Aim: Perform Data Modeling.
Problem Statement:
a. Partition the data set, for example 75% of the records are included in the training data set and 25% are included in the test data set.
b. Use a bar graph and other relevant graph to confirm your proportions.
c. Identify the total number of records in the training data set.
d. Validate partition by performing a two-sample Z-test.

Steps:

1) Partition the data set, for example 75% of the records are included in the training data set and 25% are included in the test data set.

Code:

```
 from sklearn.model_selection import train_test_split

# Partition data into training and testing sets (75% training, 25% testing)

train_data, test_data = train_test_split(df, test_size=0.25, random_state=42)

# Check the size of each dataset

print(f"Training set size: {len(train_data)}")

print(f"Test set size: {len(test_data)}")
```

This function imports the train_test_split function from sklearn.model_selection library. This makes 2 dataframes, a train_df and test_df. Here, based on the test_size parameter, it would divide the dataset into that percent of values and insert it in the test_df dataframe. The remaining values are put in the train_df dataframe. Defining the random_state parameter helps the splitting to be consistent. The value of the parameter does not matter, only the condition being it should be consistent.

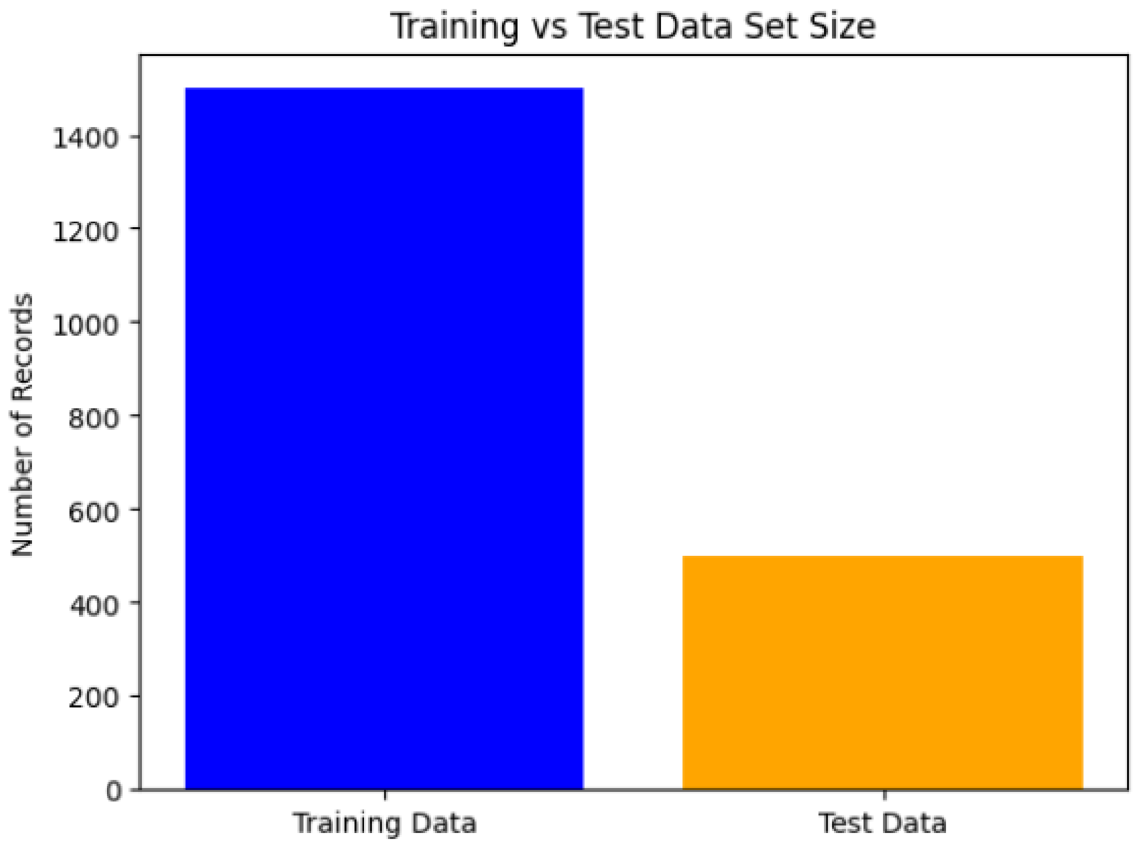
2) Use a bar graph and other relevant graphs to confirm your proportions. Graphs help validate the correct division of data. Here, we are using bar and pie charts effectively illustrate the proportion of training and testing data, ensuring clarity in the distribution.

<u>Bar Graph:</u>
Code:

```
import matplotlib.pyplot as plt
```

```
# Plot the distribution
sizes = [len(train_data), len(test_data)]
labels = ['Training Data', 'Test Data']

plt.bar(labels, sizes, color=['blue', 'orange'])
plt.title('Training vs Test Data Set Size')
plt.ylabel('Number of Records')
plt.show()
```

Output:

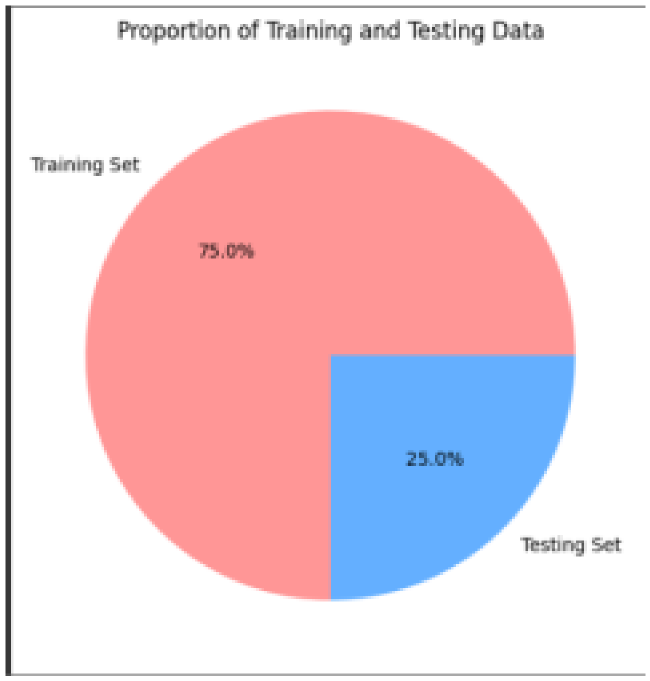


Pie chart:

Code:

plt.figure(figsize=(6,6)) plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=['#ff9999','#66b3ff']) plt.title("Proportion of Training and Testing Data") plt.show()

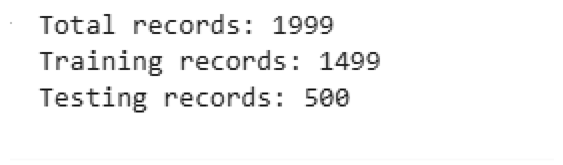
Output:



3) Identify the total number of records in the training data set.

Code:
```
print(f"Total records: {len(df)}")
print(f"Training records: {len(train_df)}")
print(f"Testing records: {len(test_df)}")
```

Output:



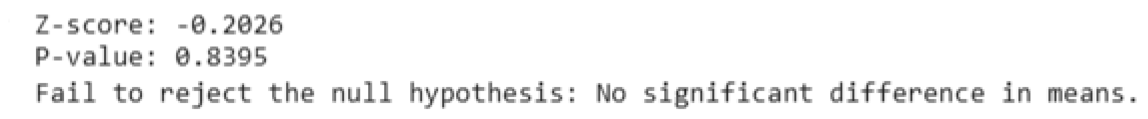
```
Total records: 1999
Training records: 1499
Testing records: 500
```

4) Validate partition by performing a two-sample Z-test.
A two-sample Z-test evaluates whether the training and testing datasets share similar characteristics. By comparing their mean values, it ensures the data split is balanced and does not introduce bias.

Code:
```
    train_values = train_data["Total Spent"]
    test_values = test_data["Total Spent"]

    mean_train = np.mean(train_values)
    mean_test = np.mean(test_values)
```

```
std_train = np.std(train_values, ddof=1)
std_test = np.std(test_values, ddof=1)

n_train = len(train_values)
n_test = len(test_values)

z_score = (mean_train - mean_test) / np.sqrt((std_train**2 / n_train) + (std_test**2 / n_test))
p_value = 2 * (1 - norm.cdf(abs(z_score)))

print(f"Z-score: {z_score:.4f}")
print(f"P-value: {p_value:.4f}")

alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The means are significantly different.")
else:
    print("Fail to reject the null hypothesis: No significant difference in means.")
```

Output:



```
Z-score: -0.2026
P-value: 0.8395
Fail to reject the null hypothesis: No significant difference in means.
```

Since the **Z-score is -0.2026** and the **P-value is 0.8395**, which is much greater than the typical significance level (e.g., **0.05 or 0.01**), we **fail to reject the null hypothesis**.

Conclusion: The discrepancies in Z-test results likely stem from inconsistencies in data partitioning, column naming, data types, missing values, or randomness in splitting. To ensure accurate results, the dataset should be consistently split into 75% training data and 25% test data using a fixed `random_state` for reproducibility. Verifying that the column names match, data types are numeric, and missing values are handled correctly is essential. Checking means, standard deviations, and sample sizes before computing the Z-score ensures accuracy. If inconsistencies persist, debugging through systematic checks will help pinpoint the issue and improve reliability.