

Machine Learning Assignment

Name: Naif Ahmed Basalib

Dataset: Wine Quality Dataset, WineQT.csv (from Kaggle)

- Tasks:
- 1. Download dataset from Kaggle
 - 2. Handle missing values
 - 3. Apply ML models with Grid Search and evaluate
 - 4. Apply two feature selection methods
 - 5. Apply models to selected features and evaluate

```
!pip install pillow
```

Requirement already satisfied: pillow in /usr/local/lib/python3.12/dist-packages (11.3.0)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_selection import SelectKBest, f_classif, RFE
import warnings
warnings.filterwarnings('ignore')
```

1. Load and Explore the Dataset

```
df = pd.read_csv('WineQT.csv')
print('Dataset shape:', df.shape)
df.head()
```

Dataset shape: (1143, 13)

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

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	pH	sulphates
count	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000
mean	8.311111	0.531339	0.268364	2.532152	0.086933	15.615486	45.914698	0.996730	3.311015	0.657000
std	1.747595	0.179633	0.196686	1.355917	0.047267	10.250486	32.782130	0.001925	0.156664	0.170000
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000
25%	7.100000	0.392500	0.090000	1.900000	0.070000	7.000000	21.000000	0.995570	3.205000	0.550000
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	37.000000	0.996680	3.310000	0.620000
75%	9.100000	0.640000	0.420000	2.600000	0.090000	21.000000	61.000000	0.997845	3.400000	0.730000
max	15.900000	1.580000	1.000000	15.500000	0.611000	68.000000	289.000000	1.003690	4.010000	2.000000

2. Handle Missing Values

```

print('Missing values per column:')
print(df.isnull().sum())
print('\nTotal missing values:', df.isnull().sum().sum())

```

```

Missing values per column:
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

```

```
Total missing values: 0
```

```

# fill numeric missing values with the column mean
for col in df.select_dtypes(include=np.number).columns:
    df[col] = df[col].fillna(df[col].mean())

print('Missing values after handling:')
print(df.isnull().sum())
print('\nDataset is clean and ready!')

```

```

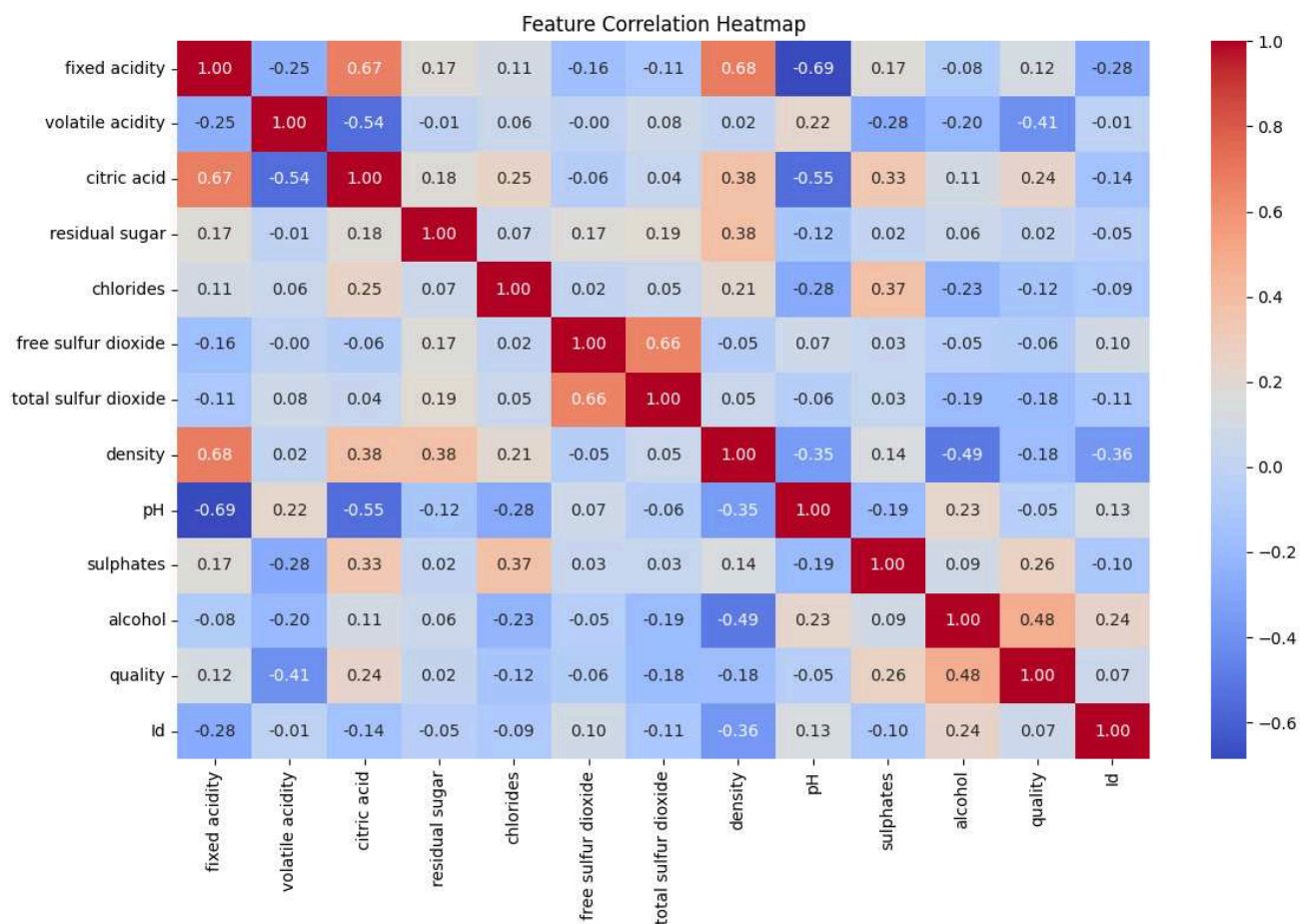
Missing values after handling:
fixed acidity      0
volatile acidity   0
citric acid        0
residual sugar     0
chlorides          0
free sulfur dioxide 0
total sulfur dioxide 0
density            0
pH                 0
sulphates          0
alcohol            0
quality            0
Id                 0

```

```
dtype: int64
```

Dataset is clean and ready!

```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(numeric_only=True), annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Feature Correlation Heatmap')
plt.tight_layout()
plt.show()
```



3. Preprocessing

```
# drop the Id column - not a feature
df = df.drop('Id', axis=1)

# convert quality to binary target: 1 = good wine (>= 7), 0 = not good (< 7)
df['quality'] = df['quality'].apply(lambda x: 1 if x >= 7 else 0)

print('Class distribution:')
print(df['quality'].value_counts())
print(df['quality'].value_counts(normalize=True).round(3))
```

```
Class distribution:
quality
0    984
1    159
Name: count, dtype: int64
quality
0    0.861
1    0.139
Name: proportion, dtype: float64
```

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

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print('Training set:', X_train.shape)
print('Test set:', X_test.shape)
```

```
Training set: (914, 11)
Test set: (229, 11)
```

4. Machine Learning Models with Grid Search

4.1 Random Forest Classifier

```
rf_params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5]
}

rf_grid = GridSearchCV(RandomForestClassifier(random_state=42), rf_params,
                        cv=5, scoring='accuracy', n_jobs=-1)
rf_grid.fit(X_train_scaled, y_train)

print('Best parameters:', rf_grid.best_params_)
print('Best CV Accuracy:', round(rf_grid.best_score_, 4))

rf_pred = rf_grid.predict(X_test_scaled)
print('\nTest Accuracy:', round(accuracy_score(y_test, rf_pred), 4))
print('\nClassification Report:')
print(classification_report(y_test, rf_pred))
```

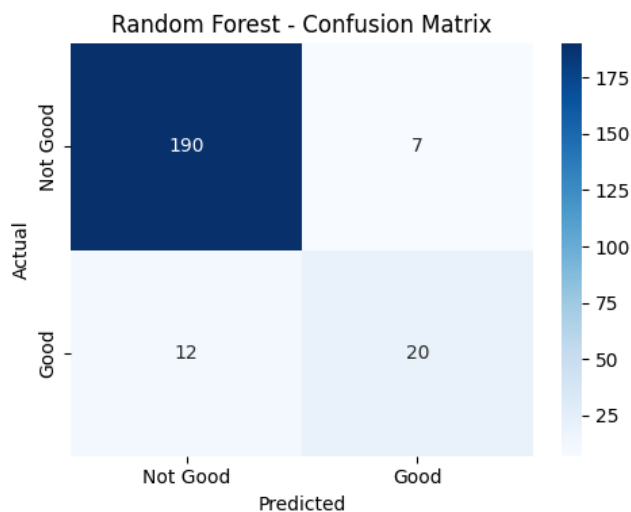
```
Best parameters: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 100}
Best CV Accuracy: 0.9059
```

```
Test Accuracy: 0.917
```

```
Classification Report:
```

	precision	recall	f1-score	support
0	0.94	0.96	0.95	197
1	0.74	0.62	0.68	32
accuracy			0.92	229
macro avg	0.84	0.79	0.82	229
weighted avg	0.91	0.92	0.91	229

```
cm = confusion_matrix(y_test, rf_pred)
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Good', 'Good'], yticklabels=['Not Good', 'Good'])
plt.title('Random Forest - Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout()
plt.show()
```



4.2 Support Vector Machine (SVM)

```
svm_params = {
    'C': [0.1, 1, 10],
    'kernel': ['rbf', 'linear'],
    'gamma': ['scale', 'auto']
}

svm_grid = GridSearchCV(SVC(), svm_params, cv=5, scoring='accuracy', n_jobs=-1)
svm_grid.fit(X_train_scaled, y_train)

print('Best parameters:', svm_grid.best_params_)
print('Best CV Accuracy:', round(svm_grid.best_score_, 4))

svm_pred = svm_grid.predict(X_test_scaled)
print('\nTest Accuracy:', round(accuracy_score(y_test, svm_pred), 4))
print('\nClassification Report:')
print(classification_report(y_test, svm_pred))
```

Best parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}

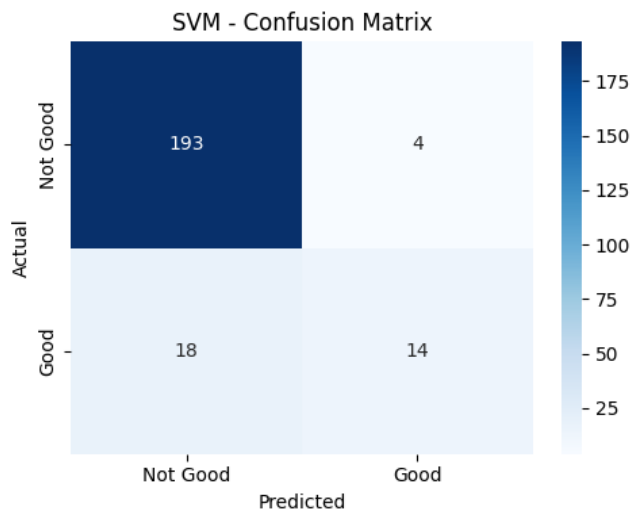
Best CV Accuracy: 0.8786

Test Accuracy: 0.9039

Classification Report:

	precision	recall	f1-score	support
0	0.91	0.98	0.95	197
1	0.78	0.44	0.56	32
accuracy			0.90	229
macro avg	0.85	0.71	0.75	229
weighted avg	0.90	0.90	0.89	229

```
cm = confusion_matrix(y_test, svm_pred)
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Good', 'Good'], yticklabels=['Not Good', 'Good'])
plt.title('SVM - Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout()
plt.show()
```



4.3 K-Nearest Neighbors (KNN)

```
knn_params = {
    'n_neighbors': [3, 5, 7, 11],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

knn_grid = GridSearchCV(KNeighborsClassifier(), knn_params, cv=5, scoring='accuracy', n_jobs=-1)
knn_grid.fit(X_train_scaled, y_train)

print('Best parameters:', knn_grid.best_params_)
print('Best CV Accuracy:', round(knn_grid.best_score_, 4))

knn_pred = knn_grid.predict(X_test_scaled)
print('\nTest Accuracy:', round(accuracy_score(y_test, knn_pred), 4))
print('\nClassification Report:')
print(classification_report(y_test, knn_pred))
```

Best parameters: {'metric': 'euclidean', 'n_neighbors': 11, 'weights': 'distance'}

Best CV Accuracy: 0.8961

Test Accuracy: 0.9039

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.97	0.95	197
1	0.73	0.50	0.59	32
accuracy			0.90	229
macro avg	0.82	0.73	0.77	229
weighted avg	0.90	0.90	0.90	229

Models Performance Summary (All Features)

```
summary_all = pd.DataFrame({
    'Model': ['Random Forest', 'SVM', 'KNN'],
    'Best CV Score': [round(rf_grid.best_score_, 4),
                     round(svm_grid.best_score_, 4),
                     round(knn_grid.best_score_, 4)],
    'Test Accuracy': [round(accuracy_score(y_test, rf_pred), 4),
                     round(accuracy_score(y_test, svm_pred), 4),
                     round(accuracy_score(y_test, knn_pred), 4)]
})
summary_all
```

	Model	Best CV Score	Test Accuracy	
0	Random Forest	0.9059	0.9170	
1	SVM	0.8786	0.9039	
2	KNN	0.8961	0.9039	

Next steps: [Generate code with summary_all](#) [New interactive sheet](#)

5. Feature Selection

5.1 Method 1: SelectKBest (ANOVA F-test)

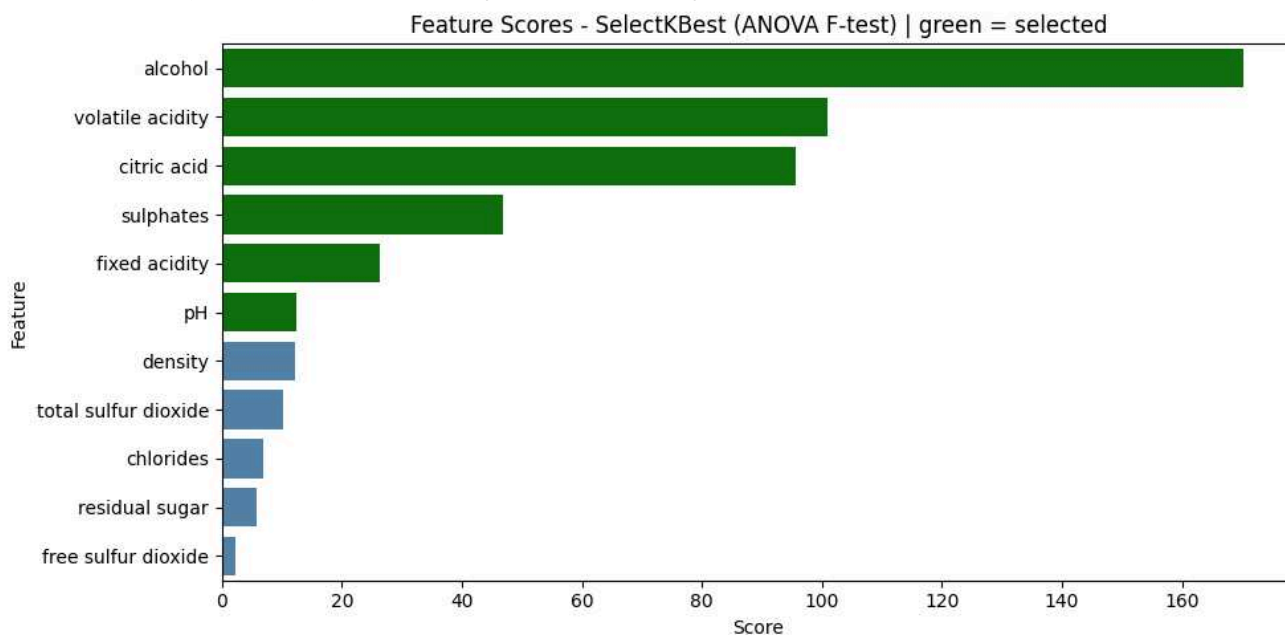
```
selector_kbest = SelectKBest(f_classif, k=6)
X_train_kbest = selector_kbest.fit_transform(X_train_scaled, y_train)
X_test_kbest = selector_kbest.transform(X_test_scaled)

selected_features_kbest = X.columns[selector_kbest.get_support()]
print('Selected features (SelectKBest):', selected_features_kbest.tolist())

scores_df = pd.DataFrame({
    'Feature': X.columns,
    'Score': selector_kbest.scores_
}).sort_values('Score', ascending=False)

plt.figure(figsize=(10, 5))
colors = ['green' if f in selected_features_kbest else 'steelblue' for f in scores_df['Feature']]
sns.barplot(x='Score', y='Feature', data=scores_df, palette=colors)
plt.title('Feature Scores - SelectKBest (ANOVA F-test) | green = selected')
plt.tight_layout()
plt.show()
```

Selected features (SelectKBest): ['fixed acidity', 'volatile acidity', 'citric acid', 'pH', 'sulphates', 'alcohol']



5.2 Method 2: Recursive Feature Elimination (RFE)

```
rfe_estimator = RandomForestClassifier(random_state=42, n_estimators=100)
rfe = RFE(estimator=rfe_estimator, n_features_to_select=6)
rfe.fit(X_train_scaled, y_train)

X_train_rfe = rfe.transform(X_train_scaled)
X_test_rfe = rfe.transform(X_test_scaled)

selected_features_rfe = X.columns[rfe.support_]
```

```
print('Selected features (RFE):', selected_features_rfe.tolist())
```

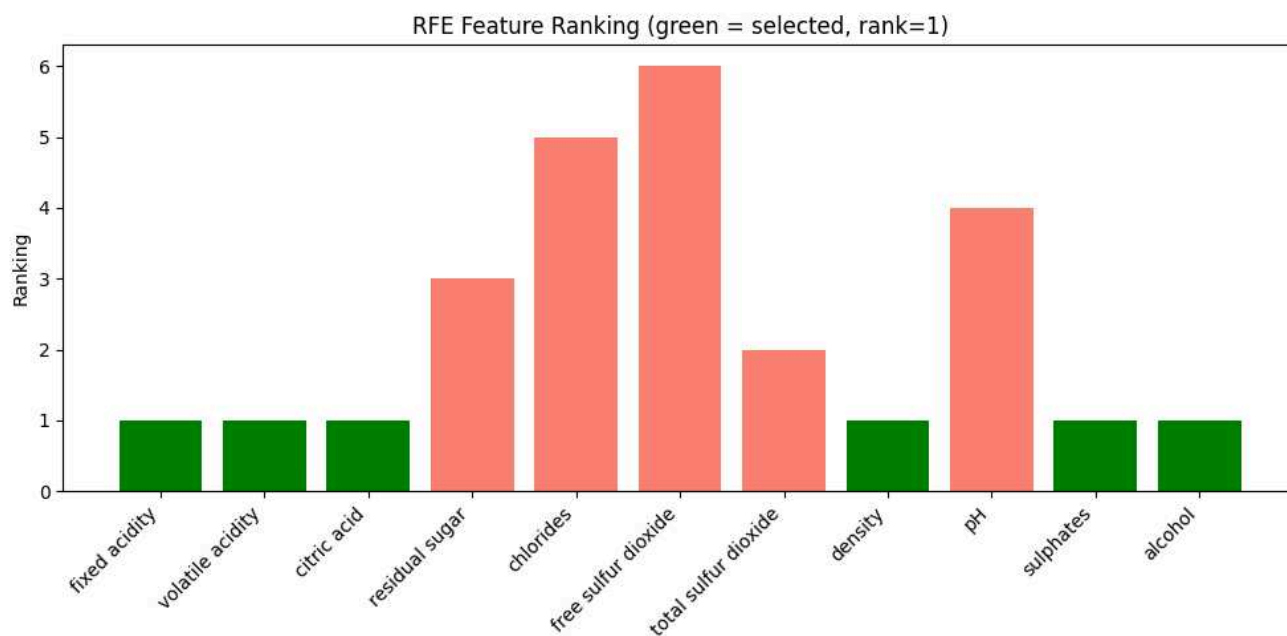
```
ranking_df = pd.DataFrame({
    'Feature': X.columns,
    'Ranking': rfe.ranking_
}).sort_values('Ranking')
print('\nFeature Rankings (1 = selected):')
print(ranking_df.to_string(index=False))
```

Selected features (RFE): ['fixed acidity', 'volatile acidity', 'citric acid', 'density', 'sulphates', 'alcohol']

Feature Rankings (1 = selected):

Feature	Ranking
fixed acidity	1
volatile acidity	1
citric acid	1
density	1
alcohol	1
sulphates	1
total sulfur dioxide	2
residual sugar	3
pH	4
chlorides	5
free sulfur dioxide	6

```
plt.figure(figsize=(10, 5))
colors = ['green' if s else 'salmon' for s in rfe.support_]
plt.bar(X.columns, rfe.ranking_, color=colors)
plt.xticks(rotation=45, ha='right')
plt.title('RFE Feature Ranking (green = selected, rank=1)')
plt.ylabel('Ranking')
plt.tight_layout()
plt.show()
```



6. Apply Models to Selected Features

6.1 Models with SelectKBest Features

```
print(f'Using {X_train_kbest.shape[1]} features: {selected_features_kbest.tolist()}\n')

rf_kbest = GridSearchCV(RandomForestClassifier(random_state=42), rf_params,
                        cv=5, scoring='accuracy', n_jobs=-1)
rf_kbest.fit(X_train_kbest, y_train)
rf_pred_kbest = rf_kbest.predict(X_test_kbest)

svm_kbest = GridSearchCV(SVC(), svm_params, cv=5, scoring='accuracy', n_jobs=-1)
svm_kbest.fit(X_train_kbest, y_train)
```

```

svm_pred_kbest = svm_kbest.predict(X_test_kbest)

knn_kbest = GridSearchCV(KNeighborsClassifier(), knn_params, cv=5, scoring='accuracy', n_jobs=-1)
knn_kbest.fit(X_train_kbest, y_train)
knn_pred_kbest = knn_kbest.predict(X_test_kbest)

print('--- SelectKBest Results ---')
print('Random Forest:', round(accuracy_score(y_test, rf_pred_kbest), 4))
print('SVM:           ', round(accuracy_score(y_test, svm_pred_kbest), 4))
print('KNN:           ', round(accuracy_score(y_test, knn_pred_kbest), 4))

Using 6 features: ['fixed acidity', 'volatile acidity', 'citric acid', 'pH', 'sulphates', 'alcohol']

--- SelectKBest Results ---
Random Forest: 0.8865
SVM:           0.8908
KNN:           0.8908

```

6.2 Models with RFE Features

```

print(f'Using {X_train_rfe.shape[1]} features: {selected_features_rfe.tolist()}\n')

rf_rfe = GridSearchCV(RandomForestClassifier(random_state=42), rf_params,
                      cv=5, scoring='accuracy', n_jobs=-1)
rf_rfe.fit(X_train_rfe, y_train)
rf_pred_rfe = rf_rfe.predict(X_test_rfe)

svm_rfe = GridSearchCV(SVC(), svm_params, cv=5, scoring='accuracy', n_jobs=-1)
svm_rfe.fit(X_train_rfe, y_train)
svm_pred_rfe = svm_rfe.predict(X_test_rfe)

knn_rfe = GridSearchCV(KNeighborsClassifier(), knn_params, cv=5, scoring='accuracy', n_jobs=-1)
knn_rfe.fit(X_train_rfe, y_train)
knn_pred_rfe = knn_rfe.predict(X_test_rfe)

print('--- RFE Results ---')
print('Random Forest:', round(accuracy_score(y_test, rf_pred_rfe), 4))
print('SVM:           ', round(accuracy_score(y_test, svm_pred_rfe), 4))
print('KNN:           ', round(accuracy_score(y_test, knn_pred_rfe), 4))

Using 6 features: ['fixed acidity', 'volatile acidity', 'citric acid', 'density', 'sulphates', 'alcohol']

--- RFE Results ---
Random Forest: 0.8996
SVM:           0.8865
KNN:           0.8996

```

7. Final Comparison

```

final_results = pd.DataFrame({
    'Model': ['Random Forest', 'SVM', 'KNN'],
    'All Features': [
        round(accuracy_score(y_test, rf_pred), 4),
        round(accuracy_score(y_test, svm_pred), 4),
        round(accuracy_score(y_test, knn_pred), 4)
    ],
    'SelectKBest (k=6)': [
        round(accuracy_score(y_test, rf_pred_kbest), 4),
        round(accuracy_score(y_test, svm_pred_kbest), 4),
        round(accuracy_score(y_test, knn_pred_kbest), 4)
    ],
    'RFE (k=6)': [
        round(accuracy_score(y_test, rf_pred_rfe), 4),
        round(accuracy_score(y_test, svm_pred_rfe), 4),
        round(accuracy_score(y_test, knn_pred_rfe), 4)
    ]
})
final_results

```

	Model	All Features	SelectKBest (k=6)	RFE (k=6)
0	Random Forest	0.9170	0.8865	0.8996
1	SVM	0.9039	0.8908	0.8865

Next steps: [Generate code with final_results](#) [New interactive sheet](#)

```
fig, ax = plt.subplots(figsize=(10, 6))
x = np.arange(3)
width = 0.25

ax.bar(x - width, final_results['All Features'], width, label='All Features (11)', color='steelblue')
ax.bar(x, final_results['SelectKBest (k=6)'], width, label='SelectKBest (k=6)', color='orange')
ax.bar(x + width, final_results['RFE (k=6)'], width, label='RFE (k=6)', color='green')

ax.set_xticks(x)
ax.set_xticklabels(final_results['Model'])
ax.set_ylabel('Test Accuracy')
ax.set_title('Model Accuracy: All Features vs Feature Selection Methods')
ax.legend()
ax.set_ylim(0.7, 1.0)
plt.tight_layout()
plt.show()
```

