Notebook

October 15, 2024

1 Machine Learning Engineer Nanodegree

1.1 Starbuck's Capstone Challenge

Matteo Giuliani
October 11th, 2024

1.2 I. Definition

1.2.1 Project Overview

Starbucks, like many companies, wants to make sure that their customers are aware of and use the special offers and promotions they send out. These offers could include discounts on coffee or snacks, or buy-one-get-one-free deals. The main challenge is to figure out how to make sure that the right offers are sent to the right customers—essentially, understanding what kinds of promotions different customers like and are likely to respond to.

Problem Domain: Starbucks sends out different types of offers to customers through its mobile app, such as discounts, BOGO (buy-one-get-one-free) deals, or even just information about a new product. But not all customers are interested in every offer. Some people might be more inclined to respond to a 20% discount, while others might be more interested in trying a new product for free. The goal is to use data to identify which types of offers are most effective for which customers, and when the best time is to send them.

Project Origin: This project comes from a real-world problem that Starbucks faces as it tries to improve customer engagement and satisfaction. By analyzing data about customer behavior and the effectiveness of different offers, we can help Starbucks better understand its customers and send out promotions that they are more likely to appreciate and use. This means a better experience for customers and more successful marketing efforts for Starbucks.

Data Sets and Input Data: The project uses data that includes: - **Customer Profiles (profiles.json)**: Information about customers, such as their age, income, and when they became members. - **Offers Data (portfolio.json)**: Details about the different offers that were sent out, including the type of offer and its duration. - **Transaction Data (transcript.json)**: Records of purchases made by customers, showing whether they responded to offers.

The challenge is to analyze this data and create a model that predicts which offers each customer is likely to respond to, allowing Starbucks to better target its promotions and improve customer satisfaction. The ultimate goal is to optimize how offers are sent out to improve both customer experience and sales.

1.2.2 Problem Statement

The primary challenge is to determine which types of promotional offers are most effective for different customers, based on their preferences and behaviors. Starbucks needs a way to match each offer type—such as discounts, BOGO (buy-one-get-one-free) deals, or new product informational to the customers who are most likely to respond positively. This problem arises from the need to improve the effectiveness of marketing efforts, which in turn could enhance customer satisfaction and increase revenue.

The goal is to build a predictive model that can analyze customer data and predict the likelihood that a customer will respond to a particular offer. This model will allow Starbucks to make data-driven decisions when sending out offers, ensuring that customers receive promotions that are relevant to their interests and habits.

To solve this problem, the following strategy will be employed:

1. **Data Exploration**: Investigate the structure and quality of the dataset, identifying key features and understanding how customer demographics, offers, and transactions are related.
2. **Data Preprocessing**: Clean and preprocess the data, handling any missing values, reaname columns or split aggregated data.
3. **Exploratory Data Analysis (EDA)**: Analyze the relationships between customer demographics, purchase behaviors, and offer responses to identify trends and insights that can inform model building.
4. **Model Selection and Training**: Train two machine learning models: a Random Forest and a Decision Tree. These models will be designed to predict the likelihood of a customer responding to an offer.
5. **Model Evaluation**: Compare the performance of the Random Forest and Decision Tree models against a benchmark K-Neighbors Classifier. The primary evaluation metric will be the F1 score, which balances precision and recall, providing a measure of a model's effectiveness in identifying positive responses to offers.

1.2.3 Anticipated Solution

The intended solution is a predictive model that identifies which offers are most suitable for each customer segment. By sending personalized offers, Starbucks can increase the engagement rate of their promotions and ensure that customers receive offers they are more likely to use.

This solution is expected to improve marketing efficiency, reducing the costs associated with sending irrelevant offers and increasing customer satisfaction. Customers benefit from receiving promotions that match their preferences, while Starbucks benefits from higher conversion rates and increased sales. Additionally, the analysis could provide deeper insights into customer behavior, helping Starbucks make more informed decisions regarding future promotions and marketing strategies.

1.2.4 Metrics

For this project, I will build two models using **RandomForestClassifier** and **DecisionTreeClassifier**, and compare their **F1 score** against a **KNeighborsClassifier** benchmark.

Metric Selection

- **F1 Score**: The primary metric for comparison, as it balances **precision** and **recall**. This is crucial for the Starbucks Challenge, where both false positives (predicting a response that doesn't occur) and false negatives (missing a responder) matter.

Model Comparison

- **RandomForestClassifier**: Uses multiple decision trees for robust predictions and reduces overfitting.
- **DecisionTreeClassifier**: A simpler model that is easier to interpret but more prone to overfitting.
- **KNeighborsClassifier**: Serves as a benchmark model, offering a straightforward comparison point for more complex models.

Each model's F1 score will be compared to see if RandomForest or DecisionTree significantly outperforms the benchmark, helping select the best model for predicting customer responses to Starbucks offers.

1.3 II. Analysis

1.3.1 Data Exploration

The dataset consists of three distinct files:

- **portfolio.json** - Contains details about various offers, including their IDs and specific attributes like type and duration.
- **profile.json** - Includes demographic details for each customer.
- **transcript.json** - Tracks all records of interactions, including transactions, receipt of offers, views, and completions.

Below is a description of the structure and details for each variable found in the files:

portfolio.json * **id** (string) - Unique identifier for each offer. * **offer_type** (string) - Describes the nature of the offer, such as "Buy One Get One," discounts, or informational. * **difficulty** (int) - The minimum expenditure required to qualify for the offer. * **reward** (int) - The incentive given upon successful completion of the offer. * **duration** (int) - Validity period of the offer, measured in days. * **channels** (list of strings) - Communication methods used for the offer.

profile.json * **age** (int) - The customer's age. * **became_member_on** (int) - The registration date when the customer joined the app. * **gender** (str) - Indicates the customer's gender (note: some entries include 'O' for non-binary or other). * **id** (str) - Unique identifier for each customer. * **income** (float) - The annual earnings of the customer.

transcript.json * **event** (str) - Describes the type of interaction (e.g., transaction, receipt of an offer, viewing of an offer). * **person** (str) - Identifies the customer associated with each interaction. * **time** (int) - Indicates the time in hours since the beginning of the testing period, starting at hour zero. * **value** (dict of strings) - Contains either a transaction amount or an offer ID, depending on the interaction type.

1.3.2 Exploratory Visualization

[1161]:
```
import pandas as pd

profile_df = pd.read_json('datasets/profile.json', orient='records', lines=True)
transcript_df = pd.read_json('datasets/transcript.json', orient='records',␣
↪lines=True)
portfolio_df = pd.read_json('datasets/portfolio.json', orient='records',␣
↪lines=True)
```

[1162]:
```
import matplotlib.pyplot as plt
import seaborn as sns

def plot_outliers(df, colName, color='#3DDBDB' , figsize=(8, 6), title_size=16,␣
↪label_size=12):

    plt.figure(figsize=figsize)
    sns.boxplot(
        x=df[colName],
        color=color,
        width=0.5)

    plt.title(f'Outliers in {colName}', fontsize=title_size, pad=15)
    plt.xlabel(colName, fontsize=label_size)
    sns.despine(left=True)

    plt.tight_layout()
    plt.show()

def column_bar_plot(df, colName, pltTitle, palette='viridis', figsize=(8, 6),␣
↪title_size=16, label_size=12):
    if df[colName].dtype in ['int64', 'float64']:
        value_counts = df[colName].value_counts().sort_index().reset_index()
    else:
        value_counts = df[colName].value_counts().reset_index()

    value_counts.columns = [colName, 'Counts']

    plt.figure(figsize=figsize)
    fig, ax = plt.subplots()

    sns.barplot(
        data=value_counts,
        x=colName,
        y='Counts',
        palette=palette,
        ax=ax,
```

```
        hue=colName if df[colName].dtype not in ['int64', 'float64'] else␣
↪colName,
        legend=False
    )

    for i, v in enumerate(value_counts['Counts']):
        ax.text(i, v + 0.05 * max(value_counts['Counts']), str(v),␣
↪color='black',
                fontsize=label_size, ha='center', fontweight='bold')

    ax.set_title(pltTitle, fontsize=title_size, pad=15)
    ax.set_xlabel(colName, fontsize=label_size)
    ax.set_ylabel('Counts', fontsize=label_size)
    plt.xticks(rotation=45, ha='right')
    sns.despine(left=True)
    plt.tight_layout()
    plt.show()


def distribution_plot(df, colName, pltTitle, palette='viridis', figsize=(8, 6),␣
↪title_size=16, label_size=12, bins=30):
    plt.figure(figsize=figsize)
    fig, ax = plt.subplots()

    sns.histplot(
        data=df,
        x=colName,
        bins=bins,
        kde=True,
        color=sns.color_palette(palette, 1)[0],
        ax=ax
    )

    ax.set_title(pltTitle, fontsize=title_size, pad=15)
    ax.set_xlabel(colName, fontsize=label_size)
    ax.set_ylabel('', fontsize=label_size)
    sns.despine(left=True)
    plt.tight_layout()
    plt.show()

def grouped_bar_plot(df, colName, hueColName, pltTitle, palette='viridis',␣
↪figsize=(15, 5), title_size=16, label_size=12):
    plt.figure(figsize=figsize)
    fig, ax = plt.subplots()

    sns.countplot(
        data=df,
```

```
        x=colName,
        hue=hueColName,
        palette=palette,
        ax=ax
    )

    for p in ax.patches:
        ax.annotate(
            f'{int(p.get_height())}',
            (p.get_x() + p.get_width() / 2., p.get_height()),
            ha='center',
            va='center',
            fontsize=label_size,
            color='black',
            xytext=(0, 5),
            textcoords='offset points'
        )

    ax.set_title(pltTitle, fontsize=title_size, pad=15)
    ax.set_xlabel(colName, fontsize=label_size)
    ax.set_ylabel('Count', fontsize=label_size)

    ax.legend(
        title=hueColName,
        title_fontsize=label_size,
        fontsize=10,
        bbox_to_anchor=(1.05, 1),
        loc='upper left'
    )

    sns.despine(left=True)

    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

Profile dataset visualization

[1163]:
```
print("the profile dataset has {0} rows and {1} columns".format(str(profile_df.
    ↪shape[0]), str(profile_df.shape[1])))
```

```
the profile dataset has 17000 rows and 5 columns
```

[1164]:
```
profile_df.describe(include='all')
```

[1164]:
```
       gender          age                                id  \
count   14825  17000.000000                             17000
unique      3           NaN                             17000
```

```
top          M          NaN  68be06ca386d4c31939f3a4f0e3dd783
freq      8484          NaN                                 1
mean       NaN    62.531412                               NaN
std        NaN    26.738580                               NaN
min        NaN    18.000000                               NaN
25%        NaN    45.000000                               NaN
50%        NaN    58.000000                               NaN
75%        NaN    73.000000                               NaN
max        NaN   118.000000                               NaN

        became_member_on         income
count       1.700000e+04   14825.000000
unique               NaN            NaN
top                  NaN            NaN
freq                 NaN            NaN
mean        2.016703e+07   65404.991568
std         1.167750e+04   21598.299410
min         2.013073e+07   30000.000000
25%         2.016053e+07   49000.000000
50%         2.017080e+07   64000.000000
75%         2.017123e+07   80000.000000
max         2.018073e+07  120000.000000
```

[1165]:
```
profile_df.head()
```

[1165]:

	gender	age	id	became_member_on	income
0	None	118	68be06ca386d4c31939f3a4f0e3dd783	20170212	NaN
1	F	55	0610b486422d4921ae7d2bf64640c50b	20170715	112000.0
2	None	118	38fe809add3b4fcf9315a9694bb96ff5	20180712	NaN
3	F	75	78afa995795e4d85b5d9ceeca43f5fef	20170509	100000.0
4	None	118	a03223e636434f42ac4c3df47e8bac43	20170804	NaN

[1166]:
```
profile_df.info()
```

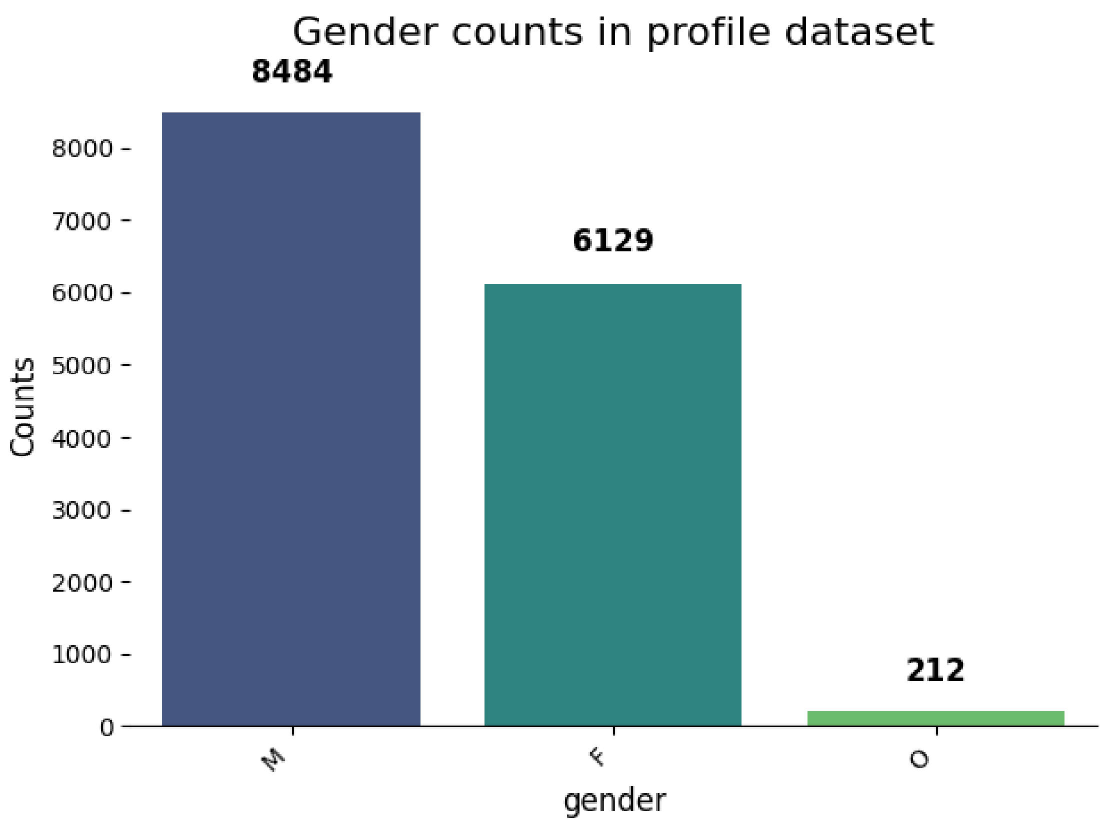
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17000 entries, 0 to 16999
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            14825 non-null  object
 1   age               17000 non-null  int64
 2   id                17000 non-null  object
 3   became_member_on  17000 non-null  int64
 4   income            14825 non-null  float64
dtypes: float64(1), int64(2), object(2)
memory usage: 664.2+ KB
```

[1167]:
```
#Check for null values
profile_df.isnull().sum()
```

[1167]:
```
gender              2175
age                    0
id                     0
became_member_on       0
income              2175
dtype: int64
```

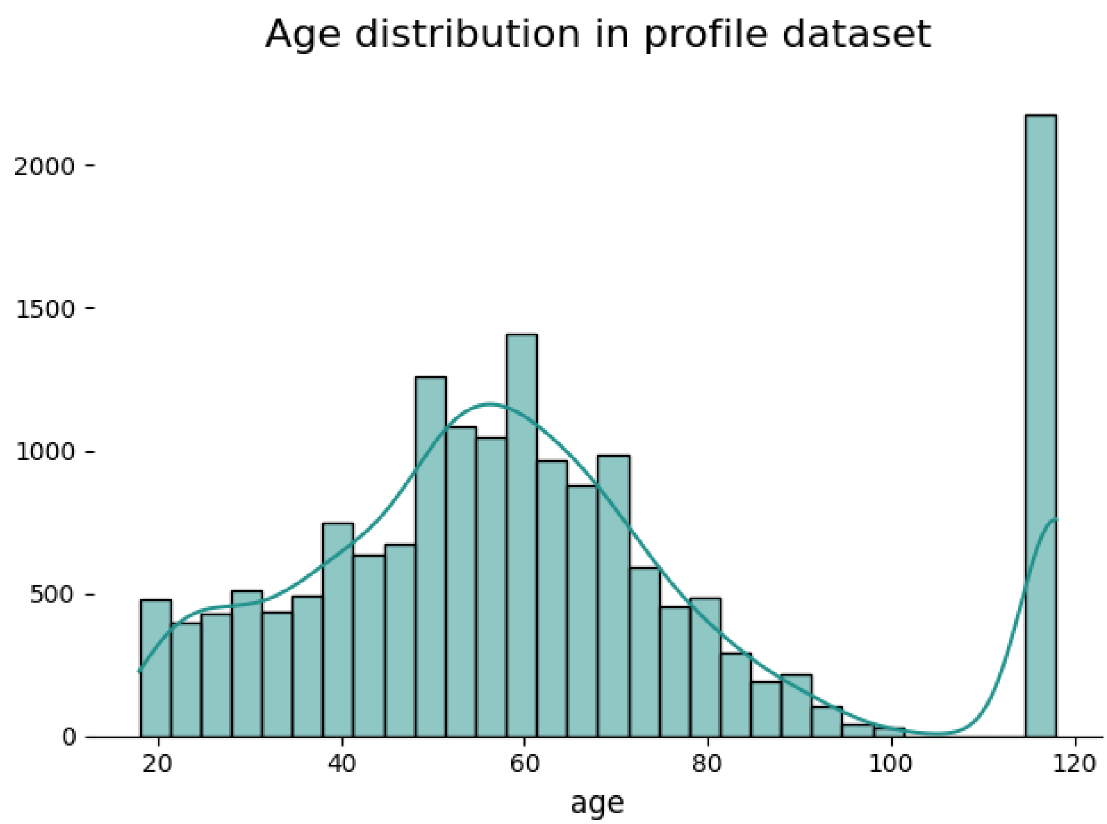
[1168]:
```
column_bar_plot(profile_df, 'gender', 'Gender counts in profile dataset')
```

<Figure size 800x600 with 0 Axes>



[1169]:
```
distribution_plot(profile_df, 'age', 'Age distribution in profile dataset')
```

<Figure size 800x600 with 0 Axes>



A large number of rows have an age value of 118. Upon examining these rows, it is evident that when the age is 118, the gender and income fields are null. I will use this information to clean the dataset later.

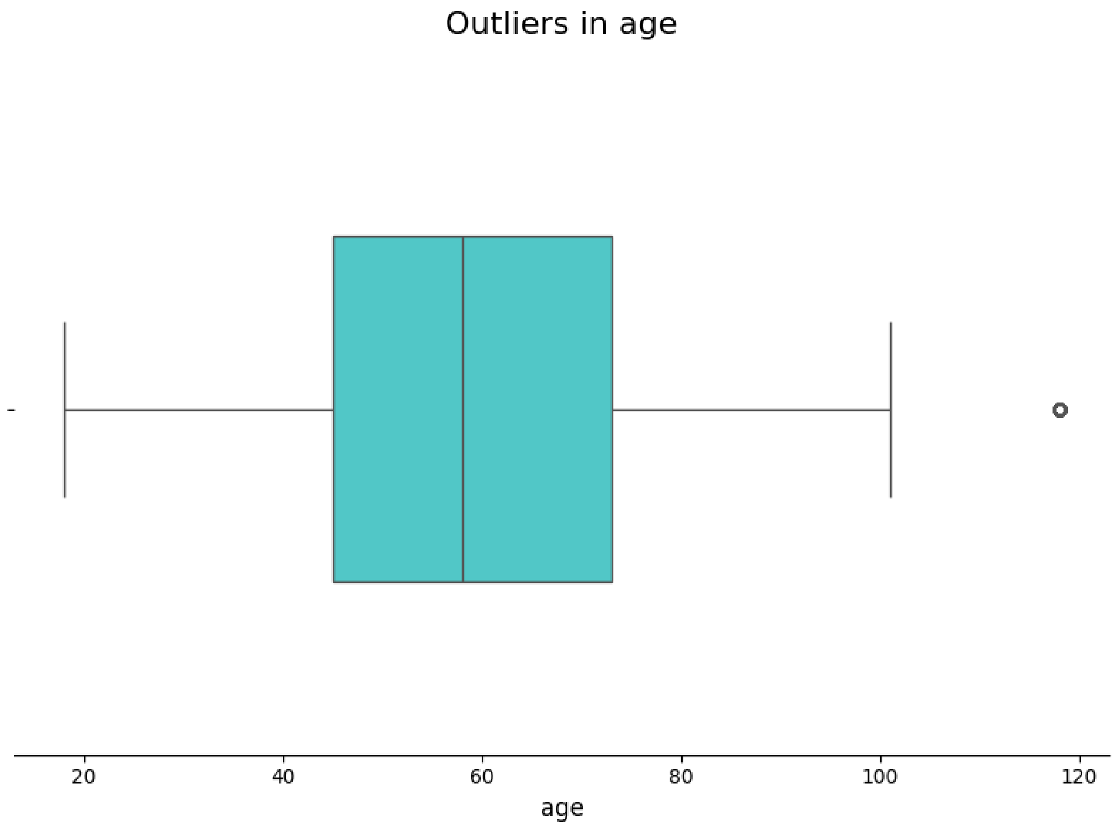
[1170]:
```
profile_df[profile_df['age']> 100]
```

[1170]:

	gender	age	id	became_member_on	income
0	None	118	68be06ca386d4c31939f3a4f0e3dd783	20170212	NaN
2	None	118	38fe809add3b4fcf9315a9694bb96ff5	20180712	NaN
4	None	118	a03223e636434f42ac4c3df47e8bac43	20170804	NaN
6	None	118	8ec6ce2a7e7949b1bf142def7d0e0586	20170925	NaN
7	None	118	68617ca6246f4fbc85e91a2a49552598	20171002	NaN
…	…	…	…	…	…
16980	None	118	5c686d09ca4d475a8f750f2ba07e0440	20160901	NaN
16982	None	118	d9ca82f550ac4ee58b6299cf1e5c824a	20160415	NaN
16989	None	118	ca45ee1883624304bac1e4c8a114f045	20180305	NaN
16991	None	118	a9a20fa8b5504360beb4e7c8712f8306	20160116	NaN
16994	None	118	c02b10e8752c4d8e9b73f918558531f7	20151211	NaN

[2180 rows x 5 columns]

```
[1171]: plot_outliers(profile_df, 'age')
```



Individuals who are older than 80 years seem to exhibit lower engagement with the app, suggesting they might also have lower beverage consumption. Therefore, I classify this age group as outliers in the dataset.

Transcript dataset visualization

```
[1172]: print("the transcript dataset has {0} rows and {1} columns".
          ↪format(str(transcript_df.shape[0]), str(transcript_df.shape[1])))
```

```
the transcript dataset has 306534 rows and 4 columns
```

```
[1173]: transcript_df.describe(include='all')
```

```
[1173]:
                                  person        event  \
count                             306534       306534
unique                             17000            4
top     94de646f7b6041228ca7dec82adb97d2  transaction
freq                                  51       138953
mean                                 NaN          NaN
std                                  NaN          NaN
```

```
min                                         NaN          NaN
25%                                         NaN          NaN
50%                                         NaN          NaN
75%                                         NaN          NaN
max                                         NaN          NaN

                                                      value           time
count                                                306534  306534.000000
unique                                                 5121            NaN
top     {'offer id': '2298d6c36e964ae4a3e7e9706d1fb8c2'}               NaN
freq                                                  14983            NaN
mean                                                    NaN     366.382940
std                                                     NaN     200.326314
min                                                     NaN       0.000000
25%                                                     NaN     186.000000
50%                                                     NaN     408.000000
75%                                                     NaN     528.000000
max                                                     NaN     714.000000
```

[1174]:

```
transcript_df.head()
```

[1174]:

```
                             person           event  \
0  78afa995795e4d85b5d9ceeca43f5fef  offer received
1  a03223e636434f42ac4c3df47e8bac43  offer received
2  e2127556f4f64592b11af22de27a7932  offer received
3  8ec6ce2a7e7949b1bf142def7d0e0586  offer received
4  68617ca6246f4fbc85e91a2a49552598  offer received

                                              value  time
0  {'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}     0
1  {'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}     0
2  {'offer id': '2906b810c7d4411798c6938adc9daaa5'}     0
3  {'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}     0
4  {'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}     0
```

[1175]:

```
transcript_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 306534 entries, 0 to 306533
Data columns (total 4 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   person  306534 non-null  object
 1   event   306534 non-null  object
 2   value   306534 non-null  object
 3   time    306534 non-null  int64
dtypes: int64(1), object(3)
memory usage: 9.4+ MB
```

[1176]:
```
#Check for null values
transcript_df.isnull().sum()
```

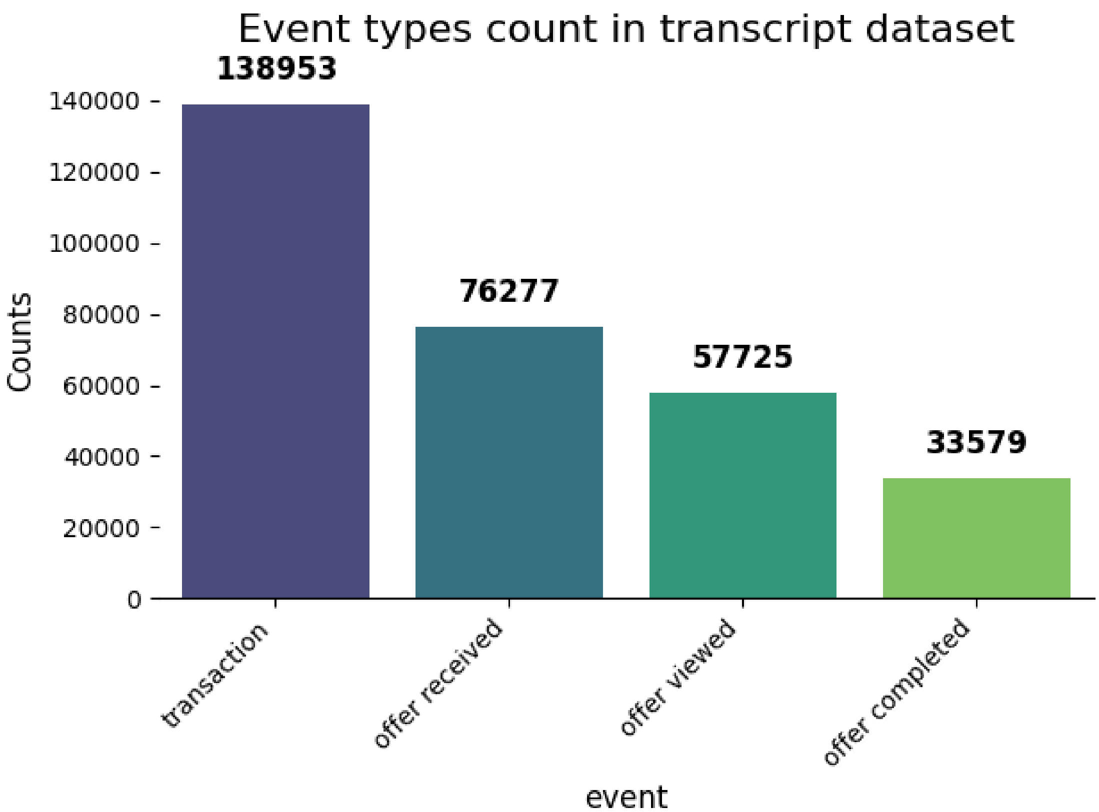
[1176]:
```
person    0
event     0
value     0
time      0
dtype: int64
```

[1177]:
```
from collections import Counter

values = transcript_df['value'].apply(lambda x: frozenset(x.keys()))
values_counts = Counter(values)

combined_key_counts = Counter()

for frozenset_key, count in values_counts.items():
    combined_key = ', '.join(sorted(frozenset_key))
    combined_key_counts[combined_key] += count

print('We have 4 different possibility in the transcript dataset for the value␣
 ↪field:')
for key, count in combined_key_counts.items():
    print(f"'{key}': {count}")
```

```
We have 4 different possibility in the transcript dataset for the value field:
'offer id': 134002
'amount': 138953
'offer_id, reward': 33579
```

[1178]:
```
column_bar_plot(transcript_df, 'event', "Event types count in transcript␣
 ↪dataset")
```

```
<Figure size 800x600 with 0 Axes>
```



Portfolio dataset visualization

[1179]:
```
print("the portfolio dataset has {0} rows and {1} columns".
  ↪format(str(portfolio_df.shape[0]), str(portfolio_df.shape[1])))
```

```
the portfolio dataset has 10 rows and 6 columns
```

[1180]:
```
portfolio_df.describe(include='all')
```

[1180]:

	reward	channels	difficulty	duration \
count	10.000000	10	10.000000	10.000000
unique	NaN	4	NaN	NaN
top	NaN	[web, email, mobile, social]	NaN	NaN
freq	NaN	4	NaN	NaN
mean	4.200000	NaN	7.700000	6.500000
std	3.583915	NaN	5.831905	2.321398
min	0.000000	NaN	0.000000	3.000000
25%	2.000000	NaN	5.000000	5.000000
50%	4.000000	NaN	8.500000	7.000000
75%	5.000000	NaN	10.000000	7.000000
max	10.000000	NaN	20.000000	10.000000

```
       offer_type                                id
count          10                                10
unique          3                                10
top          bogo  ae264e3637204a6fb9bb56bc8210ddfd
freq            4                                 1
mean          NaN                               NaN
std           NaN                               NaN
min           NaN                               NaN
25%           NaN                               NaN
50%           NaN                               NaN
75%           NaN                               NaN
max           NaN                               NaN
```

[1181]:
```
portfolio_df.head()
```

[1181]:
```
   reward                      channels  difficulty  duration     offer_type  \
0      10        [email, mobile, social]          10         7           bogo
1      10  [web, email, mobile, social]          10         5           bogo
2       0           [web, email, mobile]           0         4  informational
3       5           [web, email, mobile]           5         7           bogo
4       5                   [web, email]          20        10       discount

                                 id
0  ae264e3637204a6fb9bb56bc8210ddfd
1  4d5c57ea9a6940dd891ad53e9dbe8da0
2  3f207df678b143eea3cee63160fa8bed
3  9b98b8c7a33c4b65b9aebfe6a799e6d9
4  0b1e1539f2cc45b7b9fa7c272da2e1d7
```

[1182]:
```
portfolio_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   reward      10 non-null     int64
 1   channels    10 non-null     object
 2   difficulty  10 non-null     int64
 3   duration    10 non-null     int64
 4   offer_type  10 non-null     object
 5   id          10 non-null     object
dtypes: int64(3), object(3)
memory usage: 612.0+ bytes
```

[1183]:
```
#Check for null values
portfolio_df.isnull().sum()
```

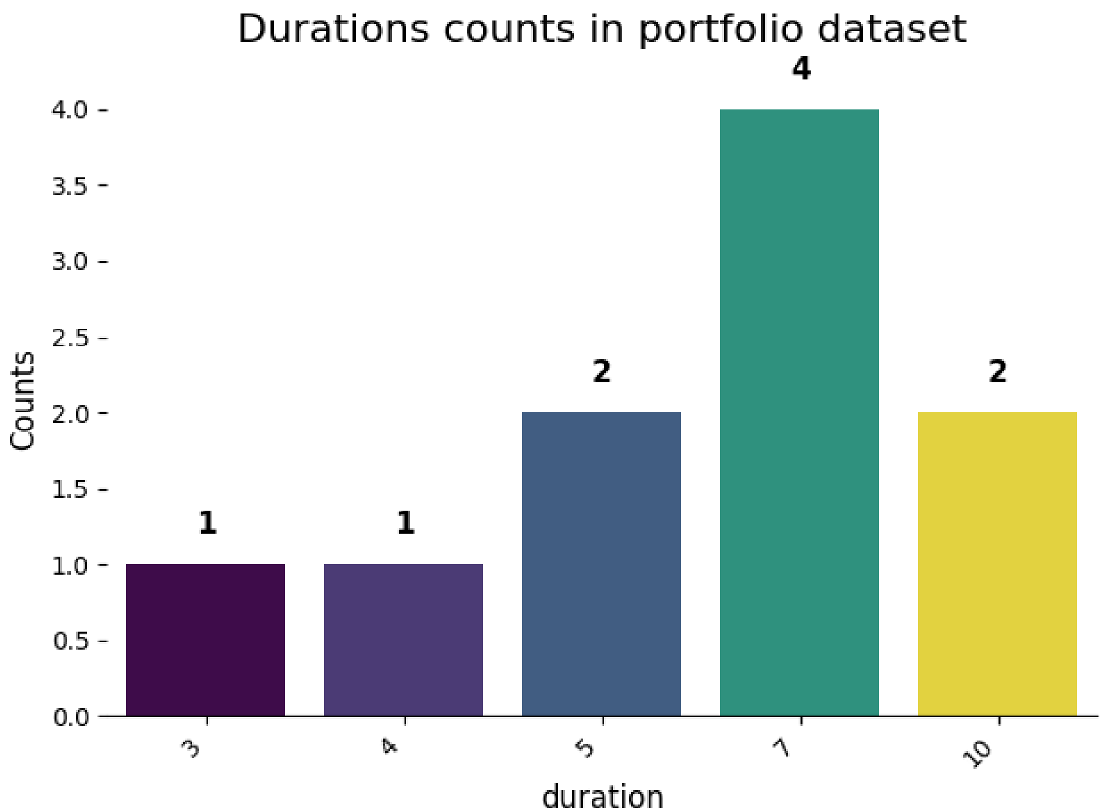
[1183]:
```
reward        0
channels      0
difficulty    0
duration      0
offer_type    0
id            0
dtype: int64
```

[1184]:
```
column_bar_plot(portfolio_df, 'offer_type', 'Offer types counts in portfolio␣
↪dataset')
```

```
<Figure size 800x600 with 0 Axes>
```

[1185]:
```
column_bar_plot(portfolio_df, 'duration', 'Durations counts in portfolio␣
↪dataset')
```

```
<Figure size 800x600 with 0 Axes>
```



1.3.3 Algorithms and Techniques

To predict customer responses to promotional offers, we will use **Random Forest** and **Decision Tree** algorithms.

1. **Random Forest**:
    - **Description**: An ensemble method that builds multiple decision trees and combines their results.
    - **Justification**:
        - **Robustness**: Reduces the risk of overfitting, which is helpful with complex customer data.
        - **Feature Importance**: Identifies which customer traits most influence response to offers.
2. **Decision Tree**:
    - **Description**: A model that splits data into branches based on feature values.
    - **Justification**:
        - **Interpretability**: Easy to understand how decisions are made based on customer data.
        - **Non-linear Relationships**: Captures complex patterns in customer responses.

Data Handling

- **Data Exploration**: We will assess the dataset to identify key features related to customer demographics and offer responses.
- **Data Preprocessing**: This step includes cleaning the data, handling missing values, and encoding categorical variables.
- **Exploratory Data Analysis (EDA)**: We will analyze trends and relationships in the data to inform model building.

1.3.4 Benchmark

We will use a **K-Neighbors Classifier (KNN)** as a benchmark for evaluating our models.

1. **Benchmark Definition**:
   - The KNN model will serve as a baseline, with performance measured using the **F1 score**, which balances precision and recall.
2. **Rationale for Benchmark**:
   - KNN is a simple yet effective algorithm that provides a good starting point for classification tasks.
   - The F1 score is particularly useful for our problem, as it addresses potential imbalances between positive and negative customer responses.
3. **Performance Measurement**:
   - We expect both the Random Forest and Decision Tree models to achieve higher F1 scores than KNN, indicating better predictive accuracy for customer responses.

1.4 III. Methodology

1.4.1 Data Preprocessing

Utlity functions

[1186]:
```
import numpy as np

def fill_missing_values(df, column, method='mean'):
    """
    Fill missing values in a DataFrame column.

    Parameters
    ----------
    df: DataFrame
        The DataFrame to process.
    column: str
        The column name for which to fill missing values.
    method: str
        The method to use for filling: 'mean', 'median', 'mode', or a specific␣
 ↪value.

    Returns
    -------
    df: DataFrame
        DataFrame with filled missing values.
    """
```

```
    if method == 'mean':
        df[column] = df[column].fillna(df[column].mean())
    elif method == 'median':
        df[column] = df[column].fillna(df[column].median())
    elif method == 'mode':
        mode = df[column].mode()[0]
        df[column] = df[column].fillna(mode)
    else:
        df[column] = df[column].fillna(method)
    return df

def remove_outliers(df, column, threshold):
    """
    Remove outliers from a DataFrame based on a threshold.

    Parameters
    ----------
    df: DataFrame
        The DataFrame to process.
    column: str
        The column name to check for outliers.
    threshold: float
        The upper limit to consider for outliers.

    Returns
    -------
    df: DataFrame
        DataFrame with outliers removed.
    """
    return df[df[column] <= threshold]

def expand_dict_column(df, column, new_columns):
    """
    Expand a dictionary-type column into separate columns, accommodating␣
 ↪different key variations.

    Parameters
    ----------
    df: DataFrame
        The DataFrame to process.
    column: str
        The name of the column containing dictionaries.
    new_columns: dict
        A dictionary where keys are new column names and values are the list of␣
 ↪keys to extract from the dictionary.

    Returns
```

```
    -------
    df: DataFrame
        DataFrame with new columns added from the dictionary.
    """
    for new_col, old_keys in new_columns.items():
        df[new_col] = df[column].apply(lambda x: next((x.get(k) for k in␣
↪old_keys if isinstance(x, dict) and k in x), 0))
    return df

def rename_cols(df, new_cols_name):
    """
    Rename columns of a DataFrame using a given mapping.

    Parameters
    ----------
    df: DataFrame
        The DataFrame whose columns need to be renamed.
    new_cols_name: dict
        A dictionary where keys are the existing column names and values are␣
↪the new column names.

    Returns
    -------
    df: DataFrame
        DataFrame with renamed columns.
    """
    df.rename(columns=new_cols_name, inplace=True)
    return df
```

Cleaning profile dataset

- To retain as much data as possible, impute missing values: use the mean for age and income, and the mode for gender.

[1187]:
```
profile_df.replace({'age': {118: np.nan}}, inplace=True)
profile_df = fill_missing_values(profile_df, 'age', method='mean')
profile_df = fill_missing_values(profile_df, 'income', method='mean')
profile_df = fill_missing_values(profile_df, 'gender', method='mode')
```

the profile dataframe has no more null values

[1188]:
```
profile_df.isnull().sum()
```

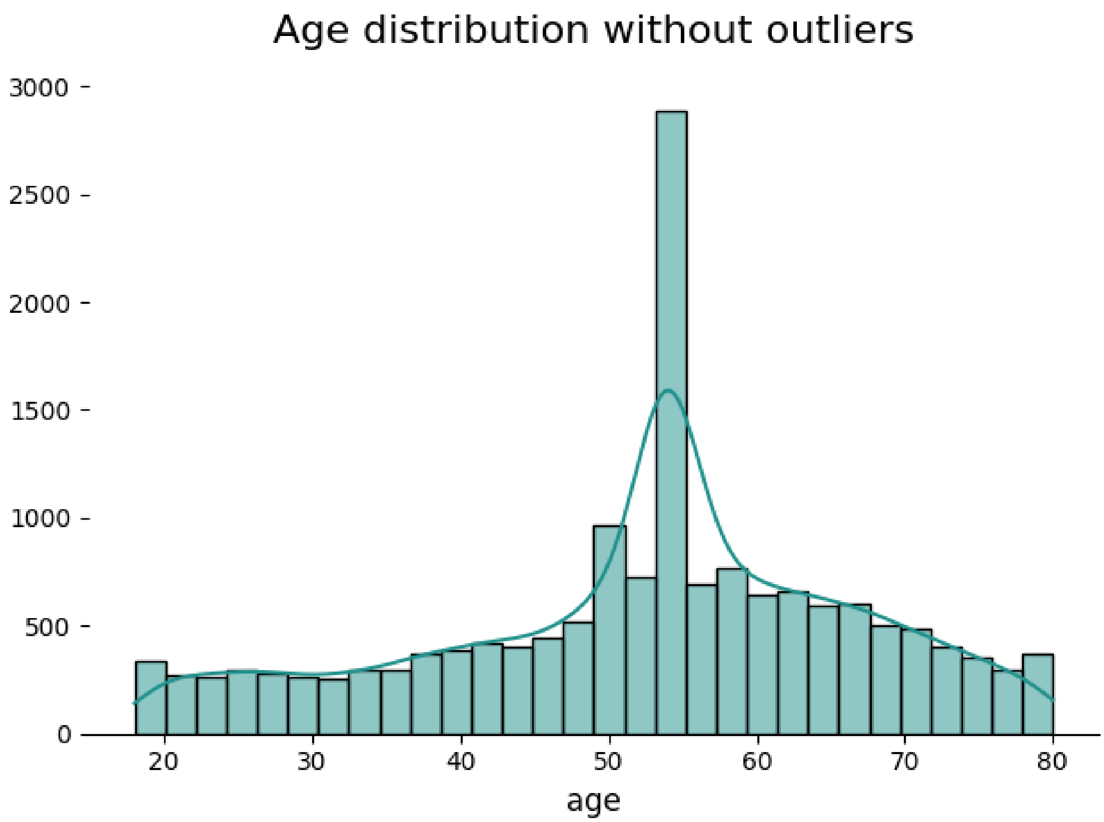
[1188]:
```
gender              0
age                 0
id                  0
became_member_on    0
income              0
```

```
dtype: int64
```

- Treat individuals over the age of 80 as outliers as emerged from the exploratory phase and exclude them from the dataset.

```
[1189]: profile_df = remove_outliers(profile_df, 'age', threshold=80)
profile_df.loc[:, 'age'] = profile_df['age'].astype(int)
distribution_plot(profile_df, 'age', 'Age distribution without outliers')
```

```
<Figure size 800x600 with 0 Axes>
```



- Categorize ages into groups for better clarity during Exploratory Data Analysis (EDA):
  - Under 20
  - 20 - 45
  - 46 - 60
  - 61 - 80

```
[1190]: profile_df.loc[(profile_df['age'] < 20), 'age_group'] = 'Under 20'
profile_df.loc[(profile_df['age'] >= 20) & (profile_df['age'] <= 45),␣
    ↪'age_group'] = '20-45'
profile_df.loc[(profile_df['age'] >= 46) & (profile_df['age'] <= 60),␣
    ↪'age_group'] = '46-60'
```

```
profile_df.loc[(profile_df['age'] >= 61), 'age_group'] = '61-80'
profile_df.drop('age', axis=1, inplace=True)
```

- Rename columns to improve readability and facilitate merging of dataframes.

[1191]:
```
cleaned_profile_df = rename_cols(profile_df, {'id':'customer_id' , 'income':
 ↪'customer_income'} )
#cleaned_profile_df['customer_id'].count()
cleaned_profile_df.head(10)
```

[1191]:
```
  gender                       customer_id  became_member_on  customer_income  \
0      M  68be06ca386d4c31939f3a4f0e3dd783          20170212     65404.991568
1      F  0610b486422d4921ae7d2bf64640c50b          20170715    112000.000000
2      M  38fe809add3b4fcf9315a9694bb96ff5          20180712     65404.991568
3      F  78afa995795e4d85b5d9ceeca43f5fef          20170509    100000.000000
4      M  a03223e636434f42ac4c3df47e8bac43          20170804     65404.991568
5      M  e2127556f4f64592b11af22de27a7932          20180426     70000.000000
6      M  8ec6ce2a7e7949b1bf142def7d0e0586          20170925     65404.991568
7      M  68617ca6246f4fbc85e91a2a49552598          20171002     65404.991568
8      M  389bc3fa690240e798340f5a15918d5c          20180209     53000.000000
9      M  8974fc5686fe429db53ddde067b88302          20161122     65404.991568

  age_group
0     46-60
1     46-60
2     46-60
3     61-80
4     46-60
5     61-80
6     46-60
7     46-60
8     61-80
9     46-60
```

Clening transcript dataset

- Expand the nested keys in the 'value' column into separate new columns.

[1192]:
```
transcript_df['value'].value_counts()
```

[1192]:
```
value
{'offer id': '2298d6c36e964ae4a3e7e9706d1fb8c2'}    14983
{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}    14924
{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}    14891
{'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}    14835
{'offer id': 'ae264e3637204a6fb9bb56bc8210ddfd'}    14374
                                                    …
{'amount': 42.31}                                       1
```

```
{'amount': 44.62}                                           1
{'amount': 42.27}                                           1
{'amount': 108.89}                                          1
{'amount': 476.33}                                          1
Name: count, Length: 5121, dtype: int64
```

[1193]:
```
transcript_df = expand_dict_column(transcript_df, 'value',
    {
        "offer_id": ["offer id", "offer_id"],
        "money_gained": ["reward"],
        "money_spent": ["amount"]
    })
transcript_df.drop(['value'], axis=1, inplace=True)
```

The "value" column contains dictionaries, with each key from these dictionaries separated into its own column.

[1194]:
```
transcript_df.head()
```

[1194]:
```
                             person           event  time  \
0  78afa995795e4d85b5d9ceeca43f5fef  offer received     0
1  a03223e636434f42ac4c3df47e8bac43  offer received     0
2  e2127556f4f64592b11af22de27a7932  offer received     0
3  8ec6ce2a7e7949b1bf142def7d0e0586  offer received     0
4  68617ca6246f4fbc85e91a2a49552598  offer received     0

                           offer_id  money_gained  money_spent
0  9b98b8c7a33c4b65b9aebfe6a799e6d9             0          0.0
1  0b1e1539f2cc45b7b9fa7c272da2e1d7             0          0.0
2  2906b810c7d4411798c6938adc9daaa5             0          0.0
3  fafdcd668e3743c1bb461111dcafc2a4             0          0.0
4  4d5c57ea9a6940dd891ad53e9dbe8da0             0          0.0
```

- Rename columns to enhance readability and simplify the process of merging dataframes.

[1195]:
```
cleaned_transcript_df = rename_cols(transcript_df, {'person':'customer_id'})
cleaned_transcript_df.head()
```

[1195]:
```
                        customer_id           event  time  \
0  78afa995795e4d85b5d9ceeca43f5fef  offer received     0
1  a03223e636434f42ac4c3df47e8bac43  offer received     0
2  e2127556f4f64592b11af22de27a7932  offer received     0
3  8ec6ce2a7e7949b1bf142def7d0e0586  offer received     0
4  68617ca6246f4fbc85e91a2a49552598  offer received     0

                           offer_id  money_gained  money_spent
0  9b98b8c7a33c4b65b9aebfe6a799e6d9             0          0.0
1  0b1e1539f2cc45b7b9fa7c272da2e1d7             0          0.0
```

```
2  2906b810c7d4411798c6938adc9daaa5          0          0.0
3  fafdcd668e3743c1bb461111dcafc2a4          0          0.0
4  4d5c57ea9a6940dd891ad53e9dbe8da0          0          0.0
```

Cleaning portfolio dataset

- Rename columns to enhance readability and simplify the process of merging dataframes.

[1196]:
```
portfolio_df.head()
```

[1196]:
```
   reward                      channels  difficulty  duration     offer_type  \
0      10       [email, mobile, social]          10         7           bogo
1      10  [web, email, mobile, social]          10         5           bogo
2       0          [web, email, mobile]           0         4  informational
3       5          [web, email, mobile]           5         7           bogo
4       5                  [web, email]          20        10       discount

                                 id
0  ae264e3637204a6fb9bb56bc8210ddfd
1  4d5c57ea9a6940dd891ad53e9dbe8da0
2  3f207df678b143eea3cee63160fa8bed
3  9b98b8c7a33c4b65b9aebfe6a799e6d9
4  0b1e1539f2cc45b7b9fa7c272da2e1d7
```

[1197]:
```
cleaned_portfolio_df = rename_cols(portfolio_df, {'difficulty':
↪'offer_difficulty' , 'id':'offer_id', 'duration':'offer_duration', 'reward':␣
↪'offer_reward'})
cleaned_portfolio_df.head()
```

[1197]:
```
   offer_reward                      channels  offer_difficulty  \
0            10       [email, mobile, social]                10
1            10  [web, email, mobile, social]                10
2             0          [web, email, mobile]                 0
3             5          [web, email, mobile]                 5
4             5                  [web, email]                20

   offer_duration     offer_type                          offer_id
0               7           bogo  ae264e3637204a6fb9bb56bc8210ddfd
1               5           bogo  4d5c57ea9a6940dd891ad53e9dbe8da0
2               4  informational  3f207df678b143eea3cee63160fa8bed
3               7           bogo  9b98b8c7a33c4b65b9aebfe6a799e6d9
4              10       discount  0b1e1539f2cc45b7b9fa7c272da2e1d7
```

Merging the dataframes

[1213]:
```
dataframe = pd.merge(cleaned_portfolio_df, cleaned_transcript_df, on='offer_id')
dataframe = pd.merge(dataframe, cleaned_profile_df, on='customer_id')
dataframe.head(10)
```

```
[1213]:    offer_reward                  channels  offer_difficulty  offer_duration  \
0            10  [email, mobile, social]                10               7
1            10  [email, mobile, social]                10               7
2            10  [email, mobile, social]                10               7
3            10  [email, mobile, social]                10               7
4            10  [email, mobile, social]                10               7
5            10  [email, mobile, social]                10               7
6            10  [email, mobile, social]                10               7
7            10  [email, mobile, social]                10               7
8            10  [email, mobile, social]                10               7
9            10  [email, mobile, social]                10               7

  offer_type                          offer_id  \
0       bogo  ae264e3637204a6fb9bb56bc8210ddfd
1       bogo  ae264e3637204a6fb9bb56bc8210ddfd
2       bogo  ae264e3637204a6fb9bb56bc8210ddfd
3       bogo  ae264e3637204a6fb9bb56bc8210ddfd
4       bogo  ae264e3637204a6fb9bb56bc8210ddfd
5       bogo  ae264e3637204a6fb9bb56bc8210ddfd
6       bogo  ae264e3637204a6fb9bb56bc8210ddfd
7       bogo  ae264e3637204a6fb9bb56bc8210ddfd
8       bogo  ae264e3637204a6fb9bb56bc8210ddfd
9       bogo  ae264e3637204a6fb9bb56bc8210ddfd

                        customer_id           event  time  money_gained  \
0  4b0da7e80e5945209a1fdddfe813dbe0  offer received     0             0
1  1e9420836d554513ab90eba98552d0a9  offer received     0             0
2  02c083884c7d45b39cc68e1314fec56c  offer received     0             0
3  676506bad68e4161b9bbaffeb039626b  offer received     0             0
4  fe8264108d5b4f198453bbb1fa7ca6c9  offer received     0             0
5  39dbcf43e24d41f4bbf0f134157e0e1e  offer received     0             0
6  3f244f4dea654688ace14acb4f0257bb  offer received     0             0
7  92e07c49ee7448fca6e48df0c96e3eec  offer received     0             0
8  f8aedd0cbea0419c806842b4265b82e5  offer received     0             0
9  8a4bc602e4424ab6b16f0b907f2f22af  offer received     0             0

   money_spent gender  became_member_on  customer_income age_group
0          0.0      M          20170909         100000.0     61-80
1          0.0      M          20170925          70000.0     20-45
2          0.0      F          20160711          30000.0     20-45
3          0.0      M          20170515          92000.0     20-45
4          0.0      F          20161009          93000.0     61-80
5          0.0      M          20140831          64000.0     61-80
6          0.0      M          20180703          71000.0     61-80
7          0.0      F          20180216          58000.0     61-80
8          0.0      F          20160811          72000.0     61-80
9          0.0      M          20171210          31000.0     46-60
```

1.4.2 Implementation

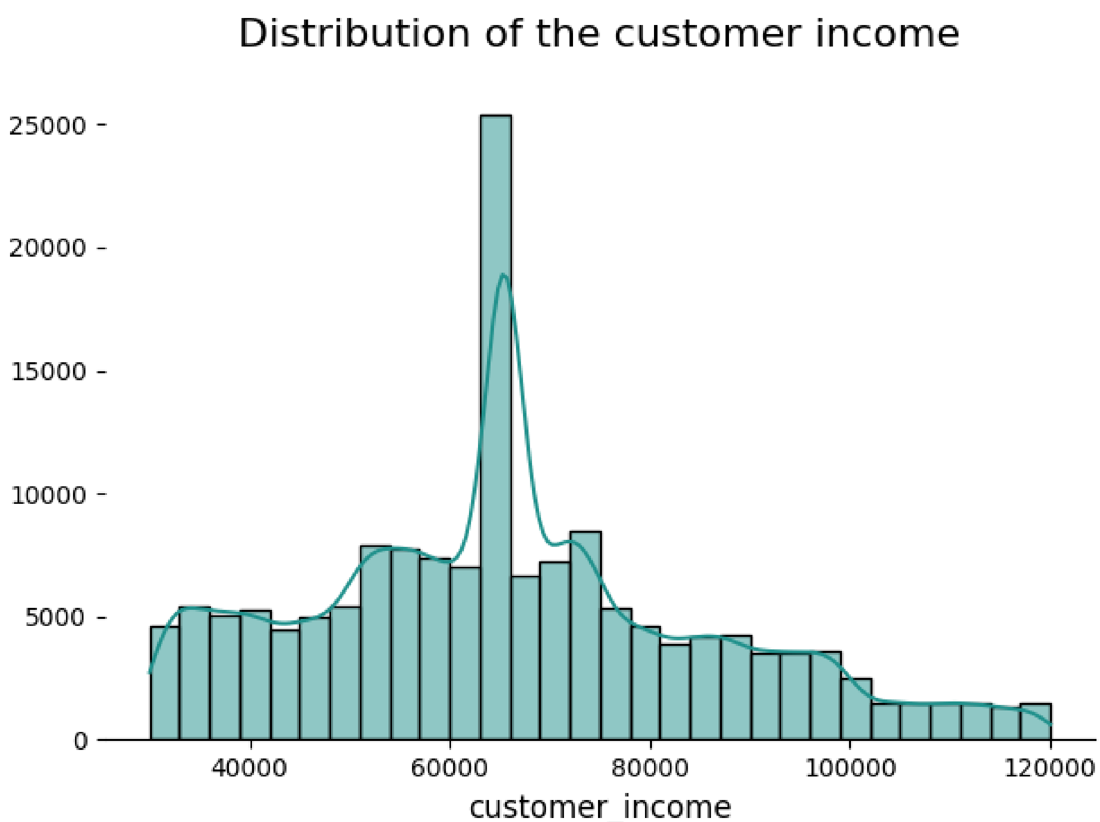
Exploratory data analysis (EDA)

[1214]:
```
average_income = float(dataframe['customer_income'].mean())
print(f'The average income of customers is: {average_income}')
```

```
The average income of customers is: 65924.49109976534
```

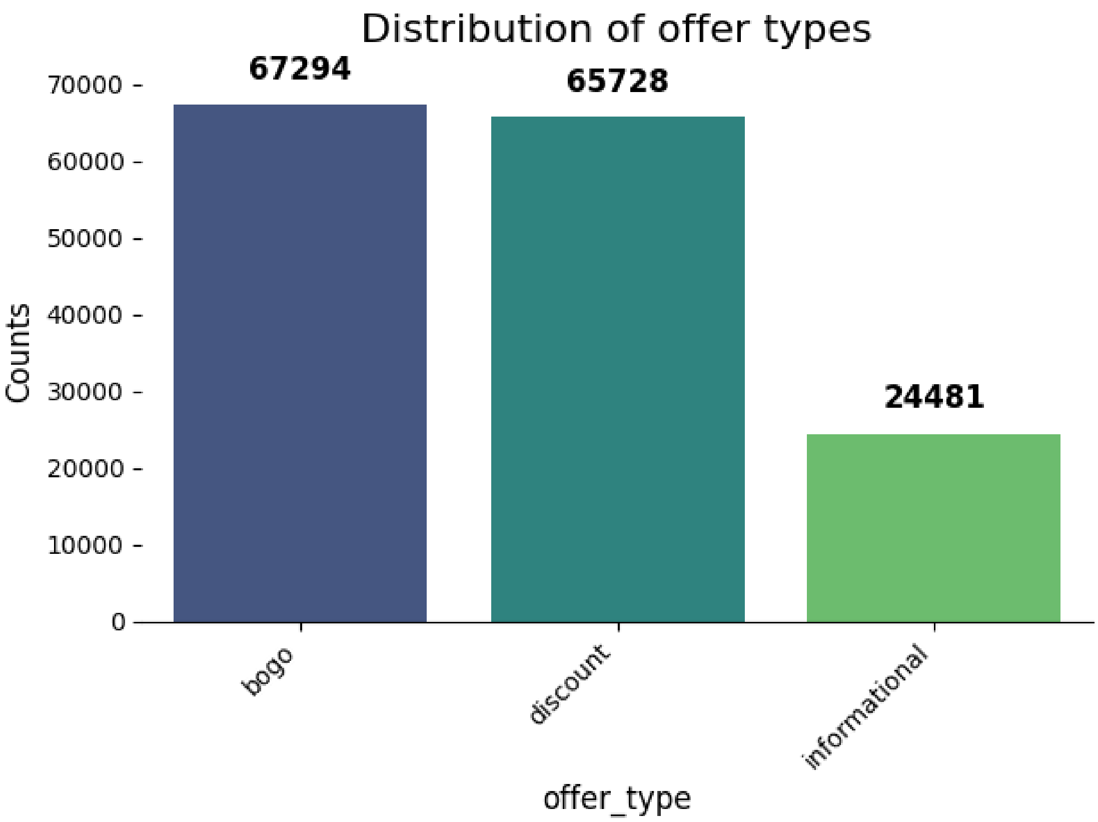
[1215]:
```
distribution_plot(dataframe, 'customer_income', 'Distribution of the customer␣
↪income')
```

```
<Figure size 800x600 with 0 Axes>
```



[1216]:
```
column_bar_plot(dataframe, 'offer_type', 'Distribution of offer types')
```

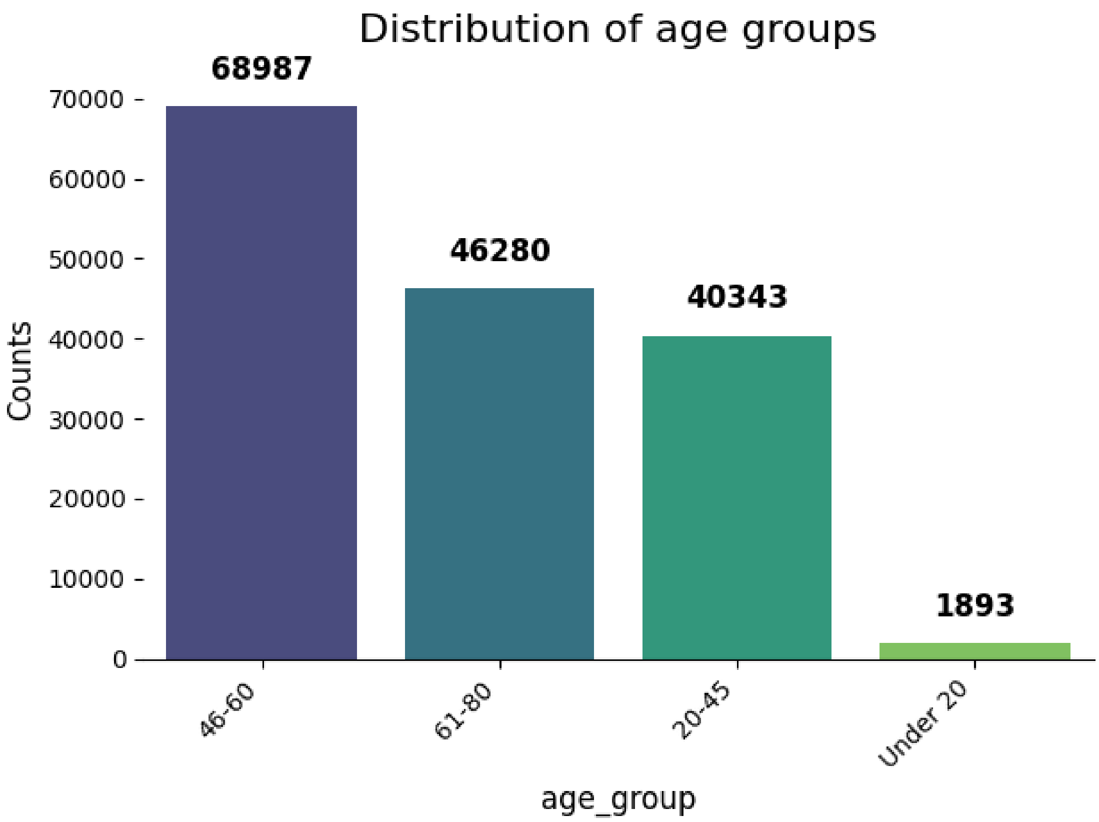
```
<Figure size 800x600 with 0 Axes>
```



The majority of the offers are BOGO and Discount.

[1217]:
```
column_bar_plot(dataframe, 'age_group', 'Distribution of age groups')
```

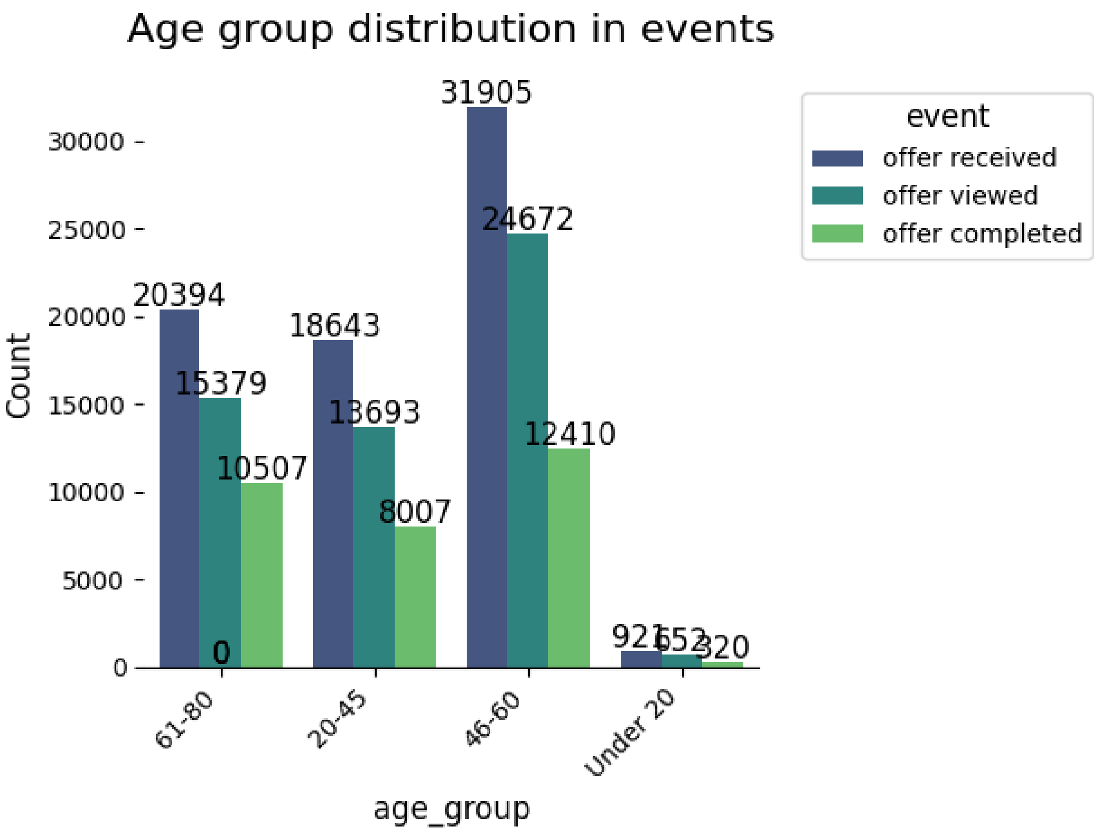
```
<Figure size 800x600 with 0 Axes>
```



Contrary to common expectations, the Starbucks app is most popular among users aged 46-60, with those aged 61-80 coming in second. Surprisingly, the younger demographic of 20-45, who are often assumed to be the primary app users, do not dominate usage in this instance.

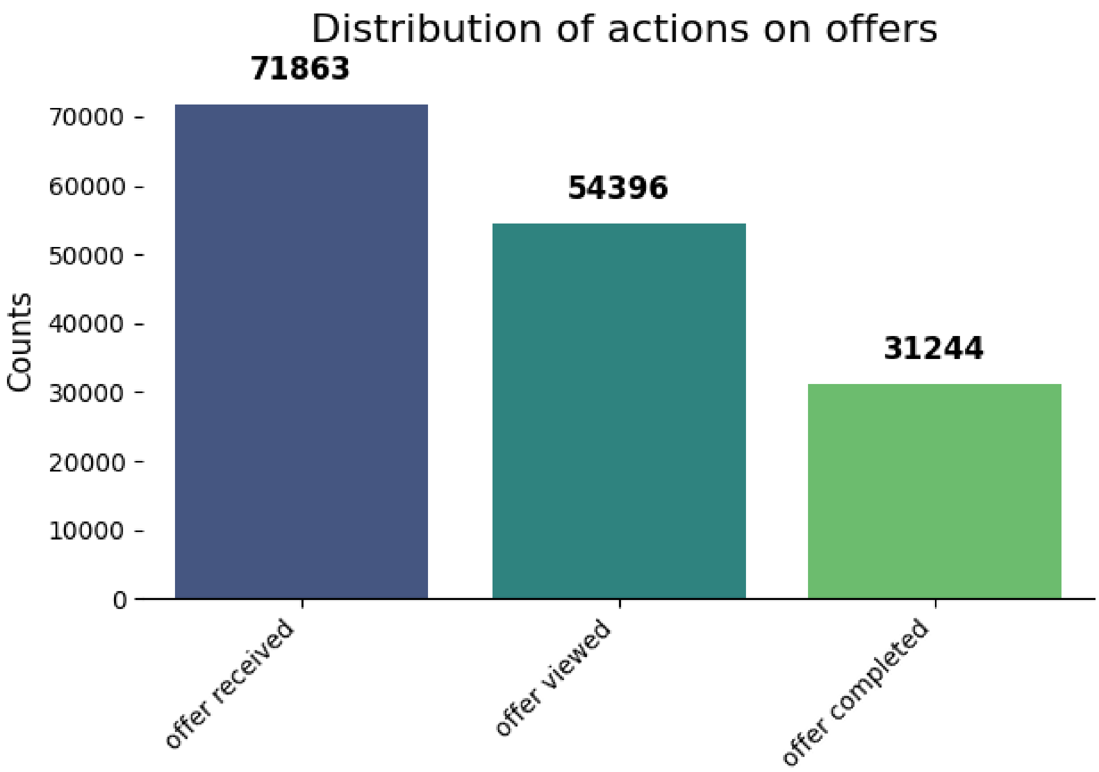
[1218]:
```
grouped_bar_plot(dataframe, 'age_group', 'event', 'Age group distribution in␣
↪events')
```

```
<Figure size 1500x500 with 0 Axes>
```



[1219]:
```
column_bar_plot(dataframe, 'event', ' Distribution of actions on offers')
```

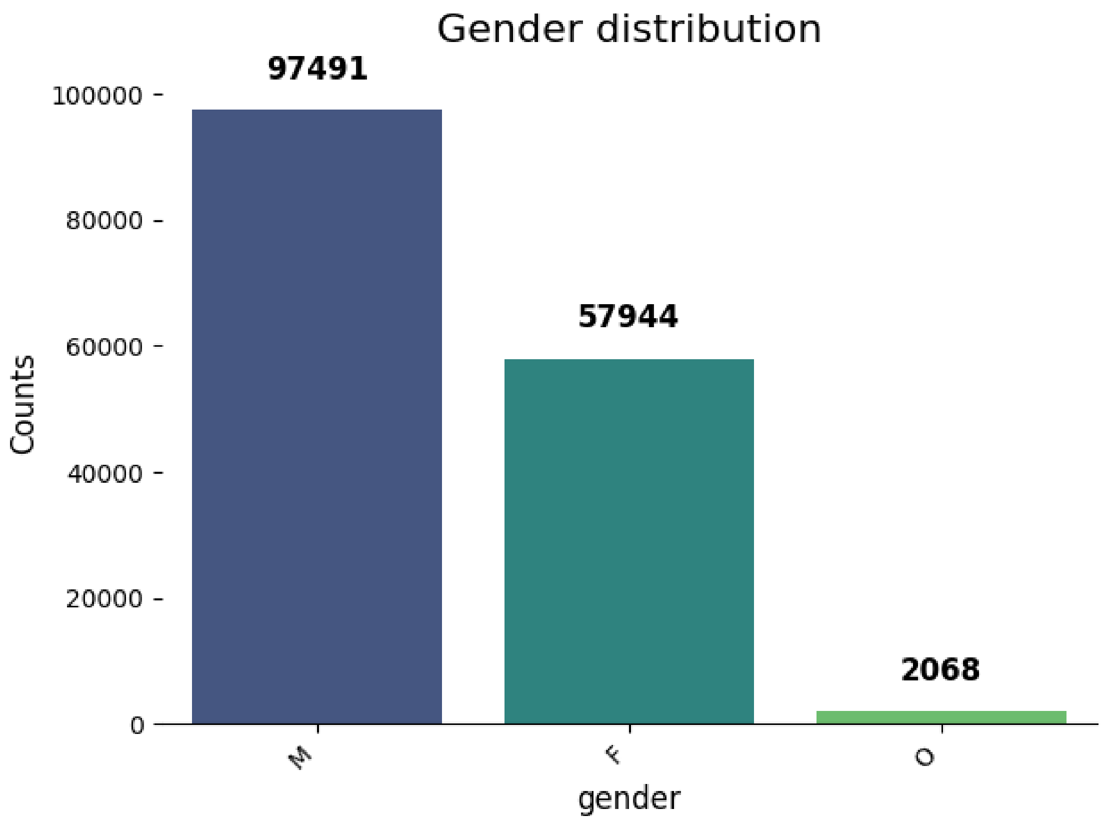
```
<Figure size 800x600 with 0 Axes>
```



This suggests that the majority of customers disregard the offer entirely, not even taking a moment to review it. Additionally, more customers simply view and dismiss the offer compared to those who proceed to complete it.

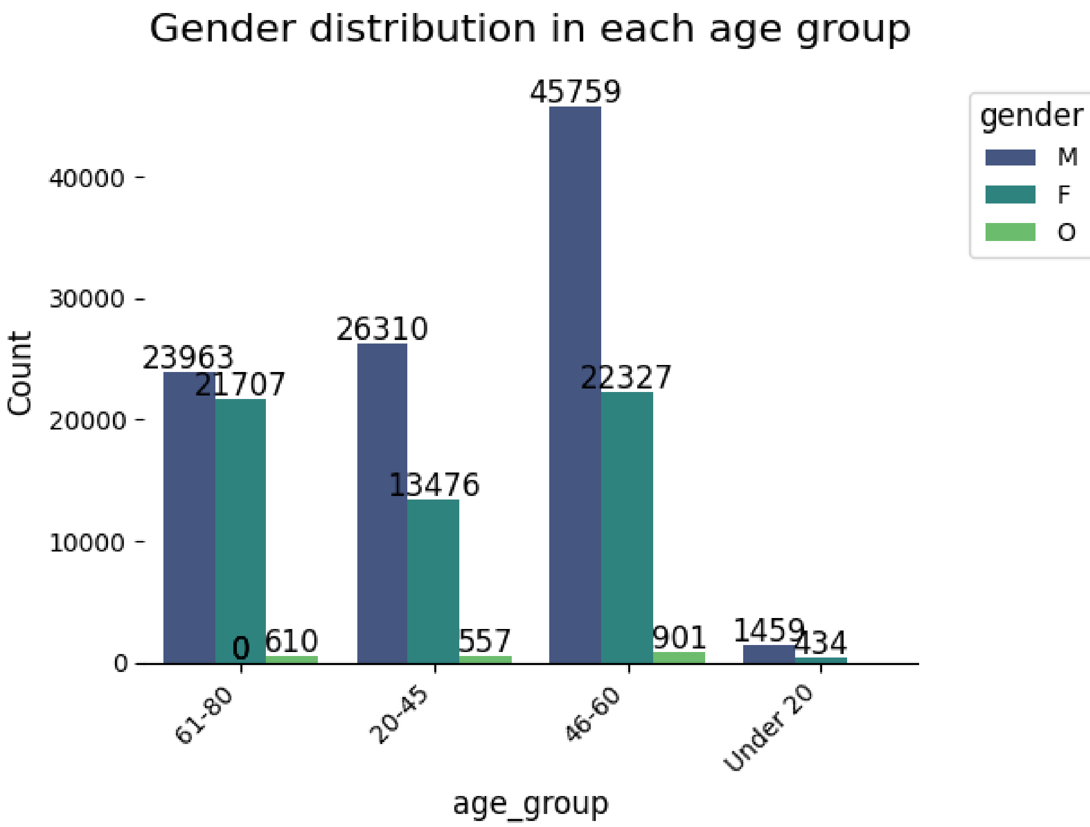
[1220]:
```
column_bar_plot(dataframe, 'gender', 'Gender distribution')
```

```
<Figure size 800x600 with 0 Axes>
```



[1221]:
```
grouped_bar_plot(dataframe, 'age_group', 'gender', 'Gender distribution in each␣
↪age group')
```

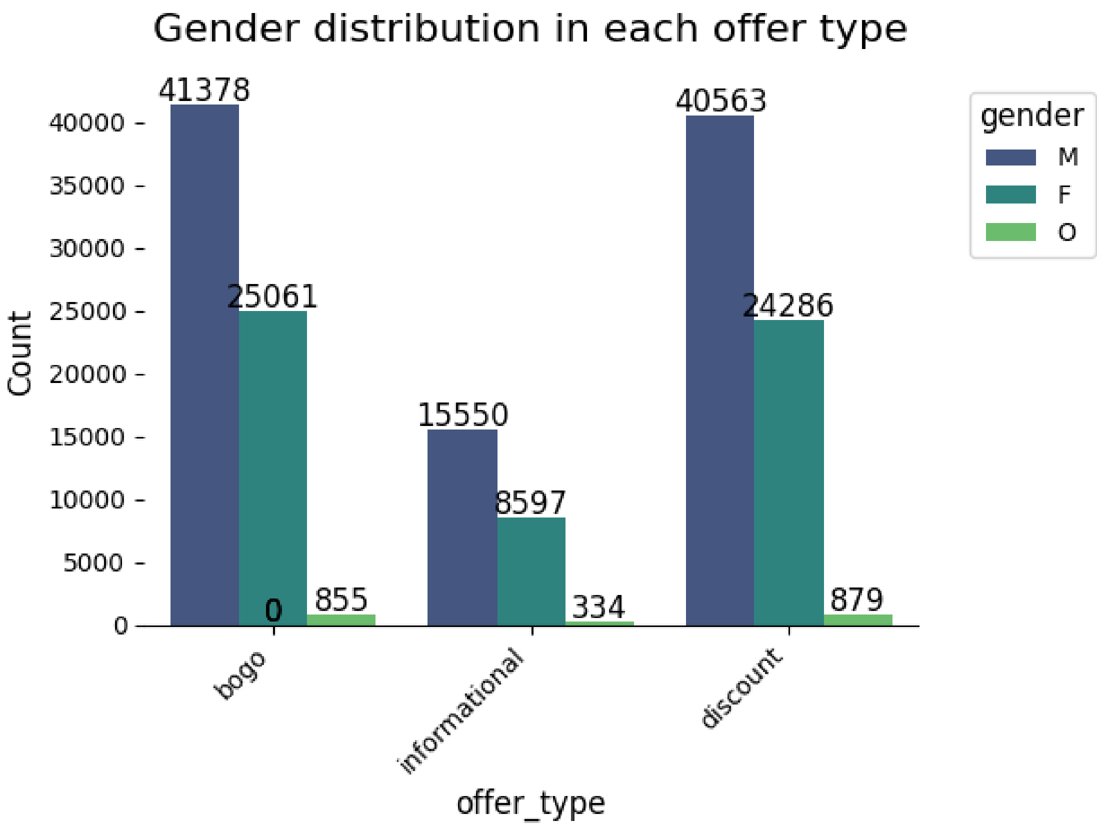
```
<Figure size 1500x500 with 0 Axes>
```



In every age group, there are more male customers than female customers

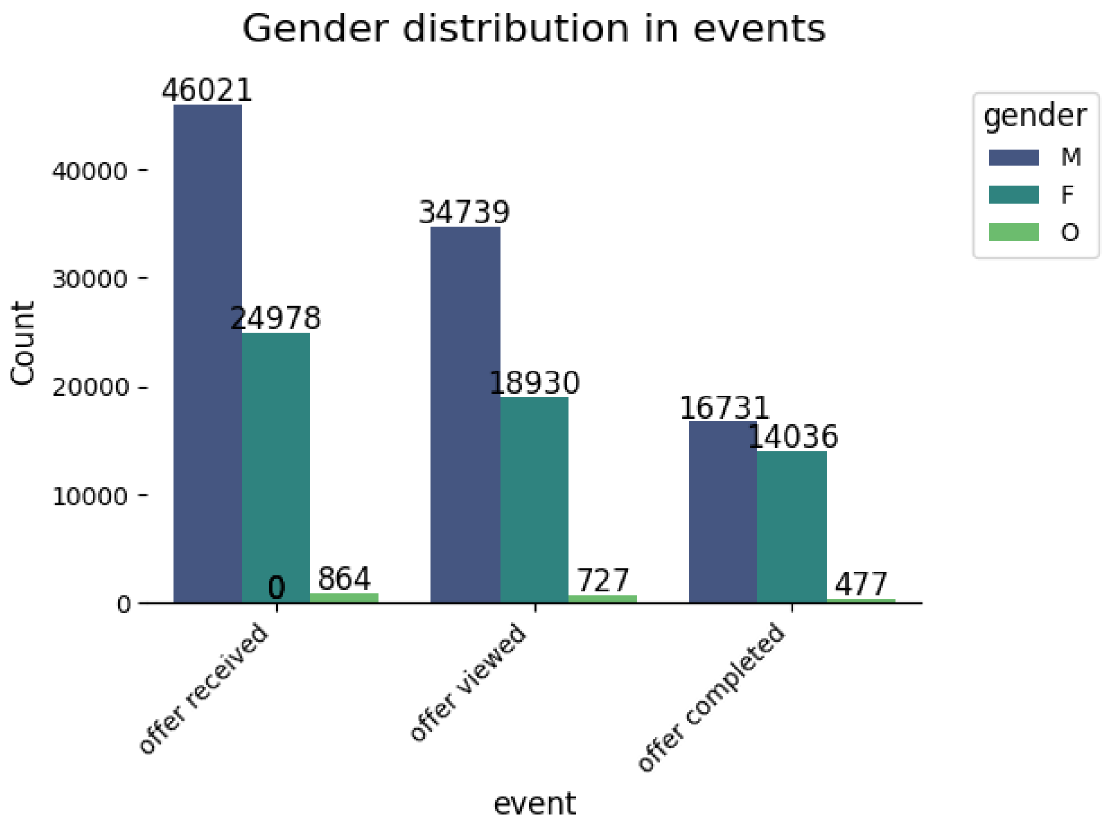
[1222]:
```
grouped_bar_plot(dataframe, 'offer_type', 'gender', 'Gender distribution in␣
↪each offer type')
```

```
<Figure size 1500x500 with 0 Axes>
```



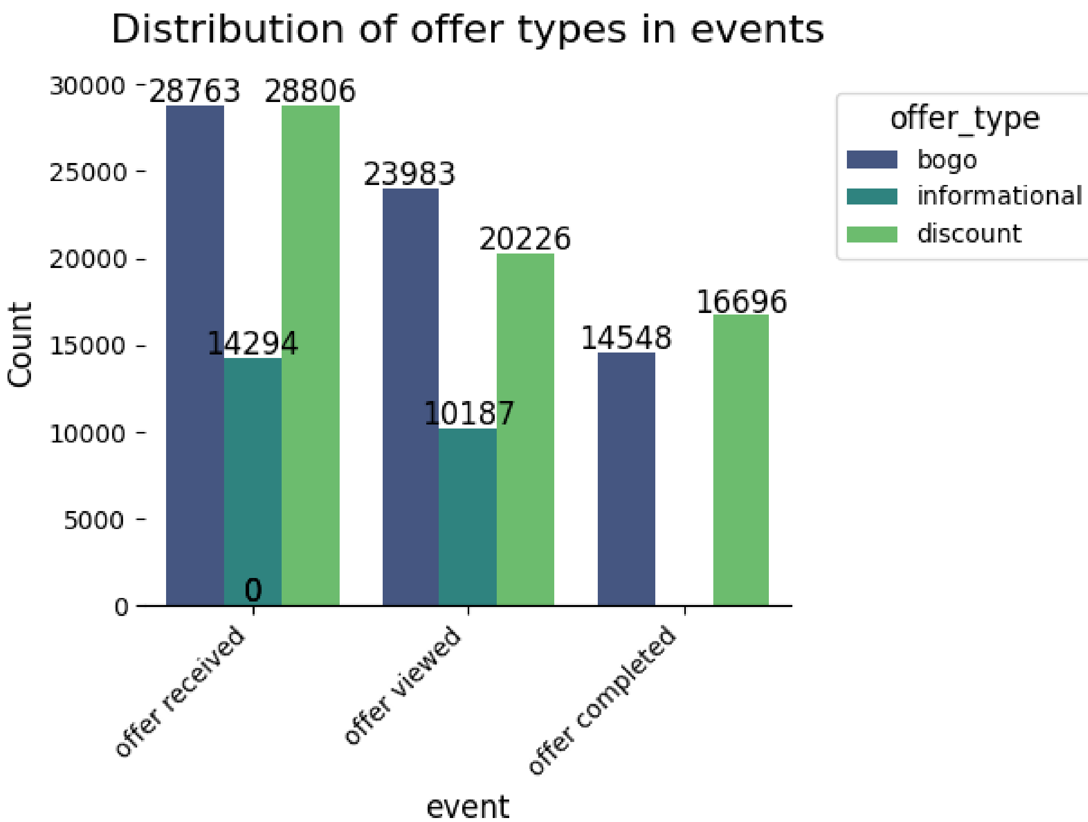
[1223]:
```
grouped_bar_plot(dataframe, 'event', 'gender', 'Gender distribution in events')
```

```
<Figure size 1500x500 with 0 Axes>
```



[1224]:
```
grouped_bar_plot(dataframe, 'event', 'offer_type', 'Distribution of offer types␣
↪in events')
```

```
<Figure size 1500x500 with 0 Axes>
```



Overall, the majority of people tend to take advantage of the discount offer.

- Observations

Males account for 62.7% of the data and tend to use the Starbucks app more frequently than females. Notably, both males and females in the 46-60 age group are the heaviest users of the app. Customers show a stronger preference for discount offers. However, there is a lower number of customers who actually complete offers compared to those who simply view and ignore them.

Training data preparation

- One-hot-encoding of columns with categorical values.

[1225]:
```
categorical_columns = ['gender', 'offer_type', 'age_group']
dataframe = pd.get_dummies(dataframe, columns = categorical_columns)
dataframe.head()
```

[1225]:
```
   offer_reward                channels  offer_difficulty  offer_duration  \
0            10  [email, mobile, social]                10               7
1            10  [email, mobile, social]                10               7
2            10  [email, mobile, social]                10               7
3            10  [email, mobile, social]                10               7
```

```
4           10  [email, mobile, social]                10               7

                             offer_id                       customer_id  \
0  ae264e3637204a6fb9bb56bc8210ddfd  4b0da7e80e5945209a1fdddfe813dbe0
1  ae264e3637204a6fb9bb56bc8210ddfd  1e9420836d554513ab90eba98552d0a9
2  ae264e3637204a6fb9bb56bc8210ddfd  02c083884c7d45b39cc68e1314fec56c
3  ae264e3637204a6fb9bb56bc8210ddfd  676506bad68e4161b9bbaffeb039626b
4  ae264e3637204a6fb9bb56bc8210ddfd  fe8264108d5b4f198453bbb1fa7ca6c9

            event  time  money_gained  money_spent  …  gender_F  gender_M  \
0  offer received     0             0          0.0  …     False      True
1  offer received     0             0          0.0  …     False      True
2  offer received     0             0          0.0  …      True     False
3  offer received     0             0          0.0  …     False      True
4  offer received     0             0          0.0  …      True     False

   gender_O  offer_type_bogo  offer_type_discount  offer_type_informational  \
0     False             True                False                     False
1     False             True                False                     False
2     False             True                False                     False
3     False             True                False                     False
4     False             True                False                     False

   age_group_20-45  age_group_46-60  age_group_61-80  age_group_Under 20
0            False            False             True               False
1             True            False            False               False
2             True            False            False               False
3             True            False            False               False
4            False            False             True               False

[5 rows x 22 columns]
```

- Encode the 'event' data with numerical values.

[1226]:
```
dataframe['event'] = dataframe['event'].replace({
    'offer received': 1,
    'offer viewed': 2,
    'offer completed': 3
})

dataframe.head()
```

```
/var/folders/1j/735j19ws2457_nw6f66bm2rr0000gn/T/ipykernel_86987/1763579747.py:1
: FutureWarning: Downcasting behavior in `replace` is deprecated and will be
removed in a future version. To retain the old behavior, explicitly call
`result.infer_objects(copy=False)`. To opt-in to the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  dataframe['event'] = dataframe['event'].replace({
```

```
[1226]:    offer_reward                 channels  offer_difficulty  offer_duration  \
        0            10  [email, mobile, social]                10               7
        1            10  [email, mobile, social]                10               7
        2            10  [email, mobile, social]                10               7
        3            10  [email, mobile, social]                10               7
        4            10  [email, mobile, social]                10               7

                                   offer_id                       customer_id  event  \
        0  ae264e3637204a6fb9bb56bc8210ddfd  4b0da7e80e5945209a1fdddfe813dbe0      1
        1  ae264e3637204a6fb9bb56bc8210ddfd  1e9420836d554513ab90eba98552d0a9      1
        2  ae264e3637204a6fb9bb56bc8210ddfd  02c083884c7d45b39cc68e1314fec56c      1
        3  ae264e3637204a6fb9bb56bc8210ddfd  676506bad68e4161b9bbaffeb039626b      1
        4  ae264e3637204a6fb9bb56bc8210ddfd  fe8264108d5b4f198453bbb1fa7ca6c9      1

           time  money_gained  money_spent  …  gender_F  gender_M  gender_O  \
        0     0             0          0.0  …     False      True     False
        1     0             0          0.0  …     False      True     False
        2     0             0          0.0  …      True     False     False
        3     0             0          0.0  …     False      True     False
        4     0             0          0.0  …      True     False     False

           offer_type_bogo  offer_type_discount  offer_type_informational  \
        0             True                False                     False
        1             True                False                     False
        2             True                False                     False
        3             True                False                     False
        4             True                False                     False

           age_group_20-45  age_group_46-60  age_group_61-80  age_group_Under 20
        0            False            False             True               False
        1             True            False            False               False
        2             True            False            False               False
        3             True            False            False               False
        4            False            False             True               False

        [5 rows x 22 columns]
```

- Convert the offer_id and customer_id_ into numerical format.

```
[1227]: dataframe['offer_id'] = pd.factorize(dataframe['offer_id'])[0]
        dataframe['customer_id'] = pd.factorize(dataframe['customer_id'])[0]
        dataframe.head()
```

```
[1227]:    offer_reward                 channels  offer_difficulty  offer_duration  \
        0            10  [email, mobile, social]                10               7
        1            10  [email, mobile, social]                10               7
        2            10  [email, mobile, social]                10               7
        3            10  [email, mobile, social]                10               7
```

```
4           10  [email, mobile, social]                10              7

   offer_id  customer_id  event  time  money_gained  money_spent  ...  \
0         0            0      1     0             0          0.0  ...
1         0            1      1     0             0          0.0  ...
2         0            2      1     0             0          0.0  ...
3         0            3      1     0             0          0.0  ...
4         0            4      1     0             0          0.0  ...

   gender_F  gender_M  gender_O  offer_type_bogo  offer_type_discount  \
0     False      True     False             True                False
1     False      True     False             True                False
2      True     False     False             True                False
3     False      True     False             True                False
4      True     False     False             True                False

   offer_type_informational  age_group_20-45  age_group_46-60  \
0                     False            False            False
1                     False             True            False
2                     False             True            False
3                     False             True            False
4                     False            False            False

   age_group_61-80  age_group_Under 20
0             True               False
1            False               False
2            False               False
3            False               False
4             True               False

[5 rows x 22 columns]
```

- Remove the 'became_member_on' column and create separate columns for the month and year.

[1228]:
```python
dataframe['became_member_on'] = pd.to_datetime(dataframe['became_member_on'].
 ↪astype(str), format='%Y%m%d')
dataframe['month_member'] = dataframe['became_member_on'].dt.month
dataframe['year_member'] = dataframe['became_member_on'].dt.year
dataframe.drop('became_member_on', axis=1, inplace=True)
dataframe.head()
```

[1228]:
```
   offer_reward                 channels  offer_difficulty  offer_duration  \
0            10  [email, mobile, social]                10               7
1            10  [email, mobile, social]                10               7
2            10  [email, mobile, social]                10               7
3            10  [email, mobile, social]                10               7
4            10  [email, mobile, social]                10               7
```

```
   offer_id  customer_id  event  time  money_gained  money_spent  …  \
0         0            0      1     0             0          0.0  …
1         0            1      1     0             0          0.0  …
2         0            2      1     0             0          0.0  …
3         0            3      1     0             0          0.0  …
4         0            4      1     0             0          0.0  …

   gender_O  offer_type_bogo  offer_type_discount  offer_type_informational  \
0     False             True                False                     False
1     False             True                False                     False
2     False             True                False                     False
3     False             True                False                     False
4     False             True                False                     False

   age_group_20-45  age_group_46-60  age_group_61-80  age_group_Under 20  \
0            False            False             True               False
1             True            False            False               False
2             True            False            False               False
3             True            False            False               False
4            False            False             True               False

   month_member  year_member
0             9         2017
1             9         2017
2             7         2016
3             5         2017
4            10         2016

[5 rows x 23 columns]
```

- Drop channel colum

[1229]:
```
dataframe.drop('channels', axis=1, inplace=True)
dataframe.head()
```

[1229]:
```
   offer_reward  offer_difficulty  offer_duration  offer_id  customer_id  \
0            10                10               7         0            0
1            10                10               7         0            1
2            10                10               7         0            2
3            10                10               7         0            3
4            10                10               7         0            4

   event  time  money_gained  money_spent  customer_income  …  gender_O  \
0      1     0             0          0.0         100000.0  …     False
1      1     0             0          0.0          70000.0  …     False
2      1     0             0          0.0          30000.0  …     False
3      1     0             0          0.0          92000.0  …     False
```

```
4      1     0           0         0.0         93000.0  …     False

   offer_type_bogo  offer_type_discount  offer_type_informational  \
0             True                False                     False
1             True                False                     False
2             True                False                     False
3             True                False                     False
4             True                False                     False

   age_group_20-45  age_group_46-60  age_group_61-80  age_group_Under 20  \
0            False            False             True               False
1             True            False            False               False
2             True            False            False               False
3             True            False            False               False
4            False            False             True               False

   month_member  year_member
0             9         2017
1             9         2017
2             7         2016
3             5         2017
4            10         2016

[5 rows x 22 columns]
```

- Scale and normalize the numerical data and remove channel column.

[1230]:
```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
numerical = ['customer_income', 'offer_difficulty', 'offer_duration',␣
↪'offer_reward', 'time', 'money_gained', 'money_spent']
dataframe[numerical] = scaler.fit_transform(dataframe[numerical])
```

[1231]:
```python
dataframe.head(10)
```

[1231]:
```
   offer_reward  offer_difficulty  offer_duration  offer_id  customer_id  \
0           1.0               0.5        0.571429         0            0
1           1.0               0.5        0.571429         0            1
2           1.0               0.5        0.571429         0            2
3           1.0               0.5        0.571429         0            3
4           1.0               0.5        0.571429         0            4
5           1.0               0.5        0.571429         0            5
6           1.0               0.5        0.571429         0            6
7           1.0               0.5        0.571429         0            7
8           1.0               0.5        0.571429         0            8
9           1.0               0.5        0.571429         0            9
```

```
   event  time  money_gained  money_spent  customer_income  …  gender_O  \
0      1   0.0           0.0          0.0         0.777778  …     False
1      1   0.0           0.0          0.0         0.444444  …     False
2      1   0.0           0.0          0.0         0.000000  …     False
3      1   0.0           0.0          0.0         0.688889  …     False
4      1   0.0           0.0          0.0         0.700000  …     False
5      1   0.0           0.0          0.0         0.377778  …     False
6      1   0.0           0.0          0.0         0.455556  …     False
7      1   0.0           0.0          0.0         0.311111  …     False
8      1   0.0           0.0          0.0         0.466667  …     False
9      1   0.0           0.0          0.0         0.011111  …     False

   offer_type_bogo  offer_type_discount  offer_type_informational  \
0             True                False                     False
1             True                False                     False
2             True                False                     False
3             True                False                     False
4             True                False                     False
5             True                False                     False
6             True                False                     False
7             True                False                     False
8             True                False                     False
9             True                False                     False

   age_group_20-45  age_group_46-60  age_group_61-80  age_group_Under 20  \
0            False            False             True               False
1             True            False            False               False
2             True            False            False               False
3             True            False            False               False
4            False            False             True               False
5            False            False             True               False
6            False            False             True               False
7            False            False             True               False
8            False            False             True               False
9            False             True            False               False

   month_member  year_member
0             9         2017
1             9         2017
2             7         2016
3             5         2017
4            10         2016
5             8         2014
6             7         2018
7             2         2018
8             8         2016
9            12         2017
```

[10 rows x 22 columns]

Split training and test data

[1232]:
```
labels = dataframe['event']
```

[1233]:
```
dataframe = dataframe.drop('event', axis=1)
```

[1234]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(dataframe, labels,␣
↪test_size=0.4, random_state=0)

print(f"Training set contains: {X_train.shape[0]} rows")
print(f"Testing set contains: {X_test.shape[0]} rows")
```

```
Training set contains: 94501 rows
Testing set contains: 63002 rows
```

Training

[1235]:
```
from sklearn.metrics import f1_score, fbeta_score

def evaluate_model_performance(model, features_train, features_test,␣
↪labels_train, labels_test):
    model.fit(features_train, labels_train)

    predictions_train = model.predict(features_train)
    predictions_test = model.predict(features_test)

    train_f1_score = f1_score(labels_train, predictions_train, average='micro')␣
↪* 100
    test_f1_score = fbeta_score(labels_test, predictions_test, beta=0.5,␣
↪average='micro') * 100

    model_name = model.__class__.__name__

    return train_f1_score, test_f1_score, model_name
```

The K-Nearest Neighbors algorithm is used to establish the benchmark, and the model's performance is evaluated using the F1 score metric.

[1238]:
```
from sklearn.neighbors import KNeighborsClassifier


benchmark_model = KNeighborsClassifier(n_neighbors = 5)
a_train_f1, a_test_f1, a_model = evaluate_model_performance(benchmark_model,␣
↪X_train, X_test, y_train, y_test)
```

```
knn = {'Benchmark Model': [ benchmark_model], 'train F1 score':[a_train_f1],␣
↪'test F1 score': [a_test_f1]}
benchmark = pd.DataFrame(knn)
```

[1239]:
```
benchmark
```

[1239]:

	Benchmark Model	train F1 score	test F1 score
0	KNeighborsClassifier()	54.287256	33.337037

Training Random Forest Model

[1240]:
```
from sklearn.ensemble import RandomForestClassifier

random_forest_model = RandomForestClassifier(random_state = 10)
b_train_f1, b_test_f1, b_model =␣
↪evaluate_model_performance(random_forest_model, X_train, X_test, y_train,␣
↪y_test)
```

Training Decision Tree Model

[1242]:
```
from sklearn.tree import DecisionTreeClassifier

decision_tree_model = DecisionTreeClassifier(random_state = 10)
c_train_f1, c_test_f1, c_model =␣
↪evaluate_model_performance(decision_tree_model, X_train, X_test, y_train,␣
↪y_test)
```

The initial results for the models are:

[1250]:
```
models = {'Model': [a_model, b_model, c_model], 'train F1 score ':[a_train_f1,␣
↪b_train_f1, c_train_f1], 'test F1 score': [a_test_f1 , b_test_f1, c_test_f1]␣
↪}
results = pd.DataFrame(models)
results
```

[1250]:

	Model	train F1 score	test F1 score
0	KNeighborsClassifier	54.287256	33.337037
1	RandomForestClassifier	95.443434	70.699343
2	DecisionTreeClassifier	95.443434	84.933812

1.4.3 Refinement

Intermediate Steps and Improvements After assessing the initial results, I noticed that both the **RandomForestClassifier** and **DecisionTreeClassifier** performed better than the benchmark **KNeighborsClassifier**. However, I aimed to improve their performance further through hyperparameter tuning and model optimization.

- Random Forest Classifier Tuning I adjusted parameters such as the number of trees (`n_estimators`) and the maximum depth of the trees (`max_depth`) to enhance the model's performance. I tested various combinations using randomized search.

[1260]:
```
import os
os.environ['PYDEVD_DISABLE_FILE_VALIDATION'] = '1'

from sklearn.model_selection import RandomizedSearchCV

random_forest_model = RandomForestClassifier(random_state=10)

param_dist = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30]
}

random_search_rf = RandomizedSearchCV(estimator=random_forest_model,
                                       param_distributions=param_dist,
                                       n_iter=10,
                                       scoring='f1_macro',
                                       cv=3,
                                       n_jobs=-1,
                                       random_state=10)

random_search_rf.fit(X_train, y_train)

best_rf_model = random_search_rf.best_estimator_

b_train_f1, b_test_f1, b_model = evaluate_model_performance(best_rf_model,␣
↪X_train, X_test, y_train, y_test)
```

```
371855.36s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.36s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.38s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.38s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.38s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.39s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.40s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.40s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.40s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.42s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.43s - pydevd: Sending message related to process being replaced timed-out
```

```
after 5 seconds
371855.43s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.44s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.44s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.44s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
371855.44s - pydevd: Sending message related to process being replaced timed-out
after 5 seconds
```

- Decision Tree Classifier Tuning For the **DecisionTreeClassifier**, I also performed hyperparameter tuning by adjusting the maximum depth and the minimum samples required to split a node.

[1254]:
```python
from sklearn.model_selection import GridSearchCV



decision_tree_model = DecisionTreeClassifier(random_state=10)

param_grid_dt = {
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10]
}

grid_search_dt = GridSearchCV(estimator=decision_tree_model,␣
 ↪param_grid=param_grid_dt, scoring='f1_macro', cv=5)
grid_search_dt.fit(X_train, y_train)
best_dt_model = grid_search_dt.best_estimator_
c_train_f1, c_test_f1, c_model = evaluate_model_performance(best_dt_model,␣
 ↪X_train, X_test, y_train, y_test)
```

[1256]:
```python
models = {'Model': [a_model, b_model, c_model], 'train F1 score ':[a_train_f1,␣
 ↪b_train_f1, c_train_f1], 'test F1 score': [a_test_f1 , b_test_f1, c_test_f1]␣
 ↪}
results = pd.DataFrame(models)
results
```

[1256]:

	Model	train F1 score	test F1 score
0	KNeighborsClassifier	54.287256	33.337037
1	RandomForestClassifier	92.096380	74.767468
2	DecisionTreeClassifier	92.454048	91.920891

1.5 IV. Results

1.5.1 Model Evaluation and Validation

The F1 score will be utilized as the primary metric for evaluating the effectiveness of the approach and identifying the model that yields the most favorable results. This score can be understood as the weighted average of precision and recall. Specifically, the balanced F-score, commonly known as the F1 score, represents the harmonic mean of precision and recall. Its values range from 0 to 100, with a score of 100 indicating optimal performance and 0 representing the worst outcome.

[1257]:
```
results
```

[1257]:

	Model	train F1 score	test F1 score
0	KNeighborsClassifier	54.287256	33.337037
1	RandomForestClassifier	92.096380	74.767468
2	DecisionTreeClassifier	92.454048	91.920891

1.5.2 Justification

The validation set was used to evaluate the performance of different machine learning models in predicting customer responses to marketing offers. The KNeighborsClassifier served as the baseline for comparison, achieving a test F1 score of **33.34**. Both the RandomForestClassifier and DecisionTreeClassifier significantly outperformed this baseline.

Among the models tested, the **DecisionTreeClassifier** achieved the highest test F1 score of **91.92**, indicating its effectiveness in classifying customer responses to promotional offers. The **RandomForestClassifier** also demonstrated strong performance, with a test F1 score of **74.77**. Both models exhibit a considerable improvement over the baseline, showcasing their capability in effectively predicting customer engagement.

Since the primary goal is to predict customer responses to marketing offers, an extremely high F1 score isn't strictly necessary. The performance metrics suggest that both the DecisionTreeClassifier and RandomForestClassifier are robust enough for practical application in this context. Consequently, their scores are deemed satisfactory for our needs.

In summary, while the baseline model has its limitations, both the RandomForestClassifier and DecisionTreeClassifier substantially enhance predictive accuracy. These models are particularly well-suited for the Starbucks Capstone Challenge, indicating their potential to effectively predict customer engagement with offers and provide valuable insights for marketing strategies.