In [88]:

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")
sns.set_context("notebook")
sns.set_context("poster")
```

Data Preprocessing

Most of Machine Learning algorithms make assumptions on your data, for example that the scales are comparable or simply they work only on numerical data. This implies that we need to pre-process the data. User oriented applications, such as BigML, do that automatically. However, when you use a language such as Python or R, you have to do it manually and decide for each attribute.

In the Machine Learning process (see figure below) pre-processing is the first step after loading and examining your data.

There are 4 basic processes that we will treat separatelty. Depending on the algorithm that we will use, we'll need to apply all of them or only some:

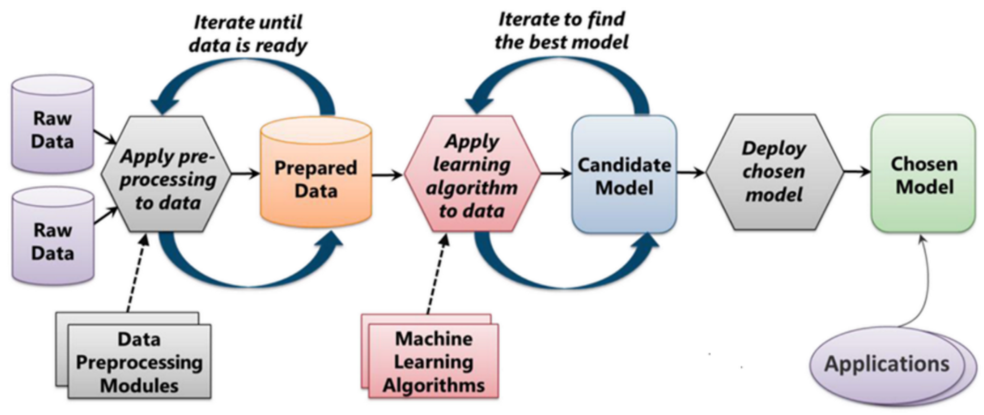
```
1) Rescale data.
2) Standarize data.
3) Normalize data.
4) Binarize data.
```

Before the pre-processing there are three(3) important steps:

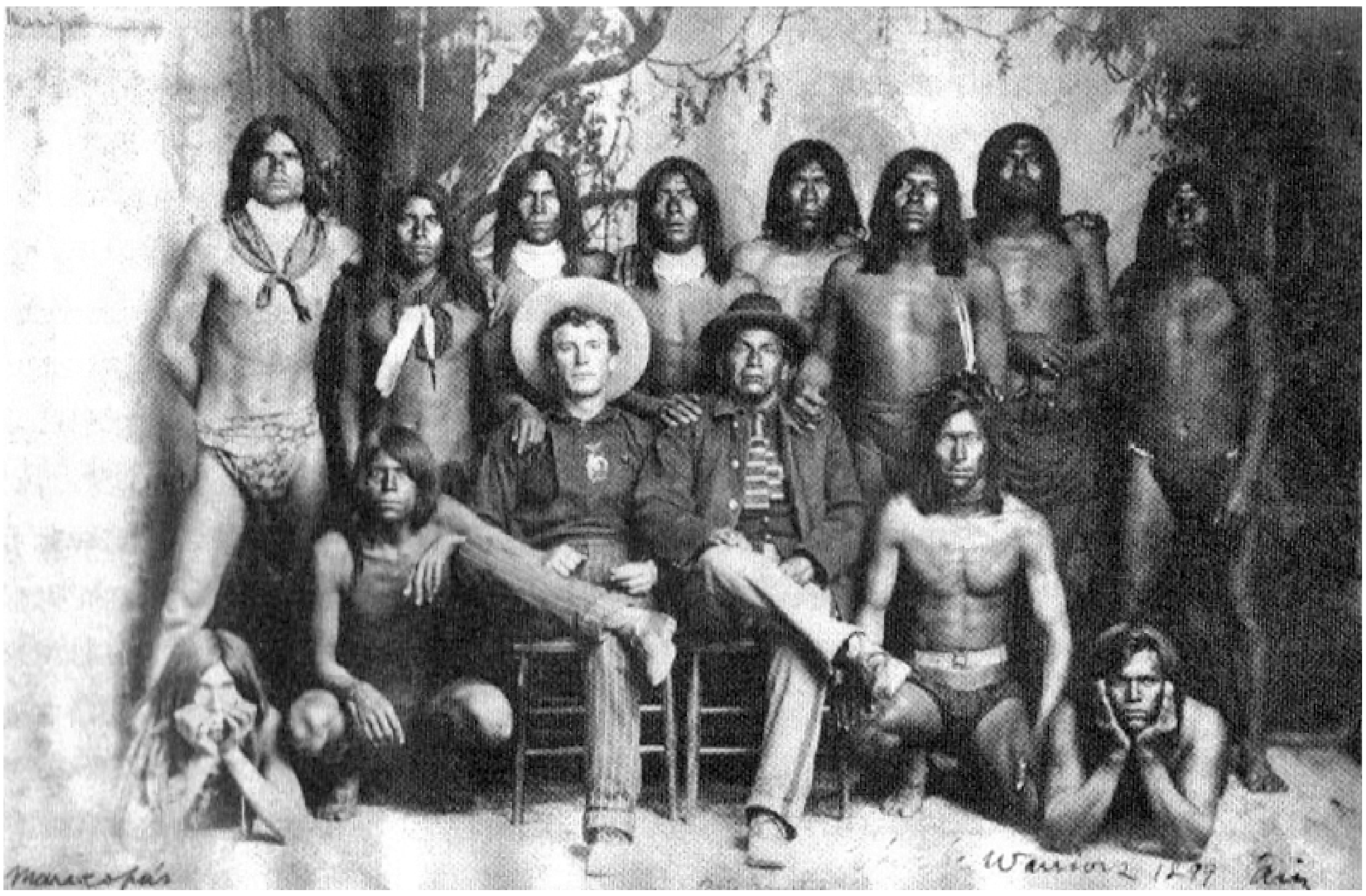
```
a) Load your dataset
b) Examine it and get rid of everything that doesn't apply.
c) Split the dataset into the input and output variables.
```

You will observe that scikit-learn provides two equivalent ways. First you can use the fit() function to prepare your data and later the transform() function. Or you can use the combined fit-and-transform.

The Machine Learning Process


Iterate until data is ready
Iterate to find the best model
Raw Data
Raw Data
Apply pre-processing to data
Prepared Data
Apply learning algorithm to data
Candidate Model
Deploy chosen model
Chosen Model
Data Preprocessing Modules
Machine Learning Algorithms
Applications

From "Introduction to Microsoft Azure" by David Chappell



In this exercise we will use one of the traditional Machine Learning dataset, the Pima Indians diabetes dataset.

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Content The datasets consists of several medical predictor variables and one target variable, **Outcome**. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

> - Pregnancies
> - Glucose
> - BloodPressure
> - SkinThickness
> - Insulin
> - BMI
> - DiabetesPedigreeFunction
> - Age
> - Outcome

In [89]:

```
# Load the Pima indians dataset and separate input and output components

from numpy import set_printoptions
set_printoptions(precision=3)

filename="pima-indians-diabetes.data.csv"
names=["pregnancies", "glucose", "pressure", "skin", "insulin", "bmi", "pedi", "age", "outc
p_indians=pd.read_csv(filename, names=names)
p_indians.head()

# First we separate into input and output components
array = p_indians.values
X = array[:,0:8]
Y = array[:,8]
X
pd.DataFrame(X).head()
```

Out[89]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Out[89]:

```
array([[  6.   , 148.   ,  72.   , ...,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , ...,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , ...,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , ...,  30.4  ,   0.315,  23.   ]])
```

Out[89]:

	0	1	2	3	4	5	6	7
0	6.0	148.0	72.0	35.0	0.0	33.6	0.627	50.0
1	1.0	85.0	66.0	29.0	0.0	26.6	0.351	31.0
2	8.0	183.0	64.0	0.0	0.0	23.3	0.672	32.0
3	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0
4	0.0	137.0	40.0	35.0	168.0	43.1	2.288	33.0

Rescale Data

It is very common that the attributes have very different scales. Therefore, many machine learning algorithms benefit from rescalling the attributes to all have the same scale. Normally between 0 and 1. This process is commonly called normalization.

This is important with optimization algorithms that use gradient descent. Also with algorithms, like regressions, that weight inputs like regression or neural networks. It is also needed when the the algorithms use distances such as the case of k-means or k-nn(K-Nearest Neighbors).

For rescaling your data, you use the **MinMaxScaler** class.

In [90]:

```
# Rescale data between 0 and 1

p_indians.head()

from sklearn.preprocessing import MinMaxScaler

# Scale between 0 and 1
scaler=MinMaxScaler(feature_range=(0,1))
rescaledX=scaler.fit_transform(X)

rescaledX
```

Out[90]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Out[90]:

```
array([[0.353, 0.744, 0.59 , ..., 0.501, 0.234, 0.483],
       [0.059, 0.427, 0.541, ..., 0.396, 0.117, 0.167],
       [0.471, 0.92 , 0.525, ..., 0.347, 0.254, 0.183],
       ...,
       [0.294, 0.608, 0.59 , ..., 0.39 , 0.071, 0.15 ],
       [0.059, 0.633, 0.492, ..., 0.449, 0.116, 0.433],
       [0.059, 0.467, 0.574, ..., 0.453, 0.101, 0.033]])
```

Standarize Data

Standarization is a technique that assumes a Gaussians distribution but different means and standard deviations. Transforming them to a Gaussian of mean 0 and standard deviation of 1.

It is most suitable for techniques that assume a Gaussian distribution in the input variables and work better with rescaled data, such as linear regression, logistic regression or LDA (linear discriminant analysis).

For standarizing you use the **StandardScaler** class.

In [91]:

```
# Standarize data (0 mean, 1 stdev)

from sklearn.preprocessing import StandardScaler

p_indians.head()

scaler=StandardScaler().fit(X)
rescaledX=scaler.transform(X)

rescaledX
```

Out[91]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Out[91]:

```
array([[ 0.64 ,  0.848,  0.15 , ...,  0.204,  0.468,  1.426],
       [-0.845, -1.123, -0.161, ..., -0.684, -0.365, -0.191],
       [ 1.234,  1.944, -0.264, ..., -1.103,  0.604, -0.106],
       ...,
       [ 0.343,  0.003,  0.15 , ..., -0.735, -0.685, -0.276],
       [-0.845,  0.16 , -0.471, ..., -0.24 , -0.371,  1.171],
       [-0.845, -0.873,  0.046, ..., -0.202, -0.474, -0.871]])
```

Mission 1

a) Sometimes we only want to standarize some attributes and not all. For the shake of example, let's say that we only want standarize glucose.

b) Create a new X with all the attributes of the all X but the standarized glucose.

c) Do the same Scaling (from 0 to 1) instead of Standarizing.

hint: we are dealing with numpy arrays and not DataFrames here, you should use np.concatenate()

In [92]:

```
# a) Sometimes we only want to standarize some attributes and not all.
# For the shake of example, let's say that we only want standarize glucose.

gluc = p_indians[["glucose"]]
stand_gluc = StandardScaler().fit(gluc).transform(gluc)
stand_gluc
```

```
       [ 1.598e-01],
       [-6.852e-01],
       [ 2.351e+00],
       [-5.929e-02],
       [ 6.918e-01],
       [ 1.285e-01],
       [ 8.170e-01],
       [-7.478e-01],
       [ 7.544e-01],
       [-1.219e-01],
       [-3.723e-01],
       [ 1.161e+00],
       [-1.030e+00],
       [-9.043e-01],
       [ 3.460e-02],
       [-5.600e-01],
       [ 5.354e-01],
       [-5.913e-01],
       [-9.669e-01],
       [-3.097e-01],
```

In [93]:

```
# b) Create a new X with all the attributes of the all X but the standarized glucose.

array = p_indians.values
bef_gluc = array[:,0:1]
aft_gluc = array[:,2:8]
X = np.concatenate((bef_gluc, stand_gluc, aft_gluc), axis = 1)
X
```

Out[93]:

```
array([[ 6.000e+00,  8.483e-01,  7.200e+01, ...,  3.360e+01,  6.270e-01,
         5.000e+01],
       [ 1.000e+00, -1.123e+00,  6.600e+01, ...,  2.660e+01,  3.510e-01,
         3.100e+01],
       [ 8.000e+00,  1.944e+00,  6.400e+01, ...,  2.330e+01,  6.720e-01,
         3.200e+01],
       ...,
       [ 5.000e+00,  3.301e-03,  7.200e+01, ...,  2.620e+01,  2.450e-01,
         3.000e+01],
       [ 1.000e+00,  1.598e-01,  6.000e+01, ...,  3.010e+01,  3.490e-01,
         4.700e+01],
       [ 1.000e+00, -8.730e-01,  7.000e+01, ...,  3.040e+01,  3.150e-01,
         2.300e+01]])
```

In [94]:

```
# c) Do the same Scaling (from 0 to 1) instead of Standarizing.

res_gluc = MinMaxScaler(feature_range=(0,1)).fit_transform(gluc)
res_gluc
```

Out[94]:

```
array([[0.744],
       [0.427],
       [0.92 ],
       [0.447],
       [0.688],
       [0.583],
       [0.392],
       [0.578],
       [0.99 ],
       [0.628],
       [0.553],
       [0.844],
       [0.698],
       [0.95 ],
       [0.834],
       [0.503],
       [0.593],
       [0.538],
```

Normalize Data

Normalization works with observations (rows) instead of attributes (columns).

The idea here is to have a length 1 for each observation (a vector of length 1 in linear algebra).

It is useful in algorithms that weigth input values as a whole, such is the case of Neural Networks and also distance algorithms such as K-nn (Nearest Neighbors)

For normalization you use the **Normalizer** class.

In [95]:

```
from sklearn.preprocessing import Normalizer

p_indians.head()

scaler = Normalizer().fit(X)
normalizedX = scaler.transform(X)

normalizedX
```

Out[95]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Out[95]:

```
array([[ 5.978e-02,  8.452e-03,  7.173e-01, ...,  3.347e-01,  6.247e-03,
         4.981e-01],
       [ 1.207e-02, -1.356e-02,  7.964e-01, ...,  3.210e-01,  4.235e-03,
         3.741e-01],
       [ 1.057e-01,  2.568e-02,  8.454e-01, ...,  3.078e-01,  8.877e-03,
         4.227e-01],
       ...,
       [ 3.547e-02,  2.342e-05,  5.108e-01, ...,  1.859e-01,  1.738e-03,
         2.128e-01],
       [ 1.220e-02,  1.950e-03,  7.321e-01, ...,  3.673e-01,  4.259e-03,
         5.735e-01],
       [ 1.169e-02, -1.021e-02,  8.184e-01, ...,  3.554e-01,  3.683e-03,
         2.689e-01]])
```

In [ ]:

Binarize Data

Binarize consist in transforming data using a binary threshold; all values above are marked as 1 and all values below as zero.

Sometimes you want to transform probabilities into crisp values. Many times it is used in feature engineering when you add a new feature.

For binarization you use the **Binarizer** class.

In [ ]:

```
```

In [96]:

```
from sklearn.preprocessing import Binarizer

p_indians.head()

binarizer=Binarizer(threshold=0.0).fit(X)
binaryX=binarizer.transform(X)

binaryX[:10,0:8]
```

Out[96]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Out[96]:

```
array([[1., 1., 1., 1., 0., 1., 1., 1.],
       [1., 0., 1., 1., 0., 1., 1., 1.],
       [1., 1., 1., 0., 0., 1., 1., 1.],
       [1., 0., 1., 1., 1., 1., 1., 1.],
       [0., 1., 1., 1., 1., 1., 1., 1.],
       [1., 0., 1., 0., 0., 1., 1., 1.],
       [1., 0., 1., 1., 1., 1., 1., 1.],
       [1., 0., 0., 0., 0., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 0., 0., 0., 1., 1.]])
```

In [ ]:

```
```

In [ ]:

```
```

Three useful functions: map, zip and filter

There are three very useful functions many times used in preprocessing for econdinf, filter or interation.

They are: map, zip and filter.

map

Let's you apply a function to a sequence of elements like a list or a dictionnary.

Very useful for encoding. Please remember that it results an iterator that must be converted to a list to be used.

In [97]:

```
#map

ln=["1","2","3","4","5"]
dn={"1":"10","2":"20","3":"30","4":"40","5":"50"}

list(map(lambda x: int(x),ln))

list(map(lambda x:int(x),dn))
```

Out[97]:

```
[1, 2, 3, 4, 5]
```

Out[97]:

```
[1, 2, 3, 4, 5]
```

In [ ]:

```

```

Mission 2

a) We want to highlight everybody with a glucose level over 140 setting it to 1.

b) We'll do the same with blood pressure over 80.

c) Finally we will create a new attribute that we will name warning, when both glucose and blood pressure is set to 1, being 0 otherwise.

d) We need a new X (let's call it X_new) with these attributes instead of the original ones.

In [98]:

```
# a) We want to highlight everybody with a glucose level over 140 setting it to 1.

glucose_array = p_indians[["glucose"]]
p_indians["gluc_over_140"] = Binarizer(threshold = 140.0).fit(glucose_array).transform(gluc

#code = {1:"Yes", 0:"No"}

p_indians["gluc_over_140"] = p_indians["gluc_over_140"]#.map(code)
p_indians
```

Out[98]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome	gluc_over_140
0	6	148	72	35	0	33.6	0.627	50	1	1
1	1	85	66	29	0	26.6	0.351	31	0	0
2	8	183	64	0	0	23.3	0.672	32	1	1
3	1	89	66	23	94	28.1	0.167	21	0	0
4	0	137	40	35	168	43.1	2.288	33	1	0
5	5	116	74	0	0	25.6	0.201	30	0	0
6	3	78	50	32	88	31.0	0.248	26	1	0
7	10	115	0	0	0	35.3	0.134	29	0	0
8	2	197	70	45	543	30.5	0.158	53	1	1
9	8	125	96	0	0	0.0	0.232	54	1	0

In [99]:

```
# b) We'll do the same with blood pressure over 80.

pressure_array = p_indians[["pressure"]]
p_indians["pressure_over_80"] = Binarizer(threshold = 80.0).fit(pressure_array).transform(p

#code_pressure = {1:"Yes", 0:"No"}

p_indians["pressure_over_80"] = p_indians["pressure_over_80"]#.map(code_pressure)
p_indians
```

Out[99]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome	gluc_over_140	pressure_ove
0	6	148	72	35	0	33.6	0.627	50	1	1	
1	1	85	66	29	0	26.6	0.351	31	0	0	
2	8	183	64	0	0	23.3	0.672	32	1	1	
3	1	89	66	23	94	28.1	0.167	21	0	0	
4	0	137	40	35	168	43.1	2.288	33	1	0	
5	5	116	74	0	0	25.6	0.201	30	0	0	
6	3	78	50	32	88	31.0	0.248	26	1	0	
7	10	115	0	0	0	35.3	0.134	29	0	0	
8	2	197	70	45	543	30.5	0.158	53	1	1	

In [100]:

```
# c) Finally we will create a new attribute that we will name warning, when both
# glucose and blood pressure is set to 1, being 0 otherwise.

name_warning_list = []
n = len(p_indians)

for x in range(n):
    if (p_indians["gluc_over_140"].iloc[x] == 1 and p_indians["pressure_over_80"].iloc[x] =
        name_warning_list.append(1)
    else:
        name_warning_list.append(0)

name_warning_array = np.asarray(name_warning_list)

p_indians["name_warning"] = name_warning_array
p_indians
```

Out[100]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome	gluc_over_140	pressure_ove
0	6	148	72	35	0	33.6	0.627	50	1	1	
1	1	85	66	29	0	26.6	0.351	31	0	0	
2	8	183	64	0	0	23.3	0.672	32	1	1	
3	1	89	66	23	94	28.1	0.167	21	0	0	
4	0	137	40	35	168	43.1	2.288	33	1	0	
5	5	116	74	0	0	25.6	0.201	30	0	0	
6	3	78	50	32	88	31.0	0.248	26	1	0	
7	10	115	0	0	0	35.3	0.134	29	0	0	
8	2	197	70	45	543	30.5	0.158	53	1	1	

In [101]:

```
# d) We need a new X (let's call it X_new) with these attributes instead of the original on

array = p_indians.values
X_new = array[:,0:12]
X_new
```

Out[101]:

```
array([[  6., 148.,  72., ...,   1.,   0.,   0.],
       [  1.,  85.,  66., ...,   0.,   0.,   0.],
       [  8., 183.,  64., ...,   1.,   0.,   0.],
       ...,
       [  5., 121.,  72., ...,   0.,   0.,   0.],
       [  1., 126.,  60., ...,   0.,   0.,   0.],
       [  1.,  93.,  70., ...,   0.,   0.,   0.]])
```

zip

It enables you to iterate over two or more lists at the same time.

In [102]:

```
#zip

first = [1, 3, 8, 4, 9]
second = [2, 2, 7, 5, 8]

# Iterate over two or more list at the same time
for x, y in zip(first, second):
    print(x + y)
```

```
3
5
15
9
17
```

filter

Similar to map() but in this case will return True or False.

In [103]:

```
#filter

ln=["1","2","3","4","5"]
dn={"1":"10","2":"20","3":"30","4":"40","5":"50"}

list(filter(lambda x:int(x)>1,ln))

list(filter(lambda x:int(x)>1,dn))
list(filter(lambda x:int(x)>10,dn.values()))
```

Out[103]:

```
['2', '3', '4', '5']
```

Out[103]:

```
['2', '3', '4', '5']
```

Out[103]:

```
['20', '30', '40', '50']
```

In [104]:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Will return true if input number is even
def even(x):
    return x % 2 == 0

even_n = filter(even, numbers)
even_n1 = filter(lambda x: x % 2==0, numbers)

list(even_n)
list(even_n1)
```

Out[104]:

```
[2, 4, 6, 8, 10]
```

Out[104]:

```
[2, 4, 6, 8, 10]
```

In [ ]:

```

```

In [ ]:

```

```