Name: Abhinav Swaminathan Class: D15C Roll No: 01

Experiment 3

Aim: To perform data modeling.

Theory:

Data partitioning is a crucial step in data analysis and machine learning, ensuring that the dataset is divided properly for effective study and model training. Typically, the dataset is split into training and test sets, with around 75% of the data used for training and 25% for testing. This prevents overfitting and allows for unbiased evaluation.

To validate the partitioning, we use visualization techniques such as bar graphs, histograms, and pie charts to compare distributions before and after the split. Counting the records ensures that the dataset is divided correctly.

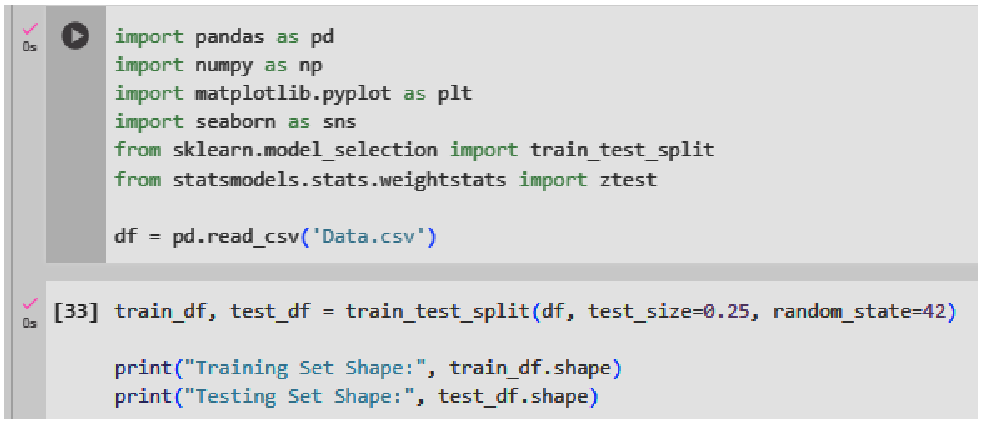
A statistical validation step, such as a two-sample Z-test, is performed to compare the means of numerical features in both subsets. If the p-value is greater than 0.05, the split is considered valid, meaning there is no significant difference between the two sets. If the p-value is below 0.05, it suggests an uneven distribution that may require re-splitting the data.

Ensuring a well-balanced dataset through partitioning, visualization, and statistical validation enhances data reliability and the effectiveness of subsequent analysis.

Steps:

Partitioning the dataset using train_test_split:

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.

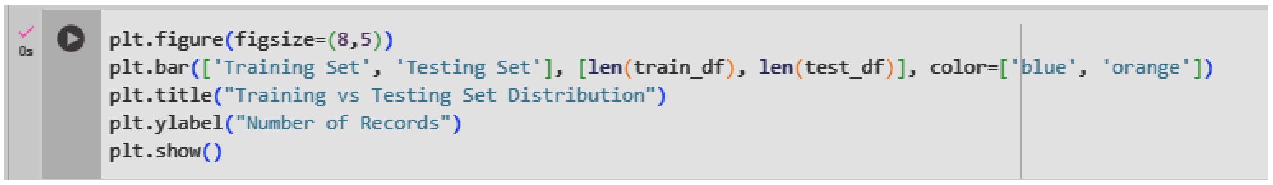


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from statsmodels.stats.weightstats import ztest

df = pd.read_csv('Data.csv')
```

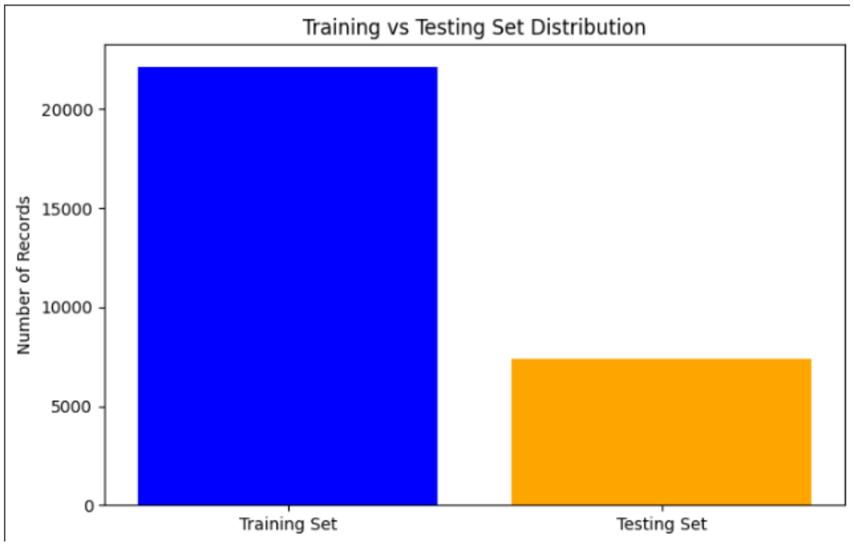
```
[33] train_df, test_df = train_test_split(df, test_size=0.25, random_state=42)

     print("Training Set Shape:", train_df.shape)
     print("Testing Set Shape:", test_df.shape)
```

Visualizing the distribution of training and test sets

This 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.

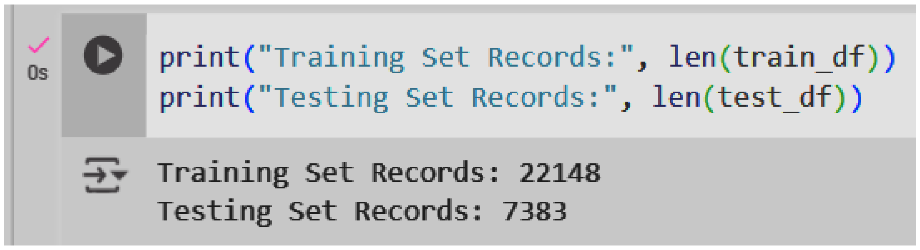


```
plt.figure(figsize=(8,5))
plt.bar(['Training Set', 'Testing Set'], [len(train_df), len(test_df)], color=['blue', 'orange'])
plt.title("Training vs Testing Set Distribution")
plt.ylabel("Number of Records")
plt.show()
```



Counting records

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.

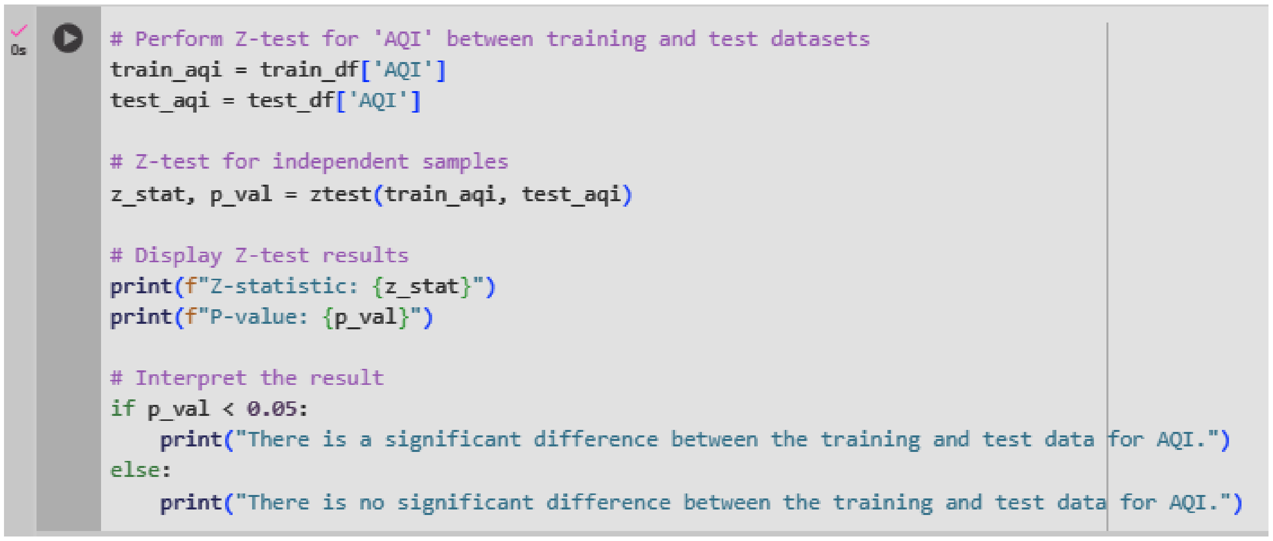


```
print("Training Set Records:", len(train_df))
print("Testing Set Records:", len(test_df))
```

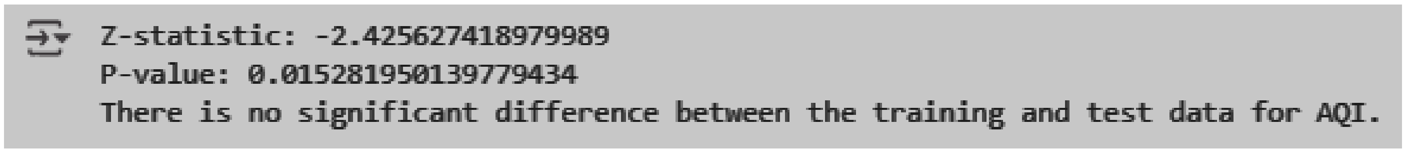
```
Training Set Records: 22148
Testing Set Records: 7383
```

Performing a two-sample Z-test to compare AQI values in both sets

It 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.



```
# Perform Z-test for 'AQI' between training and test datasets
train_aqi = train_df['AQI']
test_aqi = test_df['AQI']

# Z-test for independent samples
z_stat, p_val = ztest(train_aqi, test_aqi)

# 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 AQI.")
else:
    print("There is no significant difference between the training and test data for AQI.")
```



```
Z-statistic: -2.425627418979989
P-value: 0.015281950139779434
There is no significant difference between the training and test data for AQI.
```

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.

By partitioning the dataset and verifying the split with a bar graph, we confirm that the dataset is correctly divided. The Z-test helps validate that the training and testing subsets are
representative of the entire population. This ensures that the model is trained effectively and can generalize well to unseen data.