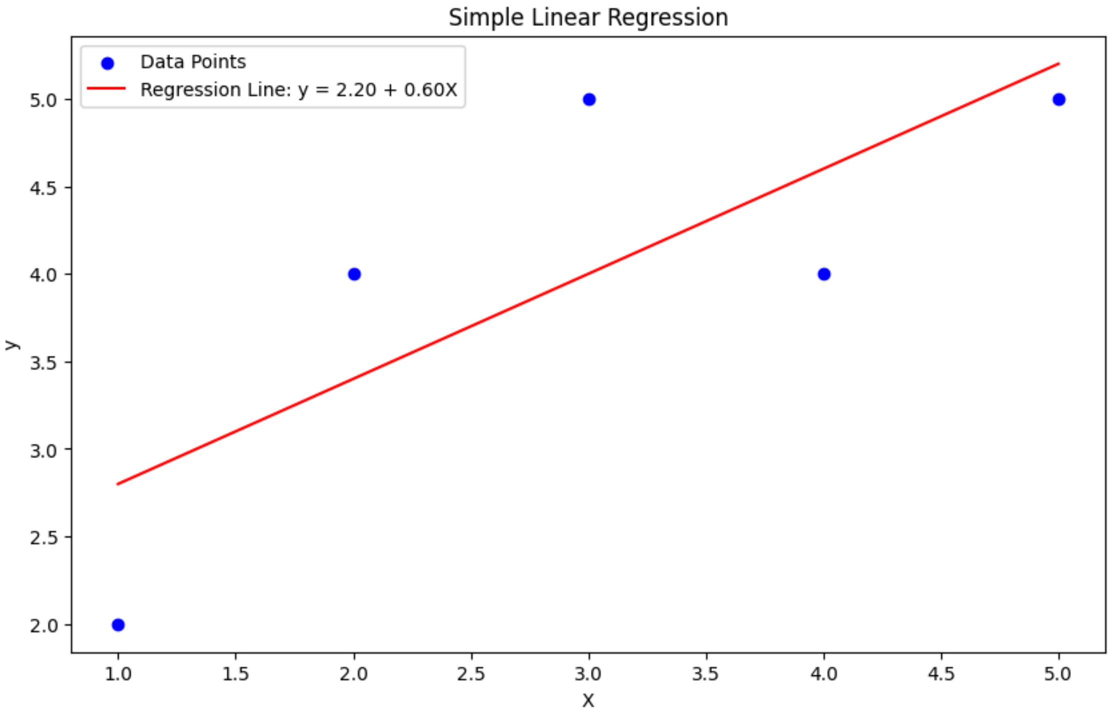
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Sample data
X = np.array([1, 2, 3, 4, 5]).reshape((-1, 1))
y = np.array([2, 4, 5, 4, 5])

# Create a linear regression model
model = LinearRegression()

# Fit the model to the data
model.fit(X, y)

# Print the coefficients
print('Intercept:', model.intercept_)
print('Slope:', model.coef_)

# Make predictions
y_pred = model.predict(X)
print('Predictions:', y_pred)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y, y_pred)
print('Mean Squared Error:', mse)

#Plotting the results
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color="blue", label="Data Points")

# Extract the slope value from the array for formatting
slope = model.coef_[0]
plt.plot(X, y_pred, color="red", label=f"Regression Line: y = {model.intercept_:.2f} + {slope:.2f}X")

plt.title("Simple Linear Regression")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```

```
Intercept: 2.2
Slope: [0.6]
Predictions: [2.8 3.4 4.  4.6 5.2]
Mean Squared Error: 0.47999999999999987
```



```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

Generate mount the google drive Close

1 of 1 Use code with caution

```
# prompt: mount the google drive

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import confusion_matrix, classification_report
```

```
df=pd.read_csv('/content/drive/My Drive/winequality-red.csv')
df.head()
```

	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

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.shape
```

```
(1599, 12)
```

```
df.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
count	1599.000000	1599.000000	1599.000000	1599.000000	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	46.467792	0.996747	3.311113	0.658149
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
```

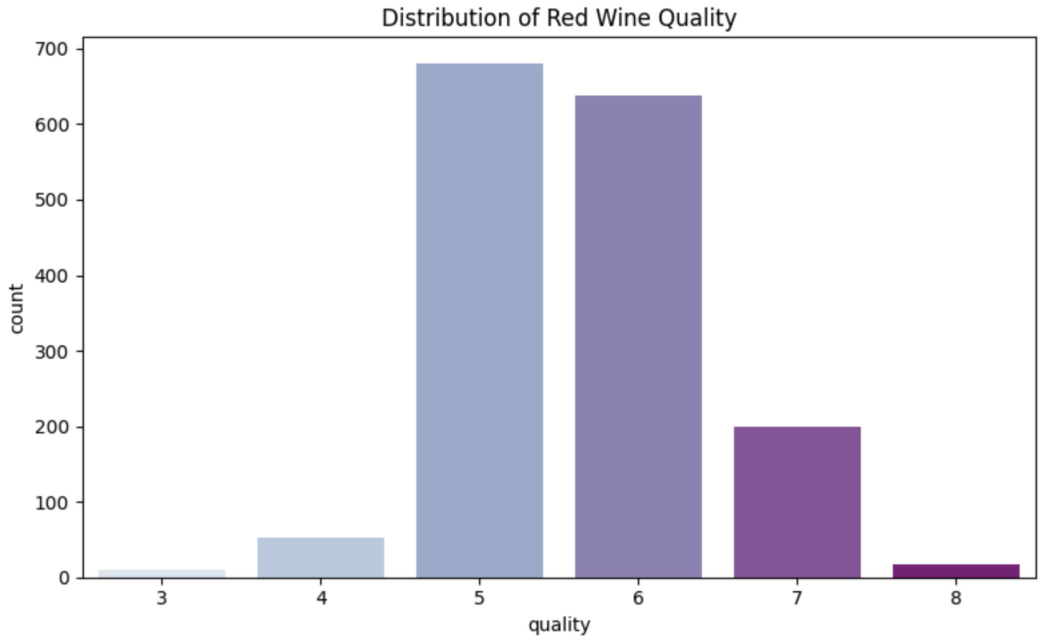
```
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   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
```

```
features = df.columns[:-1].values
label = [df.columns[-1]]

print ("The Features are:", features)
print ("The Label is:", label)
```

```
The Features are: ['fixed acidity' 'volatile acidity' 'citric acid' 'residual sugar'
 'chlorides' 'free sulfur dioxide' 'total sulfur dioxide' 'density' 'pH'
 'sulphates' 'alcohol']
The Label is: ['quality']
```

```
labels = ["3", "4", "5", "6", "7", "8"]
ticks = range(len(labels))

# Create a figure
fig, ax = plt.subplots(figsize=(8, 5))

# Plot the count plot
sns.countplot(data=df, x='quality', ax=ax, palette=sns.color_palette("BuPu"))
ax.set_title('Distribution of Red Wine Quality')
ax.set_xticks(ticks)
ax.set_xticklabels(labels)

# Adjust layout to prevent overlapping
plt.tight_layout()
plt.show()
```

```
<ipython-input-17-29f67405b1c5>:8: FutureWarning: 

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.countplot(data=df, x='quality', ax=ax, palette=sns.color_palette("BuPu"))
```




```
train_dataset, test_dataset = train_test_split(df, test_size=0.2, random_state=0)
```

```
len(train_dataset)
```

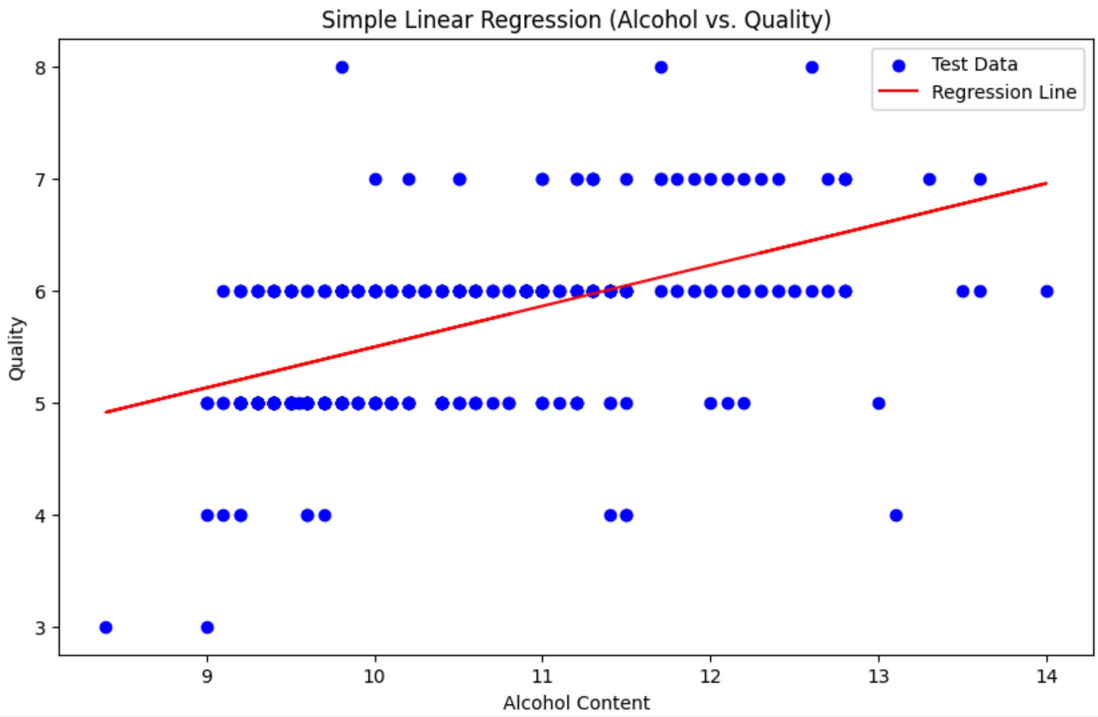
```
1279
```

```
len(test_dataset)
```

```
320
```

```
import matplotlib.pyplot as plt
# Select a single feature for simple linear regression (e.g., 'alcohol')
X = train_dataset[['alcohol']].values
y = train_dataset['quality'].values

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X, y)

# Make predictions on the test data
X_test = test_dataset[['alcohol']].values
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(test_dataset['quality'], y_pred)
print('Mean Squared Error:', mse)
print('slope and intercept values:',model.intercept_,model.coef_)

# Visualize the results
plt.figure(figsize=(10, 6))
plt.scatter(X_test, test_dataset['quality'], color="blue", label="Test Data")
plt.plot(X_test, y_pred, color="red", label="Regression Line")
plt.title("Simple Linear Regression (Alcohol vs. Quality)")
plt.xlabel("Alcohol Content")
plt.ylabel("Quality")
plt.legend()
plt.show()
```

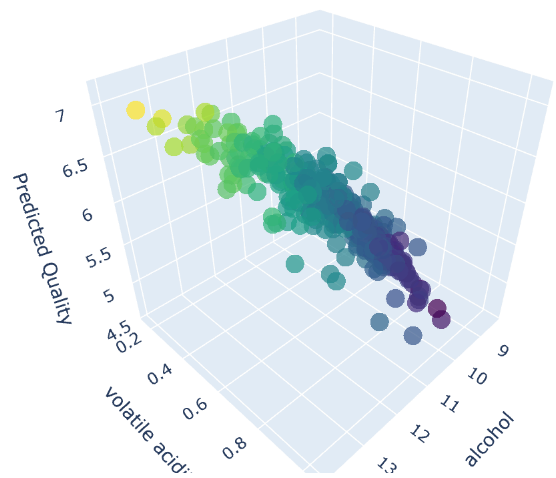
```
Mean Squared Error: 0.436459977733703 93
slope and intercept values: 1.8460491058282367 [0.36494637]
```



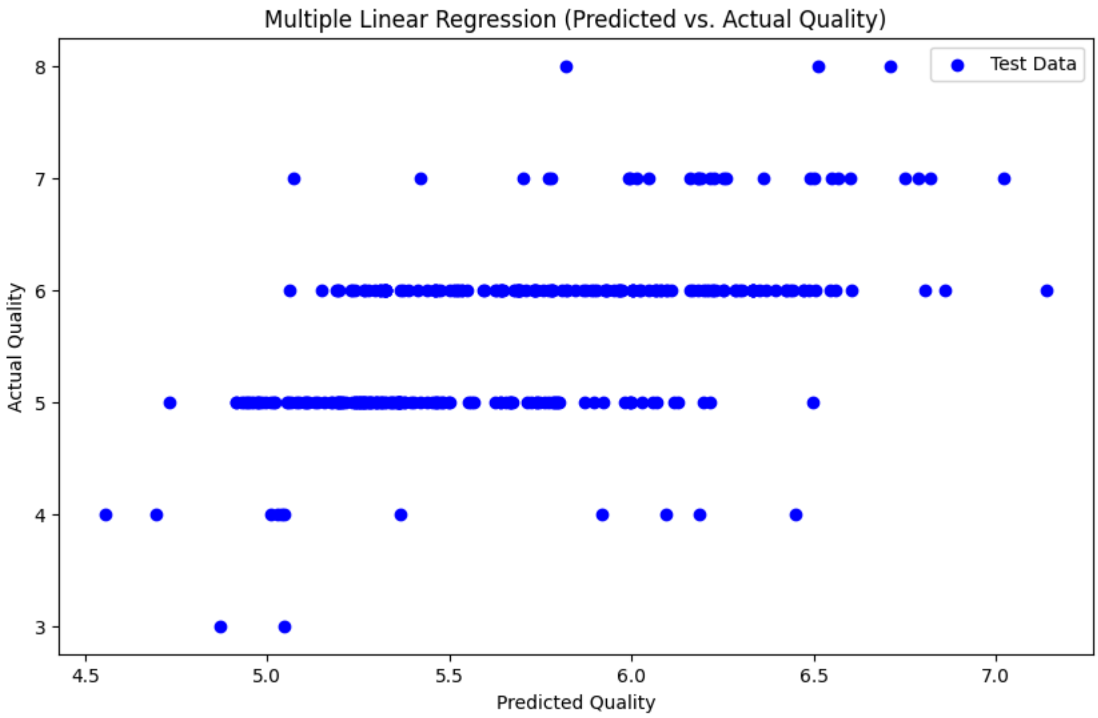
```
# Select multiple features for multiple linear regression
X = train_dataset[['alcohol', 'volatile acidity', 'sulphates', 'citric acid']].values
y = train_dataset['quality'].values

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X, y)

# Make predictions on the test data
X_test = test_dataset[['alcohol', 'volatile acidity', 'sulphates', 'citric acid']].values
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(test_dataset['quality'], y_pred)
print('Mean Squared Error (Multiple Linear Regression):', mse)

# Print the coefficients
print('Intercept:', model.intercept_)
print('Coefficients:', model.coef_)

# For visualization, let's just pick two features and plot against predicted quality
# You can change these features as needed
feature1 = 'alcohol'
feature2 = 'volatile acidity'

# Create a 3D scatter plot
fig = px.scatter_3d(test_dataset, x=feature1, y=feature2, z=y_pred,
                    color=y_pred, color_continuous_scale='Viridis',
                    opacity=0.7, labels={'z':'Predicted Quality'})

fig.update_layout(title='Multiple Linear Regression',
                  scene=dict(xaxis_title=feature1,
                         yaxis_title=feature2,
                         zaxis_title='Predicted Quality'))

fig.show()
```

```
Mean Squared Error (Multiple Linear Regression): 0.400926302923279 63
Intercept: 2.6531397319655468
Coefficients: [ 0.31487593 -1.33440129  0.67128956 -0.07662679]
```

Multiple Linear Regression




```
# Visualize the results - comparing predicted to actual
plt.figure(figsize=(10, 6))
plt.scatter(y_pred, test_dataset['quality'], color="blue", label="Test Data") # Changed from X_test to y_pred
plt.title("Multiple Linear Regression (Predicted vs. Actual Quality)") # Updated title
plt.xlabel("Predicted Quality") # Updated x-axis label
plt.ylabel("Actual Quality")
plt.legend()
plt.show()
```


Multiple Linear Regression (Predicted vs. Actual Quality)
Test Data
Actual Quality
Predicted Quality
8
7
6
5
4
3
4.5
5.0
5.5
6.0
6.5
7.0