```
Start coding or generate with AI.
```

Dataset uploading

```
# prompt: DATASET UPLOAD CODE

from google.colab import files
uploaded = files.upload()
# prompt: handle missing values for an csv file in google colab

import pandas as pd

# Load the CSV file into a pandas DataFrame
df = pd.read_csv('customer.csv')
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving customer.csv to customer (3).csv
```

Handling Missing Values

```
# prompt: check for missing values and fill the missing values in the above dataset

# Check for missing values
print(df.isnull().sum())

# Fill missing values with mean for numerical columns
numerical_cols = df.select_dtypes(include=['number']).columns
df[numerical_cols] = df[numerical_cols].fillna(df[numerical_cols].mean())

# Fill missing values with mode for categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns
df[categorical_cols] = df[categorical_cols].fillna(df[categorical_cols].mode().iloc[0])

# Verify if missing values are filled
print(df.isnull().sum())
```

```
Year                          0
Date                          0
Question_Number               0
Question                      0
Number_of_Respondents         0
Very_Satisfied                0
Satisfied                     0
Neutral                       0
Dissatisfied                  0
Very_Dissatisfied             0
Very_Satisfied_or_Satisfied   0
ObjectId                      0
dtype: int64
Year                          0
Date                          0
Question_Number               0
Question                      0
Number_of_Respondents         0
Very_Satisfied                0
Satisfied                     0
Neutral                       0
Dissatisfied                  0
Very_Dissatisfied             0
Very_Satisfied_or_Satisfied   0
ObjectId                      0
dtype: int64
```

Duplicate records

```
# prompt: check for the duplicate records and remove them

# Check for duplicate rows
duplicate_rows = df[df.duplicated()]

# Print the duplicate rows
print("Duplicate Rows:")
print(duplicate_rows)

# Remove duplicate rows
df = df.drop_duplicates()
```

```
# Print the DataFrame after removing duplicates
print("\nDataFrame after removing duplicates:")
df
```

```
Duplicate Rows:
Empty DataFrame
Columns: [Year, Date, Question_Number, Question, Number_of_Respondents, Very_Satisfied, Satisfied, Neutral, Dissatisfied, Very_Dissa
Index: []

DataFrame after removing duplicates:
```

	Year	Date	Question_Number	Question	Number_of_Respondents	Very_Satisfied	Satisfied	Neutral	Dissatisfied	Very_Dis
0	2017	2017/10/31 07:00:00+00	7-13	Overall quality of customer service	882	22.16	47.61	25.03	3.15	
1	2016	2016/10/31 07:00:00+00	26	Overall quality of customer service	1202	22.39	47.83	23.37	4.35	
2	2015	2015/10/31 07:00:00+00	Survey Not Conducted	Survey Not Conducted	99999	99999.00	99999.00	99999.00	99999.00	
3	2014	2014/10/31 07:00:00+00	10b	How easy was the City to contact	493	35.95	46.07	8.26	7.44	
4	2014	2014/10/31 07:00:00+00	10c	The way you were treated	493	44.15	40.25	8.83	4.72	
5	2014	2014/10/31 07:00:00+00	10d	The accuracy of the information you were given	493	36.76	40.55	14.08	6.51	
6	2014	2014/10/31 07:00:00+00	10e	How quickly staff responded to your request	493	38.05	37.63	13.10	7.69	
7	2014	2014/10/31 07:00:00+00	10f	How well your issue was handled	493	37.92	34.58	12.50	8.96	
8	2013	2013/10/31 07:00:00+00	14b	How easy was the City to contact	428	37.68	44.31	8.06	7.58	
9	2013	2013/10/31 07:00:00+00	14c	The way you were treated	428	44.47	40.14	8.17	4.09	
10	2013	2013/10/31 07:00:00+00	14d	The accuracy of the information you were	428	39.51	39.27	13.17	4.63	

Outliers

```
# prompt: check for the outliers in the above dataset

import pandas as pd
import numpy as np

# Assuming 'df' is your DataFrame with numerical features

def find_outliers_iqr(data):
    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data < lower_bound) | (data > upper_bound)]
    return outliers


numerical_features = df.select_dtypes(include=np.number).columns
for col in numerical_features:
    outliers = find_outliers_iqr(df[col])
    print(f"Outliers in {col}:")
```

```
print(outliers)
print("-" * 20)
```

```
Outliers in Year:
Series([], Name: Year, dtype: int64)
--------------------
Outliers in Number_of_Respondents:
0      882
1     1202
2    99999
Name: Number_of_Respondents, dtype: int64
--------------------
Outliers in Very_Satisfied:
0      22.16
1      22.39
2    99999.00
Name: Very_Satisfied, dtype: float64
--------------------
Outliers in Satisfied:
2    99999.0
Name: Satisfied, dtype: float64
--------------------
Outliers in Neutral:
0      25.03
1      23.37
2    99999.00
Name: Neutral, dtype: float64
--------------------
Outliers in Dissatisfied:
2    99999.0
Name: Dissatisfied, dtype: float64
--------------------
Outliers in Very_Dissatisfied:
2    99999.0
Name: Very_Dissatisfied, dtype: float64
--------------------
Outliers in Very_Satisfied_or_Satisfied:
2    99999.0
Name: Very_Satisfied_or_Satisfied, dtype: float64
--------------------
Outliers in ObjectId:
Series([], Name: ObjectId, dtype: int64)
--------------------
```

16	2012	2012/10/31 07:00:00+00	14e	How quickly responded to request	403	39.00	38.00	10.00	7.00
17	2012	2012/10/31 07:00:00+00	14f	How well issue was handled	403	40.00	34.00	11.00	9.00
18	2011	2011/10/31 07:00:00+00	14b	How easy was the contact	416	33.17	44.31	13.08	7.75
19	2011	2011/10/31 07:00:00+00	14c	The way you were treated	416	42.82	38.69	11.68	4.62
20	2011	2011/10/31 07:00:00+00	14d	Accuracy of information you were given	416	38.35	38.10	14.79	5.26
21	2011	2011/10/31 07:00:00+00	14e	How quickly staff responded to your request	416	36.54	37.78	13.33	6.67
22	2011	2011/10/31 07:00:00+00	14f	How well your issue was handled	416	37.06	34.83	12.69	8.46
23	2010	2010/10/31 07:00:00+00	14B	How easy was the city to contact	424	37.68	43.00	10.63	6.76
24	2010	2010/10/31 07:00:00+00	14C	The way you were treated	424	41.56	40.10	11.98	3.91
25	2010	2010/10/31	14D	Accuracy of information	424	37.50	39.07	12.78	6.63

Standardization

```
# prompt: standardize the above dataset

from sklearn.preprocessing import StandardScaler

# Assuming 'df' is your DataFrame with numerical features

# Create a StandardScaler object
scaler = StandardScaler()

# Select numerical columns for standardization
numerical_cols = df.select_dtypes(include=np.number).columns

# Fit and transform the numerical columns
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Print the standardized DataFrame
print("\nStandardized DataFrame:")
df
```

Standardized DataFrame:

	Year	Date	Question_Number	Question	Number_of_Respondents	Very_Satisfied	Satisfied	Neutral	Dissatisfied	Ve
0	2.483682	2017/10/31 07:00:00+00	7-13	Overall quality of customer service	-0.170565	-0.193284	-0.192013	-0.191776	-0.192621	
1	1.940376	2016/10/31 07:00:00+00	26	Overall quality of customer service	-0.153239	-0.193271	-0.192001	-0.191865	-0.192557	
2	1.397071	2015/10/31 07:00:00+00	Survey Not Conducted	Survey Not Conducted	5.195945	5.196152	5.196152	5.196152	5.196152	
3	0.853766	2014/10/31 07:00:00+00	10b	How easy was the City to contact	-0.191626	-0.192540	-0.192096	-0.192679	-0.192390	
4	0.853766	2014/10/31 07:00:00+00	10c	The way you were treated	-0.191626	-0.192098	-0.192409	-0.192649	-0.192537	
5	0.853766	2014/10/31 07:00:00+00	10d	The accuracy of the information you were given	-0.191626	-0.192497	-0.192393	-0.192366	-0.192440	
6	0.853766	2014/10/31 07:00:00+00	10e	How quickly staff responded to your request	-0.191626	-0.192427	-0.192551	-0.192419	-0.192377	
7	0.853766	2014/10/31 07:00:00+00	10f	How well your issue was handled	-0.191626	-0.192434	-0.192715	-0.192451	-0.192308	
8	0.310460	2013/10/31 07:00:00+00	14b	How easy was the City to contact	-0.195146	-0.192447	-0.192190	-0.192690	-0.192382	
9	0.310460	2013/10/31 07:00:00+00	14c	The way you were treated	-0.195146	-0.192081	-0.192415	-0.192684	-0.192571	
10	0.310460	2013/10/31 07:00:00+00	14d	The accuracy of the information you were given	-0.195146	-0.192348	-0.192462	-0.192415	-0.192541	

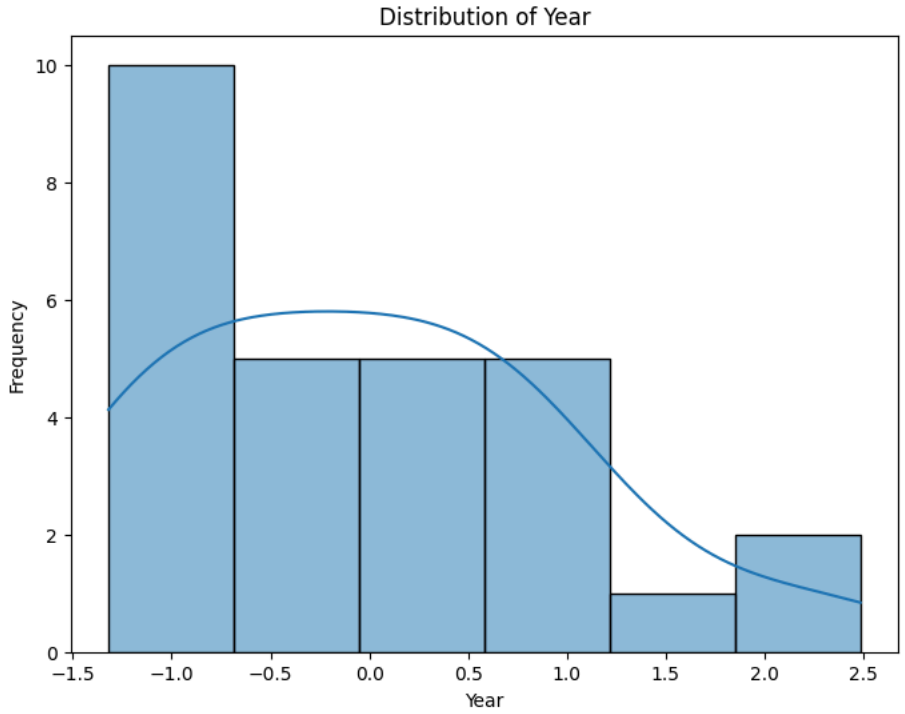
EDA

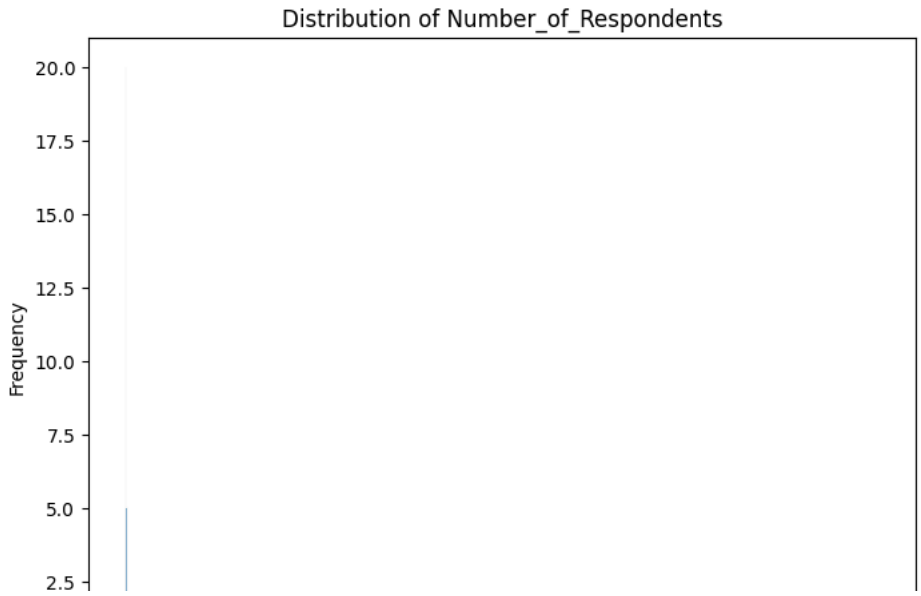
How

```
# prompt: visualize the dataset using eda by univariate, bivariate analysis

import matplotlib.pyplot as plt
import seaborn as sns

# Univariate Analysis

# Histograms for numerical features
for col in numerical_features:
    plt.figure(figsize=(8, 6))
    sns.histplot(df[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()

# Box plots for numerical features
for col in numerical_features:
    plt.figure(figsize=(8, 6))
    sns.boxplot(df[col])
    plt.title(f'Box Plot of {col}')
    plt.ylabel(col)
    plt.show()

# Count plots for categorical features
for col in categorical_cols:
```

```
    plt.figure(figsize=(8, 6))
    sns.countplot(x=col, data=df)
    plt.title(f'Count Plot of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better readability
    plt.show()


# Bivariate Analysis

# Scatter plots for numerical features
for col1 in numerical_features:
    for col2 in numerical_features:
        if col1 != col2:
            plt.figure(figsize=(8, 6))
            sns.scatterplot(x=col1, y=col2, data=df)
            plt.title(f'Scatter Plot of {col1} vs {col2}')
            plt.xlabel(col1)
            plt.ylabel(col2)
            plt.show()

# Correlation Heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(df[numerical_cols].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Numerical Features')
plt.show()

# Box plots for numerical features grouped by a categorical feature
for col in numerical_features:
  for cat_col in categorical_cols:
    plt.figure(figsize=(10,6))
    sns.boxplot(x = cat_col, y = col, data=df)
    plt.title(f"Box plot of {col} grouped by {cat_col}")
    plt.show()
```

25	-1.319456	2010/10/31 07:00:00+00	14D	information you were given	-0.195362	-0.192452	-0.192473	-0.192436	-0.192434
26	-1.319456	2010/10/31 07:00:00+00	14E	How quickly staff responded to your request	-0.195362	-0.192348	-0.192843	-0.192257	-0.192502
27	-1.319456	2010/10/31 07:00:00+00	14F	How well your issue was handled	-0.195362	-0.192520	-0.192856	-0.192263	-0.192386





Feature Engineering

```
from sklearn.preprocessing import StandardScaler

# Assuming 'df' is your DataFrame with numerical features

# Create a StandardScaler object
scaler = StandardScaler()

# Select numerical columns for standardization
numerical_cols = df.select_dtypes(include=np.number).columns

# Fit and transform the numerical columns
# Instead of directly assigning to df[numerical_cols], create a new DataFrame
scaled_data = scaler.fit_transform(df[numerical_cols])
scaled_df = pd.DataFrame(scaled_data, columns=numerical_cols, index=df.index)

# Update the original DataFrame with the scaled values
df[numerical_cols] = scaled_df[numerical_cols]


# Print the standardized DataFrame
print("\nStandardized DataFrame:")
df
```

Standardized DataFrame:

	Year	Date	Question_Number	Question	Number_of_Respondents	Very_Satisfied	Satisfied	Neutral	Dissatisfied	Ve
0	2.483682	2017/10/31 07:00:00+00	7-13	Overall quality of customer service	-0.170565	-0.193284	-0.192013	-0.191776	-0.192621	
1	1.940376	2016/10/31 07:00:00+00	26	Overall quality of customer service	-0.153239	-0.193271	-0.192001	-0.191865	-0.192557	
2	1.397071	2015/10/31 07:00:00+00	Survey Not Conducted	Survey Not Conducted	5.195945	5.196152	5.196152	5.196152	5.196152	
3	0.853766	2014/10/31 07:00:00+00	10b	How easy was the City to contact	-0.191626	-0.192540	-0.192096	-0.192679	-0.192390	
4	0.853766	2014/10/31 07:00:00+00	10c	The way you were treated	-0.191626	-0.192098	-0.192409	-0.192649	-0.192537	
5	0.853766	2014/10/31 07:00:00+00	10d	The accuracy of the information you were given	-0.191626	-0.192497	-0.192393	-0.192366	-0.192440	
6	0.853766	2014/10/31 07:00:00+00	10e	How quickly staff responded to your request	-0.191626	-0.192427	-0.192551	-0.192419	-0.192377	
7	0.853766	2014/10/31 07:00:00+00	10f	How well your issue was handled	-0.191626	-0.192434	-0.192715	-0.192451	-0.192308	
8	0.310460	2013/10/31 07:00:00+00	14b	How easy was the City to contact	-0.195146	-0.192447	-0.192190	-0.192690	-0.192382	
9	0.310460	2013/10/31 07:00:00+00	14c	The way you were treated	-0.195146	-0.192081	-0.192415	-0.192684	-0.192571	
10	0.310460	2013/10/31 07:00:00+00	14d	The accuracy of the information you were given	-0.195146	-0.192348	-0.192462	-0.192415	-0.192541	

Model Building

How

```
# prompt: build the model for the above dataset in simple

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Assuming your target variable is 'Exited'
# Check if 'Exited' column exists in the DataFrame
if 'Exited' not in df.columns:
    # If not found, print an error message and stop
    print("Error: 'Exited' column not found in the DataFrame.")
    # You might need to investigate why 'Exited' is missing and fix it
    # For example, if it's a typo, correct the column name
    # Or if it's missing from the data, you need to add it
else:
    X = df.drop('Exited', axis=1)
    y = df['Exited']

    # Convert categorical features to numerical using one-hot encoding
    X = pd.get_dummies(X, drop_first=True)

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Initialize and train a Logistic Regression model
    model = LogisticRegression()
```

```
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

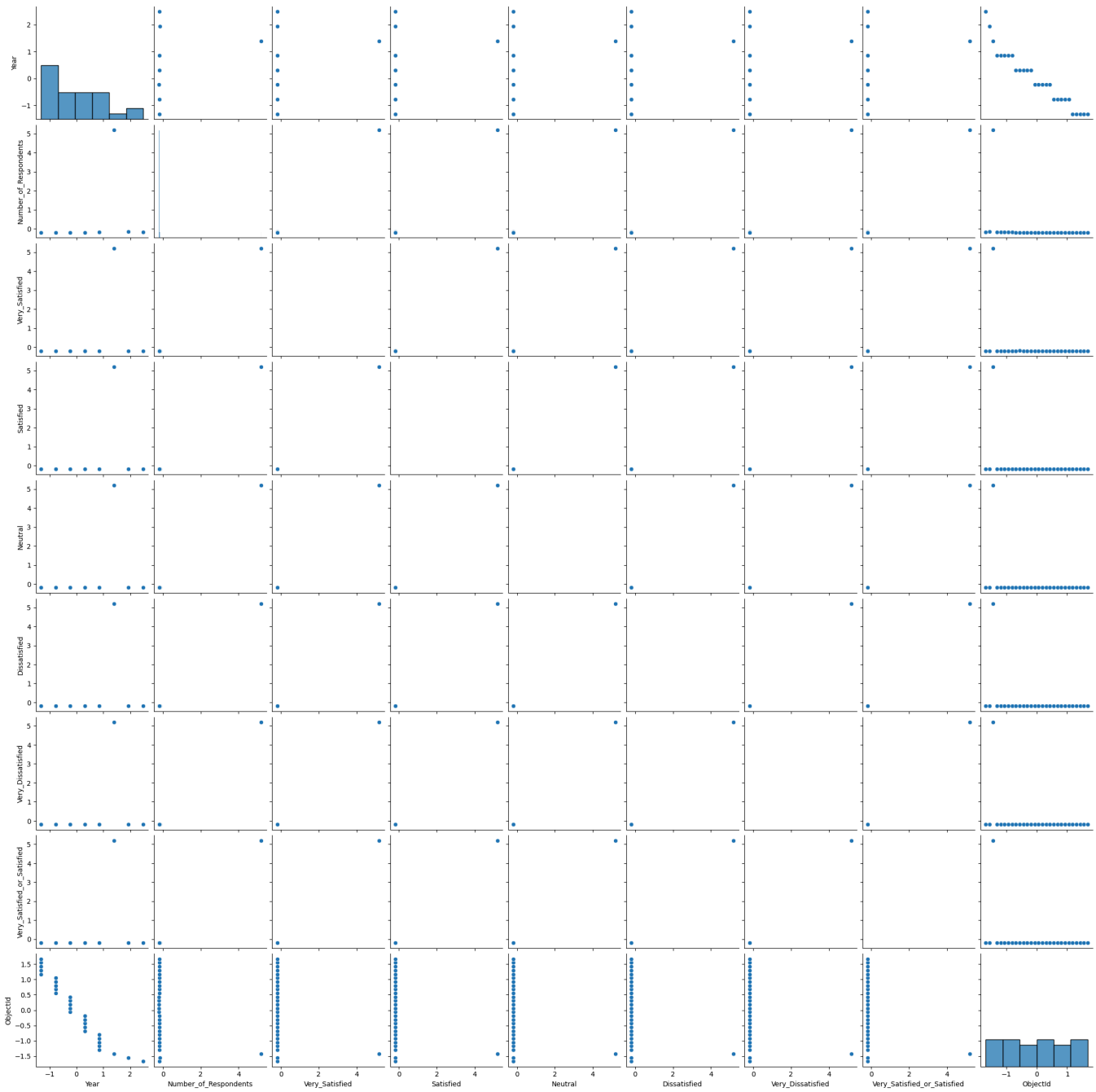
```
Error: 'Exited' column not found in the DataFrame.
```

20	-0.776151	2011/10/31 07:00:00+00	14d	Accuracy of information you were given	-0.195795	-0.192411	-0.192525	-0.192327	-0.192507

Visualization

```
# prompt: easy code for visualizing the above dataset

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame (loaded from customer.csv as in your code)

# Example 1: Pairplot for numerical features
sns.pairplot(df.select_dtypes(include=np.number))
plt.show()


# Example 2: Correlation heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()

# Example 3: Boxplot for a specific numerical feature grouped by a categorical feature
plt.figure(figsize=(10, 6))
sns.boxplot(x='Geography', y='CreditScore', data=df)  # Replace 'Geography' and 'CreditScore' as needed
plt.title("CreditScore Distribution by Geography")
plt.show()

# Example 4: Countplot for a categorical feature
plt.figure(figsize=(8, 6))
sns.countplot(x='Gender', data=df) # Replace 'Gender' as needed
plt.title("Count of Gender")
plt.show()
```

27	-1.319456	2010/10/31 07:00:00+00	14F	your issue was handled	-0.195362	-0.192520	-0.192856	-0.192263	-0.192386




```
# prompt: DATASET UPLOAD CODE

from google.colab import files
uploaded = files.upload()
# prompt: handle missing values for an csv file in google colab

import pandas as pd

# Load the CSV file into a pandas DataFrame
df = pd.read_csv('customer.csv')
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving customer.csv to customer.csv

```
ValueError: could not convert string to float: '2017/10/31 07:00:00+00'
```

```
# prompt: include sf.head() and print it

from google.colab import files
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
uploaded = files.upload()

# Load the CSV file into a pandas DataFrame
df = pd.read_csv('customer.csv')

# Assuming 'df' is your DataFrame
print(df.head())
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving customer.csv to customer (1).csv
   Year                  Date       Question_Number  \
0  2017  2017/10/31 07:00:00+00                 7-13
1  2016  2016/10/31 07:00:00+00                   26
2  2015  2015/10/31 07:00:00+00  Survey Not Conducted
3  2014  2014/10/31 07:00:00+00                  10b
4  2014  2014/10/31 07:00:00+00                  10c

                            Question  Number_of_Respondents  Very_Satisfied  \
0  Overall quality of customer service                   882           22.16
1  Overall quality of customer service                  1202           22.39
2                 Survey Not Conducted                 99999        99999.00
3     How easy was the City to contact                   493           35.95
4             The way you were treated                   493           44.15

   Satisfied   Neutral  Dissatisfied  Very_Dissatisfied  \
0      47.61     25.03          3.15               2.05
1      47.83     23.37          4.35               2.07
2   99999.00  99999.00      99999.00           99999.00
3      46.07      8.26          7.44               2.27
4      40.25      8.83          4.72               2.05

   Very_Satisfied_or_Satisfied  ObjectId
0                        69.77         1
1                        70.22         2
2                     99999.00         3
3                        82.02         4
4                        84.39         5
```

```
# Import necessary libraries
!pip install scikit-learn joblib
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression  # Or any other model you prefer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import joblib

# Assume you have a DataFrame called 'df' with 'text' and 'sentiment' columns
# X = df['text']
# y = df['sentiment']

# Create and train a TfidfVectorizer
vectorizer = TfidfVectorizer()
# X_vec = vectorizer.fit_transform(X)

# Split data into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X_vec, y, test_size=0.2, random_state=42)

# Create and train a sentiment model (e.g., Logistic Regression)
# model = LogisticRegression()
# model.fit(X_train, y_train)

# Save the trained model and vectorizer
# joblib.dump(model, 'sentiment_model.joblib')
# joblib.dump(vectorizer, 'tfidf_vectorizer.joblib')
```