Coding Bootcamp Code in Python

PANDAS

What is it?

- Library for data science
  - defines datastructures
    - `Series` (1D)
    - `DataFrame` (2D)
  - defines algorithms
    - selection
    - pivot tables
  - defines utilities
    - visualization, e.g., `scatter_matrix`
- Backed by `numpy`
- Nice to experiment with data, jupyter notebooks

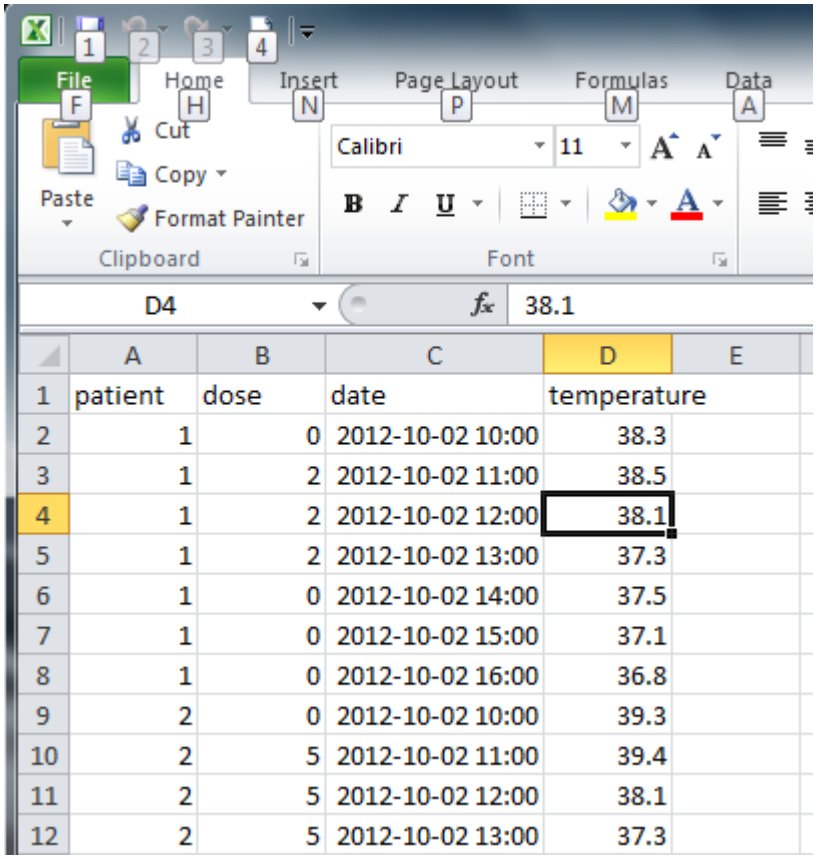
```
import pandas as pd
```

convention

R dataframes for Python

Example data

- Microsoft Excel spreadsheet



	A	B	C	D	E
1	patient	dose	date	temperature	
2	1	0	2012-10-02 10:00	38.3	
3	1	2	2012-10-02 11:00	38.5	
4	1	2	2012-10-02 12:00	38.1	
5	1	2	2012-10-02 13:00	37.3	
6	1	0	2012-10-02 14:00	37.5	
7	1	0	2012-10-02 15:00	37.1	
8	1	0	2012-10-02 16:00	36.8	
9	2	0	2012-10-02 10:00	39.3	
10	2	5	2012-10-02 11:00	39.4	
11	2	5	2012-10-02 12:00	38.1	
12	2	5	2012-10-02 13:00	37.3	

Read dataframe

- Create DataFrame from Excel file
  - Also tabular data, CSV, HDF5, SQL query, HTML page,…
- Show in notebook

```
In [3]: data = pd.read_excel('data/patients.xlsx')
        data
```

Out[3]:

	patient	dose	date	temperature
0	1	0	2012-10-02 10:00:00	38.3
1	1	2	2012-10-02 11:00:00	38.5
2	1	2	2012-10-02 12:00:00	38.1
3	1	2	2012-10-02 13:00:00	37.3
4	1	0	2012-10-02 14:00:00	37.5
5	1	0	2012-10-02 15:00:00	37.1
6	1	0	2012-10-02 16:00:00	36.8
7	2	0	2012-10-02 10:00:00	39.3
8	2	5	2012-10-02 11:00:00	39.4
9	2	5	2012-10-02 12:00:00	38.1
10	2	5	2012-10-02 13:00:00	37.3
11	2	0	2012-10-02 14:00:00	36.8
12	2	0	2012-10-02 15:00:00	36.8

Transform dataframe

- Patient data as columns, date as index: pivot table

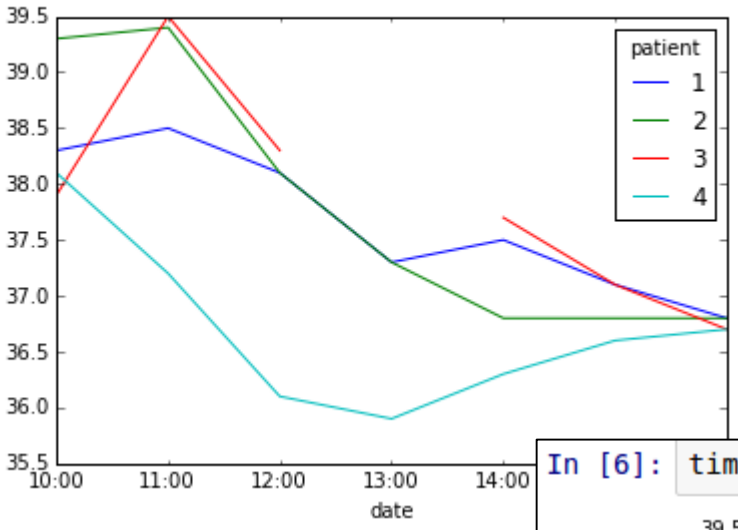
```
In [4]: timeseries = data.pivot_table(index='date', columns=['patient'])
        timeseries
```

Out[4]:

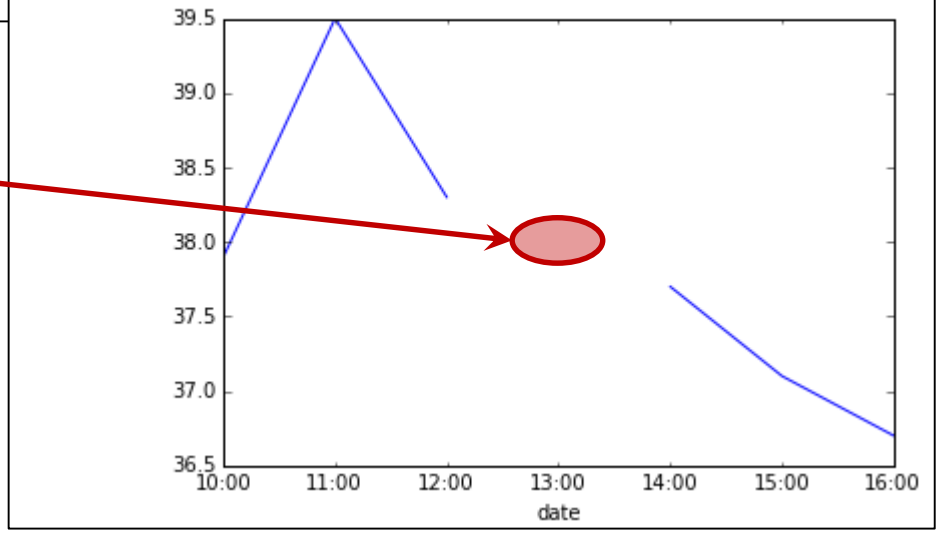
	dose				temperature			
patient	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
date								
2012-10-02 10:00:00	0	0	0	0	38.3	39.3	37.9	38.1
2012-10-02 11:00:00	2	5	2	5	38.5	39.4	39.5	37.2
2012-10-02 12:00:00	2	5	5	5	38.1	38.1	38.3	36.1
2012-10-02 13:00:00	2	5	2	0	37.3	37.3	NaN	35.9
2012-10-02 14:00:00	0	0	2	NaN	37.5	36.8	37.7	36.3
2012-10-02 15:00:00	0	0	2	0	37.1	36.8	37.1	36.6
2012-10-02 16:00:00	0	0	0	0	36.8	36.8	36.7	36.7

Plot data

```
In [5]: timeseries['temperature'].plot();
```




```
In [6]: timeseries['temperature', 3].plot();
```

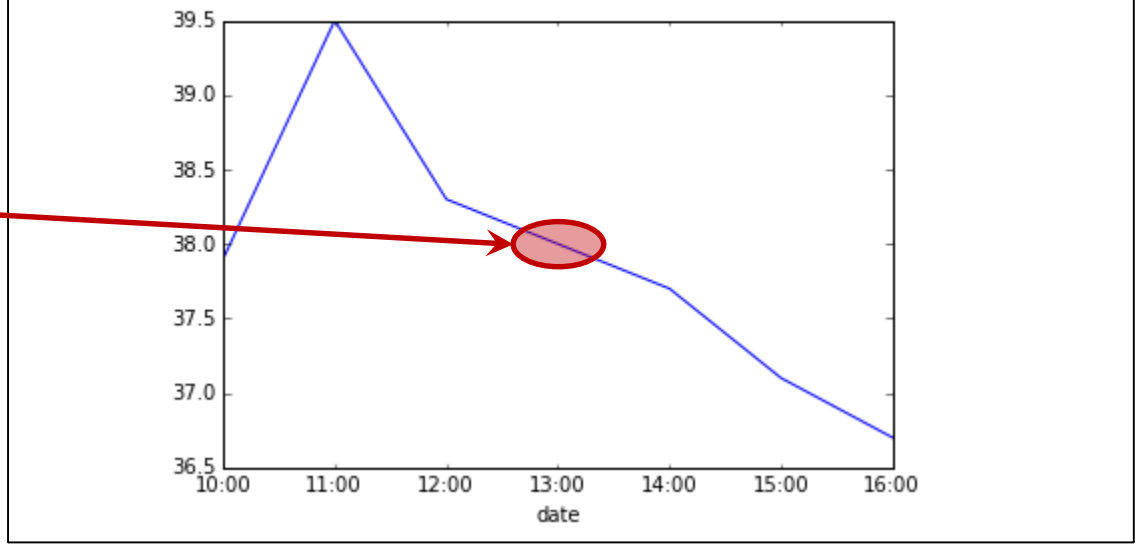


missing value

Missing data: NaNs

- Can be filled with 0, other value, or interpolated

```
In [9]: interpolated = timeseries['temperature', 3].interpolate()
        interpolated.plot();
```



interpolated value

In-place changes

- Changes produce new `DataFrame`
  - good to experiment
  - bad for performance/memory usage
- Use in place

```
In [10]: timeseries.interpolate(inplace=True)
         timeseries
```

Out[10]:

	dose				temperature				avg_temperature
patient	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	
date									
2012-10-02 10:00:00	0	0	0	0	38.3	39.3	37.9	38.1	38.400000
2012-10-02 11:00:00	2	5	2	5	38.5	39.4	39.5	37.2	38.650000
2012-10-02 12:00:00	2	5	5	5	38.1	38.1	38.3	36.1	37.650000
2012-10-02 13:00:00	2	5	2	0	37.3	37.3	38.0	35.9	36.833333
2012-10-02 14:00:00	0	0	2	0	37.5	36.8	37.7	36.3	37.075000
2012-10-02 15:00:00	0	0	2	0	37.1	36.8	37.1	36.6	36.900000
2012-10-02 16:00:00	0	0	0	0	36.8	36.8	36.7	36.7	36.750000

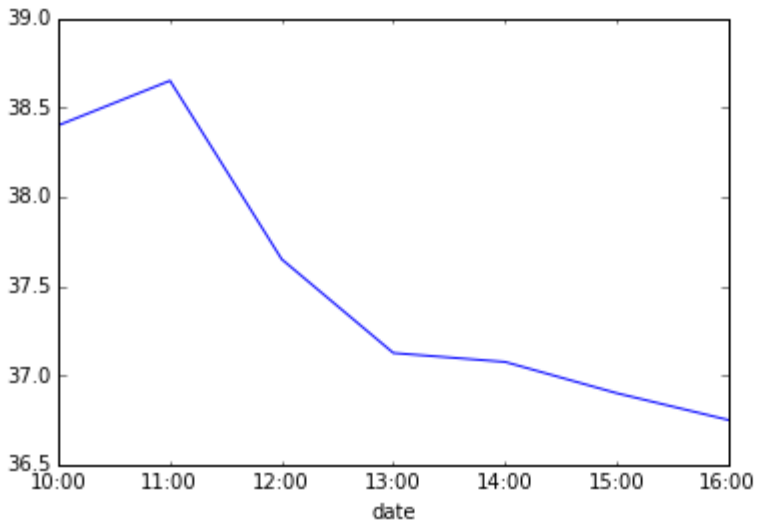
Statistics & adding columns

- Compute average temperature and add as column

add column

compute statistics

```
In [11]: timeseries['avg_temperature'] = timeseries['temperature'].mean(axis=1)
         timeseries['avg_temperature'].plot();
```



Cumulative sum

- Dose is better expressed cumulatively

compute sums

modify column

```
In [16]: for patient_id in timeseries['dose'].columns:
             timeseries['cum_dose', patient_id] = timeseries['dose', patient_id].cumsum()
         timeseries
```

Out[16]:

	dose				temperature				avg_temperature	cum_dose			
patient	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**		**1**	**2**	**3**	**4**
date													
2012-10-02 10:00:00	0	0	0	0	38.3	39.3	37.9	38.1	38.400	0	0	0	0
2012-10-02 11:00:00	2	5	2	5	38.5	39.4	39.5	37.2	38.650	2	5	2	5
2012-10-02 12:00:00	2	5	5	5	38.1	38.1	38.3	36.1	37.650	4	10	7	10
2012-10-02 13:00:00	2	5	2	0	37.3	37.3	38.0	35.9	37.125	6	15	9	10
2012-10-02 14:00:00	0	0	2	0	37.5	36.8	37.7	36.3	37.075	6	15	11	10
2012-10-02 15:00:00	0	0	2	0	37.1	36.8	37.1	36.6	36.900	6	15	13	10
2012-10-02 16:00:00	0	0	0	0	36.8	36.8	36.7	36.7	36.750	6	15	13	10

More pivot & query

- `pivot_table` **and** `query` **are powerful!**

```
In [28]: data.pivot_table(index=['patient'], values=['dose', 'temperature'],
                          aggfunc={'dose': np.sum, 'temperature': np.max})
```

Out[28]:

	dose	temperature
patient		
1	6	38.5
2	15	39.4
3	13	39.5
4	10	38.1

What is total dose versus maximum temperature for each patient?

```
In [43]: data.query('temperature > 39.0').loc[:, ['patient', 'date', 'temperature']]
```

Out[43]:

	patient	date	temperature
7	2	2012-10-02 10:00:00	39.3
8	2	2012-10-02 11:00:00	39.4
15	3	2012-10-02 11:00:00	39.5

Which patients had temperature > 39, and when?

Reading HTML tables

```
In [14]: genes_data = pd.read_html('data/genes.html', index_col=0, header=0)
         genes = genes_data[0]
         genes.columns = [int(x) for x in genes.columns]
         genes
```

Out[14]:

	1	2	3	4
FXDG	0.010199	0.042988	0.831946	-0.023656
VTUR	0.466012	0.494679	0.806661	0.518115
OVAH	0.783847	0.754150	0.208352	0.826368
AKSE	0.922433	0.929716	0.662824	0.959443
SJNN	0.325089	0.302198	0.522847	0.356346
LVKM	0.554702	0.531044	0.138988	0.600229

column names

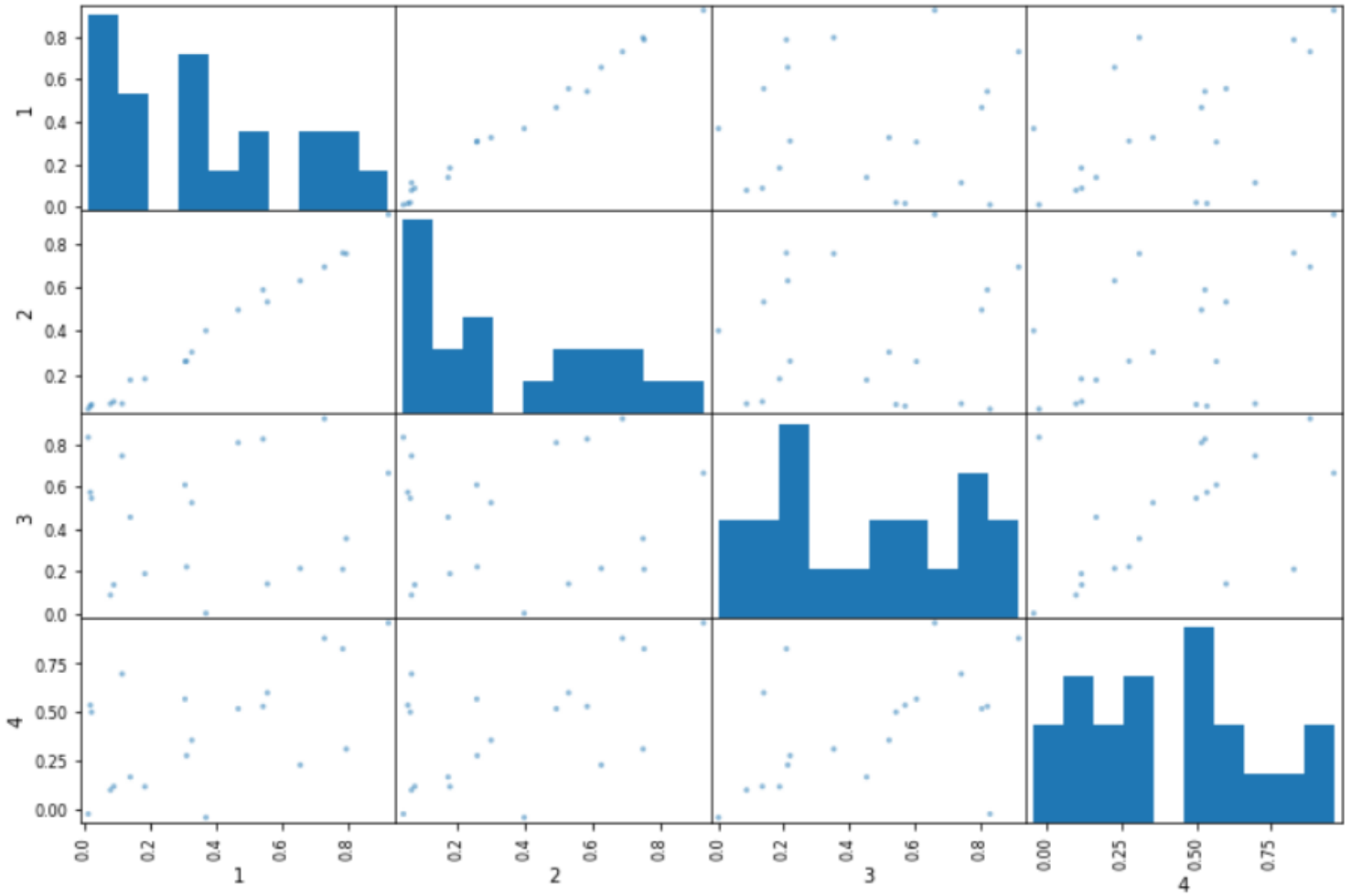
index column

`read_html` produces `list` of `DataFrame`, one per HTML table on page

Column names are `str` by default, converted to `int` for consistency with running example

Scatter matrix

```
In [24]: pd.plotting.scatter_matrix(genes, figsize=(12, 8));
```



Computing correlations

Creating `DataFrames` by hand

In [17]:
```
indices = list(genes.columns)
genes_corr = pd.DataFrame(index=indices, columns=indices)
genes_p_value = pd.DataFrame(index=indices, columns=indices)
for id1 in genes.columns:
    for id2 in genes.columns:
        genes_corr[id1][id2], genes_p_value[id1][id2] = stats.pearsonr(genes[id1], genes[id2])
```

In [18]:
```
genes_corr
```

Out[18]:

	1	2	3	4
1	1	0.9936518	0.04567655	0.5474551
2	0.9936518	1	0.0728375	0.5373312
3	0.04567655	0.0728375	1	0.4850267
4	0.5474551	0.5373312	0.4850267	1

Note:
`import scipy.stats as stats`

In [19]:
```
genes_p_value
```

Out[19]:

	1	2	3	4
1	0	1.554131e-18	0.8483549	0.01247343
2	1.554131e-18	0	0.7602363	0.01455602
3	0.8483549	0.7602363	0	0.03018841
4	0.01247343	0.01455602	0.03018841	0

Code Pack 29

Pandas

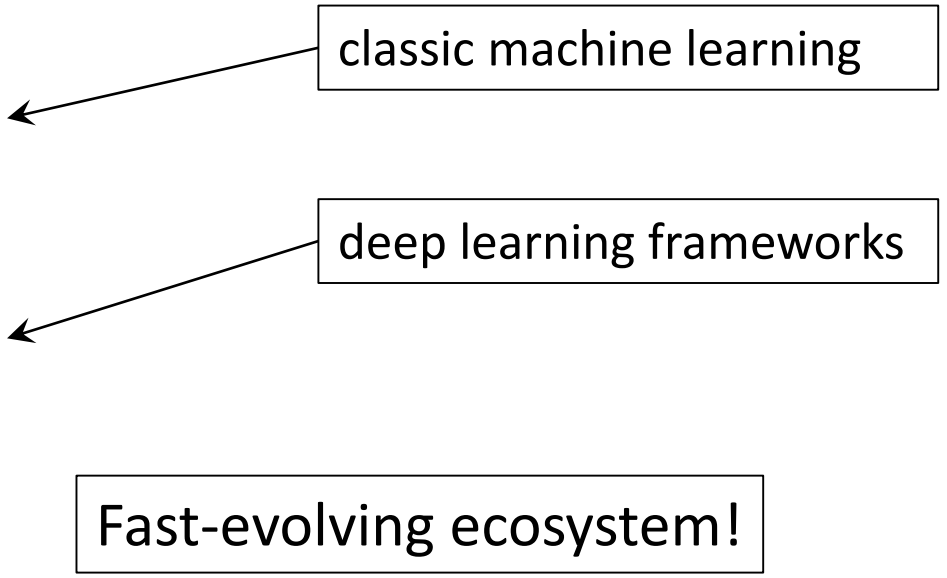
Not this one:

Coding Bootcamp Code in Python

MACHINE LEARNING

Introduction

- Machine learning is making great strides
  - Large, good data sets
  - Progress in algorithms
- Many libraries
  - scikit-learn
  - PyTorch
  - TensorFlow
  - Keras
  - …



scikit-learn

- Nice end-to-end framework
  - data exploration (+ pandas + holoviews)
  - data preprocessing (+ pandas)
    - cleaning/missing values
    - normalization
  - training
  - testing
  - application
- Classic only?

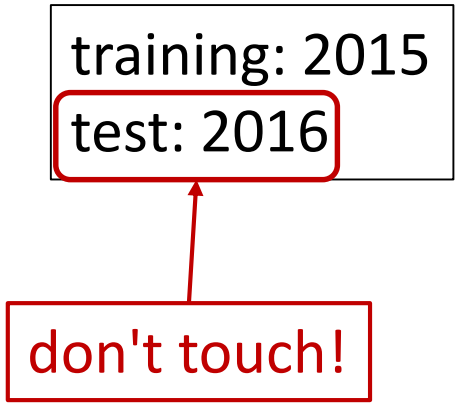
Sure, but don't jump into deep end *unless* you can swim!

Machine learning tasks

- Supervised learning
  - regression: predict numerical values
  - classification: predict categorical values, i.e., labels
- Unsupervised learning
  - clustering: group data according to "distance"
  - association: find frequent co-occurrences
  - link prediction: discover relationships in data
  - data reduction: project features to fewer features
- Reinforcement learning

Data set

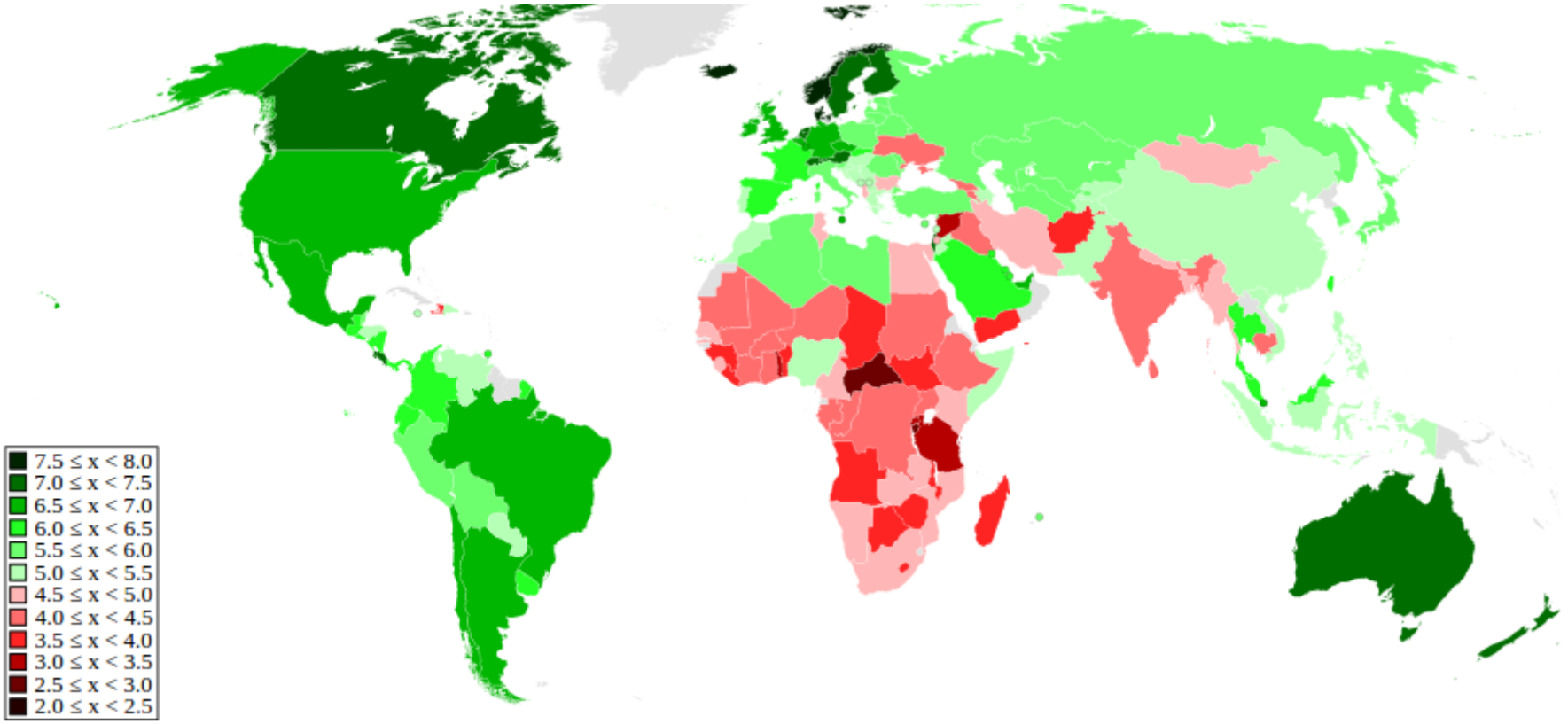
- World happiness index 2015 & 2016 (http://worldhappiness.report/)
- Happiness score for country based on
  - economic factors (GDP)
  - family situation (social network)
  - health care (life expectancy)
  - freedom
  - trust (government corruption)
  - generosity
  - dystopia residual
- Geographical region, e.g., Western Europe, Southern Asia,…



quantitative data

categorical data

World happiness



https://en.wikipedia.org/wiki/World_Happiness_Report

Task 1

- Given
  - economy
  - family
  - health
  - freedom
  - trust
  - generosity
  - dystopia residual
  - region
- Predict happiness score

Regression

Let's peek…

```
In [4]: data_2015.describe()
```

Out[4]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom	Trust (Government Corruption)	Generc
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615	0.143422	0.237
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693	0.120034	0.126
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330	0.061675	0.150
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515	0.107220	0.216
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092	0.180255	0.309
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730	0.551910	0.795

Rescaling

Missing values?

```
In [5]: data_2015.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158 entries, 0 to 157
Data columns (total 12 columns):
Country                          158 no
Region                           158 no
Happiness Rank                   158 no
Happiness Score                  158 no
Standard Error                   158 no
Economy (GDP per Capita)         158 no
Family                           158 no
Health (Life Expectancy)         158 no
Freedom                          158 no
Trust (Government Corruption)    158 no
Generosity                       158 no
Dystopia Residual                158 no
dtypes: float64(9), int64(1), object(2)
memory usage: 14.9+ KB
```

```
In [63]: data_2015.count()
Out[63]: Country                          158
         Region                           158
         Happiness Rank                   158
         Happiness Score                  158
         Standard Error                   158
         Economy (GDP per Capita)         158
         Family                           158
         Health (Life Expectancy)         158
         Freedom                          158
         Trust (Government Corruption)    158
         Generosity                       158
         Dystopia Residual                158
         dtype: int64
```

No NaNs, otherwise, use `Imputer`

Extracting data

- Part of machine learning pipeline
  - fit + transform
- Extracting columns from pandas data frame
- Transform data
  - scale numerical features to [0, 1]
  - one-hot encoding for regions

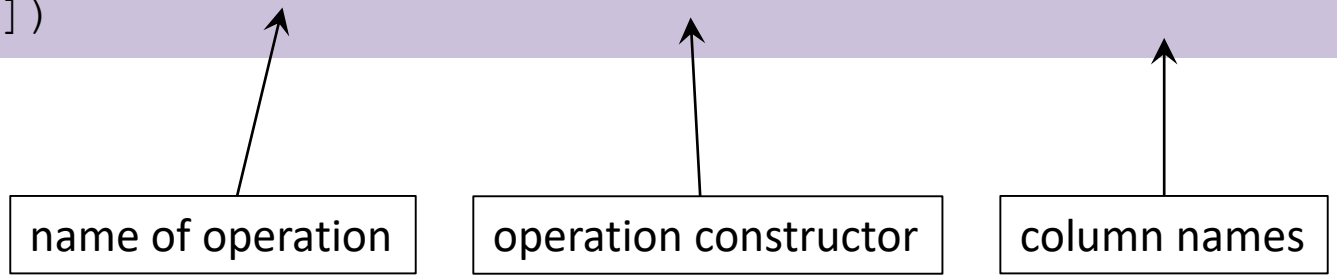
Numerical attributes pipeline

- Extract, then scale

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler

num_attrs = data_2015.columns[5:13]

num_attrs_transformer = ColumnTransformer([
    ('minimax', MinMaxScaler(), num_attr_names)
])
```



Categorical attribute pipeline

- Extract, then create one-hot attributes

Country	Region
Belgium	Western Europe
Canada	North America
Germany	Western Europe
…	…

Country	Western Europe	Northren America	…
Belgium	1	0	…
Canada	0	1	…
Germany	1	0	…
…	…	…	…

```
from sklearn.preprocessing import OneHotEncoder

region_transformer = ColumnTransformer([
    ('one_hot_encoder', OneHotEncoder(categories='auto'),
     ['Region'])
])
```

Combining pipelines & execution

- Both pipelines must be executed, results combined

```
from sklearn.pipeline import FeatureUnion

preparation_pipeline = FeatureUnion(transformer_list=[
    ('region_attr', region_attr_transformer),
    ('num_attrs', num_attrs_transformer),
])
```

- Run data through pipeline

```
train_data = preparation_pipeline.fit_transform(data_2015)
```

numpy array

Ready to start training!

Training & prediction

- Create learning algorithm

```
from sklearn.linear_model import Ridge

ridge_regr = Ridge(alpha=0.5, fit_intercept=False)
```

hyperparameters

- Train

```
X = train_data
Y = np.array(data_2015['Happiness Score'])
ridge_regr.fit(X, Y);
```

???

- Predict

```
Y_ridge_regr = ridge_regr.predict(X)
```

Score & errors

- Score

```
ridge_regr.score(X, Y)
```

→

```
0.9984
```

- Better: cross validation

```
scores = cross_val_score(
       ridge_regr, X, Y,
       scoring='neg_mean_squared_error',
       cv=10
)
```

10-fold

```
np.sqrt(-scores)
```

→

```
0.1954,  0.0215,
0.0579,  0.0478,
0.0551,  0.0649,
0.0591,  0.0451,
0.0729,  0.0803
```

Fine tuning

- Define hyper parameter search space

```
grid_params = [
    {'alpha': [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5],
     'fit_intercept': [True, False]},
]
```

- Define grid searcher

```
grid_search = GridSearchCV(ridge_regr, grid_params, cv=10,
                           scoring='neg_mean_squared_error')
```

- Search

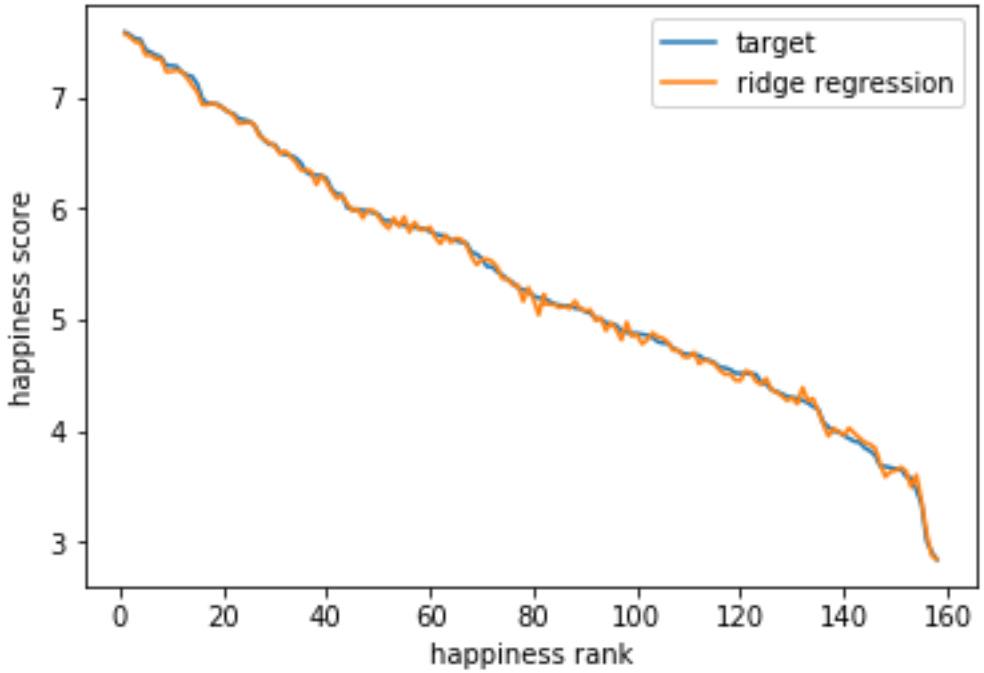
```
grid_search.fit(X, Y)
```

- Best hyper parameters

```
grid_search.best_params_
```

→ `{'alpha': 0.0005, 'fit_intercept': True}`

Training result



Testing the model

- Use pipeline

```
test_data = preparation_pipeline.transform(data_2016)
```

Note: only `transform`, no `fit_transform`!

- Predict

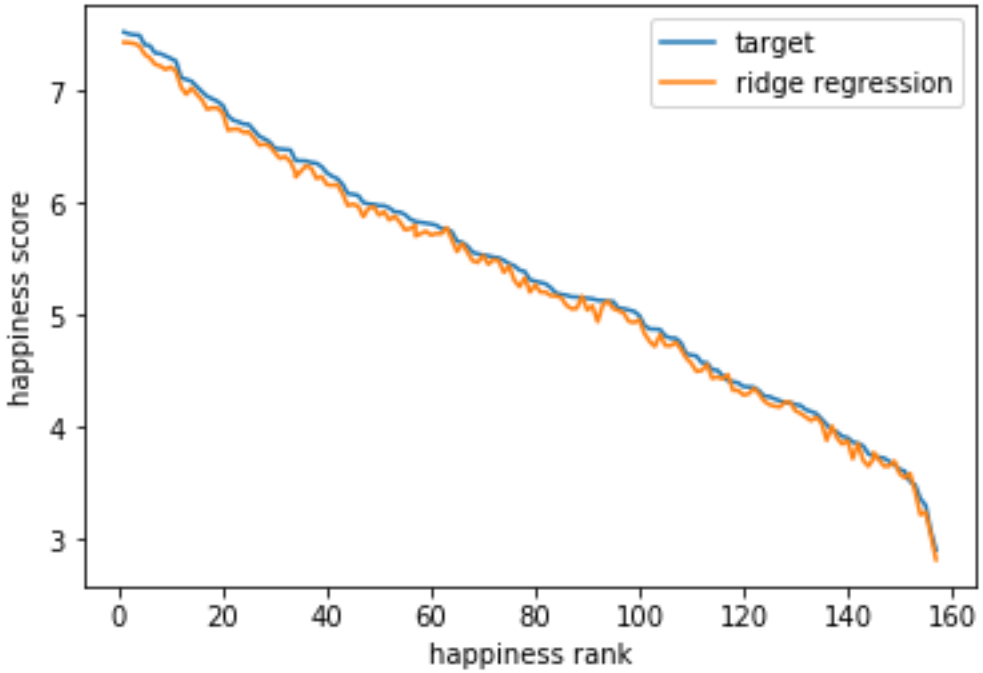
```
Y_test = ridge_regr.predict(X_test)
```

- Score

```
ridge_regr.score(X_test, Y_test)
```

→ `0.9950`

Test result


target
ridge regression
happiness score
7
6
5
4
3
0
20
40
60
80
100
120
140
160
happiness rank

Code Pack 30

1.regression_world_happiness

Task 2

- Given
  - economy
  - family
  - health
  - freedom
  - trust
  - generosity
  - dystopia residual
- Predict whether country in Western Europe

Classification

Data preparation

- Numerical pipeline is same as in task 1
  - select from pandas `DataFrame`
  - scale

```
prepared_data = num_attrs_pipeline.fit_transform(data_2015)
```

- Add column for class
  - `True` if in Western Europe, `False` otherwise

```
data_2015['Western Europe'] = (data_2015['Region'] ==
                               'Western Europe')
```

Training & scoring

- Create learning algorithm

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
```

attributes: approx. Gaussian distr.

- Train

```
nb.fit(prepared_data, data_2015['Western Europe'])
```

- Scoring

```
nb.score(prepared_data, data_2015['Western Europe'])
```

Seems reasonable → `0.9367`

Testing the model

- Use pipeline

```
test_data = num_attrs_pipeline.transform(data_2016)
```

Note: only `transform`, no `fit_transform`!

- Predict

```
Y_test = nb.predict(test_data)
```

- Score

```
nb.score(test_data, Y_test)
```

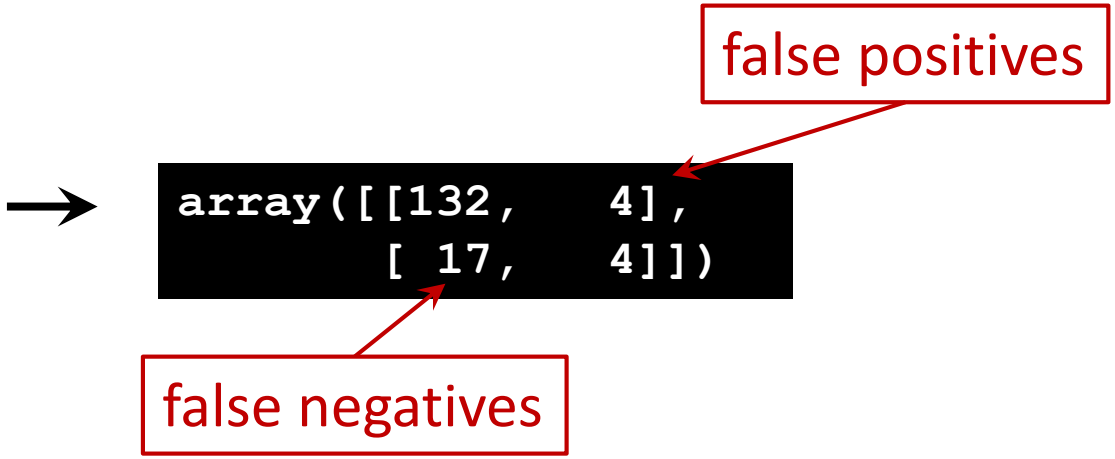
→ `0.8662`

Not so good

Actually…

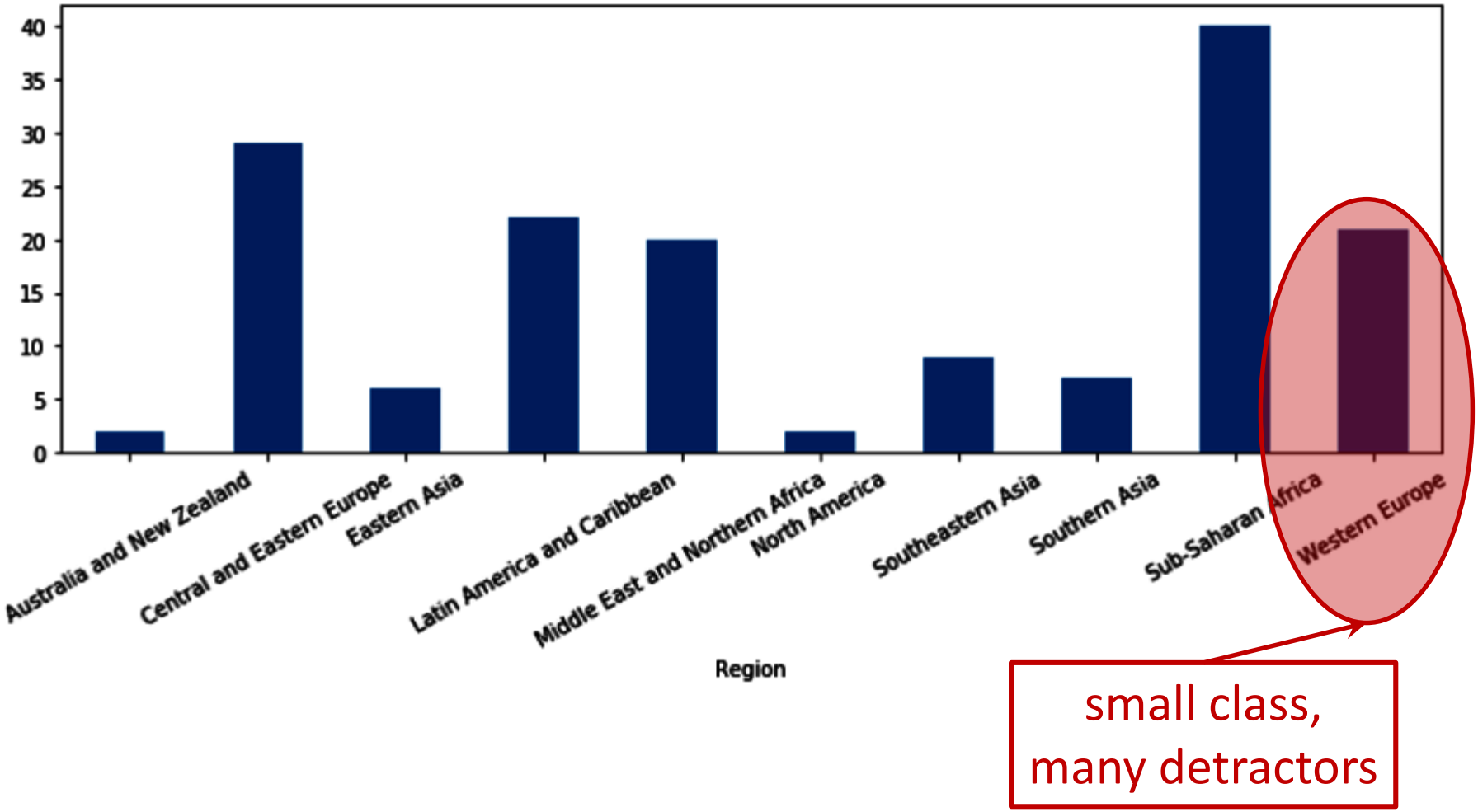
- Compute confusion matrix

```
confusion_matrix(data_2016['Western Europe'],
                 nb.predict(test_data))
```



Massive exit from Western Europe: *17* countries just left!

Data set properties



Code Pack 30

~~1.regression_world_happiness~~

2.classification_finding_regions

Task 3

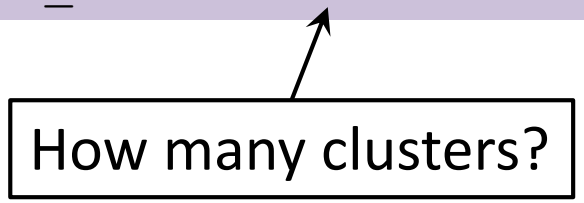
- Given
  - economy
  - family
  - health
  - freedom
  - trust
  - generosity
  - dystopia residual
- Find countries that are "close"

Clustering

Data preparation & Training

- Data preprocessing same as task 2
- Create learning algorithm

```
from sklearn.cluster import KMeans
k_means = KMeans(n_clusters=3)
```



- Train

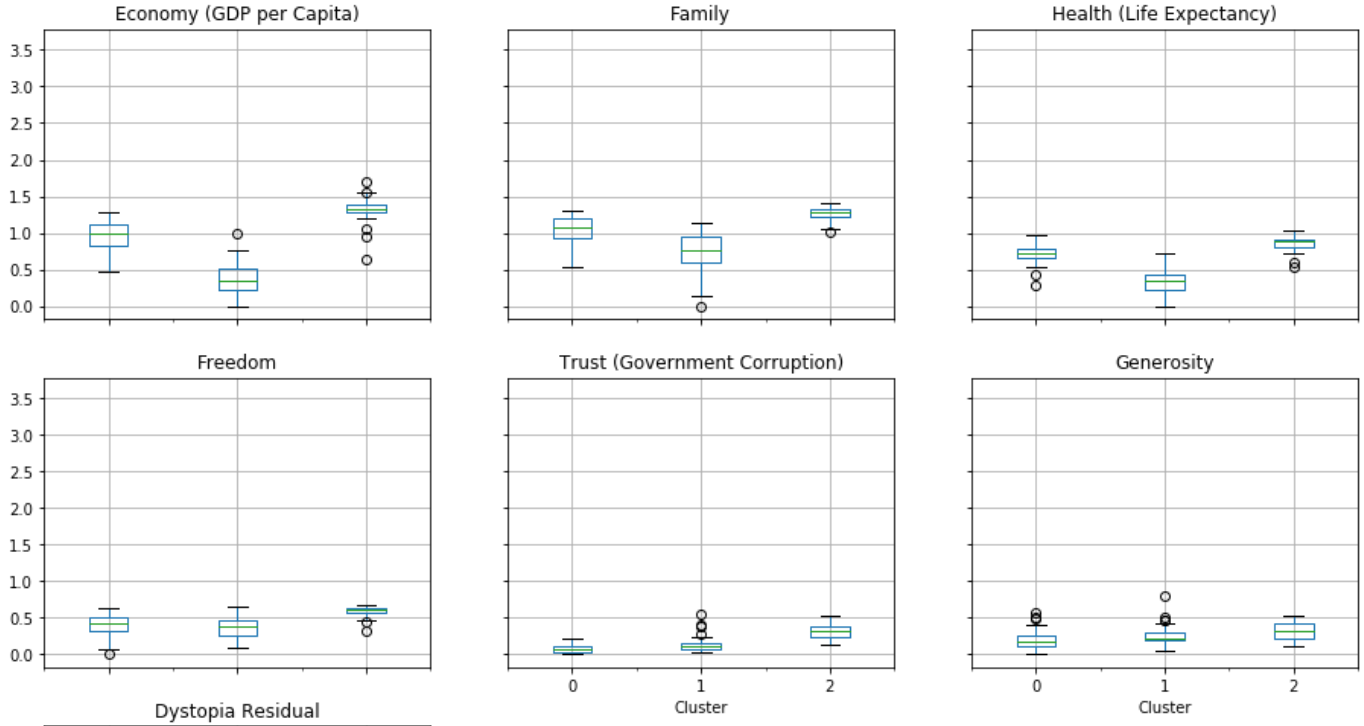
```
k_means.fit(prepared_data)
```

- Add cluster label to data set

```
data_2015['Cluster'] = 0
for i in range(3):
    data_2015.loc[clusterer.labels_ == i, 'Cluster'] = i
```

Examine clusters

```
data_2015.boxplot(by='Cluster', column=num_attr_names)
```




```
data_2015.loc[:, ('Cluster', 'Country')].groupby('Cluster').count()
```

Cluster	Country
0	77
1	50
2	31

Code Pack 30

~~1.regression_world_happiness~~

~~2.classification_finding_regions~~

3.cluster_countries