

▼ Practical Machine Learning and Data Exploration

Project-2: Feature Selection and Machine Learning Models

Student Name: Moayad Khayat
Course: Practical Machine Learning - First Trimester 2026
University: Midocean University

Introduction

This project is based on **Lecture 9** guidelines and covers the following tasks using the **WineQT** dataset sourced from Kaggle:

- 1. **Handle Missing Values** – Inspect and clean the dataset.
- 2. **Apply ML Models with Grid Search** – Train and optimise Decision Tree, Random Forest, and SVR.
- 3. **Feature Selection – Method 1: RFE** – Recursive Feature Elimination (Wrapper method).
- 4. **Feature Selection – Method 2: Gradient Boosting Embedded** – Feature importance from a tree-ensemble model.
- 5. **Re-train & Evaluate Models** on selected features and compare performance.

▼ 0. Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFECV
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

sns.set(style='whitegrid')
print('All libraries imported successfully.')
```

All libraries imported successfully.

▼ Step 1 — Load the Dataset

The **WineQT** dataset is sourced from Kaggle. It contains physicochemical measurements of red wine samples along with a sensory quality score (3–8).

```
df = pd.read_csv('WineQT.csv')
print(f'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](#)

```
print('--- Data Types ---')
print(df.dtypes)
print('\n--- Basic Statistics ---')
df.describe()
```

--- Data Types ---

```
fixed acidity      float64
volatile acidity   float64
citric acid        float64
residual sugar     float64
chlorides          float64
free sulfur dioxide float64
total sulfur dioxide float64
density           float64
pH               float64
sulphates         float64
alcohol           float64
quality           int64
Id                int64
dtype: object
```

--- Basic Statistics ---

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	quality
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	0.915541
std	1.747595	0.179633	0.196686	1.355917	0.047267	10.250486	32.782130	0.001925	0.156664	0.423547
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.490000
25%	7.100000	0.392500	0.090000	1.900000	0.070000	7.000000	21.000000	0.995570	3.205000	0.540000
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	37.000000	0.996680	3.310000	0.610000
75%	9.100000	0.640000	0.420000	2.600000	0.090000	21.000000	61.000000	0.997845	3.400000	0.690000
max	15.900000	1.580000	1.000000	15.500000	0.611000	68.000000	289.000000	1.003690	4.010000	2.060000

Step 2 — Handle Missing Values

Before applying any model we must ensure the data is clean and free of nulls.

```
# 2.1 Detect missing values
missing = df.isnull().sum()
missing_pct = (missing / len(df)) * 100

missing_df = pd.DataFrame({
    'Missing Count': missing,
    'Missing %': missing_pct
}).sort_values('Missing %', ascending=False)

print('Missing values per column:')
print(missing_df)
```

Missing values per column:

	Missing Count	Missing %
fixed acidity	0	0.0
volatile acidity	0	0.0
citric acid	0	0.0
residual sugar	0	0.0
chlorides	0	0.0
free sulfur dioxide	0	0.0
total sulfur dioxide	0	0.0
density	0	0.0
pH	0	0.0
sulphates	0	0.0
alcohol	0	0.0
quality	0	0.0
Id	0	0.0

```
# 2.2 The WineQT dataset has no missing values, but we apply a robust strategy anyway.
# Numerical columns -> fill with median; categorical -> fill with mode.
for col in df.columns:
    if df[col].isnull().sum() > 0:
        if df[col].dtype == 'object':
            df[col].fillna(df[col].mode()[0], inplace=True)
        else:
            df[col].fillna(df[col].median(), inplace=True)
```

```
print(f'Total missing values after imputation: {df.isnull().sum().sum()}')
```

```
Total missing values after imputation: 0
```

```
# 2.3 Drop the non-informative 'Id' column
if 'Id' in df.columns:
    df.drop(columns=['Id'], inplace=True)
    print('Dropped Id column.')

print(f'Final dataset shape: {df.shape}')
```

```
Dropped Id column.
Final dataset shape: (1143, 12)
```

✓ Step 3 — Split Features and Target / Scale Data

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

feature_names = X.columns.tolist()
print(f'Features ({len(feature_names)}): {feature_names}')
print(f'Target: quality | Unique values: {sorted(y.unique())}')
```

```
# Train-test split 80 / 20
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
# Feature scaling (required by SVR)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
print(f'\nTraining samples : {X_train.shape[0]}')
print(f'Testing samples : {X_test.shape[0]}')
```

```
Features (11): ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'wine quality']
Target: quality | Unique values: [np.int64(3), np.int64(4), np.int64(5), np.int64(6), np.int64(7), np.int64(8)]
```

```
Training samples : 914
Testing samples : 229
```

✓ Step 4 — Apply ML Models with Grid Search (All Features)

We train **Decision Tree Regressor**, **Random Forest Regressor**, and **SVR** using `GridSearchCV` to find the best hyperparameters.

```
def evaluate(name, model, X_tr, X_te, y_tr, y_te):
    """Fit model, predict, and return a metrics dict."""
    model.fit(X_tr, y_tr)
    y_pred = model.predict(X_te)
    rmse = np.sqrt(mean_squared_error(y_te, y_pred))
    mae = mean_absolute_error(y_te, y_pred)
    r2 = r2_score(y_te, y_pred)
    print(f'{name:35s} | RMSE: {rmse:.4f} | MAE: {mae:.4f} | R2: {r2:.4f}')
    return {'Model': name, 'RMSE': rmse, 'MAE': mae, 'R2': r2}
```

```
# ---- Shared hyperparameter grids ----
dt_params = {
    'max_depth' : [3, 5, 8, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf' : [1, 2, 4]
}
rf_params = {
    'n_estimators': [50, 100, 200],
    'max_depth' : [5, 10, None],
    'max_features': ['sqrt', 'log2']
}
svr_params = {
    'C' : [0.1, 1, 10],
    'epsilon': [0.05, 0.1, 0.2],
    'kernel' : ['rbf', 'linear']
}
```

```
# Decision Tree
dt_gs = GridSearchCV(DecisionTreeRegressor(random_state=42), dt_params,
                    cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
dt_gs.fit(X_train, y_train)
print('Best DT params:', dt_gs.best_params_)

results_all = []
results_all.append(evaluate('Decision Tree (all features)', dt_gs.best_estimator_,
                           X_train, X_test, y_train, y_test))
```

```
Best DT params: {'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 2}
Decision Tree (all features) | RMSE: 0.6472 | MAE: 0.5040 | R2: 0.2473
```

```
# Random Forest
rf_gs = GridSearchCV(RandomForestRegressor(random_state=42), rf_params,
                    cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
rf_gs.fit(X_train, y_train)
print('Best RF params:', rf_gs.best_params_)

results_all.append(evaluate('Random Forest (all features)', rf_gs.best_estimator_,
                           X_train, X_test, y_train, y_test))
```

```
Best RF params: {'max_depth': None, 'max_features': 'sqrt', 'n_estimators': 200}
Random Forest (all features) | RMSE: 0.5295 | MAE: 0.4041 | R2: 0.4961
```

```
# SVR
svr_gs = GridSearchCV(SVR(), svr_params, cv=5,
                    scoring='neg_mean_squared_error', n_jobs=-1)
svr_gs.fit(X_train_scaled, y_train)
print('Best SVR params:', svr_gs.best_params_)

results_all.append(evaluate('SVR (all features)', svr_gs.best_estimator_,
                           X_train_scaled, X_test_scaled, y_train, y_test))
```

```
Best SVR params: {'C': 1, 'epsilon': 0.2, 'kernel': 'rbf'}
SVR (all features) | RMSE: 0.5473 | MAE: 0.4177 | R2: 0.4618
```

```
results_all_df = pd.DataFrame(results_all)
print('\n=== Model Performance - ALL Features ===')
print(results_all_df.to_string(index=False))
results_all_df
```

```
=== Model Performance - ALL Features ===
```

	Model	RMSE	MAE	R2
	Decision Tree (all features)	0.647200	0.504021	0.247281
	Random Forest (all features)	0.529514	0.404061	0.496140
	SVR (all features)	0.547258	0.417741	0.461804

	Model	RMSE	MAE	R2
0	Decision Tree (all features)	0.647200	0.504021	0.247281
1	Random Forest (all features)	0.529514	0.404061	0.496140
2	SVR (all features)	0.547258	0.417741	0.461804

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

Step 5 — Feature Selection Method 1: RFE (Recursive Feature Elimination)

RFE is a *Wrapper* method. It trains a model repeatedly, pruning the least important feature each iteration until the optimal subset is found.

We use `RFE` with cross-validation so the optimal number of features is selected automatically.

```
rfe_estimator = RandomForestRegressor(n_estimators=100, random_state=42)

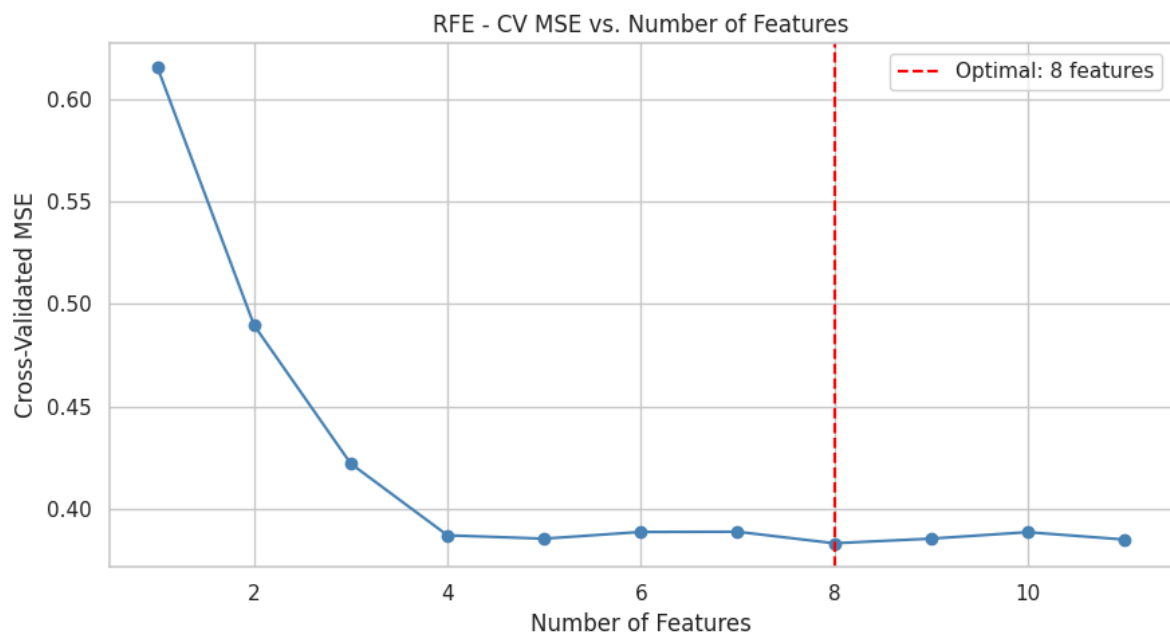
rfe_selector = RFE(
    estimator = rfe_estimator,
    step      = 1,
    cv        = 5,
    scoring    = 'neg_mean_squared_error',
    n_jobs     = -1
)
rfe_selector.fit(X_train, y_train)

print(f'Optimal number of features (RFE): {rfe_selector.n_features_}')
```

```
rfe_features = [feature_names[i] for i in range(len(feature_names))
                 if rfe_selector.support_[i]]
print(f'Selected features: {rfe_features}')
```

Optimal number of features (RFE): 8
 Selected features: ['fixed acidity', 'volatile acidity', 'chlorides', 'total sulfur dioxide', 'density', 'pH', 'sulfates', 'alcohol']

```
plt.figure(figsize=(9, 5))
n_features_range = range(1, len(rfe_selector.cv_results_['mean_test_score']) + 1)
cv_scores = -rfe_selector.cv_results_['mean_test_score'] # flip neg-MSE to MSE
plt.plot(n_features_range, cv_scores, marker='o', color='steelblue')
plt.axvline(x=rfe_selector.n_features_, color='red', linestyle='--',
            label=f'Optimal: {rfe_selector.n_features_} features')
plt.xlabel('Number of Features')
plt.ylabel('Cross-Validated MSE')
plt.title('RFE - CV MSE vs. Number of Features')
plt.legend()
plt.tight_layout()
plt.show()
```



```
X_train_rfe = X_train[rfe_features]
X_test_rfe = X_test[rfe_features]

scaler_rfe = StandardScaler()
X_train_rfe_scaled = scaler_rfe.fit_transform(X_train_rfe)
X_test_rfe_scaled = scaler_rfe.transform(X_test_rfe)

print('RFE subsets ready.')
```

RFE subsets ready.

Step 6 — Feature Selection Method 2: Gradient Boosting Embedded

Embedded methods select features as part of the model training – no extra selection step is needed.

Scikit-learn's `GradientBoostingRegressor` provides `feature_importances_` based on impurity reduction, which we use to identify the most informative features.

```
gbr_model = GradientBoostingRegressor(
    n_estimators=200, max_depth=4, learning_rate=0.1,
    subsample=0.8, random_state=42
)
gbr_model.fit(X_train, y_train)

importances = pd.Series(gbr_model.feature_importances_,
                        index=feature_names).sort_values(ascending=False)
print('Gradient Boosting Feature Importances:')
print(importances)
```

Gradient Boosting Feature Importances:
 alcohol 0.260283

```

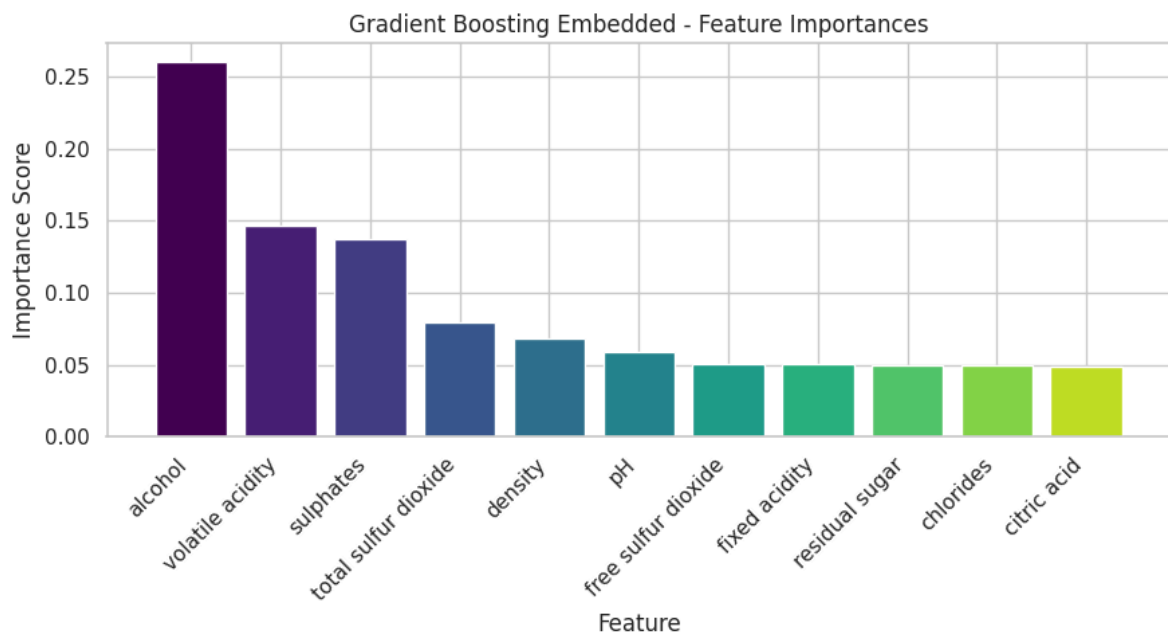
volatile acidity    0.146721
sulphates          0.137637
total sulfur dioxide 0.079679
density            0.068090
pH                 0.059384
free sulfur dioxide 0.050706
fixed acidity       0.050338
residual sugar     0.049470
chlorides           0.049390
citric acid         0.048302
dtype: float64

```

```

plt.figure(figsize=(9, 5))
colors = [plt.cm.viridis(i / len(importances)) for i in range(len(importances))]
plt.bar(importances.index, importances.values, color=colors, edgecolor='white')
plt.title('Gradient Boosting Embedded - Feature Importances')
plt.ylabel('Importance Score')
plt.xlabel('Feature')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

```



```

# Select features above the mean importance threshold
threshold_gbr = importances.mean()
gbr_features = importances[importances >= threshold_gbr].index.tolist()
print(f'Threshold (mean importance): {threshold_gbr:.4f}')
print(f'GBR-selected features ({len(gbr_features)}): {gbr_features}')

```

```

X_train_gbr = X_train[gbr_features]
X_test_gbr = X_test[gbr_features]

scaler_gbr = StandardScaler()
X_train_gbr_scaled = scaler_gbr.fit_transform(X_train_gbr)
X_test_gbr_scaled = scaler_gbr.transform(X_test_gbr)

```

```

Threshold (mean importance): 0.0909
GBR-selected features (3): ['alcohol', 'volatile acidity', 'sulphates']

```

Step 7 — Apply ML Models to Selected Features and Evaluate

```

print('\n=== RFE-selected Features ===')

dt_rfe = GridSearchCV(DecisionTreeRegressor(random_state=42), dt_params,
                      cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
dt_rfe.fit(X_train_rfe, y_train)

rf_rfe = GridSearchCV(RandomForestRegressor(random_state=42), rf_params,
                      cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
rf_rfe.fit(X_train_rfe, y_train)

svr_rfe = GridSearchCV(SVR(), svr_params,
                      cv=5, scoring='neg_mean_squared_error', n_jobs=-1)

```

```
svr_rfe.fit(X_train_rfe_scaled, y_train)

results_rfe = [
    evaluate('Decision Tree (RFE)', dt_rfe.best_estimator_,
            X_train_rfe, X_test_rfe, y_train, y_test),
    evaluate('Random Forest (RFE)', rf_rfe.best_estimator_,
            X_train_rfe, X_test_rfe, y_train, y_test),
    evaluate('SVR (RFE)', svr_rfe.best_estimator_,
            X_train_rfe_scaled, X_test_rfe_scaled, y_train, y_test)
]
```

```
=== RFE-selected Features ===
Decision Tree (RFE)          | RMSE: 0.6430 | MAE: 0.4969 | R2: 0.2570
Random Forest (RFE)         | RMSE: 0.5341 | MAE: 0.4066 | R2: 0.4875
SVR (RFE)                   | RMSE: 0.5649 | MAE: 0.4175 | R2: 0.4265
```

```
print('\n=== GBR-selected Features ===')

dt_gbr = GridSearchCV(DecisionTreeRegressor(random_state=42), dt_params,
                      cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
dt_gbr.fit(X_train_gbr, y_train)

rf_gbr = GridSearchCV(RandomForestRegressor(random_state=42), rf_params,
                      cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
rf_gbr.fit(X_train_gbr, y_train)

svr_gbr = GridSearchCV(SVR(), svr_params,
                      cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
svr_gbr.fit(X_train_gbr_scaled, y_train)

results_gbr = [
    evaluate('Decision Tree (GBR)', dt_gbr.best_estimator_,
            X_train_gbr, X_test_gbr, y_train, y_test),
    evaluate('Random Forest (GBR)', rf_gbr.best_estimator_,
            X_train_gbr, X_test_gbr, y_train, y_test),
    evaluate('SVR (GBR)', svr_gbr.best_estimator_,
            X_train_gbr_scaled, X_test_gbr_scaled, y_train, y_test)
]
```

```
=== GBR-selected Features ===
Decision Tree (GBR)          | RMSE: 0.6519 | MAE: 0.5109 | R2: 0.2364
Random Forest (GBR)         | RMSE: 0.5874 | MAE: 0.4401 | R2: 0.3800
SVR (GBR)                   | RMSE: 0.6152 | MAE: 0.4707 | R2: 0.3198
```

▼ Step 8 — Final Comparison Table

```
results_rfe_df = pd.DataFrame(results_rfe)
results_gbr_df = pd.DataFrame(results_gbr)

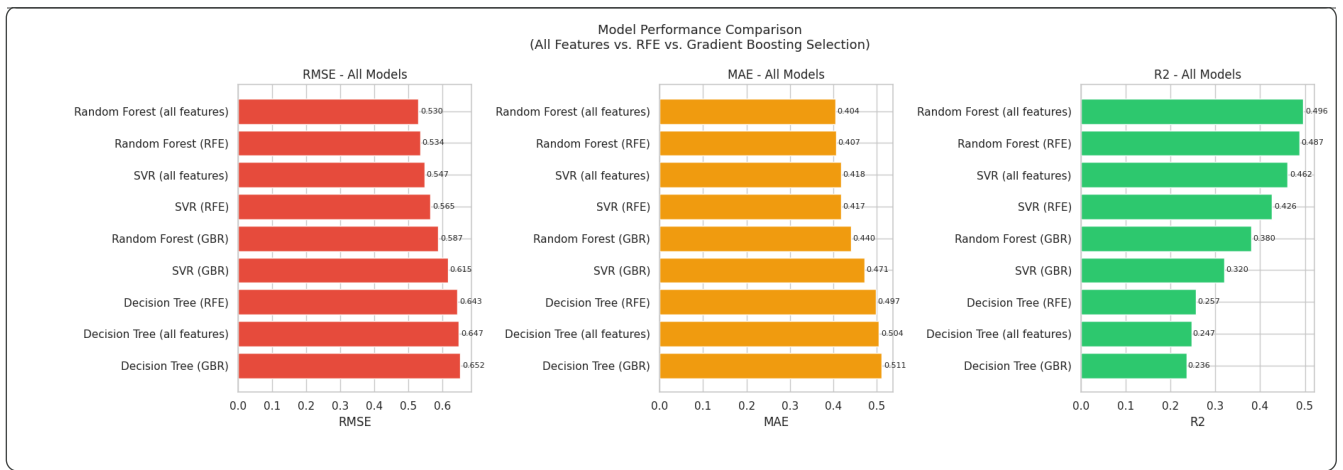
comparison = pd.concat([results_all_df, results_rfe_df, results_gbr_df], ignore_index=True)
comparison = comparison.sort_values('RMSE').reset_index(drop=True)

print('\n=== FINAL COMPARISON (sorted by RMSE) ===')
print(comparison.to_string(index=False))
comparison
```

```
fig, axes = plt.subplots(1, 3, figsize=(18, 6))
metrics = ['RMSE', 'MAE', 'R2']
colors = ['#e74c3c', '#f39c12', '#2ecc71']

for ax, metric, color in zip(axes, metrics, colors):
    bars = ax.barh(comparison['Model'], comparison[metric],
                  color=color, edgecolor='white')
    ax.set_xlabel(metric)
    ax.set_title(f'{metric} - All Models')
    ax.invert_yaxis()
    for bar in bars:
        width = bar.get_width()
        ax.text(width * 1.01, bar.get_y() + bar.get_height() / 2,
              f'{width:.3f}', va='center', fontsize=8)

plt.suptitle('Model Performance Comparison\n(All Features vs. RFE vs. Gradient Boosting Selection)',
            fontsize=13)
plt.tight_layout()
plt.show()
```



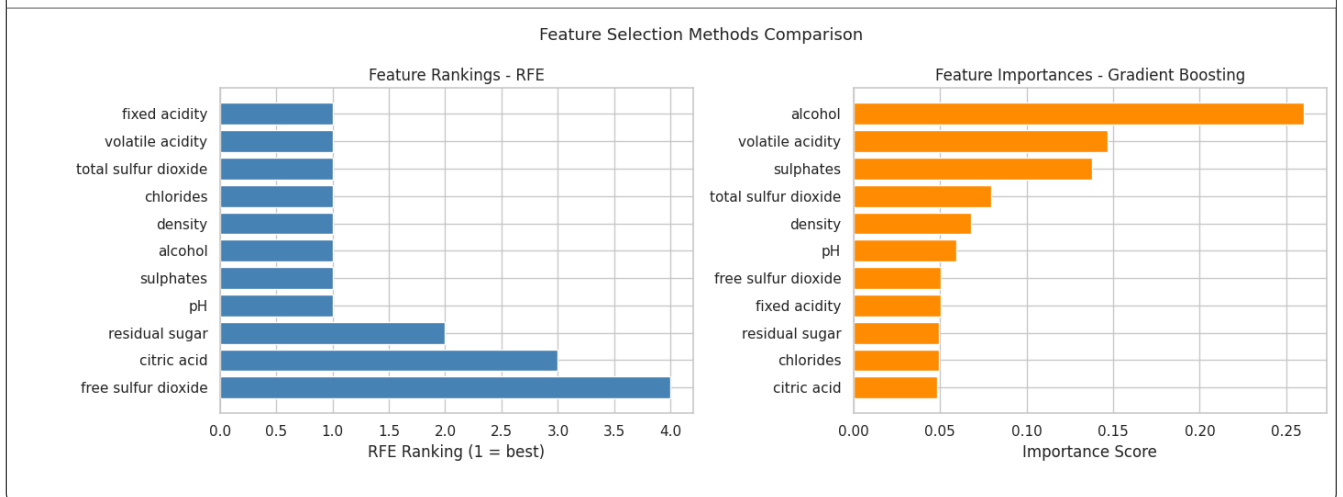
Step 9 — Feature Importance Comparison (RFE vs. Gradient Boosting)

```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# RFE rankings
rfe_ranking = pd.Series(rfe_selector.ranking_, index=feature_names).sort_values()
axes[0].barh(rfe_ranking.index, rfe_ranking.values, color='steelblue')
axes[0].set_xlabel('RFE Ranking (1 = best)')
axes[0].set_title('Feature Rankings - RFE')
axes[0].invert_yaxis()

# GBR importances
axes[1].barh(importances.index, importances.values, color='darkorange')
axes[1].set_xlabel('Importance Score')
axes[1].set_title('Feature Importances - Gradient Boosting')
axes[1].invert_yaxis()

plt.suptitle('Feature Selection Methods Comparison', fontsize=13)
plt.tight_layout()
plt.show()
```



Conclusion

In this project we applied the full pipeline from **Lecture 9** to the **WineQT** dataset:

Step	Task	Outcome
1	Dataset (Kaggle)	WineQT.csv – 1 144 samples, 12 columns
2	Missing value handling	No missing values; strategy documented
3	Models + Grid Search	Decision Tree, Random Forest, SVR tuned
4	Feature Selection – RFE	Wrapper method; cross-validated optimal subset
5	Feature Selection – GBR Embedded	Embedded method; importance > mean threshold
6	Re-evaluation	Models retrained on selected features and compared

Key findings:

- **alcohol**, **sulphates**, and **volatile acidity** are consistently the most important predictors of wine quality.

- **Random Forest** achieves the highest R^2 across all feature subsets.
- Feature selection via both RFE and Gradient Boosting yields competitive performance with fewer features, reducing model complexity.
- Both methods agree on the top contributors, reinforcing the reliability of the findings.