LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND STATISTIK
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR DATENBANKSYSTEME UND DATA MINING

Lecture Notes to

Big Data Management and Analytics

Winter Term 2018/2019

Python Best Practices

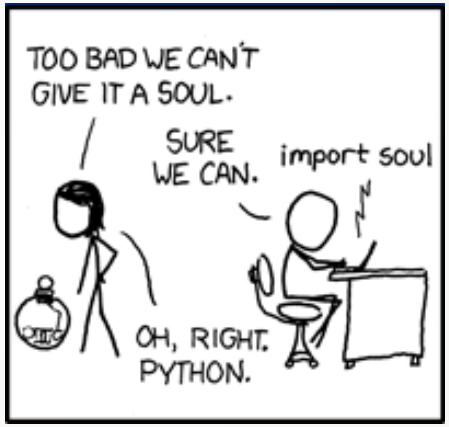
DBS

Agenda

- The KDD Process Model
  - Selection
  - Preprocessing
  - Transformation
  - Data Mining
  - Interpretation/Evaluation
- import finis

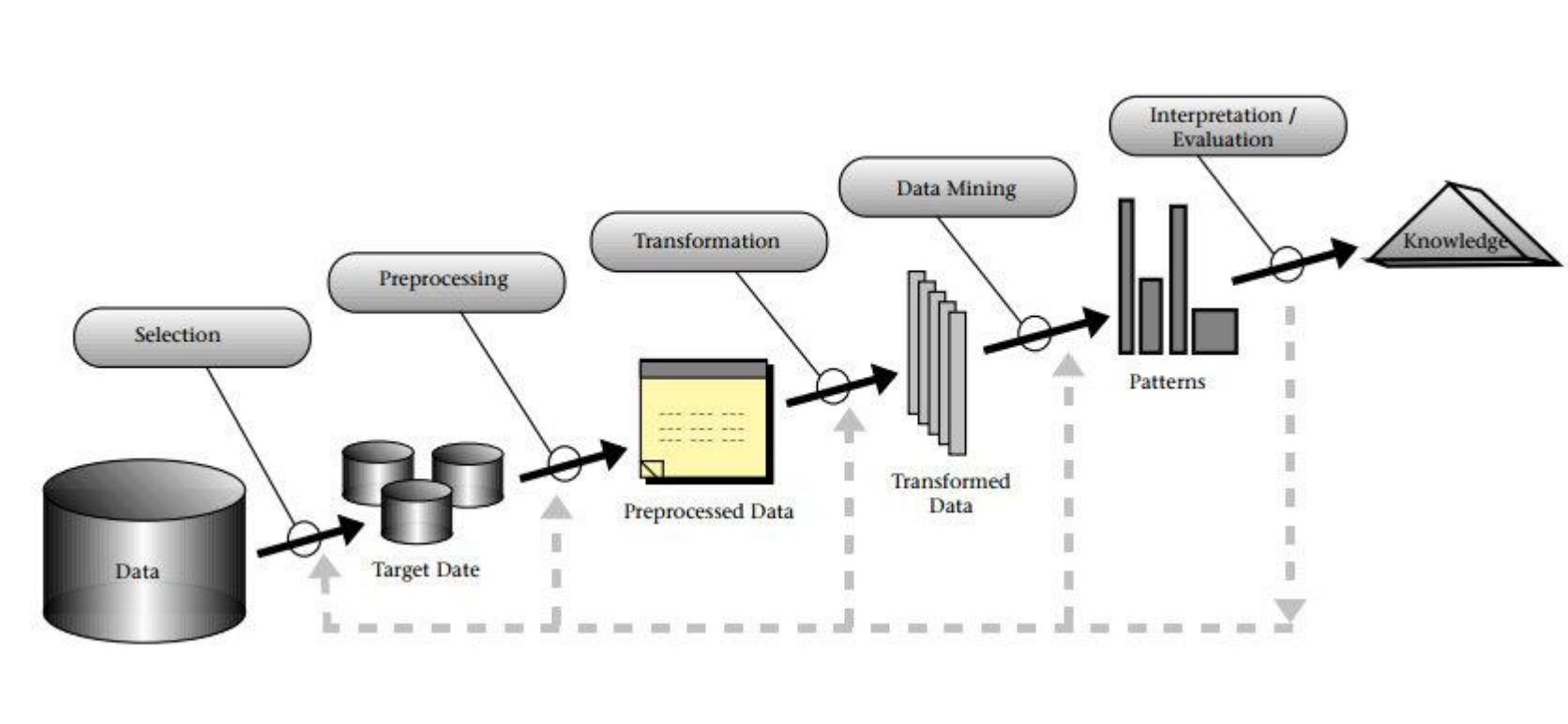
"It is a capital mistake to theorize before one has data."

Sherlock Holmes,
"A Study in Scarlett" (Arthur Conan Doyle).



[0]

The KDD Process Model



[1
]

Selection

- Data acquisition
- Managing the data
- Selection of relevant data
  - Focusing on a subset of variables or data samples

Reading in data

- From a csv file

```
import pandas as pd

#read in a csv file into a data frame (df):
df = pd.read_csv('filename.csv')

#read in a csv file … without the header
df = pd.read_csv('filename.csv', header=None)


#read in a csv file … with individual column names
df = pd.read_csv('filename.csv', names=['col0','col1'…'coln'])
```

What is a data frame?

Index	Column0	…	ColumnD
0			
1			
…			
n			

2D labeled data structure with independent columns of potentially different types.

Reading in data

- From a csv file

```
#read in a csv file … skipping the first k rows
df = pd.read_csv('filename.csv', skiprows=k)


#read in a csv file … using only specific columns
df = pd.read_csv('filename.csv', usecols=[colindexB, colindexA,…])
```

filtering data rows

- By specific conditions

```
#filtering a data frame by multiple conditions
df[(df.colName pred val) boolOP (df.colName pred val) ... boolOP (df.colName pred val)]
```

$pred \in \{<, >, \leq, \geq, ==, \neq \cdots\}$ $boolOP \in \{\&, |, !, \hat{} \ldots\}$

Pitfall time:

```
customer_id, icecream,money_spent, rating
0, chocolate, 2, awesome
., vanilla, . , awesome
2, chocolate, ., awfull
NA, vanilla, 2, okay
```

```
In [28]: df4 = pd.read_csv('icecream3f.csv')

In [33]: df4['rating']
```

Pitfall time:

```
In [28]: df4 = pd.read_csv('icecream3f.c

In [33]: df4['rating']
```

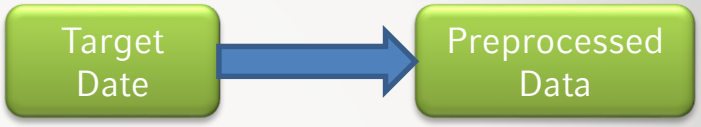
```
/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py in __getitem__(self, key)
   1962             return self._getitem_multilevel(key)
   1963         else:
-> 1964             return self._getitem_column(key)
   1965
   1966     def _getitem_column(self, key):

/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py in _getitem_column(self, key)
   1969         # get column
   1970         if self.columns.is_unique:
-> 1971             return self._get_item_cache(key)
   1972
   1973         # duplicate columns & possible reduce dimensionality

/anaconda3/lib/python3.6/site-packages/pandas/core/generic.py in _get_item_cache(self, item)
   1643         res = cache.get(item)
   1644         if res is None:
-> 1645             values = self._data.get(item)
   1646             res = self._box_item_values(item, values)
   1647             cache[item] = res

/anaconda3/lib/python3.6/site-packages/pandas/core/internals.py in get(self, item, fastpath)
   3588
   3589             if not isnull(item):
-> 3590                 loc = self.items.get_loc(item)
   3591             else:
   3592                 indexer = np.arange(len(self.items))[isnull(self.items)]

/anaconda3/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   2442                 return self._engine.get_loc(key)
   2443             except KeyError:
-> 2444                 return self._engine.get_loc(self._maybe_cast_indexer(key))
   2445
   2446         indexer = self.get_indexer([key], method=method, tolerance=tolerance)

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'rating'
```

Pitfall time:

```
1  customer_id, icecream,money_spent, rating
2  0, chocolate, 2, awesome
3  ., vanilla, . , awesome
4  2, chocolate, ., awfull
5  NA, vanilla, 2, okay
```

```
In [34]: df4[' rating']

Out[34]: 0     awesome
         1     awesome
         2      awfull
         3        okay
         Name:  rating, dtype: object
```

Preprocessing



- Integrate data from different sources
- Completion of data / handling missing values
- Denoising of data where appropriate

Reading in data

Core assumption:
All csv files in the directory have the same structure/layout

- From *multiple* csv files

```
import pandas as pd
import glob
import os

#path where all the csv files are located
path =r'/path/to/csvfiles/'

#creates a list of filepaths for each csv file
listOfFilepaths = glob.glob(os.path.join(path, "*.csv"))

#concatenate a list of data frames
pd.concat((pd.read_csv(file) for file in listOfFilepaths))
```

Module for detecting pathnames following a specific pattern.
Here: ending with .csv

"Raw data is mostly dirty" – Data cleansing

- Finding missing values

```
#returns which of the values of a column are missing
df.isna(df['colName'])

#returns for an entire data frame if values are missing
df.isna()
```

"This is not what I meant when I said 'we need better data cleansing!'"

www.iwaysoftware.com/go/dataquality

"Raw data is mostly dirty" – Data cleansing

- Replacing missing values (NaN, None, NaT etc.)

```
#fills missing values of a column with a default value
df['colName'].fillna(valueToFill)

# fills missing values of the entire data frame with a default value
df.fillna(valueToFill)
```

valueToFill can be either a string or an float, int etc.

"Raw data is mostly dirty" – Data cleansing

- Wrong data type

Problem statement:
Pandas data frame column is parsed as string but should be a numeric value

```
#convert the values of a specific column to numeric values
pd.to_numeric(df['columnName'])
```

Pitfall time

- Wrong data type

Problem statement:
Pandas data frame column is parsed as string but should be a numeric value

customer_id
0
.
2
NaN
5
6
9
10
11
12
13
14

But: what about NaN's or other non-nummeric values?

```
pd.to_numeric(df2['customer_id'])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
pandas/_libs/src/inference.pyx in pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "."

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
<ipython-input-36-81e39cee26a3> in <module>()
----> 1 pd.to_numeric(df2['customer_id'])

/anaconda3/lib/python3.6/site-packages/pandas/core/tools/numeric.py in to_numeric(arg, errors, downcast)
    124             coerce_numeric = False if errors in ('ignore', 'raise') else True
    125             values = lib.maybe_convert_numeric(values, set(),
--> 126                                                coerce_numeric=coerce_numeric)
    127
    128     except Exception:

pandas/_libs/src/inference.pyx in pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "." at position 1
```

Pitfall time

- Wrong data type

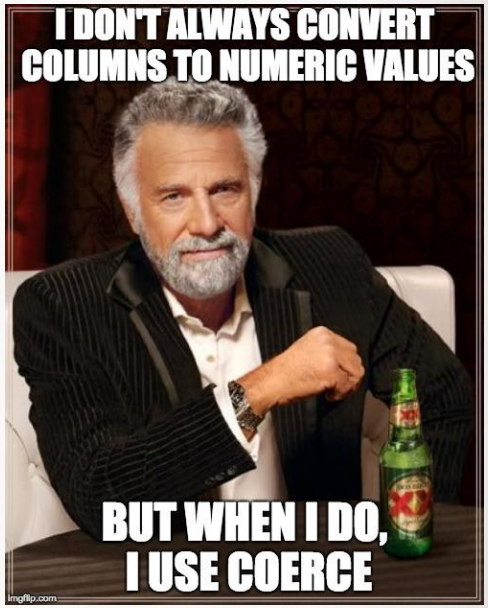
Problem statement:
Pandas data frame column is parsed as string but should be a numeric value

customer_id
0
.
2
NaN
5
6
9
10
11
12
13
14

Solution: coerce !

```
pd.to_numeric(df2['customer_id'], errors='coerce')
```

```
0     0.0
1     NaN
2     2.0
3     NaN
0     5.0
1     6.0
2     9.0
0    10.0
1    11.0
2    12.0
3    13.0
4    14.0
Name: customer_id, dtype: float64
```



Transformation

- Deriving new attributes:
  - Grouping and aggregating
- Selection of relevant attributes
  - Dimensionality reduction

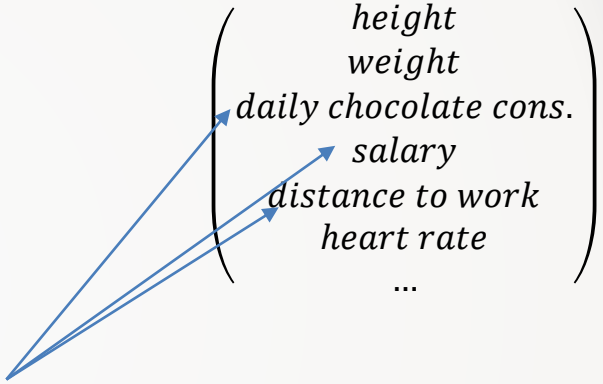
Transformation

- Deriving new attributes:
  - Grouping and aggregating

```
#group original data frame by specific column
dfgrouped = df.groupby('columnName')

#printing all grouped elements
for name,group in dfgrouped:
    print(name)
    print(group)
```

Transformation

- Dimensionality reduction
  - Principal Component Analysis (PCA)
    - Select the (k-)dimensions with the highest variance



Which one are the most important dimensions?
What does ‚most important' mean?

Given a data matrix X of the following structure:

$$X = \begin{pmatrix} \cdots \\ \vdots \quad \ddots \quad \vdots \\ \cdots \end{pmatrix}$$

Data points

dimensions

Transformation

- Steps of PCA:
  - Centering Data

Mean vector: $\hat{\mu} = \frac{1}{N} \sum_i X_i$

Centering data: $\tilde{X} = X - \hat{\mu}$

$$\begin{pmatrix} height \\ weight \\ daily\ chocolate\ cons. \\ salary \\ distance\ to\ work \\ heart\ rate \\ \dots \end{pmatrix}$$

Transformation

- Steps of PCA:
  - Compute the covariance matrix

$$cov(X) \approx \hat{\Sigma} = \frac{1}{N}\tilde{X}^T\tilde{X}$$

$$\begin{pmatrix} height \\ weight \\ daily\ chocolate\ cons. \\ salary \\ distance\ to\ work \\ heart\ rate \\ \dots \end{pmatrix}$$

What is actually the covariance matrix?

$$\begin{pmatrix} (X_1-\mu_1)^2 & \cdots & (X_1-\mu_1)(X_n-\mu_n) \\ \vdots & \ddots & \vdots \\ (X_n-\mu_n)(X_1-\mu_1) & \cdots & (X_n-\mu_n)^2 \end{pmatrix} = \begin{pmatrix} Var(X_1) & \cdots & Cov(X_1,X_n) \\ \vdots & \ddots & \vdots \\ Cov(X_n,X_1) & \cdots & Var(X_n) \end{pmatrix}$$

Transformation

- Steps of PCA:
  - Compute the eigenpairs (eigenvalues, eigenvectors)

$$\begin{pmatrix} height \\ weight \\ daily\ chocolate\ cons. \\ salary \\ distance\ to\ work \\ heart\ rate \\ \dots \end{pmatrix}$$

Compute: $\det(\hat{\Sigma} - \lambda I)$ → yields eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_d$, where d:=dimensionality

What is the intuition of eigenvalues?
→ Represent the variance along the corresponding principal component

Transformation

- Steps of PCA:
  - Compute the eigenpairs (eigenvalues, eigenvectors)

Solve $\hat{\Sigma} v_i = \lambda_i v_i$ to compute the eigenvectors

Eigenvectors: $\hat{\hat{U}} = \begin{pmatrix} v_{1_1} & \cdots & v_{d_1} \\ \vdots & \ddots & \vdots \\ v_{1_d} & \cdots & v_{d_d} \end{pmatrix}$

What is the intuition of eigenvectors?
→Directions among which the transformation is invariant.

$$\begin{pmatrix} height \\ weight \\ daily\ chocolate\ cons. \\ salary \\ distance\ to\ work \\ heart\ rate \\ \dots \end{pmatrix}$$

Transformation

- Steps of PCA:
  - Select the (k-)relevant principle components
  - Project the data points to the lower dimensional representation

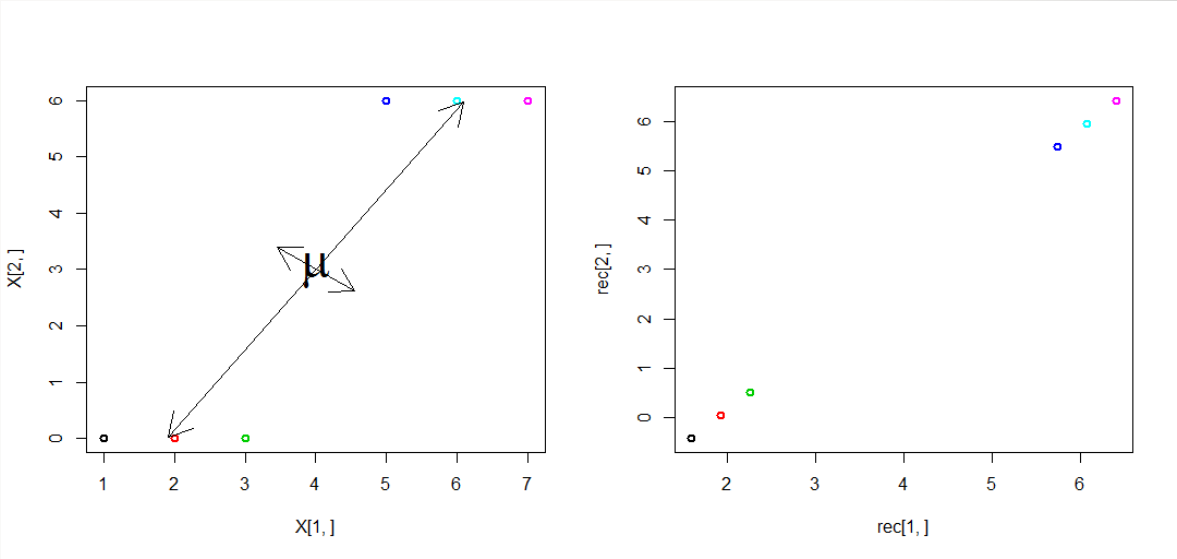
Use e.g. elbow method.

$$\begin{pmatrix} height \\ weight \\ daily\ chocolate\ cons. \\ salary \\ distance\ to\ work \\ heart\ rate \\ \dots \end{pmatrix}$$

$$Y = \tilde{X} \cdot U$$

Sketch time …

Transformation



- Steps of PCA:
  - Centering of the data
  - Computing covariance
  - Computing eigenpairs
  - Selecting top (k-)principal components
  - Projecting data points to lower dimensional representation

Transformation

- PCA in Python using sklearn

```
from sklearn import decomposition

#given a data matrix X, create PCA object
pca = decomposition.PCA()

#fit the model with X and perform dim. reduction on X
pca.fit_transform(X)
```

```
from sklearn import decomposition

x = np.array([
        [2, 4],
        [4, 6],
        [6,7],
        [7,10],])


pca = decomposition.PCA()
pca.fit_transform(x)
```

```
array([[-3.88133419, -0.24544841],
       [-1.05854569, -0.06694048],
       [ 1.0139187 ,  0.7726376 ],
       [ 3.92596117, -0.46024871]])
```

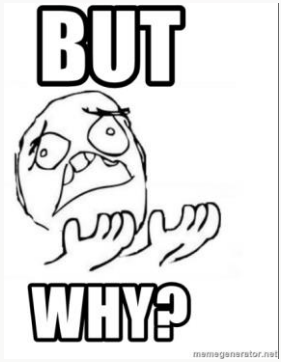
Transformation

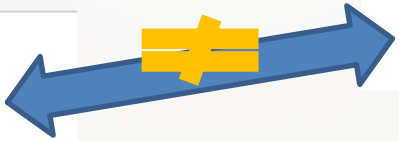
Pitfall time!

- PCA in Python using numpy

```
import numpy as np
x = np.array([
        [2, 4],
        [4, 6],
        [6,7],
        [7,10],
    ])

xcent = x - np.mean(x, axis = 0)
cov = np.cov(xcent, rowvar = False)
evals , evecs = np.linalg.eig(cov)

a = np.dot(xcent, evecs)
print(a)
```

```
[[ 0.24544841  3.88133419]
 [ 0.06694048  1.05854569]
 [-0.7726376  -1.0139187 ]
 [ 0.46024871 -3.92596117]]
```





```
from sklearn import decomposition

x = np.array([
        [2, 4],
        [4, 6],
        [6,7],
        [7,10],])


pca = decomposition.PCA()
pca.fit_transform(x)
```

```
array([[-3.88133419, -0.24544841],
       [-1.05854569, -0.06694048],
       [ 1.0139187 ,  0.7726376 ],
       [ 3.92596117, -0.46024871]])
```

Transformation

Need to sort eigenvalues and eigenvectors accordingly in descending order!

- PCA in Python using numpy

```
import numpy as np

#given a data matrix X, compute the mean vector
xcent = X – np.mean(X)

#compute the covariance matrix
cov = np.cov(xcent, rowvar = False)

#compute eigenvalues and eigenvectors
evals, evecs = np.linalg.eig(cov)

#sort the eigenvectors according to the eigenvalues
idx = np.argsort(evals)[::-1]
evecs = evecs[:,idx]
evals = evals[idx]

a = np.dot(xcent, evecs)
```

```
import numpy as np
x = np.array([
        [2, 4],
        [4, 6],
        [6,7],
        [7,10],
    ])

xcent = x - np.mean(x, axis = 0)
cov = np.cov(xcent, rowvar = False)
evals , evecs = LA.eig(cov)

idx = np.argsort(evals)[::-1]
evecs = evecs[:,idx]
evals = evals[idx]

a = np.dot(xcent, evecs)
print(a)
```

```
[[ 3.88133419  0.24544841]
 [ 1.05854569  0.06694048]
 [-1.0139187  -0.7726376 ]
 [-3.92596117  0.46024871]]
```

Transformation

Another pitfall?

- PCA in Python using numpy

```
from sklearn import decomposition

x = np.array([
        [2, 4],
        [4, 6],
        [6,7],
        [7,10],])


pca = decomposition.PCA()
pca.fit_transform(x)
```

```
array([[-3.88133419, -0.24544841],
       [-1.05854569, -0.06694048],
       [ 1.0139187 ,  0.7726376 ],
       [ 3.92596117, -0.46024871]])
```

Different signs!

But why?

```
import numpy as np
x = np.array([
        [2, 4],
        [4, 6],
        [6,7],
        [7,10],
    ])

xcent = x - np.mean(x, axis = 0)
cov = np.cov(xcent, rowvar = False)
evals , evecs = LA.eig(cov)

idx = np.argsort(evals)[::-1]
evecs = evecs[:,idx]
evals = evals[idx]

a = np.dot(xcent, evecs)
print(a)
```

```
[[ 3.88133419  0.24544841]
 [ 1.05854569  0.06694048]
 [-1.0139187  -0.7726376 ]
 [-3.92596117  0.46024871]]
```

- PCA in Python using numpy

Different covariance matrices?

numpy

```
[[ 4.91666667  5.25      ]
 [ 5.25        6.25      ]]
```

sklearn

```
array([[ 4.91666667,  5.25      ],
       [ 5.25      ,  6.25      ]])
```

- PCA in Python using numpy

Different eigenvectors?

numpy

```
[[-0.66107014 -0.75032411]
 [-0.75032411  0.66107014]]
```

≠

sklearn

```
array([[ 0.66107014,  0.75032411],
       [ 0.75032411, -0.66107014]])
```

Why?
Sklearn performs a singular value decomposition (SVD)
The results are not unique with regards to the singular vectors regarding their signs

Is that bad?
Still same eigenvectors, since eigenvectors don't have a sign
Eigenvectors still have the same eigenvalue.

Transformation

- What does a PCA aim for and when is it most helpful?

- Removing redundant or noisy features
- Reducing dimensionality to ‚relevant' dimensions
- Assumption:
  expressiveness of observed variables as a linear combinations of hidden variables.

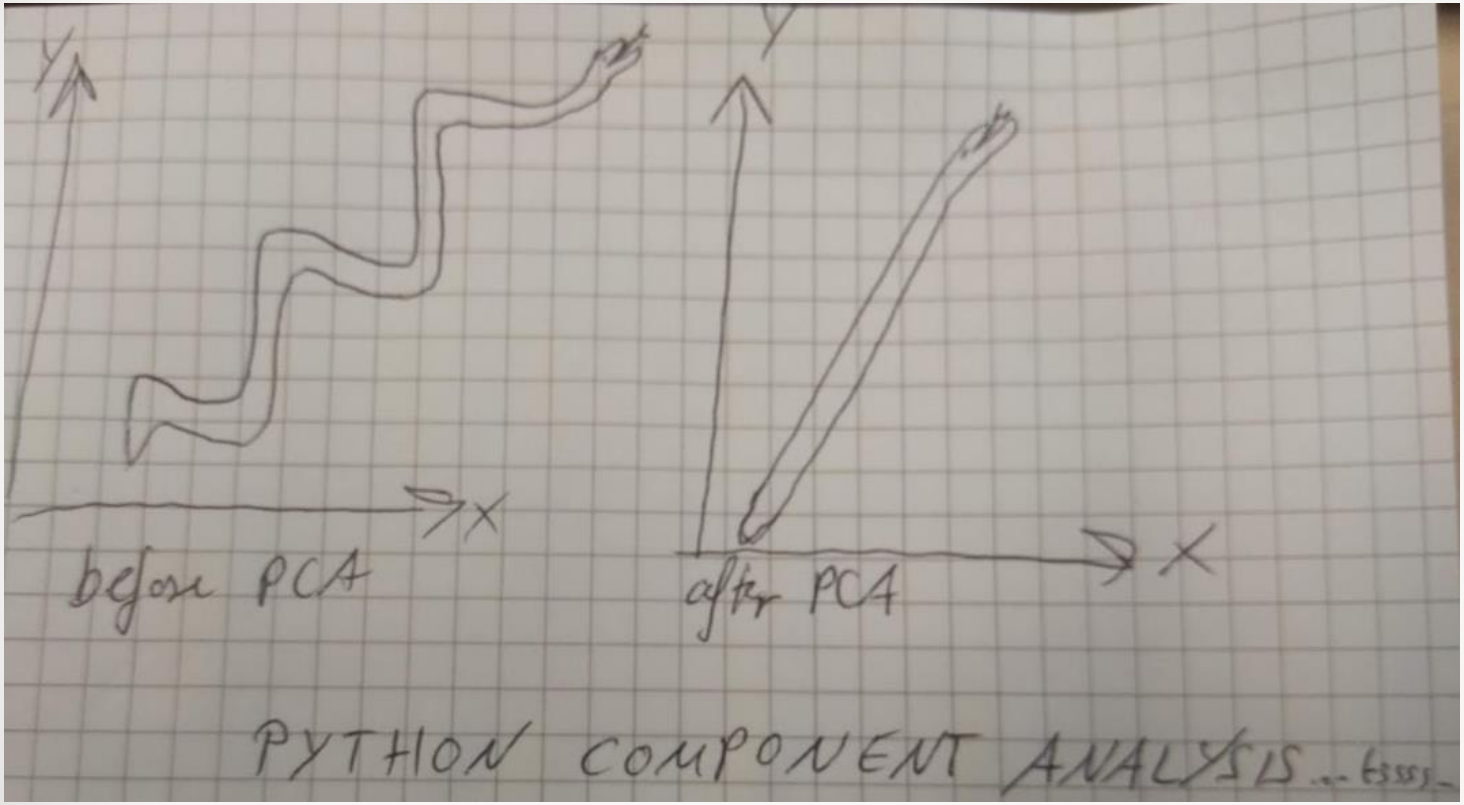
Transformation

- Words of caution:

- Loss of potentially relevant structures
- Outliers may heavily skew the PCA transforamtion

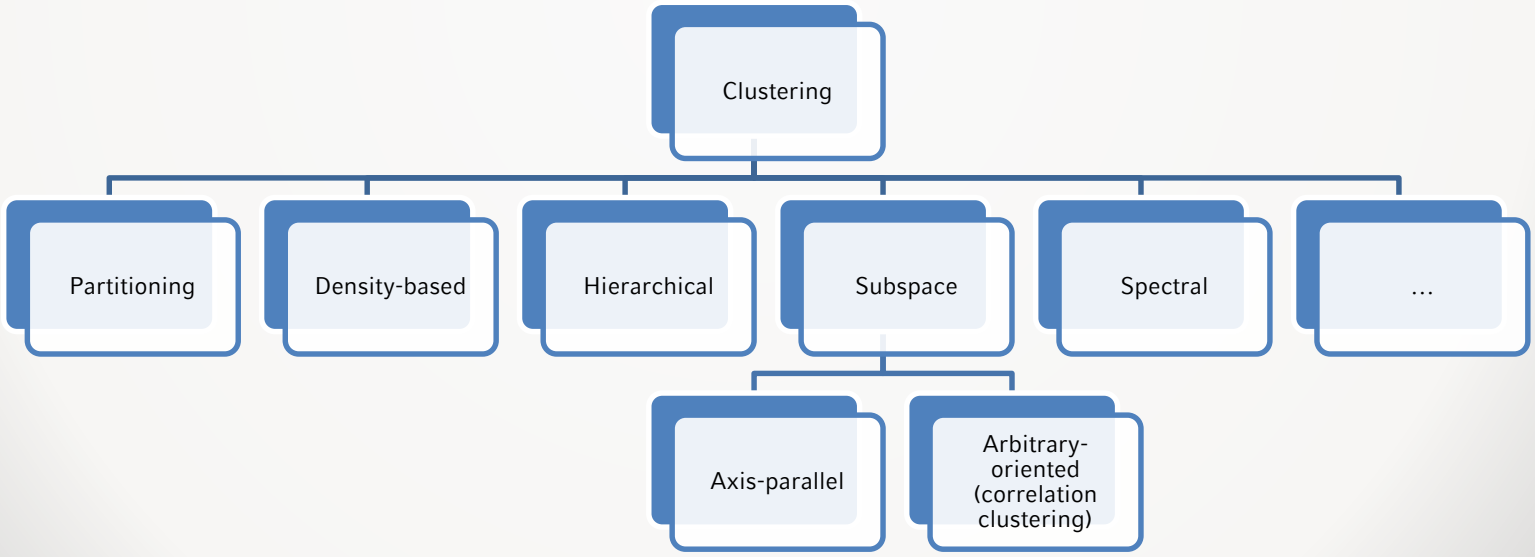
here could be your sketch…

Transformation



Data Mining

- Generating Patterns or Models
  - Applying clustering methods

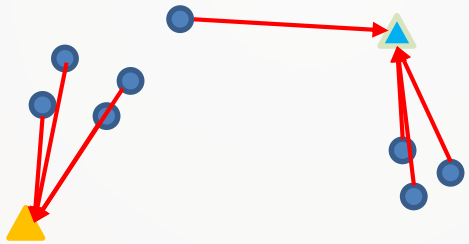


Data Mining

- Generating Patterns or Models
  - Clustering with sklearn
    - Using k-means, with k=2

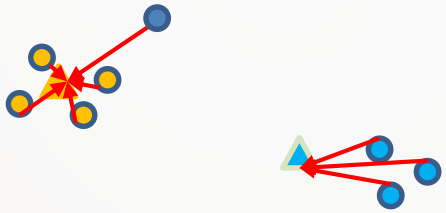
Data Mining

- Generating Patterns or Models
  - Clustering with sklearn
    - Using k-means



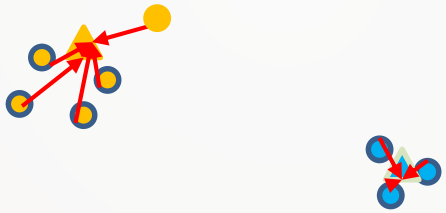
Data Mining

- Generating Patterns or Models
  - Clustering with sklearn
    - Using k-means



Data Mining

- Generating Patterns or Models
  - Clustering with sklearn
    - Using k-means



Data Mining

- Generating Patterns or Models
  - Clustering with sklearn
    - Using k-means

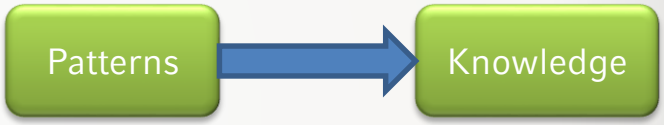
```
from sklearn.cluster import Kmeans

#given a dataset X
kmeans = KMeans(n_clusters=k).fit(X)

#cluster labels
kmeans.labels_

#given a set of points,  assign their cluster labels
kmeans.predict([p1, p2, …,pn])

#returns the centroids
kmeans.cluster_centers_
```

Rule of thumb:
Clustering functions in sklearn have in common:

- A fit function
- A labels field
- A predict function

Interpretation/Evaluation

- Evaluation of the relevance by the user
  - Visualizations
    (scatter, pairplots, heatmaps)



Interpretation/Evaluation

- Evaluation of the relevance by the user
  - Visualizations
    - Scatter Plot

```
import numpy as np
import matplotlib.pyplot as plt

#given a (numpy) array of data points dpts
#get an array for each of the dimensions
x, y = zip(*dpts)
#or...
x, y = (dpts[ : , 0], dpts[ : , 1])

plt.scatter(x, y)

plt.show()
```

Interpretation/Evaluation

- Evaluation of the relevance by the user
  - Visualizations
    - Scatter Plot

  How do we plot the resulting clusters from sklearn?

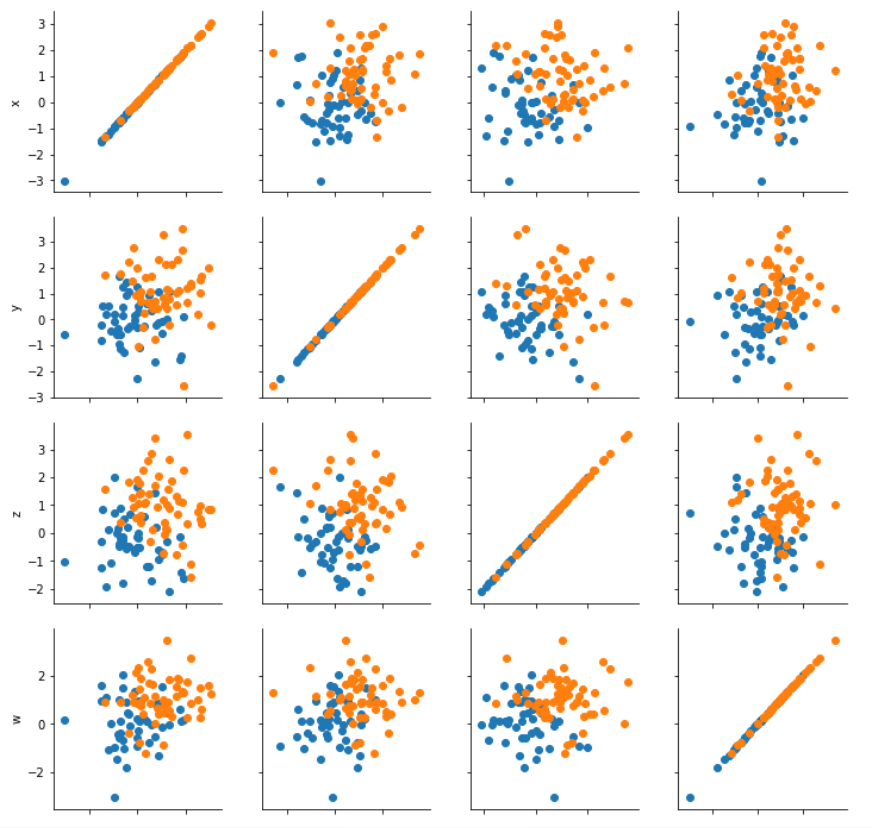
```
from sklearn.cluster import Clusteringmethod
import matplotlib.pyplot as plt

#given a (numpy) array of data points dpts
model = Clusteringmethod(clusterspecificparameters).fit(dpts)

x, y = (dpts[ : , 0], dpts[ : , 1])

plt.scatter(x,y, c=model.labels_.astype(float))
```

Interpretation/Evaluation

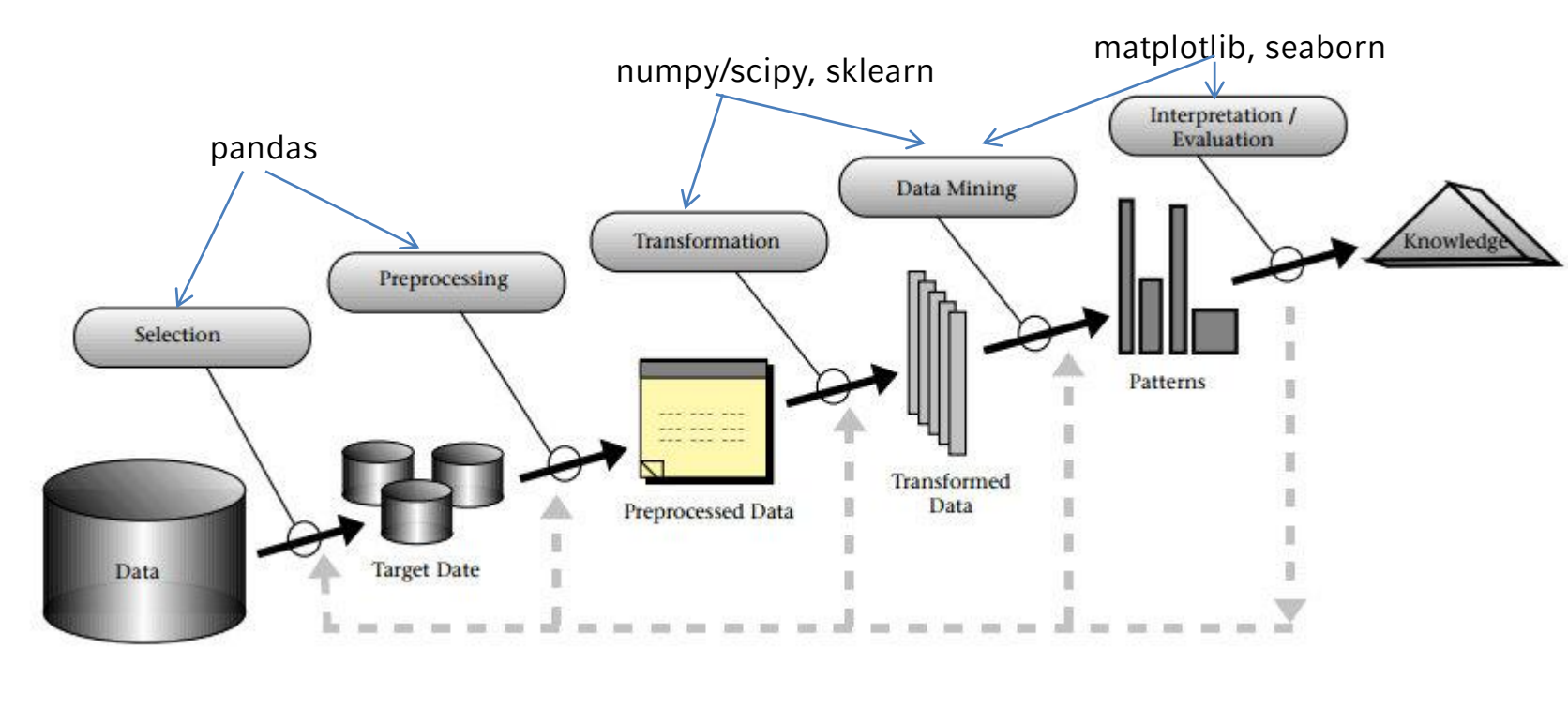
How to plot multidimensional data? → Pair Grids

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import Clusteringmethod

#given a numpy array dpts, convert it to a pandas data frame
df = pd.Data Frame(dpts, col=columnList)

#perform clustering on the data frame
model = Clusteringmethod(clusterspecificparameters).fit(df)

#create a column in the data frame for the labels
df[“label”] = model.labels_.astype(str)

g = sns.PairGrid(ds, hue="label", hue_order=["0", "1"])
g.map(plt.scatter)
g.add_legend()
```

Interpretation/Evaluation



The KDD Process Model



[1]

Parallelization?

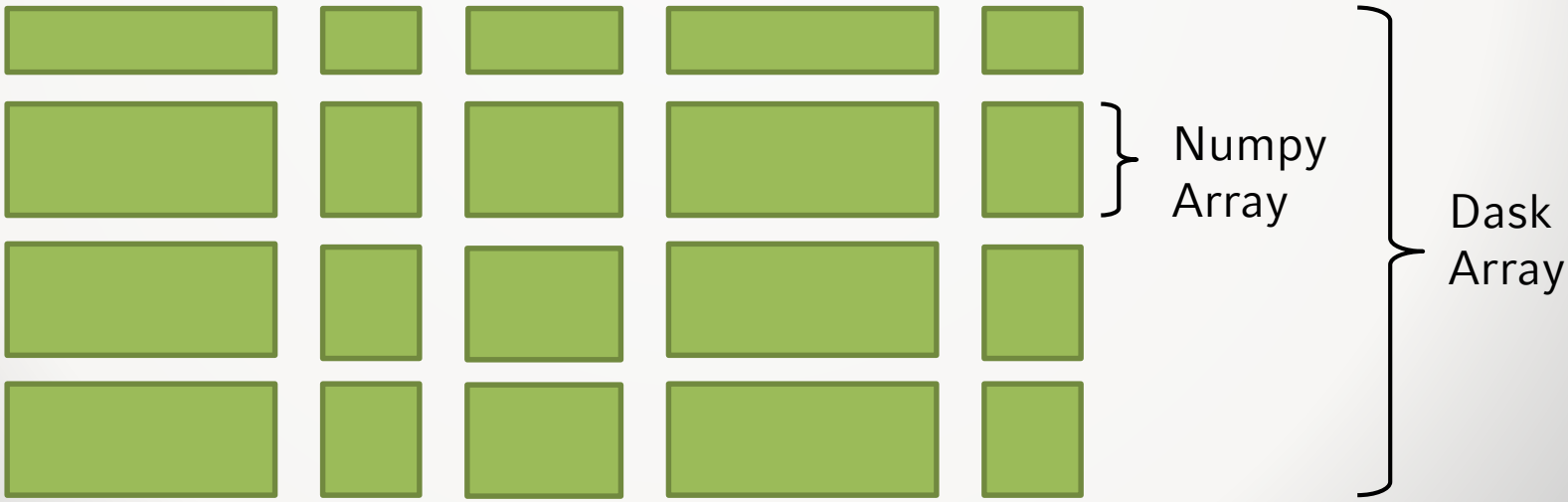
- Hadoop/Spark/Flink
  - With different ML Libraries on top

or **write same code as with NumPy/pandas/scikit-learn** with

- Dask ML
  - distributes computation on cluster

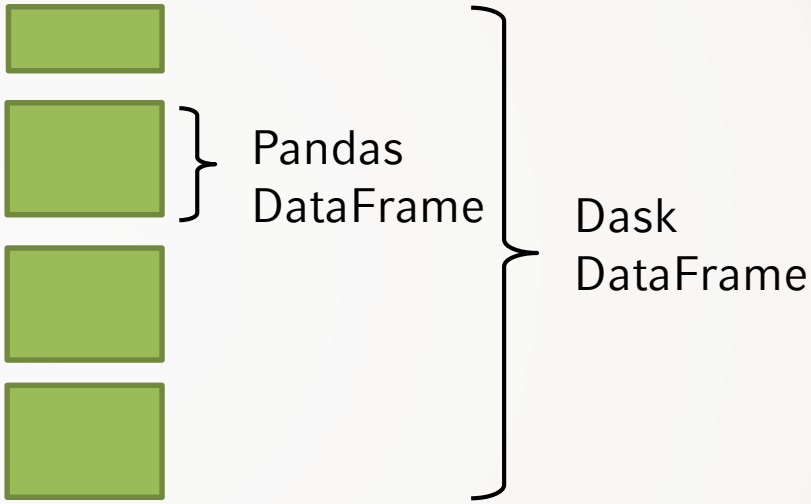
Dask arrays

```
import dask.array as da

x = da.random.random((10000, 10000), chunks=(1000, 1000))

y = x + x.T
z = y[::2, 5000:].mean(axis=1)


z.compute()
z.persist()
```



Dask dataframes

```
import dask.dataframe as dd
df = dd.read_parquet('...')

df2 = df[df.y > 0]
df2.persist()
df3 = df2.groupby('name').x.std()


df.loc['2000-01-05']
```



Scikit-learn with Dask (estimators)

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

param_grid = {"C": [0.001, 0.01, 0.1, 0.5, 1.0, 2.0, 5.0,
10.0],
              "kernel": ['rbf', 'poly', 'sigmoid'],
              "shrinking": [True, False]}

grid_search = GridSearchCV(SVC(gamma='auto',
random_state=0, probability=True),
                           param_grid=param_grid,
                           cv=3,
                           n_jobs=-1)

from sklearn.externals import joblib

with joblib.parallel_backend('dask'):
    grid_search.fit(X, y)
```

Beyond scikit-learn

```
import dask_ml.datasets
import dask_ml.cluster

X, y =
dask_ml.datasets.make_blobs(n_samples=10000000,
                            chunks=1000000,
                            random_state=0,
                            centers=3)
X = X.persist()

km = dask_ml.cluster.KMeans(n_clusters=3,
init_max_iter=2, oversampling_factor=10)
km.fit(X)
```

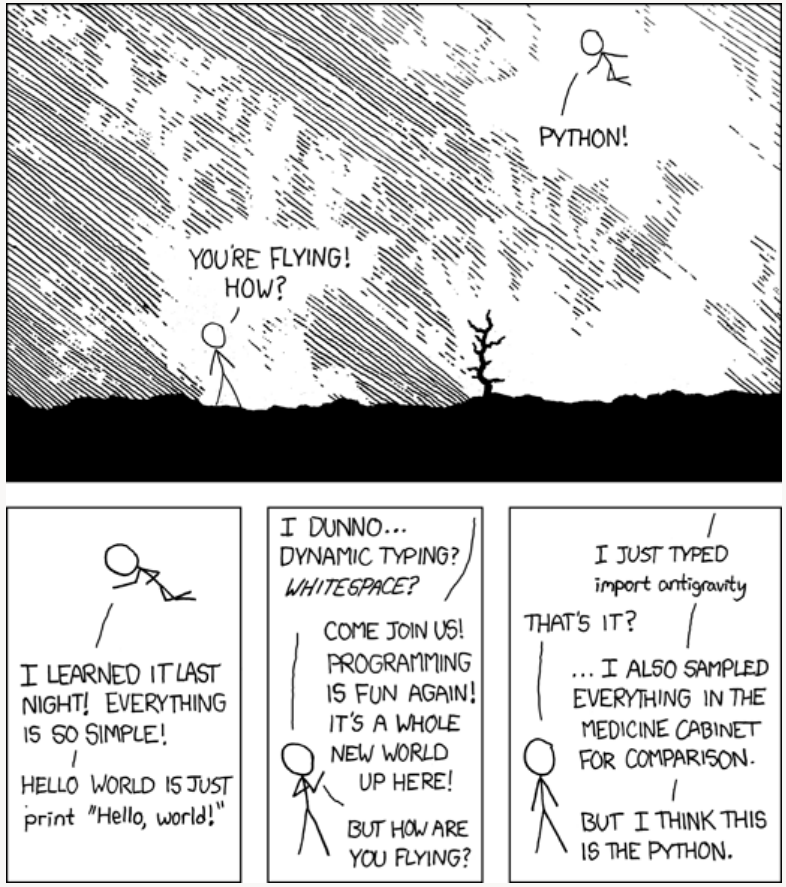
Finis

- Various frameworks available in context of the KDD process
- Some pitfalls → closer look at how things are implemented
- Most of the basic taks can be defined as templates
- We just scratched the surface
- Many more tools available
- There is no „holy grail“ package/solution

Finis

What are your experiences
with Python and data mining?

import antigravity



References

[0] https://xkcd.com/413/

[1] based on: Usama Fayyad, Gregory Piatetsky-Shapiro and Padhraic Smyth - The KDD Process for Extracting Useful Knowledge from Volumes of Data; Communications of the ACM (1996)