```
import warnings

warnings.filterwarnings('ignore')

import pandas as pd

import numpy as np

df=pd.read_csv('WineQT.csv')

df
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	Id
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	0
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5	1
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5	2
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6	3
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	4
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1138	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6	1592
1139	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	9.5	6	1593
1140	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5	1594
1141	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6	1595
1142	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5	1597

EDA

```
df.shape
```

```
(1143, 13)
```

```
df.isnull().sum().any()
```

```
False
```

```
df.head(10)
```

	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
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5	5
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	5	6
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	7	7
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	7	8

```
print(df.describe())
```

```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1143.000000       1143.000000  1143.000000     1143.000000
mean        8.311111          0.531339     0.268364        2.532152
std         1.747595          0.179633     0.196686        1.355917
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.392500     0.090000        1.900000
```

```
50%         7.900000          0.520000     0.250000        2.200000
75%         9.100000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

          chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1143.000000          1143.000000           1143.000000  1143.000000
mean      0.086933            15.615486             45.914698     0.996730
std       0.047267            10.250486             32.782130     0.001925
min       0.012000             1.000000              6.000000     0.990070
25%       0.070000             7.000000             21.000000     0.995570
50%       0.079000            13.000000             37.000000     0.996680
75%       0.090000            21.000000             61.000000     0.997845
max       0.611000            68.000000            289.000000     1.003690

                pH    sulphates      alcohol      quality           Id
count  1143.000000  1143.000000  1143.000000  1143.000000  1143.000000
mean      3.311015     0.657708    10.442111     5.657043   804.969379
std       0.156664     0.170399     1.082196     0.805824   463.997116
min       2.740000     0.330000     8.400000     3.000000     0.000000
25%       3.205000     0.550000     9.500000     5.000000   411.000000
50%       3.310000     0.620000    10.200000     6.000000   794.000000
75%       3.400000     0.730000    11.100000     6.000000  1209.500000
max       4.010000     2.000000    14.900000     8.000000  1597.000000
```

```
print(df.groupby('quality').size())
```

```
quality
3      6
4     33
5    483
6    462
7    143
8     16
dtype: int64
```

```
df.dtypes
```

```
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
```

```
df.corr()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quali
fixed acidity	1.000000	-0.250728	0.673157	0.171831	0.107889	-0.164831	-0.110628	0.681501	-0.685163	0.174592	-0.075055	0.121
volatile acidity	-0.250728	1.000000	-0.544187	-0.005751	0.056336	-0.001962	0.077748	0.016512	0.221492	-0.276079	-0.203909	-0.407
citric acid	0.673157	-0.544187	1.000000	0.175815	0.245312	-0.057589	0.036871	0.375243	-0.546339	0.331232	0.106250	0.240
residual sugar	0.171831	-0.005751	0.175815	1.000000	0.070863	0.165339	0.190790	0.380147	-0.116959	0.017475	0.058421	0.022
chlorides	0.107889	0.056336	0.245312	0.070863	1.000000	0.015280	0.048163	0.208901	-0.277759	0.374784	-0.229917	-0.124
free sulfur dioxide	-0.164831	-0.001962	-0.057589	0.165339	0.015280	1.000000	0.661093	-0.054150	0.072804	0.034445	-0.047095	-0.063
total sulfur dioxide	-0.110628	0.077748	0.036871	0.190790	0.048163	0.661093	1.000000	0.050175	-0.059126	0.026894	-0.188165	-0.183
density	0.681501	0.016512	0.375243	0.380147	0.208901	-0.054150	0.050175	1.000000	-0.352775	0.143139	-0.494727	-0.175

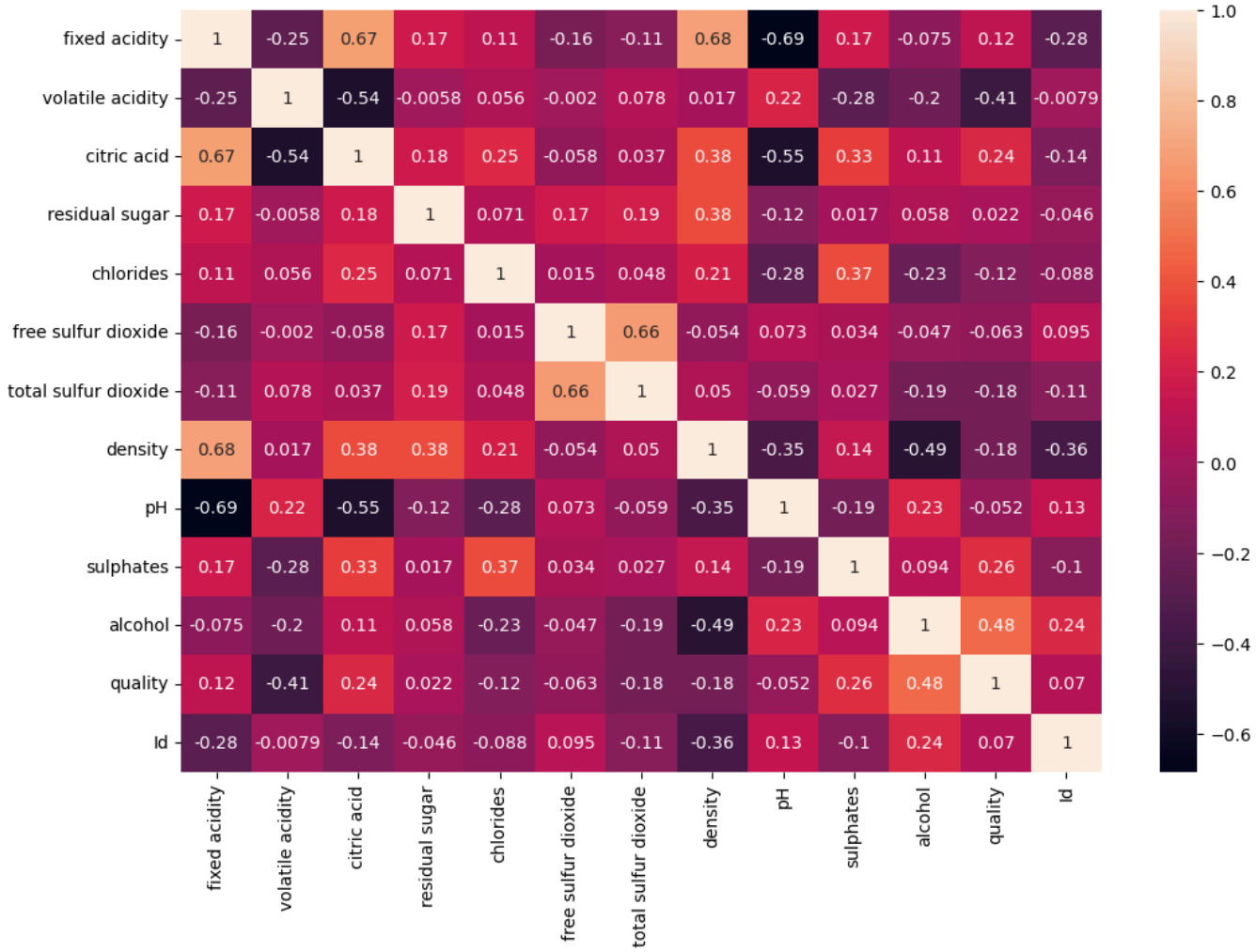
```
import warnings
```

```
warnings.filterwarnings('ignore')
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(),annot=True)
plt.show()
```

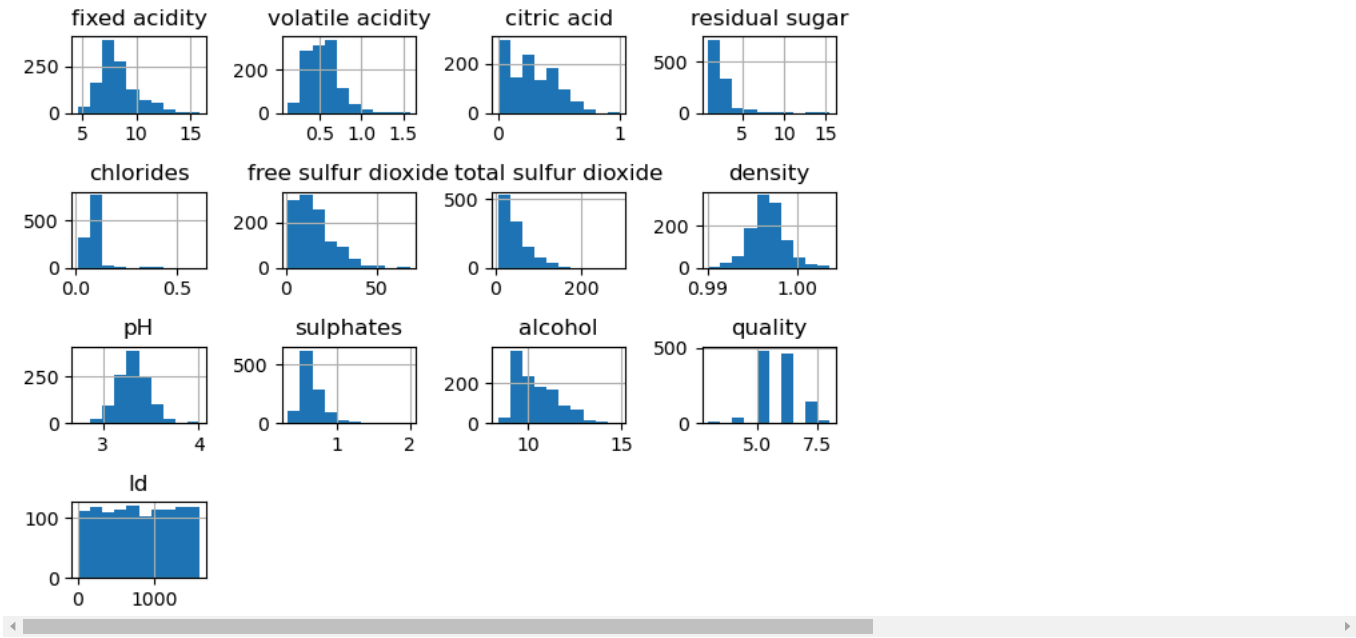



```
df.skew()
```

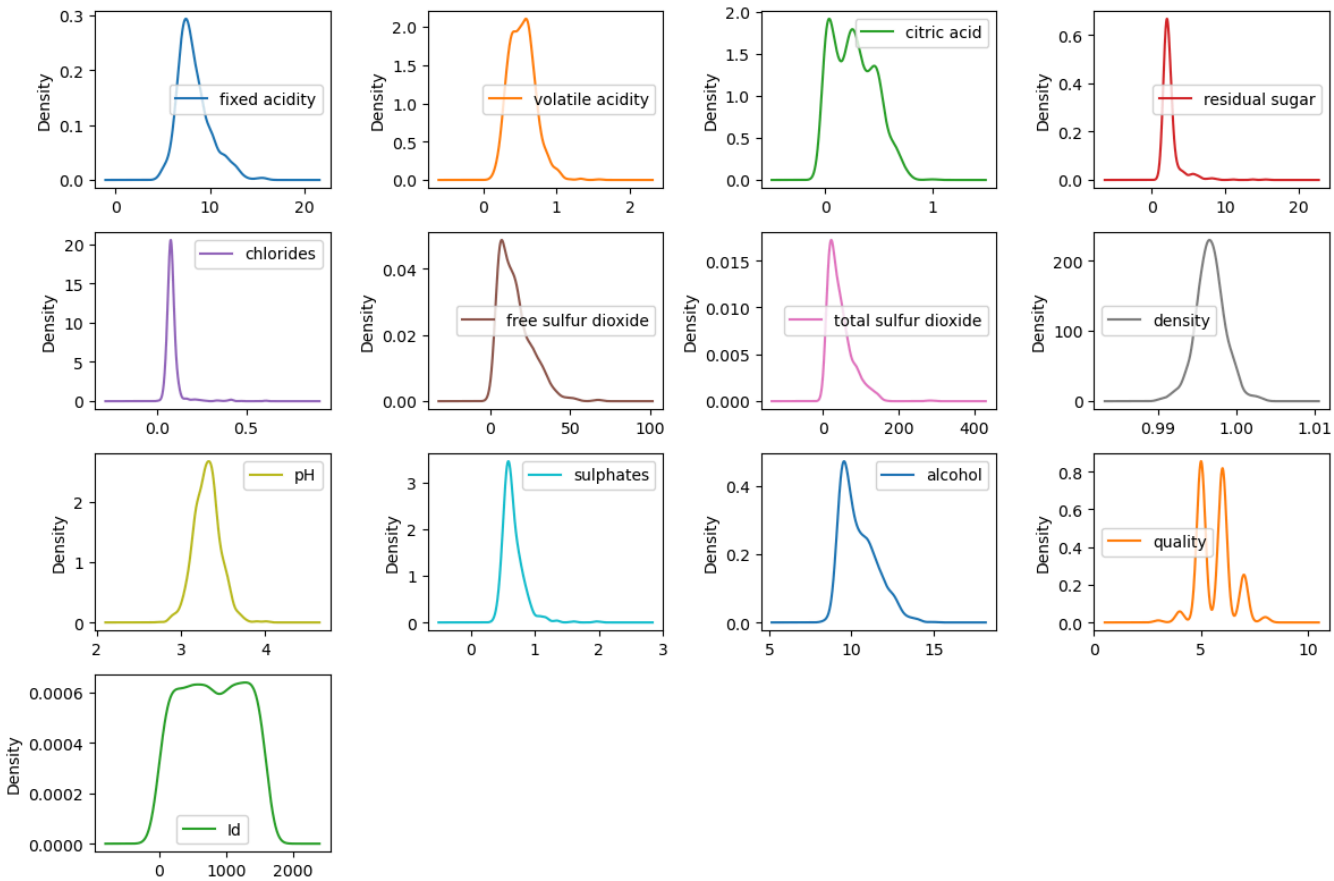
```
fixed acidity           1.044930
volatile acidity        0.681547
citric acid             0.371561
residual sugar          4.361096
chlorides               6.026360
free sulfur dioxide     1.231261
total sulfur dioxide    1.665766
density                 0.102395
pH                      0.221138
sulphates               2.497266
alcohol                 0.863313
quality                 0.286792
Id                     -0.010419
dtype: float64
```

```
plt.figure(figsize=(12,8))
df.hist()
plt.tight_layout()
plt.show()
```

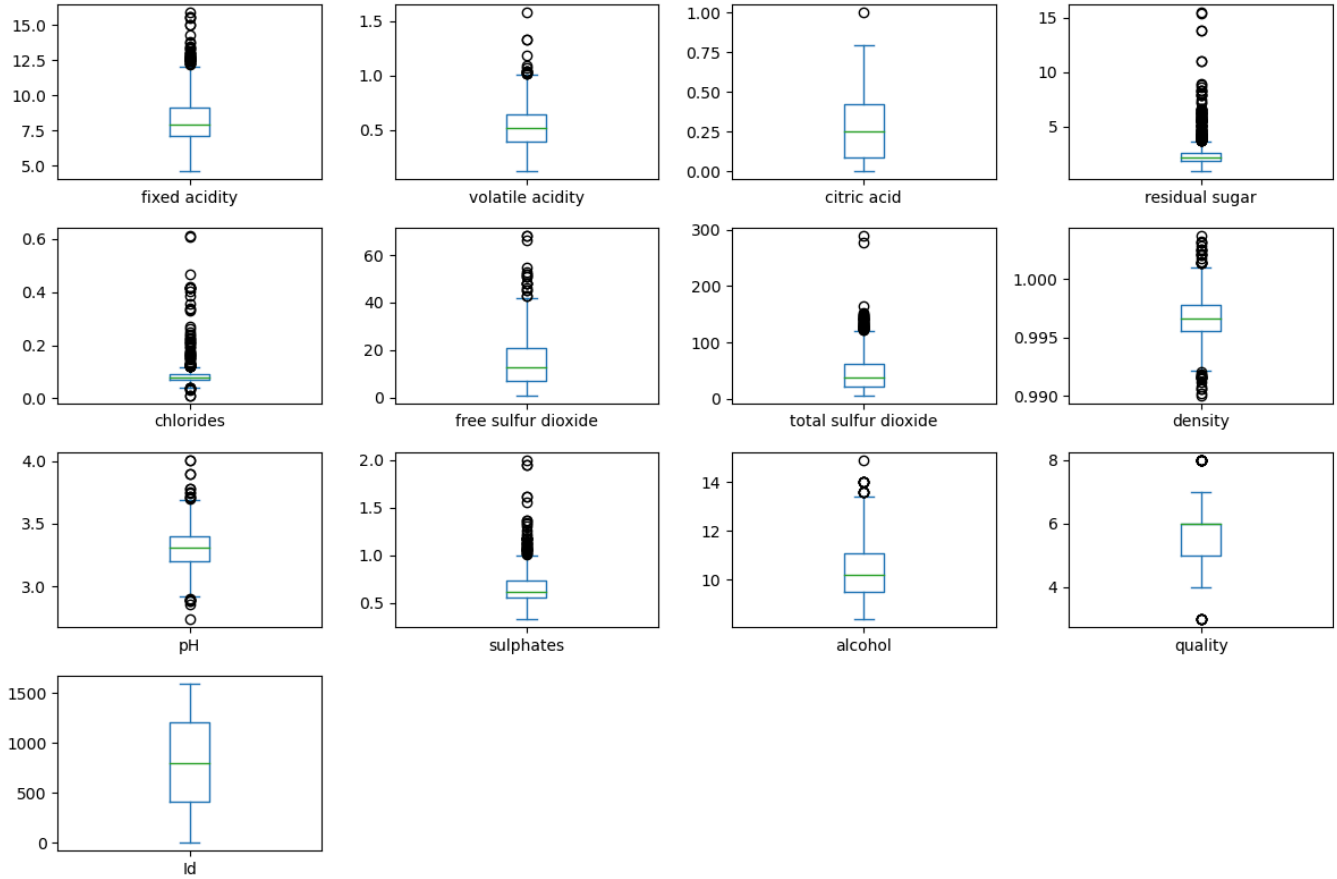
```
<Figure size 1200x800 with 0 Axes>
```




```
df.plot(kind='density', subplots=True, layout=(4,4), sharex=False,figsize=(12,8))
plt.tight_layout()
plt.show()
```




```
df.plot(kind='box', subplots=True, layout=(4,4), sharex=False, sharey=False,figsize=(12,8))
plt.tight_layout()
plt.show()
```




```
from pandas.plotting import scatter_matrix
```

```
df.columns
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality', 'Id'],
      dtype='object')
```

```
df.shape
```

```
(1143, 13)
```

```
import matplotlib.pyplot as plt
```

```
df.columns
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality', 'Id'],
      dtype='object')
```

```
classes = df['quality'].values
```

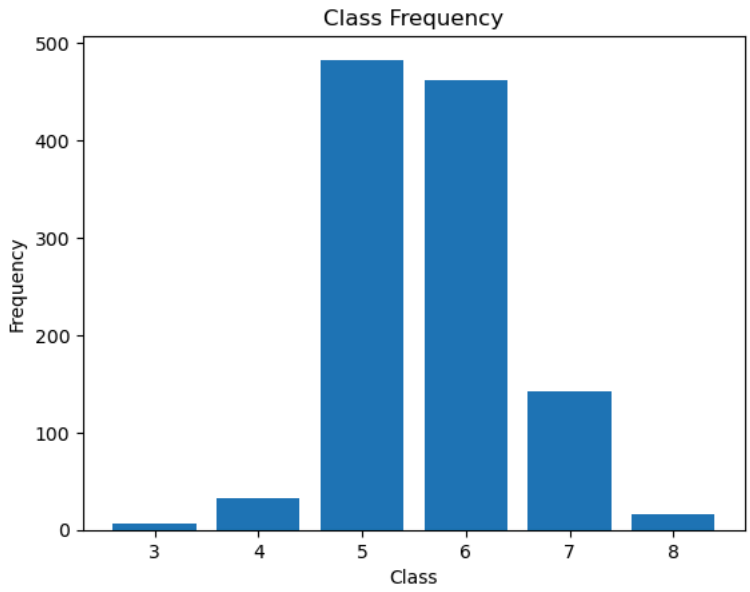
```
classes
```

```
array([5, 5, 5, ..., 5, 6, 5])
```

```
unique,counts = np.unique(classes,return_counts=True)
```

```
plt.bar(unique,counts)
plt.title('Class Frequency')
plt.xlabel('Class')
```

```
plt.ylabel('Frequency')
plt.show()
```




```
df
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	Id
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	0
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5	1
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5	2
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6	3
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	4
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1138	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6	1592
1139	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	9.5	6	1593
1140	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5	1594
1141	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6	1595
1142	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5	1597

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

```
x,y
```

```
(      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
...             ...               ...          ...             ...        ...
1138            6.3             0.510         0.13             2.3      0.076
1139            6.8             0.620         0.08             1.9      0.068
1140            6.2             0.600         0.08             2.0      0.090
1141            5.9             0.550         0.10             2.2      0.062
1142            5.9             0.645         0.12             2.0      0.075

       free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                     11.0                  34.0  0.99780  3.51       0.56
1                     25.0                  67.0  0.99680  3.20       0.68
2                     15.0                  54.0  0.99700  3.26       0.65
3                     17.0                  60.0  0.99800  3.16       0.58
4                     11.0                  34.0  0.99780  3.51       0.56
...                    ...                   ...      ...   ...        ...
1138                  29.0                  40.0  0.99574  3.42       0.75
1139                  28.0                  38.0  0.99651  3.42       0.82
1140                  32.0                  44.0  0.99490  3.45       0.58
1141                  39.0                  51.0  0.99512  3.52       0.76
1142                  32.0                  44.0  0.99547  3.57       0.71
```

```
      alcohol    Id
0         9.4     0
1         9.8     1
2         9.8     2
3         9.8     3
4         9.4     4
...       ...   ...
1138     11.0  1592
1139      9.5  1593
1140     10.5  1594
1141     11.2  1595
1142     10.2  1597

[1143 rows x 12 columns],
0       5
1       5
2       5
3       6
4       5
       ..
1138    6
1139    6
1140    5
1141    6
1142    5
Name: quality, Length: 1143, dtype: int64)
```

```
from sklearn.model_selection import train_test_split
```

```
from imblearn.over_sampling import SMOTE
```

```
from sklearn.model_selection import train_test_split
```

```
 # Split-out validation dataset
 validation_size = 0.20
```

```
Xtrain,Xtest,Ytrain,Ytest=train_test_split(x,y,test_size=validation_size)
```

```
smote = SMOTE()
```

```
Xtrain.shape,Ytrain.shape
```

```
((914, 12), (914,))
```

```
#from imblearn.over_sampling import SMOTE

# Adjust the number of neighbors
#smote = SMOTE(sampling_strategy='auto', random_state=42, k_neighbors=2)

#XtrainResampled, YtrainResampled = smote.fit_resample(Xtrain, Ytrain)
```

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=42)
XtrainResampled, YtrainResampled = ros.fit_resample(Xtrain, Ytrain)
```

```
#XtrainResampled,YtrainResampled = smote.fit_resample(Xtrain,Ytrain)
```

```
Xtrain.shape,Ytrain.shape
```

```
((914, 12), (914,))
```

```
XtrainResampled.shape,YtrainResampled.shape
```

```
((2292, 12), (2292,))
```

```
Ytrain.unique
```

```
<bound method Series.unique of 656     4
187     5
1102    5
1125    6
537     5
       ..
966     6
688     6
```

```
148     6
779     6
568     5
Name: quality, Length: 914, dtype: int64>
```

```
YtrainResampled.unique
```

```
<bound method Series.unique of 0       4
1       5
2       5
3       6
4       5
       ..
2287    8
2288    8
2289    8
2290    8
2291    8
Name: quality, Length: 2292, dtype: int64>
```

```
Ytrain
```

```
656     4
187     5
1102    5
1125    6
537     5
       ..
966     6
688     6
148     6
779     6
568     5
Name: quality, Length: 914, dtype: int64
```

```
from collections import Counter
```

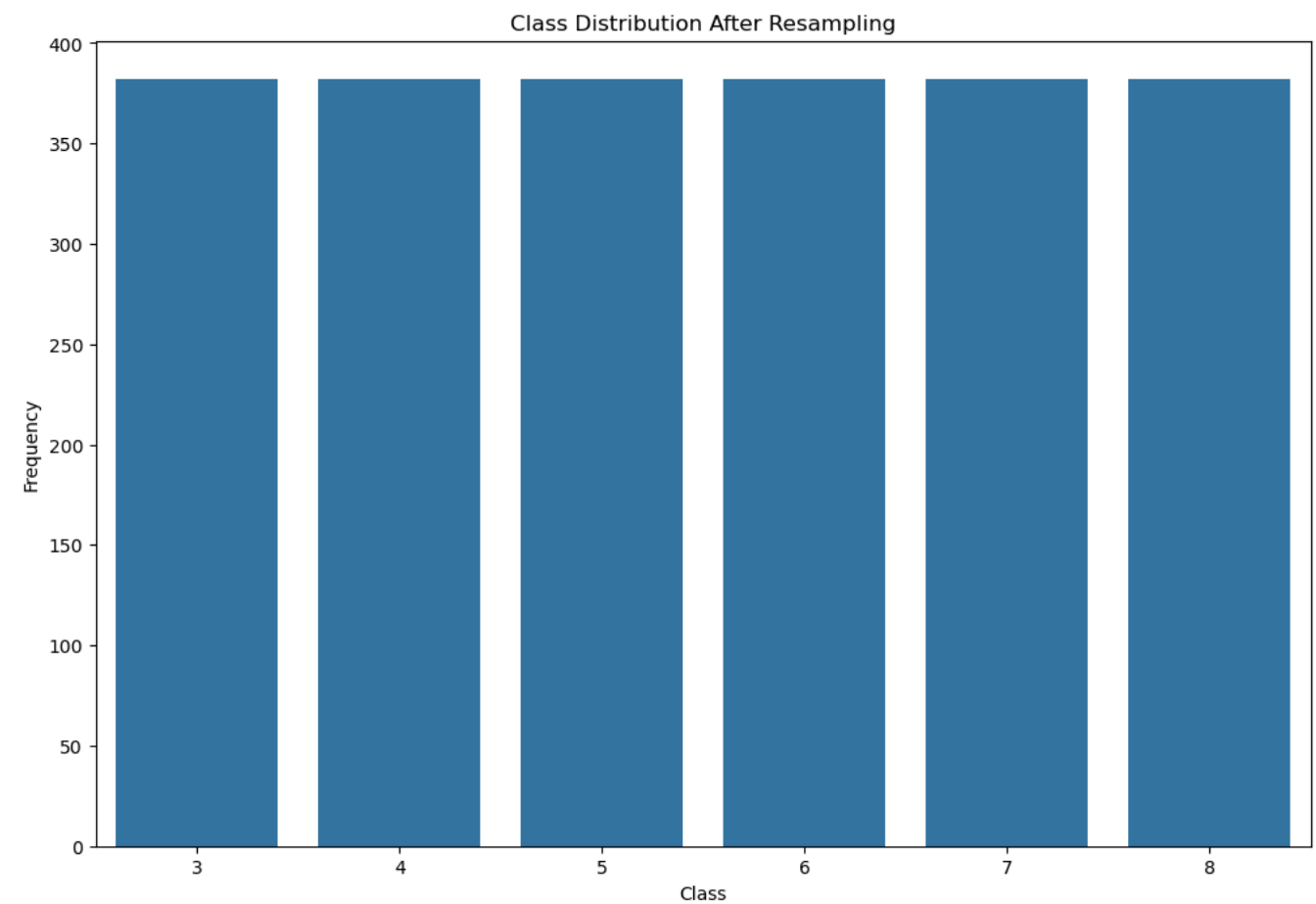
```
Counter(YtrainResampled)
```

```
Counter({4: 382, 5: 382, 6: 382, 7: 382, 8: 382, 3: 382})
```

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(12,8))
sns.countplot(x=YtrainResampled)
plt.title('Class Distribution After Resampling')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```



Classification

Algorithm Spot-Checking

Import libraries

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

Build Models

Logistic Regression (LR).

Linear Discriminant Analysis (LDA).

k-Nearest Neighbors (KNN).

Classification and Regression Trees (CART).

Gaussian Naive Bayes (NB).

Support Vector Machines (SVM).

```
# Spot-Check Algorithms
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
```

```
results = []
names = []
from sklearn.model_selection import StratifiedKFold

for name, model in models:
     kfold = StratifiedKFold(n_splits=10)
     cv_results = cross_val_score(model, XtrainResampled, YtrainResampled, cv=kfold, scoring='accuracy')
     results.append(cv_results)
     names.append(name)
     msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
     print(msg)
```

```
LR: 0.308895 (0.044565)
LDA: 0.586835 (0.025414)
KNN: 0.767477 (0.024571)
CART: 0.872611 (0.022944)
NB: 0.472525 (0.037043)
SVM: 0.373015 (0.028873)
```

Evaluate Algorithms: Standardize Data

```
from sklearn.pipeline import Pipeline
```

```
from sklearn.preprocessing import StandardScaler
```

```
# Standardize the dataset
pipelines = []
pipelines.append(('ScaledLR', Pipeline([('Scaler', StandardScaler()),('LR',LogisticRegression())])))
pipelines.append(('ScaledLDA', Pipeline([('Scaler', StandardScaler()),('LDA',LinearDiscriminantAnalysis())])))
pipelines.append(('ScaledKNN', Pipeline([('Scaler', StandardScaler()),('KNN',KNeighborsClassifier())])))
pipelines.append(('ScaledCART', Pipeline([('Scaler', StandardScaler()),('CART',DecisionTreeClassifier())])))
pipelines.append(('ScaledNB', Pipeline([('Scaler', StandardScaler()),('NB',GaussianNB())])))
pipelines.append(('ScaledSVM', Pipeline([('Scaler', StandardScaler()),('SVM', SVC())])))
results = []
names = []
for name, model in pipelines:
    kfold = KFold(n_splits=10 )
    cv_results = cross_val_score(model,XtrainResampled, YtrainResampled,  cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
ScaledLR: 0.511016 (0.211340)
ScaledLDA: 0.459060 (0.165190)
ScaledKNN: 0.778137 (0.212935)
ScaledCART: 0.866649 (0.164301)
ScaledNB: 0.428961 (0.178168)
ScaledSVM: 0.701722 (0.201835)
```

Evaluate Algorithms: Rescale Data

```
from sklearn.preprocessing import MinMaxScaler
```

```
# Rescale the dataset
pipelines = []
pipelines.append(('ScaledLR', Pipeline([('Scaler', MinMaxScaler(feature_range=(0, 1))),('LR',LogisticRegression())])))
pipelines.append(('ScaledLDA', Pipeline([('Scaler', MinMaxScaler(feature_range=(0, 1))),('LDA',LinearDiscriminantAnalysis())
pipelines.append(('ScaledKNN', Pipeline([('Scaler', MinMaxScaler(feature_range=(0, 1))),('KNN',KNeighborsClassifier())])))
pipelines.append(('ScaledCART', Pipeline([('Scaler', MinMaxScaler(feature_range=(0, 1))),('CART',DecisionTreeClassifier())])
pipelines.append(('ScaledNB', Pipeline([('Scaler', MinMaxScaler(feature_range=(0, 1))),('NB',GaussianNB())])))
```

```
pipelines.append(('ScaledSVM', Pipeline[('Scaler', MinMaxScaler(feature_range=(0, 1))),('SVM', SVC())])))
results = []
names = []
for name, model in pipelines:
    kfold = KFold(n_splits=10 )
    cv_results = cross_val_score(model,XtrainResampled, YtrainResampled,  cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
ScaledLR: 0.367830 (0.223560)
ScaledLDA: 0.459060 (0.165190)
ScaledKNN: 0.775079 (0.210048)
ScaledCART: 0.862303 (0.169811)
ScaledNB: 0.428961 (0.178168)
ScaledSVM: 0.615318 (0.216493)
```

Evaluate Algorithms: Normalize Data

```
from sklearn.preprocessing import Normalizer
```

```
# Normalize the dataset
pipelines = []
pipelines.append(('ScaledLR', Pipeline[('Scaler',Normalizer()),('LR',LogisticRegression())])))
pipelines.append(('ScaledLDA', Pipeline[('Scaler', Normalizer()),('LDA',LinearDiscriminantAnalysis())])))
pipelines.append(('ScaledKNN', Pipeline[('Scaler', Normalizer()),('KNN',KNeighborsClassifier())])))
pipelines.append(('ScaledCART', Pipeline[('Scaler', Normalizer()),('CART',DecisionTreeClassifier())])))
pipelines.append(('ScaledNB', Pipeline[('Scaler', Normalizer()),('NB',GaussianNB())])))
pipelines.append(('ScaledSVM', Pipeline[('Scaler', Normalizer()),('SVM', SVC())])))
results = []
names = []
for name, model in pipelines:
    kfold = KFold(n_splits=10 )
    cv_results = cross_val_score(model,XtrainResampled, YtrainResampled,  cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
ScaledLR: 0.017448 (0.015491)
ScaledLDA: 0.273585 (0.129469)
ScaledKNN: 0.744583 (0.236910)
ScaledCART: 0.848361 (0.186002)
ScaledNB: 0.298642 (0.312264)
ScaledSVM: 0.020507 (0.020017)
```

Improve Performance with tuning the CART

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
from imblearn.over_sampling import RandomOverSampler
from collections import Counter

# Load your dataset (replace this with your dataset)
# Assume 'df' contains the data and 'quality' is the target variable
# Example: df = pd.read_csv("your_dataset.csv")
x = df.drop('quality', axis=1)
y = df['quality']

# Split the data into training and validation sets (40% validation set)
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.4, stratify=y, random_state=42)

# Scale the features
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)

# Address class imbalance with RandomOverSampler
ros = RandomOverSampler(random_state=42)
x_train_resampled, y_train_resampled = ros.fit_resample(x_train, y_train)
print("Resampled class distribution:", Counter(y_train_resampled))
```

```
# Initialize the DecisionTreeClassifier
model = DecisionTreeClassifier(random_state=42)

# Define the parameter grid for GridSearchCV
param_grid = {
    'max_depth': [None, 10, 20, 30],  # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],    # Minimum samples required to be a leaf node
    'criterion': ['gini', 'entropy']  # Splitting criterion
}

# Set up cross-validation
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Perform GridSearchCV for parameter tuning
grid_search = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=kfold,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

# Fit the model to the resampled training data
grid_search.fit(x_train_resampled, y_train_resampled)

# Print the best parameters and the corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)

# Use the best model from GridSearchCV
best_model = grid_search.best_estimator_

# Train the model on the full resampled training set
best_model.fit(x_train_resampled, y_train_resampled)

# Validate the model on the validation set
y_val_pred = best_model.predict(x_val)

# Evaluate performance on the validation set
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print("\nClassification Report (Validation):\n", classification_report(y_val, y_val_pred))
```

```
Resampled class distribution: Counter({6: 289, 8: 289, 5: 289, 7: 289, 4: 289, 3: 289})
Fitting 10 folds for each of 72 candidates, totalling 720 fits
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
Best Parameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best Cross-Validation Accuracy: 0.8506411534117333
Validation Accuracy: 0.5982532751091703

Classification Report (Validation):
               precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        13
           5       0.66      0.74      0.70       194
           6       0.59      0.53      0.56       185
           7       0.52      0.54      0.53        57
           8       0.14      0.14      0.14         7

    accuracy                           0.60       458
   macro avg       0.32      0.33      0.32       458
weighted avg       0.59      0.60      0.59       458
```

Select the best hyperparameters and improve Performance

```
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Load a sample dataset (Iris dataset as an example)
scaler = StandardScaler().fit(XtrainResampled)
XtrainResampled = scaler.transform(XtrainResampled)
x=df.drop('quality',axis=1)
y=df['quality']

# Split the data into training and validation sets (validation size = 20%)
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.4)

from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=42)
x_trainResampled, y_trainResampled = ros.fit_resample(x_train, y_train)

# Initialize the SVM classifier with specified parameters
cart_model = DecisionTreeClassifier(
    criterion='entropy',
    max_depth=None,
    min_samples_leaf=1,
    min_samples_split=2,
    random_state=42  # Ensure reproducibility
)

# Set up 10-fold cross-validation
kfold = StratifiedKFold(n_splits=20)

# Perform cross-validation on the training data
cv_scores = cross_val_score(cart_model, x_trainResampled, y_trainResampled, cv=kfold, scoring='accuracy')

# Train the SVM model on the full training set
cart_model.fit(x_trainResampled, y_trainResampled)

# Validate the model on the validation set
y_val_pred = cart_model.predict(x_val)

# Evaluate performance
print("Cross-Validation Accuracy Scores:", cv_scores)
print("Mean CV Accuracy:", cv_scores.mean())
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print("\nClassification Report (Validation):\n", classification_report(y_val, y_val_pred))
```

```
Cross-Validation Accuracy Scores: [0.875      0.85227273 0.85227273 0.82954545 0.90909091 0.82954545
 0.89655172 0.82758621 0.87356322 0.87356322 0.85057471 0.86206897
 0.88505747 0.8045977  0.85057471 0.87356322 0.86206897 0.82758621
 0.90804598 0.93103448]
Mean CV Accuracy: 0.8637082027168234
Validation Accuracy: 0.5829694323144105

Classification Report (Validation):
               precision    recall  f1-score   support

           3       0.00      0.00      0.00         3
           4       0.00      0.00      0.00         9
           5       0.68      0.66      0.67       192
           6       0.58      0.60      0.59       189
           7       0.50      0.46      0.48        59
           8       0.00      0.00      0.00         6

    accuracy                           0.58       458
   macro avg       0.29      0.29      0.29       458
weighted avg       0.59      0.58      0.59       458
```

Improve Performance with Ensembles

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier


# ensembles
ensembles = []
```

```
ensembles.append(('AB', AdaBoostClassifier()))
ensembles.append(('GBM', GradientBoostingClassifier()))
ensembles.append(('RF', RandomForestClassifier()))
ensembles.append(('DTC',DecisionTreeClassifier()))
ensembles.append(('ET', ExtraTreesClassifier()))
results = []
names = []
for name, model in ensembles:
     kfold = KFold(n_splits=10)
     cv_results = cross_val_score(model, XtrainResampled, YtrainResampled, cv=10, scoring='accuracy')
     results.append(cv_results)
     names.append(name)
     msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
     print(msg)
```

```
AB: 0.318063 (0.008162)
GBM: 0.878703 (0.014606)
RF: 0.890063 (0.013114)
DTC: 0.863440 (0.024066)
ET: 0.903577 (0.016951)
```

```
# Standardize the dataset
pipelines = []
pipelines.append(('ScaledAB', Pipeline([('Scaler', StandardScaler()),('AB', AdaBoostClassifier())])))
pipelines.append(('ScaledGBM', Pipeline([('Scaler', StandardScaler()),('GBM', GradientBoostingClassifier())])))
pipelines.append(('ScaledRF', Pipeline([('Scaler', StandardScaler()),('RF', RandomForestClassifier())])))
pipelines.append(('ScaledDTC', Pipeline([('Scaler', StandardScaler()),('DTC',DecisionTreeClassifier())])))
pipelines.append(('ScaledETC', Pipeline([('Scaler', StandardScaler()),('ETC', ExtraTreesClassifier())])))
results = []
names = []
for name, model in pipelines:
    kfold = KFold(n_splits=10 )
    cv_results = cross_val_score(model,XtrainResampled, YtrainResampled,  cv=10, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
ScaledAB: 0.318063 (0.008162)
ScaledGBM: 0.878707 (0.017104)
ScaledRF: 0.894418 (0.013757)
ScaledDTC: 0.867805 (0.021635)
ScaledETC: 0.903586 (0.016781)
```

```
from sklearn.model_selection import GridSearchCV
```

```
scoring='accuracy'
```

```
# Tune scaled RF
scaler = StandardScaler().fit(XtrainResampled)
rescaledX = scaler.transform(XtrainResampled)
param_grid = dict(n_estimators=np.array([50,100,150,200,250,300,350,400]))
model = ExtraTreesClassifier()
kfold = KFold(n_splits=num_folds)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(rescaledX, YtrainResampled)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: -0.126797 using {'n_estimators': 100}
-0.131590 (0.161312) with: {'n_estimators': 50}
-0.126797 (0.155369) with: {'n_estimators': 100}
-0.129412 (0.159751) with: {'n_estimators': 150}
-0.134205 (0.164372) with: {'n_estimators': 200}
-0.132026 (0.161735) with: {'n_estimators': 250}
-0.127233 (0.155980) with: {'n_estimators': 300}
-0.128976 (0.158826) with: {'n_estimators': 350}
-0.129412 (0.158510) with: {'n_estimators': 400}
```

```
# Tune scaled ET
scaler = StandardScaler().fit(XtrainResampled)
rescaledX = scaler.transform(XtrainResampled)
param_grid = dict(n_estimators=np.array([50,100,150,200,250,300,350,400]))
model = RandomForestClassifier()
kfold = KFold(n_splits=num_folds)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
```

```
grid_result = grid.fit(rescaledX, YtrainResampled)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: -0.160784 using {'n_estimators': 200}
-0.164270 (0.201332) with: {'n_estimators': 50}
-0.166885 (0.204448) with: {'n_estimators': 100}
-0.168627 (0.206940) with: {'n_estimators': 150}
-0.160784 (0.196979) with: {'n_estimators': 200}
-0.163834 (0.200774) with: {'n_estimators': 250}
-0.170370 (0.209261) with: {'n_estimators': 300}
-0.167320 (0.204943) with: {'n_estimators': 350}
-0.165142 (0.202351) with: {'n_estimators': 400}
```

Selected Model with Parameter Tuning

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import accuracy_score, classification_report
from imblearn.over_sampling import RandomOverSampler
from collections import Counter

# Load your dataset (replace this with your dataset)
# Assume 'df' contains the data and 'quality' is the target variable
# Example: df = pd.read_csv("your_dataset.csv")
x = df.drop('quality', axis=1)
y = df['quality']

# Split the data into training and validation sets (40% validation set)
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.4, stratify=y, random_state=42)

# Scale the features
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)

# Address class imbalance with RandomOverSampler
ros = RandomOverSampler(random_state=42)
x_train_resampled, y_train_resampled = ros.fit_resample(x_train, y_train)
print("Resampled class distribution:", Counter(y_train_resampled))

# Initialize the ExtraTreesClassifier
model = ExtraTreesClassifier(random_state=42)

# Define the parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 500],  # Number of trees
    'max_depth': [None, 10, 20, 30],  # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],    # Minimum samples required to be a leaf node
}

# Set up cross-validation
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Perform GridSearchCV for parameter tuning
grid_search = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=kfold,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

# Fit the model to the resampled training data
grid_search.fit(x_train_resampled, y_train_resampled)

# Print the best parameters and the corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)

# Use the best model from GridSearchCV
```

```
best_model = grid_search.best_estimator_

# Train the model on the full resampled training set
best_model.fit(x_train_resampled, y_train_resampled)

# Validate the model on the validation set
y_val_pred = best_model.predict(x_val)

# Evaluate performance on the validation set
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print("\nClassification Report (Validation):\n", classification_report(y_val, y_val_pred))
```

```
Resampled class distribution: Counter({6: 289, 8: 289, 5: 289, 7: 289, 4: 289, 3: 289})
Fitting 10 folds for each of 108 candidates, totalling 1080 fits
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 200}
Best Cross-Validation Accuracy: 0.891542090226563
Validation Accuracy: 0.6877729257641921

Classification Report (Validation):
               precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        13
           5       0.71      0.82      0.76       194
           6       0.65      0.69      0.67       185
           7       0.72      0.49      0.58        57
           8       1.00      0.14      0.25         7

    accuracy                           0.69       458
   macro avg       0.51      0.36      0.38       458
weighted avg       0.67      0.69      0.67       458
```

Finalize the model for Classification

```
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Load a sample dataset (Iris dataset as an example)
scaler = StandardScaler().fit(XtrainResampled)
XtrainResampled = scaler.transform(XtrainResampled)
x=df.drop('quality',axis=1)
y=df['quality']

# Split the data into training and validation sets (validation size = 20%)
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.4)

from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=42)
x_trainResampled, y_trainResampled = ros.fit_resample(x_train, y_train)

# Initialize the SVM classifier with specified parameters
model1 = ExtraTreesClassifier(max_depth=None,
    min_samples_leaf=1,
    min_samples_split=5,
    n_estimators=200,
    random_state=42)

# Set up 10-fold cross-validation
kfold = StratifiedKFold(n_splits=20)

# Perform cross-validation on the training data
cv_scores = cross_val_score(model1, x_trainResampled, y_trainResampled, cv=kfold, scoring='accuracy')

# Train the SVM model on the full training set
model1.fit(x_trainResampled, y_trainResampled)

# Validate the model on the validation set
y_val_pred = model1.predict(x_val)

# Evaluate performance
print("Cross-Validation Accuracy Scores:", cv_scores)
print("Mean CV Accuracy:", cv_scores.mean())
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print("\nClassification Report (Validation):\n", classification_report(y_val, y_val_pred))
```

```
Cross-Validation Accuracy Scores: [0.95402299 0.88505747 0.89655172 0.88505747 0.90804598 0.94252874
 0.87356322 0.89655172 0.91860465 0.90697674 0.90697674 0.87209302
```

```
 0.91860465 0.90697674 0.91860465 0.90697674 0.90697674 0.84883721
 0.94186047 0.97674419]
Mean CV Accuracy: 0.9085805934242182
Validation Accuracy: 0.6703056768558951

Classification Report (Validation):
               precision    recall  f1-score   support

           4       0.00      0.00      0.00        12
           5       0.69      0.82      0.75       195
           6       0.66      0.64      0.65       194
           7       0.58      0.45      0.51        49
           8       0.50      0.12      0.20         8

    accuracy                           0.67       458
   macro avg       0.49      0.41      0.42       458
weighted avg       0.65      0.67      0.65       458
```

Regression

Evaluate Algorithms: Spot-check

Load libraries

```
# Load libraries
import numpy
from numpy import arange
from matplotlib import pyplot
from pandas import read_csv
from pandas import set_option
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error
```

Validation Dataset

```
df = df.drop(columns=['Id'], errors='ignore')
```

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

```
validation_size = 0.20
X_train, X_validation, Y_train, Y_validation = train_test_split(x, y,test_size=validation_size)
```

```
scoring = 'neg_mean_squared_error'
```

```
# Spot-Check Algorithms
models = []
models.append(('LR', LinearRegression()))
models.append(('LASSO', Lasso()))
models.append(('EN', ElasticNet()))
models.append(('KNN', KNeighborsRegressor()))
models.append(('CART', DecisionTreeRegressor()))
models.append(('SVR', SVR()))
```

```
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=num_folds)
    cv_results = cross_val_score(model,X_train, Y_train ,cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    print(name, cv_results.mean())
```

```
LR -0.42994960767704093
LASSO -0.6415874978061692
EN -0.6409341745854689
KNN -0.6028201525250705
CART -0.7319582057286975
SVR -0.5795850097823019
```

Evaluate Algorithms: Standardization¶

```
import numpy
from numpy import arange
from matplotlib import pyplot
from pandas import read_csv
from pandas import set_option
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error
```

```
pipelines = []
pipelines.append(('ScaledLR', Pipeline([('Scaler', StandardScaler()),('LR',LinearRegression())])))
pipelines.append(('ScaledLASSO', Pipeline([('Scaler', StandardScaler()),('LASSO',Lasso())])))
pipelines.append(('ScaledEN', Pipeline([('Scaler', StandardScaler()),('EN',ElasticNet())])))
pipelines.append(('ScaledKNN', Pipeline([('Scaler', StandardScaler()),('KNN',KNeighborsRegressor())])))
pipelines.append(('ScaledCART', Pipeline([('Scaler', StandardScaler()),('CART',DecisionTreeRegressor())])))
pipelines.append(('ScaledSVR', Pipeline([('Scaler', StandardScaler()),('SVR', SVR())])))
results = []
names = []
for name, model in pipelines:
    kfold = KFold(n_splits=num_folds)
    cv_results = cross_val_score(model,X_train, Y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
ScaledLR: -0.429950 (0.058777)
ScaledLASSO: -0.663422 (0.051715)
ScaledEN: -0.663422 (0.051715)
ScaledKNN: -0.506699 (0.053998)
ScaledCART: -0.725347 (0.050702)
ScaledSVR: -0.404119 (0.057109)
```

Improve Results With Tuning

```
import numpy as np
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
```

```
# Example: Replace 'X' and 'y' with your dataset
# X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize the dataset
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)

# Define hyperparameters for tuning
param_grid = {
    'C': [0.1, 1, 10, 100],           # Regularization parameter
    'epsilon': [0.1, 0.2, 0.5, 1.0],  # Epsilon in the epsilon-SVR model
    'kernel': ['linear', 'rbf'],      # Kernel type
    'gamma': ['scale', 'auto']        # Kernel coefficient
}

# Create the SVR model
model = SVR()

# Define cross-validation settings
num_folds = 5
kfold = KFold(n_splits=num_folds, random_state=42, shuffle=True)

# Perform GridSearchCV
scoring = 'neg_mean_squared_error'  # Metric for regression
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold, n_jobs=-1)
grid_result = grid.fit(rescaledX, Y_train)

# Output the best parameters and performance
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required fo
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required fo
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required fo
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required fo
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required fo
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required fo
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required fo
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required fo
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
Best: -0.409792 using {'C': 1, 'epsilon': 0.1, 'gamma': 'scale', 'kernel': 'rbf'}
-0.431212 (0.041127) with: {'C': 0.1, 'epsilon': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
-0.432148 (0.057956) with: {'C': 0.1, 'epsilon': 0.1, 'gamma': 'scale', 'kernel': 'rbf'}
-0.431212 (0.041127) with: {'C': 0.1, 'epsilon': 0.1, 'gamma': 'auto', 'kernel': 'linear'}
-0.432205 (0.058107) with: {'C': 0.1, 'epsilon': 0.1, 'gamma': 'auto', 'kernel': 'rbf'}
-0.429433 (0.042131) with: {'C': 0.1, 'epsilon': 0.2, 'gamma': 'scale', 'kernel': 'linear'}
-0.429025 (0.058165) with: {'C': 0.1, 'epsilon': 0.2, 'gamma': 'scale', 'kernel': 'rbf'}
-0.429433 (0.042131) with: {'C': 0.1, 'epsilon': 0.2, 'gamma': 'auto', 'kernel': 'linear'}
-0.429025 (0.058202) with: {'C': 0.1, 'epsilon': 0.2, 'gamma': 'auto', 'kernel': 'rbf'}
-0.432746 (0.052777) with: {'C': 0.1, 'epsilon': 0.5, 'gamma': 'scale', 'kernel': 'linear'}
-0.475800 (0.066372) with: {'C': 0.1, 'epsilon': 0.5, 'gamma': 'scale', 'kernel': 'rbf'}
-0.432746 (0.052777) with: {'C': 0.1, 'epsilon': 0.5, 'gamma': 'auto', 'kernel': 'linear'}
-0.475776 (0.066481) with: {'C': 0.1, 'epsilon': 0.5, 'gamma': 'auto', 'kernel': 'rbf'}
-0.442395 (0.054994) with: {'C': 0.1, 'epsilon': 1.0, 'gamma': 'scale', 'kernel': 'linear'}
-0.592369 (0.101792) with: {'C': 0.1, 'epsilon': 1.0, 'gamma': 'scale', 'kernel': 'rbf'}
-0.442395 (0.054994) with: {'C': 0.1, 'epsilon': 1.0, 'gamma': 'auto', 'kernel': 'linear'}
-0.592324 (0.102414) with: {'C': 0.1, 'epsilon': 1.0, 'gamma': 'auto', 'kernel': 'rbf'}
-0.431865 (0.040620) with: {'C': 1, 'epsilon': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
-0.409792 (0.052889) with: {'C': 1, 'epsilon': 0.1, 'gamma': 'scale', 'kernel': 'rbf'}
-0.431865 (0.040620) with: {'C': 1, 'epsilon': 0.1, 'gamma': 'auto', 'kernel': 'linear'}
-0.409879 (0.052840) with: {'C': 1, 'epsilon': 0.1, 'gamma': 'auto', 'kernel': 'rbf'}
-0.430618 (0.042785) with: {'C': 1, 'epsilon': 0.2, 'gamma': 'scale', 'kernel': 'linear'}
-0.412793 (0.055503) with: {'C': 1, 'epsilon': 0.2, 'gamma': 'scale', 'kernel': 'rbf'}
-0.430618 (0.042785) with: {'C': 1, 'epsilon': 0.2, 'gamma': 'auto', 'kernel': 'linear'}
-0.412871 (0.055625) with: {'C': 1, 'epsilon': 0.2, 'gamma': 'auto', 'kernel': 'rbf'}
-0.433512 (0.054104) with: {'C': 1, 'epsilon': 0.5, 'gamma': 'scale', 'kernel': 'linear'}
-0.436005 (0.057754) with: {'C': 1, 'epsilon': 0.5, 'gamma': 'scale', 'kernel': 'rbf'}
-0.433512 (0.054104) with: {'C': 1, 'epsilon': 0.5, 'gamma': 'auto', 'kernel': 'linear'}
-0.436099 (0.057588) with: {'C': 1, 'epsilon': 0.5, 'gamma': 'auto', 'kernel': 'rbf'}
-0.447068 (0.057428) with: {'C': 1, 'epsilon': 1.0, 'gamma': 'scale', 'kernel': 'linear'}
-0.471400 (0.061251) with: {'C': 1, 'epsilon': 1.0, 'gamma': 'scale', 'kernel': 'rbf'}
```

```
-0.447068 (0.057428) with: {'C': 1, 'epsilon': 1.0, 'gamma': 'auto', 'kernel': 'linear'}
-0.471490 (0.061713) with: {'C': 1, 'epsilon': 1.0, 'gamma': 'auto', 'kernel': 'rbf'}
-0.432184 (0.040692) with: {'C': 10, 'epsilon': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
-0.458361 (0.054848) with: {'C': 10, 'epsilon': 0.1, 'gamma': 'scale', 'kernel': 'rbf'}
-0.432184 (0.040692) with: {'C': 10, 'epsilon': 0.1, 'gamma': 'auto', 'kernel': 'linear'}
-0.457937 (0.054793) with: {'C': 10, 'epsilon': 0.1, 'gamma': 'auto', 'kernel': 'rbf'}
-0.430497 (0.042665) with: {'C': 10, 'epsilon': 0.2, 'gamma': 'scale', 'kernel': 'linear'}
-0.452197 (0.051796) with: {'C': 10, 'epsilon': 0.2, 'gamma': 'scale', 'kernel': 'rbf'}
-0.430497 (0.042665) with: {'C': 10, 'epsilon': 0.2, 'gamma': 'auto', 'kernel': 'linear'}
-0.451838 (0.052156) with: {'C': 10, 'epsilon': 0.2, 'gamma': 'auto', 'kernel': 'rbf'}
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import math
from sklearn.metrics import r2_score

# Example dataset (replace with your own data)
# Assuming X and y are already defined
# X, y = your_features, your_target

# Split the data into training and test sets

# Standardize the dataset
scaler = StandardScaler().fit(XtrainResampled)
rescaledX = scaler.transform(XtrainResampled)

from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=42)
XtestResampled, YtestResampled = ros.fit_resample(X_validation, Y_validation)


# Initialize the Decision Tree Regressor with best parameters
model = DecisionTreeRegressor(
    criterion='poisson',
    max_depth=None,
    min_samples_leaf=1,
    min_samples_split=5,
    random_state=42
)

# Train the model
model.fit(rescaledX,YtrainResampled)

# Predict on the test set
y_pred = model.predict(XtestResampled)

# Evaluate the model
mse = mean_squared_error(YtestResampled, y_pred)
print(f" Root Mean Squared Error: {math.sqrt(mse)}")
```

```
Root Mean Squared Error: 1.7795130420052185
```

Ensemble Methods

```
scoring = 'neg_mean_squared_error'
```

```
ensembles = []
ensembles.append(('ScaledAB', Pipeline([('Scaler', StandardScaler()),('AB',AdaBoostRegressor())])))
ensembles.append(('ScaledGBM', Pipeline([('Scaler', StandardScaler()),('GBM',GradientBoostingRegressor())])))
ensembles.append(('ScaledRF', Pipeline([('Scaler', StandardScaler()),('RF',RandomForestRegressor())])))
ensembles.append(('ScaledET', Pipeline([('Scaler', StandardScaler()),('ET',ExtraTreesRegressor())])))
results = []
names = []
for name, model in ensembles:
    kfold = KFold(n_splits=num_folds)
    cv_results = cross_val_score(model,X_train, Y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
ScaledAB: -0.424884 (0.054945)
ScaledGBM: -0.399110 (0.047077)
ScaledRF: -0.373412 (0.040801)
ScaledET: -0.362211 (0.030854)
```

Extra Trees Regressor with Tuning

```
df.columns
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality', 'Id'],
      dtype='object')
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from imblearn.over_sampling import RandomOverSampler
from collections import Counter

# Load your dataset (replace this with your dataset)
# Assume 'df' contains the data and 'target' is the regression target variable
# Example: df = pd.read_csv("your_dataset.csv")
x = df.drop('quality', axis=1)  # Replace 'target' with your target variable name
y = df['quality']

# Split the data into training and validation sets (40% validation set)
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.4, random_state=42)

# Scale the features
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)

# Address imbalance (if applicable, for categorical targets in regression scenarios)
# Use oversampling only if necessary


# Initialize the ExtraTreesRegressor
model = ExtraTreesRegressor(random_state=42)

# Define the parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 500],  # Number of trees
    'max_depth': [None, 10, 20, 30],  # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],    # Minimum samples required to be a leaf node
}

# Set up cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Perform GridSearchCV for parameter tuning
grid_search = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=kfold,
    scoring='neg_mean_squared_error',  # Regression scoring
    n_jobs=-1,
    verbose=1
)

# Fit the model to the resampled training data
grid_search.fit(x_train, y_train)

# Print the best parameters and the corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation MSE:", -grid_search.best_score_)

# Use the best model from GridSearchCV
best_model = grid_search.best_estimator_

# Train the model on the full resampled training set
best_model.fit(x_train, y_train)

# Validate the model on the validation set
```

```
y_val_pred = best_model.predict(x_val)

# Evaluate performance on the validation set
mse = mean_squared_error(y_val, y_val_pred)
mae = mean_absolute_error(y_val, y_val_pred)
print("Validation MSE:", mse)
print("Validation MAE:", mae)
```

```
Fitting 10 folds for each of 108 candidates, totalling 1080 fits
Best Parameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best Cross-Validation MSE: 0.3581397224517188
Validation MSE: 0.37627985670536773
Validation MAE: 0.4368733813948061
```

```
X_train.shape
```

```
(914, 11)
```

```
X_test.shape
```

```
(6192, 8)
```

Finalize the model for Regression

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import math
from sklearn.metrics import r2_score

# Example dataset (replace with your own data)
# Assuming X and y are already defined
# X, y = your_features, your_target

# Split the data into training and test sets
validation_size = 0.20
X_train, X_validation, Y_train, Y_validation = train_test_split(x, y,test_size=validation_size)

# Standardize the dataset
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
rescaledTestX = scaler.transform(X_validation)
```

```
# Initialize the Decision Tree Regressor with best parameters
model = ExtraTreesRegressor(
    n_estimators=200,
    max_depth=20,
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42
)
```

```
# Train the model
model.fit(rescaledX,Y_train)

# Predict on the test set
y_pred = model.predict(rescaledTestX)

# Evaluate the model
mse = mean_squared_error(Y_validation, y_pred)
print(f" Root Mean Squared Error: {math.sqrt(mse)}")
print(f" R^2 Score: {r2_score(Y_validation, y_pred)}")
```

```
Root Mean Squared Error: 0.5530515864933482
R^2 Score: 0.4828834253638066
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

+ Code + Text

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.