Predicting the Success of a Marketing Campaign using Machine Learning

Problem Statement & Justification for Proposed Approach

The main problem we are working to solve how to increase responses to a marketing campaign by identifying which customers are likely to respond positively versus negatively, for a store that is in-person, online, and a catalog. If responses to the stores marketing campaign can be identified and then increased, it will increase overall profits of the store. This problem will be tackled by using classification methods to identify customers who are likely to response positively to a marketing campaign sent by the store to the customer. Modeling algorithms such as logistic regression, random forest, decision tree, k-nearest neighbors, adaBoost, and linear discriminant analysis (LDA) will be used to accomplish this classification task. Before modeling, data wrangling is completed to clean and pre-process the dataset. Imputation, transformations, feature engineering, and scaling is done to increase the usability of the dataset. Once the modeling has been completed, the "best" model will be chosen based on metrics like accuracy, recall, precision, and the confusion matrix. The "best" model will then be used to predict which customers are likely to response positively to the marketing campaign, at a chosen threshold level. This list of customers will be the recommended set of people to send the marketing campaign too, since they will be the most likely to buy from the store after being directly marketed to. This "best" model can be used in the future with new customers to make predictions about their likelihood to respond to a marketing campaign, increasing profits and saving money by only sending the campaigns to customers who will make purchases after receiving it.

Importing Necessary Libraries and Packages

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from google.colab import files
import io

from sklearn import preprocessing
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.linear_model import LinearRegression, LogisticRegression, LogisticRegressionCV
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
from sklearn.feature_selection import VarianceThreshold
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis


import matplotlib.pylab as plt

!pip install dmba
import dmba
from dmba import classificationSummary
from dmba import gainsChart, liftChart
from dmba import plotDecisionTree, regressionSummary, exhaustive_search
from dmba import stepwise_selection
from dmba.metric import AIC_score
from dmba import adjusted_r2_score, AIC_score, BIC_score

from scipy.stats import skew
from sklearn.impute import SimpleImputer

%matplotlib inline

from datetime import date
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting dmba
  Downloading dmba-0.1.0-py3-none-any.whl (11.8 MB)
     |████████████████████████████████| 11.8 MB 5.0 MB/s
Installing collected packages: dmba
Successfully installed dmba-0.1.0
no display found. Using non-interactive Agg backend
```

Exploratory Data Analysis

In this section, we will explore our data both numerically and visually. Descriptive statistics of the numerical variables will help us better understand the predictors and histograms will help us better view the various skews of the predictors. A heatmap will also be used to assess the correlation/collinearity between the predictor variables.

After loading in the marketingCampaign dataset, descriptive statistics are viewed to get a better grasp on the numeric features in the dataset. The marketingCampaign dataset is made up of 2240 records, a target variable ('Reponse'), and 28 predictor variables. Of the 28 predictors, there is one ID column, four categorical variables ('Education', 'Marital_Status', 'Year_Birth', 'and 'Dt_Customer'), 17 numeric predictors ('Income', 'MntFishProducts', 'MntMeatProducts', 'MntFruits', 'MntSweetProducts', 'MntWines', 'MntGoldProds', 'NumDealsPurchases', 'NumCatelogPurchases', 'NumStorePurchases', 'NumWebPurchases', 'NumWebVisitsMonth', 'Recency', 'Kidhome', 'Teenhome', 'Z_CostContact', and 'Z_Revenue'), and 6 binary predictors ('AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', and 'Complain')

After viewing descriptive statistics, histograms and bar plots are created to better understand the various skews and distributions of all the predictors. Figure 1- Bar Plot of Responses displays the number of customers who responded either 0/No or 1/Yes to the marketing campaign. From the plot we can clearly see that the target variable is imbalanced. There are 1906 No responses and 334 Yes responses. This is something important to note when moving into the modeling phase because it may make it hard for some algorithms to classify 1/Yes class given the few responses in the positive class. Figures 2-6 show the distributions of responses to the 5 campaigns sent out to the customers. Figures 2-6 display the distributions of the 5 AcceptedCmp predictors. We can also clearly see from these figures that these categorical predictors have a lot more No/0's than Yes/1's. Figure 7- Education Distribution displays the distribution of the various levels of education. The histogram appears to be right skewed, with majority of the customers graduating. Figure 8- Marital Status Distribution displays the distribution of the various marital statuses. The histogram appears to be a slightly right skewed, with majority of customers being married, together, or single. Figure 9- Income Distribution displays the distribution of the various income levels for the customers. The histogram appears to be a very slightly left skewed, most likely because there are some outliers present in the upper range of income. Figure 10- Kidhome Distribution displays the distribution of the number of kids in each customer's home. The histogram appears right skewed, with majority of the customers having no kids in the home. Figure 11- Teenhome Distribution displays the distribution of the number of teens in customers households. The histogram appears right skewed, with majority of the customers having no teens in the home. Figure 12- Recency, since last purchase, Distribution displays the distribution of the number of days since the last purchase by a customer. The histogram appears somewhat uniformly distributed, with slightly more customers in the 30 day and 50 day range. Figures 13-18 display the distributions in dollar amounts spent in the past two years, on various products sold by the store. The 6 histograms all appear to be very right skewed, with majority of customers only spending a small amount on the various products (i.e., wine, fruit, meat, fish, sweets, and gold). Figures 19-22 display the distributions of the number of purchases made given various circumstances. The 4 histograms all appear to be right skewed, with majority of the customers only purchasing a small number of items, given various circumstances (i.e., number of purchases made with a deal, number of purchases made online, number of purchases made by catalog, and number of purchases made in store). Figure 23- Distribution of Visits to Company Website per Month displays the distribution of the number of times a customer visits the company website per month. The histogram appears slightly left skewed, with majority of customers visiting the website 5 or more times per month. Figure 24- Complaints displays the distribution of customers that have complained in the past 2 years. The histogram *appears to be very right skewed, with almost all of the customers not complaining in the past two years, and a few customers complaining in the past two years. The last two figures, Figure 25 and Figure 26, display the distribution of normalized CostContact and Revenue. Given that these two predictors have already been normalized, they display a uniform distribution.*

Now that we have further investigated the distributions and skews of the various predictors in the marketingCampaign dataset, we can focus on collinearity, the between-predictor correlations. To do this, the correlation function and heatmap function in the seaborn package are used to numerically and visually examine the correlations between the predictors in the marketingCampaign dataset. Figure 27- Predictors Heatmap displays a heatmap of the correlations. From the heatmap we can see that majority of the predictors do not display collinearity. Using a correlation coefficient threshold of 0.70, only one predictor, 'NumCatalogPurchases', displays collinearity.

Data Dictionary

Response (target)	1 if customer accepted the offer in the last campaign, 0 otherwise
ID	Numerical identification number for each customer
Accepted Cmp1	1 if customer accepted the offer in the last campaign, 0 otherwise
Accepted Cmp2	1 if customer accepted the offer in the last campaign, 0 otherwise
Accepted Cmp3	1 if customer accepted the offer in the last campaign, 0 otherwise
Accepted Cmp 4	1 if customer accepted the offer in the last campaign, 0 otherwise
Accepted Cmp 5	1 if customer accepted the offer in the last campaign, 0 otherwise
Complain	1 if customer complained in the last 2 years
Dt_Customer	Date of customer's enrollment with the company
Year_Birth	Customer's birth year
Education	Customer's level of education
Marital_Status	Customer's marital status
Kidhome	Number of small children in customer's household
Teenhome	Number of teenagers in customer's household
Income	Customer's yearly household income
MntFishProducts	Amount spent on fish products in the last 2 years
MntMeatProducts	Amount spent on meat products in the last 2 years
MntFruits	Amount spent on fruit products in the last 2 years
MntSweetProducts	Amount spent on sweet products in the last 2 years
MntWines	Amount spent on wine products in the last 2 years
MntGoldProds	Amount spent on gold products in the last 2 years
NumDealsPurchases	Number of purchases made with a discount
NumCatalogPurchases	Number of purchases made using catalog
NumStorePurchases	Number of purchases made directly in stores
NumWebPurchases	Number of purchases made through the company's website
NumWebVisitsPerMonth	Number of visits to company's website in the last month
Recency	Number of days since the last purchase
Z_CostContact	Scaled cost of contacting customer
Z_Revenue	Scaled revenue

Descriptive Statistics of Numeric Variables

In [ ]:
```
# Loading in the dataset
uploaded = files.upload()
marketingCampaign = pd.read_excel(io.BytesIO(uploaded['marketing_campaign.xls
x']))
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving marketing_campaign.xlsx to marketing_campaign.xlsx
```

In [ ]:
```
# viewing the dataset
marketingCampaign
```

Out[ ]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer
0	5524	1957	Graduation	Single	58138.0	0	0	2012-09-04
1	2174	1954	Graduation	Single	46344.0	1	1	2014-03-08
2	4141	1965	Graduation	Together	71613.0	0	0	2013-08-21
3	6182	1984	Graduation	Together	26646.0	1	0	2014-02-10
4	5324	1981	PhD	Married	58293.0	1	0	2014-01-19
...	...	...	...	...	...	...	...	...
2235	10870	1967	Graduation	Married	61223.0	0	1	2013-06-13
2236	4001	1946	PhD	Together	64014.0	2	1	2014-06-10
2237	7270	1981	Graduation	Divorced	56981.0	0	0	2014-01-25
2238	8235	1956	Master	Together	69245.0	0	1	2014-01-24
2239	9405	1954	PhD	Married	52869.0	1	1	2012-10-15

2240 rows × 29 columns

In [ ]:
```
# basic descriptive statistics of the numeric variables
marketingCampaign.describe()
```

Out[ ]:

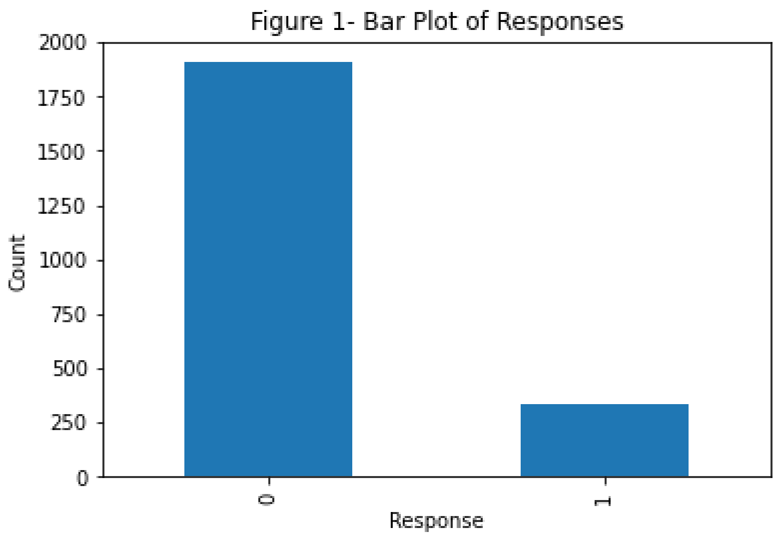
	ID	Year_Birth	Income	Kidhome	Teenhome	Recency	Mnt\
count	2240.000000	2240.000000	2216.000000	2240.000000	2240.000000	2240.000000	2240.0
mean	5592.159821	1968.805804	52247.251354	0.444196	0.506250	49.109375	303.9
std	3246.662198	11.984069	25173.076661	0.538398	0.544538	28.962453	336.5
min	0.000000	1893.000000	1730.000000	0.000000	0.000000	0.000000	0.0
25%	2828.250000	1959.000000	35303.000000	0.000000	0.000000	24.000000	23.7
50%	5458.500000	1970.000000	51381.500000	0.000000	0.000000	49.000000	173.5
75%	8427.750000	1977.000000	68522.000000	1.000000	1.000000	74.000000	504.2
max	11191.000000	1996.000000	666666.000000	2.000000	2.000000	99.000000	1493.0

8 rows × 26 columns

Plots of Predictors

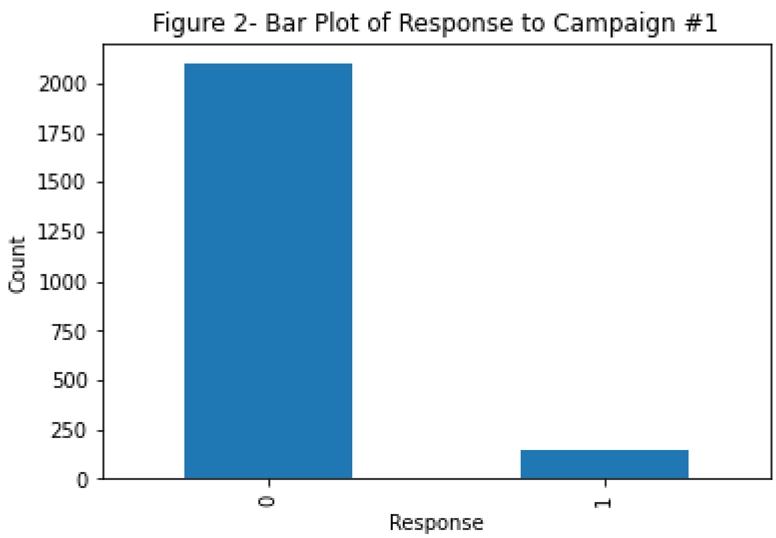
In [ ]:
```
marketingCampaign['Response'].value_counts().plot(kind = 'bar')
plt.xlabel('Response')
plt.ylabel('Count')
plt.title('Figure 1- Bar Plot of Responses')
```

Out[ ]: Text(0.5, 1.0, 'Figure 1- Bar Plot of Responses')



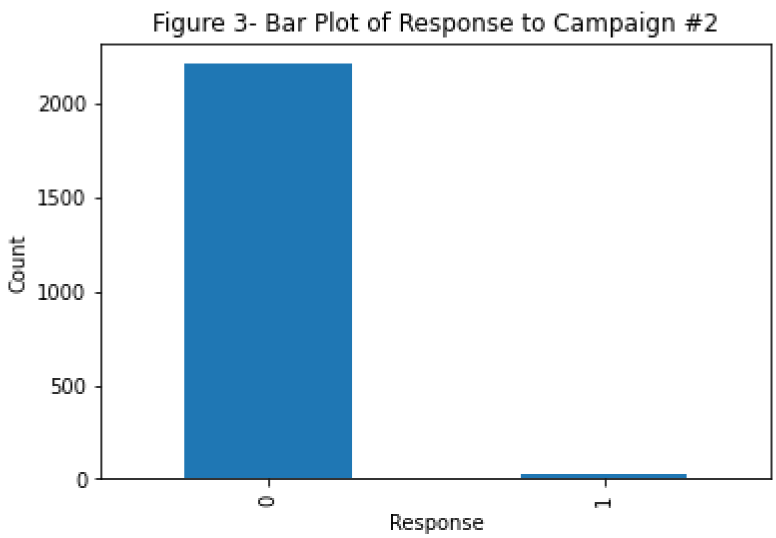
In [ ]:
```
marketingCampaign['AcceptedCmp1'].value_counts().plot(kind = 'bar')
plt.xlabel('Response')
plt.ylabel('Count')
plt.title('Figure 2- Bar Plot of Response to Campaign #1')
```

Out[ ]: Text(0.5, 1.0, 'Figure 2- Bar Plot of Response to Campaign #1')



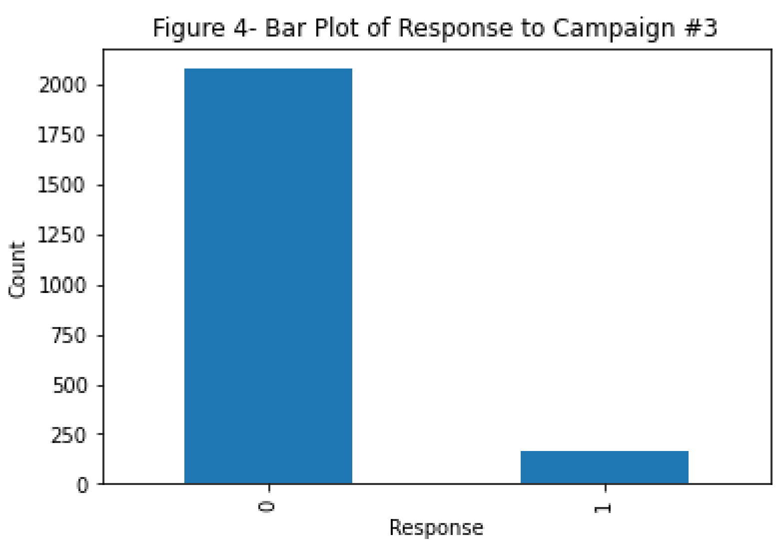
In [ ]:
```
marketingCampaign['AcceptedCmp2'].value_counts().plot(kind = 'bar')
plt.xlabel('Response')
plt.ylabel('Count')
plt.title('Figure 3- Bar Plot of Response to Campaign #2')
```

Out[ ]:
```
Text(0.5, 1.0, 'Figure 3- Bar Plot of Response to Campaign #2')
```



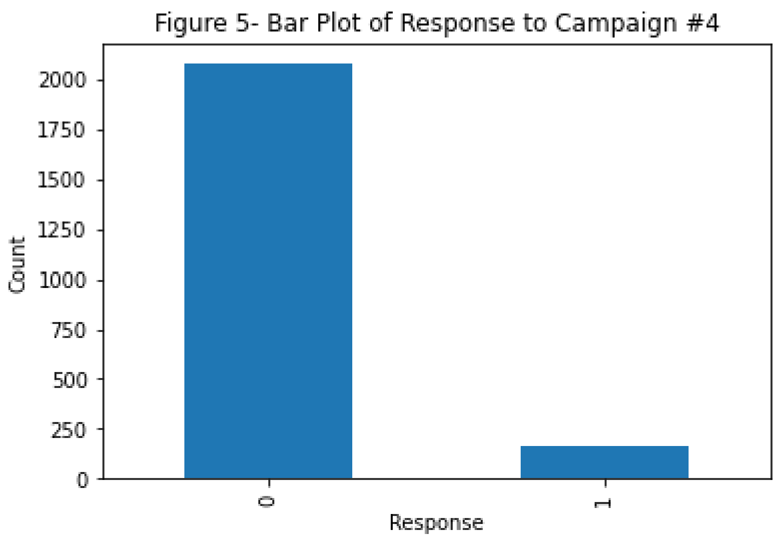
In [ ]:
```
marketingCampaign['AcceptedCmp3'].value_counts().plot(kind = 'bar')
plt.xlabel('Response')
plt.ylabel('Count')
plt.title('Figure 4- Bar Plot of Response to Campaign #3')
```

Out[ ]:
```
Text(0.5, 1.0, 'Figure 4- Bar Plot of Response to Campaign #3')
```



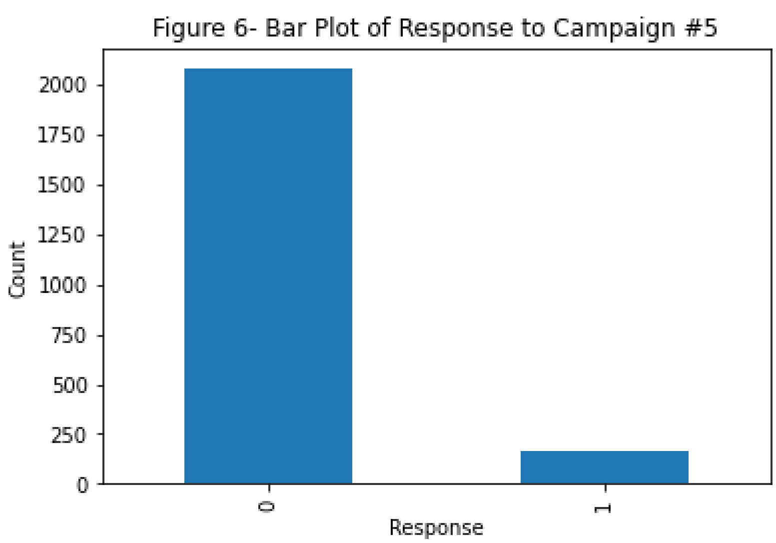
In [ ]:
```
marketingCampaign['AcceptedCmp4'].value_counts().plot(kind = 'bar')
plt.xlabel('Response')
plt.ylabel('Count')
plt.title('Figure 5- Bar Plot of Response to Campaign #4')
```

Out[ ]: `Text(0.5, 1.0, 'Figure 5- Bar Plot of Response to Campaign #4')`



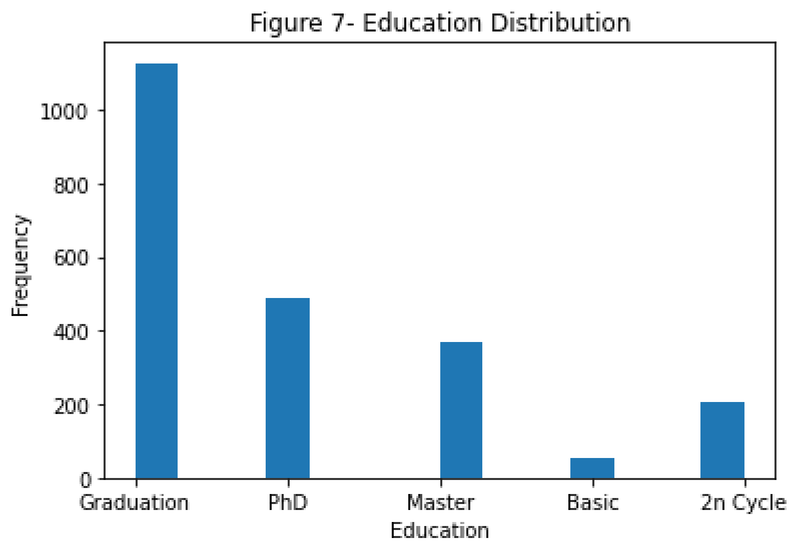
In [ ]:
```
marketingCampaign['AcceptedCmp5'].value_counts().plot(kind = 'bar')
plt.xlabel('Response')
plt.ylabel('Count')
plt.title('Figure 6- Bar Plot of Response to Campaign #5')
```

Out[ ]: `Text(0.5, 1.0, 'Figure 6- Bar Plot of Response to Campaign #5')`



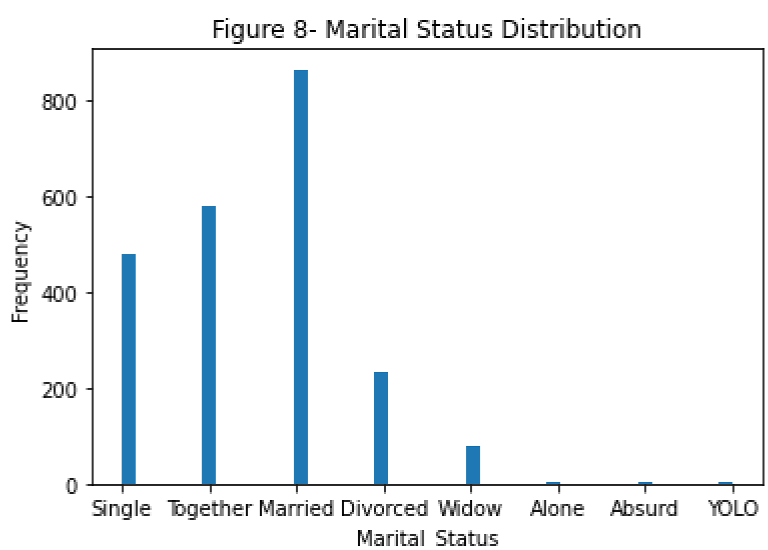
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['Education'], bins='auto')
plt.xlabel('Education')
plt.ylabel('Frequency')
plt.title('Figure 7- Education Distribution')
```

Out[ ]: Text(0.5, 1.0, 'Figure 7- Education Distribution')



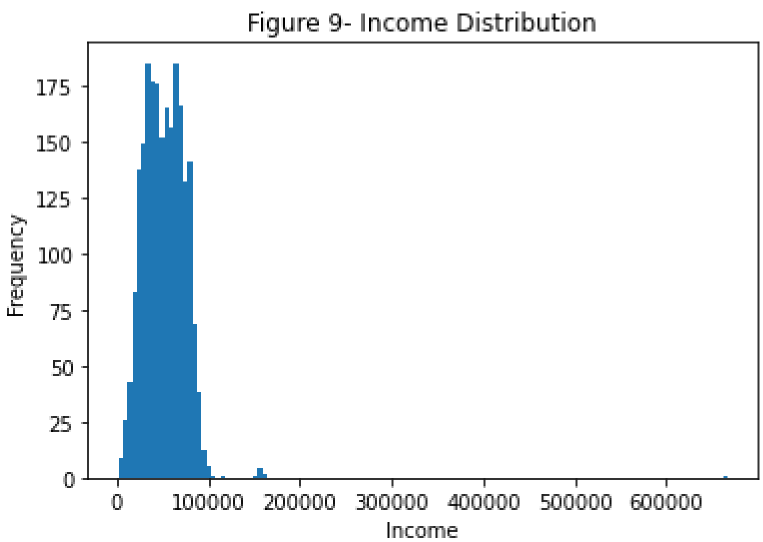
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['Marital_Status'], bins='auto'
)
plt.xlabel('Marital_Status')
plt.ylabel('Frequency')
plt.title('Figure 8- Marital Status Distribution')
```

Out[ ]: Text(0.5, 1.0, 'Figure 8- Marital Status Distribution')



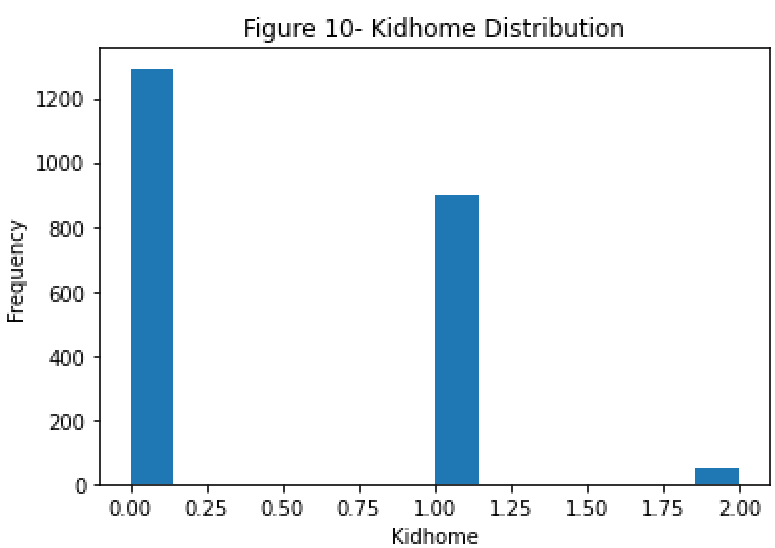
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['Income'], bins='auto')
plt.xlabel('Income')
plt.ylabel('Frequency')
plt.title('Figure 9- Income Distribution')
```

Out[ ]: `Text(0.5, 1.0, 'Figure 9- Income Distribution')`



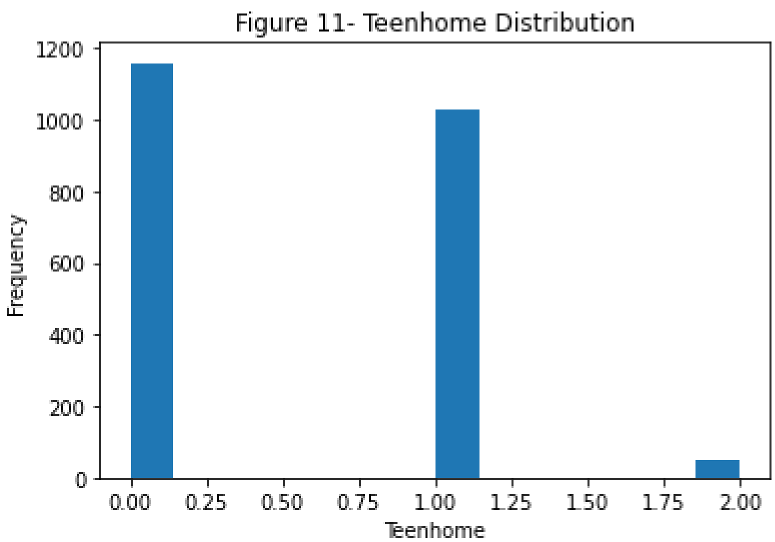
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['Kidhome'], bins='auto')
plt.xlabel('Kidhome')
plt.ylabel('Frequency')
plt.title('Figure 10- Kidhome Distribution')
```

Out[ ]: `Text(0.5, 1.0, 'Figure 10- Kidhome Distribution')`



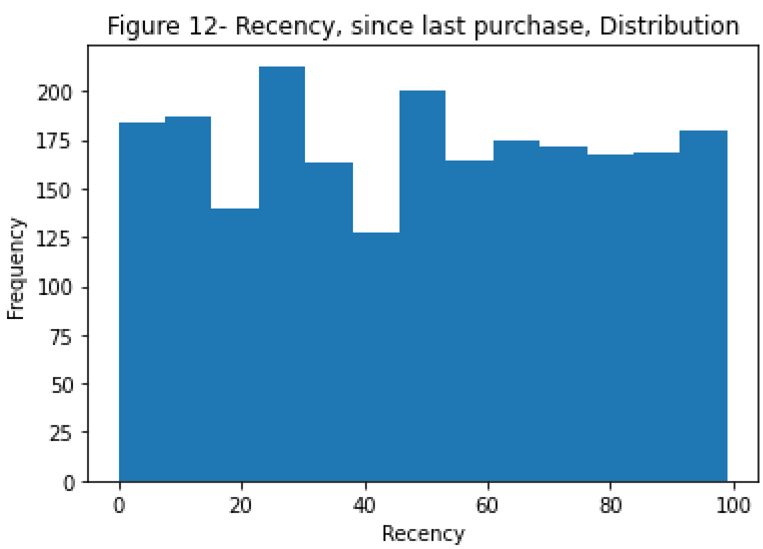
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['Teenhome'], bins='auto')
plt.xlabel('Teenhome')
plt.ylabel('Frequency')
plt.title('Figure 11- Teenhome Distribution')
```

Out[ ]: `Text(0.5, 1.0, 'Figure 11- Teenhome Distribution')`



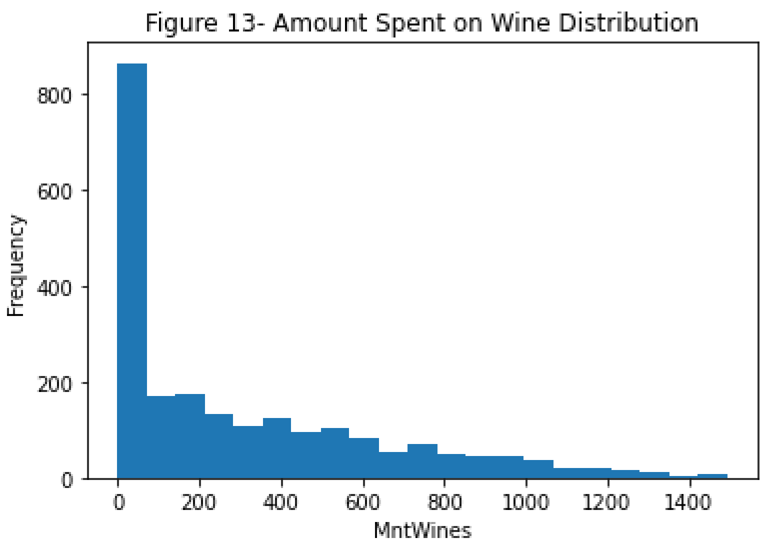
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['Recency'], bins='auto')
plt.xlabel('Recency')
plt.ylabel('Frequency')
plt.title('Figure 12- Recency, since last purchase, Distribution')
```

Out[ ]: `Text(0.5, 1.0, 'Figure 12- Recency, since last purchase, Distribution')`



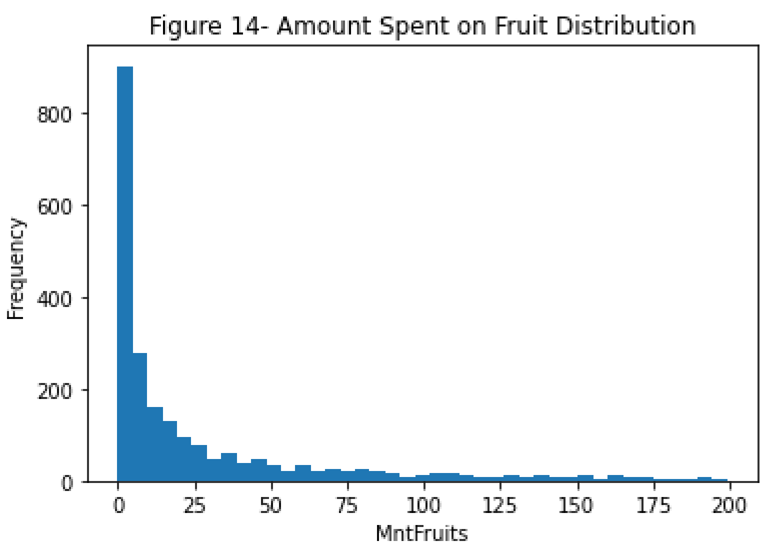
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['MntWines'], bins='auto')
plt.xlabel('MntWines')
plt.ylabel('Frequency')
plt.title('Figure 13- Amount Spent on Wine Distribution')
```

Out[ ]: Text(0.5, 1.0, 'Figure 13- Amount Spent on Wine Distribution')



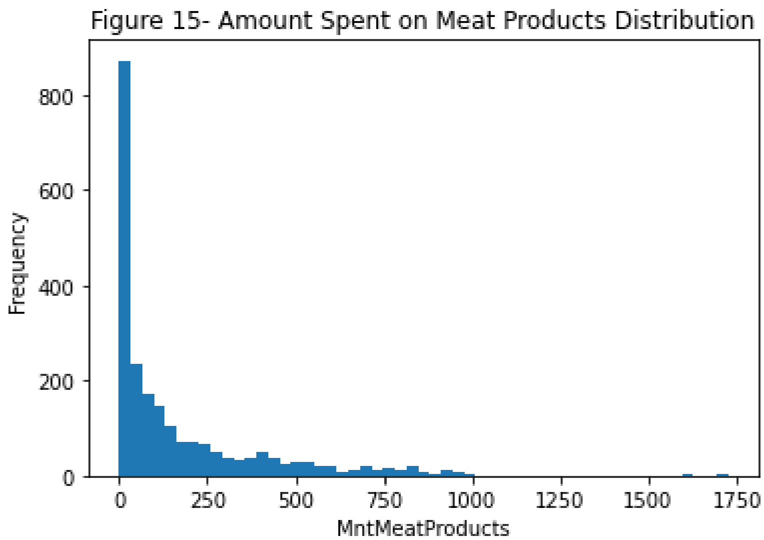
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['MntFruits'], bins='auto')
plt.xlabel('MntFruits')
plt.ylabel('Frequency')
plt.title('Figure 14- Amount Spent on Fruit Distribution')
```

Out[ ]: Text(0.5, 1.0, 'Figure 14- Amount Spent on Fruit Distribution')



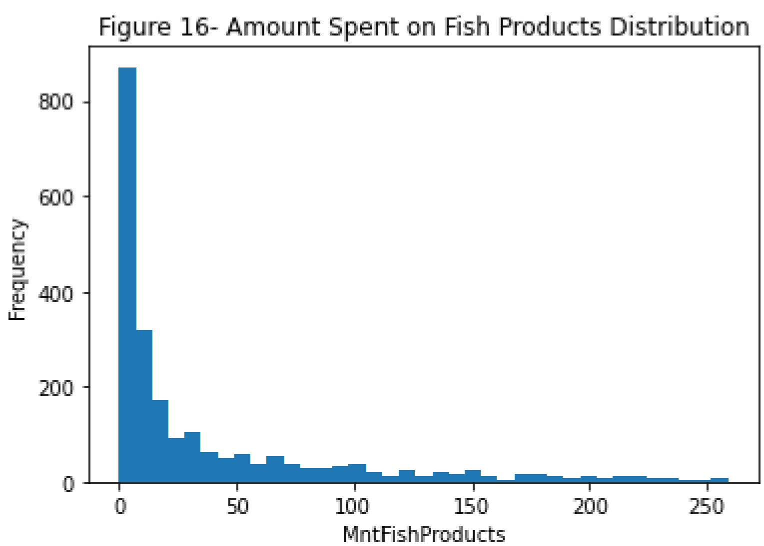
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['MntMeatProducts'], bins='auto')
plt.xlabel('MntMeatProducts')
plt.ylabel('Frequency')
plt.title('Figure 15- Amount Spent on Meat Products Distribution')
```

Out[ ]:
```
Text(0.5, 1.0, 'Figure 15- Amount Spent on Meat Products Distribution')
```



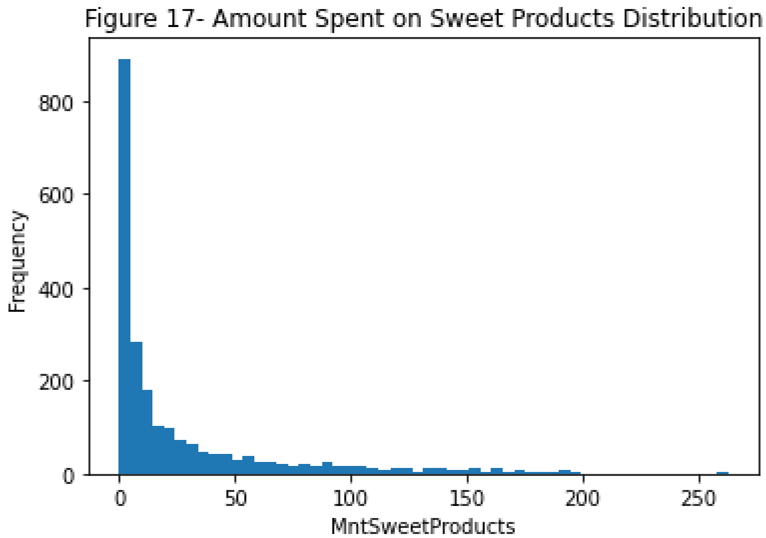
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['MntFishProducts'], bins='auto')
plt.xlabel('MntFishProducts')
plt.ylabel('Frequency')
plt.title('Figure 16- Amount Spent on Fish Products Distribution')
```

Out[ ]:
```
Text(0.5, 1.0, 'Figure 16- Amount Spent on Fish Products Distribution')
```



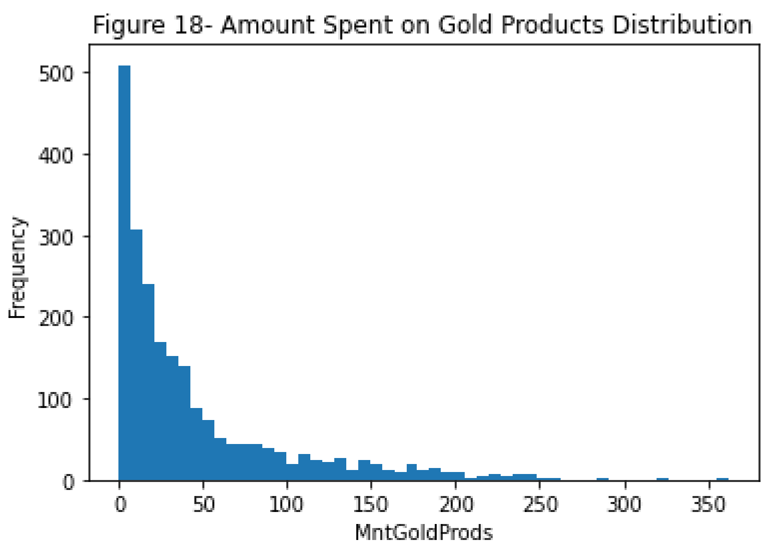
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['MntSweetProducts'], bins='aut
o')
plt.xlabel('MntSweetProducts')
plt.ylabel('Frequency')
plt.title('Figure 17- Amount Spent on Sweet Products Distribution')
```

Out[ ]: Text(0.5, 1.0, 'Figure 17- Amount Spent on Sweet Products Distribution')



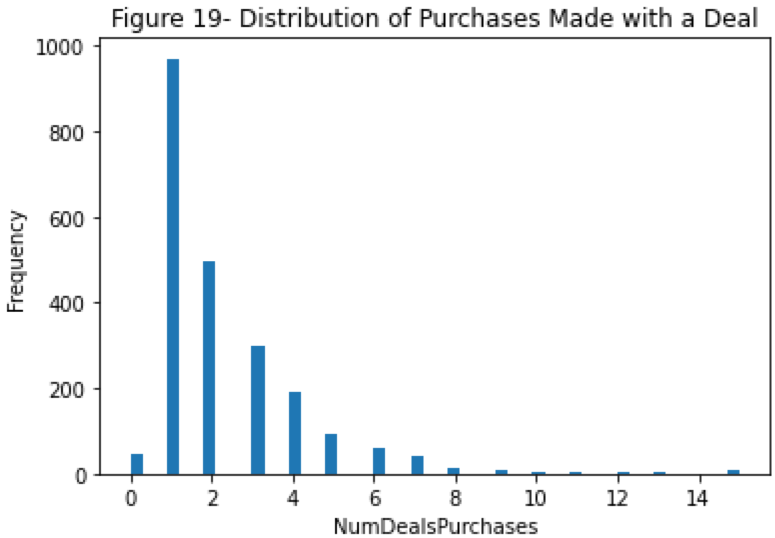
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['MntGoldProds'], bins='auto')
plt.xlabel('MntGoldProds')
plt.ylabel('Frequency')
plt.title('Figure 18- Amount Spent on Gold Products Distribution')
```

Out[ ]: Text(0.5, 1.0, 'Figure 18- Amount Spent on Gold Products Distribution')



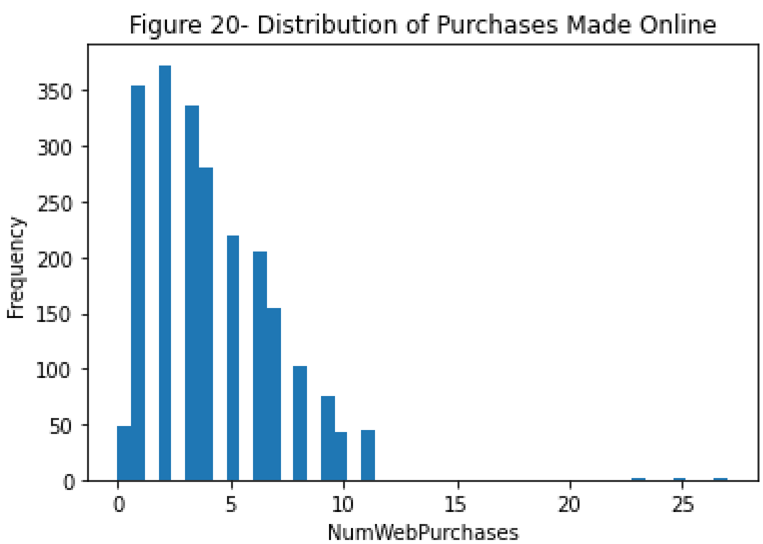
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['NumDealsPurchases'], bins='auto')
plt.xlabel('NumDealsPurchases')
plt.ylabel('Frequency')
plt.title('Figure 19- Distribution of Purchases Made with a Deal')
```

Out[ ]: Text(0.5, 1.0, 'Figure 19- Distribution of Purchases Made with a Deal')



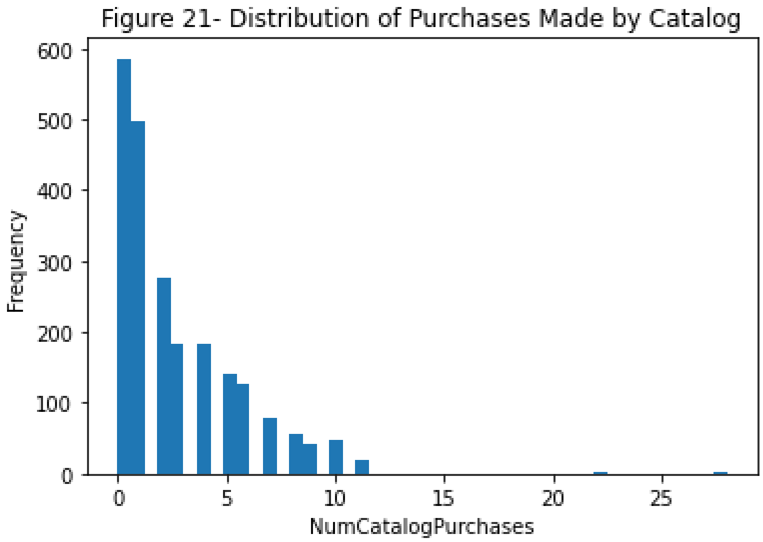
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['NumWebPurchases'], bins='auto')
plt.xlabel('NumWebPurchases')
plt.ylabel('Frequency')
plt.title('Figure 20- Distribution of Purchases Made Online')
```

Out[ ]: Text(0.5, 1.0, 'Figure 20- Distribution of Purchases Made Online')



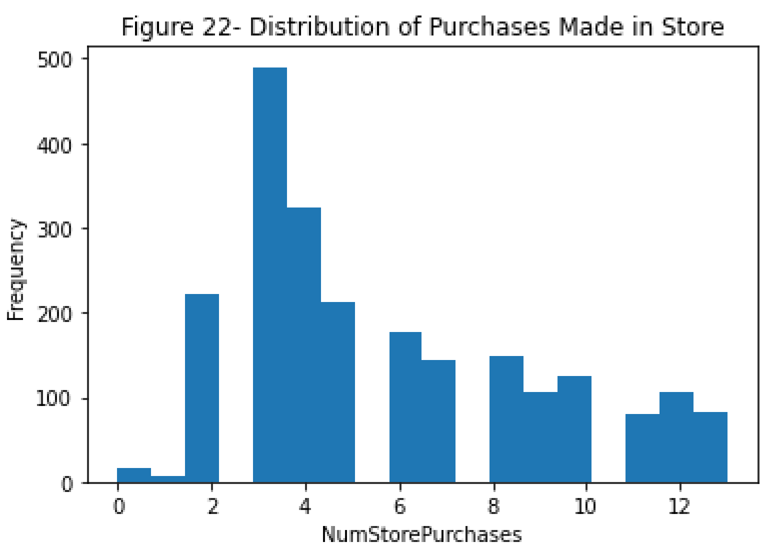
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['NumCatalogPurchases'], bins=
'auto')
plt.xlabel('NumCatalogPurchases')
plt.ylabel('Frequency')
plt.title('Figure 21- Distribution of Purchases Made by Catalog')
```

Out[ ]: Text(0.5, 1.0, 'Figure 21- Distribution of Purchases Made by Catalog')



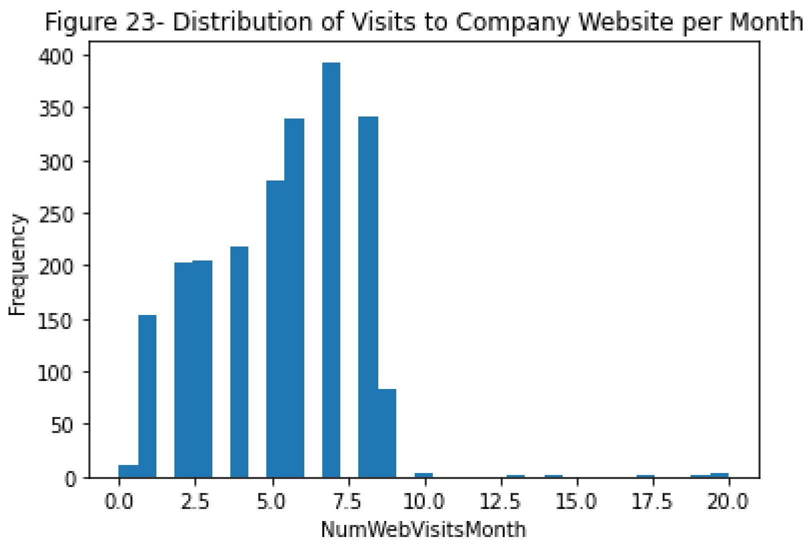
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['NumStorePurchases'], bins='au
to')
plt.xlabel('NumStorePurchases')
plt.ylabel('Frequency')
plt.title('Figure 22- Distribution of Purchases Made in Store')
```

Out[ ]: Text(0.5, 1.0, 'Figure 22- Distribution of Purchases Made in Store')



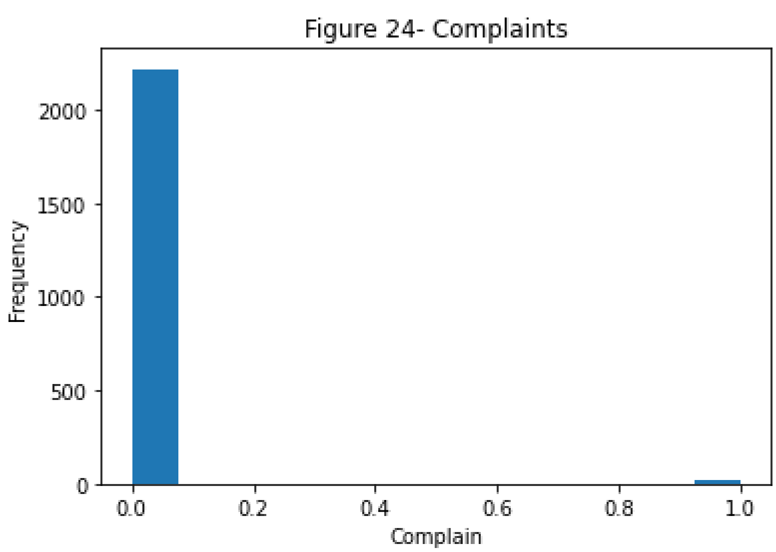
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['NumWebVisitsMonth'], bins='auto')
plt.xlabel('NumWebVisitsMonth')
plt.ylabel('Frequency')
plt.title('Figure 23- Distribution of Visits to Company Website per Month')
```

Out[ ]:
```
Text(0.5, 1.0, 'Figure 23- Distribution of Visits to Company Website per Month')
```



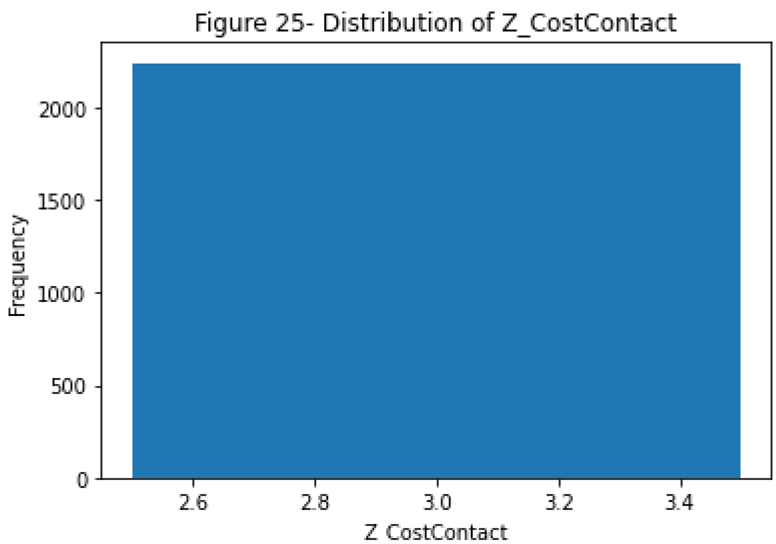
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['Complain'], bins='auto')
plt.xlabel('Complain')
plt.ylabel('Frequency')
plt.title('Figure 24- Complaints')
```

Out[ ]:
```
Text(0.5, 1.0, 'Figure 24- Complaints')
```



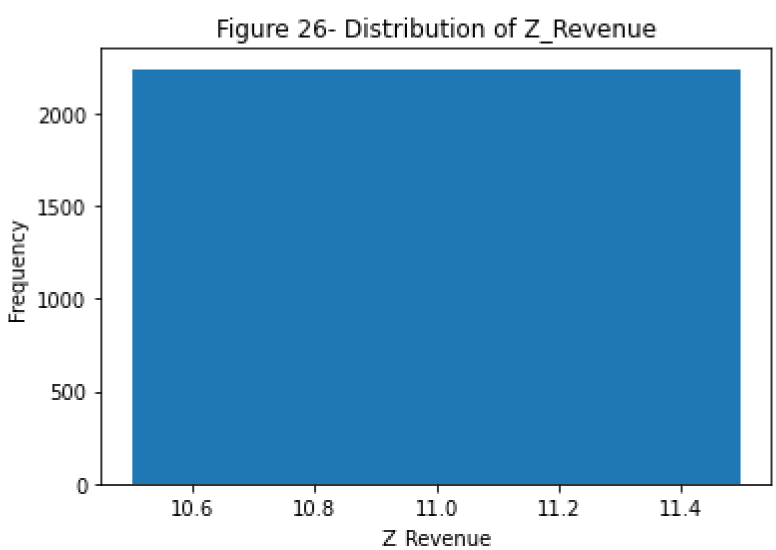
In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['Z_CostContact'], bins='auto')
plt.xlabel('Z_CostContact')
plt.ylabel('Frequency')
plt.title('Figure 25- Distribution of Z_CostContact')
```

Out[ ]:
```
Text(0.5, 1.0, 'Figure 25- Distribution of Z_CostContact')
```



In [ ]:
```
n, bins, patches = plt.hist(x=marketingCampaign['Z_Revenue'], bins='auto')
plt.xlabel('Z_Revenue')
plt.ylabel('Frequency')
plt.title('Figure 26- Distribution of Z_Revenue')
```

Out[ ]:
```
Text(0.5, 1.0, 'Figure 26- Distribution of Z_Revenue')
```

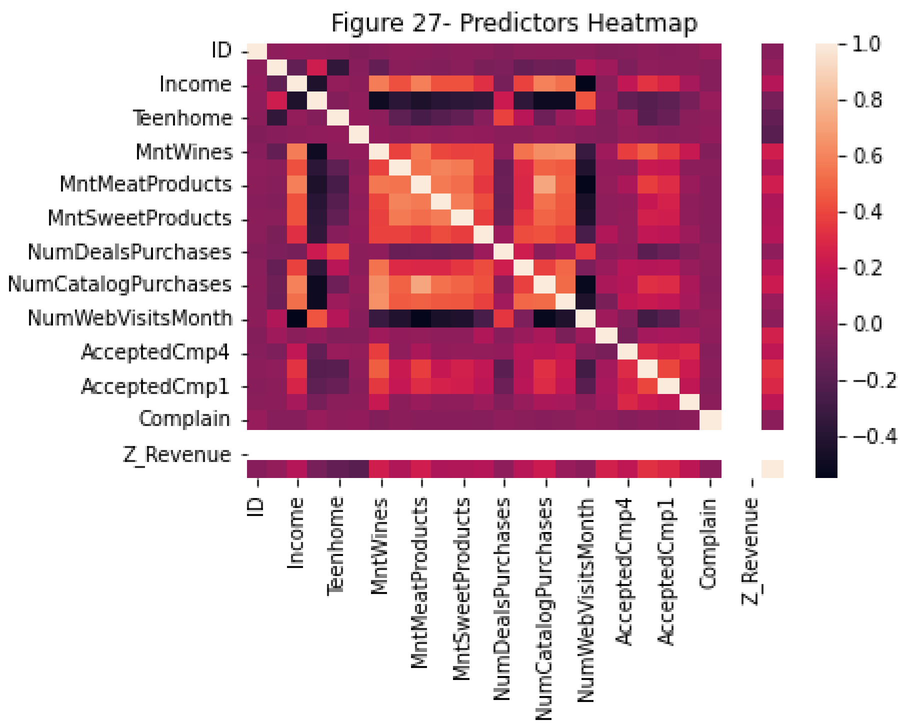


Examining Correlations and Collinearity

In [ ]:
```
correlation = marketingCampaign.corr()

ax = plt.axes()
heatmap = sns.heatmap(correlation, ax=ax)
ax.set_title('Figure 27- Predictors Heatmap')
heatmap
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe513d0eed0>



In [ ]:
```
# select upper triangle of correlation matrix
high = correlation.where(np.triu(np.ones(correlation.shape), k=1).astype(bool
))

# find features with correlation greater than 0.7
corrHigh = [column for column in high.columns if any(high[column] > 0.7)]
corrHigh
```

Out[ ]: ['NumCatalogPurchases']

Data Wrangling & Pre-Processing (handling missing values, outliers, correlated features, etc.)

Data wrangling and pre-processing is the process of transforming the dataset to get it ready for modeling purposes. This includes tasks like handling missing values, handling outliers, handling correlated features, feature engineering, etc. The first pre-processing step taken with the marketingCampaign dataset is to check for missing values and remove or impute if necessary. In the marketingCampaign dataset there are a total of 24 missing values, all within the 'Income' column. To handle these missing values, mean imputation is used. The second pre-processing step taken with the marketingCampaign dataset is to transform the 'Year_Birth' variable to an age variable. This will make the variable much more usable for modeling. To do this transformation, the current year (i.e., 2022) was subtracted from each row in the 'Year_Birth' column. The new column is called 'Age' and the old column 'Year_Birth' has been removed from the marketingCampaign dataframe. The third pre-processing step taken with the marketingCampaign dataset is to engineer a numerical variable, like days since enrollment, from the 'Dt_Customer' categorical variable. By engineering the date time categorical variable to a numerical variable in terms of number of days, the variable will be much more usable for modeling. To do this transformation, we first create a new column in the marketingCampaign dataframe called 'currentDate', which is today's date. Then we can create another new column called 'Days_Since_Enrollment' that is the 'currentDate' column subtracted by the original 'Dt_Customer' column, giving us a column of the number of days since the customer enrolled with the company. Given that we have engineered a new column from the 'Dt_Customer' column, both the 'Dt_Customer' column and the 'currentDate' column have been removed from the marketingCampaign dataframe. The fourth pre-processing step taken with the marketingCampaign dataset is to transform the two remaining categorical variables ('Education' and 'Marital_Status) to dummy variables. This is done by using the pd.get_dummie function. After the dummy variables have been created, the marketingCampaign dataframe increases to a total of 40 columns. The final pre-processing step taken with the marketingCampaign data is to scale the continuous predictors in the dataframe, using the StandardScaler function. The scaled predictors include: 'Income', 'Recency', 'MntWines', 'MntFruits', 'MntMeatProducts','MntFishProducts', 'MntSweetProducts', 'MntGoldProds','NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth', 'Age', and 'Days_Since_Enrollment'. The continuous predictors are scaled for better interpretability and to make sure no predictors is having a greater affect on the target than another, based on units. After scaling, the scaled predictor set, called marketingCampaignNorm, are concatenated back with the marketingCampaign dataframe. The original predictors are dropped from the marketingCampaignNorm dataframe, so only the scaled versions of those predictors remain. The final dataframe for modeling (marketingCampaignNorm) is comprised of 2240 rows and 40 columns.

Missing Value Imputation

In [ ]:
```
# checking for missing values
marketingCampaign.isna().sum()
```

Out[ ]:
```
ID                      0
Year_Birth              0
Education               0
Marital_Status          0
Income                 24
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
Complain                0
Z_CostContact           0
Z_Revenue               0
Response                0
dtype: int64
```

In [ ]:
```
# mean imputation for 24 missing Income values
marketingCampaign['Income'] = marketingCampaign['Income'].fillna(marketingCamp
aign['Income'].mean())
```

In [ ]:
```
# confirming that there are no missing values, after imputation
marketingCampaign.isna().sum()
```

Out[ ]:
```
ID                     0
Year_Birth             0
Education              0
Marital_Status         0
Income                 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
Complain               0
Z_CostContact          0
Z_Revenue              0
Response               0
dtype: int64
```

Transforming 'Year_Birth' to an age variable

In [ ]:
```
# creating a new column 'Age' from the 'Year_Birth' column
marketingCampaign['Age'] = (2022 - marketingCampaign.Year_Birth)

# removing the original 'Year_Birth' column
marketingCampaign = marketingCampaign.drop(columns=['Year_Birth'])
```

Transforming 'Dt_Customer' to 'Days_Since_Enrollment'

In [ ]:
```
# creating a column called 'currentDate' using today's date
marketingCampaign['currentDate'] = pd.to_datetime(date.today())

# creating a column called 'Days_Since_Enrollment'; subract 'currentDate' from
'Dt_Customer'
marketingCampaign['Days_Since_Enrollment'] = (marketingCampaign
                                              ['currentDate'] - pd.to_datetime
                                              (marketingCampaign['Dt_Customer'
])).astype('timedelta64[m]')

# removing 'currentDate' and 'Dt_Customer'
marketingCampaign = marketingCampaign.drop(columns = ['currentDate', 'Dt_Custo
mer'])
```

In [ ]:
```
# confirming new columns are in the dataframe
list(marketingCampaign.columns)
```

Out[ ]:
```
['ID',
 'Education',
 'Marital_Status',
 'Income',
 'Kidhome',
 'Teenhome',
 'Recency',
 'MntWines',
 'MntFruits',
 'MntMeatProducts',
 'MntFishProducts',
 'MntSweetProducts',
 'MntGoldProds',
 'NumDealsPurchases',
 'NumWebPurchases',
 'NumCatalogPurchases',
 'NumStorePurchases',
 'NumWebVisitsMonth',
 'AcceptedCmp3',
 'AcceptedCmp4',
 'AcceptedCmp5',
 'AcceptedCmp1',
 'AcceptedCmp2',
 'Complain',
 'Z_CostContact',
 'Z_Revenue',
 'Response',
 'Age',
 'Days_Since_Enrollment']
```

Transforming Categorical Variables to Dummies

In [ ]:
```
# transforming 'Education' and 'Marital_Status' to dummy variables
marketingCampaign = pd.get_dummies(marketingCampaign, columns = ['Education',
'Marital_Status'],
                                   prefix = ['Education', 'Marital_Status'])
```

Scaling Continuous Variables

In [ ]:
```
scaler = preprocessing.StandardScaler()
scaler.fit(marketingCampaign[['Income', 'Recency', 'MntWines', 'MntFruits', 'MntMeatProducts',
                              'MntFishProducts', 'MntSweetProducts', 'MntGoldProds',
                              'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases',
                              'NumStorePurchases', 'NumWebVisitsMonth', 'Age', 'Days_Since_Enrollment']])

# concatenating transformed predictors with original dataframe
marketingCampaignNorm = pd.concat([pd.DataFrame(scaler.transform(marketingCampaign[['Income', 'Recency',
'MntWines', 'MntFruits',
'MntMeatProducts',
'MntFishProducts',
'MntSweetProducts',
'MntGoldProds',
'NumDealsPurchases',
'NumWebPurchases',
'NumCatalogPurchases',
'NumStorePurchases',
'NumWebVisitsMonth',
'Age',
'Days_Since_Enrollment']]),
                                   columns = ['zIncome', 'zRecency', 'zMntWines', 'zMntFruits',
                                              'zMntMeatProducts', 'zMntFishProducts', 'zMntSweetProducts',
                                              'zMntGoldProds', 'zNumDealsPurchases', 'zNumWebPurchases',
                                              'zNumCatalogPurchases', 'zNumStorePurchases', 'zNumWebVisitsMonth',
                                              'zAge', 'zDays_Since_Enrollment']),
                                  marketingCampaign], axis = 1)
```

In [ ]:
```
# removing non-scaled predictors
marketingCampaignNorm = marketingCampaignNorm.drop(columns=['Income', 'Recenc
y', 'MntWines', 'MntFruits',
                                                            'MntMeatProducts',
'MntFishProducts', 'MntSweetProducts',
                                                            'MntGoldProds', 'N
umDealsPurchases', 'NumWebPurchases',
                                                            'NumCatalogPurchas
es', 'NumCatalogPurchases',
                                                            'NumStorePurchase
s', 'NumWebVisitsMonth', 'Age',
                                                            'Days_Since_Enroll
ment'])

display(marketingCampaignNorm)
```

	zIncome	zRecency	zMntWines	zMntFruits	zMntMeatProducts	zMntFishProducts	zMntSv
0	0.235327	0.307039	0.983781	1.551577	1.679702	2.462147	
1	-0.235826	-0.383664	-0.870479	-0.636301	-0.713225	-0.650449	
2	0.773633	-0.798086	0.362723	0.570804	-0.177032	1.345274	
3	-1.022732	-0.798086	-0.870479	-0.560857	-0.651187	-0.503974	
4	0.241519	1.550305	-0.389085	0.419916	-0.216914	0.155164	
...	...	...	...	...	...	...	
2235	0.358568	-0.107383	1.203678	0.419916	0.066692	0.081926	
2236	0.470064	0.237969	0.303291	-0.661449	-0.606873	-0.687068	
2237	0.189106	1.446700	1.795020	0.545656	0.221789	-0.101168	
2238	0.679035	-1.419719	0.368666	0.092992	0.208495	0.777683	
2239	0.024838	-0.314594	-0.653555	-0.586005	-0.469501	-0.650449	

2240 rows × 40 columns

Data Splitting (training and test sets)

Before partitioning the dataset, outcome and predictor sets are first created. The outcome variable is 'Response' from the marketingCampaignNorm dataframe and the predictors are the remaining 38 variables. The ID column has been removed for modeling. The predictor and outcome sets are then used to partition the data into trainX, validX, trainy, and validy train and validation sets for modeling. The data is partitioned using a 60/40 split and a random_state of 42. The training set is comprised of 1344 records, and the validation set is comprised of 896 records.

In [ ]:
```
# defining predictors and outcome variables for model
outcome = marketingCampaignNorm['Response']
predictors = marketingCampaignNorm[['zIncome', 'zRecency', 'zMntWines', 'zMntF
ruits', 'zMntMeatProducts',
                                   'zMntFishProducts', 'zMntSweetProducts', 'z
MntGoldProds', 'zNumDealsPurchases',
                                   'zNumWebPurchases', 'zNumCatalogPurchases',
'zNumStorePurchases', 'zAge',
                                   'zDays_Since_Enrollment', 'zNumWebVisitsMon
th', 'Kidhome', 'Teenhome',
                                   'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCm
p5', 'AcceptedCmp1', 'AcceptedCmp2',
                                   'Complain', 'Z_CostContact', 'Z_Revenue',
'Education_2n Cycle',
                                   'Education_Basic', 'Education_Graduation',
'Education_Master', 'Education_PhD',
                                   'Marital_Status_Absurd', 'Marital_Status_Al
one', 'Marital_Status_Divorced',
                                   'Marital_Status_Married', 'Marital_Status_S
ingle', 'Marital_Status_Together',
                                   'Marital_Status_Widow', 'Marital_Status_YOL
O']]

trainX, validX, trainy, validy = train_test_split(predictors, outcome, test_si
ze = 0.4, random_state = 42)

print(trainX.shape, validX.shape)
print(trainy.shape, validy.shape)
```

```
(1344, 38) (896, 38)
(1344,) (896,)
```

Model Strategies (describing main research questions and appropriate analytics methods)

In order to answer our research question of which customers are most likely to purchase something from the company, various classification algorithms will be used (i.e., logistic regression, random forest, decision tree, k-nn, adaBoost, LDA) to predict a response for the customers. A response of 1 indicates the customer has purchased something after receiving the marketing campaign, while a response of 0 indicates the customer has not purchased anything after receiving the marketing campaign. To answer our second research question of how to increase customer responses to the marketing campaigns, we will use our "best" model to create a list of customers that are the most likely to buy something from the company. By only sending the marketing campaign to this list of recommended customers, overall profits are likely to increase for two reasons. The first reason being that by only sending the marketing campaign to customers who are the most likely to make purchases, we will save money from the cost of sending the campaign to every customer. The second reason being that by sending direct-marketing to our already loyal customers, purchasing is likely to increase, increasing total profits.

Validation & Testing (model tuning and evaluation)

To evaluate our models, performance metrics like accuracy, precision, and recall, along with the confusion matrix will be viewed. The first modeling step taken is to perform a Logistic Regression, as the baseline model because of the visibility of the predictor coefficients. By viewing the model coefficients, the logistic regression can be used to create subsets of the predictors for computational efficiency. A logistic regression model was trained with the complete set of predictors and the performance was studied. The coefficients of the predictors were also studied to create a subset of predictors with a threshold coefficient of 0.5. There are 38 predictors and based on the threshold coefficient 18 predictors were subset and used to train a logistic regression and the performance was studied. We also worked with a third set of logistic regression with a trimmed set of predictors. The full set of predictors was trimmed by removing the zero variance and near zero variance predictors, using a variance < 0.01 threshold. Five predictors were removed for the trimmed set of the predictors and the rest of the predictors were used to train a logistic regression and the performance was studied. The accuracy, precision, and recall of the logistic regression with the full set of predictors, the subset of predictors, and the trimmed set of predictors was studied. The trimmed set had the maximum accuracy and recall, and the precision was better than full set of predictors and a little worse than the subset of predictors. So, the trimmed set of predictors, trimmed based on near zero variance, was chosen to be used to train the other models since it had a maximum recall score. For the remaining models, when necessary, tuning was performed to determine the optimal value for the model parameters.

A logistic regression model using the full set of predictors is trained as a baseline model to use it to compare the performance of the other models. The default l2 penalty is used which uses the square of the magnitude of the coefficients as in Ridge Regression. The inverse of regularization strength, C is set to 1e42. The coefficients of the predictor variables are also printed, so that the subset of the high coefficient variables will be used to train another model which will be computationally efficient if it outperforms the baseline model. After training the logistic regression model the confusion matrix, along with performance metrics are output. The baseline logistic regression performs with an accuracy of 88.73%, a precision of 65.85%, and recall of 42.52%, misclassifying a total of 101 records.

After training the full predictor set logistic regression, a subset of the predictors was used to check the performance of the logistic regression, to see a more computationally efficient model could be implemented. To define the subset, the coefficients of the predictors in the full logistic regression were used to determine predictor importance. The threshold was set to 0.5 for the coefficients of the predictors, and a subset of 18 predictors from the total 38 predictors was used to train another logistic regression model. The performance of the models with the subset of predictors and the full set of predictors will be compared. After training the subset logistic regression model the confusion matrix, along with performance metrics are output. The subset logistic regression performs with an accuracy of 88.84%, a precision of 69.56%, and recall of 37.79%, misclassifying a total of 100 records.

Even though the accuracy and the precision scores were higher than the full set baseline model, the recall value which gives us the score for the positive response outcome prediction is low compared to the full set of predictors. To try and increase the recall even further the number of predictors is trimmed from the full predictor set, by removing the near zero variance predictors. The highly correlated variables were also checked. Since it did not have impact on the models they were not removed. The six near zero variance predictors were removed from the predictors set and a logistic regression model is trained.

Baseline Logisitic Regression, using full set of predictors

In [ ]:
```
# logistic regression model with L2 penalty
lr_l2 = LogisticRegression(penalty = "l2", C = 1e42, solver = 'liblinear')
lr_l2.fit(trainX, trainy)
log_pred = lr_l2.predict_proba(validX)

# printing model coefficients
display(pd.DataFrame({'coeff': lr_l2.coef_[0]}, index = trainX.columns))

# confusion matrix to classify purcahsers and nonpurchasers
classificationSummary(validy, lr_l2.predict(validX))
print(f'precision: {precision_score(validy, lr_l2.predict(validX), zero_divisi
on=0)}')
print(f'recall: {recall_score(validy, lr_l2.predict(validX), zero_division=1)}
')
```

	coeff
zIncome	-0.309342
zRecency	-0.876463
zMntWines	-0.217176
zMntFruits	0.255229
zMntMeatProducts	0.411644
zMntFishProducts	-0.068736
zMntSweetProducts	0.065275
zMntGoldProds	0.342550
zNumDealsPurchases	0.151652
zNumWebPurchases	0.302752
zNumCatalogPurchases	0.039110
zNumStorePurchases	-0.521273
zAge	0.305550
zDays_Since_Enrollment	0.934272
zNumWebVisitsMonth	0.250871
Kidhome	-0.006177
Teenhome	-1.046165
AcceptedCmp3	2.123670
AcceptedCmp4	1.669377
AcceptedCmp5	2.548440
AcceptedCmp1	1.797676
AcceptedCmp2	1.842065
Complain	-0.703956
Z_CostContact	-0.055248
Z_Revenue	-0.202576
Education_2n Cycle	-0.216635
Education_Basic	-1.697260
Education_Graduation	0.076762
Education_Master	0.542637
Education_PhD	1.276080
Marital_Status_Absurd	3.288363
Marital_Status_Alone	1.030121
Marital_Status_Divorced	-0.266129
Marital_Status_Married	-1.361898
Marital_Status_Single	-0.100497

	coeff
Marital_Status_Together	-1.645916
Marital_Status_Widow	-0.223371
Marital_Status_YOLO	-0.739089

```
Confusion Matrix (Accuracy 0.8873)

       Prediction
Actual   0   1
     0 741  28
     1  73  54
precision: 0.6585365853658537
recall: 0.4251968503937008
```

In [ ]:
```
outcome = marketingCampaignNorm['Response']
subset_predictors = marketingCampaignNorm[['zRecency', 'zNumStorePurchases',
                                   'zDays_Since_Enrollment', 'Teenhome',
                                   'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCm
p5', 'AcceptedCmp1', 'AcceptedCmp2',
                                   'Complain', 'Education_Basic', 'Education_M
aster', 'Education_PhD',
                                   'Marital_Status_Absurd', 'Marital_Status_Al
one',
                                   'Marital_Status_Married', 'Marital_Status_T
ogether',
                                   'Marital_Status_YOLO']]

trainX, validX, trainy, validy = train_test_split(subset_predictors, outcome,
test_size = 0.4, random_state = 42)

print(trainX.shape, validX.shape)
print(trainy.shape, validy.shape)
```

```
(1344, 18) (896, 18)
(1344,) (896,)
```

In [ ]:
```
# subset logistic regression model with L2 penalty
lr_l2 = LogisticRegression(penalty = "l2", C = 1e42, solver = 'liblinear')
lr_l2.fit(trainX, trainy)
log_pred = lr_l2.predict_proba(validX)

# printing model coefficients
display(pd.DataFrame({'coeff': lr_l2.coef_[0]}, index = trainX.columns))

# confusion matrix to classify purcahsers and nonpurchasers

print(classificationSummary(validy, lr_l2.predict(validX)))
print(f'precision: {precision_score(validy, lr_l2.predict(validX), zero_divisi
on=0)}')
print(f'recall: {recall_score(validy, lr_l2.predict(validX), zero_division=1)}
')
```

	coeff
zRecency	-0.882413
zNumStorePurchases	-0.292887
zDays_Since_Enrollment	1.041889
Teenhome	-0.833360
AcceptedCmp3	2.061267
AcceptedCmp4	1.130299
AcceptedCmp5	2.420533
AcceptedCmp1	1.891469
AcceptedCmp2	1.357988
Complain	-1.348146
Education_Basic	-2.194453
Education_Master	0.273972
Education_PhD	0.926758
Marital_Status_Absurd	3.348851
Marital_Status_Alone	1.086548
Marital_Status_Married	-1.221940
Marital_Status_Together	-1.350169
Marital_Status_YOLO	-0.591629

```
Confusion Matrix (Accuracy 0.8884)

       Prediction
Actual   0   1
     0 748  21
     1  79  48
None
precision: 0.6956521739130435
recall: 0.3779527559055118
```

Assessing Near Zero Variance Predictors

In [ ]:
```
# Removing near zero variance predictors using threshold of 0.01
var_thr = VarianceThreshold(threshold = 0.01)
var_thr.fit(trainX)

var_thr.get_support()

concol = [column for column in trainX.columns
          if column not in trainX.columns[var_thr.get_support()]]

for features in concol:
    print(features)
```

```
Complain
Marital_Status_Absurd
Marital_Status_Alone
Marital_Status_YOLO
```

Trimmed Logistic Regression Model (removed variables with near zero variance-0.01 variance)

In [ ]:
```
# defining predictors and outcome variables for model
outcomeTrim = marketingCampaignNorm['Response']
predictorsTrim = marketingCampaignNorm[['zIncome', 'zRecency', 'zMntWines', 'z
MntFruits', 'zMntMeatProducts',
                                   'zMntFishProducts', 'zMntSweetProducts', 'z
MntGoldProds', 'zNumDealsPurchases',
                                   'zNumWebPurchases', 'zNumCatalogPurchases',
'zNumStorePurchases', 'zAge',
                                   'zDays_Since_Enrollment', 'zNumWebVisitsMon
th', 'Kidhome', 'Teenhome',
                                   'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCm
p5', 'AcceptedCmp1', 'AcceptedCmp2',
                                   'Education_2n Cycle',
                                   'Education_Basic', 'Education_Graduation',
'Education_Master', 'Education_PhD',
                                   'Marital_Status_Divorced',
                                   'Marital_Status_Married', 'Marital_Status_S
ingle', 'Marital_Status_Together',
                                   'Marital_Status_Widow']]

trainXTrim, validXTrim, trainyTrim, validyTrim = train_test_split(predictorsTr
im, outcomeTrim, test_size = 0.4, random_state = 42)

print(trainXTrim.shape, validXTrim.shape)
print(trainyTrim.shape, validyTrim.shape)
```

```
(1344, 32) (896, 32)
(1344,) (896,)
```

In [ ]:
```
# trimmed logistic regression model with L2 penalty
lr_l2Trim = LogisticRegression(penalty = "l2", C = 1e42, solver = 'liblinear')
lr_l2Trim.fit(trainXTrim, trainyTrim)
log_predTrim = lr_l2Trim.predict_proba(validXTrim)

# printing model coefficients
display(pd.DataFrame({'coeff': lr_l2Trim.coef_[0]}, index = trainXTrim.columns
))

# confusion matrix to classify purchasers and nonpurchasers
print(classificationSummary(validyTrim, lr_l2Trim.predict(validXTrim)))
print(f'precision: {precision_score(validyTrim, lr_l2Trim.predict(validXTrim),
zero_division=0)}')
print(f'recall: {recall_score(validyTrim, lr_l2Trim.predict(validXTrim), zero_
division=1)}')
```

	coeff
zIncome	-0.310634
zRecency	-0.868815
zMntWines	-0.208934
zMntFruits	0.258565
zMntMeatProducts	0.408514
zMntFishProducts	-0.064590
zMntSweetProducts	0.063095
zMntGoldProds	0.344315
zNumDealsPurchases	0.149434
zNumWebPurchases	0.303105
zNumCatalogPurchases	0.040715
zNumStorePurchases	-0.526027
zAge	0.309143
zDays_Since_Enrollment	0.923352
zNumWebVisitsMonth	0.244063
Kidhome	0.025225
Teenhome	-1.045022
AcceptedCmp3	2.122387
AcceptedCmp4	1.671133
AcceptedCmp5	2.528473
AcceptedCmp1	1.800485
AcceptedCmp2	1.837479
Education_2n Cycle	-0.590187
Education_Basic	-2.043790
Education_Graduation	-0.270771
Education_Master	0.201214
Education_PhD	0.912592
Marital_Status_Divorced	-0.542135
Marital_Status_Married	-1.637452
Marital_Status_Single	-0.378987
Marital_Status_Together	-1.919335
Marital_Status_Widow	-0.498386

```
Confusion Matrix (Accuracy 0.8895)

       Prediction
Actual   0   1
     0 742  27
     1  72  55
None
precision: 0.6707317073170732
recall: 0.4330708661417323
```

Random Forest Model

In [ ]:
```
rf = RandomForestClassifier()
rf.fit(trainXTrim, trainyTrim)

rf_pred = rf.predict(validXTrim)
rf_predprob = rf.predict_proba(validXTrim)[:,1]
print(classificationSummary(validyTrim, rf_pred))
print(f'precision: {precision_score(validyTrim, rf_pred, zero_division=0)}')
print(f'recall: {recall_score(validyTrim, rf_pred, zero_division=1)}')
```

```
Confusion Matrix (Accuracy 0.8728)

       Prediction
Actual   0   1
     0 751  18
     1  96  31
None
precision: 0.6326530612244898
recall: 0.2440944881889764
```

Decision Tree Model

In [ ]:
```
dt = DecisionTreeClassifier(criterion = "gini", max_depth=3, random_state=1)
dt.fit(trainXTrim, trainyTrim)

dt_pred = dt.predict(validXTrim)
dt_predprob = dt.predict_proba(validXTrim)[:,1]
print(classificationSummary(validyTrim, dt_pred))
print(f'precision: {precision_score(validyTrim, dt_pred, zero_division=0)}')
print(f'recall: {recall_score(validyTrim, dt_pred, zero_division=1)}')
```

```
Confusion Matrix (Accuracy 0.8683)

       Prediction
Actual   0   1
     0 745  24
     1  94  33
None
precision: 0.5789473684210527
recall: 0.25984251968503935
```

k-NN Model

A KNN loop is run to check the optimum K based on the recall score.

In [ ]:
```
for r in range (1, 10):
    knn = KNeighborsClassifier(n_neighbors=r)
    knn.fit(trainXTrim, trainyTrim)

    y_pred = knn.predict(validXTrim)

    print(f'neighbors: {r}')
    print(f'accuracy: {accuracy_score(validyTrim, y_pred)}')
    print(f'precision: {precision_score(validyTrim, y_pred, zero_division=0)}'
)
    print(f'recall: {recall_score(validyTrim, y_pred, zero_division=1)}')
    print('\n')
```

```
neighbors: 1
accuracy: 0.8359375
precision: 0.40384615384615385
recall: 0.33070866141732286


neighbors: 2
accuracy: 0.8549107142857143
precision: 0.4444444444444444
recall: 0.09448818897637795


neighbors: 3
accuracy: 0.8482142857142857
precision: 0.4262295081967213
recall: 0.2047244094488189


neighbors: 4
accuracy: 0.8571428571428571
precision: 0.4827586206896552
recall: 0.11023622047244094


neighbors: 5
accuracy: 0.8616071428571429
precision: 0.5319148936170213
recall: 0.1968503937007874


neighbors: 6
accuracy: 0.8616071428571429
precision: 0.5789473684210527
recall: 0.08661417322834646


neighbors: 7
accuracy: 0.859375
precision: 0.5151515151515151
recall: 0.13385826771653545


neighbors: 8
accuracy: 0.8549107142857143
precision: 0.38461538461538464
recall: 0.03937007874015748


neighbors: 9
accuracy: 0.8571428571428571
precision: 0.47368421052631576
recall: 0.07086614173228346
```

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(trainXTrim, trainyTrim)

knn_pred = knn.predict(validXTrim)
knn_predprob = knn.predict_proba(validXTrim)[:,1]
print(classificationSummary(validyTrim, knn_pred))
print(f'precision: {precision_score(validyTrim, knn_pred, zero_division=0)}')
print(f'recall: {recall_score(validyTrim, knn_pred, zero_division=1)}')
```

```
Confusion Matrix (Accuracy 0.8359)

       Prediction
Actual   0   1
     0 707  62
     1  85  42
None
precision: 0.40384615384615385
recall: 0.33070866141732286
```

AdaBoost Model

In [ ]:
```
ada_boost = AdaBoostClassifier()
ada_boost.fit(trainXTrim, trainyTrim)
ada_pred = ada_boost.predict(validXTrim)
ada_predprob = ada_boost.predict_proba(validXTrim)[:,1]
print(classificationSummary(validyTrim, ada_pred))
print(f'precision: {precision_score(validyTrim, ada_pred, zero_division=0)}')
print(f'recall: {recall_score(validyTrim, ada_pred, zero_division=1)}')
```

```
Confusion Matrix (Accuracy 0.8817)

       Prediction
Actual   0   1
     0 737  32
     1  74  53
None
precision: 0.6235294117647059
recall: 0.41732283464566927
```

Linear Discriminant Analysis (LDA)

In [ ]:
```
lda = LinearDiscriminantAnalysis()
lda.fit(trainXTrim, trainyTrim)
lda_pred = lda.predict(validXTrim)
lda_probs = lda.predict_proba(validXTrim)[:,1]
print(classificationSummary(validyTrim, lda_pred))
print(f'precision: {precision_score(validyTrim, lda_pred, zero_division=0)}')
print(f'recall: {recall_score(validyTrim, lda_pred, zero_division=1)}')
```

```
Confusion Matrix (Accuracy 0.8862)

       Prediction
Actual   0   1
     0 738  31
     1  71  56
None
precision: 0.6436781609195402
recall: 0.4409448818897638
```

Results & Final Model Selection (performance measures, etc.)

After training the trimmed logistic regression model, the confusion matrix, along with performance metrics are output. The trimmed logistic regression model performs with an accuracy of 88.95%, precision of 67.07%, and recall of 43.31%, misclassifying a total of 99 records. Given that the accuracy and the recall scores were the highest for the trimmed set of the predictors compared to the full set and the subset of the predictor models, the trimmed set of predictors are used for the rest of the modeling.

After deciding on the final set of predictors for modeling, a random forest, decision tree, k-nn, adaBoost, and LDA models are trained. The random forest model performs with an accuracy of 87.05%, a precision of 61.22%, and recall of 23.62%, misclassifying a total of 106 records. The decision tree model performs with an accuracy of 86.83%, a precision of 57.89%, and a recall of 25.98%, misclassifying a total of 118 records. The k-NN model is trained using a k value of 1, performing with an accuracy of 83.59%, a precision of 40.38%, and recall of 33.07%, misclassifying a total of 147 records. The adaBoost model performs with an accuracy of 88.17%, a precision of 62.35%, and a recall of 41.73%, misclassifying a total of 106 records. The last model trained is the LDA model, performing with an accuracy of 88.62%, a precision of 64.37%, and a recall of 44.09%, misclassifying a total of 102 records. From Figure 28- Gains Chart for All Models, we can clearly see that the k-NN models performs significantly worse than the other models, while the rest of the models performed similarly in terms of cumulative gains. Given these results, our "best" model, the trimmed logistic regression, is chosen. The trimmed logistic regression is deemed to be the "best" model because it classifies the most records correctly, had a higher recall rate compared to the other models, and is interpretable compared to the other models.

In [ ]:
```
log_df = pd.DataFrame({'actual': validyTrim,
                       'p(1)': [p[1] for p in log_predTrim],
                       'predicted': lr_l2Trim.predict(validXTrim)
                       })

rf_df = pd.DataFrame({'actual': validyTrim,
                      'p(1)': rf_predprob,
                      'predicted': rf_pred,
                       })
dt_df = pd.DataFrame({'actual': validyTrim,
                      'p(1)': dt_predprob,
                      'predicted': dt_pred,
                       })
knn_df = pd.DataFrame({'actual': validyTrim,
                      'p(1)': knn_predprob,
                      'predicted': knn_pred,
                       })
ada_df = pd.DataFrame({'actual': validyTrim,
                      'p(1)': ada_predprob,
                      'predicted': ada_pred,
                       })
lda_df = pd.DataFrame({'actual': validyTrim,
                      'p(1)': lda_probs,
                      'predicted': lda_pred,
                       })
```

In [ ]:
```
log_df = log_df.sort_values(by=['p(1)'], ascending=False)
rf_df = rf_df.sort_values(by=['p(1)'], ascending=False)
dt_df = ada_df.sort_values(by=['p(1)'], ascending=False)
knn_df = knn_df.sort_values(by=['p(1)'], ascending=False)
ada_df = ada_df.sort_values(by=['p(1)'], ascending=False)
lda_df = lda_df.sort_values(by=['p(1)'], ascending=False)
```

In [ ]:
```
ax = gainsChart(log_df.actual, label='Logistic_Regression', color='C0', figsiz
e=[5, 5])
ax = gainsChart(rf_df.actual, label='Random_Forest', color='C1', ax=ax)
ax = gainsChart(dt_df.actual, label='Decision_Tree', color='C2', ax=ax)
ax = gainsChart(knn_df.actual, label='KNN', color='C3', ax=ax)
ax = gainsChart(ada_df.actual, label='Ada_Boost', color='C4', ax=ax)
ax = gainsChart(lda_df.actual, label='LDA', color='C5', ax=ax)
ax.legend()
ax.set_title('Figure 28- Gains Chart For All Models')
plt.show()
```

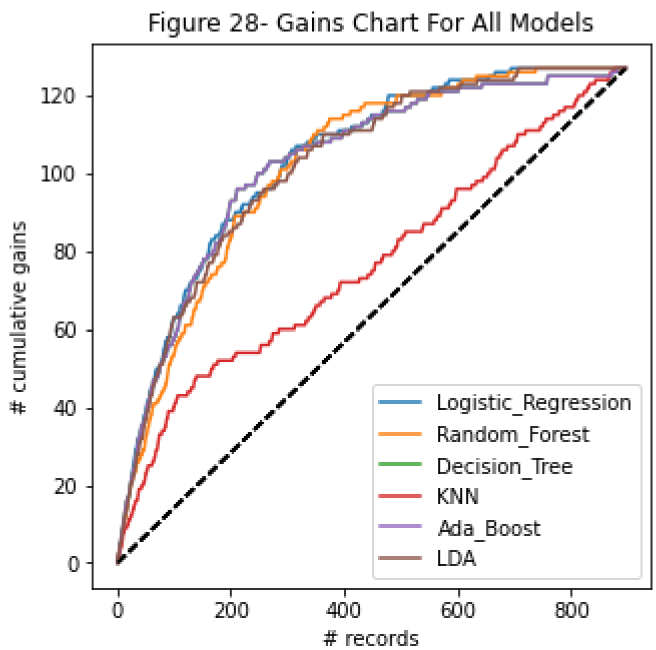


Table 1- The final performance metric table

	Accuracy	Precision	Recall
Logistic Regression	88.95	67.07	43.31
Random Forest	87.05	61.22	23.62
Decision Trees	86.83	57.89	25.98
KNN	83.59	40.38	33.07
Ada Boost	88.17	62.35	41.73
LDA	88.62	64.37	44.09

Discussion & Conclusions (address the problem statement and suggestions that could go beyond the scope of the course)

Our two main research questions are first, identifying customers who are likely to respond well to a marketing campaign, and second, how to increase customer responses to the marketing campaign. Using our "best" model, the trimmed logistic regression, we have identified a list of customers from the original list of 2240 customers, who are the most likely to make purchases after being sent the marketing campaign. Using a threshold level of 0.9, 65 customers are identified as likely to respond well to the marketing campaign and make purchases from the company. It is recommended for the future, to send the marketing campaign only to the provided list of 65 customers, instead of the whole list of 2240 customers. By sending the marketing campaign only to the provided list of 65 customers, there will be an expected profit of 520 dollars. When the marketing campaign is sent to all 2240 customers, there is a loss of 3,046 dollars. So, by sending only to the recommended list of customers, 3,046 dollars are saved, increasing overall annual revenue for the company to 520 dollars. Depending on the allotted funds given each year for the marketing campaigns, a lower threshold like 0.7 can be used instead of 0.9. By sending to the 144 customers identified using a 0.7 threshold, we can access a larger list of customers who are somewhat likely to respond to the marketing campaign, and have a profit of 1,152 dollars.

In [ ]:
```
# using full dataset of 2240 records to make predictions with trimmed logistic regression
predictors = marketingCampaignNorm[['zIncome', 'zRecency', 'zMntWines', 'zMntFruits', 'zMntMeatProducts',
                                   'zMntFishProducts', 'zMntSweetProducts', 'zMntGoldProds', 'zNumDealsPurchases',
                                   'zNumWebPurchases', 'zNumCatalogPurchases', 'zNumStorePurchases', 'zAge',
                                   'zDays_Since_Enrollment', 'zNumWebVisitsMonth', 'Kidhome', 'Teenhome',
                                   'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1', 'AcceptedCmp2',
                                   'Education_2n Cycle',
                                   'Education_Basic', 'Education_Graduation', 'Education_Master', 'Education_PhD',
                                   'Marital_Status_Divorced',
                                   'Marital_Status_Married', 'Marital_Status_Single', 'Marital_Status_Together',
                                   'Marital_Status_Widow']]

predictions = pd.DataFrame(lr_l2Trim.predict_proba(predictors)[:, 1])
display(predictions)
# filtering for predictions above a 0.7 threshold
predictions_high = predictions[predictions[0] >= 0.70]
display(predictions_high)

predictions_veryHigh = predictions[predictions[0] >= 0.90]
display(predictions_veryHigh)
```

	0
0	0.747411
1	0.016345
2	0.014370
3	0.008667
4	0.006951
...	...
2235	0.078811
2236	0.050922
2237	0.005565
2238	0.008983
2239	0.161340

2240 rows × 1 columns

	0
0	0.747411
15	0.973860
21	0.844960
27	0.975802
39	0.913456
...	...
2175	0.977574
2177	0.724105
2193	0.880808
2202	0.724105
2221	0.868146

144 rows × 1 columns

	0
15	0.973860
27	0.975802
39	0.913456
155	0.920828
203	0.939142
...	...
1940	0.903000
1961	0.996933
2093	0.943536
2167	0.994592
2175	0.977574

65 rows × 1 columns

In [ ]:
```
# expected profit of sending to 65 customers who are likely to respond
# cost of marketing campaign = $3
# expected revenue from people who purcahse something after getting the camapa
ign = $11
# original dataset has 334 yes responses out of 2240

print("Expected profit of sending to entire list of customers is", (11*334)-(2
240*3))
print("Expected profit of sending to most likely to respond customers is", (11
*65)-(65*3))
print("Expected profit of sending to somewhat likely to respond customers is",
(11*144)-(144*3))
```

```
Expected profit of sending to entire list of customers is -3046
Expected profit of sending to most likely to respond customers is 520
Expected profit of sending to somewhat likely to respond customers is 1152
```

References

Garg, S. (2022, July 12). Dropping Constant Features using VarianceThreshold: Feature Selection -1. Medium. Retrieved October 15, 2022, from https://medium.com/nerd-for-tech/removing-constant-variables-feature-selection-463e2d6a30d9 (https://medium.com/nerd-for-tech/removing-constant-variables-feature-selection-463e2d6a30d9)

How to calculate correlation between all columns and remove highly correlated ones using pandas? (2015, March 27). Stack Overflow. Retrieved October 15, 2022, from https://stackoverflow.com/questions/29294983/how-to-calculate-correlation-between-all-columns-and-remove-highly-correlated-on (https://stackoverflow.com/questions/29294983/how-to-calculate-correlation-between-all-columns-and-remove-highly-correlated-on)

Marketing Campaign. (2020, May 8). Kaggle. Retrieved October 14, 2022, from https://www.kaggle.com/datasets/rodsaldanha/arketing-campaign (https://www.kaggle.com/datasets/rodsaldanha/arketing-campaign)