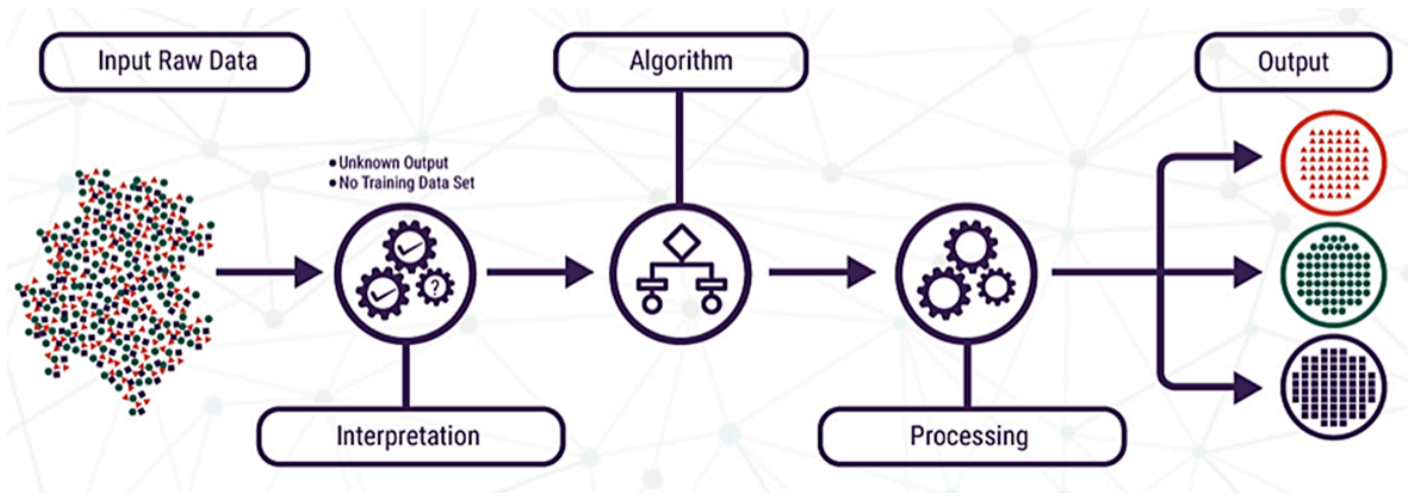


# Introduction to Unsupervised Learning

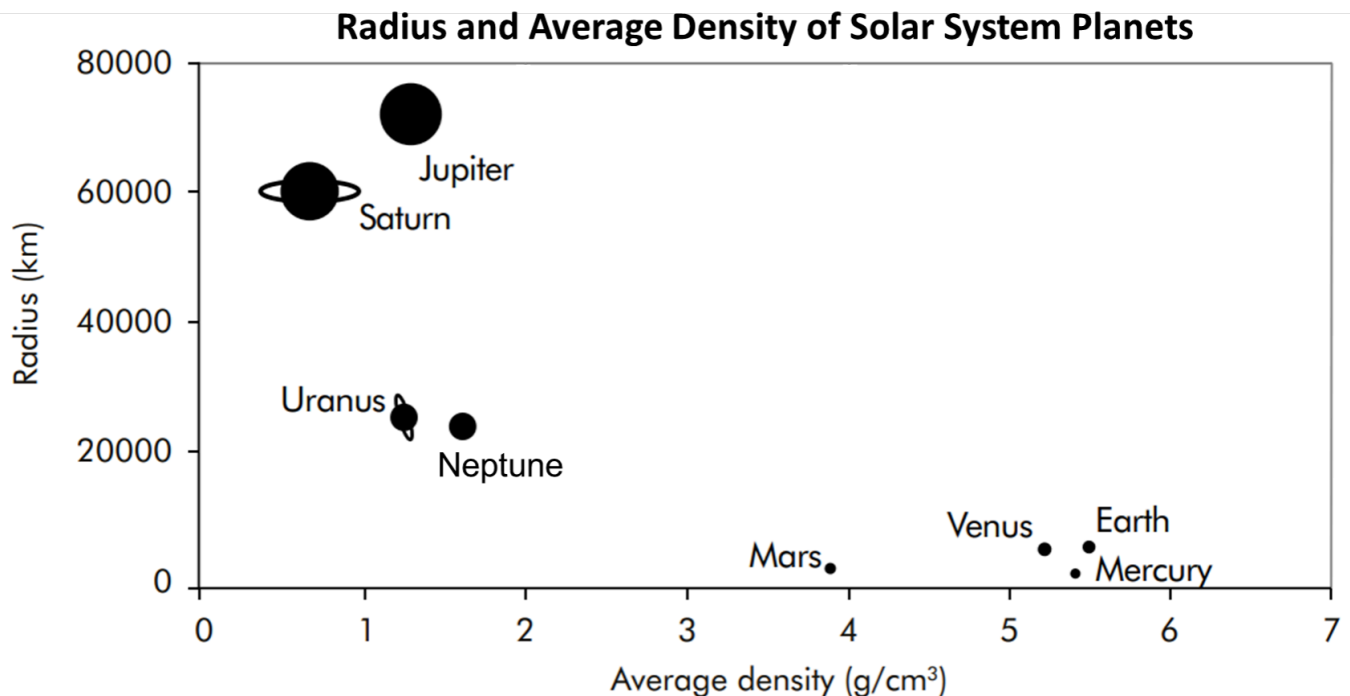
**Unsupervised learning** refers to a machine learning algorithm that infers information from a given dataset without any guidance. The algorithm relies on underlying similarities and patterns within the data to learn how the variables are related to one another. So in a way, we're "letting the data speak for itself."



This is a great tool to use for initial exploration of a dataset without any defined hypothesis. Unsupervised learning is especially valuable in assessing a large dataset with multiple variables. For today's lecture, we'll be going over two broad applications of unsupervised learning: (1) **clustering** and (2) **dimensionality reduction**.

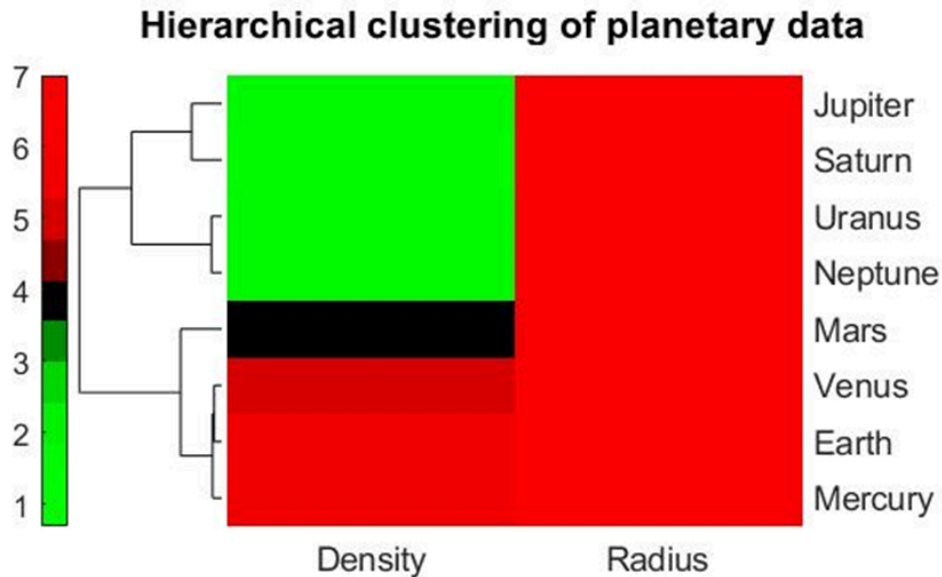
## Clustering - Identifying distinct groups based on similarity

We'll be covering two types of clustering techniques: **hierarchical** and **k-means** clustering. To introduce these methods, we'll consider the following planetary data:



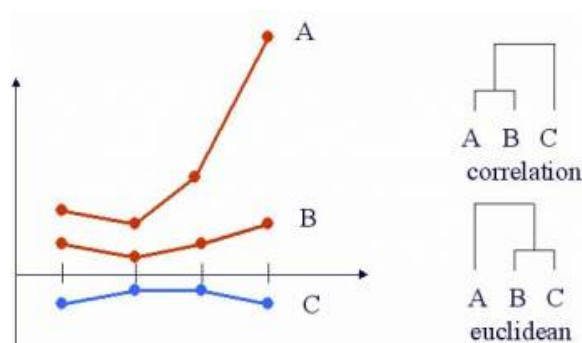
## Hierarchical Clustering

In **hierarchical clustering**, each observation starts in its own cluster at the base of the hierarchy. As we move up the hierarchy, observations that are most similar group together first until all observations are linked. This hierarchy of observations is often depicted as a **dendrogram**. In MATLAB, we can also plot a **clustergram** to visualize how similar the observations (e.g. planets) are based on different measurements (e.g. density, radius). The plot below shows an example of hierarchical clustering of planets in the solar system based on their average density and radius:



## Defining Similarity

One of the most common ways to apply unsupervised learning is to assess the similarity within a dataset. Similarity can be quantitatively described using a **distance metric** between two variables. There are various types of distance metrics that can be used, but we'll just go over two of the most common ones: **Euclidean** and **correlation** distance. The plot below shows how these distance metrics differ:

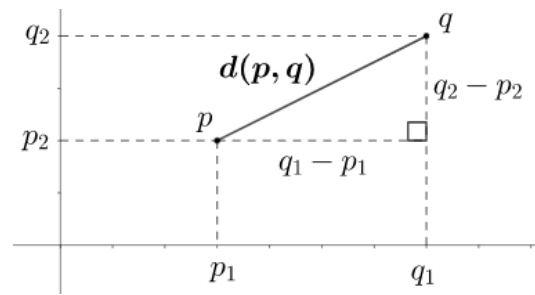


## Euclidean Distance

The **Euclidean distance** is broadly defined by the following formula:

$$d(p, q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

This is what the distance between  $p$  and  $q$  would look like when  $n = 2$ :



When analyzing biological data in MATLAB, we'll mostly be dealing with large vectors. In these cases, we let  $n$  be equal to the length of the vectors we're comparing (note: the vectors must have the same length).

For instance, let's say we're comparing vectors  $a$  and  $b$  as defined below:

```
load('distance_data.mat')
a
b
```

For this example,  $n = 4$ . We can calculate the Euclidean distance manually:

```
d_euclid_man = sqrt(sum((a-b).^2))
```

We can also use the built-in function *pdist*:

```
d_euclid_fun = pdist([a;b], 'euclidean')
```

Note that when using *pdist*, we combined vectors  $a$  and  $b$  into a matrix. This is because the *pdist* function is used to calculate the distance between all vectors in a matrix. If we had a matrix with three vectors, we would be calculating the distance for three pairwise combinations ( $C(3,2) = 3$ ):

```
c = [-1 -0.5 -0.5 -1];
d_euclid_3 = pdist([a;b;c])
```

Note: Euclidean distance is the default metric for the *pdist* function.

## Correlation Distance

Another commonly used distance metric is **correlation**, and this is broadly defined as follows:

$$d(p, q) = 1 - r$$

where  $r$  is the correlation coefficient between two vectors.

Going back to our example vectors  $a$  and  $b$ , the correlation distance can be calculated manually:

```
d_corr_man = 1-corr(a',b')
```

Or using the *pdist* function:

```
d_corr_fun = pdist([a;b], 'correlation')
```

Let's calculate the distances between vectors a, b, and c based on correlation:

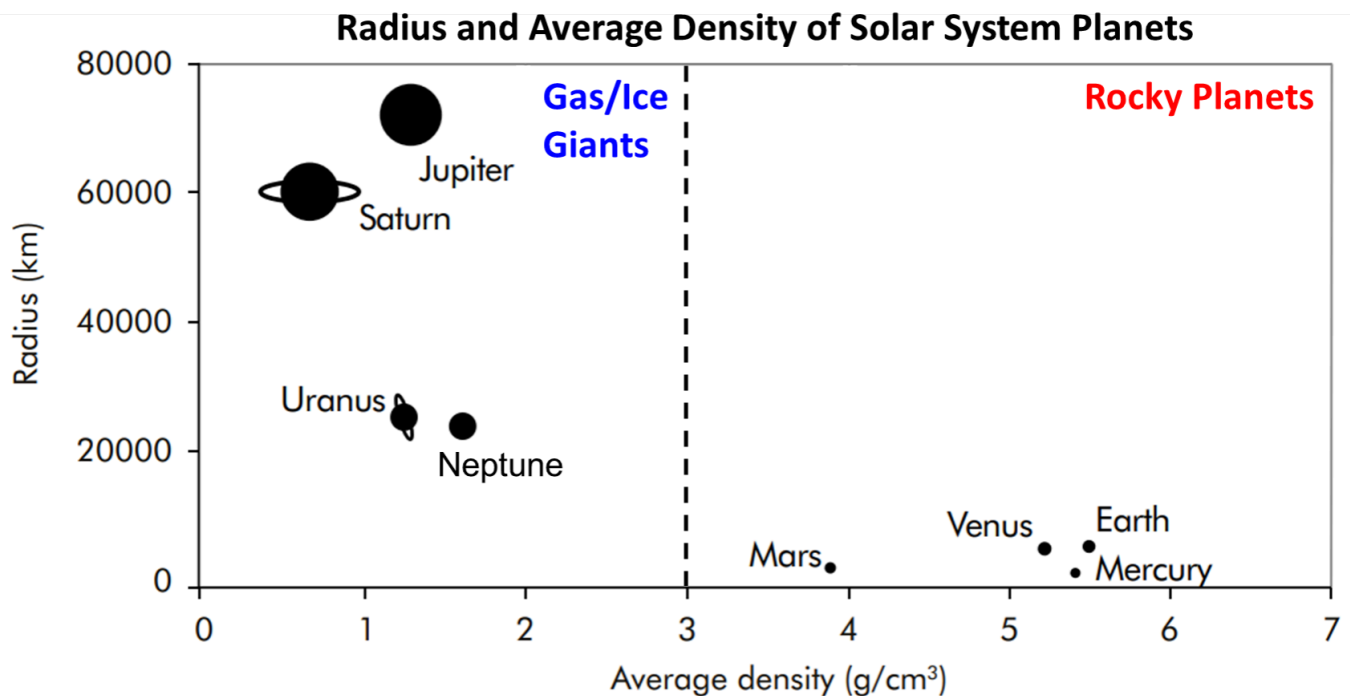
```
d_corr_3 = pdist([a;b;c], 'correlation')
```

Note that the correlation distance can only range from 0 (similar) to 2 (dissimilar).

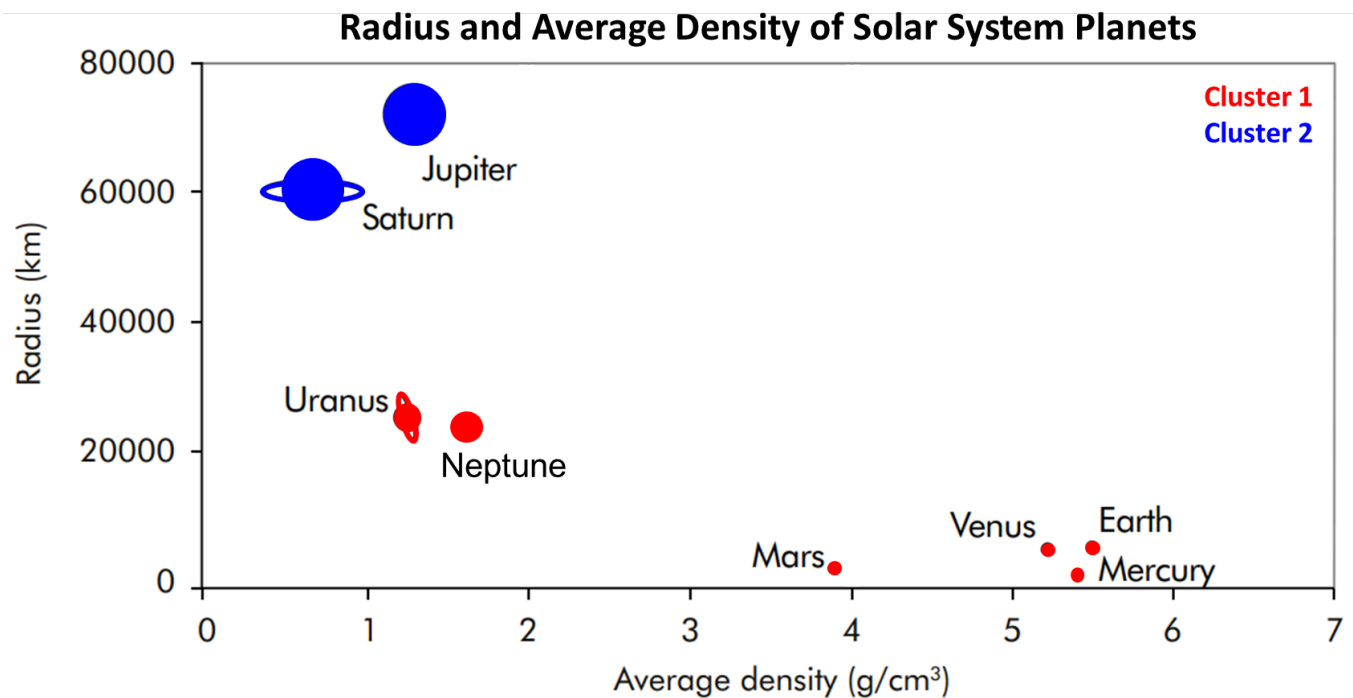
Additional information: refer to the *pdist* documentation for all available distance metrics.

## K-means Clustering

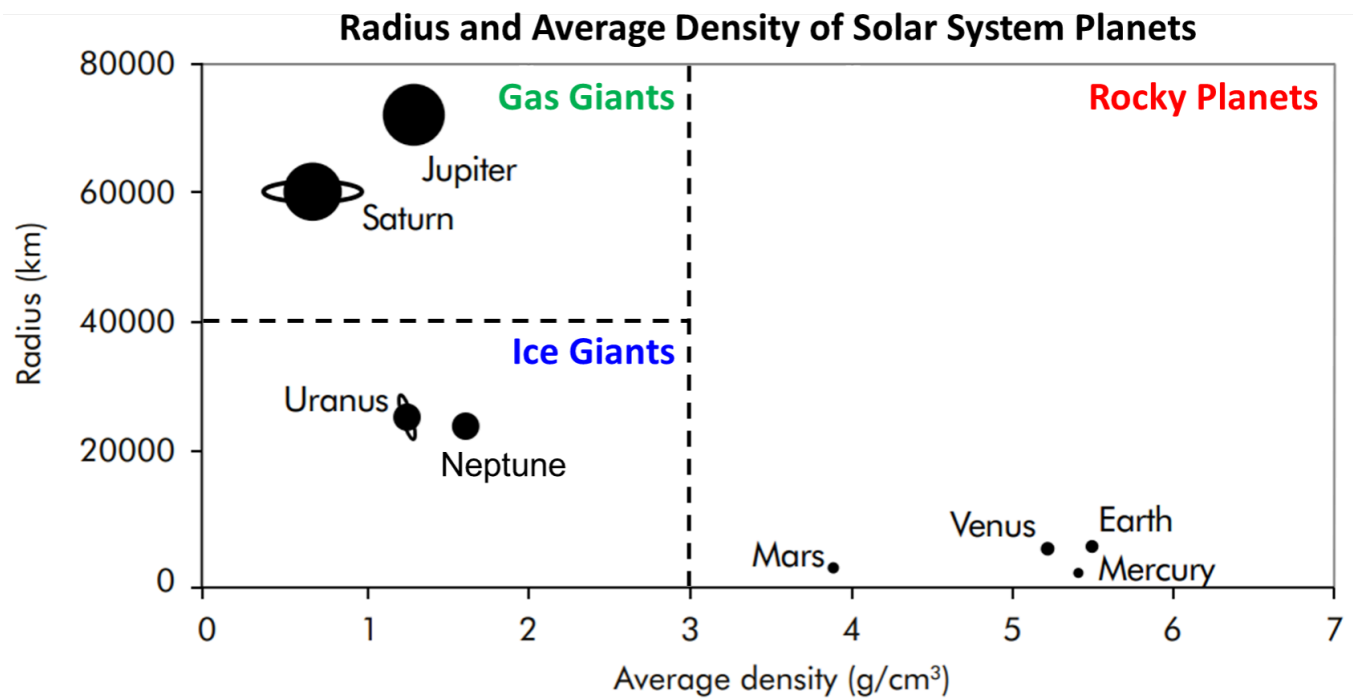
For **k-means clustering**, we specify how many distinct groups we think that the data will cluster into. K-means clustering then allocates data points to a cluster by minimizing the in-cluster sum of squares (more on this later). The value for k is typically chosen based on *a priori* information we may have on the data. Referring back to our planetary data:



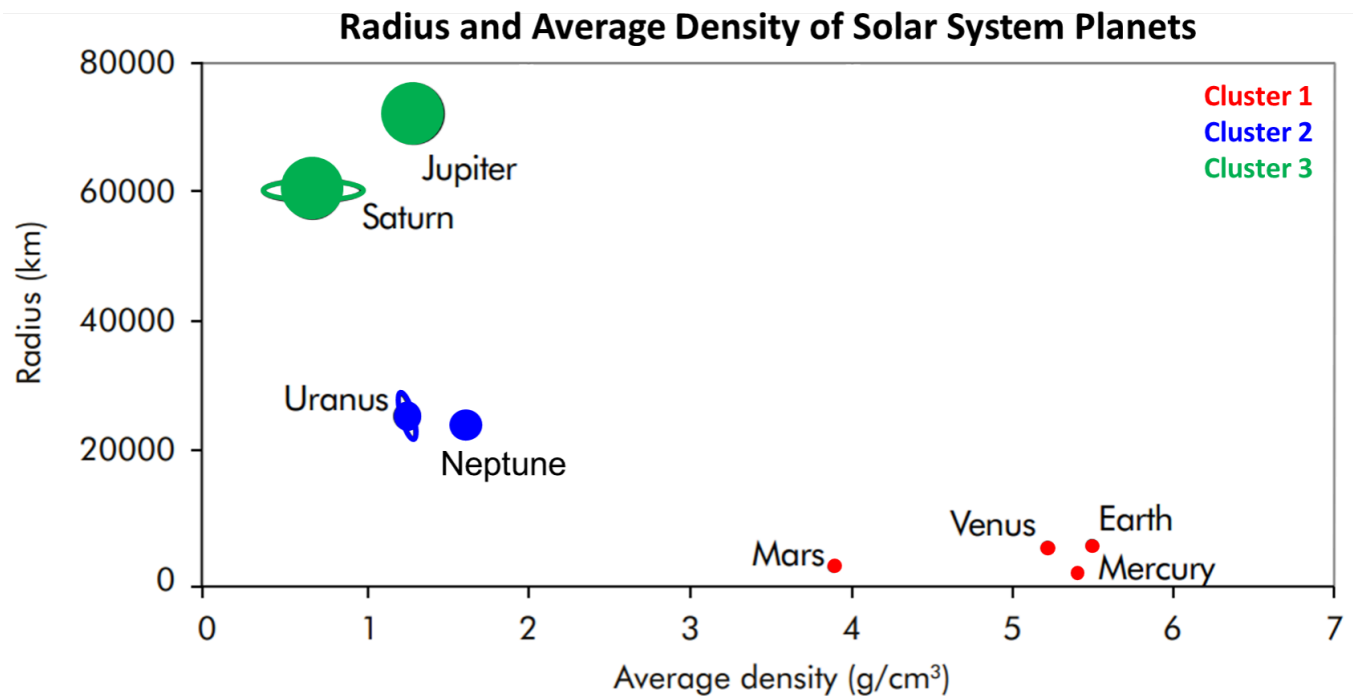
Based on this additional information, we could apply k-means clustering with  $k = 2$  to group our planets:



This is probably not how we expected the planets to cluster based on our reason to use  $k = 2$ . However, if we delve deeper into the information we were given before:



Let's see how the planets cluster based on k-means clustering with  $k = 3$ :



Now our results match what we expected based on our reason to use  $k = 3$ .

## Silhouette Analysis

You must be wondering: what if we don't have any prior knowledge of the data? What value do we use for  $k$ , and how do we know if this was a good choice? To consider these questions, we can conduct a **silhouette analysis** to determine which  $k$  value best minimizes the in-cluster sum of squares. For this analysis, we calculate the silhouette value for each observation and assess how they compare within a cluster by generating a **silhouette plot**. The silhouette value measures how similar an observation is to its own cluster compared to others based on a distance metric. This value ranges from -1 to 1, where a higher positive value implies that an observation is well-matched to its cluster and vice-versa.

One way to assess whether we chose a good value for  $k$  is to evaluate the mean of all silhouette values. For the purposes of this lesson, we'll refer to the mean of all silhouette values as the **silhouette score**. In general, a silhouette score above 0.5 indicates that the data is well separated. The following code calculates the silhouette statistic for our planetary data based on  $k$ -means clustering with different  $k$  values and visualizes the clustered data:

```
% Calculate silhouette statistic
load('planetary_data.mat')
disp(planet_table)

% load planetary data
% display data table
```

| Planet  | Density | Radius |
|---------|---------|--------|
| Mercury | 5.427   | 2439.5 |
| Venus   | 5.243   | 6052   |
| Earth   | 5.514   | 6378   |
| Mars    | 3.933   | 3396   |
| Jupiter | 1.326   | 71492  |
| Saturn  | 0.687   | 60268  |
| Uranus  | 1.271   | 25559  |
| Neptune | 1.638   | 24764  |

```

X = [density radius]; % array of planetary data
[idx,~] = kmeans(X,3); % k-means clustering
s = silhouette(X,idx); % silhouette values
s_score = mean(s) % silhouette score

```

```
s_score = 0.9705
```

```

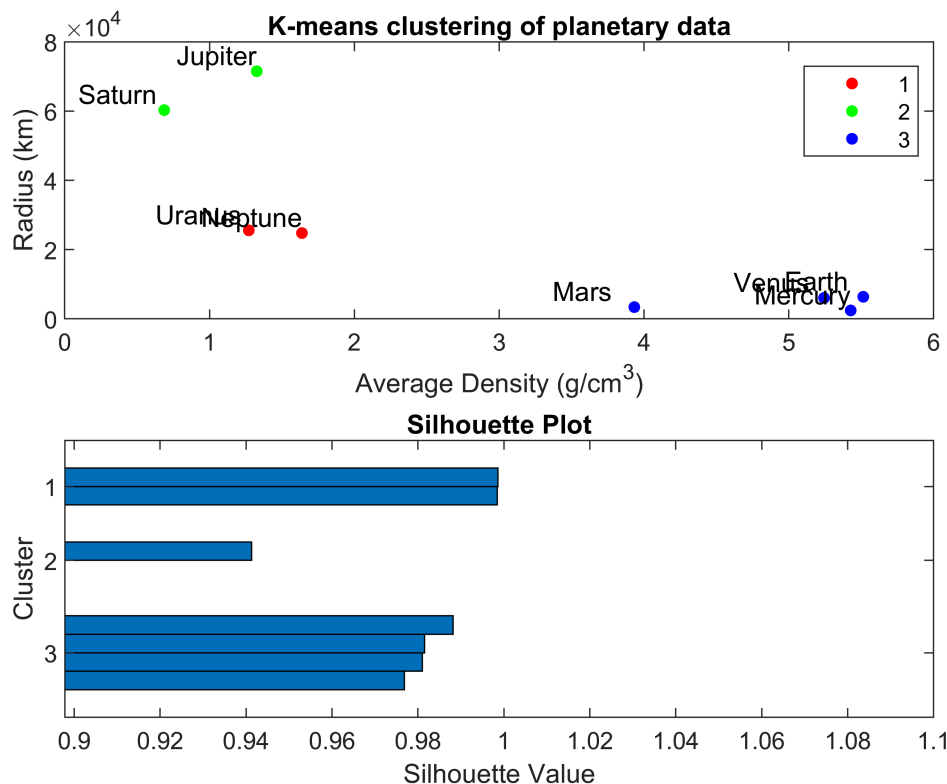
% Plot of clustered data
subplot(2,1,1)
gscatter(density, radius, idx)
labelpoints(density, radius, planet)
xlabel('Average Density (g/cm^3)')
ylabel('Radius (km)')
title('K-means clustering of planetary data')

```

```

% Silhouette plot
subplot(2,1,2)
silhouette(X,idx)
title('Silhouette Plot')

```



## Hierarchical vs. K-means Clustering

Both hierarchical clustering and k-means clustering are unsupervised learning methods for grouping data based on similarity. The difference between these two clustering methods is that hierarchical clustering takes a "bottom-up" approach, whereas k-means clustering takes a "top-down" approach.

In MATLAB, we can perform hierarchical clustering using the *clustergram* function, and k-means clustering using the *kmeans* function. The *silhouette* function is used to calculate the silhouette value for each observation based on its cluster assignment from k-means clustering. This function also generates a silhouette plot.

## In-class practice: clustering of the Framingham data

### Applying hierarchical clustering for cardiovascular health measurements

Let's say we're interested in what the subset of cardiovascular-related measurements for a small set of patients can tell us about a their diabetes status using hierarchical clustering. The first step is to load the data and define the subset we're interested in:

```
fram = readtable('frmgham2.xls');           % load the Framingham data
disp(fram.Properties.VariableNames)         % see what variables the dataset contains
```

```
Columns 1 through 7
'RANDID'    'SEX'    'TOTCHOL'    'AGE'    'SYSBP'    'DIABP'    'CURSMOKE'
Columns 8 through 13
'CIGPDAY'    'BMI'    'DIABETES'    'BPMEDS'    'HEARTRTE'    'GLUCOSE'
Columns 14 through 19
'educ'    'PREVCHD'    'PREVAP'    'PREVMI'    'PREVSTRK'    'PREVHYP'
Columns 20 through 26
'TIME'    'PERIOD'    'HDLC'    'LDLC'    'DEATH'    'ANGINA'    'HOSPMI'
Columns 27 through 32
'MI_FCHD'    'ANYCHD'    'STROKE'    'CVD'    'HYPERTEN'    'TIMEAP'
Columns 33 through 38
'TIMEMI'    'TIMEMIFC'    'TIMECHD'    'TIMESTRK'    'TIMECVD'    'TIMEDTH'
Column 39
'TIMEHYP'
```

```
cardio_vars = {'DIABETES','TOTCHOL','SYSBP','DIABP','BMI',...
               'HEARTRTE','GLUCOSE','HDLC','LDLC'} % define cardiovascular-related variables (+ diabetes)
```

```
cardio_vars = 1x9 cell array
'DIABETES'    'TOTCHOL'    'SYSBP'    'DIABP'    'BMI'    'HEARTRTE'    ...
```

```
fram_cardio = fram(:,cardio_vars);           % create variable for data of interest
```

We'll remove rows containing NaN values and get the subset for the first 50 entries. We'll also create a variable for the patient diabetes status and a numerical array for the cardiovascular-related measurements:

```
fram_cardio = rmmissing(fram_cardio);        % remove rows containing NaN values
fram_cardio = fram_cardio(1:50,:);           % subset of 50 entries
diabetes_status = repmat({'Healthy'},50,1);   % patient diabetes status (0 = healthy)
diabetes_status(fram_cardio.DIABETES == 1) = {'Diabetic'}; % patient diabetes status (1 = diabetic)
fram_cardio_data = table2array(fram_cardio(:,2:end)); % numerical data array (- diabetes status)
```

Now we can perform hierarchical clustering for this data subset. For this example, we'll use Euclidean distance (default) as our distance metric for assessing similarity. We'll also cluster along both rows and columns (default) and provide labels (rows = patient diabetes status, columns = measurements). Finally, we'll standardize each column to better compare measurements between patients:

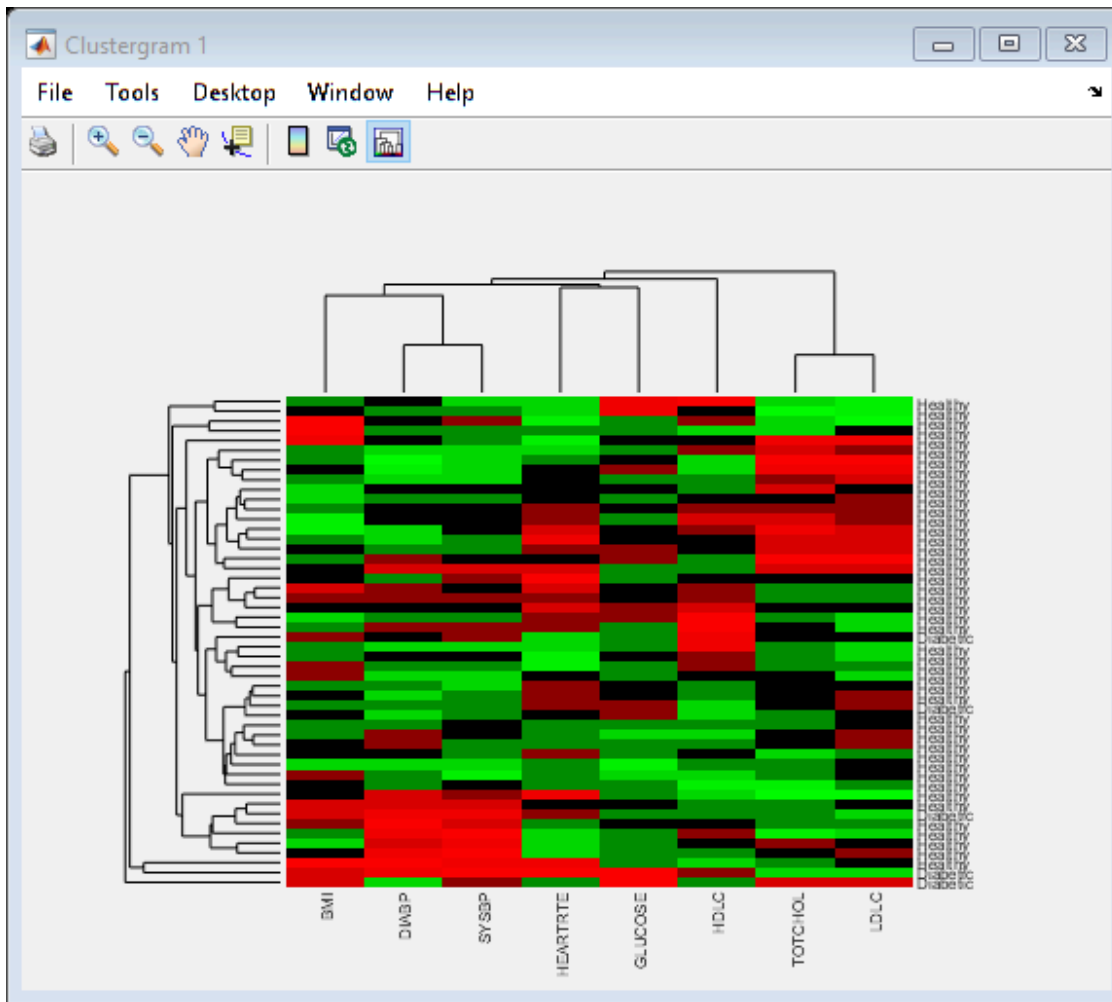
```
col_names = fram_cardio.Properties.VariableNames(2:end); % variable for column names (- diabetes status)
```



```

fram_cardio_cg = clustergram(fram_cardio_data,... % clustergram with...
    'RowLabels',diabetes_status,... % labeled rows (patient diabetes status)
    'ColumnLabels',col_names,... % labeled columns (measurements)
    'Standardize','column') % standardized columns

```



Clustergram object with 50 rows of nodes and 8 columns of nodes.

This clustergram shows how specific measurements compare between patient clusters, and which patients are most similar based on Euclidean distance.

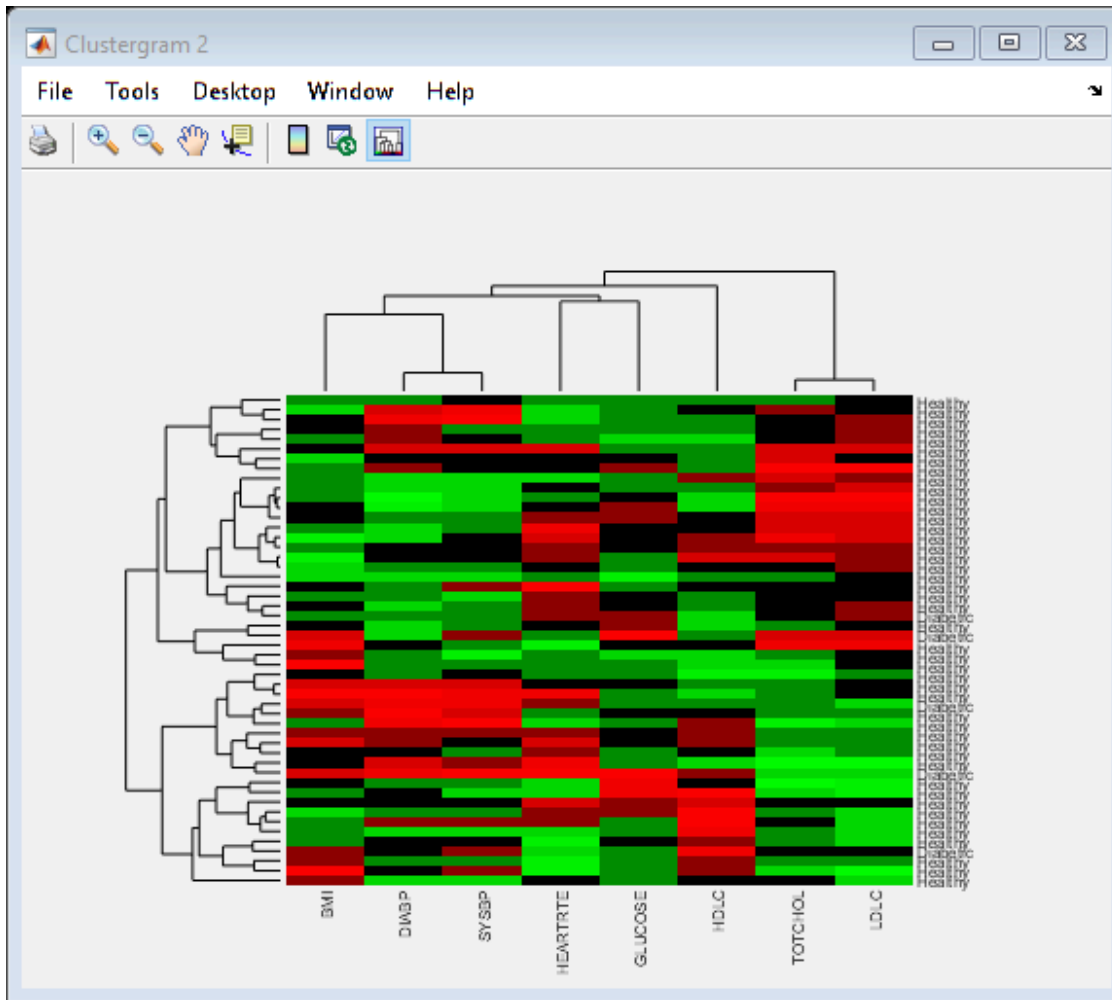
Question: Are diabetic patients similar based on these measurements?

Let's see how the clustergram changes if we use Pearson's correlation as our distance metric:

```

fram_cardio_cg_corr = clustergram(fram_cardio_data,... % clustergram with...
    'RowLabels',diabetes_status,... % labeled rows (patient diabetes status)
    'ColumnLabels',col_names,... % labeled columns (measurements)
    'Standardize','column',... % standardized columns
    'RowPDist','correlation',... % correlation as distance metric for rows
    'ColumnPDist','correlation') % correlation as distance metric for columns

```

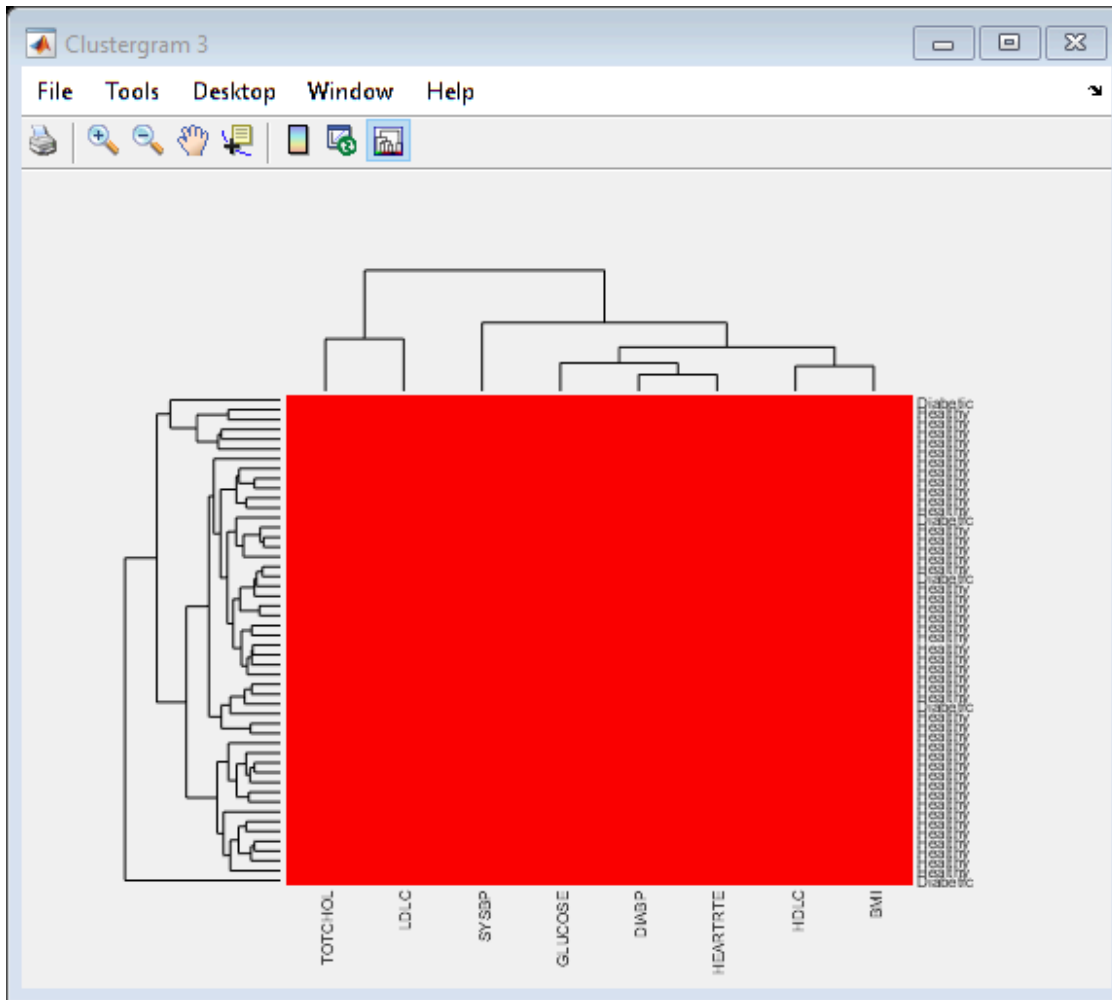


Clustergram object with 50 rows of nodes and 8 columns of nodes.

Question: How did the clustering change when using correlation as our distance metric? Are diabetic patients similar?

Note: Notice that we made a point to standardize each measurement (column) in our dataset, as this ensures uniformity when comparing different measurements. Below is an example of what the clustergram would look like if we used the data as is:

```
fram_cardio_cg_raw = clustergram(fram_cardio_data,... % clustergram with...
  'RowLabels',diabetes_status,... % labeled rows (patient diabetes status)
  'ColumnLabels',col_names) % labeled columns (measurements)
```



Clustergram object with 50 rows of nodes and 8 columns of nodes.

We can see that this clustergram is not very informative, as each column represents a different measurement with its own range of values.

## Applying k-means clustering for the Framingham data

To more easily visualize k-means clustering, we'll consider how the previous set of patients cluster solely based on BMI and blood glucose level:

```
fram_kmeans = fram_cardio(:,{'DIABETES', 'BMI', 'GLUCOSE'}); % define dataset to use
```

Let's see if we're able to pick out diabetic patients from our data using  $k = 2$ :

```
X = table2array(fram_kmeans); % numerical array of dataset
[idx,~] = kmeans(X,3); % k-means clustering (idx = cluster identity)
s = silhouette(X,idx); % vector for silhouette values
s_score = mean(s) % silhouette score
```

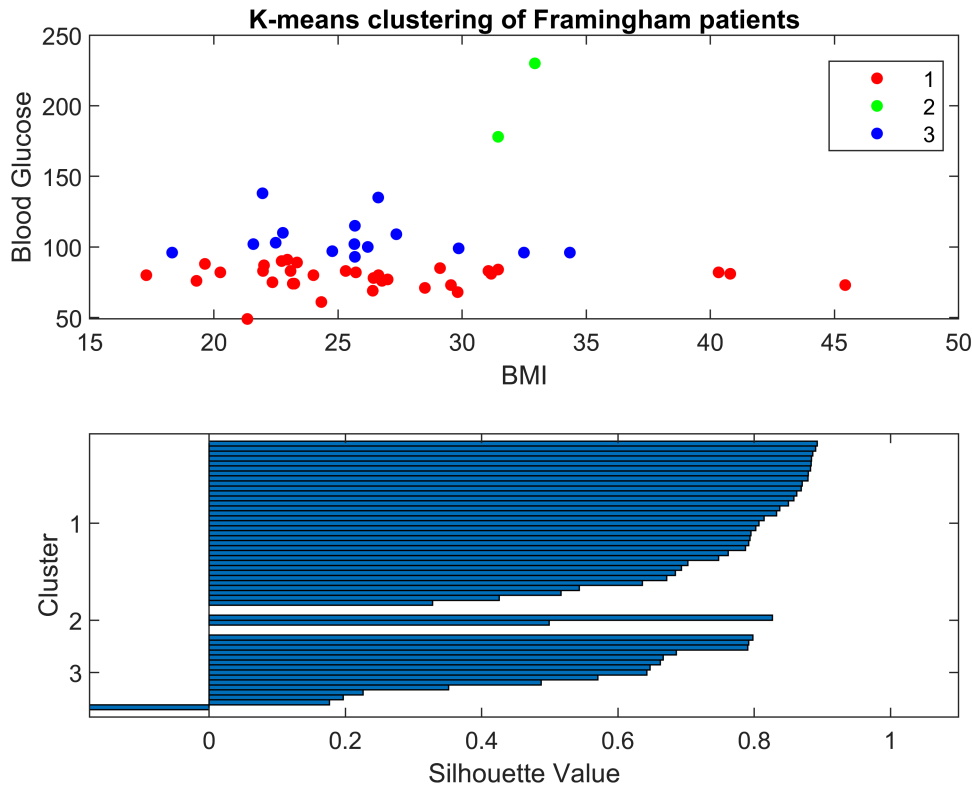
```
s_score = 0.6840
```

Let's visualize a scatter plot of our data clustered into 2 groups. We'll also look at the corresponding silhouette plot:

```

subplot(2,1,1)
gscatter(fram_kmeans.BMI,fram_kmeans.GLUCOSE,idx); % scatter plot with cluster annotation
xlabel('BMI'); ylabel('Blood Glucose') % add axis labels
title('K-means clustering of Framingham patients') % add title
subplot(2,1,2)
silhouette(X,idx) % silhouette plot

```



Question: Does one of our clusters correspond to diabetic patients?

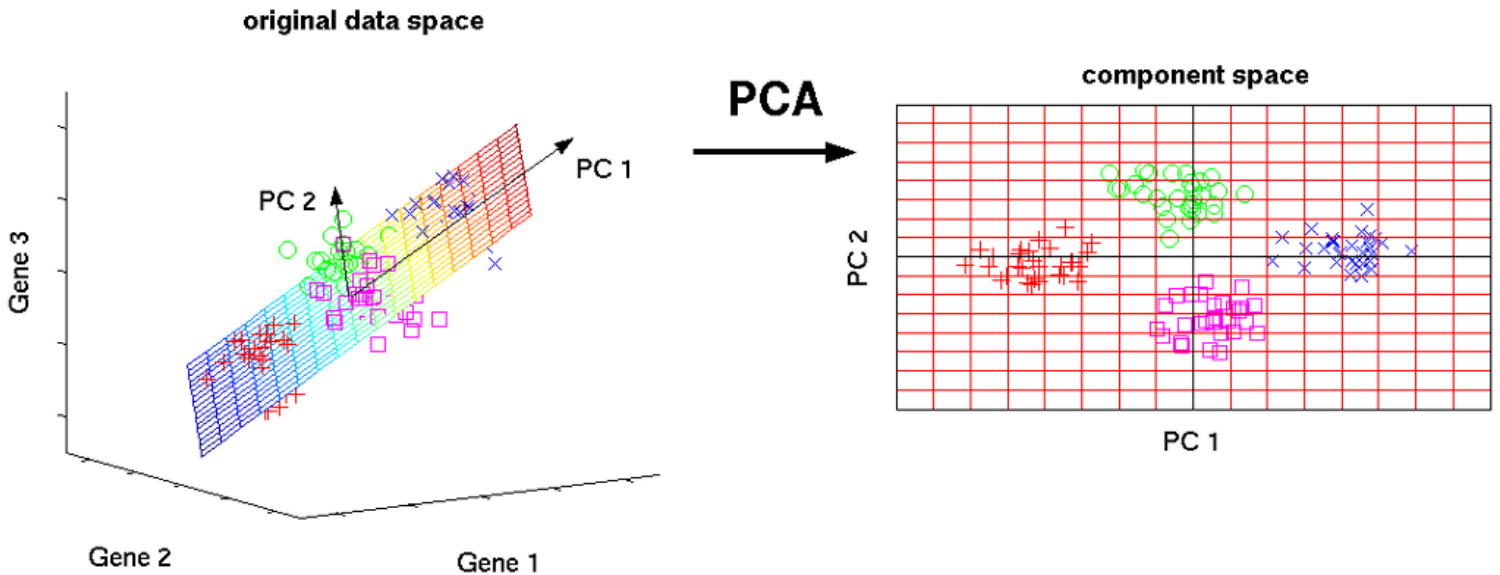
Activity: How can we modify the code above to apply k-means clustering for  $k = 3$ ? How does the  $s\_score$  value compare between  $k = 3$  and  $k = 2$ ?

## Dimensionality Reduction - Representing the variability in the dataset in lower dimensions

Biomedical data is often tremendously large and highly complex. Thus, it's hard to interpret this data when considering all the information available. By applying **dimensionality reduction**, we can transform the data into a lower-dimension representation that is much easier to interpret. In this lesson, we'll go over one type of dimensionality reduction method known as **principal component analysis**, or **PCA**.

### Principal Component Analysis

In PCA, an orthogonal linear transformation is applied to the data, yielding a new coordinate system defined by principal components.

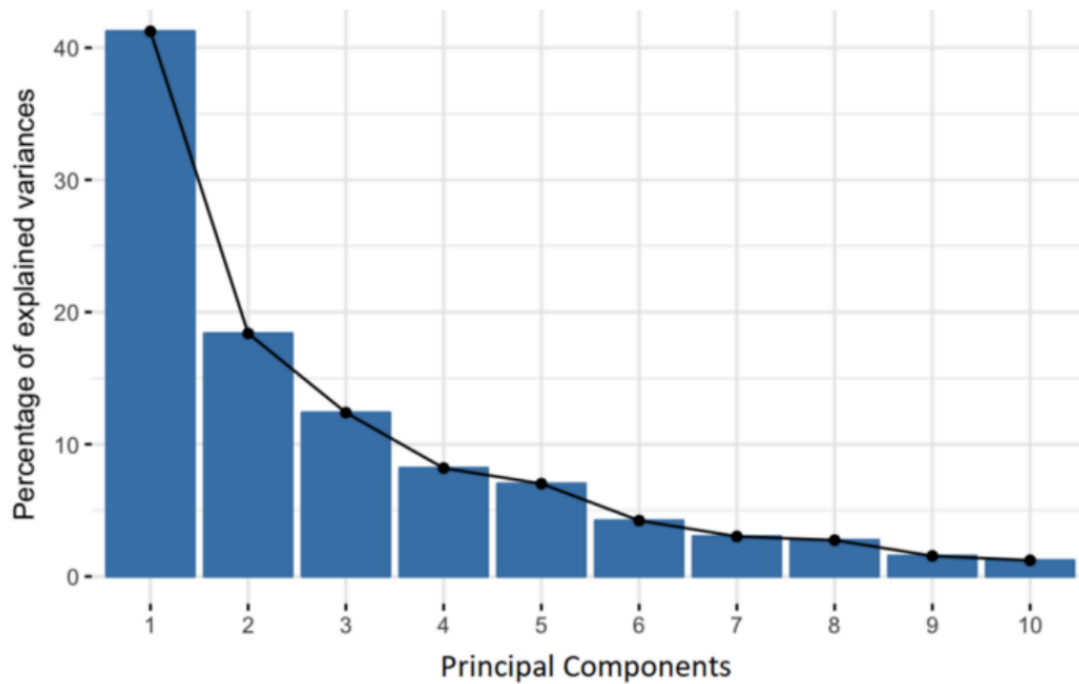


A **principal component (PC)** is a linear combination of the **features**, or descriptive variables, in the dataset. Note that the number of features dictates the number of PCs that are determined. In the image below, **P** represents the matrix of principal components that is used to linearly transform the original data (**X**) into a new representative data (**Y**) that can be visualized in a component space:

$$\underset{(m \times m)}{\mathbf{P}} \underset{(m \times n)}{\mathbf{X}} = \underset{(m \times n)}{\mathbf{Y}}$$

$$\mathbf{PX} = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_m \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1 \cdot \mathbf{x}_1 & \cdots & \mathbf{p}_1 \cdot \mathbf{x}_n \\ \vdots & \ddots & \vdots \\ \mathbf{p}_m \cdot \mathbf{x}_1 & \cdots & \mathbf{p}_m \cdot \mathbf{x}_n \end{bmatrix}$$

The main goal when conducting PCA is to determine a ranked list of linearly independent principal components that altogether describe the variance within the data. This list of PCs is ranked by decreasing percentage of variance explained, where the first PC explains the most variance, or is best representative, of the data. The figure below shows an example of how much variance each PC explains for a dataset with 10 features:



For the example shown above, we are able to represent ~60% of the variance using the first two PCs. In other words, if we were to plot the data based on PC1 and PC2, we would be visualizing a two-dimensional representation of the data that explains ~60% of the variance.

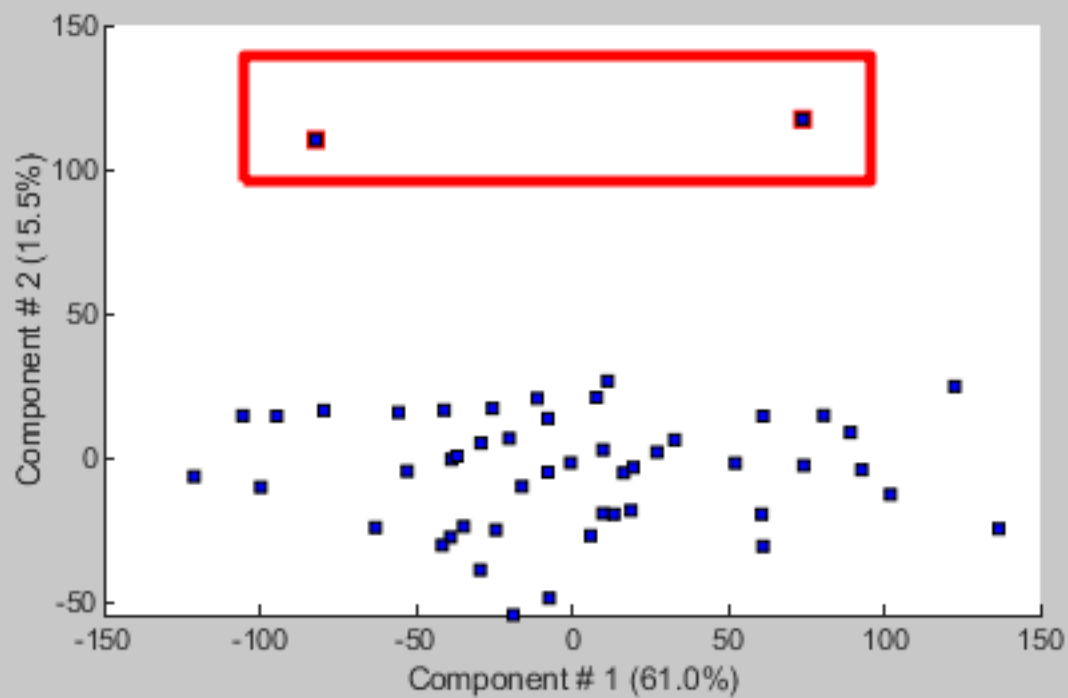
PCA is typically done by (1) **eigenvalue decomposition (EVD)** of the data covariance matrix, or (2) **singular vector decomposition (SVD)** of the data. For this lesson, we'll not delve into the specifics of how EVD or SVD is applied (for more information, refer to [this link](#) for EVD and [this link](#) for SVD).

In MATLAB, the *mapcaplot* function is used to perform PCA based on SVD. The *mapcaplot* function generates a user interface to visualize the data on the component space.

### Example: applying PCA for the Framingham data

For this analysis, we'll consider the same dataset of cardiovascular measurements we used for hierarchical clustering. Recall that this dataset contains 8 measurements, or features. This means that we'll be determining 8 linearly independent PCs by performing PCA:

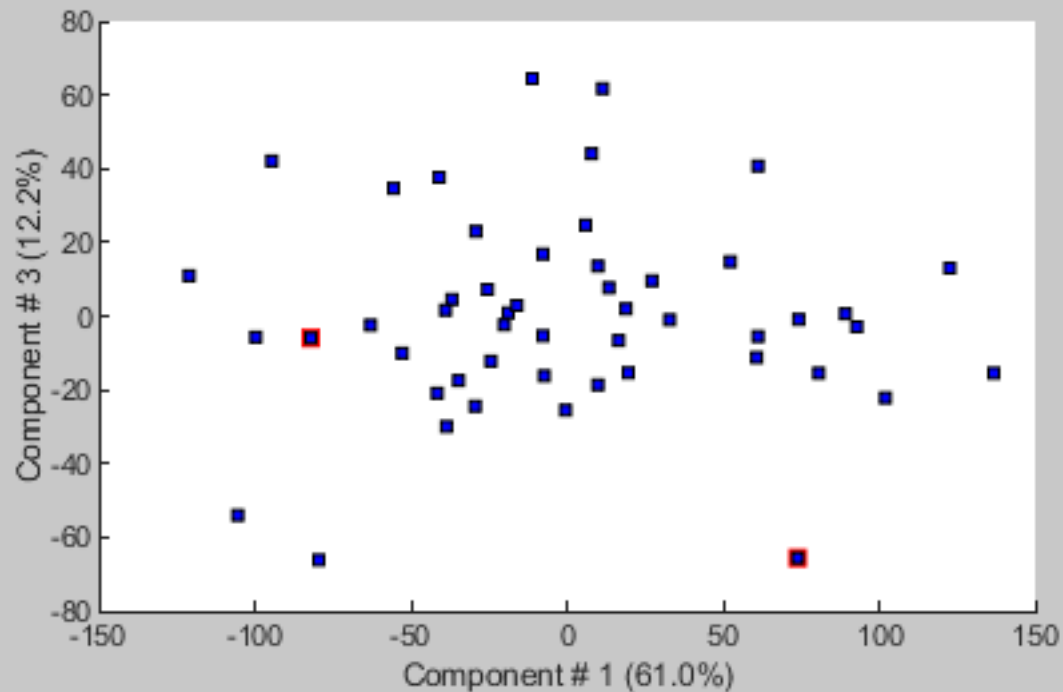
```
[coeff,score] = pca(fram_cardio_data);           % perform PCA
mapcaplot(fram_cardio_data,diabetes_status)      % visualize data in component space
```



Component # 1 (61.0%) ▾

vs

Component # 2 (15.5%) ▾



Component # 1 (61.0%) ▾

vs

Component # 3 (12.2%) ▾

Selected Data

Diabetic  
Diabetic

Export

Question: What percentage of variance do PCs 1 and 2 explain?

We can see that by performing PCA, we were able to separate 2 patients that are diabetic. Let's check how many diabetic patients there were in total:

```
sum(fram_cardio.DIABETES==1) % total number of diabetic patients
```

```
ans = 5
```

Using PCA, we were able to distinguish 2 out of 5 diabetic patients in our dataset.

We can also check the coefficient matrix (*coeff*) to see which features contribute the most to PC 2.

```
pc2_coeff = coeff(:,2); % coefficient values for PC 2
feature = col_names'; % feature names (measurements)
table(feature, pc2_coeff) % table of feature matched to its coefficient value
```

```
ans = 8x2 table
```

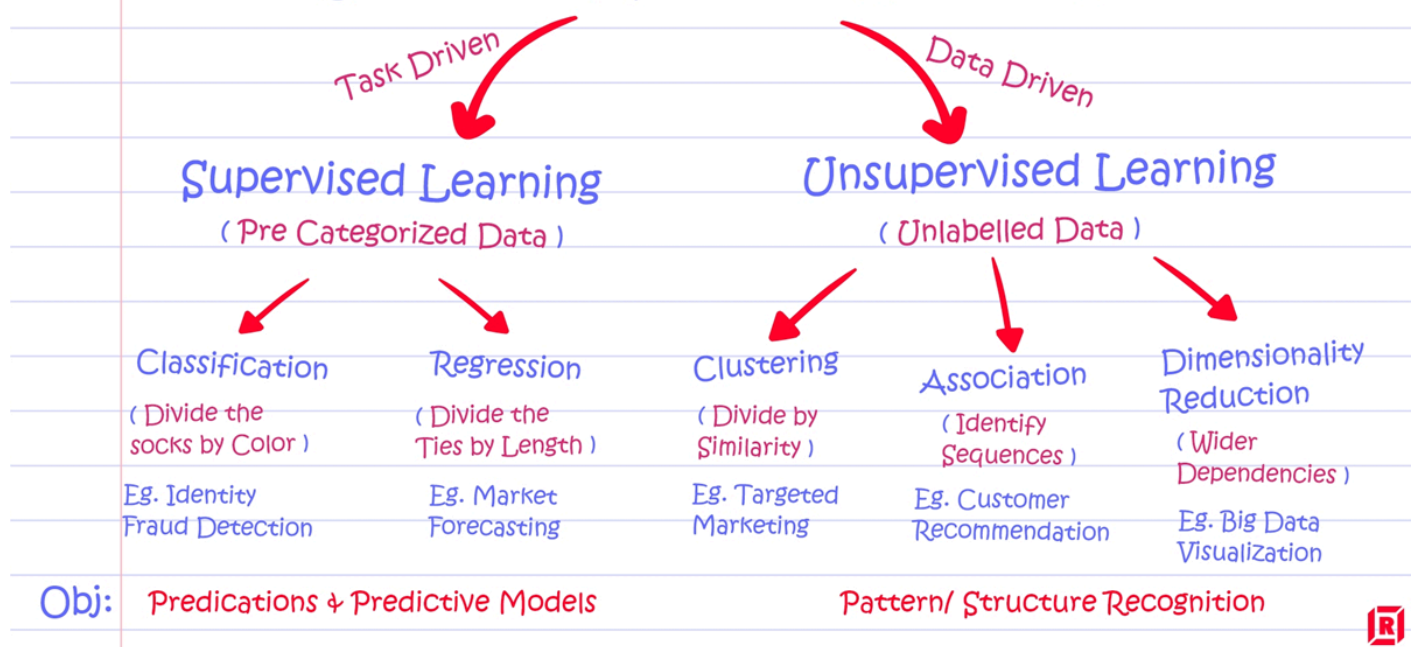
|   | feature    | pc2_coeff |
|---|------------|-----------|
| 1 | 'GLUCOSE'  | 0.7884    |
| 2 | 'SYSBP'    | 0.5481    |
| 3 | 'DIABP'    | 0.2087    |
| 4 | 'HEARTRTE' | 0.1386    |
| 5 | 'HDLC'     | 0.0858    |
| 6 | 'TOTCHOL'  | 0.0815    |
| 7 | 'BMI'      | 0.0349    |
| 8 | 'LDLC'     | 0.0039    |

## Unsupervised vs. Supervised Learning

All of the methods discussed in this lesson serve as applications for unsupervised learning. In machine learning, we can also process data using **supervised learning**, where information on the output data (e.g. labels) is considered during the learning process. While unsupervised learning is used for data exploration, supervised learning is typically used to train and develop predictive or classification tools based on labeled data.



# Classical Machine Learning



## Summary

- Unsupervised learning allows us to infer information about a dataset without any user guidance based on underlying similarities and patterns
- Common applications of unsupervised learning to analyze biomedical data include clustering (hierarchical, k-means) and dimensionality reduction (principal component analysis)
- Hierarchical clustering involves a "bottom-up" approach, whereas k-means clustering applies a "top-down" approach, to assign data into groups
- Principal component analysis (PCA) allows us to transform highly complex data into lower-dimension representations that are easier to interpret
- In contrast to unsupervised learning, supervised learning includes information on the output variables (e.g. labels) that can be used to train and develop predictive or classification tools

## Built-in MATLAB functions covered in this lesson:

- *pdist*: quantifies similarity using a distance metric (e.g. Euclidean, correlation)
- *clustergram*: performs hierarchical clustering
- *kmeans*: performs k-means clustering
- *silhouette*: calculates silhouette values and generates a silhouette plot for k-means clustering
- *pca*: performs principal component analysis (PCA)
- *mapcaplot*: visualizes data reduced into component space

## References for additional information:

- [Eigenvalue Decomposition \(EVD\)](#)
- [Singular Vector Decomposition \(SVD\)](#)