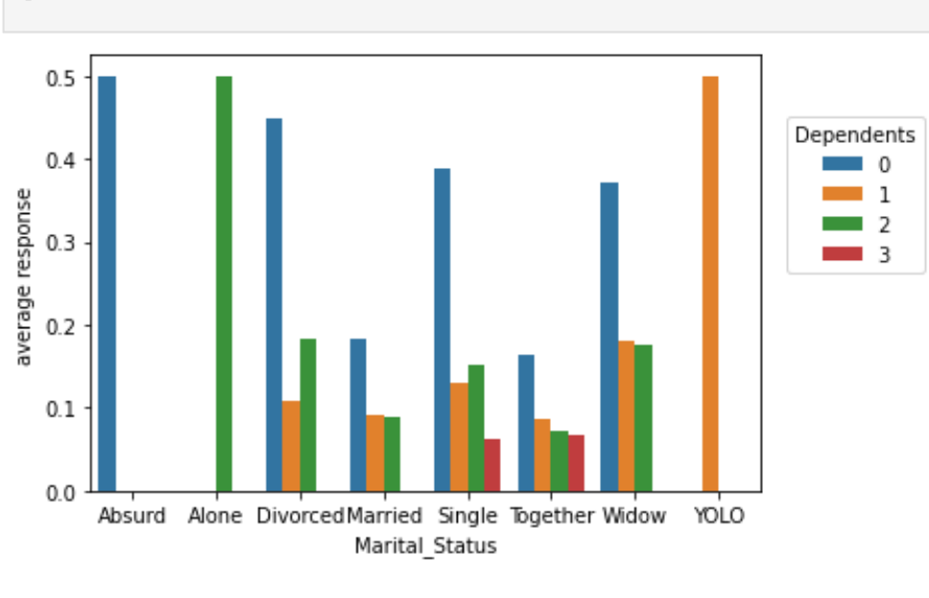
Out[56]:

Dependents	0	1	2	3
Marital_Status				
Absurd	0.500000	NaN	NaN	NaN
Alone	NaN	0.000000	0.500000	NaN
Divorced	0.448276	0.109244	0.183673	0.000000
Married	0.182609	0.091314	0.089286	0.000000
Single	0.389222	0.130045	0.150685	0.062500
Together	0.163399	0.086957	0.071429	0.066667
Widow	0.370370	0.181818	0.176471	NaN
YOLO	NaN	0.500000	NaN	NaN

Why would we want a table? Because with a table it's easier to figure out what's going on and what we should plot

Single & Divorced with no kids are more likly to respond to a campaign

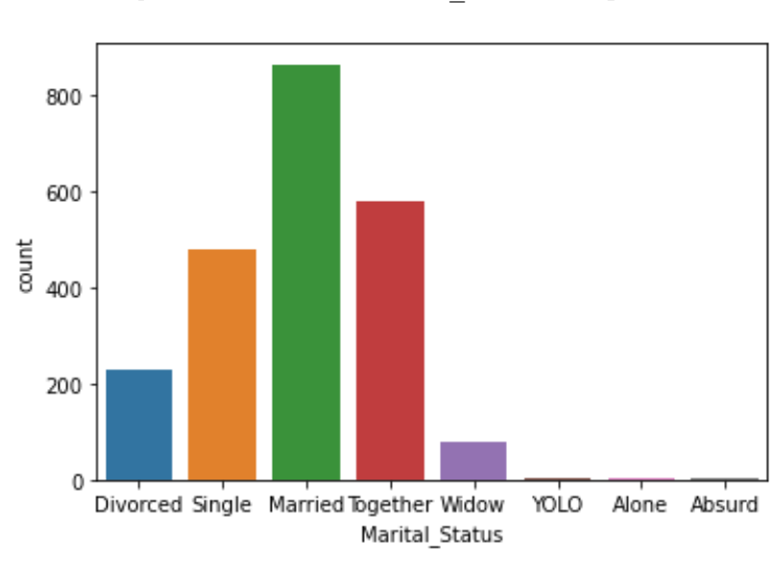
In [57]:
```
ax = sns.barplot(data = grouped_mrkt, x='Marital_Status', y='Response', hue = 'Dependents')
ax.set(ylabel='average response')
plt.legend(title='Dependents', loc=(1.04,0.5)) #the legend is outside since it didn't fit inside
plt.show()
```



remeber we plotted categorical variables?

In [58]:
```
sns.countplot(x="Marital_Status", data=mrkt_df)
```

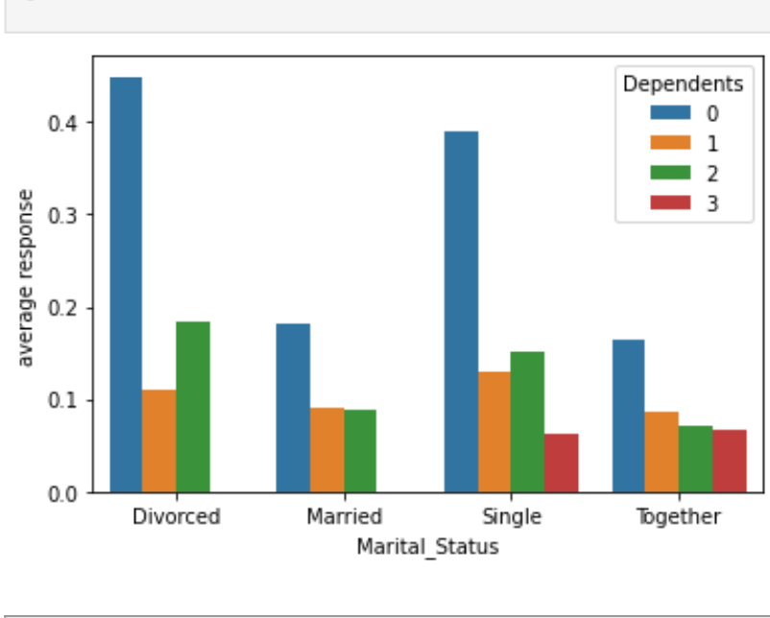
Out[58]:
```
<AxesSubplot:xlabel='Marital_Status', ylabel='count'>
```



since there are almost no widows, 'YOLO's, alones and 'Absurd's, lests ignore them

Note: this doesn't just make the figure look prettier. It's important to ignore them. We don't want to reach a conclusion if based on 2-3 YOLO's..

In [59]:
```
group_slice = grouped_mrkt[(grouped_mrkt.Marital_Status == 'Divorced') | (grouped_mrkt.Marital_Status == 'Singl
ax = sns.barplot(data = group_slice, x='Marital_Status', y='Response', hue = 'Dependents')
ax.set(ylabel='average response')
plt.legend(title='Dependents', loc=('upper right')) #the legend fits inside now
plt.show()
```



> **Observation 5**
>
> Single & Divorced with no kids are more likly to respond to a campaign - average response is higher than 30%
>
> Married & Together with no kids respond at around 20%

Is this it? No!! There is always more to do. We haven't touched country, education, or campaign responses and much more.

> A summary of new functions:
>
> - `drop()` - remove labels or columns documentation
> - `str.replace('x','y')` - replace string x with string y
> - `reset_index(drop=true)` - resets the index. As defult, the old index is added as a column. Use drop=True to avoid this. documentation
> - `plt.subplots_adjust` - a matplotlib function. adjusts space between the plots. documentation
> - `sum(axis=1)` - sums the values according to the rows, instead of the default sum by columns
> - create a list using list comprehension - "do something for data in dataframe if condition holds"
> - `catplot()` - plot categorical variables onto a grid. documentation
> - `countplot()` - counts observations for each category
> - `pd.get_dummies()` - creates one-hot encodings for categorical variables. documentation
> - `unstack()` - change the hiearachy of data. documentation
> - `pivot()` - create a pivot table to group and summarize data. documentation
>
> ---

In [ ]: