# AP-STATS

## FINAL PROJECT-CODE

## Optimizing Biomass Prediction- using regression methods and mitigating the multicollinearity present in the environmental data

## SCEENSHOTS:



### A. PART-1

```
import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

df = pd.read_csv('LINTHALL.txt', delim_whitespace=True, header=None, names=['Index', 'Loc', 'Type', 'BIO', 'H2S', 'SAL', 'Eh7', 'pH', 'BU
df = df[['BIO', 'H2S', 'SAL', 'Eh7', 'pH', 'BUF', 'P', 'K', 'Ca', 'Mg', 'Na', 'Mn', 'Zn', 'Cu', 'NH4']]

df[['BIO', 'H2S', 'SAL', 'Eh7', 'pH', 'BUF', 'P', 'K', 'Ca', 'Mg', 'Na', 'Mn', 'Zn', 'Cu', 'NH4']] = df[['BIO', 'H2S', 'SAL', 'Eh7', 'pH

df = df.dropna()

# Data Modelling
X = df[['H2S', 'SAL', 'Eh7', 'pH', 'BUF', 'P', 'K', 'Ca', 'Mg', 'Na', 'Mn', 'Zn', 'Cu', 'NH4']]
y = df['BIO']

X = sm.add_constant(X)

# Fit the OLS OLS_model
OLS_model = sm.OLS(y, X).fit()

print(OLS_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    BIO   R-squared:                       0.823
Model:                            OLS   Adj. R-squared:                  0.734
Method:                 Least Squares   F-statistic:                     9.270
Date:                Thu, 07 Dec 2023   Prob (F-statistic):           4.03e-07
Time:                        02:43:38   Log-Likelihood:                -302.70
No. Observations:                  43   AIC:                             635.4
Df Residuals:                      28   BIC:                             661.8
Df Model:                          14
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3475.9507   3441.050      1.010      0.321   -3572.720    1.05e+04
H2S            1.1544      3.048      0.379      0.708      -5.089       7.398
SAL          -19.2305     26.581     -0.723      0.475     -73.679      35.218
Eh7            2.4120      1.964      1.228      0.230      -1.612       6.435
pH           149.1615    330.050      0.452      0.655    -526.915     825.238
BUF          -19.6909    121.063     -0.163      0.872    -267.676     228.295
P             -6.1819      3.854     -1.604      0.120     -14.077       1.713
K             -1.0168      0.474     -2.144      0.041      -1.988      -0.045
Ca            -0.0657      0.125     -0.524      0.604      -0.323       0.191
Mg            -0.3667      0.273     -1.343      0.190      -0.926       0.192
Na             0.0100      0.024      0.411      0.684      -0.040       0.060
Mn            -3.6814      5.513     -0.668      0.510     -14.975       7.612
Zn            -8.0818     21.989     -0.368      0.716     -53.125      36.961
Cu           373.8948    110.351      3.388      0.002     147.852     599.938
NH4           -1.5510      3.219     -0.482      0.634      -8.145       5.043
==============================================================================
Omnibus:                       10.120   Durbin-Watson:                   1.791
Prob(Omnibus):                  0.006   Jarque-Bera (JB):               14.888
Skew:                           0.602   Prob(JB):                     0.000585
Kurtosis:                       5.619   Cond. No.                     1.22e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.22e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
# VIF for predictors
VIF_value = pd.DataFrame()
VIF_value["Variable"] = X.columns
VIF_value["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print("Variance Inflation Factor (VIF):")
print(VIF_value)

# Check for high VIF values
vif_above_threshold = VIF_value[VIF_value["VIF"] > 10]["Variable"].tolist()
if vif_above_threshold:
    print(f"\nWarning: High VIF values for variables present, Possible collinearity: {', '.join(vif_above_threshold)}")

# Check for condition number
condition_no = np.linalg.cond(X.values)
print(f"\nCondition Number: {condition_no}")

# Check for a high condition number
if condition_no > 30:
    print("\nWarning: High condition number found. Possible multicollinearity.")
```

```
Variance Inflation Factor (VIF):
   Variable          VIF
0     const  4350.771896
1       H2S     3.136506
2       SAL     3.361283
3       Eh7     1.964076
4        pH    62.564383
5       BUF    33.478422
6         P     2.884226
7         K     7.432133
8        Ca    17.343432
9        Mg    24.476419
10       Na    10.372624
11       Mn     6.737786
12       Zn    12.391033
13       Cu     4.866983
14      NH4     8.586275

Warning: High VIF values for variables present, Possible collinearity: const, pH, BUF, Ca, Mg, Na, Zn

Condition Number: 1224956.4891047853

Warning: High condition number found. Possible multicollinearity.
```

```
#correlation corr_matrix
corr_matrix = df.corr()
corr_matrix = corr_matrix.unstack()
corr_matrix = corr_matrix[abs(corr_matrix) >= 0.7]

df.corr()
```

| | BIO | H2S | SAL | Eh7 | pH | BUF | P | K | Ca | Mg | Na | Mn | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **BIO** | 1.000000 | 0.316601 | -0.074151 | 0.056594 | 0.769706 | -0.724724 | -0.565869 | -0.195804 | 0.647834 | -0.368199 | -0.257882 | -0.378190 | -0.63 |
| **H2S** | 0.316601 | 1.000000 | 0.169190 | 0.430436 | 0.259976 | -0.360190 | -0.284927 | 0.074175 | 0.097201 | -0.090801 | 0.020682 | 0.133887 | -0.29 |
| **SAL** | -0.074151 | 0.169190 | 1.000000 | 0.296771 | -0.028847 | -0.044753 | -0.061205 | -0.020650 | 0.085460 | -0.035222 | 0.140143 | -0.257924 | -0.42 |
| **Eh7** | 0.056594 | 0.430436 | 0.296771 | 1.000000 | 0.101170 | -0.163304 | -0.326211 | 0.427850 | -0.045807 | 0.294838 | 0.338063 | -0.111838 | -0.22 |
| **pH** | 0.769706 | 0.259976 | -0.028847 | 0.101170 | 1.000000 | -0.946283 | -0.579402 | 0.028564 | 0.882288 | -0.165704 | -0.023624 | -0.499060 | -0.73 |
| **BUF** | -0.724724 | -0.360190 | -0.044753 | -0.163304 | -0.946283 | 1.000000 | 0.590230 | -0.084963 | -0.797199 | 0.115988 | -0.081745 | 0.453243 | 0.72 |
| **P** | -0.565869 | -0.284927 | -0.061205 | -0.326211 | -0.579402 | 0.590230 | 1.000000 | -0.243725 | -0.391782 | -0.008379 | -0.118908 | 0.540701 | 0.66 |
| **K** | -0.195804 | 0.074175 | -0.020650 | 0.427850 | 0.028564 | -0.084963 | -0.243725 | 1.000000 | -0.259956 | 0.864980 | 0.795619 | -0.340707 | 0.06 |
| **Ca** | 0.647834 | 0.097201 | 0.085460 | -0.045807 | 0.882288 | -0.797199 | -0.391782 | -0.259956 | 1.000000 | -0.419053 | -0.248135 | -0.321979 | -0.69 |
| **Mg** | -0.368199 | -0.090801 | -0.035222 | 0.294838 | -0.165704 | 0.115988 | -0.008379 | 0.864980 | -0.419053 | 1.000000 | 0.898470 | -0.212432 | 0.35 |
| **Na** | -0.257882 | 0.020682 | 0.140143 | 0.338063 | -0.023624 | -0.081745 | -0.118908 | 0.795619 | -0.248135 | 0.898470 | 1.000000 | -0.304095 | 0.12 |
| **Mn** | -0.378190 | 0.133887 | -0.257924 | -0.111838 | -0.499060 | 0.453243 | 0.540701 | -0.340707 | -0.321979 | -0.212432 | -0.304095 | 1.000000 | 0.61 |
| **Zn** | -0.638293 | -0.291303 | -0.422449 | -0.227293 | -0.731131 | 0.728321 | 0.660655 | 0.069921 | -0.699351 | 0.350283 | 0.121679 | 0.613772 | 1.00 |
| **Cu** | 0.101182 | 0.002390 | -0.259403 | 0.102979 | 0.187439 | -0.152510 | -0.070325 | 0.692069 | -0.104983 | 0.720131 | 0.568729 | -0.228238 | 0.20 |
| **NH4** | -0.619499 | -0.423095 | -0.181808 | -0.244983 | -0.742309 | 0.848178 | 0.670013 | -0.127974 | -0.581496 | 0.098942 | -0.120313 | 0.549798 | 0.72 |

```
corr_incl_BIO = corr_matrix.loc['BIO']
print(corr_incl_BIO)
```

```
BIO    1.000000
pH     0.769706
BUF   -0.724724
dtype: float64
```

```
# Extract R-squared and MSE from the summary
r_sqrd = OLS_model.rsquared
mse = mean_squared_error(y, OLS_model.predict(X))

# Display R-squared and MSE
print("R-squared:", r_sqrd)
print("Mean Squared Error:", mse)
```

```
R-squared: 0.8225336852667445
Mean Squared Error: 76203.26233465268
```

## B. PART-2

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('LINTHALL.txt', delim_whitespace=True, header=None, names=['Index', 'Loc', 'Type', 'BIO', 'H2S', 'SAL', 'Eh7', 'pH', '

df = df[['BIO', 'H2S', 'SAL', 'Eh7', 'pH', 'BUF', 'P', 'K', 'Ca', 'Mg', 'Na', 'Mn', 'Zn', 'Cu', 'NH4']]
df[['BIO', 'H2S', 'SAL', 'Eh7', 'pH', 'BUF', 'P', 'K', 'Ca', 'Mg', 'Na', 'Mn', 'Zn', 'Cu', 'NH4']] = df[['BIO', 'H2S', 'SAL', 'Eh7', 'p

df = df.dropna()

# Separate predictors and response
X = df[['H2S', 'SAL', 'Eh7', 'pH', 'BUF', 'P', 'K', 'Ca', 'Mg', 'Na', 'Mn', 'Zn', 'Cu', 'NH4']]
y = df['BIO']


# Normalize Predictors
norm = StandardScaler()
X_norm = norm.fit_transform(X)

#Perform PCA
pca = PCA()
X_PCA = pca.fit_transform(X_norm)

# Choose the no of components using explained variance
CVR = np.cumsum(pca.explained_variance_ratio_)
no_of_comp = np.argmax(CVR >= 0.95) + 1

X_selected = X_PCA[:, :no_of_comp]

# Fit OLS model with selected principal components
PCR_model = sm.OLS(y, sm.add_constant(X_selected)).fit()
print(PCR_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    BIO   R-squared:                       0.747
Model:                            OLS   Adj. R-squared:                  0.687
Method:                 Least Squares   F-statistic:                     12.55
Date:                Thu, 07 Dec 2023   Prob (F-statistic):           3.58e-08
Time:                        02:43:38   Log-Likelihood:                -310.32
No. Observations:                  43   AIC:                             638.6
Df Residuals:                      34   BIC:                             654.5
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        991.7209     56.525     17.545      0.000     876.847    1106.594
x1           211.7561     24.855      8.520      0.000     161.246     262.267
x2           -79.7898     29.430     -2.711      0.010    -139.599     -19.980
x3          -105.9213     44.526     -2.379      0.023    -196.410     -15.433
x4           118.5306     50.912      2.328      0.026      15.064     221.997
x5           -65.1063     67.943     -0.958      0.345    -203.183      72.970
x6            -0.2428     80.564     -0.003      0.998    -163.968     163.482
x7           263.5300     91.874      2.868      0.007      76.819     450.241
x8           -52.8079    110.546     -0.478      0.636    -277.464     171.849
==============================================================================
Omnibus:                       10.353   Durbin-Watson:                   1.319
Prob(Omnibus):                  0.006   Jarque-Bera (JB):                9.712
Skew:                           1.017   Prob(JB):                      0.00778
```

```
Kurtosis:                       4.134   Cond. No.                         4.45
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
Load = pca.components_[:no_of_comp, :].T

# Obtain SE of PCR coefficients
PCR_stderr = PCR_model.bse[1:]

# Obtain SE present in original model
OG_stderr = PCR_stderr * np.sqrt(np.sum(Load**2, axis=0))



print("\nS.E in the Original Model:")
print(OG_stderr)

PCR_sumstderr = np.sum(PCR_model.resid**2)
print(f"\nSum of Squared Errors (SSE) in PCR Model: {PCR_sumstderr}")

OLS_sumstderr = np.sum(OLS_model.resid**2)
print(f"\nSum of Squared Errors (SSE) in Part I: {OLS_sumstderr}")

pcr_beta_coeff = np.dot(pca.components_[:no_of_comp, :].T, PCR_model.params[1:])
print("\nNo of Selected Components:", no_of_comp)
print("\nPCR Model Coefficients:")
print(pcr_beta_coeff)
```

```
S.E in the Original Model:
x1     24.854520
x2     29.430315
x3     44.526356
x4     50.912355
x5     67.942904
x6     80.563640
x7     91.874291
x8    110.545942
dtype: float64




Sum of Squared Errors (SSE) in PCR Model: 4671275.614573438

Sum of Squared Errors (SSE) in Part I: 3276740.280390065

No of Selected Components: 8

PCR Model Coefficients:
[ 125.70668178  -86.2186242    -8.63028393  129.88488336  -59.22878046
  -80.50078756  -11.2439618    52.10035796 -101.89851081 -191.80647504
 -106.87369348 -105.75567087  173.31065863  -11.22260787]
```

## C.PART-3

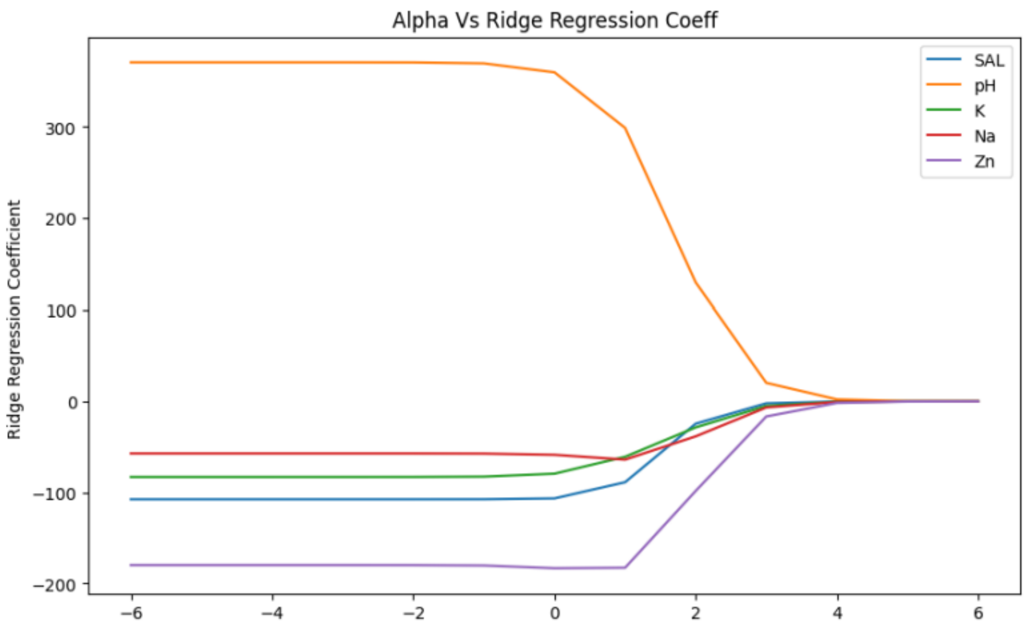
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

df = pd.read_table("LINTH-5.txt", delim_whitespace=True)

# Pred and response
Pred = ["SAL", "pH", "K", "Na", "Zn"]
response = "BIO"

# Forward Selection
selected_pred = []
alpha_E = 0.10
alpha_R = 0.10
thold = 10

while True:
    rem_preds = [p for p in Pred if p not in selected_pred]
    pval_best = float('inf')
    pred_best = None

    for predictor in rem_preds:
        model = sm.OLS(df[response], sm.add_constant(df[selected_pred + [predictor]])).fit()
        pval = model.pvalues[predictor]

        if pval < pval_best:
            pval_best = pval
            pred_best = predictor

    if pval_best < alpha_E:
        selected_pred.append(pred_best)
        print(f"Added {pred_best} to the model (p-value = {pval_best:.4f})")
    else:
        break


# Fit model
model_stepwise = sm.OLS(df[response], sm.add_constant(df[selected_pred])).fit()
print("\nFinal Model Summary:")
print(model_stepwise.summary())

# Calculate VIF
X = sm.add_constant(df[selected_pred])
vif_collinearity = pd.DataFrame()
vif_collinearity["Variable"] = X.columns
vif_collinearity["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif_collinearity = vif_collinearity[vif_collinearity["Variable"] != "const"]
print("\nVariance Inflation Factor (VIF) without the constant term:")
print(vif_collinearity)

# Collinearity check using VIF
if vif_collinearity["VIF"].max() < thold:
    print("\nThere is no significant multicollinearity, and collinearity has disappeared.")
else:
    print("\nThere is potential multicollinearity in the model.")
```

```
Added pH to the model (p-value = 0.0000)
Added Na to the model (p-value = 0.0142)


Final Model Summary:
                            OLS Regression Results
==============================================================================
Dep. Variable:                    BIO   R-squared:                       0.650
Model:                            OLS   Adj. R-squared:                  0.632
Method:                 Least Squares   F-statistic:                     37.13
Date:                Thu, 07 Dec 2023   Prob (F-statistic):           7.64e-10
Time:                        02:43:38   Log-Likelihood:                -317.31
No. Observations:                  43   AIC:                             640.6
Df Residuals:                      40   BIC:                             645.9
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -466.3748    279.219     -1.670      0.103   -1030.698      97.948
pH           400.4547     49.046      8.165      0.000     301.329     499.580
Na            -0.0227      0.009     -2.563      0.014      -0.041      -0.005
==============================================================================
Omnibus:                       10.456   Durbin-Watson:                   0.919
Prob(Omnibus):                  0.005   Jarque-Bera (JB):                9.845
Skew:                           1.082   Prob(JB):                      0.00728
Kurtosis:                       3.901   Cond. No.                     8.32e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
[2] The condition number is large, 8.32e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

Variance Inflation Factor (VIF) without the constant term:
  Variable       VIF
1       pH  1.000558
2       Na  1.000558

There is no significant multicollinearity, and collinearity has disappeared.
```

```
import pandas as pd
import numpy as np
from sklearn.linear_model import RidgeCV
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
import matplotlib.pyplot as plt

data = pd.read_table("LINTH-5.txt", delim_whitespace=True)

# Predictors and Response
X_pred = data[['SAL', 'pH', 'K', 'Na', 'Zn']]
Y_pred = data['BIO']
thold = 10

# Normalize predictors
norm = StandardScaler()
X_norm = norm.fit_transform(X_pred)

# Ridge regression
alphas = np.logspace(-6, 6, 13)
ridgereg_CV = RidgeCV(alphas=alphas, store_cv_values=True)
ridgereg_CV.fit(X_norm, Y_pred)

# Selected alpha
alpha_sel = ridgereg_CV.alpha_
ridgecv_mse = np.mean(ridgereg_CV.cv_values_, axis=0)
ridgereg_trace = pd.DataFrame({'Alpha': alphas, 'CV_MSE': ridgecv_mse})
print("Ridge Trace:")
print(ridgereg_trace)
print("\nSelected Alpha:", alpha_sel)


# Fit Ridge model
ridge_reg_model = RidgeCV(alphas=[alpha_sel])
ridge_reg_model.fit(X_norm, Y_pred)
print("\nCoefficients after Ridge Regression:")
print(ridge_reg_model.coef_)

# Collinearity check
vif_coll_df = pd.DataFrame()
vif_coll_df["Variable"] = X_pred.columns
vif_coll_df["VIF"] = [variance_inflation_factor(X_norm, i) for i in range(X_norm.shape[1])]
print(vif_coll_df)

if vif_coll_df["VIF"].max() < thold:
    print("\nThere is no significant multicollinearity, and collinearity has disappeared.")
else:
    print("\nThere is potential multicollinearity in the model.")

alphas = np.logspace(-6, 6, 13)

beta_coefficients_ridge = []

for alpha in alphas:
    ridge_model = RidgeCV(alphas=[alpha], store_cv_values=True)
    ridge_model.fit(X_norm, Y_pred)
    beta_coefficients_ridge.append(ridge_model.coef_)

beta_coefficients_ridge = np.array(beta_coefficients_ridge)

plt.figure(figsize=(10, 6))
for i in range(X_pred.shape[1]):
    plt.plot(np.log10(alphas), beta_coefficients_ridge[:, i], label=f'{X_pred.columns[i]}')

plt.xlabel('log(alpha)')
plt.ylabel('Ridge Regression Coefficient ')
plt.title('Alpha Vs Ridge Regression Coeff')
plt.legend()
plt.show()
```

```
Ridge Trace:
            Alpha         CV_MSE
0        0.000001  184039.533707
1        0.000010  184039.506122
2        0.000100  184039.230279
3        0.001000  184036.472578
4        0.010000  184008.967802
5        0.100000  183740.944616
6        1.000000  181613.423435
7       10.000000  178600.163499
8      100.000000  272801.058463
9     1000.000000  416737.992603
10   10000.000000  446451.321466
11  100000.000000  449719.685767
12 1000000.000000  450049.764103

Selected Alpha: 10.0

Coefficients after Ridge Regression:
[ -88.782382    298.9606666   -60.86999177  -63.95162596 -182.67008386]
  Variable       VIF
0      SAL  2.099364
1       pH  3.327339
2        K  2.982513
3       Na  3.311625
4       Zn  4.309322

There is no significant multicollinearity, and collinearity has disappeared.
```



```
from sklearn.linear_model import LinearRegression
from itertools import combinations
from statsmodels.stats.outliers_influence import variance_inflation_factor

df = pd.read_csv('LINTH-5.txt', delim_whitespace=True)

# Predictors and Response
X_pred = df[['SAL', 'pH', 'K', 'Na', 'Zn']]
Y_pred = df['BIO']
thold=10

# Function BIC
def BIC_function(X, Y, features):
    model_bic = sm.OLS(Y, sm.add_constant(X[features])).fit()
    bic_value = len(Y) * np.log(model_bic.mse_resid) + len(features) + 1  * np.log(len(Y))
    return bic_value

#  Best Subset Selection
def subset_selection(X, Y, max_features=2):
    df_features = X.columns
    bestvalue_bic = float('inf')
    bestsubset_feature = None

    for r in range(1, max_features + 1):
        for subset in combinations(df_features, r):
            bic_value = BIC_function(X, Y, list(subset))
            if bic_value < bestvalue_bic:
                bestvalue_bic = bic_value
                bestsubset_feature = subset

model.summary()
```

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | BIO | **R-squared:** | 0.656 |
| **Model:** | OLS | **Adj. R-squared:** | 0.629 |
| **Method:** | Least Squares | **F-statistic:** | 24.74 |
| **Date:** | Thu, 07 Dec 2023 | **Prob (F-statistic):** | 3.92e-09 |
| **Time:** | 02:43:39 | **Log-Likelihood:** | -316.96 |
| **No. Observations:** | 43 | **AIC:** | 641.9 |