Python初級數據分析員證書

（六）數據分析 及可視化專案

13. 數據分析專案 Demo 16 – Red Wine Quality

Review

- Statistics
- Hypothesis testing
- Algebra
- Linear regression
- Propositional logic
- Python
- R
- SQL

- Pandas, NumPy, SciPy
- Data Visualization, Matplotlib, Seaborn, Plotly
- Dashboard Visualization, Business Intelligence
- Storytelling

Chapter Summary

- Scenario
- Data Import
- EDA
- RandomForestClassifier
- SupportVectorClassifier
- GridSearchCV

Scenario

This chapter let’s discover the variant from Portuguese "Vinho Verde" wine. What might be an interesting thing to do, is aside from using regression modelling, is to set an arbitrary cut off for your dependent variable (wine quality) at e.g. 7 or higher getting classified as 'good/1' and the remainder as 'not good/0’.

Acknowledgements
This dataset is also available from the UCI machine learning repository, https://archive.ics.uci.edu/ml/datasets/wine+quality
Please include this citation if you plan to use this database: P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modelling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

Data import

```
1  import pandas as pd
2  import seaborn as sns
3  import matplotlib.pyplot as plt
4  from sklearn.ensemble import RandomForestClassifier
5  from sklearn.svm import SVC
6  from sklearn.linear_model import SGDClassifier
7  from sklearn.metrics import confusion_matrix, classification_report
8  from sklearn.preprocessing import StandardScaler, LabelEncoder
9  from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
10 %matplotlib inline
```

```
1  wine = pd.read_csv('RedWineQuality.csv')
2  wine.head(3)
```

	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

Variables and Output

Input variables (based on physicochemical tests):

1 - fixed acidity
2 - volatile acidity
3 - citric acid
4 - residual sugar
5 - chlorides
6 - free sulfur dioxide
7 - total sulfur dioxide
8 - density
9 - pH
10 - sulphates
11 – alcohol

Output variable (based on sensory data):

12 - quality (score between 0 and 10)

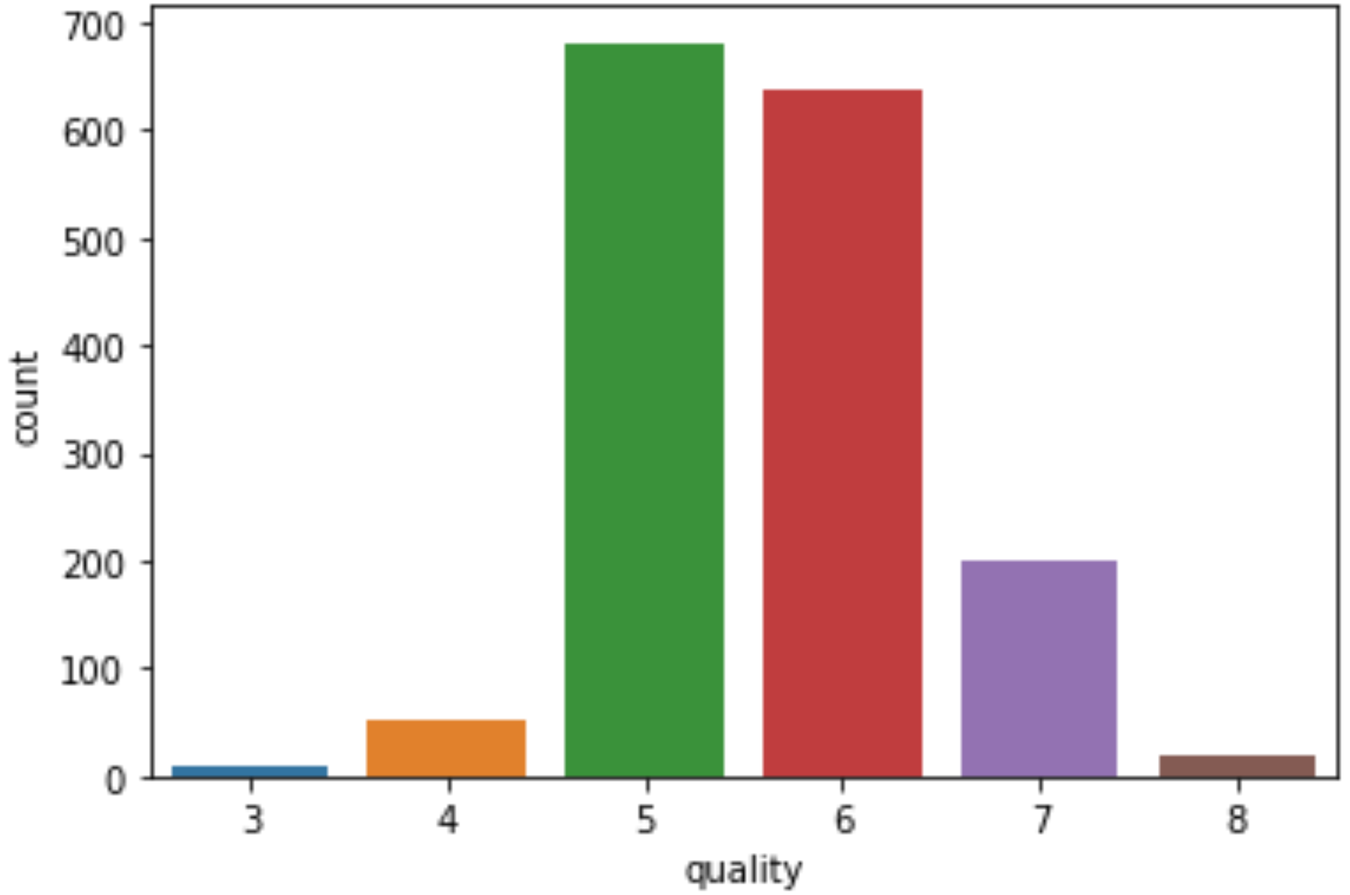
Overview all columns

```
1 wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

Quality distribution

```
1 #count of the target variable
2 sns.countplot(x='quality', data=wine)
```

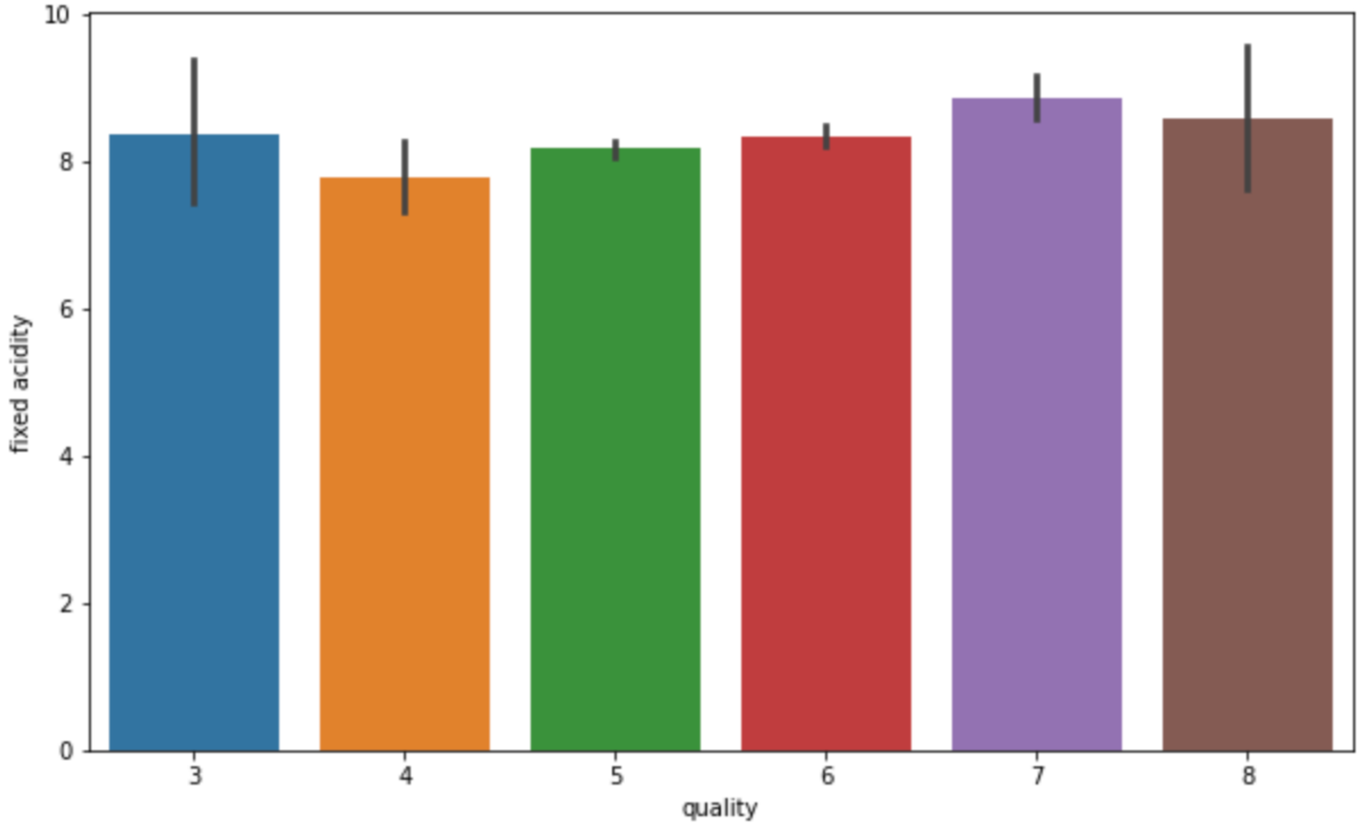


Fixed acidity

Here we see that fixed acidity does not give any specification to classify the quality.

```
1 fig = plt.figure(figsize = (10,6))
2 sns.barplot(x = 'quality', y = 'fixed acidity', data = wine)
```

```
<AxesSubplot:xlabel='quality', ylabel='fixed acidity'>
```

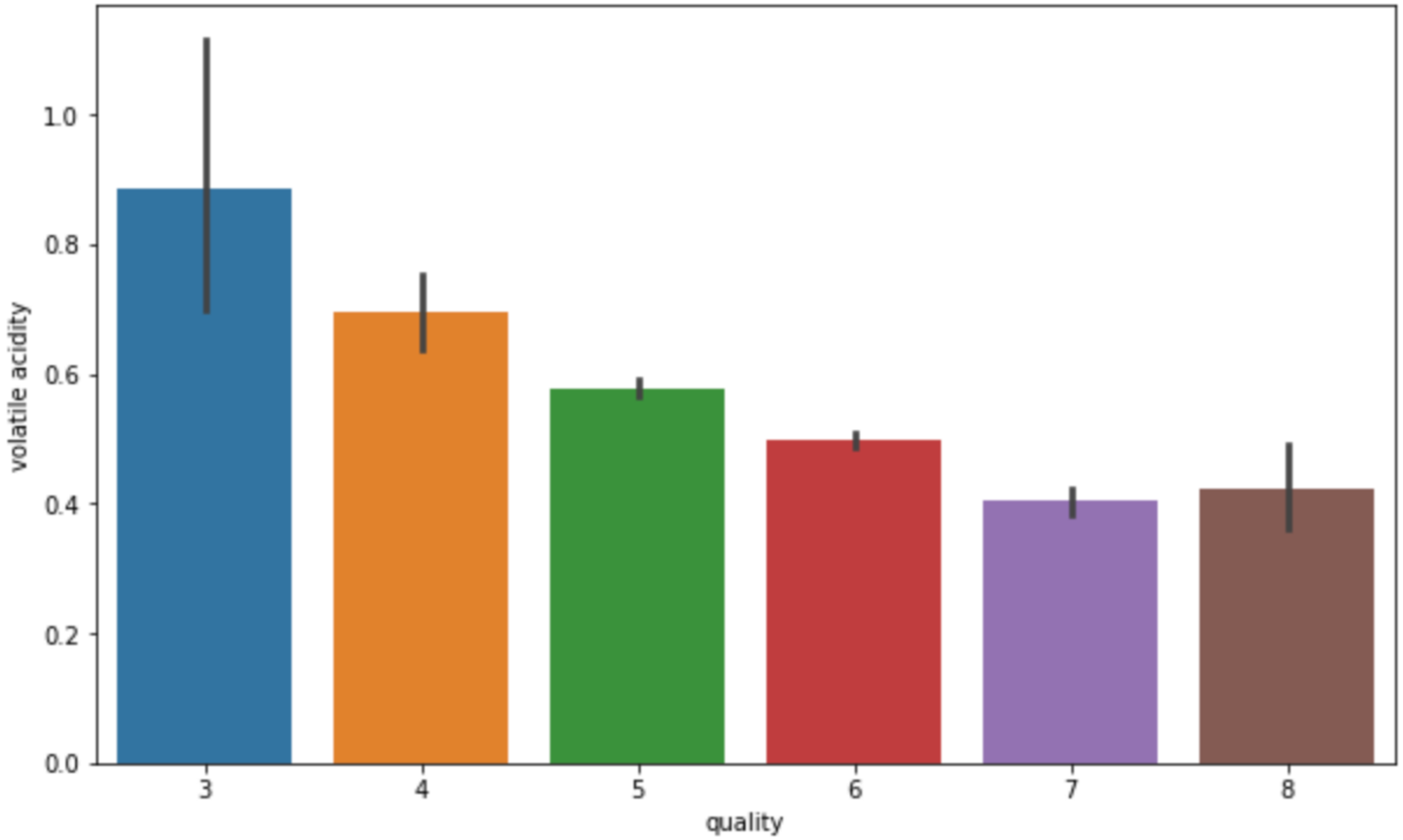


Volatile acidity

Downing trend in the volatile acidity as we go higher the quality

```
1  fig = plt.figure(figsize = (10,6))
2  sns.barplot(x = 'quality', y = 'volatile acidity', data = wine)
```

```
<AxesSubplot:xlabel='quality', ylabel='volatile acidity'>
```

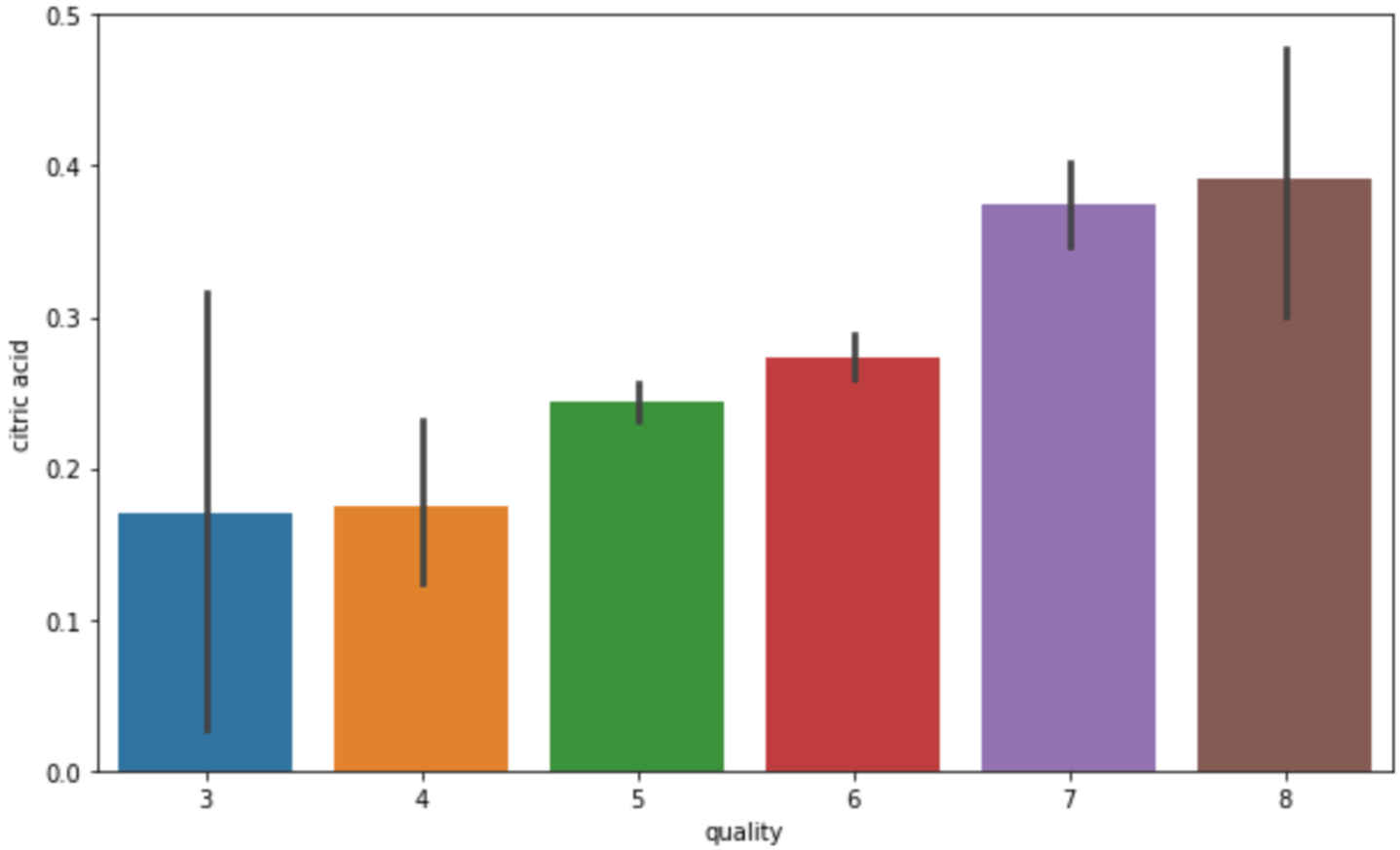


Citric acid

citric acid go higher as we go higher in the quality of the wine

```
1  fig = plt.figure(figsize = (10,6))
2  sns.barplot(x = 'quality', y = 'citric acid', data = wine)
```

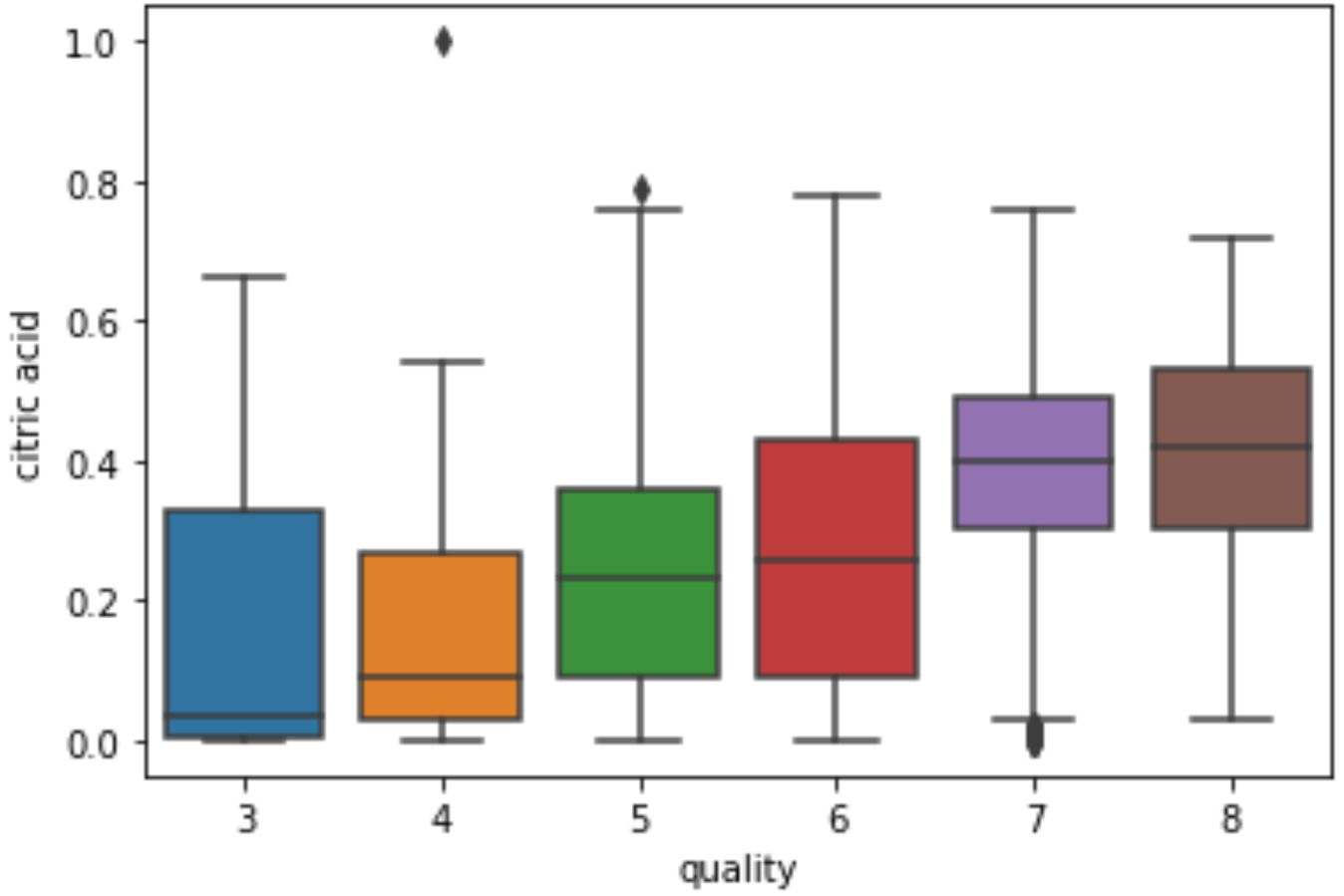
```
<AxesSubplot:xlabel='quality', ylabel='citric acid'>
```



Citric acid

```
1 sns.boxplot(x='quality', y='citric acid', data = wine)
```

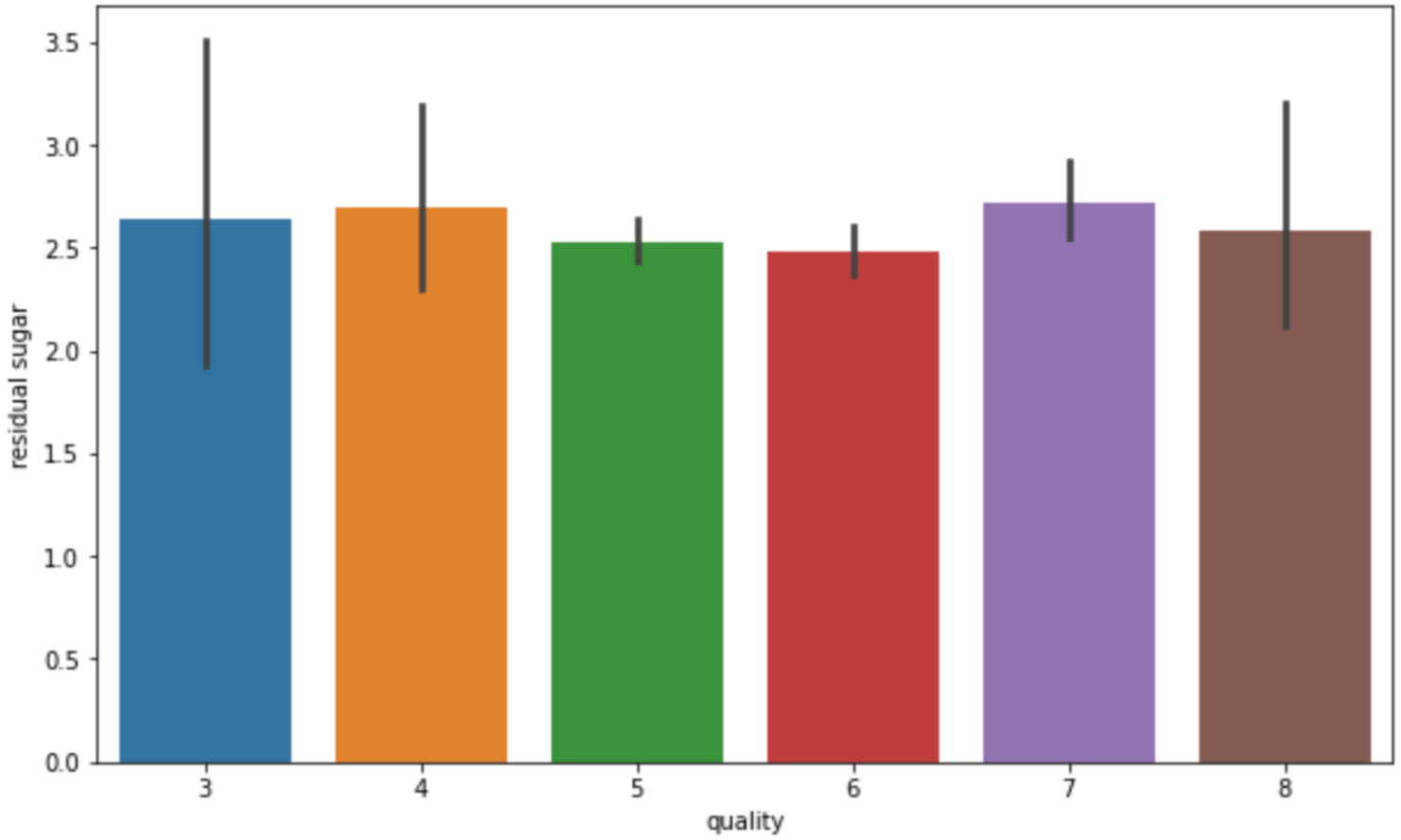
```
<AxesSubplot:xlabel='quality', ylabel='citric acid'>
```



residual sugar

```
1  fig = plt.figure(figsize = (10,6))
2  sns.barplot(x = 'quality', y = 'residual sugar', data = wine)
```

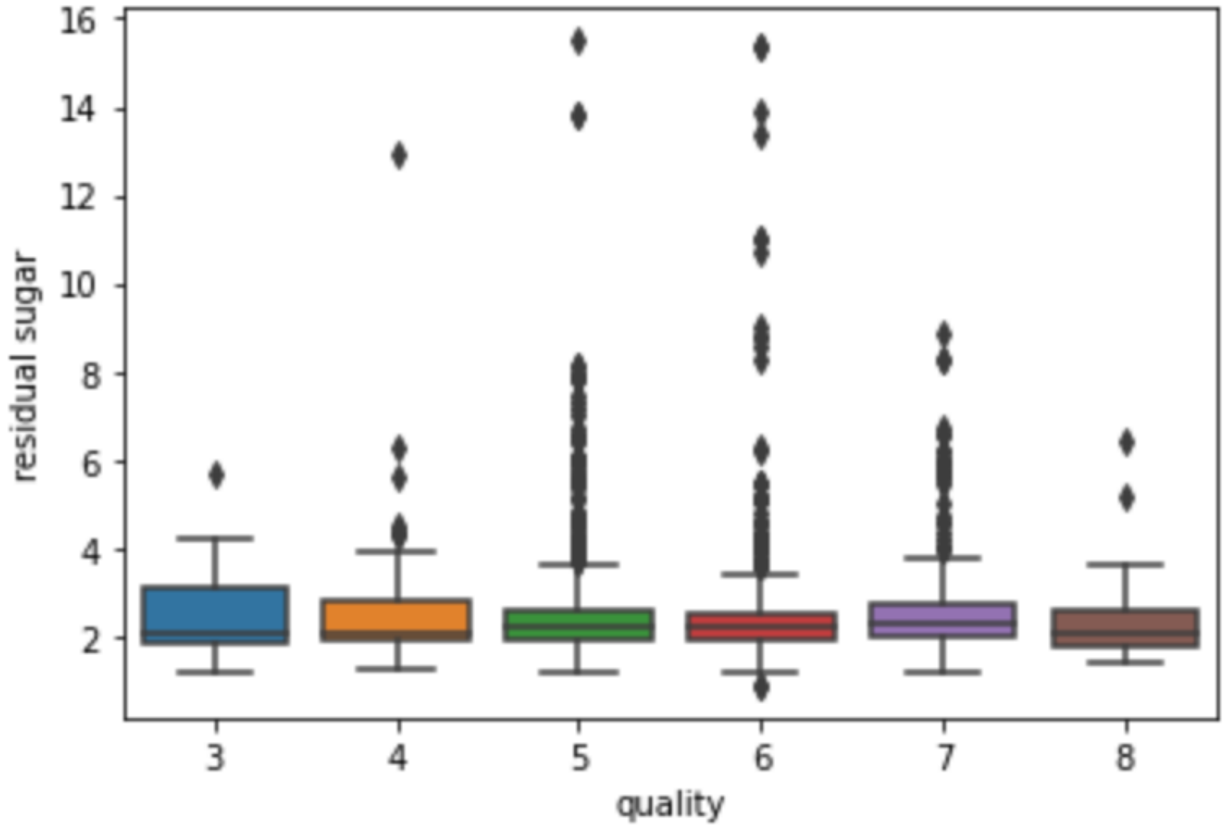
```
<AxesSubplot:xlabel='quality', ylabel='residual sugar'>
```



residual sugar

```
1 sns.boxplot(x='quality', y='residual sugar', data = wine)
```

```
<AxesSubplot:xlabel='quality', ylabel='residual sugar'>
```

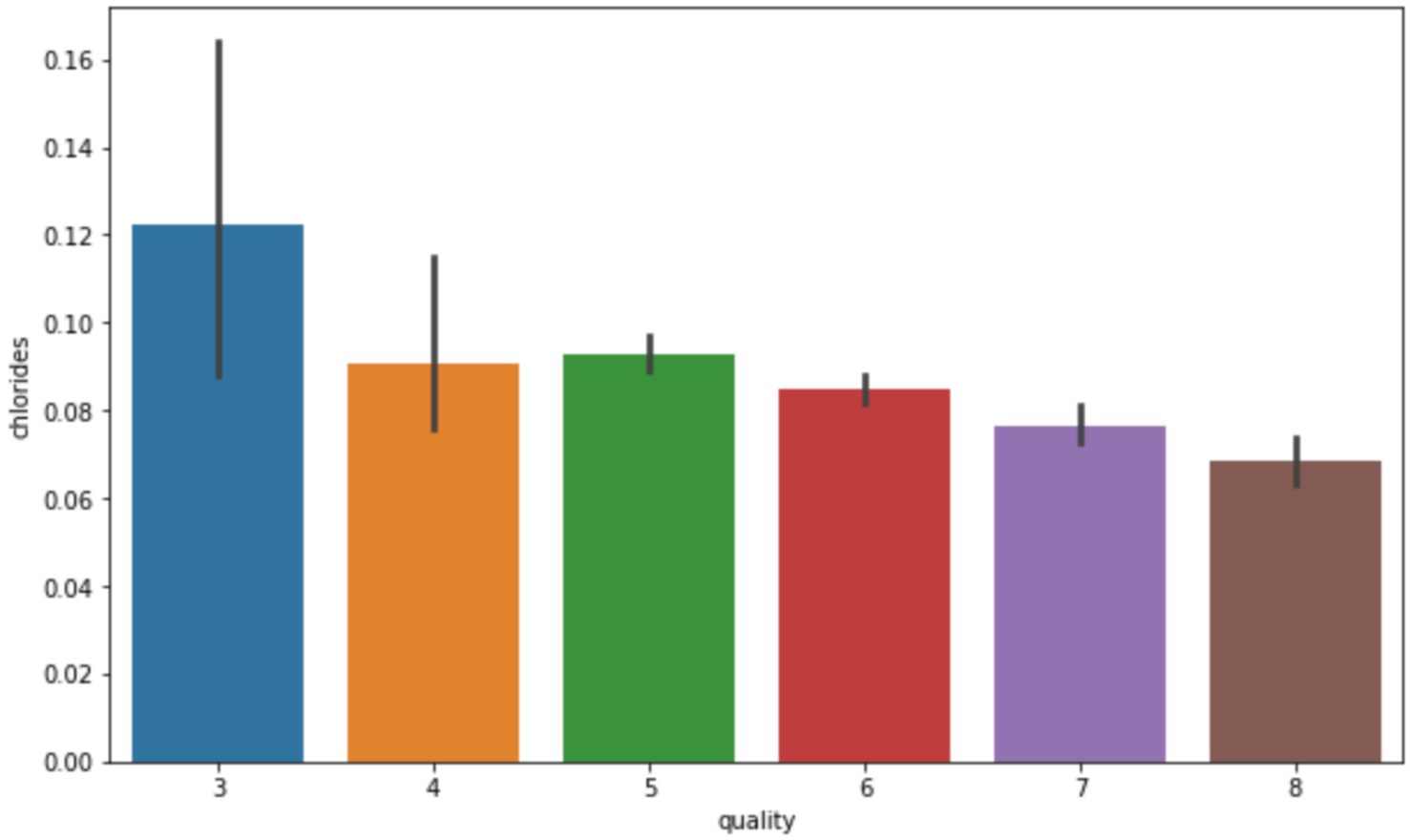


chloride

chloride also go down as we go higher in the quality of the wine

```
1  fig = plt.figure(figsize = (10,6))
2  sns.barplot(x = 'quality', y = 'chlorides', data = wine)
```

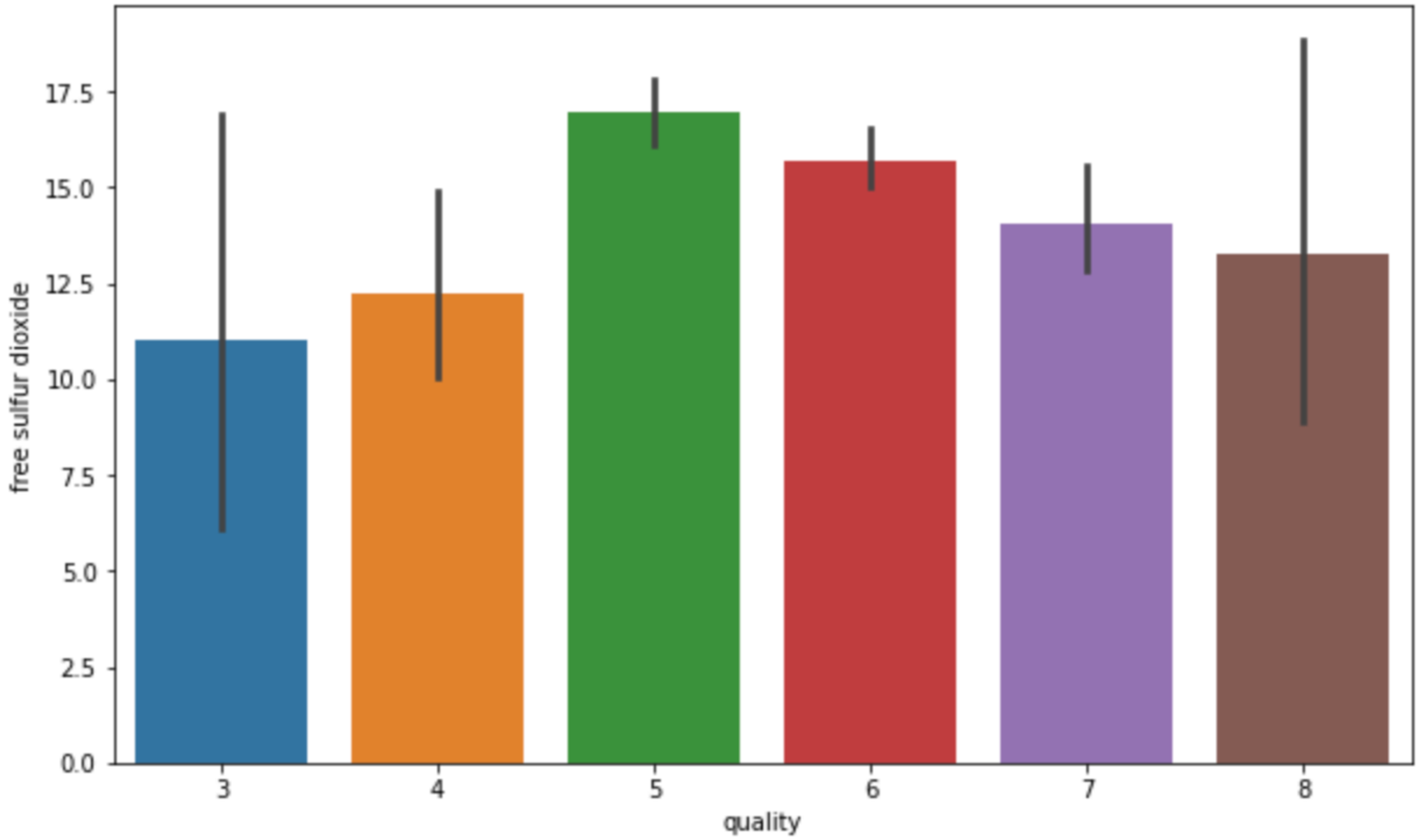
```
<AxesSubplot:xlabel='quality', ylabel='chlorides'>
```



free sulfur dioxide

```
1  fig = plt.figure(figsize = (10,6))
2  sns.barplot(x = 'quality', y = 'free sulfur dioxide', data = wine)
```

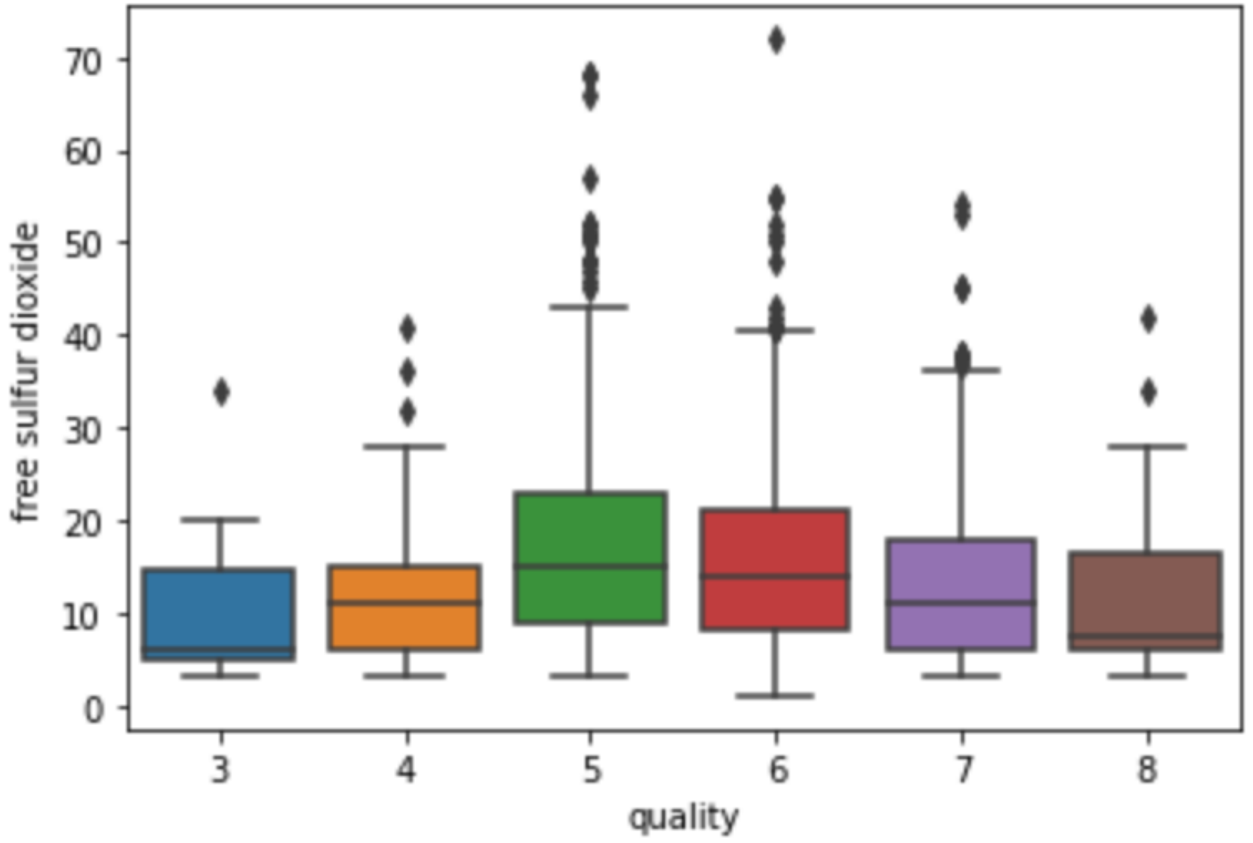
```
<AxesSubplot:xlabel='quality', ylabel='free sulfur dioxide'>
```



free sulfur dioxide

```
1 sns.boxplot(x='quality', y='free sulfur dioxide', data = wine)
```

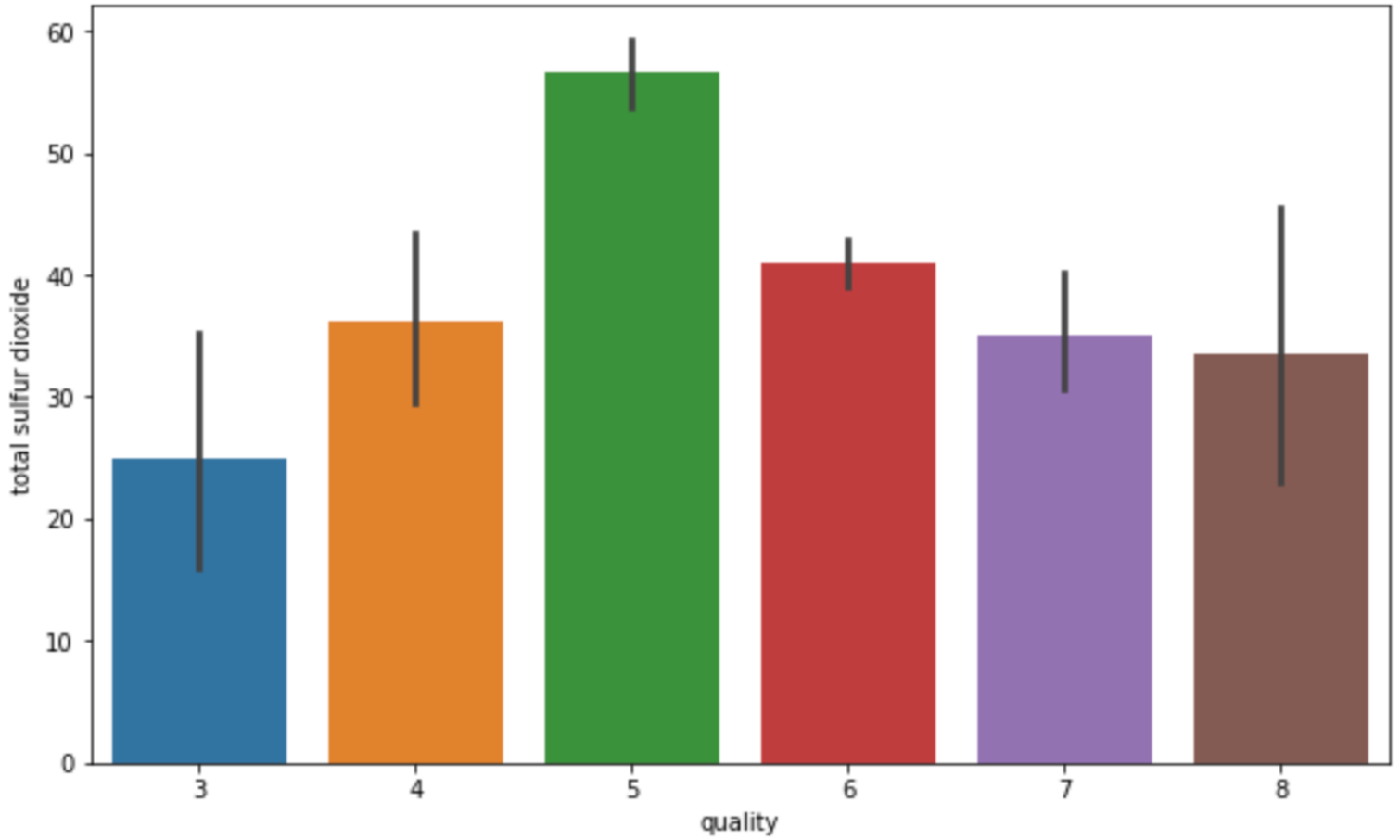
```
<AxesSubplot:xlabel='quality', ylabel='free sulfur dioxide'>
```



total sulfur dioxide

```
1  fig = plt.figure(figsize = (10,6))
2  sns.barplot(x = 'quality', y = 'total sulfur dioxide', data = wine)
```

```
<AxesSubplot:xlabel='quality', ylabel='total sulfur dioxide'>
```

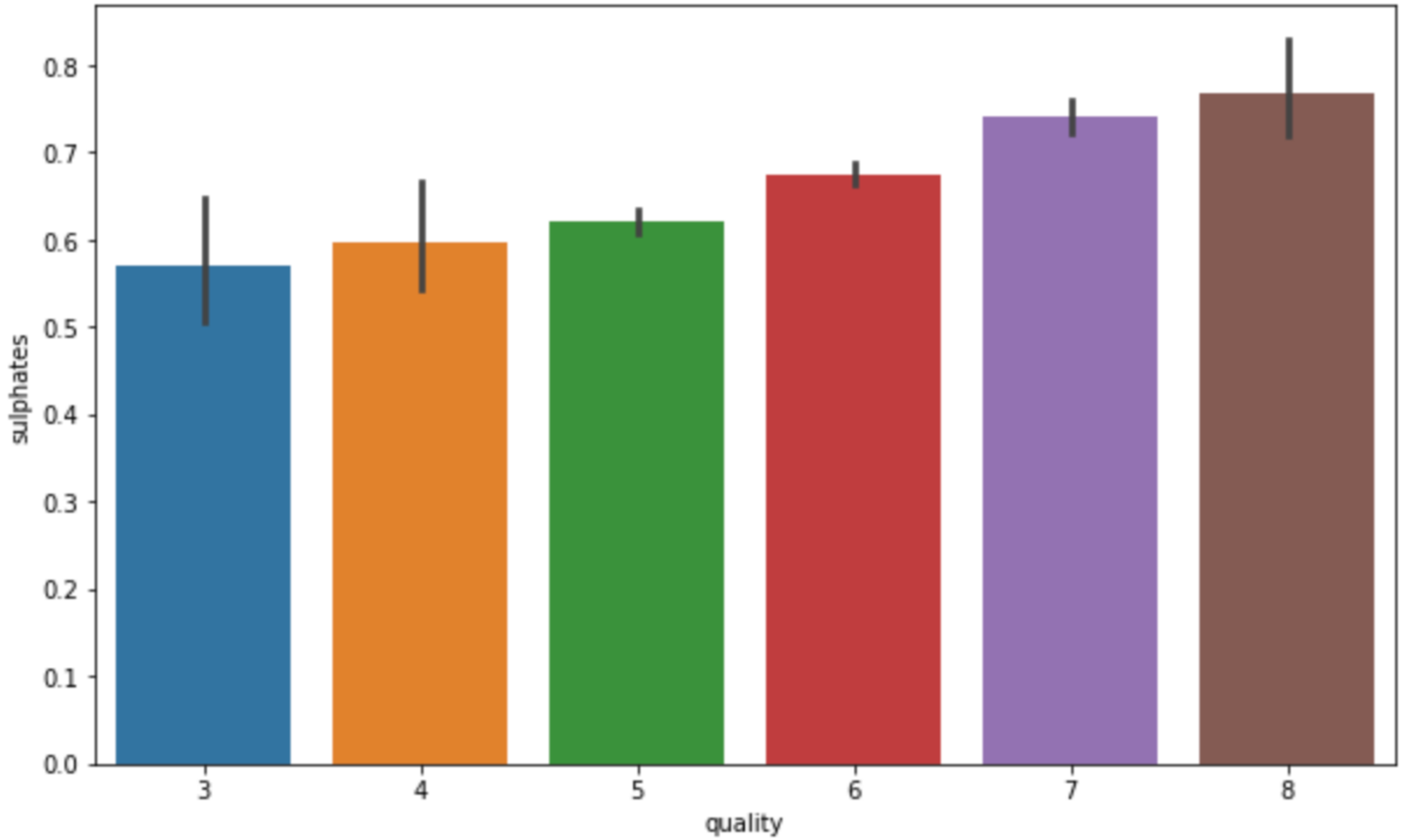


Sulphates

Sulphates level goes higher with the quality of wine

```
1  fig = plt.figure(figsize = (10,6))
2  sns.barplot(x = 'quality', y = 'sulphates', data = wine)
```

```
<AxesSubplot:xlabel='quality', ylabel='sulphates'>
```

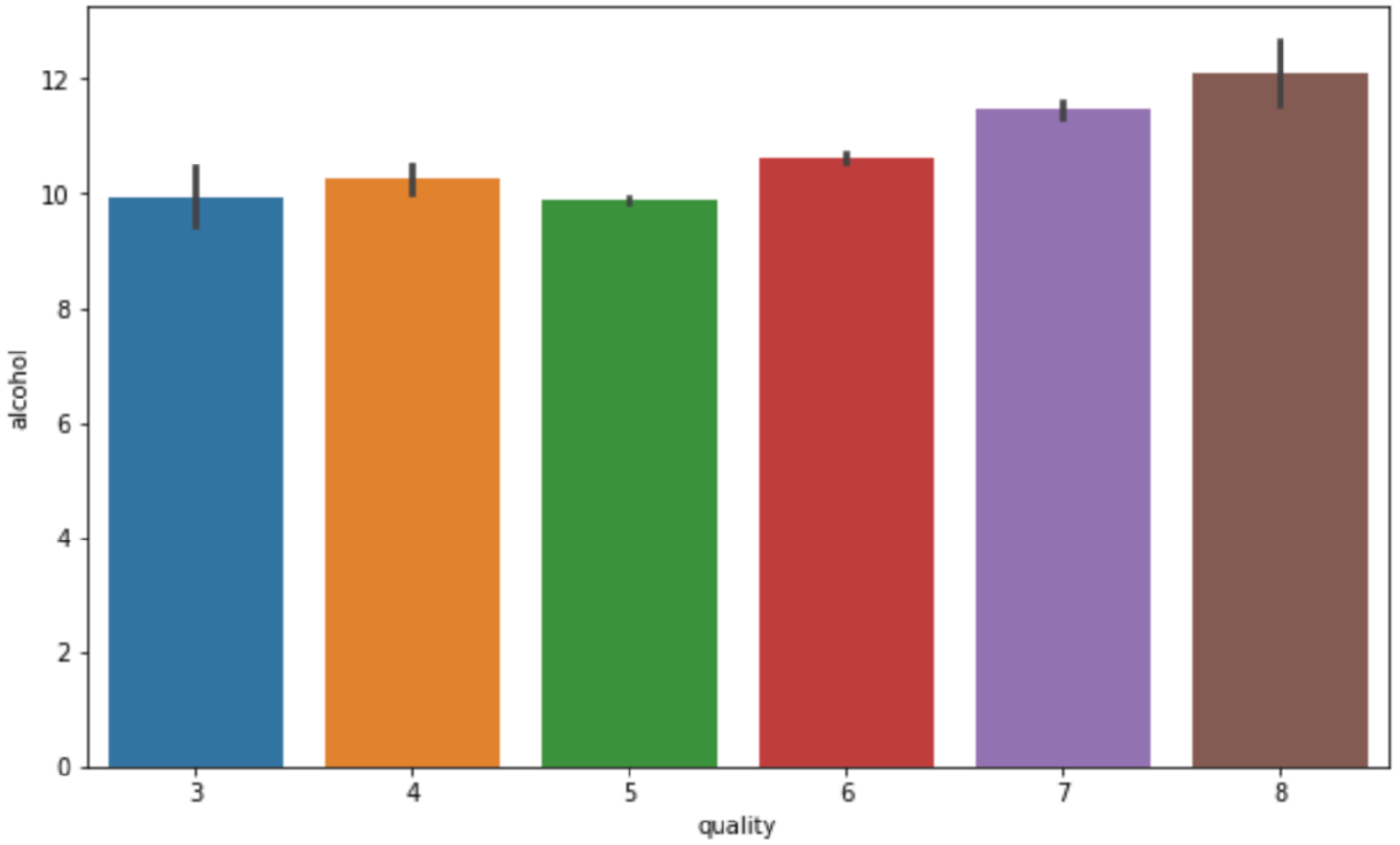


Alcohol

Alcohol level also goes higher as the quality of wine increases

```
1  fig = plt.figure(figsize = (10,6))
2  sns.barplot(x = 'quality', y = 'alcohol', data = wine)
```

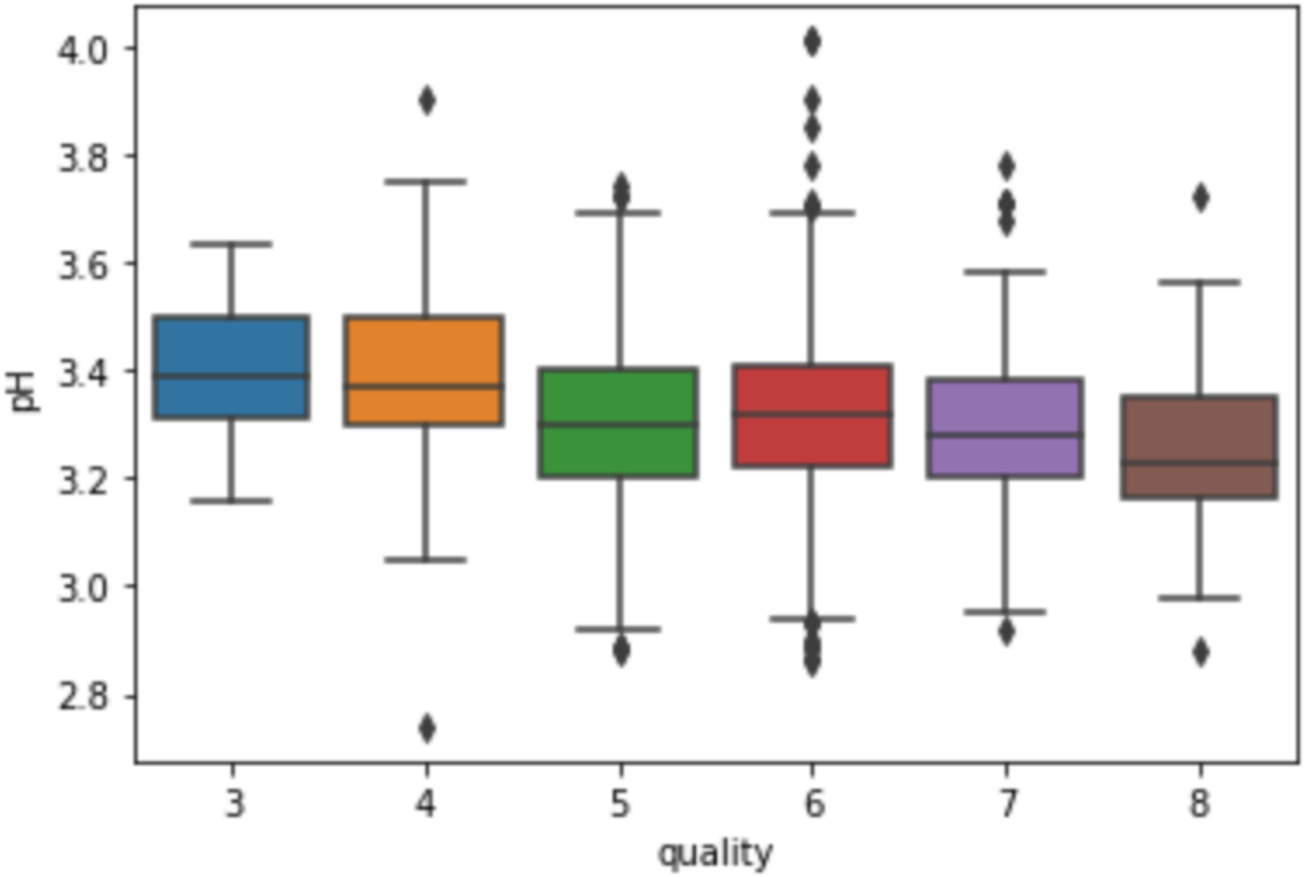
```
<AxesSubplot:xlabel='quality', ylabel='alcohol'>
```



pH

```
1 sns.boxplot(x='quality', y='pH', data = wine)
```

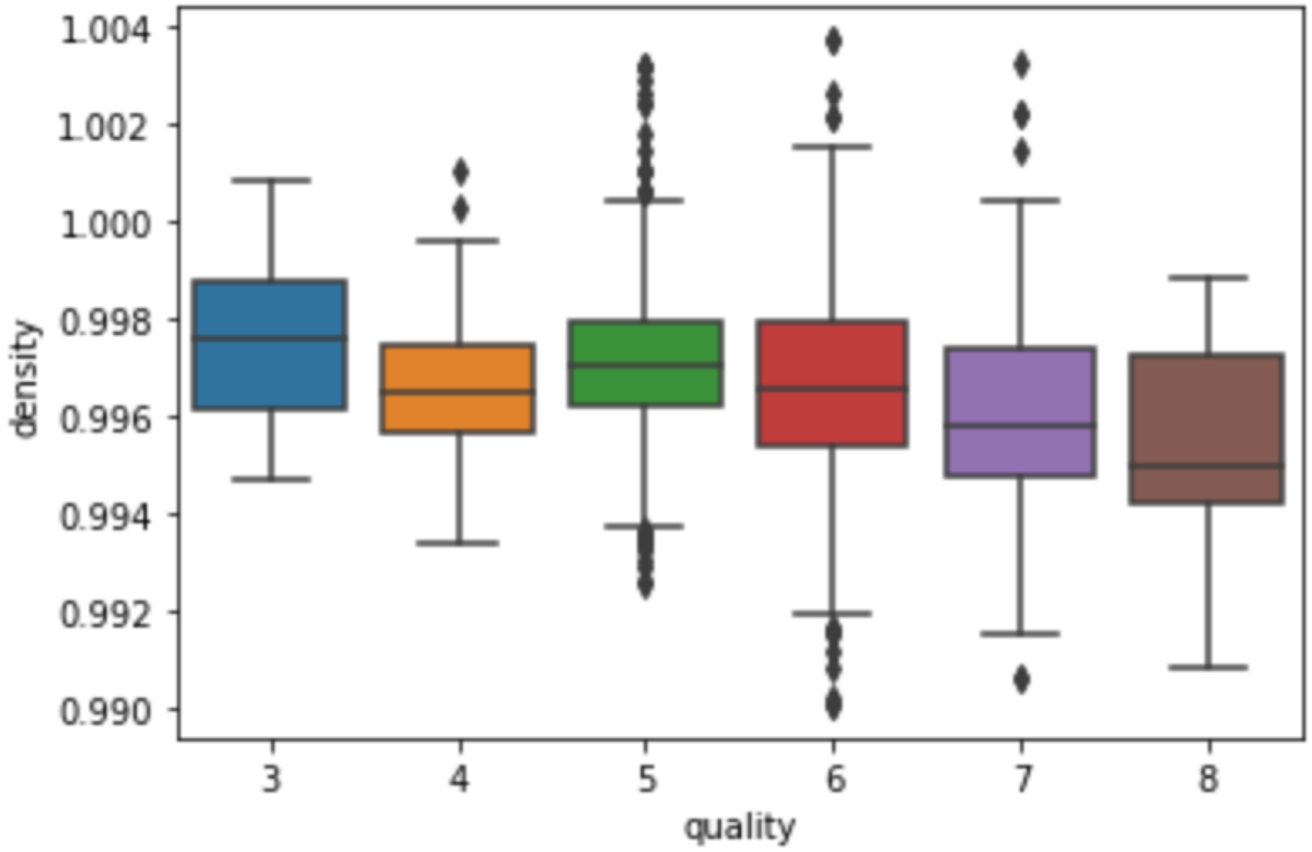
```
<AxesSubplot:xlabel='quality', ylabel='pH'>
```



Density

```
1  sns.boxplot(x='quality', y='density', data = wine)
```

```
<AxesSubplot:xlabel='quality', ylabel='density'>
```



Preprocessing Data for Machine Learning

Labelling target

```
#Making binary classificaion for the response variable.
#Dividing wine as good and bad by giving the limit for the quality
bins = (2, 6.5, 8)
group_names = ['bad', 'good']
wine['quality'] = pd.cut(wine['quality'], bins = bins, labels = group_names)
```

```
#Now lets assign a labels to our quality variable
label_quality = LabelEncoder()
```

Preprocessing Data for Machine Learning

```
1  #Bad becomes 0 and good becomes 1
2  wine['quality'] = label_quality.fit_transform(wine['quality'])
```

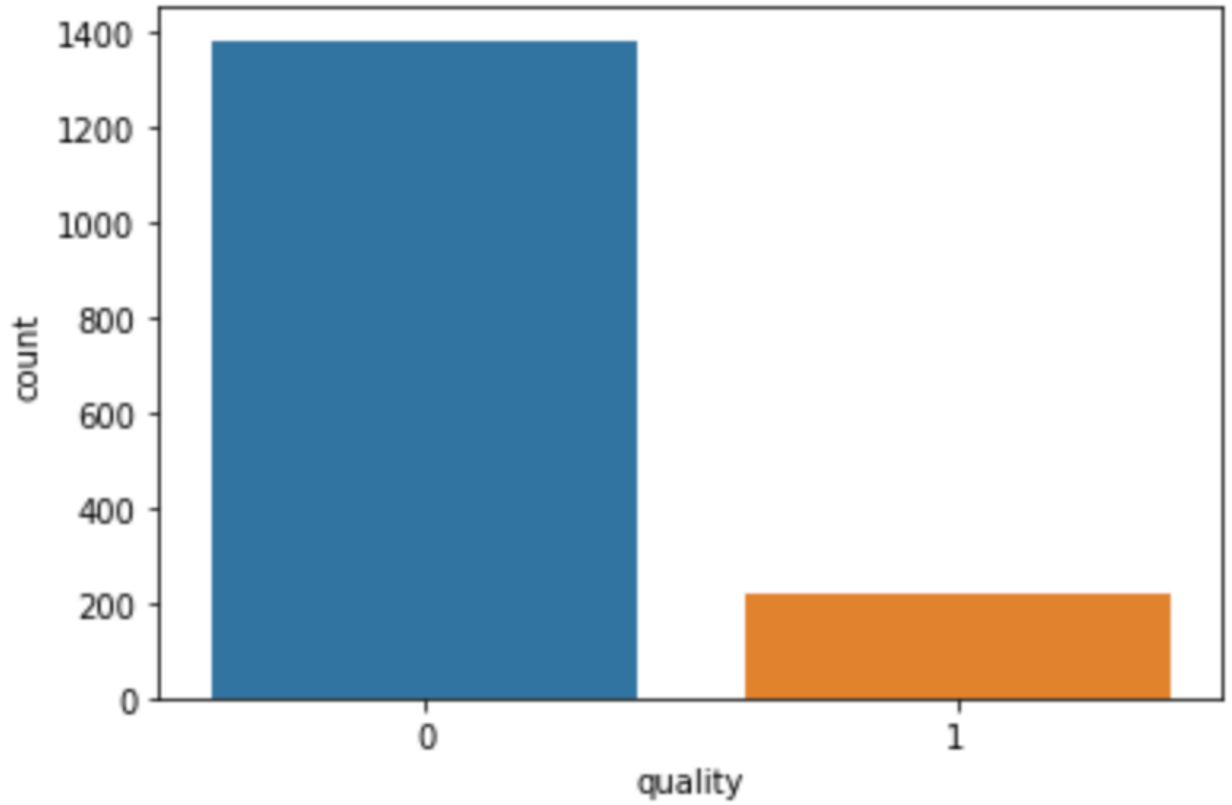
```
1  wine['quality'].value_counts()
```

```
0    1382
1     217
Name: quality, dtype: int64
```

Target countplot

```
1 sns.countplot(x=wine['quality'])
```

```
<AxesSubplot:xlabel='quality', ylabel='count'>
```



Bad : 0

good : 1

Train and Test splitting

```
#Now seperate the dataset as response variable and feature variabes
X = wine.drop('quality', axis = 1)
y = wine['quality']
```

```
#Train and Test splitting of data
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size = 0.2, random_state = 42)
```

```
#Applying Standard scaling to get optimized result
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

RandomForestClassifier

```
1 rfc = RandomForestClassifier(n_estimators=200)
2 rfc.fit(X_train, y_train)
3 pred_rfc = rfc.predict(X_test)
```

```
1 #Let's see how our model performed
2 print(classification_report(y_test, pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.90      0.97      0.94       273
           1       0.70      0.40      0.51        47

    accuracy                           0.89       320
   macro avg       0.80      0.69      0.72       320
weighted avg       0.87      0.89      0.87       320
```

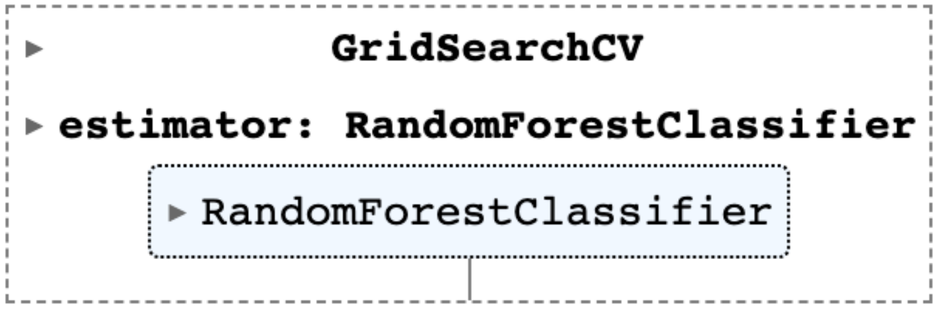
Random forest gives the accuracy of 87%

Fine tune RandomForest

```
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
```

```
%%time
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(X_train, y_train)
```

```
CPU times: user 2min 1s, sys: 340 ms, total: 2min 2s
Wall time: 2min 2s
```



Fine tune RandomForest

```
1 CV_rfc.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 8,
 'max_features': 'log2',
 'n_estimators': 500}
```

```
1 rfc1=RandomForestClassifier(random_state=42, max_features='sqrt',
2                             n_estimators= 200, max_depth=8, criterion='gini')
```

```
1 rfc1.fit(X_train, y_train)
```

```
▼ RandomForestClassifier
RandomForestClassifier(max_depth=8, n_estimators=200, random_state=42)
```

Fine tune RandomForest

We got slightly improved accuracy 0.878125 after simple GridSearch.

```
1  pred=rfc1.predict(X_test)
```

```
1  # the accuracy is slightly higher
2  from sklearn.metrics import accuracy_score
3  print("Accuracy for Random Forest on CV data: ",accuracy_score(y_test,pred))
```

```
Accuracy for Random Forest on CV data:  0.878125
```

Support Vector Classifier

```
1 svc = SVC()
2 svc.fit(X_train, y_train)
3 pred_svc = svc.predict(X_test)
```

```
1 print(classification_report(y_test, pred_svc))
```

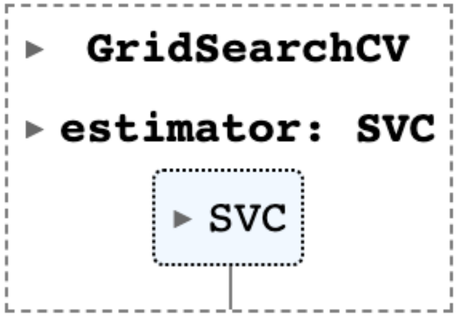
```
              precision    recall  f1-score   support

           0       0.88      0.98      0.93       273
           1       0.71      0.26      0.37        47

    accuracy                           0.88       320
   macro avg       0.80      0.62      0.65       320
weighted avg       0.86      0.88      0.85       320
```

Fine tune accuracy of models - Grid Search CV

```
#Finding best parameters for our SVC model
param = {
    'C': [0.1,0.8,0.9,1,1.1,1.2,1.3,1.4],
    'kernel':['linear', 'rbf'],
    'gamma' :[0.1,0.8,0.9,1,1.1,1.2,1.3,1.4]
}
grid_svc = GridSearchCV(svc, param_grid=param, scoring='accuracy', cv=10)
```

```
%%time
grid_svc.fit(X_train, y_train)
```

```
CPU times: user 46.1 s, sys: 120 ms, total: 46.2 s
Wall time: 46.4 s
```



Fine tune accuracy of models

```
#Best parameters for our svc model
grid_svc.best_params_
```

```
{'C': 1.2, 'gamma': 0.9, 'kernel': 'rbf'}
```

```
#Let's run our SVC again with the best parameters.
svc2 = SVC(C = 1.2, gamma =  0.9, kernel= 'rbf')
svc2.fit(X_train, y_train)
pred_svc2 = svc2.predict(X_test)
print(classification_report(y_test, pred_svc2))
```

```
              precision    recall  f1-score   support

           0       0.90      0.99      0.94       273
           1       0.89      0.34      0.49        47

    accuracy                           0.90       320
   macro avg       0.89      0.67      0.72       320
weighted avg       0.90      0.90      0.88       320
```

SVC improves from 86% to 90% using Grid Search CV

Support vector machines SVM

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

Grid Search CV

GridSearchCV implements a “fit” and a “score” method. It also implements “score_samples”, “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used.

The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

Reference & Resources

Sklearn Website:

https://scikit-learn.org/

Plotly Graph Objects:

https://plotly.com/python/graph-objects/

Seaborn:

https://seaborn.pydata.org/examples/index.html

Matplotlib:

https://matplotlib.org/