AI SOC

SESSION 3: INTRO TO PCA, CLASSIFICATION AND CLUSTERING

INNOVATE

2021

CHALLENGE WINNERS!

1. JEREMY LO
2. CASEY SU
3. LUKELEEAI

OVERVIEW

01 | Theory

Support Vector Machine

PCA

02 | Application

Preprocessing

03 | Theory

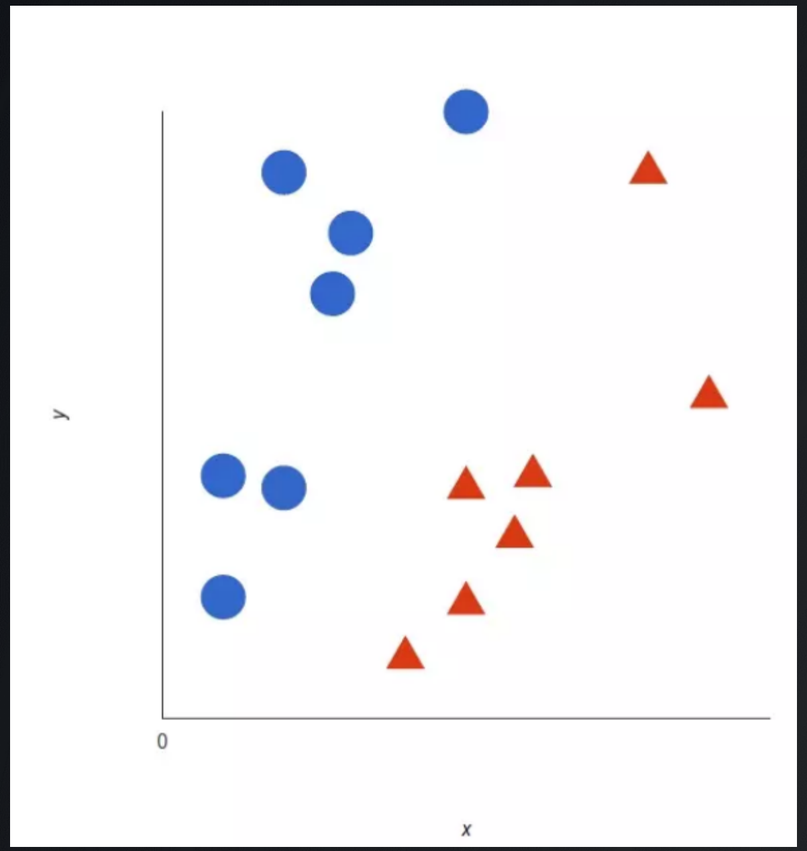
K-Means clustering

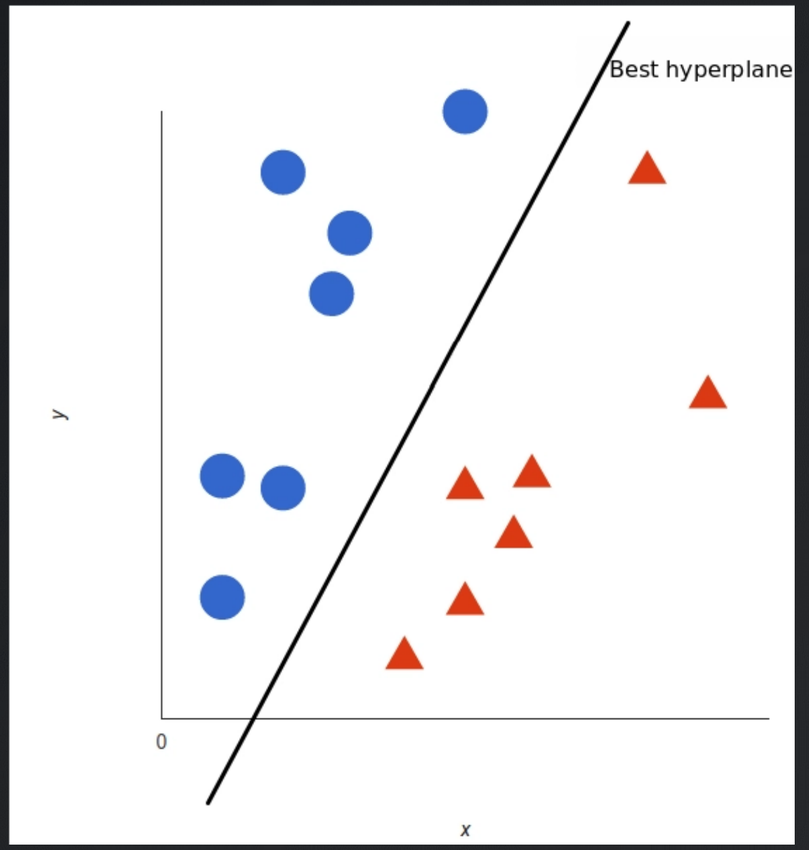
04 | Application

Building model

SVM

- Supervised model used for binary classification problems





Hypothesis

Function we are trying to fit

Cost function

Function that penalizes incorrect predictions

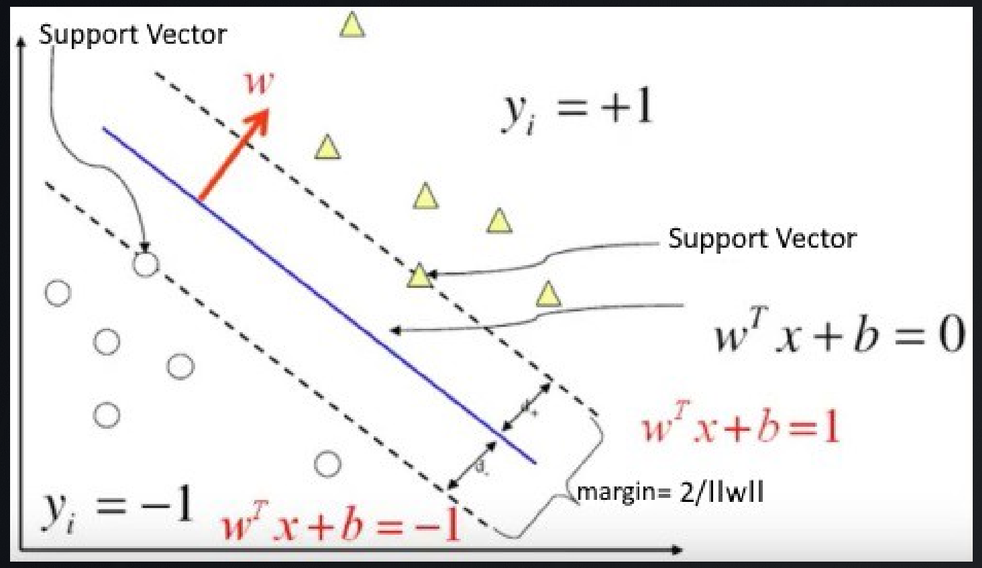
Optimization

Finding coefficient values at which cost is minimal

SVM

HYPOTHESIS

Hyperplane



Mathematics

$$\hat{y} = -1 \text{ if } w^T * x + b < 0$$
$$\hat{y} = 1 \text{ if } w^T * x + b \geqslant 0$$

COST FUNCTION

Distances

Classification

$$J(w) = \frac{[w]^2}{2} + C[\frac{1}{N}\sum_i^n max(0, 1 - y_i(wx_1 + b))]$$

```python
def compute_cost(W, X, Y):
    # calculate hinge loss
    N = X.shape[0]
    distances = 1 - Y * (np.dot(X, W))
    distances[distances < 0] = 0  # equivalent to max(0, distance)
    hinge_loss = reg_strength * (np.sum(distances) / N)

    # calculate cost
    cost = 1 / 2 * np.dot(W, W) + hinge_loss
    return cost
```

OPTIMIZATION

- We need to find optimal coefficients with the smallest cost

$$\frac{1}{N}\sum_i^n w \text{ if } max(0, 1 - y_i * (wx_i)) = 0$$
$$\frac{1}{N}\sum_i^n w - Cy_i x_i \text{ otherwise}$$

```python
def calculate_cost_gradient(W, X_batch, Y_batch):
    # if only one example is passed (eg. in case of SGD)
    if type(Y_batch) == np.float64:
        Y_batch = np.array([Y_batch])
        X_batch = np.array([X_batch])
    distance = 1 - (Y_batch * np.dot(X_batch, W))
    dw = np.zeros(len(W))
    for ind, d in enumerate(distance):
        if max(0, d) == 0:
            di = W
        else:
            di = W - (reg_strength * Y_batch[ind] * X_batch[ind])
        dw += di
    dw = dw/len(Y_batch)  # average
    return dw
```

SKLEARN IMPLEMENTATION

```
from sklearn import svm

clf = svm.SVC(kernel = 'linear')
clf.fit(X_train, y_train)
```

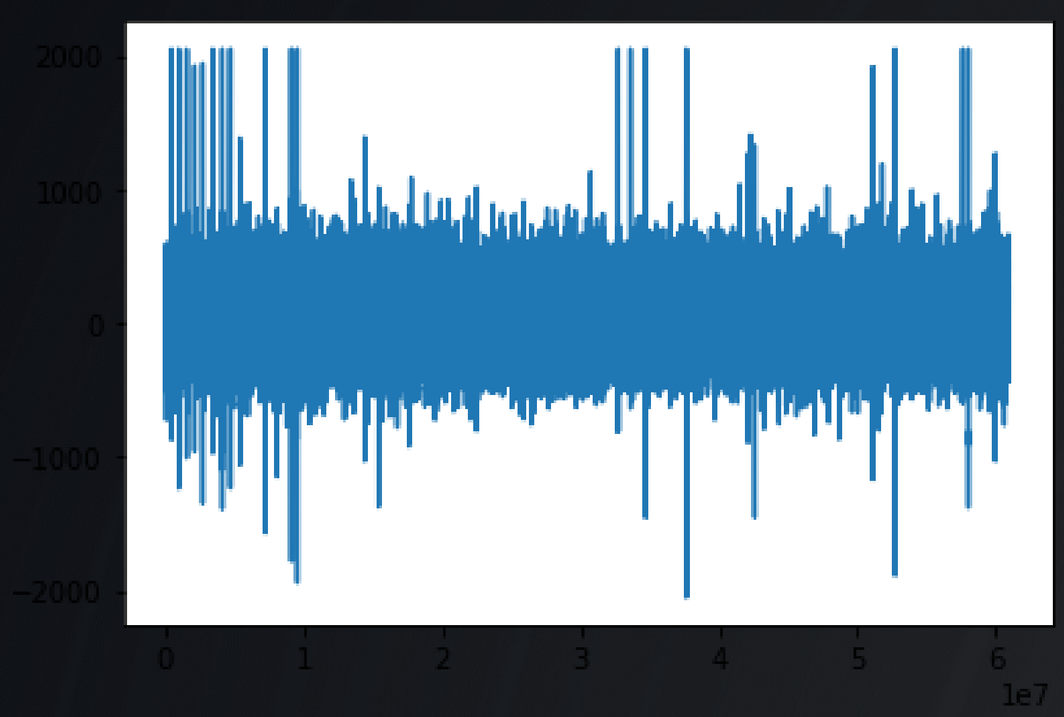
EXERCISE

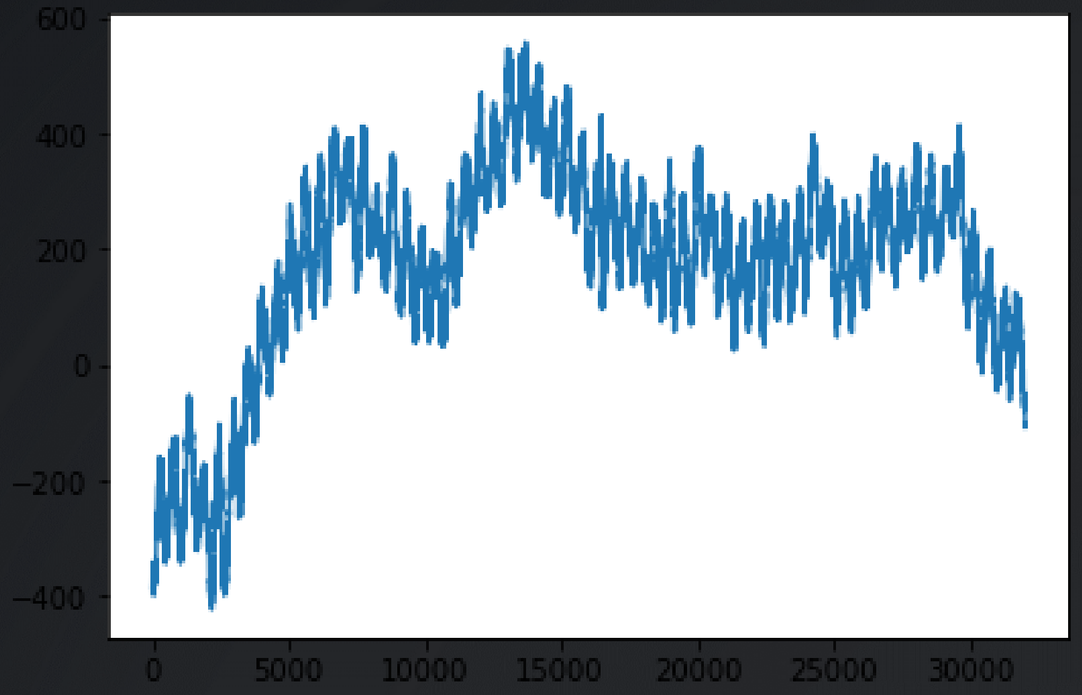
CHALLENGE

- *Neurons communicate with each other through electochemical signals*
- *It is quite hard to differentiate signals from individual neurons*

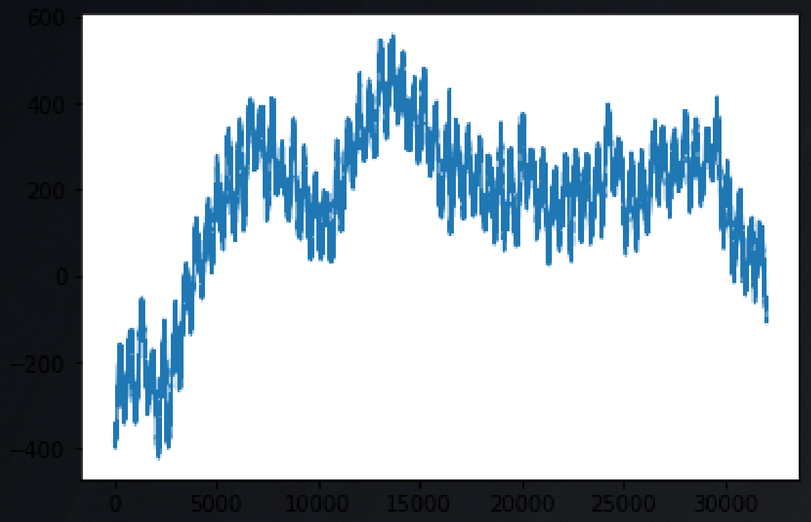
You are presented with neural activity data and your job is to build model that differentiates individual neural signals

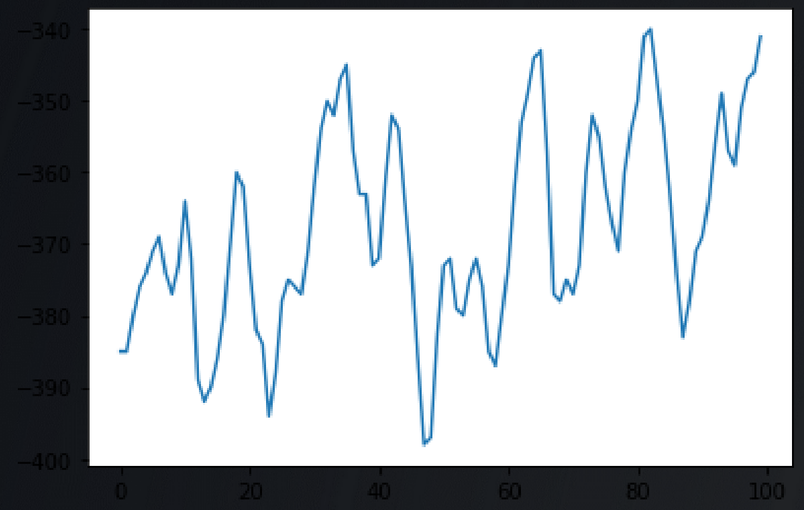
VISUALIZING DATA



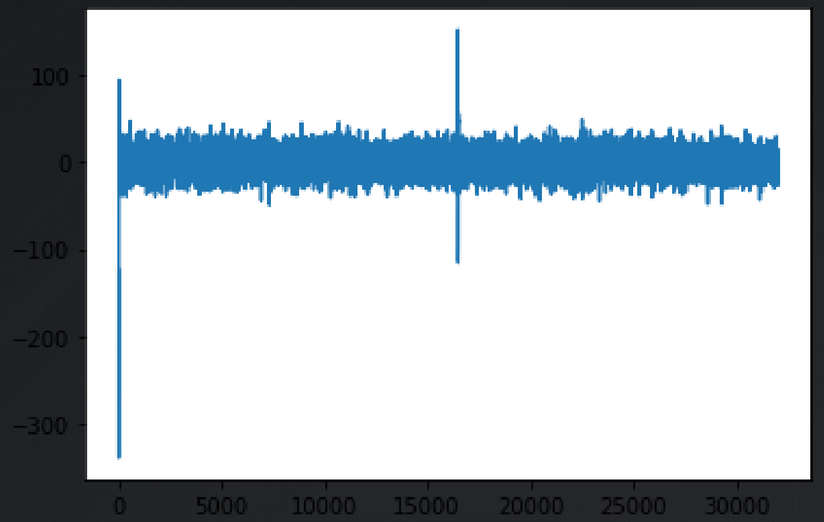


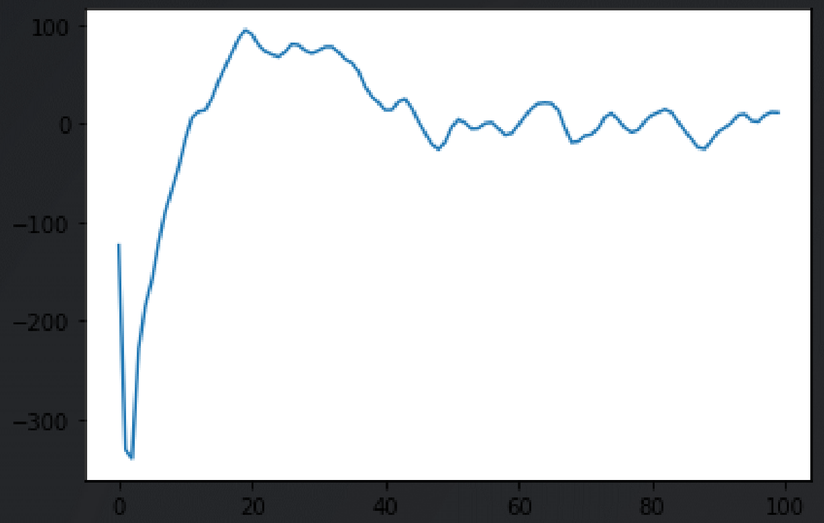
PREPROCESSING



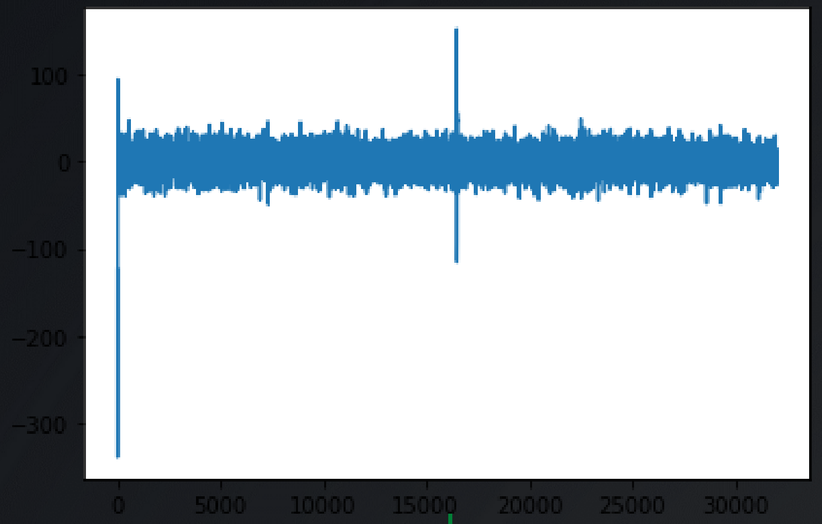


Filtering



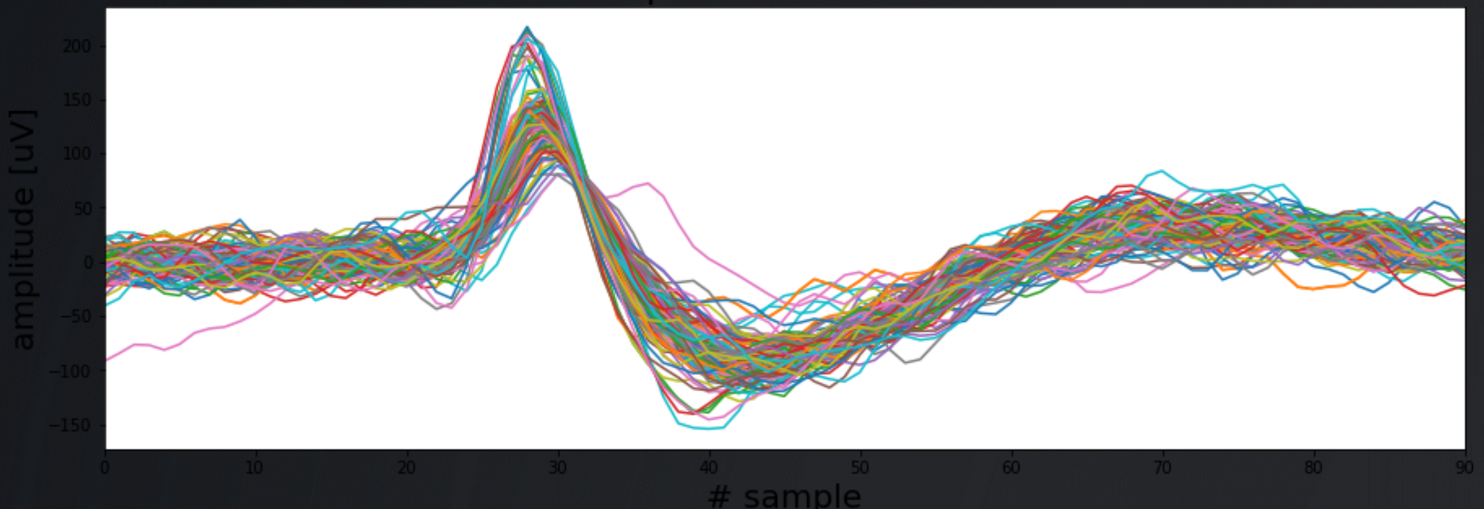


FEATURE SELECTION



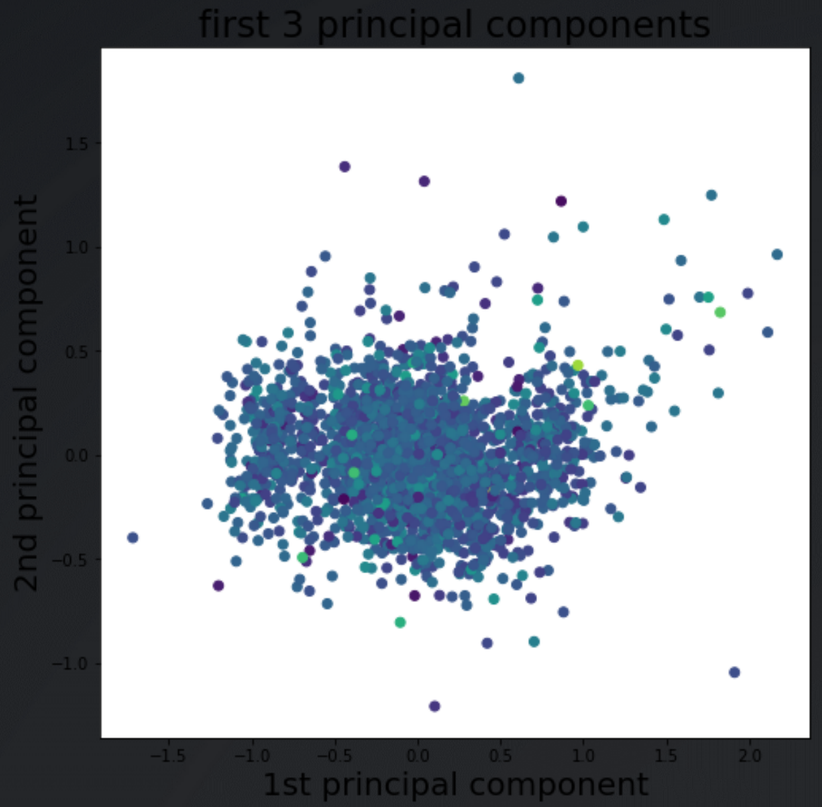
Threshold

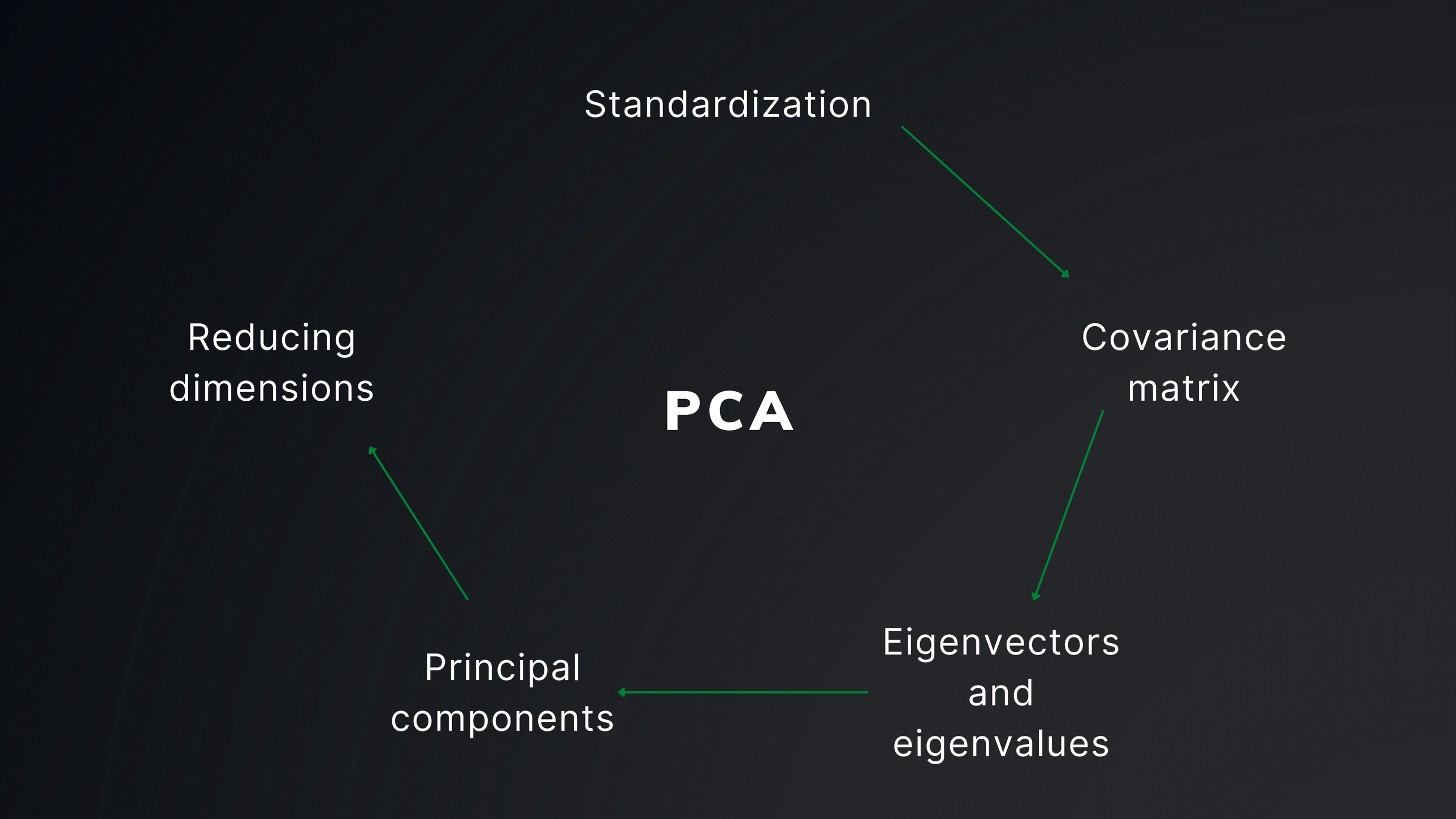
spike waveforms



PCA

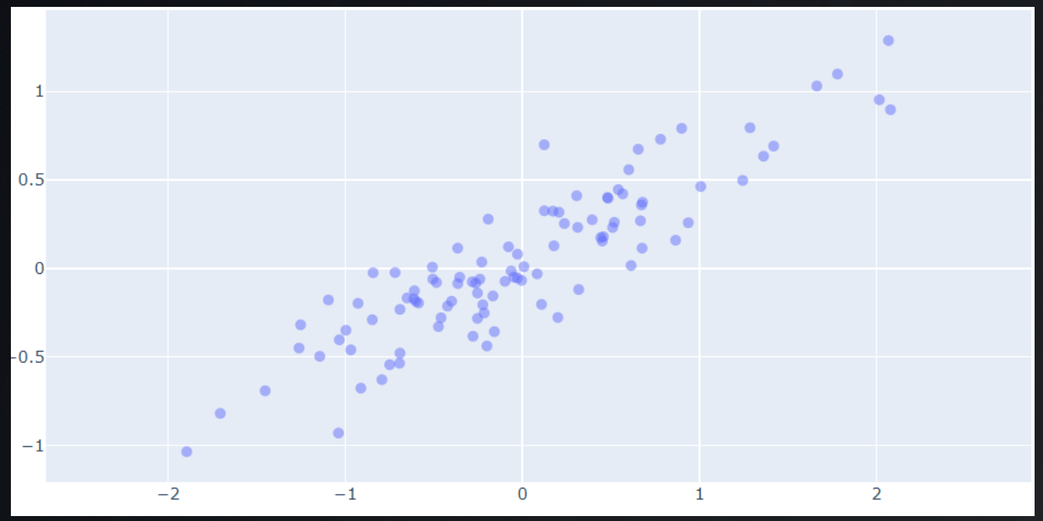
- Dimensional reduction technique
- Extracting only the most significant features




Standardization
Covariance matrix
Eigenvectors and eigenvalues
Principal components
Reducing dimensions
PCA

COVARIANCE

- Measuring level to which two variables vary



$$\begin{pmatrix} 2.5 & 3.0 \\ 3.0 & 10.0 \end{pmatrix}$$

$$\mathrm{Cov}(x, y) = \frac{1}{N-1} \sum_{j=1}^{N} (x_j - \mu_x)(y_j - \mu_j)$$

EIGENVALUES AND EIGENVECTORS

$$\mathbf{A}\vec{v}_i = \lambda_i \vec{v}_i$$

$$\mathbf{A} = \begin{bmatrix} -2 & 2 & 1 \\ -5 & 5 & 1 \\ -4 & 2 & 3 \end{bmatrix}$$

$$det(\mathbf{A} - \lambda \mathbf{I}) = \begin{vmatrix} -2-\lambda & 2 & 1 \\ -5 & 5-\lambda & 1 \\ -4 & 2 & 3-\lambda \end{vmatrix} = \mathbf{0}$$

$$\lambda = 3, 2, 1$$

$$\mathbf{A}\vec{v}_i = \lambda_i \vec{v}_i$$

$$\lambda_1 = 3: \quad \hat{v}_1 = (1/\sqrt{6}, 2/\sqrt{6}, 1/\sqrt{6})^T$$

$$\lambda_2 = 2: \quad \hat{v}_2 = (1/\sqrt{6}, 1/\sqrt{6}, 2/\sqrt{6})^T$$

$$\lambda_3 = 1: \quad \hat{v}_3 = (1, 1, 1)^T/\sqrt{3}$$

QUIZ

PCA IMPLEMENTATION

```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

def pca(X):
    #Scaling values
    X = StandardScaler().fit_transform(X)

    #Computing covariance matrix
    mean = np.mean(X, axis = 0)
    cov_mat = (X - mean).T.dot((X - mean)) / (X.shape[0]-1)

    #Calculating eigenvectors and eigenvalues
    cov_mat = np.cov(X.T)
    eig_vals, eig_vecs = np.linalg.eig(cov_mat)

    #Computing feature vectors
    eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

    return eig_pairs
```

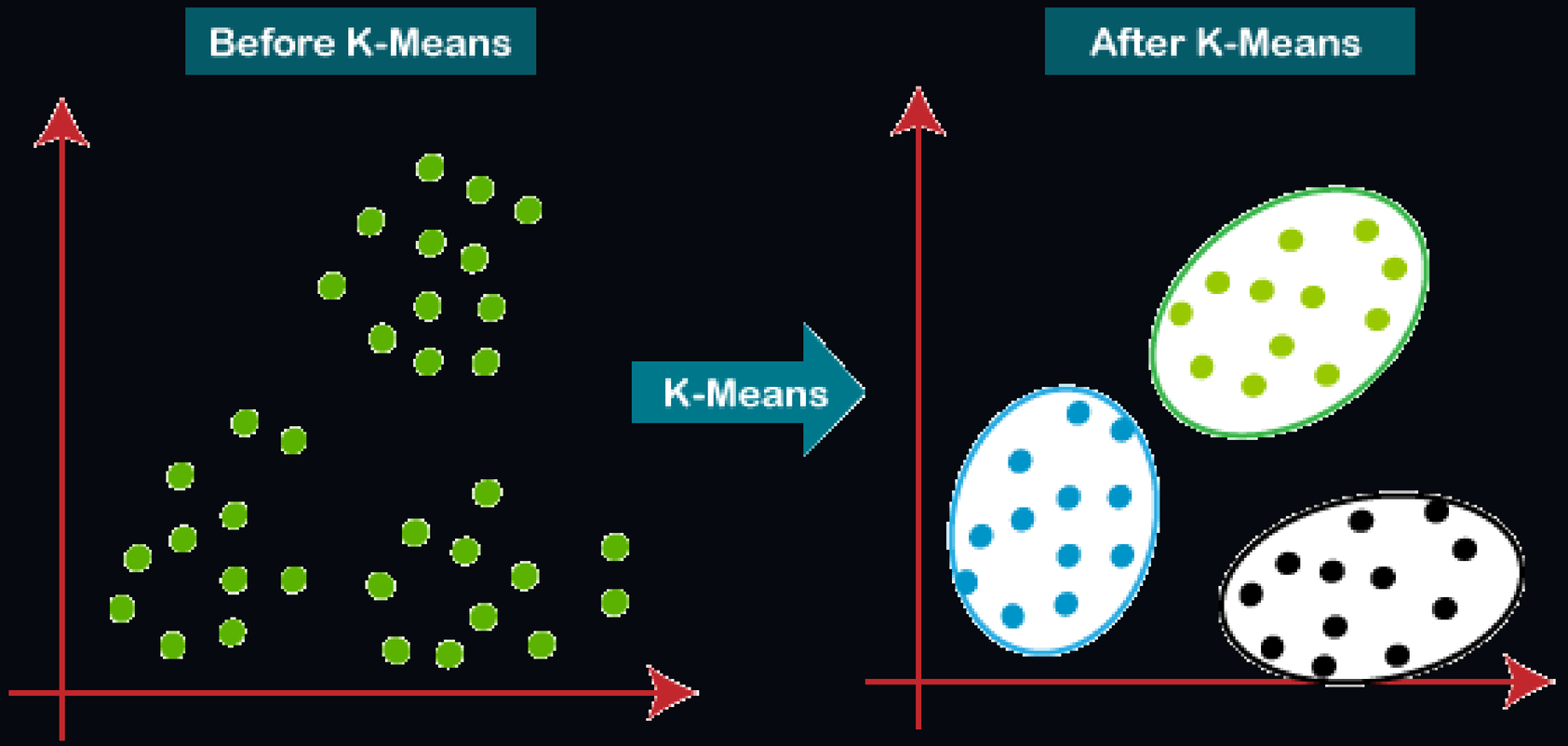
```python
def pca(X):

    pca = PCA(n_components=2)
    pca.fit_transform(X)
```

PCA FOR CHALLENGE

K-MEANS CLUSTERING

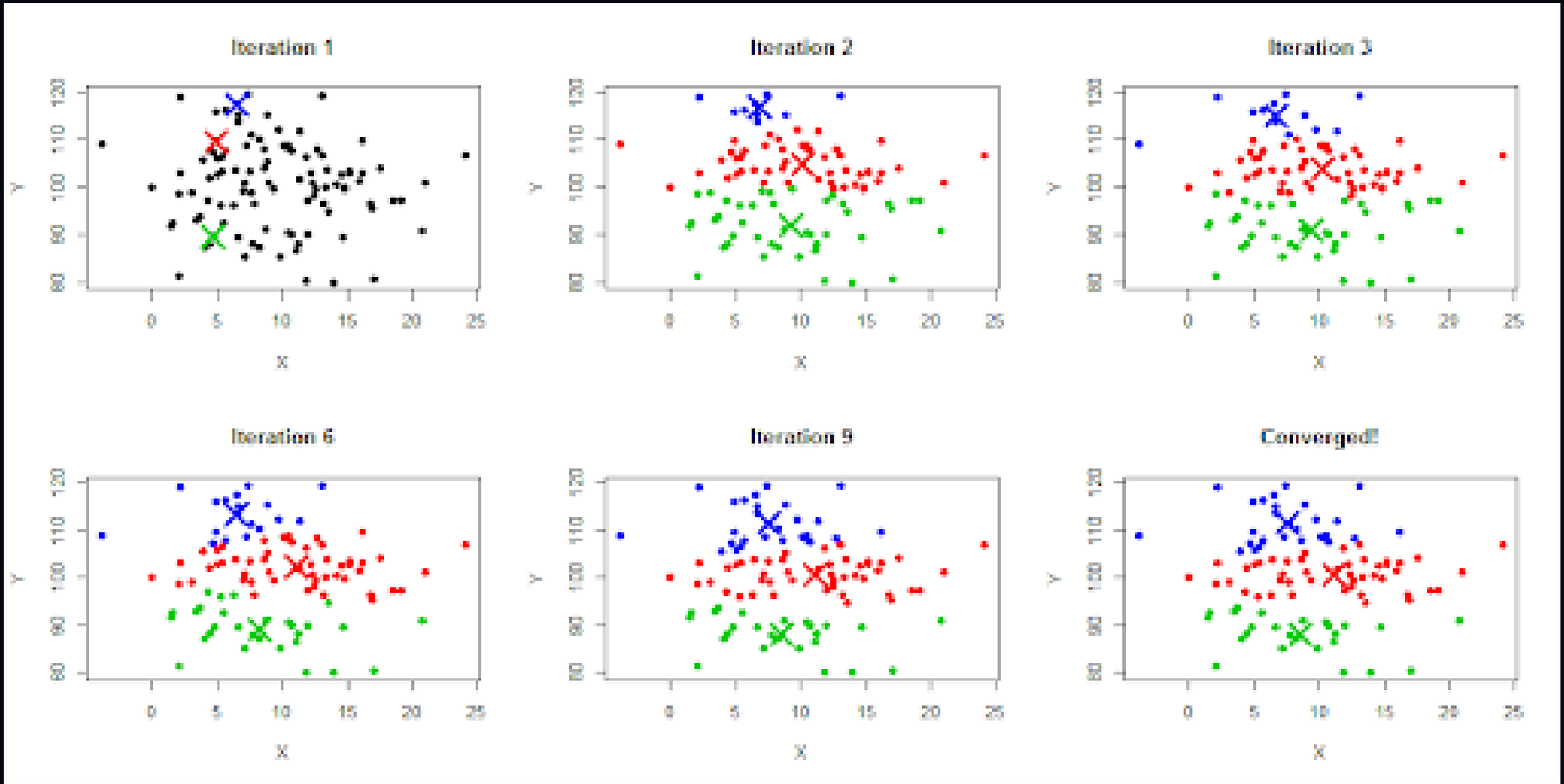
- Clustering
  - an unsupervised machine learning method of identifying and grouping similar data points

- K means
  - k centroids of clusters
  - update centroids and clusters using sum of squares


Before K-Means
K-Means
After K-Means

K-MEANS

- 1. randomly choose k centroids
- 2. compute the distance (e.g. Euclidian) from each points to the centroids
- 3. allocate each points to the nearest cluster
- 4. update the centroids to the mean points of the new clusters
- 5. stop until converged (none of the cluster assignments change)


Iteration 1
Iteration 2
Iteration 3
Iteration 6
Iteration 9
Converged!
X
Y
0
5
10
15
20
25
80
90
100
110
120

QUIZ

FINISHING CHALLENGE

QUESTIONS?

THANK YOU

SEE YOU NEXT WEEK!