Raúl Correa Ocañas | A01722401 | 05-03-2024

M3. Actividad 4. Modelos de regresión lineal y sistemas de recomendaciones

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from sklearn.model_selection import train_test_split
import statsmodels.stats as sms
import statsmodels.api as sm
```

In [ ]:
```
data = pd.read_csv(r'C:\Users\Raul\OneDrive\Escritorio\CS\TC2004B.101\M3-Act4\menu.
data.head()
```

Out[ ]:

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Tr
0	Breakfast	Egg McMuffin	4.8 oz (136 g)	300	120	13.0	20	5.0	25	
1	Breakfast	Egg White Delight	4.8 oz (135 g)	250	70	8.0	12	3.0	15	
2	Breakfast	Sausage McMuffin	3.9 oz (111 g)	370	200	23.0	35	8.0	42	
3	Breakfast	Sausage McMuffin with Egg	5.7 oz (161 g)	450	250	28.0	43	10.0	52	
4	Breakfast	Sausage McMuffin with Egg Whites	5.7 oz (161 g)	400	210	23.0	35	8.0	42	

5 rows × 24 columns

In [ ]:
```
data.isna().any()
```

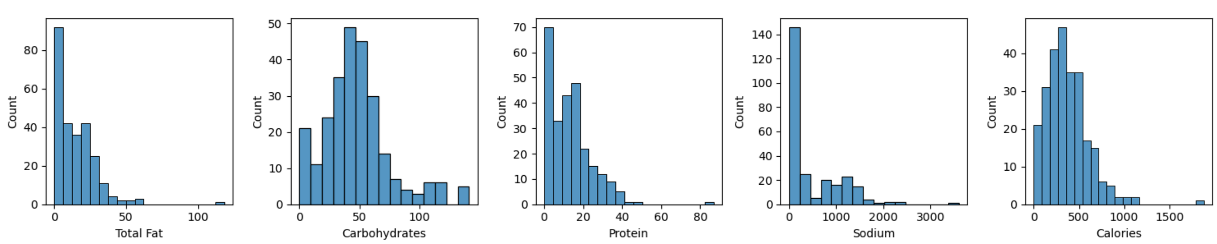
Out[ ]:
```
Category                          False
Item                              False
Serving Size                      False
Calories                          False
Calories from Fat                 False
Total Fat                         False
Total Fat (% Daily Value)         False
Saturated Fat                     False
Saturated Fat (% Daily Value)     False
Trans Fat                         False
Cholesterol                       False
Cholesterol (% Daily Value)       False
Sodium                            False
Sodium (% Daily Value)            False
Carbohydrates                     False
Carbohydrates (% Daily Value)     False
Dietary Fiber                     False
Dietary Fiber (% Daily Value)     False
Sugars                            False
Protein                           False
Vitamin A (% Daily Value)         False
Vitamin C (% Daily Value)         False
Calcium (% Daily Value)           False
Iron (% Daily Value)              False
dtype: bool
```

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260 entries, 0 to 259
Data columns (total 24 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Category                        260 non-null    object 
 1   Item                            260 non-null    object 
 2   Serving Size                    260 non-null    object 
 3   Calories                        260 non-null    int64  
 4   Calories from Fat               260 non-null    int64  
 5   Total Fat                       260 non-null    float64
 6   Total Fat (% Daily Value)       260 non-null    int64  
 7   Saturated Fat                   260 non-null    float64
 8   Saturated Fat (% Daily Value)   260 non-null    int64  
 9   Trans Fat                       260 non-null    float64
 10  Cholesterol                     260 non-null    int64  
 11  Cholesterol (% Daily Value)     260 non-null    int64  
 12  Sodium                          260 non-null    int64  
 13  Sodium (% Daily Value)          260 non-null    int64  
 14  Carbohydrates                   260 non-null    int64  
 15  Carbohydrates (% Daily Value)   260 non-null    int64  
 16  Dietary Fiber                   260 non-null    int64  
 17  Dietary Fiber (% Daily Value)   260 non-null    int64  
 18  Sugars                          260 non-null    int64  
 19  Protein                         260 non-null    int64  
 20  Vitamin A (% Daily Value)       260 non-null    int64  
 21  Vitamin C (% Daily Value)       260 non-null    int64  
 22  Calcium (% Daily Value)         260 non-null    int64  
 23  Iron (% Daily Value)            260 non-null    int64  
dtypes: float64(3), int64(18), object(3)
memory usage: 48.9+ KB
```

In [ ]:

```python
# data = pd.get_dummies(data, drop_first=True)
variables = ['Total Fat', 'Carbohydrates', 'Protein', 'Sodium', 'Calories']
data = data[variables]
data.head()


fig = plt.figure(figsize=(15,3))
counter = 0
for column in data:
    counter += 1
    plt.subplot(1,5, counter)
    sns.histplot(data, x=column)

plt.tight_layout()
```



In [ ]:
```
X = data.drop('Calories', axis=1)
y = data['Calories']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_s
```

0. Funciones

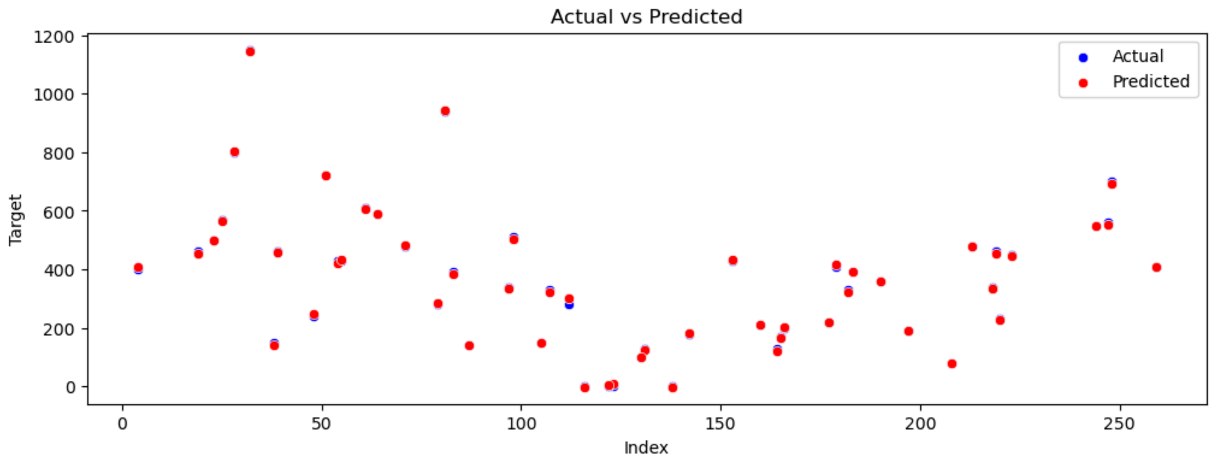
In [ ]:
```
def forward_selection(X_train, y_train, model, metric):
    testing_features = list(X_train.columns)
    final = []
    scores = [(-1, None)]
    highscore = -1

    while (max(scores)[0] < metric):
        if (len(testing_features) == 0):
            print('Metric not met.')
            print('R2:', highscore)
            return final

        scores = []
        for column in X_train[testing_features]:
            features = final + [column]
            X_temp = X_train[features]
            model.fit(X_temp, y_train)
            score = model.score(X_temp, y_train)
            scores.append((score, column))

        if (highscore < max(scores)[0]):
            final.append(max(scores)[1])
            highscore = max(scores)[0]

        testing_features.remove(max(scores)[1])
        print('Depth:', len(final), ' - ', final, highscore)
    print(final)
    return final
```

1. Lineal Multivariante

In [ ]:
```
from sklearn.linear_model import LinearRegression
reg1 = LinearRegression()

features = forward_selection(X_train, y_train, reg1, 0.99)
X_train_1 = X_train[features]
X_test_1 = X_test[features]

reg1.fit(X_train_1, y_train)

y_pred_1 = reg1.predict(X_test_1)

plt.figure(figsize=(12, 4))
sns.scatterplot(x=y_test.index, y=y_test, color='blue', label='Actual')
sns.scatterplot(x=y_test.index, y=y_pred_1, color='red', label='Predicted')
```

```
plt.xlabel('Index')
plt.ylabel('Target')
plt.title('Actual vs Predicted')
plt.legend()
plt.show()
```

```
Depth: 1  -  ['Total Fat'] 0.8132128731140438
Depth: 2  -  ['Total Fat', 'Carbohydrates'] 0.9870735117068954
Depth: 3  -  ['Total Fat', 'Carbohydrates', 'Protein'] 0.999480255395087
['Total Fat', 'Carbohydrates', 'Protein']
```



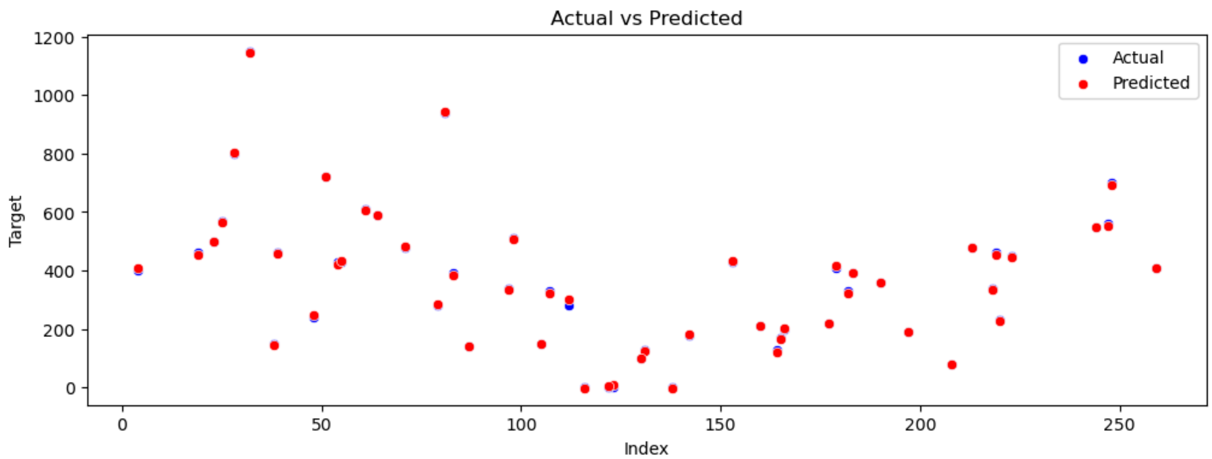
In [ ]:
```
from sklearn.metrics import r2_score

n = len(X_train_1) # Number of registers
k = len(X_train_1.columns) # Number of columns

r2 = r2_score(y_pred_1, y_test)
r2_adj = 1 - (1-r2)*(n-1)/(n-k-1)

col1 = pd.DataFrame({'Lineal Multivariante' : [r2, r2_adj]}, index=['R^2', 'R^2 Adj
```

2. Polinomial Multivariante

In [ ]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

poly_features = PolynomialFeatures(degree=3, interaction_only=False, include_bias=F
X_poly_train = pd.DataFrame(poly_features.fit_transform(X_train))
X_poly_test = pd.DataFrame(poly_features.transform(X_test))

reg2 = LinearRegression()
features = forward_selection(X_poly_train, y_train, reg2, 0.99)
X_train_2 = X_poly_train[features]
X_test_2 = X_poly_test[features]

# Ajustar el modelo a los datos
reg2.fit(X_train_2, y_train)

y_pred_2 = reg2.predict(X_test_2)

plt.figure(figsize=(12, 4))
```

```
sns.scatterplot(x=y_test.index, y=y_test, color='blue', label='Actual')
sns.scatterplot(x=y_test.index, y=y_pred_2, color='red', label='Predicted')
plt.xlabel('Index')
plt.ylabel('Target')
plt.title('Actual vs Predicted')
plt.legend()
plt.show()
```

```
Depth: 1  -  [5] 0.8175422978281982
Depth: 2  -  [5, 1] 0.8794855463590275
Depth: 3  -  [5, 1, 0] 0.9872538558739244
Depth: 4  -  [5, 1, 0, 2] 0.9994804759036104
[5, 1, 0, 2]
```



In [ ]:

```
from sklearn.metrics import r2_score

n = len(X_train_2) # Number of registers
k = len(X_train_2.columns) # Number of columns

r2 = r2_score(y_pred_2, y_test)
r2_adj = 1 - (1-r2)*(n-1)/(n-k-1)

col2 = pd.DataFrame({'Polinomial Multivariante' : [r2, r2_adj]}, index=['R^2', 'R^2
```

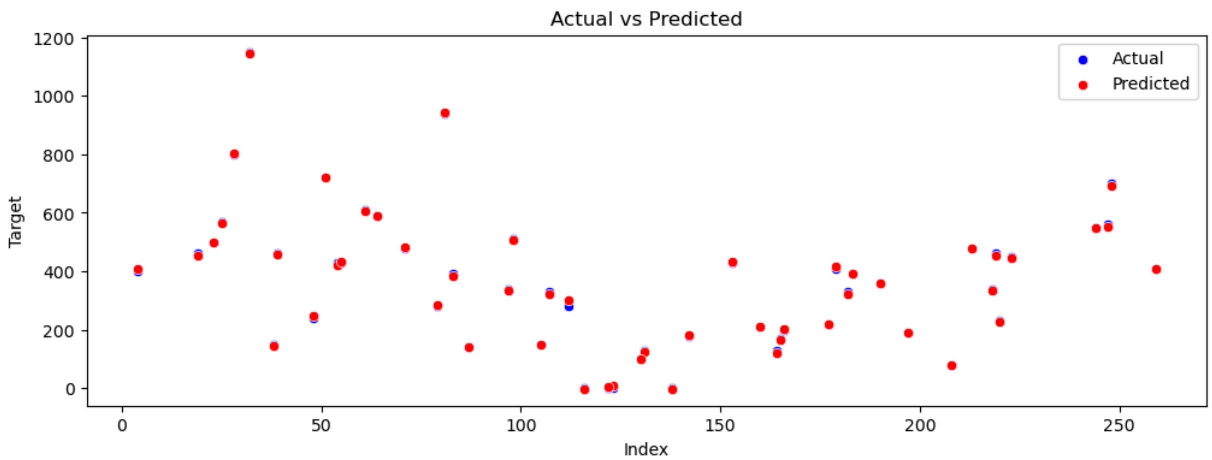
3. Polinomial Multivariante con Interacciones entre las Variables de Entrada

In [ ]:

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

poly_features = PolynomialFeatures(degree=3, interaction_only=True, include_bias=Fa
X_poly_train = pd.DataFrame(poly_features.fit_transform(X_train))
X_poly_test = pd.DataFrame(poly_features.transform(X_test))

reg3 = LinearRegression()
features = forward_selection(X_poly_train, y_train, reg3, 0.99)
X_train_3 = X_poly_train[features]
X_test_3 = X_poly_test[features]

# Ajustar el modelo a los datos
```

```python
reg3.fit(X_train_3, y_train)

y_pred_3 = reg3.predict(X_test_3)


plt.figure(figsize=(12, 4))
sns.scatterplot(x=y_test.index, y=y_test, color='blue', label='Actual')
sns.scatterplot(x=y_test.index, y=y_pred_3, color='red', label='Predicted')
plt.xlabel('Index')
plt.ylabel('Target')
plt.title('Actual vs Predicted')
plt.legend()
plt.show()
```

```
Depth: 1  -  [4] 0.8175422978281982
Depth: 2  -  [4, 1] 0.8794855463590275
Depth: 3  -  [4, 1, 0] 0.9872538558739244
Depth: 4  -  [4, 1, 0, 2] 0.9994804759036104
[4, 1, 0, 2]
```



In [ ]:

```python
from sklearn.metrics import r2_score

n = len(X_train_3) # Number of registers
k = len(X_train_3.columns) # Number of columns

r2 = r2_score(y_pred_3, y_test)
r2_adj = 1 - (1-r2)*(n-1)/(n-k-1)

col3 = pd.DataFrame({'Polinomial Multivariante con Interacciones' : [r2, r2_adj]},
```

4. Evaluación de Métricas

In [ ]:

```python
metrics = pd.concat([col1, col2, col3], axis=1)
metrics
```

Out[ ]:

	Lineal Multivariante	Polinomial Multivariante	Polinomial Multivariante con Interacciones
R^2	0.999442	0.999441	0.999441
R^2 Adjusted	0.999433	0.999430	0.999430

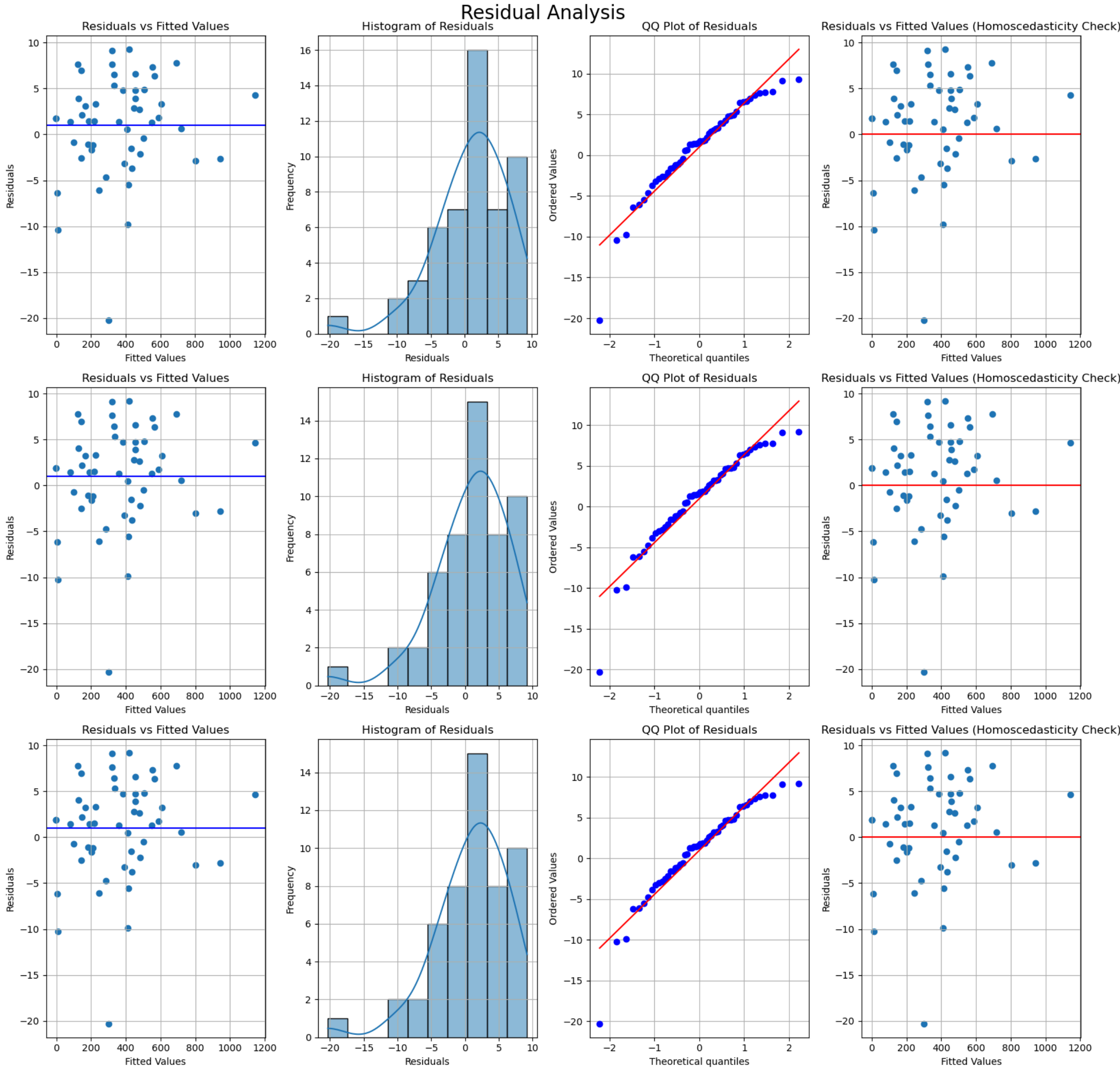
5. Supuestos de Regresión Lineal

In [ ]:

```
def vibe_check(y_test, y_preds):
    fig = plt.figure(figsize=(16,16))
    fig.suptitle('Residual Analysis', fontsize=20)  # Add title here
    index = 0

    for y_pred in y_preds:
        residuals = y_test - y_pred

        # Plot residuals vs fitted values to check for independence
        plt.subplot(3,4,1 + 4*index)
        plt.scatter(y_pred, residuals)
        plt.axhline(y=np.mean(residuals), color='b', linestyle='-')
        plt.title('Residuals vs Fitted Values')
        plt.xlabel('Fitted Values')
        plt.ylabel('Residuals')
        plt.grid(True)

        # Plot histogram of residuals
        plt.subplot(3,4,2 + 4*index)
        sns.histplot(residuals, kde=True, bins=10)
        plt.title('Histogram of Residuals')
        plt.xlabel('Residuals')
        plt.ylabel('Frequency')
        plt.grid(True)

        # Plot QQ plot of residuals
        plt.subplot(3,4,3 + 4*index)
        stats.probplot(residuals, dist="norm", plot=plt)
        plt.title('QQ Plot of Residuals')
        plt.grid(True)

        # Check homoscedasticity using predicted values and residuals
        plt.subplot(3,4,4 + 4*index)
        plt.scatter(y_pred, residuals)
        plt.axhline(y=0, color='r', linestyle='-')
        plt.title('Residuals vs Fitted Values (Homoscedasticity Check)')
        plt.xlabel('Fitted Values')
        plt.ylabel('Residuals')
        plt.grid(True)

        index += 1
```

```python
    plt.tight_layout()
    plt.show()
```

In [ ]:
```python
vibe_check(y_test, [y_pred_1, y_pred_2, y_pred_3])
```



Supuestos de una Regresión Lineal

1. **Linealidad**: Se dice que un modelo de regresión lineal cumple el supuesto de linealidad cuando la ecuación utilizada representa adecuadamente una relación lineal entre variables dependientes e independientes.

2. **Normalidad**: Se dice que un modelo de regresión lineal cumple el supuesto de normalidad cuando los residuos del modelo siguen una distribución normal. Para verificar este supuesto, se puede utilizar la prueba de Shapiro-Wilk.

3. **Homocedasticidad**:
   Se dice que un modelo de regresión lineal cumple el supuesto de homocedasticidad cuando la varianza de los residuos es constante a lo largo de los valores de las variables

independientes. Para verificar este supuesto, se puede utilizar la prueba de Breusch-Pagan.

4. **Independencia**: Se dice que un modelo de regresión lineal cumple el supuesto de independencia cuando los residuos del modelo no están correlacionados entre sí. Para verificar este supuesto, se puede utilizar la prueba de Durbin-Watson.

In [ ]:
```
from statsmodels.stats.diagnostic import het_breuschpagan, normal_ad
from statsmodels.stats.stattools import durbin_watson
from statsmodels.tools.tools import add_constant

def check_assumptions(y_test, y_preds):
    index = 1
    for y_pred in y_preds:
        residuals = y_test - y_pred

        # Normality test
        test_statistic, p_value_normality = normal_ad(residuals)
        print(f"Model {index}")
        print(f"Normality Test (Shapiro-Wilk): p-value = {p_value_normality:.4f}")

        # Homoscedasticity test
        # Add a constant column to X_test
        X_test_with_const = add_constant(X_test)
        _, p_value_homoscedasticity, _, _ = het_breuschpagan(residuals, X_test_with
        print(f"Homoscedasticity Test (Breusch-Pagan): p-value = {p_value_homosceda

        # Independence test
        durbin_watson_statistic = durbin_watson(residuals)
        print(f"Durbin-Watson Statistic: {durbin_watson_statistic:.4f}")

        index += 1
        print("---------------------")

# Check assumptions for each model
check_assumptions(y_test, [y_pred_1, y_pred_2, y_pred_3])
```

```
Model 1
Normality Test (Shapiro-Wilk): p-value = 0.0597
Homoscedasticity Test (Breusch-Pagan): p-value = 0.0851
Durbin-Watson Statistic: 2.1360
---------------------
Model 2
Normality Test (Shapiro-Wilk): p-value = 0.0614
Homoscedasticity Test (Breusch-Pagan): p-value = 0.0855
Durbin-Watson Statistic: 2.1402
---------------------
Model 3
Normality Test (Shapiro-Wilk): p-value = 0.0614
Homoscedasticity Test (Breusch-Pagan): p-value = 0.0855
Durbin-Watson Statistic: 2.1402
---------------------
```

Con estas pruebas, además de verlo gráficamente, se puede verificar que todos los modelos cumplen los supuestos de Normalidad, Homocedasticidad e Independencia. El supuesto de

Linealidad se puede corroborar con una prueba de ANOVA. Al trabajar con Scikit-Learn, no se puede realizar la prueba de ANOVA, pero se puede verificar con la gráfica de residuos, mostrando empiricamente que el modelo es lineal.