Python - Marketing Analytics

February 5, 2025

1 Retail Marketing Analytics

By Hiba Farhan

1.0.1 Import Packages

[30]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from causalimpact import CausalImpact
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from imblearn.over_sampling import SMOTE
from sklearn.cluster import KMeans
from scipy import stats
from sklearn.ensemble import RandomForestClassifier
import statsmodels.api as sm
```

1.1 Load Data Set

[2]:
```
campaigns_df = pd.read_csv('marketing_campaigns_data.csv')
sales_df = pd.read_csv('marketing_sales_data.csv')
```

1.2 Understand Data

[3]:
```
campaigns_df

# Marketing Campaigns Dataset (500 rows, 11 columns)
# Contains information about marketing campaigns
```

[3]:
```
     campaign_id ad_platform  impressions  clicks  conversions     spend  \
0           1361   Instagram       281966   25262          366   6177.57
1           1073    Facebook       316468   11546          684  32400.49
2           1374  Google Ads       388467   31963         1975  11585.87
3           1155    LinkedIn       321255   29269         2106  31359.81
```

```
4           1104    LinkedIn       481326   23650         3194  32859.85
..           ...         ...          ...     ...          ...       ...
495         1106    Snapchat       107492    3541         3774  48863.22
496         1270    Snapchat       234074     605          592   6425.31
497         1348  Google Ads       306010    9578          895  21704.87
498         1435   Instagram       401743   25503         3653   3059.21
499         1102   Instagram       272320    6590         3086  37255.30

    target_audience campaign_objective device_type  start_date    end_date
0       Retargeting   Sales Conversion      Tablet  2024-10-09  2024-12-31
1     New Customers        Video Views      Mobile  2024-08-16  2024-12-29
2       Millennials   Sales Conversion      Tablet  2024-08-14  2025-01-16
3       Millennials   Sales Conversion     Desktop  2024-09-26  2025-01-01
4       Millennials   Sales Conversion      Mobile  2024-08-22  2025-01-21
..              ...                ...         ...         ...         ...
495           Gen Z    Brand Awareness      Mobile  2024-09-13  2024-11-24
496     Retargeting   Sales Conversion     Desktop  2024-10-02  2025-01-19
497     Retargeting    Brand Awareness      Tablet  2024-09-20  2024-12-14
498     Retargeting        Retargeting      Mobile  2024-08-11  2025-01-01
499     Millennials        Video Views      Tablet  2024-08-08  2024-11-06

[500 rows x 11 columns]
```

[4]:
```
sales_df

# Marketing Sales Dataset (100,000 rows, 15 columns)
# Contains transactional sales data linked to marketing campaigns
```

[4]:
```
       transaction_id  customer_id  campaign_id purchase_date  sales_amount  \
0              500000        29961         1140    2024-09-29        421.94
1              500001        20002         1159    2024-11-09         32.99
2              500002        28397         1182    2025-01-23        250.22
3              500003        25951         1361    2024-10-28         21.10
4              500004        25626         1026    2024-12-09        372.30
...               ...          ...          ...           ...           ...
99995          599995        22298         1375    2024-08-15        428.83
99996          599996        26044         1229    2024-08-15         87.10
99997          599997        23131         1251    2024-10-25         76.04
99998          599998        27236         1456    2025-01-23         41.27
99999          599999        27620         1405    2024-10-28        138.62

       product_id         category           product_name purchase_channel  \
0            9026       Automotive         Window Brother         In-Store
1            9038           Sports            Dog Several           Online
2            9071          Fashion            Letter Sell           Online
3            9000  Health & Beauty              Clear Sit           Online
4            9013           Sports           Toward Begin         In-Store
```

```
…          …            …                      …          …
99995      9058    Automotive  International Account    In-Store
99996      9071       Fashion               Free Kid    In-Store
99997      9014          Home             Firm Heavy    In-Store
99998      9049       Fashion               Word Win    In-Store
99999      9053        Sports     Government Perform      Online

       discount_applied repeat_customer  time_since_last_purchase (days)  \
0                    10             Yes                               91
1                    10             Yes                               75
2                     0              No                              136
3                     0             Yes                               53
4                    20              No                               89
…                   …             …                              …
99995                15             Yes                                0
99996                 5              No                               80
99997                 5             Yes                                3
99998                 0              No                              113
99999                15             Yes                               67

       basket_size payment_method   location
0                4      Gift Card    Calgary
1                2    Credit Card  Vancouver
2                2      Gift Card  Vancouver
3                3     Debit Card  Vancouver
4                1     Debit Card    Toronto
…              …            …        …
99995            3           Cash    Toronto
99996            3    Credit Card  Vancouver
99997            3           Cash     Ottawa
99998            2      Gift Card    Toronto
99999            4      Gift Card    Calgary

[100000 rows x 15 columns]
```

1.3 Clean up data

[5]:
```python
# Convert dates
campaigns_df["start_date"] = pd.to_datetime(campaigns_df["start_date"])
campaigns_df["end_date"] = pd.to_datetime(campaigns_df["end_date"])
sales_df["purchase_date"] = pd.to_datetime(sales_df["purchase_date"])

# Check Data Types
campaigns_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 11 columns):
```

```
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   campaign_id         500 non-null    int64
 1   ad_platform         500 non-null    object
 2   impressions         500 non-null    int64
 3   clicks              500 non-null    int64
 4   conversions         500 non-null    int64
 5   spend               500 non-null    float64
 6   target_audience     500 non-null    object
 7   campaign_objective  500 non-null    object
 8   device_type         500 non-null    object
 9   start_date          500 non-null    datetime64[ns]
 10  end_date            500 non-null    datetime64[ns]
dtypes: datetime64[ns](2), float64(1), int64(4), object(4)
memory usage: 43.1+ KB
```

[6]:
```
# Check Data Types
sales_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 15 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   transaction_id                  100000 non-null  int64
 1   customer_id                     100000 non-null  int64
 2   campaign_id                     100000 non-null  int64
 3   purchase_date                   100000 non-null  datetime64[ns]
 4   sales_amount                    100000 non-null  float64
 5   product_id                      100000 non-null  int64
 6   category                        100000 non-null  object
 7   product_name                    100000 non-null  object
 8   purchase_channel                100000 non-null  object
 9   discount_applied                100000 non-null  int64
 10  repeat_customer                 100000 non-null  object
 11  time_since_last_purchase (days) 100000 non-null  int64
 12  basket_size                     100000 non-null  int64
 13  payment_method                  100000 non-null  object
 14  location                        100000 non-null  object
dtypes: datetime64[ns](1), float64(1), int64(7), object(6)
memory usage: 11.4+ MB
```

[7]:
```
# Check missing values
print("\nMissing values in Campaigns:\n", campaigns_df.isnull().sum())
print("\nMissing values in Sales:\n", sales_df.isnull().sum())
```

```

Missing values in Campaigns:
 campaign_id           0
```

```
ad_platform          0
impressions          0
clicks               0
conversions          0
spend                0
target_audience      0
campaign_objective   0
device_type          0
start_date           0
end_date             0
dtype: int64

Missing values in Sales:
 transaction_id                    0
customer_id                       0
campaign_id                       0
purchase_date                     0
sales_amount                      0
product_id                        0
category                          0
product_name                      0
purchase_channel                  0
discount_applied                  0
repeat_customer                   0
time_since_last_purchase (days)   0
basket_size                       0
payment_method                    0
location                          0
dtype: int64
```

[8]:
```python
# Check duplicated values
print("\nDuplicated values in Campaigns:\n", campaigns_df.duplicated().sum())
print("\nDuplicated values in Sales:\n", sales_df.duplicated().sum())
```

```
Duplicated values in Campaigns:
 0

Duplicated values in Sales:
 0
```

[9]:
```python
# Display dataset shapes
print("\nCampaigns Data Shape:", campaigns_df.shape)
print("\nSales Data Shape:", sales_df.shape)
```

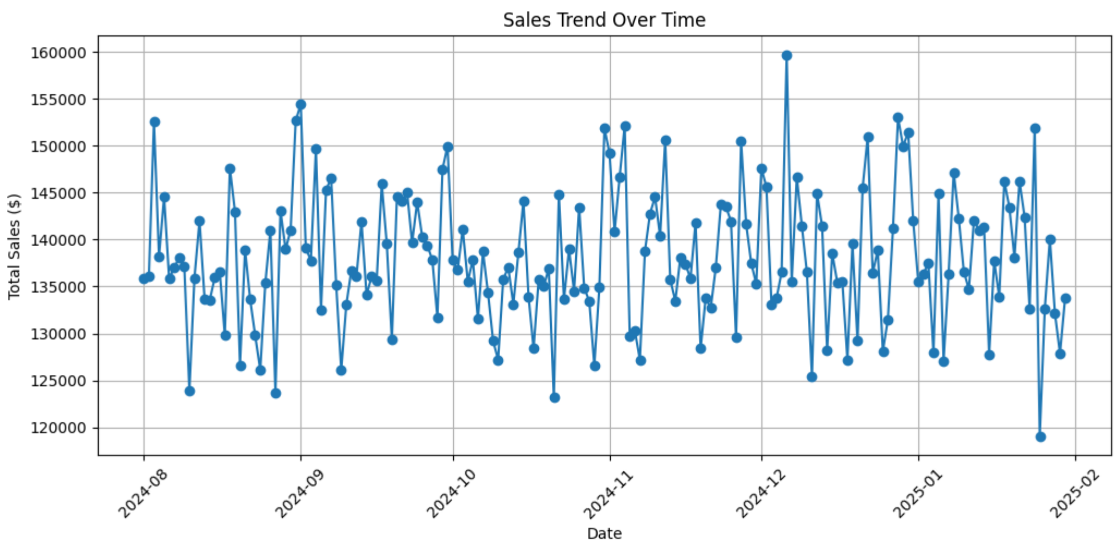
```
Campaigns Data Shape: (500, 11)

Sales Data Shape: (100000, 15)
```

1.4 Exploratory Data Analysis (EDA)

Sales Trends Over Time

Insights from Visual: - Seasonal Trends: December 2024 shows a significant sales peak, likely due to holiday shopping, while early 2025 exhibits a post-holiday dip. - Sales Variability: Daily sales fluctuate, with some spikes likely driven by specific campaigns or promotions. - Actionable Insight: Focus on replicating successful December strategies and addressing dips during slower periods like early 2025.

[10]:
```
# Plot sales trend over time
sales_trend = sales_df.groupby("purchase_date")["sales_amount"].sum()
plt.figure(figsize=(12, 5))
plt.plot(sales_trend.index, sales_trend.values, marker='o', linestyle='-')
plt.xlabel("Date")
plt.ylabel("Total Sales ($)")
plt.title("Sales Trend Over Time")
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



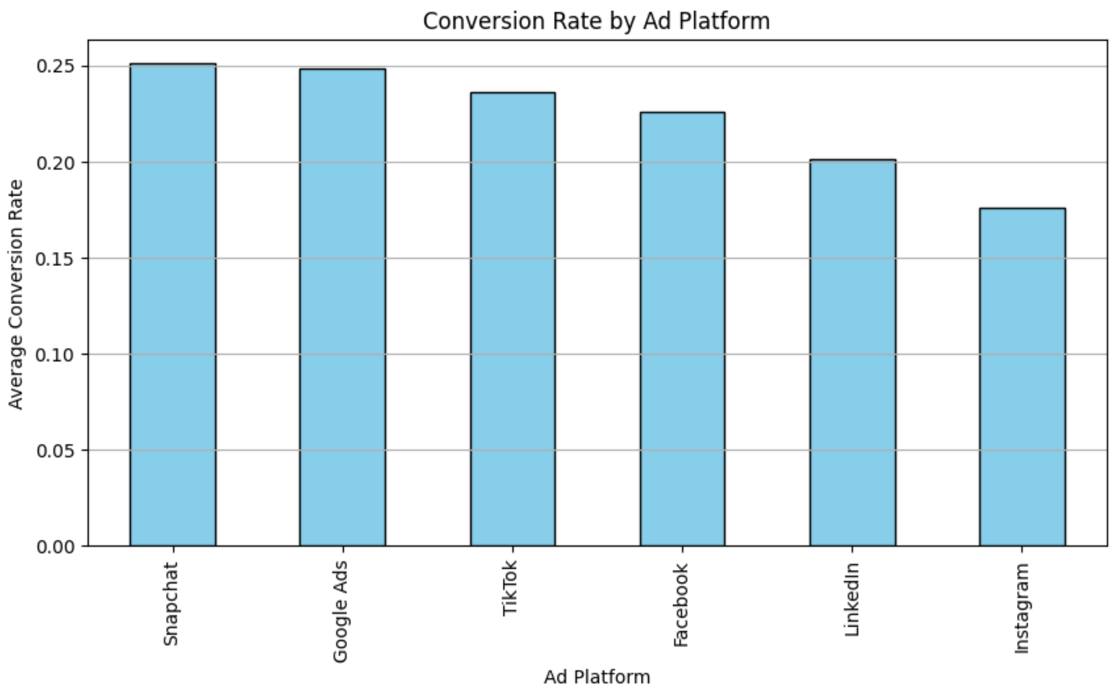
Conversion rate by campaign

Insights from visual:

- Top Platforms: Snapchat and Google Ads have the highest average conversion rates (~25%), indicating strong performance for driving conversions.
- Low Performers: Instagram has the lowest average conversion rate, suggesting less effectiveness compared to - other platforms.
- Actionable Insight: Allocate more budget and focus to high-performing platforms like

Snapchat and Google Ads while reevaluating strategies for lower-performing platforms like Instagram.

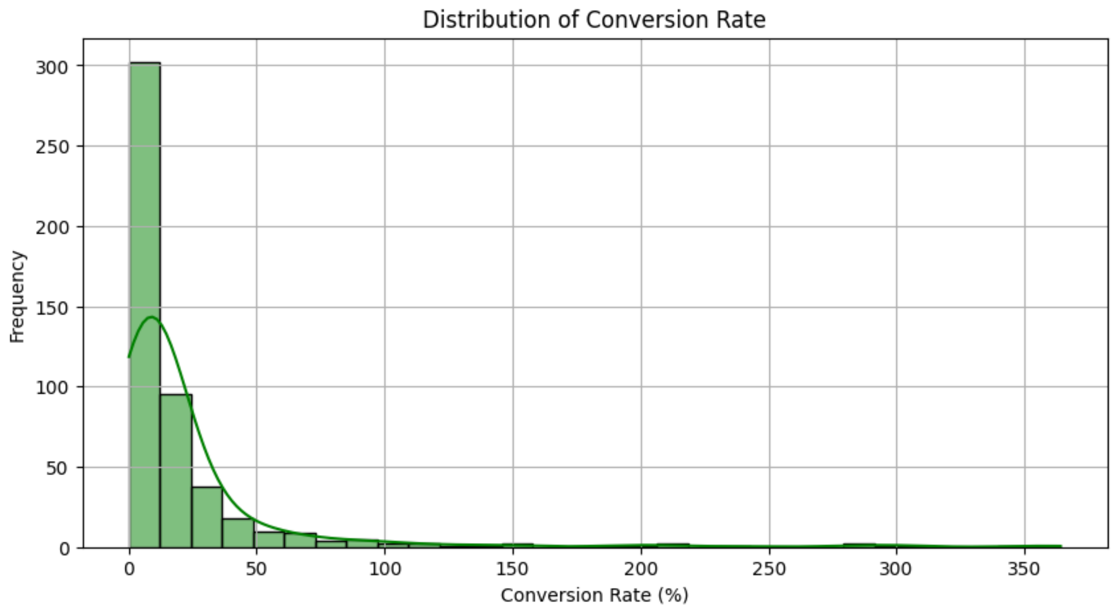
[11]:
```python
campaigns_df["conversion_rate"] = campaigns_df["conversions"] / campaigns_df["clicks"]
conversion_by_platform = campaigns_df.groupby("ad_platform")["conversion_rate"].mean().sort_values(ascending=False)

# Plot conversion rates
plt.figure(figsize=(10, 5))
conversion_by_platform.plot(kind="bar", color="skyblue", edgecolor="black")
plt.xlabel("Ad Platform")
plt.ylabel("Average Conversion Rate")
plt.title("Conversion Rate by Ad Platform")
plt.grid(axis="y")
plt.show()
```



Conversion Rate Distribution

Insights from Visual: - Highly Skewed Distribution: The majority of campaigns have conversion rates clustered at the lower end (near 0–20%), with very few campaigns achieving higher conversion rates. - Outliers: Some campaigns show exceptionally high conversion rates (above 100%), indicating either niche target success or possible data anomalies that need verification. - Optimization Opportunity: Focus on analyzing and replicating the strategies behind campaigns with higher conversion rates while identifying and addressing factors leading to poor performance in low-conversion campaigns.

[12]:
```
campaigns_df["conversion_rate"] = (campaigns_df["conversions"] /
  campaigns_df["clicks"]) * 100

# Conversion Rate Distribution
plt.figure(figsize=(10, 5))
sns.histplot(campaigns_df["conversion_rate"], bins=30, kde=True, color="green")
plt.xlabel("Conversion Rate (%)")
plt.ylabel("Frequency")
plt.title("Distribution of Conversion Rate")
plt.grid(True)
plt.show()
```



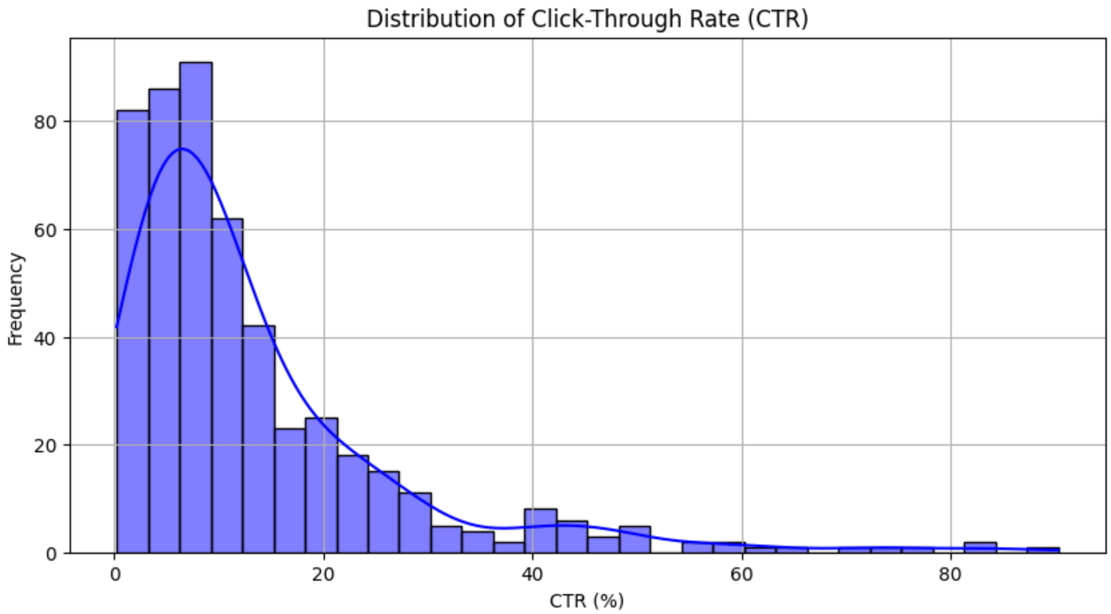
Click-Through Rate (CTR) Distribution

Insights from Visual:

- Skewed Distribution: Most campaigns have a CTR concentrated between 0–20%, indicating typical performance, with a long tail of higher CTRs.
- High CTR Outliers: A few campaigns have CTRs exceeding 40%, suggesting exceptionally engaging content or highly targeted campaigns.
- Optimization Opportunity: Investigate campaigns with high CTR to identify successful strategies (e.g., ad content, targeting) and apply those insights to underperforming campaigns concentrated at lower CTRs.

[13]:
```
campaigns_df["CTR"] = (campaigns_df["clicks"] / campaigns_df["impressions"]) *
  100

# Click-Through Rate (CTR) Distribution
```

```
plt.figure(figsize=(10, 5))
sns.histplot(campaigns_df["CTR"], bins=30, kde=True, color="blue")
plt.xlabel("CTR (%)")
plt.ylabel("Frequency")
plt.title("Distribution of Click-Through Rate (CTR)")
plt.grid(True)
plt.show()
```

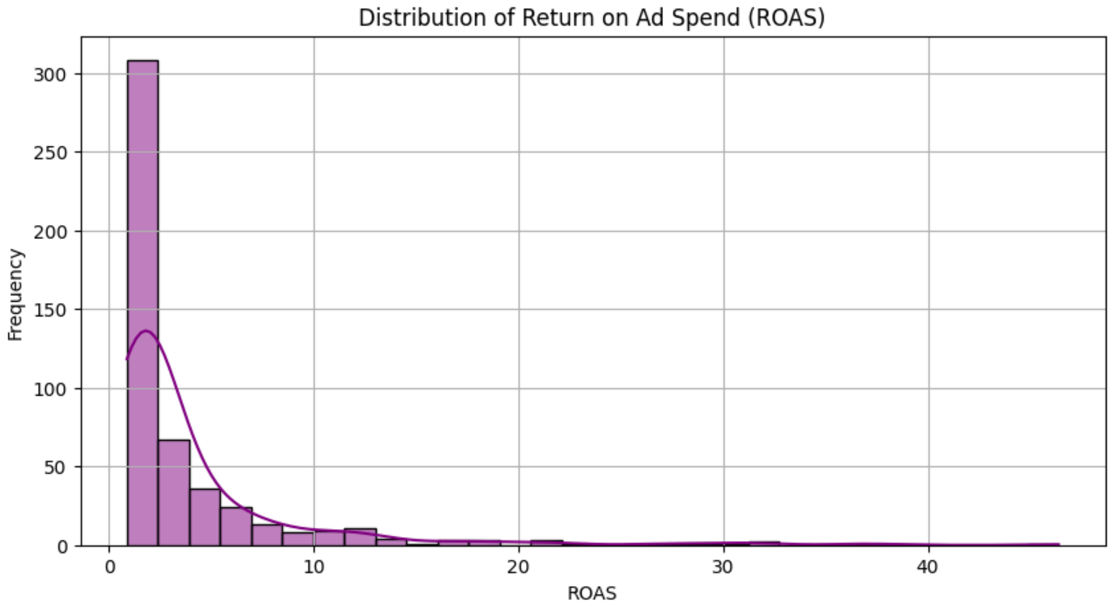


ROAS Distribution

Insights from Visual: - Majority of Campaigns: Most campaigns have a ROAS clustered between 0 and 5, indicating moderate returns on ad spend. - High ROAS Outliers: A few campaigns achieve significantly higher ROAS values (greater than 10), indicating exceptional performance and efficient ad spending. - Actionable Insight: Investigate high-performing campaigns to identify successful strategies and replicate them, while reassessing and optimizing low-performing campaigns with ROAS close to or below 1 to ensure profitability.

[16]:
```
# ROAS Calculation
campaign_sales = sales_df.groupby("campaign_id")["sales_amount"].sum()
campaign_spend = campaigns_df.set_index("campaign_id")["spend"]
roas = campaign_sales / campaign_spend
campaigns_df["ROAS"] = campaigns_df["campaign_id"].map(roas)


# ROAS Distribution
plt.figure(figsize=(10, 5))
sns.histplot(campaigns_df["ROAS"], bins=30, kde=True, color="purple")
```

```
plt.xlabel("ROAS")
plt.ylabel("Frequency")
plt.title("Distribution of Return on Ad Spend (ROAS)")
plt.grid(True)
plt.show()
```



High Value Customers

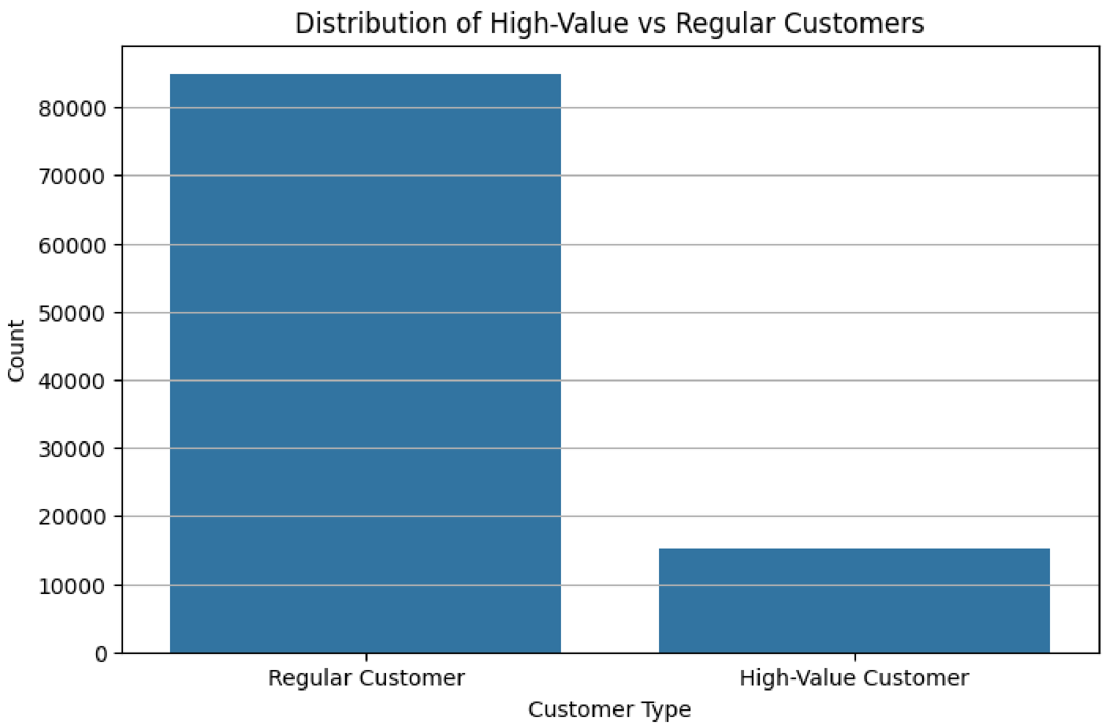
[17]:
```
# Feature Engineering: Customer Lifetime Value (CLV) and ROAS
customer_spending = sales_df.groupby("customer_id")["sales_amount"].sum()
customer_transactions = sales_df.groupby("customer_id")["transaction_id"].
 ↪count()
sales_df["customer_lifetime_value"] = sales_df["customer_id"].
 ↪map(customer_spending / customer_transactions)


# Identify high-value customers (Top 10% spenders)
threshold = customer_spending.quantile(0.90)
high_value_customers = customer_spending[customer_spending >= threshold].index
sales_df["high_value_customer"] = sales_df["customer_id"].apply(lambda x: 1 if␣
 ↪x in high_value_customers else 0)
```

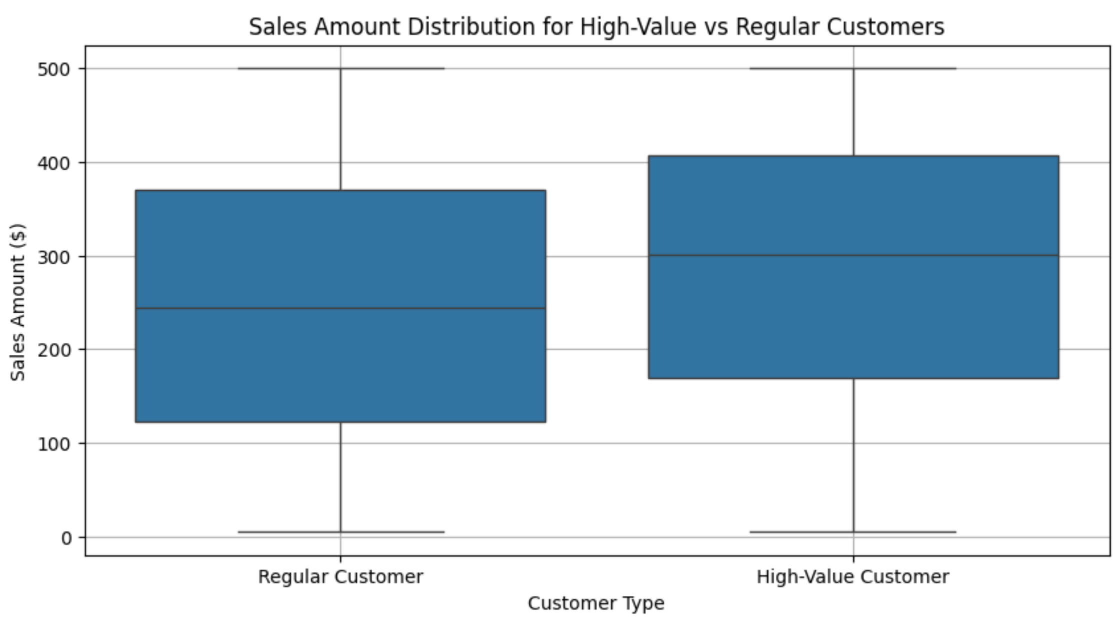
Visualization of High-Value Customers

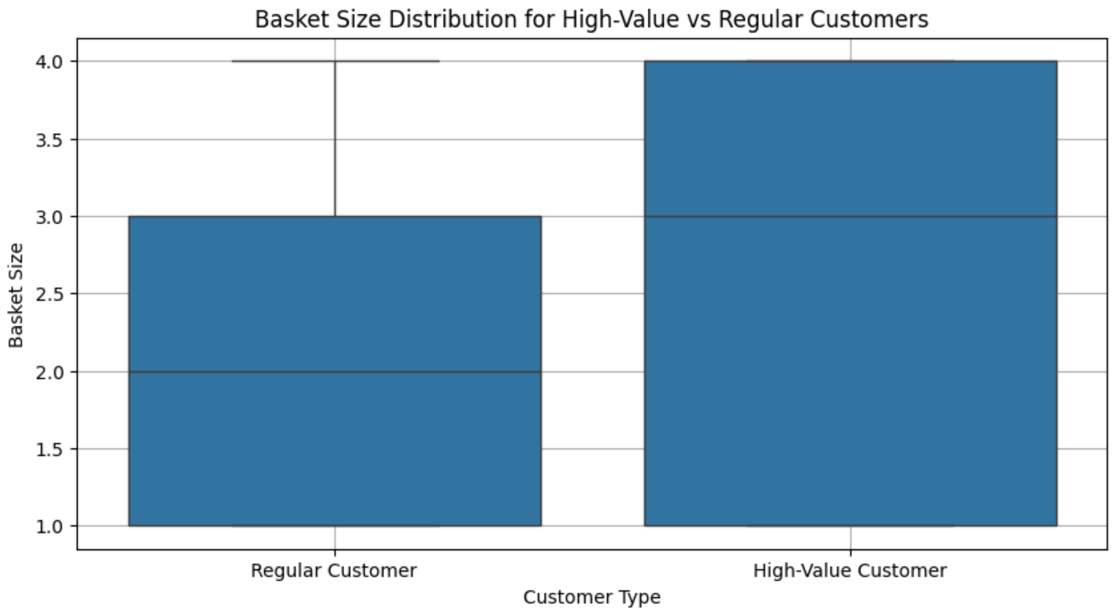
- Customer Distribution
  - The majority of customers are regular customers, with high-value customers forming a smaller portion (~10%).
- Sales Amount Distribution

  - High-value customers tend to have significantly higher transaction amounts.
  - There is a wider range of sales amounts among high-value customers.
- Basket Size Comparison
  - High-value customers generally purchase more items per transaction.
  - Some high-value customers have exceptionally large basket sizes.

[18]:
```python
# Visualizing High-Value Customers

# Count plot of High-Value vs. Regular Customers
plt.figure(figsize=(8, 5))
sns.countplot(x=sales_df["high_value_customer"])
plt.xticks(ticks=[0, 1], labels=["Regular Customer", "High-Value Customer"])
plt.xlabel("Customer Type")
plt.ylabel("Count")
plt.title("Distribution of High-Value vs Regular Customers")
plt.grid(axis="y")
plt.show()

# Sales Distribution for High-Value vs Regular Customers
plt.figure(figsize=(10, 5))
sns.boxplot(x=sales_df["high_value_customer"], y=sales_df["sales_amount"])
plt.xticks(ticks=[0, 1], labels=["Regular Customer", "High-Value Customer"])
plt.xlabel("Customer Type")
plt.ylabel("Sales Amount ($)")
plt.title("Sales Amount Distribution for High-Value vs Regular Customers")
plt.grid(True)
plt.show()

# Basket Size Distribution for High-Value vs Regular Customers
plt.figure(figsize=(10, 5))
sns.boxplot(x=sales_df["high_value_customer"], y=sales_df["basket_size"])
plt.xticks(ticks=[0, 1], labels=["Regular Customer", "High-Value Customer"])
plt.xlabel("Customer Type")
plt.ylabel("Basket Size")
plt.title("Basket Size Distribution for High-Value vs Regular Customers")
plt.grid(True)
plt.show()
```


Distribution of High-Value vs Regular Customers
Count
80000
70000
60000
50000
40000
30000
20000
10000
0
Regular Customer
High-Value Customer
Customer Type


Sales Amount Distribution for High-Value vs Regular Customers
Sales Amount ($)
500
400
300
200
100
0
Regular Customer
High-Value Customer
Customer Type



1.5 Build a predictive model for high-value customers

Uses Logistic Regression with SMOTE to predict high-value customers

Balances classes for better model performance

[19]:
```
# Predictive Modeling: Classifying High-Value Customers
features = ["sales_amount", "discount_applied", "basket_size",␣
↪"time_since_last_purchase (days)"]
X = sales_df[features]
y = sales_df["high_value_customer"]

# Standardization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Handle Class Imbalance with SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
↪test_size=0.2, random_state=42, stratify=y_resampled)

# Train Logistic Regression Model with Balanced Class Weights
model = LogisticRegression(class_weight="balanced")
model.fit(X_train, y_train)
```

```
# Predictions
y_pred = model.predict(X_test)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
print("Classification Report:\n", report)
```

```
Model Accuracy: 0.56
Classification Report:
               precision    recall  f1-score   support

           0       0.56      0.54      0.55     16975
           1       0.56      0.57      0.56     16975

    accuracy                           0.56     33950
   macro avg       0.56      0.56      0.56     33950
weighted avg       0.56      0.56      0.56     33950
```

Random Forest Model

Insights from both models:

- Random Forest Outperforms: The Random Forest model achieves 78% accuracy, significantly higher than Logistic Regression's 56%, with better precision, recall, and F1-scores.
- Strength of Random Forest: It effectively handles non-linear relationships and class balancing, making it more reliable for predicting high-value customers.
- Recommendation: Use the Random Forest model and further optimize it with hyperparameter tuning for improved performance.

[31]:
```
# Predictive Modeling: Random Forest Classifier for High-Value Customers
features = ["sales_amount", "discount_applied", "basket_size",␣
 ↪"time_since_last_purchase (days)"]
X = sales_df[features]
y = sales_df["high_value_customer"]

# Standardization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Handle Class Imbalance with SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

# Train-Test Split
```

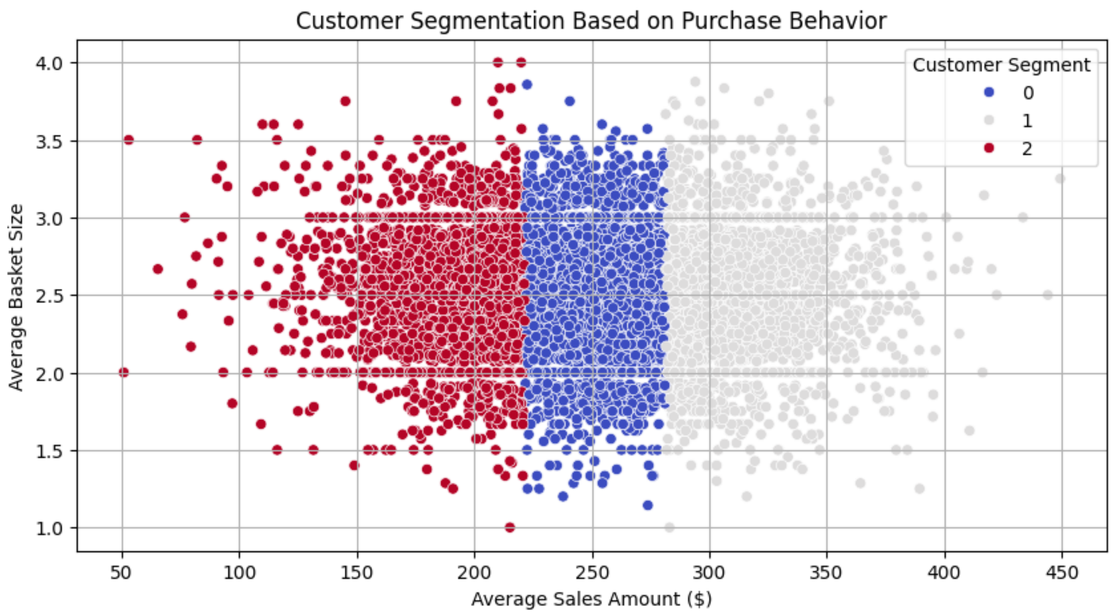
```python
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
↪test_size=0.2, random_state=42, stratify=y_resampled)

# Train Random Forest Model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predictions
y_pred = rf_model.predict(X_test)

# Model Evaluation
print(f"Random Forest Model Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Random Forest Model Accuracy: 0.78
Classification Report:
               precision    recall  f1-score   support

           0       0.78      0.77      0.77     18002
           1       0.77      0.79      0.78     18002

    accuracy                           0.78     36004
   macro avg       0.78      0.78      0.78     36004
weighted avg       0.78      0.78      0.78     36004
```

Customer Segmentation (K-Means Clustering) - Customers are grouped into three segments based on sales amount and basket size. - This can help in personalized marketing and pricing strategies.

[22]:
```python
# Customer Segmentation based on Sales Amount and Basket Size (K-Means␣
↪Clustering)



# Selecting features for clustering
customer_features = sales_df.groupby("customer_id")[["sales_amount",␣
↪"basket_size"]].mean()



# Apply K-Means Clustering (3 segments: Low, Medium, High Value)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
customer_features["Segment"] = kmeans.fit_predict(customer_features)

# Scatter plot of Customer Segments
plt.figure(figsize=(10, 5))
sns.scatterplot(x=customer_features["sales_amount"],␣
↪y=customer_features["basket_size"], hue=customer_features["Segment"],␣
↪palette="coolwarm")
```

```
plt.xlabel("Average Sales Amount ($)")
plt.ylabel("Average Basket Size")
plt.title("Customer Segmentation Based on Purchase Behavior")
plt.legend(title="Customer Segment")
plt.grid(True)
plt.show()
```



Identify Repeat Customers

[23]:
```
# Identify Repeat Customers
repeat_customers = sales_df[sales_df["repeat_customer"] == "Yes"].
↪groupby("customer_id").size()
repeat_purchase_rate = (repeat_customers.count() / sales_df["customer_id"].
↪nunique()) * 100

# Average Basket Size & Discount Impact
avg_basket_size = sales_df["basket_size"].mean()
discount_impact = sales_df.groupby("discount_applied")["sales_amount"].mean()

# Display results
print(f"\nRepeat Purchase Rate: {repeat_purchase_rate:.2f}%")
print(f"\nAverage Basket Size: {avg_basket_size:.2f}")
print("\nAverage Sales by Discount Level:\n\n", discount_impact)
```

```
Repeat Purchase Rate: 99.42%
```

```
Average Basket Size: 2.50

Average Sales by Discount Level:

 discount_applied
0     253.614549
5     252.467851
10    253.808559
15    252.756494
20    251.067507
Name: sales_amount, dtype: float64
```

1.6 A/B Testing

Interpretation

- The p-value (0.530) is greater than 0.05, meaning we fail to reject the null hypothesis.
- This suggests that there is no statistically significant difference between the average sales amounts of In-Store and Online purchases.
- The sales averages for both platforms are very close, indicating that customers spend similar amounts regardless of the purchase channel.

[24]:
```python
# Perform A/B Testing on Conversion Rates for Two Major Ad Platforms



# Identify the two most used purchase channels
platform_counts = sales_df["purchase_channel"].value_counts()
if len(platform_counts) >= 2:
    platform_A, platform_B = platform_counts.index[:2]

    # Extract conversion rates for each platform
    conv_rate_A = sales_df[sales_df["purchase_channel"] ==␣
 ↪platform_A]["sales_amount"]
    conv_rate_B = sales_df[sales_df["purchase_channel"] ==␣
 ↪platform_B]["sales_amount"]

    # Perform independent t-test
    t_stat, p_value = stats.ttest_ind(conv_rate_A, conv_rate_B, equal_var=False)

    # Display results
    ab_test_results = {
        "Platform A": platform_A,
        "Platform B": platform_B,
        "Mean Conversion Rate A": conv_rate_A.mean(),
        "Mean Conversion Rate B": conv_rate_B.mean(),
        "T-Statistic": t_stat,
        "P-Value": p_value
```

```
    }
else:
    ab_test_results = "Not enough platforms available for A/B testing."

ab_test_results
```

[24]:
```
{'Platform A': 'In-Store',
 'Platform B': 'Online',
 'Mean Conversion Rate A': 253.02789069683763,
 'Mean Conversion Rate B': 252.46058250290147,
 'T-Statistic': 0.6273956328123292,
 'P-Value': 0.5304013622493294}
```

[ ]:
```

```