大數據、機器學習與程式交易之方法實作期末報告

VIN ROUGE

第九組
財管三 傅玉婷　風管三 許佳勛
財管三 詹雅智　風管三 歐予揚
財管三 陳尚格　風管三 龔　磊

紅酒中成分簡介

- 酒精：由糖份發酵後所得，為葡萄酒增添了芳醇的味道
- 殘糖：紅酒中甜度的來源，即是發酵結束後紅酒中剩餘的糖分
- 硫酸鹽：作為防腐劑，防止紅酒變質
- 固定酸度：葡萄中的酒石酸及蘋果酸屬於固定酸，用以保持紅酒的化學穩定性和色澤
- 揮發性酸度 :檸檬酸
- PH值：大多數紅酒的pH值介於2.9至3.9之間，影響紅酒的酸度

紅酒中成分簡介

- 游離二氧化硫：防止氧氣造成紅酒的衰敗及氧化
- 總二氧化硫
- 氯化物
- 密度

研究問題

- 利用紅酒的成分評斷紅酒的品質

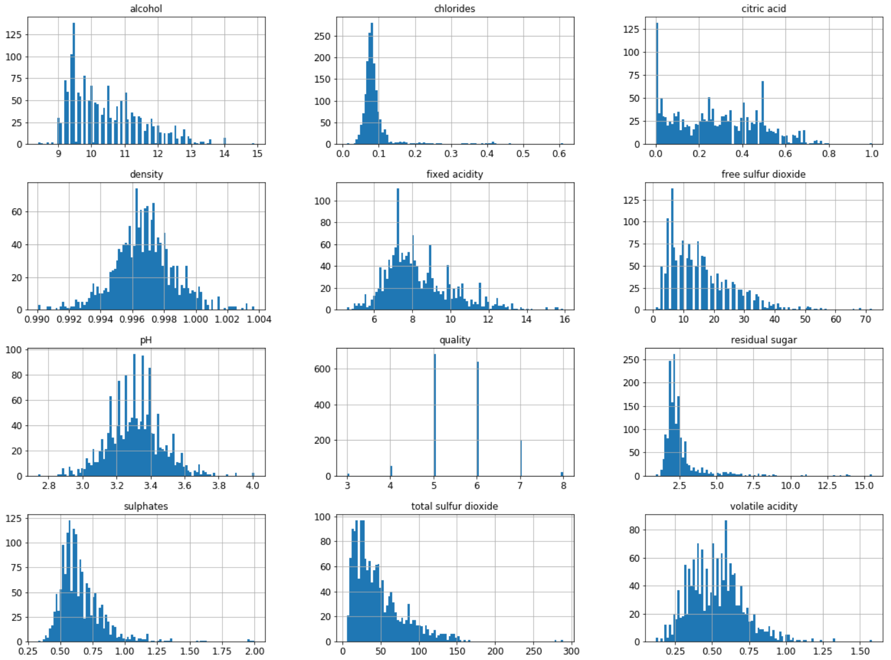
Setup

```
In [2]: import pandas as pd

        def load_csv_data(file):
            return pd.read_csv(file)

        wine = load_csv_data("winequality-red.csv")
        wine.head()
```

Out[2]:

	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
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

```
In [5]: %matplotlib inline
        import matplotlib.pyplot as plt
        wine.hist(bins=100, figsize=(20,15))
        plt.savefig("attribute_histogram_plots")
        plt.show()
```



In [7]:
```python
import numpy as np

# For illustration only. Sklearn has train_test_split()
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

In [8]:
```python
train_set, test_set = split_train_test(wine, 0.2)
print(len(train_set), "train +", len(test_set), "test")
```

```
1280 train + 319 test
```

In [12]:
```python
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(wine, test_size=0.2, random_state=42)
```

In [13]:
```python
test_set.head()
```

Out[13]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
803	7.7	0.56	0.08	2.50	0.114	14.0	46.0	0.9971	3.24	0.66	9.6	6
124	7.8	0.50	0.17	1.60	0.082	21.0	102.0	0.9960	3.39	0.48	9.5	5
350	10.7	0.67	0.22	2.70	0.107	17.0	34.0	1.0004	3.28	0.98	9.9	6
682	8.5	0.46	0.31	2.25	0.078	32.0	58.0	0.9980	3.33	0.54	9.8	5
1326	6.7	0.46	0.24	1.70	0.077	18.0	34.0	0.9948	3.39	0.60	10.6	6

In [14]:
```
wine= train_set.copy()
```

In [15]:
```
corr_matrix = wine.corr()
corr_matrix["quality"].sort_values(ascending=False)
```

Out[15]:
```
quality                 1.000000
alcohol                 0.472676
sulphates               0.242596
citric acid             0.216115
fixed acidity           0.122488
residual sugar          0.005425
pH                     -0.045185
free sulfur dioxide    -0.055860
chlorides              -0.126541
density                -0.167091
total sulfur dioxide   -0.200067
volatile acidity       -0.378372
Name: quality, dtype: float64
```

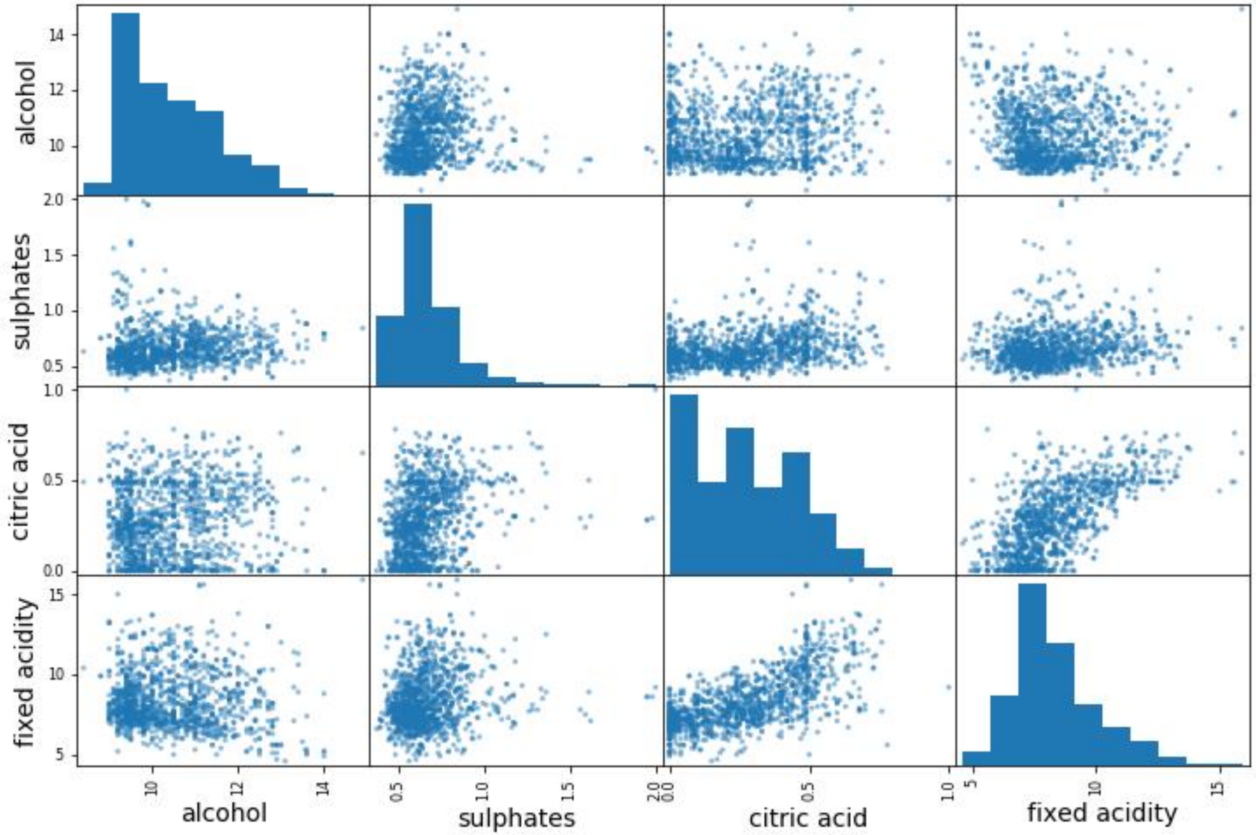
In [16]:
```
wine["sulfur_ratio"] = wine["free sulfur dioxide"]/wine["total sulfur dioxide"]
wine['acidity_ratio'] = wine["fixed acidity"]/wine["volatile acidity"]
wine["c"] = wine["pH"]/wine["alcohol"]
wine["sulphates_volatile acidity_ratio"] = wine["sulphates"]/wine["volatile acidity"]
wine["citric acid_chlorides_ratio"] = wine["citric acid"]/wine["chlorides"]
wine["alcohol_chlorides_ratio"] = wine["alcohol"]/wine["chlorides"]
```

In [17]:
```
corr_matrix = wine.corr()
corr_matrix["quality"].sort_values(ascendi
```

Out[17]:
```
quality                             1.000000
alcohol                             0.472676
sulphates_volatile acidity_ratio    0.382378
acidity_ratio                       0.336420
citric acid_chlorides_ratio         0.286235
alcohol_chlorides_ratio             0.274968
sulphates                           0.242596
citric acid                         0.216115
sulfur_ratio                        0.202364
fixed acidity                       0.122488
residual sugar                      0.005425
pH                                 -0.045185
free sulfur dioxide                -0.055860
chlorides                          -0.126541
density                            -0.167091
total sulfur dioxide               -0.200067
volatile acidity                   -0.378372
c                                  -0.486091
Name: quality, dtype: float64
```

In [

ty"]


alcohol
sulphates
citric acid
fixed acidity
14
12
10
2.0
1.5
1.0
0.5
1.0
0.5
0.0
15
10
5
10
12
14
0.5
1.0
1.5
2.0
0.0
0.5
1.0
5
10
15
alcohol
sulphates
citric acid
fixed acidity

In [20]:
```
wine = train_set.drop("quality", axis=1) # drop labels for training set
wine_labels = train_set["quality"].copy()
```

In [21]:
```
sample_incomplete_rows = wine[wine.isnull().any(axis=1)].head()
sample_incomplete_rows
```

Out[21]:

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol

In [22]:
```
try:
    from sklearn.impute import SimpleImputer # Scikit-Learn 0.20+
except ImportError:
    from sklearn.preprocessing import Imputer as SimpleImputer

imputer = SimpleImputer(strategy="median")
```

In [23]:
```
imputer.fit(wine)
```

Out[23]:
```
Imputer(axis=0, copy=True, missing_values='NaN', strategy='median', verbose=0)
```

In [24]:
```
X = imputer.transform(wine)
wine_tr = pd.DataFrame(X, columns=wine.columns,
                       index = list(wine.index.values))
```

In [25]:
```
wine_tr.loc[sample_incomplete_rows.index.values]
```

Out[25]:

fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol

In [26]:
```
wine_tr = pd.DataFrame(X, columns=wine.columns)
wine_tr.head()
```

Out[26]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	8.7	0.690	0.31	3.0	0.086	23.0	81.0	1.00020	3.48	0.74	11.6
1	6.1	0.210	0.40	1.4	0.066	40.5	165.0	0.99120	3.25	0.59	11.9
2	10.9	0.390	0.47	1.8	0.118	6.0	14.0	0.99820	3.30	0.75	9.8
3	8.8	0.685	0.26	1.6	0.088	16.0	23.0	0.99694	3.32	0.47	9.4
4	8.4	1.035	0.15	6.0	0.073	11.0	54.0	0.99900	3.37	0.49	9.9

In [27]:
```
from sklearn.base import BaseEstimator, TransformerMixin

# column index
fixed_acidity_ix, volatile_acidity_ix, citric_acid_ix, chlorides_ix, sulphates_ix = 0, 1, 2, 4, 9

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_citric_acid_chlorides_ratio = True):
        self.add_citric_acid_chlorides_ratio = add_citric_acid_chlorides_ratio
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        acidity_ratio = X[:, fixed_acidity_ix] / X[:, volatile_acidity_ix]
        sulphates_volatile_acidity_ratio = X[:, sulphates_ix] / X[:, volatile_acidity_ix]
        if self.add_citric_acid_chlorides_ratio:
            citric_acid_chlorides_ratio = X[:, citric_acid_ix] / X[:, chlorides_ix]
            return np.c_[X, acidity_ratio, sulphates_volatile_acidity_ratio,
                            citric_acid_chlorides_ratio]
        else:
            return np.c_[X, acidity_ratio, sulphates_volatile_acidity_ratio]

attr_adder = CombinedAttributesAdder(add_citric_acid_chlorides_ratio=False)
wine_extra_attribs = attr_adder.transform(wine.values)
```

In [28]:
```
wine_extra_attribs = pd.DataFrame(
    wine_extra_attribs,
    columns=list(wine.columns)+["acidity_ratio", "sulphates_volatile_acidity_ratio"])
wine_extra_attribs.head()
```

Out[28]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	acidity_ratio	sulphates_volatile_acidity_ratio
0	8.7	0.690	0.31	3.0	0.086	23.0	81.0	1.00020	3.48	0.74	11.6	12.608696	1.072464
1	6.1	0.210	0.40	1.4	0.066	40.5	165.0	0.99120	3.25	0.59	11.9	29.047619	2.809524
2	10.9	0.390	0.47	1.8	0.118	6.0	14.0	0.99820	3.30	0.75	9.8	27.948718	1.923077
3	8.8	0.685	0.26	1.6	0.088	16.0	23.0	0.99694	3.32	0.47	9.4	12.846715	0.686131
4	8.4	1.035	0.15	6.0	0.073	11.0	54.0	0.99900	3.37	0.49	9.9	8.115942	0.473430

In [29]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

wine_tr = num_pipeline.fit_transform(wine)
```

```
In [30]: wine_tr

Out[30]: array([[ 0.21833164,  0.88971201,  0.19209222, ..., -0.60592788,
                 -0.47669384,  0.08693868],
                [-1.29016623, -1.78878251,  0.65275338, ...,  1.21726382,
                  1.77377737,  1.00582301],
                [ 1.49475291, -0.78434707,  1.01104539, ...,  1.095388  ,
                  0.62532933,  0.2285152 ],
                ...,
                [-0.65195559,  0.49909822, -1.08752211, ..., -0.71636823,
                 -0.7377447 , -0.97018322],
                [-0.24582155, -1.84458448,  0.39683051, ...,  2.37650605,
                  3.31611462,  1.16329163],
                [-1.46422367, -1.34236676, -0.06383064, ...,  0.21381953,
                  0.54628966,  0.28236718]])

In [31]: num_attribs = list(wine)
         wine_prepared = num_pipeline.fit_transform(wine)

In [32]: wine_prepared.shape

Out[32]: (1279, 14)
```

Models

SVR
Random Forest
Linear Regression

1.

SVR

In [57]:
```
from sklearn.svm import SVR

svr_reg = SVR(kernel="linear")
svr_reg.fit(wine_prepared, wine_labels)
```

Out[57]:
```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
  kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

In [58]:
```
wine_predictions = svr_reg.predict(wine_prepared)
svr_mse = mean_squared_error(wine_labels, wine_predictions)
svr_rmse = np.sqrt(svr_mse)
svr_rmse
```

Out[58]:
```
0.6578738210578761
```

In [59]:
```
final_model = svr_reg

X_test = test_set.drop('quality', axis=1)
y_test = test_set["quality"].copy()

X_test_prepared = num_pipeline.fit_transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

In [60]:
```
final_predictions
```

Out[60]:
```
array([5.2106567 , 4.9463886 , 5.62103214, 5.2979142 , 5.60017076,
       5.10113022, 4.95252974, 5.0676929 , 5.64823137, 5.61821471,
       6.06119103, 5.19622758, 5.38149932, 5.07105862, 5.29468298,
       6.45887704, 5.08451824, 5.49124183, 6.56867462, 5.26608284,
       5.40218388, 5.03450651, 5.93809386, 6.3500509 , 5.25271369,
       5.31058552, 6.46757107, 5.2974069 , 5.21298527, 6.34559402,
       5.13976537, 5.32273156, 5.60614376, 5.34912516, 5.32820561,
       4.91155736, 6.13345603, 5.6346516 , 5.6076249 , 6.08225541,
       5.50660038, 5.09294911, 6.13234414, 5.10834518, 5.74931251,
       5.78591251, 6.48398764, 5.5193933 , 5.243782  , 5.46661775,
       5.23493117, 5.03376404, 5.55756485, 6.38738892, 4.87387565,
       4.85873903, 5.96764009, 5.44490261, 5.73287068, 5.17003608,
       5.42998275, 5.87833925, 5.08210097, 5.18596899, 6.50655778,
       5.30088815, 6.36471818, 5.2025488 , 6.45324016, 5.20847886,
       6.41434991, 4.77568387, 5.66931051, 5.70798781, 6.14013769,
       5.18989095, 6.82045528, 5.8225108 , 6.15204361, 6.55227946,
```

In [61]:
```
final_rmse
```

Out[61]:
```
0.6299629040488907
```

交叉驗證（Cross Validation）

In [56]:
```python
from sklearn.model_selection import cross_val_score
svr_scores = cross_val_score(svr_reg, wine_prepared, wine_labels,
                             scoring="neg_mean_squared_error", cv=10)
svr_rmse_scores = np.sqrt(-svr_scores)
display_scores(svr_rmse_scores)
```

```
Scores: [0.6357058  0.72804982 0.66052214 0.62053514 0.67279226 0.62327226
 0.62173735 0.67546618 0.50332918 0.61555172]
Mean: 0.6356961855178536
Standard deviation: 0.05532556101518299
```

網格搜索（Grid Search）

In [43]:
```
from sklearn.model_selection import GridSearchCV

parameters = {'kernel':('linear','rbf'), 'C':[1, 10, 100], 'gamma':[0.125, 0.25, 0.5 ,1, 2, 4]}

svm_reg = SVR(kernel="linear")

grid_search = GridSearchCV(svm_reg, parameters, cv=5,
                           scoring='neg_mean_squared_error', return_train_score=True)
grid_search.fit(wine_prepared, wine_labels)
```

Out[43]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
  kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'kernel': ('linear', 'rbf'), 'C': [1, 10, 100], 'gamma': [0.125, 0.25, 0.5, 1, 2, 4]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='neg_mean_squared_error', verbose=0)
```

In [44]:
```
grid_search.best_estimator_
```

Out[44]:
```
SVR(C=1, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.125,
  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

網格搜索（Grid Search）

In [66]:
```
from sklearn.svm import SVR

Svr_reg = grid_search.best_estimator_
Svr_reg.fit(wine_prepared, wine_labels)
wine_predictions = Svr_reg.predict(wine_prepared)
Svr_mse = mean_squared_error(wine_labels, wine_predictions)
Svr_rmse = np.sqrt(svr_mse)
Svr_rmse
```

Out[66]:
```
0.509766885671461
```

網格搜索（Grid Search）

In [69]:
```
final_model = grid_search.best_estimator_

X_test = test_set.drop('quality', axis=1)
y_test = test_set["quality"].copy()

X_test_prepared = num_pipeline.fit_transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

In [70]:
```
final_predictions
```

Out[70]:
```
array([5.4910721 , 5.0543006 , 5.72279344, 5.31923231, 5.75131126,
       5.05902878, 4.92111104, 4.91804081, 5.82860687, 5.61504456,
       6.74146618, 5.1265742 , 5.73685687, 5.15448221, 5.38842549,
       6.46396182, 5.12526614, 5.54498761, 6.88510682, 4.92623107,
       4.92940341, 5.22442316, 5.29512663, 6.02383691, 5.47161376,
       5.93185614, 6.16095167, 5.18741448, 5.08934549, 6.17131731,
       5.09990855, 5.1460432 , 5.66539649, 5.20316388, 5.74983151,
       5.03119235, 6.40936384, 5.85403264, 5.55520372, 6.11958158,
       5.5608377 , 5.17133468, 6.25739337, 5.0746152 , 5.80067184,
       5.90581548, 6.73530586, 5.69148242, 5.1164224 , 5.58912592,
       5.16672494, 5.12320866, 5.83158914, 6.9765507 , 4.95014995,
       5.07098094, 5.95227187, 5.95806407, 5.88498128, 5.13313513,
       5.67048659, 6.04301937, 5.0967085 , 5.1236814 , 6.45262897,
       5.48477202, 6.59455097, 5.53495927, 6.63510306, 5.31524714,
       6.12883453, 5.01628839, 5.71446368, 5.7367934 , 5.99711913,
       5.00514539, 6.70616729, 5.01535802, 5.57648045, 6.28223092,
       5.16005775, 6.77117832, 5.30596628, 5.53603404, 6.22900968,
       6.13762288, 5.02014911, 6.09282268, 6.39110032, 5.02982384,
       6.19543224, 5.34289422, 5.04611402, 5.46782739, 5.36337818,
```

In [71]:
```
final_rmse
```

Out[71]:
```
0.5872263148220179
```

	Grid search 前	Grid search 後
svr_rmse	0.6578	0.5097
final_rmse	0.6356	0.5872

2.
Random Forest

一般情形

In [42]:
```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=10, random_state=42)
forest_reg.fit(wine_prepared, wine_labels)
```

Out[42]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=42, verbose=0, warm_start=False)
```

In [43]:
```
wine_predictions = forest_reg.predict(wine_prepared)
forest_mse = mean_squared_error(wine_labels, wine_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[43]:
```
0.24371698168540737
```

交叉驗證（Cross Validation）

In [47]:
```python
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_reg, wine_prepared, wine_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [0.64086124 0.69619502 0.63854082 0.58229073 0.62998016 0.65293807
 0.553328   0.70055781 0.47145055 0.59167452]
Mean: 0.6157816923648732
Standard deviation: 0.06543484441794419
```

網格搜索（Grid Search）

In [49]:
```
from sklearn.model_selection import GridSearchCV

param_grid = [
    # try 12 (3x4) combinations of hyperparameters
    {'n_estimators': [3, 10, 30,], 'max_features': [2, 4, 6, 8]},
    # then try 6 (2x3) combinations with bootstrap set as False
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
# train across 5 folds, that's a total of (12+6)*5=90 rounds of training
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error', return_train_score=True)
grid_search.fit(wine_prepared, wine_labels)
```

Out[49]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=42, verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]}, {'boots
trap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='neg_mean_squared_error', verbose=0)
```

網格搜索（Grid Search）

```
In [51]: cvres = grid_search.cv_results_
         for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
             print(np.sqrt(-mean_score), params)
```

```
0.7192574284434768 {'max_features': 2, 'n_estimators': 3}
0.6373260697285106 {'max_features': 2, 'n_estimators': 10}
0.6226427181605234 {'max_features': 2, 'n_estimators': 30}
0.6674480695525865 {'max_features': 4, 'n_estimators': 3}
0.6147839809533946 {'max_features': 4, 'n_estimators': 10}
0.6094262872716905 {'max_features': 4, 'n_estimators': 30}
0.7007820445006676 {'max_features': 6, 'n_estimators': 3}
0.6337460762262236 {'max_features': 6, 'n_estimators': 10}
0.6147218026250044 {'max_features': 6, 'n_estimators': 30}
0.7081809690642888 {'max_features': 8, 'n_estimators': 3}
0.6247376306217943 {'max_features': 8, 'n_estimators': 10}
0.6072184724658798 {'max_features': 8, 'n_estimators': 30}
0.6895978036105577 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
0.6293511557531692 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
0.6755982364928798 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
0.6212235398580833 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
0.6871368620247045 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
0.629326308694225 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

網格搜索（Grid Search）

In [54]:
```
final_model = grid_search.best_estimator_

X_test = test_set.drop('quality', axis=1)
y_test = test_set["quality"].copy()

X_test_prepared = num_pipeline.fit_transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

In [56]:
```
final_predictions
```

Out[56]:
```
array([5.3       , 5.13333333, 5.63333333, 5.13333333, 5.96666667,
       5.1       , 5.1       , 4.5       , 6.        , 5.96666667,
       6.53333333, 5.23333333, 5.63333333, 5.1       , 5.36666667,
       6.43333333, 5.33333333, 5.7       , 6.6       , 5.03333333,
       4.86666667, 5.5       , 5.33333333, 6.        , 5.56666667,
       5.83333333, 6.43333333, 5.4       , 5.1       , 6.16666667,
       5.23333333, 5.36666667, 5.73333333, 5.46666667, 5.63333333,
       5.13333333, 6.26666667, 6.16666667, 5.26666667, 5.56666667,
       5.2       , 5.23333333, 6.36666667, 5.1       , 5.63333333,
       5.56666667, 6.6       , 5.43333333, 5.23333333, 5.63333333,
       5.03333333, 5.2       , 5.9       , 6.8       , 5.13333333,
       5.2       , 6.        , 5.9       , 5.46666667, 5.26666667,
       5.76666667, 5.96666667, 5.3       , 5.16666667, 6.73333333,
       5.43333333, 6.56666667, 5.43333333, 6.53333333, 5.36666667,
       5.96666667, 5.06666667, 5.8       , 5.5       , 6.06666667,
       4.96666667, 6.63333333, 5.33333333, 5.86666667, 6.5       ,
       5.26666667, 6.83333333, 5.16666667, 5.26666667, 5.66666667,
       6.23333333, 5.06666667, 5.83333333, 6.4       , 5.3       ,
       6.13333333, 5.53333333, 4.96666667, 5.13333333, 5.33333333,
       5.4       , 5.06666667, 5.96666667, 4.66666667, 5.53333333,
       5.06666667, 5.03333333, 5.73333333, 6.36666667, 5.5       ,
```

In [57]:
```
final_rmse
```

Out[57]:
```
0.5872990199965338
```

3.
Linear Regression

```
In [31]: from sklearn.linear_model import LinearRegression

         lin_reg = LinearRegression()
         lin_reg.fit(wine_prepared, wine_labels)

Out[31]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [32]: # let's try the full preprocessing pipeline on a few training instances
         some_data = wine.iloc[:5]
         some_labels = wine_labels.iloc[:5]
         some_data_prepared = num_pipeline.transform(some_data)

         print("Predictions:", lin_reg.predict(some_data_prepared))
         print("Labels:", list(some_labels))

Predictions: [5.70248099 6.04222528 5.67103903 5.10814467 4.80454583]
Labels: [6, 6, 6, 5, 5]
```

In [33]:
```
from sklearn.metrics import mean_squared_error

wine_predictions = lin_reg.predict(wine_prepared)
lin_mse = mean_squared_error(wine_labels, wine_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out[33]:
```
0.6510118225404404
```

In [34]:
```
from sklearn.metrics import mean_absolute_error

lin_mae = mean_absolute_error(wine_labels, wine_predictions)
lin_mae
```

Out[34]:
```
0.49947401745314934
```

In [35]:
```
final_model = lin_reg

X_test = test_set.drop('quality', axis=1)
y_test = test_set["quality"].copy()

X_test_prepared = num_pipeline.fit_transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

In [36]:
```
final_predictions
```

Out[36]:
```
array([5.32847131, 5.00996722, 5.69223701, 5.43005255, 5.68472085,
       5.23970017, 4.99290139, 5.07585415, 5.77593875, 5.66067669,
       6.04990652, 5.22001009, 5.53605794, 5.23948605, 5.41812559,
       6.40602976, 5.10661285, 5.5567362 , 6.54525118, 5.32687526,
       5.32211035, 5.18472542, 5.88328741, 6.28922889, 5.33364881,
       5.43610721, 6.35465659, 5.30762198, 5.1771224 , 6.16721335,
       5.21900716, 5.44698054, 5.75605374, 5.39175963, 5.45144927,
       4.98866114, 6.16325005, 5.68453196, 5.62780589, 6.12707259,
       5.54316951, 5.21287501, 6.2057751 , 5.14305715, 5.85195664,
       5.83666934, 6.44930838, 5.63674665, 5.11688107, 5.52616358,
       5.20132329, 5.04784888, 5.53915679, 6.31006995, 4.93988594,
       4.96381788, 5.99675206, 5.38908488, 5.79021138, 5.24083139,
       5.57771606, 5.93032765, 5.24207949, 5.28282488, 6.43851765,
       5.38827815, 6.29335715, 5.27667348, 6.45026079, 5.29709479,
       6.34964133, 4.7203089 , 5.80460326, 5.76818275, 6.12141758,
       5.27593295, 6.69248419, 5.85727711, 6.05895817, 6.411229  ,
       5.27800801, 6.42487982, 5.43767391, 5.6540473 , 5.68869137,
       6.38372194, 5.27789988, 5.82504077, 6.30636083, 5.19429432,
       6.04799061, 5.64150782, 5.74890402, 5.89092357, 5.16986564,
       5.73563468, 5.13670626, 5.68928997, 4.95109922, 5.48801781,
       5.04572503, 5.11917556, 5.78248621, 5.6930024 , 5.42420134,
       6.11899545, 5.72823355, 5.40563171, 6.03988128, 5.2256793 ,
       6.62534397, 5.22454458, 6.1786867 , 4.67707492, 5.8100759 ,
       5.97236542, 6.07865402, 5.47341996, 4.99015957, 5.78068566,
       6.15156422, 5.29367349, 5.77121159, 5.36002283, 5.38789113,
```

In [37]:
```
final_rmse
```

Out[37]:
```
0.6231238739209931
```

交叉驗證（Cross Validation）

In [38]:

```python
from sklearn.model_selection import cross_val_score

def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

lin_scores = cross_val_score(lin_reg, wine_prepared, wine_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
Scores: [0.62960105 0.71841434 0.69363621 0.70150314 0.68016265 0.67488408
 0.66711484 0.68068173 0.4986116  0.64727016]
Mean: 0.659187980463089
Standard deviation: 0.0587352795160096
```

網格搜索（Grid Search）

In [39]:
```python
from sklearn.model_selection import GridSearchCV
param_grid =  { 'fit_intercept' : [True,False] ,'normalize' :[ True,False]}
grid_search = GridSearchCV(lin_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error', return_train_score=True)
grid_search.fit(wine_prepared, wine_labels)
```

Out[39]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'fit_intercept': [True, False], 'normalize': [True, False]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='neg_mean_squared_error', verbose=0)
```

In [40]:
```python
grid_search.best_estimator_
```

Out[40]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=True)
```

網格搜索（Grid Search）

In [41]:
```python
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
0.6643025610657023 {'fit_intercept': True, 'normalize': True}
0.6643025610657023 {'fit_intercept': True, 'normalize': False}
5.7301986610454705 {'fit_intercept': False, 'normalize': True}
5.7301986610454705 {'fit_intercept': False, 'normalize': False}
```

In [42]:
```python
pd.DataFrame(grid_search.cv_results_)
```

Out[42]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_fit_intercept	param_normalize	params	split0_test_score	split1_test_score	split2_
0	0.005258	0.004755	0.000835	0.000526	True	True	{'fit_intercept': True, 'normalize': True}	-0.463220	-0.485396	
1	0.002153	0.001246	0.000324	0.000121	True	False	{'fit_intercept': True, 'normalize': False}	-0.463220	-0.485396	
2	0.001336	0.000395	0.000298	0.000086	False	True	{'fit_intercept': False, 'normalize': True}	-32.912122	-32.330829	-
3	0.001067	0.000082	0.000256	0.000011	False	False	{'fit_intercept': False, 'normalize': False}	-32.912122	-32.330829	-

4 rows × 22 columns

網格搜索（Grid Search）

In [43]:
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=True)
lin_reg.fit(wine_prepared, wine_labels)
wine_predictions = lin_reg.predict(wine_prepared)
lin_mse = mean_squared_error(wine_labels, wine_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out[43]:
```
0.6510118225404404
```

網格搜索（Grid Search）

In [44]:
```
final_model = grid_search.best_estimator_

X_test = test_set.drop('quality', axis=1)
y_test = test_set["quality"].copy()

X_test_prepared = num_pipeline.fit_transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

In [45]:
```
final_predictions
```

Out[45]:
```
array([5.32847131, 5.00996722, 5.69223701, 5.43005255, 5.68472085,
       5.23970017, 4.99290139, 5.07585415, 5.77593875, 5.66067669,
       6.04990652, 5.22001009, 5.53605794, 5.23948605, 5.41812559,
       6.40602976, 5.10661285, 5.5567362 , 6.54525118, 5.32687526,
       5.32211035, 5.18472542, 5.88328741, 6.28922889, 5.33364881,
       5.43610721, 6.35465659, 5.30762198, 5.1771224 , 6.16721335,
       5.21900716, 5.44698054, 5.75605374, 5.39175963, 5.45144927,
       4.98866114, 6.16325005, 5.68453196, 5.62780589, 6.12707259,
       5.54316951, 5.21287501, 6.2057751 , 5.14305715, 5.85195664,
       5.83666934, 6.44930838, 5.63674665, 5.11688107, 5.52616358,
       5.20132329, 5.04784888, 5.53915679, 6.31006995, 4.93988594,
       4.96381788, 5.99675206, 5.38908488, 5.79021138, 5.24083139,
       5.57771606, 5.93032765, 5.24207949, 5.28282488, 6.43851765,
       5.38827815, 6.29335715, 5.27667348, 6.45026079, 5.29709479,
       6.34964133, 4.7203089 , 5.80460326, 5.76818275, 6.12141758,
       5.27593295, 6.69248419, 5.85727711, 6.05895817, 6.411229  ,
       5.27800801, 6.42487982, 5.43767391, 5.6540473 , 5.68869137,
       6.38372194, 5.27789988, 5.82504077, 6.30636083, 5.19429432,
       6.04799061, 5.64150782, 5.74890402, 5.89092357, 5.16986564,
       5.73563468, 5.13670626, 5.68928997, 4.95109922, 5.48801781,
       5.04572503, 5.11917556, 5.78248621, 5.6930024 , 5.42420134,
       6.11899545, 5.72823355, 5.40563171, 6.03988128, 5.2256793 ,
       6.62534397, 5.22454458, 6.1786867 , 4.67707492, 5.8100759 ,
       5.97236542, 6.07865402, 5.47341996, 4.99015957, 5.78068566,
       6.15156422, 5.29367349, 5.77121159, 5.36002283, 5.38789113,
```

In [46]:
```
final_rmse
```

Out[46]:
```
0.6231238739209932
```

	Grid search 前	Grid search 後
lin_rmse	0.6510	0.6510
final_rmse	0.6231	0.6231

SVR	0.5872
Random Forest	0.5873
Linear Regression	0.6231

感謝聆聽