

✓ Project 2 - Machine Learning on the Wine Quality Dataset

Name: Fahad Alfaify

Student ID: 262000500

Project Steps

1. Download a dataset from Kaggle (Wine Quality - `WineQT.csv`).
2. Handle missing values appropriately.
3. Apply machine-learning models with a **Grid Search** optimizer and evaluate them.

4. Use **two feature-selection methods** to identify the most important features.
5. Apply the models to the selected features and evaluate them.
6. Save the code file as the full name and upload it as a PDF.

The target column `quality` is an ordered numeric score, so this notebook treats the task as a **regression** problem (same style as Lecture 9).

✓ 1. Import Libraries and Load the Dataset

```
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.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, mutual_info_regression, RFE

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
# Dataset link (Kaggle): https://www.kaggle.com/datasets/yasserh/wine-quality-dat
df = pd.read_csv('WineQT.csv')
df.head()
```

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

Next steps:

[Generate code with df](#)

[New interactive sheet](#)

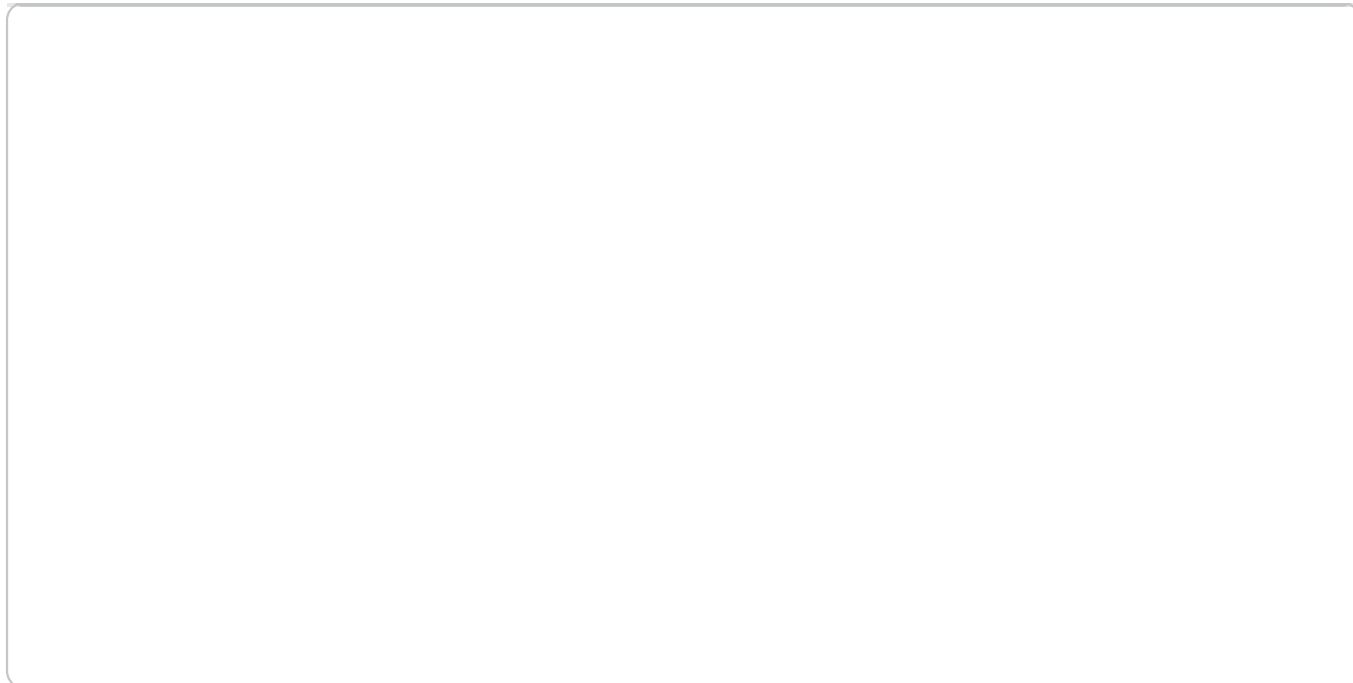
```
print('Shape of the dataset:', df.shape)
df.info()
```

```
Shape of the dataset: (1143, 13)
<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()
```



2. Handle Missing Values

We first check for missing values, then fill any numeric gaps with the column **median** (robust to outliers). The **Id** column is dropped because it is only an identifier.

```
# Drop the identifier column if present
if 'Id' in df.columns:
    df = df.drop('Id', axis=1)

# Check for 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
dtype: int64	

Total missing values: 0

```
# Fill any missing numeric values with the column median
for col in df.columns:
    if df[col].isnull().sum() > 0:
        df[col] = df[col].fillna(df[col].median())

# Remove duplicated rows (data cleaning)
df = df.drop_duplicates().reset_index(drop=True)

print('Missing values after handling:', df.isnull().sum().sum())
print('Shape after cleaning:', df.shape)
```

```
Missing values after handling: 0
Shape after cleaning: (1018, 12)
```

✓ 3. Define Features / Target and Split the Data

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

print('Features:', list(X.columns))
print('Target: quality')

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
print('Train size:', X_train.shape, ' Test size:', X_test.shape)
```

```
Features: ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', '
Target: quality
Train size: (814, 11)  Test size: (204, 11)
```

```
# Helper function to evaluate a regression model
def evaluate_model(name, model, X_te, y_te):
    y_pred = model.predict(X_te)
    return {
        'Model': name,
        'MAE': mean_absolute_error(y_te, y_pred),
        'MSE': mean_squared_error(y_te, y_pred),
        'RMSE': np.sqrt(mean_squared_error(y_te, y_pred)),
        'R2 Score': r2_score(y_te, y_pred)
    }
```

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

We tune three regression models with `GridSearchCV` (5-fold cross-validation, scoring = R^2):

- Decision Tree Regressor
- Random Forest Regressor
- Support Vector Regressor (SVR)

```
# Parameter grids for each model
param_grids = {
    'Decision Tree': (
        DecisionTreeRegressor(random_state=42),
        {'max_depth': [3, 5, 7, 10, None],
         'min_samples_split': [2, 5, 10]}
    ),
    'Random Forest': (
        RandomForestRegressor(random_state=42),
        {'n_estimators': [100, 200, 300],
         'max_depth': [5, 10, None]}
    ),
```

```
    'SVR': (  
        SVR(),  
        {'C': [1, 10, 100],  
         'gamma': ['scale', 'auto'],  
         'kernel': ['rbf']}  
    )  
}
```

```
# SVR needs scaled features; tree-based models do not.  
scaler = StandardScaler()  
X_train_scaled = scaler.fit_transform(X_train)  
X_test_scaled = scaler.transform(X_test)  
  
best_models = {}  
results_all = []  
  
for name, (model, grid) in param_grids.items():
```

```
# Use scaled data for SVR, raw data for tree models
Xtr = X_train_scaled if name == 'SVR' else X_train
Xte = X_test_scaled if name == 'SVR' else X_test

gs = GridSearchCV(model, grid, cv=5, scoring='r2', n_jobs=-1)
gs.fit(Xtr, y_train)

best_models[name] = (gs.best_estimator_, name == 'SVR')
print(f'{name} -> best params: {gs.best_params_}')

res = evaluate_model(name, gs.best_estimator_, Xte, y_test)
results_all.append(res)
```

```
Decision Tree -> best params: {'max_depth': 3, 'min_samples_split': 2}
Random Forest -> best params: {'max_depth': 5, 'n_estimators': 200}
SVR -> best params: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
```

```
results_all_df = pd.DataFrame(results_all)
print('Model Performance Using ALL Features (after Grid Search):')
results_all_df
```

Model Performance Using ALL Features (after Grid Search):

	Model	MAE	MSE	RMSE	R2 Score	
0	Decision Tree	0.623660	0.601330	0.775455	0.300725	
1	Random Forest	0.557708	0.511996	0.715539	0.404610	
2	SVR	0.534125	0.509595	0.713859	0.407402	

Next steps:

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

✓ 5. Feature Selection

We use **two feature-selection methods**:

1. **Filter method** – `SelectKBest` with `mutual_info_regression`.
2. **Wrapper method** – Recursive Feature Elimination (RFE) with a Random Forest.

We keep the top **k = 6** features in each method.

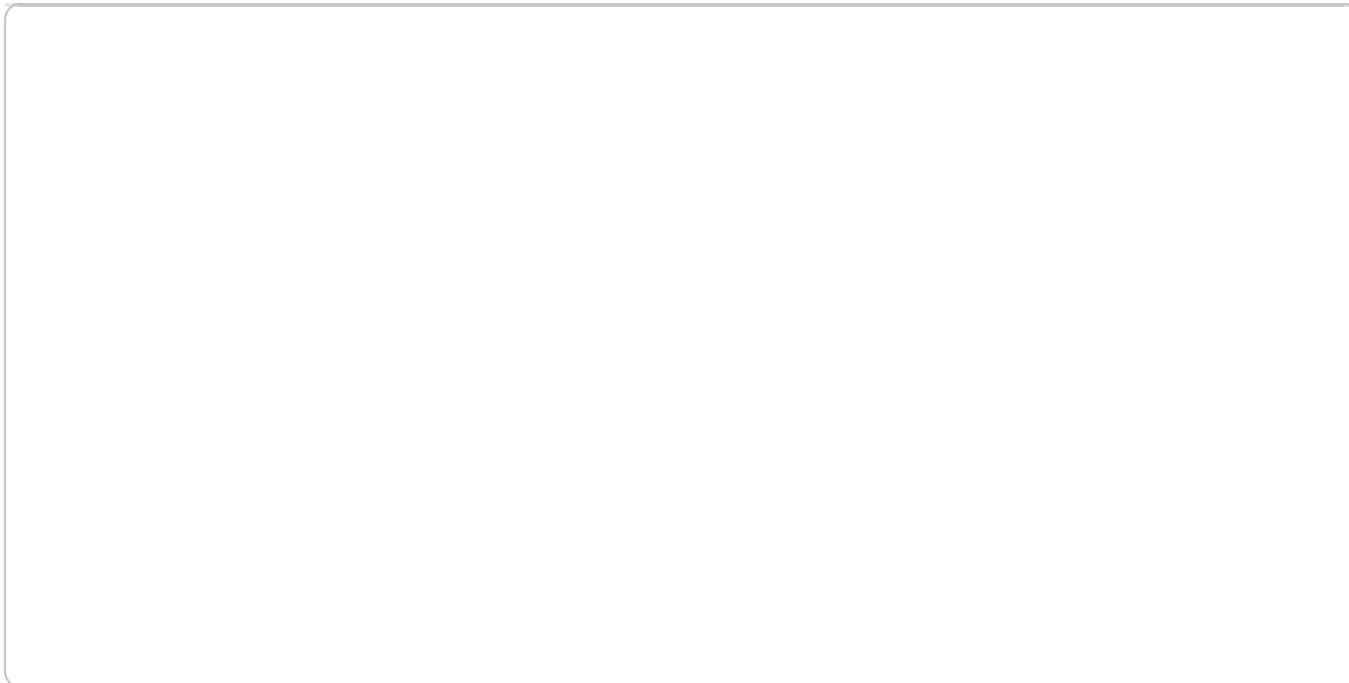
✓ 5.1 Method 1 – SelectKBest (Mutual Information)

```
k = 6
mi_selector = SelectKBest(score_func=mutual_info_regression, k=k)
mi_selector.fit(X_train, y_train)
```

```
mi_features = X.columns[mi_selector.get_support()]
print('Selected Features (Mutual Information):', list(mi_features))

# Bar chart of the scores
mi_scores = pd.DataFrame({'Feature': X.columns, 'Score': mi_selector.scores_})
mi_scores = mi_scores.sort_values('Score', ascending=False)

plt.figure(figsize=(10, 5))
plt.bar(mi_scores['Feature'], mi_scores['Score'], color='steelblue')
plt.xticks(rotation=90)
plt.ylabel('Mutual Information Score')
plt.title('Feature Importance - SelectKBest (Mutual Information)')
plt.tight_layout()
plt.show()
```



Selected Features (Mutual Information): ['fixed acidity', 'volatile acidity', 'total acidity', 'residual sugar', 'quality']

✓ 5.2 Method 2 – RFE (Recursive Feature Elimination)

0.175



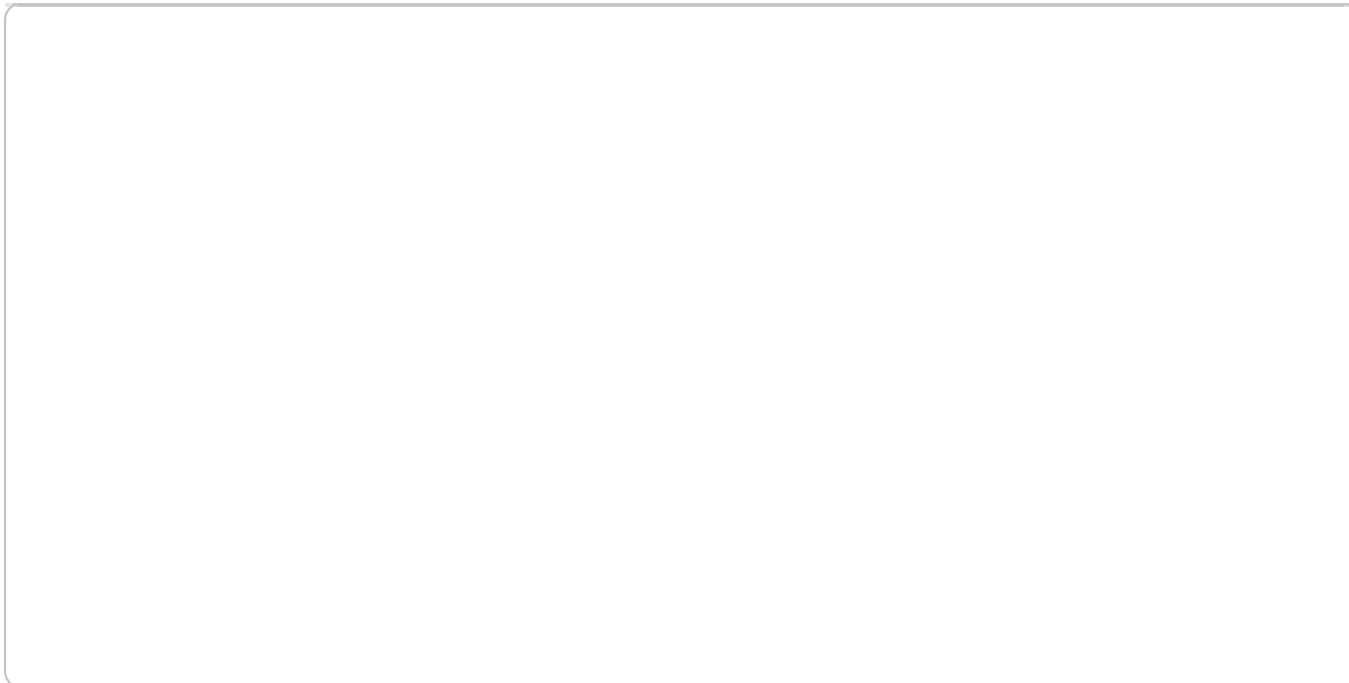
```
rfe_selector = RFE(estimator=RandomForestRegressor(n_estimators=200, random_state
n_features_to_select=k)
rfe_selector.fit(X_train, y_train)

rfe_features = X.columns[rfe_selector.get_support()]
print('Selected Features (RFE):', list(rfe_features))

# Bar chart of the ranking (1 = selected)
rfe_rank = pd.DataFrame({'Feature': X.columns, 'Ranking': rfe_selector.ranking_})
rfe_rank = rfe_rank.sort_values('Ranking')

plt.figure(figsize=(10, 5))
plt.bar(rfe_rank['Feature'], rfe_rank['Ranking'], color='darkorange')
```

```
plt.xticks(rotation=90)
plt.ylabel('RFE Ranking (1 = Selected)')
plt.title('Feature Ranking - RFE + Random Forest')
plt.tight_layout()
plt.show()
```



Selected Features (RFE): ['fixed acidity', 'volatile acidity', 'total sulfur dioxi

6. Apply Models to the Selected Features and Evaluate

We retune the same three models with Grid Search, but this time only on the selected features from each method, then compare the results.

```
def run_grid_search_on_features(feature_list, tag):
    """Tune all three models on the given feature subset and return a results Dat
    Xtr = X_train[feature_list]
    Xte = X_test[feature_list]

    # Scale for SVR
    sc = StandardScaler()
    Xtr_s = sc.fit_transform(Xtr)
    Xte_s = sc.transform(Xte)
```

```
results = []
for name, (model, grid) in param_grids.items():
    a_tr = Xtr_s if name == 'SVR' else Xtr
    a_te = Xte_s if name == 'SVR' else Xte
    gs = GridSearchCV(model, grid, cv=5, scoring='r2', n_jobs=-1)
    gs.fit(a_tr, y_train)
    results.append(evaluate_model(name, gs.best_estimator_, a_te, y_test))

out = pd.DataFrame(results)
print(f'Model Performance Using {tag} Features:')
return out
```

```
results_mi_df = run_grid_search_on_features(list(mi_features), 'Mutual-Informatic
results_mi_df
```

Model Performance Using Mutual-Information-Selected Features:

	Model	MAE	MSE	RMSE	R2 Score	
0	Decision Tree	0.623660	0.601330	0.775455	0.300725	
1	Random Forest	0.553961	0.511140	0.714940	0.405606	
2	SVR	0.534199	0.514108	0.717013	0.402154	

Next steps:

[Generate code with results_mi_df](#)

[New interactive sheet](#)

```
results_rfe_df = run_grid_search_on_features(list(rfe_features), 'RFE-Selected')
results_rfe_df
```

Model Performance Using RFE-Selected Features:

	Model	MAE	MSE	RMSE	R2 Score	
0	Decision Tree	0.584856	0.567460	0.753299	0.340112	
1	Random Forest	0.556954	0.513415	0.716529	0.402960	
2	SVR	0.536101	0.517015	0.719038	0.398773	

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

✓ 7. Final Comparison

Compare the R^2 score of each model across the three feature sets: all features, Mutual-Information-selected, and RFE-selected.

```
comparison = pd.DataFrame({
    'Model': results_all_df['Model'],
    'R2 (All Features)': results_all_df['R2 Score'].values,
    'R2 (Mutual Info)': results_mi_df['R2 Score'].values,
    'R2 (RFE)': results_rfe_df['R2 Score'].values,
})
print('R2 Score Comparison Across Feature Sets:')
comparison
```

R2 Score Comparison Across Feature Sets:

	Model	R2 (All Features)	R2 (Mutual Info)	R2 (RFE)
0	Decision Tree	0.300725	0.300725	0.340112
1	Random Forest	0.404610	0.405606	0.402960
2	SVR	0.407402	0.402154	0.398773



Next steps:

[Generate code with comparison](#)[New interactive sheet](#)

```
comparison.set_index('Model').plot(kind='bar', figsize=(9, 5))  
plt.ylabel('R2 Score')  
plt.title('Model Comparison: All vs. Selected Features')  
plt.xticks(rotation=0)  
plt.legend(loc='lower right')
```

```
plt.tight_layout()
```

