transforming-data

April 10, 2017

0.1 Transforming Data

Your goal during the data gathering phase is to record as much working data about your observations as possible since you never know which feature is going to end up being the golden one that allows your machine learning algorithm to succeed. Due to this, there are usually a few redundant or even poor features in your dataset. Think back to those long word problems in grade school that were essentially a simple math question, but came filled with red herrings to throw you off; feeding an unfiltered soup of features to your machine learning algorithms is pretty similar to trying to get it to solve those word problems.

To be effective, many machine learning algorithms need the data passed to them be discerning, discriminating and independent. In this module, you're going to discover methods to get your data behaving like that using transformers. This will help improve your own knowledge of your data, as well as improve your machine learning algorithms' performance.

A transformer is any algorithm you apply to your dataset that changes either the feature count or feature values, but does not alter the number of observations. You can use transformers to mung your data as a pre-processing step to clean it up before it's fed to other algorithms. Another popular transformer use is that of dimensionality reduction, where the number of features in your dataset is intelligently reduced to a subset of the original.

Once you've used a few basic transformers, you will also learn about some data cleansing techniques that attempt to rectify problematic observations.

0.1.1 Principal Component Analysis (PCA)

Unsupervised learning aims to discover some type of hidden structure within your data. Without a label or correct answer to test against, there is no metric for evaluating unsupervised learning algorithms. Principal Component Analysis (PCA), a transformation that attempts to convert your possibly correlated features into a set of linearly uncorrelated ones, is the first unsupervised learning algorithm you'll study.

What is principal component analysis **PCA falls into the group of dimensionality reduction algorithms**. *In many real-world datasets and the problems they represent, you aren't aware of what specifically needs to be measured to succinctly address the issue driving your data collection. So instead, you simply record any feature you can derive, usually resulting in a higher dimensionality than what is truly needed*. This is undesirable, but it's the only reliable way you know to insure you capture the relationship modeled in your data.

If you have reason to believe *your question has a simple answer*, or that *the features you've collected are actually many indirect observations of some inherent source you either cannot or do not know how to measure*, then **dimensionality reduction applies to your needs**.

telephone-front-view

telephone-above-view

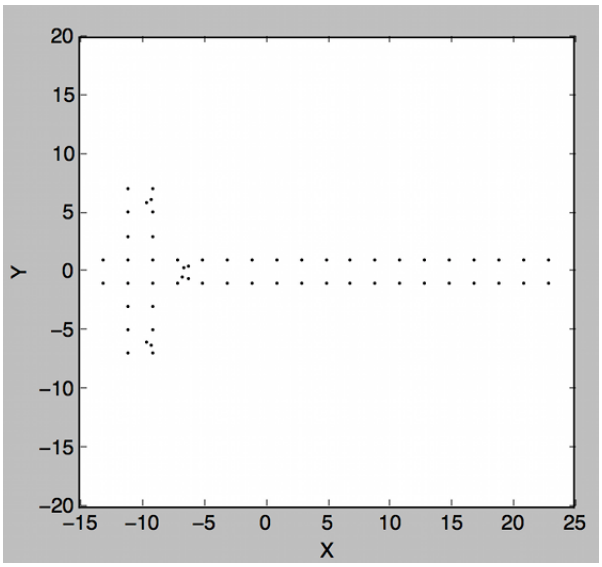
PCA's approach to *dimensionality reduction* is to **derive a set of degrees of freedom that can then be used to reproduce most of the variability of your data**. Picture one of those cartoon style telephone poles; once you have a figure in mind, compare it to this one:

Your envisioned image probably looked similar. You could have pictured it from any other viewing angle, for instance, as if you were floating directly above it looking down:

However you probably didn't, since that view doesn't contain enough variance, or information to easily be discernible as a telephone pole. The frontal view, however, does. Looking at a telephone pole or any other object from various viewing angles gives you more information about that object. If the view angles are really close to one another, the information you get from the views ends up being mostly the same, with a lot of duplicate information. However if you're able to move to a completely different angle, you can get a lot more information about the object you're examining. And if you're wise in choose your view angles, with just a few calculated glimpses of an object, you can build a rather comprehensive understanding of it. PCA calculates those best view angles:

How Does PCA Work? **PCA is one of the most popular techniques for dimensionality reduction**, and *we recommend you always start with it when you have a complex dataset*. **It models a linear subspace of your data by capturing its greatest variability**. Stated differently, **it accesses your dataset's covariance structure directly using matrix calculations and eigenvectors to compute the best unique features that describe your samples**.

An iterative approach to this would **first find the center of your data**, *based off its numeric features*. Next, it would **search for the direction that has the most variance or widest spread of values**. *That direction is the principal component vector, so it is then added to a list*. By **searching for more directions of maximal variance that are orthogonal to all previously computed vectors**, *more principal component can then be added to the list*. **This set of vectors form a new feature space that you can represent your samples with**.



telephone-best-view

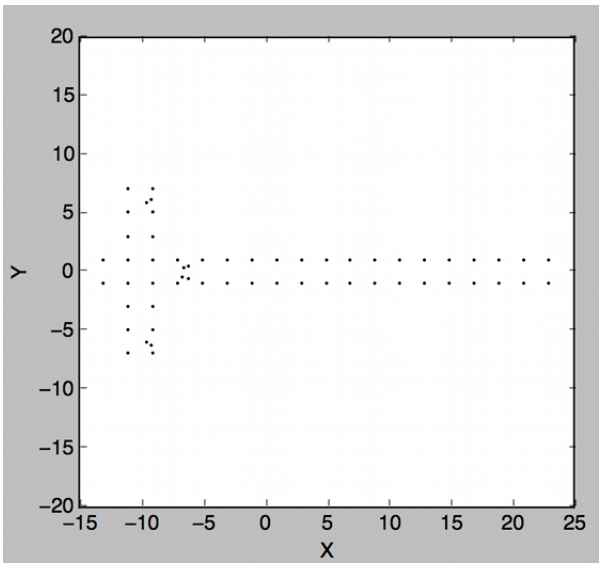
On Dimensions, Features, and Views Each sample in your dataset represents an observable phenomenon, such as an object in the real world. Each feature in your dataset tells you details about your samples. **Recall from earlier chapters that features and views are synonymous terms; this isn't accidental!** Just like looking at an object from different views gives you more information about the object, so too does examining a sample from different features. Similar or correlated features will produce an "overlapped" view of your samples, the same way similar views of an object also overlap.

PCA ensures that each newly computed view (feature) is orthogonal or linearly independent to all previously computed ones, minimizing these overlaps. PCA also orders the features by importance, assuming that the more variance expressed in a feature, the more important it is. In our telephone pole example, the frontal view had more variance than the bird's-eye view and so it was preferred by PCA.

*With the **newly computed features ordered by importance, dropping the least important features on the list intelligently reduces the number of dimensions needed to represent your dataset**, with minimal loss of information*. This has many practical uses, including boiling off high dimensionality observations to just a few key dimensions for visualization purposes, being used as a noise removal mechanism, and as a pre-processing step before sending your data through to other more processor-intensive algorithms. We'll look at more real life use cases in the next unit.

When should I use PCA? PCA, and in fact all dimesionality reduction methods, have three main uses: + To handle the clear goal of reducing the dimensionality and thus complexity of your dataset. + To pre-process your data preparation for other supervised learning tasks, suchs as regression and classification. + To make visualizing your data easier, since we can only preceive three dimensions simultaneously.

According to Nielson Tetrad Demographics, the group of people who watch the most movies are people between the ages of 24 through 35. Let's say you had a list of 100 movies and surveyed 5000 people from within this demographic, asking them to rate all the movies they've seen on a scale of 1-10. By having considerably more data samples (5000 people) than features (100 ordinal movie ratings), you're more likely to avoid the curse of dimensionality.



telephone-best-view

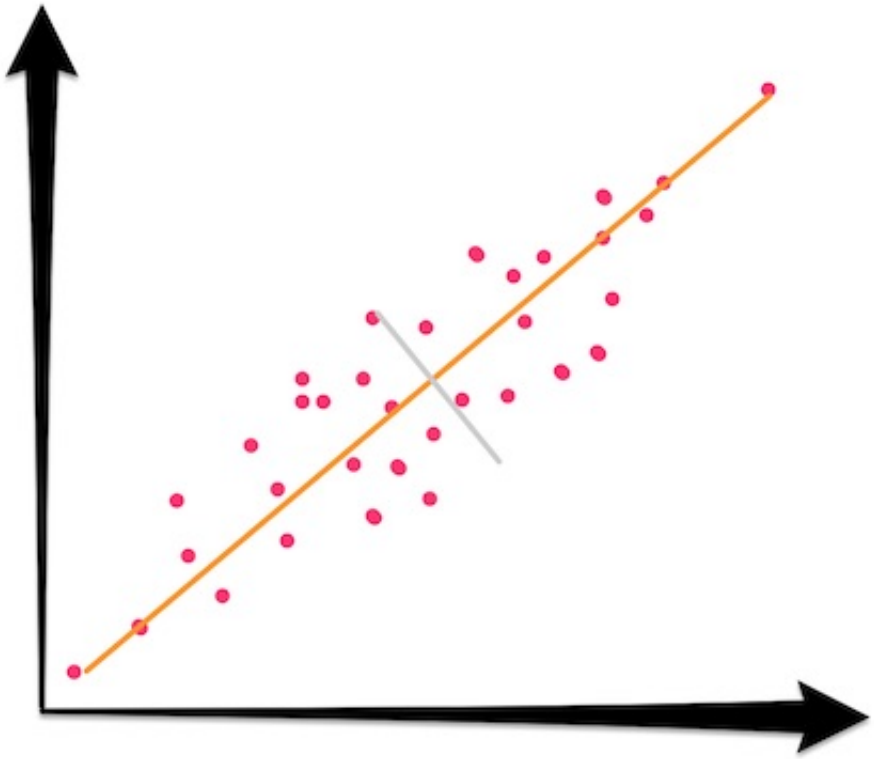
Having collected all that data, even though you asked 100 questions, what do you think truly is being measured by the survey? Overall, it is the collective movie preference per person. You could attempt to solve for this manually in a supervised way, by break down movies into well-known genres: + Action + Adventure + Comedy + Crime & Gangster + Drama + Historical + Horror + Musicals + Science Fiction + War + Western + etc.

Being unsupervised, PCA doesn't have access to these genre labels. In fact, it doesn't have or care for any labels whatsoever. This is important because it's entirely possible there wasn't a single western movie in your list of 1000 films, so it would be inappropriate and strange for PCA to derive a 'Western' principal component feature. By using PCA, rather than you creating categories manually, it discovers the natural categories that exist in your data. It can find as many of them as you tell it to, so long as that number is less than the original number of features you provided, and as long as you have enough samples to support it. The groups it finds are the principal components, and they are the best possible, linearly independent combination of features that you can use to describe your data.

Being unsupervised, PCA doesn't have access to these genre labels. In fact, it doesn't have or care for any labels whatsoever. This is important because it's entirely possible there wasn't a single western movie in your list of 1000 films, so it would be inappropriate and strange for PCA to derive a 'Western' principal component feature. By using PCA, rather than you creating categories manually, it discovers the natural categories that exist in your data. It can find as many of them as you tell it to, so long as that number is less than the original number of features you provided, and as long as you have enough samples to support it. The groups it finds are the principal components, and they are the best possible, linearly independent combination of features that you can use to describe your data.

One warning is that again, being unsupervised, PCA can't tell you exactly know what the newly created components or features mean. If you're interested in how to interpret your principal components, we've included two sources in the dive deeper section to help out with that and highly recommend you explore them.

Once you've reduced your dataset's dimensionality using PCA to best describe its variance and linear structure, you can then transform your movie questionnaire dataset from its original



pca-projection-shadow

[1000, 100] feature-space into the much more comfortable, principal component space, such as [1000, 10]. You can visualize your samples in this new space using an Andrew's plot, or scatter plot. And finally, you can base the rest of your analysis on your transformed features, rather than the original 100 feature dataset.

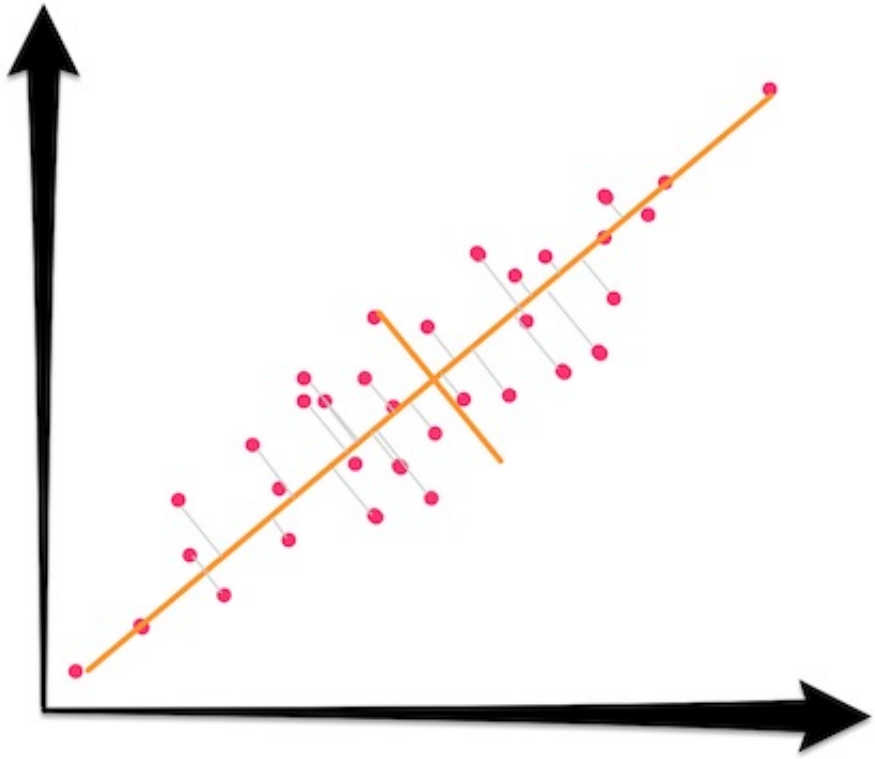
PCA is a very fast algorithm and helps you vaporizes redundant features, so when you have a high dimensionality dataset, start by running PCA on it and then visualizing it. This will better help you understand your data before continuing.

Projecting a Shadow By transforming your samples into the feature space created by discarding under-prioritized features, a lower dimensional representation of your data, also known as **shadow** or **projection** is formed. In the shadow, some information has been lost—it has fewer features after all. You can actually visualize how much information has been lost by taking each sample and moving it to the nearest spot on the projection feature space. In the following 2D dataset, the orange line represents the principal component direction, and the gray line represents the second principal component. The one that's going to get dropped:

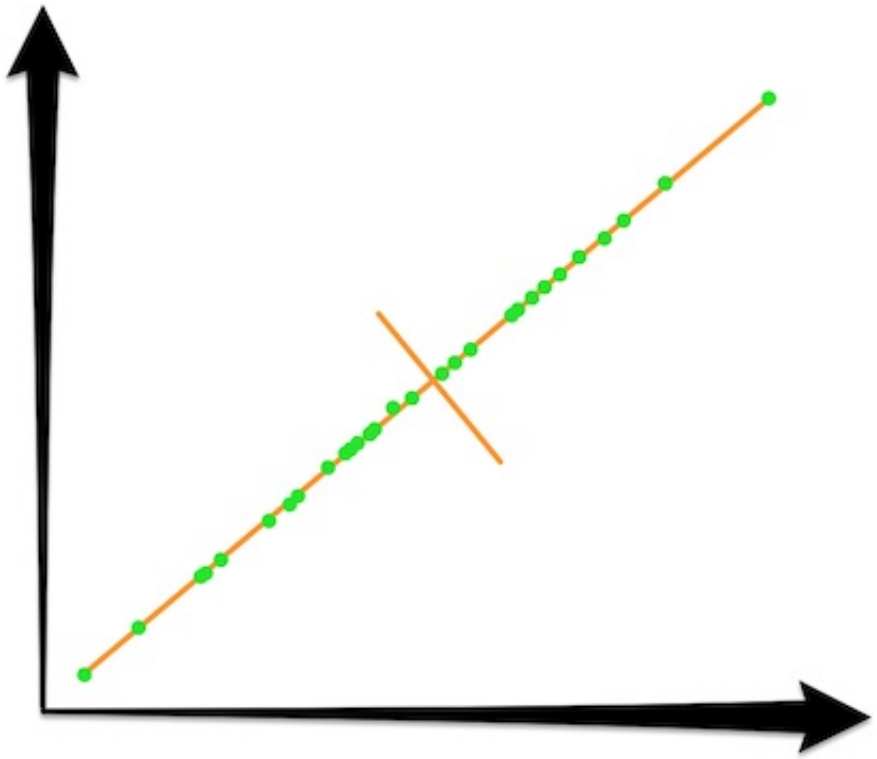
By dropping the gray component above, the goal is to project the 2D points onto 1D space. Move the original 2D samples to their closest spot on the line:

Once you've projected all samples to their closest spot on the major principal component, a shadow, or lower dimensional representation has been formed:

The summed distances traveled by all moved samples is equal to the total information lost by the projection. An an ideal situation, this lost information should be dominated by highly redundant features and random noise.



pca-projection-shadow2



pca-projection-shadow3

SciKit-Learn and PCA To get started, import PCA from `sklearn.decomposition` and then create a new instance of the model setting the `n_components parameter` to the number of dimensions you wish to keep. This value has to be less than or equal to the number of features in your original dataset, since each computed component is a linear combination of your original features:

```
In [6]: import pandas as pd
        from sklearn.decomposition import PCA

        df = pd.read_csv("Datasets/students.data")

        pca = PCA(n_components=2)
        pca.fit(df)
        print (pca)

        T = pca.transform(df)

        print (df.shape) # (430, 6) - 430 Student survey responses, 6 questions..
        print (T.shape) # (430, 2) - 430 Student survey responses, 2 principal comp
```

```
PCA(copy=True, n_components=2, whiten=False)
(649, 29)
(649, 2)
```

Once you've fit the model against your dataframe, you can use it to transform your dataset's observations (or any other observation that share its feature space) into the newly computed, principal component feature space with the `.transform()` method. This transformation is bidirectional, so you can recover your original feature values using `.inverse_transform()` so long as you don't drop any components. If even one component was removed, then after performing the inverse transformation back to the regular feature space, there will be some signs of information loss proportional to which component was dropped.

There are a few other interesting model attribute that SciKit-Learn exposes to you after you've trained your PCA model with the `.fit()` method:

- **`components_`** These are your principal component vectors and are linear combinations of your original features. As such, they exist within the feature space of your original dataset.
- **`explained_variance_`** This is the calculated amount of variance which exists in the newly computed principal components.
- **`explained_variance_ratio_`** Normalized version of `explained_variance_` for when your interest is with probabilities.

PCA Gotchas! To use PCA effectively, you should be aware of its weaknesses. The first is that **PCA is sensitive to the scaling of your features**. PCA maximizes variability based off of variance (the average squared differences of your samples from the mean), and then projects your original data on these directions of maximal variance. If your has a feature with a large variance and others with small variances, PCA will load on the larger variance feature. Being a linear transformation, it will rotate and reorient your feature space so it diverts as much of the variance of the larger-variance feature (and some of the other features' variances), into the first few principal components.

When it does this, a feature might go from having little impact on your transformation to totally dominating the first principal component, or vice versa. **Standardizing your variables ahead of time is the way to free your PCA results of such scaling issues**. *The cases where you should not use standardization are when you know your feature variables, and thus their importance, need to respect the specific scaling you've set up*. In such a case, PCA would be used on your raw data.

PCA is extremely fast, but for very large datasets it might take a while to train. All of the matrix computations required for inversion, eigenvalue decomposition, etc. boggle it down. Since most real-world datasets are typically very large and will have a level of noise in them, razor-sharp, machine-precision matrix operations aren't always necessary. If you're willing to sacrifice a bit of accuracy for computational efficiency, *SciKit-Learn offers a sister algorithm called **RandomizedPCA** that applies some approximation techniques to speed up large-scale matrix computation*. You can use this for your larger datasets.

The last issue to keep in mind is that PCA is a linear transformation only! In graphical terms, it can rotate and translate the feature space of your samples, but will not skew them. *PCA will only, therefore, be able to capture the underlying linear shapes and variance within your data and cannot discern any complex, nonlinear intricacies. For such cases, you will have to make use different dimensionality reduction algorithms, such as **Isomap***.

Knowledge checks

Review Question 1 1/1 point (graded)
Please complete the sentence so that it makes the most sense:

Principal component analysis...

- Requires you have labeled features to use as a metric for determining which features are most important
- Is a dimensionality reduction technique that builds a simpler, non-linear projection or 'shadow' of your dataset
- Asserts you have more features than samples so you can avoid the curse of dimensionality and the matrix math works out
- Ensures each newly computed feature is orthogonal to all previously computed ones, minimizing overlaps

Answer: **D** ##### Review Question 2 1 point possible (graded)
Which of these statements is wrong?

- PCA can be used to discover the underlying features being assessed by a dataset
- The results of PCA depend on the scaling of your data, so having a feature with units of 'light-years' and another feature with units of 'GHz' may be disastrous
- When applied to non-linear data, PCA generally isn't as effective as when applied to linear data
- Since PCA is sensitive to feature scaling, if you have a feature that is a linear transformation of the other, e.g. feature2 = 10 * feature1, then both features will be ignored

Answer: **D**

0.1.2 PCA Lab

Assignment 1

Lab Assignement 1 In this assignment, you're going to experiment with a real life armadillo sculpture scanned using a Cyberware 3030 MS 3D scanner at Stanford University. The sculpture is available as part of their 3D Scanning Repository, and is a very dense 3D mesh consisting of 172974 vertices! The mesh is available for you, located at /Module4/Datasets/**stanford_armadillo.ply**. It is *not* a Python file, so don't attempt to load it with a text editor!

Open up the Module4/**assignment1.py** starter code and read through it carefully. You will notice the use of a new library, Plyfile. This library loads up the 3D binary mesh for you. The mesh is further converted into a Pandas dataframe for your ease of manipulation. Complete the following tasks:

1. Before changing any of the code, go ahead and execute assignment1.py. You should see the 3D armadillo. Your goal is to reduce its dimensionality from three to two using PCA to cast a shadow of the data onto its two most important principal components. Then render the resulting 2D scatter plot.
2. Fill out the proper code in the `do_PCA()` and `do_RandomizedPCA()` methods. Be sure to **return** the result of your transformation! You may even want to read the SciKit-Learn documentation on `.transform()`, just for future reference so you know what data type comes out of it.
3. Re-run the application! Then, answer the questions below:

armadillo

Lab Question 1 1 point possible (graded)
The first time you see the armadillo in 3D, what direction was its face pointing towards?

- Left, Towards the negative X-Axis
- Up, Towards the positive Z-Axis
- Right, Towards the positive X-Axis
- Down, Towards the negative Z-Axis

Answer: **A** ##### Lab Question 2 2 point possible (graded)
Were you able to discern any **visual** differences between the transformed PCA results and the transformed RandomizedPCA results?

- Yes, the RandomizedPCA version was no longer even recognizable as an armadillo
- Yes, the RandomizedPCA version was a lot less true to the original than the regular PCA version
- Yes, but it wasn't a lot... just minor differences
- No, they pretty much looked the same to me

Which executed faster, RandomizedPCA or PCA?

- PCA
- RandomizedPCA

Answer: **RandomizedPCA**

Lab Assignement 2 In Lab Assignment 1, you applied PCA to a dataset generated by 3D-scanning an actual sculpture. Real life 3D objects are a good segue to PCA, since it's **fun** to see its effects on a dataset we can see and touch. Another benefit is that all three spatial dimensions, **x**, **y**, and **z**, each measure the same unit-length relative to one another, so no extra consideration need be made to account for PCA's weakness of requiring feature scaling.

But now the **fun** is over. Gaining some practical experience with real-world datasets, which rarely allot you the luxury of having features all on the same scale, will help you see how critical feature scaling is to PCA. In this lab, you're going to experiment with a subset of UCI's Chronic Kidney Disease data set, a collection of samples taken from patients in India over a two month period, some of whom were in the early stages of the disease. The starter code over at /Module4/**assignment2.py**.

1. Start by looking through the attribute information on the dataset website. Whenever you're given a dataset, the first thing you should do is find out as much about it as possible, both by reading up on any metadata, as well as by prodding through the actual data. Particularly, pay attention to what the docs say about these three variables: **bgr**, **rc**, and **wc**.
2. Load up the **`kidney_disease.csv`** dataset from the /Module4/Datasets/ directory, and drop **all rows** that have *any* nans. You're probably already a pro at doing that by now. In addition to getting rid of nans, did you know that the `.dropna()` method (upon completion) also automatically re-checks your features and assigns them an appropriate inferred data types?
3. Use an appropriate indexer command to select only the following columns: **bgr**, **rc**, and **wc**. Or alternatively, you can drop every other column, but it's probably easier to just use an indexer to select the one's you wish to keep.

4. Do a check of your dataframe's dtypes. Anything that didn't make it to the right type, you may want to investigate. Look through the data and identify *why* the conversion failed. These types of problems often arise when you aren't in control of how your data is organized. Luckily the issue isn't too bad so once you've identified it, you can fix it through simple numeric coercion.
5. Print the variance of your dataset, as well as a `.describe()` printout.
6. Reduce your dataset to two principal components by run it through PCA, then check out the resulting visualization.

Lab Question 1 1 point possible (graded)
The first time you see the armadillo in 3D, what direction was its face pointing towards?

12. Serum Creatinine (numerical) sc in mgs/dl
13. Serum (numerical) sod in mEg/L
14. Potassium (numerical) pot in mEg/L
15. Hemoglobin (numerical) hemo in gms

Having reviewed the dataset metadata on its website, what are the units of the **wc**, White Blood Cell Count feature? An example of where units are defined is shown above. NOTE: In case the UCI site is down, here is a mirror.

- mm/Hg
- mgs/dl
- mEq/L
- cells/cumm
- gms

Answer: ** **

Lab Question 2 2 point possible (graded)
Why did the `.dropna()` method fail to convert all of the columns to an appropriate numeric format?

- It actually did successfully convert them
- There were a few erroneous leading tab / whitespace characters
- There were a few erroneous trailing tab / whitespace characters
- The dataset comma offset was incorrect in a few rows causing nans to move into the next column
- Yes, the RandomizedPCA version was no longer even recognizable as an armadillo
- Yes, the RandomizedPCA version was a lot less true to the original than the regular PCA version
- Yes, but it wasn't a lot... just minor differences
- No, they pretty much looked the same to me

Sort the features below from the largest to the smallest variance amount.

- bgr, rc, wc
- bgr, wc, rc
- rc, wc, bgr
- rc, bgr, wc
- wc, bgr, rc
- wc, rc, bgr

Answer: ** **

You're almost there! The last thing you have to do, and the purpose of this lab really, is to see how feature scaling alters your PCA results.

1. Make a backup of your **assignment2.py** file for safe keeping.
2. Change the line that reads: `python scaleFeatures = False` So that is now reads: `python scaleFeatures = False`.
3. Also take a look inside of **assignment2_helper.py**. There are some *important* notes in there about what SKLearn's *star transform()* methods do, and why they do it. You will need to know this information for future labs!
4. Re-run your assignment and then answer the questions below:

Answer: ** ** —

Lab Questions (Continued) 2 points possible (graded)
Did scaling your features affect their variances at all? + Yes + No
Answer: ** **
After scaling your features, are the green patients without chronic kidney disease more cleanly separable from the red patients with chronic kidney disease?

- They are less separable
- There isn't much change
- They are more separable

Answer: ** **

```
In [ ]:
```