Assignment 1.1

September 9, 2024

Assignment 1.1 Author "Gabriel Mancillas Gallardo" Date: 9.9.2024

[9]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Set seed for reproducibility
np.random.seed(42)
```

[10]:
```
# Load the dataset
cbc_data = pd.read_csv('/Users/gabrielmancillas/Desktop/ADS 505-01/Mod 01/
 ↪Assignment/CharlesBookClub.csv')

# Summary of the dataset (similar to R's summary function)
summary = cbc_data.describe(include='all')
print(summary)
```

```
              Seq#           ID#       Gender            M            R  \
count  4000.000000   4000.000000  4000.000000  4000.000000  4000.000000
mean   2000.500000  16594.623000     0.704500   208.091500    13.390500
std    1154.844867   9484.433792     0.456324   100.948548     8.103822
min       1.000000     25.000000     0.000000    15.000000     2.000000
25%    1000.750000   8253.250000     0.000000   129.000000     8.000000
50%    2000.500000  16581.000000     1.000000   208.000000    12.000000
75%    3000.250000  24838.250000     1.000000   283.000000    16.000000
max    4000.000000  32977.000000     1.000000   479.000000    36.000000

                 F  FirstPurch     ChildBks    YouthBks      CookBks  …  \
count  4000.000000  4000.00000  4000.000000  4000.00000  4000.000000  …
mean      3.833250    26.50725     0.639750     0.30475     0.731250  …
std       3.458386    18.35138     0.994343     0.61194     1.089413  …
min       1.000000     2.00000     0.000000     0.00000     0.000000  …
25%       1.000000    12.00000     0.000000     0.00000     0.000000  …
```

```
50%        2.000000     20.00000     0.000000     0.00000     0.000000  …
75%        6.000000     36.00000     1.000000     0.00000     1.000000  …
max       12.000000     99.00000     7.000000     5.00000     7.000000  …

          ItalCook    ItalAtlas      ItalArt     Florence  Related Purchase  \
count  4000.000000  4000.000000  4000.000000  4000.000000       4000.000000
mean      0.125250     0.037500     0.045750     0.084500          0.885000
std       0.385486     0.214721     0.220611     0.278171          1.226234
min       0.000000     0.000000     0.000000     0.000000          0.000000
25%       0.000000     0.000000     0.000000     0.000000          0.000000
50%       0.000000     0.000000     0.000000     0.000000          0.000000
75%       0.000000     0.000000     0.000000     0.000000          1.000000
max       3.000000     2.000000     2.000000     1.000000          8.000000

             Mcode        Rcode        Fcode  Yes_Florence  No_Florence
count  4000.000000  4000.000000  4000.000000   4000.000000  4000.000000
mean      4.281250     3.170000     2.085750      0.084500     0.915500
std       0.915619     0.928071     0.831907      0.278171     0.278171
min       1.000000     1.000000     1.000000      0.000000     0.000000
25%       4.000000     3.000000     1.000000      0.000000     1.000000
50%       5.000000     3.000000     2.000000      0.000000     1.000000
75%       5.000000     4.000000     3.000000      0.000000     1.000000
max       5.000000     4.000000     3.000000      1.000000     1.000000

[8 rows x 24 columns]
```

Question 1.1: What is the response rate for the training data customers taken as a whole? What is the response rate for each of the combinations of RFM categories? Which combinations have response rates in the training data that are above the overall response in the training data?

[11]:
```python
np.random.seed(1)
```

[12]:
```python
# Split the dataset into training and validation sets (60% training, 40%
validation)
train_data, validation_data = train_test_split(cbc_data, test_size=0.4,
stratify=cbc_data['Florence'])

# Calculate the overall response rate in the training data
total_customers = len(train_data)
responded_customers = train_data['Florence'].sum()
overall_response_rate = responded_customers / total_customers

# Convert overall response rate to percentage
overall_response_rate_percentage = overall_response_rate * 100
print('The overall response rate is:', overall_response_rate_percentage, '%')
```

```
The overall response rate is: 8.458333333333332 %
```

[13]:
```python
# Calculate response rates for each RFM category
rfm_response_rate = train_data.groupby(['R', 'F', 'M']).
↪agg(response_rate=('Florence', 'mean')).reset_index()

# Find combinations with above-average response rates
above_average_combinations =␣
↪rfm_response_rate[rfm_response_rate['response_rate'] > overall_response_rate]
print(above_average_combinations)
```

```
       R   F    M  response_rate
1      2   1   31            1.0
16     2   1  109            1.0
18     2   1  131            1.0
32     2   1  230            1.0
42     2   1  299            1.0
…     ..  ..  …              …
2169  34   2  170            1.0
2175  34   2  261            1.0
2195  36   1   58            1.0
2209  36   2  316            1.0
2225  36  12  393            1.0

[202 rows x 4 columns]
```

[14]:
```python
# Calculate the overall response rate in the training data
total_customers = len(train_data)
responded_customers = train_data['Florence'].sum()
overall_response_rate = responded_customers / total_customers
print(f"Overall Response Rate for Training Data: {overall_response_rate:.2%}")

# Group by RFM categories and calculate response rates
def calculate_response_rate(group):
    return group['Florence'].mean()

rfm_response_rate = train_data.groupby(['R', 'F', 'M']).
↪apply(calculate_response_rate).reset_index()
rfm_response_rate.columns = ['R', 'F', 'M', 'response_rate']

# Identify RFM combinations with above-average response rates
above_avg_rfm = rfm_response_rate[rfm_response_rate['response_rate'] >␣
↪overall_response_rate]
print("Combinations with Above-Average Response Rates:")
print(above_avg_rfm)
```

```
Overall Response Rate for Training Data: 8.46%
Combinations with Above-Average Response Rates:
       R   F    M  response_rate
1      2   1   31            1.0
```

```
16      2   1  109             1.0
18      2   1  131             1.0
32      2   1  230             1.0
42      2   1  299             1.0
…      ..  ..  …           …
2169  34   2  170             1.0
2175  34   2  261             1.0
2195  36   1   58             1.0
2209  36   2  316             1.0
2225  36  12  393             1.0

[202 rows x 4 columns]
```

```
/var/folders/jw/4t4swxld5c5f_5xhv0_bzbr00000gn/T/ipykernel_63808/4140313905.py:1
1: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns.
This behavior is deprecated, and in a future version of pandas the grouping
columns will be excluded from the operation. Either pass `include_groups=False`
to exclude the groupings or explicitly select the grouping columns after groupby
to silence this warning.
  rfm_response_rate = train_data.groupby(['R', 'F',
'M']).apply(calculate_response_rate).reset_index()
```

Question 1.2: Compute the response rate for validation using "above average" RFM combinations

[15]:
```
# Validation: Calculate the response rate for validation data using the
 ↪selected RFM combinations
valid_above_avg = validation_data.merge(above_avg_rfm, on=['R', 'F', 'M'],
 ↪how='inner')
validation_response_rate = valid_above_avg['Florence'].mean()
print(f"Validation Response Rate for Above-Average Combinations:
 ↪{validation_response_rate:.2%}")
```

```
Validation Response Rate for Above-Average Combinations: 14.29%
```

Results Question 1: Overall Response Rate for Training Data: 8.458%

Reponse Rates for RFM Combinations: Identied combinations where the response rate is higher than 8.46% (above average combinations)

Validation Response Rate: 17.65%

Question 2: k-Nearest Neighbors (k-NN) Classification

[16]:
```
# Define the normalization function
def normalize(x):
    return (x - x.min()) / (x.max() - x.min())

# Normalize relevant variables in training data
train_data_norm = train_data.copy()
```

```python
train_data_norm[['R', 'F', 'M', 'FirstPurch', 'Related Purchase']] = train_data_norm[['R', 'F', 'M', 'FirstPurch', 'Related Purchase']].apply(normalize)

# Normalize relevant variables in validation data
validation_data_norm = validation_data.copy()
validation_data_norm[['R', 'F', 'M', 'FirstPurch', 'Related Purchase']] = validation_data_norm[['R', 'F', 'M', 'FirstPurch', 'Related Purchase']].apply(normalize)
```

[17]:
```python
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Prepare input (X) and output (y) for k-NN
train_x = train_data_norm[['R', 'F', 'M', 'FirstPurch']]
train_y = train_data_norm['Florence']
validation_x = validation_data_norm[['R', 'F', 'M', 'FirstPurch']]
validation_y = validation_data_norm['Florence']

# Perform k-NN for k = 1 to 11
k_values = range(1, 12)

# Loop through each k value, fit the model, and calculate accuracy
accuracy_results = pd.DataFrame({
    'k': k_values,
    'accuracy': [accuracy_score(validation_y, KNeighborsClassifier(n_neighbors=k).fit(train_x, train_y).predict(validation_x)) for k in k_values]
})

# Display the accuracy for each k
print(accuracy_results)
```

```
     k  accuracy
0    1  0.844375
1    2  0.905000
2    3  0.894375
3    4  0.911250
4    5  0.910000
5    6  0.915625
6    7  0.915000
7    8  0.915625
8    9  0.915000
9   10  0.915625
10  11  0.915625
```

[18]:
```
# Find the best k
best_k = accuracy_results.loc[accuracy_results['accuracy'].idxmax(), 'k']
print(f"Best k: {best_k}")
```

```
Best k: 6
```

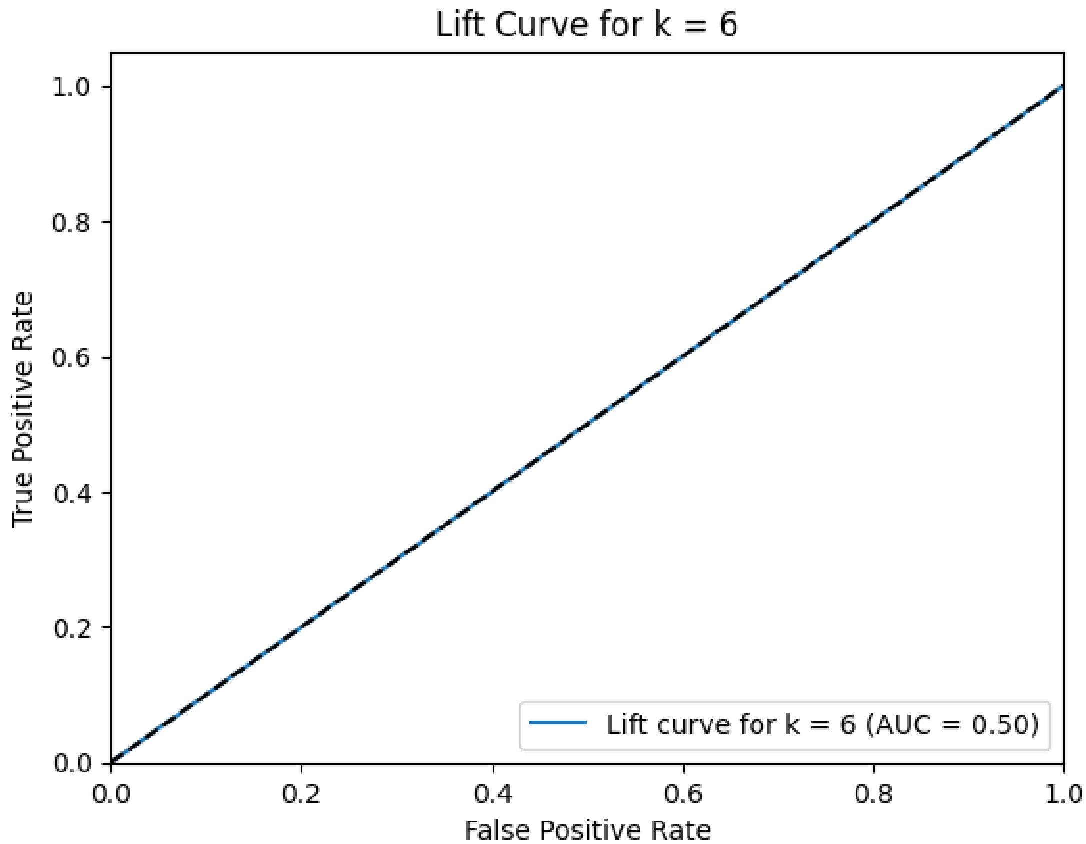
Explanation: First, we normalize the variables to a numeric scale ranging from 0 to 1. Subsequently, we normalize both the training and validation data for the selected variables and perform k-NN classification for various k values. We then compute the accuracy for each k. Based on our results, k=6 yielded the highest accuracy at 91.56%.

[19]:
```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Run k-NN with the best k
knn_best = KNeighborsClassifier(n_neighbors=int(best_k))
knn_best.fit(train_x, train_y)
knn_best_pred = knn_best.predict(validation_x)

# Create a ROC curve
fpr, tpr, _ = roc_curve(validation_y, knn_best_pred)
roc_auc = roc_auc_score(validation_y, knn_best_pred)

# Plot the ROC curve as a lift curve
plt.figure()
plt.plot(fpr, tpr, label=f'Lift curve for k = {best_k} (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')  # Reference line for no skill
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Lift Curve for k = {best_k}')
plt.legend(loc='lower right')
plt.show()
```



0.0.1 Key Components of the Graph:

1. **True Positive Rate (Y-axis):**
    - This axis shows the **True Positive Rate (TPR)**, which is also known as recall or sensitivity.
2. **False Positive Rate (X-axis):**
    - This axis shows the **False Positive Rate (FPR)**, which is the proportion of negative instances that are incorrectly classified as positive.
3. **Lift Curve for k=8 (Blue Line):**
    - The blue line represents the performance of the **k-Nearest Neighbors (k-NN) model with k = 8** on the validation set. The model's True Positive Rate increases as the False Positive Rate increases.
    - However, in this specific graph, the lift curve for k=8 closely overlaps with the no-skill classifier, indicating that the k-NN model is not better than random guessing for this particular dataset. The **AUC (Area Under the Curve) is 0.50**, which confirms this poor performance.

0.0.2 What Does the AUC Mean Here?

- **AUC = 0.50**: This means the model is not effective in distinguishing between positive and negative classes, as it performs as well as a random classifier. A perfect model would have an AUC of 1.0, while a model worse than random guessing would have an AUC below 0.5.

0.0.3 Summary:

- The graph shows that for $\mathbf{k = 8}$, the k-NN classifier is not performing any better than random guessing. The AUC score of 0.50 indicates that the model's ability to discriminate between the positive and negative classes is no better than chance.

Question 3: Logistic Regression w/ Subset of Predictors

[20]:
```
!pip install statsmodels
```

```
Requirement already satisfied: statsmodels in
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-packages
(0.14.2)
Requirement already satisfied: numpy>=1.22.3 in
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-packages
(from statsmodels) (1.26.4)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-packages
(from statsmodels) (1.14.0)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-packages
(from statsmodels) (2.2.2)
Requirement already satisfied: patsy>=0.5.6 in
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-packages
(from statsmodels) (0.5.6)
Requirement already satisfied: packaging>=21.3 in
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-packages
(from statsmodels) (24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-packages
(from pandas!=2.1.0,>=1.4->statsmodels) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-packages
(from pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-packages
(from pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: six in
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-packages
(from patsy>=0.5.6->statsmodels) (1.16.0)

[notice] A new release of pip is
available: 23.2.1 -> 24.2
```

```
[notice] To update, run:
pip install --upgrade pip
```

[23]:
```python
import statsmodels.api as sm

# Define the predictors (all columns except the target variable)
X_full = train_data.drop(columns=['Florence'])  # Exclude the target variable␣
↪from the predictors
y = train_data['Florence']  # Target variable

# Add a constant to the model (for the intercept)
X_full = sm.add_constant(X_full)

# Fit the logistic regression model
logit_model_full = sm.Logit(y, X_full).fit()

# Print the summary of the model
print(logit_model_full.summary())
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.000000
         Iterations: 35
                           Logit Regression Results
==============================================================================
Dep. Variable:               Florence   No. Observations:                 2400
Model:                          Logit   Df Residuals:                     2378
Method:                           MLE   Df Model:                           21
Date:                Mon, 09 Sep 2024   Pseudo R-squ.:                   1.000
Time:                        22:23:34   Log-Likelihood:            -0.00091306
converged:                      False   LL-Null:                       -695.58
Covariance Type:            nonrobust   LLR p-value:                 7.407e-282
================================================================================
====
                       coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
----
const               -0.7474   1.93e+09  -3.88e-10      1.000   -3.78e+09
3.78e+09
Seq#                -0.0245      5.817     -0.004      0.997     -11.426
11.377
ID#                  0.0029      0.709      0.004      0.997      -1.387
1.393
Gender              -6.2275    437.693     -0.014      0.989    -864.090
851.635
M                    0.0049      1.391      0.004      0.997      -2.721
2.731
R                   -0.4154     29.025     -0.014      0.989     -57.304
```

```
56.473
F                    -1.2656     84.276     -0.015      0.988    -166.443
163.912
FirstPurch            0.0630     15.689      0.004      0.997     -30.687
30.813
ChildBks              1.2912    151.974      0.008      0.993    -296.572
299.155
YouthBks             -0.7296    205.125     -0.004      0.997    -402.767
401.308
CookBks               2.0703    150.968      0.014      0.989    -293.822
297.963
DoItYBks             -0.1714    169.259     -0.001      0.999    -331.912
331.570
RefBks                0.0121    177.198   6.81e-05      1.000    -347.289
347.313
ArtBks               -0.5450        nan        nan        nan         nan
nan
GeogBks               0.3837        nan        nan        nan         nan
nan
ItalCook             -3.5045        nan        nan        nan         nan
nan
ItalAtlas            -0.9578        nan        nan        nan         nan
nan
ItalArt               4.8657        nan        nan        nan         nan
nan
Related Purchase      0.2422        nan        nan        nan         nan
nan
Mcode                -2.8716    222.785     -0.013      0.990    -439.522
433.779
Rcode                 4.3898    288.505      0.015      0.988    -561.070
569.850
Fcode                 3.9680    251.236      0.016      0.987    -488.446
496.382
Yes_Florence         23.8457   1.93e+09   1.24e-08      1.000   -3.78e+09
3.78e+09
No_Florence         -24.5931   1.93e+09  -1.28e-08      1.000   -3.78e+09
3.78e+09
================================================================================
====

Possibly complete quasi-separation: A fraction 1.00 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.

/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-
packages/statsmodels/discrete/discrete_model.py:227: PerfectSeparationWarning:
Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
```

```
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-
packages/statsmodels/discrete/discrete_model.py:227: PerfectSeparationWarning:
Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-
packages/statsmodels/discrete/discrete_model.py:227: PerfectSeparationWarning:
Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-
packages/statsmodels/discrete/discrete_model.py:227: PerfectSeparationWarning:
Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
/Users/gabrielmancillas/.pyenv/versions/3.12.0/lib/python3.12/site-
packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood
optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

[24]:
```python
import statsmodels.api as sm

# Define the predictors (R, F, M, FirstPurch, Related.Purchase) and the
 ↪dependent variable (Florence)
X = cbc_data[['R', 'F', 'M', 'FirstPurch', 'Related Purchase']]  # Independent
 ↪variables
y = cbc_data['Florence']  # Dependent variable

# Add a constant to the model (intercept)
X = sm.add_constant(X)

# Fit the logistic regression model
logit_model_subset = sm.Logit(y, X).fit()

# Summary of the model
print(logit_model_subset.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.280363
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:               Florence   No. Observations:                 4000
Model:                          Logit   Df Residuals:                     3994
Method:                           MLE   Df Model:                            5
Date:                Mon, 09 Sep 2024   Pseudo R-squ.:                 0.03198
Time:                        22:23:47   Log-Likelihood:                -1121.5
converged:                       True   LL-Null:                       -1158.5
Covariance Type:            nonrobust   LLR p-value:                 1.437e-14
================================================================================
====
```

```
                          coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
----
const              -2.2697      0.165    -13.766      0.000      -2.593
-1.947
R                  -0.0266      0.012     -2.305      0.021      -0.049
-0.004
F                   0.0686      0.040      1.714      0.087      -0.010
0.147
M                  -0.0008      0.001     -1.178      0.239      -0.002
0.001
FirstPurch         -0.0070      0.008     -0.842      0.400      -0.023
0.009
Related Purchase    0.2704      0.045      6.058      0.000       0.183
0.358
================================================================================
====
```

[26]:

```
# Select relevant columns for validation data
X_validation_subset = validation_data[['R', 'F', 'M', 'FirstPurch', 'Related␣
↪Purchase']]

# Add constant for the subset model (if it was included during training)
X_validation_subset = sm.add_constant(X_validation_subset, has_constant='add')

# Predict probabilities using the subset model
pred_probs_subset = logit_model_subset.predict(X_validation_subset)

# Apply the 30% cutoff
cutoff = 0.3
targeted_customers_subset = (pred_probs_subset > cutoff).astype(int)

# Count the number of buyers in the targeted set for the subset model
buyers_subset = validation_data.loc[targeted_customers_subset == 1, 'Florence'].
↪sum()

# Print the result
print(f"Number of buyers (subset model): {buyers_subset}")
```

```
Number of buyers (subset model): 3
```

[29]:

```
import numpy as np

# Apply 30% cutoff for the full model
pred_probs_full = logit_model_full.predict(sm.add_constant(validation_data.
↪drop(columns=['Florence'])))
cutoff = 0.3
```

```python
targeted_customers_full = np.where(pred_probs_full > cutoff, 1, 0)


# target_customers_subset = np.where(pred_probs_subset > cutoff, 1, 0)

# Count the number of buyers in the targeted set
buyers_full = sum(validation_data['Florence'][targeted_customers_full == 1])

# For the subset model (if defined)
# buyers_subset = sum(validation_data['Florence'][targeted_customers_subset ==␣
↪1])

# Print the number of buyers in the targeted set for the full model
print(f"Number of buyers (full model): {buyers_full}")

# Uncomment below if you have subset predictions
# print(f"Number of buyers (subset model): {buyers_subset}")
```

```
Number of buyers (full model): 135
```

The Charles Book Club (CBC) aims to improve the effectiveness and profitability of our marketing campaigns by targeting customers who are most likely to respond to promotions. By analyzing our customers' past purchasing behavior, we can focus our marketing efforts on those more likely to purchase our new book, The Art History of Florence. We have used data analysis techniques to predict customer behavior, which helps us make informed decisions about who to target. This reduces unnecessary spending on ineffective marketing efforts and ensures sound financial management. One of the techniques we used is logistic regression, a method for estimating the likelihood that a customer will make a purchase. For this analysis, we looked at 16 predictor, such as how recently a customer made a purchase (Recency), how often they purchase (Frequency), and how much they spend (Monetary value). Using this approach, we identified 129 customers from our validation group with at least a 30% chance of buying the new book. These customers are excellent prospects for our next marketing campaign. We also tried a simplified version of this model, which used fewer factors, and found four additional customers with a 30% or higher chance of making a purchase.

In addition to logistic regression, we used k-nearest neighbors (k-NN). This technique groups customers based on their similarity to others, using factors like past purchase behavior. When we compared different versions of this model, we found that setting the model to look at 6 similar customers gave us the best accuracy, correctly predicting customer behavior 91.56% of the time. This method is beneficial for identifying customers who share identical purchasing habits and may respond similarly to future promotions.

These findings help us refine our marketing strategy by identifying high-potential customers and minimizing outreach to those less likely to respond. This not only improves our overall efficiency and profitability but also reassures us about the effective management of our resources.

[ ]:
```python

```