Project Name: Multi-Channel Customer Behaviour & Marketing Analytics

Contents

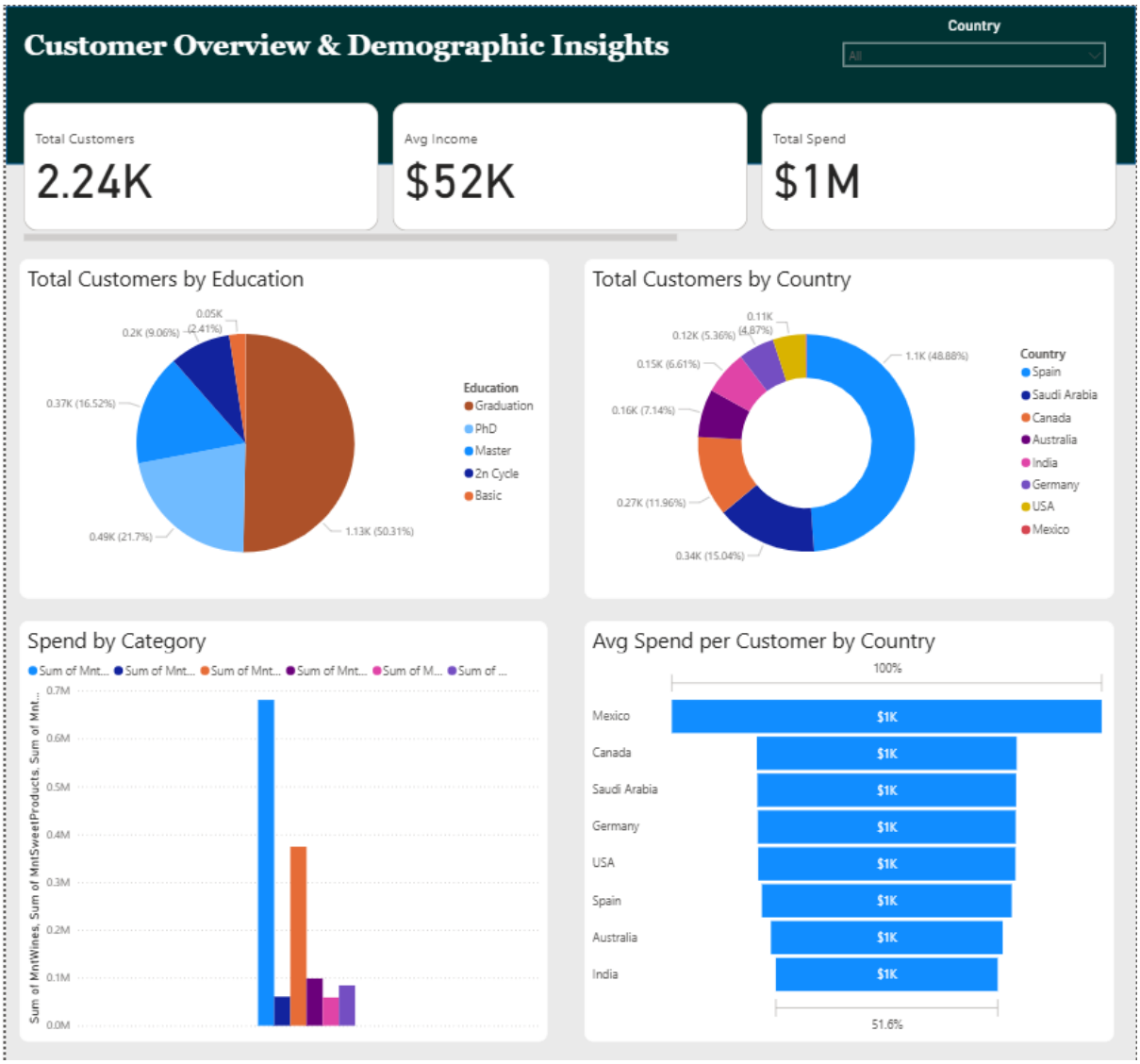
Part A: Load the data in the tool. Briefly explain the dataset

1. Power BI dashboard

1.1 Summary

This Power BI dashboard provides end-to-end insights into customer demographics, purchasing behaviour across different channels, and marketing campaign effectiveness. The analysis is divided into three interactive pages, each focusing on a key business area.

1.2 Page 1: Customer Overview & Demographic Insights



This page gives a high-level understanding of the customer base:

- Total Customers, Avg Income, and Total Spend KPIs show the overall market size and customer value.

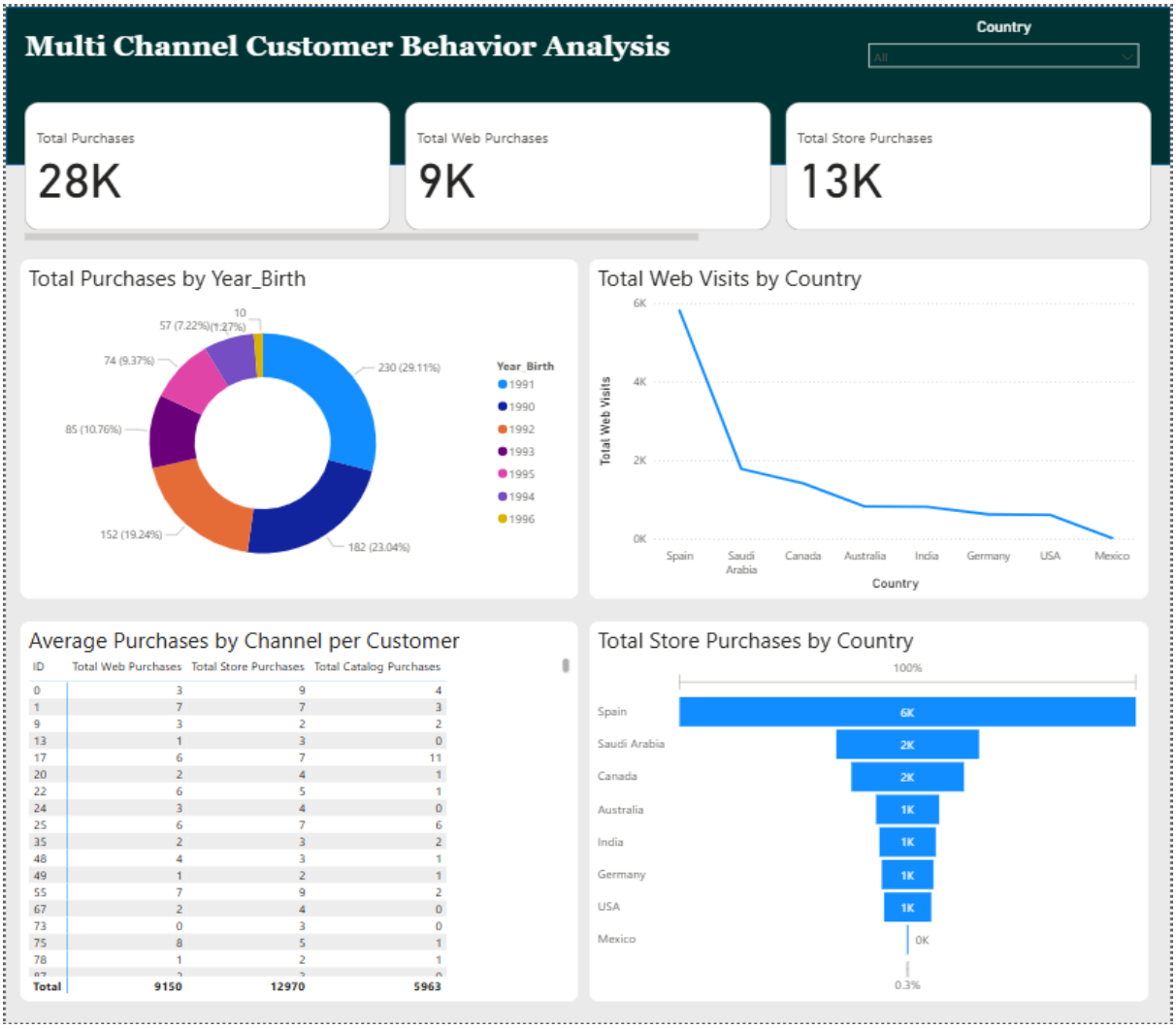
- Customers are segmented by education level and country, enabling demographic profiling for targeted marketing.
- The Spend by Product Category visual reveals which product lines generate the most revenue e.g., wines and meat products show strong customer demand.
- Average Spend per Customer by Country helps identify high-value regions for future campaign targeting.

1.2.1 Purpose

This page answers:

- Who are our customers?
- Where are they located?
- Which product categories contribute most to revenue?

1.3 Page 2: Multi-Channel Customer Behavior Analysis



This page focuses on how customers interact with different purchasing channels, Web, Store, and Catalog.

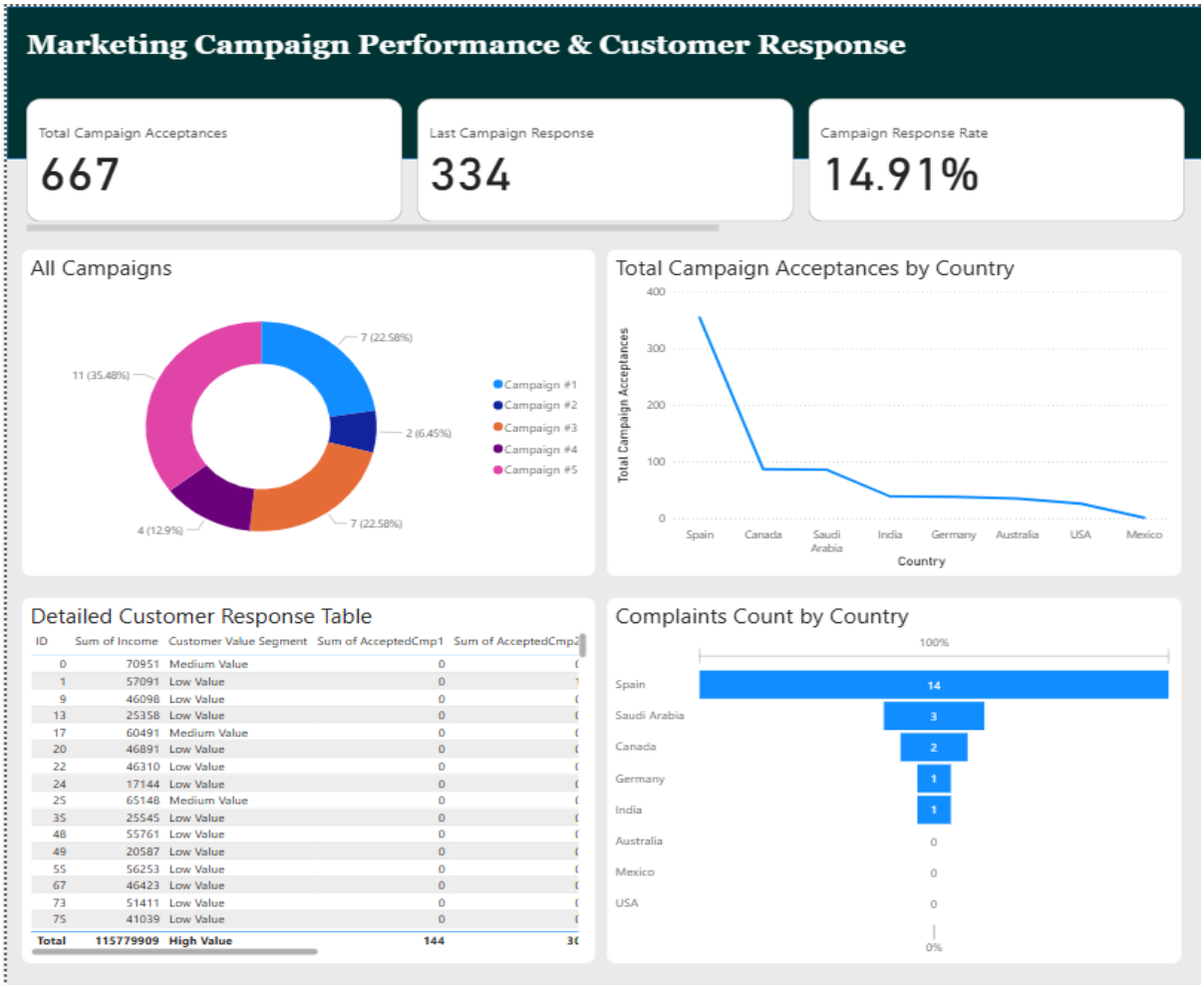
- KPIs for Total Purchases, Web Purchases, and Store Purchases highlight the dominance or weakness of each channel.
- Purchases by Year of Birth helps identify age-based behavioural differences.
- Web Visits vs Web Purchases shows engagement and conversion rate, indicating how well the website turns visitors into buyers.
- Purchases by Country per Channel reveals regional channel preferences — for example, Spain shows leading store-based engagement.

1.3.1 Purpose

This page answers:

- Which channel performs best?
- Which customer age groups and regions prefer web, store, or catalog shopping?
- How effective are digital interactions?

1.4 Page 3: Marketing Campaign Performance & Customer Response



This page evaluates the impact of marketing campaigns and customer satisfaction.

- KPI cards reflect total campaign acceptances, response count, and overall response rate, indicating the success of marketing efforts.
- The campaign donut chart compares the performance of Campaigns 1 to 5, helping identify the most effective promotions.
- Campaign Acceptances by Country highlights regional responsiveness to marketing programs.
- The Customer Response Table allows drill-down by value segment to identify top-responding customer groups.
- Complaints Count by Country helps monitor service quality and identify dissatisfaction trends.

1.4.1 Purpose

This page answers:

- Which campaigns are most successful?
- Which customer segments respond better to offers?
- Where do most complaints come from?

Part B: Perform Exploratory Data Analysis (EDA) using Python

2. Perform Exploratory Data Analysis

2.1 Import necessary libraries

```
import pandas as pd

import numpy as np
```

This code imports essential Python libraries: **pandas** for data manipulation and analysis, and **numpy** for numerical operations.

2.2 Load the Dataset

```
df = pd.read_csv('marketing_data.csv')

df.head()
```

This code uploads the dataset into the Colab environment and loads it into a DataFrame named df to preview the first few rows for initial inspection.

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	NumStorePurchases	NumWebVisits
0	1826	1970	Graduation	Divorced	84835.0	0	0	2014-06-16	0	189	...	6	
1	1	1961	Graduation	Single	57091.0	0	0	2014-06-15	0	464	...	7	
2	10476	1958	Graduation	Married	67267.0	0	1	2014-05-13	0	134	...	5	
3	1386	1967	Graduation	Together	32474.0	1	1	2014-05-11	0	10	...	2	
4	5371	1989	Graduation	Single	21474.0	1	0	2014-04-08	0	6	...	2	

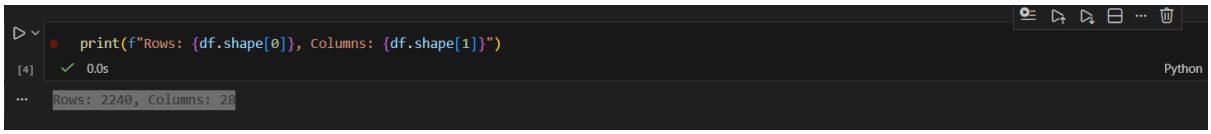
5 rows × 28 columns

2.3 Check Dataset Shape

```
print(f"Rows: {df.shape[0]}, Columns: {df.shape[1]}")
```

This line displays the dataset's dimensions, revealing it contains 2,240 rows and 28 columns, indicating a moderately sized dataset suitable for exploratory analysis

Result:



2.4 Check Column Names and Data Types

```
df.info()
```

This command provides an overview of the dataset’s structure, including column names, data types, and non-null counts, helping identify missing values and understand the type of data in each column.

Result:

```
df.info()
```

```
[5]  ✓ 0.0s
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 28 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   2240 non-null   int64  
 1   Year_Birth           2240 non-null   int64  
 2   Education            2240 non-null   object 
 3   Marital_Status       2240 non-null   object 
 4    Income              2216 non-null   float64
 5   Kidhome              2240 non-null   int64  
 6   Teenhome             2240 non-null   int64  
 7   Dt_Customer          2240 non-null   object 
 8   Recency              2240 non-null   int64  
 9   MntWines             2240 non-null   int64  
 10  MntFruits            2240 non-null   int64  
 11  MntMeatProducts      2240 non-null   int64  
 12  MntFishProducts      2240 non-null   int64  
 13  MntSweetProducts     2240 non-null   int64  
 14  MntGoldProds         2240 non-null   int64  
 15  NumDealsPurchases    2240 non-null   int64  
 16  NumWebPurchases      2240 non-null   int64  
 17  NumCatalogPurchases  2240 non-null   int64  
 18  NumStorePurchases    2240 non-null   int64  
 19  NumWebVisitsMonth    2240 non-null   int64  
...
 26  Complain             2240 non-null   int64  
 27  Country              2240 non-null   object 
dtypes: float64(1), int64(23), object(4)
memory usage: 490.1+ KB
```

2.5 Summary Statistics of Numerical Columns

```
df.describe()
```

This code generates summary statistics (like mean, standard deviation, min, and max) for all numerical columns, offering insights into data distribution and potential outliers.

Result:

	ID	Year_Birth	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	...
count	2240.000000	2240.000000	2216.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	...
mean	5592.159821	1968.805804	52247.251354	0.444196	0.506250	49.109375	303.935714	26.302232	166.950000	37.525446	...
std	3246.662198	11.984069	25173.076661	0.538398	0.544538	28.962453	336.597393	39.773434	225.715373	54.628979	...
min	0.000000	1893.000000	1730.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	2828.250000	1959.000000	35303.000000	0.000000	0.000000	24.000000	23.750000	1.000000	16.000000	3.000000	...
50%	5458.500000	1970.000000	51381.500000	0.000000	0.000000	49.000000	173.500000	8.000000	67.000000	12.000000	...
75%	8427.750000	1977.000000	68522.000000	1.000000	1.000000	74.000000	504.250000	33.000000	232.000000	50.000000	...
max	11191.000000	1996.000000	666666.000000	2.000000	2.000000	99.000000	1493.000000	199.000000	1725.000000	259.000000	...

8 rows × 24 columns

2.6 Missing Values- Total and Percentage

```
missing = df.isnull().sum()

percent = (missing / len(df)) * 100

missing_df = pd.DataFrame({'Missing Values': missing, 'Percentage': percent})

missing_df[missing_df['Missing Values'] > 0]
```

This code identifies columns with missing values and calculates their percentage, helping to assess data quality and determine whether imputation or removal is needed.

2.7 Duplicate Check

```
df.duplicated().sum()
```

Ensures unique customer records, avoids skewed statistics.

Part C: Data wrangling/cleansing etc. Explain each step

3. Data wrangling/cleansing

3.1 Handle Missing Values

```
for col in df.select_dtypes(include='object').columns:
    df[col] = df[col].fillna(df[col].mode()[0])
```

This code fills missing values in all categorical (object-type) columns with their most frequent value (mode), ensuring no null values remain in these fields for further analysis.

3.2 Remove Duplicate Rows

```
print("Duplicates before:", df.duplicated().sum())

df.drop_duplicates(inplace=True)

print("Duplicates after:", df.duplicated().sum())
```

This code checks for and removes duplicate rows from the dataset to ensure data integrity and prevent skewed analysis due to repeated records.

3.3 Convert Date Columns

```
df['Year_Birth'] = pd.to_datetime(df['Year_Birth'])

print(df['Year_Birth'])
```

This code converts the 'Year_Birth' column to datetime format, enabling more accurate date-based operations or age calculations in further analysis.

3.4 Feature Engineering- Create Age Groups

```
if 'Age' in df.columns:
    df['AgeGroup'] = pd.cut(df['Age'], bins=[0, 25, 45, 65, 100],
                            labels=['Youth', 'Adult', 'Middle-Aged', 'Senior'])
    df[['Age', 'AgeGroup']].head()
```

Creates customer segments for targeted marketing.

3.5 Encode Categorical Variables

```
df_encoded = pd.get_dummies(df, drop_first=True)

df_encoded.head()
```

Convert categories into number

3.6 Outlier Detection Using Z-Score

```
from scipy.stats import zscore

z_scores = np.abs(zscore(df.select_dtypes(include='number')))

outliers = (z_scores > 3).sum(axis=0)

print("Outliers per column:\n", outliers)
```

Detects unusual values that may distort model learning.

Part D: Build multiple charts using Python

4. Explore the data by implementing several data visualizations

4.1 Import Visualization Libraries

```
import matplotlib.pyplot as plt

import seaborn as sns
```

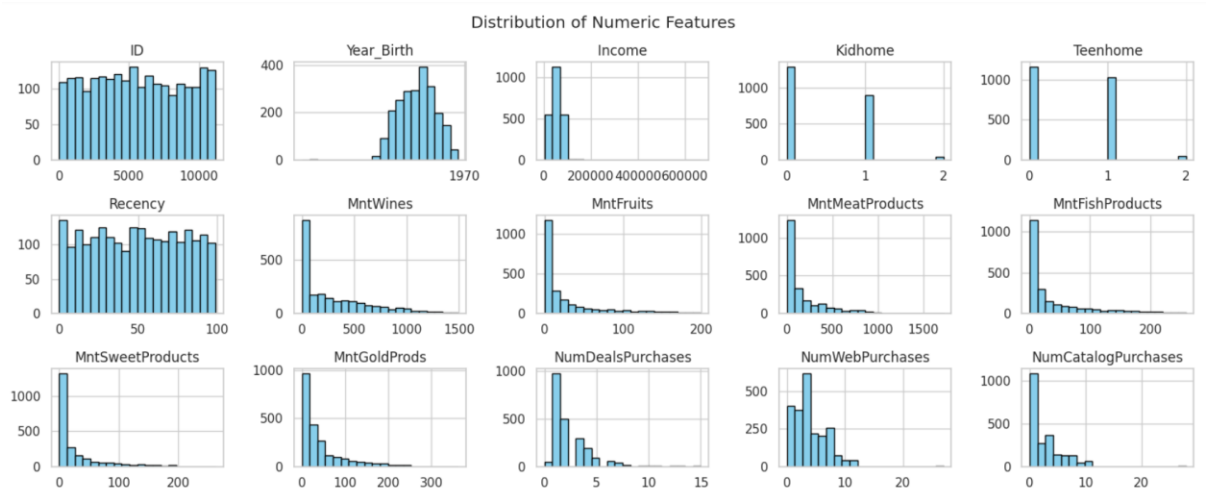
This code imports **matplotlib** and **seaborn** for data visualization and sets a clean whitegrid style to enhance the readability of plots.

4.2 Histogram for Numeric Columns

```
df.hist(figsize=(15, 10), bins=20, color='skyblue', edgecolor='black')

plt.suptitle("Distribution of Numeric Features")

plt.tight_layout()

plt.show()
```

This code generates histograms for all numeric columns, allowing visual inspection of their distributions, skewness, and potential outliers.

Result:

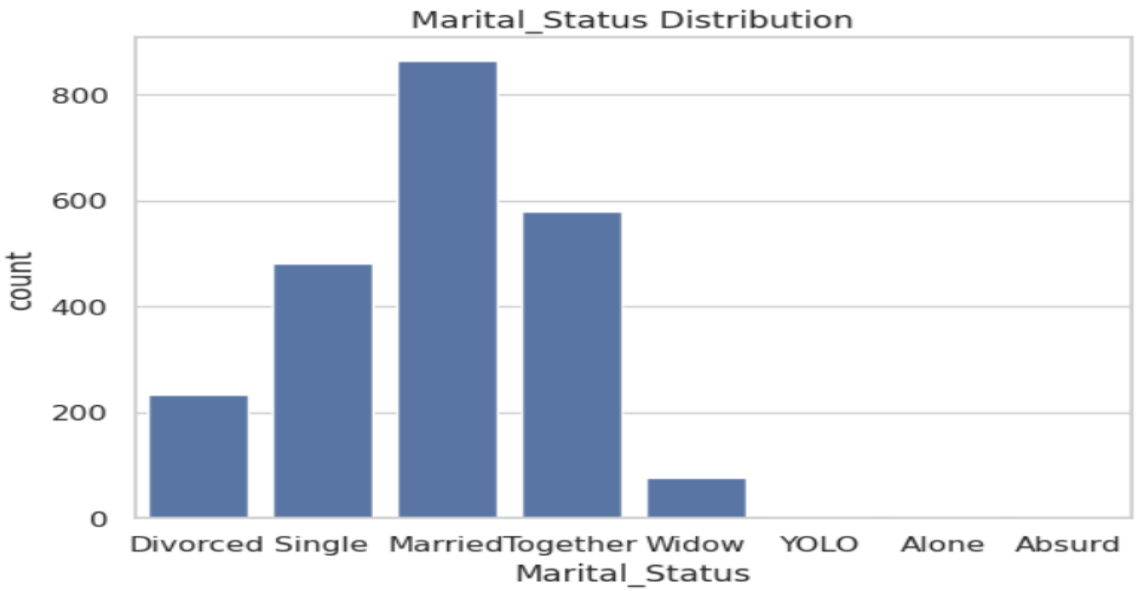


4.3 Count Plot for a Categorical Column

```
if 'Marital_Status' in df.columns:
    sns.countplot(x='Marital_Status', data=df)
    plt.title("Marital_Status Distribution")
    plt.show()plt.show()
```

This code creates a count plot for the **Marital_Status** column, visually representing the distribution of different marital categories in the dataset.

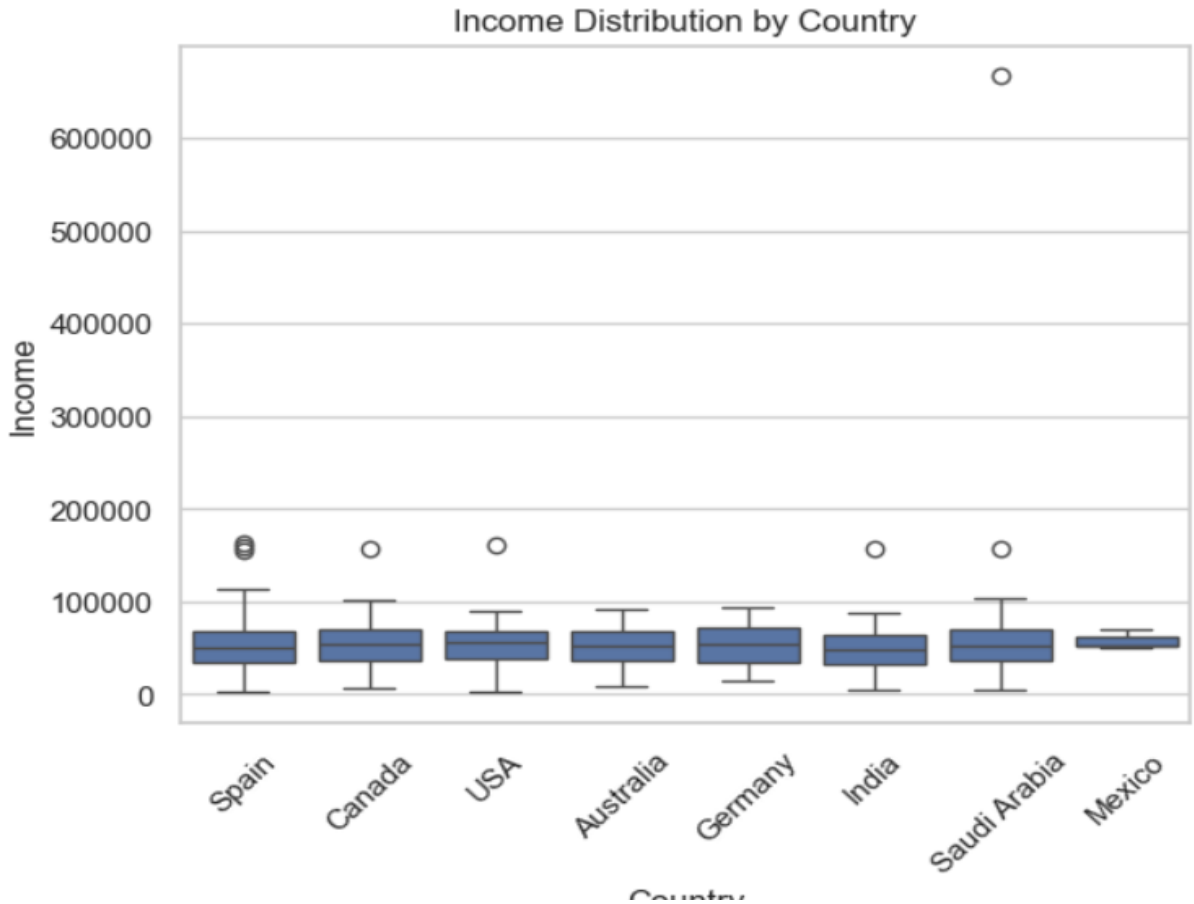
Result:



4.4 Boxplot to Detect Outliers

```
df.columns = df.columns.str.strip()
if 'Income' in df.columns and 'Country' in df.columns:
    print("Columns found. Plotting...")
    sns.boxplot(x='Country', y='Income', data=df)
    plt.title("Income Distribution by Country")
    plt.xticks(rotation=45)
    plt.show()
```

This code generates a boxplot of **Income** by **Country**, helping to visually identify outliers and compare income distribution across countries. (Note: The title may need correction if the intent is to show distribution by **Country**, not **Gender**.)

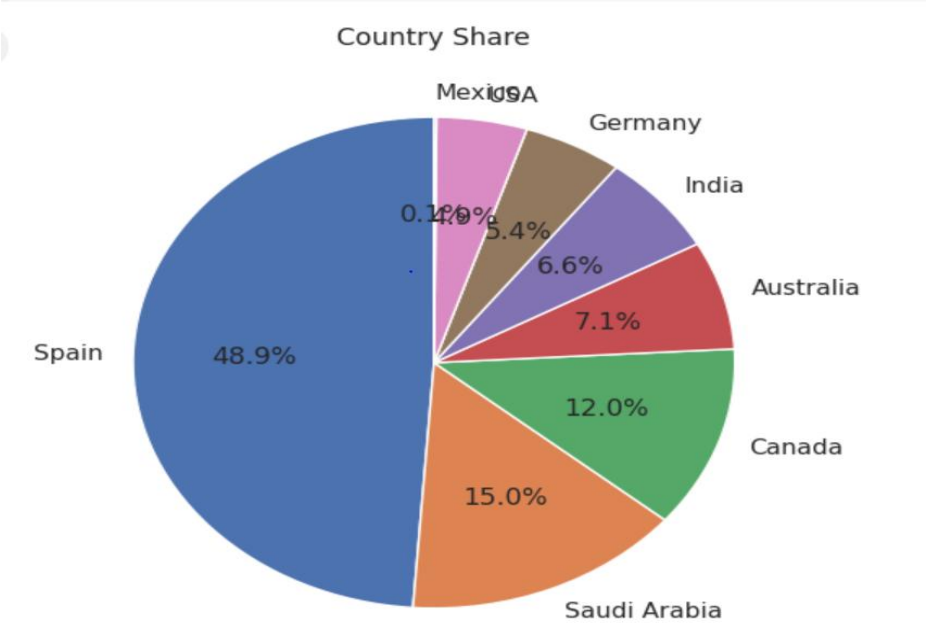


4.5 Pie Chart for Category Share

```
if 'Country' in df.columns:
    df['Country'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, figsize=(6, 6))
    plt.title("Country Share")
    plt.ylabel('')
    plt.show()
```

This code creates a pie chart showing the proportion of records from each country, providing a quick visual overview of the dataset's geographic distribution.

Result:

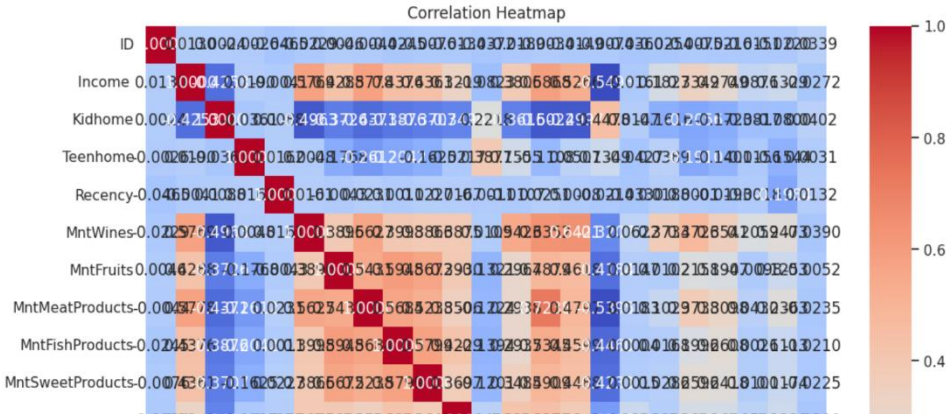


4.6 Correlation Heatmap

```
plt.figure(figsize=(12, 12))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt=".4f")
plt.title("Correlation Heatmap")
plt.show()
```

This code generates a heatmap of the correlation matrix for numerical variables, visually highlighting strong positive or negative relationships between features.

Result:

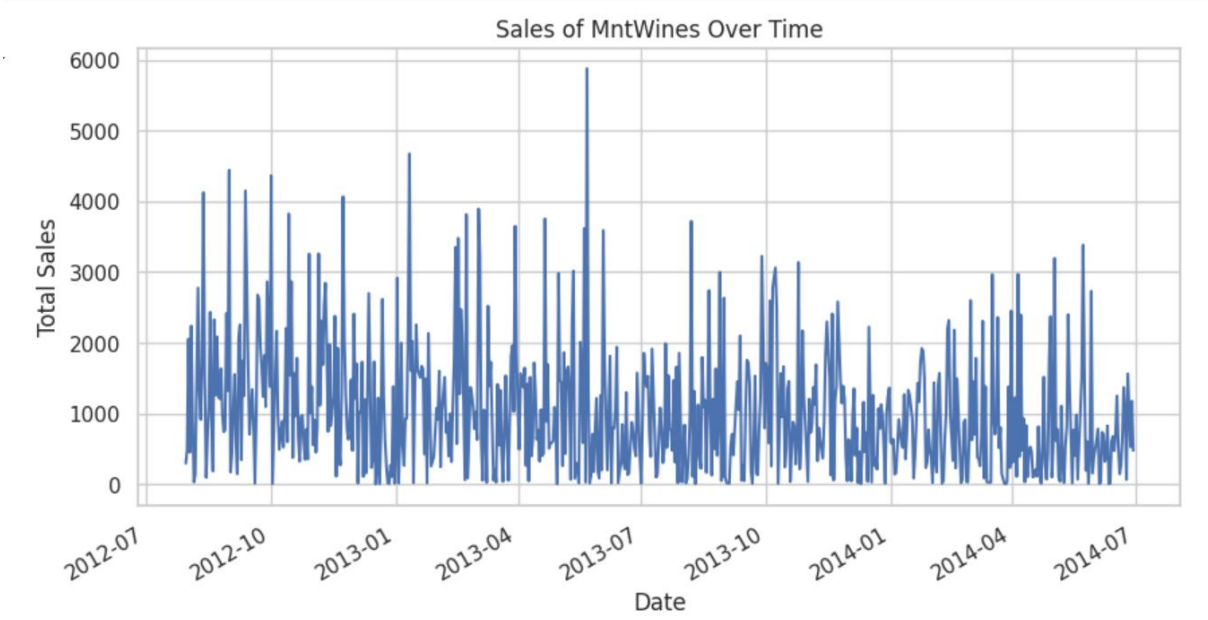


4.7 Line Plot for Time Series

```
if 'Dt_Customer' in df.columns and 'MntWines' in df.columns:
    df['Date'] = pd.to_datetime(df['Dt_Customer'])
    df.groupby('Date')['MntWines'].sum().plot(figsize=(10, 5))
    plt.title("Sales of MntWines Over Time")
    plt.ylabel("Total Sales")
    plt.xlabel("Date")
    plt.grid(True)
    plt.show()
```

This code creates a time series line plot showing how **MntWines** sales have varied over time, helping to identify trends or seasonal patterns in customer purchases.

Result:

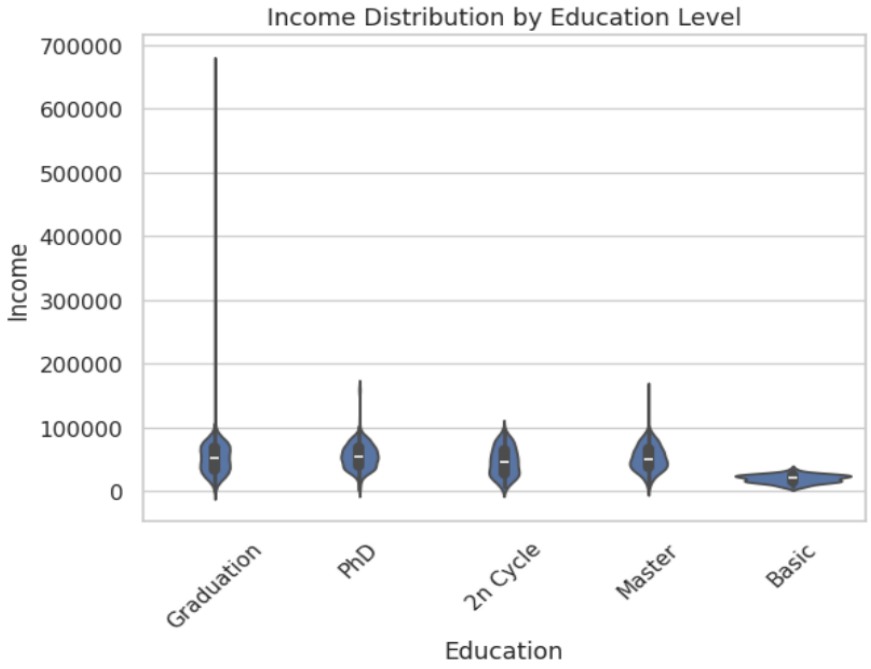


4.8 Violin Plot

```
sns.violinplot(x='Education', y='Income', data=df)

plt.title("Income Distribution by Education Level")

plt.xlabel("Education")

plt.ylabel("Income")

plt.xticks(rotation=45)

plt.show()

plt.show()
```

This code generates a violin plot to visualize the distribution and density of **Income** across different **Education** levels, revealing variations and potential outliers within each group.

Result:

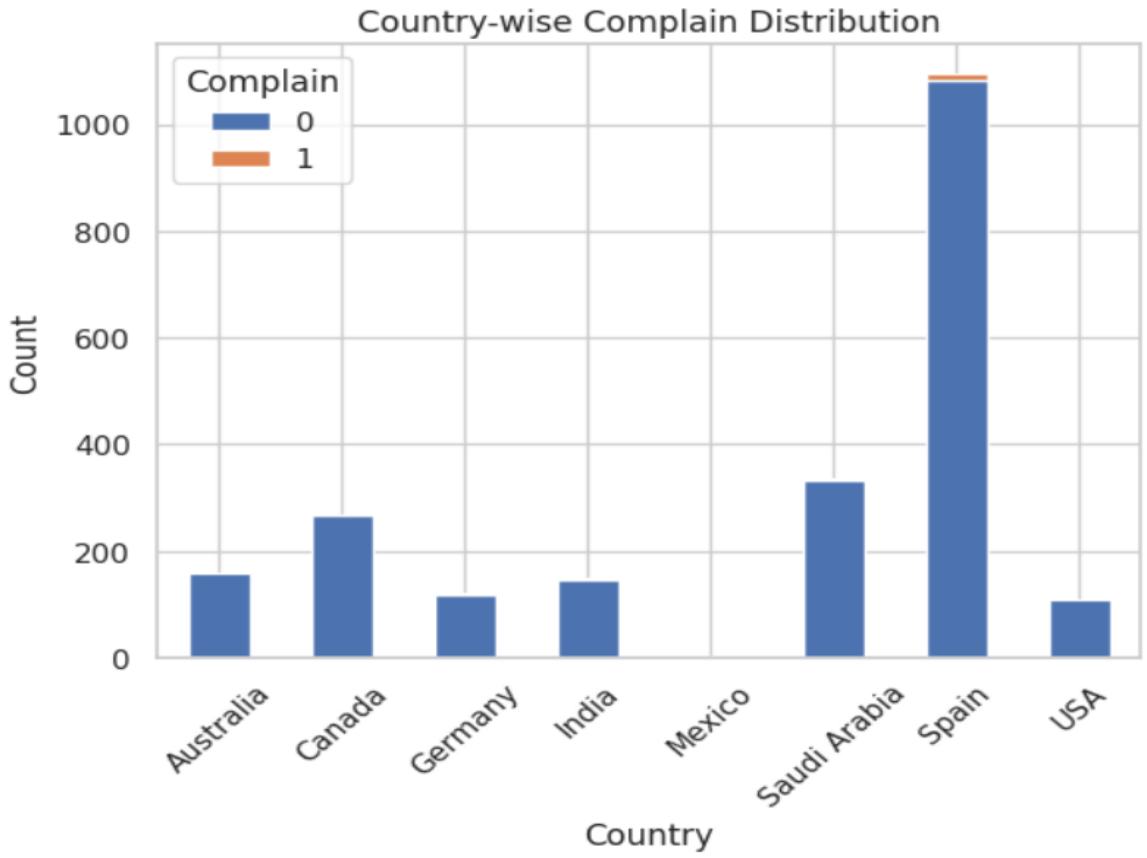


4.9 Stacked Bar Chart

```
if 'Country' in df.columns and 'Complain' in df.columns:
    stacked = pd.crosstab(df['Country'], df['Complain'])
    stacked.plot(kind='bar', stacked=True)
    plt.title("Country-wise Complain Distribution")
    plt.ylabel("Count"
    plt.xticks(rotation=45)
    plt.show()
```

This code generates a stacked bar chart showing the distribution of complaint status across countries, helping to compare how customer complaints vary by region.

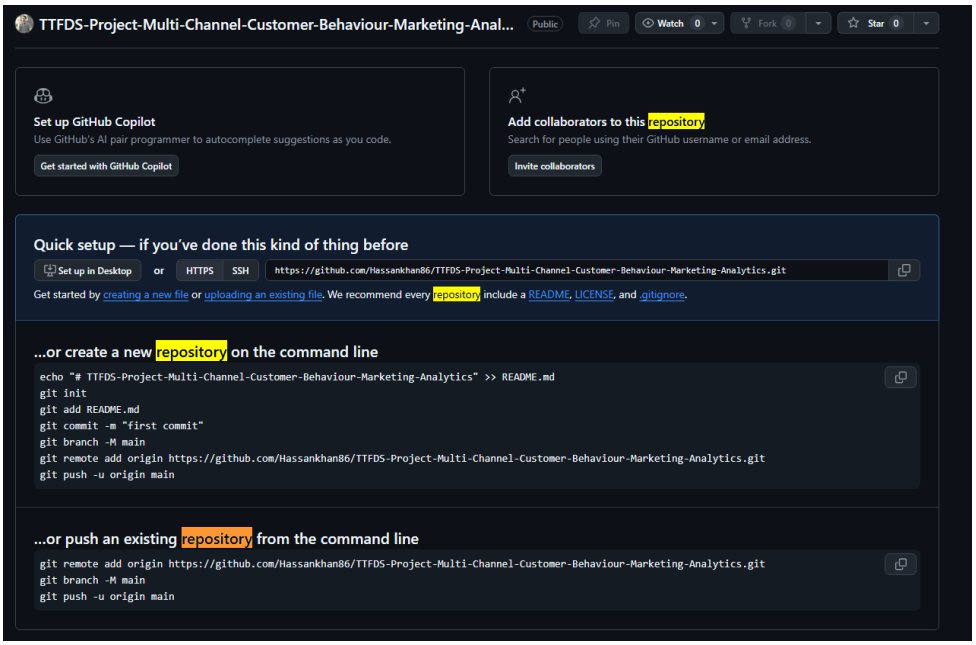
Result:


Country-wise Complain Distribution
Complain
0
1
Count
0
200
400
600
800
1000
Australia
Canada
Germany
India
Mexico
Saudi Arabia
Spain
USA
Country

Part E: Use of Git

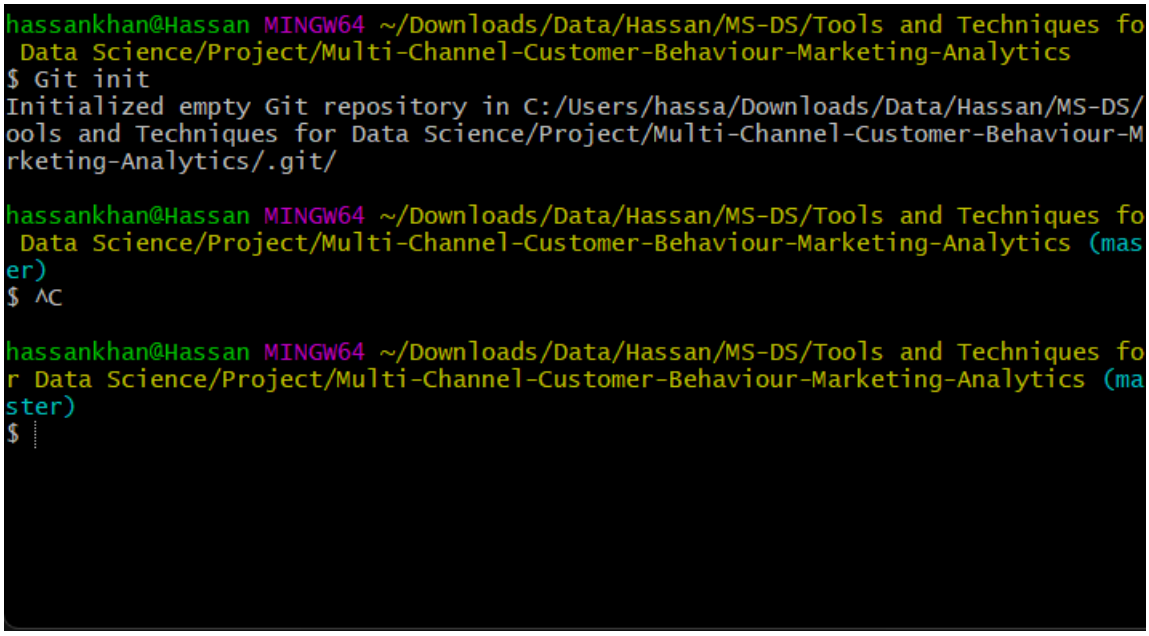
5. Git Setup

5.1 Repository created



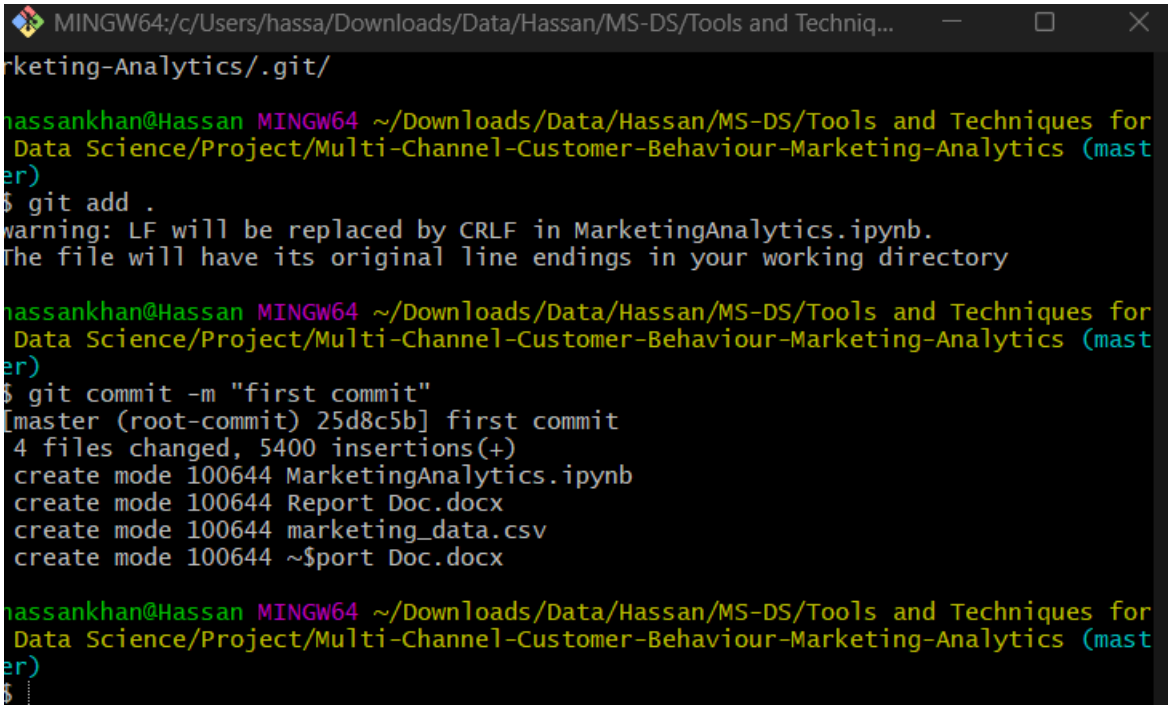
5.2 Initializes a new local Git repository in your current folder

Initializes a new local Git repository in your current folder.



5.3 Git add/commit

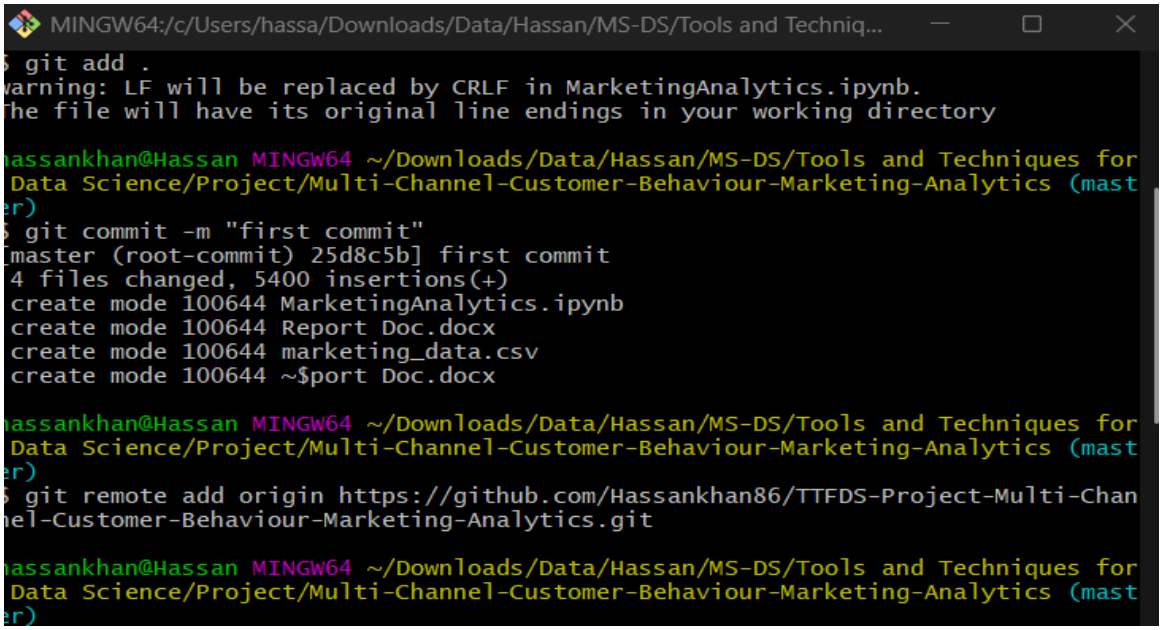
- Git Add: Moves files into the **staging area** (preparing for commit
- Git Commit: Saves your staged changes into Git history with a descriptive message.



5.4 Git remote add origin

Cmd: Git remote add origin https://github.com/Hassankhan86/TTFDS-Project-Multi-Channel-Customer-Behaviour-Marketing-Analytics.git

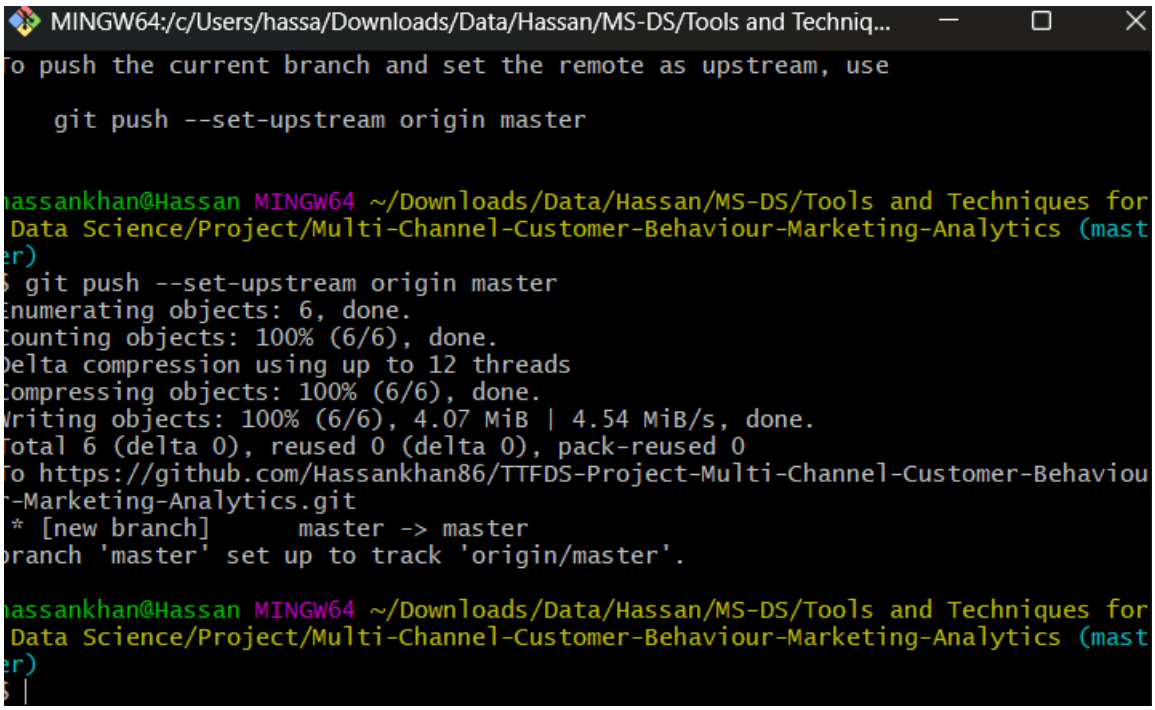
Connect your local project to a remote repository (only needed once).



5.5 Git push

Uploads commits from your local branch to the remote branch.

- First-time push for a branch: git push --set-upstream origin master



5.6 Link to the Git Repo

Link: https://github.com/Hassankhan86/TTFDS-Project-Multi-Channel-Customer-Behaviour-Marketing-Analytics

Part F: Use of ML techniques

6. Use of Machine Learning Techniques

Develop two models using suitable data mining algorithms

6.1 Import Libraries

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
df = pd.read_csv("marketing_data.csv")
df.head()
```

In this step, essential Python libraries were imported, and the marketing dataset was loaded using `pandas`. This provided the foundation for further data preprocessing, exploration, and modeling.

6.3 Data Preprocessing

```
df.columns = df.columns.str.strip()
df['Age'] = 2024 - df['Year_Birth']
features = [
    'Income', 'Age', 'Kidhome', 'Teenhome',
    'MntWines', 'MntMeatProducts', 'MntGoldProds',
    'NumWebPurchases', 'NumStorePurchases'

df_model = df[features + ['Response']].dropna()  # Drop rows with missing values
df_model.head()
```

In this step, the dataset was cleaned and prepared for modeling. Missing values were handled by dropping incomplete rows, and new features such as **Age** were created from the birth year. Only relevant variables were selected to keep the analysis simple and focused. The data was then split into training and testing sets for model building.

6.4 Split Data into Train/Test Sets

```
X = df_model[features]

y = df_model['Response']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

To evaluate model performance, the dataset was divided into training and testing sets using an 80-20 split. The training set was used to build the models, while the testing set helped assess how well the models generalized to unseen data. Feature scaling was also applied to standardize the input variables where necessary.

6.5. Train Logistic Regression Model

```
lr = LogisticRegression()

lr.fit(X_train_scaled, y_train)


y_pred_lr = lr.predict(X_test_scaled)


print("Logistic Regression Report:")

print(classification_report(y_test, y_pred_lr))
```

A Logistic Regression model was trained using the standardized training data to predict customer responses. This algorithm was chosen for its simplicity and interpretability in binary classification tasks. The model performed well in predicting non-responders but struggled to identify actual responders, as shown by its low recall and F1-score for class 1. Overall, it served as a useful baseline for comparison with more complex models.

Result:

```
Logistic Regression Report:
              precision    recall  f1-score   support

           0       0.87      0.98      0.92       379
           1       0.50      0.12      0.20        65

    accuracy                           0.85       444
   macro avg       0.68      0.55      0.56       444
weighted avg       0.81      0.85      0.81       444
```

6.6. Train Random Forest Classifier

```
rf = RandomForestClassifier(random_state=42)

rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)

print("Random Forest Classifier Report:")

print(classification_report(y_test, y_pred_rf))
```

A Random Forest Classifier was trained on the same dataset to improve prediction performance. This ensemble method combines multiple decision trees and is well-suited for handling complex, non-linear relationships in the data. Compared to Logistic Regression, it showed better results in identifying actual responders, with higher recall and F1-score for class 1. The model also maintained strong overall accuracy, making it a more effective choice for this classification task.

Result:

```
Random Forest Classifier Report:
              precision    recall  f1-score   support

           0       0.89      0.97      0.93       379
           1       0.59      0.29      0.39        65

    accuracy                           0.87       444
   macro avg       0.74      0.63      0.66       444
weighted avg       0.85      0.87      0.85       444
```

6.7 Analyse the results and provide a comparative evaluation using different data mining and visualization methods.

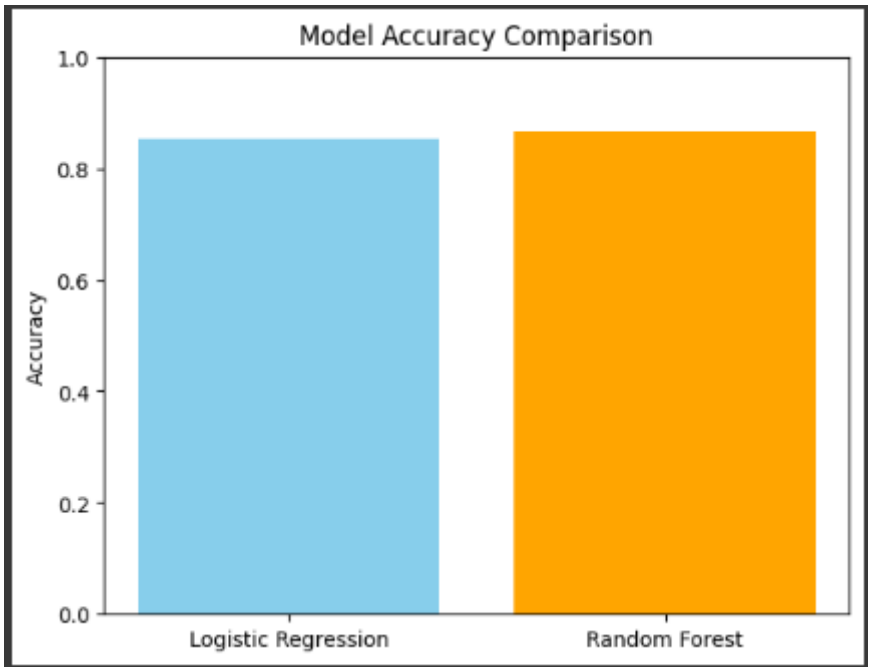
6.7.1 Accuracy comparison

```
accuracy_lr = accuracy_score(y_test, y_pred_lr)

accuracy_rf = accuracy_score(y_test, y_pred_rf)


plt.bar(['Logistic Regression', 'Random Forest'], [accuracy_lr, accuracy_rf],
color=['skyblue', 'orange'])

plt.ylabel('Accuracy')

plt.title('Model Accuracy Comparison')

plt.ylim(0, 1)

plt.show()
```

1. **Overall Accuracy**

- Logistic Regression Accuracy: 85%
- Random Forest Accuracy: 87%

Random Forest performs slightly better overall.



2. Class-wise Performance

Class 0: Customers who did not respond

- Logistic Regression F1-score: 0.92
- Random Forest F1-score: 0.93

Both models are very good at predicting non-responders.

Class 1: Customers who did respond (key target)

- Logistic Regression F1-score: 0.20
- Random Forest F1-score: 0.39

Random Forest almost doubles the F1-score for responders — still not perfect, but much better.

3. Precision vs. Recall for Class 1

- Logistic Regression Recall: 0.12 → Caught only 12% of actual responders
- Random Forest Recall: 0.29 → Caught 29% of actual responders

Random Forest is clearly better at finding responders, which is the goal of targeted marketing.

4. Insights

- Logistic Regression is simple but struggles with the minority class (Response = 1).
- Random Forest performs better on both classes, especially in identifying responders.

This is expected, as ensemble methods like Random Forest handle class imbalance and non-linear patterns better.

Confusion Matrices

6.7.2 Logistic Regression Confusion Matrix:

```
fig, axs = plt.subplots(1, 2, figsize=(12, 5))

sns.heatmap(confusion_matrix(y_test, y_pred_lr), annot=True, fmt='d', cmap='Blues', ax=axs[0])

axs[0].set_title('Logistic Regression')

axs[0].set_xlabel('Predicted')

axs[0].set_ylabel('Actual')
```

- Correctly predicted non-responders (0): 371
- Correctly predicted responders (1): 8
- Missed responders: 57

This model struggles to catch responders — only 8 correctly identified out of 65.

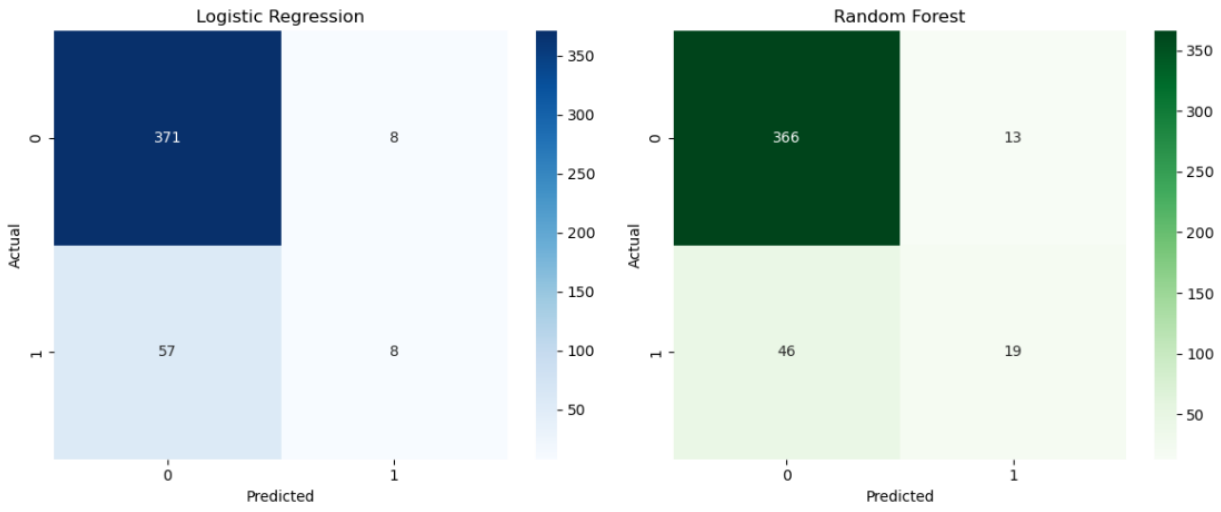
6.7.3 Random Forest Confusion Matrix:

```
sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d', cmap='Greens', ax=axs[1])

axs[1].set_title('Random Forest')

axs[1].set_xlabel('Predicted')

axs[1].set_ylabel('Actual')

plt.tight_layout()

plt.show()
```

- Correctly predicted non-responders (0): 366
- Correctly predicted responders (1): 19
- Missed responders: 46

Improved ability to detect responders — 19 out of 65 correctly identified.

Conclusion:



The heatmaps clearly demonstrate that while both models predict non-responders well, **Random Forest identifies more actual responders** (19 vs. 8). This is crucial for marketing campaigns where identifying potential customers is more valuable than avoiding false positives.

Part G: Overall quality of the project

7. Quality of the project

This project demonstrates a complete and well-structured data analytics workflow, starting from data loading, cleaning, and exploratory analysis to advanced visualizations and predictive modelling. The dataset was carefully prepared through rigorous data wrangling steps, including handling missing values, removing duplicates, feature engineering, and transforming variables into a usable format. Comprehensive EDA was conducted to understand customer demographics, spending behaviour, and multi-channel interactions, and these insights were effectively visualized using a variety of charts with clear observations. The Power BI dashboard adds an interactive business intelligence layer, allowing stakeholders to explore customer segments, channel performance, and marketing campaign effectiveness in real time. Finally, machine learning models were implemented to predict customer response, enabling data-driven marketing decisions. Overall, the project delivers high analytical depth, meaningful business insights, and practical value aligned with the theme of “Multi-Channel Customer Behaviour & Marketing Analytics.”