Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Experiment No: 6

AIM: To perform different classification algorithms on the dataset

THEORY:

Classification Modelling:

Classification modeling is a supervised machine learning technique used to **categorize data into predefined classes based on patterns learned from training data**. It involves selecting a dataset with features (inputs) and labels (outputs), training a model using algorithms like Decision Tree or Naïve Bayes, and then predicting class labels for new data. The model's performance is **evaluated using metrics like accuracy, precision, recall, and F1-score**. Classification can be **binary** (e.g., Pass/Fail) or **multiclass** (e.g., Low/Medium/High). It is widely used in applications like spam detection, medical diagnosis, and fraud detection.

Different Classifiers are as follows:

- **K-Nearest Neighbors (KNN)**: A simple, non-parametric algorithm that classifies a data point based on the majority class of its 'k' nearest neighbors in feature space. It works well for small datasets but can be slow for large datasets.

- **Naïve Bayes**: A probabilistic classifier based on Bayes' theorem, assuming independence between features. It is fast and effective for text classification and spam filtering but may not perform well if features are highly correlated.

- **Support Vector Machines (SVMs)**: A powerful algorithm that finds the optimal hyperplane to separate classes in a high-dimensional space. It works well for complex datasets but can be computationally expensive for large datasets.

- **Decision Tree**: A tree-based model that splits data into branches based on feature values, leading to a classification outcome. It is easy to interpret but prone to overfitting if not pruned properly.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

<u>DATASET</u>:

The dataset **Electric_Vehicle_Population_Data.csv** contains **information about electric vehicles**, including attributes such as vehicle type (BEV/PHEV), manufacturer, model, battery capacity, and other relevant details. It was used to train classification models to differentiate between BEVs (Battery Electric Vehicles) and PHEVs (Plug-in Hybrid Electric Vehicles).

<u>STEPS</u>:

Step 1: Import required libraries and **load dataset into dataframe**

Code:

```
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 sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
df = pd.read_csv('/content/Electric_Vehicle_Population_Data.csv')
print(df.head())
print(df.info())
```

Output:

```
0       RAV4          Battery Electric Vehicle (BEV)
1    MODEL 3          Battery Electric Vehicle (BEV)
2         X5  Plug-in Hybrid Electric Vehicle (PHEV)
3  RAV4 PRIME  Plug-in Hybrid Electric Vehicle (PHEV)
4    MODEL Y          Battery Electric Vehicle (BEV)

  Clean Alternative Fuel Vehicle (CAFV) Eligibility  Electric Range  \
0           Clean Alternative Fuel Vehicle Eligible          103.0
1           Clean Alternative Fuel Vehicle Eligible          220.0
2           Clean Alternative Fuel Vehicle Eligible           40.0
3           Clean Alternative Fuel Vehicle Eligible           42.0
4 Eligibility unknown as battery range has not b...            0.0

  Base MSRP  Legislative District  DOL Vehicle ID  \
0       0.0                  41.0       186450183
1       0.0                   1.0       478093654
2       0.0                  35.0       274800718
3       0.0                   2.0       260758165
4       0.0                  15.0       236581355

                Vehicle Location                              Electric Utility  \
0  POINT (-122.1621 47.64441)  PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)
1  POINT (-122.20563 47.76144)  PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)
2  POINT (-122.92333 47.03779)                        PUGET SOUND ENERGY INC
3  POINT (-122.81754 46.98876)                        PUGET SOUND ENERGY INC
4  POINT (-120.53145 46.65405)                                    PACIFICORP
```

```
   2020 Census Tract
0       5.303302e+10
1       5.303302e+10
2       5.306701e+10
3       5.306701e+10
4       5.307700e+10
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 232230 entries, 0 to 232229
Data columns (total 17 columns):
 #   Column                                             Non-Null Count   Dtype
---  ------                                             --------------   -----
 0   VIN (1-10)                                         232230 non-null  object
 1   County                                             232226 non-null  object
 2   City                                               232226 non-null  object
 3   State                                              232230 non-null  object
 4   Postal Code                                        232226 non-null  float64
 5   Model Year                                         232230 non-null  int64
 6   Make                                               232230 non-null  object
 7   Model                                              232230 non-null  object
 8   Electric Vehicle Type                              232230 non-null  object
 9   Clean Alternative Fuel Vehicle (CAFV) Eligibility  232230 non-null  object
 10  Electric Range                                     232203 non-null  float64
 11  Base MSRP                                          232203 non-null  float64
 12  Legislative District                               231749 non-null  float64
 13  DOL Vehicle ID                                     232230 non-null  int64
 14  Vehicle Location                                   232219 non-null  object
 15  Electric Utility                                   232226 non-null  object
 16  2020 Census Tract                                  232226 non-null  float64
dtypes: float64(5), int64(2), object(10)
memory usage: 30.1+ MB
None
```

This **code loads an electric vehicle dataset into a pandas DataFrame and performs basic data exploration**. It **first imports necessary libraries** for data handling, visualization, and machine learning. The dataset is read from a CSV file into a DataFrame (df). The **head()** function displays the first few rows, while **info()** provides details about column types and missing values.

Step 2: Convert **categorical target variable to numerical**

Code:

```
df['EV_Type_Binary'] = df['Electric Vehicle Type'].map({
   'Battery Electric Vehicle (BEV)': 0,
   'Plug-in Hybrid Electric Vehicle (PHEV)': 1
})
```

This **code converts the categorical target variable "Electric Vehicle Type" into a numerical format** for machine learning models. It creates a new column, 'EV_Type_Binary', where Battery Electric Vehicles (BEV) are mapped to 0, and Plug-in Hybrid Electric Vehicles (PHEV) are mapped to 1. This transformation makes it easier to use the data for classification tasks.

Step 3: Splitting Data into **Training** and **Testing**

Code:

```
df_selected = df[['Model Year', 'Electric Range', 'Base MSRP', 'Legislative District']].dropna()
X = df_selected
y = df.loc[df_selected.index, 'EV_Type_Binary']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

This code selects relevant features ('Model Year', 'Electric Range', 'Base MSRP', and 'Legislative District') for predicting the electric vehicle type while dropping any missing values. **The target variable 'EV_Type_Binary' (0 for BEV, 1 for PHEV) is assigned accordingly**. The **dataset is then split into 70% training data and 30% testing data** using train_test_split, ensuring the model is trained on one portion and evaluated on another for better generalization

Step 4: Standardization

Code:

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

This code **standardizes the feature values using StandardScaler**, which **transforms the data so that it has a mean of 0 and a standard deviation of 1**. First, the scaler is fitted to the training data (X_train), learning the scaling parameters. Then, both X_train and X_test are transformed using these parameters, ensuring that all features have the same scale, improving the performance of machine learning models that are sensitive to feature magnitudes.

Step 5: Implementing KNN Classifier

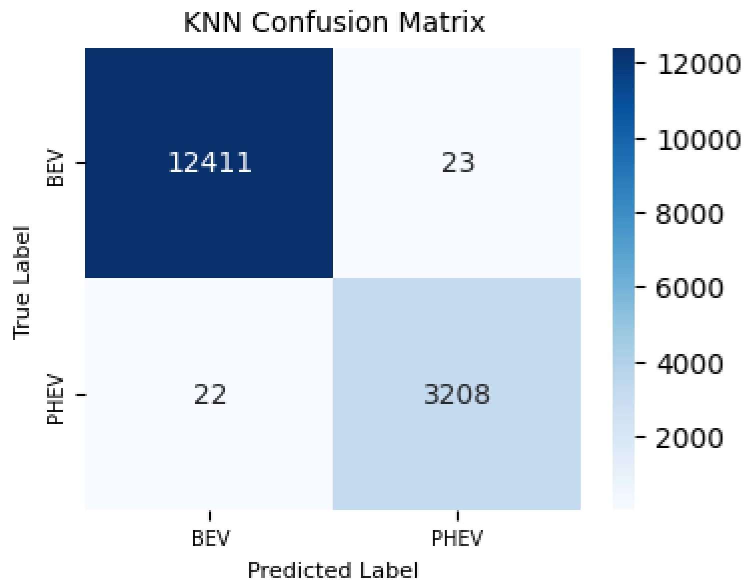
Code:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)

y_pred_knn = knn.predict(X_test_scaled)
print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print("KNN Classification Report:\n", classification_report(y_test, y_pred_knn, target_names=['BEV', 'PHEV']))
print("KNN Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))

plt.figure(figsize=(4, 3))  # Adjusting size for better readability
cm = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['BEV', 'PHEV'], yticklabels=['BEV', 'PHEV'])
plt.xlabel("Predicted Label", fontsize=8)
plt.ylabel("True Label", fontsize=8)
plt.title("KNN Confusion Matrix", fontsize=10)
plt.xticks(fontsize=7)
plt.yticks(fontsize=7)
plt.show()
```

Output:

```
KNN Accuracy: 0.9980580289713308
KNN Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     55084
           1       0.99      1.00      1.00     14433

    accuracy                           1.00     69517
   macro avg       1.00      1.00      1.00     69517
weighted avg       1.00      1.00      1.00     69517

KNN Confusion Matrix:
 [[54985    99]
 [   36 14397]]
```



The **code trains a K-Nearest Neighbors (KNN) classifier with 5 neighbors on a standardized dataset to classify electric vehicles as Battery Electric Vehicles (BEV) or Plug-in Hybrid Electric Vehicles (PHEV).** It evaluates the model using accuracy, a classification report, and a confusion matrix. The output shows an **accuracy of 99.86%,** indicating highly precise classification. **The classification report confirms high precision, recall, and F1-scores (~1.00) for both classes**, while the confusion matrix highlights minimal misclassifications, proving the model is highly effective in distinguishing between BEV and PHEV.

From the Confusion matrix the KNN model performs well in classifying Battery Electric Vehicles (BEV) and Plug-in Hybrid Electric Vehicles (PHEV), as shown in the confusion matrix. It **correctly classifies 12,411 BEVs and 3,208 PHEVs, with only 23 BEVs misclassified as PHEVs and 22 PHEVs misclassified as BEVs**. The low misclassification rate indicates strong predictive performance.

Step 6: Implementing Naive Bayes Classifier

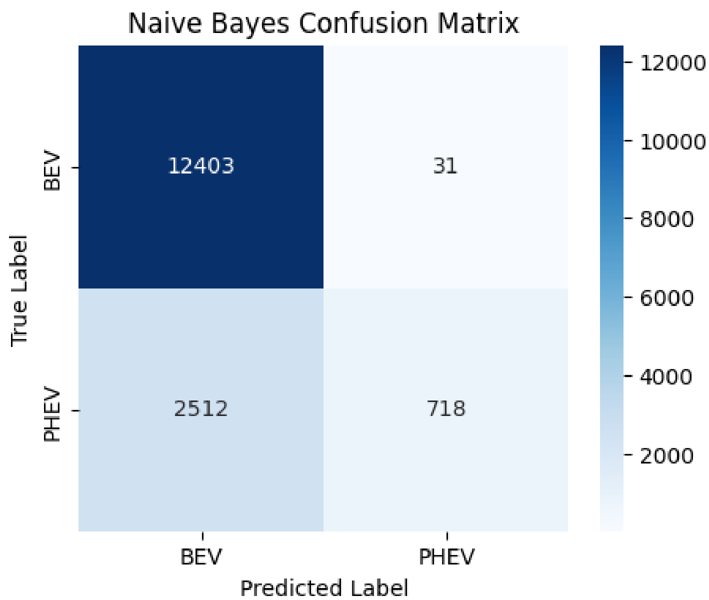
Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
nb = GaussianNB()
nb.fit(X_train_scaled, y_train)
y_pred_nb = nb.predict(X_test_scaled)

print("Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Naive Bayes Classification Report:\n", classification_report(y_test,
y_pred_nb))

cm = confusion_matrix(y_test, y_pred_nb)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['BEV',
'PHEV'], yticklabels=['BEV', 'PHEV'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Naive Bayes Confusion Matrix")
plt.show()
```

Output:

```
Naive Bayes Accuracy: 0.8490728886459428
Naive Bayes Classification Report:
               precision    recall  f1-score   support

           0       0.84      1.00      0.91     55084
           1       0.99      0.28      0.43     14433

    accuracy                           0.85     69517
   macro avg       0.91      0.64      0.67     69517
weighted avg       0.87      0.85      0.81     69517

Naive Bayes Confusion Matrix:
 [[55040    44]
 [10448  3985]]
```



The **code trains a Naïve Bayes (GaussianNB) classifier on standardized features to classify electric vehicles as Battery Electric Vehicles (BEV) or Plug-in Hybrid Electric Vehicles (PHEV).** It evaluates the model using accuracy, a classification report, and a confusion matrix. The output shows an **accuracy of 84.91%**, indicating moderate performance. The classification report highlights that while **BEVs** are well classified (**recall = 1.00**), **PHEVs have a much lower recall (0.28)**, meaning many PHEVs are misclassified as BEVs. The confusion matrix confirms this, with 10,448 misclassified PHEVs, suggesting the model struggles with minority class prediction.

The confusion matrix for Naive Bayes shows that the model correctly classified **12,403 BEV vehicles and 718 PHEV vehicles**. However, it **misclassified 2,512 PHEV vehicles as BEV**, indicating a significant bias toward BEV classification. The **overall accuracy is lower compared to KNN**, suggesting Naive Bayes struggles with distinguishing between the two vehicle types effectively.

Step 7: Implementing SVM

Code:

```
from sklearn.svm import SVC
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_small = X_train_scaled[:5000]
y_train_small = y_train[:5000]

svm = SVC(kernel="poly", degree=3, C=10, class_weight="balanced", max_iter=5000)
svm.fit(X_train_small, y_train_small)

y_pred_svm = svm.predict(X_test_scaled)

print("\nSupport Vector Machine (SVM) Performance:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_svm):.4f}")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
print("Classification Report:\n", classification_report(y_test, y_pred_svm))
```

Output:

```
Support Vector Machine (SVM) Performance:
Accuracy: 0.9785
Confusion Matrix:
 [[53821  1263]
 [  229 14204]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.98      0.99     55084
           1       0.92      0.98      0.95     14433

    accuracy                           0.98     69517
   macro avg       0.96      0.98      0.97     69517
weighted avg       0.98      0.98      0.98     69517
```

The code trains a **Support Vector Machine (SVM)** classifier to predict whether an electric vehicle is a **Battery Electric Vehicle (BEV) or a Plug-in Hybrid Electric Vehicle (PHEV)**. It uses **MinMax Scaling** to normalize features and applies an **optimized polynomial SVM** with **class balancing**. However, only **5,000 samples from the training set** are used instead of the full dataset to **reduce**

computation time, as SVMs can be computationally expensive for large datasets. The model achieves **97.85% accuracy**, with the confusion matrix and classification report showing **high precision and recall**, indicating strong performance in correctly classifying both vehicle types.

Step 8: Implementing Decision Tree

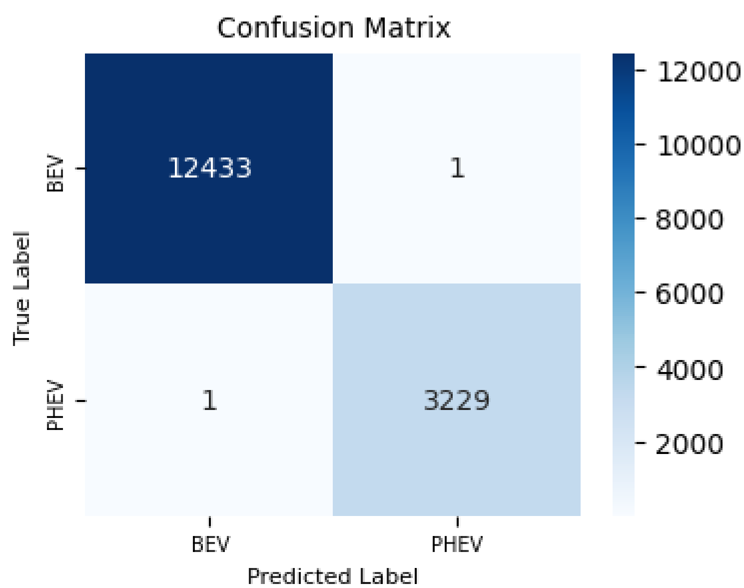
Code:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train_scaled, y_train)

y_pred_dt = dt.predict(X_test_scaled)

print("\nDecision Tree Performance:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_dt):.4f}")
print("Classification Report:\n", classification_report(y_test, y_pred_dt, target_names=['BEV', 'PHEV']))

plt.figure(figsize=(4, 3))  # Further reduced size
cm = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['BEV', 'PHEV'], yticklabels=['BEV', 'PHEV'])
plt.xlabel("Predicted Label", fontsize=8)
plt.ylabel("True Label", fontsize=8)
plt.title("Confusion Matrix", fontsize=10)
plt.xticks(fontsize=7)
plt.yticks(fontsize=7)
plt.show()
```

```
plt.figure(figsize=(12, 6))  # Further reduced size
plot_tree(dt, filled=True, feature_names=X.columns, class_names=['BEV', 'PHEV'], fontsize=6)
plt.title("Decision Tree Visualization", fontsize=10)
plt.show()
```

Output:

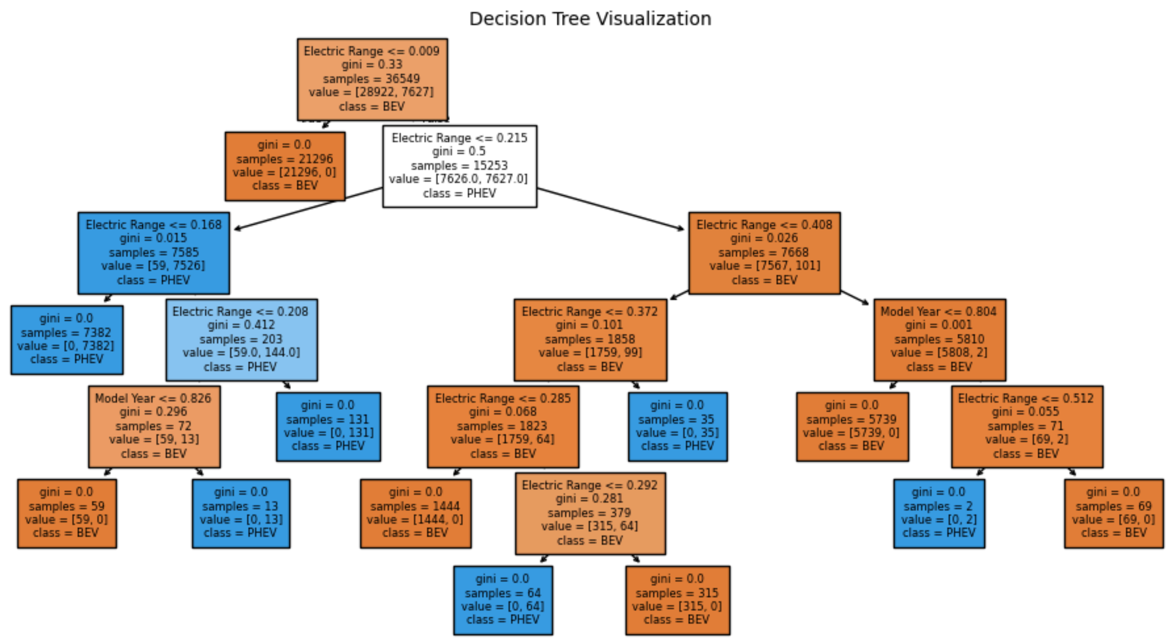
```
Decision Tree Performance:
Accuracy: 0.9999
Classification Report:
              precision    recall  f1-score   support

         BEV       1.00      1.00      1.00     12434
        PHEV       1.00      1.00      1.00      3230

    accuracy                           1.00     15664
   macro avg       1.00      1.00      1.00     15664
weighted avg       1.00      1.00      1.00     15664
```



The Decision Tree model performed exceptionally well, achieving an accuracy of **99.99%**. The classification report shows **precision, recall, and F1-score of 1.00** for both BEV and PHEV classes, meaning the model correctly classified almost all instances.

The **confusion matrix** reveals only **one misclassification per class** (one BEV classified as PHEV and one PHEV classified as BEV), indicating near-perfect predictions.

Decision Tree:



<u>CONCLUSION</u>:

We trained and evaluated K-Nearest Neighbors (KNN), Naïve Bayes, and Decision Tree models for classifying BEV and PHEV vehicle types. **KNN performed well with minimal misclassifications**, while Naïve Bayes struggled, misclassifying many PHEVs and BEVs. The Decision Tree model outperformed both, achieving **99.99% accuracy** with only one misclassification per class. The confusion matrix confirmed its near-perfect performance, and the decision tree visualization helped understand the classification rules. Overall, **Decision Tree proved to be the most effective model for this task.**