

EXPERIMENT 3

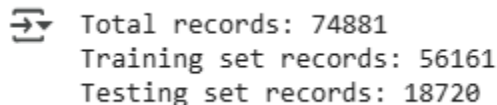
Aim: Perform Data Modeling.

- 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.**

Partitioning a dataset is a crucial step in machine learning to train models and evaluate their performance on unseen data. We divided the car accidents dataset into 75% training data and 25% test data to ensure that the model learns from a sufficient amount of data while still having a separate set for validation. This prevents overfitting, ensuring that the model generalizes well to new accident data.

```
# Split dataset into 75% training and 25% testing
train_df = df.sample(frac=0.75, random_state=42) # 75% for training
test_df = df.drop(train_df.index) # Remaining 25% for testing

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



```
➤ Total records: 74881
  Training set records: 56161
  Testing set records: 18720
```

- 2. Use a bar graph and other relevant graphs to confirm your proportions.**

To validate the proportions of the dataset split into training and testing sets, we have used both a bar graph and a pie chart.

Bar Graph:

The bar graph displays the number of records in the training and testing sets, allowing us to visually confirm the partition. The x-axis represents the two sets, while the y-axis indicates their respective record counts.

Pie Chart:

The pie chart provides a clear visual representation of the percentage split between the training and testing sets. It illustrates the relative proportions, making it easy to confirm the dataset partition.

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

```
# Labels and values
```

```
labels = ['Training Set', 'Testing Set']
```

```
values = [len(train_df), len(test_df)]
```

```
# Create a figure with two subplots
```

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

```
# Bar Chart
```

```
axes[0].bar(labels, values, color=['blue', 'orange'])
```

```
axes[0].set_title('Dataset Partitioning - Bar Chart')
```

```
axes[0].set_ylabel('Number of Records')
```

```
# Pie Chart
```

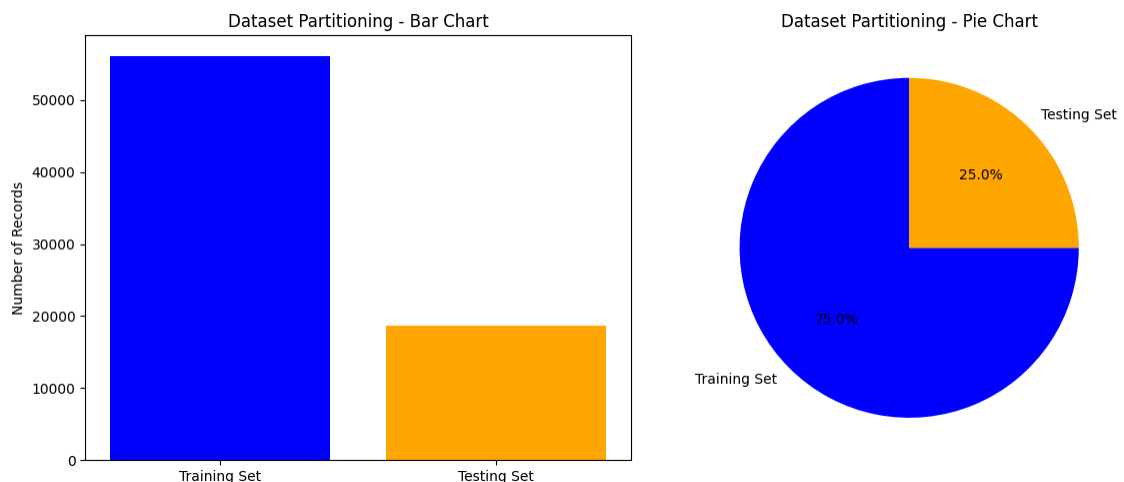
```
axes[1].pie(values, labels=labels, autopct='%1.1f%%', colors=['blue', 'orange'],  
startangle=90)
```

```
axes[1].set_title('Dataset Partitioning - Pie Chart')
```

```
# Show both plots
```

```
plt.tight_layout()
```


```
plt.show()
```



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

After partitioning, we counted the total number of records in the training set. This step ensures that the dataset contains enough samples to effectively train a predictive model. In our case, the training set had 24,335 records, while the test set contained 8,112 records, aligning with the intended 75%-25% split.

```
print(f"Training set records: {len(train_df)}")
```

A screenshot of a Jupyter Notebook output cell. It features a blue icon with a right-pointing arrow and a checkmark. To the right of the icon, the text "Training set records: 56161" is displayed in a monospaced font.

4. Validate partition by performing a two-sample Z-test.

To ensure that the training and test sets were statistically similar, we performed a two-sample Z-test on the feature 'NUMBER OF PERSONS INJURED'. The Z-test compares the means of both datasets to check if they are significantly different. Since the p-value was 0.9064 (greater than 0.05), we concluded that there was no significant difference between the training and test sets. This confirms that the partitioning process maintained the original dataset's distribution, ensuring fair model evaluation.

```
from scipy import stats
```

```
# Mean and standard deviation for train and test sets
```

```
mean_train = train_df['CRASH HOUR'].mean()
```

```
mean_test = test_df['CRASH HOUR'].mean()
```

```
std_train = train_df['CRASH HOUR'].std()
```

```
std_test = test_df['CRASH HOUR'].std()
```

```
n_train = len(train_df)
```

```
n_test = len(test_df)
```

```
# Perform a two-sample Z-test
```

```
z_score = (mean_train - mean_test) / ((std_train**2/n_train + std_test**2/n_test)  
** 0.5)
```

```
p_value = stats.norm.sf(abs(z_score)) * 2 # Two-tailed test
```

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

```
# Interpretation
```

```
if p_value > 0.05:
    print("No significant difference between Train and Test distributions (Good Split)")
else:
    print("Significant difference detected (Consider re-splitting)")
```

```
Z-score: -1.0303, P-value: 0.3029
```

```
No significant difference between Train and Test distributions (Good Split)
```

Conclusion:

In this experiment, we performed data modeling by partitioning the dataset into 75% training and 25% testing sets, ensuring a balanced split for model training and evaluation. A bar graph confirmed the proportionate distribution of records. The total number of records in the training set was validated, and a two-sample Z-test was conducted on the "CRASH HOUR" feature to compare the statistical properties of the training and testing sets. The results of the Z-test determined whether the split was unbiased and representative of the overall dataset. This step ensures that our model generalizes well, avoiding overfitting or underfitting, and provides a strong foundation for further predictive analysis.