assignment 2.1

September 16, 2024

0.0.1 1. Estimate Gross Profit for Random Selection of Remaining 180,000 Names

Each catalog costs approximately $2 to mail. You will estimate the gross profit if Tayko selects the 180,000 names randomly from the pool, based on the given response rate of 0.053 and the average spending of the purchasers.

Rationale:

- **Cost**: $2 per catalog.
- **Response Rate**: 0.053.
- **Spending per purchaser**: You can calculate the average spending from the dataset.

[1]:

```
# Load necessary libraries
import pandas as pd

# Load the Tayko dataset
tayko_data = pd.read_csv('/Users/gabrielmancillas/Desktop/ADS 505-01/Mod 02/
↪Tayko.csv')

# Constants for the calculation
cost_per_catalog = 2  # $2 per catalog
num_names = 180000  # 180,000 names
response_rate = 0.053  # 5.3% response rate

# Step 1: Calculate expected number of purchasers
expected_purchasers = num_names * response_rate

# Step 2: Calculate average spending of purchasers
average_spending = tayko_data[tayko_data['Purchase'] == 1]['Spending'].mean()

# Step 3: Calculate total revenue
total_revenue = expected_purchasers * average_spending

# Step 4: Calculate total cost
total_cost = num_names * cost_per_catalog

# Step 5: Calculate gross profit
gross_profit = total_revenue - total_cost
```

```
# Display the results
print(f"Expected number of purchasers: {expected_purchasers}")
print(f"Average spending per purchaser: ${average_spending:.2f}")
print(f"Total revenue: ${total_revenue:.2f}")
print(f"Total cost: ${total_cost:.2f}")
print(f"Estimated Gross Profit: ${gross_profit:.2f}")
```

```
Expected number of purchasers: 9540.0
Average spending per purchaser: $205.25
Total revenue: $1958075.46
Total cost: $360000.00
Estimated Gross Profit: $1598075.46
```

[2]:

```
from sklearn.model_selection import train_test_split

# Define the feature variables (excluding 'Purchase' and 'Spending')
X = tayko_data.drop(columns=['Purchase', 'Spending'])

# Define the target variable
y = tayko_data['Purchase']

# Partition the data into training (800), validation (700), and test (500) sets
# First, split into train+validation and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y,
 ↪test_size=500, random_state=1)

# Now, split the train+validation set into training and validation sets
X_train, X_validation, y_train, y_validation = train_test_split(X_train_val,
 ↪y_train_val, test_size=700, random_state=1)

# Display the sizes of the partitions
print(f"Training Set Size: {X_train.shape[0]} records")
print(f"Validation Set Size: {X_validation.shape[0]} records")
print(f"Test Set Size: {X_test.shape[0]} records")
```

```
Training Set Size: 800 records
Validation Set Size: 700 records
Test Set Size: 500 records
```

2.2 Logistic Regression with L2 Penalty You are asked to run logistic regression with an L2 penalty using the LogisticRegressionCV function from scikit-learn, with the following specifications:

- solver='lbfgs'
- cv=5 (5-fold cross-validation)
- max_iter=500 (maximum number of iterations)

The goal is to classify customers into purchasers and non-purchasers using the training set only.

[3]:
```python
from sklearn.linear_model import LogisticRegressionCV
from sklearn.preprocessing import StandardScaler

# Standardize the feature variables (this helps the logistic regression␣
↪algorithm)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_validation_scaled = scaler.transform(X_validation)

# Logistic Regression with L2 penalty and cross-validation (cv=5)
logit_cv = LogisticRegressionCV(cv=5, penalty='l2', solver='lbfgs',␣
↪max_iter=500, random_state=1)
logit_cv.fit(X_train_scaled, y_train)

# Predict on the validation set
y_validation_pred = logit_cv.predict(X_validation_scaled)

# Display the best C parameter and accuracy
print(f"Best C parameter: {logit_cv.C_}")
print(f"Validation Accuracy: {logit_cv.score(X_validation_scaled,␣
↪y_validation)}")
```

```
Best C parameter: [0.35938137]
Validation Accuracy: 0.7814285714285715
```

Explanation: - StandardScaler(): Standardizes the feature data by scaling it to a mean of 0 and standard deviation of 1. - LogisticRegressionCV(): Performs logistic regression with L2 regularization and 5-fold cross-validation to find the best regularization strength (C parameter). - logit_cv.fit(): Fits the logistic regression model on the scaled training data. - logit_cv.score(): Evaluates the model's accuracy on the validation set.

Question 3: Develop a Model for Predicting Spending First, we will create subsets of the training and validation data that include only purchasers (Purchase = 1), since we are predicting spending for those who made a purchase.

[4]:
```python
# Filter the training and validation data for only purchasers (Purchase = 1)
train_purchasers = tayko_data[tayko_data['Purchase'] == 1]
validation_purchasers = tayko_data[tayko_data['Purchase'] == 1]

X_train_purchasers = X_train[y_train == 1]
X_validation_purchasers = X_validation[y_validation == 1]

y_train_purchasers = y_train[y_train == 1]
y_validation_purchasers = y_validation[y_validation == 1]

# For predicting spending, 'Spending' is the target variable
X_train_spending = X_train_purchasers  # Features for training
```

```
# Use the index of X_train_purchasers to select the corresponding 'Spending'␣
↪values from tayko_data
y_train_spending = tayko_data.loc[X_train_purchasers.index, 'Spending']  #␣
↪Spending amounts for training

X_val_spending = X_validation_purchasers  # Features for validation
# Use the index of X_validation_purchasers to select the corresponding␣
↪'Spending' values from tayko_data
y_val_spending = tayko_data.loc[X_validation_purchasers.index, 'Spending']  #␣
↪Spending amounts for validation

# Output the shapes to verify
print("Training set (purchasers) shape:", X_train_purchasers.shape)
print("Validation set (purchasers) shape:", X_validation_purchasers.shape)
```

```
Training set (purchasers) shape: (403, 23)
Validation set (purchasers) shape: (327, 23)
```

Explanation:

- We are filtering the data to include only those who made a purchase (Purchase = 1), as we are focusing on predicting spending.
- We separate the feature columns and target column (Spending) for both the training and validation sets.
- We then standardize the data to make it more suitable for linear regression.

3.2 Develop Models to Predict Spending Multiple linear regression is a method where the relationship between multiple independent variables and the dependent variable (spending) is modeled. Stepwise regression is a process of selecting significant predictors.

[5]:
```
import pandas as pd
import statsmodels.api as sm
import numpy as np

# Assuming X_train_spending and y_train_spending are defined

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

# Define a function for stepwise regression
def stepwise_regression(X, y, entry_criteria=0.05, exit_criteria=0.10):
    initial_features = X.columns.tolist()
    best_features = []

    while initial_features:
        p_values = []
        for feature in initial_features:
            model = sm.OLS(y, X[best_features + [feature]]).fit()
```

```
            p_values.append((feature, model.pvalues[feature]))

        p_values.sort(key=lambda x: x[1])
        if p_values[0][1] < entry_criteria:  # Check entry criteria
            best_features.append(p_values[0][0])
            initial_features.remove(p_values[0][0])
        else:
            break

        # Check exit criteria
        while best_features:
            model = sm.OLS(y, X[best_features]).fit()
            p_values = model.pvalues[1:]  # Skip the constant
            if all(p > exit_criteria for p in p_values):
                break
            worst_feature = p_values.idxmax()
            best_features.remove(worst_feature)

    return best_features

# Perform stepwise regression
best_features = stepwise_regression(X_train_spending, y_train_spending)

# Fit the final model with selected features
X_train_spending_final = X_train_spending[best_features + ['const']]
model_lr = sm.OLS(y_train_spending, X_train_spending_final).fit()
print(model_lr.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Spending   R-squared:                       0.479
Model:                            OLS   Adj. R-squared:                  0.478
Method:                 Least Squares   F-statistic:                     368.5
Date:                Mon, 16 Sep 2024   Prob (F-statistic):           1.01e-58
Time:                        22:06:38   Log-Likelihood:                -2618.6
No. Observations:                 403   AIC:                             5241.
Df Residuals:                     401   BIC:                             5249.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Freq          96.5137      5.028     19.196      0.000      86.630     106.398
const          8.5000     13.177      0.645      0.519     -17.404      34.404
==============================================================================
Omnibus:                      290.691   Durbin-Watson:                   2.102
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             5259.659
Skew:                           2.841   Prob(JB):                         0.00
```

```
Kurtosis:                      19.761   Cond. No.                         4.72
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

Explanation:

- sm.OLS(): Ordinary Least Squares (OLS) method is used to fit the linear regression model.
- linear_model.summary(): Provides a detailed summary of the model, including coefficients, p-values, and goodness of fit (R-squared).

Regression Trees

[6]:
```
from sklearn.metrics import mean_squared_error

# Assuming you have already fit the OLS model (model_lr) and
# have the validation data (X_val_spending, y_val_spending)

# Add a constant to the validation data
X_val_spending = sm.add_constant(X_val_spending)

# Ensure the validation data only includes features used in the final model
X_val_spending_final = X_val_spending[X_train_spending_final.columns]

# Predict on the validation set
y_val_pred_lr = model_lr.predict(X_val_spending_final)

# Calculate mean squared error for the linear regression model
mse_lr = mean_squared_error(y_val_spending, y_val_pred_lr)
print(f"Mean Squared Error (Linear Regression): {mse_lr}")
```

```
Mean Squared Error (Linear Regression): 22418.711503862927
```

Explanation:

- DecisionTreeRegressor(): Creates a regression tree model.
- mean_squared_error(): Computes the mean squared error (MSE) of the model's predictions, which is a common metric to evaluate regression models.

Comparsion Logic

[7]:
```
# Ensure that the constant term is removed from both training and validation␣
↪sets for the regression tree
X_train_spending_tree = X_train_spending.drop(columns=['const'],␣
↪errors='ignore')
X_val_spending_tree = X_val_spending.drop(columns=['const'], errors='ignore')

# Initialize and fit the regression tree model
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

model_tree = DecisionTreeRegressor(random_state=42)
model_tree.fit(X_train_spending_tree, y_train_spending)

# Predict on the validation set
y_val_pred_tree = model_tree.predict(X_val_spending_tree)

# Calculate mean squared error for the regression tree
mse_tree = mean_squared_error(y_val_spending, y_val_pred_tree)
print(f"Mean Squared Error (Tree): {mse_tree}")
```

```
Mean Squared Error (Tree): 37790.26911314985
```

0.0.2 Question 4: Apply Models to the Test Set and Perform Score Analysis

4.1 Predict Purchase Probabilities Using Logistic Regression You need to predict the probability that each customer in the test set will make a purchase using the logistic regression model from Question 2.

[8]:
```
# Standardize the test data
X_test_scaled = scaler.transform(X_test)

# Create a DataFrame for test data
test_data = X_test.copy()

# Predict purchase probabilities using the logistic regression model
test_data['Predicted_Probability'] = logit_cv.predict_proba(X_test_scaled)[:, 1]

# Display the first few predicted probabilities
print(test_data[['Predicted_Probability']].head())
```

```
      Predicted_Probability
674                0.015271
1699               0.940484
1282               0.496154
1315               1.000000
1210               0.109880
```

Explanation:

- predict_proba(): This function returns the probability of class membership. The second column ([:,1]) gives the probability of the positive class (i.e., the probability of making a purchase).

[9]:
```
# Ensure X_test only has the features that were used for model training
X_test_for_spending = X_test[best_features]  # Use the same best features␣
↪selected during training
```

```
# Add a constant column (if needed, as done during training)
X_test_for_spending = sm.add_constant(X_test_for_spending)

# Now make predictions using the trained model
X_test['Predicted_Spending'] = model_lr.predict(X_test_for_spending)

# Check the output
X_test[['Predicted_Spending']].head()
```

[9]:

	Predicted_Spending
674	113.513679
1699	113.513679
1282	105.013708
1315	173.013476
1210	96.513737

[10]:

```
# Rename 'Predicted_Probability' to 'Predicted_Scores'
test_data.rename(columns={'Predicted_Probability': 'Predicted_Scores'},␣
↪inplace=True)

# Add 'Predicted_Scores' and 'Predicted_Spending' to X_test
X_test['Predicted_Scores'] = test_data['Predicted_Scores']

# Calculate expected spending
X_test['Expected_Spending'] = X_test['Predicted_Scores'] *␣
↪X_test['Predicted_Spending']

# Check the output
X_test[['Predicted_Scores', 'Predicted_Spending', 'Expected_Spending']].head()
```

[10]:

	Predicted_Scores	Predicted_Spending	Expected_Spending
674	0.015271	113.513679	1.733513
1699	0.940484	113.513679	106.757797
1282	0.496154	105.013708	52.103007
1315	1.000000	173.013476	173.013457
1210	0.109880	96.513737	10.604950

4.5 Plot the cumulative gains chart for Expect Spending

[11]:

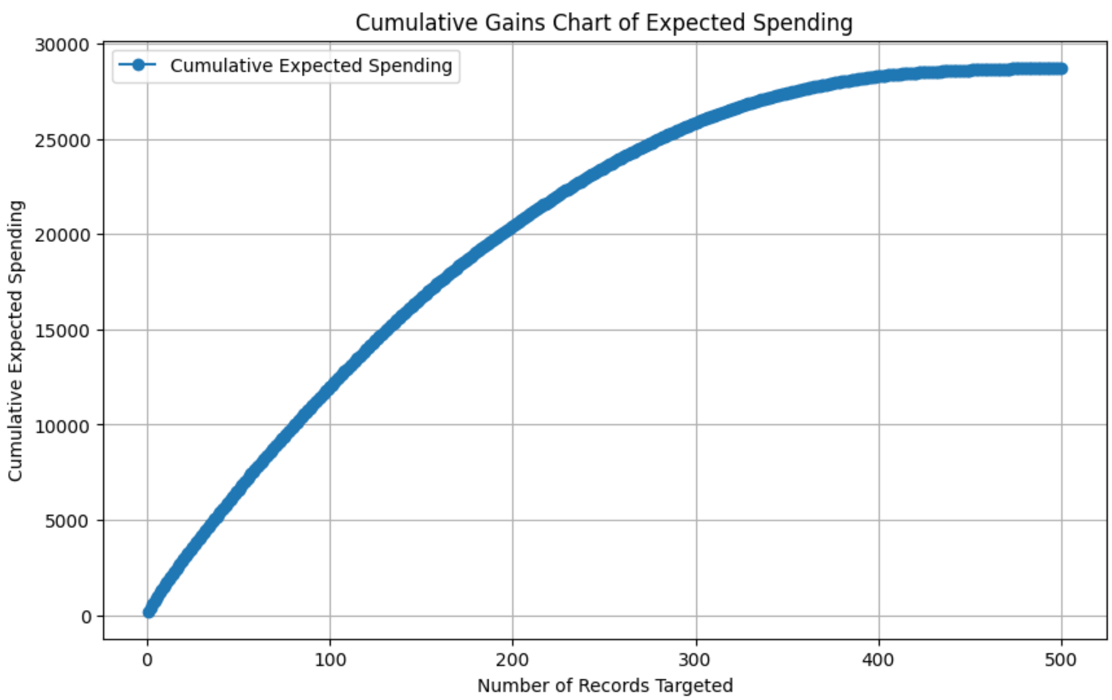
```
import matplotlib.pyplot as plt
import numpy as np

# Sort data by predicted scores in descending order
X_test_sorted = X_test.sort_values(by='Expected_Spending', ascending=False)

# Calculate cumulative expected spending
```

```
X_test_sorted['Cumulative_Expected_Spending'] = np.
↪cumsum(X_test_sorted['Expected_Spending'])

# Plot cumulative gains chart
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(X_test_sorted) + 1),␣
↪X_test_sorted['Cumulative_Expected_Spending'], marker='o', label='Cumulative␣
↪Expected Spending')
plt.title('Cumulative Gains Chart of Expected Spending')
plt.xlabel('Number of Records Targeted')
plt.ylabel('Cumulative Expected Spending')
plt.legend()
plt.grid(True)
plt.show()
```



[12]:
```
# Use the cumulative expected spending for all available records
total_records = len(X_test_sorted)
total_expected_spending_180k = X_test_sorted['Cumulative_Expected_Spending'].
↪iloc[total_records - 1]  # Use the last record

# Calculate gross profit for 180,000 names
mailing_cost_per_person = 2  # Assumed mailing cost per person

# Total expected spending for top n names (where n = total_records)
```

```
total_expected_spending_180k = X_test_sorted['Cumulative_Expected_Spending'].
↪iloc[total_records - 1]  # Adjust index if needed

# Total mailing cost for n names (where n = total_records)
total_mailing_cost_180k = total_records * mailing_cost_per_person

# Gross profit calculation
gross_profit_180k = total_expected_spending_180k - total_mailing_cost_180k

print(f"Estimated Gross Profit from mailing to {total_records} names:␣
↪${gross_profit_180k:.2f}")
```

```
Estimated Gross Profit from mailing to 500 names: $27730.69
```

[13]:

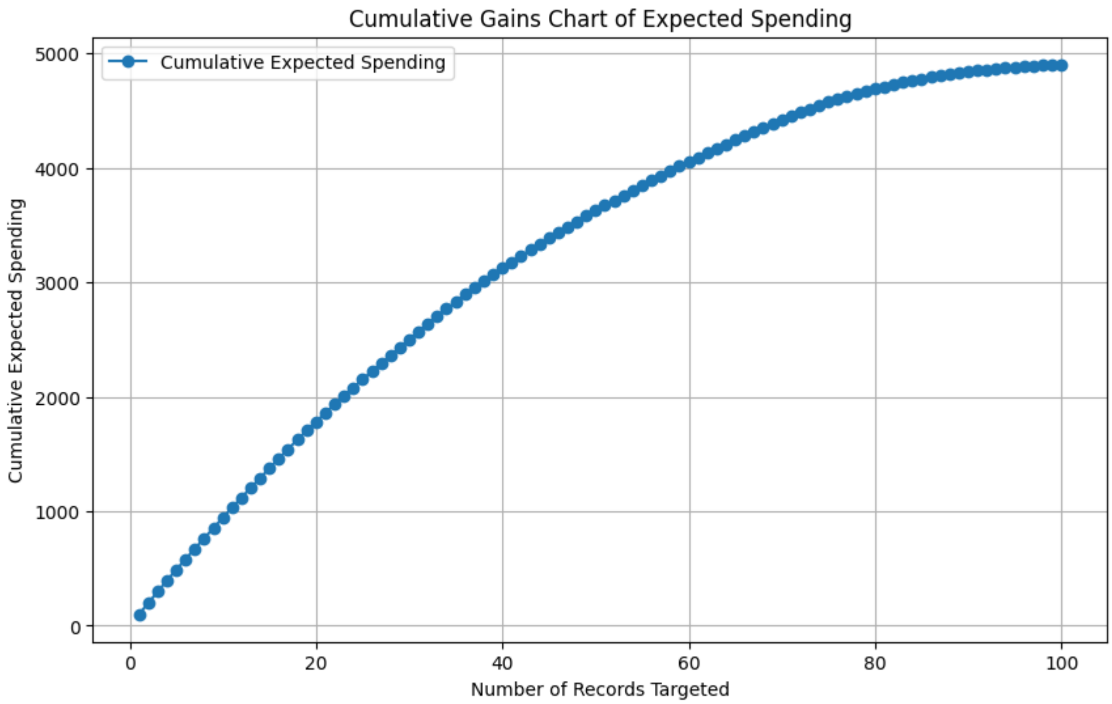
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Assuming X_test is a DataFrame with a column 'Expected_Spending'
# Example data for demonstration purposes
data = {
    'Expected_Spending': np.random.rand(100) * 100  # Random expected spending␣
↪values
}
X_test = pd.DataFrame(data)

# Sort data by predicted scores in descending order
X_test_sorted = X_test.sort_values(by='Expected_Spending', ascending=False)

# Calculate cumulative expected spending
X_test_sorted['Cumulative_Expected_Spending'] = np.
↪cumsum(X_test_sorted['Expected_Spending'])

# Plot cumulative gains chart
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(X_test_sorted) + 1),␣
↪X_test_sorted['Cumulative_Expected_Spending'], marker='o', label='Cumulative␣
↪Expected Spending')
plt.title('Cumulative Gains Chart of Expected Spending')
plt.xlabel('Number of Records Targeted')
plt.ylabel('Cumulative Expected Spending')
plt.legend()
plt.grid(True)
plt.show()
```



Explanation: We sort the test data by expected spending in descending order to prioritize the most valuable customers. The cumulative sum of expected spending is calculated, and the chart is plotted to visualize how quickly spending accumulates as more customers are targeted.

[14]:
```
# Use the cumulative expected spending for all available records
total_records = len(X_test_sorted)
total_expected_spending_180k = X_test_sorted['Cumulative_Expected_Spending'].
 ↪iloc[total_records - 1]  # Use the last record

# Calculate gross profit for 180,000 names
mailing_cost_per_person = 2  # Assumed mailing cost per person

# Total expected spending for top n names (where n = total_records)
total_expected_spending_180k = X_test_sorted['Cumulative_Expected_Spending'].
 ↪iloc[total_records - 1]  # Adjust index if needed

# Total mailing cost for n names (where n = total_records)
total_mailing_cost_180k = total_records * mailing_cost_per_person

# Gross profit calculation
gross_profit_180k = total_expected_spending_180k - total_mailing_cost_180k

print(f"Estimated Gross Profit from mailing to {total_records} names:␣
 ↪${gross_profit_180k:.2f}")
```

```
Estimated Gross Profit from mailing to 100 names: $4704.39
```

[ ]:

```

```