

# Feature extraction

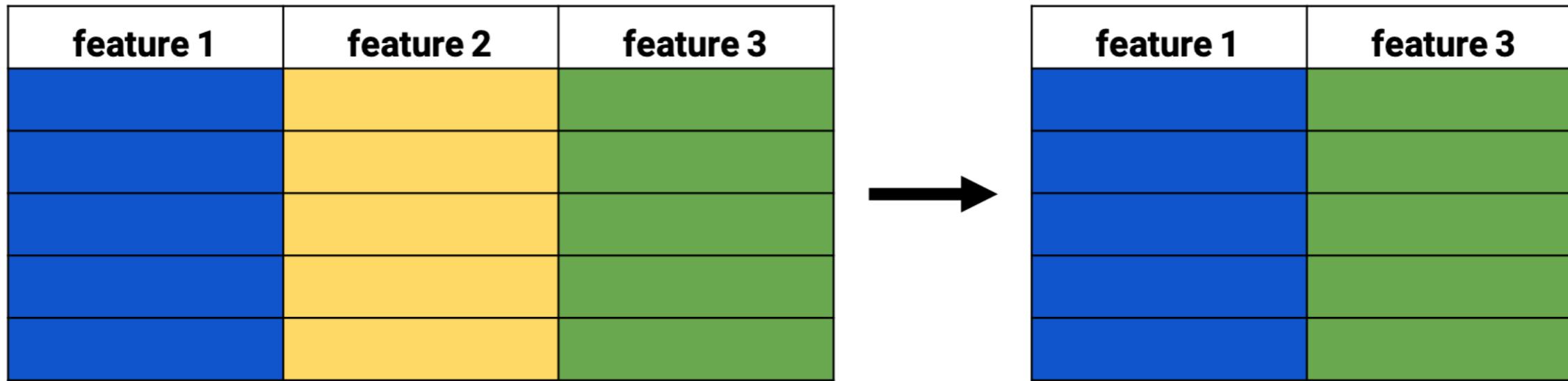
DIMENSIONALITY REDUCTION IN PYTHON



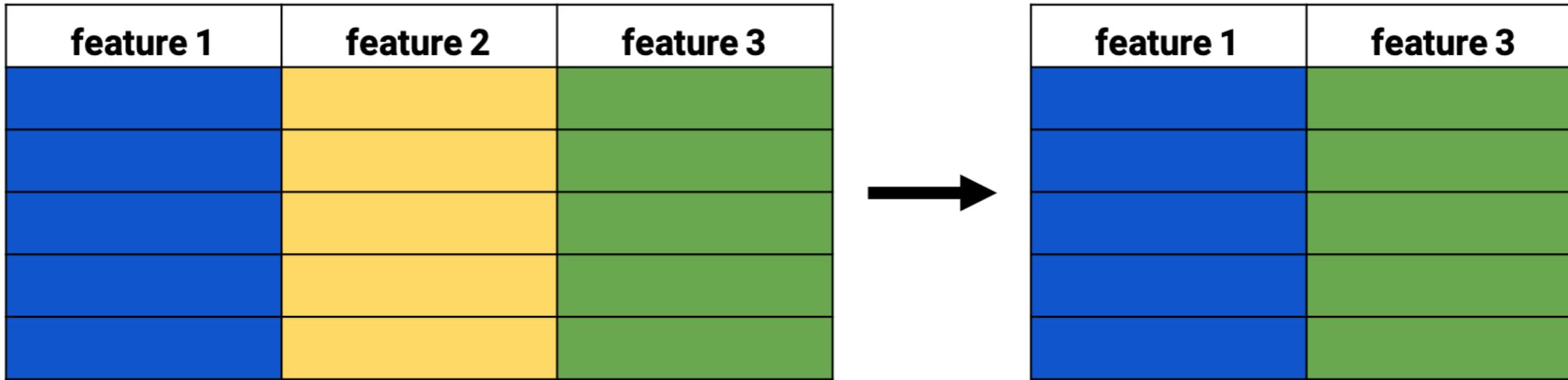
Jeroen Boeye

Machine Learning Engineer,  
Faktion

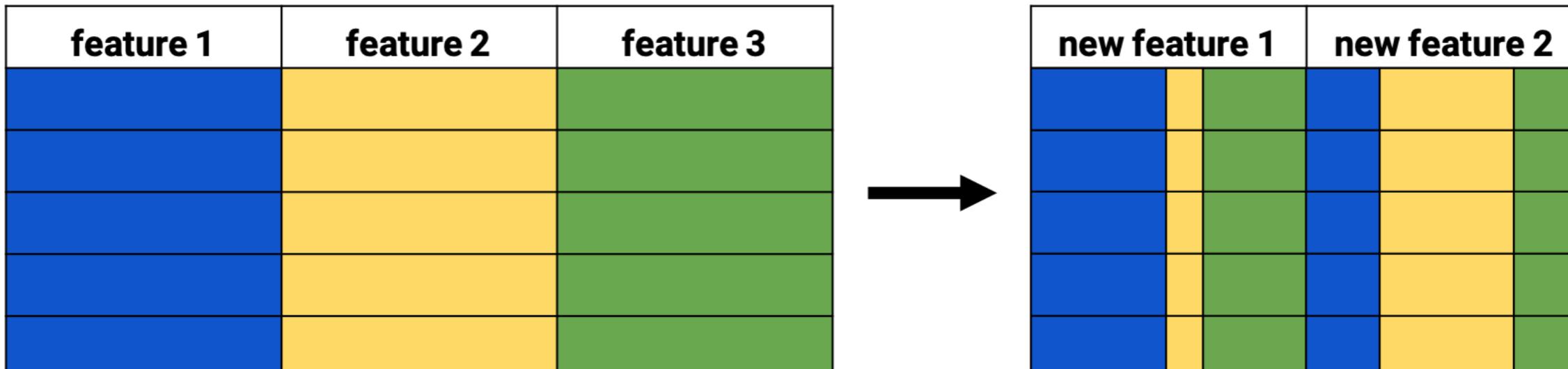
# Feature selection



# Feature selection



# Feature extraction



# Feature generation - BMI

```
df_body[ 'BMI' ] = df_body[ 'Weight kg' ] / df_body[ 'Height m' ] ** 2
```

# Feature generation - BMI

```
df_body[ 'BMI' ] = df_body[ 'Weight kg' ] / df_body[ 'Height m' ] ** 2
```

Weight kg	Height m	BMI
81.5	1.776	25.84
72.6	1.702	25.06
92.9	1.735	30.86

# Feature generation - BMI

```
df_body.drop(['Weight kg', 'Height m'], axis=1)
```

BMI
25.84
25.06
30.86

# Feature generation - averages

left leg mm	right leg mm
882	885
870	869
901	900

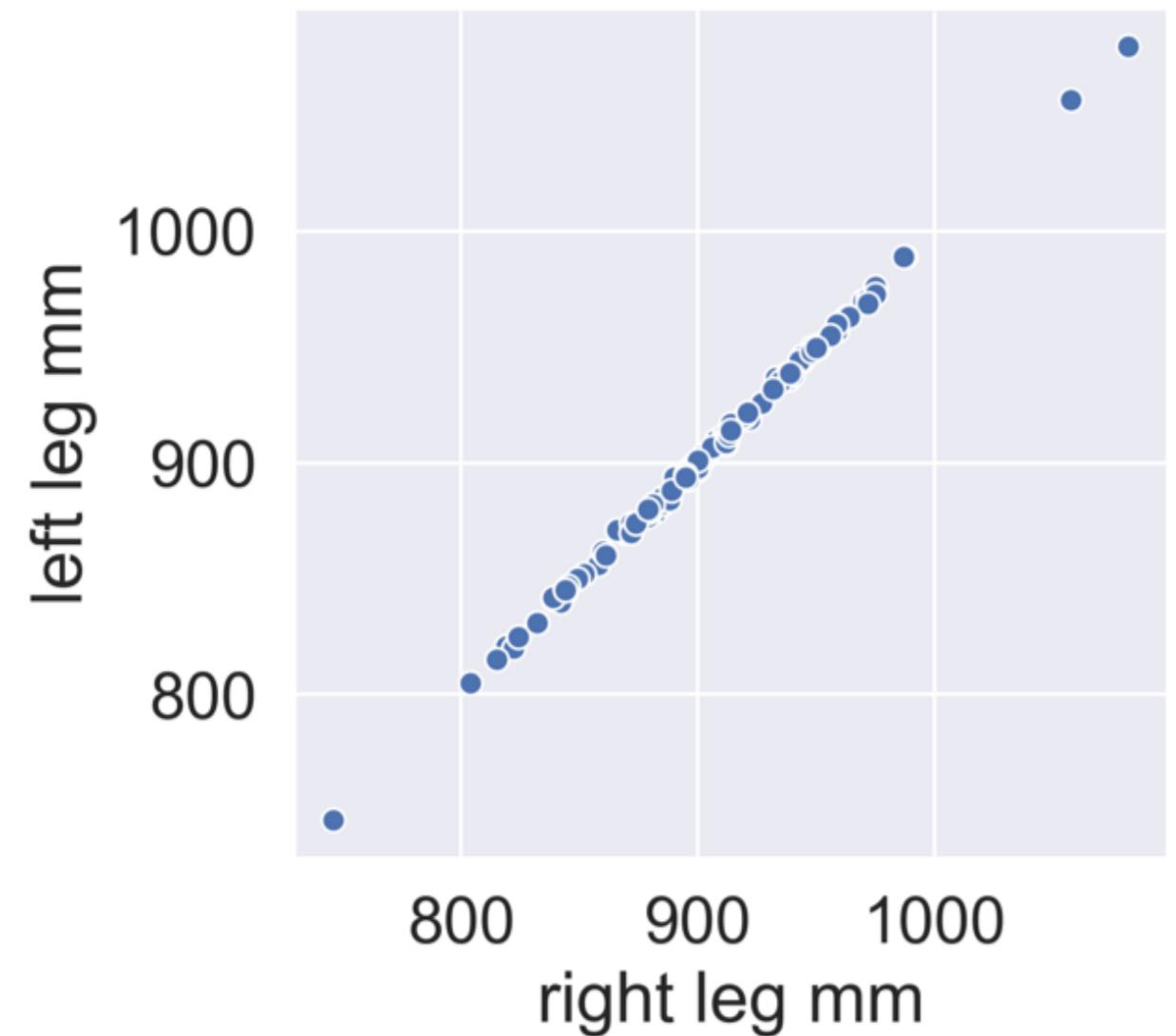
```
leg_df['leg mm'] = leg_df[['right leg mm', 'left leg mm']].mean(axis=1)
```

# Feature generation - averages

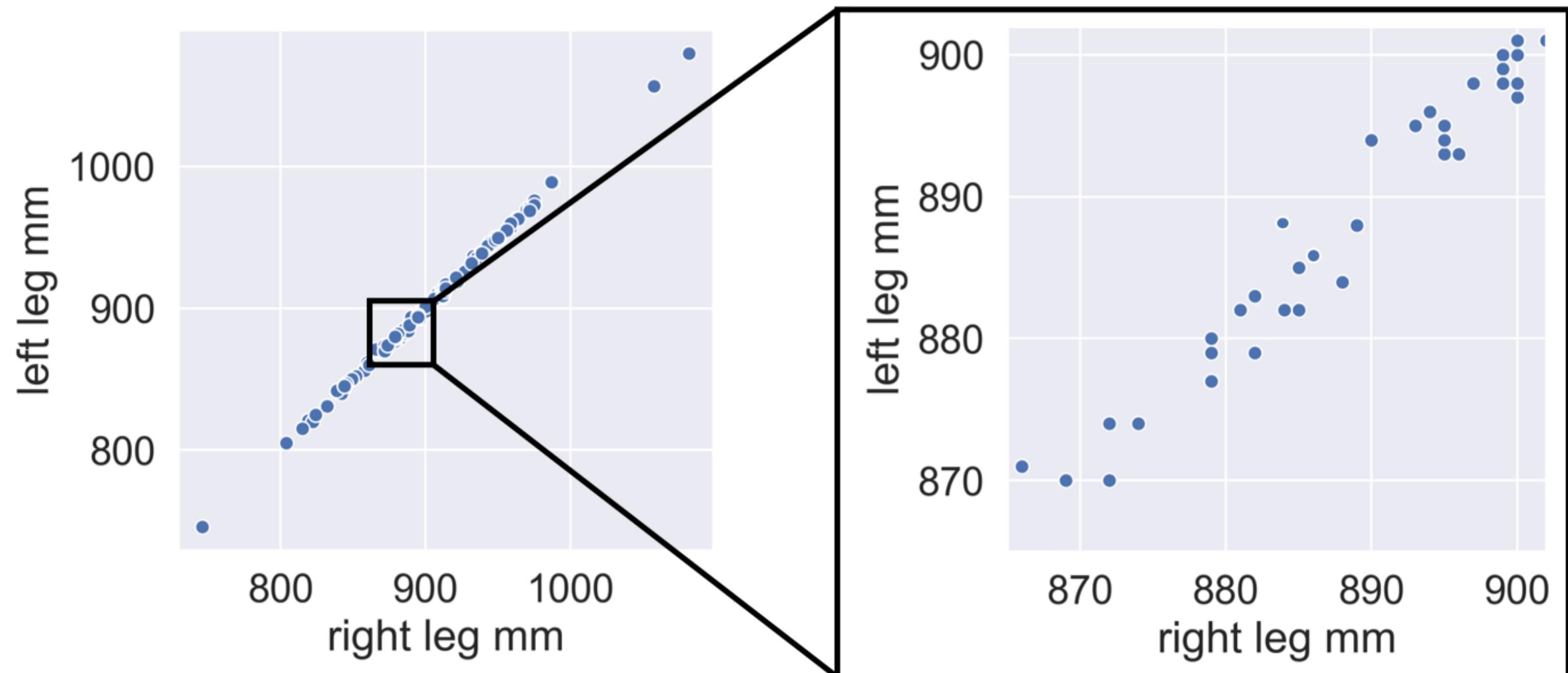
```
leg_df.drop(['right leg mm', 'left leg mm'], axis=1)
```

leg mm
883.5
869.5
900.5

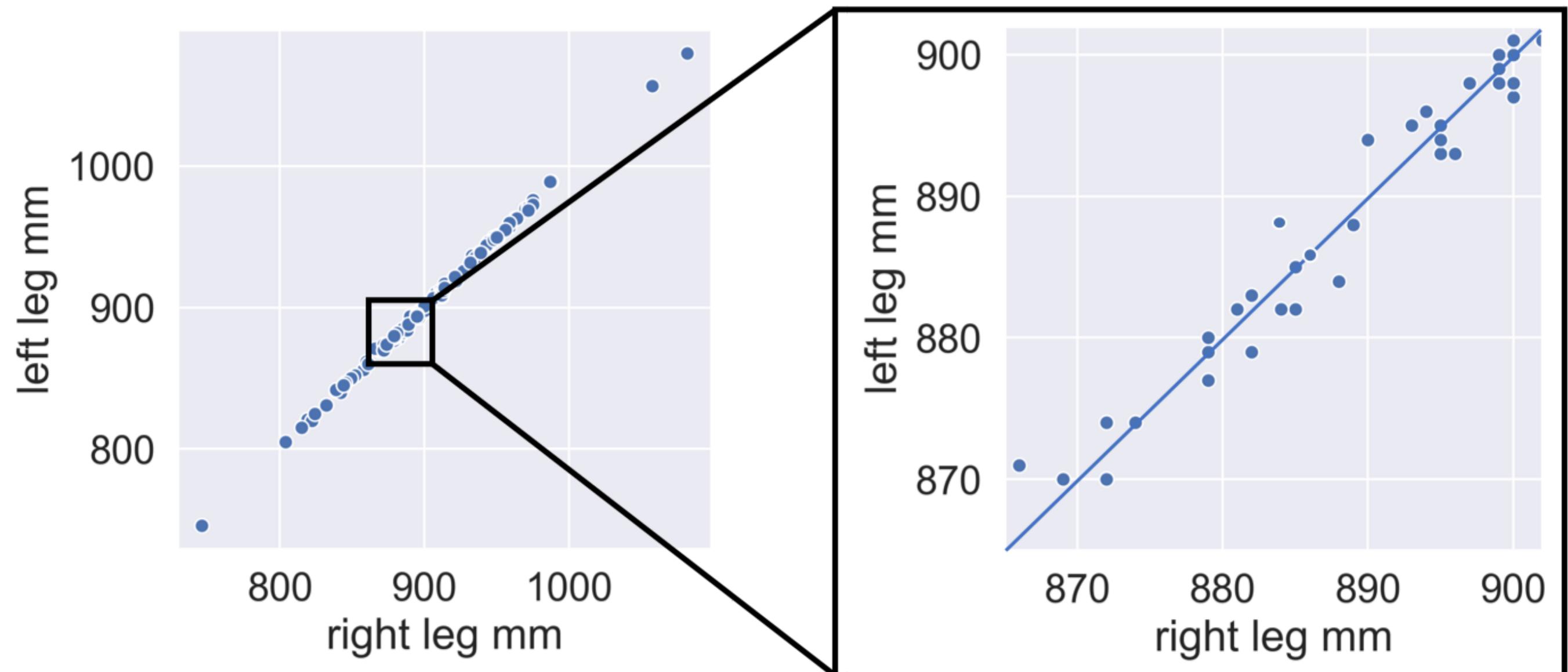
# Cost of taking the average



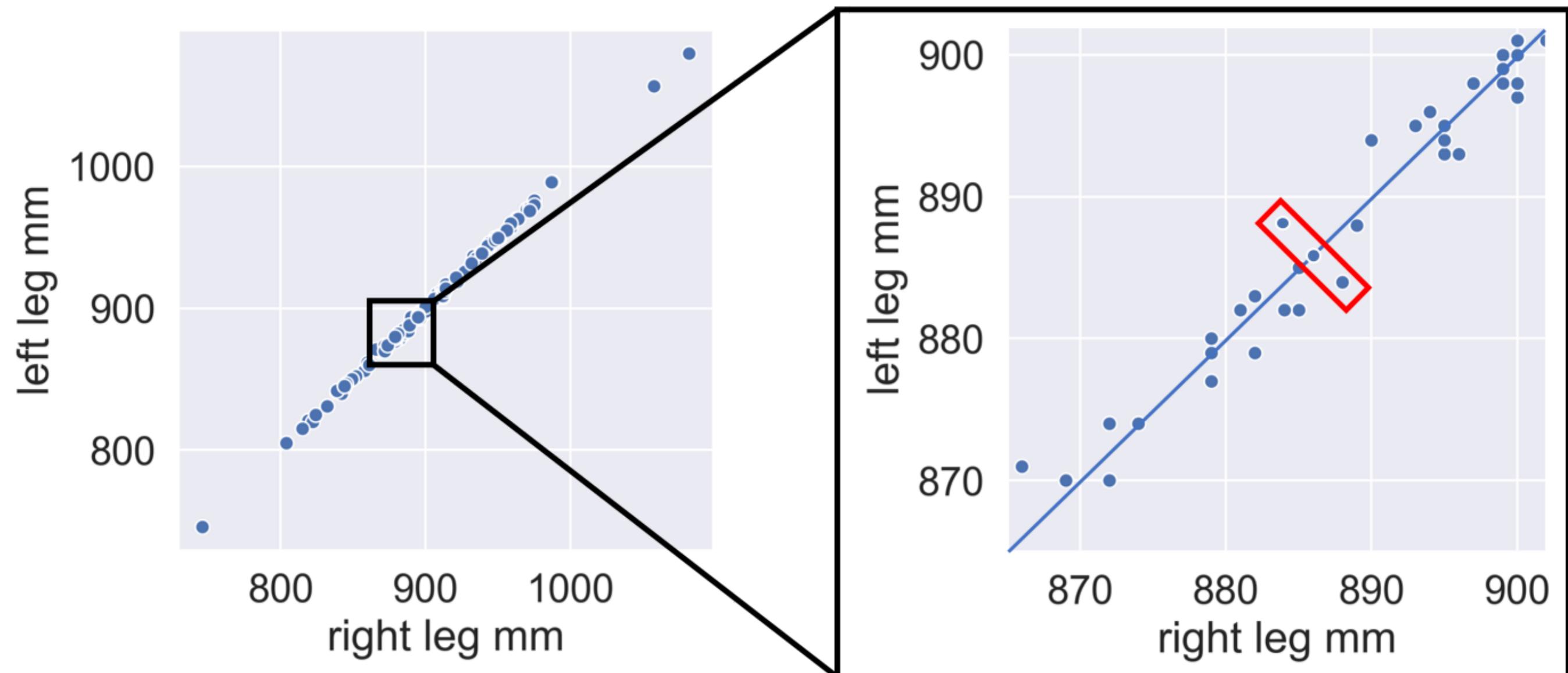
# Cost of taking the average



# Cost of taking the average

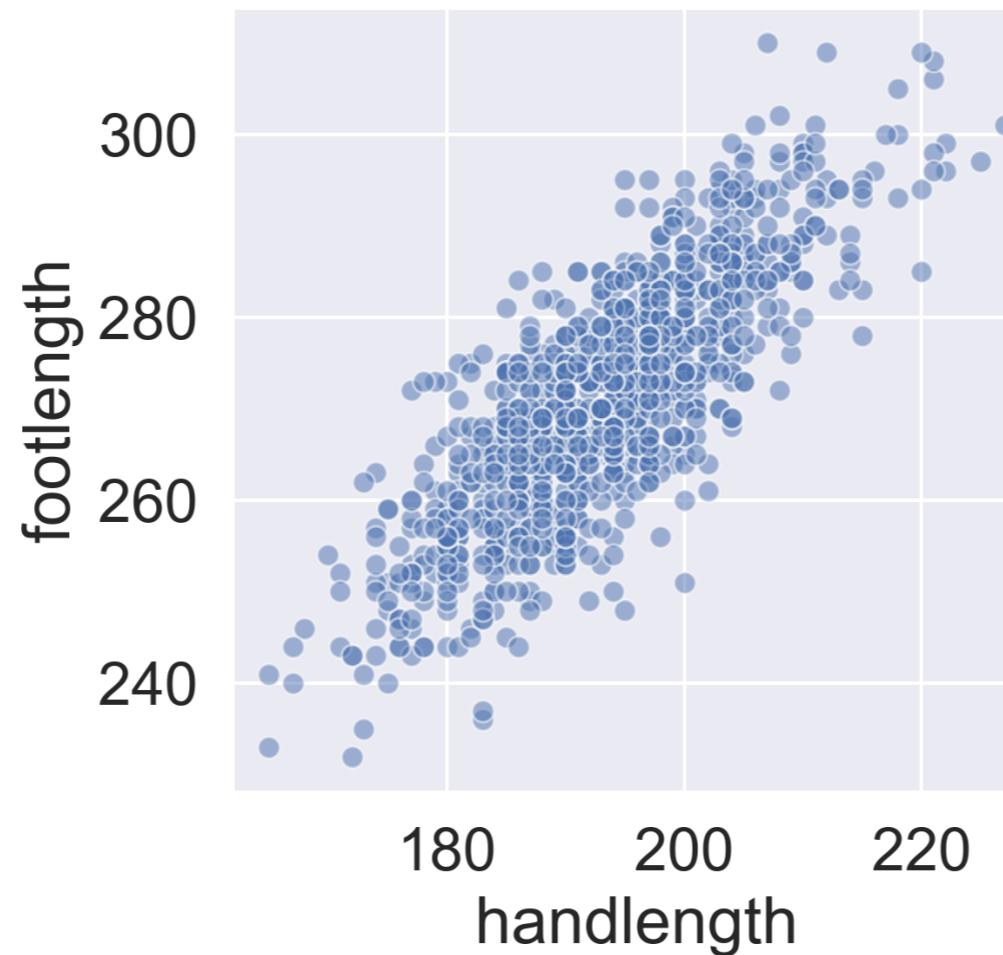


# Cost of taking the average



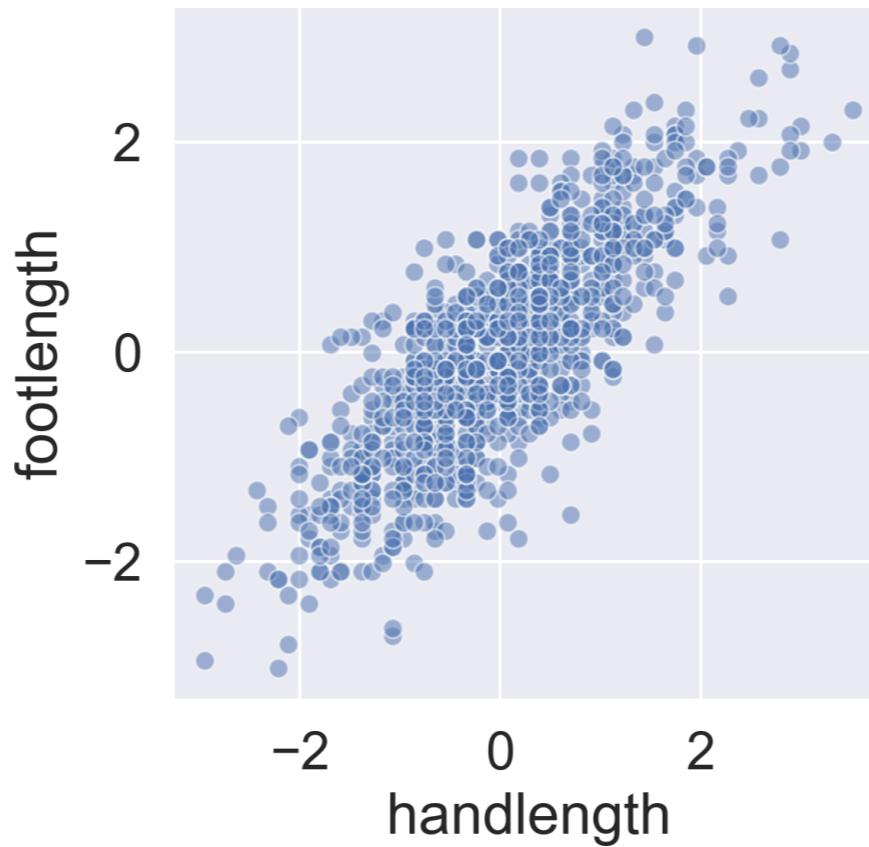
# Intro to PCA

```
sns.scatterplot(data=df, x='handlength', y='footlength')
```



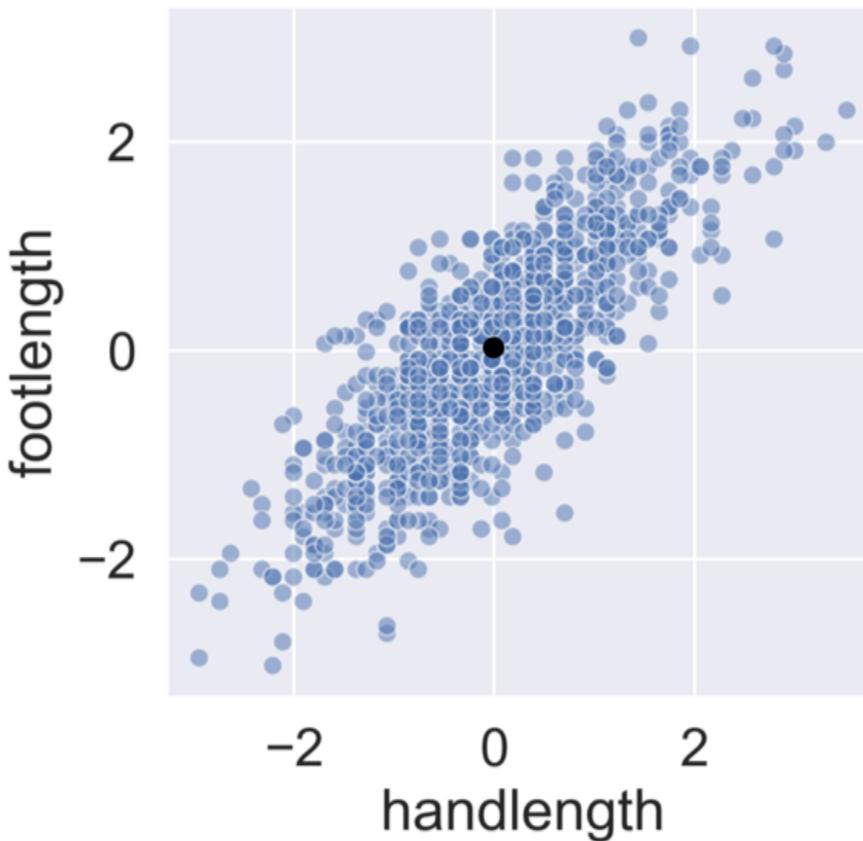
# Intro to PCA

```
scaler = StandardScaler()  
df_std = pd.DataFrame(scaler.fit_transform(df), columns = df.columns)
```



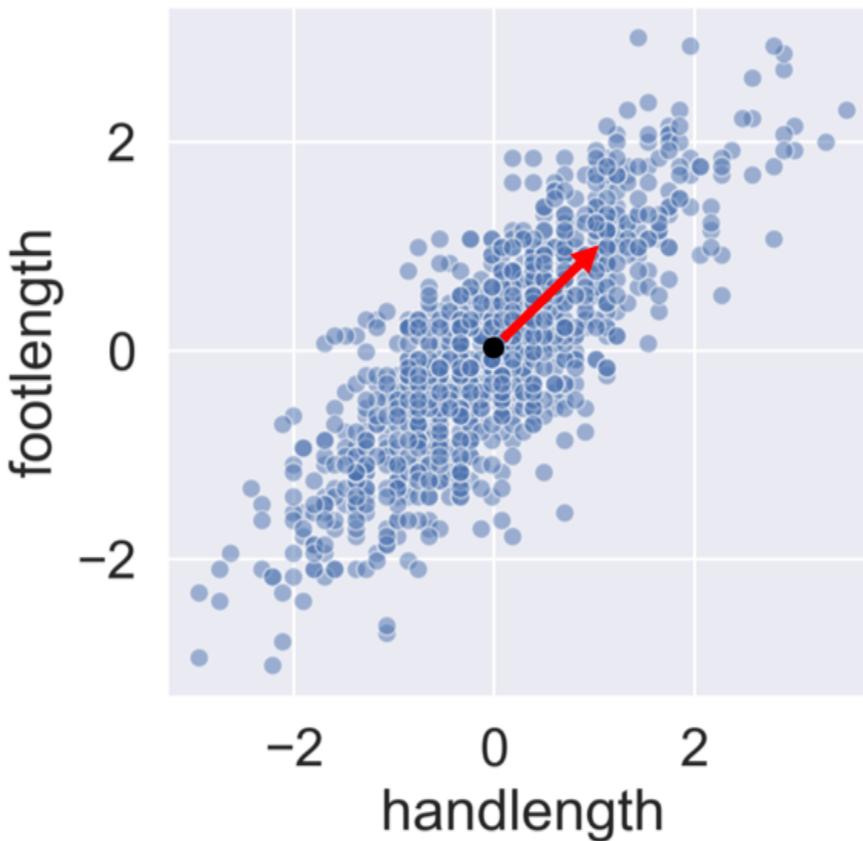
# Intro to PCA

```
scaler = StandardScaler()  
df_std = pd.DataFrame(scaler.fit_transform(df), columns = df.columns)
```



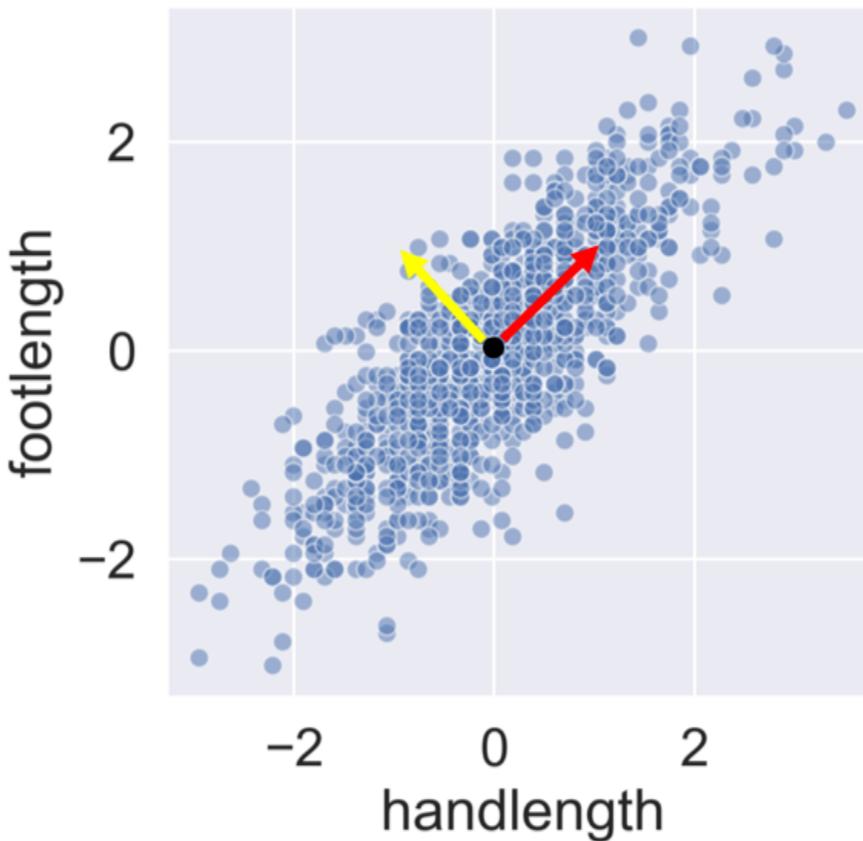
# Intro to PCA

```
scaler = StandardScaler()  
df_std = pd.DataFrame(scaler.fit_transform(df), columns = df.columns)
```



# Intro to PCA

```
scaler = StandardScaler()  
df_std = pd.DataFrame(scaler.fit_transform(df), columns = df.columns)
```



# Let's practice!

## DIMENSIONALITY REDUCTION IN PYTHON

# Principal component analysis

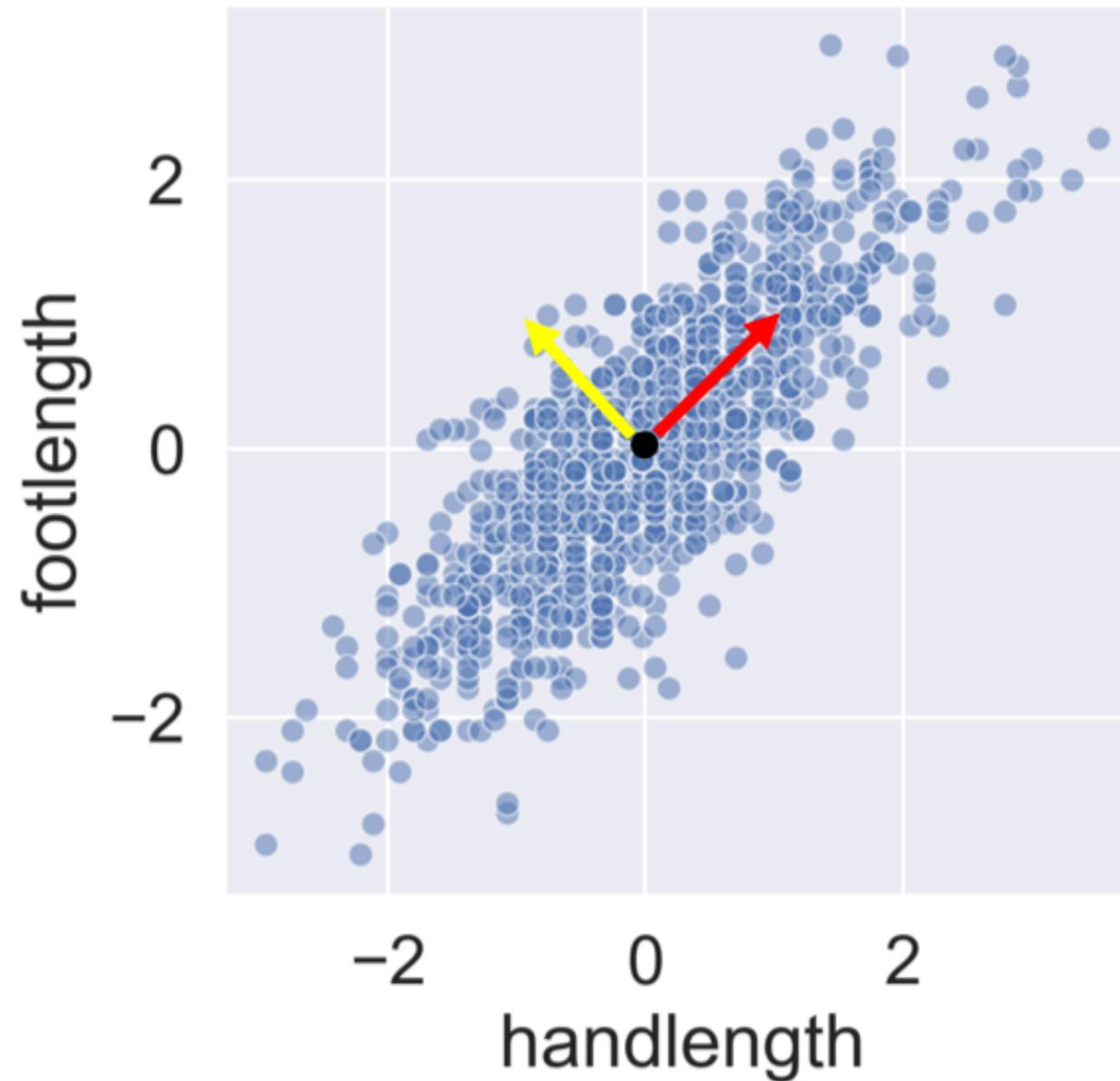
DIMENSIONALITY REDUCTION IN PYTHON



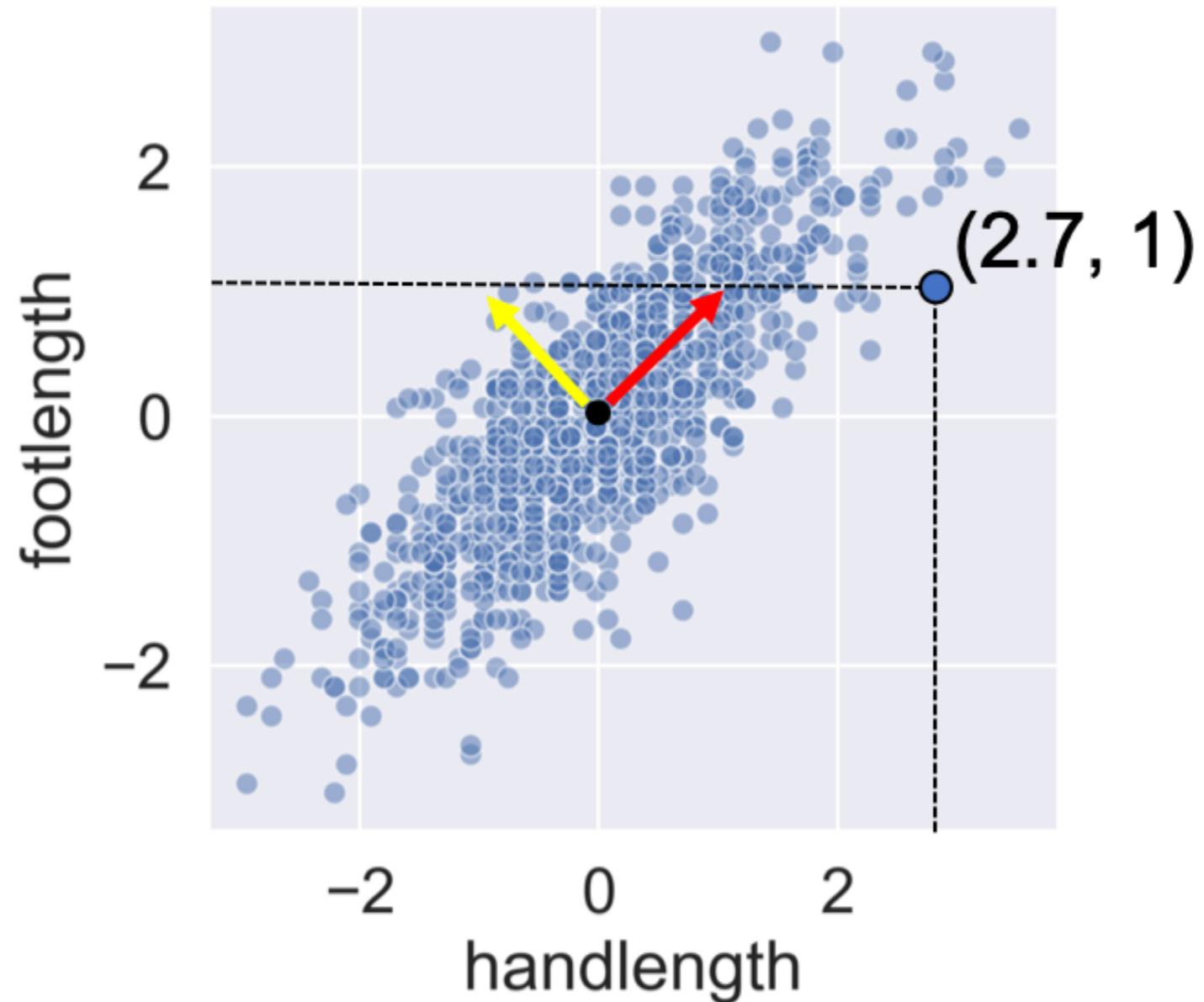
Jeroen Boeye

Machine Learning Engineer,  
Faktion

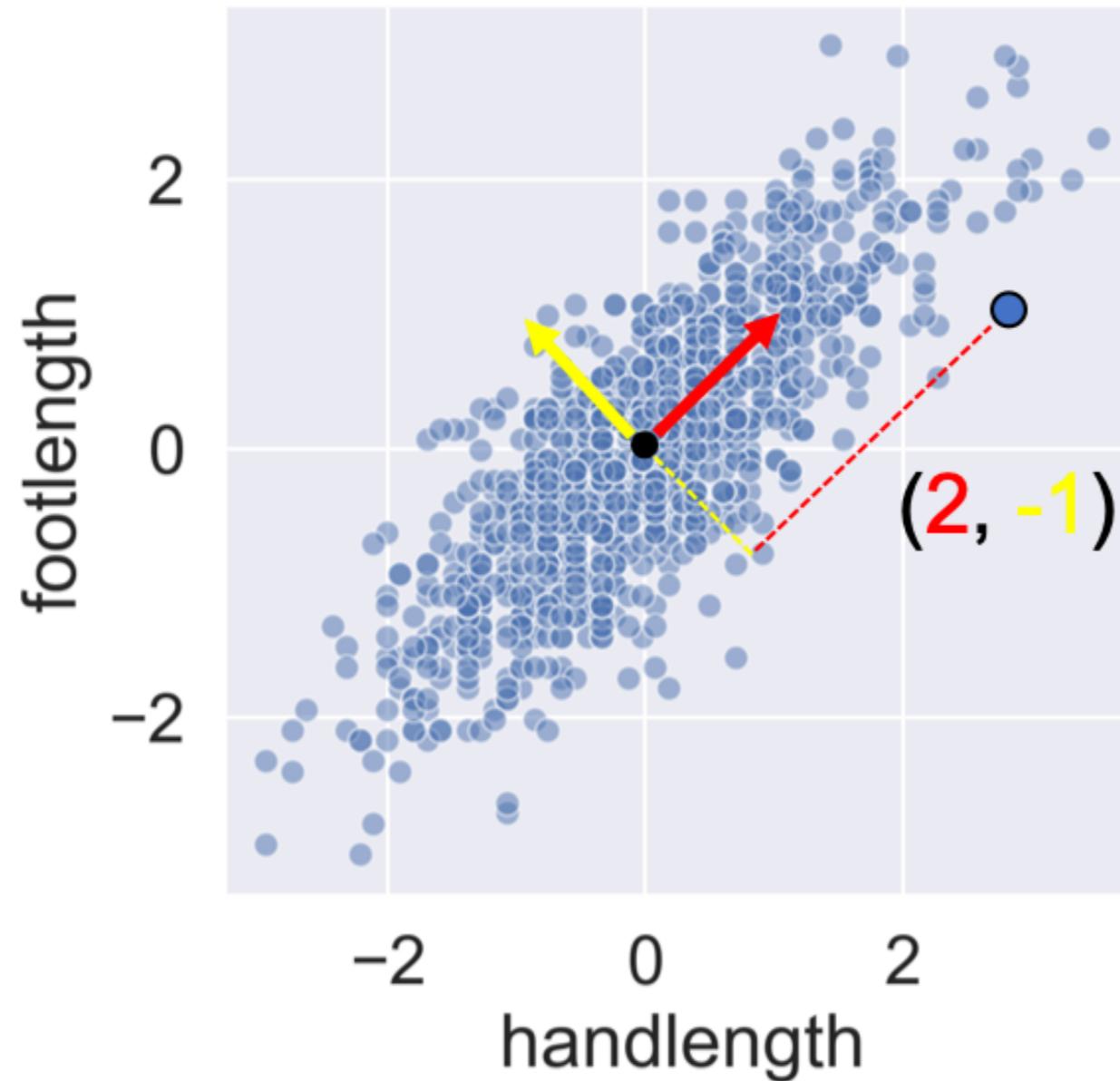
# PCA concept



# PCA concept



# PCA concept

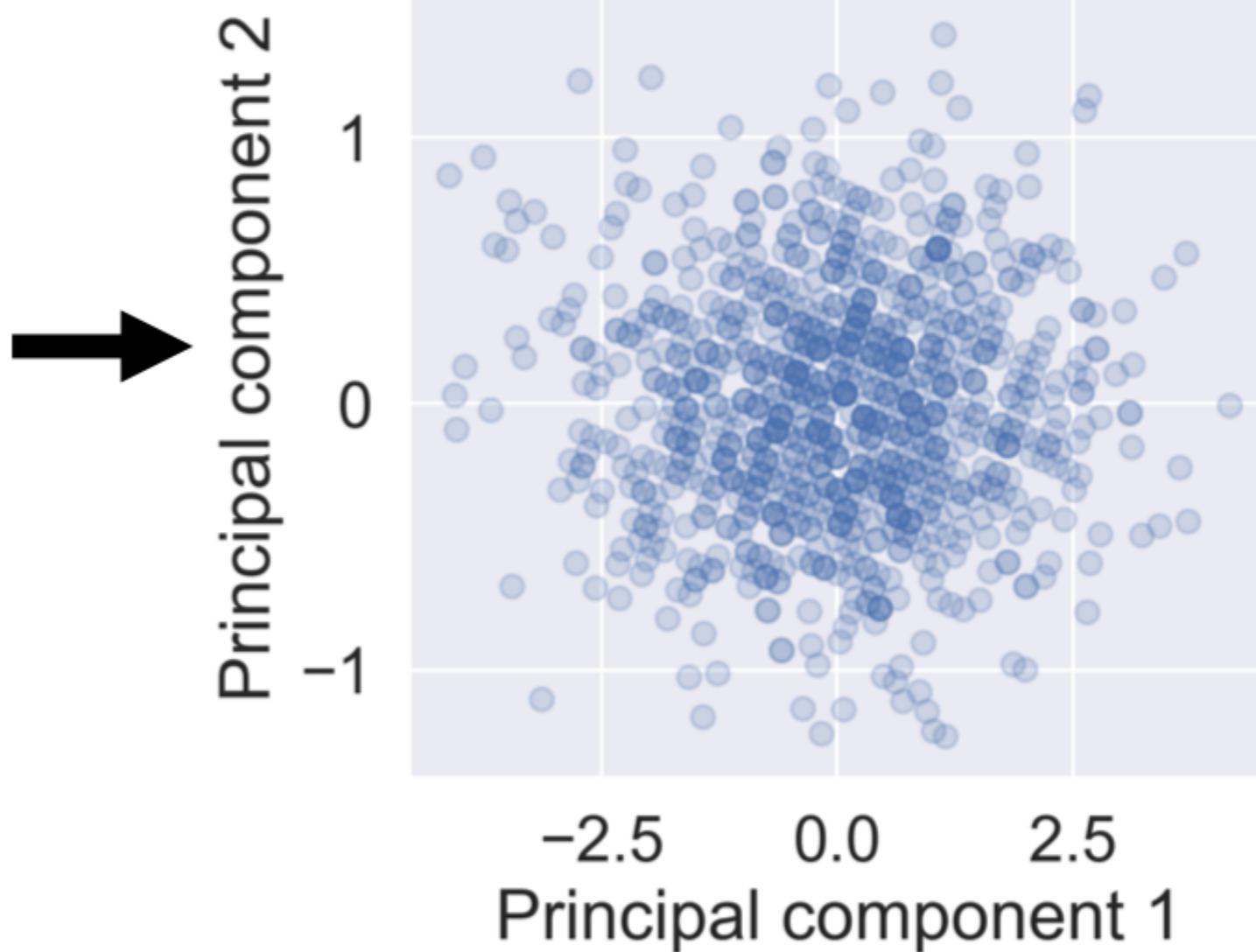
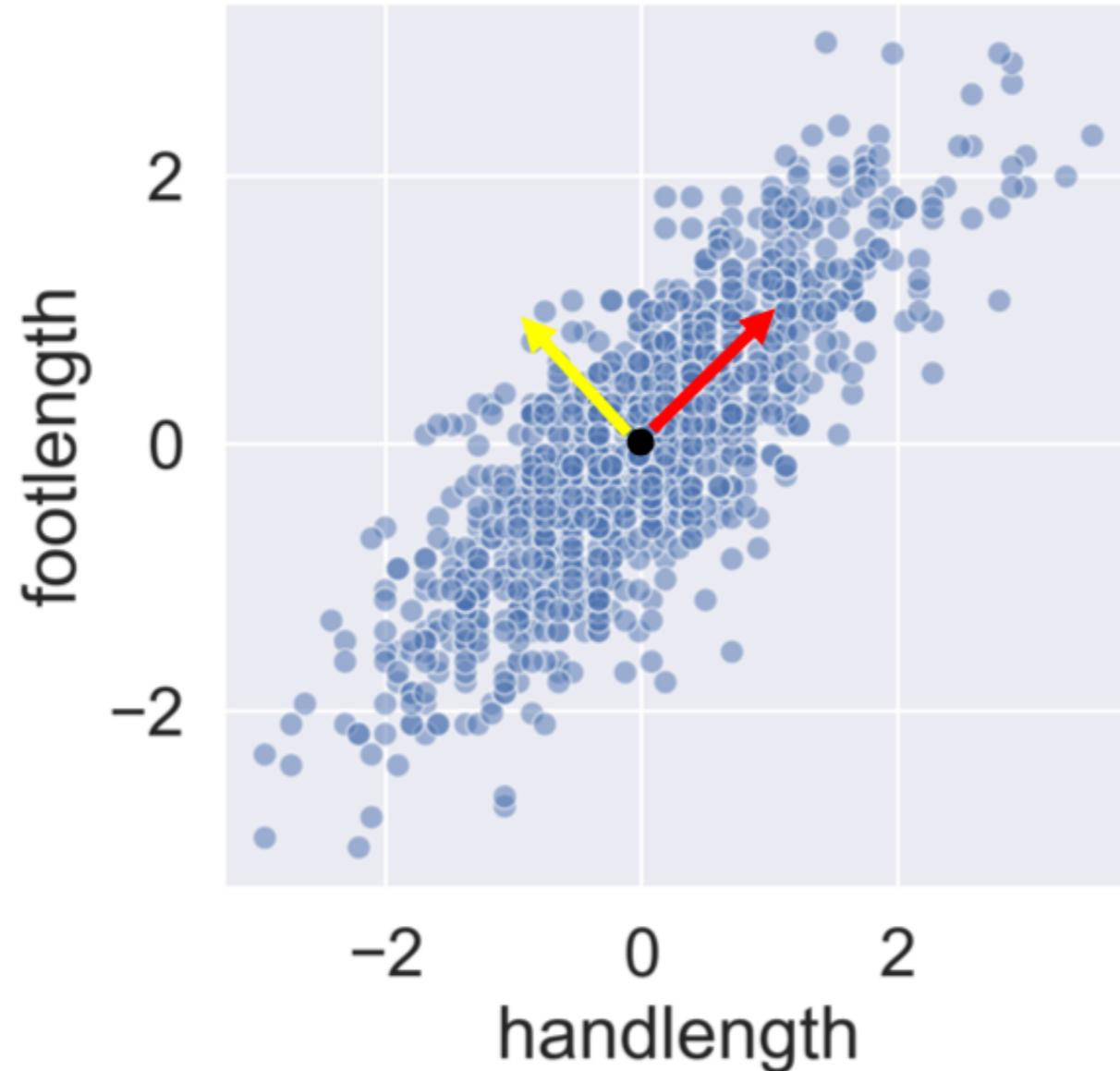


# Calculating the principal components

```
from sklearn.preprocessing import StandardScaler  
  
scaler = StandardScaler()  
std_df = scaler.fit_transform(df)  
  
from sklearn.decomposition import PCA  
  
pca = PCA()  
print(pca.fit_transform(std_df))
```

```
[ [-0.08320426 -0.12242952]  
 [ 0.31478004  0.57048158]  
 ...  
 [-0.5609523   0.13713944]  
 [-0.0448304  -0.37898246]]
```

# PCA removes correlation

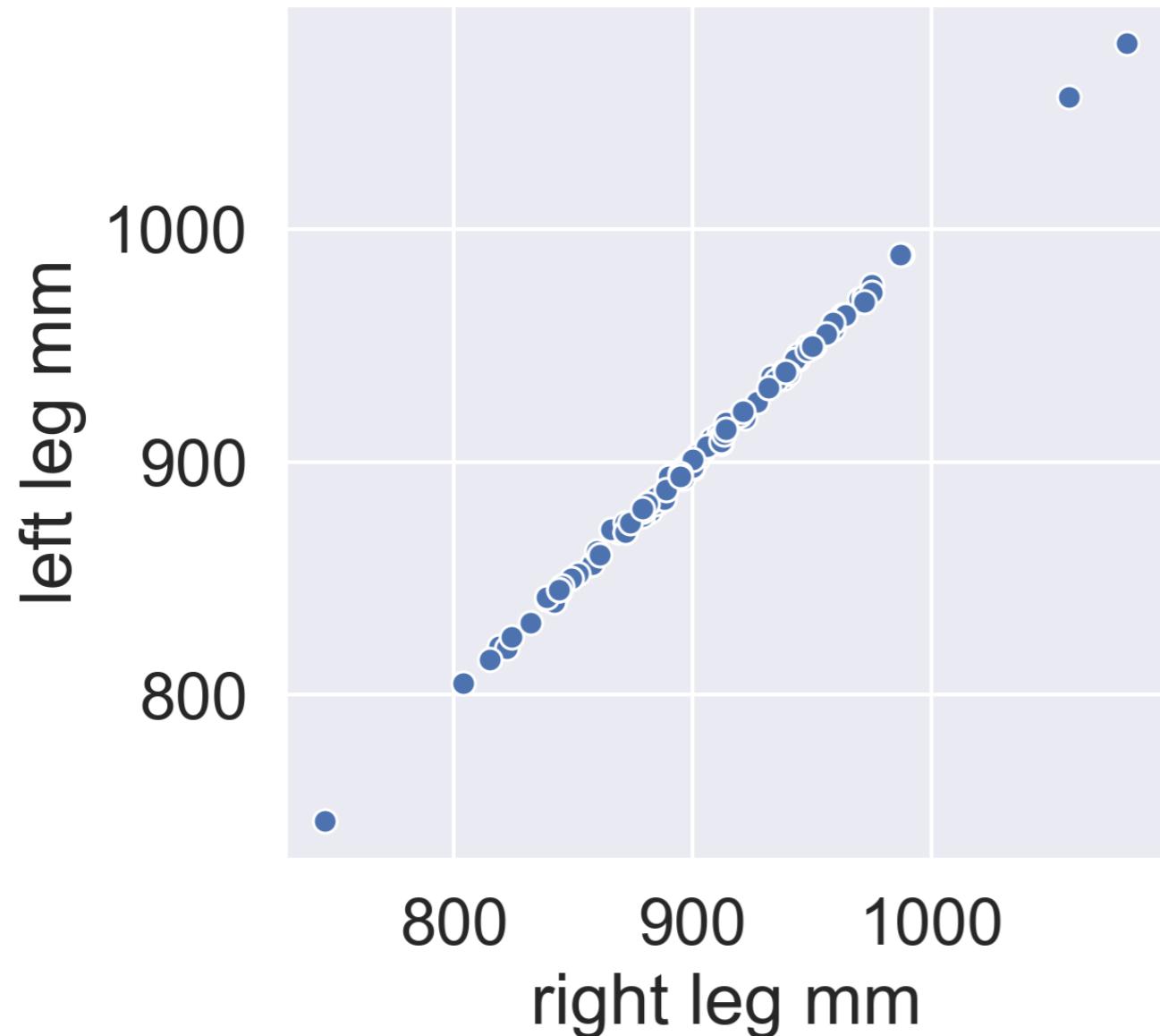


# Principal component explained variance ratio

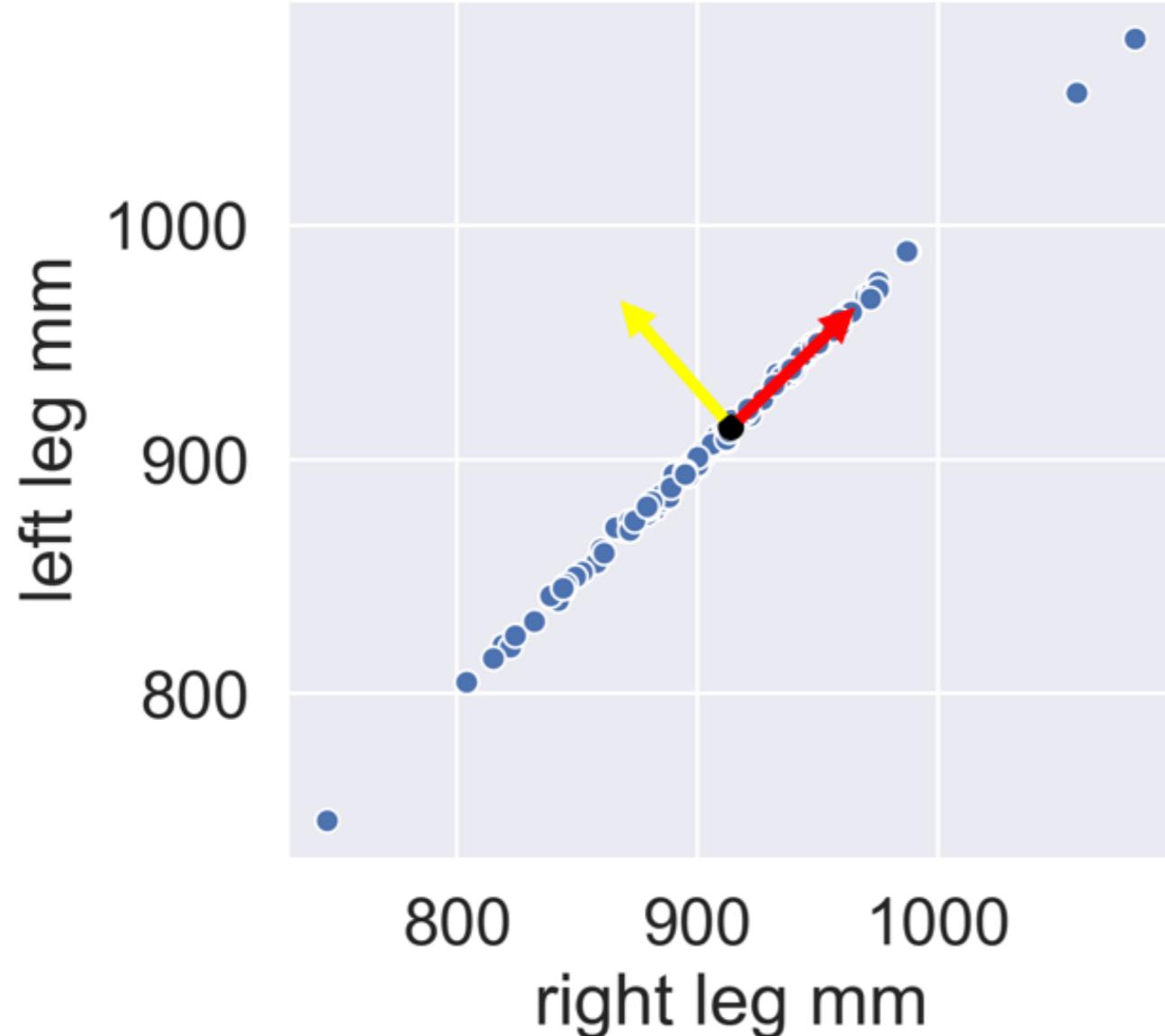
```
from sklearn.decomposition import PCA  
  
pca = PCA()  
  
pca.fit(std_df)  
  
print(pca.explained_variance_ratio_)
```

```
array([0.90, 0.10])
```

# PCA for dimensionality reduction



# PCA for dimensionality reduction



```
print(pca.explained_variance_ratio_)
```

```
array([0.9997, 0.0003])
```

# PCA for dimensionality reduction

```
pca = PCA()  
  
pca.fit(ansur_std_df)  
  
print(pca.explained_variance_ratio_)
```

```
array([0.44, 0.18, 0.04, 0.03, 0.02, 0.02, 0.02, 0.01, 0.01, 0.01, 0.01,  
     0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,  
     0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,  
     0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,  
     ...  
     0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,  
     0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,  
     0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01])
```

# PCA for dimensionality reduction

```
pca = PCA()  
  
pca.fit(ansur_std_df)  
  
print(pca.explained_variance_ratio_.cumsum())
```

```
array([0.44, 0.62, 0.66, 0.69, 0.72, 0.74, 0.76, 0.77, 0.79, 0.8 , 0.81,  
     0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.87, 0.88, 0.89, 0.89, 0.9 ,  
     0.9 , 0.91, 0.92, 0.92, 0.92, 0.93, 0.93, 0.94, 0.94, 0.94, 0.95,  
     ...  
     0.99, 0.99, 0.99, 0.99, 0.99, 1. , 1. , 1. , 1. , 1. , 1. , 1. ,  
     1. , 1. , 1. , 1. , 1. , 1. , 1. , 1. , 1. , 1. , 1. , 1. ,  
     1. , 1. , 1. , 1. , 1. ])
```

# Let's practice!

## DIMENSIONALITY REDUCTION IN PYTHON

# PCA applications

DIMENSIONALITY REDUCTION IN PYTHON



Jeroen Boeye

Machine Learning Engineer,  
Faktion

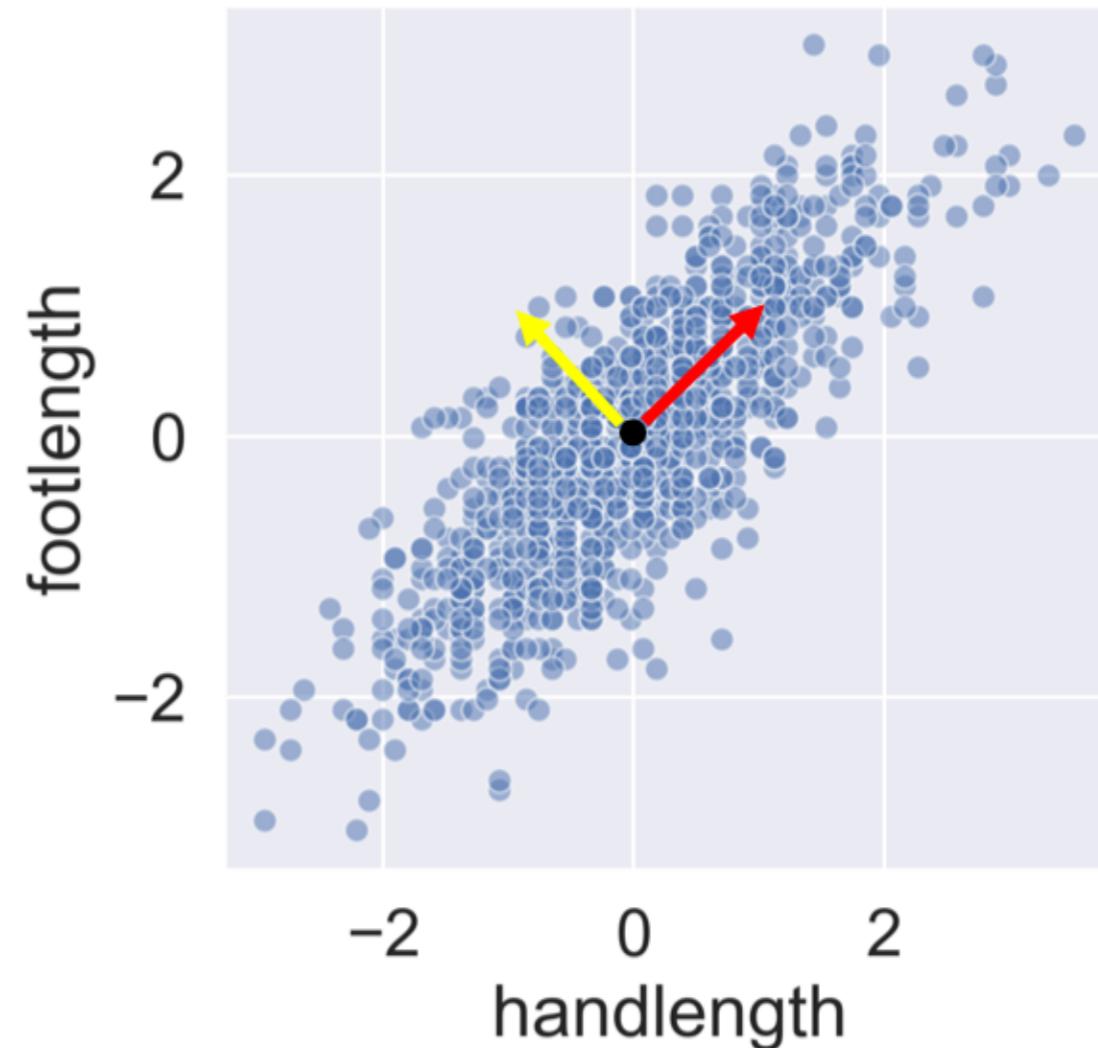
# Understanding the components

```
print(pca.components_)
```

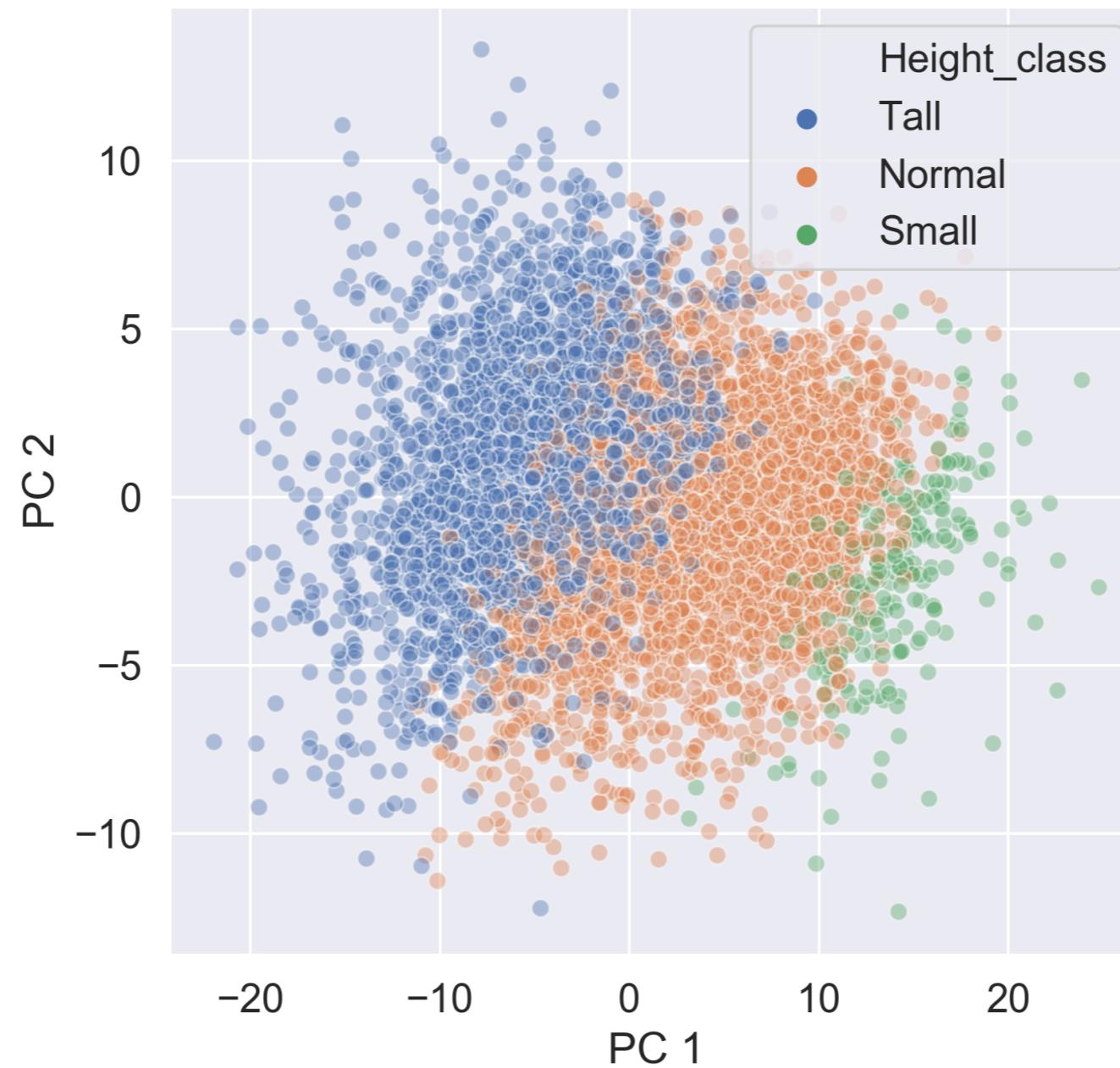
```
array([[ 0.71,  0.71],  
       [-0.71,  0.71]])
```

PC 1 = 0.71 x Hand length + 0.71 x Foot length

PC 2 = -0.71 x Hand length + 0.71 x Foot length



# PCA for data exploration



# PCA in a pipeline

```
from sklearn.preprocessing import StandardScaler  
from sklearn.decomposition import PCA  
from sklearn.pipeline import Pipeline  
  
pipe = Pipeline([  
    ('scaler', StandardScaler()),  
    ('reducer', PCA())])  
  
pc = pipe.fit_transform(ansur_df)  
  
print(pc[:, :2])
```

```
array([[-3.46114925,  1.5785215 ],  
      [ 0.90860615,  2.02379935],  
      ...,  
      [10.7569818 , -1.40222755],  
      [ 7.64802025,  1.07406209]])
```

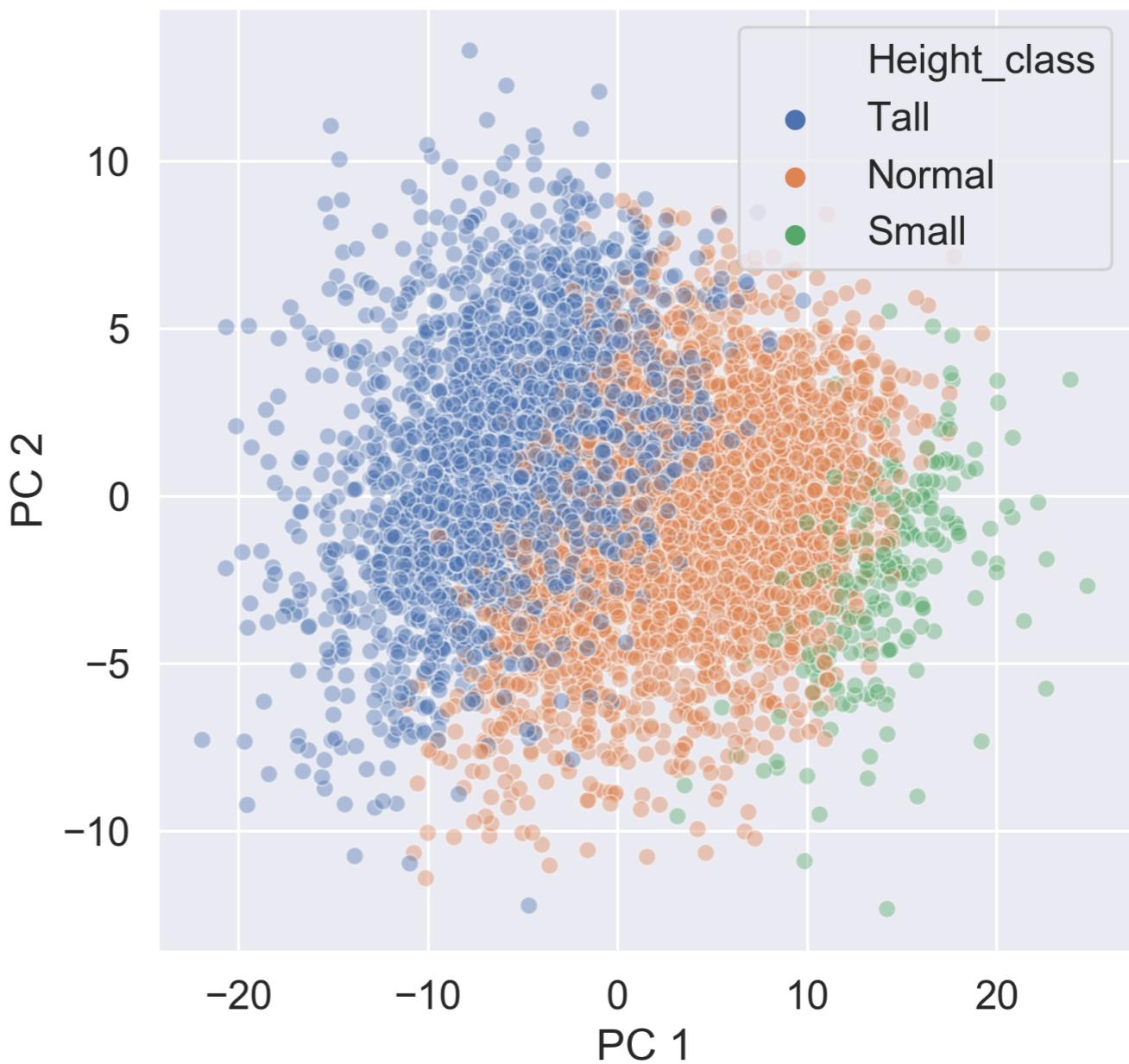
# Checking the effect of categorical features

```
print(ansur_categories.head())
```

Branch	Component	Gender	BMI_class	Height_class
Combat Arms	Regular Army	Male	Overweight	Tall
Combat Support	Regular Army	Male	Overweight	Normal
Combat Support	Regular Army	Male	Overweight	Normal
Combat Service Support	Regular Army	Male	Overweight	Normal
Combat Service Support	Regular Army	Male	Overweight	Tall

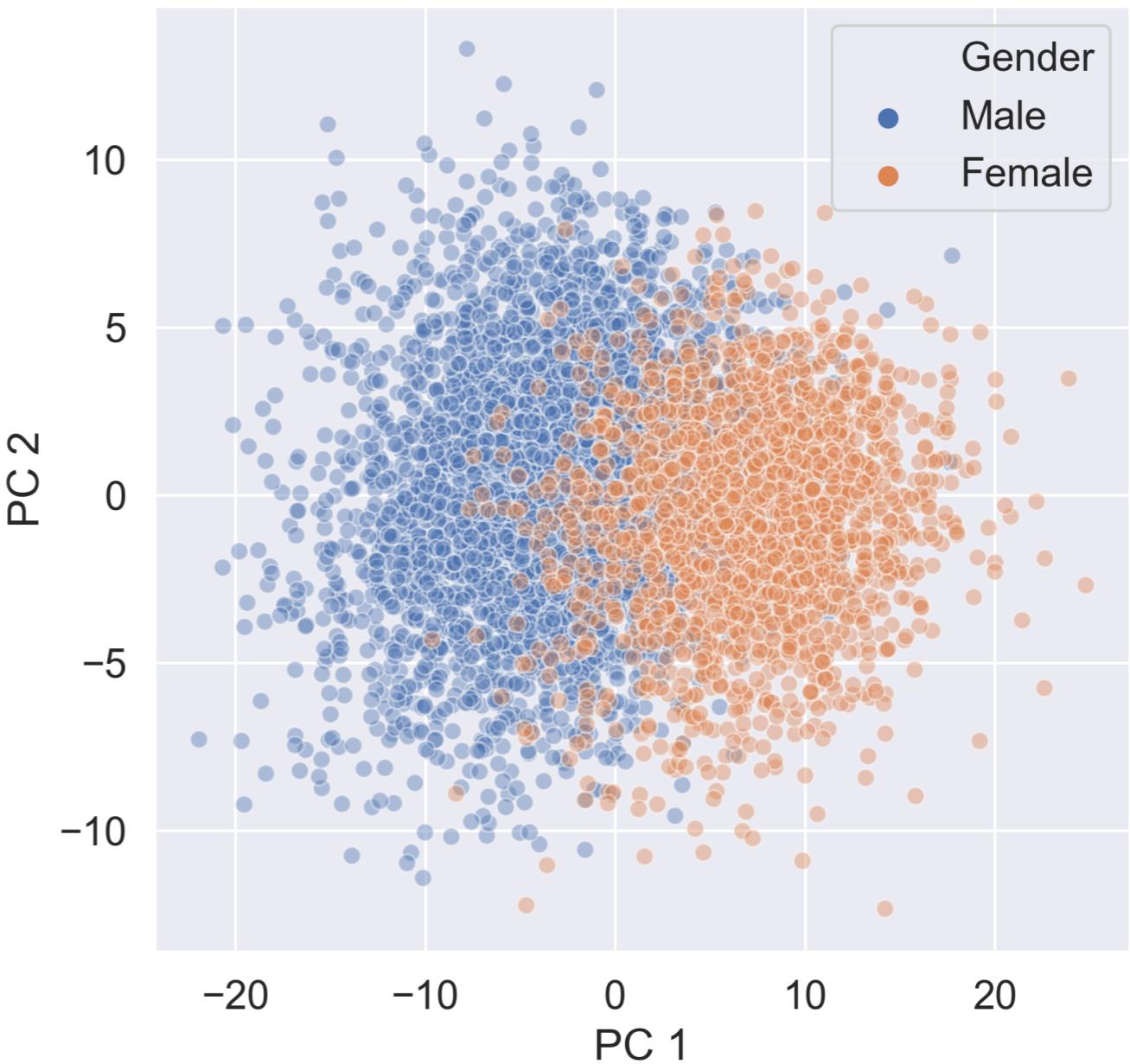
# Checking the effect of categorical features

```
ansur_categories['PC 1'] = pc[:,0]
ansur_categories['PC 2'] = pc[:,1]
sns.scatterplot(data=ansur_categories,
                 x='PC 1', y='PC 2',
                 hue='Height_class', alpha=0.4)
```



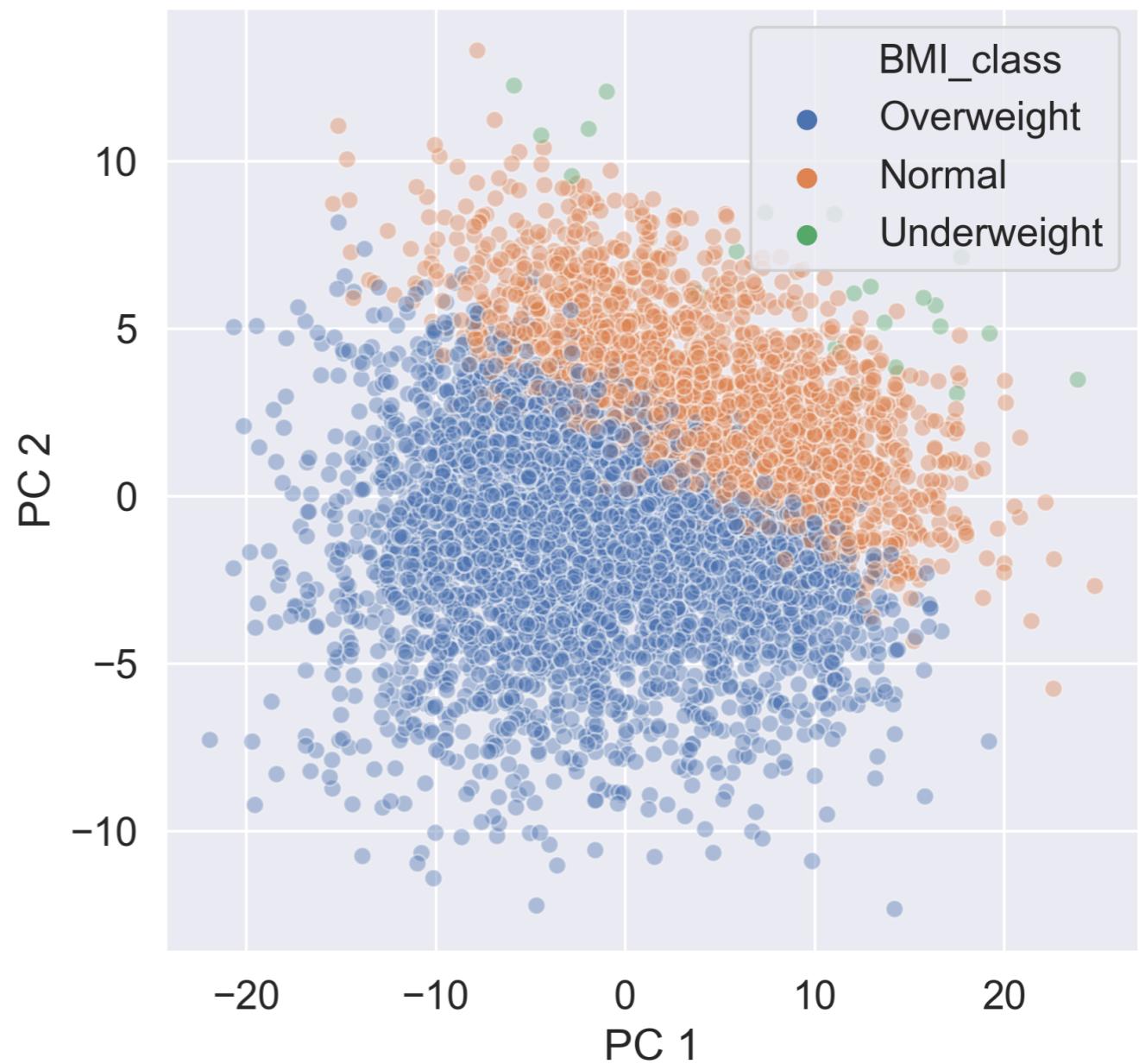
# Checking the effect of categorical features

```
sns.scatterplot(data=ansur_categories,  
                 x='PC 1', y='PC 2',  
                 hue='Gender', alpha=0.4)
```



# Checking the effect of categorical features

```
sns.scatterplot(data=ansur_categories,  
                 x='PC 1', y='PC 2',  
                 hue='BMI_class', alpha=0.4)
```



# PCA in a model pipeline

```
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('reducer', PCA(n_components=3)),
    ('classifier', RandomForestClassifier())])

pipe.fit(X_train, y_train)

print(pipe.steps[1])
```

```
('reducer',
PCA(copy=True, iterated_power='auto', n_components=3, random_state=None,
svd_solver='auto', tol=0.0, whiten=False))
```

# PCA in a model pipeline

```
pipe.steps[1][1].explained_variance_ratio_.cumsum()
```

```
array([0.56, 0.69, 0.74])
```

```
print(pipe.score(X_test, y_test))
```

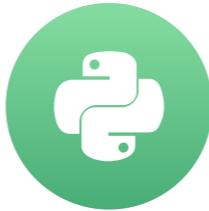
```
0.986
```

# Let's practice!

## DIMENSIONALITY REDUCTION IN PYTHON

# Principal Component selection

DIMENSIONALITY REDUCTION IN PYTHON



Jeroen Boeye

Machine Learning Engineer,  
Faktion

# Setting an explained variance threshold

```
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('reducer', PCA(n_components=0.9))])

# Fit the pipe to the data
pipe.fit(poke_df)

print(len(pipe.steps[1][1].components_))
```

5

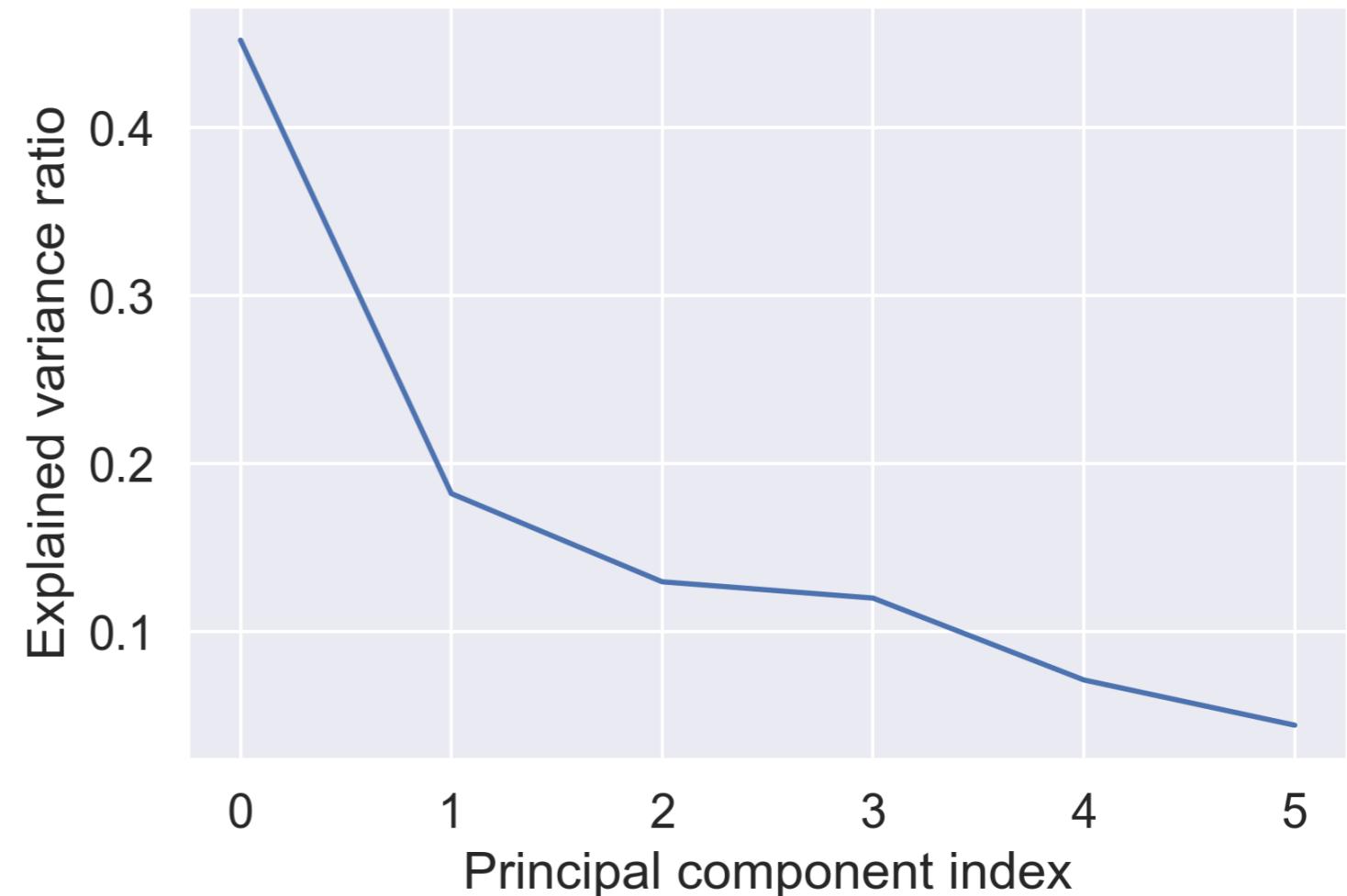
# An optimal number of components

```
pipe.fit(poke_df)

var = pipe.steps[1][1].explained_variance_ratio_

plt.plot(var)

plt.xlabel('Principal component index')
plt.ylabel('Explained variance ratio')
plt.show()
```



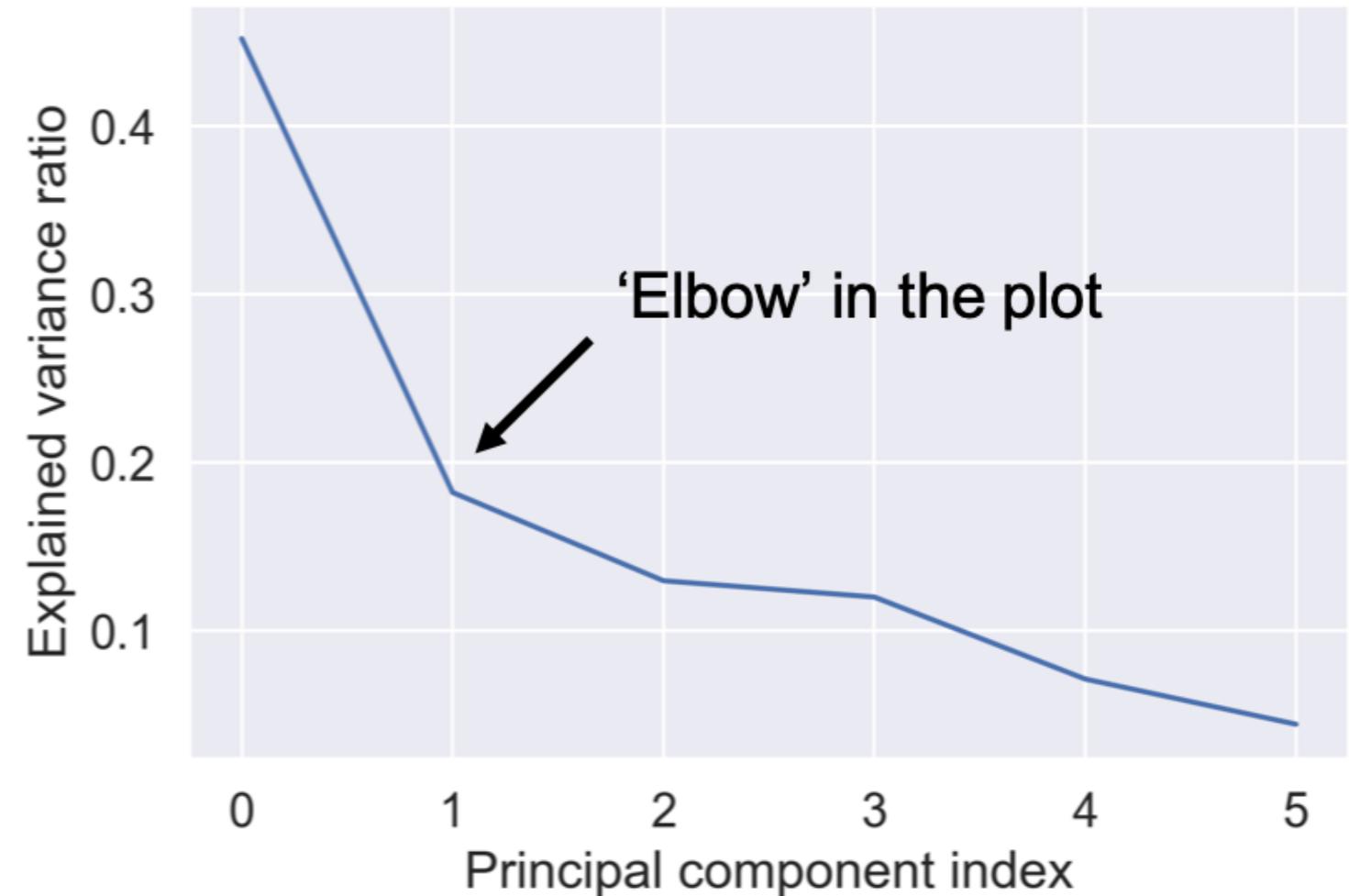
# An optimal number of components

```
pipe.fit(poke_df)

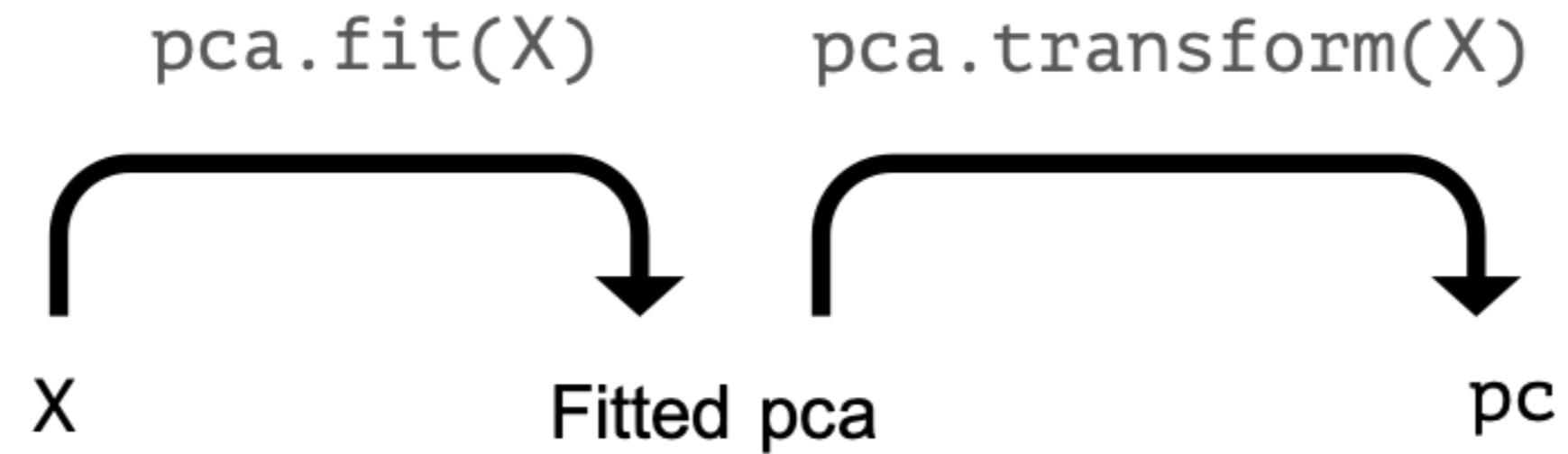
var = pipe.steps[1][1].explained_variance_ratio_

plt.plot(var)

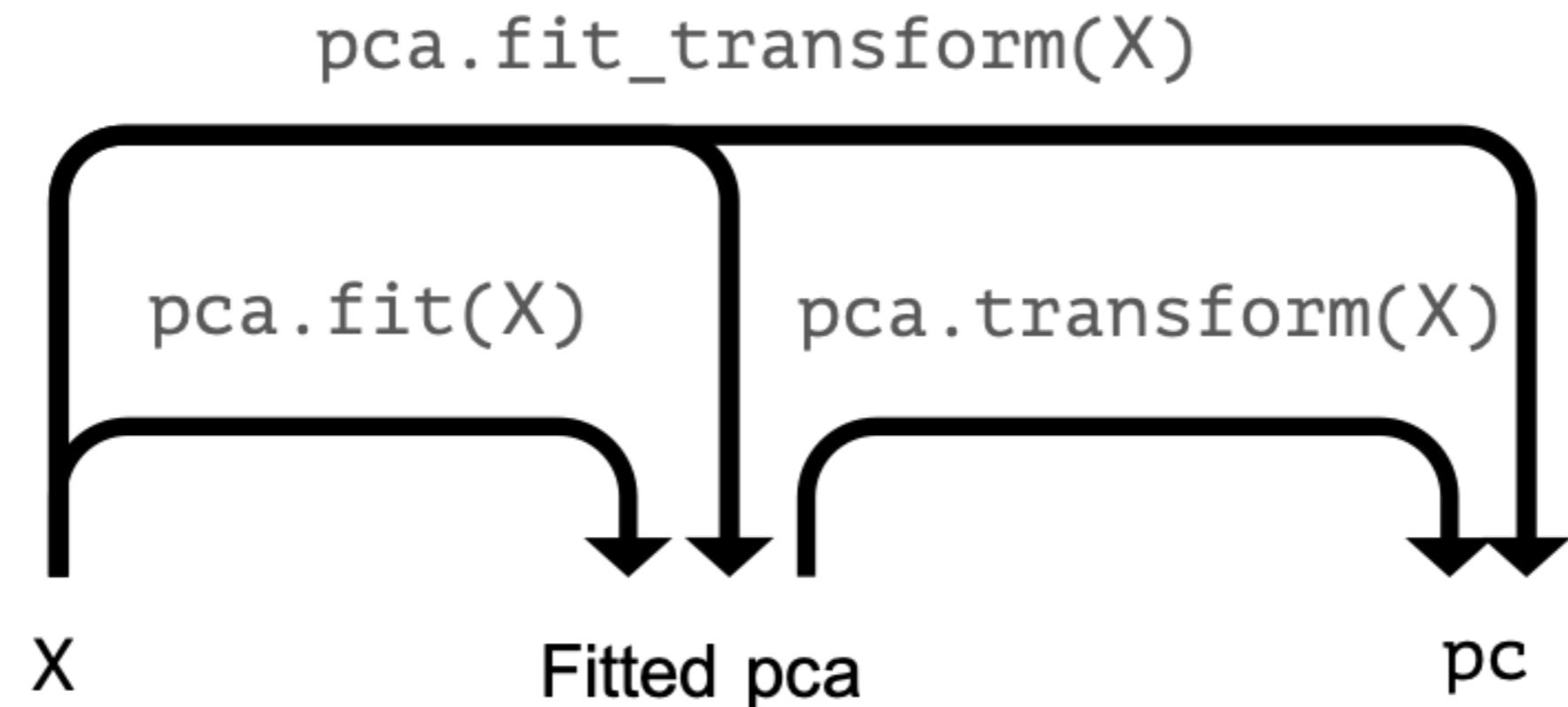
plt.xlabel('Principal component index')
plt.ylabel('Explained variance ratio')
plt.show()
```



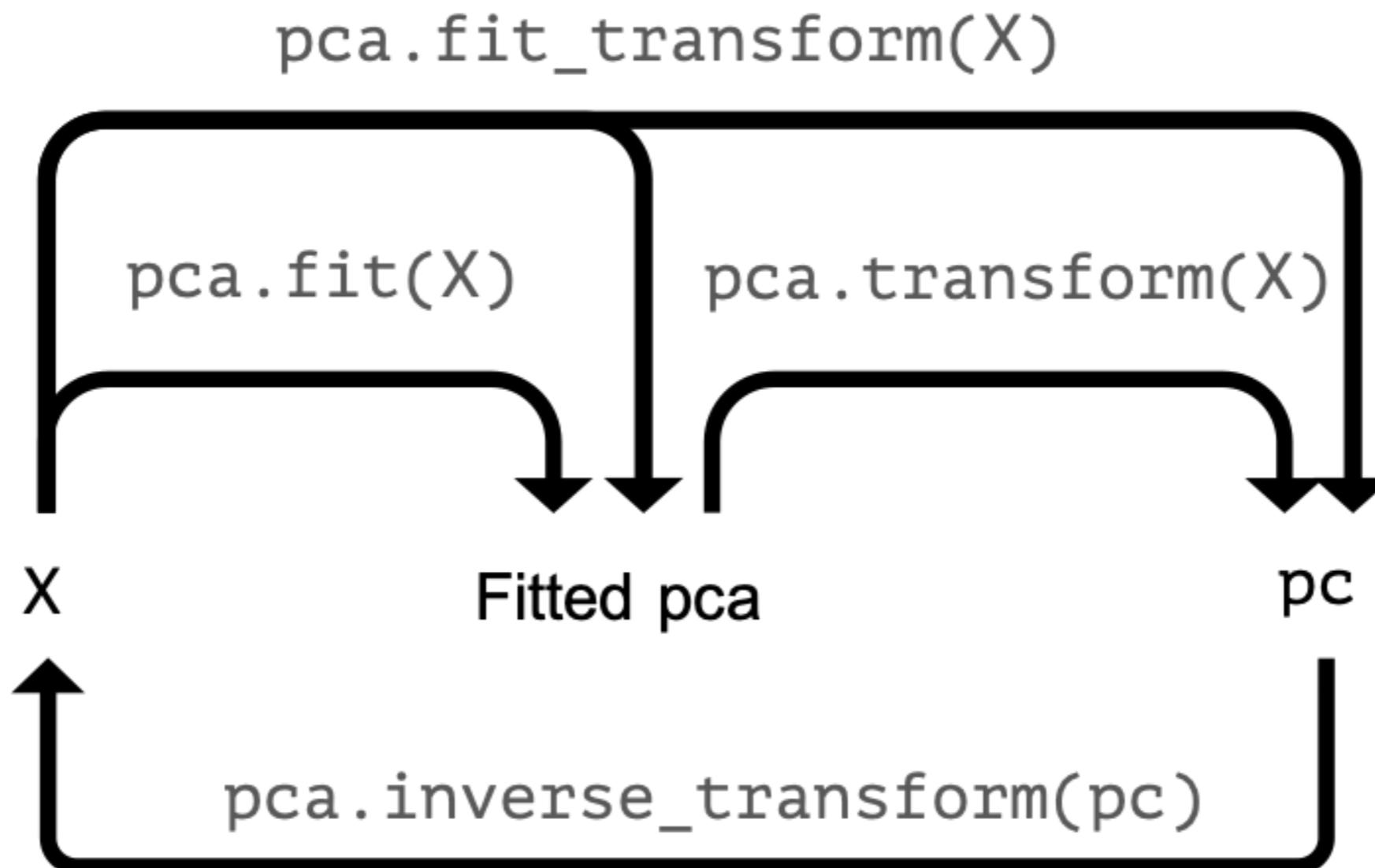
# PCA operations



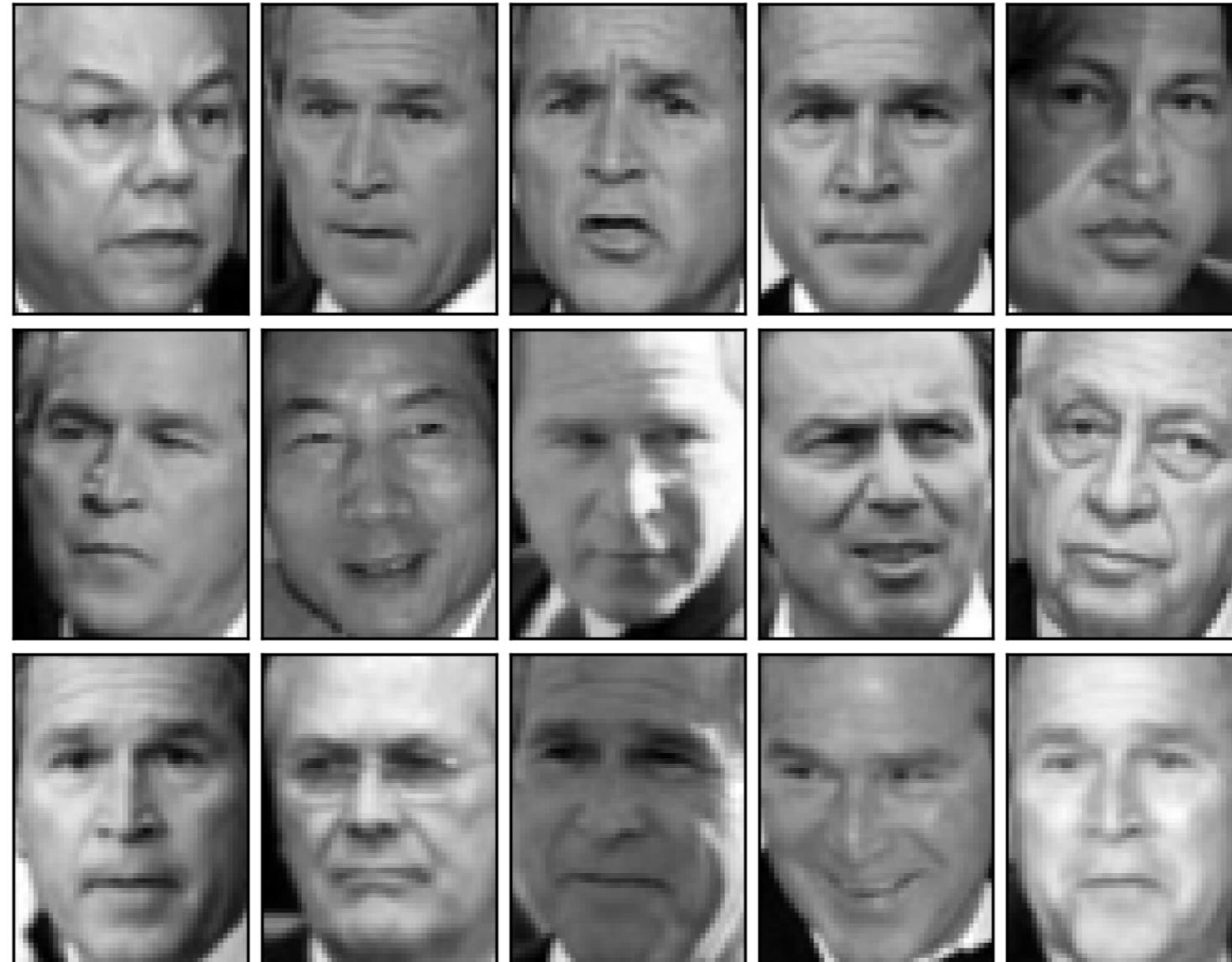
# PCA operations



# PCA operations



# Compressing images



# Compressing images

```
print(X_test.shape)
```

```
(15, 2914)
```

62 x 47 pixels = 2914 grayscale values

```
print(X_train.shape)
```

```
(1333, 2914)
```

# Compressing images

```
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('reducer', PCA(n_components=290))])

pipe.fit(X_train)

pc = pipe.fit_transform(X_test)

print(pc.shape)
```

```
(15, 290)
```

# Rebuilding images

```
pc = pipe.transform(X_test)
```

```
print(pc.shape)
```

```
(15, 290)
```

```
X_rebuilt = pipe.inverse_transform(pc)
```

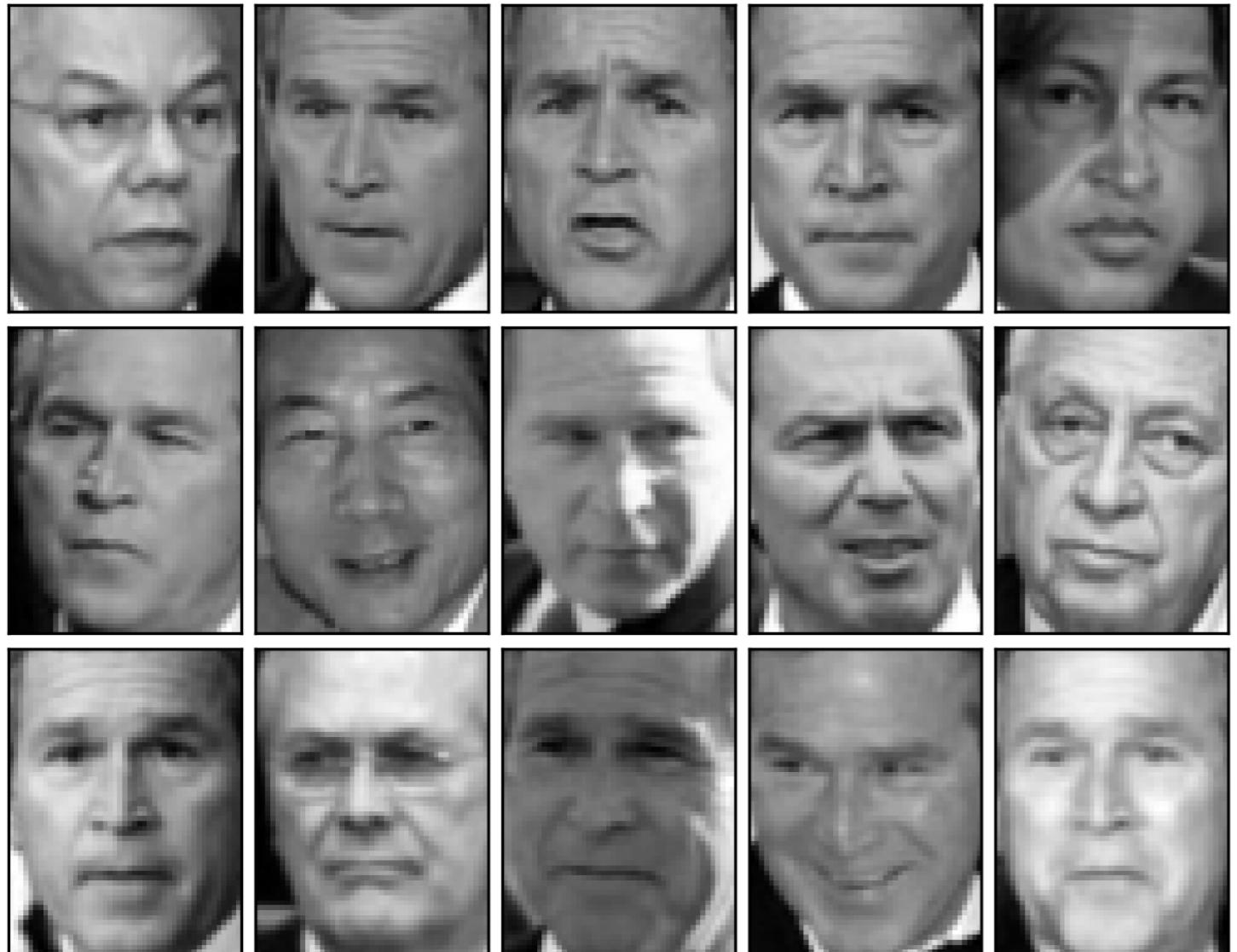
```
print(X_rebuilt.shape)
```

```
(15, 2914)
```

```
img_plotter(X_rebuilt)
```



# Rebuilding images



# Let's practice!

## DIMENSIONALITY REDUCTION IN PYTHON

# Congratulations!

DIMENSIONALITY REDUCTION IN PYTHON



Jeroen  
Machine Learning Engineer,  
Faktion

# What you've learned

- Why dimensionality reduction is important & when to use it
- Feature selection vs extraction
- High dimensional data exploration with t-SNE & PCA
- Use models to find important features
- Remove unimportant ones

# Thank you!

DIMENSIONALITY REDUCTION IN PYTHON