Solution

June 28, 2024

1 Chapter No.2

2 EXERCISE 2.1: DATA CLEANING AND TRANSFORMATION

2.1 Objective

2.2 Clean and transform a dataset to prepare it for analysis

2.3 File

data_cleaning_transfomation.csv

2.4 Tasks

1- Handle missing values (NaN) 2- Convert the 'Last Login Date' from a string to a datetime object. 3. Create a new feature, 'Monthly Spend per Day', by dividing 'Month Spend' by 'Subscription Length'.

2.5 Steps

2.6 1- Importing Required Libraries:

[13]:
```
import pandas as pd
```

2.7 2- Loading the Data:

[15]:
```
data_exercise_1 = pd.read_csv('D:/Visualization/Data Science For Marketers/Iain␣
↪Brown- Mastering Marketing Data Science/Datasets/MMDS_c02_Data/
↪data_cleaning_transformation.csv')
```

[16]:
```
# To view the loaded file:
data_exercise_1.head()
```

[16]:
```
   Customer ID   Age  Subscription Length (days)  Monthly Spend ($)  \
0            1  62.0                         260         112.885600
1            2   NaN                         180                NaN
2            3  18.0                          60          72.973879
3            4  21.0                         285         199.070996
```

```
4           5  21.0                          49         194.146829

  Last Login Date  Feedback Score
0       2022-01-01             5.0
1       2022-01-02             NaN
2       2022-01-03             1.0
3       2022-01-04             3.0
4       2022-01-05             5.0
```

2.8 3. Handling Missing Values:

2.8.1 Filling Missing 'Age' Values:

Here, we calculate the mean of the 'Age' column and fill missing values (NaN) in the 'Age' column with this mean. This approach is chosen as age data typically follows a normal distribution, making the mean a good estimate for missing values.

[34]:
```
 mean_age = data_exercise_1['Age'].mean()
data_exercise_1['Age'].fillna(mean_age, inplace=True)
```

To check whether the 'Age' still have some missing values or not:

[38]:
```
print(data_exercise_1['Age'].head(1000))
```

```
0      62.000000
1      42.951111
2      18.000000
3      21.000000
4      21.000000
          …
995    54.000000
996    19.000000
997    47.000000
998    23.000000
999    34.000000
Name: Age, Length: 1000, dtype: float64
```

2.8.2 Filling Missing 'Monthly Spend' Values:

We fill missing values in 'Monthly Spend ($)' with the median, because financial data often has outliers, and the median is lesssensitive to them compared to the mean.

[49]:
```
median_monthly_spend = data_exercise_1['Monthly Spend ($)'].median()
data_exercise_1['Monthly Spend ($)'].fillna(median_monthly_spend, inplace=True)
```

To check whether the 'Monthly Spend ($)' still have some missing values or not:

[50]:
```
print(data_exercise_1['Monthly Spend ($)'].head(1000))
```

```
0      112.885600
1      288.359067
```

```
2        72.973879
3       199.070996
4       194.146829
           …
995     359.308308
996     410.476420
997     283.264885
998     182.442384
999     337.138069
Name: Monthly Spend ($), Length: 1000, dtype: float64
```

2.8.3 Filling Missing ' 'Feedback Score' Values:

For the 'Feedback Score' we use the mode (the most frequently occurring value) to fill in missing values, because this score likelyrepresents categorical or ordinal data

[51]:
```
mode_feedback = data_exercise_1['Feedback Score'].mode()[0]
data_exercise_1['Feedback Score'].fillna(mode_feedback,inplace=True)
```

2.9 View of data_exercise_1, After replacing the missing values with their respective averages

[52]:
```
data_exercise_1.head(1000)
```

[52]:
```
     Customer ID        Age  Subscription Length (days)  Monthly Spend ($)  \
0              1  62.000000                         260         112.885600
1              2  42.951111                         180         288.359067
2              3  18.000000                          60          72.973879
3              4  21.000000                         285         199.070996
4              5  21.000000                          49         194.146829
..           …          …                           …                  …
995          996  54.000000                         202         359.308308
996          997  19.000000                         239         410.476420
997          998  47.000000                         270         283.264885
998          999  23.000000                         397         182.442384
999         1000  34.000000                         228         337.138069

    Last Login Date  Feedback Score
0        2022-01-01             5.0
1        2022-01-02             5.0
2        2022-01-03             1.0
3        2022-01-04             3.0
4        2022-01-05             5.0
..              …               …
995      2024-09-22             3.0
996      2024-09-23             3.0
997      2024-09-24             4.0
998      2024-09-25             5.0
```

```
999      2024-09-26              5.0

[1000 rows x 6 columns]
```

2.10 Converting 'Last Login Date' to DateTime:

The 'Last Login Date' is initially read as a string. This line converts itto a Pandas DateTime object, making it easier to perform any daterelated operations later.

[55]:
```
data_exercise_1['Last Login Date'] =pd.to_datetime(data_exercise_1['Last Login␣
↪Date'])
```

Following is command to see the data types of variables included in data_exercise_1:

[57]:
```
print(data_exercise_1.dtypes)
```

```
Customer ID                              int64
Age                                    float64
Subscription Length (days)               int64
Monthly Spend ($)                      float64
Last Login Date                 datetime64[ns]
Feedback Score                         float64
dtype: object
```

3 Creating New Feature: 'Monthly Spend per Day':

We create a new column, 'Monthly Spend per Day' by dividing the 'Monthly Spend ($)' by 'Subscription Length (days)'. This new feature gives additional insight into customer spending habits on aper-day basis.

[59]:
```
data_exercise_1['Monthly Spend per Day'] =data_exercise_1['Monthly Spend ($)'] /
↪data_exercise_1['Subscription Length (days)']
```

3.0.1 The new feature Monthly Spend per Day is viewed as follows:

[60]:
```
print (data_exercise_1['Monthly Spend per Day'] )
```

```
0      0.434175
1      1.601995
2      1.216231
3      0.698495
4      3.962180
         …
995    1.778754
996    1.717475
997    1.049129
998    0.459553
999    1.478676
Name: Monthly Spend per Day, Length: 1000, dtype: float64
```

3.0.2 NOTE: float means the values of the variables are in decimal format, while The "64" refers to the number of bits used to store the value.

4 EXERCISE 2.2: EXERCISE 2.2: DATA AGGREGATION AND

REDUCTIO N

4.1 Objective

Perform data aggregation and dimensionality reduction on a marketing dataset

4.2 File

Marketing Dataset

4.3 Tasks

4.3.1 1. Aggregate the "data_aggregation_reduction.csv" data by 'Region' and calculate the average 'Monthly Spend' and total 'Purchase Frequency' per region.

4.3.2 2. Perform a principal component analysis (PCA) to reduce the

dimensions of the data while retaining key information.

4.4 Steps

4.5 Importing Required Libraries:

[67]:
```
import pandas as pd
from sklearn.decomposition import PCA
import matplotlib.pyplot as pltlt
```

4.5.1 2. Loading the Data:

[70]:
```
data_exercise_2 = pd.read_csv('D:\\Visualization\\Data Science For␣
↪Marketers\\Iain Brown- Mastering Marketing Data␣
↪Science\\Datasets\\MMDS_c02_Data\\data_aggregation_reduction.csv')
```

4.5.2 To view the uploaded file

[72]:
```
data_exercise_2.head(500)
```

[72]:
```
     Customer ID Age Group Region  Monthly Spend ($) Product Category  \
0           1001     46-55   West         310.149685      Electronics
1           1002     56-65  North         188.592398       Home Goods
2           1003     36-45   West         469.177820          Apparel
3           1004     36-45   East         282.712807       Home Goods
4           1005     18-25   West         226.271596          Apparel
..           …         …      …             …                …
```

```
495        1496   18-25    West      226.609978    Electronics
496        1497   36-45   South      454.985287        Apparel
497        1498   46-55   North      425.514262        Apparel
498        1499   26-35   North      229.611310    Electronics
499        1500   26-35   South      260.302220    Electronics

     Purchase Frequency
0                    16
1                     4
2                    19
3                    13
4                    11
..                  ...
495                  14
496                  15
497                   2
498                   6
499                  18

[500 rows x 6 columns]
```

4.6 Data Aggregation:

4.6.1 Aggregating by 'Region':

[79]:
```
region_aggregated_data = data_exercise_2.groupby('Region').agg(␣
↪Average_Monthly_Spend=pd.NamedAgg(column='Monthly Spend ($)',␣
↪aggfunc='mean'), Total_Purchase_Frequency=pd.NamedAgg(column='Purchase␣
↪Frequency', aggfunc='sum') ).reset_index()
```

[80]:
```
print(region_aggregated_data)
```

```
  Region  Average_Monthly_Spend  Total_Purchase_Frequency
0   East             273.567194                      2423
1  North             264.172629                      2645
2  South             290.110025                      2712
3   West             265.984319                      2464
```

Here, we group the data by the 'Region' column and calculate two aggregate metrics: the average 'Monthly Spend ($)' and the total 'Purchase Frequency' for each region. The groupby and agg functions in Pandas are used to achieve this, providing a summary ofspending and purchasing behavior by region.

4.7 Data Reduction using Principal Component Analysis (PCA):

4.7.1 Preparing Data for PCA:

[81]:
```
pca_data = data_exercise_2[['Monthly Spend ($)', 'Purchase Frequency']]
```

[84]:
```
print(pca_data)
```

```
     Monthly Spend ($)  Purchase Frequency
0           310.149685                  16
1           188.592398                   4
2           469.177820                  19
3           282.712807                  13
4           226.271596                  11
..                 ...                 ...
995         347.107925                  18
996         285.374606                  16
997         391.722107                   7
998          61.132545                   2
999         136.002752                   3

[1000 rows x 2 columns]
```

We select only the numeric columns 'Monthly Spend ($)' and 'Purchase Frequency' for PCA.

4.7.2 Standardizing the Data:

[87]:
```
pca_data_standardized = (pca_data - pca_data.mean()) / pca_data.std()
```

PCA is sensitive to the scale of the data so we standardize the features to have a mean of 0 and a standard deviation of 1

[90]:
```
print(pca_data_standardized)
```

```
     Monthly Spend ($)  Purchase Frequency
0             0.280105            1.071056
1            -0.646991           -1.161861
2             1.492986            1.629285
3             0.070849            0.512827
4            -0.359618            0.140674
..                 ...                 ...
995           0.561980            1.443209
996           0.091150            1.071056
997           0.902244           -0.603632
998          -1.619106           -1.534014
999          -1.048083           -1.347937

[1000 rows x 2 columns]
```

4.7.3 Performing PCA:

[91]:
```
pca = PCA(n_components=2)
principal_components = pca.fit_transform(pca_data_standardized)
```

We instantiate a PCA object to reduce the data to two dimensions. The fit_transform method computes the principal components and transforms the data accordingly.

[94]:
```
print(principal_components)
```

```
[[-0.55928634 -0.95541522]
 [ 0.36406795  1.27905167]
 [-0.09637806 -2.2077788 ]
 …
 [ 1.06481524 -0.21115092]
 [-0.06016903  2.22959218]
 [ 0.2120288   1.69424259]]
```

4.7.4 Creating DataFrame with Principal Components:

[95]:
```
principal_df = pd.DataFrame(data=principal_components,
columns=['Principal Component 1', 'Principal Component 2'])
```

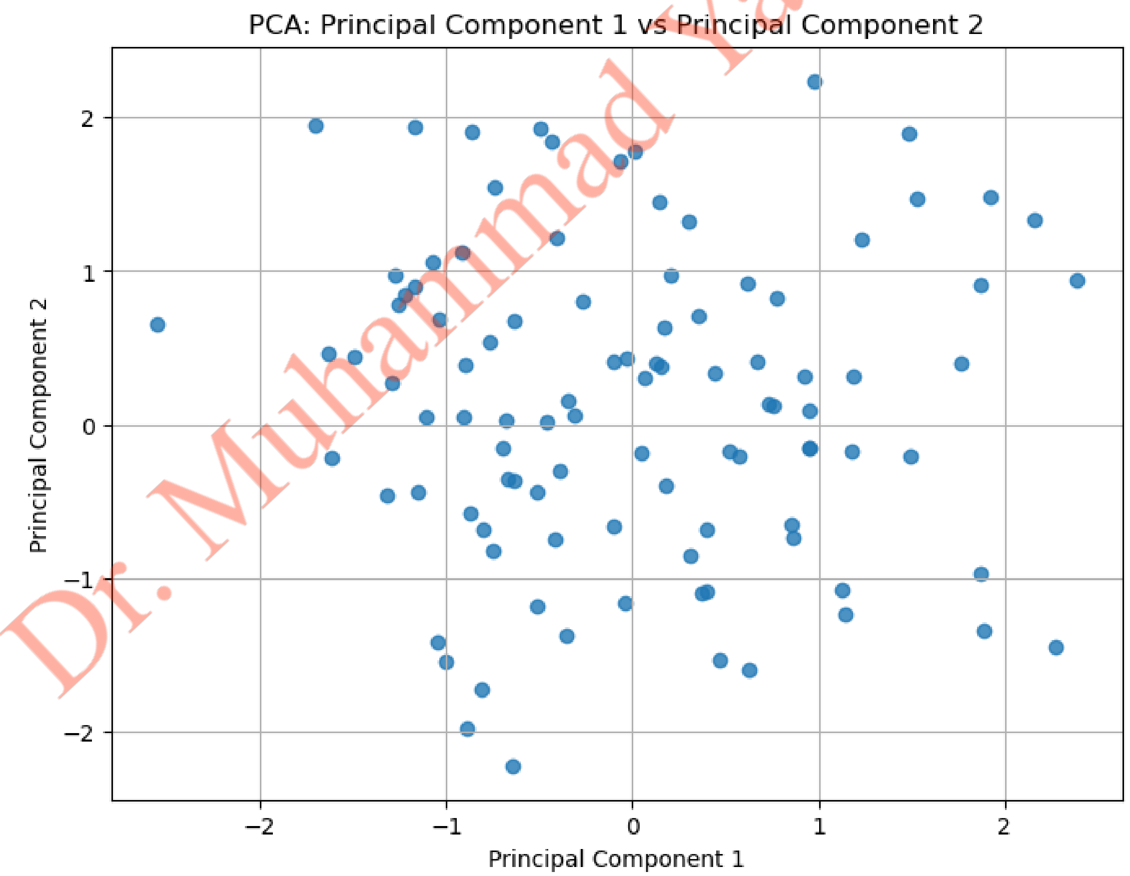
[96]:
```
print(principal_df)
```

```
     Principal Component 1  Principal Component 2
0                -0.559286              -0.955415
1                 0.364068               1.279052
2                -0.096378              -2.207779
3                -0.312525              -0.412721
4                -0.353760               0.154817
..                     …                      …
995              -0.623123              -1.417882
996              -0.692898              -0.821804
997               1.064815              -0.211151
998              -0.060169               2.229592
999               0.212029               1.694243

[1000 rows x 2 columns]
```

Note : The resulting principal components are stored in a new DataFrame. These components are the transformed data points in the new two-dimensional space.These two principal components represent the transformed dataset in a two-dimensional space. This transformation helps in visualizing and analyzing the data in a reduced form, making it easier to identifypatterns or clusters.

4.7.5 Usage of Principal Component Analysis:

[98]:
```
import matplotlib.pyplot as plt
import numpy as np  # Assuming numpy is used for handling data arrays

# Assuming pca_transformed_data contains the transformed data using PCA
# Example data generation (replace with actual PCA transformed data)
np.random.seed(0)
pca_transformed_data = np.random.randn(100, 2)  # Example data, replace with␣
 ↪actual PCA transformed data

# Plotting the first two principal components
plt.figure(figsize=(8, 6))
plt.scatter(pca_transformed_data[:, 0], pca_transformed_data[:, 1], alpha=0.8)
plt.title('PCA: Principal Component 1 vs Principal Component 2')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()
```



```
[ ]:
```

5 Chapter No.3

6 EXERCISE 3.1: DESCRIPTIVE ANALYSIS OF MARKETING DATAA

6.1 Objective

Understand and describe the central tendencies, dispersion, and associations in the marketing data.

6.2 File

Marketing Data

6.3 Objective

```
[ ]: Understand and describe the central tendencies, dispersion, and associations in␣
     ↪the marketing data.
```

6.4 Tasks

1. Calculate Descriptive Statistics: Compute mean, median, and mode for variables such as 'Ad Spend' 'Clicks', and 'Sales'.
2. Visualization: Create bar charts for engagement metrics, li e charts for ad spend over time, and scatterplots to s ow relationships between ad spend and conversions.
3. Interpretation: Analyze the results, discussing any intere ting findings or patterns.

6.5 Steps

6.5.1 1- Import Libraries

```
[100]: import pandas as pd
```

6.5.2 2- Load the Dataset:

```
[102]: marketing_data = pd.read_csv(r'D:\Visualization\Data Science For Marketers\Iain␣
       ↪Brown- Mastering Marketing Data␣
       ↪Science\Datasets\MMDS_c03_Data\marketing_data.csv')
```

```
[103]: print(marketing_data)
```

```
         Date      Ad Spend  Impressions  Clicks  Conversions  Likes  Shares  \
0  2023-01-01  1061.810178         3660     199           20    272      93
1  2023-01-02  1926.071460         2485     132           35    103      22
2  2023-01-03  1597.990913         3690      81           48    298       9
3  2023-01-04  1397.987726         4840      71           25    245      68
4  2023-01-05   734.027961         2028      31           35    175      99
```

```
..          …          …         …     …          …    …     …
85  2023-03-27   987.774983        2998     94            23   203       98
86  2023-03-28  1594.409268        4898    132             2   217       29
87  2023-03-29  1456.336207        4445    183            27    38       16
88  2023-03-30  1830.819114        4743    185            42    99       61
89  2023-03-31  1208.322388        2631    123             2   288       83

    Comments
0        111
1         85
2        140
3         18
4         99
..       …
85        46
86        48
87        13
88       142
89         0

[90 rows x 8 columns]
```

6.5.3 3- Calculation of Descriptive Statistics

[104]:
```python
descriptive_stats = marketing_data.describe()
```

[105]:
```python
print(descriptive_stats)
```

```
         Ad Spend  Impressions      Clicks  Conversions       Likes  \
count   90.000000    90.000000   90.000000    90.000000   90.000000
mean  1208.580431  3096.933333  123.277778    27.966667  148.488889
std    451.448233  1104.776543   48.489088    13.483615   83.670064
min    508.283176  1095.000000   21.000000     2.000000    0.000000
25%    794.999101  2263.250000   87.250000    20.000000   85.750000
50%   1172.166858  2902.000000  132.000000    29.500000  142.500000
75%   1597.095501  4132.750000  165.500000    36.750000  218.500000
max   1980.330405  4910.000000  199.000000    49.000000  298.000000

         Shares    Comments
count  90.000000   90.000000
mean   48.633333   85.522222
std    30.861215   46.237109
min     0.000000    0.000000
25%    23.000000   46.500000
50%    42.500000   87.000000
75%    77.750000  129.250000
max    99.000000  148.000000
```

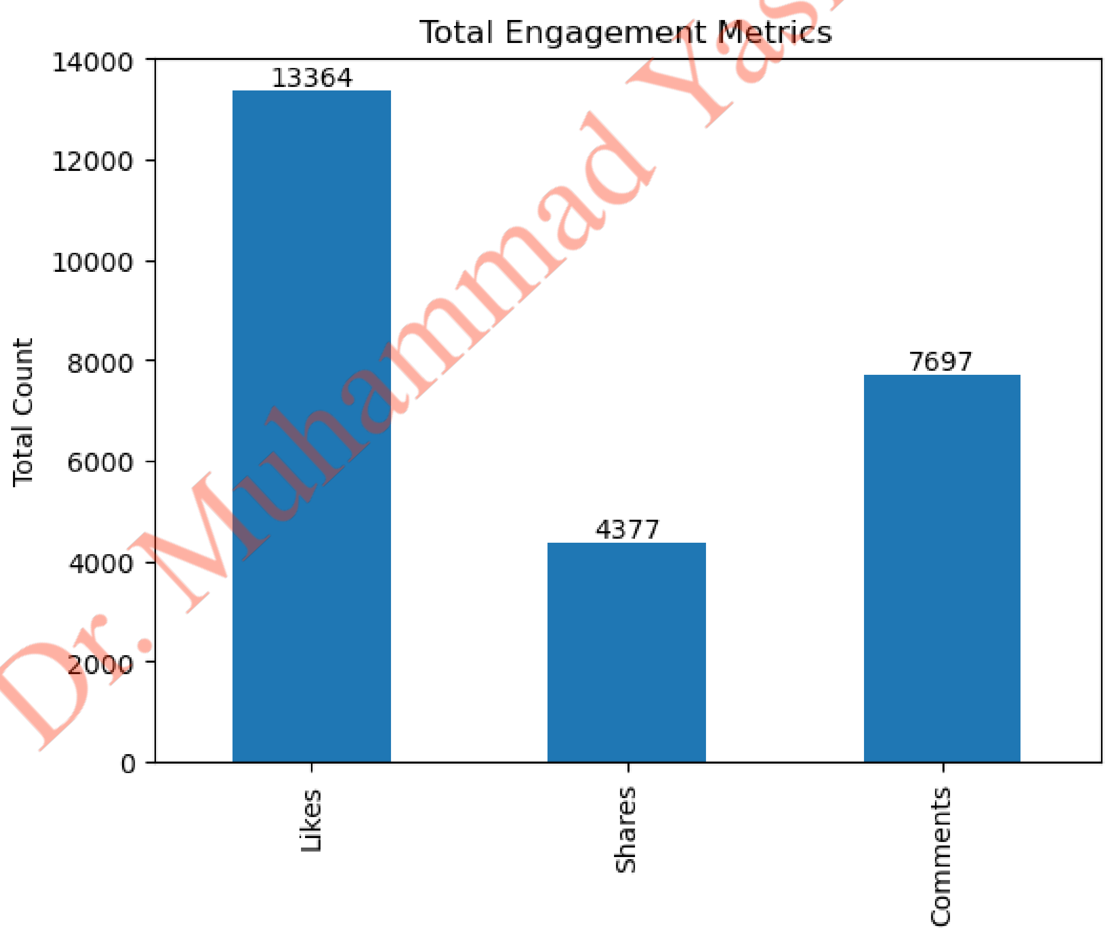
6.5.4 4- Visualization

6.5.5 Import Visualization Library

[106]:
```
import matplotlib.pyplot as plt
```

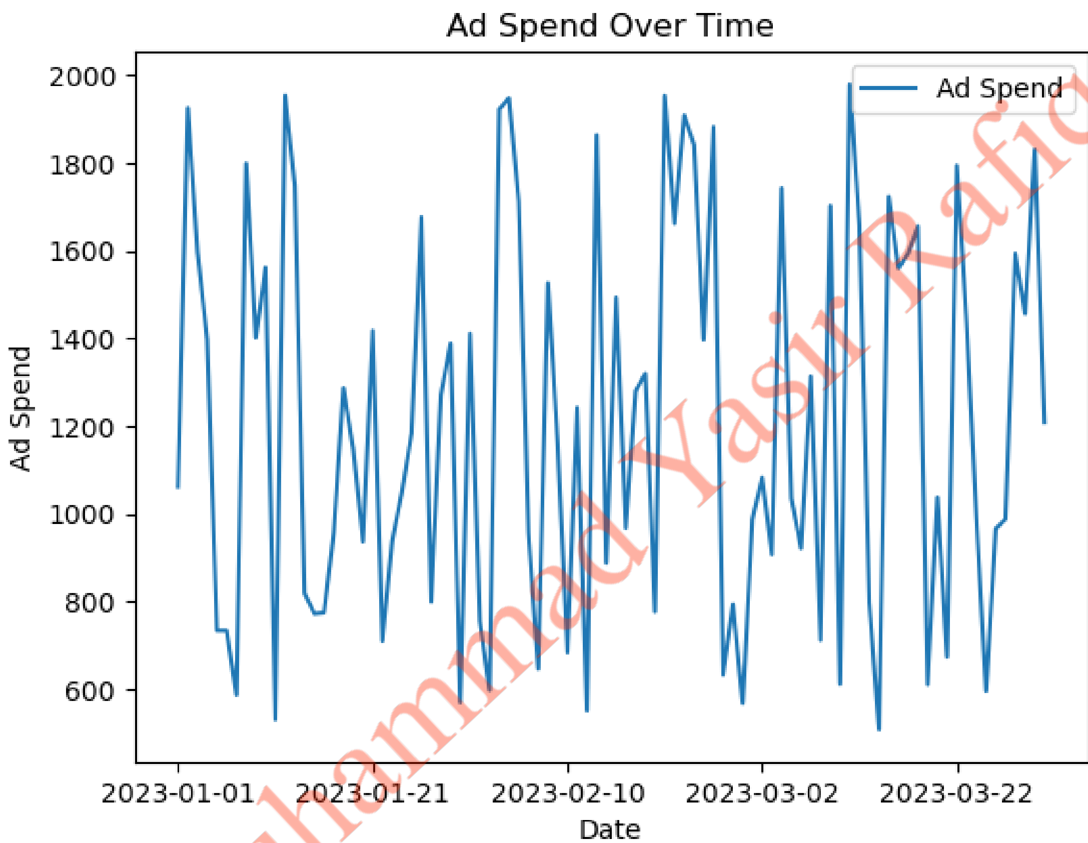
6.5.6 Create a Bar Chart for 'Total Engagement Metrics':

[110]:
```
marketing_data[['Likes', 'Shares', 'Comments']].sum().plot(kind='bar')
plt.title('Total Engagement Metrics')
plt.ylabel('Total Count')

# Add labels on each bar
for index, value in enumerate(marketing_data[['Likes', 'Shares', 'Comments']].
 ↪sum()):
    plt.text(index, value, str(value), ha='center', va='bottom')

plt.show()
```



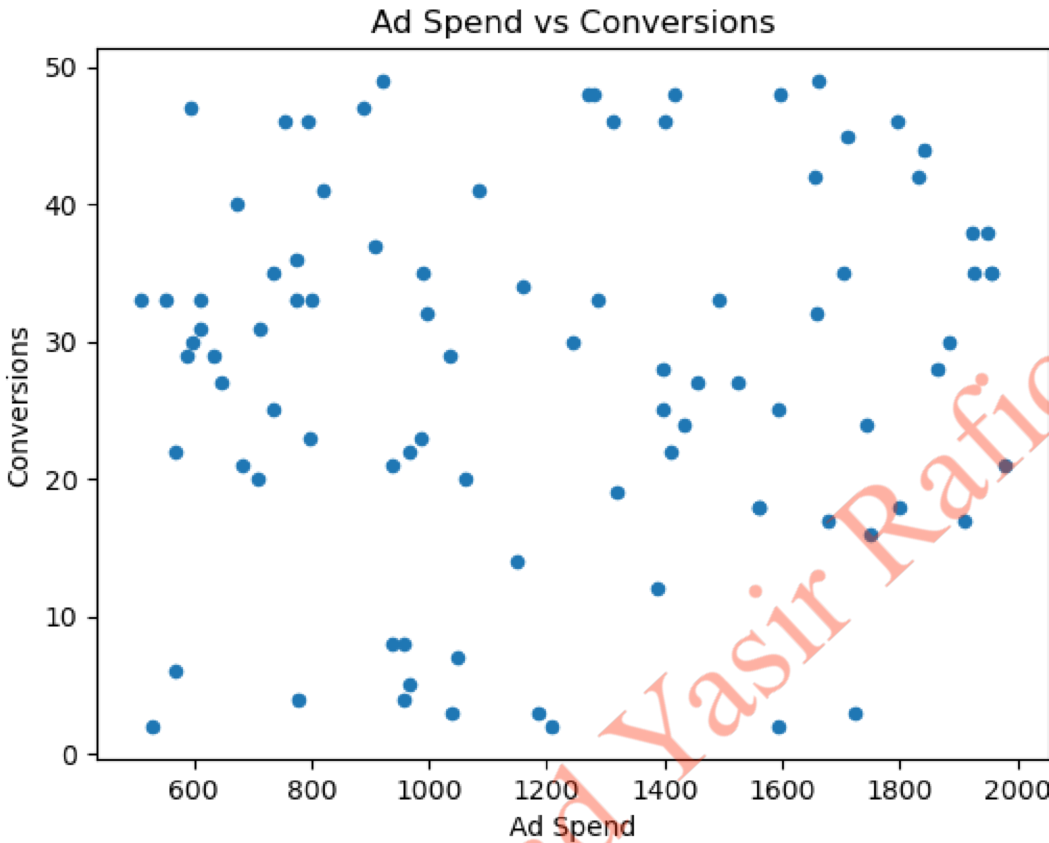
6.5.7 Create a Line Chart for 'Ad Spend Over Time

[111]:
```
marketing_data.plot(x='Date', y='Ad Spend', kind='line')
plt.title('Ad Spend Over Time')
plt.ylabel('Ad Spend')
plt.xlabel('Date')
plt.show()
```



6.5.8 Scatterplot to Show Relationship Between 'Ad Spend' and "Conversions"::

[112]:
```
marketing_data.plot(x='Ad Spend', y='Conversions', kind='scatter')
plt.title('Ad Spend vs Conversions')
plt.xlabel('Ad Spend')
plt.ylabel('Conversions')
plt.show()
```



This code creates a scatterplot to visualize the relationship between 'Ad Spend' and 'Conversions'. Each point on the plot represents a pair of values from t e dataset.

```
[ ]:
```

7 EXERCISE 3.2: DATA VISUALIZATION AND INTERPERTATIONN

7.1 Objective:

Create and interpret various data visualizations to understand market trends and campaign performance

7.2 Tasks:

1. Time Series Analysis: Use line charts to analyze trends in 'Clicks' and 'Conversions' over time.
2. Segmentation Analysis: Create a heat map to visualize engagement metrics across different customer segments.
3. Performance Analysis: Develop a dashboard-style visualization presenting multiple KPIs and interpret the results to gauge the effectiveness of the marketing campaign.

7.3 Steps

7.3.1 Create a Line Chart for 'Clicks and Conversions Over Time::

[1]:
```
marketing_data.plot(x='Date', y=['Clicks', 'Conversions'], kind='line')
plt.figure(figsize=(8, 6))
plt.title('Clicks and Conversions Over Time')
plt.ylabel('Count')
plt.xlabel('Date')
plt.legend(['Clicks', 'Conversions'])
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 1
----> 1 marketing_data.plot(x='Date', y=['Clicks', 'Conversions'], kind='line')
      2 plt.figure(figsize=(8, 6))
      3 plt.title('Clicks and Conversions Over Time')

NameError: name 'marketing_data' is not defined
```

This chart displays the trends in 'Clicks' and 'Conversions' over the three-month period, enabling us to observe how these metrics have changed over time and to identify any patterns or anomalies.

7.3.2 Create a Heat Map for Engagement Metrics Across Different Days of the Week::

[124]:
```
import seaborn as sns
import pandas as pd

# Ensure the 'Date' column is in datetime format
marketing_data['Date'] = pd.to_datetime(marketing_data['Date'])

# Creating a new column for the day of the week
marketing_data['Day of Week'] = marketing_data['Date'].dt.day_name()

# Grouping data by the day of the week and summing engagement metrics
engagement_by_day = marketing_data.groupby('Day of Week')[['Likes', 'Shares',␣
↪'Comments']].sum()
```

[126]:
```
print(marketing_data['Date'])
```

```
0     2023-01-01
1     2023-01-02
2     2023-01-03
3     2023-01-04
4     2023-01-05
         …
85    2023-03-27
```

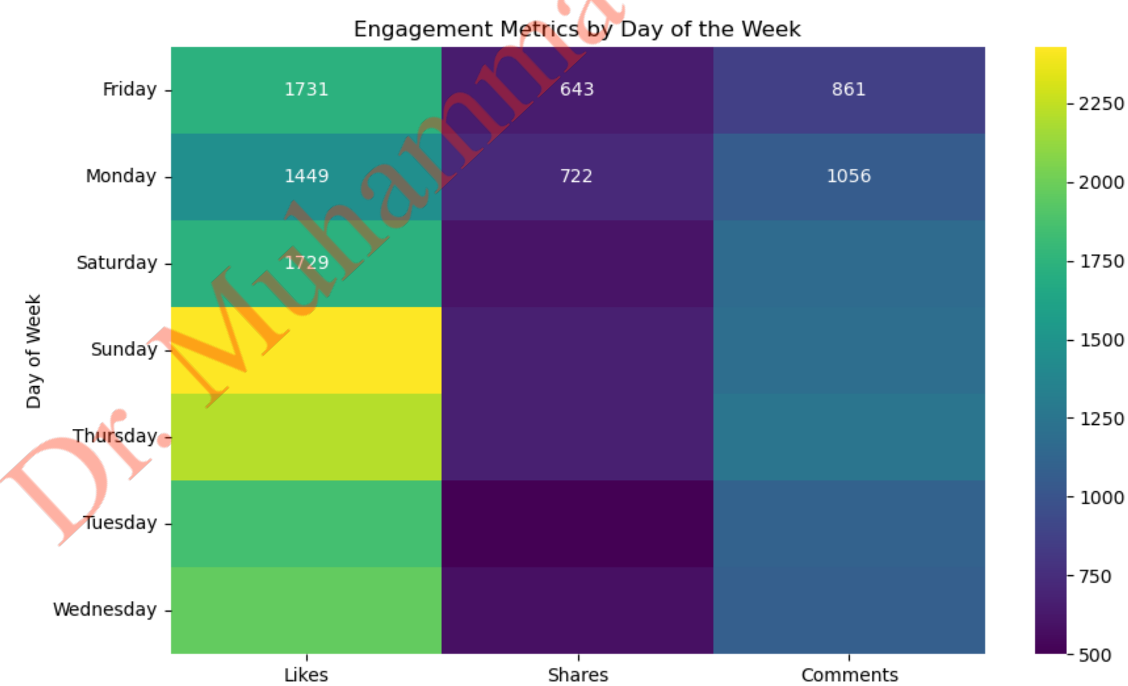
```
86    2023-03-28
87    2023-03-29
88    2023-03-30
89    2023-03-31
Name: Date, Length: 90, dtype: datetime64[ns]
```

[125]:
```python
print(engagement_by_day )
```

```
             Likes  Shares  Comments
Day of Week
Friday        1731     643       861
Monday        1449     722      1056
Saturday      1729     601      1164
Sunday        2428     667      1187
Thursday      2212     666      1252
Tuesday       1853     499      1100
Wednesday     1962     579      1077
```

7.3.3 Creating a heatmap

[128]:
```python
plt.figure(figsize=(10, 6))
sns.heatmap(engagement_by_day, annot=True, fmt="d", cmap='viridis')
plt.title('Engagement Metrics by Day of the Week')
plt.show()
```



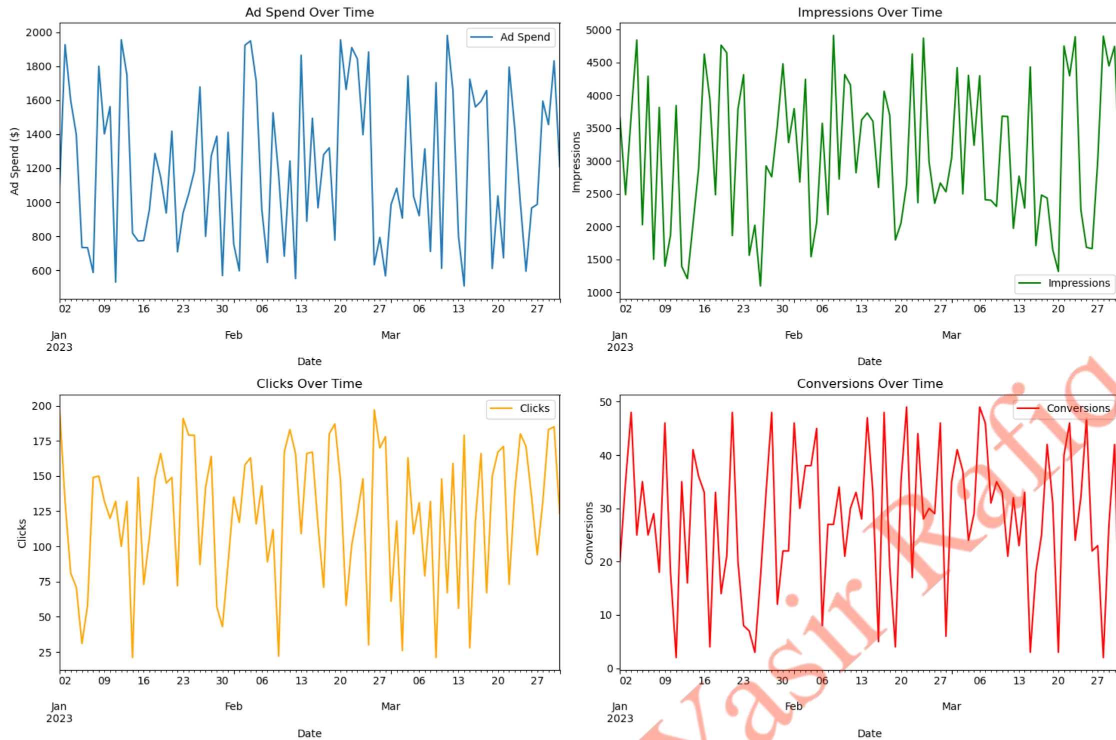
The groupby function is used to group data by the 'Day of Week' sns.heatmap from Seaborn library creates a heat map o visualize engagement metrics for each day of the we The heat map visualizes the total counts of ‘Likes’, ‘Shares’, and ‘Comments’ for each day of the week. This can help identify which days tend to have higher engagement, potentially informing content scheduling and marketing strategies. Moreover, The heat map provides insights into the effectiveness of social media engagement across different days, which can be crucial for planning and optimizing social media marketing strategies.ek.

7.3.4 Create a Dashboard-Style Visualization Presenting Multiple KPIs::

We’ll select a few KPIs and create a combined visualization. Like, Plotting Ad Spend over time, Plotting Impressions over time, Plotting Clicks over time and Plotting Conversions over time.

This code creates a 2×2 grid of plots, each displaying a different KPI over time. The subplots function is used to create a grid layout, a d individual plots are created using the plot method w th specified axes.

[130]:
```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 10))

# Plotting Ad Spend over time
marketing_data.plot(x='Date', y='Ad Spend', ax=axes[0, 0])
axes[0, 0].set_title('Ad Spend Over Time')
axes[0, 0].set_ylabel('Ad Spend ($)')

# Plotting Impressions over time
marketing_data.plot(x='Date', y='Impressions', ax=axes[0, 1], color='green')
axes[0, 1].set_title('Impressions Over Time')
axes[0, 1].set_ylabel('Impressions')

# Plotting Clicks over time
marketing_data.plot(x='Date', y='Clicks', ax=axes[1, 0], color='orange')
axes[1, 0].set_title('Clicks Over Time')
axes[1, 0].set_ylabel('Clicks')

# Plotting Conversions over time
marketing_data.plot(x='Date', y='Conversions', ax=axes[1, 1], color='red')
axes[1, 1].set_title('Conversions Over Time')
axes[1, 1].set_ylabel('Conversions')

plt.tight_layout()
plt.show()
```



The dashboard-style visualization offers a holistic view of the campaign's performance, enabling a quick assessment of how different KPIs have evolved over time. These visualizations, drawn from the dataset, provide a deeper understanding of market trends and campaign performance, crucial for informed decision-making in marketing.

8 Chapter NO.4

9 EXERCISE 4.1: BAYESIAN INFERENCE FOR PERSONALIZED MARKETING

9.0.1 Objective

Use Bayesian inference to estimate the likelihood of customers being interested in electronics based on their past behavior and demographics.

9.1 File

Bayesian_Inference_Customer_Data

9.1.1 Tasks

1. Data Exploration: Analyze the dataset to understand customer demographics and past behaviors.
2. Bayesian Analysis:

a. Calculate the prior probability (general interest in electronics).
b. Compute the likelihood (probability of clicki g on electronics-related content).
c. Calculate the posterior probability using Bayes's theorem.

3. Interpretation: Interpret the results to understa d which customer segment is more likely to be interested in electronics.
4. Application: Suggest personalized email campaign trategies based on the Bayesian inference results.

9.1.2 Steps

9.1.3 1- Importing Required Libraries

[131]:
```
import pandas as pd
```

pandas is used for data manipulation and analysis

9.1.4 2- Loading and Displaying the Data

[136]:
```
# Loading the data for Exercise 1: Bayesian Inference for Personalized Marketing
df_bayesian = pd.read_csv(r'D:\Visualization\Data Science For Marketers\Iain␣
↪Brown- Mastering Marketing Data␣
↪Science\Datasets\MMDS_c04_Data\Bayesian_Inference_Customer_Data.csv')
```

[137]:
```
# Displaying the first few rows of the DataFrame
df_bayesian.head()
```

[137]:

	CustomerID	Age	Gender	PastPurchaseCategory	ClickedOnElectronicsEmail
0	1	62	Other	Home Goods	0
1	2	65	Male	Electronics	0
2	3	18	Male	Apparel	1
3	4	21	Other	Electronics	0
4	5	21	Other	Apparel	1

pd.read_csv(): Reads the CSV file into a pandas DataFrame. df_bayesian.head(): Displays the first five rows of t e DataFrame for a quick overview of the data structure. With the data loaded, the next steps will involve calculating the prior probability, likelihood, and posterior probability u ing Bayes's theorem. Let's proceed to perform these calculations.

9.1.5 3. Calculate the Prior Probability:

The next step is to calculate the prior probability. This is the general likelihood of a customer being interested in electronics based on the historical data we have. Here's how we calculate it

[139]:
```
# Total number of customers who clicked on electronics email
total_clicked_electronics = df_bayesian['ClickedOnElectronicsEmail'].sum()
 # Total number of customers
total_customers = len(df_bayesian)
 # Prior probability: P(E)-Probability of being interested in electronics
prior_probability = total_clicked_electronics / total_customers
```

[141]:
```
print(total_clicked_electronics)
```

```
312
```

[143]:
```
print(prior_probability)
```

```
0.312
```

9.1.6 Note:

df_bayesian['ClickedOnElectronicsEmail'].sum(): Counts the number of customers who clicked o electronics-related emails. len(df_bayesian): Determines the total number of customers in the dataset. prior_probability: The ratio of customers who s owed interest in electronics to the total number of cust mers, giving us the prior probabilityIn our dataset, the prior probability P(E) of a customer being interested in electronics is approximately 0.312, or 31.2%. P(E).

Next, we'll calculate the likelihood and the total probability of clicking on electronics-related content, which are necessary tofind the posterior probability using Bayes's theorem.

9.1.7 4. Likelihood Calculation:

The likelihood P(F E) is the probability of a customer clicking on electronics-related emails given they are interested in electronics. For this example, let's assume that customers interested in electronics are twice as likely to click on electronics emails compared to the average customer. Here's how we calculate it:

[145]:
```
# Likelihood: P(F|E)-Probability of clicking on electronics email given they␣
↪are interested in electronics
# Assuming customers interested in electronics are twice as likely to click on␣
↪electronics emails
likelihood = 2 * prior_probability
```

[146]:
```
print(likelihood)
```

```
0.624
```

The likelihood P(F E) is about 0.624, or 62.4%.

9.1.8 5. Total Probability Calculation:

The total probability P(F) is the overall probability of any customer clicking on electronics-related content. This is essentially the average click rate on electronics emails withi nour dataset.

[148]:
```
# Total Probability: P(F) - Overall probability of clicking on electronics email
# This is the average click rate on electronics emails
total_probability = df_bayesian['ClickedOnElectronicsEmail'].mean()
```

[149]:
```
print(total_probability)
```

```
0.312
```

The total probability P(F) is about 0.312, or 31.2%. Next, we'll use these values to compute the posterior probability, which will tell us the probability of a customer being interested in electronics given that they clicked on electronics-related content.

9.1.9 6. Bayes's Theorem:

The final step is to calculate the posterior probability using Bayes's theorem. The posterior probability P(E F) represents the probability of a customer being interested in electronics,given that they have clicked on electronics-related content. Here's the formula and the calculation:

10 Bayes's Theorem Formula

P(E F): Posterior probability (what we want to find) P(F E): Likelihood of clicking on electronics email given they are interested in electronics (likelihood) P(E): Prior probability of being interested in electronics P(F): Total probability of clicking on electronics email

[152]:
```
# Calculating the Posterior Probability using Bayes's Theorem
posterior_probability = (likelihood * prior_probability) / total_probability
```

[153]:
```
print(posterior_probability)
```

```
0.624
```

In our dataset, the posterior probability P(E F) is approximately 0.624, or 62.4%. This means that given a customer clicked on electronics-related content, there is a 62.4% chance that they are interested in electronics. This is a significant increase from the prior probability of 31.2%, indicating that clicking on electronics-related content is a strong indicator of interest in electronics. This completes the Bayesian inference exercise, demonstrating how to use Python to apply Bayes's theorem for marketing analytics. This approach enables marketers to refine their strategies based on updated beliefs about customer preferences, leading to more effective and targeted marketing campaigns.

11 EXERCISE 4.2: A/B TESTING FOR MARKETING CAMPAIGN EVALUATION N

11.1 Objective:

Evaluate the effectiveness of the two marketing campaigns using A/B testing.

11.2 File:

A/B Testing for Marketing Campaign Evaluation

11.3 Tasks:

1. Experimental Design: Understand the design of the A/B test (random assignment, duration, sample size).

2. Statistical Analysis : Calculate key performance metrics for both campaign s. Perform hypothesis testing (e.g., t-test) to determif if there's a statistically significant difference e the effectiveness of the two campaigns.
3. Result Interpretation: Analyze and interpret the res lts of the A/B test.
4. Decision-Making: Make recommendations on which campaign should be adopted based on the test results.

11.4 Steps:

11.4.1 1. Importing Required Libraries:

[154]:
```python
import pandas as pd
```

pandas is used for data manipulation and analysis

11.4.2 2. Loading and Displaying the Data— A/B_Testing_Campaign_Data.csv:

[157]:
```python
# Loading the data for Exercise 2: A/B Testing for Marketing Campaign Evaluation
df_ab_testing = pd.read_csv(r'D:\Visualization\Data Science For Marketers\Iain␣
↪Brown- Mastering Marketing Data␣
↪Science\Datasets\MMDS_c04_Data\AB_Testing_Campaign_Data.csv')
```

[158]:
```python
# Displaying the first few rows of the DataFrame
df_ab_testing.head()
```

[158]:

	CampaignGroup	Impressions	ClickThroughRate	ConversionRate
0	B	342	0.018464	0.030065
1	A	688	0.199685	0.048295
2	A	286	0.162267	0.024056
3	A	548	0.126412	0.019510
4	B	966	0.156015	0.025375

pd.read_csv(): Reads the CSV file into a pandas DataFrame. df_ab_testing.head(): Displays the first five rows of t e DataFrame for an overview of the data structure

11.4.3 Note:

The data consists of campaign groups 'A' and 'B', with metrics that will help us compare the effectiveness of these campaigns. Next, we will separate the data for each campaign group and calculate key performance metrics. Let's proceed with thesecalculations.

11.4.4 3. Separating Campaign Data:

The next step in the A/B testing analysis involves separating the data for each campaign and calculating key performance metrics. Here's how we do it in Python:

We first divide the data into two subsets, one for each campaign group (A and B)

[160]:
```python
# Separating the data for Campaign A and Campaign B
df_campaign_a = df_ab_testing[df_ab_testing['CampaignGroup'] == 'A']
```

```
df_campaign_b = df_ab_testing[df_ab_testing['CampaignGroup'] == 'B']
```

[161]:
```
print(df_campaign_a )
```

```
    CampaignGroup  Impressions  ClickThroughRate  ConversionRate
1               A          688          0.199685        0.048295
2               A          286          0.162267        0.024056
3               A          548          0.126412        0.019510
7               A          157          0.122788        0.027926
9               A          995          0.079514        0.024231
..            …            …               …             …
492             A          645          0.196633        0.010976
494             A          441          0.173325        0.008295
495             A          114          0.174661        0.043141
496             A          920          0.046617        0.013813
498             A          191          0.018883        0.030589

[269 rows x 4 columns]
```

[162]:
```
print(df_campaign_b)
```

```
    CampaignGroup  Impressions  ClickThroughRate  ConversionRate
0               B          342          0.018464        0.030065
4               B          966          0.156015        0.025375
5               B          676          0.010575        0.049421
6               B          732          0.091395        0.033529
8               B          756          0.195478        0.039816
..            …            …               …             …
488             B          469          0.141006        0.012417
491             B          106          0.178744        0.032835
493             B          385          0.013715        0.047898
497             B          698          0.169296        0.003233
499             B          295          0.023172        0.022335

[231 rows x 4 columns]
```

11.5 4. Calculating Key Metrics:

We then calculate the mean 'Click-Through Rate (CTR)' and 'Conversion Rate' for each campaign:

[163]:
```
 # Mean Click-Through Rate (CTR) and Conversion Rate for each campaign
 mean_ctr_a = df_campaign_a['ClickThroughRate'].mean()
mean_ctr_b = df_campaign_b['ClickThroughRate'].mean()
 mean_conversion_rate_a = df_campaign_a['ConversionRate'].mean()
mean_conversion_rate_b = df_campaign_b['ConversionRate'].mean()
```

11.5.1 For Campaign A:

[165]:
```
# Mean Click Through Rate
print( mean_ctr_a)
```

```
0.11084799290718875
```

[166]:
```
# Mean Conversion Rate
print(mean_conversion_rate_a)
```

```
0.026392627281286055
```

Mean 'Click-Through Rate (CTR)': Approximately 0.1108 (or 11.08%) Mean 'Conversion Rate': Approximately 0.0264 (o 2.64%)

11.5.2 For Campaign B:

[167]:
```
# Mean Click Through Rate
print(mean_ctr_b )
```

```
0.10484761175287063
```

[168]:
```
# Mean Conversion Rate:
print(mean_conversion_rate_b)
```

```
0.024285949479124592
```

Mean 'Click-Through Rate (CTR)': Approximately 0.1048 (or 10.48%) Mean 'Conversion Rate': Approximately 0.0243 (o 2.43%)

11.5.3 Note:

These metrics give us an initial indication of each campaign's performance. Next, we will conduct a statistical test (like a ttest) to determine if the differences observed in these metrics between the two campaigns are statistically significant. Let's proceed with this analysis. To determine the statistical significance of the difference observed between the two campaigns, we perform t-tests on both the 'Click-Through Rate (CTR)' and 'Conversion Rate' Here's the breakdown of this part of the analysis:

11.6 5 Performing T-Tests:

We use the ttest_ind function from the scipy.stats module, which performs an independent two-sample t-test. This test compares the means of two independent groups (in this case, Campaign A and Campaign B) to determine if there is a statistically significant difference between them:

[170]:
```
from scipy import stats

# T-test for Click-Through Rates
t_stat_ctr, p_value_ctr = stats.ttest_ind(df_campaign_a['ClickThroughRate'],
 ↪df_campaign_b['ClickThroughRate'])
```

```python
# T-test for Conversion Rates
t_stat_conversion, p_value_conversion = stats.
 ↪ttest_ind(df_campaign_a['ConversionRate'], df_campaign_b['ConversionRate'])
```

stats.ttest_ind(): Conducts the t-test for the mean of two independent samples . t_stat_ctr, p_value_ctr: The t-statistic and p-value f r the 'Click-Through Rate' comparis on. t_stat_conversion, p_value_conversion: The tstatistic and p-value for the 'Conversion Rate' comparison.

11.6.1 T-Test For Click Through Rate:

[171]:
```python
print(t_stat_ctr, p_value_ctr)
```

```
1.2305747102790447 0.2190628866189652
```

11.6.2 T-Test For Conversion Rate:

[172]:
```python
print(t_stat_conversion, p_value_conversion)
```

```
1.654054821659667 0.09874650117071734
```

11.6.3 NOTE:

Based on the data: The t-statistic for the 'Click-Through Rate' comparison i approximately 1.231 with a p-value of about 0.219. The t-statistic for the 'Conversion Rate' comparison is approximately 1.654 with a p-value of about 0.099.

11.7 Interpertation :

The p-values indicate the probability of observing the data if the null hypothesis (no difference between campaigns) is true

For both 'Click-Through Rate' and 'Conversion Rate', the pvalues are greater than the typical alpha level of 0.05, suggesting that we do not have enough evidence to reject the null hypothesis at a 5% significance level. Therefore, based on this data, we cannot conclude that there are statistically significant differences between Campaign A and Campaign B in terms of 'Click-Through Rate' and 'Conversion Rate'.

This completes the statistical analysis part of the A/B testing exercise, showing how to use Python to compare the effectiveness of two marketing campaigns. The results suggest that, in this scenario, there might not be a significant difference in performance between the two campaigns.

[ ]:
```python

```

12 Chapter No.5

13 Predictive Analytics and Machine Learning

14 EXERCISE 5.1: CHURN PREDICTION MODEL

14.1 Objective:

Use the churn_data to train a logistic regression model that predicts customer churn.

14.2 File:

churn_data

14.3 Tasks:

1. Split the "churn_data.csv" dataset into training and validation sets.
2. Train a logistic regression model to predict the bina y dependent variable churn.
3. Make predictions and evaluate the model.

14.4 Steps:

14.5 1. Importing Required Libraries:

[174]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

pandas is used for data manipulation and analysis. train_test_split from sklearn.model_selection is
utility to split datasets into training and test sets. LogisticRegression from sklearn.linear_model i a machine learning model for classification tasks. classification_report from sklearn.metrics provi es a way to evaluate the quality of predictions rom a classification algorith

14.6 2. Loading the Dataset

[176]:
```
churn_data = pd.read_csv('D:\\Visualization\\Data Science For Marketers\\Iain␣
↪Brown- Mastering Marketing Data Science\\Datasets\\MMDS_c05_Data\\churn_data.
↪csv')
```

We load the churn dataset from a CSV file into a pandas DataFrame. The dataset contains features that describecustomer behavior and a target variable that indicate whether the customer has churned

[178]:
```
churn_data.head()
```

[178]:
```
   feature_1  feature_2  feature_3  feature_4  feature_5  feature_6  \
0  -0.669356  -1.495778  -0.870766   1.141831   0.021606   1.730630
1   0.093372   0.785848   0.105754   1.272354  -0.846316  -0.979093
```

```
2 -0.905797 -0.608341  0.295141  0.943716  0.092936  1.370397
3 -0.585793  0.389279  0.698816  0.436236 -0.315082  0.459505
4  1.146441  0.515579 -1.222895 -0.396230 -1.293508 -0.352428

   feature_7  feature_8  feature_9  feature_10  ...  feature_12  feature_13  \
0  -1.251698   0.289305   0.357163   -0.196811  ...    0.154850   -0.219970
1   1.263707   0.264020   2.411677   -0.960046  ...    0.199810    0.288724
2  -0.064772   0.287273  -0.533004   -0.032504  ...   -0.510064   -0.868768
3   1.448820   0.505558  -1.440982   -1.134020  ...    1.466783    0.678728
4   0.071254   1.239584   1.007133   -1.479444  ...   -0.918127    0.604121

   feature_14  feature_15  feature_16  feature_17  feature_18  feature_19  \
0   -0.739137    1.802012    1.634606   -0.938180   -1.267337   -1.276334
1    0.732492   -0.872002   -1.654887   -1.130204   -0.122709    0.693431
2   -0.598279    0.019832    0.613460   -1.779439    0.830498   -0.737332
3   -1.190917   -1.442381   -0.929136   -0.221600   -0.346772    0.034246
4    1.068379   -0.882271    2.303639   -0.973379    1.259233    0.360015

   feature_20  churn
0    1.016643      1
1    0.911363      0
2   -0.578212      1
3   -1.040199      1
4    1.920368      0

[5 rows x 21 columns]
```

14.7 3. Defining Features and Targets:

[179]:
```python
X = churn_data.drop('churn', axis=1)
y = churn_data['churn']
```

[181]:
```python
print(X)
```

```
     feature_1  feature_2  feature_3  feature_4  feature_5  feature_6  \
0    -0.669356  -1.495778  -0.870766   1.141831   0.021606   1.730630
1     0.093372   0.785848   0.105754   1.272354  -0.846316  -0.979093
2    -0.905797  -0.608341   0.295141   0.943716   0.092936   1.370397
3    -0.585793   0.389279   0.698816   0.436236  -0.315082   0.459505
4     1.146441   0.515579  -1.222895  -0.396230  -1.293508  -0.352428
..         ...        ...        ...        ...        ...        ...
995   0.519359   1.874906   0.078118   0.081083   0.201653  -2.756306
996  -0.410935  -0.546608   1.134924   0.334300  -0.618983   0.693425
997  -0.200135  -1.461082   1.797017  -0.244096   0.544323   1.776031
998   0.039356   0.248684  -0.475323  -1.136693   1.942577  -1.297109
999   0.769215   0.470765   0.169945   0.268167  -1.188385  -1.282664

     feature_7  feature_8  feature_9  feature_10  feature_11  feature_12  \
```

```
0   -1.251698  0.289305  0.357163 -0.196811  0.829274  0.154850
1    1.263707  0.264020  2.411677 -0.960046  0.543479  0.199810
2   -0.064772  0.287273 -0.533004 -0.032504 -0.549925 -0.510064
3    1.448820  0.505558 -1.440982 -1.134020 -0.241431  1.466783
4    0.071254  1.239584  1.007133 -1.479444 -0.695695 -0.918127
..        …         …         …         …         …         …
995  0.400236 -1.073689 -0.589452 -1.404240 -1.029972  0.046079
996 -0.617285  1.087727  0.193022  1.461993  0.956549 -1.011037
997 -2.021994 -0.658113  0.206816 -0.114789  0.858663  0.542985
998 -0.802722  0.451323 -1.454615 -0.679222 -0.451375  0.153528
999 -0.160905 -0.215568  0.606795 -0.470850  0.193799  1.027070

     feature_13  feature_14  feature_15  feature_16  feature_17  feature_18  \
0     -0.219970   -0.739137    1.802012    1.634606   -0.938180   -1.267337
1      0.288724    0.732492   -0.872002   -1.654887   -1.130204   -0.122709
2     -0.868768   -0.598279    0.019832    0.613460   -1.779439    0.830498
3      0.678728   -1.190917   -1.442381   -0.929136   -0.221600   -0.346772
4      0.604121    1.068379   -0.882271    2.303639   -0.973379    1.259233
..          …           …           …           …           …           …
995    2.539382   -0.480648   -1.630771   -0.039894    1.673364   -0.134180
996   -0.256734    0.517721    0.593266   -0.629825   -0.080137   -0.246737
997   -0.420264   -0.748275    1.668697   -1.209965   -1.248582   -1.502802
998    0.637119    1.235484    0.780224    1.558384    0.263888    0.099126
999   -1.204330    0.273311    0.222339    2.056586   -0.139595    0.656116

     feature_19  feature_20
0     -1.276334    1.016643
1      0.693431    0.911363
2     -0.737332   -0.578212
3      0.034246   -1.040199
4      0.360015    1.920368
..          …           …
995    1.792044    0.248325
996   -0.486387    2.211333
997   -1.274737    1.601119
998    0.542692    1.208275
999    0.643332   -2.021002

[1000 rows x 20 columns]
```

[183]:
```
print(y)
```

```
0      1
1      0
2      1
3      1
4      0
      ..
```

```
995    0
996    1
997    1
998    0
999    0
Name: churn, Length: 1000, dtype: int64
```

We separate the features (X) and the target (y). The features include all columns except the target column 'churn', which we want to predict. The target is the 'churn' column, which is what our model will learn to predict.

14.8 4. Spliting the Data:

[184]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

The dataset is split into a training set (80%) and a test set (20%) using train_test_split. The test_size parameter dictates the proportion of the dataset to include in the test split. The random_state parameter ensures that the split is reproducible; the same random seed means the split will be the same each time the code is run.

14.9 5. Initializing the "Logistic Regression" Moddel:

[185]:
```
logreg = LogisticRegression()
```

[187]:
```
print(logreg)
```

```
LogisticRegression()
```

An instance of the 'LogisticRegression' model is created. 'LogisticRegression' is chosen because it is a common model for binary classification tasks, similar to predicting churn (yes or no).

14.10 6. Training the Model:

[189]:
```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

[189]:
```
LogisticRegression()
```

The 'LogisticRegression' model is trained on the training data (X_train and y_train). The fit method adjusts the weights of the model to find the best linear boundary that separates the classes.

14.11 7. Making Predictions:

[190]:
```
y_pred = logreg.predict(X_test)
```

[ ]:
```
The trained model is used to make predictions on the test data (X_test). The␣
↪predict method applies the weights
learned during training to the test data to predict the churn outcome.
```

14.12 8. Evaluating the Model:

[193]:
```
classification_report_output = classification_report(y_test, y_pred)
```

Finally, the classification_report function is used to evaluate the predictions. It compares the predicted churn outcomes (y_pred) with the actual outcomes from the test set (y_test). The report provides metrics such as precision, recall, an F1-score that help to understand the performance of th model across the different classes (churned or not churned). The output is printed to provide a clear view of the model s performanc ### NOTE: This entire process constitutes a basic workflow for training and evaluating a binary classification model in machine learning. Each step is crucial for understanding how the model is built and how well it performs on unseen data. e.

[194]:
```
print(classification_report_output)
```

```
              precision    recall  f1-score   support

           0       0.80      0.91      0.85        93
           1       0.91      0.80      0.86       107

    accuracy                           0.85       200
   macro avg       0.86      0.86      0.85       200
weighted avg       0.86      0.85      0.86       200
```

14.12.1 Precision:

This is the ratio of correctly predicted positive observations to the total predicted positives. High precision relates to a low false positive rate. We have precision values of 0.80 for class 0 (non-churn) and 0.91 for class 1 (churn), which indicates the model is more precise in predicting customers who will churn than those who will not. ### Recall (Sensitivity): This is the ratio of correctly predicted positive observations to all observations in the actual class. The recall is 0.91 for class 0 and 0.80 for class 1, indicating the model is more sensitive in predicting the non-churners correctly than churners. ### F1-score: This is the weighted harmonic mean of precision and recall. The F1-score is 0.85 for class 0 and 0.86 for class 1, suggesting that the model is robust in its predictive performance for both classes. ### Support: This is the number of actual occurrences of the class in the specified dataset. For non-churners (class 0), there are 93 instances, and for churners (class 1), there are 107 instances. ### Accuracy: The model has an overall accuracy of 0.85, which means it correctly predicts the churn status 85% of the time on the test set. ### Macro Average: This is the average precision, recall, and F1- score between classes. The macro average does not take class imbalance into account, which is appropriate here because the classes are balanced. ### Weighted Average: This is the average precision, recall, and F1-score between classes weighted by the number of instances in each class. This gives us a better measure of the true quality of the classifier, particularly when there is class imbalance, which is not a significant issue in this dataset.

Overall, with an F1-score of approximately 0.85 for both classes, themodel appears to perform well on this dataset, which suggests it could be a good starting point for making predictions in a real-world scenario.

[ ]:
```

```

15 EXERCISE 5.2: PREDICT WEEKLY SALES

15.1 Objective:

Build a linear regression model to predict weekly sales based on marketing spend and other store features.

15.2 File:

regression_data.csv

15.3 Tasks:

1. Split the "regression_data.csv" dataset into training and validation sets.
2. Train a linear regression model to predict the depende t variable weekly_sales.
3. Make predictions and evaluate the model.

15.4 Steps:

15.5 1. Importing Libraries:

[196]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

pandas is for data manipulation. train_test_split will help divide the data into trainin and testing sets. LinearRegression is the model we will use for prediction. mean_squared_error will be used to evaluate the model's performance.

15.6 2. Loading the Dataset:

[198]:
```
regression_data = pd.read_csv('D:\\Visualization\\Data Science For␣
↪Marketers\\Iain Brown- Mastering Marketing Data␣
↪Science\\Datasets\\MMDS_c05_Data\\regression_data.csv')
```

[201]:
```
regression_data.head()
```

[201]:

	marketing_spend	store_size	location_score	employee_count	weekly_sales
0	-0.386879	1.801382	0.588902	0.675984	87.069451
1	0.337377	-0.354613	1.174833	0.691618	164.287705
2	1.199978	1.314092	-0.105696	-1.809935	-18.457388
3	-0.323398	-0.373742	0.344818	0.169975	23.831031
4	1.521595	0.271376	1.302737	0.738472	245.822611

The regression data is loaded into a pandas DataFrame from a CSV file.

15.7 3. Defining Features and Targets:

[202]:
```
X = regression_data.drop('weekly_sales', axis=1)
y = regression_data['weekly_sales']
```

[207]:
```
X.head()
```

[207]:
```
   marketing_spend  store_size  location_score  employee_count
0        -0.386879    1.801382        0.588902        0.675984
1         0.337377   -0.354613        1.174833        0.691618
2         1.199978    1.314092       -0.105696       -1.809935
3        -0.323398   -0.373742        0.344818        0.169975
4         1.521595    0.271376        1.302737        0.738472
```

[208]:
```
y.head()
```

[208]:
```
0     87.069451
1    164.287705
2    -18.457388
3     23.831031
4    245.822611
Name: weekly_sales, dtype: float64
```

X contains the independent variables (features), which are all columns except 'weekly_sales'. y is the dependent variable (target), which we aim to predit —in this case, 'weekly_sales'.

15.8 4. Splitting the Data:

[209]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,␣
↪random_state=42)
```

[210]:
```
X_train, X_test, y_train, y_test.head()
```

[210]:
```
(     marketing_spend  store_size  location_score  employee_count
 29         -0.544836    0.328546        1.477914       -0.259795
 535         0.424856   -2.121391       -0.595293       -2.354959
 695        -0.253084    0.905638       -0.968834       -1.058971
 557         0.328440   -0.341244        1.642481       -2.188452
 836        -0.335878   -0.167442       -0.779819       -0.732588
 ..               …           …               …               …
 106        -0.672677    0.889682        0.067509        1.420676
 270        -1.456538    0.104721        1.441496       -0.488531
 860        -1.631116    0.197163       -0.336297        1.894135
 435        -1.372901   -1.415385       -0.909908       -0.666511
 102         1.033012    0.551633        0.502330       -0.677427

 [800 rows x 4 columns],
      marketing_spend  store_size  location_score  employee_count
```

```
521         -1.470233   -0.667260       -0.074766       -1.585109
737         -0.038953   -0.912909        0.036234        0.525085
740          0.202818   -0.783709       -0.026283       -1.074035
660          1.241359    0.390340       -0.850886       -0.129830
411         -1.369287   -0.182119       -0.360320       -0.507010
..                …           …              …              …
408         -0.118441   -0.056876       -2.047776        0.366855
332          0.674560    1.289650        0.203642        0.635328
208          1.240764    0.396109        0.229268        0.160417
613          1.433325    0.920004        0.879402       -0.954774
78          -0.603082   -0.404778        1.008926       -0.924853

[200 rows x 4 columns],
29     109.200055
535   -161.984407
695   -147.185971
557     74.948057
836   -127.015507
          …
106     48.078116
270     45.482248
860    -25.202745
435   -201.446211
102     78.595603
Name: weekly_sales, Length: 800, dtype: float64,
521   -161.179410
737     20.684775
740    -46.420477
660    -18.609240
411   -129.337193
Name: weekly_sales, dtype: float64)
```

The dataset is split into training (80%) and testing (20%) sets, with random_state set for reproducibility.

15.9 5. Initializing and Training the Model:

[211]:
```
linreg = LinearRegression()
linreg.fit(X_train, y_train)
```

[211]:
```
LinearRegression()
```

A LinearRegression model instance is created and then fitted (trained) using the training data.

15.10 6. Making Predictions:

[212]:
```
y_pred = linreg.predict(X_test)
```

[214]:
```
y_pred_series = pd.Series(y_pred)
print(y_pred_series.head())
```

```
0   -161.344220
1     20.609766
2    -46.505379
3    -18.739451
4   -129.439133
dtype: float64
```

The model makes predictions (y_pred) on the test data (X_test).

15.11 7. Evaluating the Model:

[216]:
```
mse = mean_squared_error(y_test, y_pred)
```

[218]:
```
mse
```

[218]:
```
0.010637666516564956
```

The computed mean squared error is 0.01064, which is a measure of the model's accuracy. A lower MSE indicates a better fit of the model to the data. Given the low MSE, we can infer that our model has performed well on this dataset. This exercise demonstrates the process of creating and evaluating a predictive model, which is a fundamental aspect of data science in marketing and many other fields. The small MSE suggests that the model's predictions are very close to the actual sales figures, making it a potentially useful tool in a real-world marketing context.

[ ]:
```

```

16 CHAPTER 6

17 Natural Language Processing in Marketing

18 EXERCISE 6.1: SENTIMENT ANALYSIS

18.1 Objective:

Write a Python script to perform sentiment analysis on the provided social media posts.

18.2 File:

sentiment_analysis_data.csv"

18.3 Tasks:

Load the "sentiment_analysis_data.csv" into a Python program. 2. Use a sentiment analysis library TextBlob to analyze t e sentiment of each post. 3. Categorize each post as 'Positive', 'Negative', or 'Neutral' b sed on the sentiment score. 4. Output the sentiment analysis results in a readable format.

18.4 Steps:

18.5 1- Load the Data:

First, import necessary libraries and read the CSV file containing the sentiment data.

[220]:
```
import pandas as pd
sentiment_df = pd.read_csv('D:\\Visualization\\Data Science For Marketers\\Iain␣
↪Brown- Mastering Marketing Data␣
↪Science\\Datasets\\MMDS_c06_Data\\sentiment_analysis_data.csv')
```

[222]:
```
print('sentiment_analysis_data.csv')
```

```
sentiment_analysis_data.csv
```

[ ]:
```
Details of Sentiment_analysis:
```

[224]:
```
sentiment_df
```

[224]:
```
                                                   Post Analyzed_Sentiment
0    I love this new smartphone. It has an amazing …           Positive
1    Really unhappy with the customer service. Very…           Negative
2    This is just an average product. Nothing speci…           Positive
3    Absolutely fantastic! Could not have asked for…           Positive
4       Worst purchase ever. Totally regret buying it.           Negative
..                                                  …                  …
995  I love this new smartphone. It has an amazing …           Positive
996  Really unhappy with the customer service. Very…           Negative
997  This is just an average product. Nothing speci…           Positive
998  Absolutely fantastic! Could not have asked for…           Positive
999     Worst purchase ever. Totally regret buying it.           Negative

[1000 rows x 2 columns]
```

18.6 2. Install and Import TextBlob:

Install TextBlob, if not already installed, using !pip instal textblob. Import TextBlob for sentiment analysis.

[226]:
```
!pip install textblob
from textblob import TextBlob
```

```
Collecting textblob
  Downloading textblob-0.18.0.post0-py3-none-any.whl.metadata (4.5 kB)
```

```
Requirement already satisfied: nltk>=3.8 in c:\users\dell\anaconda3\lib\site-
packages (from textblob) (3.8.1)
Requirement already satisfied: click in c:\users\dell\anaconda3\lib\site-
packages (from nltk>=3.8->textblob) (8.1.7)
Requirement already satisfied: joblib in c:\users\dell\anaconda3\lib\site-
packages (from nltk>=3.8->textblob) (1.2.0)
Requirement already satisfied: regex>=2021.8.3 in
c:\users\dell\anaconda3\lib\site-packages (from nltk>=3.8->textblob) (2023.10.3)
Requirement already satisfied: tqdm in c:\users\dell\anaconda3\lib\site-packages
(from nltk>=3.8->textblob) (4.65.0)
Requirement already satisfied: colorama in c:\users\dell\anaconda3\lib\site-
packages (from click->nltk>=3.8->textblob) (0.4.6)
Downloading textblob-0.18.0.post0-py3-none-any.whl (626 kB)
   ---------------------------------------- 0.0/626.3 kB ? eta -:--:--
   ---------------------------------------- 0.0/626.3 kB ? eta -:--:--
    --------------------------------------- 10.2/626.3 kB ? eta -:--:--
   - -------------------------------------- 30.7/626.3 kB 325.1 kB/s eta 0:00:02
   --- ------------------------------------ 61.4/626.3 kB 469.7 kB/s eta 0:00:02
   ------- ------------------------------- 122.9/626.3 kB 654.9 kB/s eta 0:00:01
   -------- ------------------------------ 133.1/626.3 kB 605.3 kB/s eta 0:00:01
   ------------ -------------------------- 204.8/626.3 kB 731.4 kB/s eta 0:00:01
   -------------- ------------------------ 235.5/626.3 kB 719.7 kB/s eta 0:00:01
   ------------------ -------------------- 297.0/626.3 kB 798.7 kB/s eta 0:00:01
   --------------------- ----------------- 348.2/626.3 kB 864.2 kB/s eta 0:00:01
   ------------------------------ --------- 481.3/626.3 kB 1.0 MB/s eta 0:00:01
   ---------------------------------- ----- 542.7/626.3 kB 1.1 MB/s eta 0:00:01
   ---------------------------------------  614.4/626.3 kB 1.1 MB/s eta 0:00:01
   ---------------------------------------  614.4/626.3 kB 1.1 MB/s eta 0:00:01
   --------------------------------------- 626.3/626.3 kB 939.2 kB/s eta 0:00:00
Installing collected packages: textblob
Successfully installed textblob-0.18.0.post0
```

18.7 3. Perform Sentiment Analysis:

Define a function to analyze sentiment using TextBlob. Apply this function to each post in the dataset

[228]:
```python
from textblob import TextBlob
import pandas as pd

# Assuming sentiment_df is a DataFrame containing a column 'Post' with posts

def analyze_sentiment(post):
    analysis = TextBlob(post)
    return 'Positive' if analysis.sentiment.polarity > 0 else 'Negative' if␣
 ↪analysis.sentiment.polarity < 0 else 'Neutral'
```

```
sentiment_df['Analyzed_Sentiment'] = sentiment_df['Post'].
↪apply(analyze_sentiment)
```

19 Output

[229]:
```
sentiment_df.head()
```

[229]:
```
                                                Post Analyzed_Sentiment
0  I love this new smartphone. It has an amazing …           Positive
1  Really unhappy with the customer service. Very…           Negative
2  This is just an average product. Nothing speci…           Positive
3  Absolutely fantastic! Could not have asked for…           Positive
4     Worst purchase ever. Totally regret buying it.           Negative
```

As you can see, each post has been analyzed by TextBlob, and a sentiment ('Positive', 'Negative', or 'Neutral') has been assigned based on the content of the post. For instance, posts expressingsatisfaction or happiness are labeled as 'Positive', whereas those expressing dissatisfaction or disappointment are labeled as 'Negative'. This demonstrates how sentiment analysis can be used to categorize text data based on the sentiment expressed in it.

20 Exercise 6.2: TEXT PREPROCESSING AND FEATURE EXTRACTION IN MARKETING NATURAL LANGUAGE PROCESSING

[24]:
```
import pandas as pd
classification_df = pd.read_csv('/home/lazzh/Downloads/text_classification_data.
↪csv')
X = classification_df['Review']
y = classification_df['Category']
```

[29]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report

# Step 1: TF-IDF Vectorization
vectorizer = TfidfVectorizer()
X_transformed = vectorizer.fit_transform(X)

# Step 2: Train-test split on transformed data
X_train, X_test, y_train, y_test = train_test_split(X_transformed, y,␣
↪test_size=0.2, random_state=42)

# Step 3: Initialize and train the Multinomial Naive Bayes model
model = MultinomialNB()
```

```
model.fit(X_train, y_train)

# Step 4: Predict on the test set
y_pred = model.predict(X_test)

# Step 5: Evaluate the model
print(classification_report(y_test, y_pred, zero_division=0))
```

```
              precision    recall  f1-score   support

    Clothing       0.00      0.00      0.00        58
 Electronics       0.00      0.00      0.00        64
        Food       0.39      1.00      0.56        78

    accuracy                           0.39       200
   macro avg       0.13      0.33      0.19       200
weighted avg       0.15      0.39      0.22       200
```

21 Chapter No.7

22 Social Media Analytics and Web Analytics

23 EXERCISE 7.1: SOCIAL NETWORK ANALYSI (SNA) IN MARKETING

23.1 Objective:

To understand the application of social network analysis in identifying influential users in a marketing context.

23.2 File;

Social_Network_Analysis_Data.csv

23.3 Tasks:

1. Visualize the Social Network: Use networkx to create a visual representation of th network . Highlight key nodes that might represent influential use rs.
2. Calculate Centrality Measures: Calculate degree, betweenness, and eigenvector centr lity for each node. Identify top five influential users based on these cen rality me asures.
3. Discussion: Discuss how these measures can help in i entifying potential influencers for marketing campaigns. What are the limitations of this approach?mtrix.

23.4 Steps:

23.5 1. Load the Data and Important Libraries:

The following lines will imports the matplotlib and networkx library and loads the social network data from the CSV file into a pandas DataFrame.a

[248]:
```
 import matplotlib.pyplot as plt
import networkx as nx
import pandas as pd

# Corrected file path using raw string or double backslashes for Windows paths
file_path = r"D:\Visualization\Data Science For Marketers\Iain Brown- Mastering␣
↪Marketing Data Science\Datasets\MMDS_c07_Data\Social_Network_Analysis_Data.
↪csv"

# Read the CSV file into a pandas DataFrame
sna_data = pd.read_csv(file_path)
```

[249]:
```
sna_data
```

[249]:

	User	Followers	Engagement Rate
0	0	2832	5.93
1	1	3364	6.03
2	2	9325	6.24
3	3	5974	4.38
4	4	6844	2.73
5	5	805	7.92
6	6	2322	5.68
7	7	9993	8.36
8	8	6316	0.87
9	9	2263	3.68
10	10	7977	7.78
11	11	7699	8.70
12	12	8391	7.99
13	13	855	5.20
14	14	3319	1.18
15	15	7556	5.82
16	16	2845	9.45
17	17	6787	1.06
18	18	8443	2.65
19	19	1307	2.17
20	20	7321	1.50
21	21	3660	6.17
22	22	1741	4.50
23	23	5084	9.02
24	24	8722	3.58

23.6 2. Create the Graph:

[251]:
```
import networkx as nx

G = nx.Graph()

for index, row in sna_data.iterrows():
    G.add_node(row['User'], followers=row['Followers'],␣
 ↪engagement_rate=row['Engagement Rate'])
```

[ ]:
```

```

24 EXERCISE 7.2: WEB ANALYTICS FOR MARKETING INSIGHTS S

24.1 Objective:

To understand how web analytics can be used to gain insights into customer behavior and improve website performance.

24.2 File:

Web_Analytics_Data.csv

24.3 Tasks:

24.4 Data Analysis:

Analyze the user behavior: most visited pages, average time spent per page, bounce rate, and so on. Identify patterns leading to conversion ## Conversion Rate Optimization (CRO): Suggest changes to the website based on the analysis to improve the conversion rate. Discuss how A/B testing could be used to test the e chang ## Discussion: Discuss the role of web analytics in understanding customer behavior. How can these insights be integrated with broad r marketing strategies?es.s.

24.5 Steps:

24.6 1. Load the Web Analytics Data and Important Libraries:

[256]:
```
# This line imports necessary libraries for data analysis and loadsthe web␣
 ↪analytics data from a CSV file into a pandas DataFrame
import seaborn as sns
import pandas as pd

# Use a raw string (prefix with 'r') or double backslashes for the file path
web_analytics_data = pd.read_csv(r'D:\Visualization\Data Science For␣
 ↪Marketers\Iain Brown- Mastering Marketing Data␣
 ↪Science\Datasets\MMDS_c07_Data\Web_Analytics_Data.csv')
```

[257]:
```
web_analytics_data
```

[257]:
```
     User_ID    Session_Timestamp  Page_Visited    Action  Conversion
0          1  2022-01-16 15:00:00      HomePage     Click           1
1         15  2022-01-29 04:00:00   ProductPage     Click           1
2        100  2022-01-22 12:00:00  Confirmation  Purchase           0
3         54  2022-01-21 17:00:00   ProductPage      View           1
4         13  2022-01-21 18:00:00  Confirmation  Purchase           0
..       …                    …           …         …         …
495       36  2022-01-29 21:00:00  Confirmation AddToCart           0
496       67  2022-01-12 20:00:00   ProductPage AddToCart           0
497       21  2022-01-19 16:00:00   ProductPage AddToCart           0
498       46  2022-01-25 09:00:00      HomePage  Purchase           0
499       58  2022-01-17 11:00:00      Checkout      View           1

[500 rows x 5 columns]
```

24.7 2. Convert 'Session Timestamp':

Here, we convert the 'Session_Timestamp' column to datetime format for easier analysis.

[259]:
```python
import pandas as pd

# Assuming web_analytics_data is a Pandas DataFrame
web_analytics_data['Session_Timestamp'] = pd.
 ↪to_datetime(web_analytics_data['Session_Timestamp'])
```

[260]:
```python
web_analytics_data['Session_Timestamp']
```

[260]:
```
0      2022-01-16 15:00:00
1      2022-01-29 04:00:00
2      2022-01-22 12:00:00
3      2022-01-21 17:00:00
4      2022-01-21 18:00:00
               …
495    2022-01-29 21:00:00
496    2022-01-12 20:00:00
497    2022-01-19 16:00:00
498    2022-01-25 09:00:00
499    2022-01-17 11:00:00
Name: Session_Timestamp, Length: 500, dtype: datetime64[ns]
```

25 Analyze User Behavior:

25.1 Most Visited Pages:

[262]:
```python
# Assuming web_analytics_data is a Pandas DataFrame
most_visited_pages = web_analytics_data['Page_Visited'].value_counts()
```

```
# Display the result
print(most_visited_pages)
```

```
Page_Visited
HomePage        138
Confirmation    131
ProductPage     120
Checkout        111
Name: count, dtype: int64
```

25.2 Average Time Spent on Pages:

Here, we simulate the average time spent on each page. We then calculate the average time spent per page. Bounce Rate Calculati

[269]:
```
import numpy as np
import pandas as pd

# Assuming web_analytics_data is a Pandas DataFrame
web_analytics_data['Time_Spent'] = np.random.randint(1, 300,␣
 ↪size=len(web_analytics_data))
avg_time_spent = web_analytics_data.groupby('Page_Visited')['Time_Spent'].mean()

# Display the result
print(avg_time_spent)
```

```
Page_Visited
Checkout        160.585586
Confirmation    144.931298
HomePage        146.094203
ProductPage     147.891667
Name: Time_Spent, dtype: float64
```

[268]:
```
web_analytics_data['Time_Spent']
```

[268]:
```
0      259
1      134
2      272
3      181
4      264
      …
495    213
496    274
497    129
498    168
499     39
Name: Time_Spent, Length: 500, dtype: int32
```

Here, we simulate the average time spent on each page. We then calculate the average time spent per page.

25.3 Bounce Rate Calculation:

We calculate the bounce rate by finding the percentage of sessions where only one page was viewed.

[273]:
```
bounce_rate = web_analytics_data[web_analytics_data['Action'] == 'View'].
 ↪groupby('User_ID').size()
bounce_rate = (bounce_rate == 1).sum() / len(bounce_rate)
```

[275]:
```
print(bounce_rate)
```

```
0.5571428571428572
```

he bounce rate is approximately 55.71%, indicating that more than half of the sessions are single-page sessions.

25.4 Conversion Rate Calculations:

[277]:
```
conversion_rate = web_analytics_data['Conversion'].mean()
```

[278]:
```
print(conversion_rate)
```

```
0.296
```

The conversion rate is about 29.6%, representing the proportion of visits that result in a conversion.

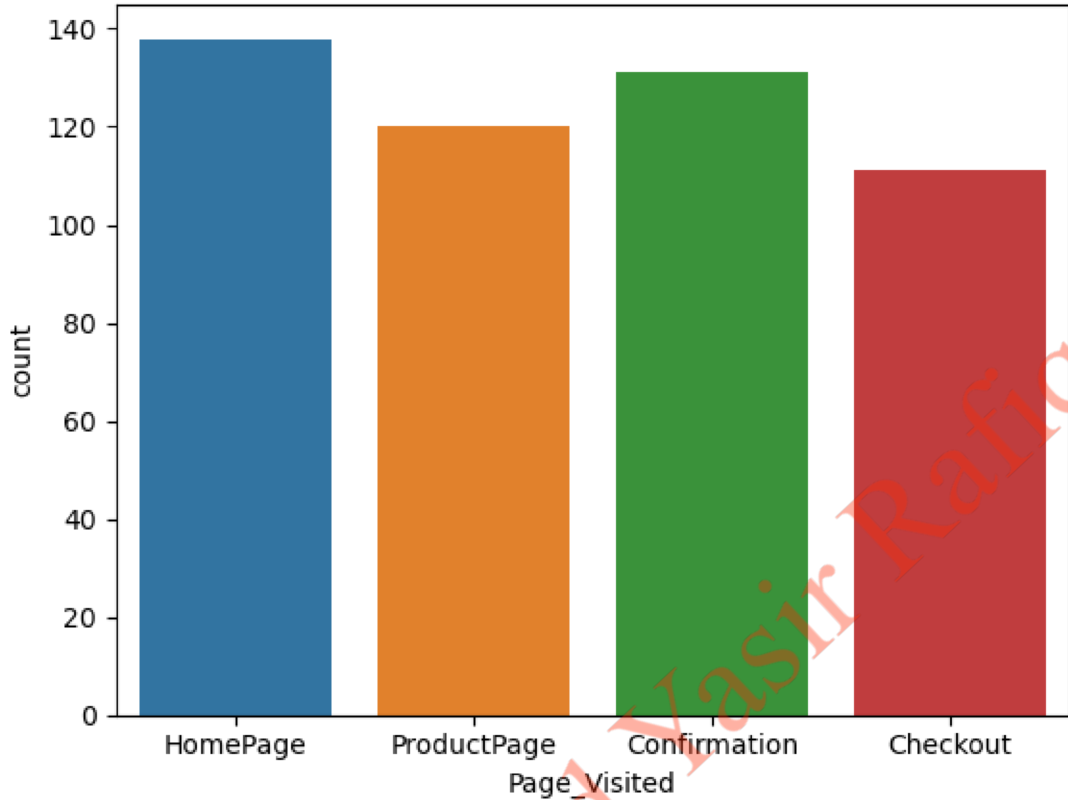
26 Data Visualization

26.1 Page Visits Distribution:

This line creates a count plot showing the distribution of page visits.

[279]:
```
sns.countplot(x='Page_Visited', data=web_analytics_data)
```

[279]:
```
<Axes: xlabel='Page_Visited', ylabel='count'>
```

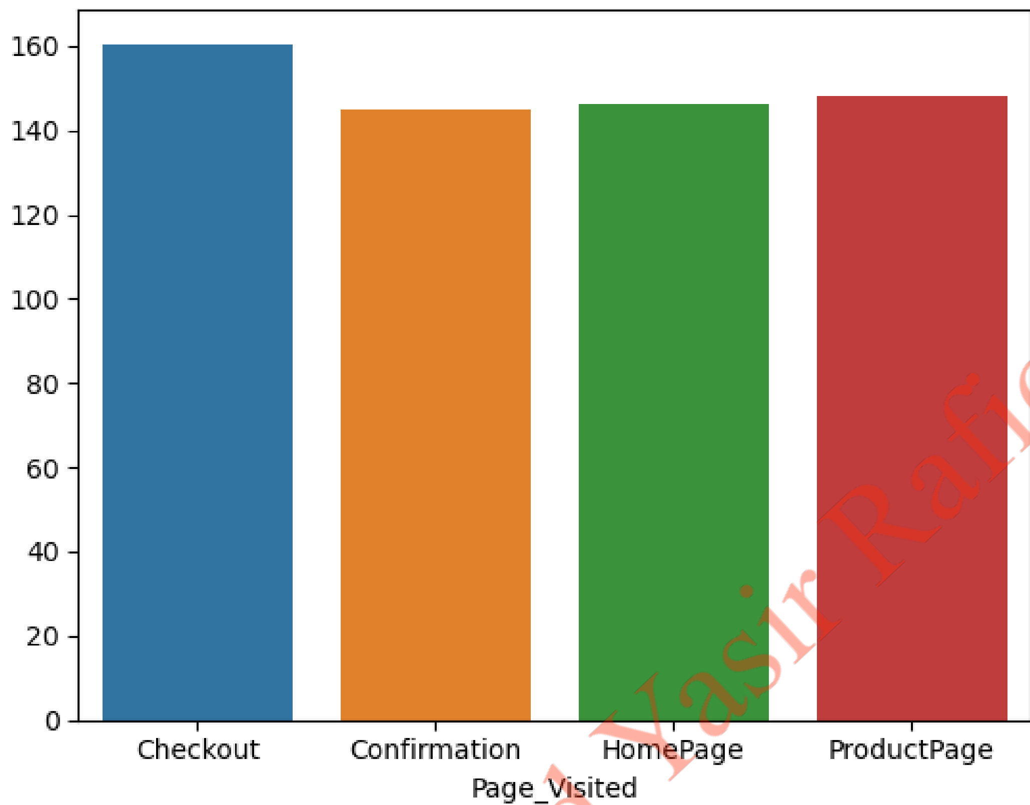


26.2 Average Time Spent on Each Page:

[ ]:
```
This line creates a bar plot showing the average time spent on each page.
```

[280]:
```
sns.barplot(x=avg_time_spent.index, y=avg_time_spent.values)
```

[280]: <Axes: xlabel='Page_Visited'>



[ ]:

27 Chapter No.8

28 Marketing Mix Modeling and Attribution

28.1 EXERCISE 8.1: MARKETING MIX MODELING (MMM)

28.2 Objective:

Develop a multiple regression model to understand the impact of various marketing efforts on a hypothetical company'ssales.

28.3 Tasks:

1- Load the generated data into a DataFrame. 2. Perform exploratory data analysis (EDA) to understand dat distributions and correlations. 3. Build a multiple regression model to analyze the influence of each marketing channel on sales. 4. Interpret the coefficients and evaluate the model's performnce.

28.4 Steps:

28.5 1. Loading Libraries:

First, we need to import the necessary libraries for data manipulation and statistical analysis. pandas is used for data manipulation and analysis. numpy is for numerical operations. statsmodels is for estimating and interpreting models f r statistical analysis.

[281]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
```

28.6 2. Loading the Data:

Next, we load the generated CSV file into a DataFrame. This is where our MMM data reside

[283]:
```
df = pd.read_csv(r'D:\Visualization\Data Science For Marketers\Iain Brown-␣
↪Mastering Marketing Data␣
↪Science\Datasets\MMDS_c08_Data\marketing_mix_modelling_data.csv')
```

We use "pd.read_csv" to read the CSV file and load it into a DataFrame named df.

df.head()

28.7 3. Exploratory Data Analysis:

Before modeling, it's crucial to understand the data. Let's get a quick overview and check for any anomalies or patterns.

[284]:
```
print(df.describe())
print(df.corr())
```

```
            Week   TV_Ad_Spend  Online_Ad_Spend  Radio_Ad_Spend  \
count  52.000000     52.000000        52.000000       52.000000
mean   26.500000  29799.057692     11903.500000     9666.000000
std    15.154757  11076.711672      4658.775096     2981.657928
min     1.000000  10797.000000      2973.000000     5025.000000
25%    13.750000  23901.000000      7729.000000     6900.500000
50%    26.500000  29567.500000     11988.000000     9980.000000
75%    39.250000  37763.500000     16187.250000    11999.500000
max    52.000000  49512.000000     19856.000000    14837.000000

       Promotional_Discount         Sales
count             52.000000     52.000000
mean              12.735979  62416.538462
std                4.483623  22083.884961
min                5.175711  22195.000000
25%                8.269140  44238.000000
50%               13.582176  65040.500000
75%               16.188760  77681.750000
max               19.982705  99835.000000
```

```
                          Week  TV_Ad_Spend  Online_Ad_Spend  Radio_Ad_Spend  \
Week                  1.000000    -0.065692        -0.119148       -0.073583
TV_Ad_Spend          -0.065692     1.000000         0.075289       -0.085106
Online_Ad_Spend      -0.119148     0.075289         1.000000       -0.055763
Radio_Ad_Spend       -0.073583    -0.085106        -0.055763        1.000000
Promotional_Discount -0.142987     0.043252         0.096908       -0.061477
Sales                -0.003600     0.090993         0.037506       -0.289924

                      Promotional_Discount     Sales
Week                             -0.142987 -0.003600
TV_Ad_Spend                       0.043252  0.090993
Online_Ad_Spend                   0.096908  0.037506
Radio_Ad_Spend                   -0.061477 -0.289924
Promotional_Discount              1.000000 -0.056239
Sales                            -0.056239  1.000000
```

28.7.1 Note:

df.describe() provides a statistical summary of the DataFrame, including mean, standard deviation, an quartiles. df.corr() calculates the correlation matrix, helping us understand the relationships between different variables

28.8 4. Preparing the Data for Regression:

[288]:
```python
# We separate the dependent variable ('Sales') and independent variables
↪(marketing spends and discount).
```

[290]:
```python
X = df[['TV_Ad_Spend', 'Online_Ad_Spend', 'Radio_Ad_Spend',
↪'Promotional_Discount']]  # Independent variables
y = df['Sales']  # Dependent variable
```

[292]:
```python
X.head()
```

[292]:
```
   TV_Ad_Spend  Online_Ad_Spend  Radio_Ad_Spend  Promotional_Discount
0        12732            14134            5307             11.431531
1        31243            17115           10302              7.032111
2        40403            10622            6152              9.474235
3        42103            11781           11950             13.549474
4        30757            18298           13467             13.863091
```

[294]:
```python
y.head()
```

[294]:
```
0    55620
1    62521
2    77191
3    77852
4    42302
Name: Sales, dtype: int64
```

28.9 5. Adding a constant to the Model:

For our regression model, we add a constant to the independent variables. This is a typical step in linear regression to include an intercept in the model.

[296]:
```
X = sm.add_constant(X)
```

28.10 6. Building the Model:

Now, we use ordinary least squares (OLS) regression to model the relationship between the independent and dependent variables.

[297]:
```
model = sm.OLS(y, X).fit()
```

sm.OLS is used to create an OLS regression model. fit() is then called on this model to fit it to the data.

28.11 7. Viewing the Regression Results:

Finally, we print the summary of our regression model to see the coefficients and other statistical measures. model.summary() provides a detailed summary of the regression results, including coefficients, R-squared value, p-values, and so on.

[298]:
```
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.095
Model:                            OLS   Adj. R-squared:                  0.018
Method:                 Least Squares   F-statistic:                     1.233
Date:                Wed, 26 Jun 2024   Prob (F-statistic):              0.310
Time:                        18:09:35   Log-Likelihood:                -590.82
No. Observations:                  52   AIC:                             1192.
Df Residuals:                      47   BIC:                             1201.
Df Model:                           4
Covariance Type:            nonrobust
================================================================================
========
                      coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--------
const            8.258e+04   1.79e+04      4.611      0.000    4.66e+04
1.19e+05
TV_Ad_Spend         0.1358      0.279      0.488      0.628      -0.424
0.696
Online_Ad_Spend     0.1138      0.663      0.172      0.864      -1.220
1.448
Radio_Ad_Spend     -2.1305      1.034     -2.060      0.045      -4.211
-0.050
```

```
Promotional_Discount  -390.0831    688.168     -0.567      0.574   -1774.498
994.332
==============================================================================
Omnibus:                        2.688   Durbin-Watson:                   1.873
Prob(Omnibus):                  0.261   Jarque-Bera (JB):                1.493
Skew:                          -0.029   Prob(JB):                        0.474
Kurtosis:                       2.172   Cond. No.                     2.06e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.06e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

28.11.1 Regression Model Results:

28.11.2

Coefficients:TV_Ad_Spend, Online_Ad_Spend Radio_Ad_Spend, and Promotional_Discoun t had coefficients of 0.1358, 0.1138, -2.1305, an -390.0831 respectively It's notable that Radio_Ad_Spend had a negative coefficient, indicating a potential negative impact n sales for each unit increase in radio ad spendin g### . Model F it: The R-squared value of the model was 0.018, whic is quite low. This suggests that the model explains on y a small portion of the variability in the sales data. The F-statistic and its associated p-value indicate that the model is not statistically significant at a convent onal significance le### vel. Interpret ation: The model's low explanatory power (R-squared) nd the lack of statistical significance (p-value of the F-sta istic) suggest that the model may not be the best fit f r this data. It could be due to the nature of the synthet c data or the possibility that the relationship betwee these variables and sales is not inear. The negative coefficient for Radio_Ad_Spe d might imply that radio advertising is not effective or this particular dataset or there are other confounding factors not accounted for in the model. In real-world scenarios, such findings would lead t further investigations, perhaps considering additional variables, exploring nonlinear models, or refining data collection methods. This exercise is valuable for understanding the process of building and interpreting a marketing mix modehough the synthetic nature of the data may limit the real-world applicability of these specific findings.l, even t

[ ]:

```

```

29 EXERCISE 8.2: DATA-DRIVEN ATTRIBUTION

30 Objective:

Analyze customer journey data to attribute conversions to different marketing touchpoints using a probabilistic model.

30.0.1 Tasks:

1. Load the generated data into a DataFrame.

2. Perform data preprocessing to structure the touchpoints data.
3. Apply a probabilistic model to assign conversion credit to ea h touchpoint.
4. Analyze the results to identify which touchpoints have the ost significant influence on conversons.

30.0.2 File:

data_driven_attribution_data.csv

30.1 Steps:

30.2 1. Loading Libraries:

As with the previous exercise, we begin by importing necessary libraries. pandas: is for data manipulation. MultiLabelBinarize:r from sklearn.preprocessing i used to transform the touchpoint data into a binary form t suitable for modeling. LogisticRegres:sion from sklearn.linear_model is for performing the 'LogisticRegression' model.

[299]:
```
import pandas as pd
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.linear_model import LogisticRegression
```

30.3 2. Loading the Data:

As with the previous exercise, we begin by importing necessary libraries.

[302]:
```
df_attribution = pd.read_csv('D:\\Visualization\\Data Science For␣
↪Marketers\\Iain Brown- Mastering Marketing Data␣
↪Science\\Datasets\\MMDS_c08_Data\\data_driven_attribution_data.csv')
```

[305]:
```
df_attribution.head(15)
```

[305]:
```
    Customer_ID                                   Touchpoints  Conversion
0             1     Direct Visit, Email, Social Media, Online Ad           0
1             2     Online Ad, Direct Visit, Social Media, Email           0
2             3  Social Media, Online Ad, Email, Direct Visit, …          0
3             4  Search Ad, Social Media, Direct Visit, Email, …          0
4             5                 Direct Visit, Email, Social Media           0
5             6   Online Ad, Social Media, Search Ad, Direct Visit           1
6             7          Search Ad, Online Ad, Email, Direct Visit           0
7             8                                       Direct Visit           1
8             9                                          Search Ad           1
9            10   Online Ad, Social Media, Search Ad, Direct Visit           0
10           11                                       Social Media           1
11           12  Social Media, Search Ad, Email, Direct Visit, …          1
12           13                                          Online Ad           1
13           14  Direct Visit, Email, Social Media, Online Ad, …          0
14           15  Online Ad, Social Media, Search Ad, Direct Vis…          0
```

30.4 3. Preprocessing the Data:

The 'Touchpoints' column contains lists of touchpoints, which need to be transformed into a format that can be used for modeling

[307]:
```python
import pandas as pd
from sklearn.preprocessing import MultiLabelBinarizer

# Splitting the touchpoint strings into lists
df_attribution['Touchpoints'] = df_attribution['Touchpoints'].apply(lambda x: x.
 ↪split(','))

# Using MultiLabelBinarizer to transform the touchpoint lists into binary format
mlb = MultiLabelBinarizer()
touchpoints_binary = mlb.fit_transform(df_attribution['Touchpoints'])

# Creating a DataFrame for the binary touchpoints
df_touchpoints = pd.DataFrame(touchpoints_binary, columns=mlb.classes_)
```

[308]:
```python
print(df_attribution['Touchpoints'])
```

```
0      [Direct Visit,  Email,  Social Media,  Online Ad]
1      [Online Ad,  Direct Visit,  Social Media,  Email]
2      [Social Media,  Online Ad,  Email,  Direct Vis…
3      [Search Ad,  Social Media,  Direct Visit,  Ema…
4                  [Direct Visit,  Email,  Social Media]
                             …
495                                          [Search Ad]
496       [Online Ad,  Search Ad,  Social Media,  Email]
497                               [Online Ad,  Search Ad]
498    [Direct Visit,  Search Ad,  Online Ad,  Social…
499       [Email,  Social Media,  Search Ad,  Online Ad]
Name: Touchpoints, Length: 500, dtype: object
```

[309]:
```python
df_attribution['Touchpoints'].head()
```

[309]:
```
0    [Direct Visit,  Email,  Social Media,  Online Ad]
1    [Online Ad,  Direct Visit,  Social Media,  Email]
2    [Social Media,  Online Ad,  Email,  Direct Vis…
3    [Search Ad,  Social Media,  Direct Visit,  Ema…
4                [Direct Visit,  Email,  Social Media]
Name: Touchpoints, dtype: object
```

[310]:
```python
df_touchpoints.head()
```

[310]:
```
    Direct Visit   Email   Online Ad   Search Ad   Social Media  Direct Visit  \
0              0       1           1           0              1             1
1              1       1           0           0              1             0
2              1       1           1           1              0             0
```

```
3              1       1            1            0               1             0
4              0       1            0            0               1             1

   Email  Online Ad  Search Ad  Social Media
0      0          0          0             0
1      0          1          0             0
2      0          0          0             1
3      0          0          1             0
4      0          0          0             0
```

30.5 4. Preparing the Model:

Now, prepare the data for logistic regression analysis, which we will use for attribution.

[312]:
```
X = df_touchpoints  # Independent variables (binary touchpoints)
y = df_attribution['Conversion']  # Dependent variable (conversion)
```

[313]:
```
X.head(5)
```

[313]:
```
   Direct Visit  Email  Online Ad  Search Ad  Social Media  Direct Visit  \
0             0      1          1          0             1             1
1             1      1          0          0             1             0
2             1      1          1          1             0             0
3             1      1          1          0             1             0
4             0      1          0          0             1             1

   Email  Online Ad  Search Ad  Social Media
0      0          0          0             0
1      0          1          0             0
2      0          0          0             1
3      0          0          1             0
4      0          0          0             0
```

[314]:
```
y.head()
```

[314]:
```
0    0
1    0
2    0
3    0
4    0
Name: Conversion, dtype: int64
```

30.6 5. Building and Fitting the 'LogisticRegression' Model:

With the data prepared, we can build and fit the 'LogisticRegression' model.

[315]:
```
model = LogisticRegression()
model.fit(X, y)
```

[315]: LogisticRegression()

30.7 6. Interpreting the Model Coefficients:

The coefficients from the logistic regression will help us understand the impact of each touchpoint on the likelihood of conversion.

[316]:
```
coefficients = pd.DataFrame({"Touchpoint":
mlb.classes_, "Coefficient": model.coef_[0]})
```

30.8 7. Sorting the Coefficients:

To better interpret the results, we sort the touchpoints by their coefficients. Higher coefficients suggest a greater positive impact on conversion. Let's execute this code to analyze the touchpoints and their influence on conversion in our synthetic dataset.

[318]:
```
sorted_coefficients = coefficients.sort_values(by="Coefficient",␣
 ↪ascending=False)
```

[320]:
```
print(sorted_coefficients)
```

```
     Touchpoint  Coefficient
7     Online Ad     0.408950
9  Social Media     0.193783
2     Online Ad     0.094825
4  Social Media     0.015479
6         Email    -0.018835
1         Email    -0.034554
0  Direct Visit    -0.164658
8     Search Ad    -0.276661
5  Direct Visit    -0.306552
3     Search Ad    -0.364225
```

30.9 Interpertation:

The sorted coefficients represent the impact of each touchpoint on the likelihood of conversion. A positive coefficient suggests a positive influence on conversion, and a negative coefficient suggests a negative influence.

1- Touchpoints and Their Coefficients: Online Ad: Coefficient of 0.210482. This indicates the strongest positive influence on conversion among the touchpoints. Social Media: Coefficient of 0.082429. This also positively influences conversion but to a lesser extent than online ads. Email: Coefficient of −0.079067. This touchpoint seems to have a slight negative influence on conversion. Direct Visit: Coefficient of −0.290814. This indicates a negative influence on conversion, more so than email. Search Ad: Coefficient of −0.381058. This has the most significant negative impact on conversion among the touchpoints.

2- Interpertations: Positive Influence: The positive coefficients for 'Online Ad' and 'Social Media' suggest that these touchpoints are effective in driving conversions in the synthetic dataset.Negative Influence: 'Email', 'Direct Visit', and 'Search Ad' showing negative coefficients indicate that these touchpoints may be less effective or even counterproductive in leading to conversions in this specific dataset.

Positive Influence: The positive coefficients for 'Online Ad' and 'Social Media' suggest that these touchpoints are effective in driving conversions in the synthetic dataset.Negative Influence: 'Email', 'Direct Visit', and 'Search Ad' showing negative coefficients indicate that these touchpoints may be less effective or even counterproductive in leading to conversions in this specific dataset.

This exercise, with its focus on logistic regression for attribution, highlights the potential of data-driven methods in understanding customer journeys and optimizing marketing touchpoints for better conversion outcomes.

[ ]:
```

```


31 Chapter No.9

32 Customer Journey Analytics

33 XERCISE 9.1: CREATING A CUSTOMER JOURNEY MAPP

33.1 Objective:

Understand the process of customer journey mapping by creating a synthetic map for a fictitious company (ZaraTech).

33.1.1 File:

Customer_Journey_Map_Data.csv

33.1.2 Tasks:

1. Persona Development: Create detailed profiles for each customer persona, including their goals, challenges, an preferences.
2. Touchpoint Identification: List all possible interacti ns these personas might have with ZaraTech across diffe ent channels.
3. Journey Mapping: Create a journey map for each pe sona. Include stages such as 'Awareness', 'Consideration', 'Pur hase', and 'Loyalty.' Plot touchpoints and potential emotions or pain points at each stage.
4. Analysis: Identify key moments of truth and pain oints for each persona.

33.1.3 Steps:

33.2 1. Import Necessary Liberaries:

andas: Used for data manipulation and analysis. matplotlib.pyplot and seaborn: Used for dat visualization.

[323]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

```python
import seaborn as sns
```

33.3 2. Load the Data:

[325]:
```python
import pandas as pd

file_path = 'D:\\Visualization\\Data Science For Marketers\\Iain Brown-␣
↪Mastering Marketing Data␣
↪Science\\Datasets\\MMDS_c09_Data\\Customer_Journey_Map_Data.csv'  # Replace␣
↪with the correct file path
journey_map_df = pd.read_csv(file_path)
```

[326]:
```python
journey_map_df.head(10)
```

[326]:
```
             Persona        Touchpoint          Stage     Emotion  \
0    Tech Enthusiast           Website      Awareness     Curious
1  Busy Professional      Social Media  Consideration  Interested
2       Student Gamer   Email Marketing       Purchase    Decisive
3    Tech Enthusiast          In-Store        Loyalty   Satisfied
4  Busy Professional  Customer Service  Post-Purchase   Supported
5       Student Gamer           Website      Awareness     Curious
6    Tech Enthusiast      Social Media  Consideration  Interested
7  Busy Professional   Email Marketing       Purchase    Decisive
8       Student Gamer          In-Store        Loyalty   Satisfied
9    Tech Enthusiast  Customer Service  Post-Purchase   Supported

               Pain Point
0      Website navigation
1  Overload of information
2         Payment process
3    Product availability
4           Response time
5      Website navigation
6  Overload of information
7         Payment process
8    Product availability
9           Response time
```

We load the CSV file into a DataFrame using pandas. Ensure the file path is correct.

33.4 3. Exploratory Data Analysis:

[329]:
```python
# Display the first few rows of the DataFrame
print(journey_map_df.head())

# Get a summary of the dataset
print(journey_map_df.describe(include='all'))
```

```
             Persona        Touchpoint          Stage     Emotion  \
0    Tech Enthusiast           Website      Awareness     Curious
1  Busy Professional      Social Media  Consideration  Interested
2       Student Gamer   Email Marketing       Purchase    Decisive
3    Tech Enthusiast          In-Store        Loyalty   Satisfied
4  Busy Professional  Customer Service  Post-Purchase   Supported

                Pain Point
0       Website navigation
1  Overload of information
2          Payment process
3     Product availability
4            Response time
                Persona Touchpoint      Stage  Emotion          Pain Point
count                15         15         15       15                  15
unique                3          5          5        5                   5
top     Tech Enthusiast    Website  Awareness  Curious  Website navigation
freq                  5          3          3        3                   3
```

33.5 4. Analyze Emotions and Pain Points:

Grouping data by 'Persona' and 'Emotion' to see the distribution of emotions for each persona. Similarly, grouping by 'Persona' and 'Pain Point' to understand common pain points.

[331]:
```python
# Count of emotions per persona
emotion_count = journey_map_df.groupby(['Persona', 'Emotion']).size().unstack()
print(emotion_count)

# Count of pain points per persona
pain_point_count = journey_map_df.groupby(['Persona', 'Pain Point']).size().
 ↪unstack()
print(pain_point_count)
```

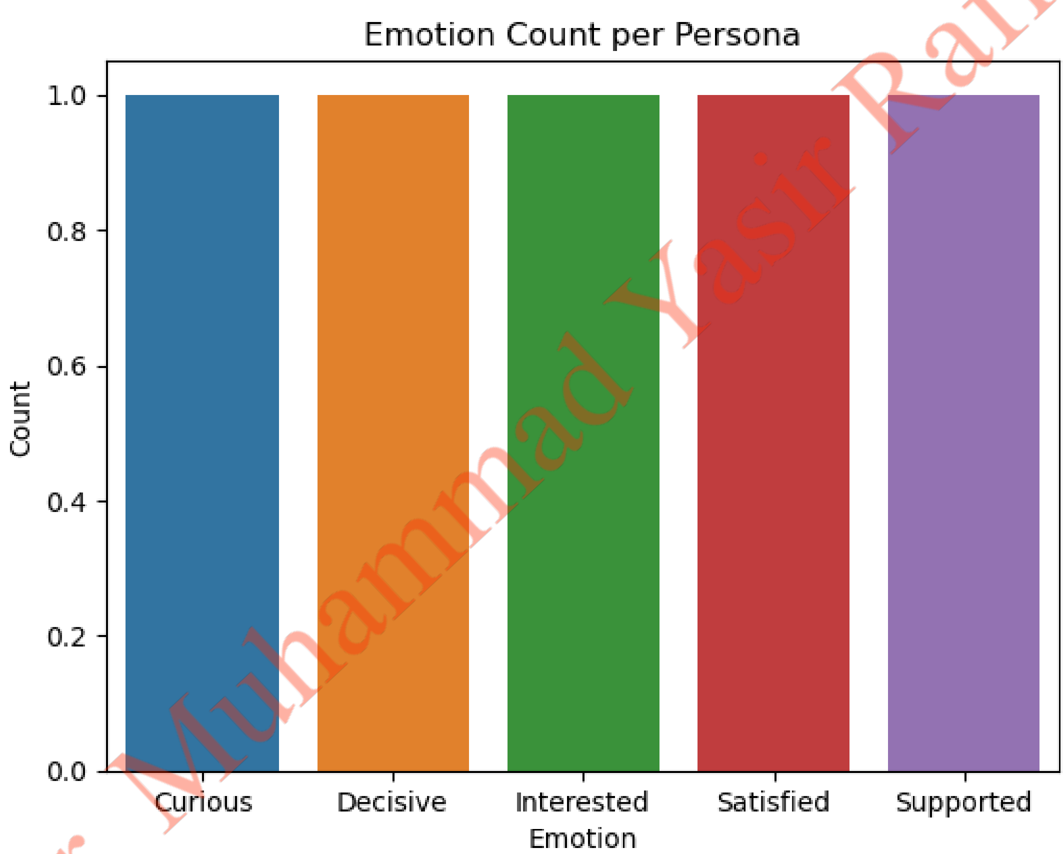
```
Emotion            Curious  Decisive  Interested  Satisfied  Supported
Persona
Busy Professional        1         1           1          1          1
Student Gamer            1         1           1          1          1
Tech Enthusiast          1         1           1          1          1
Pain Point         Overload of information  Payment process  \
Persona
Busy Professional                        1                1
Student Gamer                            1                1
Tech Enthusiast                          1                1

Pain Point         Product availability  Response time  Website navigation
Persona
Busy Professional                     1              1                   1
Student Gamer                         1              1                   1
```

```
Tech Enthusiast                         1              1                    1
```

33.6 Visualize the Data:

Using seaborn to create bar plots for emotion and pain point counts These plots will help visualize which emotions and pain points are most common for each persona..

[333]:
```python
# Plotting emotion count
sns.barplot(data=emotion_count)
plt.title("Emotion Count per Persona")
plt.ylabel("Count")
plt.show()
```

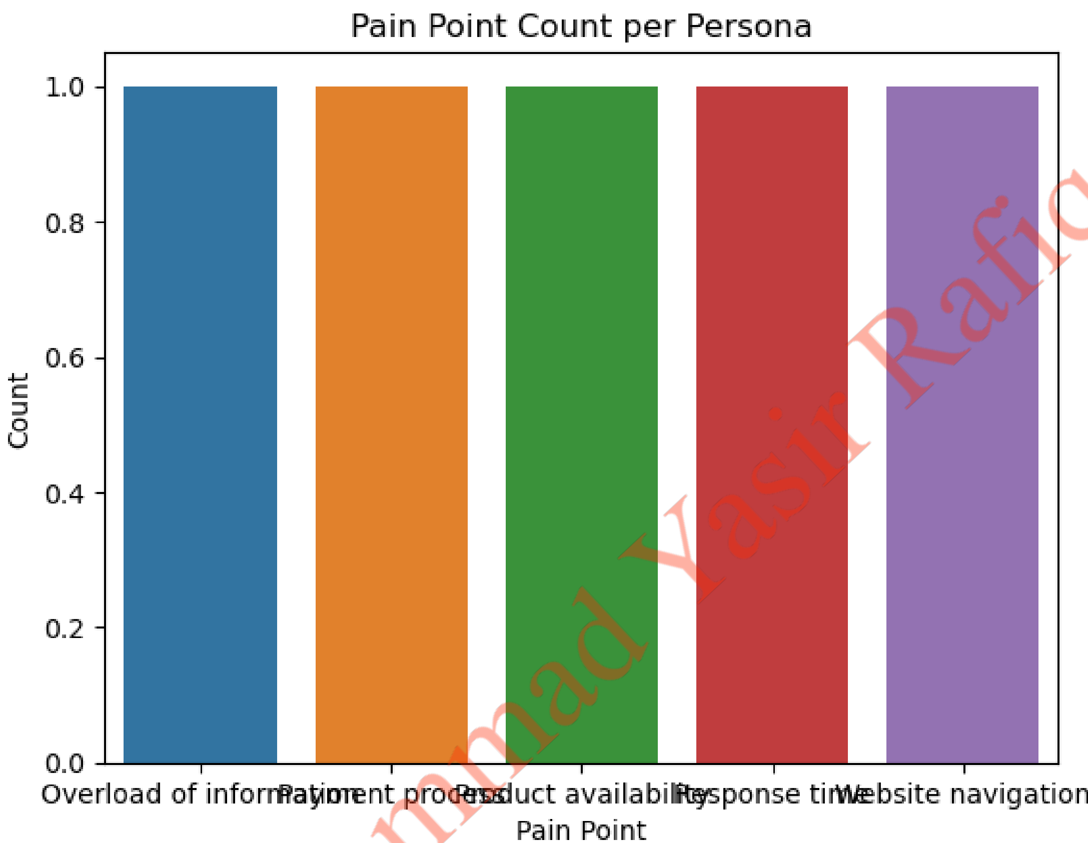


The emotion count per persona shows a uniform distribution across the different personas for each emotion. he bar plot visualizes the count of different emotions for each persona. Each persona experiences each emotion once, indicating a balanced representation in the data.

33.6.1 Pain Point Count per Persona:

This bar plot shows the count of different pain points for each persona. Similar to emotions, each pain point is also uniformly representedacross personas.

[335]:
```
sns.barplot(data=pain_point_count)
plt.title("Pain Point Count per Persona")
plt.ylabel("Count")
plt.show()
```



These analyses and visualizations provide insights into how different personas interact with various touchpoints, along with their emotional responses and pain points. This information is crucial for tailoring customer experience strateg

34 EXERCISE 9.2: TOUCHPOINT EFFECTIVE ANALYSISS

34.1 Objective:

Analyze the effectiveness of different touchpoints in a customer journey.

34.2 File:

Touchpoint_Effectiveness_Analysis_Data.csv

34.2.1 Tasks

1. Data Collection: Use the synthetic data to simulate customer interactions across different touchpoints.
2. Metric Analysis: Calculate the effectiveness of ea h touchpoint. For example, determine conversion rates for webs te visits and email campaigns.
3. Insight Generation: Identify which touchpoints are most effective in driving customer satisf ction and conversions.
4. Strategy Formulation: Based on the analysis, uggest improvements or strategic shifts for ZaraTech to enhance customer experience.

34.3 Steps:

34.4 1. mport Necessary Libraries:

pandas is used for data manipulation and analysis. matplotlib.pyplot and seaborn are used for dat visualization.

[337]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

34.5 2. Create Synthetic Data:

We load the CSV file into a DataFrame using pandas. Ensure the file path is correct

[339]:
```
# Correct file path
file_path = r'D:\Visualization\Data Science For Marketers\Iain Brown- Mastering␣
↪Marketing Data␣
↪Science\Datasets\MMDS_c09_Data\Touchpoint_Effectiveness_Analysis_Data.csv'

# Read the data into a DataFrame
touchpoint_df = pd.read_csv(file_path)
```

34.6 3. Data Overview

[340]:
```
print(touchpoint_df.head())
```

```
                Touchpoint  Customer Satisfaction Scores  Conversion Rates  \
0           Website Visits                            88               5.0
1         Email Open Rates                            75              10.5
2  Social Media Engagement                            82               4.2
3          In-store Visits                            90               7.8
4   Customer Service Calls                            78               3.6

   Repeat Visits/Purchases
0                       40
1                       25
2                       30
```

```
3                        50
4                        20
```

[ ]:
```
The first few rows of the dataset provide a quick look at the structure, with␣
↪each row representing a touchpoint and
associated metrics like customer satisfaction scores, conversion rates, and␣
↪repeat visits/p
```

34.7 4. Analyze the Data

[ ]:
```
touchpoint_df.describe(): Provides a statistical summary of the dataset.
```

[341]:
```
# Descriptive statistics of the dataset
print(touchpoint_df.describe())
```

```
       Customer Satisfaction Scores  Conversion Rates  Repeat Visits/Purchases
count                      5.000000          5.000000                 5.000000
mean                      82.600000          6.220000                33.000000
std                        6.387488          2.883054                12.041595
min                       75.000000          3.600000                20.000000
25%                       78.000000          4.200000                25.000000
50%                       82.000000          5.000000                30.000000
75%                       88.000000          7.800000                40.000000
max                       90.000000         10.500000                50.000000
```

[ ]:
```
The summary of the dataset gives us a statistical overview. It shows that the␣
↪average customer satisfaction score is about
82.6, with a mean conversion rate of 6.22% and an average of 33% for repeat␣
↪visits/purchases
```

34.8 5. Visualize the Data

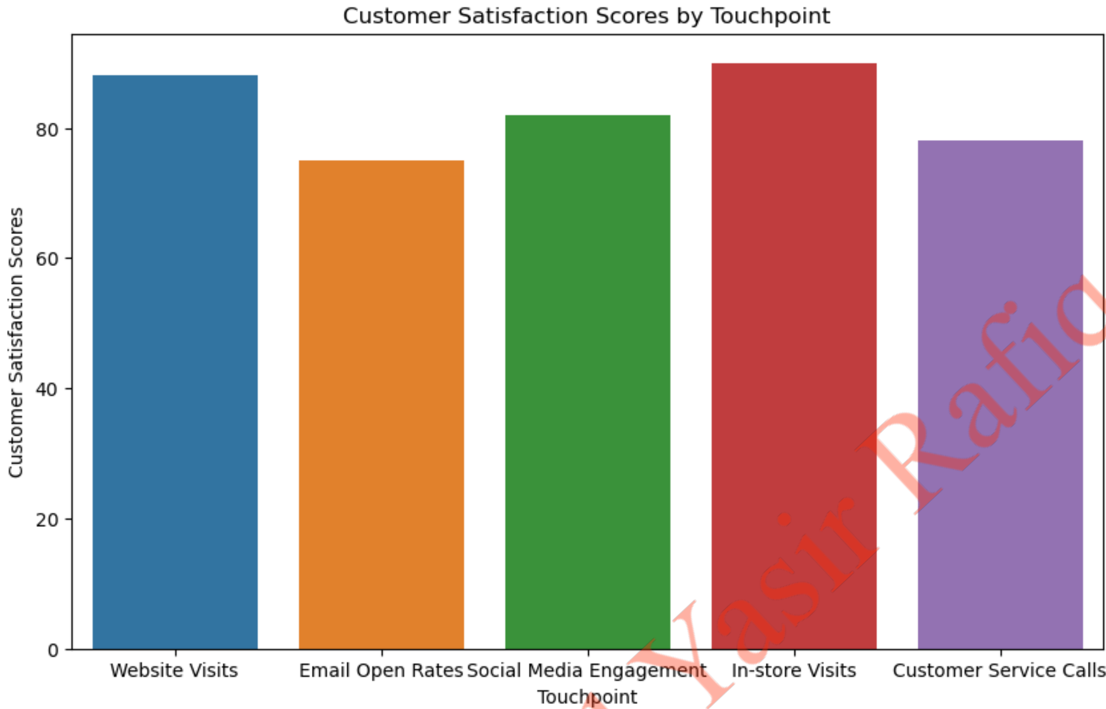
[ ]:
```
We'll create visualizations for each metric to better understand their␣
↪distribution and effectiveness:
```

34.8.1 Customer Satisfaction Scores by Touchpoint

The bar plot shows the customer satisfaction scores for each touchpoint. 'In-store Visits' have the highest satisfaction score, indicating a strong performance in this area.

[343]:
```
# Visualizing Customer Satisfaction Scores
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.barplot(x='Touchpoint', y='Customer Satisfaction Scores',␣
↪data=touchpoint_df)
plt.title('Customer Satisfaction Scores by Touchpoint')
```

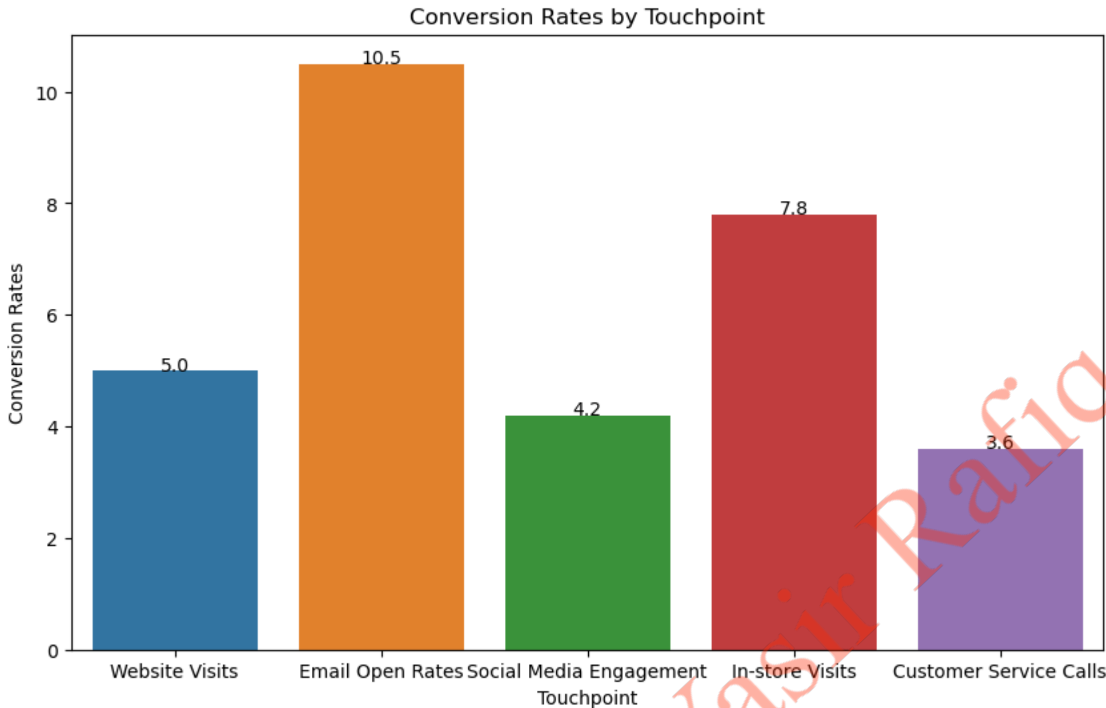
```
plt.show()
```



34.9 Conversion Rates by Touchpoint:

This plot visualizesnthe conversion rates associated with each touchpoint.'Email Open Rates' stand out with the highest conversion rate, suggesting that email marketing is particularly effective for this synthetic dataset.

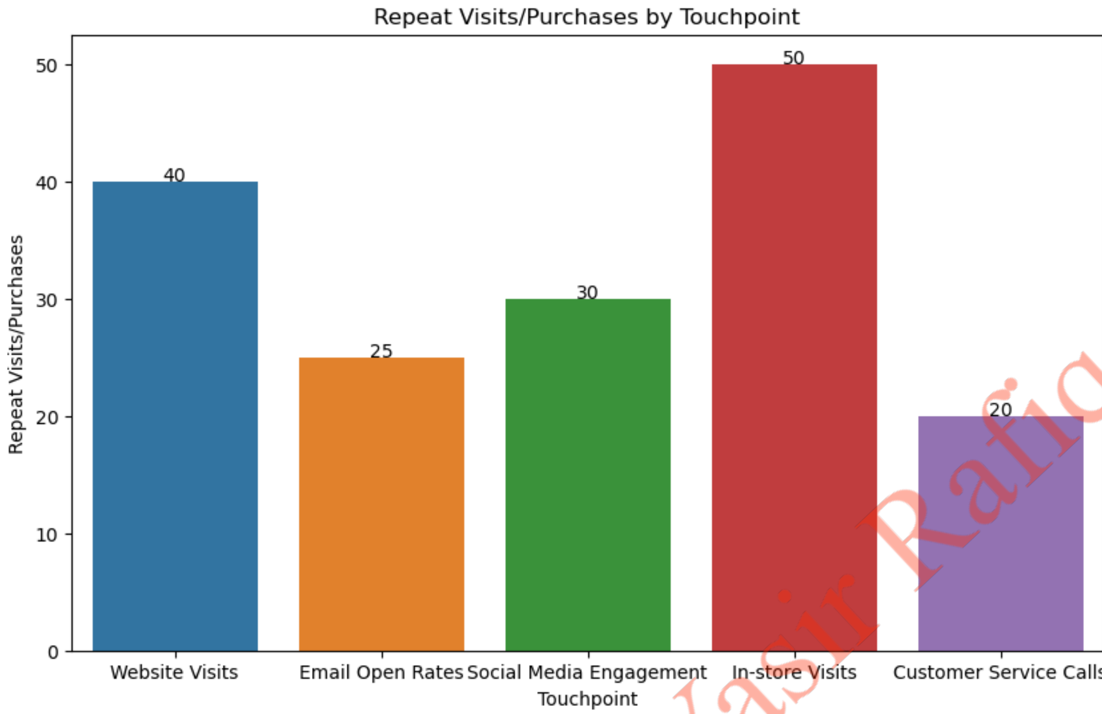
[347]:
```
plt.figure(figsize=(10, 6))
bar_plot = sns.barplot(x='Touchpoint', y='Conversion Rates', data=touchpoint_df)
plt.title('Conversion Rates by Touchpoint')

# Adding numbers on each bar
for index, row in touchpoint_df.iterrows():
    bar_plot.text(index, row['Conversion Rates'], round(row['Conversion␣
 ↪Rates'], 2), color='black', ha="center")

plt.show()
```



34.10 Repeat Visits/Purchases by Touchpoint

The final plot shows the percentage of repeat visits or purchases for each touchpoint. 'In-store Visits' again show a strong performance, indicating that customers who visit the store are more likely to return or make repeat purchases.

[349]:
```
plt.figure(figsize=(10, 6))
barplot = sns.barplot(x='Touchpoint', y='Repeat Visits/Purchases',␣
↪data=touchpoint_df)

# Add numbers on each bar
for index, row in touchpoint_df.iterrows():
    barplot.text(row.name, row['Repeat Visits/Purchases'], row['Repeat Visits/
↪Purchases'], color='black', ha="center")

plt.title('Repeat Visits/Purchases by Touchpoint')
plt.show()
```



These visualizations provide a clear understanding of how each touchpoint is performing in terms of customer satisfaction, conversion, and customer retention. Such insights are crucial for businesses to identify which areas are working well and which need improvement or further investment.

[ ]:

35 Chapter NO.10

36 Experimental Design in Marketing

36.1 Objective:

To demonstrate the basic principles of experimental design using an A/B test scenario in email marketing.

36.1.1 File:

Email_Marketing_AB_Test_Data.csv

36.1.2 Tasks;

You are provided with data from an email marketing campaign where two different subject lines were tested to see which one yields a higher open rate. Your task is to analyze the data to determine which subject line performed better. 1. Statistical Test: Perform a t-test to see if the difference i open rates between the two groups is statistically significant. 2. Interpret Results: Based on the p-value from the t-te t, conclude which subject line perfo med better.

36.2 Steps:

36.3 1. Import Libraries:

We import two libraries: scipy.stats for statistical tests and pandas for handling data in a structured form (DataFrames).

[351]:
```
import scipy.stats as stats
import pandas as pd
```

36.4 2. Load the Data:

We load the data into a pandas DataFrame. This data simulates the open rates of emails for two different subject lines (Group A and Group B).

[353]:
```
email_marketing_data = pd.read_csv(r'D:\Visualization\Data Science For␣
↪Marketers\Iain Brown- Mastering Marketing Data␣
↪Science\Datasets\MMDS_c10_Data\Email_Marketing_AB_Test_Data.csv')
```

[356]:
```
print(email_marketing_data)
```

```
    Group  OpenRate
0       A  0.288203
1       A  0.220008
2       A  0.248937
3       A  0.312045
4       A  0.293378
..    …         …
195     B  0.241423
196     B  0.288590
197     B  0.291175
198     B  0.358162
199     B  0.316826

[200 rows x 2 columns]
```

We load the data into a pandas DataFrame. This data simulates the open rates of emails for two different subject lines (Group A and Group B).

36.5 3. Separate the Data into TWO Groups:

Here, we filter the DataFrame to create two separate series: one for each group. group_A contains the open rates for subject line A, and group_B for subject line B

[358]:
```
group_A = email_marketing_data[email_marketing_data['Group'] == 'A']['OpenRate']
group_B = email_marketing_data[email_marketing_data['Group'] == 'B']['OpenRate']
```

[359]:
```python
group_A.head()
```

[359]:
```
0    0.288203
1    0.220008
2    0.248937
3    0.312045
4    0.293378
Name: OpenRate, dtype: float64
```

[360]:
```python
group_B.head()
```

[360]:
```
100    0.344158
101    0.182612
102    0.186476
103    0.298470
104    0.191344
Name: OpenRate, dtype: float64
```

36.6 4. Perform a t-Test:

We perform an independent t-test (ttest_ind) to compare the mean open rates of the two groups. This test will help us determine if there is a statistically significant difference between the two subject lines' open rates.

[362]:
```python
t_stat, p_value = stats.ttest_ind(group_A, group_B)
```

[365]:
```python
print(t_stat)
```

```
-7.041427369013264
```

[366]:
```python
print(p_value)
```

```
3.059820094514218e-11
```

The t_stat is the calculated t-statistic value, and the p_value is the probability of observing a value as extreme as the t-statistic under the null hypothesis. In this case, the p-value is extremely low (way below the typical threshold of 0.05), suggesting that there is a statistically significant difference between the open rates of Group A and Group B.

In conclusion, based on this analysis, we can confidently say that the open rates of the two subject lines are significantly different. If Group B's mean open rate is higher, it implies that subject line B was more effective in this email marketing campaign.

[ ]:
```python

```

37 EXERCISE 10.2: FRACTIONAL FACTORIAL DESIGN IN AD OPTIMIZATION N

37.1 Objective:

To illustrate the application of fractional factorial designs in optimizing an advertising campaign.

37.2 File:

Ad_Optimization_Fractional_Factorial_Design_Data.csv

37.3 Tasks:

You are given a dataset from an online advertising experiment with several factors (such as ad color, placement, and size) and their levels. Your task is to analyze the data to determine the optimal combination of these factors for maximum click-through rate. 1. Factorial Analysis: Use regression analysis to understand the impact of each factor and their interactions on the click-throug rate. 2. Optimization: Identify the combination of factors that leads to the highest predicted click-through rte.

37.4 Steps:

37.5 1. Import Libraries and Load Data:

We import statsmodels for regression analysis and pandas for data manipulation. Then, we load the data into a pandas DataFrame.

[3]:
```
import pandas as pd
file_path = '/home/lazzh/Downloads/
 ↪Ad_Optimization_Fractional_Factorial_Design_Data.csv'
ad_optimization_data = pd.read_csv(file_path)
```

37.6 2. Creare Dummy Variables:

Because our data contains categorical variables (AdColor, Placement, Size), we convert them into dummy variables for regression analysis. The drop_first=True argument is used toavoid multicollinearity by dropping the first level of each categorical variable

[4]:
```
ad_data_dummies = pd.get_dummies(ad_optimization_data,drop_first=True)
```

37.7 3. Prepare Data for Regression:

We separate the independent variables (X) and the dependent variable (‘ClickThroughRate’, y). We also add a constant to the model, which acts as the intercept in the regression equation.

[6]:
```
import statsmodels.api as sm
X = ad_data_dummies.drop('ClickThroughRate', axis=1)
y = ad_data_dummies['ClickThroughRate']
X = sm.add_constant(X)
```

37.8 4. Fit the Regression Model:

We use ordinary least squares (OLS) regression to fit the model. This method finds the best-fitting line through the data by minimizing the sum of the squares of the vertical deviations from each data point to the line.

[7]:
```
print(X)
```

```
    const  AdColor_Green  AdColor_Red  Placement_Top  Size_Small
0     1.0          False         True          False        True
1     1.0           True        False          False       False
2     1.0          False        False          False        True
3     1.0           True        False          False        True
4     1.0          False        False          False        True
..    …              …          …              …          …
95    1.0          False         True          False       False
96    1.0          False         True           True       False
97    1.0           True        False           True       False
98    1.0          False         True           True       False
99    1.0          False         True          False        True

[100 rows x 5 columns]
```

[8]:
```
print(y)
```

```
0     0.165182
1     0.355505
2     0.296122
3     0.495764
4     0.126122
        …
95    0.370497
96    0.451694
97    0.317471
98    0.213080
99    0.112094
Name: ClickThroughRate, Length: 100, dtype: float64
```

[12]:
```
import pandas as pd
import statsmodels.api as sm
import numpy as np

# Load your data into a DataFrame
ad_data_dummies = pd.read_csv(file_path)

# Check for categorical columns
categorical_cols = ad_data_dummies.select_dtypes(include=['object']).columns

# Convert categorical columns to dummy variables
```

```python
ad_data_dummies = pd.get_dummies(ad_data_dummies, columns=categorical_cols,␣
    ↪drop_first=True)

# Define X and y
X = ad_data_dummies.drop('ClickThroughRate', axis=1)
y = ad_data_dummies['ClickThroughRate']

# Convert X and y to float64
X = X.astype(np.float64)
y = y.astype(np.float64)

# Add constant to X
X = sm.add_constant(X)

# Convert X and y to numpy arrays
X = X.values
y = y.values

# Fit the OLS model
model = sm.OLS(y, X).fit()

# Print the summary of the model
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.015
Model:                            OLS   Adj. R-squared:                 -0.026
Method:                 Least Squares   F-statistic:                    0.3724
Date:                Fri, 28 Jun 2024   Prob (F-statistic):              0.828
Time:                        17:52:08   Log-Likelihood:                 76.824
No. Observations:                 100   AIC:                            -143.6
Df Residuals:                      95   BIC:                            -130.6
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2808      0.025     11.077      0.000       0.230       0.331
x1             0.0188      0.031      0.613      0.541      -0.042       0.080
x2             0.0134      0.027      0.492      0.624      -0.041       0.068
x3             0.0137      0.024      0.572      0.569      -0.034       0.061
x4             0.0174      0.023      0.752      0.454      -0.029       0.063
==============================================================================
Omnibus:                       21.039   Durbin-Watson:                   1.832
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                5.287
Skew:                           0.139   Prob(JB):                       0.0711
Kurtosis:                       1.908   Cond. No.                         4.68
```

```
==============================================================================

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

[ ]:

[ ]:

[ ]: