Pricipal Coordinate Analysis (PCA)

At times, when you're working with complex data, you have so many variables that you're not sure where to start...It's in these cases, when you have many variables to consider that I often turn to PCA.

In these situations of variable-overload, I often struggle to understand the relationships between each variable. Am I overfitting a model -- its hard to tell with so many variables? I'm also often concerned that I may be violating assumptions of a model, especially that featurse are independent.

PCA helps to reduce the dimension of your feature space. By reducing the dimension of your feature space, you have fewer relationships between variables to consider and you are less likely to overfit your model. (Note: This doesn't immediately mean that overfitting, etc. are no longer concerns!)

Reducing the dimension of the feature space is called more officially "dimensionality reduction." There are many ways to achieve dimensionality reduction, but most of these techniques fall into one of two classes:

- Feature Elimination
- Feature Extraction

Feature elimination is what it sounds like: we reduce the feature space by eliminating features. Instead of considering all 100 features, we'll only use 10. Advantages of feature elimination methods include simplicity and maintaining interpretability of your variables. As a disadvantage, though, you gain no information from those variables you've dropped (and they may be important!).

Feature extraction, however, doesn't run into this problem. Say we have ten independent variables. In feature extraction, we create ten "new" independent variables, where each "new" independent variable is a combination of each of the ten "old" independent variables. However, we create these new independent variables in a specific way and order these new variables by how well they predict our dependent variable. In the Statquest video, these were the fitted eigenvectors he discussed.

Principal component analysis is a technique for feature extraction — so it combines our input variables in a specific way, then we can drop the "least important" variables while still retaining the most valuable parts of all of the variables! As an added benefit, each of the "new" variables after PCA are all independent of one another. This is a benefit because the assumptions of a linear model require our independent variables to be independent of one another. If we decide to fit a linear regression model with these "new" variables, this assumption will necessarily be satisfied.

When should PCA be used?

- Do you want to reduce the number of variables, but aren't able to identify variables to completely remove from consideration?
- Do you want to ensure your variables are independent of one another?
- Are you comfortable making your independent variables less interpretable?

If you answered "yes" to all three questions, then PCA is a good method to use. If you answered "no" to question 3, you should not use PCA.

Content based on https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c (https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c)

Dataset for PCA

We are going to be working with the digital images of tumor cells from our previous SVM tutorial. You'll remember that we have tumor images to predict whether the tumors are malignant or benign.

For each image, ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter) b) texture (standard deviation of gray-scale values) c) perimeter d) area e) smoothness (local variation in radius lengths) f) compactness (perimeter^2 / area - 1.0) g) concavity (severity of concave portions of the contour) h) concave points (number of concave portions of the contour) i) symmetry j) fractal dimension ("coastline approximation" - 1)

Additionally, the mean, standard error and “worst” or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features.

Let's start with bringing int he dataset and taking a quick look at it....

In [1]:
```
%matplotlib inline

import pandas as pd

dataset = pd.read_csv('../data/cancer.csv')
dataset.head()
```

Out[1]:

	ID	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_m
0	842302	M	17.99	10.38	122.80	1001.0	0.11
1	842517	M	20.57	17.77	132.90	1326.0	0.08
2	84300903	M	19.69	21.25	130.00	1203.0	0.10
3	84348301	M	11.42	20.38	77.58	386.1	0.14
4	84358402	M	20.29	14.34	135.10	1297.0	0.10

5 rows × 32 columns

In [2]:
```
dataset.shape
```

Out[2]:
```
(569, 32)
```

Let's select the mean, errors, and worst columns as separate dataframes. We've done this several different ways, usign iloc, using specific column names. This method is probably one I use a lot.

In [5]:
```
print(list(dataset.columns[2:12]))
```

```
['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_me
an', 'compactness_mean', 'conc_mean', 'conc_points_mean', 'symmetry_mean', 'f
ractral_mean']
```

In [6]:
```
feature_mean = list(dataset.columns[2:12])
```

In [7]:
```
dataset_mean = dataset[feature_mean]
```

In [8]:
```
dataset_mean.corr(method='pearson')
```

Out[8]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
radius_mean	1.000000	0.323782	0.997855	0.987357	0.170581
texture_mean	0.323782	1.000000	0.329533	0.321086	-0.023389
perimeter_mean	0.997855	0.329533	1.000000	0.986507	0.207278
area_mean	0.987357	0.321086	0.986507	1.000000	0.177028
smoothness_mean	0.170581	-0.023389	0.207278	0.177028	1.000000
compactness_mean	0.506124	0.236702	0.556936	0.498502	0.659123
conc_mean	0.676764	0.302418	0.716136	0.685983	0.521984
conc_points_mean	0.822529	0.293464	0.850977	0.823269	0.553695
symmetry_mean	0.147741	0.071401	0.183027	0.151293	0.557775
fractral_mean	-0.311631	-0.076437	-0.261477	-0.283110	0.584792

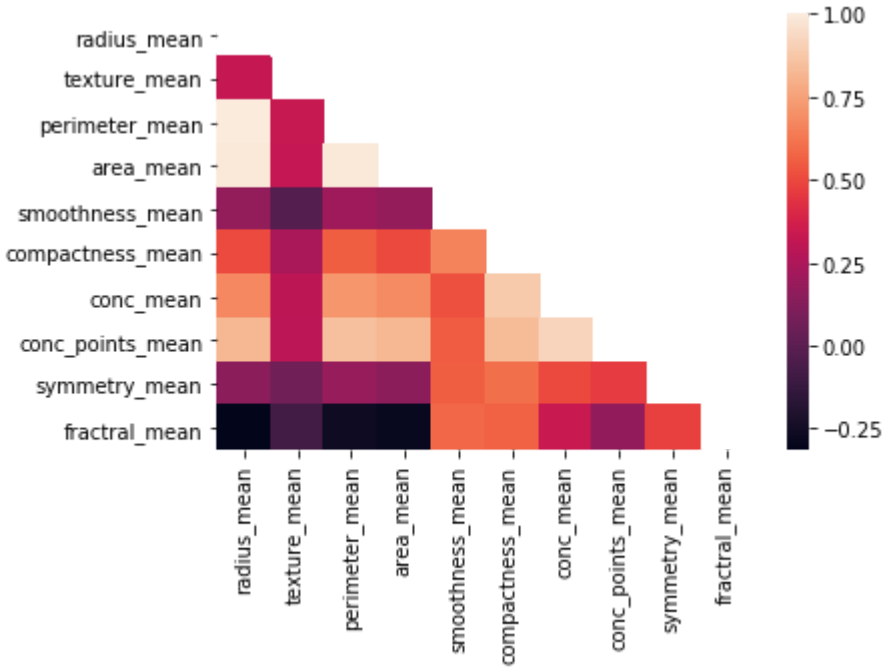
Make a correlation plot of the average values.

In [13]:
```
import seaborn as sns
import numpy as np
import matplotlib

mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
corr = dataset_mean.corr(method='pearson')

sns.heatmap(corr, mask=mask)
```

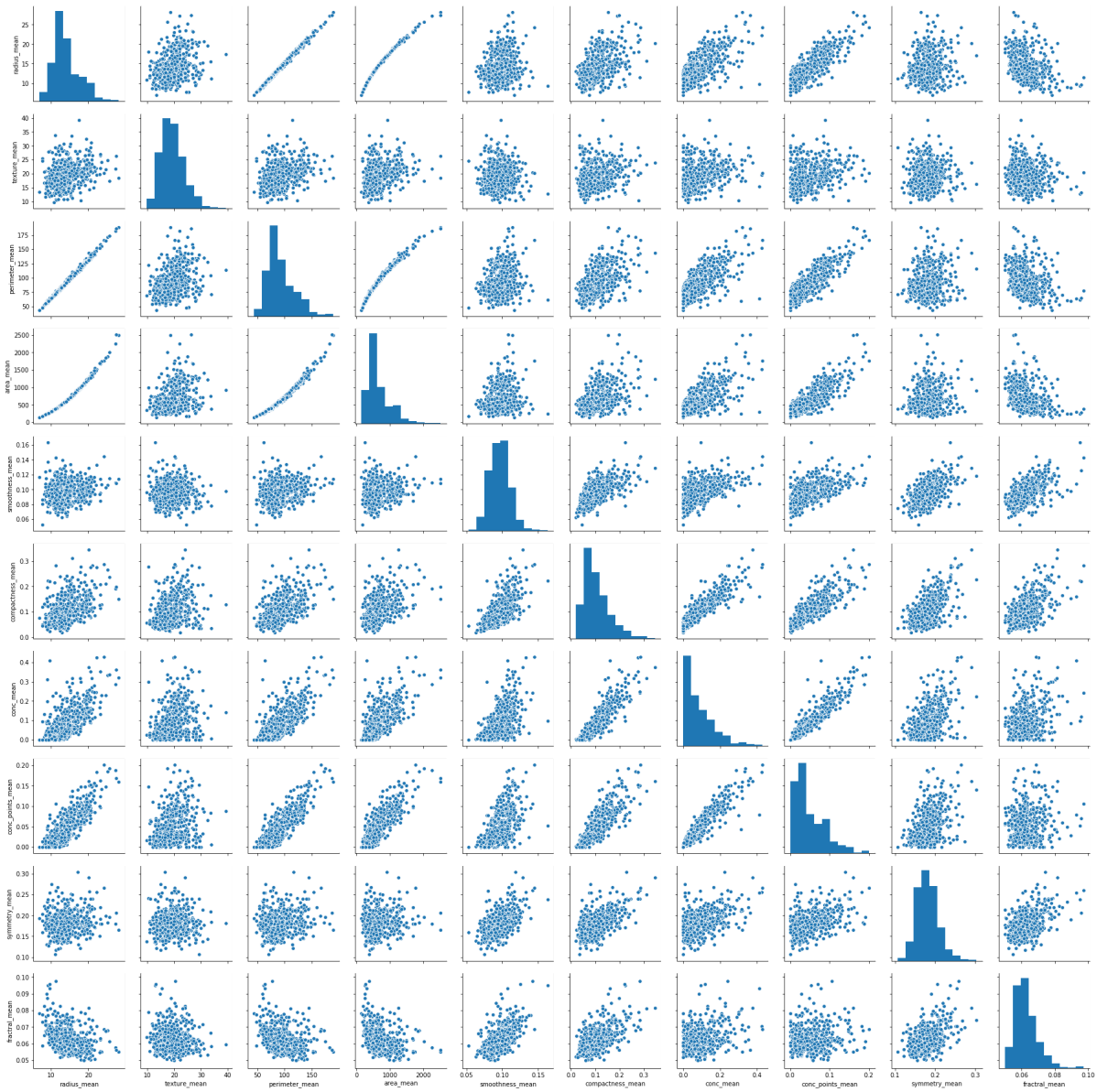
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x2cdd6e02390>



On your Own: Make at least one other plot to explore the correlations within this dataset.

In [25]:
```
sns.pairplot(dataset_mean)
```

Out[25]: <seaborn.axisgrid.PairGrid at 0x2cdd9c447b8>



Scaling data is important for PCAs

Feature scaling through standardization (or Z-score normalization) can be an important preprocessing step for many machine learning algorithms. Standardization involves rescaling the features such that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one.

While many algorithms (such as SVM, K-nearest neighbors, and logistic regression) require features to be normalized. Principle Component Analysis (PCA) is a prime example of when normalization is also important.

In PCA we are interested in the components that maximize the variance. If one component (e.g. human height) varies less than another (e.g. weight) because of their respective scales (meters vs. kilos), PCA might determine that the direction of maximal variance more closely corresponds with the 'weight' axis, if those features are not scaled. As a change in height of one meter can be considered much more important than the change in weight of one kilogram, this is clearly incorrect.

Source Documentation (https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html)

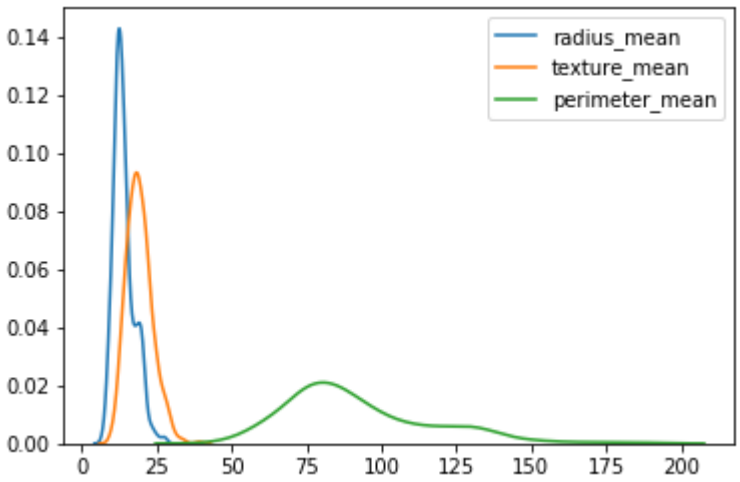
In [28]:
```
from sklearn.preprocessing import StandardScaler

X_data = dataset.iloc[:, 2:32]
Y_data = dataset.iloc[:,1]

scaled_data = StandardScaler()
scaled_X = scaled_data.fit_transform(X_data)
```

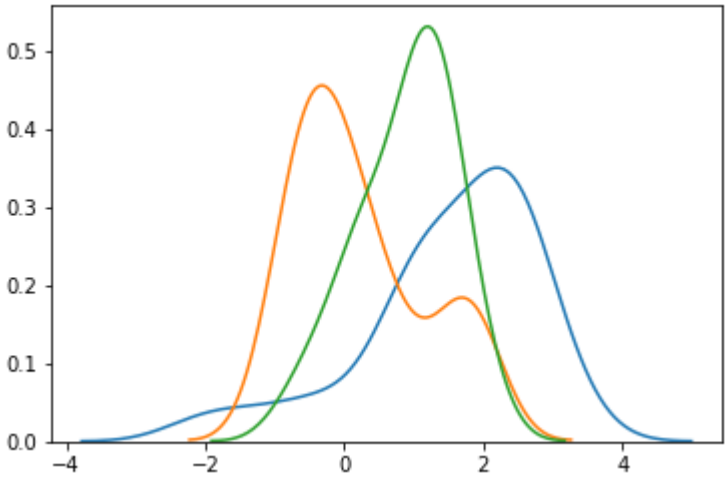
In [30]:
```
sns.kdeplot(X_data.iloc[:,0])
sns.kdeplot(X_data.iloc[:,1])
sns.kdeplot(X_data.iloc[:,2])
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x2cddc1740f0>



In [33]:
```
sns.kdeplot(scaled_X[0])
sns.kdeplot(scaled_X[1])
sns.kdeplot(scaled_X[2])
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x2cddef0ca20>



In the code above, what do the arguments 'stratify' and 'random_state' specify and when might you use them?

In [ ]:
```
```

Let's take a look at our trained dataset and how much was explained by each principle coordinate.

In [34]:
```
from sklearn.decomposition import PCA
```

In [39]:
```
pcal = PCA(n_components =6)
pcal.fit(scaled_X)
trained_pcal = pcal.transform(scaled_X)
```

In [40]:
```
trained_pcal.shape
```

Out[40]: (569, 6)

In [49]:
```
pc_df = pd.DataFrame(data=trained_pcal, columns = ['PC1', 'PC2', 'PC3', 'PC4',
'PC5', 'PC6'])
pc_df['Cluster'] = Y_data
pc_df
```

Out[49]:

	PC1	PC2	PC3	PC4	PC5	PC6	Cluster
0	9.192837	1.948583	-1.123166	3.633731	-1.195110	1.411424	M
1	2.387802	-3.768172	-0.529293	1.118264	0.621775	0.028656	M
2	5.733896	-1.075174	-0.551748	0.912083	-0.177086	0.541452	M
3	7.122953	10.275589	-3.232790	0.152547	-2.960878	3.053422	M
4	3.935302	-1.948072	1.389767	2.940639	0.546747	-1.226495	M
5	2.380247	3.949929	-2.934877	0.941037	-1.056042	-0.451039	M
6	2.238883	-2.690031	-1.639913	0.149340	0.040360	-0.128949	M
7	2.143299	2.340244	-0.871947	-0.127043	-1.427437	-1.257039	M
8	3.174924	3.391813	-3.119986	-0.601297	-1.522290	0.559545	M
9	6.351747	7.727174	-4.341916	-3.375202	1.710263	-0.723908	M
10	-0.810414	-2.659275	-0.488830	-1.672567	0.275812	0.127332	M
11	2.651100	0.066568	-1.526455	0.051262	0.331951	0.764867	M
12	8.185034	2.700976	5.730231	-1.112257	1.043470	2.594571	M
13	0.342126	-0.968279	1.717172	-0.595003	0.468011	1.007660	M
14	4.342379	4.861083	-2.816116	-1.454557	1.290063	-0.349716	M
15	4.075656	2.977061	-3.125274	-2.458071	-0.408342	0.495778	M
16	0.230055	-1.564758	-0.802519	-0.650583	-0.494711	-0.762191	M
17	4.418011	1.418670	-2.270319	-0.186272	-1.423861	-0.752489	M
18	4.948704	-4.114334	-0.314749	-0.088207	-0.056715	-1.137689	M
19	-1.237063	-0.188215	-0.593283	1.596346	-0.442154	-0.048637	B
20	-1.578161	0.572808	-1.801447	1.125276	-0.395270	0.430841	B
21	-3.557336	1.662950	0.451187	2.073765	-0.490746	-0.770068	B
22	4.733211	3.304964	-1.466537	2.041150	-0.026220	3.023044	M
23	4.208524	-5.128367	-0.752402	-0.862710	-0.470968	-0.595841	M
24	4.949632	-1.543752	-1.713194	0.046759	-1.739230	-0.812880	M
25	7.098563	2.018610	-0.029010	2.587951	-2.039340	1.184201	M
26	3.510263	2.171625	-3.894546	-1.295760	-0.068218	0.787358	M
27	3.064054	-1.876552	2.581748	0.128484	-0.116665	-0.907595	M
28	4.007264	0.537242	-2.761626	-1.898388	-0.525673	0.192849	M
29	1.715310	-1.523705	0.146187	1.911386	0.536123	0.036260	M
...	...	...	...	...	...	...	...
539	-1.142832	5.599458	1.301037	-2.188250	0.765575	-1.159227	B
540	-1.665475	2.389618	1.502249	0.875951	0.484546	-1.189518	B
541	1.011712	1.092390	-0.632698	-1.758518	1.046826	0.425895	B
542	-1.300930	-1.821415	0.373307	-1.848169	0.199627	0.765754	B

	PC1	PC2	PC3	PC4	PC5	PC6	Cluster
543	-2.373429	-1.681576	0.384528	-3.016729	-0.038247	0.068814	B
544	-1.665871	-0.213963	-0.148072	-0.197052	0.556480	-0.931582	B
545	-1.927678	-1.137740	0.478202	-1.157500	-0.202342	0.114898	B
546	-4.237217	0.184272	-0.326418	0.588303	-0.704627	-0.146091	B
547	-2.677871	2.315793	-0.053848	0.340450	0.450648	-0.678348	B
548	-3.836498	0.496250	0.923240	-0.551872	0.080036	-0.112669	B
549	-2.551440	0.228330	1.414178	-1.970790	-1.456325	0.855522	B
550	-4.694923	-0.767478	1.543965	-0.779019	-0.784897	0.851373	B
551	-2.025037	1.261242	0.504926	-1.135527	-0.114263	1.784777	B
552	-2.895948	-1.451636	0.780546	-2.970448	-0.357150	-0.583203	B
553	-3.502201	1.800832	2.766457	-0.866307	-0.625365	0.412270	B
554	-2.153904	-0.830069	0.564797	-3.011756	0.565825	-0.550403	B
555	-2.055084	1.616459	1.838959	-3.113535	0.211043	-1.641806	B
556	-3.877290	1.084255	1.859944	-0.433740	-1.687439	-0.814893	B
557	-4.063862	0.122168	3.238773	-3.469183	-2.241908	0.174909	B
558	-0.098667	-0.213560	0.388929	-1.012711	2.989068	0.012452	B
559	-1.089376	1.292848	1.429379	-3.372136	0.947342	-1.725870	B
560	-0.481771	-0.178020	1.032108	-2.010280	0.327103	-0.992062	B
561	-4.870310	-2.131106	3.414189	-5.133988	-0.488921	-1.110792	B
562	5.917613	3.482637	-3.262792	-3.917585	1.716382	1.013599	M
563	8.741338	-0.573855	0.897090	0.385150	0.679043	-0.300753	M
564	6.439315	-3.576817	2.459487	1.177314	-0.074824	-2.375193	M
565	3.793382	-3.584048	2.088476	-2.506028	-0.510723	-0.246710	M
566	1.256179	-1.902297	0.562731	-2.089227	1.809991	-0.534447	M
567	10.374794	1.672010	-1.877029	-2.356031	-0.033742	0.567936	M
568	-5.475243	-0.670637	1.490443	-2.299157	-0.184703	1.617837	B

569 rows × 7 columns

In [42]:
```
pcal.explained_variance_ratio_
```

Out[42]:
```
array([0.44272026, 0.18971182, 0.09393163, 0.06602135, 0.05495768,
       0.04024522])
```

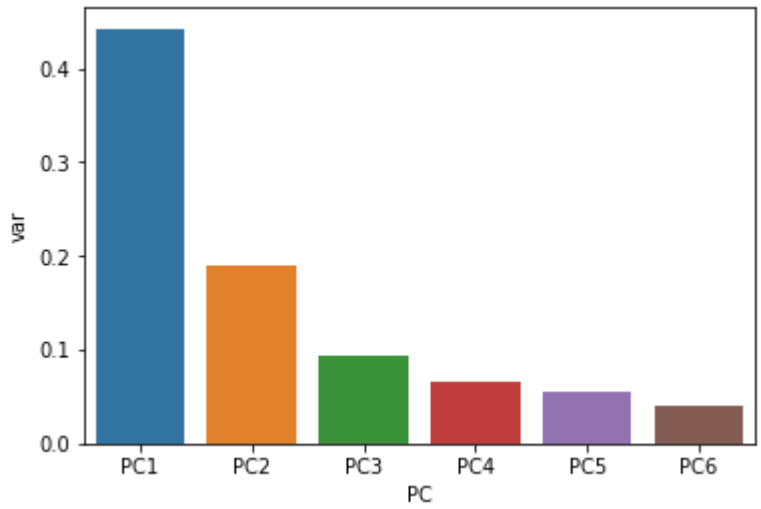
In [45]:
```
df = pd.DataFrame({'var':pcal.explained_variance_ratio_, 'PC':['PC1','PC2','PC3', 'PC4', 'PC5', 'PC6']})
df
```

Out[45]:

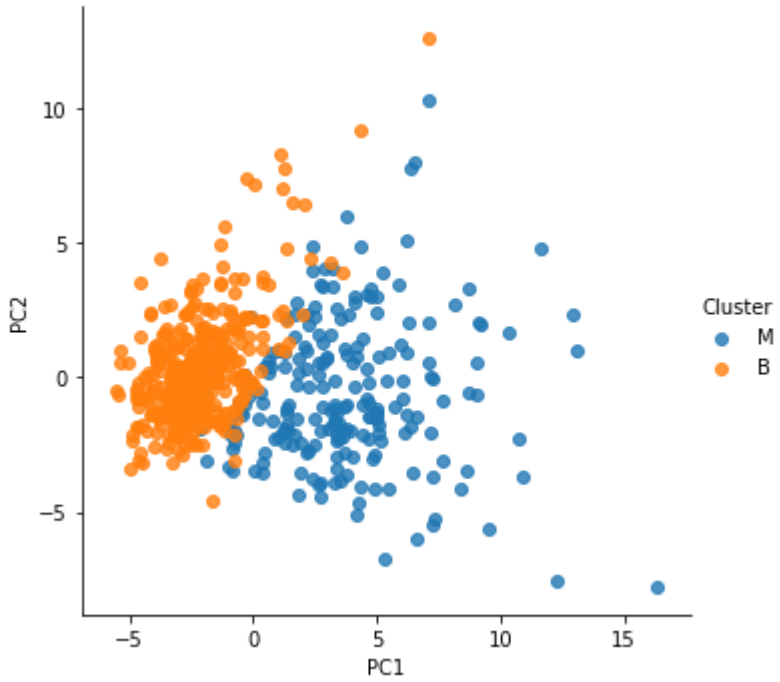
	var	PC
0	0.442720	PC1
1	0.189712	PC2
2	0.093932	PC3
3	0.066021	PC4
4	0.054958	PC5
5	0.040245	PC6

In [46]:
```
sns.barplot(x='PC', y='var', data=df)
```

Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2cddeab2d68>
```



In [50]:
```
p = sns.lmplot(x='PC1', y='PC2', data=pc_df, hue="Cluster", fit_reg=False, leg
end=True)
```



Some other tutorials (that are potentially useful):

1. PCA followed by regression: https://nirpyresearch.com/principal-component-regression-python/ (https://nirpyresearch.com/principal-component-regression-python/)
2. Manually doing a PCA, more math theory: https://sebastianraschka.com/Articles/2014_pca_step_by_step.html (https://sebastianraschka.com/Articles/2014_pca_step_by_step.html)
3. Generic PCA with a different dataset: https://medium.com/district-data-labs/principal-component-analysis-with-python-4962cd026465 (https://medium.com/district-data-labs/principal-component-analysis-with-python-4962cd026465)