individual-assignment-226034c-1

November 9, 2024

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[3]:
```
import pandas as pd

df1 = pd.read_csv( "/content/drive/MyDrive/Colab Notebooks/wine+quality/
 ↪winequality-red.csv", delimiter=';')
df1.head()
#df.info()
```

[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

[4]:
```
df = pd.read_csv( "/content/drive/MyDrive/Colab Notebooks/wine+quality/
 ↪winequality-white.csv", delimiter=';')
df.head()
```

```
#df.info()
```

[4]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.0              0.27         0.36            20.7      0.045
1            6.3              0.30         0.34             1.6      0.049
2            8.1              0.28         0.40             6.9      0.050
3            7.2              0.23         0.32             8.5      0.058
4            7.2              0.23         0.32             8.5      0.058

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 45.0                 170.0   1.0010  3.00       0.45
1                 14.0                 132.0   0.9940  3.30       0.49
2                 30.0                  97.0   0.9951  3.26       0.44
3                 47.0                 186.0   0.9956  3.19       0.40
4                 47.0                 186.0   0.9956  3.19       0.40

   alcohol  quality
0      8.8        6
1      9.5        6
2     10.1        6
3      9.9        6
4      9.9        6
```

[5]:
```
red_wine = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/wine+quality/
 ↪winequality-red.csv', delimiter=';')
white_wine = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/wine+quality/
 ↪winequality-white.csv', delimiter=';')

red_wine['type'] = 'red'
white_wine['type'] = 'white'

# Combine the two datasets
combined_wine = pd.concat([red_wine, white_wine], ignore_index=True)
```

2. Data Exploration and Preprocessing:

[6]:
```
# Preview the combined dataset
print(combined_wine.head())
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
```

```
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality type
0      9.4        5  red
1      9.8        5  red
2      9.8        5  red
3      9.8        6  red
4      9.4        5  red
```

[7]:
```
# Check the shape and structure of the dataset
print(combined_wine.shape)
print(combined_wine.info())
```

```
(6497, 13)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         6497 non-null   float64
 1   volatile acidity      6497 non-null   float64
 2   citric acid           6497 non-null   float64
 3   residual sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free sulfur dioxide   6497 non-null   float64
 6   total sulfur dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64
 12  type                  6497 non-null   object
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
None
```

Encoding catergorical variables:

[8]:
```
from sklearn.preprocessing import LabelEncoder

# Instantiate the encoder
label_encoder = LabelEncoder()

# Apply label encoding to the 'type' column
combined_wine['type'] = label_encoder.fit_transform(combined_wine['type'])
```

```
# Display the resulting DataFrame
print(combined_wine.head())

# Assuming 'red' is encoded as 1 and 'white' as 0
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality  type
0      9.4        5     0
1      9.8        5     0
2      9.8        5     0
3      9.8        6     0
4      9.4        5     0
```

Detecting missing values:

[9]:
```
print(combined_wine.isnull().sum())
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
type                    0
dtype: int64
```

Finding Duplicate values:

[10]:
```
# finding duplicates
print(combined_wine.duplicated().sum())
```

```
1177
```

[11]:
```
combined_wine = combined_wine.drop_duplicates()
```

[12]:
```
print(combined_wine.shape)
```

```
(5320, 13)
```

Detecting and handling Outliers:

[13]:
```
import pandas as pd

# Define function to find and count outliers based on IQR
def find_and_count_outliers_iqr(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1

    # Identify outliers
    outliers = data[(data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))]

    # Return outliers and their count
    outliers_count = outliers.count()  # Counts outliers in each column
    return pd.Series([outliers, outliers_count], index=['outliers',␣
 ↪'outliers_count']) # Return a Series to make unpacking work

# Apply the function to each column
result = combined_wine.apply(find_and_count_outliers_iqr, axis=0,␣
 ↪result_type='expand') # Applying along axis=0 to each column (Series)

# Extract outliers and their counts
outliers_iqr = result.loc['outliers'] # Use .loc to access the row with label␣
 ↪"outliers"
outliers_iqr_counts = result.loc['outliers_count'] # Use .loc to access the row␣
 ↪with label "outliers_count"


# Calculate the total number of outliers across all columns
total_outliers_iqr = outliers_iqr_counts.sum()

# Display results
print("Outliers detected using IQR method:")
print(outliers_iqr)  # This will print out the individual outliers for each␣
 ↪column
print("\nOutlier counts per column:")
```

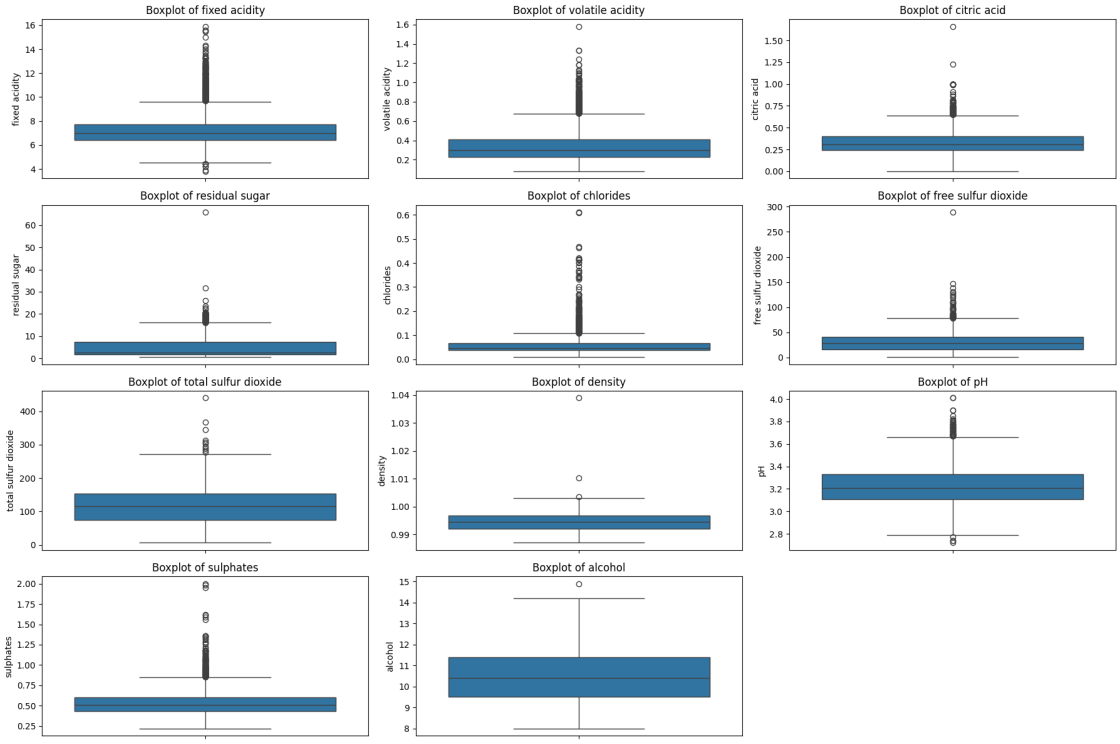
```
print(outliers_iqr_counts)  # This will print out the count of outliers for␣
  ↪each column
print("\nTotal number of outliers in dataset:", total_outliers_iqr)
```

```
Outliers detected using IQR method:
fixed acidity           3       11.2
56      10.2
74       9.7
113    …
volatile acidity        0       0.700
1       0.880
2       0.760
28  …
citric acid             81      0.70
106     0.68
151     1.00
205    …
residual sugar          1599    20.70
1613    19.25
1637    17.95
1702…
chlorides               13      0.114
14      0.176
15      0.170
17  …
free sulfur dioxide     1666     81.0
1896     82.0
1924    131.0
1986…
total sulfur dioxide    1079    278.0
1081    289.0
1924    313.0
3016…
density                 1434    1.00369
3252    1.01030
4380    1.0389…
pH                      45      3.90
94      3.75
95      3.85
142    …
sulphates               13      1.56
14      0.88
15      0.93
17     …
alcohol                         652    14.9
Name: alcohol, dtype: float64
quality                 267     8
```

```
278     8
390     8
440     8
455   …
type                                  Series([], Name: type, dtype: int64)
Name: outliers, dtype: object

Outlier counts per column:
fixed acidity           304
volatile acidity        279
citric acid             143
residual sugar          141
chlorides               237
free sulfur dioxide      44
total sulfur dioxide     10
density                   3
pH                       49
sulphates               163
alcohol                   1
quality                 183
type                      0
Name: outliers_count, dtype: object

Total number of outliers in dataset: 1557
```

[14]:
```python
# Visualizing potential outliers

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


# Assuming combined_wine is your DataFrame containing the wine data
# Exclude the target column 'quality' and 'type_white' (if not considered a␣
 ↪feature) when plotting
features = combined_wine.drop(columns=['quality', 'type'])

# Set up the plotting area with enough room for 12 box plots (e.g., 4 rows and␣
 ↪3 columns)
plt.figure(figsize=(18, 12))  # Adjust figure size for better spacing

# Loop through each feature and draw a box plot
for i, column in enumerate(features.columns, 1):
    plt.subplot(4, 3, i)  # Create a 4x3 grid for 12 features
    sns.boxplot(data=combined_wine, y=column)
    plt.title(f'Boxplot of {column}')
```

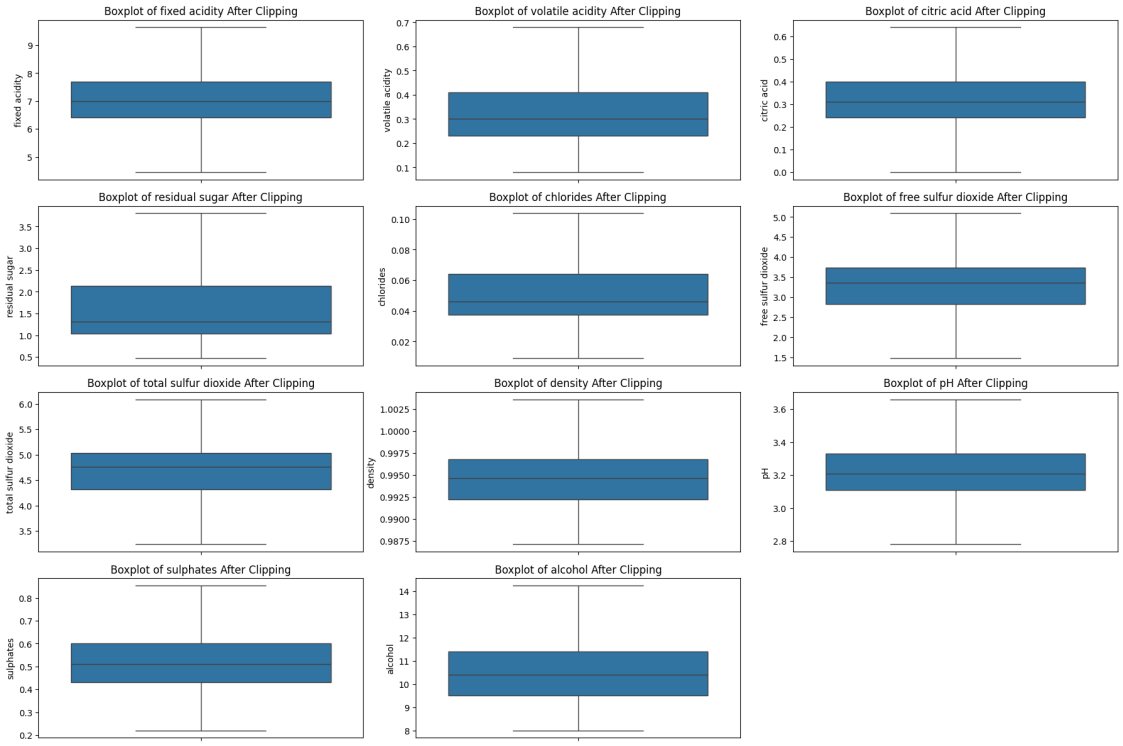
```
plt.tight_layout()  # Adjust spacing
plt.show()
```



[15]:
```
#Handling outliers
import numpy as np

# Assuming combined_wine is your DataFrame containing the wine data
features = combined_wine.drop(columns=['quality', 'type'])  # Exclude the␣
 ↪categorical columns

# Function to calculate whiskers with a flexible multiplier
def whisker_bounds(col, multiplier=1.5):
    q1, q3 = np.percentile(col, [25, 75])
    iqr = q3 - q1
    lower_bound = q1 - (multiplier * iqr)
    upper_bound = q3 + (multiplier * iqr)
    return lower_bound, upper_bound

# Log-transform for skewed features
log_transform_cols = ['residual sugar', 'chlorides', 'free sulfur dioxide',␣
 ↪'total sulfur dioxide']

# Apply log transformation to skewed features
```

```
for col in log_transform_cols:
    combined_wine[col] = combined_wine[col].apply(lambda x: np.log1p(x))  #␣
 ↪log1p handles zero values better than log

# Outlier treatment using clipping instead of capping
for col in features.columns:
    # Calculate bounds using whisker method
    lower_bound, upper_bound = whisker_bounds(combined_wine[col], multiplier=1.
 ↪5)

    # Clip values at lower and upper bounds
    combined_wine[col] = combined_wine[col].clip(lower_bound, upper_bound)
```

[16]:
```
# Set up the plotting area for box plots after outlier treatment
plt.figure(figsize=(18, 12))  # Adjust figure size for better spacing

# Loop through each feature and draw a box plot after outlier treatment
for i, column in enumerate(features.columns, 1):
    plt.subplot(4, 3, i)  # Create a 4x3 grid for 12 features
    sns.boxplot(data=combined_wine, y=column)
    plt.title(f'Boxplot of {column} After Clipping')

plt.tight_layout()  # Adjust spacing
plt.show()
```



[17]:
```
#Scaling Numerical Features:
from sklearn.preprocessing import StandardScaler

# Define the features to scale (excluding the target and already-encoded␣
 ↪columns)
features_to_scale = ['fixed acidity', 'volatile acidity', 'citric acid',␣
 ↪'residual sugar', 'chlorides',
                     'free sulfur dioxide', 'total sulfur dioxide', 'density',␣
 ↪'pH', 'sulphates']

# Initialize the scaler
scaler = StandardScaler()

# Scale the features
combined_wine[features_to_scale] = scaler.
 ↪fit_transform(combined_wine[features_to_scale])

# Preview the scaled data
print(combined_wine.head())
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0       0.238055          2.298926    -2.273272       -0.762376   1.018768
1       0.605077          2.298926    -2.273272       -0.434549   1.992247
2       0.605077          2.298926    -1.985367       -0.566472   1.728701
3       2.302554         -0.386644     1.757397       -0.762376   0.974047
5       0.238055          2.164647    -2.273272       -0.815580   0.974047

   free sulfur dioxide  total sulfur dioxide   density        pH  sulphates  \
0            -1.166466             -1.663076  1.128801  1.812423   0.240980
1             0.013476             -0.598280  0.783878 -0.151665   1.156786
2            -0.727443             -0.938443  0.852863  0.228481   0.927835
3            -0.547697             -0.772444  1.197786 -0.405096   0.393614
5            -0.931221             -1.409407  1.128801  1.812423   0.240980

   alcohol  quality  type
0      9.4        5     0
1      9.8        5     0
2      9.8        5     0
3      9.8        6     0
5      9.4        5     0
```

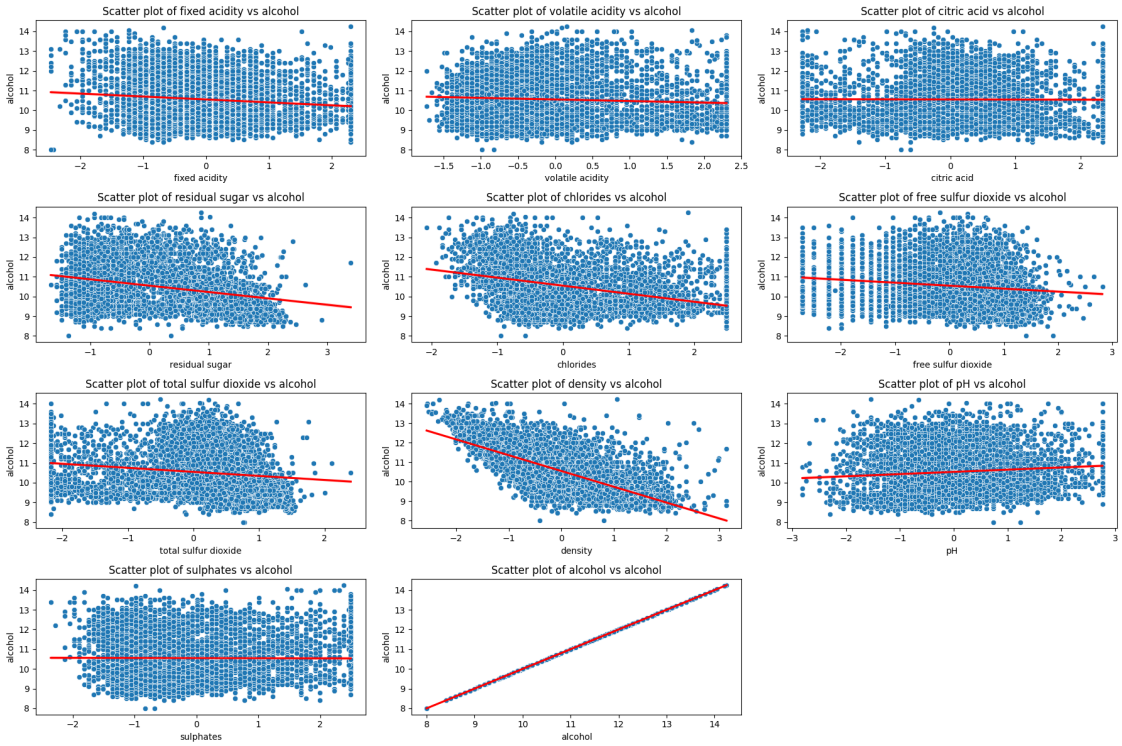
Exploratory Data Analysis (EDA)

[18]:
```
# Summary statistics (mean, std, min, max, etc.)
print(combined_wine.describe())
```

```
       fixed acidity  volatile acidity   citric acid  residual sugar  \
count   5.320000e+03      5.320000e+03  5.320000e+03    5.320000e+03
mean   -8.547883e-17      8.547883e-17 -8.547883e-17   -8.547883e-17
std     1.000094e+00      1.000094e+00  1.000094e+00    1.000094e+00
min    -2.468731e+00     -1.729429e+00 -2.273272e+00   -1.664043e+00
25%    -6.794994e-01     -7.223400e-01 -5.458422e-01   -8.155802e-01
50%    -1.289665e-01     -2.523653e-01 -4.200868e-02   -3.930082e-01
75%     5.133219e-01      4.861664e-01  6.057773e-01    8.680263e-01
max     2.302554e+00      2.298926e+00  2.333207e+00    3.393436e+00

          chlorides  free sulfur dioxide  total sulfur dioxide       density  \
count  5.320000e+03         5.320000e+03          5.320000e+03  5.320000e+03
mean  -8.547883e-17         1.282182e-16          8.975277e-16  2.863541e-15
std    1.000094e+00         1.000094e+00          1.000094e+00  1.000094e+00
min   -2.073430e+00        -2.705321e+00         -2.175280e+00 -2.558431e+00
25%   -7.105481e-01        -6.349252e-01         -4.411950e-01 -8.027702e-01
50%   -2.953187e-01         1.801223e-01          2.717356e-01  4.229235e-02
75%    5.696785e-01         7.453387e-01          7.148614e-01  7.735301e-01
max    2.490018e+00         2.815735e+00          2.399006e+00  3.137981e+00

                 pH     sulphates      alcohol      quality         type
count  5.320000e+03  5.320000e+03  5320.000000  5320.000000  5320.000000
mean   1.132594e-15 -2.778062e-16    10.549119     5.795677     0.744549
std    1.000094e+00  1.000094e+00     1.185518     0.879772     0.436155
min   -2.812689e+00 -2.353804e+00     8.000000     3.000000     0.000000
25%   -7.218847e-01 -7.511432e-01     9.500000     5.000000     0.000000
50%   -8.830777e-02 -1.406058e-01    10.400000     6.000000     1.000000
75%    6.719846e-01  5.462488e-01    11.400000     6.000000     1.000000
max    2.762789e+00  2.492337e+00    14.250000     9.000000     1.000000
```

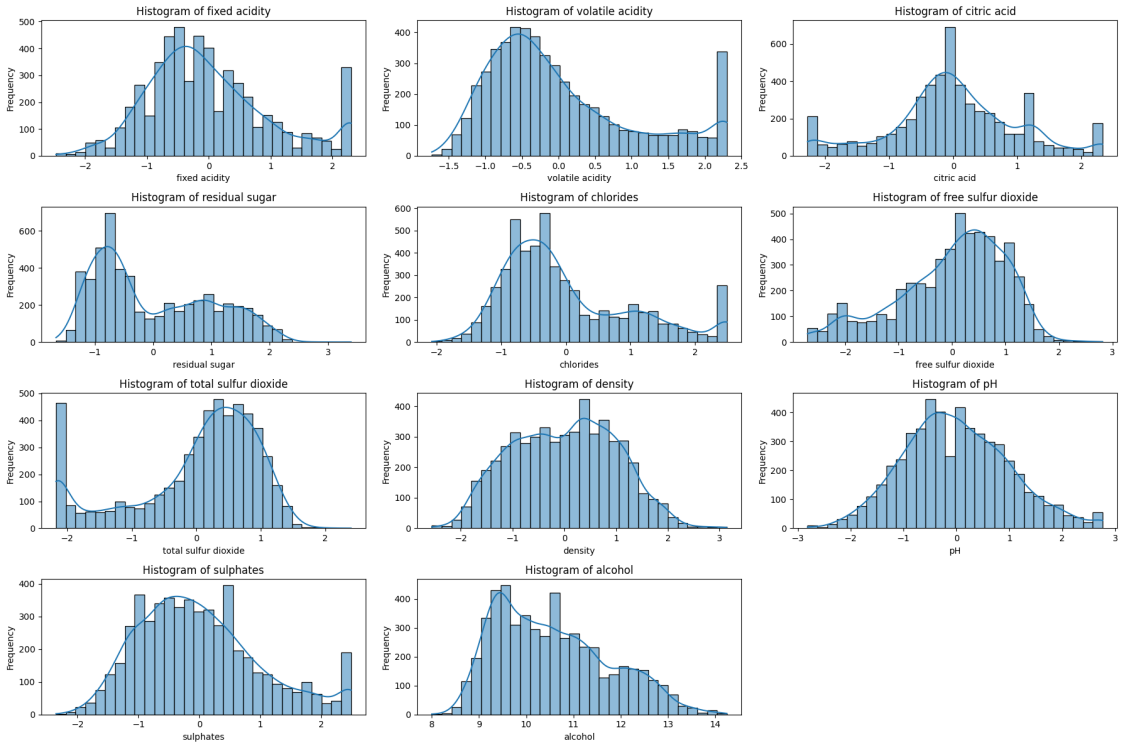
[19]:
```python
# Scatterplot -> identify relationships

# Assuming combined_wine is your DataFrame containing the wine data
features = combined_wine.drop(columns=['quality', 'type'])  # Exclude the␣
 ↪categorical columns

# Set up the plotting area with enough room for 12 scatter plots (e.g., 4 rows␣
 ↪and 3 columns)
plt.figure(figsize=(18, 12))  # Adjust figure size for better spacing

# Loop through each feature and draw a scatter plot against 'alcohol'
for i, column in enumerate(features.columns, 1):
    plt.subplot(4, 3, i)  # Create a 4x3 grid for 12 features
    sns.scatterplot(data=combined_wine, x=column, y='alcohol')
    sns.regplot(data=combined_wine, x=column, y='alcohol', scatter=False,␣
 ↪color='red')  # Add a regression line
    plt.title(f'Scatter plot of {column} vs alcohol')
```

```
    plt.xlabel(column)
    plt.ylabel('alcohol')

plt.tight_layout()  # Adjust spacing
plt.show()
```

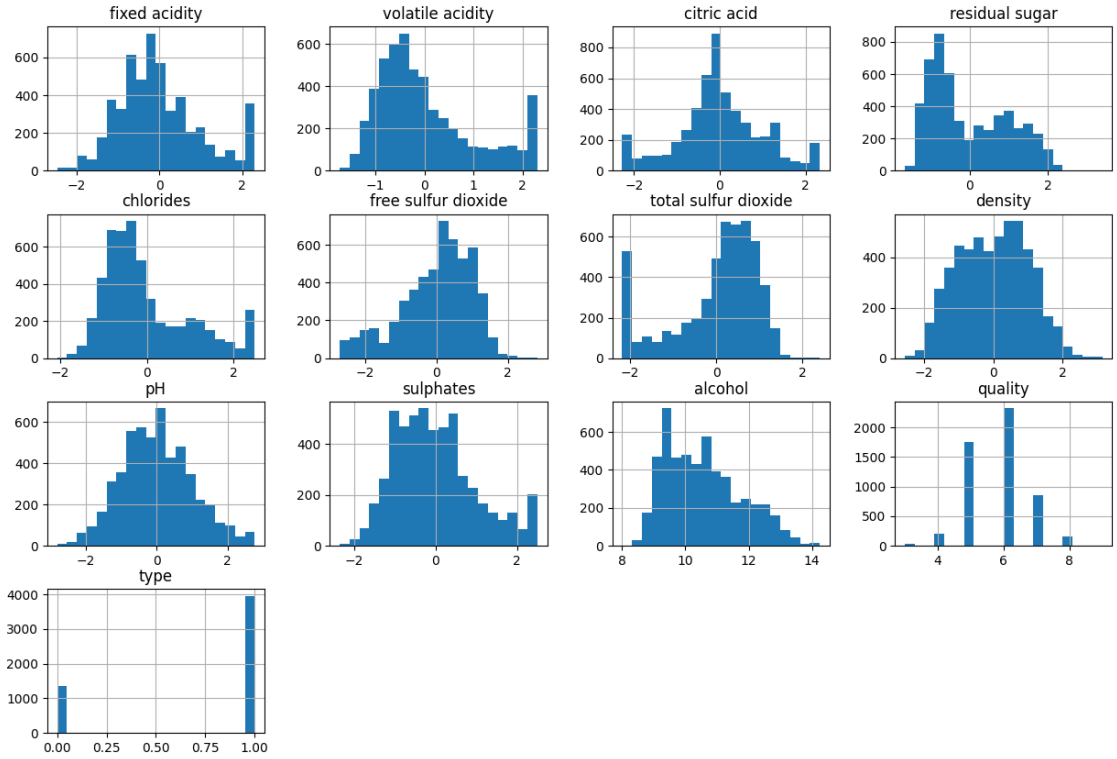


[20]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming combined_wine is your DataFrame containing the wine data
features = combined_wine.drop(columns=['quality', 'type'])  # Exclude the␣
 ↪categorical columns

# Set up the plotting area with enough room for histograms
plt.figure(figsize=(18, 12))  # Adjust figure size for better spacing

# Loop through each feature and draw a histogram
for i, column in enumerate(features.columns, 1):
    plt.subplot(4, 3, i)  # Create a 4x3 grid for 12 features
    sns.histplot(data=combined_wine, x=column, bins=30, kde=True)  # Adding␣
 ↪kde=True for kernel density estimate
    plt.title(f'Histogram of {column}')
```

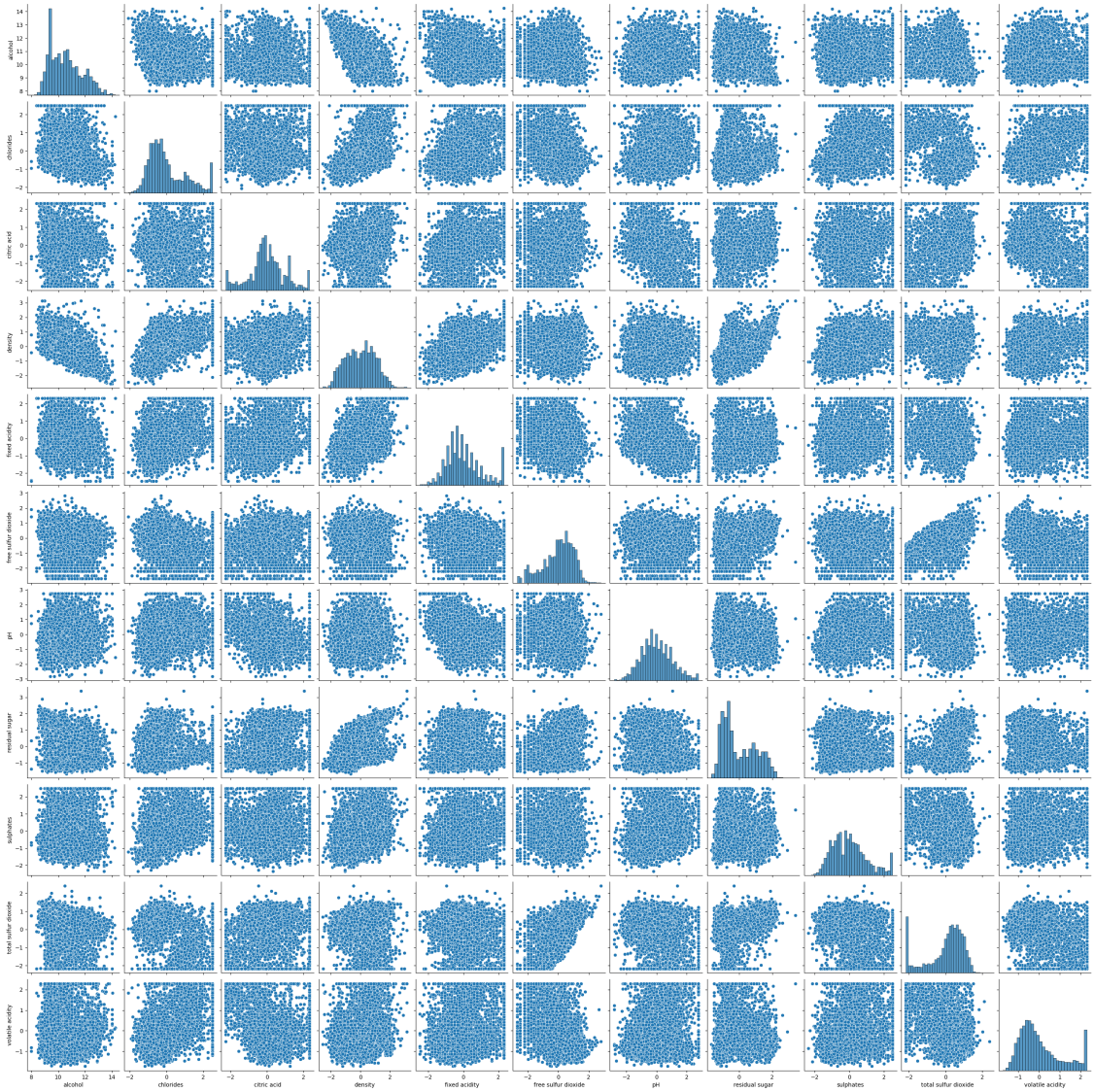
```
    plt.xlabel(column)
    plt.ylabel('Frequency')

plt.tight_layout()  # Adjust spacing
plt.show()
```



[21]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot histograms for all numerical features
combined_wine.hist(bins=20, figsize=(15, 10))
plt.show()
```



[22]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Select all quantitative variables except 'quality' and 'type'
quantitative_vars = combined_wine.select_dtypes(include=['float64', 'int64']).
 ↪columns
quantitative_vars = quantitative_vars.difference(['quality', 'type'])  #␣
 ↪Exclude 'quality' and 'type'

# Plot pairplot excluding 'quality' and 'type'
sns.pairplot(combined_wine[quantitative_vars])
plt.show()
```



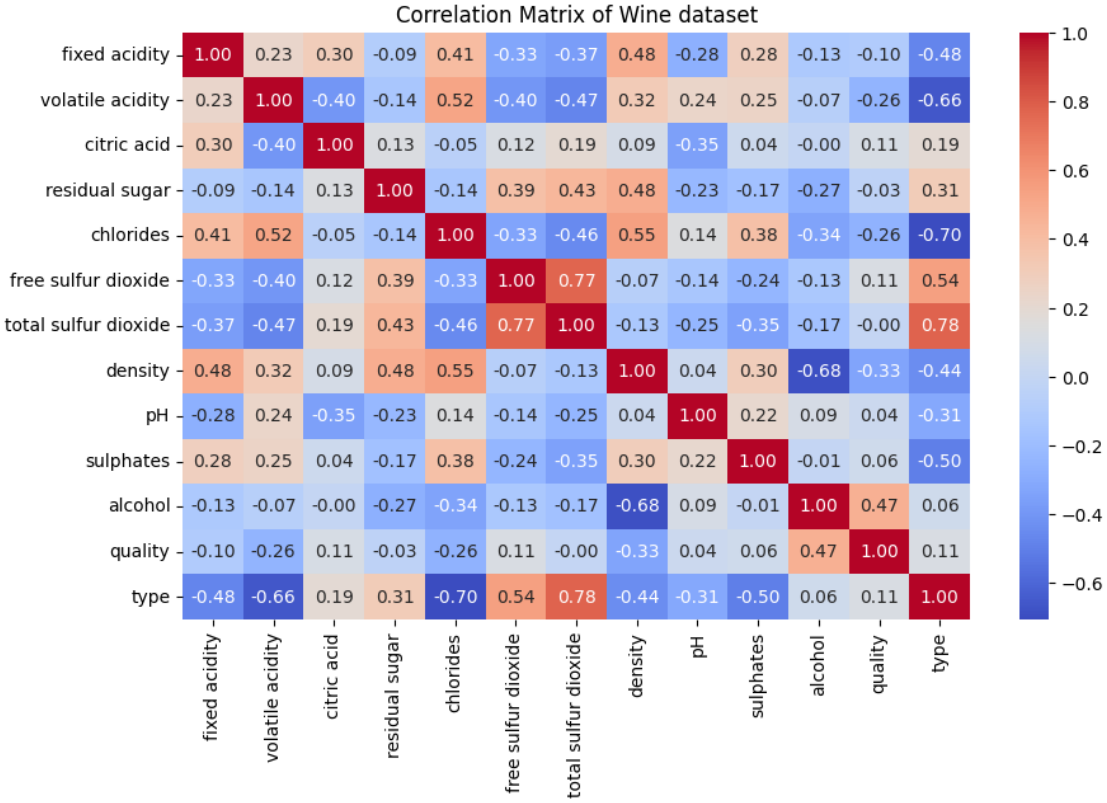
[23]:
```
#3. Identify Potential Correlations Between Variables
c = combined_wine.corr()
print(c)
```

```
                      fixed acidity  volatile acidity  citric acid  \
fixed acidity              1.000000          0.231671     0.304151
volatile acidity           0.231671          1.000000    -0.400287
citric acid                0.304151         -0.400287     1.000000
residual sugar            -0.088648         -0.142986     0.133630
chlorides                  0.410064          0.522757    -0.054785
free sulfur dioxide       -0.326909         -0.401822     0.120446
total sulfur dioxide      -0.367892         -0.472642     0.194845
density                    0.480879          0.318331     0.090624
```

```
pH                      -0.278140          0.236778    -0.352076
sulphates                0.284874          0.253390     0.041331
alcohol                 -0.125291         -0.067508    -0.004796
quality                 -0.099867         -0.257580     0.106895
type                    -0.476049         -0.661813     0.192035

                     residual sugar  chlorides  free sulfur dioxide  \
fixed acidity             -0.088648   0.410064            -0.326909
volatile acidity          -0.142986   0.522757            -0.401822
citric acid                0.133630  -0.054785             0.120446
residual sugar             1.000000  -0.144489             0.385307
chlorides                 -0.144489   1.000000            -0.332358
free sulfur dioxide        0.385307  -0.332358             1.000000
total sulfur dioxide       0.434263  -0.458410             0.768590
density                    0.482556   0.545236            -0.070994
pH                        -0.226424   0.137349            -0.142158
sulphates                 -0.174867   0.380892            -0.240364
alcohol                   -0.272488  -0.344267            -0.127454
quality                   -0.034217  -0.261244             0.114800
type                       0.311912  -0.704897             0.540486

                     total sulfur dioxide   density        pH  sulphates  \
fixed acidity                  -0.367892  0.480879 -0.278140   0.284874
volatile acidity               -0.472642  0.318331  0.236778   0.253390
citric acid                     0.194845  0.090624 -0.352076   0.041331
residual sugar                  0.434263  0.482556 -0.226424  -0.174867
chlorides                      -0.458410  0.545236  0.137349   0.380892
free sulfur dioxide             0.768590 -0.070994 -0.142158  -0.240364
total sulfur dioxide            1.000000 -0.126646 -0.245149  -0.351216
density                        -0.126646  1.000000  0.035142   0.297240
pH                             -0.245149  0.035142  1.000000   0.218223
sulphates                      -0.351216  0.297240  0.218223   1.000000
alcohol                        -0.173329 -0.684951  0.094422  -0.005131
quality                        -0.001768 -0.334503  0.040304   0.055252
type                            0.777086 -0.440746 -0.312533  -0.504576

                      alcohol   quality      type
fixed acidity       -0.125291 -0.099867 -0.476049
volatile acidity    -0.067508 -0.257580 -0.661813
citric acid         -0.004796  0.106895  0.192035
residual sugar      -0.272488 -0.034217  0.311912
chlorides           -0.344267 -0.261244 -0.704897
free sulfur dioxide -0.127454  0.114800  0.540486
total sulfur dioxide -0.173329 -0.001768  0.777086
density             -0.684951 -0.334503 -0.440746
pH                   0.094422  0.040304 -0.312533
sulphates           -0.005131  0.055252 -0.504576
alcohol              1.000000  0.469679  0.057952
```

```
quality                0.469679   1.000000   0.114809
type                   0.057952   0.114809   1.000000
```

[24]:
```
# Visualize correlations among numerical features using a heatmap
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.heatmap(combined_wine.corr(numeric_only=True), annot=True, cmap='coolwarm',␣
 ↪fmt='.2f')
plt.title('Correlation Matrix of Wine dataset')
plt.show()
```



[24]:
```
```

Model Selection, Training and Evaluation:

1. Linear Regression:

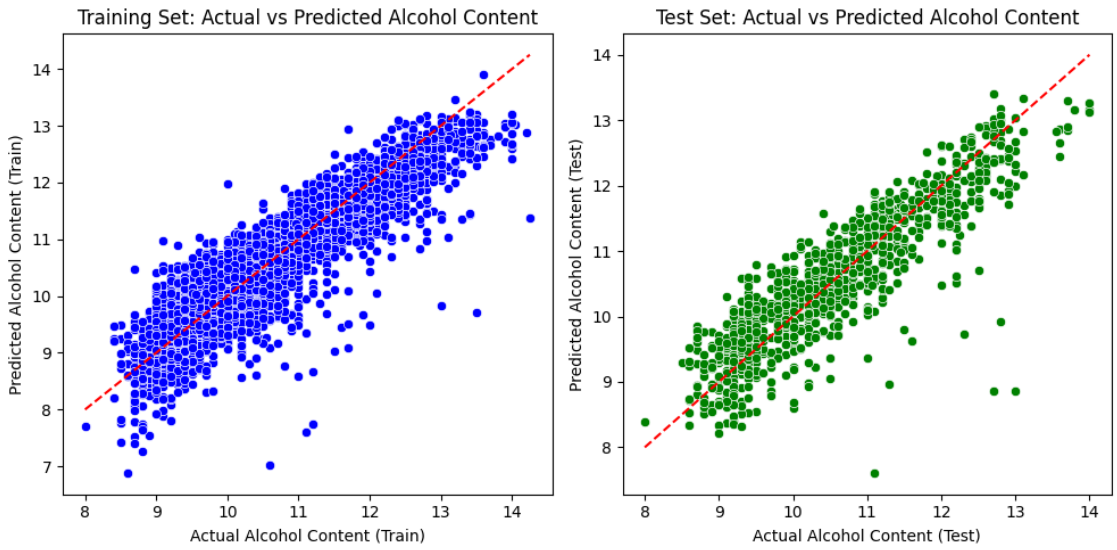
[34]:
```
# Import Required Libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```
from sklearn.metrics import mean_absolute_error


# Preprocess the data
X = combined_wine.drop('alcohol', axis=1)  # Replace 'alcohol' with your target␣
 ↪variable
y = combined_wine['alcohol']  # Set target variable

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize and train Linear Regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# Make predictions
y_pred_linear = linear_model.predict(X_test)
y_pred_linear

# Make predictions on training set
y_train_pred = linear_model.predict(X_train)

# Print performance metrics for Linear Regression on Training Set
print("Linear Regression Performance on Training Set:")
print("RMSE:", mean_squared_error(y_train, y_train_pred, squared=False))
print("Mean Absolute Error (MAE):", mean_absolute_error(y_train, y_train_pred))
print("R²:", r2_score(y_train, y_train_pred))
print("\n")  # Separate outputs for clarity

# Print performance metrics for Linear Regression on Test Set
print("Linear Regression Performance on Test Set:")
print("RMSE:", mean_squared_error(y_test, y_pred_linear, squared=False))
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred_linear))
print("R²:", r2_score(y_test, y_pred_linear))

# Visualization for Training Set: Actual vs Predicted values
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)  # Two plots side by side
sns.scatterplot(x=y_train, y=y_train_pred, color='blue')
plt.plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()], 'r--')
plt.xlabel('Actual Alcohol Content (Train)')
plt.ylabel('Predicted Alcohol Content (Train)')
plt.title('Training Set: Actual vs Predicted Alcohol Content')

# Visualization for Test Set: Actual vs Predicted values
plt.subplot(1, 2, 2)
```

```
sns.scatterplot(x=y_test, y=y_pred_linear, color='green')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Actual Alcohol Content (Test)')
plt.ylabel('Predicted Alcohol Content (Test)')
plt.title('Test Set: Actual vs Predicted Alcohol Content')

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

```
Linear Regression Performance on Training Set:
RMSE: 0.5164709821590663
Mean Absolute Error (MAE): 0.3934460349589825
R²: 0.8115484541539659


Linear Regression Performance on Test Set:
RMSE: 0.542252956960322
Mean Absolute Error (MAE): 0.4008774114853246
R²: 0.7844127317761557
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:492:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:492:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
  warnings.warn(
```

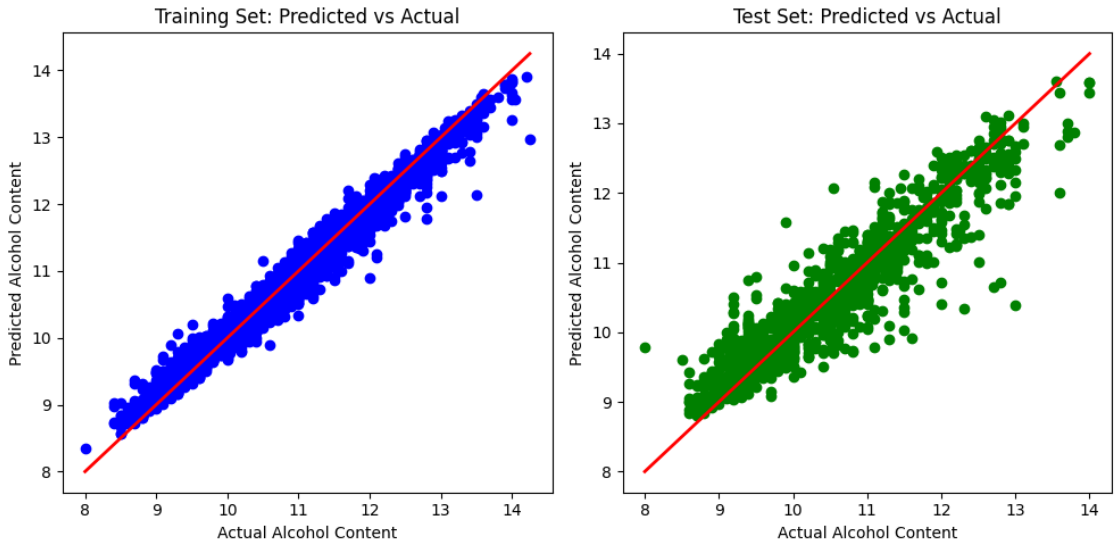


2. Random Forest Regression:

[38]:
```
# Import Required Libraries
from sklearn.ensemble import RandomForestRegressor

# Preprocess the data
X = combined_wine.drop('alcohol', axis=1)  # Replace 'alcohol' with your target␣
 ↪variable
y = combined_wine['alcohol']  # Set target variable

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize and train Random Forest Regression model
random_forest_regressor = RandomForestRegressor(n_estimators=100,␣
 ↪random_state=42)
random_forest_regressor.fit(X_train, y_train)

# Make predictions
y_test_pred_rf = random_forest_regressor.predict(X_test)
y_test_pred_rf

# Make predictions on training set
y_train_pred_rf = random_forest_regressor.predict(X_train)

# Print performance metrics for Random Forest on Training Set
print("Random Forest Performance on Training Set:")
print("RMSE:", mean_squared_error(y_train, y_train_pred_rf, squared=False))
print("Mean Absolute Error (MAE):", mean_absolute_error(y_train,␣
 ↪y_train_pred_rf))
print("R²:", r2_score(y_train, y_train_pred_rf))
print("\n")  # Separate outputs for clarity

# Print performance metrics for Random Forest on Test Set
print("Random Forest Performance on Test Set:")
print("RMSE:", mean_squared_error(y_test, y_test_pred_rf, squared=False))
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_test_pred_rf))
print("R²:", r2_score(y_test, y_test_pred_rf))

# Plotting Predicted vs Actual Values

# Training Set
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(y_train, y_train_pred_rf, color='blue')
```

```
plt.plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()],␣
 ↪color='red', linewidth=2)
plt.title('Training Set: Predicted vs Actual')
plt.xlabel('Actual Alcohol Content')
plt.ylabel('Predicted Alcohol Content')

# Test Set
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_test_pred_rf, color='green')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],␣
 ↪color='red', linewidth=2)
plt.title('Test Set: Predicted vs Actual')
plt.xlabel('Actual Alcohol Content')
plt.ylabel('Predicted Alcohol Content')

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

```
Random Forest Performance on Training Set:
RMSE: 0.1670905436197114
Mean Absolute Error (MAE): 0.12127183695846756
R²: 0.9802752459491327


Random Forest Performance on Test Set:
RMSE: 0.44258546670243815
Mean Absolute Error (MAE): 0.32309046276405307
R²: 0.8563804541294884
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:492:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:492:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
  warnings.warn(
```



3. Gradient Boosting Regression:

[39]:
```
# Import Required Libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Preprocess the data
X = combined_wine.drop('alcohol', axis=1)  # Replace 'alcohol' with your target␣
 ↪variable
y = combined_wine['alcohol']  # Set target variable

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize and train Gradient Boosting Regression model
gb_model = GradientBoostingRegressor(n_estimators=100, random_state=42)  #␣
 ↪Adjust n_estimators as needed
gb_model.fit(X_train, y_train)

# Make predictions on training set
y_train_pred_gb = gb_model.predict(X_train)

# Make predictions on test set
y_test_pred_gb = gb_model.predict(X_test)

# Print performance metrics for Gradient Boosting on Training Set
print("Gradient Boosting Performance on Training Set:")
```

```
print("RMSE:", mean_squared_error(y_train, y_train_pred_gb, squared=False))
print("Mean Absolute Error (MAE):", mean_absolute_error(y_train,␣
 ↪y_train_pred_gb))
print("R²:", r2_score(y_train, y_train_pred_gb))
print("\n")  # Separate outputs for clarity

# Print performance metrics for Gradient Boosting on Test Set
print("Gradient Boosting Performance on Test Set:")
print("RMSE:", mean_squared_error(y_test, y_test_pred_gb, squared=False))
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_test_pred_gb))
print("R²:", r2_score(y_test, y_test_pred_gb))
```

```
Gradient Boosting Performance on Training Set:
RMSE: 0.4280786758911519
Mean Absolute Error (MAE): 0.32789733029706836
R²: 0.8705342051564866


Gradient Boosting Performance on Test Set:
RMSE: 0.480437555171899
Mean Absolute Error (MAE): 0.36212517687160145
R²: 0.8307638456103524
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:492:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:492:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
  warnings.warn(
```

4. Support Vector Regression:

[28]:
```
# Import Required Libraries
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Preprocess the data
X = combined_wine.drop('alcohol', axis=1)  # Replace 'alcohol' with your target␣
 ↪variable
y = combined_wine['alcohol']  # Set target variable

# Split the data into training and test sets
```

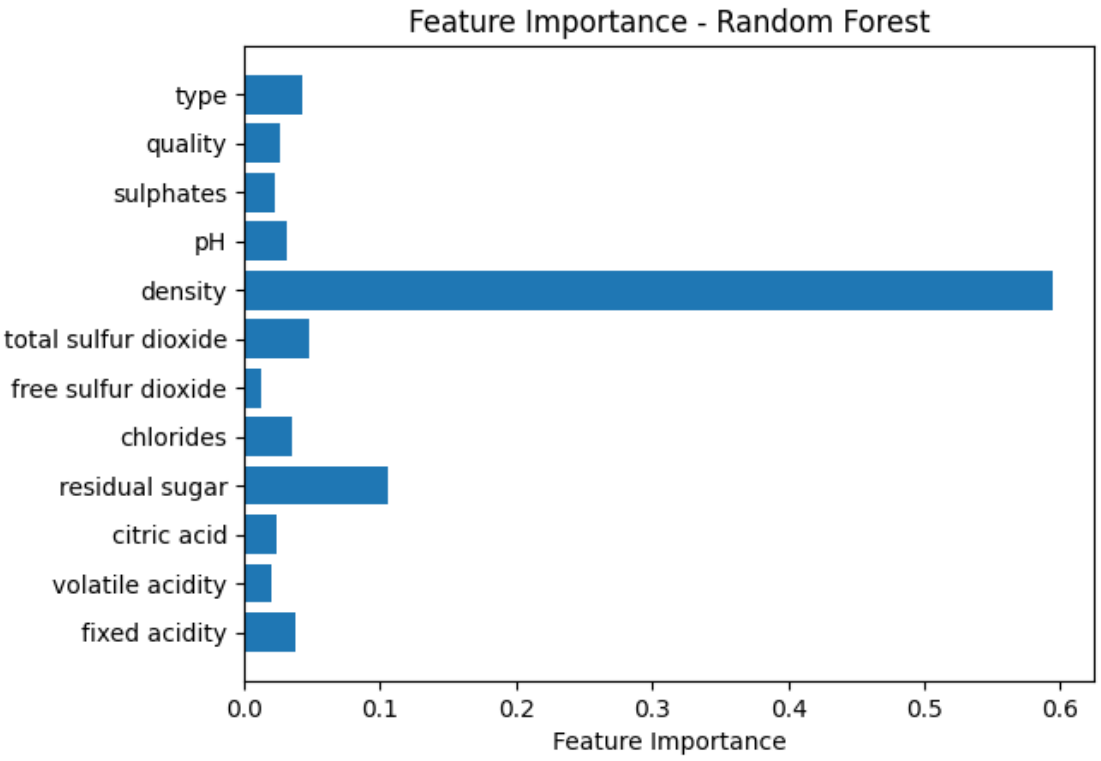
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize and train Support Vector Regression model
svr_model = SVR(kernel='rbf')  # Adjust the kernel type and parameters as needed
svr_model.fit(X_train, y_train)

# Make predictions on training set
y_train_pred_svr = svr_model.predict(X_train)

# Make predictions on test set
y_test_pred_svr = svr_model.predict(X_test)

# Print performance metrics for SVR on Training Set
print("Support Vector Regression Performance on Training Set:")
print("RMSE:", mean_squared_error(y_train, y_train_pred_svr, squared=False))
print("Mean Absolute Error (MAE):", mean_absolute_error(y_train,␣
 ↪y_train_pred_svr))
print("R²:", r2_score(y_train, y_train_pred_svr))
print("\n")  # Separate outputs for clarity

# Print performance metrics for SVR on Test Set
print("Support Vector Regression Performance on Test Set:")
print("RMSE:", mean_squared_error(y_test, y_test_pred_svr, squared=False))
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test,␣
 ↪y_test_pred_svr))
print("R²:", r2_score(y_test, y_test_pred_svr))
```

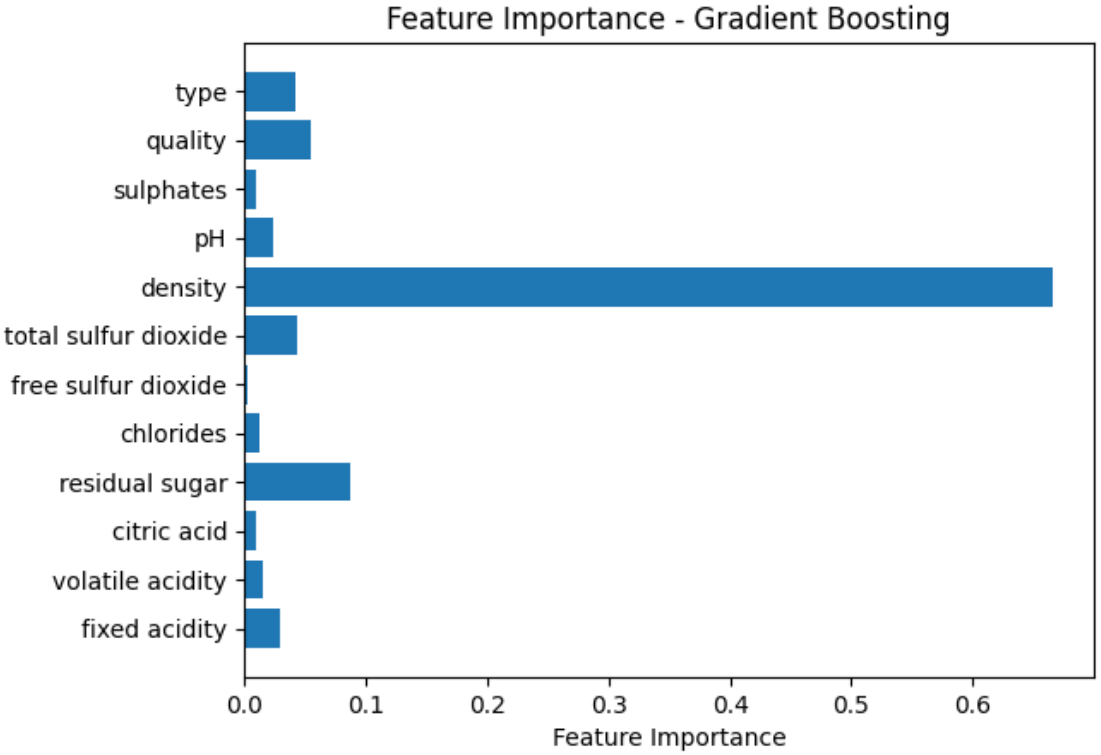
```
Support Vector Regression Performance on Training Set:
RMSE: 0.4056609763210339
Mean Absolute Error (MAE): 0.2982220529144393
R²: 0.8837389299477857


Support Vector Regression Performance on Test Set:
RMSE: 0.439526509328279
Mean Absolute Error (MAE): 0.3175648653674055
R²: 0.8583588645696486

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:492:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:492:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
```

```
  warnings.warn(
```

Feature Importance:

[29]:
```
# For Random Forest
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt

# Fit the model
model = RandomForestRegressor()
model.fit(X_train, y_train)

# Get feature importance
importances = model.feature_importances_
# Assuming X_train is your input data (with features) and you have a DataFrame
feature_names = X_train.columns  # If you're using a DataFrame like pandas

# Visualize the feature importance
plt.barh(range(len(importances)), importances)
plt.yticks(range(len(importances)), feature_names)
plt.xlabel('Feature Importance')
plt.title('Feature Importance - Random Forest')
plt.show()

# For Gradient Boosting
from sklearn.ensemble import GradientBoostingRegressor

# Fit the model
model_gb = GradientBoostingRegressor()
model_gb.fit(X_train, y_train)

# Get feature importance
importances_gb = model_gb.feature_importances_

# Visualize the feature importance
plt.barh(range(len(importances_gb)), importances_gb)
plt.yticks(range(len(importances_gb)), feature_names)
plt.xlabel('Feature Importance')
plt.title('Feature Importance - Gradient Boosting')
plt.show()
```


Feature Importance - Random Forest
type
quality
sulphates
pH
density
total sulfur dioxide
free sulfur dioxide
chlorides
residual sugar
citric acid
volatile acidity
fixed acidity
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Feature Importance


Feature Importance - Gradient Boosting
type
quality
sulphates
pH
density
total sulfur dioxide
free sulfur dioxide
chlorides
residual sugar
citric acid
volatile acidity
fixed acidity
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Feature Importance

Hyperparameter Tuning

For Random Forest Regression:

[30]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Define the parameter distribution
param_dist = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'max_features': ['sqrt', 'log2'],  # Use only supported values here
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize the RandomForestRegressor
rf = RandomForestRegressor(random_state=42)

# Initialize RandomizedSearchCV with the parameter distribution
random_search = RandomizedSearchCV(estimator=rf, param_distributions=param_dist,
                                   n_iter=100, cv=5,␣
 ↪scoring='neg_mean_squared_error',
                                   verbose=2, n_jobs=-1, random_state=42)

# Fit to the training data
random_search.fit(X_train, y_train)

# Get the best parameters and evaluate the tuned model
best_params = random_search.best_params_
print("Best Parameters:", best_params)

# Predictions and performance metrics on the test set
y_pred_rf = random_search.best_estimator_.predict(X_test)
print("RMSE:", mean_squared_error(y_test, y_pred_rf, squared=False))
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred_rf))
print("R²:", r2_score(y_test, y_pred_rf))
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits

/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning:
invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,

Best Parameters: {'n_estimators': 300, 'min_samples_split': 2,
'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': None}
```

```
RMSE: 0.506314859770606
Mean Absolute Error (MAE): 0.38151204524207377
R²: 0.8120420936363373

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:492:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
  warnings.warn(
```

Gradient Boosting Regression

[31]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Define the parameter distribution
param_dist = {
    'n_estimators': [100, 200, 300],  # Number of boosting stages to be run
    'learning_rate': [0.01, 0.1, 0.2, 0.3],  # Step size shrinkage to prevent␣
 ↪overfitting
    'max_depth': [3, 5, 7, 9],  # Maximum depth of the individual regression␣
 ↪estimators
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to␣
 ↪split an internal node
    'min_samples_leaf': [1, 2, 4]  # Minimum number of samples required to be␣
 ↪at a leaf node
}

# Initialize the GradientBoostingRegressor
gb = GradientBoostingRegressor(random_state=42)

# Initialize RandomizedSearchCV with the parameter distribution
random_search = RandomizedSearchCV(estimator=gb, param_distributions=param_dist,
                                   n_iter=100, cv=5,␣
 ↪scoring='neg_mean_squared_error',
                                   verbose=2, n_jobs=-1, random_state=42)

# Fit to the training data
random_search.fit(X_train, y_train)

# Get the best parameters and evaluate the tuned model
best_params = random_search.best_params_
print("Best Parameters:", best_params)

# Predictions and performance metrics on the test set
y_pred_gb = random_search.best_estimator_.predict(X_test)
```

```
print("RMSE:", mean_squared_error(y_test, y_pred_gb, squared=False))
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred_gb))
print("R²:", r2_score(y_test, y_pred_gb))
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best Parameters: {'n_estimators': 300, 'min_samples_split': 5,
'min_samples_leaf': 1, 'max_depth': 5, 'learning_rate': 0.1}
RMSE: 0.4137814520822752
Mean Absolute Error (MAE): 0.29959894485973176
R²: 0.8744660222818823
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:492:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
  warnings.warn(
```

Support Vector Regression:

[32]:
```
from sklearn.svm import SVR
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Define a reduced parameter distribution to reduce execution time
param_dist = {
    'kernel': ['linear', 'rbf'],  # Limiting to fewer kernel types
    'C': [0.1, 1, 10],  # Regularization parameter
    'epsilon': [0.01, 0.1],  # Epsilon in the epsilon-SVR model
    'gamma': ['scale']  # Use only 'scale' for gamma
}

# Initialize the SVR model
svr = SVR()

# Initialize RandomizedSearchCV with the parameter distribution
random_search = RandomizedSearchCV(estimator=svr,␣
 ↪param_distributions=param_dist,
                                   n_iter=20,  # Reduced number of iterations␣
 ↪for faster execution
                                   cv=5, scoring='neg_mean_squared_error',
                                   verbose=2, n_jobs=-1, random_state=42)

# Fit to the training data
random_search.fit(X_train, y_train)

# Get the best parameters and evaluate the tuned model
best_params = random_search.best_params_
```

```python
print("Best Parameters:", best_params)

# Predictions and performance metrics on the test set
y_pred_svr = random_search.best_estimator_.predict(X_test)
print("RMSE:", mean_squared_error(y_test, y_pred_svr, squared=False))
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred_svr))
print("R²:", r2_score(y_test, y_pred_svr))
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:320:
UserWarning: The total space of parameters 12 is smaller than n_iter=20. Running
12 iterations. For exhaustive searches, use GridSearchCV.
  warnings.warn(

Best Parameters: {'kernel': 'rbf', 'gamma': 'scale', 'epsilon': 0.1, 'C': 10}
RMSE: 0.40899096584026456
Mean Absolute Error (MAE): 0.2975519709970566
R²: 0.8773558940552287

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:492:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
  warnings.warn(
```