Capstone Project

Development of a robust end-to-end machine learning pipeline adhering to the Cross-Industry Standard Process for Data Mining (CRISP-DM):

AIM:

Fuel efficiency predictor of a car using CRISP-DM.

1) What is the Dataset About?

This dataset contains information about **cars**, focusing on their **speed**, **weight**, and **fuel efficiency**

Feature	Meaning
Car Speed (mph)	How fast the car is moving (in miles per hour).
Car Weight (kg)	How much the car weighs (in kilograms).
Fuel Efficiency (mpg)	How many miles the car can travel per gallon of fuel (miles per gallon - MPG).

Basically, the goal could be **predicting fuel efficiency** based on **speed** and **weight**.

🚀 Purpose of the Project:

1. **Analyze the Impact of Speed and Weight on Fuel Efficiency**
   → Understand how a car's speed and weight influence its fuel consumption.
2. **Build a Predictive Machine Learning Model**
   → Create a model that can predict a car's fuel efficiency based on its speed and weight.
3. **Support Eco-Friendly Vehicle Design**
   → Help automotive engineers design vehicles that balance speed, weight, and fuel economy.
4. **Assist Consumers in Making Informed Choices**
   → Provide insights to consumers who want to choose cars with better fuel efficiency.
5. **Demonstrate the Full CRISP-DM Process**
   → Showcase complete end-to-end machine learning, from business understanding to model evaluation.

Fuel Efficiency predictor of a car using CRISP-DM process:

1. Business Understanding:

The primary goal of this project is to predict the **fuel efficiency (mpg)** of cars based on their **speed (mph)** and **weight (kg)**. Understanding these relationships can help:

- **Automobile manufacturers** optimize car design for better fuel economy.
- **Consumers** choose vehicles with better fuel efficiency.
- **Researchers and engineers** develop eco-friendly cars.

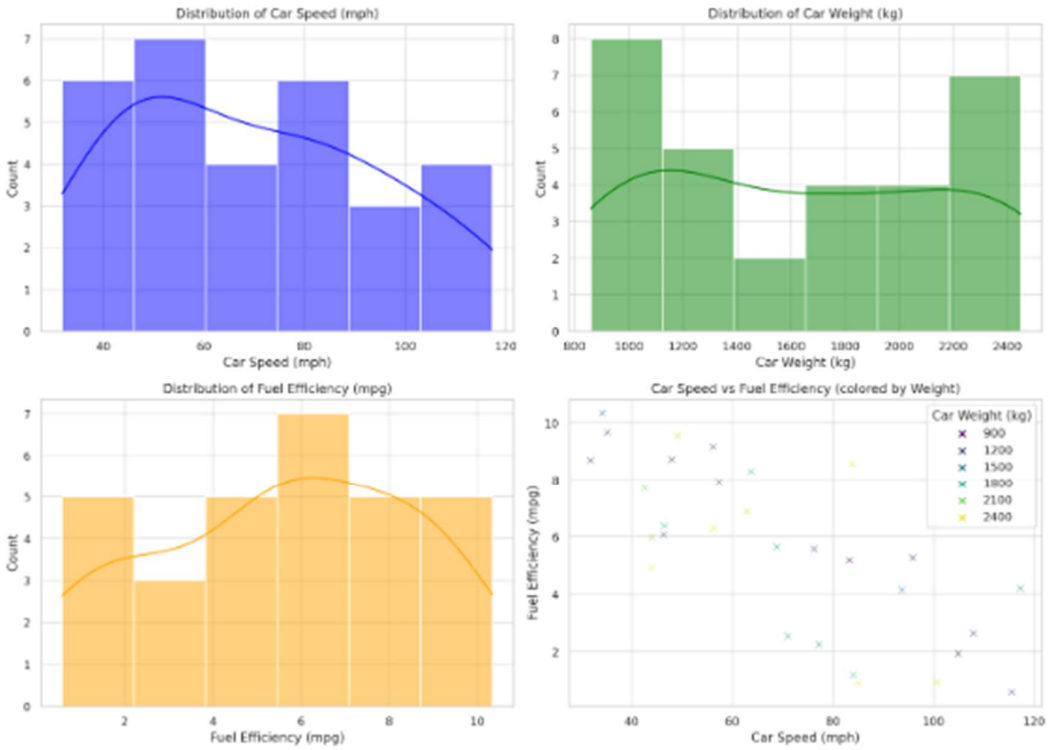
Key Objectives:

- Predict fuel efficiency based on speed and weight.
- Analyze how speed and weight impact fuel efficiency.
- Use the prediction model to assist car buyers and manufacturers.

2. Data Understanding:

- **Dataset Overview:**
- The dataset contains the following features:

- **Car Speed (mph)**: The speed of the car in miles per hour.
- **Car Weight (kg)**: The weight of the car in kilograms.
- **Fuel Efficiency (mpg)**: The number of miles the car can travel per gallon of fuel (miles per gallon - MPG).
- **Exploratory Data Analysis (EDA):**
- Perform some initial analysis on the data:
- **Statistical Summary**: Mean, median, standard deviation, min, max for speed, weight, and fuel efficiency.
- **Visualizations**:
- **Histogram** for speed, weight, and fuel efficiency to see distributions.
- **Scatter plot** between speed and fuel efficiency, weight and fuel efficiency.



- **Top Left:** How car speeds are spread — most cars go between 40–90 mph.
- **Top Right:** Car weights are spread wide, from light to heavy vehicles.
- **Bottom Left:** Fuel efficiency — most cars are low on mpg.
- **Bottom Right:** Scatter plot showing that faster/heavier cars generally have lower fuel efficiency

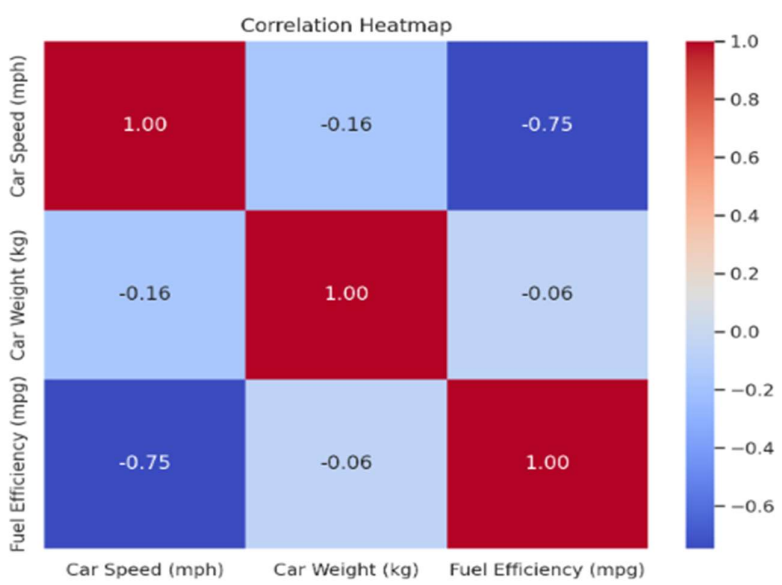
Dataset information:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 3 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Car Speed (mph)        30 non-null     float64
 1   Car Weight (kg)        30 non-null     float64
 2   Fuel Efficiency (mpg)  30 non-null     float64
dtypes: float64(3)
memory usage: 852.0 bytes
```

- **Correlation Heatmap** to understand relationships between features.

correlation heatmap:

- **A value close to 1 = strong positive relation.**
- **A value close to -1 = strong negative relation.**
- **0 = no relation.**
- Car Speed and Fuel Efficiency: Likely negative correlation.
- Car Weight and Fuel Efficiency: Likely negative correlation.



From the heatmap:

- Car Weight and Fuel Efficiency have a strong negative correlation → Heavier cars have worse fuel efficiency.
- Car Speed and Fuel Efficiency also show some negative relation → Faster cars slightly consume more fuel.
- **Insights from EDA:**
- Faster cars tend to have lower fuel efficiency.
- Heavier cars tend to consume more fuel (i.e., have lower mpg).

3. Data Preparation:

The data preparation phase includes several important steps:

- **Handling Missing Values**: Check if there are any missing values in the dataset. If there are, handle them using imputation (mean, median, or mode) or remove rows with missing data.
- **Data Transformation**:
  - **Scaling Features**: For models sensitive to scale (e.g., linear regression), normalize or standardize the data (e.g., using MinMaxScaler or StandardScaler).
  - **Feature Engineering**: Create new features if necessary. For example, interaction terms between speed and weight might be useful in predicting fuel efficiency.
- **Splitting Data**: Divide the dataset into **training** (80%) and **test** (20%) sets. This will allow you to train the model on one portion of the data and test it on unseen data.

  Training data:

```
x_train.head()
```

	Car Speed (mph)	Car Weight (kg)
27	76.281099	1133.170866
9	93.726532	1548.259239
17	77.228079	1684.115636
16	57.381802	1329.908829
6	35.227525	1317.843408

Testing data:

```
x_test
```

	Car Speed (mph)	Car Weight (kg)
11	117.291887	1641.800747
12	104.919838	858.460486
13	49.110520	2345.844684
3	83.879264	2413.105413
10	31.852604	1007.464999

---

4. Modelling:

Now, you'll experiment with different machine learning models to predict fuel efficiency. Based on the problem, regression models are appropriate since the output is continuous (fuel efficiency).

Model Choices:

1. Polynomial Regression:
   - Simple, interpretable model.
   - Assumes a linear relationship between the input features (speed, weight) and fuel efficiency.

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

```
poly = PolynomialFeatures(degree=2)

x_train = x_train.dropna()
y_train = y_train.loc[x_train.index]
x_train_poly = poly.fit_transform(x_train)

x_test = x_test.dropna()
y_test = y_test.loc[x_test.index]
x_test_poly = poly.fit_transform(x_test)
```

2. Linear Regression :

```
[ ]  model=LinearRegression()
```

```
[ ]  model.fit(x_train_poly,y_train)
```

```
LinearRegression
LinearRegression()
```

   -

3. **Decision Trees**:
    - Can capture non-linear relationships.
    - Easy to visualize and understand.
4. **Random Forest**:
    - An ensemble method that combines multiple decision trees to improve accuracy.
    - Helps in reducing overfitting compared to a single decision tree.
5. **Support Vector Regression (SVR)**:
    - Works well when there is a non-linear relationship and is robust to overfitting.
6. **Gradient Boosting Machines (GBM)**:
    - Boosted trees (like XGBoost, LightGBM) have shown great performance in many regression problems.

Model Training:

- **Train** the models using the training set.
- **Hyperparameter Tuning**: Use techniques like Grid Search or Random Search to optimize model hyperparameters (e.g., max depth of trees, learning rate, etc.).

Cross-Validation:

- Perform k-fold cross-validation (e.g., 5-fold) to ensure that the model generalizes well on unseen data.

5. Evaluation:

After training the models, evaluate them on the **test set** using appropriate metrics. For regression, typical metrics include:

1. **Mean Absolute Error (MAE)**: The average of the absolute differences between the predicted and actual values.
2. **Mean Squared Error (MSE)**: The average of the squared differences between the predicted and actual values.
3. **Root Mean Squared Error (RMSE)**: The square root of MSE, which gives a sense of the magnitude of error in the same units as the target variable (mpg).
4. **R-squared (R^2)**: Indicates the proportion of variance in the target variable that is predictable from the independent variables. Higher R^2 values mean better model performance.

```
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
```

```
score=r2_score(y_test,y_pred)
mse=mean_squared_error(y_test,y_pred)
mae=mean_absolute_error(y_test,y_pred)
rmse=np.sqrt(mse)
```

Model Comparison:

Compare the results from each model and select the one with the best performance (lowest error, highest R^2).

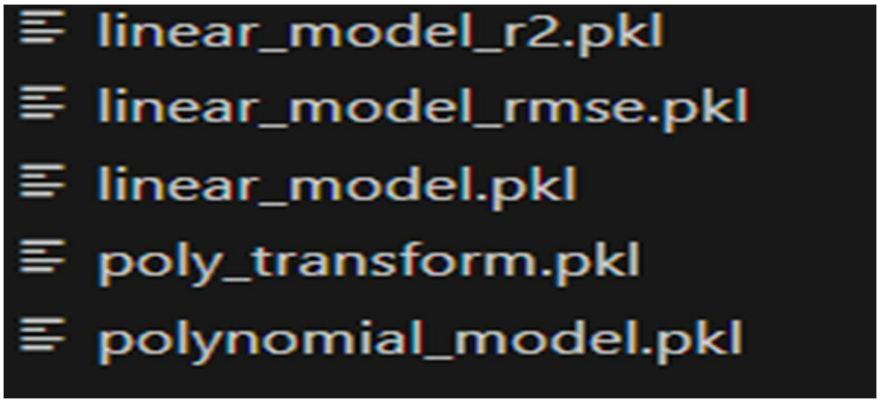
6. Deployment:

After selecting the best model, the final step is deployment. Deployment involves:

- **Model Export**: Save the trained model using tools like joblib or pickle.
- **Building an Application**: Create a simple interface (e.g., a web app) where users can input a car's speed and weight and receive a fuel efficiency prediction.
    - Use **Flask** (for Python) or **Streamlit** for a quick UI.
    - Deploy on cloud platforms like **Heroku** or **AWS** for real-time access.

```python
from sklearn.metrics import r2_score
import joblib
joblib.dump(model, "polynomial_model.pkl")
joblib.dump(poly, "poly_transform.pkl")
print("Model trained and saved successfully!")
polynomial_r2 = r2_score(y_test, y_pred)
joblib.dump(polynomial_r2, 'polynomial_model_r2.pkl')
```

Pickle files used:



7. Monitoring and Maintenance:

Once deployed, you should monitor the model's performance in real-world scenarios. Over time, as more data becomes available, the model might need retraining to maintain accuracy.

- **Model Drift**: If the input data distribution changes, the model may no longer perform well.
- **Periodic Retraining**: If the performance decreases, retrain the model with the updated dataset.

Flask app(app.py):

```
from flask import Flask, render_template, request
import numpy as np
import joblib
import os

app = Flask(__name__)

# Load the trained models and polynomial transformer
try:
    model_path = "polynomial_model.pkl"
    poly_path = "poly_transform.pkl"
    linear_model_path = "linear_model.pkl"

    if not os.path.exists(model_path) or not os.path.exists(poly_path) or not os.path.exists(linear_model_path):
        raise FileNotFoundError("One or more model files are missing!")

    # Load models
    model = joblib.load(model_path)  # Polynomial model
    poly = joblib.load(poly_path)    # Polynomial transformer
```

```
    linear_model = joblib.load(linear_model_path)  # Linear regression model


    # Load performance metrics (optional)
    linear_r2 = joblib.load('linear_model_r2.pkl')
    linear_rmse = joblib.load('linear_model_rmse.pkl')
     polynomial_r2 = 0.9  # Example R² score (you can also load this if you want, based on training/test set)


except FileNotFoundError as e:
    print(f"Error: {e}")
    model = None
    poly = None
    linear_model = None


@app.route('/')
def index():
    return render_template("index.html", prediction=None, error=None)


@app.route('/predict', methods=['POST'])
def predict():
    if model is None or poly is None or linear_model is None:
        return render_template("index.html", prediction=None, error="Model is not loaded!")


    try:
        # Get user input
        car_speed = request.form.get('car_speed', type=float)
        car_weight = request.form.get('car_weight', type=float)


        if car_speed is None or car_weight is None:
            return render_template("index.html", prediction=None, error="Invalid input values!")
```

```
        # Input data
        input_data = np.array([[car_speed, car_weight]])

        # Polynomial model prediction
        poly_input = poly.transform(input_data)
        poly_pred = model.predict(poly_input)[0]

        # Linear model prediction
        linear_pred = linear_model.predict(input_data)[0]

        # Compare models and choose the better one
        if polynomial_r2 > linear_r2:  # You can also use RMSE if you prefer
            best_prediction = round(poly_pred, 2)
            model_used = "Polynomial Regression"
        else:
            best_prediction = round(linear_pred, 2)
            model_used = "Linear Regression"

        return render_template("index.html", prediction=best_prediction, model=model_used, error=None)

    except (ValueError, TypeError):
        return render_template("index.html", prediction=None, error="Please enter valid numbers!")

if __name__ == "__main__":
    app.run(debug=True)
```

index.html:

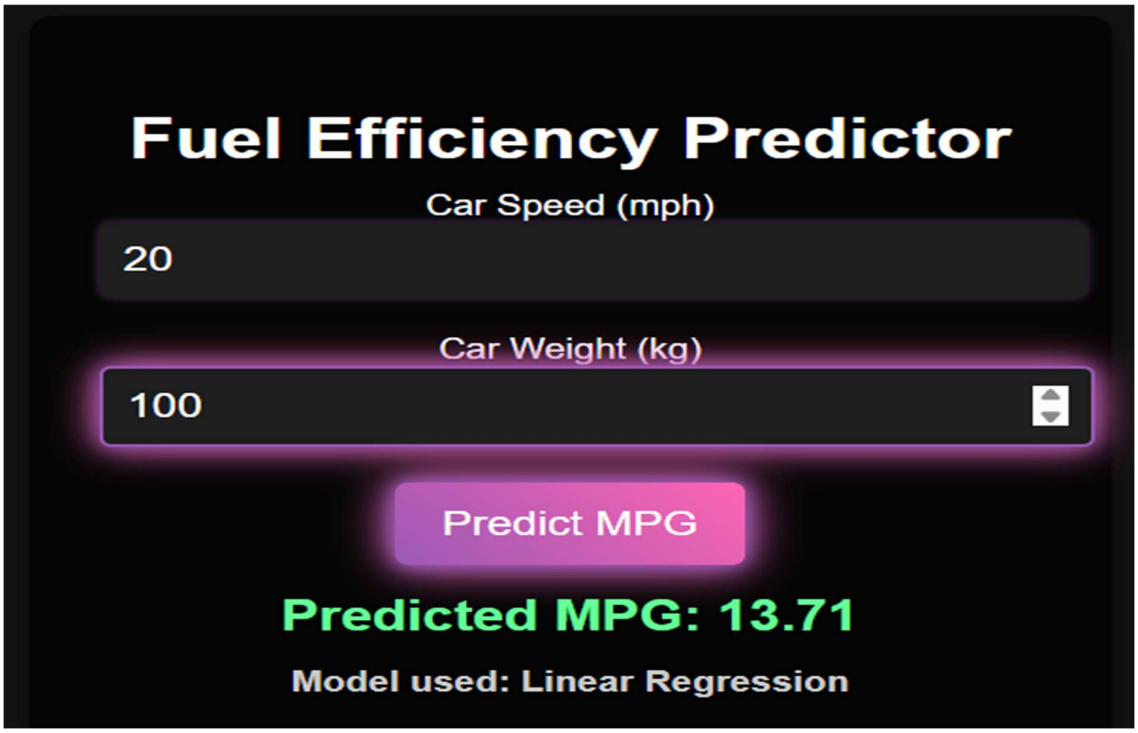
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Fuel Efficiency Predictor</title>
    <style>
        body {
            font-family: 'Arial', sans-serif;
            background: #121212;
            color: #fff;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
        }

        .container {
            background: rgba(0, 0, 0, 0.8);
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.6);
            text-align: center;
            width: 400px;
        }

        h1 {
            font-size: 2em;
            margin-bottom: 10px;
        }
```

```
input {
    width: 100%;
    padding: 10px;
    font-size: 1.2em;
    margin-bottom: 20px;
    border: none;
    border-radius: 5px;
    outline: none;
    background-color: #1e1e1e;
    color: #fff;
    transition: all 0.3s ease-in-out;
    box-shadow: 0 0 5px rgba(155, 89, 182, 0.5);
}

input:focus {
    box-shadow: 0 0 10px #9b59b6, 0 0 20px #ff66b2;
    border: 2px solid #9b59b6;
}

button {
    padding: 12px 20px;
    font-size: 1.2em;
    background: linear-gradient(45deg, #9b59b6, #ff66b2);
    color: #fff;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: all 0.3s ease-in-out;
    box-shadow: 0 0 8px #9b59b6, 0 0 16px #9b59b6, 0 0 24px #ff66b2;
}

button:hover {
```

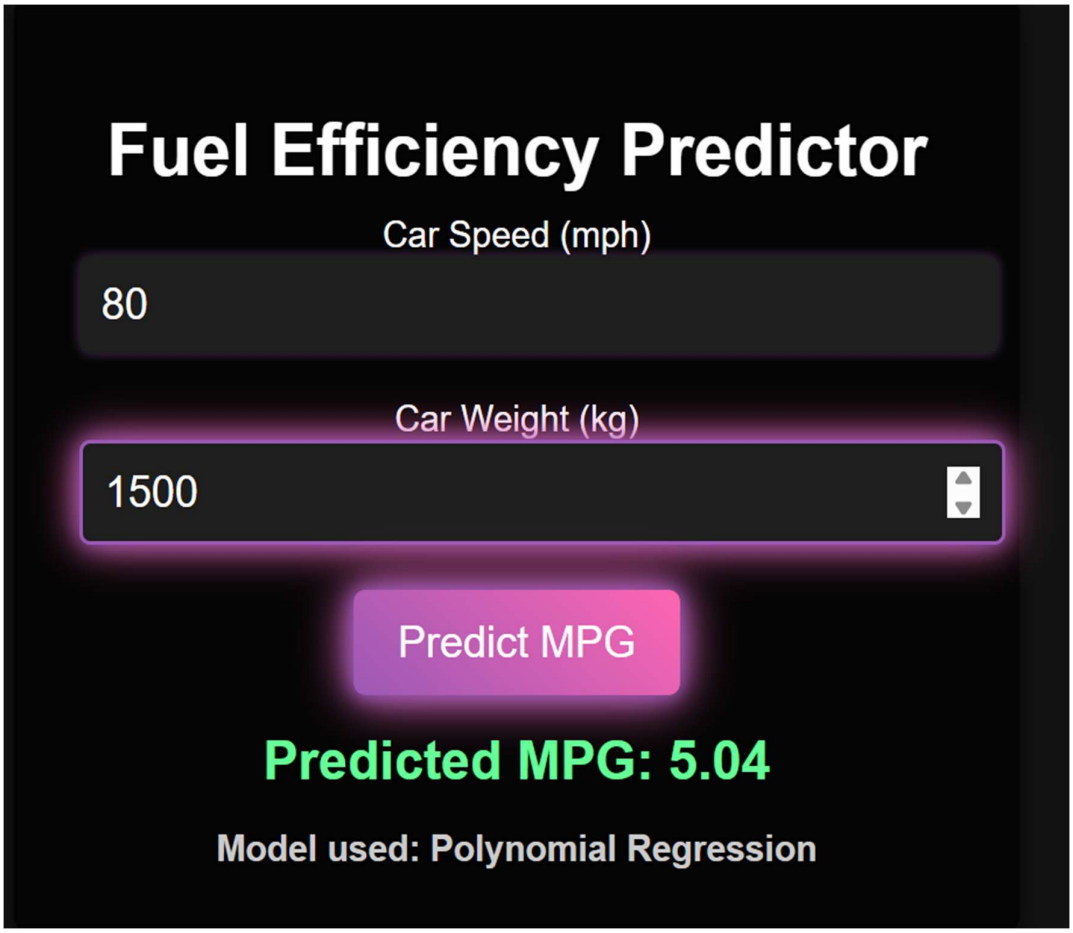
```
            box-shadow: 0 0 12px #9b59b6, 0 0 24px #9b59b6, 0 0 36px #ff66b2;
        }

        h2 {
            margin-top: 20px;
            font-size: 1.5em;
            font-weight: bold;
            color: #fff;
        }

        .error {
            color: #ff4d4d;
            font-size: 1.2em;
            margin-top: 10px;
        }

        .result {
            color: #66ff99;
            font-size: 1.5em;
            font-weight: bold;
            margin-top: 15px;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Fuel Efficiency Predictor</h1>
        <form action="/predict" method="POST">
            <label for="car_speed">Car Speed (mph)</label>
            <input type="number" id="car_speed" name="car_speed" required>

            <label for="car_weight">Car Weight (kg)</label>
            <input type="number" id="car_weight" name="car_weight" required>
```

```
        <button type="submit">Predict MPG</button>
      </form>

      {% if prediction is not none %}
        <div class="result">Predicted MPG: {{ prediction }}</div>
        <div class="result" style="font-size: 1em; color: #ccc;">Model used: {{ model }}</div>
      {% endif %}

      {% if error %}
        <div class="error">{{ error }}</div>
      {% endif %}
    </div>
</body>
</html>
```

Output:



Linear-regression model has high accuracy (for this input features) than polynomial-regression so it is predicted by linear-regression.



Polynomial-regression model has high accuracy (for this input features) than linear-regression so it is predicted by Polynomial-regression.