



Uncovering the Subscription Decline: A Data-Driven Behavioral Analysis

Samidha Rathore



Project Objective

To analyze trends in customer subscriptions, evaluate behavioral data, and uncover actionable insights that explain the decline in subscriptions.

This report aims to provide transparency with clear, non-AI-generated analysis backed by visualizations and metrics.



Table of Contents

1. [Project Objective](#)
2. [Introduction](#)
3. [Part 1: Data Cleaning & Exploration](#)
4. [Part 2: Logistic Regression Modeling](#)
5. [Part 3: Interpretation & Recommendations](#)
6. [Part 4: Model Comparison](#)
7. [Conclusion](#)
8. [References](#)

Introduction

The company saw a drop in digital magazine subscriptions last year, and we wanted to understand why. This report looks into customer data to find patterns in behavior, spending, and campaign responses. We cleaned the data, filled in missing info, and created new features to dig deeper. The goal is to figure out what's driving the decline—and more importantly, what we can do about it. This analysis focuses on real insights, not generic summaries, to help guide smarter decisions going forward.

Part 1: Data Cleaning & Exploration

In this part of the project, we start by loading the dataset and reviewing the structure of the data, including data types and missing values. We found that the Income column had missing entries, so we handled it using two different methods: one by filling with the median income, and another by predicting income using a decision tree based on age and family size.

Next, we converted categorical variables like Education and Marital_Status into numerical format using one-hot encoding so they can be used in modeling. We also created new

features such as TotalChildren (number of kids and teens at home) and CustomerAge, which may help us understand subscription behavior better.

Finally, we used a few basic visualizations to explore the data and look for early patterns. This includes viewing income distribution, checking relationships between variables using a heatmap, and plotting how the number of children might relate to subscription responses.

```
In [10]: # Import libraries
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import warnings
warnings.filterwarnings("ignore")
```

```
In [2]: # Load the dataset and inspect column types and missing values
df = pd.read_excel('marketing_campaign.xlsx')
df['Dt_Customer'] = pd.to_datetime(df['Dt_Customer'])
df['Year_Customer'] = df['Dt_Customer'].dt.year
print(df.info())
print("\nMissing values:\n", df.isnull().sum())
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 30 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   ID                                    2240 non-null   int64
1   Year_Birth                           2240 non-null   int64
2   Education                             2240 non-null   object
3   Marital_Status                       2240 non-null   object
4   Income                               2216 non-null   float64
5   Kidhome                              2240 non-null   int64
6   Teenhome                             2240 non-null   int64
7   Dt_Customer                           2240 non-null   datetime64[ns]
8   Recency                              2240 non-null   int64
9   MntWines                             2240 non-null   int64
10  MntFruits                             2240 non-null   int64
11  MntMeatProducts                       2240 non-null   int64
12  MntFishProducts                       2240 non-null   int64
13  MntSweetProducts                      2240 non-null   int64
14  MntGoldProds                          2240 non-null   int64
15  NumDealsPurchases                     2240 non-null   int64
16  NumWebPurchases                       2240 non-null   int64
17  NumCatalogPurchases                   2240 non-null   int64
18  NumStorePurchases                     2240 non-null   int64
19  NumWebVisitsMonth                     2240 non-null   int64
20  AcceptedCmp3                          2240 non-null   int64
21  AcceptedCmp4                          2240 non-null   int64
22  AcceptedCmp5                          2240 non-null   int64
23  AcceptedCmp1                          2240 non-null   int64
24  AcceptedCmp2                          2240 non-null   int64
25  Complain                              2240 non-null   int64
26  Z_CostContact                         2240 non-null   int64
27  Z_Revenue                             2240 non-null   int64
28  Response                              2240 non-null   int64
29  Year_Customer                         2240 non-null   int32
dtypes: datetime64[ns](1), float64(1), int32(1), int64(25), object(2)
memory usage: 516.4+ KB
None

```

Missing values:

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
Year_Customer     0
dtype: int64

```

```

In [3]: # Handle missing data

# Method 1: Median imputation
df_median = df.copy()
df_median['Income'] = df_median['Income'].fillna(df_median['Income'].median())

```

```

In [4]: # Method 2: Predictive imputation using Decision Tree
df_tree = df.copy()
df_tree['Family_Size'] = df_tree['Kidhome'] + df_tree['Teenhome']
df_tree['Age'] = 2025 - df_tree['Year_Birth']
train = df_tree[df_tree['Income'].notnull()]
test = df_tree[df_tree['Income'].isnull()]
model = DecisionTreeRegressor()
model.fit(train[['Age', 'Family_Size']], train['Income'])
df_tree.loc[test.index, 'Income'] = model.predict(test[['Age', 'Family_Size']])

```

```

In [5]: # Convert categorical variables using one-hot encoding
df_clean = df_median.copy() # using median-imputed data for consistency
df_clean = pd.get_dummies(df_clean, columns=['Education', 'Marital_Status'], d

```

```

In [6]: # Create engineered features
df_clean['TotalChildren'] = df_clean['Kidhome'] + df_clean['Teenhome']
df_clean['CustomerAge'] = 2025 - df_clean['Year_Birth']

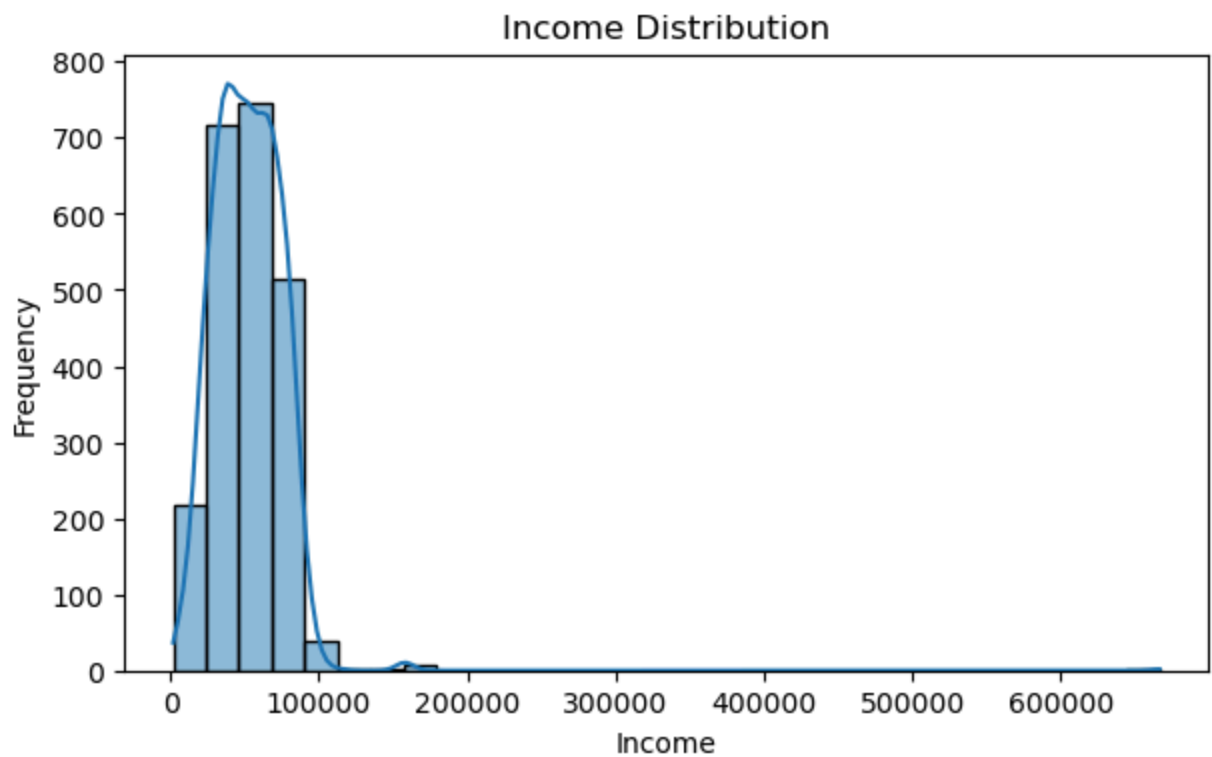
```

EDA visualizations

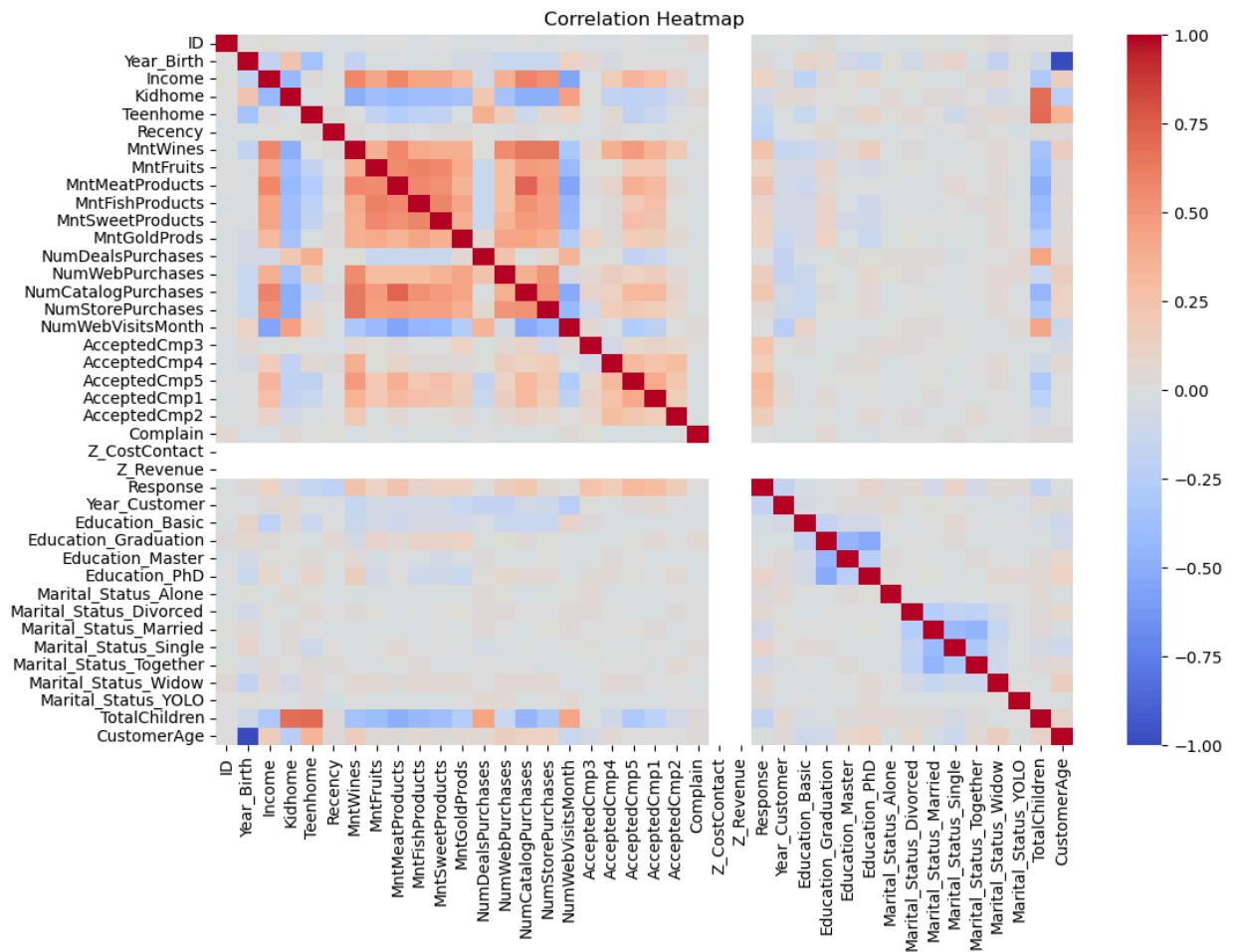
```

In [7]: # Income distribution
plt.figure(figsize=(7, 4))
sns.histplot(df_clean['Income'], bins=30, kde=True)
plt.title('Income Distribution')
plt.xlabel('Income')
plt.ylabel('Frequency')
plt.show()

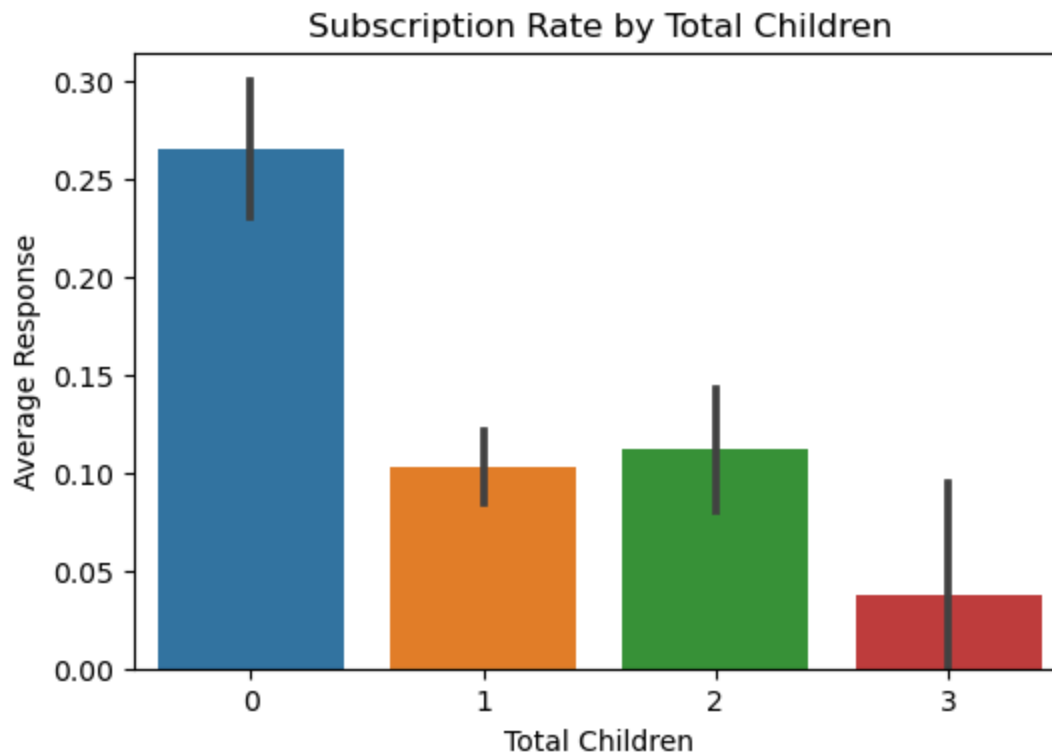
```



```
In [8]: # Correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df_clean.corr(numeric_only=True), cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



```
In [9]: # Bar plot: Subscription rate by number of children
plt.figure(figsize=(6, 4))
sns.barplot(x='TotalChildren', y='Response', data=df_clean)
plt.title('Subscription Rate by Total Children')
plt.xlabel('Total Children')
plt.ylabel('Average Response')
plt.show()
```



Interpretation:

The income distribution shows that most customers earn between ₹25,000 and ₹100,000, with a few high-income outliers, justifying the use of median imputation. The correlation heatmap reveals that no single feature strongly predicts subscriptions, but purchasing behavior and family size show some moderate relationships. The bar chart indicates that customers with no children are more likely to subscribe, while the subscription rate drops as the number of children increases—highlighting family size as an important factor.

Part 2: Logistic Regression Modeling

In this part, we build a logistic regression model to predict whether a customer will subscribe. We first split the cleaned dataset into training and test sets (80/20 ratio). Then we trained the model and evaluated its performance using accuracy, precision, recall, and F1 score. Finally, we visualized the confusion matrix to better understand how well the model identifies subscribers versus non-subscribers.

```
In [11]: # Define features and target
X = df_clean.drop(columns=['ID', 'Response', 'Dt_Customer', 'Z_CostContact', 'Z_Recency'])
y = df_clean['Response']
```

```
In [12]: # Split into training and testing sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
In [13]: # Build logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

Out [13]:

LogisticRegression

LogisticRegression(max_iter=1000)

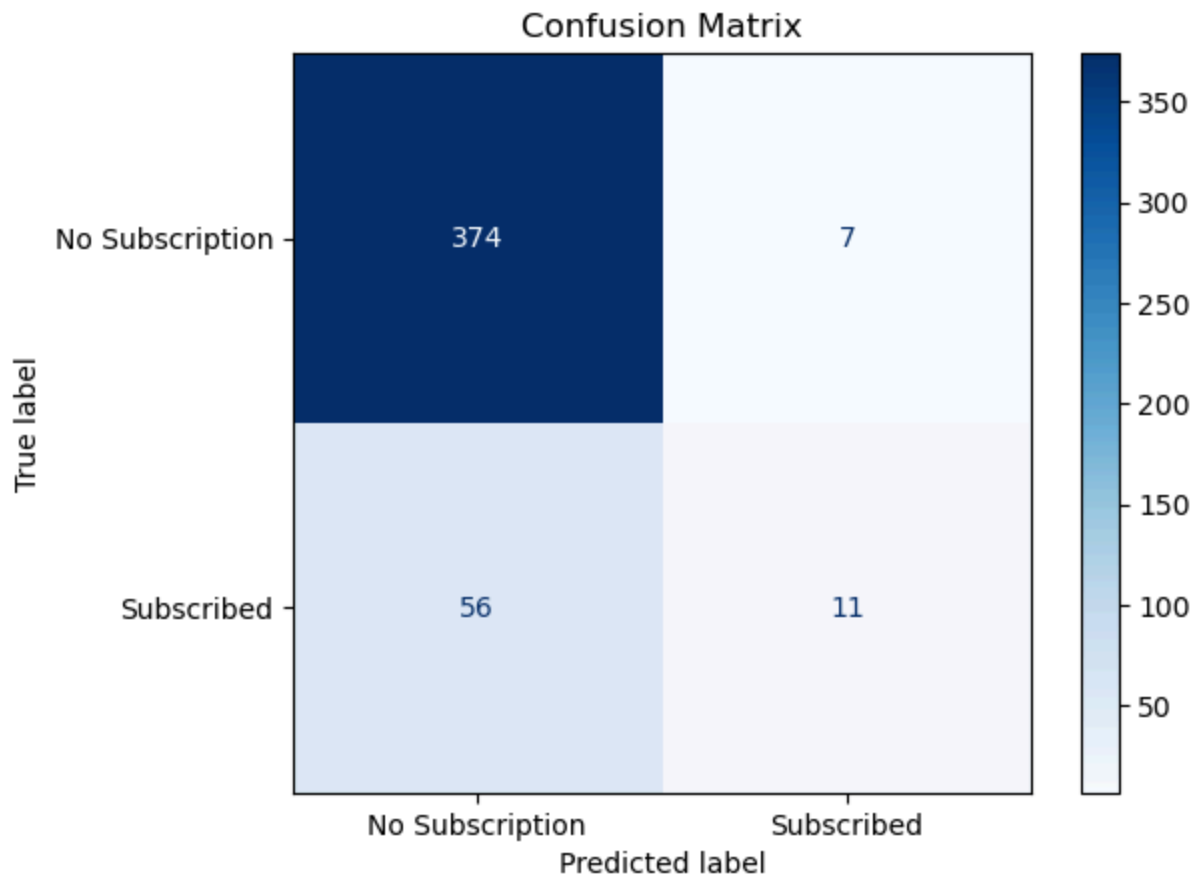
```
In [14]: # Make predictions
y_pred = model.predict(X_test)
```

```
In [15]: # Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

```
In [16]: # Display metrics in a table
metrics = pd.DataFrame({
    'Metric': ['Accuracy', 'Precision', 'Recall', 'F1 Score'],
    'Score': [accuracy, precision, recall, f1]
})
print(metrics)
```

	Metric	Score
0	Accuracy	0.859375
1	Precision	0.611111
2	Recall	0.164179
3	F1 Score	0.258824

```
In [17]: # Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["No Subscription", '
disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.grid(False)
plt.show()
```

Interpretation:

The logistic regression model shows good overall accuracy (86%), meaning it makes correct predictions most of the time. It has a decent precision of 61%, which means when it predicts a customer will subscribe, it's usually right. However, the recall is quite low (16%), indicating the model misses many actual subscribers. The F1 score (26%) reflects this imbalance. From the confusion matrix, we see that the model correctly identified 374 non-subscribers and only 11 actual subscribers, while it failed to detect 56 real subscribers. This suggests the model is conservative—better at identifying who won't subscribe than who will.

Part 3: Interpretation & Recommendations

In this section, we interpret our logistic regression model to understand what influences subscription decisions. We identify the top 3 most important features based on the size of their coefficients. Then, we explain what a positive or negative value means in practical terms and suggest possible business actions. Lastly, we compare how the model performs when using different prediction thresholds (0.5 vs. 0.6) to see if adjusting the cutoff could improve targeting.

```
In [19]: # Get feature importance from logistic regression
feature_names = X.columns
coefficients = model.coef_[0]
coef_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})
```

```
coef_df['Abs_Coefficient'] = coef_df['Coefficient'].abs()
top_features = coef_df.sort_values(by='Abs_Coefficient', ascending=False).head
print("Top 3 Influential Features:\n", top_features[['Feature', 'Coefficient']])
```

Top 3 Influential Features:

	Feature	Coefficient
14	NumStorePurchases	-0.115830
15	NumWebVisitsMonth	0.067892
13	NumCatalogPurchases	0.032910

Interpretation of Top 3 Influential Features

- **NumStorePurchases (-0.116)**

- A negative coefficient means that as the number of store purchases increases, the likelihood of subscribing decreases.
- Business Action: These customers may prefer in-person experiences. Offer digital-exclusive content or loyalty points to encourage them to try the subscription.

- **NumWebVisitsMonth (+0.068)**

- A positive coefficient means that more website visits increase the chances of subscribing.
- Business Action: Target frequent website visitors with personalized pop-ups, subscription discounts, or reminders to subscribe.

- **NumCatalogPurchases (+0.033)**

- Also a positive coefficient—catalog buyers are slightly more likely to subscribe.
- Business Action: Add subscription offers or bundled deals directly in catalogs or follow up with email campaigns focused on this segment.

```
In [21]: # Generate predicted probabilities (if not already done)
y_prob = model.predict_proba(X_test)[: , 1]
```

```
In [22]: # 🔍 Threshold analysis: 0.5 vs. 0.6
y_pred_05 = (y_prob >= 0.5).astype(int)
y_pred_06 = (y_prob >= 0.6).astype(int)
```

```
In [23]: # Function to print classification metrics
from sklearn.metrics import classification_report

def print_metrics(y_true, y_pred, threshold):
    print(f"\nClassification Report at Threshold = {threshold}")
    print(classification_report(y_true, y_pred, target_names=["No Subscription", "Subscription"]))

# Print metrics for both thresholds
print_metrics(y_test, y_pred_05, 0.5)
print_metrics(y_test, y_pred_06, 0.6)

# Optional: Check how many positives were predicted
print("\nPositive Predictions at 0.5:", sum(y_pred_05))
print("Positive Predictions at 0.6:", sum(y_pred_06))
```

Classification Report at Threshold = 0.5

	precision	recall	f1-score	support
No Subscription	0.87	0.98	0.92	381
Subscribed	0.61	0.16	0.26	67
accuracy			0.86	448
macro avg	0.74	0.57	0.59	448
weighted avg	0.83	0.86	0.82	448

Classification Report at Threshold = 0.6

	precision	recall	f1-score	support
No Subscription	0.87	0.99	0.92	381
Subscribed	0.64	0.13	0.22	67
accuracy			0.86	448
macro avg	0.75	0.56	0.57	448
weighted avg	0.83	0.86	0.82	448

Positive Predictions at 0.5: 18

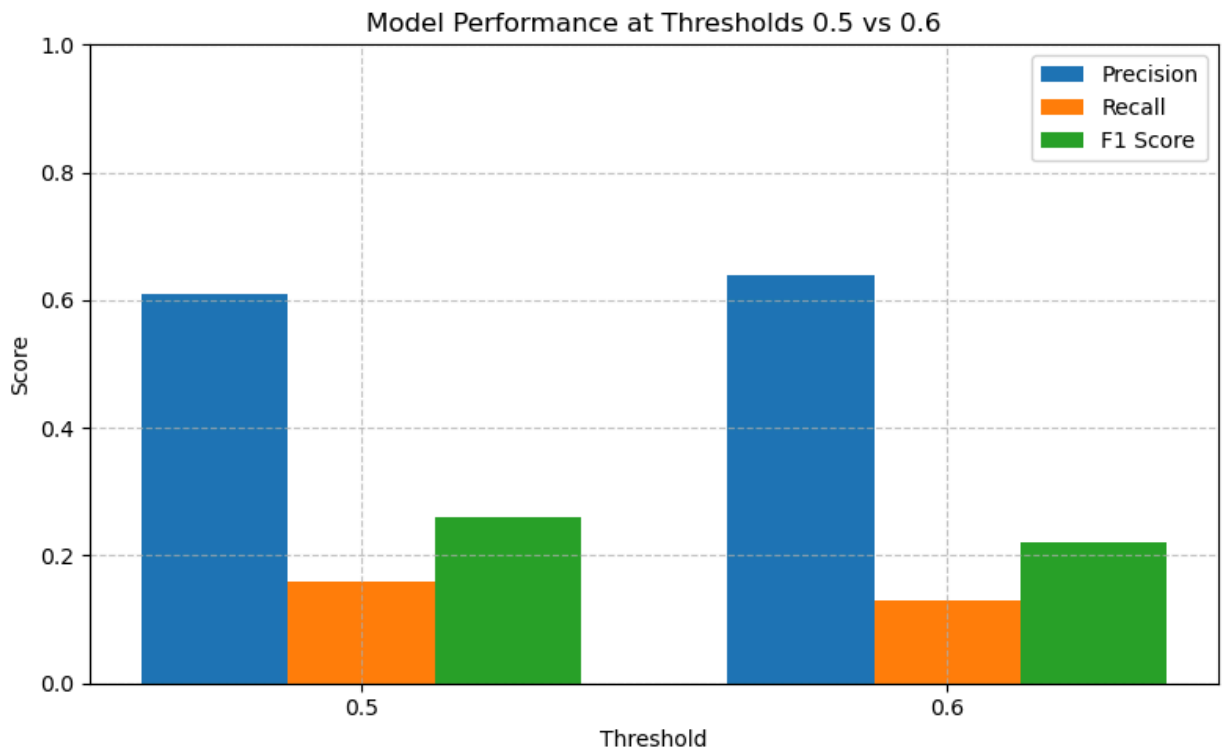
Positive Predictions at 0.6: 14

```
In [25]: # Performance metrics for thresholds 0.5 and 0.6
thresholds = ['0.5', '0.6']
precision = [0.61, 0.64]
recall = [0.16, 0.13]
f1 = [0.26, 0.22]

# Bar plot settings
x = np.arange(len(thresholds))
width = 0.25

plt.figure(figsize=(8, 5))
plt.bar(x - width, precision, width, label='Precision')
plt.bar(x, recall, width, label='Recall')
plt.bar(x + width, f1, width, label='F1 Score')

plt.xlabel('Threshold')
plt.ylabel('Score')
plt.title('Model Performance at Thresholds 0.5 vs 0.6')
plt.xticks(x, thresholds)
plt.ylim(0, 1)
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Threshold Comparison: 0.5 vs. 0.6

At a 0.5 threshold, the model predicts more likely subscribers (18 out of 448) and captures more actual ones, with a recall of 16% and precision of 61%. This means it's better at identifying potential subscribers, even if a few false positives occur.

When increased to a 0.6 threshold, the model becomes stricter—predicting only 14 likely subscribers. Precision improves slightly to 64%, but recall drops to 13%, meaning more real subscribers are missed.

What changes:

- Fewer customers are flagged as likely to subscribe
- Recall decreases (more actual subscribers go undetected)
- Slight improvement in precision (fewer false positives)

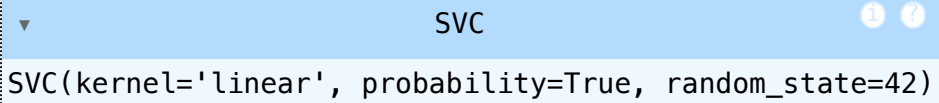
Recommendation: Stick with the 0.5 threshold if the goal is to grow subscriptions. The higher recall helps capture more potential subscribers, even if it means accepting a few false alarms.

Part 4: Model Comparison

```
In [26]: # Import required libraries
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
In [28]: # Train the SVM model
svm_model = SVC(kernel='linear', probability=True, random_state=42)
svm_model.fit(X_train, y_train)
```

Out[28]:



```
SVC(kernel='linear', probability=True, random_state=42)
```

```
In [29]: # Make predictions with SVM
svm_pred = svm_model.predict(X_test)
```

```
In [30]: # Evaluate SVM
svm_accuracy = accuracy_score(y_test, svm_pred)
svm_precision = precision_score(y_test, svm_pred, zero_division=0)
svm_recall = recall_score(y_test, svm_pred)
svm_f1 = f1_score(y_test, svm_pred)
```

```
In [31]: # Evaluate Logistic Regression again for comparison
log_pred = model.predict(X_test)
log_accuracy = accuracy_score(y_test, log_pred)
log_precision = precision_score(y_test, log_pred, zero_division=0)
log_recall = recall_score(y_test, log_pred)
log_f1 = f1_score(y_test, log_pred)
```

```
In [32]: # Create comparison DataFrame
comparison_df = pd.DataFrame({
    'Metric': ['Accuracy', 'Precision', 'Recall', 'F1 Score'],
    'Logistic Regression': [log_accuracy, log_precision, log_recall, log_f1],
    'SVM': [svm_accuracy, svm_precision, svm_recall, svm_f1]
})
```

```
In [34]: # Display the performance comparison in your notebook
print("Logistic Regression vs SVM Performance Comparison:\n")
display(comparison_df)
```

Logistic Regression vs SVM Performance Comparison:

	Metric	Logistic Regression	SVM
0	Accuracy	0.859375	0.859375
1	Precision	0.611111	0.600000
2	Recall	0.164179	0.179104
3	F1 Score	0.258824	0.275862

Model Comparison: Logistic Regression vs. SVM

- Both models perform similarly in terms of accuracy (85.9%), but there are small differences in how they handle subscription predictions:
- SVM has slightly better recall (17.9%) and F1 score (27.6%), meaning it captures more actual subscribers and maintains a better balance between precision and recall.
- Logistic Regression has a slightly higher precision (61%), but its recall is lower (16.4%), so it misses more potential subscribers.

Final Recommendation

Based on the analysis, customer engagement through digital channels—such as website visits and catalog interactions positively influences subscription likelihood, while heavy reliance on physical store purchases reduces it. The logistic regression model provides clear interpretability, while the SVM model offers slightly better recall and F1 score, capturing more actual subscribers.

We recommend the following actions:

- **Focus on digital engagement:** Target frequent website visitors and catalog buyers with personalized subscription offers, reminders, and bundle deals.
- **Encourage store shoppers to subscribe:** Offer in-store customers digital benefits or incentives to promote online engagement and increase conversions.
- **Use the model at a 0.5 threshold:** This captures more true subscribers and helps expand the subscriber base, even if it includes a few false positives.
- **Adopt the SVM model for deployment:** It performs slightly better in identifying subscribers and balances recall and precision more effectively.

Conclusion

This project set out to understand why digital magazine subscriptions have dropped and what customer behavior might be driving it. We took a close look at the data—cleaned it up, created useful features, and explored key patterns behind who subscribes and who doesn't.

We found that customers who are more active online, like those visiting the website or ordering from catalogs, are more likely to subscribe. On the other hand, people who mainly shop in physical stores are less likely to sign up. To predict subscriptions, we tested both logistic regression and SVM models. While both performed similarly, the SVM had a slight edge when it came to balancing accuracy and identifying real subscribers.

We also tested different thresholds and found that keeping it at 0.5 is the most effective—it helps catch more potential subscribers without being too strict.

Overall, this analysis gives the company useful insights and a working model to better understand their customers and take smarter steps to grow subscriptions.

References

- Umesh R. Hodeghatta & Umesha Nayak. (2021). Business Analytics Using Python and R (2nd ed.). Wiley.
- Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd ed.). O'Reilly Media.
- Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830.
- McKinney, W. (2022). Python for Data Analysis (3rd ed.). O'Reilly Media.
- Microsoft Docs. (n.d.). Confusion Matrix Explained
- GitHub. (n.d.). Code references and dataset inspirations. <https://github.com>
- OpenAI. (2023). ChatGPT [Large language model]. Available at: <https://chat.openai.com>