In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
#defines a function of two scalar variables and returns a scalar result.
#The class vectorize can be used to “vectorize “this function so that
def addmultiply(a,b):
    if a > b:
        return a * b
    else:
        return a + b
```

In [3]:

```
vec_addm = np.vectorize(addmultiply)

vec_addm([0,4,6,9],[1,6,5,7])
```

Out[3]:

```
array([ 1, 10, 30, 63])
```

In [4]:

```
import scipy.special as sc

#Binary entropy is entropy of a binary discrete random variable with probability p is writt

#implement an entropy function
def binary_entropy(x):
    return -(sc.xlogy(x, x) + sc.xlog1py(1 - x, -x))/np.log(2)

binary_entropy(0.80)
```

Out[4]:

```
0.7219280948873623
```

In [5]:

```
#using quad function find integration of a*x**2+b in the range 0,1

from scipy.integrate import quad

def integrand(x, a, b):
    return a*x**2 + b

a = 2
b = 1
I = quad(integrand, 0, 1, args=(a,b))
I
```

Out[5]:

```
(1.6666666666666667, 1.8503717077085944e-14)
```

In [6]:

```
np.sign(10)
```

Out[6]:

```
1
```

Find the inverse of a matrix

In [7]:

```
import numpy as np
from scipy import linalg ## linear algebra library
A = np.array([[1,3,5],[2,5,1],[2,3,8]])
print("Matrix: \n",A)

inv_mat = linalg.inv(A)

print("Inverse matrix:\n",inv_mat)
```

```
Matrix:
 [[1 3 5]
 [2 5 1]
 [2 3 8]]
Inverse matrix:
 [[-1.48  0.36  0.88]
 [ 0.56  0.08 -0.36]
 [ 0.16 -0.12  0.04]]
```

In [8]:

```
## Double check, dot product of matrix with its inverse matrix is a identity matrix
A.dot(linalg.inv(A))
```

Out[8]:

```
array([[ 1.00000000e+00, -1.11022302e-16, -5.55111512e-17],
       [ 3.05311332e-16,  1.00000000e+00,  1.87350135e-16],
       [ 2.22044605e-16, -1.11022302e-16,  1.00000000e+00]])
```

In [9]:

```
# create an arbitrary 3 variable linear equations system and solve the equations

A = np.array([[1, 2], [3, 4]])

print("Matrix A:/n", A)

b = np.array([[5], [6]])

print("Matrix b:\n", b)
```

```
Matrix A:/n [[1 2]
 [3 4]]
Matrix b:
 [[5]
 [6]]
```

In [10]:

```
linalg.inv(A).dot(b)  #slow
```

Out[10]:

```
array([[-4. ],
       [ 4.5]])
```

In [11]:

```
A.dot(linalg.inv(A).dot(b)) - b  # check
```

Out[11]:

```
array([[0.],
       [0.]])
```

In [12]:

```
np.linalg.solve(A, b)  # easy approch
```

Out[12]:

```
array([[-4. ],
       [ 4.5]])
```

Find the determinant of a 3X3 matrix

In [13]:

```
A = np.array([[1,2,4],[3,4,5],[3,2,1]])

A
```

Out[13]:

```
array([[1, 2, 4],
       [3, 4, 5],
       [3, 2, 1]])
```

In [14]:

```
linalg.det(A)
```

Out[14]:

```
-6.0
```

Compute eigen values and eigen vector of the above matrix A

In [15]:

```
la, v = linalg.eig(A)
```

In [16]:

```
l1, l2 , l3 = la
```

In [17]:

```
print(l1, l2, l3)   # eigenvalues
```

```
(8.222093338069005+0j) (-2.512533689762123+0j) (0.2904403516931155+0j)
```

In [18]:

```
print(v[:, 0])   # first eigenvector
```

```
[0.44582499 0.79769973 0.40609755]
```

In [19]:

```
print(v[:, 1])   # second eigenvector
```

```
[ 0.6703096   0.23231643 -0.70477948]
```

In [20]:

```
print(v[:, 2])   # third eigenvector
```

```
[ 0.46950923 -0.82021311  0.32682035]
```

Find SVD from the above matrix

In [21]:

```
M,N = A.shape ## give the matrix shape
```

In [22]:

```
U,s,Vh = linalg.svd(A) ## svd is applied
```

In [23]:

```
Sig = linalg.diagsvd(s,M,N) ## getting diagolnal matrix
```

In [24]:

```
U, Vh = U, Vh
```

In [25]:

```
U.dot(Sig.dot(Vh)) #check computation
```

Out[25]:

```
array([[1., 2., 4.],
       [3., 4., 5.],
       [3., 2., 1.]])
```

In [26]:

```
## Three decomposed matrix

U
```

Out[26]:

```
array([[-0.49253953, -0.55083721, -0.67378274],
       [-0.79553959, -0.02894716,  0.60520973],
       [-0.35287614,  0.83411056, -0.42395519]])
```

In [27]:

```
Vh
```

Out[27]:

```
array([[-0.4431719 , -0.54842283, -0.70910582],
       [ 0.7630091 ,  0.18444853, -0.61951259],
       [-0.47054837,  0.81560477, -0.3367092 ]])
```

In [28]:

```
Sig
```

Out[28]:

```
array([[8.88546113, 0.        , 0.        ],
       [0.        , 2.44381487, 0.        ],
       [0.        , 0.        , 0.27631405]])
```

Matplotlib

In [29]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
## plt.show() for non-notebook users
```

In [30]:

```
x = np.arange(0,200)
y = x*4
z = x**4
```

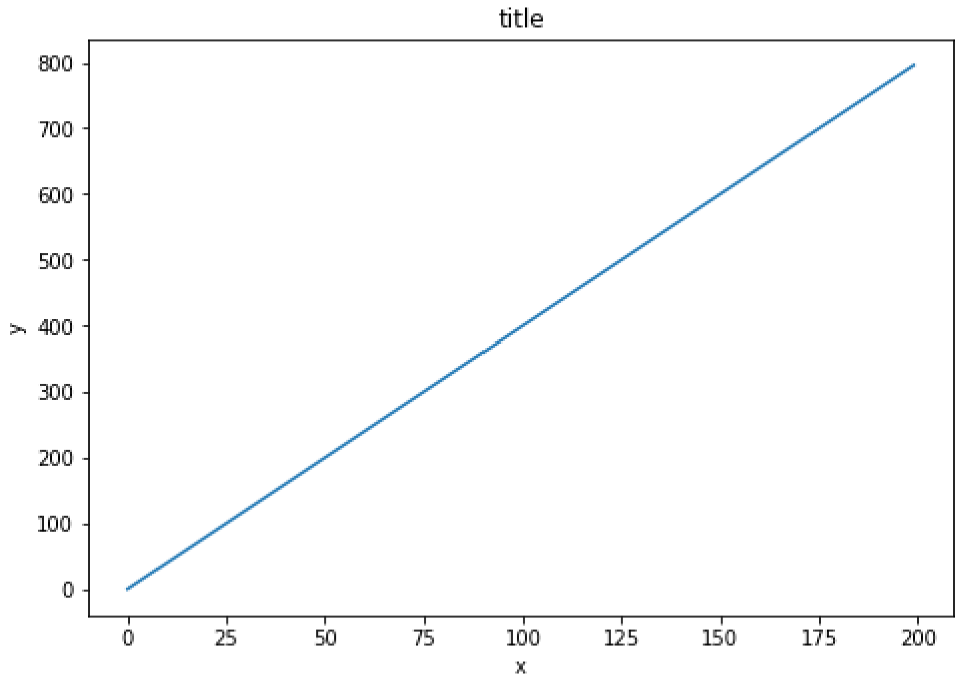
In [31]:

```
fig = plt.figure()
axis = fig.add_axes([0,0,1,1])
axis.plot(x,y)

axis.set_xlabel('x') # x labels
axis.set_ylabel('y') # y labels
axis.set_title('title')
```

Out[31]:

```
Text(0.5,1,'title')
```



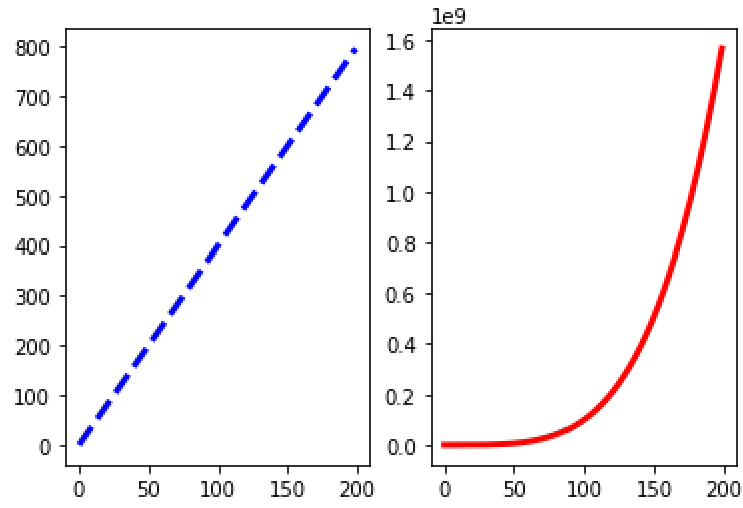
In [32]:

```
# Empty canvas of 1 by 2 subplots
fig, axes = plt.subplots(nrows=1, ncols=2)

axes[0].plot(x,y,color="blue", lw=3, ls='--')
axes[1].plot(x,z,color="red", lw=3, ls='-')
```

Out[32]:

```
[<matplotlib.lines.Line2D at 0x60b36ae48>]
```



In [33]:

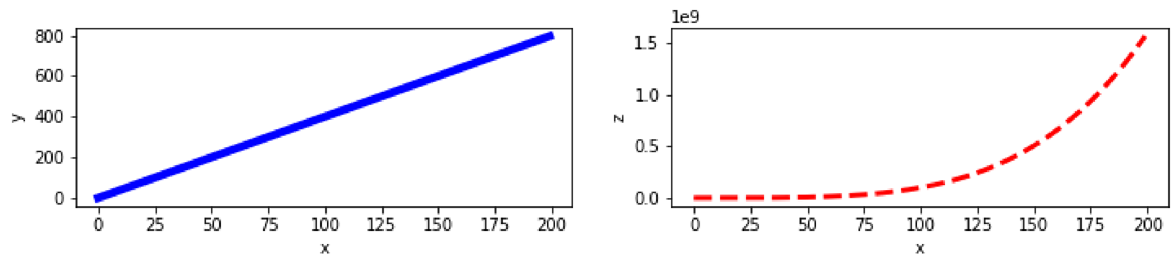
```
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(12,2))

axes[0].plot(x,y,color="blue", lw=5)
axes[0].set_xlabel('x')
axes[0].set_ylabel('y')

axes[1].plot(x,z,color="red", lw=3, ls='--')
axes[1].set_xlabel('x')
axes[1].set_ylabel('z')
```

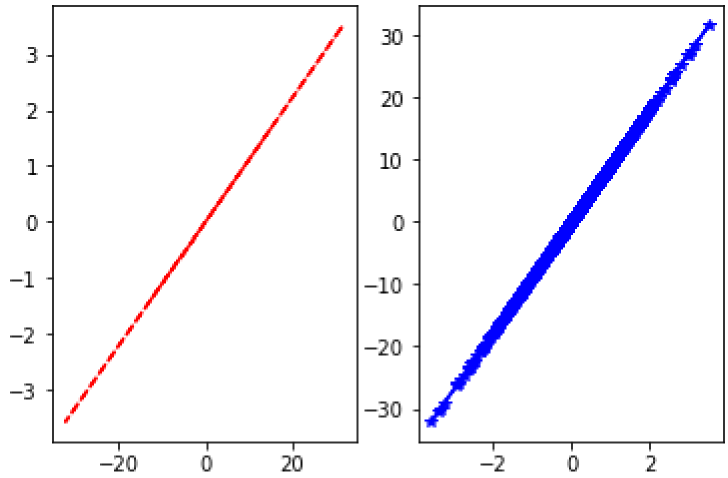
Out[33]:

```
Text(0,0.5,'z')
```



In [34]:

```
A= np.random.normal(0,10,1000)
B= A/9
# plt.subplot(nrows, ncols, plot_number)
plt.subplot(1,2,1)
plt.plot(A, B, 'r--') # More on color options later
plt.subplot(1,2,2)
plt.plot(B, A, 'b*-');
```



In [35]:

```
## Plotting on data
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/wi


df.head()
```

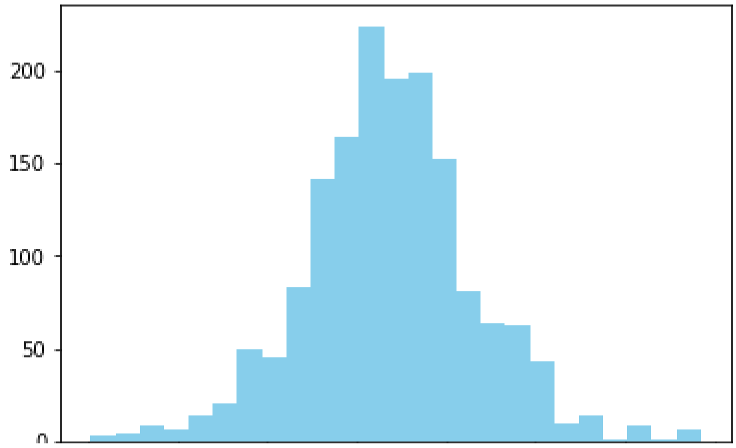
Out[35]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9

In [36]:

```
plt.hist(df.density, bins=25,  color = "skyblue" ) ## try different columns
```

```
 array([0.99007  , 0.9906148, 0.9911596, 0.9917044, 0.9922492, 0.992794 ,
        0.9933388, 0.9938836, 0.9944284, 0.9949732, 0.995518 , 0.9960628,
        0.9966076, 0.9971524, 0.9976972, 0.998242 , 0.9987868, 0.9993316,
        0.9998764, 1.0004212, 1.000966 , 1.0015108, 1.0020556, 1.0026004,
        1.0031452, 1.00369  ]),
 <a list of 25 Patch objects>)
```

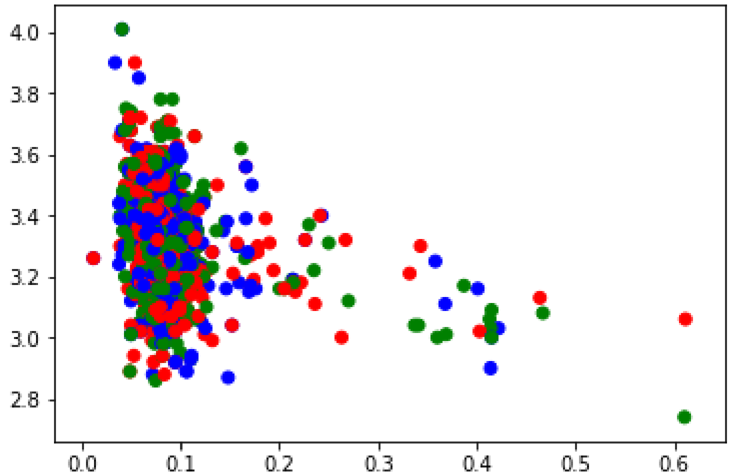


In [37]:

```
colors = ("red", "green", "blue")
plt.scatter(df.chlorides, df.pH, c=colors)
```

Out[37]:

```
<matplotlib.collections.PathCollection at 0x60c82f860>
```

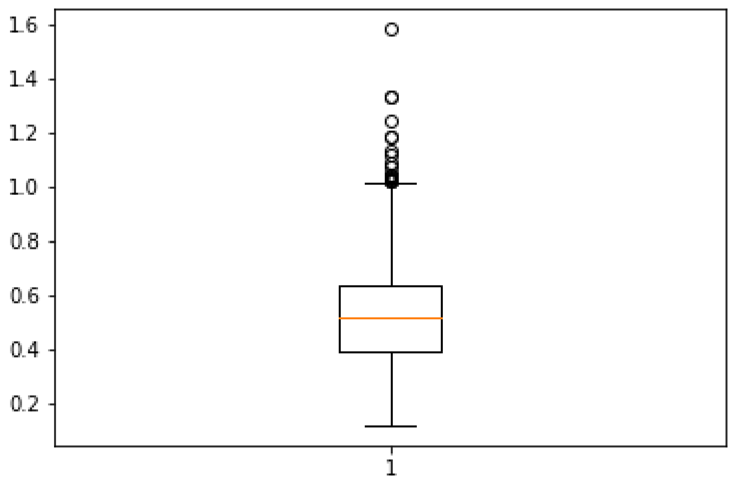


In [38]:

```python
plt.boxplot(df['volatile acidity'])
```

Out[38]:

```
{'boxes': [<matplotlib.lines.Line2D at 0x60c894f60>],
 'caps': [<matplotlib.lines.Line2D at 0x60c89f9b0>,
  <matplotlib.lines.Line2D at 0x60c89fdd8>],
 'fliers': [<matplotlib.lines.Line2D at 0x60c8a5668>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x60c8a5240>],
 'whiskers': [<matplotlib.lines.Line2D at 0x60c89f0f0>,
  <matplotlib.lines.Line2D at 0x60c89f588>]}
```



Seaborn

In [39]:

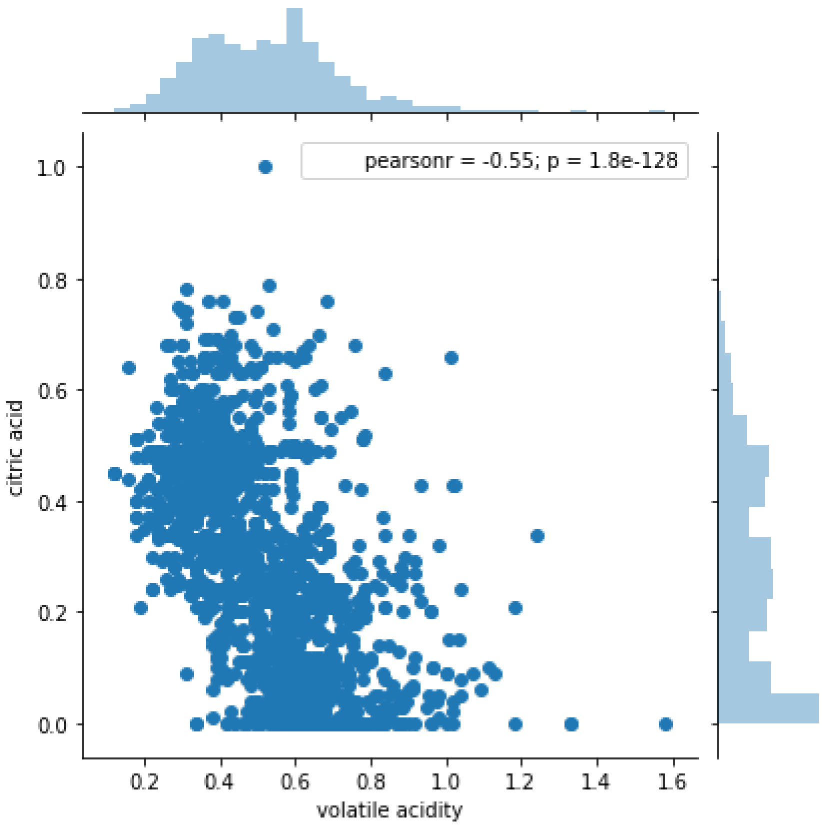
```
import seaborn as sns

## Using the same dataset

sns.jointplot(x='volatile acidity',y='citric acid',data=df)
```

Out[39]:

<seaborn.axisgrid.JointGrid at 0x60c86b518>

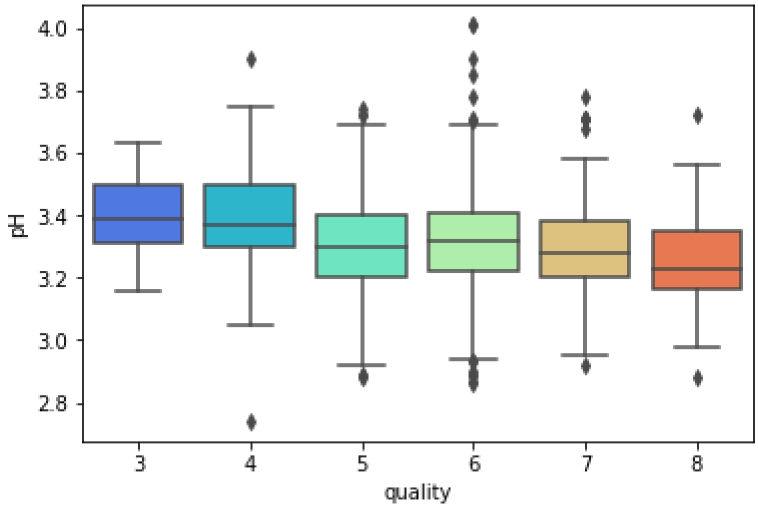


In [40]:

```
sns.boxplot(x='quality',y='pH',data=df,palette='rainbow')
```

Out[40]:

<matplotlib.axes._subplots.AxesSubplot at 0x60c4a5198>

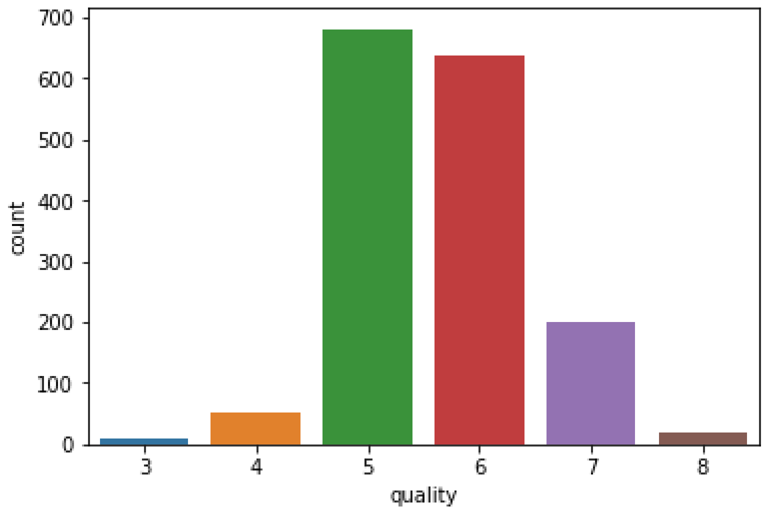


In [41]:

```
sns.countplot(x='quality',data=df)
```

Out[41]:

<matplotlib.axes._subplots.AxesSubplot at 0x60d01b550>



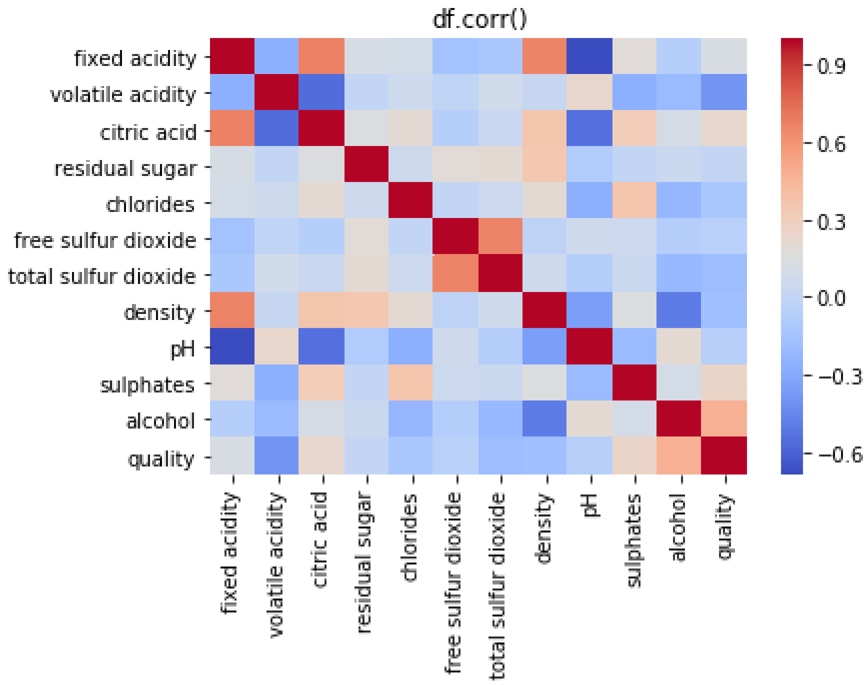
In [42]:

```
## correlation plot

sns.heatmap(df.corr(),cmap='coolwarm')
plt.title('df.corr()')
```

Out[42]:

```
Text(0.5,1,'df.corr()')
```



In [ ]: