PCA-Guided-Task-Cleaned_MJD

November 4, 2019

0.1 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

0.1.1 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. . . .

[2]:
```
%matplotlib inline

import pandas as pd

dataset = pd.read_csv('cancer.csv')
```

[3]:
```
dataset.head()
```

[3]:
```
         ID diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  conc_mean  conc_points_mean  ...  \
0          0.11840           0.27760     0.3001           0.14710  ...
```

```
1          0.08474           0.07864     0.0869           0.07017  ...
2          0.10960           0.15990     0.1974           0.12790  ...
3          0.14250           0.28390     0.2414           0.10520  ...
4          0.10030           0.13280     0.1980           0.10430  ...

   radius_worst  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0         25.38          17.33           184.60      2019.0            0.1622
1         24.99          23.41           158.80      1956.0            0.1238
2         23.57          25.53           152.50      1709.0            0.1444
3         14.91          26.50            98.87       567.7            0.2098
4         22.54          16.67           152.20      1575.0            0.1374

   compactness_worst  conc_worst  conc_points_worst  symmetry_worst  \
0             0.6656      0.7119             0.2654          0.4601
1             0.1866      0.2416             0.1860          0.2750
2             0.4245      0.4504             0.2430          0.3613
3             0.8663      0.6869             0.2575          0.6638
4             0.2050      0.4000             0.1625          0.2364

   fractral_worst
0         0.11890
1         0.08902
2         0.08758
3         0.17300
4         0.07678

[5 rows x 32 columns]
```

[4]:
```
dataset.shape
```

[4]:
```
(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.

[14]:
```
feature_mean = list(dataset.columns[2:12])
```

[15]:
```
dataset_mean = dataset[feature_mean]
```

[17]:
```
dataset_mean.corr(method='pearson')
```

[17]:
```
                   radius_mean  texture_mean  perimeter_mean  area_mean  \
radius_mean           1.000000      0.323782        0.997855   0.987357
texture_mean          0.323782      1.000000        0.329533   0.321086
perimeter_mean        0.997855      0.329533        1.000000   0.986507
area_mean             0.987357      0.321086        0.986507   1.000000
smoothness_mean       0.170581     -0.023389        0.207278   0.177028
compactness_mean      0.506124      0.236702        0.556936   0.498502
conc_mean             0.676764      0.302418        0.716136   0.685983
conc_points_mean      0.822529      0.293464        0.850977   0.823269
```

```
symmetry_mean         0.147741      0.071401         0.183027   0.151293
fractral_mean        -0.311631     -0.076437        -0.261477  -0.283110

                  smoothness_mean  compactness_mean  conc_mean  \
radius_mean              0.170581          0.506124   0.676764
texture_mean            -0.023389          0.236702   0.302418
perimeter_mean           0.207278          0.556936   0.716136
area_mean                0.177028          0.498502   0.685983
smoothness_mean          1.000000          0.659123   0.521984
compactness_mean         0.659123          1.000000   0.883121
conc_mean                0.521984          0.883121   1.000000
conc_points_mean         0.553695          0.831135   0.921391
symmetry_mean            0.557775          0.602641   0.500667
fractral_mean            0.584792          0.565369   0.336783

                  conc_points_mean  symmetry_mean  fractral_mean
radius_mean               0.822529       0.147741      -0.311631
texture_mean              0.293464       0.071401      -0.076437
perimeter_mean            0.850977       0.183027      -0.261477
area_mean                 0.823269       0.151293      -0.283110
smoothness_mean           0.553695       0.557775       0.584792
compactness_mean          0.831135       0.602641       0.565369
conc_mean                 0.921391       0.500667       0.336783
conc_points_mean          1.000000       0.462497       0.166917
symmetry_mean             0.462497       1.000000       0.479921
fractral_mean             0.166917       0.479921       1.000000
```

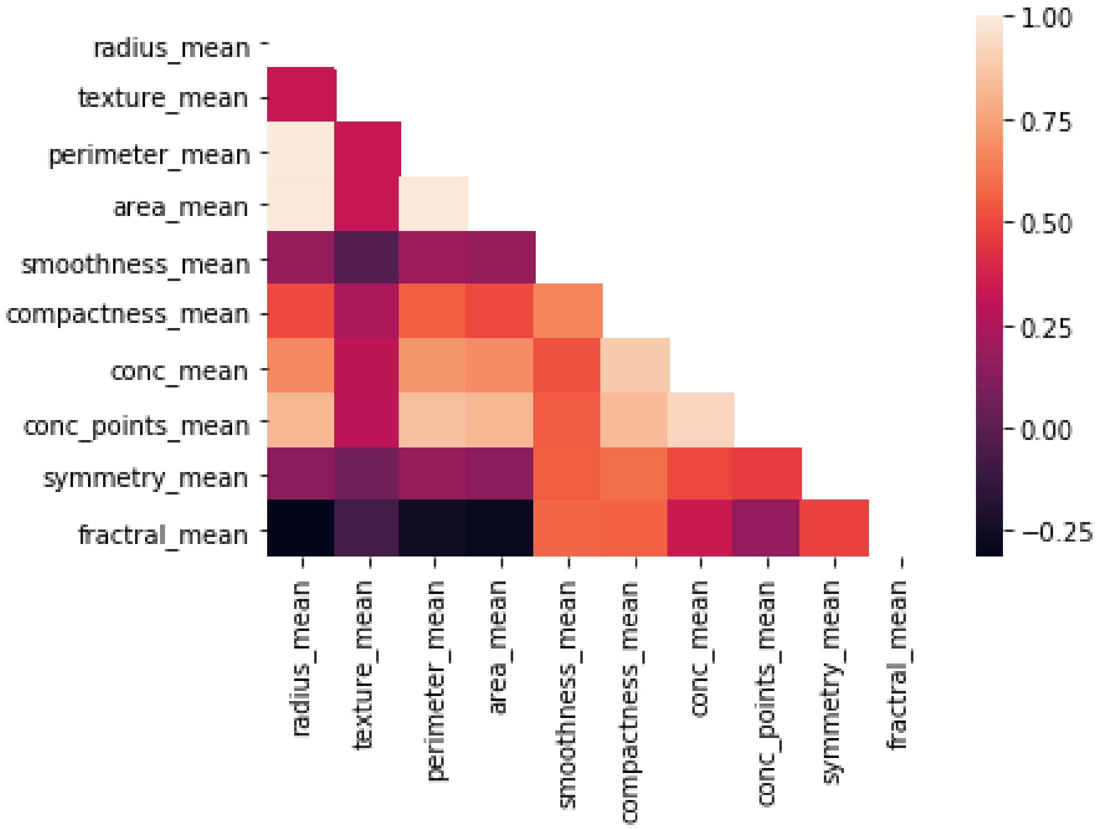
[22]:

```
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)
```

[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a7f8b38a58>
```



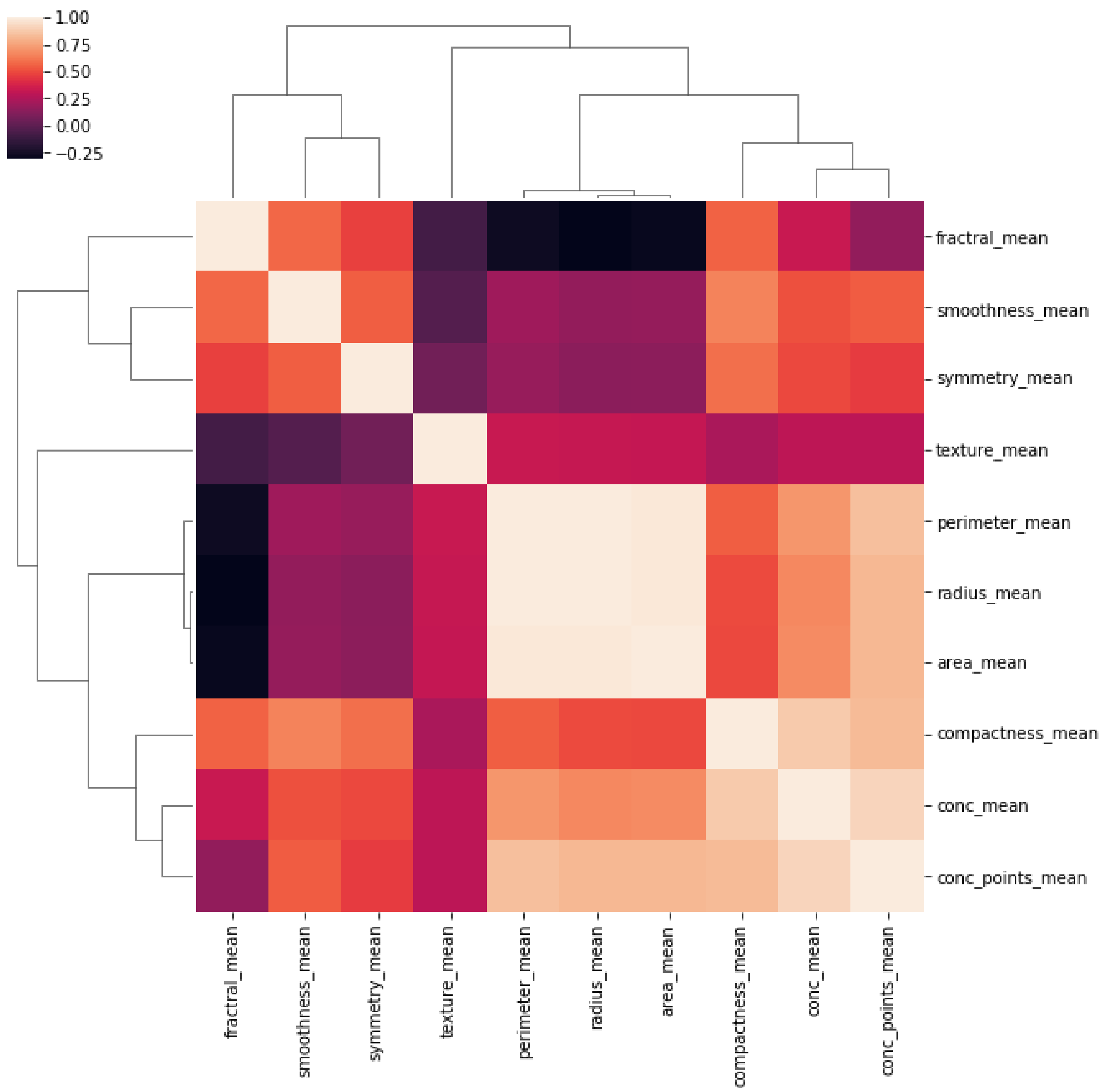
Make a correlation plot of the average values.

[ ]:

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

[35]:
```
sns.clustermap(corr)
```

[35]: `<seaborn.matrix.ClusterGrid at 0x1a7fff8eeb8>`



0.1.2 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

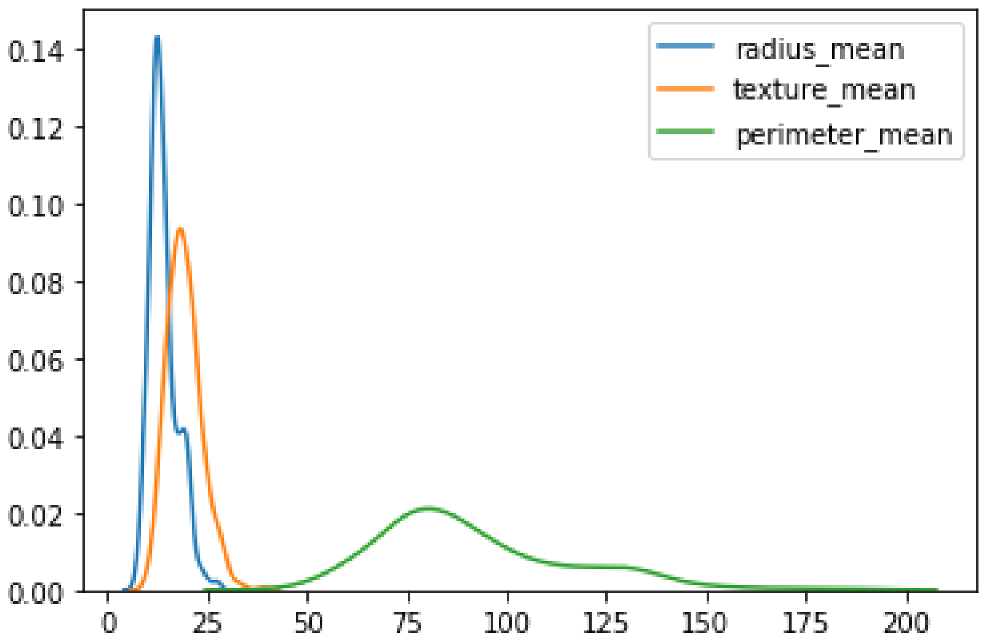
[41]:
```
from sklearn.preprocessing import StandardScaler

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

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

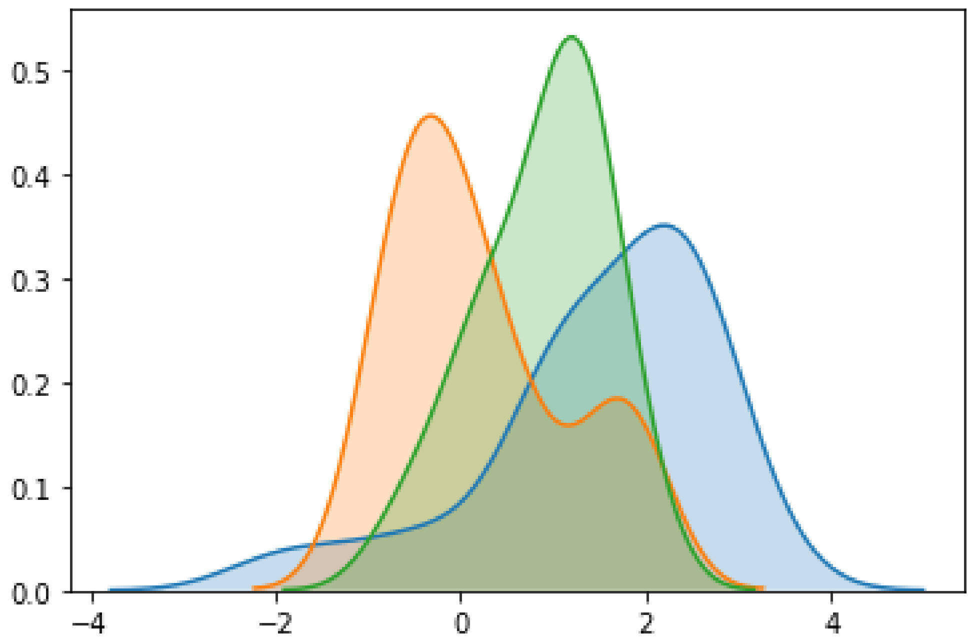
[43]:
```
sns.kdeplot(X_data.iloc[:,0])
sns.kdeplot(X_data.iloc[:,1])
sns.kdeplot(X_data.iloc[:,2])
```

[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a782a6feb8>
```



[47]:
```
sns.kdeplot(scaled_X[0], shade = True)
sns.kdeplot(scaled_X[1], shade = True)
sns.kdeplot(scaled_X[2], shade = True)
```

[47]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a782ac6d68>
```



[49]:
```python
from sklearn.decomposition import PCA
```

[50]:
```python
pca1 = PCA(n_components = 4)
pca1.fit(scaled_X)
trained_pca1 = pca1.transform(scaled_X)
```

[51]:
```python
trained_pca1.shape
```

[51]:
```
(569, 4)
```

[52]:
```python
pc_df = pd.DataFrame(data = trained_pca1, columns = ['PC1', 'PC2', 'PC3',␣
 ↪'PC4'])
```

[54]:
```python
pc_df
```

[54]:

	PC1	PC2	PC3	PC4
0	9.192837	1.948583	-1.123166	3.633731
1	2.387802	-3.768172	-0.529293	1.118264
2	5.733896	-1.075174	-0.551748	0.912082
3	7.122953	10.275589	-3.232790	0.152547
4	3.935302	-1.948072	1.389767	2.940639
5	2.380247	3.949929	-2.934877	0.941037
6	2.238883	-2.690031	-1.639913	0.149340
7	2.143299	2.340244	-0.871947	-0.127043
8	3.174924	3.391813	-3.119986	-0.601297
9	6.351747	7.727174	-4.341916	-3.375202
10	-0.810414	-2.659275	-0.488830	-1.672567
11	2.651100	0.066568	-1.526455	0.051261
12	8.185034	2.700976	5.730231	-1.112256

13	0.342126	-0.968279	1.717172	-0.595003
14	4.342379	4.861083	-2.816116	-1.454557
15	4.075656	2.977061	-3.125274	-2.458071
16	0.230055	-1.564758	-0.802519	-0.650583
17	4.418011	1.418670	-2.270319	-0.186272
18	4.948704	-4.114334	-0.314749	-0.088206
19	-1.237063	-0.188215	-0.593283	1.596346
20	-1.578161	0.572808	-1.801447	1.125276
21	-3.557336	1.662950	0.451187	2.073765
22	4.733211	3.304964	-1.466537	2.041150
23	4.208524	-5.128367	-0.752402	-0.862710
24	4.949632	-1.543752	-1.713194	0.046759
25	7.098563	2.018610	-0.029010	2.587951
26	3.510263	2.171625	-3.894546	-1.295760
27	3.064054	-1.876552	2.581748	0.128484
28	4.007264	0.537242	-2.761626	-1.898387
29	1.715310	-1.523705	0.146187	1.911386
..	...	...	...	...
539	-1.142832	5.599458	1.301037	-2.188249
540	-1.665475	2.389618	1.502249	0.875951
541	1.011712	1.092390	-0.632698	-1.758519
542	-1.300930	-1.821415	0.373307	-1.848169
543	-2.373429	-1.681576	0.384528	-3.016729
544	-1.665871	-0.213963	-0.148072	-0.197052
545	-1.927678	-1.137740	0.478202	-1.157500
546	-4.237217	0.184272	-0.326418	0.588303
547	-2.677871	2.315793	-0.053848	0.340450
548	-3.836498	0.496250	0.923240	-0.551872
549	-2.551440	0.228330	1.414178	-1.970790
550	-4.694923	-0.767478	1.543965	-0.779019
551	-2.025037	1.261242	0.504926	-1.135527
552	-2.895948	-1.451636	0.780546	-2.970448
553	-3.502201	1.800832	2.766457	-0.866307
554	-2.153904	-0.830069	0.564797	-3.011756
555	-2.055084	1.616459	1.838959	-3.113535
556	-3.877290	1.084255	1.859944	-0.433740
557	-4.063862	0.122168	3.238773	-3.469183
558	-0.098667	-0.213560	0.388929	-1.012710
559	-1.089376	1.292848	1.429379	-3.372136
560	-0.481771	-0.178020	1.032108	-2.010280
561	-4.870310	-2.131106	3.414189	-5.133988
562	5.917613	3.482637	-3.262792	-3.917586
563	8.741338	-0.573855	0.897090	0.385150
564	6.439315	-3.576817	2.459487	1.177314
565	3.793382	-3.584048	2.088476	-2.506028
566	1.256179	-1.902297	0.562731	-2.089227
567	10.374794	1.672010	-1.877029	-2.356031

```
568  -5.475243  -0.670637  1.490443 -2.299157

[569 rows x 4 columns]
```

[55]:
```python
pc_df['Cluster'] = Y_data
```

[56]:
```python
pc_df
```

[56]:
```
          PC1        PC2       PC3       PC4 Cluster
0    9.192837   1.948583 -1.123166  3.633731       M
1    2.387802  -3.768172 -0.529293  1.118264       M
2    5.733896  -1.075174 -0.551748  0.912082       M
3    7.122953  10.275589 -3.232790  0.152547       M
4    3.935302  -1.948072  1.389767  2.940639       M
5    2.380247   3.949929 -2.934877  0.941037       M
6    2.238883  -2.690031 -1.639913  0.149340       M
7    2.143299   2.340244 -0.871947 -0.127043       M
8    3.174924   3.391813 -3.119986 -0.601297       M
9    6.351747   7.727174 -4.341916 -3.375202       M
10  -0.810414  -2.659275 -0.488830 -1.672567       M
11   2.651100   0.066568 -1.526455  0.051261       M
12   8.185034   2.700976  5.730231 -1.112256       M
13   0.342126  -0.968279  1.717172 -0.595003       M
14   4.342379   4.861083 -2.816116 -1.454557       M
15   4.075656   2.977061 -3.125274 -2.458071       M
16   0.230055  -1.564758 -0.802519 -0.650583       M
17   4.418011   1.418670 -2.270319 -0.186272       M
18   4.948704  -4.114334 -0.314749 -0.088206       M
19  -1.237063  -0.188215 -0.593283  1.596346       B
20  -1.578161   0.572808 -1.801447  1.125276       B
21  -3.557336   1.662950  0.451187  2.073765       B
22   4.733211   3.304964 -1.466537  2.041150       M
23   4.208524  -5.128367 -0.752402 -0.862710       M
24   4.949632  -1.543752 -1.713194  0.046759       M
25   7.098563   2.018610 -0.029010  2.587951       M
26   3.510263   2.171625 -3.894546 -1.295760       M
27   3.064054  -1.876552  2.581748  0.128484       M
28   4.007264   0.537242 -2.761626 -1.898387       M
29   1.715310  -1.523705  0.146187  1.911386       M
..        ...        ...       ...       ...     ...
539 -1.142832   5.599458  1.301037 -2.188249       B
540 -1.665475   2.389618  1.502249  0.875951       B
541  1.011712   1.092390 -0.632698 -1.758519       B
542 -1.300930  -1.821415  0.373307 -1.848169       B
543 -2.373429  -1.681576  0.384528 -3.016729       B
544 -1.665871  -0.213963 -0.148072 -0.197052       B
545 -1.927678  -1.137740  0.478202 -1.157500       B
546 -4.237217   0.184272 -0.326418  0.588303       B
```

```
547  -2.677871   2.315793 -0.053848  0.340450       B
548  -3.836498   0.496250  0.923240 -0.551872       B
549  -2.551440   0.228330  1.414178 -1.970790       B
550  -4.694923  -0.767478  1.543965 -0.779019       B
551  -2.025037   1.261242  0.504926 -1.135527       B
552  -2.895948  -1.451636  0.780546 -2.970448       B
553  -3.502201   1.800832  2.766457 -0.866307       B
554  -2.153904  -0.830069  0.564797 -3.011756       B
555  -2.055084   1.616459  1.838959 -3.113535       B
556  -3.877290   1.084255  1.859944 -0.433740       B
557  -4.063862   0.122168  3.238773 -3.469183       B
558  -0.098667  -0.213560  0.388929 -1.012710       B
559  -1.089376   1.292848  1.429379 -3.372136       B
560  -0.481771  -0.178020  1.032108 -2.010280       B
561  -4.870310  -2.131106  3.414189 -5.133988       B
562   5.917613   3.482637 -3.262792 -3.917586       M
563   8.741338  -0.573855  0.897090  0.385150       M
564   6.439315  -3.576817  2.459487  1.177314       M
565   3.793382  -3.584048  2.088476 -2.506028       M
566   1.256179  -1.902297  0.562731 -2.089227       M
567  10.374794   1.672010 -1.877029 -2.356031       M
568  -5.475243  -0.670637  1.490443 -2.299157       B

[569 rows x 5 columns]
```

[57]:
```
pca1.explained_variance_ratio_
```

[57]:
```
array([0.44272026, 0.18971182, 0.09393163, 0.06602135])
```

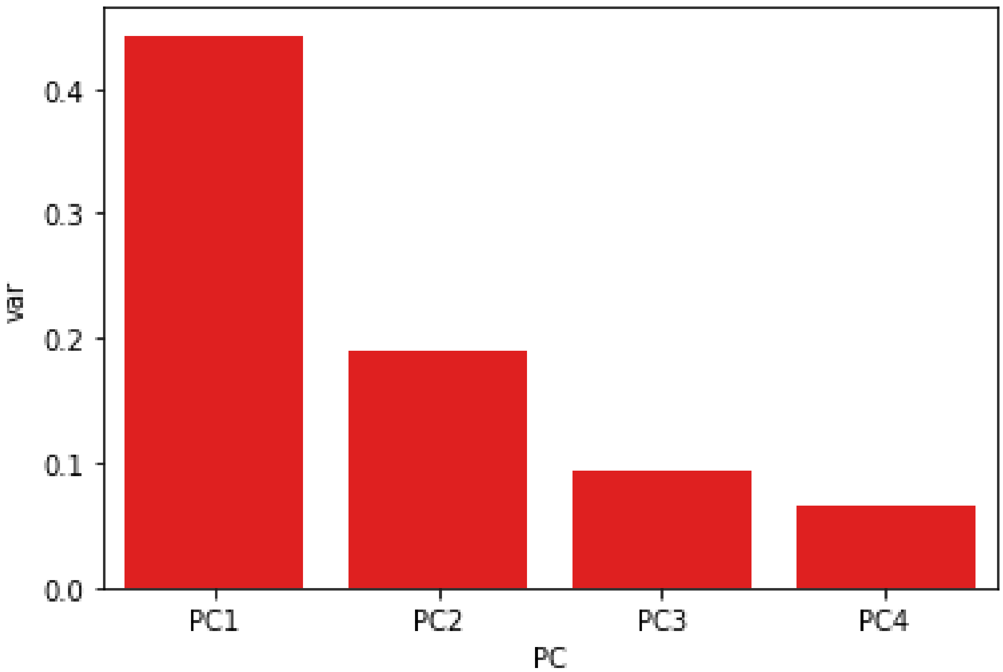
[59]:
```
df = pd.DataFrame({'var':pca1.explained_variance_ratio_, 'PC':['PC1', 'PC2',␣
 ↪'PC3', 'PC4']})
```

[60]:
```
df
```

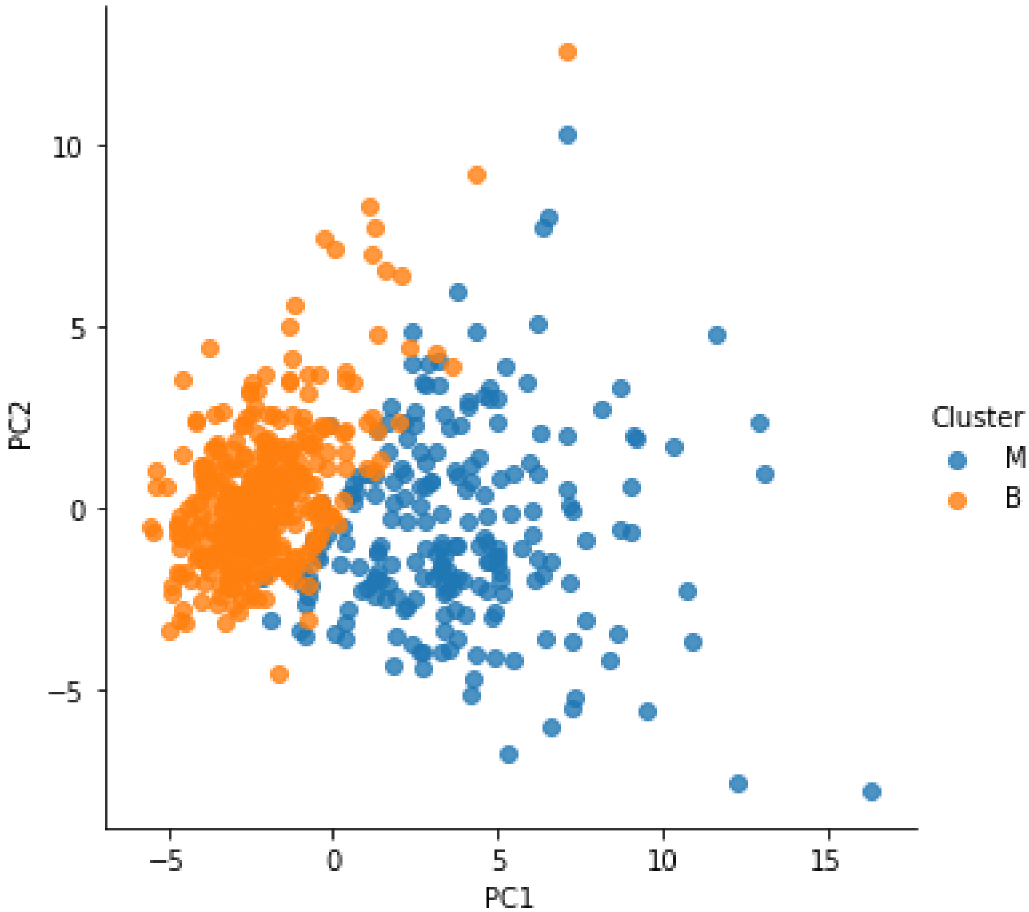
[60]:
```
        var   PC
0  0.442720  PC1
1  0.189712  PC2
2  0.093932  PC3
3  0.066021  PC4
```

[61]:
```
sns.barplot(x='PC', y ='var', data=df, color='red')
```

[61]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a785749400>
```



[62]:
```
p = sns.lmplot(x='PC1', y ='PC2', data=pc_df,hue='Cluster',fit_reg=False,␣
↪legend=True)
```



[ ]:

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

[ ]:

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

[ ]:

Some other tutorials (that are potentially useful):

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