In [13]:
```
# Import dependencies
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2

from warnings import filterwarnings
filterwarnings('ignore')
```

In [6]:
```
df= pd.read_csv(r"C:\Users\M_Ampah\Downloads\winequality-red.csv")
```

In [7]:
```
#shows First 5 rows and last 5 rows
df
```

Out[7]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56
...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66

1599 rows × 12 columns

In [8]:
```
df.shape
```

Out[8]: (1599, 12)

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [15]:
```
df.describe()
```

Out[15]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000

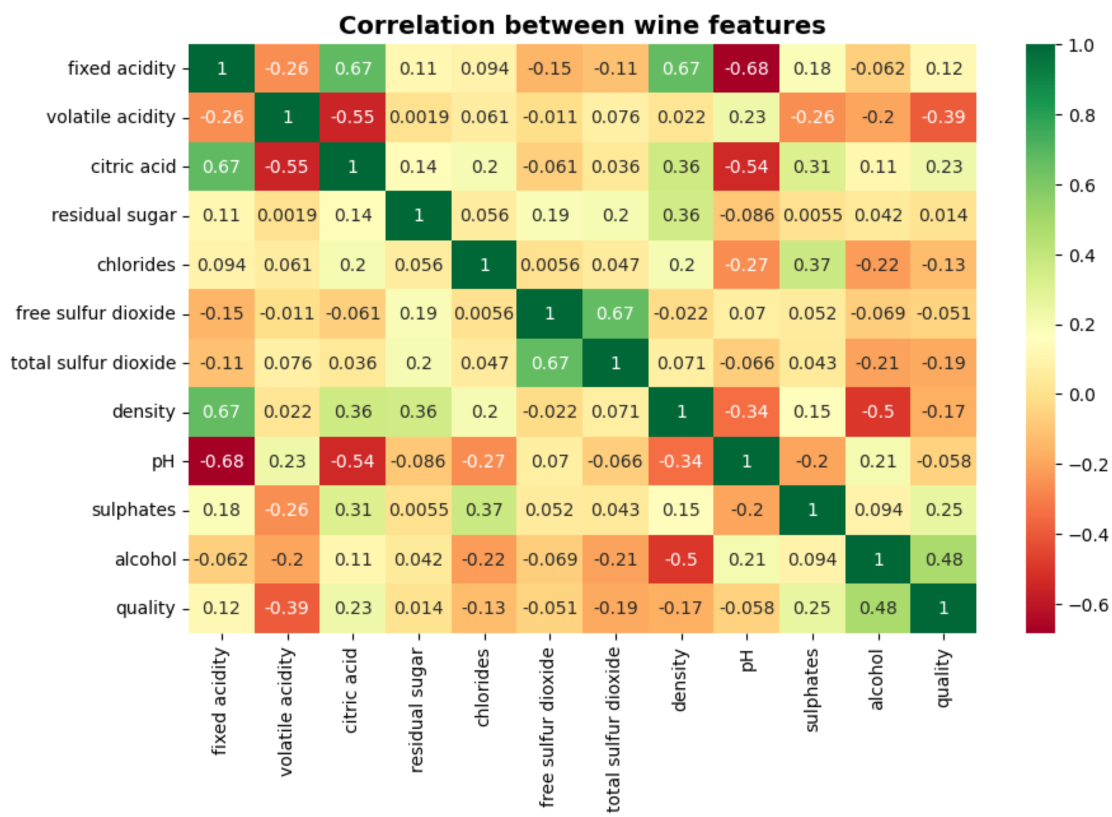
In [16]:
```
#Missing values
df.isnull().sum()
```

Out[16]:
```
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
dtype: int64
```

No missing values

In [14]:
```
# Check correlation of features with respect to the target variable
corr = df.corr()
fig, ax = plt.subplots(figsize = (10,6))
sns.heatmap(corr,ax=ax, annot= True, cmap='RdYlGn',)
plt.title('Correlation between wine features', fontsize=14, fontweight=
```

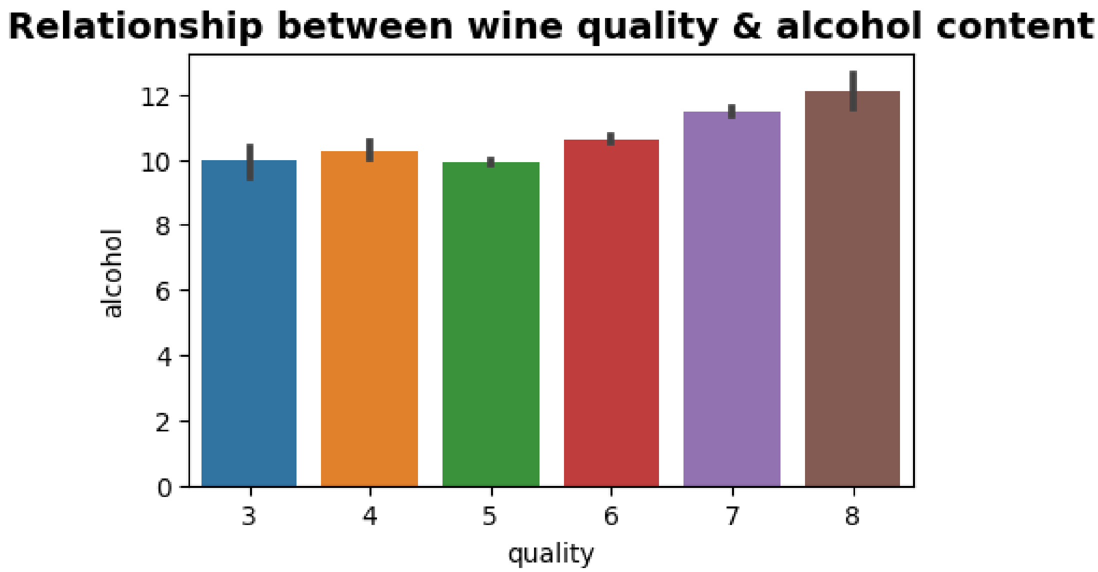
Out[14]:
```
Text(0.5, 1.0, 'Correlation between wine features')
```



In [20]:

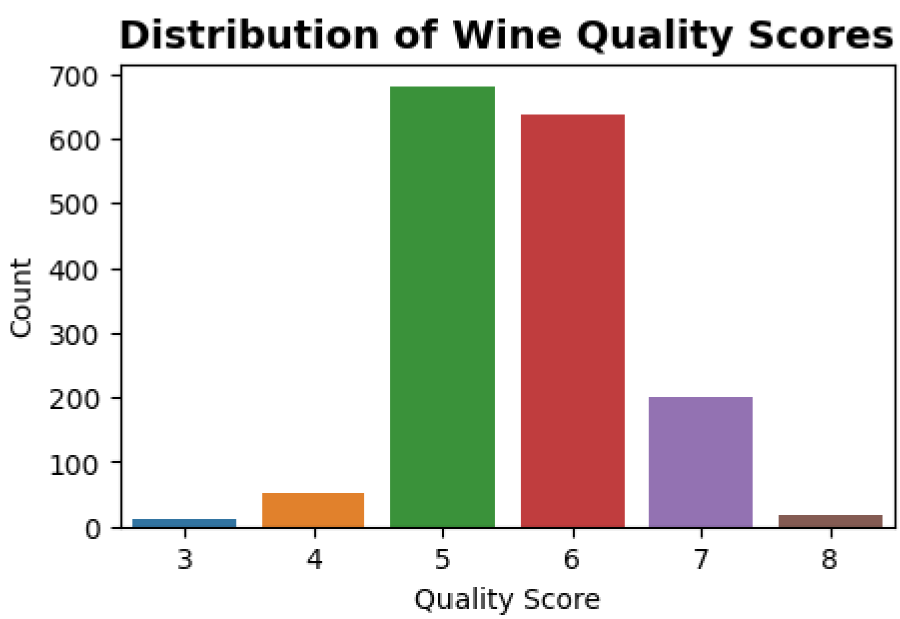
```
# Check relationship between wine quality and alcohol content
fig = plt.figure(figsize = (5,3))
sns.barplot(x = 'quality', y = 'alcohol', data = df)
plt.title('Relationship between wine quality & alcohol content', fontsi
```

Out[20]: Text(0.5, 1.0, 'Relationship between wine quality & alcohol content')



In [21]:

```
# Check for balance or imbalance in the dataset
fig = plt.figure(figsize = (5,3))
sns.countplot(x='quality', data=df)  # visualize the distribution of wi
plt.title('Distribution of Wine Quality Scores', fontsize=14, fontweigh
plt.xlabel('Quality Score')
plt.ylabel('Count')
plt.show()
```



In [22]:

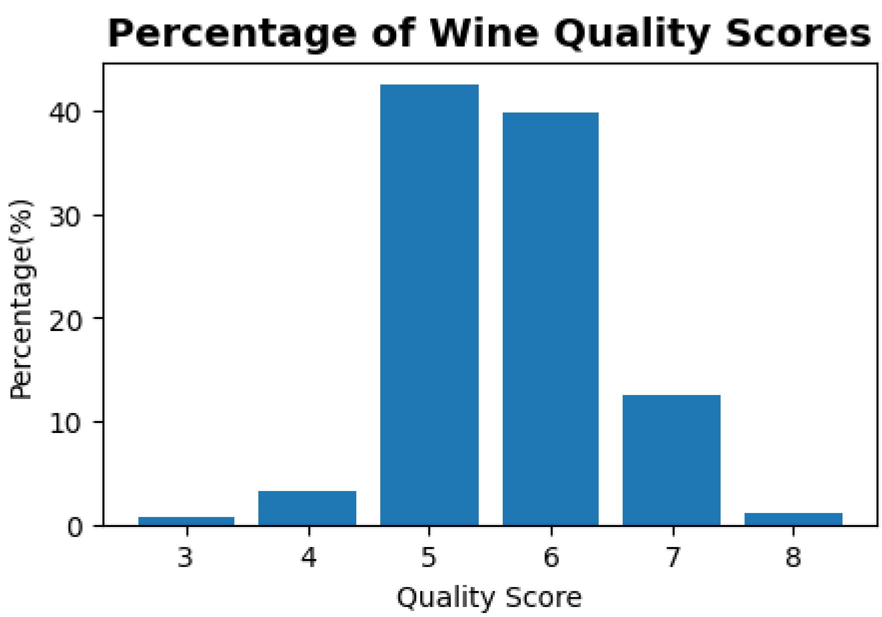
```
# Check imbalance of quality scores by percentage

# Count the number of each quality score
quality_counts = df['quality'].value_counts()

# Calculate the percentage of each quality score
quality_percents = quality_counts / quality_counts.sum() * 100

# Plot the percentage of each quality score
fig = plt.figure(figsize = (5,3))
plt.bar(quality_percents.index, quality_percents.values)
plt.title('Percentage of Wine Quality Scores', fontsize=14, fontweight=
plt.xlabel('Quality Score')
plt.ylabel('Percentage(%)')
```

Out[22]: Text(0, 0.5, 'Percentage(%)')



In [24]:

```
# Visualise the distribution of the independent features(variables)
from scipy.stats import probplot

X = df.drop(columns=['quality'])

# Create a histogram for each feature (X contains only the independent
n_cols = X.shape[1]
n_rows = int(n_cols/4) + 1

fig, axs = plt.subplots(n_rows, 4, figsize=(16, 4*n_rows))
fig.suptitle('Distribution of Independent Features', fontsize=20, fontw
for i, ax in enumerate(axs.flat):
    if i < n_cols:
        ax.hist(X.iloc[:, i], bins=30, edgecolor='black')
        ax.set_title(X.columns[i])
    else:
        ax.set_visible(False)

plt.show()
```

Data Modelling

In [26]:
```
:ing sets
:est_split(df.iloc[:, :-1], df.iloc[:, -1], test_size=0.2, random_state=
```

In [27]:
```
# perform data modelling with linear regression
linear_reg_model = LinearRegression()
linear_reg_model.fit(X_train, y_train)
```

Out[27]:
```
▾ LinearRegression
LinearRegression()
```

In [28]:
```

# predict the target variable for the testing set using the linear regr
y_pred_linear_reg = linear_reg_model.predict(X_test)
```

In [29]:
```
# evaluate the performance of the linear regression model
mae_linear_reg = mean_absolute_error(y_test, y_pred_linear_reg)
mse_linear_reg = mean_squared_error(y_test, y_pred_linear_reg)
rmse_linear_reg = np.sqrt(mse_linear_reg)
r2_linear_reg = r2_score(y_test, y_pred_linear_reg)

# Show evaluation scores
print("LINEAR REGRESSION EVALUATION METRICS")
print("Mean absolute error: {:.4f}".format(mae_linear_reg))
print("Mean squared error: {:.4f}".format(mse_linear_reg))
print("Root mean squared error: {:.4f}".format(rmse_linear_reg))
print("R2 score: {:.4f}".format(r2_linear_reg))
```

```
LINEAR REGRESSION EVALUATION METRICS
Mean absolute error: 0.4696
Mean squared error: 0.3845
Root mean squared error: 0.6201
R2 score: 0.3284
```

In [30]:
```
# perform data modelling with ridge regression
ridge_reg_model = Ridge(alpha=0.1)
ridge_reg_model.fit(X_train, y_train)
```

Out[30]:
```
▾ Ridge
Ridge(alpha=0.1)
```

In [31]:
```

# predict the target variable for the testing set using the ridge regre
y_pred_ridge_reg = ridge_reg_model.predict(X_test)
```

In [32]:

```
# evaluate the performance of the ridge regression model
mae_ridge_reg = mean_absolute_error(y_test, y_pred_ridge_reg)
mse_ridge_reg = mean_squared_error(y_test, y_pred_ridge_reg)
rmse_ridge_reg = np.sqrt(mse_ridge_reg)
r2_ridge_reg = r2_score(y_test, y_pred_ridge_reg)

# Show evaluation scores
print("RIDGE REGRESSION EVALUATION METRICS")
print("Mean absolute error: {:.4f}".format(mae_ridge_reg))
print("Mean squared error: {:.4f}".format(mse_ridge_reg))
print("Root mean squared error: {:.4f}".format(rmse_ridge_reg))
print("R2 score: {:.4f}".format(r2_ridge_reg))
```

```
RIDGE REGRESSION EVALUATION METRICS
Mean absolute error: 0.4686
Mean squared error: 0.3826
Root mean squared error: 0.6186
R2 score: 0.3316
```

In [ ]: