Marketing Campaign Analysis

Load the Data set

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

df = pd.read_excel('/content/marketing_campaign.xlsx')

df.head(5)
```

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Custome
0	5524	1957	Graduation	Single	58138.0	0	0	2012-09-0
1	2174	1954	Graduation	Single	46344.0	1	1	2014-03-0
2	4141	1965	Graduation	Together	71613.0	0	0	2013-08-2
3	6182	1984	Graduation	Together	26646.0	1	0	2014-02-1
4	5324	1981	PhD	Married	58293.0	1	0	2014-01-1

5 rows × 29 columns

Data Cleaning

Check for missing values

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

Fill missing values in 'Income' with the median value

```
df['Income'].fillna(df['Income'].median(), inplace=True)
```

```
<ipython-input-12-b0f8a841459e>:1: FutureWarning: A value is trying to be set on a co
The behavior will change in pandas 3.0. This inplace method will never work because t

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({
```

```
df['Income'].fillna(df['Income'].median(), inplace=True)
```

Convert 'Dt_Customer' to datetime format

```
df['Dt_Customer'] = pd.to_datetime(df['Dt_Customer'], format='%Y-%m-%d')
```

Drop any rows with missing values in critical columns

```
df.dropna(subset=['Income', 'Education', 'Marital_Status'], inplace=True)
```

Verify the data after cleaning

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 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               2240 non-null   float64
 5   Kidhome              2240 non-null   int64
 6   Teenhome             2240 non-null   int64
 7   Dt_Customer          2240 non-null   datetime64[ns]
 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
 20  AcceptedCmp3         2240 non-null   int64
 21  AcceptedCmp4         2240 non-null   int64
 22  AcceptedCmp5         2240 non-null   int64
 23  AcceptedCmp1         2240 non-null   int64
 24  AcceptedCmp2         2240 non-null   int64
 25  Complain             2240 non-null   int64
 26  Z_CostContact        2240 non-null   int64
```

```
 27  Z_Revenue            2240 non-null   int64
 28  Response             2240 non-null   int64
dtypes: datetime64[ns](1), float64(1), int64(25), object(2)
memory usage: 507.6+ KB
```

Ensuring Data Consistency

Check unique values in 'Marital_Status' to spot inconsistencies

```
print(df['Marital_Status'].unique())
df['Marital_Status'] = df['Marital_Status'].replace(['Alone'], 'Single')
df['Marital_Status'] = df['Marital_Status'].str.lower()
```

```
['Single' 'Together' 'Married' 'Divorced' 'Widow' 'Alone' 'Absurd' 'YOLO']
```

Standardizing Data Formats

Convert 'Dt_Customer' to datetime format

```
df['Dt_Customer'] = pd.to_datetime(df['Dt_Customer'], format='%Y-%m-%d')
```

Standardize numerical columns

```
df['Income'] = df['Income'].astype(float)
```

Convert categorical columns to lowercase and ensure consistency

```
df['Education'] = df['Education'].str.lower()
df['Marital_Status'] = df['Marital_Status'].str.lower()
```

Verify the standardized formats

```
print(df.dtypes)
```

```
ID                              int64
Year_Birth                      int64
Education                      object
Marital_Status                 object
Income                        float64
Kidhome                         int64
Teenhome                        int64
Dt_Customer            datetime64[ns]
Recency                         int64
MntWines                        int64
MntFruits                       int64
MntMeatProducts                 int64
MntFishProducts                 int64
MntSweetProducts                int64
MntGoldProds                    int64
NumDealsPurchases               int64
NumWebPurchases                 int64
NumCatalogPurchases             int64
NumStorePurchases               int64
NumWebVisitsMonth               int64
AcceptedCmp3                    int64
AcceptedCmp4                    int64
AcceptedCmp5                    int64
AcceptedCmp1                    int64
AcceptedCmp2                    int64
Complain                        int64
Z_CostContact                   int64
Z_Revenue                       int64
Response                        int64
dtype: object
```

Data Analysis

Calculate Total Revenue from marketing campaigns

```
df['Total_Revenue'] = df['MntWines'] + df['MntFruits'] + df['MntMeatProducts'] + df['MntF
```

Calculate ROI for campaigns

```
df['ROI'] = (df['Z_Revenue'] - df['Z_CostContact']) / df['Z_CostContact']
```

Calculate Cost Per Acquisition (CPA)

```
df['CPA'] = df['Z_CostContact'] / (df['AcceptedCmp1'] + df['AcceptedCmp2'] + df['Accepted
```

Analyze key metrics

```
campaign_performance = df[['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'AcceptedCmp4'
```

```
/usr/local/lib/python3.10/dist-packages/numpy/lib/function_base.py:4655: RuntimeWarni
  diff_b_a = subtract(b, a)
```

Display the performance

```
campaign_performance
```

	AcceptedCmp1	AcceptedCmp2	AcceptedCmp3	AcceptedCmp4	AcceptedCmp5	
count	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2.240000
mean	0.064286	0.013393	0.072768	0.074554	0.072768	2.666667
std	0.245316	0.114976	0.259813	0.262728	0.259813	5.374679
min	0.000000	0.000000	0.000000	0.000000	0.000000	2.666667
25%	0.000000	0.000000	0.000000	0.000000	0.000000	2.666667
50%	0.000000	0.000000	0.000000	0.000000	0.000000	2.666667
75%	0.000000	0.000000	0.000000	0.000000	0.000000	2.666667
max	1.000000	1.000000	1.000000	1.000000	1.000000	2.666667

Analyzing Campaign Performance Based on Key Metrics

Analyze the number of accepted campaigns

```
campaign_acceptances = df[['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'AcceptedCmp4'

print("Number of customers who accepted each campaign:\n", campaign_acceptances)
```

```
Number of customers who accepted each campaign:
 AcceptedCmp1    144
```

```
AcceptedCmp2     30
AcceptedCmp3    163
AcceptedCmp4    167
AcceptedCmp5    163
dtype: int64
```

Analyze the overall response rate

```
response_rate = df['Response'].value_counts(normalize=True) * 100
print("Response rate (%):\n", response_rate)
```

```
Response rate (%):
 Response
0    85.089286
1    14.910714
Name: proportion, dtype: float64
```

Check the recency of campaigns

```
print("Recency stats (days since last interaction):")
print(df['Recency'].describe())
```

```
Recency stats (days since last interaction):
count    2240.000000
mean       49.109375
std        28.962453
min         0.000000
25%        24.000000
50%        49.000000
75%        74.000000
max        99.000000
Name: Recency, dtype: float64
```

Calculating ROI and Cost Per Acquisition (CPA)

Calculate total revenue from campaigns

```
df['Total_Revenue'] = df['MntWines'] + df['MntFruits'] + df['MntMeatProducts'] + df['MntF
```

Calculate ROI for each campaign

```
df['ROI'] = (df['Total_Revenue'] - df['Z_CostContact']) / df['Z_CostContact']
```

Calculate Cost Per Acquisition (CPA)

```
df['Total_Accepted'] = df['AcceptedCmp1'] + df['AcceptedCmp2'] + df['AcceptedCmp3'] + df[
```

CPA = Total Cost / Number of Acceptances

```
df['CPA'] = df['Z_CostContact'] / df['Total_Accepted']
```

Display the average ROI and CPA

```
print("Average ROI: ", df['ROI'].mean())
print("Average CPA: ", df['CPA'].mean())
```

```
Average ROI:  200.93273809523808
Average CPA:  inf
```

Identifying Factors Contributing to Successful Campaigns

Analyze correlations between key variables and the 'Response'

```
numerical_features = df.select_dtypes(include=np.number).columns
correlation_matrix = df[numerical_features].corr()

print("Correlation between different features:\n", correlation_matrix['Response'].sort_va
```

```
Correlation between different features:
 Response              1.000000
Total_Accepted         0.426035
AcceptedCmp5           0.326634
AcceptedCmp1           0.293982
ROI                    0.265298
Total_Revenue          0.265298
AcceptedCmp3           0.254258
MntWines               0.247254
```

```
MntMeatProducts        0.236335
NumCatalogPurchases    0.220810
AcceptedCmp4           0.177019
AcceptedCmp2           0.169293
NumWebPurchases        0.148730
MntGoldProds           0.139850
Income                 0.132867
MntFruits              0.125289
MntSweetProducts       0.117372
MntFishProducts        0.111331
NumStorePurchases      0.039363
Year_Birth             0.021325
NumDealsPurchases      0.002238
Complain              -0.001707
NumWebVisitsMonth     -0.003987
ID                    -0.021968
Kidhome               -0.080008
Teenhome              -0.154446
Recency               -0.198437
CPA                   -0.324961
Z_CostContact               NaN
Z_Revenue                   NaN
Name: Response, dtype: float64
```

Group by marital status and calculate the average response rate

```
marital_response = df.groupby('Marital_Status')['Response'].mean()
print("Response rate by marital status:\n", marital_response)
```

```
Response rate by marital status:
 Marital_Status
absurd      0.500000
divorced    0.206897
married     0.113426
single      0.221532
together    0.103448
widow       0.246753
yolo        0.500000
Name: Response, dtype: float64
```

Group by education level and calculate the average response rate

```
education_response = df.groupby('Education')['Response'].mean()
print("Response rate by education level:\n", education_response)
```

```
Response rate by education level:
 Education
```

```
2n Cycle      0.108374
Basic         0.037037
Graduation    0.134871
Master        0.154054
PhD           0.207819
Name: Response, dtype: float64
```

Analyze impact of income on response rates

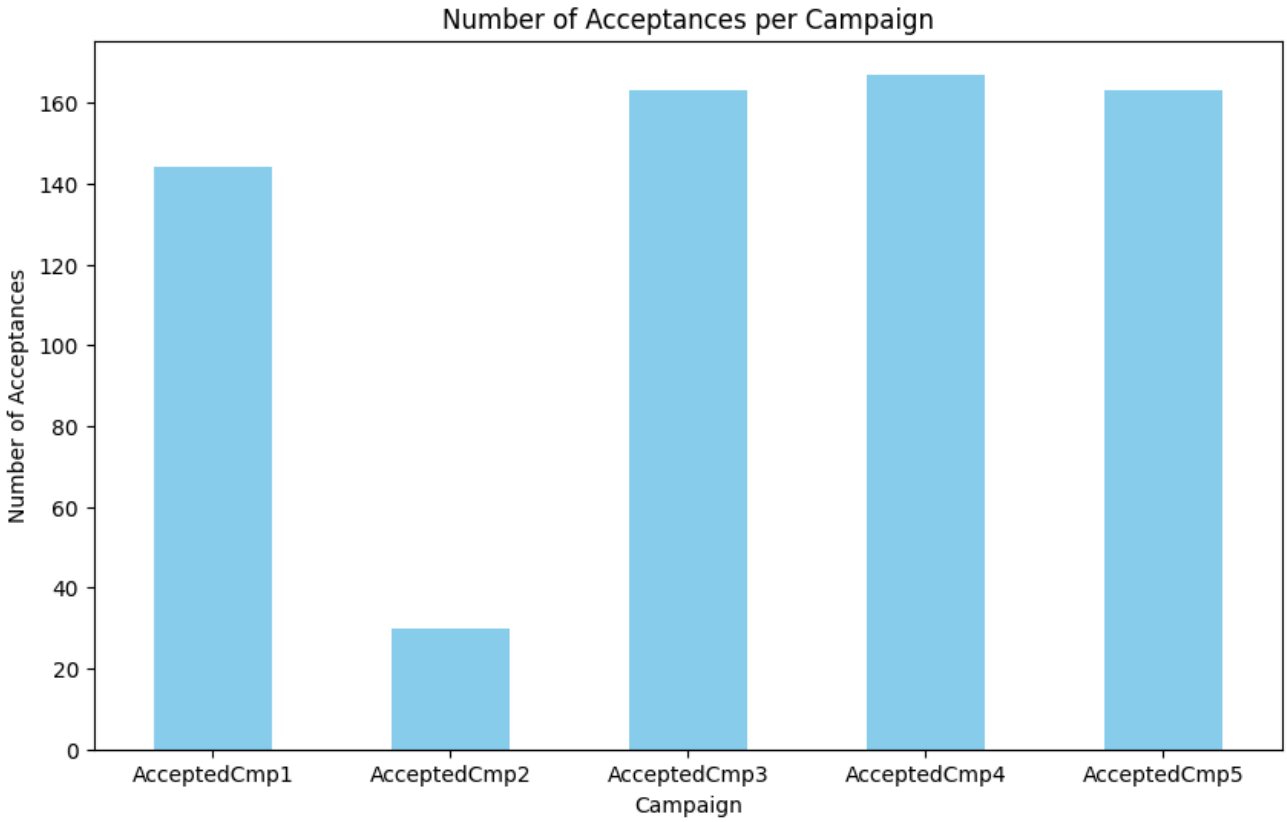
```
income_response = df.groupby(pd.cut(df['Income'], bins=5))['Response'].mean()
print("Response rate by income level:\n", income_response)
```

```
Response rate by income level:
 Income
(1065.064, 134717.2]    0.150815
(134717.2, 267704.4]    0.000000
(267704.4, 400691.6]         NaN
(400691.6, 533678.8]         NaN
(533678.8, 666666.0]    0.000000
Name: Response, dtype: float64
<ipython-input-39-099a47d32894>:1: FutureWarning: The default of observed=False is de
  income_response = df.groupby(pd.cut(df['Income'], bins=5))['Response'].mean()
```

Visualization

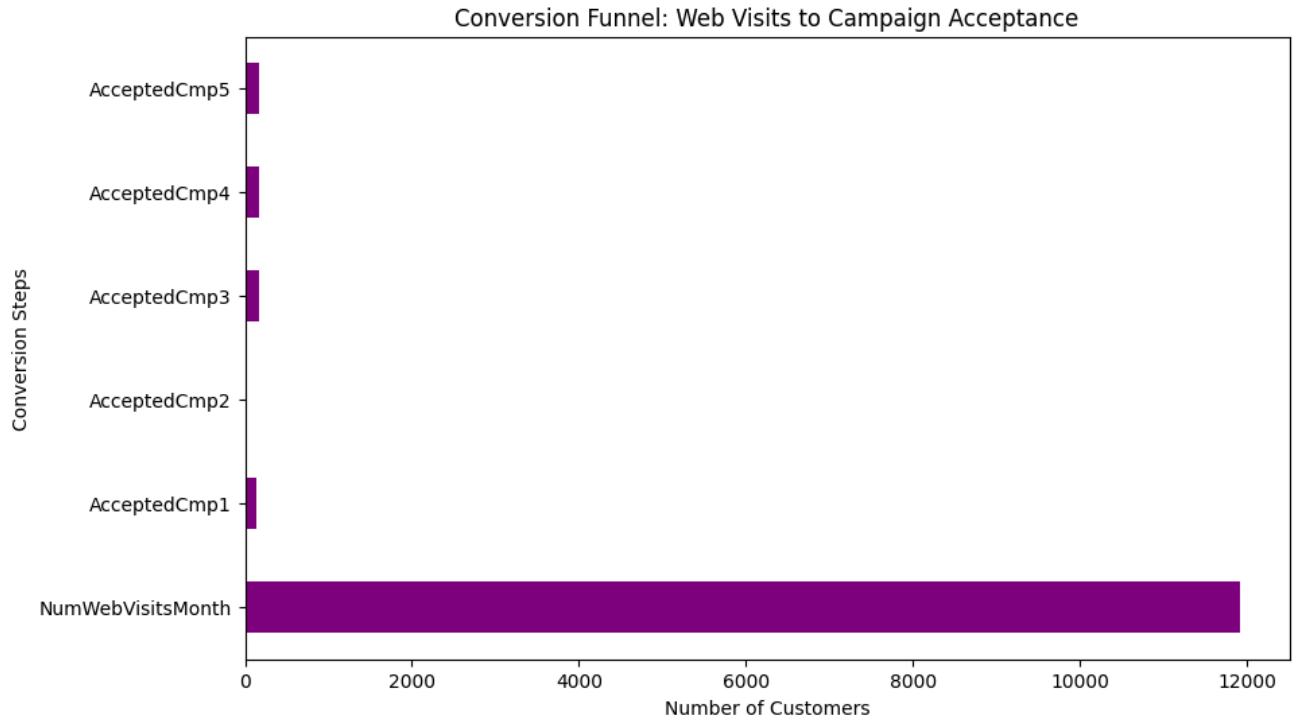
Bar Charts for Campaign Acceptance

```
accepted_cmp = ['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', 'AcceptedCmp4', 'Accepted
campaign_acceptances = df[accepted_cmp].sum()

plt.figure(figsize=(10, 6))
campaign_acceptances.plot(kind='bar', color='skyblue')
plt.title('Number of Acceptances per Campaign')
plt.xlabel('Campaign')
plt.ylabel('Number of Acceptances')
plt.xticks(rotation=0)
plt.show()
```



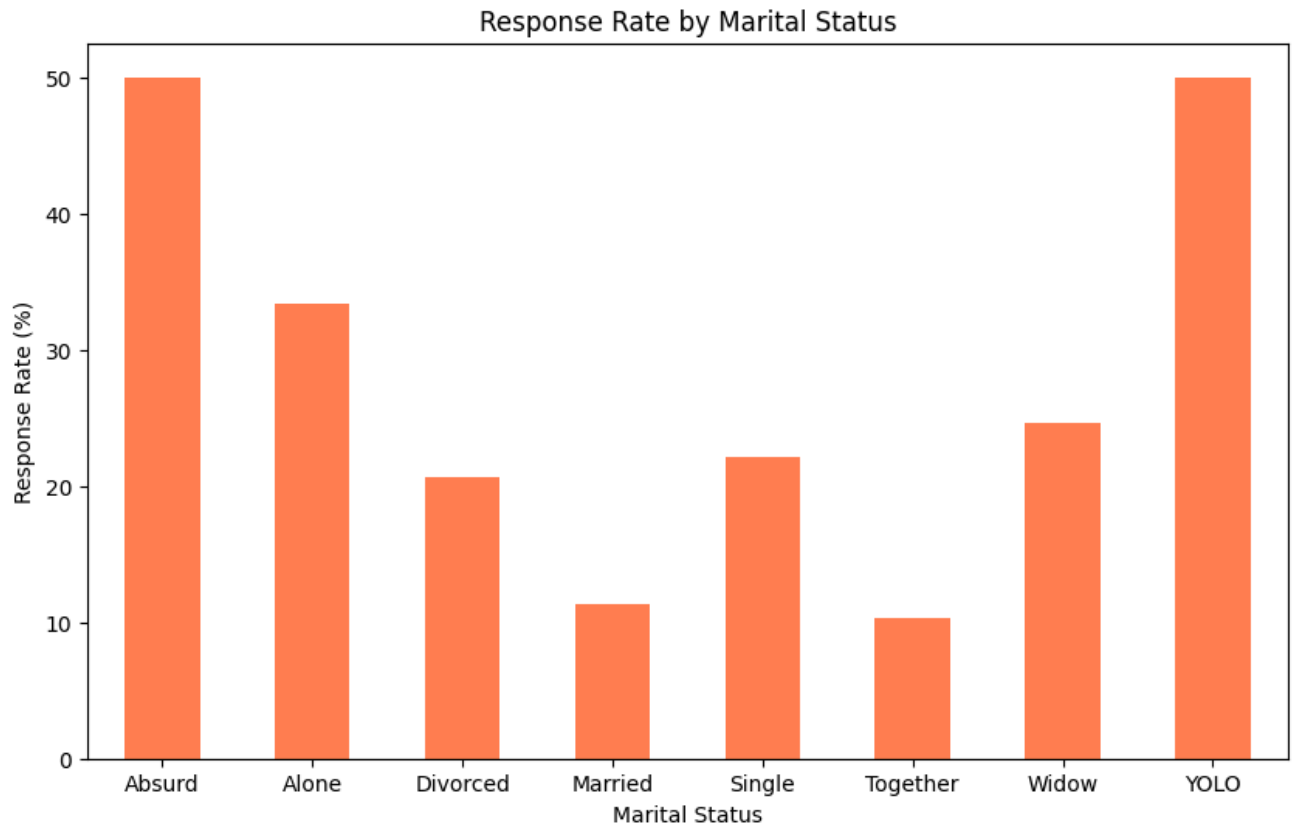
Conversion Funnel

```
conversion_steps = df[['NumWebVisitsMonth', 'AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3

plt.figure(figsize=(10, 6))
conversion_steps.plot(kind='barh', color='purple')
plt.title('Conversion Funnel: Web Visits to Campaign Acceptance')
plt.xlabel('Number of Customers')
plt.ylabel('Conversion Steps')
plt.show()
```



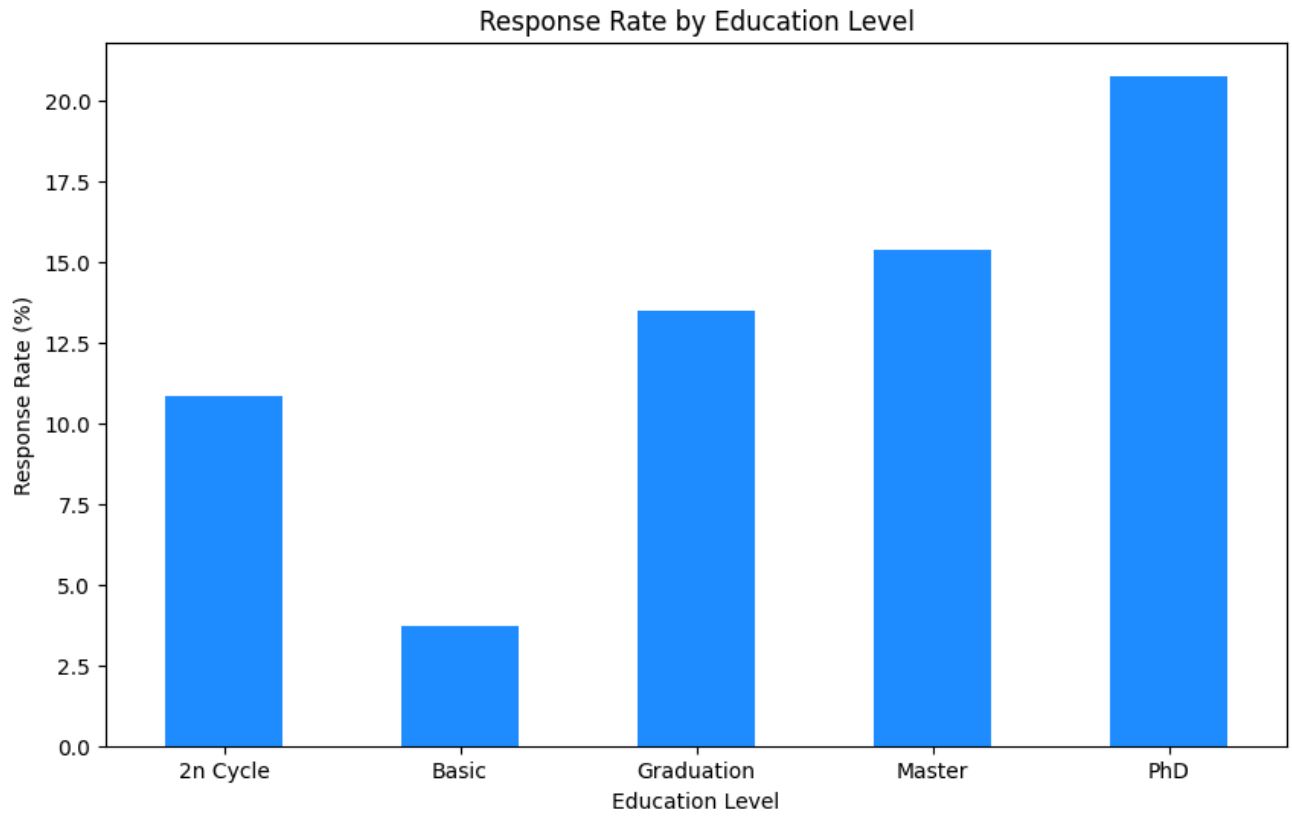
Bar Chart for Response Rate by Marital Status

```
marital_response = df.groupby('Marital_Status')['Response'].mean() * 100

plt.figure(figsize=(10, 6))
marital_response.plot(kind='bar', color='coral')
plt.title('Response Rate by Marital Status')
plt.xlabel('Marital Status')
plt.ylabel('Response Rate (%)')
plt.xticks(rotation=0)
plt.show()
```



Bar Chart for Response Rate by Education Level

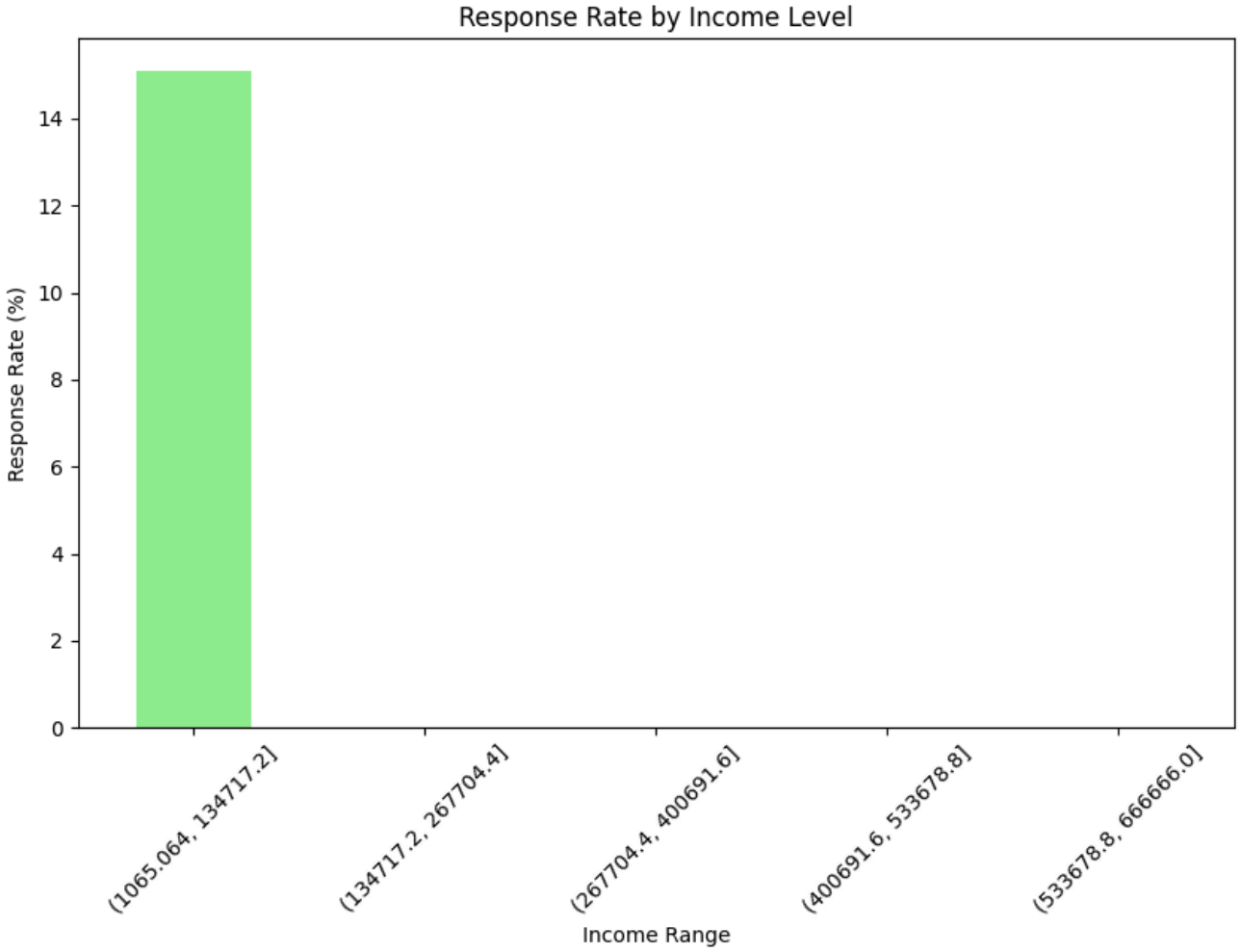
```
education_response = df.groupby('Education')['Response'].mean() * 100

plt.figure(figsize=(10, 6))
education_response.plot(kind='bar', color='dodgerblue')
plt.title('Response Rate by Education Level')
plt.xlabel('Education Level')
plt.ylabel('Response Rate (%)')
plt.xticks(rotation=0)
plt.show()
```



Bar Chart for Response Rate by Income Level

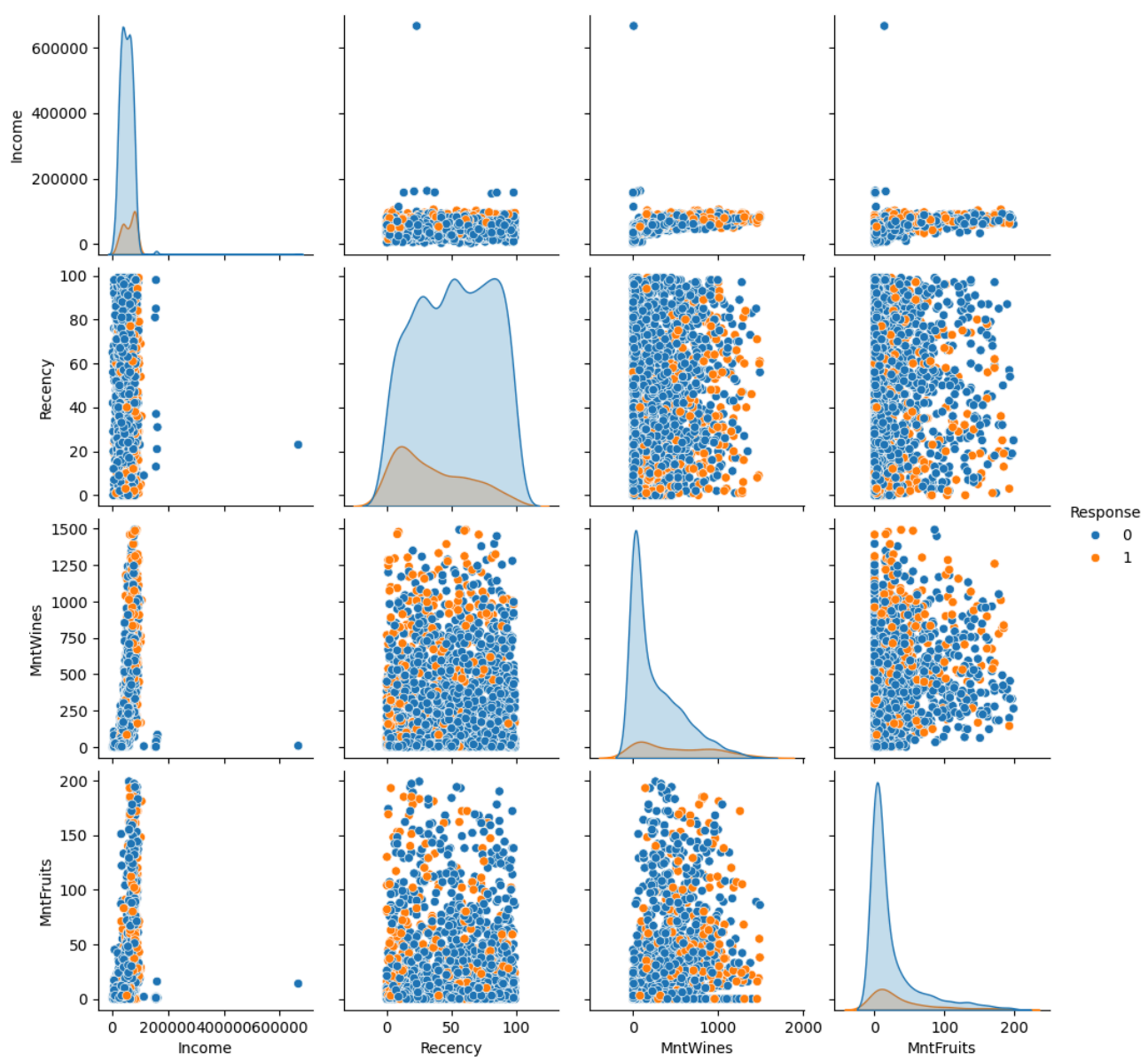
```
income_bins = pd.cut(df['Income'], bins=5)
income_response = df.groupby(income_bins)['Response'].mean() * 100

plt.figure(figsize=(10, 6))
income_response.plot(kind='bar', color='lightgreen')
plt.title('Response Rate by Income Level')
plt.xlabel('Income Range')
plt.ylabel('Response Rate (%)')
plt.xticks(rotation=45)
plt.show()
```

```
<ipython-input-48-b064766380e7>:2: FutureWarning: The default of observed=False is de
  income_response = df.groupby(income_bins)['Response'].mean() * 100
```



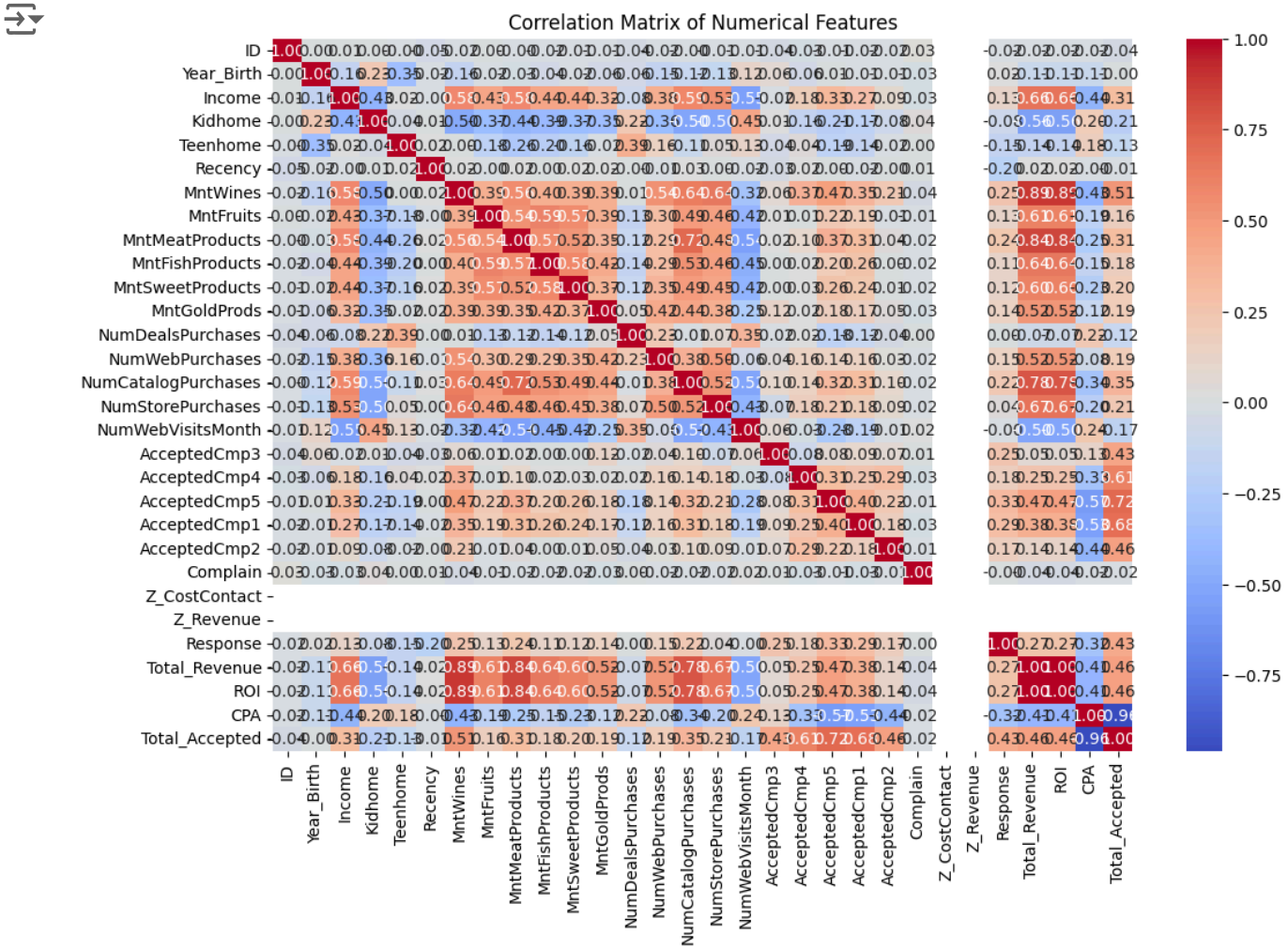
Pair Plot: Correlation between key variables

```
sns.pairplot(df[['Income', 'Recency', 'MntWines', 'MntFruits', 'Response']], hue='Respons
plt.show()
```


Income
Recency
MntWines
MntFruits
Response
0
1
0
200000
400000
600000
0
50
100
0
1000
2000
0
100
200

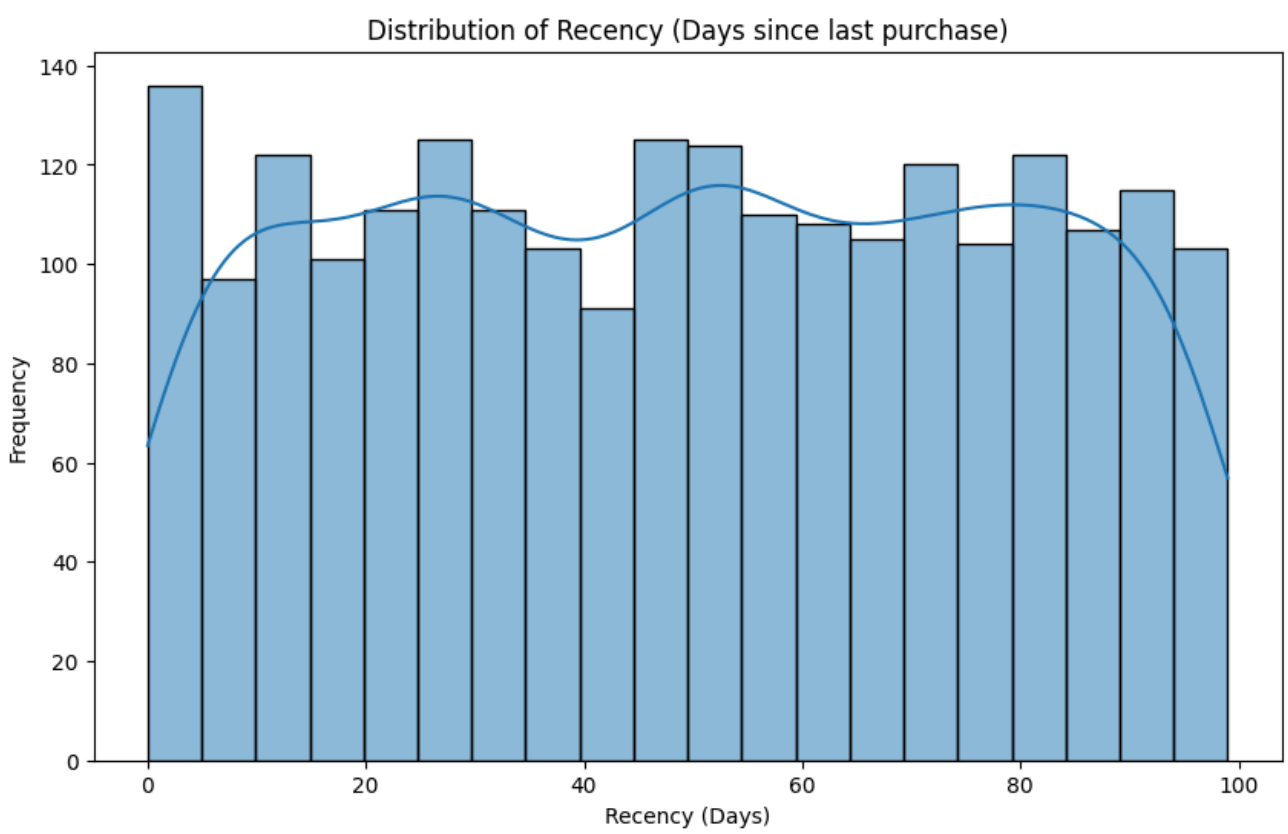
Heatmap: Correlation Matrix

```
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```



Histogram: Distribution of Recency

```
plt.figure(figsize=(10, 6))
sns.histplot(df['Recency'], bins=20, kde=True)
plt.title('Distribution of Recency (Days since last purchase)')
plt.xlabel('Recency (Days)')
plt.ylabel('Frequency')
plt.show()
```



Pie Chart: Response Rate

Suggested code may be subject to a license | MarkKostantine/Spotify-Visualization-ML-Prediction

```
plt.figure(figsize=(8, 8))
plt.pie(response_rate, labels=response_rate.index, autopct='%1.1f%%',
startangle=90)
plt.title('Overall Response Rate')
plt.show()
```