# wine-analysis

July 4, 2023

[1]:
```
import pandas as pd
```

[2]:
```
from google.colab import files
upload =files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving 1788410-1767134-1729261-1613779-Red_wine__(1) (2).csv to
1788410-1767134-1729261-1613779-Red_wine__(1) (2).csv
```

[3]:
```
df = pd.read_csv('1788410-1767134-1729261-1613779-Red_wine__(1) (2).csv')
```

[4]:
```
df.head(10)
```

[4]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076
5            7.4              0.66         0.00             1.8      0.075
6            7.9              0.60         0.06             1.6      0.069
7            7.3              0.65         0.00             1.2      0.065
8            7.8              0.58         0.02             2.0      0.073
9            7.5              0.50         0.36             6.1      0.071

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56
5                 13.0                  40.0   0.9978  3.51       0.56
6                 15.0                  59.0   0.9964  3.30       0.46
7                 15.0                  21.0   0.9946  3.39       0.47
8                  9.0                  18.0   0.9968  3.36       0.57
9                 17.0                   NaN   0.9978  3.35       0.80
```

```
   alcohol  quality
0      9.4      5.0
1      9.8      5.0
2      9.8      5.0
3      9.8      6.0
4      9.4      5.0
5      9.4      5.0
6      9.4      5.0
7     10.0      7.0
8      9.5      7.0
9     10.5      5.0
```

[5]:
```
df.shape
```

[5]:
```
(1599, 12)
```

[6]:
```
df.info()
df.isnull().sum()
```

```
<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  1598 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1598 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1598 non-null   float64
dtypes: float64(12)
memory usage: 150.0 KB
```

[6]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    1
density                 0
```

```
pH                      1
sulphates               0
alcohol                 0
quality                 1
dtype: int64
```

[7]:
```python
df.describe()
```

[7]:

| | fixed acidity | volatile acidity | citric acid | residual sugar |
|---|---|---|---|---|
| count | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 |
| mean | 8.319637 | 0.527821 | 0.270976 | 2.538806 |
| std | 1.741096 | 0.179060 | 0.194801 | 1.409928 |
| min | 4.600000 | 0.120000 | 0.000000 | 0.900000 |
| 25% | 7.100000 | 0.390000 | 0.090000 | 1.900000 |
| 50% | 7.900000 | 0.520000 | 0.260000 | 2.200000 |
| 75% | 9.200000 | 0.640000 | 0.420000 | 2.600000 |
| max | 15.900000 | 1.580000 | 1.000000 | 15.500000 |

| | chlorides | free sulfur dioxide | total sulfur dioxide | density |
|---|---|---|---|---|
| count | 1599.000000 | 1599.000000 | 1598.000000 | 1599.000000 |
| mean | 0.087467 | 15.874922 | 46.433041 | 0.996747 |
| std | 0.047065 | 10.460157 | 32.876249 | 0.001887 |
| min | 0.012000 | 1.000000 | 6.000000 | 0.990070 |
| 25% | 0.070000 | 7.000000 | 22.000000 | 0.995600 |
| 50% | 0.079000 | 14.000000 | 38.000000 | 0.996750 |
| 75% | 0.090000 | 21.000000 | 62.000000 | 0.997835 |
| max | 0.611000 | 72.000000 | 289.000000 | 1.003690 |

| | pH | sulphates | alcohol | quality |
|---|---|---|---|---|
| count | 1598.000000 | 1599.000000 | 1599.000000 | 1598.000000 |
| mean | 3.498586 | 0.658149 | 10.422983 | 5.636421 |
| std | 0.080346 | 0.169507 | 1.065668 | 0.807665 |
| min | 2.740000 | 0.330000 | 8.400000 | 3.000000 |
| 25% | 3.520000 | 0.550000 | 9.500000 | 5.000000 |
| 50% | 3.520000 | 0.620000 | 10.200000 | 6.000000 |
| 75% | 3.520000 | 0.730000 | 11.100000 | 6.000000 |
| max | 3.900000 | 2.000000 | 14.900000 | 8.000000 |

[8]:
```python
import matplotlib.pyplot as plt
import seaborn as sb
import numpy as np
%matplotlib inline
```

[9]:
```python
df['quality'].unique()
```

[9]:
```
array([ 5.,  6.,  7.,  4., nan,  8.,  3.])
```

[10]:
```python
df['quality'].value_counts()
```

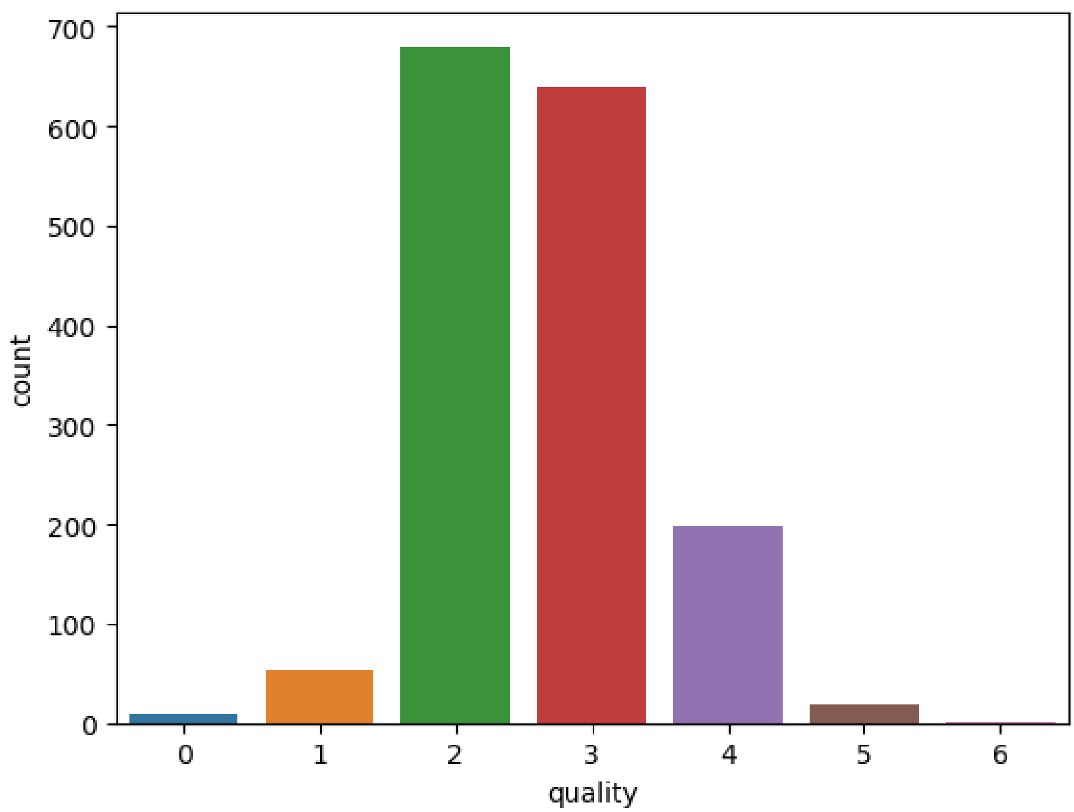
[10]:
```
5.0    680
6.0    638
7.0    199
4.0     53
8.0     18
3.0     10
Name: quality, dtype: int64
```

[54]:
```python
df['quality'].count()
```

[54]:
```
1599
```

[56]:
```python
sb.countplot(x='quality', data=df)
```

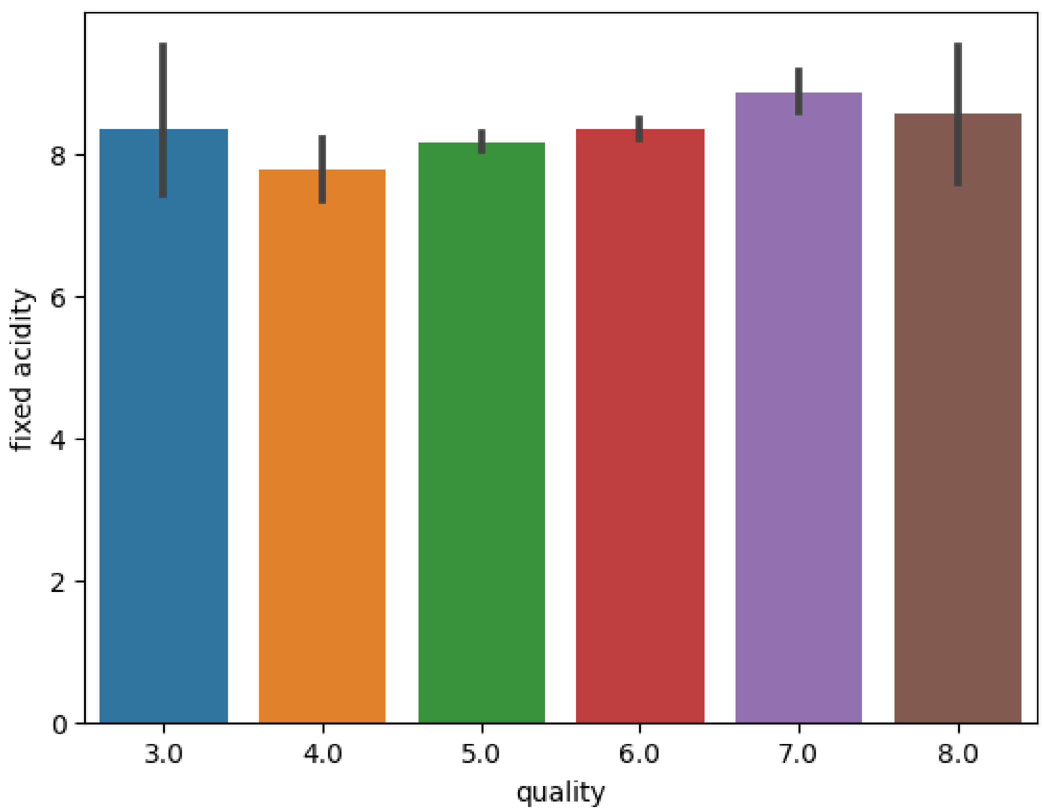
[56]:
```
<Axes: xlabel='quality', ylabel='count'>
```

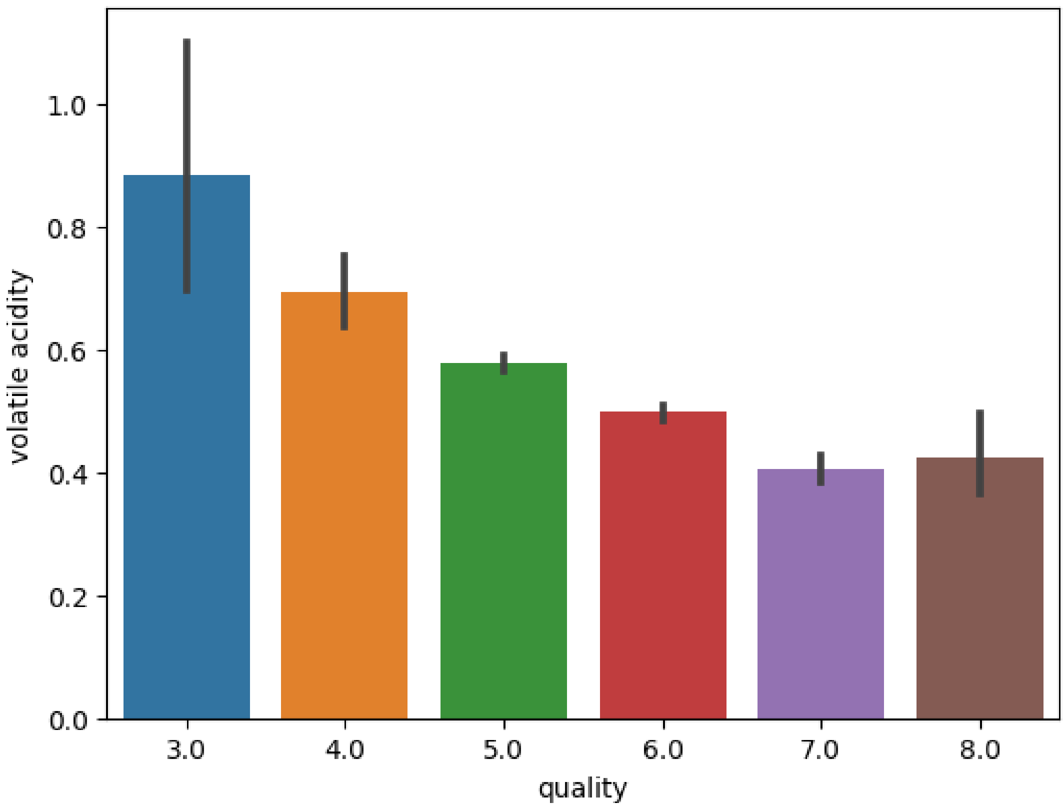


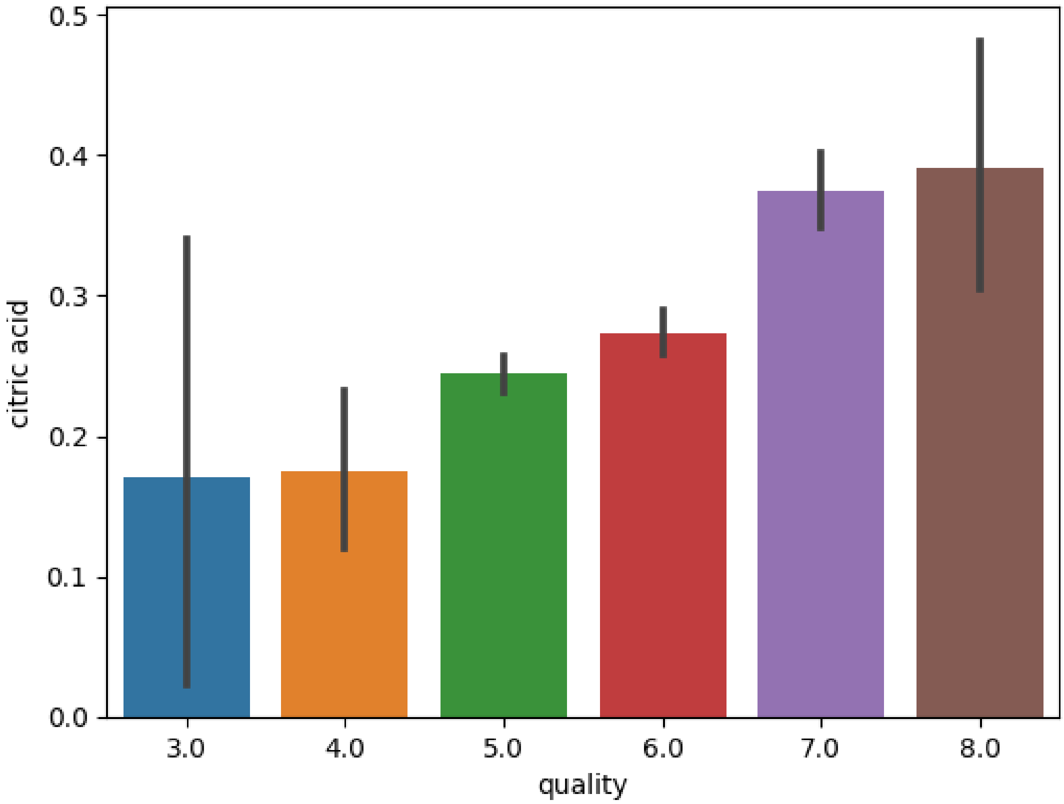

[13]:
```python
df1 = df.select_dtypes([np.int,np.float])

for i, col in enumerate(df1.columns):
```

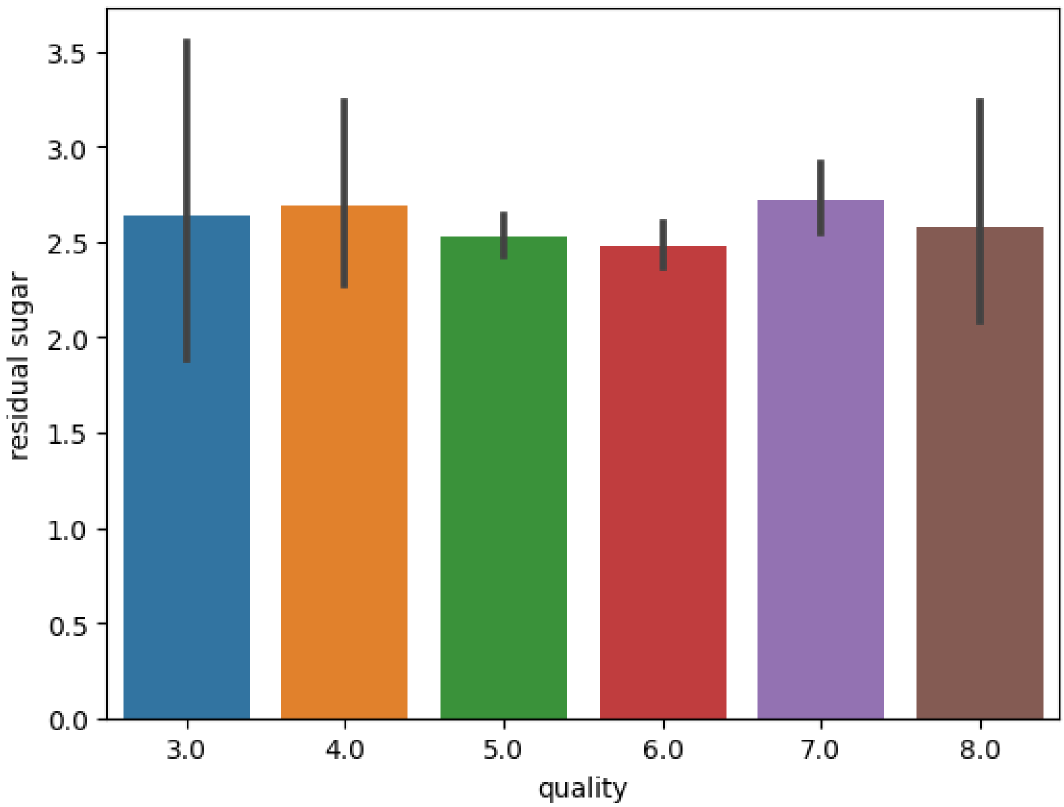
```
    plt.figure(i)
    sb.barplot(x='quality', y =col , data=df1)
```

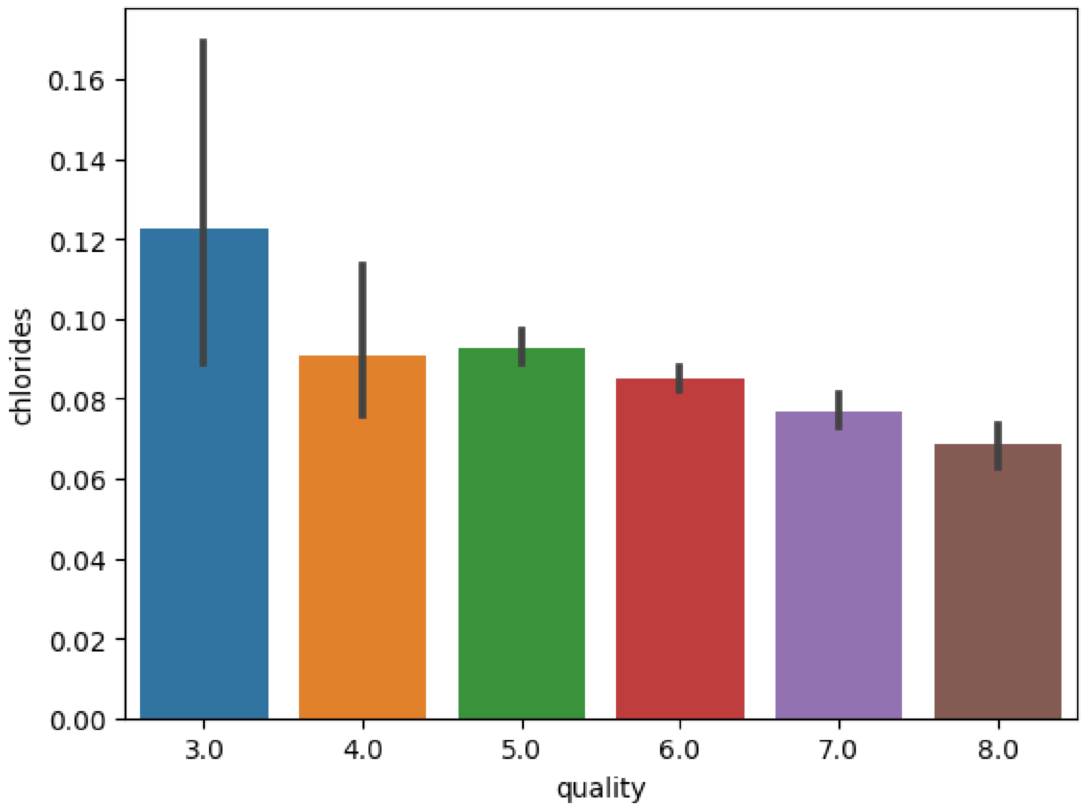
```
<ipython-input-13-2b9e59579017>:1: DeprecationWarning: `np.int` is a deprecated
alias for the builtin `int`. To silence this warning, use `int` by itself. Doing
this will not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish
to review your current use, check the release note link for additional
information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  df1 = df.select_dtypes([np.int,np.float])
<ipython-input-13-2b9e59579017>:1: DeprecationWarning: `np.float` is a
deprecated alias for the builtin `float`. To silence this warning, use `float`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  df1 = df.select_dtypes([np.int,np.float])
```



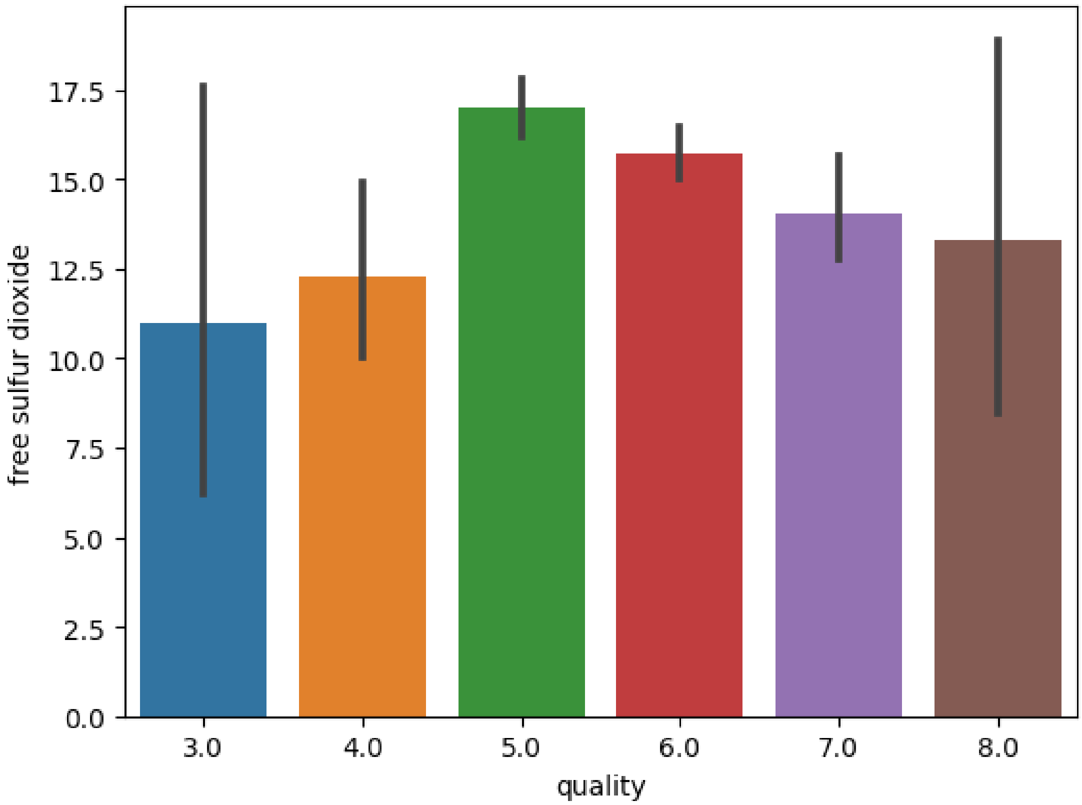

1.0
0.8
0.6
0.4
0.2
0.0
volatile acidity
3.0
4.0
5.0
6.0
7.0
8.0
quality


0.5
0.4
0.3
0.2
0.1
0.0
citric acid
3.0
4.0
5.0
6.0
7.0
8.0
quality

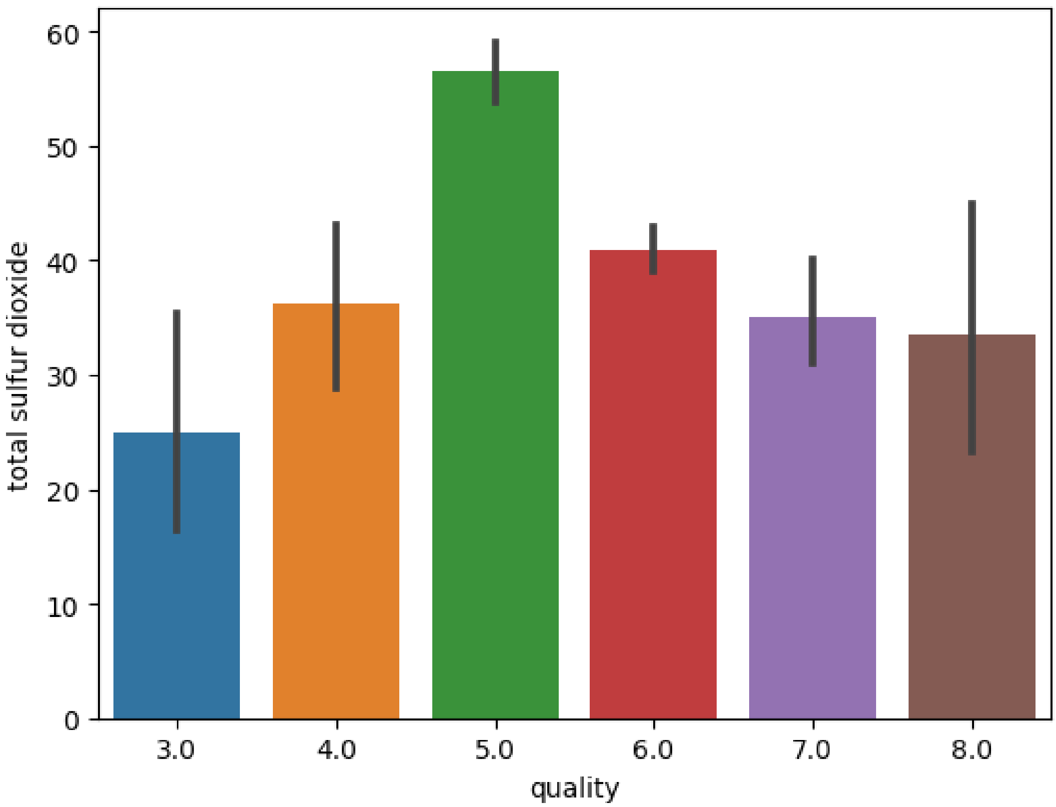

3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
residual sugar
3.0
4.0
5.0
6.0
7.0
8.0
quality

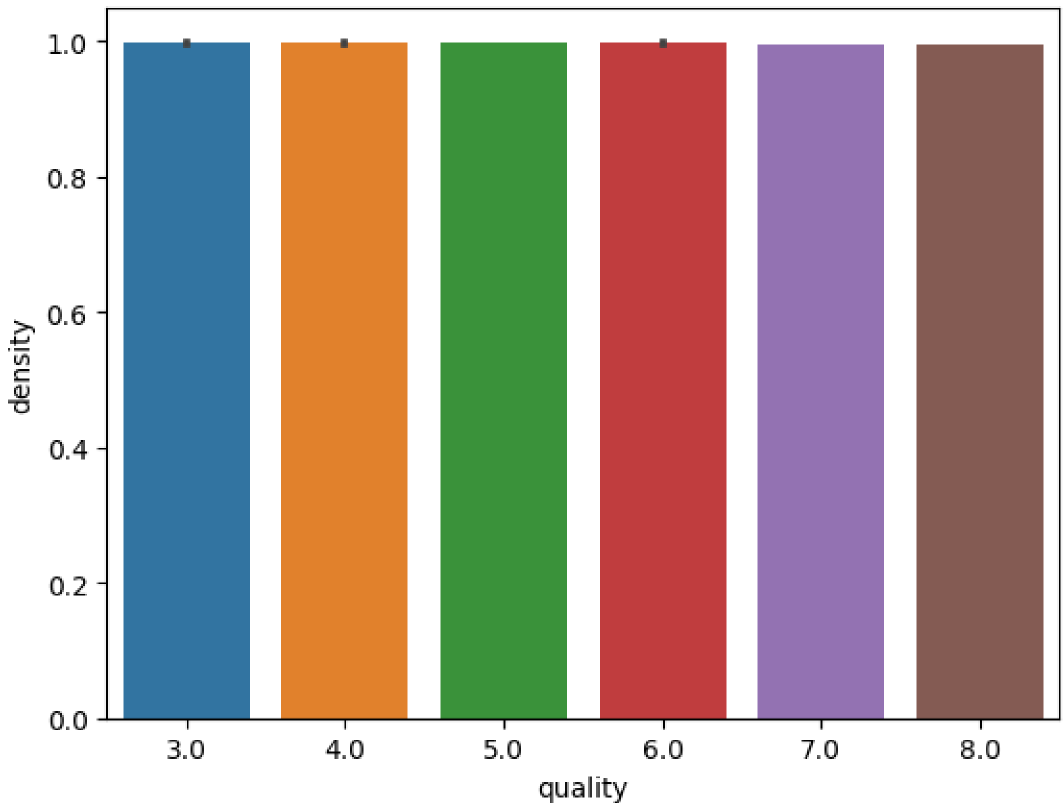


0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
chlorides
3.0
4.0
5.0
6.0
7.0
8.0
quality

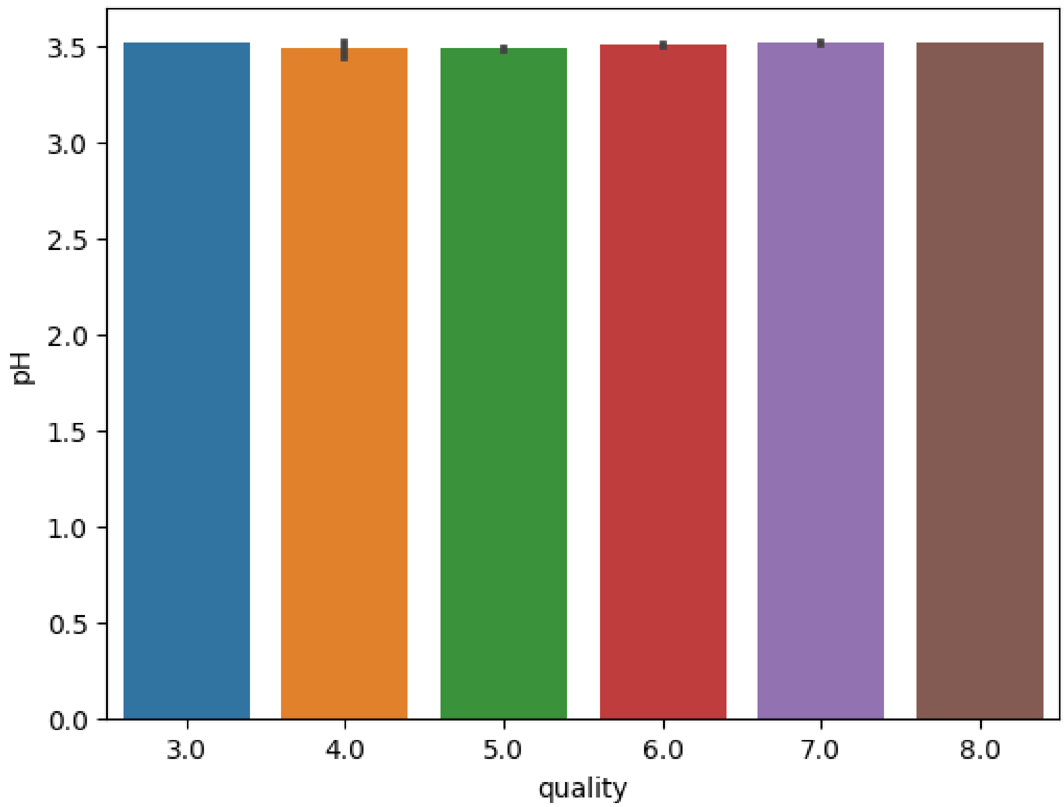


free sulfur dioxide
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
3.0
4.0
5.0
6.0
7.0
8.0
quality

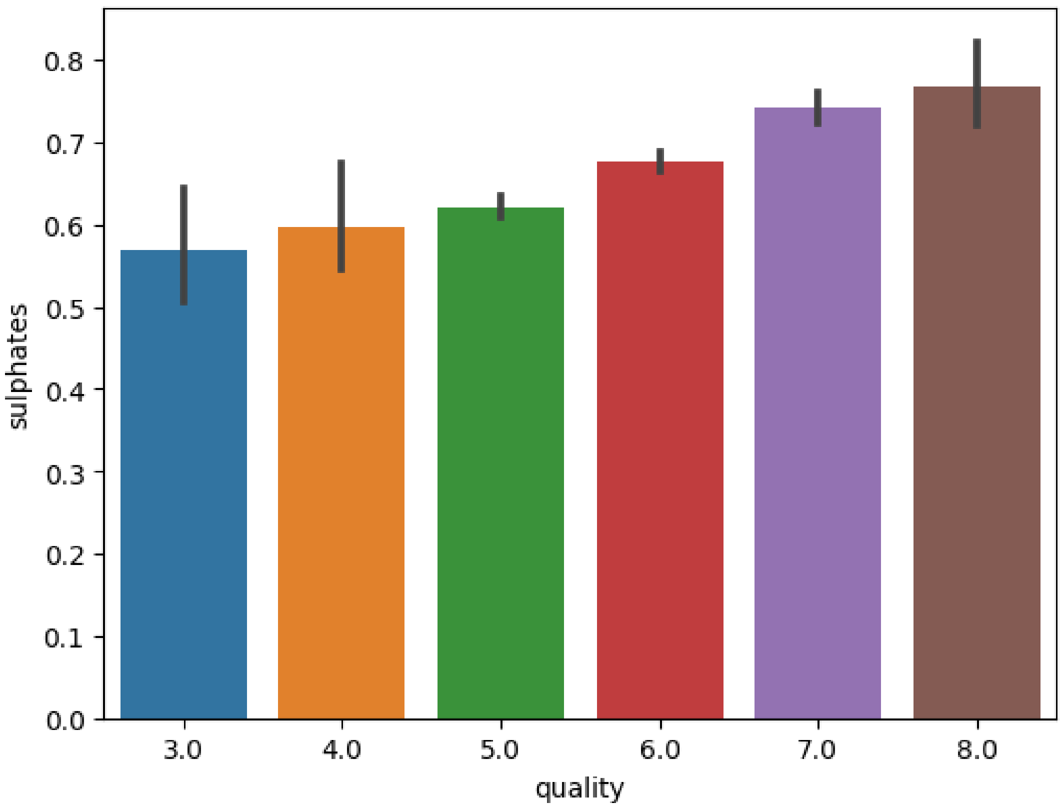

60
50
40
30
20
10
0
total sulfur dioxide
3.0
4.0
5.0
6.0
7.0
8.0
quality

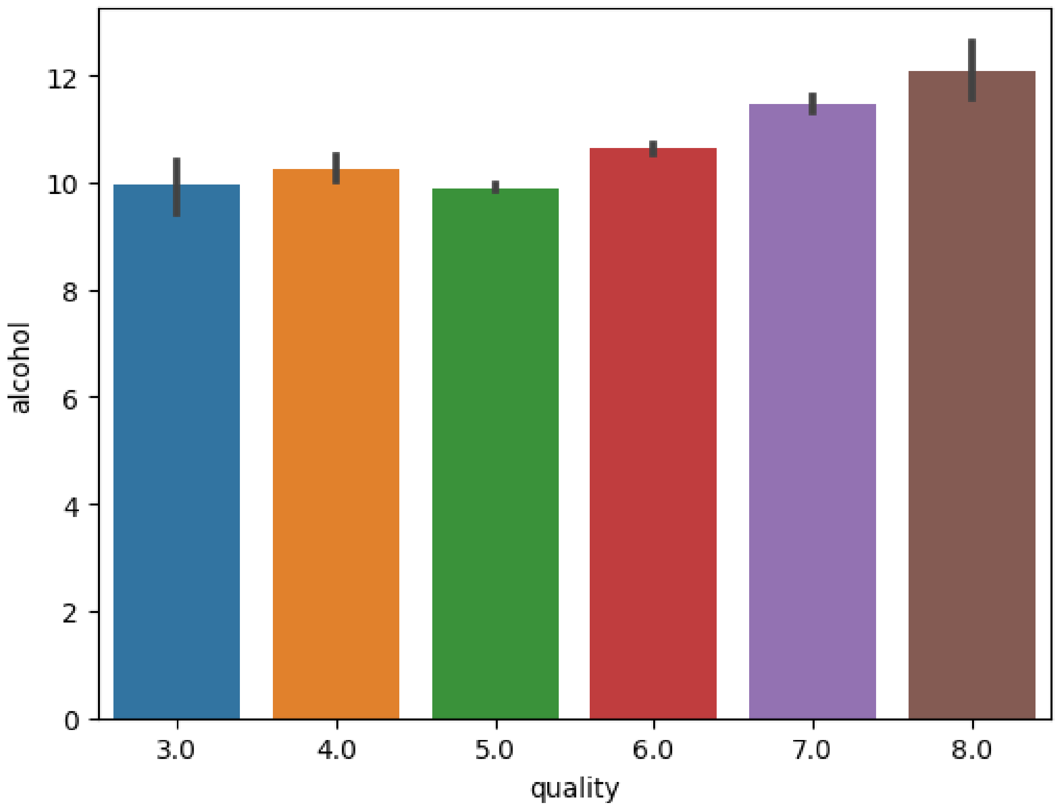


1.0
0.8
0.6
0.4
0.2
0.0
density
3.0
4.0
5.0
6.0
7.0
8.0
quality

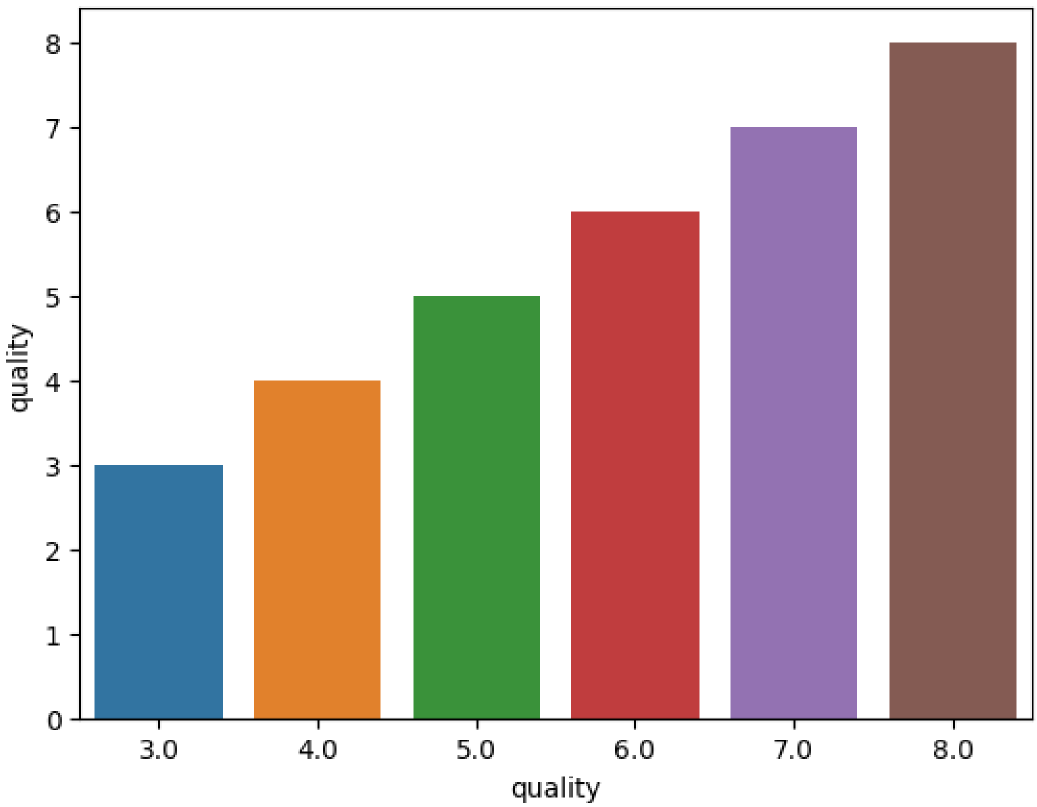


3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
pH
3.0
4.0
5.0
6.0
7.0
8.0
quality


0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
sulphates
3.0
4.0
5.0
6.0
7.0
8.0
quality



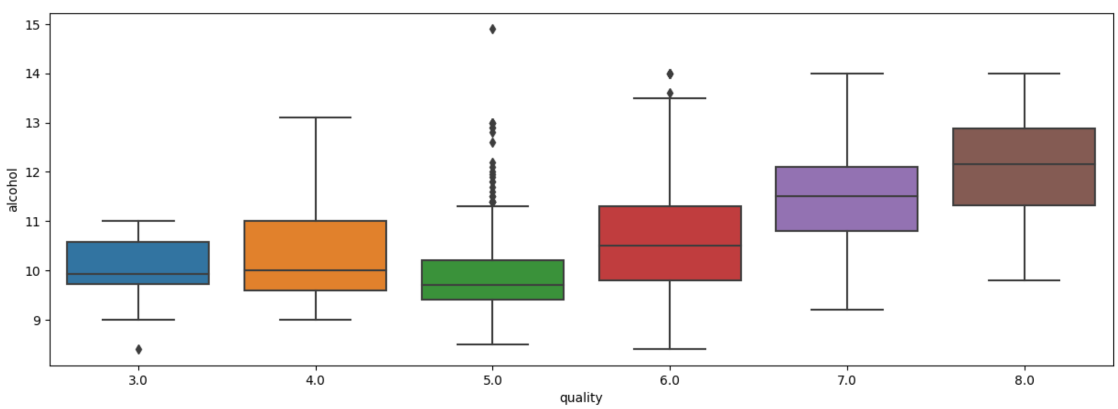
12
10
8
6
4
2
0
alcohol
3.0
4.0
5.0
6.0
7.0
8.0
quality




[15]:
```python
import seaborn as sns
```

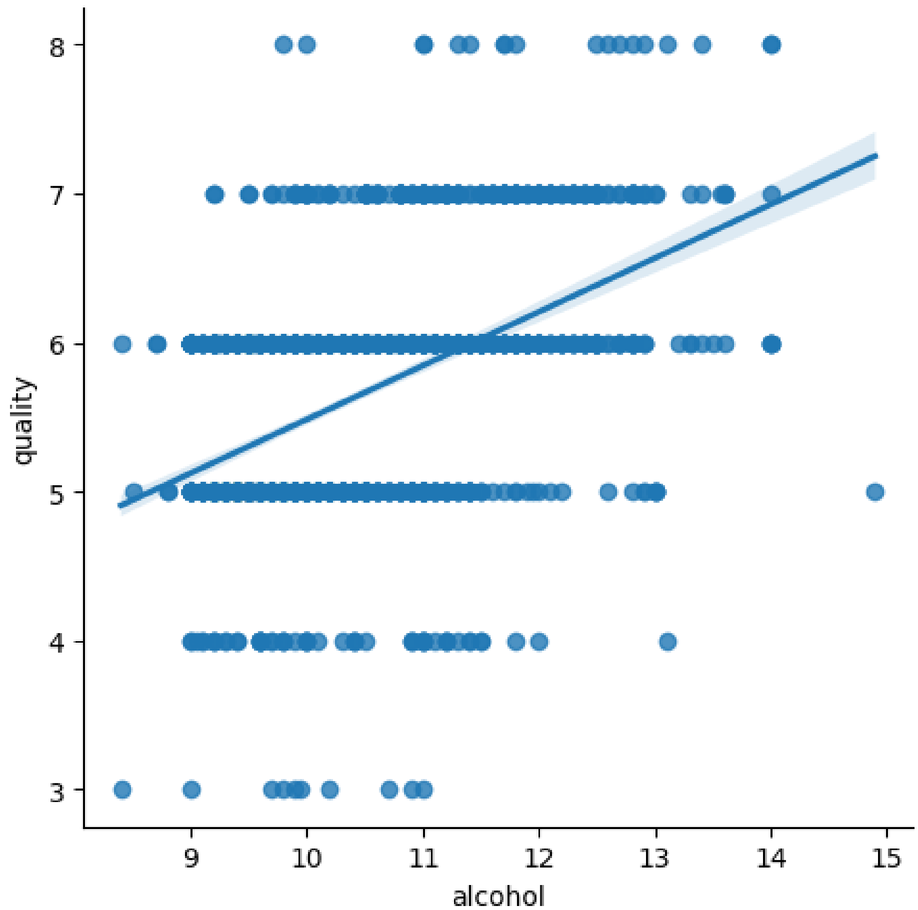
[16]:
```python
plt.figure(figsize=(15,5))
sns.boxplot(x="quality", y="alcohol", data=df)
```

[16]: <Axes: xlabel='quality', ylabel='alcohol'>



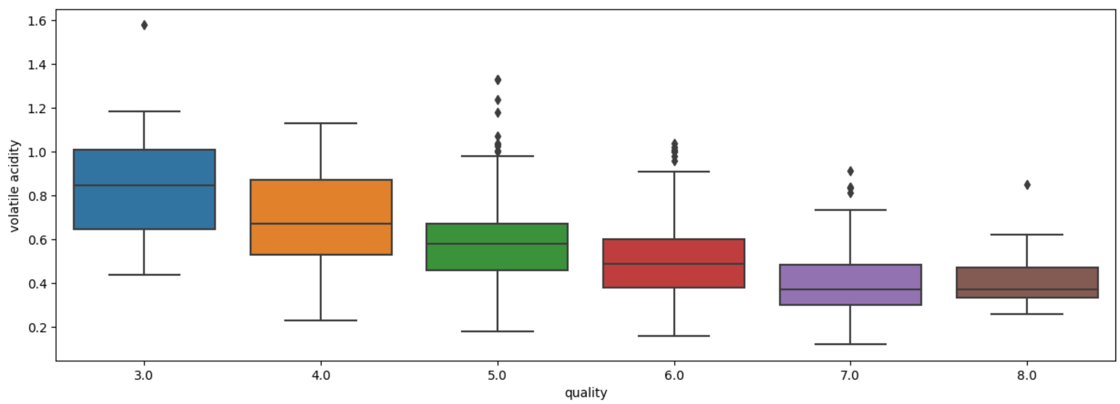
[17]:
```
sns.lmplot(x="alcohol", y="quality", data=df)
```

[17]: <seaborn.axisgrid.FacetGrid at 0x7f62a1c50280>



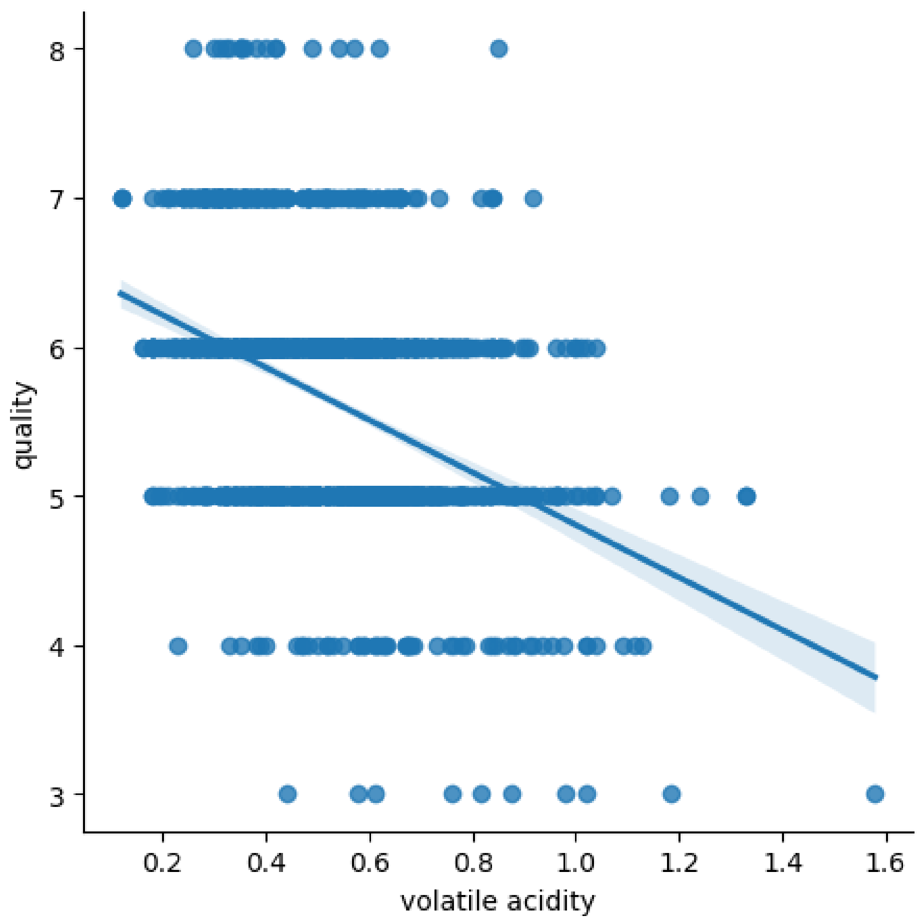

[19]:
```
plt.figure(figsize=(15,5))
sns.boxplot(x="quality", y="volatile acidity", data=df)
```

[19]: <Axes: xlabel='quality', ylabel='volatile acidity'>



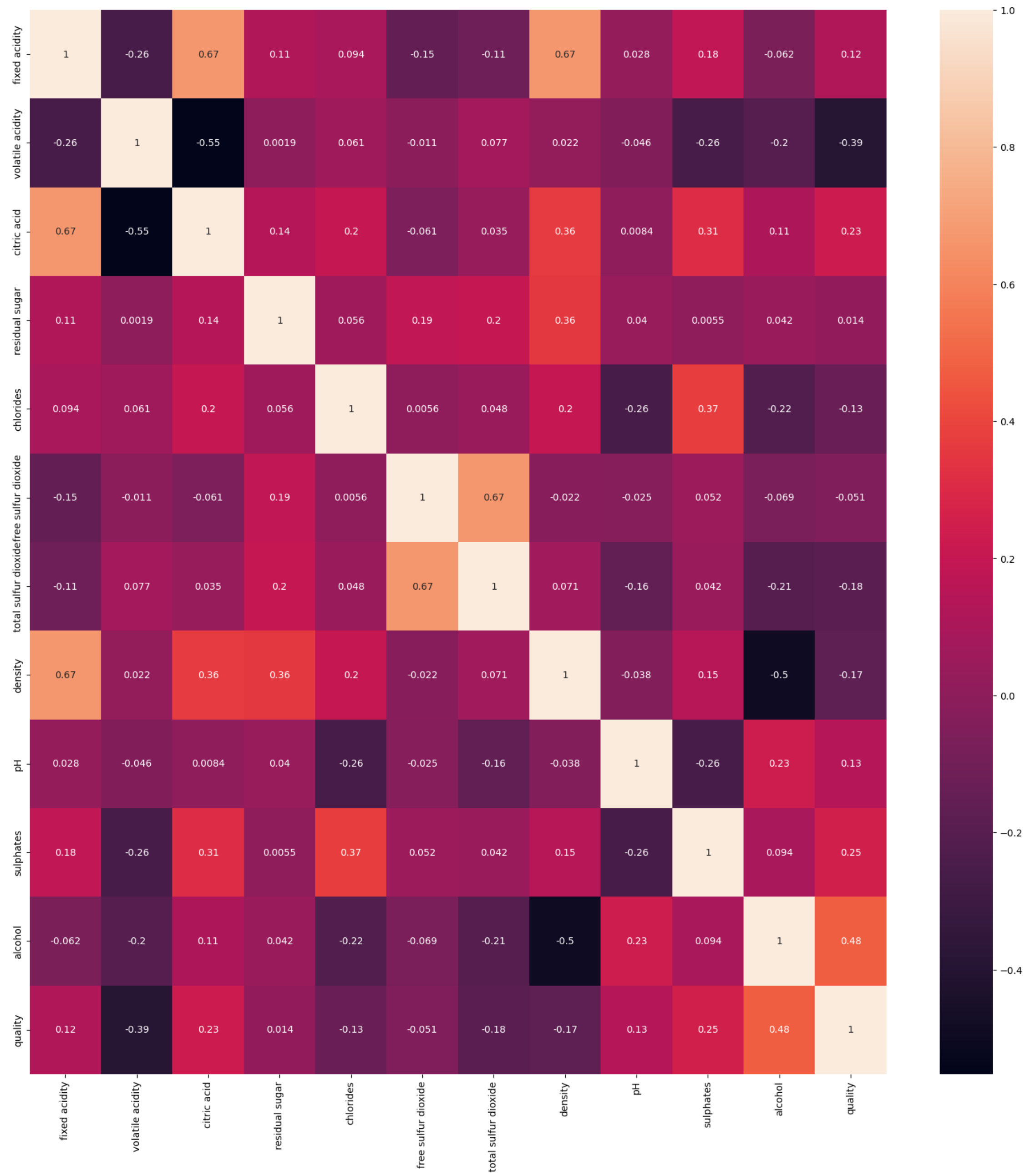

[18]:
```
sns.lmplot(x="volatile acidity", y="quality", data=df)
```

[18]: <seaborn.axisgrid.FacetGrid at 0x7f62a1c538b0>



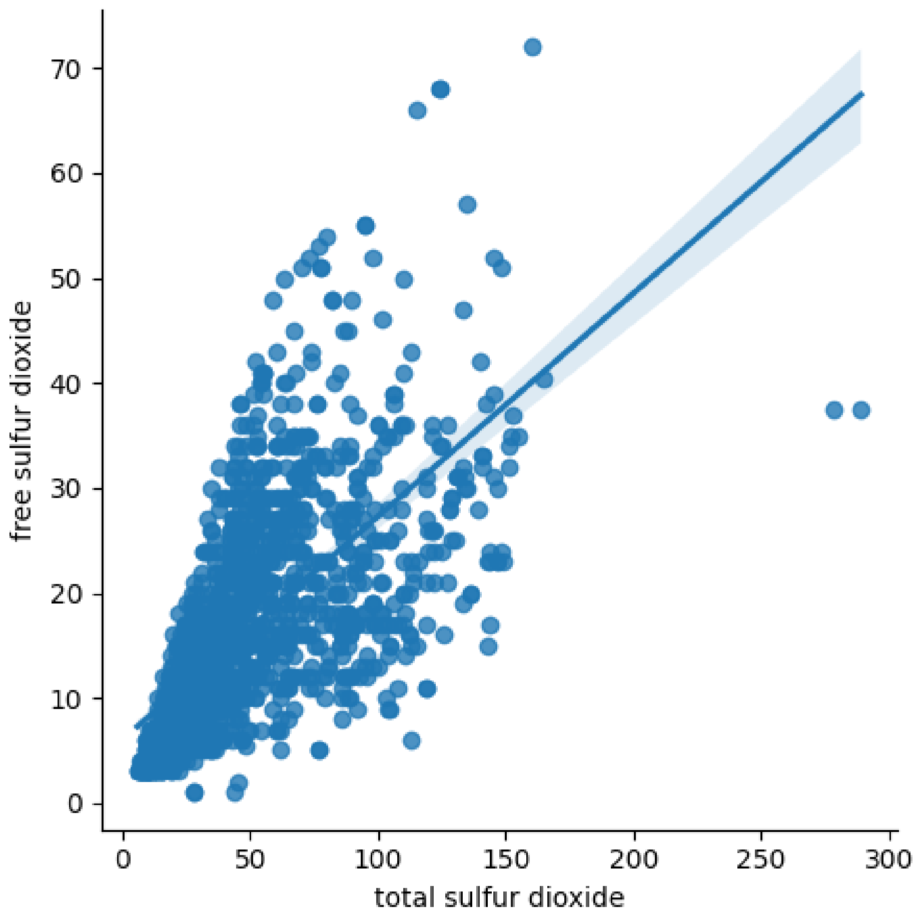
[20]:
```python
plt.figure(figsize=(20,20))
sns.heatmap(df.corr( ),color ="k", annot=True)
```

[20]:
```
<Axes: >
```




[21]:
```python
sns.lmplot(X="total sulfur dioxide", Y="free sulfur dioxide", data=df)
```

[21]:
```
<seaborn.axisgrid.FacetGrid at 0x7f62a230f370>
```




[ ]:
```
bins = (2,6.5,8)
group_names = ['bad','good']
df['quality'] = pd.cut(df['quality'], bins = bins , labels = group_names)
```

[26]:
```
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.svm import SVC,LinearSVC
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

[27]:
```
label_quality=LabelEncoder()
df['quality']=label_quality.fit_transform(df['quality'])
```

[28]:
```
df.head(10)
```

[28]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076
5            7.4              0.66         0.00             1.8      0.075
6            7.9              0.60         0.06             1.6      0.069
7            7.3              0.65         0.00             1.2      0.065
8            7.8              0.58         0.02             2.0      0.073
9            7.5              0.50         0.36             6.1      0.071

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56
5                 13.0                  40.0   0.9978  3.51       0.56
6                 15.0                  59.0   0.9964  3.30       0.46
7                 15.0                  21.0   0.9946  3.39       0.47
8                  9.0                  18.0   0.9968  3.36       0.57
9                 17.0                   NaN   0.9978  3.35       0.80

   alcohol  quality
0      9.4        2
1      9.8        2
2      9.8        2
3      9.8        3
4      9.4        2
5      9.4        2
6      9.4        2
7     10.0        4
8      9.5        4
9     10.5        2
```

[41]:
```
Y=df.quality
X=df.drop('quality',axis=1)
```

[42]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=0)
```

[43]:
```
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

[44]:

```python
def models(X_train,Y_train):

    from sklearn.linear_model import LogisticRegression
    log=LogisticRegression(random_state=0)
    log.fit(X_train,Y_train)

    from sklearn.neighbors import kNeighborsClassifier
    knn=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p=2)
    knn.fit(X_train,Y_train)

    from sklearn.svm import SVC
    svc_lin =SVC(kernel='linear',random_state=0)
    svc_lin.fit(X_train,Y_train)

    from sklearn.svm import SVC
    svc_rbf =SVC(kernel='rbf',random_state=0)
    svc_rbf.fit(X_train,Y_train)

    from sklearn.navie_bayes import GaussianNB
    gauss=GaussianNB()
    gauss.fit(X_train,Y_train)

    from sklearn.tree import DecisionTreeClassifier
    tree=DecisionTreeClassifier(criterion='entropy',random_state=0)
    tree.fit(X_train,Y_train)

    from sklearn.ensemble import RandomForestClassifier
    forest=RandomForestClassifier(n_estimators=10,criterion='entropy',random_state=0)
    forest.fit(X_train,Y_train)

    print('[0]Logistic Regression Training Accuracy:',log.score(X_train,Y_train))
    print('[1]k  Nearest Neighbor Training Accurac:',knn.score(X_train,Y_train))
    print('[2]Support Vector Machine (Linear Classifier) Training Accuracy:',svc_lin.score(X_train,Y_train))
    print('[3]Support Vector Machine (RBF Classifier) Training Accuracy:',svc_rbf.score(X_train,Y_train))
    print('[4]Gaussian Naive Bayes Training Accuracy:',gauss.score(X_train,Y_train))
    print('[5]DecisionTreeClassifier Training Accuracy:',tree.score(X_train,Y_train))
    print('[6]Random forest Classifier Training Accuracy:',forest.score(X_train,Y_train))

    return log,knn,svc_lin,svc_rbf,gauss,tree,forest
```

[ ]:
```
Model=models(X_train,Y_train)
```

[ ]:
```
from sklearn.metrics import confusion_matrix
for i in range(len(models)):
  cm = confusion_matrix(Y_test, models[i].predict(X_test))
  TN , FP , FN , TP = confusion_matrix(Y_test, models[i].predict(X_test)).
 ↪ravel()
  print(cm)
  print('Model[{}] Testing Accuracy ="{} !"'.format(i, (TP + TN) / (TP + TN +␣
 ↪FN + FP)))
  print()
```

[ ]:

[ ]: