Lab 4: Scikit Learn, Classification and Clustering

Deadline Friday 4/29/22 11:59 pm

scikit-learn is a popular machine learning package that contains a variety of models and tools.

All objects within scikitt-learn share a uniform common basic API consisting of 3 interfaces: an *estimator* interface for building and fitting models, a *predictor* interface for making predictions, and a *transformer* interface for converting data.

> The *estimator* interface defines object mechanism and a fit method for learning a model from training data. All supervised and unsupervised learning algorithms are offered as objects implementing this interface. Other machine learning tasks such as *feature extraction*, *feature selection*, and *dimensionality reduction* are provided as *estimators*.

For more information, check the scikit-learn API paper: [https://arxiv.org/pdf/1309.0238v1.pdf]

The general form of using models in scikit-learn:

```
clf = someModel( )
clf.fit(x_train , y_tain)
```

For Example:

```
clf = LinearSVC( )
clf.fit(x_train , y_tain)
```

> The *predictor* adds a predict method that takes an array x_test and produces predictions for x_test, based on the learned parameters of the *estimator*. In supervised learning, this method typically return predicted labels or values computed by the model. Some unsupervised learning estimators may also implement the predict interface, such as **k-means**, where the predicted values are the cluster labels.

```
clf.predict(x_test)
```

> *transform* method is used to modify or filter data before feeding it to a learning algorithm. It takes some new data as input and outputs a transformed version of that data. Preprocessing, feature selection, feature extraction and dimensionality reduction algorithms are all provided as *transformers* within the library.

This is usually done with **fit_transform** method. For example:

```
PCA = RandomizedPCA (n_components = 2)
x_train = PCA.fit_transform(x_train)
```

```
x_test = PCA.fit_transform(x_test)
```

In the example above, we first **fit** the training set to find the PC components, then they are transformed.

We can summarize the *estimator* as follows:

- In *all estimators*
  - `model.fit()` : fit training data. In supervised learning, fit will take two parameters: the data x and labels y. In unsupervised learning, fit will take a single parameter: the data x
- In *supervised estimators*
  - `model.predict()` : predict the label of new test data for the given model. Predict takes one parameter: the new test data and returns the learned label for each item in the test data
  - `model.score()` : Returns the score method for classification or regression methods.
- In *unsupervised estimators*
  - `model.transform()` : Tranform new data into new basis. Transform takes one parameter: new data and returns a new representation of that data based on the model

Classification: SVM

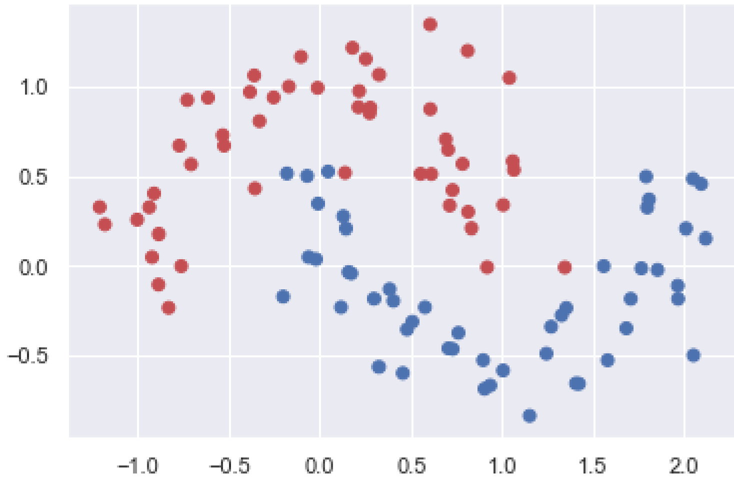
Support Vector Machines (SVM) are among the most useful and powerful supervised learning algorithm. Here we are going to look at an example of using SVM models in scikit-learn. Then, it will be your turn to try this model.

In [2]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
```

In [3]:

```
from sklearn.model_selection import train_test_split
# Import make_moons from scikit learn to generate synthetic data
from sklearn.datasets import make_moons


# 2d classification dataset
Xs , ys = make_moons( n_samples = 100,noise = 0.2 , random_state = 0)


# train-test split

Xs_train , Xs_test, ys_train, ys_test = train_test_split(Xs, ys , test_size = 0.15 )

#plot the data
colors = np.array(['r' , 'b'])
plt.scatter(Xs[:,0] , Xs[:,1]  ,c = colors[ys] )
plt.show()
```



We will perform both linear and nonlinear SVM on this synthetic dataset:

In [24]:

```
def meshGrid (x , y , h):
    '''x is data for x-axis meshgrid
       y is data for y-axis meshgrid
       h is stepsize
    '''
    x_min, x_max = x.min() - 1 , x.max() + 1
    y_min, y_max = y.min() - 1 , y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    return xx , yy
```

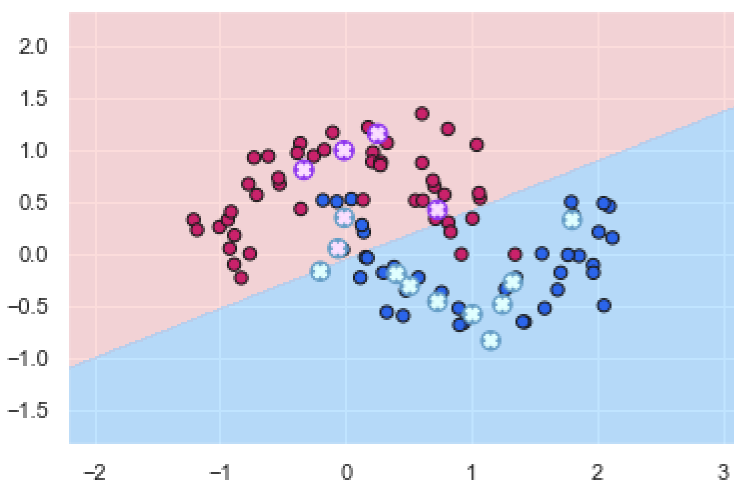
In [ ]:

In [5]:

```
#Import SVM
from sklearn import svm

from matplotlib.colors import ListedColormap
from sklearn import metrics


cmap_light = ListedColormap(['#FBBBB9', '#82CAFF'])
cmap_bold = ListedColormap(['#CA226B', '#2B65EC'])
cmap_test = ListedColormap(['#8E35EF', '#659EC7'])
cmap_predict = ListedColormap(['#FCDFFF', '#E0FFFF'])

# clf1 is a linear svm classifier
clf1 = svm.SVC(kernel = 'linear')

# Fit data
clf1.fit(Xs_train, ys_train)

# Predict
ys_predict = clf1.predict(Xs_test)


#Display the outcome of classification
print(metrics.classification_report(ys_test, ys_predict))
print(metrics.confusion_matrix(ys_test, ys_predict))
```

```
# Display the svm
xx , yy = meshGrid(Xs[:,0], Xs[:,1], 0.01)



Z = clf1.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure()
plt.contourf(xx, yy, Z, cmap=cmap_light ,levels=[-1, 0, 1] ,alpha = 0.5)

# For plotting all data use the following line
#plt.scatter(Xs[:, 0], Xs[:, 1], c=ys, cmap=cmap_bold, edgecolor='k', s=50)

# For plotting train and test and prediction separatley
plt.scatter(Xs_train[:, 0], Xs_train[:, 1], c=ys_train, cmap=cmap_bold,edgecolor='k', s
plt.scatter(Xs_test[:, 0], Xs_test[:, 1], alpha=1.0,c = ys_test, cmap=cmap_test,linewid
plt.scatter(Xs_test[:, 0], Xs_test[:, 1], alpha=1.0,c = ys_predict, cmap=cmap_predict ,
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())

plt.show()
```

```
             precision    recall  f1-score   support

          0       0.67      1.00      0.80         4
          1       1.00      0.82      0.90        11

avg / total       0.91      0.87      0.87        15

[[4 0]
 [2 9]]
```



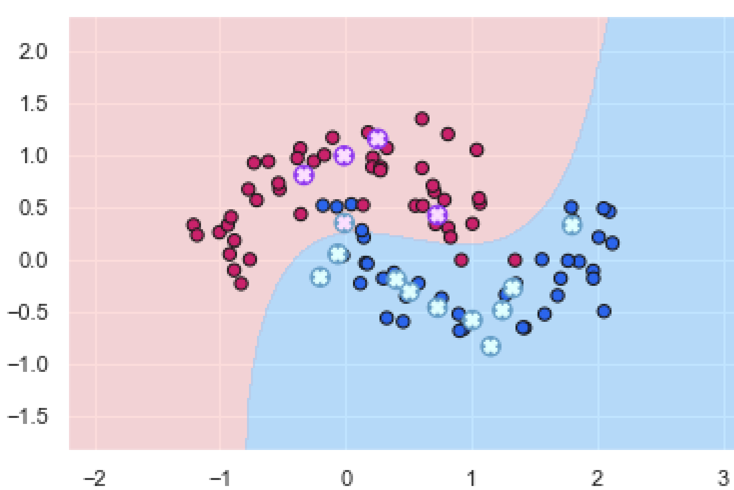
Now we apply a non-linear svm classifier

In [6]:

```
# clf2 is a nonlinear svm classifier

clf2 = svm.SVC(kernel = 'rbf')


# Fit data
```

```
clf2.fit(Xs_train, ys_train)

# Predict
ys_predict2 = clf2.predict(Xs_test)



#Display the outcome of classification
print(metrics.classification_report(ys_test, ys_predict2))
print(metrics.confusion_matrix(ys_test, ys_predict2))

# Display the svm
xx , yy = meshGrid(Xs[:,0], Xs[:,1], 0.01)



Z = clf2.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure()
plt.contourf(xx, yy, Z, cmap=cmap_light ,levels=[-1, 0, 1] ,alpha = 0.5)

# For plotting all data use the following line
#plt.scatter(Xs[:, 0], Xs[:, 1], c=ys, cmap=cmap_bold, edgecolor='k', s=50)

# For plotting train and test and prediction separatley
plt.scatter(Xs_train[:, 0], Xs_train[:, 1], c=ys_train, cmap=cmap_bold,edgecolor='k', s
plt.scatter(Xs_test[:, 0], Xs_test[:, 1], alpha=1.0,c = ys_test, cmap=cmap_test,linewid
plt.scatter(Xs_test[:, 0], Xs_test[:, 1], alpha=1.0,c = ys_predict2, cmap=cmap_predict
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())

plt.show()
```

```
             precision    recall  f1-score   support

          0       0.80      1.00      0.89         4
          1       1.00      0.91      0.95        11

avg / total       0.95      0.93      0.94        15

[[ 4  0]
 [ 1 10]]
```



SVM on Wine quality dataset

Exercise 4.1 (30 pts)

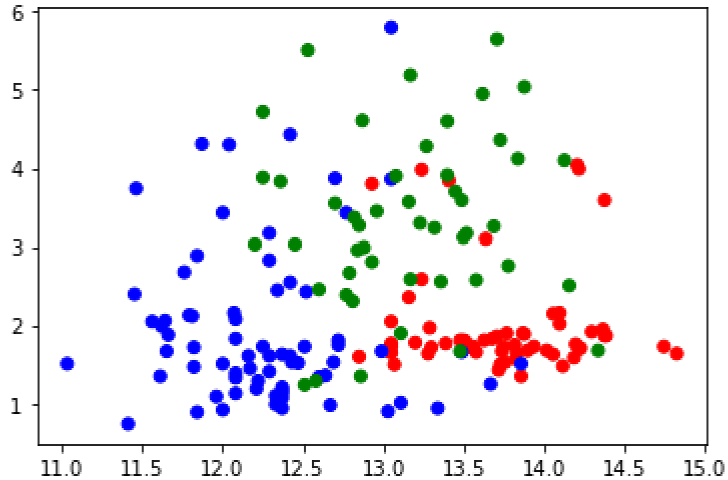
Now it's your turn to work with SVM. The wine data set is loaded below. You can learn more about the dataset by using `datasett.DESCR` . Here, you need to work with the first two features to train your model.

- Select the first two features for your X
- Split the dataset in two sets of training and testing data. Use 80% of the data for training and 20% for testing
- Perform linear and non-linear SVM on the dataset
- Display the classification report and accuracy for both models

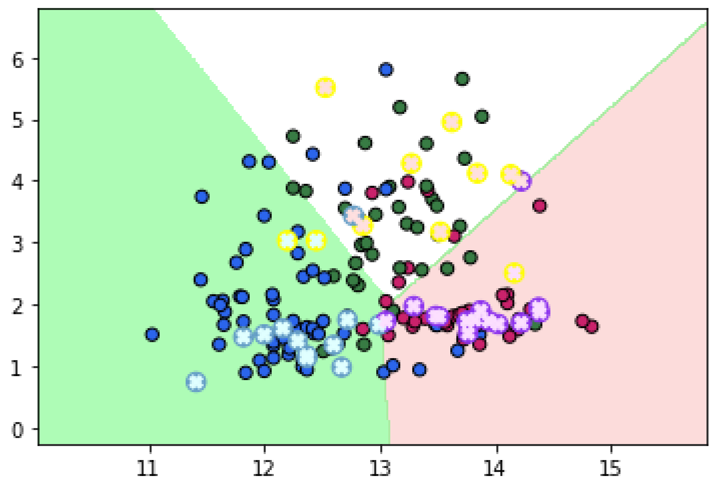
In [100…

```
from sklearn.datasets import load_wine
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import svm
from matplotlib.colors import ListedColormap
from sklearn import metrics

Xwine_full , ywine = load_wine(return_X_y = True)

#Your code here

Xc = Xwine_full [:, :2]
X_train,X_test,y_train,y_test = train_test_split(Xc, ywine, test_size = 0.20)

colors = np.array(['r' , 'b', 'g'])
plt.scatter(Xc[:,0] , Xc[:,1]  ,c = colors[ywine])
plt.show()

cmap_light = ListedColormap(['#FBBBB9', '#82CAFF', '#5EFB6E'])
cmap_bold = ListedColormap(['#CA226B', '#2B65EC', '#387C44'])
cmap_test = ListedColormap(['#8E35EF', '#659EC7', '#FFFF00'])
cmap_predict = ListedColormap(['#FCDFFF', '#E0FFFF', '#FFE0E0'])

# clf1 is a linear svm classifier
clf1 = svm.SVC(kernel = 'linear')

# Fit data
clf1.fit(X_train, y_train)

# Predict
ys_predict = clf1.predict(X_test)


#Display the outcome of classification
print(metrics.classification_report(y_test, ys_predict))
print(metrics.confusion_matrix(y_test, ys_predict))

# Display the svm
```

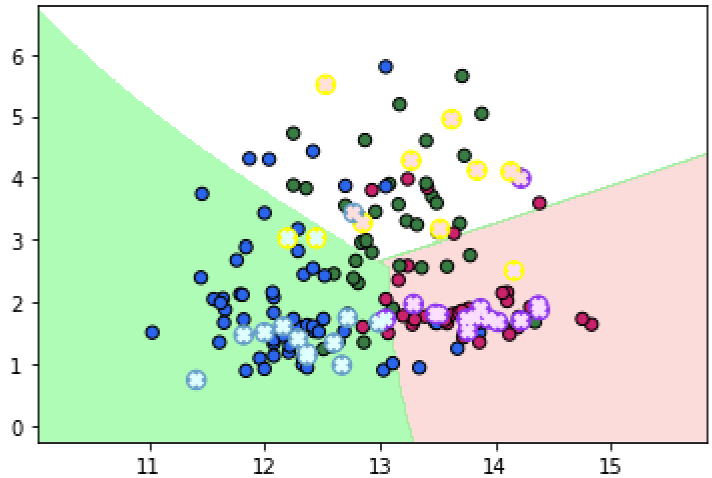
```
xx , yy = meshGrid(Xc[:,0], Xc[:,1], 0.01)



Z = clf1.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure()
plt.contourf(xx, yy, Z, cmap=cmap_light ,levels=[-1, 0, 1] ,alpha = 0.5)

# For plotting all data use the following line
#plt.scatter(Xs[:, 0], Xs[:, 1], c=ys, cmap=cmap_bold, edgecolor='k', s=50)

# For plotting train and test and prediction separatley
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap_bold,edgecolor='k', s=40
plt.scatter(X_test[:, 0], X_test[:, 1], alpha=1.0,c = y_test, cmap=cmap_test,linewidth=
plt.scatter(X_test[:, 0], X_test[:, 1], alpha=1.0,c = ys_predict, cmap=cmap_predict ,li
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())

plt.show()

clf2 = svm.SVC(kernel = 'rbf')


# Fit data
clf2.fit(X_train, y_train)

# Predict
ys_predict2 = clf2.predict(X_test)



#Display the outcome of classification
print(metrics.classification_report(y_test, ys_predict2))
print(metrics.confusion_matrix(y_test, ys_predict2))

# Display the svm
xx1 , yy1 = meshGrid(Xc[:,0], Xc[:,1], 0.01)



Z1 = clf2.predict(np.c_[xx1.ravel(), yy1.ravel()])
Z1 = Z1.reshape(xx1.shape)

plt.figure()
plt.contourf(xx1, yy1, Z1, cmap=cmap_light ,levels=[-1, 0, 1] ,alpha = 0.5)

# For plotting all data use the following line
#plt.scatter(Xs[:, 0], Xs[:, 1], c=ys, cmap=cmap_bold, edgecolor='k', s=50)

# For plotting train and test and prediction separatley
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap_bold,edgecolor='k', s=40
plt.scatter(X_test[:, 0], X_test[:, 1], alpha=1.0,c = y_test, cmap=cmap_test,linewidth=
plt.scatter(X_test[:, 0], X_test[:, 1], alpha=1.0,c = ys_predict2, cmap=cmap_predict ,l
plt.xlim(xx1.min(), xx1.max())
plt.ylim(yy1.min(), yy1.max())

plt.show()
```




```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        14
           1       0.85      0.92      0.88        12
           2       0.78      0.70      0.74        10

    accuracy                           0.86        36
   macro avg       0.85      0.85      0.85        36
weighted avg       0.86      0.86      0.86        36

[[13  0  1]
 [ 0 11  1]
 [ 1  2  7]]
```




```
              precision    recall  f1-score   support

           0       0.92      0.86      0.89        14
           1       0.79      0.92      0.85        12
           2       0.78      0.70      0.74        10

    accuracy                           0.83        36
   macro avg       0.83      0.82      0.82        36
weighted avg       0.84      0.83      0.83        36

[[12  1  1]
 [ 0 11  1]
 [ 1  2  7]]
```



Exercise 4.2 (10 pts)

Scaling features is another step that can affect the performance of your classifier. For the wine data, scale the features using `StandardScaler` and perform linear SVM. Display the classification report and accuracy. Did scaling data affect the classifier performance?

Yes, Scaling data made the performance better and it gave the higher accuracy output at some extent.

In [101…

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()


#Your code here

X_train_new = scaler.fit_transform(X_train)
X_test_new = scaler.transform(X_test)

# clf1 is a linear svm classifier
clf3 = svm.SVC(kernel = 'linear')

# Fit data
clf3.fit(X_train_new, y_train)

# Predict
ys_predict3 = clf3.predict(X_test_new)


#Display the outcome of classification
print(metrics.classification_report(y_test, ys_predict3))
print(metrics.confusion_matrix(y_test, ys_predict3))
```

```
              precision    recall  f1-score   support

           0       0.93      1.00      0.97        14
           1       0.85      0.92      0.88        12
           2       0.88      0.70      0.78        10

    accuracy                           0.89        36
   macro avg       0.88      0.87      0.87        36
```

```
weighted avg       0.89      0.89      0.88        36

[[14  0  0]
 [ 0 11  1]
 [ 1  2  7]]
```

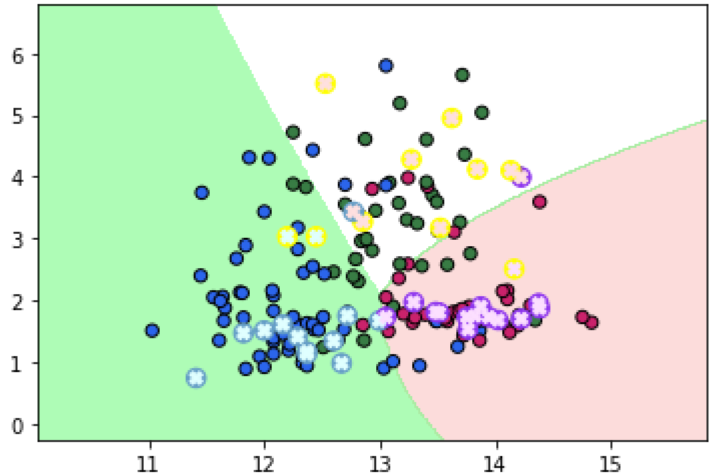
Exercise 4.3 (10 pts)

scikit-learn has many other classifiers. Pick another classifier of your choice (KNN, DecisionTree, NaiveBayes, ...) and apply it to the wine dataset. Display the classification report and accuracy.

In [102...

```python
#Your code goes here
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
model.fit(X_train, y_train)
model_predict = model.predict(X_test)

#Display the outcome of classification
print(metrics.classification_report(y_test, model_predict))
print(metrics.confusion_matrix(y_test, model_predict))

# Display the svm
xx4 , yy4 = meshGrid(Xc[:,0], Xc[:,1], 0.01)



Z4 = model.predict(np.c_[xx4.ravel(), yy4.ravel()])
Z4 = Z4.reshape(xx4.shape)

plt.figure()
plt.contourf(xx4, yy4, Z4, cmap=cmap_light ,levels=[-1, 0, 1] ,alpha = 0.5)

# For plotting all data use the following line
#plt.scatter(Xs[:, 0], Xs[:, 1], c=ys, cmap=cmap_bold, edgecolor='k', s=50)

# For plotting train and test and prediction separatley
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap_bold,edgecolor='k', s=40
plt.scatter(X_test[:, 0], X_test[:, 1], alpha=1.0,c = y_test, cmap=cmap_test,linewidth=
plt.scatter(X_test[:, 0], X_test[:, 1], alpha=1.0,c = model_predict, cmap=cmap_predict
plt.xlim(xx4.min(), xx4.max())
plt.ylim(yy4.min(), yy4.max())

plt.show()
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        14
           1       0.85      0.92      0.88        12
           2       0.78      0.70      0.74        10

    accuracy                           0.86        36
   macro avg       0.85      0.85      0.85        36
weighted avg       0.86      0.86      0.86        36

[[13  0  1]
 [ 0 11  1]
 [ 1  2  7]]
```



Clustering

You have already seen an example of clustering using scikit-learn in lecture. In this section, you will apply `KMeans` algorithm to the wine dataset.

Exercise 4.4 (30 pts)

- First choose the first two features and apply kmeans clustering.
- Display cluster evaluation metrics `homogeneity_score and completeness_score` (both belong to sklearn.metrics)
- Plot the clusters and centroids. You have the "ground truth" or labels of your data points, your plot should create a meshgrid to display the decision boundary of your model, and add the datapoints and their true labels. (This is to observe how good your model performs on the data)

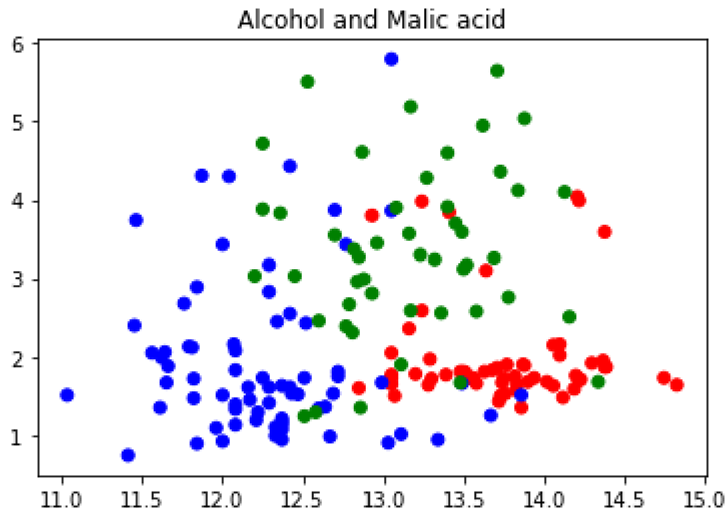
Note: For displaying decision boundaries and data points follow these steps:

1. Use meshGrid function to get the mesh for your attributes
2. Obtain labels for each point in mesh and reshape it. (Z = kmeans.predict(....))
3. Put the results into a color plot
   - Plot the colormesh --> plt.pcolormesh
   - Plot your data points --> plt.scatter
   - Plot the centroids --> plt.scatter
   - Set titles, x and y ranges
   - plt.show()

In [103…

```
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import completeness_score
from sklearn.metrics.cluster import homogeneity_score

Xwine_full , ywine = load_wine(return_X_y = True)

Xc = Xwine_full [:, :2]

colormap = np.array(['r' , 'b' , 'g'])
plt.scatter(Xc[:,0],Xc[:,1] , c = colormap[ywine])
```

```
plt.title("Alcohol and Malic acid")
plt.show()
```



In [104…

```
# Your code here
cmap_light = ListedColormap(['#FBBBB9', '#82CAFF', '#5EFB6E'])
cmap_bold = ListedColormap(['#CA226B', '#2B65EC', '#387C44'])
cmap_test = ListedColormap(['#8E35EF', '#659EC7', '#FFFF00'])
cmap_predict1 = ListedColormap(['#8105ED', '#ED05CA', '#FA0505'])

# Run k-means, Number of clusters = 2
y1_predict = KMeans(n_clusters = 3, random_state = 170).fit_predict(Xc)

#plot
print("Completness Score: ",completeness_score(ywine, y1_predict))
print("Homogeneity Score: ",homogeneity_score(ywine, y1_predict))
```

```
Completness Score:  0.4080524820388842
Homogeneity Score:  0.4103507797096971
```

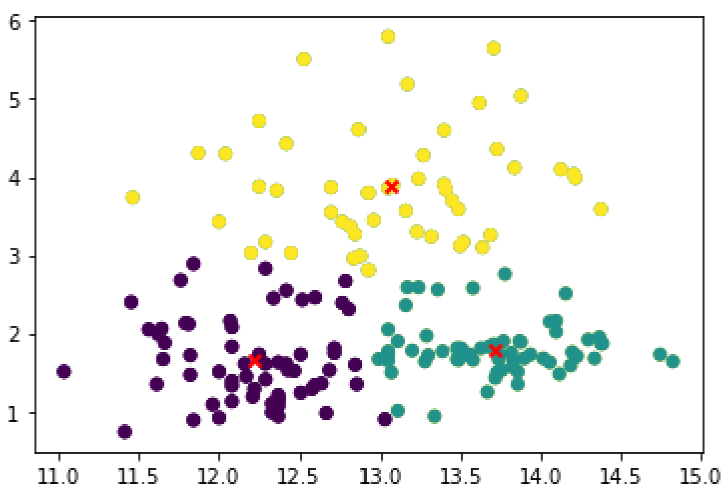
In [105…

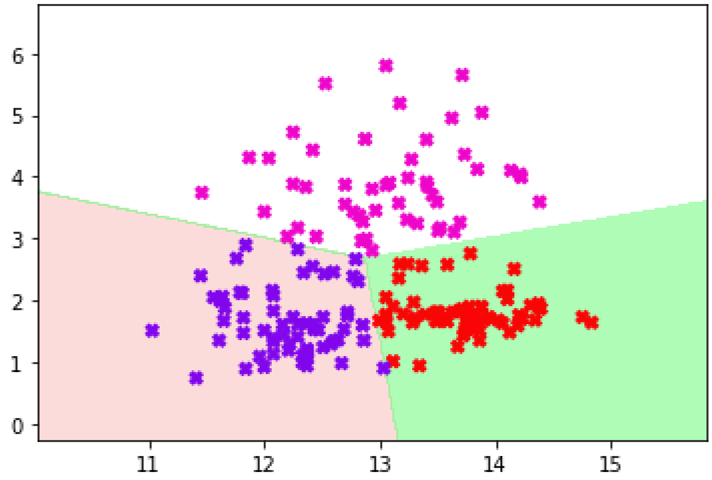
```
kmeans = KMeans(n_clusters = 3, init ='random', random_state = 200, verbose=True).fit(X

plt.scatter(Xc[:,0], Xc[:,1], c =y1_predict)
plt.scatter(Xc[:,0], Xc[:,1], c= kmeans.labels_)
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],c = 'r',marker ='
plt.show()

xx5 , yy5 = meshGrid(Xc[:,0], Xc[:,1], 0.01)

Z5 = kmeans.predict(np.c_[xx5.ravel(), yy5.ravel()])
Z5 = Z5.reshape(xx4.shape)

plt.figure()
plt.contourf(xx5, yy5, Z5, cmap=cmap_light ,levels=[-1, 0, 1] ,alpha = 0.5)

# For plotting all data use the following line
#plt.scatter(Xs[:, 0], Xs[:, 1], c=ys, cmap=cmap_bold, edgecolor='k', s=50)

# For plotting train and test and prediction separatley
plt.scatter(Xc[:, 0], Xc[:, 1], alpha=1.0,c = y1_predict, cmap=cmap_predict1 ,linewidth
plt.xlim(xx5.min(), xx5.max())
plt.ylim(yy5.min(), yy5.max())
```

```
plt.show()
```

```
Initialization complete
Iteration 0, inertia 151.7837000000001
Iteration 1, inertia 99.30364833650812
Iteration 2, inertia 96.21666479021775
Iteration 3, inertia 95.60041513507419
Iteration 4, inertia 95.55394205616612
Converged at iteration 4: strict convergence.
Initialization complete
Iteration 0, inertia 549.3189000000003
Iteration 1, inertia 189.82676406927294
Iteration 2, inertia 108.35491426116245
Iteration 3, inertia 98.55081215657327
Iteration 4, inertia 96.2556072332222
Iteration 5, inertia 96.03746916956139
Iteration 6, inertia 95.63063596469088
Iteration 7, inertia 95.57183751367191
Iteration 8, inertia 95.55394205616612
Converged at iteration 8: strict convergence.
Initialization complete
Iteration 0, inertia 226.13800000000003
Iteration 1, inertia 123.8032406649898
Iteration 2, inertia 101.2007092408733
Iteration 3, inertia 96.12908247185473
Iteration 4, inertia 95.55989885435001
Converged at iteration 4: strict convergence.
Initialization complete
Iteration 0, inertia 300.0688999999998
Iteration 1, inertia 134.91766665736398
Iteration 2, inertia 109.23193825270297
Iteration 3, inertia 97.32424214615946
Iteration 4, inertia 95.87981960678495
Iteration 5, inertia 95.55989885435001
Converged at iteration 5: strict convergence.
Initialization complete
Iteration 0, inertia 361.35339999999985
Iteration 1, inertia 191.87547238636702
Iteration 2, inertia 159.2018810445558
Iteration 3, inertia 111.37486848019257
Iteration 4, inertia 97.19406178248845
Iteration 5, inertia 96.08770320254003
Iteration 6, inertia 95.63063596469088
Iteration 7, inertia 95.57183751367191
Iteration 8, inertia 95.55394205616612
Converged at iteration 8: strict convergence.
Initialization complete
Iteration 0, inertia 164.90930000000003
Iteration 1, inertia 99.34740054621203
Iteration 2, inertia 96.55853931708775
Iteration 3, inertia 96.16699229731377
Iteration 4, inertia 95.79042113653588
Iteration 5, inertia 95.57183751367191
Iteration 6, inertia 95.55394205616612
Converged at iteration 6: strict convergence.
Initialization complete
Iteration 0, inertia 502.4055999999999
Iteration 1, inertia 164.17815920538158
Iteration 2, inertia 153.29269456209832
```

```
Iteration 3, inertia 148.8612646005445
Iteration 4, inertia 146.72308116227103
Iteration 5, inertia 146.1975250439638
Iteration 6, inertia 145.4802019123721
Iteration 7, inertia 140.6861804860087
Iteration 8, inertia 136.85367487414223
Iteration 9, inertia 133.5144659668164
Iteration 10, inertia 126.48860322654284
Iteration 11, inertia 116.26765870918304
Iteration 12, inertia 102.8478692868655
Iteration 13, inertia 96.12908247185473
Iteration 14, inertia 95.55989885435001
Converged at iteration 14: strict convergence.
Initialization complete
Iteration 0, inertia 146.70680000000002
Iteration 1, inertia 99.50908220820313
Iteration 2, inertia 95.95970345518677
Iteration 3, inertia 95.60660697105247
Iteration 4, inertia 95.55394205616612
Converged at iteration 4: strict convergence.
Initialization complete
Iteration 0, inertia 126.47810000000007
Iteration 1, inertia 99.16788129617385
Iteration 2, inertia 96.47641962235046
Iteration 3, inertia 95.68708159216268
Iteration 4, inertia 95.55394205616612
Converged at iteration 4: strict convergence.
Initialization complete
Iteration 0, inertia 336.55079999999987
Iteration 1, inertia 196.2055058589258
Iteration 2, inertia 164.5474488010176
Iteration 3, inertia 117.09913739579174
Iteration 4, inertia 98.50947900158808
Iteration 5, inertia 95.96815455205041
Iteration 6, inertia 95.63063596469088
Iteration 7, inertia 95.57183751367191
Iteration 8, inertia 95.55394205616612
Converged at iteration 8: strict convergence.
```





Exercise 4.5 (20 pts)

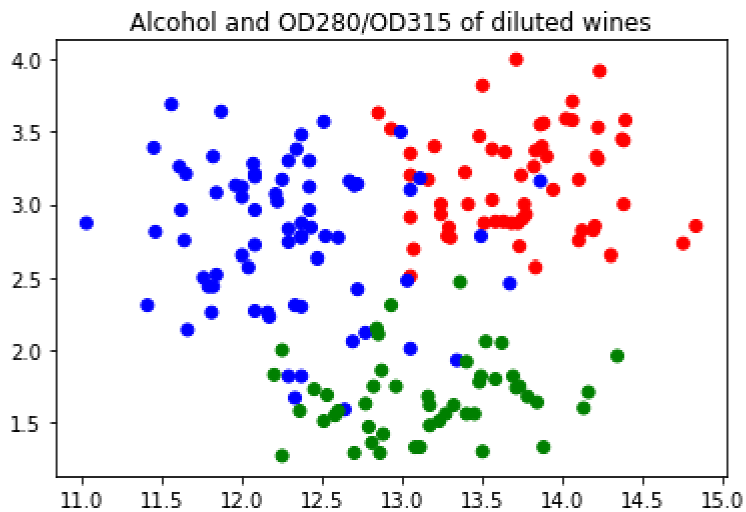
In the previous model you used the first two features: 'Alcohol' and 'Malic acid'. For this exercise, pick features 'Alcohol' and 'OD280/OD315 of diluted wines' (feature #1 and feature #12) as your two attributes and perform the tasks in Exercise 4.4. (cluster, report metrics, draw decision boundaries)

Which model performs better?

This model performance better because Completness Score: 0.7006853440435565 and Homogeneity Score: 0.7072039236692641 is way higher than compare to 4.4 model.

In [106…

```
# your code here
Xwine_full , ywine = load_wine(return_X_y = True)

Xc = Xwine_full[:, [0, 11]]

colormap = np.array(['r' , 'b' , 'g'])
plt.scatter(Xc[:,0],Xc[:,1] , c = colormap[ywine])
plt.title("Alcohol and OD280/OD315 of diluted wines")
plt.show()
```



In [107…

```
cmap_light = ListedColormap(['#FBBBB9', '#82CAFF', '#5EFB6E'])
```

```python
cmap_bold = ListedColormap(['#CA226B', '#2B65EC', '#387C44'])
cmap_test = ListedColormap(['#8E35EF', '#659EC7', '#FFFF00'])
cmap_predict1 = ListedColormap(['#8105ED', '#ED05CA', '#FA0505'])

# Run k-means, Number of clusters = 2
y1_predict = KMeans(n_clusters = 3, random_state = 200).fit_predict(Xc)

#plot
print("Completness Score: ",completeness_score(ywine, y1_predict))
print("Homogeneity Score: ",homogeneity_score(ywine, y1_predict))
```

```
Completness Score:  0.7006853440435565
Homogeneity Score:  0.7072039236692641
```

In [108…

```python
Xc = Xwine_full[:, [0, 11]]
print(Xc)
kmeans = KMeans(n_clusters = 3, init ='random', random_state = 200, verbose=True).fit(X

plt.scatter(Xc[:,0], Xc[:,1], c =y1_predict)
plt.scatter(Xc[:,0], Xc[:,1], c= kmeans.labels_)
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],c = 'r',marker ='
plt.show()

xx6 , yy6 = meshGrid(Xc[:,0], Xc[:,1], 0.01)

Z6 = kmeans.predict(np.c_[xx6.ravel(), yy6.ravel()])
Z6 = Z6.reshape(xx6.shape)

plt.figure()
plt.contourf(xx6, yy6, Z6, cmap=cmap_light ,levels=[-1, 0, 1] ,alpha = 0.5)

# For plotting all data use the following line
#plt.scatter(Xs[:, 0], Xs[:, 1], c=ys, cmap=cmap_bold, edgecolor='k', s=50)

# For plotting train and test and prediction separatley
plt.scatter(Xc[:, 0], Xc[:, 1], alpha=1.0,c = y1_predict, cmap=cmap_predict1 ,linewidth
plt.xlim(xx6.min(), xx6.max())
plt.ylim(yy6.min(), yy6.max())

plt.show()
```

```
[[14.23  3.92]
 [13.2   3.4 ]
 [13.16  3.17]
 [14.37  3.45]
 [13.24  2.93]
 [14.2   2.85]
 [14.39  3.58]
 [14.06  3.58]
 [14.83  2.85]
 [13.86  3.55]
 [14.1   3.17]
 [14.12  2.82]
 [13.75  2.9 ]
 [14.75  2.73]
 [14.38  3.  ]
 [13.63  2.88]
 [14.3   2.65]
 [13.83  2.57]
```

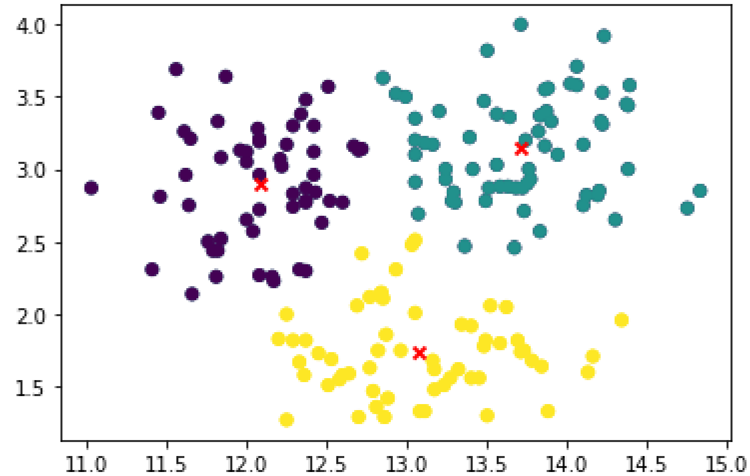
```
 [14.19  2.82]
 [13.64  3.36]
 [14.06  3.71]
 [12.93  3.52]
 [13.71  4.  ]
 [12.85  3.63]
 [13.5   3.82]
 [13.05  3.2 ]
 [13.39  3.22]
 [13.3   2.77]
 [13.87  3.4 ]
 [14.02  3.59]
 [13.73  2.71]
 [13.58  2.88]
 [13.68  2.87]
 [13.76  3.  ]
 [13.51  2.87]
 [13.48  3.47]
 [13.28  2.78]
 [13.05  2.51]
 [13.07  2.69]
 [14.22  3.53]
 [13.56  3.38]
 [13.41  3.  ]
 [13.88  3.56]
 [13.24  3.  ]
 [13.05  3.35]
 [14.21  3.33]
 [14.38  3.44]
 [13.9   3.33]
 [14.1   2.75]
 [13.94  3.1 ]
 [13.05  2.91]
 [13.83  3.37]
 [13.82  3.26]
 [13.77  2.93]
 [13.74  3.2 ]
 [13.56  3.03]
 [14.22  3.31]
 [13.29  2.84]
 [13.72  2.87]
 [12.37  1.82]
 [12.33  1.67]
 [12.64  1.59]
 [13.67  2.46]
 [12.37  2.87]
 [12.17  2.23]
 [12.37  2.3 ]
 [13.11  3.18]
 [12.37  3.48]
 [13.34  1.93]
 [12.21  3.07]
 [12.29  1.82]
 [13.86  3.16]
 [13.49  2.78]
 [12.99  3.5 ]
 [11.96  3.13]
 [11.66  2.14]
 [13.03  2.48]
 [11.84  2.52]
```

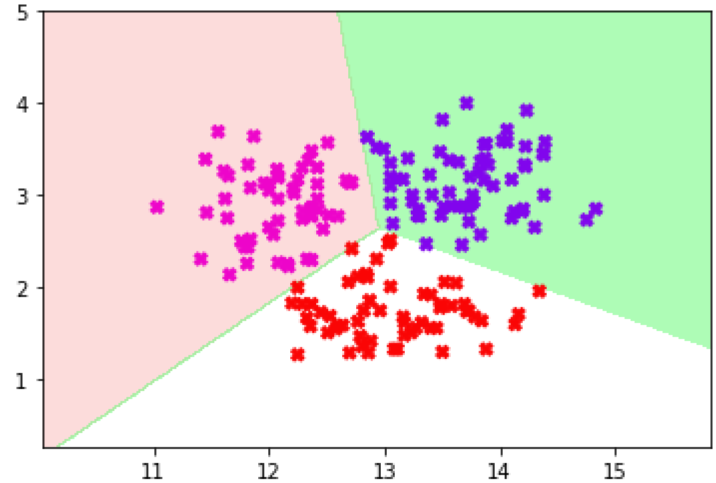
```
 [12.33  2.31]
 [12.7   3.13]
 [12.    3.12]
 [12.72  3.14]
 [12.08  2.72]
 [13.05  2.01]
 [11.84  3.08]
 [12.67  3.16]
 [12.16  2.26]
 [11.65  3.21]
 [11.64  2.75]
 [12.08  3.21]
 [12.08  2.27]
 [12.    2.65]
 [12.69  2.06]
 [12.29  3.3 ]
 [11.62  2.96]
 [12.47  2.63]
 [11.81  2.26]
 [12.29  2.74]
 [12.37  2.77]
 [12.29  2.83]
 [12.08  2.96]
 [12.6   2.77]
 [12.34  3.38]
 [11.82  2.44]
 [12.51  3.57]
 [12.42  3.3 ]
 [12.25  3.17]
 [12.72  2.42]
 [12.22  3.02]
 [11.61  3.26]
 [11.46  2.81]
 [12.52  2.78]
 [11.76  2.5 ]
 [11.41  2.31]
 [12.08  3.19]
 [11.03  2.87]
 [11.82  3.33]
 [12.42  2.96]
 [12.77  2.12]
 [12.    3.05]
 [11.45  3.39]
 [11.56  3.69]
 [12.42  3.12]
 [13.05  3.1 ]
 [11.87  3.64]
 [12.07  3.28]
 [12.43  2.84]
 [11.79  2.44]
 [12.37  2.78]
 [12.04  2.57]
 [12.86  1.29]
 [12.88  1.42]
 [12.81  1.36]
 [12.7   1.29]
 [12.51  1.51]
 [12.6   1.58]
 [12.25  1.27]
 [12.53  1.69]
```

```
 [13.49  1.82]
 [12.84  2.15]
 [12.93  2.31]
 [13.36  2.47]
 [13.52  2.06]
 [13.62  2.05]
 [12.25  2.  ]
 [13.16  1.68]
 [13.88  1.33]
 [12.87  1.86]
 [13.32  1.62]
 [13.08  1.33]
 [13.5   1.3 ]
 [12.79  1.47]
 [13.11  1.33]
 [13.23  1.51]
 [12.58  1.55]
 [13.17  1.48]
 [13.84  1.64]
 [12.45  1.73]
 [14.34  1.96]
 [13.48  1.78]
 [12.36  1.58]
 [13.69  1.82]
 [12.85  2.11]
 [12.96  1.75]
 [13.78  1.68]
 [13.73  1.75]
 [13.45  1.56]
 [12.82  1.75]
 [13.58  1.8 ]
 [13.4   1.92]
 [12.2   1.83]
 [12.77  1.63]
 [14.16  1.71]
 [13.71  1.74]
 [13.4   1.56]
 [13.27  1.56]
 [13.17  1.62]
 [14.13  1.6 ]]
Initialization complete
Iteration 0, inertia 132.7549999999999
Iteration 1, inertia 62.55360383767907
Iteration 2, inertia 58.61901063358439
Iteration 3, inertia 58.382247645340385
Iteration 4, inertia 58.325945538943834
Converged at iteration 4: strict convergence.
Initialization complete
Iteration 0, inertia 284.7988000000001
Iteration 1, inertia 99.42618673559507
Iteration 2, inertia 67.82120886410212
Iteration 3, inertia 61.80121390118983
Iteration 4, inertia 59.539565124863934
Iteration 5, inertia 58.497102132111955
Iteration 6, inertia 58.325945538943834
Converged at iteration 6: strict convergence.
Initialization complete
Iteration 0, inertia 152.9971000000002
Iteration 1, inertia 81.85833266900501
Iteration 2, inertia 65.34792635199175
```

```
Iteration 3, inertia 59.95221319907506
Iteration 4, inertia 58.54423198571707
Iteration 5, inertia 58.450151711010335
Iteration 6, inertia 58.382247645340385
Iteration 7, inertia 58.325945538943834
Converged at iteration 7: strict convergence.
Initialization complete
Iteration 0, inertia 294.46950000000015
Iteration 1, inertia 128.38851366806745
Iteration 2, inertia 94.13081578350302
Iteration 3, inertia 64.43612695395325
Iteration 4, inertia 58.481049153601376
Iteration 5, inertia 58.37116387090805
Iteration 6, inertia 58.34320751540833
Converged at iteration 6: strict convergence.
Initialization complete
Iteration 0, inertia 125.20249999999996
Iteration 1, inertia 60.62295509104096
Iteration 2, inertia 58.567803888001336
Iteration 3, inertia 58.34320751540833
Converged at iteration 3: strict convergence.
Initialization complete
Iteration 0, inertia 229.87070000000003
Iteration 1, inertia 72.47210205500825
Iteration 2, inertia 61.36077153758995
Iteration 3, inertia 58.54772661539147
Iteration 4, inertia 58.40472389803835
Iteration 5, inertia 58.347779588931495
Iteration 6, inertia 58.325945538943834
Converged at iteration 6: strict convergence.
Initialization complete
Iteration 0, inertia 251.40840000000006
Iteration 1, inertia 104.4259348097662
Iteration 2, inertia 89.30179104765773
Iteration 3, inertia 70.12392008416644
Iteration 4, inertia 60.542237960593475
Iteration 5, inertia 58.51829821590716
Iteration 6, inertia 58.38390589446463
Iteration 7, inertia 58.347779588931495
Iteration 8, inertia 58.325945538943834
Converged at iteration 8: strict convergence.
Initialization complete
Iteration 0, inertia 136.00200000000007
Iteration 1, inertia 63.392825041598414
Iteration 2, inertia 58.54352322979602
Iteration 3, inertia 58.38390589446463
Iteration 4, inertia 58.347779588931495
Iteration 5, inertia 58.325945538943834
Converged at iteration 5: strict convergence.
Initialization complete
Iteration 0, inertia 123.16800000000006
Iteration 1, inertia 61.188081596336424
Iteration 2, inertia 58.770385233423056
Iteration 3, inertia 58.399696396681904
Iteration 4, inertia 58.347779588931495
Iteration 5, inertia 58.325945538943834
Converged at iteration 5: strict convergence.
Initialization complete
Iteration 0, inertia 113.84550000000004
Iteration 1, inertia 59.57558783470234
```

```
Iteration 2, inertia 58.4556352519123
Iteration 3, inertia 58.347779588931495
Iteration 4, inertia 58.325945538943834
Converged at iteration 4: strict convergence.
```





End Of Lab