Clustering Methods Exercises

Learning Objectives

- Discuss unsupervised learning algorithms and how they can be applied
- Apply clustering
- Apply dimensionality reduction
- Apply Intel® Extension for Scikit-learn* to leverage underlying compute capabilities of hardware

scikit-learn*

Frameworks provide structure that Data Scientists use to build code. Frameworks are more than just libraries, because in addition to callable code, frameworks influence how code is written.

A main virtue of using an optimized framework is that code runs faster. Code that runs faster is just generally more convenient but when we begin looking at applied data science and AI models, we can see more material benefits. Here you will see how optimization, particularly hyperparameter optimization can benefit more than just speed.

These exercises will demonstrate how to apply **the Intel® Extension for Scikit-learn*,** a seamless way to speed up your Scikit-learn application. The acceleration is achieved through the use of the Intel® oneAPI Data Analytics Library (oneDAL). Patching is the term used to extend scikit-learn with Intel optimizations and makes it a well-suited machine learning framework for dealing with real-life problems.

To get optimized versions of many Scikit-learn algorithms using a patch() approach consisting of adding these lines of code after importing sklearn:

- **from sklearnex import patch_sklearn**
- **patch_sklearn()**

This exercise relies on installation of Intel® Extension for Scikit-learn*

If you have not already done so, follow the instructions from Week 1 for instructions

Introduction

We will be using the wine quality data set for these exercises. This 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`.

We will be using the chemical properties (i.e. everything but quality and color) to cluster the wine. Though this is unsupervised learning, it can be fun to see how our clustering results map onto color and quality.

```
from __future__ import print_function
import os
data_path = [  'data']

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.cluster import AgglomerativeClustering
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, roc_auc_score
from sklearn.model_selection import StratifiedShuffleSplit

from sklearnex import patch_sklearn
patch_sklearn()
```

```
Intel(R) Extension for Scikit-learn* enabled (https://github.com/intel/scikit-learn-intelex)
```

Question 1

- Import the data and examine the features.
- Note which are continuous, categorical, and boolean.
- How many entries are there for the two colors and range of qualities?
- Make a histogram plot of the quality for each of the wine colors.

```
import pandas as pd

# Import the data
filepath = os.sep.join(data_path + ['Wine_Quality_Data.csv'])
data = pd.read_csv(filepath)

data.head(4).T
```

	0	1	2	3
fixed_acidity	7.4000	7.8000	7.800	11.200
volatile_acidity	0.7000	0.8800	0.760	0.280
citric_acid	0.0000	0.0000	0.040	0.560
residual_sugar	1.9000	2.6000	2.300	1.900
chlorides	0.0760	0.0980	0.092	0.075
free_sulfur_dioxide	11.0000	25.0000	15.000	17.000
total_sulfur_dioxide	34.0000	67.0000	54.000	60.000
density	0.9978	0.9968	0.997	0.998
pH	3.5100	3.2000	3.260	3.160
sulphates	0.5600	0.6800	0.650	0.580
alcohol	9.4000	9.8000	9.800	9.800
quality	5	5	5	6
color	red	red	red	red

```
data.shape
```

```
(6497, 13)
```

The data types for each entry. 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). Fortunately, all the columns we will be using (everything except quality and color) are continuous.

```
data.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
color                    object
dtype: object
```

The number of entries for each wine color.

```
data.color.value_counts()
```

```
white    4898
red      1599
Name: color, dtype: int64
```

The distribution of quality values.

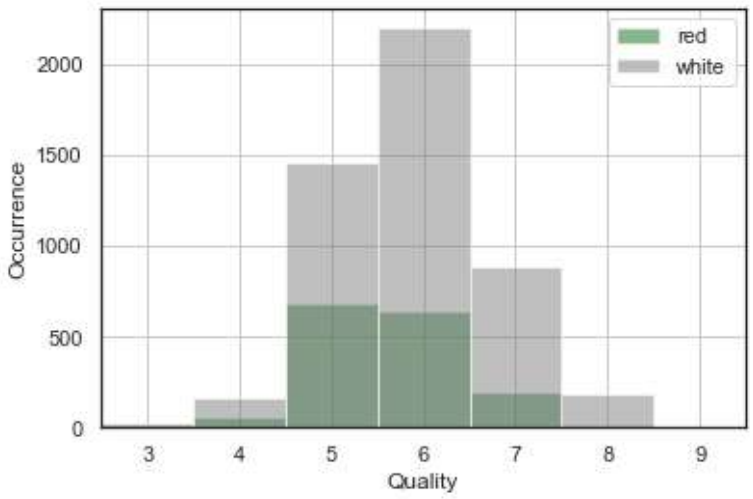
```
data.quality.value_counts().sort_index()
```

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

Now for the histogram.

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

%matplotlib inline
```

```
sns.set_context('notebook')
sns.set_style('white')
sns.set_palette('dark')

red = sns.color_palette()[2]
white = 'gray'

bin_range = np.array([3, 4, 5, 6, 7, 8, 9])

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')

ax.set_xlim(3,10)
ax.set_xticks(bin_range+0.5)
ax.set_xticklabels(bin_range);
ax.grid('off')
```



Question 2

- Example the correlation and skew of the relevant variables--everything except color and quality.
- Perform any appropriate feature transformations and/or scaling.
- Examine the pairwise distribution of the variables with pairplots to verify scaling and normalization efforts.

```
float_columns = [x for x in data.columns if x not in ['color', 'quality']]

corr_mat = data[float_columns].corr()

for x in range(len(float_columns)):
    corr_mat.iloc[x,x] = 0.0

corr_mat
```

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

```
corr_mat.abs().idxmax()
```

```
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
```

And an examination of the skew values in anticipation of transformations.

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

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

```
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
```

```
for col in skew_columns.index.tolist():
    data[col] = np.log1p(data[col])
```

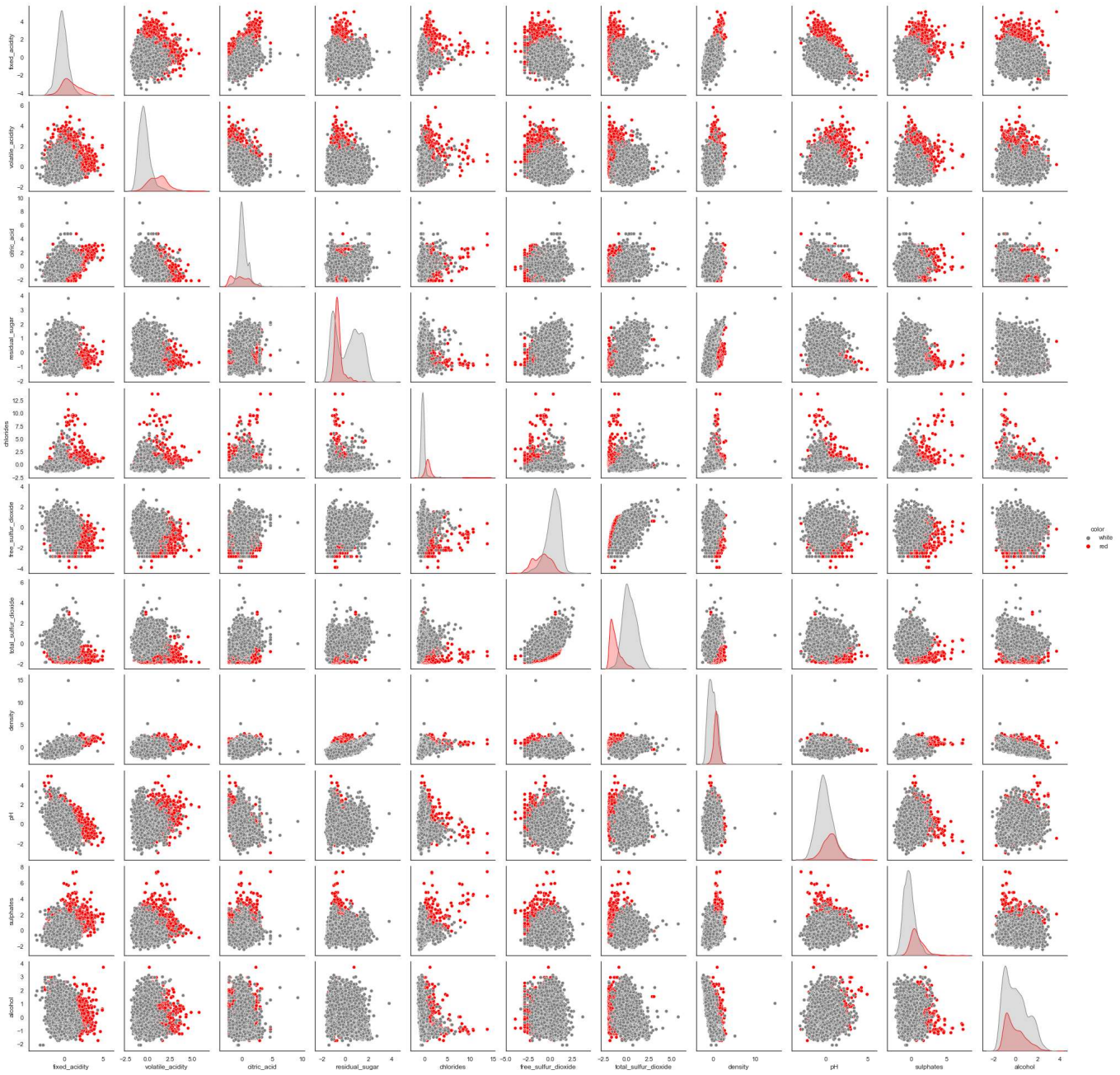
Perform feature scaling.

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

data.head(4)
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_diox
0	0.229509	2.135767	-2.192833	-0.815173	0.624554	-1.193
1	0.550261	3.012817	-2.192833	-0.498175	1.281999	-0.013
2	0.550261	2.438032	-1.917553	-0.625740	1.104012	-0.754
3	2.802728	-0.337109	1.661085	-0.815173	0.594352	-0.574

Finally, the pairplot of the transformed and scaled features.

```
temp = data[float_columns + ['color']]

sns.set_context('notebook')
sns.pairplot(temp,
             hue='color',
             hue_order=['white', 'red'],
             palette={'red':'red', 'white':'gray'});
```


fixed_acidity
volatile_acidity
citric_acid
residual_sugar
chlorides
free_sulfur_dioxide
total_sulfur_dioxide
density
pH
sulphates
alcohol
color
white
red

Question 3

- Fit a K-means clustering model with two clusters.
- Examine the clusters by wine color.

```
km = KMeans(n_clusters=2, random_state=42)
km = km.fit(data[float_columns])

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

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

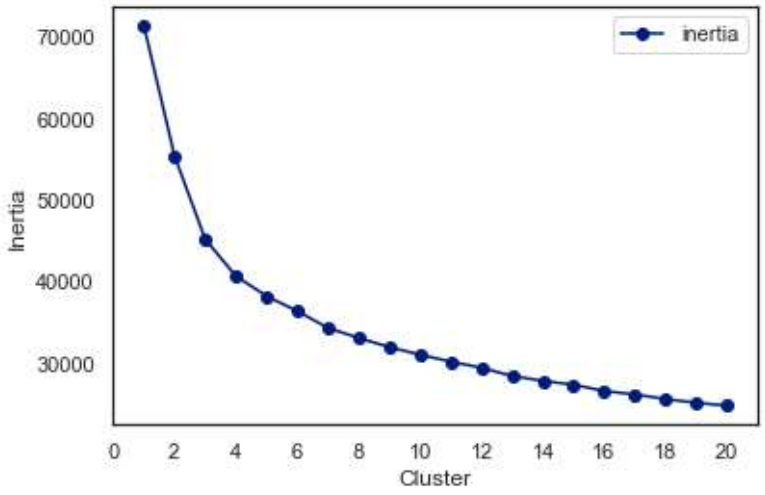
		number
color	**kmeans**	
red	**0**	1576
	1	23
white	**0**	87
	1	4811

Question 4

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

- For each model, store the number of clusters and the inertia value.
- Plot cluster number vs inertia. Does there appear to be an ideal cluster number?

```
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_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');
```



Question 5

- Fit an agglomerative clustering model with two clusters.
- Compare the results to those obtained by K-means with regards to wine color.
- Visualize the dendrogram produced by agglomerative clustering. *Hint:* SciPy has a module called `cluster.hierarchy` that contains the `linkage` and `dendrogram` functions required to create the linkage map and plot the resulting dendrogram.

```
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])
```

Note that cluster assignment is arbitrary, the respective primary cluster numbers for red and white may not be identical to the ones below and also may not be the same for both K-means and agglomerative clustering.

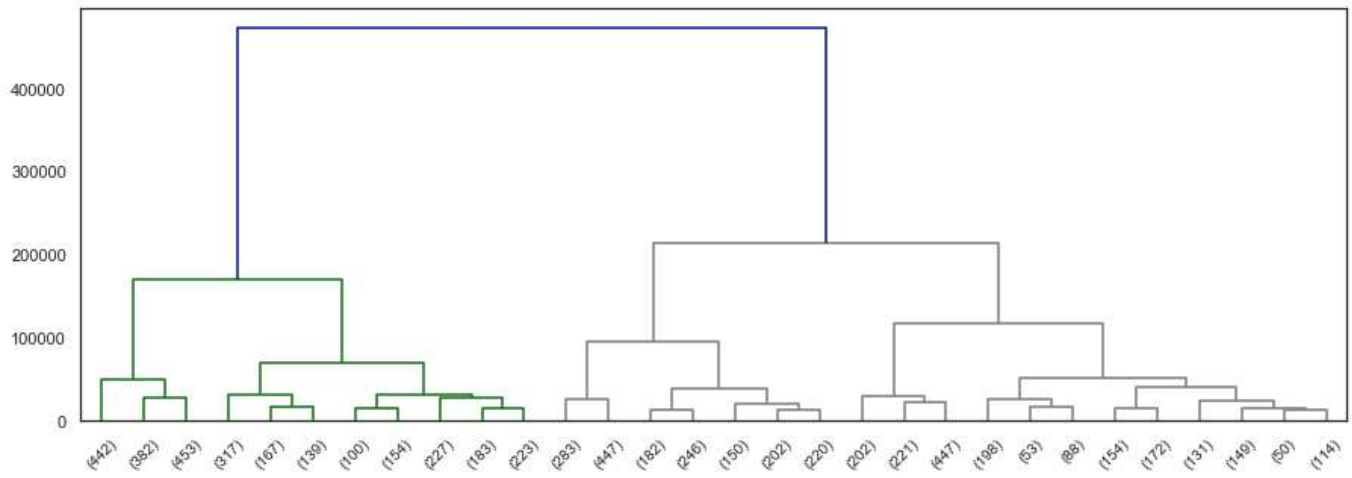
```
(data[['color','agglom','kmeans']]
 .groupby(['color','agglom','kmeans'])
 .size()
 .to_frame()
 .rename(columns={0:'number'}))
```

			number
color	**agglom**	**kmeans**	
red	**0**	**0**	13
		1	18
	1	**0**	1563
		1	5
white	**0**	**0**	38
		1	4717
	1	**0**	49
		1	94

Though the cluster numbers are not necessarily 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.

```
from scipy.cluster import hierarchy
from matplotlib import colors

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

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

dark_palette = sns.color_palette()
red = colors.to_hex(dark_palette[2])
blue = colors.to_hex(dark_palette[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)
```



Question 6

In this question, we are going to explore clustering as a form of feature engineering.

- Create a **binary** target variable `y`, denoting if the quality is greater than 7 or not.
- Create a variable called `X_with_kmeans` from `data`, by dropping the columns "quality", "color" and "agglom" from the dataset. Create `X_without_kmeans` from that by dropping "kmeans".
- For both datasets, using `StratifiedShuffleSplit` with 10 splits, fit 10 Random Forest Classifiers and average out the roc-auc scores.
- Compare the average roc-auc scores for the models using the kmeans clusters as a feature and the one that doesn't use it.

```
y = (data['quality'] > 7).astype(int)
X_with_kmeans = data.drop(['agglom', 'color', 'quality'], axis=1)
X_without_kmeans = X_with_kmeans.drop('kmeans', axis=1)
sss = StratifiedShuffleSplit(n_splits=10, random_state=6532)


def get_avg_roc_10splits(estimator, X, y):
    roc_auc_list = []
    for train_index, test_index in sss.split(X, y):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        estimator.fit(X_train, y_train)
        y_predicted = estimator.predict(X_test)
        y_scored = estimator.predict_proba(X_test)[:, 1]
        roc_auc_list.append(roc_auc_score(y_test, y_scored))
    return np.mean(roc_auc_list)

estimator = RandomForestClassifier()
roc_with_kmeans = get_avg_roc_10splits(estimator, X_with_kmeans, y)
roc_without_kmeans = get_avg_roc_10splits(estimator, X_without_kmeans, y)
print("Without kmeans cluster as input to Random Forest, roc-auc is \"{0}\"".format(roc_without_
print("Using kmeans cluster as input to Random Forest, roc-auc is \"{0}\"".format(roc_with_kmean
```

```
Without kmeans cluster as input to Random Forest, roc-auc is "0.9014603174603175"
Using kmeans cluster as input to Random Forest, roc-auc is "0.9015119047619049"
```

Let's now explore if the number of clusters have an effect in this improvement.

- Create the basis training set from `data` by restricting to float_columns.
- For $n = 1, \ldots, 20$, fit a kmeans algotihm with n clusters. One hot encode it and add it to the **basis** training set. Don't add it to the previous iteration.
- Fit 10 **Logistic Regression** models and compute the average roc-auc-score.
- Plot the average roc-auc scores.

```
X_basis = data[float_columns]
sss = StratifiedShuffleSplit(n_splits=10, random_state=6532)


def create_kmeans_columns(n):
    km = KMeans(n_clusters=n)
    km.fit(X_basis)
    km_col = pd.Series(km.predict(X_basis))
    km_cols = pd.get_dummies(km_col, prefix='kmeans_cluster')
    return pd.concat([X_basis, km_cols], axis=1)
```