Wine Quality Data

I decided to use the same dataset on wine quality that was used in K-Means Clustering lab. The data set contains various chemical properties of wine, such as acidity, sugar, pH, and alcohol. It also contains a quality metric (3-9, with highest being better) and a color (red or white). The name of the file is `Wine_Quality_Data.csv` .

The dataset contains the chemical properties (i.e. everything but quality and color) that was used to cluster the wine.

In [1]:
```
import numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as sns, os
os.chdir('data')
# make sure to create a directory called data and put the csv and colorsetup.py files in
there
from colorsetup import colors, palette
sns.set_palette(palette)
%matplotlib inline
```

Perform Exploratory Data Analysis

In [2]:
```
### BEGIN SOLUTION
# Import the data
data = pd.read_csv('Wine_Quality_Data.csv')

data.head(4).T
```

Out[2]:

	0	1	2	3
fixed_acidity	7.4	7.8	7.8	11.2
volatile_acidity	0.7	0.88	0.76	0.28
citric_acid	0	0	0.04	0.56
residual_sugar	1.9	2.6	2.3	1.9
chlorides	0.076	0.098	0.092	0.075
free_sulfur_dioxide	11	25	15	17
total_sulfur_dioxide	34	67	54	60
density	0.9978	0.9968	0.997	0.998
pH	3.51	3.2	3.26	3.16
sulphates	0.56	0.68	0.65	0.58
alcohol	9.4	9.8	9.8	9.8
quality	5	5	5	6
color	red	red	red	red

In [4]:
```
#number of rows and columns
data.shape
# there are 6497 rows and 13 columns
```

Out[4]:
```
(6497, 13)
```

Data Type of each variable/column

The implementation of K-means in Scikit-learn is designed only to work with continuous data (even though it is sometimes used with categorical or boolean types). All the columns except for quality and color are of float data type.

In [6]:
```
data.dtypes
# only quality and color are not float data type columns
```

Out[6]:
```
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
color                    object
dtype: object
```

Total number of records by wine color:

In [12]:
```
data.color.value_counts()
# there are only 2 types of wines, red and white in this dataset
```

Out[12]:
```
white    4898
red      1599
Name: color, dtype: int64
```

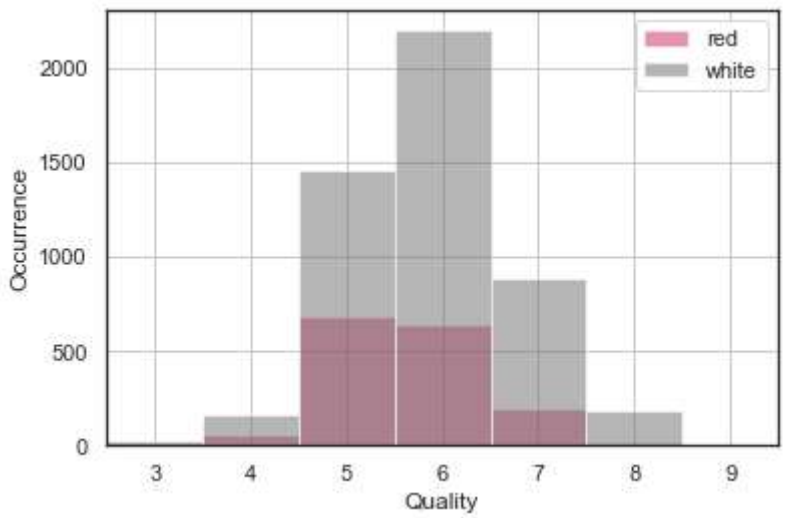
Number of records by quality ratings:

In [13]:
```
data.quality.value_counts().sort_index()
# as you can see, there are very few wines that have quality rating of either very low o
r very high
```

Out[13]:
```
3      30
4     216
5    2138
6    2836
7    1079
8     193
9       5
Name: quality, dtype: int64
```

Create histogram to show distribution of quality rating for red and white wines

In [11]:
```
# seaborn styles
sns.set_context('notebook')
sns.set_style('white')

# custom colors
red = sns.color_palette()[2]
white = sns.color_palette()[4]

# set bins for histogram
bin_range = np.array([3, 4, 5, 6, 7, 8, 9])

# plot histogram of quality counts for red and white wines
ax = plt.axes()
for color, plot_color in zip(['red', 'white'], [red, white]):
    q_data = data.loc[data.color==color, 'quality']
    q_data.hist(bins=bin_range,
                alpha=0.5, ax=ax,
                color=plot_color, label=color)


ax.legend()
ax.set(xlabel='Quality', ylabel='Occurrence')

# force tick labels to be in middle of region
ax.set_xlim(3,10)
ax.set_xticks(bin_range+0.5)
ax.set_xticklabels(bin_range);
ax.grid('off')
### END SOLUTION
```



Perform Feature Engineering

- Create correlation matrix for dependent variables

In [14]:
```
### BEGIN SOLUTION
float_columns = [x for x in data.columns if x not in ['color', 'quality']]

# The correlation matrix
corr_mat = data[float_columns].corr()

# Strip out the diagonal values for the next step
for x in range(len(float_columns)):
    corr_mat.iloc[x,x] = 0.0

corr_mat
```

Out[14]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	t
fixed_acidity	0.000000	0.219008	0.324436	-0.111981	0.298195	-0.282735	
volatile_acidity	0.219008	0.000000	-0.377981	-0.196011	0.377124	-0.352557	
citric_acid	0.324436	-0.377981	0.000000	0.142451	0.038998	0.133126	
residual_sugar	-0.111981	-0.196011	0.142451	0.000000	-0.128940	0.402871	
chlorides	0.298195	0.377124	0.038998	-0.128940	0.000000	-0.195045	
free_sulfur_dioxide	-0.282735	-0.352557	0.133126	0.402871	-0.195045	0.000000	
total_sulfur_dioxide	-0.329054	-0.414476	0.195242	0.495482	-0.279630	0.720934	
density	0.458910	0.271296	0.096154	0.552517	0.362615	0.025717	
pH	-0.252700	0.261454	-0.329808	-0.267320	0.044708	-0.145854	
sulphates	0.299568	0.225984	0.056197	-0.185927	0.395593	-0.188457	
alcohol	-0.095452	-0.037640	-0.010493	-0.359415	-0.256916	-0.179838	

In [15]:
```
# Pairwise maximal correlations
corr_mat.abs().idxmax()
```

Out[15]:
```
fixed_acidity                density
volatile_acidity    total_sulfur_dioxide
citric_acid             volatile_acidity
residual_sugar                  density
chlorides                     sulphates
free_sulfur_dioxide  total_sulfur_dioxide
total_sulfur_dioxide  free_sulfur_dioxide
density                         alcohol
pH                          citric_acid
sulphates                     chlorides
alcohol                         density
dtype: object
```

Finding highly skewed columns so log transformation could be applied to those columns

In [16]:
```
skew_columns = (data[float_columns]
                .skew()
                .sort_values(ascending=False))

skew_columns = skew_columns.loc[skew_columns > 0.75]
skew_columns
```

Out[16]:
```
chlorides              5.399828
sulphates              1.797270
fixed_acidity          1.723290
volatile_acidity       1.495097
residual_sugar         1.435404
free_sulfur_dioxide    1.220066
dtype: float64
```

In [17]:
```
# Perform log transform on skewed columns
for col in skew_columns.index.tolist():
    data[col] = np.log1p(data[col])
```

Perform feature scaling:

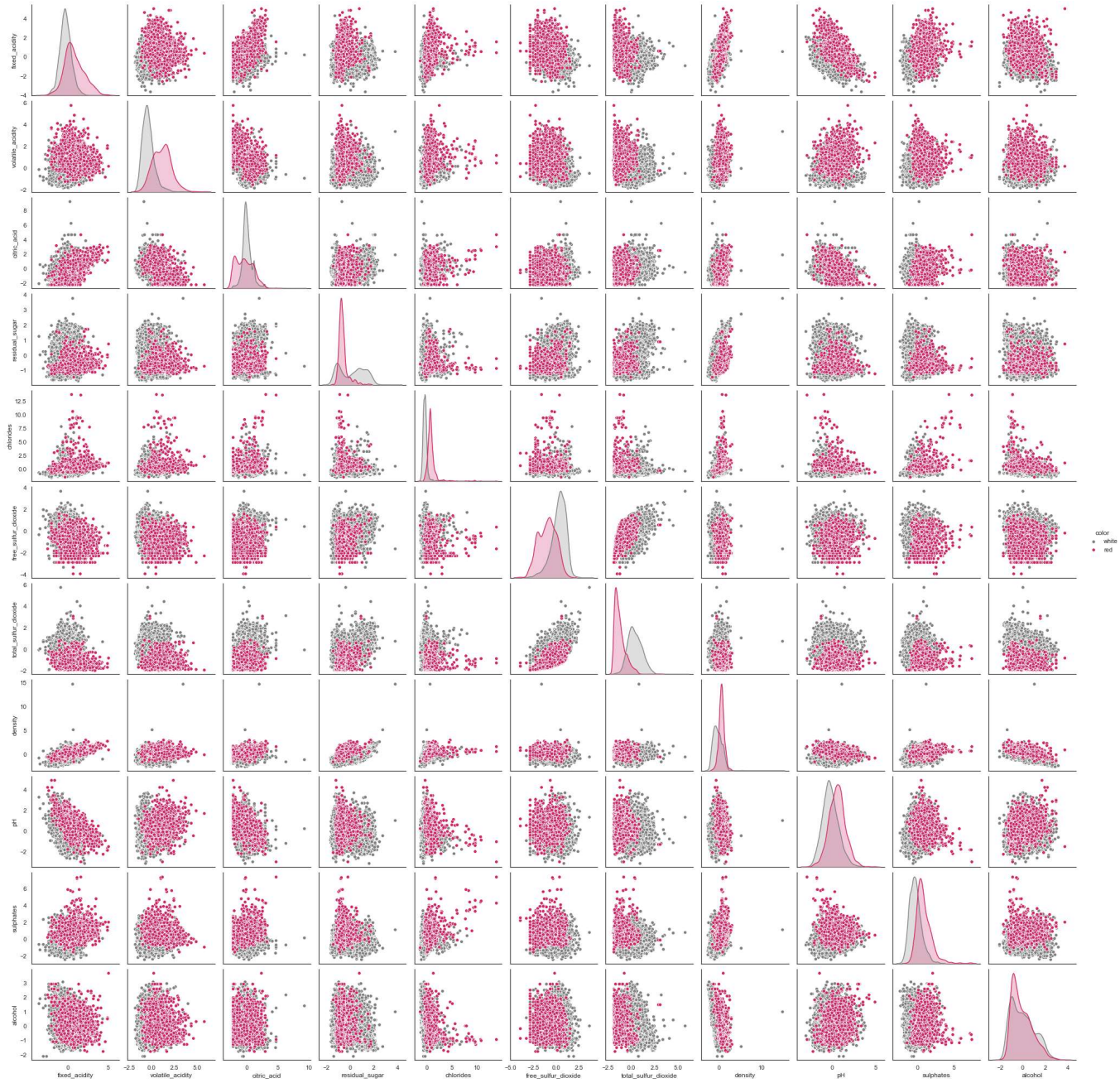
In [18]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
data[float_columns] = sc.fit_transform(data[float_columns])

data.head(4)
```

Out[18]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide
0	0.229509	2.135767	-2.192833	-0.815173	0.624554	-1.193601	-1.446359
1	0.550261	3.012817	-2.192833	-0.498175	1.281999	-0.013944	-0.862469
2	0.550261	2.438032	-1.917553	-0.625740	1.104012	-0.754684	-1.092486
3	2.802728	-0.337109	1.661085	-0.815173	0.594352	-0.574982	-0.986324

Finally, the pairplot of the transformed and scaled features.

In [19]:
```
sns.set_context('notebook')
sns.pairplot(data[float_columns + ['color']],
             hue='color',
             hue_order=['white', 'red'],
             palette={'red':red, 'white':'gray'});
### END SOLUTION
```



Summary of training

K-Means Clustering

Fit a K-means clustering model with two clusters.

In [23]:
```
from sklearn.cluster import KMeans
### BEGIN SOLUTION
km = KMeans(n_clusters=2, random_state=42)
km = km.fit(data[float_columns])

data['kmeans'] = km.predict(data[float_columns])
```

Examine the clusters by counting the number of red and white wines in each cluster

In [24]:
```
(data[['color','kmeans']]
 .groupby(['kmeans','color'])
 .size()
 .to_frame()
 .rename(columns={0:'number'}))
### END SOLUTION
```

Out[24]:

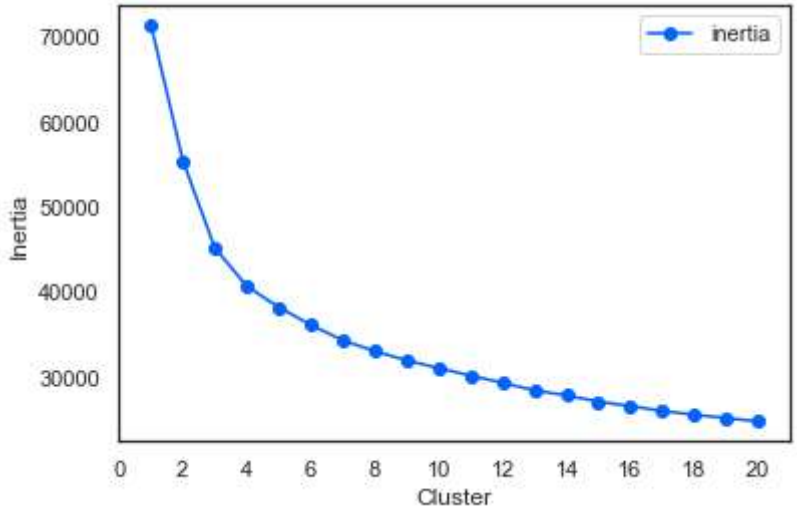
		number
kmeans	**color**	
0	**red**	23
	white	4810
1	**red**	1576
	white	88

Fit K-Means models with cluster values ranging from 1 to 20.

In [27]:
```
### BEGIN SOLUTION
# Create and fit a range of models
km_list = list()

for clust in range(1,21):
    km = KMeans(n_clusters=clust, random_state=42)
    km = km.fit(data[float_columns])

    km_list.append(pd.Series({'clusters': clust,
                              'inertia': km.inertia_,
                              'model': km}))
```

Plot cluster number vs inertia

In [28]:
```
plot_data = (pd.concat(km_list, axis=1)
             .T
             [['clusters','inertia']]
             .set_index('clusters'))

ax = plot_data.plot(marker='o',ls='-')
ax.set_xticks(range(0,21,2))
ax.set_xlim(0,21)
ax.set(xlabel='Cluster', ylabel='Inertia');
### END SOLUTION
```



Agglomerative clustering model

Fit an agglomerative clustering model with two clusters.

In [29]:
```
from sklearn.cluster import AgglomerativeClustering
### BEGIN SOLUTION
ag = AgglomerativeClustering(n_clusters=2, linkage='ward', compute_full_tree=True)
ag = ag.fit(data[float_columns])
data['agglom'] = ag.fit_predict(data[float_columns])
```

Examine the clusters by counting the number of red and white wines in agglomerative cluster

In [34]:
```
# First, for Agglomerative Clustering:
(data[['color','agglom','kmeans']]
 .groupby(['color','agglom'])
 .size()
 .to_frame()
 .rename(columns={0:'number'}))
```

Out[34]:

		number
color	**agglom**	
red	**0**	31
	1	1568
white	**0**	4755
	1	143

Examine the clusters by counting the number of red and white wines in K-Means cluster

In [ ]:
```
# Comparing with KMeans results:
(data[['color','agglom','kmeans']]
 .groupby(['color','kmeans'])
 .size()
 .to_frame()
 .rename(columns={0:'number'}))
```

Comparing the results of agglomerative and K-Means clusters

In [38]:
```
# Comparing results:
(data[['color','agglom','kmeans']]
 .groupby(['color','agglom','kmeans'])
 .size()
 .to_frame()
 .rename(columns={0:'number'}))
```

Out[38]:

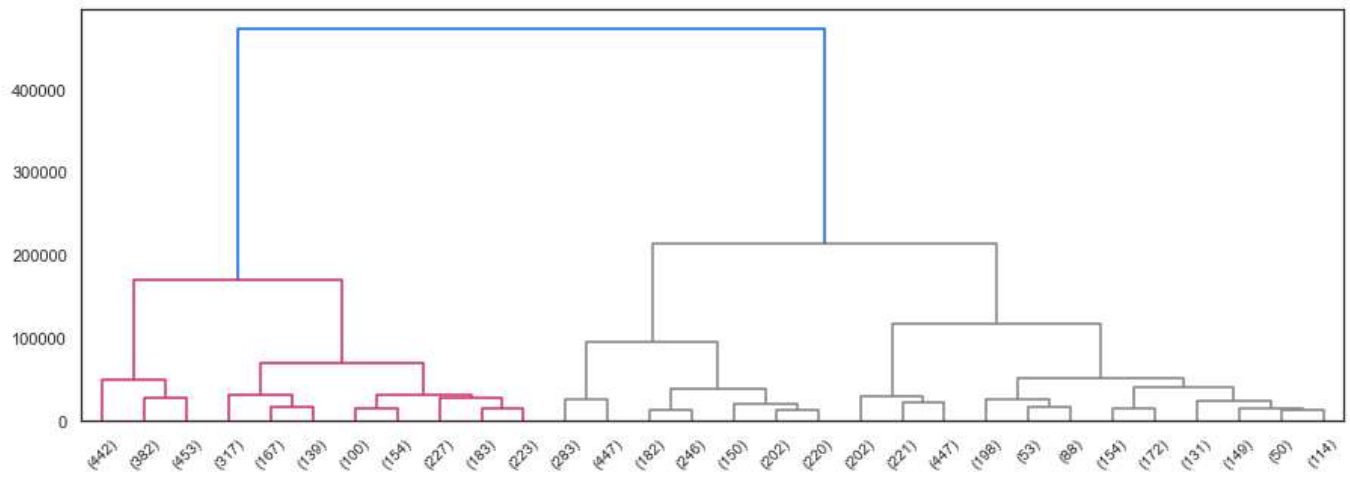
			number
color	**agglom**	**kmeans**	
red	**0**	**0**	18
		1	13
	1	**0**	5
		1	1563
white	**0**	**0**	4717
		1	38
	1	**0**	93
		1	50

Key Insight

Though the cluster numbers are not identical, the clusters are very consistent within a single wine variety (red or white).

And here is a plot of the dendrogram created from agglomerative clustering.

In [39]:
```
# First, we import the cluster hierarchy module from SciPy (described above) to obtain t
he linkage and dendrogram functions.
from scipy.cluster import hierarchy

Z = hierarchy.linkage(ag.children_, method='ward')

fig, ax = plt.subplots(figsize=(15,5))

# Some color setup
red = colors[2]
blue = colors[0]

hierarchy.set_link_color_palette([red, 'gray'])

den = hierarchy.dendrogram(Z, orientation='top',
                           p=30, truncate_mode='lastp',
                           show_leaf_counts=True, ax=ax,
                           above_threshold_color=blue)
### END SOLUTION
```



Next Steps

The next step would be to cover all the topics discussed in the course and create a one python notebook to provide end to end solution for classifying activities using unsupervised learning algorithms. The dataset used for this exercise was small. Another next step is to find a dataset that can be used to fit models using different algorithms such as Mean Shift, Ward, and DBSCAN.