### ****Summary of Applications for Your Code:****

**For Classification:**

**Naive Bayes, KNN, SVM, Linear Regression, Decision Tree Classifier, Random Forest Classifier**

**For Clustering:**

**K-Means Clustering, Agglomerative Hierarchical Clustering**

**Classification**

**1.Random Forest Classifier (Accuracy is in the messages of mother’s whatsapp)**

**Model.py**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier # Using Random Forest for classification

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import pickle

# Load dataset

data = pd.read\_csv('cars.csv')

# Step 1: Convert the 'price' column to a categorical classification problem

# Define price ranges and labels

bins = [0, 18000, 22000, float('inf')]

labels = ['Low', 'Medium', 'High']

# Create a new column with categorized price and drop the original 'price' column

data['price\_category'] = pd.cut(data['price'], bins=bins, labels=labels)

data = data.drop('price', axis=1)

# Step 2: Preprocess categorical features with one-hot encoding

data = pd.get\_dummies(data, columns=['make', 'model', 'color'], drop\_first=True)

# Define features and target for classification

X = data.drop('price\_category', axis=1)

y = data['price\_category']

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 3: Train a classification model (e.g., RandomForestClassifier)

model = RandomForestClassifier(random\_state=42)

model.fit(X\_train, y\_train)

# Save the trained model

with open('car\_price\_classification\_model.pkl', 'wb') as f:

pickle.dump(model, f)

# Step 4: Evaluate the model using classification metrics

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

# Display the confusion matrix and classification report

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

# Output the results

print(f"Model trained with accuracy: {accuracy}")

print("Confusion Matrix:")

print(conf\_matrix)

print("Classification Report:")

print(class\_report)

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Car Price Prediction</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f9;

margin: 0;

padding: 20px;

text-align: center;

}

h1 {

color: #333;

}

form {

display: inline-block;

background-color: #fff;

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

label {

display: block;

margin-bottom: 8px;

color: #555;

}

input[type="number"],

input[type="text"] {

width: 100%;

padding: 8px;

margin-bottom: 10px;

border: 1px solid #ccc;

border-radius: 4px;

}

input[type="submit"] {

background-color: #4CAF50;

color: white;

padding: 10px 20px;

border: none;

border-radius: 4px;

cursor: pointer;

}

input[type="submit"]:hover {

background-color: #45a049;

}

h2 {

color: #333;

margin-top: 20px;

}

</style>

</head>

<body>

<h1>Car Price Prediction</h1>

<form method="POST">

<label for="year">Year:</label>

<input type="number" name="year" required><br>

<label for="mileage">Mileage:</label>

<input type="number" name="mileage" required><br>

<label for="make">Make:</label>

<input type="text" name="make" required><br>

<label for="model">Model:</label>

<input type="text" name="model" required><br>

<label for="color">Color:</label>

<input type="text" name="color" required><br>

<input type="submit" value="Predict">

</form>

{% if prediction is not none %}

<h2>Predicted Price: ${{ prediction }}</h2>

{% endif %}

</body>

</html>

**app.py**

from flask import Flask, render\_template, request

import pandas as pd

import pickle

app = Flask(\_\_name\_\_)

# Load the trained classification model

with open('car\_price\_classification\_model.pkl', 'rb') as f:

model = pickle.load(f)

@app.route('/', methods=['GET', 'POST'])

def index():

prediction = None

if request.method == 'POST':

# Get user inputs

year = int(request.form['year'])

mileage = int(request.form['mileage'])

make = request.form['make']

model\_name = request.form['model']

color = request.form['color']

# Convert input to DataFrame

input\_data = pd.DataFrame({

'year': [year],

'mileage': [mileage],

'make': [make],

'model': [model\_name],

'color': [color]

})

# Preprocess input

input\_data = pd.get\_dummies(input\_data, columns=['make', 'model', 'color'], drop\_first=True)

# Ensure all input columns match the model input

input\_data = input\_data.reindex(columns=model.feature\_names\_in\_, fill\_value=0)

# Make prediction (this will predict the category, e.g., 'Low', 'Medium', 'High')

prediction = model.predict(input\_data)[0]

return render\_template('index.html', prediction=prediction)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**2.Decision Tree Classifier**

**model.py**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier # Import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import pickle

# Load dataset

data = pd.read\_csv('cars.csv')

# Step 1: Convert the 'price' column to a categorical classification problem

# Define price ranges and labels

bins = [0, 18000, 22000, float('inf')]

labels = ['Low', 'Medium', 'High']

# Create a new column with categorized price and drop the original 'price' column

data['price\_category'] = pd.cut(data['price'], bins=bins, labels=labels)

data = data.drop('price', axis=1)

# Step 2: Preprocess categorical features with one-hot encoding

data = pd.get\_dummies(data, columns=['make', 'model', 'color'], drop\_first=True)

# Define features and target for classification

X = data.drop('price\_category', axis=1)

y = data['price\_category']

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 3: Train a Decision Tree Classifier model

model = DecisionTreeClassifier(random\_state=42)

model.fit(X\_train, y\_train)

# Save the trained model

with open('car\_price\_decision\_tree\_model.pkl', 'wb') as f:

pickle.dump(model, f)

# Step 4: Evaluate the model using classification metrics

y\_pred = model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

# Display the confusion matrix and classification report

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

# Output the results

print(f"Model trained with accuracy: {accuracy}")

print("Confusion Matrix:")

print(conf\_matrix)

print("Classification Report:")

print(class\_report)

**app.py**

from flask import Flask, render\_template, request

import pandas as pd

import pickle

app = Flask(\_\_name\_\_)

# Load the trained Decision Tree model

with open('car\_price\_decision\_tree\_model.pkl', 'rb') as f:

model = pickle.load(f)

@app.route('/', methods=['GET', 'POST'])

def index():

prediction = None

if request.method == 'POST':

# Get user inputs

year = int(request.form['year'])

mileage = int(request.form['mileage'])

make = request.form['make']

model\_name = request.form['model']

color = request.form['color']

# Convert input to DataFrame

input\_data = pd.DataFrame({

'year': [year],

'mileage': [mileage],

'make': [make],

'model': [model\_name],

'color': [color]

})

# Preprocess input (One-hot encoding for categorical features)

input\_data = pd.get\_dummies(input\_data, columns=['make', 'model', 'color'], drop\_first=True)

# Ensure all input columns match the model input (fill missing columns with 0)

input\_data = input\_data.reindex(columns=model.feature\_names\_in\_, fill\_value=0)

# Make prediction

prediction = model.predict(input\_data)[0]

return render\_template('index.html', prediction=prediction)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**3.KNN Algorithm Applied on the Application**

**Model.py**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier # K-NN for classification

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import pickle

# Load dataset

data = pd.read\_csv('cars.csv')

# Step 1: Convert the 'price' column to a categorical classification problem

# Define price ranges and labels

bins = [0, 18000, 22000, float('inf')]

labels = ['Low', 'Medium', 'High']

# Create a new column with categorized price and drop the original 'price' column

data['price\_category'] = pd.cut(data['price'], bins=bins, labels=labels)

data = data.drop('price', axis=1)

# Step 2: Preprocess categorical features with one-hot encoding

data = pd.get\_dummies(data, columns=['make', 'model', 'color'], drop\_first=True)

# Define features and target for classification

X = data.drop('price\_category', axis=1)

y = data['price\_category']

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 3: Train a classification model (K-NN classifier)

knn\_model = KNeighborsClassifier(n\_neighbors=5) # You can tune n\_neighbors

knn\_model.fit(X\_train, y\_train)

# Save the trained model

with open('car\_price\_classification\_knn\_model.pkl', 'wb') as f:

pickle.dump(knn\_model, f)

# Step 4: Evaluate the model using classification metrics

# Make predictions on the test set

y\_pred = knn\_model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

# Display the confusion matrix and classification report

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

# Output the results

print(f"Model trained with accuracy: {accuracy}")

print("Confusion Matrix:")

print(conf\_matrix)

print("Classification Report:")

print(class\_report**)**

**app.py**

from flask import Flask, render\_template, request

import pandas as pd

import pickle

app = Flask(\_\_name\_\_)

# Load the trained K-NN classification model

with open('car\_price\_classification\_knn\_model.pkl', 'rb') as f:

model = pickle.load(f)

@app.route('/', methods=['GET', 'POST'])

def index():

prediction = None

if request.method == 'POST':

# Get user inputs

year = int(request.form['year'])

mileage = int(request.form['mileage'])

make = request.form['make']

model\_name = request.form['model']

color = request.form['color']

# Convert input to DataFrame

input\_data = pd.DataFrame({

'year': [year],

'mileage': [mileage],

'make': [make],

'model': [model\_name],

'color': [color]

})

# Preprocess input

input\_data = pd.get\_dummies(input\_data, columns=['make', 'model', 'color'], drop\_first=True)

# Ensure all input columns match the model input

input\_data = input\_data.reindex(columns=model.feature\_names\_in\_, fill\_value=0)

# Make prediction (this will predict the category, e.g., 'Low', 'Medium', 'High')

prediction = model.predict(input\_data)[0]

return render\_template('index.html', prediction=prediction)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**4.Naive Bayes Algorithm**

**model.py**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB # Naive Bayes classifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import pickle

# Load dataset

data = pd.read\_csv('cars.csv')

# Step 1: Convert the 'price' column to a categorical classification problem

# Define price ranges and labels

bins = [0, 18000, 22000, float('inf')]

labels = ['Low', 'Medium', 'High']

# Create a new column with categorized price and drop the original 'price' column

data['price\_category'] = pd.cut(data['price'], bins=bins, labels=labels)

data = data.drop('price', axis=1)

# Step 2: Preprocess categorical features with one-hot encoding

data = pd.get\_dummies(data, columns=['make', 'model', 'color'], drop\_first=True)

# Define features and target for classification

X = data.drop('price\_category', axis=1)

y = data['price\_category']

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 3: Train a classification model (Naive Bayes classifier)

nb\_model = GaussianNB()

nb\_model.fit(X\_train, y\_train)

# Save the trained model

with open('car\_price\_classification\_nb\_model.pkl', 'wb') as f:

pickle.dump(nb\_model, f)

# Step 4: Evaluate the model using classification metrics

# Make predictions on the test set

y\_pred = nb\_model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

# Display the confusion matrix and classification report

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

# Output the results

print(f"Model trained with accuracy: {accuracy}")

print("Confusion Matrix:")

print(conf\_matrix)

print("Classification Report:")

print(class\_report)

**app.py**

from flask import Flask, render\_template, request

import pandas as pd

import pickle

app = Flask(\_\_name\_\_)

# Load the trained Naive Bayes classification model

with open('car\_price\_classification\_nb\_model.pkl', 'rb') as f:

model = pickle.load(f)

@app.route('/', methods=['GET', 'POST'])

def index():

prediction = None

if request.method == 'POST':

# Get user inputs

year = int(request.form['year'])

mileage = int(request.form['mileage'])

make = request.form['make']

model\_name = request.form['model']

color = request.form['color']

# Convert input to DataFrame

input\_data = pd.DataFrame({

'year': [year],

'mileage': [mileage],

'make': [make],

'model': [model\_name],

'color': [color]

})

# Preprocess input

input\_data = pd.get\_dummies(input\_data, columns=['make', 'model', 'color'], drop\_first=True)

# Ensure all input columns match the model input

input\_data = input\_data.reindex(columns=model.feature\_names\_in\_, fill\_value=0)

# Make prediction (this will predict the category, e.g., 'Low', 'Medium', 'High')

prediction = model.predict(input\_data)[0]

return render\_template('index.html', prediction=prediction)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**5.SVM Algorithm**

**model.py**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC # Support Vector Classification (SVM)

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import pickle

# Load dataset

data = pd.read\_csv('cars.csv')

# Step 1: Convert the 'price' column to a categorical classification problem

# Define price ranges and labels

bins = [0, 18000, 22000, float('inf')]

labels = ['Low', 'Medium', 'High']

# Create a new column with categorized price and drop the original 'price' column

data['price\_category'] = pd.cut(data['price'], bins=bins, labels=labels)

data = data.drop('price', axis=1)

# Step 2: Preprocess categorical features with one-hot encoding

data = pd.get\_dummies(data, columns=['make', 'model', 'color'], drop\_first=True)

# Define features and target for classification

X = data.drop('price\_category', axis=1)

y = data['price\_category']

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 3: Train a classification model (Support Vector Machine)

svm\_model = SVC(kernel='linear', random\_state=42) # Linear kernel (you can try 'rbf' or others)

svm\_model.fit(X\_train, y\_train)

# Save the trained model

with open('car\_price\_classification\_svm\_model.pkl', 'wb') as f:

pickle.dump(svm\_model, f)

# Step 4: Evaluate the model using classification metrics

# Make predictions on the test set

y\_pred = svm\_model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

# Display the confusion matrix and classification report

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

# Output the results

print(f"Model trained with accuracy: {accuracy}")

print("Confusion Matrix:")

print(conf\_matrix)

print("Classification Report:")

print(class\_report)

**app.py**

from flask import Flask, render\_template, request

import pandas as pd

import pickle

app = Flask(\_\_name\_\_)

# Load the trained SVM classification model

with open('car\_price\_classification\_svm\_model.pkl', 'rb') as f:

model = pickle.load(f)

@app.route('/', methods=['GET', 'POST'])

def index():

prediction = None

if request.method == 'POST':

# Get user inputs

year = int(request.form['year'])

mileage = int(request.form['mileage'])

make = request.form['make']

model\_name = request.form['model']

color = request.form['color']

# Convert input to DataFrame

input\_data = pd.DataFrame({

'year': [year],

'mileage': [mileage],

'make': [make],

'model': [model\_name],

'color': [color]

})

# Preprocess input

input\_data = pd.get\_dummies(input\_data, columns=['make', 'model', 'color'], drop\_first=True)

# Ensure all input columns match the model input

input\_data = input\_data.reindex(columns=model.feature\_names\_in\_, fill\_value=0)

# Make prediction (this will predict the category, e.g., 'Low', 'Medium', 'High')

prediction = model.predict(input\_data)[0]

return render\_template('index.html', prediction=prediction)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**6.Linear Regression(We don’t have the metrics like accuracy instead we have R^2,MSE)**

**model.py**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression # Linear Regression for regression tasks

from sklearn.metrics import mean\_squared\_error, r2\_score

import pickle

# Load dataset

data = pd.read\_csv('cars.csv')

# Step 1: Preprocess the data

# We will not categorize the price in Linear Regression since we are predicting the exact price

# Step 2: Preprocess categorical features with one-hot encoding

data = pd.get\_dummies(data, columns=['make', 'model', 'color'], drop\_first=True)

# Define features and target for regression

X = data.drop('price', axis=1) # Using 'price' as the target variable

y = data['price'] # The target is the price itself (continuous value)

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 3: Train a Linear Regression model

lr\_model = LinearRegression()

lr\_model.fit(X\_train, y\_train)

# Save the trained model

with open('car\_price\_regression\_model.pkl', 'wb') as f:

pickle.dump(lr\_model, f)

# Step 4: Evaluate the model using regression metrics

# Make predictions on the test set

y\_pred = lr\_model.predict(X\_test)

# Calculate Mean Squared Error and R-squared

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

# Output the results

print(f"Model trained with Mean Squared Error (MSE): {mse}")

print(f"R-squared: {r2}")

**app.py**

from flask import Flask, render\_template, request

import pandas as pd

import pickle

app = Flask(\_\_name\_\_)

# Load the trained Linear Regression model

with open('car\_price\_regression\_model.pkl', 'rb') as f:

model = pickle.load(f)

@app.route('/', methods=['GET', 'POST'])

def index():

prediction = None

if request.method == 'POST':

# Get user inputs

year = int(request.form['year'])

mileage = int(request.form['mileage'])

make = request.form['make']

model\_name = request.form['model']

color = request.form['color']

# Convert input to DataFrame

input\_data = pd.DataFrame({

'year': [year],

'mileage': [mileage],

'make': [make],

'model': [model\_name],

'color': [color]

})

# Preprocess input (one-hot encoding for categorical variables)

input\_data = pd.get\_dummies(input\_data, columns=['make', 'model', 'color'], drop\_first=True)

# Ensure all input columns match the model input

input\_data = input\_data.reindex(columns=model.feature\_names\_in\_, fill\_value=0)

# Make prediction (this will predict the exact price)

prediction = model.predict(input\_data)[0]

return render\_template('index.html', prediction=prediction)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Clustering**

**7.**

1. **Means**

**model.py**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.cluster import KMeans # KMeans for clustering

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette\_score # Import silhouette\_score

import pickle

# Load dataset

data = pd.read\_csv('cars.csv')

# Step 1: Preprocess the data (no need to categorize price since we're clustering)

# Preprocess categorical features with one-hot encoding

data = pd.get\_dummies(data, columns=['make', 'model', 'color'], drop\_first=True)

# Step 2: Define features for clustering

X = data.drop('price', axis=1) # We are not using 'price' for clustering

# Step 3: Standardize the features (important for K-Means)

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Step 4: Apply K-Means Clustering

kmeans = KMeans(n\_clusters=3, random\_state=42) # We'll cluster into 3 groups (this can be adjusted)

kmeans.fit(X\_scaled)

# Save the trained KMeans model and scaler

with open('car\_price\_clustering\_model.pkl', 'wb') as f:

pickle.dump(kmeans, f)

with open('scaler.pkl', 'wb') as f:

pickle.dump(scaler, f)

# Step 5: Assign clusters to the data

data['cluster'] = kmeans.labels\_

# Displaying some results (first few rows with cluster labels)

print(data.head())

# Step 6: Evaluate the clustering with the Silhouette Score

silhouette\_avg = silhouette\_score(X\_scaled, kmeans.labels\_)

print(f"Silhouette Score: {silhouette\_avg}")

# Optional: Check the cluster centers

print(f"Cluster Centers:\n{kmeans.cluster\_centers\_}")

**app.py**

from flask import Flask, render\_template, request

import pandas as pd

import pickle

from sklearn.preprocessing import StandardScaler

app = Flask(\_\_name\_\_)

# Load the trained K-Means model and scaler

with open('car\_price\_clustering\_model.pkl', 'rb') as f:

kmeans\_model = pickle.load(f)

with open('scaler.pkl', 'rb') as f:

scaler = pickle.load(f)

@app.route('/', methods=['GET', 'POST'])

def index():

prediction = None

cluster\_info = None

if request.method == 'POST':

# Get user inputs

year = int(request.form['year'])

mileage = int(request.form['mileage'])

make = request.form['make']

model\_name = request.form['model']

color = request.form['color']

# Convert input to DataFrame (ensure it has the same structure as the model input)

input\_data = pd.DataFrame({

'year': [year],

'mileage': [mileage],

'make': [make],

'model': [model\_name],

'color': [color]

})

# Preprocess input data

input\_data = pd.get\_dummies(input\_data, columns=['make', 'model', 'color'], drop\_first=True)

# Ensure all input columns match the model input (fill missing columns with 0)

input\_data = input\_data.reindex(columns=kmeans\_model.cluster\_centers\_.columns, fill\_value=0)

# Standardize the input data using the same scaler as used in model training

input\_data\_scaled = scaler.transform(input\_data)

# Make prediction (cluster assignment)

cluster\_prediction = kmeans\_model.predict(input\_data\_scaled)

prediction = f"Cluster Assigned: {cluster\_prediction[0]}"

# Get some information about the cluster (e.g., closest center)

cluster\_info = f"Cluster Center: {kmeans\_model.cluster\_centers\_[cluster\_prediction[0]]}"

return render\_template('index.html', prediction=prediction, cluster\_info=cluster\_info)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**8.Agglomerative Hierarchical Clustering**

**model.py**

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import AgglomerativeClustering

from sklearn.metrics import silhouette\_score # Import Silhouette Score

import pickle

# Load dataset

data = pd.read\_csv('cars.csv')

# Step 1: Preprocess the data

# Preprocess categorical features with one-hot encoding

data = pd.get\_dummies(data, columns=['make', 'model', 'color'], drop\_first=True)

# Step 2: Define features for clustering

X = data.drop('price', axis=1) # We are not using 'price' for clustering

# Step 3: Standardize the features (important for clustering algorithms)

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Step 4: Apply Agglomerative Clustering

agg\_clustering = AgglomerativeClustering(n\_clusters=3) # Number of clusters

agg\_clustering.fit(X\_scaled)

# Save the trained Agglomerative Clustering model and scaler

with open('car\_price\_hierarchical\_clustering\_model.pkl', 'wb') as f:

pickle.dump(agg\_clustering, f)

with open('scaler.pkl', 'wb') as f:

pickle.dump(scaler, f)

# Step 5: Assign clusters to the data

data['cluster'] = agg\_clustering.labels\_

# Calculate Silhouette Score to evaluate clustering

sil\_score = silhouette\_score(X\_scaled, agg\_clustering.labels\_)

print(f"Silhouette Score: {sil\_score}")

# Optionally, you can print out the first few rows with their cluster assignments

print(data.head())

**app.py**

from flask import Flask, render\_template, request

import pandas as pd

import pickle

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette\_score # Import Silhouette Score

app = Flask(\_\_name\_\_)

# Load the trained Agglomerative Clustering model and scaler

with open('car\_price\_hierarchical\_clustering\_model.pkl', 'rb') as f:

agglomerative\_model = pickle.load(f)

with open('scaler.pkl', 'rb') as f:

scaler = pickle.load(f)

@app.route('/', methods=['GET', 'POST'])

def index():

prediction = None

cluster\_info = None

silhouette\_score\_value = None

if request.method == 'POST':

# Get user inputs

year = int(request.form['year'])

mileage = int(request.form['mileage'])

make = request.form['make']

model\_name = request.form['model']

color = request.form['color']

# Convert input to DataFrame (ensure it has the same structure as the model input)

input\_data = pd.DataFrame({

'year': [year],

'mileage': [mileage],

'make': [make],

'model': [model\_name],

'color': [color]

})

# Preprocess input data

input\_data = pd.get\_dummies(input\_data, columns=['make', 'model', 'color'], drop\_first=True)

# Ensure all input columns match the model input (fill missing columns with 0)

input\_data = input\_data.reindex(columns=agglomerative\_model.components\_, fill\_value=0)

# Standardize the input data using the same scaler as used in model training

input\_data\_scaled = scaler.transform(input\_data)

# Make prediction (cluster assignment)

cluster\_prediction = agglomerative\_model.fit\_predict(input\_data\_scaled)

prediction = f"Cluster Assigned: {cluster\_prediction[0]}"

# Get some information about the cluster (e.g., closest center)

cluster\_info = f"Cluster Label: {cluster\_prediction[0]}"

# Calculate Silhouette Score (using the existing dataset)

# Calculate silhouette score for the whole dataset (since user input affects only the prediction)

silhouette\_score\_value = silhouette\_score(input\_data\_scaled, cluster\_prediction)

silhouette\_score\_value = f"Silhouette Score for the cluster: {silhouette\_score\_value:.4f}"

return render\_template('index.html', prediction=prediction, cluster\_info=cluster\_info, silhouette\_score\_value=silhouette\_score\_value)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Comparison of All Algorithms Accuracy in Classification( By using Matplotlib)**

1. Matplotlib

Description: Matplotlib is a versatile plotting library that can be used to create various types of plots, including bar charts, line graphs, and histograms.

How to use it for accuracy:

You can plot a simple bar chart to compare the accuracy of different models or the accuracy of a model over multiple runs (e.g., training vs. testing).

python

Copy code

import matplotlib.pyplot as plt

# Example accuracy values

accuracies = [0.85, 0.88, 0.90] # Accuracy of different models or tests

labels = ['Model 1', 'Model 2', 'Model 3']

# Plotting

plt.bar(labels, accuracies, color=['blue', 'green', 'red'])

plt.ylabel('Accuracy')

plt.title('Model Accuracy Comparison')

plt.show()

**Comparison of all the Algorithms Silhouette Score in Clustering( By using Seaborn)**

2. Seaborn

Description: Seaborn is built on top of Matplotlib and provides a high-level interface for drawing attractive statistical graphics.

How to use it for accuracy:

You can create bar plots, heatmaps, or even distribution plots for accuracy and confusion matrices.

python

Copy code

import seaborn as sns

import matplotlib.pyplot as plt

# Example accuracy values

accuracies = [0.85, 0.88, 0.90]

labels = ['Model 1', 'Model 2', 'Model 3']

# Plotting with Seaborn

sns.barplot(x=labels, y=accuracies, palette="Blues\_d")

plt.ylabel('Accuracy')

plt.title('Model Accuracy Comparison')

plt.show()

[**https://chatgpt.com/c/6732f5c5-6674-8001-a665-b3526296d62f**](https://chatgpt.com/c/6732f5c5-6674-8001-a665-b3526296d62f)