16-PCA_Analysis

October 20, 2024

1 PCA Analysis

Principal Component Analysis (PCA) is a dimensionality reduction technique that is used to reduce the number of variables (features) in a dataset while retaining as much of the important information (variance) as possible. PCA does this by identifying directions, called principal components, along which the variance in the data is maximized.

PCA can be thought of as a way to transform a large set of possibly correlated variables into a smaller set of uncorrelated variables (the principal components). The first principal component captures the most variance in the data, the second captures the second most, and so on.

In simpler terms:

- Dimensionality reduction: PCA reduces the number of features by creating new ones (principal components) that explain most of the variability.
- Data simplification: By transforming data into fewer dimensions, PCA simplifies the data while keeping the most critical information.

1.0.1 When to Use PCA?

PCA is typically used in scenarios where:

- **The data has many features**: When dealing with high-dimensional data, PCA helps reduce complexity by creating a smaller set of meaningful features.
- **There is multicollinearity**: PCA is useful when features in the dataset are highly correlated. The principal components are uncorrelated, which helps in creating a better predictive model.
- **You need visualization of high-dimensional data**: PCA is commonly used for visualizing data in 2D or 3D, even if the original data has more dimensions.
- **To avoid overfitting**: By reducing the number of dimensions, PCA helps prevent overfitting in machine learning models.
- **Preprocessing**: Often used as a data preprocessing step before applying machine learning models like clustering (e.g., K-means), classification (e.g., logistic regression), or regression models.

PCA is used in applications like:

- **Image compression**: Reducing the number of pixels (features) while preserving image quality.

- **Genetics**: Analyzing large datasets of gene expression data.
- **Financial markets**: Reducing the number of stock price indicators while retaining information about market movement.
- **Natural language processing**: Reducing dimensionality of word embeddings or text data before applying machine learning.

1.0.2 How Does PCA Work?

PCA works through a series of mathematical steps that transform the data into a new set of coordinates, called principal components. These components are ordered such that the first one accounts for the most variance, the second for the next largest variance, and so on.

Here's how PCA works step by step:

1. Standardize the Data:

- PCA works best when data is standardized. This is because PCA is affected by the scale of the variables. Standardization transforms the features so that they have a mean of 0 and a standard deviation of 1.

2. Compute the Covariance Matrix:

- The covariance matrix shows how much the features vary from the mean with respect to each other. This helps identify patterns of correlation in the data.
- The covariance matrix is an m $\times$ m matrix (where m is the number of features), with entries representing the covariance between each pair of features.

3. Compute the Eigenvalues and Eigenvectors:

- Eigenvalues represent the amount of variance explained by each principal component, while eigenvectors represent the direction of these components.
- Eigenvalues are used to rank the principal components in order of importance (from the largest to the smallest eigenvalue).

4. Form the Principal Components:

- The principal components are linear combinations of the original features. Each component is a vector in the new feature space. The first component explains the largest amount of variance, the second explains the next largest, and so on.

5. Select the Number of Principal Components (k):

- Usually, you select the number of principal components k that explain a certain threshold of variance (e.g., 90%, 95%). This helps to retain the most important information while reducing the number of dimensions.

6. Project the Data:

- The original data is projected onto the new k-dimensional space (spanned by the selected principal components), creating a reduced-dimensional representation.

Advantages of PCA:

- **Dimensionality reduction**: Significantly reduces the number of features while retaining most of the variance, making the dataset simpler and more manageable.
- **Uncorrelated features**: The principal components are linearly uncorrelated, solving multicollinearity issues in the original features.
- **Data visualization**: PCA helps in visualizing high-dimensional data in two or three dimensions.
- **Speeds up model training**: Reducing the number of features can make machine learning models faster and less prone to overfitting.

Disadvantages of PCA:

- **Loss of interpretability**: The principal components are linear combinations of the original features, which makes them hard to interpret.
- **Sensitive to scaling**: PCA requires features to be on the same scale, so data standardization is often necessary.
- **Linear transformations only**: PCA only captures linear relationships, so it may not work well for datasets with complex, nonlinear structures.
- **Variance-focused**: PCA maximizes variance, but it does not directly optimize for predictive power, so it may not always yield the best features for prediction.

1.0.3 Real-World Applications of PCA:

- **Image Compression**: PCA can reduce the dimensionality of image data by retaining the most important pixel values, enabling image compression without significant loss of quality.
- **Genomics**: PCA is widely used in genomics to reduce the number of gene expression measurements while maintaining the most important patterns of variation across samples.
- **Finance**: In stock market analysis, PCA is used to reduce the number of correlated variables (e.g., stock prices) into principal components that explain overall market movements.
- **Natural Language Processing (NLP)**: PCA is often used to reduce the dimensionality of word vectors (embeddings) in text analysis.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

df = pd.read_csv('2022mlbteams.csv')
df
```

[1]:
```
                        Tm  #Bat  BatAge   R/G    G    PA    AB    R     H  \
0     Arizona Diamondbacks    57    26.5  4.33  162  6027  5351  702  1232
```

1	Atlanta Braves	53	27.5	4.87	162	6082	5509	789	1394
2	Baltimore Orioles	58	27.0	4.16	162	6049	5429	674	1281
3	Boston Red Sox	54	28.8	4.54	162	6144	5539	735	1427
4	Chicago Cubs	64	27.9	4.06	162	6072	5425	657	1293
5	Chicago White Sox	44	29.3	4.23	162	6123	5611	686	1435
6	Cincinnati Reds	66	29.4	4.00	162	5978	5380	648	1264
7	Cleveland Guardians	50	25.9	4.31	162	6163	5558	698	1410
8	Colorado Rockies	43	29.1	4.31	162	6105	5540	698	1408
9	Detroit Tigers	53	27.9	3.44	162	5870	5378	557	1240
10	Houston Astros	45	29.3	4.55	162	6054	5409	737	1341
11	Kansas City Royals	55	27.1	3.95	162	6010	5437	640	1327
12	Los Angeles Angels	66	27.9	3.85	162	5977	5423	623	1265
13	Los Angeles Dodgers	51	29.6	5.23	162	6247	5526	847	1418
14	Miami Marlins	56	28.9	3.62	162	5949	5395	586	1241
15	Milwaukee Brewers	51	29.1	4.48	162	6122	5417	725	1271
16	Minnesota Twins	61	26.9	4.30	162	6113	5476	696	1356
17	New York Mets	61	29.7	4.77	162	6176	5489	772	1422
18	New York Yankees	54	30.2	4.98	162	6172	5422	807	1308
19	Oakland Athletics	64	28.3	3.51	162	5863	5314	568	1147
20	Philadelphia Phillies	56	28.1	4.61	162	6077	5496	747	1392
21	Pittsburgh Pirates	68	26.3	3.65	162	5912	5331	591	1186
22	San Diego Padres	55	28.2	4.35	162	6175	5468	705	1317
23	Seattle Mariners	59	27.5	4.26	162	6117	5375	690	1236
24	San Francisco Giants	66	30.0	4.42	162	6117	5392	716	1261
25	St. Louis Cardinals	51	28.8	4.77	162	6165	5496	772	1386
26	Tampa Bay Rays	61	27.0	4.11	162	6008	5412	666	1294
27	Texas Rangers	55	28.0	4.36	162	6029	5478	707	1308
28	Toronto Blue Jays	51	27.1	4.78	162	6158	5555	775	1464
29	Washington Nationals	55	28.7	3.72	162	5998	5434	603	1351

	2B	…	OPS	OPS+	TB	GDP	HBP	SH	SF	IBB	LOB	Playoffs
0	262	…	0.689	95	2061	97	60	31	50	14	1039	0
1	298	…	0.761	109	2443	103	66	1	36	13	1030	1
2	275	…	0.695	99	2119	95	83	12	43	10	1095	0
3	352	…	0.731	102	2268	131	63	12	50	23	1133	0
4	265	…	0.698	94	2097	130	84	19	36	16	1100	0
5	272	…	0.698	97	2172	127	73	16	35	9	1117	1
6	235	…	0.676	85	2003	127	92	12	33	6	1020	0
7	273	…	0.699	102	2126	119	81	22	52	36	1156	1
8	280	…	0.713	91	2203	139	61	10	40	10	1113	1
9	235	…	0.632	82	1859	108	58	10	44	8	1015	0
10	284	…	0.743	111	2293	118	60	9	42	18	1068	1
11	247	…	0.686	93	2064	101	48	20	44	7	1091	0
12	219	…	0.687	93	2116	95	54	25	25	28	1050	0
13	325	…	0.775	115	2441	85	56	3	53	22	1159	1
14	248	…	0.658	85	1961	120	70	4	36	6	1045	0
15	251	…	0.724	103	2213	117	80	11	37	25	1102	1

```
16  269  ...  0.718  105  2195  133   62  10  46   11  1126          0
17  272  ...  0.744  113  2261  122  112  20  44   25  1158          1
18  225  ...  0.751  112  2311  121   70  14  41   36  1093          1
19  249  ...  0.626   84  1837  109   59  22  33    7   969          0
20  255  ...  0.739  108  2320  116   52   6  44   15  1075          1
21  221  ...  0.655   85  1939   96   54  19  32   14  1016          0
22  275  ...  0.700  102  2087   95   65  17  46   24  1174          1
23  229  ...  0.704  106  2094  120   89   9  45   17  1129          1
24  255  ...  0.705  100  2101  109   95   6  53   14  1115          0
25  290  ...  0.745  112  2309  112   80   5  45   11  1132          1
26  296  ...  0.686   99  2041   93   57   7  31   13  1074          1
27  224  ...  0.696   96  2166   82   47  10  38   12  1007          0
28  307  ...  0.760  117  2395  136   55   8  33   13  1111          1
29  252  ...  0.688   98  2051  141   60  20  37   12  1099          0

[30 rows x 30 columns]
```

[2]:
```
df.drop(columns=['Tm', '#Bat'], axis=1, inplace=True)
```

[3]:
```
X = df.iloc[:, 0:27]
y = df.iloc[:, 27]

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,␣
↪random_state=21)
scaleStandard = StandardScaler()
X_train = scaleStandard.fit_transform(X_train)

df.columns
```

[3]:
```
Index(['BatAge', 'R/G', 'G', 'PA', 'AB', 'R', 'H', '2B', '3B', 'HR', 'RBI',
       'SB', 'CS', 'BB', 'SO', 'BA', 'OBP', 'SLG', 'OPS', 'OPS+', 'TB', 'GDP',
       'HBP', 'SH', 'SF', 'IBB', 'LOB', 'Playoffs'],
      dtype='object')
```

[4]:
```
X_train = pd.DataFrame(X_train, columns=['BatAge', 'R/G', 'G', 'PA', 'AB', 'R',␣
↪'H', '2B', '3B', 'HR', 'RBI',
       'SB', 'CS', 'BB', 'SO', 'BA', 'OBP', 'SLG', 'OPS', 'OPS+', 'TB', 'GDP',
       'HBP', 'SH', 'SF', 'IBB', 'LOB'])


X_train
```

[4]:
```
     BatAge       R/G    G        PA        AB         R         H        2B  \
0 -0.032987 -1.708056  0.0 -2.105939 -2.081256 -1.713743 -2.374828 -0.496051
1 -2.144152  0.099809  0.0  1.070839  1.875833  0.099373  1.267922  0.293245
2 -0.296883  0.212801  0.0 -0.348122  0.578427  0.224896 -0.144856 -1.318234
3 -1.088570 -0.713730  0.0 -0.549318 -0.086494 -0.709556  0.118309 -0.561825
```

4	1.638352	1.613896	0.0	1.166142	-0.329757	1.619600	-0.144856	-1.285346
5	-1.176535	-0.352157	0.0	-0.570496	-0.491933	-0.346933	-0.338766	1.049654
6	-1.616361	0.145006	0.0	-0.369300	-1.481206	0.155161	-1.197514	-0.068515
7	-0.208917	0.777758	0.0	0.160163	0.870343	0.782778	1.018608	-0.298727
8	0.934630	-0.600738	0.0	-0.888174	-1.010896	-0.597979	-0.754289	-0.956473
9	1.110561	2.178854	0.0	1.960337	1.356871	2.177482	1.378728	2.003386
10	0.318874	-1.233491	0.0	-0.676389	-0.135146	-1.225596	0.450727	-0.397389
11	-0.384848	-0.939713	0.0	-0.898763	-0.313540	-0.946656	-0.740438	-1.482670
12	0.494804	-1.459474	0.0	-1.195263	-0.767632	-1.462696	-1.072857	-0.528938
13	0.670735	0.099809	0.0	0.456662	1.583917	0.099373	1.240220	0.523457
14	0.846665	0.642169	0.0	-0.083390	-0.540586	0.643307	0.312219	0.655006
15	1.198526	1.139331	0.0	1.208499	0.756820	1.131454	1.434131	0.260358
16	-1.176535	-0.239165	0.0	-0.136337	-0.216234	-0.235356	-0.518826	0.359020
17	0.406839	0.619570	0.0	0.869643	1.567699	0.615413	1.503384	2.891344
18	0.406839	1.139331	0.0	1.092017	0.870343	1.131454	0.935503	0.852330
19	-1.264500	0.077211	0.0	0.541376	0.545992	0.071479	0.519981	0.161696
20	-0.384848	-0.465149	0.0	0.107216	-0.281105	-0.472456	-0.352617	0.030147
21	1.462422	0.348391	0.0	0.583733	-0.816285	0.350419	-0.795842	-0.298727
22	0.670735	0.483980	0.0	0.636679	-0.410845	0.475943	-0.657334	-0.430276
23	-0.384848	-1.866244	0.0	-2.031814	-1.043331	-1.867160	-1.086707	-0.956473

	3B	HR	…	SLG	OPS	OPS+	TB	GDP	\
0	-1.023632	-1.011987	…	-1.991887	-2.198327	-1.578618	-2.106985	-0.269665	
1	1.101096	-1.305671	…	-0.440312	-0.149673	0.310476	-0.112710	0.352639	
2	-0.359654	0.779487	…	0.062902	-0.233865	-0.319222	0.163314	-1.949889	
3	2.030664	-0.982618	…	-0.566115	-0.514502	-0.634071	-0.540547	-0.767509	
4	-1.953200	2.424118	…	1.362870	1.309641	1.359973	1.163902	0.477100	
5	-0.758041	-0.953250	…	-0.691919	-0.514502	-0.004373	-0.699261	-1.265353	
6	0.171527	0.045276	…	-0.356443	-0.430311	-0.424172	-0.561249	-1.016431	
7	0.835505	0.985066	…	1.195132	0.972877	0.940174	1.226008	0.165948	
8	-0.625245	-0.453987	…	-0.901591	-0.795139	-1.473669	-0.961484	0.850483	
9	1.101096	1.190645	…	2.033822	1.983171	1.674822	2.060981	-1.763197	
10	-0.359654	-1.041355	…	-0.691919	-0.458375	-0.109323	-0.630255	1.721710	
11	1.101096	0.544539	…	-0.146771	-0.486438	-0.634071	-0.181716	-1.140892	
12	-0.359654	-0.806408	…	-1.279001	-1.300287	-1.473669	-1.251310	0.414870	
13	1.499482	-0.659566	…	0.188705	0.243219	-0.843970	0.418637	1.597249	
14	-1.289223	1.249381	…	1.279001	1.085132	1.255023	1.039691	0.290409	
15	0.569914	-0.013461	…	0.775788	1.113195	1.464923	0.818872	0.539331	
16	0.304323	-0.013461	…	-0.146771	-0.261928	-0.004373	-0.161014	-1.140892	
17	-1.422018	-0.483355	…	0.649984	0.748367	0.310476	0.867176	1.099405	
18	-0.226859	0.750118	…	1.111263	1.141259	1.359973	1.150101	-0.082974	
19	-0.625245	0.192118	…	0.314508	0.383538	0.625325	0.363432	1.223866	
20	1.101096	-0.365882	…	-0.272574	-0.177737	-0.529121	-0.312827	1.037175	
21	-0.625245	0.338960	…	-0.146771	0.018709	0.100577	-0.285225	-0.269665	
22	-0.758041	1.396224	…	0.649984	0.551920	0.415426	0.487643	0.228178	
23	0.569914	-1.804934	…	-1.991887	-2.029944	-1.788518	-1.955172	-0.331896	

```
         HBP        SH        SF       IBB       LOB
0  -0.590017  1.201209 -1.197071 -1.046090 -2.339528
1   0.776338  1.201209  1.465682  2.309111  1.440098
2  -1.335301 -0.462003 -0.496347 -0.467607 -1.571476
3  -1.273194  0.924007  0.344523 -1.046090  0.126324
4   0.093161  0.092401 -0.075912  2.309111  0.166748
5  -0.714231 -0.877807 -1.477361 -0.351910 -0.217278
6  -0.527910  2.448618  1.185393 -0.236214 -0.924695
7  -1.024766 -1.016408  0.344523 -0.120517 -0.197066
8   1.459515 -0.184801 -1.197071 -1.161786 -1.308721
9  -0.776338 -1.432211  1.605827  0.689359  1.500734
10 -0.527910  0.924007 -0.636492 -0.467607  0.288020
11 -0.900552  1.617012 -2.318231  1.383538 -0.702364
12  0.093161 -1.293610 -0.776637 -1.161786 -0.803423
13 -0.465803 -0.462003 -0.216057 -0.699000  0.570986
14 -0.527910 -0.600604  0.064233  0.226572 -0.338549
15  2.701655  0.924007  0.344523  1.036449  1.480522
16  0.900552 -0.184801  0.204378 -0.699000  0.207172
17 -0.341589 -0.184801  1.185393  0.805055  0.975224
18  0.714231 -1.155009  0.484668 -0.583304  0.955012
19 -0.403696 -0.462003  0.624813 -0.583304  0.833741
20  0.962659  0.785406 -0.776637 -0.004821  0.308232
21  1.645836 -1.016408  1.605827 -0.236214  0.611410
22  0.714231 -0.323402 -0.636492  1.036449  0.348655
23 -0.652124 -0.462003  0.344523 -0.930393 -1.409780

[24 rows x 27 columns]
```

[5]:
```python
X_train.describe().round(3)
```

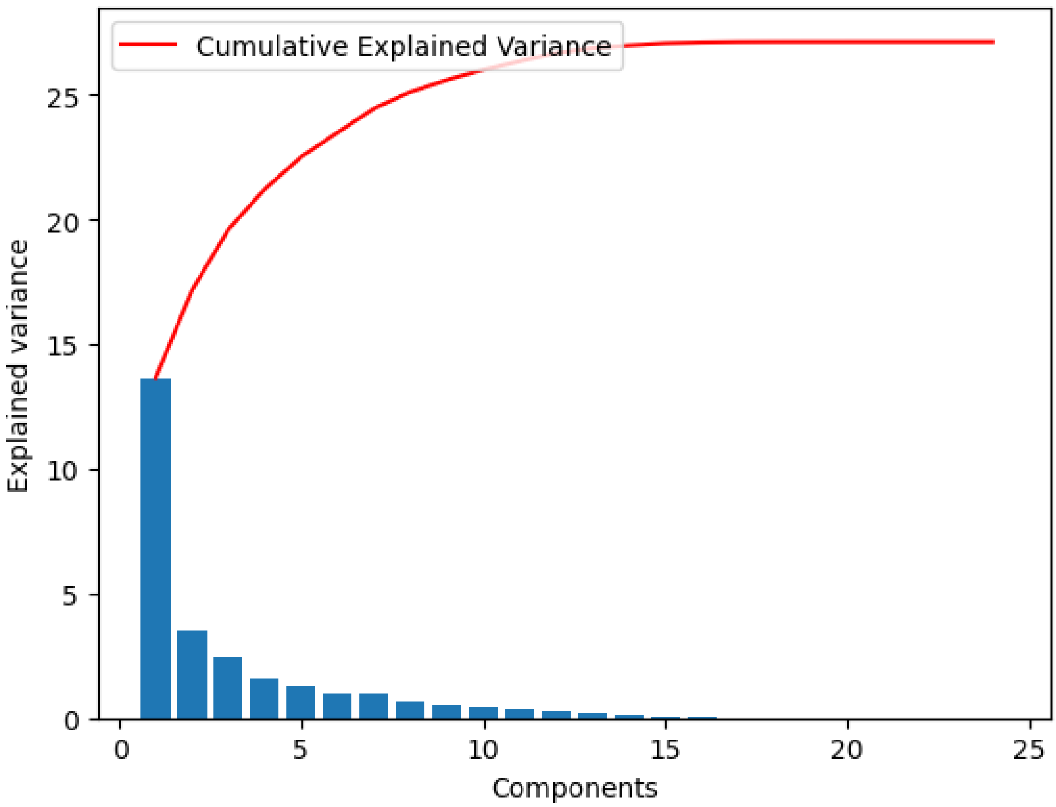
[5]:
```
       BatAge     R/G     G      PA      AB       R       H      2B      3B  \
count  24.000  24.000  24.0  24.000  24.000  24.000  24.000  24.000  24.000
mean    0.000   0.000   0.0  -0.000   0.000   0.000   0.000   0.000   0.000
std     1.022   1.022   0.0   1.022   1.022   1.022   1.022   1.022   1.022
min    -2.144  -1.866   0.0  -2.106  -2.081  -1.867  -2.375  -1.483  -1.953
25%    -0.561  -0.629   0.0  -0.597  -0.597  -0.626  -0.744  -0.537  -0.658
50%     0.143   0.100   0.0   0.012  -0.249   0.099  -0.145  -0.184  -0.293
75%     0.715   0.625   0.0   0.695   0.785   0.622   0.956   0.400   0.902
max     1.638   2.179   0.0   1.960   1.876   2.177   1.503   2.891   2.031

           HR  …     SLG     OPS    OPS+      TB     GDP     HBP      SH  \
count  24.000  …  24.000  24.000  24.000  24.000  24.000  24.000  24.000
mean   -0.000  …   0.000  -0.000  -0.000  -0.000   0.000   0.000  -0.000
std     1.022  …   1.022   1.022   1.022   1.022   1.022   1.022   1.022
min    -1.805  …  -1.992  -2.198  -1.789  -2.107  -1.950  -1.335  -1.432
25%    -0.843  …  -0.598  -0.493  -0.634  -0.579  -0.830  -0.668  -0.670
50%    -0.013  …  -0.147  -0.164  -0.004  -0.137   0.197  -0.435  -0.254
```

```
75%     0.757  …   0.681   0.804   0.704   0.831   0.617   0.730   0.924
max     2.424  …   2.034   1.983   1.675   2.061   1.722   2.702   2.449

           SF     IBB     LOB
count  24.000  24.000  24.000
mean    0.000  -0.000   0.000
std     1.022   1.022   1.022
min    -2.318  -1.162  -2.340
25%    -0.672  -0.699  -0.728
50%     0.134  -0.294   0.187
75%     0.520   0.718   0.667
max     1.606   2.309   1.501

[8 rows x 27 columns]
```

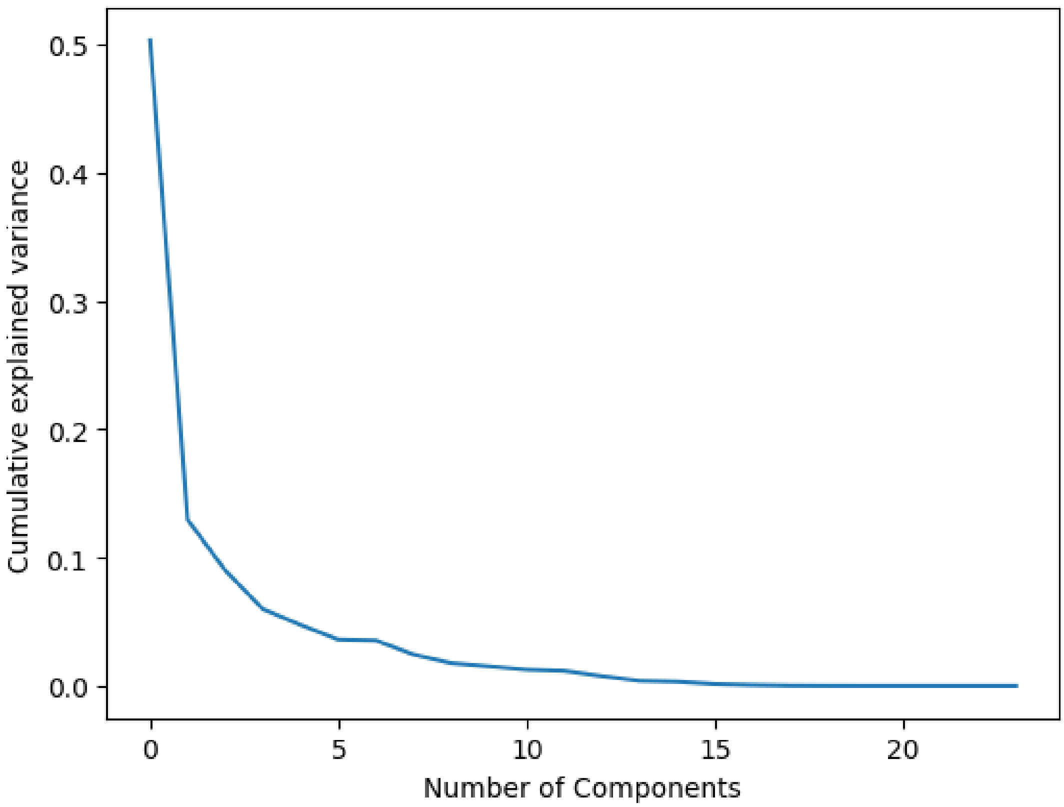
[6]:
```python
pca1 = PCA()
X_pca1 = pca1.fit_transform(X_train)
pca1.explained_variance_ratio_
```

[6]:
```
array([5.03143586e-01, 1.29565631e-01, 8.99453783e-02, 5.97627178e-02,
       4.75204409e-02, 3.60767505e-02, 3.53674555e-02, 2.44602746e-02,
       1.77297976e-02, 1.52080776e-02, 1.26222548e-02, 1.16581886e-02,
       7.54086691e-03, 3.93586244e-03, 3.24827865e-03, 1.37155962e-03,
       6.27774032e-04, 1.54100200e-04, 4.62620159e-05, 9.33531667e-06,
       4.48882814e-06, 6.43748708e-07, 2.74352526e-07, 3.89906609e-34])
```

[7]:
```python
plt.bar(range(1, len(pca1.explained_variance_) + 1), pca1.explained_variance_)
plt.ylabel("Explained variance")
plt.xlabel('Components')
plt.plot(range(1, len(pca1.explained_variance_) + 1), np.cumsum(pca1.
 ↪explained_variance_), c='red', label="Cumulative Explained Variance")
plt.legend(loc="upper left")
plt.show()
```




```
[8]: plt.plot(pca1.explained_variance_ratio_)
     plt.xlabel('Number of Components')
     plt.ylabel('Cumulative explained variance')
     plt.show()
```



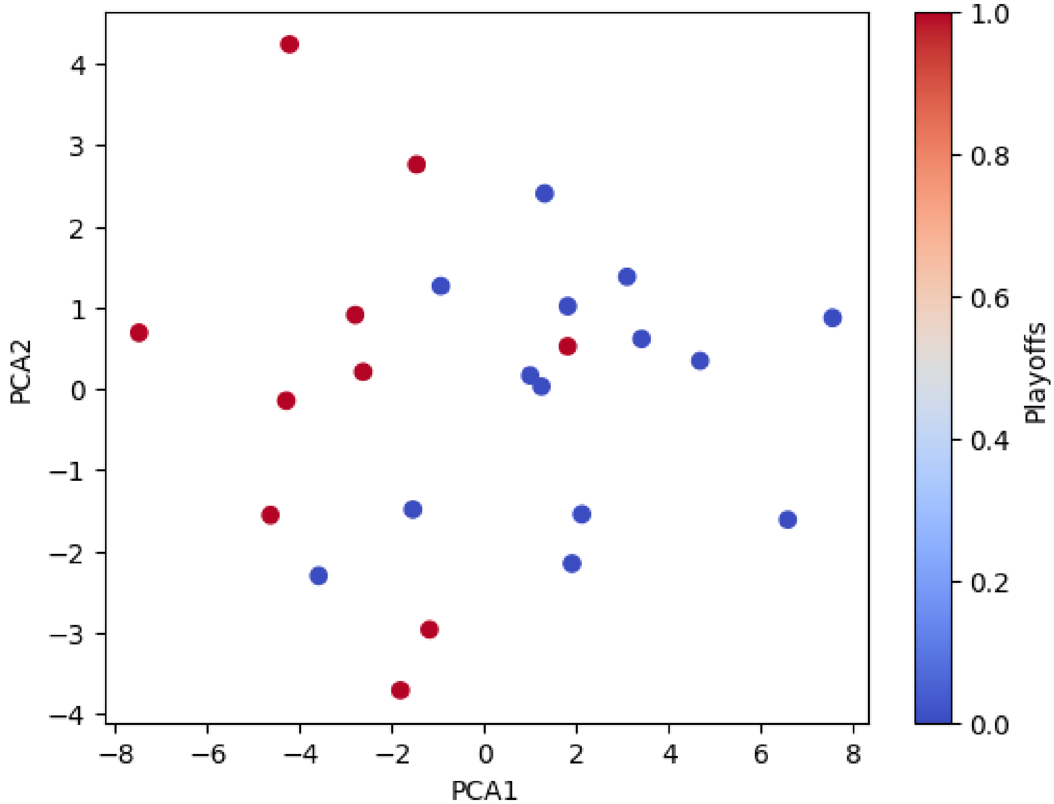
[9]:
```
pca2 = PCA(0.95)
X_pca2 = pca2.fit_transform(X_train)
X_pca2.shape
```

[9]: (24, 10)

[10]:
```
pca2.explained_variance_ratio_
```

[10]:
```
array([0.50314359, 0.12956563, 0.08994538, 0.05976272, 0.04752044,
       0.03607675, 0.03536746, 0.02446027, 0.0177298 , 0.01520808])
```

[11]:
```
pca2c = PCA(n_components=2)
X_pca2c = pca2c.fit_transform(X_train)

colormap = plt.get_cmap('coolwarm')
plt.figure()
scatter = plt.scatter(X_pca2c[:, 0], X_pca2c[:, 1], c=y_train, cmap=colormap)
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.colorbar(scatter, label="Playoffs")
plt.show()
```



PCA1
PCA2
Playoffs
−8
−6
−4
−2
0
2
4
6
8
4
3
2
1
0
−1
−2
−3
−4
1.0
0.8
0.6
0.4
0.2
0.0