7. Insight into Machine Learning

Several more advanced applications rely on Pandas structure to work. One example is the package scikit-learn, which has become one of the dominant machine learning resource in data science. We are now going to have a very quick look at how Pandas is used in that frame.

We are going to work again with our swiss towns infos and we will see if we can predict the result of a party based on that information.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import seaborn as sns
```

7.1 Data loading and features selection

We first load the data set:

In [2]:
```
towns = pd.read_excel('Datasets/2018.xls', skiprows=list(range(5))+list
(range(6,9)),
                      skipfooter=34, index_col='Commune',na_values=['*
','X'])
towns = towns.reset_index()
```

Now we have to get select what featurs we are going to use to predict the vote for UDC. We have to remove the results of the other parties, as those are of course correlated.

We also create a target by selecting only the UDC column

In [3]:
```
features = towns.drop('PDC', axis=1)
features = features.drop('PS', axis=1)
features = features.drop('Commune', axis=1)
features = features.drop('PVL', axis=1)
features = features.drop('PLR 2)', axis=1)
features = features.drop('PBD', axis=1)
features = features.drop('PST/Sol.', axis=1)
features = features.drop('PEV/PCS', axis=1)
features = features.drop('PES', axis=1)
features = features.drop('Petits partis de droite', axis=1)
features = features.drop('Code commune', axis=1)

features = features.dropna()
targets = features['UDC']
features = features.drop('UDC', axis=1)
```

In [4]:
```
features.head()
```

Out[4]:

	Habitants	Variation en %	Densité de la population par km²	Etrangers en %	0-19 ans	20-64 ans	65 ans ou plus	Taux brut de nuptialité	Ta div
0	1977	8.388158	249.936789	13.100658	20.586748	62.822458	16.590794	2.526529	3.03
1	11900	7.294203	1123.701605	27.848740	20.285714	62.201681	17.512605	5.167740	1.44
2	5435	5.349874	731.493943	14.149034	23.808648	60.717571	15.473781	5.389834	1.85
3	3571	6.279762	262.573529	14.533744	22.738729	60.403248	16.858023	4.540295	1.98
4	3687	8.123167	564.624809	14.971522	22.484405	62.110117	15.405479	7.622159	1.36

5 rows × 31 columns

In [5]:
```
targets.head()
```

Out[5]:
```
0    30.929249
1    33.785785
2    29.100156
3    34.937369
4    30.114599
Name: UDC, dtype: float64
```

7.2 Splitting the data

We need to be able to test whether our ML algorithm is capable of making predictions on data is has not been trained on. We therefore split our dataset into a training and a testing set. Luckily sklearn provides this out of the box if we pass the right dataframes.

In [6]:
```
from sklearn.model_selection import train_test_split
X, X_test, y, y_test = train_test_split(features, targets,
                                        test_size = 0.2,
                                        random_state = 42)
```

In [7]:
```
len(X_test)/len(X)
```

Out[7]:
```
0.25
```

7.3 Choosing an ML method

Sklearn offers a wide range of ML methods. We are not entering into details here and choose a Random Forest regression:

In [8]:
```
from sklearn.ensemble import RandomForestRegressor
```

Then we instantiate the model and train it (fit):

In [9]:
```
random_forrest = RandomForestRegressor(n_estimators=1000)
random_forrest.fit(X, y)
```

Out[9]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', 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=1000, n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=Fals
e)
```

Finally we can use it to make predictions. In particular we can apply it to our test sample and see how it performs:

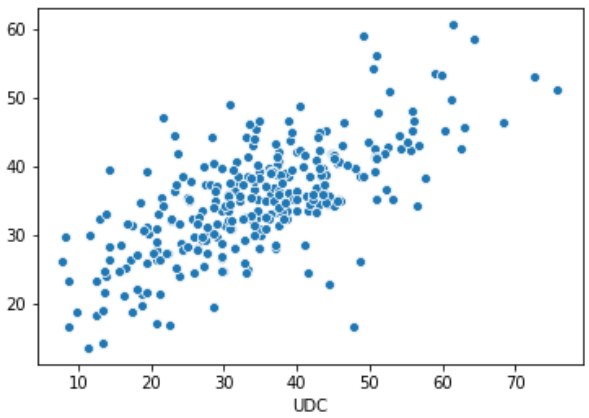
In [10]:
```
predictions = random_forrest.predict(X_test)
mae = np.mean(abs(predictions - y_test))
print(mae)
```

```
7.092857784914048
```

```
/usr/local/lib/python3.5/dist-packages/pandas/core/computation/check.py:1
9: UserWarning: The installed version of numexpr 2.4.3 is not supported i
n pandas and will be not be used
The minimum supported version is 2.6.1

  ver=ver, min_ver=_MIN_NUMEXPR_VERSION), UserWarning)
```

In [11]:
```
sns.scatterplot(x = y_test, y = predictions);
```



In [12]:
```
import scipy.stats
```

In [13]:
```
scipy.stats.pearsonr(y_test,predictions)
```

Out[13]:
```
(0.6900691335750526, 1.8302171574366992e-43)
```

7.4 Important features

A random forest classificer has the advantage that it can provide us information about how important each feature is. In other terms which features help the most in predicting:

In [14]:
```
print(random_forrest.feature_importances_)
```

```
[0.01642088 0.03087361 0.02929625 0.07733401 0.05875088 0.06434162
 0.02267082 0.03147107 0.02621289 0.02242997 0.02138494 0.01213671
 0.03465807 0.0244234  0.03648381 0.02957084 0.13884786 0.02812714
 0.02704483 0.02524017 0.01225019 0.02748501 0.01709968 0.01442152
 0.00932912 0.0553376  0.00979078 0.01229114 0.03309209 0.01542136
 0.03576174]
```

The larger the number, the better its predictions power. We can sort this list and see to what features they correspond in our feature Dataframe:

In [15]:
```
features.keys()[np.argsort(random_forrest.feature_importances_)]
```

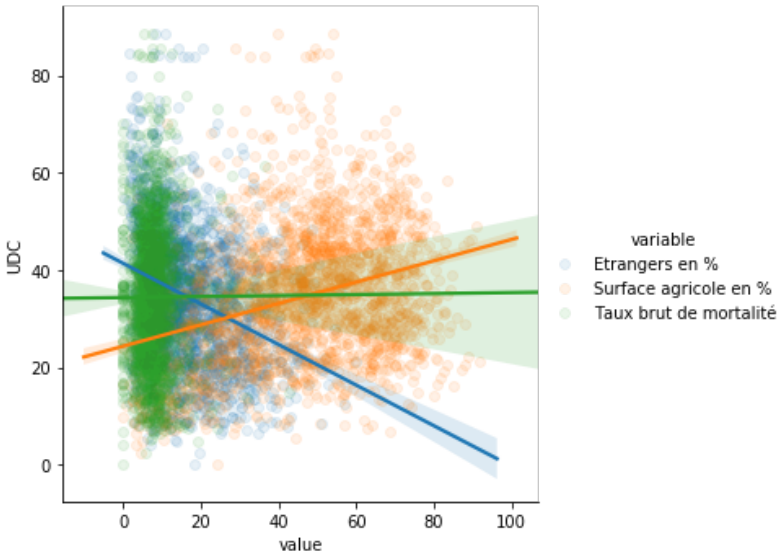
Out[15]:
```
Index(['Etablissements total', 'Secteur secondaire.1', 'Ménages privés',
       'Emplois total', 'Secteur tertiaire.1', 'Secteur tertiaire',
       'Nouveaux logements construits pour 1000 habitants', 'Habitants',
       'Secteur secondaire', 'Taux brut de mortalité', 'Taux brut de natalité',
       '65 ans ou plus', 'Surface totale en km²', 'Surface improductive en %',
       'Taux brut de divortialité', 'Surface boisée en %', 'Secteur primaire',
       'Variation en ha.1', 'Densité de la population par km²',
       'Variation en ha', 'Variation en %', 'Taux brut de nuptialité',
       'Taux de logements vacants', 'Taille moyenne des ménages en personnes',
       'Taux d'aide sociale', 'Surfaces d'habitat et d'infrastructure en %',
       'Secteur primaire.1', '0-19 ans', '20-64 ans', 'Etrangers en %',
       'Surface agricole en %'],
      dtype='object')
```

Finally, we can have a look at the actual correlations that seem to be indicated here. For this we select a few features and create a long format table for plotting:

In [16]:
```
towns_melt = pd.melt(towns, id_vars=['Commune','UDC'],
        value_vars=['Etrangers en %','Surface agricole en %','Taux brut de mortalité'])
```

In [17]:
```
sns.lmplot(data = towns_melt, x = 'value', y = 'UDC', hue = 'variable',
scatter_kws={'alpha' : 0.1});
```

```
/usr/local/lib/python3.5/dist-packages/scipy/stats/stats.py:1713: FutureW
arning: Using a non-tuple sequence for multidimensional indexing is depre
cated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this wi
ll be interpreted as an array index, `arr[np.array(seq)]`, which will res
ult either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



There are indeed strong correlations where they are expected! Notice alos that we can learn things here: the right-wing party is most successful where there's the least foreigners...