GTK CYBER

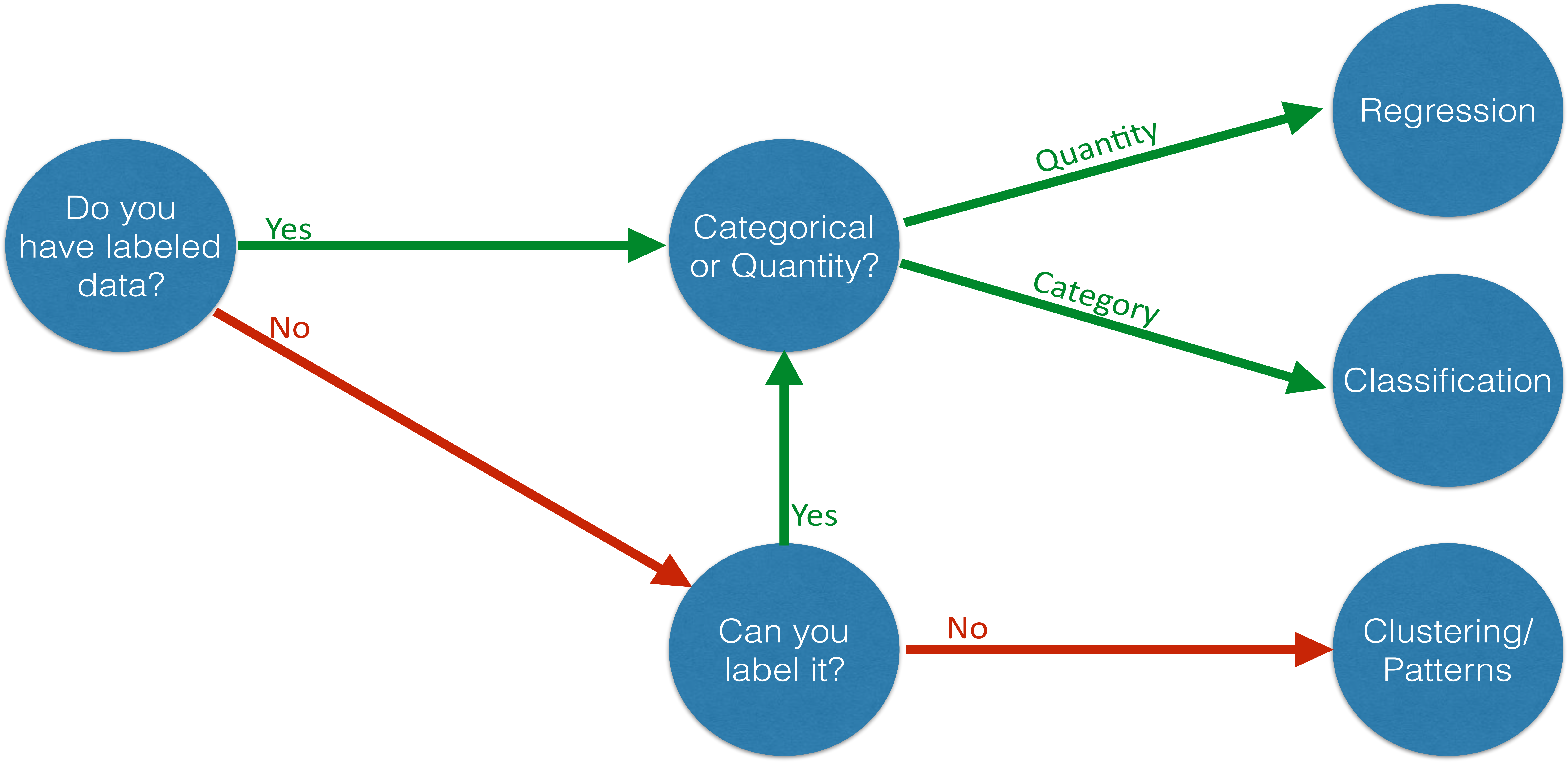
GET ON THE FAST TRACK

Machine Learning for Security Professionals - Day 3

Unsupervised Learning: Clustering

Agenda for Today

- Measuring Distances
- Math free overview of clustering techniques
- Pipelines and pickles



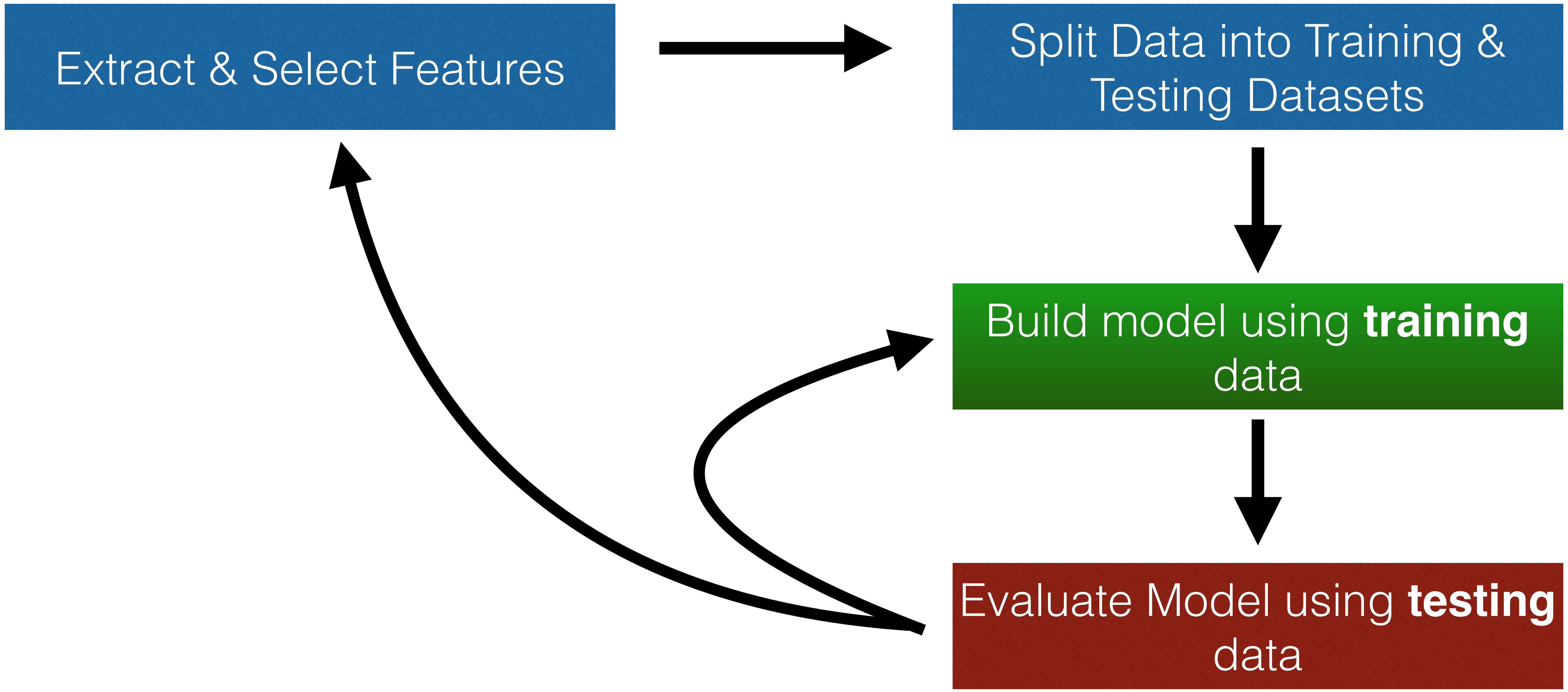
Prediction, Inference

Prediction

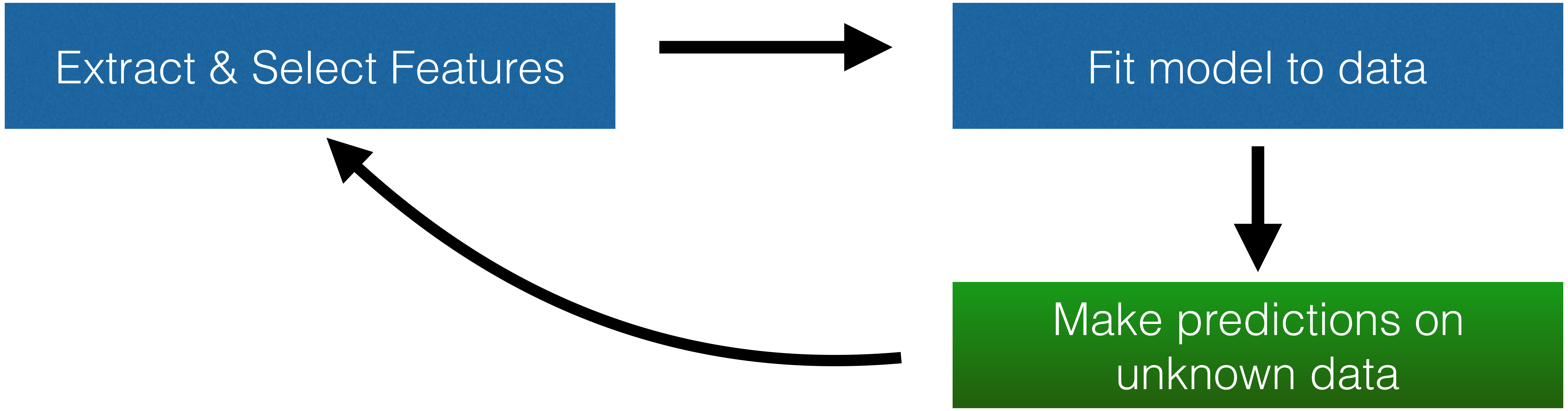
Nothing proven

Recommenders, Similar, anomalous

Supervised ML Process



Unsupervised ML Process



Unsupervised Clustering Algorithm

1. Select Features
2. Calculate a distance measure
3. Apply a clustering algorithm
4. Validate?

Which Departments are Similar?

	Malware events
Dept1	6
Dept2	1
Dept3	8

Which Departments are Similar?

	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1

Which Departments are Similar?

	Malware events	Phishing	Open Tickets
Dept1	6	6	3
Dept2	1	2	1
Dept3	8	1	9

Computing Distance

	Malware events
Dept1	6
Dept2	1
Dept3	8

Compare:

Dept1 to Dept2: | 6 - 8 | = 5

Dept2 to Dept3: | 1 - 8 | = 7

Dept1 to Dept3: | 6 - 8 | = 2

Two-Dimensional Distance

	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1

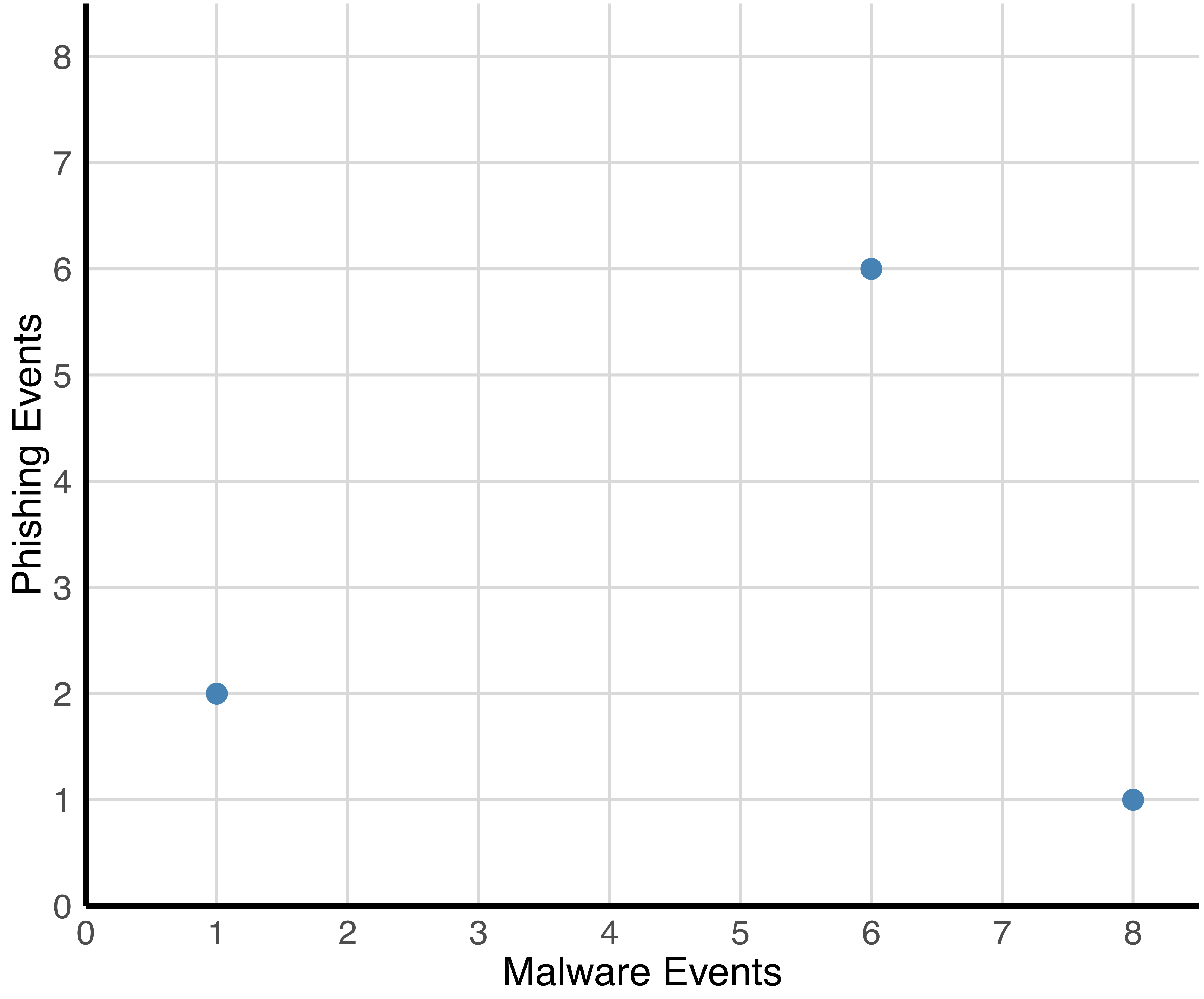
Multiple Distance methods

- Euclidean
- Manhattan
- Maximum
- Canberra
- Binary
- Minkowski

... (to name a few)

Two-Dimensional Distance

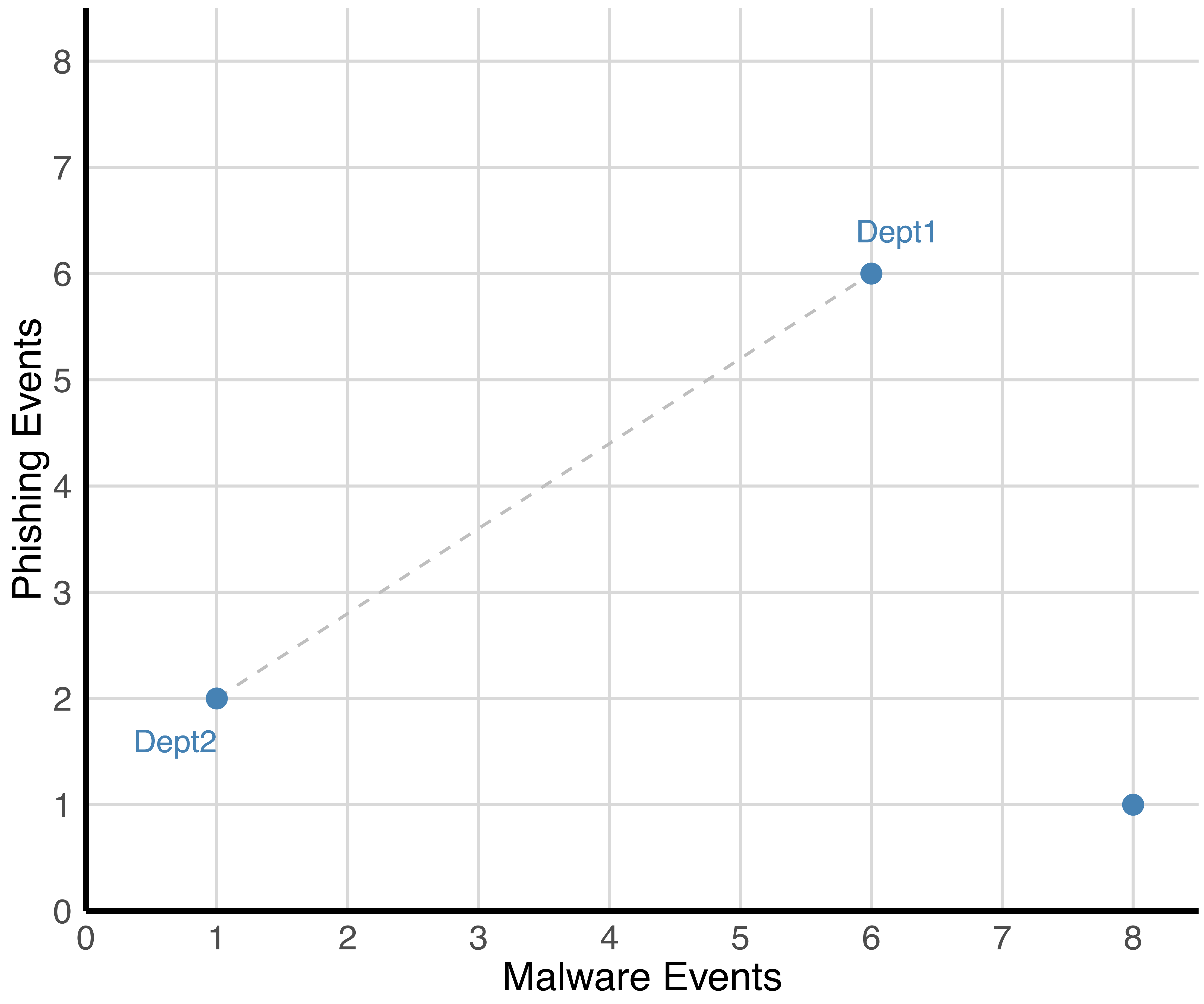
	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1



Two-Dimensional Distance

Euclidean very common and easy to understand

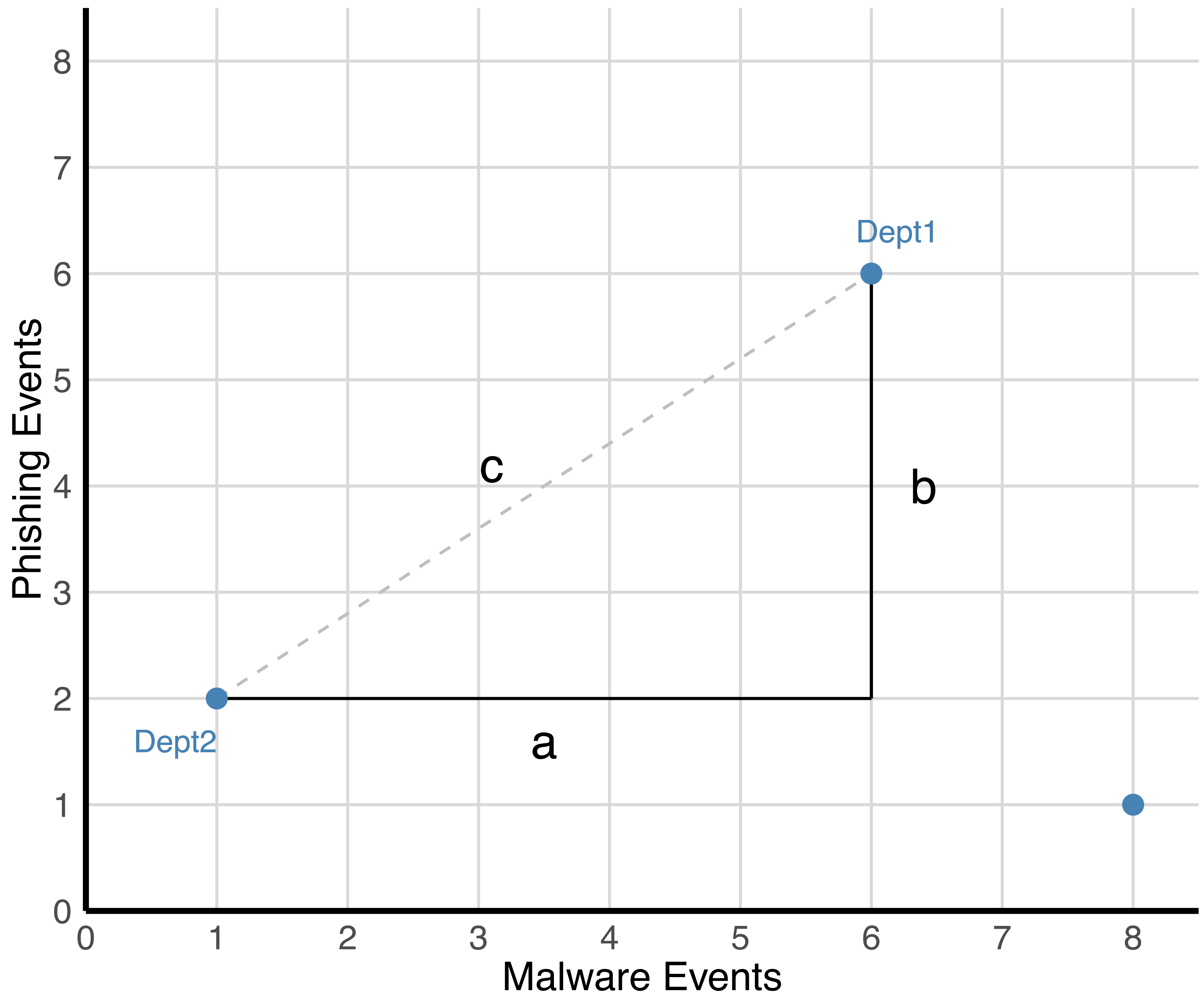
	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1



Two-Dimensional Distance

Euclidean very common and easy to understand: $a^2 + b^2 = c^2$

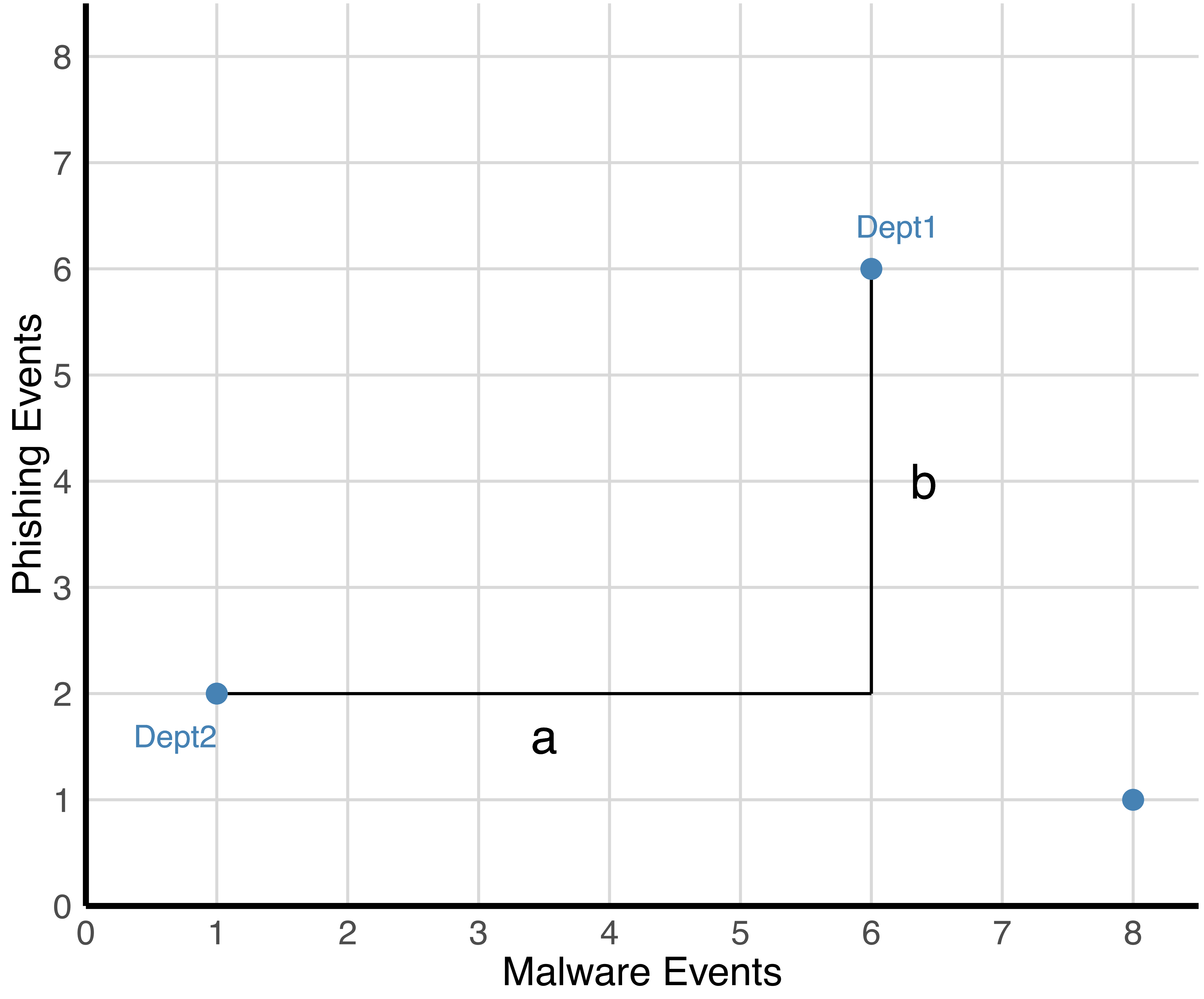
	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1



Two-Dimensional Distance

Manhattan also easy to comprehend: a + b

	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1



Computing Distance

	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1

Compare:

Dept1 to Dept2: sqrt((6-1)^2 + (6-2)^2) = **6.4**

Dept2 to Dept3: … = **7.1**

Dept1 to Dept3: … = **5.4**

Euclidean Distance calculations

```
def dist(x,y):
    return np.sqrt(np.sum((x-y)**2))

> mat = np.array([[ 6,6,3 ], [1,2,1], [8,1,9]])
> dist(mat[0], mat[1])
6.7082039324993694

> dist(mat[1], mat[2])
10.677078252031311

> dist(mat[0], mat[2])
8.0622577482985491
```

Which Departments are Similar?

	Malware events	Phishing	Open Tickets
Dept1	6	6	3
Dept2	1	2	1
Dept3	8	1	9

6.7

8.1

10.7

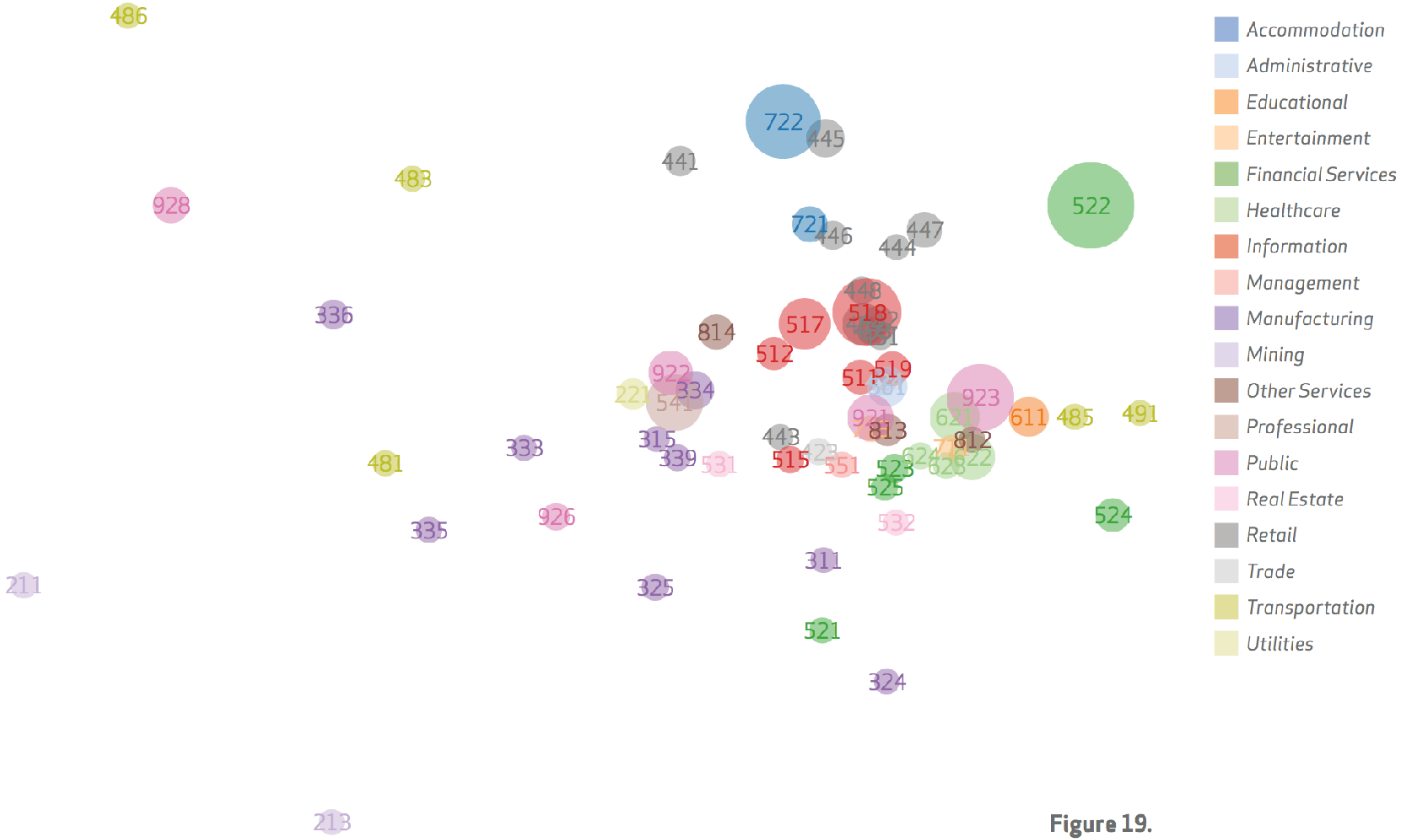
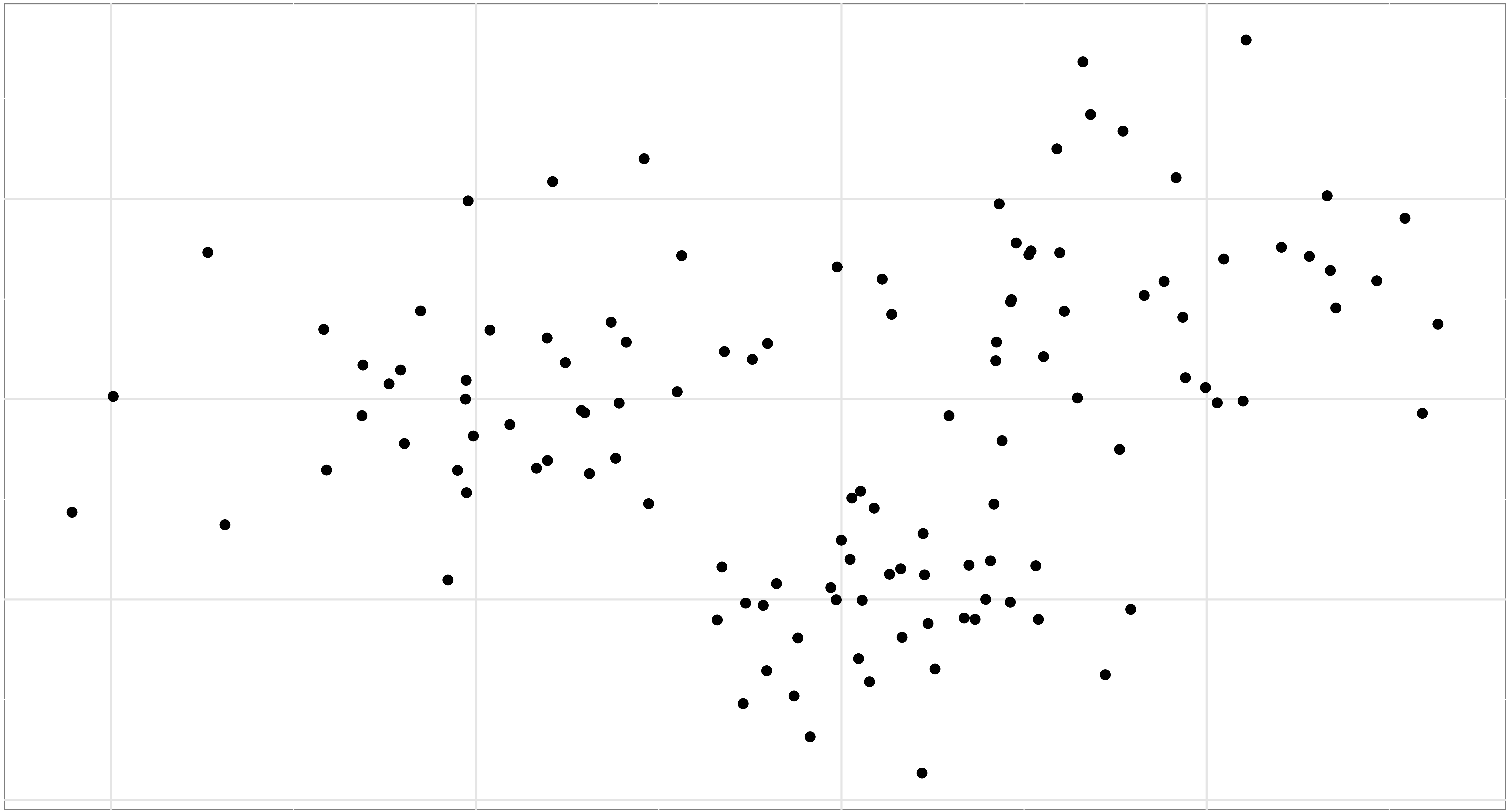


Figure 19.

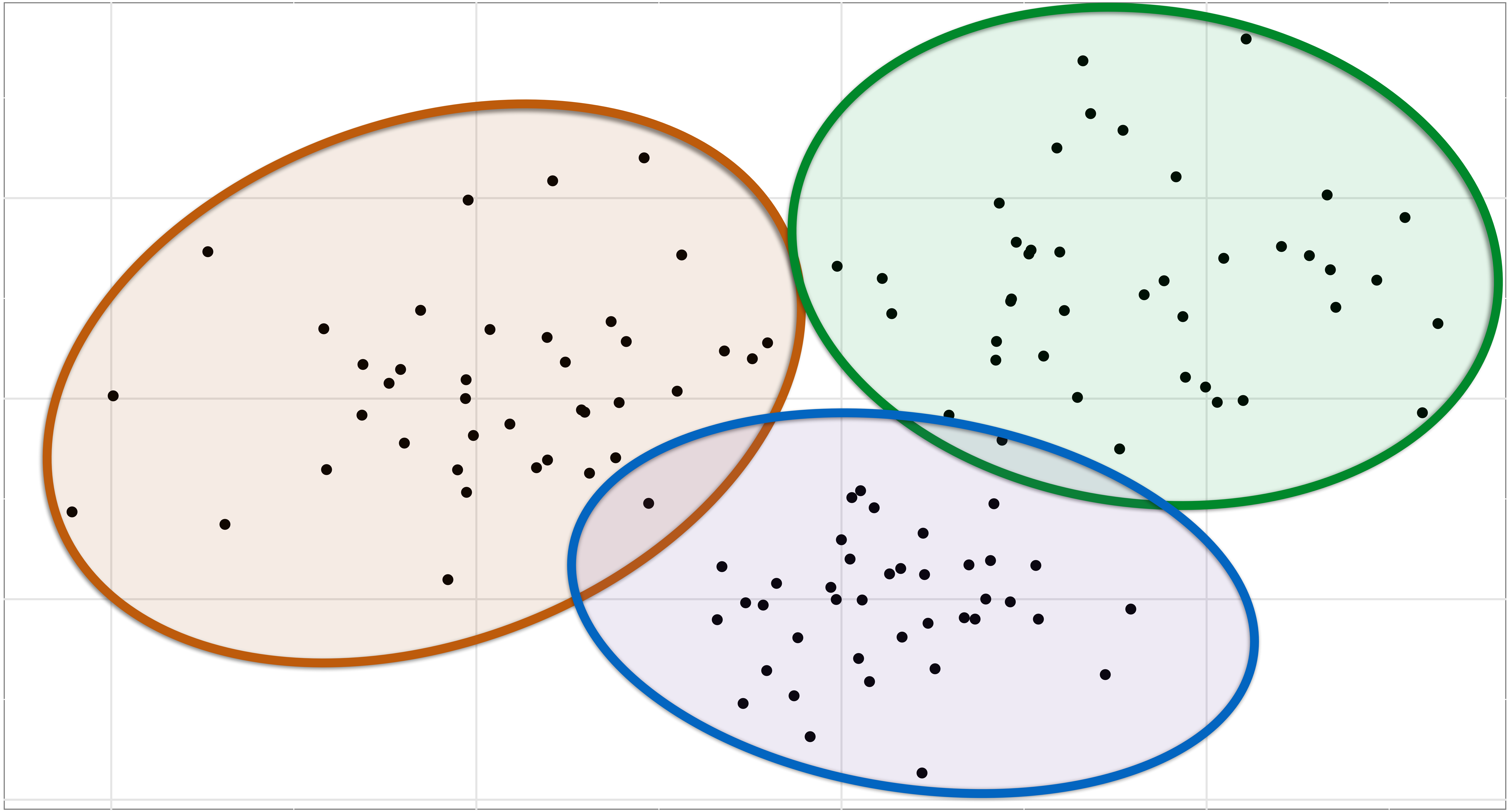
Clustering on breach data across industries

28 To look up the three-digit NAICS codes, visit: www.census.gov/eos/www/naics/index.html

Clustering…

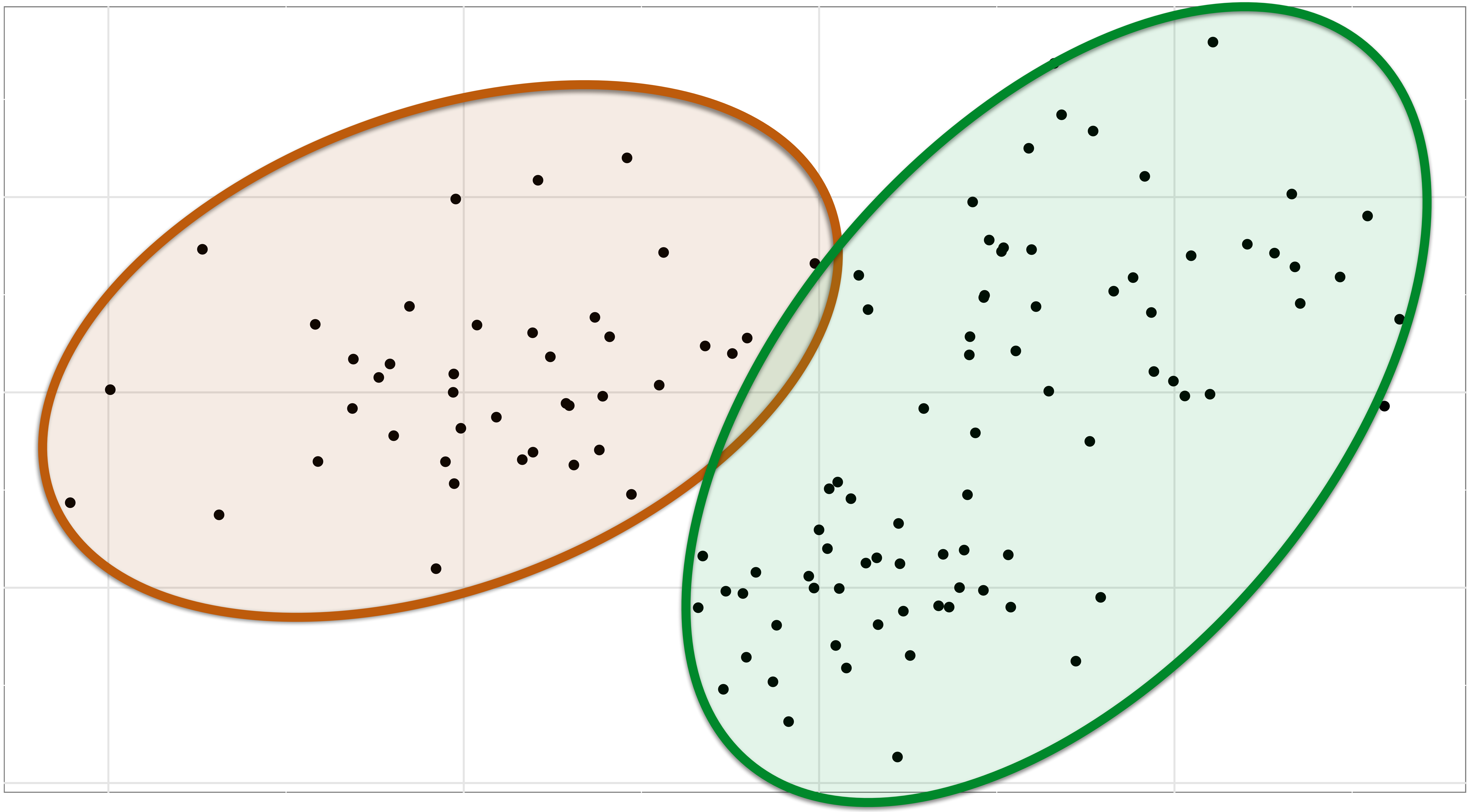


Clustering…



gtkcyber.com

Clustering…



gtkcyber.com

K-Means

Before starting, pick the number of clusters, K

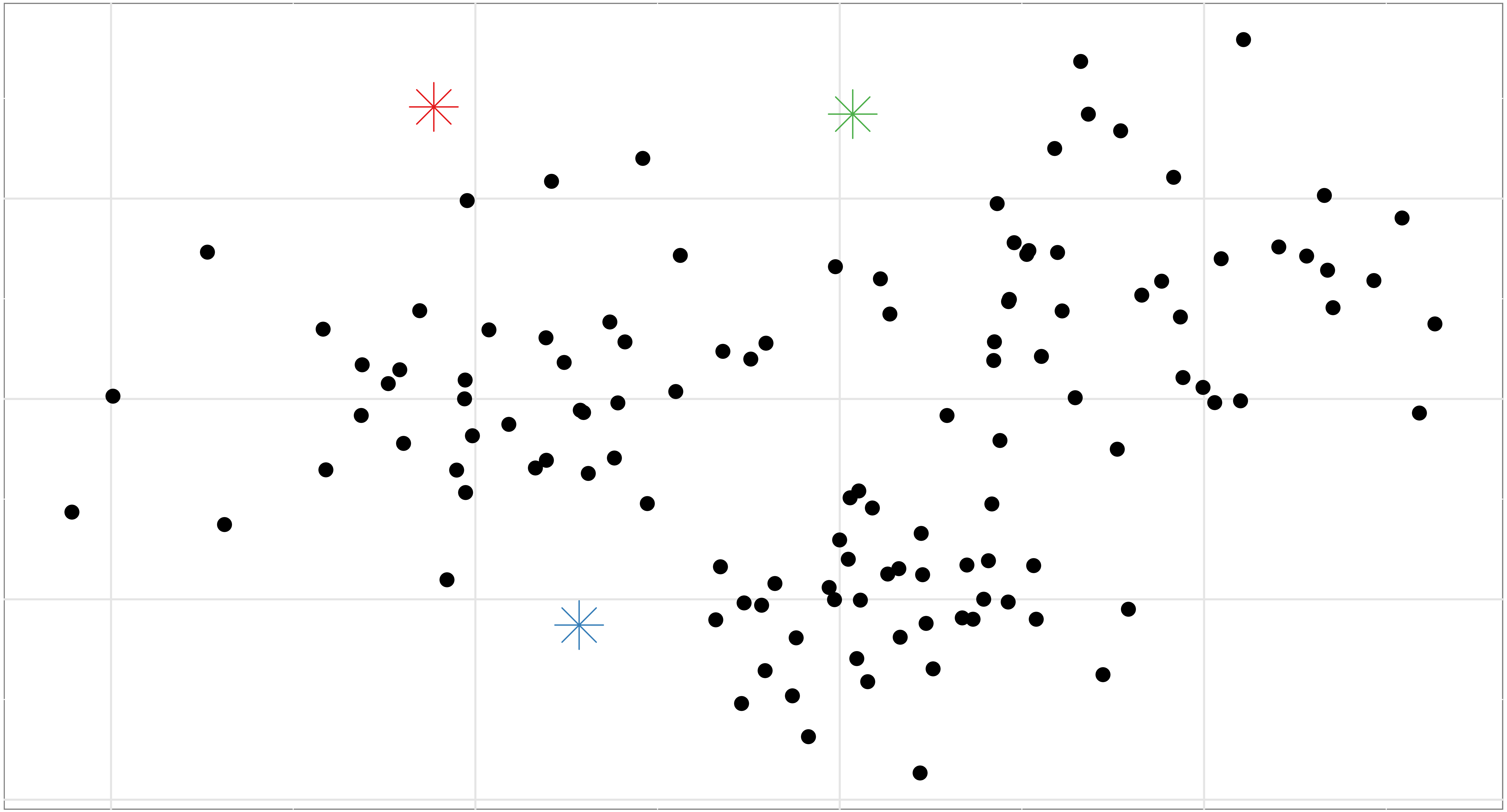
1. Pick K random centroids within data range

2. Assign each data point to the nearest centroid

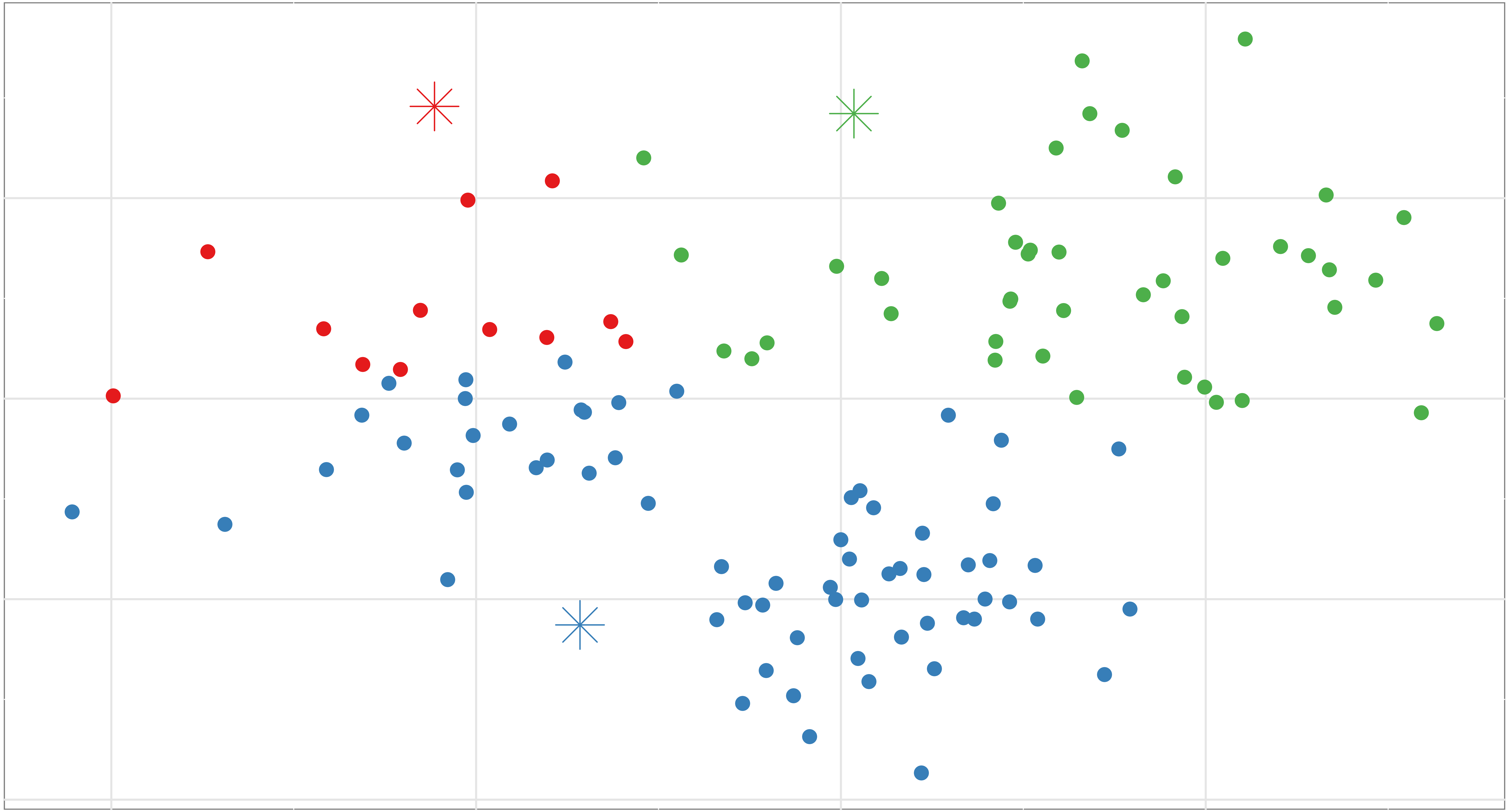
3. Move centroid to center of assigned points

4. Repeat steps 2 and 3 until centroid stops shifting

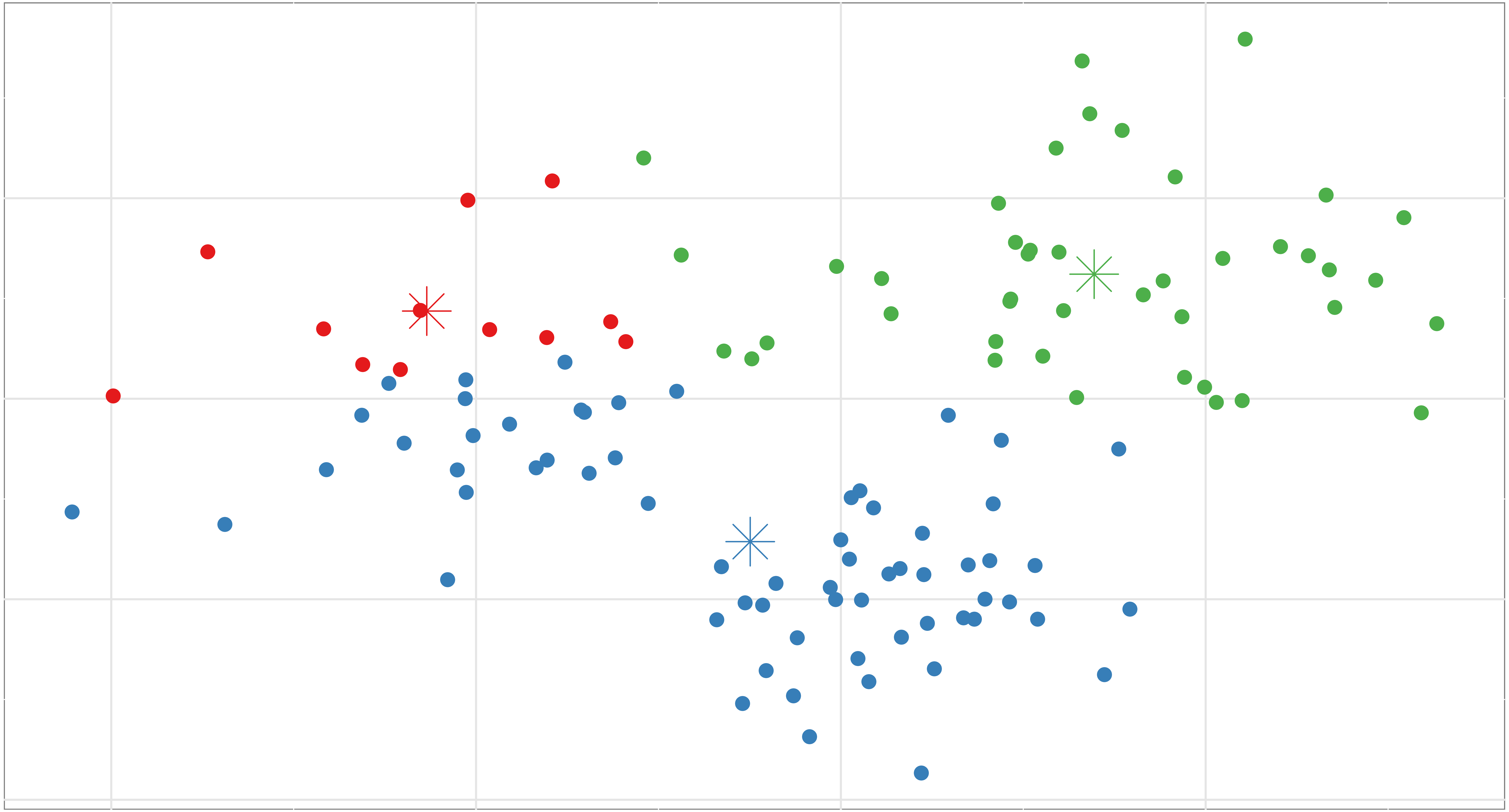
Step 1: Pick 3 random centroids within data range



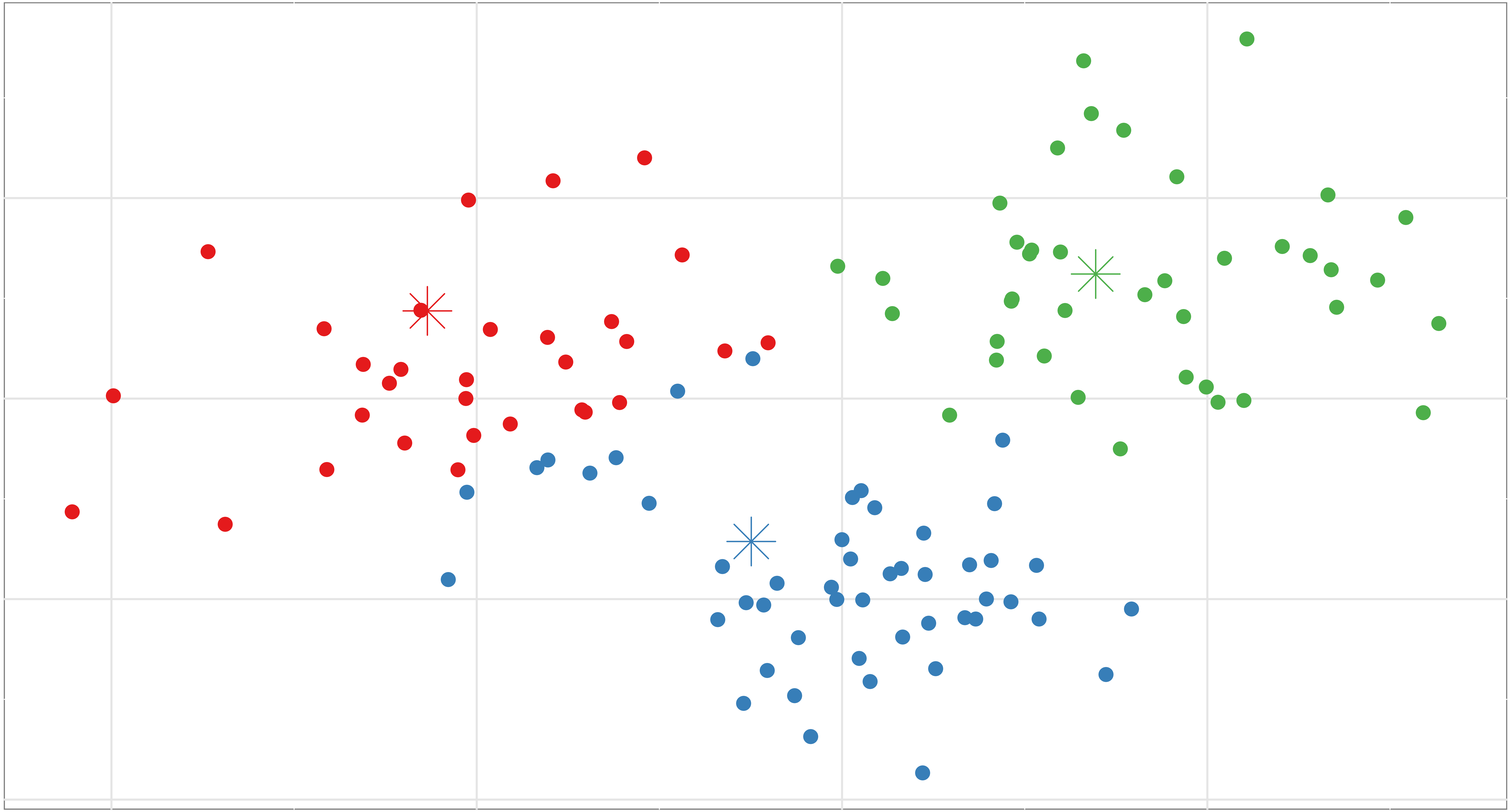
Step 2: Assign each data point to the nearest centroid (1)



Step 3: Move centroid to center of assigned points (1)

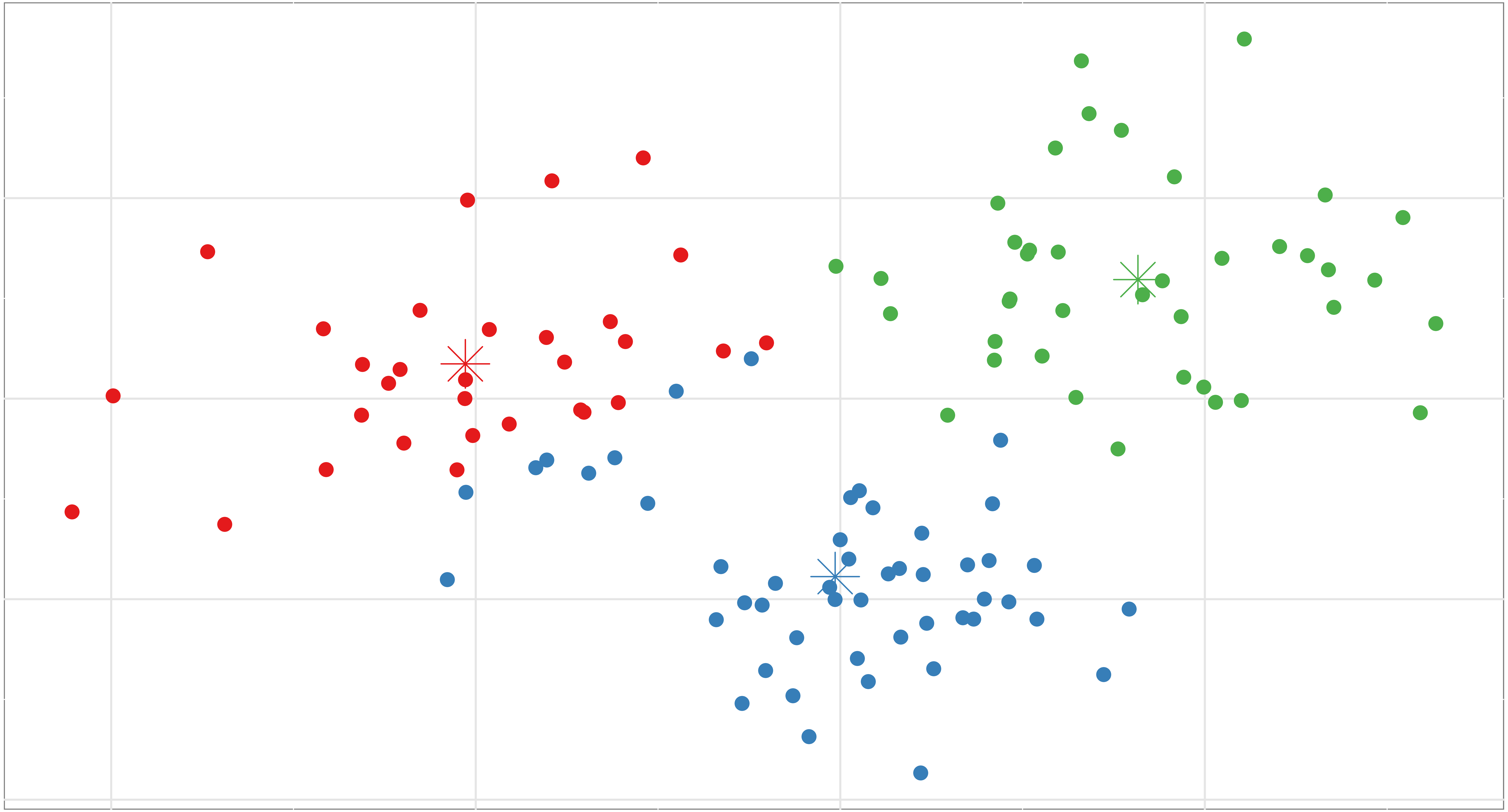


Step 2: Assign each data point to the nearest centroid (2)

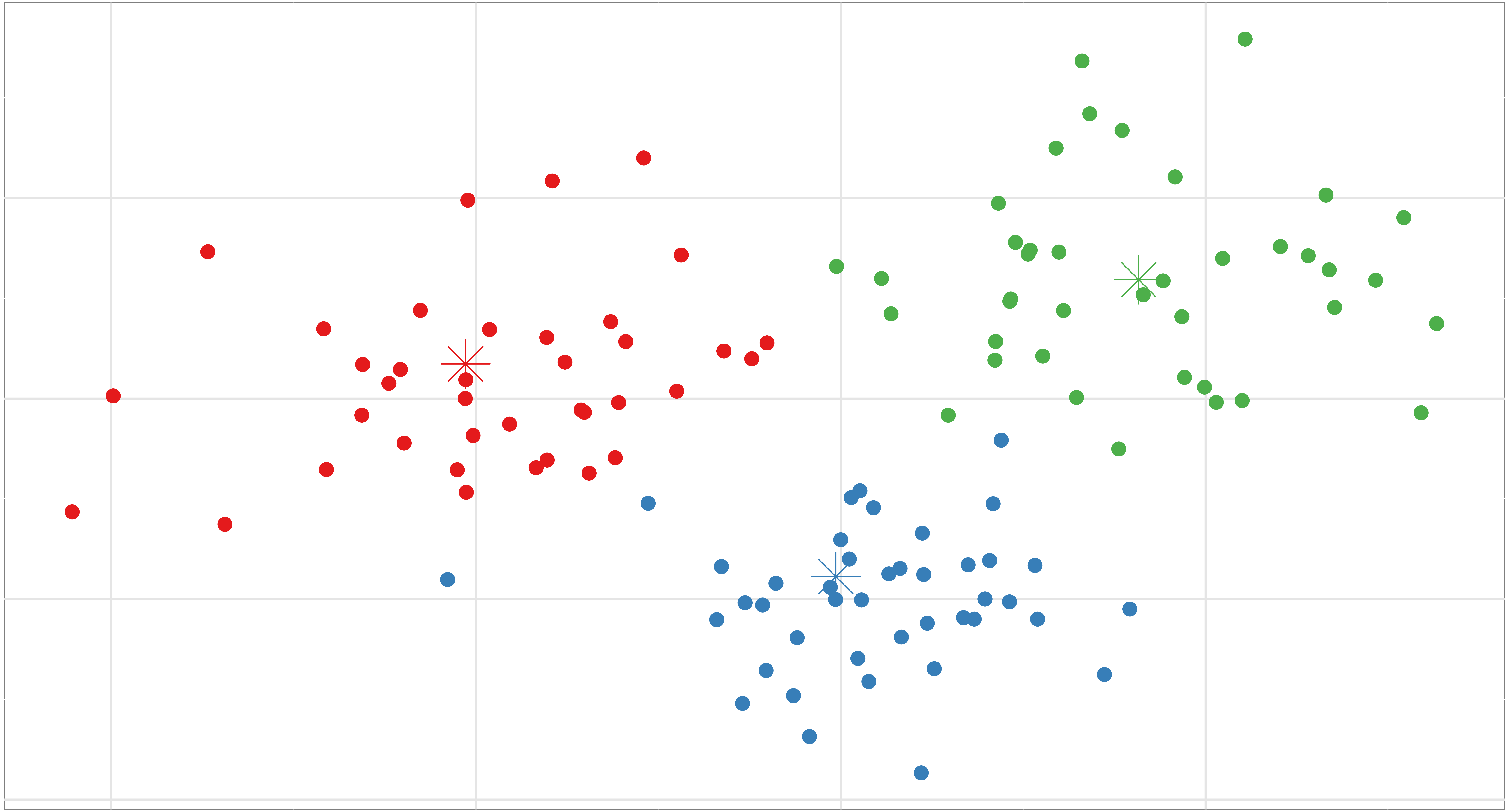


gtkcyber.com

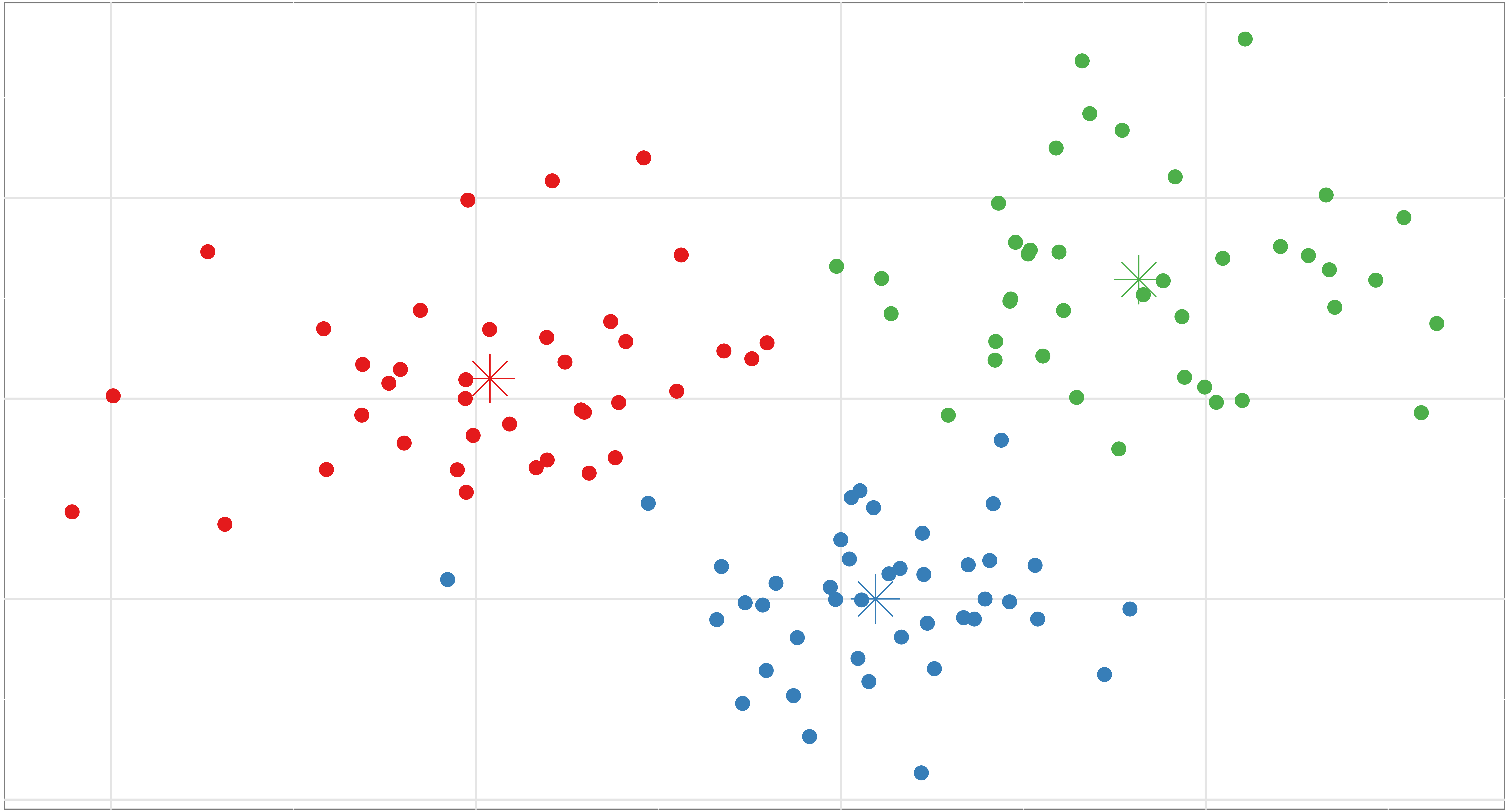
Step 3: Move centroid to center of assigned points (2)



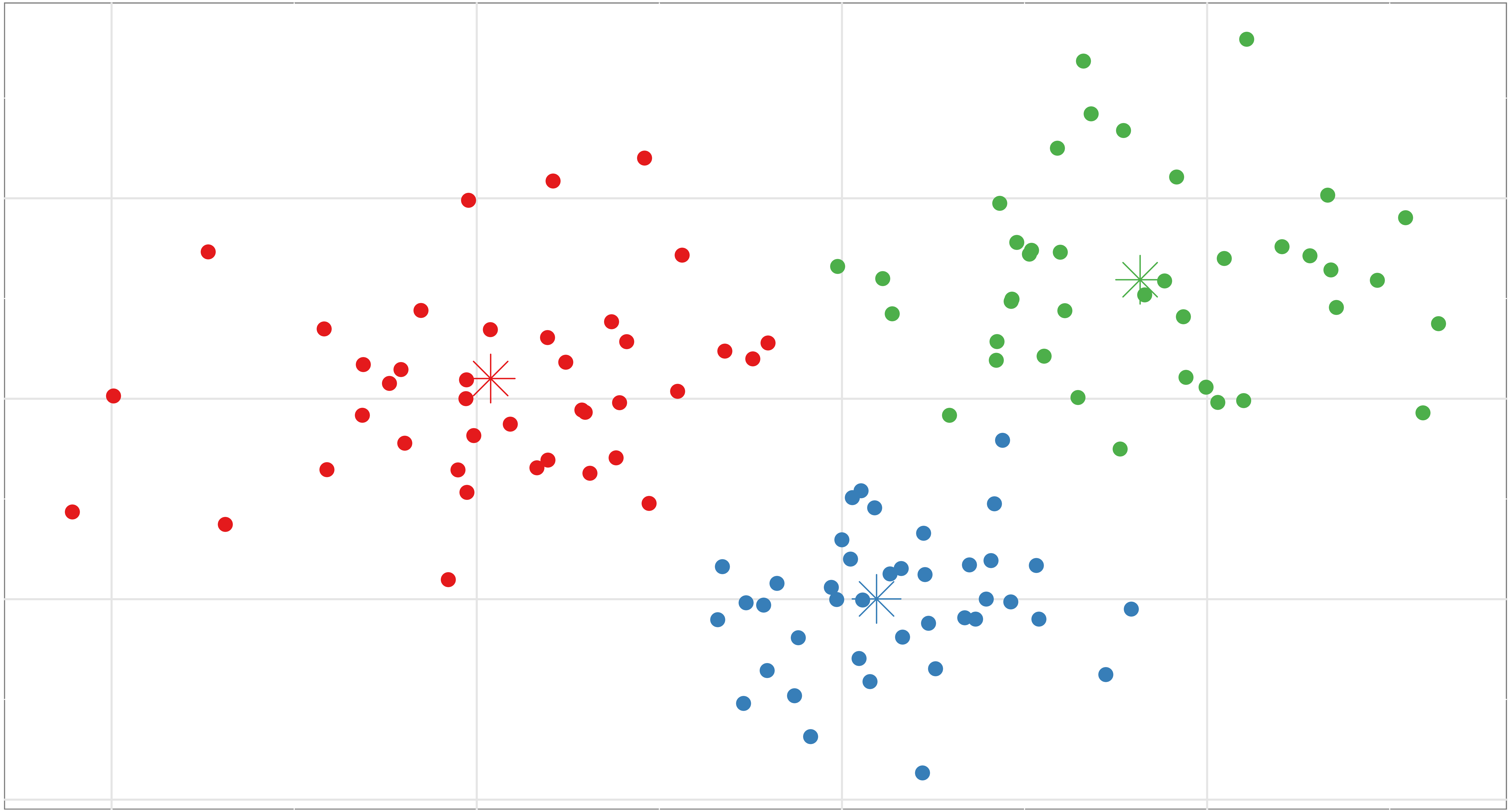
Step 2: Assign each data point to the nearest centroid (3)



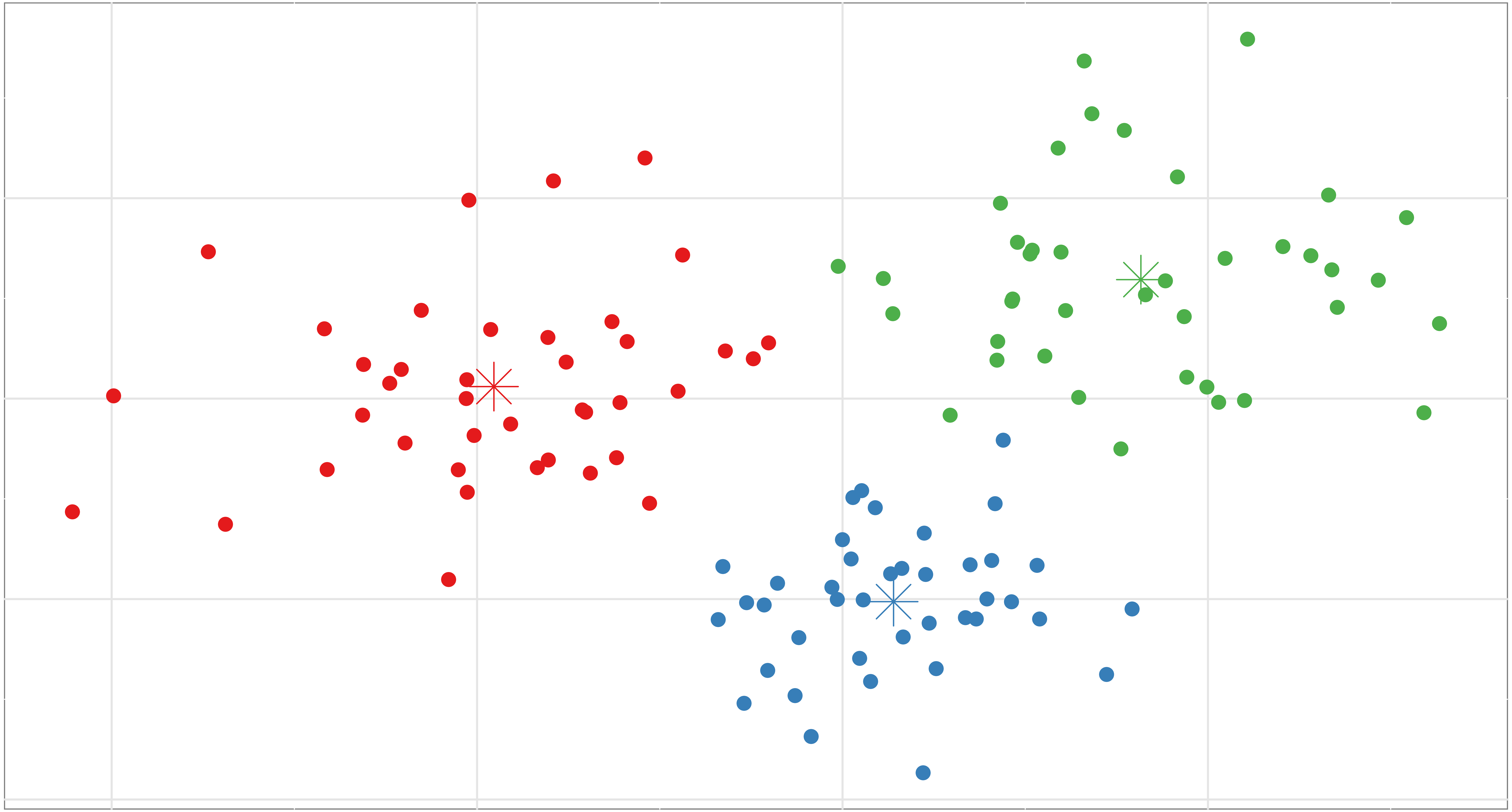
Step 3: Move centroid to center of assigned points (3)



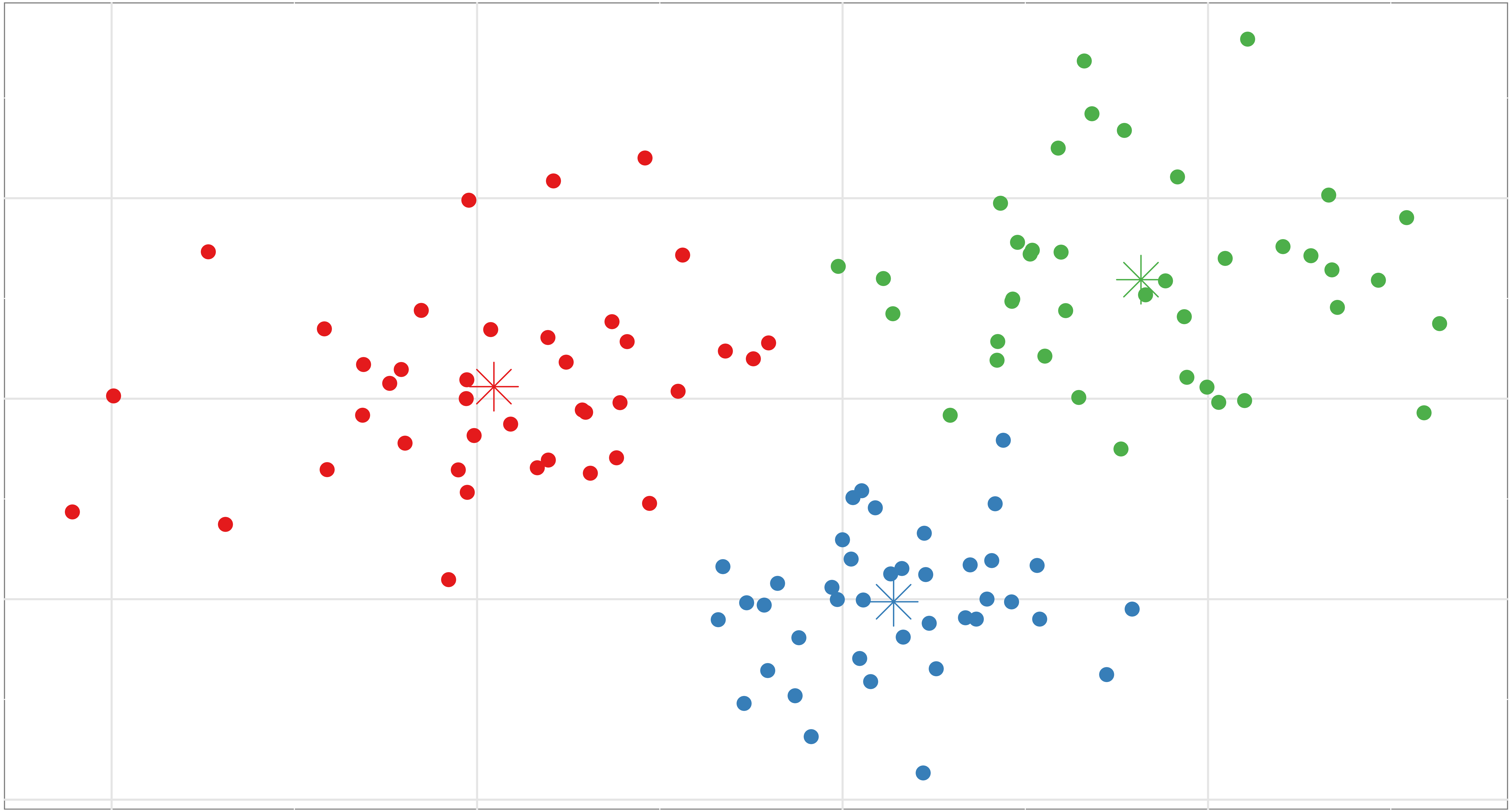
Step 2: Assign each data point to the nearest centroid (4)



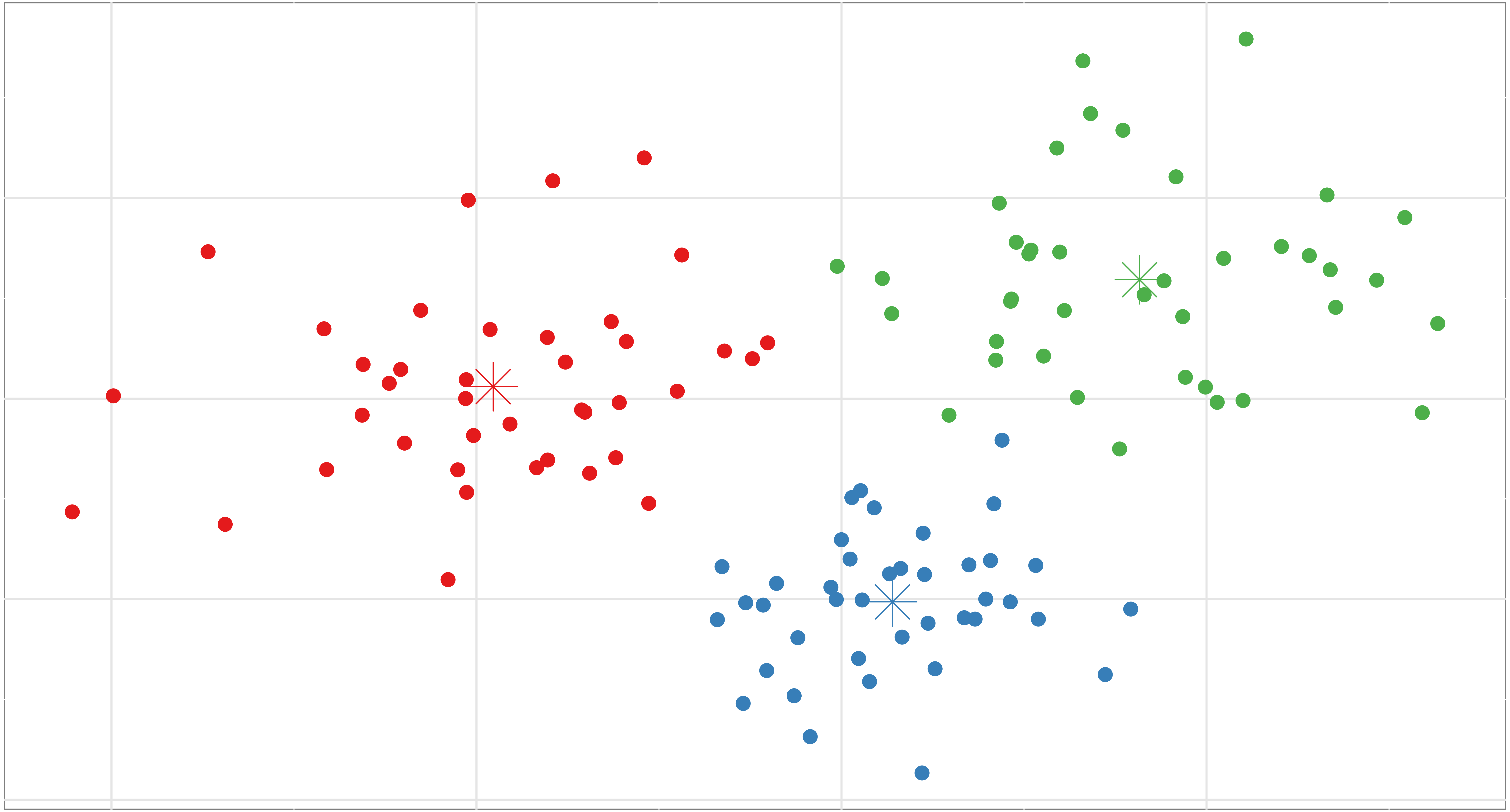
Step 3: Move centroid to center of assigned points (4)



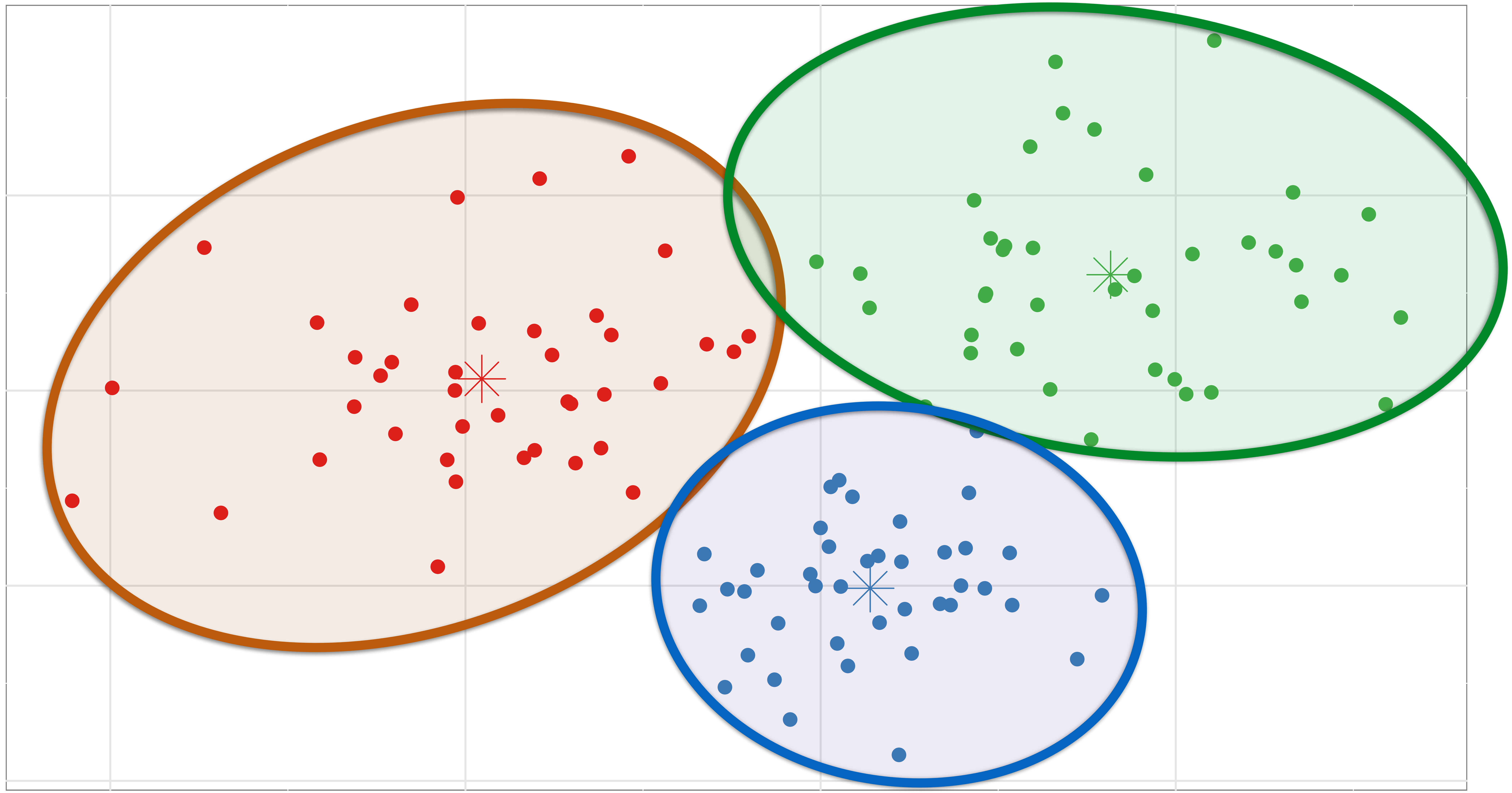
Step 2: Assign each data point to the nearest centroid (5)



Step 3: Move centroid to center of assigned points (5)



Step 3: Move centroid to center of assigned points (5)



K-Means: Got a problem with it?

Before starting, pick the number of clusters, K

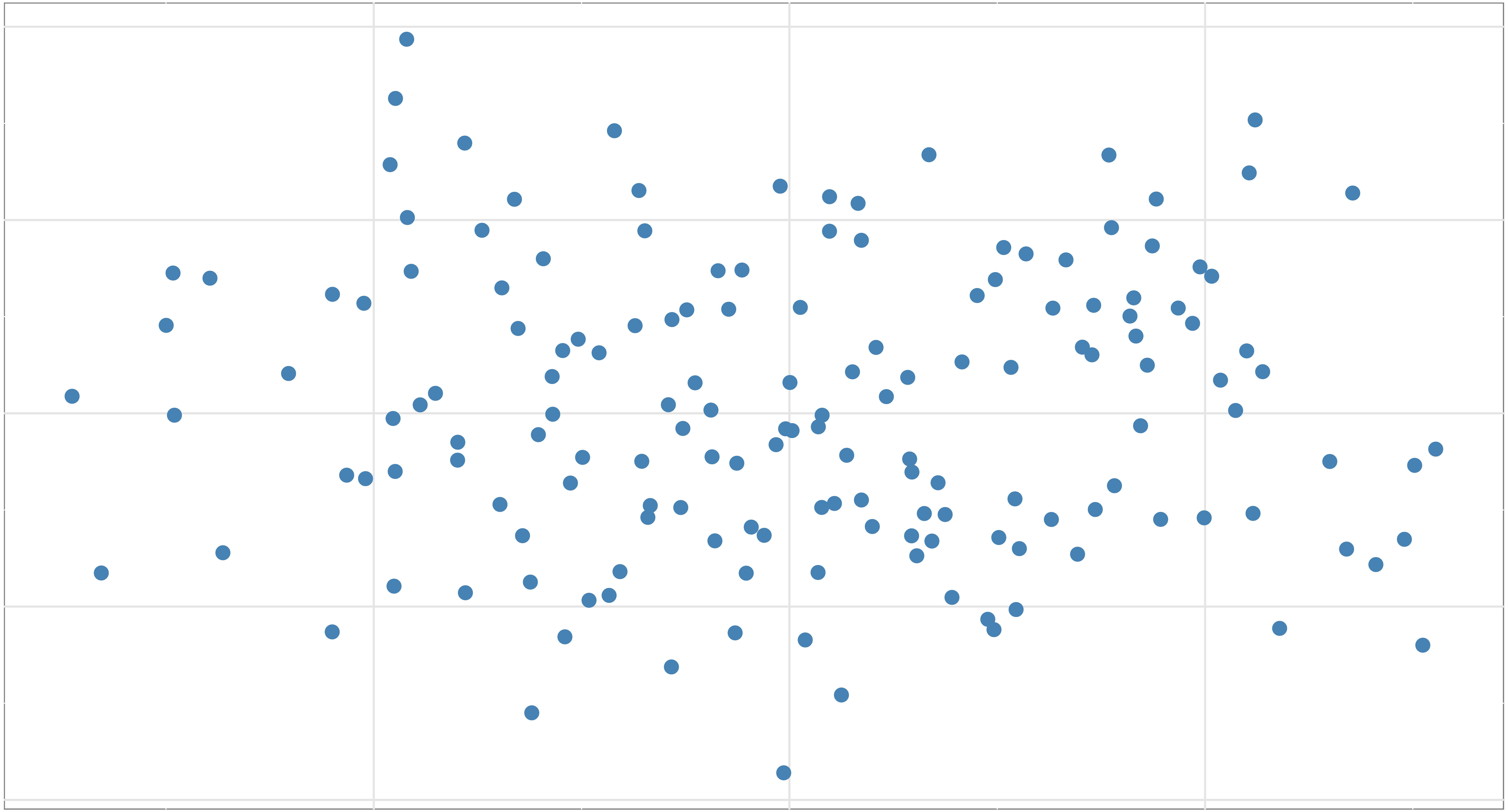
1. Pick K random centroids within data range
2. Assign each data point to the nearest centroid
3. Move centroid to center of assigned points
4. Repeat steps 2 and 3 until centroid stops shifting

K-Means: Got a problem with it?

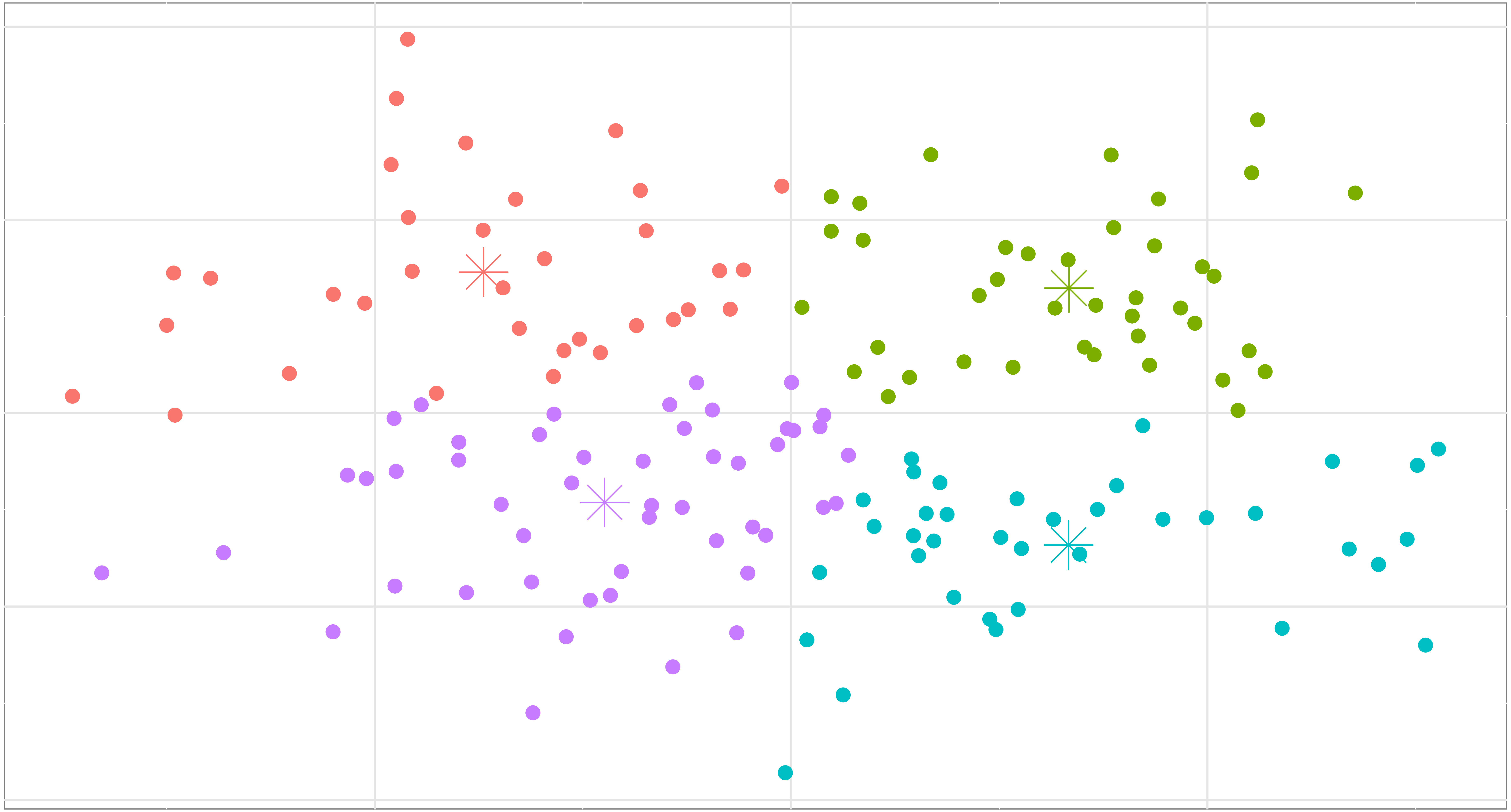
Before starting, pick the number of clusters, K   Subjective

1. Pick K random centroids within data range   Not Repeatable
2. Assign each data point to the nearest centroid   Sensitive to Scale
3. Move centroid to center of assigned points
4. Repeat steps 2 and 3 until centroid stops shifting

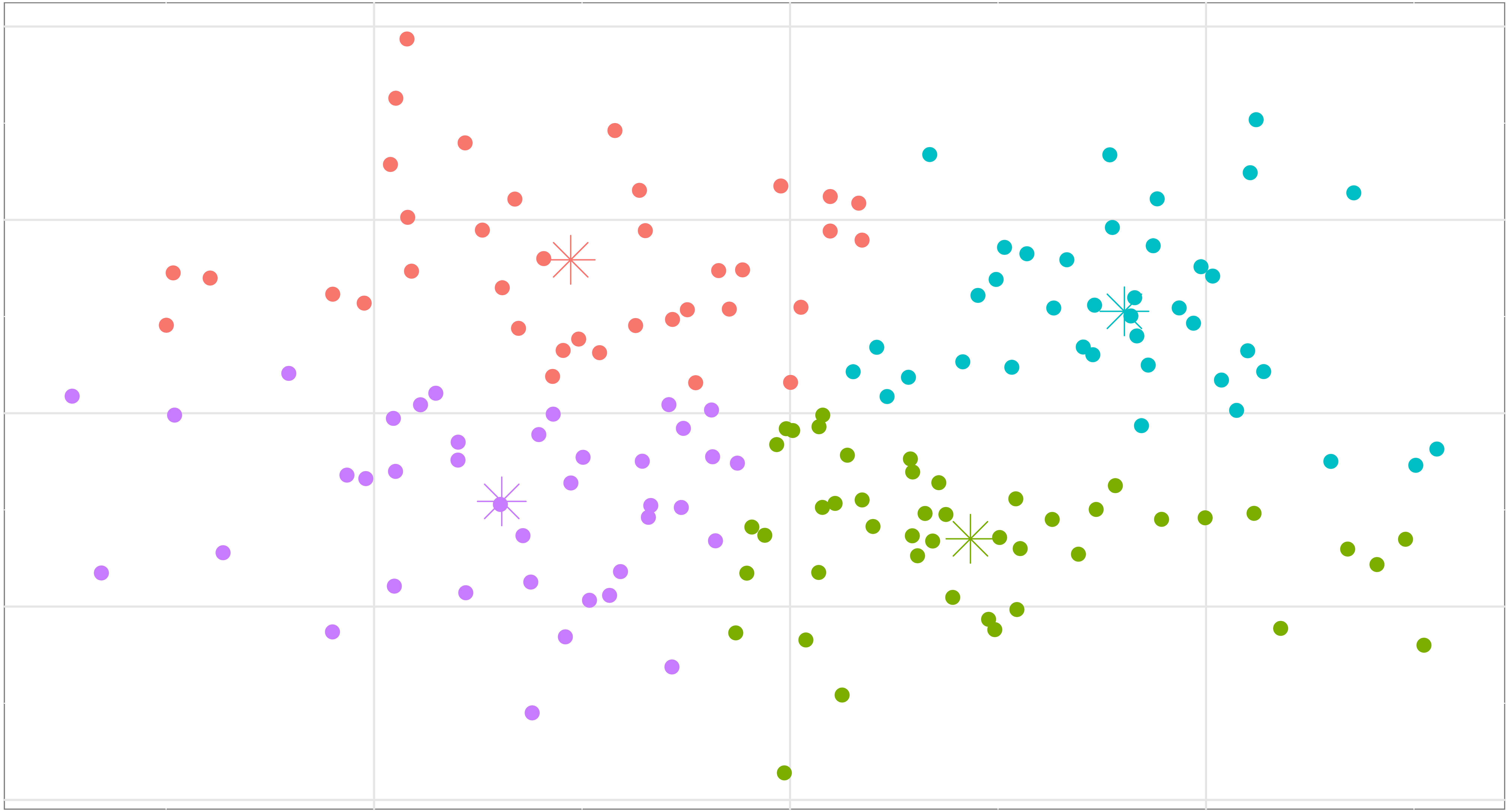
How many clusters?



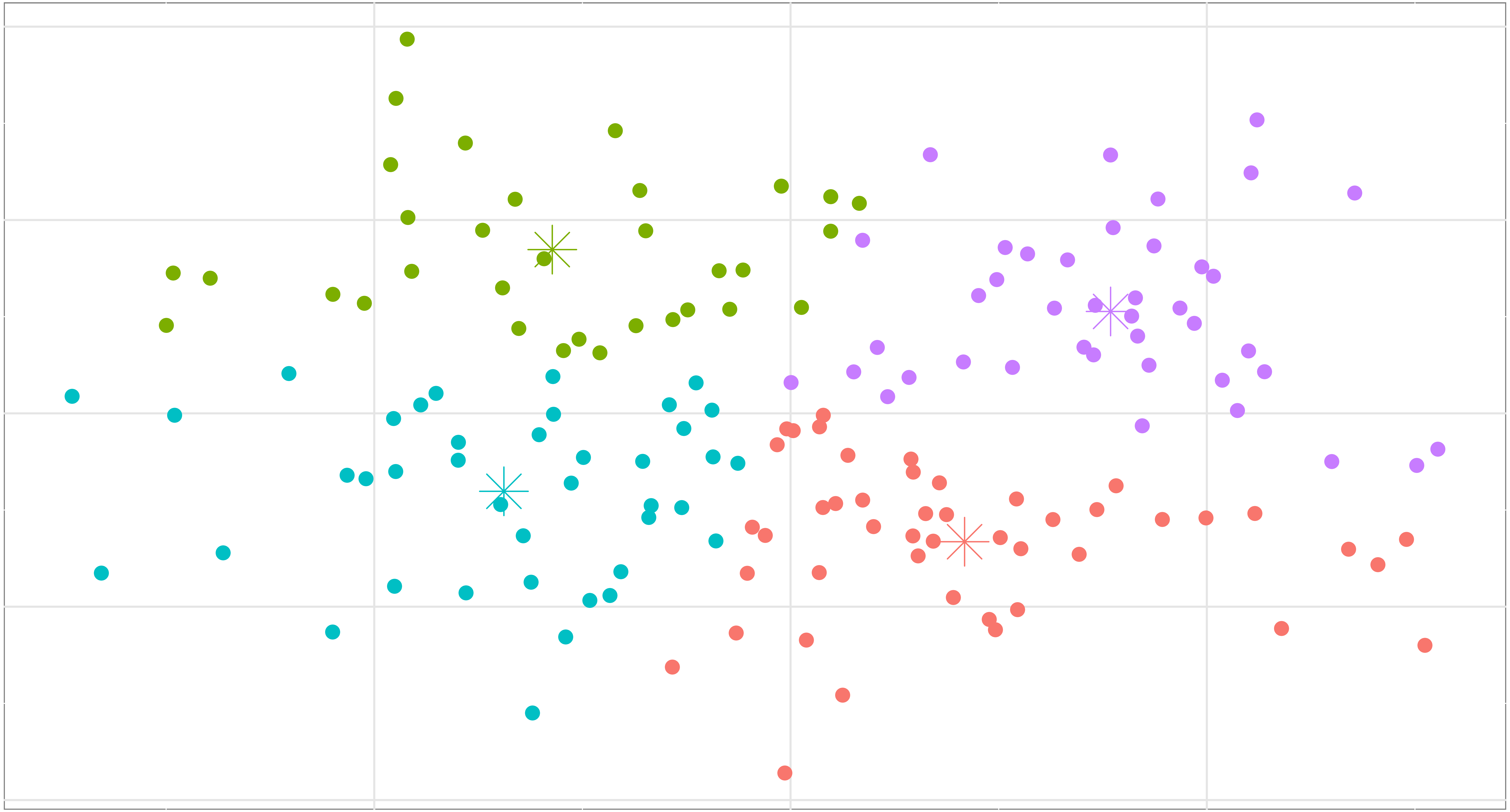
Random Start…



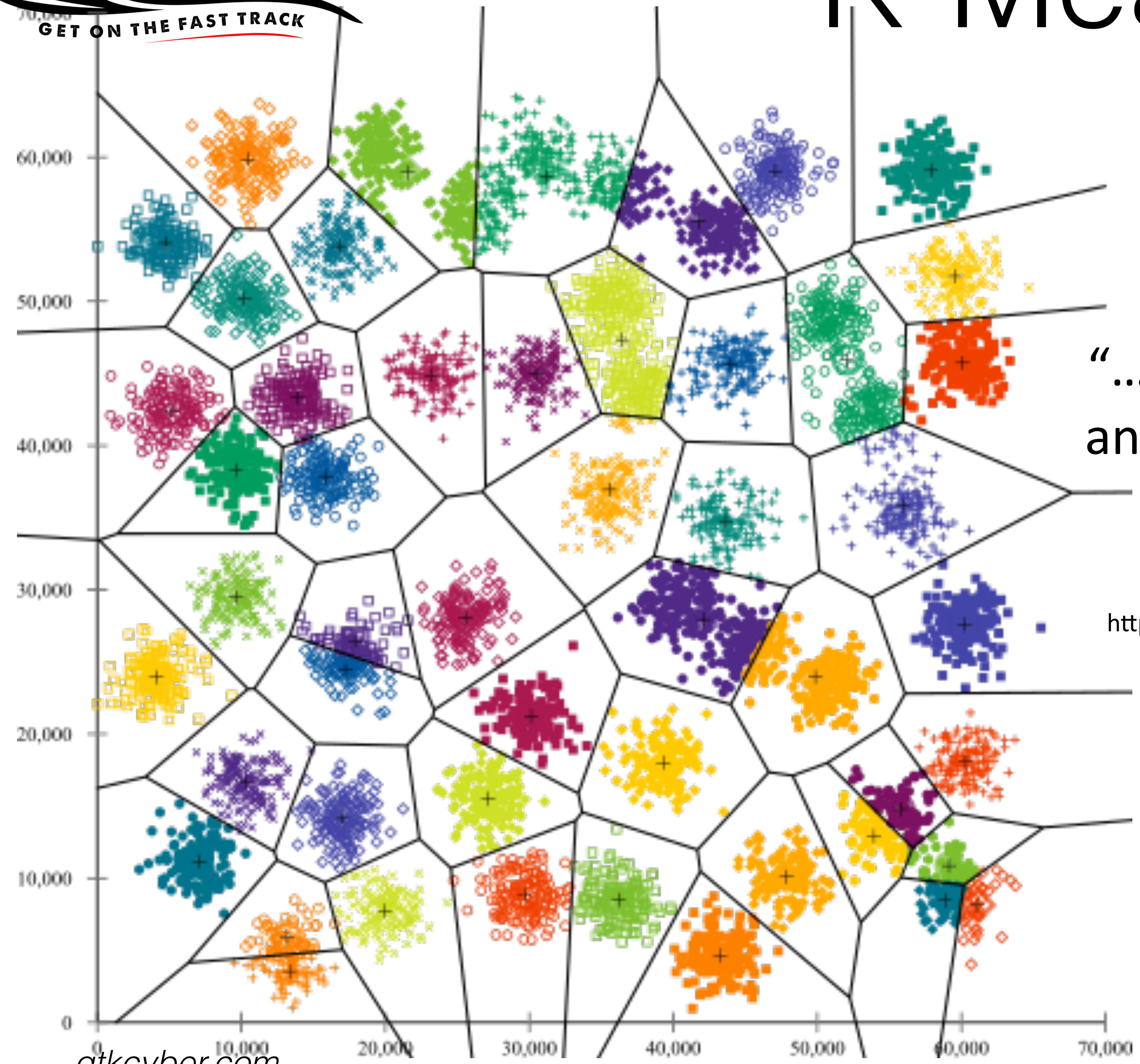
Random Start…



Random Start…



K-Means

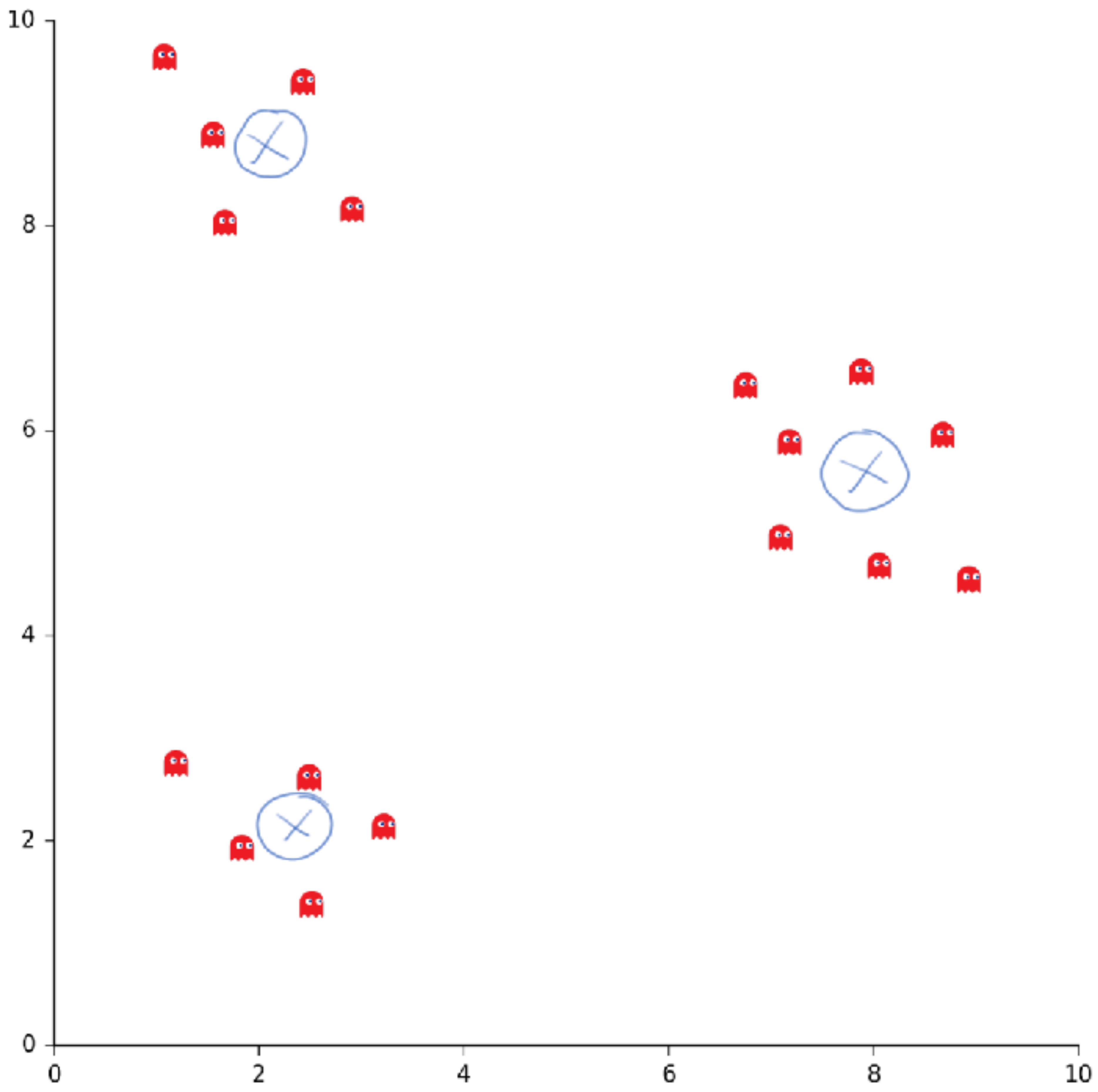


"…it's too easy to throw k-means on your data, and nevertheless get a result out (that is pretty much random, but you won't notice)."

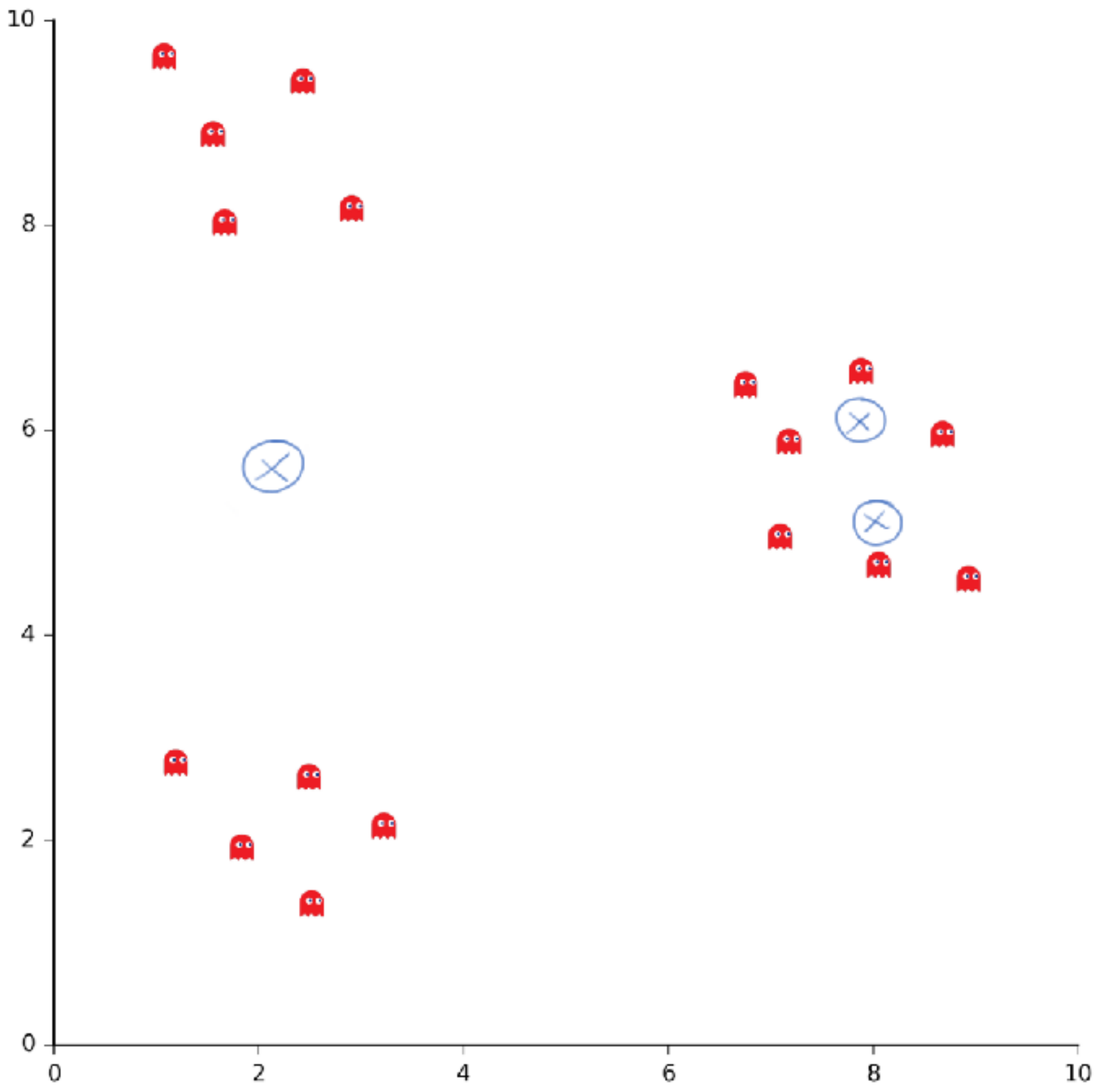
— Anony-Mousse

http://stats.stackexchange.com/questions/133656/how-to-understand-the-drawbacks-of-k-means

Which one is correct?

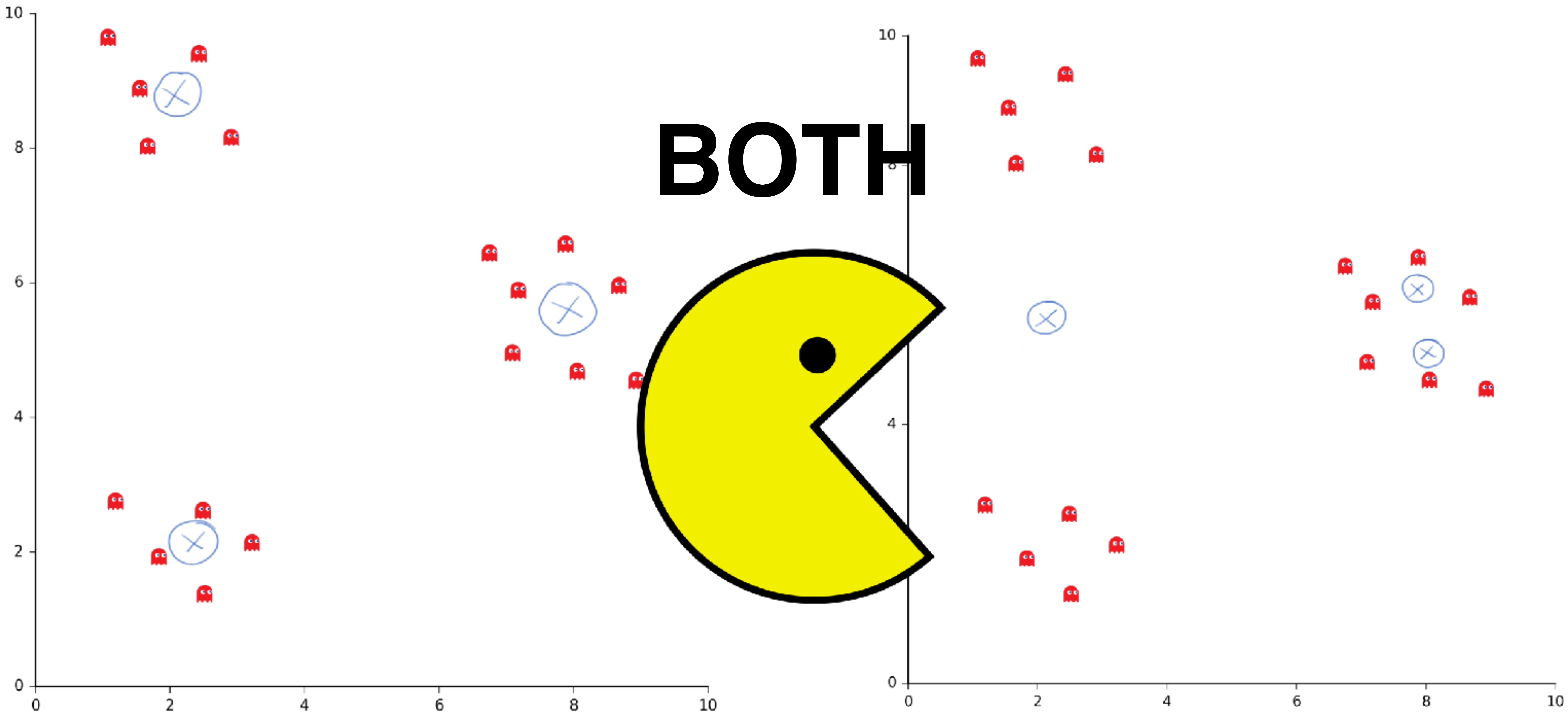


A



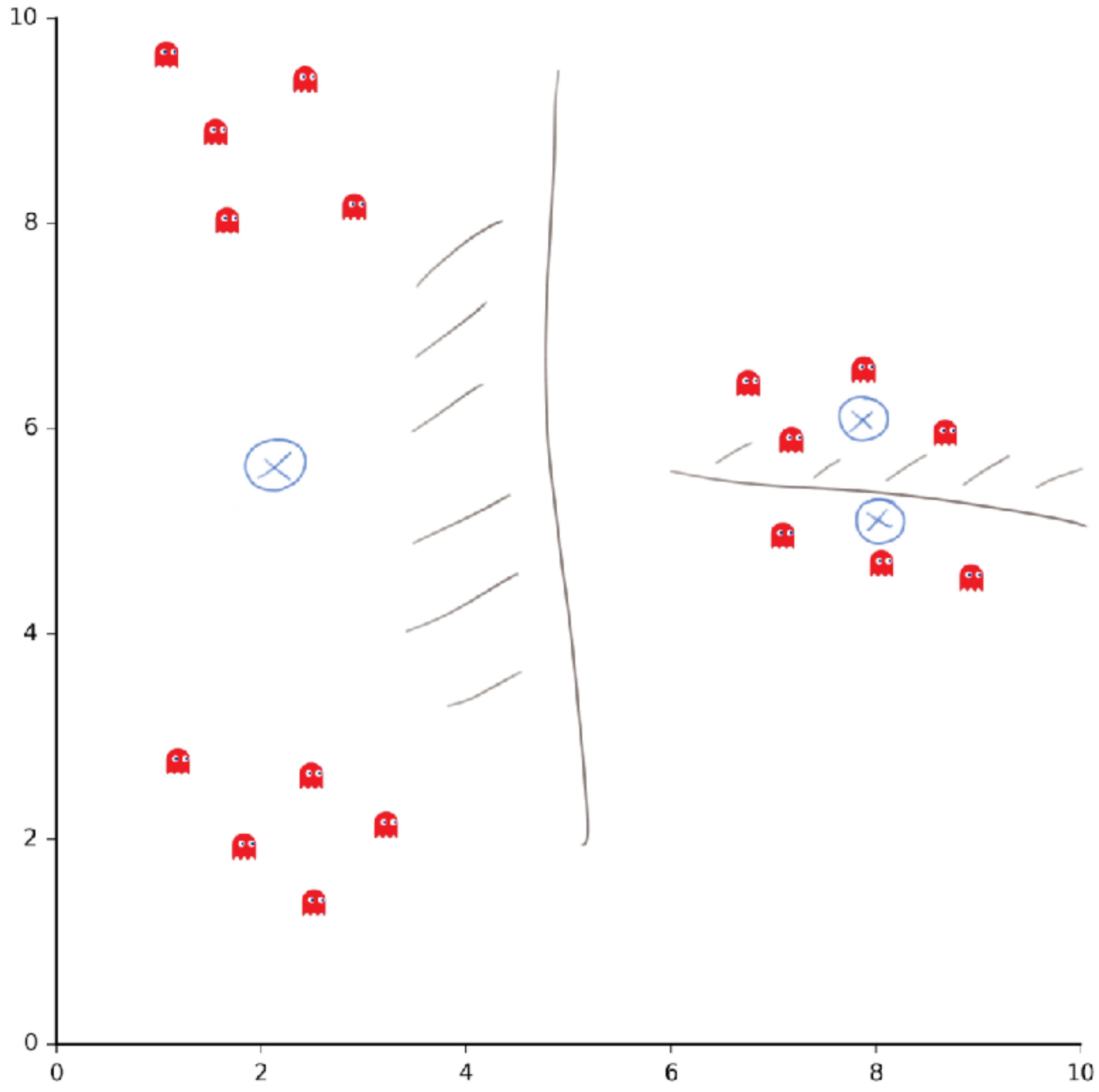
B

Which one is correct?



BOTH

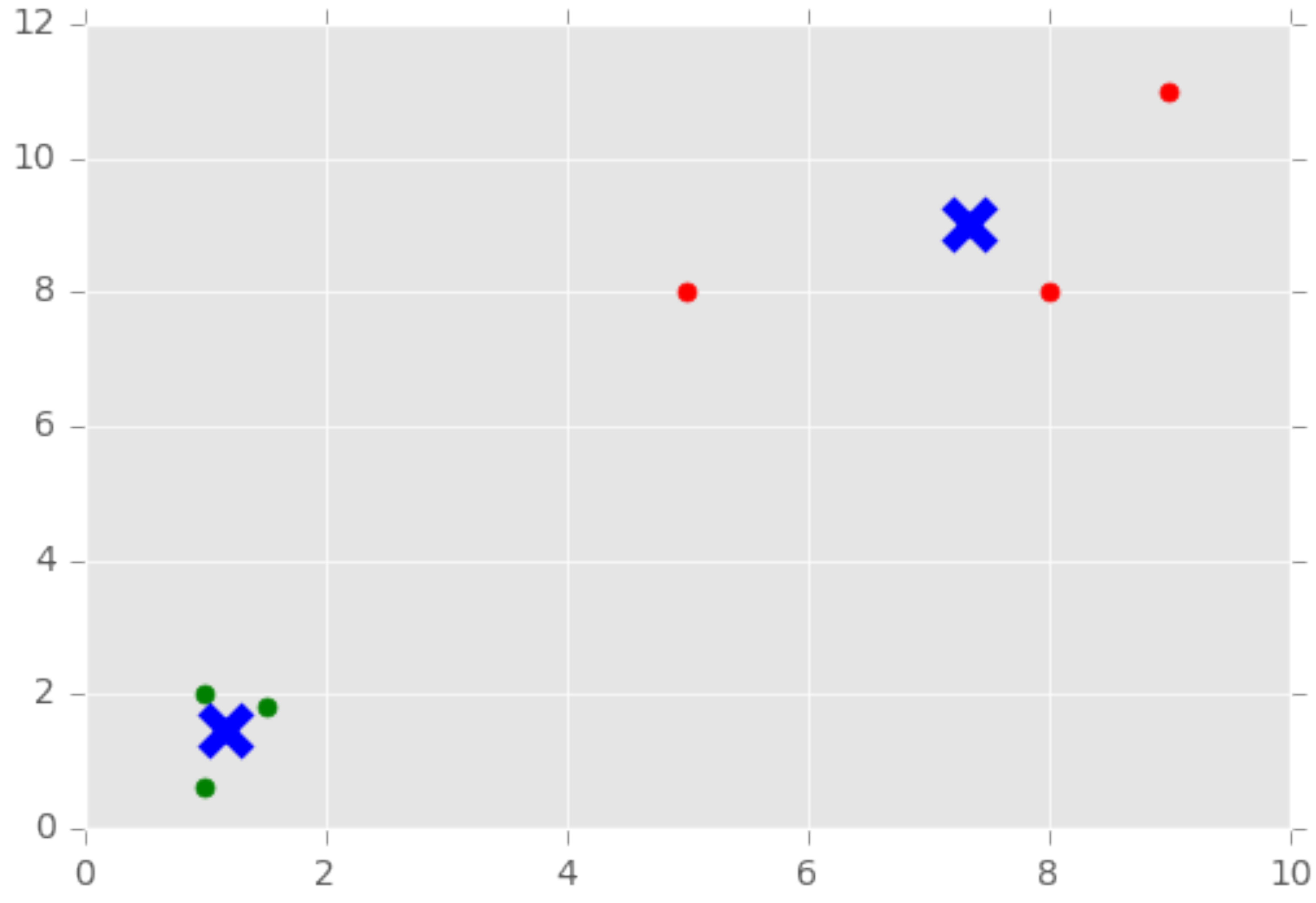
Pain of optimization…Being stuck at sub-optimal local minimum…



Initial guess matters!
Same outcome
cannot be guaranteed

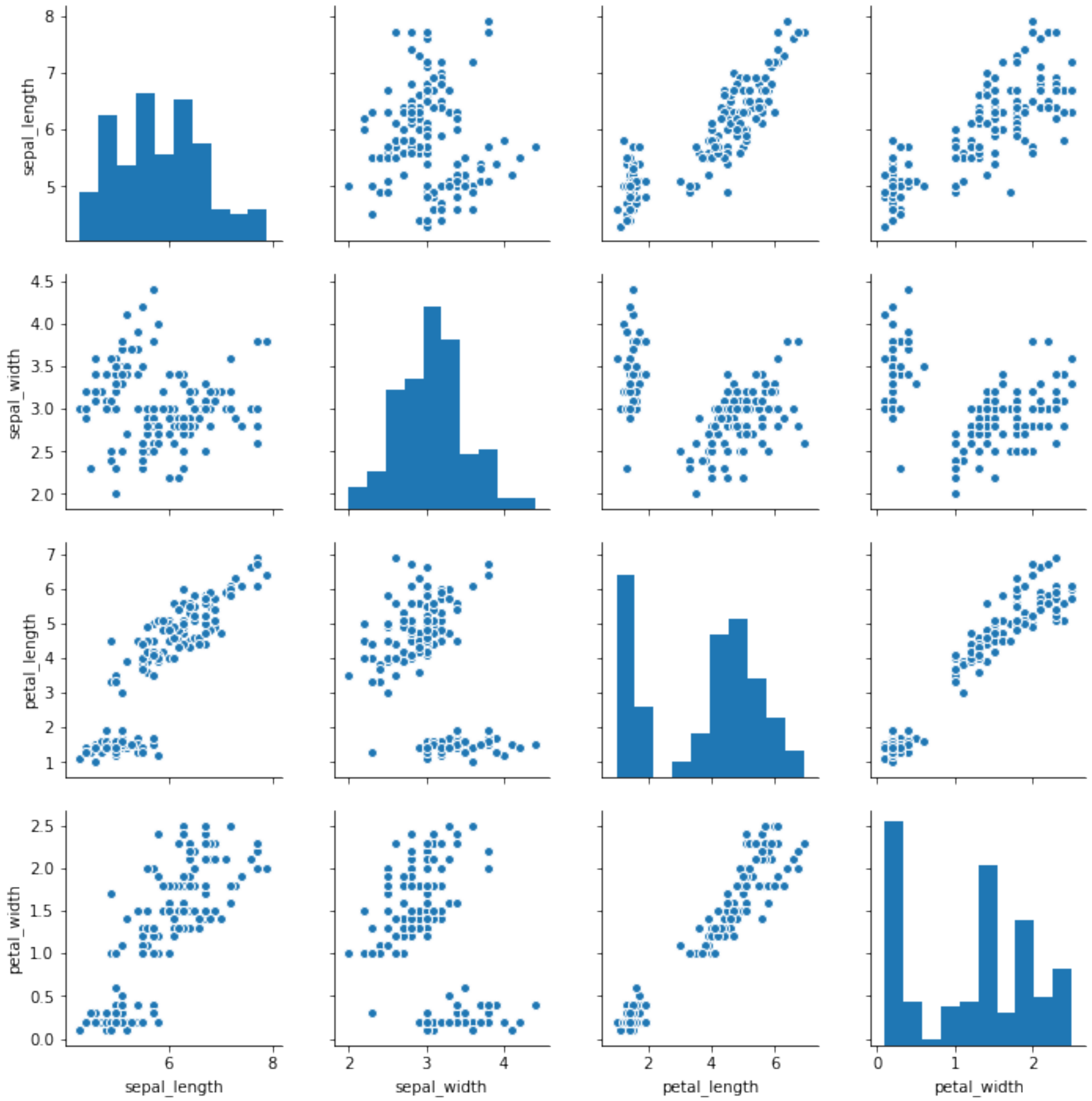
K-Means in practice (Python version)

```
#Import from Scikit-learn
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2)
kmeans.fit(data)

centroids = kmeans.cluster_centers_
labels = kmeans.labels_f
```



The Dataset

```
sns.pairplot(<data>)
```

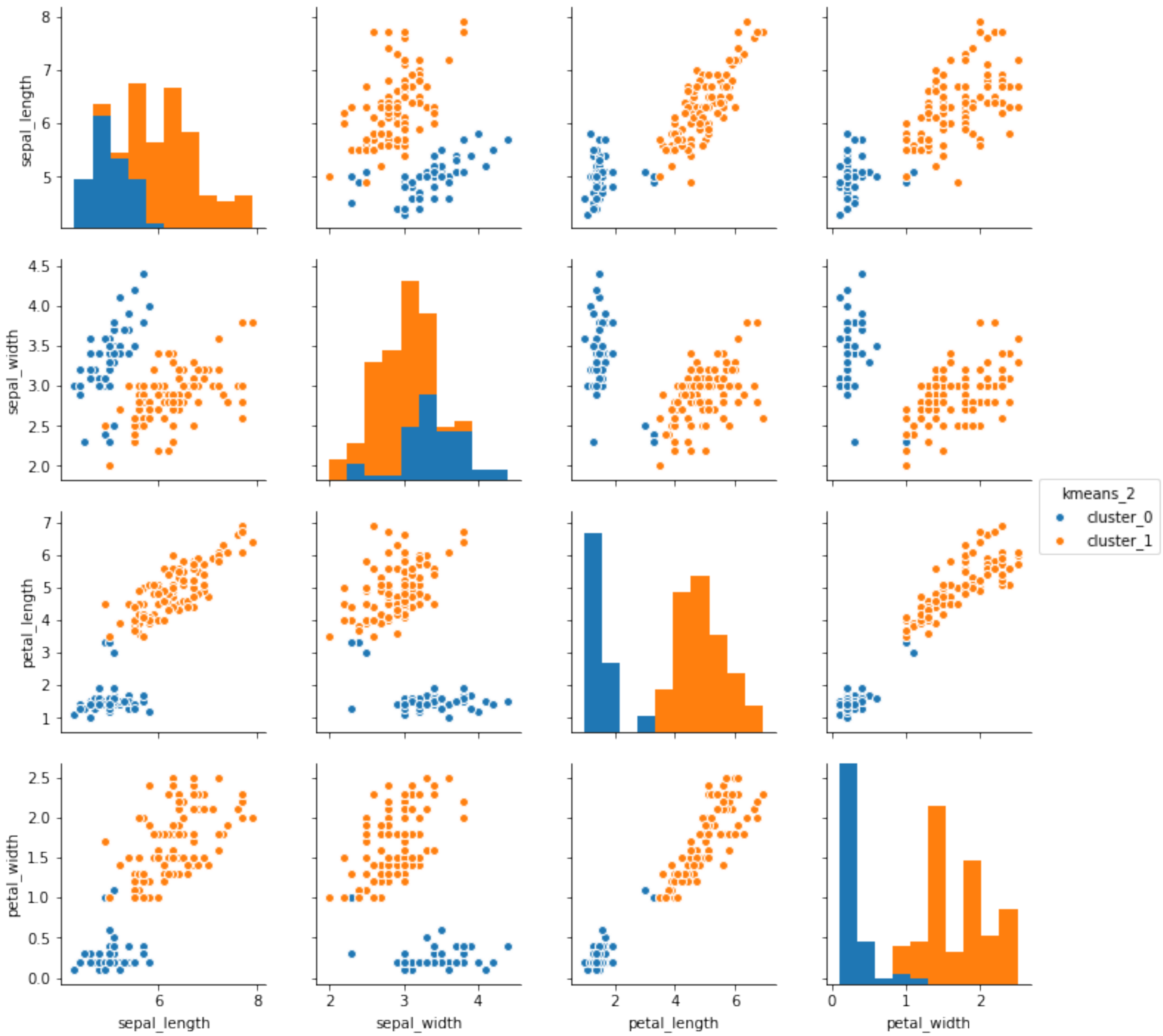


K-Means Clustering

```
kmeans = KMeans( n_clusters=2 )
kmeans.fit( <data> )
```

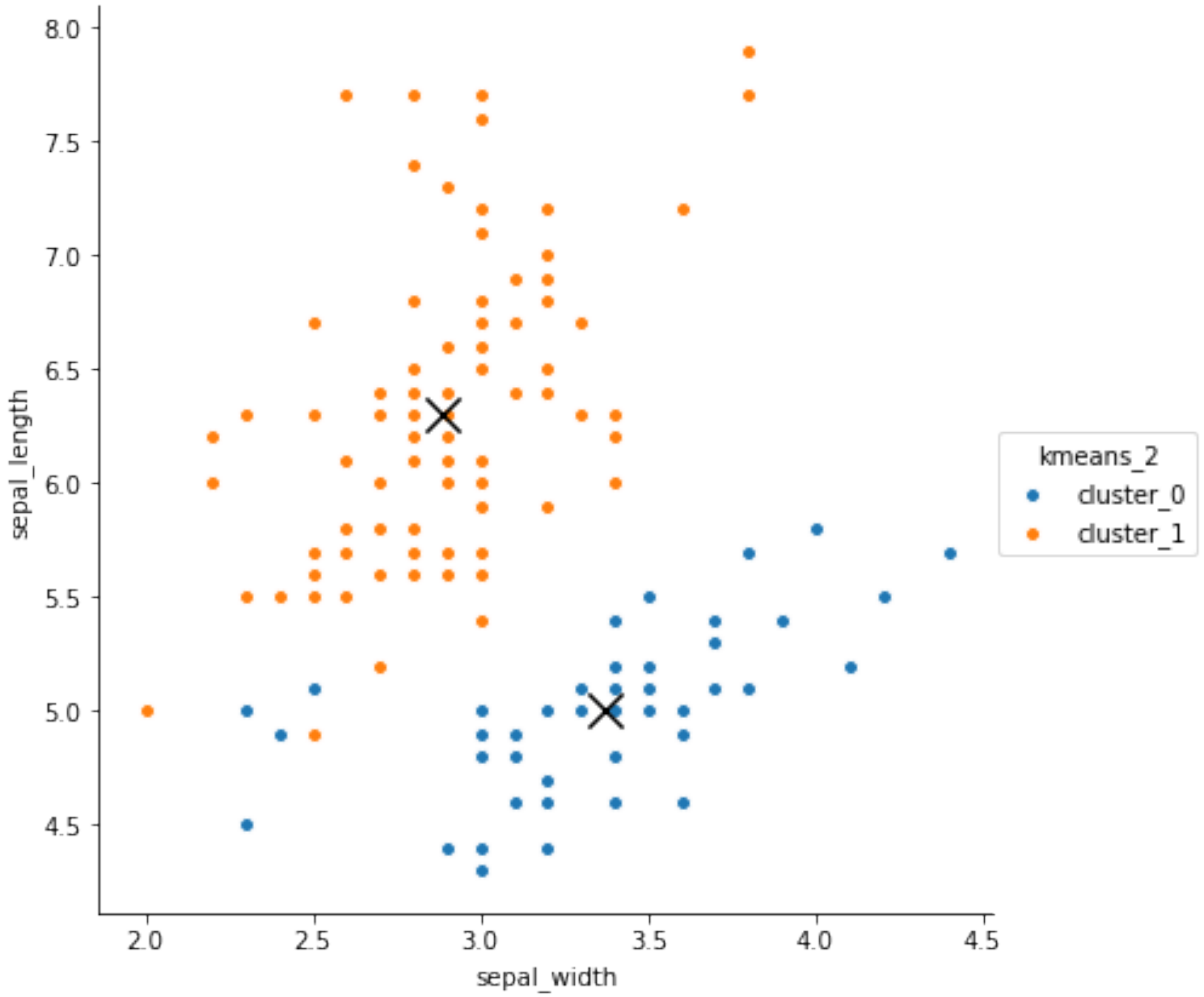
K-Means Clustering

```
sns.pairplot(<data>,hue="kmeans_2")
```



K-Means Clustering

```
sns.pairplot(<data>,x_vars="col_1",y_vars="col_2",hue="kmeans_2",size=6)
plt.scatter(<cluster_centers>,<col_2>, linewidths=3, marker='x', s=200,
c='black')
```



K-Means is affected by the scale of every feature.

Feature Scaling

For k-means clustering, features must be scaled to the same ranges of values to contribute "equally" to the euclidean distance calculation.

Each row is transformed per-column by:

- Subtracting from the element in each row the mean for each feature (column) and then taking this value and

- Dividing by that feature's (column's) standard deviation.

Feature Scaling

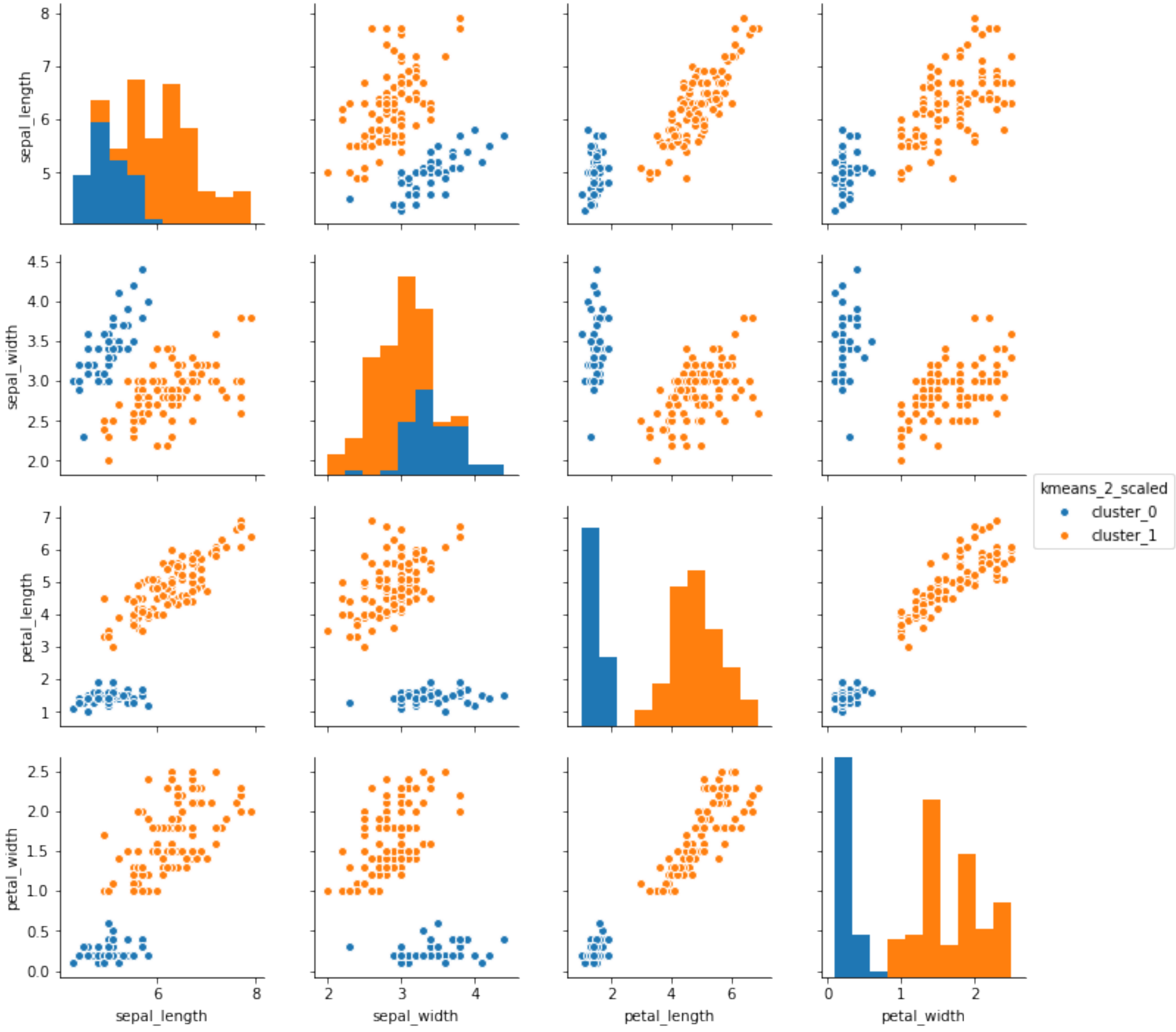
```
# center and scale the data
scaler = StandardScaler()

raw_data_scaled = scaler.fit_transform( <data> )

data_scaled = pd.DataFrame( raw_data_scaled, columns=features )
```

Feature Scaling

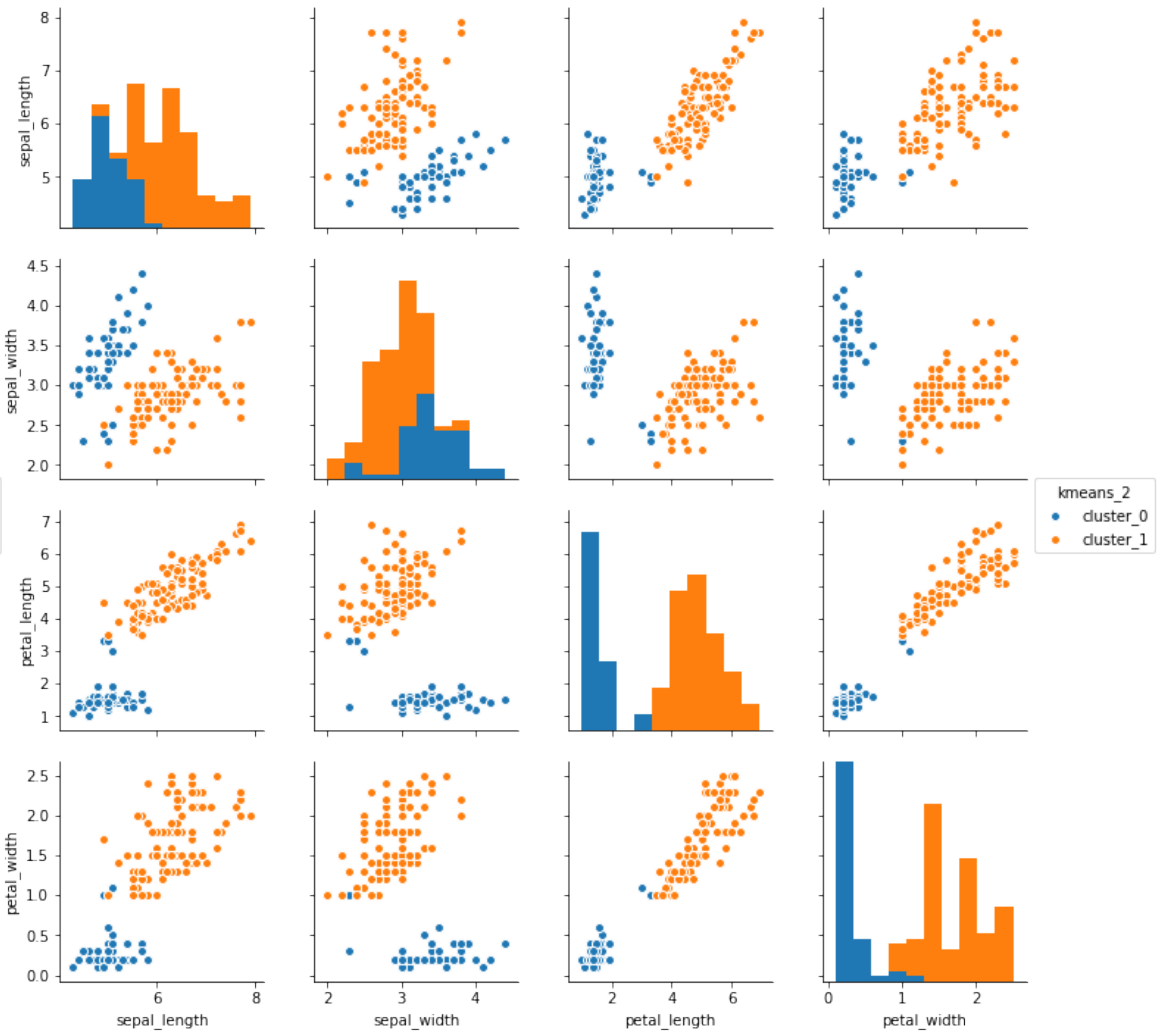
```
# K-means on scaled data
km = KMeans( n_clusters=2 )
km.fit( <scaled_data> )
```

Feature Scaling

Scaled Features



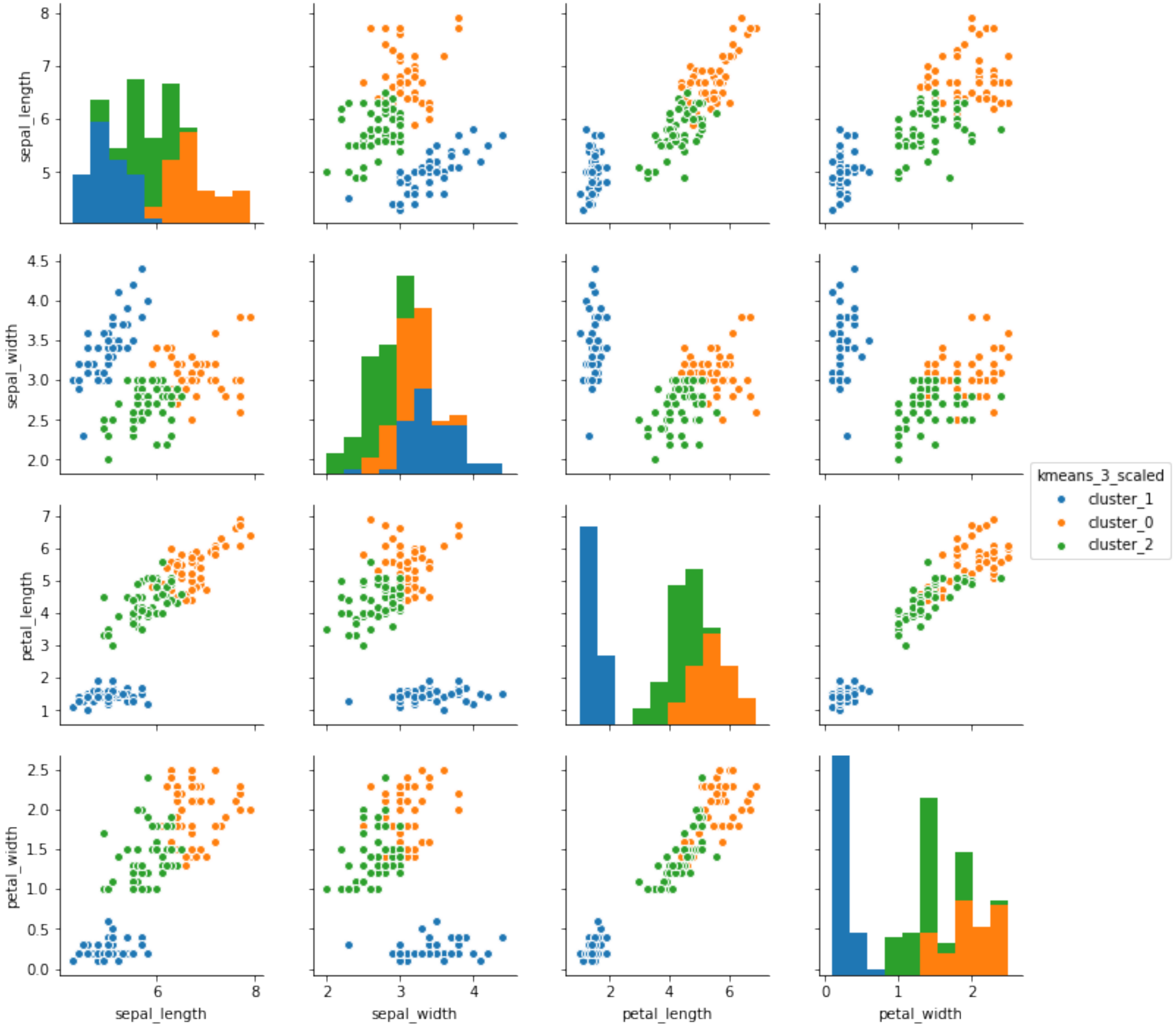
Unscaled Features



gtkcyber.com

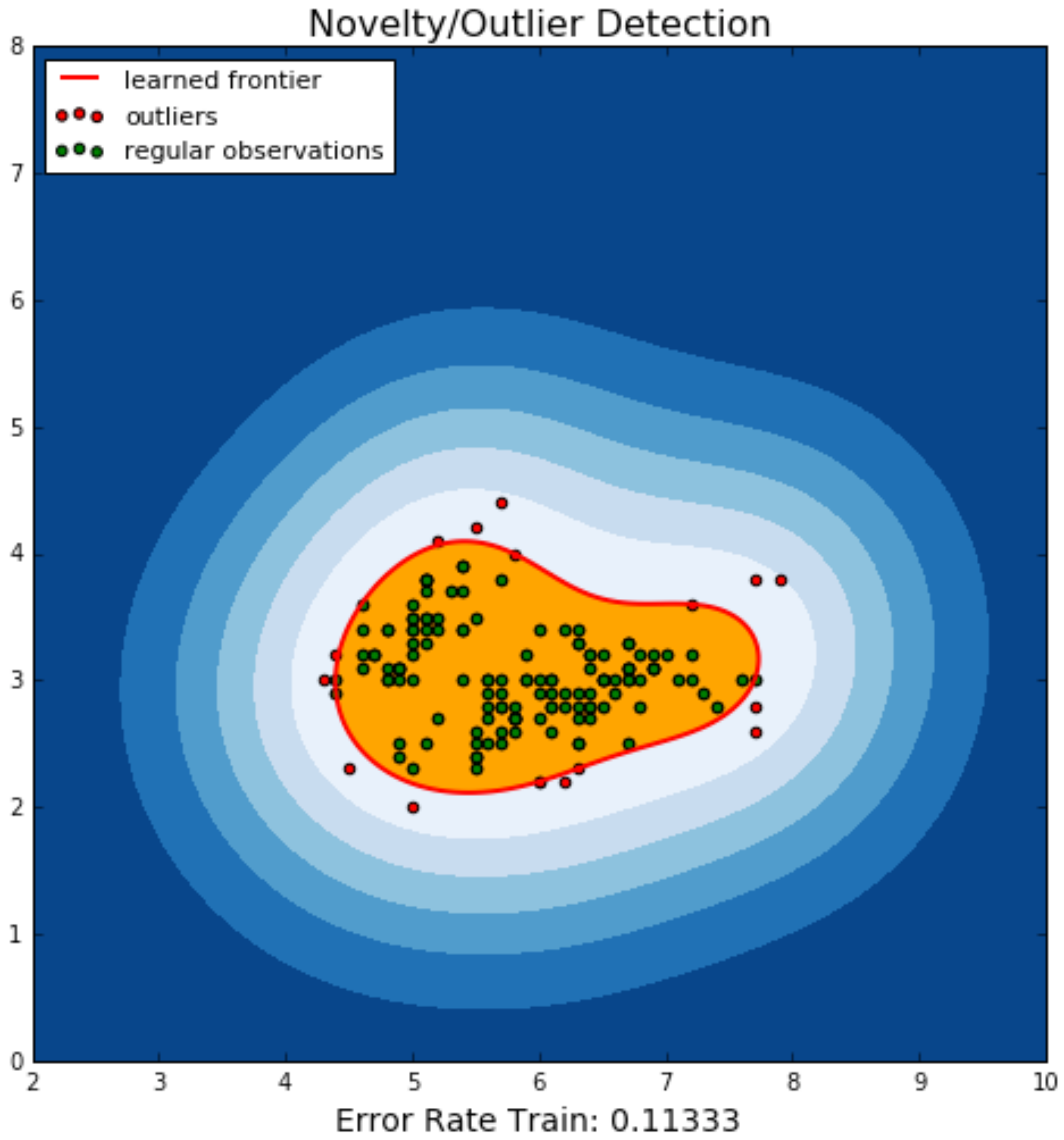
More Clusters

```
km3 = KMeans(n_clusters=3)
km3.fit(scaled_data)
```



Outlier Detection

```
clf = svm.OneClassSVM( tol=0.001, nu=0.1)
clf.fit(X)
target_pred_outliers=clf.predict(X)
```



Delete n% of “outlier data”,
here ~10%

http://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html

Evaluating your Model

The Silhouette Coefficient is a common metric for evaluating clustering "performance" in situations when the "true" cluster assignments are not known.

b = mean distance to next nearest cluster

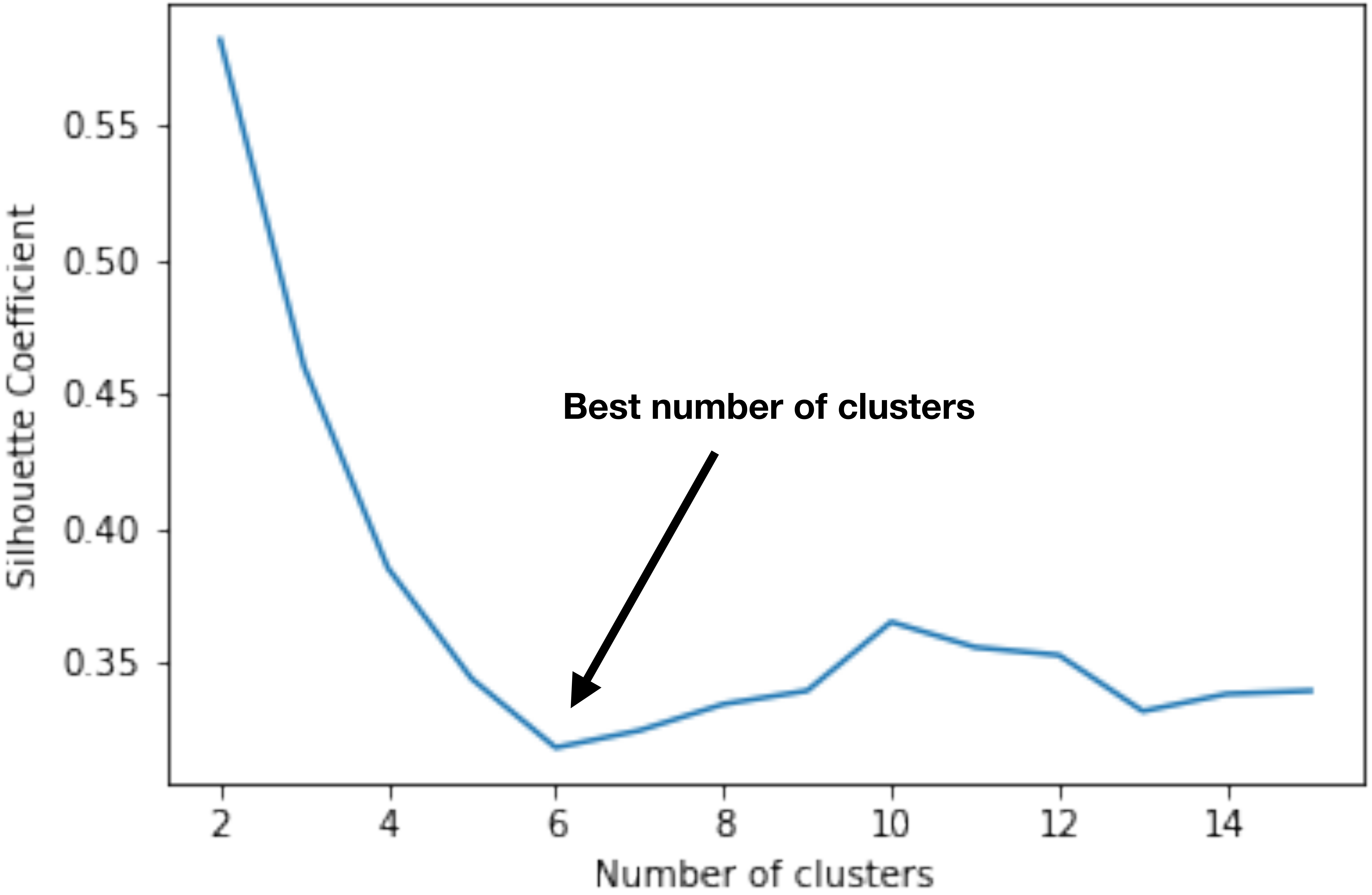
a = mean distance to other points in cluster

```
silhouette_coeff = (b - a) / max(a,b)
```

Evaluating your Model

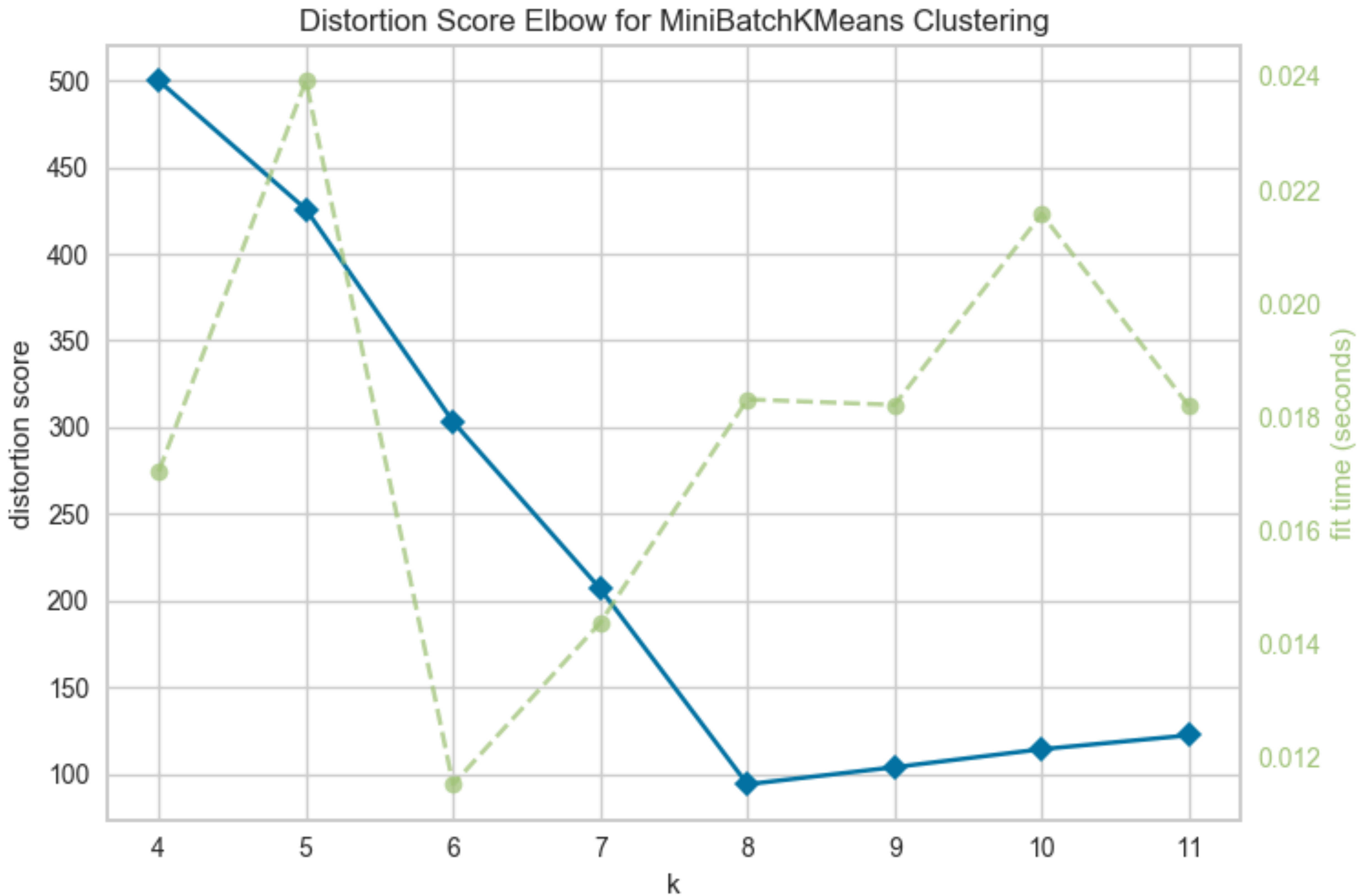
```
k_range = range(2,16)
scores = []
for k in k_range:
    km_ss = KMeans(n_clusters=k, random_state=1)
    km_ss.fit(iris_data_scaled)
    scores.append(silhouette_score(<data>,
km_ss.labels_))
```

Evaluating your Model



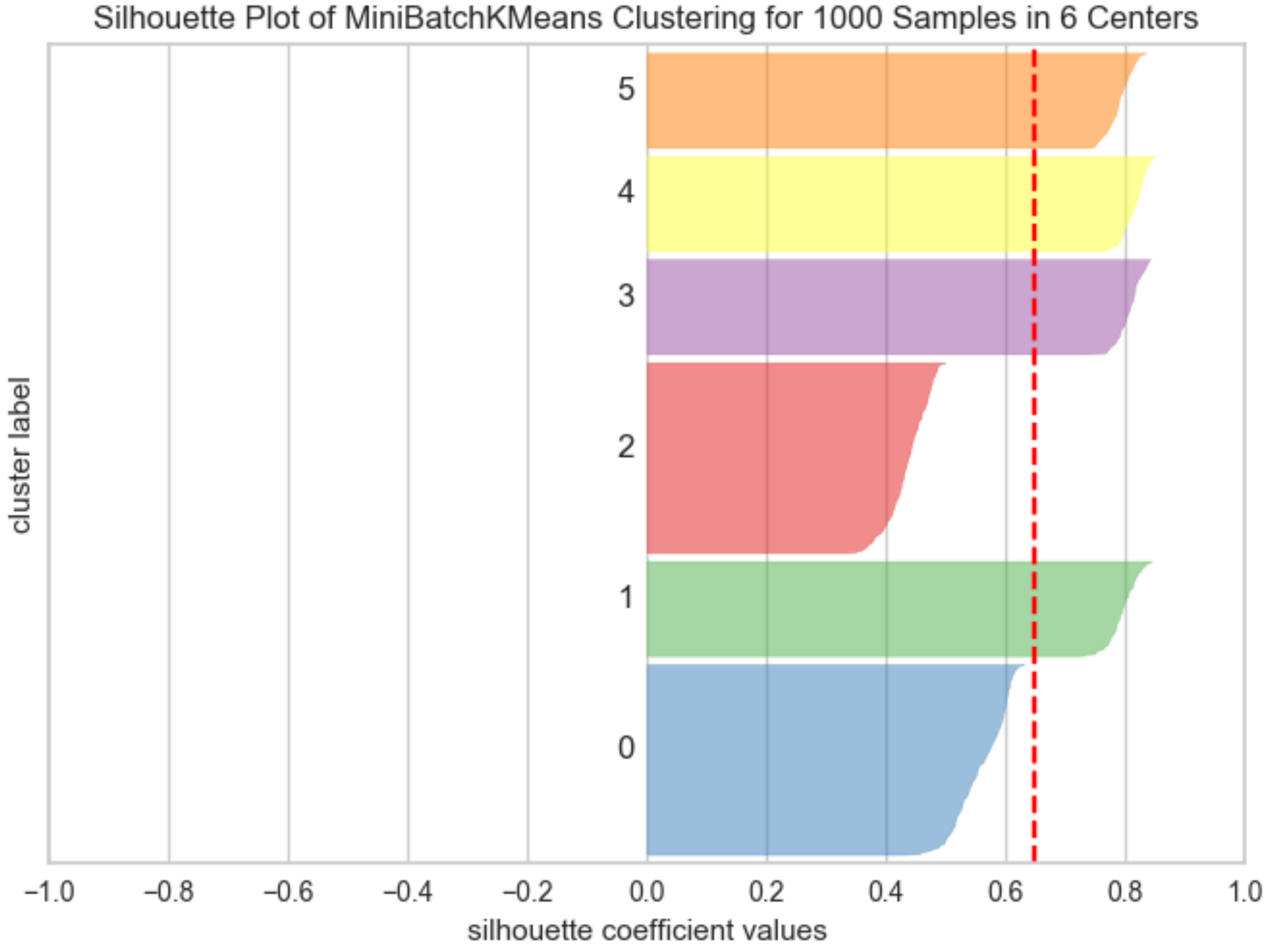
Evaluating your Model

```
from yellowbrick.cluster import KElbowVisualizer
visualizer = KElbowVisualizer(KMeans(), k=(4,12))

visualizer.fit(X)
visualizer.poof()
```



Evaluating your Model

```
from yellowbrick.cluster import SilhouetteVisualizer
model = MiniBatchKMeans(6)
visualizer = SilhouetteVisualizer(model)

visualizer.fit(X)
visualizer.poof()
```



DBSCAN

DBSCAN stands for **D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise.

Whereas K-means does not care about the density of data, DBSCAN does, under the assumption that regions of high density in your data should be treated as clusters.

DBSCAN

DBSCAN does not allow you to specify how many clusters you want. Instead, you specify 2 parameters:

- **є (epsilon)**: This is the maximum distance between two points to allow them to be neighbors

- **min_samples**: The number of neighbors a given point is allowed to have to be able to be part of a cluster

Any points that don't satisfy the criteria of being close enough to other points are labeled outliers and all fall into a single "cluster" (their cluster label by default is -1).

DBSCAN

DBSCAN works as follows:

1. Choose an arbitrary starting point in your dataset that has not been seen.

2. Retrieve this point's ϵ-neighborhood (all points that are within a distance ϵ from it), and if it contains at least ***min_samples**, a cluster is started.

3. Otherwise, the point is labeled as an outlier (-1). Note: This point might later be found in a sufficiently sized ε-environment of a different point and hence be made part of a cluster.

4. If a point is found to be a dense part of a cluster, its ϵ-neighborhood is also part of that cluster. All points that are found within the ϵ-neighborhood are added, as is their own ϵ-neighborhood when they are also dense.

5. Continue until the density-connected cluster is completely found.

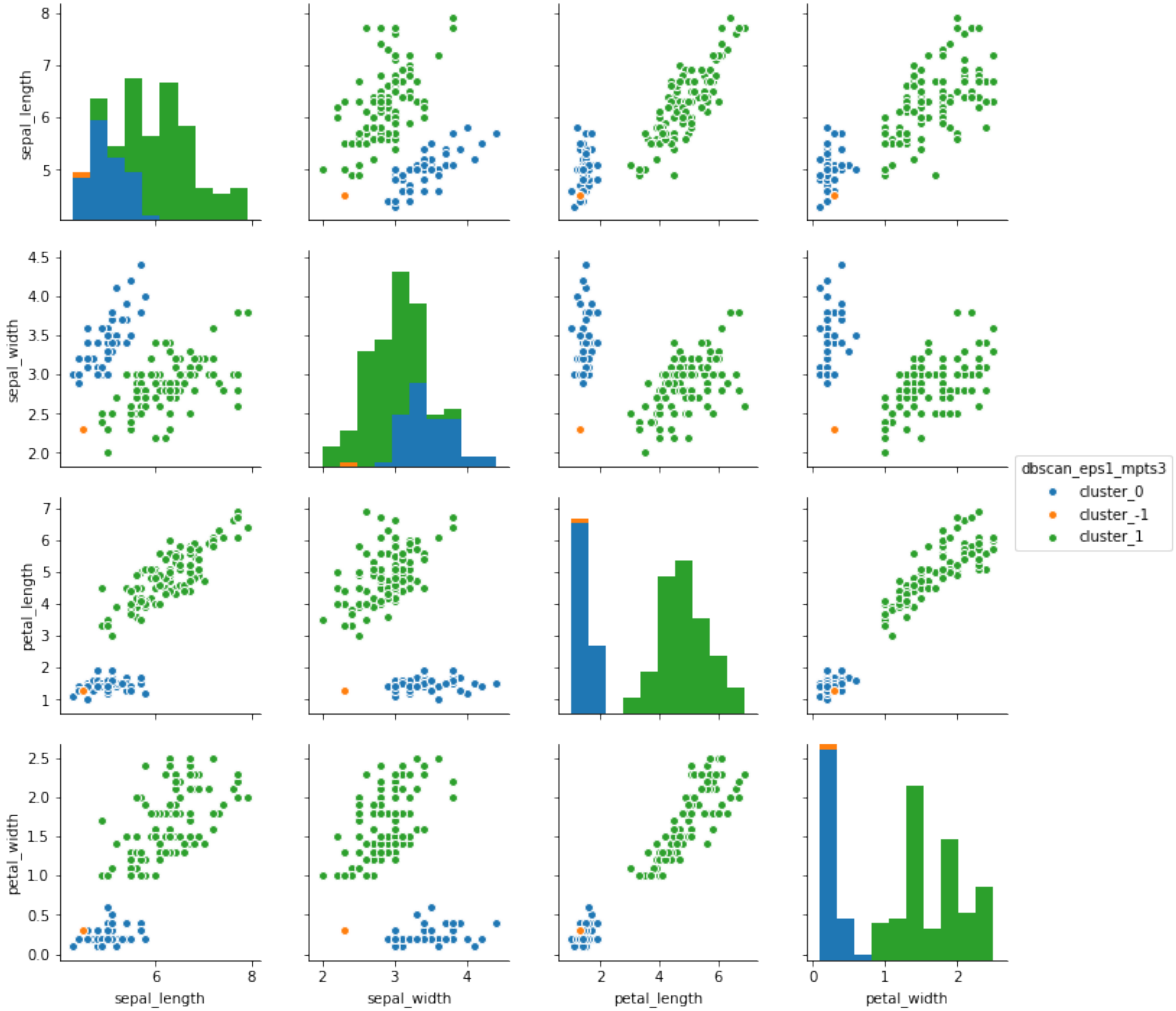
6. Find a new unvisited point to process and repeat.

DBSCAN

```
db = DBSCAN(eps=1, min_samples=3)
db.fit(<scaled_data>)
```

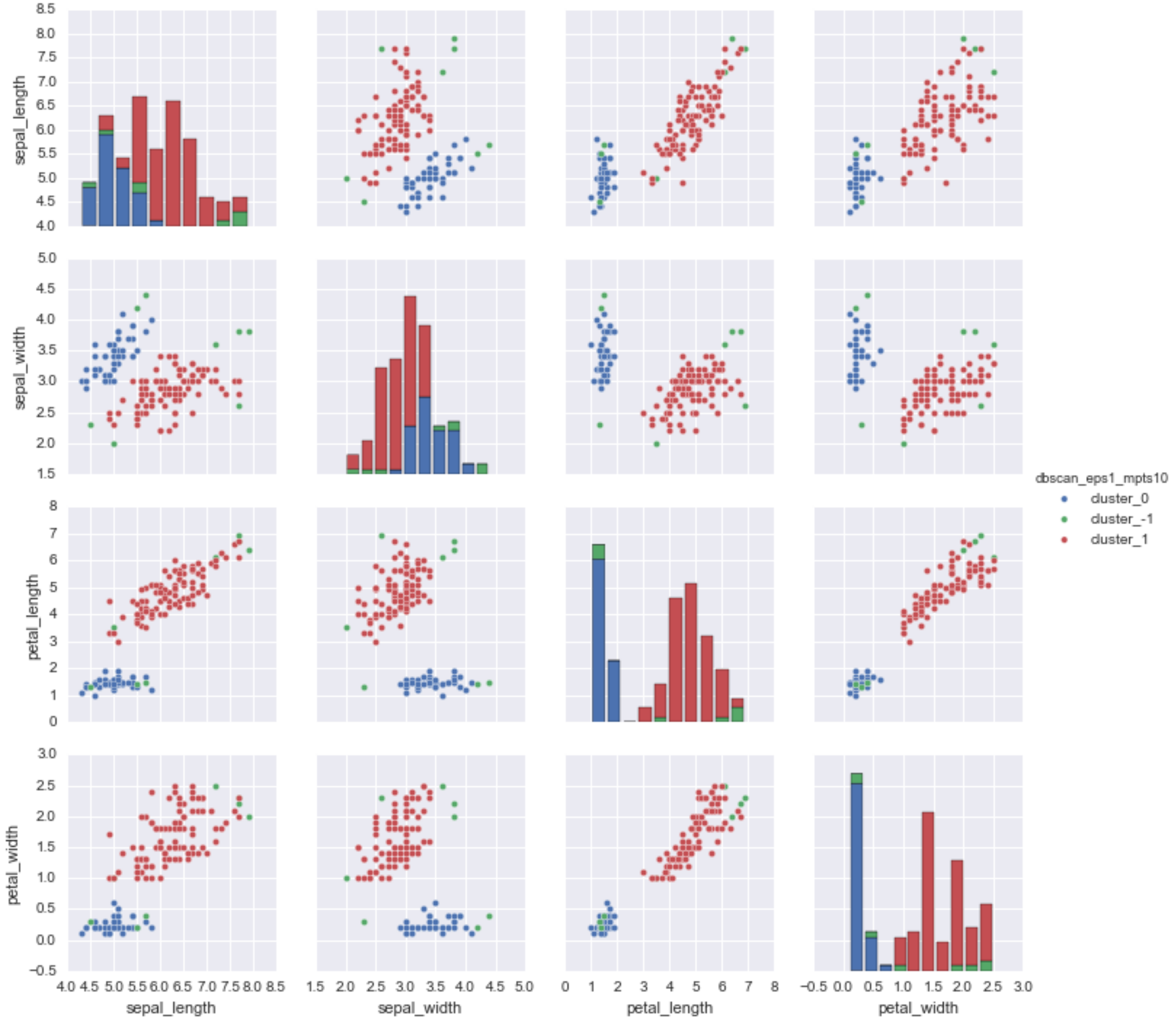
DBSCAN

```
data_no_names['dbscan_eps1_mpts3'] = [ "cluster_" + str(label) for label in db.labels_ ]
sns.pairplot(data_no_names,hue="dbscan_eps1_mpts3")
```



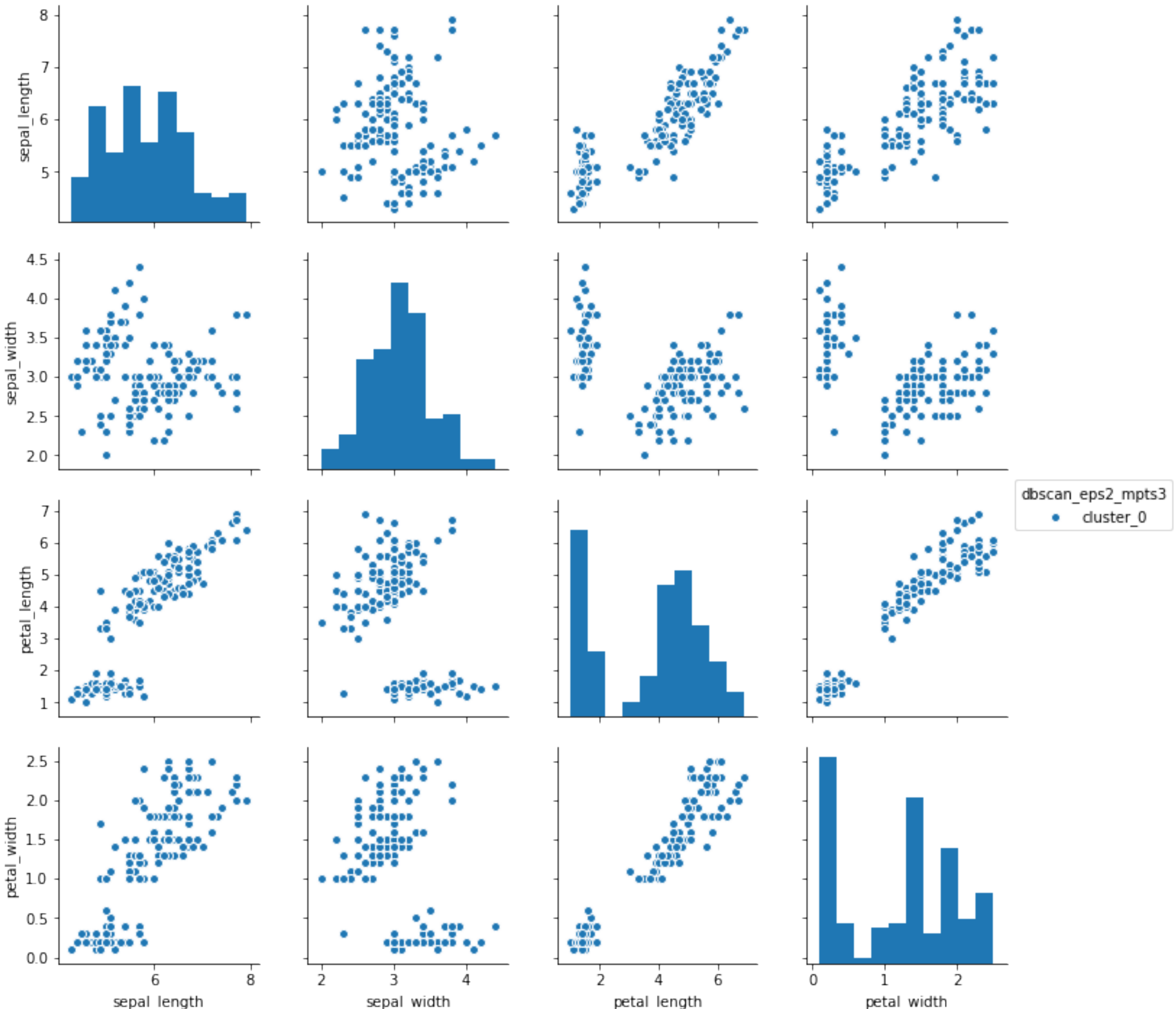
DBSCAN

```
db2 = DBSCAN(eps=1, min_samples=10)
db2.fit(data_scaled)
```



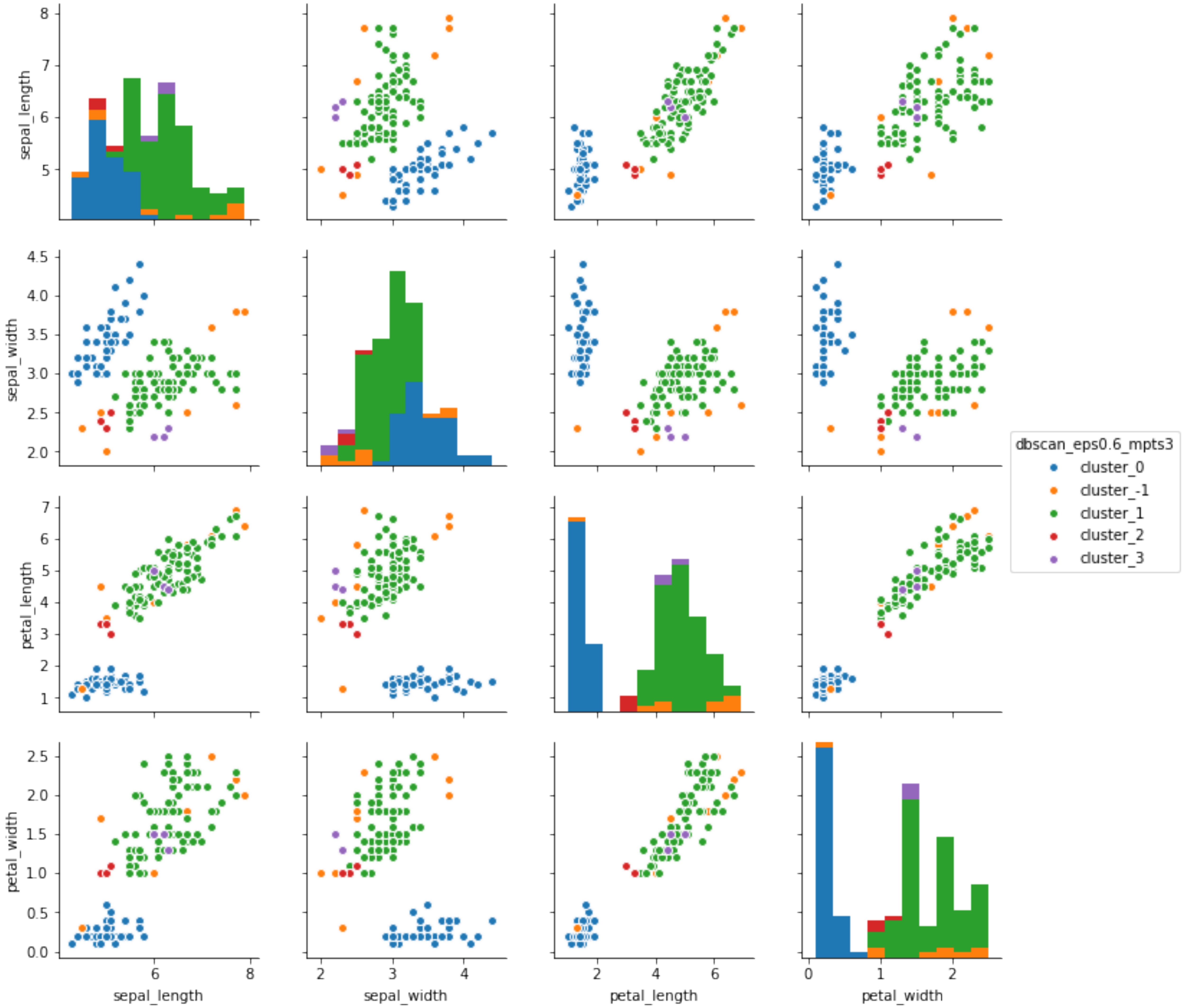
DBSCAN

```
db2 = DBSCAN(eps=2, min_samples=3)
db2.fit(iris_data_scaled)
```



DBSCAN

```
db2 = DBSCAN(eps=0.6, min_samples=3)
db2.fit(iris_data_scaled)
```



In Class Exercise

Please take 30 minutes and complete
Day 3: Clustering Worksheet

Questions?

Tuning Hyperparameters

Grid Search

```
RandomForestClassifier(bootstrap=True,
class_weight=None,
criterion='gini',
max_depth=None,
max_features='auto',
max_leaf_nodes=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1,
min_samples_split=2,
min_weight_fraction_leaf=0.0,
n_estimators=10,
n_jobs=1,
oob_score=False
)
```

Tuning these parameters

- GridSearchCV: You provide a list of possible parameters
- RandomizedSearchCV: Random combinations are searched

Grid Search

```
param_grid = [
   {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
   {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
'kernel': ['rbf']},
  ]
```

Grid Search

```
# use a full grid over all parameters
param_grid = {"max_depth": [3, None],
              "max_features": [1, 3, 10],
              "min_samples_split": [2, 3, 10],
              "min_samples_leaf": [1, 3, 10],
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}

# run grid search
grid_search = GridSearchCV(clf, param_grid=param_grid)
start = time()
grid_search.fit(X, y)

print("GridSearchCV took %.2f seconds for %d candidate parameter settings."
      % (time() - start, len(grid_search.cv_results_['params'])))
report(grid_search.cv_results_)
```

http://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html

Random Search

```
# specify parameters and distributions to sample from
param_dist = {"max_depth": [3, None],
              "max_features": sp_randint(1, 11),
              "min_samples_split": sp_randint(2, 11),
              "min_samples_leaf": sp_randint(1, 11),
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}

# run randomized search
n_iter_search = 20
random_search = RandomizedSearchCV(clf, param_distributions=param_dist,
                                   n_iter=n_iter_search)

start = time()
random_search.fit(X, y)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time() - start), n_iter_search))
report(random_search.cv_results_)
```

http://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html

What is a pipeline?

Why use a pipeline?

- It makes code more readable
- You don't have to worry about keeping track data during intermediate steps, for example between transforming and estimating.
- It makes it trivial to move ordering of the pipeline pieces, or to swap pieces in and out.
- It allows you to do GridSearchCV on your workflow

Without Pipeline

```
#get categorical features
#drop off last column because its unnecessary
X_categorical =
pd.get_dummies(df[categorical_columns]).astype(int).iloc[:,:-1]

#get and transform numeric features
X_numeric = df[numeric_columns]
X_numeric[numeric_columns] =
StandardScaler().fit_transform(X_numeric)

#get outcome variable
y = df[target]

#combine transformed categorical and numeric features
X_final = pd.concat((X_numeric,X_categorical),axis=1)
```

Without Pipeline

```
#create rf regressor and check 10-fold RMSE
rf = RandomForestRegressor()
cross_val_scores =
np.abs(cross_val_score(rf,X_final,y,scoring =
"neg_mean_squared_error", cv=10))
rmse_cross_val_scores = np.sqrt(cross_val_scores)
```

With Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScalar
from sklearn.neighbors import KNeighborsClassifier

# it takes a list of tuples as parameter
pipeline = Pipeline([
    ('scaler',StandardScaler()),
    ('clf', KNeighborsClassifier())
])

pipeline.fit(X_train,y_train)
```

With Pipeline

```
cross_val_scores =
np.abs(cross_val_score(full_pipeline,X,y,cv=10,scoring="n
eg_mean_squared_error"))
rmse_cross_val_scores = np.sqrt(cross_val_scores)
```

With Pipeline

```
full_pipeline.steps

[('all_features', FeatureUnion(n_jobs=1,
         transformer_list=[('categoricals', Pipeline(memory=None,
       steps=[('selector', ItemSelector(key=['rbc', 'pc', 'pcc', 'ba', 'htn',
'dm', 'cad', 'appet', 'pe', 'ane'])), ('imputer', Imputer(axis=0, copy=True,
missing_values=0, strategy='most_frequent',
      verbose=0)), ('encoder', OneHotEncoder(cat ... tegy='median', verbose=0)),
('scaler', StandardScaler(copy=True, with_mean=True, with_std=True))]))],
         transformer_weights=None)),
 ('rf_classifier',
  RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
              max_depth=None, max_features='auto', max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
              oob_score=False, random_state=None, verbose=0,
              warm_start=False))]
```

Pipelines

```
from sklearn.feature_selection import SelectKBest
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline

select = SelectKBest(k=100)
clf = RandomForestClassifier()

steps = [('feature_selection', select),
        ('random_forest', clf)]

pipeline = Pipeline(steps)
```

Pipelines

```
pipeline.fit( X_train, y_train )

y_prediction = pipeline.predict( X_test )

report = classification_report( y_test, y_prediction )

print(report)
```

In Class Exercise

Please take 30 minutes and complete

Day 3: Optimizing your Model

Pickling Your Model

- Pickling your model allows you to preserve your model to be used later.

Pickling Your Model

In order to rebuild a similar model with future versions of scikit-learn, additional metadata should be saved along the pickled model:

- The training data, e.g. a reference to a immutable snapshot
- The python source code used to generate the model
- The versions of scikit-learn and its dependencies
- The cross validation score obtained on the trai[illegible] data

Pickling Your Model

```
#Saving your model
from sklearn.externals import joblib
joblib.dump(clf, 'filename.pkl')

#Loading your model
clf = joblib.load('filename.pkl')
```

Case Studies in Machine Learning & Cyber Security

Finding SQL Injection Attempts

What is SQL injection?

SQL Injection is a vulnerability, most commonly in web applications in which an attacker can execute malicious queries on **YOUR** server

A normal SQL Query

```
SELECT *
FROM users
WHERE username=charles AND
password=pass1234
```

Pseudo Code for Web App Authentication

```
username = <from user>
password = <from user>

query = "SELECT * FROM users WHERE username =
username AND password = password"

query_result = db.execute(query)
if len( query_result > 0 )
   //Authenticate user
else
   //Boot them out
```

Pseudo Code for Web App Authentication

```
username = “charles”
password = “12345”   //Combination an idiot would use on their luggage
query = “SELECT *
FROM users
WHERE username = charles AND
password = 12345”

query_result = db.execute(query)
if len( query_result > 0 )
   //Authenticate user
else
   //Boot them out
```

Pseudo Code for Web App Authentication

```
username = “charles”
password = “12345 OR 1=1”   //Combination an idiot would use on their luggage
query = “SELECT *
FROM users
WHERE username = charles AND
password = 12345 OR 1=1”

query_result = db.execute(query)
if len( query_result > 0 )
   //Authenticate user
else
   //Boot them out
```

Legit vs. Malicious

```
SELECT count(category) FROM Product WHERE price >
'$20'
SELECT ctid, xmin, * FROM lockdemo
SELECT current_date FROM dual
```

Legit vs. Malicious

```
' or 'unusual' = 'unusual'

' or 'something' = 'some'+'thing'

' or 'text' = n'text'

' or 'something' like 'some%'
```

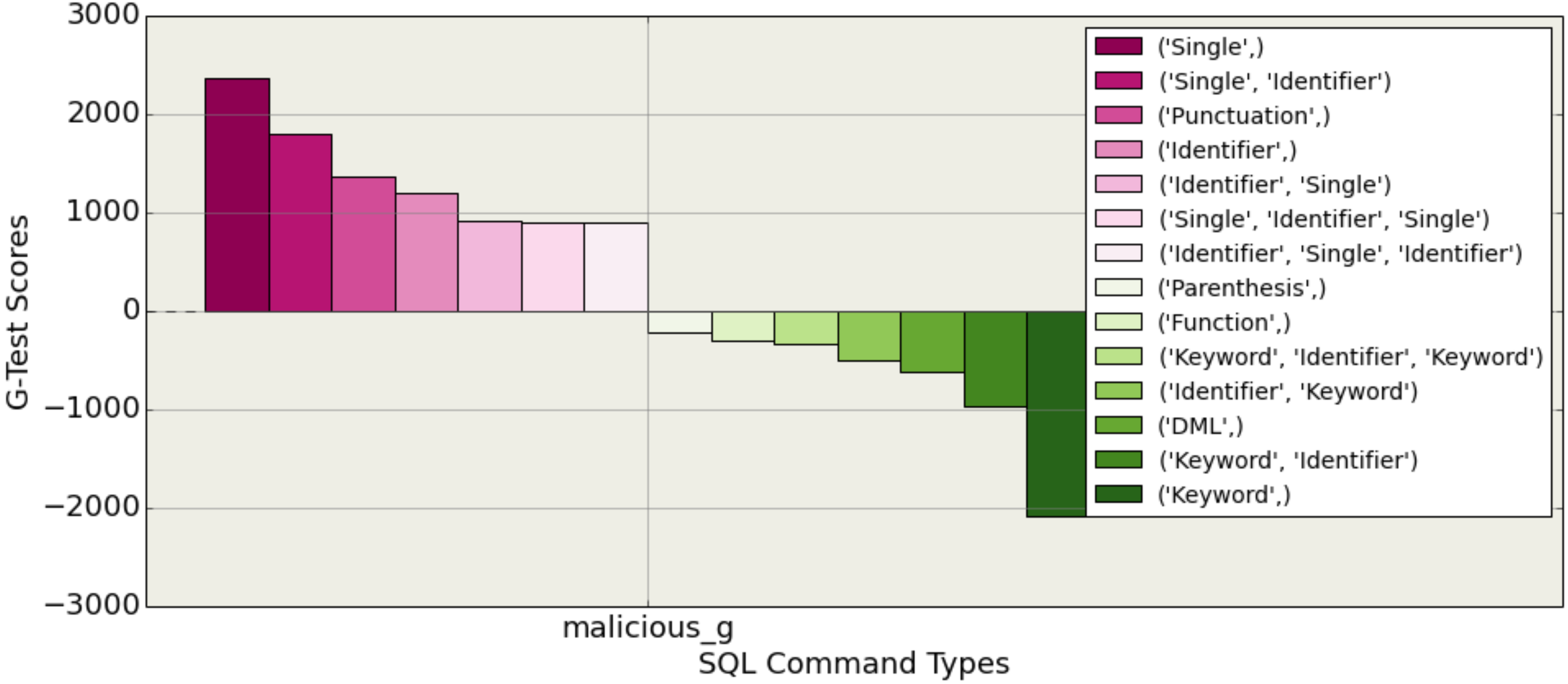
Step 1: Tokenize SQL

```
import sqlparse
def parse_it(raw_sql):
    parsed = sqlparse.parse(unicode(raw_sql,'utf-8'))
    return [token._get_repr_name() for parse in parsed for token in
parse.tokens if token._get_repr_name() != 'Whitespace']

dataframe['parsed_sql'] = dataframe['raw_sql'].map(lambda x: parse_it(x))
```

	raw_sql	type	parsed_sql
0	; exec master..xp_cmdshell 'ping	malicious	[Single, Identifier, Float, Float, Float,
1	create user name identified by	malicious	[DDL, Keyword, Identifier, Keyword,
2	create user name identified by pass123	malicious	[DDL, Keyword, Identifier, Keyword,
3	exec sp_addlogin 'name' , 'password'	malicious	[Keyword, Identifier, IdentifierList]
4	exec sp_addsrvrolemember 'name' ,	malicious	[Keyword, Identifier, IdentifierList]

https://github.com/ClickSecurity/data_hacking

Step 2: Create N-Grams



https://github.com/ClickSecurity/data_hacking

Step 2: Create N-Grams

	raw_sql	type	parsed_sql	sequences
0	; exec master..xp_cmdshell 'ping 10.10.1.2'--	malicious	[Single, Identifier, Float, Float, Float, Erro...	[('Single',), ('Identifier',), ('Float',), ('F...
44	anything' or 'x'='x	malicious	[Identifier, Single, Identifier, Single, Ident...	[('Identifier',), ('Single',), ('Identifier',)...
49	; exec master..xp_cmdshell 'ping aaa.bbb.ccc....	malicious	[Single, Identifier, Error, Single]	[('Single',), ('Identifier',), ('Error',), ('S...
54	; if not(select system_user) <> 'sa' waitfor ...	malicious	[Single, Identifier, Single, Integer, Placehol...	[('Single',), ('Identifier',), ('Single',), ('...
55	; if is_srvrolemember('sysadmin') > 0 waitfor...	malicious	[Single, Identifier, Single, Integer, Placehol...	[('Single',), ('Identifier',), ('Single',), ('...

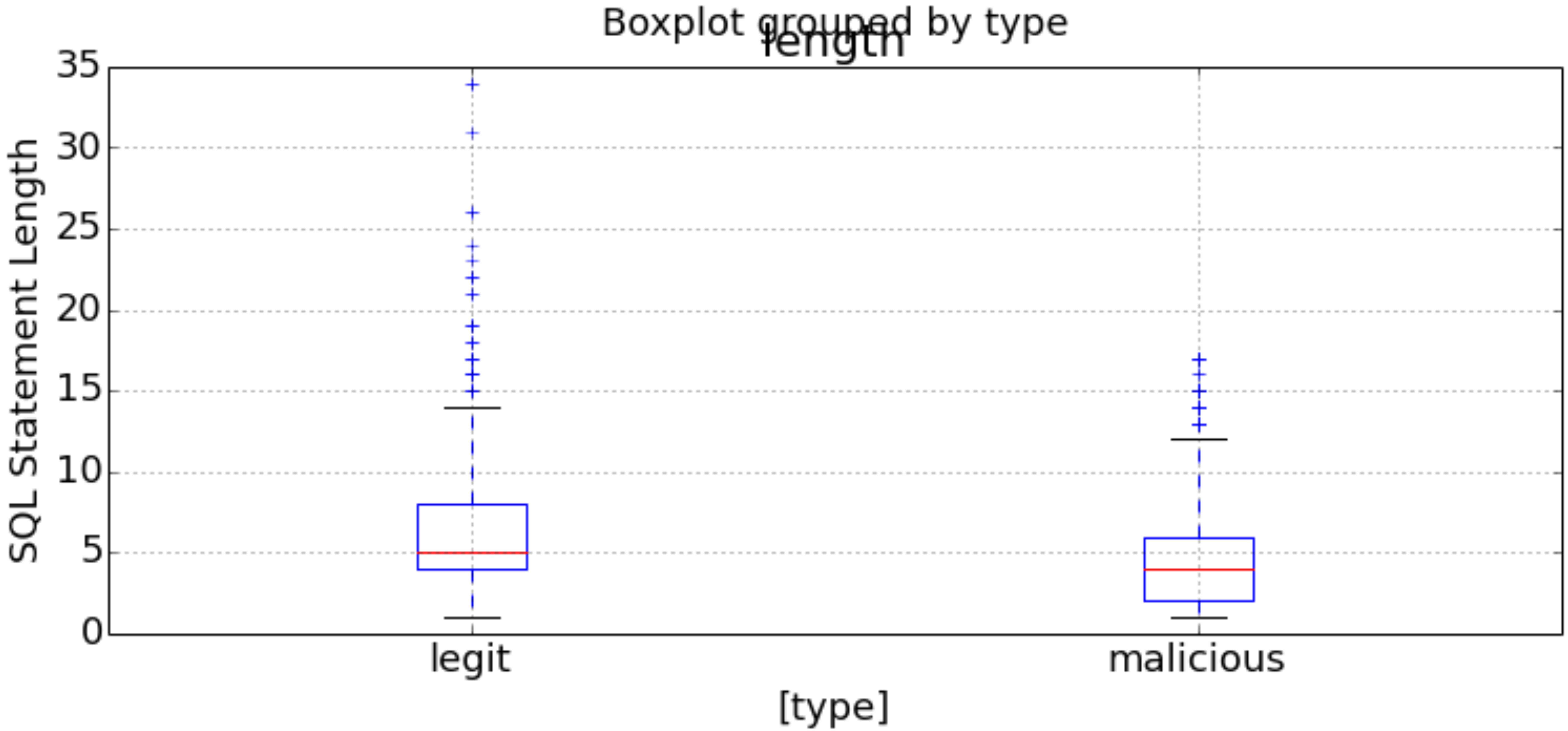
https://github.com/ClickSecurity/data_hacking

Step 3: Build Feature Vector

- Entropy
- Length
- G-Score Test

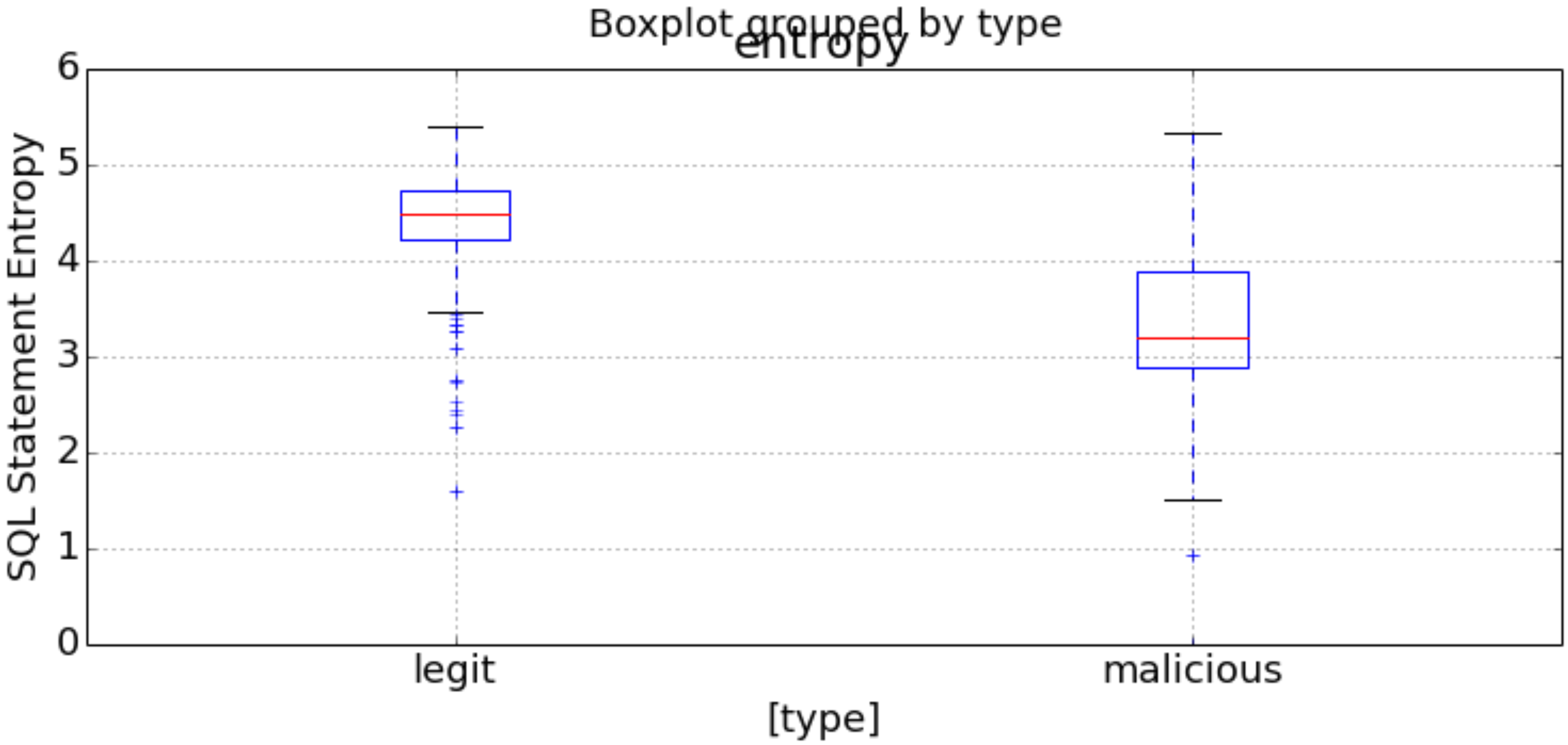
https://github.com/ClickSecurity/data_hacking

Step 3: Build Feature Vector



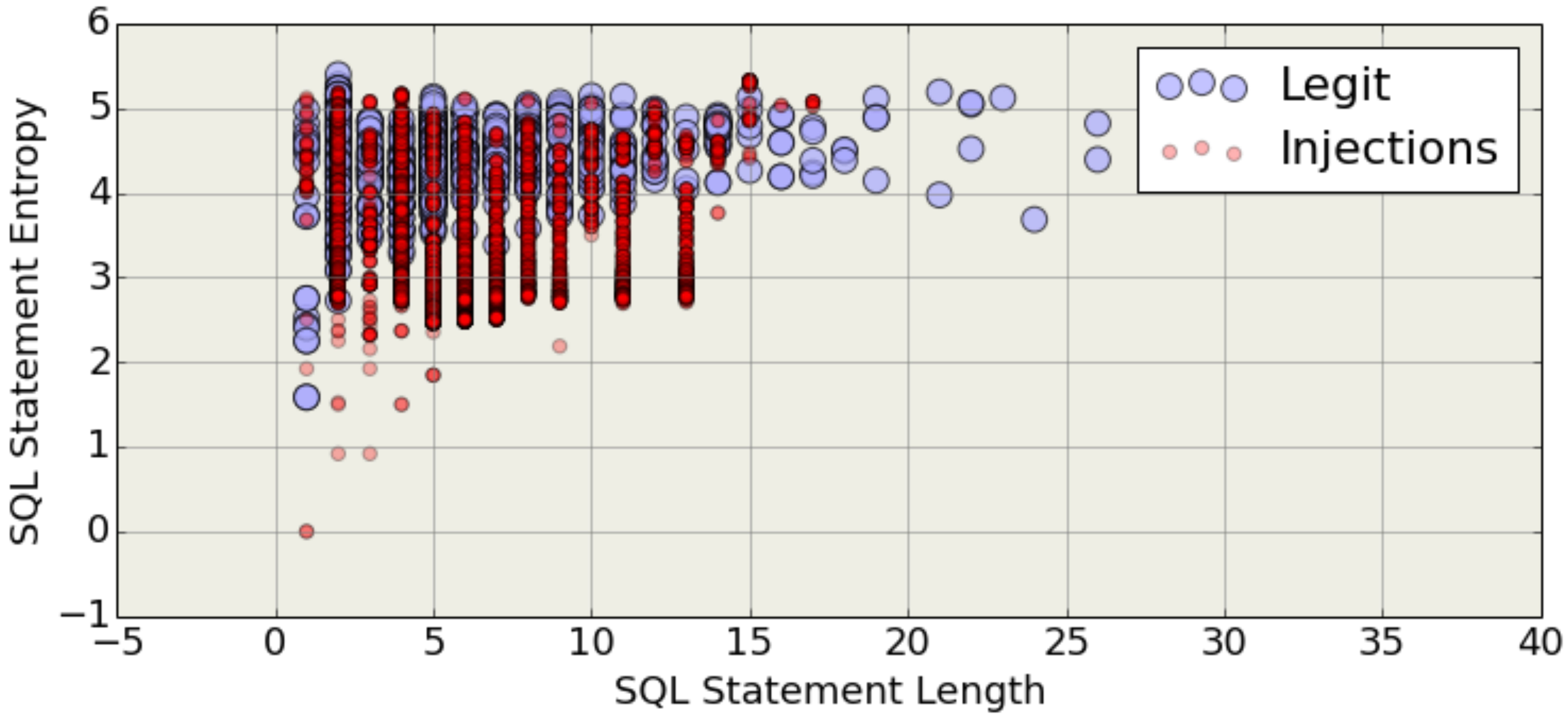
https://github.com/ClickSecurity/data_hacking

Step 3: Build Feature Vector



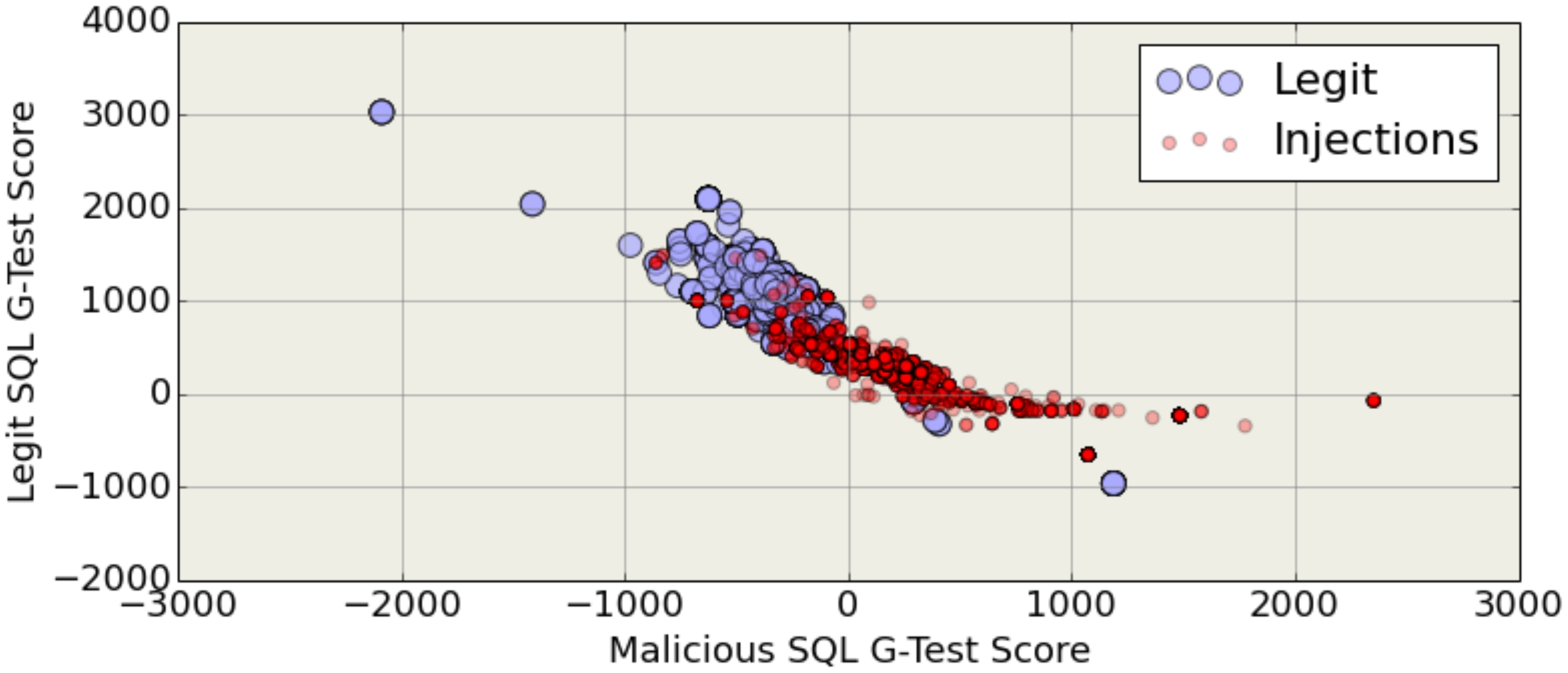
https://github.com/ClickSecurity/data_hacking

Step 3: Build Feature Vector



https://github.com/ClickSecurity/data_hacking

Step 3: Build Feature Vector



https://github.com/ClickSecurity/data_hacking

Step 4: Train Classifier

```
import sklearn.ensemble
clf =
sklearn.ensemble.RandomForestClassifier(n_estimators=20)

scores = sklearn.cross_validation.cross_val_score(clf, X, y,
cv=10, n_jobs=4)
print scores
```


https://github.com/ClickSecurity/data_hacking

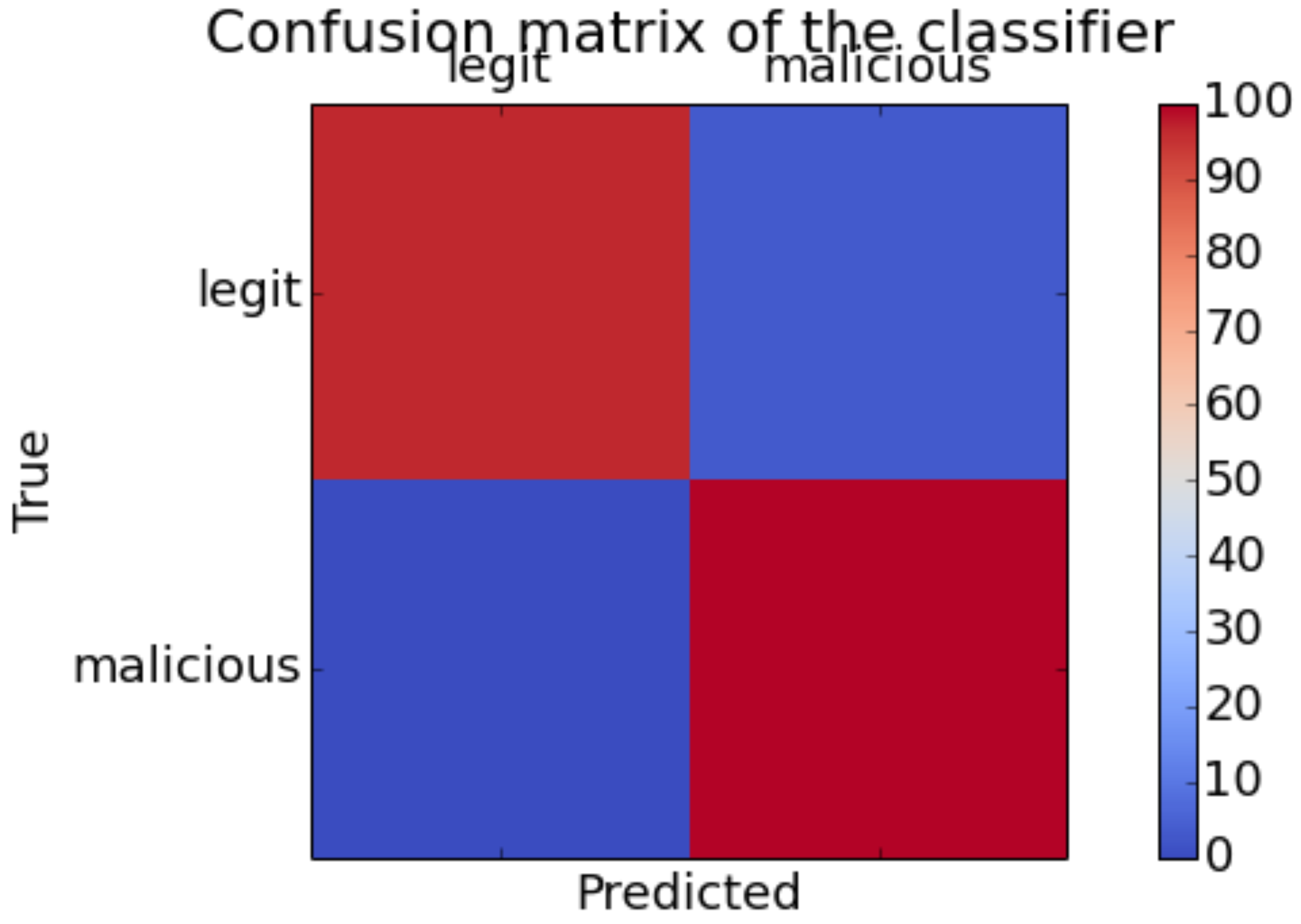
Step 5: Evaluate Model

```
scores = sklearn.cross_validation.cross_val_score(clf, X, y,
cv=10, n_jobs=4)
```

```
[ 0.99784173  0.99784173  1.          0.99784173
0.99856115  0.99784017
  0.99640029  0.99856012  0.99784017  0.99784017]
```

https://github.com/ClickSecurity/data_hacking

Step 5: Evaluate Model




```
Confusion Matrix Stats
legit/legit: 96.59% (170/176)
legit/malicious: 3.41% (6/176)
malicious/legit: 0.08% (2/2603)
malicious/malicious: 99.92% (2601/2603)
```

https://github.com/ClickSecurity/data_hacking

Thank you!