Stanford University Summer 2024

DataSci 112: Principles of Data Science

Lab 6

Due Monday, August 19, 7:00pm

Note: the deadline is **hard**, submission will be closed after the time indicated above.

Before You Do Anything Else!!!

This is the Instructor's version of this notebook. **You need to save a copy of this notebook** to your personal Google Drive. To do this, go to File > Save a copy in Drive.

Name:Janmay Panchal

Email:janmayp@stanford.edu

Discussion Section: 3

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

### Add any necessary import statements below
```

In [2]:
```
filename = "http://www.csc.calpoly.edu/~dekhtyar/301-Winter2024/data/market:

df = pd.read_csv(filename, delimiter = "\t")
df  = df.set_index("ID")
```

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

Out[3]:

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

2240 rows × 28 columns

In [4]:

```
df.columns
```

Out[4]:

```
Index(['Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome',
       'Teenhome', 'Dt_Customer', 'Recency', 'MntWines', 'MntFruits',
       'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
       'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
       'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth',
       'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1',
       'AcceptedCmp2', 'Complain', 'Z_CostContact', 'Z_Revenue', 'Respons
e'],
      dtype='object')
```

Dataset Description

This dataset is available on Kaggle

The dataset documents customer profiles for a set of 2240 customers of a company. The dataset has been constructed for the use by the company's marketing department, and of primary interest to the company here is the customer segmentation (in other words, clustering of customers into different groups). We, however, will use this dataset for several other purposes.

Attributes

People

- `ID` : Customer's unique identifier
- `Year_Birth` : Customer's birth year
- `Education` : Customer's education level
- `Marital_Status` : Customer's marital status
- `Income` : Customer's yearly household income

- `Kidhome` : Number of children in customer's household
- `Teenhome` : Number of teenagers in customer's household
- `Dt_Customer` : Date of customer's enrollment with the company
- `Recency` : Number of days since customer's last purchase
- `Complain` : 1 if the customer complained in the last 2 years, 0 otherwise

Products

- `MntWines` : Amount spent on wine in last 2 years
- `MntFruits` : Amount spent on fruits in last 2 years
- `MntMeatProducts` : Amount spent on meat in last 2 years
- `MntFishProducts` : Amount spent on fish in last 2 years
- `MntSweetProducts` : Amount spent on sweets in last 2 years
- `MntGoldProds` : Amount spent on gold in last 2 years

Promotion

- `NumDealsPurchases` : Number of purchases made with a discount
- `AcceptedCmp1` : 1 if customer accepted the offer in the 1st campaign, 0 otherwise
- `AcceptedCmp2` : 1 if customer accepted the offer in the 2nd campaign, 0 otherwise
- `AcceptedCmp3` : 1 if customer accepted the offer in the 3rd campaign, 0 otherwise
- `AcceptedCmp4` : 1 if customer accepted the offer in the 4th campaign, 0 otherwise
- `AcceptedCmp5` : 1 if customer accepted the offer in the 5th campaign, 0 otherwise
- `Response:` 1 if customer accepted the offer in the last campaign, 0 otherwise

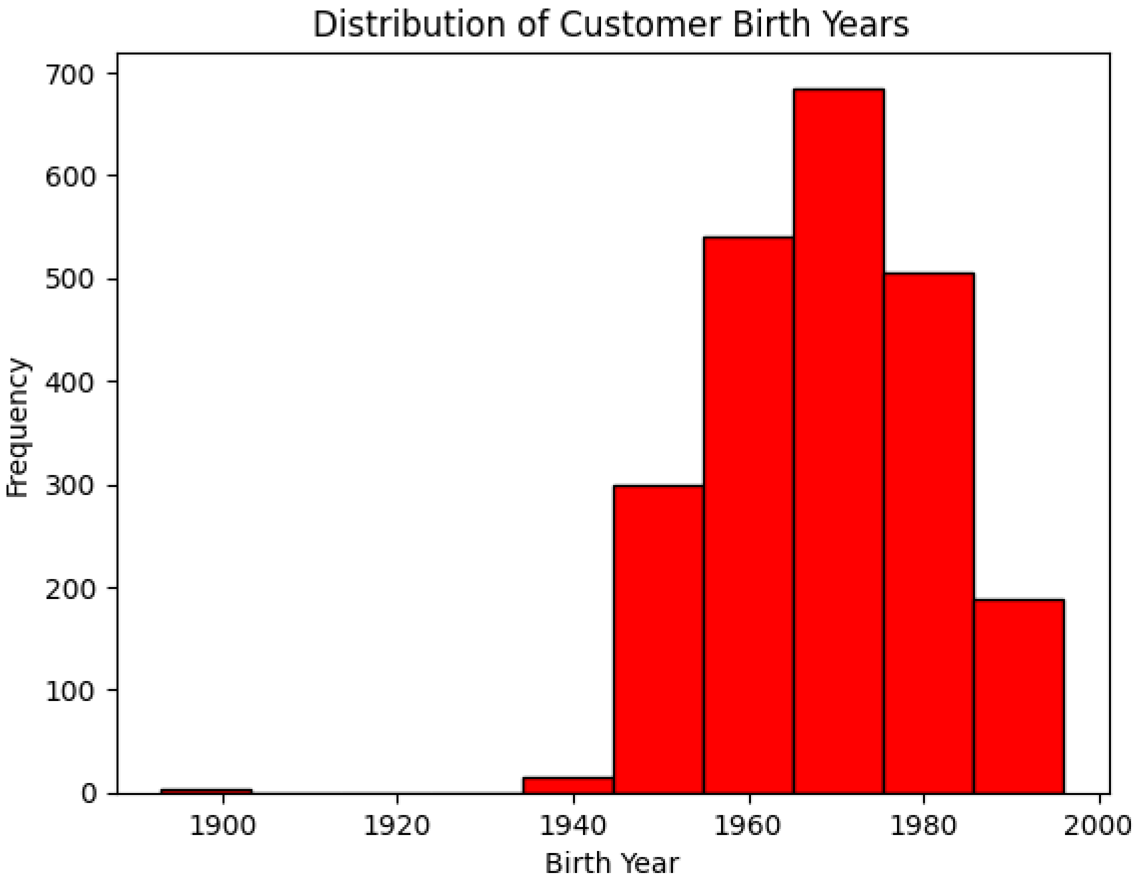
Place

- `NumWebPurchases` : Number of purchases made through the company's website
- `NumCatalogPurchases` : Number of purchases made using a catalogue
- `NumStorePurchases` : Number of purchases made directly in stores
- `NumWebVisitsMonth` : Number of visits to company's website in the last month

Part 1. Exploratory Analysis and Feature Engineering.

Question 1.1. What is the distribution of the birth year of the customers? Show using an appropriate graph. Also report the mean and the standard deviation.

In [5]:
```
### YOUR CODE HERE

plt.hist(df['Year_Birth'],color='red',edgecolor='black')
plt.xlabel('Birth Year')
plt.ylabel('Frequency')
plt.title('Distribution of Customer Birth Years')
plt.show()

mean_birth_year = df['Year_Birth'].mean()
std_birth_year = df['Year_Birth'].std()

print("Mean birth year:", mean_birth_year)
print("Standard deviation of birth year:", std_birth_year)
```



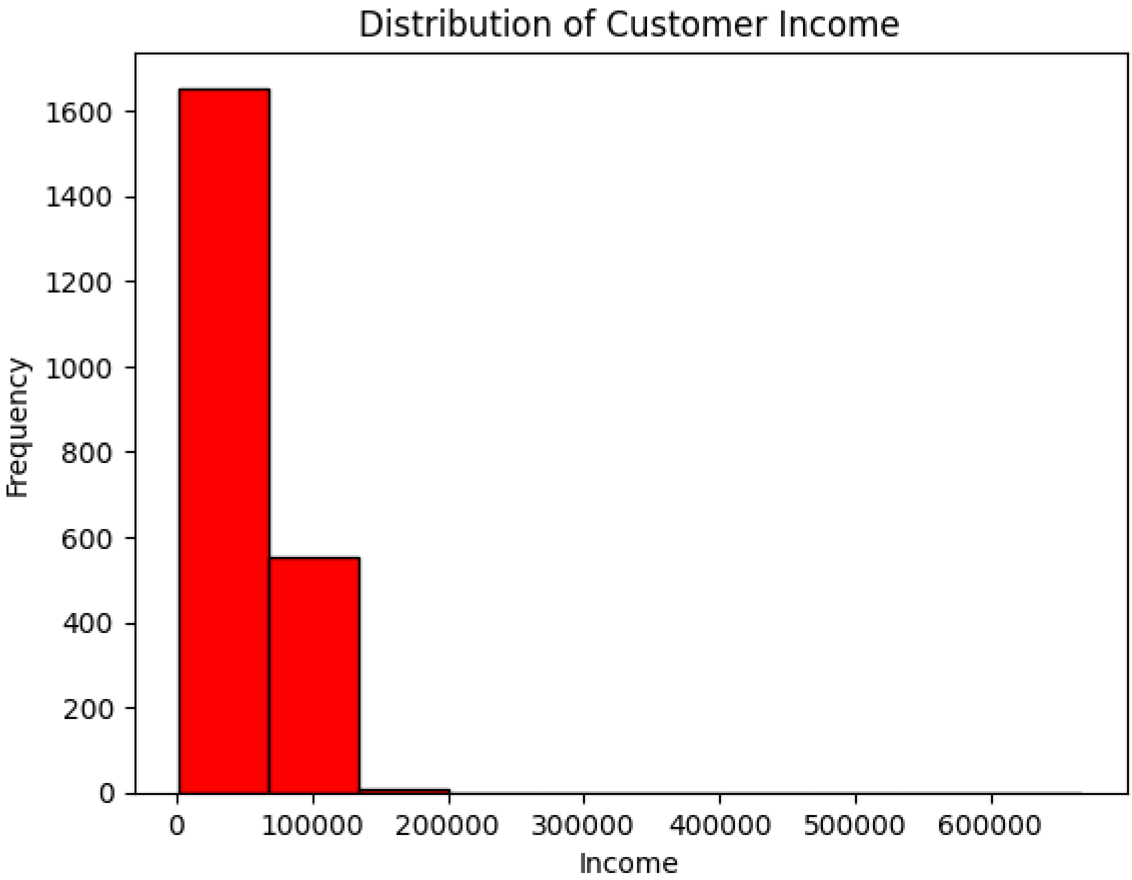

```
Mean birth year: 1968.8058035714287
Standard deviation of birth year: 11.984069456885829
```

(written response): What can you say about the company's customer base?

Most of the customers are born in the 1970-80 range.

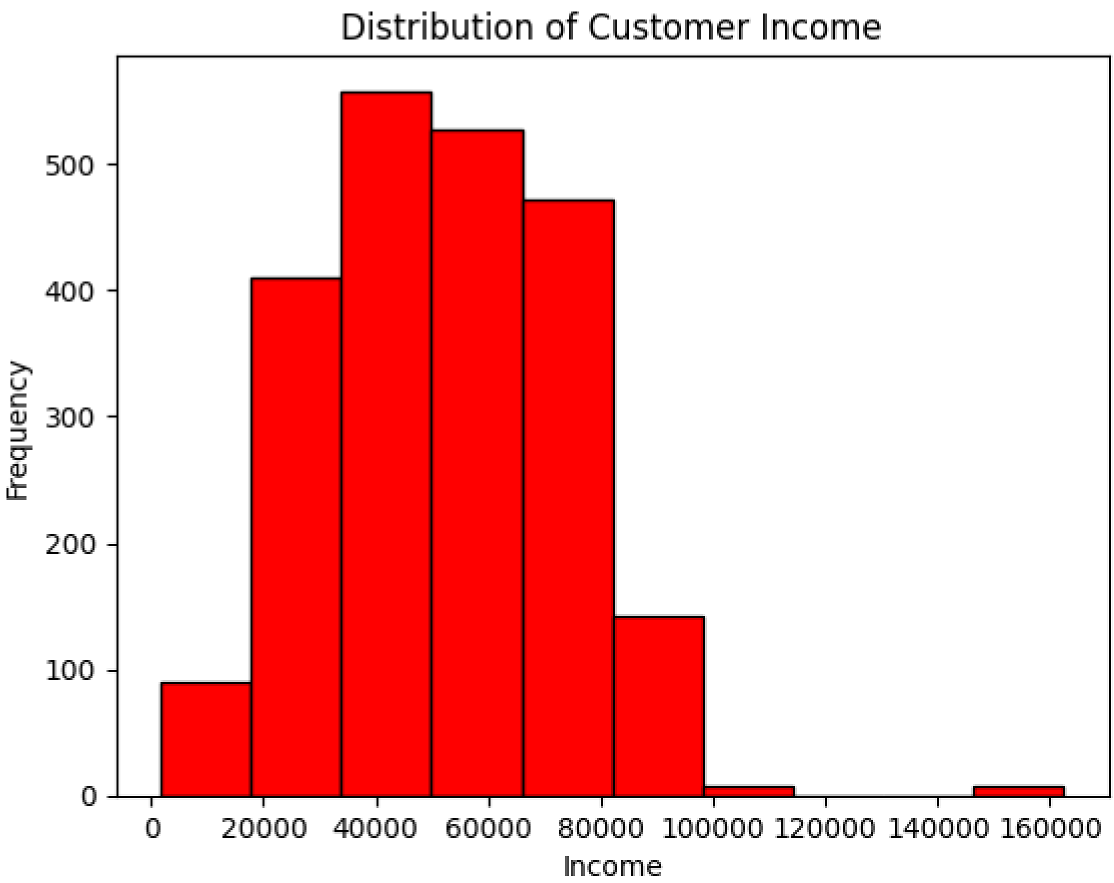
Question 1.2: Now, let us look at the income distribution. Start with the histogram:

In [6]:
```
### YOUR CODE HERE
plt.hist(df['Income'].dropna(),color='red',edgecolor='black')
plt.xlabel('Income')
plt.ylabel('Frequency')
plt.title('Distribution of Customer Income')
plt.show()
```



Eliminate the outlier(s) and try again

In [7]:
```
### YOUR CODE HERE
df = df[df['Income'] < 600000]
plt.hist(df['Income'],color='red',edgecolor='black')
plt.xlabel('Income')
plt.ylabel('Frequency')
plt.title('Distribution of Customer Income')
plt.show()
```



What can you say about the income of the main group of customers of the company?

(written resonse)

The income of the main group is in the range of 35000-45000 $ which is less and not a very high income category.

If you needed to separate the income into a relatively small number of categories, how many would you choose, and what would they be? Give a written response, and write Python code that adds to the data frame `df` a feature called `IncomeBin` which takes the values you specify using the breakdown you put together. Report the distribution of individual in each category. (Hint: you might want to implement the conversion as a Python function)

(written response):

I think there should be 3 categories. High income, medium income and low income. The income high is above 60000, low is below 30000.

In [8]:
```
### YOUR CODE HERE
def categorize_income(income):
    if income < 30000:
        return 'Low'
    elif income < 60000:
        return 'Medium'
    else:
        return 'High'
```

```
df['IncomeBin'] = df['Income'].apply(categorize_income)

print(df['IncomeBin'].value_counts())
```

```
IncomeBin
Medium    1004
High       841
Low        370
Name: count, dtype: int64
```

```
<ipython-input-8-6d2eeb87fe64>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['IncomeBin'] = df['Income'].apply(categorize_income)
```

In [9]:
```
df.head()
```

Out[9]:

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

5 rows × 29 columns

Question 1.3 What percentage of people accepted offers in each of the campaign?

In [10]:
```
### YOUR CODE HERE
accepted_all_campaigns = df[(df['AcceptedCmp1'] == 1) &
                            (df['AcceptedCmp2'] == 1) &
                            (df['AcceptedCmp3'] == 1) &
                            (df['AcceptedCmp4'] == 1) &
                            (df['AcceptedCmp5'] == 1)]

num_accepted_all = accepted_all_campaigns.shape[0]
print(num_accepted_all)

campaigns = ['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'AcceptedCmp4',

campaign_acceptance_rate = df[campaigns].mean() * 100

print(campaign_acceptance_rate)
```

```
0
AcceptedCmp1    6.410835
AcceptedCmp2    1.354402
AcceptedCmp3    7.358916
AcceptedCmp4    7.404063
AcceptedCmp5    7.313770
dtype: float64
```

Can you name the most successful and the least successful campaign?

(written response):

Most successful would be campaign 4 and least successful would be campaign 2.

Question 1.4 Create a new attribute `AcceptedCmp` which is set to `1` if the customer accepted at least one offer and to `0` otherwise. Create a bar chart showing the percentage of the customers who accepted at least one campaign offer.

In [11]:
```
### YOUR CODE HERE

df['AcceptedCmp'] = df[['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'Acc
df['AcceptedCmp'] = df['AcceptedCmp'].apply(lambda x: 1 if x > 0 else 0)
```

```
<ipython-input-11-b12bcc2f4b59>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['AcceptedCmp'] = df[['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3',
'AcceptedCmp4', 'AcceptedCmp5']].sum(axis=1)
<ipython-input-11-b12bcc2f4b59>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['AcceptedCmp'] = df['AcceptedCmp'].apply(lambda x: 1 if x > 0 else 0)
```

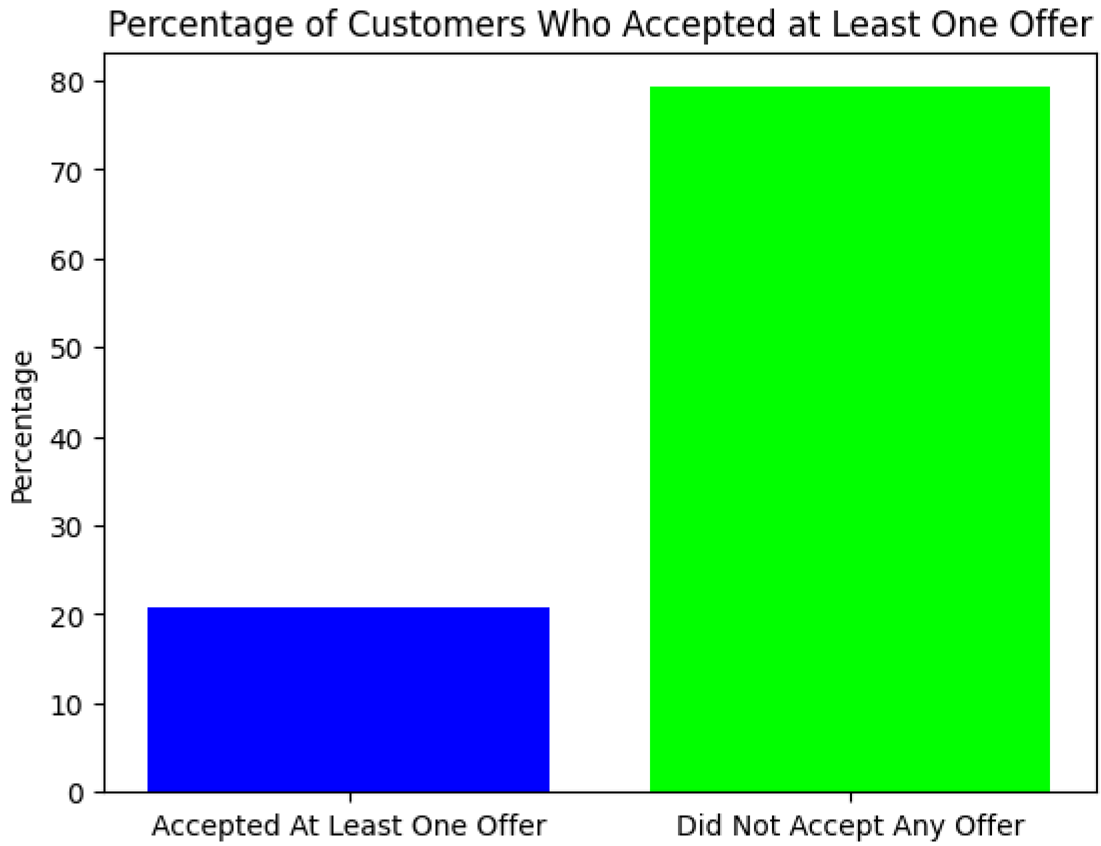
In [12]:
```
df.head()
```

Out[12]:

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

5 rows × 30 columns

In [13]:
```
percentage_accepted = df['AcceptedCmp'].mean() * 100

plt.figure()
plt.bar(['Accepted At Least One Offer', 'Did Not Accept Any Offer'],
        [percentage_accepted, 100 - percentage_accepted],
        color=['blue', 'lime'])
plt.title('Percentage of Customers Who Accepted at Least One Offer')
plt.ylabel('Percentage')
plt.show()

print("Percentage of customers who accepted at least one offer: ",percentage
```




```
Percentage of customers who accepted at least one offer:  20.72234762979684
```

Question 1.5 Now, show the crosstabs between people who accepted at least one campaign offer and people who filed at least one complaint with the company. Make sure, the cross-tab has margins.

In [14]:
```
### YOUR CODE HERE

crosstab = pd.crosstab(df['AcceptedCmp'], df['Complain'], margins=True)
crosstab
```

Out[14]:

Complain	0	1	All
AcceptedCmp			
0	1737	19	1756
1	457	2	459
All	2194	21	2215

What can you tell about the relationship between people who complained and people to made purchases as part of promotional campaigns?

(written response) There are only 21 people who complained, of them, only 2 made purchases using a promotion.

Question 1.6 Create a new feature, `Purchases` that combines together the number of purchases on the web, through the catalog and in store. Study its distribution

In [15]:
```
### YOUR CODE HERE\
df['Purchases'] = df['NumWebPurchases'] + df['NumCatalogPurchases'] + df['Nu
df['Purchases'].describe()
```

```
<ipython-input-15-089c5427fb45>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Purchases'] = df['NumWebPurchases'] + df['NumCatalogPurchases'] + df
['NumStorePurchases']
```

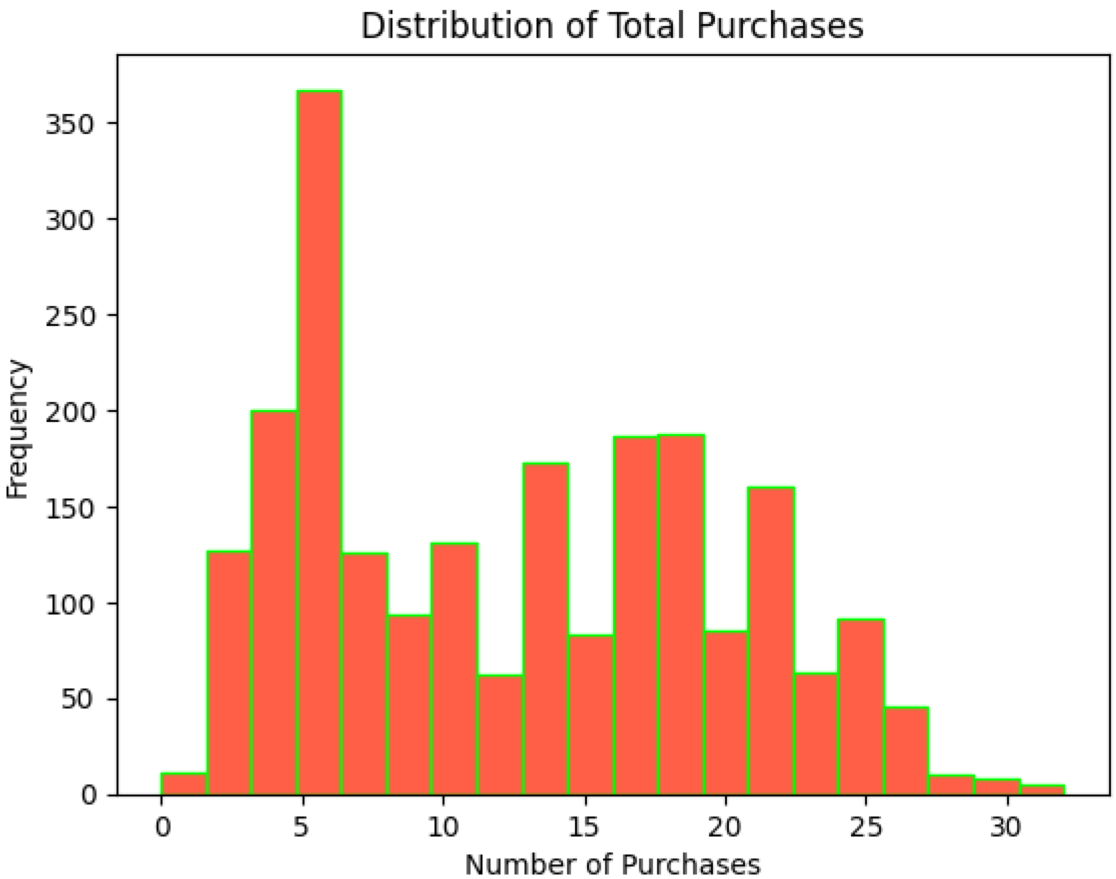
Out[15]:

	Purchases
count	2215.000000
mean	12.559819
std	7.205270
min	0.000000
25%	6.000000
50%	12.000000
75%	18.000000
max	32.000000

dtype: float64

In [16]:

```
### YOUR CODE HERE
plt.figure()
plt.hist(df['Purchases'], bins=20, color='tomato', edgecolor='lime')
plt.title('Distribution of Total Purchases')
plt.xlabel('Number of Purchases')
plt.ylabel('Frequency')
plt.show()
```



Question 1.7 Create a visualization that shows all of the following information in a single plot:

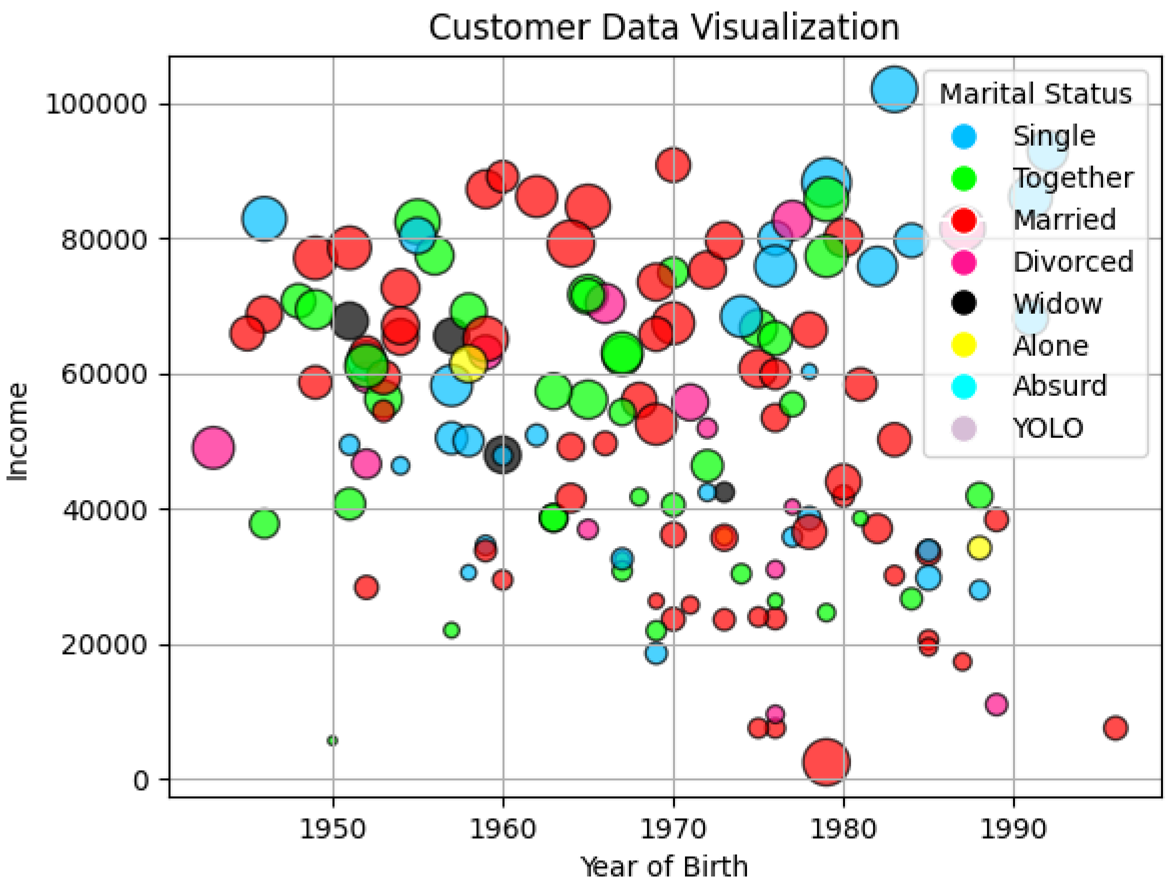
- customer year of birth
- customer income
- customer marital status
- number of purchases a customer made with the company.

Make sure to do all the prep work (color bindings, etc...) first.

Feel free to merge some outliers in marital status and level of education.

Apply this visualization to the first 150 data points from the dataset.

In [17]:
```
### YOUR CODE HERE
df_filtered = df.head(150).copy()

colors = {
    'Single': 'deepskyblue',
    'Together': 'lime',
    'Married': 'red',
    'Divorced': 'deeppink',
    'Widow': 'black',
    'Alone': 'yellow',
    'Absurd': 'aqua',
    'YOLO': 'thistle'
}
df_filtered['Color'] = df_filtered['Marital_Status'].map(colors)

plt.figure()
scatter = plt.scatter(
    x=df_filtered['Year_Birth'],
    y=df_filtered['Income'],
    c=df_filtered['Color'],
    s=df_filtered['Purchases'] * 10,
    alpha=0.7,
    edgecolor='k'
)
handles = [plt.Line2D([0], [0], marker='o', color='w', label=status,
                      markersize=10, markerfacecolor=colors[status])
           for status in colors]
plt.legend(handles=handles, title='Marital Status')

plt.title('Customer Data Visualization')
plt.xlabel('Year of Birth')
plt.ylabel('Income')
plt.grid(True)

plt.show()
```



Part 2. Regression

Question 2.1 Determine if there is a sufficiently strong relationship between the birth year (i.e., age) of a customer, and their current income. Start with a simple visualization, proceed with building as linear regression model. Draw the obtained regression line. Answer written response questions below.

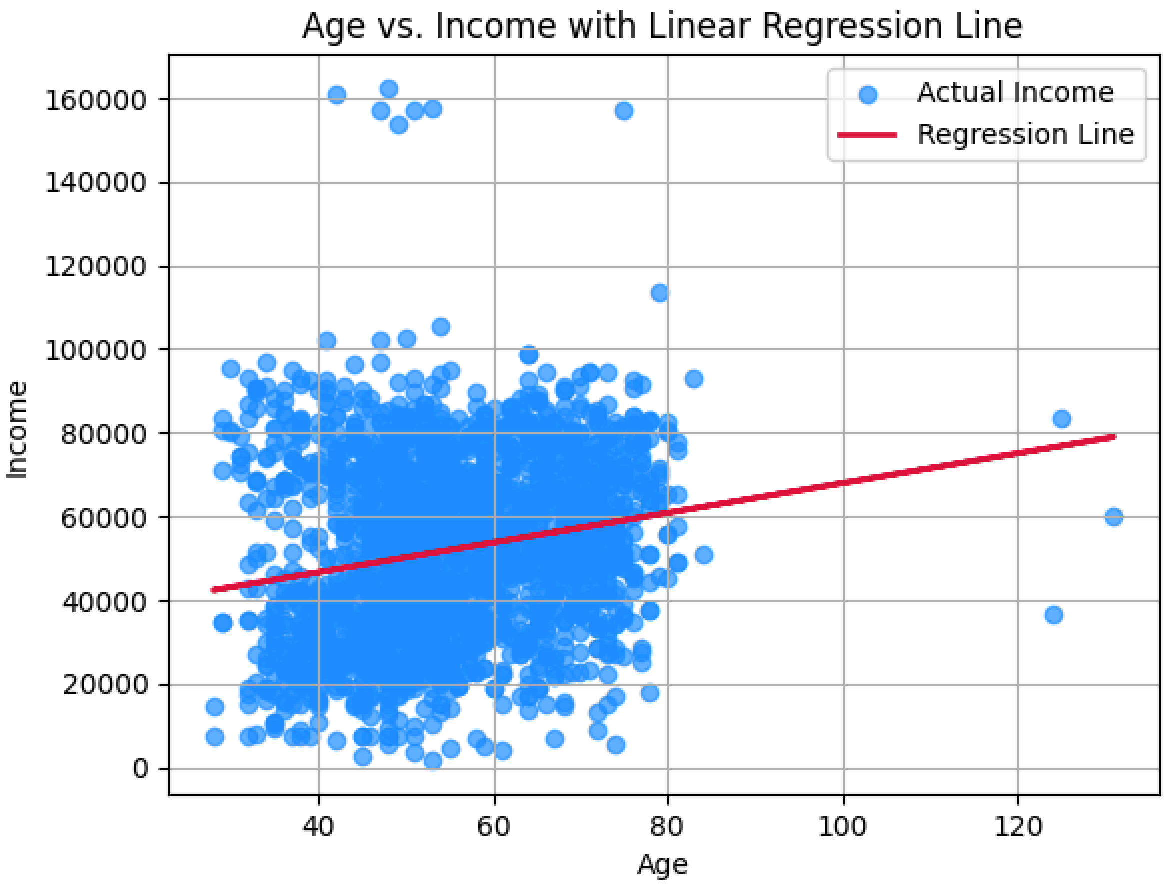
Feel free to remove the one outlier from the visualization (and from the regression model as well)

Note: There are some rows of data missing income information. For the purpose of building the regression model below, remove those from consideration.

In [18]:
```
### YOUR CODE HERE

df_filtered = df.dropna(subset=['Income'])
df_filtered['Age'] = 2024 - df_filtered['Year_Birth']

df_filtered = df_filtered[df_filtered['Income'] < 600000]
```

In [19]:
```
### YOUR CODE HERE

X = df_filtered[['Age']]
y = df_filtered['Income']

model = LinearRegression()
model.fit(X, y)

df_filtered['Predicted_Income'] = model.predict(X)
```

In [20]:
```
### YOUR CODE HERE

plt.figure()
plt.scatter(df_filtered['Age'], df_filtered['Income'], color='dodgerblue', 
plt.plot(df_filtered['Age'], df_filtered['Predicted_Income'], color='crimso

plt.title('Age vs. Income with Linear Regression Line')
plt.xlabel('Age')
plt.ylabel('Income')
plt.legend()
plt.grid(True)

plt.show()

print(f"Coefficient: {model.coef_[0]}")
print(f"Intercept: {model.intercept_}")
```




```
Coefficient: 355.68455570686774
Intercept: 32342.015382119123
```

(written response) Examine the model coefficients and the model visualization and explain what you are seeing (i.e., the behavior of the predictor).

We can clearly see that income increases as age increases. The linear line also fits in a simillar way. This is becuase we know that people get increment every year they work more.

Question 2.2. Let us now use the "amount of money spent on X" variables (the ones that start with `Mnt`) to predict income. For this exercise, use the first 2000 data points in this dataset as the training set, and reserve the remaining points as the validation set. Report the training and validation MSE, RMSE, and MAE. How good is your predictor?

(you can .dropna() the training and test sets after you construct them)

In [21]:
```
### YOUR CODE HERE
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

predictor_cols = [col for col in df.columns if col.startswith('Mnt')]
X = df[predictor_cols]
y = df['Income']

X_train = X.head(2000)
y_train = y.head(2000)
X_valid = X.iloc[2000:]
y_valid = y.iloc[2000:]

X_train = X_train.dropna()
y_train = y_train.loc[X_train.index]
X_valid = X_valid.dropna()
y_valid = y_valid.loc[X_valid.index]
```

In [22]:
```
### YOUR CODE HERE
model = LinearRegression()
model.fit(X_train, y_train)

y_train_pred = model.predict(X_train)
y_valid_pred = model.predict(X_valid)

train_mse = mean_squared_error(y_train, y_train_pred)
train_rmse = np.sqrt(train_mse)
train_mae = mean_absolute_error(y_train, y_train_pred)

valid_mse = mean_squared_error(y_valid, y_valid_pred)
valid_rmse = np.sqrt(valid_mse)
valid_mae = mean_absolute_error(y_valid, y_valid_pred)
```

In [23]:
```
### YOUR CODE HERE
print("Training MSE:", train_mse)
print("Training RMSE:", train_rmse)
print("Training MAE:", train_mae)
print("Validation MSE:", valid_mse)
print("Validation RMSE:", valid_rmse)
print("Validation MAE:", valid_mae)
```

```
Training MSE: 166674972.1284625
Training RMSE: 12910.266152502918
Training MAE: 9448.2694697446
Validation MSE: 197263228.97468573
Validation RMSE: 14045.042861261967
Validation MAE: 9674.104979434747
```

How good is your predictor?

(written response):

The training error is large and the test error is even larger which is obvious. The mean absolute error gives a more stable value but is still quite large. So our predictor is not that good.

Question 2.3 Let us add the birth year, marital status and educational background into the mix. Create the appropriate features, and find the training and test accuracy based on the same train-validation split as in **Question 2.2.**

In [24]:
```
#### YOUR CODE HERE
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

predictor_cols = [col for col in df.columns if col.startswith('Mnt')] + ['Ye
X = df[predictor_cols]
y = df['Income']

preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), ['Marital_Status',
        ('num', 'passthrough', ['Year_Birth'] + [col for col in predictor_co
    ])

X_train = X.head(2000)
y_train = y.head(2000)
X_valid = X.iloc[2000:]
y_valid = y.iloc[2000:]

X_train = X_train.dropna()
y_train = y_train.loc[X_train.index]
X_valid = X_valid.dropna()
y_valid = y_valid.loc[X_valid.index]
```

In [25]:
```
#### YOUR CODE HERE
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

model.fit(X_train, y_train)
y_train_pred = model.predict(X_train)
y_valid_pred = model.predict(X_valid)

train_mse = mean_squared_error(y_train, y_train_pred)
train_rmse = np.sqrt(train_mse)
train_mae = mean_absolute_error(y_train, y_train_pred)

valid_mse = mean_squared_error(y_valid, y_valid_pred)
valid_rmse = np.sqrt(valid_mse)
valid_mae = mean_absolute_error(y_valid, y_valid_pred)
```

In [26]:
```
#### YOUR CODE HERE
print("Training MSE:", train_mse)
print("Training RMSE:", train_rmse)
print("Training MAE:", train_mae)
print("Validation MSE:", valid_mse)
print("Validation RMSE:", valid_rmse)
print("Validation MAE:", valid_mae)
```

```
Training MSE: 153930965.5952185
Training RMSE: 12406.89185877021
Training MAE: 8940.413924463875
Validation MSE: 171225112.62505496
Validation RMSE: 13085.301396034214
Validation MAE: 8705.44688121181
```

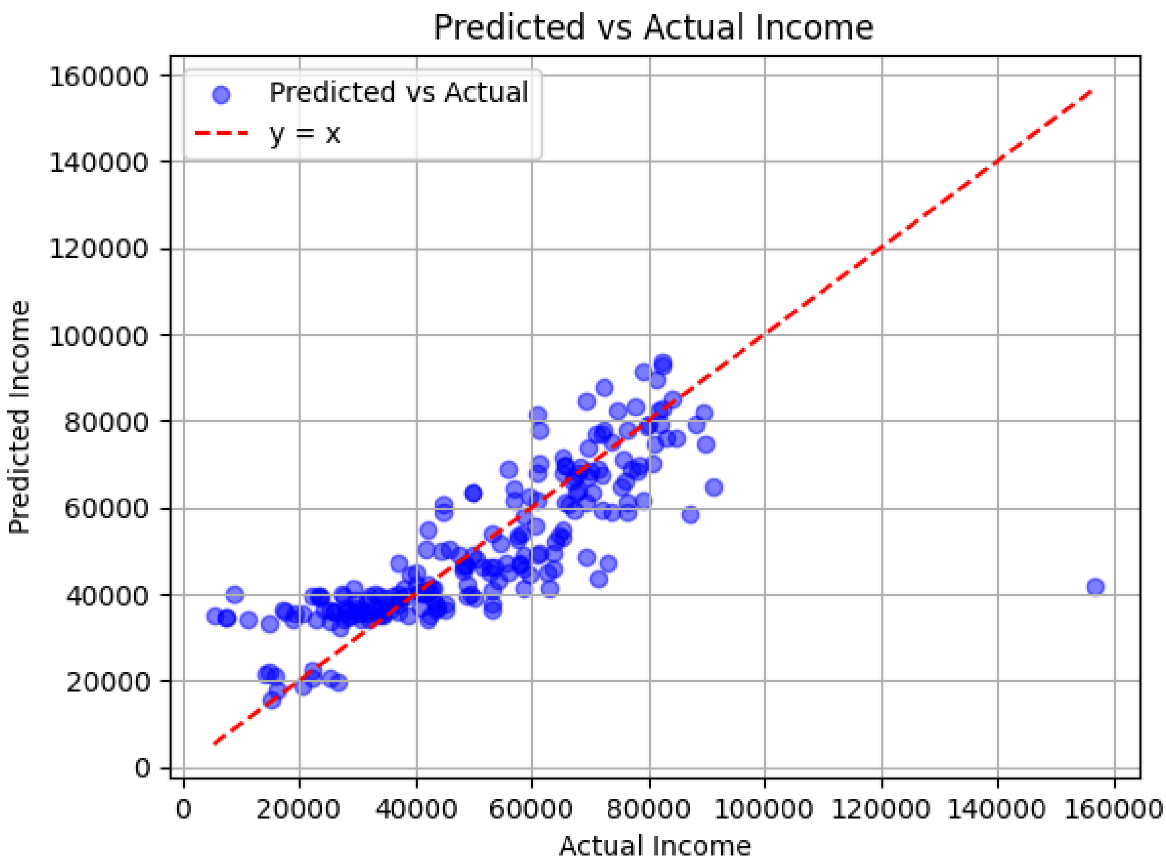
Did the training accuracy increase? If yes, did it increase significantly? Did the validation accuracy improve? If yes, did it improve significantly?

Both parameters increased and for example the mean absolute error did increase by 1000. So it is kind of significant.

Question 2.4 Plot predicted vs. actual income values returned by your **Question 2.3** model on the test set (make sure to draw the $y = x$ line for the sake of comparison as well). Based on what you are observing, can you say if there are specific income brackets where your model is having a harder time predicting?

(you can remove outliers from your graph for the sake of clarity)

In [27]:
```
### YOUR CODE HERE
y_valid_pred = model.predict(X_valid)

mask = (y_valid < 600000) & (y_valid_pred < 600000)
y_valid = y_valid[mask]
y_valid_pred = y_valid_pred[mask]

plt.figure()
plt.scatter(y_valid, y_valid_pred, alpha=0.5, color='blue', label='Predictec
plt.plot([y_valid.min(), y_valid.max()], [y_valid.min(), y_valid.max()], 'r-
plt.xlabel('Actual Income')
plt.ylabel('Predicted Income')
plt.title('Predicted vs Actual Income')
plt.legend()
plt.grid(True)
plt.show()
```



(written response). There are several points where we can observe that predicted value is higher and also many are where predicted value is lower. 20k to 40k and 60k to 80k had more wrong predictions than others.

Part 3. Classification

Question 3.1 Predict if the customer accepted the offer in the last campaign (variable named `Response`) based on whether they accepted the offers in any of the previous campaigns, and the total number of discounted purchases they made. Use a **Logistic Regression** classifier with default parameters. Do an 80-20 train-validation split (random, but you can fix the seed), and report both the training and validation accuracy, together with the training and validation confusion matrices.

In [28]:

```
### YOUR CODE HERE
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

features = ['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'AcceptedCmp4',
X = df[features]
y = df['Response']

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2,

model = LogisticRegression()
model.fit(X_train, y_train)
y_train_pred = model.predict(X_train)
y_valid_pred = model.predict(X_valid)

train_accuracy = accuracy_score(y_train, y_train_pred)
valid_accuracy = accuracy_score(y_valid, y_valid_pred)

train_conf_matrix = confusion_matrix(y_train, y_train_pred)
valid_conf_matrix = confusion_matrix(y_valid, y_valid_pred)

print("Training Accuracy:", train_accuracy)
print("Validation Accuracy:",valid_accuracy)
print("Training Confusion Matrix:")
print(train_conf_matrix)
print("Validation Confusion Matrix:")
print(valid_conf_matrix)
```

```
Training Accuracy: 0.8656884875846501
Validation Accuracy: 0.8781038374717833
Training Confusion Matrix:
[[1477   20]
 [ 218   57]]
Validation Confusion Matrix:
[[378   7]
 [ 47  11]]
```

In your opinion, is this an accurate predictor? Why or why not?

(written response) Both accuracy are very high and we can believe that the predictor is right. We can always try to bring some improvements here or try another model to see if it improves accuracy or not.

Is accuracy a good metric to compute here? Why or why not? If not - what other metric or metrics would you like to compute?

(written response) It is a good metric except for the case when there is a class imbalance. We can try precision, recall and F1 score.

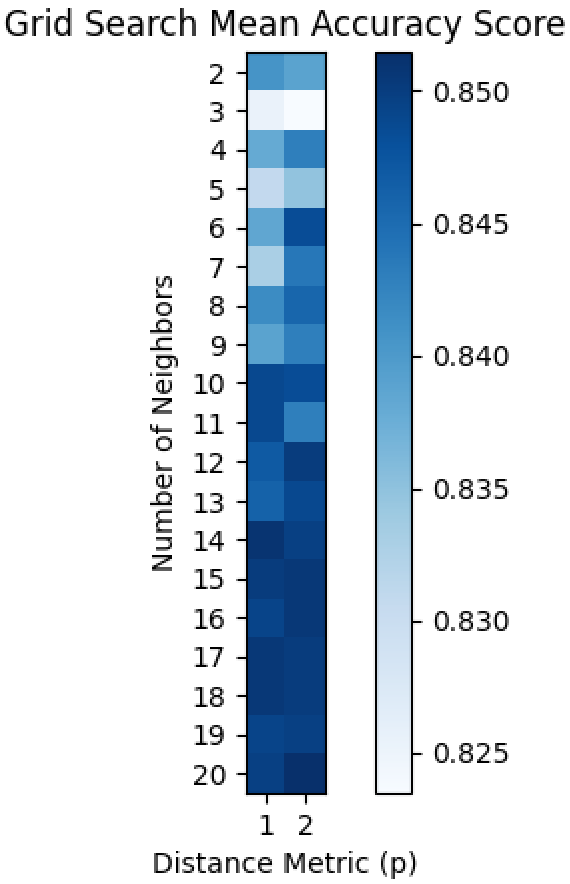
Question 3.2. Now predict the same variable (`Response`) but base it on the `Mnt` variables (amount of money spent on different categories of products) plus the *Place* variables (number of purchases made via web, catalog, or in store, and number of web site visits). Use `KNeighbors` classifier and do a grid search for the number of neighbors and the distance metric, using 5-fold cross-validation. Pick the right metric for cross validation, report the computed values of the metric for each value of C tested (both in tabular form, and as a graph).

Additionally, for the best model, extract predictions, and construct the confusion matrix (since it was cross-validated, there is only one confusion matrix needed here).

Note: use `cross_val_predict` to get the predictions at the end.

In [29]:
```
### YOUR CODE HERE
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, cross_val_predict
from sklearn.metrics import confusion_matrix, make_scorer, accuracy_score

mnt_vars = ['MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts',
place_vars = ['NumWebPurchases', 'NumCatalogPurchases', 'NumStorePurchases',
X = df[mnt_vars + place_vars]
y = df['Response']

knn = KNeighborsClassifier()
param_grid = {
    'n_neighbors': list(range(2, 21)),
    'p': [1, 2]
}
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5, scor
grid_search.fit(X, y)

best_params = grid_search.best_params_
results = pd.DataFrame(grid_search.cv_results_)

scores_matrix = results.pivot(index='param_n_neighbors', columns='param_p',
```

In [30]:
```
### YOUR CODE HERE
plt.figure()
plt.imshow(scores_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.colorbar()
plt.xticks(range(scores_matrix.shape[1]), scores_matrix.columns)
plt.yticks(range(scores_matrix.shape[0]), scores_matrix.index)
plt.xlabel('Distance Metric (p)')
plt.ylabel('Number of Neighbors')
plt.title('Grid Search Mean Accuracy Score')
plt.show()
```



In [31]:
```
### YOUR CODE HERE
best_knn = grid_search.best_estimator_

y_pred = cross_val_predict(best_knn, X, y, cv=5)

conf_matrix = confusion_matrix(y, y_pred)
print(f"Best Parameters: {best_params}")
print("Confusion Matrix:")
print(conf_matrix)
```

```
Best Parameters: {'n_neighbors': 20, 'p': 2}
Confusion Matrix:
[[1864   18]
 [ 311   22]]
```

Looking at your work, and considering that you are comparing apples to oranges a bit, do you believe that it is easier to predict the `Response` variable from this set of features, or the set of features from **Question 3.1.**? Why or why not?

(writttten response):

The KNN model got a good performance, but the confusion matrix shows a higher number of false negatives compared to the logistic regression model. This mens that it is less accurate. The logistic model used features which I believe are directly related to the customer’s likelihood of responding to a promotional campaign. so they are more useful.

Part 4. Clustering

Question 4.1 Use the `Mnt` (amount of money spent on various types of products) variables to cluster the customers using `K-means` into four (4) clusters.

Report your results by coloring a Birth Year vs. Income scatterplot. Cluster once without standardizing the observation, and once after standardizing your inputs.

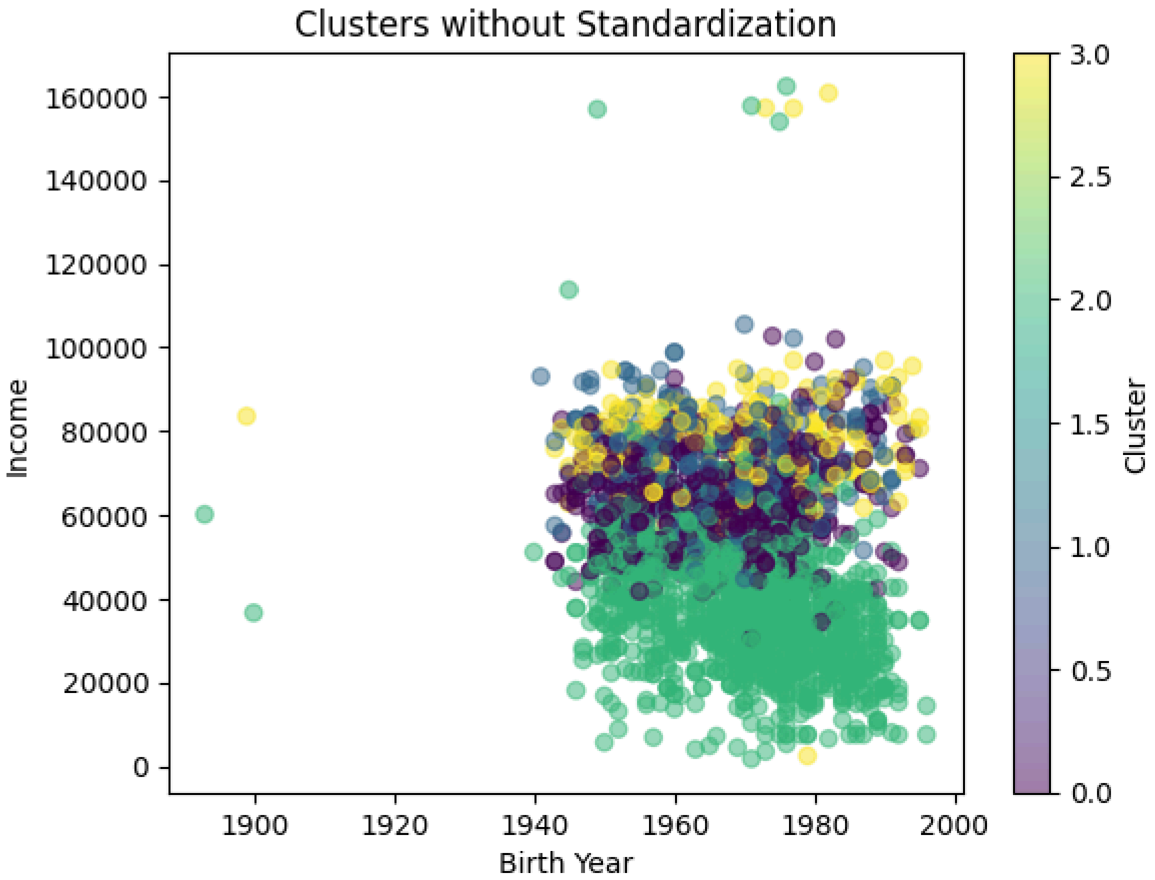
(as usual, feel free to eliminate income outliers from visualizations)

In [32]:
```
### YOUR CODE HERE
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

mnt_columns = ['MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts
df_mnt = df[mnt_columns].dropna()

df['Income'] = pd.to_numeric(df['Income'], errors='coerce')
df = df.dropna(subset=['Income'])
df = df[df['Income'] < 600000]
```

```
<ipython-input-32-4790f2e624e7>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Income'] = pd.to_numeric(df['Income'], errors='coerce')
```

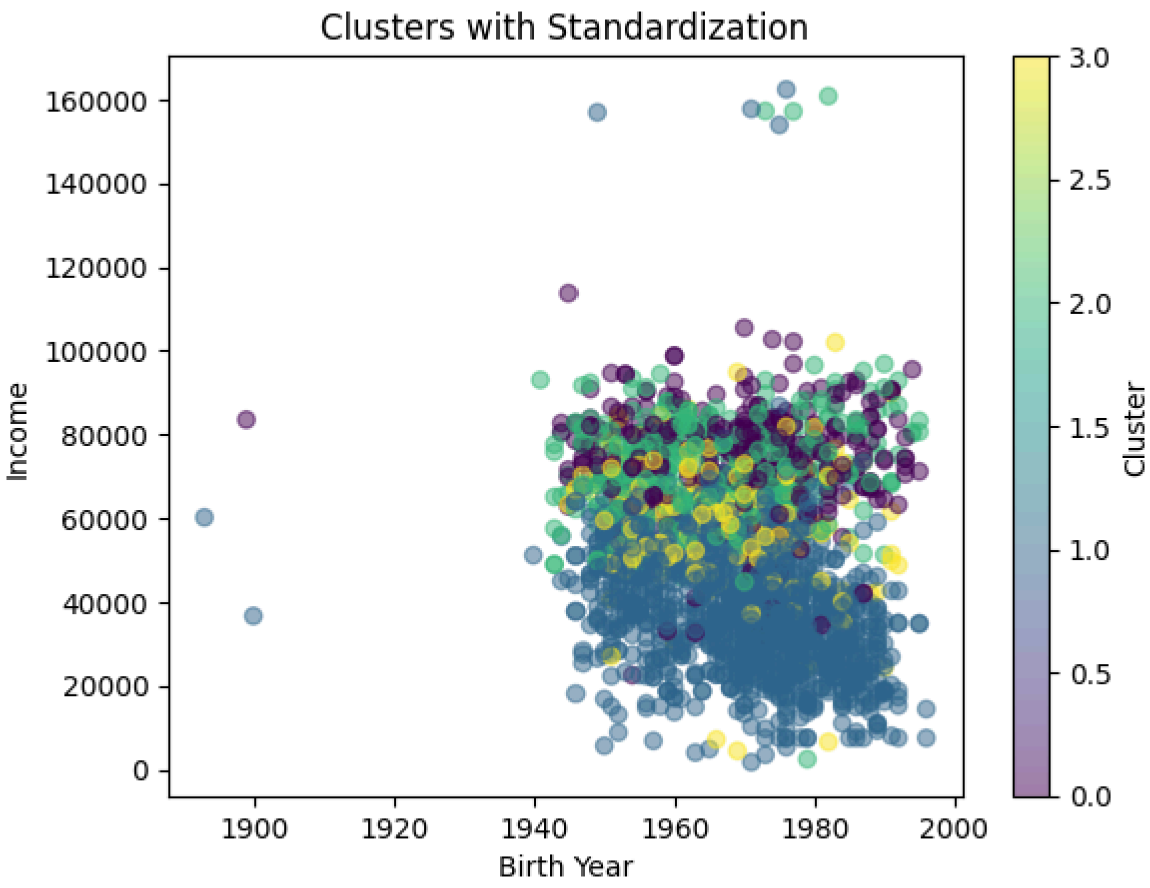
In [33]:
```
### YOUR CODE HERE
kmeans = KMeans(n_clusters=4, random_state=1)
clusters = kmeans.fit_predict(df_mnt)

df['Cluster'] = clusters

plt.figure()
scatter = plt.scatter(df['Year_Birth'], df['Income'], c=df['Cluster'], cmap=
plt.colorbar(scatter, label='Cluster')
plt.xlabel('Birth Year')
plt.ylabel('Income')
plt.title('Clusters without Standardization')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: Fu
tureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



In [34]:
```
### YOUR CODE HERE
scaler = StandardScaler()
df_mnt_scaled = scaler.fit_transform(df_mnt)

kmeans_scaled = KMeans(n_clusters=4, random_state=1)
clusters_scaled = kmeans_scaled.fit_predict(df_mnt_scaled)

df['Cluster_Scaled'] = clusters_scaled

plt.figure()
scatter = plt.scatter(df['Year_Birth'], df['Income'], c=df['Cluster_Scaled']
plt.colorbar(scatter, label='Cluster')
plt.xlabel('Birth Year')
plt.ylabel('Income')
plt.title('Clusters with Standardization')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: Fu
tureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



Question 4.2. Generally speaking, your **Question 4.1.**, clusters could take one of two different forms:

- **faceted clusters**: these highlight differences in the structure of customer spending (e.g., one cluster would emphasize spending on meat and fish products, while another - on wine and gold).
- **layered clusters**: such clusters would highlight the differences in the overall spending patterns. E.g., one cluster will group people who spend very little in general, another - people who spend a lot, and so on.

Looking at the actual clusters your obtained, do you believe they are more `faceted` or more `layered`, or (this is also possible) represent a mix of both possibilities?

(written response) One cluster is low-spenders. Two clusters are high-spenders with different spending distribution (one values meat over wine, the other - wine over meat). The remaining cluster are people who spend much more on wine than on other categories.

Submission Instructions

- Restart this notebook and run the cells from beginning to end.
  - Go to Runtime > Restart and Run All.

In [ ]:
```
# @markdown Run this cell to download this notebook as a webpage, `_NOTEBOO

import google, json, nbformat
```

```
# Get the current notebook and write it to _NOTEBOOK.ipynb
raw_notebook = google.colab._message.blocking_request("get_ipynb",
                                                      timeout_sec=30)["ipynb
with open("_NOTEBOOK.ipynb", "w", encoding="utf-8") as ipynb_file:
  ipynb_file.write(json.dumps(raw_notebook))

# Use nbconvert to convert .ipynb to .html.
!jupyter nbconvert --to html --log-level WARN _NOTEBOOK.ipynb

# Download the .html file.
google.colab.files.download("_NOTEBOOK.html")
```

- Open `_NOTEBOOK.html` in your browser, and save it as a PDF.
  - Go to File > Print > Save as PDF.
- Double check that all of your code and output is visible in the saved PDF.
- Upload the PDF to Gradescope.
  - Please be sure to select the correct pages corresponding to each question.