Preprocessing

- **Data Cleaning**
  - Missing Values
  - Noisy Data
- **Data Integration**
  - Data Merging
  - Redundancies
  - Value conflicts
- **Data Transformation**
  - Normalization
  - Attribute Construction
- **Data Reduction**
  - Data Cube Aggregation
  - Attribute Subset Selection
  - Principal Components Analysis
  - Multidimensional Scaling
  - Locally Linear Embedding

Data Cleaning

Problem	Solution
Missing Values	Approximate the missing data
Noisy Data	Use smoothing methods to remove these errors
Data Inconsistencies	

Data Cleaning

Missing Values => Approximate the missing data

Why Many data mining algorithms require a complete set of data to run.

How

1. **Ignore missing values** => large amounts of valuable data might be thrown away
2. **Statistical Measures**: Mean Values, Regression, Decision Tree based on other attributes

- Mean Values

```python
import numpy as np
revenue=np.array([7,2,6,4,14,np.nan,16,12,14,20,15,7])
profit=np.array([0.15,0.1,0.13,0.15,0.25,0.27,0.24,0.2,0.27,0.44,0.34,0.17])
# fill in mean of other numbers
revenue[np.isnan(revenue)]=int(revenue[~np.isnan(revenue)].mean())
```

- Regression

```python
from sklearn.linear_model import LinearRegression
revenue1=revenue[~np.isnan(revenue)]
profit1=np.delete(profit, np.argwhere(np.isnan(revenue))) #delete profit of June

lm_all=LinearRegression()
# reshape array into (11,1)
lm_all.fit(np.reshape(revenue1, (len(revenue1), 1)),np.reshape(profit1, (len(profit1),1)))
# x = (y-b)/a
revenue_jun=float((profit[np.argwhere(np.isnan(revenue))]-float(lm_all.intercept_))/float(lm_all.coef_))
# revenue_jun = 13.612104539202202
```

	revenue (old) 0	revenue (new) 0
0	7	7
1	2	2
2	6	6
3	4	4
4	14	14
5	nan	10
6	16	16
7	12	12
8	14	14
9	20	20
10	15	15
11	7	7

Data Cleaning

Noisy Data => Smoothing methods

Why Real world data sets often have random error within their values.

How

1. **Binning**: A local smoothing method which means it consults only the immediate neighborhood of each data point for the smoothing operation.
2. **Regression**: A global method of smoothing where we attempt to find a function which best fits the data set as a whole.

1. Binning:

- Smoothing Method:
  1. Bin Means
  2. Bin Medians
  3. Bin boundaries

Table 4.1: Example of Binning

	partitioning	smoothing by means	smoothing by bin boundaries
bin 1	2, 4, 5, 6, 9, 10	6, 6, 6, 6, 6, 6	2, 2, 2, 2, 10, 10
bin 2	12, 16, 17, 19	16, 16, 16, 16	12, 19, 19, 19
bin 3	23, 26, 27, 28	26, 26, 26, 26	23, 28, 28, 28
bin 4	31, 33, 35	33, 33, 33	31, 31, 35

Data Cleaning

- Noisy Data => Smoothing methods

```
import numpy as np
a=np.array([2,4,5,6,9,10,12,16,17,19,23,26,27,28,31,33,35])
a_split=np.split(a, [6,10,14]) #split a into 4 groups
```

a_split

0	int32	(6,)	[2 4 5 6 9 10]
1	int32	(4,)	[12 16 17 19]
2	int32	(4,)	[23 26 27 28]
3	int32	(3,)	[31 33 35]

1. Bin Means

```
a_median=a_split.copy()
for i in range(len(a_median)): # replaced by median
    median=int(np.median(a_median[i]))
    a_median[i]=np.full((1,len(a_median[i])),median)
ans=np.concatenate((a_median[0], a_median[1], a_median[2], a_median[3]), axis=None).tolist()
#[5, 5, 5, 5, 5, 5, 16, 16, 16, 16, 26, 26, 26, 26, 33, 33, 33]
```

2. Bin Medians

```
a_median=a_split.copy()
for i in range(len(a_median)): # replaced by median
    median=int(np.median(a_median[i]))
    a_median[i]=np.full((1,len(a_median[i])),median)
ans=np.concatenate((a_median[0], a_median[1], a_median[2], a_median[3]), axis=None).tolist()
#[5, 5, 5, 5, 5, 5, 16, 16, 16, 16, 26, 26, 26, 26, 33, 33, 33]
```

Data Cleaning

Noisy Data => Smoothing methods

```
import numpy as np
a=np.array([2,4,5,6,9,10,12,16,17,19,23,26,27,28,31,33,35])
a_split=np.split(a, [6,10,14]) #split a into 4 groups
```

a_split

0	int32	(6,)	[2 4 5 6 9 10]
1	int32	(4,)	[12 16 17 19]
2	int32	(4,)	[23 26 27 28]
3	int32	(3,)	[31 33 35]

3. Bin boundaries

```
a_boundry=a_split.copy()

for i in range(len(a_boundry)): # replaced by boundry
    for j in range(1, len(a_boundry[i])-1): # values without the first one and final one
        # if the value is closer to first number
        if (a_boundry[i][j]-a_boundry[i][0]) <=  (a_boundry[i][len(a_boundry[i])-1]-a_boundry[i][j]):
            a_boundry[i]=np.where(a_boundry[i]==a_boundry[i][j], a_boundry[i][0], a_boundry[i])
        else: # if the value is closer to final number
            a_boundry[i]=np.where(a_boundry[i]==a_boundry[i][j], a_boundry[i][len(a_boundry[i])-1], a_boundry[i])

ans=np.concatenate((a_boundry[0], a_boundry[1], a_boundry[2], a_boundry[3]), axis=None).tolist()
#[2, 2, 2, 2, 10, 10, 12, 19, 19, 19, 23, 28, 28, 28, 31, 31, 35]
```

Data Integration

whenever the target data is extracted from multiple data stores…

Problem	Solution
The data must be merged	Metadata
Redundancies must be removed	Correlation Analysis
Value conflicts must be resolved	Data Transformation

Data Integration

1. The data must be merged. => Metadata
2. Redundancies must be removed. => Correlation Analysis
3. Value conflicts must be resolved. => Data Transformation

```
import numpy as np
import pandas as pd
df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'],'value': [1, 2, 3, 5]})
df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'],'value': [5, 6, 7, 8]})
df_com=df1.merge(df2, left_on='lkey', right_on='rkey')
```

df1

lkey	value
foo	1
bar	2
baz	3
foo	5

+

df2

rkey	value
foo	5
bar	6
baz	7
foo	8

=

df_com

lkey	value_x	rkey	value_y
foo	1	foo	5
foo	1	foo	8
foo	5	foo	5
foo	5	foo	8
bar	2	bar	6
baz	3	baz	7

Data Integration

1. The data must be merged. => Metadata
2. Redundancies must be removed. => Correlation Analysis
3. Value conflicts must be resolved. => Data Transformation

```
a=pd.DataFrame({'feature1':[1,2,3,4,5], 'feature2':[2,4,6,8,11]})
corr=np.corrcoef(a['feature1'],a['feature2'])[0,1] #0.995893206467037=>high
del a['feature1'] # remove 'feature1' or 'feature2'
```

a

feature1	feature2
1	2
2	4
3	6
4	8
5	11

Remove “feature1”

a

feature2
2
4
6
8
11

Data Integration

1. The data must be merged. => Metadata
2. Redundancies must be removed. => Correlation Analysis
3. Value conflicts must be resolved. => Data Transformation

```
s1=pd.DataFrame({'ID':[111,555,1000],'price':[31.12,155.58,311.15]}) # TWD
s2=pd.DataFrame({'ID':[111,555,1000],'price':[1,5,10]}) # USD

s2['s2_TWD']=s2['price']*(155.58/5)
```

s1 (TWD)

ID	price
111	31.12
555	155.58
1000	311.15

s2 (USD)

ID	price
111	1
555	5
1000	10

s2 (USD=>TWD)

ID	price	s2_TWD
111	1	31.116
555	5	155.58
1000	10	311.16

Data Transformation

Why For data mining algorithms to work efficiently the input data often has to be in a certain format.

How 1. **Normalization**: Rescales data values to fit into a specified range such as 0.0 to 1.0.

Method:

- **Min-max normalization**: It will encounter an error if future input falls outside the original data range of the attribute.
- **Z-score normalization**: We do not need to provide the range of the attribute and therefore ensure that all future data will be accepted.

```
df=pd.DataFrame({'sales_num':[10,2,6], 'price':[1000,350,500]})
# Min-max normalization
df['price']=(df['price']-min(df['price']))/(max(df['price'])-min(df['price']))*(1-0)+0 # targeted range=>[0,1]
# Z-score normalization
df['price']=(df['price']-df['price'].mean())/df['price'].std()
```

df

sales_num	price
10	1000
2	350
6	500

df (Min-ma)

sales_num	price
10	1
2	0
6	0.230769

df(Z-score)

sales_num	price
10	1.12631
2	-0.783523
6	-0.342791

$$x'_{n,m} = \frac{x_{n,m} - min_m}{max_m - min_m} \cdot (max'_m - min'_m) + min'_m$$

$$x'_{n,m} = \frac{x_{n,m} - \mu_m}{\sigma_m}$$

Data Transformation

Why For data mining algorithms to work efficiently the input data often has to be in a certain format.

How

1. **Normalization**: Rescales data values to fit into a specified range such as 0.0 to 1.0.
2. **Attribute Construction:** New attributes are constructed from given data.

```
df=pd.DataFrame({'ID':[11,22,33], 'promotion1':[0,0,1], 'promotion2':[1,1,1]})
df['promotion_all']=df['promotion1']+df['promotion2']
```

df

ID	promotion1	promotion2
11	0	1
22	0	1
33	1	1

→

df (with new attribute)

ID	promotion1	promotion2	promotion_all
11	0	1	1
22	0	1	1
33	1	1	2

Data Reduction

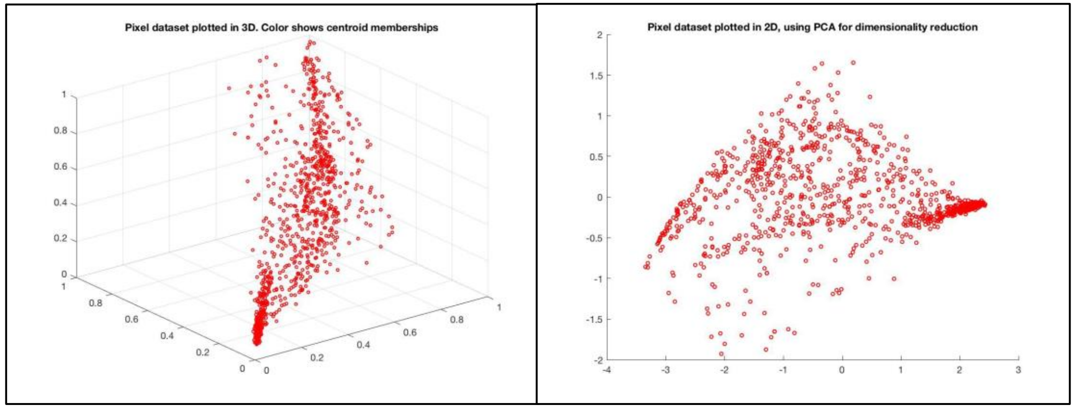
Why Data mining on huge databases is likely to take a very long time, making the analysis practically impossible.

How

1. **Data**: Data Cube Aggregation
2. **Feature**: Attribute Subset Selection
3. **Dimension**:
    - Principal Components Analysis (Linear)
    - Multidimensional Scaling (Manifold)
    - Locally Linear Embedding (Manifold)

Linear

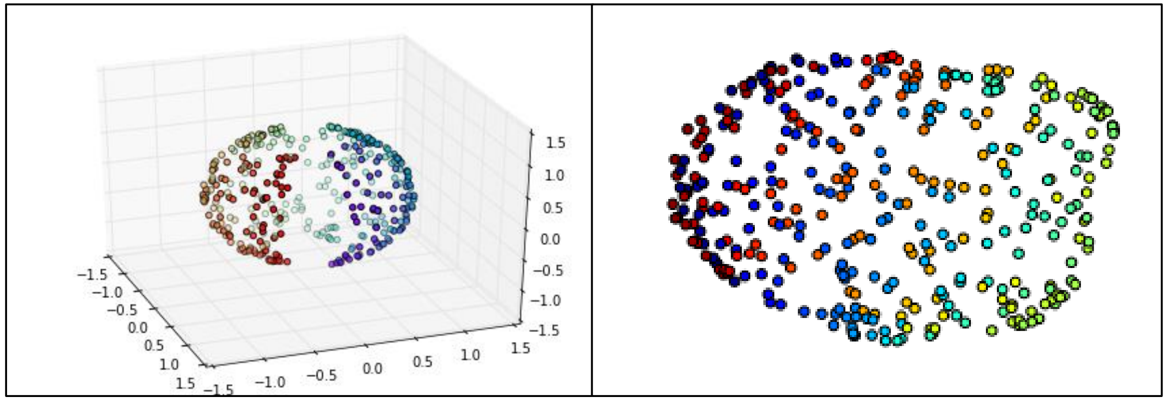
3-Dimensional 2-Dimensional



Source: https://reurl.cc/X9Dd3

Manifold(流形)

3-Dimensional 2-Dimensional

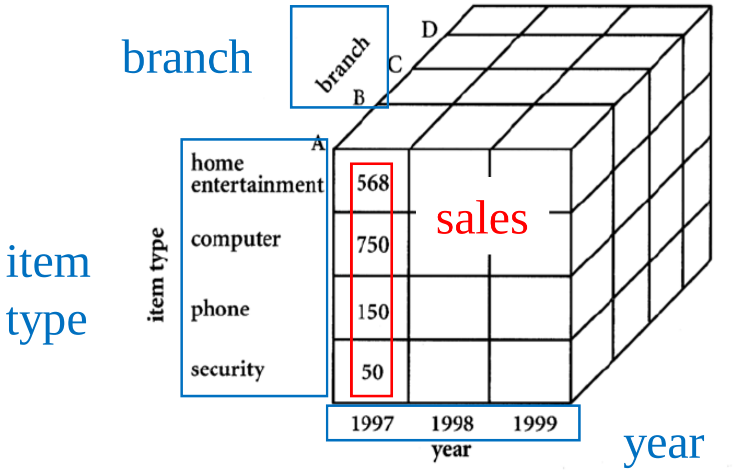


Source: https://reurl.cc/5Ek6M

Data Reduction

- **Data**: Data Cube Aggregation

Goal	Data cubes provide efficient organization of summarized information which is the foundation of on-line analytical processing as used in data exploration.
How	1. Choose the attribute to be displayed in value. 2. Choose a number of attributes according to which the data should be aggregated. 3. Choose the level of abstraction for the aggregation and determine data classes accordingly. 4. For all possible combination of classes of the dimensions calculate the aggregated value of the displayed attribute.




```
import pandas as pd
df=pd.DataFrame({'product':[1,1,1,2,2,2], 'year':[1997,1998,1999,1997,1998,1999],
                 'sales':[10,20,30,100,200,300]})
year_sales=df.groupby('year')['sales'].sum()
```

product	year	sales
1	1997	10
1	1998	20
1	1999	30
2	1997	100
2	1998	200
2	1999	300

Index	sales
1997	110
1998	220
1999	330

Data Reduction

- **Feature**: Attribute Subset Selection

Goal	Remove as many of the original attributes as possible, while maintaining its integrity with respect to a given class or concept.
How	1. **Stepwise forward selection**: Add the best attribute (decided by a given threshold) 2. **Stepwise backward elimination**: Remove the worst attribute(decided by a given threshold) 3. **Combination of forward selection and backward elimination** 4. **Decision tree induction**

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=4,
                           n_informative=2, n_redundant=0,
                          random_state=0, shuffle=False)
clf = RandomForestClassifier(n_estimators=100, max_depth=2,
                              random_state=0)
clf.fit(X, y)
fea_importance=clf.feature_importances_
```

Source: https://reurl.cc/jO7Wq

fea_importance

	0
0	0.14206
1	0.76664
2	0.0282433
3	0.0630566

Data Reduction

- **Dimension**: Principal Components Analysis

Goal	Maximize the variability along the new coordinate system, thus conserving as much of the variance of the original data with as few dimensions as possible.
How	1. Reduce the dimensionality of the data set. 2. The data is projected onto a new set of variables, called principal components. These principal components are a linear combination of the original attributes.

```
import numpy as np
from sklearn.decomposition import PCA
data=np.array([[ 1.  ,  1.  ],
               [ 0.9 ,  0.95],
               [ 1.01,  1.03],
               [ 2.  ,  2.  ],
               [ 2.03,  2.06],
               [ 1.98,  1.89]])
data.shape #(6, 2)

pca=PCA(n_components=1)
newData_shape=pca.fit_transform(data).shape #(6, 1)
```

data

	0	1
0	1	1
1	0.9	0.95
2	1.01	1.03
3	2	2
4	2.03	2.06
5	1.98	1.89

→

newData

	0
0	0.689271
1	0.796015
2	0.661272
3	-0.724666
4	-0.787874
5	-0.634018

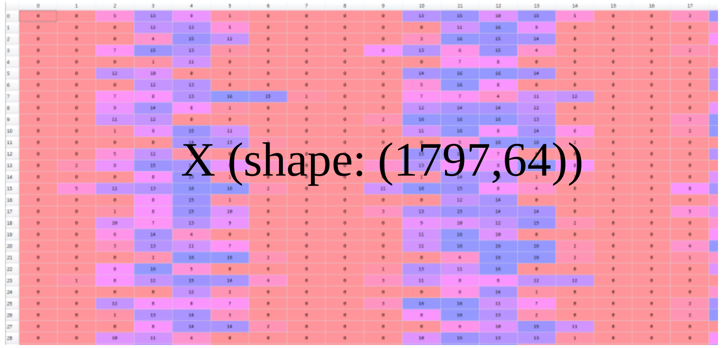
Data Reduction

- **Dimension**: Multidimensional Scaling

Goal	Deal with data aligned as a multidimensional, which PCA might not to be able to detect the underlying structure of the data at all.
How	Project the original data onto a plane, preserving the distances between all data points as good as possible.

```python
from sklearn.datasets import load_digits
from sklearn.manifold import MDS
X, _ = load_digits(return_X_y=True)
X.shape #(1797, 64)
mds = MDS(n_components=2)
X_transformed = mds.fit_transform(X[:100]) #(100, 2)
```

Source: https://reurl.cc/Azq1j



X (shape: (1797,64))

	0	1
0	27.2541	12.06
1	-20.0699	5.82912
2	-28.1604	17.143
3	23.7477	-15.1295
4	-18.3296	35.912
5	29.616	-12.9841
6	4.33321	31.9312
7	-27.0269	-35.0323
8	15.5567	1.53302
9	26.1865	-4.8136
10	13.2084	20.1081
11	-35.902	-1.9362
12	33.6404	3.42082
13	16.3893	-25.8497
14	-18.4514	29.8945
15	-7.98662	-40.0141
16	-6.91638	40.5982
17	-16.8601	-13.0509
18	-5.34684	-13.2644
19	41.3093	-4.59679
20	19.9058	20.3173
21	-35.8104	-6.17561
22	27.9681	0.804983
23	7.6896	-35.2765
24	-22.9169	28.759
25	6.79694	4.49465
26	10.2122	37.9382
27	-33.5738	-19.8815
28	4.50246	-4.33054
29	36.3612	-9.87833

X_transformed
(shape: (100,2))

Data Reduction

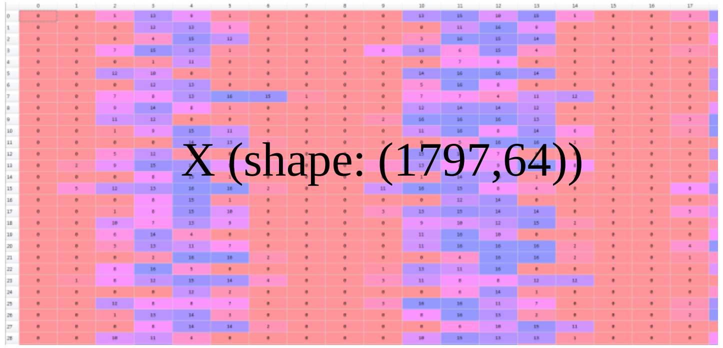
- **Dimension**: Locally Linear Embedding

Why Experience shows that even the most complex data usually follows some low-dimensional, nonlinear manifold. For classification and comparison tasks it is sufficient to regard only this manifold instead of the whole data set.

How Determine such a manifold of the data by analyzing its local correlations.

```
from sklearn.datasets import load_digits
from sklearn.manifold import LocallyLinearEmbedding
X, _ = load_digits(return_X_y=True)
X.shape #(1797, 64)
lle = LocallyLinearEmbedding(n_components=2)
X_transformed= lle.fit_transform(X[:100]) #(100, 2)
```

Source: https://reurl.cc/ylZyE



X (shape: (1797,64))

	0	1
0	27.2541	12.06
1	-20.0699	5.82912
2	-28.1604	17.143
3	23.7477	-15.1295
4	-18.3296	35.912
5	29.616	-12.9841
6	4.33321	31.9312
7	-27.0269	-35.0323
8	15.5567	1.53302
9	26.1865	-4.8136
10	13.2084	20.1081
11	-35.902	-1.9362
12	33.6404	3.42082
13	16.3893	-25.8497
14	-18.4514	29.8945
15	-7.98662	-40.0141
16	-6.91638	40.5982
17	-16.8601	-13.0509
18	-5.34684	-13.2644
19	41.3093	-4.59679
20	19.9058	20.3173
21	-35.8104	-6.17561
22	27.9681	0.804983
23	7.6896	-35.2765
24	-22.9169	28.759
25	6.79694	4.49465
26	10.2122	37.9382
27	-33.5738	-19.8815
28	4.50246	-4.33054
29	36.3612	-9.87833

X_transformed
(shape: (100,2))