Wine Quality Test

- Target = quality

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

```
df = pd.read_csv('/content/wine-qt.csv')
```

```
df.head()
```

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

Next steps: Generate code with df | New interactive sheet

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64
 12  Id                    1143 non-null   int64
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
```

```
df.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	
count	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000
mean	8.311111	0.531339	0.268364	2.532152	0.086933	15.615486	45.914698	0.996730	3.311
std	1.747595	0.179633	0.196686	1.355917	0.047267	10.250486	32.782130	0.001925	0.156
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740
25%	7.100000	0.392500	0.090000	1.900000	0.070000	7.000000	21.000000	0.995570	3.205
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	37.000000	0.996680	3.310
75%	9.100000	0.640000	0.420000	2.600000	0.090000	21.000000	61.000000	0.997845	3.400
max	15.900000	1.580000	1.000000	15.500000	0.611000	68.000000	289.000000	1.003690	4.010

```
df.isna().sum()
```

	0
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
Id	0

dtype: int64

- No missing values

```
df.duplicated().sum()
```

```
np.int64(0)
```

- No duplicate values

- All are numerical values so no need for:
  - Splitting
  - Encoding

```
X = df.drop('quality', axis=1)
y = df['quality']
```

- Splitting X and y as y is a numerical target variable
- We do not need to handle outliers of target variable

```
for col in X.columns:
    plt.figure(figsize=(10, 5))
    plt.boxplot(df[col])
    plt.title(col)
    plt.show()
```

```
for col in X.columns:
    q1 = X[col].quantile(0.25)
    q3 = X[col].quantile(0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr

    X[col] = X[col].clip(lower=lower, upper=upper)
```

- Handled outliers of each column

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

- Standardized independent features

Logistic Regression

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

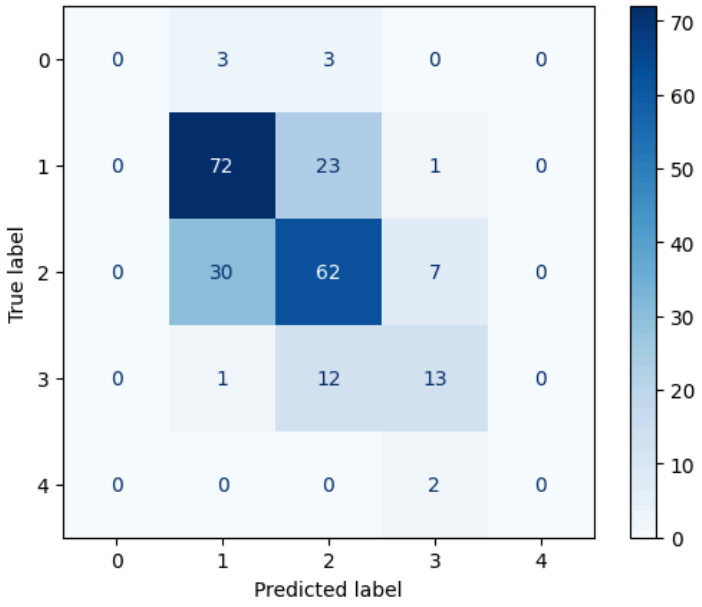
```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
y_pred = model.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d1a455aa630>
```




```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

acc = accuracy_score(y_test, y_pred)
pre = precision_score(y_test, y_pred, average='macro', zero_division=1)
rec = recall_score(y_test, y_pred, average='macro', zero_division=1)
f1 = f1_score(y_test, y_pred, average='macro', zero_division=1)
```

```
print('Accuracy: ', acc)
print('Precision: ', pre)
print('Recall: ', rec)
print('F1_score: ', f1)
```

```
Accuracy:  0.6419213973799127
Precision:  0.7728925348646432
Recall:  0.3752525252525253
```

```
F1_score:  0.37331982198322383
```

KNN Classification

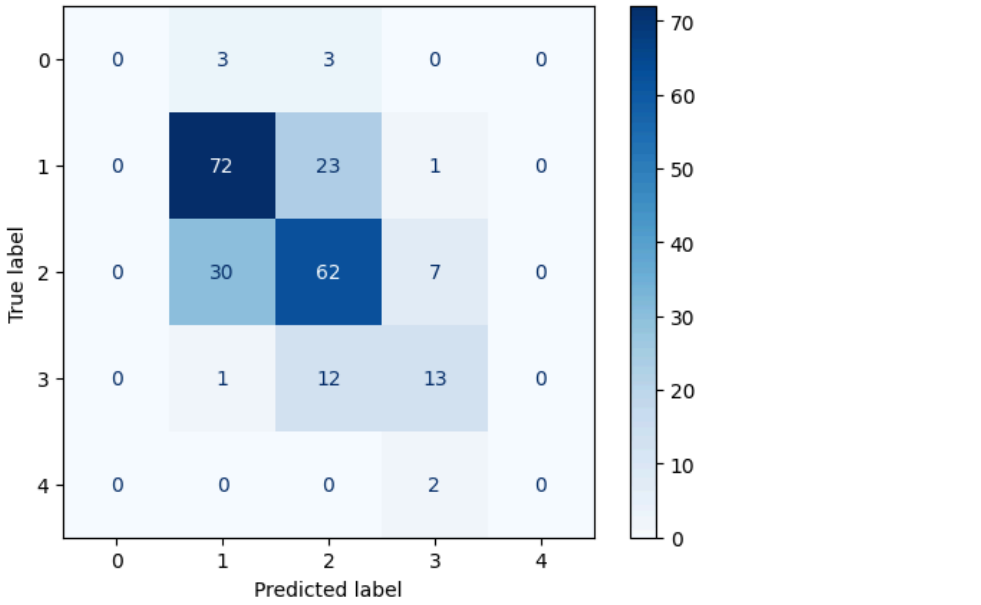
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5, metric='minkowski')
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

```
y_pred_knn = knn.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d1a5a6bfb90>
```




```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

acc = accuracy_score(y_test, y_pred)
pre = precision_score(y_test, y_pred, average='macro', zero_division=1)
rec = recall_score(y_test, y_pred, average='macro', zero_division=1)
f1 = f1_score(y_test, y_pred, average='macro', zero_division=1)
```

```
print('Accuracy: ', acc)
print('Precision: ', pre)
print('Recall: ', rec)
print('F1_score: ', f1)
```

```
Accuracy:  0.6419213973799127
Precision:  0.7728925348646432
Recall:  0.3752525252525253
F1_score:  0.37331982198322383
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(
    criterion='entropy',
```

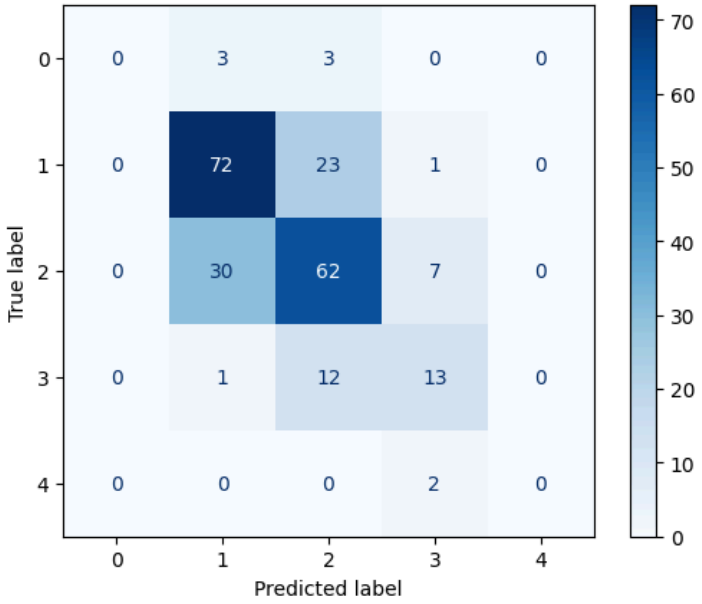
```
    random_state=42,
    max_depth=10
)
tree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=10, random_state=42)
```

```
y_pred_tree = tree.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d1a32844740>
```




```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

acc = accuracy_score(y_test, y_pred)
pre = precision_score(y_test, y_pred, average='macro', zero_division=1)
rec = recall_score(y_test, y_pred, average='macro', zero_division=1)
f1 = f1_score(y_test, y_pred, average='macro', zero_division=1)
```

```
print('Accuracy: ', acc)
print('Precision: ', pre)
print('Recall: ', rec)
print('F1_score: ', f1)
```

```
Accuracy:  0.6419213973799127
Precision:  0.7728925348646432
Recall:  0.3752525252525253
F1_score:  0.37331982198322383
```

SVM Classification

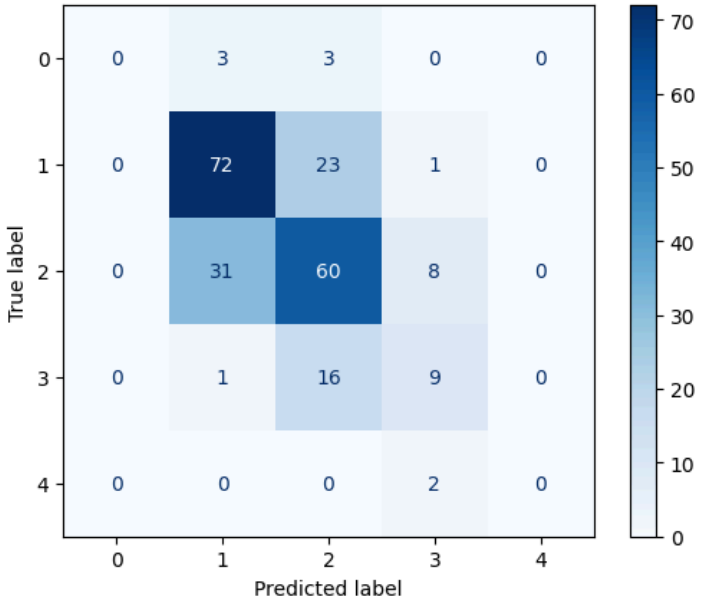
```
from sklearn.svm import SVC

svc = SVC(kernel='linear', C=1.0, gamma='scale')
svc.fit(X_train, y_train)
```

```
SVC(kernel='linear')
```

```
y_pred_svc = svc.predict(X_test)
```

```
cm = confusion_matrix(y_test, y_pred_svc)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d1a30402750>
```




```
acc = accuracy_score(y_test, y_pred_svc)
pre = precision_score(y_test, y_pred_svc, average='macro', zero_division=1)
rec = recall_score(y_test, y_pred_svc, average='macro', zero_division=1)
f1 = f1_score(y_test, y_pred_svc, average='macro', zero_division=1)
```

```
print('Accuracy: ', acc)
print('Precision: ', pre)
print('Recall: ', rec)
print('F1 score: ', f1)
```

```
Accuracy:  0.6157205240174672
Precision:  0.7422264980758659
Recall:  0.34044289044289044
F1 score:  0.33953577582211025
```

Summary

- Comparitively Logistic Regression, Decision Tree and KNN algorithmn shows higher accuracy and precision than SVM
- In this case all the three models can be suggested with equal priority

```
Start coding or generate with AI.
```