

# CA-PCA-FIFA-instructor

December 11, 2019

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(font_scale=2)
sns.set_style("whitegrid")
```

## 1 Principal Components of FIFA Dataset

Like the last class activity, we will be using the data analysis library pandas. This time we will be looking at the [FIFA 2018 Dataset](#). While this is a video game, the developers strive to make their game as accurate as possible, so this data reflects the skills of the real-life players.

Let's load the data frame using pandas.

```
In [2]: df = pd.read_csv("FIFA_2018.csv", encoding = "ISO-8859-1", index_col = 0, low_memory = Fal
```

We can take a brief look at the data by calling `df.head()`. The first 34 columns are attributes that describe the behavior (e.g. aggression) or the skills (e.g. ball control), of each player. The final columns show the player's position, name, nationality, and the club they play for.

The four positions are forward (FWD), midfielder (MID), defender (DEF), and goalkeeper (GK).

```
In [3]: df.head()
```

```
Out[3]:
```

	Acceleration	Aggression	Agility	Balance	Ball control	Composure	\
0	89	63	89	63	93	95	
1	92	48	90	95	95	96	
2	94	56	96	82	95	92	
3	88	78	86	60	91	83	
4	58	29	52	35	48	70	

	Crossing	Curve	Dribbling	Finishing	...	Sprint speed	Stamina	\
0	85	81	91	94	...	91	92	
1	77	89	97	95	...	87	73	
2	75	81	96	89	...	90	78	
3	77	86	86	94	...	77	89	
4	15	14	30	13	...	61	44	

	Standing tackle	Strength	Vision	Volleys	Position	Name \
0	31	80	85	88	FWD	Cristiano Ronaldo
1	28	59	90	85	FWD	L. Messi
2	24	53	80	83	FWD	Neymar
3	45	80	84	88	FWD	L. Suarez
4	10	83	70	11	GK	M. Neuer

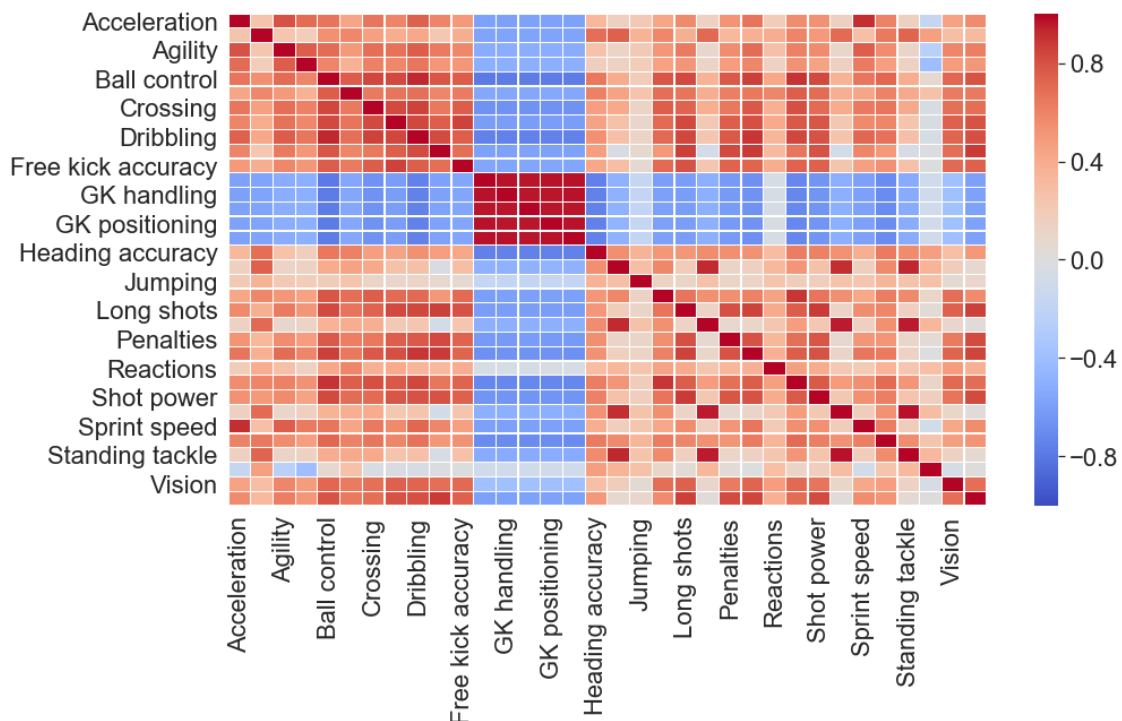
	Nationality	Club
0	Portugal	Real Madrid CF
1	Argentina	FC Barcelona
2	Brazil	Paris Saint-Germain
3	Uruguay	FC Barcelona
4	Germany	FC Bayern Munich

[5 rows x 38 columns]

A higher number signifies that an attribute is more prevalent for that player. Looking at the above rankings, Player 0 (Christiano Ronaldo) has very good ball control and composure, but is not overly aggressive.

**Correlation Matrix** We can compute the correlation matrix for these variables across all players using a "heatmap". Calling `df.corr()` provides this correlation matrix, and `seaborn.heatmap` will do the plotting.

```
In [4]: plt.figure(figsize=(15,8))
sns.heatmap(df.corr(),vmin=-1.0,vmax=1.0, linewidth=0.25, cmap='coolwarm');
```



This heatmap is dark red whenever two variables are positively correlated, and dark blue when they are negatively correlated. For example, "Sprint Speed" and "Acceleration" are positively correlated. "Balance" and "Strength" are negatively correlated however.

Notice across the diagonal, all rectangles are dark red. This is to be expected, as any variable is perfectly correlated with itself.

Also notice that all Goal-Keeping skills are positively correlated with each other, but are negatively correlated with nearly all the other variables. Maybe we can compress these into a single component/feature through principal component analysis.

## 2 Principal Component analysis

Recall that Principal Component Analysis (PCA) projects high-dimensional data into a low-dimensional representation by finding directions of maximal variance.

Let's first create a new dataframe that includes only the attributes of each player (and not the last four columns of df). Store this new dataframe as a variable X.

```
In [5]: #clear
        X = df.iloc[:, :-4].copy()
```

We can get all the attribute names and store them as labels by using `.columns.values`

```
In [6]: #clear
        labels = X.columns.values
        #labels
```

To perform PCA, we first shift the data so that each attribute has zero mean, then compute the Singular Value Decomposition (SVD) of the resulting data matrix.

Create the data frame A where each attribute has zero mean. Should we ensure each row has zero mean, or each column?

```
In [7]: #clear
        X = df.iloc[:, :-4].copy()
```

```
In [8]: # clear
        A = X - X.mean()
        A.mean()
```

```
Out[8]: Acceleration      2.167865e-13
         Aggression       -1.467107e-12
         Agility          -4.619496e-14
         Balance          -1.782434e-13
         Ball control      1.092776e-12
         Composure        1.111679e-12
         Crossing         1.600354e-12
         Curve            1.192740e-12
         Dribbling        1.508677e-12
```

```

Finishing          1.276988e-12
Free kick accuracy 1.505077e-12
GK diving          4.589444e-14
GK handling        5.571325e-14
GK kicking         9.023588e-14
GK positioning    -7.729508e-14
GK reflexes       -2.862759e-14
Heading accuracy  -1.318541e-12
Interceptions     -1.023297e-12
Jumping           7.924063e-13
Long passing       1.456835e-12
Long shots        -1.819519e-12
Marking           7.227232e-13
Penalties         1.203584e-13
Positioning       6.242075e-13
Reactions         3.090709e-13
Short passing     7.371135e-13
Shot power        -9.480112e-13
Sliding tackle    3.611079e-13
Sprint speed     -8.388917e-13
Stamina          -3.460696e-13
Standing tackle   8.578911e-13
Strength          2.334315e-13
Vision           1.559979e-12
Volleys          8.684768e-13
dtype: float64

```

Now compute the SVD of the resulting matrix. Make sure you compute the *reduced* SVD, not the full one, since the full SVD will take a long time to finish.

Once you have computed the SVD, you can plot the fraction of explained variance for each singular value

$$\frac{\sigma_i^2}{\sum_{k=1}^r \sigma_k^2} \quad i = 1, 2, \dots, r \quad (1)$$

as well as the cumulative explained variance

$$\frac{\sum_{k=1}^i \sigma_k^2}{\sum_{k=1}^r \sigma_k^2} \quad i = 1, 2, \dots, r \quad (2)$$

You can create a bar plot of the fraction of explained variance for each singular value using `plt.bar`, and a standard line plot for the cumulative explained variance.

```

In [9]: # clear
U, S, Vt = np.linalg.svd(A, full_matrices = False)
V = Vt.T

variance = S**2
sum_var = sum(variance)

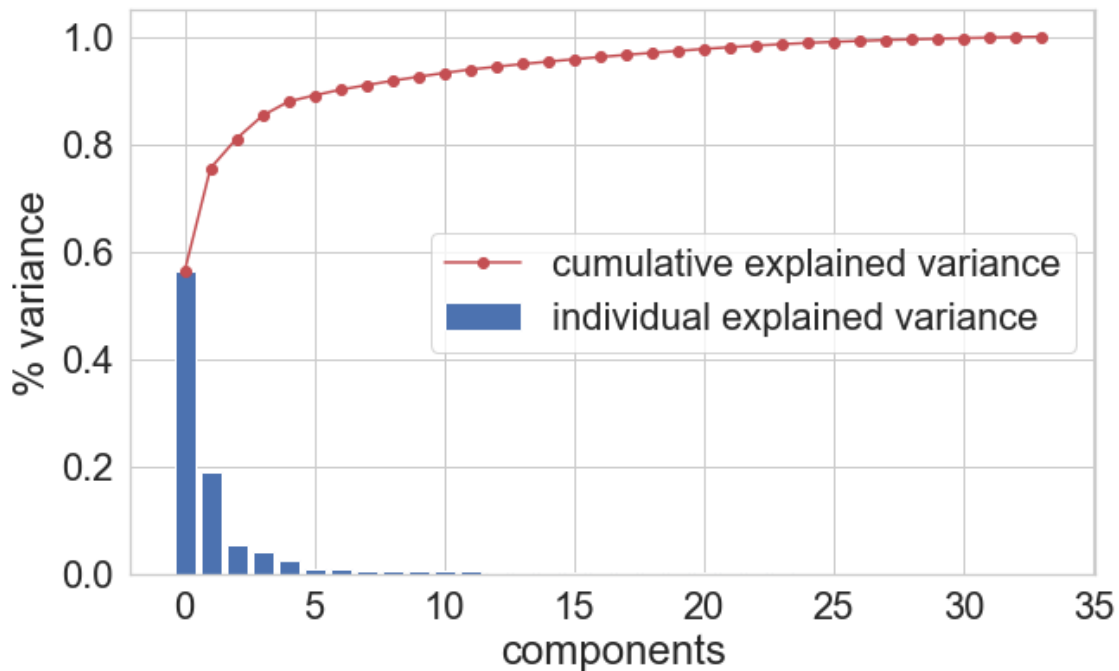
```

```

var_exp = [v/sum_var for v in variance]
cum_var = np.cumsum(var_exp)

plt.figure(figsize=(10,6))
plt.bar(range(34),var_exp,label='individual explained variance')
plt.plot(range(34),cum_var,'ro-', label='cumulative explained variance')
plt.legend(loc=5)
plt.xlabel("components")
plt.ylabel("% variance")
plt.show()

```



You should see from the graph that the first principal component is responsible for nearly 60% of the variance, and the first two principal components have well over 70%.

Recall from the SVD that  $\mathbf{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i$ . Writing the columns of  $\mathbf{A}$  as  $\mathbf{a}_k$ , this means that:

$$\mathbf{v}_i^{(1)} \begin{bmatrix} \vdots \\ \mathbf{a}_1 \\ \vdots \end{bmatrix} + \mathbf{v}_i^{(2)} \begin{bmatrix} \vdots \\ \mathbf{a}_2 \\ \vdots \end{bmatrix} + \cdots + \mathbf{v}_i^{(n)} \begin{bmatrix} \vdots \\ \mathbf{a}_n \\ \vdots \end{bmatrix} = \sigma_i\mathbf{u}_i \quad (3)$$

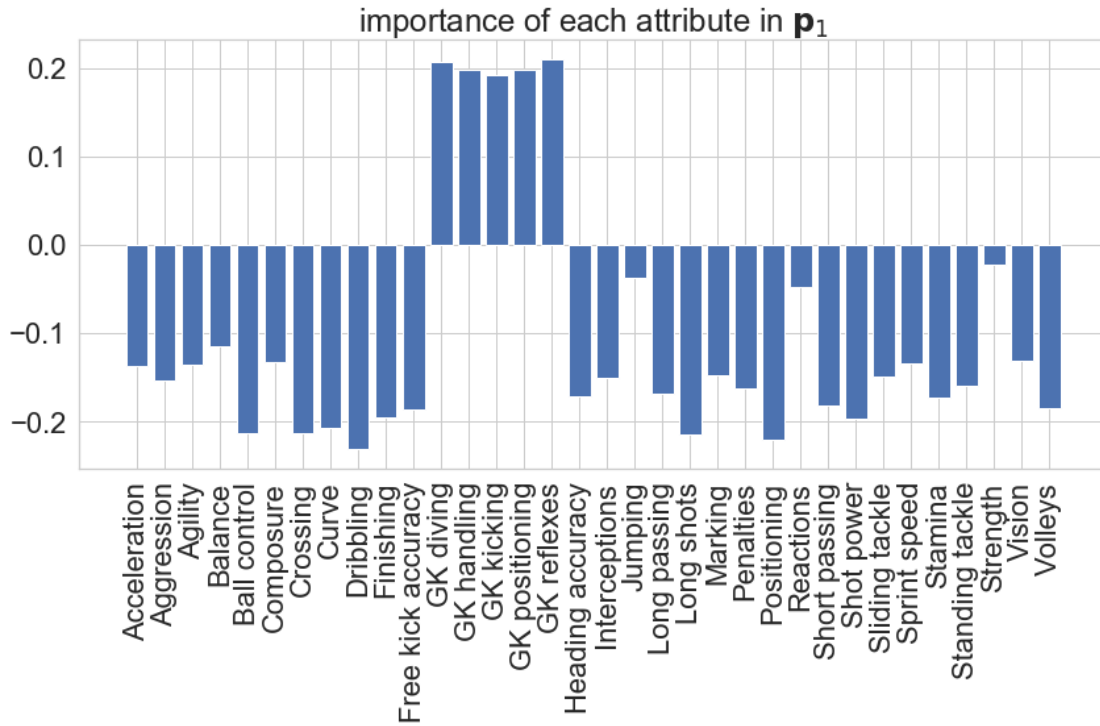
where  $\mathbf{v}_i^{(j)}$  is the  $j$ -th component of  $\mathbf{v}_i$ . Thus if we define the principal components as

$$\mathbf{p}_i = \sigma_i\mathbf{u}_i,$$

the  $i$ -th column of  $\mathbf{V}$  describes the projection of each attribute onto that principal direction.

We can visualize the weight of each attribute to a given principal component by plotting the entries of the corresponding column of  $\mathbf{V}$ . For example, the plot below illustrates the "importance" of each attribute to the first principal component ( $\mathbf{p}_1$ )

```
In [10]: plt.figure(figsize=(14,6))
plt.bar(labels,V[:,0])
plt.xticks(rotation=90);
plt.title('importance of each attribute in  $\mathbf{p}_1$ ');
```



Now, let's add two new columns to the original dataframe `df`, with headers `pc1` and `pc2`. Use the expression above to evaluate the first two principal components  $\mathbf{p}_1$  and  $\mathbf{p}_2$

```
In [11]: # clear
df['pc1'] = U[:,0]*S[0]
df['pc2'] = U[:,1]*S[1]
df.head()
```

```
Out[11]:
```

	Acceleration	Aggression	Agility	Balance	Ball control	Composure	\	
0	89	63	89	63	93	95		
1	92	48	90	95	95	96		
2	94	56	96	82	95	92		
3	88	78	86	60	91	83		
4	58	29	52	35	48	70		
	Crossing	Curve	Dribbling	Finishing	...	Standing tackle	Strength	\
0	85	81	91	94	...	31	80	
1	77	89	97	95	...	28	59	
2	75	81	96	89	...	24	53	
3	77	86	86	94	...	45	80	

```
4          15      14          30          13 ...          10          83
```

```
      Vision  Volleys  Position      Name  Nationality \
0         85      88      FWD  Cristiano Ronaldo  Portugal
1         90      85      FWD      L. Messi  Argentina
2         80      83      FWD      Neymar  Brazil
3         84      88      FWD      L. Suarez  Uruguay
4         70      11      GK      M. Neuer  Germany
```

```
      Club      pc1      pc2
0  Real Madrid CF -123.550481  90.062030
1    FC Barcelona -118.138937 108.176601
2 Paris Saint-Germain -107.616170  92.809286
3    FC Barcelona  -99.767468  71.759809
4  FC Bayern Munich  167.616507  28.394824
```

```
[5 rows x 40 columns]
```

Let's plot the data with these first two principal components.

```
In [12]: g = sns.lmplot(x = "pc1", y = "pc2", data = df, hue = "Position", fit_reg=False, height=10,
                        scatter_kws={'s':14, 'alpha':0.5})
ax = g.axes[0,0]
ax.axvline(x=0,color='k', ls = '--')
ax.axhline(y=0,color='k', ls = '--')
plt.show()
```



It looks like the first principal axis determines whether a player is a goalkeeper or not. We should double-check to make sure.

What are the attributes of  $A$  that are most positively correlated with the first principal component?

We can answer that by looking at the plot of coefficients above. Or we can do this in a systematic way, by sorting the entries of the column of  $V$  and finding the ones with highest positive values.

Find the first 5 attributes, and print their corresponding weights.

```
In [13]: # clear
         ind = np.argsort(V[:,0])
         print(ind)
         print(labels[ind[-5:]])
         print(V[ind[-5:],0])
         print(labels[ind[:5]])

[ 8 23 20  6  4  7 26  9 10 33 25 29 16 19 22 30  1 17 27 21  0  2 28  5
 32  3 24 18 31 13 12 14 11 15]
['GK kicking' 'GK handling' 'GK positioning' 'GK diving' 'GK reflexes']
[0.19212694 0.19755409 0.19848829 0.20711799 0.21038404]
['Dribbling' 'Positioning' 'Long shots' 'Crossing' 'Ball control']
```

You can see that all the goalkeeper attributes are positively correlated with the first principal component. However, all other attributes, beginning with "Strength" are negatively correlated. Try plotting the projection of "GK reflexes" onto the first two principal components

```
In [14]: g = sns.lmplot(x = "pc1", y = "pc2", data = df, hue = "Position", fit_reg=False, height=
                markers=["o", "x", "^", "s"],palette=dict(FWD="g", GK="orange", MID="r", DEF=
ax = g.axes[0,0]
ax.axvline(x=0,color='k', ls = '--')
ax.axhline(y=0,color='k', ls = '--')

scale = 400 # this will scale the size of the arrow plot
J = 3      # looking at the position "GK reflexes", corresponding to column 31
x = V[J,0] # projection of "GK reflexes" onto first principal component
y = V[J,1] # projection of "GK reflexes" onto second principal component

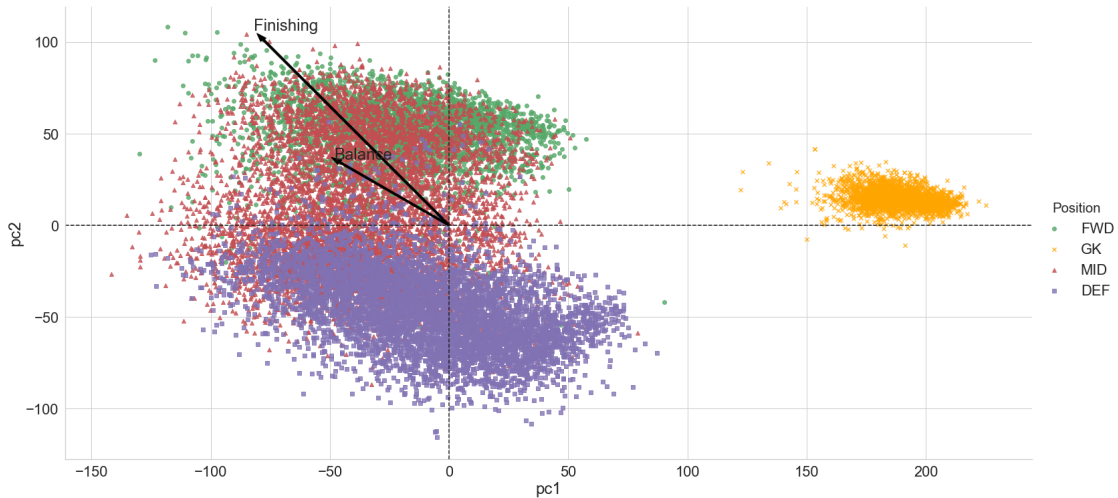
# make an arrow from the origin to a point at (x,y)
ax.arrow(0,0,scale*x,scale*y,color='black',width=1)
ax.text(x*scale*1.05,y*scale*1.05,labels[J],fontsize=24)

J = 9      # looking at the position "GK reflexes", corresponding to column 31
x = V[J,0] # projection of "GK reflexes" onto first principal component
y = V[J,1] # projection of "GK reflexes" onto second principal component

# make an arrow from the origin to a point at (x,y)
ax.arrow(0,0,scale*x,scale*y,color='black',width=1)
ax.text(x*scale*1.05,y*scale*1.05,labels[J],fontsize=24)
```

```
Out[14]: Text(-81.7848092914674, 106.27381571208693, 'Finishing')
```





If you plot any other of the GK attributes, they will essentially overlap with GK reflexes. Check that, by changing the variable J above to take the values (11,12,13,14).

Make the same plot as above, but now take a look at other attributes. In the same figure, plot the projections for the attributes in columns [1,8,9,16,28,31].

Do you think the results make sense?

In [15]: `#clear`

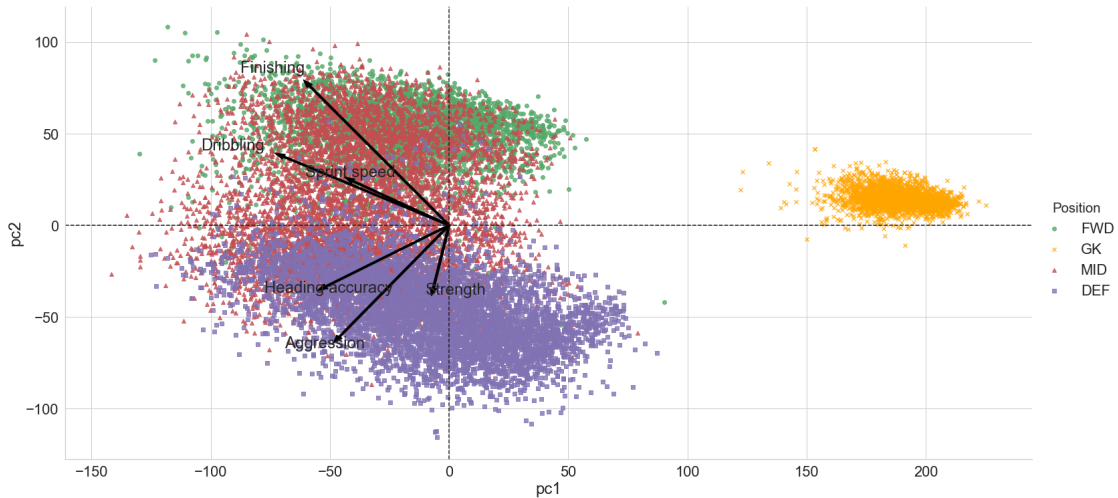
```
g = sns.lmplot(x = "pc1", y = "pc2", data = df, hue = "Position", fit_reg=False, height=10,
              markers=["o", "x", "^", "s"], palette=dict(FWD="g", GK="orange", MID="r", DEF="b"))
ax = g.axes[0,0]
ax.axvline(x=0,color='k', ls = '--')
ax.axhline(y=0,color='k', ls = '--')

scale = 300 # this will scale the size of the arrow plot

for J in [1,8,9,16,28,31]:

    x = V[J,0]
    y = V[J,1]

    # make an arrow from the origin to a point at (x,y)
    ax.arrow(0,0,scale*x,scale*y,color='black',width=1)
    ax.text(x*scale*1.5,y*scale*1.1,labels[J],fontsize=24)
```



## 2.1 Remove data and re-do PCA

The first principal component seems to mainly dictate whether a player is a goal-keeper or not. To find out more about the data, we can drop all goal-keepers and repeat PCA.

We first create a new data-frame with all goal-keepers removed:

```
In [16]: df2 = df[df["Position"] != "GK"].copy()
```

Now we remove all the columns associated with the attributes that are mostly associated with goal-keepers. We also remove the columns with pc1 and pc2

```
In [17]: df2 = df2.drop(['GK diving',
                        'GK handling',
                        'GK kicking',
                        'GK positioning',
                        'GK reflexes', 'pc1', 'pc2'], 1)
```

Repeat all the steps from the previous analysis: shift to zero-mean, obtain svd, plot explained variances.

```
In [18]: # clear

Y = df2.iloc[:, :-4].copy()

B = Y - Y.mean()
u, s, vt = np.linalg.svd(B, full_matrices = False)
v = vt.T

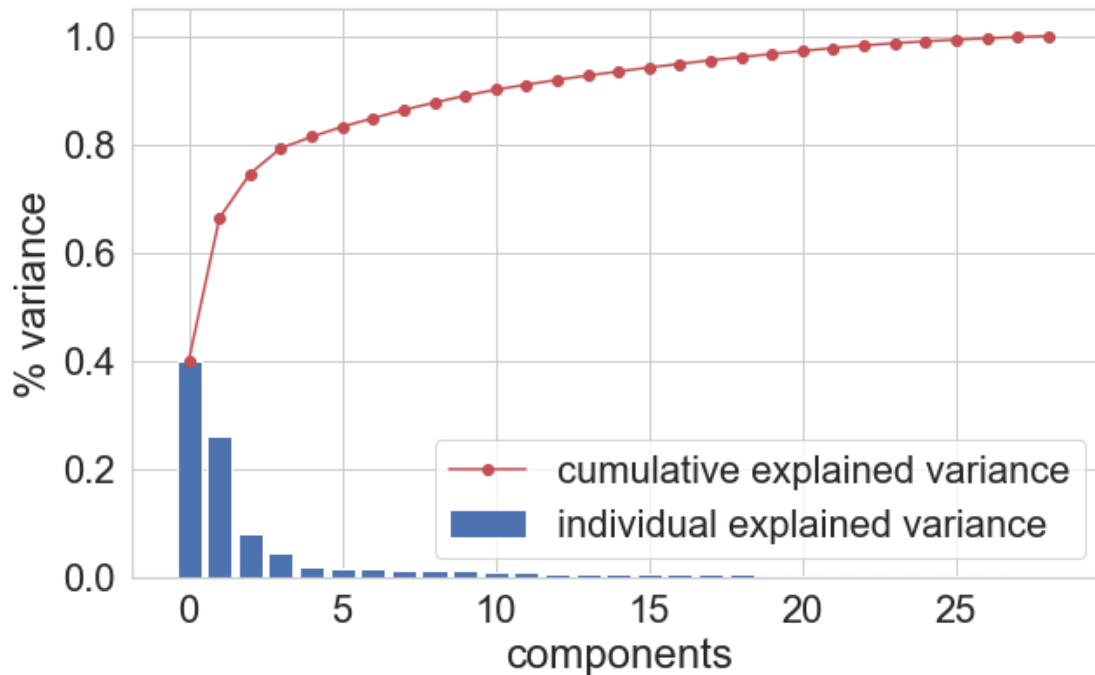
variance = s**2
sum_var = sum(variance)
var_exp = [vv/sum_var for vv in variance]
```

```

cum_var = np.cumsum(var_exp)

plt.figure(figsize=(10,6))
plt.bar(range(29), var_exp, label='individual explained variance')
plt.plot(range(29), cum_var, 'ro-', label='cumulative explained variance')
plt.legend(loc=0)
plt.xlabel("components")
plt.ylabel("% variance")
plt.show()

```



Add the first two components to the data frame and plot them in a scatter plot.

```

In [19]: # clear
df2['pc1'] = u[:,0]*s[0]
df2['pc2'] = u[:,1]*s[1]

```

```

In [20]: #clear

```

```

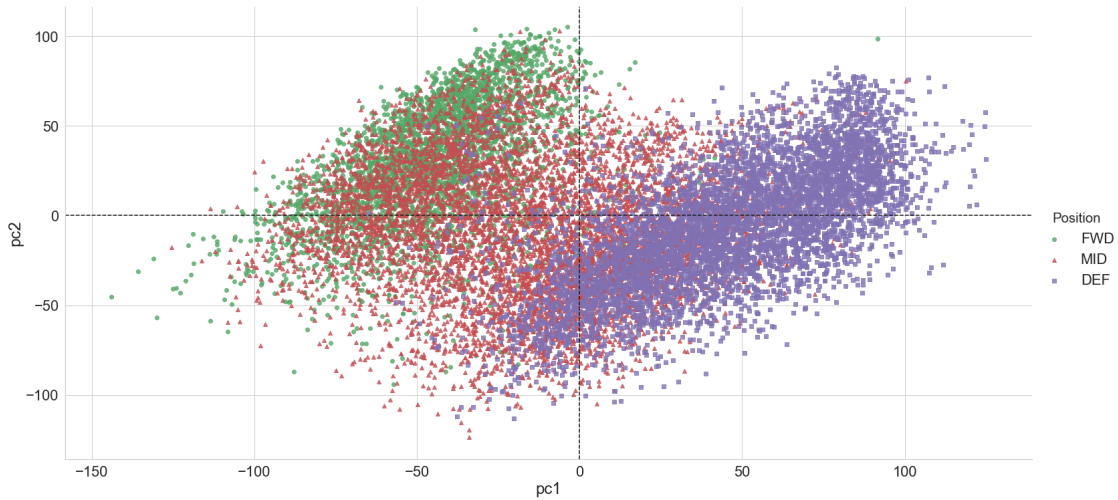
g = sns.lmplot(x = "pc1", y = "pc2", data = df2, hue = "Position", fit_reg=False, height=10,
               markers=["o", "^", "s"], palette=dict(FWD="g", MID="r", DEF="m"))
ax = g.axes[0,0]
ax.axvline(x=0,color='k', ls = '--')
ax.axhline(y=0,color='k', ls = '--')

```

```

Out[20]: <matplotlib.lines.Line2D at 0x10ed807f0>

```

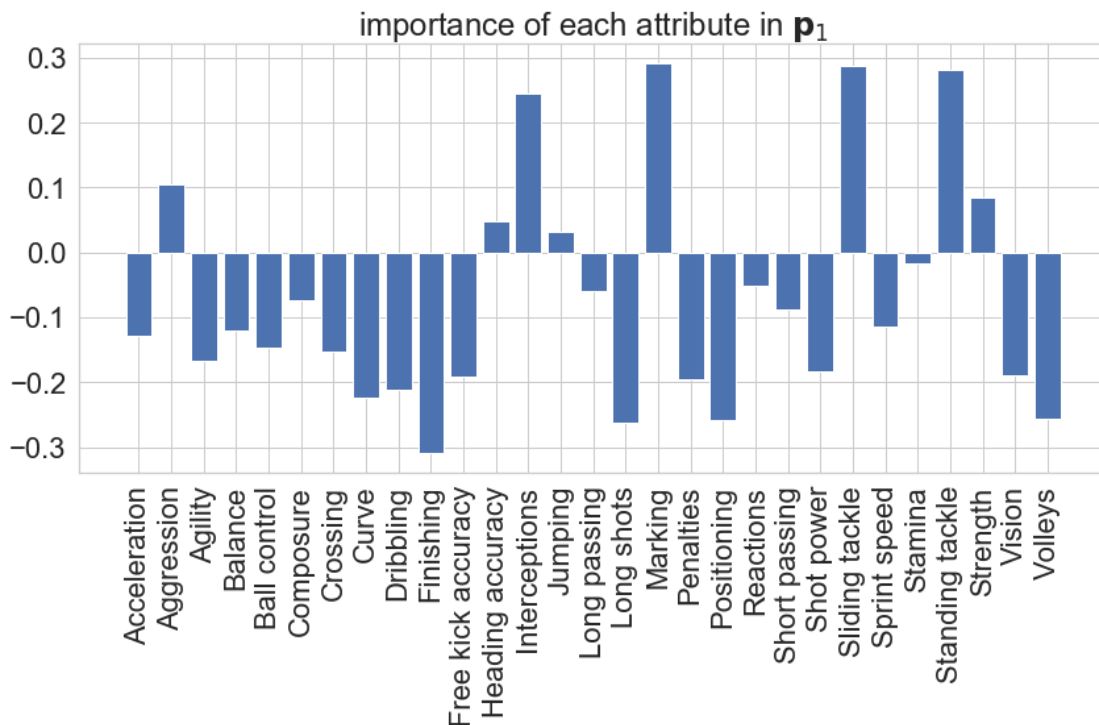


Plot the weights of each attribute corresponding to the principal component 1:

```
In [21]: #clear
```

```
labels_new = Y.columns.values
```

```
plt.figure(figsize=(14,6))
plt.bar(labels_new,v[:,0])
plt.xticks(rotation=90);
plt.title('importance of each attribute in  $\mathbf{p}_1$ ');
```



In the same figure, plot the projections for the attributes in columns [4,9,11,12,14,3,12] onto principal components 1 and 2.

In [25]: `#clear`

```
g = sns.lmplot(x = "pc1", y = "pc2", data = df2, hue = "Position", fit_reg=False, height=10,
               markers=["o", "^", "s"], palette=dict(FWD="g", MID="r", DEF="m"))
ax = g.axes[0,0]
ax.axvline(x=0,color='k', ls = '--')
ax.axhline(y=0,color='k', ls = '--')
```

```
scale = 300 # this will scale the size of the arrow plot
```

```
for J in [4,9,11,12,14,3,12]:
```

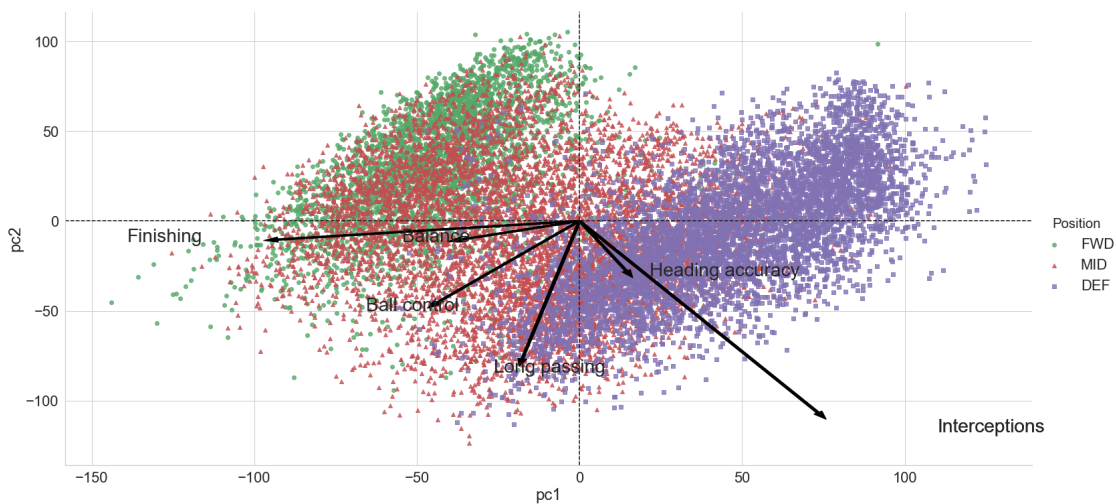
```
    x = v[J,0]
```

```
    y = v[J,1]
```

```
    # make an arrow from the origin to a point at (x,y)
```

```
    ax.arrow(0,0,scale*x,scale*y,color='black',width=1)
```

```
    ax.text(x*scale*1.5,y*scale*1.1,labels_new[J],fontsize=28)
```



In [26]: `#clear`

```
g = sns.lmplot(x = "pc1", y = "pc2", data = df2[df2['Position'] == 'DEF'], hue = "Position",
               markers=["s"], palette=dict(DEF="m"))
ax = g.axes[0,0]
```

```

ax.axvline(x=0,color='k', ls = '--')
ax.axhline(y=0,color='k', ls = '--')

scale = 300 # this will scale the size of the arrow plot

for J in [4,9,11,12,14,3,12]:

    x = v[J,0]
    y = v[J,1]

    # make an arrow from the origin to a point at (x,y)
    ax.arrow(0,0,scale*x,scale*y,color='black',width=1)
    ax.text(x*scale*1.5,y*scale*1.1,labels_new[J],fontsize=28)

```

