TASK TWO: PREDICTIVE MODELING WITH LINEAR REGRESSION

Implement a simple linear regression model using a dataset with continuous target variables. Split the data into training and testing sets, train the model on the training data, evaluate its performance using metrics like mean squared error or R-squared, and make predictions on the test set. Visualize the regression line and actual vs. predicted values to assess the model's accuracy.

In [1]:
```
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.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

In [2]:
```
# # Load the dataset
df = pd.read_csv('petrol_consumption.csv')
```

In [3]:
```
# Show the first few rows and the structure of the dataset
print(df.head())
```

```
   Petrol_tax  Average_income  Paved_Highways  Population_Driver_licence(%)  \
0         9.0            3571            1976                         0.525   
1         9.0            4092            1250                         0.572   
2         9.0            3865            1586                         0.580   
3         7.5            4870            2351                         0.529   
4         8.0            4399             431                         0.544   

   Petrol_Consumption  
0                 541  
1                 524  
2                 561  
3                 414  
4                 410  
```

In [4]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48 entries, 0 to 47
Data columns (total 5 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Petrol_tax                    48 non-null     float64
 1   Average_income                48 non-null     int64  
 2   Paved_Highways                48 non-null     int64  
 3   Population_Driver_licence(%)  48 non-null     float64
 4   Petrol_Consumption            48 non-null     int64  
dtypes: float64(2), int64(3)
memory usage: 2.0 KB
None
```

Define Features and Target Variable

In [5]:
```
# Define feature matrix (X) and target variable (y)
X = df[['Petrol_tax', 'Average_income', 'Paved_Highways', 'Population_Driver_licence(%)']]
y = df['Petrol_Consumption']
```

In [6]:
```
X
```

Out[6]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)
0	9.00	3571	1976	0.525
1	9.00	4092	1250	0.572
2	9.00	3865	1586	0.580
3	7.50	4870	2351	0.529
4	8.00	4399	431	0.544
5	10.00	5342	1333	0.571
6	8.00	5319	11868	0.451
7	8.00	5126	2138	0.553
8	8.00	4447	8577	0.529
9	7.00	4512	8507	0.552
10	8.00	4391	5939	0.530
11	7.50	5126	14186	0.525
12	7.00	4817	6930	0.574
13	7.00	4207	6580	0.545
14	7.00	4332	8159	0.608
15	7.00	4318	10340	0.586
16	7.00	4206	8508	0.572
17	7.00	3718	4725	0.540
18	7.00	4716	5915	0.724
19	8.50	4341	6010	0.677
20	7.00	4593	7834	0.663
21	8.00	4983	602	0.602
22	9.00	4897	2449	0.511
23	9.00	4258	4686	0.517
24	8.50	4574	2619	0.551
25	9.00	3721	4746	0.544
26	8.00	3448	5399	0.548
27	7.50	3846	9061	0.579
28	8.00	4188	5975	0.563
29	9.00	3601	4650	0.493
30	7.00	3640	6905	0.518
31	7.00	3333	6594	0.513
32	8.00	3063	6524	0.578
33	7.50	3357	4121	0.547
34	8.00	3528	3495	0.487
35	6.58	3802	7834	0.629
36	5.00	4045	17782	0.566
37	7.00	3897	6385	0.586
38	8.50	3635	3274	0.663
39	7.00	4345	3905	0.672
40	7.00	4449	4639	0.626
41	7.00	3656	3985	0.563
42	7.00	4300	3635	0.603
43	7.00	3745	2611	0.508
44	6.00	5215	2302	0.672
45	9.00	4476	3942	0.571
46	7.00	4296	4083	0.623
47	7.00	5002	9794	0.593

In [7]:

```
y
```

Out[7]:
```
0     541
1     524
2     561
3     414
4     410
5     457
6     344
7     467
8     464
9     498
10    580
11    471
12    525
13    508
14    566
15    635
16    603
17    714
18    865
19    640
20    649
21    540
22    464
23    547
24    460
25    566
26    577
27    631
28    574
29    534
30    571
31    554
32    577
33    628
34    487
35    644
36    640
37    704
38    648
39    968
40    587
41    699
42    632
43    591
44    782
45    510
46    610
47    524
Name: Petrol_Consumption, dtype: int64
```

Split the Data into Training and Testing Sets

In [8]:
```
# Split the dataset into 80% training and 20% testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [9]:
```
X_train
```

Out[9]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)
8	8.00	4447	8577	0.529
3	7.50	4870	2351	0.529
6	8.00	5319	11868	0.451
39	7.00	4345	3905	0.672
33	7.50	3357	4121	0.547
13	7.00	4207	6580	0.545
17	7.00	3718	4725	0.540
45	9.00	4476	3942	0.571
15	7.00	4318	10340	0.586
9	7.00	4512	8507	0.552
16	7.00	4206	8508	0.572
29	9.00	3601	4650	0.493
32	8.00	3063	6524	0.578
46	7.00	4296	4083	0.623
0	9.00	3571	1976	0.525
31	7.00	3333	6594	0.513
30	7.00	3640	6905	0.518
5	10.00	5342	1333	0.571
11	7.50	5126	14186	0.525
34	8.00	3528	3495	0.487
1	9.00	4092	1250	0.572
44	6.00	5215	2302	0.672
21	8.00	4983	602	0.602
2	9.00	3865	1586	0.580
36	5.00	4045	17782	0.566
35	6.58	3802	7834	0.629
23	9.00	4258	4686	0.517
41	7.00	3656	3985	0.563
10	8.00	4391	5939	0.530
22	9.00	4897	2449	0.511
18	7.00	4716	5915	0.724
47	7.00	5002	9794	0.593
20	7.00	4593	7834	0.663
7	8.00	5126	2138	0.553
42	7.00	4300	3635	0.603
14	7.00	4332	8159	0.608
28	8.00	4188	5975	0.563
38	8.50	3635	3274	0.663

In [10]:

```
y_train
```

Out[10]:
```
8     464
3     414
6     344
39    968
33    628
13    508
17    714
45    510
15    635
9     498
16    603
29    534
32    577
46    610
0     541
31    554
30    571
5     457
11    471
34    487
1     524
44    782
21    540
2     561
36    640
35    644
23    547
41    699
10    580
22    464
18    865
47    524
20    649
7     467
42    632
14    566
28    574
38    648
Name: Petrol_Consumption, dtype: int64
```

In [11]:
```
X_test
```

Out[11]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)
27	7.5	3846	9061	0.579
40	7.0	4449	4639	0.626
26	8.0	3448	5399	0.548
43	7.0	3745	2611	0.508
24	8.5	4574	2619	0.551
37	7.0	3897	6385	0.586
12	7.0	4817	6930	0.574
19	8.5	4341	6010	0.677
4	8.0	4399	431	0.544
25	9.0	3721	4746	0.544

In [12]:
```
y_test
```

Out[12]:
```
27    631
40    587
26    577
43    591
24    460
37    704
12    525
19    640
4     410
25    566
Name: Petrol_Consumption, dtype: int64
```

Train the Linear Regression Model

In [14]:
```
# Initialize and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[14]:
```
▾ LinearRegression
LinearRegression()
```

Make Predictions on the Test Set

In [15]:
```
# Make predictions using the test set
y_pred = model.predict(X_test)
```

In [16]:
```
y_pred
```

Out[16]:
```
array([606.69266519, 673.77944169, 584.99149034, 563.53691024,
       519.05867235, 643.46100256, 572.89761422, 687.07703573,
       547.6093662 , 530.03762971])
```

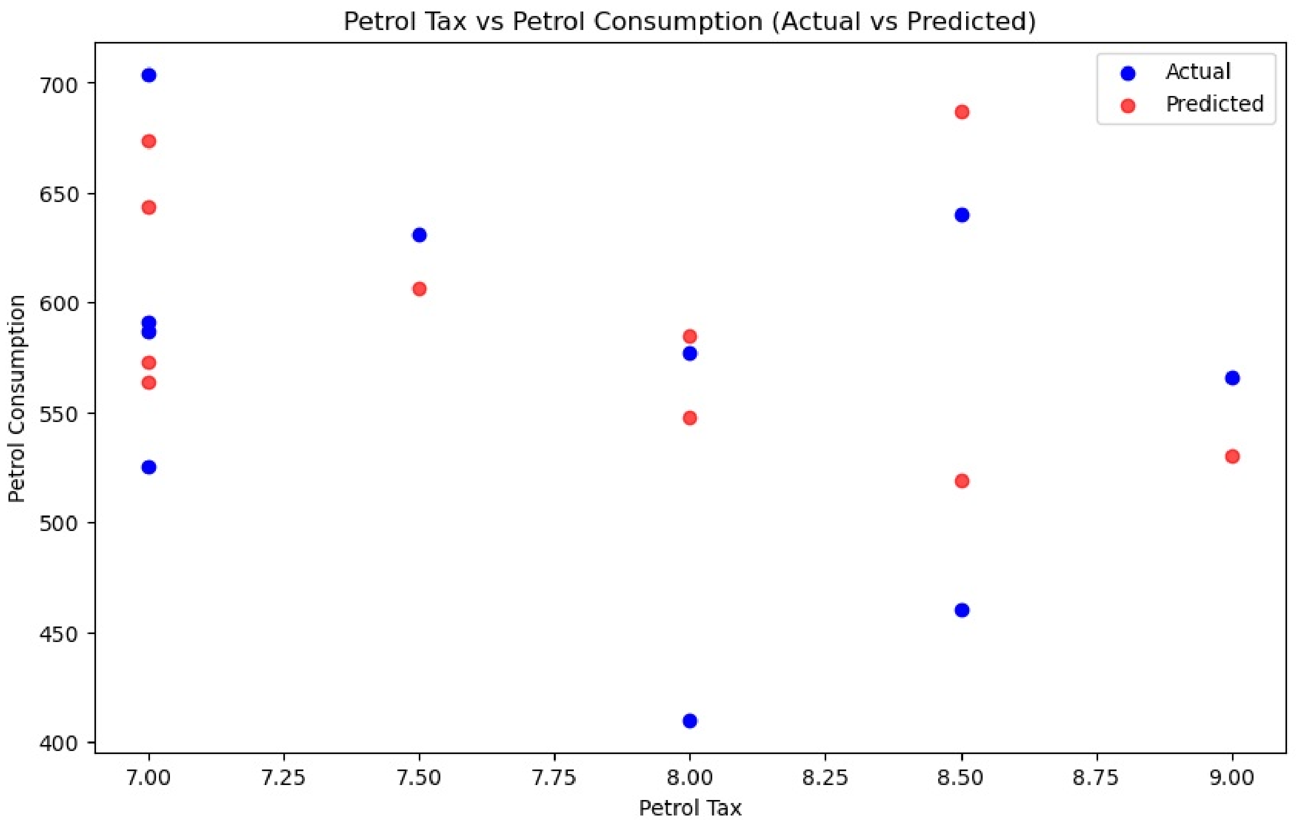
Evaluate the Model (MSE and R-squared)

In [17]:
```
# Calculate Mean Squared Error and R-squared score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 4083.255871745411
R-squared: 0.3913664001428835
```

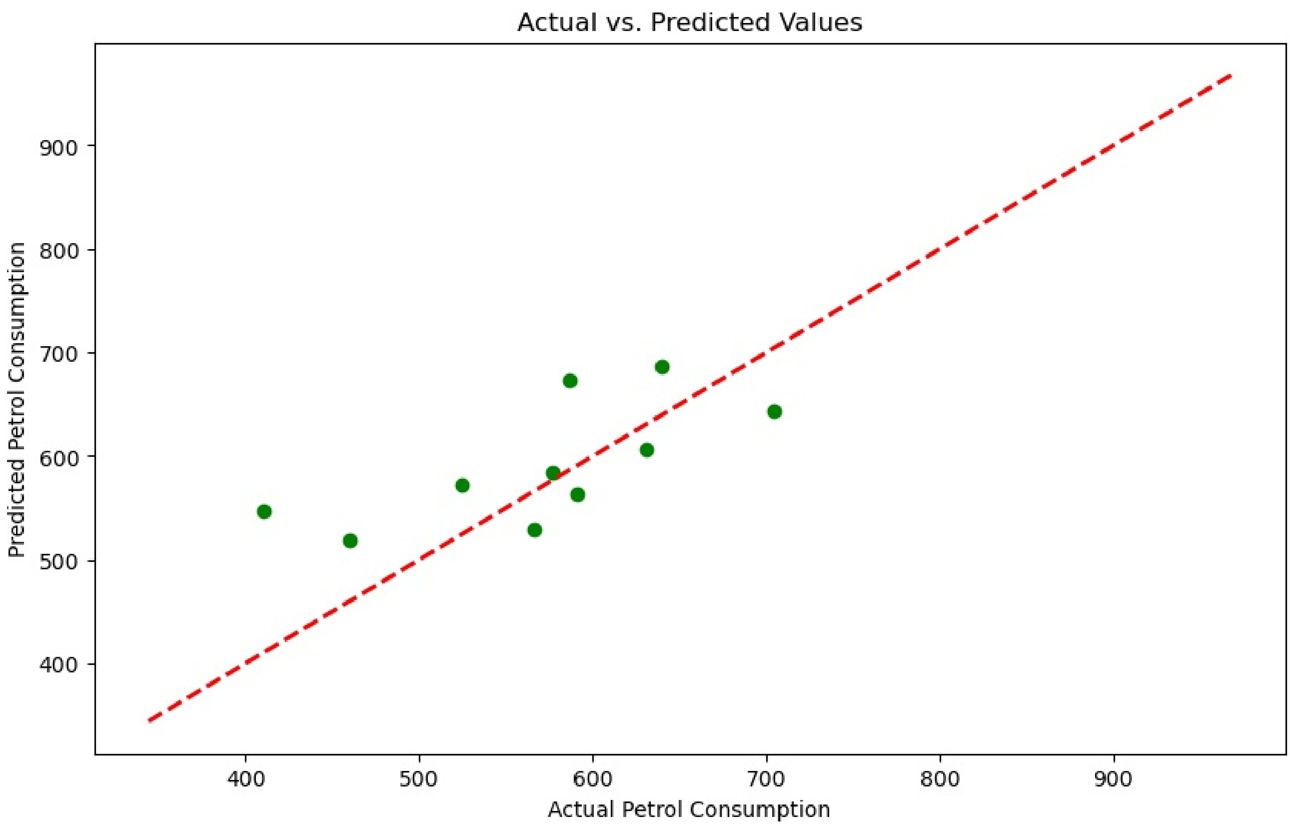
Visualize the Regression Line (for One Feature)

In [18]:
```
# Visualize the regression line for the 'Petrol_tax' feature
plt.figure(figsize=(10, 6))
plt.scatter(X_test['Petrol_tax'], y_test, color='blue', label='Actual')
plt.scatter(X_test['Petrol_tax'], y_pred, color='red', label='Predicted', alpha=0.7)
plt.title('Petrol Tax vs Petrol Consumption (Actual vs Predicted)')
plt.xlabel('Petrol Tax')
plt.ylabel('Petrol Consumption')
plt.legend()
plt.show()
```



Visualize Actual vs Predicted Values

In [19]:
```
# Visualize actual vs predicted petrol consumption
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='green')
plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', linestyle='--', linewidth=2)
plt.title('Actual vs. Predicted Values')
plt.xlabel('Actual Petrol Consumption')
plt.ylabel('Predicted Petrol Consumption')
plt.show()
```



Coefficient Interpretation

In [20]:
```
#Coefficients interpretation
coefficients = pd.DataFrame(model.coef_, X.columns, columns=['Coefficient'])
print(coefficients)
```

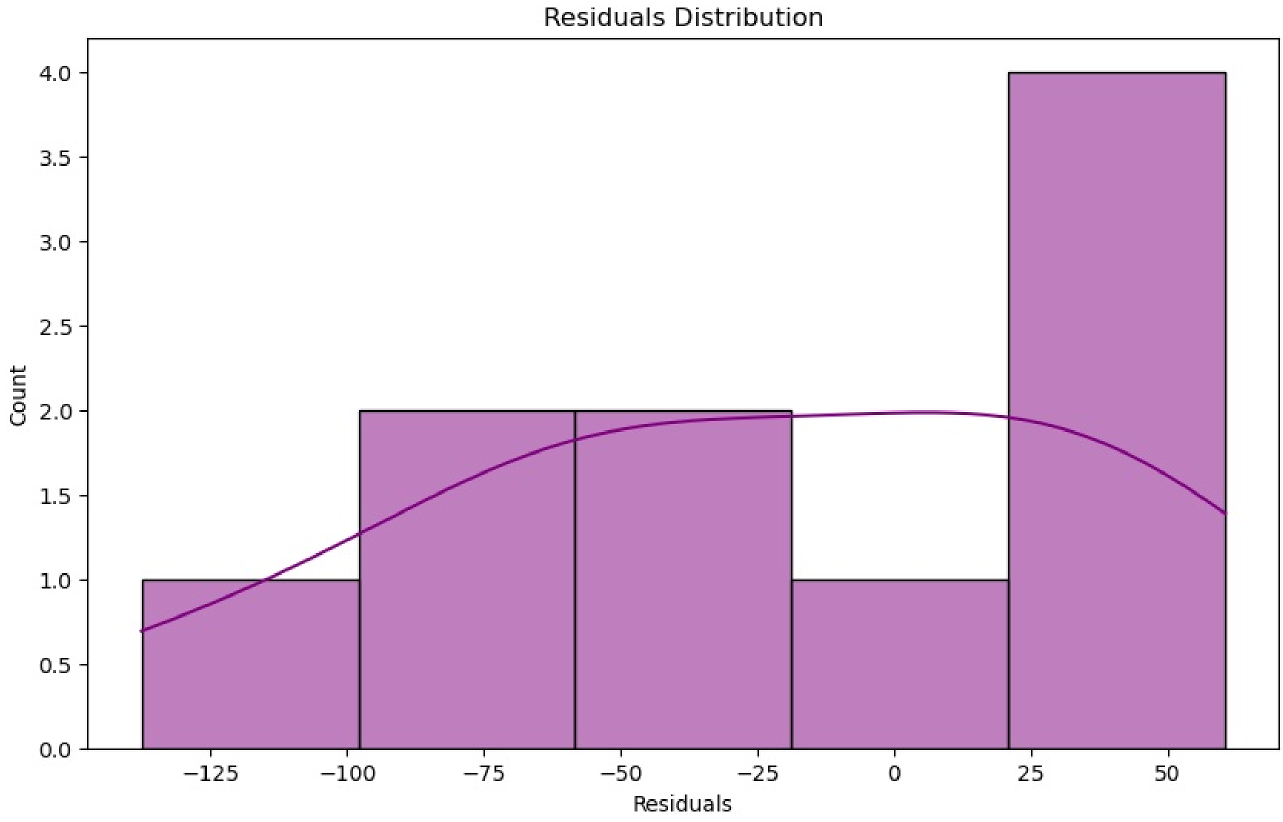
```
                              Coefficient
Petrol_tax                     -36.993746
Average_income                  -0.056536
Paved_Highways                  -0.004382
Population_Driver_licence(%)  1346.869298
```

Residual Analysis

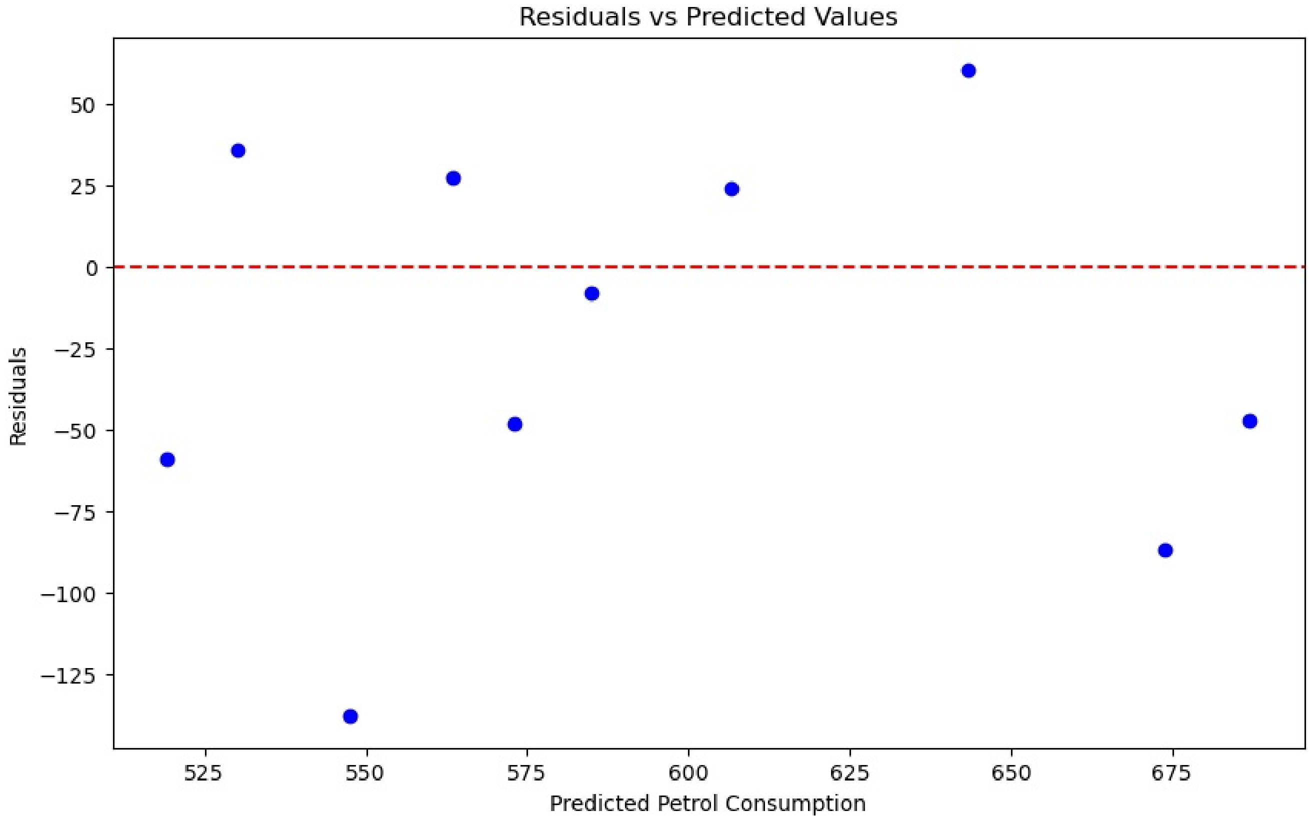
Residuals are the differences between actual and predicted values. Analyzing residuals helps check model assumptions.

In [21]:
```
residuals = y_test - y_pred

# Plot the residuals
plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True, color='purple')
plt.title('Residuals Distribution')
plt.xlabel('Residuals')
plt.show()
```



In [22]:
```
# Scatter plot of residuals vs. predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuals, color='blue')
plt.axhline(y=0, color='red', linestyle='--')
plt.title('Residuals vs Predicted Values')
plt.xlabel('Predicted Petrol Consumption')
plt.ylabel('Residuals')
plt.show()
```



Cross-Validation for Model Performance

Evaluate the model's robustness with cross-validation.

In [23]:
```
from sklearn.model_selection import cross_val_score
```

```
# Cross-validation with 5 folds
cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
mean_cv_score = np.mean(np.abs(cv_scores))

print(f'Mean Cross-Validation MSE: {mean_cv_score}')
```

```
Mean Cross-Validation MSE: 6012.505029227636
```

Outlier Detection

Outliers can significantly affect your model's performance. Here's a way to detect and visualize outliers using Z-scores

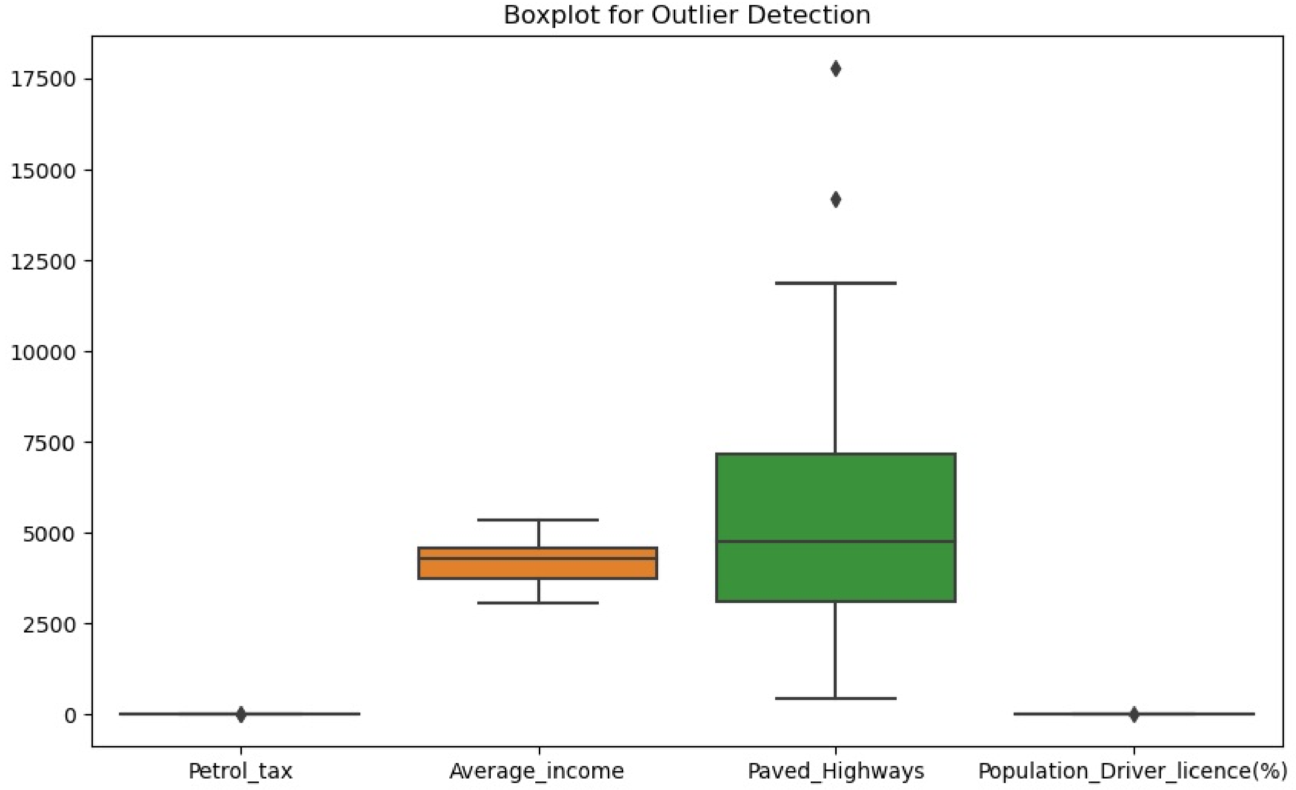
In [24]:
```
from scipy import stats

# Calculate Z-scores for detecting outliers
z_scores = np.abs(stats.zscore(X))

# Filter out rows with Z-scores greater than 3
outliers = np.where(z_scores > 3)
print(f'Outliers found at indices: {outliers}')

# Visualize outliers in the dataset
plt.figure(figsize=(10, 6))
sns.boxplot(data=X)
plt.title('Boxplot for Outlier Detection')
plt.show()
```

```
Outliers found at indices: (array([36], dtype=int64), array([2], dtype=int64))
```



Feature Scaling

Standardizing or normalizing your data ensures that features are on the same scale, which can improve model performance, especially for algorithms that rely on distance metrics.

In [26]:
```
from sklearn.preprocessing import StandardScaler

# Apply standard scaling to the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the scaled data into training and testing sets
X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled = train_test_split(X_scaled, y, test_size=0.2, ran

# Train a new linear regression model on the scaled data
model_scaled = LinearRegression()
model_scaled.fit(X_train_scaled, y_train_scaled)

# Evaluate the scaled model
y_pred_scaled = model_scaled.predict(X_test_scaled)
mse_scaled = mean_squared_error(y_test_scaled, y_pred_scaled)
```

```python
r2_scaled = r2_score(y_test_scaled, y_pred_scaled)
print(f'Scaled Model MSE: {mse_scaled}')
print(f'Scaled Model R-squared: {r2_scaled}')
```

```
Scaled Model MSE: 4083.255871745399
Scaled Model R-squared: 0.39136640014288526
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js