Credit_Card_Acceptance

August 8, 2024

1 Credit Card Offer Acceptance

1.1 Introduction

1.1.1 Project Overview

This project involves analyzing credit card offer acceptance data to predict customer likelihood of signing up for a credit card. By leveraging data on customer characteristics, the goal is to enhance targeted marketing strategies and identify the most compelling offers for different customer segments.

1.1.2 Background

The dataset used in this project comprises detailed information on customers and credit card offers. It includes data on customer demographics, such as income and credit rating, as well as specifics of the credit card offers, including reward types. This dataset is instrumental in understanding the factors that influence the acceptance rate of credit card offers. By analyzing these variables, we aim to uncover patterns and insights that drive customer decisions, ultimately improving the effectiveness of targeted marketing strategies.

1.1.3 Objectives

- **Customer Segmentation:** Classify customer profiles into distinct segments based on demographic and behavioral characteristics to better understand different customer groups.
- **Impact Analysis:** Identify and evaluate the key factors that significantly influence a customer's decision to accept or decline a credit card offer.
- **Predictive Modeling:** Develop a predictive model to forecast the likelihood of a customer accepting a credit card offer based on their individual characteristics.

1.1.4 Data Description

Dataset Overview The dataset used in this project is sourced from Data world's Credit Card Dataset. It includes 18,000 customer records and 18 features. #### Key Features - Customer Number: Unique identifier for each customer. - Offer Accepted: Indicator of whether the customer accepted the credit card offer. (Boolean) - Reward: Type of reward associated with the credit card. - Mailer Type: Method used to deliver the credit card offer. - Income Level: The customer's income level. - Number of Bank Accounts Open: Count of bank accounts held by the customer. - Overdraft Protection: Indicator of whether the customer has overdraft protection on their accounts.

(Boolean) - Credit Rating: Rating reflecting the customer's creditworthiness, based on payment history and ability to repay debt. - Number of Credit Cards Held: Number of credit cards currently held by the customer. - Number of Homes Owned: Number of homes owned by the customer. - Household Size: Size of the customer's household. - Own Your Home?: Indicator of whether the customer owns their home. (Boolean) - Average Balance: Average balance across all accounts. - Q1 Balance: Balance of the customer's accounts for the first quarter of the year. - Q2 Balance: Balance of the customer's accounts for the second quarter of the year. - Q3 Balance: Balance of the customer's accounts for the third quarter of the year. - Q4 Balance: Balance of the customer's accounts for the fourth quarter of the year. #### Data Types - Categorical: Offer Accepted, Mailer Type, Income Level - Numerical: Number of Bank Accounts Open, Overdraft Protection, Credit Rating, Number of Credit Cards Held, Number of Homes Owned, Household Size, Own Your Home?, Average Balance, Q1 Balance, Q2 Balance, Q3 Balance, Q4 Balance

1.1.5 Prediction

The goal is to predict whether a customer will accept a credit card offer based on their characteristics. The prediction model will analyze various customer attributes to determine the likelihood of acceptance.

1.1.6 Metric

F1 Score: The F1 Score is the harmonic mean of precision and recall, providing a balanced measure of a model's performance in classification tasks. It is particularly useful when dealing with imbalanced datasets, where achieving a balance between precision (the accuracy of positive predictions) and recall (the ability to identify all relevant instances) is crucial.

The F1 Score is calculated as follows:

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

This metric ensures that both false positives and false negatives are considered, offering a comprehensive assessment of the model's accuracy.

1.1.7 References

1. Dataset Source: Data World Credit Card Dataset

1.2 Exploratory Data Analysis

1.2.1 Understanding the Dataset

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```python
data = pd.read_csv("data.csv")
```

[3]:
```
data.head()
```

[3]:
```
   index  Customer Number Offer Accepted     Reward Mailer Type Income Level  \
0      0                1             No  Air Miles      Letter         High
1      1                2             No  Air Miles      Letter       Medium
2      2                3             No  Air Miles    Postcard         High
3      3                4             No  Air Miles      Letter       Medium
4      4                5             No  Air Miles      Letter       Medium

   # Bank Accounts Open Overdraft Protection Credit Rating  \
0                     1                   No          High
1                     1                   No        Medium
2                     2                   No        Medium
3                     2                   No          High
4                     1                   No        Medium

   # Credit Cards Held  # Homes Owned  Household Size Own Your Home  \
0                    2              1               4            No
1                    2              2               5           Yes
2                    2              1               2           Yes
3                    1              1               4            No
4                    2              1               6           Yes

   Average Balance  Q1 Balance  Q2 Balance  Q3 Balance  Q4 Balance
0          1160.75      1669.0       877.0      1095.0      1002.0
1           147.25        39.0       106.0        78.0       366.0
2           276.50       367.0       352.0       145.0       242.0
3          1219.00      1578.0      1760.0      1119.0       419.0
4          1211.00      2140.0      1357.0       982.0       365.0
```

[4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18000 entries, 0 to 17999
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   index                 18000 non-null  int64
 1   Customer Number       18000 non-null  int64
 2   Offer Accepted        18000 non-null  object
 3   Reward                18000 non-null  object
 4   Mailer Type           18000 non-null  object
 5   Income Level          18000 non-null  object
 6   # Bank Accounts Open  18000 non-null  int64
 7   Overdraft Protection  18000 non-null  object
 8   Credit Rating         18000 non-null  object
 9   # Credit Cards Held   18000 non-null  int64
```

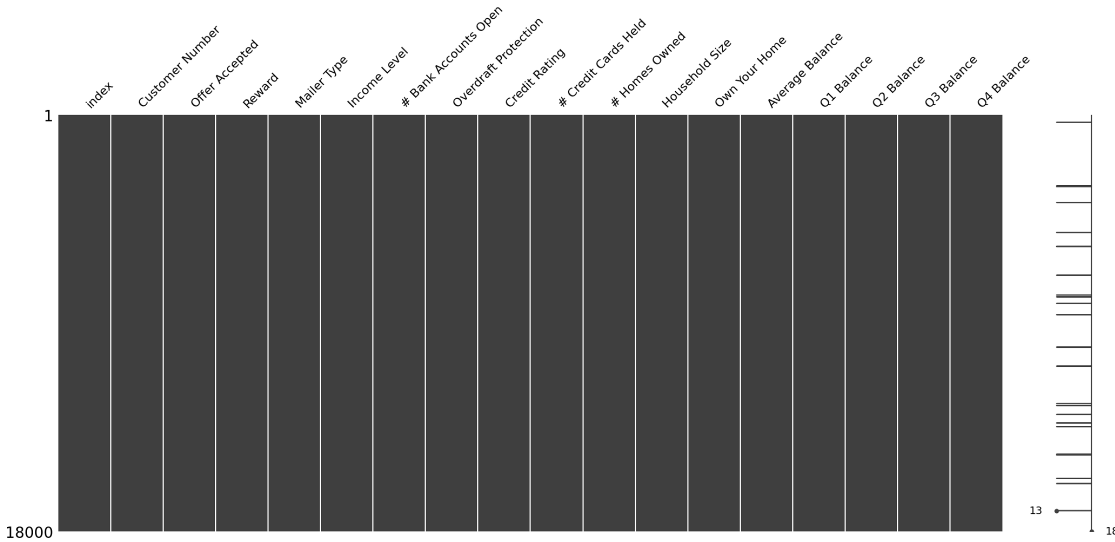
```
 10  # Homes Owned         18000 non-null  int64
 11  Household Size        18000 non-null  int64
 12  Own Your Home         18000 non-null  object
 13  Average Balance       17976 non-null  float64
 14  Q1 Balance            17976 non-null  float64
 15  Q2 Balance            17976 non-null  float64
 16  Q3 Balance            17976 non-null  float64
 17  Q4 Balance            17976 non-null  float64
dtypes: float64(5), int64(6), object(7)
memory usage: 2.5+ MB
```

[5]:
```python
data.describe()
```

[5]:
```
              index  Customer Number  # Bank Accounts Open  \
count  18000.000000     18000.000000          18000.000000
mean    8999.500000      9000.500000              1.255778
std     5196.296758      5196.296758              0.472501
min        0.000000         1.000000              1.000000
25%     4499.750000      4500.750000              1.000000
50%     8999.500000      9000.500000              1.000000
75%    13499.250000     13500.250000              1.000000
max    17999.000000     18000.000000              3.000000

       # Credit Cards Held  # Homes Owned  Household Size  Average Balance  \
count         18000.000000   18000.000000    18000.000000     17976.000000
mean              1.903500       1.203444        3.499056       940.515562
std               0.797009       0.427341        1.114182       350.297837
min               1.000000       1.000000        1.000000        48.250000
25%               1.000000       1.000000        3.000000       787.500000
50%               2.000000       1.000000        3.000000      1007.000000
75%               2.000000       1.000000        4.000000      1153.250000
max               4.000000       3.000000        9.000000      3366.250000

         Q1 Balance    Q2 Balance    Q3 Balance    Q4 Balance
count  17976.000000  17976.000000  17976.000000  17976.000000
mean     910.450656    999.392190   1042.033600    810.185803
std      620.077060    457.402268    553.452599    559.001365
min        0.000000      0.000000      0.000000      0.000000
25%      392.750000    663.000000    633.000000    363.000000
50%      772.000000   1032.000000    945.500000    703.000000
75%     1521.000000   1342.000000   1463.000000   1212.000000
max     3450.000000   3421.000000   3823.000000   4215.000000
```

[6]:
```python
import missingno as msno

msno.matrix(data)
```

[6]: <Axes: >



[7]:
```
# remove null values
data = data.dropna()
```

[8]:
```
# remove index and customer number column
data = data.drop(['index', 'Customer Number'], axis=1)
```

[9]:
```
import sweetviz as sv

# Create a Sweetviz report
report = sv.analyze(data)

# Display the report in a Jupyter notebook
report.show_notebook()
```

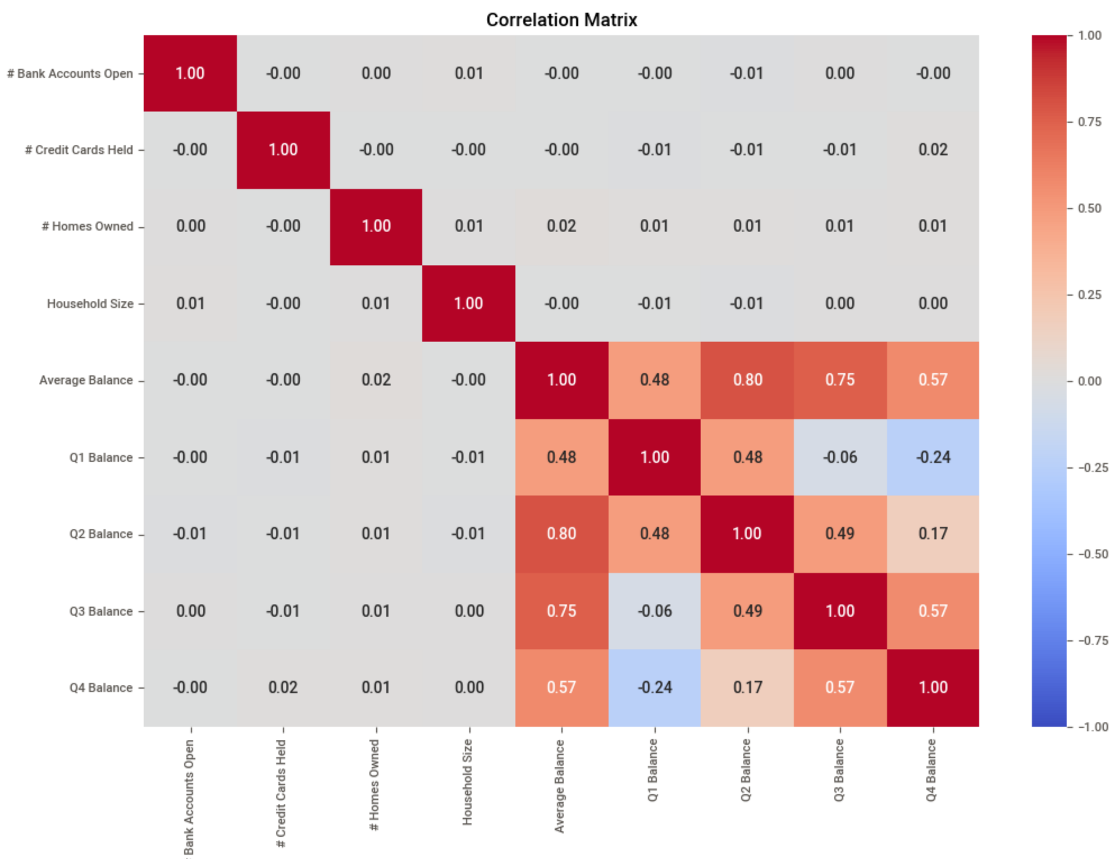
```
/home/hpark/Syncthing/Professional/DS_Projects/Credit_Card_Offer_Acceptance/.ven
v/lib/python3.10/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not
found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
Done! Use 'show' commands to display/save.   |      | [100%]   00:00 ->
(00:00 left)
```

```
<IPython.core.display.HTML object>
```

[10]:
```
numerical_columns = ['# Bank Accounts Open', '# Credit Cards Held', '# Homes␣
↪Owned', 'Household Size', 'Average Balance', 'Q1 Balance', 'Q2 Balance', 'Q3␣
↪Balance', 'Q4 Balance'] # 9
```

```
categorical_columns = ['Offer Accepted', 'Reward', 'Mailer Type', 'Income␣
↪Level', 'Overdraft Protection', 'Credit Rating', 'Own Your Home'] # 7
```

[11]:
```
# Compute the correlation matrix
corr_matrix = data[numerical_columns].corr()

# Plot the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', vmin=-1,␣
↪vmax=1)
plt.title('Correlation Matrix')
plt.show()
```

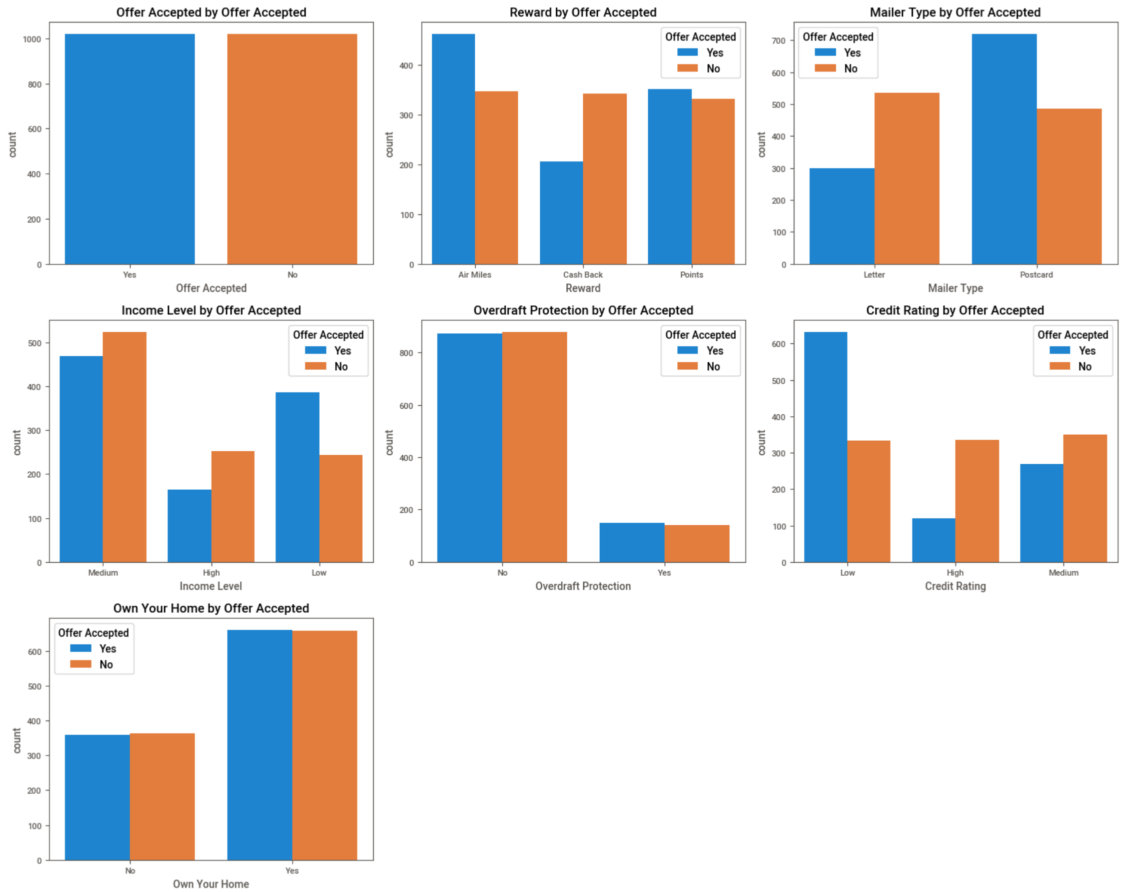


[12]:
```
minority_class = data[data['Offer Accepted'] == 'Yes']
majority_class = data[data['Offer Accepted'] == 'No']

majority_class_sampled = majority_class.sample(n=len(minority_class),␣
↪random_state=42)
```

```
balanced_data = pd.concat([minority_class, majority_class_sampled])
```

[13]:

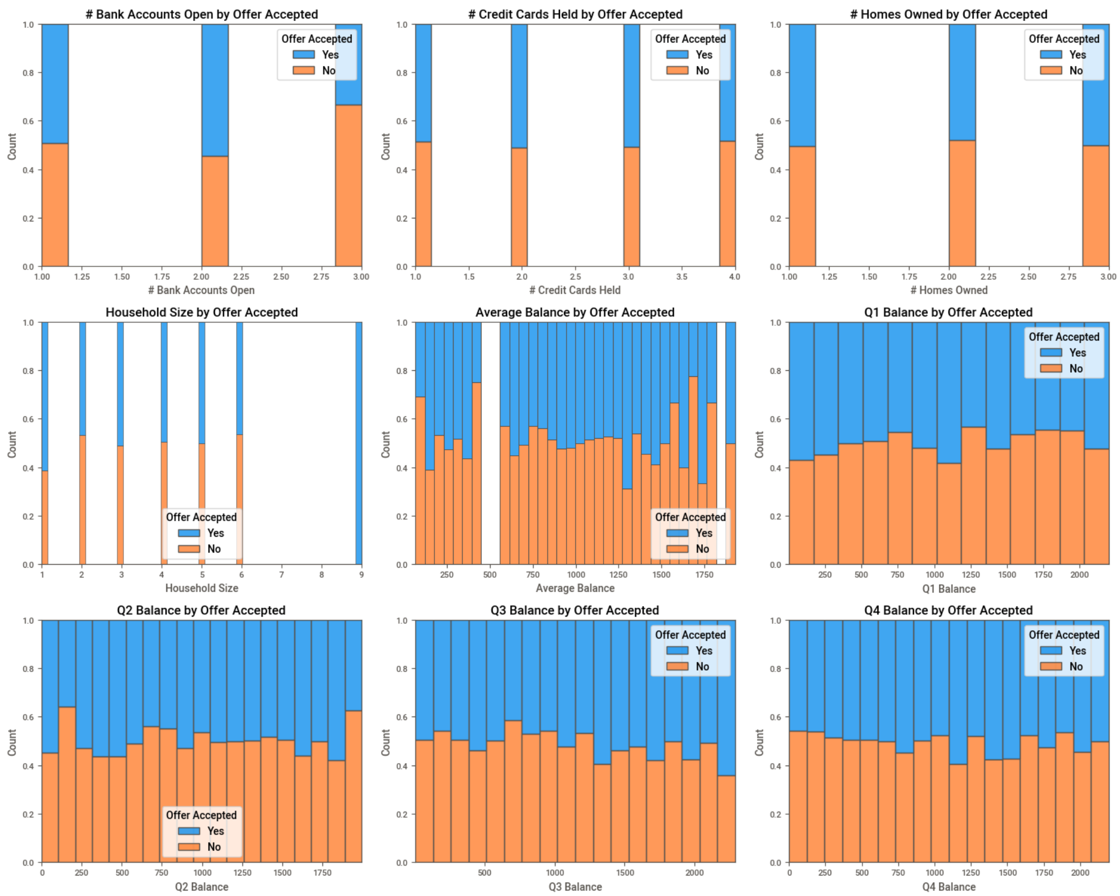
```
# Determine the number of rows and columns for subplots
num_plots = len(categorical_columns)
cols = 3  # Number of columns in the grid
rows = np.ceil(num_plots / cols).astype(int)  # Number of rows in the grid

# Create a figure with a grid of subplots
fig, axes = plt.subplots(rows, cols, figsize=(cols * 5, rows * 4))
axes = axes.flatten()  # Flatten the array of axes for easy iteration

# Loop through the columns and plot each one
for i, column in enumerate(categorical_columns):
    sns.countplot(data=balanced_data, x=column, hue='Offer Accepted',␣
 ↪ax=axes[i])
    axes[i].set_title(f'{column} by Offer Accepted')

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

# Display the plots
plt.tight_layout()
plt.show()
```



[14]:

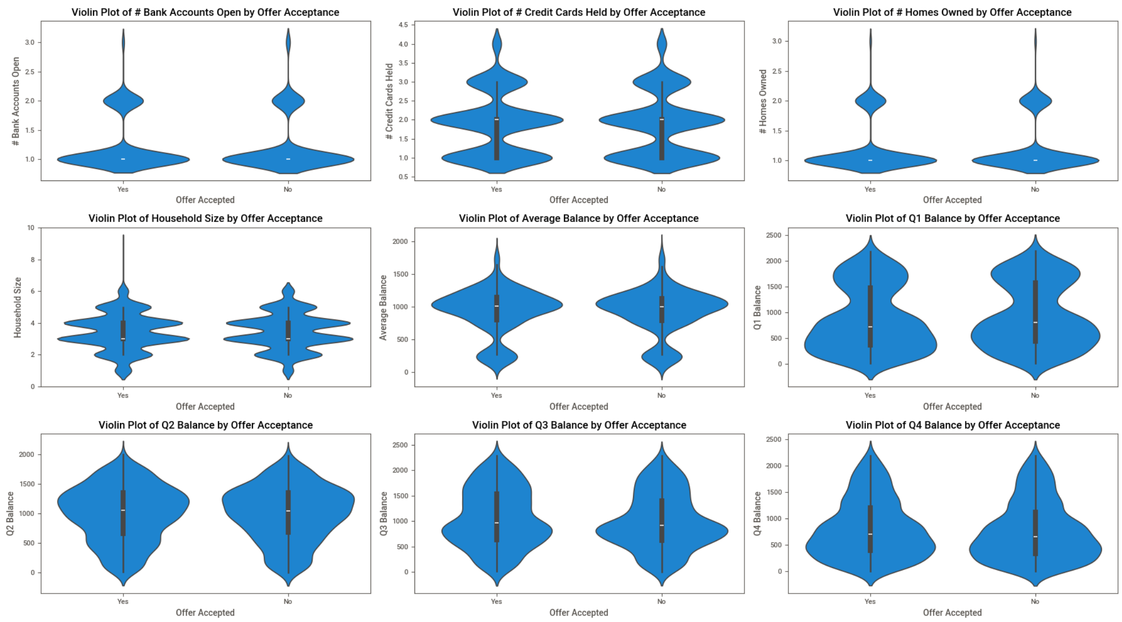
```
# Determine the number of rows and columns for subplots
num_plots = len(numerical_columns)
cols = 3  # Number of columns in the grid
rows = np.ceil(num_plots / cols).astype(int)  # Number of rows in the grid

# Create a figure with a grid of subplots
fig, axes = plt.subplots(rows, cols, figsize=(cols * 5, rows * 4))
axes = axes.flatten()  # Flatten the array of axes for easy iteration

# Loop through the columns and plot each one
for i, column in enumerate(numerical_columns):
    sns.histplot(data=balanced_data, x=column, hue='Offer Accepted', ax=axes[i], multiple='fill')
    axes[i].set_title(f'{column} by Offer Accepted')

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')
```

```python
# Display the plots
plt.tight_layout()
plt.show()
```



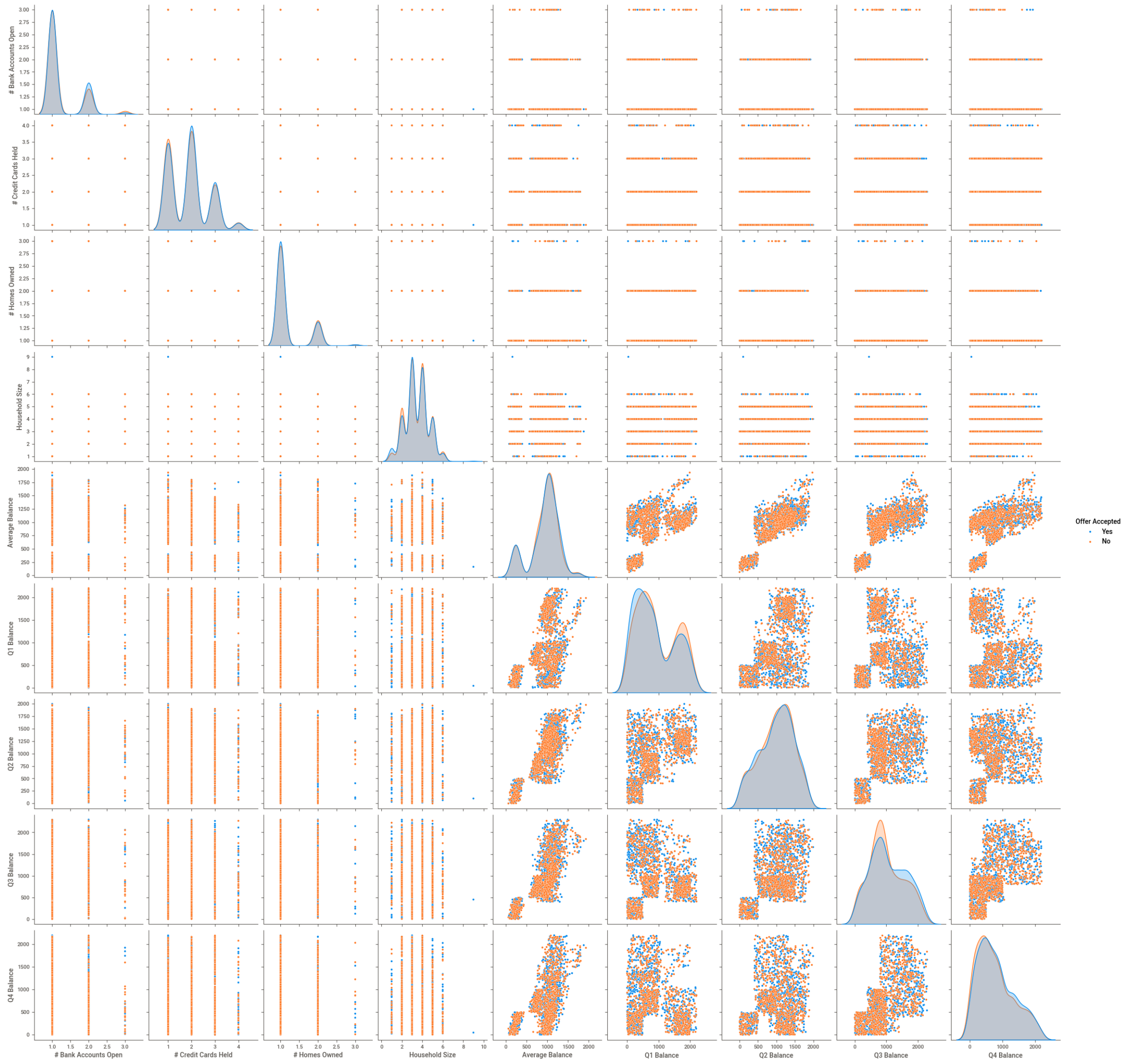
[48]:
```python
# Determine the number of rows and columns for subplots
num_plots = len(numerical_columns)
cols = 3  # Number of columns in the grid
rows = np.ceil(num_plots / cols).astype(int)  # Number of rows in the grid

# Create a figure with a grid of subplots
fig, axes = plt.subplots(rows, cols, figsize=(cols * 5, rows * 4))
axes = axes.flatten()  # Flatten the array of axes for easy iteration

# Create a violin plot for each numerical feature
for i, feature in enumerate(numerical_columns):
    sns.violinplot(x='Offer Accepted', y=feature, data=balanced_data, ax=axes[i])
    axes[i].set_title(f'Violin Plot of {feature} by Offer Acceptance')
```

```
# Remove any unused subplots if the number of features is less than the number of subplots
for j in range(len(numerical_columns), len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```



[15]:
```
# kde plots
sns.pairplot(data=balanced_data, hue='Offer Accepted', diag_kind='kde')
```

[15]: <seaborn.axisgrid.PairGrid at 0x7b152e33fcd0>



1.2.2 Cluster Analysis

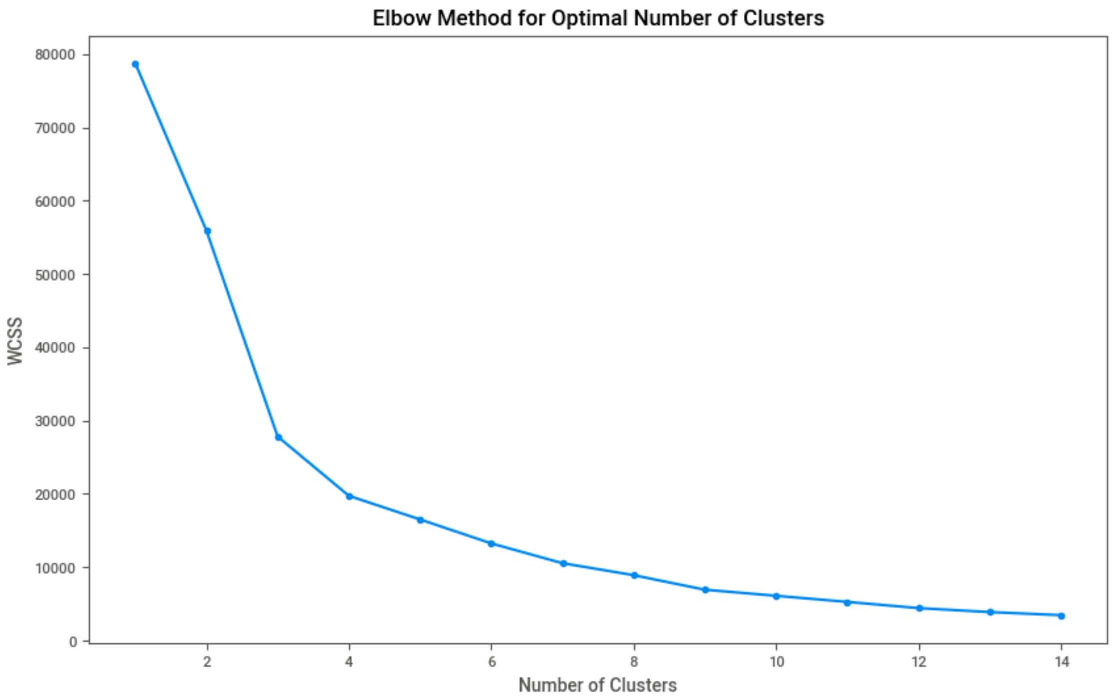
[16]:
```python
# cluster analysis using PCA
from sklearn.preprocessing import StandardScaler

features = data.drop('Offer Accepted', axis=1)

# One-hot encode categorical features
features_encoded = pd.get_dummies(features, drop_first=True)

# Standardize features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features_encoded)
```

[17]:

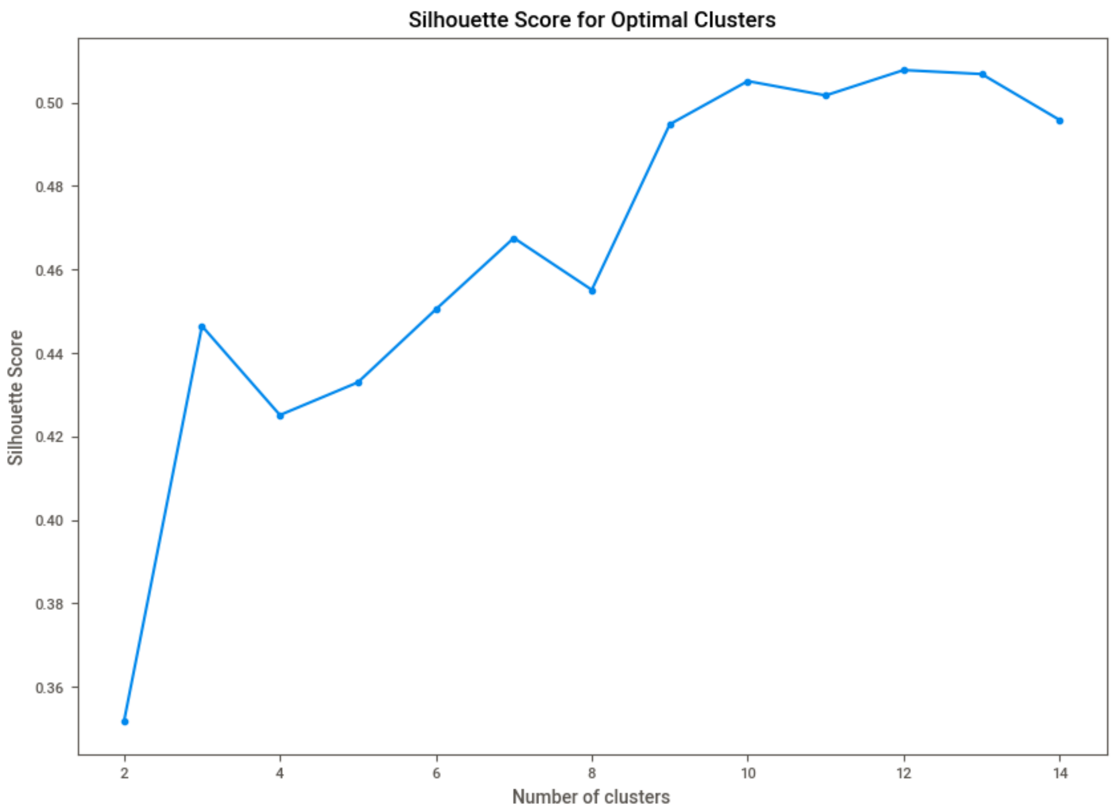
```python
from sklearn.decomposition import PCA

pca = PCA(n_components=2)  # Use 2 components for 2D visualization
pca_result = pca.fit_transform(scaled_features)
```

[18]:

```python
# elbow method
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

wcss = []

# Testing from 1 to 10 clusters
for i in range(1, 15):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(pca_result)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(range(1, 15), wcss, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.show()
```

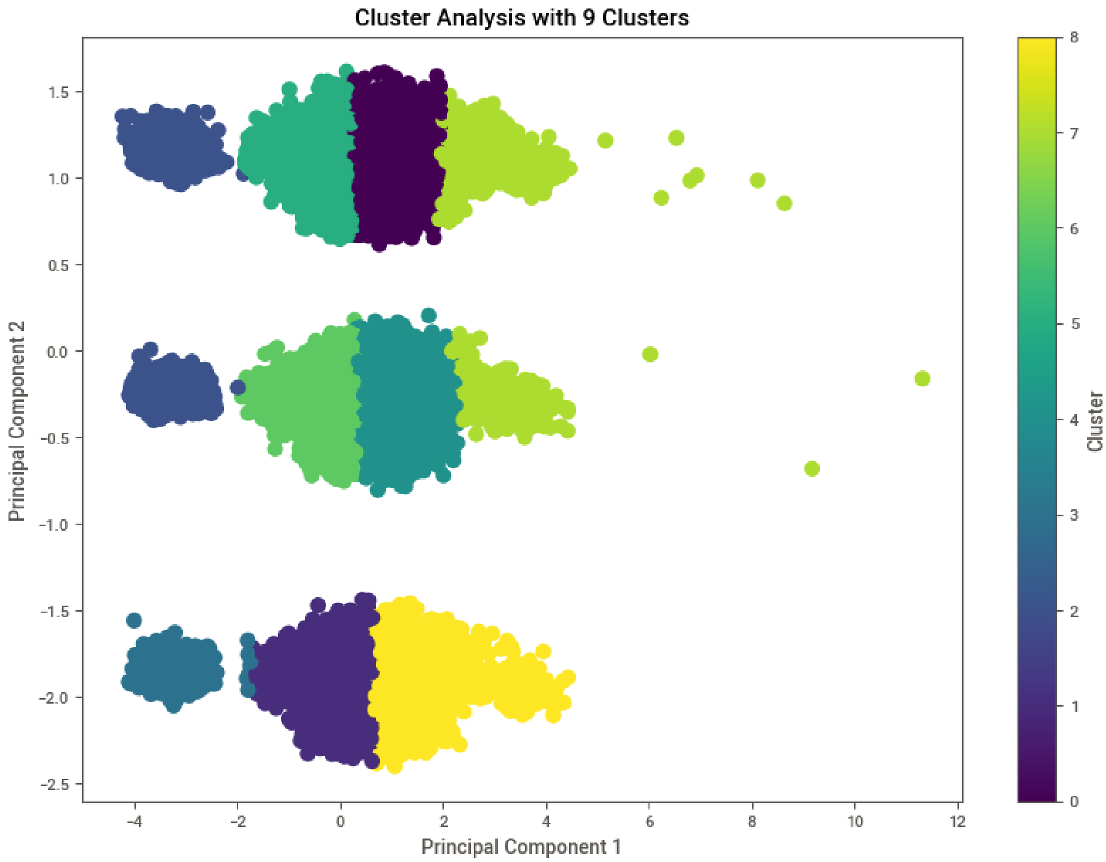


[19]:
```python
# silhouette score
from sklearn.metrics import silhouette_score

# Calculate Silhouette Scores for a range of cluster numbers
silhouette_scores = []
for n_clusters in range(2, 15):  # At least 2 clusters needed for silhouette␣
 ↪score
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    clusters = kmeans.fit_predict(pca_result)
    silhouette_avg = silhouette_score(pca_result, clusters)
    silhouette_scores.append(silhouette_avg)

# Plot the Silhouette Scores
plt.figure(figsize=(10, 7))
plt.plot(range(2, 15), silhouette_scores, marker='o')
plt.title('Silhouette Score for Optimal Clusters')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.show()
```



[20]:
```python
optimal_clusters = 9
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
clusters = kmeans.fit_predict(pca_result)
```

[21]:
```python
plt.figure(figsize=(10, 7))
plt.scatter(pca_result[:, 0], pca_result[:, 1], c=clusters, cmap='viridis',␣
 ↪s=50)
plt.colorbar(label='Cluster')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title(f'Cluster Analysis with {optimal_clusters} Clusters')
plt.show()
```

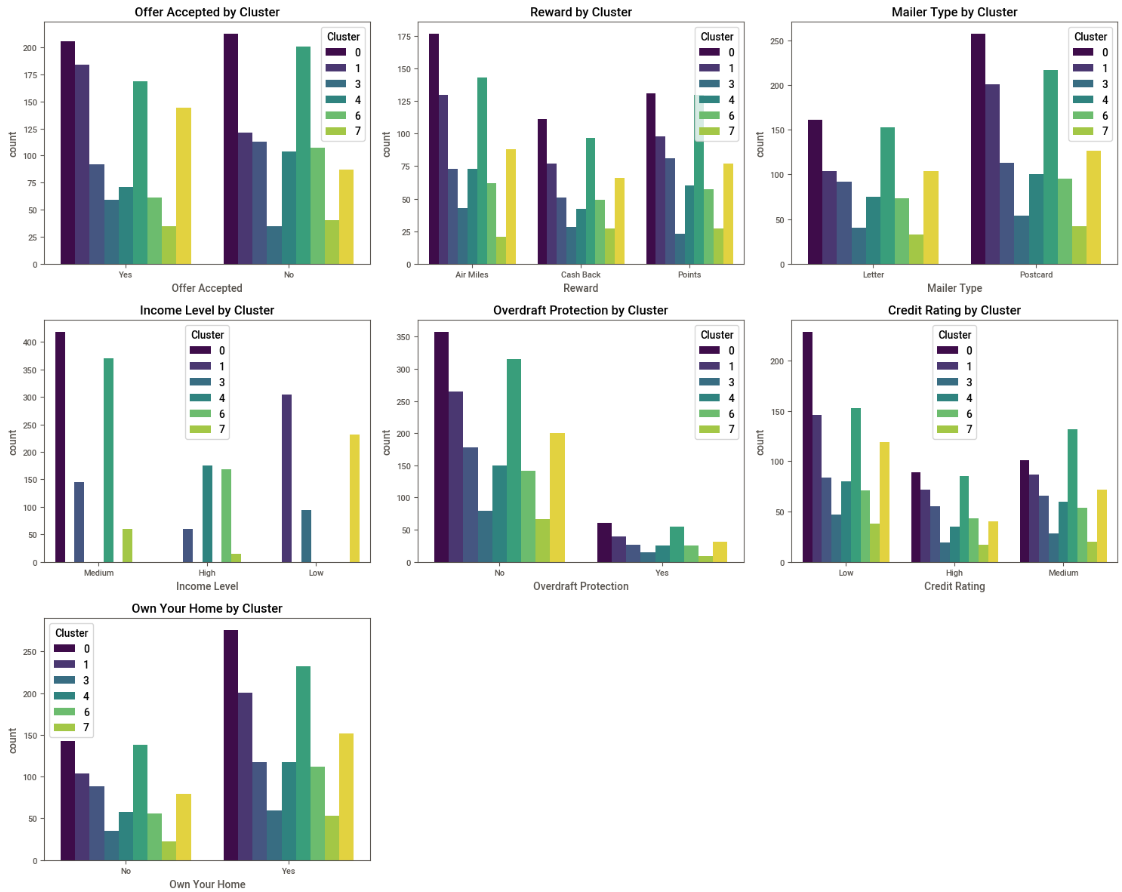


[22]:
```python
# Add cluster labels to DataFrame
data['Cluster'] = clusters
```

[23]:
```python
minority_class = data[data['Offer Accepted'] == 'Yes']
majority_class = data[data['Offer Accepted'] == 'No']
```

```
majority_class_sampled = majority_class.sample(n=len(minority_class),␣
↪random_state=42)

balanced_data = pd.concat([minority_class, majority_class_sampled])
```

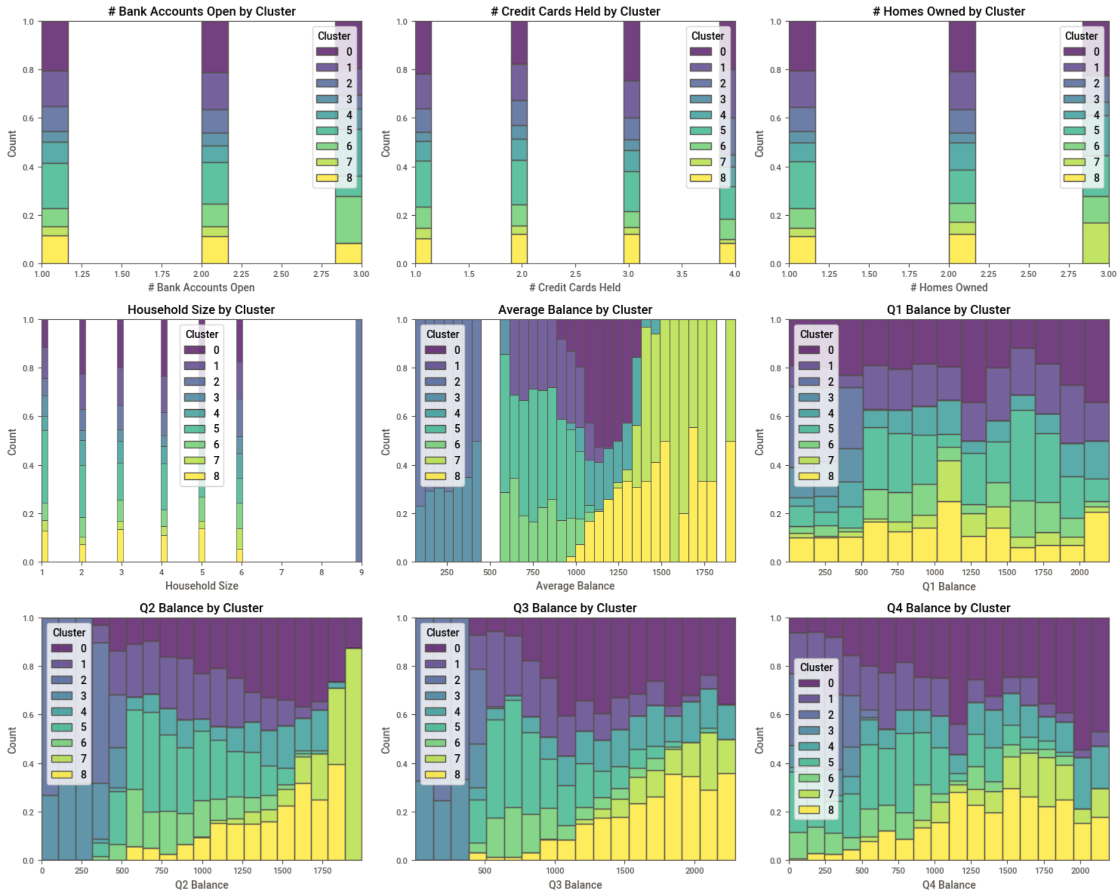
[24]:
```
# Determine the number of rows and columns for subplots
num_plots = len(categorical_columns)
cols = 3  # Number of columns in the grid
rows = np.ceil(num_plots / cols).astype(int)  # Number of rows in the grid

# Create a figure with a grid of subplots
fig, axes = plt.subplots(rows, cols, figsize=(cols * 5, rows * 4))
axes = axes.flatten()  # Flatten the array of axes for easy iteration

# Loop through the columns and plot each one
for i, column in enumerate(categorical_columns):
    sns.countplot(data=balanced_data, x=column, hue='Cluster', ax=axes[i],␣
↪palette='viridis')
    axes[i].set_title(f'{column} by Cluster')

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

# Display the plots
plt.tight_layout()
plt.show()
```



[25]:

```
# Determine the number of rows and columns for subplots
num_plots = len(numerical_columns)
cols = 3  # Number of columns in the grid
rows = np.ceil(num_plots / cols).astype(int)  # Number of rows in the grid

# Create a figure with a grid of subplots
fig, axes = plt.subplots(rows, cols, figsize=(cols * 5, rows * 4))
axes = axes.flatten()  # Flatten the array of axes for easy iteration

# Loop through the columns and plot each one
for i, column in enumerate(numerical_columns):
    sns.histplot(data=balanced_data, x=column, hue='Cluster', ax=axes[i], palette='viridis', multiple='fill')
    axes[i].set_title(f'{column} by Cluster')

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')
```

```python
# Display the plots
plt.tight_layout()
plt.show()
```



1.3 Preprocessing

1.3.1 Pipeline

[26]:
```python
df = pd.read_csv('data.csv')
```

[27]:
```python
df.head()
```

[27]:
```
   index  Customer Number Offer Accepted     Reward Mailer Type Income Level  \
0      0                1             No  Air Miles      Letter         High
1      1                2             No  Air Miles      Letter       Medium
2      2                3             No  Air Miles    Postcard         High
3      3                4             No  Air Miles      Letter       Medium
4      4                5             No  Air Miles      Letter       Medium
```

```
   # Bank Accounts Open Overdraft Protection Credit Rating  \
0                    1                   No           High
1                    1                   No         Medium
2                    2                   No         Medium
3                    2                   No           High
4                    1                   No         Medium

   # Credit Cards Held  # Homes Owned  Household Size Own Your Home  \
0                   2              1               4            No
1                   2              2               5           Yes
2                   2              1               2           Yes
3                   1              1               4            No
4                   2              1               6           Yes

   Average Balance  Q1 Balance  Q2 Balance  Q3 Balance  Q4 Balance
0          1160.75      1669.0       877.0      1095.0      1002.0
1           147.25        39.0       106.0        78.0       366.0
2           276.50       367.0       352.0       145.0       242.0
3          1219.00      1578.0      1760.0      1119.0       419.0
4          1211.00      2140.0      1357.0       982.0       365.0
```

[28]:
```
num_cols = ['# Bank Accounts Open', '# Credit Cards Held', '# Homes Owned',␣
 ↪'Household Size', 'Average Balance']
cat_cols = ['Reward', 'Mailer Type', 'Income Level', 'Credit Rating']

scale_cols = ['Average Balance']

onehot_cols = ['Reward', 'Mailer Type']
ordinal_encode_cols = ['Income Level', 'Credit Rating']
income_order = ['Low', 'Medium', 'High']
credit_order = ['Low', 'Medium', 'High']

columns_to_drop = ['index', 'Customer Number', 'Own Your Home', 'Overdraft␣
 ↪Protection', 'Q1 Balance', 'Q2 Balance', 'Q3 Balance', 'Q4 Balance']
```

[29]:
```
df_no_mv = df.dropna()
```

[30]:
```
df_no_mv = df_no_mv.copy()
df_no_mv['Cluster'] = clusters
```

[31]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.impute import KNNImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
```

```python
from feature_engine.outliers import Winsorizer

preprocessor = ColumnTransformer(
    transformers=[
        ('drop', 'drop', columns_to_drop),
        ('scale', StandardScaler(), scale_cols),
        ('onehot', OneHotEncoder(handle_unknown='ignore'), onehot_cols),
        ('ordinal_encode', OrdinalEncoder(categories=[income_order,␣
↪credit_order]), ordinal_encode_cols)
    ],
    remainder='passthrough'
    )
```

1.3.2 Train Test Split

[32]:
```python
X = df_no_mv.drop('Offer Accepted', axis=1)
y = df_no_mv['Offer Accepted']

y = y.map({'Yes': 1, 'No': 0})
```

[33]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪stratify=y, random_state=42)
```

1.4 Modelling

1.4.1 Selecting Models

logistic regression, svm, naive bayes, decision tree, random forest, xgboost

[34]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
```

[36]:
```python
# Fit the pipeline

pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', model)])
pipeline.fit(X_train, y_train)
```

```
/home/hpark/Syncthing/Professional/DS_Projects/Credit_Card_Offer_Acceptance/.ven
v/lib/python3.10/site-packages/sklearn/compose/_column_transformer.py:1623:
FutureWarning:
The format of the columns of the 'remainder' transformer in
```

```
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

[36]:

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('drop', 'drop',
                                                  ['index', 'Customer Number',
                                                   'Own Your Home',
                                                   'Overdraft Protection',
                                                   'Q1 Balance', 'Q2 Balance',
                                                   'Q3 Balance',
                                                   'Q4 Balance']),
                                                 ('scale', StandardScaler(),
                                                  ['Average Balance']),
                                                 ('onehot',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['Reward', 'Mailer Type']),
                                                 ('ordinal_encode',
OrdinalEncoder(categories=[['Low',
'Medium',
'High'],
['Low',
'Medium',
'High']]),
                                                  ['Income Level',
                                                   'Credit Rating'])])),
                ('model', LogisticRegression(n_jobs=4, random_state=42))])
```

[37]:

```
models = [
    LogisticRegression(random_state=42, n_jobs=4),
    SVC(random_state=42),
    GaussianNB(),
    DecisionTreeClassifier(random_state=42),
    RandomForestClassifier(random_state=42, n_jobs=4),
    xgb.XGBClassifier(random_state=42)
]
```

[38]:

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_predict
from imblearn.pipeline import Pipeline as imbPipeline
```

```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import confusion_matrix, f1_score
from sklearn.metrics import classification_report

for model in models:
    pipeline = imbPipeline(steps=[
        ('preprocessor', preprocessor), ('rus',␣
 ↪RandomUnderSampler(random_state=42)), ('smote', SMOTE(random_state=42)),␣
 ↪('model', model)
        ])
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)

    y_pred = cross_val_predict(pipeline, X_train, y_train, cv=kfold)
    print(f'{model.__class__.__name__} F1 score: {f1_score(y_train, y_pred):.
 ↪3f}')
    print(confusion_matrix(y_train, y_pred))
```

```
LogisticRegression F1 score: 0.197
[[8976 4587]
 [ 227  590]]
SVC F1 score: 0.193
[[8900 4663]
 [ 233  584]]
GaussianNB F1 score: 0.191
[[9138 4425]
 [ 262  555]]
DecisionTreeClassifier F1 score: 0.149
[[8142 5421]
 [ 314  503]]
RandomForestClassifier F1 score: 0.173
[[8554 5009]
 [ 265  552]]
XGBClassifier F1 score: 0.168
[[8453 5110]
 [ 272  545]]
```

Not horrible F1 Scores: LogisticRegression, SVC, GaussianNB

1.4.2 Model Tuning

[39]:
```
best_models = [
    LogisticRegression(random_state=42, n_jobs=4),
    SVC(random_state=42),
    GaussianNB(),
]
```

[40]:
```python
# boosting
from sklearn.ensemble import AdaBoostClassifier

# Define your base model
base_model = LogisticRegression(random_state=42, n_jobs=4)

# Define the boosting classifier
model = AdaBoostClassifier(estimator=base_model, n_estimators=50,␣
 ↪random_state=42)

# Create the pipeline
pipeline = imbPipeline(steps=[('preprocessor', preprocessor), ('rus',␣
 ↪RandomUnderSampler(random_state=42)), ('smote', SMOTE(random_state=42)),␣
 ↪('model', model)])

# Perform cross-validation
kfold = KFold(n_splits=3, shuffle=True, random_state=42)

y_pred = cross_val_predict(pipeline, X_train, y_train, cv=kfold)
print(f'{model.__class__.__name__} F1 score: {f1_score(y_train, y_pred):.3f}')
print(confusion_matrix(y_train, y_pred))
```

```
/home/hpark/Syncthing/Professional/DS_Projects/Credit_Card_Offer_Acceptance/.ven
v/lib/python3.10/site-packages/sklearn/ensemble/_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be
removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
/home/hpark/Syncthing/Professional/DS_Projects/Credit_Card_Offer_Acceptance/.ven
v/lib/python3.10/site-packages/sklearn/ensemble/_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be
removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
/home/hpark/Syncthing/Professional/DS_Projects/Credit_Card_Offer_Acceptance/.ven
v/lib/python3.10/site-packages/sklearn/ensemble/_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be
removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```

```
AdaBoostClassifier F1 score: 0.197
[[9206 4357]
 [ 252  565]]
```

[41]:
```python
# stacking
from mlxtend.regressor import StackingRegressor

# Initialize the base models
model0 = best_models[0]
```

```
model1 = best_models[1]
model2 = best_models[2]

# Initialize the meta model
meta_model = LogisticRegression()

# Initialize the stacking model
stacking_model = StackingRegressor(regressors=[model0, model1, model2],␣
↪meta_regressor=meta_model)

pipeline = imbPipeline(steps=[('preprocessor', preprocessor), ('rus',␣
↪RandomUnderSampler(random_state=42)), ('smote', SMOTE(random_state=42)),␣
↪('model', stacking_model)])
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
y_pred = cross_val_predict(pipeline, X_train, y_train, cv=kfold)

print(f'{stacking_model.__class__.__name__} F1 score: {f1_score(y_train,␣
↪y_pred):.3f}')
print(confusion_matrix(y_train, y_pred))
```

```
StackingRegressor F1 score: 0.194
[[8890 4673]
 [ 228  589]]
```

[42]:
```
# voting
from sklearn.ensemble import VotingClassifier

# Initialize the base models
model0 = best_models[0]
model1 = best_models[1]
model2 = best_models[2]

# Initialize the meta model
meta_model = LogisticRegression()

# Initialize the voting model
voting_model = VotingClassifier(estimators=[('lr', model0), ('svc', model1),␣
↪('gnb', model2)], voting='hard')

pipeline = imbPipeline(steps=[('preprocessor', preprocessor), ('rus',␣
↪RandomUnderSampler(random_state=42)), ('smote', SMOTE(random_state=42)),␣
↪('model', voting_model)])
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
y_pred = cross_val_predict(pipeline, X_train, y_train, cv=kfold)

print(f'{voting_model.__class__.__name__} F1 score: {f1_score(y_train, y_pred):.
↪3f}')
```

```
print(confusion_matrix(y_train, y_pred))
```

```
VotingClassifier F1 score: 0.196
[[9009 4554]
 [ 234  583]]
```

1.4.3 Feature Importance

[43]:
```
from sklearn.feature_selection import RFE

model = best_models[0]

rfe = RFE(estimator=model, n_features_to_select=10)

pipeline = imbPipeline(steps=[('preprocessor', preprocessor), ('rus',␣
↪RandomUnderSampler(random_state=42)), ('smote', SMOTE(random_state=42)),␣
↪('feature_selection', rfe), ('model', model)])

pipeline.fit(X_train, y_train)

# Retrieve coefficients from the model
model = pipeline.named_steps['model']
coefficients = model.coef_[0]

# Retrieve feature names
# After preprocessing, combine feature names
def get_feature_names(preprocessor, X):
    feature_names = []

    # For each transformer in the preprocessor
    for name, trans, columns in preprocessor.transformers_:
        if name == 'drop':
            continue  # No feature names for dropped columns
        if hasattr(trans, 'get_feature_names_out'):
            feature_names.extend(trans.get_feature_names_out())
        else:
            feature_names.extend(columns)

    return feature_names

# Get feature names after preprocessing
feature_names_after_preprocessing = get_feature_names(preprocessor, X)

# Get selected features from RFE
selected_indices = np.where(pipeline.named_steps['feature_selection'].
↪support_)[0]
```

```python
selected_feature_names = [feature_names_after_preprocessing[i] for i in␣
↪selected_indices]

print("Top 10 features:", selected_feature_names)

# Combine feature names with coefficients
feature_importances = dict(zip(feature_names_after_preprocessing, coefficients))

# Sort features by importance
sorted_importances = sorted(feature_importances.items(), key=lambda x:␣
↪abs(x[1]), reverse=True)

print("Feature Importances:")
for feature, importance in sorted_importances:
    print(f"{feature}: {importance:.4f}")
```

```
Top 10 features: ['Average Balance', 'Reward_Air Miles', 'Reward_Cash Back',
'Reward_Points', 'Mailer Type_Letter', 'Mailer Type_Postcard', 'Income Level',
'Credit Rating', '# Homes Owned', 'Household Size']
Feature Importances:
Credit Rating: -0.9123
Reward_Cash Back: -0.5283
Income Level: -0.4927
Mailer Type_Letter: -0.4591
Mailer Type_Postcard: 0.4585
Reward_Air Miles: 0.4493
Reward_Points: 0.0783
# Credit Cards Held: -0.0732
# Bank Accounts Open: 0.0711
Average Balance: -0.0168
```

1.5 Conclusion

1.5.1 Summary of Findings

- **Key Results:** All models, including ensemble methods, achieved an F1 score below 0.2, with the Logistic Regression model performing slightly better at an F1 score of 0.197. This low performance suggests that the current features in the dataset do not strongly influence the likelihood of a customer accepting a credit card offer. Exploratory data analysis supports this, showing weak associations between the target variable and the features. Interestingly, the analysis indicates that wealthier customers or those offered cash-back rewards are less likely to accept a credit card offer compared to other reward types, such as air miles.

1.5.2 Implications

- **Impact:** The project successfully identified distinct customer segments, revealing diverse behaviors. However, the low F1 scores suggest that the current models are not effective in predicting credit card offer acceptance, indicating that the results are inconclusive. To improve prediction capabilities, future work should focus on enhancing the dataset by incorporating additional relevant features and refining the model to better capture customer characteristics and behaviors.

- **Applications:** The segmentation of customers into distinct clusters enables more targeted marketing strategies. Financial institutions can use these insights to design personalized offers and optimize marketing campaigns, potentially increasing the effectiveness of credit card promotions and reducing associated costs. This approach allows for more tailored and cost-efficient marketing efforts.

1.5.3 Limitations

- **Constraints:** The dataset lacks information on the attractiveness of credit card offers and whether a sign-up bonus is included. Additionally, it does not provide details on the dates when the offers were made. These omissions limit the ability to fully understand factors influencing acceptance. Moreover, the models struggled to address the dataset's limitations, including the imbalance in the target variable.
- **Impact of Limitations:** The absence of strong correlations between the target variable and the features harms. model performance. As a result, the models developed from this dataset exhibit limited performance in predicting credit card offer acceptance, highlighting the need for more comprehensive data to improve prediction capabilities.

1.5.4 Future Work

- **Recommendations:** Utilizing a more comprehensive dataset could enhance predictive power. Future analyses could explore the impact of macroeconomic conditions on the likelihood of credit card offer acceptance and consider alternative data sources or additional variables that might provide further insights.
- **Improvements:** Adding more relevant features to the dataset, such as detailed offer attractiveness, sign-up bonuses, and timing, could improve model performance. Integrating external data sources could also provide a more robust foundation for predictions.

1.5.5 Final Thoughts

- **Reflection:** This project has deepened the understanding of factors influencing credit card offer acceptance. Key insights include the role of customer wealth and the type of reward in determining acceptance. The analysis highlights the critical importance of comprehensive data collection and the need for continuous refinement of predictive models to capture the nuances of customer behavior.
- **Acknowledgements:** Dataset