```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Define the column names provided in wine.names
column_names = ['Class', 'Alcohol', 'Malic Acid', 'Ash', 'Alcalinity of Ash', 'Magnesium',
                'Total Phenols', 'Flavanoids', 'Nonflavanoid Phenols', 'Proanthocyanins',
                'Color Intensity', 'Hue', 'OD280/OD315 of Diluted Wines', 'Proline']

# Import the data file into the Google colab and load the dataset
wine_data = pd.read_csv('wine.data', header=None, names=column_names)

# Display the first 5 rows
wine_data.head()
```

	Class	Alcohol	Malic Acid	Ash	Alcalinity of Ash	Magnesium	Total Phenols	Flavanoids	Nonflavanoid Phenols	Proanthocyanins	Color Intensity	Hue	OD280/ of D
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	

Next steps: View recommended plots

```
# Gathering information about the dataset using .info()
wine_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Class                         178 non-null    int64
 1   Alcohol                       178 non-null    float64
 2   Malic Acid                    178 non-null    float64
 3   Ash                           178 non-null    float64
 4   Alcalinity of Ash             178 non-null    float64
 5   Magnesium                     178 non-null    int64
 6   Total Phenols                 178 non-null    float64
 7   Flavanoids                    178 non-null    float64
 8   Nonflavanoid Phenols          178 non-null    float64
 9   Proanthocyanins               178 non-null    float64
 10  Color Intensity               178 non-null    float64
 11  Hue                           178 non-null    float64
 12  OD280/OD315 of Diluted Wines  178 non-null    float64
 13  Proline                       178 non-null    int64
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

```
# Set a seed for the random number generator for reproducibility
np.random.seed(0)

# Manually constructing an evaluation set of 5 data points.
# We are creating random data for 3 predictors: 'Alcohol', 'Malic Acid', and 'Ash'.
# The random data is drawn from a normal distribution centered around the mean of the actual data, with a spread corresponding t
# This creates a simulated dataset that resembles the properties of the original dataset.
evaluation_set = {
    'Alcohol': np.random.normal(wine_data['Alcohol'].mean(), wine_data['Alcohol'].std(), 5),
    'Malic Acid': np.random.normal(wine_data['Malic Acid'].mean(), wine_data['Malic Acid'].std(), 5),
    'Ash': np.random.normal(wine_data['Ash'].mean(), wine_data['Ash'].std(), 5)
}

# Convert the dictionary to a pandas DataFrame for easier handling and analysis later on.
evaluation_set_df = pd.DataFrame(evaluation_set)
```

```
# Simple Linear Regression

# Selecting the predictor variable 'Malic Acid' for the regression model.
# This variable will be used as the independent variable (X) in the model.
X = wine_data['Malic Acid']

# Selecting the target variable 'Alcohol' for the regression model.
# This variable will be predicted by the model and serves as the dependent variable (y).
y = wine_data['Alcohol']

# Adding a constant to the predictor variable.
# This is necessary because statsmodels' OLS does not include a bias term (intercept) by default.
# By adding a constant, we are effectively adding an intercept term to the model.
X_with_constant = sm.add_constant(X)

# Fitting the simple linear regression model using OLS (Ordinary Least Squares).
# 'y' is the target variable, and 'X_with_constant' contains the predictor with an added constant term.
# The '.fit()' method is used to fit the model to the data, finding the coefficients that minimize the residual sum of squares.
simple_lm_model = sm.OLS(y, X_with_constant).fit()
```

```
# Multicollinearity analysis

# Selecting predictors for analysis
predictors = wine_data[['Malic Acid', 'Ash', 'Alcalinity of Ash', 'Magnesium']]

# Adding a constant for intercept in the analysis
predictors_with_constant = sm.add_constant(predictors)
```

```
# Calculating VIF for each predictor

# Initialize a DataFrame to store VIF results
vif_data = pd.DataFrame()

# Store the names of the predictors
vif_data["Feature"] = predictors_with_constant.columns

# Calculate VIF for each predictor using a list comprehension
# VIF is calculated for each feature by iterating through all columns
vif_data["VIF"] = [variance_inflation_factor(predictors_with_constant.values, i)
                   for i in range(predictors_with_constant.shape[1])]
```

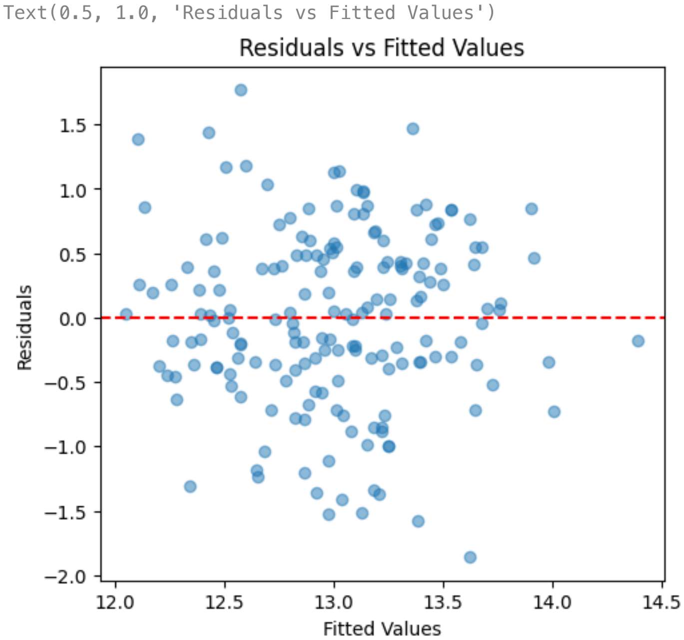
```
# Multiple Linear Regression

# Fitting a multiple linear regression model using Ordinary Least Squares (OLS) method.
# The model uses 'y' as the dependent variable and 'predictors_with_constant' as independent variables.
multiple_lm_model = sm.OLS(y, predictors_with_constant).fit()

# Generating a summary of the multiple linear regression model to review performance metrics and coefficients.
multiple_lm_summary = multiple_lm_model.summary()
```

```
# Predictions for the evaluation set with multiple predictors

# Selecting initial predictors from the evaluation set.
evaluation_set_multiple_predictors = evaluation_set_df[['Malic Acid', 'Ash']]

# Simulating 'Alcalinity of Ash' and 'Magnesium' for the evaluation set based on original data distribution.
evaluation_set_multiple_predictors['Alcalinity of Ash'] = np.random.normal(wine_data['Alcalinity of Ash'].mean(), wine_data['Alc
evaluation_set_multiple_predictors['Magnesium'] = np.random.normal(wine_data['Magnesium'].mean(), wine_data['Magnesium'].std(),

# Adding a constant term for intercept in predictions.
evaluation_set_multiple_predictors_with_constant = sm.add_constant(evaluation_set_multiple_predictors)

# Predicting 'Alcohol' content using the fitted multiple linear regression model.
evaluation_set_multiple_predictions = multiple_lm_model.predict(evaluation_set_multiple_predictors_with_constant)

# Printing summary of the multiple linear regression model for review.
print(multiple_lm_summary)
```

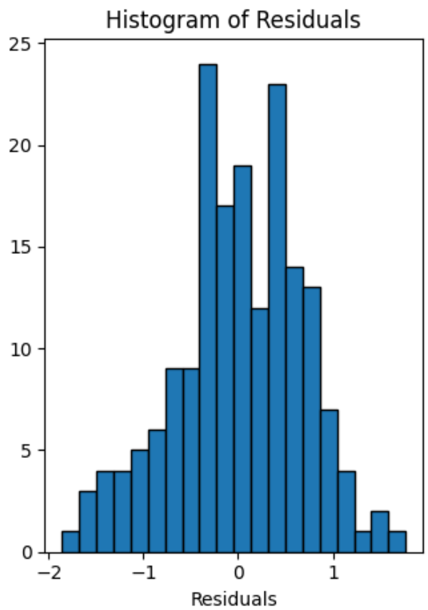
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Alcohol   R-squared:                       0.294
Model:                            OLS   Adj. R-squared:                  0.277
Method:                 Least Squares   F-statistic:                     17.98
Date:                Fri, 23 Feb 2024   Prob (F-statistic):           2.32e-12
Time:                        14:46:00   Log-Likelihood:                -184.03
No. Observations:                 178   AIC:                             378.1
Df Residuals:                     173   BIC:                             394.0
Df Model:                           4
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const                11.7846      0.544     21.655      0.000      10.710      12.859
Malic Acid            0.1383      0.049      2.844      0.005       0.042       0.234
Ash                   1.1045      0.227      4.872      0.000       0.657       1.552
Alcalinity of Ash    -0.1263      0.018     -6.885      0.000      -0.163      -0.090
Magnesium             0.0074      0.004      1.900      0.059      -0.000       0.015
==============================================================================
Omnibus:                        1.448   Durbin-Watson:                   1.272
Prob(Omnibus):                  0.485   Jarque-Bera (JB):                1.521
Skew:                          -0.207   Prob(JB):                        0.467
Kurtosis:                       2.817   Cond. No.                     1.10e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.1e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```python
# Residual Analysis for Multiple Linear Regression Model

# Calculate residuals from the model
residuals = multiple_lm_model.resid

# Visualizing the relationship between fitted values and residuals to check assumptions
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
# Scatter plot of residuals vs. fitted values
plt.scatter(multiple_lm_model.fittedvalues, residuals, alpha=0.5)
# Horizontal line at zero to aid in visualizing deviation
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs Fitted Values')
```

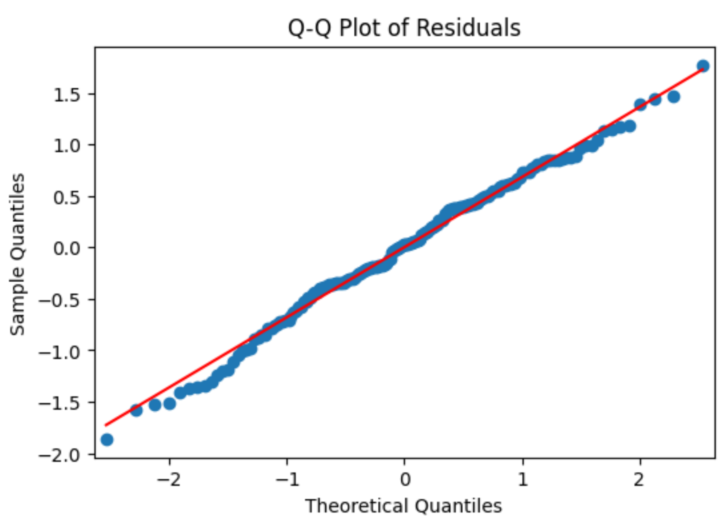
```
Text(0.5, 1.0, 'Residuals vs Fitted Values')
```



```
# Histogram of Residuals
plt.subplot(1, 2, 2)
plt.hist(residuals, bins=20, edgecolor='black')
plt.xlabel('Residuals')
plt.title('Histogram of Residuals')
plt.tight_layout()
plt.show()
```




```
# Q-Q Plot
fig, ax = plt.subplots(figsize=(6, 4))
sm.qqplot(residuals, line='s', ax=ax)
plt.title('Q-Q Plot of Residuals')
plt.show()
```



```
# Jarque-Bera Test to assess normality of residuals

# Performing the Jarque-Bera test on the model's residuals
jb_test = sm.stats.jarque_bera(residuals)

# Organizing test results into a dictionary for clarity
jb_test_results = {
    'JB statistic': jb_test[0],  # Test statistic
    'p-value': jb_test[1],       # P-value for the test
    'Skewness': jb_test[2],      # Skewness from the test
    'Kurtosis': jb_test[3]       # Kurtosis from the test
}

# Printing the test results to evaluate normality
print(jb_test_results)
```

```
{'JB statistic': 1.5211796749784372, 'p-value': 0.46739066115532457, 'Skewness': -0.20713928673279716, 'Kurtosis': 2.8170351
```