Assignment

Question 1

Introduction

WHO announced that cardiovascular diseases is the top one killer over the world. There are seventeen million people died from it every year, especially heart disease. Prevention is better than cure. If we can evaluate the risk of every patient who probably has heart disease, that is, not only patients but also everyone can do something earlier to keep illness away.

This dataset is a real data including important features of patients. This time we will build the predictable model by logistic regression model, linear discrimnant analysis, and k-nearest Neighbors.

Confusion matrix is a common technique to figure out the accuracy of the model. From the standpoint of medicine, the recall rate is more important than precision rate because no one want to be misdiagnosed if the one actually have heart disease. So we will check the recall performance. After that, roc curve can help us evaluate the model, and then we'll explore the features if the model is good enough.

Dataset is available on https://archive.ics.uci.edu/ml/datasets/Heart+Disease (https://archive.ics.uci.edu/ml/datasets/Heart+Disease)

In [1]:
```
## Importing libraries
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sn
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
```

In [2]:
```
#Importing dataset
df = pd.read_csv('heart.csv',header =0)
df.head()
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Missing values and Multicolinearity

It's always a good practice to impute missing values and check the correlation among independent variables. Luckily, we did mot face any of the problem in this data set.

In [3]:
```
pd.isnull(df).sum()
```

Out[3]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

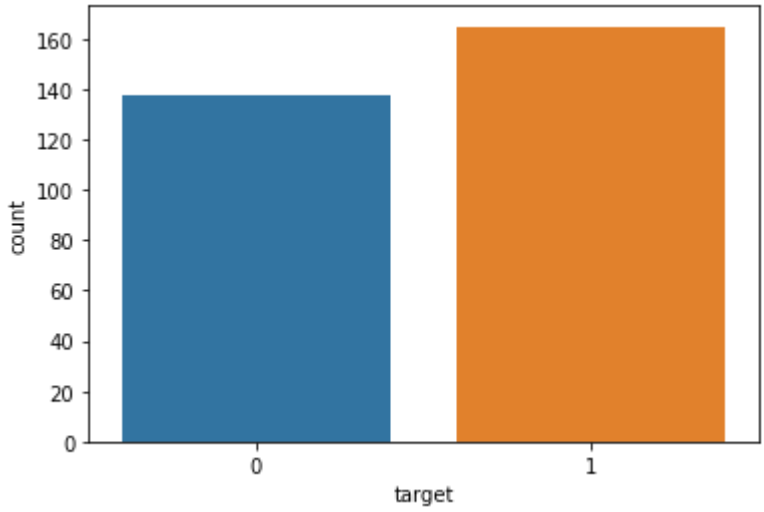
In [4]:
```
pd.crosstab(index=df['target'], columns='count')
```

Out[4]:

col_0	count
target	
0	138
1	165

In [5]:
```
sns.countplot(x='target', data=df)
```

Out[5]:
```
<AxesSubplot:xlabel='target', ylabel='count'>
```



Pairplot

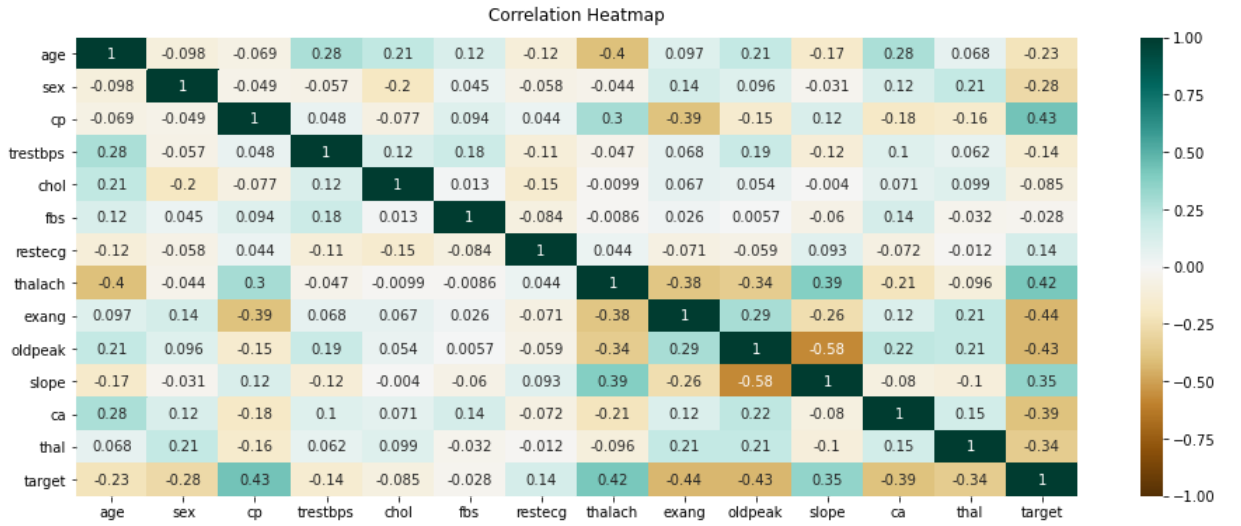
Pair plot is a one great piece of visualization to visualize all the variables at once.

In [6]:
```
sns.pairplot(df)
```

Out[6]: <seaborn.axisgrid.PairGrid at 0x1beea45ccd0>

Heat map of correlation among variables

In [7]:
```
plt.figure(figsize=(16, 6))
heatmap = sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True,cmap='BrBG')
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':12}, pad=12);
```



Splitting dataset on 70/30

In [8]:
```
from sklearn.model_selection import train_test_split
X = df.loc[:, df.columns != "target"]
y = df["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
print (X_train.shape,X_test.shape,y_train.shape, y_test.shape)
```

```
(212, 13) (91, 13) (212,) (91,)
```

Logistic Regression model

We will train model on traing dataset and compute confusion matrix and accuracy score on your testing dataset.

In [9]:
```
clf_LR = LogisticRegression(max_iter=999999999)
clf_LR.fit(X_train,y_train)
y_test_pred = clf_LR.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix
confusion_matrix(y_test, y_test_pred)
```

Out[9]:
```
array([[32, 12],
       [ 5, 42]], dtype=int64)
```

In [10]:
```
accuracy_score(y_test, y_test_pred)
```

Out[10]:
```
0.8131868131868132
```

K-Nearest Neighbors

We will train model on traing dataset and compute confusion matrix and accuracy score on your testing dataset.

In [11]:
```
knn = KNeighborsClassifier(n_neighbors=50)
knn.fit(X_train,y_train)
y_test_pred = knn.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix
confusion_matrix(y_test, y_test_pred)
```

Out[11]:
```
array([[20, 24],
       [ 5, 42]], dtype=int64)
```

In [12]:
```
accuracy_score(y_test, y_test_pred)
```

Out[12]:
```
0.6813186813186813
```

Linear Discriminant Analysis

We will train model on traing dataset and compute confusion matrix and accuracy score on your testing dataset.

In [13]:
```
lda = LinearDiscriminantAnalysis()
lda.fit(X_train,y_train)
y_test_pred = lda.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix
confusion_matrix(y_test, y_test_pred)
```

Out[13]:
```
array([[31, 13],
       [ 5, 42]], dtype=int64)
```

In [14]:
```
accuracy_score(y_test, y_test_pred)
```

Out[14]:
```
0.8021978021978022
```

Accuracy

Logistic regression gives the accuracy of 81.3187%. K-nearest neighbors gives the accuracy of 68.1319%. Linear discriminant analysis gives the accuracy of 80.22%.

Question 2

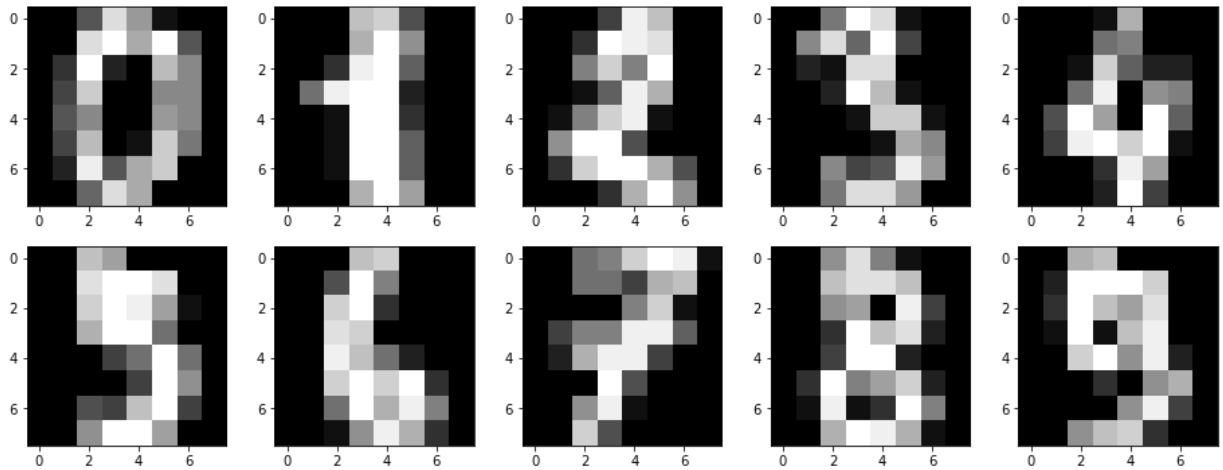
Dataset

We used a well known data setof hand written digits from 0 to 9 by MINST ("Modified National Institute of Standards and Technology") avaible on https://www.kaggle.com/c/digit-recognizer (https://www.kaggle.com/c/digit-recognizer). This dataset is also available in Sk-learn datasets. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike. We will implement principal component analysis to classify the digits.

Principal Component analysis

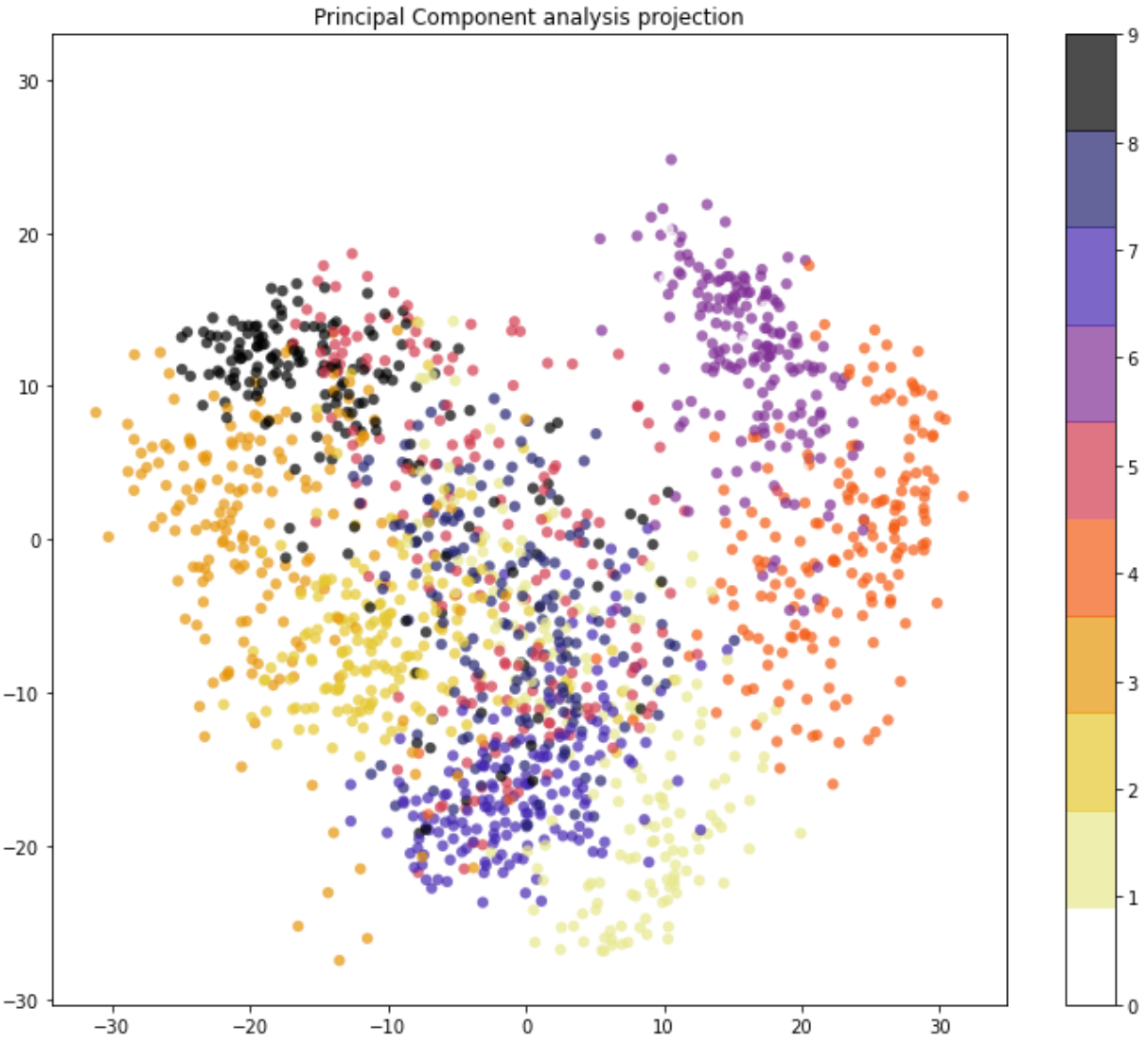
In [15]:
```
from sklearn import datasets
digits = datasets.load_digits()
X = digits.data
y = digits.target
```

In [16]:
```
# f, axes = plt.subplots(5, 2, sharey=True, figsize=(16,6))
plt.figure(figsize=(16, 6))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X[i,:].reshape([8,8]), cmap='gray');
```



In [17]:
```
from sklearn import decomposition
pca = decomposition.PCA(n_components=2)
X_reduced = pca.fit_transform(X)

print('Projecting %d-dimensional data to 2D' % X.shape[1])

plt.figure(figsize=(12,10))
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y,
            edgecolor='none', alpha=0.7, s=40,
            cmap=plt.cm.get_cmap('CMRmap_r', 10))
plt.colorbar()
plt.title('Principal Component analysis projection');
```

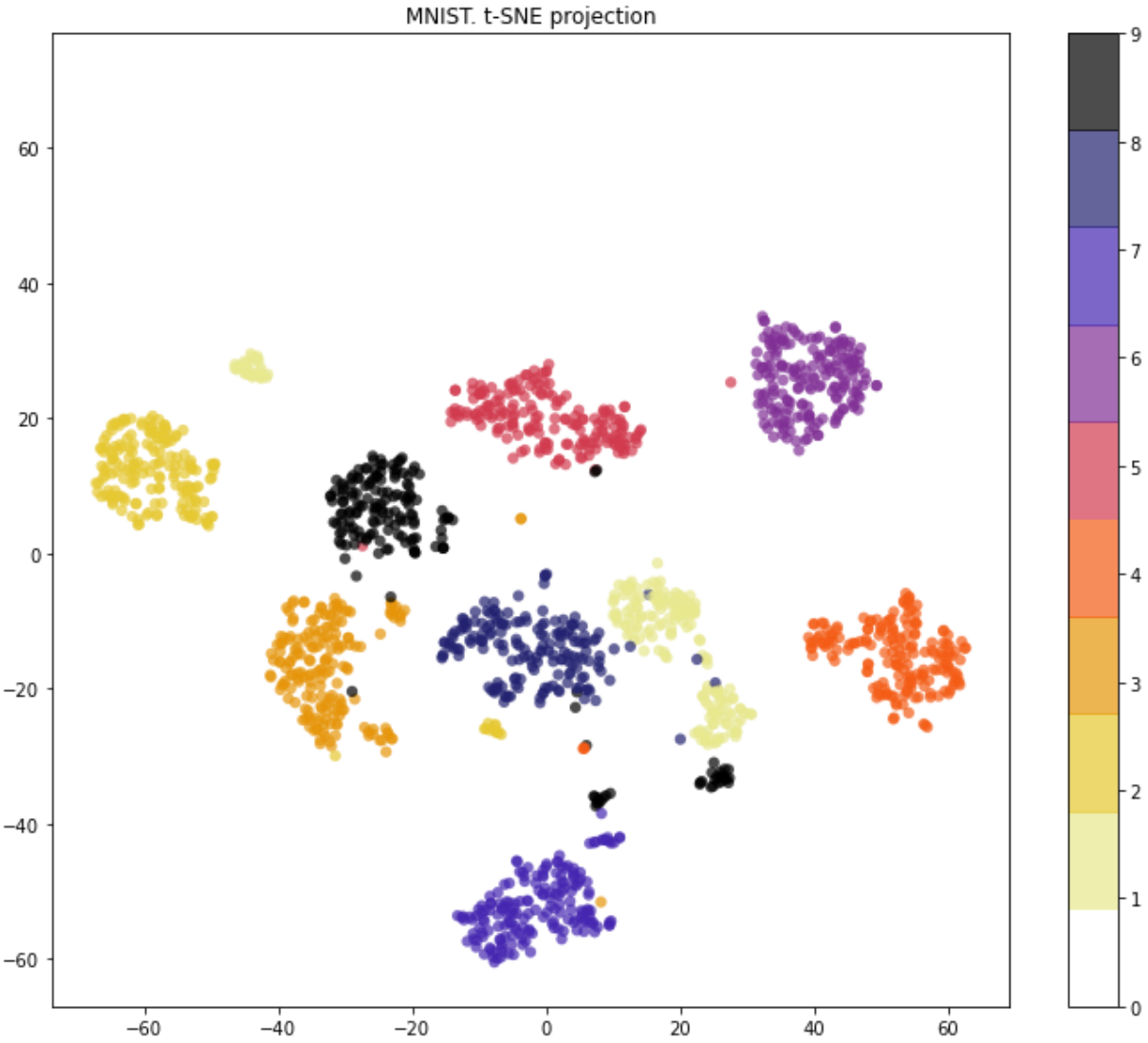
```
Projecting 64-dimensional data to 2D
```



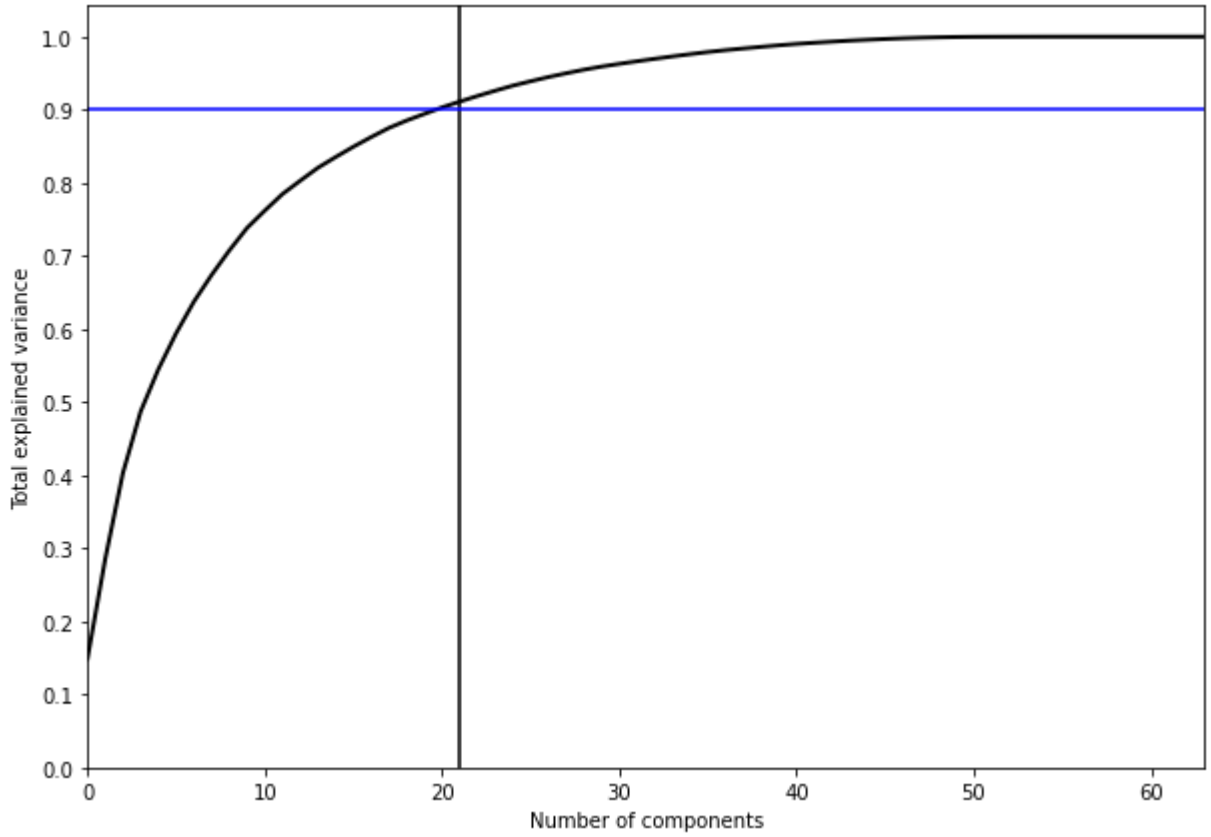
In [18]:
```
%%time

from sklearn.manifold import TSNE
tsne = TSNE(random_state=17)

X_tsne = tsne.fit_transform(X)

plt.figure(figsize=(12,10))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y,
            edgecolor='none', alpha=0.7, s=40,
            cmap=plt.cm.get_cmap('CMRmap_r', 10))
plt.colorbar()
plt.title('MNIST. t-SNE projection');
```

```
Wall time: 4.54 s
```

Out[18]:
```
Text(0.5, 1.0, 'MNIST. t-SNE projection')
```



In [19]:
```
pca = decomposition.PCA().fit(X)

plt.figure(figsize=(10,7))
plt.plot(np.cumsum(pca.explained_variance_ratio_), color='k', lw=2)
plt.xlabel('Number of components')
plt.ylabel('Total explained variance')
plt.xlim(0, 63)
plt.yticks(np.arange(0, 1.1, 0.1))
plt.axvline(21, c='k')
plt.axhline(0.9, c='b')
plt.show();
```



K-mean Clustering

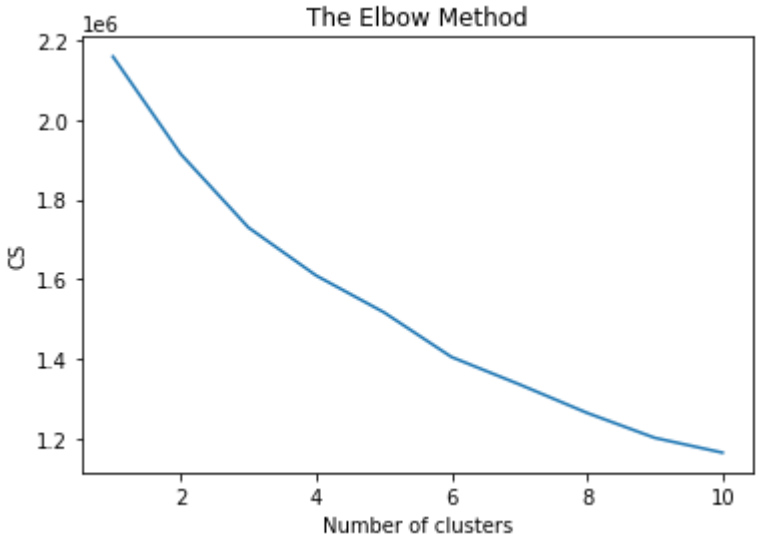
K-means algorithm is the most popular and yet simplest of all the clustering algorithms. Here is how it works:

1. Select the number of clusters k that you think is the optimal number.
2. Initialize k points as "centroids" randomly within the space of our data.
3. Attribute each observation to its closest centroid.
4. Update the centroids to the center of all the attributed set of observations.
5. Repeat steps 3 and 4 a fixed number of times or until all of the centroids are stable (i.e. no longer change in step 4).

This algorithm is easy to describe and visualize. Let's take a look.

The Elbow Method

In [20]:
```
from sklearn.cluster import KMeans
cs = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init =
    kmeans.fit(X)
    cs.append(kmeans.inertia_)
plt.plot(range(1, 11), cs)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('CS')
plt.show()
```



Silhouette method

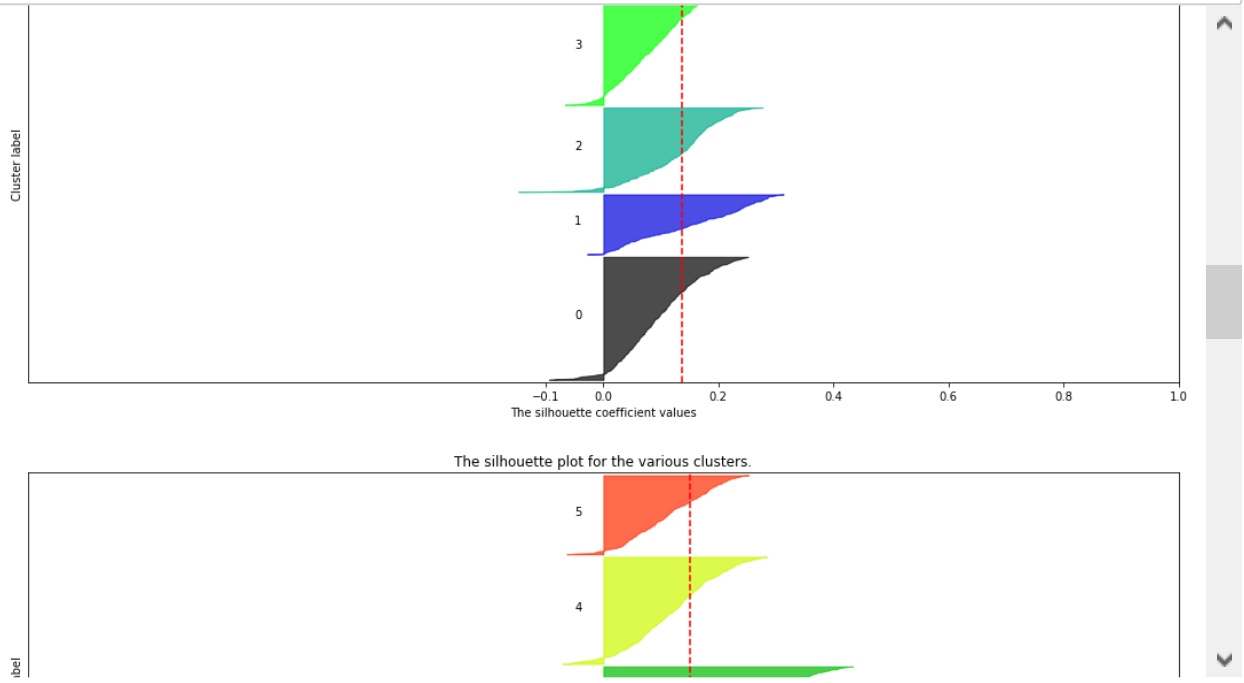
In [21]:
```
from sklearn import metrics
from sklearn import datasets
import pandas as pd
from sklearn.cluster import KMeans, AgglomerativeClustering, AffinityPropagation,


data = datasets.load_digits()
X, y = data.data, data.target

algorithms = []
algorithms.append(KMeans(n_clusters=10, random_state=1))

data = []
for algo in algorithms:
    algo.fit(X)
    data.append(({
        'ARI': metrics.adjusted_rand_score(y, algo.labels_),
        'AMI': metrics.adjusted_mutual_info_score(y, algo.labels_,
                                                  average_method='arithmetic'),
        'Homogenity': metrics.homogeneity_score(y, algo.labels_),
        'Completeness': metrics.completeness_score(y, algo.labels_),
        'V-measure': metrics.v_measure_score(y, algo.labels_),
        'Silhouette': metrics.silhouette_score(X, algo.labels_)}))

results = pd.DataFrame(data=data, columns=['ARI', 'AMI', 'Homogenity',
                                           'Completeness', 'V-measure',
                                           'Silhouette'],
                       index=['K-means'])

results
```

Out[21]:

	ARI	AMI	Homogenity	Completeness	V-measure	Silhouette
K-means	0.662295	0.736567	0.735448	0.742972	0.739191	0.182097

In [25]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import seaborn as sns

for n_clusters in range(2,11):
    fig, (ax1) = plt.subplots(1)
    fig.set_size_inches(18, 7)
    ax1.set_xlim([-1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")
```

```
    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

plt.show()
```



In [ ]: