import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import statsmodels.api as sm  
from sklearn.linear\_model import LinearRegression  
from sklearn.model\_selection import train\_test\_split  
from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  
  
# Load the car price dataset from Kaggle or local directory  
# Replace 'car\_price\_data.csv' with the path to your file  
df = pd.read\_csv('car\_price\_data.csv')  
  
# Define your independent variables (X) and dependent variable (y)  
# Assuming the target variable is 'price' and other columns are predictors  
y = df['price']  
X = df.drop(columns=['price'])  
  
# Add a constant to X for statsmodels OLS  
X = sm.add\_constant(X)  
  
# Split the data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)  
  
# Fit the OLS model  
model = sm.OLS(y\_train, X\_train).fit()  
  
# Predict on the test set  
y\_pred = model.predict(X\_test)  
  
# Calculate residuals  
residuals = y\_test - y\_pred  
  
# Assumption 1: Linearity - Check with a scatter plot of observed vs predicted values  
plt.figure(figsize=(10, 6))  
plt.scatter(y\_pred, y\_test, alpha=0.6)  
plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'r--', lw=2)  
plt.xlabel('Predicted Prices')  
plt.ylabel('Observed Prices')  
plt.title('Observed vs Predicted Prices (Linearity Check)')  
plt.show()  
  
# Assumption 2: Normality of Errors - Check with histogram and Q-Q plot of residuals  
plt.figure(figsize=(12, 6))  
  
# Histogram  
plt.subplot(1, 2, 1)  
sns.histplot(residuals, kde=True)  
plt.xlabel('Residuals')  
plt.title('Residuals Histogram (Normality Check)')  
  
# Q-Q plot  
plt.subplot(1, 2, 2)  
sm.qqplot(residuals, line='45', fit=True)  
plt.title('Q-Q Plot of Residuals (Normality Check)')  
plt.show()  
  
# Assumption 3: Homoscedasticity - Check with a scatter plot of residuals vs predicted values  
plt.figure(figsize=(10, 6))  
plt.scatter(y\_pred, residuals, alpha=0.6)  
plt.axhline(0, color='r', linestyle='--', linewidth=2)  
plt.xlabel('Predicted Prices')  
plt.ylabel('Residuals')  
plt.title('Residuals vs Predicted Prices (Homoscedasticity Check)')  
plt.show()  
  
# Assumption 4: No Multicollinearity - Check Variance Inflation Factor (VIF)  
vif\_data = pd.DataFrame()  
vif\_data["feature"] = X\_train.columns  
vif\_data["VIF"] = [variance\_inflation\_factor(X\_train.values, i) for i in range(X\_train.shape[1])]  
  
print("Variance Inflation Factors (VIF):")  
print(vif\_data)  
  
# Assumption 5: No Autocorrelation - Check using Durbin-Watson test  
from statsmodels.stats.stattools import durbin\_watson  
dw\_stat = durbin\_watson(residuals)  
print(f'Durbin-Watson statistic: {dw\_stat}')  
  
# Durbin-Watson statistic interpretation  
if dw\_stat < 1.5:  
    print("Potential positive autocorrelation.")  
elif dw\_stat > 2.5:  
    print("Potential negative autocorrelation.")  
else:  
    print("No significant autocorrelation.")

def myfunc(x,y):

if x <= 30 and y == 2:

return y

elif x > 90 and y == 2:

return y + 1

else:

return y

data['column2'] = data.apply(lambda x: myfunc(x.column1, x.column2), axis=1)

C Parameter:

* The C parameter controls the trade-off between maximizing the margin and minimizing the classification error.
* A smaller C value allows for a softer margin, meaning the classifier will tolerate more misclassifications in the training data. This can lead to a simpler decision boundary and potentially higher bias, which may result in underfitting.
* Conversely, a larger C value imposes a harder margin, meaning the classifier will aim to classify all training examples correctly. This can lead to a more complex decision boundary and potentially higher variance, which may result in overfitting, especially if the training data contains noise or outliers.

gamma parameter :

* The gamma parameter determines the influence of a single training example, with low values meaning 'far' and high values meaning 'close'.
* A small gamma value implies a large radius of influence, where points further away are considered in the calculation of the decision boundary. This can lead to a smoother decision boundary and potentially higher bias, resulting in underfitting.
* Conversely, a large gamma value implies a smaller radius of influence, where only nearby points are considered in the calculation of the decision boundary. This can lead to a more complex decision boundary and potentially higher variance, resulting in overfitting, especially if the training data is noisy or contains irrelevant features.

Small epsilon:

* When the epsilon value is small, the margin of tolerance around the predicted value is narrow. This means that the model aims to fit the training data closely, potentially leading to a more complex model with lower bias.
* Consequently, with a small epsilon, the model may be prone to overfitting, especially when the training data contains noise or outliers. The model may capture the noise or irregularities in the training data, resulting in poor generalization performance on unseen data.

large epsilon:

* Conversely, when the epsilon value is large, the margin of tolerance around the predicted value is wider. This means that the model allows for more deviations from the exact fit, potentially leading to a simpler model with higher bias.
* With a large epsilon, the model is less sensitive to individual data points and may generalize better to unseen data. It can help prevent overfitting, especially when the training data is noisy or contains outliers.