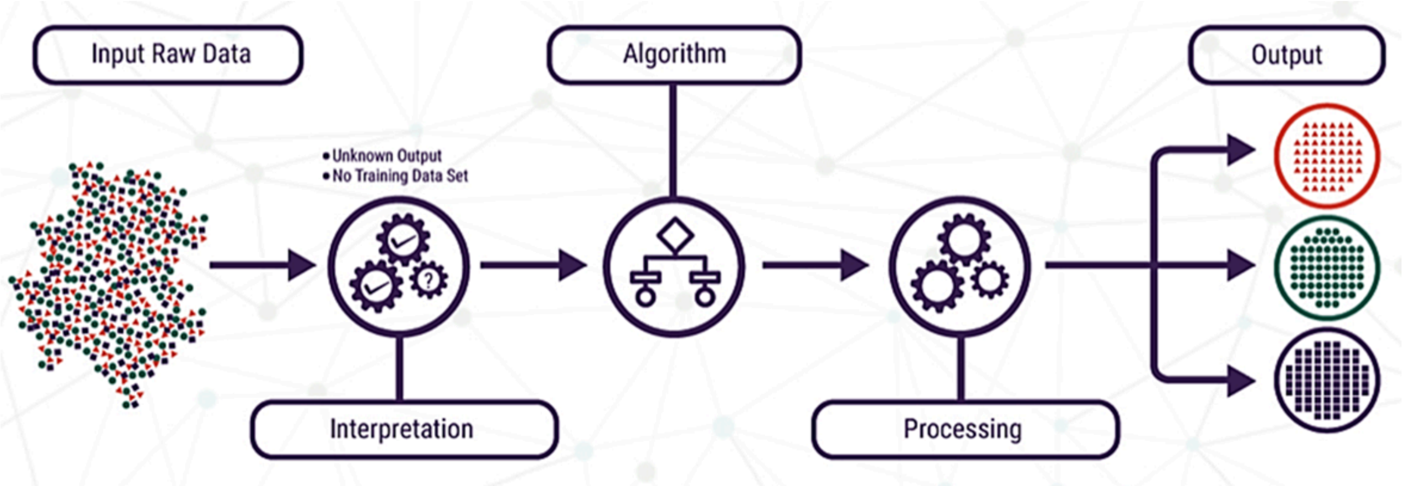
Introduction to Unsupervised Learning

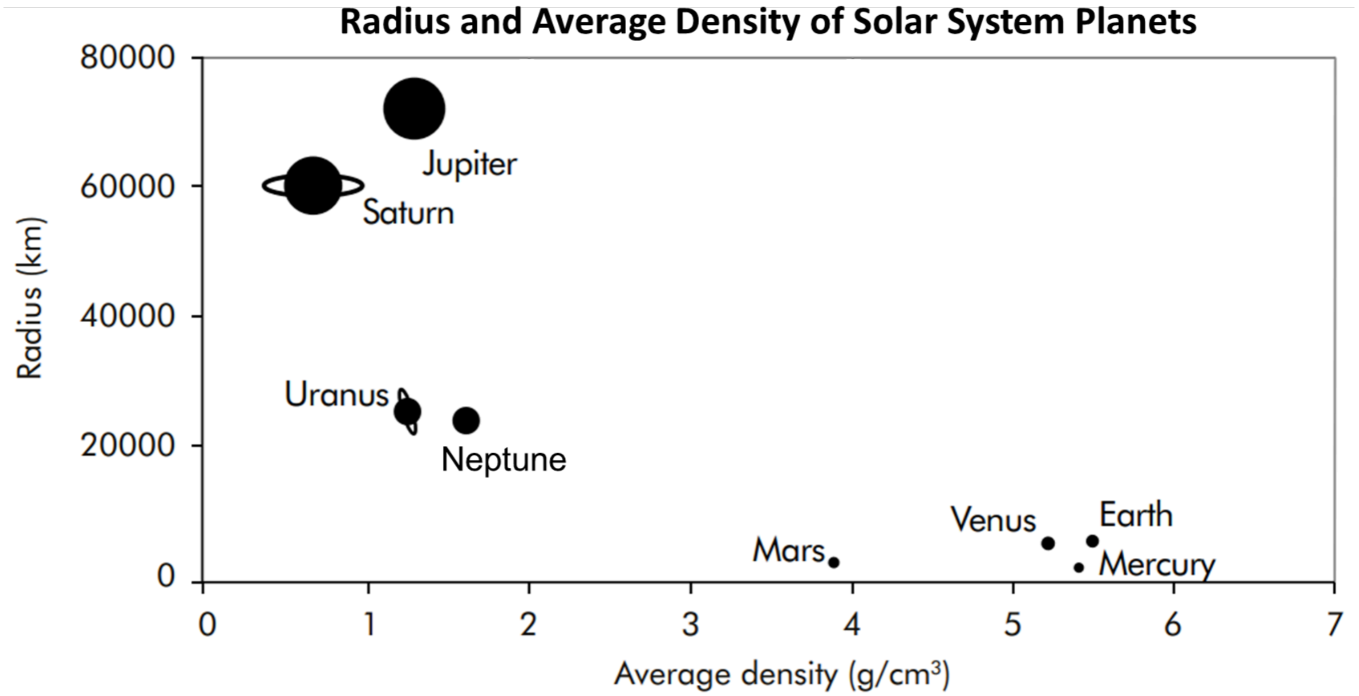
Unsupervised learning refers to a machine learning algorithm that infers information from a given dataset without any guidance. The algorithm relies on underlying similarities and patterns within the data to learn how the variables are related to one another. So in a way, we're "letting the data speak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

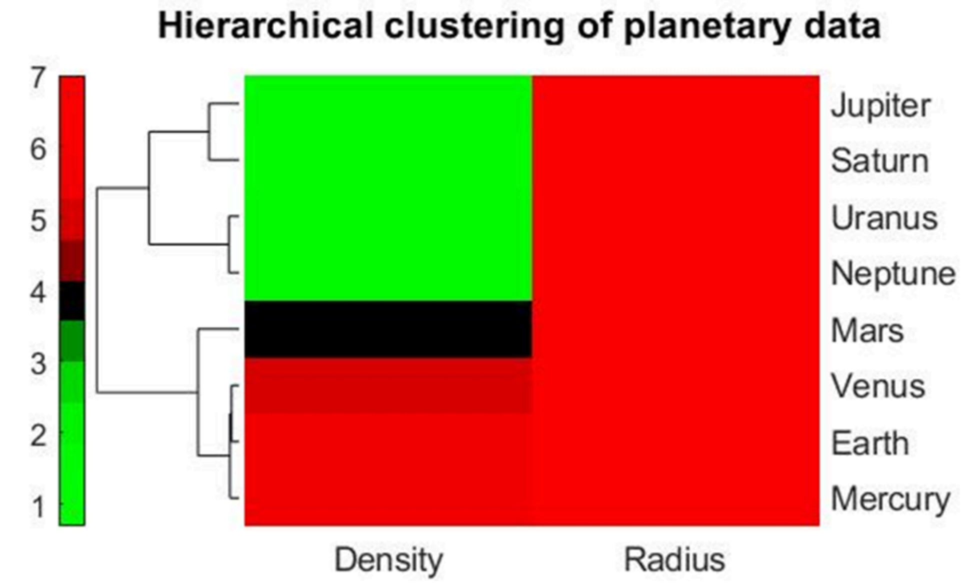
Clustering - Identifying distinct groups based on similarity

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:

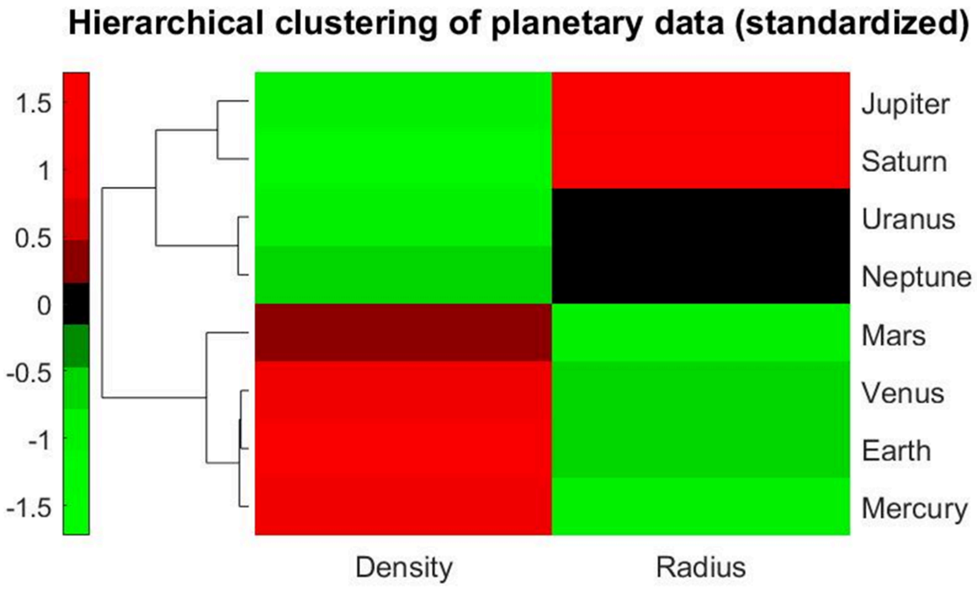


Hierarchical Clustering

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:

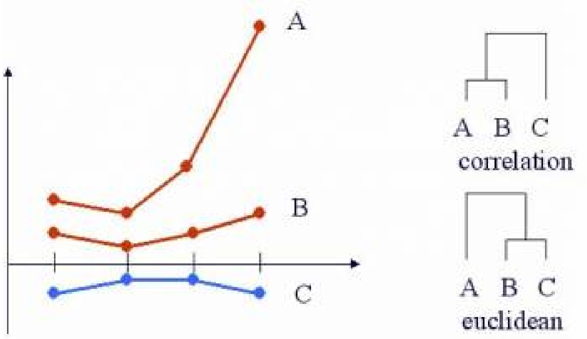


You may have noticed that the radius column doesn't show any differences between planets. This is because the range of values represented by the color gradient is limited to 1 through 7 (recall that planet radius was in the 10,000 range). To have a better visual sense in how a metric (e.g. radius) differs between observations (e.g. planets), we can **standardize** the values within a metric. Below is what the same clustergram of planetary data would look like with standardized 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:

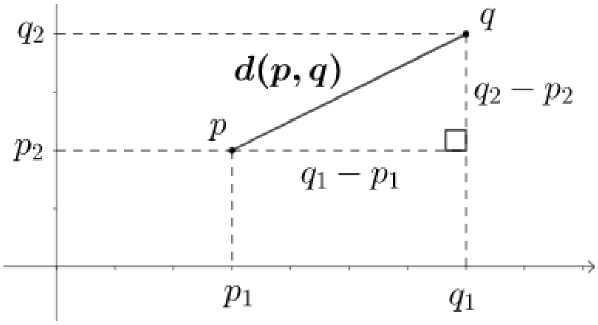


Euclidean Distance

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
```

```
a = 1×4
    2.0000    1.5000    3.0000    7.0000
```

```
b
```

```
b = 1×4
    1.0000    0.5000    1.0000    1.5000
```

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

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

```
d_euclid_man = 6.0208
```

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

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

```
d_euclid_fun = 6.0208
```

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])
```

```
d_euclid_3 = 1×3
    6.0208    9.4472    3.6742
```

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')
```

```
d_corr_man = 0.1004
```

Or using the *pdist* function:

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

```
d_corr_fun = 0.1004
```

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

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

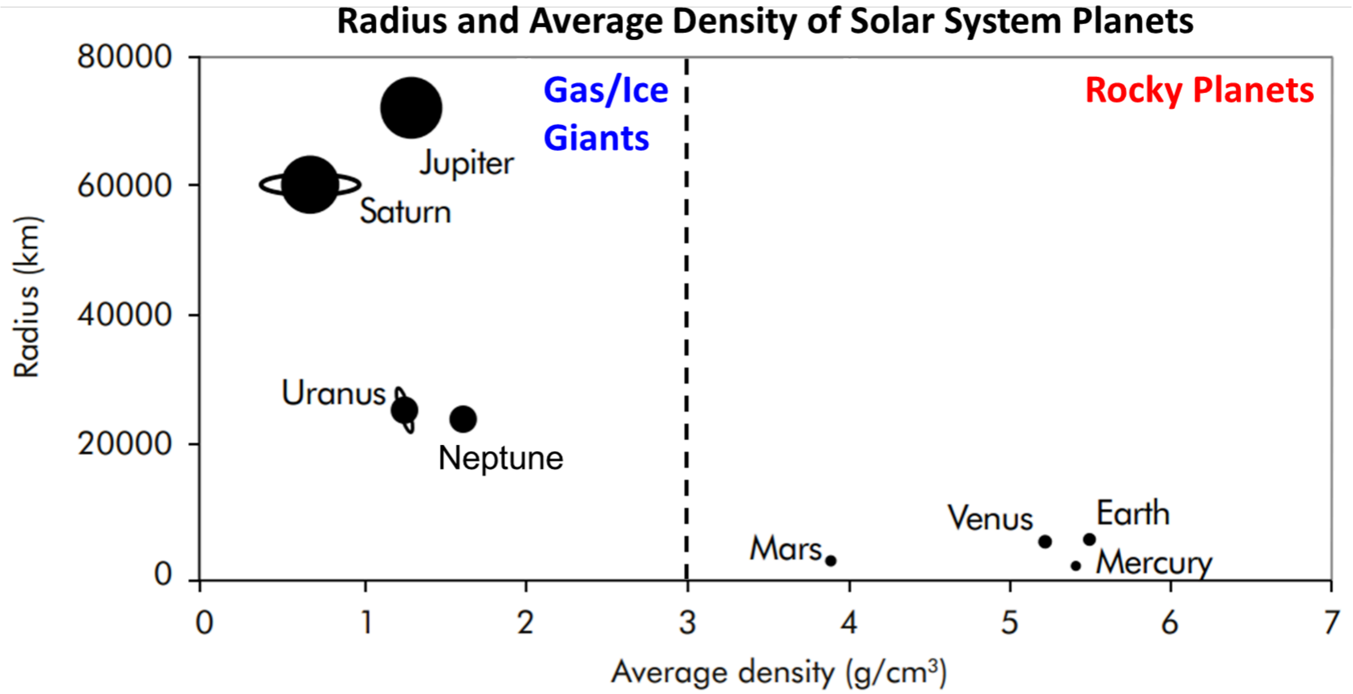
```
d_corr_3 = 1×3
    0.1004    1.5205    1.7071
```

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

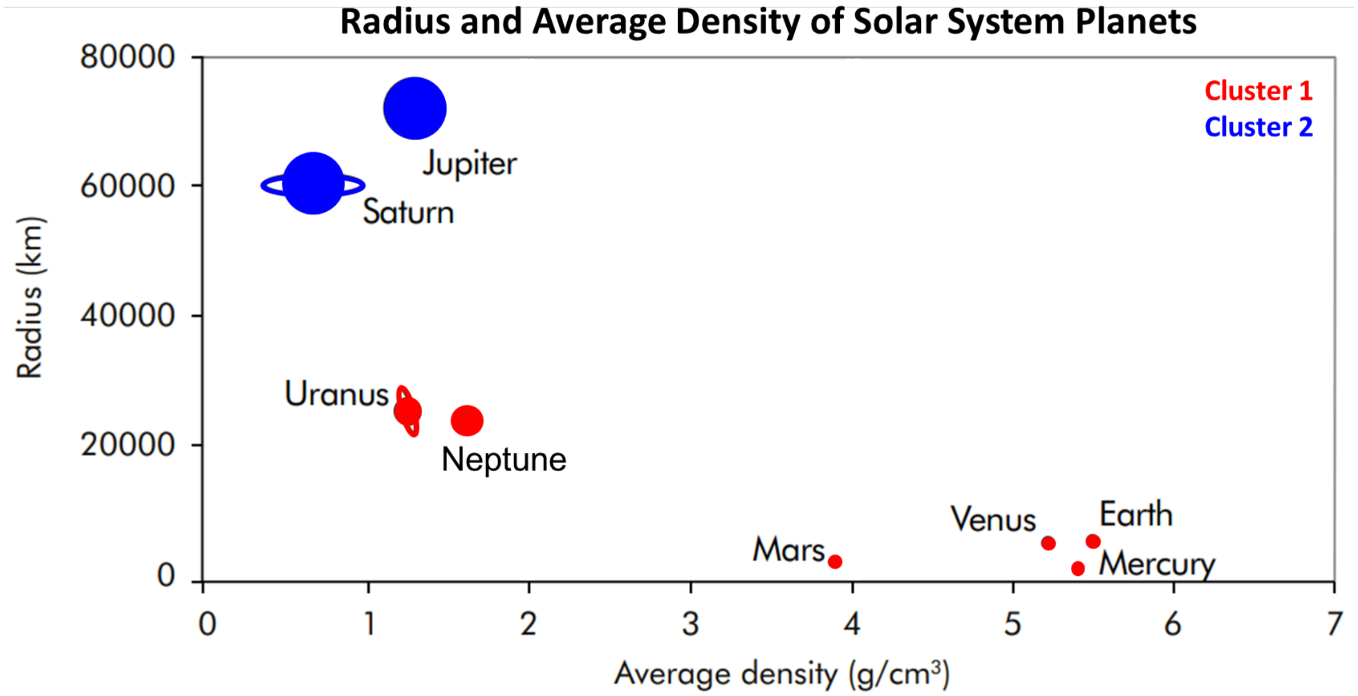
Additional information: refer to the *pdist* documentation for all available distance metrics.

K-means Clustering

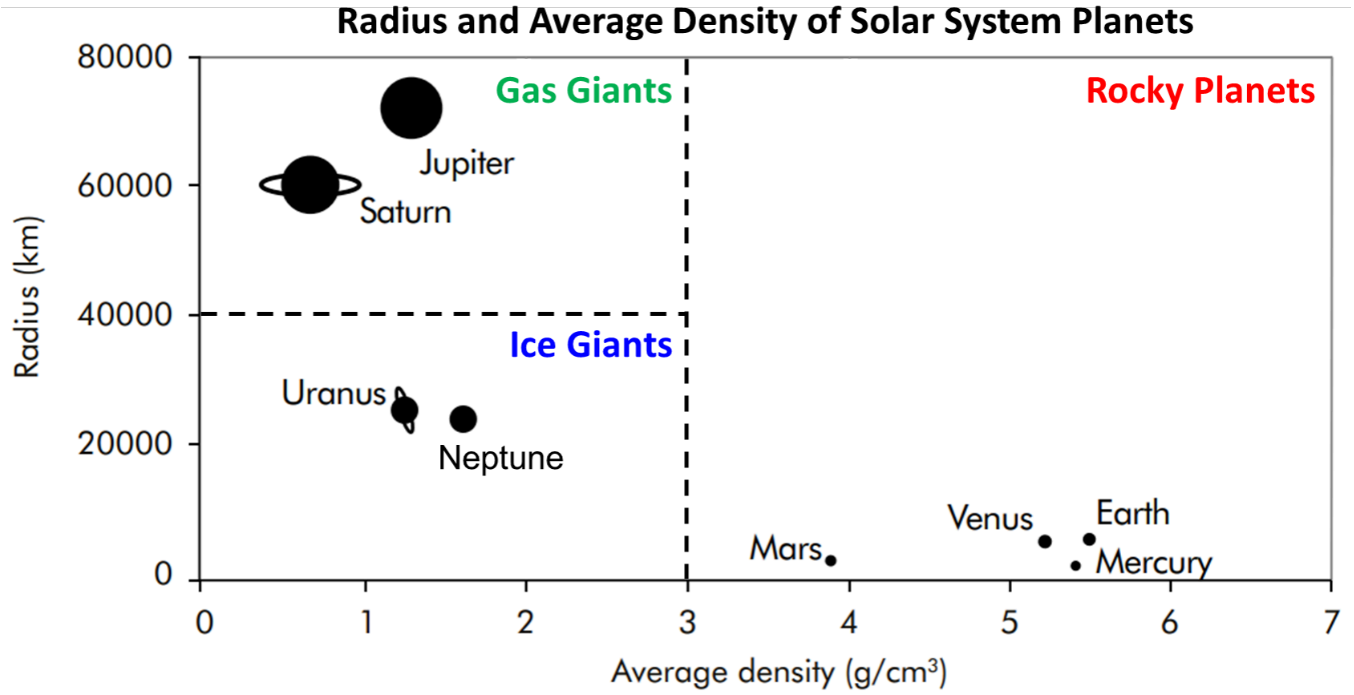
For **k-means clustering**, we specify how many distinct groups we think that the data will cluster into. K-means clustering then allocates data points to a cluster by minimizing the in-cluster sum of squares (more on this later). The value for k is typically chosen based on *a priori* information we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



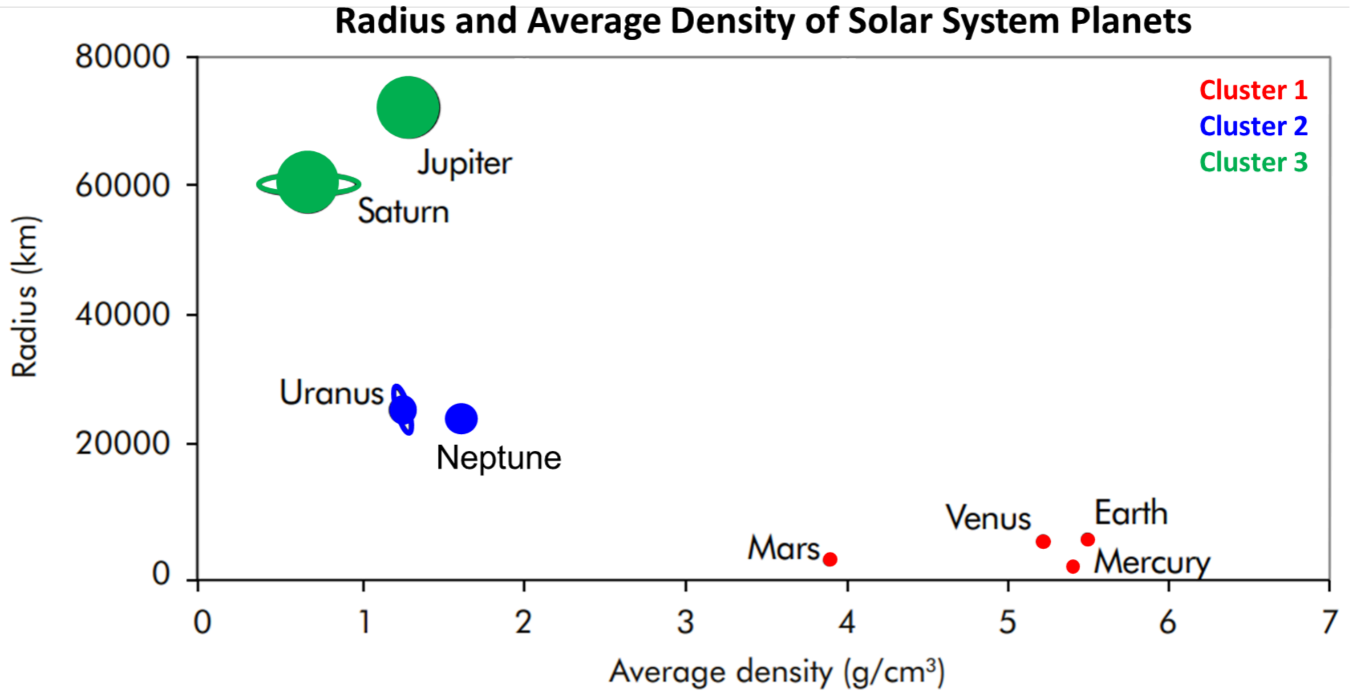
Based on this additional information, we could apply k-means clustering with k = 2 to group our planets:



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 this mean value 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 score for our planetary data based on k-means clustering with different k values and visualizes the clustered data:

```
% Load and view planetary data
load('planetary_data.mat')                       % load planetary data
disp(planet_table)                               % 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
```

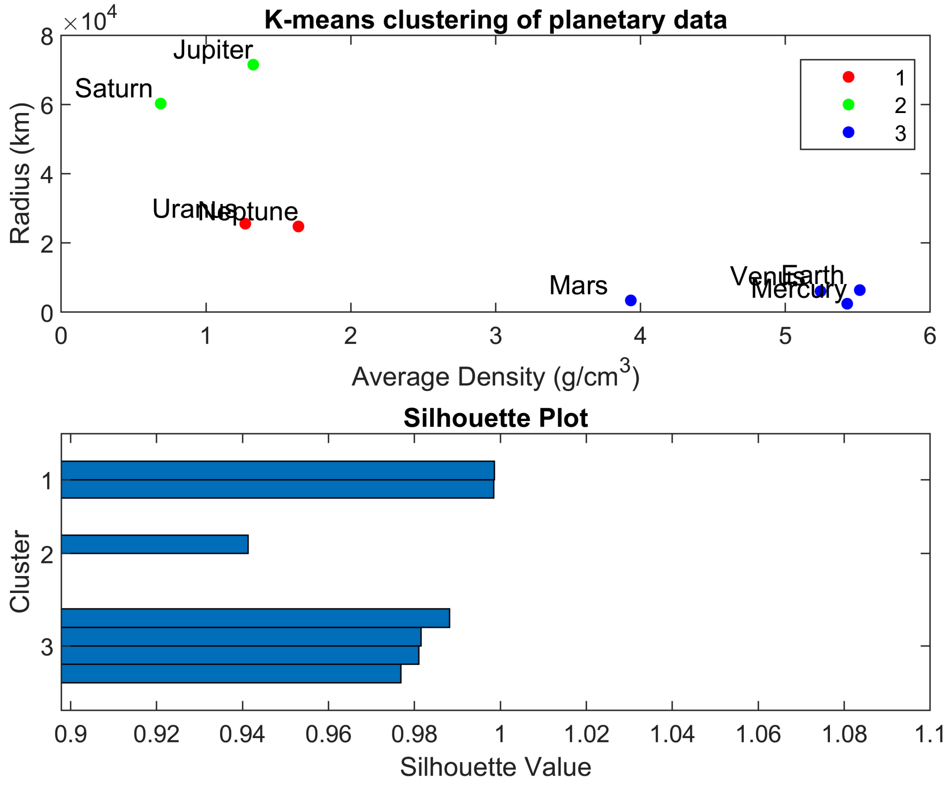
```
% Do k-means clustering and calculate silhouette score
X = [density radius];                          % array of planetary data
idx = kmeans(X,3);      % k-means clustering
s = silhouette(X,idx);                         % silhouette values
s_score = mean(s)                              % silhouette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



Note: As the placement of cluster centroid is random for every iteration, clustering results may fluctuate for different iterations. The *kmeans* function contains additional properties that can be used to address this problem. For the purposes of this lesson, we'll use the *kmeans* function as is, but we'll keep in mind that our clustering results may differ between iterations.

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 based on health measurements

Let's say we want to know if individuals who suffered from a heart attack were similar across their health measurements (e.g. heartrate, BMI) using hierarchical clustering. We would also like to know if the health profile for this group was different from that of individuals who did not have a record of heart attack. The first step is to load the data and define the subset we're interested in:

```
fram = readtable('frmgham2.xls');              % load the Framingham data
health_vars = {'HOSPMI',...                    % define health variables
    'TOTCHOL','SYSBP','DIABP','BMI',...        % (+ heart attack occurrence)
    'HEARTRTE','GLUCOSE','HDLC','LDLC'};
fram_health = fram(:,health_vars)              % create variable for data of interest
```

```
fram_health = 11627×9 table
```

...

	HOSPMI	TOTCHOL	SYSBP	DIABP	BMI	HEARTRTE	GLUCOSE	HDLC
1	1	195	106.0000	70.0000	26.9700	80	77	NaN
2	1	209	121.0000	66.0000	NaN	69	92	31
3	0	250	121.0000	81.0000	28.7300	95	76	NaN
4	0	260	105.0000	69.5000	29.4300	80	86	NaN
5	0	237	108.0000	66.0000	28.5000	80	71	54
6	0	245	127.5000	80.0000	25.3400	75	70	NaN
7	0	283	141.0000	89.0000	25.3400	75	87	NaN
8	0	225	150.0000	95.0000	28.5800	65	103	NaN
9	0	232	183.0000	109.0000	30.1800	60	89	NaN
10	0	285	130.0000	84.0000	23.1000	85	85	NaN
11	0	343	109.0000	77.0000	23.4800	90	72	NaN
12	0	NaN	155.0000	90.0000	24.6100	74	NaN	NaN
13	0	228	180.0000	110.0000	30.3000	77	99	NaN
14	0	230	177.0000	102.0000	31.3600	120	86	NaN
15	0	220	180.0000	106.0000	31.1700	86	81	46
16	0	205	138.0000	71.0000	33.1100	60	85	NaN

	HOSPMI	TOTCHOL	SYSBP	DIABP	BMI	HEARTRTE	GLUCOSE	HDLC
17	0	220	149.0000	81.0000	36.7600	80	98	NaN
18	0	313	100.0000	71.0000	21.6800	79	78	NaN
19	0	NaN	109.5000	72.5000	22.1900	75	NaN	NaN
20	0	320	110.0000	46.0000	22.0200	75	87	34
21	0	260	141.5000	89.0000	26.3600	76	79	NaN
22	0	292	132.0000	90.0000	25.3900	85	81	NaN
23	0	280	168.0000	100.0000	25.7200	92	82	44
24	0	225	162.0000	107.0000	23.6100	93	88	NaN
25	0	258	147.0000	102.0000	27.5000	75	74	NaN
26	0	211	173.0000	123.0000	29.1100	75	85	48
27	0	254	133.0000	76.0000	22.9100	75	76	NaN
28	0	NaN	127.5000	70.0000	20.3000	60	73	NaN
29	0	291	120.0000	70.0000	21.9800	62	83	60
30	0	247	131.0000	88.0000	27.6400	72	61	NaN
31	0	294	142.0000	94.0000	26.3100	98	64	NaN
32	0	288	165.0000	92.0000	26.4400	105	71	NaN
33	0	332	124.0000	88.0000	31.3100	65	84	NaN
34	0	225	120.0000	80.0000	27.9800	75	74	NaN
35	0	159	124.0000	78.0000	26.6200	68	135	53
36	0	226	114.0000	64.0000	22.3500	85	NaN	NaN
37	0	196	136.0000	89.0000	21.2400	110	76	NaN
38	0	264	141.0000	81.0000	24.7700	85	97	58
39	0	221	140.0000	90.0000	21.3500	95	70	NaN
40	0	232	138.0000	90.0000	22.3700	64	72	NaN
41	0	222	139.5000	82.0000	21.7600	70	NaN	NaN
42	0	215	144.5000	80.0000	22.9600	57	91	62
43	0	291	112.0000	78.0000	23.3800	80	89	NaN
44	0	195	122.0000	84.5000	23.2400	75	78	NaN
45	0	265	139.0000	95.0000	30.0500	90	80	NaN
46	0	212	163.0000	96.0000	31.4500	82	84	40
47	0	195	139.0000	88.0000	26.8800	85	65	NaN
48	0	201	160.0000	101.0000	26.2800	90	69	NaN
49	0	162	152.5000	101.0000	26.4300	105	78	31
50	0	190	108.0000	70.5000	21.5900	72	85	NaN

	HOSPMI	TOTCHOL	SYSBP	DIABP	BMI	HEARTRTE	GLUCOSE	HDLC
51	0	244	122.0000	77.5000	21.7700	75	89	NaN
52	0	226	126.0000	75.0000	21.5900	85	102	84
53	0	185	123.5000	77.5000	29.8900	70	NaN	NaN
54	0	270	127.0000	84.0000	34.5500	65	85	NaN
55	0	NaN	129.0000	76.0000	35.7100	76	NaN	NaN
56	1	234	148.0000	78.0000	34.1700	70	113	NaN
57	1	272	162.0000	78.0000	33.2900	60	123	NaN
58	1	283	159.0000	69.0000	32.9300	70	230	45
59	0	215	132.0000	82.0000	25.1100	71	75	NaN
60	0	256	151.0000	86.0000	22.7300	80	93	NaN
61	0	236	150.0000	89.0000	24.0100	90	80	93
62	0	270	137.5000	90.0000	21.9600	75	83	NaN
63	0	300	134.0000	88.0000	23.8100	80	82	NaN
64	0	290	132.0000	70.0000	22.7300	100	90	54
65	1	294	102.0000	68.0000	24.1800	62	66	NaN
66	0	260	110.0000	72.5000	26.5900	65	NaN	NaN
67	0	256	126.0000	82.0000	25.4300	60	77	NaN
68	0	298	109.0000	60.0000	26.2000	80	100	38
69	0	225	132.0000	91.0000	26.0900	73	83	NaN
70	0	233	123.0000	82.0000	25.0100	75	67	NaN
71	0	240	131.0000	90.0000	26.7700	70	76	40
72	0	272	182.0000	121.0000	32.8000	85	65	NaN
73	0	247	130.0000	88.0000	30.3600	72	74	NaN
74	0	248	135.0000	75.0000	30.1600	76	76	NaN
75	0	249	157.0000	87.0000	29.5500	65	73	78
76	0	295	102.0000	68.0000	28.1500	60	63	NaN
77	0	292	126.0000	90.0000	29.8400	55	65	NaN
78	0	300	125.0000	83.0000	34.3400	58	96	47
79	0	226	115.0000	85.5000	27.5700	75	75	NaN
80	0	232	123.0000	86.0000	27.8600	75	66	NaN
81	0	206	129.0000	85.0000	26.4000	84	69	47
82	1	227	125.0000	90.0000	23.6200	86	79	NaN
83	1	296	118.0000	90.0000	23.3400	60	78	NaN
84	1	320	148.0000	89.0000	22.7800	80	110	42

	HOSPMI	TOTCHOL	SYSBP	DIABP	BMI	HEARTRTE	GLUCOSE	HDLC
85	0	209	150.0000	85.0000	20.7700	90	88	NaN
86	0	175	134.0000	82.5000	18.5900	72	75	NaN
87	0	207	133.0000	74.0000	17.6500	75	74	NaN
88	0	283	145.0000	85.0000	17.2800	88	80	65
89	0	235	125.0000	74.5000	25.9000	65	73	NaN
90	0	214	147.0000	74.0000	24.7100	96	87	NaN
91	0	230	141.0000	63.0000	22.3300	65	75	NaN
92	0	199	180.0000	82.0000	21.3000	90	NaN	94
93	0	225	124.5000	92.5000	38.5300	95	83	NaN
94	0	244	156.0000	109.0000	38.5300	85	67	NaN
95	0	226	190.0000	123.0000	45.4300	100	73	34
96	0	257	153.5000	102.0000	28.0900	72	75	NaN
97	0	281	177.0000	107.0000	29.8600	75	77	NaN
98	0	226	157.0000	95.0000	29.8600	88	99	61
99	0	178	160.0000	98.0000	40.1100	75	225	NaN
100	0	155	173.0000	90.0000	36.8800	72	200	NaN

⋮

We'll remove rows containing NaN values and get the subset for the first 50 entries. We'll also create a variable for the heart attack occurrence and a numerical array for the health measurement data:

```
fram_health = rmmissing(fram_health);                % remove rows containing NaN values
fram_health = fram_health(1:50,:)                    % subset of first 50 entries
```

```
fram_health = 50×9 table
```

...

	HOSPMI	TOTCHOL	SYSBP	DIABP	BMI	HEARTRTE	GLUCOSE	HDLC
1	0	237	108.0000	66	28.5000	80	71	54
2	0	220	180.0000	106	31.1700	86	81	46
3	0	320	110.0000	46	22.0200	75	87	34
4	0	280	168.0000	100	25.7200	92	82	44
5	0	211	173.0000	123	29.1100	75	85	48
6	0	291	120.0000	70	21.9800	62	83	60
7	0	159	124.0000	78	26.6200	68	135	53
8	0	264	141.0000	81	24.7700	85	97	58
9	0	215	144.5000	80	22.9600	57	91	62

	HOSPMI	TOTCHOL	SYSBP	DIABP	BMI	HEARTRTE	GLUCOSE	HDLC
10	0	212	163.0000	96	31.4500	82	84	40
11	0	162	152.5000	101	26.4300	105	78	31
12	0	226	126.0000	75	21.5900	85	102	84
13	1	283	159.0000	69.0000	32.9300	70	230	45
14	0	236	150.0000	89.0000	24.0100	90	80	93
15	0	290	132.0000	70.0000	22.7300	100	90	54
16	0	298	109.0000	60.0000	26.2000	80	100	38
17	0	240	131.0000	90.0000	26.7700	70	76	40
18	0	249	157.0000	87.0000	29.5500	65	73	78
19	0	300	125.0000	83.0000	34.3400	58	96	47
20	0	206	129.0000	85.0000	26.4000	84	69	47
21	1	320	148.0000	89.0000	22.7800	80	110	42
22	0	283	145.0000	85.0000	17.2800	88	80	65
23	0	226	190.0000	123.0000	45.4300	100	73	34
24	0	226	157.0000	95.0000	29.8600	88	99	61
25	0	246	202.0000	111.0000	26.6300	63	80	41
26	0	216	100.0000	75.0000	29.8200	70	68	30
27	0	293	127.0000	77.0000	25.6800	90	115	52
28	0	176	139.0000	72.0000	27.0000	74	77	26
29	0	206	112.0000	80.0000	21.9600	63	138	84
30	0	241	135.0000	70.0000	25.6800	90	93	41
31	1	230	142.0000	90.5000	24.3300	70	61	30
32	0	251	132.0000	77.0000	19.3000	82	76	53
33	0	224	125.0000	68.0000	27.3500	80	109	35
34	0	241	145.0000	85.0000	25.6600	96	102	68
35	1	216	117.0000	70.0000	21.3500	72	49	42
36	0	180	195.0000	107.0000	22.3600	68	75	60
37	0	229	145.0000	77.0000	23.0900	72	83	39
38	0	262	188.0000	104.0000	20.2600	61	82	54
39	0	276	148.0000	84.0000	19.6400	80	88	39
40	1	236	125.0000	74.0000	22.4900	90	103	30
41	0	246	153.0000	78.0000	25.3100	110	83	54
42	0	194	206.0000	106.0000	31.4500	120	178	59
43	0	238	113.0000	75.0000	23.3500	90	89	46

	HOSPMI	TOTCHOL	SYSBP	DIABP	BMI	HEARTRTE	GLUCOSE	HDLC
44	0	314	143.0000	66.0000	18.3200	93	96	61
45	0	272	116.0000	66.0000	23.1800	77	74	41
46	0	213	122.0000	77.0000	31.0600	60	83	56
47	0	208	122.0000	74.0000	40.3300	75	82	35
48	0	229	141.0000	91.0000	32.4900	96	96	62
49	0	229	120.0000	70.0000	23.2400	66	74	73
50	0	198	150.0000	82.0000	40.8000	60	81	56

```
heart_attack = repmat({'No'},50,1);                     % heart attack occurrence (0 = No)
heart_attack(fram_health.HOSPMI == 1) = {'Yes'}         % heart attack occurrence (1 = Yes)
```

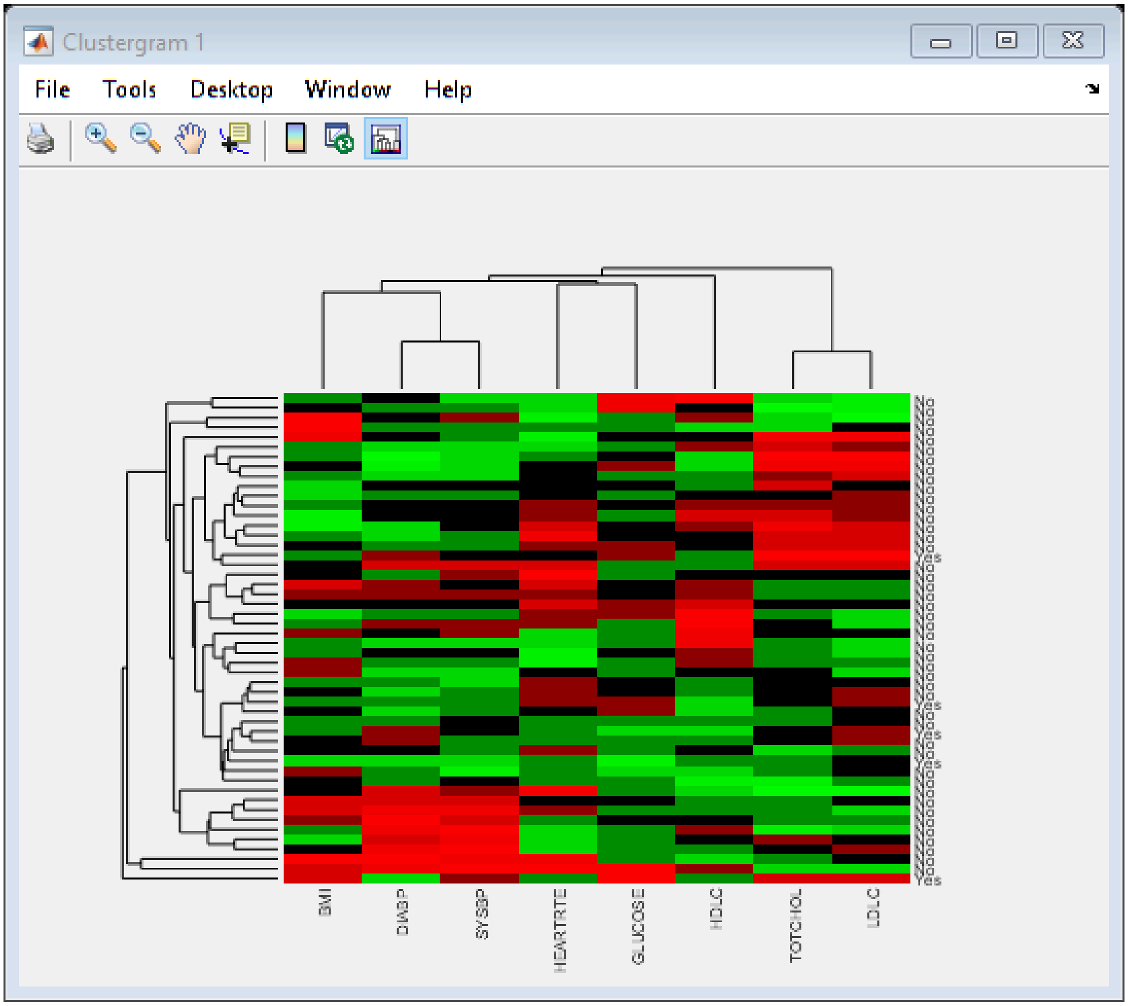
```
heart_attack = 50×1 cell array
'No'
'No'
'No'
'No'
'No'
'No'
'No'
'No'
'No'
'No'
     ⋮
```

```
fram_health_data = table2array(fram_health(:,2:end))    % numerical data array (- heart attack
```

```
fram_health_data = 50×8
  237.0000  108.0000   66.0000   28.5000   80.0000   71.0000   54.0000  141.0000
  220.0000  180.0000  106.0000   31.1700   86.0000   81.0000   46.0000  135.0000
  320.0000  110.0000   46.0000   22.0200   75.0000   87.0000   34.0000  286.0000
  280.0000  168.0000  100.0000   25.7200   92.0000   82.0000   44.0000  236.0000
  211.0000  173.0000  123.0000   29.1100   75.0000   85.0000   48.0000  163.0000
  291.0000  120.0000   70.0000   21.9800   62.0000   83.0000   60.0000  217.0000
  159.0000  124.0000   78.0000   26.6200   68.0000  135.0000   53.0000  106.0000
  264.0000  141.0000   81.0000   24.7700   85.0000   97.0000   58.0000  206.0000
  215.0000  144.5000   80.0000   22.9600   57.0000   91.0000   62.0000  135.0000
  212.0000  163.0000   96.0000   31.4500   82.0000   84.0000   40.0000  172.0000
     ⋮
```

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 = heart attack occurrence, columns = health measurements). Finally, we'll standardize each column to better compare measurements between patients:

```
col_names = fram_health.Properties.VariableNames(2:end); % column names (- heart attack occurre
fram_health_cg = clustergram(fram_health_data,...        % clustergram with...
    'RowLabels',heart_attack,...                         % labeled rows (heart attack occurrenc
    'ColumnLabels',col_names,...                         % labeled columns (health measurements
    'Standardize','column')                              % standardized columns
```



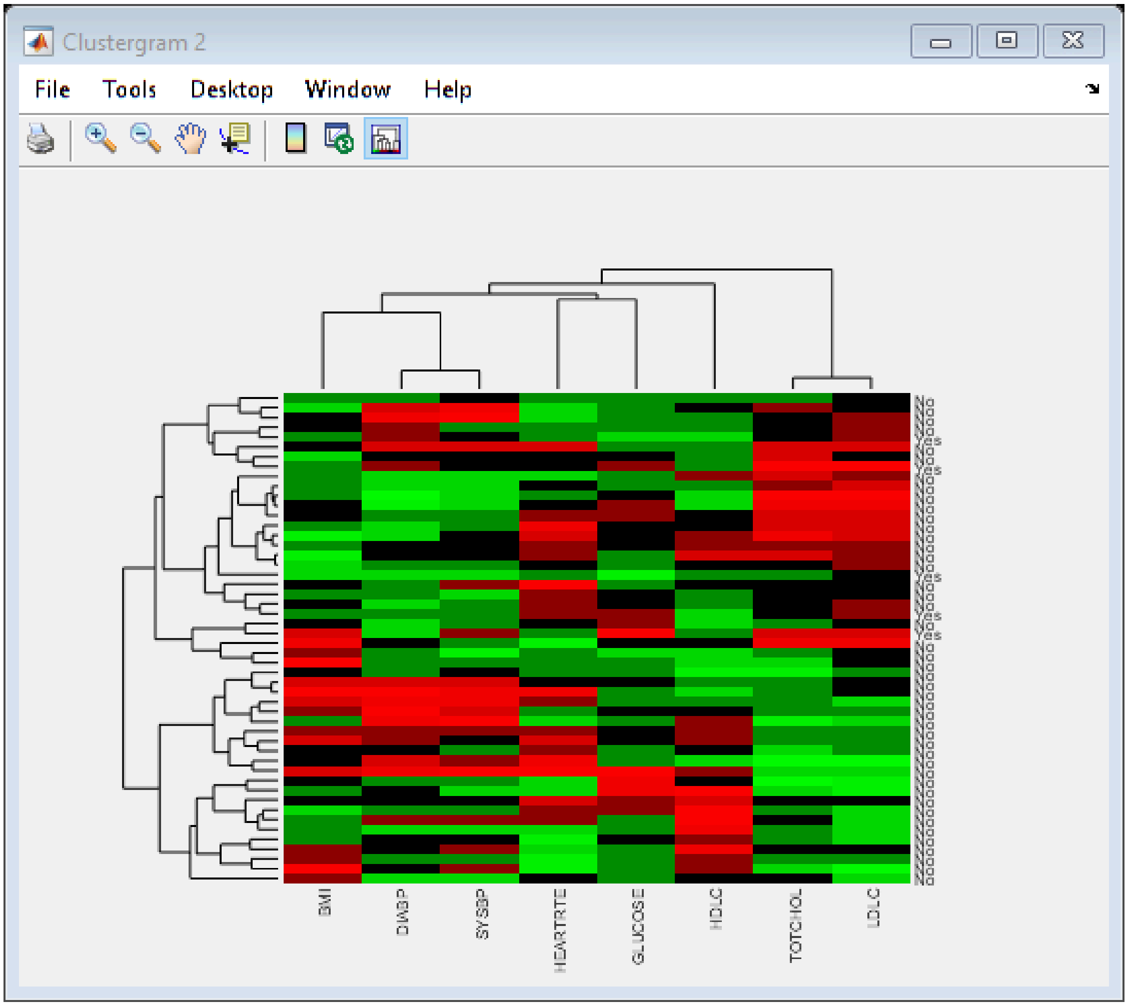
```
Clustergram object with 50 rows of nodes and 8 columns of nodes.
```

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

Question: Did individuals who suffered from a heart attack cluster together?

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

```
fram_health_cg_corr = clustergram(fram_health_data,...  % clustergram with...
    'RowLabels',heart_attack,...                        % labeled rows (heart attack occurrence
    'ColumnLabels',col_names,...                        % labeled columns (health measurements)
    'Standardize','column',...                          % standardized columns
    'RowPDist','correlation',...                        % correlation as distance metric (rows)
    'ColumnPDist','correlation')                        % correlation as distance metric (colum
```



```
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? Did individuals who suffered from a heart attack cluster together?

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 cholesterol and LDLC levels:

```
fram_kmeans = fram_health(:,{'HOSPMI','TOTCHOL','LDLC'}); % define dataset to use
```

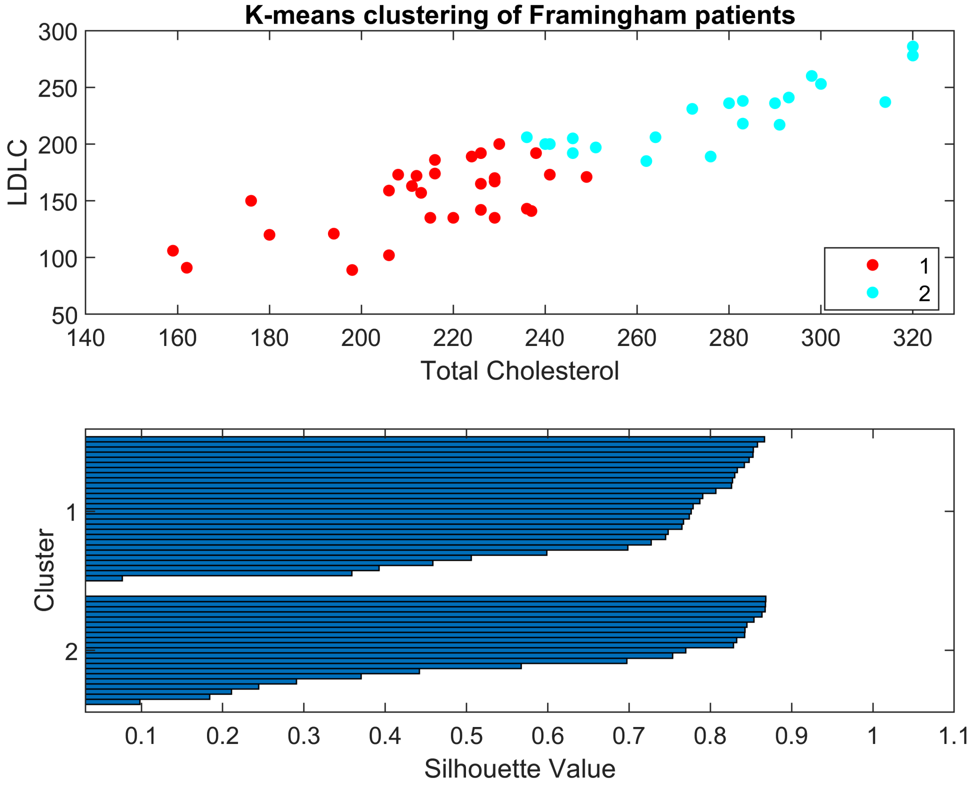
Let's see if we're able to pick out diabetic patients from our data using k = 2:

```
X = table2array(fram_kmeans);                         % numerical array of dataset
idx = kmeans(X,2);                                    % k-means clustering (idx = cluster identity)
s = silhouette(X,idx);                                % vector for silhouette values
s_score = mean(s)                                     % silhouette score
```

```
s_score = 0.6633
```

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.TOTCHOL,fram_kmeans.LDLC,idx); % scatter plot with cluster annotation
xlabel('Total Cholesterol'); ylabel('LDLC')         % 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 encompass individuals who have suffered from heart attack?

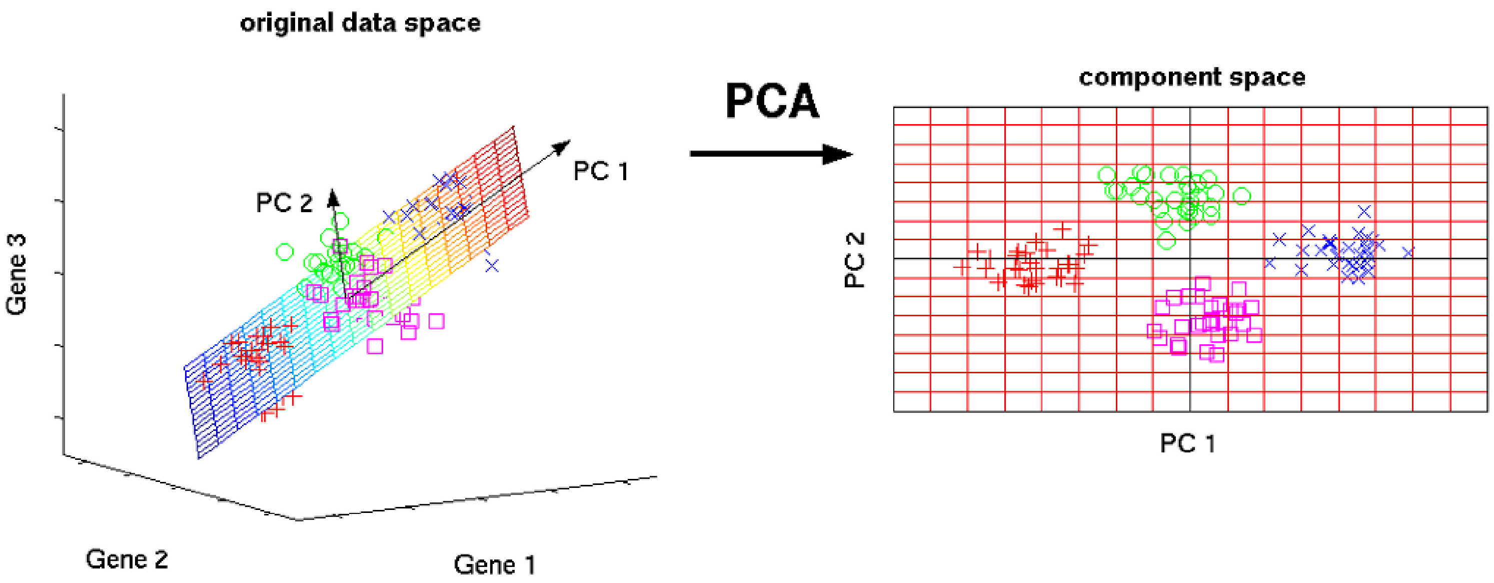
Activity: How can we modify the code above to apply k-means clustering for k = 3? How does the s_score value compare between k = 3 and k = 2?

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, projecting it onto a new coordinate system defined by principal components.



A **principal component (PC)** is a vector of weight (or coefficient) values assigned to each **feature**, or descriptive variable. 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:

$$XP = Y$$

$$X = original\ data \qquad P = matrix\ of\ PCs \qquad Y = transformed\ data$$

$$(m \times n) \qquad (n \times n) \qquad (m \times n)$$

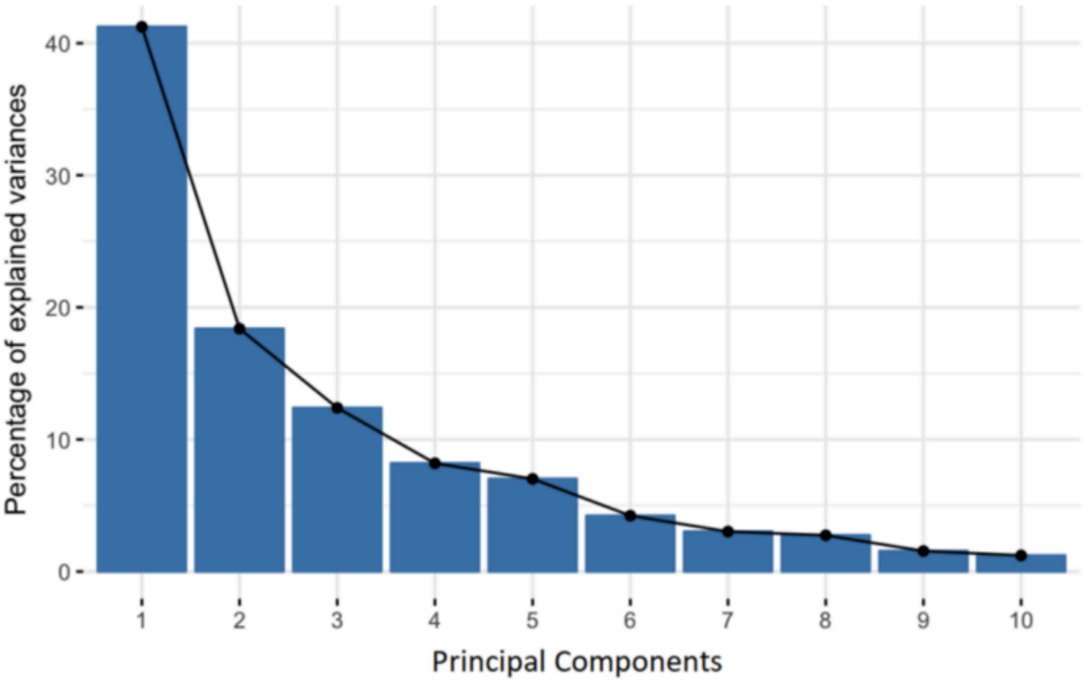
$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{bmatrix} \begin{bmatrix} p_{1,1} & \cdots & p_{1,n} \\ \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,n} \end{bmatrix} = \begin{bmatrix} y_{1,1} & \cdots & y_{1,n} \\ \vdots & \ddots & \vdots \\ y_{m,1} & \cdots & y_{m,n} \end{bmatrix}$$

In PCA, P (matrix of PCs) is typically determined 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).

Note: refer to the *pca* MATLAB documention for additional information on algorithms used for PCA.

Why do PCA?

The main goal when conducting PCA is to determine a reduced representation of the original data that can be visualized. This is done by determining 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.

Note 1: PCs differ in the **weights** (or coefficients) associated with the features. If a PC (e.g. PC1) can describe most of the variance in the data (> 50%), then we can evaluate the coefficients in this PC to see which feature(s) are most important (have the greatest absolute weight). In other words, the coefficient values for a PC can tell us which features are most responsible for the patterns seen in our data.

Note 2: As the new representations of the observations in the principal component space are linear combinations of the original variables (features), it's important to **normalize the data** before conducting PCA. This ensures a unit variance for all variables, thus allowing comparable contributions from all variables when the data is transformed. In other words, normalizing the data reduces any bias there might be for a particular feature.

In MATLAB, the *pca* function is used to perform PCA based on SVD. The *zscore* function can be used to normalize the data before doing PCA.

Example: applying PCA for the Framingham data

For this analysis, we'll consider the same dataset of health 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 peforming PCA. Let's first normalize this data (*X*) then determine the matrix of principal components (*P*), the matrix of the transformed data (*Y*), and a vector for the variance explained by each PC (*explained*):

```
% Normalize health data
X = zscore(fram_health_data);

% Do PCA (P = PC matrix, Y = transformed data, explained = variance explained)
```

```
[P, Y, ~, ~, explained] = pca(X)
```

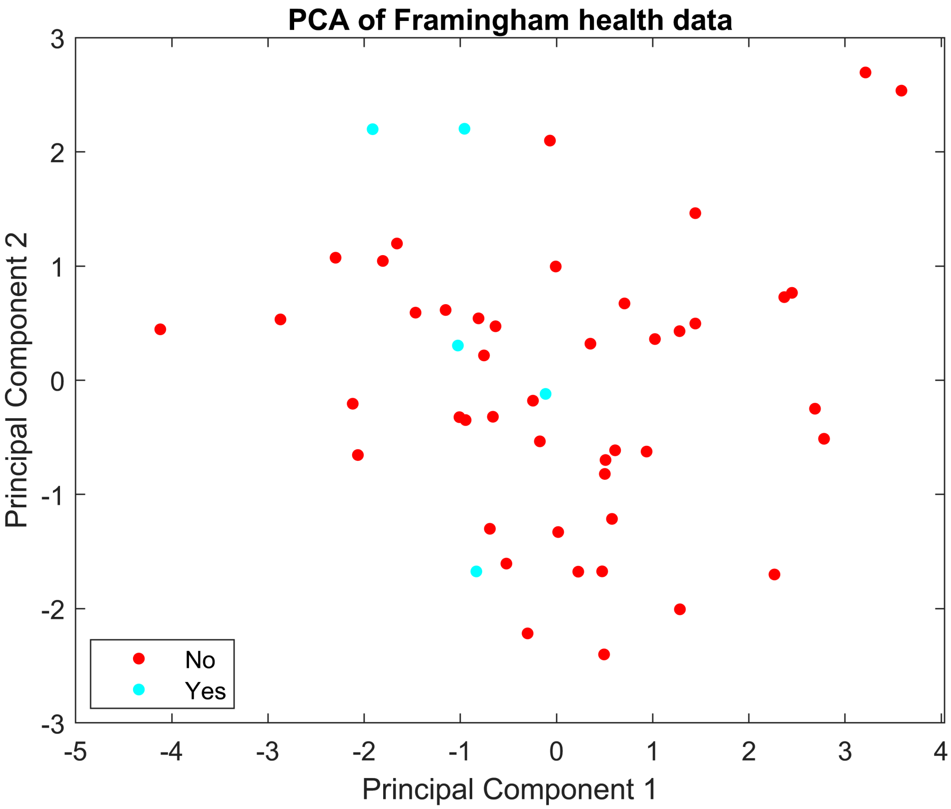
```
P = 8×8
   -0.4827    0.4007    0.1179   -0.1490   -0.2451    0.2669   -0.0702   -0.6588
    0.4259    0.4640    0.1030   -0.2430   -0.2304   -0.2009   -0.6530    0.1175
    0.4975    0.3172   -0.0428   -0.3250   -0.1760   -0.0057    0.7068   -0.1178
    0.3080    0.1195   -0.4000    0.5095   -0.1870    0.6527   -0.1008    0.0050
    0.0865    0.4261    0.2034    0.0669    0.8517    0.1980    0.0014   -0.0198
    0.0080    0.2654    0.4218    0.7255   -0.2016   -0.3907    0.1753   -0.0350
    0.0825   -0.2129    0.7552   -0.1307   -0.1855    0.5133    0.0226    0.2493
   -0.4784    0.4603   -0.1553   -0.0743   -0.1426    0.0839    0.1666    0.6888
Y = 50×8
   -0.5215   -1.6052   -0.2357    0.3907    0.6920    0.8169   -0.1963   -0.5006
    2.3683    0.7278   -0.4408   -0.4640    0.0129    0.0919   -0.2252   -0.4228
   -4.1205    0.4460   -0.7826    0.2102   -0.0142   -0.0804   -0.5994    0.0604
   -0.0685    2.0982   -0.2426   -1.0340    0.0628    0.1511    0.1661    0.0046
    2.4483    0.7653   -0.4888   -0.9148   -0.7875   -0.2474    0.9021    0.0229
   -2.0656   -0.6547    0.3909   -0.5546   -1.0506    0.2614    0.0481   -0.1554
    1.2830   -2.0055    0.5295    1.7387   -0.0868   -1.1953    0.3711    0.1954
   -0.6344    0.4727    0.6382   -0.1334    0.0484    0.2724    0.0503    0.0698
    0.4744   -1.6731    0.5839   -0.3279   -1.0722   -0.5973   -0.2519   -0.0224
    1.4433    0.4967   -0.9619    0.0117    0.0200   -0.0256   -0.0905    0.1580
      ⋮

explained = 8×1
   32.3024
   19.2774
   15.7793
   12.9420
   10.7258
    6.7674
    1.6609
    0.5448
```

Question: How much variance is exlained by PC1 and PC2?

Now we'll visualize our transformed data in the component space by plotting PC1 vs. PC2. We'll also use our variable for heart attack occurrence to label our data:

```
% Visualize transformed data on PC1 vs. PC2 plot (label using heart attack occurrence)
figure
gscatter(Y(:,1), Y(:,2), heart_attack)
xlabel('Principal Component 1')
ylabel('Principal Component 2')
title('PCA of Framingham health data')
```



Based on this plot, we can see that individuals who suffered from a heart attack cluster within the negative range along PC1. We can check the PC matrix (*P*) to see which features contribute the most for PC1:

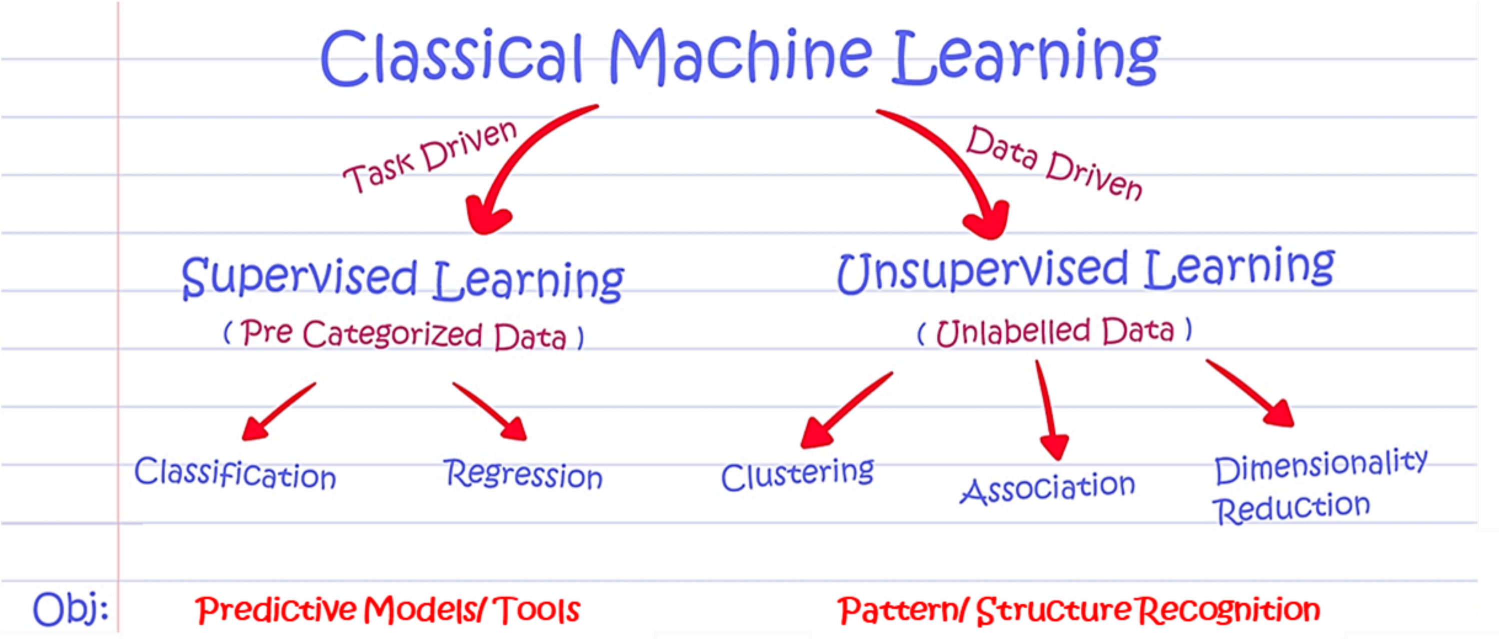
```
% Assess coefficient values for PC1
feature = col_names';   % feature names (health measurements)
PC1 = P(:,1);           % coefficient values for PC1
table(feature, PC1)     % table of features matched to coefficient values
```

```
ans = 8×2 table
```

	feature	PC1
1	'TOTCHOL'	-0.4827
2	'LDLC'	-0.4784
3	'GLUCOSE'	0.0080
4	'HDLC'	0.0825
5	'HEARTRTE'	0.0865
6	'BMI'	0.3080
7	'SYSBP'	0.4259
8	'DIABP'	0.4975

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 learing is used for data exploration, supervised learning is typically used to train and develop predictive or classification tools based on labeled data.



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)
• *zscore:*	normalizes each column in a matrix to have mean 0 and standard deviation of 1

References for additional information:

- Eigenvalue Decomposition (EVD)
- Singular Vector Decomposition (SVD)