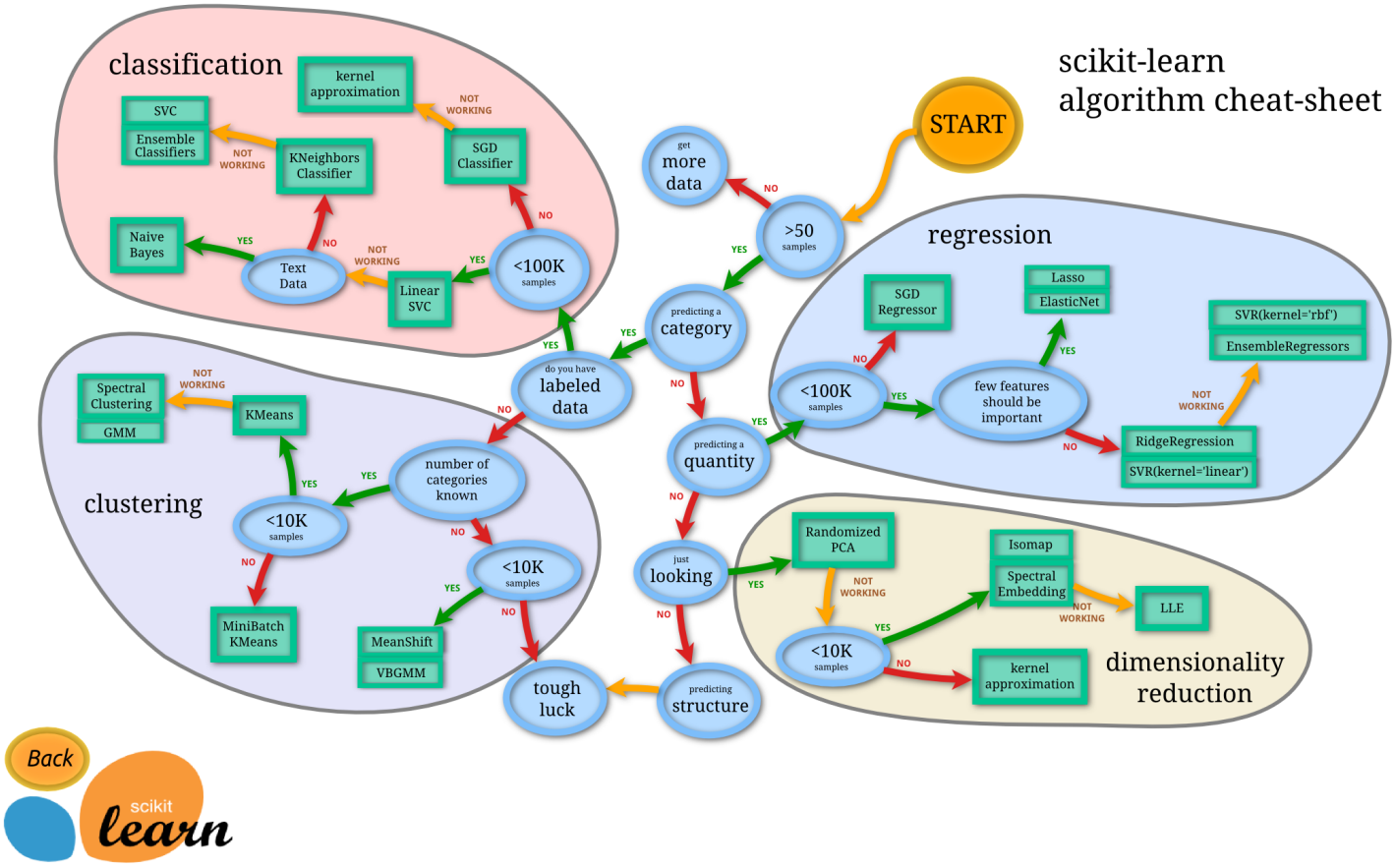
Unsupervised Learning



Supervised Learning

- Dataset = X, y
- X = feature matrix (n, p)
- y = targets vector $(n, 1)$

> Find $h_\beta(X)$ as close to y as possible

Unsupervised Algorithms

> find patterns in X, **without supervision from a target** y

Unsupervised Learning helps us **reduce dimensions**

- Feature Engineering/less features (saves time)
- Compress (saves space)

It also allows us to **cluster data** (= group data points based on similarities)

- Understand data (explore, visualize, etc.)
- Find anomalies/outliers?
- Recommendations
- Semi-supervised classifications

Plan

1. Principal Component Analysis (PCA)
2. Clustering with K-Means Algorithm

1. Principal Component Analysis (PCA)

- Squashes our high-dimensional dataset down into a lower dimension
- Aims to find the best linear combination of features (= columns) that best represents the underlying structure of the data

Remember Linear Regression Variants?

Polynomial
$\hat{y}$ =

Log transformation
$\hat{y}$ =

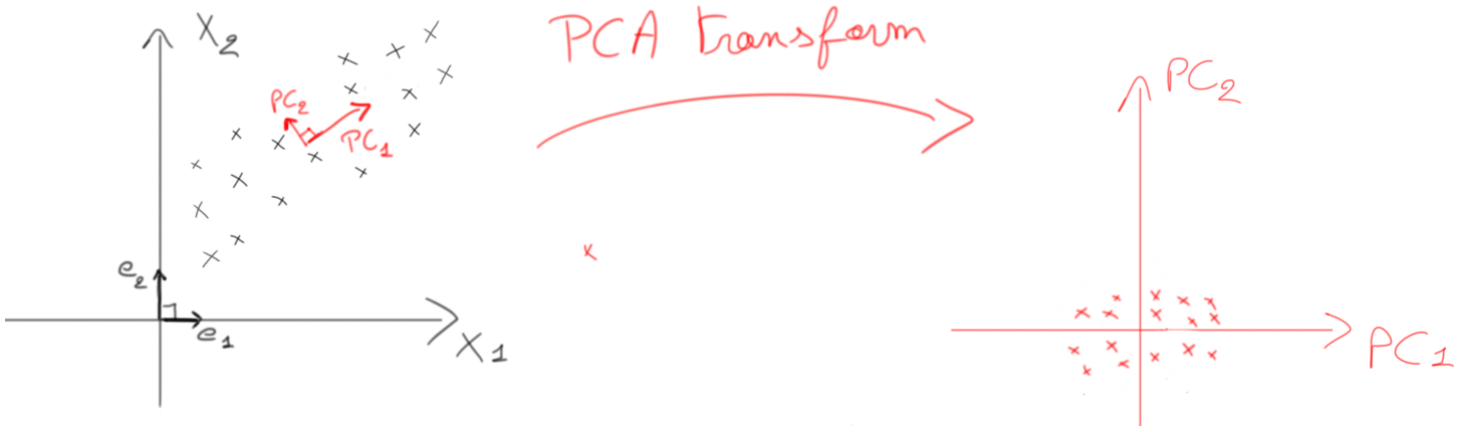
Linear combination of features (while avoiding multicollinearity)
$\hat{y}$ =

👉 PCA = finding the **best linear combination** of features

$$Z_1 = a_{11}X_1 + a_{12}X_2 + a_{13}X_3 \quad Z_2 = a_{21}X_1 + a_{22}X_2 + a_{23}X_3 \quad Z_3 = a_{31}X_1 + a_{32}X_2 + a_{33}X_3$$

- Canceling **ALL** multicollinearity
- **Ranking** the newly created PCs Z from most to least important

Z_i is a so-called Principal Component (PC)



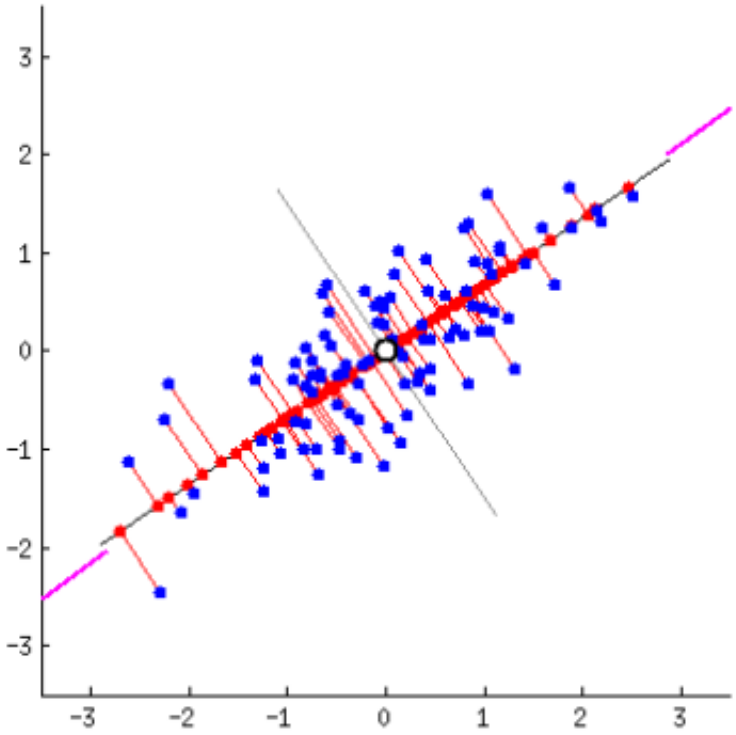
PCA is essentially a projection of the data that is

- oriented towards specific directions, defined by the **Principal Components**
- orthonormal to each other (0 multicollinearity)
- ranked by decreasing "explaining power"
  (measured by the variance of our data when projected onto this PC)

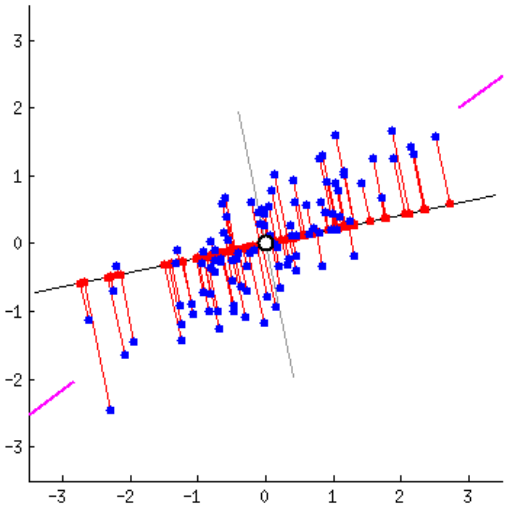
Intuition

If we had to keep **only one direction** to describe our data, this direction should

- **preserve** most of the **variance** in the data when projected onto it (see spread of red dots)
- minimize "reconstruction errors" (see red lines)



👉 stats-exchange story (https://stats.stackexchange.com/a/140579/286995)

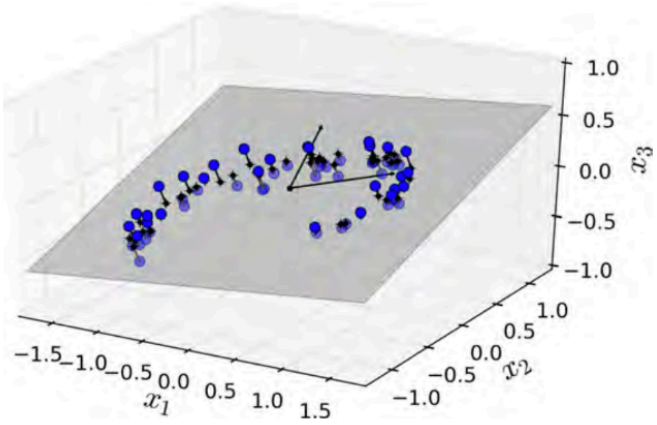


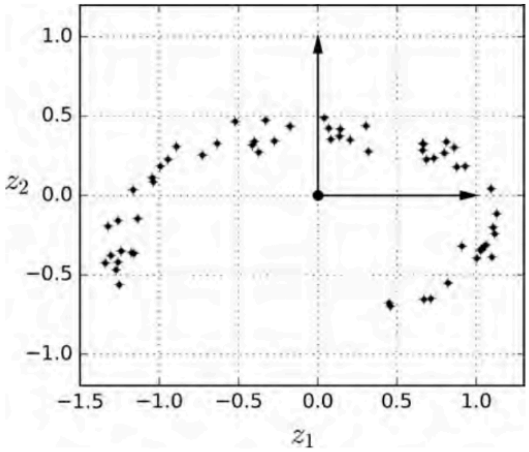
3 components

👉 interactive visuals (http://setosa.io/ev/principal-component-analysis/)

👉 **PCA helps to reduce dimensions!**

$(X_1, X_2, X_3) \sim (Z_1, Z_2)$





📚 Hands-On Machine Learning (https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/)

1.2 Let's Code an Example (with a Wine Dataset)

In [ ]:
```
from sklearn.datasets import load_wine

wine = load_wine(as_frame=True)
X = wine.data
y = wine.target
wine_features = X.columns

#⚠️ Data must be centered around its mean before applying PCA ⚠️
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = pd.DataFrame(scaler.transform(X), columns=wine_features)
X
```

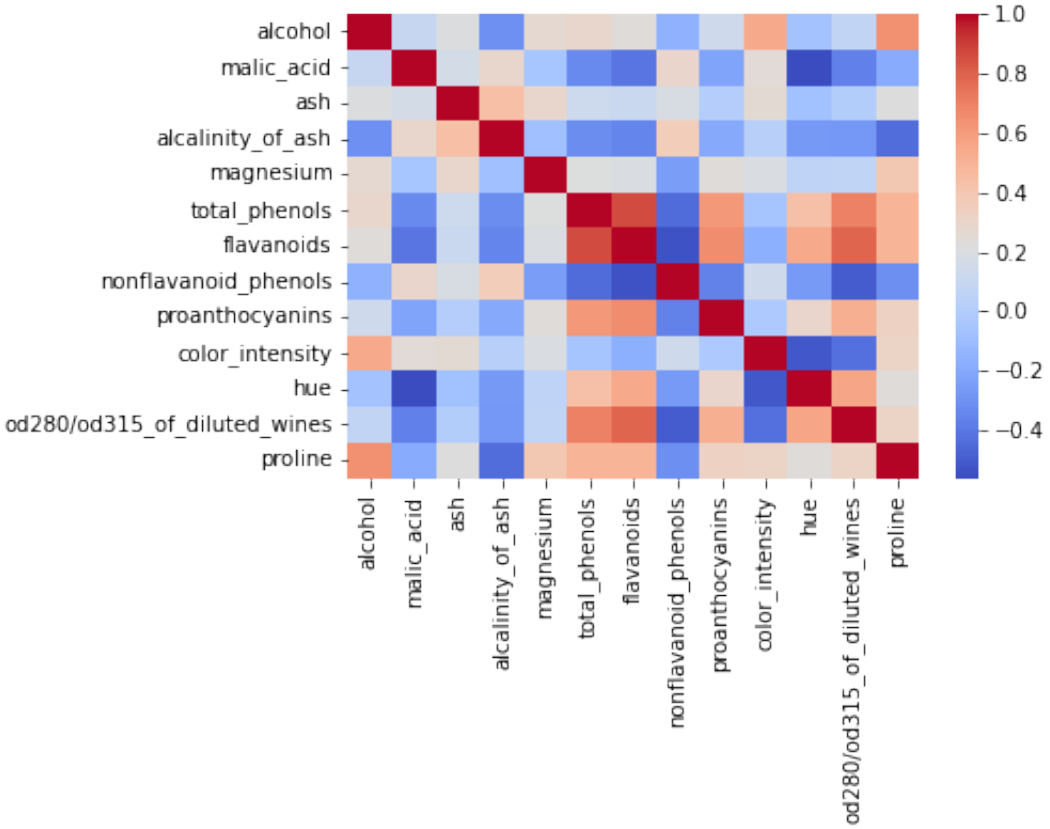
Out[ ]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids
0	1.518613	-0.562250	0.232053	-1.169593	1.913905	0.808997	1.034819
1	0.246290	-0.499413	-0.827996	-2.490847	0.018145	0.568648	0.733629
2	0.196879	0.021231	1.109334	-0.268738	0.088358	0.808997	1.215533
3	1.691550	-0.346811	0.487926	-0.809251	0.930918	2.491446	1.466525
4	0.295700	0.227694	1.840403	0.451946	1.281985	0.808997	0.663351
...	...	...	...	...	...	...	...
173	0.876275	2.974543	0.305159	0.301803	-0.332922	-0.985614	-1.424900
174	0.493343	1.412609	0.414820	1.052516	0.158572	-0.793334	-1.284344
175	0.332758	1.744744	-0.389355	0.151661	1.422412	-1.129824	-1.344582
176	0.209232	0.227694	0.012732	0.151661	1.422412	-1.033684	-1.354622
177	1.395086	1.583165	1.365208	1.502943	-0.262708	-0.392751	-1.274305

178 rows × 13 columns

In [ ]:
```
sns.heatmap(pd.DataFrame(X).corr(), cmap='coolwarm')
```

Out[ ]: <AxesSubplot:>



a) Compute the Principal Components

In [ ]:
```
from sklearn.decomposition import PCA

pca = PCA()
pca.fit(X)
```

Out[ ]: PCA()

In [ ]:
```
# Access our 13 PCs
W = pca.components_

# Print PCs as COLUMNS
W = pd.DataFrame(W.T,
                 index=wine_features,
                 columns=[f'PC{i}' for i in range(1, 14)])
W
```

Out[ ]:

	PC1	PC2	PC3	PC4	PC5	PC6
alcohol	0.144329	-0.483652	-0.207383	-0.017856	-0.265664	-0.213539
malic_acid	-0.245188	-0.224931	0.089013	0.536890	0.035214	-0.536814
ash	-0.002051	-0.316069	0.626224	-0.214176	-0.143025	-0.154475
alcalinity_of_ash	-0.239320	0.010591	0.612080	0.060859	0.066103	0.100825
magnesium	0.141992	-0.299634	0.130757	-0.351797	0.727049	-0.038144
total_phenols	0.394661	-0.065040	0.146179	0.198068	-0.149318	0.084122
flavanoids	0.422934	0.003360	0.150682	0.152295	-0.109026	0.018920
nonflavanoid_phenols	-0.298533	-0.028779	0.170368	-0.203301	-0.500703	0.258594
proanthocyanins	0.313429	-0.039302	0.149454	0.399057	0.136860	0.533795
color_intensity	-0.088617	-0.529996	-0.137306	0.065926	-0.076437	0.418644
hue	0.296715	0.279235	0.085222	-0.427771	-0.173615	-0.105983
od280/od315_of_diluted_wines	0.376167	0.164496	0.166005	0.184121	-0.101161	-0.265851
proline	0.286752	-0.364903	-0.126746	-0.232071	-0.157869	-0.119726

☝️ Each PC is a **linear combination of initial wine features**

b) *Project* our dataset into this new space of PCs

In [ ]:
```
X_proj = pca.transform(X)
X_proj = pd.DataFrame(X_proj, columns=[f'PC{i}' for i in range(1, 14)])
X_proj
```

Out[ ]:

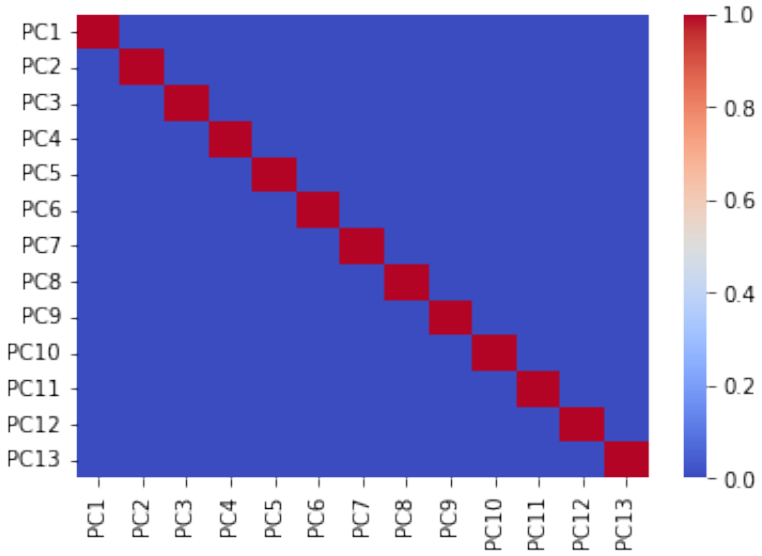
	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	
0	3.316751	-1.443463	-0.165739	-0.215631	0.693043	-0.223880	0.596427	0.065139	0.
1	2.209465	0.333393	-2.026457	-0.291358	-0.257655	-0.927120	0.053776	1.024416	-0.
2	2.516740	-1.031151	0.982819	0.724902	-0.251033	0.549276	0.424205	-0.344216	-1.
3	3.757066	-2.756372	-0.176192	0.567983	-0.311842	0.114431	-0.383337	0.643593	0.
4	1.008908	-0.869831	2.026688	-0.409766	0.298458	-0.406520	0.444074	0.416700	0.
...	...	...	...	...	...	...	...	...	
173	-3.370524	-2.216289	-0.342570	1.058527	-0.574164	-1.108788	0.958416	-0.146097	-0.
174	-2.601956	-1.757229	0.207581	0.349496	0.255063	-0.026465	0.146894	-0.552427	-0.
175	-2.677839	-2.760899	-0.940942	0.312035	1.271355	0.273068	0.679235	0.047024	0.
176	-2.387017	-2.297347	-0.550696	-0.688285	0.813955	1.178783	0.633975	0.390829	0.
177	-3.208758	-2.768920	1.013914	0.596903	-0.895193	0.296092	0.005741	-0.292914	0.

178 rows × 13 columns

☝️ 178 wine bottles, each expressed as a linear combination of 13 Principal Components

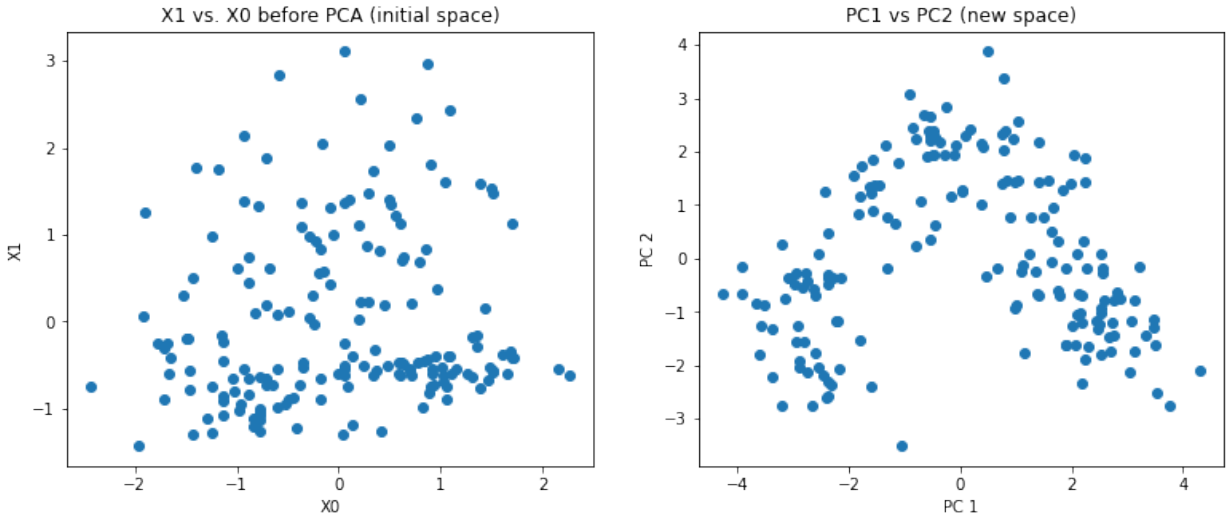
✅ As expected, the PCA reduces multicollinearity to the absolute minimum (0)!

In [ ]:
```
sns.heatmap(X_proj.corr(), cmap='coolwarm');
```

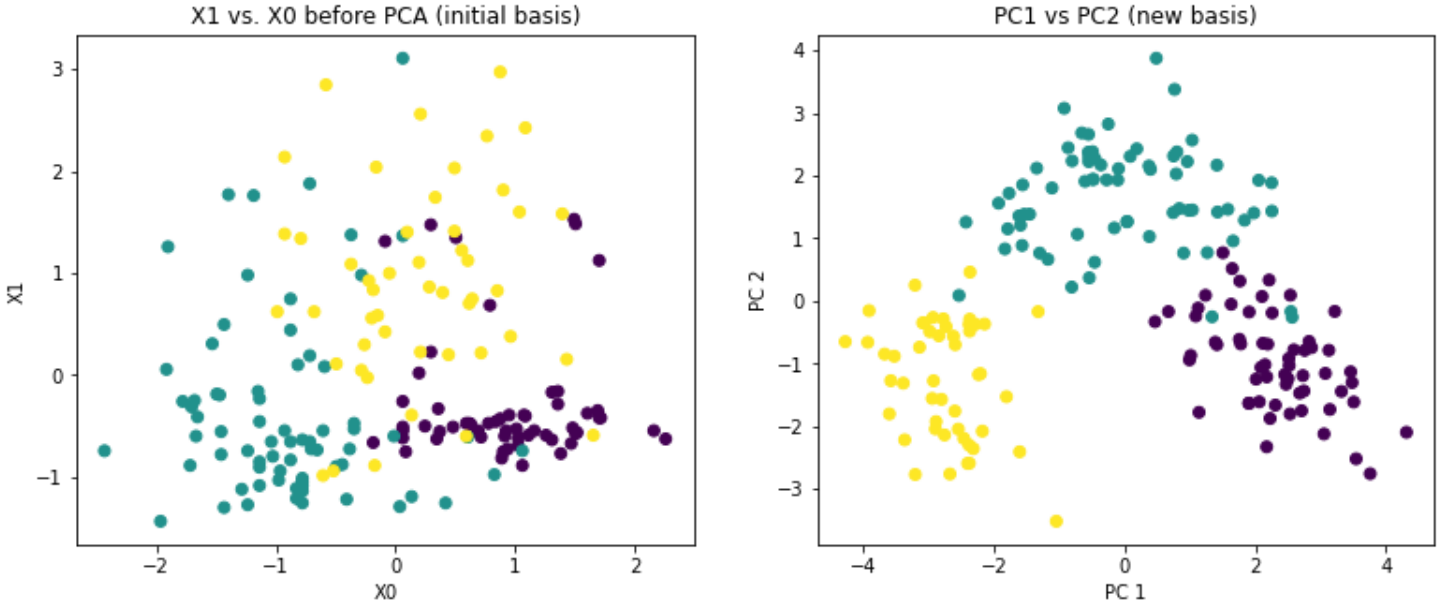


✅ Our wine dataset is also easier to observe in this new space

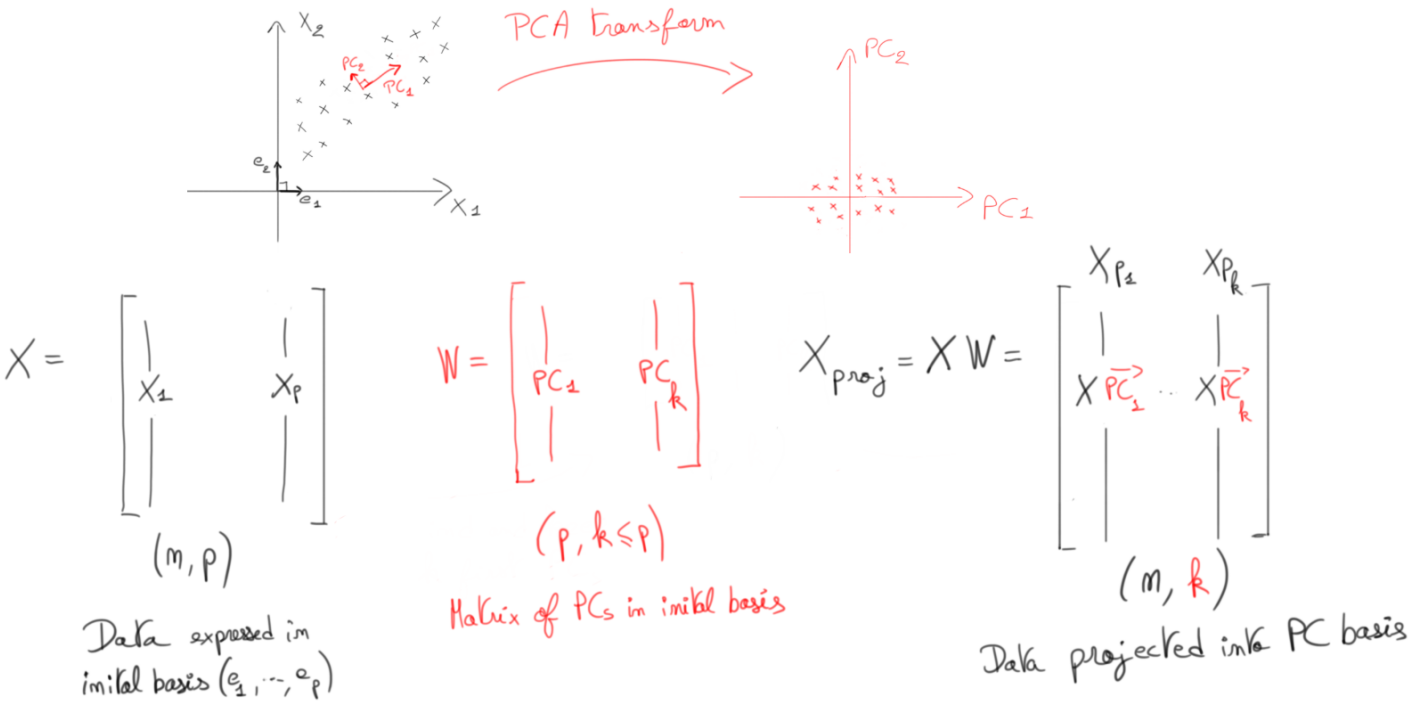
In [ ]:
```
# 2D-slice

plt.figure(figsize=(13,5))
plt.subplot(1,2,1)
plt.title('X1 vs. X0 before PCA (initial space)'); plt.xlabel('X0'); p
lt.ylabel('X1')
plt.scatter(X.iloc[:,0], X.iloc[:,1])

plt.subplot(1,2,2)
plt.title('PC1 vs PC2 (new space)'); plt.xlabel('PC 1'); plt.ylabel('P
C 2')
plt.scatter(X_proj.iloc[:,0], X_proj.iloc[:,1]);
```



Adding true labels makes it even clearer



💡 **"Projecting" data onto a new space is a simple matrix multiplication**



In [ ]:
```
# Computational proof
W = pca.components_.T
print("Shape of W: ", W.shape)
print("Shape of X", X.shape)
```

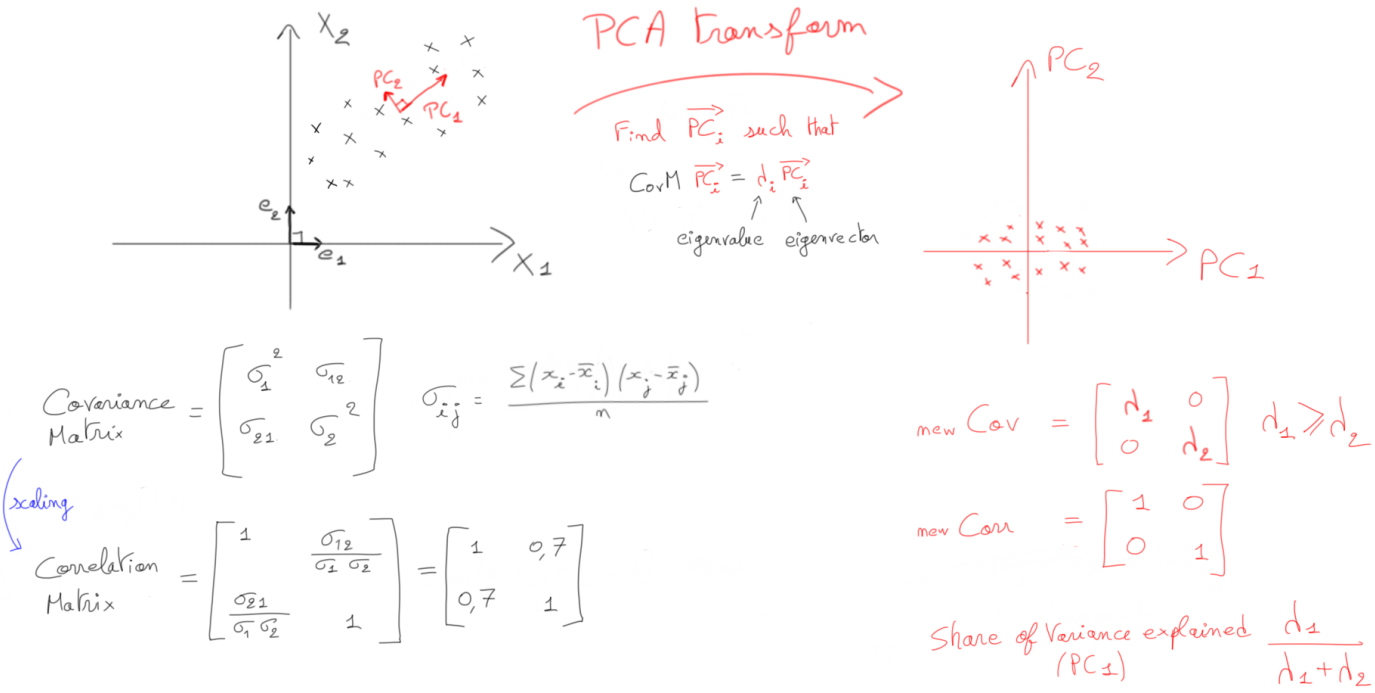
```
Shape of W:  (13, 13)
Shape of X (178, 13)
```

In [ ]:
```
np.allclose(
    pca.transform(X),
    np.dot(X,W)
)
```

Out[ ]:
```
True
```

1.3 How are Principal Components Computed (Mathematically)?

This is the hard part



We can do it with `NumPy`

- `np.linalg.eig(M)` computes the `eig_vals` and `eig_vecs` of M
- Covariance Matrix = $X^T X$ of shape `(p,p)` (if features are centered)

❗ `eig()` decomposition can take very long

ℹ️ Eigenvalues and Eigenvectors (https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors)

In [ ]:
```
# Compute PCs
eig_vals, eig_vecs = np.linalg.eig(np.dot(X.T,X))
```

In [ ]:
```
# Show all 13 principal components (unranked)
W = pd.DataFrame(eig_vecs,
                 index=wine_features,
                 columns=[f'PC{i}' for i in range(1, 14)])
W
```

Out[ ]:

	PC1	PC2	PC3	PC4	PC5	PC6
alcohol	-0.144329	0.483652	-0.207383	0.017856	-0.265664	0.213539
malic_acid	0.245188	0.224931	0.089013	-0.536890	0.035214	0.536814
ash	0.002051	0.316069	0.626224	0.214176	-0.143025	0.154475
alcalinity_of_ash	0.239320	-0.010591	0.612080	-0.060859	0.066103	-0.100825
magnesium	-0.141992	0.299634	0.130757	0.351797	0.727049	0.038144
total_phenols	-0.394661	0.065040	0.146179	-0.198068	-0.149318	-0.084122
flavanoids	-0.422934	-0.003360	0.150682	-0.152295	-0.109026	-0.018920
nonflavanoid_phenols	0.298533	0.028779	0.170368	0.203301	-0.500703	-0.258594
proanthocyanins	-0.313429	0.039302	0.149454	-0.399057	0.136860	-0.533795
color_intensity	0.088617	0.529996	-0.137306	-0.065926	-0.076437	-0.418644
hue	-0.296715	-0.279235	0.085222	0.427771	-0.173615	0.105983
od280/od315_of_diluted_wines	-0.376167	-0.164496	0.166005	-0.184121	-0.101161	0.265851
proline	-0.286752	0.364903	-0.126746	0.232071	-0.157869	0.119726

1.4 PCs are ranked by order of importance

PCs
are ranked by share of **explained variance**
$\frac{Var(PC_i)}{Var(X)}$

❗ Remember: information comes in the form of variation
❗ PC with most variance is the most important one

In [ ]:
```
# Let's compute it
X_proj.std()**2 / ((X.std()**2).sum())
```

Out[ ]:
```
PC1     0.361988
PC2     0.192075
PC3     0.111236
PC4     0.070690
PC5     0.065633
PC6     0.049358
PC7     0.042387
PC8     0.026807
PC9     0.022222
PC10    0.019300
PC11    0.017368
PC12    0.012982
PC13    0.007952
dtype: float64
```

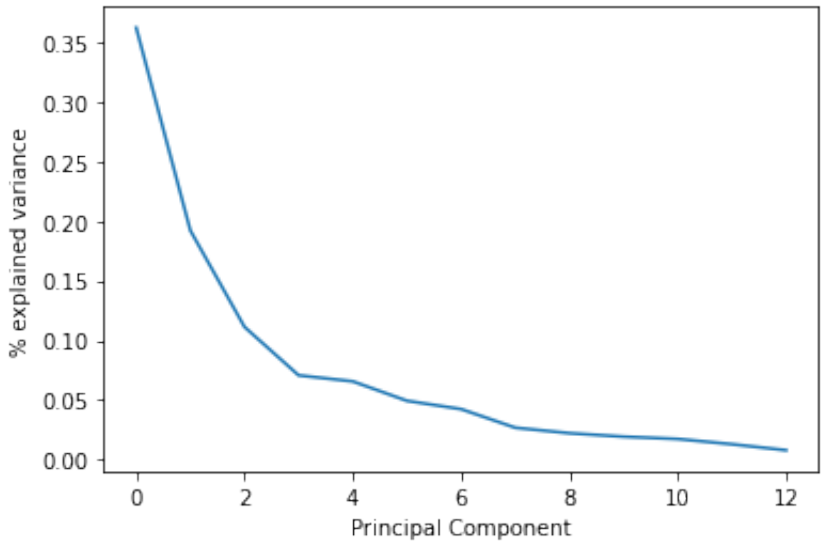
☝ scikit-learn `PCA` has indeed already ranked them

> 36% of the dataset's variance lies along the first axis

In [ ]:
```
# Sklearn provides it automatically
pca.explained_variance_ratio_
```

Out[ ]:
```
array([0.36198848, 0.1920749 , 0.11123631, 0.0706903 , 0.06563294,
       0.04935823, 0.04238679, 0.02680749, 0.02222153, 0.01930019,
       0.01736836, 0.01298233, 0.00795215])
```

In [ ]:
```
plt.plot(pca.explained_variance_ratio_)
plt.xlabel('Principal Component'); plt.ylabel('% explained variance');
```



> PCA redistributes the ratio among the new features in the **most unequal way**

1.5 PCA for Dimensionality Reduction

👉 Having computed all PCs, we can now keep only the `k` most important ones!

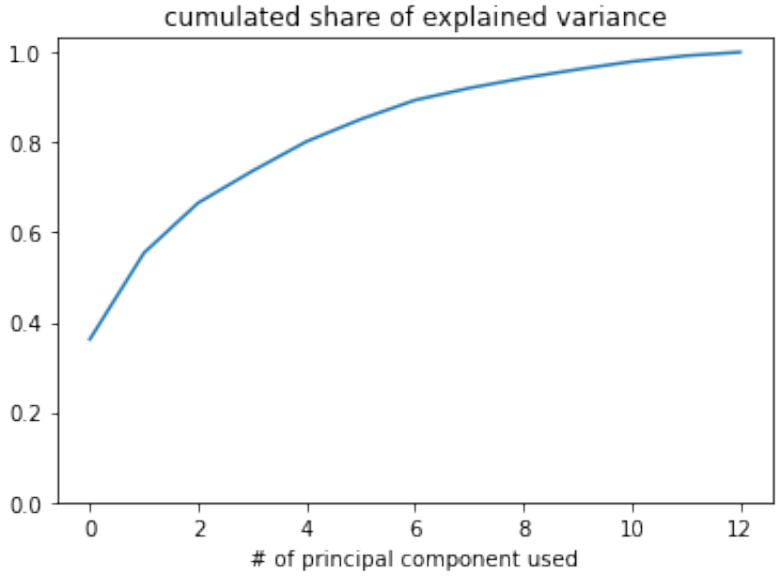
❓ Why would we want less features? Because it means we can

- compress data
- reduce model complexity & fit time
- reduce overfitting

How to choose `k` ?

It's a **trade-off** between compression and performance

In [ ]:
```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.ylim(ymin=0)
plt.title('cumulated share of explained variance')
plt.xlabel('# of principal component used');
```

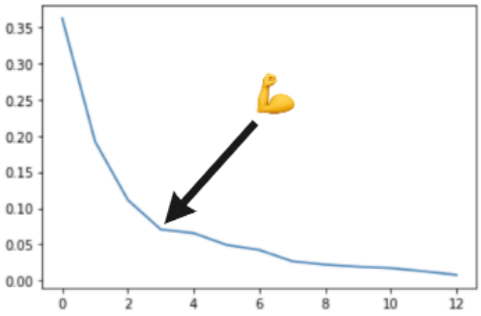


The Elbow Method

Look for the *inflection point* in the explained variance chart

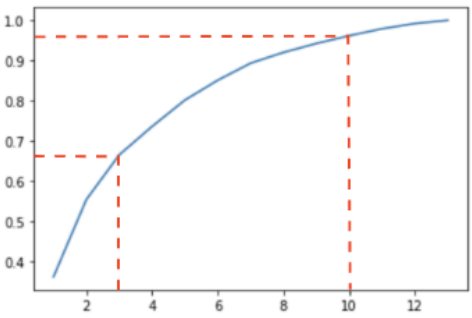
Here, `k=3` looks promising

Explained variance by dimension



Explained variance cumulative sum

```
Number of principal components 1, amount of explained variance held- 0.36
Number of principal components 2, amount of explained variance held- 0.55
Number of principal components 3, amount of explained variance held- 0.67
Number of principal components 4, amount of explained variance held- 0.74
Number of principal components 5, amount of explained variance held- 0.8
Number of principal components 6, amount of explained variance held- 0.85
Number of principal components 7, amount of explained variance held- 0.89
Number of principal components 8, amount of explained variance held- 0.92
Number of principal components 9, amount of explained variance held- 0.94
Number of principal components 10, amount of explained variance held- 0.96
Number of principal components 11, amount of explained variance held- 0.98
Number of principal components 12, amount of explained variance held- 0.99
Number of principal components 13, amount of explained variance held- 1.0
```



✏️ Test Model Performance (with `k=3` Dimensions)

In [ ]:
```
# Fit a PCA with only 3 components
pca3 = PCA(n_components=3).fit(X)

# Project your data into 3 dimensions
X_proj3 = pd.DataFrame(pca3.fit_transform(X), columns=['PC1', 'PC2', '
PC3'])

# We have "compressed" our dataset in 3D
X_proj3
```

Out[ ]:

	PC1	PC2	PC3
0	3.316751	-1.443463	-0.165739
1	2.209465	0.333393	-2.026457
2	2.516740	-1.031151	0.982819
3	3.757066	-2.756372	-0.176192
4	1.008908	-0.869831	2.026688
...	...	...	...
173	-3.370524	-2.216289	-0.342570
174	-2.601956	-1.757229	0.207581
175	-2.677839	-2.760899	-0.940942
176	-2.387017	-2.297347	-0.550696
177	-3.208758	-2.768920	1.013914

178 rows × 3 columns

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

print("accuracy 3 PCs")
print(cross_val_score(LogisticRegression(), X_proj3, y, cv=5).mean())

print("\naccuracy all 13 initial features")
print(cross_val_score(LogisticRegression(), X, y, cv=5).mean())
```

```
accuracy 3 PCs
0.9609523809523809

accuracy all 13 initial features
0.9888888888888889
```

Decompress

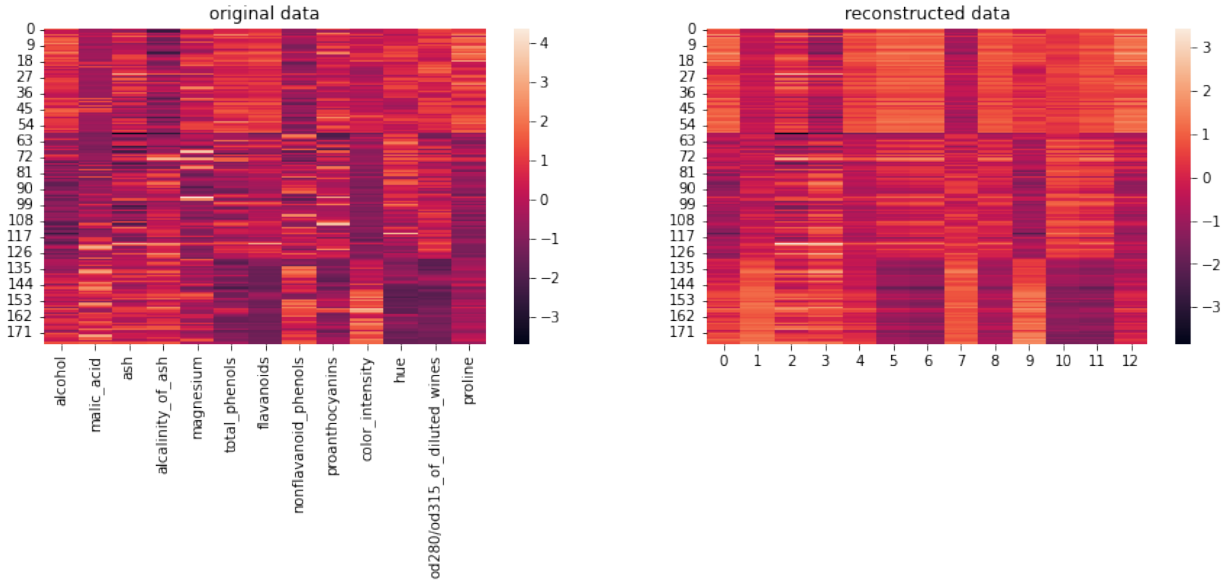
❓ Can you **perfectly** reconstruct `X` from `X_proj3` ?

- Not if you kept `k < 13` dimensions; information has been lost
- We can *approximate* `X` by reconstructing it with `inverse_transform()`

In [ ]:
```
X_reconstructed = pca3.inverse_transform(X_proj3)
X_reconstructed.shape
```

Out[ ]:
```
(178, 13)
```

In [ ]:
```
plt.figure(figsize=(15,4))
plt.subplot(1,2,1)
sns.heatmap(X)
plt.title("original data")
plt.subplot(1,2,2)
plt.title("reconstructed data")
sns.heatmap(X_reconstructed);
```

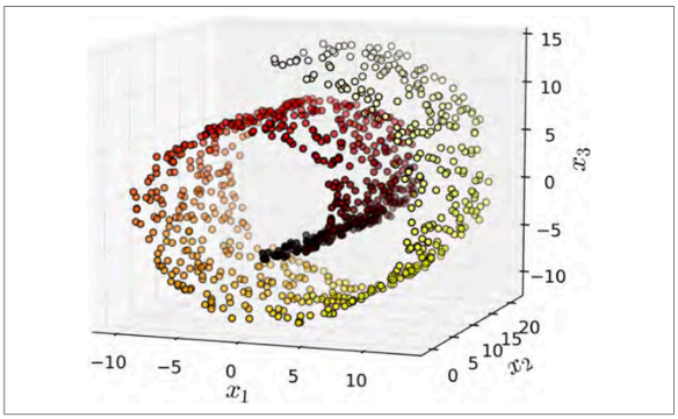


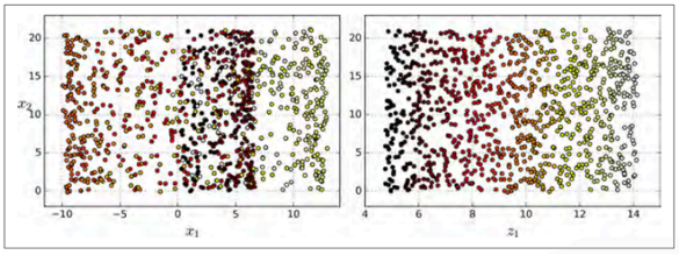
1.6 Limitations of PCA

Watch out for **manifolds** 👀

A manifold is an N-dimensional shape that can be bent and twisted into a higher dimensional shape 🌀

Below we can see our data distribution before and after PCA has been applied





📚 Hands-On Machine Learning (https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/)

Other dimensionality reduction techniques:

- **t-Distributed Stochastic Neighbor Embedding (t-SNE)** — Aims to reduce dimensionality while keeping similar observations close together and dissimilar ones apart. This is a great technique for visualizing clusters of higher dimensions
- **Kernel PCA** — Captures non-linear patterns (similar principle to SVM kernels)

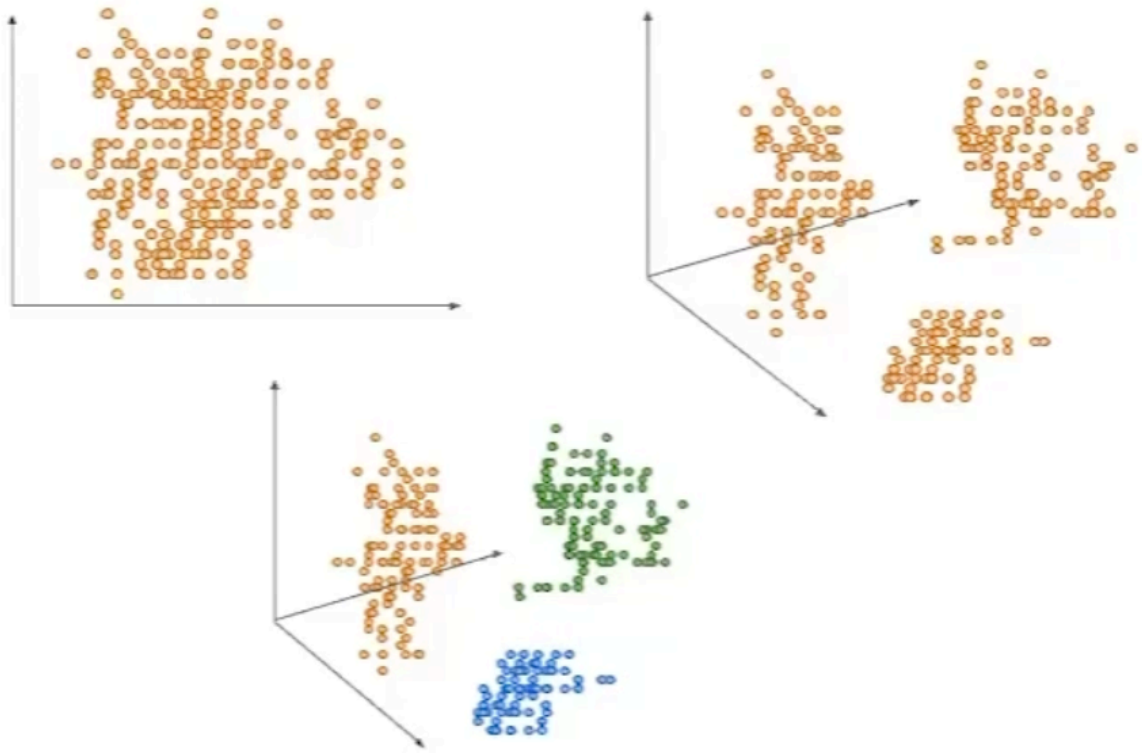
Summary

- We use PCA to deal with high-dimensional datasets; some pros are:
  - Better visualization of the data
  - Reduction of the effects of the curse of dimensionality
  - Reduction of file size
- PCA compresses the datasets into a lower-dimensional state by projecting observations onto a new space
- More variation, more information, easier to distinguish between observations
- When we use PCA we lose data interpretability

2. Clustering (Intro Through K-Means)

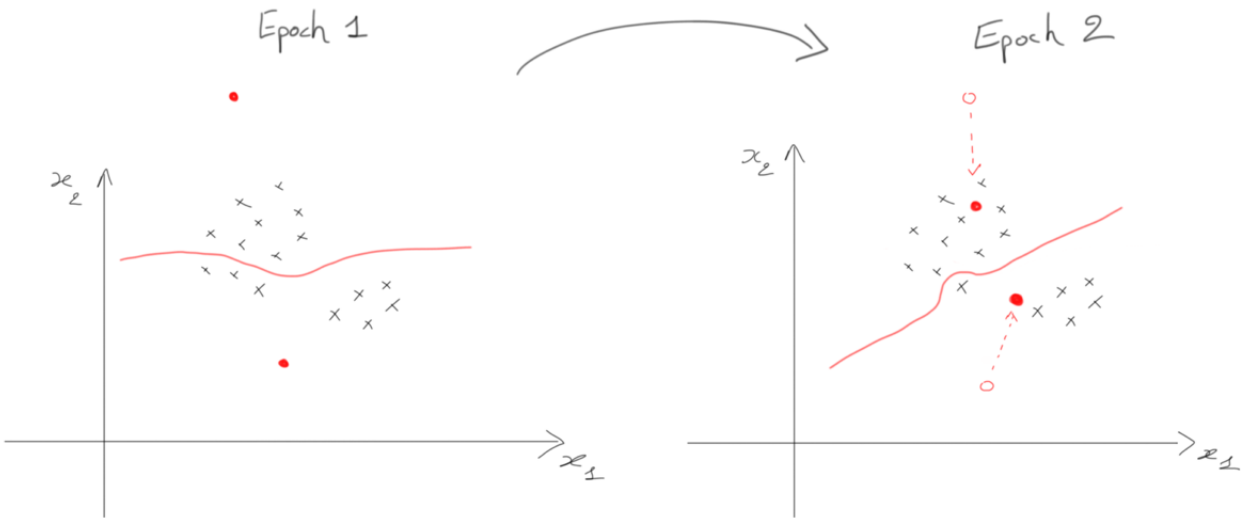
> The process of organizing data points into groups whose members are similar in some way

Find **categories** (classes, segments) of **unlabelled** data rather than just trying to reduce dimensionality



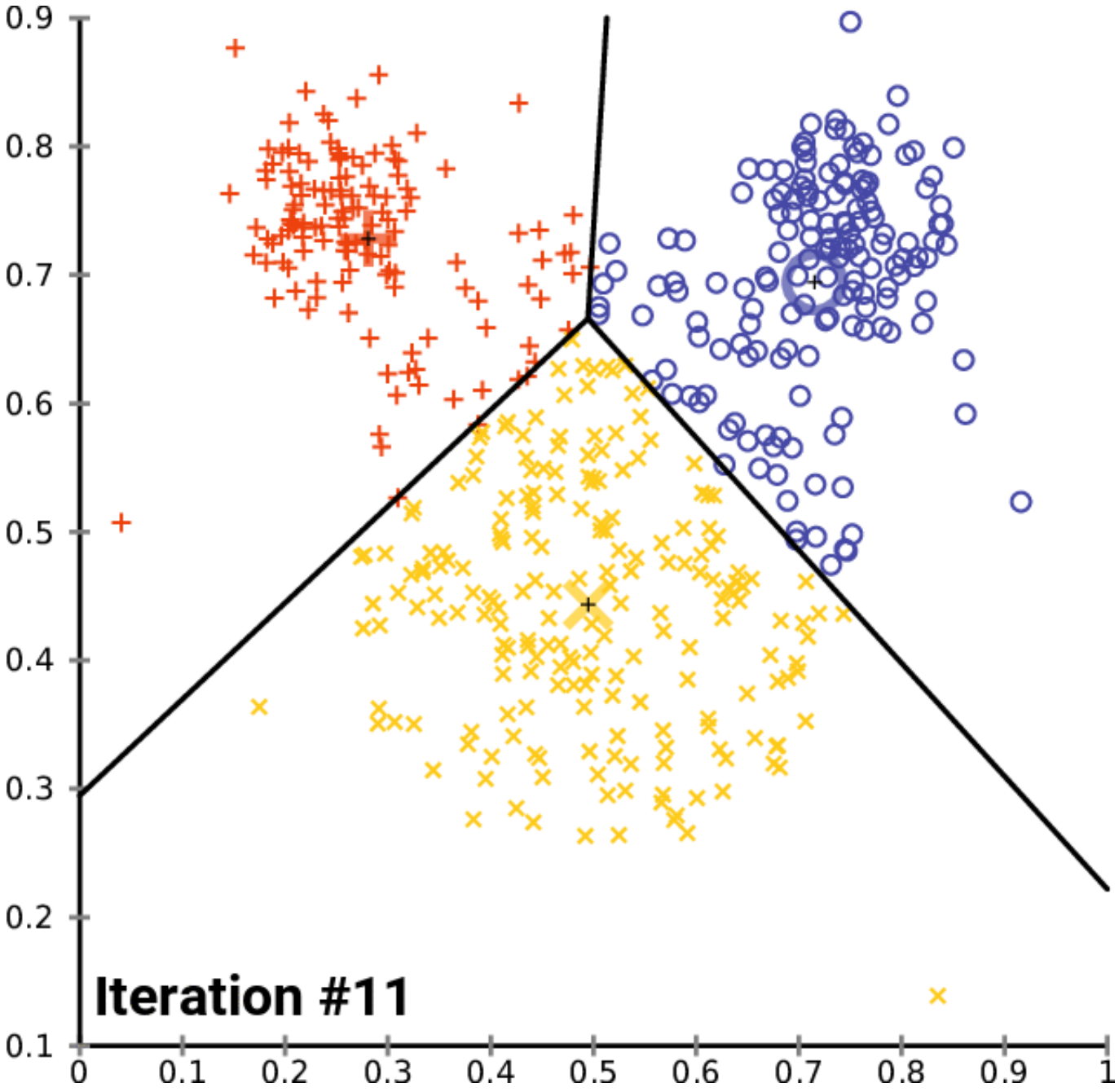
👉 Works better on data that is already clustered, geometrically speaking
👉 Use PCA for dimensionality reduction beforehand Euclidean distances work better in lower dimensions)!

2.1 K-Means Explained



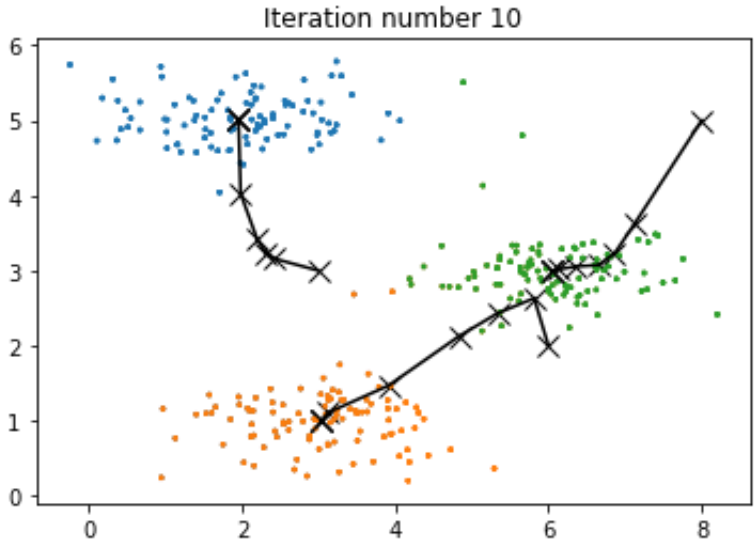
1. Choose the number of clusters `K` to look for
2. Initialize `K` **centroids** at random
3. Compute the **mean square distance** between each data point and each centroid
4. Assign each data point to the closest centroid (a cluster is formed)
5. Compute the mean
   μ_j
   of each cluster, the result of which becomes your new centroid

One epoch is done, repeat from step 3!


0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Iteration #11

In practice

- K-means is usually run a few times with different random initializations
- We can use a random mini-batch at each epoch instead of the full dataset
- The algorithm is quite fast



2.2 Implementation

In Scikit-learn

https://scikit-learn.org/stable/modules/clustering.html (https://scikit-learn.org/stable/modules/clustering.html)

Use

- `scikit.clustering.KMeans`
- `scikit.clustering.MiniBatchKMeans` — same but uses batch samples instead of the whole dataset, in order to go faster

Let's try to find `k = 3` clusters for our wine dataset

(suppose we don't know the true labels)

💡 First, let's place ourselves in the Principal Component space we had already computed

Although not mandatory, applying PCA first helps to separate data more easily!

In [ ]:
```
X_proj
```

Out[ ]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	
0	3.316751	-1.443463	-0.165739	-0.215631	0.693043	-0.223880	0.596427	0.065139	0.
1	2.209465	0.333393	-2.026457	-0.291358	-0.257655	-0.927120	0.053776	1.024416	-0.
2	2.516740	-1.031151	0.982819	0.724902	-0.251033	0.549276	0.424205	-0.344216	-1.
3	3.757066	-2.756372	-0.176192	0.567983	-0.311842	0.114431	-0.383337	0.643593	0.
4	1.008908	-0.869831	2.026688	-0.409766	0.298458	-0.406520	0.444074	0.416700	0.
...	...	...	...	...	...	...	...	...	
173	-3.370524	-2.216289	-0.342570	1.058527	-0.574164	-1.108788	0.958416	-0.146097	-0.
174	-2.601956	-1.757229	0.207581	0.349496	0.255063	-0.026465	0.146894	-0.552427	-0.
175	-2.677839	-2.760899	-0.940942	0.312035	1.271355	0.273068	0.679235	0.047024	0.
176	-2.387017	-2.297347	-0.550696	-0.688285	0.813955	1.178783	0.633975	0.390829	0.
177	-3.208758	-2.768920	1.013914	0.596903	-0.895193	0.296092	0.005741	-0.292914	0.

178 rows × 13 columns

In [ ]:
```
from sklearn.cluster import KMeans

# Fit K-means
km = KMeans(n_clusters=3)
km.fit(X_proj)
```

Out[ ]:
```
KMeans(n_clusters=3)
```

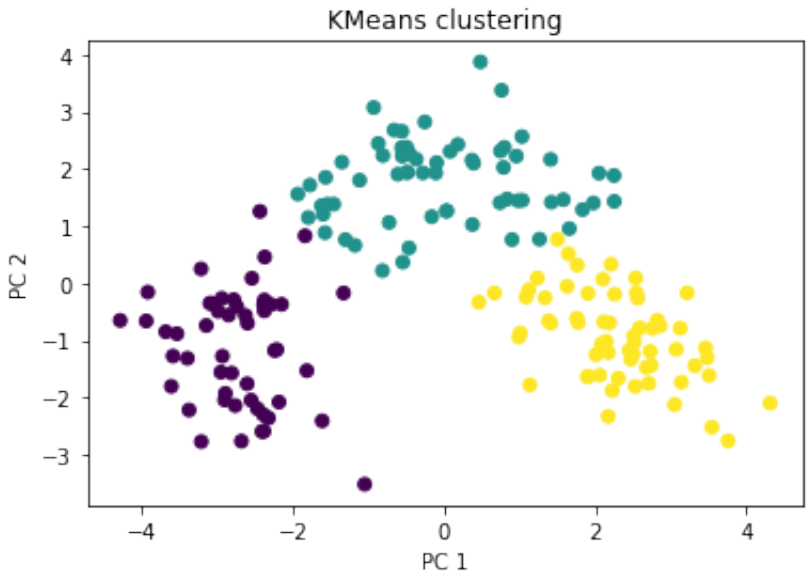
In [ ]:
```
# The 3 centroids' coordinates (expressed in the space of PCs)
km.cluster_centers_.shape
```

Out[ ]:
```
(3, 13)
```

In [ ]:
```
# The 177 observations are classified automatically
km.labels_
```

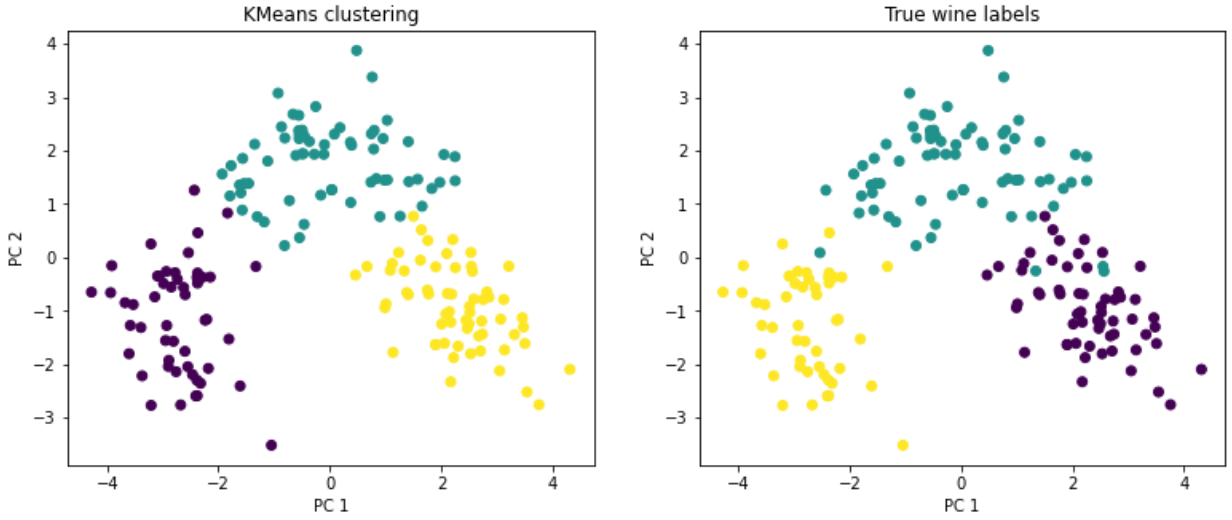
Out[ ]:
```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 0, 1, 1,
       1, 1,
       1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1,
       1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0,
       0, 0], dtype=int32)
```

In [ ]:
```
plt.scatter(X_proj.iloc[:,0], X_proj.iloc[:,1], c=km.labels_)
plt.title('KMeans clustering'); plt.xlabel('PC 1'); plt.ylabel('PC
2');
```



💡 In our case, we *know* the true labels; let's measure performance

In [ ]:
```
# Visualization
plt.figure(figsize=(13,5))

plt.subplot(1,2,1)
plt.scatter(X_proj.iloc[:,0], X_proj.iloc[:,1], c=km.labels_)
plt.title('KMeans clustering'); plt.xlabel('PC 1'); plt.ylabel('PC 2')

plt.subplot(1,2,2)
plt.scatter(X_proj.iloc[:,0], X_proj.iloc[:,1], c=y)
plt.title('True wine labels'); plt.xlabel('PC 1'); plt.ylabel('PC 2');
```



In [ ]:
```
# Accuracy
from sklearn.metrics import accuracy_score

y_pred = pd.Series(km.labels_).map({0:0, 1:2, 2:1}) # WARNING: change this manually!
accuracy_score(y_pred, y)
```

Out[ ]:
```
0.9662921348314607
```

Predict?

We can use the unsupervised K-means algorithm to **predict** (classify) a new X

In [ ]:
```
# Build DF with column names from X_proj and some random data
new_X = pd.DataFrame(data = np.random.random((1,13)), columns = X_proj.columns)

km.predict(new_X)
```

Out[ ]:
```
array([1], dtype=int32)
```

2.3 K-Means' Loss Function?

`km.fit(X)` finds parameters
β
that minimize a loss

- Each
  β_j
  parameter is the **centroid**
  μ_j
  of its respective cluster
  C_j

- The loss function is called **inertia**
  $L(\mu)$

- = **sum** of **squared distance** between each observation and their **closest centroid**
- = sum of **within-cluster sum of squares** (WCSS)
- = variance

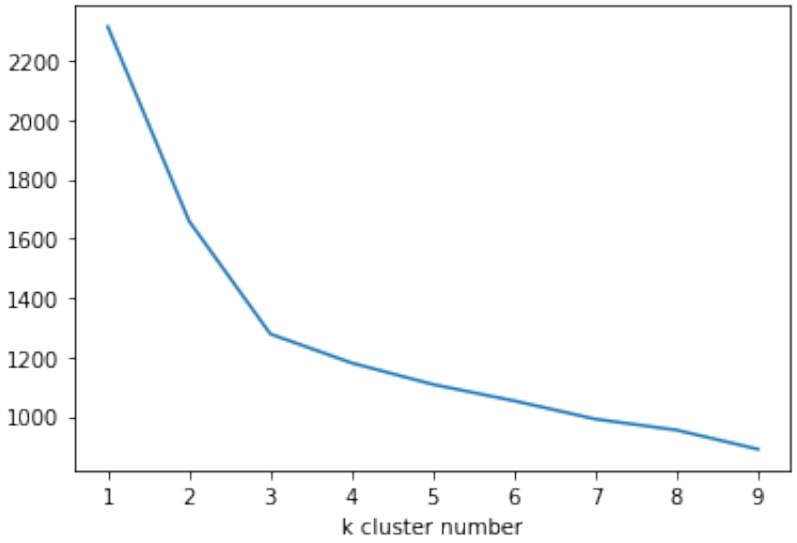
$$\text{inertia} = L(\mu) = \sum_{j=1}^{K} \sum_{x_i \in C_j} (|$$

Choosing Hyperparameter K

- Choose `K` such that the inertia (`Kmeans().inertia_`) is minimized
- Use the **elbow method** here as well

In [ ]:
```
inertias = []
ks = range(1,10)

for k in ks:
    km_test = KMeans(n_clusters=k).fit(X)
    inertias.append(km_test.inertia_)

plt.plot(ks, inertias)
plt.xlabel('k cluster number')
```

Out[ ]:
```
Text(0.5, 0, 'k cluster number')
```



What can we use it for?

👉 Document classification (finding unlabeled categories or topics)

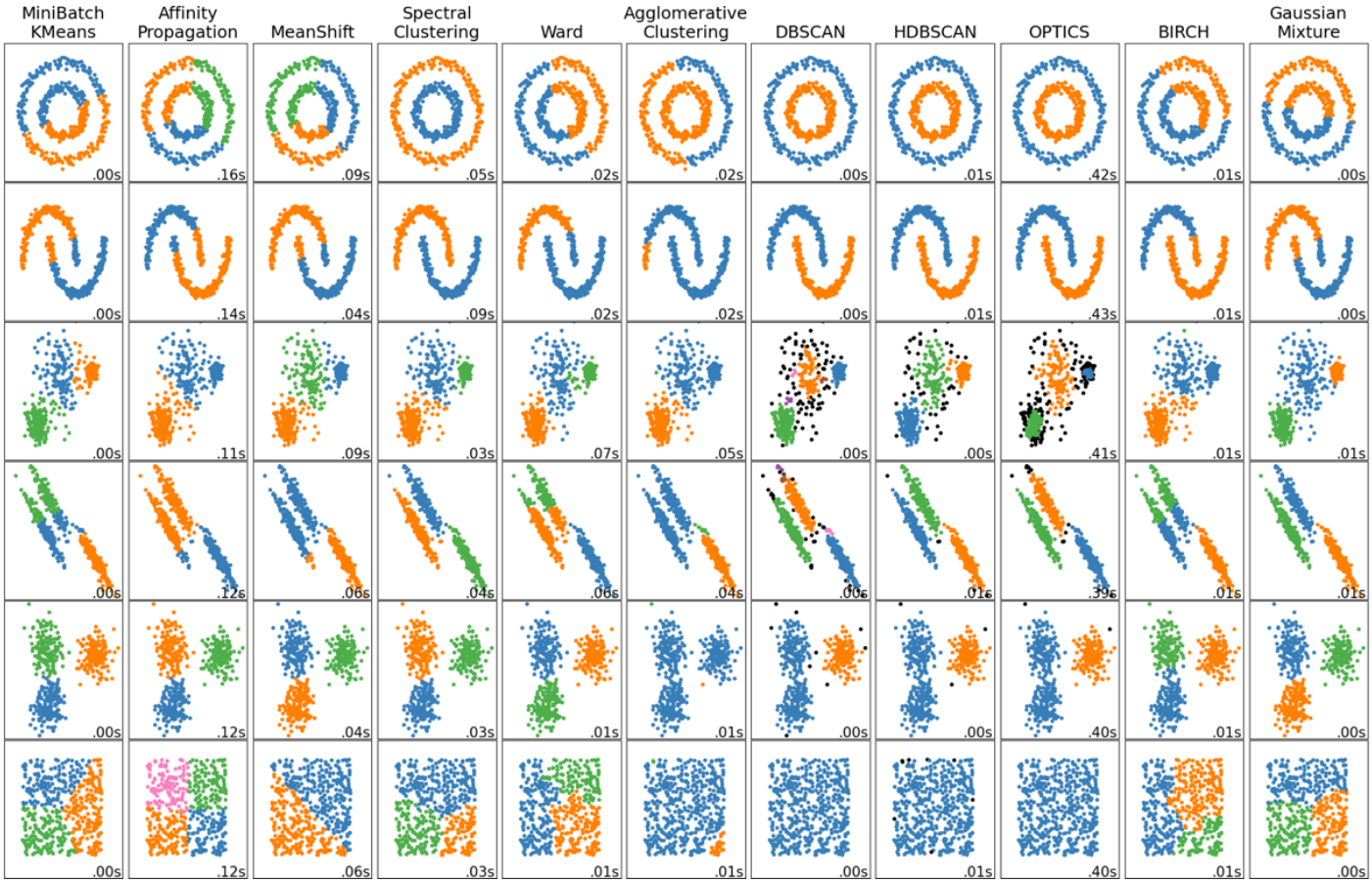
👉 Delivery store optimization (find the optimal number of launch locations)

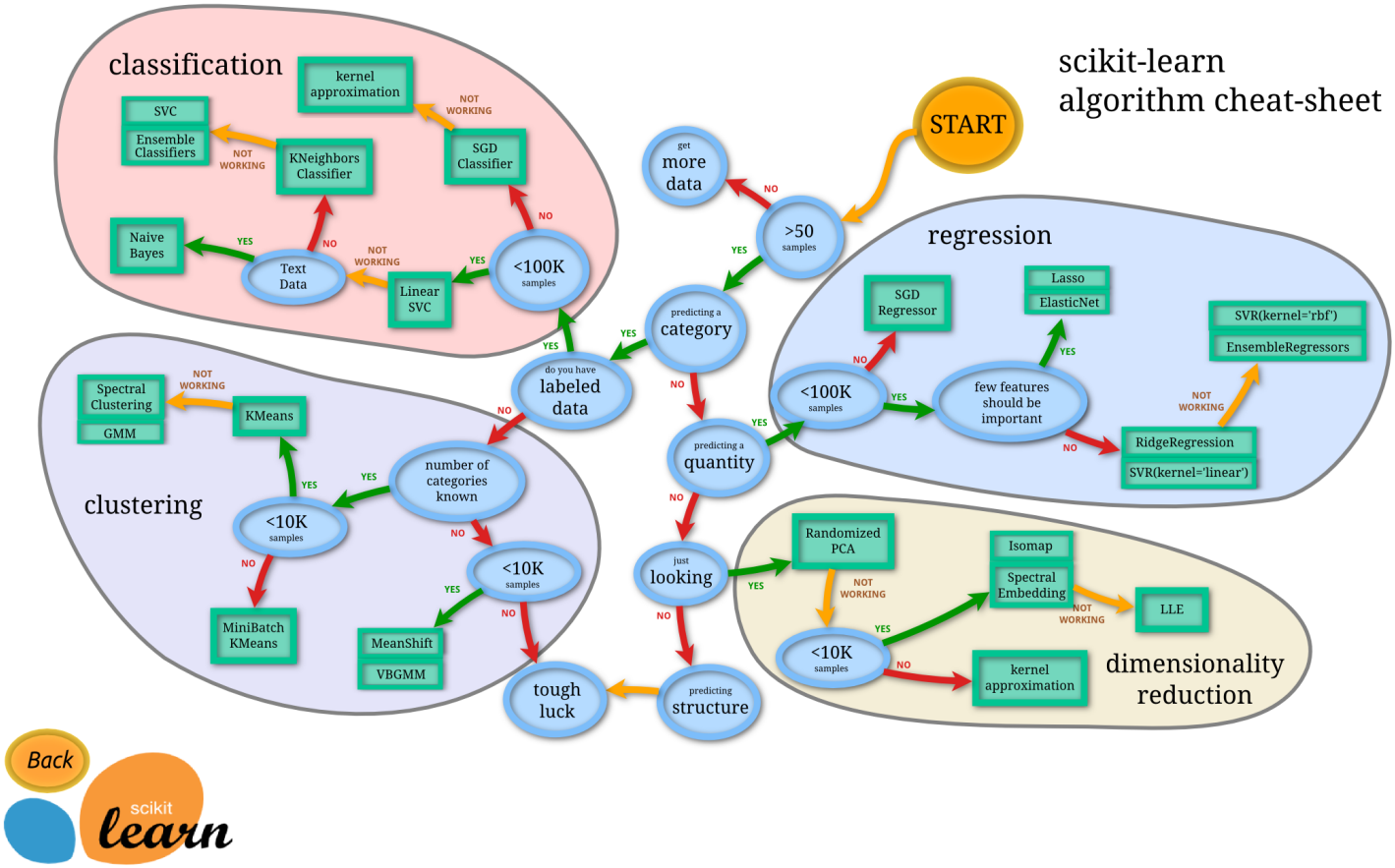
👉 Customer segmentation (classify different types of customer based on their behavior)

https://dzone.com/articles/10-interesting-use-cases-for-the-k-means-algorithm (https://dzone.com/articles/10-interesting-use-cases-for-the-k-means-algorithm)

2.4 There are many other clustering approaches

https://scikit-learn.org/stable/modules/clustering.html (https://scikit-learn.org/stable/modules/clustering.html)





Bibliography

- PCA explained to your grandmother (https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues) 1700 upvotes on StackExchange 💫
- PCA for ML (https://towardsdatascience.com/using-principal-component-analysis-pca-for-machine-learning-b6e803f5bf1e)
- KMeans explained (https://towardsdatascience.com/k-means-clustering-explain-it-to-me-like-im-10-e0badf10734a)

🚀 Your turn!