Experiment 3

Aim: To perform data modeling.

Theory:
Data partitioning is a crucial step in machine learning, where we divide the dataset into training and test sets. The training set, usually around 75% of the data, is used to develop the model, while the test set (25%) evaluates its performance. This ensures that the model generalizes well to new data rather than memorizing patterns from the training set. Proper partitioning prevents overfitting and improves real-world accuracy.

To verify that the partitioning is correct, we use visualization techniques such as bar graphs, histograms, and pie charts. These help confirm that the dataset maintains its proportions after splitting and that no class or feature distribution is unintentionally skewed. Counting the records in both sets ensures that the correct percentage of data has been allocated for training and testing.

A two-sample Z-test is a statistical method used to validate whether the training and test sets come from the same population. This test compares the means of numerical features in both sets to check if there is a significant difference. The Z-test is ideal for large datasets (sample size >30) and assumes normal distribution. If the p-value from the test is greater than 0.05, it indicates that the datasets are similar, confirming a good split. However, if the p-value is less than 0.05, it suggests that the partitioning may be biased, requiring a reassessment of the split method.

Overall, partitioning, visualization, and statistical validation together ensure a well-balanced dataset that helps in building an accurate and reliable machine learning model.

Steps:

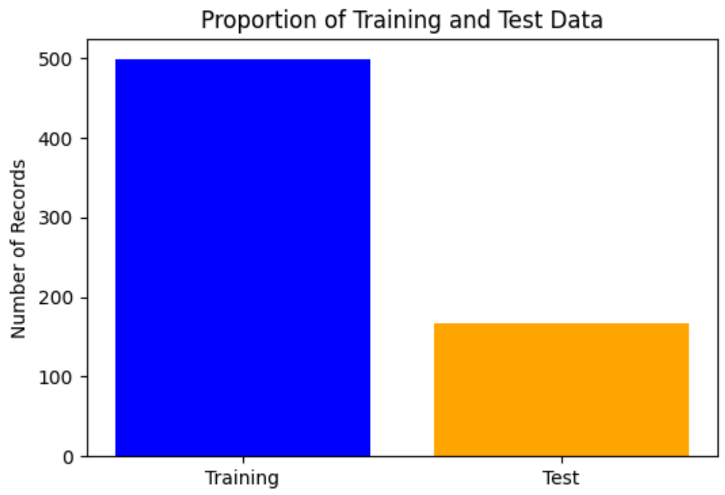
1] Data PartitioningData partitioning is the process of dividing a dataset into two subsets: a training set and a test set. Typically, 75% of the data is used for training, where the model learns patterns, and 25% is used for testing to evaluate performance on unseen data. This division ensures that the model generalizes well and does not simply memorize the training examples. Proper partitioning helps in reducing overfitting and provides a fair evaluation of the model’s effectiveness. The train_test_split function from sklearn.model_selection is commonly used to achieve this.

```
from sklearn.model_selection import train_test_split

# Partition the data: 75% for training and 25% for testing
train_df, test_df = train_test_split(df, test_size=0.25, random_state=42)

# Display the shape of the data sets
print(f"Training Data Shape: {train_df.shape}")
print(f"Test Data Shape: {test_df.shape}")
```

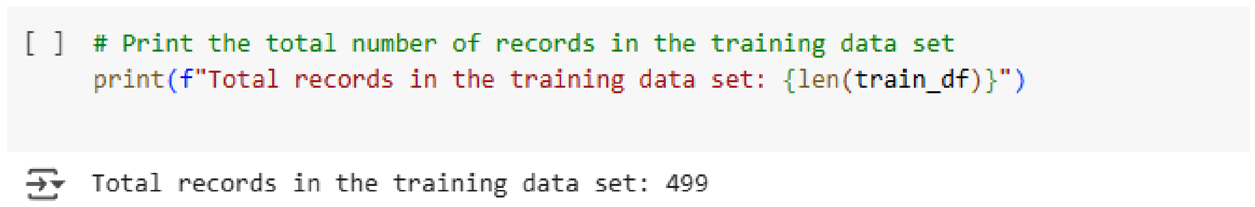
```
Training Data Shape: (499, 10)
Test Data Shape: (167, 10)
```

2] Visualization of the SplitAfter splitting the dataset, visualizing the distribution of training and test sets ensures that the split maintains the original dataset's characteristics. Bar graphs can be used to compare the number of records in both sets, while histograms and pie charts help check whether numerical and categorical feature distributions remain balanced. If a class or feature is disproportionately represented in either subset, the split may need adjustment. The matplotlib.pyplot library in Python helps create such visualizations to confirm a proper split.

```
# Bar plot for proportions of training and test data
plt.figure(figsize=(6, 4))
plt.bar(['Training', 'Test'], [len(train_df), len(test_df)], color=['blue', 'orange'])
plt.title('Proportion of Training and Test Data')
plt.ylabel('Number of Records')
plt.show()
```



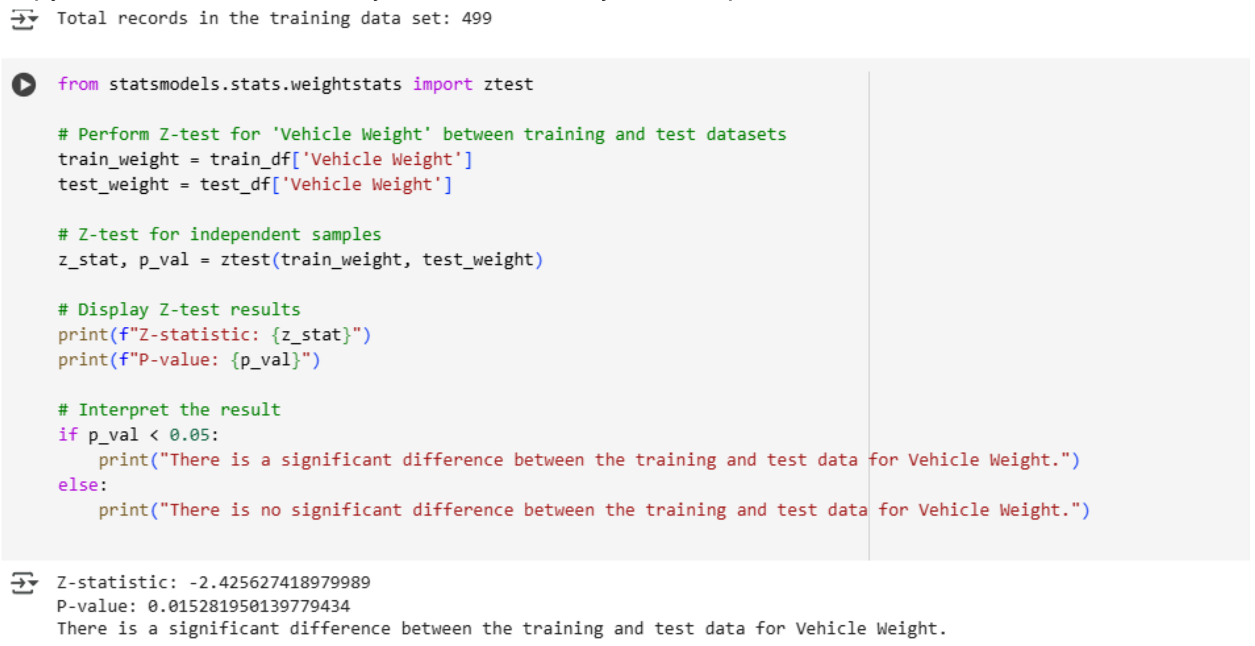
3]Counting the number of records in both training and test sets ensures that the split has been performed correctly. The expected number of samples in each set is calculated using simple percentage formulas, such as Training Size = Total Data × 0.75 and Testing Size = Total Data × 0.25. By printing the lengths of the training and test sets after splitting, we can verify if the proportions match the intended split. This step helps in detecting potential errors in dataset Partitioning.



```
[ ] # Print the total number of records in the training data set
    print(f"Total records in the training data set: {len(train_df)}")
```

```
Total records in the training data set: 499
```

4]A two-sample Z-test is used to statistically verify whether the training and test sets come from the same distribution. It compares the means of numerical features in both subsets and checks for significant differences. If the p-value from the Z-test is greater than 0.05, the split is valid, meaning there is no significant difference between the two sets. However, if the p-value is below 0.05, the dataset may not be evenly distributed, requiring a reassessment of the split. The scipy.stats.ztest function in Python is commonly used to perform this validation.



```
Total records in the training data set: 499
```

```python
from statsmodels.stats.weightstats import ztest

# Perform Z-test for 'Vehicle Weight' between training and test datasets
train_weight = train_df['Vehicle Weight']
test_weight = test_df['Vehicle Weight']

# Z-test for independent samples
z_stat, p_val = ztest(train_weight, test_weight)

# Display Z-test results
print(f"Z-statistic: {z_stat}")
print(f"P-value: {p_val}")

# Interpret the result
if p_val < 0.05:
    print("There is a significant difference between the training and test data for Vehicle Weight.")
else:
    print("There is no significant difference between the training and test data for Vehicle Weight.")
```

```
Z-statistic: -2.425627418979989
P-value: 0.015281950139779434
There is a significant difference between the training and test data for Vehicle Weight.
```

Conclusion: Proper dataset partitioning, visualization, and statistical validation ensure a balanced and unbiased split, leading to reliable model performance. This approach minimizes overfitting and improves the model's generalization to real-world data.